

ASSIGNMENT- 6.3

Name: EDDE.POOJITHA

HT.No: 2303A51356

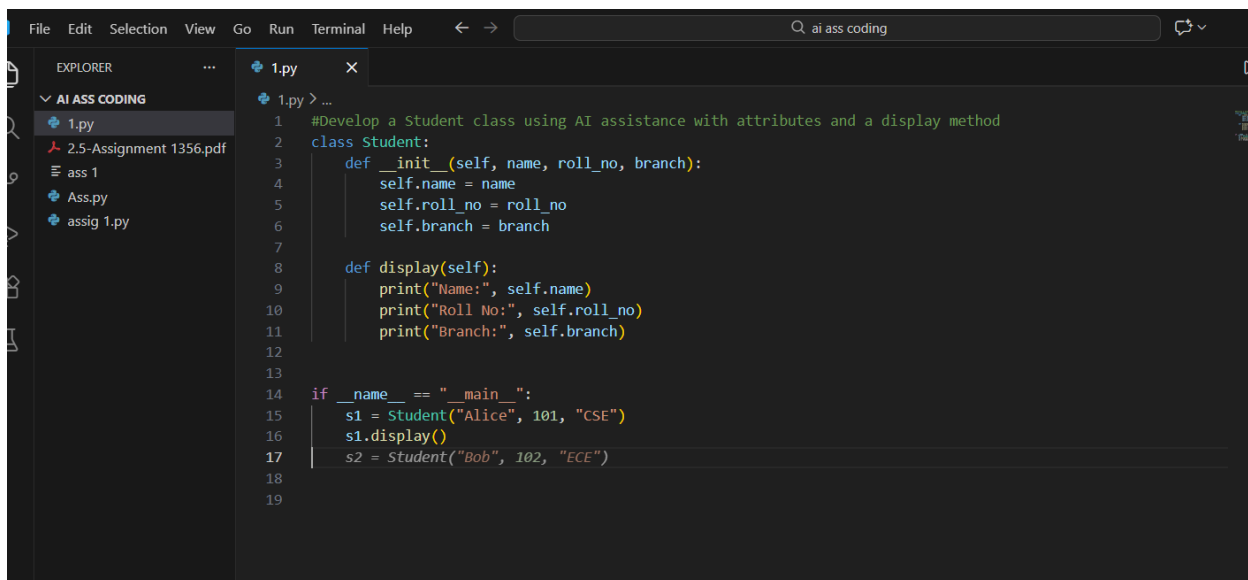
Batch: 20

Task 1: Classes – Student Class

Develop a Student class using AI assistance with attributes and a display method

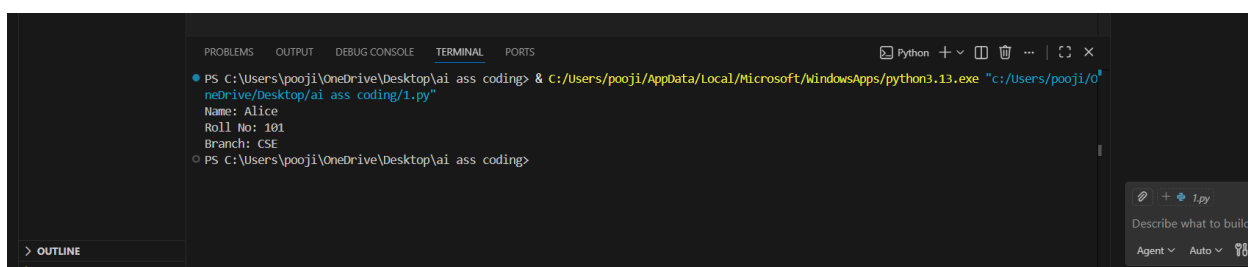
.Prompt: #Generate a Python Student class with name, roll number, and branch. Include a method to display student details..

Code:



```
1 #Develop a Student class using AI assistance with attributes and a display method
2 class Student:
3     def __init__(self, name, roll_no, branch):
4         self.name = name
5         self.roll_no = roll_no
6         self.branch = branch
7
8     def display(self):
9         print("Name:", self.name)
10        print("Roll No:", self.roll_no)
11        print("Branch:", self.branch)
12
13
14 if __name__ == "__main__":
15     s1 = Student("Alice", 101, "CSE")
16     s1.display()
17     s2 = Student("Bob", 102, "ECE")
18
19
```

Result:



```
PS C:\Users\pooji\OneDrive\Desktop\ai ass coding> & C:/Users/pooji/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/pooji/OneDrive/Desktop/ai ass coding/1.py"
Name: Alice
Roll No: 101
Branch: CSE
PS C:\Users\pooji\OneDrive\Desktop\ai ass coding>
```

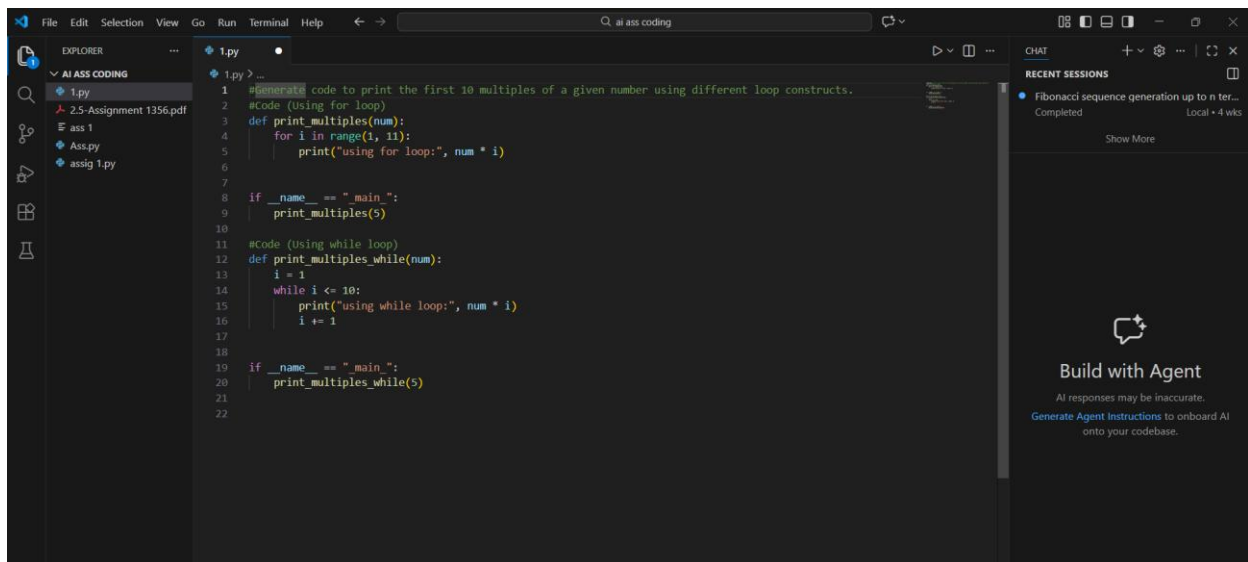
Observation:

The AI-generated class structure is clear and logically organized. The constructor correctly initializes attributes, and the display method outputs student details in a readable format. The code is simple, correct, and suitable for beginner-level object-oriented programming.

Task 2: Loops – Multiples of a Number. Generate code to print the first 10 multiples of a given number using different loop constructs.

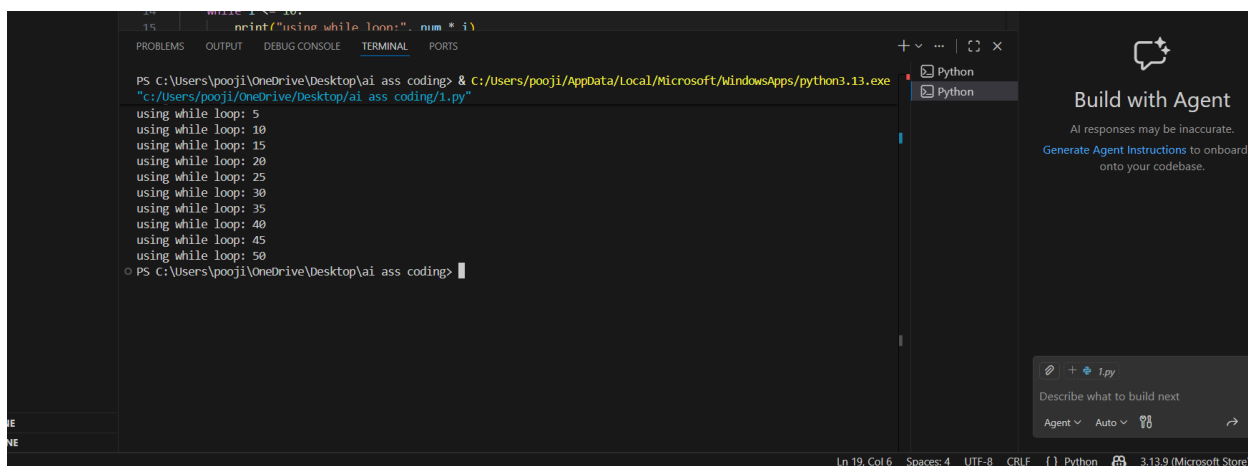
Prompt: #Generate Python code to print the first 10 multiples of a number using a loop.

Code:



```
1.py
1 #Generate code to print the first 10 multiples of a given number using different loop constructs.
2 #Code (Using for loop)
3 def print_multiples(num):
4     for i in range(1, 11):
5         print("using for loop:", num * i)
6
7
8 if __name__ == "__main__":
9     print_multiples(5)
10
11 #Code (Using while loop)
12 def print_multiples_while(num):
13     i = 1
14     while i <= 10:
15         print("using while loop:", num * i)
16         i += 1
17
18
19 if __name__ == "__main__":
20     print_multiples_while(5)
21
22
```

Result:



```
PS C:\Users\pooji\OneDrive\Desktop\ai ass coding> & C:\Users\pooji\AppData\Local\Microsoft\WindowsApps\python3.13.exe "c:/Users/pooji/OneDrive/Desktop/ai ass coding/1.py"
using while loop: 5
using while loop: 10
using while loop: 15
using while loop: 20
using while loop: 25
using while loop: 30
using while loop: 35
using while loop: 40
using while loop: 45
using while loop: 50
PS C:\Users\pooji\OneDrive\Desktop\ai ass coding>
```

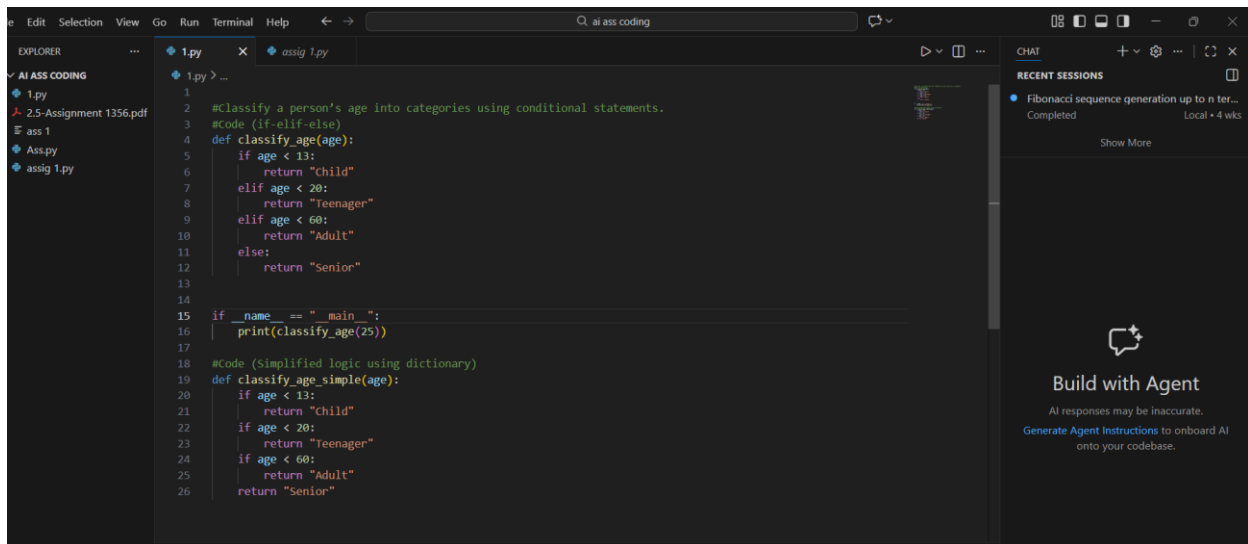
Observation:

Both loop implementations correctly generate the required output. The for-loop version is more concise and readable, while the while-loop version provides better insight into loop control and iteration. AI suggestions for both approaches are correct and efficient.

Task 3: Conditional Statements – Age Classification. Classify a person's age into categories using conditional statements.

Prompt: # Generate Python code to classify age into child, teenager, adult, and senior using if-elif-else..

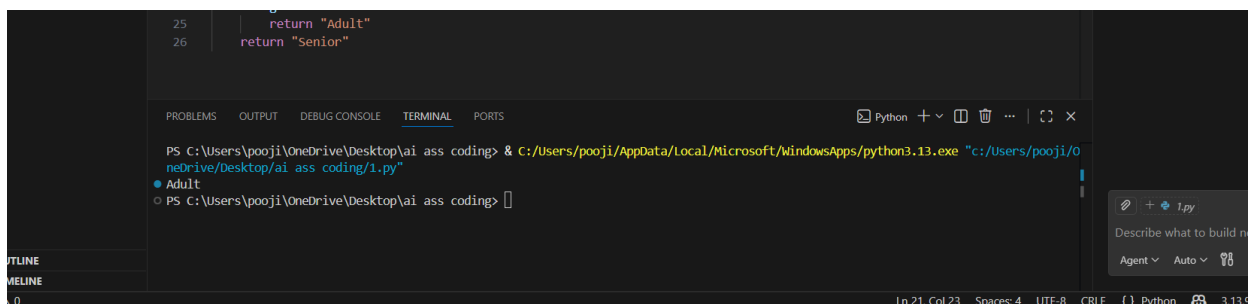
Code:



The screenshot shows a VS Code editor with a Python file named `1.py`. The code implements an age classification function using `if-elif-else` statements. The function `classify_age` takes an age as input and returns a category: "Child" for ages 0-12, "Teenager" for 13-19, "Adult" for 20-59, and "Senior" for 60 and above. A `__main__` block calls `classify_age(25)` and prints the result. A commented-out section shows a simplified logic using a dictionary.

```
1
2 #Classify a person's age into categories using conditional statements.
3 #code (if-elif-else)
4 def classify_age(age):
5     if age < 13:
6         return "Child"
7     elif age < 20:
8         return "Teenager"
9     elif age < 60:
10        return "Adult"
11    else:
12        return "Senior"
13
14
15 if __name__ == "__main__":
16     print(classify_age(25))
17
18 #Code (Simplified logic using dictionary)
19 def classify_age_simple(age):
20     if age < 13:
21         return "Child"
22     if age < 20:
23         return "Teenager"
24     if age < 60:
25         return "Adult"
26     return "Senior"
```

Result:



The screenshot shows the terminal output of the Python code. The command executed is `python3.13.exe "c:/Users/pooji/OneDrive/Desktop/ai ass coding/1.py"`. The output is `Adult`, indicating that the age 25 was correctly classified as an adult.

```
PS C:\Users\pooji\OneDrive\Desktop\ai ass coding> C:\Users\pooji\AppData\Local\Microsoft\WindowsApps\python3.13.exe "c:/Users/pooji/OneDrive/Desktop/ai ass coding/1.py"
● Adult
○ PS C:\Users\pooji\OneDrive\Desktop\ai ass coding>
```

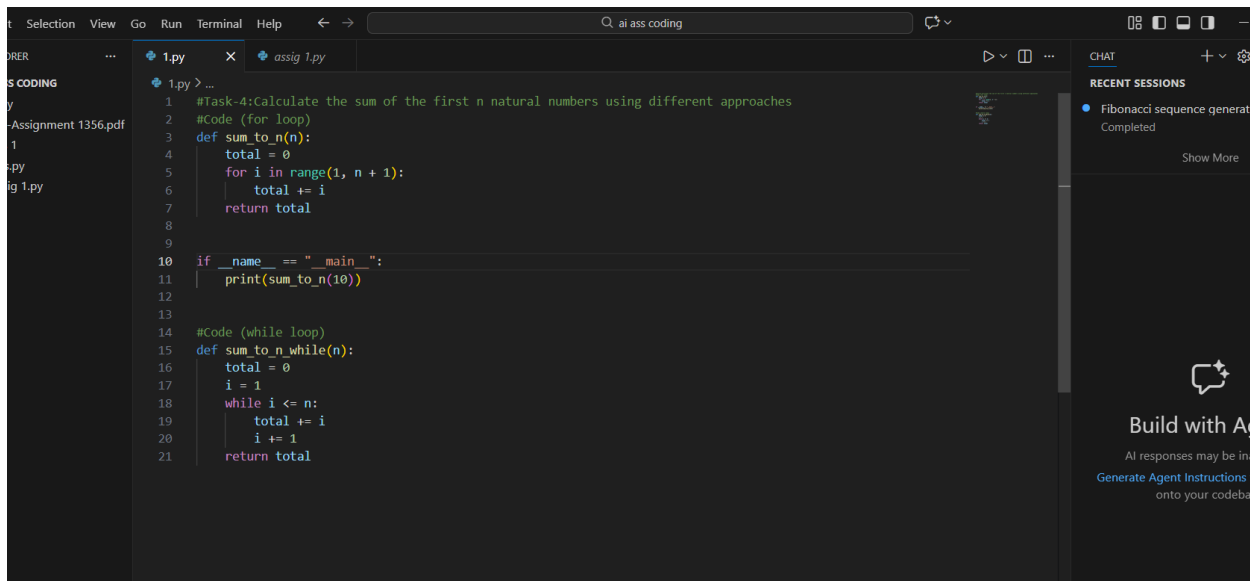
Observation:

The AI-generated conditions correctly classify age groups. The if-elif-else structure is clear and readable, while the simplified version reduces nesting and improves clarity. Both approaches are logically sound.

Task 4: For and While Loops – Sum of First n Numbers. Calculate the sum of the first n natural numbers using different approaches.

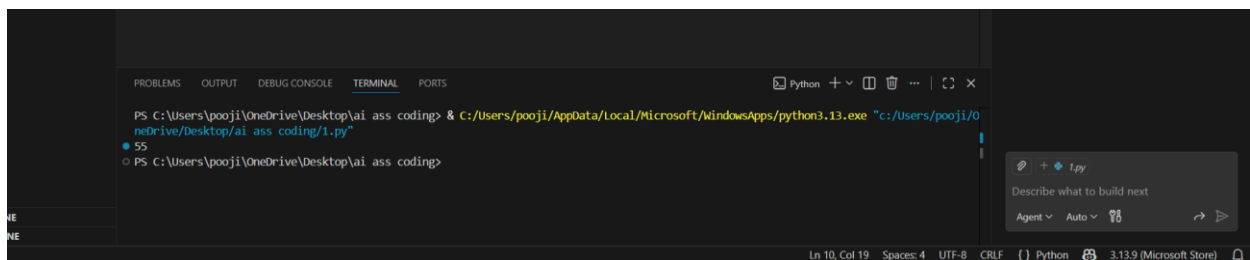
Prompt: #Generate Python code to find the sum of the first n natural numbers using loops.

Code:



```
1 #Task-4:Calculate the sum of the first n natural numbers using different approaches
2 #Code (for loop)
3 def sum_to_n(n):
4     total = 0
5     for i in range(1, n + 1):
6         total += i
7     return total
8
9
10 if __name__ == "__main__":
11     print(sum_to_n(10))
12
13
14 #Code (while loop)
15 def sum_to_n_while(n):
16     total = 0
17     i = 1
18     while i <= n:
19         total += i
20         i += 1
21     return total
```

Result:



```
PS C:\Users\pooji\OneDrive\Desktop\ai ass coding> C:\Users\pooji\AppData\Local\Microsoft\WindowsApps\python3.13.exe "c:\Users\pooji\OneDrive\Desktop\ai ass coding\1.py"
55
PS C:\Users\pooji\OneDrive\Desktop\ai ass coding>
```

Observation

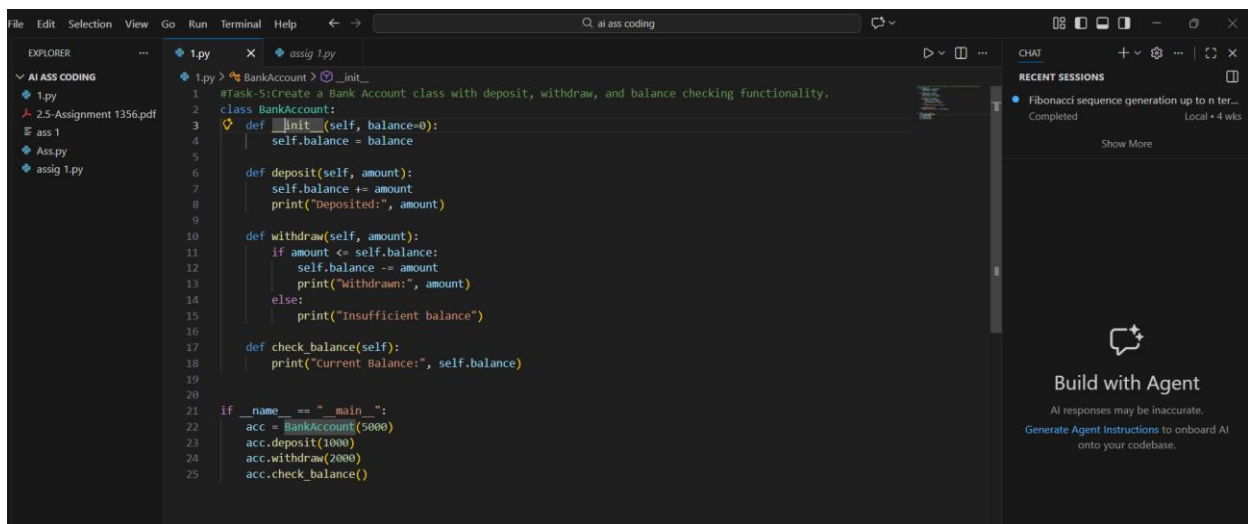
Both loop-based solutions produce the correct result. The for-loop version is more concise, while the while-loop version offers explicit control over iteration. AI-generated logic is correct and easy to understand

Task 5: Classes – Bank Account Class

Create a Bank Account class with deposit, withdraw, and balance checking functionality.

Prompt: #Generate a Python Bank Account class with deposit, withdraw, and check balance methods.

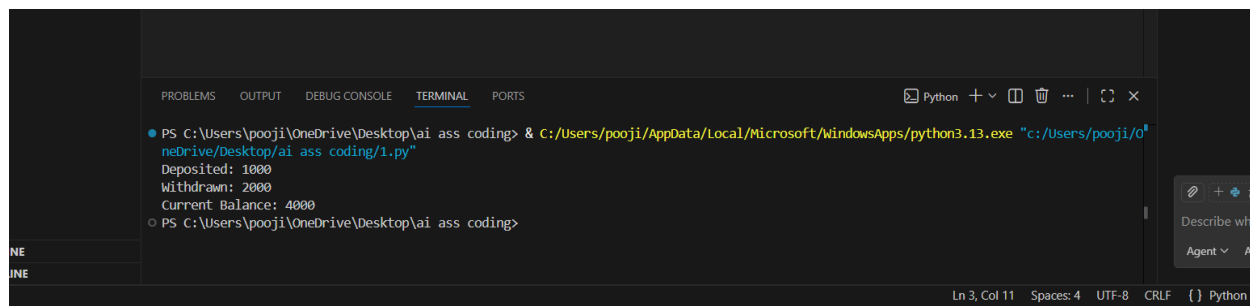
Code:



The screenshot shows a VS Code editor with a Python file named `1.py`. The code defines a `BankAccount` class with three methods: `__init__`, `deposit`, and `withdraw`, and a `check_balance` method. The `__init__` method initializes the `balance` attribute to 0. The `deposit` method adds the specified amount to the balance and prints the updated balance. The `withdraw` method checks if the specified amount is less than or equal to the current balance; if so, it subtracts the amount and prints the updated balance, otherwise, it prints "Insufficient balance". The `check_balance` method prints the current balance. A main block at the bottom creates an instance of the `BankAccount` class and demonstrates the `deposit`, `withdraw`, and `check_balance` methods.

```
1.py > % BankAccount > _init_
1 #Task-5: Create a Bank Account class with deposit, withdraw, and balance checking functionality.
2 class BankAccount:
3     def __init__(self, balance=0):
4         self.balance = balance
5
6     def deposit(self, amount):
7         self.balance += amount
8         print("Deposited:", amount)
9
10    def withdraw(self, amount):
11        if amount <= self.balance:
12            self.balance -= amount
13            print("Withdrawn:", amount)
14        else:
15            print("Insufficient balance")
16
17    def check_balance(self):
18        print("Current Balance:", self.balance)
19
20
21 if __name__ == "__main__":
22     acc = BankAccount(5000)
23     acc.deposit(1000)
24     acc.withdraw(2000)
25     acc.check_balance()
```

Result:



The screenshot shows a Visual Studio Code interface with a terminal window open. The terminal has tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The TERMINAL tab is active, showing a PowerShell prompt at the directory C:\Users\pooji\OneDrive\Desktop\ai ass coding. The user has run the command: `C:/Users/pooji/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/pooji/OneDrive/Desktop/ai ass coding/1.py"`. The output of the script is displayed in green text: `Deposited: 1000`, `Withdrawn: 2000`, and `Current Balance: 4000`. The prompt then returns to `PS C:\Users\pooji\OneDrive\Desktop\ai ass coding>`. On the right side of the terminal, there is a sidebar with a 'Describe what you're seeing' button and an 'Agent' dropdown menu. At the bottom right of the terminal window, the status bar indicates 'Ln 3, Col 11', 'Spaces: 4', 'UTF-8', 'CRLF', and the file is a 'Python' script.

```
PS C:\Users\pooji\OneDrive\Desktop\ai ass coding> & C:/Users/pooji/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/pooji/OneDrive/Desktop/ai ass coding/1.py"
Deposited: 1000
Withdrawn: 2000
Current Balance: 4000
PS C:\Users\pooji\OneDrive\Desktop\ai ass coding>
```

Observation:

The AI-generated class structure is well organized and logically correct. Methods perform expected operations, and balance updates are accurate. The code is readable, maintainable, and suitable for a basic banking application.