LAB NAME : AI ASSISTED CODING

LAB NUMBER :04

ROLL NO :2503A51L16

BRANCH : CSE

NAME : K. JASHUVA

TASK 1

Task Description: 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.

PROMPT: Create a Python class named BankAccount with attributes account_holder and balance.Implement a constructor to initialize these attributes.

deposit(amount): adds the amount to the balance. withdraw(amount): subtracts the amount if sufficient balance exists. display_balance(): prints the account holder's name and current balance

CODE:

```python
# Class to represent a bank account with basic operations
class BankAccount:
    def __init__(self, account_holder, balance=0):
        """
        Initializes a new BankAccount instance.

        Args:
            account_holder (str): The name of the account holder.
            balance (float): The initial balance of the account (default is 0).
        """
        self.account_holder = account_holder
        self.balance = balance

    def deposit(self, amount):
        """Deposits the specified amount into the account."""
        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):
        """Withdraws the specified amount from the account if sufficient funds exist."""
        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):
        """Displays the current balance of the account."""
        print(f"Account holder: {self.account_holder}, Balance: {self.balance}")

# Example usage
```

```python
# Example usage
if __name__ == "__main__":
    account = BankAccount("Alice", 1000)
    account.display_balance()
    account.deposit(500)
```

OUTPUT:

```
PS C:\Users\saian> & C:/Users/saian/anaconda3/pyth
ab 4/lab 4.1.py"
Account holder: Alice, Balance: 1000
Deposited 500. New balance: 1500
```

TASK 2

Task Description: 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.

PROMPT: Write Python code to iterate over a list of integers and calculate the sum of all even numbers. Start by defining a list called 'numbers' with some sample values.Use a loop to check each number and add it to a running total if it's even. Finally, print the total sum of even numbers.

CODE:

```
# Iterate over a list and sum all even numbers
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_sum = 0
for num in numbers:
    if num % 2 == 0:
        even_sum += num
print(f"Sum of even numbers: {even_sum}")
```

OUTPUT:

```
PS C:\Users\saian> & C:/Users/sai
ab 4/lab 4.2.py"
Sum of even numbers: 30
```

TASK 3

Task Description: 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.

PROMPT: # Write a Python function called classify_age that takes an integer 'age' as input.

 # Use if-elif-else statements to return one of the following strings based on the age:

 # - "Child" if age is 12 or below

 # - "Teenager" if age is between 13 and 19

 # - "Adult" if age is between 20 and 64.

#- "Senior" if age is 65 or above

CODE:

```python
# Function to determine age group based on age
def age_group(age):
    if age < 13:
        return "Child"
    elif age < 20:
        return "Teenager"
    elif age < 60:
        return "Adult"
    else:
        return "Senior"

# Example usage
if __name__ == "__main__":
    print(age_group(45))  # Output: Adult
    print(age_group(10))  # Output: Child
    print(age_group(17))  # Output: Teenager
    print(age_group(65))  # Output: Senior
```

OUTPUT:

```
PS C:\Users\salan> & C:/Users/sal
ab 4/lab 4.3.py"
Adult
Child
Teenager
Senior
```

TASK 4

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

PROMPT: Write Python code to reverse the digits of a given positive integer using a while loop.Start by initializing a variable 'num' with a sample value Use a while loop to extract digits and build the reversed number.Print the final reversed number.

num = 12345

CODE:

```python
# Reverse the digits of a number using a while loop
number = 1234
reversed_num = 0

while number > 0:
    digit = number % 10           # Get the last digit
    reversed_num = reversed_num * 10 + digit  # Append digit
    number = number // 10         # Remove the last digit

print(reversed_num)  # Output: 4321
```

OUTPUT:

```
PS C:\Users\saian> & C:/Users/saian/a
ab 4/lab 4.4.py"
4321
```

TASK 5

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

PROMPT: Create a Python class called `Employee` with attributes `name` and `salary`. Define an `__init__` method to initialize these attributes. Then, create a derived class called `Manager` that inherits from `Employee` and adds an additional attribute `department`. Use constructor chaining to initialize the base class attributes from the derived class. Add a method in each class to display the details of the employee or manager. Complete the implementation.

CODE:

```python
# Base class: Employee
class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

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

# Derived class: Manager inherits from Employee
class Manager(Employee):
    def __init__(self, name, salary, department):
        super().__init__(name, salary)  # Call the base class constructor
        self.department = department

    def display_info(self):
        super().display_info()  # Display name and salary from Employee
        print(f"Department: {self.department}")

# Example usage
emp1 = Employee("Amit", 50000)
mgr1 = Manager("Priya", 80000, "Marketing")

print("Employee Info:")
emp1.display_info()

print("\nManager Info:")
mgr1.display_info()
```

OUTPUT:

```
PS C:\Users\saian> & C:/Users/saian/anaconda3/python.exe  d:/s
Employee Info:
Name: Amit
Salary: ₹50000

Manager Info:
Name: Priya
Salary: ₹80000
Department: Marketing
PS C:\Users\saian>
```