IS-A Relationship [inheritance]:

- Process of extending the features of subclass by the Superclass
- In is-a relationship the members of the super class are inherited into subclass directly [not private members]

Syn:

```
<class>  <SuperClassName>:
        Attributes
        Fields [static variable | instance fields]
        Metods [ static methods | class methods | instance methods]

<class>  <SubClassName>(Superclassname(s)):
        Attributes
        Fields
        Methods
```

Example:

```
#example on is-a
class Super:
   def method1(self):
      print("super-class ins mtd-1 ")

   @classmethod
   def method2(cls):
      print("super-class class mtd-2")

   @staticmethod
   def method3():
      print("super-class static mtd-3")
```

```
class Sub(Super):
    pass

#calling
s=Sub( )  #creating an object  Sub class here s is reference of Sub class
s.method1()
s.method2()
s.method3()
```

#Example prg on single inheritance

```
class Super:
    def method1(self):
        print("Mtd-1 of Super Class ")

class Sub(Super):
    def method2(self):
        self.method1()

#calling
s=Sub()
s.method2()
```

```
#Example prg on Multiple .inheritance
class Father:
   def fatherHeight(self):
      print("Height is From Father ")

class Mother:
   def motherColor(self):
      print("Color is From Mother ")

class Son(Father,Mother):
   def properties(self):
      self.fatherHeight()
      self.motherColor()
      print("Qualications from Son")

#calling
s=Son()
s.properties()


#Example prg on Multi-Level Inheritance

class GrandFather:
   def House(self):
      print("House From GrandFather ")

class Father(GrandFather):
   def Car(self):
      print("Car From Father ")
```

```
class Son(Father):
    def bike(self):
        print("Bike From Son")

#calling
s=Son()
s.House()
s.Car()
s.bike()

#Example prg on Hierarchal .inheritance
class Father:
    def money(self):
        print("Money From Father ")

class Son(Father):
    def sonProperties(self):
        print("From Son Class")
        self.money()

class Daughter(Father):
    def daughterProperties(self):
        print("From Dau Class ")
        self.money()

#calling
s=Son()
s.sonProperties()
d=Daughter()
d.daughterProperties()
```

Student Application

```
class Person:
   def setPerson(self): #instance mtd
      self.name=input("Enter name : ")
      self.city=input("Enter city : ")

   def getPerson(self):
      print("Name is : ",self.name)
      print("city is : ",self.city)

class Student(Person):
   def setStudent(self):
      self.setPerson()
      self.course=input("Enter course :")

   def getStudent(self):
      self.getPerson()
      print("Course is : ",self.course)

#calling
s=Student()
s.setStudent()
s.getStudent()
```

Employee Application

```
class PInfo:
   def setPerson(self):
      self.name=input("Enter name : ")
      self.city=input("Enter city : ")

   def getPerson(self):
      print("Name is : ",self.name)
      print("City is :",self.city)

class OInfo(PInfo):
   def setOInfo(self):
      self.job=input("Enter Job : ")
      self.salary=int(input("Enter salary : "))

   def getNetSalary(self):
      self.hra=(self.salary*10)/100
      self.ta=(self.salary*5)/100
      self.netsalary=self.salary+self.ta+self.hra

   def getOInfo(self):
      print("Job is : ",self.job)
      print("Salary is : ",self.salary)
      print("- "*30)
      self.getNetsalary()
      print("Hra is : ",self.hra)
      print("Ta is : ",self.ta)
      print("Netsalary : ",self.netsalary)
```

```
class Employee(OInfo):
   def setEmployee(self):
      self.setPerson()
      self.setOInfo( )

   def getEmployee(self):
      self.getPerson()
      print("- "*30)
      self.getOInfo()

#calling
e=Employee()
e.setEmployee()
e.getEmployee()
```