ASSIGNMENT-6.3

2303A51546
B-10

TASK-1:

Prompt :

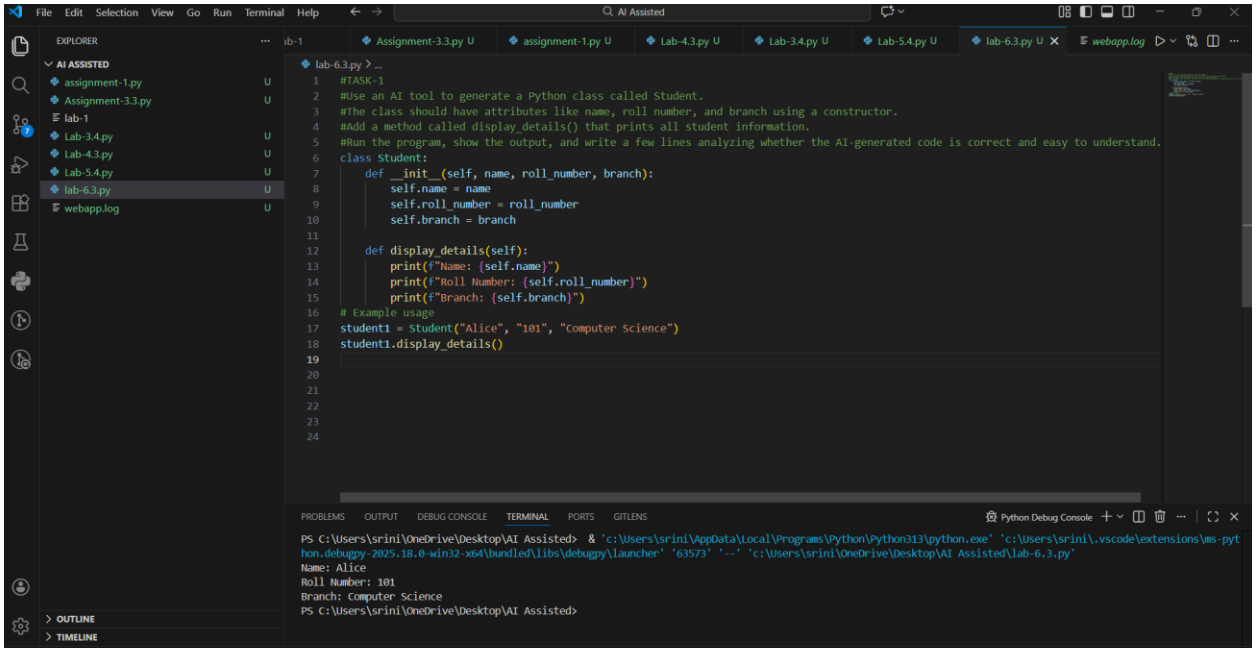
Use an AI tool to generate a Python class called Student.
The class should have attributes like name, roll number, and branch using a constructor.
Add a method called display_details() that prints all student information.
Run the program, show the output, and write a few lines analyzing whether the AI-generated code is correct and easy to understand.

Code :

```
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}")
# Example usage
student1 = Student("Alice", "101", "Computer Science")
student1.display_details()
```

Output :



```python
#TASK-1
#Use an AI tool to generate a Python class called Student.
#The class should have attributes like name, roll number, and branch using a constructor.
#Add a method called display_details() that prints all student information.
#Run the program, show the output, and write a few lines analyzing whether the AI-generated code is correct and easy to understand.
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}")
# Example usage
student1 = Student("Alice", "101", "Computer Science")
student1.display_details()
```

```
PS C:\Users\srini\OneDrive\Desktop\AI Assisted>  & 'c:\Users\srini\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\srini\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '63573' '--' 'c:\Users\srini\OneDrive\Desktop\AI Assisted\lab-6.3.py'
Name: Alice
Roll Number: 101
Branch: Computer Science
PS C:\Users\srini\OneDrive\Desktop\AI Assisted>
```

Analysis :

The AI-generated Student class is simple and easy to understand. It correctly uses a constructor (__init__) to store the student's name, roll number, and branch as attributes. The display_details() method prints all the student information clearly. The object creation and method call work properly, and the output matches the entered data. Overall, the code is correct, readable, and suitable for beginners learning about classes.

TASK-2:

Prompt :

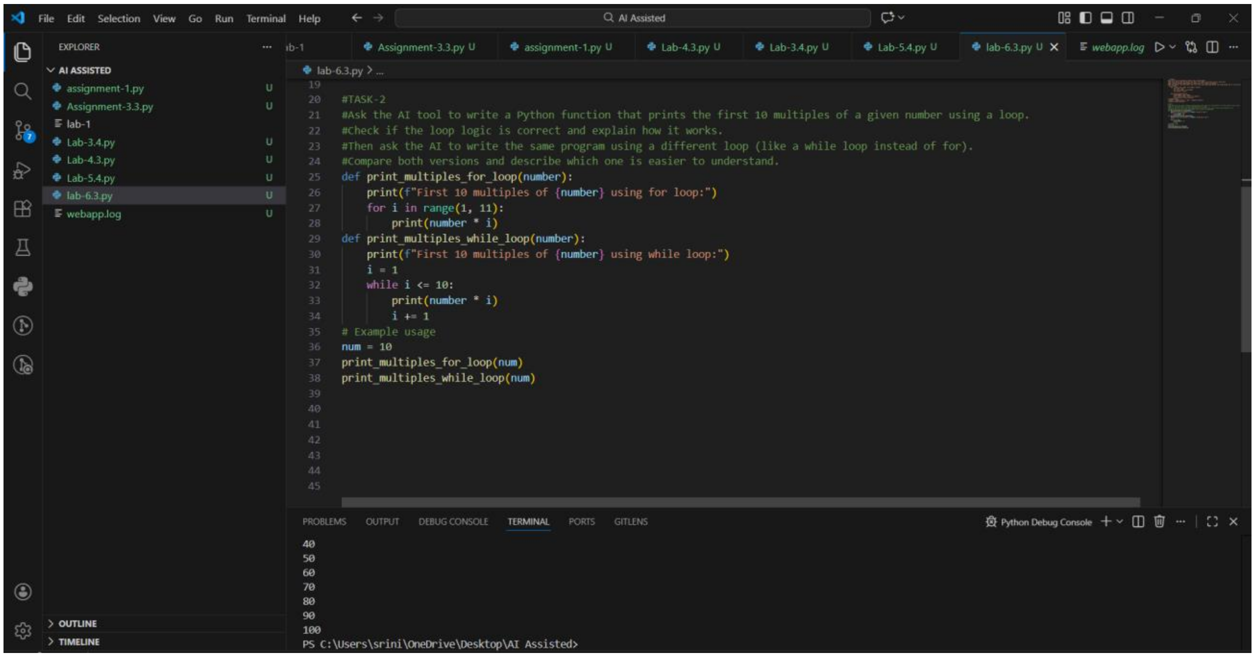
Ask the AI tool to write a Python function that prints the first 10 multiples of a given number using a loop.
Check if the loop logic is correct and explain how it works.
Then ask the AI to write the same program using a different loop (like a while loop instead of for).
Compare both versions and describe which one is easier to understand.

Code :

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

```
        i += 1
# Example usage
num = 10
print_multiples_for_loop(num)
print_multiples_while_loop(num)
```

Output :



```
#TASK-2
#Ask the AI tool to write a Python function that prints the first 10 multiples of a given number using a loop.
#Check if the loop logic is correct and explain how it works.
#Then ask the AI to write the same program using a different loop (like a while loop instead of for).
#Compare both versions and describe which one is easier to understand.
def print_multiples_for_loop(number):
    print(f"First 10 multiples of {number} using for loop:")
    for i in range(1, 11):
        print(number * i)
def print_multiples_while_loop(number):
    print(f"First 10 multiples of {number} using while loop:")
    i = 1
    while i <= 10:
        print(number * i)
        i += 1
# Example usage
num = 10
print_multiples_for_loop(num)
print_multiples_while_loop(num)
```

```
40
50
60
70
80
90
100
PS C:\Users\srini\OneDrive\Desktop\AI Assisted>
```

Analysis :

The AI correctly generated two functions using different loop structures. The for loop version is shorter and easier to read because the number of iterations (10 multiples) is clearly defined in the range. The while loop version also works correctly but requires manual control of the loop variable, which makes it slightly longer. Both methods give the same output, but the for loop is more convenient when the number of repetitions is known.

TASK-3:

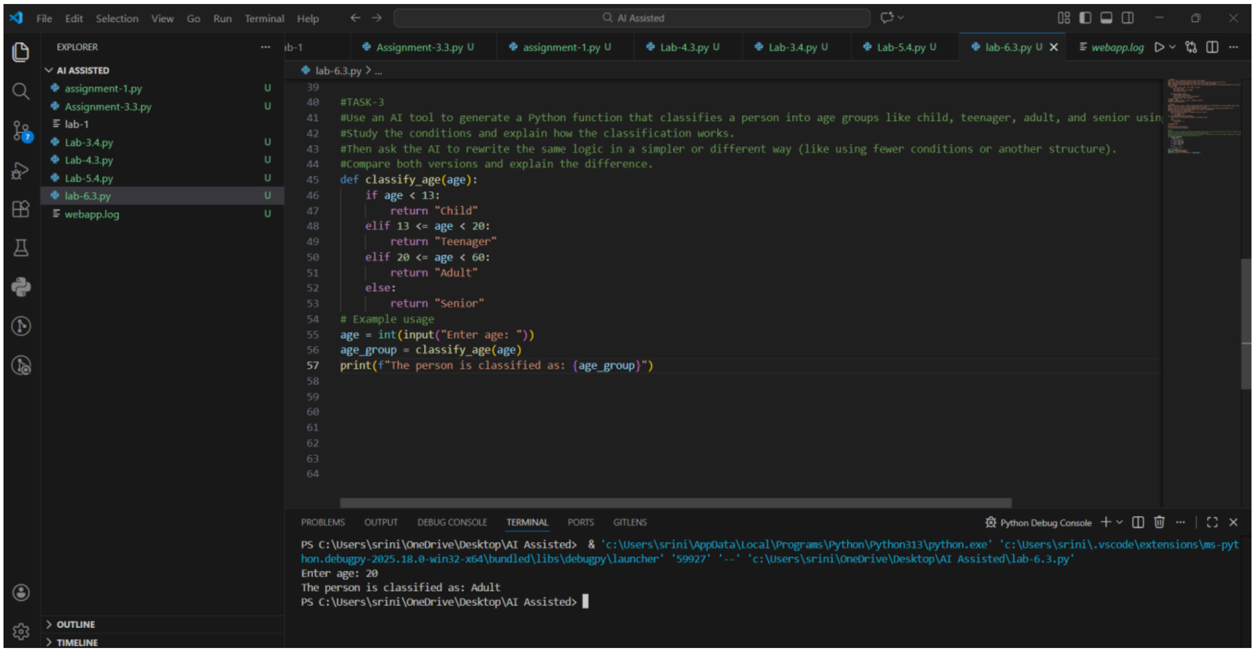
Prompt :

Use an AI tool to generate a Python function that classifies a person into age groups like child, teenager, adult, and senior using if-elif-else conditions.
Study the conditions and explain how the classification works.
Then ask the AI to rewrite the same logic in a simpler or different way (like using fewer conditions or another structure).
Compare both versions and explain the difference.

Code :

```
def classify_age(age):
    if age < 13:
        return "Child"
    elif 13 <= age < 20:
        return "Teenager"
    elif 20 <= age < 60:
        return "Adult"
    else:
        return "Senior"
# Example usage
age = int(input("Enter age: "))
age_group = classify_age(age)
print(f"The person is classified as: {age_group}")
```

Output :



```
#TASK-3
#Use an AI tool to generate a Python function that classifies a person into age groups like child, teenager, adult, and senior usin
#Study the conditions and explain how the classification works.
#Then ask the AI to rewrite the same logic in a simpler or different way (like using fewer conditions or another structure).
#Compare both versions and explain the difference.
def classify_age(age):
    if age < 13:
        return "Child"
    elif 13 <= age < 20:
        return "Teenager"
    elif 20 <= age < 60:
        return "Adult"
    else:
        return "Senior"
# Example usage
age = int(input("Enter age: "))
age_group = classify_age(age)
print(f"The person is classified as: {age_group}")
```

```
PS C:\Users\srini\OneDrive\Desktop\AI Assisted>  & 'c:\Users\srini\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\srini\.vscode\extensions\ms-pyt
hon.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '59927' '--' 'c:\Users\srini\OneDrive\Desktop\AI Assisted\lab-6.3.py'
Enter age: 20
The person is classified as: Adult
PS C:\Users\srini\OneDrive\Desktop\AI Assisted> 
```

Analysis :

The AI-generated conditional statements correctly divide ages into groups: child, teenager, adult, and senior. The conditions are arranged in a logical order from smallest age to largest. The use of elif avoids unnecessary checks once a condition is met, making the code efficient. The logic is clear and easy to modify if age ranges need changes. Overall, the classification system works accurately.

TASK-4:

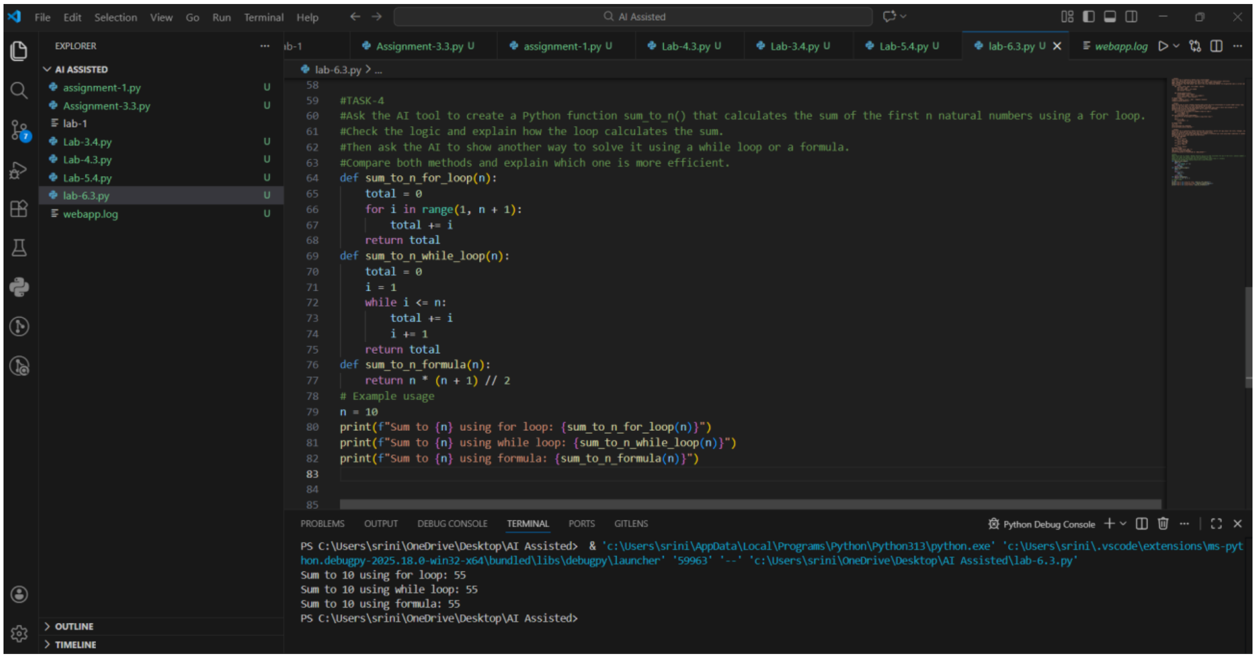
Prompt :

Ask the AI tool to create a Python function sum_to_n() that calculates the sum of the first n natural numbers using a for loop.
Check the logic and explain how the loop calculates the sum.
Then ask the AI to show another way to solve it using a while loop or a formula.
Compare both methods and explain which one is more efficient.

Code :

```
def sum_to_n_for_loop(n):
    total = 0
    for i in range(1, n + 1):
        total += i
    return total
def sum_to_n_while_loop(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
n = 10
print(f"Sum to {n} using for loop: {sum_to_n_for_loop(n)}")
print(f"Sum to {n} using while loop: {sum_to_n_while_loop(n)}")
print(f"Sum to {n} using formula: {sum_to_n_formula(n)}")
```

Output :



```
#TASK-4
#Ask the AI tool to create a Python function sum_to_n() that calculates the sum of the first n natural numbers using a for loop.
#Check the logic and explain how the loop calculates the sum.
#Then ask the AI to show another way to solve it using a while loop or a formula.
#Compare both methods and explain which one is more efficient.
def sum_to_n_for_loop(n):
    total = 0
    for i in range(1, n + 1):
        total += i
    return total
def sum_to_n_while_loop(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
n = 10
print(f"Sum to {n} using for loop: {sum_to_n_for_loop(n)}")
print(f"Sum to {n} using while loop: {sum_to_n_while_loop(n)}")
print(f"Sum to {n} using formula: {sum_to_n_formula(n)}")
```

```
PS C:\Users\srini\OneDrive\Desktop\AI Assisted>  & 'c:\Users\srini\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\srini\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '59963' '--' 'c:\Users\srini\OneDrive\Desktop\AI Assisted\lab-6.3.py'
Sum to 10 using for loop: 55
Sum to 10 using while loop: 55
Sum to 10 using formula: 55
PS C:\Users\srini\OneDrive\Desktop\AI Assisted>
```

Analysis :

The AI provided three different approaches: for loop, while loop, and mathematical formula. Both loops correctly add numbers from 1 to n step by step. However, the formula method is the most efficient because it calculates the result in one step without looping. This shows how AI can suggest optimized solutions. All methods give the same result, but the formula is faster and cleaner.

TASK-5:

Prompt :

Use an AI tool to generate a Python class called BankAccount with methods like deposit(), withdraw(), and check_balance().
Test the class by performing deposit and withdrawal operations.
Add comments to the code and explain how each method works.
Finally, analyze whether the AI-generated class is logically correct and well structured.

Code :

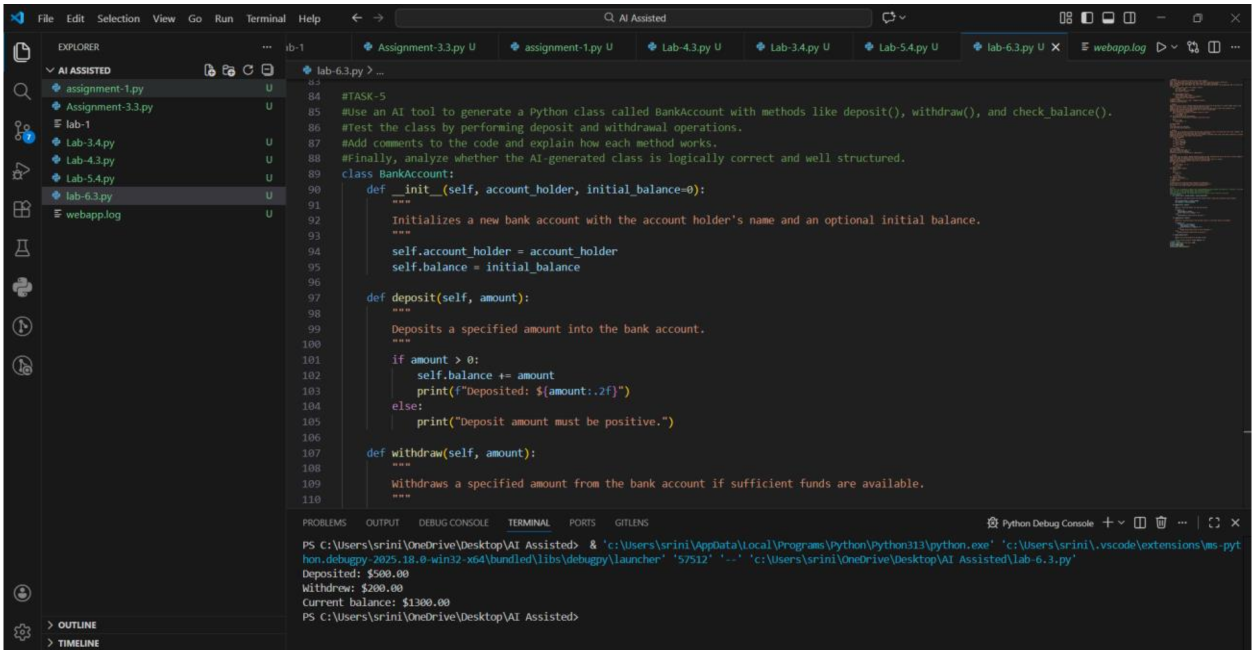
```
class BankAccount:
    def __init__(self, account_holder, initial_balance=0):
        """
        Initializes a new bank account with the account holder's name and an optional
initial balance.
        """
        self.account_holder = account_holder
        self.balance = initial_balance

    def deposit(self, amount):
        """
        Deposits a specified amount into the bank account.
        """
        if amount > 0:
            self.balance += amount
            print(f"Deposited: ${amount:.2f}")
        else:
            print("Deposit amount must be positive.")

    def withdraw(self, amount):
        """
        Withdraws a specified amount from the bank account 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):
        """
        Returns the current balance of the bank account.
        """
        return f"Current balance: ${self.balance:.2f}"
# Example usage
account = BankAccount("John Doe", 1000)
account.deposit(500)
account.withdraw(200)
print(account.check_balance())
```

Output :

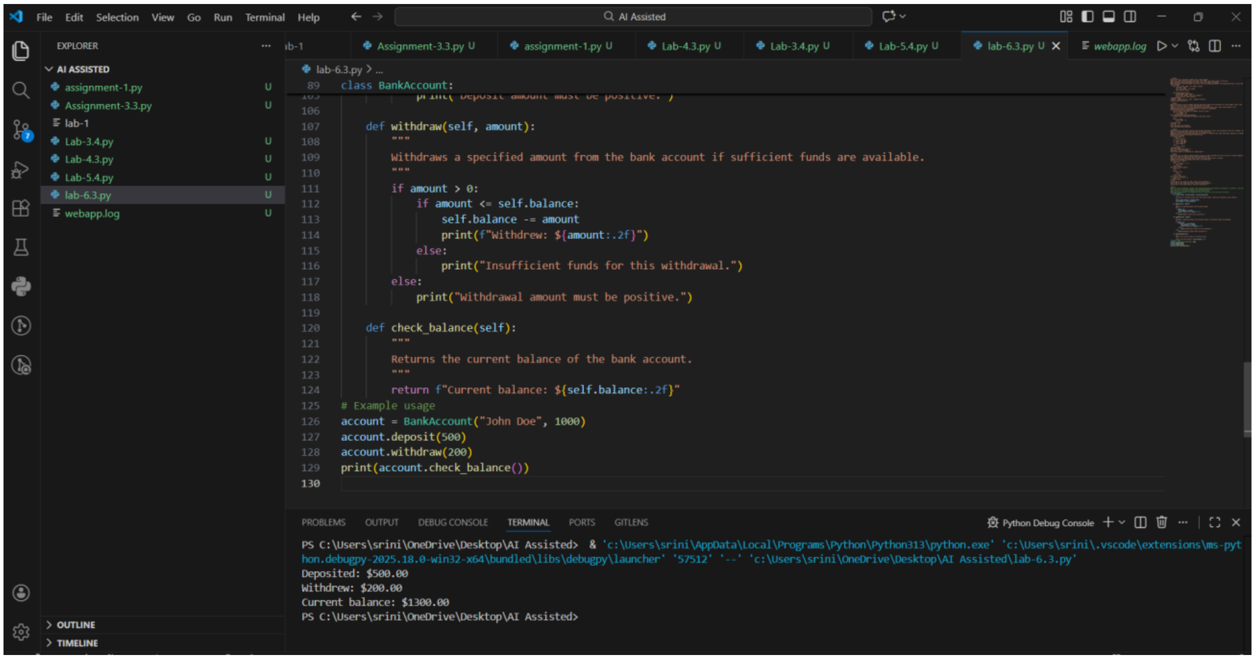


```python
#TASK-5
#Use an AI tool to generate a Python class called BankAccount with methods like deposit(), withdraw(), and check_balance().
#Test the class by performing deposit and withdrawal operations.
#Add comments to the code and explain how each method works.
#Finally, analyze whether the AI-generated class is logically correct and well structured.
class BankAccount:
    def __init__(self, account_holder, initial_balance=0):
        """
        Initializes a new bank account with the account holder's name and an optional initial balance.
        """
        self.account_holder = account_holder
        self.balance = initial_balance

    def deposit(self, amount):
        """
        Deposits a specified amount into the bank account.
        """
        if amount > 0:
            self.balance += amount
            print(f"Deposited: ${amount:.2f}")
        else:
            print("Deposit amount must be positive.")

    def withdraw(self, amount):
        """
        Withdraws a specified amount from the bank account if sufficient funds are available.
        """
```

```
PS C:\Users\srini\OneDrive\Desktop\AI Assisted>  & 'c:\Users\srini\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\srini\.vscode\extensions\ms-pyt
hon.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '57512' '--' 'c:\Users\srini\OneDrive\Desktop\AI Assisted\lab-6.3.py'
Deposited: $500.00
Withdrew: $200.00
Current balance: $1300.00
PS C:\Users\srini\OneDrive\Desktop\AI Assisted>
```



```python
class BankAccount:

    def withdraw(self, amount):
        """
        Withdraws a specified amount from the bank account 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):
        """
        Returns the current balance of the bank account.
        """
        return f"Current balance: ${self.balance:.2f}"
# Example usage
account = BankAccount("John Doe", 1000)
account.deposit(500)
account.withdraw(200)
print(account.check_balance())

```

```
PS C:\Users\srini\OneDrive\Desktop\AI Assisted>  & 'c:\Users\srini\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\srini\.vscode\extensions\ms-pyt
hon.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '57512' '--' 'c:\Users\srini\OneDrive\Desktop\AI Assisted\lab-6.3.py'
Deposited: $500.00
Withdrew: $200.00
Current balance: $1300.00
PS C:\Users\srini\OneDrive\Desktop\AI Assisted>
```

Analysis :

The AI-generated BankAccount class is well structured and realistic. It uses a constructor to store account details and includes separate methods for deposit, withdrawal, and checking balance. Proper conditions are used to prevent negative deposits and withdrawals beyond the balance, which improves reliability. The comments make the code easier to understand. The example usage shows that the balance updates correctly after each transaction. Overall, the design is logical and suitable for a basic banking system.