Module-3

PPT-5

Method Overloading

- In Python, you can create a method that can be called in different ways. So, you can have a method that has zero, one or more number of parameters. Depending on the method definition, we can call it with zero, one or more arguments.
- Given a single method or function, the number of parameters can be specified by you. This process of calling the same method in different ways is called method overloading.

```
def product(a, b):
    p = a * b
    print(p)
def product(a, b, c):
    p = a * b*c
    print(p)

product(4,5, 5)
#product(4,5)#error
```

Output???

```
class Person:
 def Hello(self, name=None):
  if name is not None:
   print('Hello ' + name)
  else:
   print('Hello ')
obj = Person()
obj.Hello()
obj.Hello('Priyanka')
```

Output???

```
class Compute:
    def area(self, x = None, y = None):
        if x != None and y != None:
            return x * y
        elif x != None:
            return x * x
        else:
            return 0
obj = Compute()
print("Area Value:", obj.area())
print("Area Value:", obj.area(4))
print("Area Value:", obj.area(3, 5))
```

Polymorphism

- A child class inherits all the methods from the parent class. However, in some situations, the method inherited from the parent class doesn’t quite fit into the child class. In such cases, you will have to re-implement method in the child class.

Polymorphism with Function and Objects

```
class Tomato():
    def type(self):
      print("Vegetable")
    def color(self):
      print("Red")
class Apple():
    def type(self):
      print("Fruit")
    def color(self):
      print("Red")
```

```
def func(obj):
      obj.type()
      obj.color()
obj_tomato = Tomato()
obj_apple = Apple()
func(obj_tomato)
func(obj_apple)
```

Polymorphism with Class Methods

- Python uses two different class types in the same way. Here, you have to create a for loop that iterates through a tuple of objects. Next, you have to call the methods without being concerned about which class type each object is. We assume that these methods actually exist in each class.

```
class India():
    def capital(self):
      print("New Delhi")
    def language(self):
      print("Hindi and English")
class USA():
    def capital(self):
      print("Washington, D.C.")
    def language(self):
      print("English")
obj_ind = India()
obj_usa = USA()
for country in (obj_ind, obj_usa):
   country.capital()
country.language()
```

Polymorphism with Inheritance

- Polymorphism in python defines methods in the child class that have the same name as the methods in the parent class. In inheritance, the child class inherits the methods from the parent class. Also, it is possible to modify a method in a child class that it has inherited from the parent class.
- This is mostly used in cases where the method inherited from the parent class doesn’t fit the child class. This process of re-implementing a method in the child class is known as Method Overriding.

```
class Bird:
    def intro(self):
      print("There are different types of birds")
    def flight(self):
      print("Most of the birds can fly but some cannot")
class parrot(Bird):
    def flight(self):
      print("Parrots can fly")
class penguin(Bird):
    def flight(self):
      print("Penguins do not fly")
```

```
obj_bird = Bird()
obj_parr = parrot()
obj_peng = penguin()
obj_bird.intro()
obj_bird.flight()
obj_parr.intro()
obj_parr.flight()
obj_peng.intro()
obj_peng.flight()
```

Abstract Classes

- An abstract class can be considered as a blueprint for other classes. It allows you to create a set of methods that must be created within any child classes built from the abstract class.
- A class which contains one or more abstract methods is called an abstract class. An abstract method is a method that has a declaration but does not have an implementation.
- While we are designing large functional units we use an abstract class.
- When we want to provide a common interface for different implementations of a component, we use an abstract class.

Why use Abstract Base Classes :

- By defining an abstract base class, you can define a common Application Program Interface(API) for a set of subclasses.
- This capability is especially useful in situations where a third-party is going to provide implementations, such as with plugins, but can also help you when working in a large team or with a large code-base where keeping all classes in your mind is difficult or not possible.
- By default, Python does not provide abstract classes.
- Python comes with a module which provides the base for defining Abstract Base classes(ABC) and that module name is ABC.
- ABC works by decorating methods of the base class as abstract and then registering concrete classes as implementations of the abstract base.
- A method becomes abstract when decorated with the keyword @abstractmethod.

```
from abc import ABC, abstractmethod
class Polygon(ABC):
        def noofsides(self): # abstract method
                pass
class Triangle(Polygon):
        def noofsides(self): # overriding abstract method
                print("I have 3 sides")
class Pentagon(Polygon):
        def noofsides(self): # overriding abstract method
                print("I have 5 sides")
class Hexagon(Polygon):
        def noofsides(self): # overriding abstract method
                print("I have 6 sides")
class Quadrilateral(Polygon):
        def noofsides(self): # overriding abstract method
                print("I have 4 sides")
```

```
R = Triangle()
R.noofsides()
K = Quadrilateral()
K.noofsides()
R = Pentagon()
R.noofsides()
K = Hexagon()
K.noofsides()
```

```
from abc import ABC, abstractmethod
class Animal(ABC):
        def move(self):
                pass
class Human(Animal):
        def move(self):
                print("I can walk and run")
class Snake(Animal):
        def move(self):
                print("I can crawl")
class Dog(Animal):
        def move(self):
                print("I can bark")
class Lion(Animal):
        def move(self):
                print("I can roar")
```

```
R = Human()
R.move()
K = Snake()
K.move()
R = Dog()
R.move()
K = Lion()
K.move()
```