```
#multilevel inheritance
class Person:
    def __init__(self, name ,age, address):
        self.name = name
        self.age = age
        self.address = address

    def eat(self): # self is object
        print(f'{self.name} is eating')

    def sleep(self):
        print(f'{self.name} is sleeping')

    def walk(self):
        print(f'{self.name} is walking')

    def info(self):
        print(f"Name: {self.name}")
        print(f"Age: {self.age}")
        print(f"Address: {self.address}")


class Student(Person): # Inheritance
    def __init__(self, name ,age, address, college, faculty, roll_no):
        super().__init__(name, age, address) # calling Person's __init__ method
        self.college = college
        self.faculty = faculty
        self.roll_no = roll_no
        self.subjects = []

    def learn(self):
        print(f"Student is learning {self.subjects}.")

    def add_subject(self, subject_name):
        if subject_name not in self.subjects:
            self.subjects.append(subject_name)

    def info(self):
        super().info() # calling Person's info method
        print(f"College: {self.college}")
        print(f"Faculty: {self.faculty}")
        print(f"Roll No: {self.roll_no}")
        print(f"Subjects: {self.subjects}")
```

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

    def eat(self): # self is object
        print(f'{self.name} is eating')

    def sleep(self):
        print(f'{self.name} is sleeping')

    def walk(self):
        print(f'{self.name} is walking')

    def info(self):
        print(f"Name: {self.name}")
        print(f"Age: {self.age}")
        print(f"Address: {self.address}")


class Student(Person): # Inheritance
    def __init__(self, name ,age, address, college, faculty, roll_no):
        super().__init__(name, age, address) # calling Person's __init__ method
        self.college = college
        self.faculty = faculty
        self.roll_no = roll_no
        self.subjects = []
```

```
    def learn(self):
        print(f"Student is learning {self.subjects}.")

    def add_subject(self, subject_name):
        if subject_name not in self.subjects:
            self.subjects.append(subject_name)

    def info(self):
        super().info() # calling Person's info method
        print(f"College: {self.college}")
        print(f"Faculty: {self.faculty}")
        print(f"Roll No: {self.roll_no}")
        print(f"Subjects: {self.subjects}")

class BachelorStudent(Student):
    def __init__(self,name, age, address, college, faculty, roll_no, university):
        # calling Student's __init__ method
        super().__init__(name, age, address, college, faculty, roll_no)
        self.university = university

    # BachelorStudent class le Student class ko info method override gareko ho
    def info(self):
        # calling Student's info method
        super().info()
        print(f"University: {self.university}")



#hierarchical inheritance
class Person:
    pass

class Student(Person):
    pass

class Teacher(Person):
    pass

class Employee(Person):
    pass

class Principal(Person):
    pass


#example #code duplication hataune

class Vehicle:
    def info(self):
        print("This is Vechile")

class Car(Vehicle):
    def car_info(self, name):
        print("Car name is:", name)

class Truck(Vehicle):
    def truck_info(self, name):
        print("Truck name is:", name)

obj1=Car()
obj1.info()
obj1.car_info("BWM")

obj2=Truck()
obj2.info()
obj2.truck_info("Ford")
```

```
This is Vechile
Car name is: BWM
This is Vechile
Truck name is: Ford
```

```
#hybrid inheritance=>sab inheritance mix vako sake samma yo inheritance use nagarne
```

```
class Vehicle:
    def vehicle_info(self):
        print("Inside Vechile class")

class Car(Vehicle):
    def car_info(self):
        print("Inside car class")

class Truck(Vehicle):
    def truck_info(self):
        print("Inside truck class")

#Sports car can inherit properties of vehicle and car
class SportsCar(Car, Vehicle):
    def sports_car_info(self):
        print("Inside SportsCar class")


#create object
s_car = SportsCar()
s_car.vehicle_info()
s_car.car_info()
s_car.sports_car_info()
```

```
Inside Vechile class
Inside car class
Inside SportsCar class
```

```
#self ko kaam
#object created outside of class

class Hello():
    def hi(self): #object created outside of class
        print(self)



h=Hello()
```

```
print(h) #object
```

```
<__main__.Hello object at 0x000001E5865D12B0>
```

```
h.hi()
```

```
<__main__.Hello object at 0x000001E5865D12B0>
```

```
Hello.hi(h)
```

```
<__main__.Hello object at 0x000001E5865D12B0>
```

```
#

class Rectangle:
    def __init__(self, length, breadth):
        #public attributes=> can be accessed outside of class
        self.__length = length #double underscore thapera private attribure banaune jaile ni
        self.__breadth = breadth#private attribute banauney jaile pani code ma

    #public method
    def area(self):
        return self.__length * self.__breadth

    #public method
    def perimeter(self):
        return 2 * (self.__length + self.__breadth)


r1 = Rectangle(3, 5)


r1.area()
```

```
15
```

```
r1.perimeter()
```

```
16
```

```
r1.length
```

```
3
```

```
r1.breadth
```

```
5
```

```
r1.length = 'ram'
```

r1.area()

```
r1.__length = 'ram'
```

r1.area()

```
#method ko through bata herne banaune
#data encapsulation=> access modifier
class Rectangle:
    def __init__(self, length, breadth):
        #public attributes=> can be accessed outside of class
        self.__length = length #double underscore thapera private attribure banaune jaile ni
        self.__breadth = breadth#private attribute banauney jaile pani code ma

    #public method
    def area(self):
        return self.__length * self.__breadth

    #public method
    def perimeter(self):
        return 2 * (self.__length + self.__breadth)


    def __validate(self, value): # code duplication hatako
        if not isinstance(value,(int,float)):
         raise ValueError("Incorrect Datatype")


    def length_getter(self):
        return  self.__length

    def length_setter(self, value): #value set garna lai yo method use garne
        self.__validate(value)
        self.__length = value

    def breadth_getter(self):
        return self.__breadth

    def breadth_setter(self, value): #value set garna lai yo method use garne
       self.__validate(value)
       self.__breadth = value
```

r1 = Rectangle(4, 5)

r1.length_setter(10)

```
r1.length_getter()
```

```
4
```

```
r1.area()
```

```
20
```

```
isinstance(4,int) #int ho ki haina vanera check garxa
```

```
True
```

```
isinstance(4,(int, float))
```

```
True
```

```
isinstance(4,(str, float))
```

```
False
```

```
r1 = Rectangle(4, 5)
```

```
r1.length_setter('ram')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[44], line 1
----> 1 r1.length_setter('ram')

Cell In[41], line 24, in Rectangle.length_setter(self, value)
     20 def length_setter(self, value): #value set garna lai yo method use garne
     21    #validation....
     23     if not isinstance(value, (int, float)):
---> 24         raise ValueError("Incorrect datatype")
     25     self.__length = value

ValueError: Incorrect datatype
```