MODULE-3

PPT-1

Python Classes/Objects

- Python is an object oriented programming language.
- Almost everything in Python is an object, with its properties and methods.
- A Class is like an object constructor, or a "blueprint" for creating objects.
- To create a class, use the keyword class:
- Create a class named MyClass, with a property named x:
- class MyClass:
- x = 5
- Now we can use the class named MyClass to create objects:
- Create an object named p1, and print the value of x:
- p1 = MyClass()
- print(p1.x)

The __init__() Function

- The __init__ method is similar to constructors in C++ and Java. Constructors are used to initialize the object's state.
- The task of constructors is to initialize(assign values) to the data members of the class when an object of class is created.
- Like methods, a constructor also contains collection of statements(i.e. instructions) that are executed at time of Object creation.
- It will run as soon as an object of a class is instantiated.
- The method is useful to do any initialization you want to do with your object.

- All classes have a function called __init__(), which is always executed when the class is being initiated.
- Use the __init__() function to assign values to object properties, or other operations that are necessary to do when the object is being created.
- **Note**: The __init__() function is called automatically every time the class is being used to create a new object.

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age
p1 = Person("John", 36)
print(p1.name)
print(p1.age)
```

```
class Complex:
    def __init__(self, realpart, imagpart):
        self.r = realpart
        self.i = imagpart
x = Complex(3.0, -4.5)
x.r, x.i


class Person:
        def __init__(self, name): # init method or constructor
                self.name = name
        def say_hi(self): # Sample Method
                print('Hello, my name is', self.name)
p = Person('Nikhil')
p.say_hi()
p.name()
```

There are two kinds of valid attribute names: data attributes and methods.

data attributes correspond to “instance variables” in Smalltalk, and to “data members” in C++. Data attributes need not be declared; like local variables, they spring into existence when they are first assigned to.

```
class Myclass:
    def __init__(self, realpart, imagpart):
        self.r = realpart
        self.i = imagpart
x = Myclass(1.0, -2.5) #x is the instance of MyClass
x.counter = 1
while x.counter < 10:
      x.counter = x.counter * 2
print(x.counter)
del x.counter
```

Object Methods

- The other kind of instance attribute reference is a **method**. A method is a function that “belongs to” an object. Objects can also contain methods. Methods in objects are functions that belong to the object.

```
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()
```

Note: The self parameter is a reference to the current instance of the class, and is used to access variables that belong to the class.

The self Parameter

- The self parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.
- It does not have to be named self , you can call it whatever you like, but it has to be the first parameter of any function in the class:

```
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()
```

- Modify Object Properties

```
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.age=40
print(p1.age)
del p1.age
print(p1.age)
```

You can delete properties on objects by using the del keyword:

The pass Statement

- Class definitions cannot be empty, but if you for some reason have a class definition with no content, put in the pass statement to avoid getting an error.

- class Person:
- pass

Class and Instance Variables-Output ?

```
class Dog:
    kind = 'canine'     # class variable shared by all instances
    def __init__(self, name):
        self.name = name # instance variable unique to each instance
d = Dog('Fido')
e = Dog('Buddy')
print(d.kind)                # shared by all dogs
print(e.kind)                # shared by all dogs
print(d.name)                  # unique to d
print(e.name)                  # unique to e
```

- Shared data can have possibly surprising effects with involving mutable objects such as lists and dictionaries. For example, the tricks list in the following code should not be used as a class variable because just a single list would be shared by all Dog instances:

```
class Dog:
    tricks = []             # mistaken use of a class variable
    def __init__(self, name):
        self.name = name
    def add_trick(self, trick):
        self.tricks.append(trick)
d = Dog('Fido')
e = Dog('Buddy')
d.add_trick('roll over')
e.add_trick('play dead')
print(d.tricks)  # unexpectedly shared by all dogs
print(e.tricks)              # unexpectedly shared by all dogs
```

Correct design of the class should use an instance variable instead:

```
class Dog:
    def __init__(self, name):
        self.name = name
        self.tricks = []    # creates a new empty list for each dog
    def add_trick(self, trick):
        self.tricks.append(trick)


d = Dog('Fido')
e = Dog('Buddy')
d.add_trick('roll over')
e.add_trick('play dead')
print(d.tricks)
print(e.tricks)
```

Output???

```
# Function defined outside the class
def f1(self, x, y):
    print(x,x+y)
    return min(x, x+y)

class C:
    f = f1
    def g(self):
        return 'hello world'
    h = g
x1=C()
print(x1.f(10,20))
print(x1.g())
```

Methods may call other methods by using method attributes of the self argument:

```
class Bag:
    def __init__(self):
        self.data = []
    def add(self, x):
        self.data.append(x)
    def addtwice(self, x):
        self.add(x)
        self.add(x)
x1=Bag()
x1.add("hello")
print(x1.data)
x1.addtwice("priyanka")
print(x1.data)
```

Practice question

Create a class called Complex for performing arithmetic with complex numbers. Write a driver program to test your class.

- Complex numbers have the form **realPart + imaginaryPart * i** where **i is** $\sqrt{-1}$
- Use floating-point numbers to represent the data of the class. Provide a constructor that enables an object of this class to be initialized when it is created. The constructor should contain default values in case no initializers are provided. Provide methods for each of the following:

a) Adding two **ComplexNumbers**: The real parts are added to form the real part of the result, and the imaginary parts are added to form the imaginary part of the result.

b) Subtracting two **ComplexNumbers**: The real part of the right operand is subtracted from the real part of the left operand to form the real part of the result, and the imaginary part of the right operand is subtracted from the imaginary part of the left operand to form the imaginary part of the result.

c) Printing **ComplexNumbers** in the form (a, b), where a is the real part and b is the imaginary part.

Practice question

Create a class **Rectangle**. The class has attributes **__length** and __width, each of which defaults to 1. It has methods that calculate the **perimeter** and the **area** of the rectangle. It has set and get methods for both **__length** and **__width.** The set methods should verify that **__length** and **__width** are each floating-point numbers larger than 0.0 and less than 20.0. Write a driver (main) program to test the class.

Note- Look into how to use get and set methods