Programming - Python

Comment – Text within the code that is ignored by the computer.

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

Inputs & Outputs

Output – Processed information that is sent out from a computer

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

Input – Data sent to a computer to be processed

```
print("Enter name")
name=input()
print("Hello", name)
```

Variables & Constants

Identifier: name of the variable

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

Variable – Value that is stored in a memory location. It can change during runtime.

```
num =12
name = "Sam"
found = False
```

Constant – Value that remains unchanged for the duration of the program. By convention we use upper case letters to identify constants.

```
PI=3.141
```

Variable Scope

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

A **global variable** is used anywhere in a program. Global variables need to defined throughout the running of the whole program. This is an inefficient use of memory resources. The issue with global variables is that one part of the code may inadvertently modify the value because global variables are hard to track.

Local variables only exist within the subroutine. Local variables are not recognized outside a function unless they are returned. Local variables are defined only when they are needed so have less demand on memory.

Data Types

Integer	Whole number	age = 12
Float/real	A number with a decimal point	height = 1.52
Character	A single letter, symbol or number	a = 'a'
String	multiple characters	name = "Bart"
Boolean	Has two values: true of false	a = True b = False

Arithmetic Operators

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

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	!=	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

Counters & Flags

A **counter** in a program can be used to increment a variable by 1 or decrement by 1

```
counter = 0
while counter < 3:
  counter+=1
```

A **flag** is useful for exiting a loop or returning a True/False from a function.

```
flag = True
while flag: # this means while flag is True
  word = input()
  if  word == 'cat':
    flag = False
```

Programming Constructs

Sequencing represents a set of steps - out in order line-by-line

Selection represents a decision in the code according to some condition , for example: IF … ELSE …Switch case statement

Iteration to repeat a set of instructions for example: WHILE/ FOR

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

```
a=0
while a<4:
  print(a)
  a=a+1
```

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

```
for a in range(3):
  print(a)
```

Random number

```
import random
random.randint(0,9)
random.choice('a','b','c')
```

Subprogram

A subprogram is a self-contained block of code.
A subprogram performs a specific/dedicated task
It can be 'called' by the main program or other subprograms, when it is needed

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 easier to read as each function performs a specific task.

Exam: Explain one benefit of using subprograms.
The subprogram may be used more than once in a program (1) so that writing, debugging, testing will save time (1) The subprogram performs one specific/contained task (1) so it can be moved away from the main program code (1) Subprograms for common tasks can be stored in libraries and reused in other programs (1) so that they don't have to be rewritten (1)

Functions can be reused within the code without having to write the code multiple times. **Functions will return a value.**

Two **input parameters**, and 1 return value

```
function add(num1,num2):
    sum=num1+num2
    return sum
greeting(1,2)
```

Procedures are subroutines that do not return values

```
procedure greeting():
    print("hello")
greeting()
```

Data Structures

Lists are used to store a collection of elements.

```
shapes=["square","circle","rectangle"]
```

Lists have an **index** starting at 0

0	1	2
square	circle	rectangle

Describe an array - is a collection of items in a list that are all the same data type.

Describe a record. - collection of items in a list where each element/field may be a different data type.

Iteration and arrays – Used to loop over every item in a data structure

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")

Loop through list	for i in range(len(shapes)):
	  print(shapes[i])
	for shape in shapes:
	  print(shape)

Reverse elements in a list	shapes.reverse()
Order elements in a list	shapes.sort()

2D lists - A list of lists

Index	0	1	2
0	23	14	17
1	12	18	37
2	16	67	83

Create a 2D list:

```
d = [ [23, 14, 17], [12, 18, 37], [16, 67, 83]]
```

Access element by index position:

```
d[1][2]
>>37
```

Reading and writing files

open(filename,access_mode)

Access Mode

r	Opens a file for reading only
w	Opens a file for writing only.
a	Append to the end of a file.

Reading text files
read – Reads in the whole file into a single string

```
f=open("filetxt","r")
print(f.read())
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

```
f=open("days.txt",'w')
f.write("Monday\n")
f.write("Tuesday\n")
f.write("Wednesday\n")
f.close()
```

Append to a file

```
day =[""]
f=open("days.txt",'a')
f.write(day)
f.close()
```

Strings

Get length of a string	len("Hello") -> 5
String to integer	a=int("12")
String to float	a=float("12.3")
integer to string	a=str(12)
Concatenation -merge multiple strings together	a="hello " b="world" c=a+b print(c) ->hello world
Return the position of a character	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 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
Make a string all uppercase	student="Harry Potter".upper()
Make a string all lowercase	student="Harry Potter".lower()

Data Validation

Check if an entered string has a **minimum length**

```
print "Enter String"
s = input()
if len(s) > 5 T
   print ("STRING OK")
else
   print("TOO SHORT")
```

Check is a string is **empty**

```
print("Enter String")
s = input()
if len(s) == 0
    print( "EMPTY STRING")
```

Check if data entered lies within a **given range**

```
print("Enter String")
s = input()
if num > 1 AND num < 10
   print("Within range")
```

Authentication

Check that a username and password exist

```
username = input("Enter Username")
password = input( "Enter Password")

while username != "bart" OR password !="abc"
   print( "Login failed")
   username = input("Enter Username")
   password = input( "Enter Password")

print("Login Successful")
```

ERRORS & TESTING

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. e.g. 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

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.

Maintainability

Indentation
Comment Code
Naming of variables and sub-routines

IDE Features

- **Code editor -** The **code editor** is a text edit area that allows developers to write, edit and save a document of code.
- **Auto-completion** (or code completion). This is designed to save time while writing code. As you start to type the first part of a function, it suggests or completes the function and any arguments or variables.
- **Syntax checks**. This recognises incorrect use of syntax and highlights any errors.
- **Translator**. This compiles or interprets the code.
- **Steppers and breakpoints**. This is used to step through code line by line or stop at certain point in the program
- **Auto documentation**. This explains the function and purpose of the code, eg by noting the modules and variables used, and its expected behaviour, and collates this into a text file that can be used by other developers to understand how and why the code was created.
- **Libraries**. These functions can be imported and used at the start of the program code. For example, in Python the Turtle Graphics library provides access to some simple drawing and graphics tools.
- **Debugger**. This is a program within the IDE that is used to detect errors. If the debugger detects errors, it may suggest what the type of error is and what line it is on.

Example programs

Search a file for a word:

```
word = "jane"
f=open("filetxt","r")
for line in f:
  if word == line.strip():
    print("found")
f.close()
```

Search a file to check username and password exists:

```
username = input()
password = input()
f=open("filetxt","r")
found =False
for line in f:
  user = line.split(',')
  if username == user[0] and password ==
user[1].strip():
    found = True
if found == False:
  print("Access Denied")
else:
  print("Authenticated")
f.close()
```

Check if a password is more than 8 characters

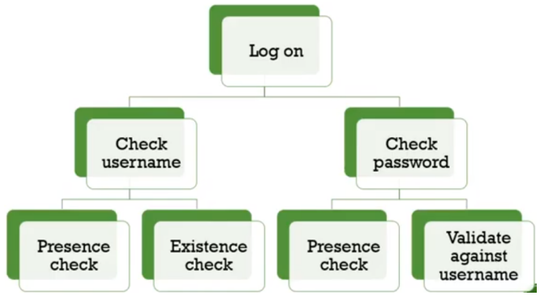
```
def checkLength(password):
   flag = False
   if len(password) > 8:
      flag = True
   return flag

password = input()
checkpassword = checkLength(password)
if checkpassword:

  print("password approved")

else:
  print("Password must be 8 or more characters")
```

Computational Thinking

Decomposition: breaking down a complex problem or system into smaller, more manageable parts.

When logging on, the program should…

- Ask the user for their username
- Ask the user for their password
- Does the username exist? Is it valid?
- If not, ask for it again
- Is the password correct?
- If not, ask for it again
- Have they used up all 3 attempts?
- Does the account then need to be locked?



Why?
It is usually easier to solve several smaller problems
Different parts of the program can be shared between the programming team to speed up development.

Abstraction: focuses on the important information only, ignoring irrelevant detail.

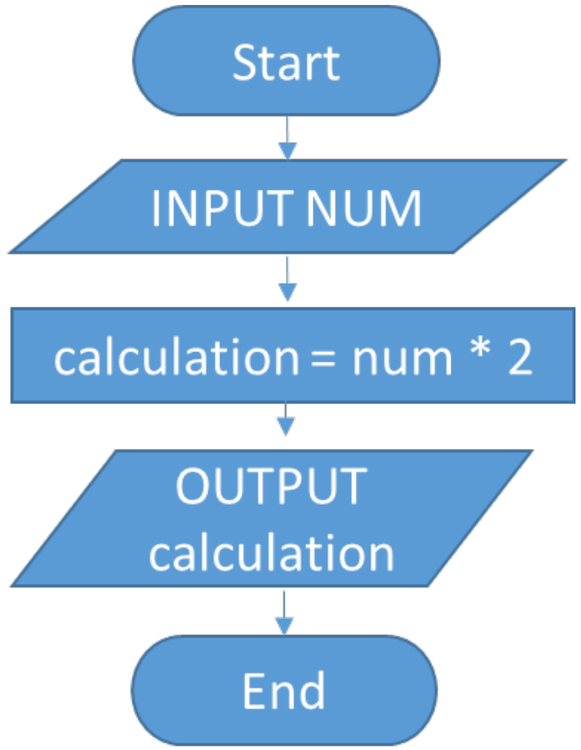
Why?
Programmers can focus only on the important details of a problem (1) because abstraction allows them to ignore (1) any detail that is not relevant.

Pattern Recognition: looking for similarities among and within problems.

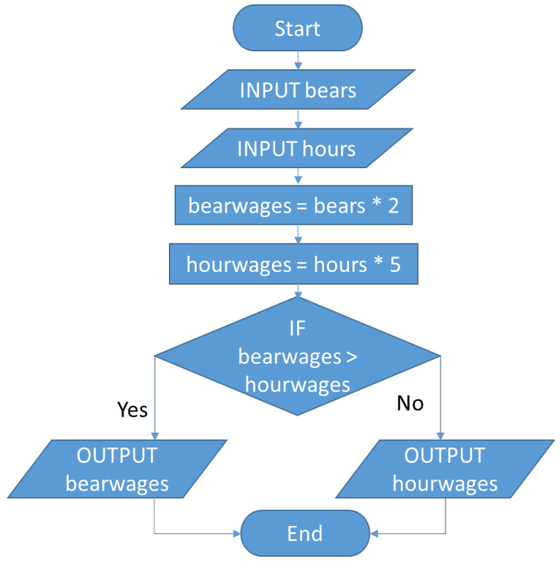
Algorithms: a step-by-step solution to the problem, or the rules to follow to solve the problem.

Symbol	Name	Usage
	Line	Represents the flow from one component to the next
Process	Process	An action
Subroutine	Subroutine	Calls a subroutine
Input/Output	Input/Output	An input or output
Decision	Decision	A yes/no/true/false decision
Start/Stop	Terminator	The start or end of the process

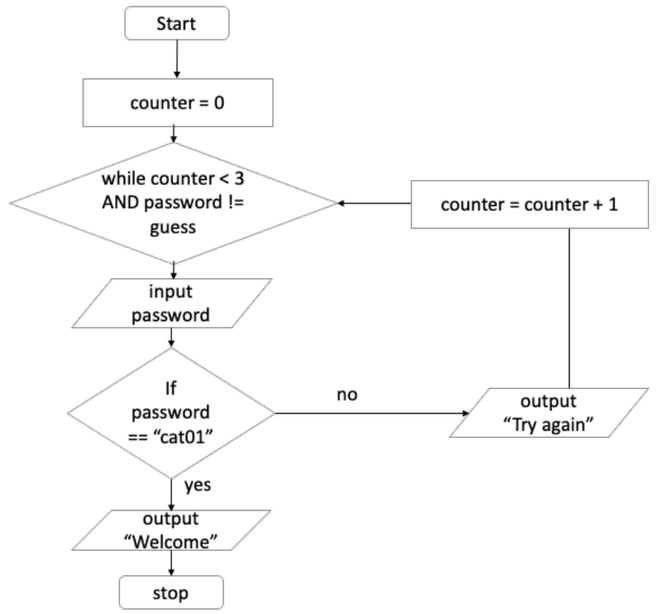
Sequence



Selection



Iteration



Tracing an algorithm

The purpose of tracing an algorithm

It helps the programmer visualise the steps in a program / find errors/bugs / check the algorithm gives the correct output (1) because they can see the value a variable holds at a given point in an algorithm (1)

Margaret owns an ice-cream shop.

This program manipulates sales figures from Margaret's shop.

```
num = 0
x = 999
y = 0
line = ""

f = open("SalesFile.txt", "r")
for line in f:
    num = int(line)
    if num < x:
        x = num
    if num > y:
        y = num
print(x, y)
f.close()
```

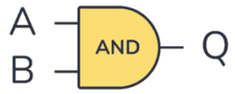
The only inputs from the file to the program are 355, 554, 199 and 409.
Complete the trace table showing the execution of the program with these four inputs. You may not need to fill in all the rows in the table.

num	x	y	Display
0	999	0	
355	355	355	
554		554	
199	199		
409			
			199 554

Logic

Truth tables and logic diagrams

AND (A AND B)

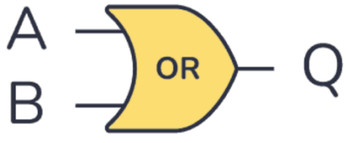


A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

NOT (NOT A)

A	Q
0	1
1	0

OR (A OR B)



A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1

Example

Algorithms control physical devices using logical operators.
A security system turns on a floodlight when the sunlight falls below a certain
level (S) and a movement sensor is activated (M).

Complete the truth table.

S	M	S AND M
0	0	0
0	1	0
1	0	0
1	1	1

Example 2:

Complete the truth table.

A	B	C	(A AND B)	(NOT C)	(A AND B) OR (NOT C)
0	0	0	0	1	1
0	0	1	0	0	0
0	1	0	0	1	1
0	1	1	0	0	0
1	0	0	0	1	1
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	1	0	1