Inheritance

Inheritance

- **Inheritance** is the capability of one class to derive or inherit the properties from some another class.

- It provides **reusability** of a code. We don’t have to write the same code again and again. Also, it allows us to add more features to a class without modifying it.
- If class B inherits from another class A, then all the subclasses of B would automatically inherit from class A.
- **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.

Parent class and Child class

```
class Profile:
  def __init__(self, name, address):
        self.name = name
        self.address = address



class Hr(Profile):
      def info(self):
    print(self.name, self.address)

x = Hr(“Ram”, “Nepal”)
x.info()
```

Example:

```
class Profile:
  def __init__(self, name, address):
    self.name = name
    self.address = address

  def info(self):
    print(self.name, self.address)

class Student(Profile):
  def __init__(self, name, address, country):
    Profile.__init__(self, name, address)
    self.country = country

  def hello(self):
    print("Hello, welcome", self.name,"from", self.address, "to ", self.country)

x = Student("John", "USA", "Nepal")
x.hello()
```

Python Multi-Level inheritance

Python Multiple inheritance

Multi level

```
class first:

    def method_first(self):
        print("class:first and method:method_first")


class second(first):

    def method_second(self):
        print("class:second and method:method_second")


class third(second):
    pass
```

Multi-Level inheritance

```
class A:
    def methodA(self):
        print("This is class A")

class B(A):
    def methodB(self):
        print("This is class B")

class C(B):
    def methodC(self):
        print("This is class C")
z = C()
z.methodC()
z.methodB()
z.methodA()
```

```
class Person:
    #defining constructor
    def __init__(self, personName, personAge):
        self.name = personName
        self.age = personAge

    #defining class methods
    def showName(self):
        print(self.name)

    def showAge(self):
        print(self.age)


class Student: # Person is the
    def __init__(self, studentId):
        self.studentId = studentId

    def getId(self):
        return self.studentId


class Resident(Person, Student): # extends both Person and Student class
    def __init__(self, name, age, id):
        Person.__init__(self, name, age)
        Student.__init__(self, id)


# Create an object of the subclass
resident1 = Resident('Ram', 25, '2')
resident1.showName()
print(resident1.getId())
```

Example

```
class Add:
    def Sum(self,a,b):
        return a+b;
class Sub:
    def Mul(self,a,b):
        return a*b;
class Derived(Add,Sub):
    def Divide(self,a,b):
        return a/b;
d = Derived()
print(d.Sum(4,5))
print(d.Mul(5,2))
print(d.Divide(7,9))
```

Python - public, private and protected Access

Public members (generally methods declared in a class) are accessible from outside the class. The object of the same class is required to invoke a public method.

Protected members of a class are accessible from within the class and are also available to its sub-classes. A variable that is protected can only be accessed by its own class and any classes derived from it.

Private members of a class are denied access from the environment outside the class. They can be handled only from within the class.

Public Attributes

```
class employee:
    def __init__(self, name, sal):
        self.name=name
        self.salary=sal
------------------------------------------------------------------
class Office:
    # constructor
    def __init__(self, name, sal):
        self.name = name
        self.sal = sal
emp = Office(“John", 999000)
emp.sal
```

Protected Attributes

```
class employee:
    def __init__(self, name, sal):
        self._name=name  # protected attribute
        self._salary=sal # protected attribute
------------------------------------------------------------------------------------------------------------------
class Office:
    def __init__(self, name, sal):
        self._name = name   # protected attribute
        self._sal = sal     # protected attribute
emp = Office(“John", 10000)
 emp._sal
```

Private Attributes

```
class employee:
    def __init__(self, name, sal):
        self.__name=name  # private attribute
        self.__salary=sal # private attribute
-------------------------------------------------------------------------------
# defining class Employee
class Office:
    def __init__(self, name, sal):
        self.__name = name    # private attribute
        self.__sal = sal      # private attribute
emp = Office("Bill", 10000)
emp.__sal
```

Example…

```
class Car():
    def __init__(self, name = 'Ram', age = 30, year = '1971', add = 'USA',
     color = 'black'):
        self.__name = name
        self._age = age
        self.__year = year
        self.__add = add
        self._color = color

    def move_forward(self, name):
        print(f"Hello My name is {self.__name}. {name} I am from {self.__add}.")

    def move_backward(self, age):
        print(f"Hello My name is {self.__name}.I am {self._age}. {age}")

mycar = Car()
print(mycar._age)             # changing to mycar.move_forward(100)
mycar.move_forward("Ajaya")
mycar.move_backward("50")
```

Examples…

```
class Company:
    def __init__(self, name, proj):
        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):
    def __init__(self, eName, sal, cName, proj):

        Company.__init__(self, cName, proj)
        self.name = eName   # public member variable
        self.__sal = sal    # private member variable
    def show_sal(self):
        print("The review of  ",self.name," is ",self.__sal,)
c = Company("BroadWay", "Java")
e = Emp("Steve", 5, c.name, c._proj)
print("Welcome to ", c.name)
print("Here ", e.name," learning ",e._proj)
e.show_sal()
```