Sorting Algorithms

Bubble Sort

- The purpose of sorting algorithms is to order an unordered list. Item can be ordered alphabetically or by number.
- Bubble sort steps through a list and compares pairs of adjacent numbers. The numbers are swapped if they are in the wrong order. For an ascending list if the left number is bigger than the right number the items are swapped otherwise the numbers are not swapped.
- The algorithm repeatedly passes through the list until no more swaps are needed.

Example

Sort the following sequence in ascending order using bubble sort: 5,3,4,1,2.

Pass 1	**5**	**3**	**4**	**1**	**2**	
	3	5	4	1	2	Compare 5 and 3 – swap
	3	4	5	1	2	Compare 5 and 4 – swap
	3	4	1	5	2	Compare 5 and 1 – swap
	3	4	1	2	5	Compare 5 and 2 – swap; end of pass 1
Pass 2	3	4	1	2	5	Compare 3 and 4 – no swap
	3	1	4	2	5	Compare 4 and 1 – swap
	3	1	2	4	5	Compare 4 and 2 – swap
	3	1	2	4	5	Compare 4 and 5 – no swap; end of pass 2
Pass 3	1	3	2	4	5	Compare 3 and 1 – swap
	1	2	3	4	5	Compare 3 and 2 – swap
	1	2	3	4	5	Compare 3 and 4 – no swap
	1	2	3	4	5	Compare 4 and 5 – no swap; end of pass 3
	1	**2**	**3**	**4**	**5**	

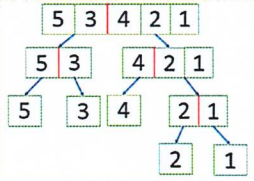
Bubble sort Pseudocode

```
A=[5,3,4,1,2]
sorted ← False
WHILE not sorted
 sorted ← True
 FOR I TO LEN(A)-1:
  IF A[i] > A[i+1]:
   temp ← A[i]
   A[i] ← A[i+1]
   A[i+1] ← temp
   sorted ← False
  ENDIF
 ENDFOR
ENDWHILE
OUTPUT A
```

Merge Sort

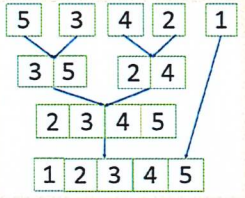
- Merge sort is a type of divide and conquer algorithm.
- There are two steps: divide and combine
- Merge sort works by dividing the unsorted list sublists. It keeps on doing this until there is 1 item in each list.
- Pairs of sublists are combined into an ordered list containing all items in the two sublists. The algorithm keeps going until there is only 1 ordered list remaining.
- Merge sort is a recursive function, that calls itself.

Step 1: Divide



Keep dividing until there is only 1 item in each list

Step2: Combine



1. The first items in the two sublists are compared, and the smallest value is copied to the parent list.
2. The copied item is then removed from the sublist.
3. When there are no items left in one of the sublists the remaining items in the other sublist are them copied in order to the parent list.

Merge sort Versus Bubble sort

	Advantages	Disadvantages
Bubble sort	Very simple and robust algorithm	Can be slow particularly for long lists. As the number of items increases the time taken for the algorithm to run increases dramatically.
Merge sort	Much faster than bubble sort especially when the number of elements is large	More complex to understand Step 1: Divide Step 2: Combine

Computer Systems

A computer system has both hardware and software.

Hardware are the physical components that make up a device or computer system. These include both the internal components (eg motherboard, CPU, RAM) and peripheral devices such as printers.

Software is the computer code, programs and algorithms that give instructions to the hardware to make it perform the desired task. Without the software the hardware will not get any instructions and it will not do anything.

Software Classification

Software is split into two types: application software and system software

Application software is a program designed to perform a specific task that the user interacts directly with (eg spreadsheets, web browser and word processor, disk defragmentation).

System software is concerned with the running of the computer. Its purpose is the control the computer hardware and manage the application software. (eg operating system, antivirus, backup tools, firewall)

The **operating system (OS)** is the most important piece of system software. The OS handles management of the processor, memory, input/output devices, applications and security.

- **Application management** - Application software does not need to concern itself with interaction and complexities of managing the hardware because this is dealt with by the operating system. Application software runs on top of operating system which is an intermediary and takes care of interaction with the hardware.

- **Processor resources** – Allows multiple applications to be run simultaneously by manages the processing time between applications and cores and switching processing between applications very quickly. Multiple applications will access the processor resources via a schedule that alternates process between applications. High priority applications will have more CPU time, but it means that lower priority applications will take longer to run.

- **Memory management** – Distributes memory resources between programs and manages transfer of data and instruction code in and out of memory. Ensures that each application does not use excessive memory.

- **Security** – Tools such as anti-virus software and firewalls help protect the computer from attack. In addition requirement for passwords and control of access rights
.
- **Input / Output devices** – OS controls interaction with input (eg keyboard) outputs (eg. Monitor) and storage (eg hard disk) using hardware drivers. Allows users to save files to the hard disk and print documents for instance.

Cloud Computing

- Can store data and files on a server elsewhere that can be accesses via the internet.
- Can use applications over the internet
- Can sync files so that all your devices see the same files
- Can share documents with others
- Can access your files anywhere if you have a good internet connection

Advantages of cloud computing
- Only pay for storage that you use
- Data and files available from anywhere in the world where there is an internet connection
- Data automatically backed up

Disadvantages of cloud computing
- Need a reliable network connection
- Files are hosted elsewhere so a security concern
- the most recent versions of software is often not available
- Transfer of data over the internet will slow down performance.

Advantages of local storage
- Files can be accessed even when there is no internet connection
- More secure as files to not need to be transferred over the network and the user has more control

Disadvantages of local storage
- Users need to organise their backup solutions
- Not so easy to share documents
- Can only access the files locally

Memory

Volatile memory (main memory) When the computer is turned off the contents of volatile memory is lost. When there is no power, volatile memory is erased.

Non-volatile memory (secondary storage) Even when here is no power, the data remain unchanged and can be accessed once again once power has been resumed. This allows you to store files for he long term.

ROM (Read Only Memory) Data can only be read from the device, and cannot the memory cannot be edited or deleted. ROM is only used for situations where you can be sure that updates will not be needed. The computer's BIOS (basic input output system) which controls the boot up sequence is stored on a ROM chip.

RAM (Random Access Memory) - When applications are executed they are loaded into RAM first. RAM is volatile.

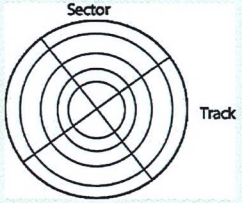
Embedded Systems

An embedded system is a computer system that is designed for a specific function, in contrast to a general-purpose computer that can carry out many tasks. Embedded systems typically have a minimal or no user interface. Thus, they can be optimised for size and power consumption, for instance. Examples of embedded systems include digital watches, MP3 players, washing machines, cars and mobile phones.

Secondary Storage

Secondary storage is necessary for saving files long and software including the operating system. Even when the computer is turned off, the data remain unchanged, and can be accessed again once the power supply has been turned on.

Magnetic Hard Disk
- Tracks on the disk platters contain tiny magnets, each holding 1 bit of data.
- The polarity (negative or positive) of the magnets determines whether the bits are 0 or 1.
- The write head modifies the polarity of the magnet as appropriate.
- The read head identifies whether each magnet is negative or positive.
- The tracks are laid out as a series of concentric rings.



Advantages
- Cheap form of storage

Disadvantages
- Less reliable because it contains moving parts that can break
- Electromagnetic surge can corrupt the data held
- Slow speed of read/write access

Optical Disks

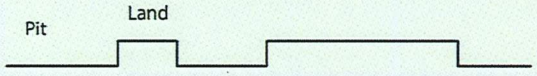
- Tracks on the disk contain pits and lands.
- The track is a spiral.
- A laser is emitted and the laser light is reflected when it hits the lands, but is scattered when it hits the pits.
- Depending on whether the light is scattered light is encoded as a binary value of 0 and reflected light is encoded as a 1.
- The sensor is able to detect light reflected, but not scattered.
- Example: Blue-Ray (25 Gb) DVD (4.7 Gb), CD (700 Mb).

Advantages

- Can transfer easily between computers

Disadvantages

- Can scratch easily
- Not much storage compared with other methods.
- No unlimited writes to the hard disk



Solid state Drive

- Use millions of switches called floating gate transistors on microchips to store data.
- Electrons are stored in gates and this is encoded as 0 when there is an electron present and encoded a 1 when there is no electron present.
- The electros remain trapped even when there is no flow of electricity.
- Contain no moving parts and are therefore more robust that optical and magnetic storage.

Advantages

- Much faster that other means of storage
- More reliable than other means if you are only reading
- Quiet

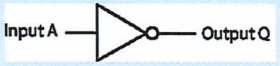
Disadvantages

- More expensive per volume of storage
- Reliability might be an issue if you do a lot of writing

Boolean Logic

NOT gate - The output is the opposite of the input

$$Q = \bar{A}$$
$$Q = NOT\ A$$



NOT truth table

Input	Output
0	1
1	0

AND gate - has two inputs and will have a true output if the two inputs are true otherwise the output will be false

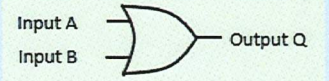
$$Q = A.B$$
$$Q = A\ AND\ B$$

AND truth table

Input - A	Input - B	Output
0	0	0
1	0	0
0	1	0
1	1	1

OR gate - has two inputs and will have a true output if either or both the inputs are true

$$Q = A + B$$
$$Q = A\ OR\ B$$

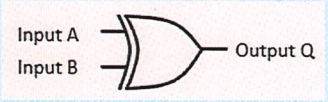


OR truth table

Input - A	Input - B	Output
0	0	0
1	0	1
0	1	1
1	1	1

XOR gate - has two inputs and will have a true output if either the inputs are true but not both

$$Q = A \oplus B$$
$$Q = A\ XOR\ B$$



OR truth table

Input A	Input B	Output
0	0	0
1	0	1
0	1	1
1	1	0

Converting a truth table to a logic circuit

There is a general approach to converting a truth table into a logic circuit.

We consider only the lines with an output of 1.
We take in the input of each and then AND.

We then OR between each statement such that
(NOT A AND B) OR (A AND NOT B). We can then draw the logic circuit.

Worked example: What is the logic circuit for the following truth table

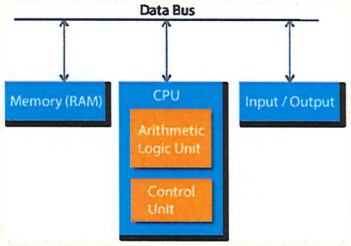
Input - A	Input - B	Output
0	0	0
1	0	1
0	1	0
1	1	1

(A AND NOT B) OR (A AND A)

System Architecture

CPU (Computer Processing Unit) or processor Fetches, decodes and executes instructions and performs logical and arithmetic operations.

Von Neumann architecture is the stored program concept, where program instructions and the data to be processed can be stored in the same memory.



Components of a CPU

Bus Wires through which data and instructions are transferred between computer components

Clock keeps all the CPU components synchronised

Arithmetic Logic Unit (ALU) Every operation takes place here. This is where the arithmetic (eg adding two binary numbers) and logic operations (eg checking to see if one number is bigger than another) take place.

Control Unit Decode the machine code instruction so that the ALU knows what to do with the instruction. Controls and monitors data transfer between different input and output hardware components

Factors affecting CPU performance

Clock speed is the number of cycles that a processor carries out per second. Each cycle of the CPU allows a single action (instruction) to be carried out. The greater the clock speed, the greater the number of operations and the faster the computer will run.

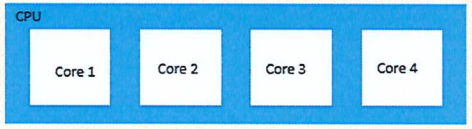
Number of processor cores A core is CPU in its own right. Nowadays most CPUs have multiple cores. Having multiple cores allows instructions to be carried out concurrently (at the same time), whereas a single core will only allow carry out instructions in serial (one at a time).

Latency Delay in transfer of data between components

Cache size Cache is a volatile memory store on the processor. Cache is much faster but smaller that RAM. Frequently used data and instructions within an application can be stored in cache instead of fetching from RAM which is quite slow. The bigger the cache the greater the volume of data and instructions that can be stored thereby reducing latency and improving performance of the CPU.

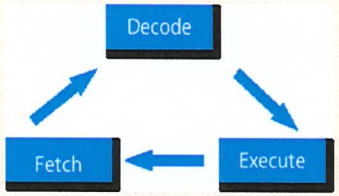
Cache type There are three levels of cache. Cache Level is a trade off between size and speed

- *Level 1 Cache* closest to the CPU and is the fastest cache (lowest latency), but does not have much capacity
- *Level 2 Cache* – is slower and further away from the CPU than L1 cache so latency is greater, but has more storage capacity.
- *Level 3 Cache* is the slower than L1 and L2 cache; much faster than RAM; has greater capacity than L1 and L2 cache.



Fetch execute cycle

1. Instructions are loaded into memory
2. Processor fetches the instruction from the main memory
3. Instruction is decoded so the CPU knows what to do with the instruction
4. Processor then executes the instruction
5. Result of the instruction can be stored in memory
6. Next instruction is then fetched from main memory and the cycle repeats itself.



Classification of programming languages

High level programming languages are closer to human language and is therefore easier to understand. A translator is used to convert the instructions into code that the computer understand. High level languages allow programs to be written that is independent of the type of computer. High level programming languages allow code to be written that is independent of the type of computer system. It is up to the compiler to translate the code into the right machine code for a particular code. There is a huge variety of high level programming languages, and the choice depends on the application.

Low level programming languages refer to machine code and assembly language. The Low level refers to low level of abstraction. The low level language is close to the language understood by the computer where operations map to the instruction in the processor instruction set. However it is difficult for humans to understand. Low level languages are appropriate for developing new operating systems, embedded systems and hardware device drivers

Machine code is expressed in binary values 0 and 1. This is the language that computers understand. All codes whether assembler or high level programming languages need to be translated into machine code. Machine code is specific to a processor.
Machine code instructions are made up of two parts the operator and the operand. The processor decodes the operator to identify the task that is to be carried out (eg. Add, load). The operand is the value or memory address that that instruction is to be operated on

Machine code instruction	
Operator	Operand
0011	10010100

Assembly language provides basic computer instructions for programs to run. There is a one to one relationship between machine code and assembly code instructions. One assembly language instruction maps to one machine code instruction, thus the structure of assembly language and machine code is the same, but where machine codes uses 0 and 1 which are very difficult for programmer to understand, assembly language uses mnemonics which is easier for the programmer.

Assembly language sample Instruction set

```
LOAD #23 # Load from RAM to processor
MOV a 23 # Transfer in number 23 into the variable a
ADD 2 3  # Add 2 values
STORE    # store data in RAM
```

Each type of processor has its own instruction set and therefore its own assembly language and machine code. So Assembly code written for one type of processor will not run on another.

Low level languages versus high level languages

	Advantages	Disadvantages
Low level	Produce code that is faster and better optimised than high level languages. Appropriate for developing new operating systems, embedded systems and hardware device drivers	Difficult to understand and modify Assembly code is written for a specific processor architecture, and so is not portable to other computer architectures
High level	High level programming languages allow code to be written that is more portable. Thus code can be run on different of the types of computer system with different processor architecture. Easier to understand Easier to modify	Needs a translator run slower because of the layers of abstraction and there is inefficiency in translator.

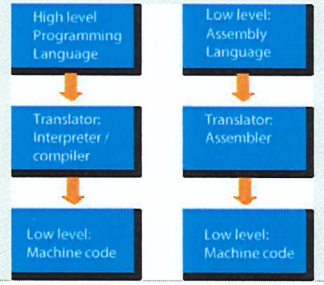
Program translators allow programs to be translated into machine code so the than programs can be run on a computer.

Interpreter converts high level languages into machine code one instruction at a time on-the-fly while the program is running. Each instruction is converted to machine code once the previous instruction has been executed. Interpreters are good for debugging code because the program stops as soon as the error has been found. However running code this way is much slower running compiled code. The machine code is not saved.

Compiler A program that converts high level languages into machine code before the program is run. A compiler saves the machine code,

so the source code is no longer needed A compiler allows a program to be run faster than interpreted code. Software is normally distributed as compiled machine code. For proprietary software this is good because other people cannot copy the code and use it for their own applications.

Assembler Assembler converts assembly language instructions into machine code.



Programming - Python

Comment – Text within the code that is ignored by the computer. A Python comment is preceeded by a #.

```
# This is an example of a comment
```

Output – Processed information that is sent out from a computer

Python	Pseudocode

`print("Hello World!")`
Hello World!
`print("Hello", "World!")`
Hello World!
`print("Hello"+"World!")`
HelloWorld!
`print("Hello\nWorld!")`
Hello
World!
	`OUTPUT "Hello World"`

Input – Data sent to a computer to be processed

`print("Enter name")`	`OUTPUT "Enter name"`
`name=input()`	`name ← USERINPUT`
`print("Hello", name)`	`OUTPUT "Hello", name`
`print("Enter age")`	`OUTPUT "Enter age"`
`age=int(input())`	`age ← USERINPUT`

Assignment - The allocation of data values to variables, constants, arrays and other data structures so that the values can be stored.

- *Variable* – Value that can change during the running of a program. By convention we use lower case to identify variables (eg `a=12`)
- *Constant* – Value that remains unchanged for the duration of the program. By convention we use upper case letters to identify constants. (e.g. `PI=3.141`)

Data Types

Integer	`age = 12`	`age ← 12`
Float (real) number	`height = 1.52`	`height ← 12`
Character	`a = 'a'`	`a ← 'a'`
String – multiple characters	`name = "Bart"`	`name ← "Bart"`
Boolean (true/false)	`a = True` `b = False`	`a ← True` `b ← False`

Arithmetic Operators

Add	`7 + 2`	`= 9`	`7 + 2`
Subtract	`7 - 2`	`= 5`	`7 - 2`
Multiply	`7 * 2`	`= 14`	`7 * 2`
Divide	`4 / 2`	`= 2`	`4 / 2`
power	`2 ** 3`	`= 8`	`2 ** 3`
Integer division	`7 // 2`	`= 3`	`7 DIV 2`
Modulus (remainder)	`7 % 2`	`= 1`	`7 MOD 2`

Relational Operators – Allows the Comparison of values

Less than	`<`	`<`	`7<2`	`-> False`
Greater than	`>`	`<`	`7 > 2`	`-> True`
Equal to	`==`	`==`	`7==2`	`-> False`
Not equal to	`!=`	≠ or <>	`7!=2`	`-> True`
Less than or equal to	`<=`	≤	`7<=2`	`-> False`
Greater than or equal to	`>=`	≥	`7>=2`	`-> True`

Boolean Operators

`AND`	`and`	`7 < 2 and 1 < 2`	`-> False`
`OR`	`or`	`7 < 2 or 1 < 2`	`-> False`
`NOT`	`not`	`not 7 < 2`	`-> True`

Sequencing represents a set of steps. Each line of code will have some operation and these operations will be carried out in order line-by-line

Using + operator for adding	

`a = 1`
`b = 2`
`c = a + b`
`print(c)  -> 3`
	`a ← 1`
	`b ← 2`
	`c ← a + b`
	`OUTPUT c`

Using + operator for concatenation	

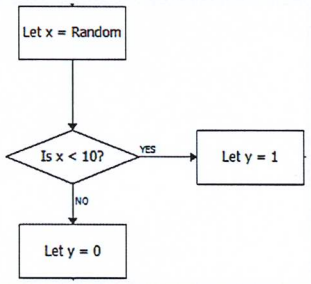
`a = 'Hello '`
`b = 'World'`
`c = a + b`
`print(c) -> Hello World`
	`a ← 'Hello '`
	`b ← 'World'`
	`c ← a + b`
	`OUTPUT c`

Random number

Random integer	`import random` `random.randint(0,9)`	`RANDOM_INT(0,9)`
Choice	`random.choice('a','b','c')`	
Random value from 0 to 1	`random.random()`	

Selection represents a decision in the code according to some condition. The condition is met then the block of code is executed otherwise it is not. Often alternative blocks of code are executed according to some condition.

```
x=RANDOM_INT()
IF  x < 10 THEN
 y=1
ELSE
 y=0
ENDIF
```



IF ...	`IF i > 2 THEN`
	`j ← 10`
	`ENDIF`
		`if i  > 2:`
		`j=10`

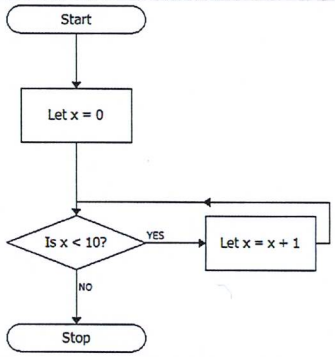
IF ... ELSE ...	`IF i > 2  THEN`
	`j ← 10`
	`ELSE`
	`j ← 3`
	`ENDIF`
		`if i > 2:`
		`j=10`
		`else:`
		`j=3`

IF ... ELSE IF ... ELSE	`IF i ==2 THEN`
	`j ← 10`
	`ELSE IF i==3`
	`j ← 3`
	`ELSE`
	`j ← 1`
	`ENDIF`
		`if i ==2:`
		`j=10`
		`elif i==3:`
		`j=3`
		`else:`
		`j=1`

Iteration Sometimes we wish the code to repeat a set of instructions

`WHILE` loops are used when the we do not know beforehand the number of iterations needed and this varies according to some condition.

```
x = 0
while (x < 10):
      x = x + 1
```



`while True:`
`print("Hello World")`
	`WHILE TRUE`
	`OUTPUT "Hello World"`
	`ENDWHILE`

`a=0`
`while a<4:`
`print(a)`
`a=a+3`
	`a ← 0`
	`WHILE a < 4`
	`OUTPUT a`
	`a ← a + 3`
	`ENDWHILE`

`FOR` loops are used when we know before hand the number of iterations we wish to make.

`for a in range(3):`
`print(a)`
	`FOR a ← 0 TO 3`
	`OUTPUT a`
	`ENDFOR`

Nested structures - Use constructs (e.g. `WHILE, FOR, IF`) inside another.

use a nested FOR loop to print out a grid	`for i in range (10):`
	`  for i in range (10):`
	`  print ("x ",end="")`
	`  print()`

Use a nested while and if to print out only even numbers	`i=0`
	`while i<51:`
	`  if (i%2==0):`
	`  print(i)`
	`  i=i+1`

Lists

Create a list	`shapes=["square","circle"]`
Access element by index pos	`shapes[1] -> circle`
Append item to list	`shapes.append("triangle")`
Remove item from list	`shapes.remove("circle")`
Remove item from list by index	`shapes.pop(1)`
Insert item into list	`shapes.insert(2,"rectangle")`
Number of elements in a list	`len(shapes)`
Get index pos of item in list	`shapes.index("triangle")`

Concatenating lists	`shapesGroup1["square","circle"]`
	`shapesGroup2=["triangle"]`
	`shapes=shapesGroup1+shapesGroup2`

Loop through list	`for i in range(len(shapes)):` `  print(shapes[i])`
Reverse elements in a list	`shapes.reverse()`
Order elements in a list	`shapes.sort()`

2D lists - A list if lists

Create a 2D list	`d = [ [23, 14, 17], [12, 18, 37], [16, 67, 83]]`

Another way to create a 2D list	`a = [23, 14, 17]`
	`b = [12, 18, 37]`
	`c = [16, 67, 83]`
	`d = [a,b,c]`

Access element by index position	`d[1][2] -> 37`

Strings

Get length of a string	`len("Hello")`	`LEN("Hello")`
Character to character code	`ord("a") -> 97`	`ORD("a")`
Character code to character	`chr(101) -> 'e'`	`CHR(101)`
String to integer	`a=int("12")`	`a=INT("12")`
String to float	`a=float("12.3")`	`a=FLOAT("12.3")`
integer to string	`a=str(12)`	`a=STR(12)`
real to string	`a=str(12.3)`	`a=STR(12.3)`

Concatenation -merge multiple strings together	`a="hello "`
	`b="world"`
	`c=a+b`
	`print(c) ->`
	`hello world`

Return the position of a character If there is more than 1 of the same character the position of the first character is returned.	`student = "Hermione"` `student.index('i')`
Find the character at a specified position	`student = "Hermione"` `print(student[2]) -> r`

sub strings - select parts of a string

Example	`student="Harry Potter"`	
Output the first two characters	`print(student[0:2])`	Ha
Output the first three characters	`print(student[:3])`	Har
Output characters 2-4	`print(student[2:5])`	Rry
Output the last 3 characters	`print(student[-3:])`	Ter
Output a middle set of characters	`print(student[4:-3])`	y Pot

*A negative value is taken from the end of the string.

Subroutines are a way of managing and organising programs in a structured way. This allows us to break up programs into smaller chunks.

- Can make the code more modular and more easy to read as each function performs a specific task.
- Functions can be reused within the code without having to write the code multiple times.

- **Procedures** are subroutines that do not return values
- **Functions** are subroutines that have both input and output

Procedure: No input parameters or return	`SUB greeting()`
	`  OUTPUT "hello"`
	`ENDSUB`
		`def greeting():`
		`  print("hello")`
		
		`call: greeting()`

Procedure: One input parameter, no return	`SUB greeting(name)`
	`  OUTPUT "Hello",name`
	`ENDSUB`
		`def greeting(name):`
		`  print("Hello",name)`
		
		`greeting("grey")`

Function: 1 input parameter, and 1 return value	`SUB add(n)`
	`  a ← 0`
	`  FOR a ← 0 TO n`
	`  a ← a + n`
	`  ENDFOR`
	`  RETURN a`
	`ENDSUB`
		`def add(n):`
		`  a=0`
		`  for a in range(n+1):`
		`  a=a+n`
		`  return a`

Function: Two input parameters, and 1 return value	`SUB (num1,num2)`
	`  sum=num1+num2`
	`  return sum`
		`def add(num1,num2):`
		`  sum=num1+num2`
		`  return sum`
		
		`greeting(1,2)`

The **scope** of a variable determines which parts of a program can access and use that variable.

A **global variable** is a variable that can be used anywhere in a program. The issue with global variables is that one part of the code may inadvertently modify the value because global variables are hard to track.

A **local variable** is a variable that can only be accessed within a certain block of code typically within a function. Local variables are not recognized outside a function unless they are returned. There is no way of modifying or changing the behavior of a local variable outside its scope.

Global variables need to defined throughout the running of the whole program. This is an inefficient use of memory resources. Local variables are defined only when they are needed an so have less demand on memory. Local variables only exist within the subroutine.

Reading and writing files

Open file Whatever we are doing to a file whether we are reading, writing or adding to or modifying a file we first need to open it using:

`open(filename,access_mode)`

There are a range of access mode depending on what we want to do to the file, the principal ones are given below:

Access Mode	Description
r	Opens a file for reading only
w	Opens a file for writing only. Create a new file if one does not exist. Overwrites file if it already exists.
a	Append to the end of a file. Create a new file if one does not exist.

Reading text files

read – Reads in the whole file into a single string	`f=open("filetxt","r")`
	`print(f.read())`
	`f.close()`

readline – Reads in each line one at a time	`f=open("file.txt","r")`
	`print(f.readline())`
	`print(f.readline())`
	`print(f.readline())`
	`f.close()`

readlines – Reads in the whole file into a list	`f=open("file.txt","r")`
	`print(f.readlines())`
	`f.close()`

Writing text files

Write in single lines at a time	`file=open("days.txt",'w')`
	`file.write("Monday\n")`
	`file.write("Tuesday\n")`
	`file.write("Wednesday\n")`
	`file.close()`

Write in a list	`say=["How\n","are\n","you\n"]`
	`file=open("say.txt",'w')`
	`file.writelines(say)`
	`file.close()`

Data Validation Routines

Check if an entered string has a minimum length	`OUTPUT "Enter String"`
	`s ← USERINPUT`
	`IF LEN(S) > 5 THEN`
	`  OUTPUT "STRING OK"`
	`ELSE`
	`  OUTPUT "TOO SHORT"`
	`ENDIF`

Check is a string is empty	`OUTPUT "Enter String"`
	`s ← USERINPUT`
	`IF LEN(S) == 0 THEN`
	`  OUTPUT "EMPTY STRING"`
	`ENDIF`

Check if data entered lies within a given range	`OUTPUT "Enter number" s num ← USERINPUT`
	`IF num > 1 AND num < 10`
	`  OUTPUT "Within range"`
	`ENDIF`

Authentication Routine

```
OUTPUT "Enter Username"
username ← USERINPUT
OUTPUT "Enter Password"
password ← USERINPUT

WHILE username != "bart" OR password !="abc"

 OUTPUT "Login failed"
 OUTPUT "Enter Username"
 username ← USERINPUT
 OUTPUT "Enter Password"
 password ← USERINPUT

ENDWHILE

OUTPUT "Login Successful"
```

Debugging

Syntax errors – Errors in the code that mean the program will not even run at all. Normally this is things like missing brackets, spelling mistakes and other typos.

Runtime errors – Errors during the running of the program. This might be because the program is writing to a memory location that does not exist for instance. eg. An array index value that does not exist.

Logical errors - The program runs to termination, but the output is not what is expected. Often these are arithmetic errors.

Test data

Code needs to be tested with a range of different input data to ensure that it works as expected under all situations. Data entered need to be checked to ensure that the input values are:

- within a certain range
- in correct format
- the correct length
- The correct data type (eg float, integer, string)

The program is tested using normal, erroneous or boundary data.

Normal data - Data that we would normally expect to be entered. For example for the age of secondary school pupils we would expect integer values ranging from 11 to 19.

Erroneous data - Data that are input that are clearly wrong. For instance, if some entered 40 for the age of a school pupil. The program should identify this as invalid data but at the same time should be able to handle this sensibly which returns a sensible message and the program does not crash.

Boundary data - Data that are on the edge of what we might expect. For instance if someone entered their age as 10, 11, 19 or 20.