NAME: Meet Raut
DIV: S21
ROLL NO: 2201084

EXPERIMENT – 3:

- ***AIM:*** **To study and implement program on the concept of classes, functions and inheritance.**

- ***THEORY:***

➢ Classes in Python:

In Python, a class is a user-defined data type that contains both the data itself and the methods that may be used to manipulate it. In a sense, classes serve as a template to create objects. They provide the characteristics and operations that the objects will employ.
Suppose a class is a prototype of a building. A building contains all the details about the floor, rooms, doors, windows, etc. we can make as many buildings as we want, based on these details. Hence, the building can be seen as a class, and we can create as many objects of this class.

Creating Classes in Python:

In Python, a class can be created by using the keyword class, followed by the class name. The syntax to create a class is given below.

Syntax

```
class ClassName:
    #statement_suite
```

In Python, we must notice that each class is associated with a documentation string which can be accessed by using **<class-name>.__doc__.** A class contains a statement suite including fields, constructor, function, etc. definition.

➢ Objects in Python:

An object is a particular instance of a class with unique characteristics and functions. After a class has been established, you may make objects based on it. By using the class constructor, you may create an object of a class in Python. The object's attributes are initialised in the constructor, which is a special procedure with the name __init__.

Syntax:

```
# Declare an object of a class
object_name = Class_Name(arguments)
```

The self-parameter:

The self-parameter refers to the current instance of the class and accesses the class variables. We can use anything instead of self, but it must be the first parameter of any function which belongs to the class.

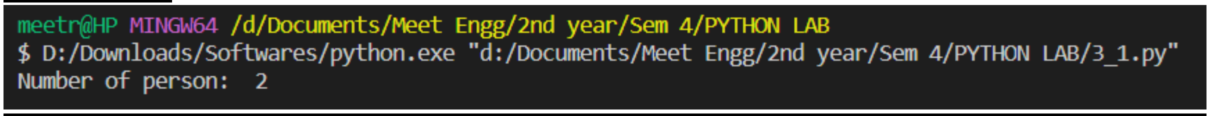
_ _init_ _ method:

In order to make an instance of a class in Python, a specific function called __init__ is called. Although it is used to set the object's attributes, it is often referred to as a constructor.
The self-argument is the only one required by the __init__ method. This argument refers to the newly generated instance of the class. To initialise the values of each attribute associated with the objects, you can declare extra arguments in the __init__ method.

- **PROGRAM 1:**

```
class Person:
    count = 0                    # This is a class variable

    def __init__(self, name, age):
        self.name = name        # This is an instance variable
        self.age = age
        Person.count += 1
person1 = Person("Aditya", 25)
person2 = Person("Prasad", 30)
print(Person.count)
```

- **OUTPUT:**

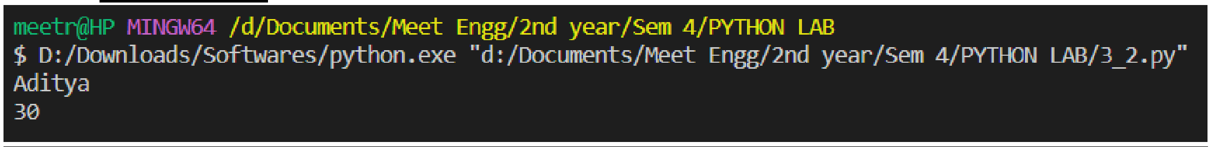


```
meetr@HP MINGW64 /d/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB
$ D:/Downloads/Softwares/python.exe "d:/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB/3_1.py"
Number of person:  2
```

- **PROGRAM 2:**

```
class Person:
    def __init__(self, name, age):
        self.name = name    # This is an instance variable
        self.age = age
person1 = Person("Aditya", 25)
person2 = Person("Bunty", 30)
print(person1.name)
print(person2.age)
```

- **OUTPUT:**



```
meetr@HP MINGW64 /d/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB
$ D:/Downloads/Softwares/python.exe "d:/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB/3_2.py"
Aditya
30
```

- **Python Functions:**

A collection of related assertions that carry out a mathematical, analytical, or evaluative operation is known as a function. An assortment of proclamations called Python Capabilities returns the specific errand. Python functions are necessary for intermediate-level programming and are easy to define. Function names meet the same standards as variable names do. The objective is to define a function and group-specific frequently performed actions. Instead of repeatedly creating the same code block for various input variables, we can call the function and reuse the code it contains with different variables.
Client-characterized and worked-in capabilities are the two primary classes of capabilities in Python. It aids in maintaining the program's uniqueness, conciseness, and structure.

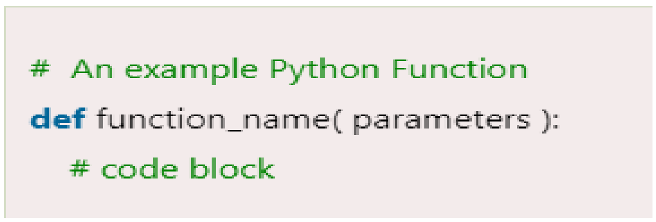
Advantages of Python Functions:

Pause We can stop a program from repeatedly using the same code block by including functions.

- Once defined, Python functions can be called multiple times and from any location in a program.
- Our Python program can be broken up into numerous, easy-to-follow functions if it is significant.
- The ability to return as many outputs as we want using a variety of arguments is one of Python's most significant achievements.
- However, Python programs have always incurred overhead when calling functions.

However, calling functions has always been overhead in a Python program.

Syntax



```
#  An example Python Function
def function_name( parameters ):
   # code block
```

The accompanying components make up to characterize a capability, as seen previously.

- The start of a capability header is shown by a catchphrase called def.
- function_name is the function's name, which we can use to distinguish it from other functions. We will utilize this name to call the capability later in the program. Name functions in Python must adhere to the same guidelines as naming variables.
- Using parameters, we provide the defined function with arguments. Notwithstanding, they are discretionary.
- A colon (:) marks the function header's end.
- We can utilize a documentation string called docstring in the short structure to make sense of the reason for the capability.
- Several valid Python statements make up the function's body. The entire code block's indentation depth-typically four spaces-must be the same.
- A return expression can get a value from a defined function.

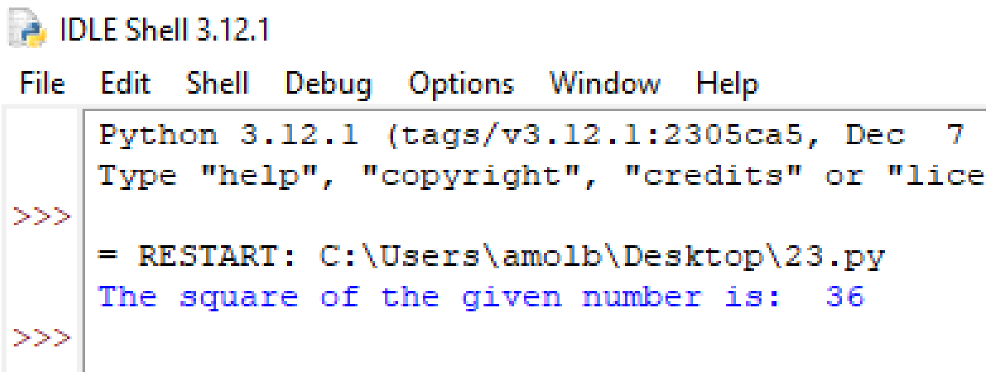
Calling a Function:

Calling a Function To define a function, use the def keyword to give it a name, specify the arguments it must receive, and organize the code block.
When the fundamental framework for a function is finished, we can call it from anywhere in the program.

- **PROGRAM 3:**

```
def square( num ):

    return num**2
object_ = square(6)
print( "The square of the given number is: ", object_ )
```

- **OUTPUT:**

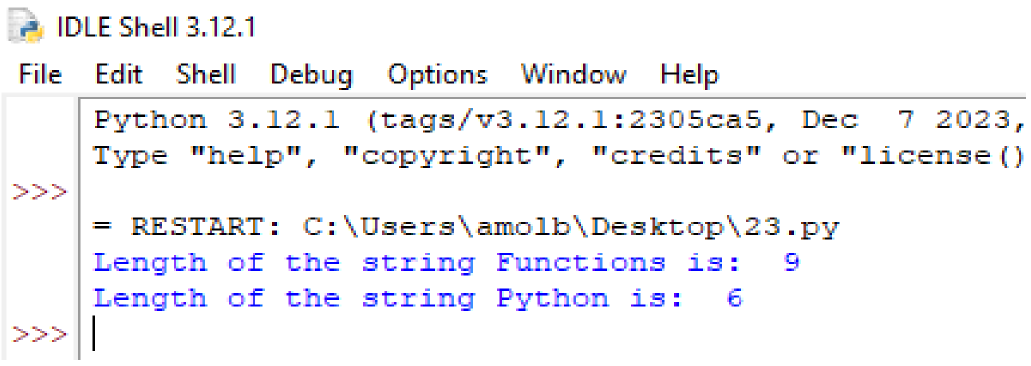


```
Python 3.12.1 (tags/v3.12.1:2305ca5, Dec  7
Type "help", "copyright", "credits" or "lice
>>>
= RESTART: C:\Users\amolb\Desktop\23.py
The square of the given number is:  36
>>>
```

- **PROGRAM 4:**

```
def a_function( string ):
    "This prints the value of length of string"
    return len(string)

# Calling the function we defined
print( "Length of the string Functions is: ", a_function( "Functions" ) )
print( "Length of the string Python is: ", a_function( "Python" ) )
```

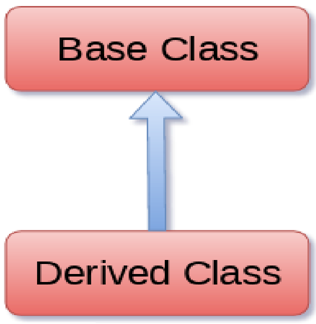
- **OUTPUT:**



```
Python 3.12.1 (tags/v3.12.1:2305ca5, Dec  7 2023,
Type "help", "copyright", "credits" or "license()
>>>
= RESTART: C:\Users\amolb\Desktop\23.py
Length of the string Functions is:  9
Length of the string Python is:  6
>>>
```

➢ **Python Inheritance:**

Inheritance is an important aspect of the object-oriented paradigm. Inheritance provides code reusability to the program because we can use an existing class to create a new class instead of creating it from scratch.
In inheritance, the child class acquires the properties and can access all the data members and functions defined in the parent class. A child class can also provide its specific implementation to the functions of the parent class. In this section of the tutorial, we will discuss inheritance in detail. In python, a derived class can inherit base class by just mentioning the base in the bracket after the derived class name. Consider the following syntax to inherit a base class into the derived class.



Syntax

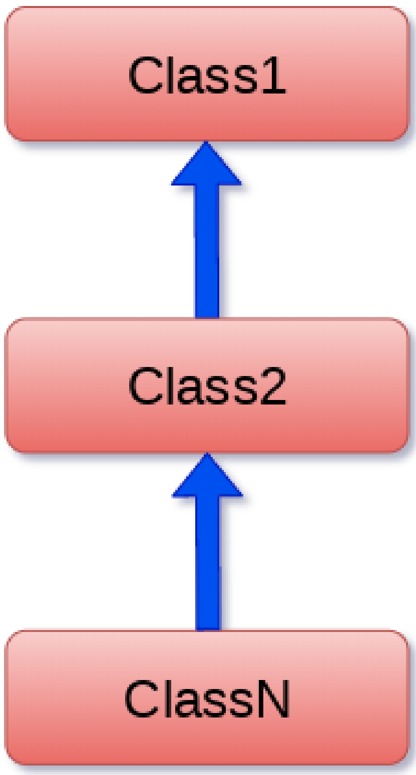
```
class derived-class(base class):
    <class-suite>
```

A class can inherit multiple classes by mentioning all of them inside the bracket. Consider the following syntax.

```
class derive-class(<base class 1>, <base class 2>, ..... <base class n>):
    <class - suite>
```

Python Multi-Level inheritance:

Multi-Level inheritance is possible in python like other object-oriented languages. Multi-level inheritance is archived when a derived class inherits another derived class. There is no limit on the number of levels up to which, the multi-level inheritance is archived in python.

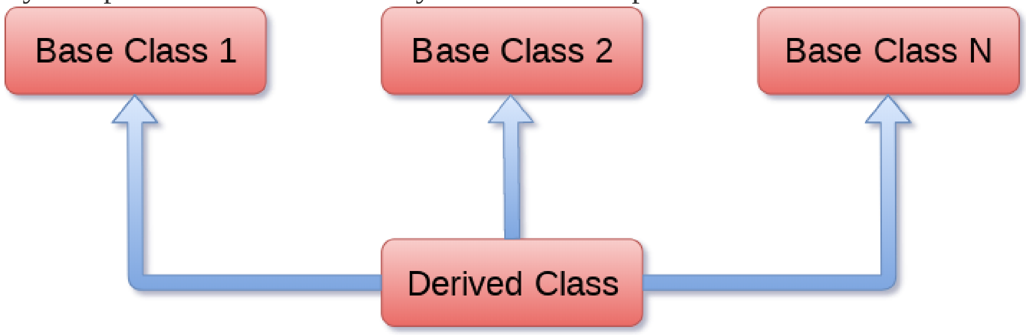


Syntax

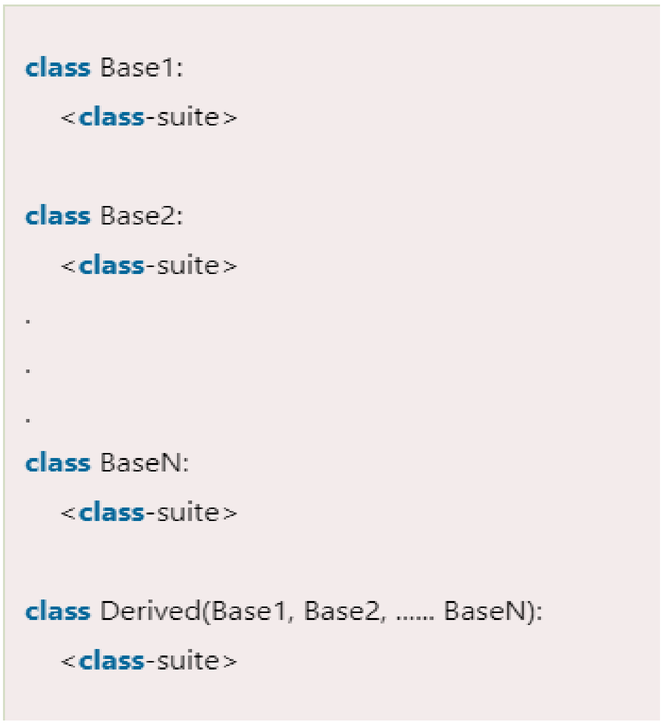
```
class class1:
    <class-suite>
class class2(class1):
    <class suite>
class class3(class2):
    <class suite>
.
```

Python Multiple inheritance:

Python provides us the flexibility to inherit multiple base classes in the child class.



Syntax

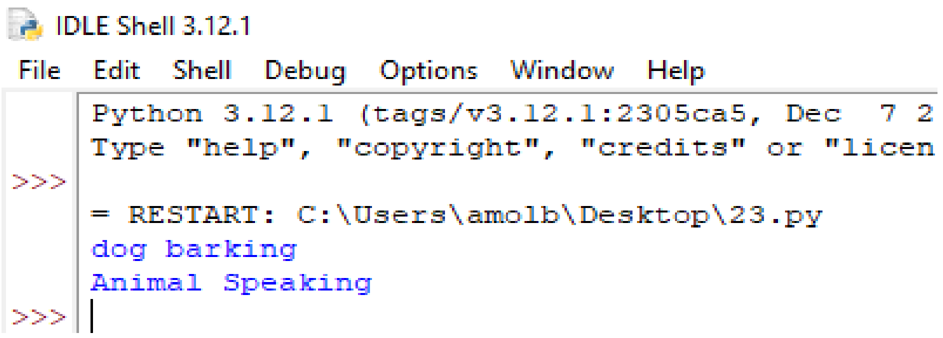


```
class Base1:
    <class-suite>

class Base2:
    <class-suite>
.
.
.
class BaseN:
    <class-suite>

class Derived(Base1, Base2, ...... BaseN):
    <class-suite>
```

➢ **PROGRAM 5:**

```
class Animal:
    def speak(self):
        print("Animal Speaking")
#child class Dog inherits the base class Animal
class Dog(Animal):
    def bark(self):
        print("dog barking")
d = Dog()
d.bark()
d.speak()
```

- **<u>OUTPUT:</u>**

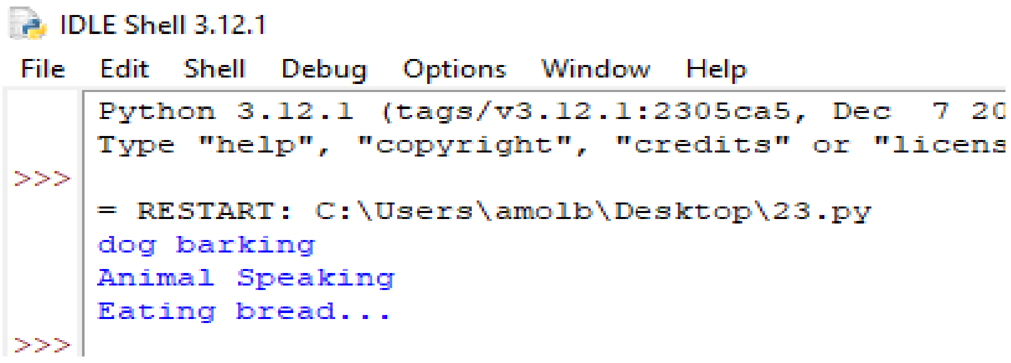


```
    Python 3.12.1 (tags/v3.12.1:2305ca5, Dec  7 2
    Type "help", "copyright", "credits" or "licen
>>>
    = RESTART: C:\Users\amolb\Desktop\23.py
    dog barking
    Animal Speaking
>>> |
```

- **PROGRAM 6:**

```
class Animal:
    def speak(self):
        print("Animal Speaking")
#The child class Dog inherits the base class Animal
class Dog(Animal):
    def bark(self):
        print("dog barking")
#The child class Dogchild inherits another child class Dog
class DogChild(Dog):
    def eat(self):
        print("Eating bread...")
d = DogChild()
d.bark()
d.speak()
d.eat()
```

- **<u>OUTPUT:</u>**

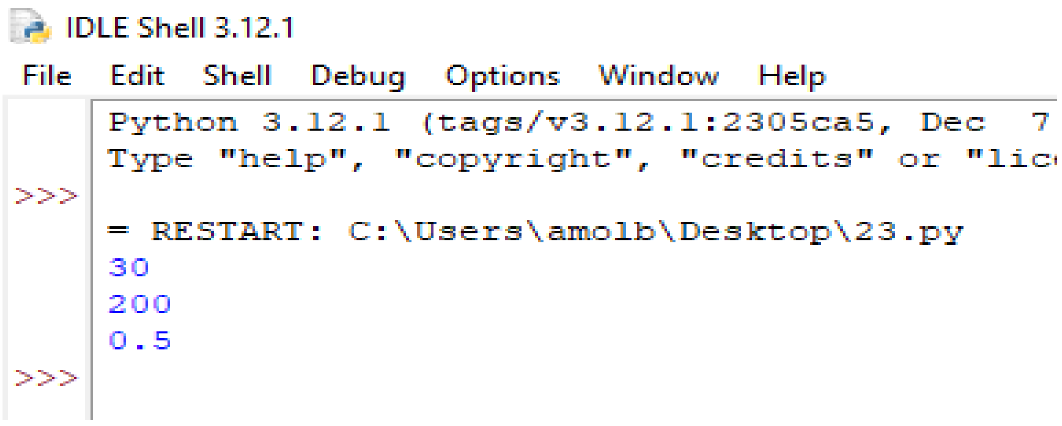


```
    Python 3.12.1 (tags/v3.12.1:2305ca5, Dec  7 20
    Type "help", "copyright", "credits" or "licens
>>>
    = RESTART: C:\Users\amolb\Desktop\23.py
    dog barking
    Animal Speaking
    Eating bread...
>>>
```

- **PROGRAM 7:**

```
class Calculation1:
    def Summation(self,a,b):
        return a+b;

class Calculation2:
    def Multiplication(self,a,b):
        return a*b;

class Derived(Calculation1,Calculation2):
    def Divide(self,a,b):
        return a/b;

d = Derived()
print(d.Summation(10,20))
print(d.Multiplication(10,20))
print(d.Divide(10,20))
```

- **<u>OUTPUT:</u>**

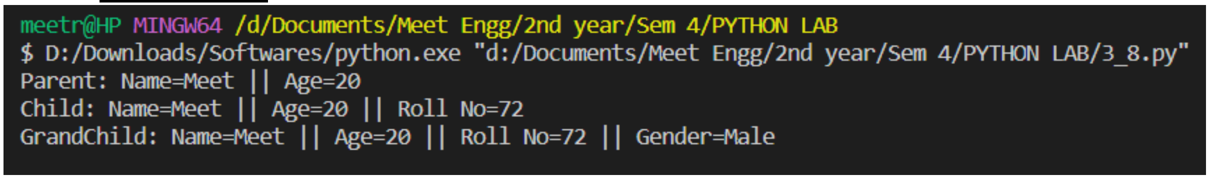


```
    Python 3.12.1 (tags/v3.12.1:2305ca5, Dec  7
    Type "help", "copyright", "credits" or "lic
>>>
    = RESTART: C:\Users\amolb\Desktop\23.py
    30
    200
    0.5
>>>
```

- **PROGRAM 8:**

```
#MULTILEVEL INHERITANCE

class A:
   def _init_ (self, Name, Age):
      self.Name=Name
      self.Age=Age
   def displayA (self):
      print("Parent: Name={0} || Age={1}".format(self.Name,self.Age))

class B(A):
   def _init_ (self, Name, Age, Roll):
      A._init_(self,Name,Age)
      self.Roll=Roll
   def displayB (self):
        print("Child: Name={0} || Age={1} || Roll No={2}".format(self.Name,self.Age,self.Roll))

class C(B):
    def _init_ (self, Name, Age, Roll, Gender):
        A._init_(self,Name,Age)
        B._init_(self,Name,Age,Roll)
        self.Gender=Gender
    def displayC (self):
        print("GrandChild: Name={0} || Age={1} || Roll No={2} || Gender={3}".format(self.Name,self.Age,self.Roll,self.Gender))

ob=C("Meet", 20, 72, "Male")
ob.displayA()
ob.displayB()
ob.displayC()
```

- **OUTPUT:**



```
meetr@HP MINGW64 /d/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB
$ D:/Downloads/Softwares/python.exe "d:/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB/3_8.py"
Parent: Name=Meet || Age=20
Child: Name=Meet || Age=20 || Roll No=72
GrandChild: Name=Meet || Age=20 || Roll No=72 || Gender=Male
```

- ***CONCLUSION:*** Hence, we have successfully implemented program on the concept of classes, functions and inheritance.