Practical-17

May 19, 2025

0.1 Practical 17 :-

Name: Sarthak Sanay
Enrollment No: 230031101611051

0.1.1 Problem Statement 1:-

Inventory Management System:
A retail company needs a software solution to streamline inventory management. The system should enable employees to add new products to the inventory, specifying details such as the product name, price, and quantity available. Additionally, the system should support an operation to combine two products, calculating the average price and total quantity of the combined product.
Python Feature: This solution will utilize Python classes to represent products and operator overloading to perform calculations when combining products.

[2]:
```
class Product:
    def __init__(self, name, price, quantity):
        self.name = name
        self.price = price
        self.quantity = quantity

    def display(self):
        print(f"Product: {self.name}")
        print(f"Price:  {self.price}")
        print(f"Quantity: {self.quantity}\n")

    # Overload '+' to combine two products
    def __add__(self, other):
        if self.name != other.name:
            raise ValueError("Cannot combine products with different names.")

        total_quantity = self.quantity + other.quantity
        avg_price = (self.price * self.quantity + other.price * other.quantity)␣
 ↪/ total_quantity
        return Product(self.name, round(avg_price, 2), total_quantity)


print("Add two products to combine them:-\n")
```

```python
# First product
name1 = input("Enter product name: ")
price1 = float(input("Enter product price: "))
qty1 = int(input("Enter product quantity: "))
product1 = Product(name1, price1, qty1)

# Second product
name2 = input("\nEnter another product name (must be same to combine): ")
price2 = float(input("Enter product price: "))
qty2 = int(input("Enter product quantity: "))
product2 = Product(name2, price2, qty2)

# Display original products
print("\nProduct 1 :-")
product1.display()

print("Product 2 :-")
product2.display()

# Combine products
try:
    combined = product1 + product2
    print("Combined Product :-")
    combined.display()
except ValueError as e:
    print(f"Error: {e}")
```

```
Add two products to combine them:-


Enter product name:  Sony PlayStation 5
Enter product price:  56000
Enter product quantity:  12

Enter another product name (must be same to combine):  Sony PlayStation 5
Enter product price:  44000
Enter product quantity:  18


Product 1 :-
Product: Sony PlayStation 5
Price:  56000.0
Quantity: 12

Product 2 :-
Product: Sony PlayStation 5
Price:  44000.0
Quantity: 18
```

```
Combined Product :-
Product: Sony PlayStation 5
Price:  48800.0
Quantity: 30
```

0.1.2 Problem Statement 2:-

Student Record Management:
A school administration requires a digital tool to efficiently manage student records. The program should allow administrators to create individual student profiles, including information such as the student's name, age, and academic grades for different subjects. Moreover, the system should provide functionality to dynamically add new grades for specific subjects and retrieve grades for any subject as needed.
Python Feature: Python classes will be used to represent student profiles, with methods to add and retrieve grades dynamically.

[6]:
```python
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age
        self.grades = {}  # subject:grade pairs

    def add_grade(self, subject, grade):
        self.grades[subject] = grade
        print(f"Grade added: {subject} = {grade}")

    def get_grade(self, subject):
        return self.grades.get(subject, "Grade not found for this subject.")

    def display_profile(self):
        print(f"\nStudent Name: {self.name}")
        print(f"Age: {self.age}")
        print("Grades:")
        if self.grades:
            for subject, grade in self.grades.items():
                print(f"  {subject}: {grade}")
        else:
            print("  No grades available.")


print("Create a Student Profile:")
name = input("Enter student's name: ")
age = int(input("Enter student's age: "))

student = Student(name, age)
```

```
while True:
    print("\nOptions:")
    print("1. Add Grade")
    print("2. Get Grade for Subject")
    print("3. Display Student Profile")
    print("4. Exit")

    choice = input("Enter your choice (1-4): ")

    if choice == '1':
        subject = input("Enter subject name: ")
        grade = input("Enter grade: ")
        student.add_grade(subject, grade)

    elif choice == '2':
        subject = input("Enter subject name to retrieve grade: ")
        print(f"{subject}: {student.get_grade(subject)}")

    elif choice == '3':
        student.display_profile()

    elif choice == '4':
        print("Program terminated successfully!")
        break

    else:
        print("Invalid choice. Please enter a number from 1 to 4.")
```

```
Create a Student Profile:

Enter student's name:  Thomas Edison
Enter student's age:  21


Options:
1. Add Grade
2. Get Grade for Subject
3. Display Student Profile
4. Exit

Enter your choice (1-4):  1
Enter subject name:  DSA
Enter grade:  A

Grade added: DSA = A

Options:
1. Add Grade
2. Get Grade for Subject
```

```
3. Display Student Profile
4. Exit

Enter your choice (1-4):  1
Enter subject name:  OOPC
Enter grade:  B

Grade added: OOPC = B


Options:
1. Add Grade
2. Get Grade for Subject
3. Display Student Profile
4. Exit

Enter your choice (1-4):  1
Enter subject name:  OS
Enter grade:  A

Grade added: OS = A


Options:
1. Add Grade
2. Get Grade for Subject
3. Display Student Profile
4. Exit

Enter your choice (1-4):  2
Enter subject name to retrieve grade:  OOPC

OOPC: B


Options:
1. Add Grade
2. Get Grade for Subject
3. Display Student Profile
4. Exit

Enter your choice (1-4):  3


Student Name: Thomas Edison
Age: 21
Grades:
  DSA: A
  OOPC: B
  OS: A

Options:
1. Add Grade
2. Get Grade for Subject
```

```
3. Display Student Profile
4. Exit

Enter your choice (1-4):  4

Program terminated successfully!
```

0.1.3 Problem Statement 3:-

Bank Account Management:
A banking institution seeks a software solution to handle customer accounts effectively. The program should facilitate the creation and management of bank accounts, storing details such as the account number, current balance, and account holder's name. Furthermore, the system should support deposit operations by overloading the addition operator, enabling customers to deposit funds into their accounts seamlessly.
Python Feature: Python classes will be used to represent bank accounts, and operator overloading will be employed to enable deposit operations.

[1]:
```python
class BankAccount:
    def __init__(self, account_number, holder_name, balance=0.0):
        self.account_number = account_number
        self.holder_name = holder_name
        self.balance = balance

    def display_account(self):
        print(f"\nAccount Holder: {self.holder_name}")
        print(f"Account Number: {self.account_number}")
        print(f"Current Balance:  {self.balance:.2f}")

    # Overload '+' operator to perform deposit
    def __add__(self, amount):
        if isinstance(amount, (int, float)) and amount > 0:
            self.balance += amount
            print(f" {amount:.2f} deposited successfully.")
        else:
            print("Invalid deposit amount.")
        return self


print("Create a Bank Account:-\n")
acc_no = input("Enter account number: ")
name = input("Enter account holder name: ")
initial_balance = float(input("Enter initial balance: "))

account = BankAccount(acc_no, name, initial_balance)

while True:
    print("\nOptions:")
    print("1. Display Account")
```

```
    print("2. Deposit Amount")
    print("3. Exit")

    choice = input("Enter your choice (1-3): ")

    if choice == '1':
        account.display_account()

    elif choice == '2':
        amount = float(input("Enter amount to deposit: "))
        account + amount

    elif choice == '3':
        print("Program terminated successfully!")
        break

    else:
        print("Invalid choice. Please try again.")
```

```
Create a Bank Account:-


Enter account number:  007
Enter account holder name:  James Bond
Enter initial balance:  700


Options:
1. Display Account
2. Deposit Amount
3. Exit
Enter your choice (1-3):  1


Account Holder: James Bond
Account Number: 007
Current Balance:  700.00

Options:
1. Display Account
2. Deposit Amount
3. Exit
Enter your choice (1-3):  2
Enter amount to deposit:  1250
 1250.00 deposited successfully.

Options:
```

```
1. Display Account
2. Deposit Amount
3. Exit

Enter your choice (1-3):  1


Account Holder: James Bond
Account Number: 007
Current Balance:  1950.00

Options:
1. Display Account
2. Deposit Amount
3. Exit

Enter your choice (1-3):  3

Program terminated successfully!
```

0.1.4 Problem Statement 4:-

Social Media Account Management:
A social networking platform aims to develop a tool for managing user profiles efficiently. The program should allow users to create and customize their profiles with information like their username and a brief bio. Additionally, the system should enable users to gain followers, with functionality to increment the follower count dynamically as users interact with each other's profiles.
Python Feature: Python classes will represent user profiles, with methods to customize profiles and dynamically manage follower counts.

[4]:
```python
class UserProfile:
    def __init__(self, username, bio):
        self.username = username
        self.bio = bio
        self.followers = 0

    def update_bio(self, new_bio):
        self.bio = new_bio
        print(f"Bio updated to: {self.bio}")

    def add_follower(self):
        self.followers += 1
        print(f"{self.username} gained a follower! Total followers: {self.
 ↪followers}")

    def display_profile(self):
        print("\nUser Profile :-")
        print(f"Username: {self.username}")
        print(f"Bio: {self.bio}")
        print(f"Followers: {self.followers}")
```

```
# Example usage
user1 = UserProfile("james_bond", "Just another tech lover!")
user1.display_profile()
user1.add_follower()
user1.add_follower()
user1.update_bio("Python Developer | Open Source Enthusiast")
user1.display_profile()
```

```
User Profile :-
Username: james_bond
Bio: Just another tech lover!
Followers: 0
james_bond gained a follower! Total followers: 1
james_bond gained a follower! Total followers: 2
Bio updated to: Python Developer | Open Source Enthusiast

User Profile :-
Username: james_bond
Bio: Python Developer | Open Source Enthusiast
Followers: 2
```

0.1.5 Problem Statement 5:-

Shopping Cart Implementation:
An online marketplace requires a robust shopping cart system to enhance the shopping experience for its customers. The system should allow users to add items to their carts while browsing products, specifying details such as the item name, price, and desired quantity. Upon checkout, the system should display a summary of the items in the cart, including their respective details, facilitating a seamless shopping process.
Python Feature: Python classes will be used to represent items and the shopping cart, allowing for easy addition and retrieval of item details.

[6]:
```
class Product:
    def __init__(self, name, price):
        self.name = name
        self.price = price


class ShoppingCart:
    def __init__(self):
        self.items = []

    def add_item(self, product, quantity):
        self.items.append((product, quantity))
        print(f"Added {quantity}x {product.name} to the cart.")
```

```
    def checkout(self):
        print("\nShopping Cart Summary:-")
        total = 0
        for product, quantity in self.items:
            item_total = product.price * quantity
            print(f"{product.name} -  {product.price} x {quantity} =␣
 ↪{item_total}")
            total += item_total
        print(f"Total Amount:  {total}")


p1 = Product("Laptop", 50000)
p2 = Product("Mouse", 500)
p3 = Product("Keyboard", 1500)

cart = ShoppingCart()
cart.add_item(p1, 1)
cart.add_item(p2, 2)
cart.add_item(p3, 1)

cart.checkout()
```

```
Added 1x Laptop to the cart.
Added 2x Mouse to the cart.
Added 1x Keyboard to the cart.

Shopping Cart Summary:-
Laptop -  50000 x 1 =  50000
Mouse -  500 x 2 =  1000
Keyboard -  1500 x 1 =  1500
Total Amount:  52500
```

[ ]: