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:

1. attributes
2. 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
```

1. Predict the output:

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

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

    def out(self):
        print(self.str1.upper())
str1 = greet()
str1.inp()
str1.out()
```

Output:
hello
HELLO

2. Predict the Output:

```
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()
```

Output:
Hello my name is John

3. **Predict the Output:**

```
class Dog:
    animal = 'dog'
    def __init__(self, breed, color):
            self.breed = breed
            self.color = color
Rodger = Dog("Pug", "brown")
Buzo = Dog("Bulldog", "black")

print('Rodge is a {} with {}'.format(Rodger.breed, Rodger.color))
print('Buzo is a {} with {}'.format(Buzo.breed, Buzo.color))
```

Output:
Rodge is a Pug with brown
Buzo is a Bulldog with black

4. **Predict the Output:**

```
class MyClass:
    hiddenVariable = 10

myObject = MyClass()
print(myObject.hiddenVariable)

class MyClass:
    hiddenVariable = 0

    def add(self, increment):
        self.hiddenVariable += increment
        print (self.hiddenVariable)

myObject = MyClass()
myObject.add(2)
myObject.add(5)
```

Output:
10
2
7

5. **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))
```

Output:
True
False
False
True

6. **Predict the Output:**

```
class Rectangle():
    def __init__(self, l, w):
        self.length = l
        self.width  = w

    def rectangle_area(self):
        return self.length*self.width

newRectangle = Rectangle(12, 10)
print(newRectangle.rectangle_area())
```

Output:
120

Exercising with some inbuild functions for handling class attribute:

7. **Predict the Output:**

```
class abc:
    a=100
obj1= abc()
print("Initial Value is",getattr(obj1, 'a'))
setattr(obj1, 'a', 110)
print("Updated Value is", getattr(obj1, 'a'))
```

Output:
Initial Value is 100
Updated Value is 110

8. **Predict the Output:**

```
class abc:
    a=100
obj1= abc()
print("Initial Value is",getattr(obj1, 'a'))
delattr(obj1, 'a')
print("Updated Value is", getattr(obj1, 'a'))
```

Output:

```
Initial Value is 100
Traceback (most recent call last):
  File "C:/Users/RITI/OneDrive - BENNETT UNIVERSITY/Desktop/BENNETT University/Python Trial/oops.py", line 75, in <module>
    delattr(obj1, 'a')
AttributeError: a
```

9. Predict the output:

```
class ex:
    def __init__(self,arg):
        self.a=arg
    def __read(self):
        self.a=input("Enter new value of a")
    def disp(self):
        obj1:__
        print("Value of a=", self.a)
obj1=ex(300)
obj1.disp()
```

Output:

```
Value of a= 300
```

10. Predict the output:

```
class ex:
    def __init__(self,arg):
        self.a=arg
    def __repr__(self):
        return("The objective value is %s" %(self.a))
    def __cmp__(self, other):
        return(cmp(self.a,other.a))
obj1=ex(300)
obj2=ex(400)
print(obj1)
print(obj2)
if(obj1==obj2):
    print("Equal")
else:
    print("Not Equal")
```

Output:

```
The objective value is 300
The objective value is 400
Not Equal
```