Task Description #1 (AI-Based Code Completion for Loops)

Task: Use an AI code completion tool to generate a loop-based program.

Prompt:

"Generate Python code to print all even numbers between 1 and N using a loop."

Expected Output:

• AI-generated loop logic.

• Identification of loop type used (for or while).

• Validation with sample inputs

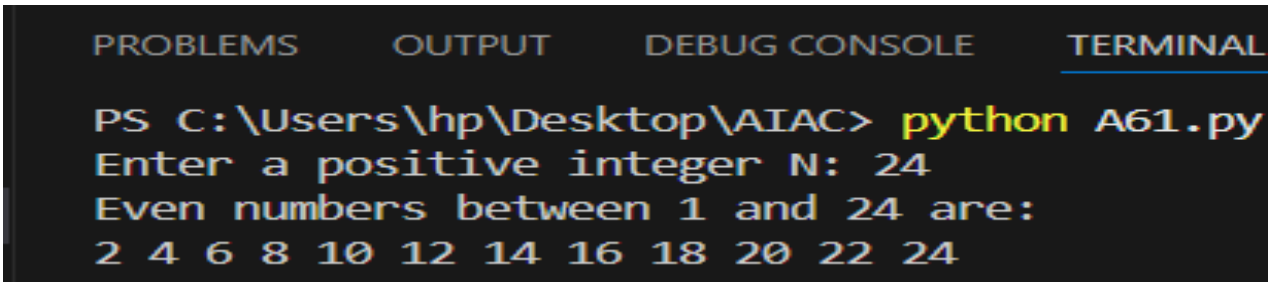
CODE:

A61.py > ...

```python
# Generate Python code to print all even numbers between 1 and N using a loop.
# Take N as user input.
# Use an efficient loop and proper condition checking.
def print_even_numbers():
    try:
        N = int(input("Enter a positive integer N: "))
        if N < 1:
            print("Please enter a positive integer greater than 0.")
            return

        print(f"Even numbers between 1 and {N} are:")
        for number in range(2, N + 1, 2):
            print(number, end=' ')
        print()  # for a new line after printing all even numbers
    except ValueError:
        print("Invalid input. Please enter a valid positive integer.")
print_even_numbers()
```

OUTPUT:



ANALYSIS:

Task 1: Loop – Even Numbers:

The program uses a loop to iterate from 1 to N and applies a conditional check to identify even numbers.
It demonstrates efficient looping logic and basic conditional filtering.

Task Description #2 (AI-Based Code Completion for Loop with Conditionals)

Task: Use an AI code completion tool to combine loops and conditionals.

Prompt:

"Generate Python code to count how many numbers in a list are even and odd."

Expected Output:

• AI-generated code using loop and if condition.

• Correct count validation.

• Explanation of logic flow.

CODE:

```python
# Generate Python code to count how many numbers in a list are even and odd.
# Use a loop and if-else conditions.
# Print the count of even and odd numbers.
def count_even_odd(numbers):
    even_count = 0
    odd_count = 0

    for number in numbers:
        if number % 2 == 0:
            even_count += 1
        else:
            odd_count += 1

    print(f"Count of even numbers: {even_count}")
    print(f"Count of odd numbers: {odd_count}")
# Example usage
numbers_list = [10, 23, 45, 66, 78, 91, 34, 57]
count_even_odd(numbers_list)
```

OUTPUT:

```
2 4 6 8 10 12 14 16 18 20 22 24
Count of even numbers: 4
Count of odd numbers: 4
```

ANALYSIS:

Task 2: Loop with Conditionals (Even & Odd Count)

The code traverses a list using a loop and classifies each element as even or odd using if-else conditions.

It ensures accurate counting and reinforces decision-making within iterative structures.

Task Description #3 (AI-Based Code Completion for Class Attributes Validation)

Task: Use an AI tool to complete a Python class that validates user input.

Prompt:

“Generate a Python class User that validates age and email using conditional statements.”

Expected Output:

• AI-generated class with validation logic.

• Verification of condition handling.

• Test cases for valid and invalid inputs.

CODE:

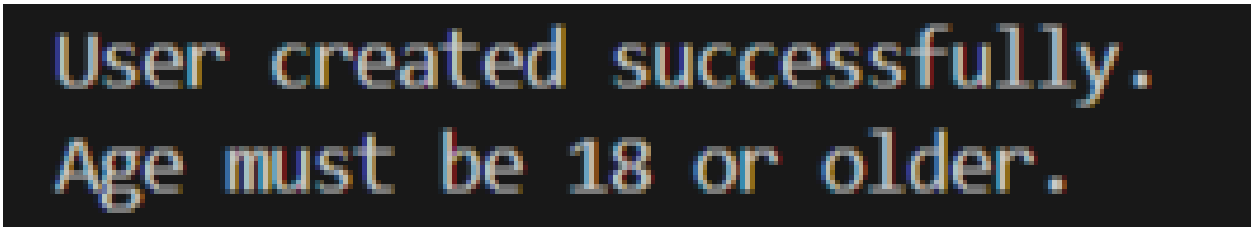
```python
# Generate a Python class named User.
# The class should validate age (age must be >= 18)
# Validate email (must contain '@' and '.')
# Use conditional statements inside methods.
class User:
    def __init__(self, name, age, email):
        self.name = name
        self.age = age
        self.email = email

        if not self.validate_age():
            raise ValueError("Age must be 18 or older.")
        if not self.validate_email():
            raise ValueError("Invalid email format.")

    def validate_age(self):
        return self.age >= 18

    def validate_email(self):
        return '@' in self.email and '.' in self.email
# Example usage
try:
    user1 = User("Alice", 20, "alice@example.com")
    print("User created successfully.")
except ValueError as e:
    print(e)
try:
    user2 = User("Bob", 16, "bobexample.com")
    print("User created successfully.")
except ValueError as e:
    print(e)
```

OUTPUT:



ANALYSIS:

Task 3: Class with Validation (User)

The class uses conditional statements to validate user age and email input.

It improves data integrity by preventing invalid values through logical checks.

Task Description #4 (AI-Based Code Completion for Classes)

Task: Use an AI code completion tool to generate a Python class for managing student details.

Prompt:

"Generate a Python class Student with attributes (name, roll number, marks) and methods to calculate total and average marks."

Expected Output:

• AI-generated class code.

• Verification of correctness and completeness of class structure.

• Minor manual improvements (if needed) with justification.

CODE:

```python
# Generate a Python class Student with attributes:
# name, roll_number, and marks (list).
# Add methods to calculate total marks and average marks.
# Display student details neatly.
class Student:
    def __init__(self, name, roll_number, marks):
        self.name = name
        self.roll_number = roll_number
        self.marks = marks

    def total_marks(self):
        return sum(self.marks)

    def average_marks(self):
        if len(self.marks) == 0:
            return 0
        return self.total_marks() / len(self.marks)

    def display_details(self):
        print(f"Student Name: {self.name}")
        print(f"Roll Number: {self.roll_number}")
        print(f"Marks: {self.marks}")
        print(f"Total Marks: {self.total_marks()}")
        print(f"Average Marks: {self.average_marks():.2f}")
# Example usage
student1 = Student("John Doe", 101, [85, 90, 78, 92])
student1.display_details()
```

OUTPUT:

```
Student Name: John Doe
Roll Number: 101
Marks: [85, 90, 78, 92]
Total Marks: 345
Average Marks: 86.25
```

ANALYSIS:

Task 4: Class – Student Details

The student class encapsulates attributes and methods to compute total and average marks.

It demonstrates object-oriented principles such as data encapsulation and method abstraction.

Task Description 5 (AI-Assisted Code Completion Review)

Task: Use an AI tool to generate a complete Python program using classes, loops, and conditionals together.

Prompt:

"Generate a Python program for a simple bank account system using class, loops, and conditional statements."

Expected Output:

• Complete AI-generated program.

• Identification of strengths and limitations of AI suggestions.

• Reflection on how AI assisted coding productivity.

CODE:

A61.py > ...

```python
# Generate a Python program for a simple bank account system.
# Use a class with deposit and withdrawal methods.
# Use loops for menu-driven operations.
# Use conditional statements for balance checks.
class BankAccount:
    def __init__(self, account_holder, initial_balance=0):
        self.account_holder = account_holder
        self.balance = initial_balance

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            print(f"Deposited: ${amount:.2f}")
        else:
            print("Deposit amount must be positive.")

    def withdraw(self, amount):
        if amount > self.balance:
            print("Insufficient balance for withdrawal.")
        elif amount <= 0:
            print("Withdrawal amount must be positive.")
        else:
            self.balance -= amount
            print(f"Withdrew: ${amount:.2f}")

    def display_balance(self):
        print(f"Current Balance: ${self.balance:.2f}")
def bank_menu():
    account_holder = input("Enter account holder name: ")
    account = BankAccount(account_holder)

```

A61.py > ...

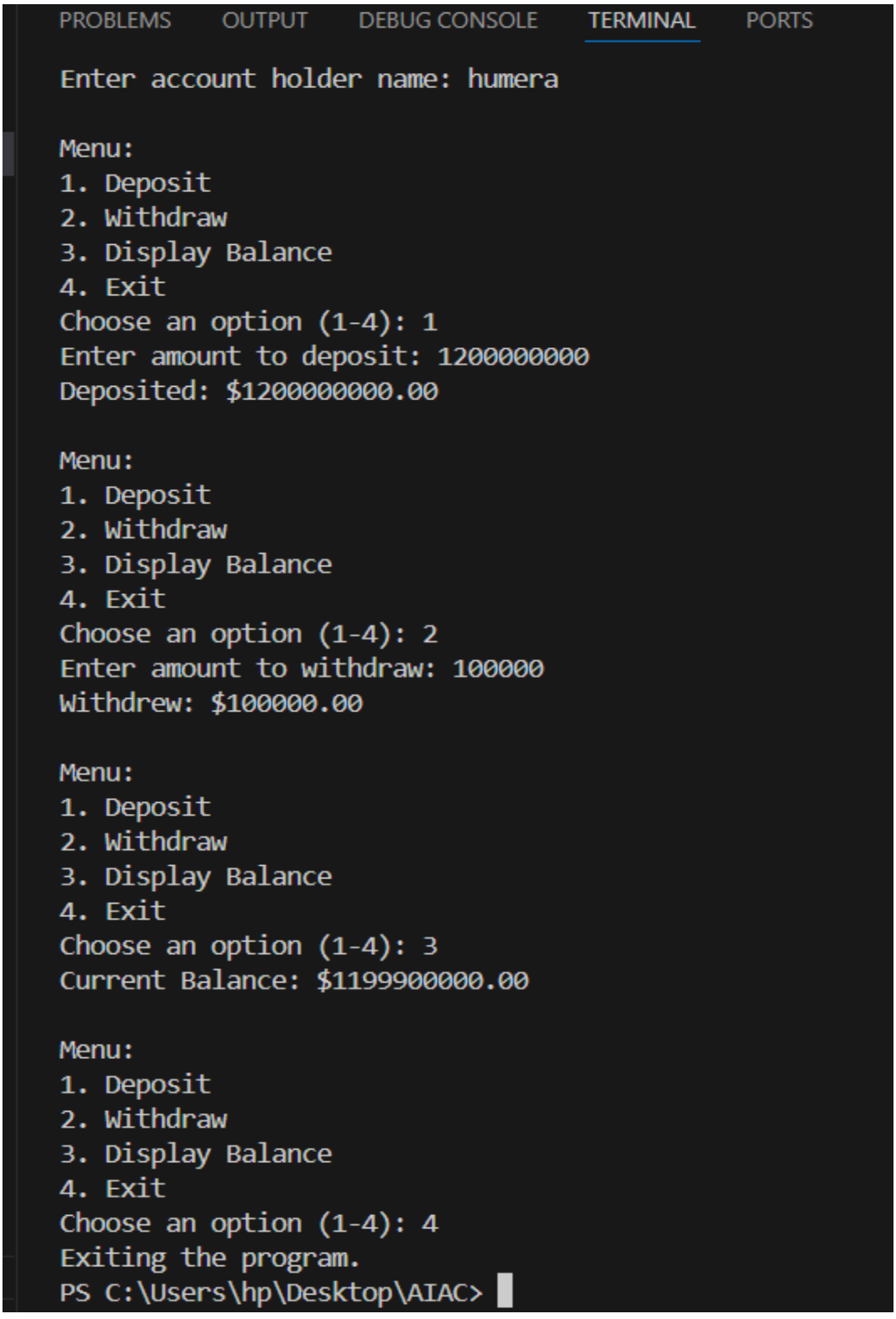
```python
def bank_menu():

    while True:
        print("\nMenu:")
        print("1. Deposit")
        print("2. Withdraw")
        print("3. Display Balance")
        print("4. Exit")

        choice = input("Choose an option (1-4): ")

        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 the program.")
            break
        else:
            print("Invalid choice. Please select a valid option.")
bank_menu()
```

OUTPUT:


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Enter account holder name: humera
Menu:
1. Deposit
2. Withdraw
3. Display Balance
4. Exit
Choose an option (1-4): 1
Enter amount to deposit: 1200000000
Deposited: $1200000000.00
Menu:
1. Deposit
2. Withdraw
3. Display Balance
4. Exit
Choose an option (1-4): 2
Enter amount to withdraw: 100000
Withdrew: $100000.00
Menu:
1. Deposit
2. Withdraw
3. Display Balance
4. Exit
Choose an option (1-4): 3
Current Balance: $1199900000.00
Menu:
1. Deposit
2. Withdraw
3. Display Balance
4. Exit
Choose an option (1-4): 4
Exiting the program.
PS C:\Users\hp\Desktop\AIAC>

ANALYSIS:

Task 5: Bank Account System (Complete Program)

The program integrates classes, loops, and conditionals to simulate real-world banking operations.
It highlights how AI-assisted code can improve productivity while still requiring logical validation.

NAME: HUMERA NUZHAT

ROLL NO: 2303A52083

BATCH: 39