Python Data Types and Variables

Variables are used to store information to be referenced and manipulated in a **computer language** . They also provide a way of labelling data with a detailed naming, so our programs can be understood more clearly by the reader and ourselves.

Python Variables

Every **variable** in Python is considered as an object. Variables in Python follow the standard nomenclature of an alphanumeric name beginning in a letter or underscore. Based on the **data type** of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. You do not need to declare variables before using them, or declare their type. Variable names are **case sensitive**. Most variables in Python are local in scope to their own function or class. **Global variables**, however, can be declared with the global keyword.

Assigning Values to Variables

When you assign a variable, you use the = symbol. The name of the variable goes on the left and the value you want to store in the variable goes on the right.

Example.

```
#integer example
x=9999
print("type of x is ", type(x))
#float example
y=3.141
print("The type of y is ", type(y))
#complex example
z=99+5j
print("The type of z is ", type(z))
```

Python Native Datatypes

A **Data type** provides a set of values from which an expression may take its values. The type defines the operations that can be done on the data, the meaning of the data, and the way values of that type can be stored. **Python** supports the following data types:

- Numbers
- String
- List
- Tuple
- Dictionary

Numbers

Python supports four distinct **numeric types** : integers, long, float and complex numbers. In addition, **Booleans** are a subtype of plain integers. Integers or int are positive or negative whole numbers with no **decimal point** . Long integers have unlimited precision and floats represent real numbers and are written with a decimal point dividing the integer and fractional parts. **Complex numbers** have a real and imaginary part, a + bc, where a is the real part and b is the imaginary part.

```
#integer example
x=9999
print("type of x is ", type(x))
#float example
y=3.141
print("The type of y is ", type(y))
#complex example
z=99+5j
print("The type of z is ", type(z))
```

output

```
Type of x is  < class 'int' >

The type of y is  < class 'float' >

The type of z is  < class 'complex' >
```

String

A **String** is an array of characters. They are formed by a list of characters, which is really an "array of characters". They are less useful when storing information for the computer to use. An important characteristic of each string is its length, which is the number of characters in it. There are numerous **algorithms** for processing strings, including for searching, sorting, comparing and transforming.

In Python, string is a sequence of **Unicode character** . Unicode was introduced to include every character in all languages and bring uniformity in encoding. We can create them simply by enclosing characters in quotes. Python treats single quotes the same as double quotes.

Ex.

```
str = "Hello World"   //double quotes

str1 = 'Hello World!'//using single quotes
```

Python strings are "immutable" which means they cannot be changed after they are created. Characters in a string can be accessed using the standard **[ ]** syntax and zero-based indexing.

Ex.

```
str = "Hello World"

print (str[0])

print (str[6:11])

print (str + " !!")

print (len(str))
```

output:-

```
H

World

Hello World !!

11
```

More about Python String will be discussed later in the chapter.

List

Python List is one of the most frequently used and very versatile datatype. Lists work similarly to strings: use the **len()** function and square brackets [ ] to access data, with the first element at index 0.

Example

```
weekdays = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']

print (weekdays[0])

print (weekdays[4])
```

Output

Monday

Friday

Tuple

A **tuple** is a container which holds a series of comma-separated values between parentheses. A tuple is similar to a list. Since, tuples are quite similar to **lists** , both of them are used in similar situations as well. The only the difference is that list is enclosed between square bracket, tuple between parenthesis and List have **mutable objects** whereas Tuple have immutable objects.

Example

```
my_Tuple_1 = (1,2,"Hello",3.14,"world")
print(my_Tuple_1)
print(my_Tuple_1[3])
my_Tuple_2 = (5,"six")
print(my_Tuple_1 + my_Tuple_2)
```

output

```
(1, 2, 'Hello', 3.14, 'world')
3.14
(1, 2, 'Hello', 3.14, 'world', 5, 'six')
```

Dictionary

Pyhton **Dictionaries** allow you store and retrieve related information in a way that means something both to humans and computers. Dictionaries are non-ordered and contain **"keys"** and **"values"** . Each key is unique and the values can be just about anything, but usually they are string, int, or float, or a list of these things. Like lists dictionaries can easily be changed, can be shrunk and grown ad libitum at run time. Dictionaries don't support the sequence operation of the sequence data types like strings, tuples and lists. Dictionaries belong to the built-in mapping type.

Example

```
my_Dictionay = {'ID': 1110, 'Name':'John', 'Age': 12}
print (my_Dictionay['ID'])
print (my_Dictionay['Age'])
#insert
my_Dictionay['Total Marks']=600
```

output

```
1110
12
{'Total Marks': 600, 'Age': 12, 'ID': 1110, 'Name': 'John'}
```