OOPS DAY-1 TASK

1. **Question 1 – Zomato Food Ordering**

Create a class Restaurant to simulate Zomato's restaurant system.

Requirements:

1. The constructor should accept restaurant_name and menu (dictionary of items → price).
2. Define a method showMenu() that displays all items with prices. o Example: Pizza - ₹200, Burger - ₹120
3. Define a method orderFood(item, qty) that: o Checks if the item is available in the menu. o If available → prints the bill (price * qty). o If not available → prints "Item not available".
4. Create two restaurant objects (Dominos, KFC) with different menus.
5. Place orders from both restaurants.

Code:-

```
class restauraunt:
    def __init__(self,r,m):
        self.restaurant=r
        self.menu=m

    def showMenu(self):
        print(f"---------The {self.restaurant} menu ---------- ")
        for item, price in self.menu.items():
            print(f"{item} - ${price}")

    def orderFood(self,item,qty):
        if item in self.menu:
            total=self.menu[item] * qty
            print(f"Order placed: {qty} x {item} = ₹{total}") else:
            print("Item not available")

dominos=restauraunt("Dominos",{
    "pizza":200,
    "garlic-bread":120,
    "pasta":150
})
kfc=restauraunt("KFC",{
    "Burger":150,
    "chicken wings":250,
    "Fries":100
    })

dominos.showMenu()
kfc.showMenu()

dominos.orderFood("Pizza", 2)
dominos.orderFood("Burger", 1)
```

```
kfc.orderFood("Fries", 3)
kfc.orderFood("Pasta", 1)
```

Output:-

```
---------The Dominos menu----------
pizza - $200
garlic-bread - $120
pasta - $150
---------The KFC menu----------
Burger - $150
chicken wings - $250
Fries - $100
Item not available
Item not available
Order placed: 3 x Fries = ₹300
Item not available
```

2. **Question 2 – Uber Ride Booking**

Create a class Driver to simulate Uber driver details.

Requirements:

1. The constructor should accept driver_name, car_model, and per_km_rate.
2. Define a method showDriver() that prints driver details.
o Example: Driver: Raj, Car: Swift, Rate: ₹20/km
3. Define a method calculateFare(distance) that calculates and prints the fare.
o Example: Distance: 10 km, Fare: ₹200
4. Create 3 driver objects with different details.
5. For each driver, calculate fare for a trip (e.g., 8 km, 12 km, 15 km)

Code:-

```
class Driver:
    def __init__(self,n,cn,pkr):
        self.name=n
        self.carName=cn
        self.kmrate=pkr

    def showDriver(self):
        print(f"Driver:{self.name}, Car:{self.carName}, Rate:{self.kmrate}")

    def calculateFair(self,distance):
        fare= distance * self.kmrate
        print(f"Distance:{distance}km, Fare:${fare}")

driver1=Driver("Ram","Porsche",50)
driver2=Driver("Shyam","Bentley",60)
driver3=Driver("Ravi","BMW",40)

driver1.showDriver()
driver2.showDriver()
```

```
driver3.showDriver()

driver1.calculateFair(5)
driver2.calculateFair(10)
driver3.calculateFair(20)
```

Output:-

```
Driver:Ram, Car:Porsche, Rate:50
Driver:Shyam, Car:Bentley, Rate:60
Driver:Ravi, Car:BMW, Rate:40
Distance:5km, Fare:$250
Distance:10km, Fare:$600
Distance:20km, Fare:$800
```

3. Question 3 – Flipkart Shopping Cart

Create a class Product to simulate shopping on Flipkart.

Requirements:

1. The constructor should accept product_name, price, and stock.
2. Define a method showDetails() that displays product details.
   o Example: Laptop - ₹50000, Stock: 10
3. Define a method buyProduct(qty) that:
   o If enough stock → reduces stock and prints total cost.
   o If not enough stock → print "Out of Stock".
4. Create 3 product objects (Laptop, Phone, Shoes) with different stock and price.
5. Simulate a shopping cart by:
   o Buying 2 items from Laptop.
   o Buying 1 item from the Phone.
   o Trying to buy more shoes than available stock.

Code:-

```
class Product:
    def __init__(self,pn,p,s):
        self.Product_name=pn
        self.price=p
        self.stock=s

    def showDetails(self):
        print(f"{self.Product_name} -${self.price}, Stock:{self.stock}")

    def buyProduct(self,qty):
        if qty <= self.stock:
            total= self.price * qty
            self.stock -= qty
            print(f"Purchased {qty} x {self.Product_name} = ${total}")
            print(f"Remaining Stock of {self.Product_name}: {self.stock}")
        else:
            print(f"Out of Stock! Only {self.stock} {self.Product_name}(s) left.")
```

```
laptop=Product("Lenovo ideapad3",50000,25)
phone=Product("Realme C3",20000,5)
shoes=Product("Nike amad",5000,3)

laptop.showDetails()
phone.showDetails()
shoes.showDetails()

laptop.buyProduct(2)
phone.buyProduct(1)
shoes.buyProduct(4)
```

Output:-

```
Lenovo ideapad3 -$50000, Stock:25
Realme C3 -$20000, Stock:5
Nike amad -$5000, Stock:3
Purchased 2 x Lenovo ideapad3 = $100000
Remaining Stock of Lenovo ideapad3: 23
Purchased 1 x Realme C3 = $20000
Remaining Stock of Realme C3: 4
Out of Stock! Only 3 Nike amad(s) left.
```