Inheritance

- OOP allows to define new classes from existing classes using the concept of inheritance.
- Inheritance is the capability of one class to derive or inherit the methods and properties of another class.
- **Parent/Base class/General class/Super class** - the class being inherited from
- **Child/Derived class/Specialized class/ Sub class** - the class that inherits from another class

Syntax:

```
class BaseClass:
   {Body}
class DerivedClass(BaseClass):
   {Body}
```

Creating a Parent Class

- Any class can be a Parent class, so the syntax is the same as creating any other class.

Example:

```
# Creating Parent class
class Parent:
   def func1(self):
      print("This function is in parent class.")
```

Creating a Child Class

- To create a Child class, give the Parent class_name as a parameter in the parantheses while declaring the child class.

Syntax:

class **child_class_name (base_class_name)**:

Example:

```
# Derived class
class Child(Parent):
   def func2(self):
      print("This function is in child class.")
```

Creating Object of Child Class

```
object = Child()
object.func1()
object.func2()
```

Example:

```
class Operations:
    a = 10
    b = 20
    def add(self):
        sum = self.a + self.b
        print(“Sum of a and b is: “, sum)

class MyClass(Operations):
    c = 50
    d = 10
    def sub(self):
        sub = self.c – self.d
        print(“Subtraction of c and d is: ”, sub)

ob = MyClass()
ob.add()
ob.sub()
```

Output:

```
Sum of a and b is: 30
Subtraction of c and d is: 40
```

Example:

#Creating a Person class with Display methods.

```
class Person:
  def __init__(self, name, id):
    self.name = name
    self.id = id

  def Display(self):
    print(self.name, self.id)

emp = Person("S", 102)
emp.Display()
```

#Creating a Child Class

Here Emp is another class which is going to inherit the properties of the Person class (base class).

```
class Emp(Person):
  def Print(self):
    print("Child class is called")

Emp_details = Emp("Mayank", 103)
Emp_details.Display()
Emp_details.Print()
```

Output:

```
Mayank 103
Emp class is called
```

Example:

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

   # To get name
   def getName(self):
      return self.name

   # To check if this person is an employee
   def isEmployee(self):
      return False

class Employee(Person):
   # Here we return true
   def isEmployee(self):
      return True

emp = Person("P1")
print(emp.getName(), emp.isEmployee())

emp = Employee("P2")
print(emp.getName(), emp.isEmployee())
```

Output:

```
P1 False
P2 True
```

Note:

When both Parent and Child contains constructor i.e. __init__()

- The child's __init__() function overrides the inheritance of the parent's __init__() function.
- To keep the inheritance of the parent's __init__() function, it is required to add a call to the parent's __init__() function using the Parent Class name
- **Syntax:**

  ```
  ParentClassName.__init__()
  ```

Example:

```
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):
     print(self.fname)
     Person.__init__(self, fname, lname)        #To call the constructor of Parent class

x = Student("Elon", "Musk")
x.printname()
```

Output:

```
Elon
Elon Musk
```

Example:

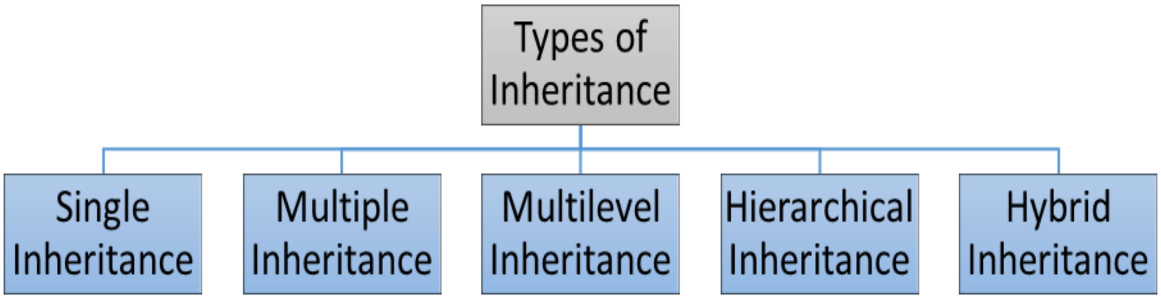
```
class Person:
    def __init__(self, name, idnumber):
        self.name = name
        self.idnumber = idnumber

    def display(self):
        print(self.name)
        print(self.idnumber)

# child class
class Employee(Person):
    def __init__(self, name, idnumber, salary, post):
        self.salary = salary
        self.post = post
        print(self.salary)
        print(self.post)

        # invoking the __init__ of the parent class
        Person.__init__(self, name, idnumber)

#a = Employee('Rahul', 886012) This will show error, two arguments missing.
a = Employee('Rahul', 886012, 200000, "Intern")
a.display()
```

Output:

```
200000
Intern
Rahul
886012
```

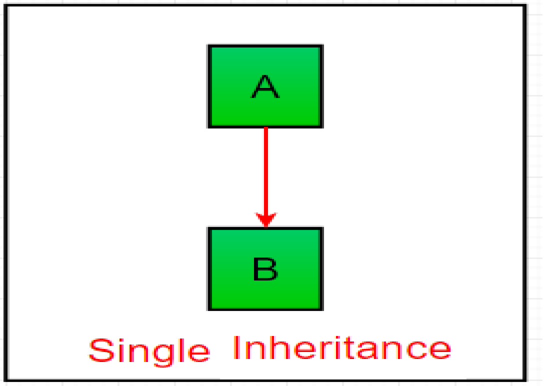
Types of Inheritance

Depending upon the number of child and parent classes involved inheritance id categorized into the following five types:



1) Single Inheritance

Single inheritance enables a derived class to inherit properties from a single parent class only.



Example:

```
# Base class
class Parent:
   def func1(self):
       print("This function is in parent class.")

# Derived class
 class Child(Parent):
   def func2(self):
       print("This function is in child class.")
object = Child()
object.func1()
object.func2()
```

Example: #Single Inheritance

```
class Operations:
    a = 10
    b = 20
    def add(self):
        sum = self.a + self.b
        print(“Sum of a and b is: “, sum)

class MyClass(Operations):
    c = 50
   d = 10
   def sub(self):
       sub = self.c – self.d
       print(“Subtraction of c and d is: ”, sub)

ob = MyClass()
ob.add()
ob.sub()
```

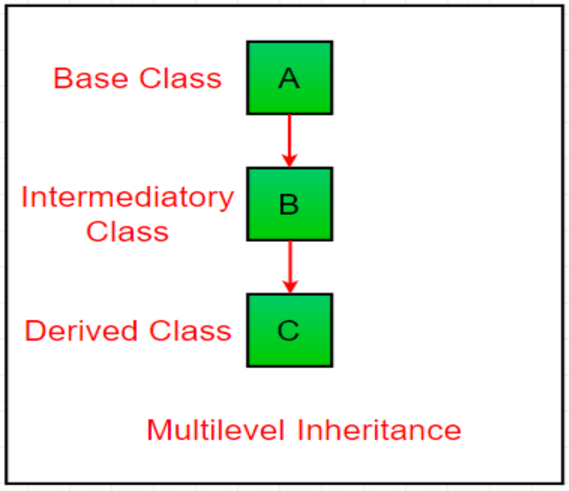
Output:

```
Sum of a and b is: 30
Subtraction of c and d is: 40
```

In the above example, we are inheriting the properties of the ‘Operations’ class into the class ‘MyClass’. Hence, we can access all the methods or statements present in the ‘Operations’ class by using the MyClass objects.

2) Multilevel Inheritance

- When a class can be derived from a class which is already derived from another base class than type of inheritance is called multiple inheritances.
- This means the second class will inherit the properties of the first class and the third class will inherit the properties of the second class. So, the second class will act as both the Parent class as well as Child class.



Example:

```
# Python program to demonstrate multilevel inheritance
class Addition:
    a = 10
    b = 20
    def add(self):
        sum = self.a + self.b
        print(“Sum of a and b is: ”, sum)

class Subtraction(Addition):
    def sub(self):
        sub = self.b-self.a
        print(“Subtraction of a and b is: ”, sub)

class Multiplication(Subtraction):
    def mul(self):
        multi = self.a * self.b
        print(“Multiplication of a and b is: ”, multi)

ob = Multiplication ()
ob.add()
ob.sub()
ob.mul()
```

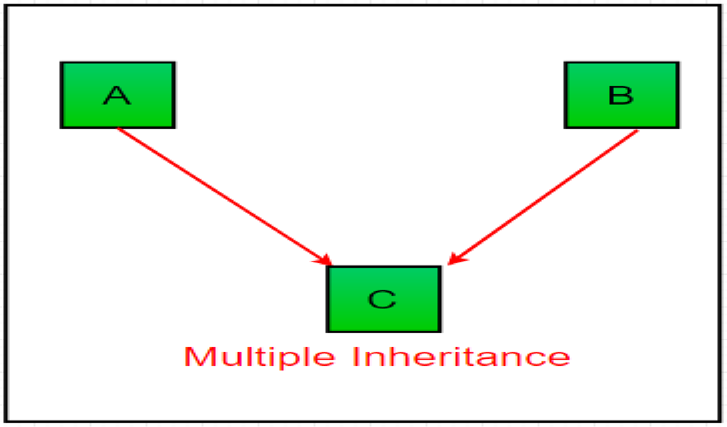
Output:

```
Sum of a and b is: 30
Subtraction of a and b is: 10
Multiplication of a and b is: 200
```

In the above example, class ‘Subtraction’ inherits the properties of class ‘Addition’ and class ‘Multiplication’ will inherit the properties of class ‘Subtraction’. So class ‘Subtraction’ will act as both Base class and derived class.

3) Multiple Inheritance

- The class which inherits the properties of multiple classes is called Multiple Inheritance.
- In multilevel inheritance, features of the base class and the derived class are further inherited into the new derived class.



Example:

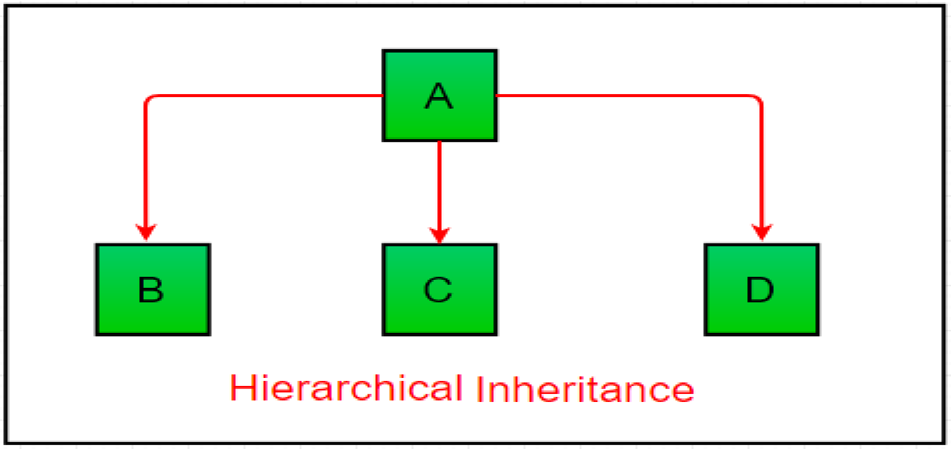
```
# Python program to demonstrate multiple inheritance
class Addition:
    a = 10
    b = 20
    def add(self):
        sum = self. a+ self.b
        print(“Sum of a and b is: “, sum)

class Subtraction:
    c = 50
    d = 10
    def sub(self):
        sub = self.c-self.d
        print(“Subtraction of c and d is: ”, sub)

class Multiplication(Addition,Subtraction):
    def mul(self):
        multi = self.a * self.c
        print(“Multiplication of a and c is: ”, multi)

ob = Multiplication ()
ob.add()
ob.sub()
ob.mul()
```

Output:

```
Sum of a and b is: 30
Subtraction of c and d is: 10
Multiplication of a and c is: 500
```

4) Hierarchical Inheritance

When more than one derived class are created from a single base this type of inheritance is called hierarchical inheritance.



In the program given below, we have a parent (base) class and two child (derived) classes.

Example:

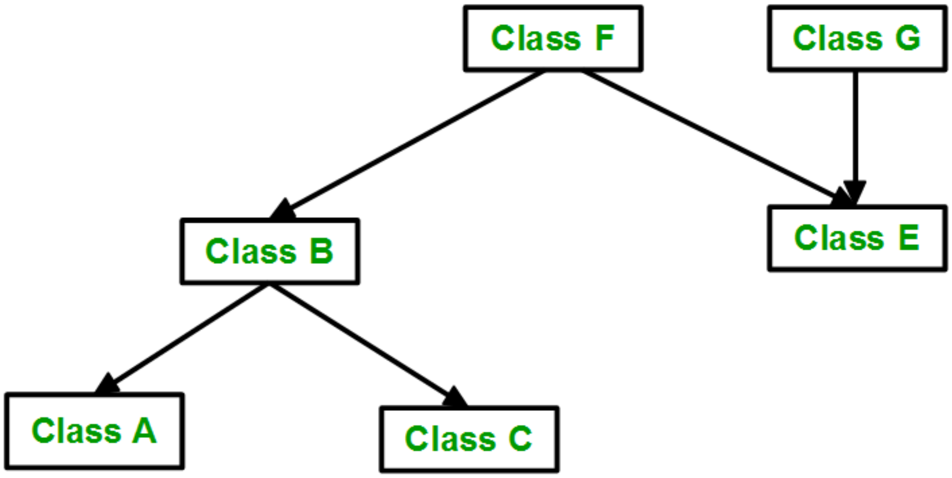
```
# Python program to demonstrate Hierarchical inheritance

# Base class
class Parent:
    def func1(self):
        print("This function is in parent class.")

# Derived class1
 class Child1(Parent):
    def func2(self):
        print("This function is in child 1.")

# Derivied class2
class Child2(Parent):
    def func3(self):
        print("This function is in child 2.")

object1 = Child1()
object2 = Child2()
object1.func1()
object1.func2()
object2.func1()
object2.func3()
```

Output:

```
This function is in parent class.

This function is in child 1.

This function is in parent class.

This function is in child 2.
```

5) Hybrid Inheritance

Inheritance consisting of different types of inheritance is called hybrid inheritance.



Example:

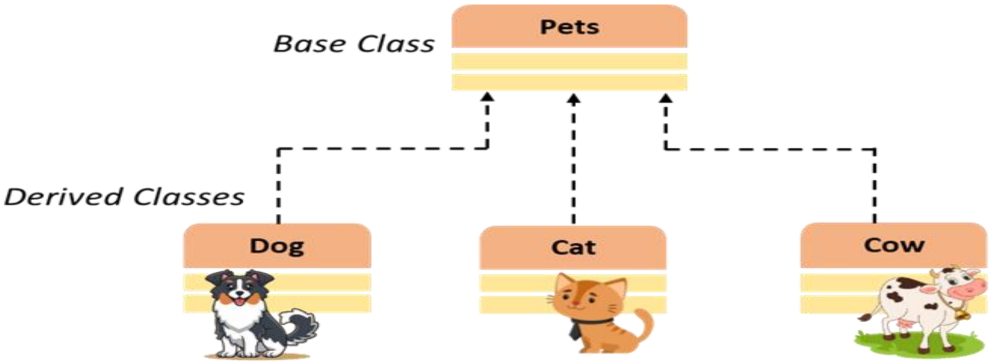
```
# Python program to demonstrate hybrid inheritance
class School:
    def func1(self):
        print("This function is in school.")


class Student1(School):
    def func2(self):
        print("This function is in student 1. ")


class Student2(School):
    def func3(self):
        print("This function is in student 2.")


class Student3(Student1, School):
    def func4(self):
        print("This function is in student 3.")

object = Student3()
object.func1()
object.func2()
```

Output:

```
This function is in school.

This function is in student 1.
```

Additional Examples:

Hierarchical Inheritance



```
#Python Program demonstrating hierarchical inheritance

#Base Class
class Pet:
    def __init__(self, pet_type, name, bdate):
    self.pet_type = pet_type
    self.name = name
    self.bdate = bdate

    def details(self):
    print("I am pet")

#Derived Class 1
class Cat(Pet):
    def __init__(self, pet_type, name, bdate):
    self.name = "Grey " + name
    self.pet_type = pet_type

    def details(self):
    print('I am cute pet', self.pet_type, 'people call me', self.name)

#Derived Class 2
class Dog(Pet):
    def __init__(self, pet_type, name, bdate, breed):
    super().__init__(pet_type, name, bdate)
    self.breed = breed

    def sounds(self, sound):
    return sound

    def details(self):
    print('I am', self.name,',a', self.breed)
```

```
pet1 = Pet('cat', 'Tiffiny', '2019-07-08')
pet2 = Cat('cat', 'Gatsby', '2018-07-08')
pet3 = Dog('dog', 'Toby', '2018-07-08', 'bull dog')
pet4 = Dog('dog', 'Max', '2018-07-08', 'Tibetan Mastiff')

print(pet1.name)
print(pet2.name, "is a chubby", pet2.pet_type)
pet2.details()
print(pet3.name, "is a",pet3.breed, "and it always",pet3.sounds("growls"))
```

Hybrid Inheritance

When inheritance consists of multiple combinations of different inheritance models that we discussed till now, we call that hybrid inheritance

.

```
#Python program demonstrating Hybrid inheritance

#Base Class 1
class Pets:
    def pets_info(self):
        print("Legalized Pet")

class Cat(Pets):
    def cat_info(self):
        print("I am cat")

class Dog(Pets):
    def dog_info(self):
        print("I am a fiercely loyal dog")

# Wild_Cat inherits properties of Pets and Cat
class Wild_Cat(Cat, Pets):
    def wildcat_info(self):
        print("A mighty wild cat")

# create object
w_cat = Wild_Cat()

w_cat.pets_info()
w_cat.cat_info()
w_cat.wildcat_info()
```