Darshan UNIVERSITY
योग: कर्मसु कौशलम्

Python Programming - 2301CS404

Lab - 13

Jeet Bhalodi (23031701006)
03-04-2025

OOP

01) Write a Program to create a class by name Students, and initialize attributes like name, age, and grade while creating an object.

In [1]:
```
class Students:
    def __init__(self, name, age, grade):
        self.name = name
        self.age = age
        self.grade = grade

    def display_data(self):
        print(f"Name: {self.name}")
        print(f"Age: {self.age}")
        print(f"Grade: {self.grade}")

student1 = Students("Jeet", 20, "A++")

student1.display_data()
```

```
Name: Jeet
Age: 20
Grade: A++
```

02) Create a class named Bank_Account with Account_No, User_Name, Email,Account_Type and Account_Balance data members. Also create a method GetAccountDetails() and

DisplayAccountDetails(). Create main method to demonstrate the Bank_Account class.

In [11]:
```python
class Bank_Account:
    def __init__(self, Account_No, User_Name, Email, Account_Type, Account_Balance)
        self.Account_No = Account_No
        self.User_Name = User_Name
        self.Email = Email
        self.Account_Type = Account_Type
        self.Account_Balance = Account_Balance

    def GetAccountDetails(self):
        self.Account_No = input("Enter Account Number: ")
        self.User_Name = input("Enter User Name: ")
        self.Email = input("Enter Email: ")
        self.Account_Type = input("Enter Account Type (Savings/Current): ")
        self.Account_Balance = float(input("Enter Account Balance: "))

    def DisplayAccountDetails(self):
        print()
        print(f"Account Number: {self.Account_No}")
        print(f"User Name: {self.User_Name}")
        print(f"Email: {self.Email}")
        print(f"Account Type: {self.Account_Type}")
        print(f"Account Balance: {self.Account_Balance}")


account = Bank_Account("", "", "", "", 0.0)
account.GetAccountDetails()
account.DisplayAccountDetails()
```

```
Account Number: 8200695584
User Name: Jeet bhalodi
Email: bhavy@gmail.com
Account Type: Saving
Account Balance: 450000.0
```

03) WAP to create Circle class with area and perimeter function to find area and perimeter of circle.

In [ ]:
```python
import math

class Circle:
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return math.pi * (self.radius ** 2)

    def perimeter(self):
        return 2 * math.pi * self.radius

circle = Circle(5)
```

```
print(f"Area of the circle: {circle.area():.2f}")
print(f"Perimeter of the circle: {circle.perimeter():.2f}")
```

04) Create a class for employees that includes attributes such as name, age, salary, and methods to update and display employee information.

In [13]:
```
class Employee:
    def __init__(self, name, age, salary):
        self.name = name
        self.age = age
        self.salary = salary

    def update_info(self, name=None, age=None, salary=None):
        if name:
            self.name = name
        if age:
            self.age = age
        if salary:
            self.salary = salary

    def display_info(self):
        print(f"Name: {self.name}")
        print(f"Age: {self.age}")
        print(f"Salary: {self.salary}")

employee = Employee("Bhavy Bhalodi", 19, 50000)

print("Initial Employee Information:")
employee.display_info()

employee.update_info(name="Jeet Bhalodi", age=20, salary=95000)

print("\nUpdated Employee Information:")
employee.display_info()
```

```
Initial Employee Information:
Name: Bhavy Bhalodi
Age: 19
Salary: 50000

Updated Employee Information:
Name: Jeet Bhalodi
Age: 20
Salary: 95000
```

05) Create a bank account class with methods to deposit, withdraw, and check balance.

In [35]:
```
class BankAccount:
    def __init__(self, account_number, account_holder, balance=0):
        self.account_number = account_number
        self.account_holder = account_holder
        self.balance = balance
    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            print(f"Deposited {amount} successfully!")
        else:
            print("Deposit amount should be positive.")

    def withdraw(self, amount):
        if amount > 0:
            if self.balance >= amount:
                self.balance -= amount
                print(f"Withdrew {amount} successfully!")
            else:
                print("Insufficient balance.")
        else:
            print("Withdrawal amount should be positive.")

    def check_balance(self):
        print(f"Account Balance: {self.balance}")

account = BankAccount("8200695584", "Jeet Bhalodi", 250000)

account.check_balance()
account.deposit(50000)
account.check_balance()
account.withdraw(20000)
account.check_balance()
```

```
Account Balance: 250000
Deposited 50000 successfully!
Account Balance: 300000
Withdrew 20000 successfully!
Account Balance: 280000
```

06) Create a class for managing inventory that includes attributes such as item name, price, quantity, and methods to add, remove, and update items.

In [37]:
```
class Inventory:
    def __init__(self):
        self.items = []

    def add_item(self, item_name, price, quantity):
        self.items.append({"item_name": item_name, "price": price, "quantity": quan
        print(f"Added {item_name} successfully!")

    def remove_item(self, item_name):
        for item in self.items:
            if item["item_name"] == item_name:
                self.items.remove(item)
                print(f"Removed {item_name} successfully!")
                return
        print(f"Item {item_name} not found in inventory.")
    def update_item(self, item_name, price=None, quantity=None):
        for item in self.items:
            if item["item_name"] == item_name:
                if price is not None:
                    item["price"] = price
                if quantity is not None:
                    item["quantity"] = quantity
                print(f"Updated {item_name} successfully!")
                return
        print(f"Item {item_name} not found in inventory.")

    def display_inventory(self):
        if not self.items:
            print("Inventory is empty.")
        else:
            for item in self.items:
                print(f"Item: {item['item_name']}, Price: {item['price']}, Quantity

inventory = Inventory()

inventory.add_item("Laptop", 1500, 10)
inventory.add_item("Phone", 800, 20)

inventory.display_inventory()

inventory.update_item("Laptop", price=1400, quantity=8)

inventory.remove_item("Phone")

inventory.display_inventory()
```

```
Added Laptop successfully!
Added Phone successfully!
Item: Laptop, Price: 1500, Quantity: 10
Item: Phone, Price: 800, Quantity: 20
Updated Laptop successfully!
Removed Phone successfully!
Item: Laptop, Price: 1400, Quantity: 8
```

07) Create a Class with instance attributes of your choice.

In [39]:
```
class Book:
    def __init__(self, title, author, year_published, genre):
        self.title = title
        self.author = author
        self.year_published = year_published
        self.genre = genre

    def display_details(self):
        print(f"Title: {self.title}")
        print(f"Author: {self.author}")
        print(f"Year Published: {self.year_published}")
        print(f"Genre: {self.genre}")

    def update_info(self, title=None, author=None, year_published=None, genre=None)
```

```
        if title:
            self.title = title
        if author:
            self.author = author
        if year_published:
            self.year_published = year_published
        if genre:
            self.genre = genre
        print("Book information updated successfully!")

book = Book("1984", "George Orwell", 1949, "Dystopian")

print("Initial Book Details:")
book.display_details()

book.update_info(title="Animal Farm", year_published=1945)

print("\nUpdated Book Details:")
book.display_details()
```

```
Initial Book Details:
Title: 1984
Author: George Orwell
Year Published: 1949
Genre: Dystopian
Book information updated successfully!

Updated Book Details:
Title: Animal Farm
Author: George Orwell
Year Published: 1945
Genre: Dystopian
```

08) Create one class student_kit

Within the student_kit class create one class attribute principal name (Mr ABC)

Create one attendance method and take input as number of days.

While creating student take input their name .

Create one certificate for each student by taking input of number of days present in class.

In [41]:
```
class StudentKit:
    principal_name = "Mr. ABC"

    def __init__(self, student_name):
        self.student_name = student_name
        self.attendance_days = 0

    def record_attendance(self, days):
        self.attendance_days = days
    def generate_certificate(self):
        print(f"Certificate of Attendance")
        print(f"Principal: {StudentKit.principal_name}")
        print(f"Student Name: {self.student_name}")
        print(f"Days Present: {self.attendance_days} days")

student_name = input("Enter student name: ")
student = StudentKit(student_name)

days_present = int(input(f"Enter number of days {student_name} was present: "))
student.record_attendance(days_present)

student.generate_certificate()
```

```
Certificate of Attendance
Principal: Mr. ABC
Student Name: Jeet Bhalodi
Days Present: 300 days
```

09) Define Time class with hour and minute as data member. Also define addition method to add two time objects.

In [43]:

```
class Time:
    def __init__(self, hour, minute):
        self.hour = hour
        self.minute = minute

    def add_time(self, other_time):
        total_minutes = self.minute + other_time.minute
        total_hours = self.hour + other_time.hour + (total_minutes // 60)
        remaining_minutes = total_minutes % 60

        return Time(total_hours % 24, remaining_minutes)  # Assuming 24-hour format

    def display_time(self):
        print(f"{self.hour:02}:{self.minute:02}")

time1 = Time(2, 45)
time2 = Time(1, 30)

added_time = time1.add_time(time2)

print("First Time:")
time1.display_time()

print("Second Time:")
time2.display_time()

print("Added Time:")
added_time.display_time()
```

```
First Time:
02:45
Second Time:
01:30
Added Time:
04:15
```