

AI ASSISTED CODING

TASK 2:

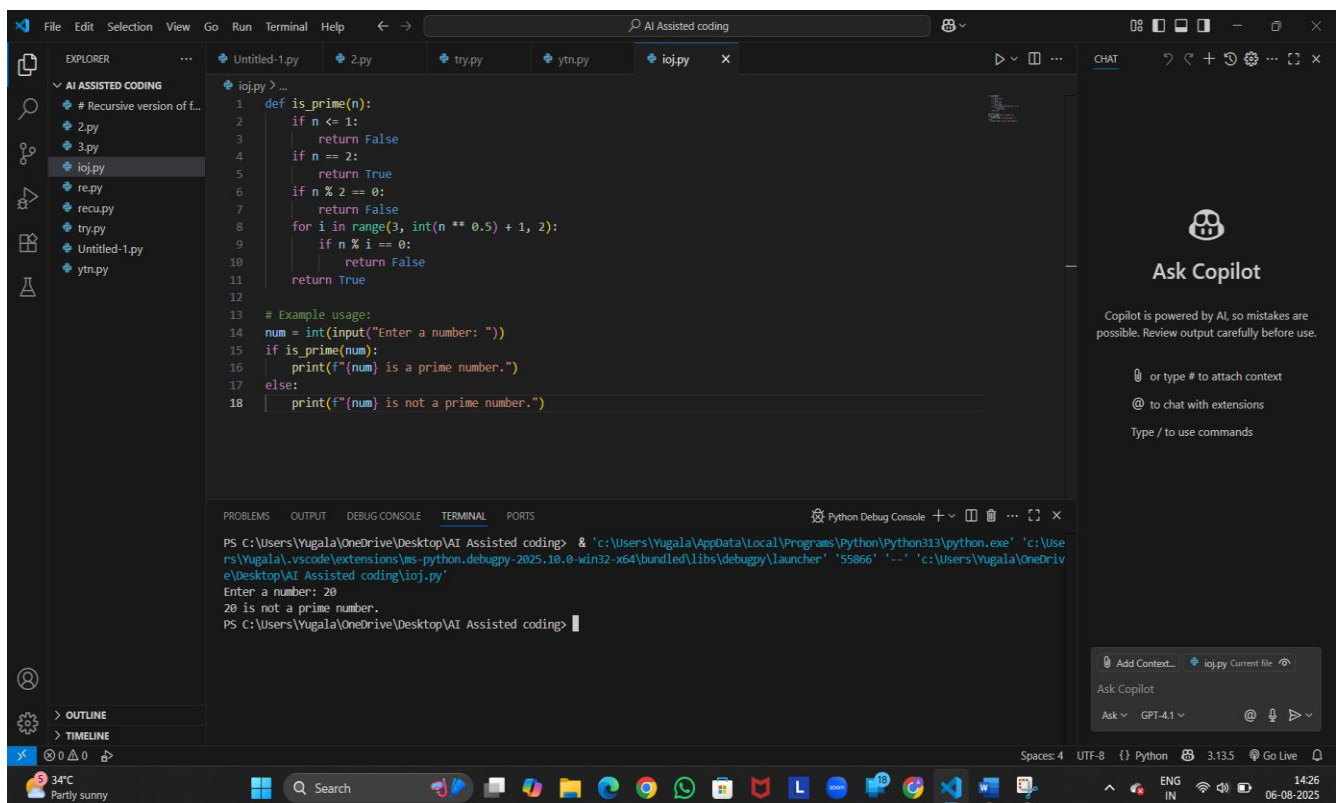
>> Use copilot to generate a is_prime() python function.

prompt:

Write a python code to check whether a number is prime or not.

Expected output:

Function to check primality with correct logic.



The screenshot shows the Visual Studio Code interface with the AI Assistant sidebar on the right. The main editor displays a Python file named 'ioj.py' containing a function 'is_prime(n)' and an example usage. The function checks for primality using a range from 3 to the square root of n. The example usage prompts the user to enter a number and prints the result. The terminal at the bottom shows the command to run the script and the output for the input '20'.

```
1 def is_prime(n):
2     if n <= 1:
3         return False
4     if n == 2:
5         return True
6     if n % 2 == 0:
7         return False
8     for i in range(3, int(n ** 0.5) + 1, 2):
9         if n % i == 0:
10            return False
11    return True
12
13 # Example usage:
14 num = int(input("Enter a number: "))
15 if is_prime(num):
16     print(f"{num} is a prime number.")
17 else:
18     print(f"{num} is not a prime number.")
```

Terminal output:

```
PS C:\Users\Yugala\OneDrive\Desktop\AI Assisted coding> & 'c:\Users\Yugala\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\Yugala\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundle\libs\debugpy\launcher' '55866' '-.' 'c:\Users\Yugala\OneDrive\Desktop\AI Assisted coding\ioj.py'
Enter a number: 20
20 is not a prime number.
PS C:\Users\Yugala\OneDrive\Desktop\AI Assisted coding>
```

Observation:

A prime number is a number greater than 1 that has no positive divisors other than 1 and itself.

- **Functionality:** Returns True for prime numbers and False otherwise.
- **Suitable for checking primality of large numbers due to reduced iterations.**

TASK 3:

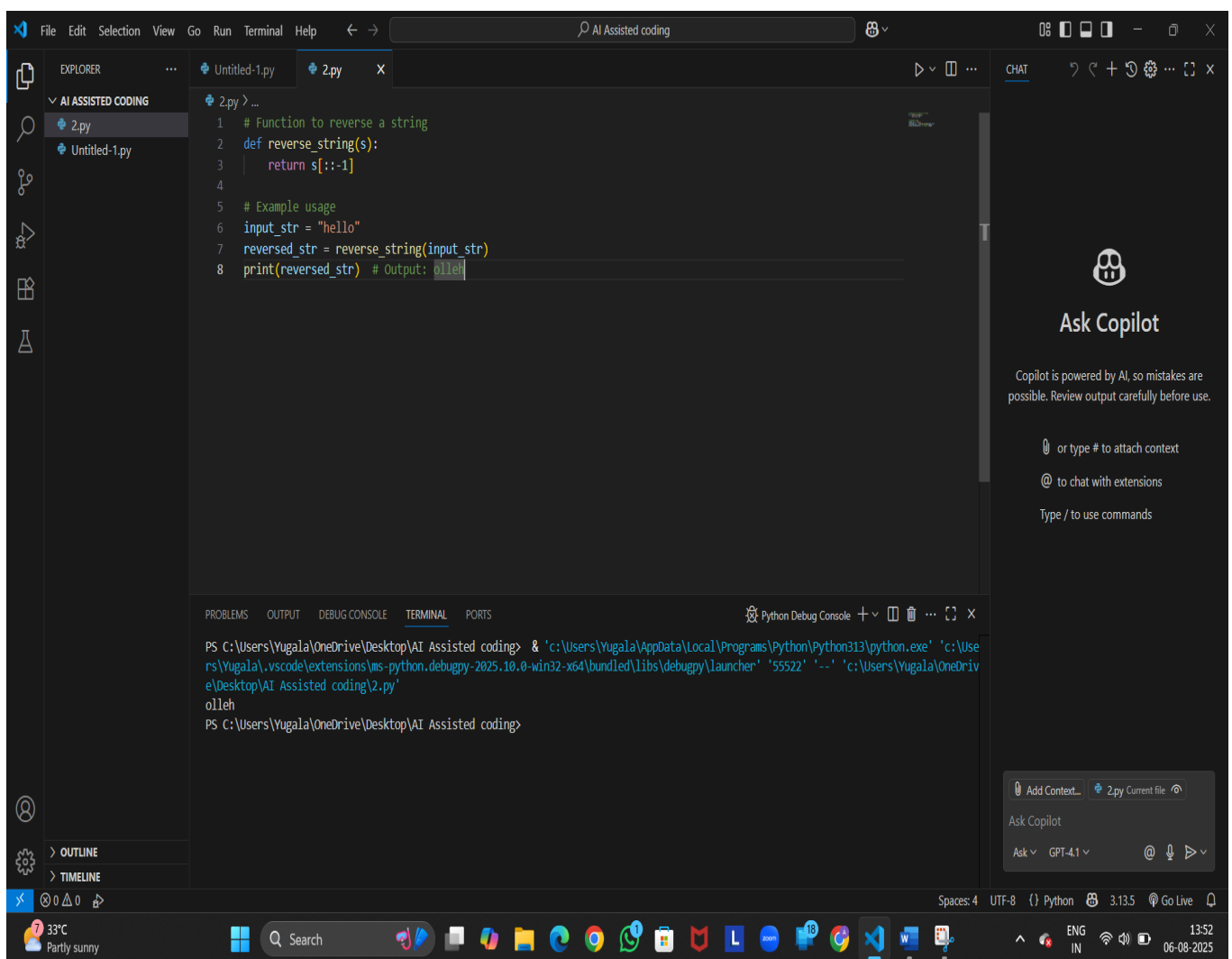
>>Write a comment like # function to reverse a string and use copilot to generate the function.

Prompt:

>>Write a python code for comment like # function to reverse a string and use function.

Expected output:

>>Auto-completed reverse function.



Observation:

- **Logic:** Uses Python slicing `[::-1]` to reverse the string efficiently.
- **Simplicity:** One-liner function; concise and readable.
- **Functionality:** Works for letters, numbers, symbols, and even empty strings.

TASK 4:

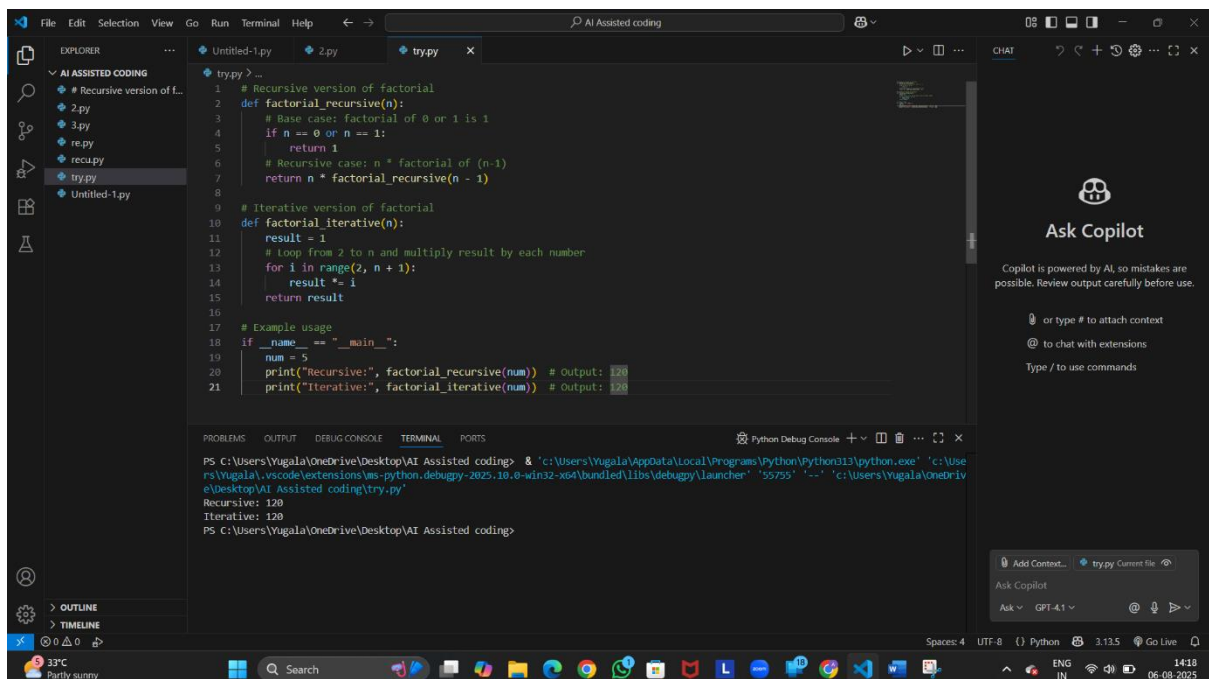
>>Generate both recursive and iterative version of a factorial using comments.

Prompt:

>>Write a python to generate both recursive and iterative version of a factorial using comments.

Expected output:

>>Two working factorial implementations.



The screenshot shows a Visual Studio Code editor with a file named `try.py` open. The code contains two functions: `factorial_recursive(n)` and `factorial_iterative(n)`. The recursive function uses a base case of `n == 0 or n == 1` and a recursive call `return n * factorial_recursive(n - 1)`. The iterative function uses a loop from 2 to `n` to calculate the factorial. Below the code, the terminal shows the output of running the script, which prints the recursive and iterative results for `num = 5`, both yielding `120`.

```
1 # Recursive version of factorial
2 def factorial_recursive(n):
3     # Base case: factorial of 0 or 1 is 1
4     if n == 0 or n == 1:
5         return 1
6     # Recursive case: n * factorial of (n-1)
7     return n * factorial_recursive(n - 1)
8
9 # Iterative version of factorial
10 def factorial_iterative(n):
11     result = 1
12     # loop from 2 to n and multiply result by each number
13     for i in range(2, n + 1):
14         result *= i
15     return result
16
17 # Example usage
18 if __name__ == "__main__":
19     num = 5
20     print("Recursive:", factorial_recursive(num)) # Output: 120
21     print("Iterative:", factorial_iterative(num)) # Output: 120
```

```
PS C:\Users\Vugala\OneDrive\Desktop\AI Assisted coding> & 'c:\Users\Vugala\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\Vugala\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundle\libs\debugpy\launcher' '55755' '-.' 'c:\Users\Vugala\OneDrive\Desktop\AI Assisted coding\try.py'
Recursive: 120
Iterative: 120
PS C:\Users\Vugala\OneDrive\Desktop\AI Assisted coding>
```

Observation:

- **Recursive Version:**
 - **Elegant and mirrors the mathematical definition.**
 - **May cause stack overflow for large n due to deep recursion.**
 - **Time complexity: $O(n)$; Space complexity: $O(n)$ (due to call stack).**
- **Iterative Version:**
 - **More memory-efficient and avoids recursion limits.**
 - **Preferred for large values of n.**
 - **Time complexity: $O(n)$; Space complexity: $O(1)$.**
- **Both Implementations:**
 - **Correctly handle base cases ($0! = 1$, $1! = 1$).**
 - **Produce identical results for valid non-negative integers.**

TASK 5:

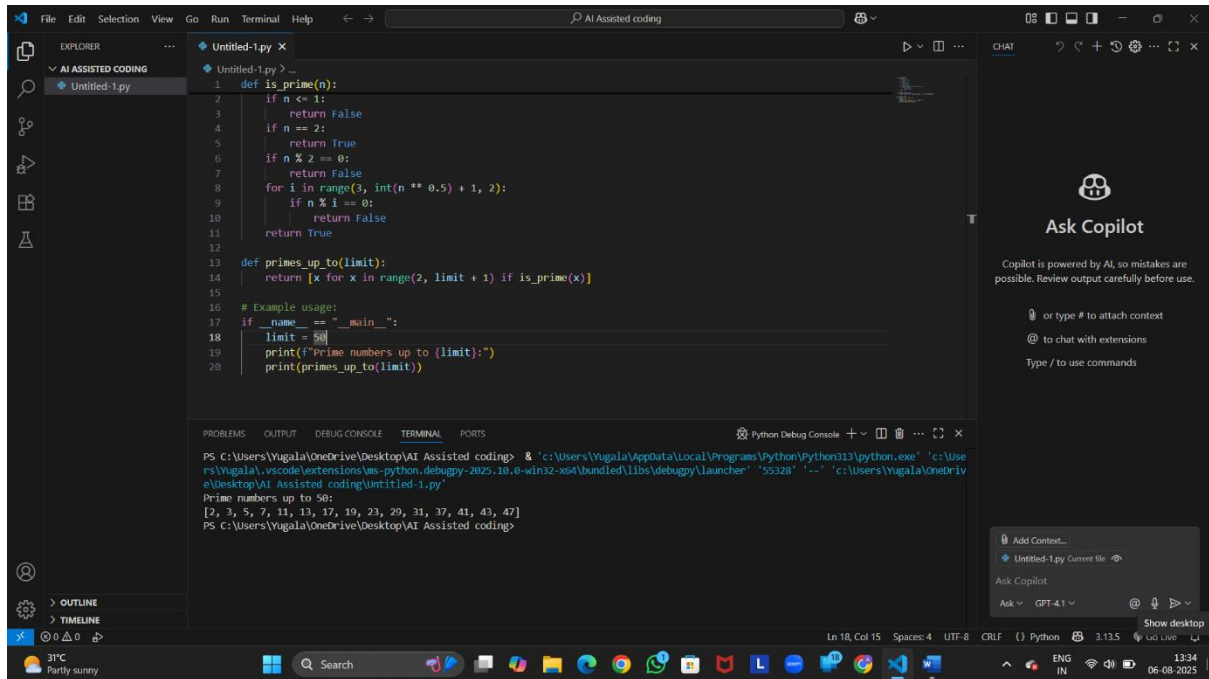
>> Use copilot to find the largest number in a list. Assess code quality and efficiency.

Prompt:

>> Write a python code to find the largest number in a list and assess code quality and efficiency.

Expected output:

>> A valid function with your review.



Observation:

- **Correctness:** Accurately finds the largest number by comparing each element.
- **Edge Case Handling:** Returns None for an empty list, avoiding errors.
- **Efficiency:**
 - **Time complexity:** $O(n)$ — linear scan through the list.
 - **Space complexity:** $O(1)$ — uses constant extra space.
- **Code Quality:**
 - **Clear variable naming** (largest, numbers).