Algorithmic Representation

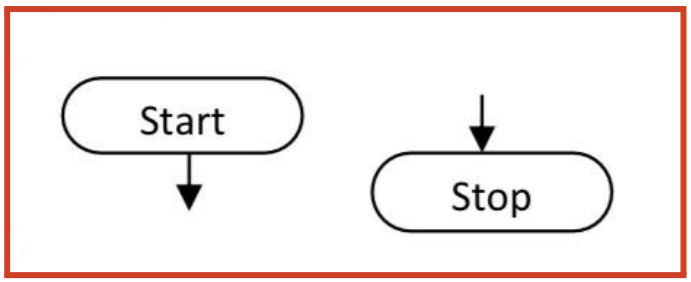
definitions are highlighted!

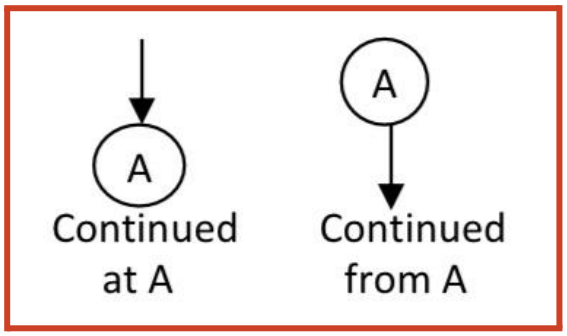
Algorithms

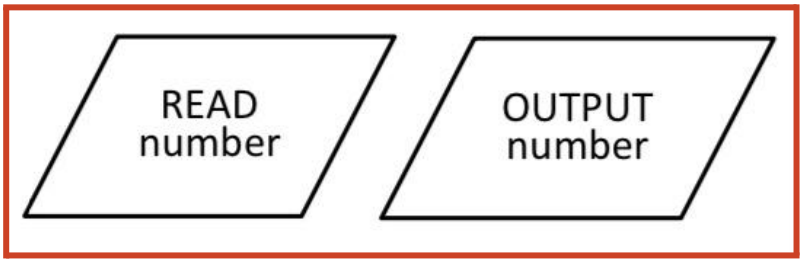
- A sequence of steps designed to perform a particular task
- *A well-ordered collection of (#1) unambiguous and (#2) effectively computable operations*
    - Well-ordered — the order in which to do the operations is clear
    - Unambiguous — can be understood and carried out directly by a computing agent, without needing to explain or simplify further
    - Effectively computable — there exists a computational process that allows the computing agent to complete the operation successfully
- Solve problems — to know whether a solution is correct, the algorithm must produce an observable result, so that the user can determine whether it is the desired one.
    - The result must be produced after the execution of a finite number of operations.
- e.g. recipe, instructions
- Can be represented in any suitable form such as a programming language, pseudo-code (easily communicated and easily translated into any suitable programming language) and diagrams like flowcharts

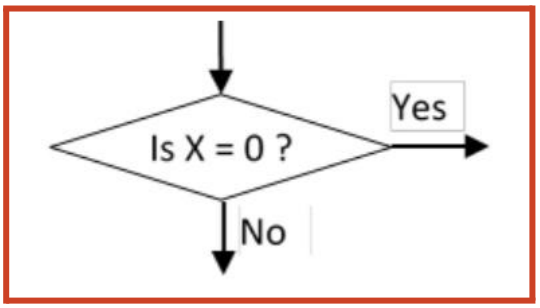
Flowcharts

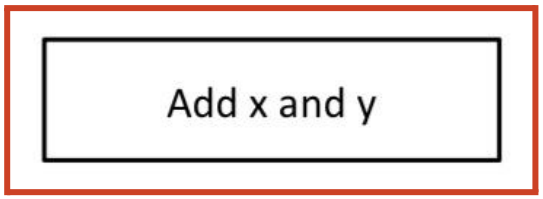
- *A graphical representation of the operations involved in a process or system*
- Uses symbols to represent operations and flow lines to indicate the sequence of operations
    - Start/Stop (Begin/End-terminal)

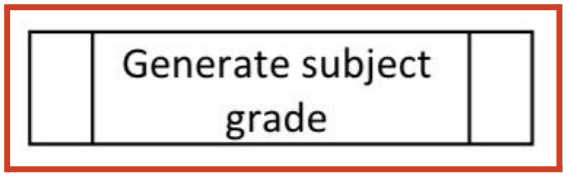
- 

  - A flowchart will have only one start point but there may be more than one stop point.
- Connector
  - 

  - Used to indicate that the flowchart is continued elsewhere at an identically labelled connector point
- Input/Output
  - 

  - Used for any input or output operation
  - The description in the box may include the device or medium for the process.
- Decision box
  - 

        - Used to indicate points in a program where decisions are made
    - Processing box
        - 

        - For any operation or sequence of instructions that does not involve a decision
    - Subroutine symbol
        - 

        - Used to indicate a subroutine call — a set of program instructions performing a specific task, but not a complete program; it must be incorporated into a program in order to be used.
    - Flow lines may have arrows to show direction. If there are no arrows, a top-down, left-right convention applies.
- To present an algorithm as a flowchart, categorise the problem into the following: Input, Process, Output.
    - The basic operation of a computer is: input (through the keyboard), process (through the processor), output (through the monitor).

Pseudo-code

- *A method of describing an algorithm or program design*
- Uses control structures, keywords and identifiers similar to those found in programming languages, but without following the syntax (the precise way in which a program must be written to be understood by

the computer; a set of rules for combining elements of the programming language in the forms that a computer would understand) of any programming language

- Looks like a combination of English and a programming language; there is no standard way of writing it, but we follow a convention
- A programmer can translate pseudo-code to program code using a high level programming language (e.g. Python)
- Terms:
    - Identifiers
        - *Name or label chosen by the programmer to represent an object within a program*
            - The object could be a variable, function, procedure, data type or any other element defined within a program
        - Can contain letters, digits and the underscore symbol (_), but the first character should not be a digit
        - Cannot contain keywords
        - Should be named meaningfully
        - Single letters for conventional uses:
            - Array indices: i, j
            - Coordinates: x, y
    - Keywords (33 of them in Python)
        - A keyword is a *reserved name used to recognise the structure of a programming language*.
        - Has its own meaning that is defined in the language; cannot be used as identifiers
        - e.g. break, continue, while, for, return
    - Comments
        - Make the algorithm or program code easier to understand, especially for parts that are more difficult to understand

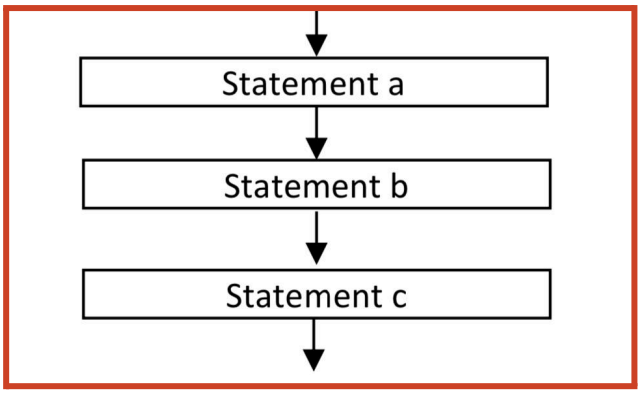
        - Can be used to document program code, specify the name of the program, author, date written, purpose of the program etc.
        - Use 2 forward slashes (//) in pseudo-code
    - Data type
        - Determines:
            - The type of values that a variable can have
            - The operations that can be performed on the variable
            - The range of values that the memory location can hold
        - Some basic data types:
            - Every data type takes up a certain amount of space (in bytes) in the computer memory.
            - Integers, real numbers — 4 bytes
            - A single character — 1 byte
            - String — 1 byte per character
            - Boolean — 1 byte
    - Variables
        - *A storage location for a data value that has an identifier*
        - In any program input, data has to be stored in the computer's memory and accessed for the process of the program; these memory locations are called variables.
        - In pseudo-code, these variables should be explicitly declared.
            - e.g. "DECLARE age: INTEGER"
        - Subset of identifier
    - Constants
        - Hold values that never change while the program is run
        - Usually declared like this: "CONSTANT radius = 3.0"

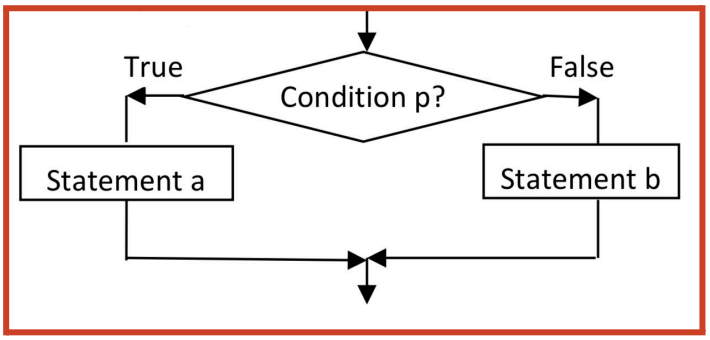
- Assignments
  - Assign a name (identifier) with a value, or the value associated with a given identifier is changed
  - You can use either an arrow that points backwards (←) or an equal sign (=).
- Operators
  - Arithmetic operators: For the manipulation of numbers
    - +
    - -
    - *
    - /
  - Relational operators: Compare data and produce a result of BOOLEAN data type
    - >
    - <
    - >=
    - <=
    - =
    - <> (not equal to)
  - Logic operators: The operands and results of these operations are always of data type BOOLEAN.
    - e.g. AND, OR, NOT; example usage: “number < 7 AND number > 2”

- **Note**: For a range in pseudo-code, both ends are inclusive!

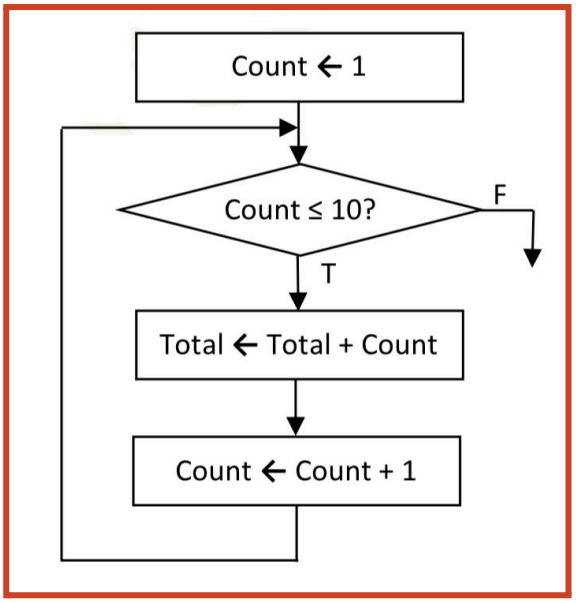
Control structures

- Manage the flow of the program — i.e. different ways in which the instructions can be executed

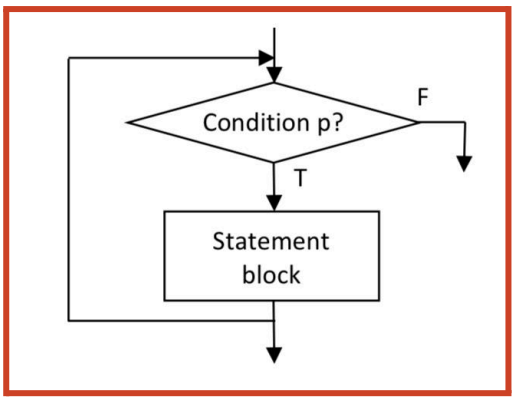
- Sequence
  - 

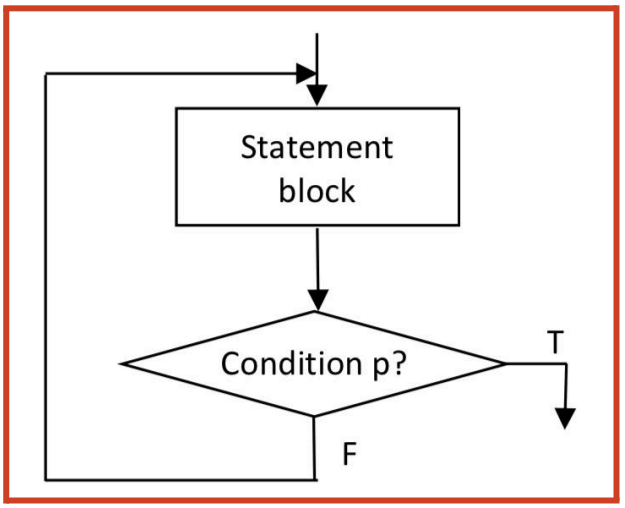
  - Execution of one processing step, one after another
- Selection
  - 

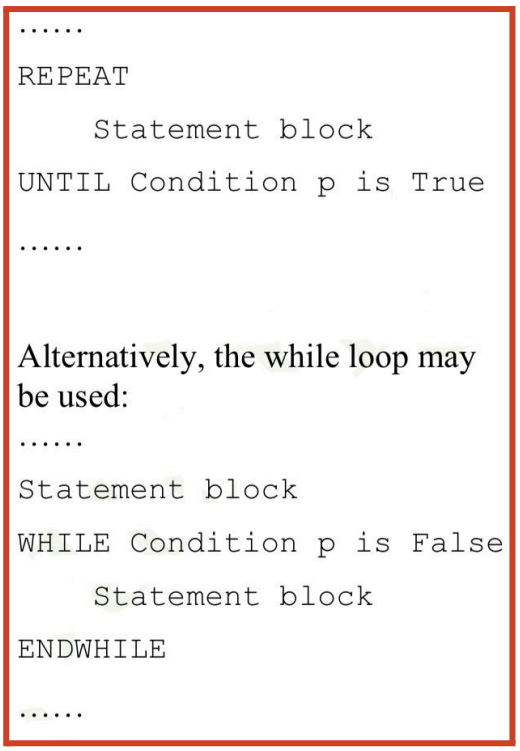
  - A condition is presented and a choice is made between 2 actions depending on whether the condition is true or false
  - Uses an IF-THEN-ELSE construct in pseudo-code
- Iteration (repetition/loop)
  - The presentation of a set of instructions to be performed repeatedly as long as a condition is true
  - Based on the given context, we can choose one (or more) of these 3 methods:
    - FOR loop (aka count-controlled loop)

- 

- ```
.....
FOR Count ← 1 TO 10 STEP 1
  IF Count <= 10
    THEN
      Total ← Total + Count
  ENDIF
ENDFOR Count
.....
```
    - This means that “Count” can take on any value between 1 to 10 (both inclusive), increasing by 1 each time.
    - By default, if “STEP” is not defined, it is taken to be 1.
- Loop is executed a fixed number of times, which is determined before the loop begins
```

- WHILE loop (aka pre-test loop)
  - 

  - ```
    .....
    WHILE Condition p is True
        Statement block
    ENDWHILE
    .....
    ```
  - Tests the condition before entering the loop (so it may not even enter the loop)
- REPEAT-UNTIL loop (aka post-test loop)
  - 
    ```



```
.....
REPEAT
    Statement block
UNTIL Condition p is True
.....
```

Alternatively, the while loop may be used:

```
.....
Statement block
WHILE Condition p is False
    Statement block
ENDWHILE
.....
```

- Tests the condition after the body of the loop is executed (so it executes the body of the loop at least once)
- The body of the loop keeps repeating until the condition is true.

Modular design (aka top-down design)

- *A method of organising a large computer program into self-contained parts, called modules, which can be developed simultaneously by different programmers*
- Start with original problem and split the problem into smaller and smaller parts — modules — until they become manageable
    - These modules are easier to understand than trying to write one whole program to calculate the wages.
    - When each module has been solved, they are combined to produce a complete solution to the whole problem.
- Modules are incorporated through interfaces.

- Can be represented as a structure diagram, to show how a complex problem can be broken down into modules
- Advantages:
    - More than one person can be engaged in solving parts of the same problem simultaneously.
    - The person producing a module is likely to make fewer mistakes.
    - Makes testing the program code and elimination of errors easier, thereby improving the reliability of the program.
    - Modules can be reused in more than one project, if a similar solution is needed.
        - Collections of modules like these are known as software/program libraries.
- Modules can be implemented as subroutines or subprograms — a named block of code that performs a specific task within a program. There are 2 types of subroutines:
    - *Functions — returns a result after running*
        - "Use a function for [task] because it should return the value of … so that it may be used for other modules of the program, such as …"
    - *Procedures — does not return a value after running*
        - "Use a procedure for [task] because it does not need to return any value to the calling program"

Decision table

- A precise way of modelling logic
    - Some programs require multiple decisions to be considered to produce the correct output.
    - Managing all possible outcomes of multiple decisions can be difficult, so a decision table can organise possible decision outcome combinations.

- Each possible combination of conditions is considered in turn and the action required.
- There are 3 types of decision tables:
    - Limited entry decision table
        - Limited condition (yes/no)
        - Limited action
    - Mixed entry decision table (focus on this)
        - Limited condition (yes/no)
        - Extended action
    - Extended entry decision table (focus on this)
        - Extended condition
        - Extended action
- Format:

Conditions (questions) — conditions tested	Outcomes — possible outcomes
Actions — possible action on conditions	Action entries (results) — action(s) adopted on outcomes

e.g.

CONDITIONS	OUTCOMES			
	1	2	3	4
#1…	Y	Y	N	N
#2…	Y	N	Y	N
ACTIONS				
#1…	X			
#2…				X

    - In the bottom-right quadrant:
        - For limited action entries, use ‘X’ to denote that a particular action has been chosen, a blank (‘ ’) to ignore, or

'*' to indicate that no such outcomes exist and these columns should be omitted (when removing redundancies).
        - For extended action entries, any statement may be used.
- Number of possible outcomes:
    - To every decision, there are usually 2 possible outcomes (Y or N).
    - So, for n mutually exclusive decisions, there will be 2^n possible outcomes.
        - However, usually, several possibilities may result from these related decisions, and we may be able to combine some decisions into the same column.
    - In our syllabus, there can be a maximum of 3 decisions i.e. 8 columns.
- Advantages of using a decision table:
    - Possible to check all combinations (2^n)
    - Easy to trace from actions to conditions and vice versa
    - Standardised format — easier to draw and change compared to a flowchart
    - Compact and easier to follow one column — a small table can replace several pages of flowcharts
    - Alternatives (both actions and conditions) can be grouped to facilitate analysis, providing concise descriptions of logically complex situations
- Limitations of using a decision table:
    - Large decision tables can become incomprehensible and difficult to modify.
    - Flowcharts are better to express the total sequence of events needed to solve a problem.
- Columns can be combined to reduce redundancies.
    - To find redundancies, we look at each action and then check whether all the conditions are required.

    - We can use a dash (“—”) in cells where the action will be the same no matter whether the condition is true or false.
- Decision tree
    - Shows possible choices to take in reaching an outcome
    - Identical to probability tree
    - Extremely easy to use — working from the root, we can easily see which condition satisfies any particular situation and then follow the appropriate branch of the tree. Eventually, after all the appropriate conditions have been fulfilled, we will end up at the leaf of the tree where the action to be taken is encountered.

@star.notes