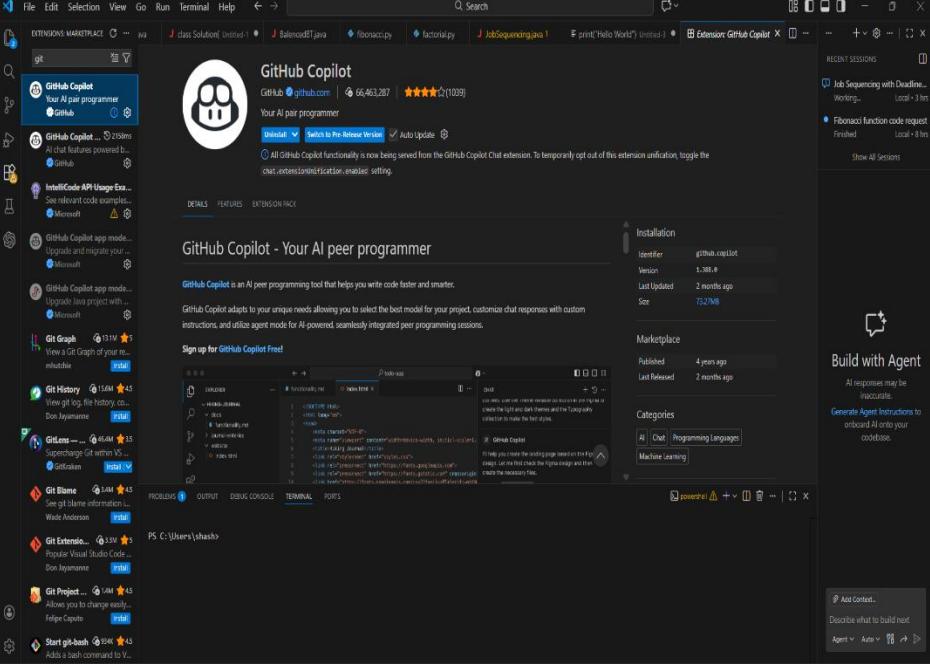


Name:Kruthankiran
H.NO:2303A51404
Batch:26

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech		Assignment Type: Lab	
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 – GitHub Copilot and VS Code Integration +		Week1 -

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



The screenshot shows the GitHub Copilot extension page in the VS Code Marketplace. The extension has a rating of 5 stars (1039 reviews) and over 66,463,287 installs. It is described as an AI peer programmer. The interface includes sections for Installation (version 1.108.0, last updated 2 months ago, size 7527MB), Marketplace (published 4 years ago, last released 2 months ago), and Categories (Chat, Programming Languages, Machine Learning). On the right, there's a 'Build with Agent' section with a 'Generate Agent Instructions' button.

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\debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '64496' '--' 'C:\Users\kruth\OneDrive\Desktop\ai.java'
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\ai.java'
PS C:\Users\kruth\OneDrive\Desktop\java> ^C
PS C:\Users\kruth\OneDrive\Desktop\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

The screenshot shows a Visual Studio Code interface. On the left is a code editor with a dark theme containing the following Java code:

```
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
```

Below the code editor is a terminal window showing the output of running the program. The terminal tab is selected at the top.

```
PROBLEMS 32 OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\kruth\OneDrive\Desktop\java> & 'c:\Users\kruth\AppData\Local\debugpy-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

The screenshot shows a code editor with a Java file named `Ai2.java`. The code defines a function `is_prime` that checks if a number is prime by iterating from 2 to the square root of the number. It prints "True" if no divisors are found and "False" if any divisor is found. The main program prompts the user for a number, calls the `is_prime` function, and prints the result. Below the code editor is a terminal window showing the execution of the Java code. The terminal output shows two runs: one where the user enters 7 and gets "7 is Prime", and another where the user enters 10 and gets "10 is Not Prime".

```
PROBLEMS 63 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\kruth\OneDrive\Desktop\java> c:; cd 'c:\Users\kruth\OneDrive\Desktop\java> c:; cd '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\Desktop\java> c:; cd '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

The screenshot shows a code editor interface with a dark theme. On the left, there is a code editor window displaying a Python script named `task 2.py`. The script performs a prime number check using two different approaches: inline logic (Task 1) and modular functions (Task 3). The right side of the interface shows a terminal window with the command `java saves > task 2.py` and its execution results, including the prime check for the number 997 and the execution time.

```
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("Execution 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.
```

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

The screenshot shows a code editor interface with a dark theme. The main area displays a Python script named `task 2.py`. The code implements a modular approach to prime number checking, using functions for better reuse and readability.

```
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
```

Below the code editor, there is a terminal window showing the execution of the script. It first runs a task for inline logic (no functions) and then runs the modular approach task. Both tasks prompt for a number (997), output "Prime", and show an execution time of 0.000000 seconds.

```
--- 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
```

The screenshot shows a terminal window with the following content:

```

Run Terminal Help ← → Search
Array.java J class Solution( Untitled-1 ● J BalencedBT.java fibonaci.py factorial.py J Job
C:\> java saves > task 2.py > ...
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 > Inline (separation of concerns)")
72     print("Reusability: Modular >> Inline (call function anywhere)")
73     print("Debugging Ease: Modular > Inline (test function independently)")
74     print("Suitability for Large-Scale: Modular >> 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 > Inline (separation of concerns)
Reusability: Modular >> Inline (call function anywhere)
Debugging Ease: Modular > Inline (test function independently)
Suitability for Large-Scale: Modular >> Inline (promotes clean architecture)
PS C:\java saves>

```

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

The screenshot shows a code editor with three tabs: `fibonacci.py`, `fibonacci1.py`, and `factorial.py`. The `fibonacci1.py` tab is active, displaying the following Python code:

```

J AI3.java > 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: O(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

```

Below the code editor is a terminal window showing the output of a Java program. The terminal interface includes tabs for PROBLEMS (84), OUTPUT, DEBUG CONSOLE, TERMINAL (which is selected), and PORTS.

```

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

```

The screenshot shows a code editor window in VS Code with several tabs at the top: fibonacci.py, fibonacci1.py, factorial.py, and Solution.java 3. The main editor area contains the following Python code:

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

Below the code editor is a terminal window showing the execution of the script:

```
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-x64\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-x64\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.