Introduction to Python

Python is an easy-to-learn and a powerful Object-Oriented Programming language. It is a very high-level programming language.

Why Python?

1. **Easy to Use**: Python is comparatively an easier-to-use language as compared to other programming languages.
2. **Expressive Language**: The syntax of Python is closer to how you would write pseudocode. Which makes it capable of expressing the code's purpose better than many other languages.
3. **Interpreted Language**: Python is an interpreted language; this means that the Python installation interprets and executes the code a line-at-a-time.
4. Python is one of the most popular programming languages to be used in **Web Development** owing to the variety of Web Development platforms built over it like Django, Flask, etc.

Python Download

The very first step towards Python Programming would be to download the tools required to run the Python language. We will be using Python 3 for the course. You can download the latest version of Python 3 from **https://www.python.org/downloads/**

Note:- If you are using Windows OS, then while installing Python make sure that *"Add Python to PATH"* is checked.

Getting an IDE for writing programs:

You can use any IDE of your choice, however, you are recommended to use Jupyter Notebook. You can download it from **https://jupyter.org/install**

Working in Python

Once you have Python installed on your system, you are ready to work on it. You can work in Python in two different modes:-

a) **Interactive Mode:** In this mode, you type one command at a time and Python executes the same. Python's interactive interpreter is also called Python Shell.

b) **Script Mode**: In this mode, we save all our commands in the form of a program file and later run the entire script. After running the script, the whole program gets compiled and you'll see the overall output.

First Program in Python

As we are just getting started with Python, we will start with the most fundamental program which would involve printing a standard output to the console. The `print()` function is a way to print to the standard output. The syntax to use `print()` function is as follows:-

```
In[] : print(<Objects>)
```

- <Objects> means that it can be one or more comma-separated 'Objects' to be printed.
- <Objects> must be enclosed within parentheses.

Example: If we want to print "Hello, World!" in our code, we will write it in the following way:-

```
In[] : print("Hello, World!")
```

and, we get the output as:

```
Out[] : Hello, World!
```

Python executed the first line by *calling* the `print()` function. The string value of `Hello, World!` was *passed* to the function.

Note:- The quotes that are on either side of `Hello, World!` were not printed to the screen because they are used to tell Python that they contain a string. The quotation marks delineate where the string begins and ends.

Variables in Python

What are Variables?

A variable in Python represents a named location that refers to value and whose values can be used and processed during the program run. In other words, variables are labels/names to which we can assign value and use them as a reference to that value throughout the code.

Variables are fundamental to programming for two reasons:

- **Variables keep values accessible**: For example, The result of a time-consuming operation can be assigned to a variable, so that the operation need not be performed each time we need the result.
- **Variables give values context:** For example, The number 56 could mean lots of different things, such as the number of students in a class, or the average weight of all students in the class. Assigning the number 56 to a variable with a name like **num_students** would make more sense, to distinguish it from another variable **average_weight**, which would refer to the average weight of the students. This way we can have different variables pointing to different values.

How are Values Assigned to A Variable?

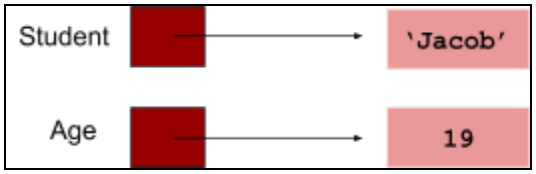
Values are assigned to a variable using a special symbol "=", called the **assignment operator**. An operator is a symbol, like = or +, that performs some operation on one or more values. For example, the + operator takes two numbers, one to the left of the

operator and one to the right, and adds them together. Likewise, the "=" operator takes a value to the right of the operator and assigns it to the name/label/variable on the left of the operator.

For Example: Now let us create a variable namely **Student** to hold a student's name and a variable **Age** to hold a student's age.

```
>>> Student = "Jacob"
>>> Age = 19
```

Python will internally create labels referring to these values as shown below:



Now, let us modify the first program we wrote.

```
greeting = "Hello, World!"
print(greeting)
```

Here, the Python program assigned the value of the string to a variable `greeting`, and then when we call `print(greeting)`, it prints the value that the variable, `greeting`, points to i.e. `"Hello, World!"`
We get the output as:-

```
Hello, World!
```

Naming a Variable

You must keep the following points in your mind while naming a variable:-

- Variable names can contain letters, numbers, and underscores.
- They cannot contain spaces.
- Variable names cannot start with a number.

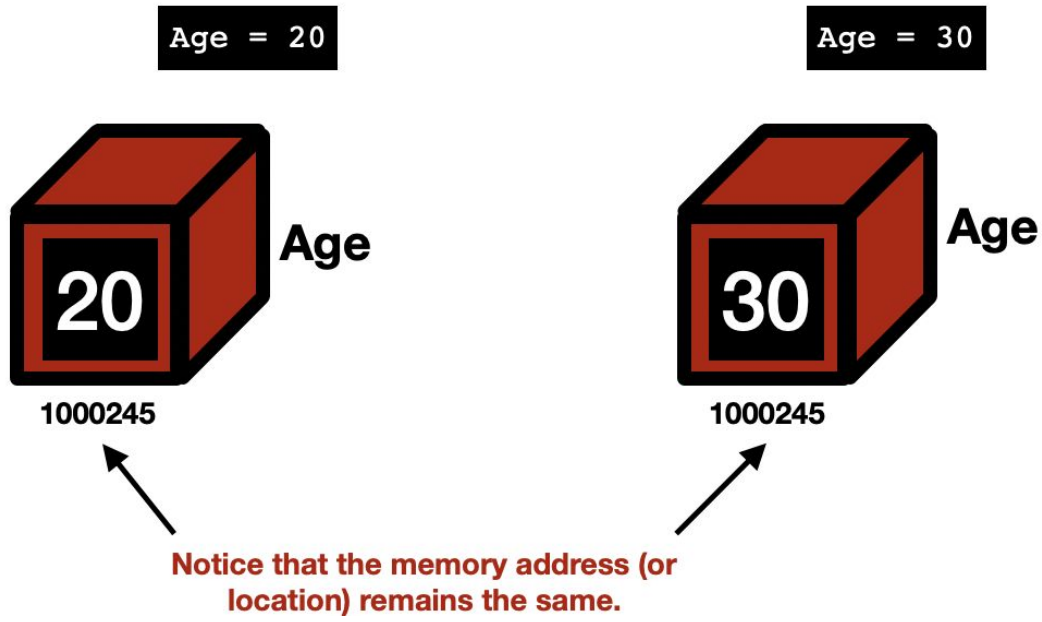
- Variable names are case sensitive. For example:- The variable names **Temp** and **temp** are different.
- While writing a program, creating self-explanatory variable names help a lot in increasing the readability of the code. However, too long names can clutter up the program and make it difficult to read.

Traditional Programming Languages' Variables in Memory

Let us study how variables and the values they are assigned, are represented in memory, in traditional programming languages like C, C++, Java, etc.
In these languages, variables are like **storage containers**. They are like named storage locations that store some value. In such cases, whenever we declare a new variable, a new storage location is given to that name/label and the value is stored at that named location. Now, whenever a new value is reassigned to that variable, the storage location remains the same. However, the value stored in the storage location is updated. This can be shown from the following illustration.

Consider the following script:

```
Age = 20
Age = 30 # Re-assigning a different value to the same variable
```



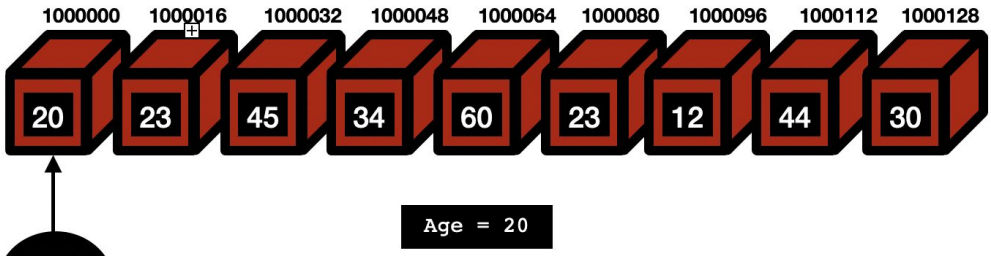
In the above script, when we declare a new variable `Age`, a container box/ Memory Location is named **Age** and the value 20 is stored in the memory address `1000245` with name/label, **Age**. Now, on reassigning the value 30 to `Age`, the value 30 is stored in the same memory location. This is how the variables behave in Traditional programming languages.

Python Variables in Memory

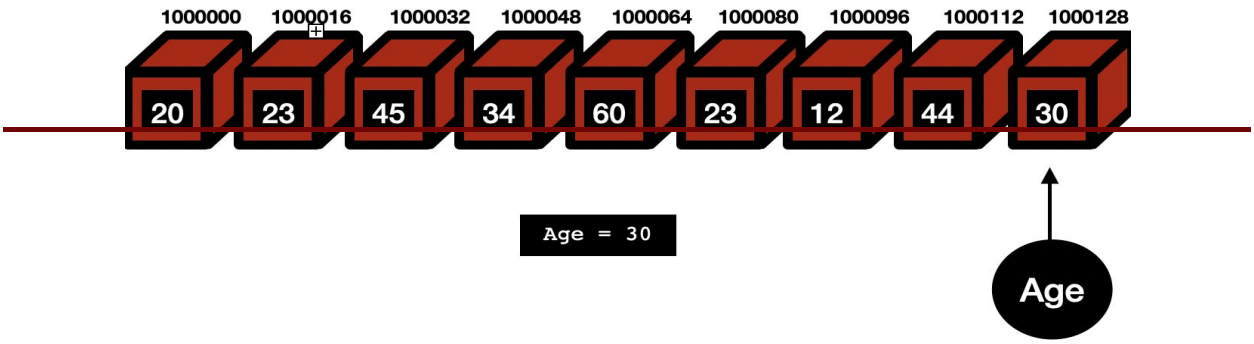
Python variables are not created in the form most other programming languages do. These variables do not have fixed locations, unlike other languages. The locations they refer/point to changes every time their value changes.

Python preloads some commonly used values in an area of memory. This memory space has values/literals at defined memory locations and all these locations have different addresses.

When we give the command, `Age = 20`, the variable `Age` is created as a label pointing to a memory location where 20 is already stored. If 20 is not present in any of the memory locations, then 20 is stored in an empty memory location with a unique address and then the `Age` is made to point to that memory location.



Now, when we give the second command, `Age = 30`, the label `Age` will not have the same location as earlier. Now it will point to a memory location where 30 is stored. So this time the memory location for the label `Age` is changed.



Data Types

Introduction

Data types are the classification or categorization of data items. Data types represent a kind of value that determines what operations can be performed on that data. Numeric, non-numeric, and Boolean (true/false) data are the most used data types. However, each programming language has its classification largely reflecting its programming philosophy. Python offers the following built-in data types:

- Numbers
    - Integers
    - Floating Point Numbers
    - Complex Numbers
- Strings
- Boolean Values
- List, Tuple, and Dictionary *(To be covered later in the course)*

Type Code	Description	Default Size (In Bytes)
`int`	Integers	4
`float`	Floating Point Numbers	4
`bool`	Boolean Values	1

Note:- If a variable has been assigned a value of some data type. It can be reassigned a value belonging to some other Data Type in the future.

```
a= "Raw" # String Data Type
a= 10 # Integer Data Type
a= 5.6 # Floating Point Number Data Type
a= 1+8j # Complex Number
a= True # Boolean Value
```

Python Numbers

Introduction

Number data types store numerical values. Python supports Integers, floating-point numbers, and complex numbers. They are defined as `int`, `float`, and `complex` classes.

- Integers can be of any length (Only limited by the memory available). They do not have a decimal point and can be positive or negative.
- A floating-point number is a number having a fractional part. The presence of a decimal point indicates a floating-point number. They have a precision of up to 15 digits.
- 1 is an integer, 1.0 is a floating-point number.
- Complex numbers are of the form, x + yj, where x is the real part and y is the imaginary part.

We can use the `type()` function to know which class a variable or a value belongs to. Similarly, the `isinstance()` function is used to check if an object belongs to a particular class.

Here are a few examples:-

```
b = 5
print(b, "is of type", type(b))
b = 2.0
print(b, "is of type", type(b))
b = 1+2j
print(b, "is complex number?", isinstance(b,complex))
```

And we will get the output as:

```
5 is of type <class 'int'>
```

```
2.0 is of type <class 'float'>
```

```
1+2j is complex number? True
```

Arithmetic Operators in Python

The Arithmetic Operators are used in Python in the same way as they are used in Mathematics.

OPERATOR	DESCRIPTION
+	Add two operands
-	Subtracts second operand from the first
*	Multiplies two operands
/	Divides numerator by denominator (Floating Point Division)
//	Divides numerator by denominator (Floor Division) - Acts as a floor function
**	**Exponent Operator**- The first operand raised to the power of the second operand
%	**Modulo Operator-** Calculates remainder left after dividing first by second

Let us see how these operators work:-

```
In[] : print(5 + 2) # Addition
Out[] : 7
In[] : print(5 - 2) # Subtraction
Out[] : 3
In[] : print(5 * 2) # Multiplication
Out[] : 10
In[] : print(5 / 2) # Floating Point Division
Out[] : 2.5
In[] : print(5 // 2) # Floor Division
Out[] : 2 # 5 divided by 2 gives 2.5 and value of floor(2.5) is 2
In[] : print(5 ** 2) # Calculate Exponent
Out[] : 25 # 5 raised to the power of 2 is 25
In[] : print(5 % 2) # Modulus
Out[] : 1 # Remainder 1 is left after dividing 5 by 2
```

Taking User Input

Developers often need to interact with users, either to get data or to provide some sort of result.

How to take User Input?

To get the input from the user interactively, we can use the built-in function, `input()`. This function is used in the following manner:

```
variable_to_hold_the_input_value = input(<Prompt to be displayed>)
```

For example:

```
In[] : age = input("What is your age?")
```

The above statement will display the prompt as:-

```
What is your age?_________ ←{User input here}
```

We will get the following interactive output:

```
In[] : name = input("Enter your name: ")
Enter your name: Rishabh #User Input
In[] : age = input("Enter your age: ")
Enter your age: 20 #User Input
In[] : name
Out[] : 'Rishabh'
In[] : age
Out[] : '19'
```

Note:- `input()` function always returns a value of the **String** type. Notice that in the above script the output for both name and age, Python has enclosed the output in quotes, like `'Rishabh'` and `'19'`, which implies that it is of **String** type. This is just because, whatever the user inputs in the `input()` function, it is treated as a **String**. This would mean that even if we input an integer value like 20, it will be treated like a string `'19'` and not an integer. Now, we will see how to read Numbers in the next section.

Reading Numbers

Python offers two functions `int()` and `float()` to be used with the `input()` function to convert the values received through `input()` into the respective numeric types integer and floating-point numbers. The steps will be:-

1. Use the `input()` function to read the user input.
2. Use the `int()` and `float()` function to convert the value *read* into integers and floating-point numbers, respectively. This process is called **Type Casting**.

The general way of taking Input:

```
variableRead = input(<Prompt to be displayed>)
updatedVariable = int(variableRead)
```

Here, `variableRead` is a String type that was read from the user. This string value will then be converted to Integer using the `int()` function and assigned to `updatedVariable`.
This can even be shortened to a single line of code as shown below:-

```
updatedVariable = int(input(<Prompt to be displayed>))
```

Let us take an example:-

```
In[] : age= int(input("Enter Your Age: "))
Enter Your Age: 19
In[] : age
Out[] : 19
```

Here, the output will be `19` and not `'19'`, i.e. the output is an Integer and not a String.

Similarly, if we want to read a floating-point number, we can do the following:-

```
In[] : weight= float(input("Enter Your Age: "))
Enter Your Weight: 65.5
In[] : weight
Out[] : 65.5
```