

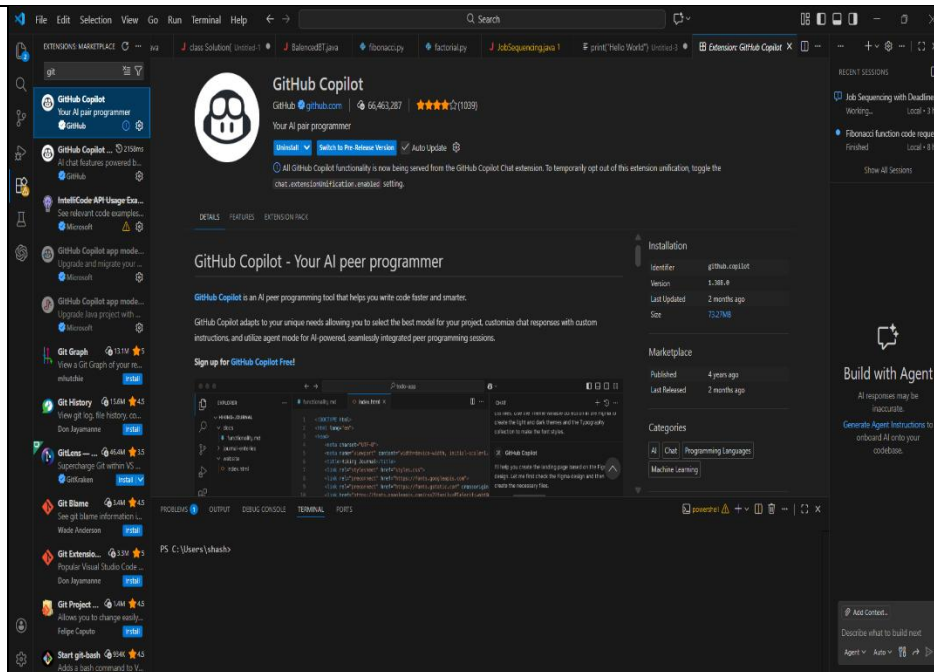
Name:Kruthankiran

H.NO:2303A51404

Batch:26

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech		Assignment Type: Lab	Academic Year:2025-2026
Course Coordinator Name		Dr. Rishabh Mittal	
Instructor(s) Name		Mr. S Naresh Kumar	
		Ms. B. Swathi	
		Dr. Sasanko Shekhar Gantayat	
		Mr. Md Sallauddin	
		Dr. Mathivanan	
		Mr. Y Srikanth	
		Ms. N Shilpa	
		Dr. Rishabh Mittal (Coordinator)	
		Dr. R. Prashant Kumar	
		Mr. Ankushavali MD	
		Mr. B Viswanath	
		Ms. Sujitha Reddy	
		Ms. A. Anitha	
		Ms. M.Madhuri	
		Ms. Katherashala Swetha	
		Ms. Velpula sumalatha	
Mr. Bingi Raju			
CourseCode	23CS002PC304	Course Title	AI Assisted Coding
Year/Sem	III/II	Regulation	R23
Date and Day of Assignment	Week1 – Thursday	Time(s)	23CSBTB01 To 23CSBTB52
Duration	2 Hours	Applicable to Batches	All batches
Assignment Number:1.3(Present assignment number)/24(Total number of assignments)			
Q.No.	Question		Expected Time to complete
1	Lab 1: Environment Setup – <i>GitHub Copilot and VS Code Integration +</i>		Week1 -

	<p><i>Understanding AI-assisted Coding Workflow</i></p> <p><b>Lab Objectives:</b></p> <ul style="list-style-type: none"> <li>● To install and configure GitHub Copilot in Visual Studio Code.</li> <li>● To explore AI-assisted code generation using GitHub Copilot.</li> <li>● To analyze the accuracy and effectiveness of Copilot's code suggestions.</li> <li>● To understand prompt-based programming using comments and code context</li> </ul> <p><b>Lab Outcomes (LOs):</b> After completing this lab, students will be able to:</p> <ul style="list-style-type: none"> <li>● Set up GitHub Copilot in VS Code successfully.</li> <li>● Use inline comments and context to generate code with Copilot.</li> <li>● Evaluate AI-generated code for correctness and readability.</li> <li>● Compare code suggestions based on different prompts and programming styles.</li> </ul>	Monday
	<p>Task 0</p> <ul style="list-style-type: none"> <li>● Install and configure GitHub Copilot in VS Code. Take screenshots of each step.</li> </ul> <p>Expected Output</p> <ul style="list-style-type: none"> <li>● Install and configure GitHub Copilot in VS Code. Take screenshots of each step.</li> </ul>	



## 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 / non-prime result
- Screenshots showing Copilot-generated code suggestions
- Sample inputs and outputs

```
J ai:java
1  number = int(input("Enter a number: "))
2
3  if number < 2:
4      print("Not Prime")
5  else:
6      is_prime = True
7      for i in range(2, number):
8          if number % i == 0:
9              is_prime = False
10             break
11
12     if is_prime:
13         print("Prime")
14     else:
15         print("Not Prime")
16
```

PROBLEMS 32 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\kruth\OneDrive\Desktop\java> & 'c:\Users\kruth\AppData\Local\Microsoft\
ebugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '64496' '--' 'C:\Users\k
Enter a number: 7
Prime
PS C:\Users\kruth\OneDrive\Desktop\java> ^C
PS C:\Users\kruth\OneDrive\Desktop\java>
PS C:\Users\kruth\OneDrive\Desktop\java> c:: cd 'c:\Users\kruth\OneDrive\Desktop\
'c:\Users\kruth\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\
.java'
Enter a number: 10
Not Prime
PS C:\Users\kruth\OneDrive\Desktop\java> 
```

## 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 and optimized code versions
- Explanation of how the improvements reduce time complexity

```
J ai.java
1  number = int(input("Enter a number: "))
2
3  if number < 2:
4      print("Not Prime")
5  else:
6      is_prime = True
7      for i in range(2, number):
8          if number % i == 0:
9              is_prime = False
10             break
11
12     if is_prime:
13         print("Prime")
14     else:
15         print("Not Prime")
16
```

PROBLEMS 32 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\kruth\OneDrive\Desktop\java> & 'c:\Users\kruth\AppData\Local\
ebugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '64496' '--' 'c:
Enter a number: 7
Prime
PS C:\Users\kruth\OneDrive\Desktop\java> ^C
PS C:\Users\kruth\OneDrive\Desktop\java>
PS C:\Users\kruth\OneDrive\Desktop\java> c:: cd 'c:\Users\kruth\OneDrive\
'c:\Users\kruth\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\
.java'
Enter a number: 10
Not Prime
PS C:\Users\kruth\OneDrive\Desktop\java> 
```

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
  - Screenshots documenting Copilot’s function generation
  - Sample test cases and outputs

class Solution.java   fibonacci.py   fibonacci1.py   factorial.py   J

J   Ai2.java

11   `return False # Numbers less than 2 are not prime`

12

13   `# Check for divisibility up to square root of n`

14   `for i in range(2, int(math.sqrt(n)) + 1):`

15   `if n % i == 0:`

16   `return False # Found a divisor`

17

18   `return True # No divisors found, it's prime`

19

20

21   `# Main program`

22   `if __name__ == "__main__":`

23   `number = int(input("Enter a number: "))`

24   `if is_prime(number):`

25   `print(f"{number} is Prime")`

26   `else:`

27   `print(f"{number} is Not Prime")`

28

PROBLEMS 63   OUTPUT   DEBUG CONSOLE   **TERMINAL**   PORTS

PS C:\Users\kruth\OneDrive\Desktop\java> c;; cd 'c:\Users\kruth\OneDrive\Deskt  
'c:\Users\kruth\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundl  
2.java'  
Enter a number: 7  
7 is Prime  
PS C:\Users\kruth\OneDrive\Desktop\java> ^C  
PS C:\Users\kruth\OneDrive\Desktop\java>  
PS C:\Users\kruth\OneDrive\Desktop\java> c;; cd 'c:\Users\kruth\OneDrive\Deskt  
'c:\Users\kruth\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundl  
2.java'  
Enter a number: 10  
10 is Not Prime  
PS C:\Users\kruth\OneDrive\Desktop\java> |

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

```

C:\> java saves > task 2.py > ...
1  import math
2  import time # For timing execution to empirically compare efficiency
3
4  # === TASK 1 APPROACH: INLINE LOGIC (NO FUNCTIONS) ===
5  # This is the non-modular version: All logic in main block.
6  # Pros: Simple for one-off scripts. Cons: Hard to reuse/debug.
7  def run_inline_prime_check():
8      print("\n--- Task 1: Inline Logic (No Functions) ---")
9      number = int(input("Enter a number for inline check: "))
10
11     start_time = time.time()
12
13     if number < 2:
14         print("Not Prime")
15     else:
16         is_prime = True
17         # Basic loop: Checks up to sqrt(n) for efficiency (as optimized in Task 2)
18         for i in range(2, int(math.sqrt(number)) + 1):
19             if number % i == 0:
20                 is_prime = False
21                 break
22         if is_prime:
23             print("Prime")
24         else:
25             print("Not Prime")
26
27     end_time = time.time()
28     print(f"Execution time: {end_time - start_time:.6f} seconds")
29
30 # === TASK 3 APPROACH: MODULAR WITH FUNCTIONS ===
31 # This is the reusable version: Logic encapsulated in a function.
32 # Pros: Reusable, easier to test/debug. Cons: Slight overhead for tiny scripts.

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\java saves> ^C
PS C:\java saves>
PS C:\java saves> c; cd 'c:\java saves'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' 025.18.0-win32-x64\bundled\libs\debugpy\launcher' '64514' '--' 'c:\java saves\task 2.py'
Task 4: Comparative Analysis Runner
Running both approaches... (Enter same number for fair comparison)

--- Task 1: Inline Logic (No Functions) ---
Enter a number for inline check: 997
Prime
Execution time: 0.000000 seconds
```

```
... DArray.java class Solution[ Untitled-1 BalancedBT.java fibonacci.py factorial.py JobSequenc

C:\> java saves > task 2.py > ...

30 # === TASK 3 APPROACH: MODULAR WITH FUNCTIONS ===
31 # This is the reusable version: Logic encapsulated in a function.
32 # Pros: Reusable, easier to test/debug. Cons: Slight overhead for tiny scripts.
33 def is_prime_modular(n):
34     """
35     Checks if n is a prime number.
36     Returns True if prime, False otherwise.
37     Optimized by checking divisors up to sqrt(n).
38     """
39     if n < 2:
40         return False
41     for i in range(2, int(math.sqrt(n)) + 1):
42         if n % i == 0:
43             return False
44     return True
45
46 def run_modular_prime_check():
47     print("\n--- Task 3: Modular with Functions ---")
48     number = int(input("Enter a number for modular check: "))
49
50     start_time = time.time()
51
52     result = is_prime_modular(number)
53     if result:
54         print("Prime")
55     else:
56         print("Not Prime")
57
58     end_time = time.time()
59     print(f"Execution time: {end_time - start_time:.6f} seconds")
60

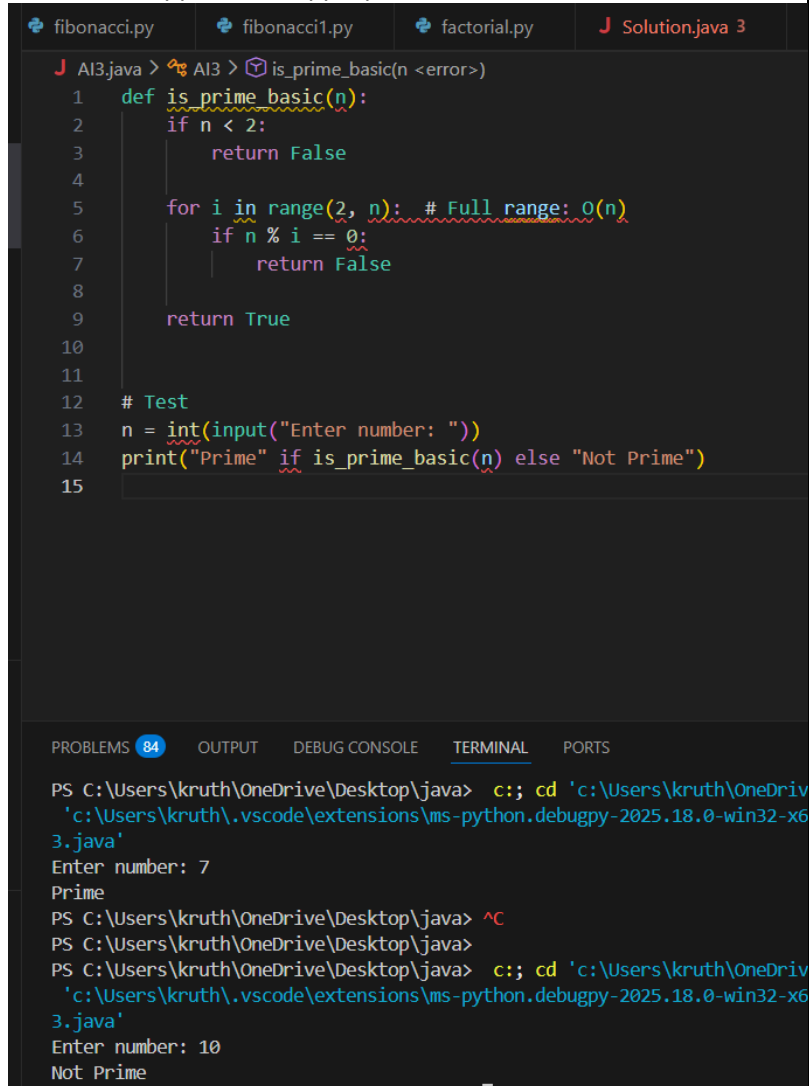
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

--- Task 1: Inline Logic (No Functions) ---
Enter a number for inline check: 997
Prime
Execution time: 0.000000 seconds

--- Task 3: Modular with Functions ---
Enter a number for modular check: 997
Prime
Execution time: 0.000000 seconds
```

	 <pre>Run Terminal Help ← → Q Search Array.java J class Solution[ Untitled-1 J BalancedBT.java fibonacci.py factorial.py J Job C:\java saves&gt; task 2.py &gt; ... 46 def run_modular_prime_check(): 52     result = is_prime_modular(number) 53     if result: 54         print("Prime") 55     else: 56         print("Not Prime") 57 58     end_time = time.time() 59     print(f"Execution time: {end_time - start_time:.6f} seconds") 60 61 # === MAIN RUNNER: Executes both for comparison === 62 if __name__ == "__main__": 63     print("Task 4: Comparative Analysis Runner") 64     print("Running both approaches... (Enter same number for fair comparison)") 65 66     run_inline_prime_check() 67     run_modular_prime_check() 68 69 # Simple text-based comparison summary (could be expanded with Copilot) 70 print("\n--- Quick Comparison Summary ---") 71 print("Code Clarity: Modular &gt; Inline (separation of concerns)") 72 print("Reusability: Modular &gt;&gt; Inline (call function anywhere)") 73 print("Debugging Ease: Modular &gt; Inline (test function independently)") 74 print("Suitability for Large-Scale: Modular &gt;&gt; Inline (promotes clean architecture)")  PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS  --- Task 3: Modular with Functions --- Enter a number for modular check: 997 Prime Execution time: 0.000000 seconds  --- Quick Comparison Summary --- Code Clarity: Modular &gt; Inline (separation of concerns) Reusability: Modular &gt;&gt; Inline (call function anywhere) Debugging Ease: Modular &gt; Inline (test function independently) Suitability for Large-Scale: Modular &gt;&gt; Inline (promotes clean architecture) PS C:\java saves&gt;</pre>	
	<p>Task 5: AI-Generated Iterative vs Recursive Fibonacci Approaches (Different Algorithmic Approaches to Prime Checking)</p> <ul style="list-style-type: none"><li>❖ <b>Scenario</b> Your mentor wants to evaluate how AI handles <b>alternative logical strategies</b>.</li><li>❖ <b>Task Description</b> Prompt GitHub Copilot to generate:<ul style="list-style-type: none"><li>➤ A <b>basic divisibility check</b> approach</li><li>➤ An <b>optimized approach</b> (e.g., checking up to <math>\sqrt{n}</math>)</li></ul></li><li>❖ <b>Expected Output</b></li></ul>	

- Two correct implementations
- Comparison discussing:
  - Execution flow
  - Time complexity
  - Performance for large inputs
  - When each approach is appropriate



```
fibonacci.py  fibonacci1.py  factorial.py  Solution.java 3
J AI3.java > AI3 > is_prime_basic(n <error>)
1  def is_prime_basic(n):
2      if n < 2:
3          return False
4
5      for i in range(2, n): # Full range: 0(n)
6          if n % i == 0:
7              return False
8
9      return True
10
11
12 # Test
13 n = int(input("Enter number: "))
14 print("Prime" if is_prime_basic(n) else "Not Prime")
15

PROBLEMS 84  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
PS C:\Users\kruth\OneDrive\Desktop\java> c;; cd 'c:\Users\kruth\OneDrive\Desktop\java'
PS C:\Users\kruth\OneDrive\Desktop\java> Enter number: 7
PS C:\Users\kruth\OneDrive\Desktop\java> Prime
PS C:\Users\kruth\OneDrive\Desktop\java> ^C
PS C:\Users\kruth\OneDrive\Desktop\java>
PS C:\Users\kruth\OneDrive\Desktop\java> c;; cd 'c:\Users\kruth\OneDrive\Desktop\java'
PS C:\Users\kruth\OneDrive\Desktop\java> Enter number: 10
PS C:\Users\kruth\OneDrive\Desktop\java> Not Prime
```

```
fibonacci.py  fibonacci1.py  factorial.py  J Solution.java 3

J AI3.java > AI3 > is_prime_basic(n <error>)
1  def is_prime_basic(n):
2      if n < 2:
3          return False
4
5      for i in range(2, n): # Full range: 0(n)
6          if n % i == 0:
7              return False
8
9      return True
10
11
12 # Test
13 n = int(input("Enter number: "))
14 print("Prime" if is_prime_basic(n) else "Not Prime")
15

PROBLEMS 84  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\kruth\OneDrive\Desktop\java> c;; cd 'c:\Users\kruth\OneDrive
'c:\Users\kruth\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x6
3.java'
Enter number: 7
Prime
PS C:\Users\kruth\OneDrive\Desktop\java> ^C
PS C:\Users\kruth\OneDrive\Desktop\java>
PS C:\Users\kruth\OneDrive\Desktop\java> c;; cd 'c:\Users\kruth\OneDrive
'c:\Users\kruth\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x6
3.java'
Enter number: 10
Not Prime
```

■

■

**Note:** Report should be submitted as a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots.