Assignment 6.1

Name: Zobiya Fatima

Batch No: 14

Hall Ticket No: 2303A51879

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.

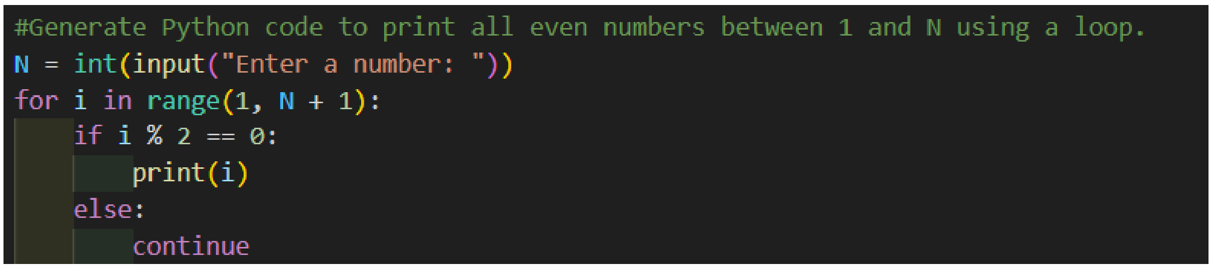
• Identification of loop type used (for or while).

• Validation with sample inputs.

PROMPT:

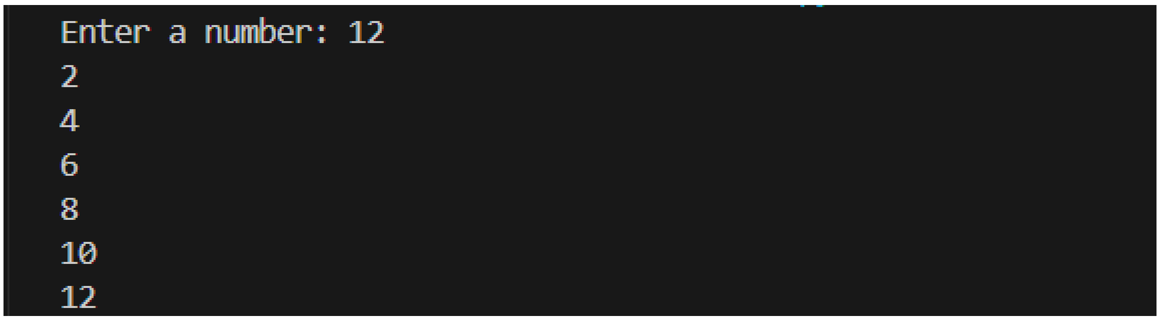
#Generate Python code to print all even numbers between 1 and N using a loop.

CODE:



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

OUTPUT:



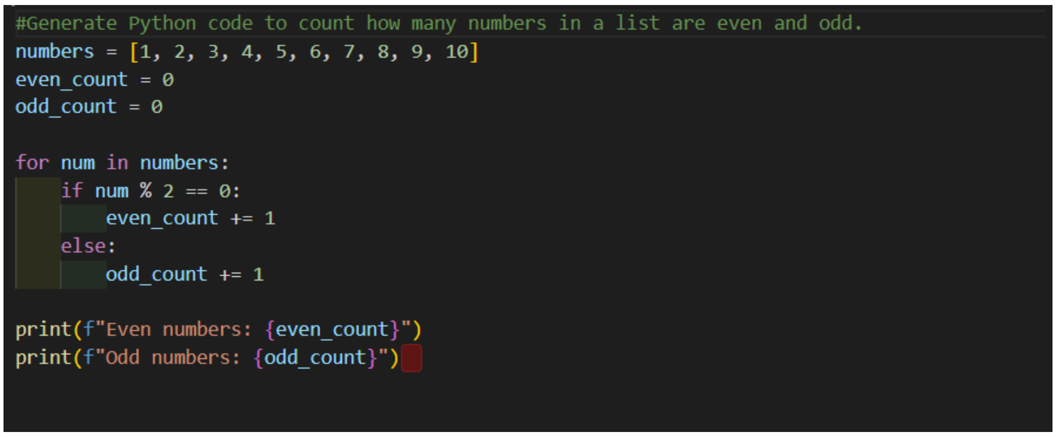
```
Enter a number: 12
2
4
6
8
10
12
```

Justification :
AI-based code completion helps programmers quickly generate correct loop structures without worrying about syntax errors. This saves time and ensures that the logic for iterating from 1 to N is implemented properly, especially for beginners who are still learning loop concepts.

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:

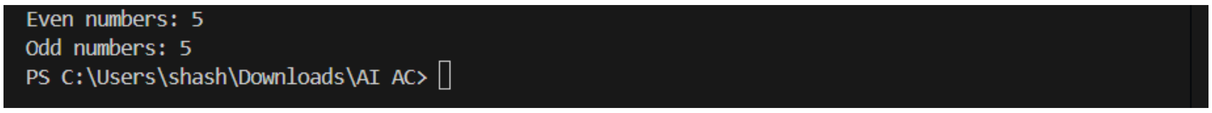


```python
#Generate Python code to count how many numbers in a list are even and odd.
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_count = 0
odd_count = 0

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

print(f"Even numbers: {even_count}")
print(f"Odd numbers: {odd_count}")
```

Output:



```
Even numbers: 5
Odd numbers: 5
PS C:\Users\shash\Downloads\AI AC> 
```

Justification

AI-generated code accurately combines loops with conditional statements, reducing logical mistakes such as incorrect conditions or misplaced counters. This ensures reliable results when counting even and odd numbers in a list.

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.

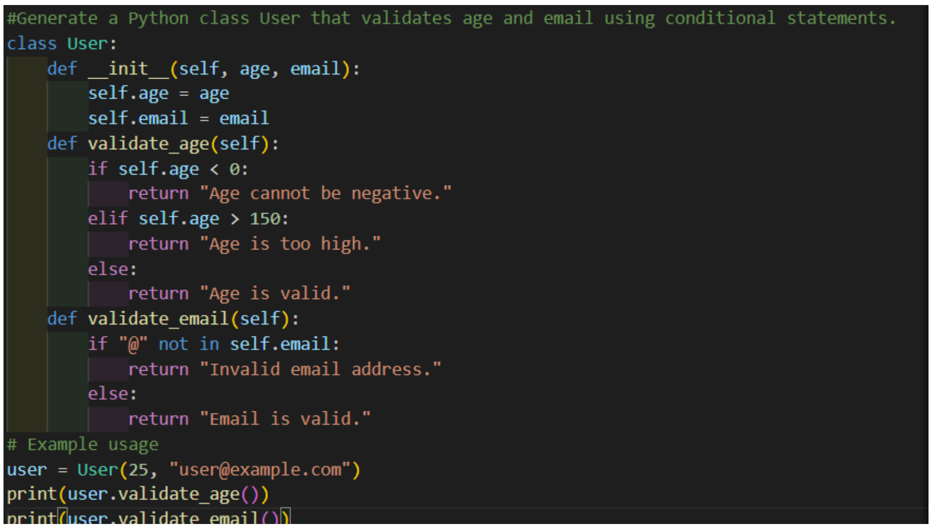
• Verification of condition handling.

• Test cases for valid and invalid inputs.

PROMPT:

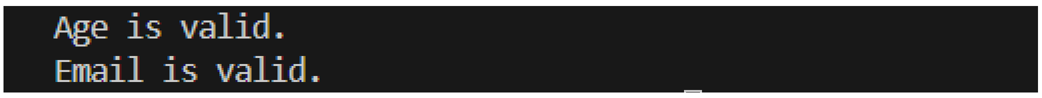
#Generate a Python class User that validates age and email using conditional statements.

CODE:



```
#Generate a Python class User that validates age and email using conditional statements.
class User:
    def __init__(self, age, email):
        self.age = age
        self.email = email
    def validate_age(self):
        if self.age < 0:
            return "Age cannot be negative."
        elif self.age > 150:
            return "Age is too high."
        else:
            return "Age is valid."
    def validate_email(self):
        if "@" not in self.email:
            return "Invalid email address."
        else:
            return "Email is valid."
# Example usage
user = User(25, "user@example.com")
print(user.validate_age())
print(user.validate_email())
```

OUTPUT:

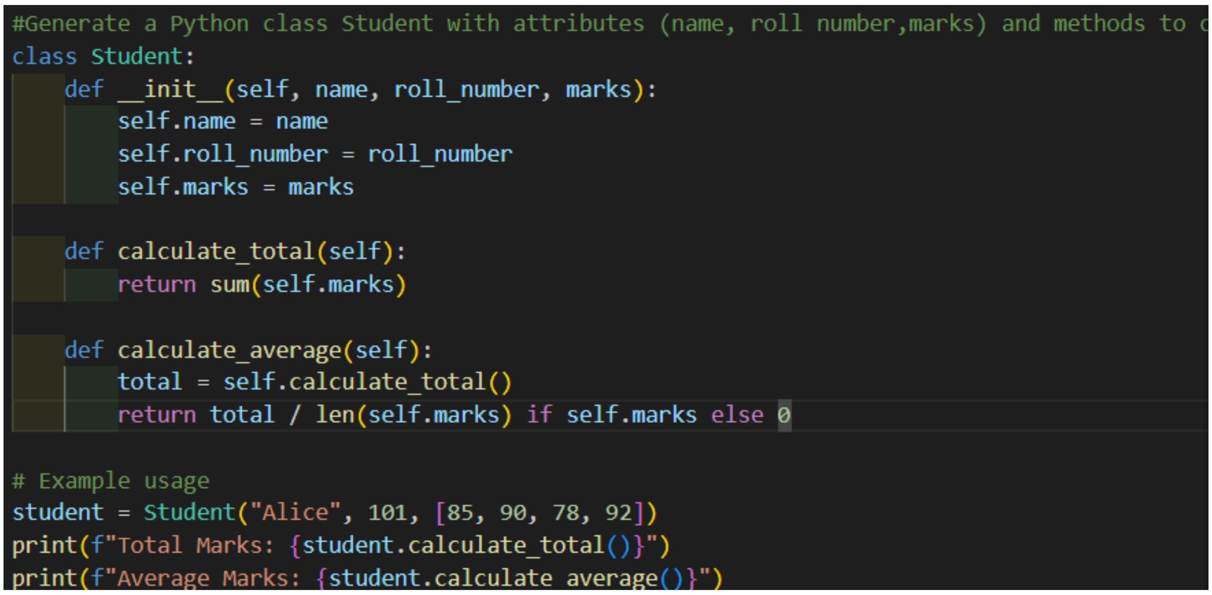


```
Age is valid.
Email is valid.
```

Justification :
AI simplifies the creation of validation logic inside classes, ensuring that conditions like age limits and email format checks are implemented correctly. This improves data reliability and prevents invalid inputs from being accepted.

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:



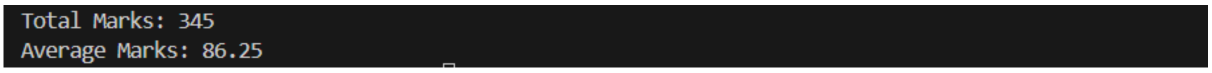
```
#Generate a Python class Student with attributes (name, roll number,marks) and methods to c
class Student:
    def __init__(self, name, roll_number, marks):
        self.name = name
        self.roll_number = roll_number
        self.marks = marks

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

    def calculate_average(self):
        total = self.calculate_total()
        return total / len(self.marks) if self.marks else 0

# Example usage
student = Student("Alice", 101, [85, 90, 78, 92])
print(f"Total Marks: {student.calculate_total()}")
print(f"Average Marks: {student.calculate_average()}")
```

OUTPUT:



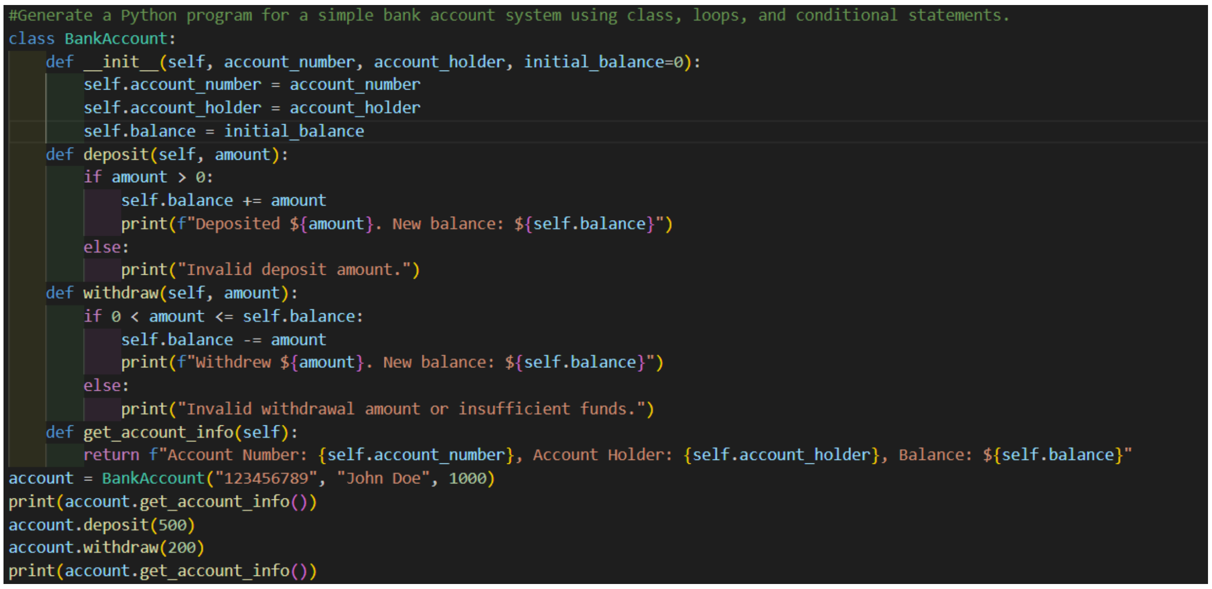
```
Total Marks: 345
Average Marks: 86.25
```

Justification :
AI assists in quickly generating complete class structures with attributes and methods, reducing the effort needed to design classes from scratch. This helps developers focus more on logic rather than boilerplate code.

Task Description #5 (AI-Assisted Code Completion Review)

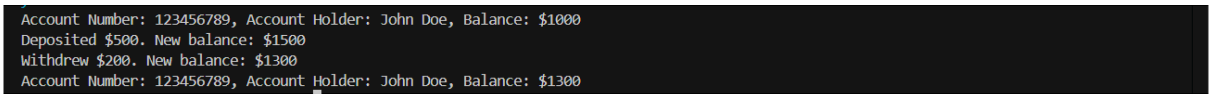
Task: Use an AI tool to generate a complete Python program using classes, loops, and conditionals together.
Prompt:
"Generate a Python program for a simple bank account system using class, loops, and conditional statements."
Expected Output:
• Complete AI-generated program.
• Identification of strengths 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 conditional statements.
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_account_info(self):
        return f"Account Number: {self.account_number}, Account Holder: {self.account_holder}, Balance: ${self.balance}"
account = BankAccount("123456789", "John Doe", 1000)
print(account.get_account_info())
account.deposit(500)
account.withdraw(200)
print(account.get_account_info())
```

OUTPUT:



```
Account Number: 123456789, Account Holder: John Doe, Balance: $1000
Deposited $500. New balance: $1500
Withdrew $200. New balance: $1300
Account Number: 123456789, Account Holder: John Doe, Balance: $1300
```

Justification :

AI-assisted coding significantly improves productivity by generating a complete program that integrates classes, loops, and conditionals. This reduces development time and allows programmers to quickly build functional applications