Subject: **Computer Science**

Checklist for: Year 10 Examination, April 2022

Programming – Python

What do you need to know?	Where can you find the information?	First Revision Date	RAG	Second Revision Date	RAG	Third Revision Date	RAG	Fourth Revision Date	RAG
Structuring programs into modular parts with clear documented interfaces to enable them to design appropriate modular structures for solutions	https://www.bbc.co.uk/bitesize/guides/zh66pbk/revision/1 https://searchsoftwarequality.techtarget.com/definition/structured-programming-modular-programming								
Including authentication and data validation systems/routines within their computer programs	https://www.bbc.co.uk/bitesize/guides/zfnny4j/revision/1 https://corporatefinanceinstitute.com/resources/knowledge/data-analysis/data-validation/								
Writing, debugging and testing programs to enable them to develop the skills to articulate how programs work and argue using logical reasoning for the correctness of programs in solving specified problems	https://www.bbc.co.uk/bitesize/guides/zb33rwx/revision/1 https://www.aqa.org.uk/subjects/computer-science-and-it/gcse/computer-science-8520/subject-content/aspects-of-software-development								
Designing and applying test data (normal, boundary and erroneous) to the testing of programs so that they are familiar with these test data types and the purpose of testing refining programs in response to testing outcomes.	https://www.bbc.co.uk/bitesize/guides/z4cck2p/revision/1 https://www.bbc.co.uk/bitesize/guides/z8n3d2p/revision/7								

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

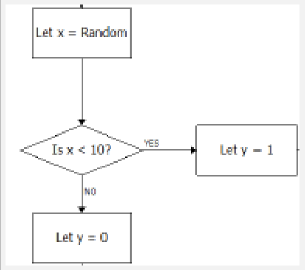
Random number

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

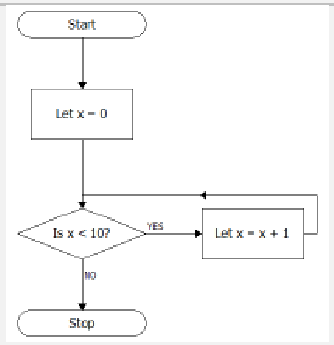
IF ... ELSE ...	IF i > 2 THEN
	 j ← 10
	ELSE
	 j ← 3
	ENDIF
		if i > 2:
		  j=10
		else:
		  j=3

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.

Q1.

Write a Python program that inputs a password and checks if it is correct.

Your program should work as follows:

- input a password and store it in a suitable variable
- if the password entered is equal to `secret` display the message `Welcome`
- if the password entered is not equal to `secret` display the message `Not welcome`.

You **should** use meaningful variable name(s), correct syntax and indentation in your answer.

The answer grid below contains vertical lines to help you indent your code accurately.

(Total 5 marks)

Q2.

Write a Python program that allows a taxi company to calculate how much a taxi fare should be.

The program should:

- allow the user to enter the journey distance in kilometres (no validation is required)
- allow the user to enter the number of passengers (no validation is required)
- calculate the taxi fare by
    - charging £2 for every passenger regardless of the distance
    - charging a further £1.50 for every kilometre regardless of how many passengers there are
- output the final taxi fare.

You **should** use meaningful variable name(s), correct syntax and indentation in your answer.

The answer grid below contains vertical lines to help you indent your code accurately.

(Total 7 marks)

Q3.

Write a Python program that inputs a character and checks to see if it is lowercase or not.

Your program should work as follows:

- gets the user to enter a character and store it in a suitable variable

- determines if the entered character is a lowercase character
- outputs `LOWER` if the user has entered a lowercase character
- outputs `NOT LOWER` if the user has entered any other character.

You **should** use meaningful variable name(s), correct syntax and indentation in your answer.

The answer grid below contains vertical lines to help you indent your code accurately.

(Total 7 marks)

Q4.

Write a Python program that calculates an estimate of the braking distance in metres for a new model of go-kart that is travelling between `10` and `50` kilometres per hour (kph).

Your program should:

- keep asking the user to enter a speed for the go-kart until they enter a speed that is between `10` and `50` (inclusive)
- calculate the braking distance in metres by dividing the speed by 5
- ask the user if the ground is wet (expect the user to enter `yes` if it is)
- if the ground is wet, multiply the braking distance by 1.5
- output the final calculated braking distance.

You **should** use meaningful variable name(s), correct syntax and indentation in your answer.

The answer grid below contains vertical lines to help you indent your code accurately.

(Total 8 marks)

Subject Specific Vocabulary

The following list provides definitions of key terms used in our GCSE Computer Science 8525 specification.

Students should be familiar with, and gain understanding from, all these terms.

Variable declaration

Variables are defined as a space in memory, given a name, assigned a value that can be changed while a program is running. Once a variable is declared it is then assigned a value, this is called initialization.

```
Name ←"Bob Smith"
```

Constant declaration

Constants are similar in definition to variables, the difference in definition being a space in memory, given a name, assigned a value that **cannot** be changed while a program is running. Constants are identified in their declaration and should be capitalized:

```
CONST PI ← 3.14
```

Assignment

Assignment is a term used to show the setting of variables to values.

```
Answer ←Num1 + Num2
```

Selection

Selection is a blanked term to refer to a programming statement that allows the changing of the flow of the program, based on a condition that is met.

```
IF Game = "won" THEN
    PRINT("you have won the game")
END IF
```

The above shows an example of nested selection, that shows if the first condition is met, then it checks the second selection statement.

Iteration

Iteration is a blanket term to refer to a programming statement that repeats code (loops). This can be both count controlled or condition controlled.

Count controlled loop

A loop that has a definite number of times to run, this can be known as definite iteration. A for loop is an example of a Count Controlled loop.

```
FOR i ← 1 TO 5
   … Instructions here …
ENDFOR
```

This can also be achieved with a WHILE loop, as long as the loop as a set number of iterations.

Count controlled loop

A loop that has a definite number of times to run, this can be known as definite iteration. A for loop is an example of a Count Controlled loop.

```
FOR i ← 1 TO 5
   … Instructions here …
ENDFOR
```

This can also be achieved with a WHILE loop, as long as the loop as a set number of iterations.

```
WHILE I <= 5
     …instructions here…. I = I + 1
END WHILE
```

Condition controlled loop

Condition controlled loops are those in which the end of the loop is not known, it will continue indefinitely until the condition is met.

```
WHILE NotSolved
   … Instructions here …
ENDWHILE
```

The same effect can be achieved with a REPEAT…UNTIL loop, the benefit of which is that the program can enter the loop, gain an input then test the condition at the end, making it potentially more efficient.

```
REPEAT
   … Instructions here …
UNTIL Solved
```

Revision Activities

For Loops

Computer Experiment 2

This is how a for loop works:

```
n=10
for n in range(10,20):
    print n
    n=n+1


print'end of loop'
```

Run this program a few times, and for each run change the values in inside the bracket, and sometimes the number added to n.

Exercise 1

Write a program where the loop runs from 20 to 10 and print these value.

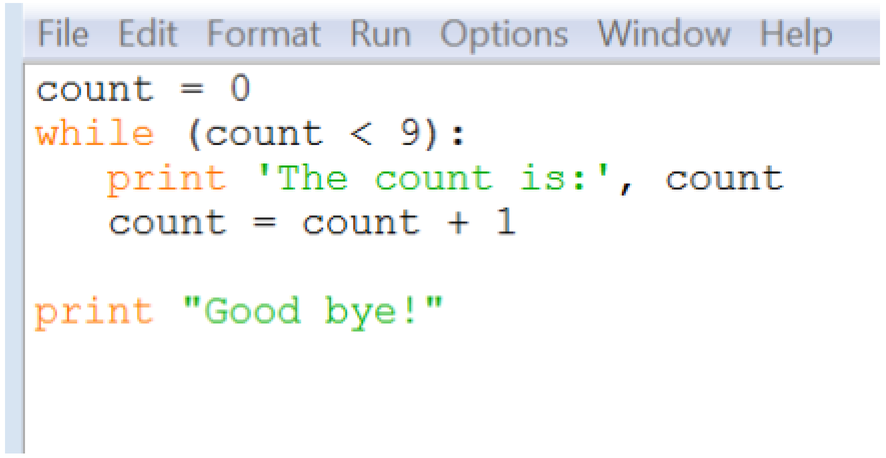
Exercise 2

Write a program where the user enter a number between 10 and 100. A loop runs between 1 and the value entered. Using % remainder print out all the factors of that number.

Exercise 2a

Modify the above program using conditional statements inside a loop, to determine if the enter number is a prime number. **Hint** design the program first using a flow diagram.

While loops:
While loops depend on a condition see the following program:

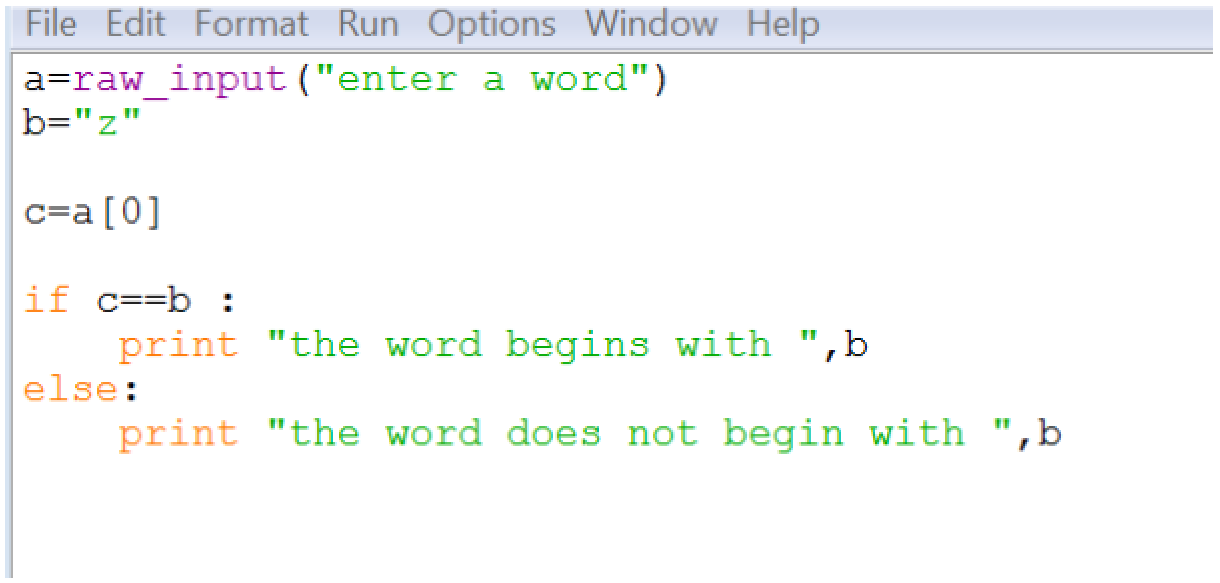


```
count = 0
while (count < 9):
   print 'The count is:', count
   count = count + 1

print "Good bye!"
```

Computer Experiment 3
Run the above program change the starting value of count, and the number 9, and run a few times.

Exercise 3
Write a program where a user enters a number n. The program prints the sentence “tell me why I don't like mondays?" n times. Can the program be written using both types of loops?

Exercise 4
Write a program where the user enters their full name, and a letter. The program counts the number of times the chosen letter is in the name. Hint you will need to use the len(string) statement. There is more than one way to write this program.

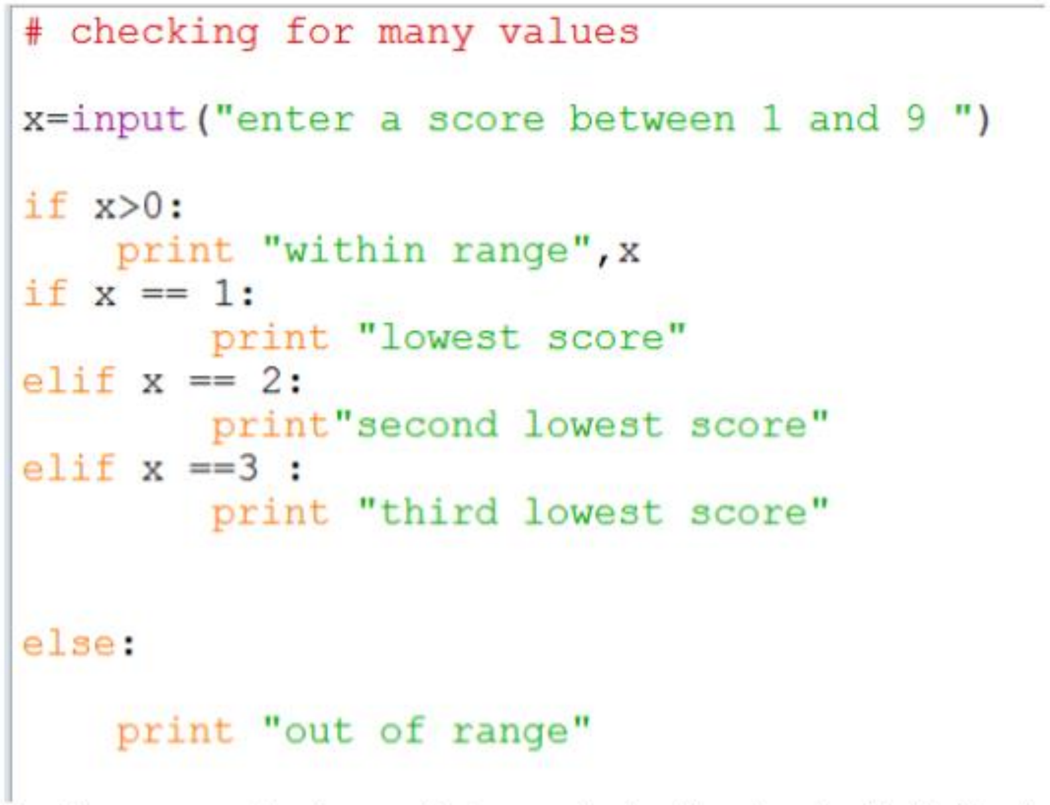


```
a=raw_input("enter a word")
b="z"

c=a[0]

if c==b :
    print "the word begins with ",b
else:
    print "the word does not begin with ",b
```

Computer Experiment 2
Run the above program a few times. Enter words starting with z and some that do not start with z. Is the program case sensitive? I.e. does capitalization make any difference?

Exercise 1
Write a program where the person enters their first name. The program determines the length of the name, and then determines if the name ends with the letter "a". If the name ends with an a, then the person wins a 1000 dollar prize. The program informs the person that they have or have not won the prize. You will need to use the len(string) method, look it up here:
https://www.tutorialspoint.com/python/python_strings.htm

Exercise 2
The rules for winning the prize have got stricter. The first name must now have more than 5 characters (as well as end in the letter a). Modify the program in Exercise 1. You will have to use nested if statements:
https://www.tutorialspoint.com/python/nested_if_statements_in_python.htm
Can you draw a flow diagram for your program?



```
# checking for many values

x=input("enter a score between 1 and 9 ")

if x>0:
    print "within range",x
if x == 1:
        print "lowest score"
elif x == 2:
        print"second lowest score"
elif x ==3 :
        print "third lowest score"


else:

    print "out of range"
```

Run the program a few times so that you understand how it works. Modify the above program so that the program prints a comment for every score between 1-9.

Exercise 4
A high level maths students takes a test. Write a program where the student enters their percentage mark. The program outputs their grade according to:
70% or more grade 7
Between 60 and 69 grade 6
Between 50 and 59 grade 5
Between 40 and 49 grade 4
Between 30 and 39 grade 3
Between 20 and 29 grade 2
Less than 20 a grade 1