```python
"""
1. Instance Variables
   1. Instance Variables will be different for each object.
   2. They do not share the same state.
   3. These are defined inside __init__ method.
   4. These can be accessed with the self keyword.
   When to Use:
   Each Object has different values and
   one object value doesn't depend on another.
2. Class Variables
   1. Class Variables are same across all the objects.
   2. They share the same state unless modified by the object.
   3. These are written outside of all the methods.
   4. These can be accessed with the self keyword, cls keyword (or)
      classname.
   When to Use:
   All the values share the same value.
   Ex: Count the number of objects created.
"""
class Vehicle:
    def __init__(self, n_tyres):
        self.n_tyres = n_tyres # Instance Variable


bike = Vehicle(2)
car = Vehicle(4)
print(bike.n_tyres) # 2
print(car.n_tyres) # 4


class Vehicle:
    vehicle_type = "Road Ways" # Class Variable

    def __init__(self, n):
        self.n_tyres = n # Instance Variable

bike = Vehicle(2)
car = Vehicle(2)
print(bike.n_tyres)
print(car.n_tyres)
print(bike.vehicle_type)
print(car.vehicle_type)

# Vehicle.vehicle_type = "Air Ways"
print(bike.vehicle_type)
print(car.vehicle_type)

ship = Vehicle(0)
ship.vehicle_type = "Water Ways"
print(ship.vehicle_type)
print(bike.vehicle_type)
print(car.vehicle_type)

Vehicle.vehicle_type = "Air Ways"
print(ship.vehicle_type)
print(bike.vehicle_type)
print(car.vehicle_type)
```

```
"""
1. Instance Methods:
   1. Instance Methods are used to access and modify
      the state of the instance variables.
   2. They accept self as the first parameter which refers to the
   object that invoked the method.
   3. They can access Instance Variables and also Class Variables.
   4. No Decorator needed to declare the Instance Methods.
   When to Use:
   We need to access and modify the instance variables.
2. Class Methods:
    1. Class Methods are used to access and modify state
       of the Class Variables.
    2. They accept cls as the first parameter which refers to the
    class that invoked the method.
    3. They can access only Class Variables and cannot access Instance Variables.
    4. We use @classmethod decorator to create class methods.
    When to Use:
    We need to access and modify only class variables and
    not instance variables.
3. Static Methods:
   1. Static Methods are used as utility functions.
   2. They neither accept self keyword (or) cls keyword as their
      first parameter.
   3. They can neither access instance variables nor class variables.
   4. We use @staticmethod decorator to create static methods.
   When to Use:
   Used as utility functions so that
   we can use them without the object creation.
"""
class Vehicle:
    vehicle_type = None

    def __init__(self, tyres, vehicle):
        self.n_tyres = tyres
        self.vehicle = vehicle

    def print_vehicle_details(self): # Instance Method
        print(f"{self.vehicle} is of type {self.vehicle_type} and it has
{self.n_tyres} tyres.")


Vehicle.vehicle_type = "Roadways"
bike = Vehicle(2, "Bike")
car = Vehicle(4, "Car")
bike.print_vehicle_details()
car.print_vehicle_details()


class Vehicle:
    vehicle_type = None

    def __init__(self, tyres, vehicle):
        self.n_tyres = tyres
        self.vehicle = vehicle

    def print_vehicle_details(self): # Instance Method
        print(f"{self.vehicle} is of type {self.vehicle_type} and it has
{self.n_tyres} tyres.")

    @classmethod
    def change_vehicle_type(cls, new_vehicle_type):
        cls.vehicle_type = new_vehicle_type
        # print(cls.vehicle)
        # print(cls.n_tyres)


Vehicle.vehicle_type = "Roadways"
bike = Vehicle(2, "Bike")
car = Vehicle(4, "Car")
bike.print_vehicle_details()
car.print_vehicle_details()

Vehicle.change_vehicle_type("Airways")
plane = Vehicle(3, "Plane")
plane.print_vehicle_details()


class Mathematics:
    @staticmethod
    def add(x, y):
        return x + y

    @staticmethod
    def subtract(x, y):
        return x - y

    @staticmethod
    def multiply(x, y):
        return x * y


result = Mathematics.add(10, 20)
print(result)
result = Mathematics.subtract(100, 50)
print(result)
result = Mathematics.multiply(10, 20)
print(result)


class Cart:
    discount_value = 20
    min_bill_amount = 100

    def __init__(self):
        self.items = {}

    def add_item(self, item, quantity):
        self.items[item] = quantity

    def display_items(self):
        print(self.items)

    @classmethod
    def update_min_bill_amount(cls, new_amount):
        cls.min_bill_amount = new_amount
    @classmethod
    def update_discount(cls, new_discount):
        cls.discount_value = new_discount


person1 = Cart()
person1.add_item("Learn Python Programming", 1)
person1.add_item("Laptop", 1)
person1.add_item("Pen", 10)

person2 = Cart()
person2.add_item("Rice 25Kg Bag", 1)
person2.add_item("Sugar 1Kg Packet", 2)

person1.display_items()
person2.display_items()
print(person1.discount_value)
print(person2.discount_value)
print(person1.min_bill_amount)
print(person2.min_bill_amount)

Cart.update_discount(30)
Cart.update_min_bill_amount(200)

print(person1.discount_value)
print(person2.discount_value)
print(person1.min_bill_amount)
print(person2.min_bill_amount)
```