

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: M.Tech. and MCA		Assignment Type: Lab	AcademicYear:2025-2026
Course Coordinator Name		Venkataramana Veeramsetty	
Course Code		Course Title	AI Assisted Problem Solving Using Python
Year/Sem	I/I	Regulation	R24
Date and Day of Assignment	Week3 - Monday	Time(s)	
Duration	2 Hours	Applicable to Batches	M.Tech. and MCA
AssignmentNumber:6.3(Present assignment number)/24(Total number of assignments)			
<p style="text-align: center;">ASSIGNMENT – 6</p> <p style="text-align: center;">Name: Rimsha Mujeeb Roll Number:2503B05138 (M.Tech - CSE)</p>			
Q.No.	Question		
Q1.	<p>Task Description#1 (Classes)</p> <ul style="list-style-type: none"> • Use AI to complete a Student class with attributes and a method. • Check output • Analyze the code generated by AI tool <p>Expected Output#1</p> <ul style="list-style-type: none"> • Class with constructor and display_details() method <p>PROMPT:</p> <pre># Create a Python class named Student. # Include attributes for name, roll number, and grade. # Add a constructor to initialize these attributes. # Implement a method display_details() to print all the details.</pre> <p>CODE ANALYSIS:</p> <ol style="list-style-type: none"> 1. Constructor __init__ initializes ID, name, age, and grade. 2. display_details() method prints details using f-strings. 3. Main block takes user input and creates a Student object. 4. Displays the entered student details clearly. 		

5. Demonstrates basic OOP concepts — class, object, and method.
6. Code is simple, readable, and effective.
7. Can be improved with input validation and multiple records support.

```

1  class Student:
2      def __init__(self, student_id, name, age, grade):
3          """Initialize student attributes"""
4          self.student_id = student_id
5          self.name = name
6          self.age = age
7          self.grade = grade
8
9      def display_details(self):
10         """Display student information"""
11         print(f"Student ID: {self.student_id}")
12         print(f"Name: {self.name}")
13         print(f"Age: {self.age}")
14         print(f"Grade: {self.grade}")
15
16     if __name__ == "__main__":
17         # Get input from user
18         student_id = input("Enter student ID: ")
19         name = input("Enter student name: ")
20         age = int(input("Enter student age: "))
21         grade = input("Enter student grade: ")
22
23         # Create a student object with user input
24         student1 = Student(student_id, name, age, grade)
25
26         # Display student details
27         print("\nStudent Details:")
28         student1.display_details()

```

```

PS C:\Users\HP\Desktop\Mtech\AIPP\ASSIGNMENT-6> & "C:/Program Files/Python313/python.exe" c:/Users/HP/Desktop/Mtech/AIPP/ASSIGNMENT-6/Q1.py
Enter student ID: 101
Enter student name: Rimsha
Enter student age: 22
Enter student grade: B.Tech

Student Details:
Student ID: 101
Name: Rimsha
Age: 22
Grade: B.Tech
PS C:\Users\HP\Desktop\Mtech\AIPP\ASSIGNMENT-6>

```

Q2.

Task Description#2 (Loops)

- Prompt AI to complete a function that prints the first 10 multiples of a number using a loop.
- Analyze the generated code
- Ask AI to generate code using other controlled looping

Expected Output#2

Correct loop-based implementation

PROMPT :

- 1- Write a Python function that takes a number as input and prints its first 10 multiples using a loop.

- 2- Generate an alternative implementation using a while loop. Include comments explaining the code.(modify the given program without changing previous one)

```
File Edit Selection View Go Run Terminal Help
Q1.py Q2.py
Q2.py > print_multiples_for
1 # Q2.py
2 # Task: print the first 10 multiples of a number using loops
3 # Contains two controlled-loop implementations and a short analysis.
4
5 def print_multiples_for(n):
6     """Print first 10 multiples of n using a for loop."""
7     for i in range(1, 11):
8         print(f"{n} x {i} = {n * i}")
9
10 def print_multiples_while(n):
11     """Print first 10 multiples of n using a while loop (alternative)."""
12     i = 1
13     while i <= 10:
14         print(f"{n} x {i} = {n * i}")
15         i += 1
16
17 # Analysis (brief):
18 # - Both functions produce the same output: multiples 1..10 of the given n.
19 # - for loop version is concise and idiomatic for a known fixed range.
20 # - while loop version demonstrates manual counter control.
21 # - Time complexity: O(k) where k=10 (constant), effectively O(1) for this fixed task.
22
23 if __name__ == "__main__":
24     try:
25         num = int(input("Enter an integer to print its first 10 multiples: ").strip())
26     except ValueError:
27         print("Invalid input. Please enter an integer.")
28     else:
29         print("\nUsing for loop:")
30         print_multiples_for(num)
31         print("\nUsing while loop:")
32         print_multiples_while(num)
33
34 # If you want another controlled-loop variant (e.g., recursion, generator, itertools),
35 # ask to generate that alternative implementation.
```

```
PS C:\Users\HP\Desktop\Mtech\AIPP\ASSIGNMENT-6> "C:/Program Files/Python313/python.exe" c:/Users/HP/Desktop/Mtech/AIPP/ASSIGNMENT-6/Q2.py
Enter an integer to print its first 10 multiples: 5

Using for loop:
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50

Using while loop:
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
PS C:\Users\HP\Desktop\Mtech\AIPP\ASSIGNMENT-6>
```

Q3.

Task Description#3 (Conditional Statements)

- Ask AI to write nested if-elif-else conditionals to classify age groups.
- Analyze the generated code
- Ask AI to generate code using other conditional statements

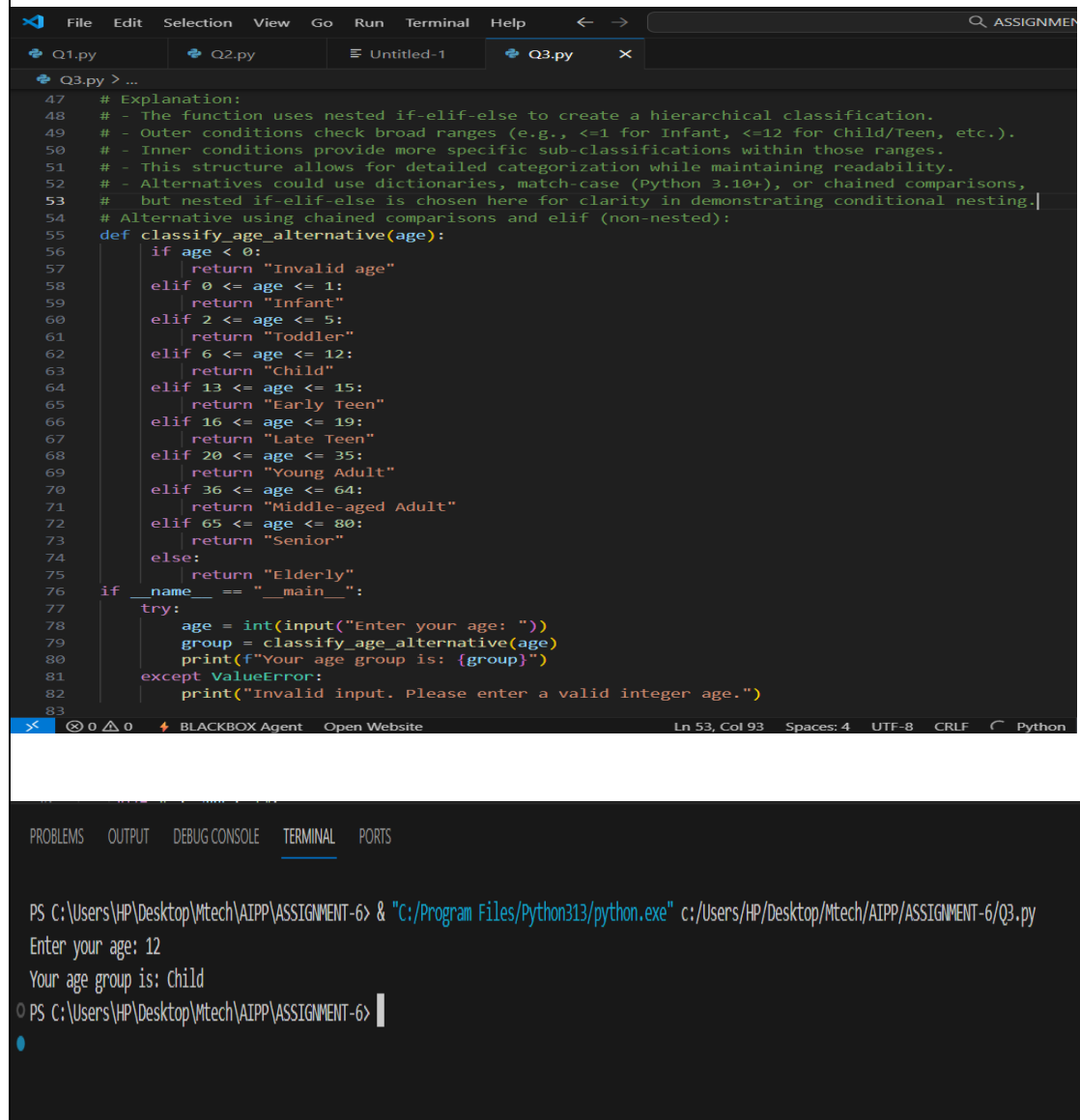
Expected Output#3

- Age classification function with appropriate conditions and with explanation

```
File Edit Selection View Go Run Terminal Help ← → Q ASSIGNMEN
Q1.py Q2.py Untitled-1 Q3.py X
Q3.py > classify_age
1 # Age Classification Function using Nested If-Elif-Else
2 # This function classifies a person's age into different groups based on standard age ranges.
3 # It uses nested conditional statements to handle the classification logic.
4
5 def classify_age(age):
6     """
7     Classifies the given age into age groups using nested if-elif-else statements.
8     Parameters:
9     age (int): The age of the person (must be a non-negative integer).
10    Returns:
11    str: The age group classification.
12    """
13    if age < 0:
14        return "Invalid age"
15    elif age <= 1:
16        return "Infant"
17    elif age <= 12:
18        if age <= 5:
19            return "Toddler"
20        else:
21            return "Child"
22    elif age <= 19:
23        if age <= 15:
24            return "Early Teen"
25        else:
26            return "Late Teen"
27    elif age <= 64:
28        if age <= 35:
29            return "Young Adult"
30        else:
31            return "Middle-aged Adult"
32    else:
33        if age <= 80:
34            return "Senior"
35        else:
36            return "Elderly"
37
Ln 9, Col 71 Spaces: 4 UTF-8 CRLF ( ) Python
```

```
File Edit Selection View Go Run Terminal Help ← →
Q1.py Q2.py Untitled-1 Q3.py X
Q3.py > classify_age
5 def classify_age(age):
19     return "Toddler"
20     else:
21         return "Child"
22 elif age <= 19:
23     if age <= 15:
24         return "Early Teen"
25     else:
26         return "Late Teen"
27 elif age <= 64:
28     if age <= 35:
29         return "Young Adult"
30     else:
31         return "Middle-aged Adult"
32 else:
33     if age <= 80:
34         return "Senior"
35     else:
36         return "Elderly"
37
38 # Example usage: Ask user for their age and display the age group
39 if __name__ == "__main__":
40     try:
41         age = int(input("Enter your age: "))
42         group = classify_age(age)
43         print(f"Your age group is: {group}")
44     except ValueError:
45         print("Invalid input. Please enter a valid integer age.")
46
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Enter your age: 23
Your age group is: Young Adult
PS C:\Users\HP\Desktop\Mtech\AIPP\ASSIGNMENT-6>
Ln 9, Col 71 Space
```

ALTERNATIVE CODE:



The image shows a code editor window with a dark theme. The editor has tabs for Q1.py, Q2.py, Untitled-1, and Q3.py. The Q3.py tab is active, showing a Python script. The script includes a multi-line comment explaining the use of nested if-elif-else for hierarchical classification. The function `classify_age_alternative(age)` uses a series of elif statements to categorize ages into groups like 'Invalid age', 'Infant', 'Toddler', 'Child', 'Early Teen', 'Late Teen', 'Young Adult', 'Middle-aged Adult', 'Senior', and 'Elderly'. Below the function, a main block uses `__name__ == '__main__':` to prompt the user for their age, call the function, and print the result. It also includes a `try/except` block to handle `ValueError` exceptions for invalid input. The status bar at the bottom indicates 'Ln 53, Col 93', 'Spaces: 4', 'UTF-8', 'CRLF', and 'Python'.

```
47 # Explanation:
48 # - The function uses nested if-elif-else to create a hierarchical classification.
49 # - Outer conditions check broad ranges (e.g., <=1 for Infant, <=12 for Child/Teen, etc.).
50 # - Inner conditions provide more specific sub-classifications within those ranges.
51 # - This structure allows for detailed categorization while maintaining readability.
52 # - Alternatives could use dictionaries, match-case (Python 3.10+), or chained comparisons,
53 #   but nested if-elif-else is chosen here for clarity in demonstrating conditional nesting.
54 # Alternative using chained comparisons and elif (non-nested):
55 def classify_age_alternative(age):
56     if age < 0:
57         return "Invalid age"
58     elif 0 <= age <= 1:
59         return "Infant"
60     elif 2 <= age <= 5:
61         return "Toddler"
62     elif 6 <= age <= 12:
63         return "Child"
64     elif 13 <= age <= 15:
65         return "Early Teen"
66     elif 16 <= age <= 19:
67         return "Late Teen"
68     elif 20 <= age <= 35:
69         return "Young Adult"
70     elif 36 <= age <= 64:
71         return "Middle-aged Adult"
72     elif 65 <= age <= 80:
73         return "Senior"
74     else:
75         return "Elderly"
76 if __name__ == "__main__":
77     try:
78         age = int(input("Enter your age: "))
79         group = classify_age_alternative(age)
80         print(f"Your age group is: {group}")
81     except ValueError:
82         print("Invalid input. Please enter a valid integer age.")
83
```

Below the code editor is a terminal window. The terminal shows the command to run the script: `PS C:\Users\HP\Desktop\Mtech\AIPP\ASSIGNMENT-6> & "C:/Program Files/Python313/python.exe" c:/Users/HP/Desktop/Mtech/AIPP/ASSIGNMENT-6/Q3.py`. The output shows the prompt 'Enter your age: 12' and the response 'Your age group is: Child'. The terminal prompt is now `PS C:\Users\HP\Desktop\Mtech\AIPP\ASSIGNMENT-6>`.

Q4.

Task Description#4 (For and While loops)

- Generate a `sum_to_n()` function to calculate sum of first n numbers
- Analyze the generated code
- Get suggestions from AI with other controlled looping

Expected Output#4

- Python code with explanation

```
File Edit Selection View Go Run Terminal Help ← → Q ASSIGNME
Q1.py Q2.py Q4.py X
Q4.py > sum_to_n_for
1 # Sum to N Function using For Loop
2 # This function calculates the sum of the first n natural numbers using a for loop.
3
4 def sum_to_n_for(n):
5     """
6     Calculates the sum of the first n natural numbers using a for loop.
7
8     Parameters:
9     n (int): A positive integer representing the number up to which to sum.
10
11     Returns:
12     int: The sum of the first n natural numbers.
13     """
14     if n < 1:
15         return 0
16     total = 0
17     for i in range(1, n + 1):
18         total += i
19     return total
20
21 # Sum to N Function using While Loop
22 # This function calculates the sum of the first n natural numbers using a while loop.
23
24 def sum_to_n_while(n):
25     """
26     Calculates the sum of the first n natural numbers using a while loop.
27
28     Parameters:
29     n (int): A positive integer representing the number up to which to sum.
30
31     Returns:
32     int: The sum of the first n natural numbers.
33     """
34     if n < 1:
35         return 0
36     total = 0
37     i = 1
```

```
File Edit Selection View Go Run Terminal Help ← → Q ASSIGNMENT-6
Q1.py Q2.py Q4.py X
Q4.py > sum_to_n_for
24 def sum_to_n_while(n):
38     while i <= n:
39         total += i
40         i += 1
41     return total
42
43 # Mathematical Formula Approach (for comparison)
44 # This is not a loop but uses the formula n*(n+1)/2 for efficiency.
45
46 def sum_to_n_formula(n):
47     """
48     Calculates the sum of the first n natural numbers using the mathematical formula.
49
50     Parameters:
51     n (int): A positive integer representing the number up to which to sum.
52
53     Returns:
54     int: The sum of the first n natural numbers.
55     """
56     if n < 1:
57         return 0
58     return n * (n + 1) // 2
59
60 # Example usage
61 if __name__ == "__main__":
62     n = 10
63     print(f"Sum of first {n} numbers using for loop: {sum_to_n_for(n)}")
64     print(f"Sum of first {n} numbers using while loop: {sum_to_n_while(n)}")
65     print(f"Sum of first {n} numbers using formula: {sum_to_n_formula(n)}")
66
67 # Explanation:
68 # - The for loop version iterates from 1 to n, accumulating the sum in a variable.
69 # - The while loop version uses a counter variable to achieve the same result.
70 # - Both loop-based functions have O(n) time complexity, while the formula has O(1) time complexity.
71 # - The formula is the most efficient for large n, but loops demonstrate iterative control structures.
72 # - Input validation ensures n is at least 1; otherwise, returns 0.
73
```

```

66
67 # Explanation:
68 # - The for loop version iterates from 1 to n, accumulating the sum in a variable.
69 # - The while loop version uses a counter variable to achieve the same result.
70 # - Both loop-based functions have O(n) time complexity, while the formula has O(1) time complexity.
71 # - The formula is the most efficient for large n, but loops demonstrate iterative control structures.
72 # - Input validation ensures n is at least 1; otherwise, returns 0.
73
74 # Suggestions for other controlled looping:
75 # 1. Use recursion: def sum_to_n_recursive(n): return n + sum_to_n_recursive(n-1) if n > 0 else 0
76 # 2. Use list comprehension with sum: def sum_to_n_list(n): return sum([i for i in range(1, n+1)])
77 # 3. Use generator expression: def sum_to_n_gen(n): return sum(i for i in range(1, n+1))
78 # 4. Use reduce from functools: from functools import reduce; def sum_to_n_reduce(n): return reduce(lambda x, y: x + y, range(1, n+1), 0)
79 # These alternatives vary in readability, efficiency, and Pythonic style.

```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

```

● PS C:\Users\HP\Desktop\Mtech\AIPP\ASSIGNMENT-6> & "C:/Program Files/Python313/python.exe" c:/Users/HP/Desktop/Mtech/AIPP/ASSIGNMENT-6/Q4.py
Sum of first 10 numbers using for loop: 55
Sum of first 10 numbers using while loop: 55
● Sum of first 10 numbers using formula: 55
○ PS C:\Users\HP\Desktop\Mtech\AIPP\ASSIGNMENT-6>

```

ANALYZING CODE:

1. Sum_to_n_for(n) – uses a for loop to iterate from 1 to n and accumulate the sum.
2. Sum_to_n_while(n) – performs the same task using a while loop with a counter variable.
3. Sum_to_n_formula(n) – applies the mathematical formula $n*(n+1)//2$, which is the most efficient ($O(1)$ time).
4. All functions check if n is less than 1 and return 0 in that case.
5. In the main section, the code demonstrates each method for $n = 10$ and prints the results.

Overall, it effectively compares iterative and formula-based approaches, showing how different control structures can solve the same problem with varying efficiency.

Task Description#5 (Class)

- Use AI to build a BankAccount class with deposit, withdraw, and balance methods.
- Analyze the generated code
- Add comments and explain code

Expected Output#5

- Python code with explanation

PROMPT:

A simple BankAccount class that allows depositing money, withdrawing money, and checking the current balance.

Q5.

```
File Edit Selection View Go Run Terminal Help ← → Q ASSIG
Q1.py Q2.py Q4.py Q5.py X
Q5.py > BankAccount > _init_
1 class BankAccount:
2     """
3     A simple BankAccount class that allows depositing money, withdrawing money,
4     and checking the current balance.
5     """
6     def __init__(self):
7         """
8         Initializes a new BankAccount instance with a balance of 0.
9         """
10        self._balance = 0.0 # Private attribute to store the account balance
11    def deposit(self, amount):
12        """
13        Deposits a specified amount into the account.
14        Args:
15            amount (float): The amount to deposit. Must be positive.
16        Raises:
17            ValueError: If the amount is not positive.
18        """
19        if amount <= 0:
20            raise ValueError("Deposit amount must be positive.")
21        self._balance += amount # Add the amount to the balance
22    def withdraw(self, amount):
23        """
24        Withdraws a specified amount from the account if sufficient funds are available.
25        Args:
26            amount (float): The amount to withdraw. Must be positive.
27        Raises:
28            ValueError: If the amount is not positive or if there are insufficient funds.
29        """
30        if amount <= 0:
31            raise ValueError("Withdrawal amount must be positive.")
32        if amount > self._balance:
33            raise ValueError("Insufficient funds.")
34        self._balance -= amount # Subtract the amount from the balance
35
36    def balance(self):
37        """
```

```
File Edit Selection View Go Run Terminal Help ← → Q ASSIGNME
Q1.py Q2.py Q4.py Q5.py X
Q5.py > BankAccount > _init_
1 class BankAccount:
35
36    def balance(self):
37        """
38        Returns the current balance of the account.
39
40        Returns:
41            float: The current account balance.
42        """
43        return self._balance # Return the private balance attribute
44
45
46 # Test the BankAccount class with user input
47 if __name__ == "__main__":
48     account = BankAccount()
49     print("Welcome to BankAccount Tester!")
50     print(f"Initial balance: {account.balance()}")
51
52     while True:
53         action = input("Enter 'deposit', 'withdraw', 'balance', or 'quit': ").strip().lower()
54         if action == 'quit':
55             break
56         elif action == 'balance':
57             print(f"Current balance: {account.balance()}")
58         elif action == 'deposit':
59             try:
60                 amount = float(input("Enter deposit amount: "))
61                 account.deposit(amount)
62                 print(f"Deposited {amount}. New balance: {account.balance()}")
63             except ValueError as e:
64                 print(f"Error: {e}")
65         elif action == 'withdraw':
66             try:
67                 amount = float(input("Enter withdrawal amount: "))
68                 account.withdraw(amount)
69                 print(f"Withdrew {amount}. New balance: {account.balance()}")
70             except ValueError as e:
```



```
File Edit Selection View Go Run Terminal Help
Q1.py Q2.py Q4.py Q5.py X
Q5.py > BankAccount > _init_
56 elif action == 'balance':
57     print(f"Current balance: {account.balance()}")
58 elif action == 'deposit':
59     try:
60         amount = float(input("Enter deposit amount: "))
61         account.deposit(amount)
62         print(f"Deposited {amount}. New balance: {account.balance()}")
63     except ValueError as e:
64         print(f"Error: {e}")
65 elif action == 'withdraw':
66     try:
67         amount = float(input("Enter withdrawal amount: "))
68         account.withdraw(amount)
69         print(f"Withdrew {amount}. New balance: {account.balance()}")
70     except ValueError as e:
71         print(f"Error: {e}")
72 else:
73     print("Invalid action. Try again.")
74
```

```
File Edit Selection View Go Run Terminal Help
Q1.py Q2.py Q4.py Q5.py X
Q5.py > BankAccount
1 class BankAccount:
4     ...and checking the current balance. Chat (CTRL + I) / Share (CTRL + L)
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\HP\Desktop\Mtech\AIPP\ASSIGNMENT-6> & "C:/Program Files/Python313/python.exe" c:/Users/HP/Desktop/Mtech/AIPP/ASSIGNMENT-6/Q5.py
Welcome to BankAccount Tester!
Initial balance: 0.0
Enter 'deposit', 'withdraw', 'balance', or 'quit': DEPOSIT
Enter deposit amount: 20000
Deposited 20000.0. New balance: 20000.0
Enter 'deposit', 'withdraw', 'balance', or 'quit': DEPOSIT
Enter deposit amount: 14000
Deposited 14000.0. New balance: 34000.0
Enter 'deposit', 'withdraw', 'balance', or 'quit': WITHDRAW
Enter withdrawal amount: 25000
Withdrew 25000.0. New balance: 9000.0
Enter 'deposit', 'withdraw', 'balance', or 'quit': BALANCE
Current balance: 9000.0
Enter 'deposit', 'withdraw', 'balance', or 'quit': QUIT
PS C:\Users\HP\Desktop\Mtech\AIPP\ASSIGNMENT-6>
```

