Unit – I Getting started with python
Introduction to python, features, program output, program input and raw_input(),comments in python, operators, Code blocks and indentation.

Unit – I Getting started with python

Introduction to python:

Python is an elegant and robust programming language that delivers both the power and general applicability of traditional compiled languages with the ease of use (and then some) of simpler scripting and interpreted languages. It allows you to get the job done, and then read what you wrote later.

Work on Python began in late 1989 by Guido van Rossum, then at CWI (Centrum voor Wiskunde en Informatica, the National Research Institute for Mathematics and Computer Science) in the Netherlands.It was eventually released for public distribution in early 1991.

Python is a high-level scripting language which can be used for a wide variety of text processing, system administration and internet-related tasks. Unlike many similar languages, its core language is very small and easy to master, while allowing the addition of modules to perform a virtually limitless variety of tasks. Python is a true object-oriented language, and is available on a wide variety of platforms. There's even a python interpreter written entirely in Java, further enhancing python's position as an excellent solution for internet-based problems.

Features:

Python is a dynamic, high level, free open source and interpreted programming language. It supports object-oriented programming as well as procedural oriented programming. In Python, we don't need to declare the type of variable because it is a dynamic typed language.

There are a few features of python which are different than other programming languages.

High Level: Python is a high-level language.When we write programs in python, we do not need to remember the system architecture, nor do we need to manage the memory.
Object Oriented: Object-oriented programming (OOP) adds another dimension to structured and procedural languages where data and logic are discrete elements of programming. OOP allows for associating specific behaviors, characteristics, and/or capabilities with the data that they execute on or are representative of. Python is an object-oriented (OO) language, all the way down to its core. However, Python is not just an OO language like Java or Ruby. It is actually a pleasant mix of multiple programming paradigms. For instance, it even borrows a few things from functional languages like Lisp and Haskell.

Scalable: The term "scalable" is most often applied to measuring hardware throughput and usually refers to additional performance when new hardware is added to a system. Python provides basic building blocks on which you can build an application, and as those needs expand and grow, Python's pluggable and modular architecture allows your project to flourish as well as maintain manageability.

Extensible: Extensibility in a language provides engineers with the flexibility to add-on or customize their tools to be more productive, and to develop in a shorter period of time. Although this feature is selfevident in mainstream third-generation languages (3GLs) such as C, C++, and even Java, the ease of writing extensions to Python in C is a real strength of Python. Furthermore, tools like PyRex, which understands a mix of C and Python, make writing extensions even easier as they compile everything to C for you.

Python extensions can be written in C and C++ for the standard implementation of Python in C (also known as CPython). The Java language implementation of Python is called Jython, so extensions would be written using Java. Finally, there is IronPython, the C# implementation for the .NET or Mono platforms. You can extend IronPython in C# or Visual Basic.NET.

Portable: Python can run on a wide variety of hardware platforms and has the same interface on all platforms.

Easy to Learn: Python has relatively few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language in a relatively short period of time. What may perhaps be new to beginners is the OO nature of Python. Those who are not fully versed in the ways of OOP may be apprehensive about jumping straight into Python, but OOP is neither necessary nor mandatory. Getting started is easy, and you can pick up OOP and use when you are ready to.

Easy to Read: Python code is more clearly defined and visible to the eyes.

Easy to Maintain: Python's source code is fairly easy-to-maintain. Much of Python's success is that source code is fairly easy to maintain, dependent, of course, on size and complexity.

Robust: When your Python crashes due to errors, the interpreter dumps out a "stack trace" full of useful information such as why your program crashed and where in the code (file name, line number, function call, etc.) the error took place. These errors are known as exceptions. Python even gives you the ability to monitor for errors and take an evasive course of action if such an error does occur during runtime.

These exception handlers can take steps such as defusing the problem, redirecting program flow, perform cleanup or maintenance measures, shutting down the application gracefully, or just ignoring it.

Python's robustness is beneficial for both the software designer and the user.

Large Standard Library: Python has a large standard library which provides rich set of module and functions so you do not have to write your own code for every single thing. There are many libraries present in python for such as regular expressions, unit-testing, web browsers etc.

Dynamically Typed Language: Python is dynamically-typed language. That means the type (for example- int, double, long etc) for a variable is decided at run time not in advance because of this feature we don’t need to specify the type of variable.

Interpreted and (Byte-) Compiled: Python is an Interpreted Language. because python code is executed line by line at a time. like other language c, c++, java etc there is no need to compile python code this makes it easier to debug our code.The source code of python is converted into an immediate form called bytecode.

Program output, program input and raw_input():

A Program needs to interact with the user to accomplish the desired task; this is done using Input-Output facility. Input means the data entered by the user of the program. In python, we have input() and raw_input ( ) function available for Input.

1) input()

```
Syntax:
input (expression)
```

If prompt is present, it is displayed on monitor, after which the user can provide data from keyboard. Input takes whatever is typed from the keyboard and evaluates it. As the input provided is evaluated, it expects valid python expression. If the input provided is not correct then either syntax error or exception is raised by python.

```
Example:
>>>x= input ("Enter data:")
Enter data:  34.78
>>>print(x)
34.78
```

2) raw_input()

```
Syntax:
raw_input (expression)
```

This input method fairly works in older versions (like 2.x).

If prompt is present, it is displayed on the monitor after which user can provide the data from keyboard. The function takes exactly what is typed from keyboard, convert it to string and then return it to the variable on LHS of '='.

```
Example: In interactive mode
>>>x=raw_input ('Enter your name: ')
```

```
Enter your name: ABC
```

x is a variable which will get the string (ABC), typed by user during the execution of program. Typing of data for the raw_input function is terminated by enter key.

We can use raw_input() to enter numeric data also. In that case we typecast, i.e., change the data type using function, the string data accepted from user to appropriate Numeric type.

Example:

```
>>>y=int(raw_input("Enter your roll no."))
Enter your roll no. 5
```

It will convert the accepted string i.e., 5 to integer before assigning it to 'y'.

Print statement

Syntax:

```
print (expression/constant/variable)
```

Print evaluates the expression before printing it on the monitor. Print statement outputs an entire (complete) line and then goes to next line for subsequent output (s). To print more than one item on a single line, comma (,) may be used.

Example:

```
>>> print ("Hello")

Hello

>>> print (5.5)

5.5

>>> print (4+6)

10
```

comments in python:
Comments can be used to explain Python code.
Comments can be used to make the code more readable.
Comments can be used to prevent execution when testing code.
Comments starts with a #, and Python will ignore them:

```
#This is a comment
print("Hello, World!")
```

Multi Line Comments: Python does not really have a syntax for multi line comments.

To add a multiline comment you could insert a # for each line:

```
#This is a comment

#written in

#more than just one line
```

```
print("Hello, World!")
```

Operators:

The standard mathematical operators that you are familiar with work the same way in Python as in most other languages.
+ - * / // % **
Addition, subtraction, multiplication, division, and modulus (remainder) are all part of the standard set of operators. Python has two division operators, a single slash character for classic division and a doubleslash for "floor" division (rounds down to nearest whole number). Classic division means that if the operands are both integers, it will perform floor division, while for floating point numbers, it represents true division. If true division is enabled, then the division operator will always perform that operation, regardless of operand types.
There is also an exponentiation operator, the double star/asterisk (**). Although we are emphasizing the mathematical nature of these operators, please note that some of these operators are overloaded for use with other data types as well, for example, strings and lists.
Example:

```
>>> print -2 * 4 + 3 ** 2
1
```

As you can see, the operator precedence is what you expect: + and - are at the bottom, followed by *, /, //, and %; then comes the unary + and -, and finally, we have ** at the top. ((3 ** 2) is calculated first, followed by (-2 * 4), then both results are summed together.)

Python also provides the standard comparison operators, which return a Boolean value indicating the truthfulness of the expression:

< <= > >= == != <>
Trying out some of the comparison operators we get:

```
>>> 2 < 4
True
>>> 2 == 4
False
>>> 2 > 4
False
>>> 6.2 <= 6
False
>>> 6.2 <= 6.2
True
>>> 6.2 <= 6.20001
True
```

Python currently supports two "not equal" comparison operators, != and <>.

Python also provides the expression conjunction operators:

and or not

We can use these operations to chain together arbitrary expressions and logically combine the Boolean results:

```
>>> 2 < 4 and 2 == 4
False
>>> 2 > 4 or 2 < 4
True
>>> not 6.2 <= 6
True
>>> 3 < 4 < 5
True
```

The last example is an expression that may be invalid in other languages, but in Python it is really a short way of saying:

```
>>> 3 < 4 and 4 < 5
```

Code blocks and indentation:

Code blocks are identified by indentation rather than using symbols like curly braces. Without extra symbols, programs are easier to read. Also, indentation clearly identifies which block of code a statement belongs to. Of course, code blocks can consist of single statements, too.

Indentation in Python refers to the (spaces and tabs) that are used at the beginning of a statement. The statements with the same indentation belong to the same group called a suite or code block.

Consider the example of a correctly indented Python code statement mentioned below.

```
Example:
if a==1:
   print(a)
   if b==2:
      print(b)
print('end')
```

In the above code, the first and last line of the statement is related to the same suite because there is no indentation in front of them. So after executing first "if statement", the Python interpreter will go into the next statement, and if the condition is not true it will execute the last line of the statement.