Define a property that must have the same value for every class instance (object)
Define a class attribute”color” with a default value white. I.e., Every Vehicle should be white.

In [10]:
```
class vehicle:
    color="white"
    def __init__(self,name,speed):
        self.name=name
        self.speed=speed
    def seating_capacity(self,capacity):
        self.capacity=capacity
        print("Vehicle Name:", self.name, "Speed:", self.speed, "Color:", vehicle.color,"Capacity: ",self.ca
```

In [11]:
```
b=bus("Volvo",120)
b.seating_capacity(50)
# b.info()
```

```
Vehicle Name: Volvo Speed: 120 Color: white Capacity:  50
```

Create a Bus child class that inherits from the Vehicle class. The default fare charge of any vehicle is seating capacity * 100. If Vehicle is Bus instance, we need to add an extra 10% on full fare as a maintenance charge. So total fare for bus instance will become the final amount = total fare + 10% of the total fare.

In [13]:
```
class vehicle:
    color="white"
    def __init__(self,name,speed,capacity):
        self.name=name
        self.speed=speed
        self.capacity=capacity
    def fare(self):
        total_fare=self.capacity*100
        return total_fare
class bus(vehicle):
    def fare(self):
        total_amount=super().fare()
        total_amount+=total_amount*10/100
        print("Vehicle Name:", self.name ,"Speed:", self.speed,"Capacity: ",self.capacity, "Total amount of
```

In [14]:
```
b=bus("Volvo",120,50)
```

In [15]:
```
b.fare()
```

```
Vehicle Name: Volvo Speed: 120 Capacity:  50 Total amount of fare :  5500.0
```

In [16]:
```
# Write a program to determine which class a given Bus object belongs to.
```

In [17]:
```
class vehicle:
    color="white"
    def __init__(self,name,speed,capacity):
        self.name=name
        self.speed=speed
        self.capacity=capacity
    def fare(self):
        total_fare=self.capacity*100
        return total_fare
class bus(vehicle):
    def fare(self):
        total_amount=super().fare()
        total_amount+=total_amount*10/100
        print("Vehicle Name:", self.name ,"Speed:", self.speed,"Capacity: ",self.capacity, "Total amount of
```

In [18]:
```
b=bus("Volvo",120,50)
print(type(b))
```

```
<class '__main__.bus'>
```

In [20]:
```
print(isinstance(b,vehicle))
```

```
True
```

In [ ]:
```
Modify Object Properties
```

In [29]:
```
class vehicle:
    color="white"
    def __init__(self,name,speed,capacity):
        self.name=name
        self.speed=speed
        self.capacity=capacity
    def fare(self):
        total_fare=self.capacity*100
        print(f"Name : {self.name}")
        return total_fare
```

In [30]:
```
b1=vehicle("Volvo",120,50)
```

In [31]:
```
b1.fare()
```

```
Name : Volvo
```

Out[31]:
```
5000
```

In [32]:
```
b1.name
```

Out[32]:
```
'Volvo'
```

In [33]:
```
b1.name="Travels"
```

In [34]:
```
b1.fare()
```

```
Name : Travels
```

Out[34]:
```
5000
```

In [35]:
```
# Delete object properties
del b1.name
```

In [36]:
```
b1.name
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[36], line 1
----> 1 b1.name

AttributeError: 'vehicle' object has no attribute 'name'
```

In [37]:
```
del b1
```

In [38]:
```
b1.fare()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[38], line 1
----> 1 b1.fare()

NameError: name 'b1' is not defined
```

In [55]:
```
class employee_details:
    def __init__(self, name, age, department, position, salary):
        self.name=name
        self.age=age
        self.department=department
        self.position=position
        self.__salary=salary
    def promotion(self, new_position, new_salary):
        self.new_position=new_position
        self.__new_salary=new_salary
        if self.age>25 and self.__salary >30000:
            return f"{self.name} is promoted for {self.new_position} and salary : {self.__new_salary}"
        else:
            return f"{self.name} not promoted for {self.new_position} and salary : {self.__salary}"
```

In [56]:
```
OBJ=employee_details("riya",25,"IT","manager",50000)
print(OBJ.promotion("developer",60000))
```

```
riya not promoted for developer and salary : 50000
```

In [57]:
```
OBJ.salary
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[57], line 1
----> 1 OBJ.salary

AttributeError: 'employee_details' object has no attribute 'salary'
```

In [58]:
```
OBJ1=employee_details("harry",30,"IT","manager",50000)
print(OBJ1.promotion("developer",60000))
```

```
harry is promoted for developer and salary : 60000
```

In [59]:
```
OBJ1.salary
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[59], line 1
----> 1 OBJ1.salary

AttributeError: 'employee_details' object has no attribute 'salary'
```

In [60]:
```
OBJ1.new_salary
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[60], line 1
----> 1 OBJ1.new_salary

AttributeError: 'employee_details' object has no attribute 'new_salary'
```

In [61]:
```
# salary and new_salary are private variables
```

In [ ]:
```
# to acces private variable from the class
```

In [62]:
```
OBJ1._employee_details__new_salary
```

Out[62]:
```
60000
```

In [63]:
```
OBJ._employee_details__salary
```

Out[63]:
```
50000
```

In [ ]:
```
# to access protected variable
```

In [66]:
```
class employee_details:
    def __init__(self, name, age, department, position, salary):
        self.name=name
        self._age=age
        self.department=department
        self.position=position
        self.__salary=salary
    def promotion(self, new_position, new_salary):
        self.new_position=new_position
        self.__new_salary=new_salary
        if self._age>25 and self.__salary >30000:
            return f"{self.name} is promoted for {self.new_position} and salary : {self.__new_salary}"
        else:
            return f"{self.name} not promoted for {self.new_position} and salary : {self.__salary}"
```

In [67]:
```
OBJ1=employee_details("harry",30,"IT","manager",50000)
print(OBJ1.promotion("developer",60000))
```

```
harry is promoted for developer and salary : 60000
```

In [70]:
```
OBJ1._age
```

Out[70]:
```
30
```

In [71]:
```
# Getters and Setters Methods
```

In [86]:
```
class employee:
    def __init__(self,e_id,name,age,experience,salary):
        self.e_id=e_id
        self.name=name
        self_age=age
        self.experience=experience
        self.__salary=salary
    def get_age(self):
        return self._age
    def set_age(self,age):
        self._age=age
    def get_salary(self,new_salary):
        if self.experience > 5:
            self.__new_salary=new_salary
            return self.__new_salary
        else:
            return self.__salary
```

In [92]:
```
emp=employee(1,"riya",23,1,30000)
```

In [93]:
```
emp.get_salary(35000)
```

Out[93]:
```
30000
```

In [94]:
```
emp=employee(1,"riya",23,7,30000)
emp.get_salary(450000)
```

Out[94]:
```
450000
```

In [ ]: