An introduction to Python programming

Computer Science Department

2016-17

Your Name: _____

Year Group: _____

TABLE OF CONTENTS

Section 1 - First Steps

1.1 - Downloading Python

Python is a freely available programming language. You will need to install Python on your computer at home in order to revise and complete homework. **Version 3.4.2** is recommended. This is the same version we have installed in the school computers and will provide you with everything you need. **NB:** Do not download Python version 2 as this is an older version and some of the code in this booklet will not work! It must be version 3.4.2 or higher!
Please see below the links to the *Python Downloads* page for the two main operating systems:

- If you're running **Windows**, see all versions of Python releases for Windows here: https://www.python.org/downloads/windows/
- If you are using a **Mac**, see all versions of Python releases for Mac OS X here: https://www.python.org/downloads/mac-osx/

1.2 - The IDLE: Interactive Mode vs. Script Mode

The Python interpreter is called IDLE (**Integrated DeveLopment Environment**). We will use IDLE to write, save and run Python files (please, realise that Python files will be saved with the filename extension **.py**). IDLE has two different modes (or shells): **interactive** and **script**:

The interactive mode will give you immediate feedback for each statement, that is, what you type is immediately run when you click ENTER. Try typing 1+1 in and Python will respond 2. The prompter **>>>** only appears in the interactive mode. You need to be careful when using the interactive mode to avoid mistakes (syntax errors). Unfortunately if you have a mistake on the interactive mode, you would need to start the whole code from the beginning.

The script mode does not automatically display results; however, it is a better option when we need to write several lines of code. Once you are on the interactive mode click on **File** > **New File**. The window opening is in script mode! Once you have finished your code in this window, click on **Run** > **Run Module** (or **F5**). You will be prompted to save your work! Once the work is saved, the output will appear in the interactive mode. If there is a mistake, just go back to the script mode, correct it and run it again. All done!

1.3 - My first program: Maths Operators

Maths Operators. Order of Operators: **BIDMAS** (Brackets, Indices, Division, Multiplication, Addition and Subtraction)		
Operator	**Operation**	**Examples**
**	Exponent	2**3 = 8 (2 x 2 x 2)
%	Modulus (remainder)	23 % 7 = 2 (23/7 = 3, reminder is 2)
//	Integer division	23 // 7 = 3
/	Division	23 / 7 = 3.2857142857142856
*	Multiplication	5 * 3 = 15
+	Addition	5 + 3 = 8
–	Subtraction	5 – 3 = 2

You could start typing simple commands at the prompt (>>>). Try the following:

```
>>>24+2 ------------------------------> 26
>>>187-87
>>>3*7
>>>2+7*6
>>>(2+7)*6
>>>12345*6789
>>>2**3
>>>22/7
>>>22//7
>>>22%7
>>>3     +     8
>>>(5-2)*((7+2)/(9-6))
```

An **expression** is just a value (such us **2, "Peter"** or **False**). When we have a combination of "numerical" values and operators (such us **+, *** or **/**), Python always **evaluates** (that is, *reduces*) the expression to a single value. *Ex: When we write 2 + 2 in Python, this is an expression that is evaluated down (when we click ENTER) to a single value 4.*

1.4 - Single vs. Double quotation marks: Hello World

On the interactive shell, try the following code:

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

Which one is correct and which one is not? Do you know why? …

1.5 - Comments

Comments are descriptions (written by the programmer) that can only be read by humans. Computers cannot read comments, so anything you write as a comment will be ignored by the computer. There are 2 main types of "comments":

Single-line comments: They always start with # and the font will be in red by default. Example: # Task 1. Date 16th/09/16.

Multi-line comments: Use this if you need to write more than 4 lines in a row. They always start with ' ' ' (or " " ") and finish with ' ' ' (or " " "). The font will be in green by default. Multi-line comments are recommended to be in script mode only. Example:

```
''' This is an example
of a multi-line comment,
which is the one I should
use when have to write
more than 4 lines in a row'''
```

1.6 - How to find help

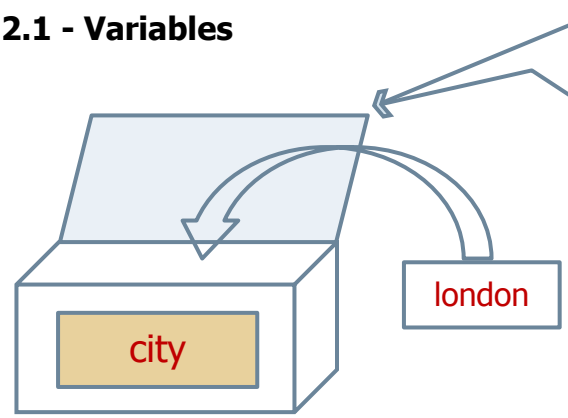
Let's cause an error on purpose by entering >>> "60" + 8 into the interactive mode window. You probably still don't know what this code means; however, I can tell you that "60" is a *string* while 8 is a *number* (or *integer*). We cannot add up strings and numbers, so the computer will give us an error:

```
>>> "60" + 8
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    "60" + 8
TypeError: Can't convert 'int' object to str implicitly
>>>
```

The error message ❷ appeared here because Python cannot understand the code. The traceback part ❶ of the error message shows the specific instruction and line number that Python had trouble with. You can search online for the exact error message entering **"TypeError: Can't convert 'int' object to str implicitly"** (including the quotes) into Google, and you finds links explaining what the error message means and what causes it.

Section 2 - Data Types

2.1 - Variables



A **variable** is a space (like a box) labelled with a **name** (like city) where you can store a **value** (like london).

An **assignment statement** consists of a variable name + an equal sign (*assignment operator*) + the value stored. ex: city = london

The Name: You can name a variable as you like; however, there are **4 rules** that you need to take into consideration: 1) It needs to be logical, 2) It needs to be one word, 3) It cannot start with a number, and 4) If you need more than one word or a space, you will use underscore (_).

The Value: When we create a variable, we say we are **declaring** a variable. When we give a value to that variable, we say we are **initialising** the variable. When we give a new value to that same variable, we say we are **overwriting** the variable, *ex: age = 9 / age = age + 1*.
NB: Please, see also the meaning of **expression** in page 3.

Data Types: There are 3 types of Data Types: **Strings** ("Peter"), **Numbers** (2) and **Boolean** (True/False).

2.2 - Strings: Concatenation and Reduplication

A **string** (or **str**) is any value that is considered to be a text. Always starts with quotation marks " (or `) and finish with " (or '). They could be:

- a letter: "W" (lower case or capital)
- a word: "Walker"
- a sentence: "What is your surname?"
- a paragraph: "My name is Peter Walker and **\n** I learn Computing in a secondary school in London. **\n** I love all type of new technologies" (**\n** is a line break or new line).
- or even a number (that is not considered a number): "18" (it looks like number 18; however, it is within quotation marks, so the computer will consider this to be text rather than a number).

Try the following code:

```
>>>addition1 = "2 + 2"
>>>addition 2 = 2 + 2
```

What is the difference between addition1 and addition2?

EXTRA - **Concatenation** and **Reduplication**. Try the following code and explain each case:

```
>>>"Peter" + "Maria" ⟶
>>>"Peter" * 4 ⟶
>>>"Peter" + 36 ⟶
>>>"Peter" * John ⟶
```

2.3 - Numbers: Integers, Floating-Points and Built-in Functions

There are two types of numbers:

Integers (or **int**) are whole numbers: -3, -2, -1, 0, 1, 2, 3, 4, 55, 100 …

Floating-Points (or **float**) are numbers with decimal places: -2.35, -1.5, 0.0, 3.2, 4.666 …

Try the following codes and explain the differences among the two cases:

>>>14/2 ⟶

>>>14//2 ⟶

Built-in functions: There are 68 built-in functions in Python, among which, some of the most useful ones are the **int ( )**, **print ( )**, **input ( )**, **float ( )**, **len ( )** and the **str ( )** functions.

Please, try the following codes:

```
>>>number = 2.25
>>>new_number = int (number)
>>>print (new_number)
```

The **int ( ) function** forces a number with decimal places to turn into a whole number. It will not round the number up; it just will take the whole number part. The **print ( ) function** will output the value of the variable inside the parentheses.

```
>>>age = int (input ("How old are you? "))
```

The **input ( ) function** will be expecting the user to insert a value and press ENTER. In this particular case, the value inserted needs to be a whole number; otherwise the result will be a ValueError. This situation is forced by **int ( ) function**.

```
>>>money = 9
>>>new_money = float (money)
>>>print (new_money)
```

In this case, the initial value is a whole number (or integer); however, once printed the **float ( ) function** is forcing the output to get decimal places.

```
>>>len ("hello")
>>>len ("hello my frieds")
>>>len (" ")
```

In these cases, the **len ( ) function** will count the number of characters in a string.

```
>>>print ("I have" + 10 + "pounds")
>>>print ("I have" + str(10) + "pounds")
```

The **str ( ) function** = "xxx" or 'xxx'. That is, we can use it to turn an integer into a string.

2.4 - Boolean: Comparison Operators and Boolean Operators

Boolean data type (named after mathematician George Boole) has only two values: **True** and **False**. These values are never written between quotation marks and always start with capital T or F followed by lowercase.

1) They can be used as expressions and can be stored in variables:

```
>>>red_light = False
>>>red_light
False
```

2) If you do not use the proper case, Python will give you an error message:

>>>red_light = false ~~(X)~~ >>>red_light = False (**OK**)

3) If you try to use True or False for variable-names, Python will give you an error message:

```
>>> False = 40
SyntaxError: can't assign to keyword
>>> false = 40
>>> false
40
```

Comparison Operators: They are used to compare two values and *evaluate down* (*reduce*) to a single Boolean value.

Comparison Op.	Meaning	Examples

==	Equal to	>>> 23 == 23 (True)
		>>> "Juan" == "Juan" (True)
		>>> True == True (True)
		>>> 68 == 68.0 (True)
		>>> 23 == 24 (False)
		>>> "Juan" == "juan" (False)
		>>> 68 == "68" (False)

!=	Not equal to	>>> 23 != 24 (True)
		>>> "Juan" != "John" (True)
		>>> True != False (True)
		>>> 23 != 23 (False)
		>>> True != True (False)

<	Less than	>>> 68 < 70 (True) >>> 68 < 68 (False)
>	Greater than	>>> 70 > 68 (True) >>> 68 > 70 (False)
<=	Less than or equal to	>>> myAge = 40 >>> myAge <=40 (True)
>=	Greater than or equal to	>>> myAge = 40 >>> myAge >=60 (False)

Important to notice:

```
>>> 60==60
True
>>> 60==60.0
True
>>> 60==60.00000
True
>>> 60=60.0
SyntaxError: can't assign to literal
>>> 60=60
SyntaxError: can't assign to literal
```

The **== operator** (equal to) asks whether two values are the same as each other

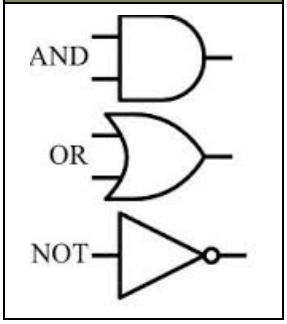
The **= operator** (assignment) only indicates the value of a variable

Boolean Operators:

The **and** operator evaluates an expression to True only if both Boolean values are True; otherwise, it evaluates to False.

```
>>> True and True
True
>>> True and False
False
>>> False and True
False
>>> False and False
False
```

There are three Boolean operators:



The **or** operator evaluates an expression to True if at least one of the two Boolean values is True. Only when both are false, it evaluates to False.

```
>>> True or False
True
>>> False or True
True
>>> True or True
True
>>> False or False
False
```

The **not** operator only evaluates to the opposite Boolean value. True becomes False and False becomes True.

```
>>> not True
False
>>> not not True
True
>>> not not not True
False
>>> not not not not True
True
>>> not False
True
```

Mixing Comparison and Boolean Operators

>>> True and True

True

```
>>> (3<8) and (4<9)
True
>>> (3<8) and (9<4)
False
>>> (3==3) and (4==4)
True
>>> (3==3) and (4==9)
False
```

```
>>> (3==3) or (4==9)
True
>>> not (3==3)
False
>>> not (3==8)
True
>>> not not (3==3)
True
```

```
>>> 4+4==8 and not 3+3==7 and 3*2==4+5
False
```

>>> True and (not False) and False

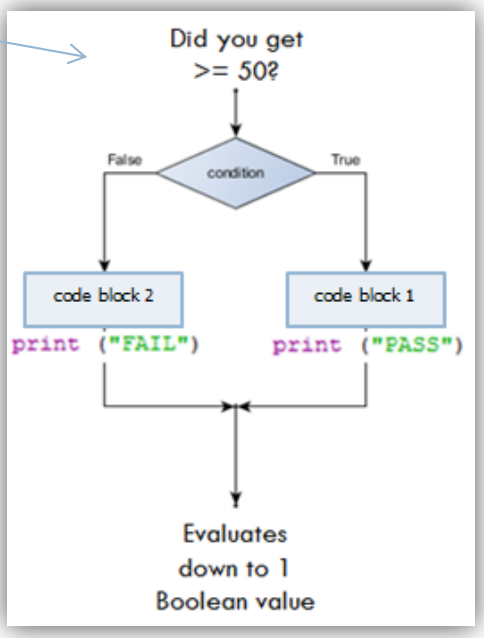
Section 3 - Flow Control

Flow control statements often start with a condition and are followed by code blocks.

3.1 - Conditions and Code Blocks

Conditions are expression that (always) evaluate down to a Boolean value (True or False). A *flow control statement* decides what to do based on whether the answer to its condition is True or False.

```
percentage = int (input ("What did you get? "))
if percentage >= 50:
    print ("PASS")
else:
    print ("FAIL")
```



Code Block are lines of Python code that are grouped and executed together (in blocks), one after the other:

- Code Blocks begin when the indentation increases.
- Blocks can contain other blocks.
- Blocks end when the indentation decreases.

```
def fib(n):
    print 'n = ', n
    if n > 1:
        return n * fib(n-1)
    else:
        print 'end'
        return 1
```

code block 1

code block 2

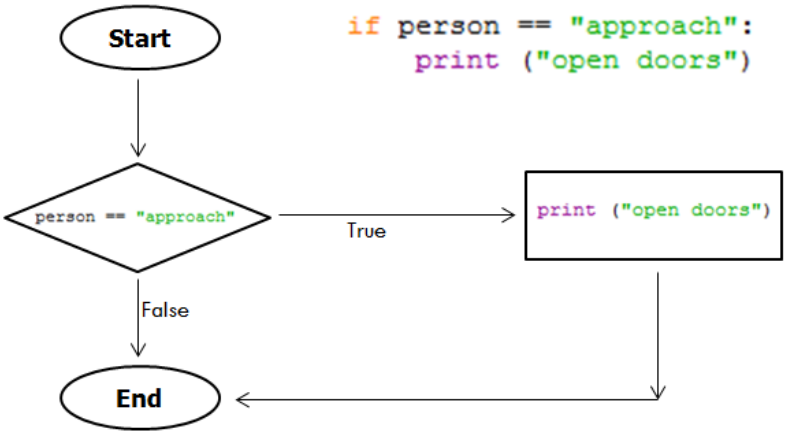
code block 3

3.2 - Selection: *if*, *else* and *elif* statements

Selection means choosing what to do next. The statements **if**, **elif**, and **else** are represented by a diamond  in flowcharts.

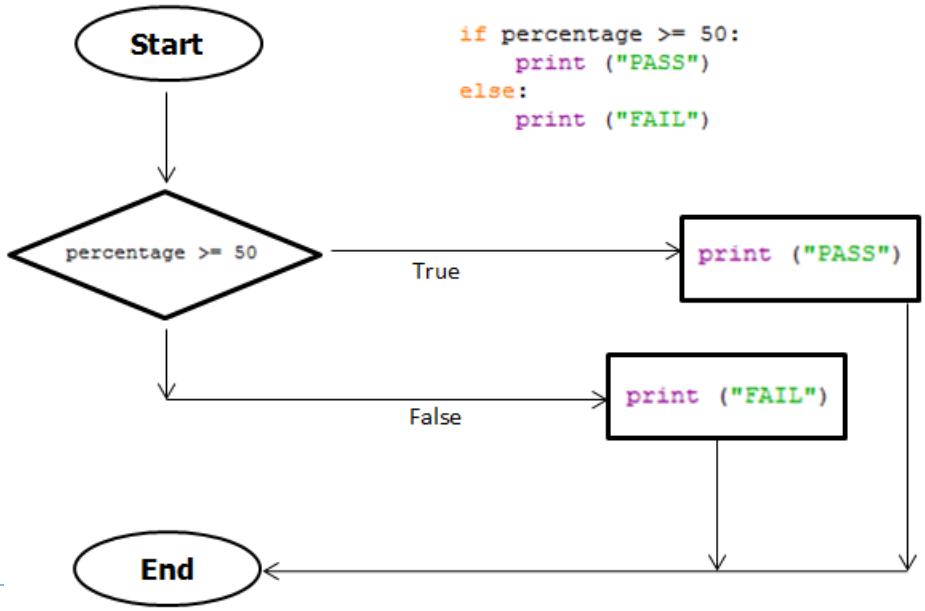
if Statements: basically mean "**if** this condition is true, execute this code".

Example: You go to a shopping centre and realise that doors open automatically when you approach the main entrance. This is due to a sensor that reads the following code:
"**if** someone approaches the main entrance, open the doors".



else Statements: basically mean "**if** this condition is true, execute this code; otherwise (**else**), execute that other code".

Example: A students has just finished a test on line and is ready to click on "submit". The computer will be able to give the student, straight away, a "PASS" or a "FAIL" after reading the following code: "**if** the student gets 50% or more, write PASS; otherwise (**else**), write FAIL".



elif Statements: basically mean "**if** this condition is true, execute this first code;
otherwise (**elif**), check if this second condition is true and execute this second code;
otherwise (**elif**), check if this third condition is true and execute this third code;
otherwise (**elif**), check if this fourth condition is true and execute this fourth code;
otherwise (**elif**), check if this fifth condition is true and execute this fifth code;
otherwise (**else**), if any of the previous conditions are true, execute this last code".

You can use as many **elif** as necessary

NB: There is only 1 **if** and 1 **else** clause, but you can have as many **elif** clauses as needed, always located between the **if** and the **else**.

Example: The teacher's computer has a program that reads the following code:

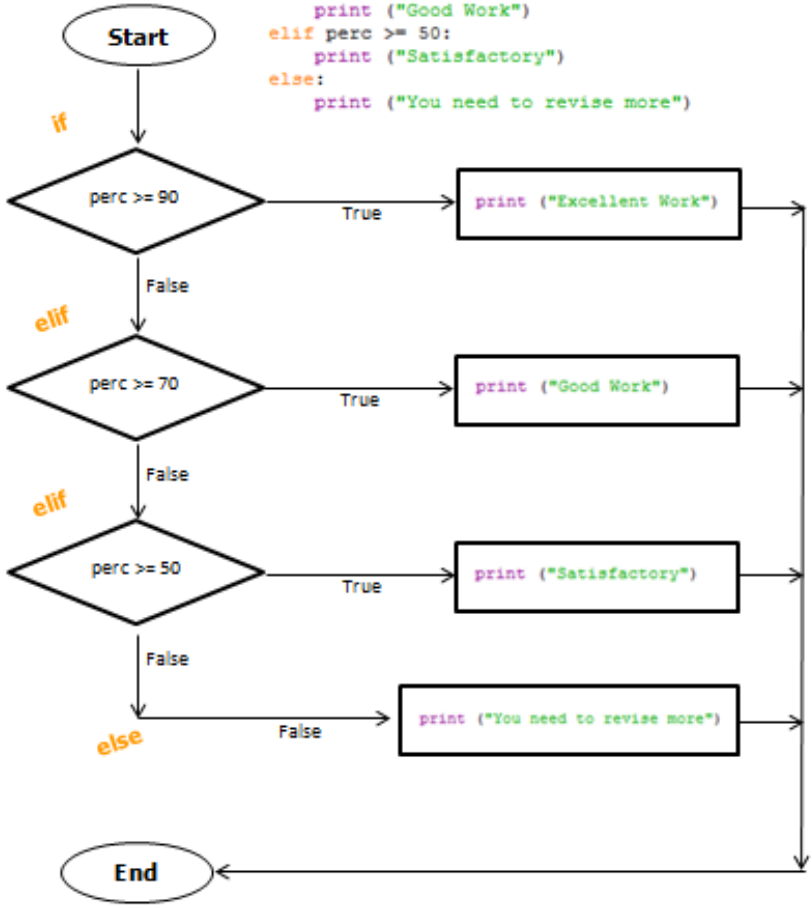
if a student gets 90% or more, write "Excellent Work";
otherwise (**elif**), check if the student gets 70% or more and write "Good Work";
otherwise (**elif**), check if the student gets 50% or more and write "Satisfactory";
otherwise (**else**), write "You need to revise more"

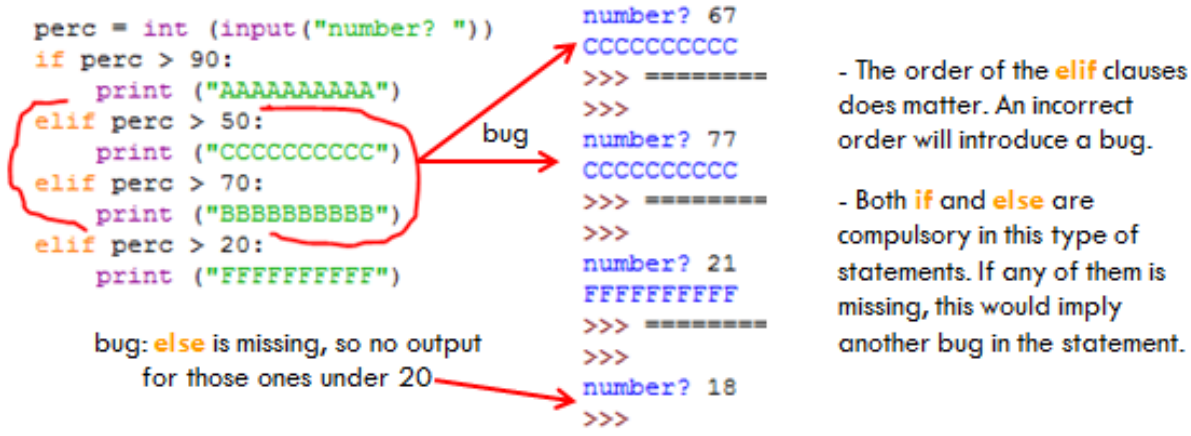
Script mode

```
perc = int(input("What percentage? "))
if perc >= 90:
    print ("Excellent Work")
elif perc >= 70:
    print ("Good Work")
elif perc >= 50:
    print ("Satisfactory")
else:
    print ("You need to revise more")
```

Interactive mode

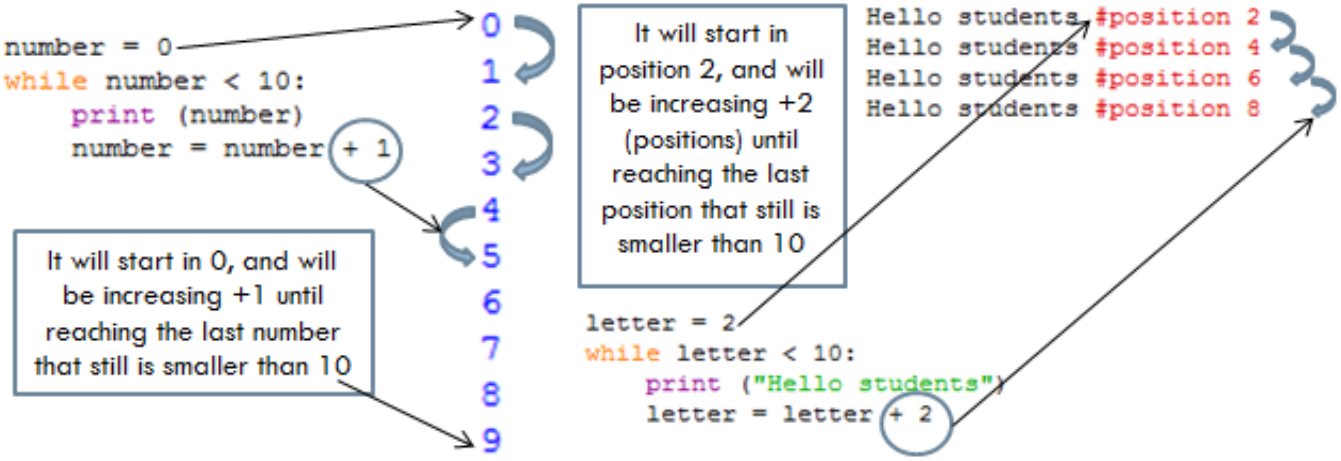
```
Python 3.4.2 (v3.4.2:ab2c023
tel)] on win32
Type "copyright", "credits"
>>> ========================
>>>
What percentage? 98
Excellent Work
>>> ========================
>>>
What percentage? 75
Good Work
>>> ========================
>>>
What percentage? 54
Satisfactory
>>> ========================
>>>
What percentage? 44
You need to revise more
```





3.3 - Iteration: *while loop* and *for loop* statements

while loops: are code blocks executed over and over again as long as their condition continues being True.

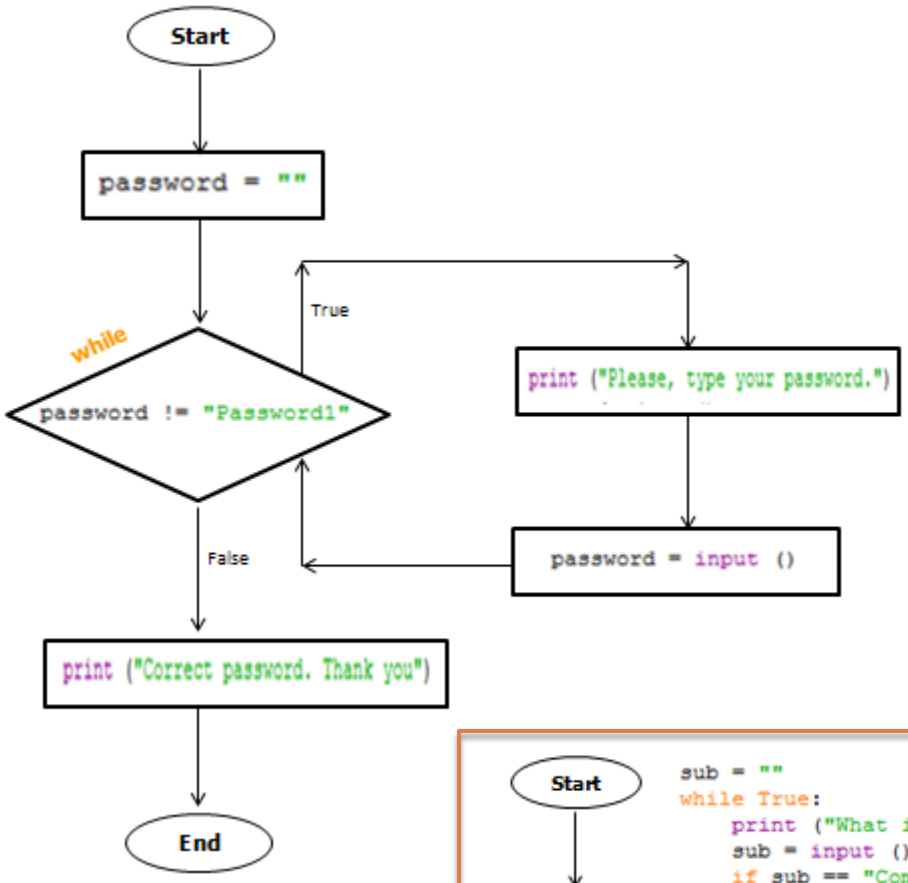


Try this case of a *while* loop:

```
password = ""
while password != "Password1":
    print ("Please, type your password.")
    password = input ()
print ("Correct password. Thank you")
```

```
Please, type your password.
bluesky
Please, type your password.
redsea
Please, type your password.
lightmoon
Please, type your password.
Password1
Correct password. Thank you
```

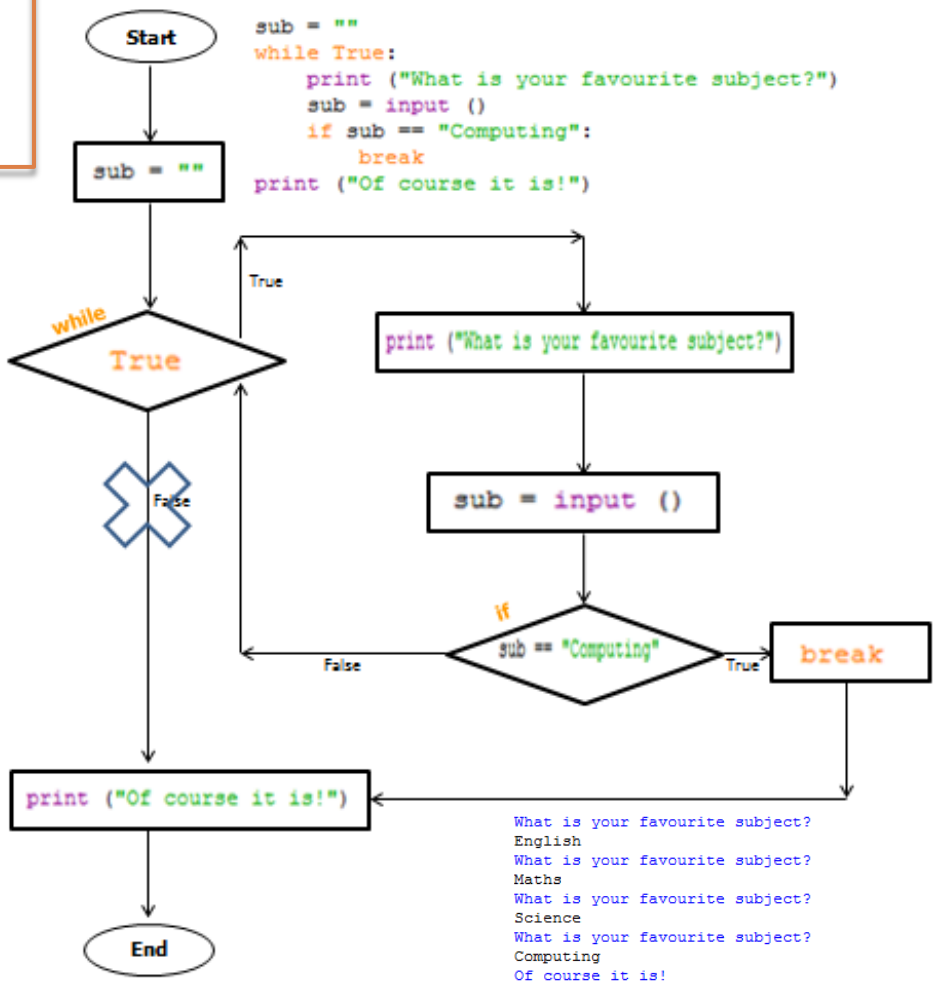
Please, see the flowchart on the next page.



The variable password is empty (i.e. has no value), so it is TRUE that the value of password is not equal to Password1. For that reason the **while** statement will be asking the user to input a new password over and over again until the user writes Password1, in which case the condition Password1 != Passrod1 will be False, and will continue running the code until the end.

Try this case of an infinite loop with a break statement:

```
sub = ""
while True:
    print ("What is your favourite subject?")
    sub = input ()
    if sub == "Computing":
        break
print ("Of course it is!")
```




```
What is your favourite subject?
English
What is your favourite subject?
Maths
What is your favourite subject?
Science
What is your favourite subject?
Computing
Of course it is!
```

infinite loop:

Sometimes, a program has a bug that does not allow the loop to end. We call it **infinite**. When this happens, the best way to stop the program is pressing Ctrl + C

break statements:

When the condition (**if** in this case) is True, the **break** statement is activated and will help the execution to move out of the infinite look.

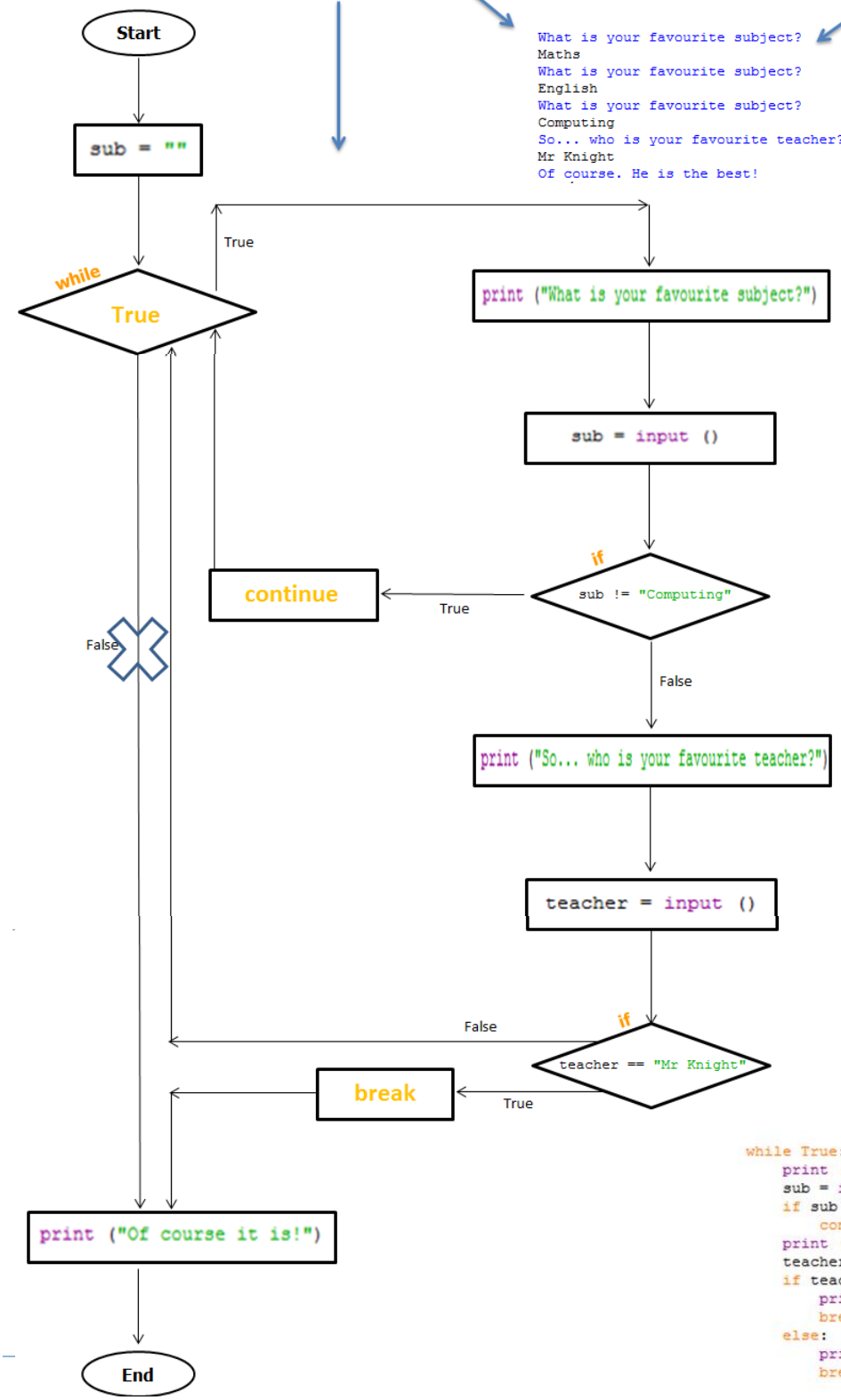
continue statements:

```
sub = ""
while True:
    print ("What is your favourite subject?")
    sub = input ()
    if sub != "Computing":
        continue
    print ("So... who is your favourite teacher?")
    teacher = input ()
    if teacher == "Mr Knight":
        break
print ("Of course. He is the best!")
```

=

```
sub = ""
while True:
    sub = input ("What is your favourite subject? ")
    if sub != "Computing":
        continue
    teacher = input ("So... who is your favourite teacher? ")
    if teacher == "Mr Knight":
        break
print ("Of course. He is the best!")
# if input needs a number, then a "int" is needed
```

```
What is your favourite subject?
Maths
What is your favourite subject?
English
What is your favourite subject?
Computing
So... who is your favourite teacher?
Mr Knight
Of course. He is the best!
```



Try another example:

```
while True:
    print ("What is your favourite subject? ")
    sub = input ()
    if sub != "Computing":
        continue
    print ("So... who is your favourite teacher? ")
    teacher = input ()
    if teacher == "Mr C":
        print ("Of course it is!")
        break
    else:
        print ("Wrong answer")
        break
```