

LAB NAME : AI ASSISTED CODING

LAB NUMBER :04

ROLL NO :2503A51L31

BRANCH : CSE

NAME : ch.abishek

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:

```
1  # Class to represent a bank account with basic operations
2  class BankAccount:
3      def __init__(self, account_holder, balance=0):
4          """
5          Initializes a new BankAccount instance.
6
7          Args:
8              account_holder (str): The name of the account holder.
9              balance (float): The initial balance of the account (default is 0).
10         """
11         self.account_holder = account_holder
12         self.balance = balance
13
14     def deposit(self, amount):
15         """Deposits the specified amount into the account."""
16         if amount > 0:
17             self.balance += amount
18             print(f"Deposited {amount}. New balance: {self.balance}")
19         else:
20             print("Deposit amount must be positive.")
21
22     def withdraw(self, amount):
23         """Withdraws the specified amount from the account if sufficient funds exist."""
24         if amount > 0:
25             if amount <= self.balance:
26                 self.balance -= amount
27                 print(f"Withdrew {amount}. New balance: {self.balance}")
28             else:
29                 print("Insufficient funds.")
30         else:
31             print("Withdrawal amount must be positive.")
32
33     def display_balance(self):
34         """Displays the current balance of the account."""
35         print(f"Account holder: {self.account_holder}, Balance: {self.balance}")
36
37 # Example usage
```

```
37  # Example usage
38  if __name__ == "__main__":
39      account = BankAccount("Alice", 1000)
40      account.display_balance()
41      account.deposit(500)
```

OUTPUT:

```
PS C:\Users\saian> & C:/Users/saian/anaconda3/python
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:

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

OUTPUT:

```
PS C:\Users\sai> & 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:

```

1  # Function to determine age group based on age
2  def age_group(age):
3      if age < 13:
4          return "Child"
5      elif age < 20:
6          return "Teenager"
7      elif age < 60:
8          return "Adult"
9      else:
10         return "Senior"
11
12  # Example usage
13  if __name__ == "__main__":
14      print(age_group(45)) # Output: Adult
15      print(age_group(10)) # Output: Child
16      print(age_group(17)) # Output: Teenager
17      print(age_group(65)) # Output: Senior

```

OUTPUT:

```

PS C:\Users\Sahar> & C:/Users/sahar/OneDrive/Desktop/lab 4/lab 4.3.py"
Adult
Child
Teenager
Senior

```

TASK 4

TASK DESCRIPTION: 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:

```
1 # Reverse the digits of a number using a while loop
2 number = 1234
3 reversed_num = 0
4
5 while number > 0:
6     digit = number % 10          # Get the last digit
7     reversed_num = reversed_num * 10 + digit # Append digit
8     number = number // 10       # Remove the last digit
9
10 print(reversed_num) # Output: 4321
```

OUTPUT:

```
PS C:\Users\saian> & C:/Users/saian/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:

```
# 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\saihan> & C:/Users/saihan/anaconda3/python.exe -d:/S
Employee Info:
Name: Amit
Salary: ₹50000

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

