**DATE:13/08/2022**

**Assignment No:01**

**Introduction To Python**

**Python** is a general purpose, dynamic, [high-level](https://www.javatpoint.com/classification-of-programming-languages), and interpreted programming language. It supports Object Oriented programming approach to develop applications. It is simple and easy to learn and provides lots of high-level data structures.

Python is easy to learn yet powerful and versatile scripting language, which makes it attractive for Application Development.

Python's syntax and dynamic typing with its interpreted nature make it an ideal language for scripting and rapid application development.

**Why We Learn Python?**

Python provides many useful features to the programmer. These features make it most popular and widely used language. We have listed below few-essential feature of Python.

* Easy to use and Learn
* Expressive Language
* Interpreted Language
* Object-Oriented Language
* Open Source Language
* Extensible
* Learn Standard Library
* GUI Programming Support
* Integrated
* Embeddable
* Dynamic Memory Allocation
* Wide Range of Libraries and Frameworks

**Where Is Python Used:**

Python is a general-purpose, popular programming language and it is used in almost every technical field. The various areas of Python use are given below.

* Data Science
* Date Mining
* Desktop Applications
* Console-based Applications
* Mobile Applications
* Software Development
* Artificial Intelligence
* Web Applications
* Enterprise Applications
* 3D CAD Applications
* Machine Learning
* Computer Vision or Image Processing Applications.
* Speech Recognitions

**Python Popular Frameworks And Libraries**

Python has wide range of libraries and frameworks widely used in various fields such as machine learning, artificial intelligence, web applications, etc. We define some popular frameworks and libraries of Python as follows.

* **Web development (Server-side) -** Django Flask, Pyramid, CherryPy
* **GUIs based applications -** Tk, PyGTK, PyQt, PyJs, etc.
* **Machine Learning -** TensorFlow, PyTorch, **Scikit-learn**, Matplotlib, Scipy, etc.
* **Mathematics -** Numpy, Pandas, etc

**Python Variables**

Variable is a name that is used to refer to memory location. Python variable is also known as an identifier and used to hold value.

In Python, we don't need to specify the type of variable because Python is a infer language and smart enough to get variable type.

Variable names can be a group of both the letters and digits, but they have to begin with a letter or an underscore.

It is recommended to use lowercase letters for the variable name. Rahul and rahul both are two different variables.

**Identifier Naming**

Variables are the example of identifiers. An Identifier is used to identify the literals used in the program. The rules to name an identifier are given below.

* The first character of the variable must be an alphabet or underscore ( \_ ).
* All the characters except the first character may be an alphabet of lower-case(a-z), upper-case (A-Z), underscore, or digit (0-9).
* Identifier name must not contain any white-space, or special character (!, @, #, %, ^, &, \*).
* Identifier name must not be similar to any keyword defined in the language.
* Identifier names are case sensitive; for example, my name, and MyName is not the same.
* Examples of valid identifiers: a123, \_n, n\_9, etc.
* Examples of invalid identifiers: 1a, n%4, n 9, etc

**Declaring Variable and Assigning Values**

Python does not bind us to declare a variable before using it in the application. It allows us to create a variable at the required time.

We don't need to declare explicitly variable in Python. When we assign any value to the variable, that variable is declared automatically.

The equal (=) operator is used to assign value to a variable.

**Data Types**

Variables can hold values, and every value has a data-type. Python is a dynamically typed language; hence we do not need to define the type of the variable while declaring it. The interpreter implicitly binds the value with its type.

a = 10

The variable **a** holds integer value ten and we did not define its type. Python interpreter will automatically interpret variables **a** as an integer type.

Python enables us to check the type of the variable used in the program. Python provides us the **type()** function, which returns the type of the variable passed.

Consider the following example to define the values of different data types and checking its type.

a=20

b=”python”

c=”30.5

**print**(type(a))

**print**(type(b))

**print**(type(c))

**Output:**

<type 'int'>

<type 'str'>

<type 'float'>

**Python Literals**

Python Literals can be defined as data that is given in a variable or constant.

Python supports the following literals:

**1**.**String**:String literals can be formed by enclosing a text in the quotes. We can use both single as well as double quotes to create a string.

Example:

"Aman" , '12345'

**2.Boolean:**A Boolean literal can have any of the two values: True or False.

**Example - Boolean Literals**

x = (1 == True)

y = (2 == False)

**Output:**

x is True

y is False

**3.Integer:** It Is Used To Store The Neumeric Values.

Example:

A=5

B=10

**4.Float:** It is used to store the floating point values.

Example:

A=10.67

B=34.89

**Collection Literals:**

Python provides the four types of literal collection such as List literals, Tuple literals, Dict literals, and Set literals.

**List:**

* List contains items of different data types. Lists are mutable i.e., modifiable.
* The values stored in List are separated by comma(,) and enclosed within square brackets([]). We can store different types of data in a List.

**Example - List literals**

list=['John',678,20.4,'Peter']

list1=[456,'Andrew']

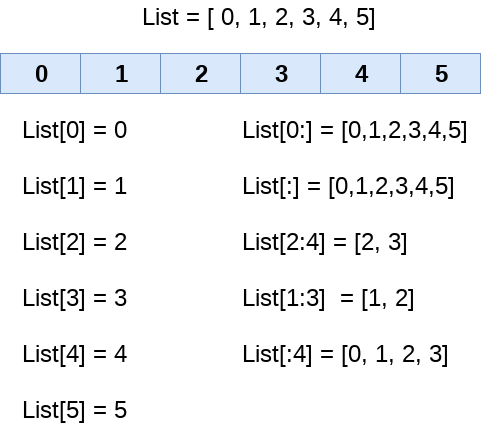
**print**(list)

**print**(list + list1)

**List Indexing And Slicing**

The indexing is processed in the same way as it happens with the strings. The elements of the list can be accessed by using the slice operator [].

The index starts from 0 and goes to length - 1. The first element of the list is stored at the 0th index, the second element of the list is stored at the 1st index, and so on.



We can get the sub-list of the list using the following syntax.

1. list\_varible(start:stop:step)

* The **start** denotes the starting index position of the list.
* The **stop** denotes the last index position of the list.
* The **step** is used to skip the nth element within a **start:stop**

Consider the following example:

1. list = [1,2,3,4,5,6,7]
2. **print**(list[0])
3. **print**(list[1])
4. **print(list**[2])
5. **print**(list[3])
6. # Slicing the elements
7. **print**(list[0:6])
8. # By default the index value is 0 so its starts from the 0th element and go for index -1.
9. **print**(list[:])
10. **print**(list[2:5])
11. **print**(list[1:6:2])

**Output:**

1

2

3

4

[1, 2, 3, 4, 5, 6]

[1, 2, 3, 4, 5, 6, 7]

[3, 4, 5]

[2, 4, 6]

Unlike other languages, Python provides the flexibility to use the negative indexing also. The negative indices are counted from the right. The last element (rightmost) of the list has the index -1; its adjacent left element is present at the index -2 and so on until the left-most elements are encountered.

Python Lists

Let's have a look at the following example where we will use negative indexing to access the elements of the list.

1. list = [1,2,3,4,5]
2. **print**(list[-1])
3. **print**(list[-3:])
4. **print**(list[:-1])
5. **print**(list[-3:-1])

**Output:**

5

[3, 4, 5]

[1, 2, 3, 4]

[3, 4]

As we discussed above, we can get an element by using negative indexing. In the above code, the first print statement returned the rightmost element of the list. The second print statement returned the sub-list, and so on.