Python variables and Basis
UCAB
Python Variables Uses And Explanations

Main features of Python

- Python is an amazing programming language, not only because of its ease, or how flexible it is, **cross-platform**, **multi-paradigm**, **works at a very high level**, **easy to learn**, **object oriented**, **interpreted**, a lot of frameworks that allow you to do machine learning (Keras), games (PyGame), web pages (Django), GUI's (PyQt5), but also Python has many things that make you ask yourself, 'why is it that other languages do not have this? ' The answer to this question is that Python is very high level, which allows you to focus on things at the very top of the machine (you don't have to worry about things like stack overflow or variables being cached), however, this also means that, in short, you will have less freedom to do things in some cases, but the ability to do a lot of things in very little time and in very few lines of code in most of these), let's get started.

What does PIP means??

- **PIP** stands for **python instalation package**, it is a python program that allows you to download specific Python packages, mainly libraries and frameworks.

- Unlike other languages such as C or C++, Python code does not necessarily have to be stored in a "main" function to be executed first, but rather, the declaration of the "main" function is done implicitly, this allows us to simply write the code and this could be interpreted as it is automatically stored inside "main".

Hello World with Python

- The Python installation includes an IDE (a program where to write the code) called IDLE, very basic, but that will be useful in our first steps in the language. If you already have your favorite code editor you can also use it.

- To create our first program we will open IDLE and select the menu File > New File to create a new document. Then, we will write the following.

```
print("¡Hello world!")
```

- This is real Python code - just one line! In it we call the built-in print() function and pass it a string as an argument to print to the screen. It is said to be built-in because it is a tool that the language always makes available to us in our programs. There are many others that we will get to know along the way.

- To be able to execute this small code we must first save it. For it, in IDLE we are going to go to the menu File > Save and we will save it in the desktop as hello.py. Now, let's remember that Python is an interpreted language. This means that there is no compiler program that transforms our source code (hello.py) and turns it into an executable file (hello.exe, for example); rather, there is a program called an interpreter to which we indicate that we want to execute a given file. All code editors can do this automatically (for example, in IDLE, by pressing F5), however, in this tutorial we are going to do it manually, that is, by invoking the interpreter from the terminal. This will give us a broader picture of how everything works in the Python world.

- Then, as we were saying, let's open the terminal. Every operating system has some shortcut for this. In Windows, you can press CTRL + R and type cmd, or search for the program named "Command Prompt". The first step is to go to the path where we have saved our file (the desktop) via the cd command. Once this is done, we run our Python script by typing python or py followed by the file name.

```
Linux/Mac:
> cd Desktop
> python hello.py
¡Hello world!

Windows:
> cd Desktop
> py hello.py
¡Hello world !
```

Congratulations! You have executed your first Python code.

- Remember that the programs we write in Python are by default console applications. Double-clicking on hello.py will cause the interpreter to run our file, but once the message is printed it will close automatically (as this is what happens with every program when it reaches the last line of code).

Python Variables

- Variables are one of the two basic components of any program. At its core, a program is composed of data and instructions that manipulate that data. Normally, data is stored in memory (RAM) so that we can access it. So what is a variable? A variable is a way to identify, in a simple way, a piece of data that is stored in the computer's memory. Imagine that a variable is a container in which a piece of data is stored, which can change during the flow of the program. A variable allows us to easily access this data to be manipulated and transformed. For example, suppose we want to display the result of adding 1 + 2. To display the result, we must tell the program where the data

is located in memory and, to do so, we use a variable:

```
suma = 1 + 2
print(suma)
```

Assigning a value to a variable in Python

- As we have seen in the previous example, the assignment operator "=" is used to assign a value (data) to a variable.
Three parts are involved in the assignment operation:

- The assignment operator =
- An identifier or variable name, to the left of the operator
- A literal, an expression, a function call or a combination of all of them to the right of the assignment operator.

Sample:

```
#Assign the variable <a> the value 1.
a = 1
#Assign to the variable <a> the result of the expression 3 * 4
a = 3 * 4
#Assigns to the variable <a> the string 'Pythonista'.
a = 'Pythonista'
```

- When we assign a value to a variable for the first time, the variable is said to be defined and initialized at that place. In a script or program written in Python, we can define variables anywhere in the script. However, it is a good practice to define the variables that we are going to use at the beginning.

- If we try to use a variable that has not been previously defined/initialized, the interpreter will show us an error:

```
>>> print(a)
Traceback (most recent call last):
File "<input>", line 1, in <module>
NameError: name 'a' is not defined
```

- To finish this section on variables in Python, let's introduce the concept of data type. When you assign a value to a variable, that value belongs to a set of values known as a data type. A data type defines a number of characteristics about that data and the variables that contain it, such as the operations that can be performed on it. In Python, the data types are numeric (integer, real, and complex), boolean (True, False), and character strings.

Let's see an example:

```
a = 1 # int type
b = 'Hello' # string type
```

- Unlike other languages, in Python it is not necessary to indicate the data type when defining a variable. Moreover, at any time, a variable that is of one type can become a variable of any other

type:

-It is said that Python (the interpreter) infers the data type to which a variable belongs at runtime, i.e., it is when the program is executed that it knows its data type and what operations can be performed with it.
Changing the value of a variable in Python

- To change the value of a variable in Python, simply assign a new value to it at any time and place after the definition.

Sample:

```
>>> a = 1
>>> print(a)
1
>>> b = 'Hello'
>>> a = 3
>>> print(a)
3
```

- As we can see in the previous example, we have modified the data of the variable a. In line 1 we have assigned it the value 1. In line 1 we have assigned it the value 1. Later, in line 5, we have updated its value to 3.

Literal Values

- As I mentioned in a previous section, a variable can be assigned a literal, an expression, a function call or a combination of all of them.

- Now, what is a literal? We have already seen examples of literals in the previous sections. A literal is nothing more than a raw data that can be assigned directly to a variable or be part of an expression.

In the following example, the variable a is assigned the literal 1:

```
a = 1
```

The literal 1 represents the numeric value of the integer 1.

There are several types of literals:

- Numeric (Integer, real and complex): 1, 3, 3.14, -1, 3.14j, ...
- Character strings: 'Hello', 'I like Python', ...
- Booleans: True and False

- The special literal None. None means no value and is widely used in programming.

Assigning a value to multiple variables

- Now you are going to discover how to assign the same value to multiple variables at the same time. If you have to define several variables with the same data, you can use the following structure:

```
>>> a = b = c = 1
>>> print(a)
1
>>> print(b)
1
>>> print(c)
1
```

Assign multiple values to multiple variables

- It is also possible to initialize several variables with a different value each as follows:

```
>>> a, b, c = 1, 2, 3
>>> print(a)
1
>>> print(b)
2
>>> print(c)
3
```

However, I advise against doing it this way because the code is less readable.

Variables and memory

- Finally, to end this section, I want to make it really clear to you what this is about a variable and how Python assigns memory addresses to data and variables.

- To begin with, I'll tell you that in Python everything is an object. Yes, we haven't looked at object-oriented programming with Python yet, but keep in mind that in Python everything is an object.
So what happens when I assign the variable a the value 1?
a = 1

- Actually, the variable a refers to the object that represents the integer with value 1. If you now create a new variable b and assign it the value 1 as well, b is referring to the same object as the variable a. In short, a and b refer to the same object and are therefore associated with the same memory address. This is not so in other languages, but this is Python, haha. In other languages a and b would be associated to different memory addresses.

- To help you understand it better I will introduce you to the id() function that comes with Python. The id() function returns a unique identifier of the object passed as a parameter. This identifier is an integer which is guaranteed to be unique and immutable for the lifetime of the object in memory.

☐ In the Python CPython implementation (the most common), this identifier is actually the in-memory address of the object.

Let's see all of the above with an example:

```
# a and b refer to the object representing the integer 1.
# they refer to the same memory address

>>> a = 1
>>> b = 1
>>> id(1)
4299329328
>>> id(a)
4299329328
>>> id(b)
4299329328

# b is mapped with the object representing the integer 2
# a and b refer to different memory addresses
# a keeps the reference to integer 1

>>> b = 2
>>> id(a)
4299329328
>>> id(b)
4299329360

# a is assigned with the value of b
# a and b refer to the same object and, therefore,
# to the same memory address

>>> a = b
>>> id(a)
4299329360
>>> a
2
```