

# Lab - 01 Assignment

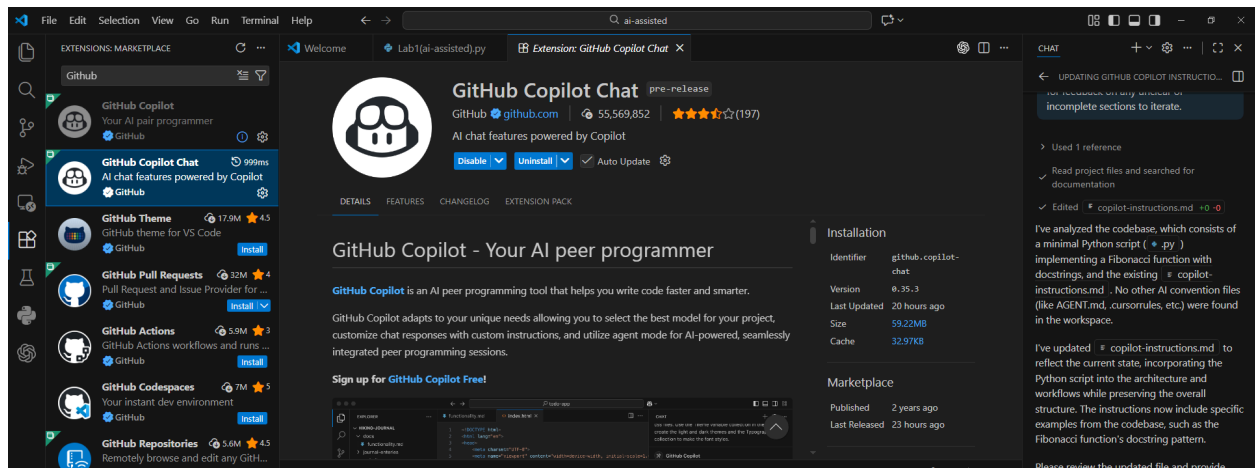
Name: Zobiya Fatima

Roll No: 2303A51879

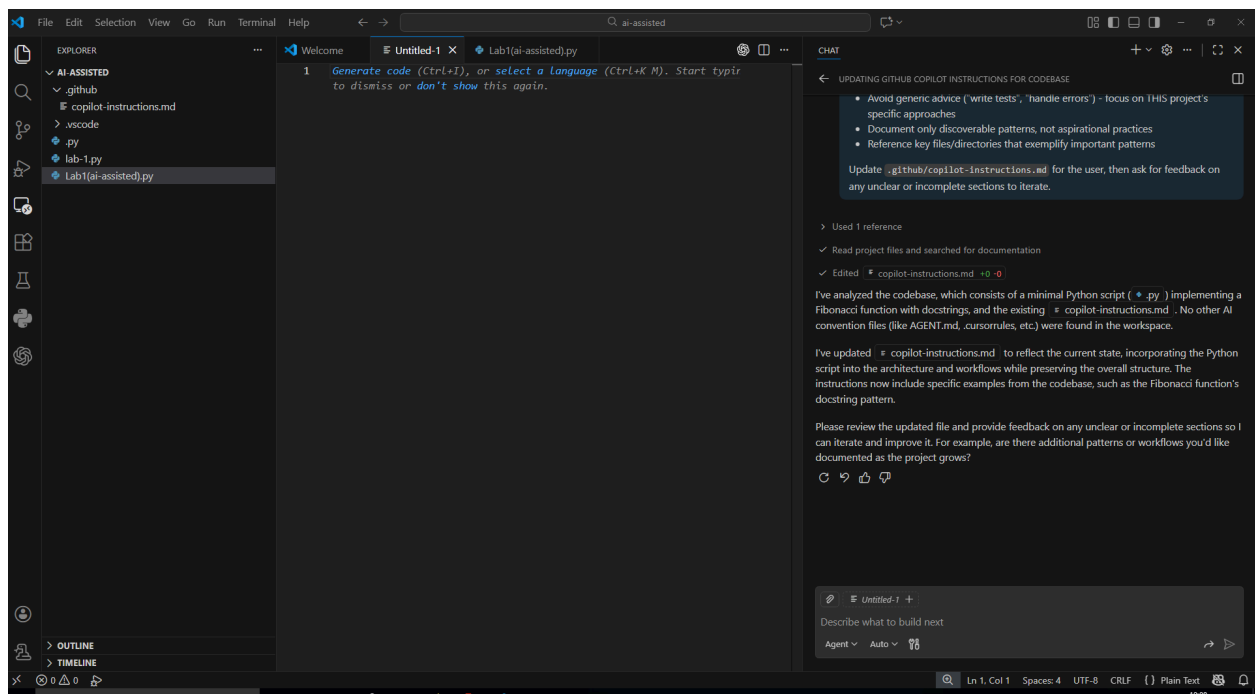
Batch No: 14

## Task 0

- Install and configure GitHub Copilot in VS Code. Take screenshots of each step.



- Install and configure GitHub Copilot in VS Code. Take screenshots of each step.



# Task 1: AI-Generated Logic Without Modularization (Prime Number Check Without Functions)

## ❖ Scenario

➤ You are developing a basic validation script for a numerical learning Application.

## ❖ Task Description

Use GitHub Copilot to generate a Python program that:

- Checks whether a given number is prime
- Accepts user input
- Implements logic directly in the main code
- Does not use any user-defined functions

## ❖ Expected Output

- Correct prime result

The screenshot displays the Visual Studio Code interface. The main editor window shows a Python file named `ai-assisted.py` with the following code:

```
1 """Checks whether a given number is prime or not.
2 Implements logic directly in the main code and does not use any user-defined functions.
3 """
4
5
6 try:
7     n = int(input("Enter an integer: ").strip())
8 except Exception:
9     print("Invalid input")
10 else:
11     if n <= 1:
12         print(f"{n} is not prime")
13     elif n == 2:
14         print("2 is prime")
15     elif n % 2 == 0:
16         print(f"{n} is not prime")
17     else:
18         is_prime = True
19         i = 3
20         import math
21         limit = int(math.isqrt(n))
22         while i <= limit:
23             if n % i == 0:
24                 is_prime = False
25                 break
26             i += 2
27         print(f"{n} is ('prime' if is_prime else 'not prime')")
```

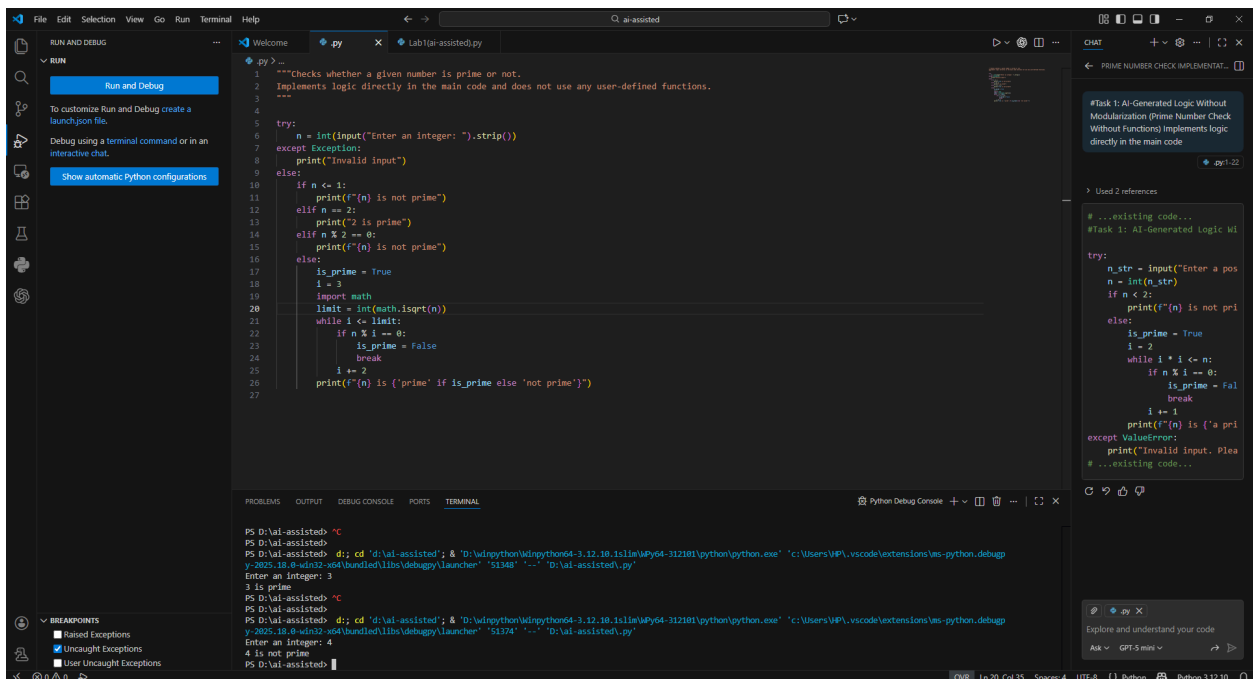
The terminal at the bottom shows the execution of the script:

```
PS D:\ai-assisted> ^C
PS D:\ai-assisted>
PS D:\ai-assisted> cd 'd:\ai-assisted'; & 'D:\winpython\winpython64-3.12.10.1\lib\MyPy04-312101\python\python.exe' 'c:\Users\VP\vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\unpacked\libs\debugpy\launcher' '51348' '...' 'D:\ai-assisted\ai.py'
Enter an integer: 3
3 is prime
PS D:\ai-assisted>
```

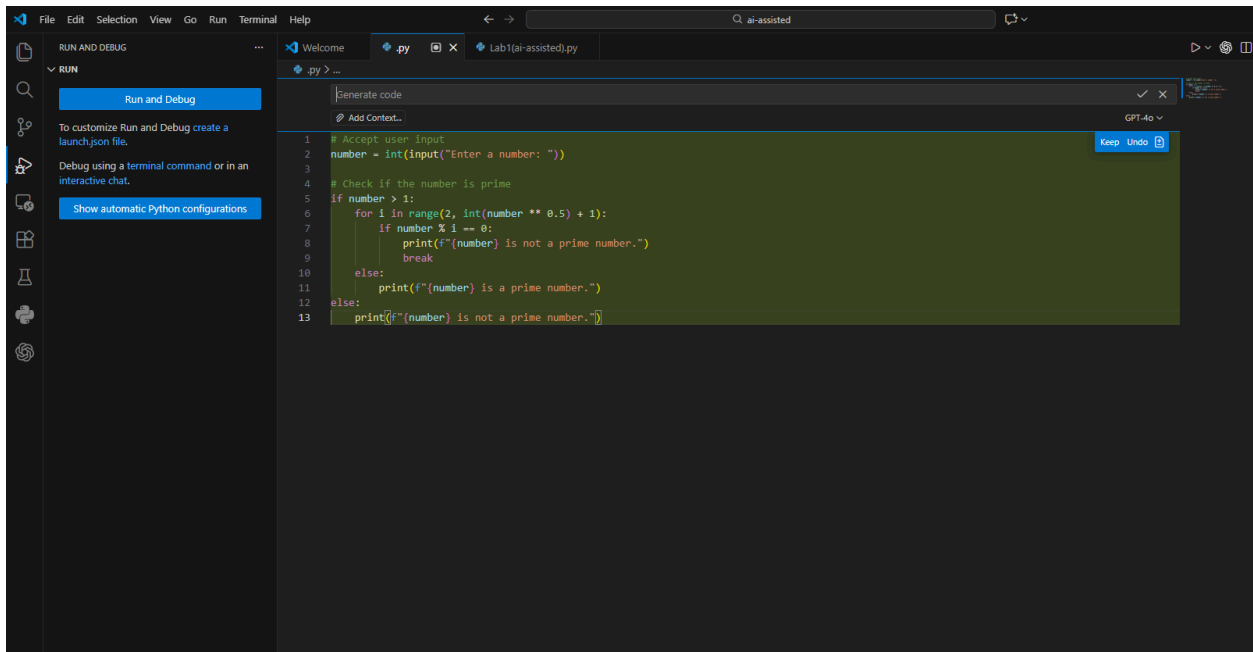
On the right side, the Chat window shows a conversation with GitHub Copilot. The user's prompt is: "#Task 1: AI-Generated Logic Without Modularization (Prime Number Check Without Functions) Implements logic directly in the main code". The AI's response is a Python script that implements the prime number check logic directly in the main code, without using any user-defined functions. The AI's code is as follows:

```
try:
    n_str = input("Enter a pos
    n = int(n_str)
    if n < 2:
        print(f"{n} is not pri
    else:
        is_prime = True
        i = 2
        while i * i <= n:
            if n % i == 0:
                is_prime = Fal
                break
            i += 1
        print(f"{n} is ('a pri
except ValueError:
    print("Invalid input. Plea
# ...existing code...
```

## ➤ Non-prime result



## ➤ Screenshots showing Copilot-generated code suggestions



## Task 2: Efficiency & Logic Optimization (Cleanup)

### ❖ Scenario

The script must handle larger input values efficiently.

### ❖ Task Description

Review the Copilot-generated code from Task 1 and improve it by:

- Reducing unnecessary iterations
- Optimizing the loop range (e.g., early termination)
- Improving readability
- Use Copilot prompts like:
  - “Optimize prime number checking logic”
  - “Improve efficiency of this code”

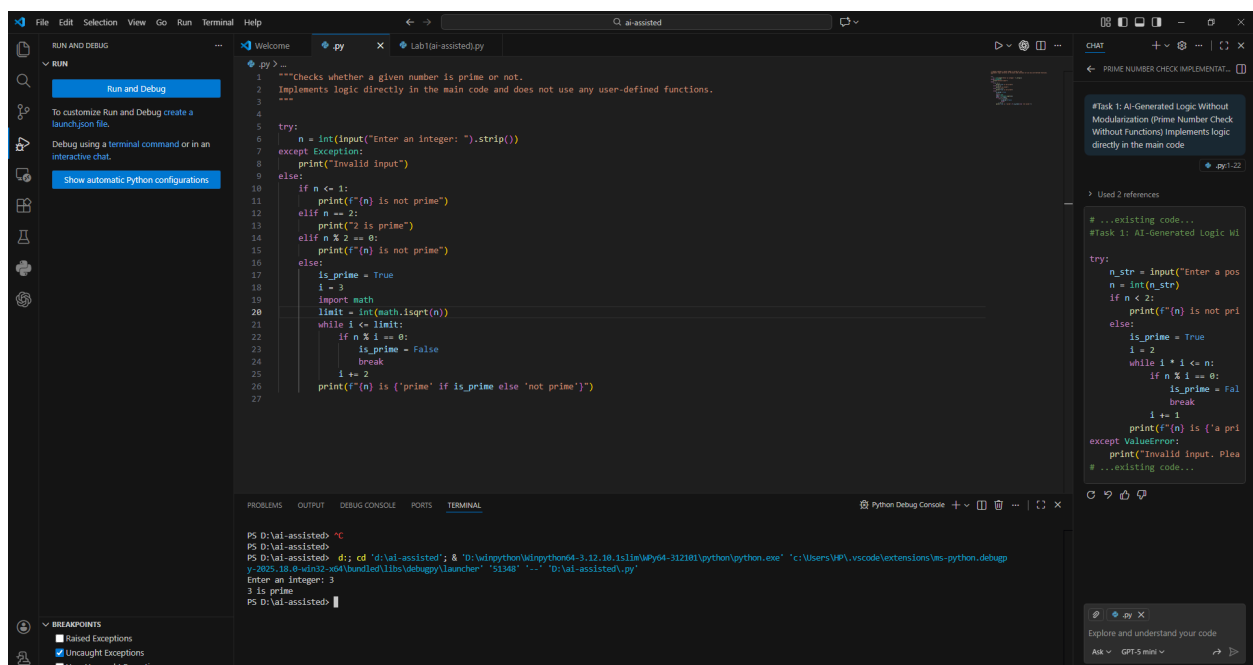
Hint:

Prompt Copilot with phrases like

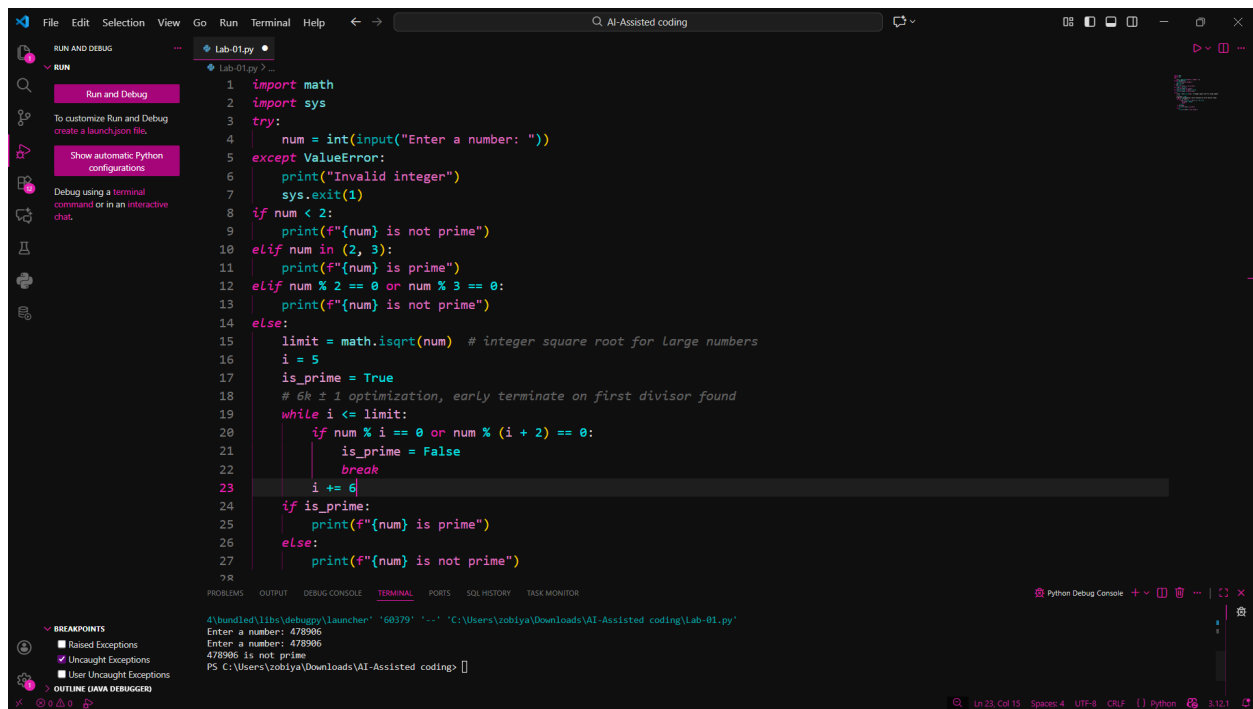
“optimize this code”, “simplify logic”, or “make it more readable”

### ❖ Expected Output

#### ➤ Original



## ➤ Optimized code versions



```
1 import math
2 import sys
3 try:
4     num = int(input("Enter a number: "))
5 except ValueError:
6     print("Invalid integer")
7     sys.exit(1)
8 if num < 2:
9     print(f"{num} is not prime")
10 elif num in (2, 3):
11     print(f"{num} is prime")
12 elif num % 2 == 0 or num % 3 == 0:
13     print(f"{num} is not prime")
14 else:
15     limit = math.isqrt(num) # integer square root for large numbers
16     i = 5
17     is_prime = True
18     # 6k ± 1 optimization, early terminate on first divisor found
19     while i <= limit:
20         if num % i == 0 or num % (i + 2) == 0:
21             is_prime = False
22             break
23         i += 6
24     if is_prime:
25         print(f"{num} is prime")
26     else:
27         print(f"{num} is not prime")
28
```

Python Debug Console

```
4\bundled\lib\debugpy\launcher '68379' '-' 'C:\Users\zobiya\Downloads\AI-Assisted coding\Lab-01.py'
Enter a number: 478906
Enter a number: 478906
478906 is not prime
PS C:\Users\zobiya\Downloads\AI-Assisted coding>
```

## ➤ Explanation of how the improvements reduce time complexity

### Time Complexity Improvements:

1. Early exit for  $n < 2$ ,  $n \in \{2, 3\}$ , and multiples of 2 or 3:  $O(1)$  checks for common cases.
2. Integer square root (isqrt): test divisors only up to  $\sqrt{n}$ , reducing checks from  $O(n)$  to  $O(\sqrt{n})$ .
3.  $6k \pm 1$  wheel: after removing 2 and 3, all primes have form  $6k \pm 1$ . Test only  $\sim 1/3$  of candidates.
  - Each iteration checks  $i$  and  $i+2$ , then jumps by 6, avoiding multiples of 2 and 3
4. Early termination: break on first divisor found, so composites with small factors exit quickly.

Result: Worst-case  $O(\sqrt{n})$ , but with constant factor  $\sim 1/3$  vs. naive trial division.

Efficient for moderate integers.

## Task 3: Modular Design Using AI Assistance (Prime Number Check Using Functions)

### ❖ Scenario

The prime-checking logic will be reused across multiple modules.

### ❖ Task Description

Use GitHub Copilot to generate a function-based Python program that:

- Uses a user-defined function to check primality
- Returns a Boolean value
- Includes meaningful comments (AI-assisted)

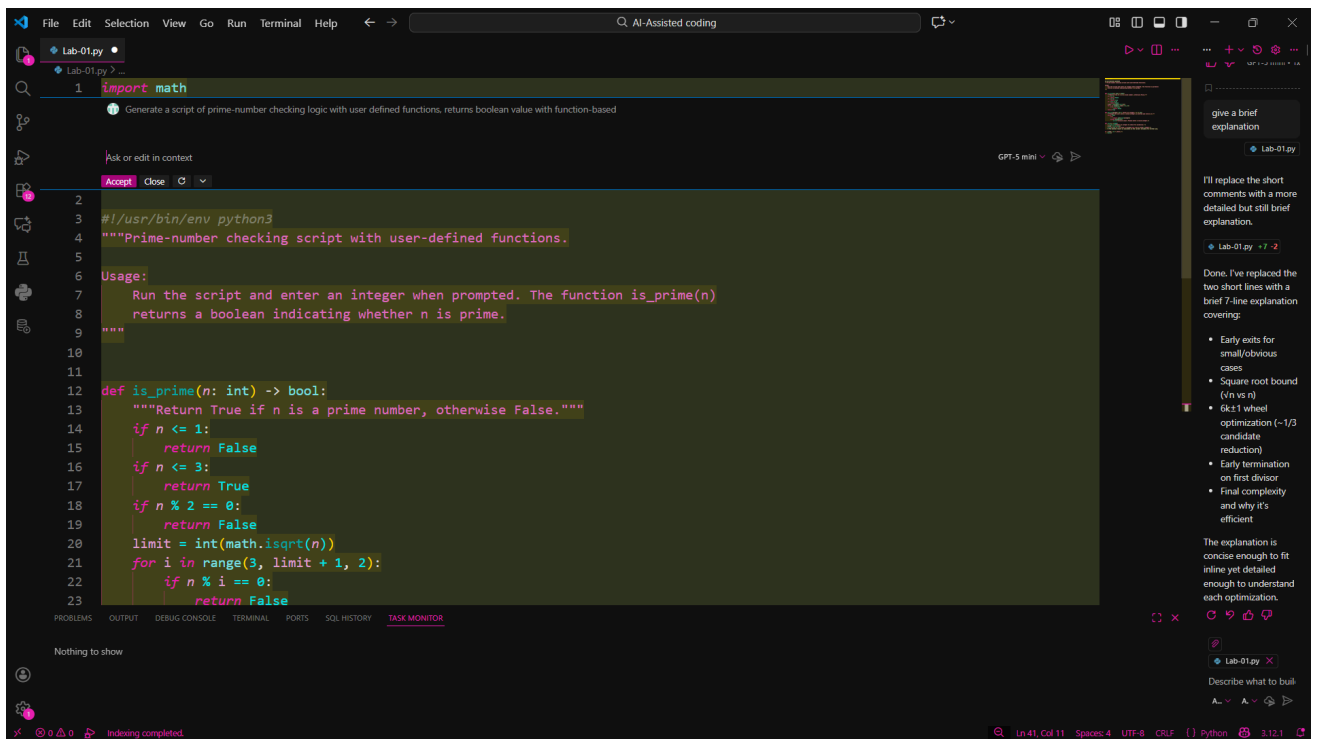
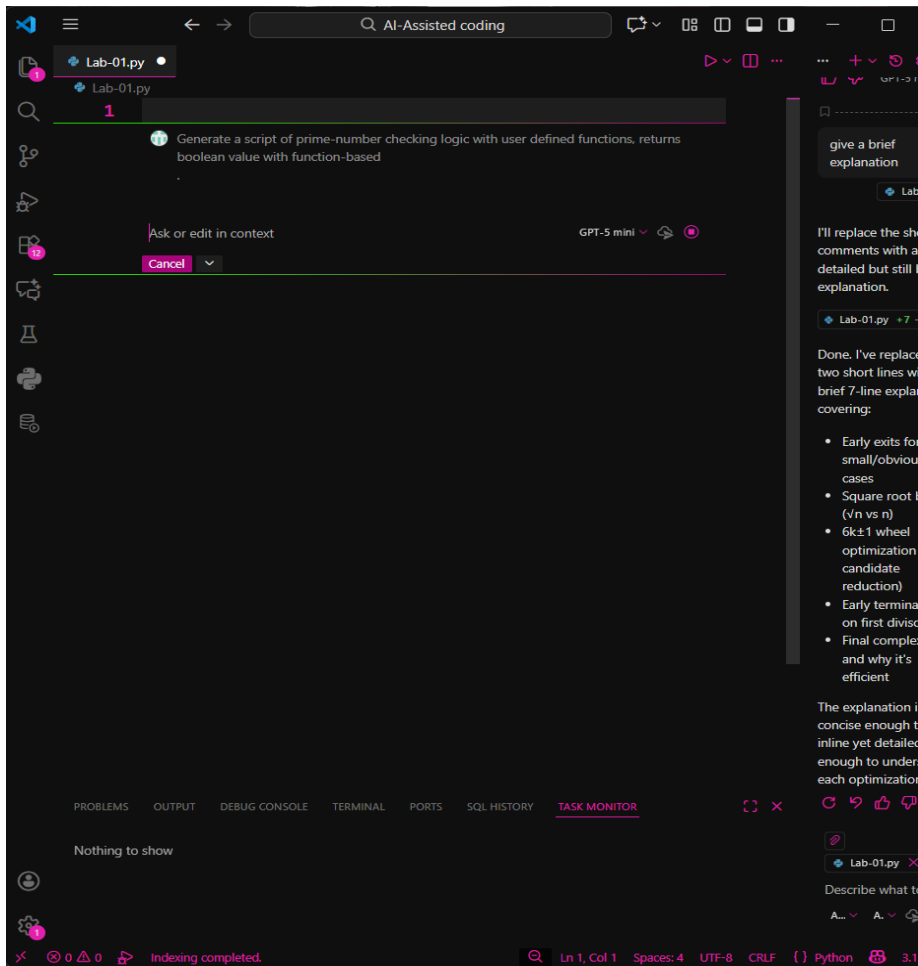
### ❖ Expected Output

- Correctly working prime-checking function

```
1 import math
2 #!/usr/bin/env python3
3 """Prime-number checking script with user-defined functions.
4 Usage:
5     Run the script and enter an integer when prompted. The function is_prime(n)
6     returns a boolean indicating whether n is prime.
7 """
8 def is_prime(n: int) -> bool:
9     """Return True if n is a prime number, otherwise False."""
10    if n <= 1:
11        return False
12    if n <= 3:
13        return True
14    if n % 2 == 0:
15        return False
16    limit = int(math.isqrt(n))
17    for i in range(3, limit + 1, 2):
18        if n % i == 0:
19            return False
20    return True
21 def get_int(prompt: str = "Enter an integer: ") -> int:
22     """Prompt the user until a valid integer is entered and return it."""
23     while True:
24         try:
25             return int(input(prompt))
26         except ValueError:
27             print("Invalid input. Please enter a valid integer.")
28
29 def main() -> None:
30     n = get_int("Enter an integer to check for primality: ")
31     prime = is_prime(n)
32     print(f"{n} is {'a prime' if prime else 'not a prime'} number.")
33     # The boolean result is available in the 'prime' variable for further use.
34 if __name__ == "__main__":
35     main()
```

Python Debug Console: Python312\python.exe: 'c:\Users\zobiy\vscode\extensions\ms-python.debugpy-2023.16.4-win32-x64\PS C:\Users\zobiy\Downloads\AI-Assisted coding> & 'c:\Users\zobiy\AppData\Local\Programs\Python\Python312\python.exe' 'c:\Users\zobiy\Downloads\AI-Assisted coding> cd 'c:\Users\zobiy\Downloads\AI-Assisted coding' & 'c:\Users\zobiy\AppData\Local\Programs\Python\Python312\python.exe' 'c:\Users\zobiy\Downloads\AI-Assisted coding\Lab-01.py' Enter an integer to check for primality: 54062 'c:\Users\zobiy\Downloads\AI-Assisted coding\Lab-01.py' 54062 is not a prime number.

## ➤ Screenshots documenting Copilot's function generation



## ➤ Sample test cases and outputs

The screenshot shows a VS Code editor with a Python script named `Lab-01.py`. The script defines a function `is_prime(n)` to check if a number is prime and a `main()` function to interact with the user. The terminal shows the execution of the script with test cases for 2864 and 1243467.

```
1 import math
2 #!/usr/bin/env python3
3 """Prime-number checking script with user-defined functions.
4 Usage:
5     Run the script and enter an integer when prompted. The function is_prime(n)
6     returns a boolean indicating whether n is prime.
7 """
8 def is_prime(n: int) -> bool:
9     """Return True if n is a prime number, otherwise False."""
10    if n <= 1:
11        return False
12    if n <= 3:
13        return True
14    if n % 2 == 0:
15        return False
16    limit = int(math.isqrt(n))
17    for i in range(3, limit + 1, 2):
18        if n % i == 0:
19            return False
20    return True
21 def get_int(prompt: str = "Enter an integer: ") -> int:
22     """Prompt the user until a valid integer is entered and return it."""
23     while True:
24         try:
25             return int(input(prompt))
26         except ValueError:
27             print("Invalid input. Please enter a valid integer.")
28
29 def main() -> None:
30     n = get_int("Enter an integer to check for primality: ")
31     prime = is_prime(n)
32     print(f"{n} is {'a prime' if prime else 'not a prime'} number.")
33     # The boolean result is available in the 'prime' variable for further use.
```

Terminal Output:

```
PS C:\Users\zobiya\Downloads\AI-Assisted coding> cd 'C:\Users\zobiya\Downloads\AI-Assisted coding' & 'C:\Users\zobiya\AppData\Local\Programs\Python\Python312\python.exe' 'C:\Users\zobiya\.vscode\extensions\ms-python.debugpy-2025.16.8-win32-x64\bu
ntion\lib\debugpy\launcher' '58638' '-c' 'C:\Users\zobiya\Downloads\AI-Assisted coding\Lab-01.py'
Enter an integer to check for primality: 2864
2864 is not a prime number.
PS C:\Users\zobiya\Downloads\AI-Assisted coding>
```

This screenshot shows the same VS Code environment with the `Lab-01.py` script. The terminal shows a new execution where the user enters 1243467, which is correctly identified as a prime number.

```
PS C:\Users\zobiya\Downloads\AI-Assisted coding> cd 'C:\Users\zobiya\Downloads\AI-Assisted coding' & 'C:\Users\zobiya\AppData\Local\Programs\Python\Python312\python.exe' 'C:\Users\zobiya\.vscode\extensions\ms-python.debugpy-2025.16.8-win32-x64\bu
ntion\lib\debugpy\launcher' '58638' '-c' 'C:\Users\zobiya\Downloads\AI-Assisted coding\Lab-01.py'
Enter an integer to check for primality: 1243467
1243467 is not a prime number.
PS C:\Users\zobiya\Downloads\AI-Assisted coding>
```

## Task 4: Comparative Analysis – With vs Without Functions

### ❖ Scenario

You are participating in a technical review discussion.

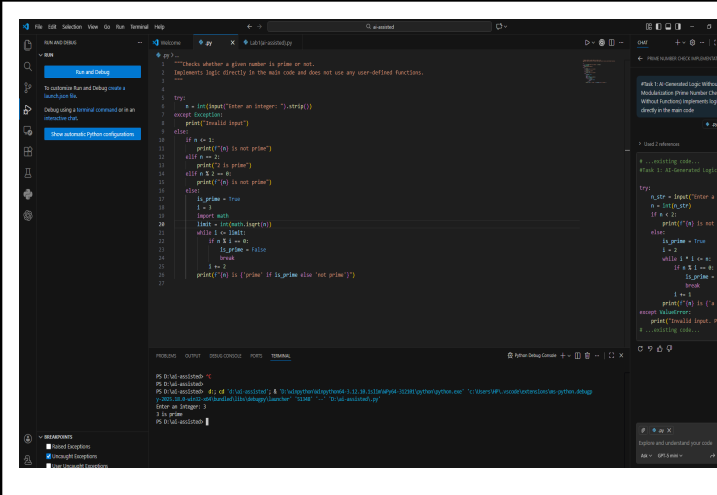
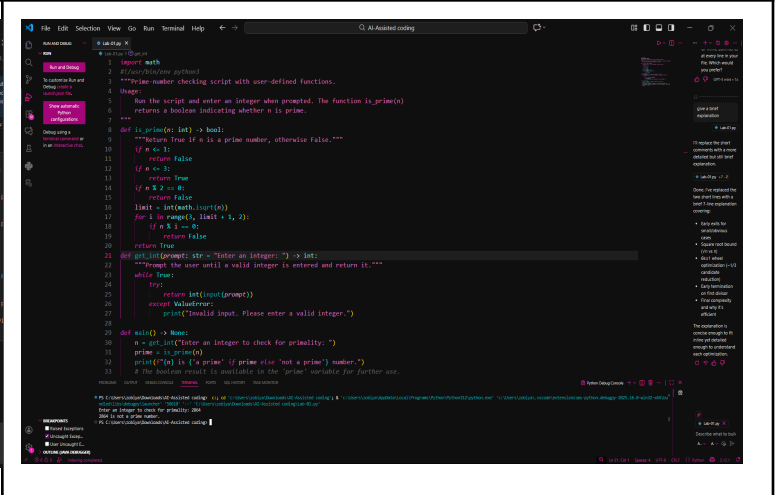
### ❖ Task Description

Compare the Copilot-generated programs:

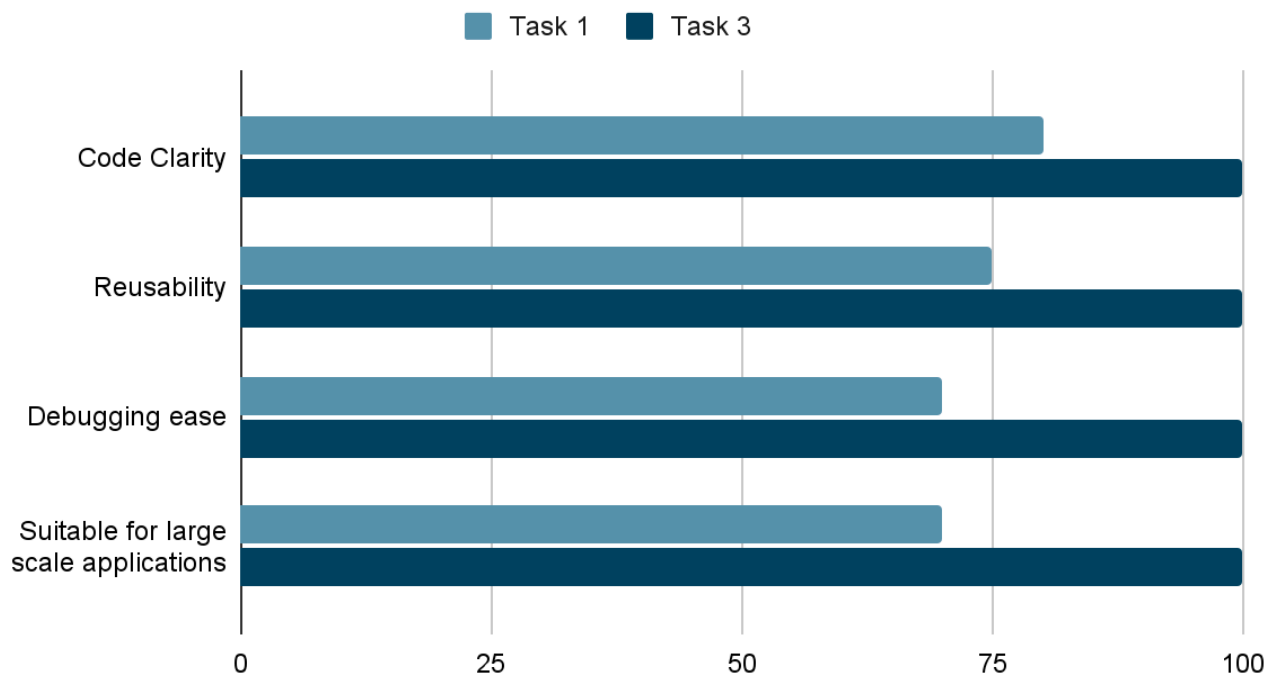
- Without functions (Task 1)
- With functions (Task 3)
- Analyze them based on:
  - Code clarity
  - Reusability
  - Debugging ease
  - Suitability for large-scale applications

### ❖ Expected Output

Comparison table or short analytical report

	
<h3>Task 1</h3>	<h3>Task 3</h3>
<ul style="list-style-type: none"><li>• Logic is correct, but dense</li><li>• Many things happen in one file scope</li><li>• Variable reuse (<code>is_prime</code> as both function concept and variable) can confuse beginners</li><li>• Harder to skim</li></ul>	<ul style="list-style-type: none"><li>• Meaningful function names: <code>is_prime</code>, <code>get_int</code></li><li>• Type hints (<code>n: int -&gt; bool</code>) improve readability</li><li>• Docstrings explain <i>what</i> each function does</li><li>• Logic is broken into small, understandable units</li></ul>

## Comparison of Task 1 VS Task 3



## Task 5: AI-Generated Iterative vs Recursive Fibonacci Approaches (Different Algorithmic Approaches to Prime Checking)

### ❖ Scenario

Your mentor wants to evaluate how AI handles alternative logical strategies.

### ❖ Task Description

Prompt GitHub Copilot to generate:

- A basic divisibility check approach
- An optimized approach (e.g., checking up to  $\sqrt{n}$ )

### ❖ Expected Output

- Two correct implementations

## ● Approach 1:

```
1 # Lab-01.py
2 # Basic prime checking by testing divisibility from 2 to n-1
3 def is_prime(n: int) -> bool:
4     """Return True if n is prime, False otherwise.
5     Uses a basic divisibility check from 2 to n-1.
6     """
7     if n <= 1:
8         return False
9     if n == 2:
10        return True
11    for i in range(2, n): # basic check as requested
12        if n % i == 0:
13            return False
14    return True
15
16 if __name__ == "__main__":
17     try:
18         s = input("Enter an integer to check for primality: ").strip()
19         num = int(s)
20     except ValueError:
21         print("Invalid integer.")
22     else:
23         print(f"{num} is prime." if is_prime(num) else f"{num} is not prime.")
```

2864 is not a prime number.  
PS C:\Users\zobiya\Downloads\AI-Assisted coding> cd 'c:\Users\zobiya\Downloads\AI-Assisted coding' & 'c:\Users\zobiya\AppData\Local\Programs\Python\Python312\python.exe' 'c:\Users\zobiya\vscode\extensions\ms-python.debugpy-2025.16.0-win32-x64\bundled\libs\debugpy\launcher' '58631' '-' 'C:\Users\zobiya\Downloads\AI-Assisted coding\Lab-01.py'  
Enter an integer to check for primality: 1243467  
1243467 is not a prime number.  
PS C:\Users\zobiya\Downloads\AI-Assisted coding> cd 'c:\Users\zobiya\Downloads\AI-Assisted coding' & 'c:\Users\zobiya\AppData\Local\Programs\Python\Python312\python.exe' 'c:\Users\zobiya\vscode\extensions\ms-python.debugpy-2025.16.0-win32-x64\bundled\libs\debugpy\launcher' '59223' '-' 'C:\Users\zobiya\Downloads\AI-Assisted coding\Lab-01.py'  
Enter an integer to check for primality: 456  
456 is not prime.  
PS C:\Users\zobiya\Downloads\AI-Assisted coding>

## ● Approach 2:

```
1 from math import isqrt
2 # Optimized prime checking up to sqrt(n)
3 def is_prime_sqrt(n: int) -> bool:
4     """Return True if n is prime, False otherwise.
5     Checks divisibility up to floor(sqrt(n)) and skips even numbers.
6     """
7     if n <= 1:
8         return False
9     if n <= 3:
10        return True
11    if n % 2 == 0:
12        return False
13    limit = isqrt(n)
14    for i in range(3, limit + 1, 2):
15        if n % i == 0:
16            return False
17    return True
18
19 if __name__ == "__main__":
20     try:
21         s = input("Enter an integer to check for primality: ").strip()
22         num = int(s)
23     except ValueError:
24         print("Invalid integer.")
25     else:
26         print(f"{num} is prime." if is_prime_sqrt(num) else f"{num} is not prime.")
```

PS C:\Users\zobiya\Downloads\AI-Assisted coding> cd 'c:\Users\zobiya\Downloads\AI-Assisted coding' & 'c:\Users\zobiya\AppData\Local\Programs\Python\Python312\python.exe' 'c:\Users\zobiya\vscode\extensions\ms-python.debugpy-2025.16.0-win32-x64\bundled\libs\debugpy\launcher' '53833' '-' 'C:\Users\zobiya\Downloads\AI-Assisted coding\Lab-01.py'  
Enter an integer to check for primality: 543  
543 is not prime.  
PS C:\Users\zobiya\Downloads\AI-Assisted coding>

➤ Comparison discussing:

Basic divisibility check	Optimized Approach
<ul style="list-style-type: none"><li>▪ <b>Execution flow:</b> The algorithm checks divisibility of the number by every integer from 2 up to <math>n-1</math>. It stops immediately when a divisor is found.</li></ul>	<ul style="list-style-type: none"><li>▪ <b>Execution flow:</b> The algorithm checks divisibility only up to <math>\sqrt{n}</math> and skips even numbers. This reduces the number of iterations significantly.</li></ul>
<ul style="list-style-type: none"><li>▪ <b>Time complexity:</b> Time complexity is <math>O(n)</math> since it may check almost all numbers up to <math>n</math>.</li></ul>	<ul style="list-style-type: none"><li>▪ <b>Time complexity:</b> Time complexity is <math>O(\sqrt{n})</math>, which is much more efficient.</li></ul>
<ul style="list-style-type: none"><li>▪ <b>Performance for large inputs:</b> Performance degrades significantly as input size increases. Large numbers take a long time to evaluate.</li></ul>	<ul style="list-style-type: none"><li>▪ <b>Performance for large inputs:</b> Performs well even for large input values. Execution time increases slowly as <math>n</math> grows.</li></ul>
<ul style="list-style-type: none"><li>▪ <b>When each approach is appropriate:</b> Suitable for learning purposes and very small input values. Useful when simplicity is more important than efficiency.</li></ul>	<ul style="list-style-type: none"><li>▪ <b>When each approach is appropriate:</b> Suitable for real-world applications and large datasets. Preferred when performance and scalability are important.</li></ul>

