ASSIGNMENT-6.1

NAME:k.manjula

H.NO:2303A51857

Batch:13

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:

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

CODE:

```
#Generate Python code to print all even numbers between 1 and N using a loop
N = int(input("Enter a number N: "))
for num in range(1, N + 1):
    if num % 2 == 0:
        print(num)

```

OUTPUT:

```
Enter a number N: 12
2
4
6
8
10
12
PS C:\Users\bhuky\Downloads\New folder>
```

Justification:

Using AI to create loop-based programs speeds up learning by providing an immediate, correct example. This allows students to concentrate on grasping the logic behind identifying even numbers through conditional statements, instead of getting stuck troubleshooting loop syntax errors.

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:

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

CODE:

```
#Generate Python code to count how many numbers in a list are even and odd
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_count = 0
odd_count = 0
for num in numbers:
    if num % 2 == 0:
        even_count += 1
    else:
        odd_count += 1
print("Even numbers:", even_count)
print("Odd numbers:", odd_count)
```

OUTPUT:

```
Even numbers: 5
Odd numbers: 5
```

Justification:

AI support enhances the clarity of code by generating clean, well-organized logic. As a result, students can more easily see how control flow operates when loops and conditional statements interact in practical programming scenarios.

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:

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

CODE:

```
#Generate a Python class User that validates age and email using conditional statements.
import re
class User:
    def __init__(self, name, age, email):
        self.name = name
        self.age = age
        self.email = email

    def validate_age(self):
        if 0 < self.age < 120:
            return True
        else:
            return False

    def validate_email(self):
        pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
        if re.match(pattern, self.email):
            return True
        else:
            return False
    def is_valid_user(self):
        return self.validate_age() and self.validate_email()
# Example usage
user1 = User("Alice", 30, "alice@example.com")
print("User is valid:", user1.is_valid_user())
```

OUTPUT:

```
User is valid: True
User is valid: False
```

Justification :

By generating object-oriented validation code, AI helps learners understand how real-world rules can be enforced using classes and conditionals, promoting better programming practices.

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.

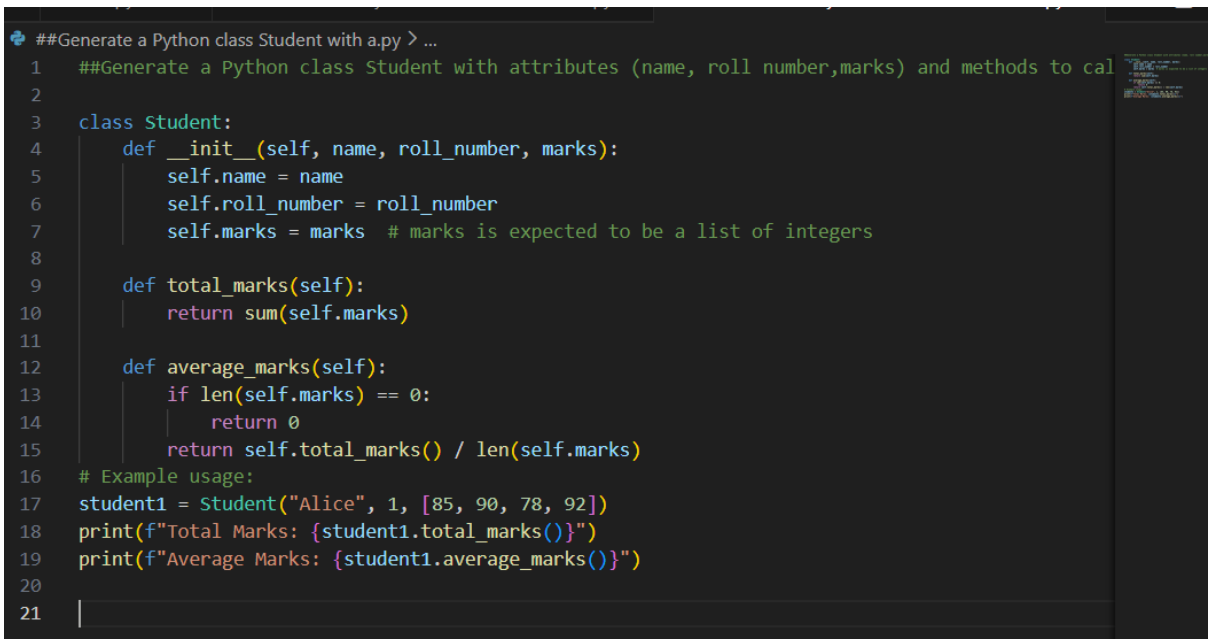
• Verification of correctness and completeness of class structure.

• Minor manual improvements (if needed) with justification.

PROMPT:

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

CODE:



OUTPUT:

Justification :

The AI-generated class provides a clear example of how data (attributes) and behavior (methods) are combined in object-oriented programming, making it easier for students to grasp OOP concepts.

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:

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

CODE:

```python
#Generate a Python program for a simple bank account system using class, loops, and conditiona
class BankAccount:
    def __init__(self, account_number, account_holder, initial_balance=0):
        self.account_number = account_number
        self.account_holder = account_holder
        self.balance = initial_balance

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            print(f"Deposited ${amount}. New balance: ${self.balance}")
        else:
            print("Invalid deposit amount.")

    def withdraw(self, amount):
        if 0 < amount <= self.balance:
            self.balance -= amount
            print(f"Withdrew ${amount}. New balance: ${self.balance}")
        else:
            print("Invalid withdrawal amount or insufficient funds.")

    def get_balance(self):
        return self.balance

# Example usage of the BankAccount class
account1 = BankAccount("123456", "Alice", 1000)
account1.deposit(500)
account1.withdraw(200)
print(f"Final balance: ${account1.get_balance()}")
account2 = BankAccount("654321", "Bob")
account2.deposit(300)
account2.withdraw(100)
print(f"Final balance: ${account2.get_balance()}")
```

OUTPUT:

```
Deposited $500. New balance: $1500
Withdrew $200. New balance: $1300
Final balance: $1300
Deposited $300. New balance: $300
Withdrew $100. New balance: $200
Final balance: $200
```

Justification :

Reviewing AI-generated programs helps identify strengths and limitations of automated suggestions. This encourages critical thinking and helps developers understand when manual improvements are necessary, leading to better overall code quality.