

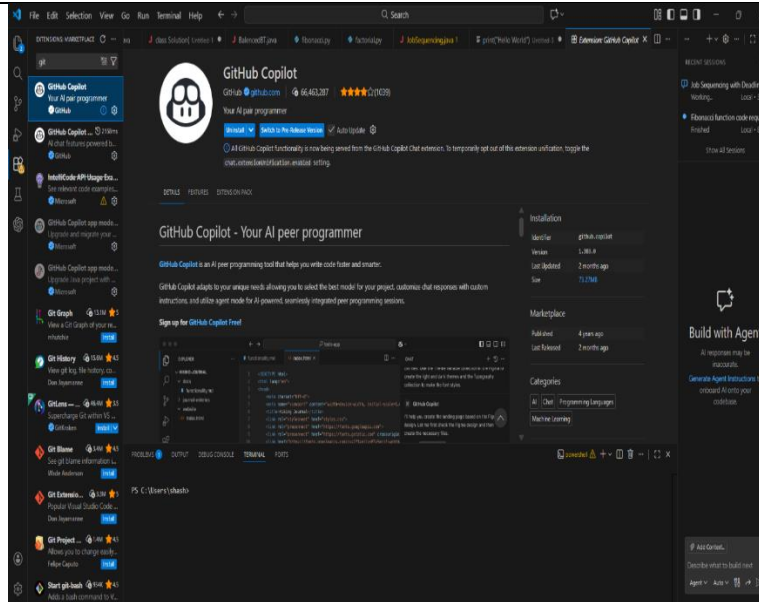
Name:G.Bhagath

H.No:2303A51807

Batch: 26

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech		Assignment Type: Lab	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. Rapelly Nandini	
		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 - Tuesday	Time(s)	23CSBTB01 To 23CSBTB52
Duration	2 Hours	Applicable to Batches	All batches
Assignment Number:1.2(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 + Understanding AI-assisted Coding Workflow</i>	Week1 - Monday	

	<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> <hr/> <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 (Factorial without Functions)

- **Scenario**

You are building a **small command-line utility** for a startup intern onboarding task. The program is simple and must be written quickly without modular design.

- **Task Description**

Use GitHub Copilot to generate a Python program that computes a mathematical product-based value (factorial-like logic) directly in the main execution flow, without using any user-defined functions.

- **Constraint:**

- Do not define any custom function
- Logic must be implemented using loops and variables only

- **Expected Deliverables**

- A working Python program generated with Copilot assistance
- Screenshot(s) showing:
  - The prompt you typed
  - Copilot's suggestions
  - Sample input/output screenshots
  - Brief reflection (5–6 lines):
    - How helpful was Copilot for a beginner?
    - Did it follow best practices automatically?

```
C:\> java saves > task1.py > ...
1 # Simple command-line program to compute factorial of a number n
2 # Use a loop to calculate n! without any functions
3 # Take input from user, print result
4 n = int(input("Enter a number: "))
5 result = 1
6 for i in range(1, n + 1):
7     result *= i
8 print(f"The factorial of {n} is {result}")

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS

path was included, verify that the path is correct and try again.
At line:1 char:1
+ conda activate Shashidhar
+ ~~~~~
+ CategoryInfo          : ObjectNotFound: (conda:String) [], CommandNotFoundException
+ FullyQualifiedErrorId : CommandNotFoundException

PS C:\java saves> & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' 'c:\User
ed\libs\debugpy\launcher' '50660' '--' 'c:\java saves\task1.py'
Enter a number: 5
The factorial of 5 is 120
PS C:\java saves>
```

## Task 2: AI Code Optimization & Cleanup (Improving Efficiency)

### ❖ Scenario

Your team lead asks you to **review AI-generated code** before committing it to a shared repository.

### ❖ Task Description

Analyze the code generated in **Task 1** and use Copilot again to:

- Reduce unnecessary variables
- Improve loop clarity
- Enhance readability and efficiency

Hint:

Prompt Copilot with phrases like

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

### ❖ Expected Deliverables

- Original AI-generated code
- Optimized version of the same code
- Side-by-side comparison
- Written explanation:
  - What was improved?
  - Why the new version is better (readability, performance, maintainability).
  - 
  -

```
> java saves > task1.py > ...
1 # Optimized factorial computation
2 n = int(input("Enter a number: "))
3 fact = 1
4 for num in range(1, n + 1):
5     fact *= num
6 print(f"Factorial of {n}: {fact}")5

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\java saves> & 'c:\Users\shash\anaconda3\envs\Shash
cd\libs\debugpy\launcher' '50660' '--' 'c:\java saves\tas
Enter a number: 5
The factorial of 5 is 120
PS C:\java saves> ^C
PS C:\java saves>
PS C:\java saves> c;; cd 'c:\java saves'; & 'c:\Users\sh
025.18.0-win32-x64\bundled\libs\debugpy\launcher' '49935'
Enter a number: 5
Factorial of 5: 120
PS C:\java saves>
```

### Task 3: Modular Design Using AI Assistance (Factorial with Functions)

#### ❖ Scenario

The same logic now needs to be reused in **multiple scripts**.

#### ❖ Task Description

Use GitHub Copilot to generate a **modular version** of the program by:

- Creating a **user-defined function**
- Calling the function from the main block

#### ❖ Constraints

- Use meaningful function and variable names
- Include inline comments (preferably suggested by Copilot)

#### ❖ Expected Deliverables

- AI-assisted function-based program
- Screenshots showing:
  - Prompt evolution
  - Copilot-generated function logic
- Sample inputs/outputs
- Short note:

- How modularity improves reusability.
- 

```

C:\java saves> task1.py > ...
1 def factorial(n):
2     """Compute factorial of n using iteration."""
3     if n < 0:
4         return None # Handle negative input
5     result = 1
6     for i in range(1, n + 1):
7         result *= i # Multiply incrementally
8     return result
9
10 # Main execution
11 if __name__ == "__main__":
12     n = int(input("Enter a number: "))
13     fact = factorial(n)
14     if fact is not None:
15         print(f"Factorial of {n}: {fact}")
16     else:
17         print("Invalid input: Factorial not defined for negative numbers.")

```

```

PS C:\java saves>
PS C:\java saves> c:; cd 'c:\java saves'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' 'c:\025.18.0-win32-x64\bundle\libs\debugpy\launcher' '57609' '--' 'c:\java saves\task1.py'
Enter a number: 5
Factorial of 5: 120
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:\025.18.0-win32-x64\bundle\libs\debugpy\launcher' '57635' '--' 'c:\java saves\task1.py'
Enter a number: -3
Invalid input: Factorial not defined for negative numbers.
PS C:\java saves>

```

#### Task 4: Comparative Analysis – Procedural vs Modular AI Code (With vs Without Functions)

##### ❖ Scenario

As part of a **code review meeting**, you are asked to justify design choices.

##### ❖ Task Description

Compare the **non-function** and **function-based** Copilot-generated programs on the following criteria:

- Logic clarity
- Reusability
- Debugging ease
- Suitability for large projects
- AI dependency risk

❖ **Expected Deliverables**

Choose **one**:

➤ A comparison table

**OR**

➤ A short technical report (300–400 words).

```
View Go Run Terminal Help ← → 🔍 Search
... DArray.java class Solution[ Untitled-1 ] BalancedBT.java fibonacci.py factorial.py

C:\> java saves > task1.py > ...
1 # Optimized procedural factorial computation (no functions)
2 # Computes factorial inline for quick utility
3
4 n = int(input("Enter a number: "))
5
6 # Check for invalid input
7 if n < 0:
8     print("Invalid input: Factorial not defined for negative numbers.")
9 else:
10     fact = 1
11     for num in range(1, n + 1):
12         fact *= num # Multiply incrementally
13     print(f"Factorial of {n}: {fact}")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

--- 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> ^C
PS C:\java saves>
PS C:\java saves> c:; cd 'c:\java saves'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\py
025.18.0-win32-x64\bundled\libs\debugpy\launcher' '63490' '-' 'c:\java saves\task1.py'
Enter a number: 5
Factorial of 5: 120
PS C:\java saves> |
```

```
DArray.java  J class Solution{ Untitled-1  J BalancedBT.java  fibonacci.py  factorial.py

C:\> java saves > task1.py > ...
1  # Modular factorial program using a function for reusability
2
3  def factorial(n):
4      """
5      Compute factorial of n using iteration.
6      Handles negative inputs gracefully.
7      """
8      if n < 0:
9          return None # Handle negative input
10     result = 1
11     for i in range(1, n + 1):
12         result *= i # Multiply incrementally
13     return result
14
15 # Main execution block
16 if __name__ == "__main__":
17     n = int(input("Enter a number: "))
18     fact = factorial(n)
19     if fact is not None:
20         print(f"Factorial of {n}: {fact}")
21     else:
22         print("Invalid input: Factorial not defined for negative numbers.")

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\java saves>
PS C:\java saves> c;; cd 'c:\java saves'; & 'c:\Users\shash\anaconda3\envs\Shashidh\025.18.0-win32-x64\bundle\libs\debugpy\launcher' '63490' '--' 'c:\java saves\task1.py'
Enter a number: 5
Factorial of 5: 120
PS C:\java saves> ^C
PS C:\java saves>
PS C:\java saves> c;; cd 'c:\java saves'; & 'c:\Users\shash\anaconda3\envs\Shashidh\025.18.0-win32-x64\bundle\libs\debugpy\launcher' '63554' '--' 'c:\java saves\task1.py'
Enter a number: 5
Factorial of 5: 120
PS C:\java saves>
```

## Task 5: AI-Generated Iterative vs Recursive Thinking

### ❖ Scenario

Your mentor wants to test how well AI understands different computational paradigms.

### ❖ Task Description

Prompt Copilot to generate:

An **iterative** version of the logic

A **recursive** version of the same logic

### ❖ Constraints

Both implementations must produce identical outputs

Students must **not manually write the code first**

### ❖ Expected Deliverables

Two AI-generated implementations

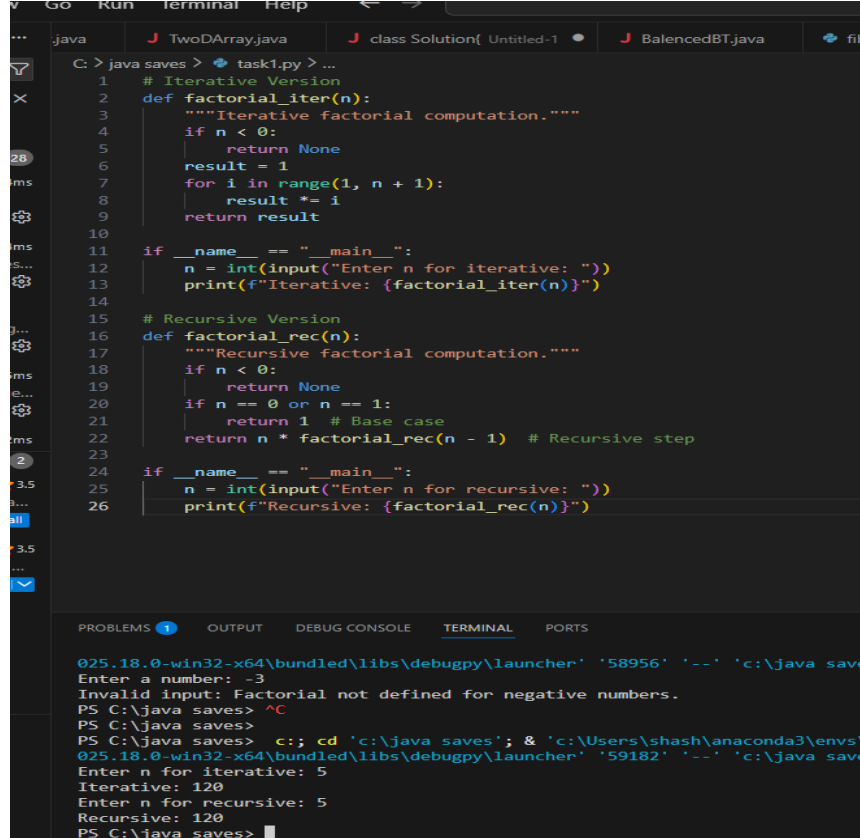
Execution flow explanation (in your own words)

Comparison covering:

- Readability
- Stack usage
- Performance implications



➤ When recursion is *not* recommended.



```
1 # Iterative Version
2 def factorial_iter(n):
3     """Iterative factorial computation."""
4     if n < 0:
5         return None
6     result = 1
7     for i in range(1, n + 1):
8         result *= i
9     return result
10
11 if __name__ == "__main__":
12     n = int(input("Enter n for iterative: "))
13     print(f"Iterative: {factorial_iter(n)}")
14
15 # Recursive Version
16 def factorial_rec(n):
17     """Recursive factorial computation."""
18     if n < 0:
19         return None
20     if n == 0 or n == 1:
21         return 1 # Base case
22     return n * factorial_rec(n - 1) # Recursive step
23
24 if __name__ == "__main__":
25     n = int(input("Enter n for recursive: "))
26     print(f"Recursive: {factorial_rec(n)}")
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
025.18.0-win32-x64\bundled\libs\debugpy\launcher' '58956' '--' 'c:\java save
Enter a number: -3
Invalid input: Factorial not defined for negative numbers.
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' '59182' '--' 'c:\java save
Enter n for iterative: 5
Iterative: 120
Enter n for recursive: 5
Recursive: 120
PS C:\java saves>
```

### Submission Requirements

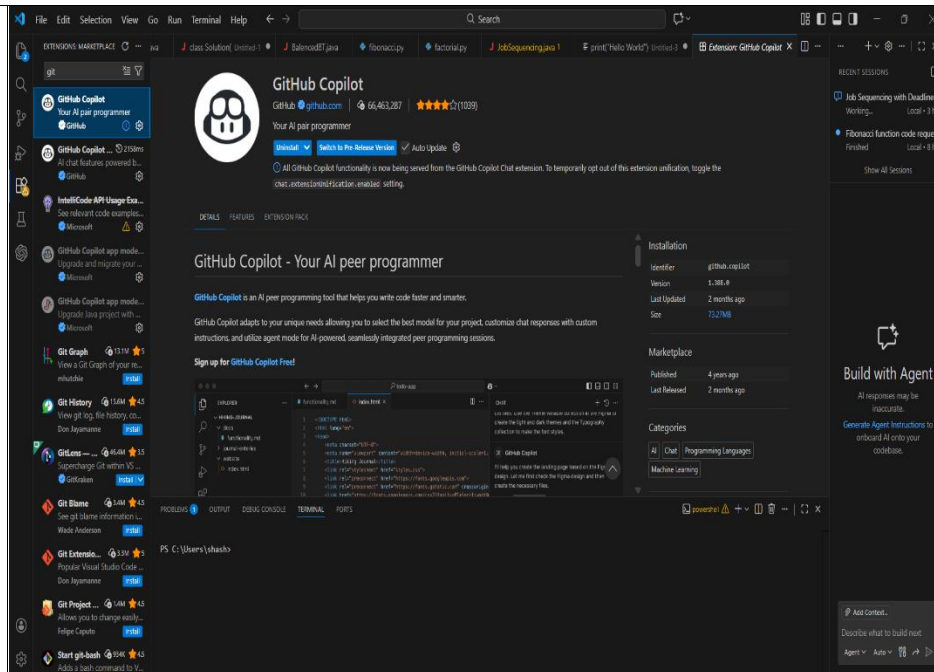
1. Generate code for each task with comments.
2. Screenshots of Copilot suggestions.
3. Comparative analysis reports (Task 4 and Task 5).
4. Sample inputs/outputs demonstrating correctness.

**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.

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	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 + Understanding AI-assisted Coding Workflow</i>	Week1 - Monday	

	<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> <hr/> <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

The screenshot shows a VS Code editor with a Python file named `task2.py`. The code is a prime number checker. It prompts the user to enter a number. If the number is less than 2, it prints "Not Prime". Otherwise, it assumes the number is prime and checks for divisibility from 2 to `number-1`. If a divisor is found, it prints "Not Prime"; otherwise, it prints "Prime".

```
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")
```

The terminal at the bottom shows the execution of the script. It runs `python task2.py` and shows the output for two inputs: 7 (Prime) and 10 (Not Prime).

```
PS C:\java saves> python task2.py
Enter a number: 7
Prime
PS C:\java saves> ^C
PS C:\java saves> python task2.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

```

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")

```

```

PS C:\java saves>
PS C:\java saves> c; cd 'c:\java saves'; & 'c:\Users\shash\anaconda3\envs\Shashid
025.18.0-win32-x64\bundled\libs\debugpy\launcher' '61305' '--' 'c:\java saves\task
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\Shashid
025.18.0-win32-x64\bundled\libs\debugpy\launcher' '61339' '--' 'c:\java saves\task
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

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

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")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

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> c;; 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

FileRunTerminalHelp←→Search

DArray.javaJ class Solution[Untitled-1]BalancedBT.javafibonacci.pyfactorial.pyJobS

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.

PROBLEMSOUTPUTDEBUG CONSOLETERMINALPORTS

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
```

```
Run Terminal Help ← → Q Search
Array.java J class Solution[ Untitled-1 J BalancedBT.java fibonacci.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**.

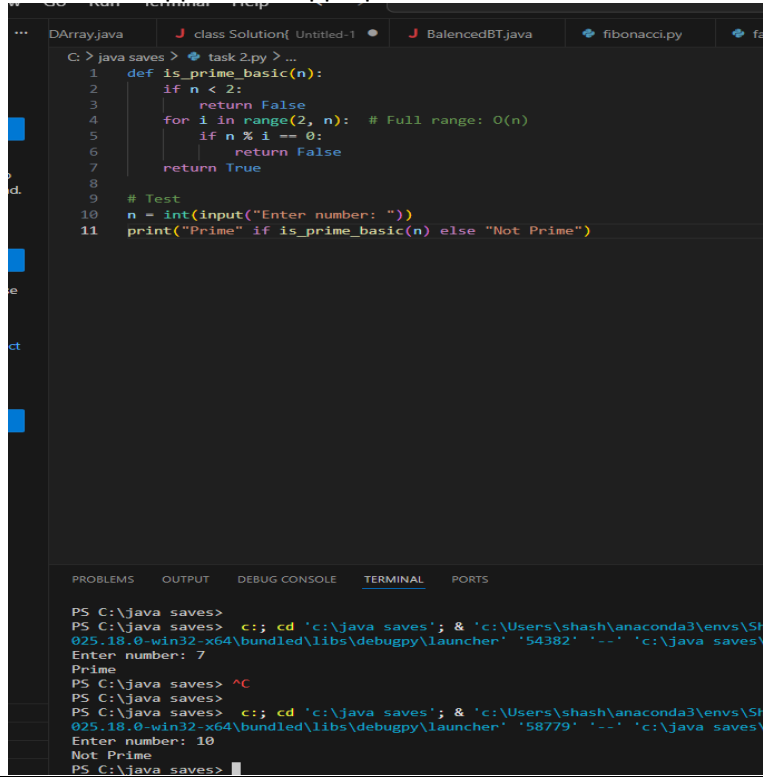
### ❖ Task Description

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



```
DArray.java  class Solution{  Untitled-1  BalancedBT.java  fibonacci.py  fa

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>
```

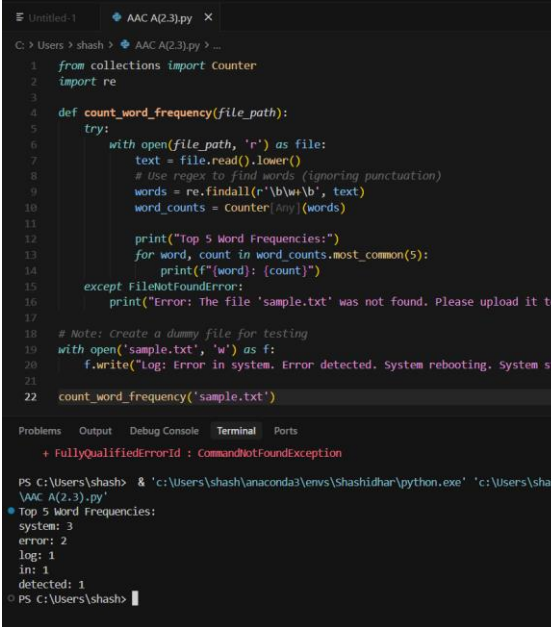
The screenshot shows a code editor with a Python script named `task 2.py` and a terminal window below it. The script defines a function `is_prime_optimized(n)` that checks if a number is prime by testing divisibility up to its square root. It includes a test section where the user is prompted to enter a number. The terminal shows the script being executed twice: first with input 7, which is prime, and then with input 10, which is not prime.

```
DArray.java | J class Solution{ Untitled-1 | J BalancedBT.java | fibonacci.py |  
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 sqrt(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")  
  
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 save  
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 save  
Enter number: 10  
Not Prime  
PS C:\java saves> '
```

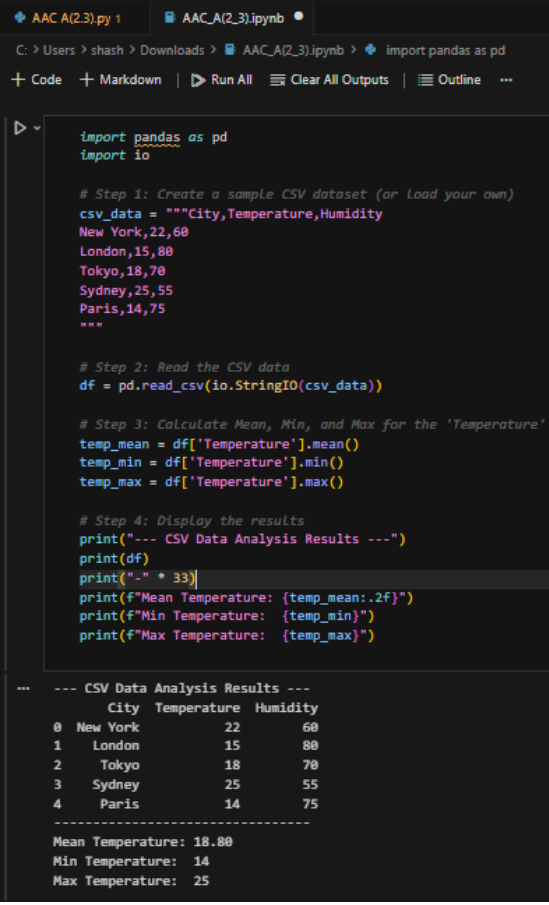
**Note:** Report should be submitted as a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots.

NAME:G.Bhagath H.NO:2303A51807 BATCH:26

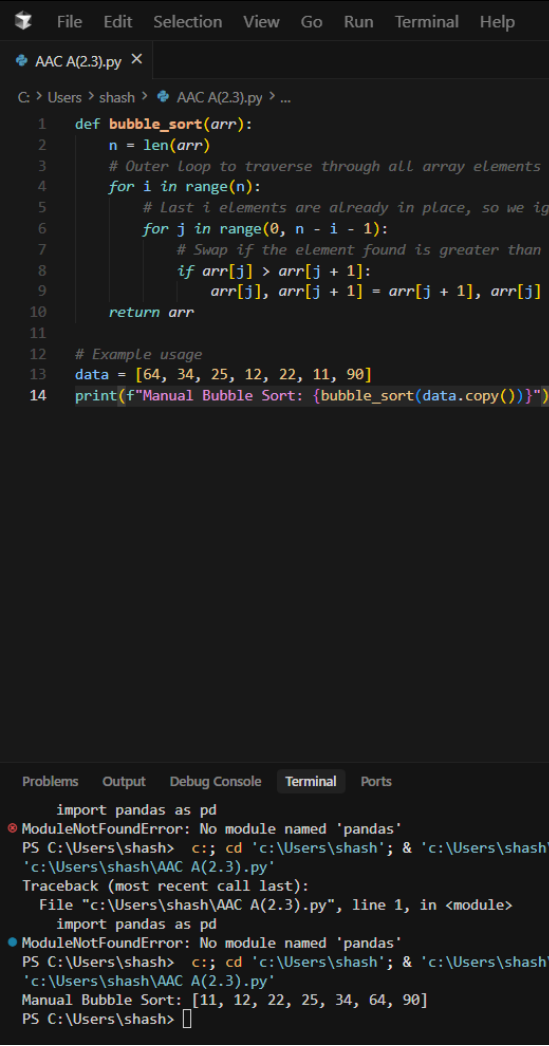
SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE				DEPART	
Program Name: B. Tech				Assignment Type: Lab	
Course Coordinator Name				Dr. Rishabh Mittal	
Instructor(s) Name					
				Mr. S Naresh Kumar	
				Ms. B. Swathi	
				Dr. Sasanko Shekhar Gantaya	
				Mr. Md Sallauddin	
				Dr. Mathivanan	
				Mr. Y Srikanth	
				Ms. N Shilpa	
				Dr. Rishabh Mittal (Coordinat	
				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
Year/Sem	III/II			Regulation	R23
Date and Day of Assignment	Week1 – Wednesday			Time(s)	23CSBTB0
Duration	2 Hours			Applicable to Batches	All batches
			Assignment Number:1.3(Present assignment number)/24(Total n		
Q.No.		Question			
1		Lab 2: Exploring Additional AI Coding Tools beyond C and Cursor AI  <b>Lab Objectives:</b>			

		<ul style="list-style-type: none"><li>❖ To explore and evaluate the functionality of AI-assisted coding within Google Colab.</li><li>❖ To understand and use Cursor AI for code generation and refactoring.</li><li>❖ To compare outputs and usability between Gemini and Cursor AI.</li><li>❖ To perform code optimization and documentation.</li></ul> <p><b>Lab Outcomes (LOs):</b></p> <p>After completing this lab, students will be able to:</p> <ul style="list-style-type: none"><li>❖ Generate Python code using Google Gemini in a Jupyter Notebook.</li><li>❖ Analyze the effectiveness of code explanations generated by Gemini.</li><li>❖ Set up and use Cursor AI for AI-powered coding.</li><li>❖ Evaluate and refactor code using Cursor AI features.</li><li>❖ Compare AI tool behavior and code quality across different scenarios.</li></ul> <hr/> <p><b>Task 1: Word Frequency from Text File</b></p> <ul style="list-style-type: none"><li>❖ <b>Scenario:</b> You are analyzing log files for keyword frequency.</li><li>❖ <b>Task:</b> Use Gemini to generate Python code that reads a text file, calculates word frequency, then explains the code.</li><li>❖ <b>Expected Output:</b><ul style="list-style-type: none"><li>➤ Working code</li><li>➤ Explanation</li><li>➤ Screenshot</li></ul></li></ul>  <pre>from collections import Counter import re  def count_word_frequency(file_path):     try:         with open(file_path, 'r') as file:             text = file.read().lower()             # Use regex to find words (ignoring punctuation)             words = re.findall(r'\b\w+\b', text)             word_counts = Counter(words)             print("Top 5 Word Frequencies:")             for word, count in word_counts.most_common(5):                 print(f"{word}: {count}")     except FileNotFoundError:         print("Error: The file 'sample.txt' was not found. Please upload it to the workspace.")  # Note: Create a dummy file for testing with open('sample.txt', 'w') as f:     f.write("Log: Error in system. Error detected. System rebooting. System started.")  count_word_frequency('sample.txt')</pre> <p>PS C:\Users\shash&gt; &amp; 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' 'c:\Users\shash\VAAC A(2.3).py'</p> <p>VAAC A(2.3).py</p> <p>Top 5 Word Frequencies:</p> <p>system: 3</p> <p>error: 2</p> <p>log: 1</p> <p>in: 1</p> <p>detected: 1</p> <p>PS C:\Users\shash&gt;</p>
--	--	---

		<div><div>Task 2: File Operations Using Cursor AI</div><div><div>❖ Scenario:</div><div>You are automating basic file operations.</div></div><div><div>❖ Task:</div><div>Use Cursor AI to generate a program that:<div><div>➤ Creates a text file</div><div>➤ Writes sample text</div><div>➤ Reads and displays the content</div></div></div></div><div><div>❖ Expected Output:</div><div><div>➤ Functional code</div><div>➤ Cursor AI screenshots</div></div></div></div>
		<div><div>Task 3: CSV Data Analysis</div><div><div>❖ Scenario:</div><div>You are processing structured data from a CSV file</div></div><div><div>❖ Task:</div><div>Use Gemini in Colab to read a CSV file and calculat</div></div></div>

		<div><div><div>❖ Expected Output:</div><div><div>➤ Correct output</div><div>➤ Screenshot</div></div></div><div><pre>import pandas as pd import io  # Step 1: Create a sample CSV dataset (or Load your own) csv_data = """City,Temperature,Humidity New York,22,60 London,15,80 Tokyo,18,70 Sydney,25,55 Paris,14,75 """  # Step 2: Read the CSV data df = pd.read_csv(io.StringIO(csv_data))  # Step 3: Calculate Mean, Min, and Max for the 'Temperature' temp_mean = df['Temperature'].mean() temp_min = df['Temperature'].min() temp_max = df['Temperature'].max()  # Step 4: Display the results print("--- CSV Data Analysis Results ---") print(df) print("\n * 33]") print(f"Mean Temperature: {temp_mean:.2f}") print(f"Min Temperature: {temp_min}") print(f"Max Temperature: {temp_max}")  --- --- CSV Data Analysis Results ---    City  Temperature  Humidity 0  New York         22        60 1   London         15        80 2   Tokyo          18        70 3  Sydney          25        55 4   Paris          14        75 ----- Mean Temperature: 18.80 Min Temperature: 14 Max Temperature: 25</pre></div><div><div>➤</div></div></div> <div><div>Task 4: Sorting Lists – Manual vs Built-in</div><div><div>❖ Scenario:</div><div>You are reviewing algorithm choices for efficiency.</div></div><div><div>❖ Task:</div><div>Use <b>Gemini</b> to generate:<div><div>➤ Bubble sort</div><div>➤ Python's built-in sort()</div><div>➤ Compare both implementations.</div></div></div></div><div><div>❖ Expected Output:</div><div><div>➤ Two versions of code</div><div>➤ Short comparison</div></div></div></div>
--	--	---



		 <p>The screenshot displays a code editor with a Python file named <code>AAC A(2.3).py</code>. The code implements a bubble sort function and includes example usage. Below the code, a terminal window shows the execution of the script, which results in a <code>ModuleNotFoundError: No module named 'pandas'</code> error and the output of the bubble sort function.</p> <pre>def bubble_sort(arr):     n = len(arr)     # Outer Loop to traverse through all array elements     for i in range(n):         # Last i elements are already in place, so we ignore         # them         for j in range(0, n - i - 1):             # Swap if the element found is greater than             # the next element             if arr[j] &gt; arr[j + 1]:                 arr[j], arr[j + 1] = arr[j + 1], arr[j]     return arr  # Example usage data = [64, 34, 25, 12, 22, 11, 90] print(f"Manual Bubble Sort: {bubble_sort(data.copy())}")</pre> <p>Terminal Output:</p> <pre>import pandas as pd ModuleNotFoundError: No module named 'pandas' PS C:\Users\shash&gt; c:; cd 'c:\Users\shash'; &amp; 'c:\Users\shash\AAC A(2.3).py' Traceback (most recent call last):   File "c:\Users\shash\AAC A(2.3).py", line 1, in &lt;module&gt;     import pandas as pd ModuleNotFoundError: No module named 'pandas' PS C:\Users\shash&gt; c:; cd 'c:\Users\shash'; &amp; 'c:\Users\shash\AAC A(2.3).py' Manual Bubble Sort: [11, 12, 22, 25, 34, 64, 90] PS C:\Users\shash&gt;</pre>
--	--	--

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

NAME:G.BHAGATH

H.NO:2303A51807

BATCH:26

<b>SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE</b>		<b>DEPARTMENT OF COMPUTER SCIENCE ENGINEERING</b>																		
<b>Program Name:</b> B. Tech		<b>Assignment Type:</b> Lab	<b>Academic Year:</b> 2025-2026																	
<b>Course Coordinator Name</b>		Dr. Rishabh Mittal																		
<b>Instructor(s) Name</b>		<table border="1"> <tr><td>Mr. S Naresh Kumar</td></tr> <tr><td>Ms. B. Swathi</td></tr> <tr><td>Dr. Sasanko Shekhar Gantayat</td></tr> <tr><td>Mr. Md Sallauddin</td></tr> <tr><td>Dr. Mathivanan</td></tr> <tr><td>Mr. Y Srikanth</td></tr> <tr><td>Ms. N Shilpa</td></tr> <tr><td>Dr. Rishabh Mittal (Coordinator)</td></tr> <tr><td>Dr. R. Prashant Kumar</td></tr> <tr><td>Mr. Ankushavali MD</td></tr> <tr><td>Mr. B Viswanath</td></tr> <tr><td>Ms. Sujitha Reddy</td></tr> <tr><td>Ms. A. Anitha</td></tr> <tr><td>Ms. M.Madhuri</td></tr> <tr><td>Ms. Katherashala Swetha</td></tr> <tr><td>Ms. Velpula sumalatha</td></tr> <tr><td>Mr. Bingi Raju</td></tr> </table>		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
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																				
<b>CourseCode</b>	23CS002PC304	<b>Course Title</b>	AI Assisted Coding																	
<b>Year/Sem</b>	III/II	<b>Regulation</b>	R23																	
<b>Date and Day of Assignment</b>	Week2	<b>Time(s)</b>	23CSBTB01 To 23CSBTB52																	
<b>Duration</b>	2 Hours	<b>Applicable to Batches</b>	All batches																	
<b>Assignment Number: 3.4</b> (Present assignment number)/ <b>24</b> (Total number of assignments)																				
<b>Q.No.</b>	<b>Question</b>	<b>Expected Time to complete</b>																		
1	Lab 4: Advanced Prompt Engineering – Zero-shot, One-shot, and Few-shot Techniques	Week2																		

	<p><b>Task 1: Zero-shot Prompt – Fibonacci Series Generator</b></p> <p><b>Task Description #1</b></p> <ul style="list-style-type: none"><li>• Without giving an example, write a single comment prompt asking GitHub Copilot to generate a Python function to print the first N Fibonacci numbers.</li></ul> <p><b>Expected Output #1</b></p> <ul style="list-style-type: none"><li>• A complete Python function generated by Copilot without any example provided.</li><li>• Correct output for sample input N = 7 → 0 1 1 2 3 5 8</li><li>• Observation on how Copilot understood the instruction with zero context.</li></ul>	
--	--	--

```

C: > Users > shash > AAC A(3.4).py > ...
1  def print_fibonacci(n):
2      if n <= 0:
3          return
4      a, b = 0, 1
5      print(a, end=" ")
6      if n > 1:
7          print(b, end=" ")
8      for i in range(2, n):
9          a, b = b, a + b
10         print(b, end=" ")
11     print()
12
13     # Test with input N = 7
14     print_fibonacci(7)

```

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\shash> c;; cd 'c:\Users\shash'; & 'c:\Users\shash\anaconda3\envs\Sha
Customer Charges (CC): $50.00
Electricity Duty (ED): $16.00
-----
TOTAL BILL AMOUNT: $326.00
=====
PS C:\Users\shash> c;; cd 'c:\Users\shash'; & 'c:\Users\shash\anaconda3\envs\Sha
PS C:\Users\shash> 7
7
PS C:\Users\shash> c;; cd 'c:\Users\shash'; & 'c:\Users\shash\anaconda3\envs\Sha
0 1 1 2 3 5 8
PS C:\Users\shash> 

```

## Task 2: One-shot Prompt – List Reversal Function

### Task Description #2

- Write a comment prompt to reverse a list and provide one example below the comment to guide Copilot.

### Expected Output #2

- Copilot-generated function to reverse a list using slicing or loop.
- Output: [3, 2, 1] for input [1, 2, 3]
- Observation on how adding a single example improved Copilot's

accuracy.

```
C: > Users > shash > AAC A(3.4).py > ...
1  def reverse_list(lst):
2      return lst[::-1]
3
4  # Test with input [1, 2, 3]
5  result = reverse_list([1, 2, 3])
6  print(result)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\shash> c::; cd 'c:\Users\shash'; & 'c:\Users\shash\anaconda3\er
-----
TOTAL BILL AMOUNT: $326.00
=====
● PS C:\Users\shash> c::; cd 'c:\Users\shash'; & 'c:\Users\shash\anaconda3\er
● PS C:\Users\shash> 7
7
● PS C:\Users\shash> c::; cd 'c:\Users\shash'; & 'c:\Users\shash\anaconda3\er
0 1 1 2 3 5 8
● PS C:\Users\shash> c::; cd 'c:\Users\shash'; & 'c:\Users\shash\anaconda3\er
[3, 2, 1]
○ PS C:\Users\shash> █
```

### Task 3: Few-shot Prompt – String Pattern Matching

#### Task Description #3

- Write a comment with 2–3 examples to help Copilot understand how to check if a string starts with a capital letter and ends with a period.

#### Expected Output #3

- A function is `_valid()` that checks the pattern.
- Output: True or False based on input.

- Students reflect on how multiple examples guide Copilot to generate more accurate code.

```

C:\Users\shash > AAC A(3,4).py > ...
1  def is_valid(s):
2      if not s: # Empty string
3          return False
4      return s[0].isupper() and s[-1] == '.'
5
6  # Test inputs
7  print(is_valid("Hello.")) # True
8  print(is_valid("hello.")) # False
9  print(is_valid("Hello")) # False

```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

```

PS C:\Users\shash> c:: cd 'c:\Users\shash'; & 'c:\Users\shash\anaconda3\envs\Shashash\python.exe' AAC A(3,4).py
PS C:\Users\shash> 7
PS C:\Users\shash> c:: cd 'c:\Users\shash'; & 'c:\Users\shash\anaconda3\envs\Shashash\python.exe' AAC A(3,4).py
0 1 1 2 3 5 8
PS C:\Users\shash> c:: cd 'c:\Users\shash'; & 'c:\Users\shash\anaconda3\envs\Shashash\python.exe' AAC A(3,4).py
[3, 2, 1]
PS C:\Users\shash> c:: cd 'c:\Users\shash'; & 'c:\Users\shash\anaconda3\envs\Shashash\python.exe' AAC A(3,4).py
True
False
False
PS C:\Users\shash>

```

#### Task 4: Zero-shot vs Few-shot – Email Validator

##### Task Description #4

- First, prompt Copilot to write an email validation function using zero-shot (just the task in comment).
- Then, rewrite the prompt using few-shot examples.

##### Expected Output #4

- Compare both outputs:

Zero-shot may result in basic or generic validation.

Few-shot gives detailed and specific logic (e.g., @ and domain checking).

- Submit both code versions and note how few-shot improves

reliability.

```
C: > Users > shash > AAC A(3.4).py > ...
1  import re
2
3  def validate_email(email):
4      pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
5      return bool(re.match(pattern, email))
6
7  # Test inputs
8  print(validate_email("user@example.com")) # True
9  print(validate_email("user@"))           # False
10 print(validate_email("user.example.com")) # False
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\shash> c:; cd 'c:\Users\shash'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.e
PS C:\Users\shash> c:; cd 'c:\Users\shash'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.e
[3, 2, 1]
PS C:\Users\shash> c:; cd 'c:\Users\shash'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.e
True
False
False
PS C:\Users\shash> c:; cd 'c:\Users\shash'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.e
True
False
False
PS C:\Users\shash>
```

## Task 5: Prompt Tuning – Summing Digits of a Number

### Task Description #5

- Experiment with 2 different prompt styles to generate a function that returns the sum of digits of a number.

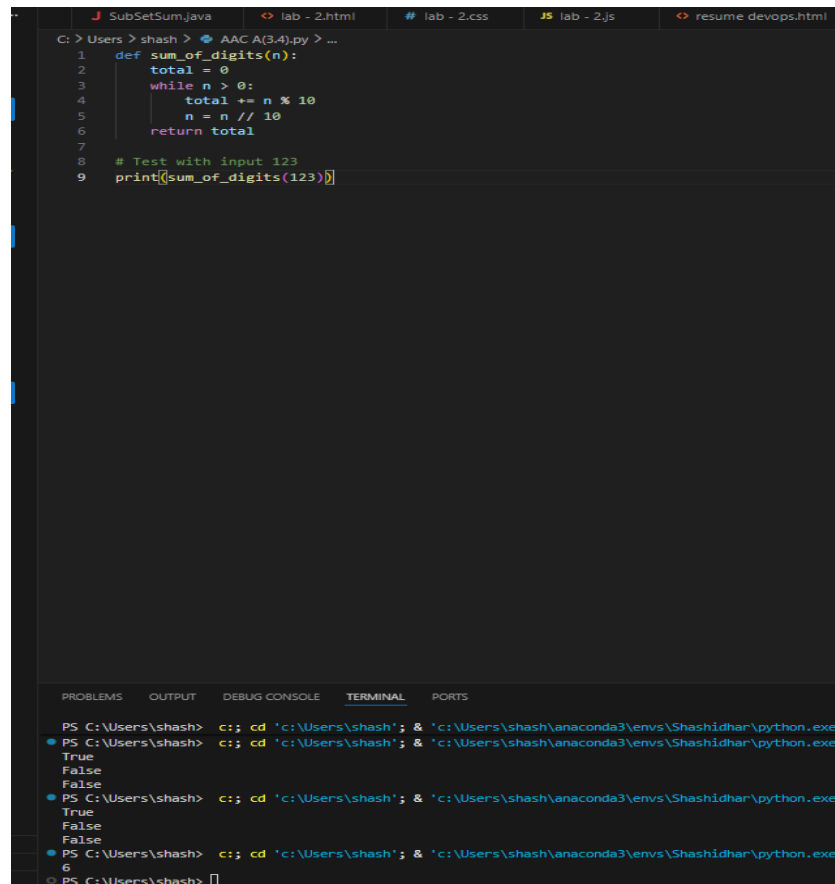
Style 1: Generic task prompt

Style 2: Task + Input/Output example

### Expected Output #5

- Two versions of the `sum_of_digits()` function.
- Example Output: `sum_of_digits(123) → 6`
- Short analysis: which prompt produced cleaner or more

optimized code and why?



```
C: > Users > shash > AAC A(3.4).py > ...
1 def sum_of_digits(n):
2     total = 0
3     while n > 0:
4         total += n % 10
5         n = n // 10
6     return total
7
8 # Test with input 123
9 print(sum_of_digits(123))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\shash> c:; cd 'c:\Users\shash'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe'
PS C:\Users\shash> c:; cd 'c:\Users\shash'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe'
True
False
False
PS C:\Users\shash> c:; cd 'c:\Users\shash'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe'
True
False
False
PS C:\Users\shash> c:; cd 'c:\Users\shash'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe'
6
PS C:\Users\shash>
```

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