

```

1 print("Welcome!\n")
2     "Let me help you to find out how many carrots we c
3     "Let's go!")
4 height=float(input("Please enter the height of your vega
5 width=float(input("Please enter the width of your vegeta
6 area=width*height
7 rowVal=width//8
8 colVal=height//8
9 totCarrots=rowVal*colVal
10 print("In a vegetable bed with an area equal to:", area
11 answer=input("Was that helpful? (yes/no) \n")
12 answer.lower()
13
14 ##### SORTING MEANS LISTING IN ORDER (FOR EXAMPLE FROM 1
15 numbers=[10,200,3,400,50]
16 print(number
17 numbers.reve
18 print("rever
19 numbers.appe
20 print(number
21 numbers.sort
22 ##### IN
23 numbers.inse
24 print("numbe
25 numbers.reve
26 print("numbe
27 numbers.rem
28 print("after
29 cnt=numbers.
30 print(cnt)
31 print(" Welc
32 carMake=inp
33 carMake.uppe
34 carModel=inf
35 carModel.up
36 mileage=flo
37 noDoors=int(input("4) Number of doors: "))
38 print("The vehicle you're here to register is a", carMa
39 confirm=input("Is that information right? (yes/no)")
40 confirm.lower()
41 if confirm=="yes":
42     print("Thanks for the cooperation, your car was regi
43     new = input("Would you like to register another vehi
44     new.lower()

```

```

1 ##### LISTING THE ITEMS IN THE LIST
2 vegs = ["broccoli", "carrot", "leek", "onion", "parsnip"]
3 for veg in vegs:
4     print(veg)
5
6 ##### PRINTING PART OF THE LIST
7 print(vegs[0])
8 print(vegs[1])
9 print(vegs[4])
10
11 ##### PRINT THE LIST IN REVERSE
12 vegs.reverse()
13 print("reversed list will be: ")
14 for veg in vegs
15     print(veg)

```

Mariya Tsvetanova Todorova
Student Number: 1910173
Assessment n° 1: Software
Development
Software Development
Lecturer: Renuga Jayakumar

```

16 print(studentName, studentNo, assignmentInfo, moduleInfo, lecturer, university, campus, date, sep="\n")
17 print("THIS IS MY SOFTWARE DEVELOPMENT ASSIGNMENT \n",
18       "this is the wonderful world of Python. \n",
19       "Enjoy the tour! 😊 ")
20
21 print("
22     o0o         o0o         \n"
23     "    o000o        o000o        \n"
24     "    o00000o       o00000o       \n"
25     "    o0000000o      o0000000o      \n"
26     "    o000000000o     o00000000o     \n"
27     "    o0000000000o    o000000000o    \n"
28     "    o00000000000o   o0000000000o   \n"
29     "    o000000000000o  o00000000000o  \n"
30     "    o0000000000000o o000000000000o \n"
31     "    o00000000000000o \n"
32     "    o000000000000000o \n"
33     "    o000000000000000o \n"
34     "    o000000000000000o \n"
35     "    o000000000000000o \n"
36     "    o000000000000000o \n"
37     "    o000000000000000o \n"
38     "    o000000000000000o \n"
39     "    o000000000000000o \n"
40     "    o000000000000000o \n"
41     "    o000000000000000o \n"
42     "    o000000000000000o \n"
43     "    o000000000000000o \n"
44     "    o000000000000000o \n"
45     "    o000000000000000o \n"
46     "    o000000000000000o \n"
47     "    o000000000000000o \n"
48     "    o000000000000000o \n"
49     "    o000000000000000o \n"
50     "    o000000000000000o \n"
51     "    o000000000000000o \n"
52     "    o000000000000000o \n"
53     "    o000000000000000o \n"
54     "    o000000000000000o \n"
55     "    o000000000000000o \n"
56     "    o000000000000000o \n"
57     "    o000000000000000o \n"
58     "    o000000000000000o \n"
59     "    o000000000000000o \n"
60     "    o000000000000000o \n"
61     "    o000000000000000o \n"
62     "    o000000000000000o \n"
63     "    o000000000000000o \n"
64     "    o000000000000000o \n"
65     "    o000000000000000o \n"
66     "    o000000000000000o \n"
67     "    o000000000000000o \n"
68     "    o000000000000000o \n"
69     "    o000000000000000o \n"
70     "    o000000000000000o \n"
71     "    o000000000000000o \n"
72     "    o000000000000000o \n"
73     "    o000000000000000o \n"
74     "    o000000000000000o \n"
75     "    o000000000000000o \n"
76     "    o000000000000000o \n"
77     "    o000000000000000o \n"
78     "    o000000000000000o \n"
79     "    o000000000000000o \n"
80     "    o000000000000000o \n"
81     "    o000000000000000o \n"
82     "    o000000000000000o \n"
83     "    o000000000000000o \n"
84     "    o000000000000000o \n"
85     "    o000000000000000o \n"
86     "    o000000000000000o \n"
87     "    o000000000000000o \n"
88     "    o000000000000000o \n"
89     "    o000000000000000o \n"
90     "    o000000000000000o \n"
91     "    o000000000000000o \n"
92     "    o000000000000000o \n"
93     "    o000000000000000o \n"
94     "    o000000000000000o \n"
95     "    o000000000000000o \n"
96     "    o000000000000000o \n"
97     "    o000000000000000o \n"
98     "    o000000000000000o \n"
99     "    o000000000000000o \n"
100    "    o000000000000000o \n"

```

University of Wales Trinity
Saint David
London Campus
30/04/2021

```

37 print(vegs)
38 while search==False:
39     print("would you like to search again? y/n")
40     search=input("enter the vegetable you want to check?: ")
41

```

Contents

➤ Introduction:	2
➤ Python:	3
✓ Variables:	3
✓ Variable naming rules:	4
✓ IF condition in Python:	7
✓ Different ways of using the if condition in Python:	7
If statement:	7
If - else statement:	8
If- elif - else statement:	8
Nested if statement:	9
✓ Looping in Python:	12
✓ For loop in Python:	12
✓ While loop in Python:	15

➤ Introduction:

Nowadays programming languages have become popular around the world because of industry needs and easy access for the passionate ones. To have an advantage in understanding and moving easier from one language another, it may be enough knowing the basic rules for one of them and apply it to the others. That doesn't mean that the syntax will be the same for all of them, but you will be able to face many similarities about it and the meaning of the main components (called keywords) will remain untouched.

A few examples of programming languages available out there for different industry fields and purposes are as follows:

1. **C:** a high-level programming language which has different variants for different industry fields like **C#** and **C++**. It is commonly used to develop system applications; that means that usually OS like Windows and Linux run with C programs. C is also used for games, calculation and graphic programs development.

If we choose to compare it with Python we can find a few points that goes in Python's favour:

- 1) "Easier to learn
 - 2) Smaller code
 - 3) Supported as open source
 - 4) Allows use of multiplatform development environment
 - 5) Easy to extend using Java and C/C++ " [1]
2. **Java:** a general-purpose language. It is a **class** and **object-oriented language**, which means that it works mainly with arrays and objects. These, have the advantage of being able to read and modify information stored in their own objects and then proceed with the operations requested by the user. Usually, to be able to read java codes you must have java scripts installed otherwise you will not be able to see properly websites or run applications as they were developed to run.

Compared with Java, Python has some stronger points here as well:

- 1) "Easier to learn
 - 2) Smaller code
 - 3) Enhanced variables that can hold different kinds of data while running (dynamic typing), based on the application's needs
 - 4) Faster development (in terms of time)" [1]
3. **R:** a unique low-level language commonly used in academic and big company fields for statistical and data mining operations and graphic techniques software. It includes some software extensions which allows you to calculate, display and manipulate easily data from a statistical point of view.
 4. **Python:** recommended and commonly advised by companies and universities, is a high-level programming language for beginners and a good object-oriented based one. It follows the standards of **C** but with an easier set of rules. It's easy to read and write and it is versatile and easy to use for general purpose; also particularly used for AI and machine learning courses. Its peculiarity is the fact that it's an open-source program developed in **C**.

For the purpose of this assignment we will examine and take under consideration just the characteristics of this last one, focusing on some main functions in order to learn and understand some of the main components needed to be able to code with Python.

➤ Python:

✓ Variables:

As you might have understood, we will base our knowledge and understanding on Python for the reasons specified above.

When you're using a high-level language, you will need one of the two following translators. These will allow your computer to convert a high-level language (which is readable by humans) to a low-level language also called machine language (which is the language the computers write and understands) and to read/execute it. [2]

- 🔧 Compiler: converts high-level language into machine language (binary/ 0s and 1s) translating the whole code in one go. It's the more frequently used by programmers and it's considered to be time saving because of the way it analyses and converts the code. By translating the whole code in one go, it can display all the errors (if present) in the code at one time.
- 🔧 Interpreter: translates the code line by line. If an error occurs, it stops the program's execution. Once the error is fixed the interpreter starts again its job from the beginning of the code. If no error occurs, the interpreter reads the code as it was projected to be executed.

The main interaction between the user and Python occurs by a sequence of **inputs** and **outputs**.

We have an **input** when we request the user to insert some data or to execute an action. Thus, it means that an input occurs every time a message that request the user's interaction is displayed on the screen (for example: each time the computer requests the user for a password to access the system in order to be able to use the computer). In other words, it's called **input** whatever requests the user to make an actual action, to allow the computer to return an output.

We are talking about **outputs**, instead, when the information must be displayed by the computer to the user. A good example might be the ERROR message which appears each time a program fails to execute or when we browse the web and ask our web browser to display a web page, the webpage in this case will be our output.

The easiest way of communication between Python and the User is by the **print("")** statement. This basically says to the program "display what is in between the two brackets and the quotation marks to the user".

One thing to bear in mind when you think about writing programs in Python is that:

Python is case sensitive, that means that a word written in upper-case letters will have a different meaning compared to the same word written in lower-case letters.

When you start working with Python one of the first things you learn to use is variables.

A variable is a programming element commonly used to "store data" of different types. This data can be either:

- Integers: represented by  **int("")**
- Decimal numbers or Floating Point Numbers/Floats: represented by  **float("")**
- Letters: represented by  **str("")**

That means **variables** can store three different types of data, and can be considered as “etiquettes” which helps us to store and remember, once needed, important information during the coding process.

In most of the commonly used high-level programming languages, like **C**, the declaration (specification) of the type of variable which will be used is **mandatory**; this phenomenon is referred to as **strong-typed**. Once that’s done, the language will be able to reserve the dedicated space needed for the storage of the determinate variable.

In Python, this **is not needed**. Python deduces the variables’ type on its own which classifies it as a **weakly-typed programming language**.

To facilitate the recognisance process of the variables, we must assign (**declare**) a name to each of them by following this simple “scheme” containing the rules for their name assignments’:

Name of the variable + **equal sign** = + **the specification of the variable type we are going to use** + **opening round bracket** (+ **opening quotation marks** “ + **the value you want to assign to the variable** + **closing quotation marks** “ + **closing round bracket**)

Example: `myVar = str ( “ This variable contains a string. ” )`

Bear in mind: now that you’re aware of the structure to follow while creating a variable, you must learn the specific rules required for its name assignment.

✓ Variable naming rules:

As you might have thought, there is rules to follow about variables naming process as well.

A variable name:

1. **Cannot** be a **reserved word***;
2. Is **case-sensitive**;
3. **Must** start with a letter or an underscore (`_`);
4. **Cannot** start by a number;
5. Can **only contain alphanumeric characters** and **underscore**;
6. Should not start by uppercase letters;
7. Must be **mnemonic/self-explicative** (the name tells what the variable’s about).

*a reserved word is a keyword which has a specific function in Python’s language. If you ever try to use a reserved word as a name of a variable, your program will show you an error message and will stop its execution.

There is another non-written rule, which is more as a good behaviour habit for programmers, it’s called **Camel Casing**.

This is a programmers’ convention that makes the code easier to read and establishes that:

- If the variables’ name is long or formed by many words, you should/can alternate upper and lower case characters to help/improve readability;
- The name of the variable should be self-explicative.

Once all this is set, we can start using the variable with Python's `print()` statement.

The scheme to follow in order to invoke the variable will be the one below:

print + opening round bracket (+ Name of the variable + closing round bracket)

So, if we consider the variable created before and we type:

`Print(myVar)`

The outcome will be:



`>>> This variable contains a string.`

Which is the value we have previously assigned to `myVar`.

Following you will be able to see some code which shows the use of variables and their names:

```
# write a program with FOUR VARIABLES which must contain different types of
values
#### 1st variable should contain: car make
#### 2nd variable should contain: car model
#### 3rd variable should contain: mileage
#### 4th variable should contain: number of doors
##### THE PROGRAM SHOULD ALLOW THE USER TO ENTER THE DATA AND THEN
DISPLAY THEM ON THE SCREEN
#The variables should be appropriately typed for the data required.
print(" Welcome to the DVLA application, here you can register your vehicle
by entering these information: \n")
#The "input" command in the following line allow the user to insert
information which will be stored by the program to be displayed later on,
in case of need
carMake=input("1) Car Make: ") #THIS IS A STR variable
#If we want to be picky about the grammar to use in the above and the next
row, the code should look like: carMake=str(input("1) Car Make: "))
carMake.upper() #This line of code tells to Python that if the user uses
lower-case letters, the program should convert them to upper-case letters
in order to identify and accept them
carModel=input("2) Car Model: ") # THIS IS A STR variable
#and: carModel=str(input("2) Car Model:")) BUT, there is not really need to
specify the "str" when we write a string,
# INSTEAD, if it consist of int and float numbers we MUST specify which
kind of variable type we are using/need to obtain
carModel.upper()
#The following line of code tells Python that the input the user should
insert, must consist of floating point numbers or decimal numbers
mileage=float(input("3) Mileage: ")) #THIS IS A FLOAT variable
#The following line of code tells Python that the input from the user
should be an integer number
noDoors=int(input("4) Number of doors: ")) #THIS IS AN INT variable
print("The vehicle you're here to register is a", carMake, carModel,"with
", mileage,"Km and", noDoors,"doors.")
confirm=input("Is that information right? (yes/no) \n")
confirm.lower()
if confirm=="yes" and confirm!="no": #This line tells
Python that if the last input from the user is: "yes" then python should
print and then ask the following two lines of code
    print("Thanks for the cooperation, your car was registered
successfully.")
```

```
new = input("Would you like to register another vehicle? (yes/no) \n")
new.lower()
while new=="yes" and new!="no":                                #This line tells
Python that while the answer to the variable named as "new" is: "yes",
Python must display the following lines
    print("Please enter the following...\n")
    carMake = input("1) Car Make: ")
    carMake.upper()
    carModel = input("2) Car Model: ")
    carModel.upper()
    mileage = float(input("3) Mileage: "))
    noDoors = int(input("4) Number of doors: "))
    print("The vehicle you're here to register is a", carMake,
carModel, "with ", mileage, "Km and", noDoors,
        "doors.")
    confirm = input("Is that information right? (yes/no) \n")
    confirm.lower()
    if confirm == "yes":                                         #IF the input to the "confirm"
variable is: "yes" Python must display the following 2 lines of code
        print("Thanks for the cooperation, your car was registered
successfully.")
        new = input("Would you like to register another vehicle?
(yes/no) \n")
        new.lower()
    else:                                                         #IF the input to the "confirm"
variable is: "no"
        break                                                    # Python must stop the
execution of the program and print the last line of code (line 48)
else: #This line of code is related to the input regardin the "new"
variable settled on line n° 26 and it states that if the input is:"no"
Python should display the following 2 lines
    print("Sorry to know that...")
    again=input("Would you like to try to enter again the information about
your vehicle? (yes/no) \n")
    again.lower()
    if again=="yes" and again!="no":
        print("Please enter the following...\n")
        carMake = input("1) Car Make: ")
        carMake.upper()
        carModel = input("2) Car Model: ")
        carModel.upper()
        mileage = float(input("3) Mileage: "))
        noDoors = int(input("4) Number of doors: "))
        print("The vehicle you're here to register is a", carMake,
carModel, "with ", mileage, "Km and", noDoors,
            "doors.")
        confirm = input("Is that information right? (yes/no) \n")
        confirm.lower()
    else:
        breakpoint()      #This line will stop the execution of your
program and will print the last line of code on the display
print("Thanks for the cooperation. Have a lovely day 😊 .")
print("END OF THE PROGRAM")
```

✓ IF condition in Python:

When we talk about the “if” condition/statement in Python language, we use it to check if one or more conditions are true or false, and depending on their result, Python will act in one way rather than another following the instructions settled during the if statement.

For example:

If you want to type a code which checks a number you allow the user to input, and based on its value, in one case you set up Python to tell you “That’s a nice number!” rather than “Mmmh, not sure this is the number we’re looking for”, your code will look something similar to the following one

```
numb=input("Please choose a number from 1 to 10 \n")  
  
if numb=="5":  
    print("That's a nice number!")  
  
else:  
    print("Mmmh, not sure this is the number we're looking for")
```

In the code we can see above, **numb** is the variable which allow us to input a number. The line which follows, tells the program that **IF** (in case the condition that follows the **if** statement matches/happens) the program should print on the screen “That’s a nice number!” otherwise: the program should show the user the following message “Mmmh, not sure this is the number we’re looking for”.

There is a few ways of using the **if** statement. The one we’ve seen above is one of them. A basilar rule to make your code work and don’t crush is to **always remember the line after an if statement should always be indented** otherwise your code will display an error message and will not execute.

if condition may be associated with comparison operators (**!=** [not equal to]; **==** [equal to]; **>=** [greater or equal to]; **<=** [smaller or equal to]; **>** [greater of]; **<** [smaller of]) as also with logical operators (**and**; **or**; **not**) and membership operators (**in**; **not in**).

✓ Different ways of using the if condition in Python:

When we are at a point in our program, that asks us to make a decision and act accordingly, we now know that we should refer to the **if** condition for help.

It’s better now to understand how to use it. There are, in fact, a few different ways [3] to do so.

If statement:

The simple way of using an **if condition** it’s with its simple statement. That means that’s enough to set a simple condition as per the example that follows.

```
year=5  
  
if year<5:  
    print ("I knew you were a noob here! 😊")
```

As you can see in the code above the condition is really simple. When executed, it will be checked, and if it will have the value indicated, the program will display the statement enclosed after the **print**

statement. If the check doesn't meet the requirements, nothing will happen, the program will keep going with the lines of code that follows after.

If - else statement:

This statement is commonly used if you want to give two options to be followed. Usually, the first option (the one that follows the **if** statement), will be executed if the code is corresponding to a **True** statement. The **else** condition instead, will be met only if the code is corresponding to a **False** statement.

Example:

```
socks=input("What colour are your socks today? \n")

if socks=="blue":

    print("That's amazing! Mine as well! 😊")

else:

    print("Cool! I've just bought a bowtie of the same colour yesterday!")
```

In this example, we ask the user to input the colour of their socks and we state that **if** the input is **blue** the output on the screen should be "That's amazing! Mine as well! 😊" (this condition can be summarized as follows:  if the condition is true, display the following statement). Otherwise, **if** the input is not **blue** the output displayed on the screen will be "Cool! I've just bought a bowtie of the same colour yesterday!" (this condition can be summarized as follows: if the  condition is false, display the following statement).

The **if-else** condition is usually used to check only two conditions. If we want to check some more we may need the following statements.

If- elif - else statement:

The **if-elif-else** condition allow us to test more than two options in order to achieve the desired output with our code. The advantage of using this, instead of the previous if statement, is that here you can have as many conditions to be checked as you prefer. The important thing to bear in mind is just that to make your code work, you must follow the structure below:

```
If
elif
else
```

in this order. If you choose to put the **else** statement before the **elif**, you will probably have a different output than the one desired.

Remember: the **else** statement tells where the condition checkings should end, so if you put it before the **elif** conditions, they will not going to be read and evaluated.

The **if-elif-else** statement is useful because you can check multiple conditions at the same time, you just have to remember to assign each condition to a different **elif** statement.

An example of a multiple **if-elif-else** condition can be as it follows.

```
socks=input("What colour are your socks today? \n")
if socks=="blue":
    print("That's amazing! Mine as well! 😊")
elif socks=="white":
    print("Cool! White as the snow that falls in winter.")
elif socks=="pink":
    print("My daughter loves pink! She has a box plenty of pink bracelets")
else:
    print("Cool! I've just bought a bowtie of the same colour yesterday!")
```

Nested if statement:

There is another way of setting an if condition as well. That is called the nested way. This, allow us to be much peakier on the actions of our program, and it works more or less as it follows.

You set the main if condition:

```
socks=input("What colour are your socks today? \n")
if socks=="blue":
    print("That's amazing! Mine as well! 😊")
else:
    print("Cool! I've just bought a bowtie of the same colour yesterday!")
```

and in between the two statements, you insert another two (or more) statements to be checked.



```
socks=input("What colour are your socks today? \n")
if socks=="blue":
    print("That's amazing! Mine as well! 😊")
    if socks=="white":
        print("Cool! White as the snow that falls in winter.")
    else socks=="pink":
        print("My daughter loves pink! She has a box plenty of pink bracelets")
else:
    print("Cool! I've just bought a bowtie of the same colour yesterday!")
```

That means that once the first if condition has been checked and its outcome was recognized as true, the second part of evaluation will have place, so the “secondary” **if condition** will be checked and evaluated as well. If in this case, the output of the second, is a **False**, the program will then jump back executing the main **else** condition.

You can use the nested statement with the **if-elif-else** as well, the important is that you bear in mind the indention rule for each block of if condition.

Following you will be able to see how do **if-else** condition works in an actual program.

The following program allows a user, which wants to start growing his own carrots, to input the height and width of its vegetables bed, in order to know how many carrots he will be able to grow in there.

```
### write a program which allows the user to input the height and
width of their carrot bed and then calculates how many ENTIRE
carrots can be grown in that carrot bed
#### REMEMBER #### each carrot, to be well grown, needs 8x8 cm
squared soild bed, a total of 64 cm quared x single carrot
print("Welcome!\n"
      "Let me help you to find out how many carrots we can grow!
☺ \n"
      "Let's go!")
height=float(input("Please enter the height of your vegetable
bed: \n"))      #This line requests the INPUT of the height of
the veg. bed from the user
width=float(input("Please enter the width of your vegetable bed:
\n"))          #This line requests the INPUT of the width of the
veg. bed from the user
area=width*height
#This line calculates the area of the veg. bed
rowVal=width//8                                #Based on the
width of the veg. bed this line calculates the amount of carrots
which can be grown in a row of this veg. bed
colVal=height//8                                #Based on the
height of the veg. bed this line calculates the amount of carrots
which can be grown in a column of this veg
totCarrots=rowVal*colVal                        #This line
calculates the estimated carrots the user can grow in his veg.
bed based on the dimension inputted
print("In a vegetable bed with an area equal to:", area,"you may
grow a total of: ", totCarrots)
quest=input("Was that helpful? y/n \n")
#This line allow the user to try the measurement again by
inputting yes or no
quest.lower()
#This line tells python to convert the input assigned to question
var, ALL in lower-case letters
answ="yes" and "y"
if quest!="yes":                                #This line states
that if (and only if) this condition appears, the following lines
of instruction myst be executed by the program
```

```
print("Ok, let's try again then!")
height=float(input("Please enter the height of your vegetable
bed: \n"))
width=float(input("Please enter the width of your vegetable
bed: \n"))
area=width*height
rowVal=width//8
colVal=height//8
totCarrots=rowVal*colVal
print("In a vegetable bed with an area equal to: ",area, "you
can grow a total of: ", totCarrots,"\n ")
quest.lower()
quest=input("Would you like to calculate again? y/n \n")
while quest=="yes" and answ!="no":
#The above line tells to python that, until the answer to the
question is : yes python MUST execute the following lines. If the
answer is no: python will finish the program
    print("Ok, let's try again then!")
    height = float(input("Enter the height of your vegetable
bed: \n"))
    width = float(input("Enter the width of your vegetable
bed: \n"))
    area = width * height
    rowVal = width // 8
    colVal = height // 8
    totCarrots = rowVal * colVal
    print("In a vegetable bed with an area equal to: ", area,
"you can grow a total of: ", totCarrots,"\n ")
    quest.lower()
    quest = input("Would you like to calculate again? y/n
\n")
else:
    print("Have an amazing harvest! 😊\n",
          "END OF THE PROGRAM\n",
          "Have a nice day! 😊\n",
          "Thanks for choosing this program!")
    exit()
#The following line of code will indicate to python what to print
in case the user input is: "yes" for the first question or no
during the if condition
#OR if the answer to the question is:"no" during the while loop
print("Have an amazing harvest! 😊\n",
      "END OF THE PROGRAM\n",
      "Have a nice day! 😊\n",
      "Thanks for choosing this program!")
```

✓ Looping in Python:

Looping process occurs when there is a section, or multiple sections, of our code which must be repeated. Usually **loops** are used with **iterations** (specification of how many times the looping will be run before it stops) and substitute lines and lines of **if condition** code. If you have a code that requires you to repeat many times the same if condition along the whole program, you may prefer making your life easier and make your code more readable by using a loop instead.

There is two type of looping in Python language:

- 📁 **for**
- 📁 **while**

✓ For loop in Python:

The **for** loop is commonly used in association with lists (no matter if they contain numbers or strings). We can say that the **for loop** is a **defined loop** [2] because usually you are aware of where it starts and where it ends (based on the list specification). "The loop executes a fixed number of times, and you know the number of times it will execute before the loop even begins" [1]. It means that a **for loop** will last until it will be gone through all the iteration of the list (one by one) it is associated with.

An example of a simple **for loop** can be as it follows:

```
bands = ["Linkin Park", "Kiss", "Rolling Stones", "Nirvana", "Korn", "Slipknot"]
```

```
for band in bands:
```

```
    print("Our guest band is", band)
```

```
print("Enjoy some good old rock! ")
```

The output will be:



Our guest band is Linkin Park

Our guest band is Kiss

Our guest band Rolling Stones

Our guest band is Nirvana

Our guest band is Korn

Our guest band is Slipknot

Enjoy some good old rock!

So based on this example we can see that Python has recognized the **bands** list. When we have set the **for loop** for the **bands** list, Python has started reading and displaying them line after line as he was reading one name-one line.

You can choose to use the **for** loop for counting as well. It can count increasing, decreasing, increasing or decreasing by definite numbers (example: count from 1 to 10, adding 2 numbers at each count).

An example of this last case is:

a counting loop that goes from 0 to 12

```
for x in range(0,13):
```

```
    print("Counting...", x)
```

As an output you will see 12 lines of "Counting..." followed by the numbers increasing from 0 to 12. Once it will reach 12, it will stop the count.

a counting loop that goes from 0 to 16 adding 2 each time it counts (example: 0, 2, 4 ,6 and so on)

the last number in between the parentheses is called step value and defines by how much the count should be increased

```
for x in range(0,17,2):
```

```
    print("Counting...", x)
```

As an output you will see 9 lines of "Counting..." followed by the numbers increasing from 0 to 12 which will be increased itself of two each time.

As you may have noticed in the previous two examples, when I was specifying the last value for the loops (12 and 16), in the code they were increased by one each. That happened because the count in the looping starts from 0, and not from 1 as we are used to count, which means that every time it will have to display the last value, it will appear with one value less. So, if my last value in a for loop is 12, I will have to expect an 11 on the screen as an output.

The **for** loop, by default, tend to count increasing by 1 position, unless specified otherwise.

A good example to understand the **for loop** may be what follows.

The program you will see below, allows the user to enter 5 products and respective prices in 2 different lists. It then calculates the partial price to pay and the total price to be paid once a staff discount of 15% is applied.

Let's have a look at it together! You will find all the comments line by line explaining what each line does.

```
#### write a program that allows the user to enter the name of 5
products into an array or list and their respective prices into a
second array or list
#### when all the products and prices have been entered the
program should print them on screen
#### then it should calculate the total price of all the products
#### as last thing it should calculate the total price with a 15%
of staff discount
products=[]          #### I'm creating an empty list where to
store my items
itemnumb=[x for x in range(1,6)]    ####creates a list that allows
5 numbers in it and will count from n°1 till n°5 automatically,
adding 1 each time
prices=[]            #### empty list where prices will be
stored
for itemnumb in itemnumb:    ### asking to print each number of
itemnumb for its whole value (in this case will be 5 times in
total)
    print("Enter product n°",itemnumb)  ## this line will print "
Enter the product n°" and will be followed by a single number of
the itemnumb list (which will increase of one each time it will
be repeated)
    products.append(input(""))          #### products.append() tells
to python to save and store in the list each product that will be
inserted by the user via each input
    print("Enter price n°", itemnumb)
    prices.append(float(input("")))      ### this line works
exactly the same as the one of the products.append
print("Products entered:\n")
for (a,b) in zip(products, prices):    ### this line puts on the
same line the products in the products list, and the prices in
the prices list
    print(a,b)                        #### here the program is
asked to print one product followed by one price of their
respective lists side by side
subtot=sum(prices)                    ### sums together all the prices
of the prices list
print("Total price is: £",subtot)
discount=(subtot/100)*15              ##### calculates the discount to
be applied
total= subtot-discount                 #### calculates the final price
to be applied to the costumer, once the staff discount is applied
print("Total to pay with staff discount applied is: £", total)
```

✓ While loop in Python:

We have understood that the **for** loop is a definite looping; the **while loop** instead, is an **indefinite looping**. **While loops** are commonly used to process data of unknown sizes. Because of this, it means they can go on for indefinite (infinite) time. To avoid that, sometimes, **break** and **continue** instructions come to help us. To understand this loop you can actually just get it literally as it is. In contrast with his brother, the **while** loop, **does not increment by default**, which means that if you don't specify the increment in the coding line, an infinite loop will occur and the only option you will have to stop it will be closing your compiler or switching off your computer.

An example of a **while loop** may be as it follows:

program that allows the user to input the number from which to start the countdown, once the counter number is set from the user, the countdown begins

```
print("Let's count down to 1.\n")
counter = int(input("Where shall we start? \n"))
print("Let's go!")
while counter >= 1:
    print(counter)
    counter = counter - 1
```

As per the **if condition**, the **while** and **for** loop may be nested as well.

References

- [1] J. P. Mueller, "Beginning Programming with Python For Dummies, 2nd Edition," in *Beginning Programming with Python for Dummies*, Hoboken, New Jersey, USA, John Wiley & Sons Inc., 2018, p. 408/411.
- [2] C. R. Severance, Python for everybody, Exploring data in Python 3, vol. 2, Elliott Hauser, Sue Blumenberg, 2016.
- [3] L. Tagliaferri, "DigitalOcean," 15 06 2017. [Online]. Available: <https://www.digitalocean.com/community/tutorials/how-to-write-conditional-statements-in-python-3-2>. [Accessed 01 05 2021].
- [4] J. P. Mueller, Programming with Python for dummies, hoboken: john Wiley & sons, inc., 2014.

