AI Assisted Coding

# Assignment-6.1

## K.Sindhu meenan(2403A51250)-Batch11

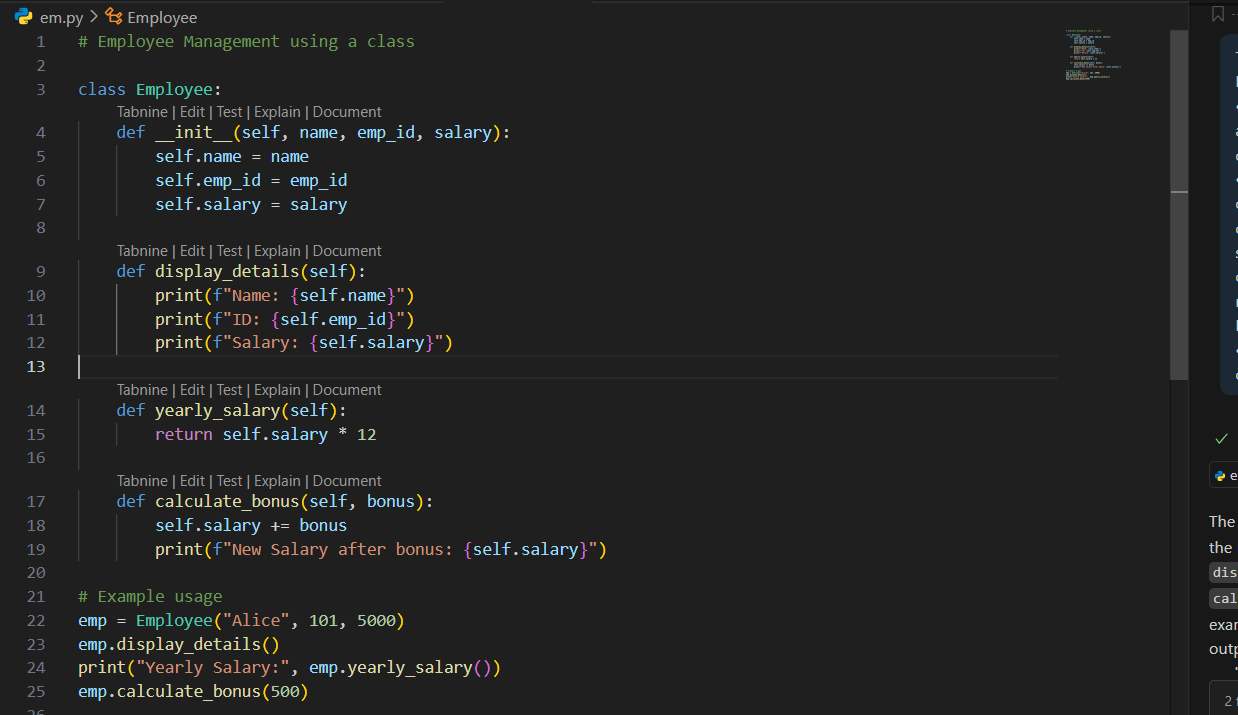
Task1: Use AI to create an Employee class with attributes (name, id, salary) and a method to calculate yearly salary.

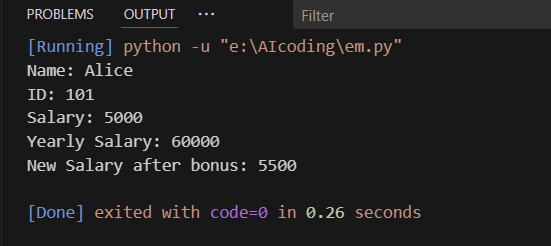
Expected Output #1:

* A class with constructor, display\_details(), and calculate\_bonus() methods.

Prompt: Create an Employee class with attributes (name, id, salary) and a method to calculate yearly salary. class should contain constructor, display\_details(), and calculate\_bonus() methods.

Code:



Output: 

Task2: Prompt AI to generate a function that displays all Automorphic numbers between 1 and 1000 using a for loop. And ask AI to regenerate using a while loop and compare both implementations.

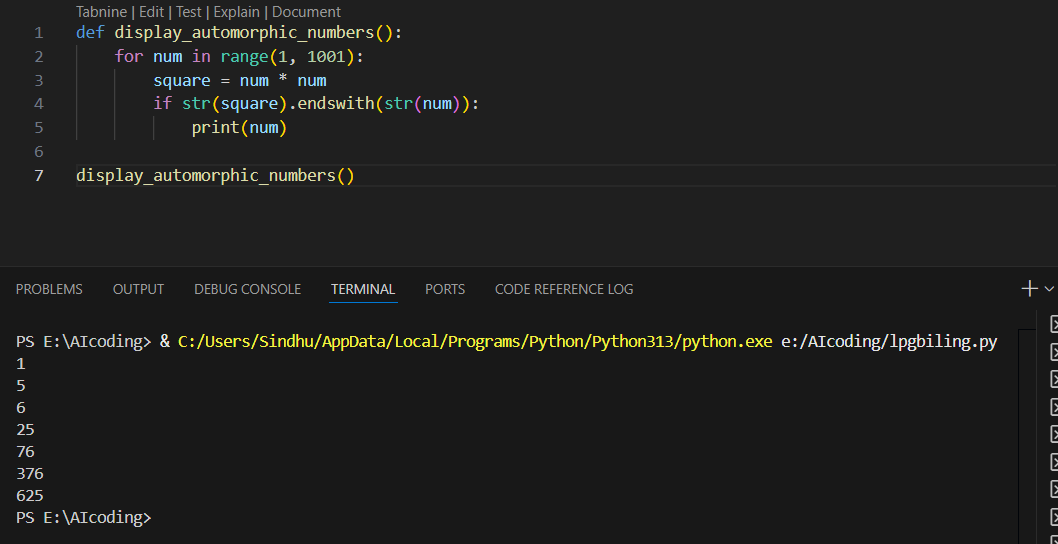
Expected Output #2:

* Correct implementation that lists Automorphic numbers using both loop types, with explanation.

Prompt:

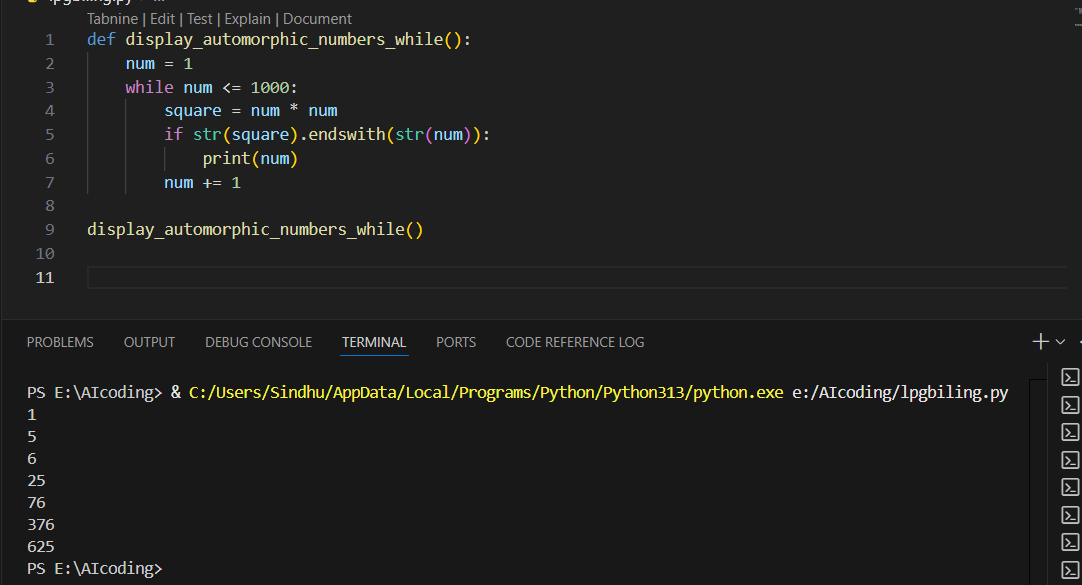
For loop: Generate a Python function to display all Automorphic numbers between 1 and 1000 using a for loop.

Code and output:



While loop: Regenerate using while loop.

Code and output:



For loop:

* The for loop automatically takes care of **initialization**, **condition**, and **increment**.
* The code is **compact** and very **pythonic**.
* It is easy to read and understand because the range is fixed (1 to 1000).

While loop:

* The while loop requires **manual initialization** (num = 1), an **explicit condition** (num <= 1000), and **manual increment** (num += 1).
* Slightly more **verbose** compared to the for loop.
* Offers **more flexibility** if the exit condition changes (e.g., stopping when a certain property is met instead of a fixed range).

Conclusion:

* The **for loop implementation** is the better choice when dealing with a fixed range of values because it is **shorter, cleaner, and easier to read**.
* The **while loop implementation** is more useful when the **exit condition is not predefined** (e.g., based on user input or dynamic conditions).

In summary, the **for loop is preferred in this problem** due to its readability and simplicity, while the **while loop remains a valuable alternative** in scenarios where more control over the loop is required.

Task3: Ask AI to write nested if-elif-else conditions to classify online shopping feedback as Positive, Neutral, or Negative based on a numerical rating (1–5). And to rewrite using dictionary-based or match-case structure.

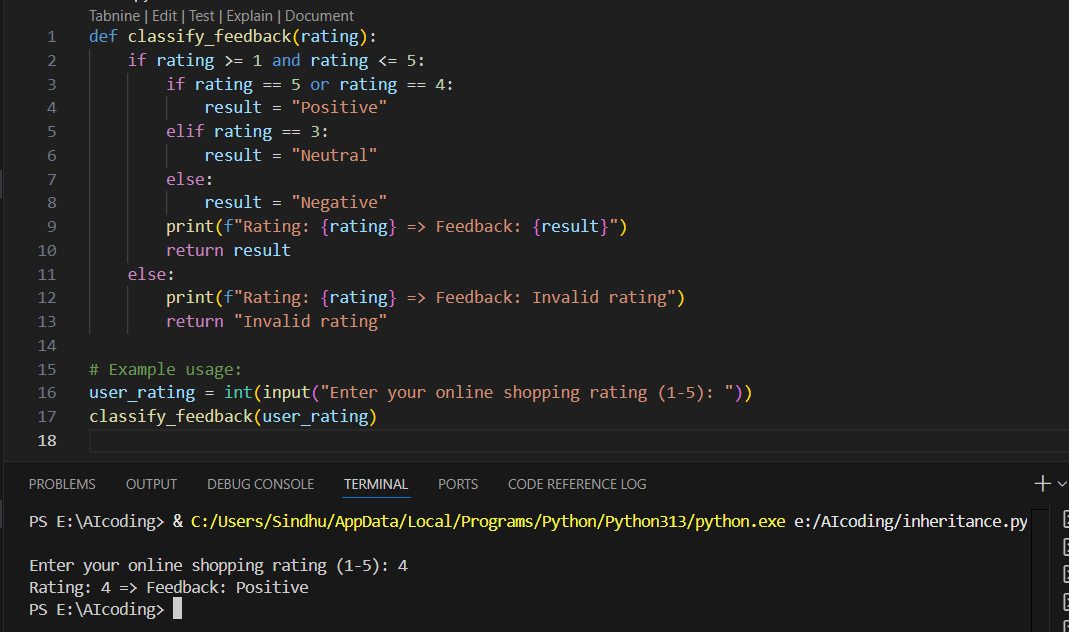
Expected Output #3:

* Feedback classification function with explanation and an alternative approach.

Prompt:

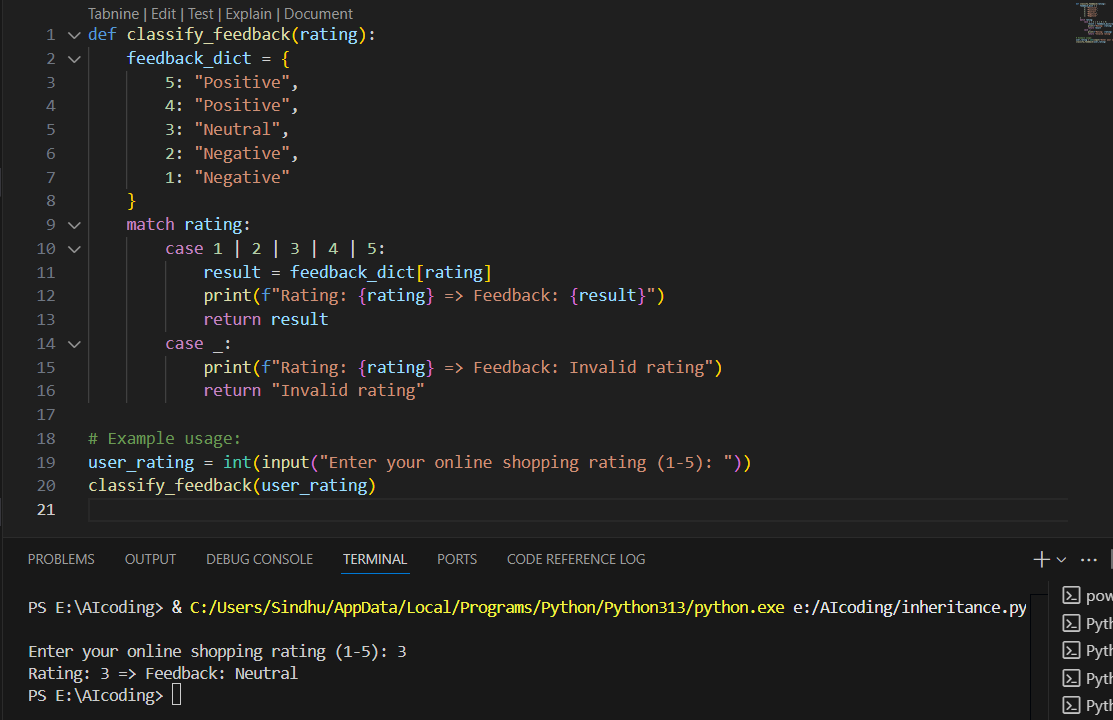
Nested if-elif-else: Generate nested if-elif-else conditions to classify online shopping feedback as Positive, Neutral, or Negative based on a numerical rating (1–5).

Code and output:



dictionary-based or match-case structure: Regenerate using dictionary-based or match-case structure.

Code and output:



Nested if-elif-else code Explanation:

* The function first checks if the rating is within the valid range (**1–5**).
* Inside that condition, nested if-elif-else statements classify the rating:
  + **4 or 5 → Positive**
  + **3 → Neutral**
  + **1 or 2 → Negative**
* If the input is outside the range, it returns **Invalid rating**.

**Correctness**: The logic is accurate and works as expected.  
**Readability**: For small cases, it is fine, but if more categories were added, nested conditions would become long and harder to maintain.

Dictionary-based or match-case structure code explanation:

* A **dictionary** directly maps rating values to feedback categories.
* The **match-case** statement (Python 3.10+) checks if the rating is valid.
  + If valid, the dictionary is used to fetch the feedback.
  + If invalid, the case \_ block handles it.

**Correctness**: Matches the same logic as the first version.  
**Readability**: Cleaner and easier to extend for more categories.

Conclusion:

Both approaches correctly classify online shopping ratings into **Positive, Neutral, Negative, or Invalid**.

* The **nested if-elif-else** solution is **simple and beginner-friendly**.
* The **dictionary + match-case** solution is **more modern and elegant**.

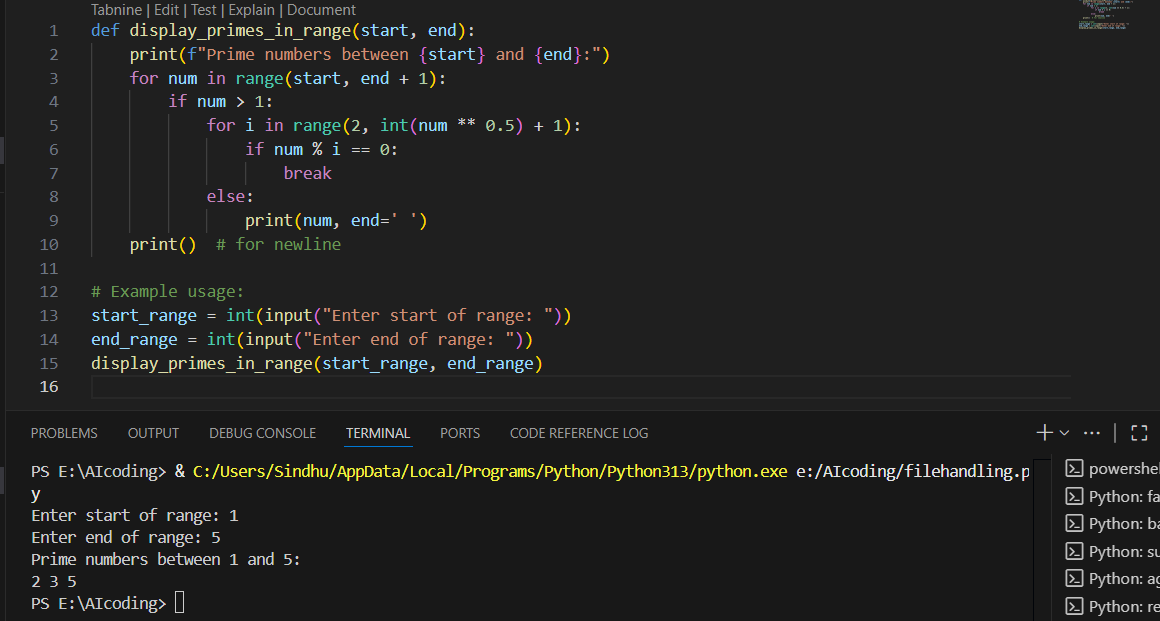
Task4: Generate a function using AI that displays all prime numbers within a user-specified range (e.g., 1 to 500) using a for loop. And Ask AI to regenerate an optimized version (e.g., using the square root method).

Expected Output #4:

* Python program that lists all prime numbers within a given range, with an optimized version and explanation.

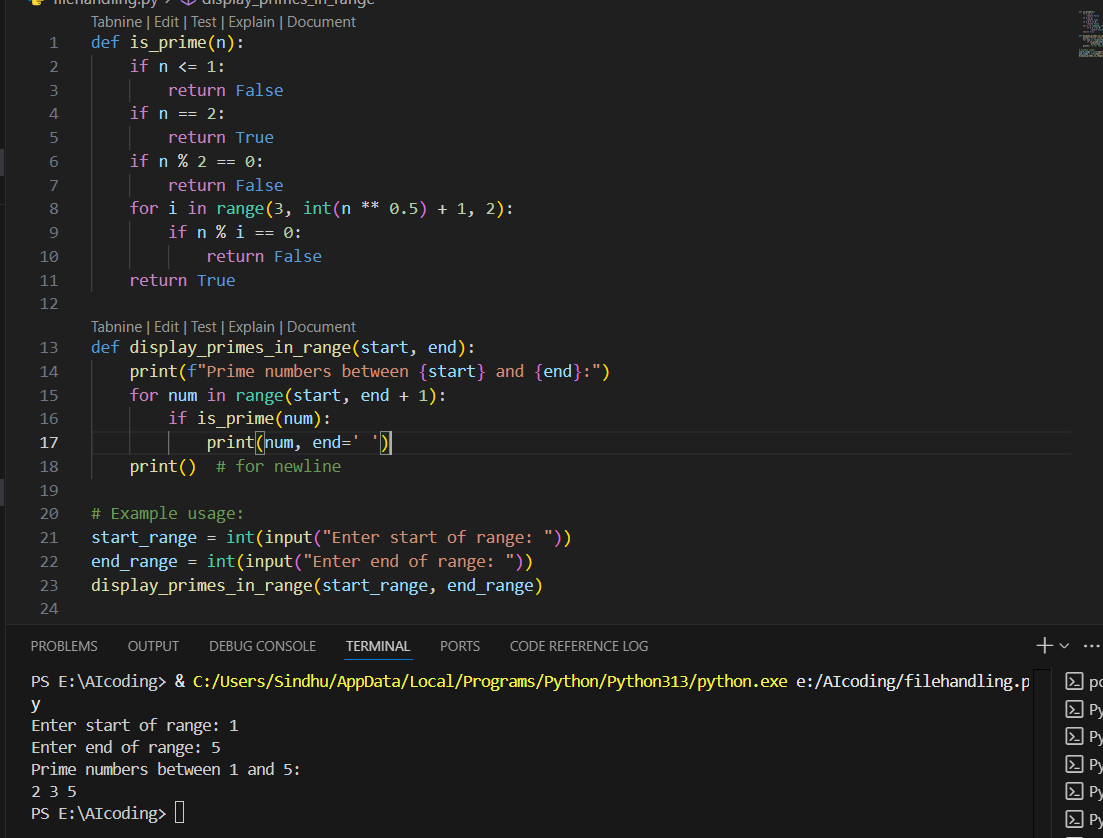
For loop: Generate a function that displays all prime numbers within a user-specified range (e.g., 1 to 500) using a for loop.

Code and output:



Optimized version: regenerate an optimized version (e.g., using the square root method).

Code and output:



Analysis of for loop code:

* Correctly lists prime numbers in the given range.
* Uses a for loop for iteration.
* Prime-checking logic is inside the main loop, making it less reusable.

Analysis of optimized code:

* Prime-checking logic is separated into is\_prime(), making the program modular and reusable.
* Efficiency improved by checking divisibility only up to the square root of the number.
* Skips even numbers beyond 2 for faster execution.
* Much better suited for larger ranges (e.g., 1 to 500).

Conclusion:

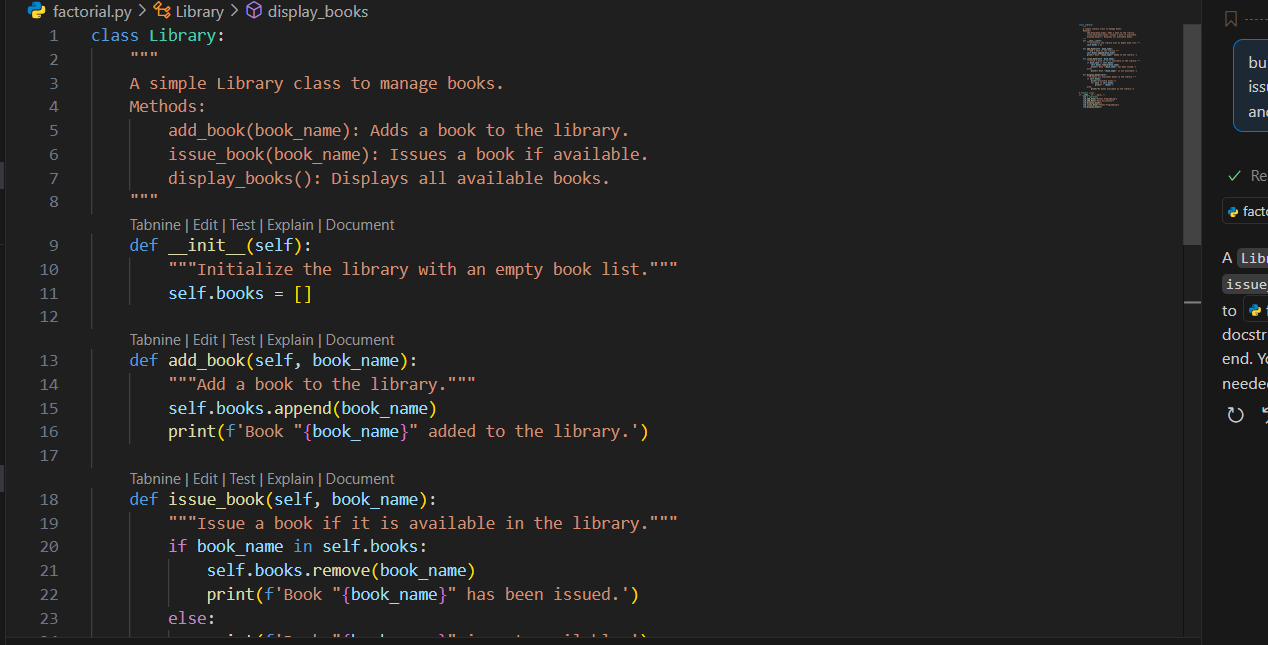
* The **initial version** works correctly but mixes prime-checking logic inside the main loop.
* The **optimized version** is modular, efficient, and scalable for larger ranges.
* For small ranges, both approaches are fine, but for real applications, the **optimized version is preferred**.

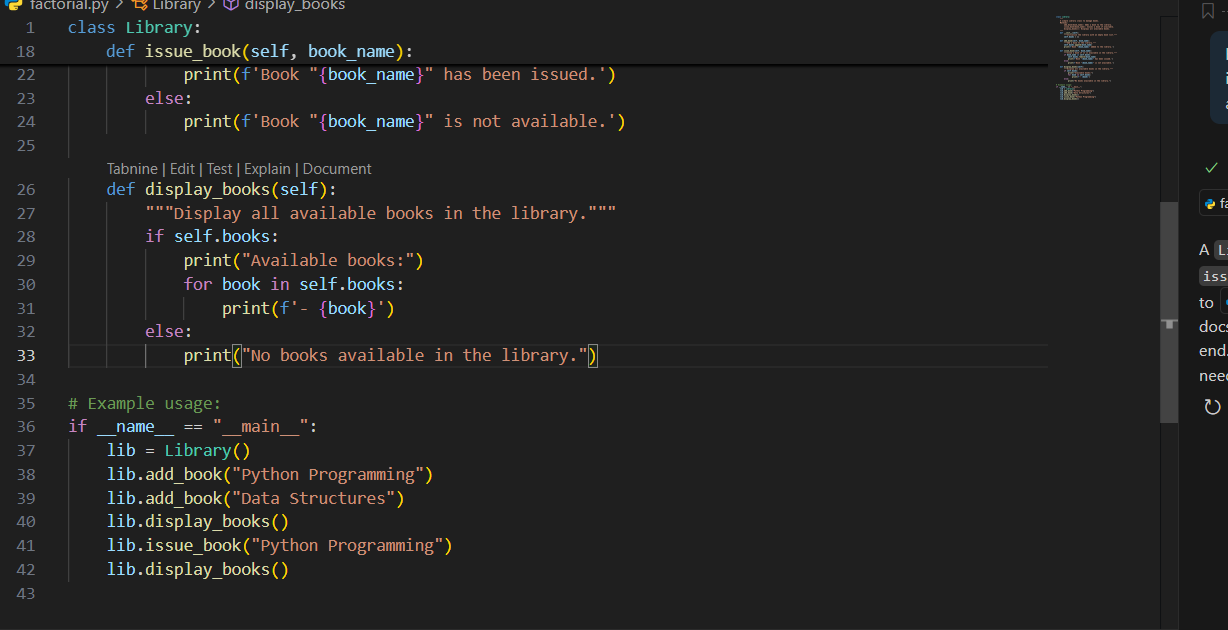
Task5: Use AI to build a Library class with methods to add\_book(), issue\_book(), and display\_books(). Analyze if methods handle edge cases (e.g., issuing unavailable books). And Ask AI to add comments and documentation.

Expected Output #5:

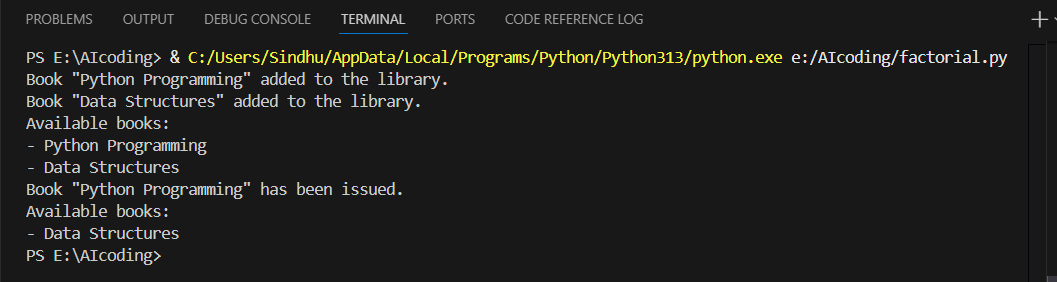
* Library class with all methods, inline comments, and explanation.

Code:





Output:



Explanation:

This program successfully demonstrates:

* **Class and Object usage** in Python.
* **Encapsulation** of book management logic within methods.
* A simple real-world application of OOP concepts for **library management**.