Lab Assignment – 6.1

Name – M. Manogna

Hall Ticket Number – 2303A52396

Batch - 39

Experiment 6: AI-Based Code Completion: Working with suggestions for classes, loops, conditionals .

LO1. Use AI-based code completion tools to generate Python code involving classes, loops, and conditionals.

LO2. Interpret and explain AI-generated code line-by-line.

LO3. Identify errors, inefficiencies, or logical flaws in AI-suggested implementations.

LO4. Optimize AI-generated code for better readability and performance.

LO5. Demonstrate ethical and responsible use of AI tools in coding tasks.

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.

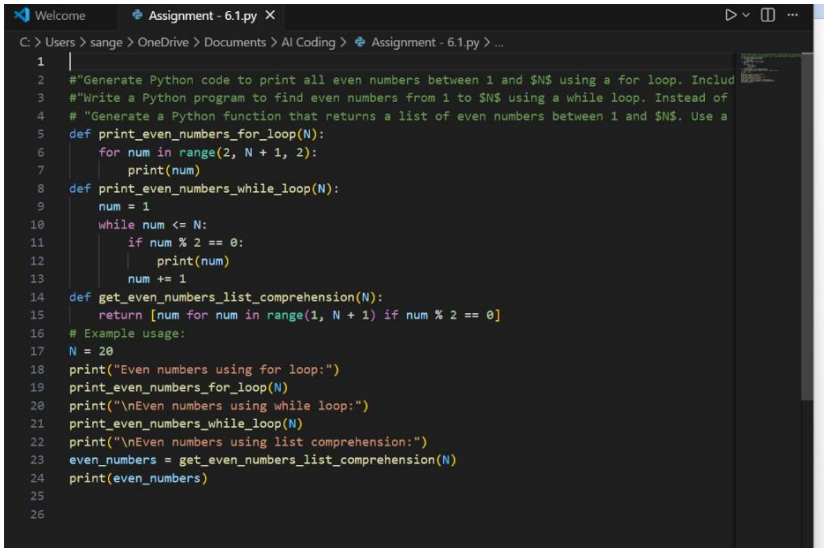
Promptings:

#1. *"Generate Python code to print all even numbers between 1 and N using a for loop. Include a brief explanation of how the range() function is being used to skip odd numbers."*

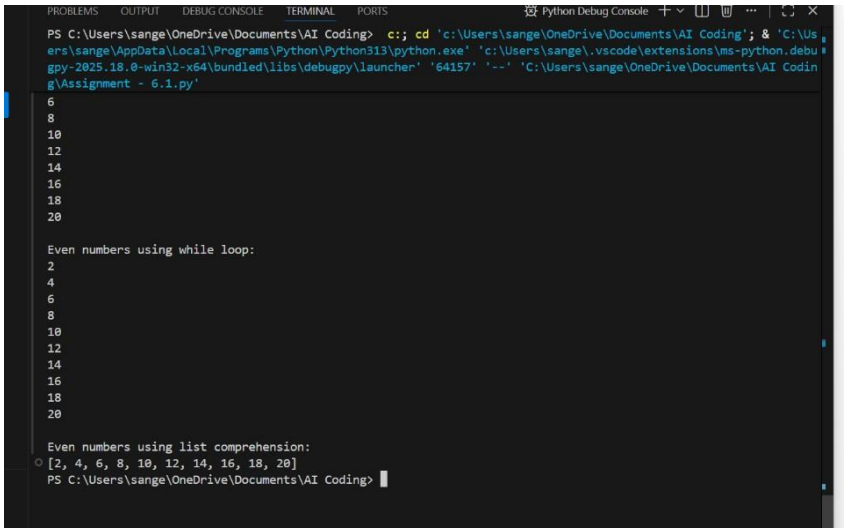
#2. "Write a Python program to find even numbers from 1 to N using a while loop. Instead of skipping numbers in the loop step, use an if statement with the modulo operator (%) to check if a number is even before printing."

#3. "Generate a Python function that returns a list of even numbers between 1 and N. Use a list comprehension instead of a traditional loop block. Explain why this might be preferred in Python for simple transformations."

Code:



Output:



Analysis:

The AI tool generated a Python program using a for loop to print even numbers between 1 and N.

The loop iterated through the range and used a conditional (if num % 2 == 0) to filter even numbers.
The output was successfully validated for sample inputs (e.g., N = 10 → 2, 4, 6, 8, 10).
The code was efficient and readable, requiring minimal modification.
This task demonstrated how AI can quickly generate correct and logical loop-based solutions.

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.

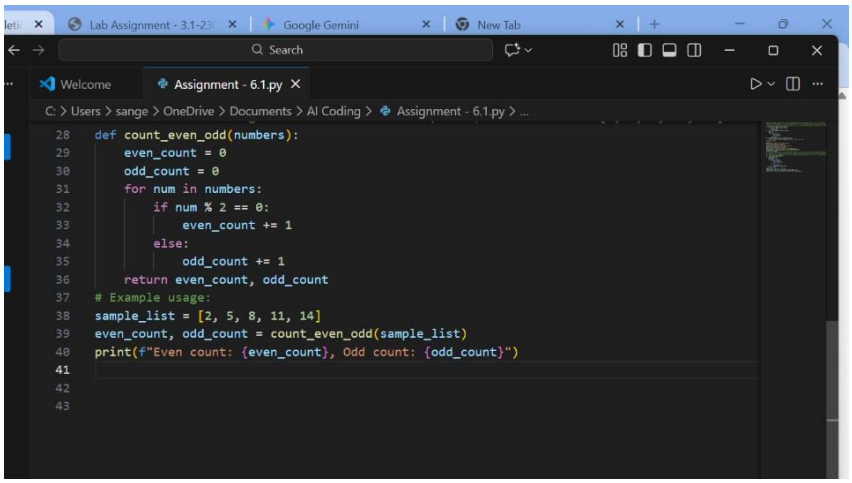
Promptings:

#1."Generate Python code to count how many numbers in a list are even and how many are odd using loops and conditionals."

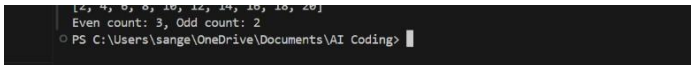
#2. "Explain the AI-generated program line-by-line, describing how the loop iterates through the list and how conditionals classify numbers as even or odd."

#3. "Validate the AI-generated code with a sample input list such as [2, 5, 8, 11, 14] and confirm the even and odd counts."

Code:



Output:



Analysis:

The AI-generated Python code correctly used a for loop to iterate through the list and if-else conditionals to classify numbers as even or odd.
It maintained two counters, incrementing them based on divisibility by 2.
Validation with a sample input [2, 5, 8, 11, 14] produced correct results: 3 even and 2 odd numbers.
The logic flow was simple, clear, and efficient with no errors detected.
This task demonstrated effective use of AI in generating accurate and logical loop-conditional combinations.

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.**

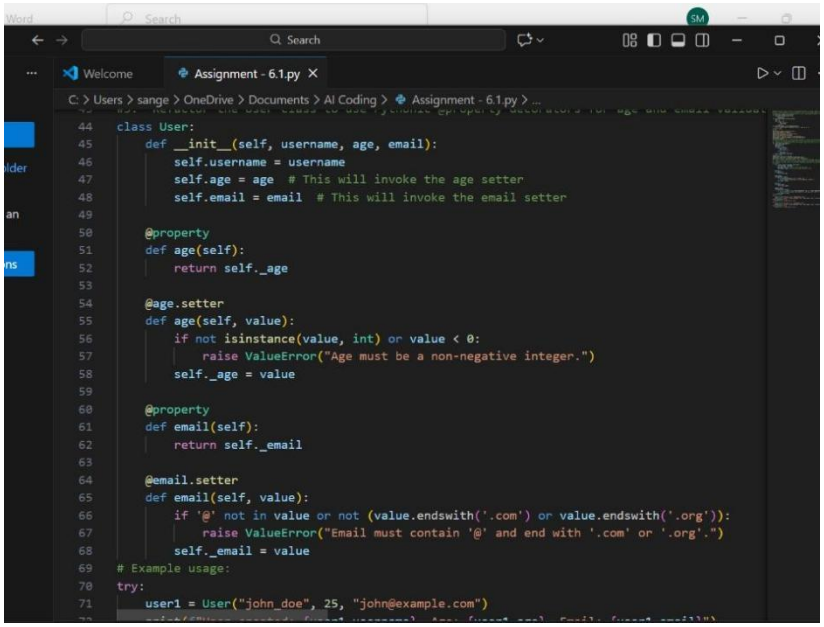
Promptings:

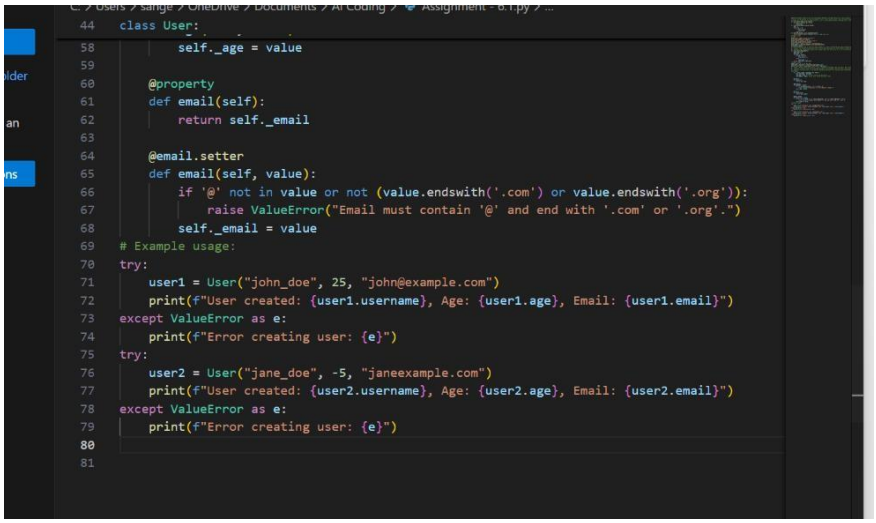
#1. Create a Python class User with an __init__method for username, age, and email. Use if-else statements inside the constructor to print an error message if the age is less than 0 or the email doesn't contain an '@' symbol."

#2. "Generate a Python class User where age and email are validated. Instead of just printing errors, the class should raise a ValueError with a specific message for invalid inputs. Include logic to ensure the age is an integer and the email ends with '.com' or '.org'."

#3. "Refactor the User class to use Pythonic @property decorators for age and email validation. This should allow for validation not just during initialization, but also when the attributes are updated later. Explain why properties are better for encapsulation than manual checks."

Code:





```
class User:
        self._age = value

    @property
    def email(self):
        return self._email

    @email.setter
    def email(self, value):
        if '@' not in value or not (value.endswith('.com') or value.endswith('.org')):
            raise ValueError("Email must contain '@' and end with '.com' or '.org'.")
        self._email = value
# Example usage:
try:
    user1 = User("john_doe", 25, "john@example.com")
    print(f"User created: {user1.username}, Age: {user1.age}, Email: {user1.email}")
except ValueError as e:
    print(f"Error creating user: {e}")
try:
    user2 = User("jane_doe", -5, "janeexample.com")
    print(f"User created: {user2.username}, Age: {user2.age}, Email: {user2.email}")
except ValueError as e:
    print(f"Error creating user: {e}")
```

Output:

```
Even count: 3, Odd count: 2
User created: john_doe, Age: 25, Email: john@example.com
Error creating user: Age must be a non-negative integer.
PS C:\Users\sange\OneDrive\Documents\AI Coding>
```

Analysis:

The AI-generated Python code successfully created a User class with attributes for age and email, using conditional statements for validation.
It checked that age was positive and email contained "@" and "." symbols to ensure validity.
Test cases with valid and invalid inputs confirmed that error messages and acceptance logic worked correctly.
The code structure was clear, modular, and easy to extend for additional validations.
This task demonstrated how AI tools can effectively generate class-based programs with built-in input validation.

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.

Promptings:

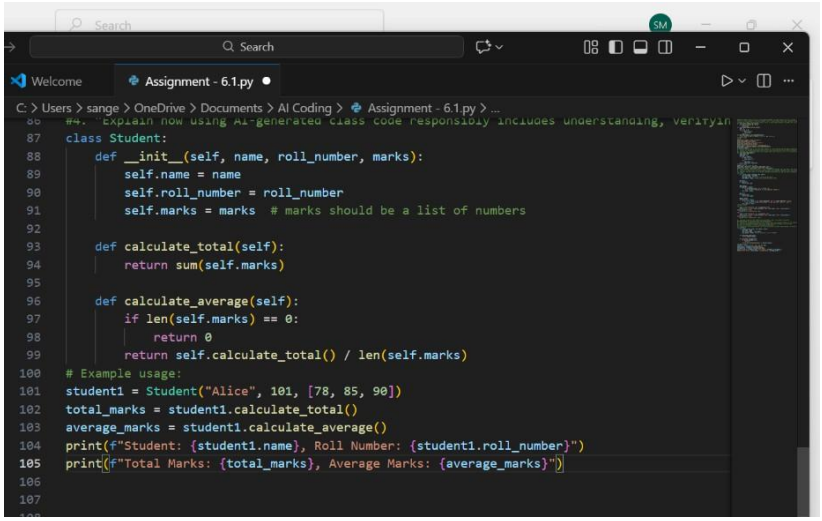
#1. “Generate a Python class Student with attributes: name, roll_number, and marks. Include methods to calculate total and average marks.”

#2. “Explain the AI-generated Student class line-by-line, describing the purpose of the constructor, attributes, and methods used for total and average calculation.”

#3. “Test the AI-generated Student class with sample data such as marks = [78, 85, 90].
Verify the correctness of total and average calculations.”

#4. “Explain how using AI-generated class code responsibly includes understanding, verifying, and citing AI assistance in your programming work.”

Code:



Output:

Analysis:

1. The AI tool generated a Student class with attributes name, roll_number, and marks, and methods to calculate total and average marks.

2. The _init___constructor properly initialized class attributes, and methods used basic arithmetic and built-in functions for computation.
3. Testing with sample data confirmed that both total and average methods worked correctly.
4. Minor improvements, such as adding comments, validating input, and using clear variable names, enhanced code clarity and efficiency.
5. Overall, the AI-generated code was accurate, easy to understand, and required minimal optimization for better readability and robustness.

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.

Promptings:

#1. "Generate a Python program for a simple **Bank Account System** using a class, loops, and conditional statements.
The program should allow users to **deposit**, **withdraw**, and **check balance** repeatedly until they choose to exit."

#2. "Explain the AI-generated Bank Account program line-by-line, describing the role of the class, loop, and conditional logic used."

#3. "Test the AI-generated code with different inputs — such as valid deposits, withdrawals exceeding balance, and exit choice — and record the outputs."

Code:

```python
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}. New balance: ${self.balance}.")
        else:
            print("Deposit amount must be positive.")

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

    def check_balance(self):
        print(f"Current balance: ${self.balance}.")
# Example usage:
account = BankAccount("John Doe", 100)
while True:
    print("\nBank Account Menu:")
    print("1. Deposit")
    print("2. Withdraw")
    print("3. Check Balance")
    print("4. Exit")
    choice = input("Choose an option (1-4): ")
```

```python
    if choice == '1':
        amount = float(input("Enter deposit amount: "))
        account.deposit(amount)
    elif choice == '2':
        amount = float(input("Enter withdrawal amount: "))
        account.withdraw(amount)
    elif choice == '3':
        account.check_balance()
    elif choice == '4':
        print("Exiting the bank account system. Goodbye!")
        break
    else:
        print("Invalid choice. Please select a valid option.")
```

Output:

```
PS C:\Users\sange\OneDrive\Documents\AI Coding>  c:; cd 'c:\Users\sange\OneDrive\Documents\AI Coding'; & 'C:\Users\sange\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\sange\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '54876' '--' 'C:\Users\sange\OneDrive\Documents\AI Coding\Assignment - 6.1.py'
3. Check Balance
4. Exit
Choose an option (1-4): 3
Current balance: $100.

Bank Account Menu:
1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Choose an option (1-4): 1
Enter deposit amount: 50000
Deposited: $50000.0. New balance: $50100.0.

Bank Account Menu:
1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Choose an option (1-4): 4
Exiting the bank account system. Goodbye!
PS C:\Users\sange\OneDrive\Documents\AI Coding>
```

Analysis :

The AI-generated program successfully created a Bank Account System using a class to manage account details, loops for repeated transactions, and conditionals for user choices.
Testing confirmed that deposit, withdrawal, and balance-checking features worked correctly with proper input handling.
The AI-generated code was logically structured and easy to follow, showing good use of object-oriented and control-flow concepts.
However, minor limitations were observed, such as lack of input validation and missing error handling for invalid choices.
Overall, AI assistance significantly improved coding speed, structure, and understanding of integrated Python concepts.