In [2]:
```
#Q1
'''
Python is an object oriented programming language.
In Python, almost everything is an object with properties and functions.
Class-A class functions as an object function Object() { [native code] } or an obje
We construct the house in accordance with these descriptions.
The item is a house.
Object- Object is like an instance of class. For example, suppose Bike is a class t
'''
class Dog:

    attr1 = "mamal"
    attr2 = "dog"


    def fun(self):
        print("I'm a", self.attr1)
        print("I'm a", self.attr2)
#Object instantiation
Rodger = Dog()

# Accessing class attributes
# and method through objects
print(Rodger.attr1)
Rodger.fun()
```

```
mamal
I'm a mamal
I'm a dog
```

In [ ]:
```
#Q2
'''
The four basic pillars of oops are:
1.Inheritance
2.Polymorphism
3.Encapsulation
4.Abstraction
'''
```

In [4]:
```
#Q3
'''
In Python classes, "__init__" is a reserved method. A function Object() { [native c
When an object is made from a class, this method is invoked, enabling the class to
'''
# A Sample class with init method
class Person:

        # init method or constructor
        def __init__(self, name):
                self.name = name

        # Sample Method
        def say_hi(self):
                print('Hello, my name is', self.name)
```

```


p = Person('Amit')
p.say_hi()
```

```
Hello, my name is Amit
```

In [5]:
```
#Q4
'''self is a representation of the class instance. In Python, we can access the cla
It binds the given arguments and the attributes.
Python does not refer to instance attributes using the @ syntax, which is why you m
'''
```

Out[5]:
```
'self is a representation of the class instance. In Python, we can access the clas
s\'s attributes and methods by using the "self" keyword. \nIt binds the given argu
ments and the attributes. \nPython does not refer to instance attributes using the
@ syntax, which is why you must use self.\n'
```

In [6]:
```
#Q5
'''
1.Inheritance is referred to as 'IS A' relationship, implying that one class should
2.There are 5 types of inheritances
        Single Inheritance
        Multiple Inheritance
        Multilevel Inheritance
        Hierarchical Inheritance
        Hybrid Inheritance
'''
#Single Inheritance
class A:
    def display(self):
        print("Hello")

class B(A):
    def display(self):
        super().display()#By inheriting class A in class B, we can access the prope


        print("World")

b = B()
b.display()
```

```
Hello
World
```

In [7]:
```
#Multiple Inheritance
print("Multiple Inheritance Example")
class A:
    def sayHi(self):
        print("Hi")

class B:
    def sayBye(self):
        print("Bye")


```

```
class C(A, B):
    def display(self):
        super().sayHi()
        super().sayBye()

c = C()
c.display()
```

```
Multiple Inheritance Example
Hi
Bye
```

In [8]:

```
#Multilevel Inheritance
print("Multilevel Inheritance Example")
class A:
    def display(self):
        print("Class A")

class B(A):
    def display(self):
        super().display()
        print("Class B")

class C(B):
    def display(self):
        super().display()
        print("Class C")


c = C()
c.display()
```

```
Multilevel Inheritance Example
Class A
Class B
Class C
```

In [9]:

```
#Hierarchical Inheritance
class A:
    def display(self, output):
        print(output)

class B(A):
    def display(self):
        super().display('Hello from B')

class C(A):
    def display(self):
        super().display('Hello from C')

b = B()
b.display()

c = C()
c.display()
```

```
Hello from B
Hello from C
```

In [10]:

```python
#Hybrid Inheritance
class A:
    def display(self):
        print("Super Parent display method")


""" class B used as intermediate class
to call class A's display method """
class B(A):
    def display(self):
        super().display()

''' child classes '''
class C(B):
    def display(self):
        super().display()
        print("Class C display method")

class D(B):
    def display(self):
        super().display()
        print("Class D display method")

c = C()
c.display()

d = D()
d.display()
```

```
Super Parent display method
Class C display method
Super Parent display method
Class D display method
```

In [ ]: