OOP(Class, Object, Members)

Object Oriented Programming:

- Python is a multi-paradigm programming language. It supports different programming approaches.
- This concept focuses on creating reusable code and also known as DRY (Don't Repeat Yourself).
- One of the popular approaches to solve a programming problem is by creating objects. This is known as Object-Oriented Programming (OOP).
- An object has two characteristics:
  - attributes
  - behavior
- One example is: Parrot, He is having **color, age, name** as **attributes**, and **calling out names, dancing** as **behavior.**

Class:

- A class is a blueprint for the object.
- It contains all the details about the name, colors, size etc, it can be created as:

```
class Parrot:
        Pass:
```

- From class, we construct instances. An instance is a specific object created from a particular class.

Object:

- An object (instance) is an instantiation of a class. It can be product (for example iphone is an object, if we have the blueprint (class) how we can develop an iphone then we can create multiple of it).
- When class is defined, only the description for the object is defined. Therefore, no memory or storage is allocated.

```
obj=Parrot()
```

How to create a class and access object:

```
class Parrot:

  # class attribute
  species = "bird"

  # instance attribute
  def __init__(self, name, age):
    self.name = name
    self.age = age

# instantiate the Parrot class
blu = Parrot("Blu", 10)
woo = Parrot("Woo", 15)

# access the class attributes
print("Blu is a {}".format(blu.__class__.species))
print("Woo is also a {}".format(woo.__class__.species))

# access the instance attributes
print("{} is {} years old".format( blu.name, blu.age))
print("{} is {} years old".format( woo.name, woo.age))
```

Output:

```
Blu is a bird
Woo is also a bird
Blu is 10 years old
Woo is 15 years old
```

- In the above program a class has been created with name "Parrot", then attributes has been defined.
- These attributes are defined inside the `__init__` method of the class, It is the initializer method that is first run as soon as the object is created.
- Then, we create instances of the *Parrot* class. Here, *blu* and *woo* are references (value) to our new objects.
- We can access the class attribute using `__class__.species`. Class attributes are the same for all instances of a class. Similarly, we access the instance attributes using `blu.name` and `blu.age`.

Method:

Methods are functions defined inside the body of a class. They are used to define the behaviors of an object.

```
class Parrot:

    # instance attributes
    def __init__(self, name, age):
        self.name = name
        self.age = age

    # instance method
    def sing(self, song):
        return "{} sings {}".format(self.name, song)

    def dance(self):
        return "{} is now dancing".format(self.name)

# instantiate the object
blu = Parrot("Blu", 10)

# call our instance methods
print(blu.sing("'Happy'"))
print(blu.dance())
```

Output:

```
Blu sings 'Happy'
Blu is now dancing
```

Q1. Predict the output

```
class Dog:

    # class attribute
    attr1 = "mammal"

    # Instance attribute
    def __init__(self, name):
        self.name = name

# Driver code
# Object instantiation
Rodger = Dog("Rodger")
Tommy = Dog("Tommy")

# Accessing class attributes
print("Rodger is a {}".format(Rodger.__class__.attr1))
print("Tommy is also a {}".format(Tommy.__class__.attr1))

# Accessing instance attributes
print("My name is {}".format(Rodger.name))
print("My name is {}".format(Tommy.name))
```

Sol.

```
Rodger is a mammal
Tommy is also a mammal
My name is Rodger
My name is Tommy
```

Q2. Predict the output

```
class Dog:

    # class attribute
    attr1 = "mammal"

    # Instance attribute
    def __init__(self, name):
        self.name = name

    def speak(self):
        print("My name is {}".format(self.name))

# Driver code
# Object instantiation
Rodger = Dog("Rodger")
Tommy = Dog("Tommy")

# Accessing class methods
Rodger.speak()
Tommy.speak()
```

Sol.

```
     My name is Rodger
     My name is Tommy
```

Q3. Predict the output

```
class Person:
    def __init__(self, name):
        self.name = name
    def say_hi(self):
        print('Hello, my name is', self.name)
p = Person('John')
p.say_hi()

class Test:
    def fun(self):
        print("Hello")
obj = Test()
obj.fun()
```

Sol.

```
Hello, my name is John
Hello
```

Q4. Predict the output

```
class Person(object):

    # __init__ is known as the constructor
    def __init__(self, name, idnumber):
        self.name = name
        self.idnumber = idnumber

    def display(self):
        print(self.name)
        print(self.idnumber)

    def details(self):
        print("My name is {}".format(self.name))
        print("IdNumber: {}".format(self.idnumber))

# child class
class Employee(Person):
    def __init__(self, name, idnumber, salary, post):
        self.salary = salary
        self.post = post

        # invoking the __init__ of the parent class
        Person.__init__(self, name, idnumber)

    def details(self):
        print("My name is {}".format(self.name))
        print("IdNumber: {}".format(self.idnumber))
        print("Post: {}".format(self.post))


# creation of an object variable or an instance
a = Employee('Rahul', 886012, 200000, "Intern")

# calling a function of the class Person using
# its instance
a.display()
a.details()
```

Sol.

```
Rahul
886012
My name is Rahul
IdNumber: 886012
Post: Intern
```

Q5. Predict the output

```
class CSStudent:
    stream = 'cse'
    def __init__(self, roll):
        self.roll = roll
    def setAddress(self, address):
        self.address = address
    def getAddress(self):
        return self.address

a = CSStudent(101)
a.setAddress("BU, Greater Noida")
print(a.getAddress())
```

Sol.

```
BU, Greater Noida
```

Q6. Predict the output

```
class Bird:

    def intro(self):
        print("There are many types of birds.")

    def flight(self):
        print("Most of the birds can fly but some 
cannot.")

class sparrow(Bird):
    def flight(self):
        print("Sparrows can fly.")

class ostrich(Bird):

    def flight(self):
        print("Ostriches cannot fly.")

obj_bird = Bird()
obj_spr = sparrow()
obj_ost = ostrich()

obj_bird.intro()
obj_bird.flight()

obj_spr.intro()
obj_spr.flight()

obj_ost.intro()
obj_ost.flight()
```

Sol.

```
There are many types of birds.
Most of the birds can fly but some cannot.
There are many types of birds.
Sparrows can fly.
There are many types of birds.
Ostriches cannot fly.
```

Q7. Predict the output

```
class Base:
    def __init__(self):
        self.a = "BU"
        self.__c = "Greater Noida"

# Creating a derived class
class Derived(Base):
    def __init__(self):

        # Calling constructor of
        # Base class
        Base.__init__(self)
        print("Calling private member of base class: ")
        print(self.__c)


# Driver code
obj1 = Base()
print(obj1.a)
```

Sol.

```
  BU
```

Q8. Predict the output

```
class Base(object):
    pass   # Empty Class

class Derived(Base):
    pass

print(issubclass(Derived, Base))
print(issubclass(Base, Derived))

d = Derived()
b = Base()

# check is b is an instance of Derived ?
print(isinstance(b, Derived))

# check d is an instance of Base ?
print(isinstance(d, Base))
```

Sol.

```
     True
     False
     False
     True
```

Q9. Predict the output (e.g., multiple inheritances)

```
class Base1(object):
    def __init__(self):
        self.str1 = "Employee1"
        print ("Base1")

class Base2(object):
    def __init__(self):
        self.str2 = "Employee2"
        print ("Base2")

class Derived(Base1, Base2):
    def __init__(self):

        # Calling constructors of Base1
        # and Base2 classes
        Base1.__init__(self)
        Base2.__init__(self)
        print ("Derived")

    def printStrs(self):
        print(self.str1, self.str2)

ob = Derived()
ob.printStrs()
```

Sol.

```
Base1
Base2
Derived
Employee1 Employee2
```

Q10. Predict the output

```
class Base(object):

    def __init__(self, x):
        self.x = x

class Derived(Base):

    def __init__(self, x, y):
        Base.x = x
        self.y = y

    def printXY(self):

       print(Base.x, self.y)

d = Derived(200, 100)
d.printXY()
```

Sol.

```
    200 100
```

Q11. Predict the output

```
class Solution:
   def solve(self, nums):
      s = sorted(nums)
      count = 0
      for i in range(len(nums)):
         if s[i] == nums[i]:
            count += 1
      return count
ob = Solution()
print(ob.solve([2, 2, 4, 3, 11]))
```

Sol.

```
     3
```

Q12. Predict the output

```
class India():
    def capital(self):
        print("New Delhi is the capital of India.")

    def language(self):
        print("Hindi is the most widely spoken language")

    def type(self):
        print("India is a developing country.")

class USA():
    def capital(self):
        print("Washington, D.C. is the capital of USA.")

    def language(self):
        print("English is the primary language of USA.")

    def type(self):
        print("USA is a developed country.")

def func(obj):
    obj.capital()
    obj.language()
    obj.type()

obj_ind = India()
obj_usa = USA()

func(obj_ind)
func(obj_usa)
```

Sol.

```
New Delhi is the capital of India.
Hindi is the most widely spoken language
India is a developing country.
Washington, D.C. is the capital of USA.
English is the primary language of USA.
USA is a developed country.
```