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 (e.g., Create a class)

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```

Sol.

```
John
36
```

Q2. Predict the output

```
class IOString():
    def __init__(self):
        self.str1 = ""

    def get_String(self):
        self.str1 = input()

    def print_String(self):
        print(self.str1.upper())

str1 = IOString()
str1.get_String()
str1.print_String()
```

Sol.

Input: `king`

Output: `KING`

Q3. Predict the output

```
class py_solution:
    def rwords(self, s):
        return ' '.join(reversed(s.split()))

pt = py_solution()
print(pt.rwords('Program1 .py'))
```

Sol.

```
.py Program1
```

Q4. Predict the output (e.g., Object Method)

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)
p1.myfunc()
```

Sol.

```
Hello my name is John
```

Q5. Predict the output

```
class Person:
  def __init__(mysillyobject, name, age):
    mysillyobject.name = name
    mysillyobject.age = age

  def myfunc(abc):
    print("Hello my name is " + abc.name)

p1 = Person("John", 36)
p1.myfunc()
```

Sol.

```
Hello my name is John
```

Q6. 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
```

Q7. Predict the output

```
# parent class
class Bird:

    def __init__(self):
        print("Bird is ready")

    def whoisThis(self):
        print("Bird")

    def swim(self):
        print("Swim faster")

# child class
class Penguin(Bird):

    def __init__(self):
        # call super() function
        super().__init__()
        print("Penguin is ready")

    def whoisThis(self):
        print("Penguin")

    def run(self):
        print("Run faster")

peggy = Penguin()
peggy.whoisThis()
peggy.swim()
peggy.run()
```

Sol.

```
Bird is ready
Penguin is ready
Penguin
Swim faster
Run faster
```

Q8. Predict the output

```
class Computer:

    def __init__(self):
        self.__maxprice = 900

    def sell(self):
        print("Selling Price: {}".format(self.__maxprice))

    def setMaxPrice(self, price):
        self.__maxprice = price

c = Computer()
c.sell()

# change the price
c.__maxprice = 1000
c.sell()

# using setter function
c.setMaxPrice(1000)
c.sell()
```

Sol.

```
Selling Price: 900
Selling Price: 900
Selling Price: 1000
```

Q9. Predict the output

```
class Parrot:

    def fly(self):
        print("Parrot can fly")

    def swim(self):
        print("Parrot can't swim")

class Penguin:

    def fly(self):
        print("Penguin can't fly")

    def swim(self):
        print("Penguin can swim")

# common interface
def flying_test(bird):
    bird.fly()

#instantiate objects
blu = Parrot()
peggy = Penguin()

# passing the object
flying_test(blu)
flying_test(peggy)
```

Sol.

```
Parrot can fly
Penguin can't fly
```

Q10. Predict the output

```
class program1:
   def press(self, x, n):
        if x==0 or x==1 or n==1:
            return x
        if x==-1:
            if n%2 ==0:
                return 1
            else:
                return -1
        if n==0:
            return 1
        if n<0:
            return 1/self.press(x,-n)
        val = self.press(x,n//2)
        if n%2 ==0:
            return val*val
        return val*val*x

print(program1().press(4, 2))
```

Sol.

```
    16
```

Q11. 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
```

Q12. Predict the output

```
class Person(object):
    def __init__(self, name):
        self.name = name

    def getName(self):
        return self.name

    def isEmployee(self):
        return False

class Employee(Person):
    def __init__(self, name, eid):

        super(Employee, self).__init__(name)
        self.empID = eid

    def isEmployee(self):
        return True

    def getID(self):
        return self.empID

emp = Employee("Jonh", "Ep:1001")
print(emp.getName(), emp.isEmployee(), emp.getID())
```

Sol.

```
Jonh True Ep:1001
```