Practical No. 1: Install and configure Python IDE

Installing Python in Windows:

- Open any internet browser. Type http:///www.Python.org/downloads/ in address bar and Enter.
- Home page of Python will have displayed as shown in Fig. 1

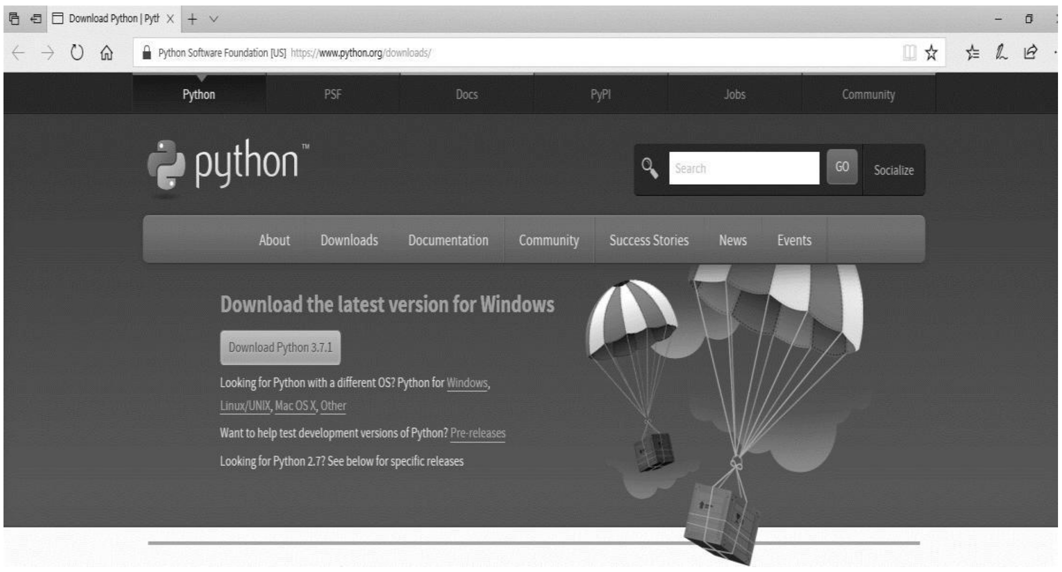
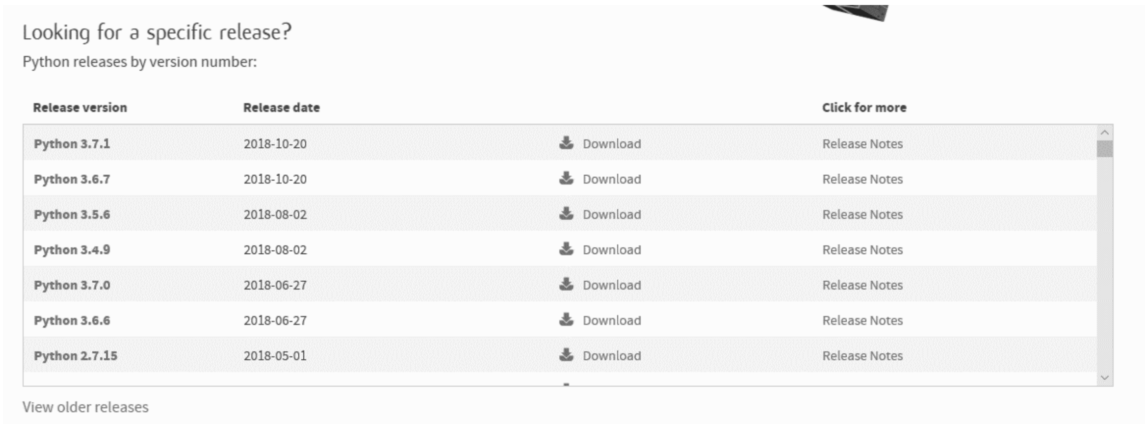

Python
PSF
Docs
PyPI
Jobs
Community
python
About
Downloads
Documentation
Community
Success Stories
News
Events
Download the latest version for Windows
Download Python 3.7.1
Looking for Python with a different OS? Python for Windows, Linux/UNIX, Mac OS X, Other
Want to help test development versions of Python? Pre-releases
Looking for Python 2.7? See below for specific releases

Fig. 1: Home Page

- Click on download the latest version for windows, which shows latest version as shown in Fig. 2



Looking for a specific release?

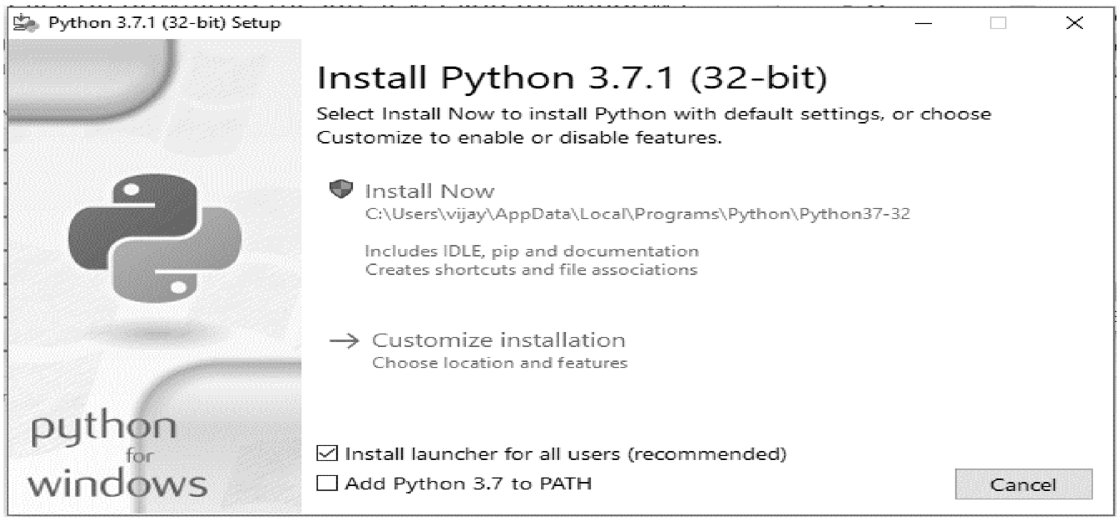
Python releases by version number:

Release version	Release date		Click for more
Python 3.7.1	2018-10-20	Download	Release Notes
Python 3.6.7	2018-10-20	Download	Release Notes
Python 3.5.6	2018-08-02	Download	Release Notes
Python 3.4.9	2018-08-02	Download	Release Notes
Python 3.7.0	2018-06-27	Download	Release Notes
Python 3.6.6	2018-06-27	Download	Release Notes
Python 2.7.15	2018-05-01	Download	Release Notes

View older releases

Fig.2: Python release versions

- Open the Python 3.7.1 version pack and double click on it to start installation and installation windows will be open as shown in Fig. 3.



Install Python 3.7.1 (32-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

Install Now
C:\Users\vijay\AppData\Local\Programs\Python\Python37-32

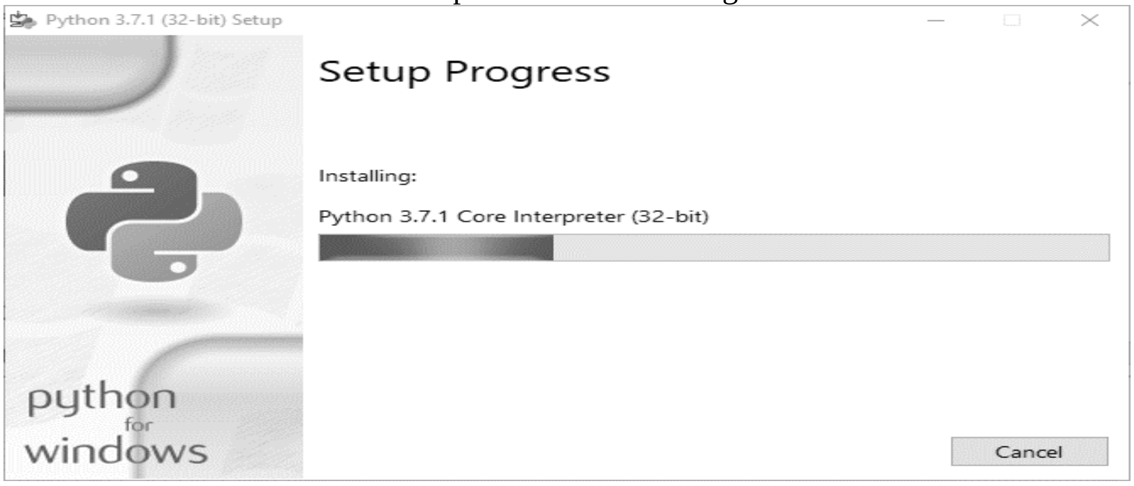
Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
☐ Add Python 3.7 to PATH

Fig. 3: Installation Type

- Click on next install now for installation and then Setup progress windows will be opened as shown in Fig. 4.



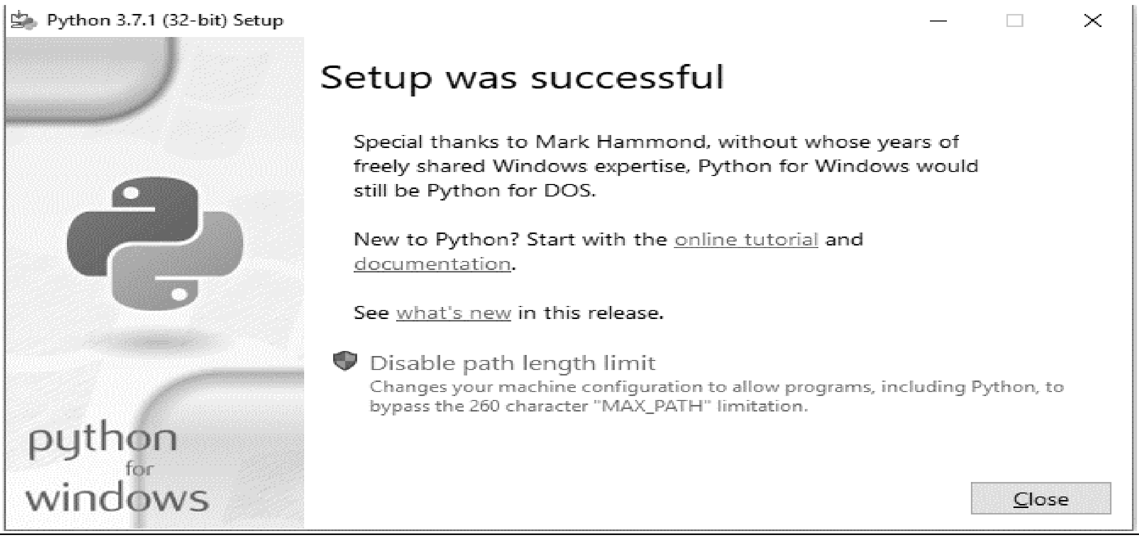
Setup Progress

Installing:

Python 3.7.1 Core Interpreter (32-bit)

Fig. 4: Setup Progress

- After complete the installation, Click on close button in the windows as shown in Fig. 5.



Setup was successful

Special thanks to Mark Hammond, without whose years of freely shared Windows expertise, Python for Windows would still be Python for DOS.

New to Python? Start with the online tutorial and documentation.

See what's new in this release.

Disable path length limit
Changes your machine configuration to allow programs, including Python, to bypass the 260 character "MAX_PATH" limitation.

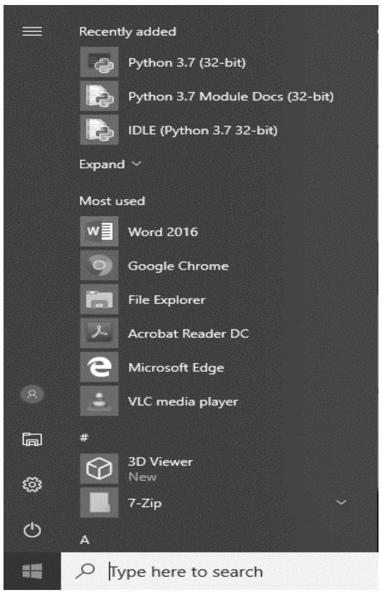
Fig. 5: Setup

Completion Starting Python in different

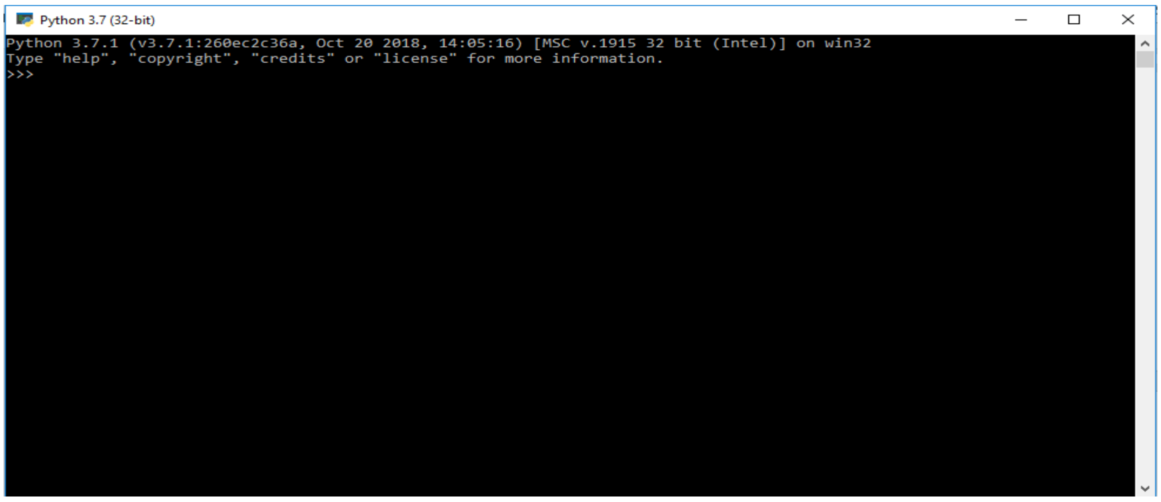
modes:

1) Starting Python (Command Line)

- Press start button

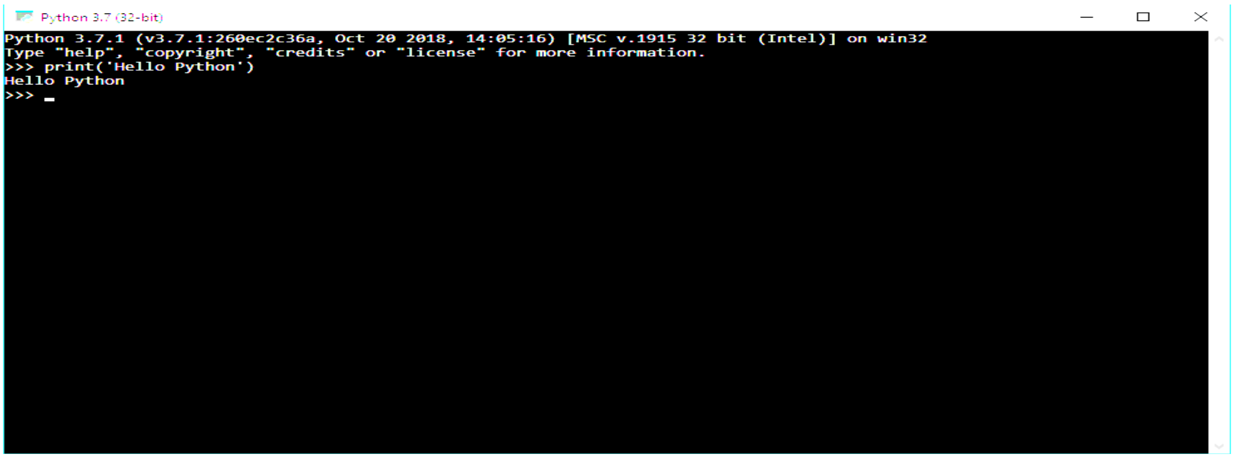


- Click on all programs and then click on Python 3.7 (32 bit). You will see the Python interactive prompt in Python command line.



```
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:05:16) [MSC v.1915 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- Python command prompt contains an opening message >>> called command prompt. The cursor at command prompt waits for to enter Python command. A complete command is called a statement. For example check first command to print message.

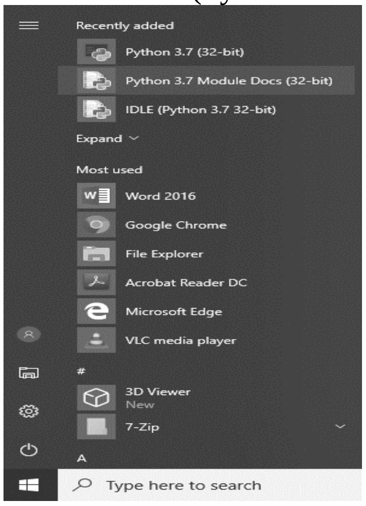


```
Python 3.7 (32-bit)
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:05:16) [MSC v.1915 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print('Hello Python')
Hello Python
>>> _
```

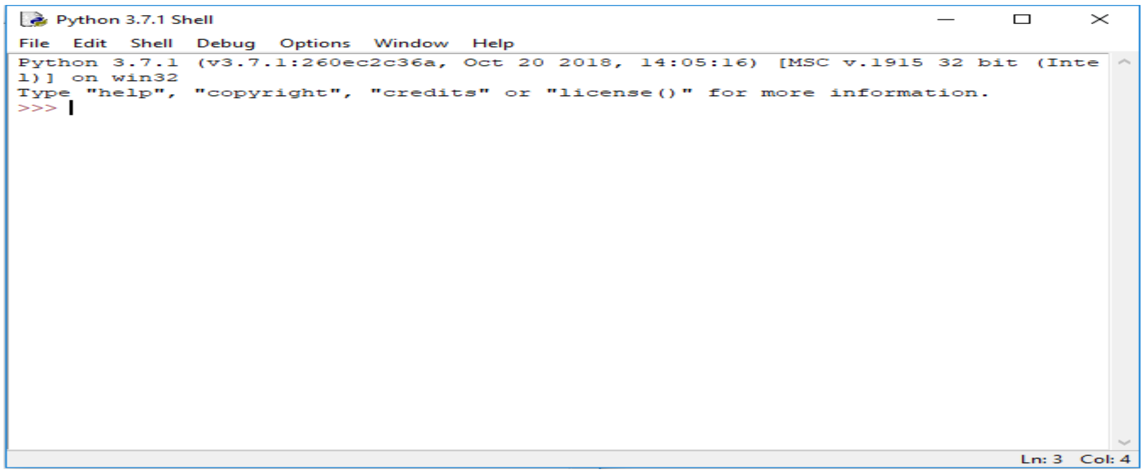
- To exit from the command line of Python, press Ctrl+z followed by Enter or Enter exit() or quit() and Enter.

2) Starting Python IDLE

- Press start button and click on IDLE (Python 3.7 32 bit) options.

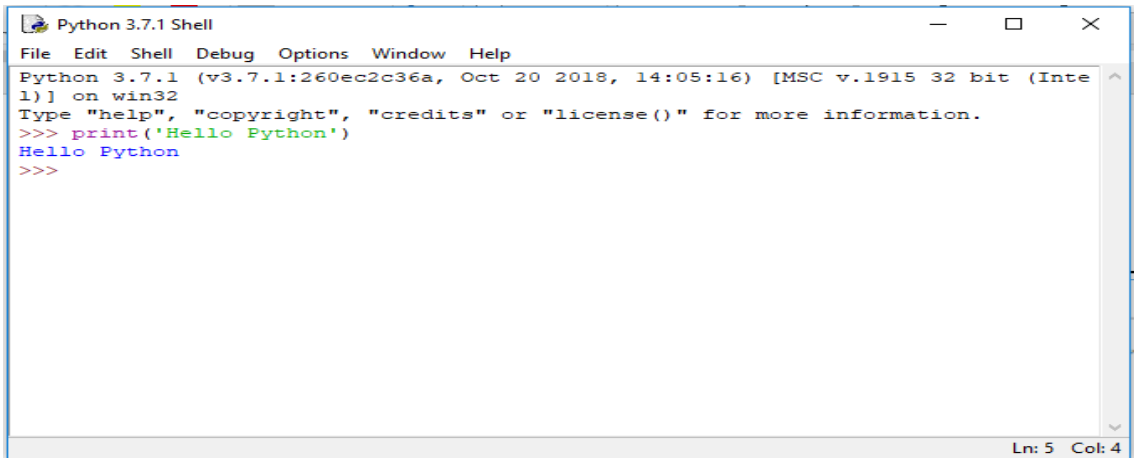


- You will see the Python interactive prompt i.e. interactive shell.



```
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:05:16) [MSC v.1915 32 bit (Inte
l)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 
```

- Python interactive shell prompt contains opening message >>>, called shell prompt. A cursor is waiting for the command. A complete command is called a statement. When you write a command and press enter, the Python interpreter will immediately display the result.



```
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:05:16) [MSC v.1915 32 bit (Inte
l)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print('Hello Python')
Hello Python
>>>
```

Practical No.2 : To study and execute python program for Tuple and Dictionary

THEORY-

An object may be classified into two categories

1) Mutable
2) Immutable

Mutable are the objects whose value can be altered after assigning a particular value.
Immutable are those objects whose value can not be altered after assigning a value.
Examples of mutable data type are list and dictionary
Example of immutable data type is tuple

LIST

1) It is a collection of data in which data is stored in ordered form
2) It is mutable
3) The square bracket is used to represent list .The item inside the list are separated by comma.
4) It supports indexing and slicing
5) Eg. a=[10, 22.75, “Sanskrit”, 4+5j, true]

TUPLE

1) It is a collection of data in which data is stored in ordered form
2) It is immutable
3) The normal bracket ( ) is used to represent list .The item inside the list are separated by
comma.
4) It supports indexing and slicing
5) Eg. a=(10, 22.75, “Sanskrit”, 4+5j, true)

□

DICTIONARY

1) It is a collection of data in which it is not in a sequential order like a list
2) It do not support indexing.
3) It consists of key and its value.
4) It is mutable
5) The bracket used for dictionary is curly bracket ‘{}’
6) Eg. dl ={“Name”:”Sanskrit”,”Age”:20,”roll no”:162}

PROGRAM –

# PROGRAM FOR LIST

```
a=[10,12.25,"Sanskrit",True, 4+5j, 10]
print(" List of elements in list are ",a)
print(type(a))
print(" Item at index 2 in list is ",a[2])
print(" Index 1 to 4 elements in list are ",a[1:5])

a.append("IT")
print(" New list with append elements is ",a)
a.insert(2,10)
print(" New list with inserted element is ",a)
a.remove("Sanskrit")
```

```
print(" New list with removed element is ",a)
c=a.count(10)
print(" No of 10 in list is ",c)
```

#PROGRAM FOR TUPLE

```
t=(10,12.25,"Sanskrit",True,4+5j)
print(" List of elements in tuple are ",t)
print(type(t))
print(" Item at index 2 in tuple is ",t[2])
print(" Index 1 to 4 elements in tuple are ",t[1:5])
x=t.index(True)
print("Index of True in tuple is = ",x)
y=t.count(10)
print(" No of 10 in list is ",y)
```

#PROGRAM FOR DICITIONARY

```
dl={"Name":"Anand","Age":20,"Roll no":121}
print("Keys available in dictionary d1 are ", dl.keys())
print("Values available in dictionary d1 are ",dl.values())
print("Items available in dictionary d1 are ",dl.items())
print("Dictionry items are ",dl)
print("Printing of value whose key is Name and value = ",dl["Name"])
dl.update({"branch":"IT","college":"PCE"})
print("Updated dictionary items are" ,dl)
del(dl["branch"])
print("Dictionary items after delection of branch key ",dl)
```

Testing Output

```
List of elements in list are [10, 12.25, 'Sanskrit', True, (4+5j), 10]
<class 'list'>
Item at index 2 in list is Sanskrit
Index 1 to 4 elements in list are [12.25, 'Sanskrit', True, (4+5j)]
New list with append elements is [10, 12.25, 'Sanskrit', True, (4+5j), 10, 'IT']
New list with inserted element is [10, 12.25, 10, 'Sanskrit', True, (4+5j), 10, 'IT']
New list with removed element is [10, 12.25, 10, True, (4+5j), 10, 'IT']
No of 10 in list is 3
List of elements in tuple are (10, 12.25, 'Sanskrit', True, (4+5j))
<class 'tuple'>
Item at index 2 in tuple is Sanskrit
Index 1 to 4 elements in tuple are (12.25, 'Sanskrit', True, (4+5j))

Index of True in tuple is = 3
No of 10 in list is 1
Keys available in dictionary d1 are dict_keys(['Name', 'Age', 'Roll no'])
Values available in dictionary d1 are dict_values(['Anand', 20, 121])
Items available in dictionary d1 are dict_items([('Name', 'Anand'), ('Age', 20), ('Roll
no', 121)])
Dictionry items are {'Name': 'Anand', 'Age': 20, 'Roll no': 121}
Printing of value whose key is Name and value = Anand
Updated dictionary items are {'Name': 'Anand', 'Age': 20, 'Roll no': 121, 'branch':
'IT', 'college':
'PCE'}
Dictionary items after delection of branch key {'Name': 'Anand', 'Age': 20, 'Roll no':
```

```
121,
'college': 'PCE'}
```

Result:
Thus the python program for Tuple and Dictionary was successfully executed.

Practical No.3: To study and execute python program for Function, Scoping and Abstraction

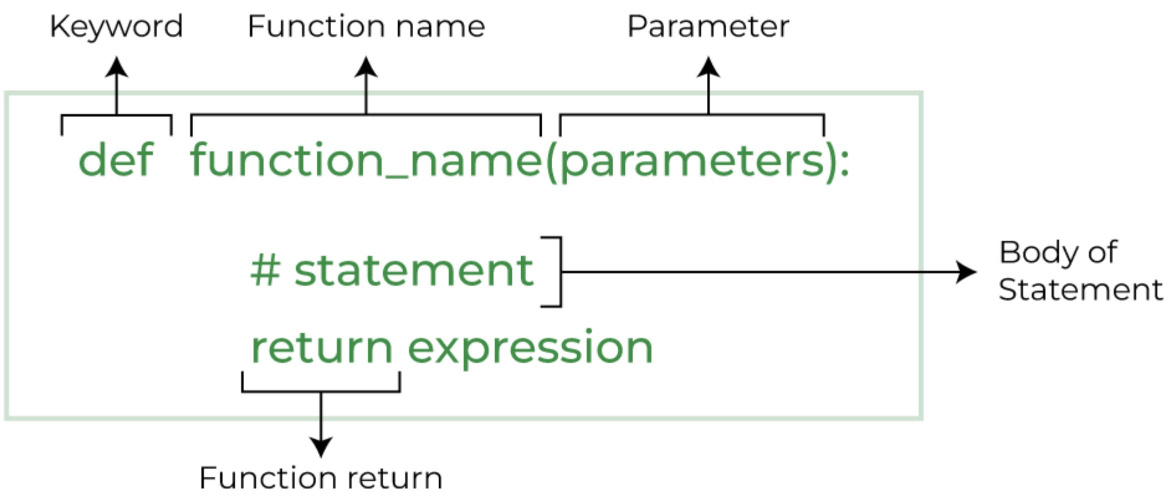
THEORY-

Function

Python Functions is a block of statements that return the specific task. The idea is to put some commonly or repeatedly done tasks together and make a function so that instead of writing the same code again and again for different inputs, we can do the function calls to reuse code contained in it over and over again.

Some Benefits of Using Functions
Increase Code Readability

Python Function Declaration
The syntax to declare a function is:



Types of Functions in Python

Below are the different types of functions in Python:

- Built-in library function: These are Standard functions in Python that are available to use.
- User-defined function: We can create our own functions based on our requirements.

Creating a Function in Python
We can define a function in Python, using the def keyword. We can add any type of functionalities and properties to it as we require.

```
# A simple Python function
def fun():
   print("Welcome to Python Programming")
```

Python Scope variable

The location where we can find a variable and also access it if required is called the **scope of a variable**.

Python Local variable

Local variables are those that are initialized within a function and are unique to that function. It cannot be accessed outside of the function. Let’s look at how to make a local variable.

Python Global variables

Global variables are the ones that are defined and declared outside any function and are not specified to any function. They can be used by any part of the program.

Abstraction in Python

Abstraction is used to hide the internal functionality of the function from the users. The users only interact with the basic implementation of the function, but inner working is hidden. User is familiar with that "what function does" but they don't know "how it does."

In simple words, we all use the smartphone and very much familiar with its functions such as camera, voice-recorder, call-dialing, etc., but we don't know how these operations are happening in the background. Let's take another example - When we use the TV remote to increase the volume. We don't know how pressing a key increases the volume of the TV. We only know to press the "+" button to increase the volume.

In Python, an abstraction is used to hide the irrelevant data/class in order to reduce the complexity. It also enhances the application efficiency.

Abstraction classes in Python

In Python, abstraction can be achieved by using abstract classes and interfaces.

A class that consists of one or more abstract method is called the abstract class. Abstract methods do not contain their implementation. Abstract class can be inherited by the subclass and abstract method gets its definition in the subclass. Abstraction classes are meant to be the blueprint of the other class. An abstract class can be useful when we are designing large functions. An abstract class is also helpful to provide the standard interface for different implementations of components. Python provides the abc module to use the abstraction in the Python program. Let's see the following syntax.

Syntax

```
        from abc import ABC

    class ClassName(ABC):
```

We import the ABC class from the **abc** module.

Interface in Python

Generally, the interface is not part of Python, but we can implement it using the ABC module.

An interface is a design template for creating classes that share common methods. The methods defined in an interface are abstract, meaning they are only outlined and lack implementation. It is unlike classes, which provide a concrete implementation for their methods. Instead, classes that implement an interface will fill in the abstract methods, giving them specific meaning and functionality.

By defining interfaces, developers can create a standard contract that classes can adhere to, regardless of their inheritance hierarchy. It allows for better code organization and reduces the likelihood of code duplication.

PROGRAM –
#Program to check whether the given number is Even or Odd using Function

```
class EvenOrOdd:
   def __init__(self,number):
      self.num = number

   def EvenOrOddd(self):
      if num % 2 == 0:
         print("Entered number is even.")
      else:
         print("Entered number is odd.")

num = int(input("Enter the number: "))
obj = EvenOrOdd(num)
obj.EvenOrOddd()
```

#Program to implement scoping in Python

```
glob = "I am globally available."

def function1():
   local = "I am locally available."
   print(local)
   print(glob)

function1()
print(glob)
```

#Program to implement Abstraction using Abstract class

```
from abc import ABC, abstractmethod

class Shape(ABC):
   @abstractmethod
   def calculate_area(self):
      pass

class Circle(Shape):
   def __init__(self, radius):
      self.radius = radius
   def calculate_area(self):
      return 3.14 * self.radius *self.radius

class Rectangle(Shape):
    def __init__(self, length, width):
        self.length = length
        self.width = width
    def calculate_area(self):
        return self.length * self.width

circle = Circle(5)
rectangle = Rectangle(4, 6)

print("Area of Circle: ", circle.calculate_area())
print("Area of Rectangle: ", rectangle.calculate_area())
```

OUTPUT:

#Output to check whether the given number is Even or Odd using Python

```
Enter the number: 5
Entered number is odd.
```

#Output of Program to implement scoping in Python

```
I am locally available.
I am globally available.
I am globally available.
```

#Output of Program to implement Abstraction using Abstract class

```
Area of Circle:  78.5
Area of Rectangle:  24
```

Result:

Thus the python program for Function, Scoping and Abstraction is successfully

Practical No.4: To study and execute python program for classes , objects, Inheritance and Polymorphism.

THEORY-

Python Classes/Objects

Python is an object oriented programming language.

Almost everything in Python is an object, with its properties and methods.

A Class is like an object constructor, or a "blueprint" for creating objects.

ALGORITHM-

```
Step 1- Create a class with name student
Step 2- Create a default method __init__ with three arguments
Step 3- Create a method with name avg for calculation of average marks
Step 4- Create two objects with name s1 and s2
Step 5- Calculate average marks of object s1 and s2 and display the result
```

PROGRAM –

```
class student:
def __init__(self,m1,m2,m3):
        self.m1=m1
        self.m2=m2
        self.m3=m3

def avg(self):
      return (self.m1+self.m2+self.m3)/3

s1=student(40,50,60)
s2=student(70,80,90)
avg1=s1.avg()
avg2=s2.avg()
print("Average marks of 1st student=", avg1)
print("Average marks of 1st student=", avg2)
```

Testing Output

```
Average marks of 1st student= 50.0
Average marks of 1st student= 80.0
```

Python Inheritance

Inheritance allows us to define a class that inherits all the methods and properties from another class.

Parent class is the class being inherited from, also called base class.

Child class is the class that inherits from another class, also called derived class.

ALGORITHM- for Inheritance

```
Step 1- Create a class with name parent
Step 2- Create a method with name spouse details
Step 3- Create a another method with name children details
Step 4- Create a another class child with inheritance facility
Step 5- Create a method with name qualification details
Step 6- Access parent class methods through child class
```

PROGRAM

```
class parent:
    def spouse(self):
        print(" My wife name is Manda")
    def children(self):
        print(" I have two child, one is boy and another is girl")

class child(parent):
    def qualification(self):
        print(" My first child is Engineer and second is doctor")

c1=child()
c1.spouse()
c1.children()
c1.qualification()
```

Testing Output

```
My wife name is Manda
I have two child, one is boy and another is girl
My first child is Engineer and second is doctor
```

Python Polymorphism

The word "polymorphism" means "many forms", and in programming it refers to methods/functions/operators with the same name that can be executed on many objects or classes.

Class Polymorphism

Polymorphism is often used in Class methods, where we can have multiple classes with the same method name. For example, say we have three classes: Car, Boat, and Plane, and they all have a method called move():

PROGRAM :

```
class Shape:
   def __init__(self,x,y):
      self.x = x
      self.y = y

   def area(self):
      return self.x * self.y

class Circle(Shape):
   def __init__(self, radius):
      self.radius = radius
      super().__init__(radius, radius)

   def area(self):
      return 3.14 * super().area()

c = Circle(5)
print("Area of Circle is ", c.area())
```

Output:

```
Area of Circle is  78.5
```