ASSIGNMENT-6

Q1. Explain Class and Object with respect to Object-Oriented Programming. Give a suitable example.

A-

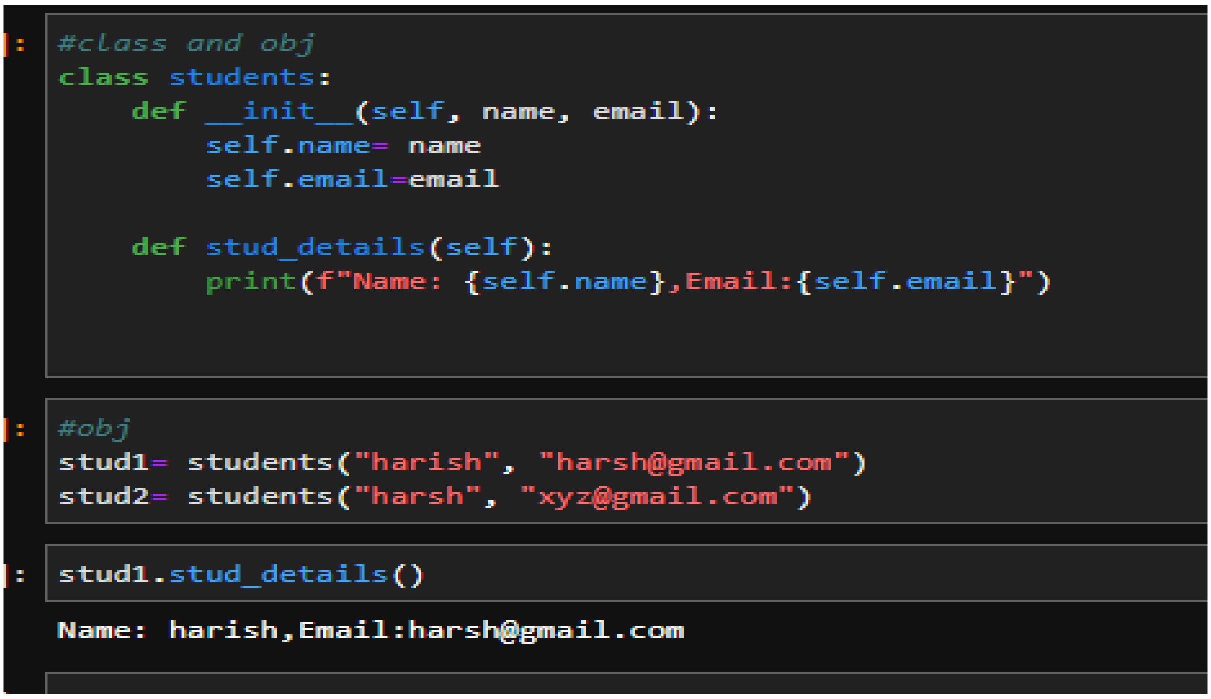
A class is a user-defined blueprint or prototype from which objects are created.

- **Class Definition:**

```
class ClassName:
   # Statement
```

- **Object Definition:**

```
obj = ClassName()
print(obj.atrr)
```



```
#class and obj
class students:
    def __init__(self, name, email):
        self.name= name
        self.email=email

    def stud_details(self):
        print(f"Name: {self.name},Email:{self.email}")
```

```
#obj
stud1= students("harish", "harsh@gmail.com")
stud2= students("harsh", "xyz@gmail.com")
```

```
stud1.stud_details()
```

```
Name: harish,Email:harsh@gmail.com
```

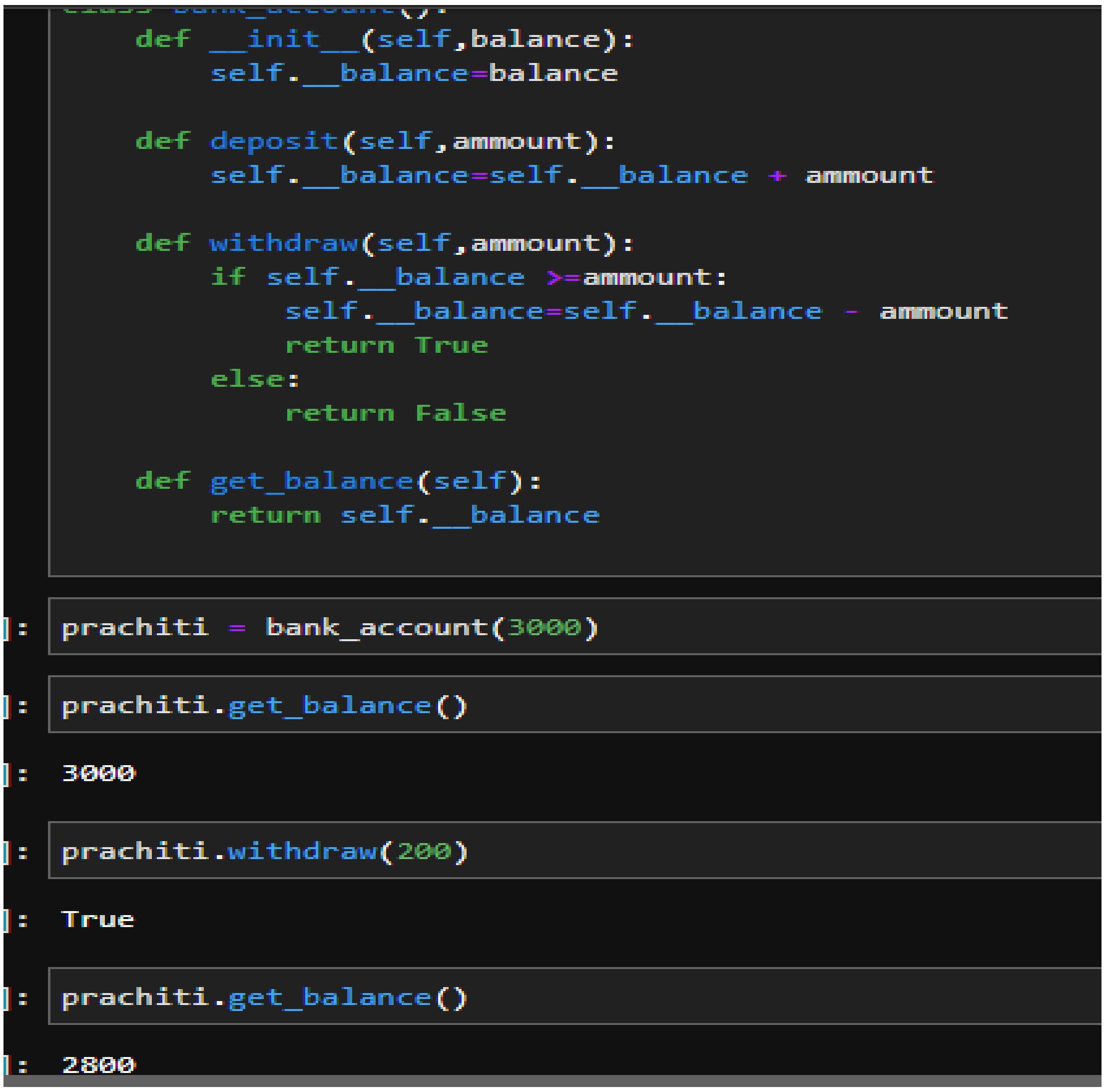
Q2. Name the four pillars of OOPs.

A-

Encapsulation

Encapsulation is the concept of bundling data (attributes) and methods (functions) that operate on that data into a single unit called a class.

It hides the internal details of how a class works from the outside.

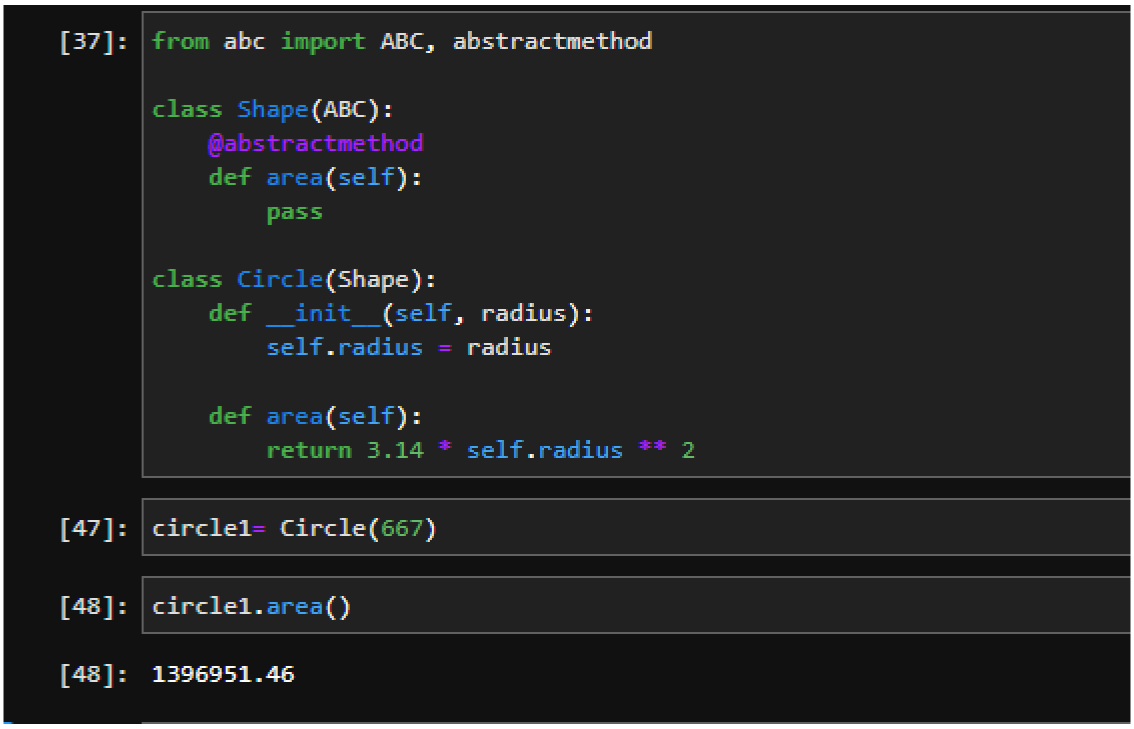


```
    def __init__(self,balance):
        self.__balance=balance

    def deposit(self,ammount):
        self.__balance=self.__balance + ammount

    def withdraw(self,ammount):
        if self.__balance >=ammount:
            self.__balance=self.__balance - ammount
            return True
        else:
            return False

    def get_balance(self):
        return self.__balance
```

```
]: prachiti = bank_account(3000)

]: prachiti.get_balance()

]: 3000

]: prachiti.withdraw(200)

]: True

]: prachiti.get_balance()

]: 2800
```

Abstraction:

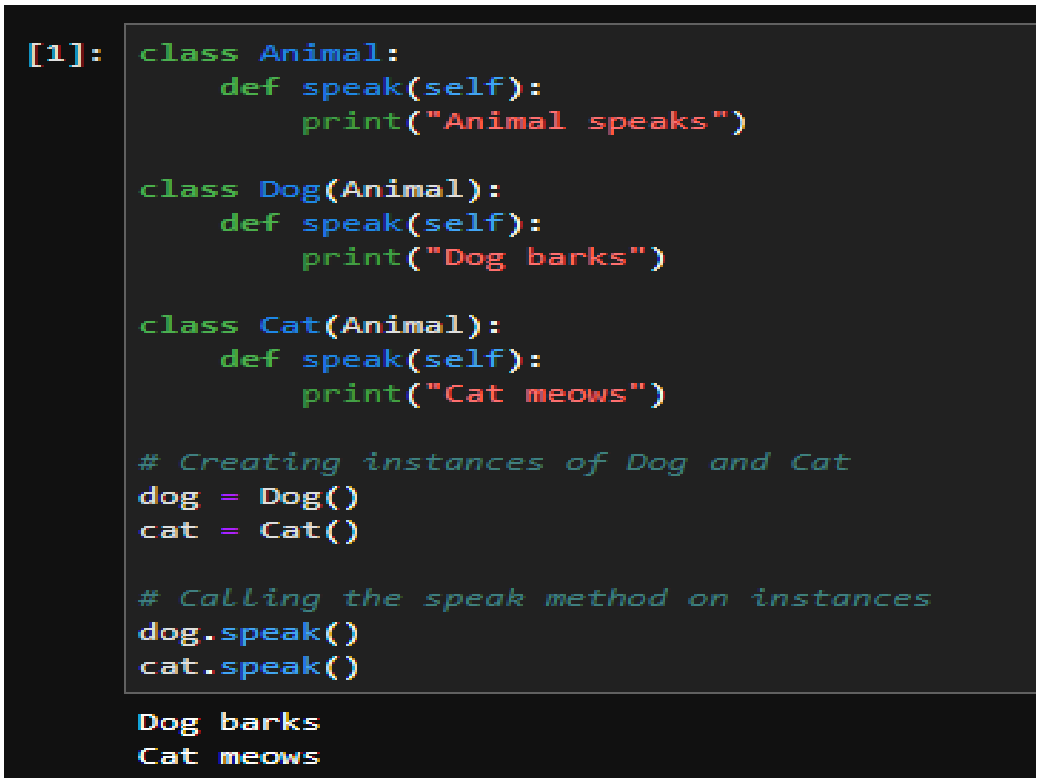
Abstraction allows you to simplify complex systems by modeling classes based on their essential characteristics and hiding unnecessary details.



```
[37]: from abc import ABC, abstractmethod

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

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

          def area(self):
              return 3.14 * self.radius ** 2

[47]: circle1= Circle(667)

[48]: circle1.area()

[48]: 1396951.46
```

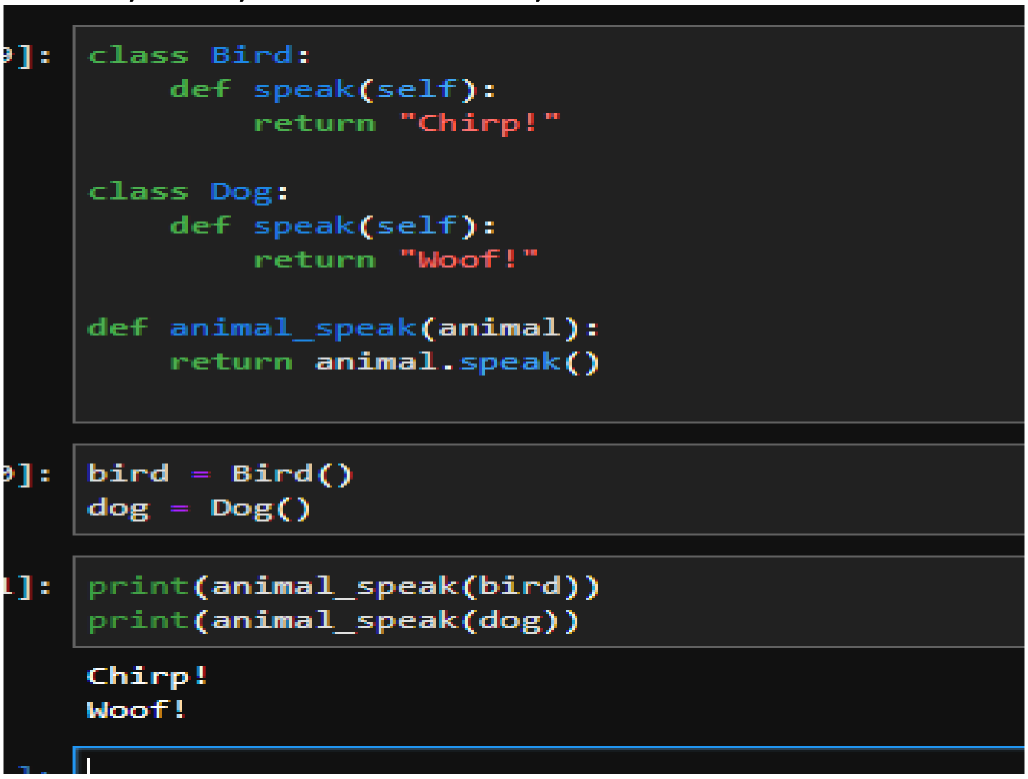
Inheritance:

Inheritance is a mechanism that allows you to create a new class by deriving properties and behaviors from an existing class.



```
[1]: class Animal:
         def speak(self):
             print("Animal speaks")

     class Dog(Animal):
         def speak(self):
             print("Dog barks")

     class Cat(Animal):
         def speak(self):
             print("Cat meows")

     # Creating instances of Dog and Cat
     dog = Dog()
     cat = Cat()

     # Calling the speak method on instances
     dog.speak()
     cat.speak()

     Dog barks
     Cat meows
```

Polymorphism:
Polymorphism means the ability of different objects to respond to the same method or function call in a way that is specific to their class. It allows for flexibility and dynamic behavior in your code.



```
class Bird:
    def speak(self):
        return "Chirp!"

class Dog:
    def speak(self):
        return "Woof!"

def animal_speak(animal):
    return animal.speak()
```

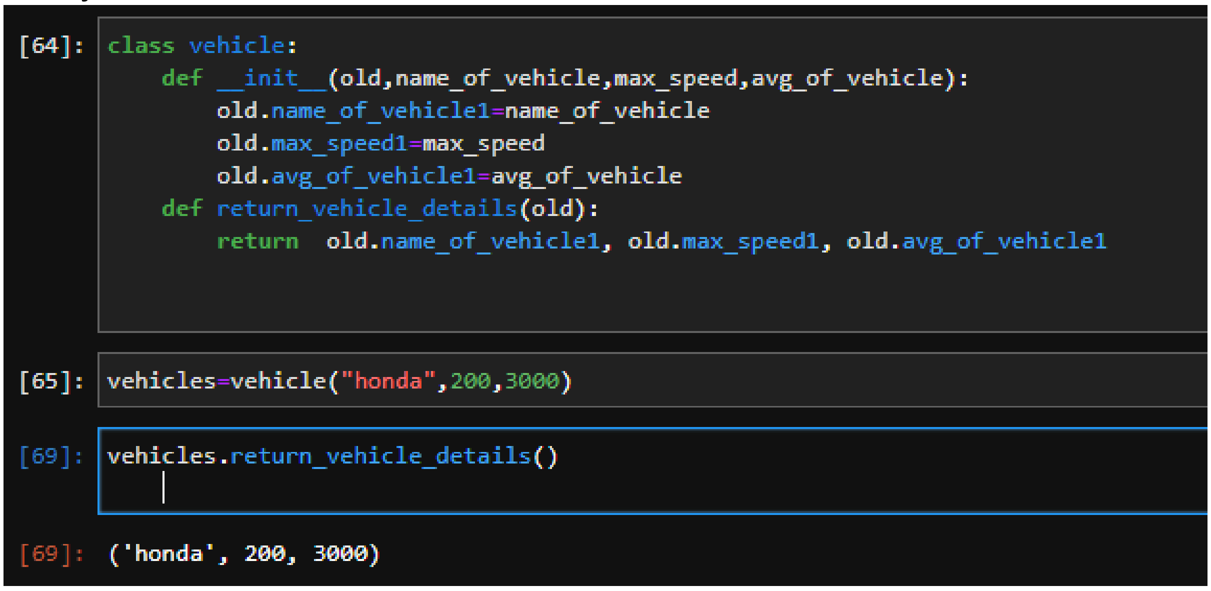
```
bird = Bird()
dog = Dog()
```

```
print(animal_speak(bird))
print(animal_speak(dog))
```

```
Chirp!
Woof!
```

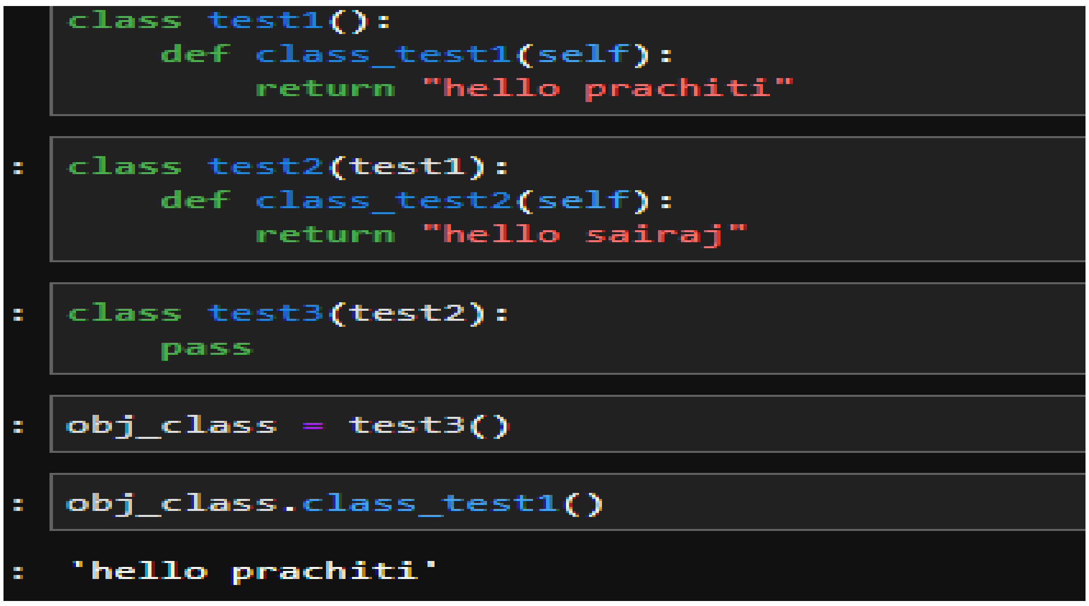
Q3. Explain why the __init__() function is used. Give a suitable example.

A: The __init__ function in Python is a special method (also known as a "magic" or "dunder" method) used for initializing objects when you create an instance of a class. It is one of the fundamental methods in object-oriented programming in Python and plays a crucial role in the construction and setup of objects.



```
[64]: class vehicle:
          def __init__(old,name_of_vehicle,max_speed,avg_of_vehicle):
              old.name_of_vehicle1=name_of_vehicle
              old.max_speed1=max_speed
              old.avg_of_vehicle1=avg_of_vehicle
          def return_vehicle_details(old):
              return  old.name_of_vehicle1, old.max_speed1, old.avg_of_vehicle1

[65]: vehicles=vehicle("honda",200,3000)

[69]: vehicles.return_vehicle_details()

[69]: ('honda', 200, 3000)
```

Q4. Why self is used in OOPs?
A:
In Python, the self keyword is used within the methods of a class to refer to the instance of the class itself. It is a convention and not a reserved keyword, but it is widely followed and understood by Python developers. When you define a class and its methods.

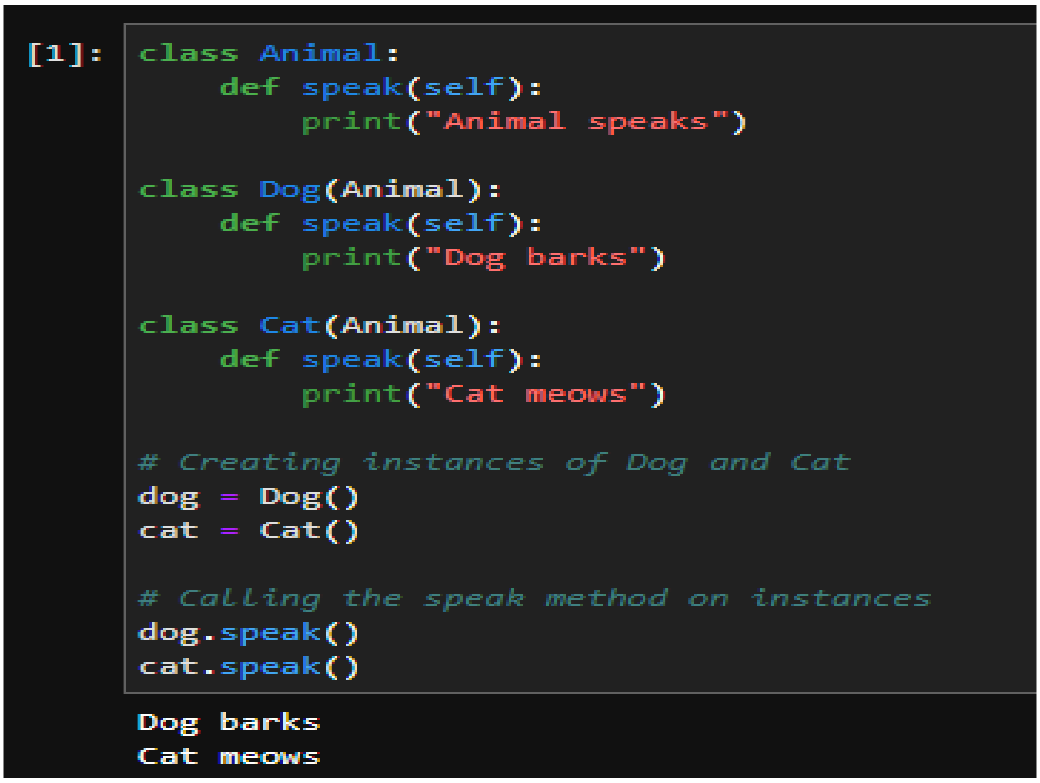


```
  class test1():
      def class_test1(self):
          return "hello prachiti"

: class test2(test1):
      def class_test2(self):
          return "hello sairaj"

: class test3(test2):
      pass

: obj_class = test3()

: obj_class.class_test1()

:  'hello prachiti'
```

Q5. What is inheritance? Give an example for each type of inheritance.

A:

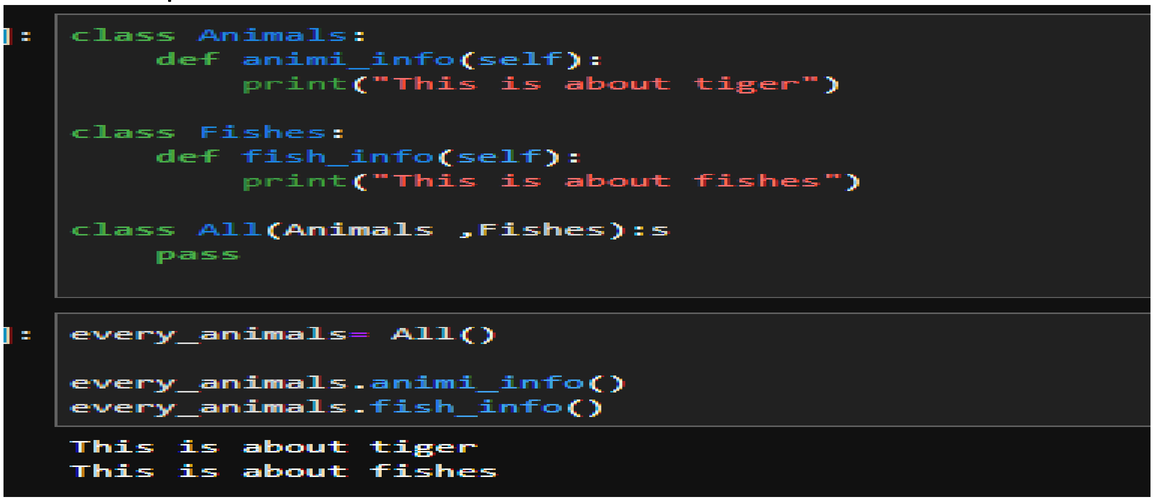
Inheritance is a mechanism that allows you to create a new class by deriving properties and behaviors from an existing class.



```
[1]: class Animal:
         def speak(self):
             print("Animal speaks")

     class Dog(Animal):
         def speak(self):
             print("Dog barks")

     class Cat(Animal):
         def speak(self):
             print("Cat meows")

     # Creating instances of Dog and Cat
     dog = Dog()
     cat = Cat()

     # Calling the speak method on instances
     dog.speak()
     cat.speak()

     Dog barks
     Cat meows
```

Multiple Inheritance:

Multiple inheritance is a feature in object-oriented programming that allows a class to inherit attributes and methods from more than one parent class. In Python, multiple inheritance is supported, which means a class can inherit from multiple base classes.



```
]: class Animals:
       def animi_info(self):
           print("This is about tiger")

   class Fishes:
       def fish_info(self):
           print("This is about fishes")

   class All(Animals ,Fishes):s
       pass

]: every_animals= All()

   every_animals.animi_info()
   every_animals.fish_info()

   This is about tiger
   This is about fishes
```