**PYTHON**



**Submitted By -**

**Group – 7**

**Anupam Das – 1401CS52**

**Sathya Prakash – 1401CS06**

**Jitendra Kumar – 1401CS19**

1. **Full name** of the programming language :: **Python**

**Introduction:**

**Origin:**

* **Python** was conceived in the late **1980s**, and its implementation began in December 1989 by **Guido van Rossum** at **Centrum Wiskunde & Informatica (CWI)** in the Netherlands as a successor to the ABC language (itself inspired by SETL) capable of exception handling and interfacing with the operating system Amoeba.
* **Python** is an **open Source Scripting language** named after **Monty Python**.

**Creator:**

* Van Rossum is Python's principal author.



**Publications:**

* **Python 2.0** was released on **16 October 2000** and had many major new features, including a cycle-detecting garbage collector and support for Unicode. With this release the development process was changed and became more transparent and community-backed.
* **Python 3.0** (which early in its development was commonly referred to as **Python 3000** or **py3k**), a major, backwards-incompatible release, was released on **3 December 2008** after a long period of testing. Many of its major features have been backported to the backwards-compatible Python 2.6.x and 2.7.x version series.

1. **Paradigm ::**

* Python is a **multi-paradigm programming language**:
  + **object-oriented programming**

Everything in Python is an object that has:

* an identity(id)
* value (mutable or immutable)

But it does not support strong encapsulation.

This is what is meant when we say, for example, that functions are first-class objects. Functions, classes, strings, and even types are objects in Python: like any object, they have a type, they can be passed as function arguments, and they may have methods and properties. In this understanding, Python is an object-oriented language.

However, unlike Java, Python does not impose object-oriented programming as the main programming paradigm. It is perfectly viable for a Python project to not be object-oriented, i.e. to use no or very few class definitions, class inheritance, or any other mechanisms that are specific to object-oriented programming.

The Python language provides a simple yet powerful syntax called ‘**decorators’**. A decorator is a function or a class that wraps (or decorates) a function or a method. The ‘decorated’ function or method will replace the original ‘undecorated’ function or method. Because functions are first-class objects in Python, this can be done ‘manually’, but using the @decorator syntax is clearer and thus preferred.

Examples:

* Decorators:

def foo():

*# do something*

def decorator(func):

*# manipulate func*

return func

foo = decorator(foo) *# Manually decorate*

@decorator

def bar():

*# Do something*

*# bar() is decorated*

* **OOP:**

class Employee:

'Common base class for all employees'

empCount = 0

def \_\_init\_\_(self, name, salary):

self.name = name

self.salary = salary

Employee.empCount += 1

def displayCount(self):

print "Total Employee %d" % Employee.empCount

def displayEmployee(self):

print "Name : ", self.name, ", Salary: ", self.salary

* + **structured programming**

In python, we make extensive use of subroutines, block structures, for and while loops—in contrast to using simple tests and jumps such as the goto statement which could lead to "spaghetti code" which is difficult both to follow and to maintain. Thus improving the clarity, quality, and development time of a computer program.

* + **functional programming** and **aspect-oriented programming** (including by **metaprogramming** and **metaobjects** (magic methods)) too supported.

1. Python is an **interpreted** **language**. This means that there is software on your computer that reads the Python code, and sends the "instructions" to the machine.

It is a programming language for which most of its implementations execute instructions directly, without previously compiling a program into machine-language instructions. The interpreter executes the program directly, translating each statement into a sequence of one or more subroutines already compiled into machine code.

Also, it does not compile to the native machine's code. Instead, it compiles to a byte code that is used by a virtual machine. The virtual machine is itself a compiled program. This is very similar to how Java works;

> If we would like to run a .py program

Assumed that our Python installation directory is

C:\Python24. And our .py file resides in C:\Temp and

is called Test.py.

We need to go to the command line on our OS.

At the command prompt type

$ C:\Python24\Python.exe C:\Temp\Test.py

This is the longhand way of doing it.

A shorter way is to add Python to your environment path.

Then we can shorten the command to:

$ Python C:\Temp\Test.py