MODULE-3

PPT-3

Inheritance

- Inheritance allows us to define a class that inherits all the methods and properties from another class.

- Parent class is the class being inherited from, also called base class.

- Child class is the class that inherits from another class, also called derived class.

Create a Parent Class

Create a class named Person, with firstname and lastname properties, and a printname method:

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname
  def printname(self):
    print(self.firstname, self.lastname)
```

#Use the Person class to create an object, and then execute the printname method:

```
x = Person("John", "Doe")
x.printname()
```

Create a Child Class

- To create a class that inherits the functionality from another class, send the parent class as a parameter when creating the child class:
- Create a class named Student, which will inherit the properties and methods from the Person class:

```
class Student(Person):
  pass
```

Note: Use the pass keyword when you do not want to add any other properties or methods to the class.

- Now the Student class has the same properties and methods as the Person class.
- Use the Student class to create an object, and then execute the printname method:

```
x = Student("Mike", "Olsen")
x.printname()
```

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  pass

x = Student("Mike", "Olsen")
x.printname()
```

Add the __init__() Function

- So far we have created a child class that inherits the properties and methods from its parent.
- We want to add the __init__() function to the child class (instead of the pass keyword).
- Note: The __init__() function is called automatically every time the class is being used to create a new object.
- Add the __init__() function to the Student class:

```
class Student(Person):
  def __init__(self, fname, lname):
    #add properties etc.
```

- When you add the __init__() function, the child class will no longer inherit the parent's __init__() function.
- Note: The child's __init__() function overrides the inheritance of the parent's __init__() function.

- To keep the inheritance of the parent's __init__() function, add a call to the parent's __init__() function:

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname
  def printname(self):
    print(self.firstname, self.lastname)
class Student(Person):
  def __init__(self, fname, lname):
    Person.__init__(self, fname, lname)
x = Student("Mike", "Olsen")
x.printname()
```

Now we have successfully added the __init__() function, and kept the inheritance of the parent class, and we are ready to add functionality in the __init__() function.

Use the super() Function

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname
  def printname(self):
    print(self.firstname, self.lastname)
class Student1(Person):
  def __init__(self, fname, lname):
    super().__init__(fname, lname)
    print(self.firstname, self.lastname)
class Student2(Student1):
  def __init__(self, fname, lname):
    super().__init__(fname, lname)
x = Student1("Vivek", "Singh")
y = Student2("Priyanka", "Singh")
```

Python also has a super() function that will make the child class inherit all the methods and properties from its parent.

Output???

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname
  def printname(self):
    print(self.firstname, self.lastname)
class Student(Person):
  def __init__(self, fname, lname):
    super().__init__(fname, lname)
    self.graduationyear = 2020
x = Student("Priyanka", "Singh")
x.printname()
print(x.graduationyear)
```

Output??

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname
  def printname1(self):
    print(self.firstname, self.lastname)
class Student(Person):
  def __init__(self, fname, lname, year):
    super().__init__(fname, lname)
    self.graduationyear = year
  def printname2(self):
    print(self.firstname, self.lastname, self.graduationyear)
x = Student("Priyanka", "Singh", 2020)
x.printname2()
```

Multilevel Inheritance

- We can also inherit from a derived class. This is called multilevel inheritance. It can be of any depth in Python.
- In multilevel inheritance, features of the base class and the derived class are inherited into the new derived class.

```
class Base:
    pass
class Derived1(Base):
    pass
class Derived2(Derived1):
    pass
```

output??

```
class Base(object):
    def __init__(self, name):
        self.name = name
class Child(Base):
    def __init__(self, name, age):
        Base.__init__(self, name)
        self.age = age
class GrandChild(Child):
    def __init__(self, name, age, address):
        Child.__init__(self, name, age)
        self.address = address
    def getAddress(self):
        print( self.name, self.age, self.address)
x = GrandChild("Vivek", 23, "varanasi")
x.getAddress()
```

Multiple Inheritance

- Python supports a form of multiple inheritance as well. A class definition with multiple base classes looks like this:

- class DerivedClassName(Base1, Base2, Base3):
-     <statement-1>
-     .
-     .
-     .
-     <statement-N>

Output ???

```
class TeamMember(object): # Parent class 1
   def __init__(self, name, uid):
      self.name = name
      self.uid = uid
class Worker(object):          # Parent class 2
   def __init__(self, pay, jobtitle):
      self.pay = pay
      self.jobtitle = jobtitle
class TeamLeader(TeamMember, Worker):  # Deriving from the two parent classes
   def __init__(self, name, uid, pay, jobtitle, exp):
      self.exp = exp
      TeamMember.__init__(self, name, uid)
      Worker.__init__(self, pay, jobtitle)
      print("Name: {}, Pay: {}, Exp: {}".format(self.name, self.pay, self.exp))
TL = TeamLeader('Jake', 10001, 250000, 'Scrum Master', 5)
```

Output????

```
class Team:
    def show_Team(self):
        print("This is our Team:")
# Testing class inherited from Team
class Testing(Team):
    TestingName = ""
    def show_Testing(self):
        print(self.TestingName)
# Dev class inherited from Team
class Dev(Team):
    DevName = ""
    def show_Dev(self):
        print(self.DevName)
# Sprint class inherited from Testing and Dev classes
class Sprint(Testing, Dev):
    def show_parent(self):
        print("Testing :", self.TestingName)
        print("Dev :", self.DevName)
s1 = Sprint()  # Object of Sprint class
s1.TestingName = "James"
s1.DevName = "Barter"
s1.show_Team()
s1.show_parent()
```

Output????

```
class Company:
    def __init__(self, name, proj): # constructor
        self.name = name      # name(name of company) is public
        self._proj = proj     # proj(current project) is protected
    def show(self):
        print("The code of the company is = ",self.ccode)
class Emp(Company):# define child class Emp
    def __init__(self, eName, sal, cname, cproj):# constructor
        # calling parent class constructor
        Company.__init__(self, cname, cproj)
        self.n = eName   # public member variable
        self.__sal = sal    # private member variable
    def show_sal(self):
        print("The project of ",self.name," is ",self._proj)
        print("The salary of ",self.n," is ",self.__sal,)
c = Company("Stark Industries", "Mark 4")
e = Emp("Steve", 9999999, c.name, c._proj)
e.show_sal()
```