AI ASSISTED CODEING
ASSIGNMENT - 6.1
ROLL NO.2303A510C8
B-27

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.

Prompt:

```
#Write a Python program that takes an integer N as input and displays all even numbers from 1 up to N using a loop.
```

CODE:

```
# Get user input
N = int(input("Enter an integer N: "))
# Display even numbers from 1 to N
print(f"Even numbers from 1 to {N}:")
for num in range(2, N + 1, 2):
    print(num)
```

Output:

```
Enter an integer N: 15
Even numbers from 1 to 15:
2
4
6
8
10
12
```

```
14
```

Conclusion:

The AI proved to be an efficient tool for generating standard control flow structures, providing not just the raw logic but also the necessary scaffolding (input/output handling) to make the code a complete, runnable program.

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.

Prompt:

```
#Create a Python program that takes a list of integers and counts how many elements are even and how many are odd using a loop and conditional statements.
```

Code:

```
# Get user input
numbers = list(map(int, input("Enter a list of integers separated by spaces:
").split()))
# Initialize counters
even_count = 0
odd_count = 0
# Count even and odd numbers
for num in numbers:
```

```
    if num % 2 == 0:
        even_count += 1
    else:
        odd_count += 1
# Display the counts
print(f"Even numbers count: {even_count}")
print(f"Odd numbers count: {odd_count}")
```

Output:

```
Enter a list of integers separated by spaces: 3 4 5 6 8 9
Even numbers count: 3
Odd numbers count: 3
```

Conclusion:

Synthesize Control Structures: It successfully nested a conditional statement (`if/else`) inside an iterative structure (`for`loop), which is a fundamental programming pattern.

State Management: It correctly identified the need for external variables (`even_count`, `odd_count`) to track state across iterations.

Logic Precision: The use of the modulo operator (`%`) was correctly applied to distinguish between binary states (even vs. odd).

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.

Prompt:

```
#Create a Python class named User that accepts age and email as input and validates them using conditional statements.
```

Code:

```
class User:
    def __init__(self, age, email):
        self.age = age
        self.email = email
        self.validate_age()
        self.validate_email()

    def validate_age(self):
        if self.age < 0:
            print("Invalid age: Age cannot be negative.")
        elif self.age < 18:
            print("User is a minor.")
        else:
            print("User is an adult.")

    def validate_email(self):
        if "@" in self.email and "." in self.email.split("@")[-1]:
            print("Valid email address.")
        else:
            print("Invalid email address.")
# Get user input
age = int(input("Enter your age: "))
email = input("Enter your email address: ")
# Create User object
user = User(age, email)
```

Output:

```
Enter your age: 21
Enter your email address: 2303A510C8@sru.edu.in
```

```
User is an adult.
Valid email address.

Enter your age: 17
Enter your email address: 2303A510C8@sru.edu.in
User is a minor.
Valid email address.
```

Conclusion:

Encapsulation: The validation logic was encapsulated within setter methods (`set_age`, `set_email`) rather than leaving the raw data exposed.

Defensive Programming: The AI generated code that anticipates bad input (negative numbers, malformed strings) and handles it gracefully using Exceptions.

Scalability: This structure allows for more complex rules (e.g., password complexity checks) to be added easily without breaking the main object structure.

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.

Prompt:

```
# Generate a Python class Student with attributes name, roll number, and marks, and include methods to calculate total and average marks.
```

Code:

```
class Student:
    def __init__(self, name, roll_number, marks):
        self.name = name
        self.roll_number = roll_number
        self.marks = marks  # marks should be a list of integers

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

    def calculate_average(self):
        if len(self.marks) == 0:
            return 0
        return self.calculate_total() / len(self.marks)
# Example usage:
student1 = Student("Alice", 101, [85, 90, 78, 92])
total_marks = student1.calculate_total()
average_marks = student1.calculate_average()
print(f"Student Name: {student1.name}")
print(f"Roll Number: {student1.roll_number}")
print(f"Total Marks: {total_marks}")
print(f"Average Marks: {average_marks}")
```

Output:

```
Student Name: Alice
Roll Number: 101
Total Marks: 345
```

```
Average Marks: 86.25
```

Conclusion:

The AI successfully generated the core logic required. The manual intervention focused on readability and type safety, which are essential for long-term code maintenance but often omitted in basic AI snippets.

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.

Prompt:

```
# Design a Python program for a basic bank account system using a class, a looping menu, and conditional statements for user actions.
```

Code:

```
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 0 < amount <= self.balance:

            self.balance -= amount

            print(f"Withdrew: ${amount:.2f}")

        else:

            print("Insufficient funds or invalid withdrawal amount.")



    def check_balance(self):

        print(f"Current balance: ${self.balance:.2f}")
```

```
def main():

    print("Welcome to the Basic Bank Account System")

    account_holder = input("Enter account holder name: ")

    account = BankAccount(account_holder)



    while True:

        print("\nMenu:")

        print("1. Deposit")

        print("2. Withdraw")

        print("3. Check 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.check_balance()

        elif choice == '4':

            print("Thank you for using the bank account system. Goodbye!")

            break
```

```
        else:
            print("Invalid choice. Please select a valid option.")
if __name__ == "__main__":
    main()
```

Output:

```
Welcome to the Basic Bank Account System
Enter account holder name: Srinivas

Menu:
1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Choose an option (1-4): 1
Enter amount to deposit: 4000
Deposited: $4000.00

Menu:
1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Choose an option (1-4): 3
Current balance: $4000.00
```

```
Menu:
1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Choose an option (1-4): 2
Enter amount to withdraw: 400
Withdrew: $400.00

Menu:
1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Choose an option (1-4): 3
Current balance: $3600.00

Menu:
1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Choose an option (1-4): 4
Thank you for using the bank account system.
Goodbye!
```

Conclusion:

The AI provided a robust foundation that satisfies all prompt requirements. While it excels at logical flow and syntax,

human intervention remains necessary to implement complex features like data security and persistent storage.