```python
#multilevel inheritance (yeuta banda badi level ma inherit garnu paryo bhane)
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}")

s1 = BachelorStudent(name="Ram", age=23, address="KTM", college="NCIT", faculty="IT", roll_no=1, university='TU')
s1.info()
```

```
Name: Ram
Age: 23
Address: KTM
College: NCIT
Faculty: IT
Roll No: 1
Subjects: []
University: TU
```

```python
#hierarchical inheritance
class Person:
    pass
class Student(Person):
    pass
class Teacher(Person):
    pass
class Employee(Person):
```

```
    pass
class Principal(Person):
    pass
```

```
class Vehicle:
    def info(self):
        print ("This is called vehicle")

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 called vehicle
Car name is : BWM
This is called vehicle
Truck name is : Ford
```

```
#Hybrid Inheritance => sphageti code(sab inheritance ko mix max )
#sake samma ahybrid inheritance use nagarne

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

```
class Rectangle:
    def __init__(self, length, breadth):
        #public attribute => can be accessed outside of class
        self.length = length
        self.breadth = breadth
    #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()
```

```
'ramramramramram'
```

```
class Rectangle:
    def __init__(self, length, breadth):
        #public attribute => can be accessed outside of class
        self.__length = length
        self.__breadth = breadth
    #public method
    def area(self):
        return self.__length * self.__breadth
    #public method
    def perimeter(self):
        return 2 * (self.__length + self.__breadth)
        #sake samma private banauna parchhaa so method ko aagadi __ thapeko (private attribute banako)




r1 = Rectangle(4,2)


r1.__length
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[61], line 1
----> 1 r1.__length

AttributeError: 'Rectangle' object has no attribute '__length'
```

```
r1.__length ='ram'


r1.area()
```

```
8
```

```
r1.perimeter()
```

```
12
```

```
#method ko through bata herna melchha
class Rectangle:
    def __init__(self, length, breadth):
        #public attribute => can be accessed outside of class
        self.__length = length
        self.__breadth = breadth
    #public method
    def area(self):
        return self.__length * self.__breadth
    #public method
    def perimeter(self):
        return 2 * (self.__length + self.__breadth)
        #sake samma private banauna parchhaa so method ko aagadi __ thapeko (private attribute banako)


    def length_getter(self):
        return self.__length

    def length_setter(self,value):

        self.__length= value

    def breadth_getter(self):
        return self.__breadth

    def breadth_setter(self,value):
        self.__readth= value



r1 = Rectangle(4,5)


r1.length_setter(10)


r1.length_getter()
```

```
10
```

```
r1.area()
```

```
50
```

```
#self
#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 0x0000022BA642B380>
```

```
h.hi()
```

```
<__main__.Hello object at 0x0000022BA642B380>
```

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

```
<__main__.Hello object at 0x0000022BA642B380>
```

```
# int , float  hoki haina bahnera chack garnaa instance use garencha
```

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

```
True
```

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

```
False
```

```
#method ko through bata herna melchha
#data encapsulation => access modifier
class Rectangle:
    def __init__(self, length, breadth):
        #public attribute => can be accessed outside of class
        self.__length = length
        self.__breadth = breadth
    #public method
    def area(self):
        return self.__length * self.__breadth
    #public method
    def perimeter(self):
        return 2 * (self.__length + self.__breadth)
        #sake samma private banauna parchhaa so method ko aagadi __ thapeko (private attribute banako)



    def length_getter(self):
        return self.__length

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

        if not isinstance(value,(int,float)):
            raise ValueError("Incorrect Datatype")

        self.__length= value

    def breadth_getter(self):
        return self.__breadth

    def breadth_setter(self,value):

        if not isinstance(value,(int,float)):
            raise ValueError("Incorrect Datatype")

        self.__breadth= value



r1 = Rectangle(3,5)
```

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

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[69], line 1
----> 1 r1.length_setter('ram')

Cell In[65], line 23, in Rectangle.length_setter(self, value)
     19 def length_setter(self,value):# value set garna lai yo method use garne
     20     #validation....
     22     if not isinstance(value,(int,float)):
---> 23         raise ValueError("Incorrect Datatype")
     25     self.__length= value

ValueError: Incorrect Datatype
```

```
#method ko through bata herna melchha
#data encapsulation => access modifier
class Rectangle:
    def __init__(self, length, breadth):
        #public attribute => can be accessed outside of class
        self.__length = length
        self.__breadth = breadth
    #public method
```

```
    def area(self):
        return self.__length * self.__breadth
    #private method
    def perimeter(self):
        return 2 * (self.__length + self.__breadth)
        #sake samma private banauna parchhaa so method ko aagadi __ thapeko (private attribute banako)

    def __validate(self,value):
        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(True)
        self.__length= value

    def breadth_getter(self):
        return self.__breadth

    def breadth_setter(self,value):
        self.__validate(True)
        self.__breadth= value



Start coding or generate with AI.
```