

NAME:G.Bhagath H.NO:2303A51807 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 + Understanding AI-assisted Coding Workflow		Week1 - Monday

	<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. <hr/> <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 Microsoft Visual Studio Code (VS Code) interface. At the top, there's a menu bar with File, Edit, Selection, View, Go, Run, Terminal, Help, and a search bar. Below the menu is a tab bar with several open files: EXTENSIONS MARKETPLACE, class Solution1 (Untitled-1), J SaleroDJava, fibonaccipy, factorial.py, JobSequencing.java, print("Hello World") (Untitled-3), and Extension: GitHub Copilot (Untitled-4). The main area displays the GitHub Copilot extension details. The extension icon is a white robot head inside a blue circle. It has a rating of 4.5 stars (1039 reviews). The description reads "Your AI peer programmer". There are tabs for DETAILS, FEATURES, and EXTENSION PAGE. The DETAILS tab shows a brief description: "GitHub Copilot is an AI peer programming tool that helps you write code faster and smarter. GitHub Copilot adapts to your unique needs allowing you to select the best model for your project, customize chat responses with custom instructions, and utilize agent mode for AI-powered, seamlessly integrated peer programming sessions." Below this is a "Sign up for GitHub Copilot Free!" button. To the right, there's an "Installation" section with details: Identifier: github.copilot, Version: 1.100.0, Last Updated: 2 months ago, Size: 7527MB. Further right is a "Marketplace" section with a "Build with Agent" button and a note about AI responses being inaccurate. At the bottom, there's a "Add Context..." button and a "Describe what to build next" input field.

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

The screenshot shows the VS Code interface with a Python file named `task 2.py` open. The code checks if a given number is prime by testing divisibility from 2 up to the number itself. If any divisor is found, it prints "Not Prime" and exits early. Otherwise, it prints "Prime". The terminal below shows the script being run and a user inputting the number 7, which is correctly identified as prime.

```

Selection View Go Run Terminal Help ← → Q Search
OPENED DArray.java J class Solution( Untitled-1 ● J BalancedBT.java fibonaci.py factorial.py J JobSequencing.java task1.py task 2
C:\> java saves & task 2.py > ...
1 # Import necessary modules (none needed here)
2 number = int(input("Enter a number: "))
3
4 # Check if number is less than 2 (not prime)
5 if number < 2:
6     print("Not Prime")
7 else:
8     is_prime = True # Assume it's prime initially
9
10 # Check divisibility from 2 to number-1
11 for i in range(2, number):
12     if number % i == 0:
13         is_prime = False # Found a divisor, not prime
14         break # Early exit if divisor found
15
16 # Output result
17 if is_prime:
18     print("Prime")
19 else:
20     print("Not Prime")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\java saves> & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' 'c:\Users\shash\.vscode\extensions\ms-python.debugpy-2025.18.0\libs\debugpy\launcher' '54755' '--' 'c:\java saves\task 2.py'
Enter a number: 7
Prime
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' 'c:\Users\shash\.vscode\extensions\ms-python.debugpy-2025.18.0\libs\debugpy\launcher' '54816' '--' 'c:\java saves\task 2.py'
Enter a number: 10
Not Prime
PS C:\java saves>

```

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

```

  ... DArray.java   J class Solution{ Untitled-1 ● J BalancedBT.java   fibonacci.py   factorial.py
C > java saves > task 2.py > ...
1 import math # For sqrt function
2
3 number = int(input("Enter a number: "))
4
5 if number < 2:
6     print("Not Prime")
7 else:
8     is_prime = True
9
10    # Optimized: Check divisors only up to sqrt(number)
11    for i in range(2, int(math.sqrt(number)) + 1):
12        if number % i == 0:
13            is_prime = False
14            break
15
16        if is_prime:
17            print("Prime")
18        else:
19            print("Not Prime")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

PS C:\java saves>
PS C:\java saves> c;; cd 'c:\java saves'; & 'c:\Users\shash\anaconda3\envs\Shashidhar_025.18.0-win32-x64\bundled\libs\debugpy\launcher' '61305' '--' 'c:\java saves\task 2.py'
Enter a number: 7
Prime
PS C:\java saves> ^C
PS C:\java saves>
PS C:\java saves> c;; cd 'c:\java saves'; & 'c:\Users\shash\anaconda3\envs\Shashidhar_025.18.0-win32-x64\bundled\libs\debugpy\launcher' '61339' '--' 'c:\java saves\task 2.py'
Enter a number: 10
Not Prime
PS C:\java saves>

```

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 interface with a dark theme. In the top navigation bar, there are several tabs: DArray.java, class Solution (Untitled-1), BalancedBT.java, fibonacci.py, factorial.py, and JobSeq. The main area displays the following Python code:

```

C:\> java saves > task 2.py > ...
1 import math
2
3 # Function to check if a number is prime using optimized logic
4 def is_prime(n):
5     """
6         Checks if n is a prime number.
7         Returns True if prime, False otherwise.
8         Optimized by checking divisors up to sqrt(n).
9     """
10    if n < 2:
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 # Main program
21 if __name__ == "__main__":
22     number = int(input("Enter a number: "))
23     if is_prime(number):
24         print(f"{number} is Prime")
25     else:
26         print(f"{number} is Not Prime")

```

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

```

025.18.0-win32-x64\bundled\libs\debugpy\launcher' '57954' '--' 'c:\java saves\task 2.py'
Enter a number: 7
7 is Prime
PS C:\java saves> 10
10
PS C:\java saves> ^C
PS C:\java saves>
PS C:\java saves> cd 'c:\java saves'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe'
025.18.0-win32-x64\bundled\libs\debugpy\launcher' '54522' '--' 'c:\java saves\task 2.py'
Enter a number: 10
10 is Not Prime
PS C:\java saves>

```

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. At the top, there is a navigation bar with tabs for 'Go', 'Run', 'Terminal', and 'Help'. Below the navigation bar is a search bar labeled 'Search'. The main area contains several tabs: 'DArray.java' (disabled), 'J class Solution{ Untitled-1' (disabled), 'J BalencedBT.java' (disabled), 'fibonacci.py' (disabled), 'factorial.py' (disabled), and 'J JobSe...'. The current tab is 'task 2.py'.

```
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.
```

Below the code, there are tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL', and 'PORTS'. The 'TERMINAL' tab is selected, showing the following output:

```
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 terminal tab active at the bottom. The terminal window displays two sets of command-line interactions. The first interaction, labeled 'Task 1: Inline Logic (No Functions)', shows a user entering '997' and receiving output indicating it's a prime number with an execution time of 0.000000 seconds. The second interaction, labeled 'Task 3: Modular with Functions', also shows a user entering '997' and receiving the same output. The code itself is a Python script named 'task 2.py'.

```
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
```

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 Jupyter Notebook environment with several tabs at the top: Home, Run, Kernel, Help, and three more tabs whose names are partially visible. The main area displays a code cell containing Java code for checking if a number is prime. Below the code cell, the terminal output shows the command `C:\java saves` being run, followed by the Java code execution, and finally the output "Not Prime".

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

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\java saves>
PS C:\java saves> c;; cd 'c:\java saves'; & 'c:\Users\shash\anaconda3\envs\Sh
025.18.0-win32-x64\bundled\libs\debugpy\launcher' '54382' '--' 'c:\java saves'
Enter number: 7
Prime
PS C:\java saves> ^C
PS C:\java saves>
PS C:\java saves> c;; cd 'c:\java saves'; & 'c:\Users\shash\anaconda3\envs\Sh
025.18.0-win32-x64\bundled\libs\debugpy\launcher' '58779' '--' 'c:\java saves'
Enter number: 10
Not Prime
PS C:\java saves>
```

The screenshot shows a code editor interface with several tabs at the top: DArray.java, Solution (Untitled-1), BalencedBT.java, and fibonacci.py. The main area displays the following Python code:

```
C:\> java saves > task 2.py > ...
1 import math
2 def is_prime_optimized(n):
3     if n < 2:
4         return False
5     for i in range(2, int(math.sqrt(n)) + 1): # Up to √n: O(√n)
6         if n % i == 0:
7             return False
8     return True
9
10 # Test
11 n = int(input("Enter number: "))
12 print("Prime" if is_prime_optimized(n) else "Not Prime")
```

Below the code editor is a terminal window with the following output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\java saves>
PS C:\java saves> c:> cd 'c:\java saves'; & 'c:\Users\shash\anaconda3\envs\025.18.0-win32-x64\bundled\libs\debugpy\launcher' '51709' '--' 'c:\java saves>
Enter number: 7
Prime
PS C:\java saves> ^C
PS C:\java saves>
PS C:\java saves> c:> cd 'c:\java saves'; & 'c:\Users\shash\anaconda3\envs\025.18.0-win32-x64\bundled\libs\debugpy\launcher' '51734' '--' 'c:\java saves>
Enter number: 10
Not Prime
PS C:\java saves>
```

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.