

Chapter 1: Introduction to Python

Contents

- 1 Getting started with Python
 - 1.1 Version used
 - 1.2 Platforms for Python programming
- 2 Starting Python with Hello World
 - 2.1 IDLE
 - 2.2 Python file
 - 2.3 Jupyter Notebook
- 3 Python Syntax
 - 3.1 Indentation and white space
 - 3.2 Naming conventions
 - 3.3 Write comments

Appendix A – Tips for using IDLE

Appendix B – Jupyter keyboard shortcuts

Appendix C – The Importance of Good Comments: A Tale of the Baker's Apprentice

Syllabus Learning Outcomes

- 2.1 Coding Standards
 - Use common coding standards for programming style (which is dependent on programming language used).
 - 2.1.1 Use indentation and white space.
 - 2.1.2 Use naming conventions (e.g. meaningful identifier names).
 - 2.1.3 Write comments (name of programmer, date written, program description and version bookkeeping/control).

References

Computational Fairy Tales: The Importance of Good Comments: A Tale of the Baker's Apprentice (computationaltales.blogspot.com)

https://www.w3schools.com/python/python_syntax.asp

1 Getting started with Python

1.1 Versions used

The latest version of Python for Windows is 3.10.1. However, we are using version 3.6.4 for the A-Level H2 Computing Syllabus 9569.

1.2 Platforms for Python programming

There are many ways to do programming in Python.

- 1) One way is to use Python's **Integrated Development and Learning Environment** a.k.a. IDLE. It is installed with Python installation.
- 2) Creating a Python file and run it using IDLE.
- 3) Jupyter notebook is another platform for Python programming. The version we will be using is 5.5.0.
- 4) Google Colaboratory. This is a convenient way to do Python programming, as there is no installation of software required. It is a free application available in Google drive.
- 5) CodeCollab is an online real-time collaborative code editor and compiler. Its web-based application allows users to collaborate in real-time over the internet. It is a powerful editor that supports many other languages on top of Python.
<https://codecollab.io/>

2 Starting Python with Hello World

We will illustrate how to write Python programs using the first three ways mentioned in section 1.2. These three ways are most essential to A-Level H2 computing.

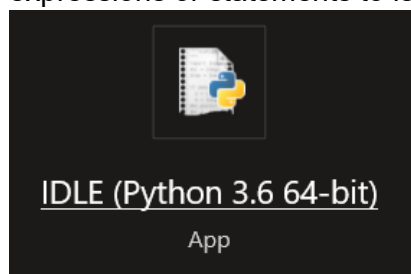
Let's start by printing "Hello World!"

2.1 IDLE

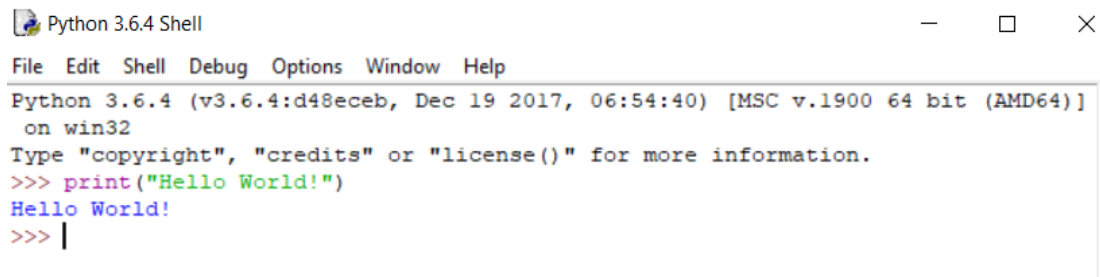
The code is simple as such:

```
print("Hello World!")
```

1. Launch the IDLE app to open a Python shell. Shell is good for experimenting with short expressions or statements to learn new features of the language.



2. Type `print("Hello World!")` in the shell and press Enter. The text "Hello World!" will be displayed.



```

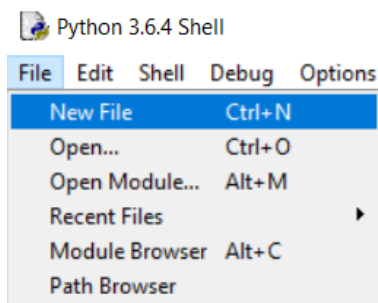
Python 3.6.4 Shell
File Edit Shell Debug Options Window Help
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:54:40) [MSC v.1900 64 bit (AMD64)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello World!")
Hello World!
>>> |

```

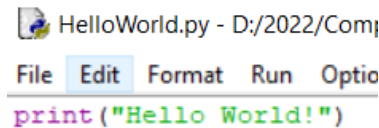
2.2 Python File

When we need to construct complex programs and test them, we can create and save files in program libraries to reuse or share with others.

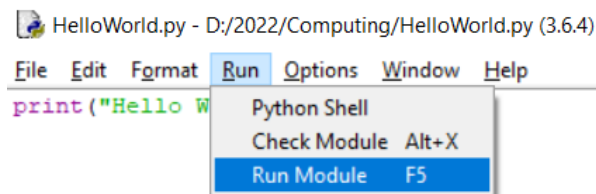
1. Launch IDLE to open a Python shell. Click on the File tab and open a new file in the shell.



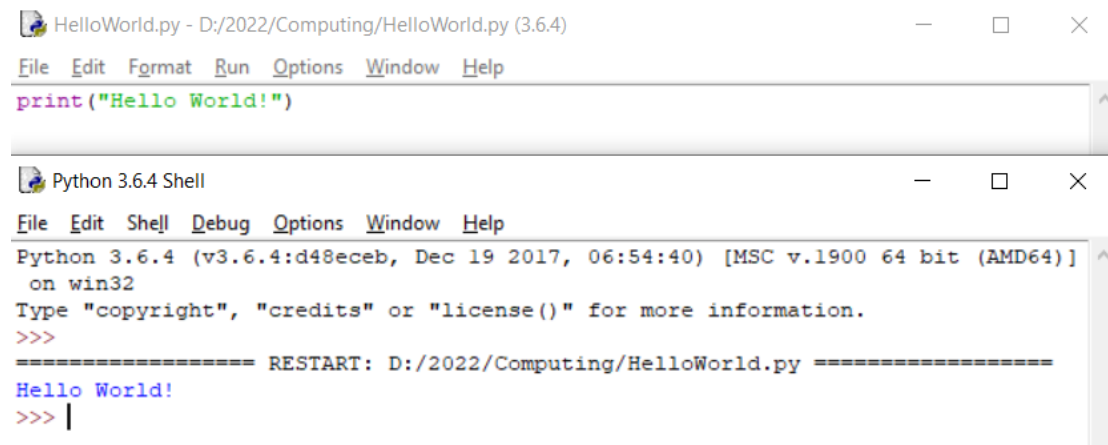
2. Enter your code into the file and save the file using .py extension.



3. Run the Python file by clicking on the Run tab followed by Run Module or using the F5 key.



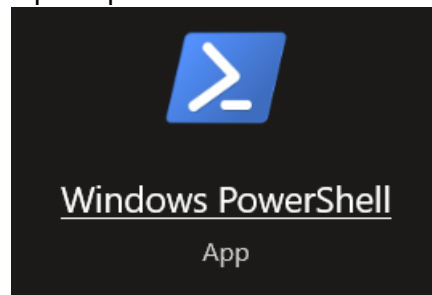
The output will be displayed in the Python shell.



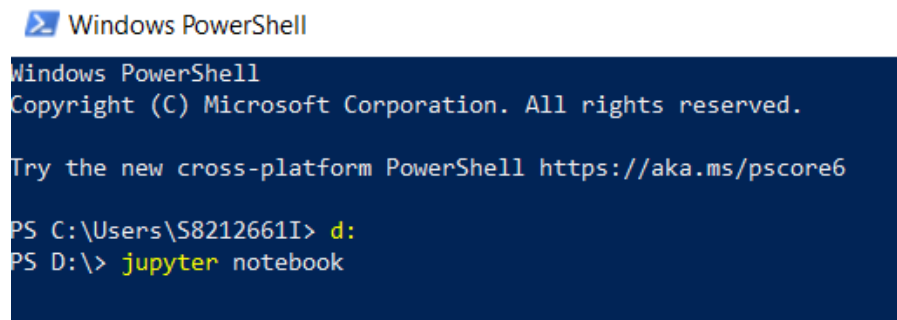
The screenshot shows two windows. The top window, titled 'HelloWorld.py - D:/2022/Computing/HelloWorld.py (3.6.4)', contains the code `print("Hello World!")`. The bottom window, titled 'Python 3.6.4 Shell', shows the output of running the script: `Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:54:40) [MSC v.1900 64 bit (AMD64)] on win32`, followed by a prompt `>>>` and the output `Hello World!`.

2.3 Using Jupyter Notebook

1. Open up Windows Powershell



2. Change to d: drive (or the drive with your working files). Type jupyter notebook and press Enter

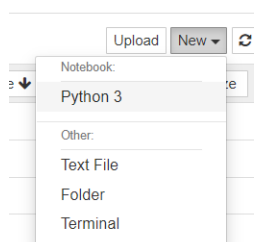


The screenshot shows a Windows PowerShell terminal window. The prompt is `PS C:\Users\S8212661I>`. The user has typed `d:` to change to the D: drive, and the prompt has changed to `PS D:\>`. The user has then typed `jupyter notebook`.

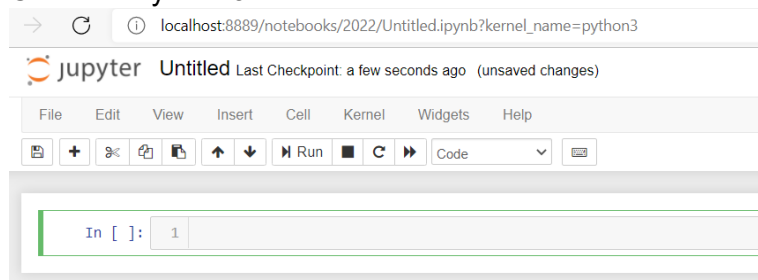
3. Jupyter notebook window will open, displaying the folders/files in the drive.



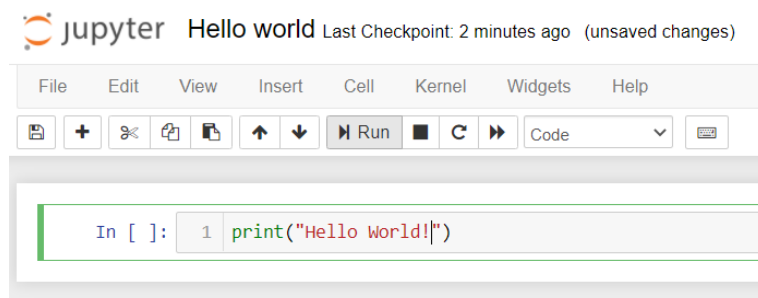
4. Go to a folder of your choice and start a new jupyter notebook file by clicking on the “new” tab on the top right corner.



5. Click on “Python 3” to start a new notebook file.



6. Change the title of the file to “Hello world”. Type `print("Hello World!")` and click “Run”.



7. “Hello World!” will be displayed.

```
In [1]: 1 print("Hello World!")
Hello World!
```

3 Python Syntax

3.1 Indentation and white space

Indentation refers to the spaces at the beginning of a code line. Tab or 4 spaces per indentation level is commonly used.

Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.

Python uses indentation to indicate a block of code.

```
1 # Example of using indentation
2
3 if 5 > 2:
4     print("Five is greater than two!")
```

Five is greater than two!

The example above shows that `print("Five is greater than two!")` is a block of code under the condition `if 5 > 2`.

Python will give an error if indentation is skipped.

```
1 # Example of using indentation
2
3 if 5 > 2:
4     print("Five is greater than two!")
```

```
File "<ipython-input-5-848779db54fd>", line 4
    print("Five is greater than two!")
    ^
```

IndentationError: expected an indented block

The number of spaces must be the same in the same block of code. Otherwise, Python will give you an error:

```

1  # Example of using indentation
2
3  if 5 > 2:
4      print("Five is greater than two!")
5      print("Two is smaller than five!")

```

File "<ipython-input-7-604e66f0a2e8>", line 5
 print("Two is smaller than five!")
 ^
IndentationError: unexpected indent

That being said, extraneous spaces, tabs and empty lines generally have no effect to the program (except in string literals and for indentation). They should be used adequately and sparingly for readability of the program code.

3.2 Naming conventions

Meaningful identifier names should be used to represent a variable, function, class, module and any other object.

Look at the code below. Are the variable names meaningful?

```

1  x = "Spiderman"
2  y = 3
3
4  print("{} is {} years old".format(x,y))

```

Spiderman is 3 years old

Can you suggest variable names that are more meaningful than x and y?

Python Keywords

Python keywords are special words which are reserved and have specific meanings. Python has a set of keywords that cannot be used as variables in programs.

```

1 from keyword import kwlist
2
3 for i in range(0, len(kwlist), 3):
4     print("{}\t{}\t{}\t".format(kwlist[i], kwlist[i+1], kwlist[i+2]))

```

False	None	True
and	as	assert
break	class	continue
def	del	elif
else	except	finally
for	from	global
if	import	in
is	lambda	nonlocal
not	or	pass
raise	return	try
while	with	yield

Guidelines for Creating Identifiers

- Identifier name can contain following characters
 - a sequence of letters either in lowercase (a to z) or uppercase (A to Z)
 - digits (0 to 9)
 - underscore (_)
- Identifier name cannot begin with digits
- Special characters are not allowed
- Identifier names are case-sensitive (age and Age are different variables)
- Avoid using Python reserved keywords as identifier names

Best Practices

- Modules should have short, all-lowercase names. Underscores can be used in the module name if it improves readability.
- Class names should normally use the CapWords convention. Eg. MyVariableName
- Function, variable, and method names should be lowercase, with words separated by underscores as necessary to improve readability. Eg. my_variable_name
- Constants are usually defined on a module level and written in all capital letters with underscores separating words. Eg. MY_VARIABLE_NAME
- Declare private identifiers by using the single-underscore _ as their first letter.
- Don't use double-underscores __ as the leading and trailing character in an identifier as Python built-in types use this notation.

Which of the following do you think will give an error?

- 1) message = 'hello'
- 2) _message = 'hello'
- 3) lmessage = 'hello'
- 4) message? = 'hello'
- 5) mes-sage = 'hello'

3.3 Write comments

Why writing comments is important? Read Appendix C.

Comments are additional text within code that provide explanation. Comments can greatly improve the readability of code by providing insight or summarization. For example, a clear comment before a large block of numeric statements might simply explain "Solve $a*x^2 + b*x + c = 0$ for x ", allowing the user to understand exactly what the following block of code is meant to do.

Comments are also used as part of documentation to help oneself or others understand the program code after a period of time.

Comments start with a #, and Python will render the rest of the line as a comment:

```
1 # This is a comment
2 print("Hello World!")
```

Hello World!

Compare the two block of codes below:

Code example 1

```
with open("COURSE_INTAKE_BY_YEAR.CSV") as f:
    reader = csv.reader(f)
    header = next(reader)
    intake_data = [row for row in reader]
```

Code example 2

```
with open("COURSE_INTAKE_BY_YEAR.CSV") as f: # "with open" ensures automatic file closure
    reader = csv.reader(f)
    header = next(reader) # Process 1st row as the list header
    intake_data = [row for row in reader] # Iteratively add each subsequent row of data as a sub-list
```

Code example 1 may be obvious to you, but may not be obvious to others. It may be obvious to you now, but does not mean that you will remember it.

Code example 2 is obvious to you and others. It is obvious to you now and in future when you do not remember it.

Be convinced and write comments in your codes to allow yourself and the user to understand exactly what your code meant to do now as well as in the future.

Appendix A – Tips for using IDLE

IDLE is Python's Integrated Development and Learning Environment. It is installed with Python installation.

What does it provide?

- Cross-platform, available on Windows, Unix and MacOS.
- Interactive Python interpreter (shell window)
- Multi-window text editor with syntax highlighting, code completion
- Debugger with breakpoints, stepping, viewing of variables

Read-Eval-Print-Loop (REPL)

The default window of IDLE is an interactive Read-Eval-Print-Loop (REPL) environment, where user can type command directly. The interpreter will

- Reads the command entered by user
- Evaluate and execute the command
- Print the output (if any) to the console
- Loop back and repeat the process

Autocomplete

REPL has autocompletion feature. Use TAB key to activate autocompletion.

Type pr and press TAB key

```
Type "help", "copyright", "  
>>> print('Hello world')  
Hello world  
>>>  
>>> pr  
print  
property  
quit  
range  
repr
```

A screenshot of the IDLE Read-Eval-Print-Loop (REPL) window. It shows a sequence of commands and outputs: 'Type "help", "copyright", "' followed by '>>> print('Hello world')' which outputs 'Hello world'. Then '>>>' is entered, followed by '>>> pr'. A dropdown menu appears below 'pr', listing 'print', 'property', 'quit', 'range', and 'repr'. The 'print' option is highlighted in blue. A small upward arrow icon is visible to the right of the list.

Copy a line

Use UP arrow key to move up to previous command, press Enter key.

The command at the cursor will be copied

Execute command with conditional statements, loops etc

Type command and press Enter key twice to execute

Appendix B – Jupyter keyboard shortcuts

Reference: <https://towardsdatascience.com/jupyter-notebook-shortcuts-bf0101a98330>

First, we need to know that they are 2 modes in the Jupyter Notebook app: command mode and edit mode. The left side of the cell is blue when it is in command mode. It is in green when it is in edit mode.

Shortcuts in both modes:

Shift + Enter: run the current cell, select below

Ctrl + Enter: run selected cells

Alt + Enter: run the current cell, insert below

Ctrl + S: save and checkpoint

While in command mode (press Esc to activate):

Enter: take you into edit mode

H: show all shortcuts

Up: select cell above

Down: select cell below

Shift + Up: extend selected cells above

Shift + Down: extend selected cells below

A: insert cell above

B: insert cell below

X: cut selected cells

C: copy selected cells

V: paste cells below

Shift + V: paste cells above

D, D (press the key twice): delete selected cells

Z: undo cell deletion

S: Save and Checkpoint

Y: change the cell type to Code

M: change the cell type to Markdown

P: open the command palette.

While in edit mode (press Enter to activate)

Esc: take you into command mode

Tab: code completion or indent

Shift + Tab: tooltip

Ctrl +]: indent

Ctrl + [: dedent

Ctrl + A: select all

Ctrl + Z: undo

Ctrl + Shift + Z or Ctrl + Y: redo

Ctrl + Home: go to cell start

Ctrl + End: go to cell end

Ctrl + Left: go one word left

Ctrl + Right: go one word right

Ctrl + Shift + P open the command palette

Down: move cursor down

Up: move cursor up

Appendix C – The Importance of Good Comments: A Tale of the Baker's Apprentice

As part of his world renowned teaching program, the famous baker Breadtista insisted that each apprentice thoroughly document his or her work. Thus, apprentices created a reference that would live beyond their apprenticeship. All recipes, tips, and tricks would be clearly documented. Breadista strongly believed that such a resource provided his apprentices with a significant advantage.

As Breadtista was fond of pointing out, the key to a great recipe was capturing the perfect amount of information. Where other bakers wrote recipe books filled with lists of bland instructions, such as: "Add two eggs and blend well", Breadtista insisted on commenting his recipes. These comments added context and explanation.

Here is an excerpt from Breadtista's famous eight-chili strawberry muffins:

Step 5: Cut the strawberries into cubes of 1cm per side. [This size allows the flavors to blend while preserving the fruit's texture.]

Step 6: Add the strawberries to the dough.

Step 7: Gently stir until the strawberries are mixed into the dough.

Since comments were not part of the instructions themselves, Breadtista clearly denoted his comments with braces [] to avoid confusing other bakers.

Further, Breadtista insisted that all comments provide meaningful information to the recipe itself. He hated recipe books by famous castle chefs that contained thirty page digressions of the author's lifelong quest for the perfect pancake. It was especially egregious if the recipe then produced chewy pancakes. Pancakes were not meant to be chewy, and comments were not meant to be life stories.

Despite his firm belief in the importance of good comments, Breadtista found it difficult to explain this philosophy to each new apprentice. Every year, he heard the same arguments:

- "Why should I comment? The only people reading this are bakers, and they should already know this."
- "I won't forget why I did that."
- "Isn't that step obvious?"

And every year, he repeated the same answers:

- "Just because it is obvious to you, does not mean that it will be obvious to others."
- "Just because it is obvious to you now, as you are writing the recipe, does not mean that you will remember it."
- "If it is so obvious, why did I have to draw you an example on the chalk board? Twice?"

Each year, after lecturing on the importance of clear comments, Breadtista would have at least one apprentice who took the idea too far. For example, one year he had an apprentice who wrote the following instructions for making simple bread:

Step 1: Measure out 1/4 of an ounce of yeast into a small bowl. [We put it into a bowl so that we can add water in step 2 to activate the yeast.]

Step 2: Add two table spoons of warm water to the bowl with the yeast. [We add the water to the bowl in order to activate the yeast.]

Step 3: Wait 5 minutes. [We wait 5 minutes in order to allow the yeast to activate in the water].

Breadtista never made it past step 85 in that recipe. He "accidentally" spilled a full pitcher of water over the apprentice's parchment so as to spare anyone else from reading it.

It took ten years of trial and error, but eventually Breadtista found the perfect way to reinforce the importance of GOOD comments. The first three weeks of each apprenticeship, Breadtista would make each new apprentice work from the worst notes of previous students. Although it meant that he would now hear cries of "WHO WROTE THIS?" and "BUT WHAT DOES IT MEAN?" from the kitchen, this process saved him from having to repeat the same reasoning over and over.

Nothing reinforces the concept of understandable, well documented recipes more than having to read other people recipes.

Computer science concepts as told through fairy tales.

By Jeremy Kubica

Chapter 2: Data Types and Operators

Contents

- 1 Data Types
 - 1.1 Type conversion
 - 1.2 Input
 - 1.3 Output
- 2 Numbers
 - 2.1 Integer in different bases
 - 2.2 Arithmetic Operators
- 3 Boolean
 - 3.1 Boolean operators
 - 3.2 Logical operators
- 4 Strings
 - 4.1 ASCII code and Unicode
 - 4.2 Create Strings
 - 4.3 Escape character
 - 4.4 Indexing
 - 4.5 Slicing
 - 4.6 Immutable
 - 4.7 String operators
 - 4.8 Built-in String methods

Syllabus Learning Outcomes

- 2.2 Programming Elements and Constructs
 - Use programming language elements and constructs to write recursive and non-recursive programs to solve a variety of problems.
 - 2.2.1 Understand the different types: integer, real, char, string and Boolean and initialise arrays (1- dimensional and 2-dimensional).
 - 2.2.2 Use common library functions for input/output, strings and mathematical operations.
- 3.1 Data Representation
 - Understand that values can be represented in different number bases: denary, binary and hexadecimal.
 - 3.1.1 Represent data in binary and hexadecimal forms.
 - 3.1.2 Write programs to perform the conversion of positive integers between different number bases: denary, binary and hexadecimal forms; and display results.
- 3.2 Character Encoding
 - Understand the use of ASCII code and Unicode to represent characters.
 - 3.2.1 Give examples of where or how Unicode is used.
 - 3.2.2 Use ASCII code in programs.

1 Data Types

Multiple data types are developed for convenient data manipulation and management. In the table below are some of the more common data types:

Data Types	Description	Example
Integer	A signed whole number	1, 1234, -15
Float (real)	A signed number with a decimal point	1.0, 2.345, - 54.321
Boolean	The logical values TRUE and FALSE	TRUE, FALSE
String	A sequence of zero or more characters	'hello', 'action 123', 'A-level', "

The data type is set when you assign a value to a variable. The `type()` function can be used to check the data type of a variable.

```
1 # Example on integer and float (real numbers)
2 num1 = 123
3 num2 = 12.3
4
5 print(type(num1), type(num2))
```

```
<class 'int'> <class 'float'>
```

```
1 # Example on char and string
2 check = True
3 string1 = 'hello!'
4
5 print(type(check), type(string))
```

```
<class 'bool'> <class 'str'>
```

1.1 Type conversion

We can convert one type of number into another. Operations like addition, subtraction can convert integer to float implicitly (automatically), if one of the operands is float.

```
1 print(1 + 2.0)
```

```
3.0
```

We can also use built-in functions like `int()`, `float()` and `str()` to convert between types explicitly.

```

1  # Example of converting float to int and vice versa
2  num1 = 2.3
3  num2 = -2.8
4  num3 = 5
5
6  print(type(num1), type(num2), type(num3))
7
8  print(type(int(num1)), type(int(num2)), type(float(num3)))
9
10 # Example of converting int to str
11
12 s = str(num2)
13 print(type(s))
14

```

<class 'float'> <class 'float'> <class 'int'>
 <class 'int'> <class 'int'> <class 'float'>
 <class 'str'>

1.2 Input

We can ask user for input using Python built-in `input()` function.

- When `input()` function is called, the program will stop and wait for user to key in data.
- It is optional to add prompt as function parameter.
- It always returns a string value.

```

1  # Example of using input()
2
3  age = input("Enter your age: ")
4  print(type(age))

```

Enter your age: 5
 <class 'str'>

Notice that even when the value entered by user is an integer, the returned value will still be of the `str` type. We need to convert the type of the returned value using the built-in functions for further manipulation of the returned value if required.

```

1  # Example of using input()
2
3  age = input("Enter your age: ")
4  print(type(int(age)))

```

Enter your age: 5
 <class 'int'>

1.3 Output

Python provides a built-in function `print()` to display data to the console or a file object.

The `print()` function

- can accept multiple values, and uses `sep` value to separate them in output. By default, the `sep` value is a space,
- automatically appends an end string at the end. By default, the end value is a new line (`\n`), and
- outputs data to a file object defined in `file`, which has a default value `sys.stdout` (console).

```
1 # Examples of different output using print()
2 print('hello', 'world')
3 print()
4 print('have', 'a', 'good', 'day', sep='-')
```

hello world

have-a-good-day

```
1 # output is printed in a new line
2 print("Hello")
3 print("World!")
4
5 print()
6
7 # append a space at the end instead of a new line
8 print("Hello", end = " ")
9 print("World!")
```

Hello
World!

Hello World!

2 Numbers

2.1 Integer in different bases

The numbers we deal with every day are of the decimal (base 10) number system. However, computer programmers (generally embedded programmers) need to work with binary (base 2), hexadecimal (base 16) and octal (base 8) number systems.

In Python, we can represent these numbers by appropriately placing a prefix before that number. The following table lists these prefixes.

Number System	Prefix
Binary	'0b' or '0B'
Octal	'0o' or '0O'
Hexadecimal	'0x' or '0X'

```

1  # Example of binary numbers
2  print(0b1101011)
3
4  # Example of addition of hexadecimal and binary numbers
5  print(0xFB + 0b10)
6
7  # Example of octal numbers
8  print(0o15)

```

107

253

13

We can use `bin()` and `hex()` to convert a decimal number to a binary and hexadecimal respectively. We can also use `int()` to convert other number systems to decimals.

```

1  x = 100
2
3  # convert decimal to binary
4  print(bin(x))
5
6  # convert decimal to hexadecimal
7  print(hex(x))
8
9  y = '100'
10 # convert binary to decimal
11 print(int(y,2))
12
13 # convert hexadecimal to decimal
14 print(int(y,16))

```

0b1100100

0x64

4

256

2.2 Arithmetic Operators

Arithmetic operators perform common mathematical operations on values.

Operator	Name	Expression	Example	Answer
+	Addition	$x + y$	2+3	5
-	Subtraction	$x - y$	7-3	4
*	Multiplication	$x * y$	2*3	6
/	Division	x / y	20/8	2.5
//	Integer division	$x // y$	20//8	2
%	Modulus	$x \% y$	20%8	4
**	Exponentiation	$x ** y$	4**2	16


```

1  # Examples of using arithmetic operators
2  x = 8
3  y = 4
4
5  # Addition
6  a = x+y
7  print(a)
8  print(type(a))
9
10 # Subtraction
11 print(x-y)
12
13 # Multiplication
14 print(x*y)
15
16 # Division
17 b=x/y
18 print(b)
19 print(type(b))
20
21 # Floor Division (returns quotient without remainder)
22 print(x//y)
23
24 # Modulus (Remainder)
25 print(x%y)
26
27 # Exponentiation (x to the power of y)
28 print(x**y)
29

```

```

12
<class 'int'>
4
4
2.0
<class 'float'>
2
0
4096

```

Sometimes we want to do more than just usual arithmetic operation. Python has a built-in function that gives us access to more functions.

```

1 # Example of importing math module
2
3 import math
4
5 print(math.pi)

```

3.141592653589793

3 Boolean

A Boolean variable can represent either `True` or `False`.

3.1 Boolean operators

Comparison operators are used to compare two values. It returns a boolean value.

Operator	Name	Example
>	Greater than	<code>x > y</code>
<	Less than	<code>x < y</code>
>=	Greater than or equal to	<code>x >= y</code>
<=	Less than or equal to	<code>x <= y</code>
==	Equal	<code>x == y</code>
!=	Not equal	<code>x != y</code>

```

1 # Example on boolean types
2 print(type(True), type(False))
3
4 print(5 > 4)
5 print(5 < 4)
6 print(5 == 4)
7 print(5 != 4)

```

```

<class 'bool'> <class 'bool'>
True
False
False
True

```

An object can also be evaluated to be `True` or `False` using built-in `bool()` function.

```
1 print(bool(2))
2 print(bool(-2))
3 print(bool(0.2))
4 print(bool(0))
5 print(bool('Hi'))
6 print(bool(''))
```

```
True
True
True
False
True
False
```

3.2 Logical Operators

Sometimes we need to make decisions based on multiple conditions. Logical operators are used to combine conditional statements.

- Operands shall be conditions which can result in a boolean value.
- The outcome of such an operation is either true or false too.

Operator	Description	Example
and	Returns True if both statements are true	$x < 5$ and $x < 10$
or	Returns True if at least one of the statements is true	$x < 5$ or $x < 10$
not	Reverse the result, returns False if the result is true	not($x < 5$ and $x < 10$)

Given that $x = 6$, predict the outcome in the "Example" column from the table above.

4 Strings

Characters in programming are not limited to the 26 English characters. It can also be space character, question mark, Chinese character, etc.

A string is a sequence of characters. It can be a word, a sentence or a paragraph.

4.1 ASCII code and Unicode

Today's programs need to be able to handle a wide variety of characters. Applications are often internationalized to display messages and output in a variety of user-selectable languages; the same program might need to output an error message in English, French, Japanese, Hebrew, or Russian. Web content can be written in any of these languages and can also include a variety of emoji symbols.

Two standard character sets used in displaying text are ASCII code and Unicode.

ASCII stands for American Standard Code for Information Interchange. The ASCII character set is a 7-bit set of codes that allows 128 different characters. That is enough for every upper-case letter, lower-case letter, digit and punctuation mark on most keyboards. ASCII is only used for the English language.

This table shows some examples of letters represented using the ASCII character set:

Character	Denary value	Binary value	Hex
N	78	1001110	4E
O	79	1001111	4F
P	80	1010000	50
Q	81	1010001	51
R	82	1010010	52

Extended ASCII code is an 8-bit character set that represents 256 different characters, making it possible to use characters such as é or ©. Extended ASCII is useful for European languages.

Unicode uses between 8 and 32 bits per character (UTF-8, UTF-16 and UTF-32), so it can represent characters from languages from all around the world.

```

1 # Example of using unicode
2 print('\u7434')
```

琴

It is commonly used across the internet. As it is larger than ASCII, it might take up more storage space when saving documents. Global companies, like Facebook and Google, would not use the ASCII character set because their users communicate in many different languages.

We can get the ASCII code of a character by using the build-in function `ord()`. We can also retrieve the character from an ASCII code by using `chr()`.

```

1  # Getting ASCII code of character
2  ascii_code = ord("A")
3
4  # Retrieve character from ASCII code
5  character = chr(ascii_code)
6
7  print(ascii_code)
8  print(character)

```

65

A

4.2 Create Strings

String is one of the most commonly used data type in Python. Strings in Python are objects.

There are 3 ways to declare strings. Strings can be declared in either single-quote, double-quotes or triple double-quotes.

```

1  x = 'hi'
2  y = "hello"
3  z = """world"""

```

```

1  # Using single quotes
2  m = 'I\'m here'
3
4  # Using single and double quotes
5  n = "You're there"
6
7  print(m)
8  print(n)

```

I'm here

You're there

Triple double-quotes are commonly used to initialize multi-line strings.

```

1  z = """How
2  are
3  you?"""
4  print(z)

```

How

are

you?

Triple quotes are also used to create docstring, which we will go into more detail when we learn functions.

4.3 Escape character

In Python strings, the backslash `"\"` is a special character, also called the "escape" character. It is used in representing certain whitespace characters: `"\t"` is a tab, `"\n"` is a newline.

```
1 print('apple\torange') # tab between apple and orange
2 print('apple\norange') # print on a new line after apple
```

```
apple    orange
apple
orange
```

Conversely, prefixing a special character with `"\"` turns it into an ordinary character. This is called "escaping". For example, `"\"` is the single quote character.

```
1 # Both print statements give the same output
2 print('It\'s raining')
3 print("It's raining")
```

```
It's raining
It's raining
```

Likewise, `"\"` can be escaped: `"\"hello\""` is a string begins and ends with the literal double quote character. Finally, `"\"` can be used to escape itself: `"\""` is the literal backslash character.

```
1 print("\"hello\"")
2 print("'\" is the backslash')
```

```
"hello"
"\" is the backslash
```

The escape character can also simply be used to break a long line of code.

```
1 if 5 > 2 and 2 < 8 and \
2 3 < 5:
3     print("This is \
4     correct!")
```

```
This is correct!
```

4.4 Indexing

Each character in a string can be accessed by its index (location).

In Python, index value starts from 0. Python also supports negative indexes. Index of '-1' represents the last character of the String.

```
1 x = "spiderman"
2 print(x[0]) # refers to first character of string
3 print(x[1]) # refers to second character of string
4 print(x[-1]) # refers to last character of string
5 print(x[-2]) # refers to second last character of string
```

s
p
i
d
e
r
m
a
n

The `len()` function is one useful method that returns the number of items in an object. When the object is a string, the `len()` function returns the number of characters in the string.

```
1 x = "spiderman"
2
3 print(len(x)) # prints the number of characters in the string x
```

9

The function `len()` can also be used in indexing.

```
1 # different ways to print out the last character in a string
2 print(x[len(x)-1]) # refers to last character of string, x[len(x)-1] same as x[9-1]
3 print(x[-1])
4 print(x[8])
```

```
n
n
n
n
```

What will be the output for `print(x[20])`?

4.5 Slicing

Slicing is to retrieve a range of characters in a String.

The `x[a:b]` returns all characters in `x` from index `a` to `b-1`.

- Note: Character at index 'b' is not included in output.

The `x[:b]` returns all characters before index `b`.

- Start-index will be set to 0 if it is omitted.
- Character at index 'b' is not included.

The `x[a:]` returns all characters from index `a`.

- End-index will be length of `x` if it is omitted.
- Character at index 'a' is included.

```
1 # Examples on slicing
2 x = "spiderman"
3
4 print(x[0:3]) # print characters from index 0 to 2
5 print(x[:3]) # print characters from index 0 to 2
6 print(x[3:]) # print characters from index 3 to last character
7 print(x[3:len(x)]) # print characters from index 3 to last character
8 print(x[2:5]) # print charactes from index 2 to 4
```

```
spi
spi
derman
derman
ide
```

What will be the output for `print(x[:20])`, `print(x[1:-1])` and `print(x[::-1])`?

4.6 Immutable

An immutable object cannot be changed after it is created.

String objects are immutable!

Do you think I can change the letter 'r' in the String "spiderman" to capital letter by doing `x[0] = 'R'`? Test it out.

4.7 String operators

Operator	Name	Expression
+	Concatenation	<code>x + y</code>
*	Repetition	<code>x * y</code>
in	Exists	<code>x in y</code>
not in	Not Exists	<code>x not in y</code>

```

1  # Concatenation
2  s1 = "spider"
3  s2 = "man"
4  print(s1 + s2)
5
6  # Repetition
7  s = "Ha"
8  print(s * 4)
9
10 # Returns True if a string is present in another string, otherwise returns False
11 print("spider" in s1)
12
13 # Returns True if a string is NOT present in another string, otherwise returns False
14 print("bat" in s1)

```

```

spiderman
HaHaHaHa
True
False

```

4.8 Built-in String methods

String object comes with a number of built-in methods to manipulate strings. All string methods returns new values. They do not change the original string.

Method	Description
count()	Counts how many times str occurs in string or in a substring of string
find()	Determine if str occurs in string or in a substring of string, returns index if found and -1 otherwise.
Index()	Same as find(), but raises an exception if str not found.
lower()	Converts all letters in string to lowercase.
upper()	Converts all letters in string to uppercase.
title()	Converts the first character of each word to upper case
strip()	Remove white space characters from the beginning or the end by default. One or more characters can be specified to be stripped.
lstrip()	Removes all leading whitespace in string.
rstrip()	Removes all trailing whitespace of string.
isalnum()	Returns true if string has at least 1 character and all characters are alphanumeric and false otherwise.
isalpha()	Returns true if string has at least 1 character and all characters are alphabetic and false otherwise.
isdigit()	Returns true if string contains only digits and false otherwise.
islower()	Returns true if string has at least 1 cased character and all cased characters are in lowercase and false otherwise.
isnumeric()	Returns true if a unicode string contains only numeric characters and false otherwise.
isupper()	Returns true if string has at least one cased character and all cased characters are in uppercase and false otherwise.
isspace()	Returns true if string contains only whitespace characters and false otherwise.
startswith()	Returns True if the string starts with the specified value
endswith()	Returns true if the string ends with the specified value

String formatting

The format() method formats the specified value(s) and insert them inside the string's placeholder.

The placeholder is defined using curly brackets: {}. The format() method returns the formatted string. It is the recommended string formatting tool. It automatically handle data conversion for some data types.

```
1 # Example on format
2 print('Hi {}'.format('Peter')) # the placeholder{} will be replaced by 'peter'
```

Hi Peter

The `format()` method takes unlimited number of arguments, and are placed into the respective placeholders.

```
1 name = 'Mark'
2 age = 18
3 print('{} is {} years old'.format(name,age))
```

Mark is 18 years old

Index numbers can be used, for eg. `{0}`, to be sure the arguments are placed in the correct placeholders. The same index number can be used more than once.

```
1 print('His name is {1}. {1} is {0} years old'.format(age, name)) #positioning
```

His name is Mark. Mark is 18 years old

The `format()` method can also be used to format numbers.

```
1 print(type('{:0>6}'.format(1233333333))) # format a number into a fixed number of digits with zeros appended to the front
2 print(type('{:x}'.format(10))) # converts an integer into hexadecimal
3 print('{0:o}'.format(10)) # converts an integer into an octal number
4 print('{0:b}'.format(10)) # converts an integer into a binary number
5 print('{:.3f}'.format(3.1415)) # format a number to be displayed as a number with specific number of decimal places
6 print('{:.3}'.format(3.1415)) # format a number to be displayed as a number with specific number of significant figures
7
8 print('{:10.3}'.format(3.1415)) # format a number to be displayed with 10 column width, right align by default
9 print('{:<10.3}'.format(3.1415)) # format a number to be displayed with 10 column width and left align
10 print('{:^10.3}'.format(3.1415)) # format a number to be displayed with 10 column width and center
```

```
<class 'str'>
<class 'str'>
12
1010
3.142
3.14
      3.14
3.14
      3.14
```

Chapter 3: Control Flow

Contents

- 1 What is control flow?
 - 1.1 Algorithm
 - 1.2 Pseudo code
 - 1.3 Flow chart
- 2 Control flow statements
 - 2.1 If
 - 2.1.1 Short hand if
 - 2.1.2 Short hand if-else
 - 2.1.3 Evaluating non-zero numerical values
 - 2.1.4 Evaluating non-empty strings
 - 2.1.5 Built-in bool() method
 - 2.2 While
 - 2.3 For
 - 2.3.1 The range() function
 - 2.3.2 Iterating by Sequence Index
 - 2.4 Break, continue and pass
 - 2.4.1 Break
 - 2.4.2 Continue
 - 2.4.3 Pass
- 3 Decision table

Syllabus Learning Outcomes

- 1.1 Algorithmic Representation

Write algorithms in pseudo-code and flowchart for given problems.

 - 1.1.1 Use appropriate techniques or tools such as pseudo-code and flowchart to show program flow.
 - 1.1.2 Use standard flowchart symbols.
 - 1.1.3 Use a combination of various control structures.
 - 1.1.4 Use decision tables to explore the actions for combinations of different input conditions. Note: up to three conditions
- 2.2 Programming Elements and Constructs

Use programming language elements and constructs to write recursive and non-recursive programs to solve a variety of problems.

 - 2.2.3 Apply the fundamental programming constructs to control the flow of program execution:
 - Sequence
 - Selection
 - Iteration

1 What is control flow?

In the examples that we have seen till now, there has always been a series of statements faithfully executed by Python in exact top-down order. What if you wanted to change the flow of how it works? For example, you want the program to take some decisions and do different things depending on different situations, such as printing 'Good Morning' or 'Good Evening' depending on the time of the day?

As you might have guessed, this is achieved using control flow statements. There are three control flow statements in Python - if, for and while.

1.1 Algorithm

An algorithm is a step-by-step procedure (well-defined instructions) to solve a given problem.

For instance, the algorithm to find the sum of two numbers is

1. Take two numbers
2. Add the numbers
3. Display the result

Or the algorithm to log in to VJC portal

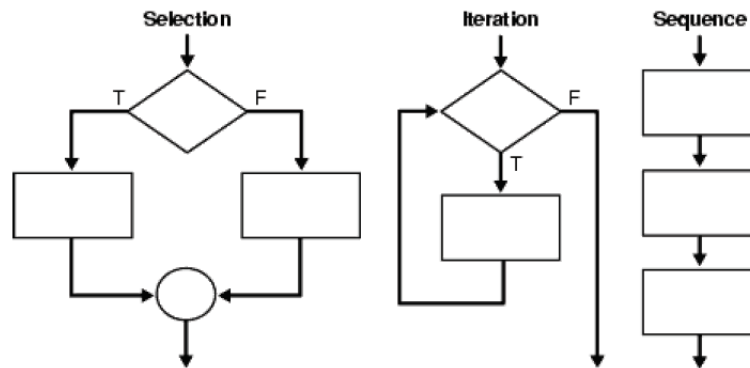
1. Go to <https://portal.vjc.sg/>
2. Enter vjc email and password
3. Click on Sign in button
4. Enter OTP
5. Click on Submit button

Sometimes there might be different steps depending on some other conditions. For example, if the day is a weekday, you will wake up at 6:00 am, else you will wake up at 10:00 am. Many problems have more than one solution. Sometimes it is a personal preference which solution to choose. Sometimes one solution will be measurably better than another.

We can write algorithm using pseudo code or flowchart.

1.2 Pseudo code

Pseudo code is a description of the steps in an algorithm using natural language (such as English) mixed with sequence, selection and iteration constructs.



Sequence construct is when a number of steps are performed one after the other.

Selection construct allows some steps to be performed under certain conditions, otherwise different steps (or no steps) are performed.

Repetition construct is when a sequence of steps is performed a number of times. This is also known as iteration or looping.

A general idea to solving a problem involves inputting data to the computer, processing the data and outputting results.

	English	Pseudo code	
		Example	General
Assignment and sequence	Set A to 34 Increment B	A $\leftarrow$ 34 B $\leftarrow$ B + 1	<identifier> $\leftarrow$ <value>
Selection	If A is greater than B then do something, else do another thing	IF A > B THEN Do something ELSE Do another thing ENDIF	IF <condition> THEN <statements> ELSE <statements> ENDIF
Repetition	Repeat some steps until A is equal to B	REPEAT Some steps UNTIL A = B	REPEAT <statements> UNTIL <condition>
	While Num is smaller than 10, increment Num	WHILE Num < 10 DO Num $\leftarrow$ Num + 1 ENDWHILE	WHILE <condition> DO <statements> ENDWHILE
	For index 1 to 30, assign an empty string to Names	FOR Index $\leftarrow$ 1 TO 30 Names[Index] $\leftarrow$ '' ENDFOR	FOR <identifier> $\leftarrow$ <value1> TO <value2> <statements> ENDFOR
Input	Take in an input and assign to A with and without a prompt	INPUT A INPUT "Enter name: " A	INPUT <identifier>
Output	Output/print "Message" Output B	OUTPUT "Message" OUTPUT B OUTPUT "Your age is ", B	OUTPUT <values(s)>

A condition consists of at least one logic proposition. Logic propositions use the relational operators.

Operator	Comparison
=	Is equal to
<	Is less than
>	Is greater than
<=	Is less than or equal to
>=	Is greater than or equal to
<>	Is not equal to

The pseudo code presented in the table is the recommended style of writing pseudo code in examination papers. In the recommended style, keywords are in uppercase with indentation at appropriate parts of the algorithm.

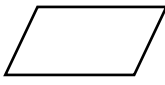
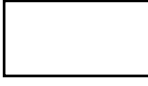
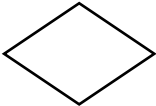
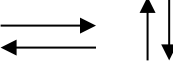
However, there is no fixed or defined syntax. It is acceptable as long as the logic correctly solves the problem.

1.3 Flow chart

A flowchart is the graphical or pictorial representation of an algorithm with the help of different symbols, shapes, and arrows to demonstrate a process or a program.

There are 6 basic symbols commonly used in Flowchart

(We focus only on the first 5)

Symbols	Name	Description
	Start/End	Indicates the starting or ending of the program
	Input/Output	Use for Input/Output(I/O) operation i.e. taking input and showing output
	Process	Indicates any type of internal operations like initialization, calculation etc.
	Decision	Use for asking questions that can have either TRUE or FALSE (YES or NO) as an answer
	Control flow	Show direction of flow
	Connector	Connectors are used to connect breaks in the flowchart. If a flowchart takes more than one page, then we use the connector to connect the flowchart between pages. (However, A-level questions on flowchart are usually within 1 page.)

2 Control flow statements

There are three control flow statements in Python: `if`, `for` and `while`.

2.1 IF statement

The `if` statement is a selection construct that is used to check a condition: if the condition is true, we run a block of statements (called the if-block), else we process another block of statements (called the else-block). The else clause is optional.

If we need to have another if-else statement in an else-block, we can simplify the code using if-elif-else statement.

Syntax	Flowchart
<pre>if <condition>: <statement></pre>	<pre> graph TD Entry(()) --> Cond{condition} Cond -- True --> S1[Statement 1] S1 --> Exit1(()) Cond -- False --> Exit1 style Entry fill:none,stroke:none style Exit1 fill:none,stroke:none </pre>
<pre>if <condition>: <statement> else: <statement></pre>	<pre> graph TD Entry(()) --> Cond{condition} Cond -- True --> S1[statement1] Cond -- False --> S2[statement2] S1 --> Exit1(()) S2 --> Exit1 style Entry fill:none,stroke:none style Exit1 fill:none,stroke:none </pre>
<pre>if <condition>: <statement> elif <condition>: <statement> else: <statement></pre>	<pre> graph TD Entry(()) --> E1{expression 1} E1 -- True --> S1[statement 1] E1 -- False --> E2{expression 2} E2 -- True --> S2[statement 2] E2 -- False --> S3[statement 3] S1 --> Exit(()) S2 --> Exit S3 --> Exit style Entry fill:none,stroke:none style Exit fill:none,stroke:none </pre>

```

1 # Example of using if statement
2 a = int(input("Enter an integer: "))
3 b = int(input("Enter another integer: "))
4
5 if a < b:
6     print('a is smaller than b!')

```

Enter an integer: 3
Enter another integer: 4
a is smaller than b!

```

1 # Example of using if-else statement
2 a = int(input("Enter an integer: "))
3 b = int(input("Enter another integer: "))
4
5 if a < b:
6     print('a is smaller than b!')
7 else:
8     print('a is larger than b!')

```

Enter an integer: 3
Enter another integer: 2
a is larger than b!

```

1 # Example of using if-elif-else statement
2 a = int(input("Enter an integer: "))
3 b = int(input("Enter another integer: "))
4
5 if a < b:
6     print('a is smaller than b!')
7 elif a > b:
8     print('a is larger than b!')
9 else:
10    print('a is equal to b!')

```

Enter an integer: 34
Enter another integer: 34
a is equal to b!

When there is more than one condition to be evaluated, logical operators can be used.

```

1 num = int(input("Enter an integer between 0 and 100 (inclusive): "))
2
3 if 0 <= num and num <= 100:
4     print("Well done!")
5 else:
6     print("Number is not between 0 and 100.")

```

Enter an integer between 0 and 100 (inclusive): 8
Well done!

2.1.1 Short hand if

If you have only one statement to execute, you can put it on the same line as the `if` statement.

```
1 a = 10
2 b = 3
3
4 if a > b: print("a is greater than b")
```

a is greater than b

2.1.2 Short hand if-else

If you have only one statement to execute, one for `if`, and one for `else`, you can put it all on the same line.

```
1 score = int(input("Enter your score: "))
2
3 print('pass') if score >= 50 else print('fail')
```

Enter your score: 60
pass

2.1.3 Evaluating non-zero numerical values

Non-zero values are evaluated to `True`

Zero (0 or 0.0) is evaluated to `False`

```
1 age = int(input("Enter your age: "))
2
3 if age:
4     print("You are " + str(age) + " years old!")
5 else:
6     print("You cannot be 0 years old!")
```

Enter your age: 0
You cannot be 0 years old!

2.1.4 Evaluating non-empty strings

Non-empty strings are evaluated to `True`.

Empty string is evaluated to `False`.

```
1 name = input("Enter your name: ")
2
3 if name:
4     print("Hi " + name + "!")
5 else:
6     print("You cannot have no name!")
```

Enter your name:
You cannot have no name!

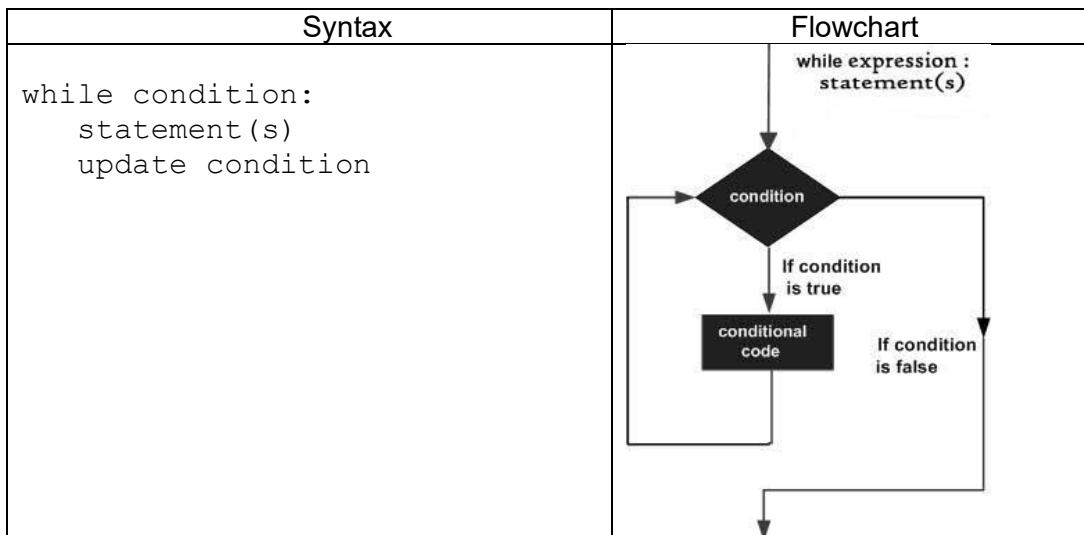
2.2 WHILE

In general, statements are executed sequentially. There may be a situation when you need to execute a block of code a number of times.

A loop statement allows us to execute a statement or group of statements multiple times.

The while statement allows you to repeatedly execute a block of statements as long as a condition is true. A while statement is an example of a looping statement.

It is important that the condition will eventually become False, else the program will go into an infinite loop.



Here, statement(s) may be a single statement or a block of statements. The loop iterates and execute the statement(s) while the condition is true. The condition may be any expression.

When the condition becomes false, program control passes to the line immediately following the loop.

All the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code.

```
1 count = 1
2 while count <= 5:
3     print(count)
4     count = count + 1
```

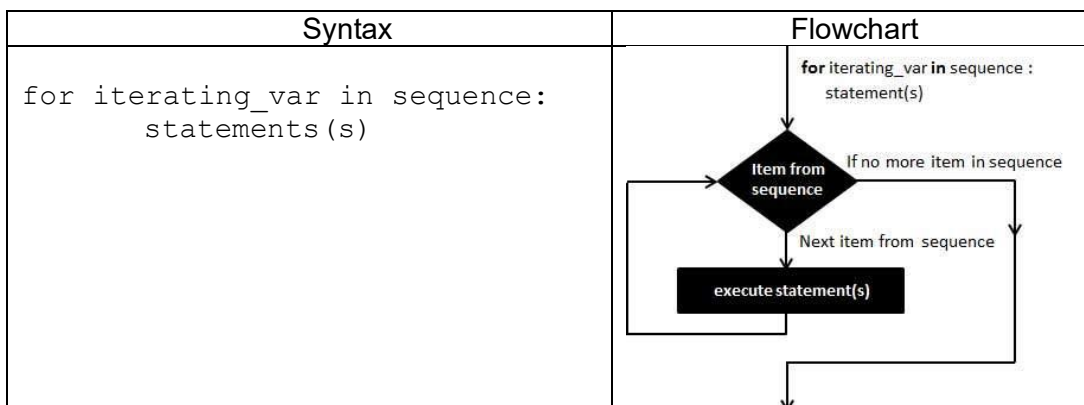
```
1
2
3
4
5
```

The block here, consisting of the print and increment statements, is executed repeatedly until count is no longer less than and equal to 5.

Note that there will be a possibility of a while loop running into an infinite loop. A loop becomes an infinite loop if a condition never becomes FALSE. Use while loop with caution and always check that the condition will be updated to become TRUE for the program control to exit the while loop.

2.3 FOR

A `for` loop is used for iterating over the items of any sequence, such as a string or a list etc.



The first item in the sequence is assigned to the iterating variable `iterating_var`. Next, the statements block is executed. Each item in the sequence is assigned to `iterating_var`, and the statement(s) block is executed until the entire sequence is exhausted.

```

1 # for each letter in the string "banana"
2 # print the letter on a new line
3 for letter in "banana":
4     print(letter)
```

```

b
a
n
a
n
a
```

2.3.1 The range() function

To loop through a set of code a specified number of times, we can use the `range()` function

The `range()` function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

```
1 for x in range(5):  
2     print(x)
```

```
0  
1  
2  
3  
4
```

Note that `range(5)` is not the values of 0 to 5, but the values 0 to 4.

The `range()` function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: `range(2, 5)`, which means values from 2 to 4 (but not including 5):

```
1 for x in range(2,5):  
2     print(x)
```

```
2  
3  
4
```

The `range()` function defaults to increment the sequence by 1, however, it is possible to specify the increment value by adding a third parameter: `range(2, 11, 2)`:

```
1 for x in range(2,11,2):  
2     print(x)
```

```
2  
4  
6  
8  
10
```

2.3.2 Iterating by Sequence Index

An alternative way of iterating through each item in a sequence is by index offset into the sequence itself.

```

1 word = "banana"
2
3 for index in range(len(word)):
4     print(word[index])

```

b
a
n
a
n
a

Generally, if we do not know in advance how many times to execute a block of statements, we can choose to use `while` loop. If we know the number of times to execute the code, we can use `for` loop.

2.4 Break, continue and pass

Using loops in programming allows automation of tasks efficiently. However, there may be cases when we want to exit the loop prematurely, skip an iteration or we are not sure what we want to do (or should do) in the loop. In this section, we will introduce 3 statements to achieve the actions stated above.

2.4.1 Break

In Python, the `break` statement provides you with the opportunity to exit a loop prematurely before looping through a sequence completed.

```

1 for x in "spiderman":
2     if x == "m":
3         break
4     print(x)

```

s
p
i
d
e
r

In the example above, the program control exited the loop when `x` is 'm', before 'm' is being printed.

2.4.2 Continue

With the `continue` statement we can stop the current iteration of the loop, and continue with the next.

```
1 count = 0
2
3 while count < 5:
4     count += 1
5     if count == 3:
6         continue
7     print(count)
```

```
1
2
4
5
```

In the example above, all the numbers from 1 to 5 are printed except when the number is 3.

2.4.3 Pass

The `pass` statement acts as a "placeholder" in the development of code, when we are not sure how (or what) to do when a condition is met.

```
1 for n in range(5):
2     if n == 2:
3         pass
4     print(n)
```

```
0
1
2
3
4
```

Will the above code still run if `pass` is omitted?

3 Decision table

A decision table is a precise way of modelling logic. A decision table is a tool for documenting complicated logic, which is a part of some business problem.

The aim is to state all possible combinations of conditions and the respective actions that result.

A Decision Table is usually divided into four quadrants as in the table below.

Condition statements (Conditions being tested)	Condition alternatives
Action statements (Possible actions to take)	Action entries

Let's look at an example.

A shop gives some customers a discount on goods totaling more than \$20. The discounts are:

- 5% for goods totaling more than \$100
- 5% with a discount card
- 10% with a discount card and goods totaling more than \$100

Step 1: Analyse the requirements and create the first column

Identify the conditions that allows customers to get a discount.

Conditions
Goods totaling more than \$20
Goods totaling more than \$100
Has discount card
Actions
No discount
5% discount
10% discount

Step 2: Add columns

Calculate how many columns are needed in this table. The number of columns depends on the number of conditions and the number of alternatives for each condition. If there are two conditions and each condition can be either true or false, you need four columns. If there are three conditions there will be eight columns, and so on. Mathematically, the number of columns is $2^{\text{conditions}}$. In the example above, $2^3 = 8$ columns.

No of conditions	No of columns
1	1
2	4
3	8
4	16
5	32

C o n d i t i o n s	Goods totaling more than \$20	Y	Y	Y	Y	N	N	N	N
	Goods totaling more than \$100	Y	Y	N	N	Y	Y	N	N
	Has discount card	Y	N	Y	N	Y	N	Y	N
A c t i o n s	No discount								
	5% discount								
	10% discount								

Step 3: Determine the actions

Enter actions for each column in the table. You will be able to find this information in the requirement.

		Rules							
C o n d i t i o n s	Goods totaling more than \$20	Y	Y	Y	Y	N	N	N	N
	Goods totaling more than \$100	Y	Y	N	N	Y	Y	N	N
	Has discount card	Y	N	Y	N	Y	N	Y	N
A c t i o n s	No discount				X	X	X	X	X
	5% discount		X	X					
	10% discount	X							

Step 4: Reduce the table

Simplify the solution by removing redundancies in the decision table. Mark the insignificant values with values “-”.

		Rules

C o n d i t i o n s	Goods totaling more than \$20	Y	Y	Y	Y	N	N	N	N
	Goods totaling more than \$100	Y	Y	N	N	Y	Y	N	N
	Has discount card	Y	N	Y	N	Y	N	Y	N
A c t i o n s	No discount				X	X	X	X	X
	5% discount		X	X					
	10% discount	X							

		Rules				
C o n d i t i o n s	Goods totaling more than \$20	Y	Y	Y	Y	N
	Goods totaling more than \$100	Y	Y	N	N	-
	Has discount card	Y	N	Y	N	-
A c t i o n s	No discount				X	X
	5% discount		X	X		
	10% discount	X				

Note that as long as the goods do not total up to more than \$20, there will be no discount despite having a discount card. Hence, the only determining condition for having no discount is when the goods do not total up to \$20. We can remove the redundancies by using dashes for the other conditions.

Chapter 4: Data Structures

Contents

- 1 Lists
 - 1.1 Create
 - 1.2 Add list items
 - 1.2.1 Append
 - 1.2.2 Insert
 - 1.2.3 Extend
 - 1.2.4 Using operator
 - 1.3 Read
 - 1.4 Update
 - 1.5 Delete
 - 1.5.1 Remove item by value
 - 1.5.2 Remove item by index
 - 1.5.3 Del keyword
 - 1.5.4 Clear a list
 - 1.6 Working with List
 - 1.6.1 Min, Max and Sum
 - 1.6.2 Reversing
 - 1.6.3 Sorting
 - 1.6.4 Membership and searching
 - 1.7 Copying List
 - 1.7.1 By Slicing
 - 1.7.2 By Constructor
 - 1.7.3 The copy() method
 - 1.8 List Comprehension
- 2 Tuples
 - 2.1 Unpacking Tuples
- 3 Dictionaries
 - 3.1 Create
 - 3.2 Access
 - 3.3 Update
 - 3.4 Add
 - 3.5 Delete

Syllabus Learning Outcomes

- 2.2 Programming Elements and Constructs
 - Use programming language elements and constructs to write recursive and non-recursive programs to solve a variety of problems.
 - 2.2.1 Understand the different types: integer, real, char, string and Boolean and initialise arrays (1-dimensional and 2-dimensional).

1 Lists

Lists are the most commonly used data structure in Python. Lists are used to store multiple items in a single variable. List items are ordered, mutable and allow duplicate values.

A list collection is mutable means its items can be added, updated and removed. Each of these data can be accessed by calling its index value.

1.1 Create

Lists are created using square brackets. There are a few ways to initialise a list.

```

1  # initialise a list
2
3  # creating an empty list using square brackets
4  nums = []
5  print(nums)
6
7  # creating an empty list using list constructor
8  nums = list()
9  print(nums)
10
11 # create a list with elements
12 nums = [1, 2, 3, 4, 5]
13 print(nums)

```

```

[]
[]
[1, 2, 3, 4, 5]

```

List items can be of any data type. A list can also contain different data types.

```

1  # list can be of any data types
2  nums = list(range(10))
3  print(nums)
4
5  characters = list("spiderman")
6  print(characters)
7
8  fruits = ["apple", "banana", "orange"]
9  print(fruits)
10
11 # list can also be of mixed data types
12 list1 = ["abc", 34, True, 40, "male"]
13 print(list1)

```

```

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
['s', 'p', 'i', 'd', 'e', 'r', 'm', 'a', 'n']
['apple', 'banana', 'orange']
['abc', 34, True, 40, 'male']

```

A list can also contain sublists, which is known as a nested list.

```

1  # Example of nested list
2
3  list2 = [[1,"apple"],[2,"orange"],[3,"pear"]]
4
5  print(list2[0])
6  print(list2[0][0])
7  print(list2[0][1])
8

```

```
[1, 'apple']
```

```
1
```

```
apple
```

1.2 Add list items

1.2.1 Append

The `append()` method is used to append an element at the end of the list.

```

1  # append adds an item to the end of the list
2  num = [1, 2, 3, 4, 5]
3  num.append(6)
4  print(num)
5
6
7  num = [1, 2, 3, 4, 5]
8  num.append([6,7,8])
9  print(num)

```

```
[1, 2, 3, 4, 5, 6]
```

```
[1, 2, 3, 4, 5, [6, 7, 8]]
```

1.2.2 Insert

The `insert()` method inserts an item at the specified index.

```

1  # insert
2  fruits = ["apple", "banana", "orange"]
3  fruits.insert(2, "cherry")
4  print(fruits)

```

```
['apple', 'banana', 'cherry', 'orange']
```

1.2.3 Extend

A list can also be extended with items from another list using `extend()` method. It will modify the first list. The resultant list will contain all the elements of the lists that were added, i.e. the resultant list is NOT a nested list.

```

1 # extending the list
2 num = [1, 2, 3, 4, 5]
3 num.extend([6, 7, 8])
4 print(num)

```

[1, 2, 3, 4, 5, 6, 7, 8]

1.2.4 Using operator

Two lists can also be join together simply using + operator.

```

1 # adding 2 lists together
2 num = [1,2,3,4,5]
3 num = num + [6,7,8]
4 print(num)

```

[1, 2, 3, 4, 5, 6, 7, 8]

Similar to String, we can repeat a list multiple times with * operator.

```

1 num = [1, 2, 3]
2 num = num*5
3 print(num)

```

[1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]

1.3 Access list items

List items are indexed and you can access them by referring to the index number. The first item has index 0.

```

1 fruits = ['apple', 'banana', 'cherry', 'durian']
2 print(fruits[0]) #print element at index 0
3 print(fruits[1]) #print element at index 1

```

apple
banana

We can also access the elements in the list using negative indexing. That is the last element has an index of -1, and second last element has index of -2 and so on.

```

1 fruits = ['apple', 'banana', 'cherry', 'durian']
2 print(fruits[-1]) #print element at index 0
3 print(fruits[-2]) #print element at index 1

```

durian
cherry

To iterate through a list, we can use for loop.

```

1 fruits = ['apple', 'banana', 'cherry', 'durian']
2
3 for item in fruits:
4     print(item)

```

```

apple
banana
cherry
durian

```

Similar to Strings, we can use the `len()` method to get the number of elements in a given list.

```

1 fruits = ['apple', 'banana', 'cherry', 'durian']
2 print(len(fruits))

```

```

4

```

Indexing was only limited to accessing a single element. Slicing on the other hand is accessing a sequence of data inside the list.

Slicing is done by defining the index values of the first element and the last element from the parent list that is required in the sliced list, similar to Strings.

```

sub = num[a : b]
sub = num[a : ]
sub = num[: b]
sub = num[:]

```

if both `a` and `b` are specified, `a` is the first index, `b - 1` is the last index. Note that the upper bound index is NOT inclusive.

if `b` is omitted, it will slice till last element.

if `a` is omitted, it will start from first element.

if neither `a` or `b` is specified, it is effectively copy the whole list

```

1 # Example of slicing
2 fruits = ['apple', 'banana', 'cherry', 'durian']
3 print(fruits[1:3]) # prints elements from index 1 to 2
4 print(fruits[:3]) # prints elements from index 0 to 2
5 print(fruits[1:len(fruits)]) # print elements from index 1 to last element
6 print(fruits[1:]) # print elements from index 1 to last element
7 print(fruits[1:-1]) # print elements from index 1 to second last element

```

```

['banana', 'cherry']
['apple', 'banana', 'cherry']
['banana', 'cherry', 'durian']
['banana', 'cherry', 'durian']
['banana', 'cherry']

```


The `index()` method returns the position at the first occurrence of the specified value. If there are multiple items of the same value, only the first index value of that item is returned. You can add 2nd argument to start searching from a particular index onwards.

```
1 # example of using index() method
2 fruits = ['apple', 'banana', 'cherry', 'durian', 'banana']
3 print(fruits.index("cherry"))
4 print(fruits.index("banana"))
5 print(fruits.index("banana",2))
6
```

```
2
1
4
```

1.4 Update

An item in the list can be updated by its index value.

```
1 fruits = ['apple', 'banana', 'cherry', 'durian']
2 fruits[1] = 'blackcurrent'
3 print(fruits)
```

```
['apple', 'blackcurrent', 'cherry', 'durian']
```

You can update more than one item by specifying a range of index numbers where you want to insert the new items.

```
1 fruits = ['apple', 'banana', 'cherry', 'durian']
2 fruits[1:3] = ['blackcurrent', 'watermelon']
3 print(fruits)
```

```
['apple', 'blackcurrent', 'watermelon', 'durian']
```

If you insert more items than you replace/update, the new items will be inserted where you specified, and the remaining items will move accordingly.

```
1 fruits = ['apple', 'banana', 'cherry', 'durian']
2 fruits[1:3] = ['blackcurrent', 'watermelon', 'strawberry']
3 print(fruits)
```

```
['apple', 'blackcurrent', 'watermelon', 'strawberry', 'durian']
```

1.5 Delete

1.5.1 Remove item by value

The `remove()` method remove an item based on its value. If there are multiple items of same value, it will only remove 1st item.

It will throw an exception if the value is not found in the list.

```
1 num = [10,20,30]
2 num.remove(20)
3 print(num)
4 num.remove(40)
```

[10, 30]

```
-----
ValueError                                Traceback (most recent call last)
<ipython-input-33-caf5ef45a1ad> in <module>()
      2 num.remove(20)
      3 print(num)
----> 4 num.remove(40)

ValueError: list.remove(x): x not in list
```

1.5.2 Remove item by index

The `pop()` method removes the element at the specified position. If no position is specified, last item is removed by default. The `pop()` method returns removed value.

```
1 num = [1, 2, 3, 4, 5]
2 num.pop() #last item is removed
3 print(num)
4
5 num = [1, 2, 3, 4, 5]
6 x = num.pop(1) # remove and return item at index 1
7 print(num)
8 print(x)
```

[1, 2, 3, 4]

[1, 3, 4, 5]

2

1.5.3 Del keyword

The **del** keyword is used to delete objects. In Python, everything is an object, so the **del** keyword can also be used to delete variables, lists, or parts of a list etc.

```
1 num = [1, 2, 3, 4, 5]
2
3 del num[2] #del element at index 2
4 print(num)
5 del num
6 print(num)
```

[1, 2, 4, 5]

```
-----
NameError                                Traceback (most recent call last)
<ipython-input-36-fc63cb7c0723> in <module>()
      4 print(num)
      5 del num
----> 6 print(num)

NameError: name 'num' is not defined
```

del num deleted the num list entirely, hence an error is encountered when the `print(num)` command is executed.

1.5.4 Clear a list

To clear all elements in a list, use its `clear()` method.

```
1 num = [1,2,3]
2 print(num)
3 num.clear()
4 print(num)
```

[1, 2, 3]
[]

1.6 Working with List

1.6.1 Min, Max and Sum

If the list consists of all integer elements, the `min()`, `max()` and `sum()` gives the minimum item, maximum item and total sum value of the list.

```
1 num = [0,1,2,3,4,5,6,7,8,9]
2 print(min(num))
3 print(max(num))
4 print(sum(num))
```

0
9
45

For a list with elements as string, the `max()` and `min()` is still applicable. The `max()` method would return a string element whose ASCII value is the highest. The `min()` method is used to return the lowest. Take note that only the first index of each element is considered each time and if they value is the same then second index considered so on and so forth.

```
1 poly = ['np', 'sp', 'tp', 'rp', 'nyp']
2 print(min(poly))
3 print(max(poly))
```

np
tp

1.6.2 Reversing

The entire elements present in the list can be reversed by using the `reverse()` method.

```
1 num = [1, 2, 3, 4, 5, 6, 7, 8, 9]
2 num.reverse()
3 print(num)
```

[9, 8, 7, 6, 5, 4, 3, 2, 1]

1.6.3 Sorting

List has a built in method `sort()` to arrange the elements in ascending order by default. For descending order, specify the named argument `reverse = True`.

```
1 num = [24, 9, 65, 43, 13]
2 num.sort() # numbers are sorted in ascending order by default
3 print(num)
4 num.sort(reverse = True) # adding reverse = True will sort list in descending order
5 print(num)
```

[9, 13, 24, 43, 65]
[65, 43, 24, 13, 9]

Note that both the `sort()` and `reverse()` methods mentioned above will alter the original sequence of items in the list.

The `sorted()` method sorts the given sequence either in ascending order or in descending order and returns the sorted list. This method does not effect the original sequence of items in the list.

```

1 num = [24, 9, 65, 43, 13]
2 x = sorted(num)
3 y = sorted(num, reverse = True)
4 print(x)
5 print(y)
6 print(num) #the original list is not sorted

```

[9, 13, 24, 43, 65]

[65, 43, 24, 13, 9]

[24, 9, 65, 43, 13]

1.6.4 Membership and searching

You can use the in (not in) statement for membership of an item in a list.

```

1 names = ['duck', 'chicken', 'goose']
2 print('duck' in names)
3 print('turkey' not in names)

```

True

True

The count() method can count the occurrence of a particular item in a list.

```

1 num = [1, 2, 3, 4, 5, 5, 5, 8, 9]
2 print(num.count(5))

```

3

1.7 Copying List

You cannot copy a list simply by typing list2 = list1, because list2 will only be a reference to list1, and changes made in list1 will automatically also be made in list2.

```

1 list1= [0,1,2,3]
2 list2 = list1
3 print(list1 == list2)
4 print(list1 is list2)
5
6 # Modify a list, both lists are affected
7 list1.pop()
8 print(list1)
9 print(list2)

```

True

True

[0, 1, 2]

[0, 1, 2]

1.7.1 By Slicing

There are ways to make a copy, one way is to copy by slicing. Slicing without start and end index returns all items in the list.

```

1 list1= [0, 1, 2, 3]
2 list2 = list1[:]
3 print(list1 == list2)
4 print(list1 is list2)
5
6 # Modify a list, the other is not affected
7 list1.pop()
8 print(list1)
9 print(list2)

```

```

True
False
[0, 1, 2]
[0, 1, 2, 3]

```

1.7.2 By Constructor

list() is a constructor function to create a list. When it accepts another list as argument, it creates a duplicate list.

```

1 list1= [0, 1, 2, 3]
2 list2 = list(list1)
3 print(list1 == list2)
4 print(list1 is list2)
5
6 # Modify a list, the other is not affected
7 list1.pop()
8 print(list1)
9 print(list2)

```

```

True
False
[0, 1, 2]
[0, 1, 2, 3]

```

1.7.3 The copy() method

The copy() method copies a list into another list.

```

1 list1= [0, 1, 2, 3]
2 list2 = list1.copy()
3 print(list1 == list2)
4 print(list1 is list2)
5
6 # Modify a list, the other is not affected
7 list1[0] = 10
8 list2[0] = 20
9 print(list1)
10 print(list2)

```

True

False

[10, 1, 2, 3]

[20, 1, 2, 3]

list2 is stored at different memory location, which is confirmed by values of id() functions. The id() function returns the address of the list. Modifying list2 doesn't affect the values in list1.

```

1 print(id(list1), id(list2))

```

2369564014280 2369564063688

Do note that the copy() method only performs shallow copy. It copies the list itself, which contains references to the objects in the list. If an object itself is mutable, change in this object will be reflected in both lists. An example would be nested list.

```

1 # copying a nested list using copy()
2 list1 = [1,[2,3,4],5]
3 list2 = list1.copy()
4 print("List1: ",list1)
5 print("List2: ",list2)
6 print()
7
8 list2[1][1] = 9
9 print("List1: ",list1)
10 print("List2: ",list2)
11 print()
12
13 list2.append(6)
14 print("List1: ",list1)
15 print("List2: ",list2)

```

List1: [1, [2, 3, 4], 5]

List2: [1, [2, 3, 4], 5]

List1: [1, [2, 9, 4], 5]

List2: [1, [2, 9, 4], 5]

List1: [1, [2, 9, 4], 5]

List2: [1, [2, 9, 4], 5, 6]

For situations like the above, the copy module provides a function `deepcopy()` to perform deep copy of objects, such that it will also create a duplicate for the nested objects.

```

1  from copy import deepcopy
2
3  list1 = [1,[2,3,4],5]
4  list2 = deepcopy(list1)
5  print("List1: ",list1)
6  print("List2: ",list2)
7  print()
8
9  list2[1][1] = 9 # modifying list2
10 print("List1: ",list1)
11 print("List2: ",list2)
12 print()
13
14 list2.append(6) # modifying list2
15 print("List1: ",list1)
16 print("List2: ",list2)
17 print()

```

```
List1:  [1, [2, 3, 4], 5]
List2:  [1, [2, 3, 4], 5]
```

```
List1:  [1, [2, 3, 4], 5]
List2:  [1, [2, 9, 4], 5]
```

```
List1:  [1, [2, 3, 4], 5]
List2:  [1, [2, 9, 4], 5, 6]
```

1.8 List Comprehension

List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list. The syntax is as follows:

```
newlist = [expression for item in iterable if condition == True]
```

We can use either a for or while loop to create a list of numbers from 0 to 9.

```

1  # Creating a list of numbers from 0 to 9
2
3  nums = []
4  for num in range(10):
5      nums.append(num)
6
7  print(nums)

```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```


With list comprehension, we can create the same list of numbers 0 to 9 in one simple statement.

```

1  # Example of using basic list comprehension
2  # to create a list of numbers from 0 to 9
3
4  nums = [x for x in range(10)]
5  print(nums)

```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

We can create a list of even numbers by adding in a condition. The condition is like a filter that only accepts the items that evaluate to True.

```

1  # Adding a condition
2  # to create a list of even numbers from 0 to 9
3
4  nums = [x for x in range(10) if x%2 == 0]
5  print(nums)

```

```
[0, 2, 4, 6, 8]
```

The expression is the outcome, which you can manipulate it before you append it into the list.

```

1  # double the even values before adding into the list
2
3  nums = [x*2 for x in range(10) if x%2 == 0]
4  print(nums)

```

```
[0, 4, 8, 12, 16]
```

The expression can also contain conditions, not like a filter, but as a way to manipulate the outcome.

```

1  # create a list of numbers such that
2  # the number is doubled when the number is even
3  # the number is tripled when the number is odd
4  # where the number is from 0 to 9
5
6  nums = [x*2 if x%2 == 0 else x*3 for x in range(10) ]
7  print(nums)

```

```
[0, 3, 4, 9, 8, 15, 12, 21, 16, 27]
```

2 Tuples

Tuple behaves like a list but it is **immutable**. Tuples are written with round brackets.

```

1  # initialise a tuple
2
3  # creating an empty tuple using round brackets
4  nums = ()
5  print(nums)
6
7  # creating an empty list using tuple constructor
8  nums = tuple()
9  print(nums)
10
11 # create a tuple with elements
12 nums = (1, 2, 3, 4, 5)
13 fruits = ("apples", "bananas", "oranges")
14 print(nums)
15 print(type(nums))
16 print(fruits)

```

```

()
()
(1, 2, 3, 4, 5)
<class 'tuple'>
('apples', 'bananas', 'oranges')

```

To create a tuple containing a single item, you have to include a comma; else, Python will not recognise it as a tuple.

```

1  # creating a tuple with only 1 item
2
3  nums = (1)
4  print(type(nums))
5
6  nums = (1,)
7  print(type(nums))

```

```

<class 'int'>
<class 'tuple'>

```

Tuples are immutable, meaning that we cannot change, add or remove items after the tuple has been created.

```

1  nums = (1, 2, 3, 4, 5)
2  nums[0] = 100

```

```

-----
TypeError                                Traceback (most recent call last)
<ipython-input-8-992d5c4524b8> in <module>()
      1 nums = (1, 2, 3, 4, 5)
----> 2 nums[0] = 100

TypeError: 'tuple' object does not support item assignment

```

```

1 nums = (1, 2, 3, 4, 5)
2
3 del nums[0]

```

TypeError Traceback (most recent call last)
 <ipython-input-9-aa31e1f960cf> in <module>()
 1 nums = (1, 2, 3, 4, 5)
 2
 ----> 3 del nums[0]
TypeError: 'tuple' object doesn't support item deletion

We can create a list from a tuple using type casting and vice versa.

```

1 nums1 = (1, 2, 3, 4, 5)
2 print(type(nums1))
3
4 nums2 = list(nums1)
5 print(nums2)
6 print(type(nums2))
7
8 nums3 = tuple(nums2)
9 print(type(nums3))

```

```

<class 'tuple'>
[1, 2, 3, 4, 5]
<class 'list'>
<class 'tuple'>

```

Tuples respond to all of the general sequence operations we used on strings or lists in the prior chapter.

Python Expression	Results	Description
len((1, 2, 3))	3	Length
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)	Concatenation
('Hi!') * 4	('Hi!', 'Hi!', 'Hi!', 'Hi!')	Repetition
3 in (1, 2, 3)	True	Membership
for x in (1, 2, 3): print x,	1 2 3	Iteration

2.1 Unpacking Tuples

When we create a tuple, we normally assign values to it. This is called "packing" a tuple. In Python, we are allowed to extract the values back into variables. This is called "unpacking" a tuple.

```
1 # unpacking one-dimensional tuple
2 nums = (1, 2, 3)
3 a,b,c = nums
4 print(a)
5 print(b)
6 print(c)
```

1
2
3

```
1 # unpacking nested tuples
2 nums = (1, 2, (3,4))
3 a,b,c = nums
4 print(a)
5 print(b)
6 print(c)
7 print()
8
9 nums = (1, 2, (3,4))
10 a,b,(c,d) = nums
11 print(a)
12 print(b)
13 print(c)
14 print(d)
15 print()
```

1
2
(3, 4)

1
2
3
4

3 Dictionaries

3.1 Create

Dictionaries are used to store items in key:value pairs. Each key is separated from its value by a colon (:), the items are separated by commas, and the whole thing is enclosed in curly braces. An empty dictionary without any items is written with just two curly braces.

```

1 # empty dictionary
2 dict1 = {}
3 print(dict1)
4
5 dict2 = dict()
6 print(dict2)
7
8 # dictionary with mixed data type
9 dict3 = {'name': 'John', 'age': 7, 'family': ['Father', 'Mother']}
10 print(dict3)

```

```

{}
{}
{'name': 'John', 'age': 7, 'family': ['Father', 'Mother']}

```

Dictionary items are unordered (in Python 3.6 and earlier), changeable, and does not allow duplicates. Keys are unique within a dictionary while values may not be. The values of a dictionary can be of any type, but the keys must be of an immutable data type such as strings, numbers, or tuples. Duplicate values will overwrite existing values

```

1 # dictionary does not allow duplicates
2 dict3 = {'name': 'John', 'age': 7, 'family': ['Father', 'Mother'], 'family': ['Brother']}
3 print(dict3)

```

```

{'name': 'John', 'age': 7, 'family': ['Brother']}

```

3.2 Access

Dictionary items can be accessed by using the corresponding keys.

```

1 # accessing values using corresponding key
2 dict3 = {'name': 'John', 'age': 7, 'family': ['Father', 'Mother']}
3 print("Name: ", dict3['name'])
4 print("Age: ", dict3['age'])

```

```

Name: John
Age: 7

```

If we attempt to access a data item with a key, which is not part of the dictionary, we get an error.

```

1 # accessing values using corresponding key
2 dict3 = {'name': 'John', 'age': 7, 'family': ['Father', 'Mother']}
3 print("Name: ", dict3['name'])
4 print("Age: ", dict3['age'])
5 print("Address", dict3['address'])

```

Name: John

Age: 7

```

-----
KeyError                                Traceback (most recent call last)
<ipython-input-6-238ae73aac70> in <module>()
      3 print("Name: ", dict3['name'])
      4 print("Age: ", dict3['age'])
----> 5 print("Address", dict3['address'])

```

KeyError: 'address'

We can use the method `get()` to access the values with the corresponding keys. This method is more forgiving as it does not throw an error if key does not exist. However, it returns a `None` object.

```

1 # accessing values using get()
2 dict3 = {'name': 'John', 'age': 7, 'family': ['Father', 'Mother']}
3 print("Name: ", dict3.get('name'))
4 print("Age: ", dict3.get('age'))
5 print("Address", dict3.get('address'))

```

Name: John

Age: 7

Address None

The `keys()` method will return a list of all the keys in the dictionary. The `values()` method will return a list of all the values in the dictionary. The `items()` method will return each item in a dictionary, as tuples in a list.

```

1 # keys() and values()
2 dict3 = {'name': 'John', 'age': 7, 'family': ['Father', 'Mother']}
3 print(dict3.keys())
4 print(dict3.values())
5 print(dict3.items())

```

`dict_keys(['name', 'age', 'family'])`

`dict_values(['John', 7, ['Father', 'Mother']])`

`dict_items([('name', 'John'), ('age', 7), ('family', ['Father', 'Mother'])])`

To determine if a specified key is present in a dictionary, we can use the `in` keyword.

```

1 dict3 = {'name': 'John', 'age': 7, 'family': ['Father', 'Mother']}
2 print('name' in dict3)
3 print('address' in dict3)

```

True

False

We can also use `in` to access all keys in the dictionary and the corresponding values.

```
1 dict3 = {'name': 'John', 'age': 7, 'family': ['Father', 'Mother']}
2
3 for k in dict3:
4     print(k, dict3[k])
```

```
name John
age 7
family ['Father', 'Mother']
```

We can also print all key:value pair by using `items()`.

```
1 dict3 = {'name': 'John', 'age': 7, 'family': ['Father', 'Mother']}
2
3 for k,v in dict3.items():
4     print(k, v)
```

```
name John
age 7
family ['Father', 'Mother']
```

3.3 Update

We can change the value of a specific item by referring to its key.

```
1 # update value by using its key
2 food = {'a': 'Apple', 'b': 'Berries',
3         'c': 'Cereal', 'd': 'Doughnut'}
4
5 food['b'] = 'Banana'
6 print(food)
```

```
{'a': 'Apple', 'b': 'Banana', 'c': 'Cereal', 'd': 'Doughnut'}
```

We can also change the values of a specific item by using the `update()` method.

```
1 # update using update() method
2 food = {'a': 'Apple', 'b': 'Berries',
3         'c': 'Cereal', 'd': 'Doughnut'}
4
5 food.update({'c': 'Carrot'})
6 print(food)
```

```
{'a': 'Apple', 'b': 'Berries', 'c': 'Carrot', 'd': 'Doughnut'}
```

3.4 Add

Adding an item to the dictionary can be done by using a new index key and assigning a value to it.

```
1 # add a new food
2 food = {'a': 'Apple', 'b': 'Berries',
3         'c': 'Cereal', 'd': 'Doughnut'}
4
5 food['e'] = 'Eggplant'
6 print(food)
```

```
{'a': 'Apple', 'b': 'Berries', 'c': 'Cereal', 'd': 'Doughnut', 'e': 'Eggplant'}
```

The update() method can also be used to add new items into dictionaries.

```
1 # add new items using update()
2 food = {'a': 'Apple', 'b': 'Berries',
3         'c': 'Cereal', 'd': 'Doughnut'}
4
5 food.update({'e': 'Eggplant', 'f': 'Fish'})
6 print(food)
```

```
{'a': 'Apple', 'b': 'Berries', 'c': 'Cereal', 'd': 'Doughnut', 'e': 'Eggplant', 'f': 'Fish'}
```

3.5 Delete

An item can be deleted by using the pop() method. The method removes an item by its key and returns the value. It throws an exception if key is not found.

```
1 # delete item by using pop()
2 food = {'a': 'Apple', 'b': 'Berries',
3         'c': 'Cereal', 'd': 'Doughnut'}
4 food.pop('c')
5 print(food)
6 food.pop('e')
```

```
{'a': 'Apple', 'b': 'Berries', 'd': 'Doughnut'}
```

```
-----
KeyError                                Traceback (most recent call last)
<ipython-input-29-0bea29e7add1> in <module>()
      4 food.pop('c')
      5 print(food)
----> 6 food.pop('e')

KeyError: 'e'
```

The del keyword removes the item with the specified key name.

```
1 # delete item by using del
2 food = {'a': 'Apple', 'b': 'Berries',
3         'c': 'Cereal', 'd': 'Doughnut'}
4 del food['d']
5 print(food)
```

```
{'a': 'Apple', 'b': 'Berries', 'c': 'Cereal'}
```


The del keyword can also delete the dictionary completely.

```
1 # delete dictionary by using del
2 food = {'a': 'Apple', 'b': 'Berries',
3         'c': 'Cereal', 'd': 'Doughnut'}
4 del food
5 print(food)
```

```
-----
NameError                                Traceback (most recent call last)
<ipython-input-31-2fc0c25beec7> in <module>()
      3         'c': 'Cereal', 'd': 'Doughnut'}
      4 del food
----> 5 print(food)
```

NameError: name 'food' is not defined

The clear() method empties the dictionary.

```
1 # empty dictionary using clear()
2 food = {'a': 'Apple', 'b': 'Berries',
3         'c': 'Cereal', 'd': 'Doughnut'}
4 food.clear()
5 print(food)
```

```
{}
```

Chapter 5: Functions

Contents

1. Introduction to Subroutines
 - 1.1. Advantages of Subroutines
 - 1.2. Basic Syntax
 - 1.3. Examples of Functions
 - 1.4. Docstrings
2. Defining Subroutines: Functions and Procedures
 - 2.1. A procedure that does nothing
 - 2.2. A procedure without both input parameter and return statement
 - 2.3. A function with input parameters
 - 2.4. A function with both input parameter and return statement
 - 2.5. A function with default arguments
 - 2.6. A function with a variable number of arguments
3. Variable Scope: Global vs Local
 - 3.1. Local Variables
 - 3.2. Global Variables
4. Referenced Readings

Syllabus Learning Outcomes

- 2.2 Programming Elements and Constructs
 - 2.2.4 Use functions and procedures to modularise problems into chunks of code.
-

1. Introduction to Subroutines

Subroutines allow us to manage and structure our programs. This allows us to break up programs into smaller chunks of codes. This is called modular design. Such modular design allows you to debug easily, especially when you are dealing with a large program.

There are two types of subroutines.

- **Functions:** Subroutines that allow us to have both input and output. Functions accept inputs and process them into output. They must return at least one value, aka “return value”.
- **Procedures:** Subroutines that do not return values.

1.1. Advantages of Subroutines

The following are some of the advantages of using subroutines.

- **Organization:** Subroutines help break programs into smaller and modular chunks. They are almost like a mini-program that we can write separately from the main program, without having to think about the rest of the program while we write it. This allows us to reduce a complicated program into smaller, more manageable chunks, which reduces the overall complexity of our program.
- **Abstraction:** We can use a subroutine at any time, using its unique name and appropriate inputs.
- **Reusability:** It supports code reusability and reduces repetitive code. Subroutines can also be shared with other programs, reducing the need to code from scratch.
- **Modifying:** When we need to extend our program to handle a case it didn't handle before, subroutines allow us to make the change in one place and have that change take effect every time the subroutine is called.
- **Testing:** The program is easier to test and debug since each subroutine is self-contained. Furthermore, once a subroutine is working properly, there is no need to test the subroutine again unless it has been modified

1.2. Basic Syntax

```
def function_name(parameters):
    """docstring"""
    statement(s)
```

- Keyword def marks the start of a function.
- Every function has a name as uniquely identification.
- It may take in optional values, known as parameters, before running the code block.
- It may return value, as a result, using a return statement.
 - If no return statement, the function will return None at the end of the function.
- Optional documentation string (docstring) describes what the function does. It may be multiple lines of string.
- All statements must be equally identified, which is usually 4 spaces.

Note: Do not indent your program with a mix of spaces and tabs.

1.3. Examples of Functions

We have used a few functions when we learned about Python basic data types. Can you recall the following functions?

- How do you check data type of an object?
- How do you type cast a value from one data type to another?
- How do you print out the value(s) or output of your programs?

1.4. Docstrings

Recall what we get when we run `help(len)` or `?len`. Docstring is actually the documentation that gets returned when we run the help function.

```
[ ] help(len)

Help on built-in function len in module builtins:

len(obj, /)
    Return the number of items in a container.

[ ] def test():
    """
    Hi I'm the docstring.
    I am supposed to tell you more about this function.
    """

    pass

help(test)
```

2. Defining Subroutines: Functions and Procedures

Let's get familiar with how to define the following procedures and functions.

- A procedure that does nothing
- A procedure without both input parameter and return statement
- A function with input parameters
- A function with both input parameter and return statement
- A function with default arguments
- A function with a variable number of arguments

2.1. A procedure that does nothing

Let's define the simplest function which does nothing.

- Keyword `pass` is used to indicate nothing to be done in the function body.
- Usually used when we want to define a function to do something, but we are not ready to code it yet.

```
[3] def does_nothing():
    pass

[4] type(does_nothing())

NoneType
```

2.2. A procedure without both input parameter and return statement

Let's define a basic `hello_world` function without input parameter and return statement.

- It simply prints "Hello World".
- This is an example of a procedure.

```
[5] def hello_world():
    print("Hello World")

    print("Have a good day")
    hello_world()

Have a good day
Hello World

# more about procedure
# when you assign the procedure to a variable, the function is run once. However, what is the value assigned to x?
x = hello_world()

print(x)

Hello World
None
```

2.3. A function with input parameters

Let's use an input parameter to make the hello function more flexible.

- It takes in an input value `who`, and prints "Hello {who}".

```

def hello_who(who):
    print("Hello,", who)

hello_who("Singapore")

```

Hello, Singapore

2.4. A function with both input parameter and return statement

Let's define a function `my_max` which returns the greater value of 2 input values.

```

def my_max(x, y):
    if x > y:
        return x
    else:
        return y

print(my_max(10, 20))

def my_max2(x, y):
    return x if x > y else y

print(my_max2(23, 18))

```

20
23

The return statement will exit the function. Consider the following function.

```

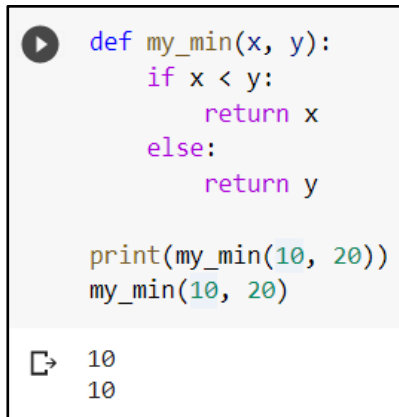
def my_max3(x, y):
    return x if x > y else y
    print("We should see this line if we did not exit the function")

my_max3(103, 320)

```

320

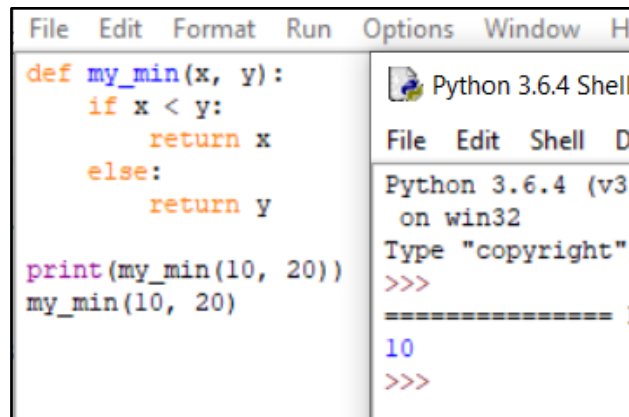
Let's define a function `my_min` which returns the smaller value of 2 input values.



```
def my_min(x, y):
    if x < y:
        return x
    else:
        return y

print(my_min(10, 20))
my_min(10, 20)
```

10
10



```
def my_min(x, y):
    if x < y:
        return x
    else:
        return y

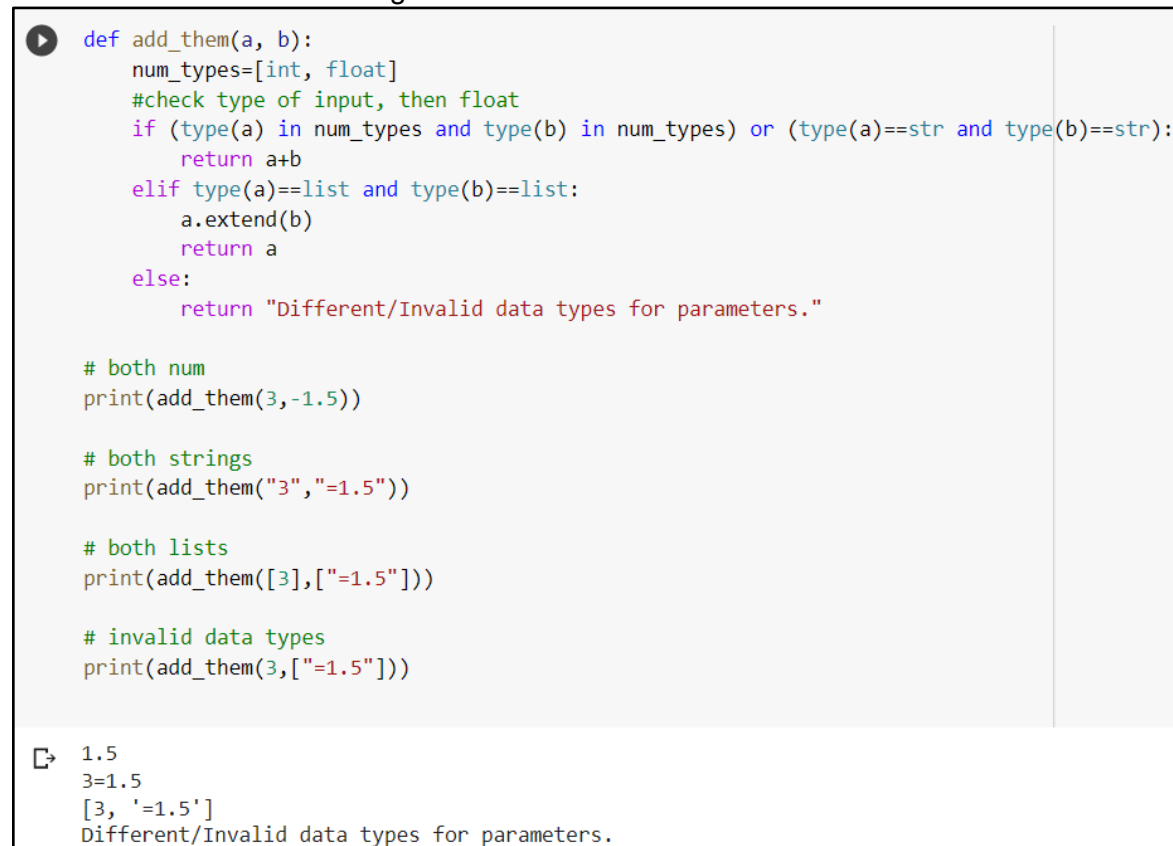
print(my_min(10, 20))
my_min(10, 20)
```

Python 3.6.4 Shell
File Edit Shell D
Python 3.6.4 (v3.
on win32
Type "copyright",
>>>
=====
10
>>>

Note the difference between invoking the function in Python Notebook (left image) and Python IDLE (right image). For the latter, you need to call the print function to display the output of the function.

Let us create a function `add_them()` that will take two input parameters `a` and `b` and:

- return the sum of numbers if the input parameters are integers or floats,
- return a joint string if the input parameters are strings,
- return a joint list if the input parameters are lists,
- return an error message if otherwise



```
def add_them(a, b):
    num_types=[int, float]
    #check type of input, then float
    if (type(a) in num_types and type(b) in num_types) or (type(a)==str and type(b)==str):
        return a+b
    elif type(a)==list and type(b)==list:
        a.extend(b)
        return a
    else:
        return "Different/Invalid data types for parameters."

# both num
print(add_them(3,-1.5))

# both strings
print(add_them("3","=1.5"))

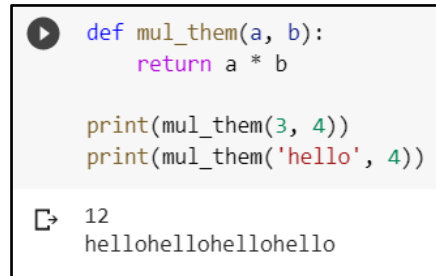
# both lists
print(add_them([3],["=1.5"]))

# invalid data types
print(add_them(3,["=1.5"]))
```

1.5
3=1.5
[3, '=1.5']
Different/Invalid data types for parameters.

Now, consider a function `mul_them(a, b)` that will take 2 input parameters `a` and integer `b` and:

- return the product if the input parameter `a` is an integer or float
- return the string `a` for `b` times if `a` is a string



```
def mul_them(a, b):  
    return a * b  
  
print(mul_them(3, 4))  
print(mul_them('hello', 4))
```

12
hellohellohellohello

2.5. A function with default arguments

Python allows function arguments to have default values. If the function is called without the argument, the argument gets its default value.

A parameter is a variable defined in the function definition, while an argument is an actual value passed to the function.


```
[20] def mul_them2(a, b=1):
      return a * b

      print(mul_them2(3, 4))
      print(mul_them2('hello'))

12
hello

[21] #The default arguments must always be placed after the parameters without default values.

def mul_them3(a=1, b):
    return a * b

File "<ipython-input-21-88591bd12d03>", line 3
def mul_them3(a=1, b):
    ^
SyntaxError: non-default argument follows default argument

SEARCH STACK OVERFLOW

[22] # What happens if we have many default arguments and we only want to change the second one?
def mul_them4(a=1, b=3):
    return a * b

# how should we call mul_them 4 if we want b to be 5 instead?
```

2.6. A function with a variable number of arguments

We can pass a variable number of arguments to a function using:

- `*args` (Non-keyword arguments)
- `**kwargs` (Keyword arguments)

We use `*args` and `**kwargs` as an argument when we are unsure about the number of arguments to pass in the functions.

This is not in the A-level syllabus. You may refer to <https://www.programiz.com/python-programming/args-and-kwargs> for a simple overview.

3. Variable Scope: Global vs Local

The scope of a variable determines the portion of the program where you can access a particular variable. There are two basic variable scopes in Python

- **Global variables:** variables defined outside a function body
- **Local variables:** variables defined inside a function body

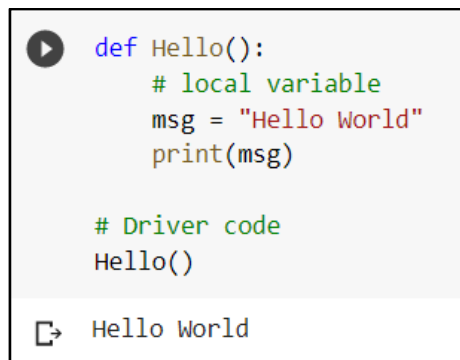
Global and local variables are in different scopes. Local variables can be accessed only inside the function in which they are declared, while global variables can be accessed throughout the program body.

It is good practice to use local variables wherever possible. There are several advantages listed below:

- Avoid changing the value that is being stored somewhere else in the program.
- You could use the same variable in different sections of the code, and each could be treated as a separate variable.
- You free up memory each time we are done with a local variable. It is removed from memory.

3.1. Local Variables

Consider the following function. The local variable `msg` is initialised inside a function and therefore, belongs only to that particular function. It cannot be accessed anywhere outside the function.



```
def Hello():  
    # local variable  
    msg = "Hello World"  
    print(msg)  
  
# Driver code  
Hello()  
  
Hello World
```

Analyse what happens when you try to access the variable `msg` outside the `Hello` function.

```

def Hello():
    # local variable
    msg = "Hello World"
    print(msg)

# Driver code
Hello()
print(msg)

```

 Hello World

```

NameError                                Traceback (most recent call last)
<ipython-input-3-9c367fc89c7b> in <module>()
      6 # Driver code
      7 Hello()
----> 8 print(msg)

NameError: name 'msg' is not defined

```

3.2. Global Variables

Consider the following function. The global variable `msg` is defined outside the `Hello` function and therefore, is accessible throughout the program i.e. inside and outside of the function.

```

def Hello():
    print("Inside Function", msg)

# Global scope
msg = "Hello World"
Hello()
print("Outside Function", msg)

```

 Inside Function Hello World
 Outside Function Hello World

What if there is a variable with the same name initialised inside a function as well as globally? Consider the following code:

```

def Hello():
    msg = "Hello World"
    print(msg)

# Global scope
msg = "Hello World from me too"
Hello()
print(msg)

```

Hello World
 Hello World from me too

So, we can conclude that if a variable with the same name is defined inside the scope of a function as well as globally, the function will use the value given inside the function only.

But, what if we want to change/edit/assign the value of the global variable?

In such cases, we should use the `global` keyword in the function. This `global` keyword is not needed if you just want to print/access the global variable. Consider the following code:

```

msg = "Hello World"

# Uses global because there is no local 'a'
def Hello1():
    print('Inside Hello1() : ', msg)

# Variable 'a' is redefined as a local
def Hello2():
    msg = "Hello Singapore"
    print('Inside Hello2() : ', msg)

# Uses global keyword to modify global 'a'
def Hello3():
    global msg
    msg = "Hello Universe"
    print('Inside Hello3() : ', msg)

# Global scope
print('global : ', msg)
Hello1()
print('global : ', msg)
Hello2()
print('global : ', msg)
Hello3()
print('global : ', msg)

```

```
global : Hello World
Inside Hello1() : Hello World
global : Hello World
Inside Hello2() : Hello Singapore
global : Hello World
Inside Hello3() : Hello Universe
global : Hello Universe
```

As you can see in `Hello3()`, using the `global` keyword will allow you to edit the global variable.

4. Referenced Readings

- What are functions and procedures for? - Functions, procedures and modules - GCSE Computer Science Revision. (2022). BBC Bitesize. <https://www.bbc.co.uk/bitesize/guides/z9hykqt/revision/1>
- Tagliaferri, L. (2021, August 20). An Introduction to String Functions in Python 3. DigitalOcean Community. <https://www.digitalocean.com/community/tutorials/an-introduction-to-string-functions-in-python-3>
- Mester, T. (2021, November 25). Python Built-in Functions and Methods (Python & Data Science - Basics #3). Data36. <https://data36.com/python-built-in-functions-methods-python-data-science-basics-3/>
- Langfield, S., & Duddell, D. (2019). Cambridge International AS and A Level Computer Science Coursebook with Cambridge Elevate Edition (2 Years) (2nd ed.). Cambridge University Press.
- Rouse, G. (2015). OCR a Level Computer Science (UK ed.). Hodder Education.
- Reeves, B. (2015). Aqa a Level Computer Science (UK ed.). Hodder Education.

Chapter 6: Data Representation

Contents

- 1 Number systems
 - 1.1 Denary numbers
 - 1.2 Binary numbers
 - 1.3 Hexadecimal numbers

- 2 Conversion
 - 2.1 Denary to Binary
 - 2.2 Binary to Denary
 - 2.3 Hexadecimal to Binary
 - 2.4 Binary to Hexadecimal
 - 2.5 Denary to Hexadecimal
 - 2.6 Hexadecimal to Denary

Syllabus Learning Outcomes

- 3.1 Data Representation
 - Understand that values can be represented in different number bases: denary, binary and hexadecimal.
 - 3.1.1 Represent data in binary and hexadecimal forms.
 - 3.1.2 Write programs to perform the conversion of positive integers between different number bases: denary, binary and hexadecimal forms; and display results.

1 Number systems

In this chapter, we will learn about three number systems – denary, binary and hexadecimal. Learning about number systems will allow you to understand how computers represent data and how to interpret them.

1.1 Denary numbers

The number system that we are most familiar with and typically used is called the denary number system. It is also known as the decimal number system and is made up of 10 distinct digits – 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. This is therefore a base 10 number system.

Consider a denary number 346. The table below shows the representation of the denary number using place values.

Place value	$10^2 = 100$	$10^1 = 10$	$10^0 = 1$
Digit	3	4	6
Product of digit and place value	$3 \times 10^2 = 300$	$4 \times 10^1 = 40$	$6 \times 10^0 = 6$

This gives $(3 \times 10^2) + (4 \times 10^1) + (6 \times 10^0) = 346$

1.2 Binary numbers

The number system used by computers consists of two unique digits – 0 and 1. It is called the binary number system. A binary digit is referred to as a bit. A group of eight bits is called a byte.

Humans tend to use the denary number system. Hence, denary numbers must be converted into their binary equivalent before a computer can use them.

As with a denary number, the value of a binary number is defined by place values.

Consider the binary number 101110.

Place value	$2^5 = 32$	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
Digit	1	0	1	1	1	0
Product of digit and place value	$1 \times 2^5 = 32$	$0 \times 2^4 = 0$	$1 \times 2^3 = 8$	$1 \times 2^2 = 4$	$1 \times 2^1 = 2$	$0 \times 2^0 = 0$

This gives $(1 \times 2^5) + (0 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (0 \times 2^0) = 46$

1.3 Hexadecimal numbers

There is another number system, called the hexadecimal number system. This number system consists of 16 unique digits – 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F. The symbols A through to F represent the denary values 10 to 15.

Hexadecimal digit	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Denary equivalent	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The value of a number is defined by place values. Consider the hexadecimal number 2A6.

Place value	$16^2 = 256$	$16^1 = 16$	$16^0 = 1$
Digit	2	A	6
Product of digit and place value	$2 \times 16^2 = 512$	$10 \times 16^1 = 160$	$6 \times 16^0 = 6$

Adding up the values in the bottom row shows that the equivalent denary number is 678.

2 Conversion

2.1 Denary to Binary

There are two methods for converting a denary number to binary.

Method 1: division by 2

1. Divide the denary number by 2 using division with remainder and write down the quotient(result) and the remainder
2. Repeat the division until you get a result of 0
3. Write the remainders in reverse (from the last calculated remainder to the first) to get the binary number

For example, to convert the denary number 135_{10} into binary using method 1, the result would be calculated like this:

Denary	Quotient	Remainder
$135 / 2$	67	1
$67 / 2$	33	1
$33 / 2$	16	1
$16 / 2$	8	0
$8 / 2$	4	0
$4 / 2$	2	0
$2 / 2$	1	0
$1 / 2$	0	1

Therefore, $135_{10} = 10000111_2$

Method 2: place values table

1. Draw a table with the place values of each digit
2. Find the highest place value(2^n number) that is equal or less than the denary number
3. Set the digit that corresponds to that place value to 1
4. Subtract the place value from the denary number
5. Take the result of the subtraction and repeat steps 1–4 until you have a result of 0
6. Set the remaining digits to 0
7. Read the binary number from left to right

For example, to convert the denary number 135_{10} into binary using method 2, the result would be calculated like this:

For an 8-bit number, the place values table is:

128	64	32	16	8	4	2	1

The maximum place value divisor of 135 is 128 so the corresponding digit is set to 1:

128	64	32	16	8	4	2	1
1							

$135 - 128 = 7$. The maximum place divisor of 7 is 4, so the corresponding digit is set to 1:

128	64	32	16	8	4	2	1
1					1		

$7 - 4 = 3$. The maximum place divisor of 3 is 2, so the corresponding digit is set to 1:

128	64	32	16	8	4	2	1
1					1	1	

$3 - 2 = 1$. The maximum place divisor of 1 is 1, so the corresponding digit is set to 1:

128	64	32	16	8	4	2	1
1					1	1	1

$1 - 1 = 0$. The subtraction process ends, and all remaining digits are set to 0:

128	64	32	16	8	4	2	1
1	0	0	0	0	1	1	1

Hence, $135_{10} = 10000111_2$

2.2 Binary to Denary

Each digit of a binary number has a specific place value that is a power of 2.
For an 8-bit number, the place values table is:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1

To convert a binary number into denary, you add up the place values only for the binary digits that are set to 1.

Consider the binary number 11000110.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	0	0	0	1	1	0

$$(1 \times 2^7) + (1 \times 2^6) + (0 \times 2^5) + (0 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (0 \times 2^0) = 198$$

$$128 + 64 + 4 + 2 = 198$$

Its denary equivalent is 198_{10}

2.3 Hexadecimal to Binary

To convert a hexadecimal number to binary:

1. Break up the number into its individual hexadecimal digits and
2. Using the table below as a reference, replace each hexadecimal digit with its four-digit binary equivalent.
3. Read the binary number from left to right.

Hexadecimal digit	0	1	2	3	4	5	6	7
Binary equivalent	0000	0001	0010	0011	0100	0101	0110	0111
Hexadecimal digit	8	9	A	B	C	D	E	F
Binary equivalent	1000	1001	1010	1011	1100	1101	1110	1111

To convert $B03_{16}$ into binary:

Hexadecimal digit	B	0	3
Binary equivalent	1011	0000	0011

Therefore, the binary number is 101100000011_2 .

2.4 Binary to Hexadecimal

To convert a binary number to its hexadecimal equivalent:

1. Split the number into groups of four binary digits, starting from the right.
2. If there is a group on the left that does not have the full set of four binary digits, add leading zeros to that group until it is a full set of four binary digits.
3. Using the table below as a reference, convert each four-digit group of binary numbers into its corresponding hexadecimal digit.

Hexadecimal digit	0	1	2	3	4	5	6	7
Binary equivalent	0000	0001	0010	0011	0100	0101	0110	0111
Hexadecimal digit	8	9	A	B	C	D	E	F
Binary equivalent	1000	1001	1010	1011	1100	1101	1110	1111

Consider converting $10\ 1000\ 0110\ 1011_2$ to hexadecimal.

Split the binary number into groups of four digits, starting from the right

Binary	10	1000	0110	1011

Notice that the leftmost group only contains two digits, we add two leading zeros to its left to form a full set of four digits.

Binary	0010	1000	0110	1011
Hexadecimal	2	8	6	B

Each group of four binary digits is then converted into a hexadecimal digit.

Hence, the hexadecimal equivalent of $10\ 1000\ 0110\ 1011_2$ is $286B_{16}$.

2.5 Denary to Hexadecimal

There are two methods for converting a denary number to its hexadecimal equivalent.

Method 1: division by 16

This method follows the same principle of repeated divisions with remainder used for converting a denary number into binary:

1. Divide the denary number by 16, and write down the result and the remainder
2. Repeat the division until you get a result of 0
3. Convert the denary remainders to their hexadecimal equivalent
4. Write the remainders in reverse (from the last calculated remainder to the first) to get the final result

For example, to convert the denary number 1899_{10} into hexadecimal using method 1, the result would be calculated like this:

Denary	Quotient	Remainder	Hexadecimal
$1899 / 16$	118	11	B
$118/16$	7	6	6
$7/16$	0	7	7

Therefore, $1899_{10} = B67_{16}$

Method 2: place values table

1. Draw a table with the place values of each digit
2. Find the highest place value(16^n number) that is equal or less than the denary number
3. Find largest hexadecimal digit below this place value such that its product with this place value is equal to or less than the denary number.
4. Subtract this product from the denary number
5. Take the result of the subtraction and repeat steps 1–4 until you have a result of 0
6. Set the remaining digits to 0
7. Read the hexadecimal number from left to right starting from the first non-zero digit.

For example, to convert the denary number 1899_{10} into hexadecimal using method 2, the result would be calculated like this:

The place values table is:

Place value	65536	4096	256	16	1
	16^4	16^3	16^2	16^1	16^0

The maximum place value that is equal or less than 1899 is 256. The largest hexadecimal digit such that its product with 256 is equal to or less than 1899 is 7_{16} .

Place value	65536	4096	256	16	1
	16^4	16^3	16^2	16^1	16^0
Hexadecimal			7		

Subtract the product $7 \times 16^2 = 1792$ from 1899 will give 107 which is a non-zero result. Hence, we repeat the previous step.

The maximum place value that is equal or less than 107 is 16. The largest hexadecimal digit such that its product with 16 is equal to or less than 107 is 6_{16} .

Place value	65536	4096	256	16	1
	16^4	16^3	16^2	16^1	16^0
Hexadecimal			7	6	

Subtract the product $6 \times 16^1 = 96$ from 107 will give 11 which is a non-zero result. Hence, we repeat the previous step again.

The maximum place value that is equal or less than 11 is 1. The largest hexadecimal digit such that its product with 1 is equal to or less than 11 is 11_{10} . Its hexadecimal equivalent is B_{16} .

Place value	65536	4096	256	16	1
	16^4	16^3	16^2	16^1	16^0
Hexadecimal			7	6	B

Subtract the product $11 \times 16^0 = 11$ from 11 will give 0. As the result is 0, we can proceed to the last step of the algorithm. In this case, all the place values after the first non-zero hexadecimal digit are already filled.

Hence, reading the first non-zero hexadecimal digit from the left, the hexadecimal equivalent of $1899_{10} = 76B_{16}$.

Method 3: via binary

1. Convert the denary number into binary.
2. Then perform the steps for the binary-to-hexadecimal conversion

2.6 Hexadecimal to Denary

To convert a hexadecimal number to its denary equivalent:

1. Convert each hexadecimal digit to its equivalent denary number.
2. Multiply each denary number by its place value.
3. Sum the product of each denary number and its place value.

Consider converting $C7F_{16}$ to denary:

Place value	256	16	1
	16^2	16^1	16^0
Hexadecimal digit	C	7	F
Denary equivalent	12	7	15

Multiply the denary equivalent of each digit with its place value as shown in the table above and sum up the products.

$$(12 \times 16^2) + (7 \times 16^1) + (15 \times 16^0) = 3199$$

Hence, the denary equivalent of $C7F_{16}$ is 3199.

Chapter 7: File Input/Output (I/O)

Contents

1. Introduction	2
2. Open and Close Files	2
2.1 Opening a non-existent file	3
2.2 Using the with method	3
3. Read from a file	3
3.1 Example: <code>read()</code>	4
3.2 Example: <code>readline()</code>	5
3.3 Example: <code>readlines()</code>	5
4. Writing and Appending to a File	6
4.1 Example: <code>write()</code>	6
4.2 Example: <code>writelines()</code>	7
5. Text File & Binary File	8
6. CSV File I/O	8
6.1 Python CSV Module	8
6.2 Example: <code>csv.reader()</code>	9
6.3 Example: Save result as a list	9
6.4 Example: Save header and data as two separate lists	10
6.5 Using <code>csv.writer()</code>	10
6.6 Example: <code>writerow()</code> and <code>writerows()</code>	10
7. JSON Processing	11
7.1 Example: <code>dumps()</code>	12
7.2 Example: More readable version	12
8. Types of files	13
8.1 Serial Files	13
8.2 Sequential Files	13
9. File Paths	14
9.1 Absolute Path	14
9.2 Relative Path	14
Referenced Readings	16

Syllabus Learning Outcomes

2.3 Implementing Algorithms and Data Structures

2.3.4 Store data in and retrieve data from serial and sequential text files.

1. Introduction

File is a named location on a disk to store related information.

- It uses non-volatile memory, e.g. hard disk, to store data permanently.
- A file can be in text or binary format.

Advantages of using a file:

- It is more efficient to read large amounts of data from a file, compared to manual data entry.
- We are able to output the results to a file for ease of future reference.

A file operation takes place in the following order:

- Open a file
- Perform operations
- Close the file

2. Open and Close Files

Python has a built-in function `open(file_path)` to open a file.

- The `open()` function returns a file object, also called a file handler, as it is used to read or modify the file accordingly.

To read data from a file object, use its `read()` method.

To close a file object, use its `close()` method. It flushes any unwritten information and closes the file object, after which no more writing can be done. This prevents accidental access to an open file.

Python automatically closes a file when the reference object of a file is reassigned to another file. It is a good practice to use the `close()` method to close a file to release the resources.

2.1 Opening a non-existent file

It is important to ensure the file resides in the same folder as your Python notebook or file. Assuming `test.txt` is not in the same folder as the Python file, you will get similar error messages.

```
[ ] file = open("test.txt")

-----
FileNotFoundError                                Traceback (most recent call last)
<ipython-input-1-c1d6406a1bac> in <module>()
----> 1 f = open("test.txt")

FileNotFoundError: [Errno 2] No such file or directory: 'test.txt'
```

2.2 Using the with method

Another way of opening a file is using the **with** method. This method automatically handles the closing of the file.

```
[ ] with open('quote.txt') as file:
    print(type(file))

    print(file.closed)

<class '_io.TextIOWrapper'>
True
```

3. Read from a file

There are 3 main modes when opening a file

- read from the file
- write to the file
- append to the file

The default mode when calling `open()` is to read from the file. We can choose to include the default mode when opening the file, as a reminder to ourselves.

There are 3 methods to read a file.

Operator	How it reads
<code>read()</code>	Reads the entire text as a single string
<code>readline()</code>	Reads the text line by line
<code>readlines()</code>	Read the entire text and returns a set of lines

3.1 Example: `read()`

Make sure you have downloaded `quote.txt` in the same folder as your Python notebook or file.

```
[ ] with open('quote.txt', 'r') as file: # mode to open with is after the file path
    text = file.read()
    print(text)

    print()

    # for developers we prefer to print repr()
    with open('quote.txt') as file: # the mode 'r' is optional in this case because it is the default mode
        text = file.read()
        print(repr(text)) # prints a representation of the text
        # note the quotation marks and \n
```

Give a man a program, frustrate him for a day.
Teach a man to program, frustrate him for a lifetime.

'Give a man a program, frustrate him for a day.\nTeach a man to program, frustrate him for a lifetime.'

3.2 Example: readline()

```
[ ] # read the text line by line
    with open('quote.txt', 'r') as file:
        line = file.readline()
        print(line)

# how do we print all the lines?

with open('quote.txt', 'r') as file:
    while True: # assume we do not know how many lines there are
        line = file.readline()
        if line: # we have not reached the end of the file
            print(line.strip())
        else: # we have reached the end of the file
            break
```

Give a man a program, frustrate him for a day.

Give a man a program, frustrate him for a day.

Teach a man to program, frustrate him for a lifetime.

3.3 Example: readlines()

```
[ ] # read the text line by line
    with open('quote.txt', 'r') as file:
        lines = file.readlines()
        print(lines)
```

```
['Give a man a program, frustrate him for a day.\n', 'Teach a man to program, frustrate him for a lifetime.']
```

Often, we do not store the files in the same directory as the script or notebook. In this case, we have to make use of the `os` module or `Path` module. For the sake of simplicity, this will not be covered here but you are encouraged to find out how to use the above modules to access files in a different directory. A good starting point is to go to website link 7 in the Referenced Readings section of this document.

4. Writing and Appending to a File

When we want to write to a file, we open the file with the mode `w`.

Things to note when using the write mode.

- If the file does not exist, a new file will be created.
- Warning: If the file exists, it will be overwritten. (All previous content will be removed.)
- `write()` is similar to `print()`, except it does not include newline character (hence the `\n`)
- Data is written as a text string.

When using append mode,

- If the file does not exist, a new file will be created.
- If the file exists, new data will be added to the end of the string.
- As with write mode, use `write()`.
- Data is written as a text string.

4.1 Example: `write()`

Complete the following operations using **with** code block.

- Write `"Alexa, "` to a file `test.txt`. This operator will overwrite any content in the file.
- Append `"Good morning!\n"` to the file `test.txt`.
- Append `"What time is it?"` to the file `test.txt`.
- Read and print out content from the file `test.txt`.

```
[1] # write mode
    with open('test.txt', 'w') as file:
        file.write("Alexa, ")

    # read mode
    with open('test.txt', 'r') as file:
        print(file.read())
```

☞ Alexa,

```

▶ #append mode
with open('test.txt', 'a') as f:
    f.write("Good morning!\n" )
    f.write("What time is it?")

# default mode is read
with open('test.txt') as f:
    print(f.read())

# clear text file
with open('test.txt', 'w') as f:
    pass # we are not writing anything, hence this will clear the file

with open('test.txt', 'r') as f:
    print(f.read())
    print("end")

```

 Alexa, Good morning!
 What time is it?

end

4.2 Example: writelines()

Similar to readlines(), writelines() will write a list of strings into the target file.

Write the strings in text_list into test.txt.

```

[ ] text_list = ['Hello', 'World', '\nfrom', '\nVJC']

with open ('test.txt', 'w') as f:
    f.writelines(text_list)

with open('test.txt', 'r') as f:
    print(f.read())

```

HelloWorld
 from
 VJC

5. Text File & Binary File

By default, `open()` assumes the file is a text file. To work with binary files, e.g. images and sound files, adding `b` to the mode. Examples of binary content include `jpg` files.

- Use `rb` to read a binary file
- Use `wb` to write binary content to a file

6. CSV File I/O

Comma-Separated Values (CSV) is a common text file format to store tabular data.

- It has a simple structure where each row contains one record
- A record may have multiple attributes delimited by common delimiter such as `,`
- Normally, the first line is the header

Sample CSV data:

```
Name,Email,Phone Number,Address
Bob Smith,bob@example.com,123-456-7890,123 Fake Street
Mike Jones,mike@example.com,098-765-4321,321 Fake Avenue
```

Why Use CSV?

- CSV is a common format for data exchange because it is simple and compact.
- Most relational databases provide tools to import and export CSV files.
- CSV files can be easily opened in Excel.

6.1 Python CSV Module

While we could use the built-in `open()` function to work with CSV files in Python, there is a dedicated `csv` module that makes working with CSV files much easier. It contains following built-in functions:

```
csv.reader
csv.writer
writerow()
writerows()
```

6.2 Example: csv.reader()

Make sure you have the file `info.csv` in the same folder as the notebook. Print out all the rows in the file.

```
[ ] import csv

with open('info.csv', 'r') as file:
    reader = csv.reader(file)
    for r in reader:
        print(r)
```

```
['id', 'name', 'dob', 'email', 'mobile']
['1', 'Lim Ah Seng', '1/1/1995', 'limahseng@hotmail.com', '12345678']
['2', 'Tan Ah Lian', '31/12/1995', 'tanahlian@yahoo.com', '87654321']
```

6.3 Example: Save result as a list

```
 import csv # by right, we only need to import it once, just in case we want to run this cell on its own.

result = []

with open('info.csv', 'r') as file:
    reader = csv.reader(file)
    for r in reader:
        result.append(r)

print(result)
```

```
[[ 'id', 'name', 'dob', 'email', 'mobile'], ['1', 'Lim Ah Seng', '1/1/1995', 'limahseng@hotmail.com', '12345678'], ['2', 'Tan Ah Lian', '31/12/1995', 'tanahlian@yahoo.com', '87654321']]
```

6.4 Example: Save header and data as two separate lists

```
[ ] result = []

with open('info.csv', 'r') as file:
    reader = csv.reader(file)
    header = next(reader) # this is done before the for loop
    for r in reader: # start appending data from where the cursor is at
        result.append(r)

print(header)
print()
print(result)
```

```
['id', 'name', 'dob', 'email', 'mobile']
```

```
[[ '1', 'Lim Ah Seng', '1/1/1995', 'limahseng@hotmail.com', '12345678'], [ '2', 'Tan Ah Lian', '31/12/1995', 'tanahlian@yahoo.com', '87654321']]
```

6.5 Using csv.writer()

A `csv.writer()` can be used to write a CSV file. The `csv.writer()` function returns a writer object that converts the user's data into a delimited string and write to file using its `writerow()` function.

The `newline` argument is set to “” when opening a file which the `csv.writer()` will write each row in a line.

6.6 Example: `writerow()` and `writerows()`

Write the following data to a csv file.

```
Name,Email,Phone Number,Address
Robert,bob@example.com,123-456-7890,123 Fake Street
Michael,mike@example.com,098-765-4321,321 Fake Avenue
```



```
[ ] header = ('Name','Email','Phone Number','Address')

data = [
    ('Robert','bob@example.com','123-456-7890','123 Fake Street'),
    ('Michael','mike@example.com','098-765-4321','321 Fake Avenue')
]

import csv

# write mode
with open('test2.csv', 'w', newline='') as file: #newline set to '',to remove empty rows
    writer = csv.writer(file)
    writer.writerow(header)

with open('test2.csv', 'r') as file:
    reader = csv.reader(file)
    for r in reader:
        print(r)

print()
# append mode
with open('test2.csv', 'a', newline='') as file: #newline set to '',to remove empty rows
    writer = csv.writer(file)
    writer.writerows(data)

with open('test2.csv', 'r') as file:
    reader = csv.reader(file)
    for r in reader:
        print(r)
```

```
['Name', 'Email', 'Phone Number', 'Address']
```

```
['Name', 'Email', 'Phone Number', 'Address']
```

```
['Robert', 'bob@example.com', '123-456-7890', '123 Fake Street']
```

```
['Michael', 'mike@example.com', '098-765-4321', '321 Fake Avenue']
```

7. JSON Processing

JSON is another popular syntax for storing and exchanging data. JSON is text, written with JavaScript object notation. Python denotes JSON data as a dictionary.

Python has a built-in package called `json`, which can be used to work with JSON data. Some useful built-in functions include

- `loads`
- `dumps`

7.1 Example: dumps()

```
[ ] import json

# Python data (can be int, float, string, unicode, list, dictionary, tuple)
pdata = [{'a': 1, 'c': 3, 'b': 2}]
print('Python data:', pdata)

# encode python data as json, resultant is a string
encoded = json.dumps(pdata)
print('json data:', repr(encoded))
```

```
Python data: [{'a': 1, 'c': 3, 'b': 2}]
json data: '[{"a": 1, "c": 3, "b": 2}]'
```

7.2 Example: More readable version

```
 encoded_sort_indent = json.dumps(pdata, sort_keys=True, indent=2)
print(encoded_sort_indent)
```

```
 [
    {
        "a": 1,
        "b": 2,
        "c": 3
    }
]
```

8. Types of files

There is a wide variety of file types. Whatever the file type, the content is stored using a defined binary code that allows the file to be used that it is intended for.

For storing data to be used by a program, there are only 2 defined file types- text file or a binary file. A text file contains data stored according to a defined character code. A binary file stores data in its internal representation and is created using a specific program. For instance, integer values may be stored in 2 bytes in two's complement representation.

The organisation of a binary file is based on the concept of a record. A file contains records and each record contains fields. Each field contains a value.

There are generally 2 types of files that we will be dealing with - serial and sequential. The main difference between the two is that in sequential files, the order of the data is maintained.

Other types of files include index sequential files for large data sets and random files or direct-access files which allow you to access any item without searching through the file from the start.

8.1 Serial Files

Serial files contain records which have no defined order. Specifically, they are stored in chronological order, that is, as each record is received it is stored in the next available storage position. This type of file organisation means that the records are in no particular order and therefore, to retrieve a single record, the whole file needs to be read from beginning to end. An example of a serial file would be a bank's records of transactions.

A text file can be considered a type of serial file but it is different because the file has repeating lines which are defined by end-of-line characters. There is no end-of-record character. This means that a record in a serial file must have a defined format to allow data to be input and output correctly.

Refer to this video for more details: https://www.youtube.com/watch?v=ULHh_3tBhvU

8.2 Sequential Files

Sequential files are serial files whose records are sorted in ascending or descending on a particular key field. Key fields contain values that are unique and sequential but not necessarily consecutive. They are the types of files suited to long term storage of data.

Therefore, they can be considered as an alternative to a database but do note that primary keys in a database table are required to be unique but ~~not~~ to be sequential. In a sequential file, a particular record is found sequentially reading the value of the key field until the required value is found.

Note that the physical order of the records on the disk is not necessarily sequential, as most manufacturers support an organisation where certain records (inserted after the file has been set up) are held in a logical sequence but are physically placed into an overflow area. They are no longer physically contiguous with the preceding and following logical records, but they can be retrieved in sequence.

Refer to this video for more details: <https://www.youtube.com/watch?v=5czzGLILjVY>

9. File Paths

File paths specify the location of individual files so that we can access these files. There are 2 types of file paths - absolute and relative.

9.1 Absolute Path

Absolute file paths are the exact file location in the computer or server. We usually use them to specify the location of files over the internet and will always include a domain name as part of the link. For example, in CSS, we can denote the location of a file as a such:

```
<img src = "https://www.booble.com/images/bee.png">
```

Other examples:

```
http://website.com/assets/image.jpg
```

Meaning: Go to this domain to access the image

```
//website.com/assets/image.jpg
```

Meaning: The image is uploaded to the above location using http or https protocols

9.2 Relative Path

Relative file path points to the location of the file in the root folder of an individual web project with reference to the current working file. We usually use this in our coding and programming projects.

Refer to the examples below. As you can see, we use relative paths for internal use, provided the image is on the same server.

```
image.jpg
```

Meaning: The image is in the same location as the document e.g. python file which is calling the image.

```
./image.jpg
```

Meaning: Same as above. The image is in the same place as the document calling the image.

```
/assets/image.jpg
```

Meaning: This is similar to the absolute path method except that the protocol and domain name are omitted. It is telling the computer to search for the image starting from the root folder /, and then look for it in assets/

```
assets/image.jpg
```

Meaning: The assets folder is in the same location as the document. The image is in the assets folder.

```
../assets/image.jpg
```

Meaning: From where the document is, go one folder back ../ and look for the image in the assets folder.

```
../../image.jpg
```

Meaning: From where the document is, go two folders back ../../ and access the image.

```
../../assets/image.jpg
```

Meaning: From where the document is, go two folders back ../../ and then, look into assets to find the image.

Referenced Readings

1. https://computersciencewiki.org/images/b/b1/Programming_companion_Python.pdf
2. <https://towardsdatascience.com/getting-start-with-python-i-o-9a28ce635ba4>
3. <https://realpython.com/working-with-files-in-python/>
4. <https://realpython.com/read-write-files-python/>
5. https://www.w3schools.com/python/python_file_handling.asp
6. <https://www.analyticsvidhya.com/blog/2021/08/python-tutorial-working-with-csv-file-for-data-science/>
7. <https://medium.com/@ageitgey/python-3-quick-tip-the-easy-way-to-deal-with-file-paths-on-windows-mac-and-linux-11a072b58d5f>
8. <https://discuss.codecademy.com/t/what-does-the-newline-argument-do/463575>

9. [https://medium.com/@Linda_Ikechukwu/understanding -file-paths-165c07ec5cf0#:~:text=File%20paths%20specify%20the%20location%20of%20individual%20files.,two%20types%20%3A%20Absolute%20and%20Relative.&text=R%20relative%20file%20paths%20on%20the,to%20the%20current%20working%20file.](https://medium.com/@Linda_Ikechukwu/understanding-file-paths-165c07ec5cf0#:~:text=File%20paths%20specify%20the%20location%20of%20individual%20files.,two%20types%20%3A%20Absolute%20and%20Relative.&text=R%20relative%20file%20paths%20on%20the,to%20the%20current%20working%20file.)
10. Langfield, S., & Duddell, D. (2018) *Cambridge International as and A level computer science*. Cambridge University Press.

Chapter 8: Recursion

Contents

- 1 Trace tables
 - 1.1 Examples
 - 1.2 Exercises
- 2 Recursion
 - 2.1 Recursion without return value
 - 2.2 Recursion with return value
 - 2.3 Using stacks to store return addresses
 - 2.4 Benefits and drawbacks of recursion

Syllabus Learning Outcomes

- 2.2 Programming Elements and Constructs
 - Use programming language elements and constructs to write recursive and non-recursive programs to solve a variety of problems.
 - 2.2.5 Understand the concept of recursion.
 - 2.2.6 Trace the steps and list the results of recursive and non-recursive programs.
 - 2.2.7 Understand the use of stacks in recursive programming.

1 Trace tables

Using trace tables is a technique used to test an algorithm and predict step by step how the computer will run the algorithm.

Trace tables are tables that consist of columns. Each column can represent a variable, a condition, or an output. Not every variable, condition or output in an algorithm needs a column in a trace table.

The purpose of the table is that we can run through an algorithm and simulate what a computer would do if the program were to execute. We complete the table to show how the variables change, what the conditions would resolve to, and/or what outputs would be displayed.

There are two main reasons that trace tables are used. The first is to determine what an algorithm does by running through it to see what happens as the algorithm runs. The second is to test the logic of an algorithm if there are errors that are not easily spotted.

1.1 Examples

A trace table can be used to track the values of variables as they change from line to line while the program is running.

Algorithm		Trace Table			
1	number = 3	Line	number	i	OUTPUT
2	PRINT number	1	3		
3	FOR i from 1 to 3:	2			3
4	number = number + 5	3		1	
5	PRINT number	4	8		
6	PRINT " ? "	5			8
		3		2	
		4	13		
		5			13
		3		3	
		4	18		
		5			18
		6			?

"Line" may not always be a column in a trace table. The above trace table can be drawn with columns consisting only the variables and output.

number	i	OUTPUT
3		3
8	1	8
13	2	13
18	3	18
		?

The following examples show some basic algorithms and their trace tables.

Example 1

The purpose of the algorithm in this example is to swap the values of x and y.

Algorithm	Trace table									
<pre>1 x = 3 2 y = 4 3 4 temp = x 5 x = y 6 y = temp</pre>	<table><tr><th>x</th><th>y</th><th>temp</th></tr><tr><td>3</td><td>4</td><td>3</td></tr><tr><td>4</td><td>3</td><td></td></tr></table>	x	y	temp	3	4	3	4	3	
x	y	temp								
3	4	3								
4	3									

Example 2

This algorithm includes a conditional statement - IF/ELSE.

Algorithm	Trace table									
<pre>1 code = 61 2 IF code MOD 2 == 0 THEN 3 code = code * 3 4 ELSE 5 code = code * 2 6 ENDIF 7 OUTPUT(code)</pre>	<table><tr><th>code</th><th>code MOD 2 == 0</th><th>OUTPUT</th></tr><tr><td>61</td><td>False</td><td>3</td></tr><tr><td>122</td><td></td><td>122</td></tr></table>	code	code MOD 2 == 0	OUTPUT	61	False	3	122		122
code	code MOD 2 == 0	OUTPUT								
61	False	3								
122		122								

Example 3

The algorithm below contains 2 variables (`num`, `count`), 1 condition (`num < 500`) and 1 output (`OUTPUT num`). The variable `count` will keep track of how many times the while loop has repeated. The trace table shows how this algorithm would run if the user entered an input of 16.

Algorithm	Trace table																																				
<pre>1 num ← USERINPUT 2 count ← 0 3 4 WHILE num < 500 THEN 5 num ← num * 2 6 count ← count + 1 7 ENDWHILE 8 9 OUTPUT num 10 OUTPUT count</pre>	<table><tr><th>num</th><th>count</th><th>num < 500</th><th>OUTPUT</th></tr><tr><td>16</td><td>0</td><td>True</td><td></td></tr><tr><td>32</td><td>1</td><td>True</td><td></td></tr><tr><td>64</td><td>2</td><td>True</td><td></td></tr><tr><td>128</td><td>3</td><td>True</td><td></td></tr><tr><td>256</td><td>4</td><td>True</td><td></td></tr><tr><td>512</td><td>5</td><td>False</td><td></td></tr><tr><td></td><td></td><td></td><td>512</td></tr><tr><td></td><td></td><td></td><td>5</td></tr></table>	num	count	num < 500	OUTPUT	16	0	True		32	1	True		64	2	True		128	3	True		256	4	True		512	5	False					512				5
num	count	num < 500	OUTPUT																																		
16	0	True																																			
32	1	True																																			
64	2	True																																			
128	3	True																																			
256	4	True																																			
512	5	False																																			
			512																																		
			5																																		

1.2 Exercises

Complete the trace tables for the following algorithms.

Algorithm 1	Trace table										
<pre> 1 code = "" 2 code = code + "7" 3 code = "4" + code 4 code = code + "3" 5 OUTPUT (code) </pre>	<table> <tr> <th>num</th><th>OUTPUT</th></tr> <tr> <td></td><td></td></tr> <tr> <td></td><td></td></tr> <tr> <td></td><td></td></tr> <tr> <td></td><td></td></tr> </table>	num	OUTPUT								
num	OUTPUT										

Algorithm 2

```

1 number = 5
2 factorial = number
3
4 WHILE number > 2
5     number = number - 1
6     factorial = factorial * number
7 END WHILE
8
9 OUTPUT(factorial)

```

Trace table

number	factorial	number > 2	OUTPUT

Algorithm 3

```

1 code = ""
2
3 FOR i FROM 1 TO 3
4     digit = i * 2
5     code = code + STR(digit)
6 ENDFOR
7
8 OUTPUT(code)

```

Trace table

i	digit	code	OUTPUT

2 Recursion

The process in which a function calls itself directly or indirectly is known as recursion. The corresponding function is called a recursion function.

Generally, a recursive function has two main features:

1. Calls itself with one or more similar but smaller problems.
2. Repeats itself until a base case or terminating case is reached. A base case or terminating condition is the smaller problem that is small enough to be solved trivially.

Any iteration can possibly be rewritten into a recursive function and vice versa.

2.1 Recursion without return value

Let's look at a `countdown()` function using while loop.

It takes in a parameter `n`, which is the starting number to count down to 1.

```
1 def countdown(n):
2     while n > 0:
3         print(n)
4         n = n - 1
5     print('Done!')
6
7 # main
8 countdown(3)
```

3
2
1
Done!

Can this function be converted into a recursive function? YES!

How do we convert the countdown function into a recursive function?

Analysis steps:

- What are the common actions in each loop? (Hint: check the loop statements)
- What is the base case (or terminating case)?
- What does it do when it is the base/terminating case?

Analysis result:

- In each iteration, it prints out current `n` value.
- The base case is when `n = 0`.
- If base case is `True`, it prints `Done`.

Thus,

- In the recursive function, it checks for the base case.
- Once the base case is true, it prints `Done`, else it prints current `n` value and recurses (calls the same function again)

Let's convert the `countdown()` function into a recursive function `countdown_r()`.

```

1 def countdown_r(n):
2     if n == 0:
3         print("Done")
4     else:
5         print(n)
6         countdown_r(n-1)
7
8 # main
9 countdown_r(3)

```

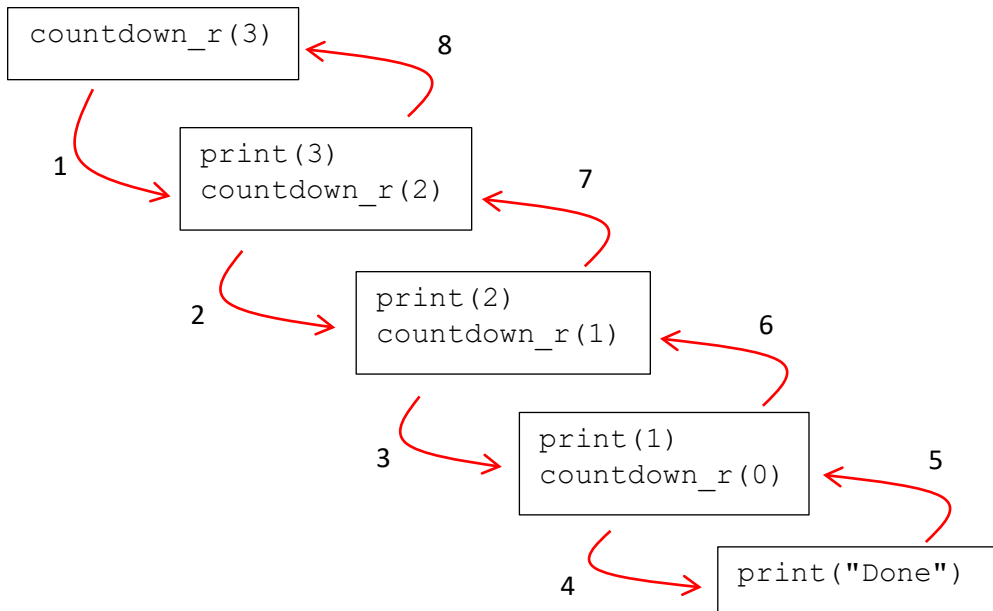
3
2
1
Done

How do you visualise the above recursive function execution?

1. Trace table

Call number	Procedure call	n	n == 0	OUTPUT
1	countdown_r(3)	3	False	3
2	countdown_r(2)	2	False	2
3	countdown_r(1)	1	False	1
4	countdown_r(0)	0	True	"Done"

2. Trace tree



Let's modify the `countdown()` function into a recursive function count up function `countup_r()`.

```

1  def countup_r(n):
2      if n == 0:
3          print("Done")
4      else:
5          countup_r(n-1)
6          print(n)
7
8
9  # main
10 countup_r(3)

```

Done
1
2
3

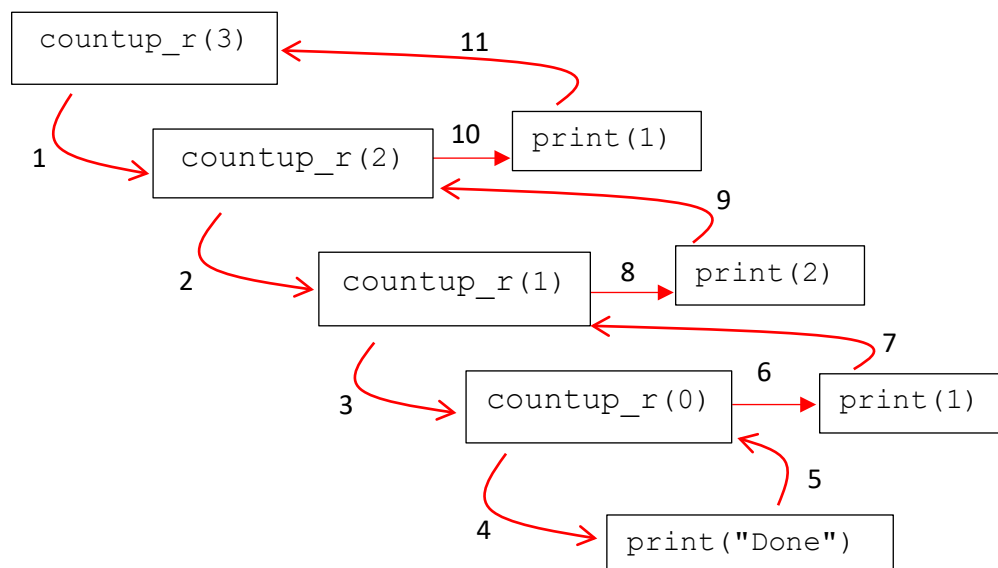
By switching the code on line 5 and 6, we can totally change a count down to a count up as the order of execution has been switched. Instead of printing `n` followed by calling the function, the program recursively calls the function until it reaches the base case then it starts the printing.

Let's visualise the execution order of the `countup_r` function.

1. Trace table

Call number	Procedure call	n	n == 0	OUTPUT
1	countup_r(3)	3	False	
2	countup_r(2)	2	False	
3	countup_r(1)	1	False	
4	countup_r(0)	0	True	"Done"
3	countup_r(1)	1	False	1
2	countup_r(2)	2	False	2
1	countup_r(3)	3	False	3

2. Trace tree



2.2 Recursion with return value

In the above count-down and count-up examples, the recursion function does not return any value. It is common for recursion functions to return value back to the calling function.

Let's look at a recursion function with return value.

Factorial

The factorial value of an explicit number n is typically represented as $n!$, and $F_n = n * F_{n-1}$.

$$n! = n * (n-1) * (n-2) * \dots * 1$$

$$n! = n * (n-1)!$$

We can implement an iterative function `factorial_i(n)` to calculate the factorial value of an integer n .


```

1 def factorial_i(n):
2     factorial = 1
3     while n > 0:
4         factorial *= n
5         n -= 1
6
7     return factorial
8
9 print(factorial_i(4))

```

24

How do we convert the iterative factorial function into a recursive function?

Analysis steps:

- What are the common actions in each iteration? (Hint: check the loop statements)
- What is the base case (or terminating case)?
- What does it do when it is the base/terminating case?

Analysis result:

- It sets `factorial = factorial * n` and repeats with `n-1`
- The base case is when `n = 1`
- If base case is `True`, program will execute while loop a last time and exit the loop. After which, it returns `factorial`.

Let's now implement the `factorial_r(n)` recursive function to compute `factorial(4)`.

```

1 def factorial_r(n):
2     if n == 1:
3         return 1
4     else:
5         return n * factorial_r(n-1)
6
7 print(factorial_r(4))

```

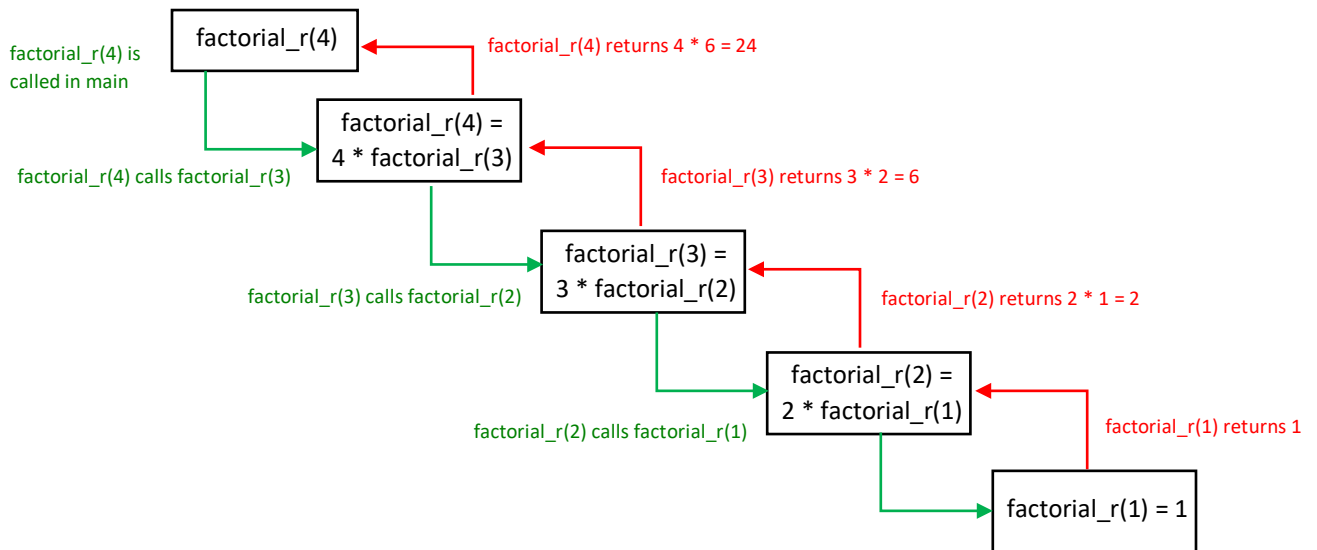
24

Let's visualise the execution order of the `factorial_r(n)` recursive function using a trace table and a trace tree.

1. Trace table

Call number	Procedure call	n	n == 1	Return value
1	factorial_r(4)	4	False	
2	factorial_r(3)	3	False	
3	factorial_r(2)	2	False	
4	factorial_r(1)	1	True	1
3	factorial_r(2)	2	False	2
2	factorial_r(3)	3	False	6
1	factorial_r(4)	4	False	24

2. Trace tree



2.3 Using stacks to store return addresses

When a function is called, a certain amount of memory is set aside for that function to use for purposes such as storing local variables. This memory, called a stack frame, is also used by the computer to store information about the function such as the function's address in memory; this allows the program to return to the proper place after a function call (for example, if you write a function that calls `printf()`, you would like control to return to your function after `printf()` completes; this is made possible by the frame).

When a function calls another function, the caller's stack space is kept intact, and a new space (stack frame) is created to handle the new function call.

When a function finishes its work and returns to its caller, its associated space is released.

Illustration of stack using factorial_r(4):

```
1 def factorial_r(n):  
2     if n == 1:  
3         return 1  
4     else:  
5         return n * factorial_r(n-1)  
6  
7 print(factorial_r(4))  
  
24
```

What happens when the program above is executed?

The first program statement to be executed is line 7. The actual parameter `n` takes in the value of 4. The function call causes the return address to be put on the stack. Program execution jumps to line 1 to execute the function `factorial_r(4)`.

The execution of `factorial_r(4)` begins. Since it is not base case, line 5 will be executed, `n` is decremented by 1 and becomes 3. The function call, `factorial_r(3)`, causes the return address of `factorial_r(4)` at line 5 to be stored on the stack, together with the current contents of the local variables. The locations used to store these values are referred to as a **stack frame**. Each recursive call will add another stack frame to the stack until the base case is reached.

When the base case `factorial_r(1)` is reached, 1 is returned by pushing it onto the stack. This value is popped off the stack and returned to the caller of this function. With each return from a function call, the corresponding stack frame is taken off and the values of the local variables are restored. Eventually, control is returned to main at line 7.

Description	Stack
1 st call factorial_r(4) is made by the main program. Return address to main is pushed into stack.	Return address to main
2 nd call factorial_r(3) is made by factorial_r(4). Return address to factorial_r(4) is pushed into stack.	Return address to factorial_r(4) 4 Return address to main
3 rd call factorial_r(2) is made by factorial_r(3). Return address to factorial_r(3) is pushed into stack.	Return address to factorial_r(3) 3 Return address to factorial_r(4) 4 Return address to main
4 th call factorial_r(1) is made by factorial_r(2). Return address to factorial_r(2) is pushed into stack.	Return address to factorial_r(2) 2 Return address to factorial_r(3) 3 Return address to factorial_r(4) 4 Return address to main
Base case reached. Push 1 onto stack.	1 Return address to factorial_r(2) 2 Return address to factorial_r(3) 3 Return address to factorial_r(4) 4 Return address to main

Pop 1 and stack frame. Return to factorial_r(2). Push new value of $1*2$ onto stack.	2 Return address to factorial_r(3) 3 Return address to factorial_r(4) 4 Return address to main
Pop 2 and stack frame. Return to factorial_r(3). Push new value of $2*3$ onto stack.	6 Return address to factorial_r(4) 4 Return address to main
Pop 6 and stack frame. Return to factorial_r(4). Push new value of $6*4$ onto stack. Return to main	24 Return address to main

2.3 Benefits and drawbacks of recursion

Recursion solutions are often more elegant and use less program code than iterative solutions. A complex task can be broken down into simpler sub-problems using recursion where iterative solutions may be difficult to program. When designing a solution to a mathematical problem that is recursive by nature, recursive solution is easier to implement than an iterative one.

However, repeated recursive calls can carry large amounts of memory usage and processor time from the multiple function calls, each time storing of return addresses and copies of local or temporary variables. If the recursion continues too long, computer may run out of memory and the program will crash.

Chapter 9: Search Algorithms

Contents

Syllabus Learning Outcomes	2
1. Pre-requisites	2
2. Introduction to Search Algorithms	3
3. Common Types of Search Algorithms	3
3.1 Linear SearchAlgorithm	3
3.2 Binary Search Algorithm	3
4. Linear Search algorithm	4
4.1 Linear Search on unsorted lists concept	4
4.2 Linear Search on sorted lists	4
4.3 Example: Linear Search Algorithm in a List Using Forloop	4
4.4 Example: Linear Search Algorithm in a List using while-loop	5
5. Binary Search algorithm	5
5.1 Binary Search Concepts	6
5.2 Linear and Binary SearchDemo on Sorted Lists	6
5.3 Example: Binary Search Using Iterative Loop	6
5.4 Example: Binary Search using Recursive Function	7
6. Time Complexity	8
6.1 For Linear Search	8
6.2 For Binary Search	8
7. Hash Table Search	9
7.1 Introduction to Hash Table	9
7.2 Implementing a Hash Table	9
7.2.1 Create an entry	10
7.2.2 Read a stored entry	12
7.2.3 Update a record	13
7.2.4 Delete record from file	15
8. Problems with implementation so far	17
9. Handling Collisions	18
9.1 Closed Addressing	18
9.1.1 Changes to be made	18

9.2 Open Addressing	21
9.2.1 Implement Linear Probing	21
10. Comparisons Between Open and Closed Addressing	25
10.1 Advantages of Closed Addressing	25
10.2 Advantages of Open Addressing	25
11. Good Hash Table	26
11.1 Size of hash table	26
11.2 Good hash function	26
Referenced Readings	26

Syllabus Learning Outcomes

1.2 Fundamental Algorithms

1.2.3 Implement search algorithms.

- Linear search
- Binary search
- Hash table search

1.2.4 Use examples to explain search algorithms.

1.2.5 Compare and describe the efficiencies of the search algorithms using Big-O notation for time complexity (worst case). Exclude space complexity

2.3 Implementing Algorithms and Data Structures

2.3.2 Implement search programs.

- Linear search
- Binary search
- Hash table search

1. Pre-requisites

- **How to sort items in a List?**

Reading: <https://developers.google.com/edu/python/sorting>

Video: <https://youtu.be/D3JvDWO-BY4>

- **How to find a single (multiple) item(s) in a List?**

Reading: <https://able.bio/rhett/find-items-in-a-python-list-82irtvz>

Video: <https://youtu.be/nr5vmCdnwBA>

- **How to find keys by values in a Dictionary?**

Reading: https://www.w3schools.com/python/python_dictionaries.asp

Video: <https://youtu.be/daefaLgNkw0>

- **Iterative vs Recursive Function**

Video on Iterative Loops: <https://www.youtube.com/watch?v=6iF8Xb7Z3wQ>

Video on Recursive Function:

<https://www.youtube.com/watch?v=wMNRSM5RFMc>

2. Introduction to Search Algorithms

A search algorithm is an algorithm that takes a collection and a target item and determines whether the target is in the collection, often returning the index of the target. Sometimes we return a collection of indices if we are interested in finding all the indices corresponding to the value.

A simpler version will return whether the target item is in the collection. (Similar to membership testing)

3. Common Types of Search Algorithms

3.1 Linear Search Algorithm

The simplest search algorithm is a “linear search”, which traverses the items of the collection in order, stopping if it finds the target. In the worst case, it has to traverse the entire collection, so the run time is linear. (See section below on Big O Notation for more details.)

3.2 Binary Search Algorithm

If the elements of the sequence are in order (i.e. sorted), you can use a bisection search. Bisection search is similar to the algorithm you might use to look a word up in a dictionary (a paper dictionary, not the data structure). Instead of starting at the beginning and checking each item in the order, you start with the item in the middle and check whether the word you are looking for comes before or after. If it comes before, then you search the first half of the sequence. Otherwise, you should search for the second half.

4. Linear Search algorithm

Linear search searches an item in a given list sequentially till the end of the collection. It is one of the simplest searching algorithms and it commonly uses for -loop or while-loop to iterate through the collection.

Click on the links to play and visualise how linear search algorithms work.

4.1 Linear Search on unsorted lists concept

Watch the following videos and strive to build a mental image/animation of how this works.

- <https://www.youtube.com/watch?v=PDE8pCQ9Tz4>
- https://youtu.be/vDlrhF8V_FE [Original video: <https://youtu.be/UldZOLylez4>]

4.2 Linear Search on sorted lists

Watch this video and have a mental image/animation of how it works.

<https://www.cs.usfca.edu/~galles/visualization/Search.html>

4.3 Example: Linear Search Algorithm in a List Using For-loop

Write program code to search an item in a given list A. Return the index of any occurrence of target if found else return -1 if target is not in A.

```
def linear_searchf(A, target):
    for i in range(len(A)): #using for-loop
        if target == A[i]:
            return i
    return -1
```

```
from random import randint
```

```
target = randint(1,100)
print(target)
```

```
A = [randint(1,100) for n in range(20)]
print(A)
```

```
linear_searchf(A, target)
```

87

```
[31, 84, 23, 87, 66, 74, 53, 2, 70, 55, 28, 100, 5, 86, 44, 68, 88,
2, 7, 97]
```

3

4.4 Example: Linear Search Algorithm in a List using while -loop

```
def linear_searchw(A,target):
    i=0
    while i < len(A):          #using while-loop
        if A[i] == target:
            return i
        i=i+1
    return -1
```

```
from random import randint
```

```
target = randint(1,100)
print(target)
```

```
A = [randint(1,100) for n in range(20)]
print(A)
```

```
linear_searchw(A, target)
```

83

```
[2, 9, 61, 90, 63, 48, 77, 15, 3, 55, 14, 5, 50, 2, 4, 81, 93, 40,
34, 60]
```

-1

Think about this :

When will linear search be the most efficient?

When will linear search be the least efficient?

5. Binary Search algorithm

Binary Search uses a divide-and-conquer methodology. It is faster than linear search. However, it requires the array to be sorted.

The binary search involves searching a sorted array by repeatedly dividing the search interval in half, beginning with an interval spanning the entire array.

- First, check the MIDDLE element in the list.
- If it is the value we want, we can stop.
- If it is HIGHER than the value we want, we repeat the search process with the portion of the list BEFORE the middle element.
- If it is LOWER than the value we want, we repeat the search process with the portion of the list AFTER the middle element.

5.1 Binary Search Concepts

Familiarise yourself with the concepts by watching this video.

<https://youtu.be/Q83XYPIZ4p8> [Original Video: https://youtu.be/DE_ye0t0oxE]

5.2 Linear and Binary Search Demo on Sorted Lists

The video below will be useful for your Linear and Binary Search visualisations exercise.

<https://www.cs.usfca.edu/~galles/visualization/Search.html>

Note that the binary search algorithm can be written using either iterative loops or recursive functions.

5.3 Example: Binary Search Using Iterative Loop

Given a list A, return the index of any instance of the target if found, else return -1 if target is not in A.

```
def bin_search_iterative(A, target):
    lb = 0 #index of lower bound
    ub = len(A)-1 #index of upper bound
    while lb <= ub: #lower <= upper
        mid = (lb+ub)//2 #middle index
        if target == A[mid]:
            return mid
        else:
            if target > A[mid]:
                lb = mid +1 # lower bound more than prev mid, consider second
            half
```

```

        else:
            ub = mid - 1 # upper bound less than prev mid, consider first
half
        return -1

## Test the code
import random
A = [random.randint(1,20) for i in range(10)]
A.sort()
target = random.randint(1,20)
print(A, target)
print( bin_search_iterative(A,target))

```

```

[3, 3, 6, 9, 12, 13, 14, 17, 18, 19] 4
-1

```

5.4 Example: Binary Search using Recursive Function

```

def bin_search_recur(A, lb, ub, target):

    if lb > ub:
        return -1
    else:
        mid = (lb + ub)//2
        if target == A[mid]:
            result.append(mid)
            return mid
        elif target > A[mid]:
            return bin_search_recur(A, mid+1, ub, target)
        else:
            return bin_search_recur(A, lb, mid-1, target)

## Test the code
import random
A = [random.randint(1,20) for i in range(10)]
A = sorted(A)
target = random.randint(1,20)
print(A, target)
print(bin_search_recur(A, 0, len(A)-1, target))

```

```

[1, 2, 2, 2, 2, 11, 12, 12, 13, 20] 5
-1

```

6. Time Complexity

Now we know why a binary search algorithm is much preferred to a linear search algorithm!

In a binary search algorithm, you cut the number of remaining items in half. If the sequence has 1,000,000 items, it will take about 20 steps to find the word or conclude that it's not there. So that's about 50,000 times faster than a linear search.

Watch the video below to see how we calculate time complexity.

https://youtu.be/nf_iZIRSMT8 [Original videos: https://youtu.be/nf_iZIRSMT8, https://www.youtube.com/watch?v=__vX2sjlpXU]

6.1 For Linear Search

All items are searched one-by-one to find the required item.

If the sample size is n ,

- The best-case lookup to find an item is 1, i.e., the item is at the head of the list.
- The worst-case lookup to find an item is n , i.e. the item is at the end of the list.
- The average lookup to find an item is $n/2$.

The time complexity of the linear search is $O(n)$, meaning that the time taken to execute increases with the number of items in our input list.

6.2 For Binary Search

If the same size is n ,

- The best-case lookup is 1.
- The worst-case lookup is $\log(n)/\log(2)$.
- Every iteration, the sample size is halved: $n/2, n/4, \dots$ which become 1 after $\log(n)/\log(2)$ iterations.

Though bisection search can be much faster than linear search, it requires the sequence to be in order, which might require extra work.

Thus, there is another data structure, called a hashtable that is even faster—it can do a search in constant time—and it doesn't require the items to be sorted. Python dictionaries

are implemented using hash tables, which is why most dictionary operations, including the `in` operator, are constant time.

7. Hash Table Search

Watch the video clip on Introduction to Hash Table Search at

<https://youtu.be/iJM3liZWCFE>.

(Taken from https://www.youtube.com/watch?v=KyUTuwz_b7Q&t=104s.)

As seen from the video, in a good hash table, the complexity is independent of the amount of data to be stored. This means that the complexity of this search is $O(1)$.

We will discuss the characteristics of a good hash table at the end of the notes.

7.1 Introduction to Hash Table

A hash table is a data structure where a calculated value is used to mark the position in the table where the data item is to be stored. This enables the data to be accessed directly.

The calculated address is found by passing a key through a hash function. This key could be part of the data item or the entire data item.

The positions where data can be stored are sometimes known as buckets.

By coding our own hash table, we are able to customise and control how the keys are hashed, and how collisions are handled.

7.2 Implementing a Hash Table

There are 2 components that we will need:

1. hash table (to be implemented as a list)
2. hash function

Like what we have learnt under data structures, we need to be able to perform CRUD operations.

```
# initialize an hash table
```

```
def hash_table(size):
```

```

    return [None for n in range(size)]

size = 11
table = hash_table(size)
print(table)

# define the hash function
def hash1(key):
    # like the video we are going to find the ascii for each character in the
    key,
    #sum them up then take modulo <size>
    key = str(key) # convert to string in case we get other data types

    total = 0
    for char in key:
        total += ord(char)

    return total % size

print(hash1('Mia'))

```

```

[None, None, None, None, None, None, None, None, None, None, None]
4

```

7.2.1 Create an entry

Let's write a function that takes in a key and data to be stored and do the following:

- For every key calculate the position where the data will be stored
- If the location is empty, we will insert the data at the location We will talk about the case when the location is not empty later

```

# initialize an hash table

def hash_table(size):
    return [None for n in range(size)]

size = 11
table = hash_table(size)
print(table)

```

```

# define the hash function
def hash1(key):
    # like the video we are going to find the ascii for each character in the
    key,
    # sum them up then take modulo <size>
    key = str(key) # convert to string in case we get other data types

    total = 0
    for char in key:
        total += ord(char)

    return total % size

# print(hash1('Mia'))

# define insert function, without collision handling
def insert(key, data):
    pos = hash1(key)

    if table[pos] is None: # pos is available
        table[pos] = data

# Suppose we want to store the phone numbers of some contacts

phone_numbers = {'Mia': '91234567', 'Tommy': '83894789', 'Sue': '93876342'}

for k, v in phone_numbers.items():
    insert(k, [k,v])

print(table)

# why is sue not in the table?
for k in phone_numbers.keys():
    print(k, hash1(k))

# There is a collision between Mia and Sue

```

```

4
[None, None, None, None, ['Mia', '91234567'], None, ['Tommy',
'83894789'], None, None, None, None]
Mia 4

```


Tommy 6

Sue 4

7.2.2 Read a stored entry

Let's write a function that takes in a key and do the following:

- For the key calculate the position where the data should be stored
- If the location is empty, data not found
- If the location is not empty and
 - if the data stored at the location has the same key, data is found
 - if the data stored at the location has a different key, data is not found (for now)

We will talk about the case of collision later

```
def hash_table(size):
    return [None for n in range(size)]

size = 11
table = hash_table(size)
print(table)

# define the hash function
def hash1(key):
    # like the video we are going to find the ascii for each character in the
    key,
    # sum them up then take modulo <size>
    key = str(key) # convert to string in case we get other data types

    total = 0
    for char in key:
        total += ord(char)

    return total % size

# print(hash1('Mia'))

# define insert function, without collision handling
def insert(key, data):
    pos = hash1(key)
```

```

    if table[pos] is None: # pos is available
        table[pos] = data

# define search function, without collision handling
def read(key):
    pos = hash1(key)

    if table[pos] is not None and table[pos][0] == key:
        print("Contact found")
        return pos
    else:
        print("Contact not Found")
        return

# Suppose we want to store the phone numbers of some contacts

phone_numbers = {'Mia': '91234567', 'Tommy': '83894789', 'Sue': '93876342'}

for k, v in phone_numbers.items():
    insert(k, [k,v])

print(table)
print()

for key in ['Mia', 'Sue', 'Timmy']:
    pos = read(key)
    if pos: print(table[pos])

```

```

[None, None, None, None, ['Mia', '91234567'], None, ['Tommy',
'83894789'], None, None, None, None]

```

```

Contact found
['Mia', '91234567']
Contact not Found
Contact not Found

```

7.2.3 Update a record

Let's create a function that takes in a key and new data and do the following:

- searches for the record given the key
 - if the record is found, new data is stored in place of the old data
 - if the record is not found, output an appropriate message

```
# initialize an hash table

def hash_table(size):
    return [None for n in range(size)]

size = 11
table = hash_table(size)
print(table)

# define the hash function
def hash1(key):
    # like the video we are going to find the ascii for each character in the
    key,
    # sum them up then take modulo <size>
    key = str(key) # convert to string in case we get other data types

    total = 0
    for char in key:
        total += ord(char)

    return total % size

# print(hash1('Mia'))

# define insert function, without collision avoidance
def insert(key, data):
    pos = hash1(key)

    if table[pos] is None: # pos is available
        table[pos] = data

# define search function, without collision handling
def read(key):
    pos = hash1(key)

    if (table[pos] is not None) and (table[pos][0] == key):
        print("Contact found")
        return pos
    else:
```

```

        print("Contact not Found")
        return

# define update function
def update(key, new_data):
    pos = read(key)
    if pos is not None: # key is found at the original position, because no
collision handling
        table[pos] = [key, new_data]
        print("Contact has been updated")
    else: print("Contact not updated")

# Suppose we want to store the phone numbers of some contacts

phone_numbers = {'Mia': '91234567', 'Tommy': '83894789', 'Sue': '93876342'}

for k, v in phone_numbers.items():
    insert(k, [k,v])

print(table)

update('Mia', '81111111')
update("Sue", "83321747")
print(table)

```

```

[None, None, None, None, ['Mia', '91234567'], None, ['Tommy',
'83894789'], None, None, None, None]
Contact found
Data has been updated
Contact not Found
Data not updated
[None, None, None, None, ['Mia', '81111111'], None, ['Tommy',
'83894789'], None, None, None, None]

```

7.2.4 Delete record from file

Let's write a function that takes in a key and do the following:

- searches for the record given the key
 - if the record is found, delete the data
 - if the record is not found, output an appropriate message

```

# initialize an hash table

def hash_table(size):
    return [None for n in range(size)]

size = 11
table = hash_table(size)
print(table)

# define the hash function
def hash1(key):
    # like the video we are going to find the ascii for each character in the
    # key, sum them up then take modulo <size>
    key = str(key) # convert to string in case we get other data types

    total = 0
    for char in key:
        total += ord(char)

    return total % size

# print(hash1('Mia'))

# define insert function, without collision avoidance
def insert(key, data):
    pos = hash1(key)

    if table[pos] is None: # pos is available
        table[pos] = data

# define search function, without collision avoidance
def read(key):
    pos = hash1(key)

    if table[pos] is not None and table[pos][0] == key:
        print("Contact found")
        return pos
    else:
        print("Contact not Found")
        return

# define update function
def update(key, new_data):

```

```

    pos = read(key)
    if pos is not None: # key is found at the original position, because no
collision handling
        table[pos] = [key, new_data]
        print("Contact has been updated")
    else: print("Contact not updated")

# defina a delete function
def delete(key):
    pos = read(key)
    if pos is not None:
        table[pos] = None
        print('Contact deleted')
    else: print('Contact not deleted')

# Suppose we want to store the phone numbers of some contacts

phone_numbers = {'Mia': '91234567', 'Tommy': '83894789', 'Sue': '93876342'}

for k, v in phone_numbers.items():
    insert(k, [k,v])

print(table)

delete('Mia')
delete("Sue")
print(table)

```

```

[None, None, None, None, ['Mia', '91234567'], None, ['Tommy',
'83894789'], None, None, None, None]
Contact found
Data deleted
Contact not Found
Data not deleted
[None, None, None, None, None, None, ['Tommy', '83894789'], None,
None, None, None]

```

8. Problems with implementation so far

Our program has several problems with its implementation.

- What should happen if two different keys are mapped to the same location?

9. Handling Collisions

Collisions happen when the hash function maps different keys to the same location. The methods to handle collisions are divided into 2 categories:

- Closed Addressing
- Open Addressing

9.1 Closed Addressing

Watch the video clip on Closed Addressing at <https://youtu.be/kQJ1cWMFC-k>. (Taken from https://www.youtube.com/watch?v=KyUTuwz_b7Q&t=104s)

For Closed Addressing, the data is always stored in the same position it is hashed to. Some other forms of data structure are usually used to store all the data at the calculated position.

The video suggested that we can implement closed addressing using linked lists. We can also implement it using dynamic arrays. (Python lists are dynamic arrays. Arrays in other languages may be static arrays.)

9.1.1 Changes to be made

Each position in the hash table is now implemented as a dynamic array (Python List). We will call the position a *bucket*. We are not just interested in the position in the hash table, but also the index in the bucket where the data is stored.

```
# initialize an hash table

def hash_table(size):
    return [None for n in range(size)]

size = 11
table = hash_table(size)
print(table)

# define the hash function
def hash1(key):
    # like the video we are going to find the ascii for each character in the
    # key, sum them up then take modulo <size>
    key = str(key) # convert to string in case we get other data types
```

```

    total = 0
    for char in key:
        total += ord(char)

    return total % size

# define insert_c
def insert_c(key, data):
    pos = hash1(key)
    if table[pos] is None:
        table[pos] = [data] # impement as a dynamic array
        print(f'{key} has been added.')
    else:
        bucket = table[pos]
        total = sum([1 for item in bucket if item[0] == key]) # check if key
already exists in bucket
        if total > 0:
            print("Key already exists, use update_c() instead.")
        else:
            bucket.append(data)
            print(f'{key} has been added.')

# define read_c
def read_c(key):
    pos = hash1(key)
    bucket = table[pos]
    if bucket is not None and sum([1 for item in bucket if item[0] == key]) !=
0:
        print("Contact found")
        return pos
    else:
        print("Contact not found")
        return

# define update_c
def update_c(key, new_data):
    pos = read_c(key)
    if pos is not None:
        bucket = table[pos]
        bucket_keys = [item[0] for item in bucket]
        idx = bucket_keys.index(key)
        bucket[idx] = new_data
        print("Contact updated")
    else: print("Contact not updated")

```



```
# define delete_c
def delete_c(key):
    pos = read_c(key)
    if pos is not None:
        bucket = table[pos]
        bucket_keys = [item[0] for item in bucket]
        idx = bucket_keys.index(key)
        bucket.pop(idx)
        print("Contact deleted")

    if not bucket: # check if bucket is empty
#         bucket = None
        table[pos] = None # if bucket is empty, we replace the empty bucket
with None
    else: print("Contact not deleted")
```

```
[None, None, None, None, None, None, None, None, None, None, None]
Mia has been added.
Tommy has been added.
Sue has been added.
Key already exists, use update_c() instead.
[None, None, None, None, [['Mia', '91234567'], ['Sue', '93876342']],
None, [['Tommy', '83894789']], None, None, None, None]
```

4

```
Contact found
['Mia', '91234567']
Contact found
['Sue', '93876342']
Contact not found
Contact found
['Tommy', '83894789']
Contact not found
Contact found
Contact updated
Contact not found
Contact not updated
[None, None, None, None, [['Mia', '919191'], ['Sue', '93876342']],
None, [['Tommy', '83894789']], None, None, None, None]
Contact found
```

Contact deleted

```
[None, None, None, None, [['Sue', '93876342']], None, [['Tommy', '83894789']], None, None, None, None]
```

Contact **not** found

Contact **not** deleted

```
[None, None, None, None, [['Sue', '93876342']], None, [['Tommy', '83894789']], None, None, None, None]
```

Contact found

Contact deleted

```
[None, None, None, None, None, None, [['Tommy', '83894789']], None, None, None, None]
```

9.2 Open Addressing

Watch the video clip on Open Addressing on

https://www.youtube.com/watch?v=sdrDbg3QR_c.

(Taken from https://www.youtube.com/watch?v=KyUTuwz_b7Q&t=104s.)

For open addressing, when a collision is detected, we will search for another available slot in the hash table. This will mean that we are unable to add any more data if the hash table is full (there are no more available slots)

Common methods to search for another slot include linear probing and quadratic probing. For the sake of simplicity at this juncture, we will look at only linear probing in detail. You will be given the opportunity to work on quadratic probing in the tutorial.

9.2.1 Implement Linear Probing

When inserting a data item and a collision is detected, we will insert the data item at the next available slot, until we have finished trying all the slots

When searching for an item, we will calculate its original position based on the key. If the position is empty, data item is not found. If the item is at the original slot, the data item is found. Else, we will search the next slot until we find the data item or reach an empty slot.

Update and delete functions are similar to the earlier versions.

Do you foresee any problems with this implementation?

To get around the issue that we discussed, there needs to be a special deletion. Upon deleting an entry, it will not be the initial empty slot, but rather a "tombstone" indicating there used to be data there.

```
def hash_table(size):
    return [None for n in range(size)]

size = 10
table = hash_table(size)
# print(table)

# define the hash function
def hash2(key):
    # we are going to use a easier hash function to illustrate this example
    return key%size

# define insert_o
def insert_o(key, data):
    pos = hash2(key)

    if key in table:
        print(f'{key} exists. Use update_o() instead.')
        return

    i = pos
    while i < pos + size:
        n = i % size

        if table[n] is None or table[n] == "#": #"" is the "tombstone"
            table[n] = data
            print(f'{key} has been added.')
            return

        i += 1

    print("Table is Full, Data not added")

def read_o(key):
```

```

    pos = hash2(key)

    i = pos
    while i < pos + size:
        n = i % size

        if table[n] is None:
            print("Data not found")
            return
        elif table[n] == key:
            print("Data found")
            return n

    i += 1

    print("Data not found")
    return

def update_o(key, new_data):
    n = read_o(key)
    if n is not None:
        table[n] = new_data
        print(f'{key} updated')
    else: print(f'Data not updated')

def delete_o(key):
    n = read_o(key)
    if n is not None:
        table[n] = ""
        print(f'{key} deleted')
    else: print('Data not deleted')

# test insert_o
for n in range(5, 115, 10):
    insert_o(n,n)
insert_o(75,75)

print(table)
print()

# # test read_o
# for key in range(24, 27):
#     pos = read_o(key)

```

```

#     if pos: print(f'{key} found at {pos}')
# print()

# # test update_o
# update_o(65,1005)
# print(table)
# update_o(66,67)
# print(table)
# print()

for key in [55,55, 56]:
    delete_o(key)
    print(table)

read_o(65)

insert_o(10,10)
print(table)

```

```

5 has been added.
15 has been added.
25 has been added.
35 has been added.
45 has been added.
55 has been added.
65 has been added.
75 has been added.
85 has been added.
95 has been added.
Table is Full, Data not added
75 exists. Use update_o() instead.
[55, 65, 75, 85, 95, 5, 15, 25, 35, 45]

Data found
55 deleted
['', 65, 75, 85, 95, 5, 15, 25, 35, 45]
Data not found
Data not deleted
['', 65, 75, 85, 95, 5, 15, 25, 35, 45]
Data not found

```

```
Data not deleted
['', 65, 75, 85, 95, 5, 15, 25, 35, 45]
Data found
10 has been added.
[10, 65, 75, 85, 95, 5, 15, 25, 35, 45]
```

10. Comparisons Between Open and Closed Addressing

Characteristic structure (colors denote "home" bucket):	Characteristic structure (colors denote "home" bucket):
0:	0:
1: ①	1: ①
2: ②	2: ②②②
3: ②	3:
4: ②	4: ④
5: ④	5:
6:	6:
7: ⑦	7: ⑦⑦
8: ⑦	8:
9: ⑨	9: ⑨

Source: <https://programming.guide/hash-tables-open-vs-closed-addressing.html>

10.1 Advantages of Closed Addressing

- No theoretical maximum load factor (No of keys / size)
- Easier delete implementation (no tombstone markers)
- No issues with clustering
- Typically performs better with high load factor (increased collisions)

10.2 Advantages of Open Addressing

- Hash table has to be as large as the number of keys
- No size overhead (Uses only memory allocated for the hash table)

11. Good Hash Table

The requirements of a good hash table is to have a low chance of collision and to spread the data items evenly. There are 2 possible ways to address this.

11.1 Size of hash table

As we can see, if the size of the hash table is small, compared to the number of items to be stored, there will be a high chance of collision. In practice, the hash table is usually 1.5 times the maximum data size. We also sometimes use want the size of the hash table to be a prime number to minimize clustering or cyclic allocation.

11.2 Good hash function

We can clearly see that our `_hash()` function is overly simplistic. There is a high chance that it will map different key values to the same index. A good hash function should be fast to compute and able to distribute the keys uniformly, to avoid clustering and collision. Python 3 does have a `hashlib` module that implements a common interface to many different secure hash and message digest algorithms. (Read <https://docs.python.org/3/library/hashlib.html> for the documentation).

Theoretically, if we know all the possible key values, we should be to come up with a hash function that has no collisions.

Referenced Readings

All links to the referenced readings are found in the notes above.

In addition, you may refer to

1. Reeves, B. (2015). Aqa a Level Computer Science (UK ed.). Hodder Education.
2. Langfield, S., & Duddell, D. (2019). Cambridge International AS and A Level Computer Science Coursebook with Cambridge Elevate Edition (2 Years) (2nd ed.). Cambridge University Press.
3. Heathcote, P. M., & Heathcote, R. S. U. (2016). OCR AS and A Level Computer Science (Illustrated ed.). PG Online Limited.
4. Data Structure Visualization. (2011). David Galles Data Structure Visualizations. <https://www.cs.usfca.edu/%7Egalles/visualization/Algorithms.html>

5. Visualising data structures and algorithms through animation - VisuAlgo. (2011).
Dr Steven Halim VisuAlgo. <https://visualgo.net/en>

Chapter 10 Sorting Algorithms

Contents

- 1 Bubble sort
 - 1.1 Non-optimised bubble sort
 - 1.2 Optimised bubble sort
 - 1.3 Time complexity of bubble sort
- 2 Insertion sort
 - 2.1 Time complexity of insertion sort
- 3 Merge sort
 - 3.1 Time complexity of merge sort
- 4 Quicksort
 - 4.1 Out-of-place quicksort
 - 4.2 In-place quicksort
 - 4.3 Time complexity of quicksort
- 5 Merge sort vs Quick sort

Syllabus Learning Outcomes

- 1.2 Fundamental Algorithms

Understand algorithms for sorting and searching methods such as insertion sort, bubble sort, quicksort, merge sort, linear search, binary search and hash table search, and use examples to explain these methods.

 - 1.2.1 Implement sort algorithms.
 - Insertion sort
 - Bubble sort
 - Quicksort
 - Merge sort
 - 1.2.2 Use examples to explain sort algorithms.
 - 1.2.5 Compare and describe the efficiencies of the sort and search algorithms using Big-O notation for time complexity (worst case).

Exclude: space complexity
- 2.3 Implementing Algorithms and Data Structures

Use programming language elements and constructs to implement sort and search algorithms such as insertion sort, bubble sort, quicksort, merge sort, linear search, binary search, and hash table search, as well as data structures such as stacks, queues, linear linked lists, and binary search trees.

 - 2.3.1 Implement sort programs.
 - Insertion sort
 - Bubble sort
 - Quicksort
 - Merge sort

Sorting algorithms are used to put data in order. This data may be numbers, strings, records or objects. The four sorting algorithms you are expected to know are bubble sort, insertion sort, merge sort and quicksort.

Sorting algorithms could be compared based on

- Time taken to complete the work
- Memory requirement to complete the work
- Stability in keeping the input orders of items having the same sorting order

1. Bubble sort

Bubble sort is one of the easiest sorting algorithms to understand and implement; however, it is very inefficient compared to other alternatives.

1.1 Non-optimised bubble sort

Let's look at how a simple bubble sort is implemented.

It works as follows:

Compare the first and second values. If the first value is larger than the second value, swap them.

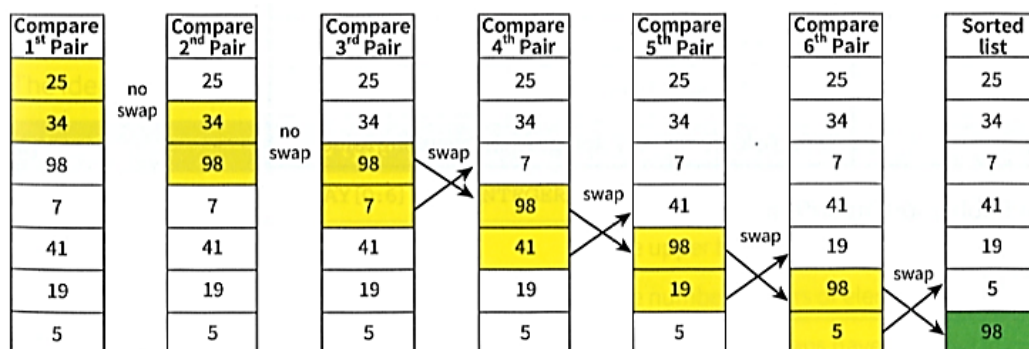
Compare the second and third values. If the second value is larger than the third value, swap them.

Keep on comparing adjacent values, swapping them if necessary, until the last two values in the list have been processed.

When we have completed the first pass through the entire array, the largest value is in the correct position at the end of the array. The other values may or may not be in the correct order.

We need to work through the array again and again with each pass having one less value to compare.

Illustration of bubble sort in one pass:



Swapping values working down the array in one pass

Original list	After pass 1	After pass 2	After pass 3	After pass 4	After pass 5	After pass 6
25	25	25	7	7	7	5
34	34	7	25	19	5	7
98	7	34	19	5	19	19
7	41	19	5	25	25	25
41	19	5	34	34	34	34
19	5	41	41	41	41	41
5	98	98	98	98	98	98

States of the array after each pass

The algorithm of non-optimised bubble sort in pseudocode is:

```

FOR passes ← 0 TO length of data - 2
  FOR j ← 0 TO length of data - 2 - passes
    IF data[j] > data[j + 1]
      THEN
        data[j], data[j + 1] ← data[j + 1], data[j]
      ENDIF
    ENDFOR
  ENDFOR

```

The values to be sorted may already be in the correct order before the outer loop has been through all its iterations. Look at the list of values below.

Original list	After pass 1	After pass 2	After pass 3	After pass 4	After pass 5	After pass 6
5	5	5	5	5	5	5
34	34	7	7	7	7	7
98	7	34	19	19	19	19
7	41	19	25	25	25	25
41	19	25	34	34	34	34
19	25	41	41	41	41	41
25	98	98	98	98	98	98

States of the array after each pass

After the third pass, the values are all in the correct order but the algorithm will continue to run. This means that we are making comparisons when no further swaps need to be made.

1.2 Optimised bubble sort

If we have gone through one pass without swapping any values, this means that the values must be in the correct order. So, we can improve the algorithm by using a variable `swapped` to store whether a swap has taken place during the current pass.

When we swap a pair of values, we set `swapped` to `True`. This indicates that the list was not sorted. If `swapped` is `False` at the end of one pass through the list, this indicates that none of the values are swapped. If none of the values are swapped, the list must be sorted, and we can exit the sort algorithm. This allows us to cut down on any unnecessary comparisons and passes which improves the bubble sort algorithm to make it optimised.

The algorithm of optimised bubble sort in pseudocode is:

```
swapped ← TRUE
passes ← length of data - 1

WHILE swapped DO
    swapped ← FALSE
    FOR j ← 0 TO passes - 1
        IF data[j] > data[j + 1]
            THEN
                data[j], data[j + 1] ← data[j + 1], data[j]
                swapped ← TRUE
            ENDIF
    ENDFOR
    passes ← passes - 1
ENDWHILE
```

1.3 Time complexity of bubble sort

Iteration	No. of comparisons
1	n-1
2	n-2
...	...
n-1	1

First, we assume that we will be sorting the array (of size n) using bubble sort in an ascending order. At the first iteration of the bubble sort, we are iterating through $n-1$ elements, until the largest element 'bubbles' to the end of the array. At the second iteration, we are iterating through $n - 2$ elements. After the second iteration, the second largest element bubbles up to the second last position of the array. This process continues until the whole array is sorted (or no more swaps occur, in the optimised case).

$$\text{No. of comparisons} = (n-1) + (n-2) + (n-3) + \dots + 3 + 2 + 1 = \frac{(n-1)n}{2}$$

Best case time complexity: $\Omega(n)$

Given a sorted list, we only need one pass with $n-1$ comparisons on the values in the list to check that it is sorted. This example will give a time complexity of $\Omega(n)$.

Worst case time complexity: $O(n^2)$

The non-optimised bubble sort algorithm will give the worst case time complexity of $O(n^2)$ as it needs to make $(n-1)*(n-2)$ comparisons to complete the sort.

2. Insertion Sort

Insertion sort works by dividing an array into two parts: sorted and unsorted. Elements are inserted one by one into their correct position in the sorted section.

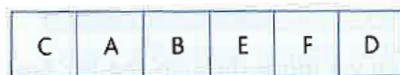
We start with the first element in the array and make this the sorted list. Then we consider the next element in the unsorted array. If it is smaller than the first element, we swap them. Next we consider the third element in the array. We swap it leftwards until it is in its proper order with the first two elements. We continue in this manner until the whole array is sorted.

The pseudo code of insertion sort algorithm is:

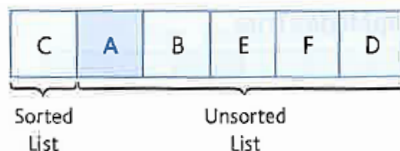
```
FOR i ← 1 TO length of data -1
    curr ← data[i] //Take the first item of the unsorted list
    position ← i
    WHILE position > 0 AND curr < data[position - 1] DO
        data[position] ← data[position - 1]
        position ← position - 1
    ENDWHILE
    data[position] ← curr
ENDFOR
```

Illustration of insertion sort:

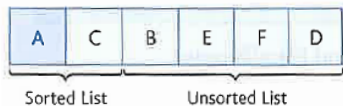
Let's start with a list of unsorted letters.



The first element is C and it becomes a member of the sorted list.

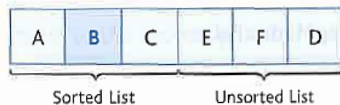


A, the first item of the unsorted list is smaller than C so is shuffled to the left of it.



A and C are now both in the sorted list.

B is now the first item in the unsorted list. B is less than C so is shuffled to the left of it. B is not less than A, so it does not get shuffled any further.

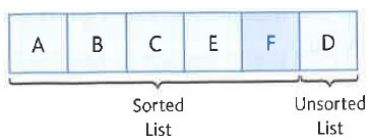


E is now the first item of the unsorted list.

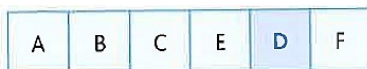
E is not less than C so it does not need shuffling anywhere.



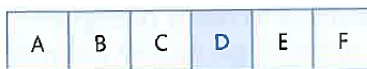
Similarly, F is not less than E, so it then joins the sorted list without being shuffled.



D is now the only member of the unsorted list. It is less than F so shuffles left.



D is less than E so shuffles left again.



D is not less than C so is now in its correct place.

All items in the list are now members of the sorted list.

2.1 Time complexity of insertion sort

The outer for-loop in insertion sort always iterates $n-1$ times.

The inner while loop will make $(n-1)*1$ comparisons in best case if all values are in correct order. The inner while loop will make $1 + 2 + 3 + \dots + (n-2) + (n-1)$ comparisons in worst case.

Best case time complexity: $\Omega(n)$

Worst case time complexity: $O(n^2)$

3. Merge Sort

A divide-and-conquer algorithm solves problem by breaking it into subproblems, recursively solves the subproblems, and finally combines the solutions to the subproblems to solve the original problem. Some key benefits of using a divide and conquer approach are speed and it simplifies complexity. A well constructed divide and conquer algorithm is typically very fast. It breaks a complex topic down into small and easy to manage chunks, which is most likely easy to solve.

Merge sort is a divide and conquer algorithm, which has elegant recursive formulations and is highly efficient. The implementation of the merge sort algorithm needs two functions:

1. A function that recursively splits the input array into half (`merge_sort(array)`)
2. A function that merges both halves, producing a sorted array. (`merge(list1, list2)`)

To understand merge sort, we first need to understand how we merge lists. If we have two sorted lists in ascending order, List1 and List2, we compare the first item in both lists. If the first item in List1 is smaller than that in List2, the first item is removed from List1 and added to a new list. If the first item in List2 is smaller than that in List1, the first item is removed from List2 and added to the new list. This comparison is repeated until either List1 or List2 becomes empty. The remaining items of the other list are added to the new list and merge sort is complete.

The merge algorithm in pseudo code:

```

FUNCTION merge (list1 : ARRAY, list2 : ARRAY) RETURNS ARRAY
    i ← 0
    j ← 0
    new_list ← []
    WHILE i < length of list1 AND j < length of list2 DO
        IF list1[i] < list2[j]
            THEN
                Append list1[i] to new_list
                i ← i + 1
            ELSE
                Append list2[j] to new_list
                j ← j + 1
            ENDIF
    ENDWHILE

    //If list1 still has remaining items, add remaining items in list1
    to //new_list, else add remaining items in list2 to new_list
    IF i < length of list1
        THEN
            new_list ← new_list + list1[i: ]
        ELSE
            new_list ← new_list + list2[j: ]
        ENDIF
    RETURN new_list
ENDFUNCTION

```

Illustration of merge function:



List1 and List2 are sorted. The first item of List2 (A) is lower than the first item of List1 (B), so the first item of List2 is added to the new list.



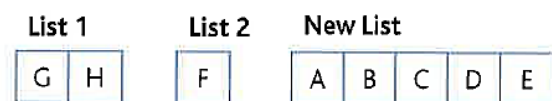
Now the first item in List1 (B) is smaller than first item in List2 (D), so the first item of List1 is added to the new list.



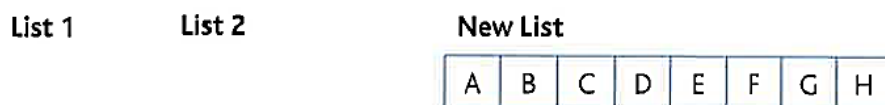
Again, the first item of List1 is smaller than first item of List2, so the first item of List1 is added to the new list.



This process of comparing the first items of the two lists continues until either list has no more remaining items to compare.



Finally, List2 has no more items left to compare. We therefore, append the remaining items of List1 to the new list.



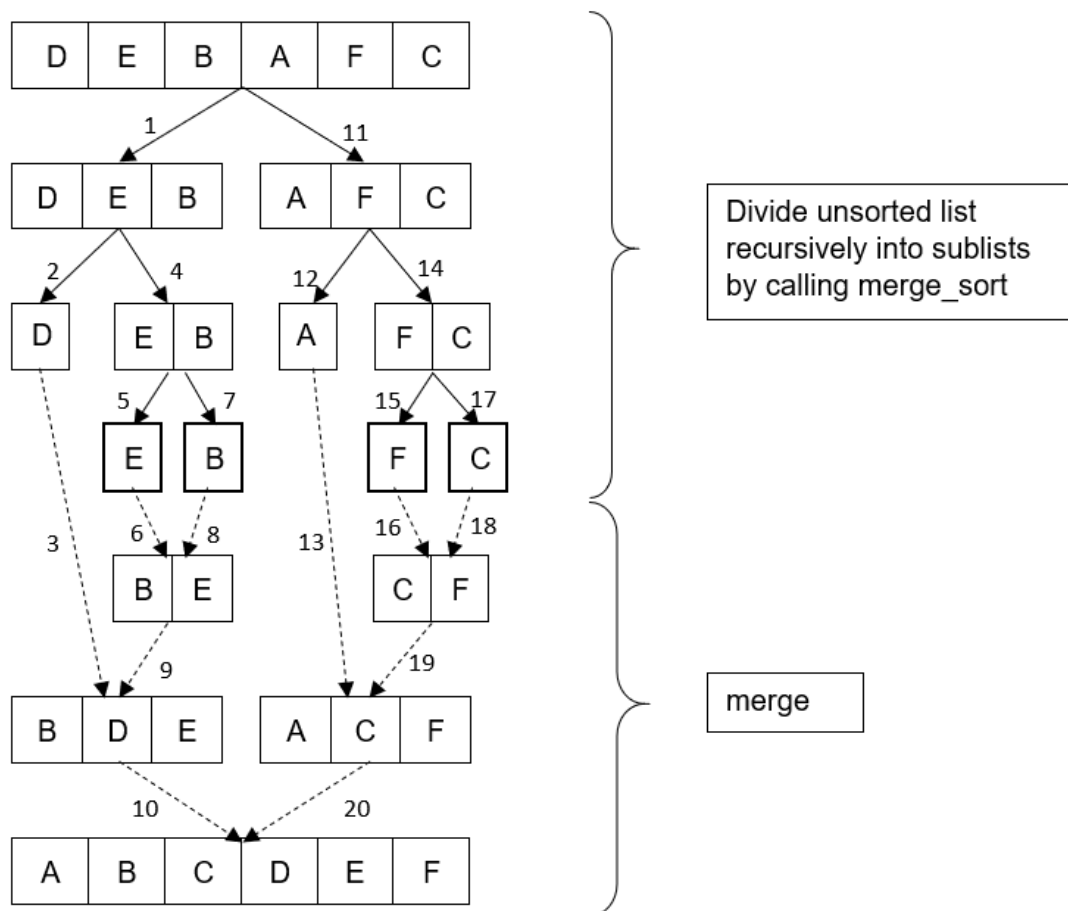
Merge sort recursively divides an unsorted array into subarrays, until each containing one element (an array of one element is considered a sorted array). Thereafter, the algorithm repeatedly pairs up the subarrays and merge each pair into a new sorted subarray, using the above process of merging arrays, until there is only one subarray remaining and that will be the sorted array.

```

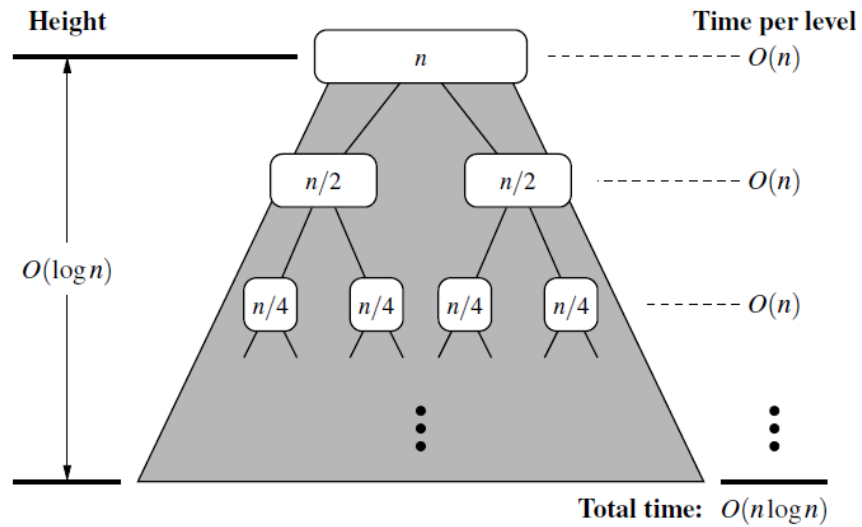
FUNCTION merge_sort(A : ARRAY) RETURNS ARRAY
  DECLARE mid : INTEGER
  IF Len(A) > 1
    mid ← length of A // 2
    list1 ← A[: Mid] //copy of first half of A
    list2 ← A[Mid :] //copy of second half of A
    sorted_list1 ← merge_sort(list1) //sort copy of first half
    sorted_list2 ← merge_sort(list2) //sort copy of second half
    RETURN merge(sorted_list1, sorted_list2) //merge sorted halves
  ENDIF
  RETURN A
ENDFUNCTION

```

Illustration of merge sort:



3.1 Time complexity of merge sort



As we have already learned in binary search that whenever we divide a number into half in every step, the time complexity can be represented using a logarithmic function, which is $O(\log n)$.

The time complexity of merging the subarrays of n elements is $O(n)$

Hence, merge sort will give a time complexity of $O(n \cdot \log n) = O(n \log n)$.

The time complexity for both worst and best cases is $O(n \log n)$ as merge sort always divides the array into two halves and take linear time to merge two halves.

Best case time complexity: $\Omega(n \log n)$

Worse case time complexity: $O(n \log n)$

4. Quicksort

Like merge sort, quicksort is also based on the divide and conquer algorithm, but it uses this technique in an opposite manner, as all the hard work is done before the recursive calls.

4.1 Out-of-place quicksort

Quicksort first selects a pivot element. The pivot element can be first, last, middle or any random element in the array. The algorithm then partitions the array around the pivot, putting every element less than the pivot in a `less_than` array, and every element greater than and equal to the pivot in a `greater_equal` array. The `equal` array will just hold one element – the pivot itself. The process repeats for the `less_than` array and `greater_equal` array. Eventually, the elements are combined in order by first inserting the elements of `less_than`, then those of `equal`, and finally those of `greater_equal`.

The algorithm of out-of-place quicksort in pseudo code:

```

FUNCTION quicksort (A : ARRAY) RETURNS ARRAY
  IF NOT A // If A is empty
    RETURN []

  less_than ← an array of all elements in A less than pivot
  equal ← pivot (element in the middle of the array)
  greater_equal ← an array of all elements in A greater than or
                  equal to pivot

  result ← quicksort(less_than) + equal + quicksort(greater_equal)

  RETURNS result
ENDFUNCTION

```

This works well if all elements are unique. If the original array contains elements with duplicates, the order of occurrence of two similar elements in the array may be altered and this makes the algorithm an unstable sorting algorithm.

An improvement can be made to make the above algorithm into a stable one. Instead of adding elements greater than and equal to the pivot into the `greater_equal` array, elements equal to the pivot can be stored in an `equal` array and elements greater than pivot can be stored in a `greater` array. This will preserve the original order of occurrence of the elements.

```

pivot ← element in the middle of the array
less ← an array of all elements in A less than pivot
equal ← an array of all elements in A equal to pivot
greater ← an array of all elements in A greater than pivot

```

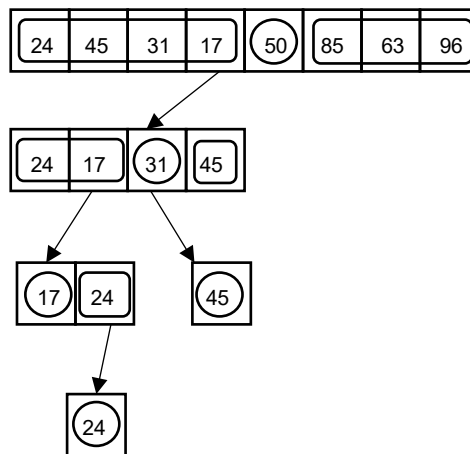
Illustration of quicksort:

85	24	63	45	50	31	96	17

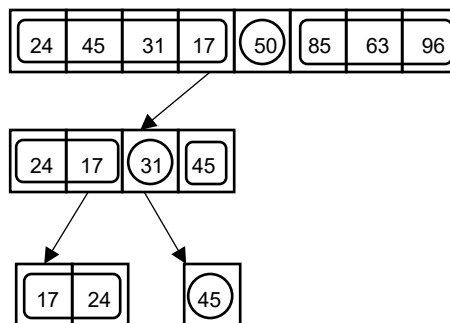
1. Starting with the array above, we take the middle element as pivot (it does not need to be the middle element, it can be any). We then divide the input array into 3 subarrays – `less`, `greater` and `equal`. Elements less than pivot will be appended to the `less` subarray, elements equal to pivot will be appended to the `equal` subarray and elements greater than pivot will be appended to the `greater` subarray. There is no attempt in sorting the arrays. Elements are just added in order.

24	45	31	17	50	85	63	96

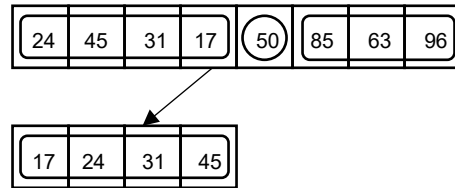
2. The same process is repeated for the `Less` subarray



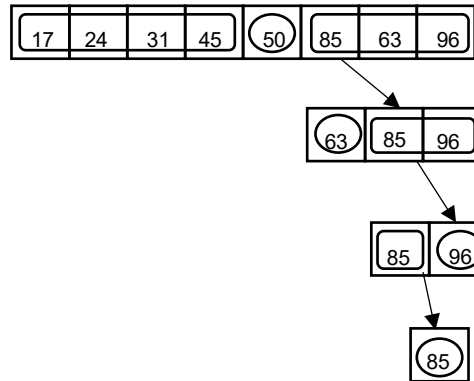
3. The subarrays are concatenated and returned.



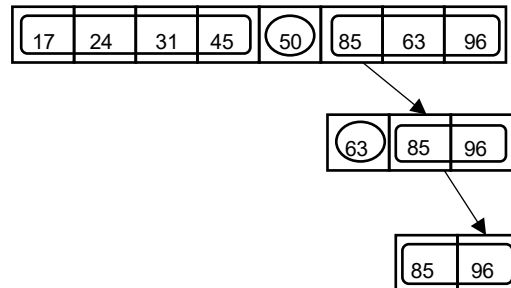
4. The subarray continues to concatenate and return.



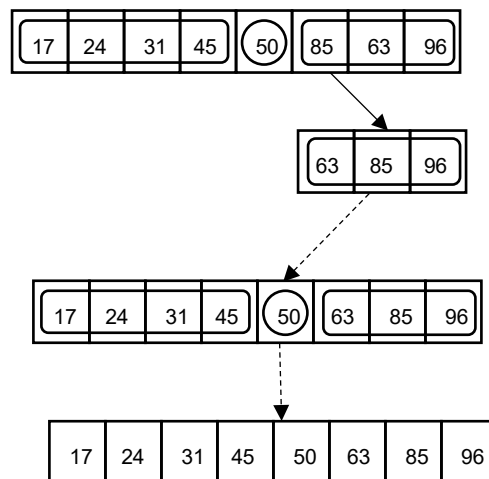
5. The same process is repeated for the `Greater` subarray.



6. The subarrays are concatenated and returned.



7. The subarray continues to concatenate and return. Eventually we get our sorted array.



4.2 In-place quicksort

Whilst a tremendously powerful method, using recursion on large data sets can be problematic. The computer can run out of memory, causing the dreaded ‘stack overflow’ error.

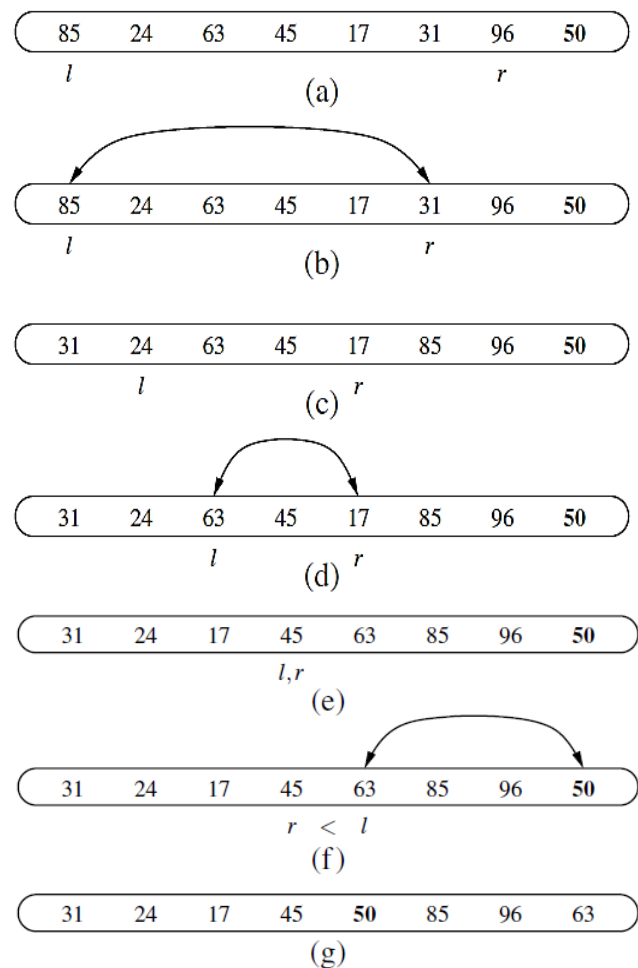
To avoid this problem, an **in-place** algorithm can be used. It uses only a small amount of memory in addition to that needed for the original input. The previous implementation of quicksort does not qualify as **in-place** because we use additional arrays `less`, `equal` and `greater` when dividing a sequence within each recursive call.

Assuming that the input sequence is given as a Python list of elements, **in-place** quicksort modifies the input sequence using element swapping and does not explicitly create subarrays.

Illustration of in-place quicksort:

Taking the last element as pivot.

1. The divide step is performed by scanning the array using left index as l , which advances forward, and right index as r , which advances backward.
2. A swap is performed when l is at an element larger than the pivot and r is at an element smaller than the pivot.
3. l continues advances forward and r advances backwards
4. Swap is performed again as l is at an element larger than the pivot and r is at an element smaller than the pivot.
5. l continues advances forward and r advances backwards
6. When the left index and the right index pass each other, a final swap with the pivot completes the divide step of in-place quick sort.
7. The algorithm completes by recurring on these two subarrays.



4.3 Time Complexity of quicksort

We would expect quicksort to run quickly, and it often does in practice. The best case for quick sort on a sequence of distinct elements occurs when the subarrays have roughly the same size. In that case, as we saw with merge sort, the tree has height $O(\log n)$ and therefore quicksort runs in $O(n \log n)$ time.

However, the worst case scenario is when the pivot chosen is always either the smallest or the largest element. This would create partitions of size $n-1$, causing recursive calls $n-1$ times. This leads to a worst case time complexity of $O(n^2)$.

Best case time complexity: $O(n \log n)$

Worse case time complexity: $O(n^2)$

5 Merge sort vs Quick sort

Quicksort is an unstable sorting technique. It might change the occurrence of two similar elements in the array while sorting. Merge sort is a stable algorithm. It doesn't change the occurrence of similar elements (i.e., equal elements are ordered in the same order in the sorted list) and preserve the order in which these elements appeared in the original array.

In-place quicksort does not need additional memory space to perform sorting. Merge sort requires a temporary array to merge the sorted arrays. Hence, merge sort needs more memory space compared to in-place quicksort.

Chapter 11: Data Validation and Verification

Contents

Syllabus Learning Outcomes	Error! Bookmark not defined.
1. Data validation and verification	2
1.1 Validation of data entry	2
1.2 Verification of data entry	3
2. Date	3
2.1 Get current date and time using datetime class	4
2.2 Print year, month, day	4
2.3 Print time	4
2.4 Get today's date using date class	4
2.5 Initiate a date object	5
2.6 Print year, month and day	5
2.7 Formatting dates	5
3. Regular Expressions	7
4. Workflow for data validation	8
4.1 Example: A program to validate age	9
5. Errors	10
5.1 Syntax errors	11
5.2 Runtime errors	11
5.3 Logic errors	11
6. Exception Handling	12
6.1 Catching specified exceptions	13
6.2 Catching exceptions	14
6.3 Try, except, finally	14
6.4 Raising an exception	15
7. Check digit	16
7.1 Method of calculating check digits	16
7.2 Example: Program to calculate the check digit for the number 1587	16
7.3 Example: Program to validate ISBN	17
Referenced Readings	19
Mrs. Yong's Video Lectures 2021	19

Syllabus Learning Outcomes

2.4 Data Validation and Program Testing

Use data validation techniques and design test cases

2.4.1 Explain the difference between data validation and data verification.

2.4.2 Understand data validation techniques such as:

- range check
- format check
- length check
- presence check
- check digit.

2.4.3 Identify, explain and correct syntax, logic, and runtime errors.

2.4.4 Design appropriate test cases using normal, abnormal and extreme data for testing and debugging programs.

1. Data validation and verification

Data integrity can never be guaranteed, but the chances are improved if appropriate measures are taken when data enters a system or when it is transmitted from one system to another.

1.1 Validation of data entry

Data validation is a process of ensuring that the input data supplied to a system satisfies a set of rules such that it is sensible, complete, and within acceptable boundaries. Its purpose is to avoid data errors. It does not guarantee that data is accurate. It will reject data entry that does not satisfy the set of rules. For example, if the entry of a name is expected but the wrong name is entered, it will still be recognised as a name and therefore accepted as valid.

Data validation techniques include

- **range check** - Checks that a value falls within the specified range, e.g. The expiry date of a card must have a month number between 1 and 12, and the date must be later than today's date.
- **format check** - Checks the data is in the right format, e.g. A date has to be dd/mm/yyyy.

- **length check** - Checks the data isn't too short or too long, e.g. A telephone number must have a length of 8 numbers.
- **presence check** - Checks that data has been entered into a field, e.g. ensure that an entry field is not left blank.
- **check digit** - The last one or two digits in a code are used to check the other digits are correct, e.g. Use digits of the code number to do a calculation to add an extra digit to the end of a code number.

1.2 Verification of data entry

Verification is the process of getting the user to confirm that the data entered was what was intended to be entered. Essentially, it is a way of preventing errors when data is copied from one medium to another. It does not check if the data makes sense or is within acceptable boundaries. It only checks that the data entered is identical to the original source.

There are two methods of verification.

Double entry is one method of verification. The process of entering data twice, with the second entry being compared with the first is to ensure that it is accurate. It is common in batch processing for a second data entry operator to key in a batch of data to verify it.

Proofreading data is another method that involves someone checking the data entered against the original document.

2. Date

One often needs to manage dates in programs, eg date of creation/update, loan/ due date

Common tasks include:

- generate the current system date
- read date from keyboard/ file
- date arithmetic
- write date to display/ file

```
# datetime module has many useful functions to manipulate date and time, so use it.
```

```
import datetime
```

2.1 Get current date and time using datetime class

```
# current system date and time
current_datetime = datetime.datetime.now()

print(current_datetime)
print(type(current_datetime))
```

```
2022-05-11 19:43:55.904043
<class 'datetime.datetime'>
```

2.2 Print year, month, day

```
print(current_datetime.year)
print(current_datetime.month)
print(current_datetime.day)
```

```
2022
5
11
```

2.3 Print time

```
print(current_datetime.hour)
print(current_datetime.minute)
print(current_datetime.second)
print(current_datetime.microsecond)
```

```
19
43
55
904043
```

2.4 Get today's date using date class

```
# current system date
```

```
today = datetime.date.today()

print(today)
print(type(today))
# current date is a datetime date object giving year, month and day eg
datetime.date(2021, 3, 15)
```

```
2022-05-11
<class 'datetime.date'>
```

2.5 Initiate a date object

```
d = datetime.date(2019, 4, 13)
print(d)
```

```
2019-04-13
```

2.6 Print year, month and day

```
print(today.day)
print(type(today.day))
print(today.month)
print(today.year)
```

```
28
<class 'int'>
3
2021
```

2.7 Formatting dates

```
# return date as string in the format YYYY-MM-DD
```

```
print(today.isoformat())

print('2020-03-26' > '2020-04-01')
```

```
2022-05-11
False
```

```
# conversion of datetime object to strings of different formats
# %d - dd
# %m - mm
# %y - yy
# %Y - YYYY
# %D - mm/dd/yy
# %A - day spelt in full
# %a - day spelt in short form
# %B - month spelt out in full
# %b - month spelt in short form
print(today.strftime('%d-%m-%y'))
print(today.strftime('%d-%m-%Y'))
print(today.strftime('%d%m%Y'))
print(today.strftime('%Y%m%d'))
print(today.strftime('%D'))
print(today.strftime('%a'))
print(today.strftime('%b'))
```

```
11-05-22
11-05-2022
11052022
20220511
05/11/22
Wed
May
```

Refer to https://www.w3schools.com/python/python_datetime.asp for more details.

```
# days of week and month
print(today.strftime('%A %d %B %Y'))
print(today.strftime('%a %d %b %Y'))
```

Wednesday 11 May 2022

Wed 11 May 2022

```
# check for valid date format
date = input("Enter date in DD-MM-YYYY: ")

#strptime() method creates a datetime object from the given string
try: # normal processing
    valid_date = datetime.datetime.strptime(date, "%d-%m-%Y") # p - parse
    print(valid_date)
    print("date ok")
except ValueError:
    # if error happens
    print("Invalid date format")
```

Enter date in DD-MM-YYYY: 12-05-2022

2022-05-12 00:00:00

date ok

3. Regular Expressions

- A powerful mechanism to check against patterns
- Used in data validation to avoid complicated (nested) if-else statements
- Refer to <https://docs.python.org/3.6/library/re.html> for more information

```
# gender validation
import re

# get gender
gender = input("Enter gender: ")

# accept both upper and lowercase f and m
pattern = re.compile("[fFmM]")

if pattern.match(gender):
    print("Cool! You are not in between")
else:
    print("Oh. What are you then?")
```

```
Enter gender: p
Oh. What are you then?
```

```
# # NRIC validation
import re

# get nric
nric = input("Enter NRIC: ")

# ^ - starts with, {n} - exactly n times, $ - ends with
pattern = re.compile("^[sStTfFgG][0-9]{7}[a-zA-Z]$")

if pattern.match(nric):
    print("Welcome to Singapore!")
else:
    print("You have entered an invalid NRIC!")
```

```
Enter NRIC: W1234)
You have entered an invalid NRIC!
```

4. Workflow for data validation

- loop to get valid data
- if user provides invalid data, prompt error message and allow re-entry
- until user provides valid data

4.1 Example: A program to validate age

Pseudocode:

```

valid_age <- false

REPEAT
    INPUT age
    IF empty input
        OUTPUT empty input message and allow re-entry
    ELIF not a number
        OUTPUT wrong data type and allow re-entry
    ELIF not within valid range
        OUTPUT invalid range and allow re-entry
    ELSE
        set valid_age to true
    END IF
UNTIL valid_age is true

```

Code:

```

# age validation
# set boolean valid_age to false
valid_age = False
# loop until valid_age is true
while not valid_age:
    # get age
    age = input("Enter age (1-100): ")

    # presence check
    # if empty input display empty input message and allow re-entry
    if len(age) == 0:
        print("Empty input. Try again.")

    # data type check
    # if not a number display wrong data type and allow re-entry
    elif not age.isdigit():
        print("Age is a number. Try again.")

    # range check
    # if not within valid range display invalid range and allow re-entry
    elif not (1 <= int(age) <= 100):
        print("Are you human? Try again.")

```



```
# else set valid_age to true
else:
    valid_age = True
print("Great job! You are", age)
```

```
Enter age (1-100):
Empty input. Try again.
Enter age (1-100): abc
Age is a number. Try again.
Enter age (1-100): 1000
Are you human? Try again.
Enter age (1-100): 30
Great job! You are 30
```

5. Errors

Errors or mistakes in a program are often referred to as bugs. The process of finding and eliminating errors is called debugging. Errors can be classified into three major groups:

- Syntax errors
- Runtime errors
- Logical errors

5.1 Syntax errors

Syntax errors are mistakes in the use of the Python language.

Some examples of Python syntax errors:

- leaving out a keyword
- putting a keyword in the wrong place
- leaving out a symbol, such as a colon, comma or brackets
- misspelling a keyword
- incorrect indentation
- empty block

5.2 Runtime errors

If a program is syntactically correct – that is, free of syntax errors – it will be run by the Python interpreter. However, the program may exit unexpectedly during execution if it encounters a runtime error – a problem that was not detected when the program was parsed but is only revealed when a particular line is executed.

Some examples of Python runtime errors:

- division by zero
- performing an operation on incompatible types
- using an identifier that has not been defined
- accessing a list element, dictionary value, or object attribute which doesn't exist
- trying to access a file that doesn't exist

5.3 Logic errors

Logical errors are the most difficult to fix. They occur when the program runs without crashing but produces an incorrect result. The error is caused by a mistake in the program's logic. You won't get an error message, because no syntax or runtime error has occurred. You will have to find the problem by reviewing all the relevant parts of the code.

Sometimes there can be absolutely nothing wrong with your Python implementation of an algorithm – the algorithm itself can be incorrect. However, more frequently these kinds of errors are caused by programmer carelessness. Here are some examples of mistakes that lead to logical errors:

- using the wrong variable name
- indenting a block to the wrong level
- using integer division (//) instead of floating-point division (/)
- getting operator precedence wrong
- making a mistake in a boolean expression
- off-by-one, and other numerical errors

6. Exception Handling

Run-time errors can occur for many reasons as stated above. These errors are called 'exceptions'. Python distinguishes between different types of exceptions, such as and not limited to

- IOError: for example, a file cannot be opened
- ImportError: Python cannot find the module
- ValueError: an argument has an inappropriate value
- KeyboardInterrupt: the user presses Ctrl+C or Ctrl+Del
- EOFError: a fileread meets an end-of-file condition
- ZeroDivisionError: a division by zero has been attempted

The exceptions can be resolved with an exception handler, rather than let the program crash.

Pseudocode:

```
TRY
    <statementsA>
EXCEPT
    <statementsB>
ENDTRY
```

6.1 Catching specified exceptions

```
try:
    result = 100/2
    print(result)
except ZeroDivisionError:
    print("No division by zero!")
```

50.0

```
# Multiple exceptions

x = 10
try:
    result = 100/0 #ZeroDivisionError
    result = int('abc') #ValueError
    y = x + "hello" #TypeError
    print(y)
except ZeroDivisionError:
    print("No division by zero!")
except ValueError:
    print("There is an error in the value")
except TypeError:
    print("There is a type error")
```

```
x = 10
try:
    result = 100/0
    result = int('abc')
    y = x + "hello"
    print(y)
except ZeroDivisionError:
    print("No division by zero!")
```

```
except (ValueError, TypeError):
    print("An error occurred")
```

6.2 Catching exceptions

```
randomList = ['a', 0, 2]

for entry in randomList:
    try:
        print("The entry is", entry)
        r = 1/int(entry)
        break
    except Exception as e:
        print("Oops!", repr(e), "occurred.") # The repr() function returns a
        printable representation of the given object.
        print("Next entry.")
        print()
print("The reciprocal of", entry, "is", r)
```

```
The entry is a
Oops! ValueError("invalid literal for int() with base 10: 'a'",) occurred.
Next entry.
```

```
The entry is 0
Oops! ZeroDivisionError('division by zero',) occurred.
Next entry.
```

```
The entry is 2
The reciprocal of 2 is 0.5
```

6.3 Try, except, finally

The finally block, if specified, will be executed regardless if the try block raises an error or not.

```
#The try block will raise an error when trying to write to a read-only file:

try:
    f = open("demofile.txt")
    f.write("Lorum Ipsum")
except Exception as e:
    print("Oops!", repr(e), "occurred.")
```

```

    print("Something went wrong when writing to the file.")
    print()
else:
    print("Closing file.")
    f.close()
finally:
    print("Closing program.")

#The program can continue, without leaving the file object open

```

```

Oops! FileNotFoundError(2, 'No such file or directory') occurred.
Something went wrong when writing to the file.

Closing program.

```

6.4 Raising an exception

We can use `raise` to throw an exception if a condition occurs. The program comes to a halt and displays the exception, offering clues about what went wrong.

```

x = 'a'

if type(x) != int:
    raise TypeError("Only integers are allowed!")

y = x + 2
print(y)

```

```

-----
TypeError                                Traceback (most recent call last)
<ipython-input-26-f77d69be6dda> in <module>()
      2
      3 if type(x) != int:
----> 4     raise TypeError("Only integers are allowed!")
      5
      6 y = x + 2

```

```

TypeError: Only integers are allowed!

```

7. Check digit

A check digit is a digit added to a string of numbers for error detection purposes. Code numbers such as customer number, employee number, or product number are often lengthy and prone to error when being keyed in. One way of preventing these errors from occurring is to add an extra digit to the end of a code number, which has been calculated from the digits of the code number. In this way the code number with its extra check digit is self-checking.

Some examples of real-life data that use check digits are

- Singapore NRIC
- Singapore car plate registration number
- book ISBN
- credit card number

7.1 Method of calculating check digits

The best-known method of calculating check digits is the modulus-11 system, which traps over 99% of all errors. The most common errors are transcription errors (a single incorrect digit being entered) or transposition errors (two digits swapped). The calculation of a check digit is shown below:

1. Each digit of the code number is assigned a 'weight'. The right-hand (least significant) digit is given a weight of 2, the next digit to the left 3, and so on.
2. Each digit is multiplied by its weight and the products are added together.
3. The sum of the products is divided by 11 and the remainder is obtained.
4. The remainder is subtracted from 11.
5. The result is divided again by 11 and the remainder is the check digit. The only exception is that if the remainder is 10 and only a single check digit is allowed, the Roman number 'X' is used as a check digit.

More details can be found here: <http://www.pgrocer.net/Cis51/mod11.html>

7.2 Example: Program to calculate the check digit for the number 1587

```
number = "1587" #let the number be a string so that we can use indexing
```

```

weight = 2 # the smallest weight starts from 2
total = 0

for i in range(len(number)):
    # multiply smallest weight starting from the right most digit
    total += weight * int(number[len(number)-1-i])
    # weight increases as we iterate the digits from right to left
    weight += 1

remainder = total % 11
result = 11 - remainder
check_digit = result % 11

if check_digit == 10:
    check_digit = 'X'

print("The check_digit of {} is {}".format(number, check_digit))

```

```
The check_digit of 1587 is 3.
```

7.3 Example: Program to validate ISBN

An ISBN (International Standard Book Number) is a 10 digit number that is used to identify a book. All books have an ISBN which has a modulus11 check digit.

The previous program can be modified to include:

- asking for a user to input
- a function to return the check digit of the ISBN number
- a boolean function to determine if a user-input ISBN number is valid.

```

def check_d(isbn):
    # the smallest weight starts from 2
    weight = 2
    total = 0

    for i in range(len(isbn)-1):
        # multiply smallest weight starting from the second right most digit,
        # as the right most digit is the check digit

```



```

        total += weight * int(isbn[len(isbn)-2-i])
        # weight increases as we iterate the digits from right to left
        weight += 1

    remainder = total % 11
    result = 11 - remainder
    check_digit = result % 11 # WHY????

    # if check_digit is 10 then replace it with X
    if check_digit == 10:
        check_digit = 'X'

    return check_digit

# compare the last digit of isbn with the check_digit found
def is_valid(last_digit, check_digit):
    return last_digit == check_digit

# main
again = True

while again:
    isbn = input("Enter an ISBN: ")

    check_digit = check_d(isbn)

    valid = is_valid(isbn[-1],str(check_digit))
    if valid:
        print("{} is valid.".format(isbn))
    else:
        print("{} is not valid.".format(isbn))

    another = input("Enter another ISBN? (Y/N) ")

    if another.lower() == 'n':
        again = False
        print("Goodbye!")

```

```

Enter an ISBN: 1858051703
1858051703 is valid.
Enter another ISBN? (Y/N) Y
Enter an ISBN: 058227544X
058227544X is valid.
Enter another ISBN? (Y/N) N

```

Goodbye!

Referenced Readings

1. Reeves, B. (2015). Aqa a Level Computer Science (UK ed.). Hodder Education.
2. Langfield, S., & Duddell, D. (2019). Cambridge International AS and A Level Computer Science Coursebook with Cambridge Elevate Edition (2 Years) (2nd ed.). Cambridge University Press.
3. Heathcote, P. M., & Heathcote, R. S. U. (2016). OCR AS and A Level Computer Science (Illustrated ed.). PG Online Limited.
4. Agarwal, A. (2021, March 16). Data Validation and Data Verification | Validation and Verification of Data. Analytics Vidhya.
<https://www.analyticsvidhya.com/blog/2021/03/data-validation-and-data-verification-from-dictionary-to-machine-learning/>
5. Validation - Data validation and verification - GCSE ICT Revision. (2022). BBC Bitesize. <https://www.bbc.co.uk/bitesize/guides/zdvrd2p/revision/1>
6. Validation. (2021, January 16). Computer Science GCSE GURU.
<https://www.computerscience.gcse.guru/theory/validation>
7. RegexOne- Learn Regular Expressions- Lesson 1: An Introduction, and the ABCs. (2022). RegexOne- Learn Regular Expressions with Simple, Interactive Exercises.
<https://regexone.com/>
8. Test plan - Development and testing - GCSE Computer Science Revision. (2022). BBC Bitesize.
<https://www.bbc.co.uk/bitesize/guides/z8n3d2p/revision/5>
<https://www.bbc.co.uk/bitesize/guides/z8n3d2p/revision/6>
<https://www.bbc.co.uk/bitesize/guides/z8n3d2p/revision/7>
9. Testing and Test Data. (2021, January 18). Computer Science GCSE GURU.
<https://www.computerscience.gcse.guru/theory/testing-and-test-data>

Mrs. Yong's Video Lectures 2021

IGCSE ICT Verification vs Validation - Theory Revision

<https://www.youtube.com/watch?v=OeHe61pT9CI>

Definitions

<https://www.youtube.com/watch?v=xI81QcwAKDA>

Datetime Module

https://www.youtube.com/watch?v=xfbTr_Fv98c

Formatting dates

<https://www.youtube.com/watch?v=R9fU3s7IShl>

Regular Expressions

<https://www.youtube.com/watch?v=G6pFK7S5f28>

Exercise on validation techniques

<https://www.youtube.com/watch?v=hT6DpoBPLLQ>

Try and Except

<https://www.youtube.com/watch?v=Zxo8gtuC8o8>

Check Digit

<https://www.youtube.com/watch?v=dz1mXBTQgpg>

Chapter 12 Object Oriented Programming (OOP)

Contents

- 1 Components of Object Oriented Programming (OOP)
- 2 Encapsulation
- 3 Inheritance
- 4 Polymorphism
- 5 Advantages to Object Oriented Programming

Syllabus Learning Outcomes

- 2.5 Fundamentals of Object-Oriented Programming
Understand concepts of encapsulation, inheritance, and polymorphism.
 - 2.5.1 Define and understand classes and objects.
 - 2.5.2 Understand encapsulation and how classes support information hiding and implementation independence.
 - 2.5.3 Understand inheritance and how it promotes software reuse.
 - 2.5.4 Understand polymorphism and how it enables code generalisation.
Exclude: method overloading and multiple inheritance

1 Components of Object Oriented Programming (OOP)

Object-oriented programming (OOP) is a form of programming that groups the data items and the subroutines that operate on the data items in one place called an **object**.

The data of an object are called **attributes** (or **properties**) and the subroutines acting on the attributes are called **methods**.

The 2 main building blocks of an object-oriented program are **classes** and **objects**.

A **class** is a blueprint that defines a related group of things. It contains attributes which describe what the data are. It also contains methods, which access and assign values to the data. A class can be represented using a class diagram. Each class is made up of 3 sections: the class name, attributes and methods.

Class
Properties
Methods

Classes are templates for objects. When a class type has been defined, it can be used to create one or more objects of this class type. Every object is an instance of a class, so it will have the same attributes and methods as the class from which it was built. Creating a new object is known as **instantiation**.

When we first set up an object of a particular class, we use a constructor. A **constructor** instantiates the object and assigns initial values to the attributes.

The class diagram below shows an example of a `Person` class.

Person
name mobile_no
constructor() get_name() get_mobile_no() set_name() set_mobile_no()

A `Person` class has 2 attributes: `name` and `mobile_no`. When we first create a `Person` object, we use a constructor method `__init()` to instantiate the object and assign values to the attributes.

```

1 class Person:
2
3     # constructor
4     def __init__(self, name, mobile_no):
5         self.name = name
6         self.mobile_no = mobile_no
7

```

The constructor method will automatically be called to initialize the newly created object.

The first input parameter for all instance methods must be `self`. The `self` parameter is referred to the current object of the class, i.e. the instance calling the method. To access any instance method or instance attribute in the class, you need to prefix it with `self`.

Let's now create a new object: John with mobile number 91234567.

```

9 # main
10 p1 = Person("John", "91234567")
11 print(p1)
12

```

```
<__main__.Person object at 0x000001CA64C5F7B8>
```

Line 10 of the code above shows a new object `p1` is created and the object is stored at a particular address. To print out useful information of the object that is readable, we can implement the `__str__()` method.

```

1 class Person:
2
3     # constructor
4     def __init__(self, name, mobile_no):
5         self.name = name
6         self.mobile_no = mobile_no
7
8     def __str__(self):
9         return "Name: {} \nMobile no: {}".format(self.name, self.mobile_no)
10
11
12 # main
13 p1 = Person("John", "91234567")
14 print(p1)
15

```

```
Name: John
Mobile no: 91234567
```

We can also write a display method to display the details of the data in the class.

```

22     def display_details(self):
23         print("Name: {} \nMobile no: {}".format(self.name, self.mobile_no))
24
25     # main
26     p1 = Person("John", "91234567")
27     p1.display_details()
28
29     p2 = Person("Mary")
30     p2.display_details()
31

```

```

Name: John
Mobile no: 91234567
Name: Mary
Mobile no:

```

Note that when you call an object's method, you do not need to give a value for the `self` parameter. Python provides it automatically.

2 Encapsulation

The idea behind OOP is that attributes can only be accessed through methods. The direct path to the data is unavailable. The attributes can only be manipulated using methods provided by the class definition. Attributes are referred to as 'private'. The methods to access the data are made available to programmers, so these are 'public'. The feature of data being combined with the subroutines acting on this data is known as **encapsulation**. In short, encapsulation is the combining of private properties and public methods into a class and ensuring private properties are only accessed/alterd by calls to public methods.

In a class diagram, negative prefix denotes 'private' and positive prefix denotes 'public'. Hence, data should be tagged with negative signs to signify that they are private attributes and methods should be tagged with positive signs to signify that they are public methods.

Person
-name -mobile_no
+constructor() +get_name() +get_mobile_no() +set_mobile_no() +set_mobile_no()

When designing a class, we need to think about the attributes we want to store and the methods we need to access the data. These methods are usually referred to as **getters**. They get an attribute of the object.

Any attributes that might be updated after instantiation will need subroutines to update their values. These are referred to as **setters**. Some attributes get set only at instantiation. These don't need setters, which means such attributes are designed not to be changed. Hence, setters are provided only for attributes values that are to be changed.

```

1 class Person:
2
3     # constructor
4     def __init__(self, name, mobile_no = ""):
5         self.name = name
6         self.mobile_no = mobile_no
7
8     # getters
9     def get_name(self):
10        return self.name
11
12    def get_mobile_no(self):
13        return self.mobile_no
14
15    # setters
16    def set_name(self, name):
17        self.name = name
18
19    def set_mobile_no(self, mobile_no):
20        self.mobile_no = mobile_no
21
22    def __str__(self):
23        return "Name: {} \nMobile no: {}".format(self.name, self.mobile_no)

```

```

25 # main
26 p1 = Person("John", "91234567")
27 print(p1)
28
29 p2 = Person("Mary")
30 print(p2)
31

```

```

Name: John
Mobile no: 91234567
Name: Mary
Mobile no:

```

Note that the default of the parameter `mobile_no` can be set as an empty string as not everyone may have a mobile number.

Back to the idea of private attributes, they should always be declared as 'Private'. One way of declaring an attribute as private is to put an underscore before an attribute name.

```

1 class Person:
2
3     # constructor
4     def __init__(self, name, mobile_no = ""):
5         self._name = name
6         self._mobile_no = mobile_no

```

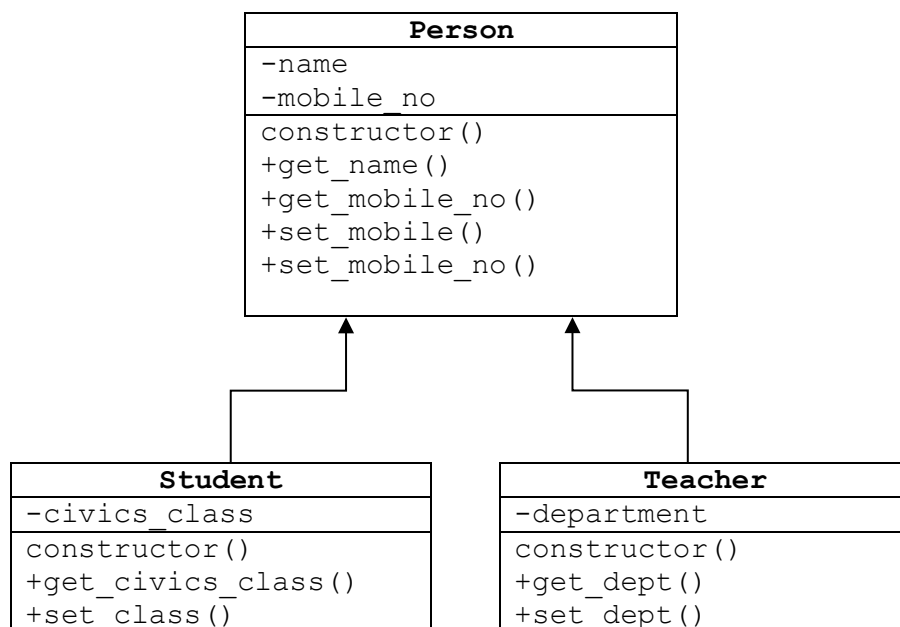

However, in Python, all the attributes are available to all users, even with an underscore prefix to the attribute names. While it does not prevent users from modifying the attributes, the underscore is to signify that the attributes are private, and it reminds the users that they would need to bear the consequences if their modifications to the attributes cause errors.

3 Inheritance

The advantage of OOP is that we can design a class (a base class or a superclass) and then derive further classes (subclasses) from this base class. This means that we write the code for the base class only once and the subclasses make use of the attributes and methods of the base class, as well as having their own attributes and methods. This is known as **inheritance**.

For example, in a school setting, there can be two types of persons, students and teachers. Both students and teachers have names and mobile numbers, however, students belong to classes while teachers belong to departments. Hence students and teachers will have common attributes of name and mobile number, and at the same time, they will have their own attributes and methods.

Below is an example of the class diagram showing superclass and subclasses.



Class diagrams with inheritance always have the superclass (base class) at the top, and the subclass(es) shown beneath. The direction of inheritance is always from subclass(es) to superclass.

The code below shows how subclasses are declared in Python. The `super()` function allows us to call a method from the parent class, in this case, the `__init__()` method.

```

1  class Student(Person):
2
3      def __init__(self, name, mobile_no, c_class):
4          super().__init__(name, mobile_no)
5          self.civics_class = c_class
6
7      def get_class(self):
8          return self.civics_class
9
10     def set_class(self, c_class):
11         self.civics_class = c_class
12
13
14  class Teacher(Person):
15
16     def __init__(self, name, mobile_no, dept):
17         Person.__init__(self, name, mobile_no)
18         self.dept = dept
19
20     def get_dept(self):
21         return self.dept
22
23     def set_dept(self, dept):
24         self.dept = dept

```

```

1  s1 = Student("Peter", "91111111", "22S57")
2  s1.display_details()
3
4  t1 = Teacher("Mrs Yong", "92222222", "Computing")
5  t1.display_details()

```

```

Name: Peter
Mobile no: 91111111
Name: Mrs Yong
Mobile no: 92222222

```

4 Polymorphism

All attributes and methods are inherited from the parent class to the subclasses. The same methods can be redefined and used in different ways, depending on the data in the subclasses. Although the method in the subclass will have the same identifier as in the superclass, the method will actually behave differently. This is known as **polymorphism**.

Looking at the code snippets above, the constructor of the superclass is redefined in the subclasses. The constructors in the subclasses call the constructor of the superclass and initializes the attributes in the superclass. This is also known as method overriding.

The `display_details()` method is currently only defined in the superclass. This means that we can only get information on the attributes that are part of the superclass. To include the additional attributes from the subclass, we need to declare the method again.

```

1 class Student(Person):
2
3     def __init__(self, name, mobile_no, c_class):
4         super().__init__(name, mobile_no)
5         self.civics_class = c_class
6
7     def get_class(self):
8         return self.civics_class
9
10    def set_class(self, c_class):
11        self.civics_class = c_class
12
13    def display_details(self):
14        print("Name: {} \nMobile no: {} \nCivics class: {}".format(self.name, self.mobile_no, self.civics_class))
15
16
17 class Teacher(Person):
18
19     def __init__(self, name, mobile_no, dept):
20         Person.__init__(self, name, mobile_no)
21         self.dept = dept
22
23     def get_dept(self):
24         return self.dept
25
26     def set_dept(self, dept):
27         self.dept = dept
28
29     def display_details(self):
30         print("Name: {} \nMobile no: {} \nDepartment: {}".format(self.name, self.mobile_no, self.dept))

```

The code above shows that the subclasses have overridden the method `display_details()` in the super class. The `display_details()` method in the subclasses now displays the new attributes of the respective subclasses.

```

1 s1 = Student("Peter", "91111111", "22S57")
2 s1.display_details()
3
4 t1 = Teacher("Mrs Yong", "92222222", "Computing")
5 t1.display_details()

```

```

Name: Peter
Mobile no: 91111111
Civics class: 22S57
Name: Mrs Yong
Mobile no: 92222222
Department: Computing

```

5 Advantages to Object Oriented Programming

There are several advantages to implementing OOP.

- Improve code readability: Program structure is short and concise. OOP allows breaking the program into bite-sized problems that can be solved easily (one class at a time).
- Improve code reliability: Avoid code duplication and easier to debug.
- Reuse quality code: Reuse existing code which is already tested.
- OOP systems can be easily upgraded from small to large systems.

Chapter 13 Linked Lists

Contents

- 1 Abstract Data Type (ADT)
- 2 Linked Lists
- 3 Operations of Singly Linked Lists using OOP
 - 3.1 Creating a new linked list
 - 3.2 Inserting a node
 - 3.3 Deleting a node
 - 3.4 Search for data
 - 3.5 Access all nodes stored in the Linked List
- 4 Linked list using arrays
 - 4.1 Inserting a node
 - 4.2 Deleting a node
- 5 Free space list

Syllabus Learning Outcomes

- 1.3 Data Structures

Understand concept and write algorithms for stack, queue (linear and circular), linear linked list and binary search tree.

 - 1.3.1 Understand the concept of static allocation of memory.
 - 1.3.2 Understand the concept of dynamic allocation of memory.
 - 1.3.4 Understand the concept of free space list (which could be another linked list or an array).
 - 1.3.5 Create, update (edit, insert, delete) and search operations for a linear linked list.

Exclude: doubly-linked list and circular linked list
- 2.3 Implementing Algorithms and Data Structures

Use programming language elements and constructs to implement sort and search algorithms such as insertion sort, bubble sort, quicksort, merge sort, linear search, binary search and hash table search, as well as data structures such as stacks, queues, linear linked lists and binary search trees.

 - 2.3.3 Write programs to implement operations for stacks, queues (linear and circular), linear linked lists and binary search trees.

Exclude: doubly-linked list and circular linked list

1 Abstract Data Type (ADT)

An **Abstract Data Type (ADT)** is a collection of data and a set of associated operations:

- create a new instance of the data structure
- insert a new element into the data structure
- delete an element from the data structure
- find/update an element in the data structure
- access all elements stored in the data structure in a systematic manner.

One can use an ADT's operations without knowing how the operations are implemented or how the data is stored. Examples of ADTs: Linked lists, stacks, queues and binary trees.

2 Linked Lists

An array is defined as a collection of items that are stored at contiguous (adjacent) memory locations. It is a container which can hold a fixed number of items, and these items should be of the same type.

A linked list is a dynamic data structure, which holds a collection of elements. The individual element may not be stored in contiguous memory locations but at whatever location that is available, and the elements are linked into an ordered sequence using pointers.

An element in the list is called a **node**. A node contains both data and a "link" to the next element. The "link" is usually referred to as a **pointer**, which is a variable that stores the address of the next node it points to.

A **pointer** that does not point at anything, indicating no further elements, is called a **null pointer**. It is usually represented by $\emptyset$. In program code, we set it to **None**. The **start/head** pointer stores the address of the first element (a link that points to the first node).

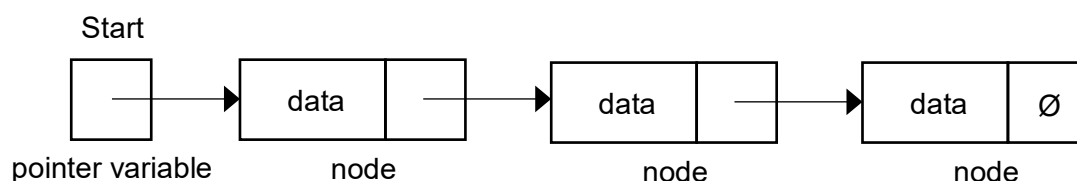
There are different types of linked list: singly linked list, doubly linked list and circular linked list.

A **singly linked list** is a collection of nodes, each containing one pointer pointing to the next node. The `start` pointer points to the first node, and the `pointer` in the last node points to null. When the list is empty, the `start` pointer points to null. With only one pointer, it can only traverse forward along the list.

A **doubly linked list** is a collection of nodes, each containing two pointers, one pointing to the previous node and the other pointing to the next node respectively. With two pointers, we can traverse forward and backward along the list. (Not in syllabus)

A **circular linked list** is similar to a singly linked list, but with the pointer in the last node pointing back to the first node. (Not in syllabus)

For the remaining of this chapter, we will focus on singly linked list.

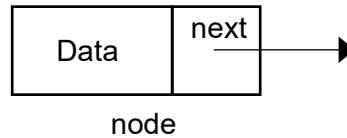


Conceptual diagram of a singly linked list

3. Operations of Singly Linked Lists using OOP

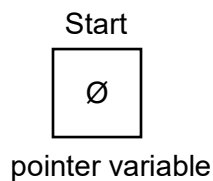
3.1 Creating a new linked list

In the examples which follow, we will assume each node consists of a `data` and a `next` pointer, pointing to the next node in the list.



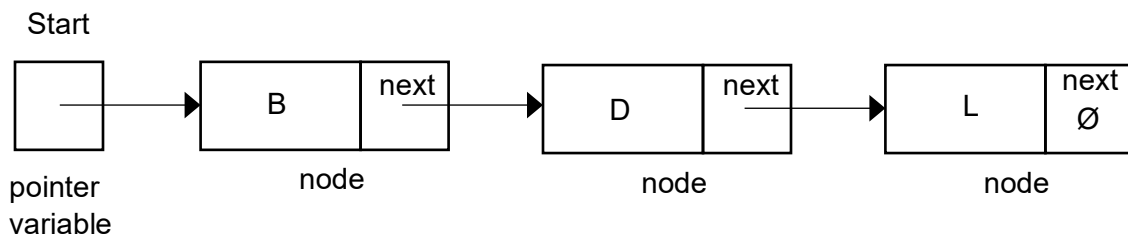
A class `Node` contains two attributes: `data` and a `next` pointer.

```
1 class Node:
2
3     def __init__(self, data):
4         """ constructor to initialize new node """
5         self.data = data
6         self.next = None
```



A class `LinkedList` contains one attribute: `start`. The constructor will initialize an empty linked list and set the `start` pointer to null.

```
11 class LinkedList:
12     """ collection of Node objects """
13     def __init__(self):
14         """ constructor to initialize empty linked list """
15         self.start = None
```

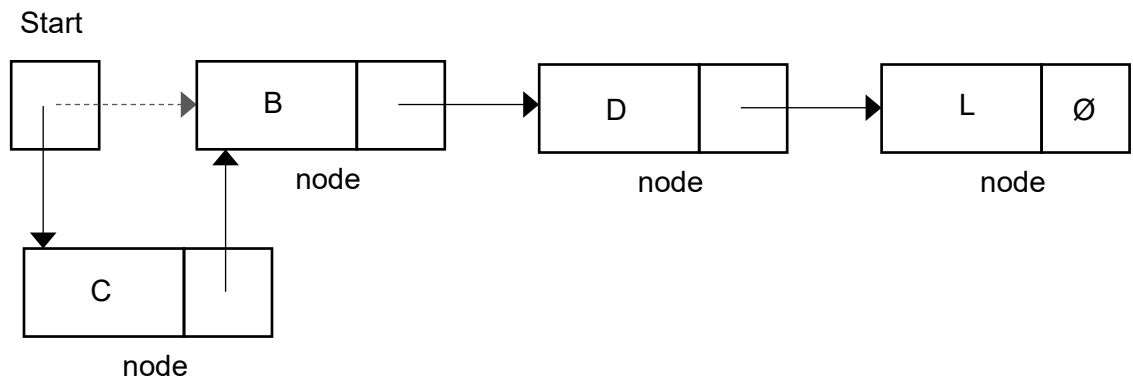


Conceptual diagram of a linked list

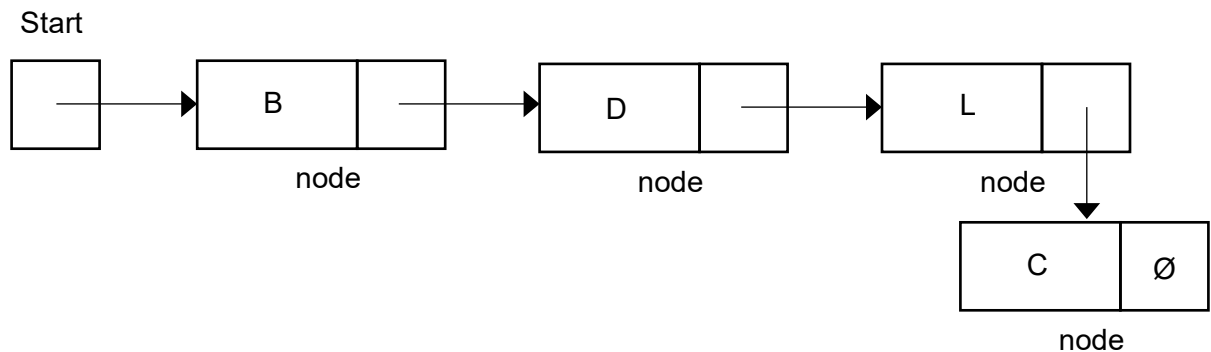
3.2 Inserting a node

A linked list can be either an ordered or unordered linked list. For an unordered linked list, a new node can be inserted either at the beginning of the list or at the end of the list.

To insert a new node, C, at the beginning of the list, the pointer in the new node, C, has to point to the content of the *Start* pointer and the *Start* pointer is then set to point to the new node, C.

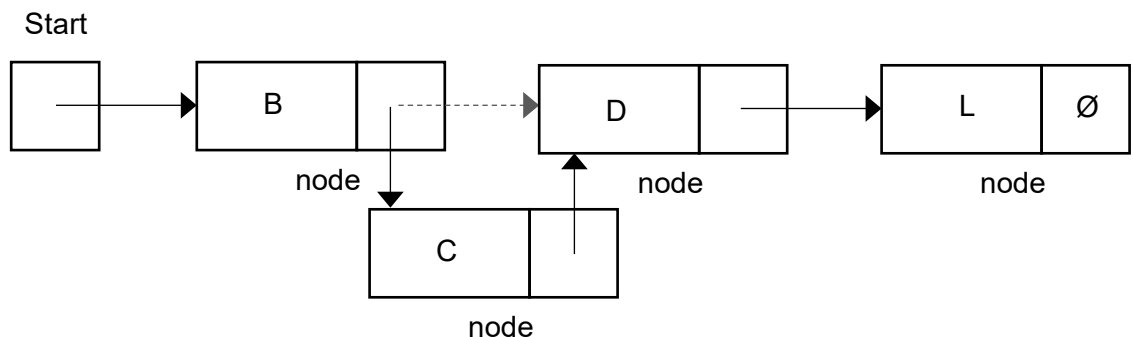


To insert the new note, C, at the end of the list, the pointer of node L points to the new node, C. The pointer of the new node, C, contains the null pointer.



To insert a new node into an ordered linked list, the new node may need to be inserted between two existing nodes.

To insert a new node, C, between existing nodes, B and D, the pointer of node C has to point to the content of the pointer in B and the pointer in B is then set to point to the new node, C.



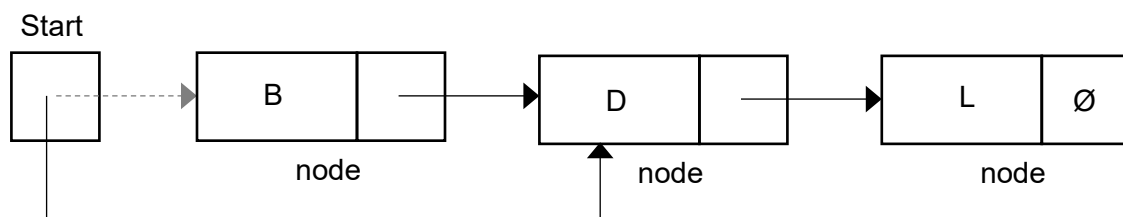
The algorithm for inserting a node into an ordered linked list:

```
PROCEDURE insert(self, data)
  //Create a node object
  new_node ← Node(data)

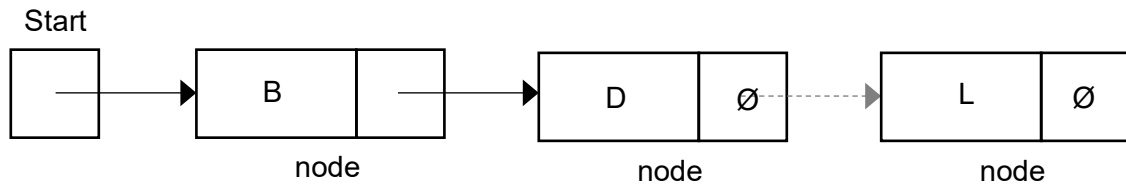
  IF self.start IS NONE //Linked list is empty
    THEN
      self.start ← new_node //insert new_node as first node
    ELSE //Find insertion point
      current ← self.start //start at beginning of list
      previous ← current
      WHILE current IS NOT NONE AND current.data < data
        previous ← current //remember current node
        current ← current.next //follow the pointer to the next node
      ENDWHILE
      IF previous == current //insert new_node at the start of list
        THEN
          new_node.next ← current
          self.start ← new_node
        ELSE
          IF current IS NONE //insert new_node at the end of list
            THEN
              previous.next ← new_node
            ELSE //insert new_node between previous and current
              new_node.next ← current
              previous.next ← new_node
            ENDIF
          ENDIF
        ENDIF
      ENDIF
    ENDPROCEDURE
```

3.3 Deleting a node

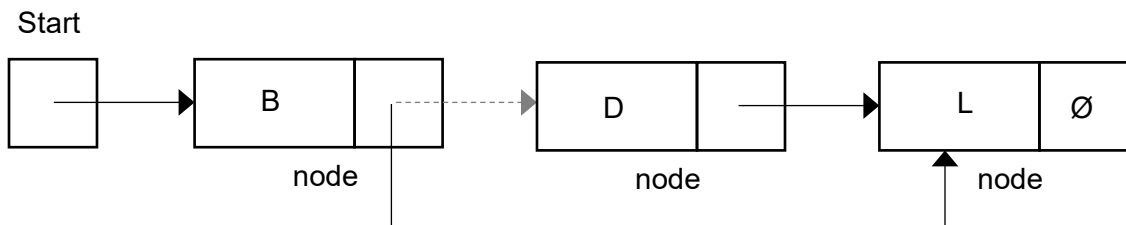
To delete the first node in the list, the `Start` pointer is set to point to the content of the pointer of the first node.



To delete the last node in the list, we set the pointer of the node before the last node to be null pointer.



To delete a node, D, within the list, we set the pointer of node B to point to the content of the pointer in D.



The algorithm for deleting a node from an ordered linked list:

```

PROCEDURE delete(self, data)
  IF self.start IS NONE
  THEN
    OUTPUT "Empty linked list."
  ELSE
    current ← self.start
    previous ← current
    WHILE current IS NOT NONE AND current.data != data
      previous ← current
      current ← current.next
    ENDWHILE
    IF current IS NONE //if data does not exist
    THEN
      OUTPUT "Data does not exist!"
    ELSE //data exists in current node
      IF current == self.start //If data exists in first node
      THEN
        self.start ← current.next //delete first node
      ELSE
        previous.next ← current.next //delete current node
      ENDIF
    ENDIF
  ENDIF
ENDPROCEDURE

```

3.4 Search for data

With the `next` pointer, we can traverse down a list to search for data.

The algorithm for search in an ordered linked list:

```
FUNCTION search(self, data)
  IF self.start IS NONE //Empty linked list
    THEN
      OUTPUT "Empty linked list."
    ELSE
      current ← self.start
      WHILE current IS NOT NONE AND current.data < data
        current ← current.next
      ENDWHILE
      IF current is not None and current.data = data
        THEN
          OUTPUT "Data found!"
          RETURN TRUE
        ELSE
          OUTPUT "Data not found!"
          RETURN FALSE
      ENDIF
    ENDIF
  ENDFUNCTION
```

3.5 Access all nodes stored in the Linked List

```
PROCEDURE display(self)
  IF self.start IS NONE
    THEN
      OUTPUT "Empty Linked List"
    ELSE
      current ← self.start
      WHILE current IS NOT NONE
        OUTPUT current.data
        current ← current.next
      ENDWHILE
    ENDIF
  ENDPROCEDURE
```

4. Linked list using arrays

One reason why linked list is preferable to array is that only pointers need changing in a linked list when reordering is required, however, all data items would need to be shifted for insertions or deletions at interior positions of an array. Using linked lists saves time, however, we need more storage space for the pointer fields. With the extra storage space for pointers, we can implement linked lists using arrays without the expensive shifting of elements within the arrays.

Linked list can be implemented by having an array of nodes. Each node contains data and a pointer. The pointers then link up the nodes by storing the index of the next node in the array.

For the follow examples, we will set the null pointer to be -1.

An empty linked list is initialised by setting the `start` pointer to -1.



Diagram shows an empty linked list

New nodes added into the linked list are stored in contiguous locations, however, the pointers link up the nodes in an ordered sequence.

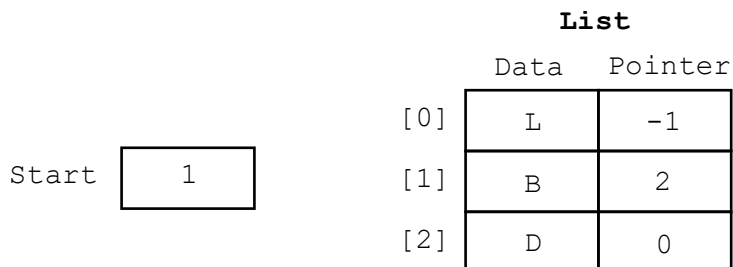
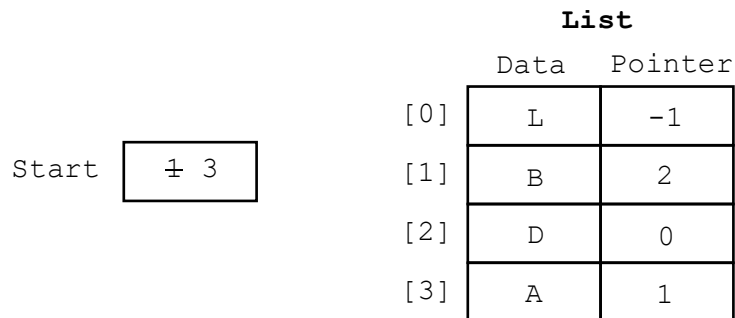


Diagram shows an ordered linked list in an array

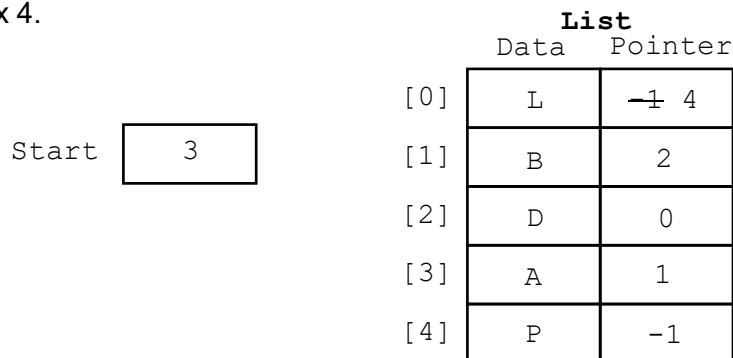
4.1 Inserting a node

To insert a new node, A, to the beginning of the linked list, it is first added at index 3. Following which, pointer of A is set to point to the content of the `start` pointer and the `start` pointer is adjusted to point to the new node A at index 3.



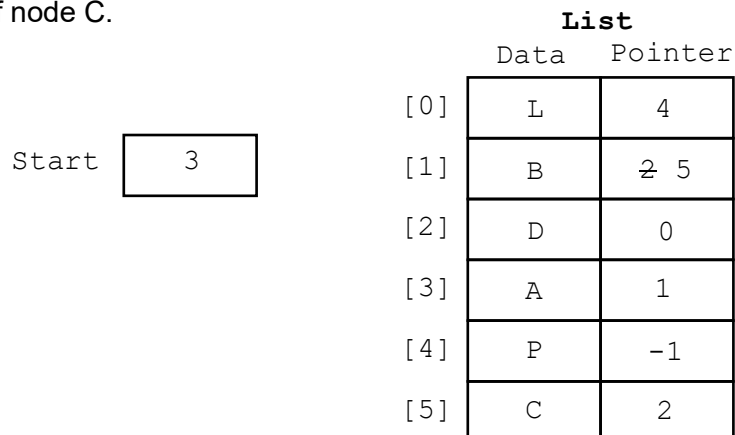
Adding a new node to the beginning of a linked list

To insert a new node, P, at the end of a linked list, it is first added at index 4. The node that previously contained the null pointer (-1) has its pointer adjusted to point to the new node at index 4.



Adding a new node to the end of a linked list

To insert a node, C, into the list, it is first added at index 5. The pointer of C will then point to the index that the pointer of node B is pointing at, and the pointer of node B will point to the index of node C.

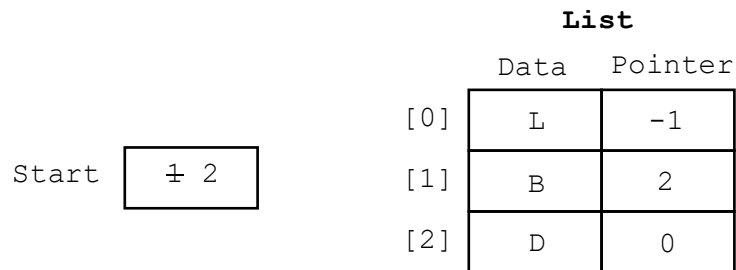


Adding a new node in between 2 nodes

4.2 Deleting a node

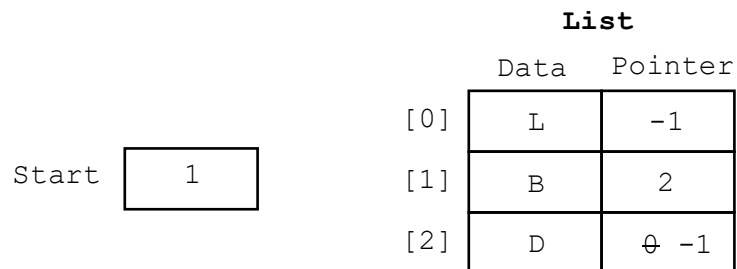
When deleting a node, only pointers need to be adjusted. The old data can remain in the array, but it will no longer be accessible as no pointer will point to it.

The diagram below shows how the `start` pointer is adjusted to delete the first node of the linked list. The `start` pointer now contains the pointer value of the deleted node.



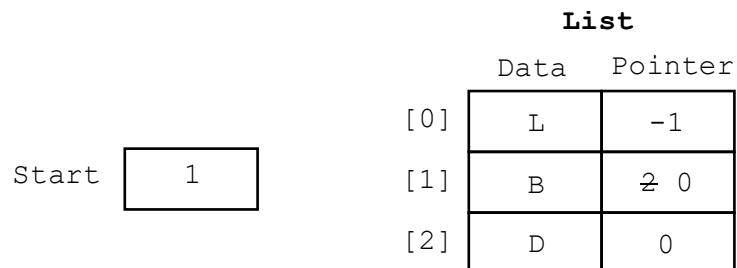
Deleting the first node of the linked list

To delete the last node in the linked list, the pointer of the node before the last node is set to -1.



Deleting the last node of the linked list

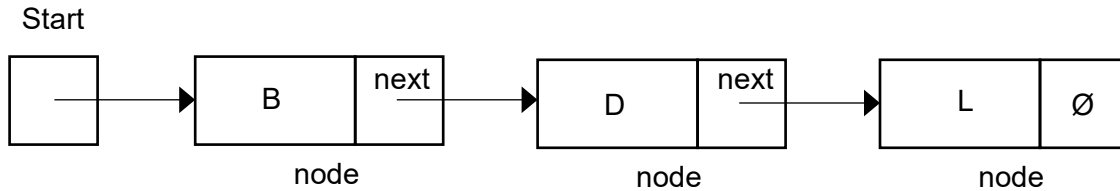
To delete a node D that is between 2 nodes, the pointer of B will be set to point to the content of the pointer in D.



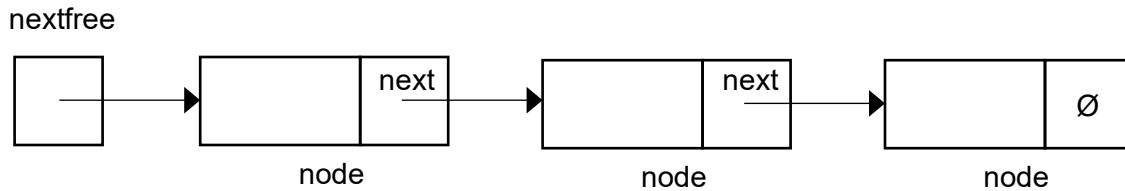
Deleting a node within the linked list

5. Free space list

As we can see from section 4.2, Nodes that are deleted will remain in the array and no longer be accessible as no pointer will point to it. This space, which is free, will then be wasted. One way of managing the free space is to keep two linked lists: one for the actual data, and one for the free space. A free space list can be created by linking unused nodes (empty nodes initially). In this manner, new nodes can be created using the unused nodes and nodes that are deleted will be returned to the free space list and can be reused again.



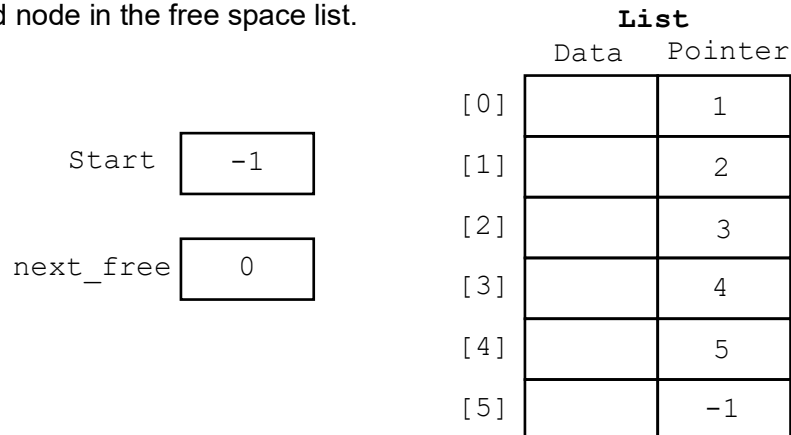
Conceptual diagram of a linked list with data with a `start` pointer



Conceptual diagram of a free space list with a `next_free` pointer

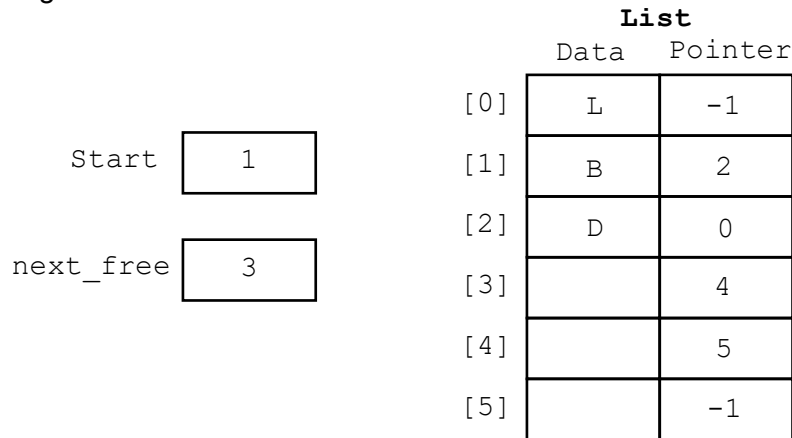
For the follow examples, we will implement the free space list using arrays. We will need to create an array of empty nodes. Data in empty nodes are set to null or empty strings. The empty nodes are then linked up using the pointers.

When an array of empty nodes is first initialised to work as a linked list, the linked list is considered to be empty, with the `start` pointer set as the null pointer, -1. All unused nodes need to be linked to form the free space list. The `next_free` pointer will point to the first unused node in the free space list.

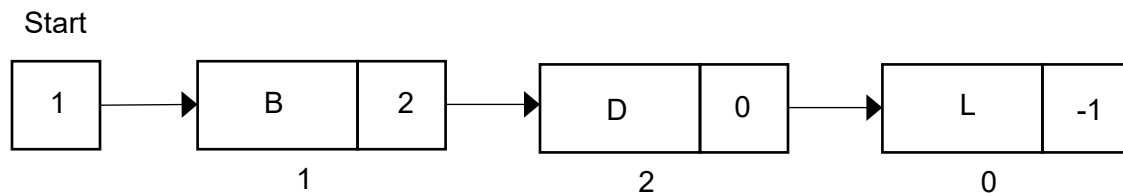


An empty linked list and a free space list of 6 linked unused nodes

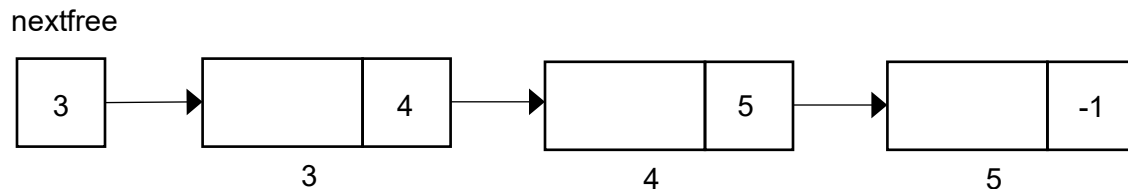
Assume data “L”, “B” and “D” were added to the linked list and be kept in alphabetical order. The diagram below shows how a linked list with data and a free space list of unused nodes are being maintained.



Linked list with data and free space list



Conceptual diagram of a linked list with data with a `start` pointer

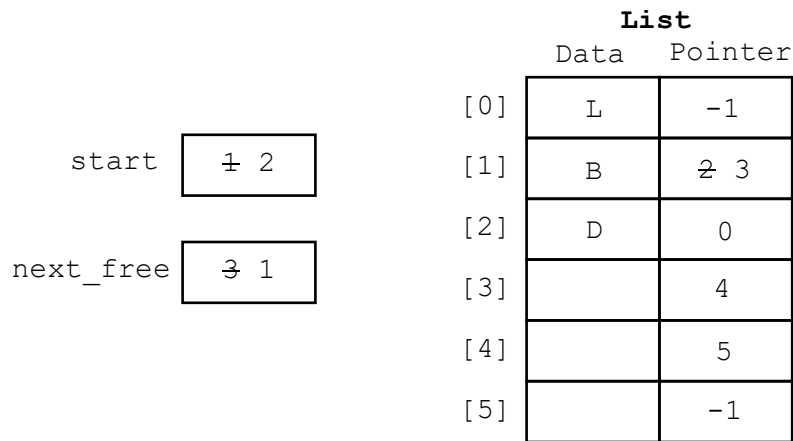


Conceptual diagram of a free space list with a `next_free` pointer

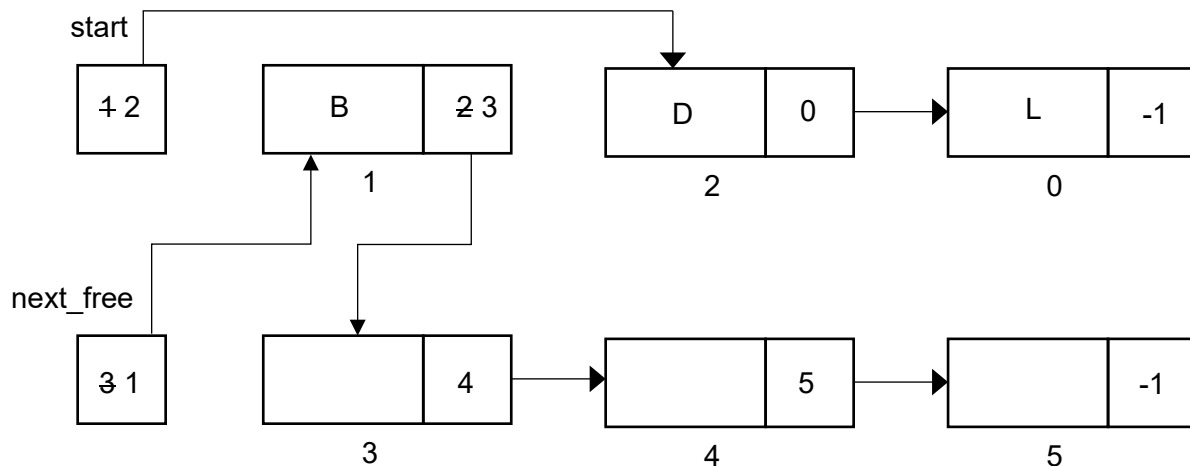
To insert data, we need to:

1. Check if there is any unused nodes in the free space list
2. Set a `new_node_pointer` to point to the index that the `next_free` pointer is pointing to.
3. Store the required data into the new node at the index that `new_node_pointer` is pointing to.
4. Set `next_free` pointer to point to index of next free node
5. Traverse the linked list find the correct position to insert new node.
6. To insert new node to the start of linked list, set the `next` pointer of new node to point to the index that `start` pointer is pointing to and set `start` pointer to point to index of new node.
7. To insert new node within the linked list, set the `next` pointer of new node to point to the index of the next node and set the pointer of the previous node to point to the index of new node.

If node B is to be deleted, the array element of that node needs to be linked back into the free space list. This space, at index 1, will now be accessible and can be reused again.



Linked list and free space list implemented using arrays



Conceptual diagram of deleting a node from linked list and adding the deleted node back to free space list as an unused node.

To delete node, we need to

1. Check if linked list is empty
2. Traverse the linked list find if data exists in the nodes
3. To delete the first node, set the `start` pointer to point to the index of the `next` pointer in the first node. Set the index of the `next` pointer in the first node to point to the index of the next free node and set the `next_free` pointer to point to the index of the node to be deleted.
4. To delete a node within the linked list, set the pointer of the previous node to point to the index of the next pointer in the node to be deleted. Set the `next` pointer in the node to be deleted to point to the index of next free node and set the `next_free` pointer to point at index of deleted node.

Chapter 15 Stack and Queue

Contents

- 1 Stack
 - 1.1 Stack implementation
- 2 Queue
 - 2.1 Linear queue implementation
 - 2.2 Circular queue implementation

Syllabus Learning Outcomes

1.3 Data Structures

Understand concept and write algorithms for stack, queue (linear and circular), linear linked list and binary search tree.

1.3.3 Create, insert, and delete operations for stack and queue (linear and circular).

2.3 Implementing Algorithms and Data Structures

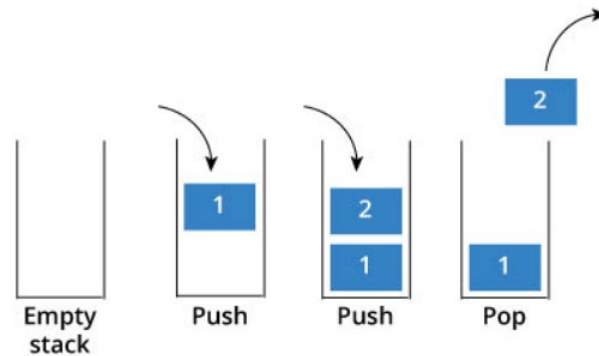
Use programming language elements and constructs to implement sort and search algorithms such as insertion sort, bubble sort, quicksort, merge sort, linear search, binary search and hash table search, as well as data structures such as stacks, queues, linear linked lists and binary search trees.

2.3.3 Write programs to implement operations for stacks, queues (linear and circular), linear linked lists and binary search trees.

1. Stack

Stack is an abstract data type (ADT) which inserts and removes items according to the **Last-In-First-Out (LIFO)** principle. A user may insert items into a stack at any time, but may only access or remove the last item inserted.

The name “stack” is derived from the metaphor of a stack of plates in a spring-loaded, cafeteria plate dispenser. In this case, the fundamental operations involve the “pushing” and “popping” of plates on the stack. When we need a new plate from the dispenser, we “pop” the top plate off the stack, and when we add a plate, we “push” it down on the stack to become the new top plate. The same terms are also used in the stack ADT.



Stacks are used in many applications such as the Internet Web browsers storing the addresses of recently visited sites, reversing data as well as matching opening and closing symbols.

1.1 Stack implementation

Stack can be implemented using either array or linked list.

The basic operations of a stack is to

- `push()`: add an item to the top of the stack
- `pop()`: remove and return an item from the top of the stack

Other supporting operations to be added are

- `is_empty()`: check if stack is empty
- `size()`: return number of items in the stack
- `peek()`: return the top item without removing it
- `display()`: show all items in the stack

2. Queue

Queue is another ADT which is a collection of items that are inserted and removed according to the **First-In-First-Out (FIFO)** principle. That is, items can be inserted at any time, but only the item that has been in the queue the longest can be next removed.

We usually say that items enter a queue at the rear and are removed from the front. A metaphor for this terminology is a line of people waiting to get on an amusement park ride. People waiting for such a ride enter at the back of the line and get on the ride from the front of the line. There are many other applications of queues. Stores, theatres, reservation centres, and other similar services typically process customer requests according to the **FIFO** principle. A queue would therefore be a logical choice for a data structure to handle calls to a customer service centre, or a wait-list at a restaurant. Many computing devices, such as a networked printer, or a Web server responding to requests, also use queues.



2.1 Linear queue implementation

Queue can be implemented using either array or linked list.

The basic operations of a queue is to

- enqueue(): add an item to the rear of the queue
- dequeue(): remove an item from the front of the queue

Other supporting operations to be added are

- is_empty(): check if queue is empty
- size(): return number of items in the queue
- peek(): return the next item to be removed without removing it
- display(): show all items in the queue

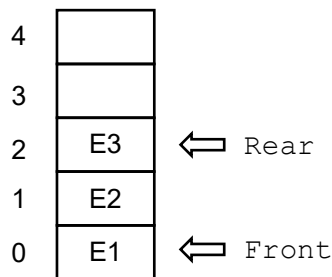
2.2 Circular queue implementation

For the stack ADT, we created a simple mapping to Python list methods. It may be very tempting to use a similar approach for supporting the queue ADT. We could enqueue element e by calling `append(e)` to add it to the end of the list. We could use the syntax `pop(0)`, as opposed to `pop()`, to intentionally remove the first element from the list when dequeuing.

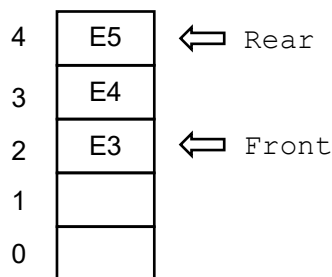
As easy as this would be to implement, it is tragically inefficient. When `pop` is called on a list with a non-default index, a loop is executed to shift all elements beyond the specified index to the left, so as to fill the hole in the sequence caused by the `pop`. Therefore, a call to `pop(0)` always causes the worst-case behaviour of $O(n)$ time.

A method to overcome the problem is to implement the queue as a circular queue, a queue with wrap-around. It can be implemented with a pointer, `Front`, pointing to the front of the queue and a pointer, `Rear`, pointing to the back of the queue. A variable `MaxQueueSize`, holding the maximum capacity of the queue, is needed, and it is useful to have an additional variable `NumberInQueue`, giving the number of elements already in the queue.

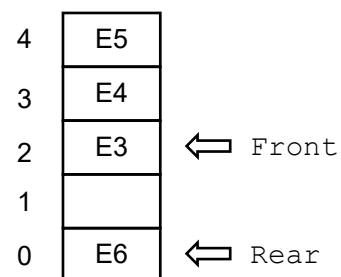
Illustration of a circular queue:



i) inserted 3 elements in the order of E1, E2 and E3 in a Queue



ii) the Queue with some elements inserted (E4 and E5) and removed (E1 and E2)



iii) after inserting E6, the rear arrow is wrapped around and is now below the front arrow

Take note that both stacks and queues can be implemented using either array or linked list, whereby both Python list and linked list allow for dynamic memory allocation of its data elements. However, we can also implement stacks and queues having a fixed size, such that we can implement a fixed size array, which allows the static memory allocation of its data elements. In the case of the circular queue implementation, it follows the static memory allocation of its data elements.

Chapter 15: Binary Search Tree

Contents

Contents	1
Syllabus Learning Outcomes	1
1. Trees	2
1.1 Binary Tree	3
1.1.1 Implementation of nodes	3
1.2 Binary Search Tree (OOP Implementation)	4
1.2.1 Insert Operation	4
1.2.1.1 Insertion Walkthrough	5
1.2.1.2 Implementation of insertion	5
1.2.2 Search Operation	6
1.2.2.1 Implementation of Search	6
1.2.3 Deletion Operation [No coding in the syllabus]	7
1.2.3.1 Deletion Walkthrough	10
1.2.4 Update Operation [No coding in the syllabus]	11
1.3 Binary Search Tree (Array Implementation)	12
1.4 Time Complexity of BST for Search Operation	13
1.4.1 Comparison of BST, Array, and Linked List	14
1.5 Tree Traversals	Error! Bookmark not defined.
1.5.1 Pre-order Traversal	Error! Bookmark not defined.
Readings	14

Syllabus Learning Outcomes

1.3 Data Structures

Understand concept and write algorithms for binary search tree.

1.3.6 Create, update (edit, insert, delete*) and search operations for a binary search tree. **Exclude: deletion of nodes from binary search tree*

1.3.7 Understand pre-order, in-order and post-order tree traversals; and application of in-order tree traversal for binary search tree.

2.3 Implementing Algorithms and Data Structures

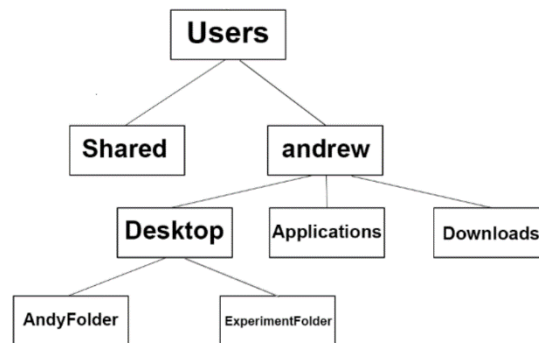
Use programming language elements and constructs to implement binary search trees.

2.3.3 Write programs to implement operations for binary search trees.

1. Trees

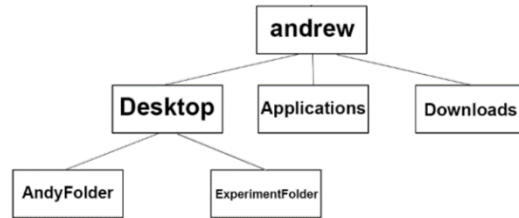
A tree is an abstract data type like linked lists, stacks and queues. However, it has a non linear structure, unlike the other data types. This means that each node can have more than 1 child node (analogous to the next node in the linear data structures).

An example of a tree is the directory structure in computers:



In the example above:

- Each directory is called a node
 - **Users** at the top or start of the structure is called the root node.
- The nodes next down in the structure are known as child nodes.
 - So, **Users** is also a parent node. It has 2 child nodes: **Shared** and **andrew**.
 - **andrew** is in turn the parent node for 3 child nodes: **Desktop**, **Applications** and **Downloads**.
 - **Desktop** is in turn a parent node for **AndyFolder** and **ExperimentFolder**.
- The lines that join 2 nodes are called branches. In some diagrams, arrows may be used to represent branches, and they will point from the parent node to the child node.
- Notice that **shared**, **AndyFolder** and **ExperimentFolder** have no child nodes. They are also called leaf nodes.
- **andrew** together with all the nodes below it will form a sub-tree as shown below.



1.1 Binary Tree

Binary trees are a special type of tree. Each node in a binary tree is allowed to have 0, 1 or 2 child nodes. Since there are only 2 child nodes, they are usually known as the left node (or left child) and the right node (or right child).

In its implementation, typically by OOP, a node object will have the attributes:

- data
- left pointer (points to the left child)
- right pointer (points to the right child)

1.1.1 Implementation of nodes

Open BST.ipynb.

Google Colab Notebook Link:

<https://drive.google.com/file/d/1fhkFAOY90sF5cYMLOXoqQjwxoSuISC6n/view?usp=sharing>

Implement the class Node that takes in parameters data, left and right. The default values for left and right are **None**.

We will also implement the usual setters and getters to implement encapsulation.

We will define `__str__()` method as follows:

```
def __str__(self):
    return f"{self.data}({self.left.get_data() if self.left else ''},{self.right.get_data() if self.right else ''})"
```

This `__str__()` method allows us to see the data values of the node and the child nodes, if any.

Test the class with the following code:

```
n1 = Node(2, right = Node(10))  
print(n1)
```

Output:

```
2(,10)
```

1.2 Binary Search Tree (OOP Implementation)

The binary search tree (BST) is a special type of binary tree. It automatically sorts the data as nodes are added.

1.2.1 Insert Operation

The following pseudocode explains the insert operation:

```
IF root is NONE  
  THEN  
    root ← Node(value)  
  ELSE  
    parent ← root  
  
    WHILE value not inserted DO  
      IF value < parent's value // branch left  
        THEN  
          IF left branch is empty  
            THEN  
              left node ← Node(value)  
            ELSE  
              parent ← left node  
          ENDIF  
        ELSE // branch right, we assume no duplicate values  
          IF right branch is empty  
            THEN
```

```

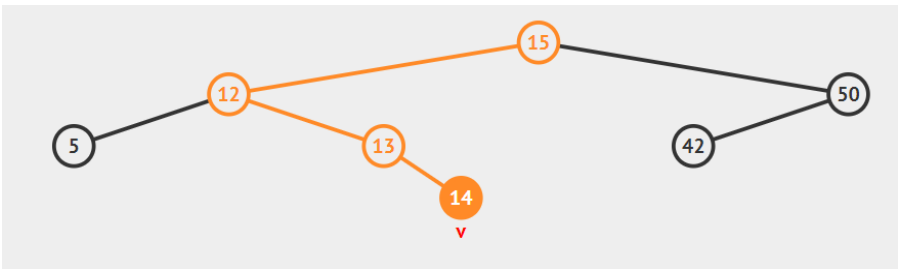
        right node ← Node(value)
    ELSE
        parent ← right node
    ENDIF

ENDIF
ENDWHILE
ENDIF

```

1.2.1.1 Insertion Walkthrough

Example 1: Use the algorithm described to add the following data values.

Values to be added	Binary Search Tree
1. 15 2. 12 3. 5 4. 50 5. 42 6. 13 7. 14	

Example 2: Use the algorithm described to add the following data values.

Values to be added	Binary Search Tree
1. 5 2. 12 3. 13 4. 15 5. 42 6. 50 7. 14	

1.2.1.2 Implementation of insertion

Part of the BST has been coded for you. Implement the `insert` method according to the pseudocode above.

Google Colab Notebook link:

<https://drive.google.com/file/d/1fhkFAOY90sF5cYMLOXoqQjwxoSuISC6n/view?usp=sharing>

Test the `insert` method with the following code:

```
t2 = BST()
t2.show()
vals = [15, 12, 5, 50, 42, 13, 14]

for val in vals:
    t2.insert(val)
t2.show()
```

1.2.2 Search Operation

Besides having a method to insert nodes, the class should also have a method to search for nodes. The process of searching is similar to what we did during the binary search. However, instead of starting the search with the middle item in a sorted array, we will start from the root node.

The search method can be implemented using the following algorithm:

```
CurrentNode ← RootNode
REPEAT
    IF value < CurrentNodeValue
        THEN
            CurrentNode ← LeftChildNode
        ELSE
            CurrentNode ← RightChildNode
    ENDIF

UNTIL CurrentNodeValue EQUALS value OR
      CurrentNode has no corresponding child node
```

1.2.2.1 Implementation of Search

Implement the `search` method according to the algorithm above. For this task, we will want to return the node if it is found, else return `None`.

Test the search method with the following code:

```
for val in range(12,17):
    print(t2.search(val))
```

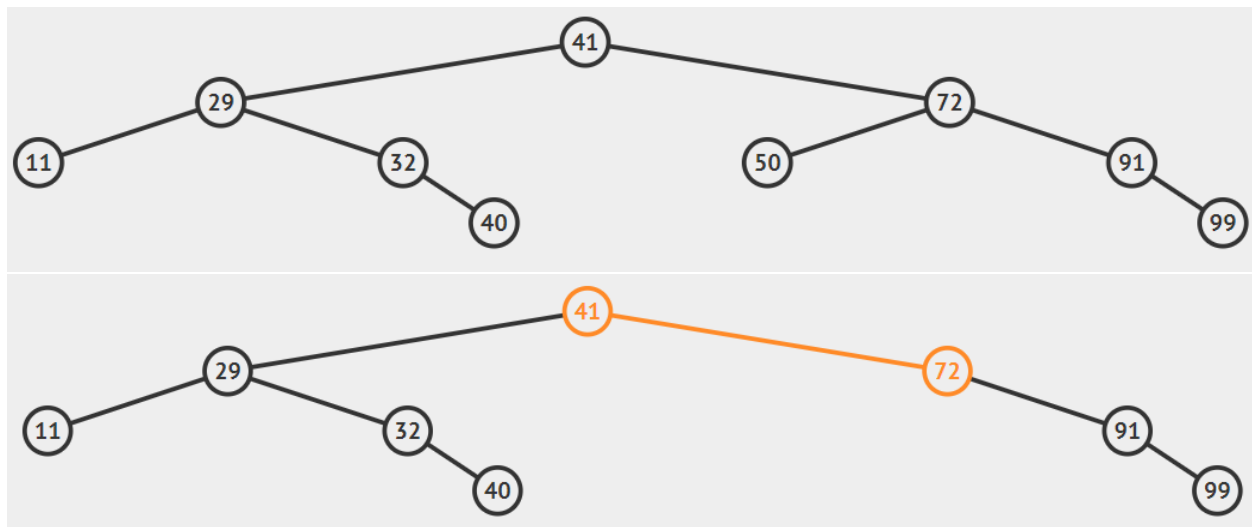
1.2.3 Deletion Operation [No coding in the syllabus]

For the removal of nodes from the tree, there are 3 cases to consider, and is dependent on how many child nodes the removed node has.

Consider the cases below.

Case 1: No child nodes

Suppose we want to remove a leaf node, say `Node(50)`. We can simply remove `Node(50)` without any extra steps.



Original:

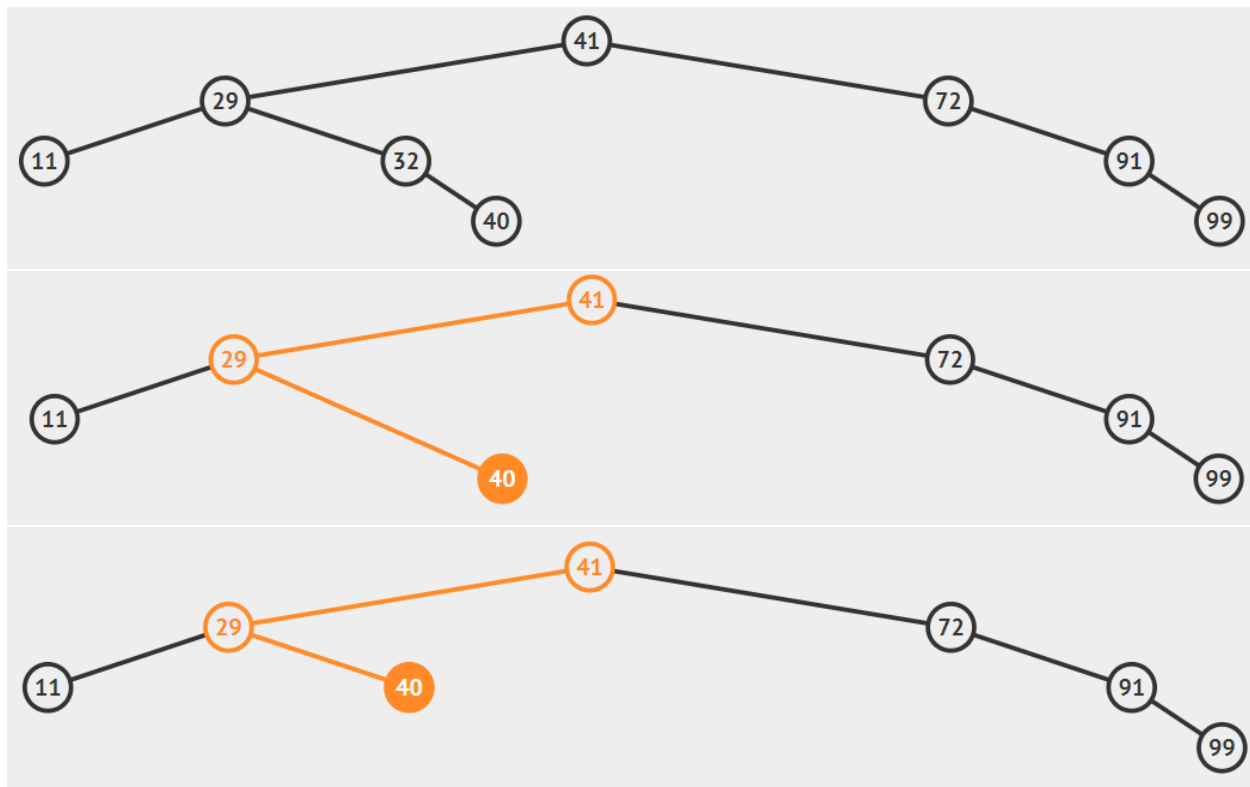
`Node(72).get_left` should return `Node(50)`

To delete:

`Node(72).set_left(None)`

Case 2: 1 child node

Suppose we want to remove a node with only 1 child node, say `Node(32)`. We will replace `Node(32)` with its child node, `Node(40)`. This includes `Node(40)` and its descendants.



Case 3(a): 2 child nodes (left child present)

Suppose we want to remove a node with 2 child nodes, say Node(65).

We first look for the node with the next larger value. We do this by first branching right from the node once to Node(91), then branching left until there is no left child node, i.e. Node(72).

We will then delete the original Node(65), and replace the data of Node(65) with the data of Node(72) only. Its descendants are not included.

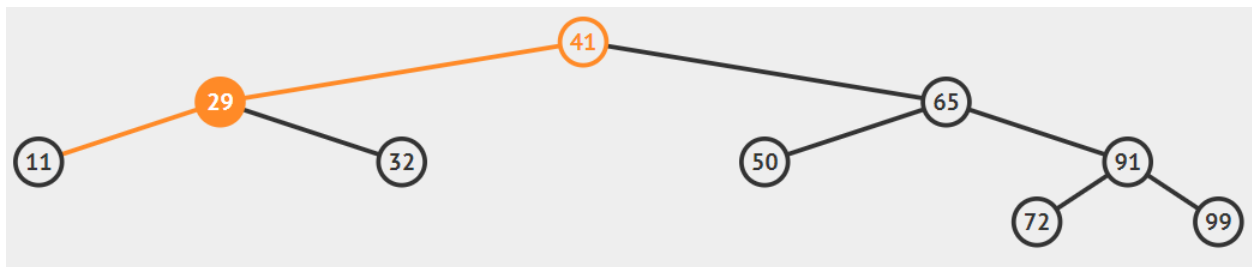
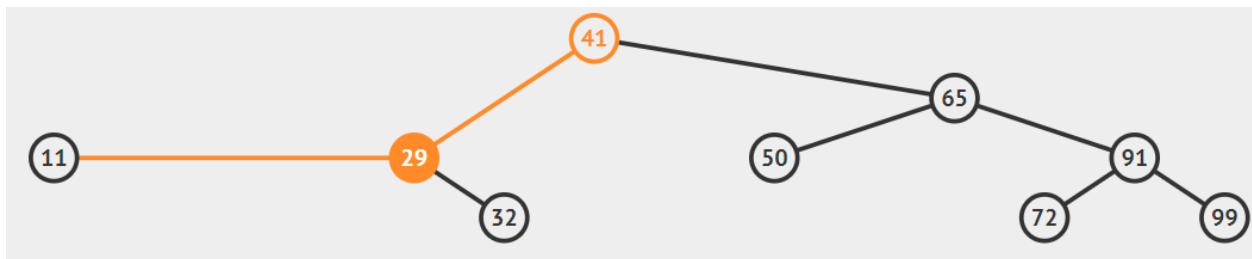
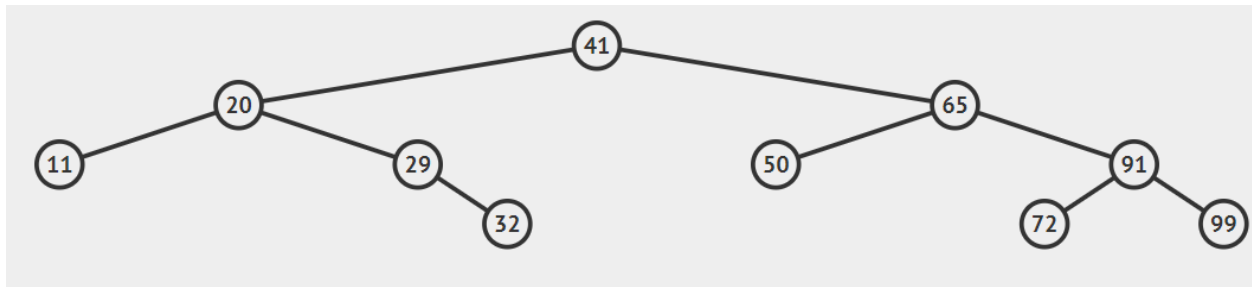


Case 3(b): 2 child nodes (left child non-existent)

Suppose we want to remove a node with 2 child nodes, say Node(20).

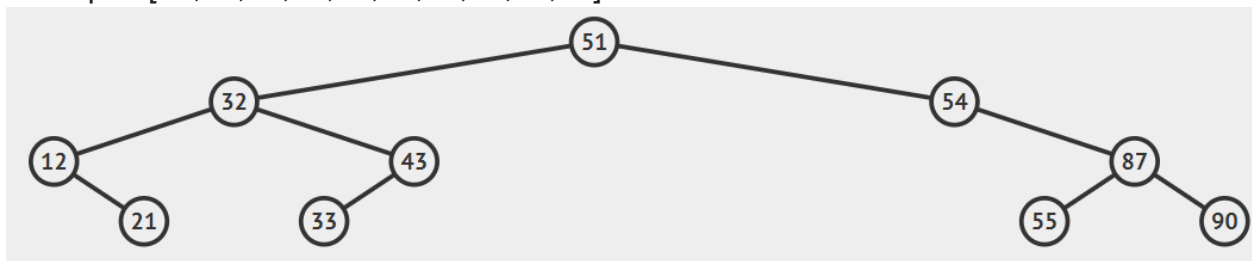
We first look for the node with the next larger value. We do this by first branching right from the node once to Node(29), then branching left until there is no left child node but in this case, there is no left child node.

Therefore, we will then delete the original Node(**20**) and replace the data of Node(**20**) with the data of Node(**29**) only. Its descendants are not included.



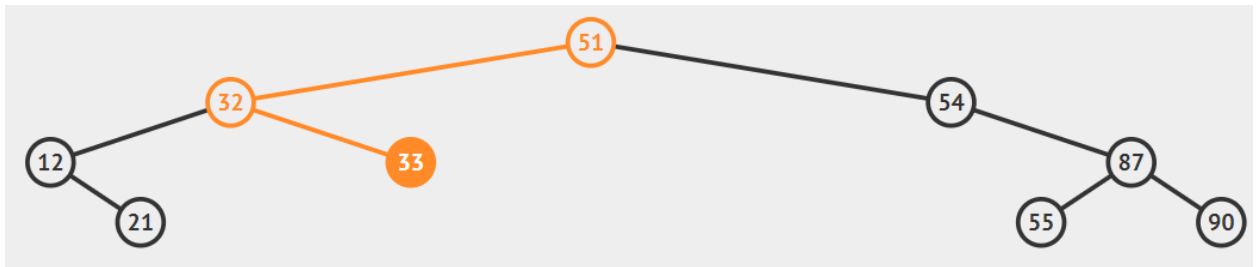
1.2.3.1 Deletion Walkthrough

Example: [51,32,43,54,12,33,87,90,55,21]



On separate diagrams, draw the resultant tree when we delete:

1) Node 43



2) Node 54



3) Node 87



4) Node 12



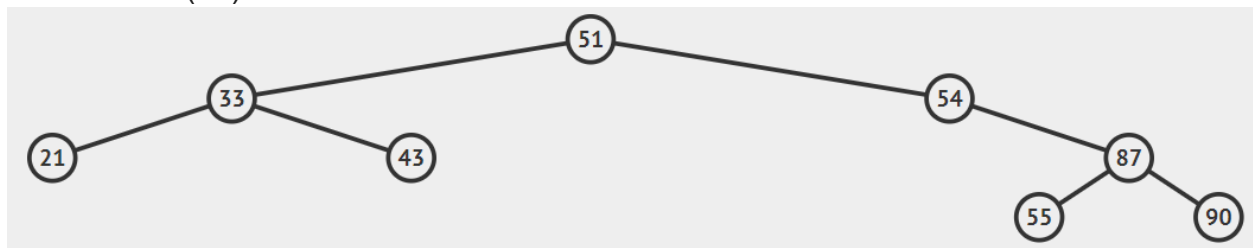
1.2.4 Update Operation [No coding in the syllabus]

To update the value of a node, say from 32 to 35, we can simply delete Node(32) and then, insert Node(35).

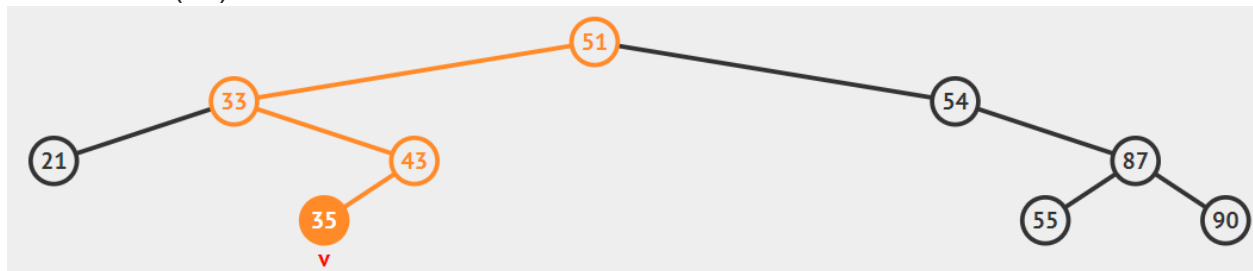


Draw the resultant tree.

Delete Node(32)



Insert Node(35)

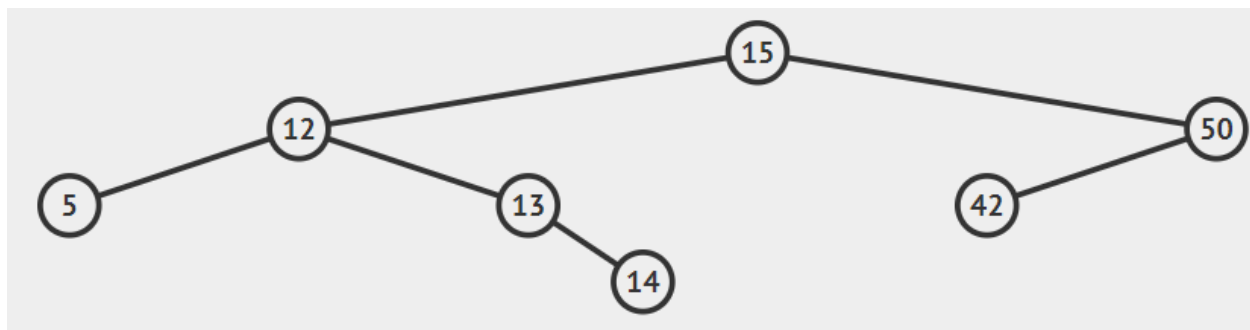


1.3 Binary Search Tree (Array Implementation)

Similar to linked lists, we can choose to implement a BST either using OOP or arrays.

The design for the implementation using an array could be a 2-D array with 3 columns. Using **None** as null pointers, when adding 15, 12, 50, 5, 42, 13, 14 in this order to a BST, we can have something like:

Draw out the BST.



Complete the table.

Index	Data	Left Pointer	Right Pointer
0	15	1	2
1	12	3	5
2	50	4	None
3	5	None	None
4	42	None	None
5	13	None	6
6	14	None	None

1.4 Time Complexity of BST for Search Operation

The height of a BST is the height of its root node. The height also corresponds to the maximum number of comparisons needed when we implement the search method.

Similar to the other algorithms that utilise the “divide and conquer” strategy, in the best-case scenario (a balanced tree), the time complexity is $O(\log n)$. However, in the worst-case scenario, the time complexity is $O(n)$.

The difference between $O(\log n)$ and $O(n)$ is significant for large n . In order to maintain the search performance, sometimes we may need to rebalance the tree. This involves re-creating the BST with precisely the same data items. The data items are inserted in an order such that the height of the tree is minimum.

1.4.1 Comparison of BST, Array, and Linked List

Criterion	Array	Linked list	BST
Insertion/deletion	Have to shift all elements after the insertion/deletion point	Change the next pointer	Change the left and right pointer
Time Complexity	$O(1)$, allows access via index	$O(n)$, iterate through the items one by one	$O(\log n)$, “divide and conquer” strategy

Refer to this link for more details:

<https://www.geeksforgeeks.org/time-complexities-of-different-data-structures/#:~:text=Time%20Complexity%20is%20a%20concept,to%20process%20a%20given%20input>

Readings

1. Reeves, B. (2015). Aqa a Level Computer Science (UK ed.). Hodder Education.
2. Langfield, S., & Duddell, D. (2019). Cambridge International AS and A Level Computer Science Coursebook with Cambridge Elevate Edition (2 Years) (2nd ed.). Cambridge University Press.
3. Heathcote, P. M., & Heathcote, R. S. U. (2016). OCR AS and A Level Computer Science (Illustrated ed.). PG Online Limited.
4. joejamesusa. (2015, January 22). *Binary search trees (BST) explained in animated demo*. YouTube. Retrieved, from <https://www.youtube.com/watch?v=mtvbVLK5xDQ>
5. Binary Tree Bootcamp https://backtobackswe.com/platform/content/binary-tree-bootcamp?utm_source=youtube&utm_medium=video
6. MIT 6.006 Lecture 5: Binary Search Trees, Bst Sort <https://ocw.mit.edu/courses/6-006-introduction-to-algorithms-fall-2011/resources/lecture-5-binary-search-trees-bst-sort/>
7. MIT 6.006 Lecture 6: Avl Trees, Avl Sort <https://ocw.mit.edu/courses/6-006-introduction-to-algorithms-fall-2011/resources/lecture-6-avl-trees-avl-sort/>

8. MIT 6.006 Recitation 5: Recursion Trees, Binary Search Trees
<https://ocw.mit.edu/courses/6-006-introduction-to-algorithms-fall-2011/resources/recitation-5-recursion-trees-binary-search-trees/>
9. CS50x 2022, Week 5 Data Structures
<https://cs50.harvard.edu/x/2022/weeks/5/>
10. Writing a Binary Search Tree in Python with Examples (Using one Node or Tree class)
<https://blog.boot.dev/computer-science/binary-search-tree-in-python/>
11. Binary Search Tree(Princeton University)
<https://algs4.cs.princeton.edu/32bst/>
12. Insertion in a BST – Iterative and Recursive Solution
<https://www.techiedelight.com/insertion-in-bst/>

Chapter 16 SQL and databases

Part 1

Contents

- 1 How is data stored?
- 2 Data redundancy
- 3 Relational database
- 4 Normalisation
- 5 Entity-Relationship diagram

Syllabus Learning Outcomes

3.3 Databases and Data Management

Understand, create and use SQL and NoSQL databases, as well as understand techniques to protect the privacy and integrity of data.

- 3.3.1 Determine the attributes of a database: table, record and field.
- 3.3.2 Explain the purpose of and use primary, secondary, composite and foreign keys in tables.
- 3.3.3 Explain with examples, the concept of data redundancy and data dependency.
- 3.3.4 Reduce data redundancy to third normal form (3NF).
- 3.3.5 Draw entity-relationship (ER) diagrams to show the relationship between tables.
- 3.3.6* Understand how NoSQL database management system addresses the shortcomings of relational database management system (SQL).
- 3.3.7* Explain the applications of SQL and NoSQL.
- 3.3.8* Use a programming language to work with both SQL and NoSQL databases.

*Note: NoSQL will be addressed in later chapter

1 How is data stored?

Data can be stored in a flat file database. A **database** is a collection of data stored in an organised or logical manner. Storing data in a database allows us to access and manage the data.

A **flat file database** is a database that contains a single table.

Field				
RegNo	Name	Gender	CivicsClass	CivicsTutor
1	Adam	M	18S12	Peter Lim
2	Adrian	M	18S12	Peter Lim
3	Agnes	F	18S12	Peter Lim
4	Aisha	F	18S12	Peter Lim
5	Ajay	M	18S12	Peter Lim
6	Alex	M	18S12	Peter Lim
7	Alice	F	18S12	Peter Lim
8	Amy	F	18S12	Peter Lim
9	Andrew	M	18S12	Peter Lim
10	Andy	M	18S12	Peter Lim

Record

A **table** contains a collection of **records** for a particular theme. In the Student table above, it is a collection of student data from a civics class.

A **record (a row in a table)** is a complete set of data about a single item. In the table above, each record contains the data of one student.

A table has a number of fields. A column in a table is called a **field**. **Attributes** are the describing characteristics or properties that define all items pertaining to a certain category applied to all cells of a column.

In the example above, the table has 10 records and 5 fields, and the attributes are RegNo, Name, Gender, CivicsClass and CivicsTutor.

Are there any potential issues that might arise from storing the data in a flat file database?

Peter Lim is the Civics Tutor for the Civics Class 18S12. In a flat file database, his details would need to be entered many times as long as a student is in the class 18S12.

This is known as **data redundancy**.

Exercise 1

Colour	Price	Stock
Red	0.50	30
Green	0.50	18
Yellow	0.80	43
Blue	0.90	66
White	0.85	39

How many records and fields does the table above has? What are the attributes?

2 Data redundancy

Data redundancy refers to the same data being stored more than once.

Having to enter data multiple times means the database is at an **increased risk** of having inaccurate data. It would be hard to update because every occurrence of the data item will need to be changed. This can lead to **data inconsistency**.

This will cause issues when inserting, updating and deleting data from the database.

RegNo	Name	Gender	CivicsClass	CivicsTutor
1	Adam	M	18S12	Peter Lim
2	Adrian	M	18S12	Peter Lim
3	Agnes	F	18S12	Peter Lim
4	Aisha	F	18S12	Peter Lim
5	Ajay	M	18S12	Peter Lim
6	Alex	M	18S12	Peter Lim
7	Alice	F	18S12	Peter Lim
8	Amy	F	18S12	Peter Lim
9	Andrew	M	18S12	Peter Lim
10	Andy	M	18S12	Peter Lim

Referring to the Student table above, see below for possible issues.

Inserting data	A new student from the same Civics Class to be inserted will need to insert duplicated data on the Civics tutor.
Updating data	If Peter Lim decide to leave the college, all the records with his name as the Civics Tutor in the Student table would need to be updated. If we miss any record, it will lead to inconsistent data.
Deleting data	If all the records of the Student table are deleted, information on civics class and civics tutor will be lost.

To address this issue, a more efficient way of storing data has to be adopted.

Exercise 2

RegNo	Name	Subject	Hod
1	Adam	CSE	Mr X
2	Adrian	CSE	Mr X
3	Alex	CSE	Mr X
4	Amy	CSE	Mr X

Is there data redundancy in the table above? Explain why this will be an issue.

3 Relational database

The most common model for a database is a relational database model.

A relational database is a collection of relational tables where data are organised in one or more tables with relationships between them.

In general, a table in a relational database can be expressed as:

TableName (Attribute1, Attribute2, Attribute3, ...)

For example,

Student (RegNo, Name, Gender, CivicsClass, CivicsTutor)

A table needs to fulfil the following conditions:

- Values are atomic (Information in each column cannot be an array or another table, it only contains one piece of information which cannot be broken down further.)
- Columns are of the same kind
- Rows are unique
- The order of columns is insignificant
- Each column must have a unique name

Exercise 3

Provide the table description of the following Balloon_sale table.

Colour	Price	AmountSold	Stock
Red	0.50	40	30
Green	0.50	17	18
Yellow	0.80	57	43
Blue	0.90	24	66
White	0.85	36	39

It is important to be able to identify each record in table uniquely using a key field.

A **key field**, or **key** in short, is a combination of one or more columns in a database that uniquely identifies a row in a table.

Keys allows for the establishment of relationships between tables and allows for the identification of relation between tables. Keys also help to enforce identity and integrity in the relationship.

There are different type of keys.

Candidate keys

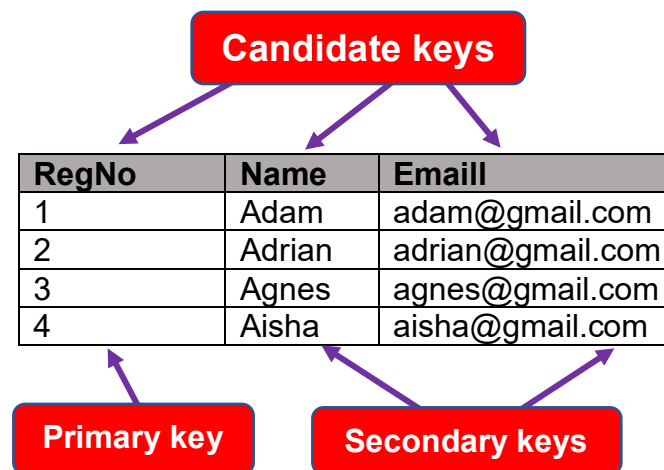
A **candidate key** is defined as a minimal set of fields which can uniquely identify each record in a table. It tells a particular record apart from another record. A candidate key should never be NULL or empty. There can be more than one candidate key for a table. A candidate key can also be a combination of more than one field.

Primary keys

A **primary key** is a field or a set of fields in a table whose values uniquely identifies each record in a table and should not change over time. That is, a primary key tells a particular record apart from another record. It is also a candidate key that is most appropriate to become the main key for a table.

Secondary keys

Candidate keys that are not selected as primary key are known as **secondary keys**. A secondary key is an additional key, or alternate key, which can be used in addition to the primary key to locate specific data.



In the given table above, RegNo, Name, and Email are candidate keys which help us to uniquely identify the student record in the table. Any one of the candidate keys can be selected as the primary key, for example, RegNo. The rest of the two keys would be the secondary keys.

Composite keys

Sometimes, more than one field is needed to uniquely identify a record. A **composite key** is a combination of two or more fields in a table that can be used to uniquely identify each record in a table. Uniqueness is only guaranteed when the fields are combined. When taken individually, the fields do not guarantee uniqueness.

RegNo	Name	CivicsClass
1	Adam	18A10
2	Adrian	18A10
1	Adam	18S12
2	Bala	18S12

In the example above, both RegNo and CivicsClass are needed to form a composite key to uniquely identify a record.

Foreign keys

A **foreign key** is an attribute (field) in one table that refers to the primary key in another table. It links to a primary key in the other table and form a relationship between the tables.

Another table, ClassInfo, as shown below, stores the information about each civics class.

CivicsClass	CivicsTutor	HomeRoom
18S12	Peter Lim	BlkA-L2-3
18A10	Pauline Lee	BlkC-L1-4

Notice that the CivicsClass field is the primary key in the table ClassInfo and is related or linked to the CivicsClass field in table Student. This makes CivicsClass field in table Student a foreign key (FK).

One benefit of a relational database is that all data can be updated at the original **source**. Data doesn't need to be repeated and relationships can be made using the source table's **unique identifier**.

4 Normalisation

Normalising the tables in the relational database is an important process of organising the tables so as to reduce data redundancy and prevent inconsistent data. There are at least three normal forms associated with normalisation: first normal form (1NF), second normal form (2NF), and third normal form (3NF).

First Normal Form

For a table to be in 1NF, all columns must be atomic. This means there can be no multi-valued columns – i.e., columns that would hold a collection such as an array or another table. In another words, the information in each column cannot be broken down further.

Consider the following table:

MatricNo	Name	Gender	CivicsClass	CivicsTutor	HomeRoom	CCAIInfo
1	Adam	M	18S12	Peter Lim	TR1	Tennis Teacher IC = Adrian Tan
2	Adrian	M	18S12	Peter Lim	TR1	Choir Teacher IC = Adeline Wong, Student Council Teacher IC = David Leong
3	Adam	M	18A10	Pauline Lee	TR2	Rugby Teacher IC = Andrew Quah
4	Bala	M	18A10	Pauline Lee	TR2	Badminton Teacher IC = Lilian Lim
5	Bee Lay	F	18A10	Pauline Lee	TR2	Choir Teacher IC = Adeline Wong, Chess Club Teacher IC = Edison Poh

It is assumed that every civics class only has one civics tutor, and each CCA has only one teacher in-charge (IC).

This table is not in 1NF because the CCAInfo column contains multiple values.

In order for the table to be in 1NF, we split CCAInfo into two single-value columns CCAName and CCATeacherIC. Notice that the students with MatricNo 2 and MatricNo 5 have multiple CCAs. We keep this information intact by splitting their records into multiple records, each corresponding to a different CCA. The resulting table is shown below.

MatricNo	Name	Gender	CivicsClass	CivicsTutor	HomeRoom	CCAName	CCATeacherIC
1	Adam	M	18S12	Peter Lim	TR1	Tennis	Adrian Tan
2	Adrian	M	18S12	Peter Lim	TR1	Choir	Adeline Wong
2	Adrian	M	18S12	Peter Lim	TR1	Student Council	David Leong
3	Adam	M	18A10	Pauline Lee	TR2	Rugby	Andrew Quah
4	Bala	M	18A10	Pauline Lee	TR2	Badminton	Lilian Lim
5	Bee Lay	F	18A10	Pauline Lee	TR2	Choir	Adeline Wong
5	Bee Lay	F	18A10	Pauline Lee	TR2	Chess Club	Edison Poh

The values for CCAName and CCATeacherIC are now atomic for each row. The primary key for the above table will be the composite key formed by MatricNo and CCAName.

Second Normal Form

For a table to be in 2NF, it must satisfy two conditions:

- 1) The table should be in 1NF
- 2) Every non-key attribute must be functionally dependent on the entire primary key.

Attribute Y is **functionally dependent** on attribute X (usually the primary key), if for every valid instance of X, the value of X uniquely determines the value of Y ($X \rightarrow Y$).

This means that for every non-key attribute, the entire primary key uniquely determines the value of that attribute, i.e. no attribute can depend on only part of the primary key.

The table above is not in 2NF. Name, Gender, CivicsClass, CivicsTutor and HomeRoom is dependent on only part of the primary key, MatricNo. CCATeacherIC is dependent only on CCAName. Thus, we decompose the table into three tables as shown below.

Student

MatricNo	Name	Gender	CivicsClass	CivicsTutor	HomeRoom
1	Adam	M	18S12	Peter Lim	TR1
2	Adrian	M	18S12	Peter Lim	TR1
3	Adam	M	18A10	Pauline Lee	TR2
4	Bala	M	18A10	Pauline Lee	TR2
5	Bee Lay	F	18A10	Pauline Lee	TR2

StudentCCA

MatricNo	CCAName
1	Tennis
2	Choir
2	Student Council
3	Rugby
4	Badminton
5	Choir
5	Chess Club

CCAInfo

CCAName	CCATeacherIC
Tennis	Adrian Tan
Choir	Adeline Wong
Student Council	David Leong
Rugby	Andrew Quah
Badminton	Lilian Lim
Chess Club	Edison Poh

The primary key for table Student will be MatricNo. MatricNo and CCAName will form a composite key for table StudentCCA and CCAName will be the primary key for table CCAInfo.

Third Normal Form

For a table to be in 3NF, it must satisfy two conditions:

- 1) The table should be in 2NF
- 2) The table should not have transitive dependencies – all fields must only be determined by the primary/composite key, not by other non-key attributes

A functional dependency $X \rightarrow Z$ is said to be **transitive** if there exists an attribute Y such that $X \rightarrow Y$ and $Y \rightarrow Z$

Note that $X \rightarrow Y$ **does not** necessarily imply the converse $Y \rightarrow X$.

Looking at Student table, CivicsTutor and HomeRoom are dependent on CivicsClass and CivicsClass is dependent on MatricNo. Therefore, CivicsTutor and HomeRoom are transitively dependent on MatricNo. To remove the transitive dependency, we decompose the Student table as follows:

Student

MatricNo	Name	Gender	CivicsClass
1	Adam	M	18S12
2	Adrian	M	18S12
3	Adam	M	18A10
4	Bala	M	18A10
5	Bee Lay	F	18A10

Civics

CivicsClass	CivicsTutor	HomeRoom
18S12	Peter Lim	TR1
18A10	Pauline Lee	TR2

MatricNo remains the primary key for Student table and CivicsClass will be the primary key of the newly formed Civics table.

The final design after normalisation is represented below:

Student (MatricNo, Name, Gender, CivicsClass)

Civics (CivicsClass, CivicsTutor, HomeRoom)

StudentCCA (MatricNo, CCAName)

CCAINfo (CCAName, CCATeacherIC)

The primary key for each table is indicated by underlining one or more attributes using solid lines. Foreign keys are indicated by using a dashed underline.

This design could also be represented using an **Entity-Relationship** diagram, also known as an **E-R diagram**.

Exercise 5

Normalise the table below and write the table descriptions for each table.

Underline all primary keys with solid lines and all foreign keys with dashed lines.

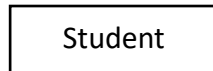
employee_name	home_address	department_name	date	location	manager
Emy laurent	107 Jurong Road	operations	12/2/2006	10 Western Plaza	Emy Laurent
Jain Singh	10 Johor Way	human resource	2/2/2006	10 Western Plaza	Jeremy Sentoso
Jeremy Sentoso	107 Jurong Road	human resource	4/3/2006	10 Western Plaza	Jeremy Sentoso
John McMallen	107 Jln Kampung Baru	sales	6/12/2006	107 Jurong Road	Peter Ho
Kowk Li Shin	19 Johor Way	sales	11/12/2006	107 Jurong Road	Peter Ho
Nancy Santi	123 Pantai Street	human resource	3/3/2006	10 Western Plaza	Jeremy Sentoso
Nancy Santi	123 Pantai Street	sales	1/4/2006	107 Jurong Road	Peter Ho
Paul Smith	22 Bukit Ridge	accounting	7/5/2006	107 Jurong Road	Paul Smith
Peter Ho	22 Bukit Ridge	operations	12/12/2006	10 Western Plaza	Emy Laurent
Peter Ho	22 Bukit Ridge	sales	1/11/2020	107 Jurong Road	Peter Ho
Putri Bte Mohamed	12 Raffles Road	accounting	4/3/2007	107 Jurong Road	Paul Smith
Reza Ernamwati	19 Johor Way	accounting	6/2/2006	107 Jurong Road	Paul Smith
Yao Chen	123 Pantai Street	sales	11/11/2006	107 Jurong Road	Peter Ho
Zhou Li	19 Bencoleen Lane	operations	8/5/2007	10 Western Plaza	Emy Laurent

5 Entity-Relationship diagram

Designers of a large database will normally construct a diagram of the planned database. An example of this is entity-relationship (E-R) diagram.

An **entity** is a specific object of interest (usually indicating a table in the database). Collective nouns or nouns are usually used to name entities. Entities are represented by rectangles.

For example,



Relationship describe a link between two entities.

Relationships can be:

- One to one
- One to many
- Many to many

One of the following three relationships can exist between two entities.

1. One to One

A **one to one** relationship means that **one record** in a table relates to a **single** record in another table.



For example, at a concert each ticket entitles you to a particular seat and each seat is linked to only one ticket.



2. One to Many

A **one to many** relationship means that **one record** in a table relates to **multiple** records in another table.



For example, a student can belong to only one civics class but a civics class can have many students.



3. Many to Many

A **many to many** relationship means that **multiple** records in a table relate to **multiple** records in another table.



For example, a student can join many CCAs and one CCA can have many students.



As it is difficult to implement a many-to-many relationship in a database system as it may indicate transitive dependencies that must be removed for 3NF.

An example of many-to-many relationship tables is shown below.

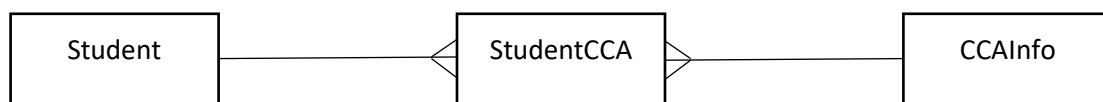
Student table

MatricNo	Name	Gender	CivicsClass	CCAName
1	Adam	M	18S12	Tennis
2	Adrian	M	18S12	Choir
2	Adrian	M	18S12	Student Council
3	Adam	M	18A10	Rugby
4	Bala	M	18A10	Badminton
5	Bee Lay	F	18A10	Choir
5	Bee Lay	F	18A10	Chess Club

CCAInfo table

MatricNo	CCAName	CCATeacherIC
1	Tennis	Adrian Tan
2	Choir	Adeline Wong
2	Student Council	David Leong
3	Rugby	Andrew Quah
4	Badminton	Lilian Lim
5	Choir	Adeline Wong
5	Chess Club	Edison Poh

We will usually decompose a many-to-many relationship into two (or more) one-to-many relationships. For example,



Student

MatricNo	Name	Gender	CivicsClass
1	Adam	M	18S12
2	Adrian	M	18S12
3	Adam	M	18A10
4	Bala	M	18A10
5	Bee Lay	F	18A10

StudentCCA

MatricNo	CCAName
1	Tennis
2	Choir
2	Student Council
3	Rugby
4	Badminton
5	Choir
5	Chess Club

CCAInfo

CCAName	CCATeacherIC
Tennis	Adrian Tan
Choir	Adeline Wong
Student Council	David Leong
Rugby	Andrew Quah
Badminton	Lilian Lim
Chess Club	Edison Poh

The E-R Diagram can then be directly translated into database designs.

Representing the normalised tables,

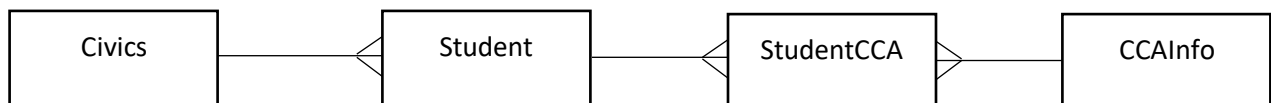
Student (MatricNo, Name, Gender, CivicsClass)

Civics (CivicsClass, CivicsTutor, HomeRoom)

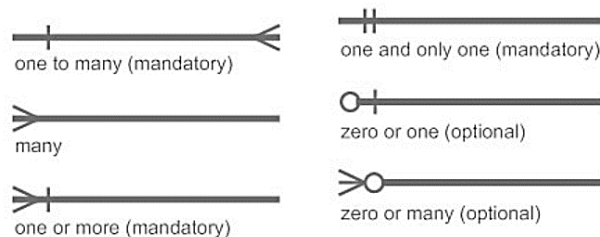
StudentCCA (MatricNo, CCAName)

CCAInfo (CCAName, CCATeacherIC)

Using an E-R diagram, we will have the following:



Other relationships which may come in handy for E-R diagrams:



Exercise 6

A small library wants to keep track of its collections (e.g., fiction, non-fiction, journals, etc.), the items in those collections, the physical location in the library of these collections, the members of the library and the items they borrow from the various collections.

- A collection must contain at least one item
- An item is contained in one collection
- A collection is housed in one location
- A location can house more than one collection
- A member can borrow more than one item
- An item can be borrowed by more than one member

Draw the entity-relationship diagram for the database above.

References

1. Burdett, A. (2016). *BCS glossary of computing*. BCS Learning and Development Limited.
2. Captain, F. A. (2015). *Six-step relational database design: A step by step approach to relational database design and development*. Place of publication not identified: Publisher not identified.
3. LANGFIELD, S. D. (2015). *Cambridge International AS and A Level Computer Science Coursebook*. Place of publication not identified: Cambridge University Press.
4. Archiveddocs. (n.d.). Lesson 2: Designing a Normalized Database. Retrieved from [https://docs.microsoft.com/en-us/previous-versions/tn-archive/cc505842\(v=technet.10\)](https://docs.microsoft.com/en-us/previous-versions/tn-archive/cc505842(v=technet.10))
5. Introduction to Database Keys. (n.d.). Retrieved from <https://www.studytonight.com/dbms/database-key.php>
6. Watt, A. (2014, October 24). Database Design - 2nd Edition. Retrieved from <https://opentextbc.ca/dbdesign01/>
7. Singh, C., & Prasad. (2018, December 14). Functional dependency in DBMS. Retrieved from <https://beginnersbook.com/2015/04/functional-dependency-in-dbms/>
8. Chapter 6. Entity-Relationship Modelling. (n.d.). Retrieved from https://www.cs.uct.ac.za/mit_notes/database/htmls/chp06.html#one-to-one-relationships-between-two-entities

Chapter 16 SQL and Databases

Part 2

Contents

- 1 Structured Query Language
- 2 SQLite
- 3 Database Operations
- 4 DB Browser for SQLite
- 5 CREATE
 - 5.1 Create database
 - 5.2 Create tables
 - 5.3 Insert data
- 6 UPDATE
- 7 DELETE
- 8 READ
 - 8.1 SELECT
 - 8.2 ORDER BY
 - 8.3 GROUP BY
 - 8.4 JOIN
 - 8.4.1 Cross join
 - 8.4.2 Inner join and Left outer join

Syllabus Learning Outcomes

3.3 Databases and Data Management

Understand, create and use SQL and NoSQL databases, as well as understand techniques to protect the privacy and integrity of data.

- 3.3.1 Determine the attributes of a database: table, record and field.
- 3.3.2 Explain the purpose of and use primary, secondary, composite and foreign keys in tables.
- 3.3.3 Explain with examples, the concept of data redundancy and data dependency.
- 3.3.4 Reduce data redundancy to third normal form (3NF).
- 3.3.5 Draw entity-relationship (ER) diagrams to show the relationship between tables.
- 3.3.6* Understand how NoSQL database management system addresses the shortcomings of relational database management system (SQL).
- 3.3.7* Explain the applications of SQL and NoSQL.
- 3.3.8* Use a programming language to work with both SQL and NoSQL databases.

*Note: NoSQL will be addressed in later chapter

1 Structured Query Language

Structured Query Language (SQL) is a standard computer language for the operation and management of relational databases. It is a language used to communicate with a database. SQL can be used to query, insert, update and modify data.



There are many types of SQL database engines, e.g. MySQL, Microsoft SQL, SQLite, PostgreSQL etc. A database engine is the software that a database management system (DBMS) uses to create, read, update and delete (CRUD) data from a database. For our syllabus, you will be using SQLite.

2 SQLite

SQLite supports the concept of type affinity on columns/fields. Type affinity refers to the preferred data type stored in a column.

Each column in a SQLite table is assigned one of the following type affinities:

Type affinities	Meanings
INTEGER	The value is a signed integer, stored in 0, 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the value.
TEXT	The value is a text string, stored using the database encoding (UTF-8, UTF-16BE or UTF-16LE).
REAL	The value is a floating point value, stored as an 8-byte IEEE floating point number.
NUMERIC	Values entered are converted to any one of the datatypes which is most appropriate.
BLOB	The value is a blob of data, stored exactly as it was input. Usually used to store large binary data, such as images or multimedia in a database.

SQLite does not have a separate Boolean datatype. Instead, Boolean values are stored as integers 0 (false) and 1 (true).

SQLite recognizes the keywords "TRUE" and "FALSE", as of version 3.23.0 (2018-04-02), but those keywords are just alternative spellings for the integer literals 1 and 0 respectively.

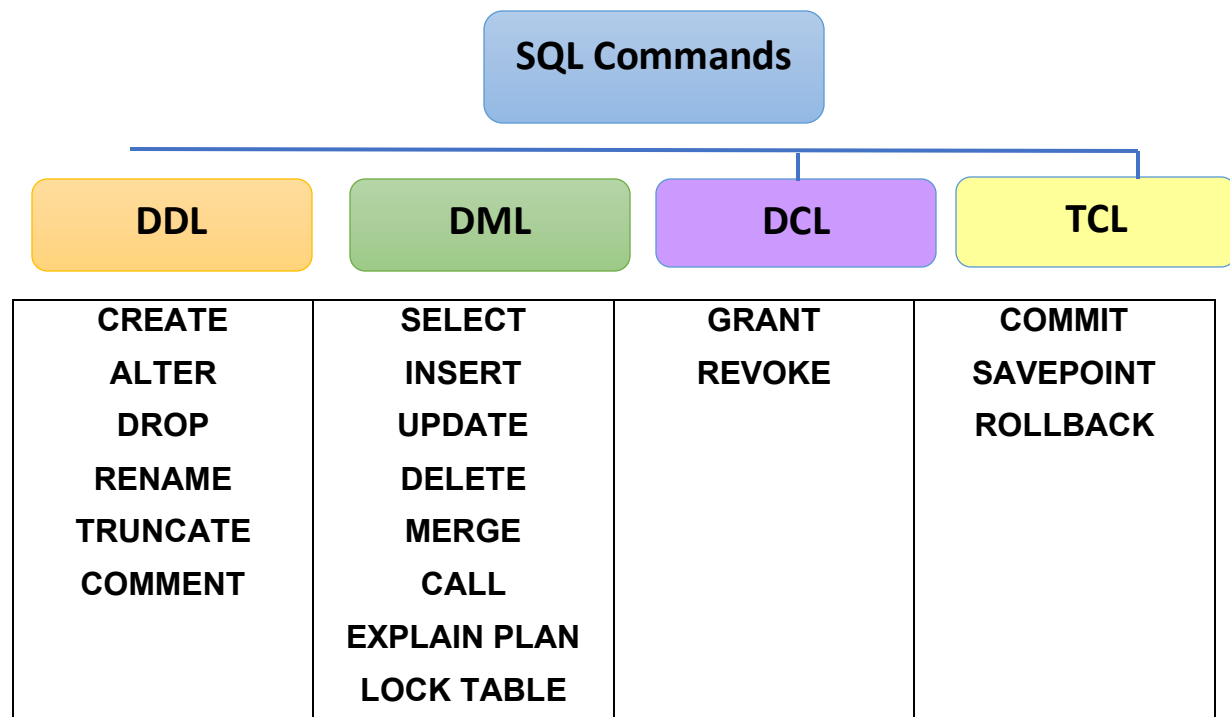
Type affinities are relevant in database migrations. This maximizes the compatibility between SQLite and other database management systems. For example, a column declared as VARCHAR in one database is assigned the type affinity of TEXT if it was migrated to and stored in a SQLite database instead. This feature helps to ensure portability across databases.

In the current syllabus, only INTEGER, TEXT and REAL will be used.

3 Database Operations

In industry-based database applications, SQL commands are grouped into four categories listed below.

- Data Definition Language (**DDL**) defines database schemas.
- Data Manipulation Language (**DML**) is used to retrieve and modify data.
- Data Control Language (**DCL**) is used to control access to a database.
- Transaction Control Language (**TCL**) is used to manage changes to a database, usually at transactional level.



Some of the more advanced commands under DCL and TCL are more relevant to industry-specific roles, such as database administrators. For the purposes of our learning, you will need to understand and apply these basic CRUD database operations:

Operation	SQL Command
<u>C</u> REATE	INSERT
<u>R</u> EAD (RETRIEVE)	SELECT
<u>U</u> PDATE (MODIFY)	UPDATE
<u>D</u> ELETE (DESTROY)	DELETE

4 DB Browser for SQLite

DB Browser for SQLite is a simple and easy to use Graphical User Interface (GUI) - based software for the creation and editing of database files compatible with SQLite. It abstracts and hides the details of complex SQL commands while providing an easy to user interface for performing the same database operations.

The downloadable version of it is available at <http://sqlitebrowser.org/>.

An online version of it is known as SQLite Online is found at <https://sqliteonline.com/>

5 CREATE

You will now create a library database with four tables (Borrower, Publisher, Book and Loan) by entering SQL commands using DB Browser for SQLite.

The library contains books that can be on loan to borrowers.

- A borrower can take one or more loans.
- Each loan record belongs to only one borrower.
- A book can be loaned many times.
- A publisher publishes one or more books.
- A book can be published by zero or one publisher.
For example, school lecture notes are not published by an official publishing house.

Exercise 1

Draw the E-R diagram for the database above.

5.1 Create database

A database needs to be created to store the tables. The steps to create a database is as follows:

1. Open DB Browser for SQLite.
2. Click **File**, then **New Database**.
3. Save your database file as **library**. The default extension is .db.

Note: other database file extensions are sqlite/sqlite3/db3

Note: A “edit table definition” window may automatically pop up when database is created.

5.2 Create tables

When we create tables, we can set **constraints**. Constraints are the rules enforced on data columns in a table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.

The following constraints are commonly used in SQL:

- **NOT NULL** – ensures that a field cannot be empty or has a NULL value
- **UNIQUE** - ensures that all values in a column are different
- **PRIMARY KEY** – A combination of a NOT NULL and UNIQUE. Uniquely identifies each row/record in a table.
- **AUTOINCREMENT** – value increases automatically with each new record inserted.
- **FOREIGN KEY** – references to the **PRIMARY KEY** in another table which is used to enforce "exists" relationships between tables. Prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the referenced table

After understanding constraints, the first table you will create is the **Borrower** table, with the constraints, as shown below.

Borrower

Column Name	Type	Constraints
ID	INTEGER	PRIMARY KEY, AUTOINCREMENT
FirstName	TEXT	NOT NULL
Surname	TEXT	NOT NULL
Contact	TEXT	NOT NULL

General SQL statement:

```
CREATE TABLE table_name (  
    column1_name COLUMN1_TYPE COLUMN1_CONSTRAINTS,  
    column2_name COLUMN2_TYPE COLUMN2_CONSTRAINTS,  
    ...  
    PRIMARY KEY (column1_name, column2_name,...)  
    FOREIGN KEY (column1_name) REFERENCES table_name(column_name)  
);
```

SQL statement to create Borrower table:

```
CREATE TABLE Borrower (  
    ID    INTEGER PRIMARY KEY AUTOINCREMENT,  
    FirstName TEXT NOT NULL,  
    Surname TEXT NOT NULL,  
    Contact TEXT NOT NULL  
);
```

Let's look at the SQL statement carefully.

INTEGER and TEXT are data types, ID, FirstName, Surname and Contact are field names. The ID field is the primary key of the table Borrower, while Autoincrement means the ID value is automatically given by the database. NOT NULL means that the fields do not accept null values. By default, a PRIMARY KEY field will not accept null values and must be unique.

If a table already exist in the database, creating the same table again will introduce an error saying that the table already exists. Hence, to avoid having such errors, "IF NOT EXISTS" can be included in the CREATE TABLE command.

```
CREATE TABLE IF NOT EXISTS table_name (  
    ...  
    ...  
);
```

Exercise 2

After creating the library database and Borrower table successfully, you will now create the **Publisher**, **Book** and **Loan** tables and apply the relevant constraints. You will need to take note of special constraints which help to maintain inter-table dependencies and the integrity of related data in different tables. They will affect the order in which tables are created.

Publisher

Column Name	Type	Constraints
ID	INTEGER	PRIMARY KEY, AUTOINCREMENT
Name	TEXT	NOT NULL

Note: The Publisher table has to be created before the Book table because of the foreign key reference to ID in the Publisher table. Tables with foreign keys have to be created after the referenced tables are created.

Book

Column Name	Type	Constraints
ID	INTEGER	PRIMARY KEY, AUTOINCREMENT
Title	TEXT	NOT NULL
PublisherID	INTEGER	FOREIGN KEY to ID in Publisher table
Damaged	INTEGER	NOT NULL

Note: Damaged is an attribute that tracks the condition of the book. A value of 0 means that the book is not damaged, while a value of 1 means that the book is damaged.

Loan

Column Name	Type	Constraints
ID	INTEGER	PRIMARY KEY, AUTOINCREMENT
BorrowerID	INTEGER	FOREIGN KEY to ID in the Borrower table
BookID	INTEGER	FOREIGN KEY to ID in the Book table
DateBorrowed	TEXT	

5.3 Insert data

The INSERT INTO command is used to insert a new record in a table.

General SQL statement:

```
INSERT INTO table_name (column1_name, column2_name, ...)  
VALUES (column1_value, column2_value, ...);
```

SQL statement to insert a new borrower:

```
INSERT INTO Borrower (FirstName, Surname, Contact)  
VALUES ("Peter", "Tan", 99299345)
```

This will insert a new borrower into the Borrower table.

Note that no number was inserted into ID field. The ID column is an auto-increment field and will be generated automatically when a new record is inserted into the table.

Exercise 3

Insert records into Borrower, Publisher, Book and Loan tables.

Borrower table

FirstName	Surname	Contact
Sarah	Lee	81111123
Kumara	Ravi	94456677
Some	User	11111111

Note that a semicolon needs to be included when entering more than one SQL statement at a time.

Publisher table

ID	Name
1	NPH
2	Unpop
3	Appleson
4	Squirrel
5	Yellow Flame

Book table

ID	Title	PublisherID	Damaged
1	The Lone Gatsby	5	0
2	A Winter's Slumber	4	1
3	Life of Pie	4	0
4	A Brief History Of Primates	3	0
5	To Praise a Mocking Bird	2	0
6	The Catcher in the Eye	1	1
123	H2 Computing Ten Year Series	NULL	0

Loan table

ID	BorrowerID	BookID	DateBorrowed
1	3	2	20180220
2	3	1	20171215
3	2	3	20171231
4	1	5	20180111

Exercise 4

Try this SQL statement:

```
INSERT INTO Loan (BorrowerID, BookID, DateBorrowed) VALUES (2, 8, "20180506")
```

Can this record be inserted into the Loan table?

6 UPDATE

The UPDATE command allows the editing of data values in a database. One or more records may be updated at the same time.

General SQL statement:

```
UPDATE table_name SET
    column1_name = column1_expression,
    column2_name = column2_expression,
    ...
WHERE condition;
```

The **WHERE** clause is used to filter records. It is used to extract only those records that fulfill a specified condition.

SQL statement to update a record in Book table:

```
UPDATE Book Set Title = 'Book: ' || Title
```

This will update the values of Title in the Book table such that each book title now starts with 'Book: ' Note the use of || for string concatenation (adding of two strings).

Consider another example.

```
UPDATE Loan Set BorrowerID = 2 WHERE BorrowerID = 3
```

This will update all the books borrowed by Kumara (BorrowerID 3) to Sarah (BorrowerID 2).
Note: The WHERE clause is not only used in UPDATE statements, it is also used in SELECT, DELETE, etc.!

Exercise 5

All the books borrowed by Sarah (BorrowerID 2) should have been borrowed by Peter (BorrowerID 1). Write down the SQL statement to update the Loan table.

7 DELETE

The DELETE command is used to delete existing records in a table.

General SQL statement:

```
DELETE FROM table_name  
WHERE condition;
```

SQL statement to delete a record in Book table:

```
DELETE FROM Book WHERE Title = 'The Catcher in the Eye'
```

This will delete the book 'The Catcher in the Eye' from the Books table.

Note: text is case-sensitive, which means that the following 2 SQL statements are different.

```
DELETE FROM Book WHERE Title = 'The Catcher in the Eye'  
DELETE FROM Book WHERE Title = 'the catcher in the eye'.
```

Be careful when using the DELETE statement. If WHERE conditions are accidentally left out, the DELETE statement will delete all entries from the Loan table.

```
DELETE FROM Loan
```

This will delete all entries from the Loans table!

That said, DELETE statements only remove records from tables, they do not remove empty tables. To remove tables, either empty or with contents, DROP TABLE command is used.

General SQL statement:

```
DROP TABLE table_name
```

SQL statement to delete Loan table:

```
DROP TABLE Loan
```

This will remove the entire Loan table from the database, together with any records in it.

Exercise 6

Try keying the SQL statements below:

- 1) DROP TABLE Publisher
- 2) DELETE FROM Borrower

Is it possible? Why?

8 READ

8.1 SELECT

The SELECT statement allows the user to retrieve data from the database.

General SQL statement:

```
SELECT column1_name, column2_name, ...  
FROM table_name  
WHERE condition
```

To select all fields, use *.

SQL statement to display records in Book table:

```
SELECT * FROM Book
```

This statement will display all fields of all the books in library.

```
SELECT ID, Title FROM Book
```

This statement will display the ID and Title fields of all the books in library.

```
SELECT * from Book WHERE Damaged = 1
```

This statement will display all the damaged books.

```
SELECT * from Book WHERE PublisherID IS NULL
```

This statement will display books with no publishers (no value on the PublisherID field).

The IS operator is used to check for NULL values.

More than one conditions can be inserted using AND and OR binary operators.

```
SELECT * FROM Book WHERE Title = 'Life of Pie' AND Damaged = 0
```

This statement returns the books with Title 'Life of Pie' **and** are not damaged.

8.2 ORDER BY

The sequence of display of records can also be in either ascending order (default) or descending order of choice.

General SQL statement:

Ascending order

```
SELECT column1_name, column2_name, ...  
FROM table_name  
WHERE condition  
ORDER BY order_expression ASC;
```

Note: data retrieved are displayed in ascending order by default, hence, it is optional to include ASC in the SQL statement.

Descending order

```
SELECT column1_name, column2_name, ...  
FROM table_name  
WHERE condition  
ORDER BY order_expression DESC;
```

8.3 GROUP BY

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of books published by each publisher".

The GROUP BY statement is often used with aggregate functions to group the result-set by one or more columns.

Aggregate functions help to count / calculate results from the database.

Aggregate function	Description
COUNT	Number of values
MAX	Maximum value
MIN	Minimum value
SUM	Sum of all values
AVG	Average of values

General SQL statement:

```
SELECT aggregate_func(column_name)  
FROM table_name  
WHERE condition  
GROUP BY column_name
```

Exercise 7

Write the SQL select statements to display the following:

1. All records where there is a publisher
2. All the Names of the publishers inside the Publisher table
3. The Book titles of undamaged books by Publisher with PublisherID 4
4. The Titles of the books which have PublisherID 1 or 2
5. The loan records after 1st Jan 2018
6. All the loan records ordered by BookID
7. All the records in Loan table arranged in descending order of BookID with BorrowerID = 3
8. Total number of loans

8.4 JOIN

A JOIN allows the combination of data from two sets of data (i.e. two tables). There are different types of joins. We look at three types: **cross join**, **inner join** and **left outer join**.

8.4.1 Cross join

Cross join returns the Cartesian product of rows from the tables in the join. It combines each row in the first table with each row in the second table.

For example, the following SQL statement

```
SELECT * FROM Book, Publisher
```

produces the following table.

ID	Title	PublisherID	Damaged	ID	Name
1	The Lone Gatsby	5	0	1	NPH
1	The Lone Gatsby	5	0	2	Unpop
1	The Lone Gatsby	5	0	3	Appleson
1	The Lone Gatsby	5	0	4	Squirrel
1	The Lone Gatsby	5	0	5	Yellow Flame
2	A Winter's Slumber	4	1	1	NPH
2	A Winter's Slumber	4	1	2	Unpop
2	A Winter's Slumber	4	1	3	Appleson
2	A Winter's Slumber	4	1	4	Squirrel
2	A Winter's Slumber	4	1	5	Yellow Flame
3	Life of Pie	4	0	1	NPH
3	Life of Pie	4	0	2	Unpop
3	Life of Pie	4	0	3	Appleson
3	Life of Pie	4	0	4	Squirrel
3	Life of Pie	4	0	5	Yellow Flame
4	A Brief History Of Primates	3	0	1	NPH
4	A Brief History Of Primates	3	0	2	Unpop
4	A Brief History Of Primates	3	0	3	Appleson
4	A Brief History Of Primates	3	0	4	Squirrel
4	A Brief History Of Primates	3	0	5	Yellow Flame
5	To Praise a Mocking Bird	2	0	1	NPH
5	To Praise a Mocking Bird	2	0	2	Unpop
5	To Praise a Mocking Bird	2	0	3	Appleson
5	To Praise a Mocking Bird	2	0	4	Squirrel
5	To Praise a Mocking Bird	2	0	5	Yellow Flame
6	The Catcher in the Eye	1	1	1	NPH
6	The Catcher in the Eye	1	1	2	Unpop
6	The Catcher in the Eye	1	1	3	Appleson
6	The Catcher in the Eye	1	1	4	Squirrel
6	The Catcher in the Eye	1	1	5	Yellow Flame
123	H2 Computing Ten Year Series	NULL	0	1	NPH
123	H2 Computing Ten Year Series	NULL	0	2	Unpop
123	H2 Computing Ten Year Series	NULL	0	3	Appleson
123	H2 Computing Ten Year Series	NULL	0	4	Squirrel
123	H2 Computing Ten Year Series	NULL	0	5	Yellow Flame

Notice that there are two columns of ID. The first ID column is from Book table, while the second ID column is from the Publisher table.

Aliases can be used to give a column in a table a temporary name to remove the confusion. An alias is created with the AS keyword.

For example,

```
SELECT Book.ID AS BID, Publisher.ID AS PID
FROM Book, Publisher
```

8.4.2 Inner join and Left outer join

The table above is a big table, with many unnecessary rows. It is more useful to display the publisher of each book. To do that, we can add a condition.

```
SELECT * FROM Book, Publisher
WHERE Book.PublisherID = Publisher.ID
```

The SQL statement above gives the following results:

ID	Title	PublisherID	Damaged	ID	Name
1	The Lone Gatsby	5	0	5	Yellow Flame
2	A Winter's Slumber	4	1	4	Squirrel
3	Life of Pie	4	0	4	Squirrel
4	A Brief History Of Primates	3	0	3	Appleson
5	To Praise a Mocking Bird	2	0	2	Unpop
6	The Catcher in the Eye	1	1	1	NPH

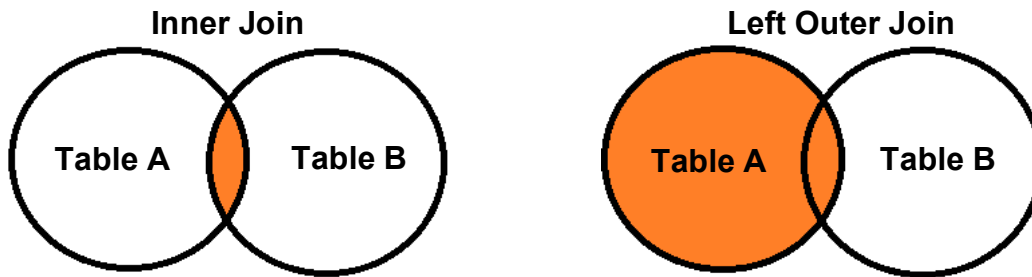
This table is more meaningful as it links the titles to the publishers.

This is an example of inner join. Inner join can also be explicitly defined.

```
SELECT * FROM Book
INNER JOIN Publisher
ON Book.PublisherID = Publisher.ID
```

However, notice that the results only include all books which have publishers. The title 'H2 Computing Ten Year Series' is not displayed as it does not have a PublisherID and hence has no publisher.

To display all books with and without a publisher, left outer join can be used.



Inner join selects all records from Table A and Table B which meet the join condition. Left outer join selects all records from Table A, along with records from Table B which meet the join condition.

The results will be different if we change the inner join to left outer join as shown below:

```
SELECT * FROM Book
LEFT OUTER JOIN Publisher
ON Book.PublisherID = Publisher.ID
```

This is because all records in Book are included in the left outer join, even books with no publishers, unlike the inner join.

ID	Title	PublisherID	Damaged	ID	Name
1	The Lone Gatsby	5	0	5	Yellow Flame
2	A Winter's Slumber	4	1	4	Squirrel
3	Life of Pie	4	0	4	Squirrel
4	A Brief History Of Primates	3	0	3	Appleson
5	To Praise a Mocking Bird	2	0	2	Unpop
6	The Catcher in the Eye	1	1	1	NPH
123	H2 Computing Ten Year Series	NULL	0	NULL	NULL

Exercise 8

Write the SQL select statements to display all the books that are not damaged with their Publishers.

References

1. Burdett, A. (2016). *BCS glossary of computing*. BCS Learning and Development Limited.
2. Captain, F. A. (2015). *Six-step relational database design: A step by step approach to relational database design and development*. Place of publication not identified: Publisher not identified.
3. LANGFIELD, S. D. (2015). *Cambridge International AS and A Level Computer Science Coursebook*. Place of publication not identified: Cambridge University Press.
4. Archiveddocs. (n.d.). Lesson 2: Designing a Normalized Database. Retrieved from [https://docs.microsoft.com/en-us/previous-versions/tn-archive/cc505842\(v=technet.10\)](https://docs.microsoft.com/en-us/previous-versions/tn-archive/cc505842(v=technet.10))
5. Introduction to Database Keys. (n.d.). Retrieved from <https://www.studytonight.com/dbms/database-key.php>
6. Watt, A. (2014, October 24). Database Design - 2nd Edition. Retrieved from <https://opentextbc.ca/dbdesign01/>
7. Singh, C., & Prasad. (2018, December 14). Functional dependency in DBMS. Retrieved from <https://beginnersbook.com/2015/04/functional-dependency-in-dbms/>
8. Chapter 6. Entity-Relationship Modelling. (n.d.). Retrieved from https://www.cs.uct.ac.za/mit_notes/database/htmls/chp06.html#one-to-one-relationships-between-two-entities

Chapter 16 SQL and Databases

Part 3: Using SQLite in Python

Contents

- 1 Connecting to SQLite database with sqlite3
- 2 Execute CRUD operations
- 3 Commit changes
- 4 Revert changes
- 5 Enclosing user input
- 6 Retrieving data

Syllabus Learning Outcomes

3.3 Databases and Data Management

Understand, create and use SQL and NoSQL databases, as well as understand techniques to protect the privacy and integrity of data.

- 3.3.1 Determine the attributes of a database: table, record and field.
- 3.3.2 Explain the purpose of and use primary, secondary, composite and foreign keys in tables.
- 3.3.3 Explain with examples, the concept of data redundancy and data dependency.
- 3.3.4 Reduce data redundancy to third normal form (3NF).
- 3.3.5 Draw entity-relationship (ER) diagrams to show the relationship between tables.
- 3.3.6* Understand how NoSQL database management system addresses the shortcomings of relational database management system (SQL).
- 3.3.7* Explain the applications of SQL and NoSQL.
- 3.3.8* Use a programming language to work with both SQL and NoSQL databases.

*Note: NoSQL will be addressed in later chapter

1 Connecting to SQLite database with sqlite3

Python comes with sqlite3 module which allows working with SQLite databases using Python.

Generally, to work with the database,

1. We first **establish connection** to the database with `connect()` method
2. Execute some SQL statements, with `execute()` method
3. **Save** the changes we made to the database with the `commit()` method
4. **Close** the database using the `close()` method. This is similar to how we handle file I/O earlier.

Connect to database	
1	<code>import sqlite3</code>
2	
3	<code>connection = sqlite3.connect("school.db")</code>
4	<code>connection.close()</code>

To connect to a database:

Step 1: import sqlite3 module

Step 2: make a connection with database using `sqlite3.connect`.

Note: If the database of interest, for e.g. `school.db`, does not exist, an empty `school.db` database will be created.

Step 3: To ensure the database file is closed properly, use `connection.close()`.

2 Execute CRUD operations

After loading an SQLite file and getting a connection, we can execute SQL statements by calling the connection object's `execute()` method with a str containing the SQL statement we wish to run. For instance, the following Python program creates a new table named `student` in a new SQLite file named `school.db`:

Execute SQL statements	
1	<code>import sqlite3</code>
2	
3	<code>connection = sqlite3.connect("school.db")</code>
4	<code>connection.execute("CREATE TABLE student " +</code>
5	<code> "(ExamNo INTEGER PRIMARY KEY," +</code>
6	<code> " Name TEXT, Class TEXT)")</code>
7	<code>connection.close()</code>

After running the program, we can open `school.db` using DB Browser to check that a `student` table was indeed created.

We can also make the statement into a string and parse the string as an argument to be executed.

Parse statement as a string	
1	<code>import sqlite3</code>
2	
3	<code>connection = sqlite3.connect("school.db")</code>
4	
5	<code>statement = """CREATE TABLE student (</code>
6	<code> ExamNo INTEGER PRIMARY KEY,</code>
7	<code> Name TEXT NOT NULL,</code>
8	<code> Class TEXT NOT NULL</code>
9	<code>); """</code>
10	
11	<code>connection.execute(statement)</code>
12	
13	<code>connection.close()</code>

If we try to run the program again, we will get the following error:

```
OperationalError: table student already exists
```

Any errors caused by running the SQL (such as the error telling us the student table already exists) are reported as Python exceptions and can be handled as usual.

To avoid this error, we can use "IF NOT EXISTS" when creating tables.

IF NOT EXISTS	
1	import sqlite3
2	
3	connection = sqlite3.connect("school.db")
4	
5	statement = """CREATE TABLE IF NOT EXISTS student (
6	ExamNo INTEGER PRIMARY KEY,
7	Name TEXT NOT NULL,
8	Class TEXT NOT NULL
9	); """
10	
11	connection.execute(statement)
12	
13	connection.close()

Now, let us try using INSERT to put data into the student table:

Insert record	
1	import sqlite3
2	
3	connection = sqlite3.connect("school.db")
4	
5	statement = "INSERT INTO student(ExamNo, Name, Class)" +
6	"VALUES(2150601, 'Jack Neo', '21S56')"
7	
8	connection.execute(statement)
9	
10	connection.close()
11	

This program runs with no errors. However, if we open student.db using DB Browser, we find that the inserted data is missing. What happened to the data? It was discarded because we did not save the changes we made to the database.

3 Commit changes

It is important to run the `commit()` method to save the changes to the database. This is equivalent to the action Write Changes in DB Browser.

Commit method	
1	<code>import sqlite3</code>
2	
3	<code>connection = sqlite3.connect("school.db")</code>
4	
5	<code>statement = """INSERT INTO student(ExamNo, Name, Class)</code>
6	<code>VALUES(2150601, 'Jack Neo', '21S56')"""</code>
7	
8	<code>connection.execute(statement)</code>
9	
10	<code>connection.commit()</code>
11	<code>connection.close()</code>

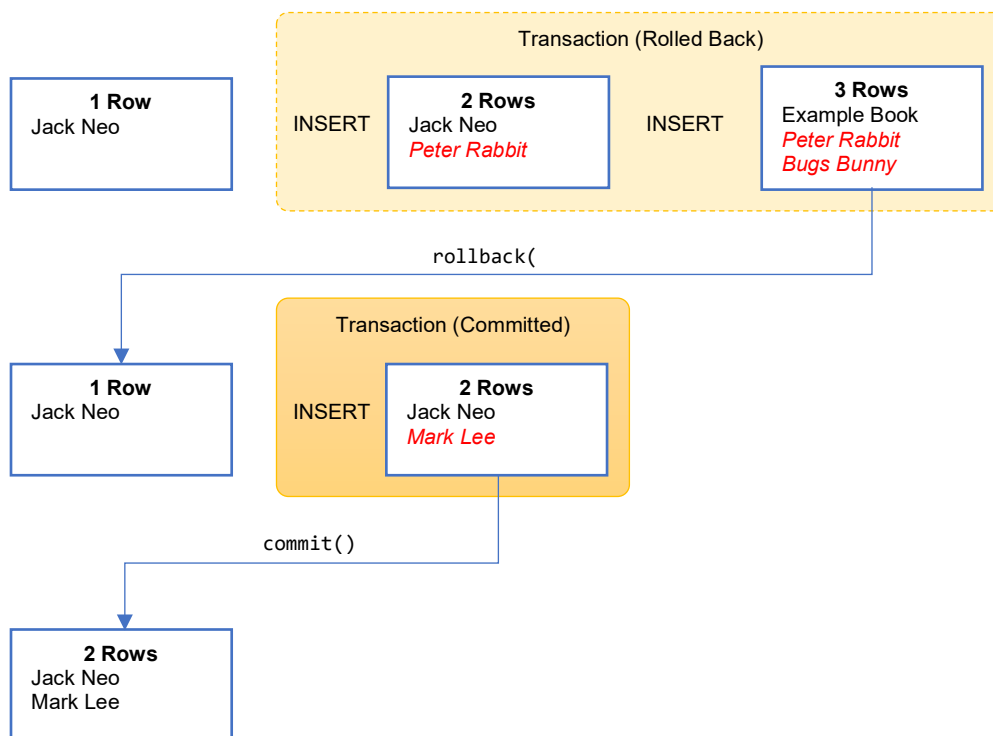
Adding the `commit()` method in line 10, the data is inserted and saved correctly.

4 Revert changes

We can discard all the changes made since the last commit was executed by calling the connection object's `rollback()` method.

Using rollback to revert changes	
1	<code>import sqlite3</code>
2	
3	<code>connection = sqlite3.connect("school.db")</code>
4	
5	<code>connection.execute("INSERT INTO student(ExamNo, Name, Class)" +</code>
6	<code>"VALUES(2150602, 'Peter Rabbit', '21S56')")</code>
7	<code>connection.execute("INSERT INTO student(ExamNo, Name, Class)" +</code>
8	<code>"VALUES(2150603, 'Bugs Bunny', '21S56')")</code>
9	<code>connection.rollback()</code>
10	
11	<code>connection.execute("INSERT INTO student(ExamNo, Name, Class)" +</code>
12	<code>"VALUES(2150602, 'Mark Lee', '21S56')")</code>
13	<code>connection.commit()</code>
14	
15	<code>connection.close()</code>

In the above example, the first two INSERT statements are rolled back so they have no effect on the database. On the other hand, the last INSERT statement is committed so it does affect the database. This process is illustrated by the following diagram:



Comparing the contents of the student table before and after running the program shows that the first two INSERT statements are indeed ignored.

Before	After
	only committed transaction is saved

5 Enclosing user input

When generating SQL commands, we often need to include some data that is provided by the user. We may be tempted to use str concatenation in order to generate the required SQL command. Unfortunately, this is insecure as special characters or keywords in the user's input are not escaped and malicious users can take advantage of this to inject his/her own SQL commands.

Instead, we should use *parameter substitution* to safely include data that is provided by the user. We use the question-mark character ? as a placeholder in the SQL for any data that is provided by the user. We then provide a second argument to execute, that is a tuple of values which will replace the placeholders. This ensures that the provided values are escaped properly and cannot be misinterpreted as SQL.

Using ? as placeholder

```
1 import sqlite3
2
3 connection = sqlite3.connect("school.db")
4
5 user_input = input("Enter Student Exam no to delete: ")
6
7 statement = "DELETE FROM student WHERE ExamNo = ?"
8
9 connection.execute(statement, (user_input,))
10
11 connection.commit()
12 connection.close()
```

The program above safely asks for the ExamNo of a student and deletes the corresponding row from the database.

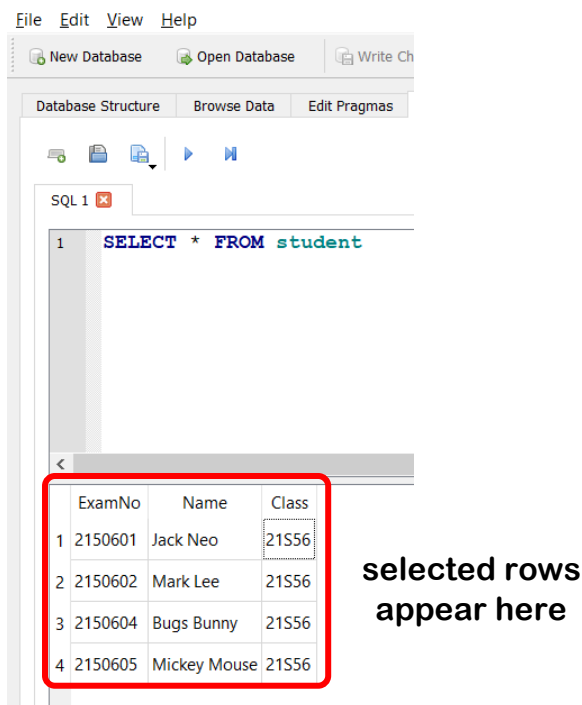
Parameter substitution follows the same order in which the placeholders appear in the SQL. This is illustrated by the following diagram:

```
connection.execute("INSERT INTO student (ExamNo, Name, Class)
VALUES (?, ?, ?)", (2150605, 'Mickey Mouse', '21S56'))
```



6 Retrieving data

As you have already learned, the SELECT command is used to select data from the database. When you run a SELECT command in DB Browser, the selected rows are usually displayed in a table below the SQL statement that was executed.



Using Python sqlite3 module, we can take 5 approaches to retrieve the data.

Approach 1: Iterate the Cursor object, which is also created when we run `execute()` method on Connection object,

Approach 2: Use the `fetchone()` method of the Cursor object - returns a tuple of the columns in the current row.

Approach 3: Use the `fetchall()` method of the Cursor object - fetch all the rows at once and keep them in a list. This method returns a list of tuples with each tuple containing the selected columns for a single row.

Approach 4: Cast Cursor object as a list – converts cursor object to a list object

Approach 5: Setting the row factory attribute of the Connection object as `sqlite3.Row` object. This configures the SQLite connection so that each row is retrieved as a dict, mapping column names to values.

5 approaches to retrieve data

```
1 cursor = connection.execute("SELECT * FROM student")
2
3 #Approach 1
4 for row in cursor:
5     print(row)
6
7 #Approach 2
8 row = cursor.fetchone()
9 while row is not None:
10     print(row)
11     row = cursor.fetchone()
12
13 #Approach 3
14 rows = cursor.fetchall()
15 print(rows)
16
17 #Approach 4
18 rows = list(cursor)
19 print(rows)
20
21 #Approach 5
22 #setting the `row_factory` attribute of `Connection` object
23 connection.row_factory = sqlite3.Row
24
25 query = "SELECT * FROM student"
26
27 cursor = connection.execute(query)
28
29 rows = cursor.fetchall()
30
31 for row in rows: # row is now a dictionary
32     print(row["ExamNo"])
33     print(row["Name"])
34     print(row["Class"])
```

sqlite3 Module Summary

Name	Description
<code>connect(filename)</code>	Creates a Connection object using SQLite file with the given filename
Row	Can be used as a Connection object's <code>row_factory</code> so <code>fetchone()</code> returns a dict mapping column names to field values instead of returning a tuple of values

Connection Class Summary

Name	Description
<code>commit()</code>	Saves changes to (but does not close) SQLite file
<code>close()</code>	Closes (but does not save changes to) SQLite file
<code>execute(sql)</code>	Runs the given SQL command on the database and returns a Cursor object
<code>execute(sql, values_tuple)</code>	Runs the given SQL command (first argument) after substituting question marks with the corresponding values in the given tuple (second argument) and returns a Cursor object.
<code>rollback()</code>	Undoes any changes made since the last call to <code>commit()</code> .
<code>row_factory</code>	Can be set to Row so <code>fetchone()</code> returns a dict mapping column names to field values instead of returning a tuple of values

Cursor Class Summary

Name	Description
<code>fetchone()</code>	Returns a tuple of values from next row of the query result or None if there are no more values Note: This function returns a dict mapping column names to field values instead if <code>row_factory</code> is set to Row
<code>fetchall()</code>	Calls <code>fetchone()</code> repeatedly until it returns None and returns a list of the non-None results

Alternative ways to execute SQL in Python

Create a cursor

```
1 import sqlite3
2
3 connection = sqlite3.connect("school.db")
4 c = connection.cursor()
5 c.execute("CREATE TABLE student " +
6           "(ExamNo INTEGER PRIMARY KEY," +
7           " Name TEXT, Class TEXT)")
8 c.execute("INSERT INTO student(ExamNo, Name, Class)" +
9           "VALUES(2150603, 'Peter Rabbit', '21S56')")
10 print(c.fetchone())
11
12 connection.commit()
13 connection.close()
```

Using with connection – connection can be used in a with block to auto-commit transactions

```
1 import sqlite3
2
3 connection = sqlite3.connect("school.db")
4
5 with connection:
6     connection.execute("CREATE TABLE student " +
7                         "(ExamNo INTEGER PRIMARY KEY," +
8                         " Name TEXT, Class TEXT)")
9     connection.execute("INSERT INTO student" +
10                        "(ExamNo, Name, Class)" +
11                        "VALUES(2150603, 'Peter Rabbit', '21S56')")
12
13 connection.close()
```

Resources:

https://www.tutorialspoint.com/sqlite/sqlite_python.htm

<https://stackoverflow.com/questions/10660411/difference-between-cursor-and-connection-objects>

<https://docs.python.org/3/library/sqlite3.html>

Chapter 17: NoSQL (Part 1: NoSQL & MongoDB)

Contents

Contents	1
Syllabus Learning Outcomes	1
1. Introduction to NoSQL	1
2. Differences between relational databases and NoSQL databases	2
3. How NoSQL addresses shortcomings of relational databases	2
4. Applications of SQL and NoSQL Databases	3
5. Introduction to MongoDB	4
5.1 Terms used in MongoDB	4
5.2 Document in MongoDB	5
5.3 Starting the MongoDB server	5
5.4 Using the MongoDB shell	5
6. MongoDB Shell Familiarisation Exercise (Optional)	7
References	8

Syllabus Learning Outcomes

3.3 Databases and Data Management

3.3.6 Understand how NoSQL database management system addresses the shortcomings of relational database management system (SQL).

3.3.7 Explain the applications of SQL and NoSQL.

1. Introduction to NoSQL

Relational databases (commonly referred to as SQL databases) work well with structured data since each table's schema (the precise description of the data to be stored and the relationships between them) is always clearly defined. However, with the increasing number of ways to gather and generate data, we often need to deal with unstructured

data. For example, a convenience store that frequently refreshes the services it provides may sell both mobile phones as well as groceries. To run the store, information about both mobile phones (e.g., model names and prices) and groceries (e.g., prices and descriptions) need to be stored. In the future, the store may also start selling mobile phone subscription plans as well. Storing all this data in the same relational database may not be easy. In this case, non-relational databases also referred to as NoSQL databases can offer a better choice.

There are four main types of NoSQL databases: key-value databases, document databases, wide-column databases, and graph databases. For the syllabus, we focus on MongoDB, a type of document database. Document databases work like a hash table, but each key can point to an embedded key-value structure, also known as a document, instead of just a single value. (Recall that in a hashtable, each key points to a single value or data item.)

2. Differences between relational databases and NoSQL databases

Relational databases have a fixed, predefined schema that their tables follow but NoSQL databases usually have no predefined schema, which is dynamic and can change easily.

Relational databases contain tables that must strictly adhere to standard schema definitions while NoSQL databases like MongoDB contain collections. The data types of each field in the table are fixed for relational databases but it is flexible for NoSQL databases like MongoDB.

Relational databases represent data in tables and rows while NoSQL databases usually store data as collections of documents.

For relational databases, joins are usually used to get data across tables, while for NoSQL databases like MongoDB there are usually no such joins. Thus it is easier to use relational databases for complex queries rather than NoSQL databases.

3. How NoSQL addresses shortcomings of relational databases

Predefined schema: Relational databases have a predefined schema that is difficult to change. Even if you wish to add a field to a small number of records, you still need to include the field for the entire table. Therefore, it can be difficult to support the processing of unstructured data using relational databases compared to NoSQL databases.

Hierarchical Data Storage: NoSQL supports hierarchical data storage where data is stored in a tree-like structure, i.e. the data is joined to their parent and child data. The data at the root node will be most frequently accessed and likely to be stored in more

expensive, faster storage devices. Less frequently used data is moved to cheaper, slower storage devices.

However, in relational databases like SQL, tables are joined to each other via primary and foreign keys. Hence, relational databases do not usually support hierarchical data storage. This means that the cost of storing data in a relational database is more expensive than storing the same amount of data in a NoSQL database.

Ease of scaling: Relational databases are mainly vertically scalable while NoSQL databases are mainly horizontally scalable. Vertically scalable means that improving the performance of a relational database server usually requires upgrading an existing server with faster processors and more memory. Such high-performance components can be expensive and upgrades are limited by the capacity of a single machine. On the other hand, horizontally scalable means that the performance of a NoSQL database can be improved by simply increasing the number of servers. This is relatively cheaper as mass produced average-performance computers are easily available at low prices.

Storage in multiple servers: Relational databases are stored in a server, which makes the database unavailable when the server fails. NoSQL databases are designed to take advantage of multiple servers so that if one server fails, the other servers can continue to support applications.

4. Applications of SQL and NoSQL Databases

The choice of whether to use a SQL or NoSQL database depends on the type of data being stored as well as the nature of tasks that the database is required to perform.

SQL databases should be used if:

- The data has a fixed schema.
- Complex and varied queries will be frequently performed.
- There will be a high number of simultaneous transactions.
- The transactions must conform to atomicity, consistency, isolation and durability (ACID) properties to protect the integrity of the database.

ACID Properties

- **Atomicity:** A change in the database must either be performed completely or not performed at all. The software must prevent a half-performed transaction from being saved.

- **Consistency:** A transaction must take a whole database from one consistent state to another consistent state, for example in a bank transfer transaction the amount of money in the whole system must be the same at the end of the transaction as it was at the beginning.
- **Isolation:** It is important that a transaction should be performed in isolation so that other users or processes cannot have access to the data concerned until the new consistent state has been committed (saved). In practice, this means that while an operation is being performed in a record, the record is locked. This may involve making the record invisible to others or it may only lock the record for writing. After the transaction is committed, the record may be unlocked again.
- **Durability:** Once a change has been made to a database, the change must not be lost because of any subsequent system failure or operator error. Ideally, the transaction is written immediately to secondary storage.

NoSQL databases should be used if:

- The data being stored has a dynamic schema, (i.e., unstructured data with flexible data types).
- Data storage needs to be performed quickly.
- There will be an extremely large amount of data (i.e., Big Data).

5. Introduction to MongoDB

5.1 Terms used in MongoDB

Here are the terms in MongoDB with the corresponding terms in SQL.

MongoDB Term	SQL Term
Database	Database
Collection	Table
Document	Record (Row)
Field	Field (Column)

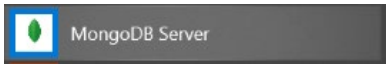
5.2 Document in MongoDB

A document in MongoDB looks similar to a python dictionary or JSONlike. In fact, we can operate on the document as though it was a dictionary in Python.

```
{
  "name": "Amanda",
  "age": "45"
}
```

5.3 Starting the MongoDB server

MongoDB is a NoSQL database. To start the MongoDB server, click on MongoDB Server icon in the Start menu or go to your MongoDB directory and run mongod.

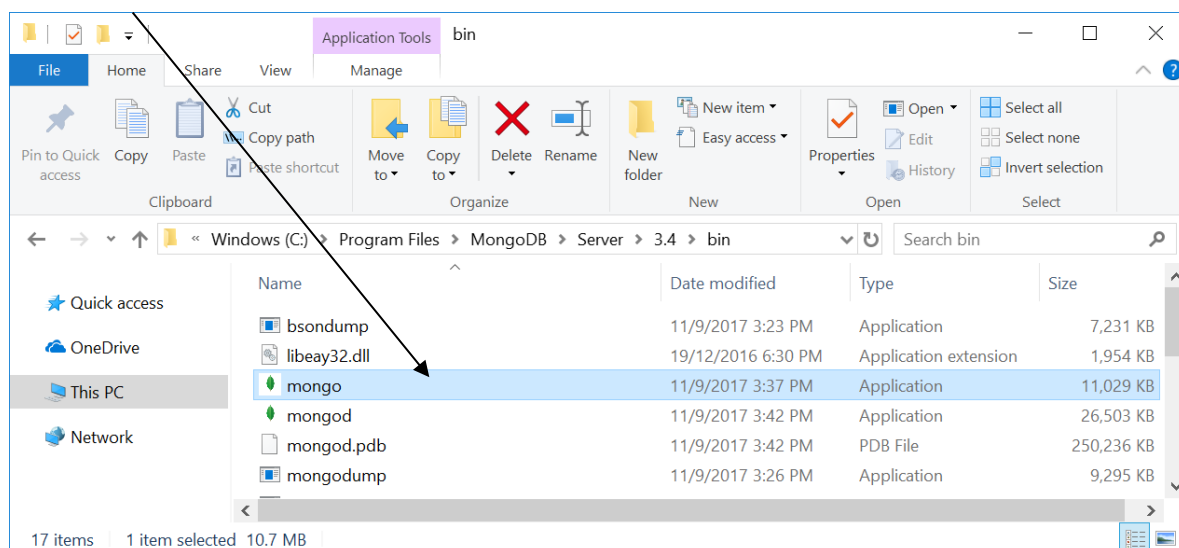


This will start the database server on your device. We can access the MongoDB server either using Mongo Shell or PyMongo. In the A-level environment, we will focus on using PyMongo.

5.4 Using the MongoDB shell

To interact with MongoDB databases, you need to start a connection to the server. To do so, use Windows Explorer and go to the MongoDB directory.

Start the mongo program.

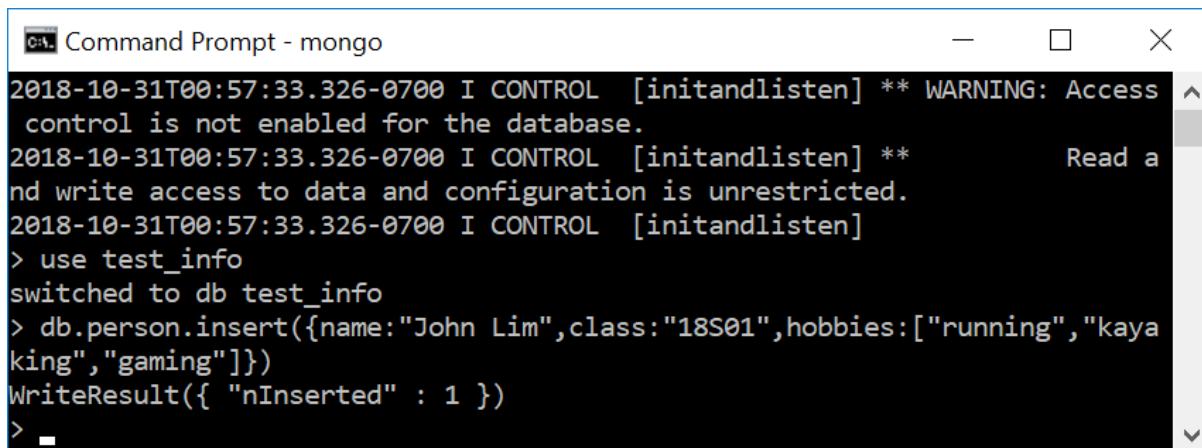


Starting the mongo shell program allows you to connect to the MongoDB server. You will then enter an interactive shell designed specifically for MongoDB. You can use variables and mathematical operations similar to Python shell.

Let's create a database with the use statement and then, insert documents into a collection using insert statement.

```
> use test_info
> db.person.insert({name:"John
Lim",class:"18S01",hobbies:[ "running","kayaking","gaming"]})
WriteResult({"nInserted" : 1})
```

The first statement tells MongoDB to use the database test_info. The second statement will insert a document into the person collection. Your screen will look like below. This states that one document has been inserted.



```
Command Prompt - mongo
2018-10-31T00:57:33.326-0700 I CONTROL [initandlisten] ** WARNING: Access
control is not enabled for the database.
2018-10-31T00:57:33.326-0700 I CONTROL [initandlisten] **           Read a
nd write access to data and configuration is unrestricted.
2018-10-31T00:57:33.326-0700 I CONTROL [initandlisten]
> use test_info
switched to db test_info
> db.person.insert({name:"John Lim",class:"18S01",hobbies:["running","kaya
king","gaming"]})
WriteResult({ "nInserted" : 1 })
> _
```

NB:

- The database test_info is created only when a document is inserted.
- The collection is automatically created when a document is added. You can manually create the person collection using the statement db.createCollection("person").

6. MongoDB Shell Familiarisation Exercise (Optional)

1. Run the following statements in the MongoDB.

```
db.person.insert({name:"Ben",class:"18S01"})  
WriteResult({"nInserted" : 1})
```

2. Notice that we now have 2 documents in the `person` collection. The first document was created in Section 5.4.
3. Run the following statement to insert the student Mary from class 18S02 with the hobby “running” into the `person` collection.

```
db.person.insert({name:"John Lim",class:"18S01",hobbies:["running"]})
```

4. Use the `find` function to find Ben’s document.

```
db.person.find({name:"Ben"})
```

5. Now, find the people with running as a hobby.

```
db.person.find({hobbies:"running"})
```

6. To find all documents, run the following statement.

```
db.person.find( {} )
```

7. Let’s find all information on students in class 18S01.

```
db.person.find({class:"18S01"})
```

8. To list all the names of students in class 18S01, run the following statement.

```
db.person.find({name:1}, class: "18S01")
```

9. To get all the names and classes of the students, run the following statement.

```
db.person.find({}, {name:1, class:1})
```

10. To remove the `person` collection, run the following statement.

```
db.person.drop()
```

11. To remove the test_info database, run the following statement.

```
db.dropDatabase()
```

12. To exit Mongo shell, type quit().

13. To shutdown the MongoDB server, press Ctrl-C.

References

1. MOE CPDD NoSQL Starter Kit
2. <https://www.thegeekstuff.com/2014/01/sql-vs-nosql-db/>
3. <https://www.3pillarglobal.com/insights/exploring-the-different-types-of-nosql-databases>
4. <https://www.mongodb.com/scale/types-of-nosql-databases>
5. Dan Sullivan (2015). NoSQL for Mere Mortals. Person Education
6. <https://www.mongodb.com/what-is-mongodb>
7. <http://api.mongodb.com/python/current/tutorial.html>
8. <https://www.mongodb.com/nosql-explained/nosql-vs-sql>
9. SQL vs NoSQL | What's the Difference?
<https://www.youtube.com/watch?v=Pf-9pjJK1e0>
10. MongoDB in 5 Minutes with Eliot Horowitz
<https://www.youtube.com/watch?v=EE8ZTQxa0AM>
11. MongoDB Explained in 10 Minutes | SQL vs NoSQL | Jumpstart
<https://www.youtube.com/watch?v=RGfFpQF0NpE>

Chapter 17: NoSQL (Part 2: MongoDB & PyMongo)

Contents

Contents	1
Syllabus Learning Outcomes	1
1. What is PyMongo?	2
2. Connecting to the MongoDB server	2
3. Create a Mongo database	3
4. Inserting documents	3
4.1 Inserting one document	3
4.2 Inserting multiple documents	4
4.3 Importing from a text file	4
4.4 Importing from a JSON file	5
5. Searching for documents	6
5.1 Find all documents	7
5.2 Find documents that satisfy certain criteria	7
5.3 Find documents that satisfy multiple criteria	8
6. Update documents	8
6.1 Add a new field	9
6.2 Remove a field	9
7. Remove documents	10
8. Remove a collection	12
9. Remove a database	12
References	12

Syllabus Learning Outcomes

3.3 Databases and Data Management

3.3.8 Use a programming language to work with NoSQL databases.

1. What is PyMongo?

MongoDB databases can be accessed using different programming languages like C, Java and Python. To access MongoDB databases using Python, we use the Python driver for MongoDB - PyMongo.

PyMongo is a Python library that allows us to perform basic CRUD operations on the MongoDB database via dictionary-like syntax.

Before running PyMongo, ensure that you have a MongoDB server instance running in the background. The MongoDB server window should remain open while you want to access the MongoDB database.

2. Connecting to the MongoDB server

To use PyMongo, create a `MongoClient` to connect to the MongoDB server. You may use Jupyter Notebook or Python IDLE type the codes. Avoid using cloud IDEs such as Google Colab as your MongoDB server is running on your localhost.

```
1 from pymongo import MongoClient
2 client = MongoClient() #connect on default host and port
3 client.close() #closes connection to the server
```

You can also specify the host and port. In the example below, we are connecting to the local MongoDB server which is usually at port 27017. You can see the port number when you start the MongoDB server.

```
1 from pymongo import MongoClient
2 client = MongoClient("Local_Host",27017)
3 client.close()
```

We can check the databases in the server using the method `database_names()`.

```
1 from pymongo import MongoClient
2 client = MongoClient()
3 #Retrieves the databases' names in Python list
4 dbs = client.database_names()
5 print("The databases in the Mongo DB server are:")
6 print(dbs)
7 client.close()
```

3. Create a Mongo database

As an example, let's create a database to store details on movie information.

```
1  from pymongo import MongoClient
2  client = MongoClient()
3  db = client.get_database("entertainment")
4  coll = db.get_collection("movies")
5  dbs = client.database_names()
6  print("The databases in the Mongo DB server are:")
7  print(dbs)
8  client.close()
```

We note that at this time, the databases in the server is the same as the previous part. Note that MongoDB waits until you have inserted at least one document before it actually creates the database and collection.

4. Inserting documents

We will look at two ways of inserting documents into a collection.

1. inserting one or more documents,
2. inserting by importing from a file.

For large amounts of data, it is more efficient to import the data from a file.

4.1 Inserting one document

We can insert documents one by one, using the `insert_one()` method. Notice that not all fields are required for insertion, as shown.

```
1  from pymongo import MongoClient
2  client = MongoClient()
3  db = client.get_database("entertainment")
4  coll = db.get_collection("movies")
5  #All fields are not null
6  coll.insert_one({"_id":1, "title":"Johnny Maths",
7  "genre":"comedy"})
8  #No _id, genre fields
9  coll.insert_one({"title":"Detection"})
10  dbs = client.database_names()
11  print("The databases in the Mongo DB server are:")
```

```

12 print(dbs)
13 cs = db.collection_names()
14 print("The collections in entertainment database are", cs)
15 client.close()

```

We should see one more database in the server after documents are added to the collection in the database. We can also see the collections (as a list) in the database. MongoDB will assign a unique `_id` to each document. You can customise the `_id` by stating it during the insertion process, as shown on line 6. This will also mean we cannot run this code again because a document with `_id = 1` already exists. However, if we comment out line 5, and run this code again, we will create a duplicate of `{"title": "Detection"}` with another `_id`.

4.2 Inserting multiple documents

To insert multiple documents, we can use the `insert_many()` method to insert a list of documents.

```

1  from pymongo import MongoClient
2  client = MongoClient()
3  db = client.get_database("entertainment")
4  coll = db.get_collection("movies")
5  #List of movies
6  movies = [{"title": "Badman", "genre": "adventure",
7            "year": 2015},
8            {"title": "Averages", "genre": ["science fiction", "adventure"],
9            "year": 2017},
10           {"title": "Octopus Man", "genre": "adventure", "year": 2017},
11           {"title": "Fantastic Bees", "genre": "adventure", "year": 2018},
12           {"title": "Underground", "genre": "horror", "year": 2014}]
13 coll.insert_many(movies)
14 coll_names = db.collection_names("entertainment")
15 print("Collections in entertainment database: ", coll_names)
16 client.close()

```

4.3 Importing from a text file

The program below reads from a delimited text file `input.txt` and inserts the documents into the database.

```
input.txt
```

```
Amanda,45
```

```
Bala,28
```

```
Charlie,33
```

```
Devi,29
```

```
1 import pymongo, csv
2
3 client = pymongo.MongoClient("127.0.0.1", 27017)
4 db = client.get_database("entertainment")
5 coll = db.get_collection("users")
6
7 with open('input.txt') as csv_file:
8     csv_reader = csv.reader(csv_file, delimiter=',')
9     for row in csv_reader:
10         coll.insert_one({"name":row[0], "age":row[1]})
11 client.close()
```

4.4 Importing from a JSON file

A JSON file stores simple data structures and objects in JavaScript Object Notation (JSON) format, which is a standard data interchange format. It is primarily used for transmitting data between a web application and a server. JSON files are lightweight, text based, human-readable, and can be edited using a text editor.

To import from JSON file, use the `load()` function. The program below reads from a delimited text file `data.json` and inserts the documents into the database.

```
data.json
[
  {
    "name": "Amanda",
    "age": "45"
  },
  {
    "name": "Bala",
    "age": "28"
  },
  {
    "name": "Charlie",
    "age": "33"
  },
  {
    "name": "Devi",
    "age": "29"
  }
]
```

```
1 import pymongo, json
2 client = pymongo.MongoClient()
3 with open('data.json') as file:
4     data = json.load(file)
5 client['entertainment']['users'].insert_many(data)
6 client.close()
```

For the code above, we no longer use `get_database()` and `get_collection()` to access the required collection. In Line 5, we are using syntax similar to array indexing to access the collection.

5. Searching for documents

Similar to `SELECT * FROM <table>` in SQL, we need a method to display the contents of a collection in NoSQL.

The following is a list of commonly used query operators. This is also listed in your exam's Insert document.

\$eq	Equals to
\$gt	Greater than
\$gte	Greater than or equal to
\$lt	Less than
\$lte	Less than or equal to
\$ne	Not equal to
\$in	In a specified list
\$nin	Not in a specified list
\$or	Logical OR
\$and	Logical AND
\$not	Logical NOT
\$exists	Matches documents which has the named field

5.1 Find all documents

```

1  import pymongo
2  client = pymongo.MongoClient()
3  db = client.get_database("entertainment")
4  coll = db.get_collection("movies")
5  result = coll.find()
6  for doc in result:
7      print(doc)
8  print(f"There are {result.count()} documents in the
9  collection")
10 client.close()
```

The method `find()` in line 5 returns a `Cursor` of all the documents in the `movie` collection. The results can be printed with a loop. We can think of the `Cursor` object as a collection of all the documents. The `count()` method gives the number of documents in the `movie` collection.

5.2 Find documents that satisfy certain criteria

To find all the movies in the adventure genre, we can do the following.

```

1  import pymongo
2  client = pymongo.MongoClient()
3  db = client.get_database("entertainment")
4  coll = db.get_collection("movies")
```

```

5 result = coll.find({"genre":"adventure"})
6 for doc in result:
7     print(doc)
8     client.close()

```

5.3 Find documents that satisfy multiple criteria

To find all the movies in the adventure genre that are released after 2016, we can do the following.

```

1 import pymongo
2 client = pymongo.MongoClient()
3 db = client.get_database("entertainment")
4 coll = db.get_collection("movies")
5 query1 = {"genre":"adventure","year":{"$gt":2016}}
6 result = coll.find(query1)
7 for doc in result:
8     print(doc.get("title"))
9     client.close()

```

Line 5 of the code creates the query to find the documents with adventure genre and year greater than 2016. It can be rewritten using the `$and` operator as well.

```
query2 = {'$and': [{'genre': 'adventure'}, {'year': {'$gt': 2016}}]}
```

Each document is just a Python dictionary, so you can use the usual built-in methods for dictionary. For example, line 8 uses the `get()` method to retrieve the value of title. This allows you to extract the value for a particular field in the document.

6. Update documents

To modify the content in the database, use the `update_one()` method to modify the first document that matches the query, or the `update_many()` method to modify all documents that matches the query.

The program below illustrates these.

```

1 import pymongo
2 client = pymongo.MongoClient()

```

```

3 db = client.get_database("entertainment")
4 coll = db.get_collection("movies")
5 result = coll.find()
6 print("Before Update:")
7 for doc in result:
8     print(doc)
9 print()
10
11 search = {'year':{'$gt':2016}}
12 update = {'$set':{'year':2015}}
13 coll.update_one(search, update) #update one document
14 result = coll.find()
15 print("After update_one:")
16 for doc in result:
17     print(doc)
18 print()
19
20 coll.update_many(search, update) #update many documents
21 result = coll.find()
22 print("After update_many:")
23 for doc in result:
24     print(doc)

```

6.1 Add a new field

In the code above, line 12 uses `$set` to set all the year values greater than 2016 to be 2015. The search query in line 11 says that we are going to update this for movies that are released after 2016.

The `$set` operator can also be used to add a new field with the specified value if the field does not exist. If we specify multiple field-value pairs, the operator will update or add each field as required.

6.2 Remove a field

Besides adding new fields or updating existing fields, we can also remove fields using the `$unset` operator.

```

1  import pymongo
2  client = pymongo.MongoClient()
3  db = client.get_database("entertainment")
4  coll = db.get_collection("movies")
5  result = coll.find()
6  print("Before Update:")
7  for doc in result:
8      print(doc)
9  print()
10
11  search = {'year':{'$eq':2015}}
12  update = {'$unset':{'year':0}} #remove fields
13  coll.update_many(search, update)
14
15  result = coll.find()
16  print("All documents in movies collection after unset:")
17  for doc in result:
18      print(doc)
19
20  client.close()

```

Take a look at line 12. Note that even though `$unset` operator removes the given fields, there is still a requirement to have a second argument, thus `0` is placed even though it won't be updated.

7. Remove documents

To delete documents, you can use the `delete_one()` method to delete the first document that matches the given condition, or `delete_many()` method to delete all the documents that match the condition.

```

1  import pymongo
2  client = pymongo.MongoClient("127.0.0.1", 27017)
3  db = client.get_database("entertainment")
4  coll = db.get_collection("movies")
5

```

```
6 result = coll.find()
7 print("All documents in movies collection:")
8 for document in result:
9     print(document)
10
11 coll.delete_one({'year':2015}) #delete first relevant doc
12
13 result = coll.find()
14 print("All documents in movies collection after deleting
15 one:")
16 for document in result:
17     print(document)
18
19 coll.delete_many({'year':2015}) #delete many documents
20
21 result = coll.find()
22 print("All documents in movies collection after deleting
23 all:")
24 for document in result:
25     print(document)
26
27 client.close()
```

8. Remove a collection

To clear a collection, you can use `drop_collection()` as shown below.

```
1  import pymongo
2  client = pymongo.MongoClient()
3  db = client.get_database("entertainment")
4  coll = db.get_collection("tv")
5
6  coll.insert_one({'year':2015})
7
8  cs = db.collection_names()
9  print("All collections in entertainment database:")
10 print(cs)
11
12 db.drop_collection("tv") #remove collection
13 cs = db.collection_names()
14 print("After tv collection is dropped:")
15 print(cs)
16
17 client.close()
```

9. Remove a database

To remove the entire `entertainment` database, you can use the following statement. That will remove all the collections and the documents within it.

```
client.drop_database("entertainment")
```

References

1. MOE CPDD NoSQL Starter Kit
2. <https://www.analyticsvidhya.com/blog/2020/02/mongodb-in-python-tutorial-for-beginners-using-pymongo/>
3. https://github.com/giffy/MongoDB_PyMongo_Tutorial

4. <https://www.mongodb.com/what-is-mongodb>
5. <http://api.mongodb.com/python/current/tutorial.html>

Chapter 18 Part 1: Web Applications (Principles)

Syllabus Learning Outcomes	2
1. Native Applications	2
2. Web Applications	2
3. “Newer” Types of Applications	2
3.1 Hybrid Applications	2
3.2 Progressive Web Applications (PWA)	3
4. Native Applications vs Web Applications	3
5. Usability Principles in the Design of Web Applications	5
5.1 Visibility of system status	5
5.2 Match between system and the real world	5
5.3 User control and freedom	5
5.4 Consistency and standards	5
5.5 Error prevention	6
5.6 Recognition rather than recall	6
5.7 Flexibility and efficiency of use	6
5.8 Aesthetic and minimalist design	6
5.9 Help users recognize, diagnose, and recover from errors	7
5.10 Help and documentation	7
References	7
6. ASCII	8
7. Unicode	8
8. Differences Between ASCII and Unicode	9
References	9

Syllabus Learning Outcomes

4.2 Web Applications

Understand the concepts and techniques for developing web applications.

4.2.1 Describe the differences between web applications and native applications.

4.2.2 State and apply usability principles in the design of web applications.

3.2 Character Encoding

Understand the use of ASCII code and Unicode to represent characters.

3.2.1 Give examples of where or how Unicode is used.

3.2.2 Use ASCII code in programs.

1. Native Applications

Native applications are installed directly on the device that can work with no internet connection depending on the nature of the application. They are developed specifically for a particular platform or operating system. They can also be designed to take advantage of the device features such as the power of the processor, specific hardware such as GPS and other components of the system such as the notification system and gestures.

Examples: Calculator,

2. Web Applications

Web applications are internet-enabled applications that can be accessed through the device's web browser and do not depend upon a specific operating system. They are typically developed as web pages in HTML and CSS while the interactive components of the applications are developed in programming languages such as JQuery and JavaScript. Increasingly, many websites today use HTML5 which blur the distinction between web applications and regular web pages.

Examples: LuminPDF,

3. "Newer" Types of Applications

3.1 Hybrid Applications

These are web applications that are packaged in a native wrapper/container/shell and are usually developed using HTML5, CSS3 and JavaScript. These applications combine the elements of both native applications and web applications. As such, they need to be

installed on the device. The device's web browser is used to render the HTML etc while the native Application Programming Interface (API) is used to access the device-specific hardware.

Examples: Gmail/Meet,

3.2 Progressive Web Applications (PWA)

These are similar to hybrid applications in that they both combine the elements of native and web applications. However, PWAs can be accessed directly through a web browser and be added as a shortcut on your home screen, which can then be accessed offline.

PWAs currently are not able to access internal APIs and device-specific resources.

Examples: Spotify,

4. Native Applications vs Web Applications

Deployment and maintenance (updates) for a web-based application require deployment on a single set of server machines. However, for native and desktop applications, the deployment and any maintenance/patch has to be done on individual client machines separately.

Web applications can also be accessed from anywhere (most locations), so there is no location constraint. However, as native applications are confined to a standalone machine, they can only be accessed from the machines they are deployed in.

Web applications thus, rely heavily on internet connectivity, for their operations. And thus, developers need to account for network reliability before deciding to embark on such projects. Desktop applications, on the other hand, don't require the internet for their operations. Some applications just require internet connectivity at the time of update.

Web applications are also often platform-independent, i.e. they can work on different types of platforms with the only requirement being the provision of a web browser. Desktop applications, on the other hand, need to be developed separately for different platform machines. (Windows, Linux, Unix, Mac etc)

Web applications lend themselves to higher security risks as they are inherently designed to increase accessibility. This is a trade off that developers must take note and circumvent accordingly with the latest technologies and cyber security intelligence. Desktop applications, on the other hand, have better authorization and administrators have better control, hence they are more secure.

The table below shows a summary of the above points of difference.

Native Applications	Web Applications
Developed for a specific operating system or platform. As native apps are confined to a standalone machine, they can only be accessed from the machines they are deployed in.	Developed to be accessed via the device's internet browser. Web applications can be accessed from anywhere (most locations), so there is no location constraint.
Need to be installed in the device to function. Native applications need to be developed separately for different platform machines. (Windows, Linux, Unix, Mac etc)	No need to be downloaded or installed. Web applications are platform - independent, they can work on different types of platforms with the only requirement of a web browser.
Have access to system resources such as GPS, camera etc.	No access to system resources such as GPS, camera etc.
May be able to work offline. Native applications don't require the internet for their operations. Some applications just require internet connectivity at the time of update.	Need an active internet connection to work. Web applications rely heavily on internet connectivity, for their operation.
Comparatively faster.	Comparatively slower, depending on the internet connection.
Relatively safer and more secure. Native applications, on the other hand, have better authorization and administrators have better control, hence more secure.	Relatively less secure. Web applications are at higher security risks as they are inherently designed to increase accessibility.
May incur maintenance and update costs. Deployment and any maintenance/patch are done on individual client machines separately.	Updated automatically. Deployment and maintenance (updates) for a web-based application require deployment on a single set of server machines.

5. Usability Principles in the Design of Web Applications

Adapted from: [Jakob Nielsen's 10 general principles for interaction design](#)

Videos: [Explanatory videos](#)

5.1 Visibility of system status

The application design should always update the users of the system's current status clearly. Prompt and meaningful feedback should be provided so that the users can determine the next steps. Such predictable interactions will create trust in the system.

5.2 Match between system and the real world

The terms, images, icons, concepts and words used in the application should also be familiar to the user and their mental models. This can be achieved by following realworld conventions so that the information can appear in a natural and logical order which leads to desired outcomes. This is what we call natural mapping. With such intuitive design, users can easily learn how to use the interface and therefore, enhances user experience.

5.3 User control and freedom

Applications that follow good usability principles can accommodate users' mistakes by allowing them to undo and redo easily. They also provide a clear way to exit any unwanted state or interaction. Such design gives users a sense of freedom and control and avoids frustration of getting stuck at a certain state.

5.4 Consistency and standards

It is also important to improve users' learnability by ensuring internal and external consistency in using words, symbols and actions. Internal consistency refers to ensuring that your application design follows the convention within a single digital product or a family of products. External consistency refers to following the established industry conventions.

Jakob's Law of Internet UX states that people spend most of their time using their digital products other than yours. As such, users already have certain expectations in using their digital devices. Failure to maintain consistency may increase the users' cognitive load as they are forced to learn something new or contrary to what they are used to.

5.5 Error prevention

Web applications should also be developed in such a way that potential problems can be prevented. This can be done by either eliminating such error-prone conditions or by confirming with the users before they commit to a particular action.

In general, there are two types of errors: slips and mistakes.

Slips are unconscious errors caused by inattention. Slips can be avoided by providing helpful constraints and good defaults.

Mistakes are conscious errors based on a mismatch between the user's mental model and the design. Mistakes can be prevented or minimised by allowing users to undo certain actions as well as by notifying or warning the users before they execute the actions. It is also paramount that the design minimises the users' memory load to lower the probability of users making mistakes.

5.6 Recognition rather than recall

Humans have limited short-term memories. As such, interfaces that promote recognition reduce the amount of cognitive effort required from the users.

Users' memory load can be minimised by making the elements, actions and options in your application visible and easily retrievable. The users should not have to remember information from one part of the interface to another. For example, if the users need certain information to use the application such as field labels or menu items, these elements should be visible and easily retrievable to the users.

5.7 Flexibility and efficiency of use

A well-designed application allows differentiation for both novice users and expert users. Accelerators like keyboard shortcuts and touch gestures that are hidden from novice users may speed up the interaction for the expert and experienced users. The application should also be developed in such a way that the users can personalise how they prefer to interact with the application by customizing or tailoring their frequent actions, content and other functionality. An application that allows customisations will provide users to make selections about how they want the product to work.

5.8 Aesthetic and minimalist design

The content and visual design of the User Interface (UI) should also focus on the essentials. The interfaces should not contain irrelevant content or rarely-needed information. This is because every extra unit of information in an interface competes with

the relevant units of information and diminishes their relative visibility. As such, it is important not to distract the users with unnecessary elements and to ensure that the visual elements of the interface support the user's primary goals.

5.9 Help users recognize, diagnose, and recover from errors

Errors cannot be eradicated completely. As such, helpful error messages are vital in web applications. These messages should attract users' immediate attention by ensuring that the messages follow the traditional visuals such as bold, red text. The messages should also be expressed in simple language and not in error codes or other technical jargon. The system should then indicate the problem and suggest constructive suggestions to solve the problem.

5.10 Help and documentation

Help and documentation help users to understand how to use the applications and to complete their tasks. These should be easy to search and tailored to the context in which the users are currently in. The help and support should also be concise by listing the concrete steps that need to be taken.

References

[The Encyclopedia of Human-Computer Interaction](#)

[Jakob Nielsen's 10 general principles for interaction design](#)

UI elements guide: <https://www.usability.gov/how-to-and-tools/methods/user-interface-elements.html>

UI best practices:

<https://www.usability.gov/what-and-why/user-interface-design.html>

<https://clearbridgemobile.com/mobile-app-development-native-vs-web-vs-hybrid/>

6. ASCII

ASCII stands for 'American Standard Code for Information Interchange'. It was defined in 1963 and was one of the most common character sets used. The ASCII standard defines how numbers are used to represent common characters that can be typed using a keyboard, such as uppercase and lower-case letters.

Using ASCII, only 128 distinct characters can be represented because ASCII codes are defined to be exactly seven bits long. This means that the value of ASCII codes can only vary from $(000\ 0000)_2$ to $(1111111)_2$, which is the same as $(00)_{16}$ to $(7F)_{16}$ or 0 to 127.

The first 32 ASCII codes (0 to 31) represent non-printing characters that are not written symbols or the space character. These codes are instead used to control devices or to represent special input such as typing the Backspace and Esc keys on the keyboard. The last ASCII code 127 is also used to represent the special input of typing the Delete key on the keyboard.

Hexadecimal digits are often used to express ASCII codes as only two hexadecimal digits are needed to express any ASCII code. This is more compact than using three denary digits or seven binary digits.

[ASCII Code Website](#)

7. Unicode

Besides ASCII, there is another expanded standard called Unicode. Unicode provides a unique number for every character, no matter what the platform, program or language is. Therefore, Unicode is used to represent characters in languages other than English such as Tamil and Mandarin.

While ASCII codes are defined to be exactly seven bits long, Unicode is not as restrictive. In Unicode, the number of bits used to represent each character can vary from 8 to 32 bits depending on the encoding scheme used. This flexibility means that Unicode can be used to represent over a million unique characters from many different written languages all over the world, which makes it a preferred choice for use over the ASCII standard.

The largest number that can be used to represent a character in Unicode is 1,114,111, which is $(10000111111111111111)_2$ and requires 21 binary digits. This means that if Unicode values are stored directly as binary numbers, each character would take up 21 bits. Doing so would be wasteful, however, as not all Unicode characters are used equally often. The designers of Unicode realised that it would be more efficient to use fewer bits for frequently used characters at the expense of using more bits for rarely used

characters. This explains why the number of bits used to represent a Unicode character can vary between 8 and 32.

[Unicode official website](#)

8. Differences Between ASCII and Unicode

#create a comparison table or otherwise here.

#You may use the links given above as well as in the References section to help you.

References

[ASCII and Unicode](#)

[ASCII vs Unicode](#)

[Unicode in Python](#)

<https://www.bbc.co.uk/bitesize/guides/zp73wmn/revision/5>

<https://www.bbc.co.uk/bitesize/guides/zsnbr82/revision/5>

Chapter 18 Part 2: Web Applications (HTML)

Instructional Objectives:

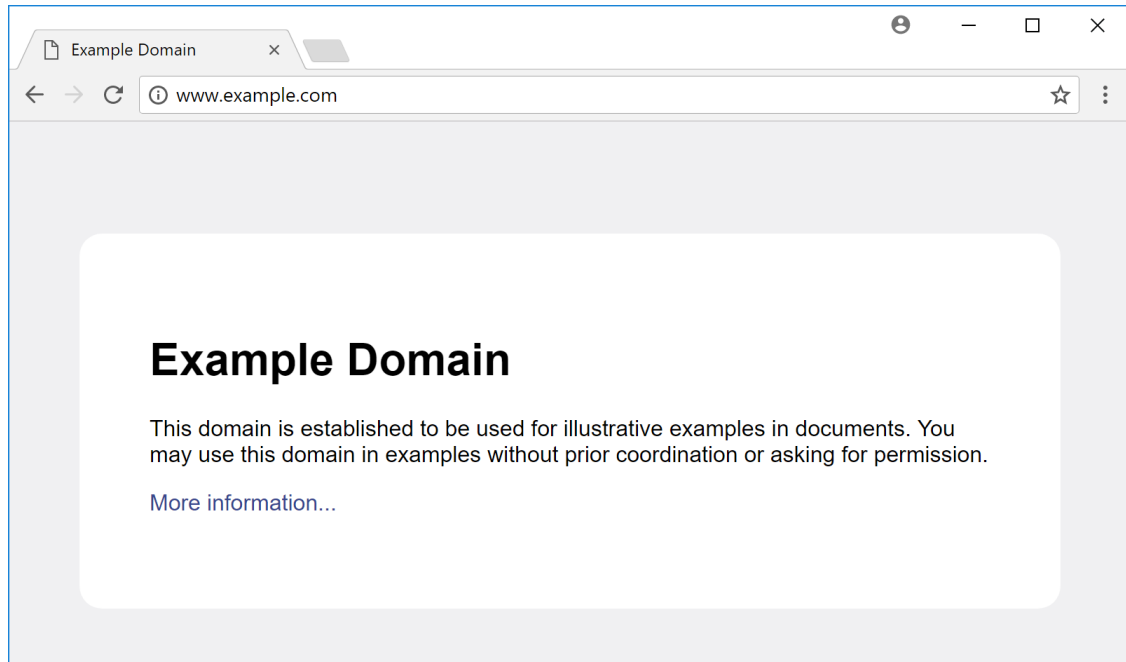
By the end of this task, you should be able to:

- State that web pages are written using the HyperText Markup Language (HTML) and Cascading Style Sheets (CSS) language
 - Use the "View Source" feature of a web browser to examine the HTML source code of any web page
 - Identify the tags, attributes, character references and comments that are present in a given static HTML document
 - Explain how HTML only describes the structure of a web page (in terms of elements) and not its presentation
 - Distinguish between tags for normal elements that require an end tag and tags for void elements that do not have an end tag
 - Use the `<html>`, `<head>`, `<title>`, `<body>`, `<h1>`, `<h2>`, `<h3>`, `<p>`, `<a>`, `<img>`, `<table>`, `<tr>`, `<th>`, `<td>`, `<form>`, `<input>` and `<textarea>` tags with relevant attributes to create web pages that meet a given set of requirements without producing any syntax errors
 - Use the Developer Tools of a web browser to examine how form data is encoded differently for HTTP GET requests and HTTP POST requests
-

Part 1: Behind the Scenes of a Web Page

Open a web browser such as Google Chrome and visit www.example.com. You should see a simple web page as shown in the following screenshot.

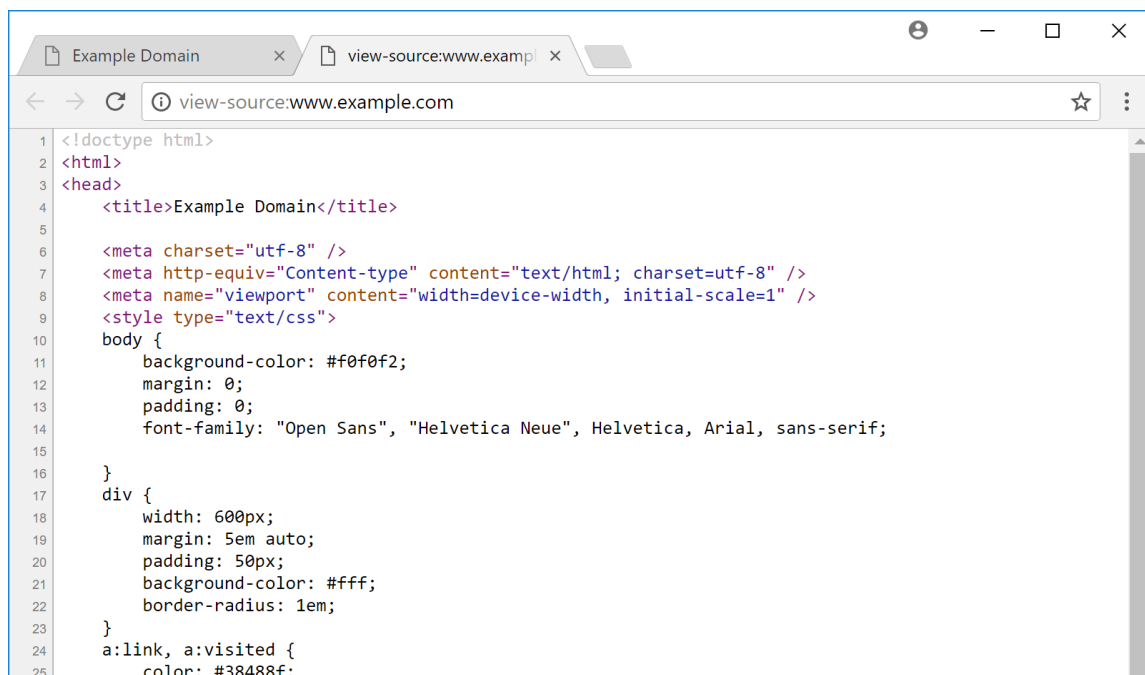
Chapter 18 Part 2: Web Applications (HTML)



How do you think web pages like this one are made?

Just like how programs are written in a programming language such as Python, web pages are written using the **HyperText Markup Language (HTML)**. Unlike programming languages that are specialized for describing step-by-step instructions, HTML is specialized for describing the structure of web pages.

To view the HTML source code of a web page in Chrome, press Ctrl-U. (Alternatively, right-click and select "View page source".) Try this now for the web page on www.example.com:



Chapter 18 Part 2: Web Applications (HTML)

- 1 Try to find the portion of the HTML that represents the "More information..." link shown below and copy it into the blank space provided:



```
<p><a href="http://www.iana.org/domains/example">More  
information...</a></p>
```

Examine the HTML source code on the web browser. Do you see that the contents of the web page are surrounded by text enclosed in angle brackets (i.e., < and >)?

```
<h1>Example Domain</h1>
```

The text surrounded by angle brackets are special processing instructions for the web browser called **tags**. These tags are underlined for clarity below:

```
<h1>Example Domain</h1>
```

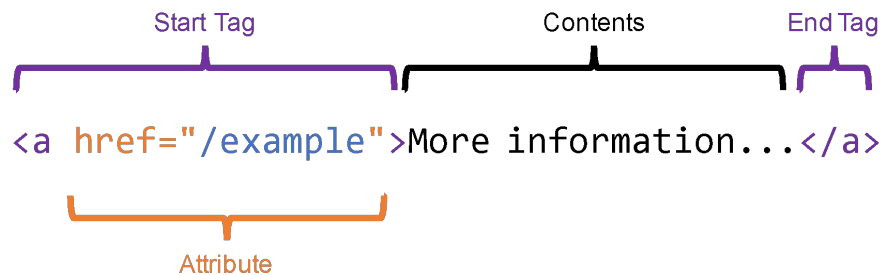
You may also notice that the first part of the HTML document includes some code that appears to be written in another computer language. This is a **style sheet** written in the **Cascading Style Sheets (CSS)** language that controls the web page's appearance. Without style sheets, web pages have a very limited appearance by default (i.e., black *Times New Roman* text on a white background).

In this practical task, we will focus on content first before looking at layout with CSS in the following practical tasks.

Chapter 18 Part 2: Web Applications (HTML)

Part 2: Anatomy of a HTML Document

You have learnt to identify tags in a HTML document as portions of text that are enclosed in angle brackets. Tags can be further classified into **start tags** and **end tags**. Start tags may also have one or more **attributes**:



- 2 How do you tell the difference between start tags and end tags?

End tags always start with a forward slash (/).

Like Python, certain characters (such as the angle brackets < and >) have special meanings in HTML and would cause a syntax error if they are used without escaping them. In HTML, escape codes are called **character references** that start with an ampersand & and end with a semicolon ;.

Some common character references are as follows:

Character	Ampersand	Less-Than Sign	Greater-Than Sign	Quotation Mark
	&	<	>	"
Character Reference	&	<	>	"

- 3 Circle all start tags and underline all end tags in the following HTML document. (Ignore the first line that reads <!doctype html> for now. It will be explained in a later part of this practical task.)

```
<!doctype html>
<html>
<head><title>Welcome Page</title></head>
<body>
  <p>Welcome to our R&D department.</p>
  
</body>
</html>
```

Chapter 18 Part 2: Web Applications (HTML)

- 4 In the previous document, which start tags do not have matching end tags?

Only .

- 5 Circle the syntax error in the following HTML snippet.

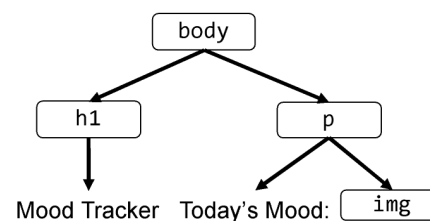
```
<p>I was sent to A&E in an ambulance.</p>
```

Rewrite the snippet and fix the error so it works as intended on all browsers:

<p>I was sent to A&E in an ambulance.</p>

HTML tags are used to describe the structure of a web page by organizing its contents into a tree of **elements**. In general, each start tag corresponds to a single element. For instance, the HTML snippet on the left also describes the tree of elements on the right:

```
<body>
  <h1>Mood Tracker</h1>
  <p>
    Today's Mood:
    
  </p>
</body>
```



You may have noticed that some start tags always have a matching end tag while other start tags do not. Start tags with a matching end tag (such as <body>, <h1> and <p>) correspond to **normal elements** that may contain a combination of text contents and other elements. On the other hand, start tags that do not have a matching end tag (such as and <input>) correspond to **void elements** that must not contain any other content.

Finally, like Python programs, HTML documents may include comments that are readable by humans but are ignored by the web browser. Comments in HTML must

Chapter 18 Part 2: Web Applications (HTML)

start with `<!--` and end with `-->`. Furthermore, the contents of the comment must NOT contain two consecutive hyphen characters `--`.

- 6 For each code snippet below, place a tick in the corresponding box if the snippet is a HTML comment with no syntax errors.

Code Snippet	Valid?
<code><!-- The markup language of the web -- HTML. --></code>	
<code><!-- The markup language of the web - HTML. --></code>	✓
<code><!-- The markup language of the web - HTML. -></code>	

Part 3: Your Own Web Page

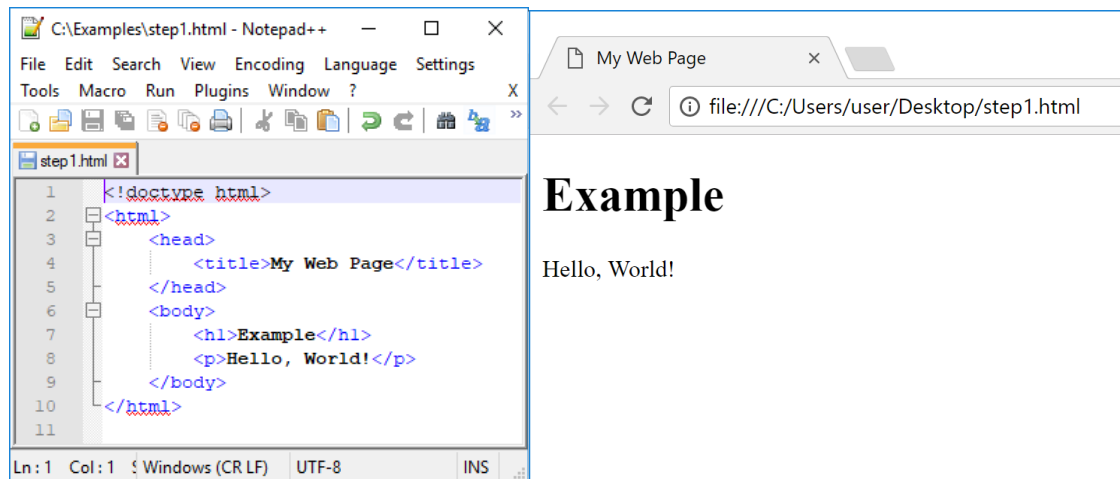
Now that we know about the basics of HTML, it is time to write our own web page. Like Python programs, HTML documents are just plain text files except that they use a `.html` extension instead of a `.py` extension. While we can write HTML using any text editor, we will use Notepad++ so that our code will be displayed with syntax highlighting.

Open Notepad++, create a new file and save it in your user directory as `example.html`. Then enter the following HTML document:

```
<!doctype html>
<html>
  <head>
    <title>My Web Page</title>
  </head>
  <body>
    <h1>Example</h1>
    <p>Hello, World!</p>
  </body>
</html>
```

Save the document again and double-click `example.html` in Windows Explorer to open it in a web browser. You should see something similar to the following:

Chapter 18 Part 2: Web Applications (HTML)



Congratulations! You have just created a web page.

Just like how Python has many functions, HTML has a vocabulary of tags such as `<head>`, `<title>`, `<body>`, `<h1>` and `<p>` that you should become familiar with:



The remainder of this practical task will introduce these HTML tags by organising them into five groups: required tags, tags for describing sections, tags for describing text and media, tags for describing tables and tags for describing forms.

Part 4: Required Tags

All HTML documents have a number of required elements. First, every HTML document must start with a **doctype** that looks like this:

```
<!doctype html>
```


Chapter 18 Part 2: Web Applications (HTML)

This declares that the remainder of the file is a HTML document. (Note that the doctype is technically not considered a tag. It tells the web browser the version of HTML used in the document. This is important as the web browser supports multiple versions of HTML.)

After the doctype, the remainder of the document must be enclosed between a `<html>` start tag and `</html>` end tag:

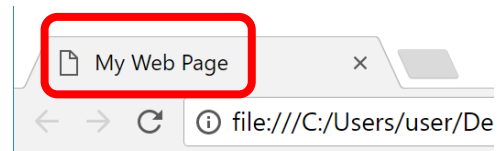
```
<!doctype html>
<html>
    ...document goes here...
</html>
```

This describes a single `html` element that serves at the **root element** of the document's structure. Inside the `html` element, there must be a `head` element for metadata (data that describes other data), followed immediately by a `body` element for the main content:

```
<!doctype html>
<html>
    <head>
        ...metadata goes here...
    </head>
    <body>
        ...main content goes here...
    </body>
</html>
```

Inside the `head` element, there must be a `title` element to set the title of the web page that appears in the browser window or tab:

```
<!doctype html>
<html>
    <head>
        <title>My Web Page</title>
    </head>
    <body>
        ...main content goes here...
    </body>
</html>
```



Chapter 18 Part 2: Web Applications (HTML)

These required tags serve to ensure that all HTML documents are contained inside a single `<html>` element and that metadata is placed in a `<head>` element, separate from the main content in the `<body>` element. While `<title>` is the only metadata tag that is required inside `<head>`, for the next practical task we will learn about other examples of metadata that can also be included.

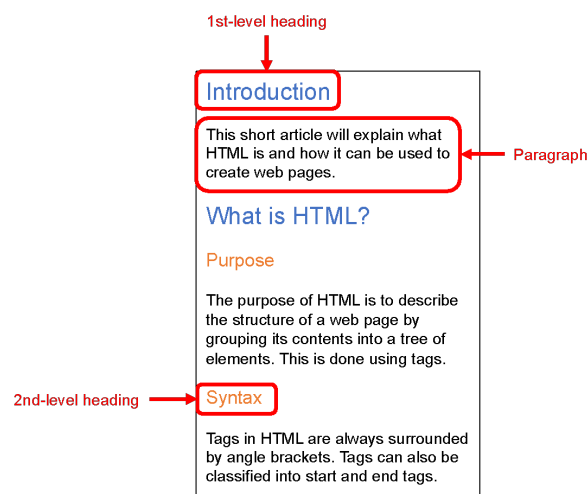
- 7 For each tag in the table below, circle "Normal" if it describes a normal element and circle "Void" if it describes a void element:

Tag	Description	Element Type
<code><html></code>	Root element	Normal / Void
<code><head></code>	Metadata	Normal / Void
<code><title></code>	Document title	Normal / Void
<code><body></code>	Main content	Normal / Void

Part 5: Structural Tags

Now, let us focus on the main content that goes into the `<body>` tag.

For ease of reading and navigation, long pieces of content are typically divided into logical parts such as headings and paragraphs.



Chapter 18 Part 2: Web Applications (HTML)

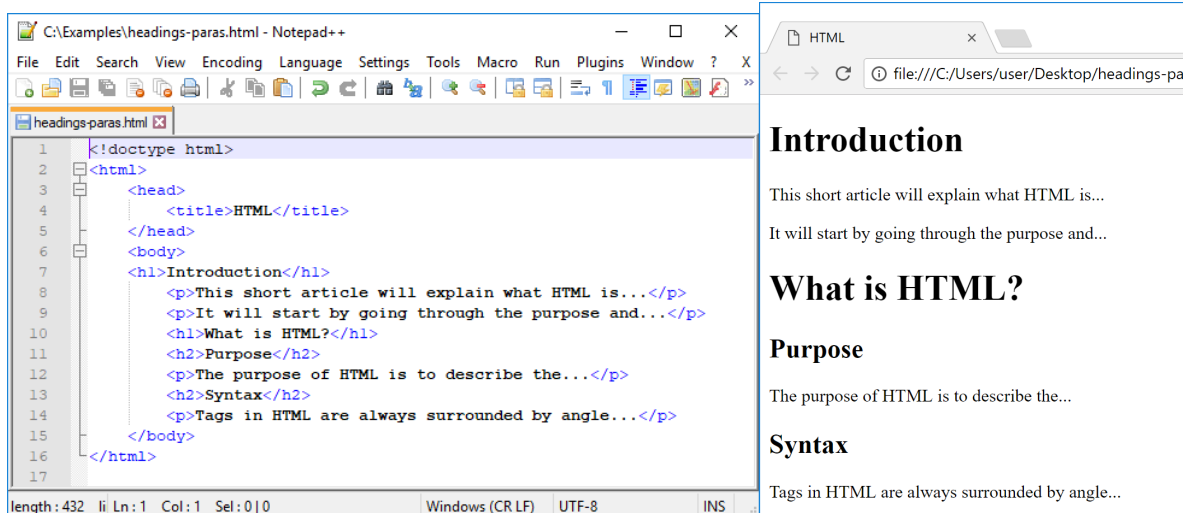
HTML provides us with 6 tags, namely <h1>, <h2>, <h3>, <h4>, <h5> and <h6>, to describe 1st-level headings, 2nd-level headings, 3rd-level headings and so on:

```
<body>
  <h1>Introduction</h1>
  <h1>What is HTML?</h1>
  <h2>Purpose</h2>
  <h2>Syntax</h2>
</body>
```

Content can also be organized into paragraphs using the <p> tag:

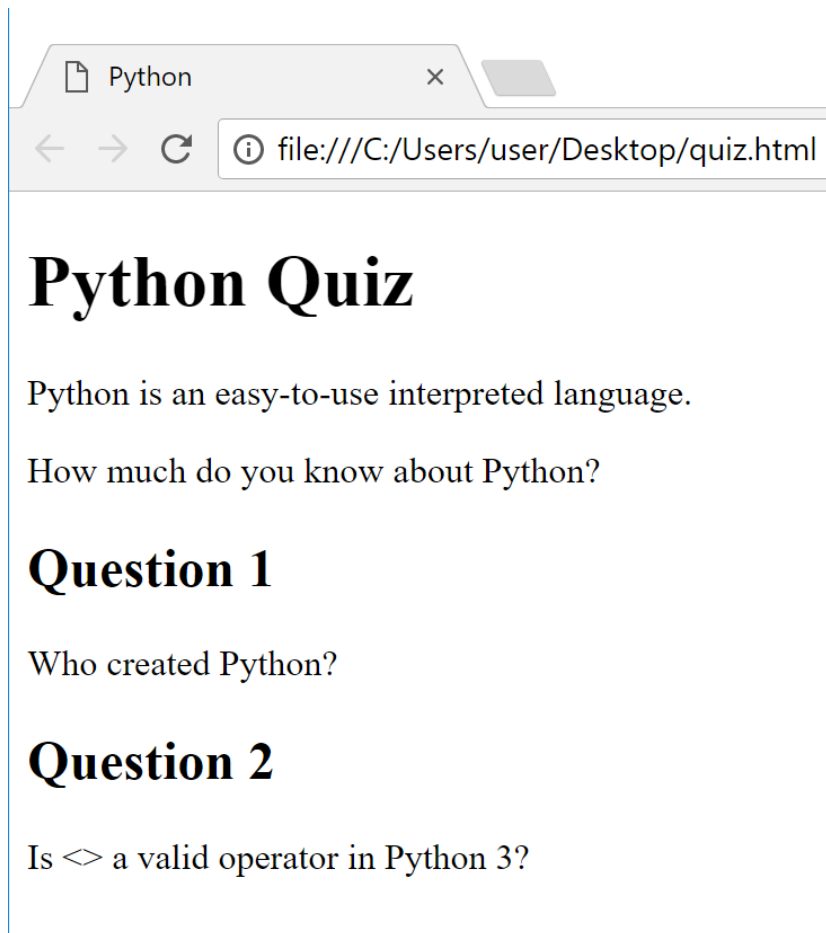
```
<body>
  <h1>Introduction</h1>
  <p>This short article will explain what HTML is...</p>
  <p>It will start by going through the purpose and...</p>
  <h1>What is HTML?</h1>
  <h2>Purpose</h2>
  <p>The purpose of HTML is to describe the...</p>
  <h2>Syntax</h2>
  <p>Tags in HTML are always surrounded by angle...</p>
</body>
```

Use a text editor to type out the above HTML, add any required tags (e.g., give the document a title of "HTML") and save the result as headings-paras.html. Double-click the document in Windows Explorer to open it in a web browser and verify that the result is similar to the screenshot on the right:



Chapter 18 Part 2: Web Applications (HTML)

- 8 Use only the `<html>`, `<head>`, `<title>`, `<body>`, `<h1>`, `<h2>` and `<p>` tags to create the following web page and save it as `quiz.html`:



- 9 For each tag in the table below, circle "Normal" if it describes a normal element and circle "Void" if it describes a void element:

Tag	Description	Element Type
<code><h1></code> to <code><h6></code>	Heading	Normal / Void
<code><p></code>	Paragraph	Normal / Void

Part 6: Text and Media Tags

Previously, we mentioned that CSS should be used to control the appearance of web pages. However, for basic text formatting, by convention we can also use the `<b>` and `<i>` tags to format text as bold and italic respectively:

Chapter 18 Part 2: Web Applications (HTML)

Text can be `<b>bold</b>`, `<i>italic</i>` or `<b><i>both</i></b>`.

Besides basic formatting, a web page may also contain images and links to other web pages. For example, the following HTML snippet describes a link to `www.example.com` using an `<a>` tag with a `href` attribute:

```
<a href="http://www.example.com">Example</a>
```

The `<a>` tag stands for "anchor" as it attaches some portion of the web page to a fixed URL, just like how a ship's anchor attaches the ship to a fixed location. The `href` attribute stands for "hypertext reference" and is set to which URL the `<a>` tag's content will be linked to.



Note that when linking to an external site, the `href` attribute must use an **absolute URL**. This means that the attribute value must include the scheme component of the URL (i.e., `http://www.example.com`) and cannot be written as `www.example.com`. (This is because `www.example.com` is interpreted as linking to a file named `www.example.com` and not the external site.)

Chapter 18 Part 2: Web Applications (HTML)

If a URL does not include the scheme component, it is considered to be a **relative URL**. This means that the web browser will treat the URL as a path name starting from the web page's initial directory. For instance, the following HTML snippet describes a link to the `question1.html` web page in the `quiz` sub-directory:

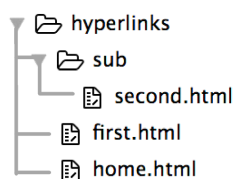
```
<a href="quiz/question1.html">View Question 1</a>
```

Be aware that while Windows uses the backslash (\) to separate directory names in a path, URLs always use the forward slash (/). You should also be aware that `..` is a special directory name that refers to the parent directory (i.e., the directory that is one level up from the current directory).

The following table shows what the resulting URL will be after following a relative URL from a given current URL of the page:

Current URL	Relative URL	Resulting URL
http://www.sg/en	example	http://www.sg/example
http://www.sg/en/	example	http://www.sg/en/example
http://www.sg/en/sg	example/	http://www.sg/en/example/
http://www.sg/en/sg/	example/	http://www.sg/en/sg/example/
http://www.sg/en/sg	../example	http://www.sg/example
http://www.sg/en/sg/	../example	http://www.sg/en/example
http://www.sg/en/sg/	/example	http://www.sg/example

- 10 Create three HTML documents using the following directory structure and contents. Open `home.html` and verify that the two links to `first.html` and `second.html` work as expected.



home.html

```
<!doctype html>
<html>
  <head><title>Home Page</title></head>
  <body>
    <p><a href="first.html">First link</a></p>
    <p><a href="sub/second.html">Second link</a></p>
  </body>
</html>
```

Chapter 18 Part 2: Web Applications (HTML)

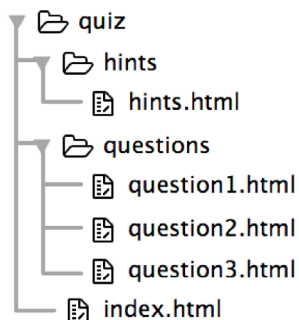
first.html

```
<!doctype html>
<html>
  <head><title>First Page</title></head>
  <body>
    <p>You are on the first page.</p>
  </body>
</html>
```

second.html

```
<!doctype html>
<html>
  <head><title>Second Page</title></head>
  <body>
    <p>You are on the second page.</p>
  </body>
</html>
```

- 11 You are given the following directory structure:



- a. Which HTML snippet would correctly create a link from question1.html to index.html?

A	<code>Home</code>
B	<code>Home</code>
C	<code>Home</code>
D	<code>Home</code>

(C)

Chapter 18 Part 2: Web Applications (HTML)

- b. Which HTML snippet would correctly create a link from question1.html to hints.html?

A	<code>Hints</code>
B	<code>Hints</code>
C	<code>Hints</code>
D	<code>Hints</code>

(C)

- 12 Which of the following links to the MOE home page using the text "MOE"?

A	<code>www.moe.gov.sg</code>
B	<code>http://www.moe.gov.sg</code>
C	<code>MOE</code>
D	<code>MOE</code>

(D)

Besides links, web pages can also include images using an `<img>` tag with the `src` and `alt` attributes:

```

```

The `<img>` tag is short for "image" and its `src` attribute (short for "source") must be set to the URL of an image file. For maximum compatibility, the image file should be in either GIF, JPEG or PNG format. Furthermore, to support users who cannot or do not wish to view the image, the compulsory `alt` attribute provides an "alternative" text description that can be used as a replacement for the image.

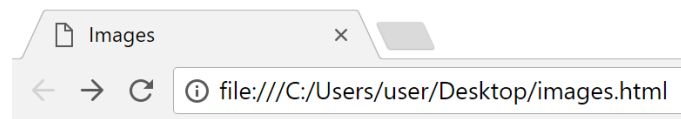
As with the `href` attribute of the `<a>` tag, the `src` attribute of an `<img>` tag can be set to either an absolute or relative URL depending on whether the image is stored on an external site.

- 13 Create a small image of a star in PNG format (using graphics software such as Paint) and save it as `star.png`. Create the following HTML document and save it as `images.html` in the same directory as `star.png`.

Chapter 18 Part 2: Web Applications (HTML)

```
<!doctype html>
<html>
  <head><title>Images</title></head>
  <body>
    <h1>Images can appear...</h1>
    <h2>...on their own!</h2>
    
    <h2>...in the middle of a paragraph!</h2>
    <p>I  HTML!</p>
    <h2>...in a hyperlink!</h2>
    <p>
      Click:
      <a href="http://www.example.com">
        
      </a>
    </p>
  </body>
</html>
```

Open `images.html` and verify that the page appears as follows.



Images can appear...

...on their own!



...in the middle of a paragraph!

I  HTML!

...in a hyperlink!

Click: 

Also check that the last star works as a hyperlink to www.example.com.

You can also delete `star.png` and reopen `images.html` to see what happens when the required image is missing.

Chapter 18 Part 2: Web Applications (HTML)

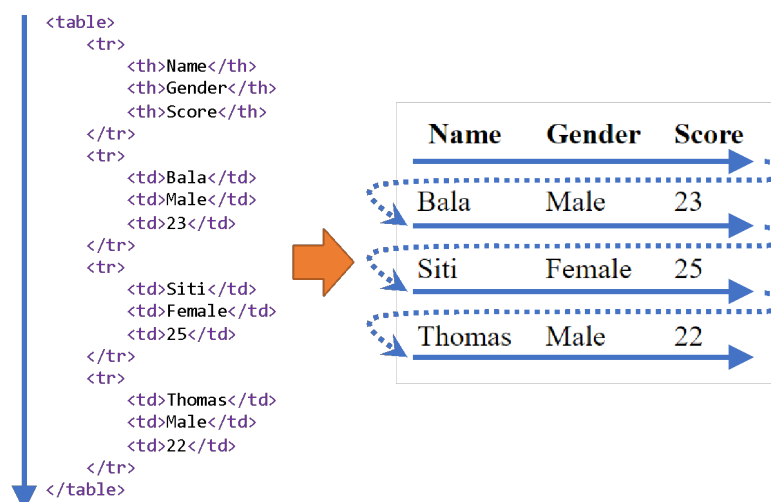
- 14 For each tag in the table below, circle "Normal" if it describes a normal element and circle "Void" if it describes a void element:

Tag	Description	Element Type
<code></code>	Bold (by convention)	Normal / Void
<code><i></code>	Italic (by convention)	Normal / Void
<code><a></code>	Hyperlink	Normal / Void
<code></code>	Image	Normal / Void

Part 7: Table Tags

So far, we have learn learnt how to describe headings, paragraphs, links and images in HTML. These elements follow the document's flow and are arranged top-to-bottom, left-to-right in the same order that they appear in the HTML code:

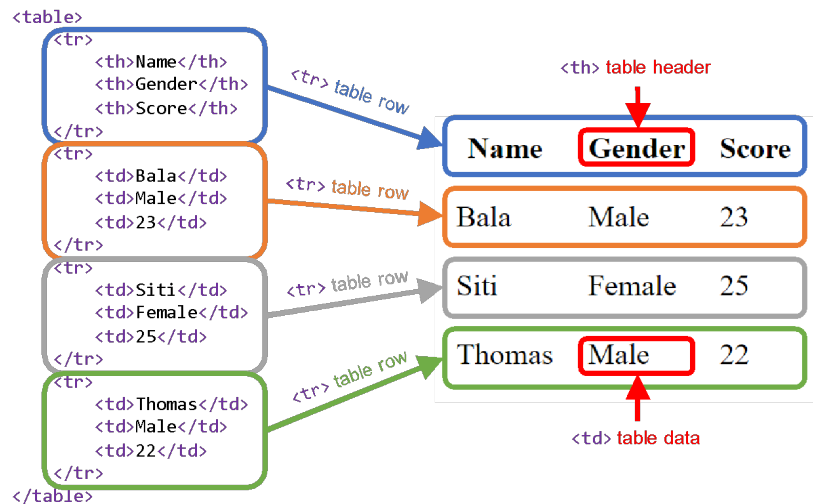
We can use the same top-to-bottom, left-to-right ordering to describe a table using the `<table>`, `<tr>`, `<th>` and `<td>` tags:



`<table>` indicates the start of one table, while `</table>` indicates the end of the table. Within those tags, each row (starting from the top) is represented by a `<tr>`

Chapter 18 Part 2: Web Applications (HTML)

(table row) tag. In turn, each row contains a sequence of <th> (table header) or <td> (table data) tags, with each tag describing a cell in that table row from left to right.



Note that while both <th> and <td> are used to describe individual table cells, <th> should only be used for **header cells** that contain row or column titles.

- 15 Use the <table>, <tr>, <th> and <td> tags to create the following table and save it in risk-assessment.html:

	Low Probability	High Probability
Low Impact	Low Risk	Medium Risk
High Impact	Medium Risk	High Risk

- 16 For each tag in the table below, circle "Normal" if it describes a normal element and circle "Void" if it describes a void element:

Tag	Description	Element Type
<table>	Table	Normal / Void
<tr>	Table row	Normal / Void
<th>	Table header cell	Normal / Void
<td>	Table cell	Normal / Void

Chapter 18 Part 2: Web Applications (HTML)

Do not worry about the lack of table borders at the moment. You will learn about how to add them in the next practical task.

Part 8: Form Tags

An important advantage of web pages over conventional documents is that web pages can collect inputs from the reader using forms such as the following example:

Feedback Form

Name:

Feedback:

Submit

Such forms are created in HTML using the `<form>`, `<input>` and `<textarea>` tags:

```
<form action="http://www.example.com">
  <h1>Feedback Form</h1>
  <p>Name: <input name="username" type="text" value=""></p>
  <p>Feedback:</p>
  <textarea name="feedback">Enter feedback here.</textarea>
  <p><input type="submit"></p>
</form>
```

Each form is contained in a separate `<form>` tag with an `action` attribute set to the URL where the submitted data will be sent to. (In future practical tasks, you will learn how to write a Python program that runs on a web server to process this data.) Inside the `<form>` tag, you can use the `<input>` and `<textarea>` tags in addition to any of the section, text, media and table tags that you have already learnt. Each `<input>` and `<textarea>` tag represents an **input control** and may have a unique `name` attribute to help the server retrieve these inputs.

An `<input>` tag can represent either a single-line text box or submit button, depending on whether the `type` attribute is set to `"text"` or `"submit"`. For a text box, the `value` attribute determines the text box's initial contents. For a submit button, the `value` attribute determines the button's label. Note that for either case, the `<input>` tag does not have an end tag.

Chapter 18 Part 2: Web Applications (HTML)

An `<input>` tag can also set its type attribute to "file". This allows the user to upload files for submission with the form. We shall learn how to handle file uploads in a future practical task.

The `<textarea>` tag, on the other hand, represents a multi-line text box. Unlike the `<input>` tag that does not have an end tag, the `<textarea>` tag does require an end tag and the text inside will be used as the text box's initial content.

- 17 Use the `<html>`, `<head>`, `<title>`, `<body>`, `<form>`, `<input>`, `<textarea>`, `<h1>` and `<p>` tags to create the following form and save it in `search.html`. (You may choose any URL for the action attribute of the form.)

Search Form

Search:

- 18 Use the `<html>`, `<head>`, `<title>`, `<body>`, `<form>`, `<input>`, `<textarea>`, `<h1>`, `<p>` and the table tags to create the following form and save it in `game.html`. (You may choose any URL for the action attribute of the form.)

Tic-Tac-Toe

Enter O in one of the text boxes below.

X	X	O
O		
		X

- 19 For each tag in the table below, circle "Normal" if it describes a normal element and circle "Void" if it describes a void element:

Tag	Description	Element Type
<code><form></code>	User-submittable form	Normal / Void
<code><input></code>	Input control	Normal / Void
<code><textarea></code>	Text input area	Normal / Void

Chapter 18 Part 2: Web Applications (HTML)

Chapter 18 Part 3: Web Applications (CSS)

Instructional Objectives:

By the end of this task, you should be able to:

- Explain how CSS only describes the presentation of a web page and not its structure or contents
- Use the <link> tag to associate a web page to a CSS style sheet
- Identify the rules, selectors, declarations and properties that are present in a given CSS style sheet
- Use an element selector to limit effect of a CSS rule to a particular element type
- Use an id attribute in HTML and an id selector in CSS to limit effect of a CSS rule to a single element
- State that it is an error for a web page to have more than one HTML element with the same id attribute
- Use a class attribute in HTML and a class selector in CSS to limit effect of a CSS rule to elements with the specified class
- Use a descendent selector in CSS to limit the effect of a CSS rule to elements that are nested in a particular order
- Distinguish between elements that have a block appearance by default and elements that have an inline appearance by default
- Identify and distinguish between the margin, border and padding of elements that have a block appearance
- Use the <div> tag to represent structural elements that should have a block appearance by default and are not represented by other HTML tags
- Use the tag to represent structural elements that should have an inline appearance by default and are not represented by other HTML tags

Chapter 18 Part 3: Web Applications (CSS)

- Use the display, background, color, height, width, border, border-bottom, border-left, border-right, border-top, margin, margin-bottom, margin-left, margin-right, margin-top, padding, padding-bottom, padding-left, padding-right, padding-top, font-family, font-size, font-style, font-weight, text-align and text-decoration properties to control the presentation of a web page without producing any syntax errors

Part 1: Why Another Language?

So far, the web pages we have created look rather plain:

The Benefits of CSS

Without CSS, web pages may look rather boring. For instance, by default, all text is black and set in a serif font. Tables are also displayed without borders. Here is a comparison of web pages without CSS and web pages with CSS:

Without CSS	With CSS
Background is white by default	Background color can be customised
Text is black by default	Text color can be customised
Text uses a serif font by default	Text font can be customised
Tables are displayed without borders	Tables can be displayed with borders

Comments

Submit Comment

However, with the help of CSS, we can improve the appearance of such web pages greatly with almost no change to the HTML:

The Benefits of CSS

Without CSS, web pages may look rather boring. For instance, by default, all text is black and set in a serif font. Tables are also displayed without borders. Here is a comparison of web pages without CSS and web pages with CSS:

Without CSS	With CSS
Background is white by default	Background color can be customised
Text is black by default	Text color can be customised
Text uses a serif font by default	Text font can be customised
Tables are displayed without borders	Tables can be displayed with borders

Comments

Submit Comment

You may wonder why we need CSS to control a web page's appearance. The answer lies in a computer science principle called **separation of concerns**, where a program is divided into distinct sections such that each section only deals with one aspect of the final product and has minimal knowledge of the other parts.

Chapter 18 Part 3: Web Applications (CSS)

In general, a web page's structure (how the content is organized) and presentation (how the content is displayed) are largely independent of each other and therefore should be handled separately. In particular, we define the web page's structure using HTML in one file and its presentation using CSS in a separate file. Separating these two components lets us modify the web page's structure without affecting its presentation and vice versa. Thus, as an example, a person can be in charge of the content of the website, while another person can design how the website is presented without worrying over the content of the website. It also lets us specialise the HTML and CSS languages so that each language is more suited for its purpose.

To demonstrate, open Google Chrome and visit www.csszengarden.com. Click on any of the available designs to see how changing CSS style sheets can dramatically affect the appearance of a web page without modifying its HTML. This is possible because the web page's structure and presentation are controlled separately. (You may view the source code by pressing Ctrl-U to check that the HTML is exactly the same for each design.)

Part 2: Anatomy of a CSS Style Sheet

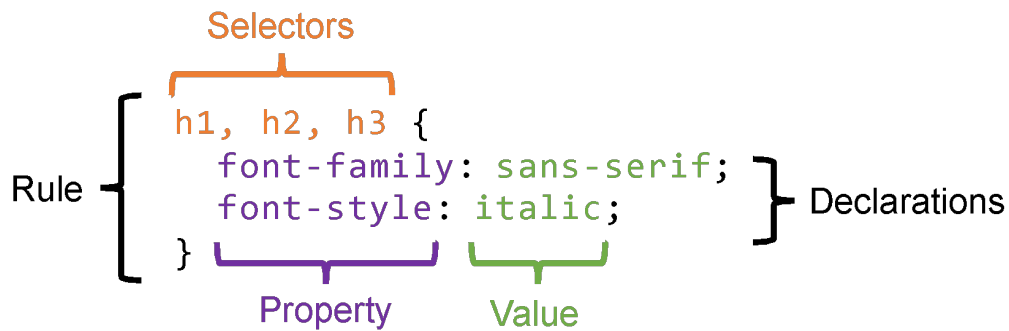
Open Google Chrome, visit www.example.com and press Ctrl-U. Notice that the first part of the HTML document includes some CSS inside a `<style>` tag:



For simplicity, this web page has chosen to include its style sheet inside the HTML file using a `<style>` tag. However, in general we write our style sheet in a separate `.css` file and link it to a HTML file using a `<link>` tag. We will learn how to do this in the next part of this practical task.

Chapter 18 Part 3: Web Applications (CSS)

Focusing on the CSS between the <style> and </style> tags, we see that the style sheet is made of multiple **rules**. Each rule starts with one or more **selectors** separated by commas, followed by curly braces surrounding a number of **declarations**. These terms are illustrated for a single rule as follows:



Each declaration is made of two parts: a **property** name and one or more values separated by spaces. Multiple declarations in a rule are separated by semicolons.

- 1 Circle all selectors and underline all properties in the following style sheet:

```
body {  
  margin: 0;  
  padding: 0;  
}  
  
h1, h2, #danger {  
  border: 1px solid red;  
}
```

- 2 For each code snippet below, place a tick in the corresponding box if the snippet is valid CSS. Note that background and color are both valid CSS property names.

Code Snippet	Valid?
<pre>body (background: white;)</pre>	
<pre>body { background: white color: black }</pre>	

Chapter 18 Part 3: Web Applications (CSS)

<pre>body { background: white; color: black; }</pre>	
<pre>{ color: black; }</pre>	
<pre>body { color: black; }</pre>	

Part 3: Your Own Style Sheet

Now that we know about the basics of CSS, it is time to create our own style sheet. Like HTML documents, CSS style sheets are just text files with a `.css` extension. You may wish to use Notepad++ to edit CSS files as it offers syntax highlighting. Open a text editor, create a new file and save it in your user directory as `example.css`. Then enter the following style sheet:

```
h1 { color: red; font-family: sans-serif; }  
p { color: blue; font-style: italic; }
```

To use a stylesheet, we need an accompanying document. Create a new file and save it as `example.html` in the same directory as `example.css`. Then enter the following HTML document:

```
<!doctype html>  
<html>  
  <head>  
    <title>CSS Example</title>  
    <link rel="stylesheet" href="example.css">  
  </head>  
  <body>  
    <h1>CSS Example</h1>  
    <p>This is an example of using CSS to style a web page.</p>  
  </body>  
</html>
```

Chapter 18 Part 3: Web Applications (CSS)

Notice that we have added a `<link>` tag under the head element. The `rel` attribute (short for "relationship") has a value of "stylesheet", which indicates that the `href` attribute (short for "hyperlink") contains the URL of a CSS style sheet. This is how we normally associate HTML documents with style sheets.

Now, double-click `example.html` in Windows Explorer to open it in a web browser. You should see something similar to the following:

CSS Example

This is an example of using CSS to style a web page.

Congratulations! You have just used CSS to control the presentation of a web page.

- 3 Use a text editor to create a HTML document and CSS style sheet using the following directory structure and contents. Open `index.html` and add a `<link>` tag so that the web page is styled using `colours.css`. Use relative paths so the style sheet is always linked correctly even if the `colourful` directory and its contents are moved to another location.



colours.css

```
body { background: cyan; }

h1 { color: blue; }

p {
    background: yellow;
    color: red;
}
```

index.html

```
<!doctype html>
<html>
  <head>
    <title>Colourful</title>
    <link rel="stylesheet" href="styles/colours.css">
```

Chapter 18 Part 3: Web Applications (CSS)

```
</head>
<body>
  <h1>Welcome</h1>
  <p>This is a colourful document!</p>
</body>
</html>
```

Part 4: Types of Selectors

The elements which are affected by each CSS rule are determined by the selectors at the start of that rule. Each selector picks out (i.e., selects) a set of elements from the HTML document to be affected by the rule's declarations.

We shall organize selectors into four groups: element selectors, id selectors, class selectors and descendent selectors.

Element Selectors

An element selector picks out *all* elements of a particular type from the HTML document. To use an element selector, we simply enter the name of a tag or element type (e.g., body, h1, table, etc.). For instance, the second rule in `example.css` uses an element selector for p elements:

```
p { color: blue; font-style: italic; }
```

Based on this rule, all p elements on the web page will appear as blue italic text.

Sometimes, however, we do not want to select *all* the elements of a particular type. In such cases, we can fine-tune our selection by making use of two special attributes that are valid for all HTML tags: `id` and `class`.

id Selectors

An id selector picks out the unique element that has a particular value for its `id` attribute. To use an id selector, we enter a hex symbol (#) followed immediately by the desired element's id attribute value. Since id attributes on a web page cannot be repeated, an id selector will always pick out exactly *one* element if it exists.

For instance, suppose we have the following HTML and CSS files in the same folder:

```
id-example.html
```

Chapter 18 Part 3: Web Applications (CSS)

```
<!doctype html>
<html>
  <head>
    <title>ID Selectors Example</title>
    <link rel="stylesheet" href="id-example.css">
  </head>
  <body>
    <p>This is a normal paragraph.</p>
    <p id="special">This paragraph is special.</p>
    <p>This is a normal paragraph.</p>
  </body>
</html>
```

id-example.css

```
#special { color: red; }
```

If we open id-example.html, we see that only the second p element with an id of special is formatted as red.

This is a normal paragraph.

This paragraph is special.

This is a normal paragraph.

class Selectors

A class selector picks out all elements that are associated with a particular class. To use a class selector, we enter a period (.) followed immediately by class name we wish to reference.

For instance, suppose we have the following HTML and CSS files in the same folder:

class-example.html

```
<!doctype html>
<html>
  <head>
    <title>Class Selectors Example</title>
    <link rel="stylesheet" href="class-example.css">
  </head>
  <body>
```

Chapter 18 Part 3: Web Applications (CSS)

```
<p>This is the main paragraph.</p>
<p class="info major">This is some major information.</p>
<p class="info minor">This is some minor information.</p>
</body>
</html>
```

[class-example.css](#)

```
.info { color: silver; }
```

If we open `class-example.html`, we see that only the second and third `p` elements with a `class` of `info` are formatted as silver. These `p` elements also have additional classes (i.e., "major" and "minor" respectively), but these have no effect as they do not match any of the selectors used in the style sheet.

This is the main paragraph.

This is some major information.

This is some minor information.

To summarise the use of `id` and `class` attributes:

- The `id` attribute is used to *uniquely* identify an element in a HTML document. If set, its value must be unique within the document and cannot contain any whitespace. A web page cannot have more than one element with the same `id` attribute.
- The `class` attribute, on the other hand, is used to associate an element with one or more classes (i.e., categories). If set, its value must be a space-separated list of class names. Unlike `id` attributes, multiple elements on a web page can have the same value for their `class` attributes.

Descendent Selectors

Sometimes, element selectors, `id` selectors and class selectors are not sufficient and it is necessary to select an element only if the element has a parent element that matches another selector. This can be achieved using the descendent selector. To use a descendent selector, separate any two selectors using a space: the corresponding rule will only be applied for elements that match the selector on the right *and* have a parent element that matches the selector on the left.

Chapter 18 Part 3: Web Applications (CSS)

For instance, suppose we have the following HTML and CSS files in the same folder:

[descendent-example.html](#)

```
<!doctype html>
<html>
  <head>
    <title>Descendent Selectors Example</title>
    <link rel="stylesheet" href="descendent-example.css">
  </head>
  <body>
    <h1>Heading with <i>Italics</i></h1>
    <p>This paragraph has <i>italics</i>.</p>
    <i>Bare italics</i>
  </body>
</html>
```

[descendent-example.css](#)

```
p i { color: red; }
```

If we open `descendent-example.html`, we see that only the `p` element has its italics portion formatted as red. The other `i` elements remain unformatted as they do not match the specific requirements of the selector, which requires the `i` element to be a descendent of a `p` element.

Heading with *Italics*

This paragraph has *italics*.

Bare italics

To further illustrate how the different selector types work, consider the following HTML document. In this example, we introduce a new `<div>` tag that is used as a generic element for grouping other elements. We will learn more about the `<div>` tag later in this practical task.

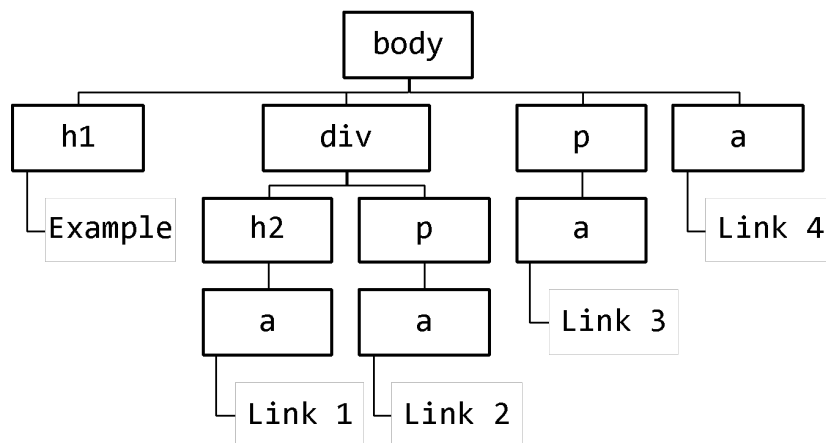
[multiple-example.html](#)

```
<!doctype html>
```

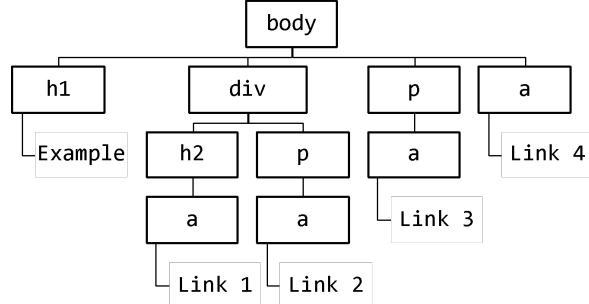

Chapter 18 Part 3: Web Applications (CSS)

```
<html>
  <head><title>Example</title></head>
  <body>
    <h1>Example</h1>
    <div>
      <h2><a id="link1" href="link1.html">
        Link 1
      </a></h2>
      <p><a id="link2" href="link2.html">
        Link 2
      </a></p>
    </div>
    <p><a class="simple" href="link3.html">Link 3</a></p>
    <a class="simple" href="link4.html">Link 4</a>
  </body>
</html>
```

This HTML describes the following tree structure of elements:



Based on this HTML document, the following table shows which elements from the tree are matched for different CSS selectors:

CSS	Matched Elements	Result
<i>No style sheet</i>		Example Link 1 Link 2 Link 3 Link 4

Chapter 18 Part 3: Web Applications (CSS)

<pre>a { background: red; }</pre>		Example Link 1 Link 2 Link 3 Link 4
<pre>#link2 { background: red; }</pre>		Example Link 1 Link 2 Link 3 Link 4
<pre>.simple { background: red; }</pre>		Example Link 1 Link 2 Link 3 Link 4
<pre>div a { background: red; }</pre>		Example Link 1 Link 2 Link 3 Link 4

Note that an element may be selected for more than one rule in the style sheet. If these rules have conflicting declarations, the rule with a selector that is most specific to the element has priority. The CSS language defines an algorithm for calculating the **specificity** of a selector. However, we can avoid worrying about specificity by crafting our style sheets to avoid rule conflicts in the first place.

Now that we know how CSS determines which elements are affected by each rule, let us focus on the various properties that can be set by rule declarations. We shall organize the properties into three groups: common properties, box model properties and typography properties.

Chapter 18 Part 3: Web Applications (CSS)

- 4 Use a text editor to create the following HTML document. Without changing the HTML, add a CSS file such that only the text "RED" is coloured in red:

```
<!doctype html>
<html>
  <head>
    <title>Selectors Quiz</title>
    <link rel="stylesheet" href="selectors.css">
  </head>
  <body>
    BLACK
    <h1>RED</h1>
    <p>BLACK <b>BLACK</b> <i>BLACK</i></p>
    <p id="test1">BLACK <b>RED</b></p>
    <p class="test2">BLACK <i>RED</i></p>
  </body>
</html>
```

Part 5: Common Properties

A number of properties can be used with *any* element. For instance, in many of the CSS examples so far, we have been using the background and color properties to set the background and text color of elements respectively.

The following are some of the color names that can be used with the background and color properties:

red	orange	yellow	green	blue	purple
black	gray	silver	white	transparent	
				(no color)	

If a desired color does not match any of the above names, we can also specify a color in terms of its red, green and blue components. Each component is expressed as an integer between 0 to 255 (inclusive) and the color is written as `rgb(R, G, B)`, where R is the red component integer, G is the green component integer and B is the blue component integer. For example, the following CSS sets the page background to a shade of pale yellow:

```
body { background: rgb(255, 255, 128); }
```

Chapter 18 Part 3: Web Applications (CSS)

The same color can be expressed as three hexadecimal numbers of 2 digits each (including a leading zero if needed). The color can thus be written as #RRGGBB, where RR is the red component in hexadecimal, GG is the green component in hexadecimal and BB is the blue component in hexadecimal. For example, the shade of pale yellow shown previously can also be written as:

```
body { background: #ffff80; }
```

Notice that ff is the integer 255 in hexadecimal, while 80 is the integer 128 in hexadecimal. Also note that hexadecimal digits are not case sensitive.

For convenience, if each of the three hexadecimal numbers is made of repeated digits (e.g. 00, 11, 22, ..., FF), then the color can be shortened to #RGB, where R is the repeated hexadecimal digit for red, G is the repeated hexadecimal digit for green and B is the repeated hexadecimal digit for blue. For example, while #FFFF80 cannot be shortened, the color #00FFCC can be shortened to #0FC.

Besides colors, we can also use a repeating image as the background by using url(PATH), where PATH is the path and filename of the background image. For instance, the following CSS uses pattern.png (which is located in the same folder with the style sheet) as a repeating background image for the page.

```
body { background: url(pattern.png); }
```

The following HTML and pattern.png image demonstrates this:

HTML	pattern.png	Result
<pre><body> <p>Hello, World</p> </body></pre>		

Finally, all elements have a display property that can be used to hide the element by setting its value to none. The display property can also be set to either block or inline. These settings will be explained in the next part of this task.

Part 6: Box Model Properties

Block and Inline Appearance

Chapter 18 Part 3: Web Applications (CSS)

You may have noticed that some HTML tags such as `<h1>` and `<p>` always start on a new line and force the following element to also start on a new line. On the other hand, tags such as `<b>` and `<i>` do not:

HTML	Result
<pre><h1>h1</h1> <p>p</p> b <i>i</i></pre>	h1 p b <i>i</i>

This is because the `<h1>` and `<p>` tags have a block appearance by default while `<b>` and `<i>` have an inline appearance by default. However, we can use the `display` property to override these defaults. For instance, the following CSS changes all the elements to use a block appearance instead:

CSS	Result
<pre>h1, p, b, i { display: block; }</pre>	h1 p b <i>i</i>

Similarly, the following CSS changes all the elements to use an inline appearance:

CSS	Result
<pre>h1, p, b, i { display: inline; }</pre>	h1 p b <i>i</i>

- 5 Experiment with the HTML tags you have learnt previously. For each tag in the table below, circle "Block" if it creates a block element by default and circle "Inline" if it creates an inline element by default:

Tag	Default Appearance
<code><h1></code>	Block / Inline

Tag	Default Appearance
<code><a></code>	Block / Inline

Chapter 18 Part 3: Web Applications (CSS)

<h2>	Block / Inline		Block / Inline
<h3>	Block / Inline	<table>	Block / Inline
<p>	Block / Inline	<form>	Block / Inline
	Block / Inline	<input>	Block / Inline
<i>	Block / Inline	<textarea>	Block / Inline

<div> and Tags

Sometimes, we may need a generic block or inline element to represent some structural feature of a document that the built-in HTML tags do not provide. The <div> and tags can be used to create block and inline elements respectively for this purpose. These tags are generally used in combination with the id and class attributes to help clarify their purpose.

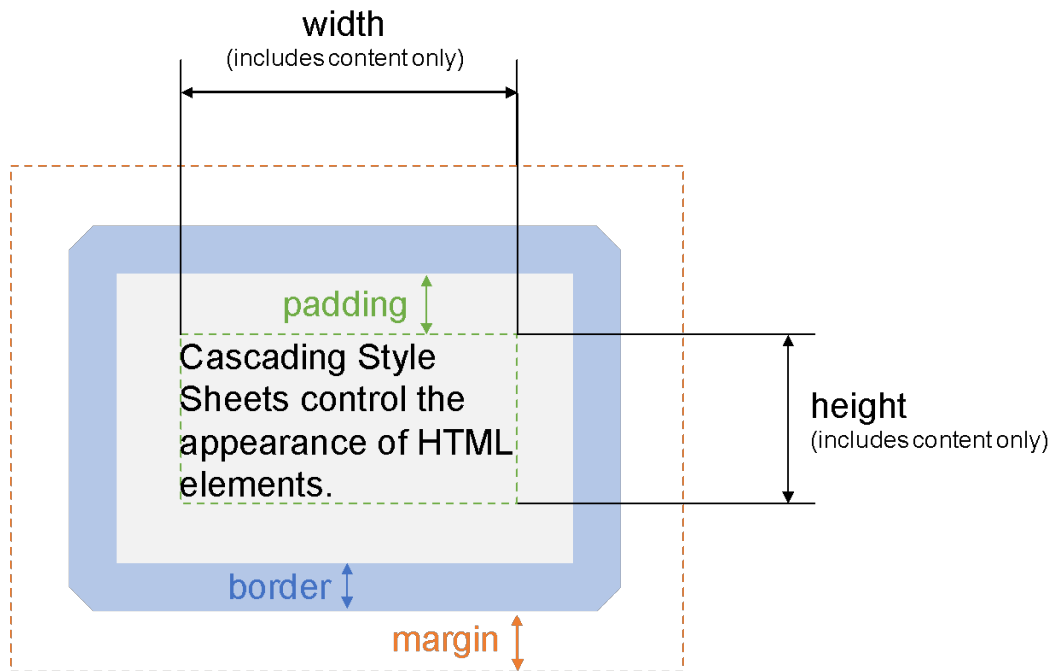
For instance, the following HTML document on Python syntax uses <div> and tags so that Python's rules and keywords are marked out and styled:

HTML	CSS	Result
<pre><h1>Functions</h1> <div class="rule"> Python functions are defined using the def keyword. </div></pre>	<pre>.rule { background: silver; } .keyword { color: red; }</pre>	Functions Python functions are defined using the def keyword.

Margin, Border, Padding, Width and Height

Besides background and text colour, block elements have a number of additional properties that can be specified. In particular, the margin, border, padding, width and height properties determine how the box surrounding the element is sized and displayed. This **box model** is illustrated below:

Chapter 18 Part 3: Web Applications (CSS)



The various box model properties are summarized in the following table:

Property Name	Description
border	Specifies the thickness of the optionally-coloured border around the element
margin	Specifies the thickness of the transparent space surrounding the border
padding	Specifies the thickness of the space between the content and the border that is filled with the element's background colour or pattern
width	Specifies the element content's width, regardless of the surrounding margin, border and padding
height	Specifies the element content's height, regardless of the surrounding margin, border and padding

When setting a box's width and height or the thickness of its margin, border and padding, we must specify a unit of measurement. One common unit of measurement is pixels, represented by the px suffix. For instance, to set the padding of p elements to 10 pixels, we would use the following CSS:

CSS	Result
<pre>p { background: silver; padding: 10px; }</pre>	This is an example of a paragraph with additional padding written using the <p> tag.

Chapter 18 Part 3: Web Applications (CSS)

To specify that a border should be drawn with a solid colour, we use the value of a thickness, followed by a space, the word "solid", another space and finally the colour we wish to use for the border. For instance, the following CSS gives p elements a 5 pixel thick red border:

CSS	Result
<pre>p { border: 5px solid red; }</pre>	This is an example of a paragraph with a 5 pixel thick red border written using the <p> tag.

By default, the margin, border and padding properties control the appearance for all four sides of the element's box. However, we can append -bottom, -left, -top or -right to any of these properties so that we control the appearance for only one side of the box. For instance, the following table shows how the border-bottom, border-left, border-top and border-right properties can be used to draw a line on only one side of a p element:

CSS	Result
<pre>p { border-bottom: 1px solid gray; }</pre>	This is an example of a paragraph.
<pre>p { border-left: 1px solid gray; }</pre>	This is an example of a paragraph.
<pre>p { border-top: 1px solid gray; }</pre>	_____ This is an example of a paragraph.
<pre>p { border-right: 1px solid gray; }</pre>	This is an example of a paragraph.

Finally, if a box's width is less than the document's width, we can center the box horizontally by setting its margin-left and margin-right properties to auto:

CSS	Result
<pre>p { background: silver; margin-left: auto; margin-right: auto; width: 100px; }</pre>	This is an example of a paragraph that is forced to be only 100 pixels wide and centered.

Chapter 18 Part 3: Web Applications (CSS)

- 6 For each HTML document below, create a CSS file so that the completed web page appears as closely to the required appearance as possible.

HTML	Required Appearance
<pre><h1>Introduction</h1> <p>This is an example of using borders to make headings prettier.</p> <h1>Explanation</h1> <p>Each heading has a line below it that extends across the entire screen.</p></pre>	Introduction This is an example of using borders to make headings prettier. Explanation Each heading has a line below it that extends across the entire screen.
<pre><p>Printing documents can be a very tricky business. There are many things to look out for.</p> <p class="tip">Tip: Make sure the printer has paper before printing!</p> <p>However, if you get the process right, you will be rewarded with a marvelous- looking printout.</p></pre>	Printing documents can be a very tricky business. There are many things to look out for. Tip: Make sure the printer has paper before printing! However, if you get the process right, you will be rewarded with a marvelous-looking printout.
<pre><h1>Heading</h1> <p>This is an example of a paragraph that has both margin and padding set.</p></pre>	Heading This is an example of a paragraph that has both margin and padding set.

Part 7: Typography Properties

Besides controlling the size and spacing of box elements, you can also specify how text in each element is displayed using the font-family, font-size, font-style, font-weight, text-align and text-decoration properties.

Chapter 18 Part 3: Web Applications (CSS)

The `font-family` property specifies which typeface is used to display the text. Its value is most often a comma-separated list of font names, ending with either `serif` or `sans-serif`. Note that if a font name has spaces, it must be enclosed in quotes (e.g., "Times New Roman"). The browser will use the first font in the list that is installed. However, if none of the named fonts are installed, it will fall back to using a generic `serif` or `sans-serif` font instead.

A serif font such as "Times New Roman" has lines extending from the ends of each letter stroke. Such fonts are traditionally used for long pieces of printed text. A sans-serif font such as "Arial", however, does not have these additional lines. The following examples illustrate how to set a serif or sans-serif font using CSS:

CSS	Result
<code>p { font-family: serif; }</code>	This is an example of a paragraph.
<code>p { font-family: sans-serif; }</code>	This is an example of a paragraph.

The `font-size` property can be used to specify text size in pixels:

CSS	Result
<code>p { font-size: 24px; }</code>	This is an example of a paragraph.

The `font-style` property specifies whether an italic font is used. The most common values for this property are `normal` and `italic`. Similarly, the `font-weight` property specifies whether a bold font is used. The most common values for this property are `normal` and `bold`:

CSS	Result
<code>p { font-style: italic; }</code>	<i>This is an example of a paragraph.</i>
<code>p { font-weight: bold; }</code>	This is an example of a paragraph.
<code>p { font-style: italic; font-weight: bold; }</code>	<i>This is an example of a paragraph.</i>

Chapter 18 Part 3: Web Applications (CSS)

The text-align property specifies how the text is aligned in its box. The most common values for this property are left, center, right and justify:

CSS	Result
<pre>p { text-align: left; }</pre>	This is an example of a paragraph with enough content to see how text-align works.
<pre>p { text-align: center; }</pre>	This is an example of a paragraph with enough content to see how text-align works.
<pre>p { text-align: right; }</pre>	This is an example of a paragraph with enough content to see how text-align works.
<pre>p { text-align: justify; }</pre>	This is an example of a paragraph with enough content to see how text-align works.

Finally, the text-decoration property specifies whether additional elements of the font are displayed. The most common values of this property are none, underline and line-through. The line-through value causes text to appear cancelled or struck through using a horizontal line:

CSS	Result
<pre>p { text-decoration: underline; }</pre>	<u>This is an example of a paragraph.</u>
<pre>p { text-decoration: line-through; }</pre>	~~This is an example of a paragraph.~~

- 7 For each HTML document below, create a CSS file so that the completed web page appears as closely to the required appearance as possible. However, there is no need to match the required appearance exactly, so if a font size is needed, use an approximate value.

HTML	Required Appearance
<pre><h1>Reminder</h1> <p>This is an important reminder to try new & different ideas.</p></pre>	<i>Reminder</i> This is an <u>important</u> reminder to try new & different ideas.

Chapter 18 Part 3: Web Applications (CSS)

<pre><h1>Heading 1</h1> <h2>Heading 2</h2> <p>Paragraph</p></pre>	Heading 1 <i>Heading 2</i> Paragraph
<pre><p>This is a link to the root of your web site.</p></pre>	<u>This is a link to the root of your web site.</u>

Chapter 18 Part 4: Web Applications (Flask)

Instructional Objectives:

By the end of this task, you should be able to:

- Use the `flask.Flask()` constructor to create a Flask application
 - Use the `flask.Flask.route()` decorator to associate functions with either a static path or a path containing one or more variable sections
 - Use `flask.Flask.run()` to start a Flask application
 - Use `flask.render_template()` to simplify sending of long HTML responses
 - Use `flask.redirect()` to send a HTTP 302 (Found) status code and have the browser request a different address
 - Use the `flask.request.form` dictionary to retrieve decoded form data
 - Use the Jinja2 template language and `flask.url_for()` to dynamically generate links and URLs in HTML responses
-

Part 1: Dynamic Web Pages

Although we can use the web to simply request for and retrieve static HTML and CSS documents, the web becomes much more useful if the web server can customise its response to each HTTP request. For example, the following program uses what we have learnt so far about sockets, HTTP, HTML and CSS to generate a web page with random colours and text every time it is requested.

Program 1: `p01_server_without_flask.py`

```
1 import random
2 import socket
3
4 # HTTP uses \r\n to end lines instead of \n.
5 EOL = b'\r\n'
6
7 # List of possible colours.
8 COLOURS = [
9     'red', 'orange', 'blue', 'purple',
10    '#C0C000', # dark yellow
```

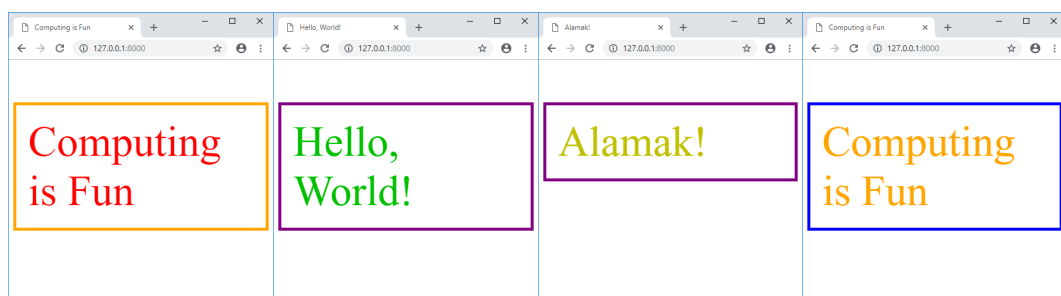
Chapter 18 Part 4: Web Applications (Flask)

```
11     '#00C000'    # dark green
12 ]
13
14 # List of possible messages.
15 MESSAGES = ['Hello, World!', 'Computing is Fun', 'Alamak!']
16
17 # Start listening for web browser requests on port 8000.
18 listen_socket = socket.socket()
19 listen_socket.bind(('127.0.0.1', 8000))
20 listen_socket.listen()
21
22
23 def handle_request(new_socket):
24     # Keep loading request until end of first line (\r\n).
25     request = b''
26     while EOL not in request:
27         received = new_socket.recv(1024)
28         if received == b'':
29             new_socket.close()
30             return
31         request += received
32
33     # Extract first line and split it into three.
34     # Requested path is always the second part.
35     index = request.index(EOL)
36     first_line = request[:index].decode()
37     path = first_line.split()[1]
38
39     # Start response with standard HTTP status line.
40     response = b'HTTP/1.1 200 OK' + EOL
41
42     # Generate CSS document for response if path is '/css'.
43     # Otherwise, generate HTML document for response.
44     if path == '/css':
45         # Generate CSS document.
46         border_colour = random.choice(COLOURS)
47         text_colour = random.choice(COLOURS)
48         body = 'p {{ border: 5px solid {0}; color: {1};'
49         body += 'font-size: 72px; padding: 20px; }}'
50         body = body.format(border_colour, text_colour)
51         body = body.encode()
52
53         # Append HTTP header fields to response.
54         response += b'Content-Type: text/css' + EOL
55         response += b'Content-Length: '
56         response += str(len(body)).encode() + EOL
57     else:
58         # Generate HTML document.
59         msg = random.choice(MESSAGES)
```

Chapter 18 Part 4: Web Applications (Flask)

```
60     body = '<!DOCTYPE html>\n<html>'
61     body += '<head><title>{0}</title>'
62     body += '<link rel="stylesheet" href="/css"></head>'
63     body += '<body><p>{0}</p></body></html>'
64     body = body.format(msg).encode()
65
66     # Append HTTP header fields to response.
67     response += b'Content-Type: text/html' + EOL
68     response += b'Content-Length: '
69     response += str(len(body)).encode() + EOL
70
71     # Append empty line to response.
72     response += EOL
73
74     # Append document as message body.
75     response += body
76
77     # Send the response and close the socket.
78     new_socket.sendall(response)
79     new_socket.close()
80
81
82 while True:
83     new_socket, addr = listen_socket.accept()
84
85     handle_request(new_socket)
```

Run the program in the background, then view <http://127.0.0.1:8000/> in a web browser. Reload the browser a few times to see the web page's text and colours change each time:



How does the program work?

1. Lines 17 to 20 start the server by listening on port 8000.
2. When a web browser visits <http://127.0.0.1:8000/>, it sends a HTTP request via the new socket that is returned by `accept()` on line 83. The new socket is then passed to the `handle_request()` function on line 23.

Chapter 18 Part 4: Web Applications (Flask)

3. Lines 24 to 37 read the HTTP request until a line ending is detected. However, if `recv()` ever returns an empty bytes object, the connection has been closed so the program stops processing the request immediately. Otherwise, it extracts the path component from the first line of the request.
4. Line 40 starts a new HTTP response with only a status line.
5. If path matches `'/css'` exactly, lines 45 to 51 generate a CSS document using random colours and store it in `body`. Lines 53 to 56 then generate the required HTTP header fields and append them to the response.
6. Otherwise, lines 58 to 64 generate a HTML document containing a random phrase and store the HTML document in `body`. Lines 66 to 69 then generate the required HTTP header fields and append them to the response.
7. Regardless of whether a CSS or HTML document was generated, lines 71 to 78 end the header fields with an empty line, append the generated document to the response and send the completed response back to the web browser.
8. Finally, line 79 closes the socket so the server can handle the next HTTP request and so on until Ctrl-C is pressed.

Writing dynamic web pages in this way can be complex and tedious. In particular, `p01_server_without_flask.py` requires the programmer to parse or construct HTTP request/status lines and header fields to follow the HTTP standard. It also requires the programmer to assemble long pieces of HTML and CSS code from Python strings.

Instead of having the programmer spend time on these tedious and error-prone tasks, it is usually better to use a **framework** instead. A framework is typically a module or library with ready-made generic solutions that a programmer can selectively override to customise certain behaviours. This lets the programmer get more done in less time as the framework already provides a working solution that the programmer only needs to configure or customise for his or her needs.

-
- 1 The following program runs a web server at `http://127.0.0.1:5000/` that simply repeats the first line of each HTTP request back to the browser.

```
import socket

EOL = b'\r\n'

listen_socket = socket.socket()
listen_socket.bind(('127.0.0.1', 5000))
listen_socket.listen()

def handle_request(new_socket):
    request = b''
```


Chapter 18 Part 4: Web Applications (Flask)

```
while EOL not in request:
    received = new_socket.recv(1024)
    if received == b'':
        return
    request += received
index = request.index(EOL)
first_line = request[:index]

response = b'HTTP/1.1 200 OK' + EOL
response += b'Content-Type: text/plain' + EOL
response += b'Content-Length: '
response += str(len(first_line)).encode() + EOL
response += EOL
response += first_line

new_socket.sendall(response)
new_socket.close()

while True:
    new_socket, addr = listen_socket.accept()

    handle_request(new_socket)
```

Run the program and visit each of the URLs below to tabulate the output that is displayed in the browser. An example has been completed for you.

No.	URL	Output
e.g.	http://127.0.0.1:5000/hello	GET /hello HTTP/1.1
1	http://127.0.0.1:5000/	GET / HTTP/1.1
2	http://127.0.0.1:5000/100/200	GET /100/200 HTTP/1.1
3	http://127.0.0.1:5000/200/300/	GET /200/300/ HTTP/1.1
4	http://127.0.0.1:5000//300//400//	GET //300//400// HTTP/1.1
5	http://127.0.0.1:5000/?key=value	GET /?key=value HTTP/1.1
6	http://127.0.0.1:5000/#fragment	GET / HTTP/1.1

Chapter 18 Part 4: Web Applications (Flask)

7	<code>http://127.0.0.1:5000/?key#hash</code>	<code>GET /?key HTTP/1.1</code>

Part 2: The Flask Framework

To create dynamic web sites, we can use a web application framework named **Flask**. Unlike the other libraries and modules that we have used so far, Flask is not built into Python and typically needs to be installed separately. However, it should already be installed for you in the computers at your school.

To check if Flask is installed, run the following line of Python (bold text only):

```
>>> import flask
```

If you get an error on your home computer, you may need to install Flask by starting Command Prompt or PowerShell as an administrator and running (bold text only):

```
PS C:\Users\user> pip3 install flask
```

To run the above command successfully, you must have Internet access and Python must be included in your **PATH**, which is the list of directories that Command Prompt or PowerShell automatically searches through when looking for a command. Search the Internet or ask your teacher for help if you are unable to install Flask on your own.

Without any customisations, Flask already provides a basic web server that correctly implements HTTP and its many requirements. To create and run this basic web server, we need to create a `flask.Flask` object with the module's `__name__` as an argument and call the object's `run()` method, like in the following program:

Program 2: p02_minimal.py

```
1 import flask
2
3 app = flask.Flask(__name__)
4
5 if __name__ == '__main__':
6     app.run()
```

When this program is run, you should see some start-up messages that indicate the server can be accessed at `http://127.0.0.1:5000/`. However, as the default web server is not configured to recognise any paths yet, you will receive a 404 (Not Found) error when you visit that URL using a web browser. Nevertheless, you should notice that Flask already provides a complete web server that correctly implements HTTP without additional work from the programmer.

Chapter 18 Part 4: Web Applications (Flask)

Note that 5000 is just the default port number used by Flask. To use another port number, call the `run()` method with a different port argument. For example, to use port 12345 instead, we would replace line 6 with:

```
app.run(port=12345)
```

Technical Note

The latest version of Flask (1.0) and IDLE have a known incompatibility that causes an `io.UnsupportedOperation: fileno` error to occur when calling `app.run()` on Windows. To work around this issue, use Flask 0.12 instead. To replace your current version of Flask, start Command Prompt or PowerShell as an administrator and run (bold text only):

```
PS C:\Users\user> pip3 install flask==0.12
```

An appropriate version of Flask should already be installed on your school's computers.

To stop the Flask server, press Ctrl-C in the IDLE's shell window or restart the shell by running Ctrl-F6.

Part 3: HTTP Requests and Routing

The first and most important way to customise the web server that Flask provides is to configure which paths are recognised. Recall that each HTTP request starts with a request line specifying a **method** and a **path** as well as the version of HTTP being used. Whether a HTTP request succeeds typically depends on whether the path refers to a web document that is recognised by the server and whether the method used is allowed for that web document.

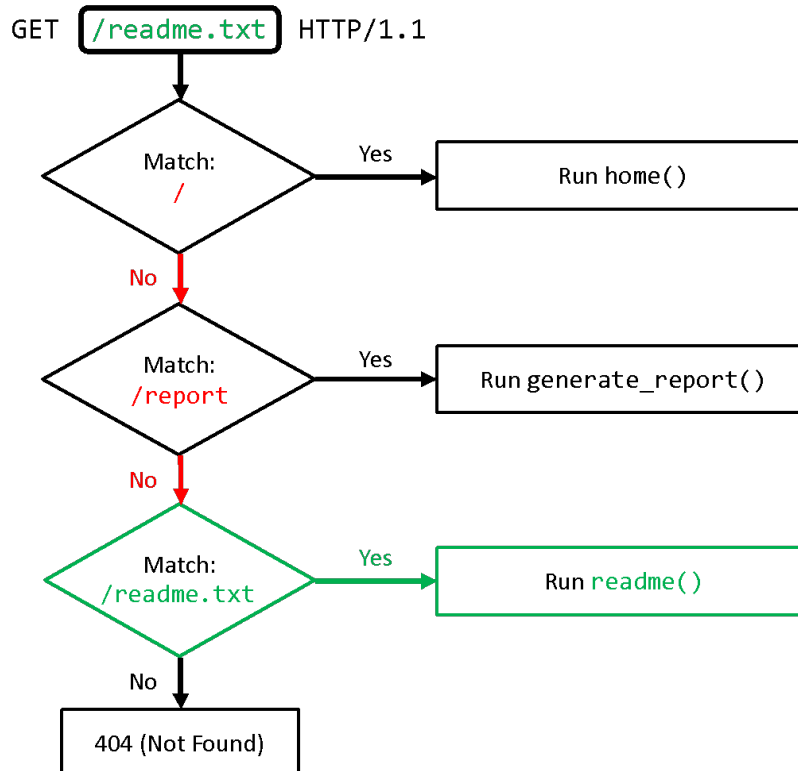
For example, suppose a HTTP server is running on 127.0.0.1 and listening on port 5000. When we visit `http://127.0.0.1:5000/readme.txt`, the browser sends a HTTP request with a first line that contains the path `/readme.txt`, like so:

```
GET /readme.txt HTTP/1.1
```

Note that even though HTTP paths such as `/readme.txt` look similar to file paths that we may use to open and save files in Windows or macOS, to a web server they are just strings and do NOT need to refer to real files or folders that are stored on the computer. For instance, in this example there does not need to be a real file named `readme.txt` for the web server to provide a valid response. Another example is how some web servers dynamically generate an HTML error page when we visit a URL that the server does not recognise.

Chapter 18 Part 4: Web Applications (Flask)

To handle HTTP requests, we can map specific paths to Python functions. For instance the path `/` can be mapped to a function named `home()` and `/readme.txt` can be mapped to a function named `readme()`. When a HTTP request is received, Flask examines the received path and looks for a mapping. If a mapping is found, Flask runs the associated Python function to generate a response. Otherwise, no mapping is found and a 404 (Not Found) status code is produced instead.



This process is called **routing** and it is similar to how phone operators forward calls to different departments based on each caller's needs. Similarly, each HTTP request is routed to a Python function for processing based on the requested path and method used. Each mapping of a path to a Python function is called a **route**.

To declare a route and associate a path to a Python function, we use a feature of Python called **decorators**. In Python, decorators let us alter the behaviour of a function without modifying its source code. We do this by adding **decorations** immediately before the function's definition. Each decoration starts with an at-sign or "pie" symbol `@` followed by a decorator. For Flask, we typically use decorators that are generated using the `route` method of the main Flask object named `app`, as demonstrated on lines 5, 9 and 13 of the following example:

Program 3: p03_simple_routes.py

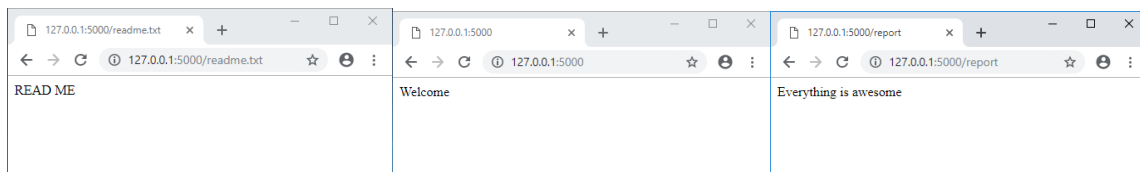
```
1 import flask
2
3 app = flask.Flask(__name__)
4
5 @app.route('/')

```

Chapter 18 Part 4: Web Applications (Flask)

```
6 def home():
7     return 'Welcome'
8
9 @app.route('/report')
10 def generate_report():
11     return 'Everything is awesome'
12
13 @app.route('/readme.txt')
14 def readme():
15     return 'READ ME'
16
17 if __name__ == '__main__':
18     app.run()
```

Run `03_simple_route.py` and visit `http://127.0.0.1:5000/readme.txt` to see the message "READ ME" returned by the `readme()` function. This works because the path `/readme.txt` is mapped to `readme()` by the decoration on line 13. Similarly, if we visit `http://127.0.0.1:5000/`, we see the message "Welcome" and if we visit `http://127.0.0.1:5000/report`, we see the message "Everything is awesome". However, visiting any other URL that starts with `http://127.0.0.1:5000/` (For example `http://127.0.0.1:5000/nothing`) results in a 404 (Not Found) error as no other paths have been mapped.



The next example uses multiple decorators to declare a variety of routes:

Program 4: `p04_complex_routes.py`

```
1 import flask
2
3 app = flask.Flask(__name__)
4
5 @app.route('/')
6 def index():
7     return 'Routed to index()'
8
9 @app.route('/css')
10 def css():
11     return 'Routed to css()'
12
13 @app.route('/no_slash')
14 def no_slash():
15     return 'Routed to no_slash()'
```

Chapter 18 Part 4: Web Applications (Flask)

```
16
17 @app.route('/optional_slash/')
18 def optional_slash():
19     return 'Routed to optional_slash()'
20
21 @app.route('/one/')
22 @app.route('/one/two/')
23 @app.route('/three/two/one')
24 def multiple():
25     return 'Routed to multiple()'
26
27 @app.route('/string/<s>/')
28 def string_variable(s):
29     return 'Routed to string_variable(), s = {}'.format(s)
30
31 @app.route('/integer/<int:i>/')
32 def integer_variable(i):
33     return 'Routed to integer_variable(), i = {}'.format(i)
34
35 @app.route('/post_only/', methods=['POST'])
36 def post_only():
37     return 'Routed to post_only()'
38
39 if __name__ == '__main__':
40     app.run()
```

Fixed Routes

Run `04_complex_routes.py` and visit `http://127.0.0.1:5000/` with a web browser. You should see a simple page with the message "Routed to index()" indicating that your request was handled by the `index()` function defined on lines 6 to 7. This is because the path component of `http://127.0.0.1:5000/` is just a slash character `/`, which matches the route specified on line 5 for the `index()` function.

Now visit `http://127.0.0.1:5000/css` and you should see the message "Routed to css()" instead. In this case, the path component of `http://127.0.0.1:5000/css` is just `/css`, which matches the route specified on line 9. Hence, the request is handed over to the corresponding `css()` function defined on lines 10 and 11.

Note that routes are case-sensitive, so visiting `http://127.0.0.1:5000/CSS` instead of `http://127.0.0.1:5000/css` would lead to a 404 Not Found error.

Trailing Slashes

In general, a request's path and a route's path must match **exactly** for the route to be chosen. For instance, the route for `/no_slash` on line 13 is matched only by `http://127.0.0.1:5000/no_slash` (without a trailing slash). If we try visiting

Chapter 18 Part 4: Web Applications (Flask)

`http://127.0.0.1:5000/no_slash/` (with a trailing slash), we will get a 404 (Not Found) error instead.

However, if Flask fails to find a route for a path that does NOT end with a slash, it will append a slash to the path and try again before giving up completely. For instance, both `http://127.0.0.1:5000/optional_slash` (without a trailing slash) and `http://127.0.0.1:5000/optional_slash/` (with a trailing slash) both match the route for `/optional_slash/` and will run the associated `optional_slash()` function.



Multiple Decorators

If needed, multiple paths can also be routed to the same function. For instance, the three decorations on lines 21 to 23 associate all of the following URLs with the `multiple()` function defined on lines 24 and 25:

- `http://127.0.0.1:5000/one/`
- `http://127.0.0.1:5000/one/two/`
- `http://127.0.0.1:5000/three/two/one`

Note that trailing slashes are treated in exactly the same way as described previously, so `http://127.0.0.1:5000/three/two/one/` will result in a 404 (Not Found) error since there is no route that matches `/three/two/one/` (with a trailing slash) exactly.

Variable Routes

Flask routes can also have **variable** parts. Each variable part in the route's path has a name surrounded by angled brackets `<` and `>`. By default, each variable part matches any non-empty sequence of characters that does not contain a slash `/`. The route is matched if all variable parts in the path can be matched in this way. When this happens, the variable parts are extracted into `str` values and passed to the associated Python function as keyword arguments.

For instance, line 27 defines a route specified by `/string/<s>/` that contains one variable part named `s`. This route matches any path starting with `/string/` followed by 1 or more non-slash characters and an optional trailing slash. If a match is found, the variable part is extracted into a `str` value and passed to the function

Chapter 18 Part 4: Web Applications (Flask)

`string_variable()` as a keyword argument named `s`. Otherwise, Flask will try to find another route that matches, returning a 404 (Not Found) error if none can be found.

`http://127.0.0.1:5000/string/variable_part/`

Variable part of entered URL is extracted into a variable named `s`...

```
@app.route('/string/<s>/')

```

...which is then passed to the function as a keyword argument.

```
def string_variable(s):
    return 'Routed to string_variable(), s = {}'.format(s)
```

The following table tabulates some URLs that match the `'/string/<s>/'` route and some URLs that do not. Note how the variable `s` cannot be matched to an empty string or any portion of the path that contains a slash `/`,

URL	Result	Content of Variable <code>s</code>
<code>http://127.0.0.1:5000/string/hello/</code>	200 OK	hello
<code>http://127.0.0.1:5000/string/hello</code>	200 OK	hello
<code>http://127.0.0.1:5000/string/123</code>	200 OK	123
<code>http://127.0.0.1:5000/string/</code>	404 Not Found	
<code>http://127.0.0.1:5000/string/hi/there</code>	404 Not Found	
<code>http://127.0.0.1:5000/string//</code>	404 Not Found	

Variable parts can also specify a **converter** using the syntax `<converter:name>` to modify the matching algorithm and convert the matched string to another type before being passed over to the function as a keyword argument.

For instance, line 31 defines a route specified by `'/integer/<int:i>'` with one variable part named `i` that uses the `int` converter. The `int` converter modifies the matching algorithm for `i` so only digits (i.e., 0 to 9) are accepted. This means that only paths that start with `/integer/` followed by 1 or more digits and an optional trailing slash will result in a match.

If a match is found, the digits portion of the path is extracted and converted to an `int` before being passed to the `integer_variable()` function as an `int` parameter. Otherwise, the match fails and Flask will try to find another route that matches, returning a 404 (Not Found) if no matching route can be found.

URL	Result	Content of Variable <code>i</code>
<code>http://127.0.0.1:5000/integer/123/</code>	200 OK	123
<code>http://127.0.0.1:5000/integer/123</code>	200 OK	123
<code>http://127.0.0.1:5000/integer/0</code>	200 OK	0
<code>http://127.0.0.1:5000/integer/</code>	404 Not Found	
<code>http://127.0.0.1:5000/integer/-123</code>	404 Not Found	
<code>http://127.0.0.1:5000/integer/one</code>	404 Not Found	

Chapter 18 Part 4: Web Applications (Flask)

Note that as the `int` converter only accepts digits, negative integers that start with a minus sign CANNOT be matched by `<int:i>`.

- 2 Complete the following program so it greets the user when the site is visited with the user's name in the path (as long as the name does not include a slash):

```
import flask
app = flask.Flask(__name__)

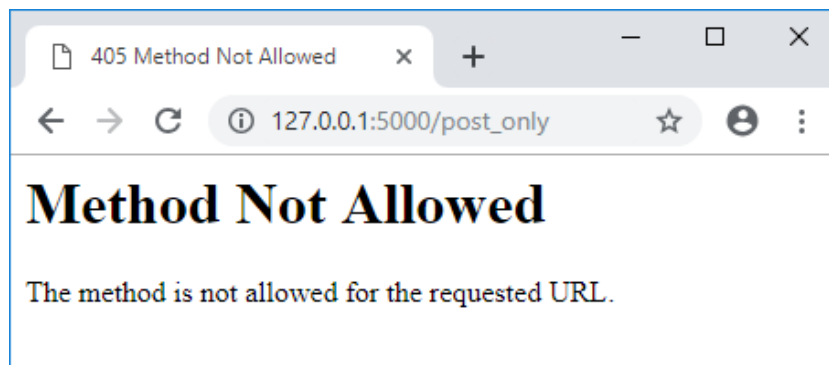
@app.route('/<name>')
def home(name):
    return 'Hello, {}!'.format(name)

if __name__ == '__main__':
    app.run()
```

Routing by HTTP Methods

Besides matching based on paths, each route can also specify if it only applies to GET requests, POST requests or both. Recall that GET is used to retrieve data without making changes while POST is used to submit or make changes to the server's data. To comply with this definition, a route for something that changes the server's data permanently (e.g., deletion) should not be accessible using GET. In other words, a user should not be able to add, delete or update data on the server without submitting a form and sending a POST request instead.

To limit the HTTP methods accepted by a route, we pass in a list of permitted HTTP methods as a keyword argument named `methods` in the decorator. For instance, line 35 specifies a route for `/post_only` that is only accessible using POST. If we try to access `http://127.0.0.1:5000/post_only` using GET by entering it directly into the address bar of a web browser, we get a 405 (Method Not Allowed) error instead.



To produce a POST request, we need to reach the route by submitting a HTML form. This will be demonstrated later in this lesson.

Generating Paths from Function Names

Chapter 18 Part 4: Web Applications (Flask)

Routes provide a mapping from paths to Python functions. However, we often need to go in the opposite direction and generate the path for a given Python function. To do this, we call the `url_for()` function in the `flask` module and pass it a string with the function's name. If the path has any variables (e.g. `s` in `"/string/<s>"`), they should be provided as keyword arguments to `url_for()`.

The following example specifies some routes and prints three generated paths in the shell or command prompt window when the "root" site `http://127.0.0.1:5000/` is visited. Notice how `url_for()` is used on lines 8 to 10.

Program 5: p05_url_lookup.py

```
1 import flask
2 from flask import url_for
3
4 app = flask.Flask(__name__)
5
6 @app.route('/')
7 def home():
8     url1 = url_for('fixed_route')
9     url2 = url_for('string_variable', s='example')
10    url3 = url_for('integer_variable', i=2020)
11    print(url1)
12    print(url2)
13    print(url3)
14    return 'Check your shell or command prompt window'
15
16 @app.route('/fixed/')
17 def fixed_route():
18     return 'Routed to fixed()'
19
20 @app.route('/string/<s>')
21 def string_variable(s):
22     return 'Routed to string_variable(), s = {}'.format(s)
23
24 @app.route('/integer/<int:i>')
25 def integer_variable(i):
26     return 'Routed to integer_variable(), i = {}'.format(i)
27
28 if __name__ == '__main__':
29     app.run()
```

The expected output for `p05_url_lookup.py` is as follows (you may need to look for these lines among the other logging output from Flask):

```
/fixed/
/string/example
/integer/2020
```

Chapter 18 Part 4: Web Applications (Flask)

These correspond to calling `fixed_route()`, `string_variable(s='example')` and `integer_variable(i=2020)` respectively.

- 3 The following program runs a web server at `http://127.0.0.1:5000/`.

```
import flask

app = flask.Flask(__name__)

NAMES = [
    'January', 'February', 'March', 'April',
    'May', 'June', 'July', 'August',
    'September', 'October', 'November', 'December'
]

@app.route('/')
def home():
    return 'Home'

@app.route('/<int:month>/')
def name_month(month):
    if month in range(1, 13):
        return 'Month {}: {}'.format(month, NAMES[month - 1])
    return 'Invalid month'

@app.route('/compare/<float:temp>/')
def compare_temp(temp):
    if temp > 35.5:
        return 'It\'s hot!'
    if temp < 25.5:
        return 'It\'s cold!'
    return 'It\'s normal!'

@app.route('/greet/')
def greet():
    return 'Hello!'

@app.route('/greet/<name>/')
def greet_name(name):
    return 'Hello, {}'.format(name)

@app.route('/data/', methods=['POST'])
def post_data():
    return 'You are using POST'

@app.route('/data/', methods=['GET'])
```

Chapter 18 Part 4: Web Applications (Flask)

```
def get_data():  
    return 'You are using GET'  
  
if __name__ == '__main__':  
    app.run()
```

With this program running in the background, predict the output when each of the URLs below is visited using a browser. If you predict that a HTTP error would occur instead, write "ERROR" as the output.

No.	URL	Output
1	http://127.0.0.1:5000/	Home
2	http://127.0.0.1:5000/10	Month 10: October
3	http://127.0.0.1:5000/10/20/	ERROR
4	http://127.0.0.1:5000/20/	Invalid month
5	http://127.0.0.1:5000/compare/35.4	It's normal!
6	http://127.0.0.1:5000/compare/35.6/	It's hot!
7	http://127.0.0.1:5000/compare/	ERROR
8	http://127.0.0.1:5000/greet/world/	Hello, world!
9	http://127.0.0.1:5000/greet/worLD/	Hello, worLD!
10	http://127.0.0.1:5000/Greet/world/	ERROR
11	http://127.0.0.1:5000/greet/Mei Yi/	Hello, Mei Yi!
12	http://127.0.0.1:5000/greet/	Hello!

Chapter 18 Part 4: Web Applications (Flask)

13	http://127.0.0.1:5000/data/	You are using GET

After completing the table, visit the URLs to check your answers.

Part 4: HTTP Responses and Status Codes

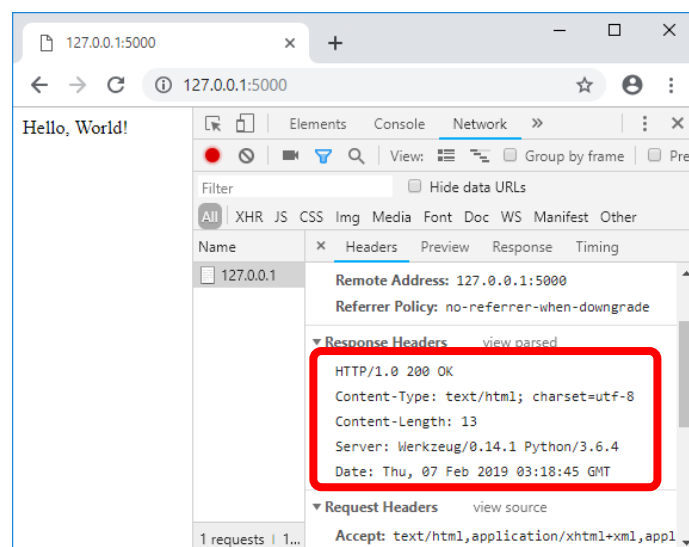
So far, our functions have been returning short strings that appear to display without problem in the web browser. However, Flask has actually been prepending various headers behind the scenes to produce valid HTTP responses.

Run the following program:

Program 6: p06_hello_world.py

```
1 import flask
2
3 app = flask.Flask(__name__)
4
5 @app.route('/')
6 def index():
7     return 'Hello, World!'
8
9 if __name__ == '__main__':
10     app.run()
```

With Google Chrome's Developer Tools open, visit <http://127.0.0.1:5000/> and examine the response headers. You should see that Flask has actually added various headers to form a complete HTTP response.



Chapter 18 Part 4: Web Applications (Flask)

Together with the content, the complete HTTP response sent to your browser is similar to the following:

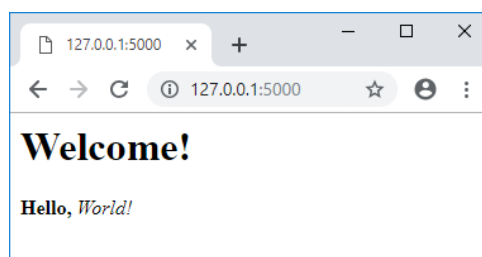
```
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 13
Server: Werkzeug/0.14.1 Python/3.7.0
Date: Tue, 22 Jan 2019 08:01:52 GMT

Hello, World!
```

Notice that Flask assumes that our output has a Content-Type of "text/html". This means that Flask actually expects our function to return a full HTML document and not just a plain string. However, most browsers are very forgiving and will treat our string as a snippet of HTML intended for the document's <body>. For instance, the following program demonstrates that HTML tags can be used in our return value:

Program 7: p07_returning_html.py

```
1 import flask
2
3 app = flask.Flask(__name__)
4
5 @app.route('/')
6 def index():
7     return '<h1>Welcome!</h1> <b>Hello,</b> <i>World!</i>'
8
9 if __name__ == '__main__':
10     app.run()
```



However, be aware that is non-standard behaviour and our functions that are mapped to routes should really return full HTML documents.

Changing the Status Code

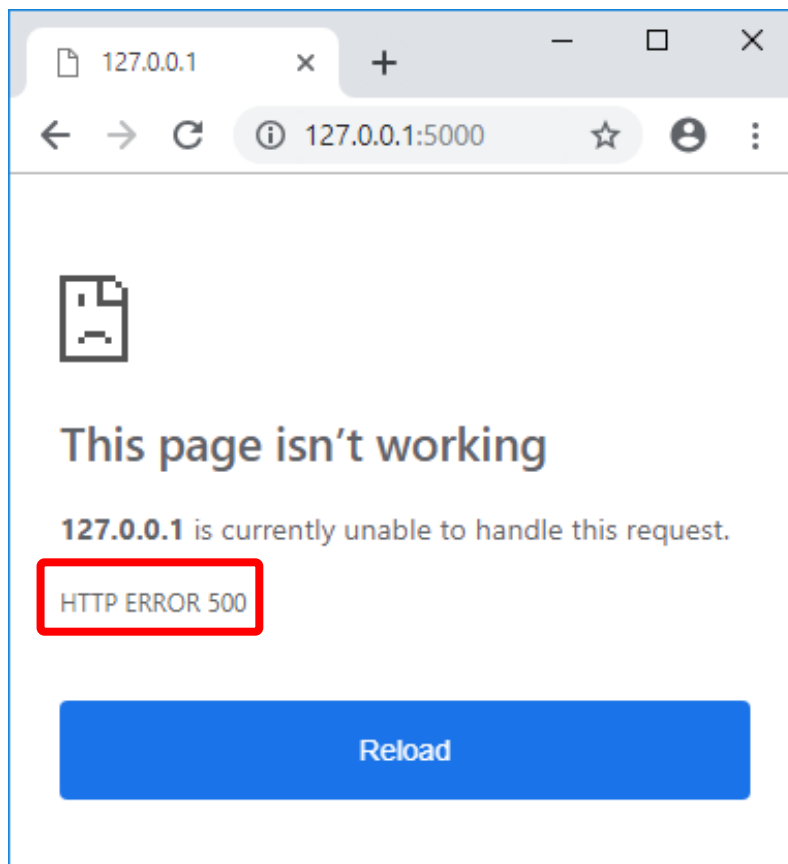
By default, Flask also assumes that our responses have a HTTP status code of 200 (OK). This is usually what we want, but if needed we can override this by returning a tuple instead of just a string. The replacement HTTP status code should be provided as the second item of the tuple, as demonstrated on line 7 of the following example:

Chapter 18 Part 4: Web Applications (Flask)

Program 8: p08_returning_status_500.py

```
1 import flask
2
3 app = flask.Flask(__name__)
4
5 @app.route('/')
6 def index():
7     return ('', 500)
8
9 if __name__ == '__main__':
10     app.run()
```

Recall that a HTTP status code of 500 represents an Internal Server Error. If we run this program and try to visit `http://127.0.0.1:5000/`, we should be greeted with an HTTP 500 (Internal Server Error) status code, as expected:



Changing the Response Headers

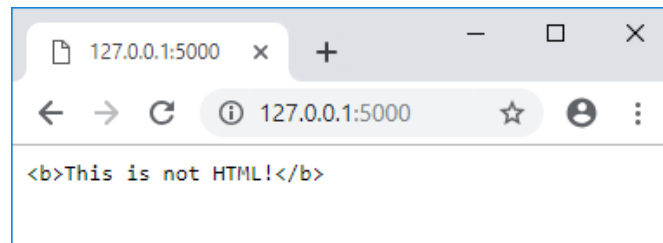
We can also add additional response headers by putting them into a Python dictionary and returning this dictionary as the third item of our return tuple. For instance, if we want the web browser to treat our response as plain text instead of HTML, we can replace the Content-Type header value with "text/plain" instead:

Chapter 18 Part 4: Web Applications (Flask)

Program 9: p09_returning_plain_text.py

```
1 import flask
2
3 app = flask.Flask(__name__)
4
5 @app.route('/')
6 def index():
7     headers = {'Content-Type': 'text/plain'}
8     return ('<b>This is not HTML!</b>', 200, headers)
9
10 if __name__ == '__main__':
11     app.run()
```

Running this program and visiting `http://127.0.0.1:5000/` demonstrates that the string we return is no longer treated as HTML by the browser:



Redirecting to Another URL

Besides overriding the HTTP status code and response headers, Flask also lets us generate a response that tells the web browser to load a different URL instead. This is called a **redirect** and is useful when the location of a document has moved or when we want to let another Flask route take over the handling of a request. To perform a redirect, import the `redirect()` function from the `flask` module and call it with the destination URL or path that as the first argument. Then, use the response generated by `redirect()` as the return value of the function, as in the following example:

Program 10: p10_redirect_example.py

```
1 import flask
2 from flask import redirect
3
4 app = flask.Flask(__name__)
5
6 @app.route('/')
7 def index():
8     return redirect('http://example.com')
9
10 if __name__ == '__main__':
11     app.run()
```


Chapter 18 Part 4: Web Applications (Flask)

Instead of redirecting to an external site, we usually want to redirect the user to another of our routes instead. In such cases, we should use `url_for()` to look up the correct path based on the function that we want to reach:

Program 11: `p11_redirect_using_url_for.py`

```
1 import flask
2 from flask import redirect, url_for
3
4 app = flask.Flask(__name__)
5
6 @app.route('/new_url/')
7 def moved_index():
8     return 'You have reached the new URL!'
9
10 @app.route('/')
11 def index():
12     return redirect(url_for('moved_index'))
13
14 if __name__ == '__main__':
15     app.run()
```

Notice how `url_for()` is used to look up the path for `redirect()` on line 12 instead of hardcoding the redirected path as a string.

Part 5: Templates and Rendering

Instead of returning short HTML snippets or redirecting users, the following program illustrates how we can return full HTML documents complete with headings and hyperlinks in our Flask application:

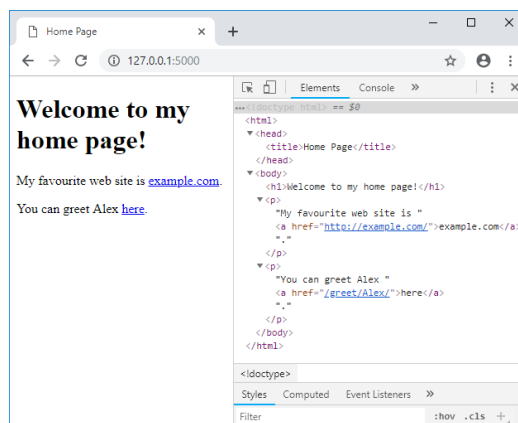
Program 12: `p12_html_response_without_templates.py`

```
1 import flask
2
3 app = flask.Flask(__name__)
4
5 @app.route('/')
6 def home():
7     html = '<!DOCTYPE html>\n<html>'
8     html += '<head><title>Home Page</title></head>'
9     html += '<body><h1>Welcome to my home page!</h1>'
10    html += '<p>My favourite web site is '
11    html += '<a href="http://example.com/">'
```

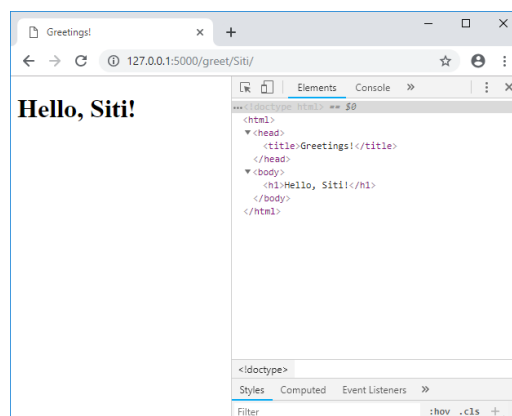
Chapter 18 Part 4: Web Applications (Flask)

```
12     html += 'example.com</a>.</p>'
13     html += '<p>You can greet Alex '
14     html += '<a href="/greet/Alex/">here</a>.</p>'
15     html += '</body></html>'
16     return html
17
18 @app.route('/greet/<name>/')
19 def greet(name):
20     html = '<!DOCTYPE html>\n<html>'
21     html += '<head><title>Greetings!</title></head>'
22     html += '<body><h1>Hello, {}!</h1>'.format(name)
23     html += '</body></html>'
24     return html
25
26 if __name__ == '__main__':
27     app.run()
```

Run the above program and visit <http://127.0.0.1:5000/>. Notice that a full HTML page appears, complete with a page title, headings and hyperlinks. Use the View Source feature by pressing Ctrl-U to verify that the HTML for the page comes directly from the `home()` function of the above Python program.



Now visit <http://127.0.0.1:5000/greet/Siti/>. Notice that the result has a page title and the greeting for Siti is formatted to look like a heading using the `<h1>` tag.



Chapter 18 Part 4: Web Applications (Flask)

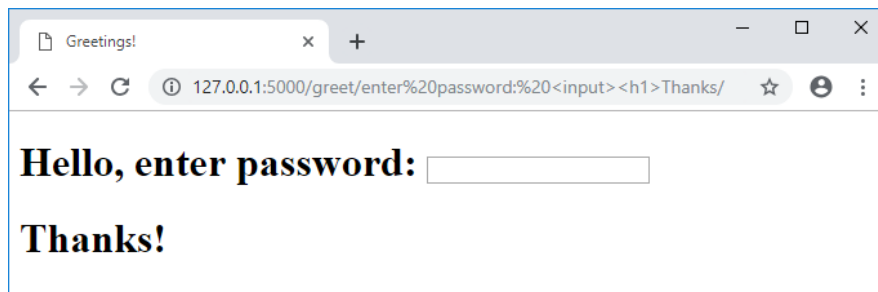
Why Use Templates?

Generating dynamic HTML content in this way is powerful but also dangerous. Notice that in the `greet_name()` function we inserted the `name` variable into our output without checking its contents. This lets a malicious user inject his or her own code into the output and potentially cause all kinds of mischief.

For instance, if we follow the following link:

<http://127.0.0.1:5000/greet/enter%20password:%20<input><h1>Thanks/>

We will get what appears to be a legitimate form asking for the user's password. However, in reality, the form does not come from our application at all and was injected into the page from HTML code in the URL's path.



Besides possible security issues, constructing HTML documents by joining Python strings can also become quite messy. For instance, it is easy to be confused between the use of HTML and Python in the `home()` and `greet_name()` functions.

This is why, in practice, we usually do not generate HTML responses by manually manipulating strings in Python code. Instead, we put the HTML content in a separate file called a **template** with placeholders for where the dynamic content should be inserted. Whenever we need to output HTML, we use a **template engine** to load the template and fill in the placeholders. The template engine also helps to escape special characters such as `<` and `>` when filling in the placeholders so that HTML injection similar to the case above is avoided. This process of filling in the placeholders to produce the final HTML that is used for the response is called **rendering**.

The Jinja2 Template Engine

Flask provides a built-in template engine named **Jinja2**. By default, Flask expects all templates to be located in a subfolder named `templates`.

To start using Jinja2, create a subfolder named `templates` in the folder where your Flask programs are stored, then save the following file in this folder as `home.html`:

Template 1: templates/home.html

```
1 <!DOCTYPE html>
2 <html>
```

Chapter 18 Part 4: Web Applications (Flask)

3	<head><title>Home Page</title></head>
4	<body>
5	Welcome to my home page!
6	<p>My favourite web site is
7	example.com.</p>
8	<p>You can greet Alex
9	here.
10	</p>
11	</body>
12	</html>

Also save the following file in the templates folder as greet.html:

Template 2: templates/greet.html

1	<!DOCTYPE html>
2	<html>
3	<head><title>Greetings!</title></head>
4	<body>
5	Hello, {{ visitor }}!
6	</body>
7	</html>

In a Jinja2 template, placeholders are surrounded by double braces and have the form `{{ expression }}` where expression is a **Jinja2 expression**. When using templates, be careful not to confuse Jinja2 expressions with Python expressions. Although they are very similar, Jinja2 expressions and Python expressions have different syntaxes and operate in different environments. For instance, `len()` and many other standard Python functions are not available in Jinja2 expressions. You will learn more about the differences between Jinja2 and Python in the following sections.

Recall that the process of filling in the placeholders of a template is called **rendering**. The task of rendering a template is typically performed by a function named `render_template()` that can be imported from the flask module, like so:

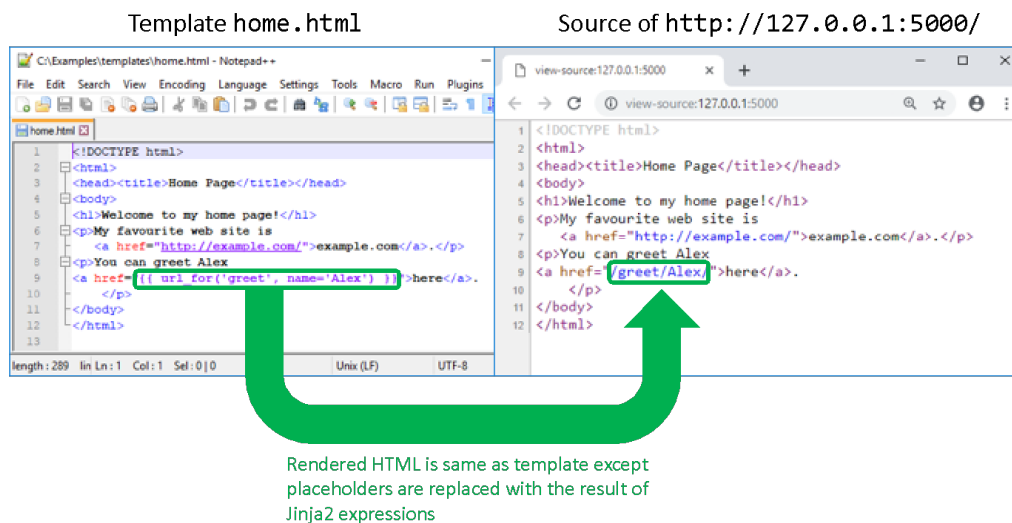
Program 13: p13_html_response_with_templates.py

1	import flask
2	from flask import render_template
3	
4	app = flask.Flask(__name__)
5	
6	@app.route('/')
7	def home():
8	return render_template('home.html')
9	

Chapter 18 Part 4: Web Applications (Flask)

```
10 @app.route('/greet/<name>/')
11 def greet(name):
12     return render_template('greet.html', visitor=name)
13
14 if __name__ == '__main__':
15     app.run()
```

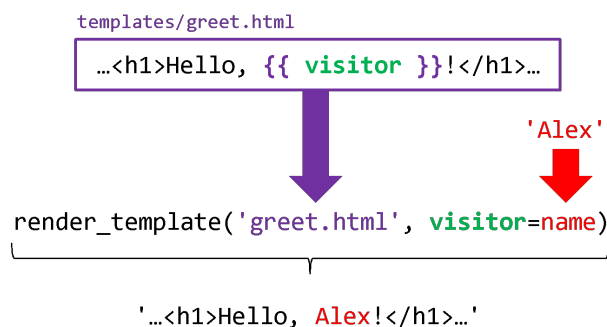
Run this program and visit <http://127.0.0.1:5000/>. View Source on the page by pressing Ctrl-U to verify that the template is indeed rendered and no placeholders are present. Click the link to visit <http://127.0.0.1:5000/greet/Alex/> and verify that there are no placeholders in its HTML source as well.



Passing Values to Templates

The `render_template()` function accepts the name of a template in the templates subfolder as its first argument, followed by 0 or more keyword arguments assigning values to Jinja2 variables that may be used by the template.

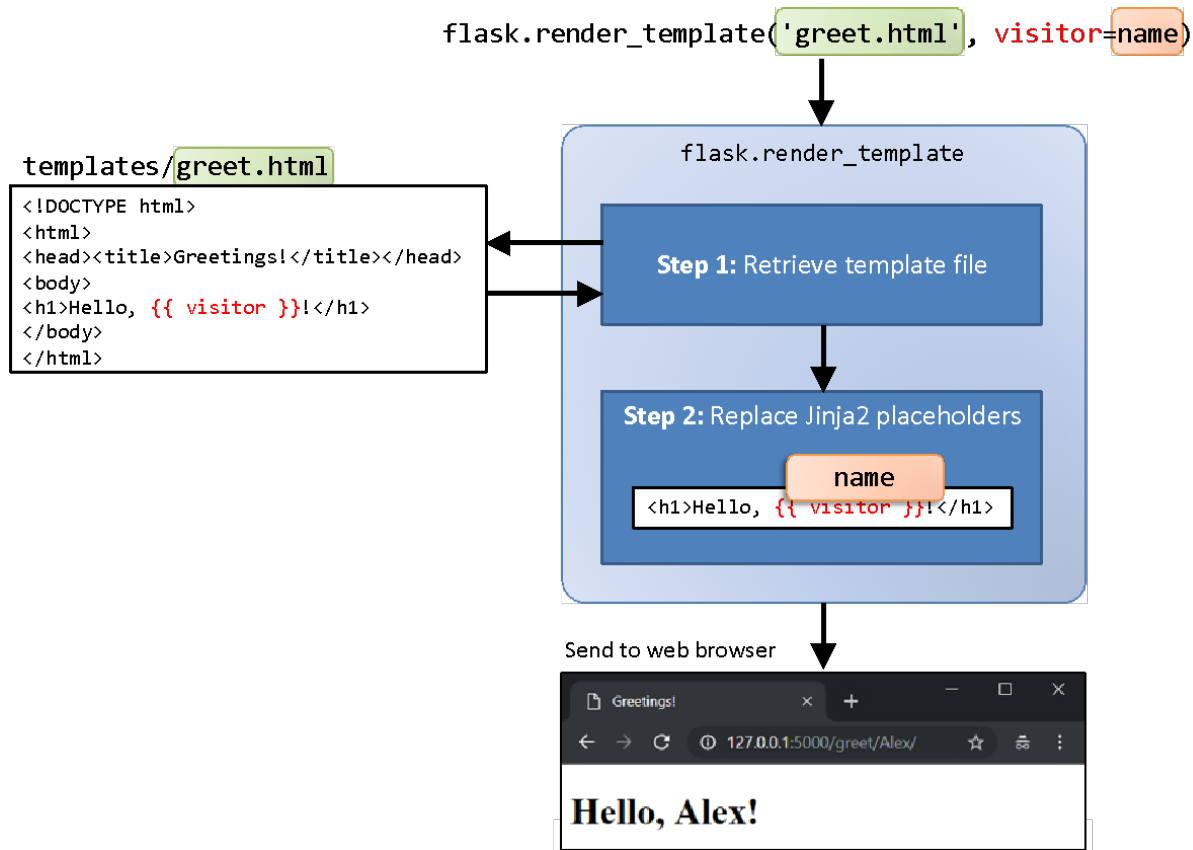
For instance, line 12 of `p13_html_example_with_templates.py` renders a template named `greet.html` and specifies that within the template, the value of `name` should be assigned to a Jinja2 variable named `visitor`. This corresponds to line 5 of `greet.html` where the Jinja2 placeholder `{{ visitor }}` is replaced by this value.



Chapter 18 Part 4: Web Applications (Flask)

Do not confuse Jinja2 variables with Python variables. If `render_template()` is called without any keyword arguments, values from the Python environment are NOT passed to the template. To use Python values in the generated HTML, we must pass them over as keyword arguments when `render_template()` is called.

The following diagram illustrates how `render_template()` retrieves a template file and replaces its placeholders to produce the final HTML that is sent to the browser:



- 4 Create the following four templates and save them in a `templates/` subfolder:

`templates/A.html`

```
<!doctype html>
<html>
<head><title>A</title></head>
<body>A</body>
</html>
```

`templates/B.html`

```
<!doctype html>
<html>
<head><title>B</title></head>
<body>B</body>
</html>
```

Chapter 18 Part 4: Web Applications (Flask)

templates/C.html

```
<!doctype html>
<html>
<head><title>C</title></head>
<body>C</body>
</html>
```

templates/D.html

```
<!doctype html>
<html>
<head><title>D</title></head>
<body>D</body>
</html>
```

Each template is meant to be shown for a different URL:

URL	Template to use
http://127.0.0.1:5000/A	A.html
http://127.0.0.1:5000/B	B.html
http://127.0.0.1:5000/C	C.html
http://127.0.0.1:5000/D	D.html

Complete the following program accordingly:

app.py

```
import flask
app = flask.Flask(__name__)
@app.route('/A')
def view_A():
    return flask.render_template('A.html')

@app.route('/B')
def view_B():
    return flask.render_template('B.html')

@app.route('/C')
def view_C():
    return flask.render_template('C.html')

@app.route('/D')
def view_D():
    return flask.render_template('D.html')

if __name__ == '__main__':
    app.run()
```

Chapter 18 Part 4: Web Applications (Flask)

- 5 Generalise the previous program and templates so that the application only needs one template instead of four templates:

templates/template.html

```
<!doctype html>
<html>
<head><title>{{ letter }}</title></head>
<body>{{ letter }}</body>
</html>
```

app.py

```
import flask
app = flask.Flask(__name__)

@app.route('/<letter>')
def view(letter):
    return flask.render_template('template.html',
                                { 'letter': letter })

if __name__ == '__main__':
    app.run()
```

Notice that this application allows other combinations of URL, for example:

<http://127.0.0.1:5000/BBB>, <http://127.0.0.1:5000/q>.

Go further! Edit the program above so only the four cases given in Question 4 are allowed and return error code 500 for other cases.

Avoiding Hardcoded Links

Although most Python functions are not available from Jinja2, `render_template()` automatically includes Flask's `url_for()` function so paths can be generated based on the Python function that needs to be called instead being hardcoded. This is most useful for creating HTML links, as demonstrated by lines 8 and 9 of `home.html`:

```
<p>You can greet Alex
  <a href="{{ url_for('greet', name='Alex') }}">here</a>.
```

Using `url_for()` is more future-proof than hardcoding the path like this:

```
<p>You can greet Alex <a href="/greet/Alex/">here</a>.
```

Hardcoded links are shorter but need to be updated every time we change the path that is routed with a function. Using `url_for()` lets us update our paths without having to change every template that links to it.

Chapter 18 Part 4: Web Applications (Flask)

6 Enter the following program:

app.py

```
import flask
app = flask.Flask(__name__)

@app.route('/')
def home():
    return flask.render_template('links.html')

@app.route('/greeting')
def hello():
    return 'Hello, World!'

@app.route('/<int:year>')
def report(year):
    return 'year is ' + str(year)

if __name__ == '__main__':
    app.run()
```

Use the `url_for()` function to complete the following template so that each link will display the respective message when clicked:

templates/links.html

```
<!doctype html>
<html>
<head><title>Links</title></head>
<body>

    <p><a href="{{ url_for('hello') }}">Hello, World!</a></p>

    <p><a href="{{ url_for('report', year=2017) }}">year is
2017</a></p>

</body>
</html>
```

The safe Filter

With `p13_html_response_with_templates.py` running, visit the URL that previously allowed us to inject HTML into the greeting page:

<http://127.0.0.1:5000/greet/enter%20password:%20<input><h1>Thanks/>

This time, however, the injected code is displayed as plain text instead of being treated as HTML. If you press Ctrl-U to view source, you would notice that the less-than < and greater-than > signs have been escaped with the appropriate HTML entities:

Chapter 18 Part 4: Web Applications (Flask)



Jinja2 automatically performs this escaping for us so user input can be used in Jinja2 expressions freely without any safety concerns.

However, if we are sure that the result of a Jinja2 expression is safe and should be treated as raw HTML, we can notify Jinja2 of this fact by using the **safe filter**. To apply a filter to an expression, we follow the expression with a bar character | and the name of the filter, as in the example below:

Template 3: templates/custom.html

```
1 <!DOCTYPE html>
2 <html>
3 <head><title>Custom HTML</title></head>
4 <body>
5 <h1>Custom HTML</h1>
6 {{ my_html|safe }}
7 </body>
8 </html>
```

Save this custom.html in the templates subfolder and run the following program from the parent folder:

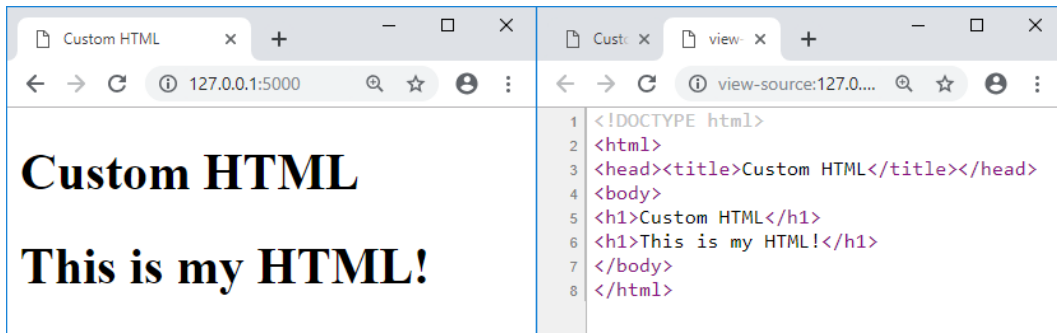
Program 14: p14_raw_html_with_safe_filter.py

```
1 import flask
2 from flask import render_template
3
4 app = flask.Flask(__name__)
5
6 @app.route('/')
7 def home():
```

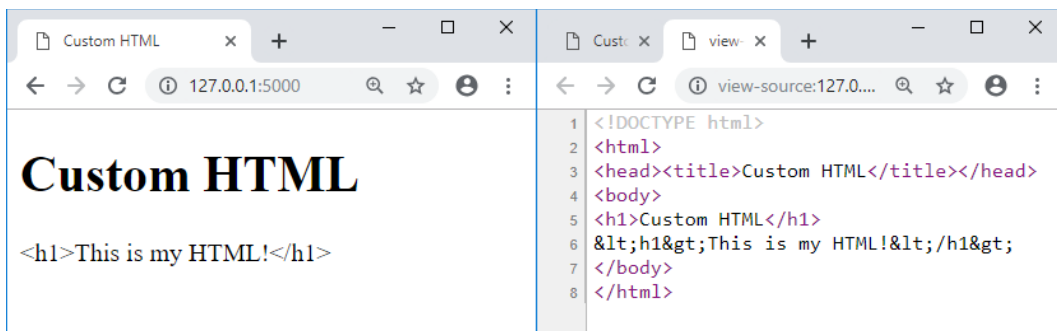
Chapter 18 Part 4: Web Applications (Flask)

```
8     return render_template('custom.html',
9                             my_html='<h1>This is my HTML!</h1>')
10
11 if __name__ == '__main__':
12     app.run()
```

Visit <http://127.0.0.1:5000/> and you should see that the string passed to the template as `my_html` is rendered as raw HTML:



However, if we remove the safe filter from `custom.html`, `my_html` is rendered such that the special characters `<` and `>` are escaped, as expected:



Technical Note

You must restart your program for any changes (such as removing the safe filter) to take effect. If your program appears to stop reflecting changes that you have made, restart the program, then hold down the Ctrl key while refreshing so your browser bypasses its cache and performs a fresh HTTP request.

Alternatively, you may wish to run Flask in debug mode by replacing `app.run()` with `app.run(debug=True)`. This lets Flask reload itself whenever it detects any file changes so manual restarts are no longer needed. However, running Flask in debug mode has several incompatibilities with IDLE:

1. When using debug mode, the usual start-up messages that appear when `app.run()` is called will not appear in IDLE's shell window.

Chapter 18 Part 4: Web Applications (Flask)

2. Similarly, output from `print()` and error messages produced by your program will not appear in IDLE's shell window.
3. Pressing Ctrl-C or restarting IDLE's shell will not stop the server properly, so the next time you try to restart your program, you may receive an error message informing you that port number 5000 is already in use. As a workaround, save your work, then press Ctrl-Alt-Del to start Task Manager and close all tasks with the name `python.exe` or `pythonw.exe`. This will close IDLE and any other stray servers that may still be running. After all copies of Python are closed, restart IDLE and try running your program again.

These issues only occur when using IDLE. If you still wish to use debug mode, it is highly recommended that you run your program from the Command Prompt or PowerShell instead or use an alternative IDE.

The Length Filter

Since the `len()` function is not available in Jinja2 expressions, you might wonder how it is possible to output the length of a string or list from a template. One solution would be to perform the calculation in Python and pass the value over to the template as a separate Jinja2 variable. However, for simple length checks Jinja2 provides a `length` filter that gives the same result as `len()` but uses a different syntax:

Template 4: `templates/length.html`

1	<code><!DOCTYPE html></code>
2	<code><html></code>
3	<code><head><title>Length of Name</title></head></code>
4	<code><body></code>
5	<code><h1>Length of Name</h1></code>
6	<code>Hello {{ name }},</code>
7	<code>your name is {{ name length }} characters long!</code>
8	<code></body></code>
9	<code></html></code>

Save the above `length.html` in the `templates` subfolder and run the following program from the parent folder:

Program 15: `p15_name_with_length_filter.py`

1	<code>import flask</code>
2	<code>from flask import render_template</code>
3	

Chapter 18 Part 4: Web Applications (Flask)

```
4 app = flask.Flask(__name__)
5
6 @app.route('/<name>/')
7 def length_of_name(name):
8     return render_template('length.html', name=name)
9
10 if __name__ == '__main__':
11     app.run()
```

Visit different URLs such as <http://127.0.0.1:5000/Elizabeth/> or <http://127.0.0.1:5000/Siti/> and examine how line 7 of the template uses the length filter to determine the number of characters in name.

Jinja2 Statements

In a typical Flask application, the majority of data processing and computation should be done using Python code. The outputs from this computation are then passed to Jinja2 as numbers, strings, lists and other plain data objects. This data is then used to fill in a template to produce the final HTML. Although it is possible to perform some data processing and computation using Jinja2, performing complex logic in a template is not recommended.

Nevertheless, sometimes it is useful to perform some simple logic in a template, such as to selectively render parts of a template or to repeat a portion of the template for every item in a list. To perform these tasks, Jinja2 supports control flow in a template using **Jinja2 statements** made up of commands surrounded by `{%` and `%}`. Unlike placeholders surrounded by double braces `{{` and `}}` that are usually replaced with output, the contents of `{%` and `%}` do not produce any output. For our needs, we will cover two types of Jinja2 statements: `if` statements and `for-in` statements.

if Statement

Similar to the `if` statement in Python, the Jinja2 `if` statement is used to selectively include or exclude portions of the template. Excluded portions of a template are simply not rendered. However, unlike Python, Jinja2 is not grouped by indentation, so there needs to be an `endif` command to indicate where the `if` statement ends. In addition, since the `if`, `elif` and `else` clauses are now demarcated by separate `{%` and `%}` blocks, colons are no longer needed:

Template 5: templates/results.html

```
1 <!DOCTYPE html>
2 <html>
3 <head><title>Results</title></head>
4 <body>
5 <h1>Results</h1>
6
7 {% if greet %}
```

Chapter 18 Part 4: Web Applications (Flask)

```
8      <p>Hello, {{ name }}.</p>
9      {% endif %}
10
11      {% if show_score %}
12          <p>Your score is {{ score }}%.</p>
13      {% elif score >= 50 %}
14          <p>You passed.</p>
15      {% else %}
16          <p>You failed.</p>
17      {% endif %}
18
19  </body>
20  </html>
```

Run the following program and visit <http://127.0.0.1:5000/> to see the template rendered for a score of 72:

Program 16: p16_results_using_if.py

```
1  import flask
2  from flask import render_template
3
4  app = flask.Flask(__name__)
5
6  @app.route('/')
7  def home():
8      return render_template('results.html', greet=True,
9                             name='Alex', show_score=False, score=72)
10
11  if __name__ == '__main__':
12      app.run()
```

To explore further, adjust the `greet`, `name`, `show_score` and `score` keyword arguments on lines 8 and 9, restart the server and visit <http://127.0.0.1:5000/> again to see how the template is rendered differently. In each case, verify that the template's behaviour matches up with what you would expect.

for-in Statement

Similar to the `for-in` statement in Python, the Jinja2 `for-in` statement is used to repeat the rendering of a portion of the template for every item in a list, tuple, string or dictionary. However, since Jinja2 is not grouped by indentation, like the `if` statement there needs to be an `endfor` command to indicate where the `for` statement ends.

A typical use of `for-in` is to output the contents of a collection in a table:

Chapter 18 Part 4: Web Applications (Flask)

Template 6: templates/table.html

```
1 <!DOCTYPE html>
2 <html>
3 <head><title>Table of Results</title></head>
4 <body>
5 <h1>Table of Results</h1>
6
7 <table>
8 <tr><th>Subject Name</th><th>Score</th></tr>
9 {% for subject in results %}
10     <tr>
11         <td>{{ subject }}</td>
12         <td>{{ results[subject] }}</td>
13     </tr>
14 {% endfor %}
15 </table>
16
17 </body>
18 </html>
```

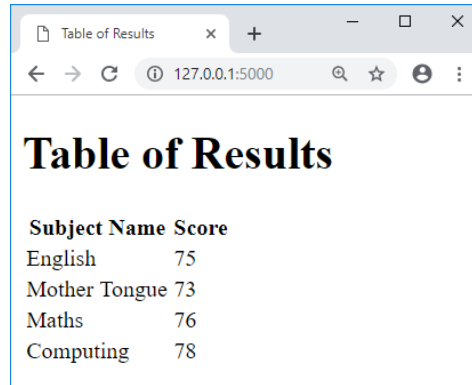
For this example, table.html expects a Jinja2 variable named results containing a dictionary mapping subject names to scores. The following program creates a hardcoded dictionary matching this format and passes it to the template for rendering:

Program 17: p17_table_using_for.py

```
1 import flask
2 from flask import render_template
3
4 app = flask.Flask(__name__)
5
6 @app.route('/')
7 def home():
8     results = {
9         'English': 75,
10        'Mother Tongue': 73,
11        'Maths': 76,
12        'Computing': 78
13    }
14    return render_template('table.html', results=results)
15
16 if __name__ == '__main__':
17     app.run()
```

Run the following program and visit <http://127.0.0.1:5000/> to see how the hardcoded data in the dictionary is rendered as a HTML table:

Chapter 18 Part 4: Web Applications (Flask)



Subject Name	Score
English	75
Mother Tongue	73
Maths	76
Computing	78

In a real web application, we would most likely retrieve the subject names and scores from an SQLite database instead of hardcoding them.

Part 6: Static Files

Recall that requested paths are treated as strings for route matching by Flask and do not usually refer to the location of real files or folders that are stored on the server. This lets us return a dynamically-generated response using `render_template()` instead of returning a static file.

However, most HTML documents do not exist in isolation. They typically request for additional resources such as style sheets and images from additional URLs in order to display properly. However, unlike the main content, these additional resources do not change often, so instead of running a Python function to generate the response, Flask lets us create a subfolder named `static` (in the same location as the `templates` subfolder) to store these resources. Flask then sets up a view named `'static'` that (by default) routes any path starting with `/static/` to the contents of this subfolder.

For example, create a subfolder named `static` in the folder where your Flask programs are stored, then save the following file in this folder as `styles.css`:

Style Sheet 1: `static/styles.css`

```
1 body {
2     background: yellow;
3 }
4
5 h1 {
6     border-bottom: 1px solid red;
7     color: red;
8 }
```


Chapter 18 Part 4: Web Applications (Flask)

Now, let's create a template that makes use of this style sheet. For simplicity, we can hardcode the URL of the style sheet `/static/styles.css`. However, it is preferred to use `url_for()` in case the path for static files changes in the future:

Template 7: templates/stylish.html

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <title>Stylish Page</title>
5     <link
6         rel="stylesheet"
7         href="{ url_for('static', filename='styles.css') }}">
8     <!--
9         The above can also be written as:
10    <link rel="stylesheet" href="/static/styles.css">
11    -->
12 </head>
13 <body>
14 <h1>Stylish Page</h1>
15 <p>Look at how stylish this page is!</p>
16 </body>
17 </html>
```

Note how the path of the static file within the `static` subfolder is passed to `url_for()` as a keyword argument named `filename` on line 7.

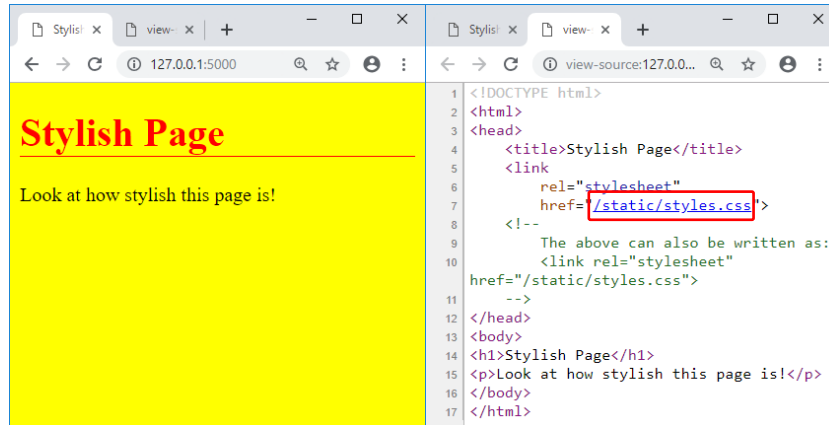
Finally, let us write a simple Python program to render this template:

Program 18: p18_static_style_sheet.py

```
1 import flask
2 from flask import render_template
3
4 app = flask.Flask(__name__)
5
6 @app.route('/')
7 def home():
8     return render_template('stylish.html')
9
10 if __name__ == '__main__':
11     app.run()
```

Visit `http://127.0.0.1:5000/` with this program running and you should see the styled page. If you use the View Source feature by pressing Ctrl-U, you can also verify that the style sheet URL placeholder is replaced with a path starting with `/static/`:

Chapter 18 Part 4: Web Applications (Flask)



Part 7: Processing Form Data

So far, we have been able to get input from the user by extracting variables from request URLs. However, it is more common for users to provide input to a web application by filling in a HTML form and submitting it. When a HTML form is submitted, the browser collects all the input as key-value pairs and encodes it into a single string. This string is then sent to the server using a HTTP request. Depending on how the HTML form is configured, either a GET request or POST request is made.

GET Requests

The default behaviour for HTML forms is to submit a GET request. When this method is used, the encoded form data is visible in the query portion of the request URL. For instance, the URL `http://example.com/hello?name=bala&age=18` contains a query with two key-value pairs: one for the key 'name' and another for 'age'.

Parsing key-value pairs encoded in query strings can be tedious and error-prone. Thankfully, if a query string is present, Flask does this parsing for us and lets us access it as a dictionary from a request object that can be imported from the flask module.

To illustrate this, create the following two templates in your templates subfolder. The first template is used to display the HTML form:

Template 8: templates/form_with_get.html

```
1 <!DOCTYPE html>
2 <html>
3 <head><title>GET Form</title></head>
4 <body>
5   <form action="{{ url_for('process_with_get') }}">
6     <p>Input s: <input name="s"></p>
7     <p><input type="submit"></p>
8   </form>
9 </body>
```

Chapter 18 Part 4: Web Applications (Flask)

10	</html>

Notice that on line 5 we use `url_for()` with a function name of `'process_with_get'` to generate the path that the form data will be submitted to. We will write the code for `process_with_get()` later.

The second template is used to display the results of processing. In this case, let us display the number of vowels and number of words in the submitted name. Save the following file in the same folder as `analysis_results.html`:

Template 9: templates/analysis_results.html

1	<!DOCTYPE html>
2	<html>
3	<head><title>String Analysis</title></head>
4	<body>
5	You entered s: {{ s }}
6	<p>This string has
7	{{ num_vowels }} vowels(s) and
8	{{ num_words }} word(s).
9	</body>
10	</html>

Next, let us write a Flask application with two routes: the root `/` route simply renders the form while the `/process/` route is associated with a Python function named `process_with_get()` that analyses `s` and renders the result:

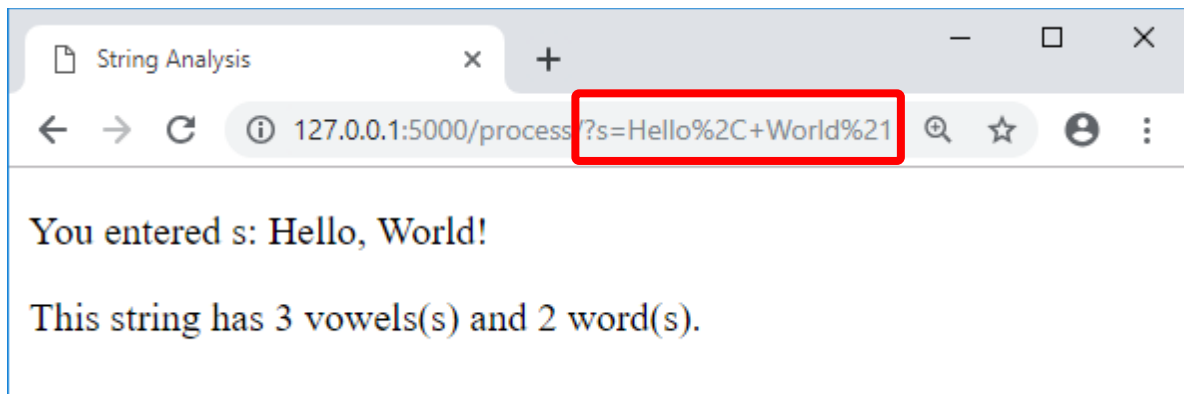
Program 19: p19_analysis_with_get.py

1	import flask
2	from flask import render_template, request
3	
4	VOWELS = ['a', 'e', 'i', 'o', 'u']
5	
6	app = flask.Flask(__name__)
7	
8	@app.route('/') def form():
9	return render_template('form_with_get.html')
10	
11	
12	@app.route('/process/') def process_with_get():
13	if 's' in request.args:
14	s = request.args['s']
15	lower_s = s.lower()
16	num_vowels = 0
17	for vowel in VOWELS:
18	if vowel in lower_s:

Chapter 18 Part 4: Web Applications (Flask)

```
19         num_vowels += lower_s.count(vowel)
20         num_words = len(s.split())
21         return render_template('analysis_results.html',
22                                s=s, num_vowels=num_vowels, num_words=num_words)
23     return 'No form data found!'
24
25 if __name__ == '__main__':
26     app.run()
```

Notice that the request object is imported from the flask module on line 2 and that the submitted value of s is accessed via the request.args dictionary on line 15. Visit <http://127.0.0.1:5000/> and fill in the form. When you submit the form, notice that the value of s you entered is encoded in the resulting URL.



Also note that it is possible to manually visit <http://127.0.0.1:5000/process/> without filling in the form. If you do so, the request.args dictionary will not include a value for 's' and the error message on line 23 will be returned instead.

POST Requests

Submitting form data using a GET request has several disadvantages:

1. Firstly, the submitted form data is recorded in the resulting URL, which lets anyone view your browser history to obtain the form data that was sent.
2. Secondly, GET requests are not supposed to make changes to the server's data. If we use data submitted with a GET request to add, delete or update data from a database, we are not following the HTTP standard.
3. Finally, some browsers and server software limit the length of URLs, so there is a risk that overly long form data submitted using GET may get truncated.

To overcome these disadvantages, we can configure HTML forms to submit the data using POST requests instead. We do this by setting the method attribute of the <form> tag to "post" or "POST". Using a different HTTP method also lets us distinguish between requests from users clicking a link or entering the URL in an address bar

Chapter 18 Part 4: Web Applications (Flask)

versus requests from users submitting a form. We can take advantage of this to use the same URL for both *displaying* the form as well as *processing* the form.

To illustrate this, create the following template in your templates subfolder:

Template 10: templates/form_with_post.html

```
1 <!DOCTYPE html>
2 <html>
3 <head><title>POST Form</title></head>
4 <body>
5 <form method="post">
6     <p>Input s: <input name="s"></p>
7     <p><input type="submit"></p>
8 </form>
9 </body>
10 </html>
```

Notice that besides adding the method attribute on line 5, we have also removed the action attribute so the form is submitted to the same URL used to generate the form.

Next, let us adapt 19_analysis_with_get.py to combine both routes and distinguish between the two situations by looking at the HTTP method used. The HTTP method can be easily determined via the request.method string:

Program 20: p20_analysis_with_post.py

```
1 import flask
2 from flask import render_template, request
3
4 VOWELS = ['a', 'e', 'i', 'o', 'u']
5
6 app = flask.Flask(__name__)
7
8 @app.route('/', methods=['GET', 'POST'])
9 def index():
10     if request.method == 'GET':
11         return render_template('form_with_post.html')
12     if 's' in request.form:
13         s = request.form['s']
14         lower_s = s.lower()
15         num_vowels = 0
16         for vowel in VOWELS:
17             num_vowels += lower_s.count(vowel)
18         num_words = len(s.split())
19         return render_template('analysis_results.html',
20                               s=s, num_vowels=num_vowels, num_words=num_words)
21     return 'No form data found!'
```

Chapter 18 Part 4: Web Applications (Flask)

```
22
23 if __name__ == '__main__':
24     app.run()
```

Here, `index()` checks if the request method is 'GET', and if so, simply renders the form and exits. Otherwise, the request method is assumed to be 'POST' and the submitted form data is processed instead.

Notice that unlike GET requests where submitted form data is available from the `request.args` dictionary, for POST the submitted form data is placed in a separate `request.form` dictionary instead. For instance, the submitted value of `s` is accessed via the `request.form` dictionary on line 13.

Also note that the route decorator on line 8 must explicitly include 'POST' in its list of allowed methods as Flask routes are only accessible via 'GET' by default.

Now, with the program running, visit `http://127.0.0.1:5000/` and fill in the form. When you submit the form, notice that the value of `s` you entered is NOT visible in the resulting URL and that the URL displayed in the address bar remains the same.

Notice on line 12 that we still check if the `request.form` dictionary has a value for 's' before proceeding. Even though it is relatively hard for an average user to send phony POST requests, it is not technically difficult and our program must be prepared to handle POST requests with incomplete or invalid form data. In this case, if there is no value for 's' in `request.form`, we return an error message on line 21 instead.

7 Enter the following and save it as a template named `form.html`:

`templates/form.html`

```
<!doctype html>
<html>
<head><title>Greeting Form</title></head>
<body>
    <form method="POST">
        <input type="text" name="name">
        <input type="text" name="address">
        <input type="submit">
    </form>
</body>
</html>
```

Complete the following program and template so visiting `http://127.0.0.1:5000/` will render `form.html` and submitting the form will display "Hello *name*, your address is: *address*", where *name* is the content of the first text field and *address* is the content of the second text field:

`templates/results.html`

Chapter 18 Part 4: Web Applications (Flask)

```
<!doctype html>
<html>
<head><title>Greeting</title></head>
<body>
    Hello {{ name }}, your address is: {{ address }}
</body>
</html>
```

app.py

```
import flask
from flask import render_template, request
app = flask.Flask(__name__)

# Use the space below
@app.route('/', methods=['GET', 'POST'])
def home():
    if request.method == 'GET':
        return render_template('form.html')
    return render_template('results.html',
        name=request.form['name'],
        address=request.form['address'])

if __name__ == '__main__':
    app.run()
```

Part 8: Handling File and Image Uploads

Recall that an <input> tag can have its type attribute set to "file". This allows the user to upload files for submission with the form. However, for file uploads to work properly, the enclosing <form> must also be configured to use POST and include an additional enctype attribute that is set to "multipart/form-data". This attribute means that one or more sets of data are combined in a single body.

For example, the following template presents a form for uploading photos as well as a link for seeing all the photos that have been uploaded:

Template 11: templates/form_with_file_upload.html	

1	<!DOCTYPE html>
2	<html>
3	<head><title>Photo Upload</title></head>
4	<body>
5	<form method="post" enctype="multipart/form-data">
6	Photo: <input name="photo" type="file">
7	<input type="submit">

Chapter 18 Part 4: Web Applications (Flask)

8	</form>
9	View photos
10	</body>
11	</html>

The template for viewing the uploaded photos is provided below:

Template 12: templates/view_file_uploads.html

1	<!DOCTYPE html>
2	<html>
3	<head><title>View Photos</title></head>
4	<body>
5	{% for photo in photos %}
6	
8	{% endfor %}
9	Home
10	</body>
11	</html>

For the main Flask application, we create three routes: one for uploading photos, one for seeing all the uploaded photos and one for retrieving the image data of an uploaded photo given its filename. We also initialize and use a simple SQLite database to keep track of the photos uploaded:

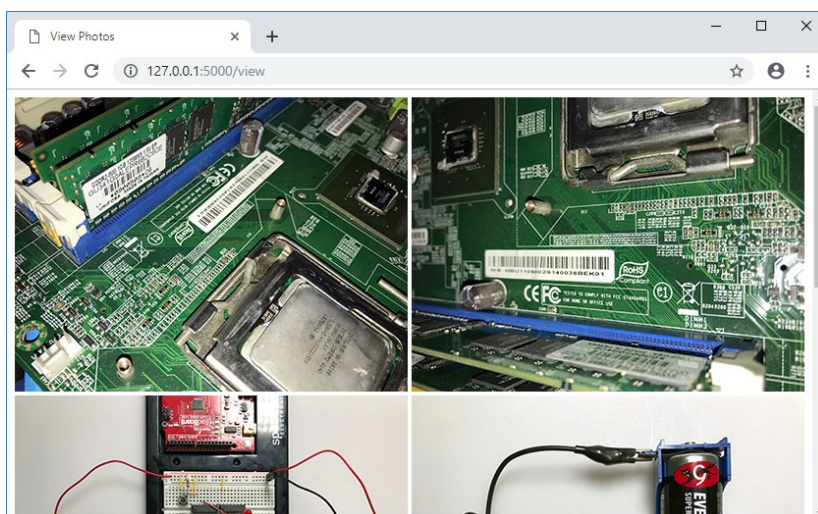
Program 21: p21_file_uploads.py

1	import flask, os, sqlite3
2	from flask import render_template, request
3	from flask import send_from_directory
4	from werkzeug.utils import secure_filename
5	
6	if not os.path.isfile('db.sqlite3'):
7	db = sqlite3.connect('db.sqlite3')
8	db.execute('CREATE TABLE photos(photo TEXT)')
9	db.commit()
10	db.close()
11	
12	app = flask.Flask(__name__)
13	
14	@app.route('/', methods=['GET', 'POST'])
15	def home():
16	if request.method == 'POST' and \
17	request.files and 'photo' in request.files:
18	# Save file
19	photo = request.files['photo']

Chapter 18 Part 4: Web Applications (Flask)

```
20     filename = secure_filename(photo.filename)
21     path = os.path.join('uploads', filename)
22     photo.save(path)
23     # Add filename to database
24     db = sqlite3.connect('db.sqlite3')
25     db.execute('INSERT INTO photos(photo) VALUES(?)',
26               (filename,))
27     db.commit()
28     db.close()
29     return render_template('form_with_file_upload.html')
30
31 @app.route('/view')
32 def view():
33     db = sqlite3.connect('db.sqlite3')
34     cur = db.execute('SELECT photo FROM photos')
35     photos = []
36     for row in cur:
37         photos.append(row[0])
38     db.close()
39     return render_template('view_file_uploads.html',
40                           photos=photos)
41
42 @app.route('/photos/<filename>')
43 def get_file(filename):
44     return send_from_directory('uploads', filename)
45
46 if __name__ == '__main__':
47     app.run()
```

Before running the program, create an empty uploads subfolder to store the uploaded photos. Next, run the program and visit <http://127.0.0.1:5000/> to upload some photos (i.e., GIF, JPG or PNG files). When you are done, click on the "View" link to see the uploaded photos.



Chapter 18 Part 4: Web Applications (Flask)

Go further! Try to upload non-photo files such as PDF files. The file will still be uploaded but will not be displayed properly. Can you edit Program 21 to only allow files with extensions GIF, JPG or PNG to be uploaded?

Note that unlike normal form data, submitted files are not accessed from `request.form` but from a separate `request.files` dictionary (e.g., line 19). Each value in this dictionary is a file object with a `filename` attribute as well as a `save()` method that accepts a file path and writes the submitted file onto the server's file system using the provided file path.

In general, we should be careful whenever we let users specify filenames for reading or writing files on the server's file system as file paths can use special folder names such as `..` to access parent folders that contain source code or your server's configuration files. To prevent this from happening, on line 20 we pass the filename through the `secure_filename()` function provided in the `werkzeug.utils` module first. This function returns a modified filename with all special characters replaced so it can be safely treated like a normal filename. We then use this filename on line 21 to form a file path that is guaranteed to be in our uploads subfolder.

To actually view uploaded photos, we need two routes: one for generating the HTML document that will display all the photos on a single page and another for accessing each photo's file data. For the second route, we use `send_from_directory()` on line 44 to avoid the same security issues that come from using paths or filenames provided by users. To use `send_from_directory()`, we call it with the name of a subfolder that the requested file must be stored in as well as the file's filename. We then return the result of this call directly.

As an alternative to configuring a new route to access each uploaded photo, we can instead save our uploads in the `static` subfolder and make use of the existing `'static'` route that Flask provides by default. However, this requires us to be careful not to let users overwrite files that they are not supposed to.

Below is a summary of how `request.args`, `request.form` and `request.files` are used to access the different kinds of data submitted by the user:

Attribute	<code>request.args</code>	<code>request.form</code>	<code>request.files</code>
Contents	Dictionary of field names and their associated values from query portion of URL	Dictionary of field names and their associated values	Dictionary of file upload names and their associated <code>FileStorage</code> objects
HTTP Method	Usually GET, but also works with POST if URL has query portion	POST only	POST only
Typical Use	Reading form data submitted using GET	Reading form data submitted using POST	Saving files submitted using POST

Chapter 18 Part 4: Web Applications (Flask)

Other			Form must specify enctype= "multipart/form- data"

Part 9: Conclusion

In this practical task, you have learned how to use Flask to route HTTP requests, send HTTP responses, use templates and process form data. Combined with the ability to read and write from SQLite databases, you now have the building blocks to create all manners of creative web applications.

As a conclusion, look back at `p01_server_without_flask.py` from the beginning of this practical task. The following program demonstrates how the same application that displays random colours and text can be written using the Flask framework that you are now familiar with:

Program 22: p22_server_with_flask.py

```
1  import random
2  import flask
3
4  app = flask.Flask(__name__)
5
6  # List of possible colours.
7  COLOURS = [
8      'red', 'orange', 'blue', 'purple',
9      '#C0C000', # dark yellow
10     '#00C000'  # dark green
11 ]
12
13 # List of possible messages.
14 MESSAGES = ['Hello, World!', 'Computing is Fun', 'Interesting']
15
16 @app.route('/css')
17 def css():
18     border_colour = random.choice(COLOURS)
19     text_colour = random.choice(COLOURS)
20     return (
21         flask.render_template('example.css',
22                               border_colour=border_colour,
23                               text_colour=text_colour),
24         { 'Content-Type': 'text/css' }
25     )
26
```

Chapter 18 Part 4: Web Applications (Flask)

```
27 @app.route('/')
28 def html():
29     msg = random.choice(MESSAGES)
30     return flask.render_template('example.html', msg=msg)
31
32 if __name__ == '__main__':
33     app.run()
```

This program also requires the following templates:

Template 13: templates/example.html

```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5     <title>{{ msg }}</title>
6     <link rel="stylesheet" href="{{ url_for('css') }}">
7 </head>
8
9 <body>
10     <p>{{ msg }}</p>
11 </body>
12
13 </html>
```

Template 14: templates/example.css

```
1 p {
2     border: 5px solid {{ border_colour }};
3     color: {{ text_colour }};
4     font-size: 72px;
5     padding: 20px;
6 }
```

Compare `p01_server_without_flask.py` to `p22_server_with_flask.py`. Which do you think is easier to write and maintain? Why do you think so?

While there are many other alternative approaches to developing web applications, we hope that learning Flask provides you with a firm foundation for the underlying principles and makes further exploration of the topic easier in the future.

Chapter 18 Part 4: Web Applications (Flask)

flask Module Summary

Name	Description
<code>Flask(name)</code>	Creates a Flask application object
<code>redirect(path)</code>	Redirects user to the given path when used as a return value from a decorated function
<code>render_template(filename, var=value)</code>	Renders Jinja2 template with the given filename using the given variable values and returns the rendered response as a str
<code>request</code>	Accesses current Request object
<code>send_from_directory(folder, filename)</code>	Sends file from given directory (first argument) with given filename (second argument) when used as a return value from a decorated function
<code>url_for(name, var=value)</code>	Returns the path that is mapped to the given function name and given variable values

werkzeug Module Summary

Name	Description
<code>secure_filename(filename)</code>	Replaces all characters that have special meanings (e.g., path separators) in the given str with underscores

Flask Class Summary

Name	Description
<code>route(path, methods=["GET"])</code>	Maps the given path to the decorated function but limits access to given list of HTTP methods; the HTTP methods are "GET" and "POST"
<code>run()</code>	Runs the Flask application with debugging disabled
<code>run(debug=True)</code>	Runs the Flask application with debugging enabled

Request Class Summary

Name	Description
<code>args</code>	Returns a dictionary of field names and their associated values from query portion of URL
<code>files</code>	Returns a dictionary of file upload names and their associated <code>FileStorage</code> objects
<code>form</code>	Returns a dictionary of field names and their associated values
<code>method</code>	Returns either "GET" or "POST"

FileStorage Class Summary

Name	Description
<code>filename</code>	Returns the name of the uploaded file
<code>save(path)</code>	Saves the uploaded file to the given path

Chapter 19: Data Management

Syllabus Learning Outcomes	1
1. What is data?	1
2. Data privacy	2
2.1 Understanding the difference between data privacy and data security	4
3. Data Integrity	4
3.1 Understanding the difference between data integrity and data security	4
4. Threats to data stored in a computer system	4
4.1 Lapses in users' behaviour	4
4.2 Mismanagement by multiple users	5
4.3 Natural disasters	5
4.4 Unauthorised intrusion into the system	5
5. Methods to protect data	5
5.1 Data backup	5
5.2 Data archive	6
5.2.1 Differences between backup and archive	6
5.2 Restricting access	7
5.3 Version Control	7
5.4 File Naming Convention (FNC)	7
5.5 Validation and Verification	8
5.6 Disaster recovery	8
5.7 Data encryption	9
5.8 Data Erasure	9
6. What is PDPA?	9
6.1 Do Not Call Registry	10
6.2 Use of NRIC	10
6.3 Data Obligations	1
6.4 Protecting your Personal Data	1
References	1

Syllabus Learning Outcomes

3.3 Databases and Data Management

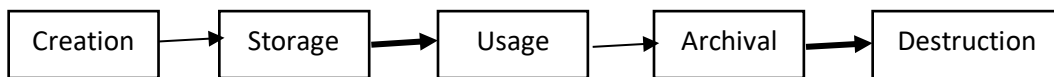
- 3.3.9 Understand the need for privacy and integrity of data.
- 3.3.10 Describe methods to protect data.
- 3.3.11 Explain the difference between backup and archive.
- 3.3.12 Describe the need for version control and naming convention.
- 3.3.13 Explain how data in Singapore is protected under the Personal Data Protection Act to govern the collection, use and disclosure of personal data.

1. What is data?

In computer science, data is information coded and structured for subsequent processing. It is any sequence of one or more symbols.

Data requires interpretation to become information. In computers, data is stored digitally, i.e. data is represented using the binary number system of ones and zeros, instead of analog representation. This allows us to use the computer to process the data.

The data life cycle refers to the entire period of time that data exists in the system. This life cycle encompasses all the stages that the data goes through.



2. Data privacy

Data privacy is the requirement for data to be accessed by or disclosed to authorised persons only. In other words, it is about keeping data private, rather than allowing it to be available in the public domain. This applies to both individuals and organisations.

It is important that unauthorised people do not have access to data they are not supposed to have. Unfortunately, in today's digitised society, it is easier than ever to gather a person's data and use it to obtain valuable insights, track his/her movements, or commit fraud.

For instance, data on which websites you visit often can reveal which products you are more likely to purchase as a shopper. This information can be highly valuable to an advertiser or a company.

As more services become available online, the risk of fraudulent use of data increases. For instance, with a photo of your identity card, a person can impersonate you and register for a new phone line on a telco website. Previously, such a transaction would have required the person to personally register over the counter with a physical identity card.

As technology becomes increasingly powerful, machines can gather information on a person easily, like performing facial recognition on surveillance videos to track down the whereabouts of an individual in a particular area.

Organisations also not only have to restrict access to data that is private to them, such as their management meeting minutes, but they are also responsible under privacy protection legislation to protect the data that they collect from individuals.

In Singapore, personal data is protected under the Personal Data Protection Act (PDPA). Personal data refers to data, whether true or not, about an individual who can be identified from that data; or from that data and other information to which the organisation has or is likely to have access. We want our personal data to be private so that unauthorised people cannot use our personal data for unauthorised use.

Video that discusses data privacy in Singapore context:

<https://www.channelnewsasia.com/news/video-on-demand/why-it-matters/losing-yourself-11075194>

2.1 Understanding the difference between data privacy and data security

Data security protects data from malicious threats. On the other hand, data privacy addresses responsible governance or use of that data. It is about ensuring only those who are authorised to access the data can do so. As such, when developing data security policies, the focus of protection measures is on preventing unauthorised access to data. If data is encrypted, that data is private, but it isn't necessarily secure.

3. Data Integrity

In order for computers to function properly, the data fed to them needs to be correct. As such, we must ensure that our data has not been altered in an unauthorised manner.

Data integrity is a concept and process that ensures the accuracy, completeness, consistency, and validity of an organisation's data. Data integrity covers data in storage, during processing, and while in transit.

Data integrity can be compromised in several ways. Each time data is replicated or transferred, it should remain intact and unaltered between updates. Error-checking methods and validation procedures are typically relied on to ensure the integrity of data that is transferred or reproduced without the intention of alteration.

3.1 Understanding the difference between data integrity and data security

Data security is the protection of data from unauthorised users who may have an interest in the data and an intention in corrupting it. On the other hand, data integrity means that the data is both correct and consistent. For example, to achieve this outcome, the data stored in the database must satisfy certain types of constraints or rules.

In other words, data integrity refers only to the validity and accuracy of data rather than the act of protecting data. So, data security is necessary to ensure data integrity and at the same time, data integrity is a desired result of data security.

4. Threats to data stored in a computer system

4.1 Lapses in users' behaviour

Users may not be careful when dealing with data. This includes making errors in data entry, using weak passwords, or revealing password/OTP to others.

Case study:

- <https://www.straitstimes.com/singapore/courts-crime/2278m-lost-to-top-10-scams-in-first-half-of-2022-as-overall-crime-rises-by-36>

4.2 Mismanagement by multiple users

Multiple users working on the same file may accidentally overwrite each other's data, causing data to be inaccurate or invalid.

4.3 Natural disasters

Natural disasters can destroy the computer system and/or storage device physically.

4.4 Unauthorised intrusion into the system

If the system is vulnerable to hackers, data may be corrupted or lost. Other cyberattacks such as virus, worms and trojan are equally damaging to the system and its data.

Case studies:

- <https://www.straitstimes.com/singapore/personal-info-of-15m-singhealth-patients-including-pm-lee-stolen-in-singapores-most>
- https://en.wikipedia.org/wiki/WannaCry_ransomware_attack
- <https://www.wired.com/story/confessions-marcus-hutchins-hacker-who-saved-the-internet/>

Videos to extend your knowledge on why we need to protect data.

- Secret Ways A Hacker Will Steal Your Identity:
<https://www.youtube.com/watch?v=9SVwPZ5scEU>
- How ads follow you around the internet:
<https://www.youtube.com/watch?v=HFyaW50GFOs>

5. Methods to protect data

5.1 Data backup

Modern storage equipment usually provides built-in disk clustering and redundancy in the event data in the primary storage is lost or corrupted. Users can also duplicate data or do a data backup.

Data backup involves creating copies of data and storing them separately. It is intended to be used as a safety precaution or prevention against unexpected loss of data due to unintended data corruption or errors such as disk drive failing, files accidentally being deleted, or a data center going offline during a catastrophic event.

A backup is typically a short-term solution and may be performed on a regular basis (e.g., daily, weekly) to ensure that data is up to date and readily available in case of a data loss event.

5.2 Data archive

Archiving data involves storing data that is not actively used but kept for historical references or auditing purposes. It ensures that important records remain available years after they were created. In most cases, the purpose of data archiving is to meet legal and compliance requirements. For example, a doctor's office might be required to keep patient records for a certain period of time. Similarly, a bank may need to retain transaction records.

5.2.1 Differences between backup and archive

As far as the fundamental difference between backups and archives is concerned, backup helps with data recovery in the event of recent data loss or corruption or hardware failure. Archives, on the other hand, serve the dual purposes of long-term retention and space management.

Comparison Chart

	Backup	Archive
Data Storage Method	The original data remains in place, while a backup copy is stored in another location	Archived data is moved from its original location to an archive storage location
Data State	Backed-up data is constantly changing	Once you create an archive, you do not modify it

Data Retention Policy	You periodically delete or overwrite data backups that are too old to be useful	Data archives are designed for long-term storage
Storage Type	Hot cloud storage or easily accessible local storage locations	Cold cloud storage or tape archives
Data Scope	All of your data, with the exception of unimportant information like temporary files	Specific files that you must retain for compliance purposes

Note: Hot storage refers to fast, easy-to-access data storage like your local hard drive or a quick-access cloud storage provider like Google Drive. In contrast, cold storage is for archival data that is rarely accessed and usually stored off-site.

5.2 Restricting access

We can also ensure that only authorised users can access our data. This includes using:

- Passwords
 - It is important that users follow general guidelines to ensure the password is strong. Certain organisations may have more stringent rules.
- Two-factor Authentication
 - This can be in the form of security tokens or One-Time-Password (OTP)
- Biometrics
 - This can be in the form of facial recognition and fingerprint scanning.

Such authentication and authorisation controls help us verify credentials and assure that user privileges are applied correctly. These measures are typically used as part of an identity and access management (IAM) solution and in combination with role-based access controls (RBAC).

5.3 Version Control

Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.

Git is a commonly used version tool. It allows users to rollback to previous working versions of processes to fix mistakes while minimising disruption to other team members. Git has three main states that your files can reside in: modified, staged, and committed:

- Modified means that you have changed the file but have not committed it to your database yet.

- Staged means that you have marked a modified file in its current version to go into your next commit snapshot.
- Committed means that the data is safely stored in your local database.

More details can be found here:

<https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control>

5.4 File Naming Convention (FNC)

File naming convention is essentially a framework for naming files in a way that describes what they contain and how they relate to other files. This helps to minimise the chances of files being misplaced or lost unintentionally due to poor organisation of files. Such a convention also enables users to locate files quickly.

Developing an FNC is done by identifying the key elements of the project, and the important differences and commonalities between your files.

File naming best practices:

- Files should be named consistently
- File names should be short but descriptive (<25 characters) (Briney, 2015)
- Avoid special characters or spaces in a file name
- Use capitals and underscores instead of periods or spaces or slashes
- Use date format ISO 8601: YYYYMMDD
- Include a version number (Creamer et al. 2014)
- Write down naming convention in data management plan

Elements to consider using in a naming convention are:

- Date of creation (putting the date in the front will facilitate computer -aided date sorting)
- Short Description
- Work
- Location
- Project name or number
- Sample
- Analysis
- Version number

Examples of files following an FNC:

- 20160104_ProjectA_Ex1Test1_SmithE_v1.xlsx
- 20160104_ProjectA_MeetingNotes_SmithE_v2.docx

5.5 Validation and Verification

We have learnt how to implement data validation and verification earlier in the Data Validation and Verification lesson. Essentially, data validation and verification are measures to check data when it originally enters the system (i.e. 1st stage of the data life cycle) or when it is transmitted from one system (or process) to another. (subsequent stages)

5.6 Disaster recovery

Disaster recovery is a set of practices and technologies that determine how an organisation deals with a disaster, such as a cyberattack, natural disaster, or large-scale equipment failure. The disaster recovery process typically involves setting up a remote disaster recovery site with copies of protected systems and switching operations to those systems in case of disaster.

5.7 Data encryption

Data encryption alters data content according to an algorithm that can only be reversed with the right encryption key. Encryption protects your data from unauthorised access even if data is stolen by making it unreadable.

5.8 Data Erasure

Data erasure is more secure than standard data wiping because it uses software to completely overwrite data on any storage device. It verifies that the data is unrecoverable. limits liability by deleting data that is no longer needed. This can be done after data is processed and analysed or periodically when data is no longer relevant. Erasing unnecessary data is a requirement of many compliance regulations.

6. What is PDPA?

The PDPA is a data protection law comprising various rules that govern the collection, use, disclosure, and care of personal data. It recognises both the rights of individuals to protect their personal data, including rights of access and correction, as well as the needs of organisations to collect, use or disclose personal data for legitimate and reasonable purposes.

It takes into account the following:

- **Consent:** Organisations must obtain an individual's knowledge and consent to collect, use or disclose his/her personal data (with some exceptions).
- **Notification :** Organisations must inform individuals of the purposes for collecting, using, or disclosing their personal data.

- **Appropriateness:** Organisations may collect, use or disclose personal data only for purposes that would be considered appropriate to a reasonable person under the given circumstances.
- **Accountability :** Organisations must make information about their personal data protection policies available on request. They should also make available the business contact information of the representatives responsible for answering questions relating to the organisations' collection, use, or disclosure of personal data.

To administer and enforce the PDPA, Singapore set up the Personal Data Protection Commission (PDPC) in 2013.

6.1 Do Not Call Registry

Have you received calls from unknown companies who seem to know your name and perhaps try to sell products to you? Your telephone number could have been gathered from unexpected sources, such as a lucky draw form that you filled up long ago. With technology, companies can easily gather and consolidate personal information. In fact, it can even automate the making of such calls.

To prevent you from getting unnecessary marketing calls, you can register in the Do Not Call (DNC) Registry to opt out of marketing messages and calls. The PDPA prohibits organisations from sending marketing messages to Singapore telephone numbers, including mobile, fixed-line, residential, and business numbers that are registered with the DNC Registry.

There are three DNC registers that individuals can choose to register in:

- No Voice Call Register
- No Text Message Register
- No Fax Message Register

Registering the phone number in each register is to opt out of receiving marketing messages through voice calls, text messages and fax messages.

Note that organisations which have an ongoing relationship with a subscriber or user of a Singapore telephone number may send marketing messages on similar or related products, services, and memberships to that Singapore telephone number via text or fax without checking against the DNC Registry. However, each exempt message must also contain an opt-out facility that the recipient may use to opt-out from receiving such telemarketing messages. If a recipient opts out, organisations must stop sending such messages to his/her Singapore telephone number after 30 days. This means that, for instance, you can get a marketing message from your mobile phone company even if you

listed the number in the DNC Registry. However, the message will include a link for you to opt -out of receiving their marketing messages. Once you opt -out, the mobile phone company can no longer send you marketing messages on that number.

6.2 Use of NRIC

The Singapore National Registration Identification Card (NRIC) number is a unique identifier assigned to Singapore citizens and permanent residents. Similarly, the Foreign Identification Number (FIN) is a unique identifier that is assigned to foreigners. The NRIC/FIN contains personal information about the person, such as his/her date of birth and address. As unique identifiers like NRIC and FIN are permanent, irreplaceable, and used in a variety of government transactions, we need to be careful with such data.

Individuals should not readily provide their NRIC/FIN and personal particulars to companies/strangers. Consent is required before organisations can obtain a person's data. Under the PDPA, from 1 September 2019, organisations are generally not allowed to collect, use or disclose NRIC numbers (or copies of NRIC) except in the following circumstances:

- Collection, use or disclosure of NRIC numbers (or copies of NRIC) is required under the law (or an exception under the PDPA applies); or
- Collection, use or disclosure of NRIC numbers (or copies of NRIC) is necessary to accurately establish or verify the identities of the individuals to a high degree of fidelity.

For example, a medical clinic needs to see the NRIC of a patient to identify the person. The clinic will need to keep the name, address, NRIC, and contact number of the person with the medical notes for future reference. The PDPA allows for that. However, a shopping mall cannot collect the photographs of NRICs of all the shoppers that want to participate in their lucky draw. It is unnecessary to collect the photographs to verify the lucky draw participant. Instead, the participants can be identified with their mobile number, or be asked to give the last 4 characters of the NRIC (i.e. partial NRIC) for verification purposes. This reduces the security risks if the data collected is unintentionally revealed.

Note that PDPA does not apply to public agencies and organisations acting on behalf of them, thus, for example, the police can collect your personal information, including NRIC/FIN. Data collected by public agencies are protected by other acts. See <https://www.mci.gov.sg/pressroom/news-and-stories/pressroom/2019/2/mcis-response-to-pq-on-public-agencies-exemption-from-pdpa> for more information.

6.3 Data Obligations

Organisations are required to abide by the following 9 main personal data obligations:

1. Consent Obligation

Only collect, use or disclose personal data for purposes for which an individual has given his or her consent.

2. Purpose Limitation Obligation

An organisation may collect, use or disclose personal data about an individual for the purposes that a reasonable person would consider appropriate in the circumstances and for which the individual has given consent.

3. Notification Obligation

Notify individuals of the purposes for which your organisation is intending to collect, use or disclose their personal data on or before such collection, use, or disclosure of personal data.

4. Access and Correction Obligation

Upon request, the personal data of an individual and information about the ways in which his or her personal data has been or may have been used or disclosed within a year before the request should be provided. However, organisations are prohibited from providing individual access if the provision of personal data or other information could reasonably be expected to cause harmful effects.

Organisations are also required to correct any error or omission in an individual's personal data that is raised by the individual.

5. Accuracy Obligation

Make a reasonable effort to ensure that personal data collected by or on behalf of your organisation is accurate and complete, if it is likely to be used to make a decision that affects the individual, or if it is likely to be disclosed to another organisation.

6. Protection Obligation

Make reasonable security arrangements to protect the personal data that your organisation possesses or controls to prevent unauthorised access, collection, use, disclosure, or similar risks.

7. Retention Limitation Obligation

Cease retention of personal data or remove the means by which the personal data can be associated with particular individuals when it is no longer necessary for any business or legal purpose.

8. Transfer Limitation Obligation

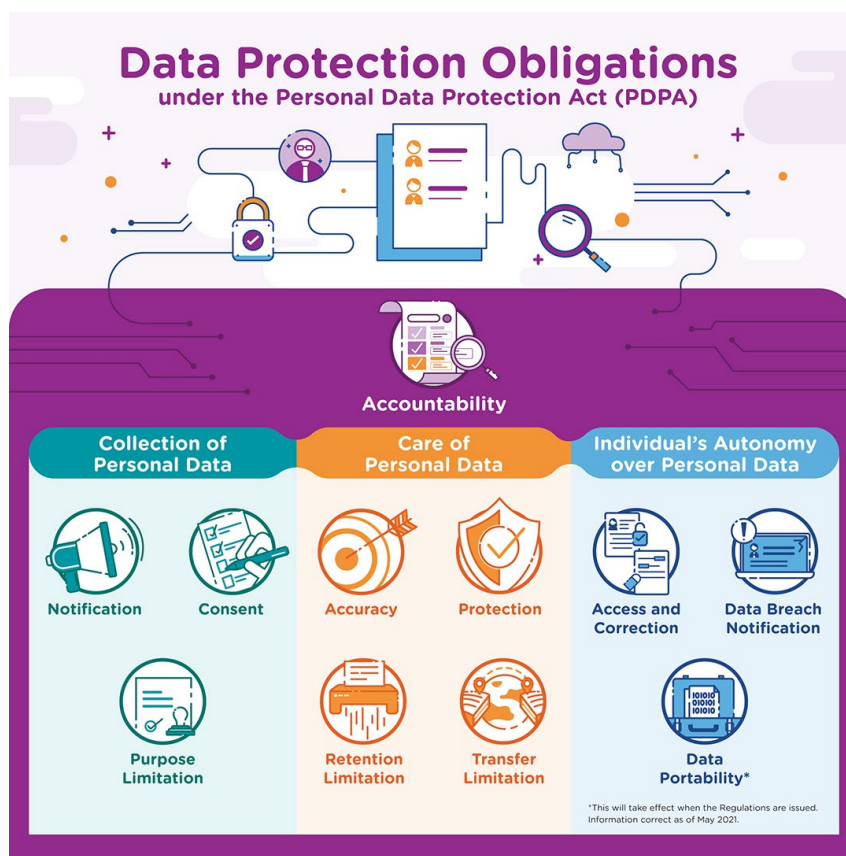
Transfer personal data to another country only according to the requirements prescribed under the regulations, to ensure that the standard of protection provided to the personal data so transferred will be comparable to the protection under the PDPA, unless exempted by the PDPC.

9. Accountability Obligation

Make information about your data protection policies, practices and complaints process available on request. Designate a Data Protection Officer to ensure that your organisation complies with the PDPA.

More information on PDPA is available at the PDPC website:

<http://www.pdpc.gov.sg/>



6.4 Protecting your Personal Data

You can take various measures to protect your personal data. Don't reveal your personal data to unknown sources. For phone calls, ensure that the caller is who he or she is before giving your personal information. For websites and applications, read the privacy or data protection policies of the website to understand how your data is used.

Websites/applications require you to explicitly agree with the terms when you submit the data. An example is shown below.

☐ I have agree with the **Terms and Conditions**.

☐ I agree to the collection, use, disclosure or otherwise processing of my personal data in accordance with the **Privacy Policy**.

Submit

Please note that once you agreed, the company can contact you, even if your number is listed in the Do Not Call Registry. If you have queries on personal data, or to withdraw consent, you can contact the data protection officer (DPO). Under PDPA, companies are required to appoint one or more persons to be DPO to oversee the data protection responsibilities within the organisation and ensure compliance with the PDPA.

Also, be careful when throwing away paper containing your personal data such as application forms or letters from schools/banks. Tear or shred the paper so that people cannot use the paper to obtain personal data about yourself.

References

- Burdett, A. (2016). BCS glossary of computing. BCS Learning and Development Limited.
- LANGFIELD, S. D. (2015). Cambridge International AS and A Level Computer Science Coursebook. Place of publication not identified: Cambridge University Press.
- Watson, D., & Williams, H. (2019, May 31). Cambridge International AS & A Level Computer Science. London: Hodder Education
- Personal Data Protection Commission Singapore. Retrieved from <https://www.pdpc.gov.sg/Overview-of-PDPA/The-Legislation/Personal-Data-Protection-Act>
- Personal Data Protection Commission Singapore. (2019, September 26). Guide to Notification. Retrieved from <https://www.pdpc.gov.sg/-/media/Files/PDPC/PDF-Files/Other-Guides/Guide-to-Notification-260919.pdf>.
- [DPO Challenge Game \(pdpc.gov.sg\)](https://www.pdpc.gov.sg/DPO-Challenge-Game)
- <https://www.fortinet.com/resources/cyberglossary/data-integrity>
- <https://cloudian.com/guides/data-protection/data-protection-and-privacy-7-ways-to-protect-user-data/#protection-v-privacy>
- <https://www.msp360.com/resources/blog/backup-vs-archive/#:~:text=As%20far%20as%20the%20fundamental,term%20retention%20and%20space%20management.>
- <https://libguides.princeton.edu/c.php?g=102546&p=930626>
- MOE CPDD PDPA Notes

Chapter 20: Ethics and Impact of Computing

Syllabus Learning Outcomes	1
1. Ethics	2
2. The Professionalism of Computing Professionals	2
3. Legal Issues	3
3.1 Personal Data Protection Act	3
3.2 Copyright Act	3
3.3 Computer Misuse Act	3
3.4 Cybersecurity Act	3
4. Social Issues	3
5. Economic Issues	4
6. AI Ethics & Governance	4
7. Conclusion	5
References	6

Syllabus Learning Outcomes

- 3.4 Social, Ethical, Legal and Economic Issues
 - 3.4.1 Understand the code of ethics (conduct) of a Computing professional.
 - 3.4.2 Describe the impact of computing on lifestyle and workplace for social and economic developments.
 - 3.4.3 Discuss the social, ethical, legal and economic issues of computing and technology.

1. Ethics

Ethics is a set of rules of behaviour based on ideas of what is morally good and bad and what is morally right and wrong. Every society has its rules about whether certain issues are ethical or not. As the use of computers becomes more widespread in every aspect of our lives, discussions in computer ethics have reached a somewhat common consensus.

The consensus can be represented in the Ten Commandments of Computing Ethics:

- 1) Thou shalt not use a computer to harm other people.
- 2) Thou shalt not interfere with other people's computer work
- 3) Thou shalt not snoop around in other people's files
- 4) Thou shalt not use a computer to steal
- 5) Thou shalt not use a computer to bear false witness
- 6) Thou shalt not use or copy software for which you have not paid
- 7) Thou shalt not use other people's computer resources without authorization
- 8) Thou shalt not appropriate other people's intellectual output
- 9) Thou shalt think about the social consequences of the program you write
- 10) Thou shalt use a computer in ways that show consideration and respect

For a more detailed explanation of each of the ten commandments, please visit:

https://en.wikipedia.org/wiki/Ten_Commandments_of_Computer_Ethics ,
https://www.youtube.com/watch?v=ASm_-2t31Ptg

2. The Professionalism of Computing Professionals

Computing professionals includes hardware designers, software engineers, database administrators, system analysts, and computer scientists. Today, with more adaptations of technology into our daily life, the scope of computing professionals also increase. No matter what their particular specialisation is, any professional person is expected to act ethically.

In Singapore, the Singapore Computing Society (SCS) has a code of conduct that gives guidance under four headings:

- Integrity
- Full Responsibility
- Competence
- Professionalism

For more details, refer to <https://www.scs.org.sg/membership/code-of-conduct>

3. Legal Issues

Legal issues are issues where a law has been passed by a government. While you are not expected to learn the details of all the Acts listed you may be asked to consider the issues in the legal context.

One thing to note is that as technology develops rapidly, laws often become outdated rapidly. Many acts are introduced/amended in response to current events. For example, the Electronic Transactions Act was recently amended in 2021. The new Copyright Act which was passed in Parliament in September 2021, came into force on 21 November 2021.

You may find the summary of regulations on the IMDA's website:

<https://www.imda.gov.sg/Regulations-and-Licences/Regulations>

3.1 Personal Data Protection Act

This has been discussed in Chapter 19 Data Management.

3.2 Copyright Act

More details can be found at

<https://www.ipos.gov.sg/news/press-releases/ViewDetails/commencement-of-the-copyright-act/>

3.3 Computer Misuse Act

More details can be found at

<https://sso.agc.gov.sg/Act/CMA1993>

3.4 Cybersecurity Act

More details can be found at

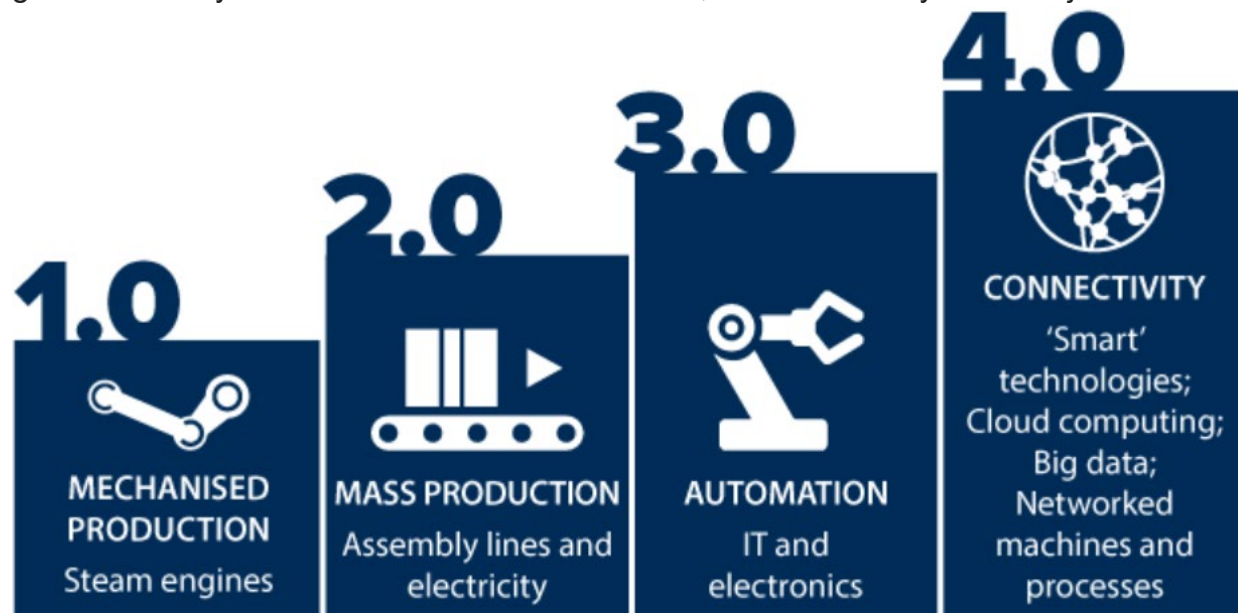
<https://www.csa.gov.sg/Legislation/Cybersecurity-Act>

4. Social Issues

Social issues are factors that influence the beliefs attitudes and actions of people within society. There are elements of computer use that have a cultural impact in that they can change our attitudes, beliefs and actions.

5. Economic Issues

modern industry has seen great advances since its earliest iteration at the beginning of the industrial revolution in the 18th century. This has changed the pace of the economic growth of many countries. On the individual level, it creates many different jobs.



What economic changes do you think Industry 4.0 will bring?

6. AI Ethics & Governance

The rapid advances in AI tools and technologies, and the increasing deployment and embedding of such tools in apps or solutions push the need for a Body of Knowledge (BoK).

It is, therefore, vital to building stakeholder confidence in AI. That can happen if solution providers and deploying organisations ensure the responsible use of AI to manage different risks in AI deployment.

Moreover, such organisations will also have to take measures to align internal policies, structures and processes with relevant accountability-based practices in data management and protection

The Body of Knowledge (BoK) is based on the Infocomm Media Development Authority's (IMDA) Model AI Governance Framework and it is tailored for practical issues related to human safety, fairness and the prevailing approaches to privacy, data governance and general ethical values.

Through this initiative, the AI Ethics and Governance Toolkit has been formulated. It provides an overview of AI governance issues and explores a risk-based approach to deploying AI. This Toolkit can be immensely useful for tech and non-tech professionals to understand the ethical issues surrounding AI.

For access to the Toolkit, please view

<https://files-scs-prod.s3-ap-southeast-1.amazonaws.com/public/uploads/200721016+SCS+Booklet+6+Nov.pdf>

For more information on BoK, please visit <https://www.scs.org.sg/ai-ethics-bok>

7. Conclusion

Ultimately, it is undeniable that advances in computing and technology have impacted our lives greatly and are here to stay for the foreseeable future. It brings along with its benefits as well as risks. What matters is that we must learn to embrace the limitless possibilities that it brings, and learn to manage the risks.

References

- <https://www.scs.org.sg/ai-ethics-bok>
- <https://www.imda.gov.sg/Regulations-and-Licences/Regulations>
- <https://www.ipos.gov.sg/news/press-releases/ViewDetails/commencement-of-the-copyright-act>
- <https://sso.agc.gov.sg/Act/CMA1993>
- <https://www.csa.gov.sg/Legislation/Cybersecurity-Act>
- <https://www.scs.org.sg/membership/code-of-conduct>
- <https://www.scs.org.sg/ai-ethics-bok>
- <https://www.acm.org/code-of-ethics/case-studies>

Chapter 21: Fundamentals of Computer Networks (Part 1)

Syllabus Learning Outcomes	1
1. Network Communication Protocols	2
1.1 OSI Model (Open Systems Interconnection Model)	3
• Physical Layer (Layer 1)	4
• Data Link Layer (DLL) (Layer 2)	4
• Network Layer (Layer 3)	5
• Transport Layer (Layer 4)	5
• Session Layer (Layer 5)	5
• Presentation Layer (Layer 6)	5
• Application Layer (Layer 7)	6
1.1.1 How data flows through the OSI model	6
1.2 Internet Protocol (TCP/IP Protocol Suite)	7
1.2.1 Purpose of the Internet Protocol	10
1.2.2 How Internet Protocol Works	11
1.3 Well-known Protocols	15
1.3.1 How HTTP Works (How the Server Handles Requests from Clients)	17
1.3.2 File Transfer Protocol (FTP)	19
1.3.3 Simple Mail Transfer Protocol (SMTP)	20
1.3.4 Post Office Protocol (POP) and Internet Message Access Protocol (IMAP)	20
2. Network Core	21
2.1 Packet-switching	23
2.1.1 Store-and-Forward Transmission	23
2.1.2 Queuing Delays and Packet Loss	24
2.1.2.1 What if the internet is "busy"	25
2.1.2.2 Can a packet get lost?	26
2.1.3 Forwarding Tables and Routing Protocols	26
2.1.3.1 How do forwarding tables get set?	26
2.2 Circuit-switching	27
2.3 Packet-switching vs Circuit-switching	28
Referenced Readings	28

Syllabus Learning Outcomes

4.1 Fundamentals of Computer Networks

Understand computer network technology

4.1.3 Explain the need for communication protocols in a network.

4.1.4 Explain how data is transmitted in a packet-switching network.

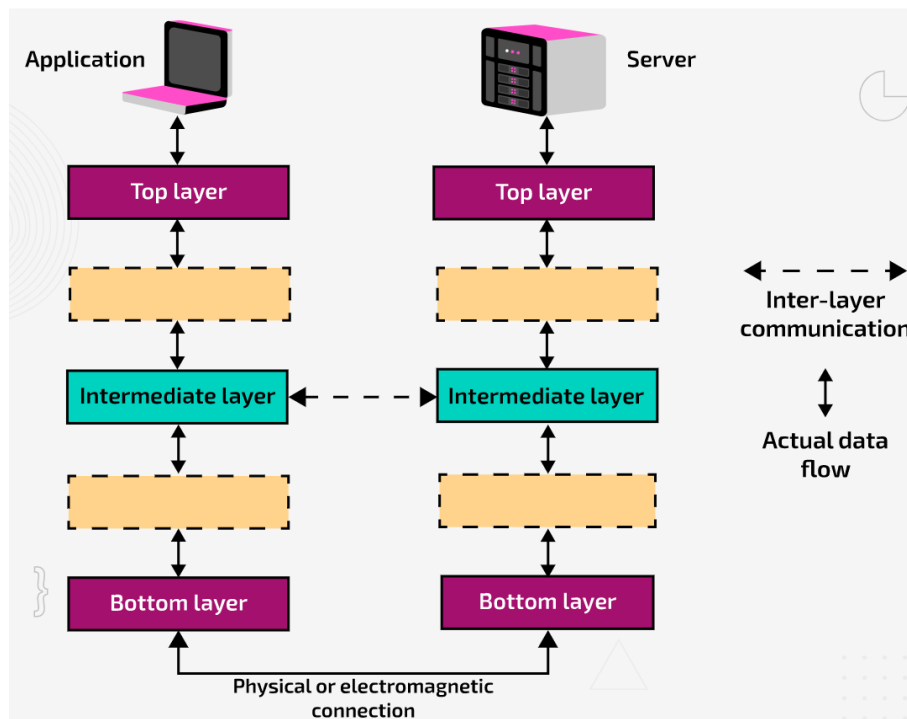
1. Network Communication Protocols

A protocol is a set of rules for data transmission which are agreed upon by both sender and receiver. To handle complexity in data communication, we can break the system down and organise its components into layers where each layer provides functionality that depends solely on the layer directly below it.

The protocols used to transmit data over the Internet can be organised into layers just like the protocols used to deliver parcels. The table below provides a comparison.

Postage Protocol	Internet Protocol
For delivering parcels	For transmitting packets
Address information attached using labels	Address information prepended using headers
Address is made up of unit numbers and postal codes	Address is made up of IP and MAC addresses

Study the following diagram that shows a generic protocol stack.



Each layer in a protocol stack is responsible for some part of the communications process and uses a protocol that is suitable for the needs of the data to be communicated.

The protocol in the layer at the sending end of the communications link communicates with the corresponding protocol in the corresponding layer at the receiving end of the link. The data is sent between these protocols, using the protocols in the layers below, as illustrated in Figure 1.

The protocol in the top layer receives its data from the software application at one end of the link. It processes the data and sends it to the same protocol in the corresponding layer at the receiving end of the link using the layers below. This protocol then passes the data to the receiving application, usually on a server.

The protocols in the intermediate layers receive the data from the layer above, process it, and pass it to the protocol in the layer below. At the other end of the link, the reverse happens. In this way, the protocols in the intermediate layers communicate with the corresponding protocols in the corresponding layer at the receiving end of the link.

The protocol in the bottom layer is responsible for converting the data into a suitable signal for transmission on the media. This could be an electrical signal for Ethernet, an electromagnetic signal for WiFi, or pulses of light for a fibre cable.

The specification for the interface between each layer is well documented. This means that software developers just need to make sure that the data is in the correct format (so that it can move from one layer to the next). The advantage of this is that each layer is independent of the others. For example, you can use the same web browser regardless of whether you connect to the internet using a local area network or through a mobile phone mast.

1.1 OSI Model (Open Systems Interconnection Model)

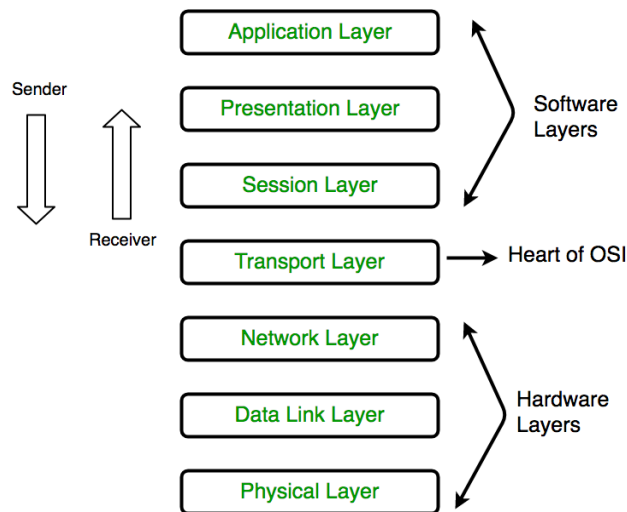
Refer to

<https://www.geeksforgeeks.org/layers-of-osi-model/>

<https://www.cloudflare.com/learning/ddos/glossary/open-systems-interconnection-model-osi/>

The Open Systems Interconnection (OSI) reference model has served as the most basic element of computer networking since the inception in 1984. The OSI Reference Model is based on a proposal developed by the International Standards Organization (ISO).

The OSI Model can be seen as a universal language for computer networking. It is based on the concept of splitting up a communication system into seven abstract layers, each one stacked upon the last. Each layer of the OSI Model handles a specific job and communicates with the layers above and below itself.



- **Physical Layer (Layer 1)**

The physical layer contains information in the form of bits. It is responsible for transmitting individual bits from one node to the next. When receiving data, this layer will get the signal received and convert it into 0s and 1s and send them to the Data Link layer, which will put the frame back together.

* Hub, Repeater, Modem, Cables are Physical Layer devices.

- **Data Link Layer (DLL) (Layer 2)**

The data link layer is responsible for the node-to-node delivery of the message. The main function of this layer is to make sure data transfer is error-free from one node to another, over the physical layer. When a packet arrives in a network, it is the responsibility of DLL to transmit it to the Host using its MAC address.

* Packet in Data Link layer is referred to as Frame.

** Data Link layer is handled by the NIC (Network Interface Card) and device drivers of host machines.

*** Switch & Bridge are Data Link Layer devices.

- **Network Layer (Layer 3)**

The network layer works for the transmission of data from one host to the other located in different networks. It also takes care of packet routing i.e. selection of the shortest path to transmit the packet, from the number of routes available. The sender & receiver's IP addresses are placed in the header by the network layer.

* Segment in Network layer is referred to as Packet.

**** Network layer is implemented by networking devices such as routers.**

- **Transport Layer (Layer 4)**

The transport layer provides services to the application layer and takes services from the network layer. It is responsible for the End to End Delivery of the complete message. It also provides the acknowledgment of the successful data transmission and re-transmits the data if an error is found.

*** Data in the Transport Layer is called as Segments.**

**** Transport layer is operated by the Operating System. It is a part of the OS and communicates with the Application Layer by making system calls.**

- **Session Layer (Layer 5)**

This layer is responsible for the establishment of connection, maintenance of sessions, authentication, and also ensures security.

****All the below 3 layers, including Session Layer are integrated as a single layer in the TCP/IP model as “Application Layer”.**

****Implementation of these 3 layers is done by the network application itself. These are also known as Upper Layers or Software Layers.**

- **Presentation Layer (Layer 6)**

The presentation layer is also called the Translation layer. The data from the application layer is extracted here and manipulated as per the required format to transmit over the network.

- **Application Layer (Layer 7)**

At the very top of the OSI Reference Model stack of layers, we find the Application layer which is implemented by the network applications. These applications produce the data, which has to be transferred over the network. This layer also serves as a window for the application services to access the network and for displaying the received information to the user.

***Application Layer is also called Desktop Layer.**

In summary, the OSI model breaks down communication over networks into seven different layers, each with a specific role and purpose in the process of transmitting data from one device to another.

OSI model acts as a reference model and is not implemented on the Internet. The current model being used is the TCP/IP model. Although the modern Internet does not

strictly follow the OSI Model (it more closely follows the simpler Internet protocol suite), the OSI Model is still very useful for troubleshooting network problems. It can help to break down the problem and isolate the source of the trouble. If the problem can be narrowed down to one specific layer of the model, a lot of unnecessary work can be avoided.

1.1.1 How data flows through the OSI model

For human-readable information to be transferred over a network from one device to another, the data must travel down the seven layers of the OSI Model on the sending device and then travel up the seven layers on the receiving end.

For example, Mr. Cooper wants to send Ms. Palmer an email. Mr. Cooper composes his message in an email application on his laptop and then hits 'send'. His email application will pass his email message over to the application layer, which will pick a protocol (SMTP) and pass the data along to the presentation layer. The presentation layer will then compress the data and then it will hit the session layer, which will initialize the communication session.

The data will then hit the sender's transport layer where it will be segmented, then those segments will be broken up into packets at the network layer, which will be broken down even further into frames at the data link layer. The data link layer will then deliver those frames to the physical layer, which will convert the data into a bitstream of 1s and 0s and send it through a physical medium, such as a cable.

Once Ms. Palmer's computer receives the bit stream through a physical medium (such as her wifi), the data will flow through the same series of layers on her device, but in the opposite order. First, the physical layer will convert the bitstream from 1s and 0s into frames that get passed to the data link layer. The data link layer will then reassemble the frames into packets for the network layer. The network layer will then make segments out of the packets for the transport layer, which will reassemble the segments into one piece of data.

The data will then flow into the receiver's session layer, which will pass the data along to the presentation layer and then end the communication session. The presentation layer will then remove the compression and pass the raw data up to the application layer. The application layer will then feed the human-readable data along to Ms.

Palmer's email software, which will allow her to read Mr. Cooper's email on her laptop screen.

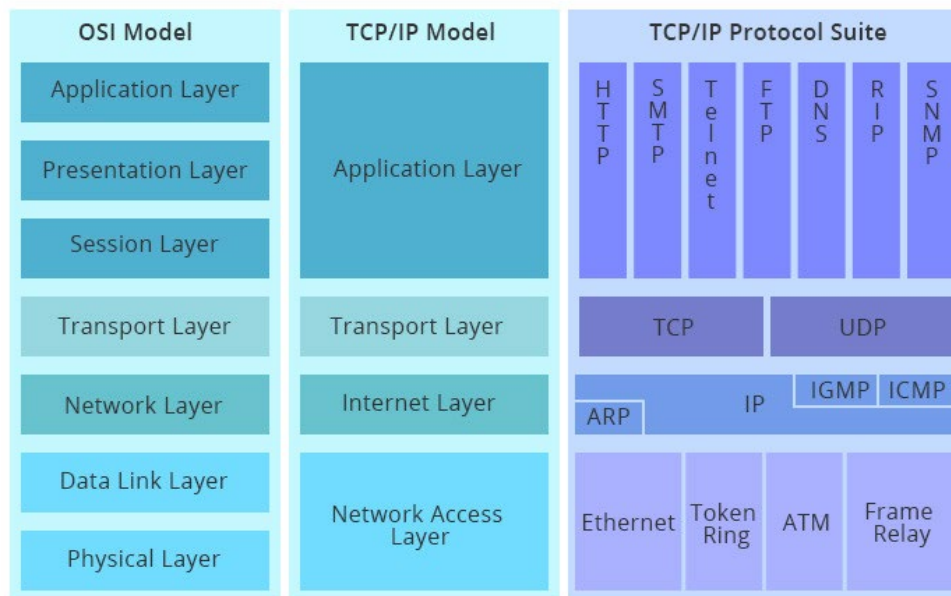
1.2 Internet Protocol (TCP/IP Protocol Suite)

The OSI Model we just looked at is just a reference/logical model. It was designed to describe the functions of the communication system by dividing the communication procedure into smaller and simpler components. TCP/IP was developed in the 1970s by the Department of Defense (DoD) and adopted as the protocol standard for ARPANET (the predecessor to the Internet) in 1983. It stands for Transmission Control Protocol/Internet Protocol. The TCP/IP model is a concise version of the OSI model.

The Internet protocols are organized into 5 layers:

- The Application layer, which provides high-level functionality to end-users
- The Transport layer, which provides functionality to transmit messages between any two programs
- The Internet layer, which provides functionality to determine a route between any two devices
- The Link layer, which provides functionality to transmit packets from one device to an adjacent device
- The Physical layer, which provides functionality to transmit individual bits through a transmission medium.

Note that in some literature, the Link and Physical Layers are known as the Network Access Layer.



Source: <https://community.fs.com/blog/tcpip-vs-osi-whats-the-difference-between-the-two-models.html>

TCP/IP	OSI
Implementation of OSI model	Reference model
Model around which Internet is developed	This is a theoretical model
Has only 4 layers	Has 7 layers
Considered more reliable	Considered a reference tool
Protocols are not strictly defined	Stricter boundaries for the protocols
Horizontal approach	Vertical approach
Combines the session and presentation layer in the application layer	Has separate session and presentation layer
Protocols were developed first and then the model was developed	Model was developed before the development of protocols
Supports only connectionless communication in the network layer	Supports connectionless and connection-oriented communication in the network layer
Protocol dependent standard	Protocol independent standard

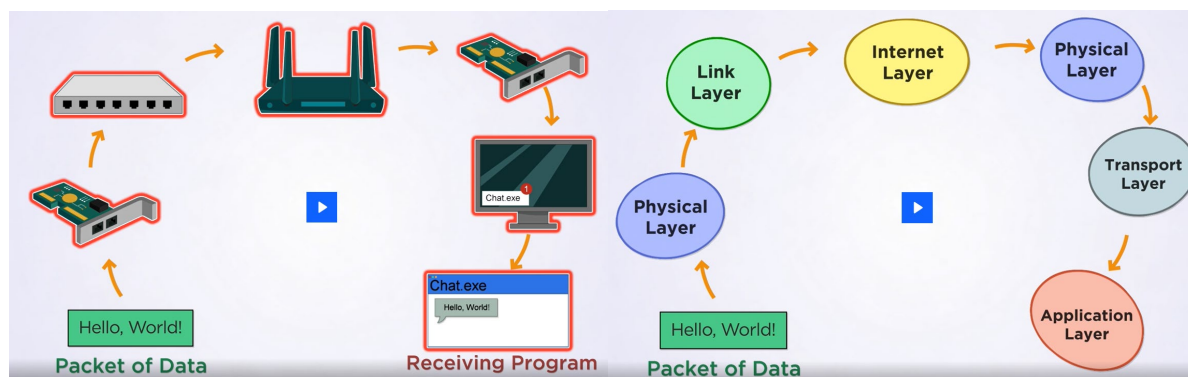
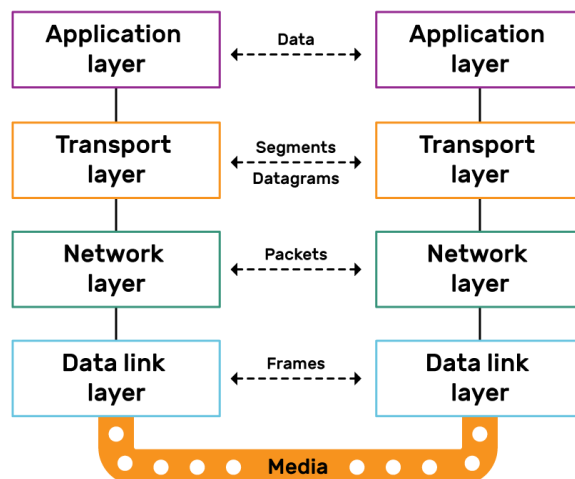
Figure 5: Differences between OSI and TCP/IP Model [28].

Source: Ravali, P(2015). A Comparative Evaluation of OSI and TCP / IP Models.

Note: The OSI model is often referred to as a vertical approach because it has a strict hierarchy of layers, with each layer building on the services provided by the layer below it. In contrast, the TCP/IP model is often called a horizontal approach because it

emphasizes the interaction and interconnection of different protocols and technologies at the same level, without necessarily following a strict hierarchy.

The diagram below shows how an application communicates directly with a server. We now know this does not happen directly. The application uses the services of the TCP/IP model to encapsulate its message in various layers of protocols. Ultimately, the data is transferred over the media to the internet.



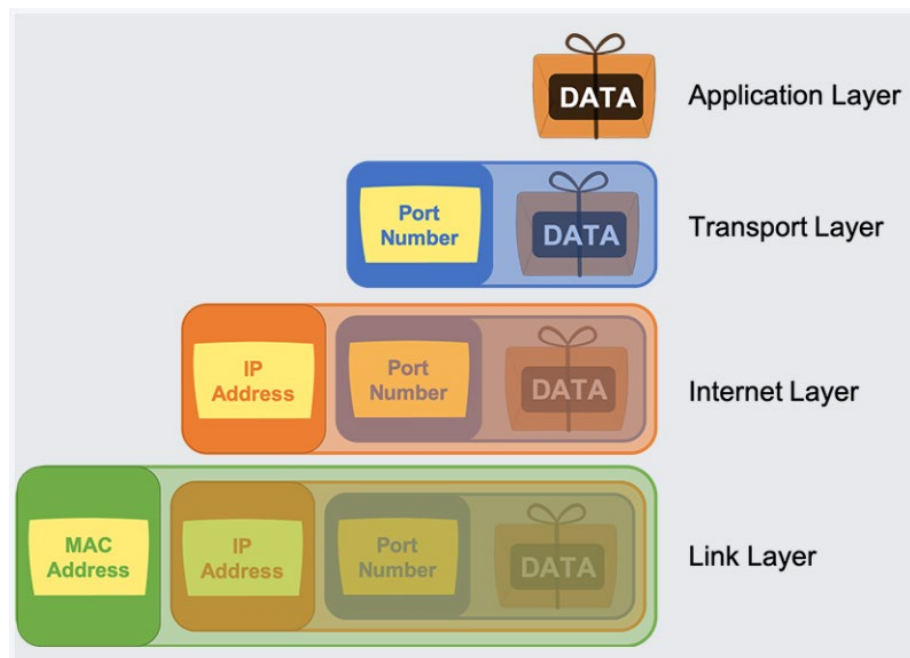
Summary of Network hardware devices:

https://adacomputerscience.org/concepts/network_devices?examBoard=all&stage=all&topic=networking

1.2.1 Purpose of the Internet Protocol

The following diagram shows how the Transport layer defines a header to specify the destination port number, how the Internet layer defines a separate header to specify the destination IP address number, and how the Link layer defines its own header to

specify the destination MAC address. These headers provide all the address information necessary to deliver the network packet successfully.



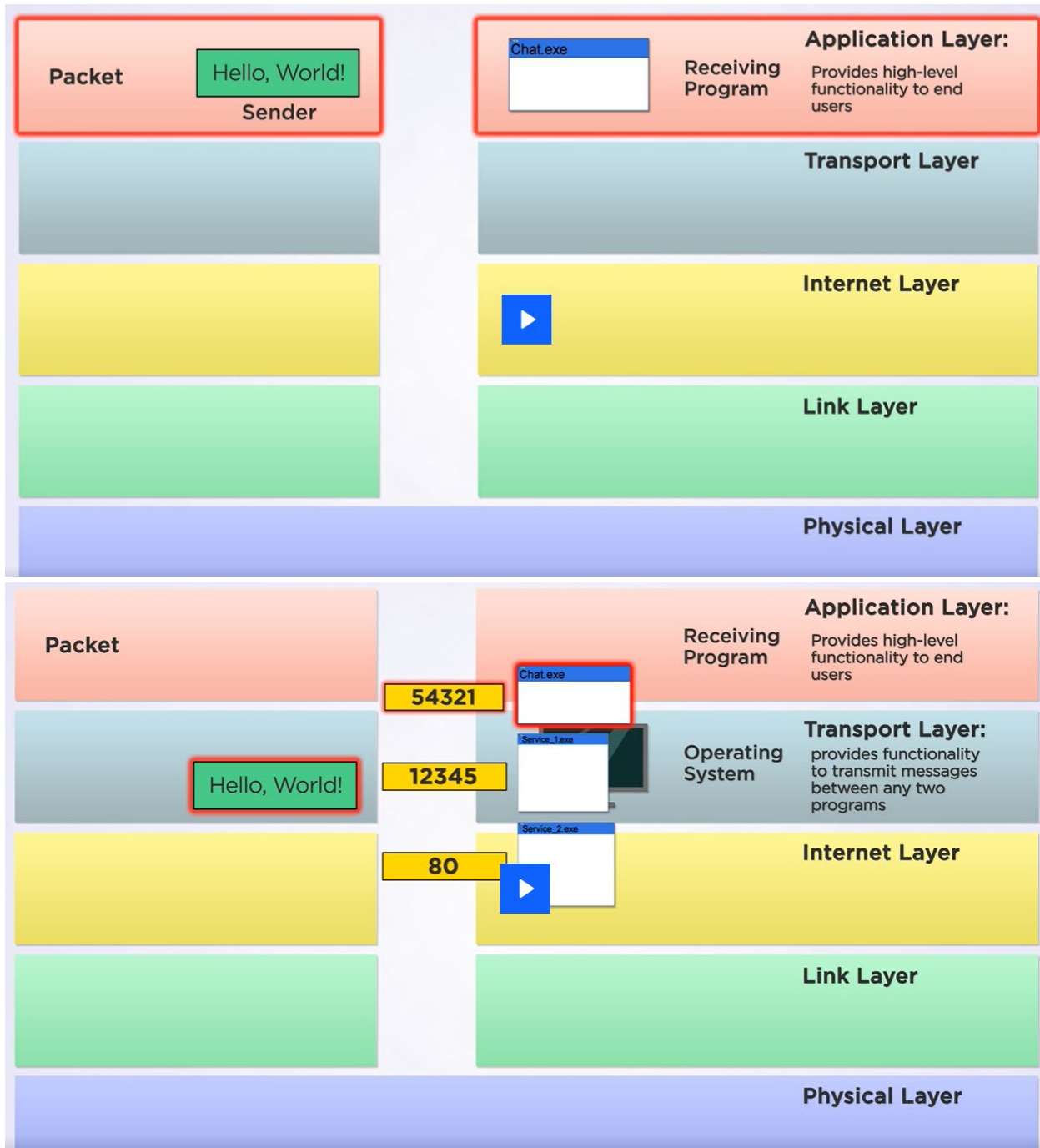
Before a packet is sent, the sending program attaches a MAC address, IP address, and port number so the packet has enough information to reach the receiving program.

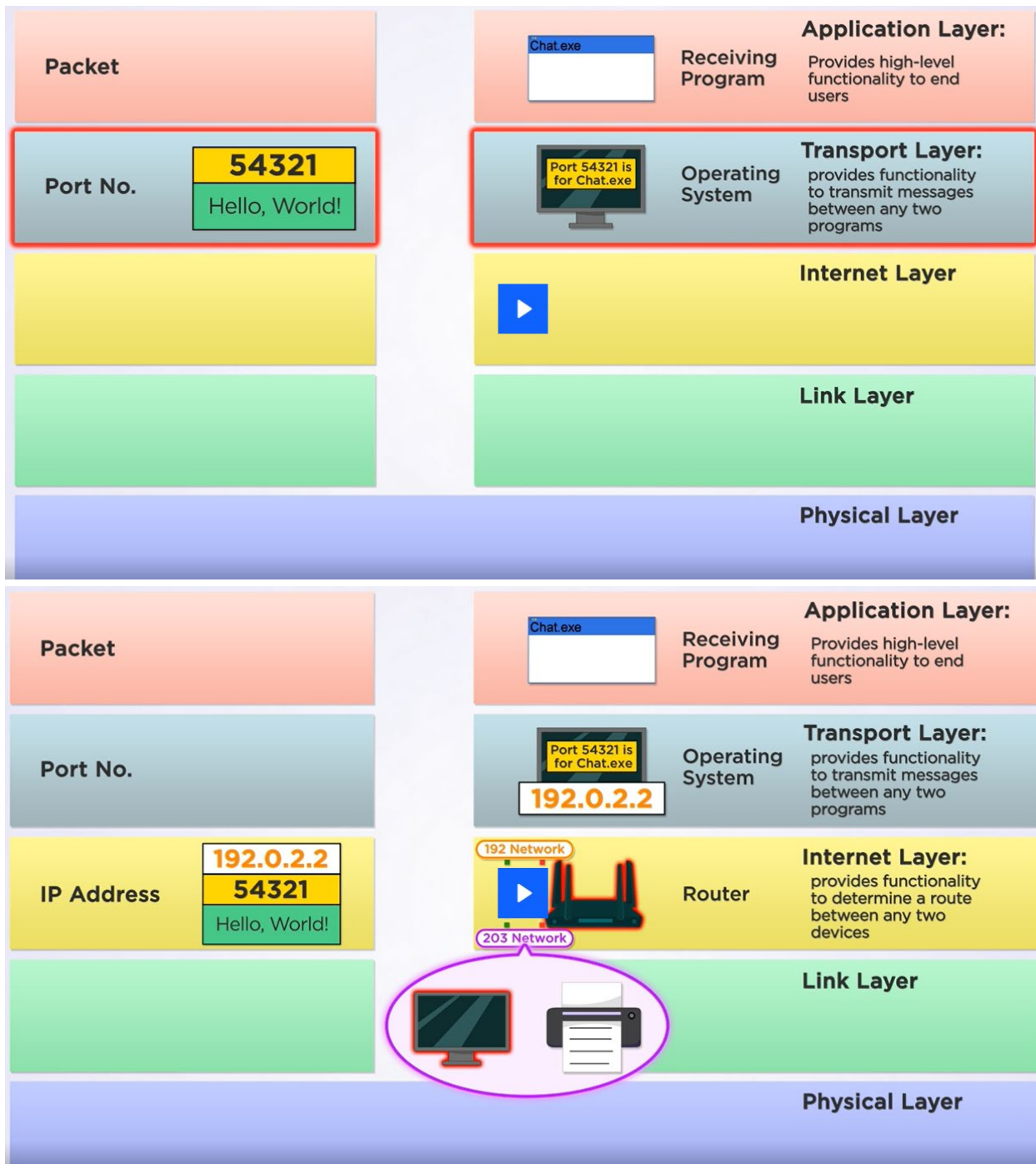
After the packet is sent, each layer acts on the packet using the attached information, usually without the knowledge of how the higher layers are implemented.

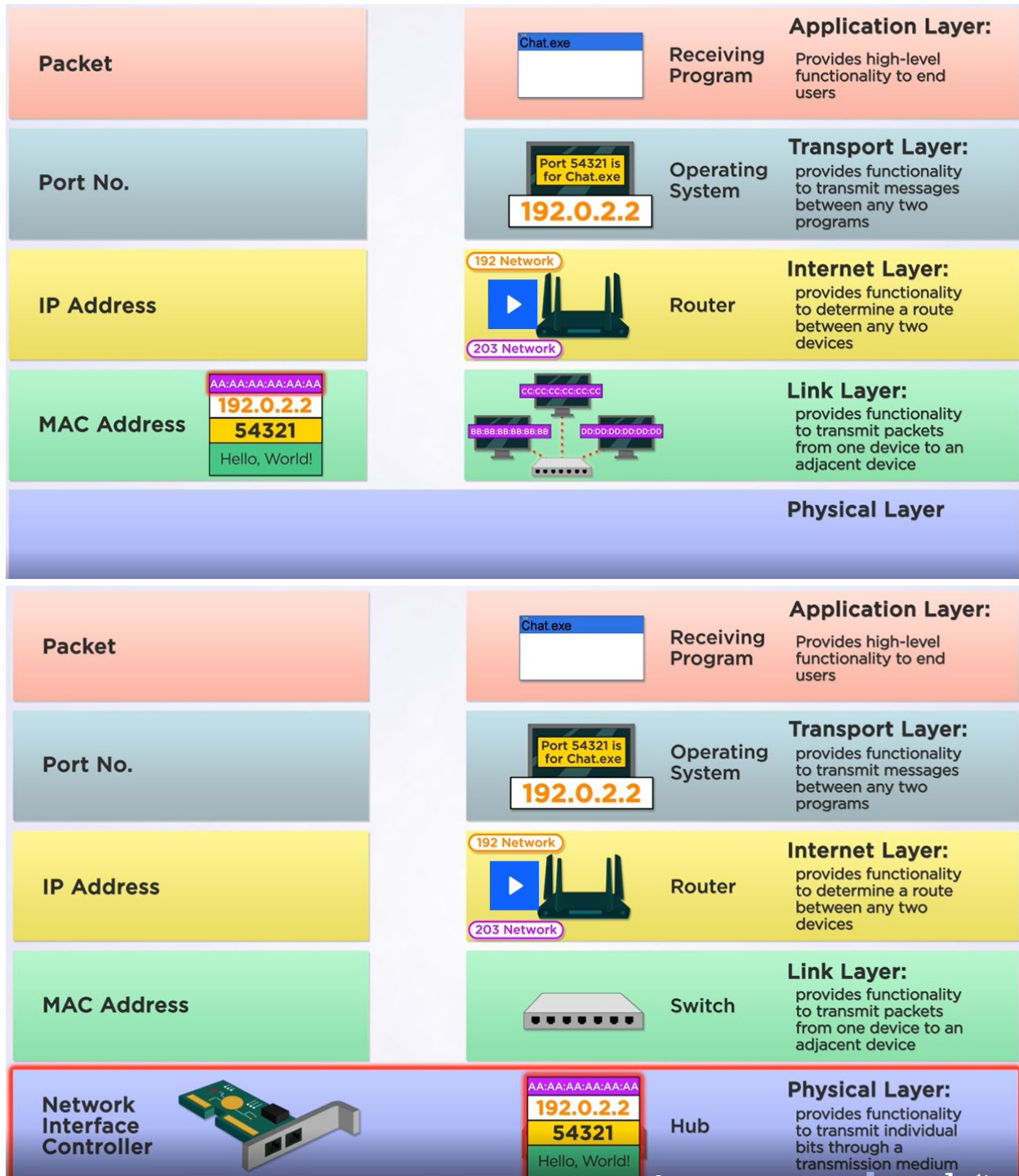
It is important to understand that the Link layer only transmits packets between devices that are "adjacent" and belong to the same local network in your home. To travel outside the local network, we need a router on the Internet layer that serves as a gateway to other networks. The router determines which other network may have a route to the destination network and retransmits the packet to that network. In this way, the packet may travel through multiple gateways and different networks before reaching its destination.

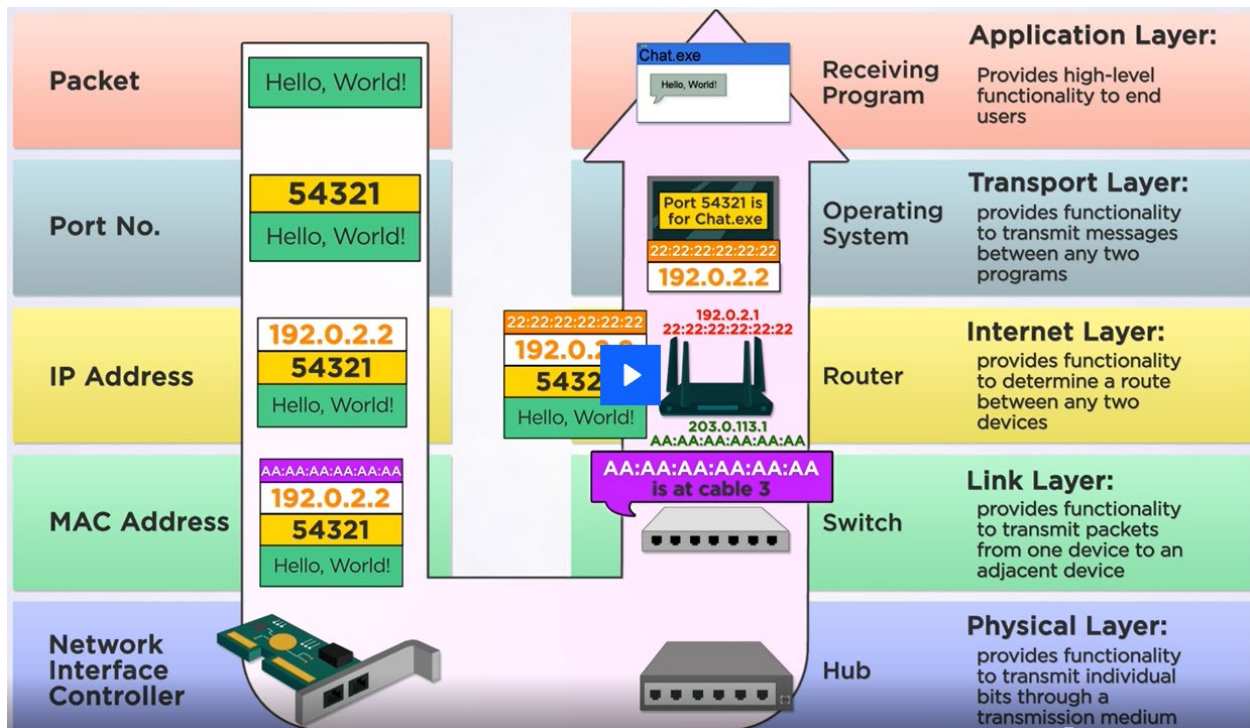
1.2.2 How Internet Protocol Works

Please refer to the video on SLS for more information. The link is provided earlier.

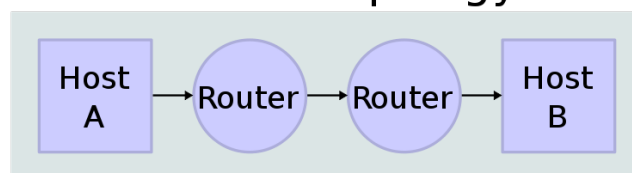




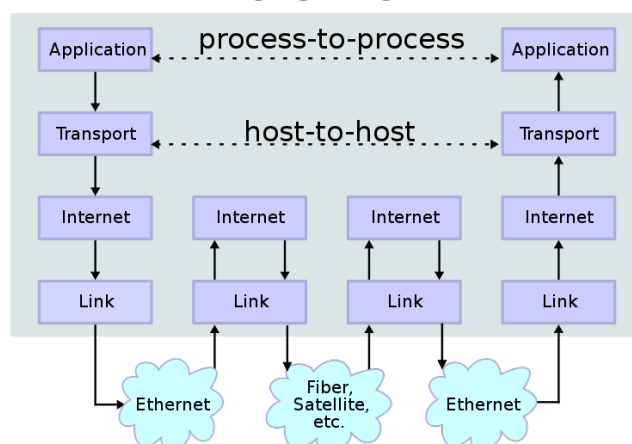




Network Topology



Data Flow



Source: https://en.wikipedia.org/wiki/Internet_protocol_suite#/media/File:IP_stack_connections.svg

If you wonder what network topologies are, refer to this link:

https://adacomputerscience.org/concepts/network_topologies?examBoard=all&stage=all&topic=networking

1.3 Well-known Protocols

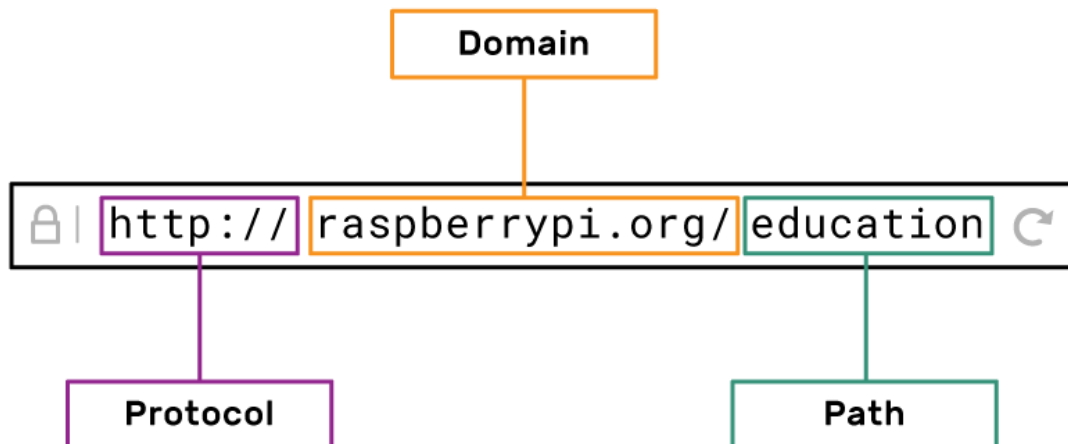
Protocol	Layer	Full name and purpose
HTTP	Application	Hypertext Transfer Protocol. Used to make a request for a webpage. The server returns the page or an error code if there was a problem with the request.
HTTPS	Application	Hypertext Transfer Protocol Secure. Sends an encrypted request for a webpage. The server returns the encrypted page or an error code if there was a problem with the request.
FTP	Application	File Transfer Protocol. Used to upload or download a file. The server opens a data connection (over which the file will be transferred) or sends an error code if there was a problem with the request.
SMTP	Application	Simple Mail Transfer Protocol. Used to send an email to an email server. The server returns a code indicating whether or not the email could be delivered.
POP	Application	Post Office Protocol. Used to request any new emails for a specific email account. The server returns the emails (if there are any).

IMAP	Application	Internet Message Access Protocol. Used to synchronise a client email account with an account on a mail server. The server returns new emails (if there are any) and deletes any emails that were deleted locally on the client application. This allows a user to use multiple devices to access their email account.
DHCP	Application	Dynamic Host Configuration Protocol. Used to assign IP addresses and other configuration options to devices in a network.
TCP	Transport	Transmission Control Protocol. When data is to be sent (whether from client or server), the data is split into packets and each packet is given a sequence number. This is a reliable transmission protocol. At the receiving end, the packets are checked. If any packets go missing, they will be resent.
UDP	Transport	User Datagram Protocol. Data is split into packets (as with TCP). However, this is an unreliable transmission protocol. If any packets arrive out of sequence or are missing, they are ignored. UDP is suitable where data does not have to be 100% accurate but speed is important, e.g. with some video streaming services or VOIP.
IP	Network (Internet)	Internet Protocol. Creates a new packet (imagine putting the packet from the transport layer into a new envelope). Adds the source and destination IP addresses to allow the packet to be delivered. These are the packets that are routed across the internet.
Ethernet and Wi-Fi protocols	Link (data link)	Encapsulates the data from the previous layer into frames (another kind of packet) with a source and destination MAC address, and manages multiple transmissions on the media.

Source:

https://adacomputerscience.org/concepts/network_protocols?examBoard=all&tag=e=all&topic=networking

1.3.1 How HTTP Works (How the Server Handles Requests from Clients)



HTTP is a protocol for any data exchange on the Web such as fetching resources. It is a client-server protocol, which means requests are initiated by the recipient, usually the Web browser. HTTP requests web pages from a web server and can send information when a user fills in a form. The web pages that get sent back are formatted using Hypertext Markup Language (HTML). A complete document is reconstructed from the different sub -documents fetched, for instance, text, layout description, images, videos, scripts, and more.

HTTP is gradually being replaced with Hypertext Transfer Protocol Secure (HTTPS). HTTPS encrypts the traffic between the application and the server to protect user's data. Many browsers denote HTTPS with a padlock symbol to show the link is secure. You should only enter personal information into websites that support HTTPS.

Refer to the following for more details on HTTP.

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview>

https://adacomputerscience.org/concepts/internet_application_layer?examBoard=all&stage=all&topic=the_internet

As a quick summary, the HTTP protocol works as follows:

- The client (usually a browser) opens a connection to the server and sends a request.
- The server processes the request generates a response and closes the connection if it finds a **Connection: Close** header. The request consists of a line indicating a method such as GET or POST, a Uniform Resource Identifier (URI) indicating which resource is being requested, and an HTTP protocol version separated by spaces. This is normally followed by a number of headers, a blank line indicating the end of the headers, and sometimes body data. Headers may provide various information about the request or the client's body data. Headers are typically only sent for POST and PUT methods. The example request shown below would be sent by a browser to request the server foo.com to send back the resource in /index.html. In this example, no body data is sent because the method is GET (the point of the request is to get some data, not to send it).

```
GET /index.html HTTP/1.0
User-agent: Mozilla
Accept: text/html, text/plain, image/jpeg, image/gif, */*
Host: foo.com
```

The server receives the request and processes it. It handles each request individually, although it may process many requests simultaneously. Each request is broken down into a series of steps that together make up the request-handling process.

When this request message reaches the server, the server can take either one of these actions:

- The server interprets the request received, maps the request into a file under the server's document directory, and returns the file requested to the client.
- The server interprets the request received, maps the request into a program kept in the server, executes the program, and returns the output of the program to the client.

- The request cannot be satisfied, the server returns an error message.

The server generates a response that includes the HTTP protocol version, HTTP status code, and a reason phrase separated by spaces. This is normally followed by a number of headers. The end of the headers is indicated by a blank line. The body data of the response follows. A typical HTTP response might look like this:

```
HTTP/1.0                200                OK
Server:                  Sun-Java           System-Web-Server/6.1
content-type:            text/html
Content-length:          83

<HTML>
<HEAD><TITLE>Hello      World</Title></HEAD>
<BODY>Hello             World</BODY>
</HTML>
```

The status code and reason phrase tell the client how the server handled the request. Normally the status code 200 is returned, indicating that the request was handled successfully and the body data contains the requested item. Other result codes indicate a redirection to another server or the browser's cache, or various types of HTTP errors such as 404 Not Found.

The browser receives the response message, interprets the message and displays the contents of the message on the browser's window according to the media type of the response (as in the Content-Type response header). Common media type include "text/plain", "text/html", "image/gif", "image/jpeg", "audio/mpeg", "video/mpeg", "application/msword", and "application/pdf".

To learn about other Application-layer protocols in action, refer to: https://adacomputerscience.org/concepts/internet_application_layer?examBoard=all&stage=all

1.3.2 File Transfer Protocol (FTP)

As the name suggests, FTP moves files from a client device to a file server. The FTP application may be a dedicated application on the client device or built into a web browser or other software.

Some examples where FTP would be used are:

- A website developer designing a website on a local device and then transferring all the files to the web server
- Downloading software
- Downloading a music file
- Some video clips embedded in web pages

Generally, if you end up with an actual file in a folder on your device, you probably used FTP to get that file.

Just like HTTP, FTP is not secure, but there is a secure version of it: SFTP.

1.3.3 Simple Mail Transfer Protocol (SMTP)

Email was initially one of the main uses of the internet and still accounts for a significant number of data transactions.

Users have email addresses such as test@raspberrypi.org. A specific email server manages all the accounts with the same address after the @.

When a user sends a message, using an email client, an app, or a webmail page, the message is sent to the mail server using a protocol called SMTP.

If the mail is for someone in the same organisation, the mail is stored until that user connects. If not, the domain name of the destination is resolved using the DNS and the email is forwarded to the destination mail server. Again SMTP is used to transfer the email.

1.3.4 Post Office Protocol (POP) and Internet Message Access Protocol (IMAP)

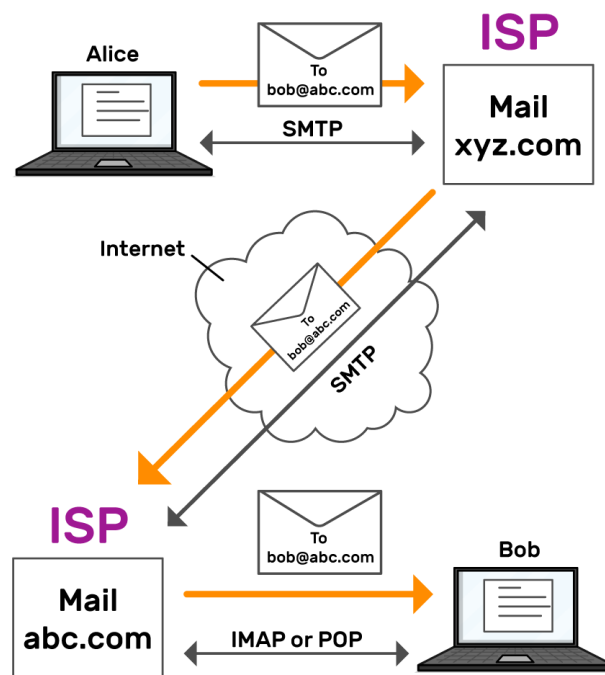
Once an email arrives at the destination mail server, it is stored in the recipient's 'mailbox' until they connect to read it.

If a user connected using the Post Office Protocol, or POP, then the actual email is transferred to the client's device and deleted from the server.

This used to cause problems when groups of people were sharing an email account, e.g. a specific company account that people message when they need support (such as support@acme.com).

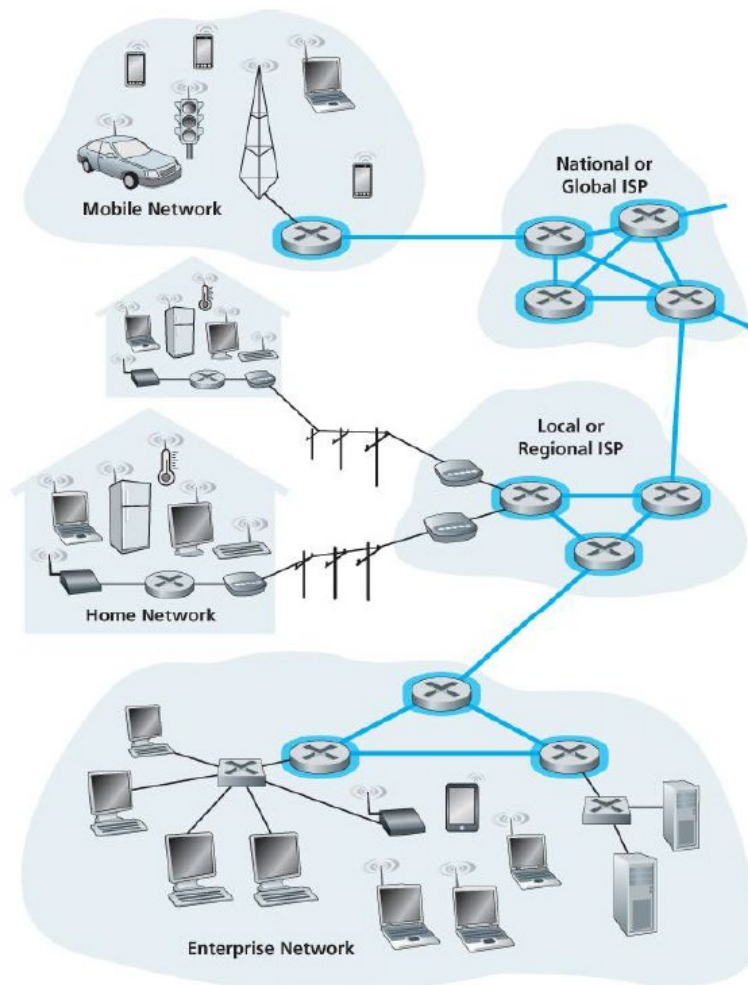
To combat this and give additional functionality, the Internet Message Access Protocol, or IMAP was developed. IMAP allows messages to be retrieved while a copy is maintained on the server; a specific request to delete an email is sent when the email is no longer needed.

With webmail, IMAP is used to pull the emails from the web server so they are displayed on the web page.



2. Network Core

The figure below highlights the network core with thick, shaded lines.



Source: Kurose, J., & Ross, K. (2016). *Computer Networking: A Top-Down Approach* (7th ed.). Pearson.

In a network application, end systems exchange messages with each other. Messages may perform a control function or can contain data, such as an e-mail message, a JPEG image, or an MP3 audio file. To send a message from a source end system to a destination end system, the source breaks long messages into smaller chunks of data known as packets. Between the source and destination, each packet travels through communication links and packet switches such as routers and link-layer switches.

There are two fundamental approaches to moving data through a network of links and switches: circuit switching and packet switching.

In circuit-switched networks, the resources needed along a path such as buffers and link transmission rate to provide for communication between the end systems are reserved for the duration of the communication session between the end systems.

In packet-switched networks such as the Internet, these resources are not reserved as each session's messages use the resources on demand. As a result, if one of the links is congested because other packets need to be transmitted over the link at the same time, then the packet will have to wait in a buffer at the sending side of the transmission link and suffer a delay. The Internet makes its best effort to deliver packets in a timely manner, but it does not make any guarantees.

2.1 Packet-switching

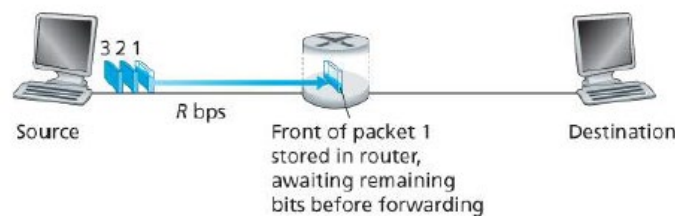
There are two types of services in packet switching: connectionless service or connection-oriented service.

In a connectionless service, a packet is dispatched with no knowledge of whether or not the receiver is ready to accept it.

In a connection-oriented service, the first packet sent includes a request for an acknowledgment. If this is received, the sender transmits further packets. If no acknowledgment is received the sender tries again with the first packet.

2.1.1 Store-and-Forward Transmission

Most packet switches use store-and-forward transmission at the inputs to the links. This means that the packet switch must receive the entire packet before it can begin to transmit the first bit of the packet onto the outbound link.



Source: Kurose, J., & Ross, K. (2016). *Computer Networking: A Top-Down Approach* (7th ed.). Pearson.

A router will typically have many incident links since its job is to switch an incoming packet onto an outgoing link. As such, to aid in the explanation, we will consider a simple network consisting of two end systems connected by a single router, as shown above. The router has the rather simple task of transferring a packet from one input link to the only other attached link. We will also ignore propagation delay, which is the time it takes for the bits to travel across the wire at nearly the speed of light.

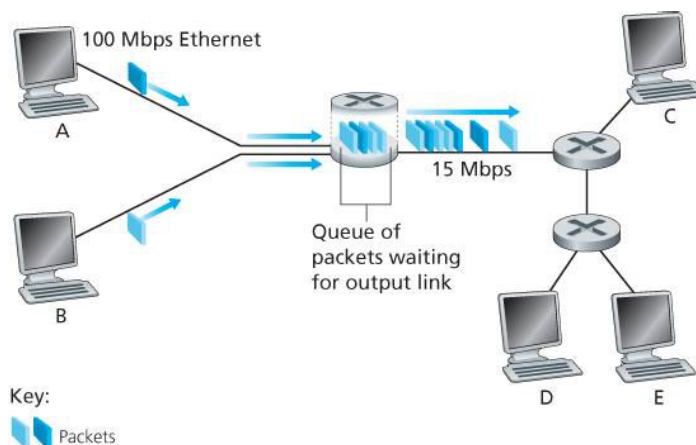
In this example, the source has three packets, each consisting of L bits, to send to the destination. At the time of the snapshot of the diagram above, the source has transmitted some of packet 1, and the front of packet 1 has already arrived at the router. Because the router employs store-and-forward, at this instant of time, the router cannot transmit the bits it has received; instead, it must first buffer or store the packet's bits. Only after the router has received all of the packet's bits, can it begin to transmit or forward the packet onto the outbound link.

2.1.2 Queuing Delays and Packet Loss

Each packet switch has multiple links attached to it. So, for each attached link, the packet switch has an output buffer or output queue, which stores packets that the router is about to send into that link.

If an arriving packet needs to be transmitted onto a link but finds the link busy with the transmission of another packet, the arriving packet must wait in the output buffer. Thus, in addition to the store-and-forward delays, packets suffer output buffer queuing delays. These delays are variable and depend on the level of congestion in the network.

Since the amount of buffer space is finite, an arriving packet may find that the buffer is completely full with other packets waiting for transmission. In this case, packet loss will occur. This means that either the arriving packet or one of the already -queued packets will be dropped.



Source: Kurose, J., & Ross, K. (2016). *Computer Networking: A Top-Down Approach* (7th ed.). Pearson.

Consider the above simple packet-switched network. Packets are represented by three-dimensional slabs. The width of a slab represents the number of bits in the packet. In this figure, all packets have the same width and length.

Suppose Hosts A and B are sending packets to Host E. Hosts A and B first send their packets along 100 Mbps Ethernet links to the first router. The router then directs these packets to the 15 Mbps link. If the arrival rate of packets to the router (when converted to bits per second) exceeds 15 Mbps, congestion will occur at the router as packets queue in the link's output buffer before being transmitted onto the link.

2.1.2.1 What if the internet is "busy"

At times, some routers may receive packets faster than they are able to route them on. These packets are buffered in memory and this introduces delays (referred to as a 'high latency'). For most traffic this is not an issue; it just means that the web page takes longer to load or a file download takes more time than expected. However, for other network traffic these delays may have more of an impact, particularly if you are talking over a voice call or playing an online game. If the buffering is severe, the router may run out of memory and packets are simply discarded.

IP packets contain a Type of Service (ToS) field in the packet header; this makes it possible to mark packets with a priority level and thus request special treatment i.e. to be placed at the front of the queue of packets to be routed. However, routers may choose to implement or ignore these requests.

Source:

https://adacomputerscience.org/concepts/internet_structure?examBoard=all&stage=all&topic=the_internet

2.1.2.2 Can a packet get lost?

A situation can occur where packets are sent to a destination address that is unreachable. Routers, unaware of this, may route such packets toward a default device. This default device may also pass the packet on, causing a loop to occur. In such circumstances, a packet could loop forever between routers. To prevent this, internet packets have a time to live (TTL) counter in their header. This is initially set when the packet is created, and reduced by one every time it goes through a router. If the counter reaches zero, the packet is discarded.

Source:

https://adacomputerscience.org/concepts/internet_structure?examBoard=all&stage=all&topic=the_internet

2.1.3 Forwarding Tables and Routing Protocols

How does the router determine which link it should forward the packet onto? Packet forwarding is actually done in different ways in different types of computer networks. Here, we briefly describe how it is done on the Internet.

On the Internet, every end system has an address called an IP address. When a source end system wants to send a packet to a destination end system, the source includes the destination's IP address in the packet's header.

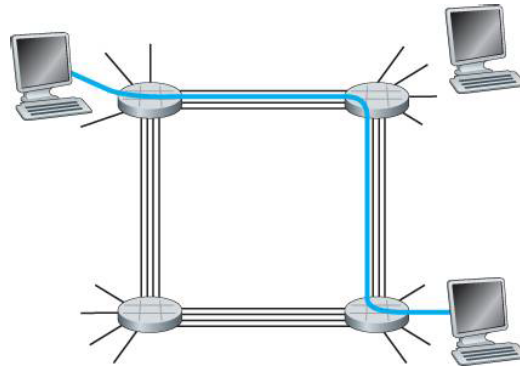
Each router has a forwarding table that maps destination addresses or portions of the destination addresses to that router's outbound links. When a packet arrives at a router, the router examines the address and searches its forwarding table, using this destination address, to find the appropriate outbound link. The router then directs the packet to this outbound link.

2.1.3.1 How do forwarding tables get set?

The Internet has a number of special routing protocols that are used to automatically set the forwarding tables. A routing protocol may, for example, determine the shortest path from each router to each destination and use the shortest path results to configure the forwarding tables in the routers.

You may access a Traceroute program to see the end-to-end route that packets take on the Internet. One such program can be found at www.traceroute.org. Choose a source in a particular country, and trace the route from that source to your computer.

2.2 Circuit-switching



Source: Kurose, J., & Ross, K. (2016). *Computer Networking: A Top-Down Approach* (7th ed.). Pearson.

The above figure illustrates a circuit-switched network. In this network, the four circuit switches are interconnected by four links. Each of these links has four circuits so that each link can support four simultaneous connections. The hosts such as PCs and workstations are each directly connected to one of the switches.

When two hosts want to communicate, the network establishes a dedicated end-to-end connection between the two hosts. Thus, in order for Host A to communicate with Host B, the network must first reserve one circuit on each of the two links. In this example, the dedicated end-to-end connection uses the second circuit in the first link and the fourth circuit in the second link. Because each link has four circuits, for each link used by the end-to-end connection, the connection gets one-fourth of the link's total transmission capacity for the duration of the connection. Thus, for example, if each link between adjacent switches has a transmission rate of 1 Mbps, then each end-to-end circuit-switch connection gets 250 kbps of the dedicated transmission rate.

2.3 Packet-switching vs Circuit-switching

Critics of packet switching have often argued that packet switching is not suitable for real-time services, for example, telephone calls and video conference calls. This is due to its variable and unpredictable end-to-end delays such as queuing delays.

Proponents of packet switching argue that it offers better sharing of transmission capacity than circuit switching and it is simpler, more efficient, and less costly to implement than circuit switching.

Why is packet switching more efficient?

Circuit switching pre-allocates the use of the transmission link regardless of demand, with allocated but unneeded link time going unused. Packet switching, on the other hand, allocates link use on demand. Link transmission capacity will be shared on a packet-by-packet basis only among those users who have packets that need to be transmitted over the link.

Although packet switching and circuit switching are both prevalent in today's telecommunication networks, the trend has certainly been in the direction of packet switching. Many of today's circuit-switched telephone networks are slowly migrating towards packet switching. In particular, telephone networks often use packet switching for the expensive overseas portion of telephone calls.

A good summary of packet switching and circuit switching is available here:

https://adacomputerscience.org/concepts/internet_structure?examBoard=all&stage=all&topic=the_internet

Referenced Readings

MOE Student Learning Space (SLS) Network and Communication, Protocols and Protocol Layers, Simple Home Networks

Kurose, J., & Ross, K. (2016). Computer Networking: A TopDown Approach (7th ed.). Pearson.

https://gaia.cs.umass.edu/kurose_ross/lectures.php

Langfield, S., & Duddell, D. (2015). Cambridge International AS and A Level Computer Science Coursebook. Cambridge University Press.

GeeksforGeeks. (2021, June 28). Computer Network Tutorials. Retrieved February, 2022, from <https://www.geeksforgeeks.org/computer-network-tutorials/?ref=lbp>

(IT) Information Technology Certifications | CompTIA IT Certifications. (n.d.). Default. <https://www.comptia.org/resources>

Cisco - Networking, Cloud, and Cybersecurity Solutions. (2022, January 28). Cisco.

<https://www.cisco.com/>

<https://www.netacad.com/#learners>

<https://www.netacad.com/courses/packet-tracer>

How the Server Handles Requests from Clients (Sun Java System Web Server 6.1 SP11 NSAPI Programmer's Guide). (2010). Docs.Oracle.Com.

<https://docs.oracle.com/cd/E19857-01/820-7655/abvah/index.html>

In Introduction to HTTP Basics. (2009). Chua Hock-Chuan.

https://www3.ntu.edu.sg/home/ehchua/programming/webprogramming/http_basics.html

<https://www.internetsociety.org/>

<https://www.webfx.com/blog/web-design/the-history-of-the-internet-in-a-nutshell/>

<https://www.computerhistory.org/timeline/networking-the-web/>

<https://techdifferences.com/difference-between-connection-oriented-and-connection-less-services.html>

<https://adacomputerscience.org/?examBoard=all&stage=all>

<https://www.cloudflare.com/learning/ddos/glossary/open-systems-interconnection-model-osi/>

<https://skillsforall.com/course/networking-basics?userLang=en-US>

<https://www.open.edu/openlearn/digital-computing/discovering-computer-networks-hands-on-the-open-networking-lab/content-section-overview?active-tab=description-tab>

<https://tryhackme.com/room/httpindetail>

<https://tryhackme.com/room/whatisnetworking>

2023/VJC/JC2/H2Computing/9569

<https://tryhackme.com/room/introtonetworking>

Chapter 21: Fundamentals of Computer Networks (Part 2)

Syllabus Learning Outcomes

4.1 Fundamentals of Computer Networks

Understand computer network technology

4.1.1 Explain the concepts of LAN, WAN, intranet, and the structure of the internet.

4.1.2 Understand the concepts of IP addressing and domain name server (DNS).

1 Local Area Network (LAN)

1.1 What is LAN

A local area network (LAN) is a collection of devices connected together in one physical location, such as a building, office, or home. A LAN can be small or large, ranging from a home network with one user to an enterprise network with thousands of users and devices in an office or school.

Regardless of size, a LAN's single defining characteristic is that it connects devices that are in a single, limited area. In contrast, a wide area network (WAN) or metropolitan area network (MAN) covers large geographic areas. Some WANs and MANs connect many LANs together.

A LAN comprises cables, access points, switches, routers, and other components that enable devices to connect to internal servers, web servers, and other LANs via wide area networks.

The rise of virtualization has also fueled the development of virtual LANs, which enable network administrators to logically group network nodes and partition their networks without a need for major infrastructure changes. For example, in an office with multiple departments, such as accounting, IT support, and administration, each department's computers could be logically connected to the same switch but segmented to behave as if they are separate.

Types of LAN Topologies and Advantages/Disadvantages:

<https://www.comparitech.com/net-admin/network-topologies-advantages-disadvantages/>

https://adacomputerscience.org/concepts/network_topologies?examBoard=all&stage=all&topic=networking

1.2 Types of LAN

In general, there are two types of LANs: client/server LANs and peer-to-peer LANs.

A client/server LAN consists of several devices, i.e. the clients, connected to a central server. The server manages file storage, application access, device access, and network traffic. A client can be any connected device that runs or accesses applications or the Internet. The clients connect to the server either with cables or through wireless connections.

Typically, suites of applications can be kept on the LAN server. Users can access databases, email, document sharing, printing, and other services through applications running on the LAN server, with read and write access maintained by a network or IT administrator. Most midsize to large businesses, government, research, and education networks are client/server-based LANs.

A peer-to-peer LAN doesn't have a central server and cannot handle heavy workloads like a client/server LAN can, and so they are typically smaller. On a peer-to-peer LAN, each device shares equally in the functioning of the network. The devices share resources and data through wired or wireless connections to a switch or router. Most home networks are peer-to-peer.

1.3 Wireless Local Area Network (WLAN)

A Wireless Local Area Network (WLAN) is a group of colocated computers or other devices that form a network based on radio transmissions rather than wired connections. A Wi-Fi network is a type of WLAN. It wasn't until the wide deployment of Wi-Fi technology that LANs became commonplace in nearly every type of environment as more types of devices can be connected to a LAN.

In wired LAN, Gigabit Ethernet and Fast Ethernet are most commonly used. They are capable of consistently transmitting data at speeds of 1000 Mbps and 100 Mbps respectively. In comparison, the newest Wi-Fi standards operate as follows:

802.11ah: Operates on frequency bandwidths below 1GHz, meaning it's capable of longer distance transmission than other standards. The top speed for 802.11ah is 347Mbps.

802.11ad: Operates on a bandwidth of 60GHz at speeds up to 6.7Gbps but only works up to around 3 metres away from the access hub.

802.11ac (Wi-Fi 5): Operates on a bandwidth of 5GHz or 2.4GHz, depending on the router, with speeds up to 3.46Gbps.

The latter two standards are faster than Gigabit Ethernet, but these theoretical Wi-Fi speeds may not match up with the actual speeds one will experience.

Wi-Fi standards are designed to allow a non-stationary device's connection to jump from one access point to another. With non-overlapping access points, the device's connection is simply paused until a connection with the next access point is established.

When access points overlap, they can be configured to help optimise the network by sharing and managing loads. This is known as building a mesh network. It basically extends a WLAN's reach and performance. A mesh network provides multiple transmission paths and with intelligent algorithms, it can manage routing to improve performance.

1.4 Components Necessary for a LAN Connection

To build a LAN or WLAN, the following components are needed:

1. Computers with Network Interface Cards (NICs)

Most computers come with a NIC for Ethernet and one for Wi-Fi. This allows the computer to connect to a network. The NIC should come with driver software that the operating system automatically configures and updates.

2. Cables

Unshielded Twisted Pair (UTP) cables are the standard (CatX cables, where X is a number that indicates the speed a cable will fetch). For fast speeds, fibre optic cables are used..

3. Switches and Hubs

Most LANs use both switches and hubs. A hub sends or broadcasts data packets to all the computers on a LAN, while switches dedicate packet transmission to a single computer, which reduces network traffic.

4. Routers:

Network routers allow you to connect to other LANs and they assign IP addresses to your devices. A wireless router will allow computers with Wi-Fi NICs to connect with each other and with the internet.

5. Modem:

To connect a LAN to other LANs and to the internet to form a WAN, a modem facilitates signal conversion and reversion (analog to digital and vice versa)

6. Software:

Network software should come with the operating system of the computers. Security software is also needed to protect your devices.

7. Dynamic Host Configuration Protocol (DHCP) Server:

A PC with plenty of RAM and hard drive space is needed to act as a DHCP server.

1.5 How is a WLAN created?

A WLAN can be configured in one of two ways: Infrastructure or Ad hoc.

In infrastructure mode, the endpoints are all connected through a base station and in turn, to the Internet. A home or office Wi-Fi network is an example of a WLAN set up in infrastructure mode. A basic infrastructure WLAN can be set up with just a few parts: a wireless router, which acts as the base station, and endpoints, which can be

computers, mobile devices, printers, and other devices. In most cases, the wireless router is also the internet connection.

In an ad hoc setup, a WLAN connects endpoints such as computer workstations and mobile devices without the use of a base station. Use of Wi-Fi Direct technology is common for an ad hoc wireless network to provide basic peer-to-peer (P2P) communication. It is also easy to set up as it requires only two or more endpoints with built-in radio transmission, such as computers or mobile devices. After adjusting network settings for ad hoc mode, one user initiates the network and becomes visible to the others.

2 Wide Area Network (WAN)

A Wide Area Network (WAN) is a form of telecommunication network that can connect devices from multiple locations and across the globe. It is essentially a series of LANs or other networks such as Metropolitan Area Networks (MANs) linked together to form a network in an extended area. WANs are typically operated by telecommunications companies or businesses that need a network comprising multiple remote locations. The internet is considered as the largest WAN in the world.

These networks are often established by service providers that then lease their WAN to businesses, schools, governments or the public. These customers can use the network to relay and store data or communicate with other users, no matter their location, as long as they have access to the established WAN. Access can be granted via different links, such as virtual private networks (VPNs) or lines, wireless networks, cellular networks or internet access.

Fibre optic cable is the preferred transmission medium for WANs because fibre optics can transmit large amounts of data at high speeds.

For further readings on the types of networks, you may refer to this:

https://www.networkstraining.com/different_types-of-networks/

3 Intranet

An intranet is a private communications network based on TCP/IP and Web technologies, and it resembles a private version of the Internet. Intranets can span a

single network segment, a building, or all offices of a company. Web browsers act as interfaces to existing network applications and databases.

4 Extranet

Extranets are an extension of a company network or a collaborative Internet connection to key customers, suppliers, and partners. Extranets use private WAN lines or virtual private networks (VPNs) to provide access to specific company information and to build closer business relationships. VPNs use the Internet to create secure pathways between a company and its extranet partners. VPNs provide the same types of network services you expect in your private network, but they do it over the Internet, thus saving money. A typical extranet solution requires a router at each end, a firewall, authentication software, a server, and a dedicated WAN line or VPN over the Internet.

5 Internet

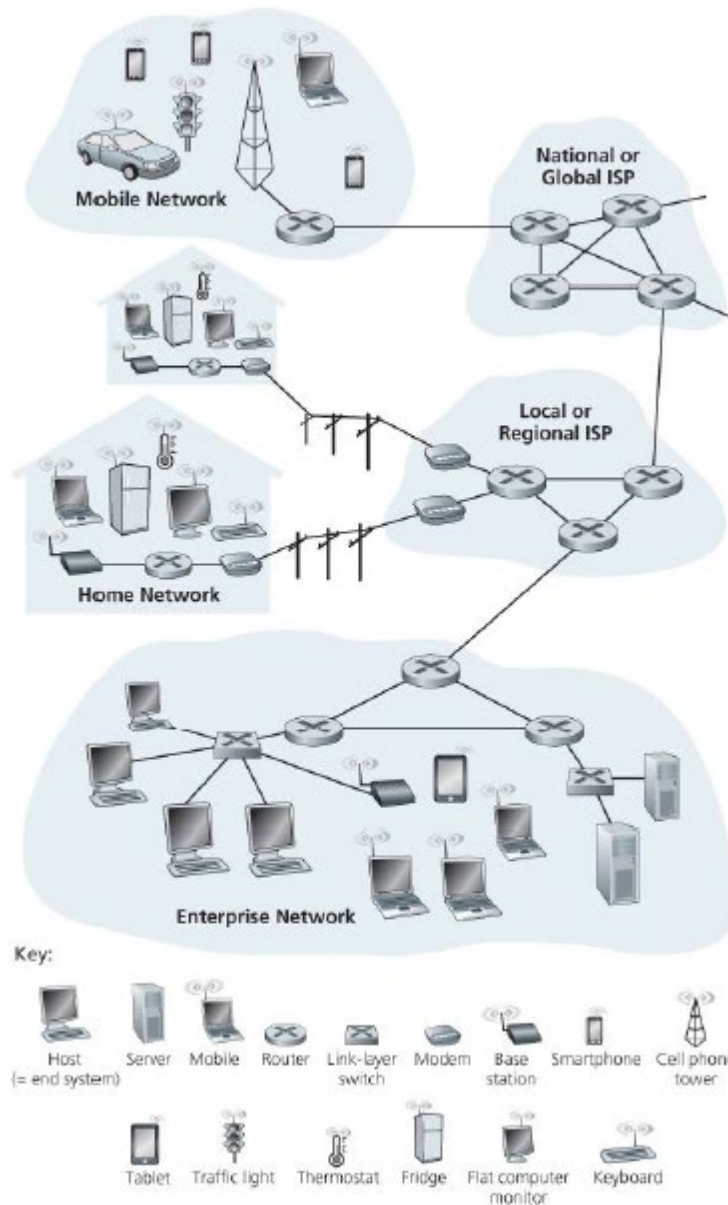
Internet is the global system of interconnected computer networks that uses the Internet protocol suite (TCP/IP) to communicate between networks and devices. It is a network of networks that consists of private, public, academic, business, and government networks of local to global scope, linked by a broad array of electronic, wireless, and optical networking technologies. It is essentially an infrastructure that provides services to applications.

The Internet carries a vast range of information resources and services, such as the inter-linked hypertext documents and applications of the World Wide Web (WWW), electronic mail, telephony, and file sharing. The applications are said to be distributed applications, since they involve multiple end systems that exchange data with each other.

Importantly, Internet applications run on end systems, i.e. they do not run in the packet switches in the network core. Although packet switches facilitate the exchange of data among end systems, they are not concerned with the application that is the source or sink of data. The computers and other devices connected to the Internet are often referred to as end systems. They are referred to as end systems because they sit at the edge of the Internet.

As shown in the figure below, the Internet's end systems include desktop computers (e.g., desktop PCs, Macs, and Linux boxes), servers (e.g., Web and mail servers), and

mobile devices (e.g., laptops, smartphones, and tablets). Furthermore, an increasing number of non-traditional things are being attached to the Internet as end systems (Internet of Things or IoTs). End systems are also referred to as hosts because they host application programs such as a Web browser program, a Web server program, an e-mail client program, or an e-mail server program.



Source: Kurose, J., & Ross, K. (2016). *Computer Networking: A Top-Down Approach* (7th ed.). Pearson.

5.1 Connecting to the internet

The networks you connect to at home, work or out and about all connect to the internet using another network. This secondary network comes from an internet service provider, or ISP.

ISPs connect with one another directly or through other networks. Large corporations and telecommunications providers also connect to these networks. The connections might use fibre-optic cables, satellite links, or radio transmissions, and this vast web forms the infrastructure of the internet.

The internet gives you access to networks hosting thousands of servers. These 'server farms' provide services such as email, storage, media streaming, and more.

5.2 Finding the IP address

An app on your smartphone may connect to the internet using a dedicated server, with a fixed IP address. In this case, the app will have the destination IP address built into its software.

If you're using a web browser, you can enter the exact URL of the site you want, follow a link on a web page, or use a search engine to find a website.

Once you have the URL you want, your browser tries to find the IP address belonging with that URL. This is where the domain name system (DNS) comes in: the browser sends a query to its local DNS server asking for the IP address of the URL in question. If the server knows the IP address, it replies with that address. If not, it sends the query on to a DNS server with a more global list of URLs. Eventually, your device will get back the IP address matching the URL.

5.3 Routing the IP packets

Once it has the IP address it needs, your device encapsulates your data in an IP packet with the destination address of the server. The IP packet travels over the network to the first router. But how does the router know where to send the packet next? Your home router only knows about the local networks it's connected to: it only has one default connection to the internet and the ISP.

More complex routers that your IP packet passes through will use the IP address to identify the network they need to reach. For example, all IP addresses starting 200.11 belong to a large telecom company. Inside the company, routers advertise the networks they know about to neighbouring routers. The neighbours advertise to other neighbouring routers, throughout the whole organisation. This way, the routers know which link to use to send an IP packet to reach any networks within that organisation.

These large organisations then advertise their summary routes to each other. This dramatically reduces the number of individual networks that each router must learn. With this information, each router can quickly send IP packets via the best links on its journey around the globe. Routers can also choose between multiple paths based on speed or bandwidth.

On the way, the changing links encapsulate the IP packet in a variety of different frames.

Finally, the IP packet arrives at a router connected to the same LAN as the destination server, and the IP packet will be delivered to that device.

5.4 Servers respond

The server responds, sending data back to the source. This data usually consists of many thousands of bytes, split into a series of fragments. Each fragment is encapsulated in its own IP packet and sent back over the internet.

This quantity of data could potentially congest the network and cause it to fail. So the returning packets may take different routes back through the network, and may arrive out of sequence.

The destination device reassembles the packets, decapsulates the data, and passes it to the app or web browser that requested it.

6 Differences between Internet, Intranet and Extranet

Point of difference	Internet	Intranet	Extranet
Accessibility of network	Public	Private - Specific organisation	Private - Suppliers and Vendors of organisation
Coverage	All over the world.	Restricted area up to an organisation.	Restricted area up to an organisation and its stakeholders.
Accessibility of content	Accessible to everyone connected.	Accessible only to the members of the organisation.	Accessible only to the members of the organisation and external members with logins.
No. of computers connected	It is the largest in number of connected devices.	The minimal number of devices are connected.	The connected devices are comparable to Intranet.
Owner	No one.	Single organisation.	Single/ Multiple organisations.
Purpose of the network	To share information throughout the world.	To share information throughout the organisation.	To share information between members and external members.
Security	It is dependent on the user of the device connected to the network.	It is enforced via a firewall.	It is enforced via a firewall that separates internet and extranet.

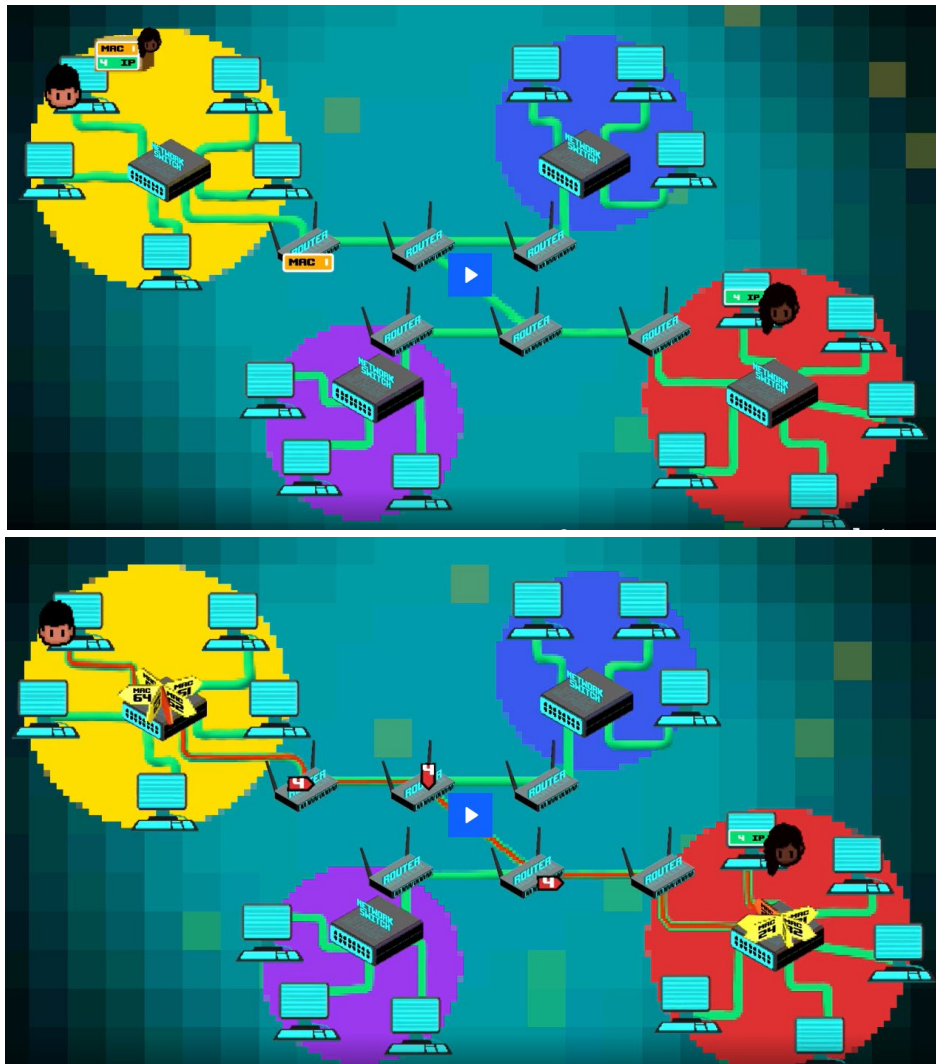
Point of difference	Internet	Intranet	Extranet
Policies behind setup	There is no hard and fast rule for policies.	Policies of the organisation are imposed.	Policies of the organisation are imposed.
Maintenance	It is maintained by ISPs.	It is maintained by CIO. HR or communication department of an organisation.	It is maintained by CIO. HR or communication department of an organisation.

7 Addresses in Networking

7.1 Introduction

Computers on the Internet need two different kinds of addresses (IP address and MAC address) for data transmission across the network. Both MAC Address and IP Address are used to uniquely define a device on the internet. Network Interface Card (NIC)'s Manufacturer provides the MAC Address while the Internet Service Provider (ISP) provides IP Address.

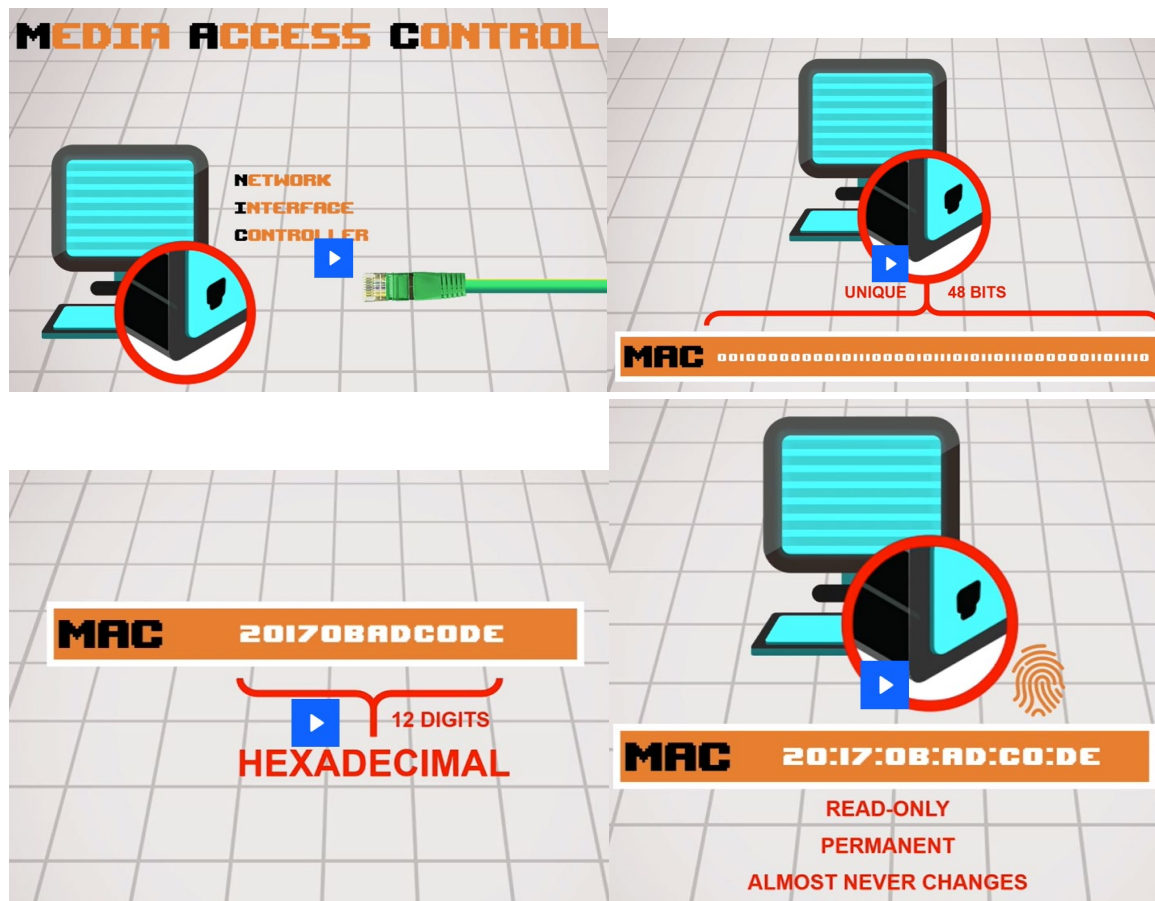
The main difference between MAC and IP address is that MAC Address is used to ensure the physical address of the computer. It uniquely identifies the devices on a network. In other words, MAC Addresses handle the physical connection from computer to computer. IP addresses are used to uniquely identifies the connection of the network in which the device is in. As such, we can say that IP Addresses handle the logical routeable connection from both computer to computer as well as network to network.

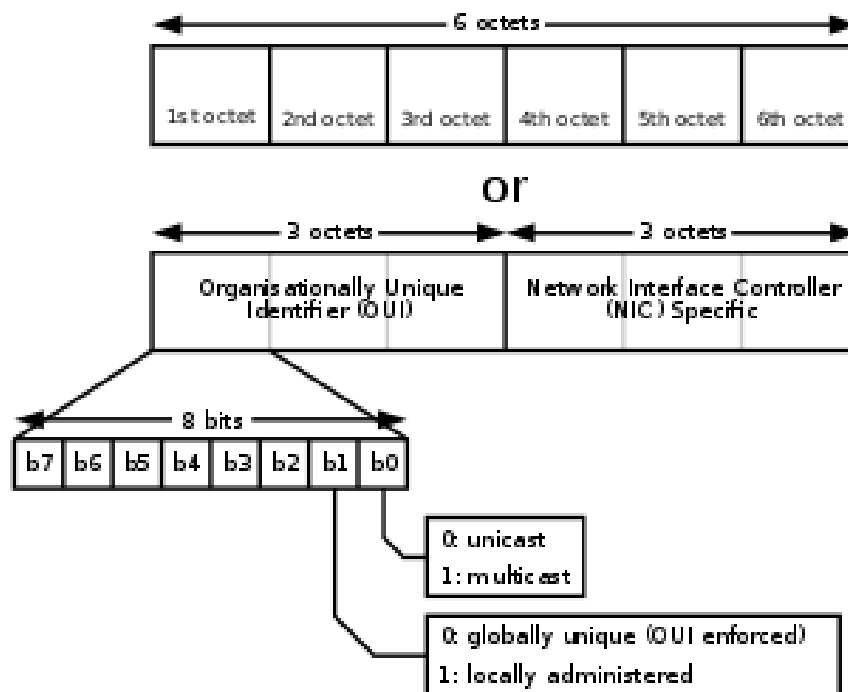


7.2 MAC Address

MAC Addresses are unique 48 -bits hardware number of a computer, which is embedded into network card (known as Network Interface Card) during the time of manufacturing. MAC Address is also known as Physical Address of a network device. MAC address is used by Media Access Control (MAC) sublayer of DataLink Layer.

Each MAC address is unique to the network card installed on a device, but the number of device-identifying bits is limited, which means manufacturers do reuse them. Each manufacturer has about 1.68 million available addresses, so when it burns a device with a MAC address ending in FF -FF-FF, it starts again at 00 -00-00. This approach





7.3 IP Address

IP address is a unique address having information about how to reach a specific host. It controls how devices on the internet communicate and defines the behavior of internet routers. It corresponds to Layer 3, the network layer. An IP address is a 32 bit unique address having an address space of 2^{32} .

An IP address identifies a device on the global internet, acting as the device's logical address to identify that network connection.

When IP addresses were invented, in the 1970s, it was decided to make them 32 bits long. 32 bits allow for 4 billion unique addresses, enough so that each person on Earth at the time could have one.

7.4 IPv4 Address

An IPv4 address consists of 32 bits or four 8 -bit chunks (bytes). It is, usually written as four decimal numbers, or a dotted quad. In other word s, the address is written in dotted decimal notation, with the four decimal values separated by dots.

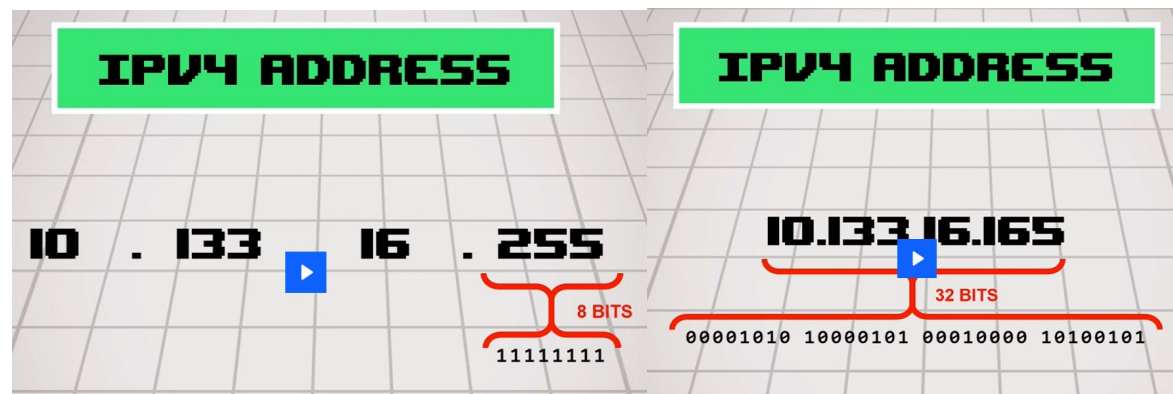
So the IP address 11000000101010000000111000010111 becomes

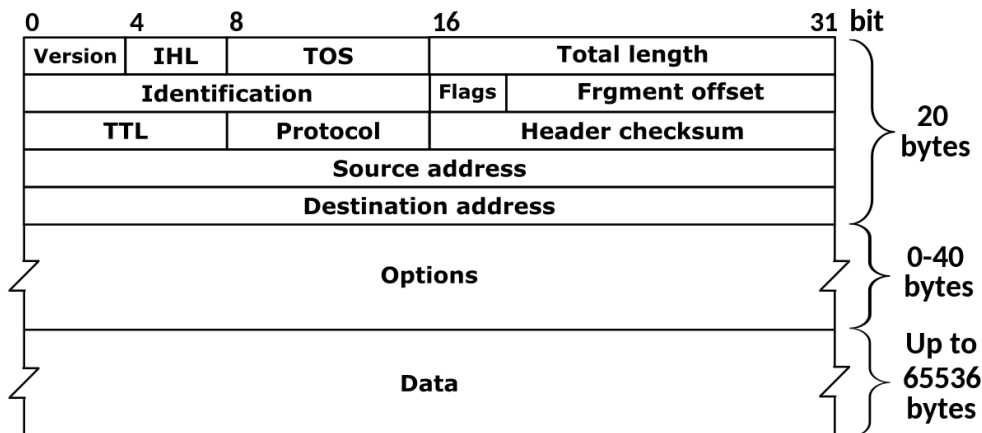
Binary	11000000	10101000	00001110	00010111
Dotted decimal	192	168	14	23

11000000 10101000 00001110 00010111 = 192.168.14.23

Possible values range from 000.000.000.000 through 255.255.255.255, although many possible addresses are disallowed or reserved for specific purposes.

The address combines network identification and device identification data. The network prefix is anywhere from eight to 31 bits, and the remainder identify the device on the network. Steady, rapid growth in the number of internet-connected devices has led to the looming exhaustion of the IPv4 address list, one of several reasons for the development of IPv6.





- IHL indicates the header length.
- Packets use Type of Service to request special treatment (e.g. being put at the front of any queues).
- The identification, flags, and fragmentation offset fields keep track of each fragment when an IP packet is split up into smaller packets.
- The time to live is set when the IP packet is sent off, and the number is reduced by 1 each time the packet goes through a router. When the time to live gets to 0, the packet is deleted. This stops packets from circulating in a loop.

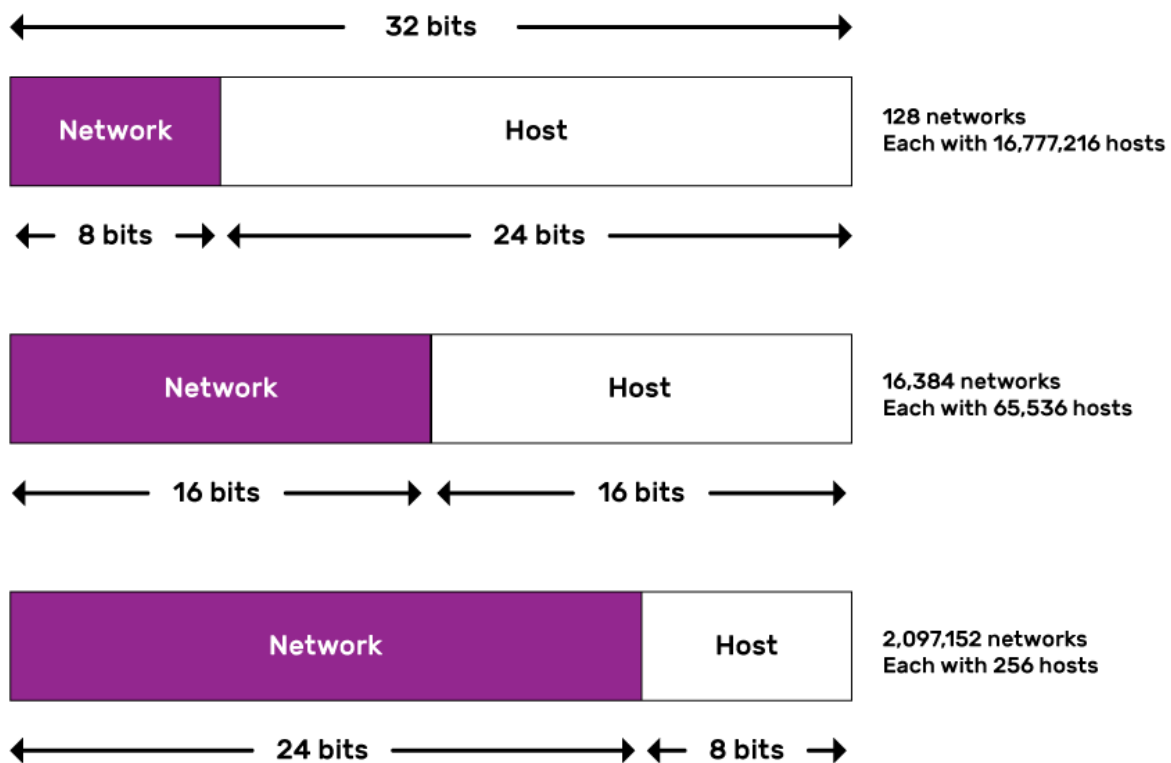
The first value of an IP address indicates the address is split into network bits and host bits. Network bits are the initial bits in an IP address. They identify a particular network. The remaining bits are called host bits and identify a device (or 'host') in that network.

There are three classes of addresses identified by this initial value:

Class A uses 8 network bits and 24 host bits

Class B uses 16 network bits and 16 host bits

Class C uses 24 network bits and 8 host bits



The more network bits, the more networks can be identified. But more network bits mean fewer host bits, meaning smaller networks.

Within each IP address class, a specific range is allocated for private IP addresses. That means these addresses can be used many times over on different networks, as long as these networks do not become connected. In class C networks, all networks starting with 192.168. are private.

7.5 IPv6 Address

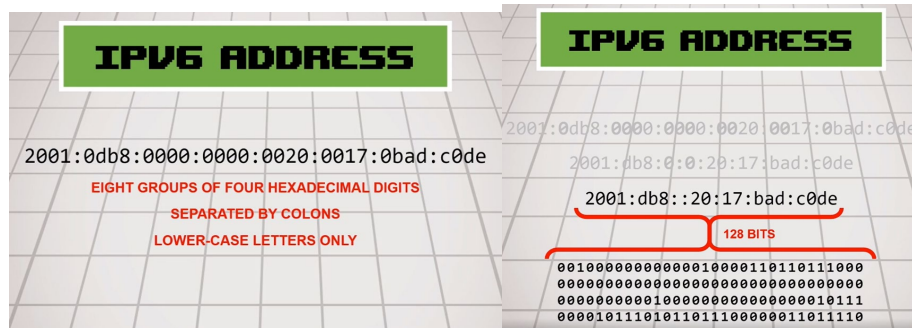
The current version of IP is version 4 (IPv4). The 32 bits in IPv4 give 4,294,967,296 addresses, but we are running out.

Therefore, IP version 6 (IPv6) has been developed, with 128 bits allocated for each address. This translates to a colossal 3.4×10^{38} (340,282,366,920,938,463,463,374,607,431,768,211,456) addresses.

IPv6 has been designed to coexist with IPv4, so both systems can run in parallel.

An IPv6 address consists of 128 bits, with the first 64 reserved for network identification and the second 64 dedicated to identifying a device on the network.

IPv6 address is written as eight sets of four hexadecimal digits separated by colons - for example, FEDC:BA98:7654:3210:0123:4567:89AB:CDEF.



0	3	11	15	23	31
Version	Traffic class		Flow label		
Payload length			Next header	Hop limit	
Source address					
Destination address					

7.6 MAC vs IPv4 vs IPv6

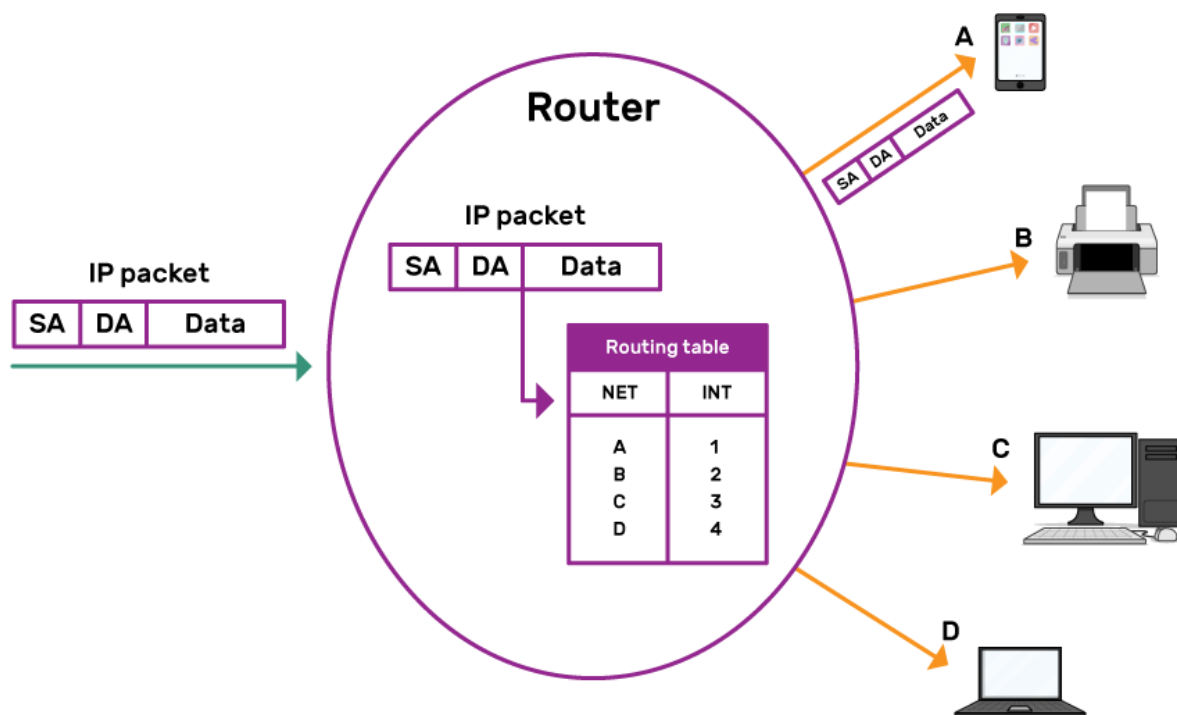
	MAC	IPv4	IPv6
Function	Used by network switches to direct data within the same network	Used by routers to direct data across different networks	Used by routers to direct data across different networks
Length of Address	48 bits	32 bits	128 bits
Address Format	Six groups of two hexadecimal digits separated by colons or dashes	Four denary numbers between 0 and 255 separated by dots	Eight groups of four hexadecimal digits separated by colons Can be shortened by omitting leading zeros as well as a section of consecutive zeroes
Permanent	Yes	No	No

For a more detailed discussion on the IP vs MAC, refer to this:

https://askleo.com/whats_the_difference_between_a_mac_address_and_an_ip_address/

7.7 IP Packets, Routers and Routing

Routers got their name because they route IP Packets across networks: a router device connects different links. It examines IP packet headers, looking at the destination address and consulting a routing table of known networks. The table indicates which connections to send the packet on as the next link. The connections on a router are called interfaces: through the router, they interface between different link types.

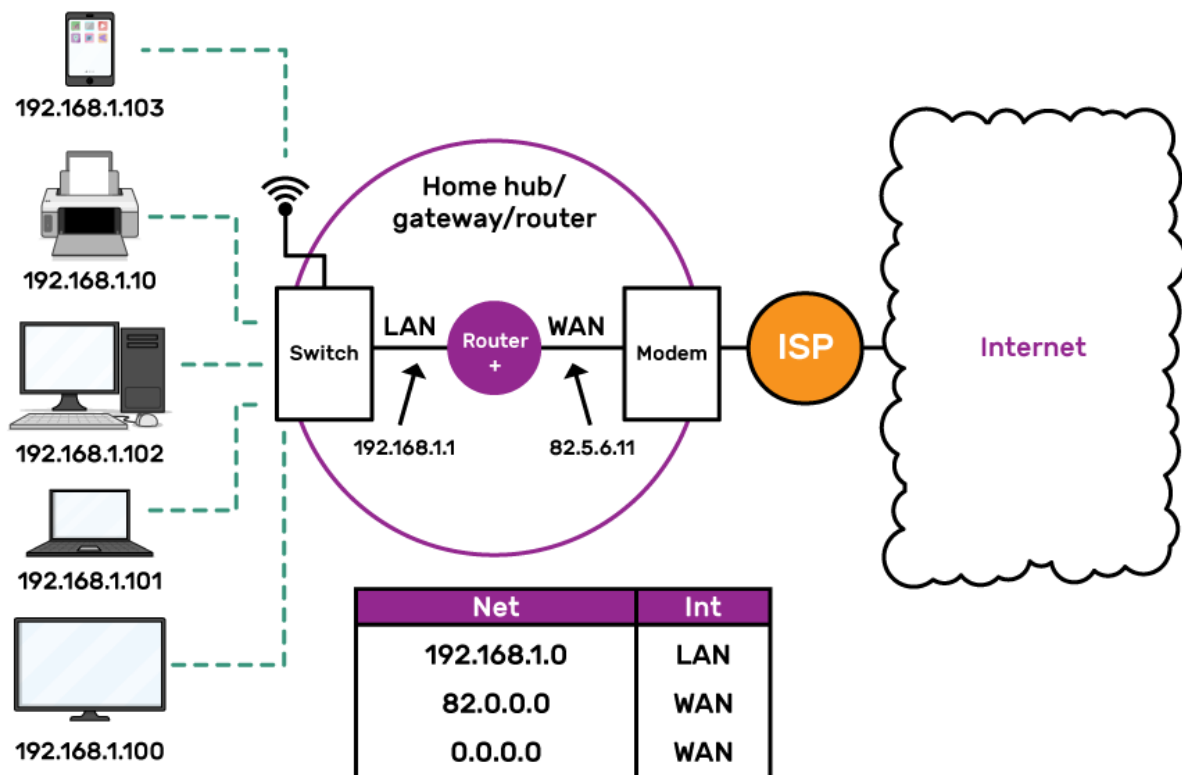


In the diagram above, an IP packet arrives at a router, which compares the packet's destination address with the networks in the routing table. The packet is identified as belonging to network A, so the router routes it out of interface 1.

Often, routers learn how to reach distant networks by exchanging information with their neighbouring routers and building their own routing tables. This exchange of information to learn about other networks is called a routing protocol. You can find [more information about routing protocols](#) on Wikipedia.

7.7.1 Example: Home or Small business network

A home or small-business network usually has a single LAN, which a router connects to an ISP.



- Each of the network's devices, including the actual interface on the router for the LAN, will have a 192.168.1.X address. The internal LAN's network address is therefore 192.168.1.0.
- The Router's WAN interface has the address 82.5.6.11, which is part of the ISP's 82.0.0.0 network.

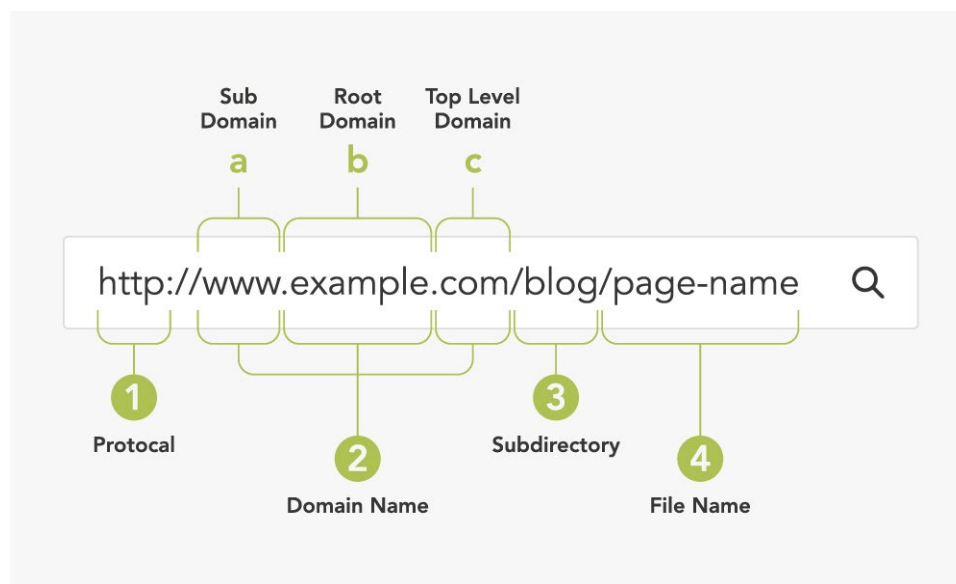
So the router is connected to two different IP networks, which are automatically placed into the routing table. The default network, 0.0.0.0, gets configured so it sends data out via the WAN interface and onto the next router.

For more information on 0.0.0.0, refer to this :

<https://forum.huawei.com/enterprise/en/what-it-means-when-you-see-the-0-0-0-0-ip-address/thread/809389-861>

8 Domain Name Server

The Domain Name System (DNS) is the primary way users refer to resources on the Internet. While computers talk to each other using MAC addresses and IP addresses, people prefer to use names and words instead



Source: <https://goup.la/technical-seo/learn/best-domain-strategy-for-international-expansion/>

DNS associates various information with domain names. It serves as the phone book for the Internet by translating human-readable computer hostnames (eg: `www.openbookproject.net`) into IP addresses (eg: `152.19.134.41`) that networking equipment needs to deliver information. DNS also stores other information such as the list of mail servers.

DNS makes it possible to assign Internet names to organisations independent of the physical routing hierarchy represented by the numerical IP address. It distributes the responsibility for assigning domain names and mapping them to IP networks by allowing an authoritative name server for each domain to keep track of its changes thus, avoiding the need for a central register to be continually consulted and updated.

8.1 How DNS works

Suppose you have just created a small company that has several servers, including a public Web

server that describes your company's products and services, a mail server from which your employees obtain their e-mail messages, and a DNS server. Naturally, you would like the entire world to be able to visit your website to learn about your products and services. Moreover, you would like your employees to be able to send and receive emails to potential customers throughout the world.

To meet these goals, you first need to obtain Internet connectivity, which is done by contracting with

and connecting to a local Internet service provider (ISP). Your company will have a gateway router, which will be connected to a router in your local ISP. This connection might be a DSL connection through the existing telephone infrastructure, a leased line to the ISP's router, or other access solutions. Your local ISP will also provide you with an IP address range.

Once you have your physical connectivity and your IP address range, you will assign one of the IP addresses (in your address range) to your Web server, one to your mail server, one to your DNS server, one to your gateway router, and other IP addresses to other servers and networking devices in your company's network.

In addition to contracting with an ISP, you will also need to contract with an Internet registrar to obtain a domain name for your company. Your company must also obtain a presence in the DNS system. Because outsiders will want to contact your DNS server to obtain the IP addresses of your servers, you will also need to provide your registrar with the IP address of your DNS server. Your registrar will then put an entry for your DNS server (domain name and corresponding IP address) in the .com top-level domain (TLD) servers.

After this step is completed, any user who knows your domain name (e.g., MyCompany.com) will be able to obtain the IP address of your DNS server via the DNS system. For people to discover the IP addresses of your Web server, you will need to include entries that map the hostname of your Web server (e.g., www.MyCompany.com) to its IP address, in your DNS server. You will want to have similar entries for other publicly available servers in your company, including your mail server. In this manner, if Alice wants to browse your Web server, the DNS system will contact your DNS server, find the IP address of your Web server, and give it to Alice. Alice can then establish a TCP connection directly with your Web server. See the next section for a step-by-step explanation.

However, there still remains one other necessary and crucial step to allow outsiders from around the world to access your Web server. Consider what happens when Alice, who knows the IP address of

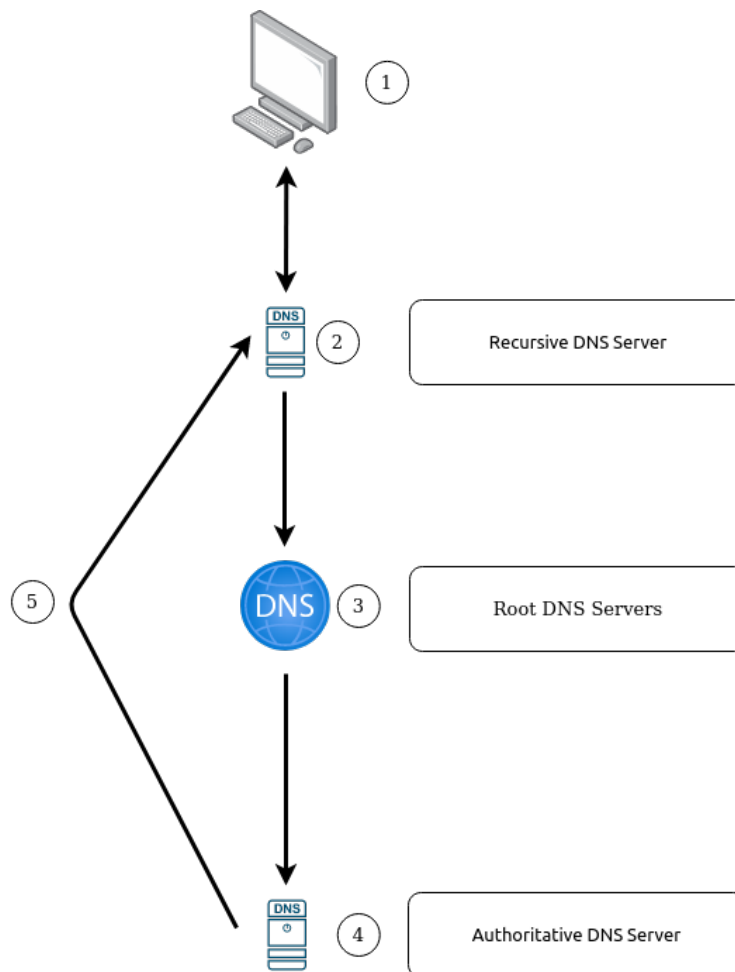
your Web server sends an IP datagram to that IP address. This datagram will be routed through the Internet, visiting a series of routers in many different Autonomous Systems (ASs)¹, and eventually reach your Web server. When any one of the routers receives the datagram, it is going to look for an entry in its forwarding table to determine on which outgoing port it should forward the datagram.

Therefore, each of the routers needs to know about the existence of your company's prefix (i.e., an address range). When your company contracts with a local ISP and gets assigned a prefix, your local ISP will use Border Gateway Protocol (BGP)² to advertise your prefix to the ISPs to which it connects. Those ISPs will then, in turn, use BGP to propagate the advertisement. Eventually, all Internet routers will know about your prefix (or about some aggregate that includes your prefix) and thus be able to appropriately forward datagrams destined to your Web and mail servers.

¹ An autonomous system (AS) is a network or a collection of networks that are all managed and supervised by a single entity or organisation. An AS has many different subnetworks with combined routing logic and common routing policies. Each subnetwork is assigned a globally unique 16 digit identification number (known as the AS number or ASN) by the Internet Assigned Numbers Authority (IANA).

² Border Gateway Protocol (BGP) is a standardised exterior gateway protocol designed to exchange routing and reachability information among autonomous systems (AS) on the Internet.

8.2 Process of DNS request



Source: tryhackme.com, DNS in Detail course

1. When you request a domain name, your computer first checks its local cache to see if you've previously looked up the address recently; if not, a request to your Recursive DNS Server will be made.
2. A Recursive DNS Server is usually provided by your ISP, but you can also choose your own. This server also has a local cache of recently looked up domain names. If a result is found locally, this is sent back to your computer, and your request ends here (this is common for popular and heavily requested services such as Google, Facebook, Twitter). If the request cannot be found locally, a journey begins to find the correct answer, starting with the internet's root DNS servers.

3. The root servers act as the DNS backbone of the internet; their job is to redirect you to the correct Top Level Domain Server, depending on your request. If, for example, you request `www.tryhackme.com`, the root server will recognise the Top Level Domain of `.com` and refer you to the correct TLD server that deals with `.com` addresses.
4. The TLD server holds records for where to find the authoritative server to answer the DNS request. The authoritative server is often also known as the nameserver for the domain. For example, the name server for `tryhackme.com` is `kip.ns.cloudflare.com` and `uma.ns.cloudflare.com`. You'll often find multiple nameservers for a domain name to act as a backup in case one goes down. In summary, this authoritative DNS server is responsible for storing the DNS records for a particular domain name and where any updates to your domain name DNS records would be made.
5. Depending on the record type, the DNS record is then sent back to the Recursive DNS Server, where a local copy will be cached for future requests and then relayed back to the original client that made the request. DNS records all come with a TTL (Time To Live) value. This value is a number represented in seconds that the response should be saved for locally until you have to look it up again. Caching saves on having to make a DNS request every time you communicate with a server.

Try this for a practical experience: <https://tryhackme.com/room/dnsindetail>

Referenced Readings

1. MOE Student Learning Space (SLS) learning package “MAC and IP Addresses”
2. Kurose, J., & Ross, K. (2016). Computer Networking: A TopDown Approach (7th ed.). Pearson.
3. Langfield, S., & Duddell, D. (2015). Cambridge International AS and A Level Computer Science Coursebook. Cambridge University Press.

4. Geeks forGeeks . (2021, June 28). Computer Network Tutorials . Retrieved February, 2022, from <https://www.geeksforgeeks.org/computer-network-tutorials/?ref=lbp>
5. (IT) Information Technology Certifications | CompTIA IT Certifications . (n.d.). Default. <https://www.comptia.org/resources>
6. Cisco - Networking, Cloud, and Cybersecurity Solutions . (2022, January 28). Cisco. <https://www.cisco.com/>
7. Techtarget.com. Searchnetworking. Retrieved from <https://www.techtarget.com/searchnetworking/>
8. DNS in detail. TryHackMe. (n.d.). Retrieved from <https://tryhackme.com/room/dnsindetail>
9. <https://tryhackme.com/room/introtolan>
10. <https://adacomputerscience.org/>
11. <https://www.futurelearn.com/courses/introduction-to-networking>

Chapter 21: Fundamentals of Computer Networks (Part 3)

Syllabus Learning Outcomes

4.1 Fundamentals of Computer Networks

Understand computer network technology

4.1.5 Explain client-server architecture.

4.1.6 Implement an iterative server with socket programming. Given the server code, students should be able to implement the client code for a given scenario, and vice-versa, e.g. for a tic-tac-toe game.

1 Introduction

When we use the internet, we often use a web browser such as Internet Explorer, Firefox, or Chrome. The web browser is an example of an application, meaning a program used by the end user. It is also referred to as a client because it starts the exchange of information. It requests web pages from a web server, which is a different program running on another device. A server serves the information requested by the client. This is the client-server model, and most programs that access the internet follow it.

Applications such as email, file transfer, media streaming, and online games all communicate via the internet with their respective servers.

In peer-to-peer networking, each device can act as a client requesting files from another device, as a server sharing files with another device, or as both simultaneously.

Messages sent between the application (i.e. the client) and the server use a specific protocol designed for the application. The application layer provides these protocols.

2 Computer Network Architecture: Client and Server Processes

A network application consists of pairs of processes that send messages to each other over a network. For example, in the Web application, a client browser process

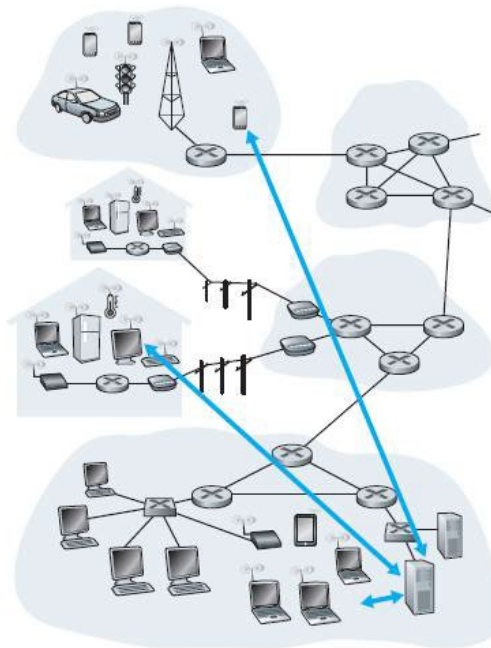
exchanges messages with a Web server process. In a Peer-to-peer (P2P) file-sharing system, a file is transferred from a process in one peer to a process in another peer.

For each pair of communicating processes, we typically label one of the two processes as the client and the other process as the server. With the Web, a browser is a client process and a Web server is a server process. With P2P file sharing, the peer that is downloading the file is labelled as the client, and the peer that is uploading the file is labelled as the server. As you would have observed, a process can be both a client and a server. For example, a process in a P2P file-sharing system can both upload and download files.

In the context of a communication session between a pair of processes, the process that initiates the communication is labelled as the client. The process that waits to be contacted to begin the session is the server.

In the Web, a browser process initialises contact with a Web server process. Hence, the browser process is the client and the Web server process is the server. In P2P file sharing, when Peer A asks Peer B to send a specific file, Peer A is the client and Peer B is the server in the context of this specific communication session. Sometimes, we also use the terminology “client side and server side of an application.”

2.1 Client-server Architecture



Source: Kurose, J., & Ross, K. (2016). *Computer Networking: A Top-Down Approach* (7th ed.). Pearson.

In a client-server architecture, there is always one host, called the server, which services requests from many other hosts, called clients. A classic example is the Web application for which an always-on Web server services requests from browsers running on client hosts. When a Web server receives a request for an object from a client host, it responds by sending the requested object to the client host. The clients, i.e. the web browsers in this case, do not directly communicate with each other.

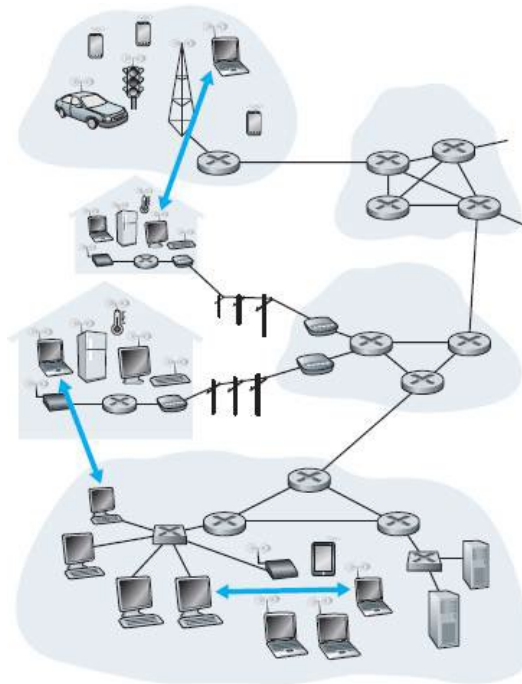
Another characteristic of the client-server architecture is that the server has a fixed, well-known address called an IP address. Because the server has a fixed, well-known address, and because the server is always on, a client can always contact the server by sending a packet to the server's IP address. Some of the better-known applications with a client-server architecture include the Web, FTP, Telnet and email.

Often in a client-server application, a single-server host is incapable of keeping up with all the requests from clients. For this reason, a data centre, housing a large number of hosts, is often used to create a powerful virtual server. The most popular Internet services such as search engines (e.g. Google, Bing, Baidu), Internet commerce (e.g. Amazon, eBay, Alibaba), Web-based email (e.g. Gmail and Yahoo Mail), social networking (e.g. Facebook, Instagram, Twitter, and WeChat) employ one or more data centres. Google, for example, has 30 to 50 data centres distributed around the world,

which collectively handle search, YouTube, Gmail, and other services. A data centre can have hundreds of thousands of servers, which must be powered and maintained.

Additionally, the service providers must pay recurring interconnection and bandwidth costs for sending data from their data centres.

2.2 Peer-to-peer (P2P) Architecture



Source: Kurose, J., & Ross, K. (2016). *Computer Networking: A Top-Down Approach* (7th ed.). Pearson.

In a P2P architecture, there is minimal or no reliance on dedicated servers in data centres. Instead, the application exploits direct communication between pairs of intermittently connected hosts, called peers. The peers are not owned by the service provider, but are desktops and laptops controlled by users, residing in homes, universities, and offices. Because the peers communicate without passing through a dedicated server, the architecture is called peer-to-peer.

Many of today's most popular and traffic-intensive applications are based on P2P architectures. These applications include file sharing (e.g. BitTorrent), peer-assisted

download acceleration (e.g. Xunlei), and Internet telephony and video conference (e.g. Skype).

2.3 Hybrid Architecture

Some applications have hybrid architectures, combining both client-server and P2P elements. For example, for many instant messaging applications, servers are used to track the IP addresses of users, but user-to-user messages are sent directly between user hosts, without passing through intermediate servers.

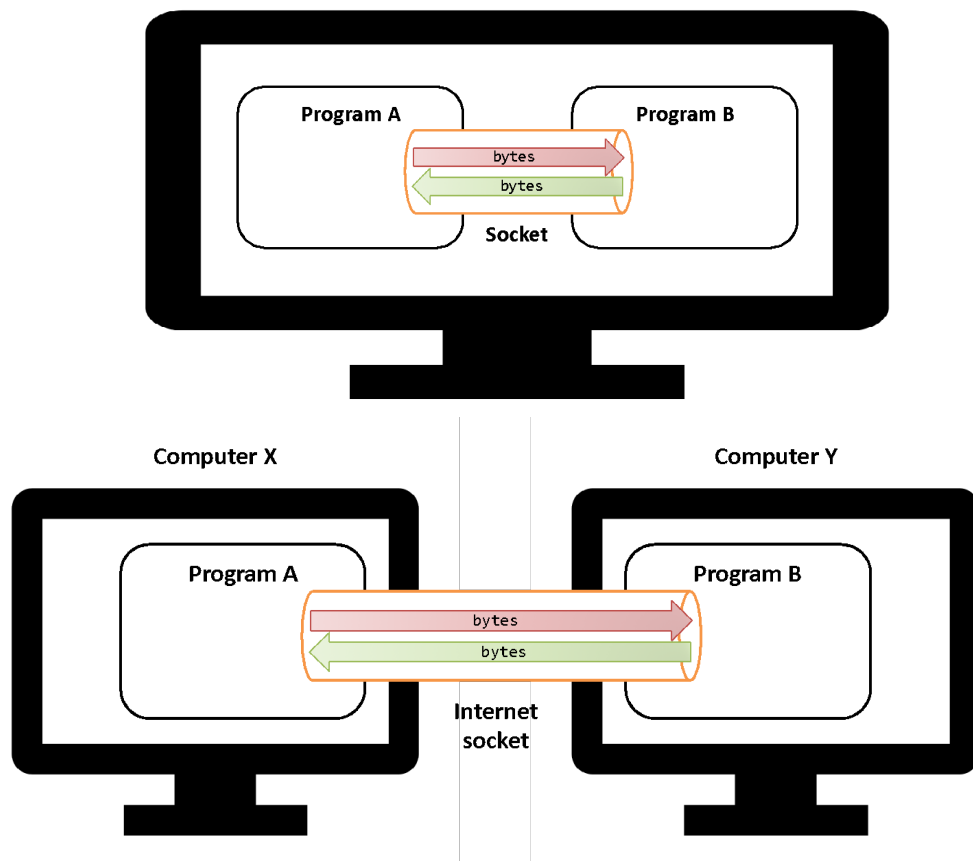
One of the most compelling features of P2P architectures is their self-scalability. For example, in a P2P file-sharing application, although each peer generates workload by requesting files, each peer also adds service capacity to the system by distributing files to other peers.

P2P architectures are also cost-effective, since they normally don't require significant server infrastructure and server bandwidth in contrast with client-server designs with data centres. However, P2P applications face challenges of security, performance, and reliability due to their highly decentralised structure.

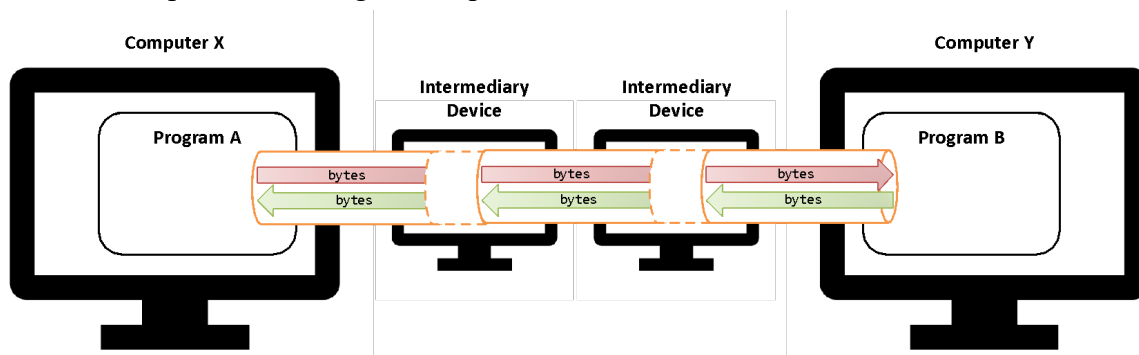
3 The Interface Between the Process and the Computer Network

As noted above, most applications consist of pairs of communicating processes, with the two processes in each pair sending messages to each other. Any message sent from one process to another must go through the underlying network. A process sends messages into and receives messages from, the network through a software interface called a socket. You can picture a socket connection as a pipe between two running programs. The pipe is bidirectional and can carry data (represented by bytes) in both directions.

There are many kinds of sockets, but the kind that is most often discussed is called an Internet socket. Internally, Internet sockets deliver data using the same Transmission Control Protocol and Internet Protocol suite (TCP/IP) that is used to transmit data over the Internet. This means that Internet sockets can deliver data between any two programs, programs that are running on different computers, as long as the two computers can access each other over the network.



For simplicity, we illustrate an Internet socket as a pipe that is only attached to the two computers. In reality, however, data that is transmitted through an Internet socket may pass through multiple devices before reaching its destination. You should be aware that any of these devices can steal or modify the data that passes through a socket unless you encrypt the data first. A more accurate illustration of a socket that shows how the data passes through multiple devices is shown below:

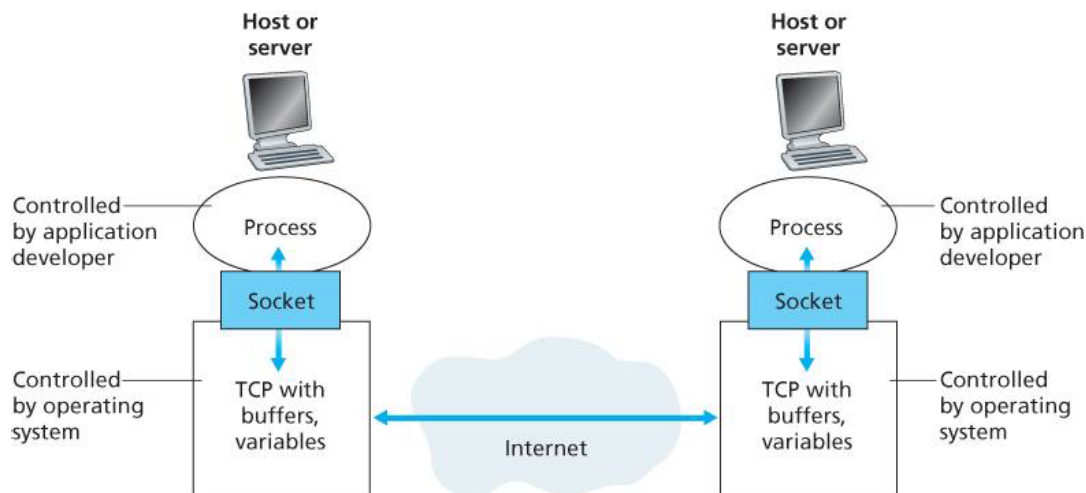


3.1 Socket Communication

The figure below illustrates socket communication between two processes that communicate over the Internet, with the TCP protocol as the underlying transport protocol.

Why do we need a protocol? As networks can become congested, we cannot assume that data sent over Internet sockets will be transmitted instantaneously. For instance, a program may receive only the first half of a message before the second half arrives some time later. To avoid working with incomplete data, we will need to define a protocol so that the start and end of messages can be detected unambiguously.

As shown in the figure, a socket is the interface between the application layer and the transport layer within a host. It is also referred to as the Application Programming Interface (API) between the application and the network, since the socket is the programming interface with which network applications are built. The application developer has control of everything on the application layer side of the socket but has little control of the transport-layer side of the socket.



Source: Kurose, J., & Ross, K. (2016). *Computer Networking: A Top-Down Approach* (7th ed.). Pearson.

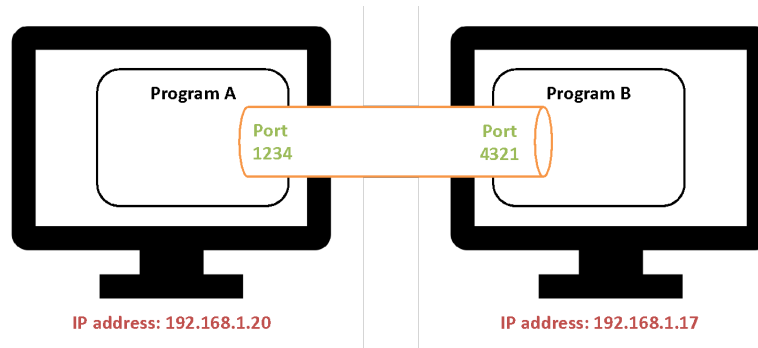
The only control that the application developer has on the transport-layer side is:

- the choice of transport protocol, if such provision is available and,
- the ability to fix a few transport-layer parameters such as maximum buffer and maximum segment sizes.

In order for a process running on one host to send packets to a process running on another

host, the receiving process needs to have an address. To identify the receiving process, two pieces of information need to be specified:

- The IP address of the host and,
- an identifier that specifies the receiving process in the destination host, i.e. port number.



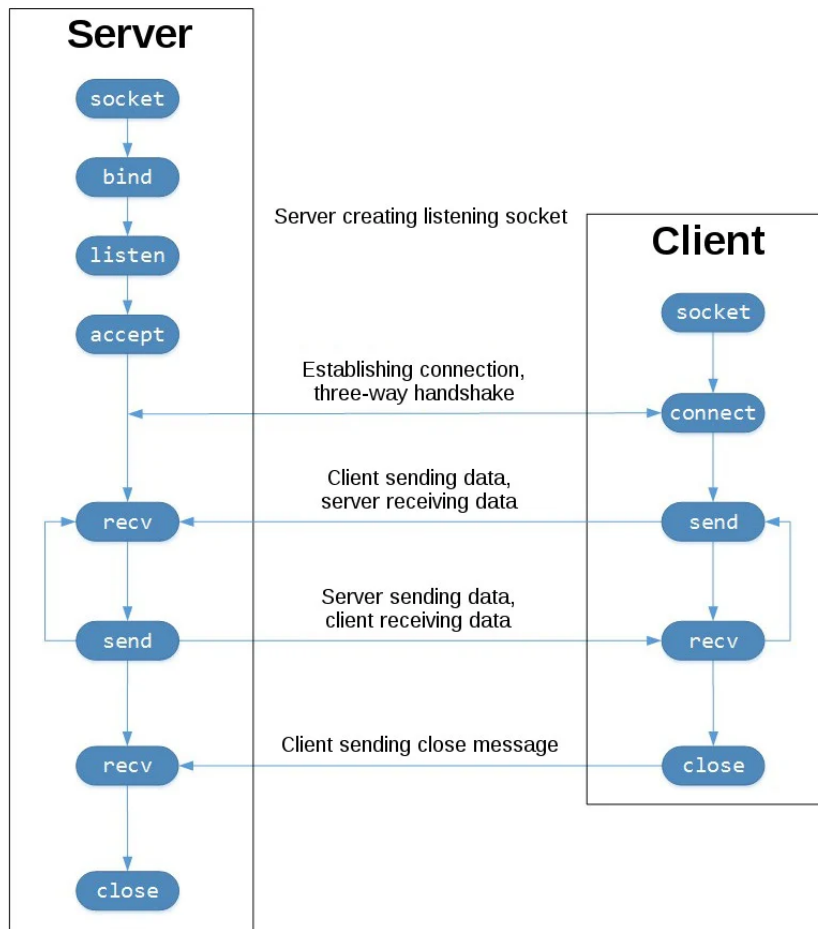
The IP address uniquely identifies the host. The sending process must also identify the receiving process, i.e., the receiving socket running in the host. This information is needed because a host could be running many network applications. A destination port number serves this purpose. On each device, port numbers are used to distinguish between attached sockets. The device also keeps track of which program is associated with each port and which port numbers are still available for use by new sockets.

Port numbers can range from 0 to 65,535. However, the first 1,024 port numbers are reserved for specific kinds of programs and should not be used for other purposes. For example, a Web server is identified by port number 80. A mail server process using the SMTP protocol is identified by port number 25. A list of well-known port numbers for all Internet standard protocols can be found at www.iana.org.

3.2 TCP Socket API

Creating a socket connection is a multi-step process that requires one program to be the server and another program to be the client. The server's IP address and port number for accepting connections must also be known ahead of time by the client.

The figure below shows an overview of the TCP Socket API.

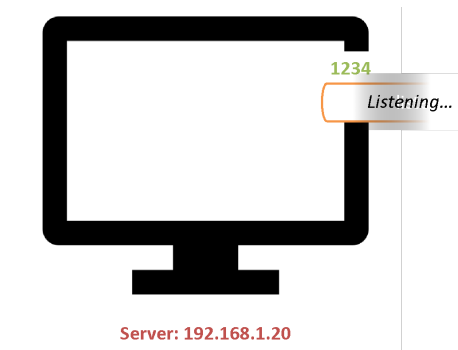


1

Source: https://commons.wikimedia.org/wiki/File:InternetSocketBasicDiagram_zhtw.png

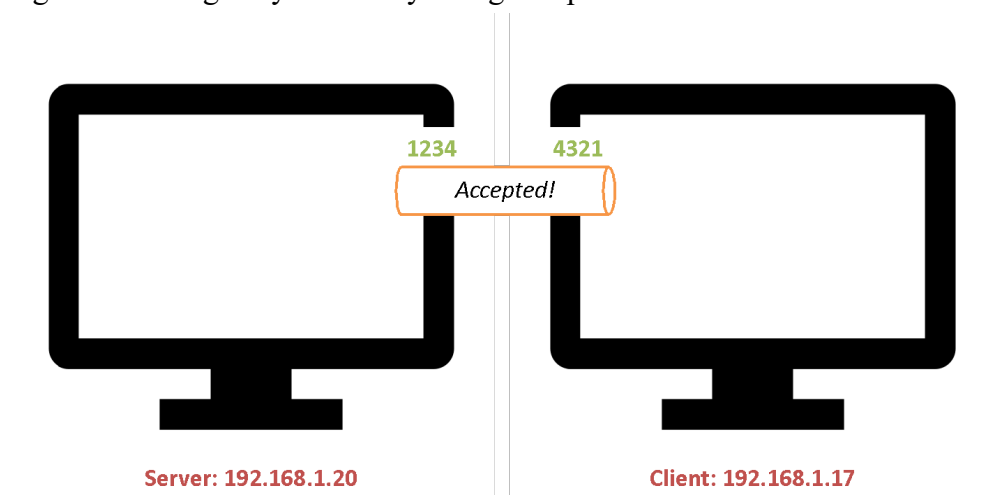
First, the server creates a passive socket, binds it to the pre-chosen port number and listens for an incoming connection. (A passive socket is not connected and merely waits for an incoming connection.)

¹ For a quick understanding of the TCP 3-way handshake, refer to <https://www.geeksforgeeks.org/tcp-3-way-handshake-process/>

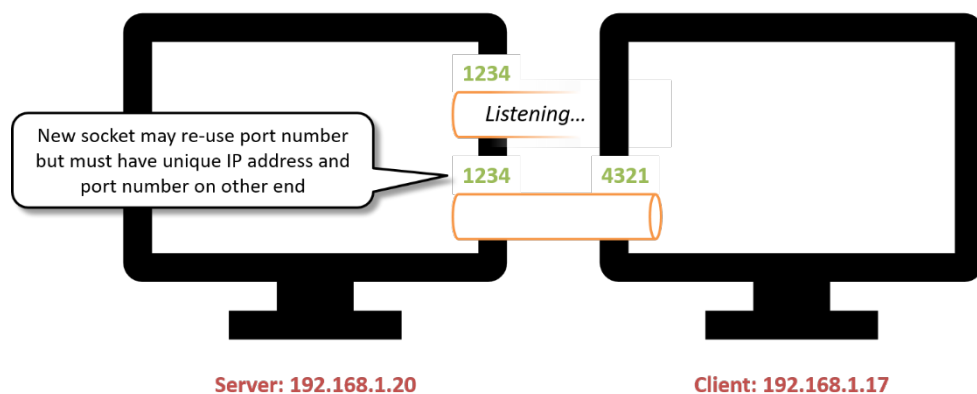


Next, the client initiates a connection request using the server's IP address and port number. If no server is listening on the chosen port, the connection will be refused.

On the other hand, if the connection request reaches an IP address and port number that a server is listening on, the server accepts and creates a new socket for the requesting client using a dynamically assigned port number.



The passive socket goes back to listening for new connections while the client and server can now exchange data using the newly-created socket.



Note that the newly-created socket is symmetrical: data sent on one end is received on the other end and vice versa. Once a socket is established, it can send data both from the client to the server and from the server to the client.

3.3 Python's Socket Module

You can create and manage sockets in Python by importing the `socket` module and creating socket objects. The methods of the `socket` class are summarised below:

Methods	Description
<code>bind((host, port))</code>	Binds socket object to the given address tuple (host, port), where host is an IPv4 address and port is a port number
<code>listen()</code>	Enables socket to listen for incoming connections from clients
<code>accept()</code>	Waits for an incoming connection and returns a tuple containing a new socket object for the connection and an address tuple (host, port), where host is the IPv4 address of the connected client and port is its port number
<code>connect((host, port))</code>	Initiates a connection to the given address tuple (host, port), where host is the IPv4 address of the server and port is its port number
<code>recv(max_bytes)</code>	Receives and returns up to the given number of bytes from the socket

sendall(bytes)	Sends the given bytes to the socket

Referenced Readings

1. Kurose, J., & Ross, K. (2016). Computer Networking: A TopDown Approach (7th ed.). Pearson.
2. MOE CPDD Starter Kit: Socket Programming
3. GeeksforGeeks. Socket Programming in Python.
https://www.geeksforgeeks.org/socket_programming-python/
4. Real Python. Socket Programming in Python (Guide). Realpython.Com.
<https://realpython.com/python-sockets/>
5. Chong, Stephen (2011). Network Programming with Sockets, CS61, Lecture 23, Harvard University, School of Engineering and Applied Sciences.

Chapter 22: Network Security

Syllabus Learning Outcomes	1
1. Computer Security	2
1.1 Network Security.....	3
2. Malware	5
3. Denial of Service (DoS)	8
4. Packet Sniffer	9
5. IP Spoofing	10
6. Confidentiality	11
6.1 Encryption.....	11
6.2 File Permissions.....	13
7. Message integrity	14
7.1 Cryptographic Hash Functions.....	15
7.2 Digital Signature.....	17
8. End-point authentication	19
8.1 Passwords	19
8.2 Security Tokens (Two-Factor Authentication)	21
8.3 Biometrics	22
9. Operational Security	22
9.1 Firewalls.....	23
9.2 Intrusion Detection System (IDS) and Intrusion Prevention System (IPS)	24
Referenced Readings	26

Syllabus Learning Outcomes

4.3 Network Security

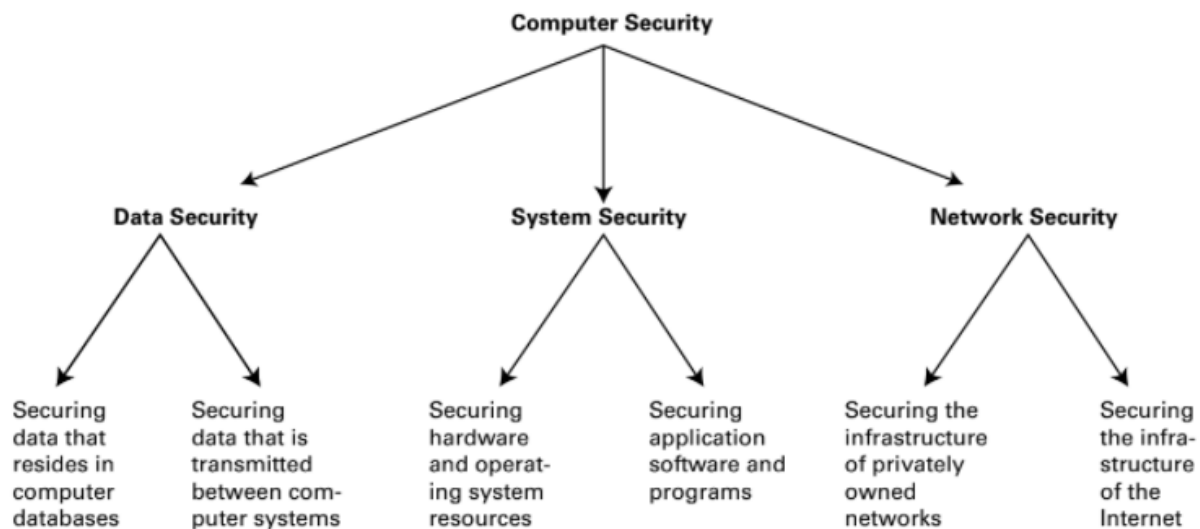
Understand computer network security in terms of threats, protection mechanisms, and secure access.

- 4.3.1 Understand how malware (e.g. worms and viruses) and denial of service (DOS) attacks can compromise computer systems.
- 4.3.2 Understand how firewall (filtering function), intrusion detection system (IDS), and intrusion prevention system (IPS) can be used to restrict network access and their limitations.

4.3.3 Understand how encryption, digital signature, and authentication can ensure the security of network applications.

1. Computer Security

The three kinds of computer security differ according to the range of concerns and vulnerabilities.



Three kinds of computer security

Source: Tavani, H. T. (2010). *Ethics and Technology: Controversies, Questions, and Strategies for Ethical Computing* (3rd ed.). Wiley.

Richard Epstein (2007) suggests that security concerns affecting computers and cyber-technology can be viewed in terms of three key elements:

- confidentiality,
- integrity,
- accessibility.

Whereas confidentiality is “about preventing unauthorized persons from gaining access to unauthorized information,” integrity, in computer security contexts, is about “preventing an attacker from modifying data.” In Epstein’s scheme, accessibility has to do with “making sure that resources are available for authorized users.” In this context, confidentiality is a set of rules that limits access to information, integrity is the assurance that the information is trustworthy and accurate, and availability is a guarantee of reliable access to the information by authorized people.

These are also known as the CIA triad, a model designed to guide policies for information security within an organization. The model is also sometimes referred to as the AIC triad (availability, integrity and confidentiality) to avoid confusion with the Central Intelligence Agency. The elements of the triad are considered the three most crucial components of security.

Peter Neumann (2004) notes that, in addition to providing desired confidentiality, integrity, and accessibility, cybersecurity aims at preventing “misuse, accidents, and malfunctions” with respect to computer systems. Neumann also notes, however, that cybersecurity can be a “double-edged sword”; for example, it can be used to protect privacy, but it can also be used to undermine “freedom of access” to information for users.

1.1 Network Security

Network security is concerned with securing computer networks—i.e., from privately owned computer networks (such as LANs and WANs) to the Internet itself—against various kinds of attacks. The Internet’s infrastructure has been the victim of several attacks. These attacks have ranged from programs launched by individuals with malicious intentions to individuals who claimed their intentions were benign. In many cases, these attacks have severely disrupted activities on segments of the Internet. In a few cases, they have also rendered the Internet virtually inoperable.

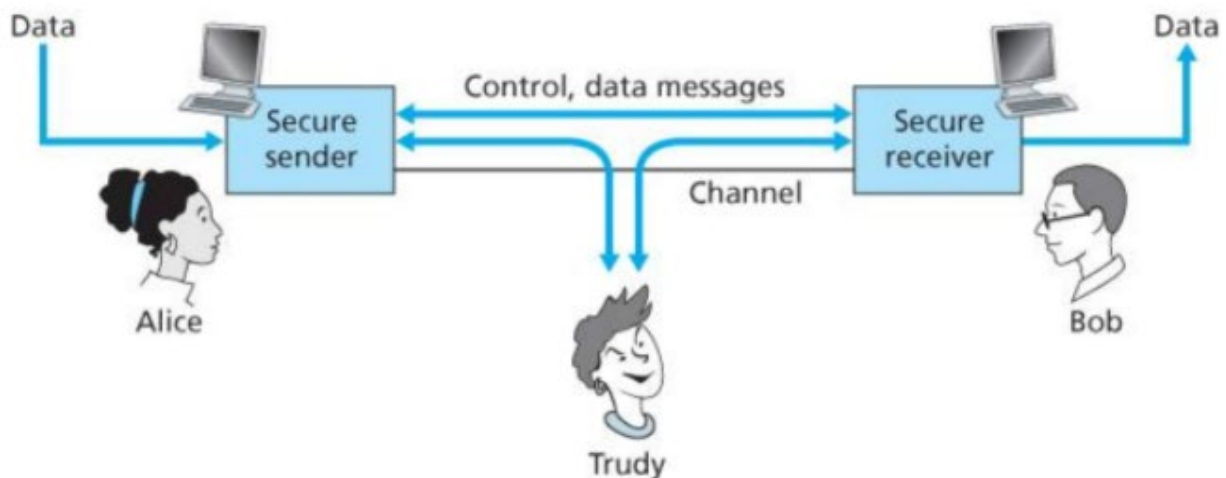
We should note that it is not always easy to determine whether a major computer network disruption is the result of the work of malicious individuals who launch various kinds of malware or is due to the failure of some aspect of the network infrastructure itself. For example, a significant power outage experienced by the AT&T long-distance telephone service in 1990 was attributed to a software glitch in the system’s programming code that caused the network to crash. However, some have questioned the official explanation given by AT&T, suggesting instead that the crash may have resulted from an attack involving malware.

Because many nations now depend on secure cyberspace for their physical infrastructures, including power grids, there has been increased concern over threats from international hacking groups, including governments and state-sponsored organisations.

Having established what we mean by network security, let's next consider exactly what information an intruder may have access to, and what actions can be taken by the intruder.

An intruder can potentially perform

- eavesdropping—sniffing and recording control and data messages on the channel.
- modification, insertion, or deletion of messages or message content.



Sender, receiver, and intruder (Alice, Bob and Trudy)

Source: Kurose, J., & Ross, K. (2016). *Computer Networking: A Top-Down Approach* (7th ed.). Pearson.

We can identify the following desirable properties of secure communication.

1.1.1 Confidentiality

Only the sender and intended receiver should be able to understand the contents of the transmitted message. Because eavesdroppers may intercept the message, this necessarily requires that the message be somehow encrypted so that an intercepted message cannot be understood by an interceptor. This aspect of confidentiality is probably the most commonly perceived meaning of the term secure communication.

1.1.2 Message integrity

Alice and Bob want to ensure that the content of their communication is not altered, either maliciously or by accident, in transit.

1.1.3 End-point authentication

Both the sender and receiver should be able to confirm the identity of the other party involved in the communication—to confirm that the other party is indeed who or what they claim to be. Face-to-face human communication solves this problem easily by visual recognition. When communicating entities exchange messages over a medium where they cannot see the other party, authentication is not so simple. When a user wants to access an inbox, how does the mail server verify that the user is the person he or she claims to be?

1.1.4 Operational security

Almost all organizations (companies, universities, and so on) today have networks that are attached to the public Internet. These networks therefore can potentially be compromised. Attackers can attempt to deposit worms into the hosts in the network, obtain corporate secrets, map the internal network configurations, and launch DoS attacks. Operational devices such as firewalls and intrusion detection systems are used to counter attacks against an organization's network. A firewall sits between the organization's network and the public network, controlling packet access to and from the network. An intrusion detection system performs "deep packet inspection," alerting the network administrators about suspicious activity.

2. Malware

Malware (malicious software) is the general term for software that is specifically designed to disrupt, damage, or gain unauthorized access to a computer system. The site <https://threatmap.checkpoint.com/> allows us to visualise some types of live cyberattacks worldwide.

Malware	Description
Ransomware	A type of malware that blocks access to the victim's computer system until a certain amount of money, usually in bitcoin or other cryptocurrency is paid. The most recent category of malware is ransomware, which garnered headlines in 2016 and 2017 when

	ransomware infections encrypted the computer systems of major organizations and thousands of individual users around the globe.
Scareware	It is a program that attempts to frighten the victim into buying unnecessary software or providing their financial data. Scareware pops up on a user's desktop with flashing images or loud alarms, announcing that the computer has been infected. It usually urges the victim to quickly enter their credit card data and download a fake antivirus program.
Spyware	A hidden program that secretly collects personal information about users and sends the information to attackers without the user's knowledge, without causing data corruption or data loss. Spyware may record the websites the user visits, information about the user's computer system and vulnerabilities for a future attack, or the user's keystrokes. Spyware that records keystrokes is called a keylogger. Keyloggers steal credit card numbers, passwords, account numbers, and other sensitive data simply by logging what the user types.
Adware	Adware pushes unwanted advertisements at users and spyware secretly collects information about the user.
Fileless malware	Unlike traditional malware, fileless malware does not download code onto a computer, so there is no malware signature for a virus scanner to detect. Instead, fileless malware operates in the computer's memory and may evade detection by hiding in a trusted utility, productivity tool, or security application. An example is Operation RogueRobin, which was uncovered in July 2018. RogueRobin is spread through Microsoft Excel Web Query files that are attached to an email. It causes the computer to run PowerShell command scripts, providing an attacker access to the system. As PowerShell is a trusted part of the Microsoft platform, this attack typically does not trigger a security alert. Some fileless malware is also clickless, so a victim does not need to click on the file to activate it.
Cookies	A small piece of data used by websites to store personal information on a user's web browser.

	It is misused by attackers to collect personal information about users.
Pharming	The interception of requests sent from a computer to a legitimate website and redirection to a fake website to steal personal data or credit card details. The attacker can use the personal details to access the victims' bank account in the bank's actual website.
Phishing	The use of emails and fake websites that appear to be from reputable companies. It is used to steal personal information such as passwords and credit card numbers from users.
Spamming	The mass distribution of unwanted messages or advertising sent to email addresses collected from sources such as public mailing lists, social networking sites, company websites and blogs. Emails are usually easily sent to users and the emails sent are used to lure users to enter their personal information and steal their personal data.
Trojan horse	A computer program that pretends to be a harmless file or useful application. When the trojan horse is run, it does something harmful such as giving intruders unauthorised access to the computer instead.
Virus	A computer program that attaches itself to a normally harmless program and modifies it. When the modified program is run by a user, the virus attaches copies of itself to any other programs it can find, thus infecting it.
Worm	A standalone computer program that runs automatically and attempts to spread copies of itself over a network. Unlike a virus, it does not attach itself to a program and it spreads by exploiting a vulnerability in the infected system or through email as an attachment masquerading as a legitimate file. Worms consume bandwidth and overload web servers, harming the host network.

	A graduate student created the first worm (the Morris worm) in 1988 as an intellectual exercise. Unfortunately, it replicated itself quickly and soon spread across the internet.

3. Denial of Service (DoS)

A compromised host may also be enrolled in a network of thousands of similarly compromised devices, collectively known as a botnet, which the bad guys control and leverage for spam e-mail distribution or distributed denial-of-service attack against targeted hosts.

Denial-of-service (DoS) attacks are another broad class of security threats. As the name suggests, a DoS attack renders a network, host, or other pieces of infrastructure unusable by legitimate users. Web servers, e-mail servers, DNS servers, and institutional networks can all be subject to DoS attacks. Internet DoS attacks are extremely common, with thousands of DoS attacks occurring every year.. The site Digital Attack Map (<https://www.digitalattackmap.com/>) allows us to visualize the top daily DoS attacks worldwide.

Most Internet DoS attacks fall into one of three categories:

- **Vulnerability attack**

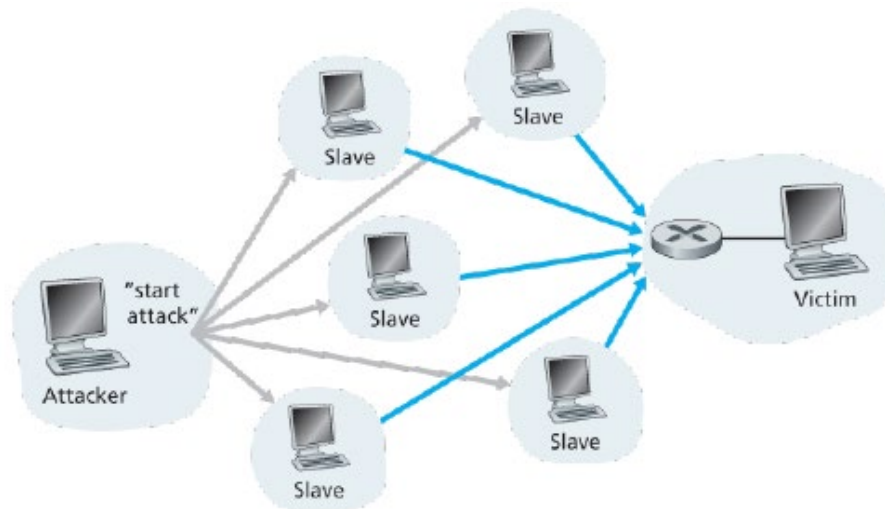
This involves sending a few well-crafted messages to a vulnerable application or operating system running on a targeted host. If the right sequence of packets is sent to a vulnerable application or operating system, the service can stop or, worse, the host can crash.

- **Bandwidth flooding**

The attacker sends a deluge of packets to the targeted host—so many packets that the target's access link becomes clogged, preventing legitimate packets from reaching the server.

- **Connection flooding**

The attacker establishes a large number of half-open or fully open TCP connections at the target host. The host can become so bogged down with these bogus connections that it stops accepting legitimate connections.



A distributed denial-of-service attack

Source: Kurose, J., & Ross, K. (2016). *Computer Networking: A Top-Down Approach* (7th ed.). Pearson.

Consider a server that has an access rate of R bps, then the attacker will need to send traffic at a rate of approximately R bps to cause damage. If R is very large, a single attack source may not be able to generate enough traffic to harm the server. Furthermore, if all the traffic emanates from a single source, an upstream router may be able to detect the attack and block all traffic from that source before the traffic gets near the server.

In a distributed DoS (DDoS) attack, illustrated above, the attacker controls multiple sources and has each source blast traffic at the target. With this approach, the aggregate traffic rate across all the controlled sources needs to be approximately R to cripple the service.

DDoS attacks leveraging botnets with thousands of comprised hosts are a common occurrence today. DDoS attacks are much harder to detect and defend against than a DoS attack from a single host.

4. Packet Sniffer

The convenience and ease of accessing the Internet nowadays creates a major security vulnerability—by placing a passive receiver in the vicinity of the wireless transmitter, that receiver can obtain a copy of every packet that is transmitted. These packets can contain all kinds of sensitive information, including passwords, social

security numbers, trade secrets, and private personal messages. A passive receiver that records a copy of every packet that flies by is called a packet sniffer.

In wired broadcast environments such as Ethernet LANs, the cable access technologies also broadcast packets and are thus, vulnerable to sniffing. A packet sniffer can obtain copies of broadcast packets sent over the LAN.

A malicious attacker who gains access to an institution's access router or access link to the Internet may be able to plant a sniffer that makes a copy of every packet going to/from the organization. Sniffed packets can then be analysed offline for sensitive information.

Packet-sniffing software is freely available on various Web sites and as commercial products. An example is the Wireshark lab which is usually used in networking courses. Because packet sniffers are passive—that is, they do not inject packets into the channel—they are difficult to detect. So, when we send packets into a wireless channel, we must accept the possibility that some bad guy may be recording copies of our packets. As you may have guessed, some of the best defences against packet sniffing involve cryptography.

5. IP Spoofing

IP spoofing is the ability to inject packets into the Internet with a false source address. It is surprisingly easy to create a packet with an arbitrary source address, packet content, and destination address and then transmit this hand-crafted packet into the Internet, which will dutifully forward the packet to its destination. Imagine the unsuspecting receiver (e.g. an Internet router) who receives such a packet, takes the (false) source address as being truthful, and then performs some command embedded in the packet's contents (e.g. modifies its forwarding table).

To solve this problem, we will need end-point authentication, that is, a mechanism that will allow us to determine with certainty if a message originates from where we think it does.

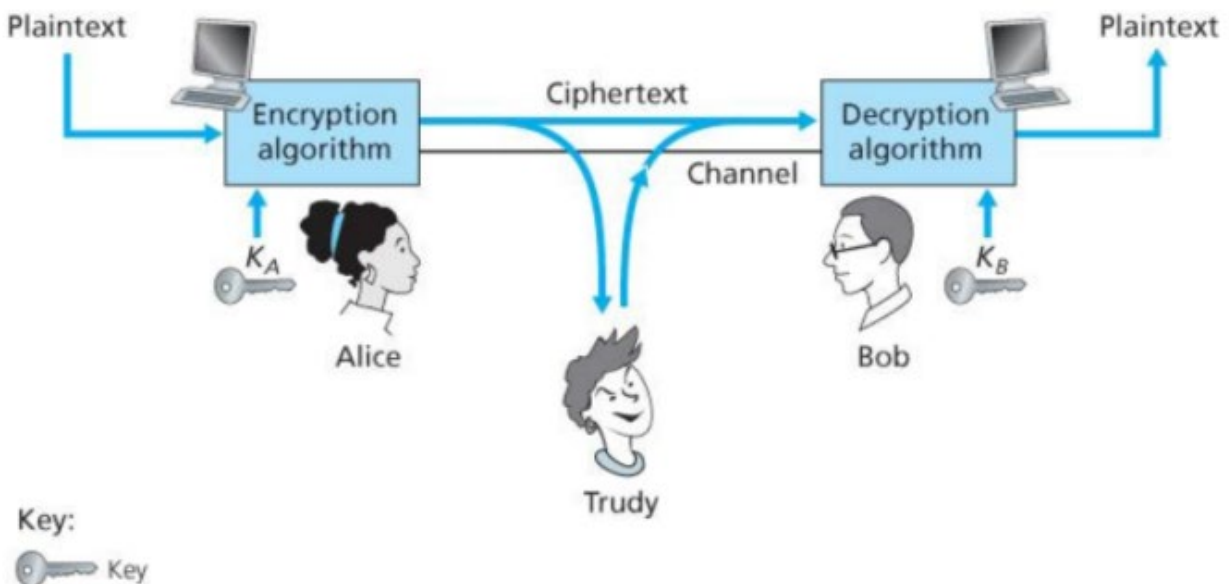
6. Confidentiality

As mentioned earlier, to ensure secure communication between the sender and the intended recipient, the message transmitted must be encrypted so that an intruder can gain no information from the intercepted data.

6.1 Encryption

Encryption is the scrambling of information so that can't be read without a key to decrypt it. Like passwords, the secret key is usually provided as a sequence of bytes. Before the encrypted data is decoded using the secret key, it appears as random, meaningless data. Encryption is often used to protect data from unauthorised access by allowing only authorised users to have the secret key. It can also be used in combination with file permissions so that an unauthorised user who is able to bypass file permissions would still be unable to use the accessed data without knowing the secret key.

Although cryptography is used for confidentiality, cryptographic techniques are inextricably woven into authentication, message integrity, and non-repudiation. Non-repudiation is the assurance that someone cannot deny the validity of something.



Cryptographic components

Source: Kurose, J., & Ross, K. (2016). *Computer Networking: A Top-Down Approach (7th ed.)*. Pearson.

There are typically two types of encryption - symmetric key cryptography and public key cryptography. We will not go through the details of each here. It would suffice at the moment to understand how encryption works.

Examples of old symmetric key cryptography are Caesar cipher, monoalphabetic cipher, polyalphabetic encryption. Modern symmetric key cryptography is categorised into two broad areas - stream ciphers and block ciphers. The latter typically, uses a technique called Cipher Block Chaining (CBC).

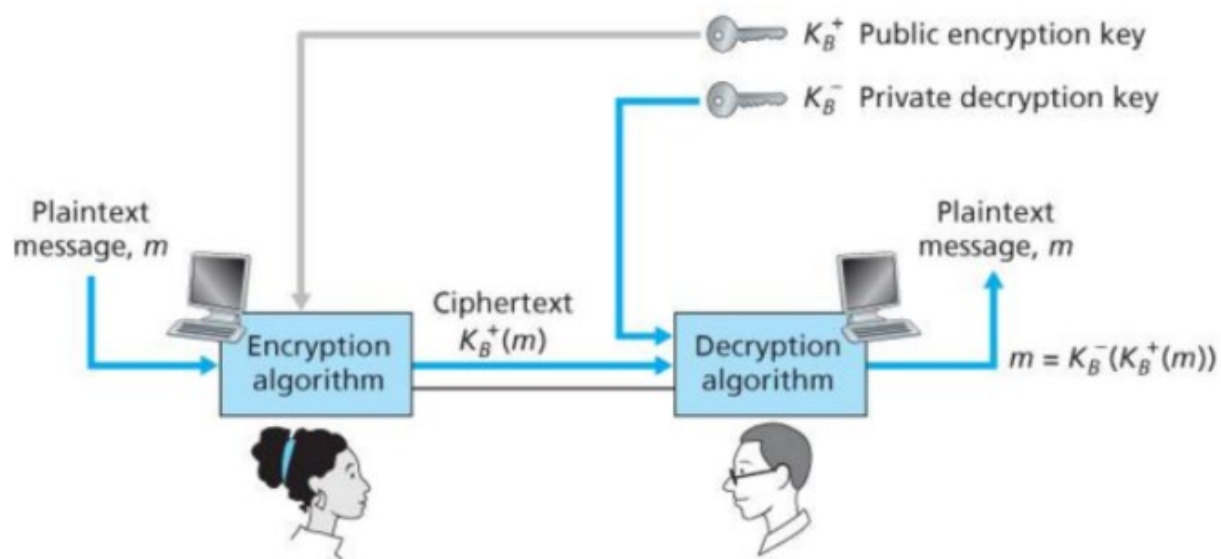
Examples of public key cryptography are RSA, RSA with session keys, and Diffie-Hellman.

Consider the diagram above. Suppose now that Alice wants to send a message to Bob. Alice's message in its original form (for example, " Bob, I love you. Alice ") is known as plaintext or cleartext. Alice encrypts her plaintext message using an encryption algorithm so that the encrypted message, known as ciphertext, looks unintelligible to any intruder. Interestingly, in many modern cryptographic systems, including those used on the Internet, the encryption technique itself is known—published, standardized, and available to everyone (for example, [RFC 1321; RFC 3447; RFC 2420; NIST 2001]), even a potential intruder. Clearly, if everyone knows the method for encoding data, then there must be some secret information that prevents an intruder from decrypting the transmitted data. This is where keys come in.

Alice provides a key, K_A , a string of numbers or characters, as input to the encryption algorithm. The encryption algorithm takes the key and the plaintext message, m , as input and produces ciphertext as output. The notation $K_A(m)$ refers to the ciphertext form (encrypted using the key K_A) of the plaintext message, m . The actual encryption algorithm that uses key K will be evident from the context.

Similarly, Bob will provide a key, K_B , to the decryption algorithm that takes the ciphertext and Bob's key as input and produces the original plaintext as output. That is if Bob receives an encrypted message $K_A(m)$, he decrypts it by computing $K_B(K_A(m))=m$.

In symmetric key systems, Alice's and Bob's keys are identical and are secret. In public key systems, a pair of keys is used. One of the keys is known to both Bob and Alice (indeed, it is known to the whole world). The other key is known only by either Bob or Alice (but not both) as shown in the diagram below.



Public key cryptography

Source: Kurose, J., & Ross, K. (2016). *Computer Networking: A Top-Down Approach* (7th ed.). Pearson.

Websites' address that start with <https://>, symbolised by a padlock icon, is a secure, encrypted way for browsers to open web pages, without anyone in between able to read the contents.

End-to-end encryption means that our messages are encrypted between us and who we are talking to, so even if we are using a third-party chat or video service, the companies in between are not able to decrypt, or read the contents of our communications.

6.2 File Permissions

Most operating systems have settings to control the ability of users to view or make changes to specific files or folders. These settings are called permissions.

Typically, users can only change the permissions for any files or folders that they own. However, most operating systems allow for a special user called an administrator to override the permissions for almost any file or folder. Besides being able to override file permissions, an administrator is usually able to perform other tasks related to authentication and authorisation, such as creating and removing user accounts or resetting passwords.

A normal user may also be given special administrator rights that allow him or her to override the permissions for certain files or folders, just like an administrator.

Unfortunately, managing permissions and administrative rights can be a complex task and it is possible to unintentionally grant access to a file or administrative rights to an unauthorised user. Such a user can then make use of such mistakes to gain unauthorised access to data and resources. In addition, authentication for the administrator must be especially strong, as an intruder that successfully claims to be the administrator can bypass file permissions entirely.

File permissions also do not prevent an intruder with physical access to a storage device from accessing files or folders directly without going through the operating system. To prevent such unauthorised access, it is necessary to use encryption.

File permissions can be used as access control for both computers that are connected to a network as well as computers that are not connected to a network but are shared by multiple users.

7. Message integrity

Suppose Bob receives a message (which may be encrypted or may be in plaintext) and he believes this message was sent by Alice. To authenticate this message, Bob needs to verify:

1. The message indeed originated from Alice.
2. The message was not tampered with on its way to Bob.

As such, we need to understand an important message integrity technique called cryptographic hash functions.

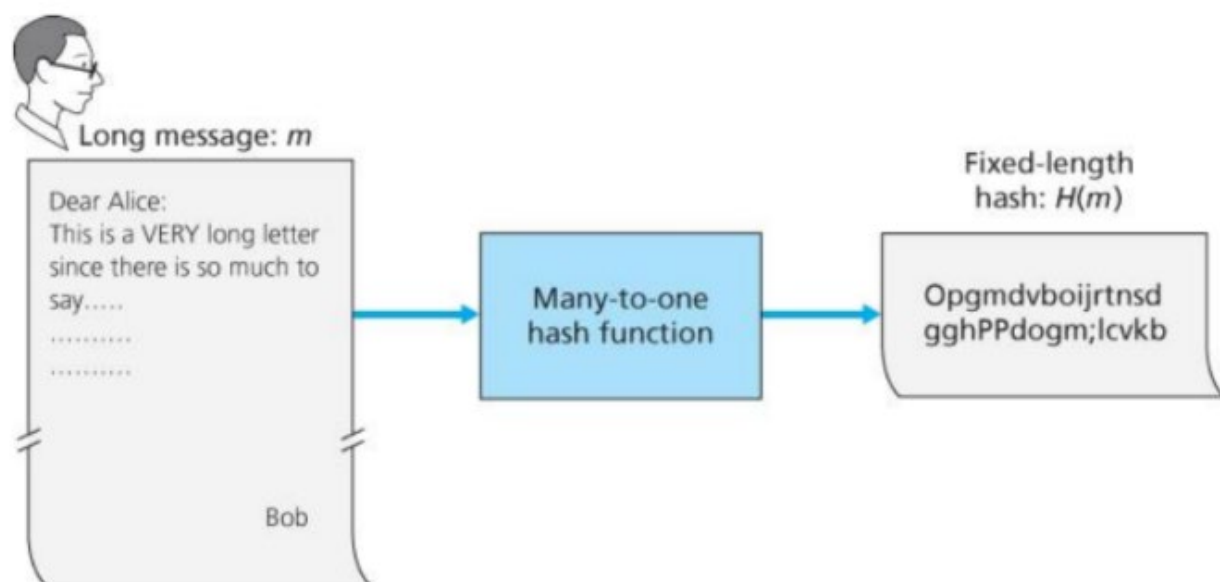
7.1 Cryptographic Hash Functions

A hash function takes an input, m , and computes a fixed-size string $H(m)$ known as a hash. The Internet checksum and cyclic redundancy check (CRC) meet this definition.

A cryptographic hash function is required to have the following additional property:

It is computationally infeasible to find any two different messages x and y such that $H(x)=H(y)$.

This means that it must be computationally infeasible for an intruder to forge the contents of another message that has the same hash value as the original message.



Hash functions

Source: Kurose, J., & Ross, K. (2016). *Computer Networking: A Top-Down Approach* (7th ed.). Pearson.

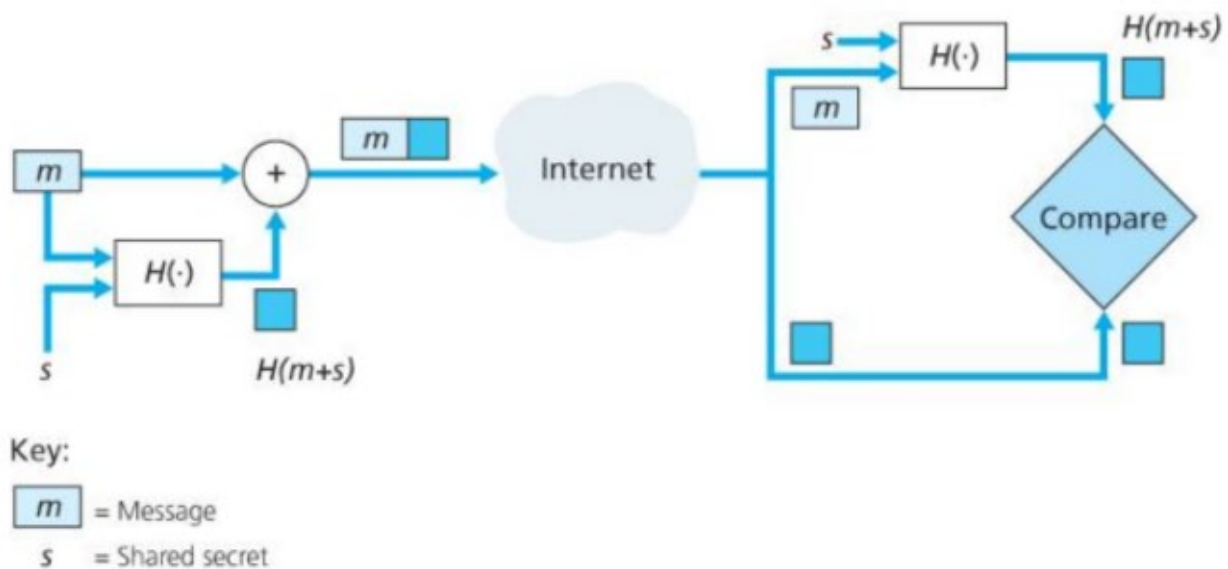
A widely used hash algorithm is the MD5 hash algorithm of Ron Rivest. We will not go through the details here. Essentially, it computes a 128-bit hash in a four-step process.

Another popular hash algorithm is Secure Hash Algorithm (SHA-1) which produces a 160-bit message digest. This longer output length makes it more secure.

To perform message integrity, in addition to using cryptographic hash functions, Alice and Bob will need a shared secret s . This shared secret, which is nothing more than a string of bits, is called the authentication key.

Using this shared secret, message integrity can be performed as follows:

1. Alice creates message m , concatenates s with m to create $m+s$, and calculates the hash $H(m+s)$ (for example with SHA-1). $H(m+s)$ is called the message authentication code (MAC).
2. Alice then appends the MAC to the message m , creating an extended message $(m, H(m+s))$, and sends the extended message to Bob.
3. Bob receives an extended message (m, h) and knowing s , calculates the MAC $H(m+s)$. If $H(m+s)=h$, Bob concludes that everything is fine.



Message Authentication Code (MAC)

Source: Kurose, J., & Ross, K. (2016). *Computer Networking: A Top-Down Approach* (7th ed.). Pearson.

The most popular MAC standard today is HMAC, which can be used either with MD5 or SHA-1. It runs data and the authentication key through the hash function twice.

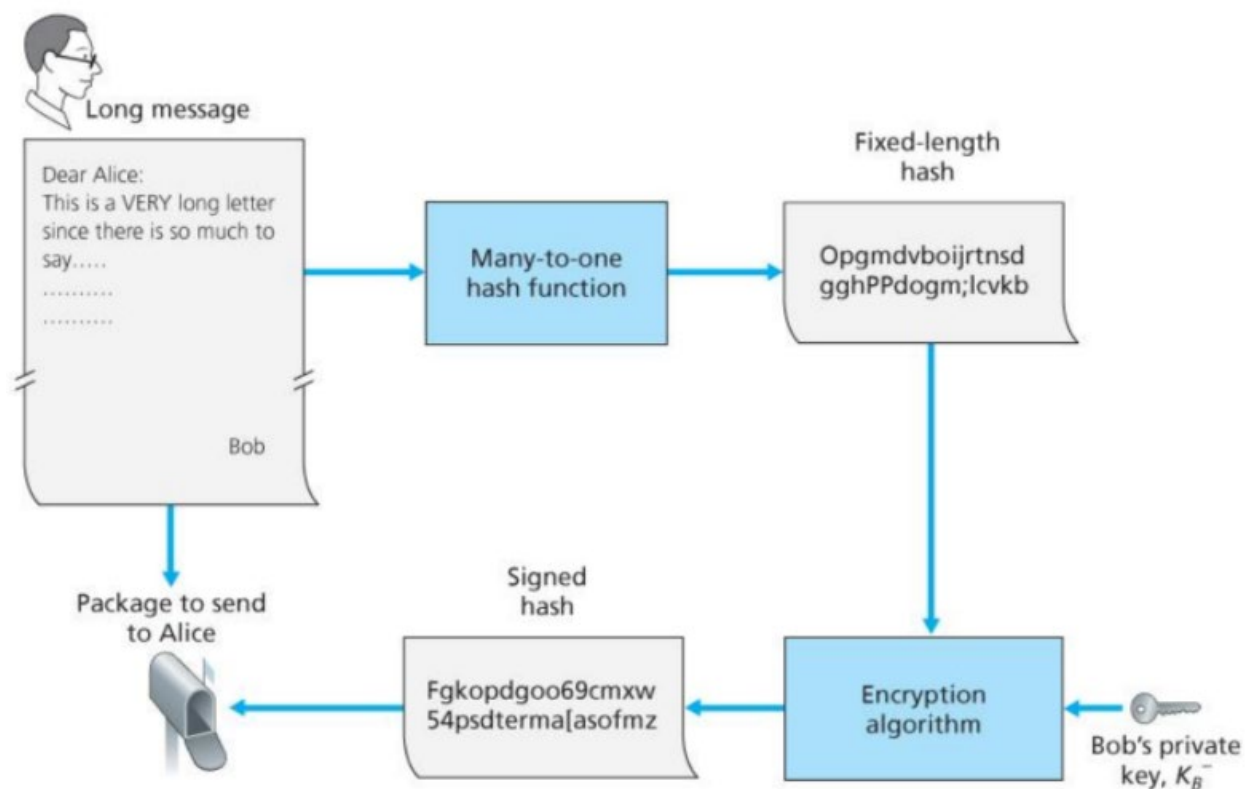
7.2 Digital Signature

A digital signature ensures the integrity of data and authenticates the sender. It is a cryptographic technique to indicate the owner or creator of a resource or to signify one's agreement with a document's content in a digital world. Therefore, it must be possible to prove that a document signed by an individual was indeed signed by that

individual (the signature must be verifiable) and that only that individual could have signed the document (the signature cannot be forged).

One may suggest that digital signatures can be generated via encryption techniques. However, one concern with signing data by encryption is that encryption and decryption are computationally expensive. Given the overheads of encryption and decryption, signing data via complete encryption/decryption can be overkill. A more efficient approach is to introduce hash functions into the digital signature. A hash function is a function that takes a set of inputs of any arbitrary size and fits them into a table or other data structure that contains fixed-size elements.

Recall that a hash algorithm takes a message, m , of arbitrary length and computes a fixed-length “fingerprint” of the message, denoted by $H(m)$. Using a hash function, Bob signs the hash of a message rather than the message itself, that is, Bob calculates $K_B^-(H(m))$. Since $H(m)$ is generally much smaller than the original message m , the computational effort required to create the digital signature is substantially reduced.



Sending a digitally signed message

In summary,

1. The sender uses a hash algorithm to create a hashed version of the message
2. The sender uses its private key to encrypt the hash to the digital signature
3. Both the message (encrypted or not) and the digital signature are sent to the receiver
4. The receiver uses the sender's public key to decrypt the digital signature back to the sender's version of hash
5. The receiver uses the same hash algorithm to create a new hash from the received message. If the two hashes match, it means the data is not altered and is sent by the known sender

An important application of digital signatures is public key certification, that is, certifying that a public key belongs to a specific entity. Public key certification is used in many popular secure networking protocols, including IPsec and SSL.

8. End-point authentication

End-point authentication is the process of one entity proving its identity to another entity over a computer network, for example, a user proving its identity to an e-mail server. A concrete example is a user authenticating him or herself to an e-mail server. This is a subtly different problem from proving that a message received at some point in the past did indeed come from that claimed sender.

Often, network elements such as routers and client/server processes must authenticate each other. Here, authentication must be done solely on the basis of messages and data exchanged as part of an authentication protocol. Typically, an authentication protocol would run before the two communicating parties run some other protocol. The authentication protocol first establishes the identities of the parties to each other's satisfaction; only after authentication do the parties get down to the work at hand.

Authentication usually requires the user to prove his or her identity by providing evidence from one or more of the following categories:

1. Something the user knows, such as a password
2. Something the user owns, such as a mobile phone

3. Something unique that is measured from a physical part of the user, such as a thumbprint

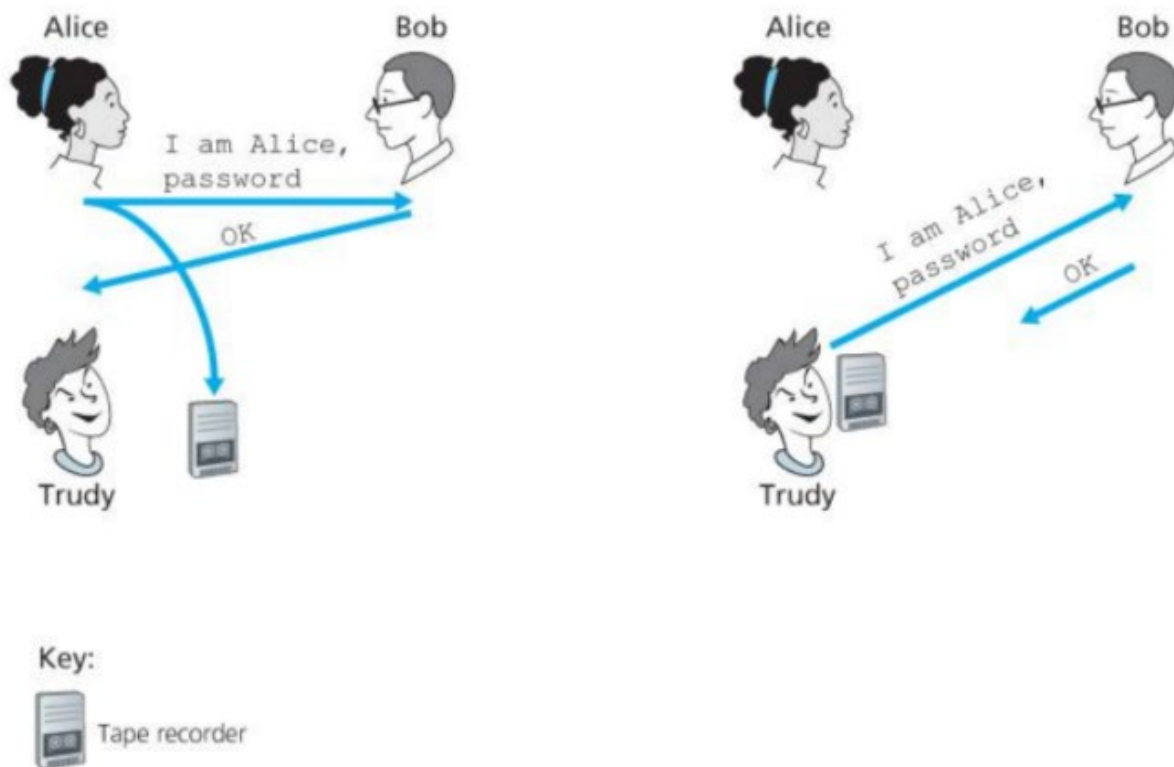
Each category of evidence that is used for authentication is called an authentication factor.

8.1 Passwords

One classic approach to authentication is to use a secret password. The password is a shared secret between the authenticator and the person being authenticated. Gmail, Facebook, telnet, FTP, and many other services use password authentication.

Unfortunately, passwords can be a poor form of authentication if they are chosen poorly or not well-kept as a secret. Avoid using birthdates and surnames as passwords as they can be easily guessed by an intruder. Instead, use hard-to-guess passwords that are a mixture of lower-case letters, upper-case letters, numbers and symbols.

Avoid re-using passwords or leaving them unchanged for a long time as doing so makes it easier for an intruder to guess the password. Instead, use unique passwords for each computer or online account and update them at least once every 90 days.



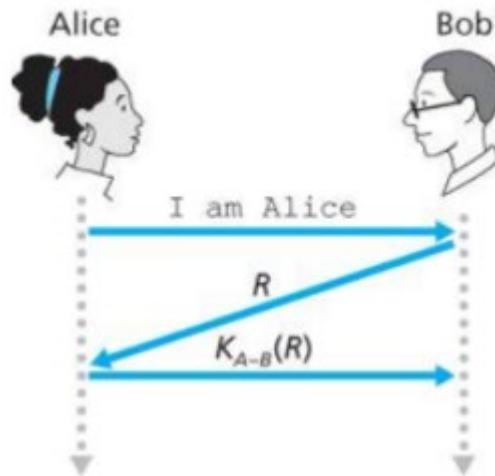
Authentication Protocol (Left) and Possible Failure Scenario (Right)

The failure scenario in the above diagram resulted from the fact that Bob could not distinguish between the original authentication of Alice and the later playback of Alice's original authentication. As such, we can implement something like the three-way TCP handshake protocol using ACK and SYN segments.

A more secure authentication protocol:

1. Alice sends the message " I am Alice " to Bob.
2. Bob chooses a nonce, R , and sends it to Alice. A nonce is a number that a protocol will use only once in a lifetime.
3. Alice encrypts the nonce using Alice and Bob's symmetric secret key, and sends the encrypted nonce, (R) , back to Bob. The fact that Alice knows and uses it to encrypt a value, lets Bob know that the message he receives was generated by Alice. The nonce is used to ensure that Alice is live.

4. Bob decrypts the received message. If the decrypted nonce equals the nonce he sent Alice, then
5. Alice is authenticated.



A more secure Authentication Protocol

8.2 Security Tokens (Two-Factor Authentication)

More stringent authentication systems often require evidence from more than one authentication factor. For instance, banks typically issue a device called a security token to users who wish to access their bank accounts online.

To access his or her bank account online, the user has to confirm his or her identity by providing a secret password or personal identification number (PIN), followed by a one-time password (OTP) generated from the security token or a mobile phone that the user owns. This kind of authentication that uses evidence from both something the user knows and something the user owns is called two-factor authentication.

An OTP is usually a string of seemingly random numeric or alphanumeric characters. For security reasons, an OTP is usually valid only for a short period of time, after which a new one has to be requested. This reduces the possibility that the OTP is obtained by an intruder between the time it is generated and when it is accepted by the bank. OTPs may either be generated by the bank and sent securely to the user's device, or generated directly on the device using a secret algorithm that only the bank knows.

Two-factor authentication is stronger than using only a password as it is much more difficult for an intruder to both guess a password and steal the user's security token. This is why it is important to keep the security token in a secure location at all times and to report a missing security token as soon as possible.

However, if an OTP is sent wirelessly to the user's mobile phone, it may be intercepted and used by an intruder during the transmission process. Alternatively, if the secret algorithm used to generate OTPs is poorly chosen or accidentally revealed, an intruder may find out how to generate OTPs without needing the security token at all. Unfortunately, there is not much that a user can do to prevent such intrusion attempts.

8.3 Biometrics

Biometrics is a type of authentication that is based on the measurement of human physical characteristics. For example, biometrics is used to identify a user by fingerprint or voice.

Other common characteristics used in biometrics include the face, iris, retina, and deoxyribonucleic acid (DNA).

Compared to passwords, the use of biometric identification is more secure as the physical characteristics measured are typically unique to the individual and cannot be easily replicated. Thus, biometrics can help guard against attempts to establish fraudulent identities and prevent identity theft.

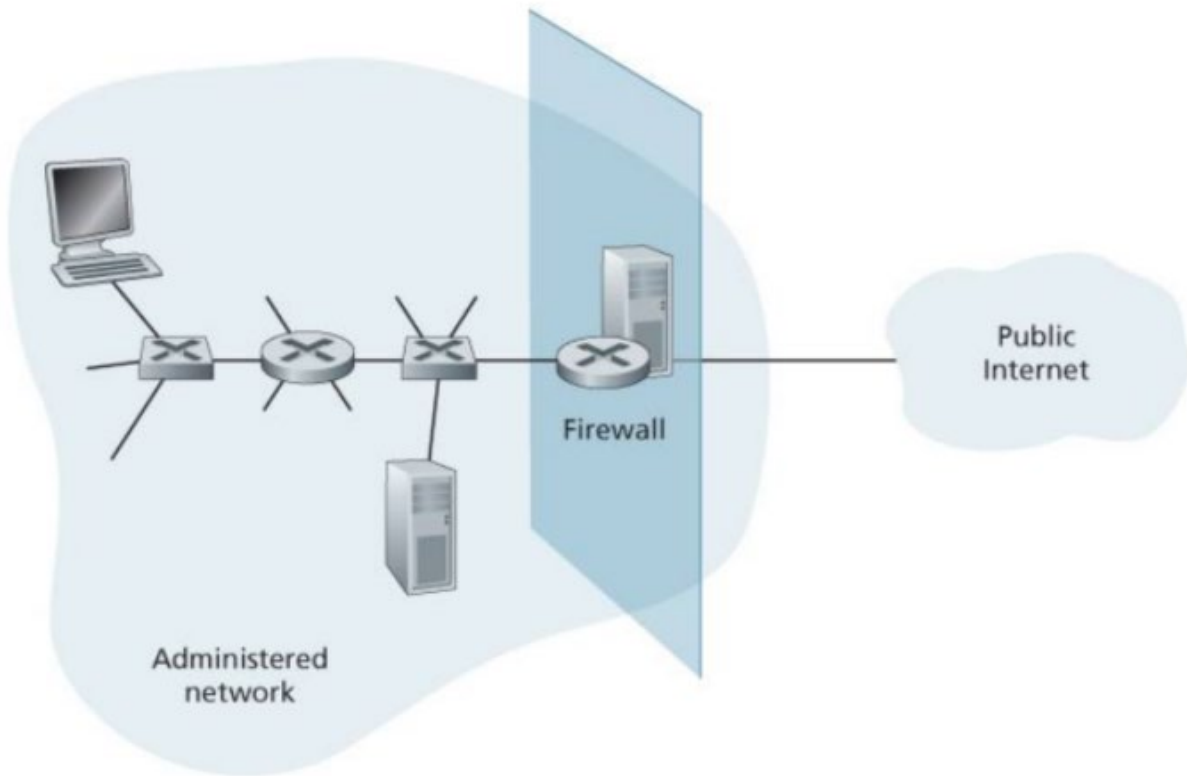
9. Operational Security

Earlier, we mentioned that operational devices are used to counter attacks against a network. Traffic entering/leaving a network is security-checked, logged, dropped, or forwarded. This is done by operational devices known as firewalls, intrusion detection systems (IDSs), and intrusion prevention systems (IPSs).

9.1 Firewalls

A firewall is a combination of hardware and software that isolates an organization's internal network from the Internet at large, allowing some packets to pass and blocking others. A firewall allows a network administrator to control access between the outside

world and resources within the administered network by managing the traffic flow to and from these resources.



A firewall has three goals:

1. All traffic from outside to inside, and vice versa, passes through the firewall, which sits squarely at the boundary between the administered network and the rest of the Internet. While large organizations may use multiple levels of firewalls or distributed firewalls, locating a firewall at a single access point to the network makes it easier to manage and enforce a security-access policy.
2. Only authorised traffic, as defined by the local security policy, will be allowed to pass.
3. The firewall itself is immune to penetration as it is a device connected to the network.

Firewalls can be classified into three categories: traditional packet filters, stateful packet filters, and application gateways. For the sake of simplicity, details on these are not provided here.

Traditional packet filters occur at a gateway router that connects the internal network to its ISP. It examines each datagram based on the administrator-specific rules.

Stateful packet filters track TCP connections and use this information to make filtering decisions.

Application gateways are application-specific servers through which all application data (inbound and outbound) must pass. They look beyond the IP/TCP/UDP headers and make policy decisions based on application data.

You may refer to this resource for more in-depth discussion on firewalls:

<https://www.ciscopress.com/articles/printfriendly/1823359>

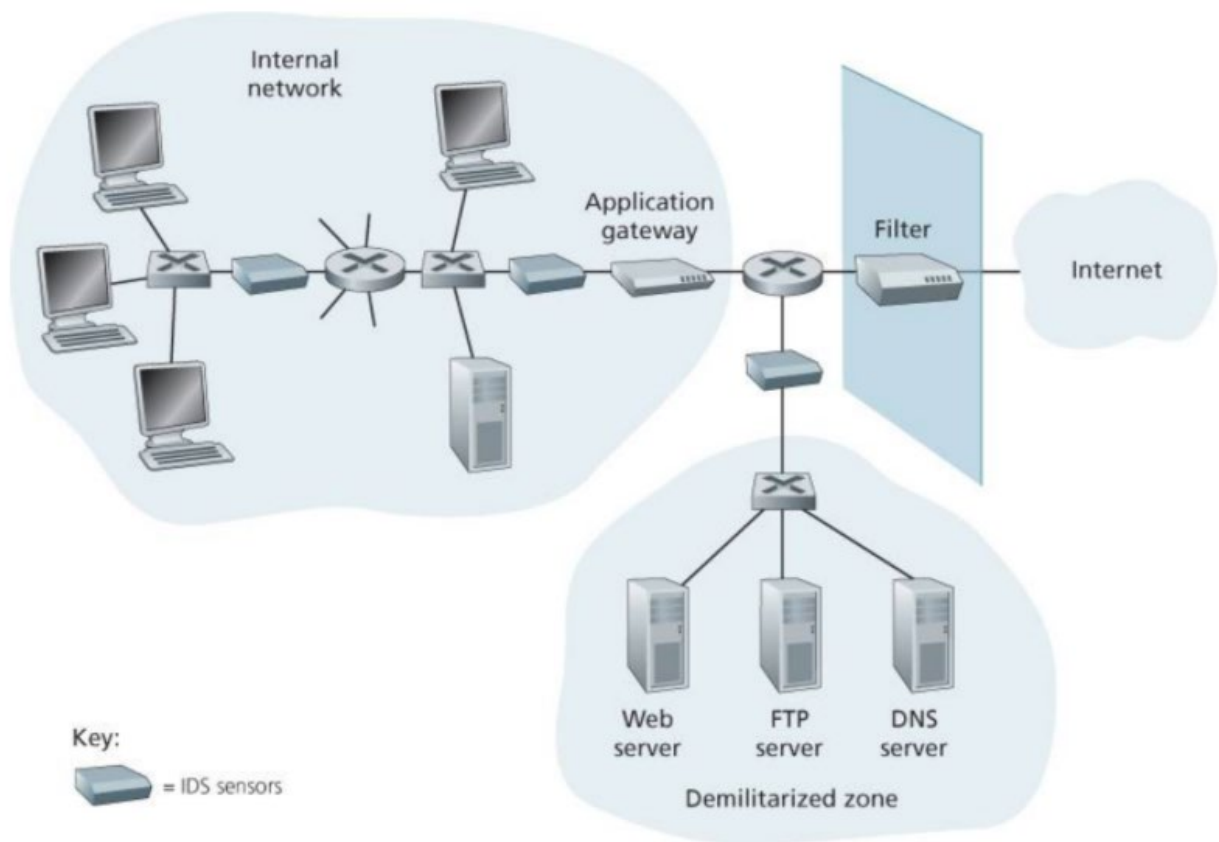
9.2 Intrusion Detection System (IDS) and Intrusion Prevention System (IPS)

To detect many attack types, we need to perform deep packet inspection, that is, look beyond the header fields and into the actual application data that the packets carry. Application gateways often do deep packet inspection but this is only possible for a specific application.

Thus, IDS and IPS are needed. IDS is a device that generates alerts when it observes potentially malicious traffic. IPS is a device that filters out suspicious traffic. Normally, we collectively refer to IDS and IPS systems as IDS systems.

An IDS can be used to detect a wide range of attacks, including network mapping (emanating, for example, from nmap), port scans, TCP stack scans, DoS bandwidth-flooding attacks, worms, and viruses, OS vulnerability attacks, and application vulnerability attacks.

An organisation may deploy one or more IDS sensors in its organizational network. When multiple sensors are deployed, they typically work in concert, sending information about suspicious traffic activity to a central IDS processor, which collects and integrates the information and sends alarms to network administrators when deemed appropriate.



In the above diagram, the organisation has partitioned its network into two regions - a high-security region, protected by a packet filter and an application gateway and monitored by IDS sensors; and a lower-security region also known as a demilitarized zone (DMZ), which is protected by the packet filter, and monitored by IDS sensors. Note that the DMZ includes the organisation's servers that need to communicate with the outside world, such as its public Web server and its authoritative DNS server.

IDS not only needs to do deep packet inspection but must also compare each passing packet with tens of thousands of signatures. Each signature is a set of rules pertaining to an intrusion activity such as the packets' source and destination port numbers, protocol type, and a specific string of bits in the packet payload. This can be a significant amount of processing and may actually fail to detect malicious packets.

By placing the IDS sensors further downstream, each sensor sees only a fraction of the organisation's traffic, and can more easily keep up. Nevertheless, high-

performance IDS and IPS systems are available today, and many organisations can actually get by with just one sensor located near their access router.

IDS systems are broadly classified as either signature-based systems or anomaly-based systems. For the sake of simplicity, details of these systems will not be covered here.

A signature-based IDS maintains an extensive database of attack signatures and thus, may be blind to new attacks that have yet to be recorded. Even if the signature is matched, it may not be the result of an attack.

An anomaly-based IDS creates a traffic profile as it observes traffic in normal operation. It then looks for packet streams that are statistically unusual, for example, an inordinate or excessive percentage of ICMP packets or a sudden exponential growth in port scans and ping sweeps. It does not rely on previous knowledge about existing attacks. However, it is an extremely challenging problem to distinguish between normal traffic and statistically unusual traffic.

You may refer to this resource for a more in-depth discussion on IDS/IPS:

<https://www.socinvestigation.com/ids-vs-ips-key-differences-rule-structure-pros-and-cons/>

<https://www.giac.org/paper/gsec/235/limitations-network-intrusion-detection/100739>

Referenced Readings

1. Kurose, J., & Ross, K. (2016). Computer Networking: A Top-Down Approach (7th ed.). Pearson.
2. O-level Computing Textbook (3rd ed.). CPDD, MOE
3. <https://cheapsslsecurity.com/blog/digital-signature-vs-digital-certificate-the-difference-explained/>
4. <https://www.docusign.com/how-it-works/electronic-signature/digital-signature/digital-signature-faq#:~:text=Digital%20signatures%20are%20like%20electronic,document%20in%20a%20recorded%20transaction.>

