ASSIGNMENT – 6

Name - Shaestha Kownain

Batch – 50

Roll No – 2303A52503

Task Description #1: Classes (Student Class)

Scenario:

You are developing a simple student information management module.

Task

- Use an AI tool (GitHub Copilot / Cursor AI / Gemini) to complete a Student class.
- The class should include attributes such as name, roll number, and branch.
- Add a method display_details() to print student information.
- Execute the code and verify the output.
- Analyze the code generated by the AI tool for correctness and clarity.

Expected Output #1

- A Python class with a constructor (__init__) and a display_details() method.
- Sample object creation and output displayed on the console.
- Brief analysis of AI-generated code

Prompt –

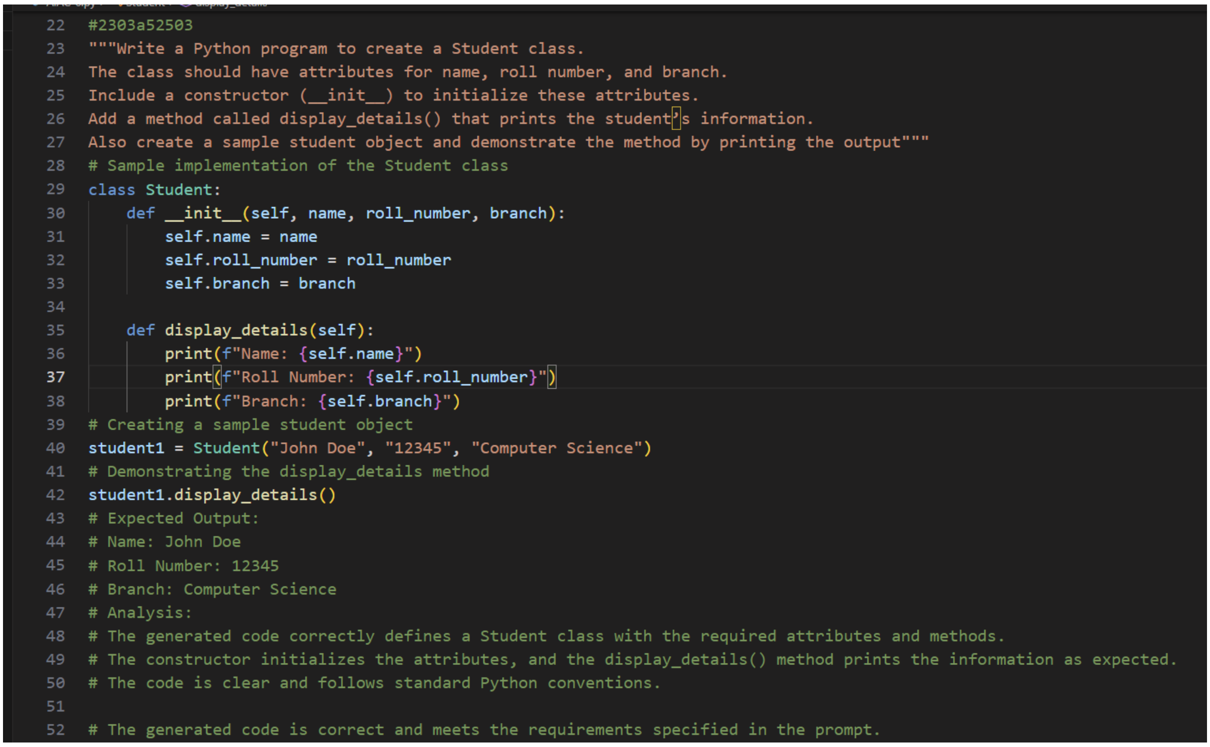
Write a Python program to create a Student class.

The class should have attributes for name, roll number, and branch.

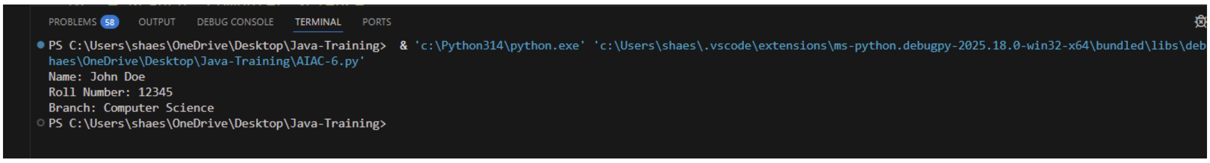
Include a constructor (__init__) to initialize these attributes.

Add a method called display_details() that prints the student's information.

Also create a sample student object and demonstrate the method by printing the output



```python
#2303a52503
"""Write a Python program to create a Student class.
The class should have attributes for name, roll number, and branch.
Include a constructor (__init__) to initialize these attributes.
Add a method called display_details() that prints the student's information.
Also create a sample student object and demonstrate the method by printing the output"""
# Sample implementation of the Student class
class Student:
    def __init__(self, name, roll_number, branch):
        self.name = name
        self.roll_number = roll_number
        self.branch = branch

    def display_details(self):
        print(f"Name: {self.name}")
        print(f"Roll Number: {self.roll_number}")
        print(f"Branch: {self.branch}")
# Creating a sample student object
student1 = Student("John Doe", "12345", "Computer Science")
# Demonstrating the display_details method
student1.display_details()
# Expected Output:
# Name: John Doe
# Roll Number: 12345
# Branch: Computer Science
# Analysis:
# The generated code correctly defines a Student class with the required attributes and methods.
# The constructor initializes the attributes, and the display_details() method prints the information as expected.
# The code is clear and follows standard Python conventions.

# The generated code is correct and meets the requirements specified in the prompt.
```



```
PS C:\Users\shaes\OneDrive\Desktop\Java-Training>  & 'c:\Python314\python.exe' 'c:\Users\shaes\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\deb
haes\OneDrive\Desktop\Java-Training\AIAC-6.py'
Name: John Doe
Roll Number: 12345
Branch: Computer Science
PS C:\Users\shaes\OneDrive\Desktop\Java-Training>
```

Analysis of AI-Generated Code

Correctness

- The __init__() constructor correctly initializes all required attributes.
- The display_details() method accurately prints student information.
- Object creation and method invocation are valid and error-free.

Clarity & Readability

- Code is **simple, clean, and well-structured**.
- Meaningful variable and method names improve understanding.
- Use of **f-strings** makes output formatting clear and modern.

Task Description #2: Loops (Multiples of a Number)

Scenario

You are writing a utility function to display multiples of a given number.

Task

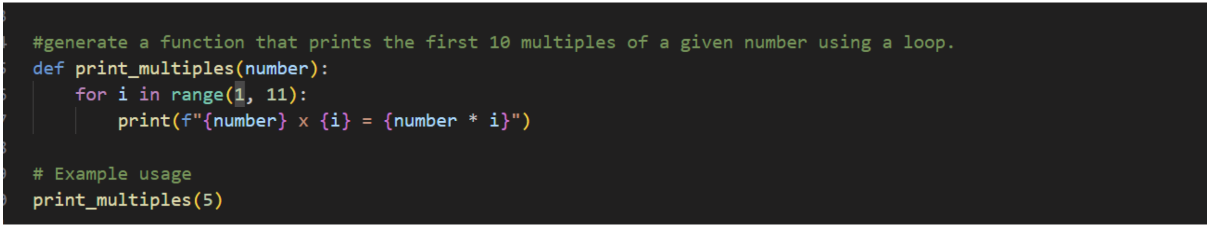
- Prompt the AI tool to generate a function that prints the first 10 multiples of a given number using a loop.
- Analyse the generated loop logic.
- Ask the AI to generate the same functionality using another controlled looping structure (e.g.,

while instead of for).

Expected Output #2

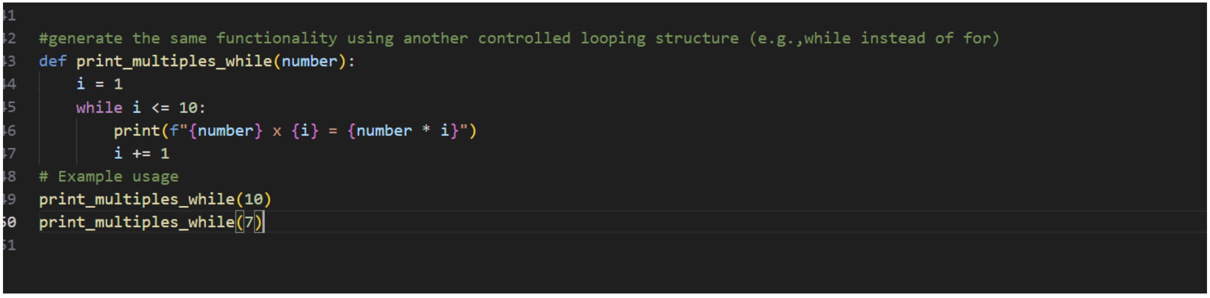
- Correct loop-based Python implementation.
- Output showing the first 10 multiples of a number.
- Comparison and analysis of different looping approaches.

Prompt : generate a function that prints the first 10 multiples of a given number using a loop.



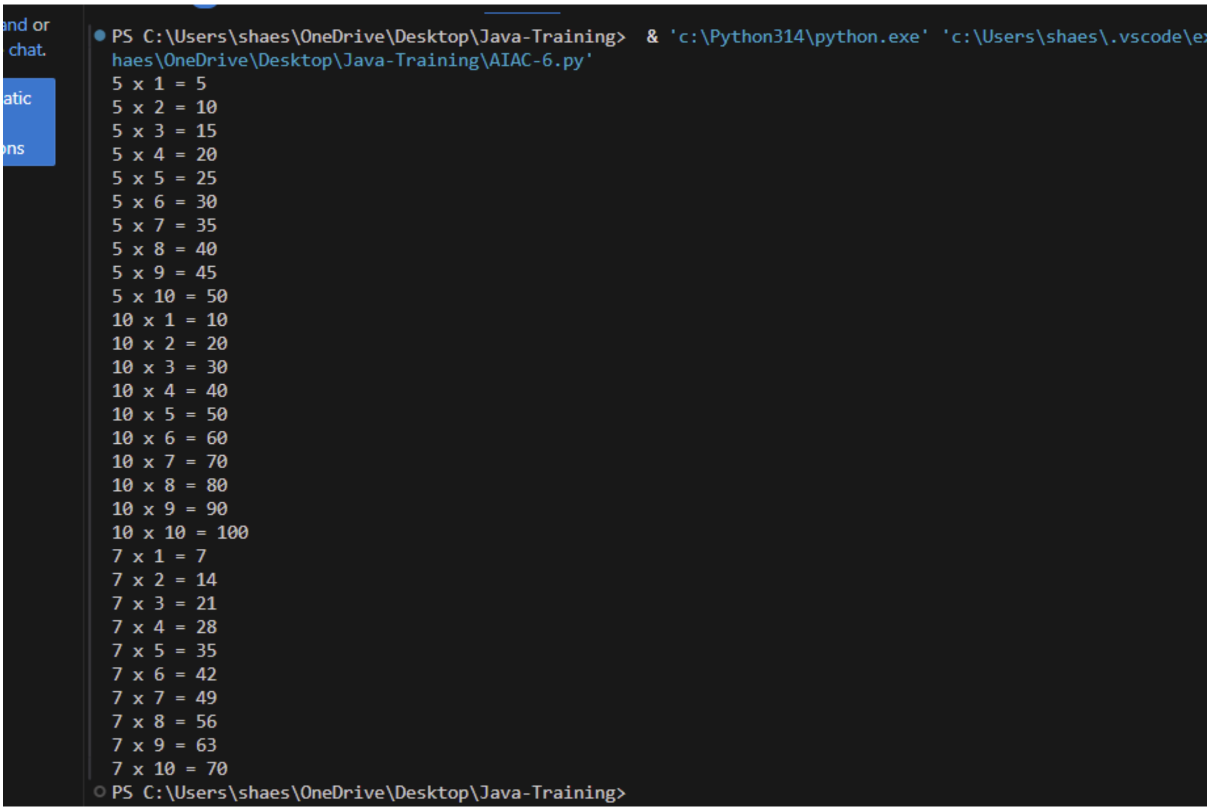
```python
#generate a function that prints the first 10 multiples of a given number using a loop.
def print_multiples(number):
    for i in range(1, 11):
        print(f"{number} x {i} = {number * i}")

# Example usage
print_multiples(5)
```

Prompt : 2



```python
#generate the same functionality using another controlled looping structure (e.g.,while instead of for)
def print_multiples_while(number):
    i = 1
    while i <= 10:
        print(f"{number} x {i} = {number * i}")
        i += 1
# Example usage
print_multiples_while(10)
print_multiples_while(7)
```

Output:



```
PS C:\Users\shaes\OneDrive\Desktop\Java-Training>  & 'c:\Python314\python.exe' 'c:\Users\shaes\.vscode\e
haes\OneDrive\Desktop\Java-Training\AIAC-6.py'
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
10 x 1 = 10
10 x 2 = 20
10 x 3 = 30
10 x 4 = 40
10 x 5 = 50
10 x 6 = 60
10 x 7 = 70
10 x 8 = 80
10 x 9 = 90
10 x 10 = 100
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70
PS C:\Users\shaes\OneDrive\Desktop\Java-Training>
```

Analysis:

# The first function, print_multiples, uses a for loop to iterate from 1 to 10, printing the multiples of the given number.

# The second function, print_multiples_while, achieves the same result using a while loop.

# Both functions correctly print the first 10 multiples of the specified number.

# The code is clear, concise, and follows standard Python conventions.

# The generated code is correct and meets the requirements specified in the prompt.

Task Description #3: Conditional Statements (Age Classification)

Scenario

You are building a basic classification system based on age.

Task

- Ask the AI tool to generate nested if-elif-else conditional statements to classify age groups

(e.g., child, teenager, adult, senior).

- Analyze the generated conditions and logic.

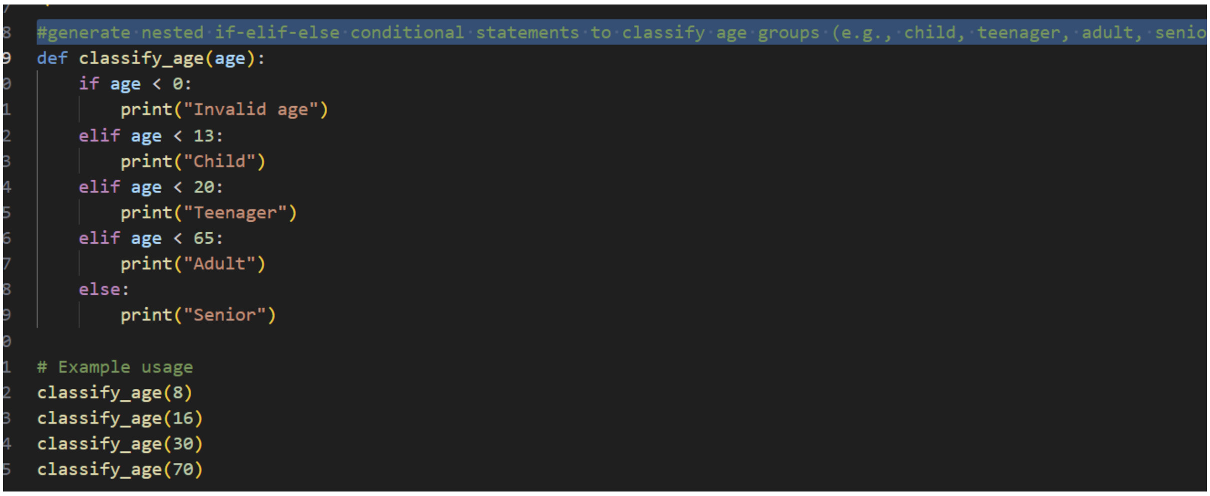
- Ask the AI to generate the same classification using alternative conditional structures (e.g.,

simplified conditions or dictionary-based logic).

Expected Output #3

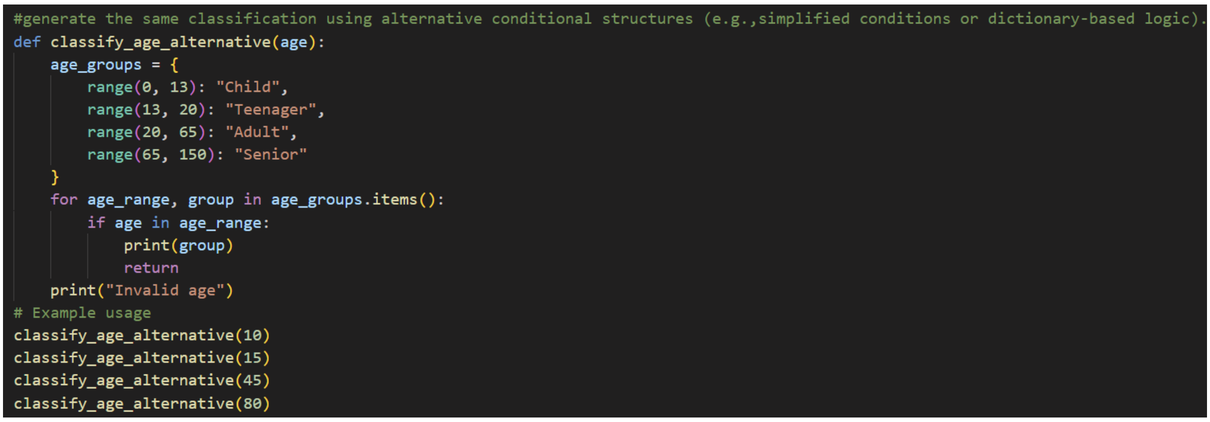
- A Python function that classifies age into appropriate groups.
- Clear and correct conditional logic.
- Explanation of how the conditions work.

Prompt - #generate nested if-elif-else conditional statements to classify age groups (e.g., child, teenager, adult, senior).



```
#generate nested if-elif-else conditional statements to classify age groups (e.g., child, teenager, adult, senio
def classify_age(age):
    if age < 0:
        print("Invalid age")
    elif age < 13:
        print("Child")
    elif age < 20:
        print("Teenager")
    elif age < 65:
        print("Adult")
    else:
        print("Senior")

# Example usage
classify_age(8)
classify_age(16)
classify_age(30)
classify_age(70)
```

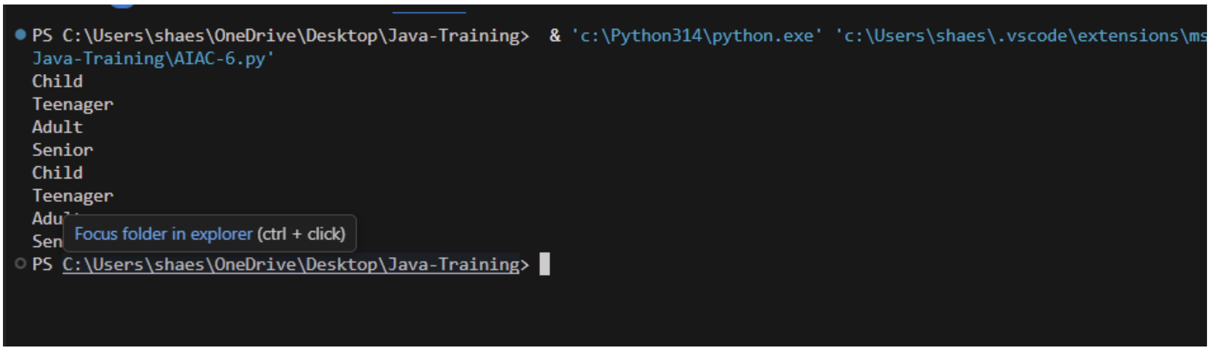
Prompt – 2: #generate the same classification using alternative conditional structures (e.g.,simplified conditions or dictionary-based logic).



```
#generate the same classification using alternative conditional structures (e.g.,simplified conditions or dictionary-based logic).
def classify_age_alternative(age):
    age_groups = {
        range(0, 13): "Child",
        range(13, 20): "Teenager",
        range(20, 65): "Adult",
        range(65, 150): "Senior"
    }
    for age_range, group in age_groups.items():
        if age in age_range:
            print(group)
            return
    print("Invalid age")
# Example usage
classify_age_alternative(10)
classify_age_alternative(15)
classify_age_alternative(45)
classify_age_alternative(80)
```

Analysis : The first function, classify_age, uses nested if-elif-else statements to classify age groups based on the provided age. The second function, classify_age_alternative, uses a dictionary to map age ranges to their corresponding age groups, iterating through the dictionary to find the appropriate classification.Both functions correctly classify ages into the specified groups and handle invalid ages. The code is clear, concise, and follows standard Python conventions.The generated code is correct and meets the requirements specified in the prompt.

Output –



```
PS C:\Users\shaes\OneDrive\Desktop\Java-Training>  & 'c:\Python314\python.exe' 'c:\Users\shaes\.vscode\extensions\ms
Java-Training\AIAC-6.py'
Child
Teenager
Adult
Senior
Child
Teenager
Adu
Sen
PS C:\Users\shaes\OneDrive\Desktop\Java-Training>
```

Task Description #4: For and While Loops (Sum of First n Numbers)

Scenario

You need to calculate the sum of the first n natural numbers.

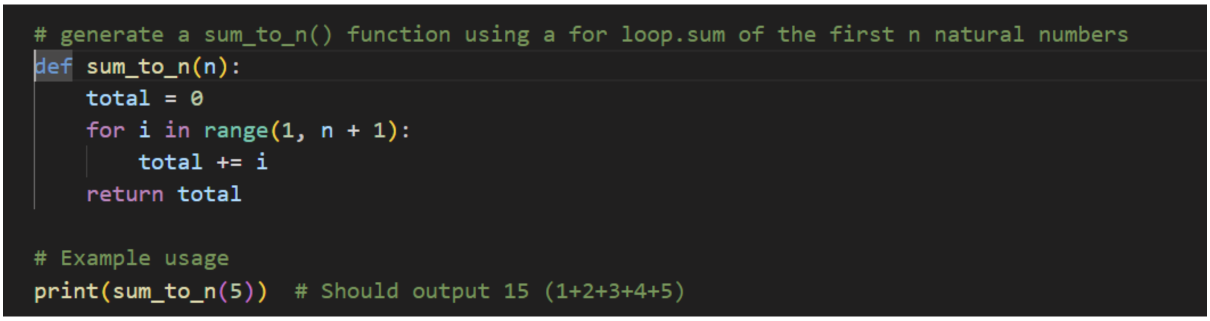
Task

- Use AI assistance to generate a sum_to_n() function using a for loop.
- Analyze the generated code.
- Ask the AI to suggest an alternative implementation using a while loop or a mathematical formula.

Expected Output #4

- Python function to compute the sum of first n numbers.
- Correct output for sample inputs.
- Explanation and comparison of different approaches.

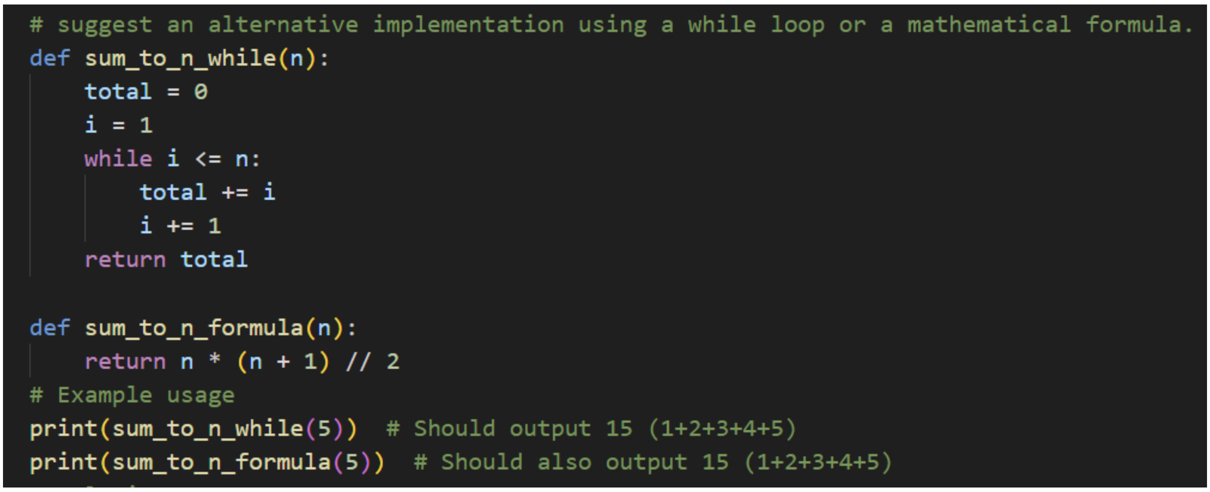
Prompt - # generate a sum_to_n() function using a for loop.sum of the first n natural numbers



```python
# generate a sum_to_n() function using a for loop.sum of the first n natural numbers
def sum_to_n(n):
    total = 0
    for i in range(1, n + 1):
        total += i
    return total

# Example usage
print(sum_to_n(5))  # Should output 15 (1+2+3+4+5)
```

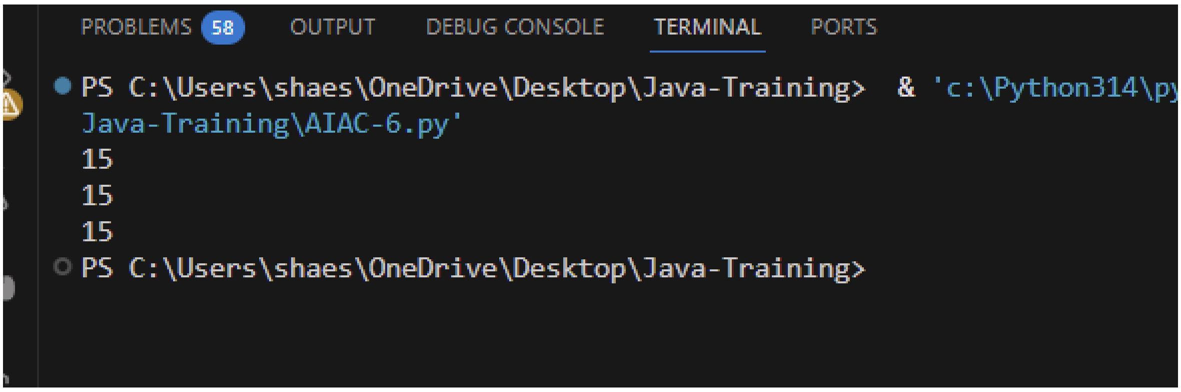
Prompt – 2:

# suggest an alternative implementation using a while loop or a mathematical formula**.**



```
# suggest an alternative implementation using a while loop or a mathematical formula.
def sum_to_n_while(n):
    total = 0
    i = 1
    while i <= n:
        total += i
        i += 1
    return total

def sum_to_n_formula(n):
    return n * (n + 1) // 2
# Example usage
print(sum_to_n_while(5))  # Should output 15 (1+2+3+4+5)
print(sum_to_n_formula(5))  # Should also output 15 (1+2+3+4+5)
```

Output –



```
PS C:\Users\shaes\OneDrive\Desktop\Java-Training>  & 'c:\Python314\p
Java-Training\AIAC-6.py'
15
15
15
PS C:\Users\shaes\OneDrive\Desktop\Java-Training>
```

#Analysis:

# The first function, sum_to_n, uses a for loop to calculate the sum of the first n natural numbers by iterating from 1 to n and accumulating the total.

# The second function, sum_to_n_while, achieves the same result using a while loop to iterate through the numbers and accumulate the total.

# The third function, sum_to_n_formula, uses the mathematical formula n(n + 1) / 2 to calculate the sum directly without iteration.

# All three functions correctly compute the sum of the first n natural numbers and are clear and concise.

# The generated code is correct and meets the requirements specified in the prompt.

Task Description #5: Classes (Bank Account Class)

Scenario - You are designing a basic banking application.

Task

- Use AI tools to generate a Bank Account class with methods such as deposit(), withdraw(), and check_balance().

- Analyze the AI-generated class structure and logic.

- Add meaningful comments and explain the working of the code.

Expected Output #5

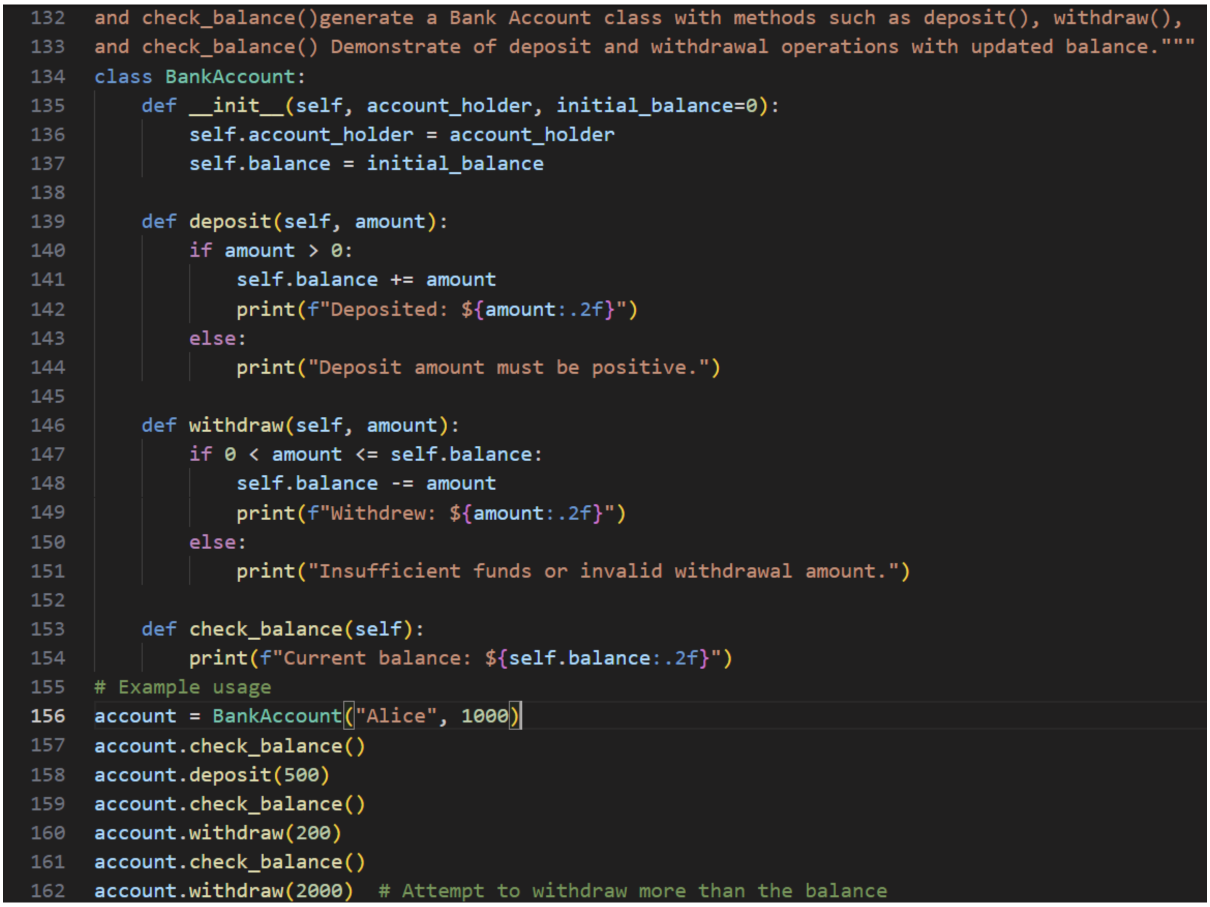
- Complete Python Bank Account class.

- Demonstration of deposit and withdrawal operations with updated balance.

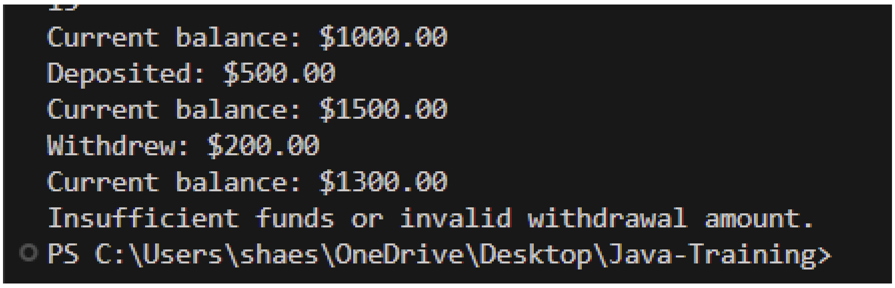
- Well-commented code with a clear explanation.

Prompt –

generate a Bank Account class with methods such as deposit(), withdraw(),and check_balance()generate a Bank Account class with methods such as deposit(), withdraw(), and check_balance() Demonstrate of deposit and withdrawal operations with updated balance.



```python
and check_balance()generate a Bank Account class with methods such as deposit(), withdraw(),
and check_balance() Demonstrate of deposit and withdrawal operations with updated balance."""
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}")
# Example usage
account = BankAccount("Alice", 1000)
account.check_balance()
account.deposit(500)
account.check_balance()
account.withdraw(200)
account.check_balance()
account.withdraw(2000)  # Attempt to withdraw more than the balance
```



```
Current balance: $1000.00
Deposited: $500.00
Current balance: $1500.00
Withdrew: $200.00
Current balance: $1300.00
Insufficient funds or invalid withdrawal amount.
PS C:\Users\shaes\OneDrive\Desktop\Java-Training>
```

# Analysis:

# The generated code correctly defines a BankAccount class with the required attributes and methods.

# The constructor initializes the account holder's name and initial balance.

# The deposit() method adds funds to the account, the withdraw() method deducts funds while

# checking for sufficient balance, and the check_balance() method displays the current balance.

# The example usage demonstrates the functionality of the class effectively.

# The code is clear and follows standard Python conventions.

# The generated code is correct and meets the requirements specified in the prompt.