OOP

learn about Object Oriented Programming by answering these questions

▼ 1. Basic Class and Object
Problem: Create a Car class with attributes like brand and model. Then create an instance of this class.
▼ 2. Class Method and Self
Problem: Add a method to the Car class that displays the full name of the car (brand and model).
▼ 3. Inheritance
Problem: Create an ElectricCar class that inherits from the Car class and has an additional attribute battery_size.
▼ 4. Encapsulation
Problem: Modify the Car class to encapsulate the brand attribute, making it private, and provide a getter method for it.
▼ 5. Polymorphism
Problem: Demonstrate polymorphism by defining a method fuel_type in both Car and ElectricCar classes, but with different behaviors.
▼ 6. Class Variables
Problem: Add a class variable to Car that keeps track of the number of cars created.
▼ 7. Static Method
Problem: Add a static method to the Car class that returns a general description of a car.
▼ 8. Property Decorators
Problem: Use a property decorator in the Car class to make the model attribute read-only.
▼ 9. Class Inheritance and isinstance() Function
Problem: Demonstrate the use of isinstance() to check if my_tesla is an instance of Car and ElectricCar.
▼ 10. Multiple Inheritance
Problem: Create two classes Battery and Engine, and let the ElectricCar class inherit from both, demonstrating multiple inheritance.

▼ 1. Basic Class and Object
Problem: Create a Car class with attributes like brand and model. Then create an instance of this class.

```
class Car:
  brand = None
  model = None

my_car = Car()
print(my_car)
```

```
<__main__.Car object at 0x7f5b0d53d0f0>
```

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

my_car = Car("Toyota","Corolla")
print(my_car)
print(my_car.brand)
print(my_car.model)

my_new_car = Car("Tata","Safari")
print(my_new_car.brand)
print(my_new_car.model)
```

```
<__main__.Car object at 0x7f5b0e18b2b0>
Toyota
Corolla
Tata
Safari
```

▼ 2. Class Method and Self

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

```
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("Toyota","Corolla")
print(my_car)
print(my_car.brand)
print(my_car.model)
print(my_car.full_name())
```

```
<__main__.Car object at 0x7a487cf42470>
Toyota
Corolla
Toyota Corolla
```

▼ 3. Inheritance

Problem: Create an ElectricCar class that inherits from the Car class and has an additional attribute battery_size.

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

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

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

my_tesla = ElectricCar("Tesla","Model S","85Kwh")
print(my_tesla.brand)
print(my_tesla.full_name())
```

```
Tesla
Tesla Model S
```

▼ 4. Encapsulation

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

```
class Car:
  def __init__(self, brand,model):
    self.__brand = brand
    self.model = model

  def get_brand(self):
    return self.__brand

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

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

my_tesla = ElectricCar("Tesla","Model S","85Kwh")

print(my_tesla.get_brand())
# print(my_tesla.__brand)
```

```
Tesla
```

▼ 5. Polymorphism

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

```
class Car:
  def __init__(self, brand,model):
    self.__brand = brand
    self.model = model

  def get_brand(self):
    return self.__brand

  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_tesla = ElectricCar("Tesla","Model S","85Kwh")
safari = Car("Tata","Safari")

print(safari.fuel_type())
print(my_tesla.fuel_type())
```

```
Petrol or Diesel
Electric Charge
```

▼ 6. Class Variables

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

```
class Car:
  total_car = 0
  def __init__(self, brand,model):
    self.__brand = brand
    self.model = model
    Car.total_car+=1
    #self.total_car+=1

  def get_brand(self):
    return self.__brand

  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_tesla = ElectricCar("Tesla","Model S","85Kwh")
safari = Car("Tata","Safari")

print(safari.fuel_type())
print(my_tesla.fuel_type())
print(Car.total_car)
```

```
Petrol or Diesel
Electric Charge
2
```

▼ 7. Static Method

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

```
class Car:
    total_car = 0
    def __init__(self, brand,model):
        self.__brand = brand
        self.model = model
        Car.total_car+=1
        #self.total_car+=1

    def get_brand(self):
        return self.__brand

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

    def fuel_type(self):
        return "Petrol or Diesel"
    @staticmethod
    def gen_desc():
      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_tesla = ElectricCar("Tesla","Model S","85Kwh")
  safari = Car("Tata","Safari")

  print(Car.gen_desc())
```

```
Cars are means of transport
```

▼ 8. Property Decorators

Problem: Use a property decorator in the Car class to make the model attribute read-only.

```
class Car:
  total_car = 0
  def __init__(self, brand,model):
    self.__brand = brand
    self.__model = model
    Car.total_car+=1
    #self.total_car+=1

  def get_brand(self):
    return self.__brand

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

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

  @staticmethod
  def gen_desc():
    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_tesla = ElectricCar("Tesla","Model S","85Kwh")
safari = Car("Tata","Safari")
# safari.model = "city"
print(safari.model)
```

```
Safari
```

▼ 9. Class Inheritance and isinstance() Function

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

```
class Car:
  total_car = 0
  def __init__(self, brand,model):
    self.__brand = brand
    self.__model = model
    Car.total_car+=1
    #self.total_car+=1

  def get_brand(self):
    return self.__brand

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

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

  @staticmethod
  def gen_desc():
    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_tesla = ElectricCar("Tesla","Model S","85Kwh")
print(isinstance(my_tesla, Car))
print(isinstance(my_tesla, ElectricCar))
```

```
True
True
```

▼ 10. Multiple Inheritance
Problem: Create two classes Battery and Engine, and let the ElectricCar class inherit from both, demonstrating multiple inheritance.

```
class Car:
    total_car = 0

    def __init__(self, brand, model):
        self.__brand = brand
        self.__model = model
        Car.total_car += 1

    def get_brand(self):
        return self.__brand + " !"

    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():
        return "Electric charge"


# my_tesla = ElectricCar("Tesla", "Model S", "85kWh")

# print(isinstance(my_tesla, Car))
# print(isinstance(my_tesla, ElectricCar))

# print(my_tesla.__brand)
# print(my_tesla.fuel_type())

# my_car = Car("Tata", "Safari")
# my_car.model = "City"
```

```python
# Car("Tata", "Nexon")


# print(my_car.general_description())
# print(my_car.model)


# my_car = Car("Toyota", "Corolla")
# print(my_car.brand)
# print(my_car.model)
# print(my_car.full_name())

# my_new_car = Car("Tata", "Safari")
# print(my_new_car.model)



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

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

class ElectricCarTwo(Battery, Engine, Car):
    pass

my_new_tesla = ElectricCarTwo("Tesla", "Model S")
print(my_new_tesla.engine_info())
print(my_new_tesla.battery_info())
```