Python

Initiation Workshop

Presented by : Sofiane MEDJKOUNE
Email : sofiane.medjkoune@protonmail.com

What's a programming language ?

- *A **programming language** is a special language programmers use to develop software programs for computers.*
- *A Software program is a set of instructions that the computer can read and understand (After few transformations of course).*
- *There are many programming languages, some of them are compiled and interpreted languages (further informations at : https://www.lifewire.com/compiled-language-2184210), some of them are :*
    - ***Compiled :** C, C++, Java, Pascal...etc.*
    - ***Interpreted :** Perl, PHP, **Python**...etc.*
- *Choosing the language to use depends on your need, for example to develop a web app you can use PHP, to create an android application you can choose java, to manage critical system applications you can choose C...etc*
- *In this course we'll be choosing **PYTHON.** It's one of the most used languages in 2019 because of its portability, ease to learn and use, rich documentation and great community. (More informations at : http://www.bestprogramminglanguagefor.me/why-learn-python)*

Variables

- A variable is simply a storage space in our memory that contains an **information.** Informations must be stored somewhere (that somewhere is our variable) so that we can use this information in our code, read it, update it, use it elsewhere and in certain cases remove it .
- Variables are represented by their **identifier** (symbolic name) this name must follow the following conventions : https://docs.snowflake.net/manuals/sql-reference/identifiers-syntax.html .
- In addition to the identifier, a variable has a **type** and a **value**. The most common types are : (more informations at : https://www.tutorialspoint.com/python/python_variable_types.htm)
  - ***int*** or integer: a whole number, ex : ***1, -5, 0, 132482, -3213 ...etc***
  - **float** or floating point value: ie a number with a fractional part. ***ex : 1.0, 3232.4, -3.5 ...etc***
  - **char-strings** : a single character - multiple characters put together ***ex : 'A', 'D' - "Welcome to Open Minds Club !", do not forget the " " for the characters and strings !***
  - ***booleans*** : whether a statement is true or false ***ex : True, False .***
- In python there's no need to declare the variable type, we just assign the value to the variable.

Variable operations

- ***Affectation*** : my_fisrt_variable = 1
- ***Multiple affectation*** : my_first_variable, any_other_variable = 2.5, “Yes affecting a string now”
- ***Arithmetic operations*** : a = b + c, a = b - c (basic arithmetic operators are : +, -, *, /, %)
- ***incrementation*** : a = a + 1 can be written as : a+=1 , The general form is : variable = variable (arithmetic operator) (value or variable) ===> variable(arithmetic operator)=(value or variable) for example : a = a - b ===> a-=b
- ***Boolean affectation*** : my_crazy_variable_name = True
- **Boolean Expressions** : Boolean operations are based on comparators (>, <, >=, <=, ==, is, !=, not, and, &&, or, ||) for example we can have :

a, b = 2, 10

my_crazy_variable_name = (a > b) ***(this operation will give the Boolean value : False to my variable)***

my_crazy_variable_name = (a <= b) (this operation will give the Boolean value : True to my variable)

Conditions

- In programming, conditions are one of the most important operations, it tells the program whether to choose between option 1, option 2, option 3… depending on a condition, *for example : if i follow this course the good way ===> i'll get a certificate, if i don't ===> i'm not getting the certificate* .
- In python conditions follow the syntaxe :

```
if BOOLEAN EXPRESSION:
        STATEMENTS
elif BOOLEAN EXPRESSION:
        STATEMENTS
else:
        STATEMENTS
```

- ***Important rules :***
  - DON'T FORGET THE INDENTATION, in python if a statement belongs to a condition, loop, function, exception or any other structure we need to put a tab under the structure.
  - Don't forget the : (two points) after Boolean expression or the else statement.

Loops

- Loops prevent us from re-writing the same instruction multiple times.
- Loops help us parse through structures in a simple and fast way.
- The most used ones are the **For** and **While** loops.
    - **While :** is mostly used when we do not know how many times we have to repeat an operation and is written :

while BOOLEAN EXPRESSION :
SET OF INSTRUCTIONS

Example :

```
my_age = 17
while my_age < 18 :
    can't go to a dance club
    if birthday :
        my_age += 1
```

Important : don't forget the indentation as mentioned previously and don't forget the 2 points ":" after the Boolean expression.

Loops

- The **for** loops can be used literally everywhere and is the most used one. In general we use the **for** loop when we know how many times we have to repeat a certain instruction, for example if we want to print the multiplication table for 6, **i know** i have to print a certain results 10 times : 6x1 = 6, 6x2 = 12...etc
- In addition the **for** loop is used to parse over structures, for example if we need to read a file that contains hundreds of words we can use the **for** loop to get these words (We'll see that in an upcoming course).

Examples :

```
for i in range (1,11) :
        print (" 6 x",i," = ", 6*i)

for character in "print each character alone" :
        print (character)
```

New instructions

- ***print()*** : used to print a text, or the content of a variable to the console

  print(“Hello this is my first print and i wanna print the content of the variable”,my_agee)

- ***input()*** : gives the hand to the user to input a value, generally used this way :

  my_variable = input()

- ***in*** : is a key word generally used in BOOLEAN EXPRESSION to say that the values of our variable will be **in** a certain range of values (as we have seen in the previous slide)

Practice time!

Write a program that asks the user a mathematical operation, and he has to answer ,if the answer is correct, we congratulate him, if the answer is not correct, he has to write the sentence : I suck at math, i need more practice. he has to write it a 1000 times !!!!

Functions

- Functions group a certain set of instructions so that we can use it anywhere in our code without having to write that set of instructions again. For example if you want to get the results of a second degree equation you won't need to write to calculate the delta and the 2 solutions each time, we'll be writing a function that returns the solutions and we'll use it whenever needed.
- We have seen a certain number of functions already : print(), input(), range().
- A function is made of :
  - Identifier (follows the same rules as variables for example : my_function_id())
  - Parameters : are variables that we give to the function, for example to calculate a second degree equation $ax^2 + bx + c = 0$ we'll be giving the parameters (a, b, c), to call the function we'll use : second_degree_equation(a, b, c).
  - a set of instructions
  - a return statement, as we said, a function can return a value (this is optional)

How to create a function ?

- The basic syntax for the functions is :

  def function_identifier (parameter1, parameter2,...):
  Set of instructions
  return something (if you want to)

- You are free to give one, two, three, n parameters (or none) depending on your needs ! the parameters must be separated by a “,”. This parameter can be any type, int, float, Boolean, string, list...etc
- *DON’T FORGET THE TWO POINTS “:” AND THE INDENTATION*, the set of instructions is contained in your function so you need to respect the indentation as we’ve seen in conditions and loops.
- The return instruction is optional, for example if your function prints a person’s informations, you don’t return anything so no need for the return (not recommended though).

Function parameters

- As we said previously a function can have a certain number of parameters let's have an example :

```
def body_mass_index(weight, height):
        BMI = weight / (height * height)
        return BMI
```

- We can also define a default value for a parameter, for example i wanna print the multiplication table for 6, generally we print from 1x6 to 10x6, it will suffice to some users but other users will want to print all the way to 20x6, what to do to satisfy both of them ? USE A DEFAULT PARAMETER

```
def mult_table(value,max=10):
        for i in range(1,max+1):
                print("ix",value,"=",(i*value))
```

Practice time!

Same exercise but you have to check the validity of the answer using a function !

Lists

- In python a ***list*** is an ***object*** that can contain other variables of any type, this is how it goes :
  - declaration : list_identifier = [] #you can also write = list()
  - you can add elements to your list at the creation : my_first_list = [12, -6, True, “yes even a string”, [“another list baby !”]]
- To ***add*** elements to your list **after** you declare it, this is what to do :

  my_second_list = []
  my_second_list.append(your element)
- To access an element of your ***list*** you can use something you’ve seen before in other languages : my_first_list[2], and this will print True .
- You can modify an element of the list this way : my_first_list[2] = False
- There are a huge number of functions (methods) that are used to : add elements, modify, delete, sort...etc and can all be found in the official documentation (it’s really important to learn to access and read documentations). https://docs.python.org/3/tutorial/datastructures.html

Tuples

- A ***tuple*** is an element that resembles a list, only difference, a set is immutable (you can't change it's values once created)
- Creating a ***tuple*** is as easy as creating a list : my_set_identifier = ()
- You can add values to the set this way :
    - my_first_set = (1.83, 79)
    - any_other_set = (1, 4, 7, 8)
    - You can create a **tuple** as result of an ***affectation*** after a ***function return***

Practice !

In this practice, we'll be taking the BMI example. Get the height and weight of five people, send them to a function that will return a list containing each person's BMI and send that list to another function that will do a quick classification (skinny, well shaped, fat...etc)

New structure, the dictionary

- Python dictionary is an unordered collection of items. While other compound data types have only value as an element, a dictionary has a **key: value** pair.
- If you need to pair an information stored with its value, the dictionary is your structure. Take last course's exercice about BMI, we could've used the dictionary to store the name of the person as a **key** and his personal informations as a **value.**

- In python, to create a dictionary you can use the following syntax :
  - my_variable_id = dict()
  - my_variable_id = {} #Just like with lists right ?
  - You can also initialize your dictionary with values like we've seen with the lists, remember the syntax **key:value** . ages_dict = **{'sofiane': 23, 'Lyes': 12}**
  - Or ages_dict = **dict({'sofiane': 23, 'Lyes': 12})**

Dictionary modification

- To add values to your dictionary, simple syntaxe : ***my_dict[key] = value***
- Your key can be of any type (String, integer,…)
- You can update a value the following way :

```
my_dict['Lyes'] = 12 #Oops i feel like i exaggerated
my_dict['Lyes'] = 18 #Now my key: value pair has been updated
```

- To access a the **value** of a **key,** all we have to do is use the variable : ***my_dict[key]***, for example :

```
name = "Lyes"
print("the age of ",name," is : ",my_dict[name])
```

- To get all the **keys** as a **list**, it's easy, just use : your_dict.keys()
- To get all the **values** as a **list**, well you guessed it : your_dict.values()

Parse through your dict !

- You'll obviously need to parse through all of your dictionary, here's what to do :

```
for key in my_dict :
    print(my_dict[key])
```

- You may need the **keys** and **values** directly, here's the solution for you :

```
for key, value in my_dict.items():
    print(key)
    print(value)
```

- There's many other ways to use your dictionaries depending on your needs, don't forget the check the official documentation for more useful methods : https://docs.python.org/3/tutorial/datastructures.html#dictionaries

Practice !

Now that we have used the lists to solve our problem, can we use dictionaries ?

Files, what and why ?

- Until now we didn’t save any information. Basically if you stop your program or turn off your computer while your program is still running, you won’t save any information.
- Files are a way to store your data within your computer and retrieve them anytime you want.
- Files on your computer have a name and extension for example : omc_members.csv (CSV for Comma-separated values, this is a format that’s we’ll be using a lot during this course) You can find others extensions here : https://www.computerhope.com/issues/ch001789.htm

Open a file

- In your computer, phone or tablet, if you want to read a book for example you need to ***open*** that file, and sames goes for programming in general, if you wanna read a file or write in it you need to ***open*** it, don't forget to ***close it*** when you finish you work.
- To open and close a file all you have to is :

```
omc_members = open(path, opening_mode)
omc_members.close()
```

  - The path to your file can either be an ***absolute*** or ***relative*** path. https://www.coffeecup.com/help/articles/absolute-vs-relative-pathslinks/
  - The opening modes are ***'r'*** to read, ***'w'*** to write (be careful this mode erases the content of the file) and ***'a'***, to add content at the end of the file.

Read your file and write on it

When you read your file you can either read : omc_file = open(‘omc.csv’, ‘r’)

- The whole content at onces

  omc_members = omc_file***.read()***

- Line by line :

```
omc_members = omc_file.readLines() # a list of all the lines of the file

for member in omc_members :
    print(member)
```

To write something in your file, it’s also simple : omc_file = open(omc.csv, ‘a’) ***#If i opened with ‘w’ it would’ve erased the whole content***

```
omc_file.write(‘yes i’m putting a new line’)
```

Don’t forget to close your file : omc_file.close()

Additional informations

- When you open your file, an error may occur, this is why we can use the *with* statement :

```
with open(path, open_mode) as my_file:

    # my bloc of instructions
```

- The write method we've seen lets us write strings only, two solutions :
  - Convert our variables to strings.
  - use the pickle module https://docs.python.org/3.1/library/pickle.html

Split and join

- When we're manipulating strings (in files for example) we may need the informations separately (in CSV format for example, informations are separated by a ‘,’), to do so, we have the ***split*** function. This functions **splits a string and returns a list.**

```
list_of_words = my_string.split(separator)
 #This separator can be anything it could be a blank in a casual text file or a coma in your csv files.
```

- If you need to add a person in the omc.csv file, you have to use the ***join*** method, that works in the opposite way of split. This function **joins the content of a list to make a string.**

```
personnal_informations = [‘sofiane’,23,’sofiane.medjkoune@protonmail.com’]
‘,’.join(personnal_informations)
```

Practice !

Let's work with the previous BMI exercise, only this time we'll write the personal informations in a csv file, read it, for each person calculate the BMI and add it to our csv file