AI Assisted Coding

Assignment 6.3

Name: Syed Murtaza

Hall ticket no: 2303A51259

Batch no: 19

Task 1: Classes (Student Class)

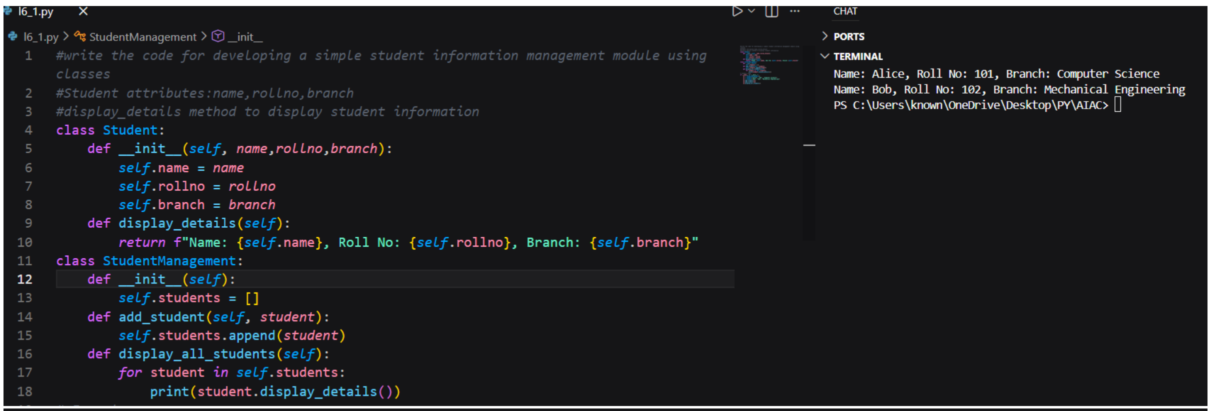
Prompt:

#Write the code for developing a simple student information management module using classes

#Student attributes:name,rollno,branch

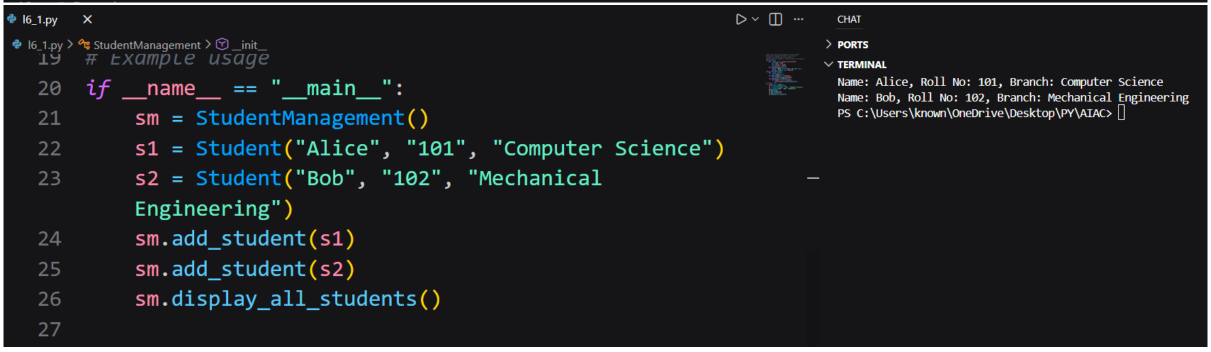
#display_details method to display student information

Code & Output:



```python
#write the code for developing a simple student information management module using classes
#Student attributes:name,rollno,branch
#display_details method to display student information
class Student:
    def __init__(self, name,rollno,branch):
        self.name = name
        self.rollno = rollno
        self.branch = branch
    def display_details(self):
        return f"Name: {self.name}, Roll No: {self.rollno}, Branch: {self.branch}"
class StudentManagement:
    def __init__(self):
        self.students = []
    def add_student(self, student):
        self.students.append(student)
    def display_all_students(self):
        for student in self.students:
            print(student.display_details())
```

```
Name: Alice, Roll No: 101, Branch: Computer Science
Name: Bob, Roll No: 102, Branch: Mechanical Engineering
PS C:\Users\known\OneDrive\Desktop\PY\AIAC> 
```



```python
# Example usage
if __name__ == "__main__":
    sm = StudentManagement()
    s1 = Student("Alice", "101", "Computer Science")
    s2 = Student("Bob", "102", "Mechanical Engineering")
    sm.add_student(s1)
    sm.add_student(s2)
    sm.display_all_students()
```

```
Name: Alice, Roll No: 101, Branch: Computer Science
Name: Bob, Roll No: 102, Branch: Mechanical Engineering
PS C:\Users\known\OneDrive\Desktop\PY\AIAC> 
```

Explanation:

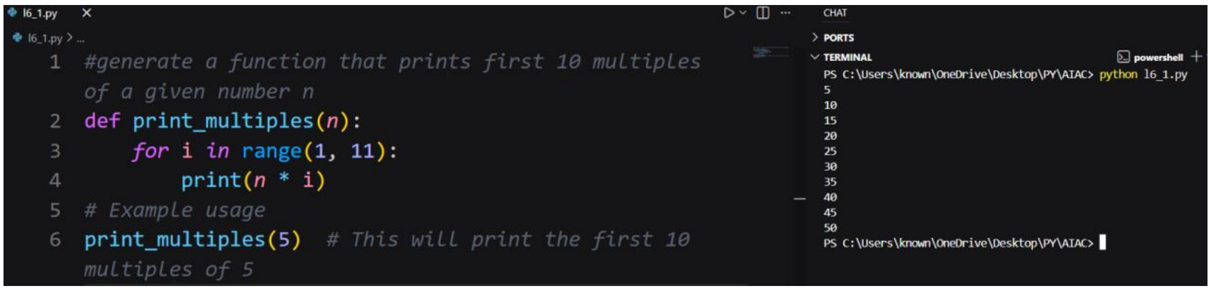
The AI-generated code correctly defines a Student class using object-oriented principles. The constructor initializes student attributes, and the display_details() method prints them clearly. The class structure is simple, readable, and functions correctly when an object is created and executed.

Task 2: Loops (Multiples of a Number)

Prompt:

Generate a function that prints first 10 multiples of a given number n

Code & Output:



```python
#generate a function that prints first 10 multiples of a given number n
def print_multiples(n):
    for i in range(1, 11):
        print(n * i)
# Example usage
print_multiples(5)  # This will print the first 10 multiples of 5
```

```
PS C:\Users\known\OneDrive\Desktop\PY\AIAC> python l6_1.py
5
10
15
20
25
30
35
40
45
50
PS C:\Users\known\OneDrive\Desktop\PY\AIAC>
```

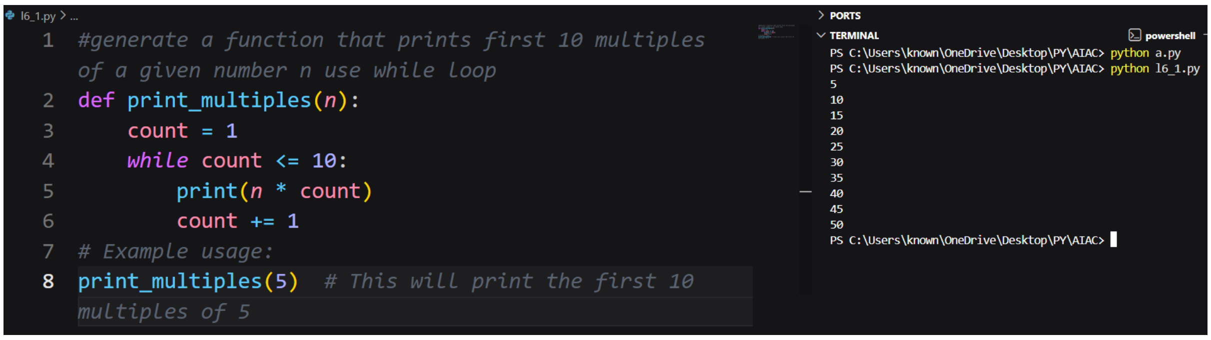
Explanation:

The AI-generated function uses a for loop to iterate from 1 to 10 and prints the multiples of the given number. In each iteration, the loop variable is multiplied by the input number. The loop boundaries are correctly defined, and the logic produces accurate results. This implementation is efficient and readable, making it ideal for tasks with a fixed number of iterations.

Prompt (Alternative Loop):

Generate a function that prints first 10 multiples of a given number n using while loop

Code & Output:



```python
#generate a function that prints first 10 multiples of a given number n use while loop
def print_multiples(n):
    count = 1
    while count <= 10:
        print(n * count)
        count += 1
# Example usage:
print_multiples(5)  # This will print the first 10 multiples of 5
```

```
PS C:\Users\known\OneDrive\Desktop\PY\AIAC> python a.py
PS C:\Users\known\OneDrive\Desktop\PY\AIAC> python l6_1.py
5
10
15
20
25
30
35
40
45
50
PS C:\Users\known\OneDrive\Desktop\PY\AIAC>
```

Explanation:

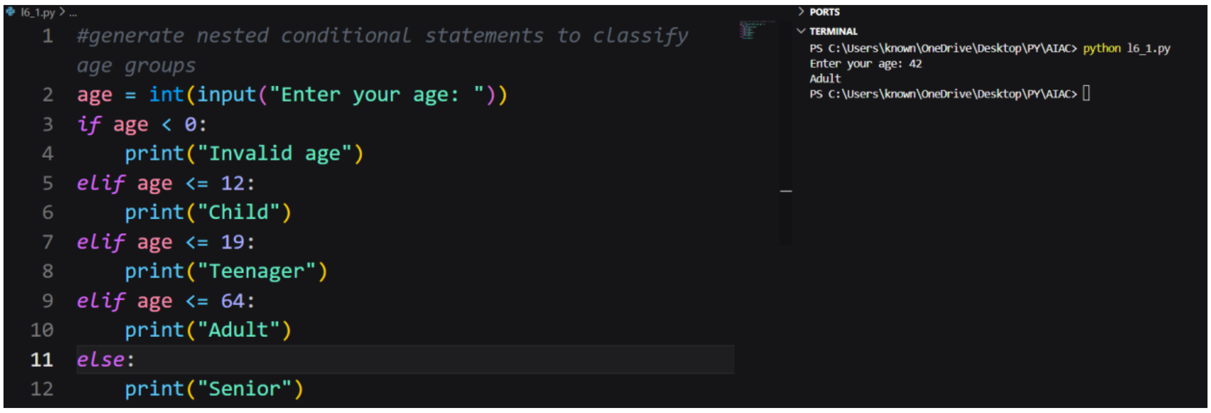
The while-loop version produces the same output as the for-loop version. While loops require manual control of the counter variable, making the for loop slightly cleaner and more readable for fixed iterations.

Task 3: Conditional Statements (Age Classification)

Prompt:

Generate nested conditional statements to classify age groups.

Code & Output:



```python
#generate nested conditional statements to classify
age groups
age = int(input("Enter your age: "))
if age < 0:
    print("Invalid age")
elif age <= 12:
    print("Child")
elif age <= 19:
    print("Teenager")
elif age <= 64:
    print("Adult")
else:
    print("Senior")
```

```
PS C:\Users\known\OneDrive\Desktop\PY\AIAC> python l6_1.py
Enter your age: 42
Adult
PS C:\Users\known\OneDrive\Desktop\PY\AIAC>
```

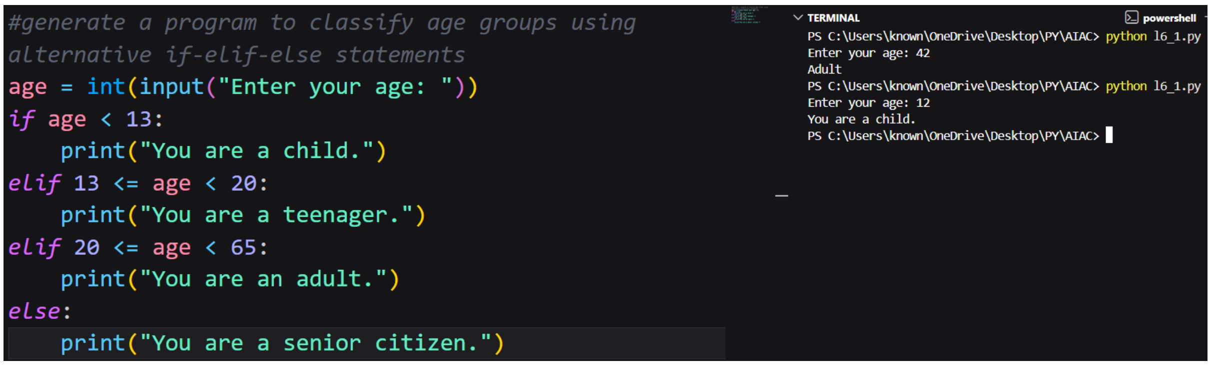
Explanation:

The AI-generated function uses nested if-elif-else conditions to classify age groups. Each condition checks a specific age range in increasing order. The structure ensures that only one category is returned for a given age. The logic is clear, correct, and easy to verify, making the code understandable for beginners and suitable for real-world classification tasks.

Prompt (Alternative Logic):

Generate a program to classify age groups using alternative if-elif-else statements

Code & Output:



```python
#generate a program to classify age groups using
alternative if-elif-else statements
age = int(input("Enter your age: "))
if age < 13:
    print("You are a child.")
elif 13 <= age < 20:
    print("You are a teenager.")
elif 20 <= age < 65:
    print("You are an adult.")
else:
    print("You are a senior citizen.")
```

```
PS C:\Users\known\OneDrive\Desktop\PY\AIAC> python l6_1.py
Enter your age: 42
Adult
PS C:\Users\known\OneDrive\Desktop\PY\AIAC> python l6_1.py
Enter your age: 12
You are a child.
PS C:\Users\known\OneDrive\Desktop\PY\AIAC>
```

Explanation:

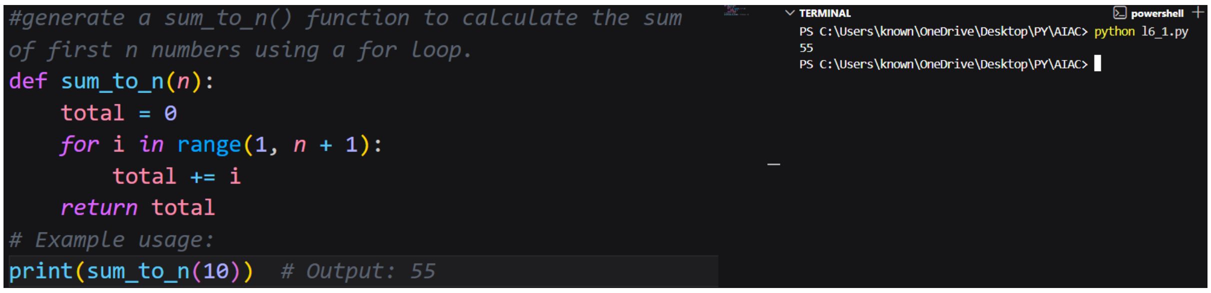
This alternative approach uses a dictionary and a loop to determine the age group. While this method is flexible and scalable, it is more complex than the if-elif-else approach. For simple classification problems, the original conditional structure is more readable and easier to maintain.

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

Prompt:

Generate a Python function to calculate the sum of first n natural numbers using a for loop.

Code & Output:



```python
#generate a sum_to_n() function to calculate the sum
of first n numbers using a for loop.
def sum_to_n(n):
    total = 0
    for i in range(1, n + 1):
        total += i
    return total
# Example usage:
print(sum_to_n(10))  # Output: 55
```

```
TERMINAL
PS C:\Users\known\OneDrive\Desktop\PY\AIAC> python l6_1.py
55
PS C:\Users\known\OneDrive\Desktop\PY\AIAC>
```

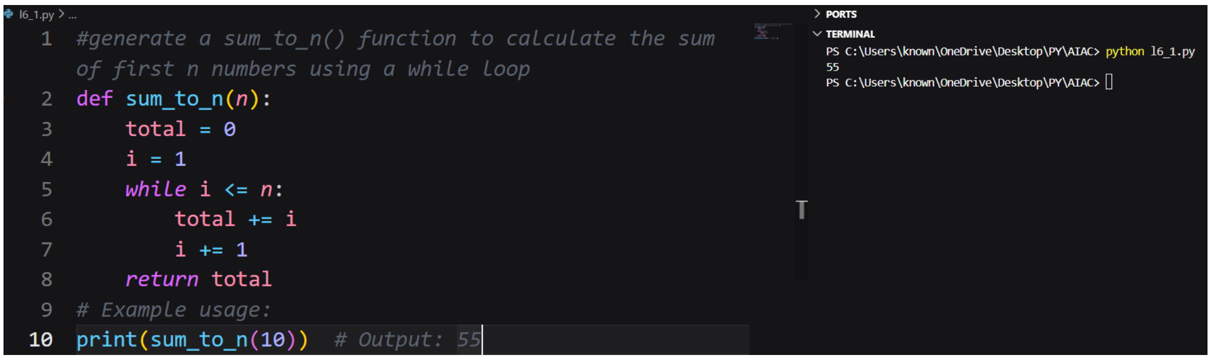
Explanation:

The AI-generated function calculates the sum by iterating through numbers from 1 to n and adding them to a total variable. The loop logic is correct and produces accurate results. This approach is easy to understand and works efficiently for small to moderate values of n.

Prompt (Alternative Loop):

Generate a sum_to_n() function to calculate the sum of first n numbers using a while loop

Code & Output:



```python
#generate a sum_to_n() function to calculate the sum
of first n numbers using a while loop
def sum_to_n(n):
    total = 0
    i = 1
    while i <= n:
        total += i
        i += 1
    return total
# Example usage:
print(sum_to_n(10))  # Output: 55
```

```
PORTS
TERMINAL
PS C:\Users\known\OneDrive\Desktop\PY\AIAC> python l6_1.py
55
PS C:\Users\known\OneDrive\Desktop\PY\AIAC>
```

Explanation:

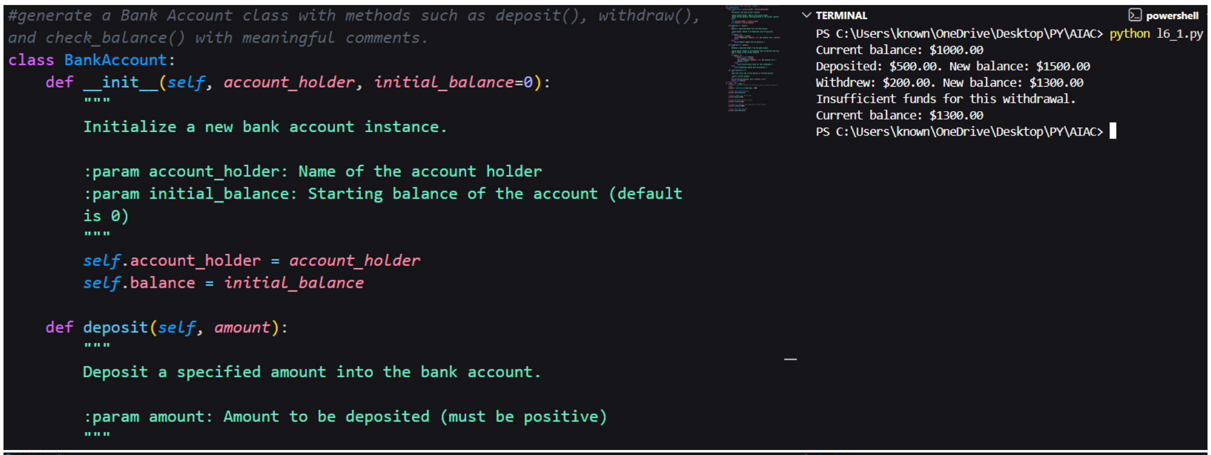
The while-loop version produces the same output as the for-loop version. While loops require manual control of the counter variable, making the for loop slightly cleaner and more readable for fixed iterations.

Task 5: Classes (Bank Account Class)

Prompt:

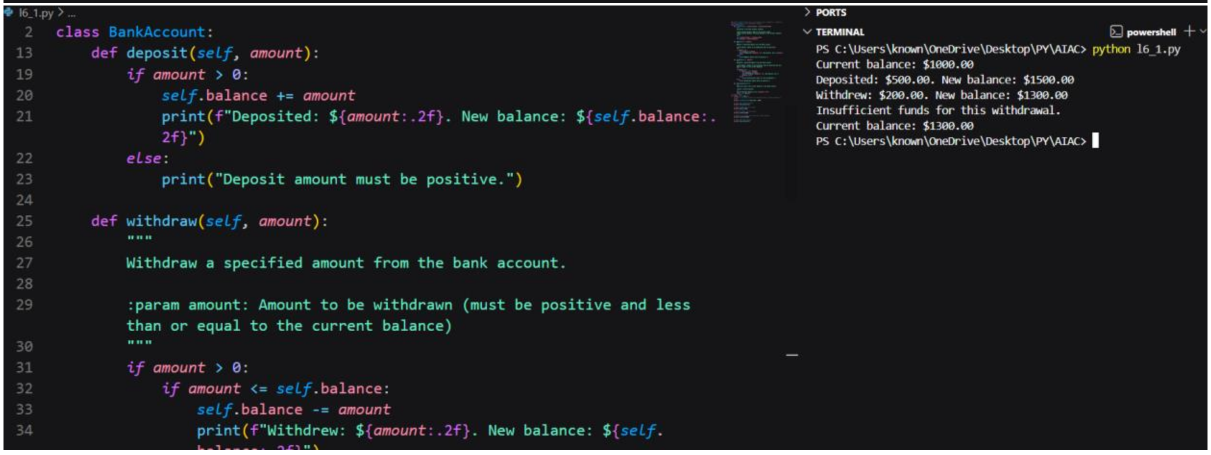
Generate a Bank Account class with methods such as deposit(), withdraw(),and check_balance() with meaningful comments.

Code & Output:



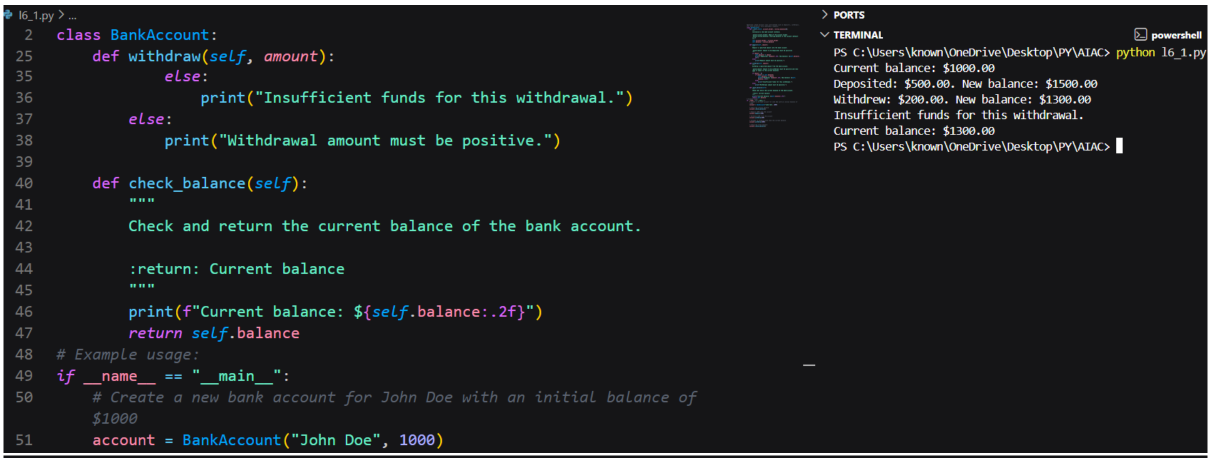
```
#generate a Bank Account class with methods such as deposit(), withdraw(),
and check_balance() with meaningful comments.
class BankAccount:
    def __init__(self, account_holder, initial_balance=0):
        """
        Initialize a new bank account instance.

        :param account_holder: Name of the account holder
        :param initial_balance: Starting balance of the account (default
        is 0)
        """
        self.account_holder = account_holder
        self.balance = initial_balance

    def deposit(self, amount):
        """
        Deposit a specified amount into the bank account.

        :param amount: Amount to be deposited (must be positive)
        """
```

```
TERMINAL
PS C:\Users\known\OneDrive\Desktop\PY\AIAC> python l6_1.py
Current balance: $1000.00
Deposited: $500.00. New balance: $1500.00
Withdrew: $200.00. New balance: $1300.00
Insufficient funds for this withdrawal.
Current balance: $1300.00
PS C:\Users\known\OneDrive\Desktop\PY\AIAC>
```



```
l6_1.py > ...
class BankAccount:
    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            print(f"Deposited: ${amount:.2f}. New balance: ${self.balance:.
            2f}")
        else:
            print("Deposit amount must be positive.")

    def withdraw(self, amount):
        """
        Withdraw a specified amount from the bank account.

        :param amount: Amount to be withdrawn (must be positive and less
        than or equal to the current balance)
        """
        if amount > 0:
            if amount <= self.balance:
                self.balance -= amount
                print(f"Withdrew: ${amount:.2f}. New balance: ${self.
                balance:.2f}")
```

```
PORTS
TERMINAL
PS C:\Users\known\OneDrive\Desktop\PY\AIAC> python l6_1.py
Current balance: $1000.00
Deposited: $500.00. New balance: $1500.00
Withdrew: $200.00. New balance: $1300.00
Insufficient funds for this withdrawal.
Current balance: $1300.00
PS C:\Users\known\OneDrive\Desktop\PY\AIAC>
```



l6_1.py > ...

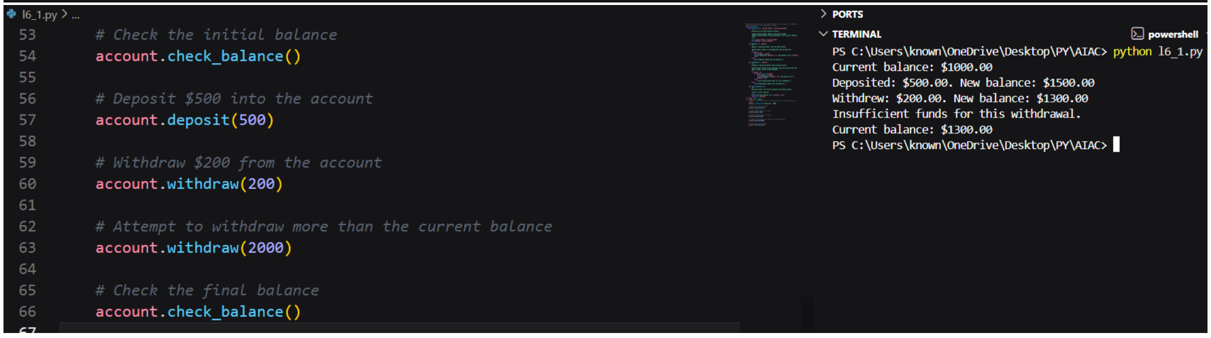
```python
class BankAccount:
    def withdraw(self, amount):
            else:
                print("Insufficient funds for this withdrawal.")
        else:
            print("Withdrawal amount must be positive.")

    def check_balance(self):
        """
        Check and return the current balance of the bank account.

        :return: Current balance
        """
        print(f"Current balance: ${self.balance:.2f}")
        return self.balance
# Example usage:
if __name__ == "__main__":
    # Create a new bank account for John Doe with an initial balance of
    $1000
    account = BankAccount("John Doe", 1000)
```

PORTS

TERMINAL

powershell

```
PS C:\Users\known\OneDrive\Desktop\PY\AIAC> python l6_1.py
Current balance: $1000.00
Deposited: $500.00. New balance: $1500.00
Withdrew: $200.00. New balance: $1300.00
Insufficient funds for this withdrawal.
Current balance: $1300.00
PS C:\Users\known\OneDrive\Desktop\PY\AIAC>
```



l6_1.py > ...

```python
    # Check the initial balance
    account.check_balance()

    # Deposit $500 into the account
    account.deposit(500)

    # Withdraw $200 from the account
    account.withdraw(200)

    # Attempt to withdraw more than the current balance
    account.withdraw(2000)

    # Check the final balance
    account.check_balance()
```

PORTS

TERMINAL

powershell

```
PS C:\Users\known\OneDrive\Desktop\PY\AIAC> python l6_1.py
Current balance: $1000.00
Deposited: $500.00. New balance: $1500.00
Withdrew: $200.00. New balance: $1300.00
Insufficient funds for this withdrawal.
Current balance: $1300.00
PS C:\Users\known\OneDrive\Desktop\PY\AIAC>
```

Explanation:

The AI-generated Bank Account class demonstrates effective use of object-oriented programming. The constructor initializes the balance, and the methods allow depositing, withdrawing, and checking the balance. Conditional logic prevents withdrawal when the balance is insufficient. The code is clear, logically sound, and easy to extend, making it suitable for a basic banking application.