Variables in Python

What are Variables?

A variable in Python represents a named location that refers to a 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 many different things, such as the number of students in a class or the average weight of all students. 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.

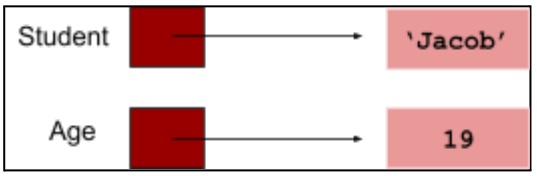
How are Values Assigned to A Variable?

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 operator's right and assigns it to the name/label/variable on the left of the operator.

For example, 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.

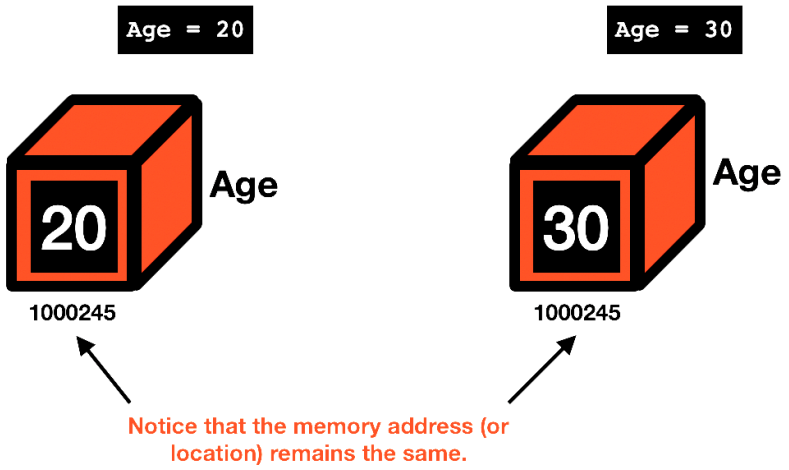
Traditional Programming Languages' Variables in Memory

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
```



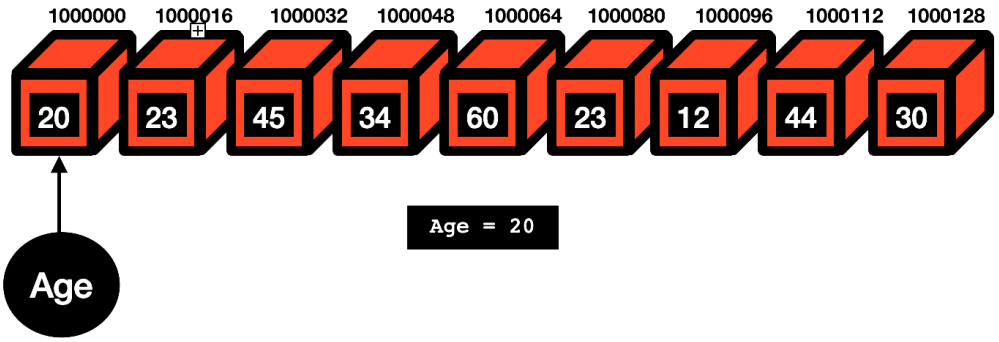
In the above script, when we declare a new variable `Age`, a container box/ Memory Location is named **Age** and the value 20 is stored in the memory address `1000245` with name/label, **Age**. Now, on reassigning the value 30 to `Age`, the value 30 is stored in the same memory location. This is how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



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.