Name: S. SATHWIK

Batch-12

2403A51273

Lab-4

Task 1:

Write a class definition comment and start the constructor for a class called BankAccount with account_holder and balance attributes. Use GitHub Copilot to auto-complete the rest of the class, including methods to deposit, withdraw, and display balance.

Code:

```
class BankAccount:
    def __init__(self, account_holder, balance=0.0):
        self.account_holder = account_holder
        self.balance = balance

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            print(f"Deposited {amount}. New balance: {self.balance}")
        else:
            print("Deposit amount must be positive.")

    def withdraw(self, amount):
        if amount > 0:
            if amount <= self.balance:
                self.balance -= amount
                print(f"Withdrew {amount}. New balance: {self.balance}")
            else:
                print("Insufficient funds.")
```

```
        else:
            print("Withdrawal amount must be positive.")

    def display_balance(self):
        print(f"Account holder: {self.account_holder}")
        print(f"Current balance: {self.balance}")

if __name__ == "__main__":
    name = input("Enter account holder name: ")
    initial_balance = float(input("Enter initial balance: "))
    account = BankAccount(name, initial_balance)

    while True:
        print("\n1. Deposit\n2. Withdraw\n3. Display Balance\n4. Exit")
        choice = input("Choose an option: ")
        if choice == '1':
            amount = float(input("Enter amount to deposit: "))
            account.deposit(amount)
        elif choice == '2':
            amount = float(input("Enter amount to withdraw: "))
            account.withdraw(amount)
        elif choice == '3':
            account.display_balance()
        elif choice == '4':
            print("Exiting...")
            break
        else:
            print("Invalid option. Please try again.")
```

output:

Enter account holder name: abhi

Enter initial balance: 200000

1. Deposit

2. Withdraw

3. Display Balance

Choose an option: 2

Enter amount to withdraw: 20000

Withdrew 20000.0. New balance: 180000.0

1. Deposit

2. Withdraw

3. Display Balance

4. Exit

Choose an option: 4

Exiting...

2.task

Write a comment and the initial line of a loop to iterate over a list. Allow GitHub Copilot to complete the logic to sum all even numbers in the list.

Code:

```
my_list = [1, 2, 3, 4, 5, 6]  # Sample input
total = 0
for num in my_list:
    if num % 2 == 0:
        total += num
print("Sum of even numbers:", total)
```

output:

Sum of even numbers: 12

3.task

Start a function that takes age as input and returns whether the person is a child, teenager, adult, or senior using if-elif-else. Use Copilot to complete the conditionals.

Code:

```
my_list = [1, 2, 3, 4, 5, 6]  # Sample input

total = 0

for num in my_list:

   if num % 2 == 0:

      total += num

print("Sum of even numbers:", total)


# Function to determine age group

def age_group(age):

   if age < 13:

      return "Child"

   elif age < 20:

      return "Teenager"

   elif age < 60:

      return "Adult"

   else:

      return "Senior"
```

output:

(45) → Adult

4.task

Write a comment and start a while loop to reverse the digits of a number. Let Copilot complete the loop logic.

Code:

```
# Ask user for input and reverse the digits of the number

num = int(input("Enter a number to reverse: "))

rev = 0
```

```
# Start a while loop to reverse the digits
while num > 0:
    rev = rev * 10 + num % 10
    num = num // 10
print("Reversed number:", rev)
```

output:

Enter a number to reverse: 1234

Reversed number: 4321

5.task

Begin a class Employee with attributes name and salary. Then, start a derived class Manager that inherits from Employee and adds department. Let GitHub Copilot complete the methods and constructor chaining

Code:

```
class Employee:
    def _init_(self, name, salary):
        self.name = name
        self.salary = salary

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



class Manager(Employee):
    def _init_(self, name, salary, department):
        super()._init_(name, salary)
        self.department = department

    def display_info(self):
        super().display_info()
        print(f"Department: {self.department}")
```

```
# Taking input from the user
name = input("Enter the employee's name: ")
salary = float(input("Enter the employee's salary: "))
department = input("Enter the manager's department: ")

# Creating a Manager object
manager = Manager(name, salary, department)

# Displaying the manager's information
manager.display_info()
```

output:

```
Enter the employee's name: sathwik
Enter the employee's salary: 50000
Enter the manager's department: 10
Name: sathwik, Salary: 50000.0
Department: 10
```