AI Assisted Coding

Assignment 6.3

Name: Dinesh

Hallticket:2303A52329

Task_01:

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:

""" assume you are a developer working on creating a student information module.
Your task is to implement the following functionality:
create a student class with attributes
name:String
rollNumber: unique integer
branch : string
create a method to display_details to print student information in a proper manner
so i want the expected output
A py class with a __init____and display_details method and aslo a sample obj creation and output displayed on console """

Code:

```
A py class with a __init__ and display_details method and also a sample obj creation and output displayed on c
class Student:
    def __init__(self, name: str, rollNumber: int, branch: str):
        self.name = name
        self.rollNumber = rollNumber
        self.branch = branch

    def display_details(self):
        print(f"Student Name: {self.name}")
        print(f"Roll Number: {self.rollNumber}")
        print(f"Branch: {self.branch}")
# Sample object creation and displaying details
name = input("Enter student name: ")
rollNumber = int(input("Enter roll number: "))
branch = input("Enter branch: ")
student = Student(name, rollNumber, branch)
student.display_details()
```

Output:

```
PS C:\Users\Vivek\OneDrive\Desktop\3-2\AI-Assisted-Coding> python -u "c:\Users
Enter student name: Vivek
Enter student name: Vivek
Enter roll number: 2009
Enter branch: CSE
Student Name: Vivek
Roll Number: 2009
Branch: CSE
PS C:\Users\Vivek\OneDrive\Desktop\3-2\AI-Assisted-Coding>
```

Justification:

The Student class is designed to encapsulate student information with clear attributes and methods. The __init__method initializes the object's state, ensuring that each student has a name, roll number, and branch. The display_details method provides a straightforward way to output the student's information in a readable format. By prompting the user for input, we ensure that the class can be used interactively, making it versatile for various applications. This design promotes code organization, readability, and ease of maintenance.

Task 02:
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.
• Analyze 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:
Assume you are a developer developing a mode to make the multiples of a number by using python so your task it to:
create a function that takes an interger from user and print the multiples of that number upto 10 multiples using a loop
for better understanding i want to use the following program with 2 types of loops for loop and while loop and compare the both code

Code:

```
def print_multiples_for_loop(number):
    print(f"Multiples of {number} using for loop:")
    for i in range(1, 11):
        print(f"{number} × {i} = {number * i}")
def print_multiples_while_loop(number):
    print(f"Multiples of {number} using while loop:")
    i = 1
    while i <= 10:
        print(f"{number} × {i} = {number * i}")
        i += 1
```

BLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL 1 PORTS POSTMAN CONSOLE

Output:

```
PS C:\Users\Vivek\OneDrive\Desktop\3-2\AI-Assisted
Enter an integer to print its multiples: 10
Multiples of 10 using for loop:
10 × 1 = 10
10 × 2 = 20
10 × 3 = 30
10 × 4 = 40
10 × 5 = 50
10 × 6 = 60
10 × 7 = 70
10 × 8 = 80
10 × 9 = 90
10 × 10 = 100
Multiples of 10 using while loop:
10 × 1 = 10
10 × 2 = 20
10 × 3 = 30
10 × 4 = 40
10 × 5 = 50
10 × 6 = 60
10 × 7 = 70
10 × 8 = 80
10 × 9 = 90
10 × 10 = 100
PS C:\Users\Vivek\OneDrive\Desktop\3-2\AI-Assisted
n*!  ⊗ 0 ⚠ 0 ⓘ 1
```

Comparison:

Aspect	For Loop	While Loop
Readability	Clear and concise for fixed iterations	Slightly less clear due to manual control of loop
Control Flow	Implicit control with range function	Explicit control with initialization and increment
Use Case	Best for known iteration count	Best for unknown iterations or complex conditions
Performance	Slightly faster due to less overhead	Slightly slower due to additional control logic
Error-Prone	Less prone to infinite loops due to fixed iteration count	More prone to infinite loops if not properly managed

Justification:

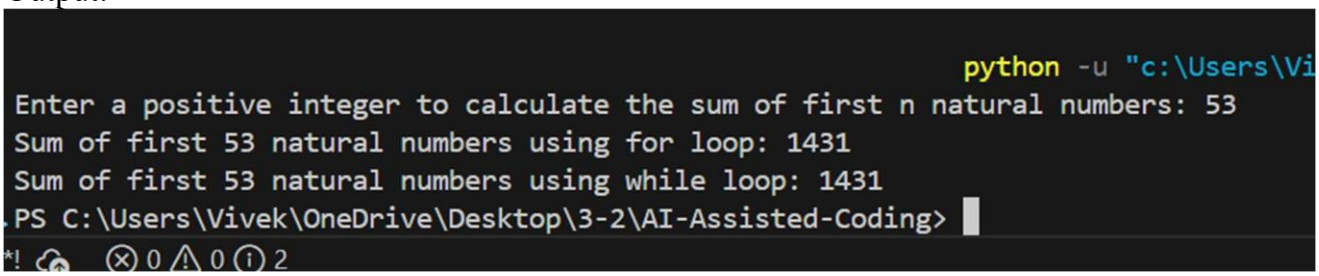
The code defines two functions to print the multiples of a given integer using different looping constructs: a for loop and a while loop. The for loop version is concise and straightforward, iterating over a predefined range to generate multiples. In contrast, the while loop version offers more control over the iteration process, explicitly managing the loop variable. Both approaches effectively achieve the same outcome, demonstrating the versatility of Python's looping mechanisms. This comparison highlights the strengths of each method, allowing developers to choose based on their specific needs and preferences.

Task 04:
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:
"""Assume you are a developer working on creating a module to compare for loop and while loop in python
so your task is to implement the following functionality:
create a function that takes an integer from user
you need to calculate the 1st sum of n natural numbers using for loop and while loop both and compare the both code
First i want to execute the code using for loop and then using while loop and then compare the both code """
Code:

```python
def sum_natural_numbers_for_loop(n):
    total = 0
    for i in range(1, n + 1):
        total += i
    return total
def sum_natural_numbers_while_loop(n):
    total = 0
    i = 1
    while i <= n:
        total += i
        i += 1
    return total
# Taking input from user
n = int(input("Enter a positive integer to calculate the sum of first n natural numbers: "))
sum_for_loop = sum_natural_numbers_for_loop(n)
sum_while_loop = sum_natural_numbers_while_loop(n)
print(f"Sum of first {n} natural numbers using for loop: {sum_for_loop}")
print(f"Sum of first {n} natural numbers using while loop: {sum_while_loop}")
```

Output:



Comparison:

comparison points of For Loop vs While Loop for Summing Natural Numbers
for loop :
1. Readability: For loops are generally more readable for fixed iterations.
2. Control Flow: Implicit control with range function.
3. Use Case: Best for known iteration count.
4. Performance: Slightly faster due to less overhead.
5. Error-Prone: Less prone to infinite loops due to fixed iteration count.
while loop :
1. Readability: Slightly less clear due to manual control of loop variable.
2. Control Flow: Explicit control with initialization and increment.
3. Use Case: Best for unknown iterations or complex conditions.
4. Performance: Slightly slower due to additional control logic.
5. Error-Prone: More prone to infinite loops if not properly managed.

Task_03

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:

assume you are building a simple python module to classify the ages of people into different categories based on their age.
using conditionals statements we need to classify the ages into child, teenager, adult, and senior citizen.
"""

Code:

```
def classify_age(age):
    if age < 0:
        return "Invalid age"
    elif age <= 12:
        return "Child"
    elif age <= 19:
        return "Teenager"
    elif age <= 59:
        return "Adult"
    else:
        return "Senior Citizen"
# Example usage
age = int(input("Enter age: "))
category = classify_age(age)
print(f"The person is classified as: {category}")
```

Output:

```
PS C:\Users\Vivek\OneDrive\Desktop\3-2\AI-Assisted-Coding> python -u
Enter age: 22
The person is classified as: Adult
PS C:\Users\Vivek\OneDrive\Desktop\3-2\AI-Assisted-Coding>
in*!  ⊗ 0 ⚠ 0 ⓘ 9
```

Simplified code:

```
# print(f"The person is classified as: {category}")
""" give another implementation of the same code using nested if else statements """
def classify_age_nested(age):
    if age < 0:
        return "Invalid age"
    else:
        if age <= 12:
            return "Child"
        else:
            if age <= 19:
                return "Teenager"
            else:
                if age <= 59:
                    return "Adult"
                else:
                    return "Senior Citizen"
# Example usage
age = int(input("Enter age: "))
category = classify_age_nested(age)
print(f"The person is classified as: {category}")
```

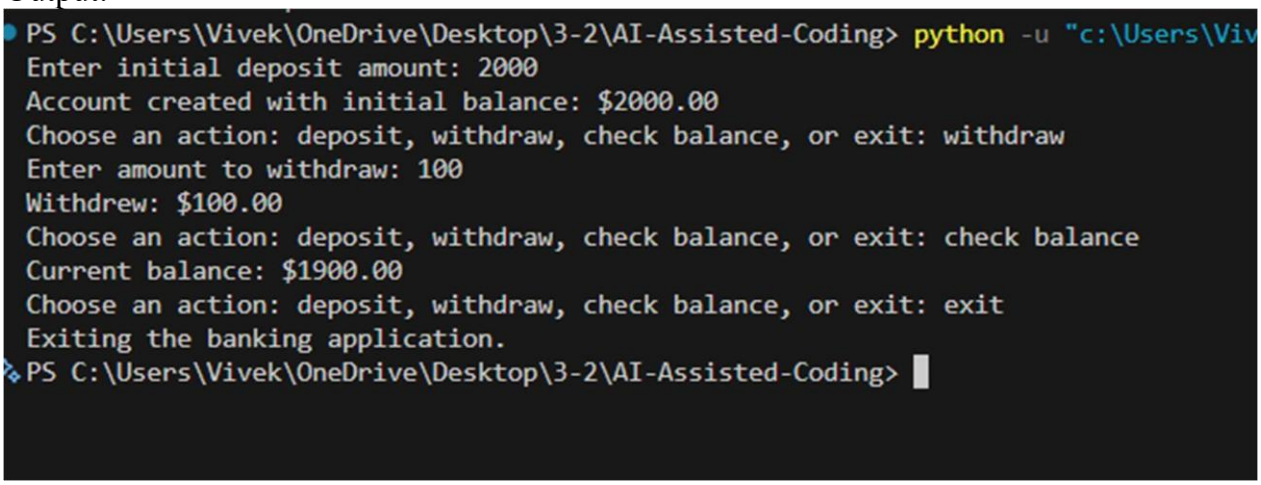
Task Description #5: Classes (Bank Account Class)
You are designing a basic banking application.
Task
• Use AI tools to generate a Bank Account class with methods such as deposit(), withdraw(), and check_balance().
• Analyze the AI-generated class structure and logic.
• Add meaningful comments and explain 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:
"""Act as a developer creating a module that your are developing a banking application.
Your task is to implement the following functionality:
create a basic function that such as deposit() withdraw() and check balance().
also add the step by step explanation of the code"""

Code:

Code:

```
class BankAccount:
    def __init__(self, initial_balance=0):
        # Initialize the bank account with an initial balance
        self.balance = initial_balance
    def deposit(self, amount):
        # Add the specified amount to the account balance
        if amount > 0:
            self.balance += amount
            print(f"Deposited: ${amount:.2f}")
        else:
            print("Deposit amount must be positive.")
    def withdraw(self, amount):
        # Subtract the specified amount from the account balance if sufficient funds are available
        if amount > 0:
            if amount <= self.balance:
                self.balance -= amount
                print(f"Withdrew: ${amount:.2f}")
            else:
                print("Insufficient funds for this withdrawal.")
        else:
            print("Withdrawal amount must be positive.")
    def check_balance(self):
        # Return the current account balance
        return self.balance
# Example usage
initial_deposit = float(input("Enter initial deposit amount: "))
account = BankAccount(initial_deposit)
print(f"Account created with initial balance: ${account.check_balance():.2f}")
while True:
    action = input("Choose an action: deposit, withdraw, check balance, or exit: ").lower()
    if action == "deposit":
        amount = float(input("Enter amount to deposit: "))
        account.deposit(amount)
    elif action == "withdraw":
        amount = float(input("Enter amount to withdraw: "))
        account.withdraw(amount)
    elif action == "check balance":
        print(f"Current balance: ${account.check_balance():.2f}")
    elif action == "exit":
        print("Exiting the banking application.")
        break
    else:
```

Output:



Justification:

The BankAccount class encapsulates the functionality of a simple banking application. The _init__method initializes the account with an optional initial balance. The deposit method allows users to add funds to their account, ensuring that only positive amounts are accepted. The withdraw method enables users to remove funds, checking for sufficient balance before proceeding. The check_balance method provides a way to view the current account balance. The example usage demonstrates how a user can interact with the banking application, performing deposits, withdrawals, and balance checks in a loop until they choose to exit. This design promotes clarity, usability, and basic error handling for common banking operations.