Python Programming

Today we have discussed on :

1. Python - Basic Syntax

2. Python - Variable Types

1. Python - Basic Syntax:-

The Python language has many similarities to Perl, C, and Java. However, there are some definite differences between the languages.

First Python Program through Command Prompt:

- Open the Command Prompt
- To add the Python directory to the path for a particular session in Windows −
- At the command prompt − type path %path%;C:\Python and press Enter.
- Now you are on Python Prompt
- Type the following text at the Python prompt and press the Enter −

```
>>> print "Hello, Python!"
```

First Python Program through Integrated Development Environment:

You can also run Python program from a Graphical User Interface (GUI) as well, if you have a GUI application on your system that support Python.

- **PythonWin** is the first Windows interface for Python and is an IDE with a GUI.

Note:- Make sure the Python environment is properly set up and working perfectly fine.

Let us write a simple Python program in a script. Python files have extension .py. Type the following source code in a test.py file –

```
print "Hello, Python!"
```

We assume that you have Python interpreter set in PATH variable. Now, try to run this program as follows−

```
$ python test.py
```

This produces the following result –

```
Hello, Python!
```

Python Identifiers:-

A Python identifier is a name used to identify a variable, function, class, module or other object. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).

Python does not allow punctuation characters such as @, $, and % within identifiers. Python is a case sensitive programming language. Thus, Manpower and manpower are two different identifiers in Python.

Here are naming conventions for Python identifiers –

- Class names start with an uppercase letter. All other identifiers start with a lowercase letter.
- Starting an identifier with a single leading underscore indicates that the identifier is private.
- Starting an identifier with two leading underscores indicates a strongly private identifier.
- If the identifier also ends with two trailing underscores, the identifier is a language-defined special name.

Reserved Keywords:-

The following list shows the Python keywords. These are reserved words and you cannot use them as constant or variable or any other identifier names. All the Python keywords contain lowercase letters only.

and, exec, not, assert, finally, or, break, for, pass, class, from, print, continue, global, raise, def, if, return, del, import, try, elif, in, while, else, is, with, except, lambda, yield

2. Python - Variable Types:

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

Variable Names:-

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume). Rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)

- Variable names are case-sensitive (age, Age and AGE are three different variables)

Example

Legal variable names:

```
myvar = "John"

my_var = "John"

_my_var = "John"

myVar = "John"

MYVAR = "John"

myvar2 = "John"
```

Assigning Values to Variables:-

Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.

The operand to the left of the = operator is the name of the variable and the operand to the right of the = operator is the value stored in the variable.

For example –

```
#!/usr/bin/python
counter = 100
miles   = 1000.0
name    = "Lalit"
print counter
print miles
print name
```

Here, 100, 1000.0 and "John" are the values assigned to counter, miles, and name variables, respectively. This produces the following result –

```
100
1000.0
Lalit
```

Multiple Assignment:-

One Value to Multiple Variables:

Python allows you to assign a single value to several variables simultaneously.

For example –

```
a = b = c = 1
```

Other example -

```
x = y = z = "Orange"
print(x)
print(y)
print(z)
```

This produces the following result –

```
Orange
Orange
Orange
```

<u>**Many Values to Multiple Variables:**</u>

Python allows you to assign values to multiple variables in one line:

For example –

```
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

This produces the following result –

```
Orange

Banana

Cherry
```

Python Standard Data Types

The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.

Python has five standard data types –

i. Numbers

ii. String

iii. List

iv. Tuple

v. Dictionary

i. <u>Python Numbers Data Type:</u>

Number data types store numeric values. Number objects are created when you assign a value to them.

For example –

var1 = 1

var2 = 10

Python supports four different numerical types –

1. int (signed integers)

   For example:- int 10, int 100, int -786, int 080

2. long (long integers, they can also be represented in octal and hexadecimal)

   For Example:- long 51924361L, long -0x19323L

3. float (floating point real values)

   For example:- float 9.0, float 5.4, float 15.20

4. complex (complex numbers)

   For example:- complex 3.14j, complex 9.322e-36j

ii. <u>Python Strings:</u>

Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes. Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.

The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.

For example –

```
#!/usr/bin/python

str = 'Hello World!'

print str

print str[0]

print str[2:5]

print str[2:]

print str * 2

print str + "TEST"
```

This will produce the following result –

```
Hello World!

H

llo
```

```
llo World!
Hello World!Hello World!
Hello World!TEST
```

iii. Python Lists:-

Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]). To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.

The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

For example –

```
#!/usr/bin/python
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']
print list
print list[0]
print list[1:3]

print list[2:]

print tinylist * 2

print list + tinylist
```

This produce the following result –

```
['abcd', 786, 2.23, 'john', 70.2]

abcd

[786, 2.23]

[2.23, 'john', 70.2]

[123, 'john', 123, 'john']

['abcd', 786, 2.23, 'john', 70.2, 123, 'john']
```

iv. Python Tuples

A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.

The main differences between lists and tuples are: Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated. Tuples can be thought of as read-only lists.

For example

```
#!/usr/bin/python
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2  )
tinytuple = (123, 'john')
print tuple
print tuple[0]
print tuple[1:3]
print tuple[2:]
print tinytuple * 2
print tuple + tinytuple
```

This produce the following result –

```
('abcd', 786, 2.23, 'john', 70.2)
abcd
(786, 2.23)
(2.23, 'john', 70.2)
(123, 'john', 123, 'john')
```

```
('abcd', 786, 2.23, 'john', 70.2, 123, 'john')
```

Python Dictionary:

Python's dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).

For example –

```
#!/usr/bin/python

dict = {}

dict['one'] = "This is one"

dict[2]     = "This is two"

tinydict = {'name': 'john','code':6734, 'dept': 'sales'}

print dict['one']

print dict[2]

print tinydict

print tinydict.keys()
print tinydict.values()
```

This produce the following result –

```
This is one
This is two
{'dept': 'sales', 'code': 6734, 'name': 'john'}
['dept', 'code', 'name']
['sales', 6734, 'john']
```

Data Type Conversion

Sometimes, you may need to perform conversions between the built-in types. To convert between types, you simply use the type name as a function.

There are several built-in functions to perform conversion from one data type to another. These functions return a new object representing the converted value.

1. int(x [,base])

Converts x to an integer. base specifies the base if x is a string.

2. long(x [,base])

Converts x to a long integer. base specifies the base if x is a string.

3. float(x)

Converts x to a floating-point number.

4. complex(real [,imag])

Creates a complex number.

5. str(x)

Converts object x to a string representation.

6. repr(x)

Converts object x to an expression string.

7. eval(str)

Evaluates a string and returns an object.

8. tuple(s)

Converts s to a tuple.

9. list(s)

Converts s to a list.

10. set(s)

Converts s to a set.

11. dict(d)

Creates a dictionary. d must be a sequence of (key,value) tuples.

12. frozenset(s)

Converts s to a frozen set.

13. chr(x)

Converts an integer to a character.

14. unichr(x)

Converts an integer to a Unicode character.

15. ord(x)

Converts a single character to its integer value.

16. hex(x)

Converts an integer to a hexadecimal string.

17. oct(x)

Converts an integer to an octal string.

Note:- In next class we will discuss on Python basic Operators.