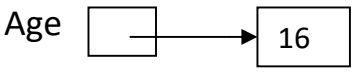
Variable and Datatypes

Variable and Assignments

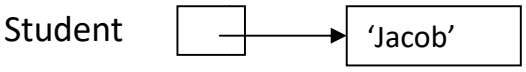
Variable represent named storage location whose value can be manipulated during program run. Python variables are created by assigning value of desired type to them e.g. to create a numeric variable, assign a numeric value to variable_name.

Age = 16



To create string variable, assign a string value to variable_name.

Student = 'Jacob'



Multiple Assignment

Python is very versatile with assignments.

1. Assigning same value to multiple variables

   ```
   a = b = c = 10
   ```

   It will assign value 10 to all three variables a, b & c.

2. Assigning multiple values to multiple variables

   ```
   x, y, z = 10, 20, 30
   ```

   It will assign values order wise i.e. x- 10, y – 20, z - 30

   ```
   x, y = 25, 50
   print(x, y)
   x - 25
   y -  50
   ```

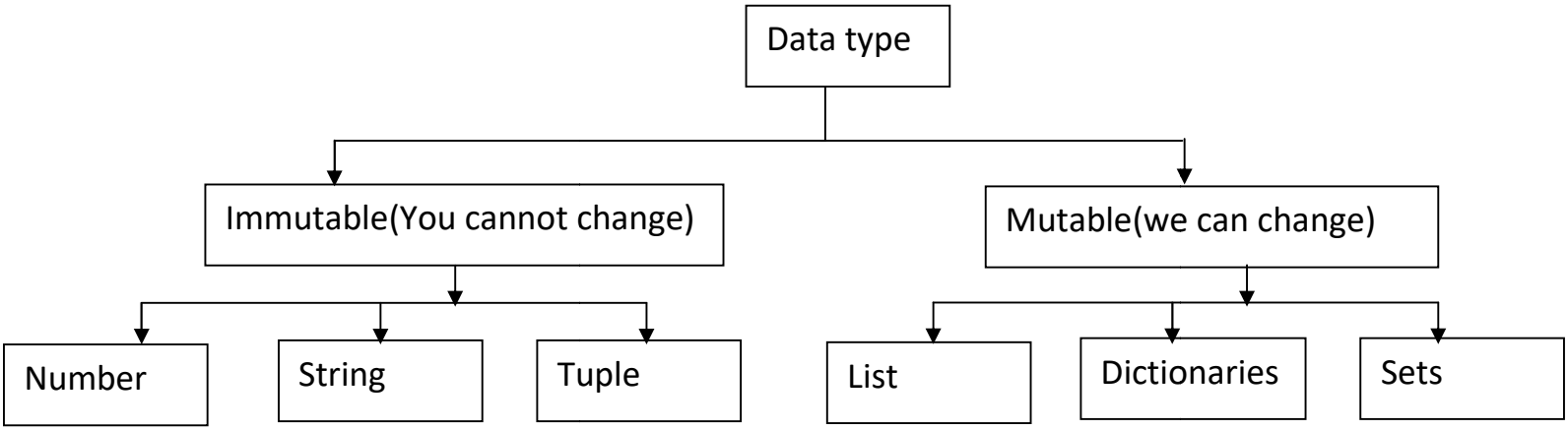
   if you want to swap values of x and y, you just need to write

   ```
   x, y  =  y, x
   print(x, y)
   x - 50
   y -  25
   ```

   Variable is created when you first assign a value to it. It also means that a variable is not created until some values is assigned to it.

Data types

Python is loosely typed language therefore no need to define the data type of variable. No need to declare variable before using them



Numbers

As it is clear by the name the number data types are used to store numeric value in Python.

Integers : Integers are whole numbers such as 5, 39, 1912, 0 etc. There are three types of integer in Python:

i) Plain integers ii) Long Integer iii) Boolean

Plan Integers

It is the normal integer representation and it uses 32 bits (4 bytes) generally to store a value. That means these represent numbers in the range -2147483648 to 2147483647. However, if the result of an operation would fall outside this range, the result is normally returned as a long integer.

Long Integers

The numbers larger than the range of plan integers are known as Long integers. There is no size limit to store a long integer in Python. That is, Python can store a long integer as big as supported by the underlying machine.

Boolean

These represent the truth values False and True. The Boolean type is a subtype pf plain integer, and Boolean values False and True behave like the value 0 and 1 respectively.

>>>bool(0)
False
>>>bool(1)
True

Float

A Number having fractional part is a floating-point number. For example 3.14159 is a floating pint number. The floating point number can be written in two forms:

i) Fractional Form
ii) Exponent Notation

Complex Number in Python

Python represent complex numbers in the form of A + Bj. That is, to represent imaginary number, Python uses j (or J) in place of traditional i. Os in Python j = √-1

String

A string data type lets you hold string data. That is any number or valid character into a set of quotation marks.

Python offers two string types

(i) ASCII Strings

This string holds string of zero or more ASCII characters. The value belonging to this string type is represented as string are represented traditionally.
for e.g. 'boy', "Zig 1953", "Her' s"

(ii) Unicode String

This string type can hold strings containing Unicode characters. Unicode string values are written as – normal string prefixed with letter u(not in quote).
for e.g. u 'abc', u "Hello", u "αβ"

Lists and Tuples

List in Python represent a list of comma-separated values of any data type between square brackets e.g.

[1, 2, 3, 4, 5]
['a', 'e', 'i', 'o', 'u']
['Vikas', 102, 79.5]
>>>a = [1, 2, 3, 4, 5]
to change first value in a list, we may write

>>>a[0]= 7

list item can be change because list is mutable type.

Tuples are those list which cannot be changed i.e. are not modifiable. Tuples are represented as list of comma separated values of any data type within parentheses. for e.g.

p = (1, 3, 4 , 6 , 7)

Dictionary

Dictionary data type is another feature in Python's hat. The dictionary is an unordered set of comma-separated key:value pairs within {}, with the requirement that within a dictionary, no two keys can be same.

for e.g.

>>> vowels = {'a':1, 'e':2, 'i':3, 'o':4, 'u':5}

>>> vowels['a']

1

>>>vowels['u']

5

Mutable and Immutable types

1. Immutable types

   The immutable types are those that can never change their values. In Python, the following types are immutable: integers, floating point numbers, Booleans, strings, tuples.

   In order to understand this, consider the code below:

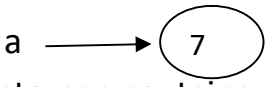
```
p = 5
q = p
r = 5
# will give 5, 5, 5
p =10
r = 7
q = r
```

   After reading the above code, you can say that values of integer variables p, q, r could be change effortlessly. You may think that integer types can change values. But holds : It is not the case. Let's see how:

In Python, variable-names are just the references to value-object i.e., data value. The variable-names do not store values themselves i.e. they are not storage containers.

```
a = 7
```

Python variable is a reference to a value i.e .

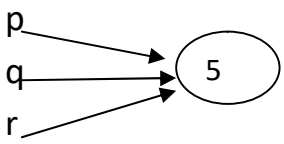


Not a storage container

Now consider the sample code given above. Internally how Python processes these assignment is explained.

So although it appears that the value of variable p/q/r is changing; values are not changing **"in place"** the fact is that the variable-names are instead made refer to new immutable integer object. (Changing **in place** means modifying the same value.)

p = 5
q = p
r = 5

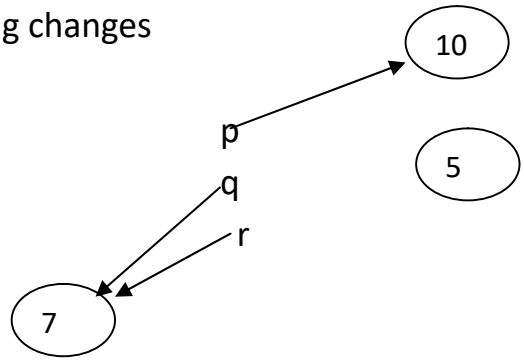


Integer value 5 is an immutable object to which variable-name p, q, r are currently referring

---

Now after making changes

p = 10
r = 7
q = r



p now references integer object 10

q and r now referencing integer object 7

2. Mutable types

The mutable types are those whose values **can be changed in place.** Examples of mutable data type in python are lists, dictionaries etc.

Variable Internals

Python is an object oriented language. Therefore Python calls every entity that stores any value or any type of data as an **object.**

Every Python object has three key attribute associated to it:

i) The type of an object

The type of an object determines the operations that can performed on the object. Built-in function type() returns the type of an object.

```
>>> a= 4
>>>type(a)
<class 'int'>
```

ii) The value of an object

It is the data-item contained in the object. For a literal, the value is the literal itself and for a variable the value is the data-item it is currently referencing. Using print() we can display value of an object.

```
>>> print(a)
>>> 4
```

iii) The id of an object.

The id of an object is generally the memory location of the object. Although id is implementation it returns the memory location of the object.

```
>>> id(4)
1585829728
>>> a = 4
>>> id(a)
1585829728
```

As it is clear from above example that value-objects are stored in a memory-location; a variable storing a value actually means in Python – it is referring to the memory-location of the value, as label.