Object Oriented Programming (OOPs) Concepts

Create a Car class with attributed like brand and model. Then create an instence of this class.

In [8]:
```python
class Car:
    def __init__(self,brand , model):
        self.brand = brand
        self.model = model

my_car = Car("Audi", "RBV10")
print(my_car.brand)
print(my_car.model)
```

```
Audi
RBV10
```

In [9]:
```python
my_car = Car("BMW", "7 Series")
print(my_car.brand)
print(my_car.model)
```

```
BMW
7 Series
```

Class Method and self

Add a method to the Car class that displays the full name of the car(brand and model).

In [16]:
```python
class Car:
    def __init__(self,brand , model):
        self.brand = brand
        self.model = model
    def full_name(self):
        return f"{self.brand} {self.model}"

my_car = Car("BMW", "7 Series")
print(my_car.full_name())
```

```
BMW 7 Series
```

Inheritance

create an Electric Car class that inherits from tha Car class and has an additional attribute battery_size.

In [27]:
```python
class Car:
    def __init__(self,brand, model):
        self.brand = brand
        self.model = model
    def full_name(self):
        return f"{self.brand} {self.model} {self.battery_size}"
class ElectricCar(Car):
    def __init__(self, brand, model, battery_size):
        super().__init__(brand, model)
        self.battery_size = battery_size

my_car = ElectricCar("audie", "R8V10", "94R")
print(my_car.model)
print(my_car.brand)
print(my_car.battery_size)
print(my_car.full_name())
```

```
R8V10
audie
94R
audie R8V10 94R
```

Encapsulation

Modify the Car class fo encapsulate the brand attribute, making it private, and provide a getter method for it.

In [3]:
```
class Car:
    def __init__(self,brand, model):
        self.__brand = brand           #Made it private
        self.model = model
    def get_brand(self):
        return self.__brand + "!"   #Make it accessable with getter mathod

    def full_name(self):
        return f"{self.__brand} {self.model} {self.battery_size}"
class ElectricCar(Car):
    def __init__(self, brand, model, battery_size):
        super().__init__(brand, model)
        self.battery_size = battery_size

my_car = ElectricCar("audie", "R8V10", "94R")
print(my_car.model)
#print(my_car.__brand)
print(my_car.get_brand())
print(my_car.full_name())
```

```
R8V10
audie!
audie R8V10 94R
```

Polymorphism

Demonstrate polymorphism by defining a method fuel_type in both Car and ElectricCar classes, but with different behaviors.

In [13]:
```
class Car:
    def __init__(self,brand, model):
        self.__brand = brand           #Made it private
        self.model = model
    def get_brand(self):
        return self.__brand + "!"   #Make it accessable with getter mathod

    def full_name(self):
        return f"{self.__brand} {self.model}"

    def fuel_type(self):
        return "Petrol or Diesel."
```

```
class ElectricCar(Car):
    def __init__(self, brand, model, battery_size):
        super().__init__(brand, model)
        self.battery_size = battery_size

    def fuel_type(self):
        return "Electric charge."

my_car = Car("audie", "Tesla")
print(my_car.fuel_type())

E_car = ElectricCar("Tesla", "Model S", "85kwh")
print(E_car.fuel_type())
#print(my_car.__brand)
#print(my_car.fuel_type())
#print(my_car.get_brand())
#print(my_car.full_name())
```

```
Petrol or Diesel.
Electric charge.
```

Two/many different object can use one function in two/many different way.

Class Veriables

Add a class variable to Car that keeps track of the number of cars created.

In [6]:
```
class Car:
    total_car = 0
    def __init__(self,brand, model):
        self.__brand = brand          #Made it private
        self.model = model
        Car.total_car +=1

    def get_brand(self):
        return self.__brand + "!"   #Make it accessable with getter mathod

    def full_name(self):
        return f"{self.__brand} {self.model}"

    def fuel_type(self):
        return "Petrol or Diesel."

class ElectricCar(Car):
    def __init__(self, brand, model, battery_size):
        super().__init__(brand, model)
        self.battery_size = battery_size

    def fuel_type(self):
        return "Electric charge."

my_car = Car("audie", "R8V10")
#print(my_car.fuel_type())
my_car1 = Car("audie", "Tesla")
#print(my_car1.fuel_type())
E_car = ElectricCar("Tesla", "Model S", "85kwh")
#print(E_car.fuel_type())
E_car1 = ElectricCar("TATA", "E6", "85kwh")
#print(E_car1.fuel_type())
```

```
print(Car.total_car) #we can directly get the access of the attribute from the class
print(E_car.total_car)
```

```
4
4
```

Static Method

Add a static method to the Car class that returns a general description of a car.

In [29]:
```
class Car:
    total_car = 0
    def __init__(self,brand, model):
        self.__brand = brand          #Made it private
        self.model = model
        Car.total_car +=1

    def get_brand(self):
        return self.__brand + "!"   #Make it accessable with getter mathod

    def full_name(self):
        return f"{self.__brand} {self.model}"

    def fuel_type(self):
        return "Petrol or Diesel."
    def general_description(self):
        return "Cars are means of transport"

class ElectricCar(Car):
    def __init__(self, brand, model, battery_size):
        super().__init__(brand, model)
        self.battery_size = battery_size

    def fuel_type(self):
        return "Electric charge."

my_car = Car("audie", "R8V10")
my_car1 = Car("audie", "Tesla")
E_car = ElectricCar("Tesla", "Model S", "85kwh")
E_car1 = ElectricCar("TATA", "E6", "85kwh")
print(my_car.general_description())
print(Car.general_description())
```

```
Cars are means of transport
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[29], line 32
     30 E_car1 = ElectricCar("TATA", "E6", "85kwh")
     31 print(my_car.general_description())
---> 32 print(Car.general_description())

TypeError: Car.general_description() missing 1 required positional argument: 'self'
```

In [9]:
```
class Car:
    total_car = 0
    def __init__(self,brand, model):
        self.__brand = brand          #Made it private
        self.model = model
        Car.total_car +=1

    def get_brand(self):
        return self.__brand + "!"   #Make it accessable with getter mathod
    def full_name(self):
        return f"{self.__brand} {self.model}"

    def fuel_type(self):
        return "Petrol or Diesel."
    @staticmethod
    def general_description():
        return "Cars are means of transport"

class ElectricCar(Car):
    def __init__(self, brand, model, battery_size):
        super().__init__(brand, model)
        self.battery_size = battery_size

    def fuel_type(self):
        return "Electric charge."

my_car = Car("audie", "R8V10")
print(my_car.general_description())
print(Car.general_description())
```

```
Cars are means of transport
Cars are means of transport
```

Property Decorators

Use a property decorator in the Car class to makes the model attribute read-only.

In [27]:
```
class Car:
    total_car = 0
    def __init__(self,brand, model):
        self.__brand = brand          #Made it private
        self.__model = model
        Car.total_car +=1

    def get_brand(self):
        return self.__brand + "!"    #Make it accessable with getter mathod

    def full_name(self):
        return f"{self.__brand} {self.___model}"

    def fuel_type(self):
        return "Petrol or Diesel."
    @staticmethod
    def general_description():
        return "Cars are means of transport"

class ElectricCar(Car):
    def __init__(self, brand, model, battery_size):
        super().__init__(brand, model)
        self.battery_size = battery_size

    def fuel_type(self):
        return "Electric charge."

my_car = Car("audie", "R8V10")
E_car = ElectricCar("Tesla", "Model S", "85kwh")
my_car.model = "BMW"

print(my_car.model)  #here model is overridable
```

```
BMW
```

In [32]:
```python
class Car:
    total_car = 0
    def __init__(self,brand, model):
        self.__brand = brand          #Made it private
        self.__model = model
        Car.total_car +=1

    def get_brand(self):
        return self.__brand + "!"   #Make it accessable with getter mathod

    def full_name(self):
        return f"{self.__brand} {self.__model}"

    def fuel_type(self):
        return "Petrol or Diesel."
    @staticmethod
    def general_description():
        return "Cars are means of transport"
    @property
    def model(self):
        return self.__model

class ElectricCar(Car):
    def __init__(self, brand, model, battery_size):
        super().__init__(brand, model)
        self.battery_size = battery_size

    def fuel_type(self):
        return "Electric charge."

my_car = Car("audie", "R8V10")
#my_car.model = "BMW"      # Can't override

print(my_car.model)    # Property decorator gives us method as attribute to access
```

```
R8V10
```

Class Inheritance and isinstance() Function

Demonstrate the use of isinstance() to check if my_tesla is an instance of Car and ElectricCar.

In [15]:
```python
class Car:
    total_car = 0
    def __init__(self,brand, model):
        self.__brand = brand          #Made it private
        self.__model = model
        Car.total_car +=1

    def get_brand(self):
        return self.__brand + "!"   #Make it accessable with getter mathod

    def full_name(self):
        return f"{self.__brand} {self.__model}"

    def fuel_type(self):
        return "Petrol or Diesel."
    @staticmethod
    def general_description():
        return "Cars are means of transport"
    @property
    def model1(self):
        return self.__model

class ElectricCar(Car):
    def __init__(self, brand, model, battery_size):
        super().__init__(brand, model)
        self.battery_size = battery_size

    def fuel_type(self):
        return "Electric charge."

#my_car = Car("audie", "R8V10")
my_tesla = ElectricCar("Tesla", "Model S", "85kwh")
print(isinstance(my_car, ElectricCar))
print(isinstance(my_tesla, ElectricCar))
```

```
False
True
```

Multiple Inheritance

Create two classes Battery and Engine, and let the ElectricCar class inherit from both, demonstrating multiple inheritance.

In [33]:
```
class Car:
    total_car = 0
    def __init__(self,brand, model):
        self.__brand = brand          #Made it private
        self.__model = model
        Car.total_car +=1

    def get_brand(self):
        return self.__brand + "!"    #Make it accessable with getter mathod

    def full_name(self):
        return f"{self.__brand} {self.__model}"

    def fuel_type(self):
        return "Petrol or Diesel."
    @staticmethod
    def general_description():
        return "Cars are means of transport"
    @property
    def model1(self):
        return self.__model

class ElectricCar(Car):
    def __init__(self, brand, model, battery_size):
        super().__init__(brand, model)
        self.battery_size = battery_size

    def fuel_type(self):
        return "Electric charge."

class Battery:
    def battery_info(self):
        return "This is battery"

class Engine:
    def engine_info(self):
        return "This is engine"

class E_car(Battery, Engine, Car):
```

```python
    pass

my_new_tesla = E_car("tesla", "Model_5")

print(my_new_tesla.engine_info())
print(my_new_tesla.battery_info())
```

```
This is engine
This is battery
```

In [23]:
```python
class Battery:
    def battery_info(self):
        return "This is battery"

class Engine:
    def engine_info(self):
        return "This is engine"

class E_car(Battery, Engine,Car):
    pass

my_new_tesla = E_car("tesla", "Model_5")

print(my_new_tesla.engine_info())
print(my_new_tesla.battery_info())
```

```
This is engine
This is battery
```