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 – *GitHub Copilot and VS Code Integration + Understanding AI-assisted Coding Workflow*	Week1 - Monday

Lab Objectives:

- 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

Lab Outcomes (LOs):
After completing this lab, students will be able to:

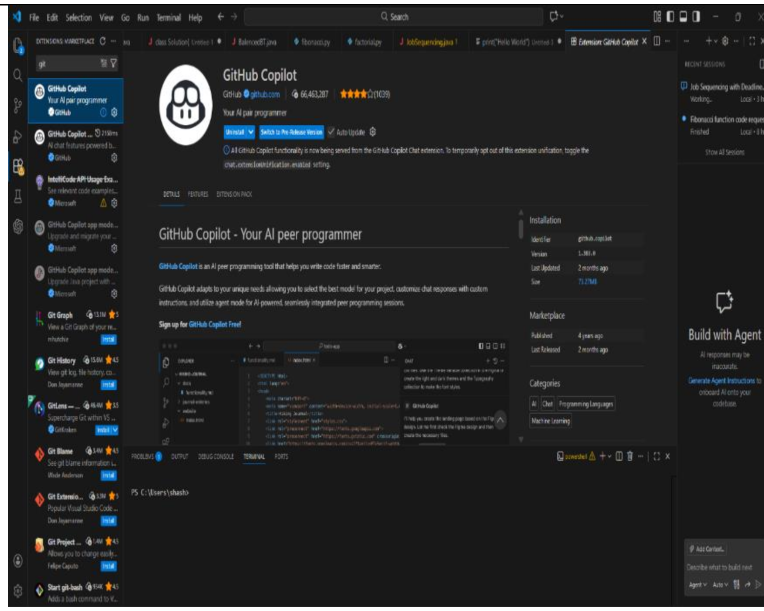
- Set up GitHub Copilot in VS Code successfully.
- Use inline comments and context to generate code with Copilot.
- Evaluate AI-generated code for correctness and readability.
- Compare code suggestions based on different prompts and programming styles.

Task 0

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

Expected Output

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

- 

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**

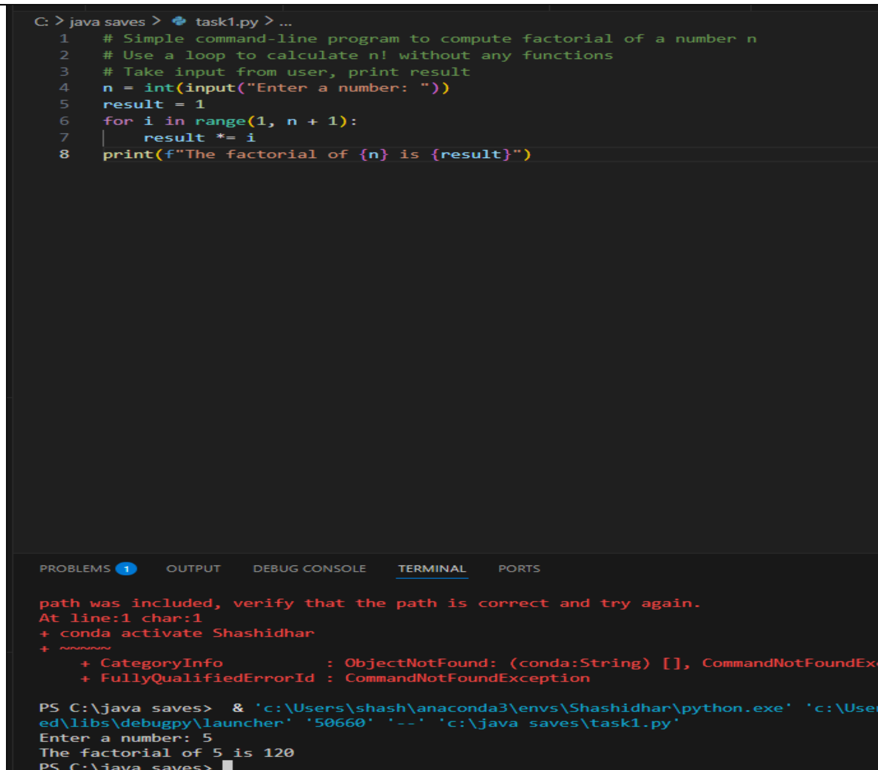
Use GitHub Copilot to generate a Python program that computes a mathematical product-based value (factorial-like logic) directly in the main execution flow, without using any user-defined functions.

- **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?



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

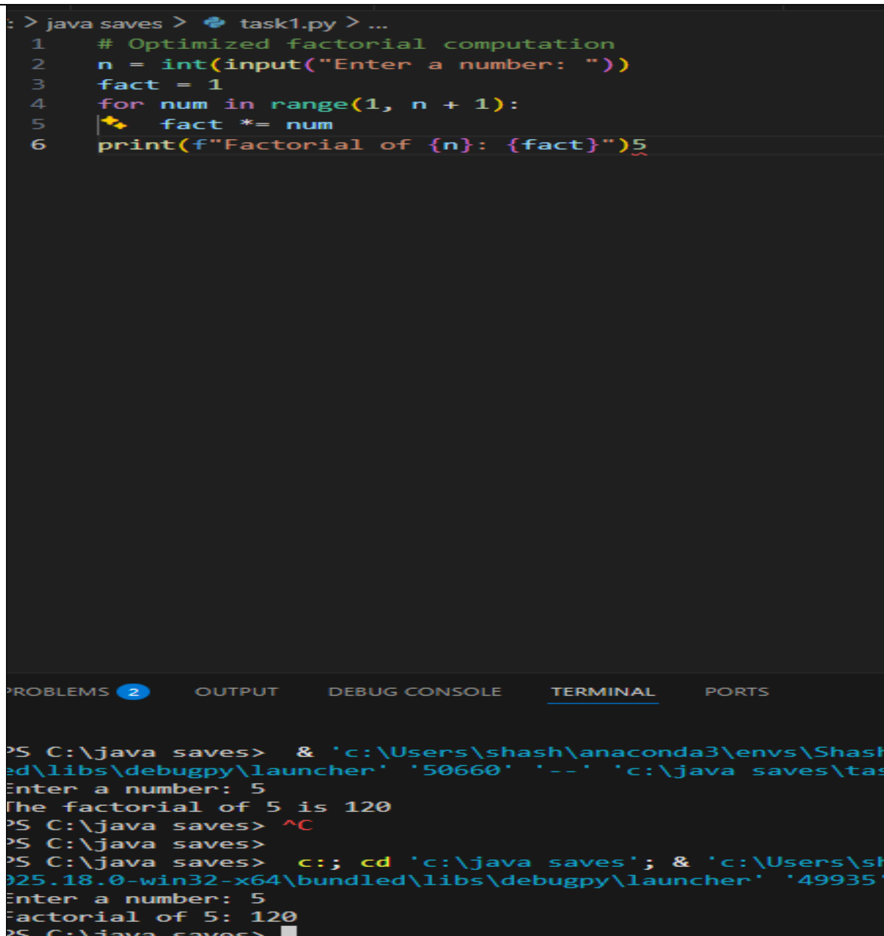
Hint:

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.



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

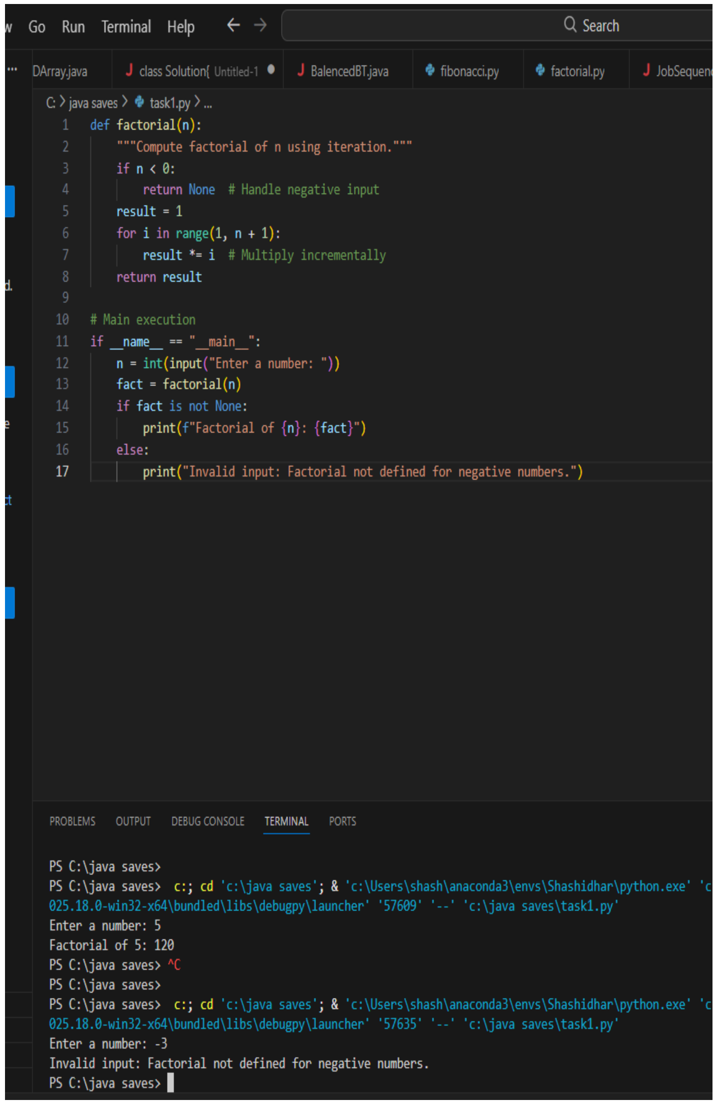
❖ **Constraints**

- 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.
-



-

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

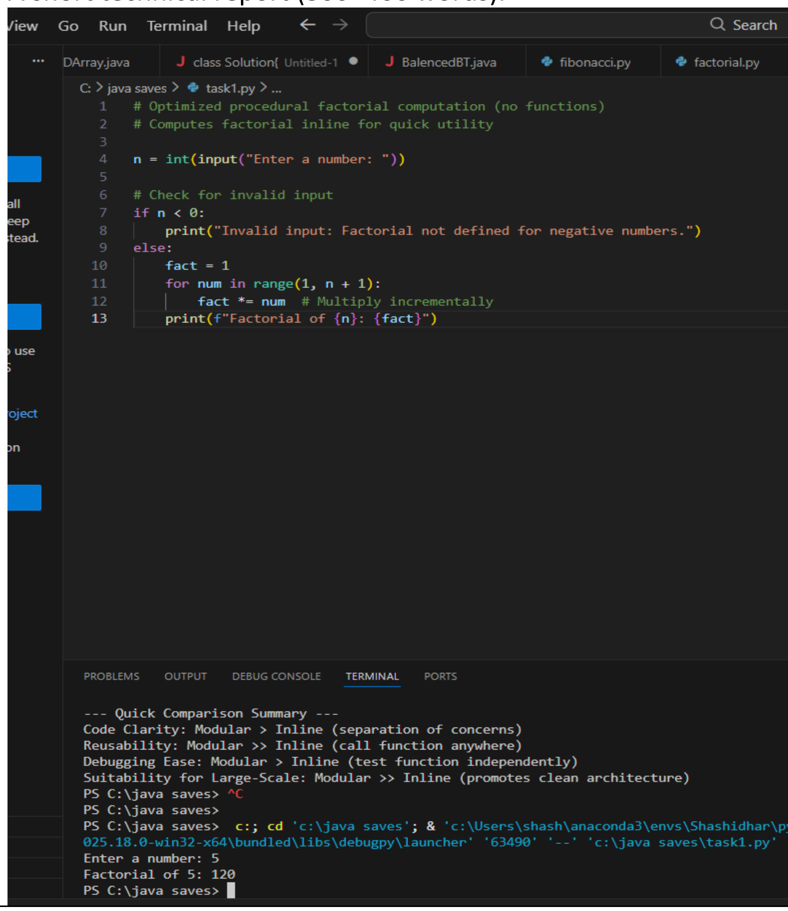
❖ **Expected Deliverables**

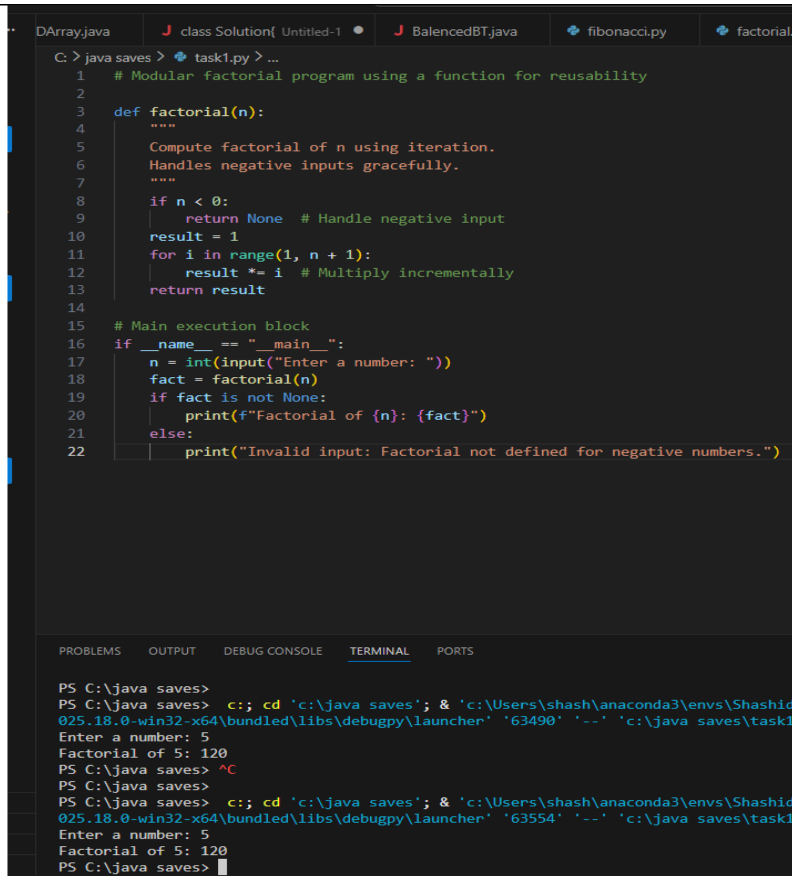
Choose **one**:

➢ A comparison table

OR

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





Task 5: AI-Generated Iterative vs Recursive Thinking

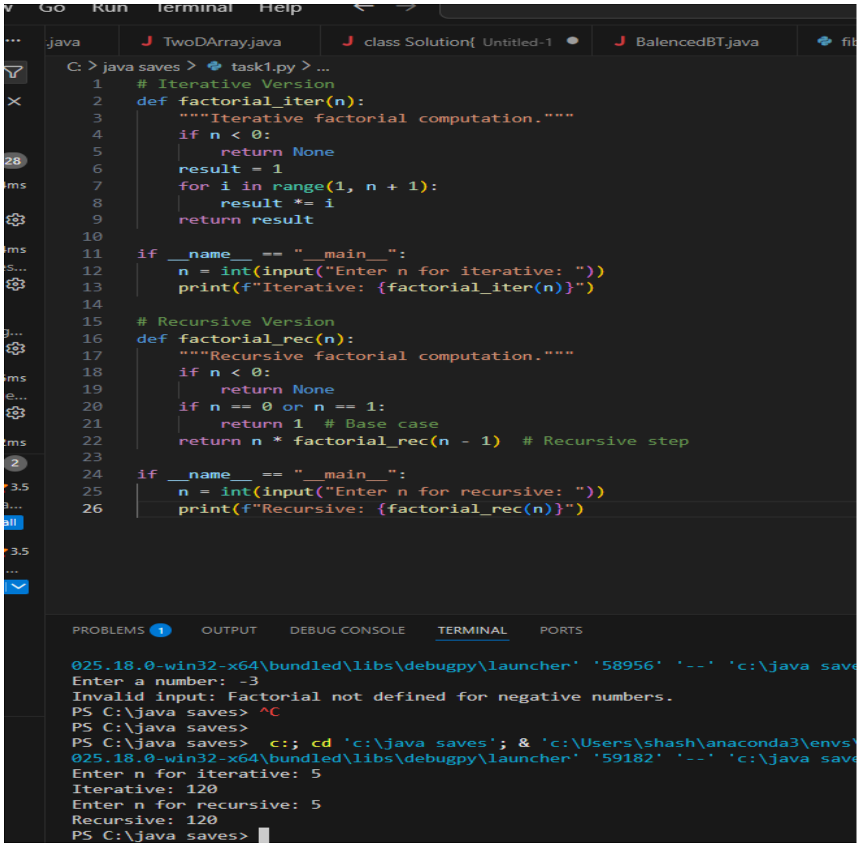
- **Scenario**
  Your mentor wants to test how well AI understands different computational paradigms.

- **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.
-



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 – *GitHub Copilot and VS Code Integration + Understanding AI-assisted Coding Workflow*	Week1 - Monday

Lab Objectives:

- 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

Lab Outcomes (LOs):
After completing this lab, students will be able to:

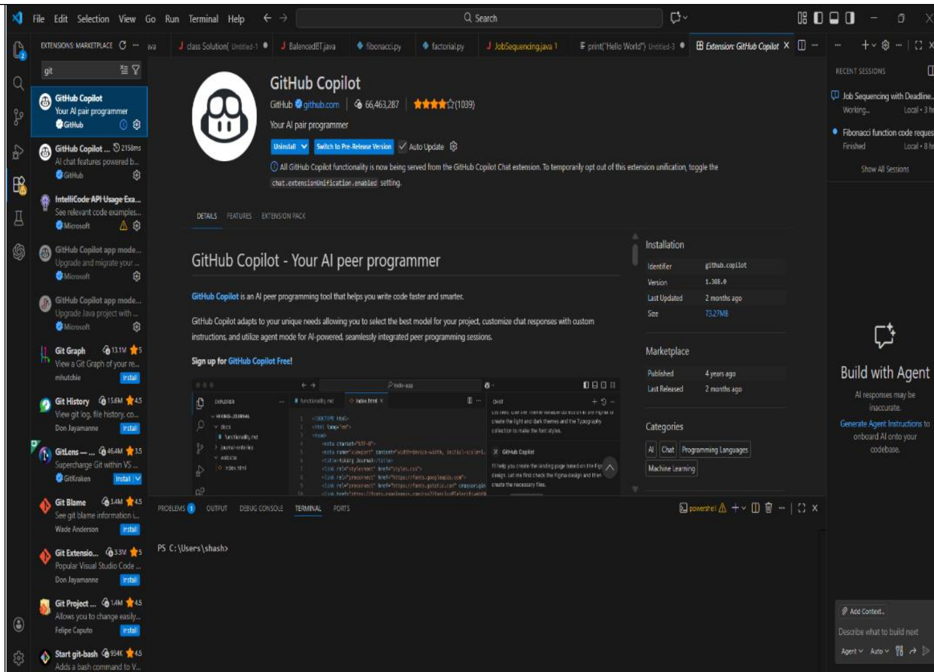
- Set up GitHub Copilot in VS Code successfully.
- Use inline comments and context to generate code with Copilot.
- Evaluate AI-generated code for correctness and readability.
- Compare code suggestions based on different prompts and programming styles.

Task 0

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

Expected Output

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



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

❖ **Scenario**

- You are developing a **basic validation script** for a numerical learning application.

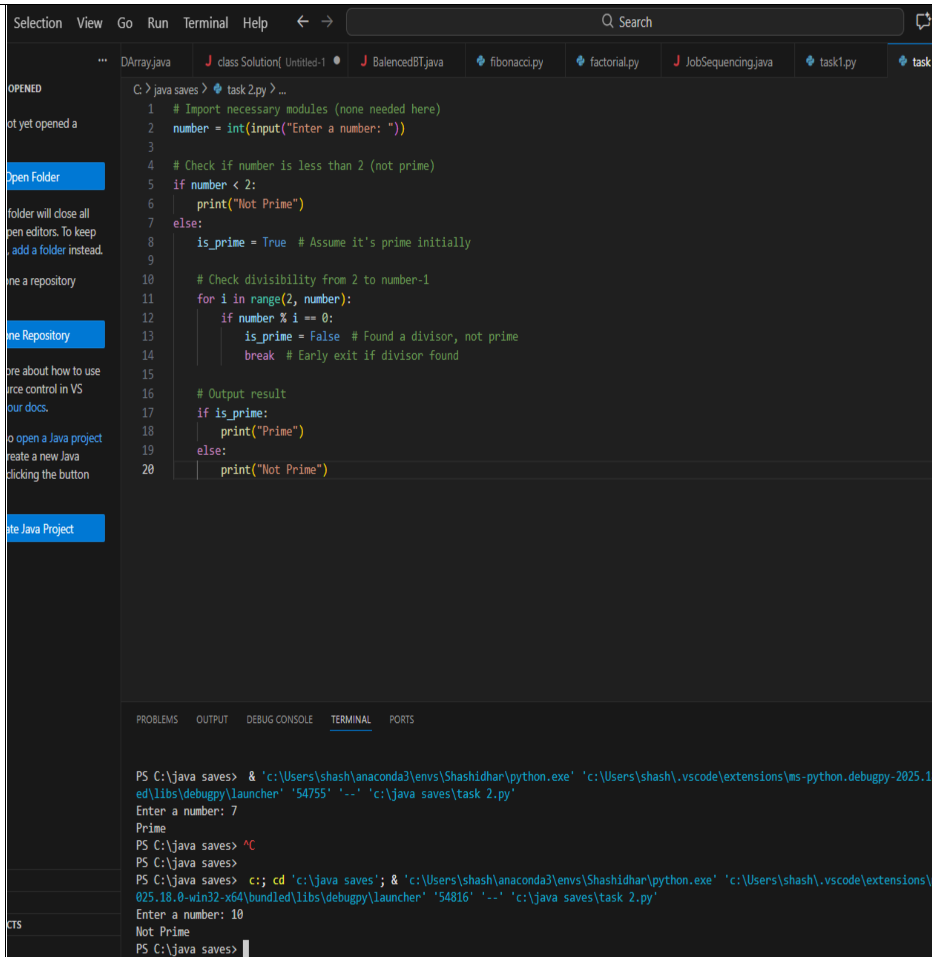
❖ **Task Description**

Use GitHub Copilot to generate a Python program that:

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

❖ **Expected Output**

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



Task 2: Efficiency & Logic Optimization (Cleanup)

- **Scenario**

  The script must handle larger input values efficiently.

- **Task Description**

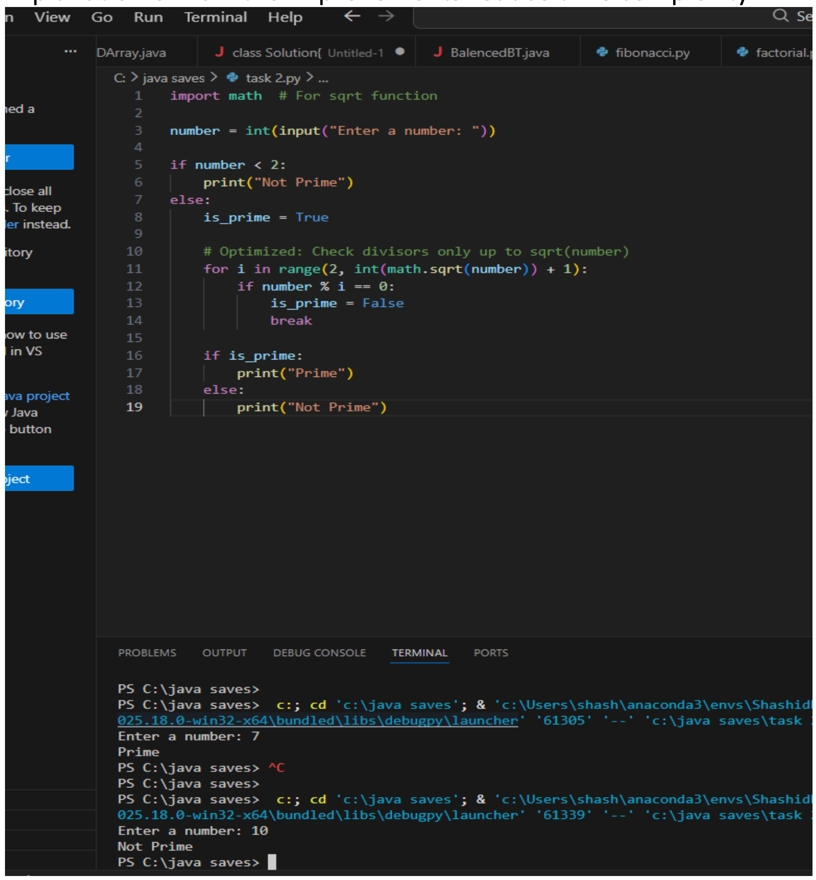
  Review the Copilot-generated code from Task 1 and improve it by:

  - 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



➢

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

❖ **Scenario**

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

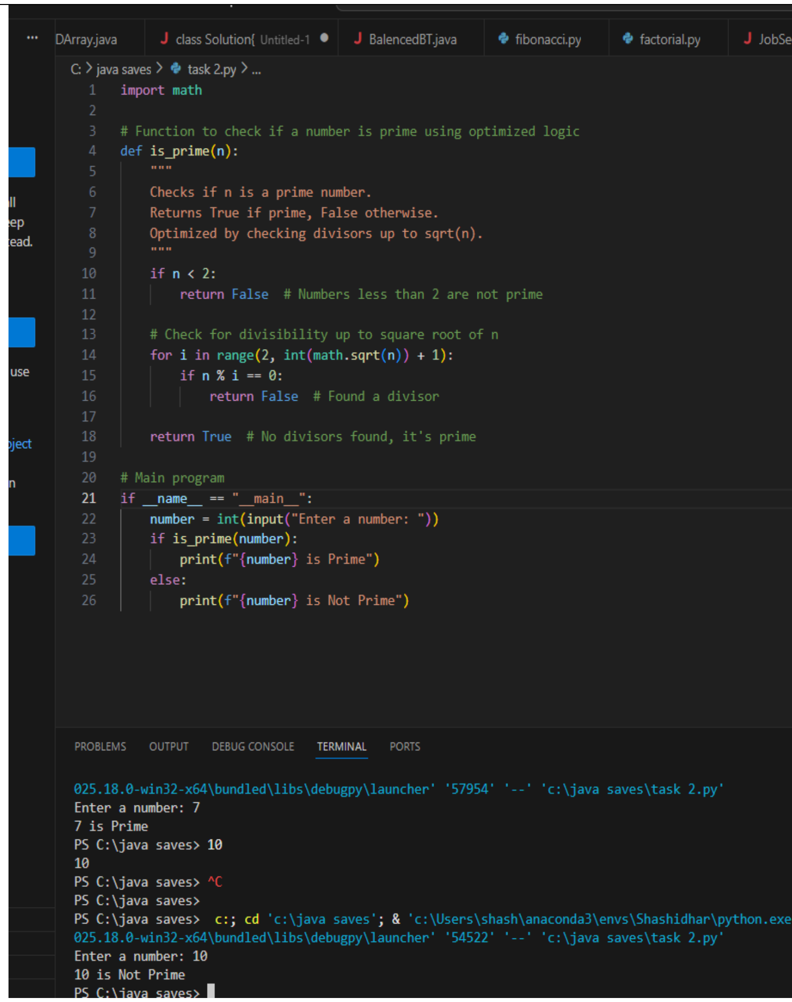
❖ **Task Description**

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

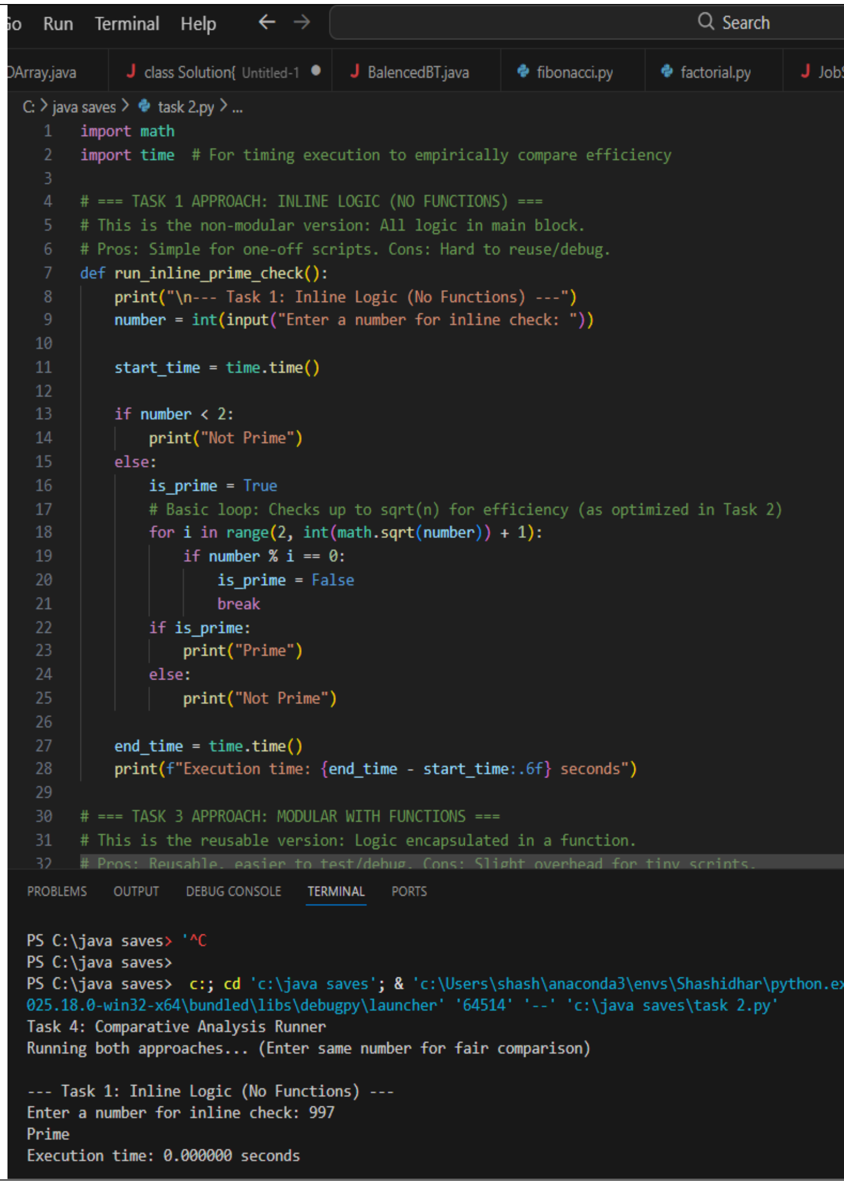


Task 4: Comparative Analysis –With vs Without Functions

- **Scenario**
  You are participating in a technical review discussion.

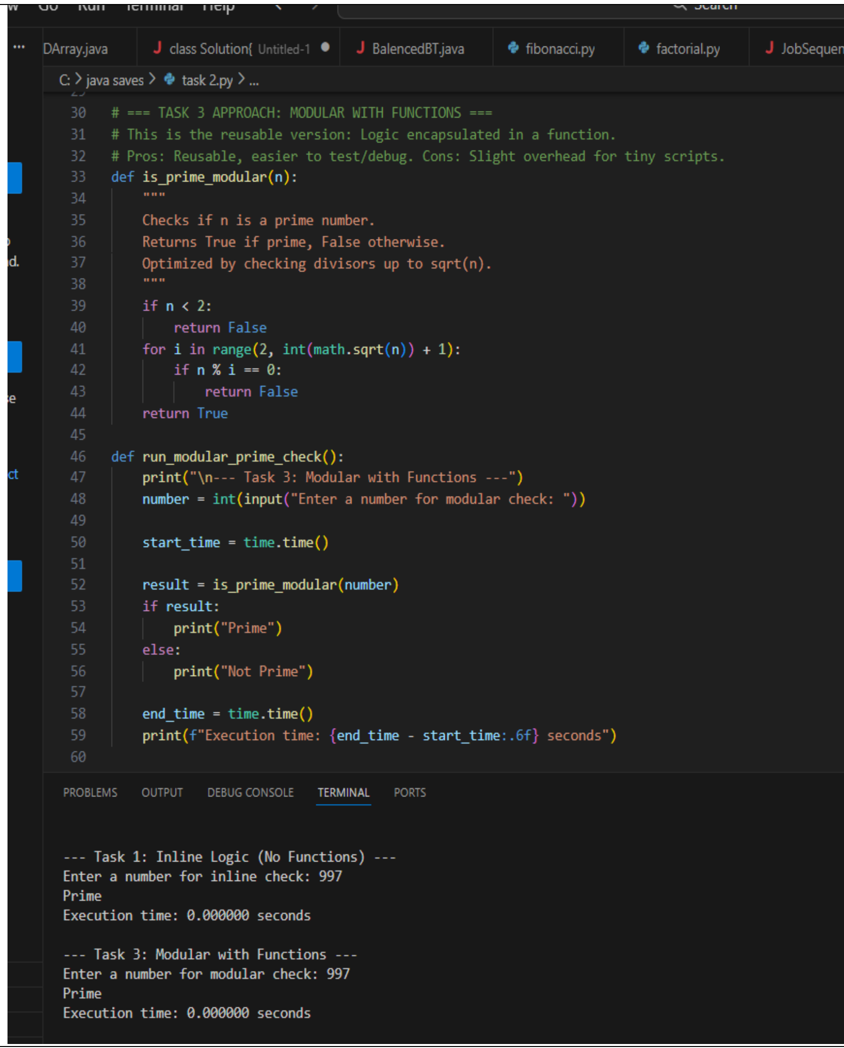
- **Task Description**
  Compare the Copilot-generated programs:
  - Without functions (Task 1)
  - With functions (Task 3)
  - Analyze them based on:
  - Code clarity
  - Reusability
  - Debugging ease
  - Suitability for large-scale applications

- **Expected Output**
  Comparison table or short analytical report



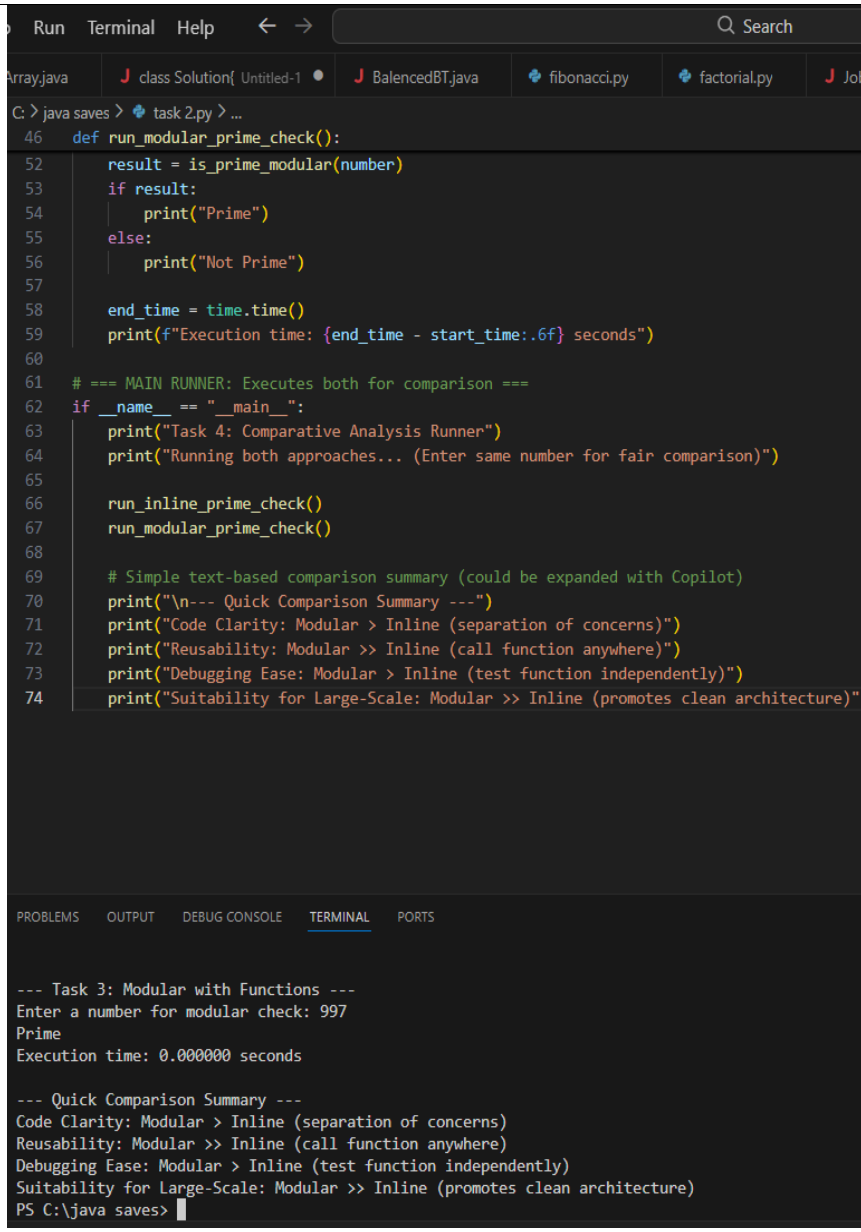
```
import math
import time  # For timing execution to empirically compare efficiency

# === TASK 1 APPROACH: INLINE LOGIC (NO FUNCTIONS) ===
# This is the non-modular version: All logic in main block.
# Pros: Simple for one-off scripts. Cons: Hard to reuse/debug.
def run_inline_prime_check():
    print("\n--- Task 1: Inline Logic (No Functions) ---")
    number = int(input("Enter a number for inline check: "))

    start_time = time.time()

    if number < 2:
        print("Not Prime")
    else:
        is_prime = True
        # Basic loop: Checks up to sqrt(n) for efficiency (as optimized in Task 2)
        for i in range(2, int(math.sqrt(number)) + 1):
            if number % i == 0:
                is_prime = False
                break
        if is_prime:
            print("Prime")
        else:
            print("Not Prime")

    end_time = time.time()
    print(f"Execution time: {end_time - start_time:.6f} seconds")

# === TASK 3 APPROACH: MODULAR WITH FUNCTIONS ===
# This is the reusable version: Logic encapsulated in a function.
# 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
```



```python
# === TASK 3 APPROACH: MODULAR WITH FUNCTIONS ===
# This is the reusable version: Logic encapsulated in a function.
# Pros: Reusable, easier to test/debug. Cons: Slight overhead for tiny scripts.
def is_prime_modular(n):
    """
    Checks if n is a prime number.
    Returns True if prime, False otherwise.
    Optimized by checking divisors up to sqrt(n).
    """
    if n < 2:
        return False
    for i in range(2, int(math.sqrt(n)) + 1):
        if n % i == 0:
            return False
    return True

def run_modular_prime_check():
    print("\n--- Task 3: Modular with Functions ---")
    number = int(input("Enter a number for modular check: "))

    start_time = time.time()

    result = is_prime_modular(number)
    if result:
        print("Prime")
    else:
        print("Not Prime")

    end_time = time.time()
    print(f"Execution time: {end_time - start_time:.6f} 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
```



```python
def run_modular_prime_check():
    result = is_prime_modular(number)
    if result:
        print("Prime")
    else:
        print("Not Prime")

    end_time = time.time()
    print(f"Execution time: {end_time - start_time:.6f} seconds")

# === MAIN RUNNER: Executes both for comparison ===
if __name__ == "__main__":
    print("Task 4: Comparative Analysis Runner")
    print("Running both approaches... (Enter same number for fair comparison)")

    run_inline_prime_check()
    run_modular_prime_check()

    # Simple text-based comparison summary (could be expanded with Copilot)
    print("\n--- Quick Comparison Summary ---")
    print("Code Clarity: Modular > Inline (separation of concerns)")
    print("Reusability: Modular >> Inline (call function anywhere)")
    print("Debugging Ease: Modular > Inline (test function independently)")
    print("Suitability for Large-Scale: Modular >> Inline (promotes clean architecture)")
```

```
--- 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**

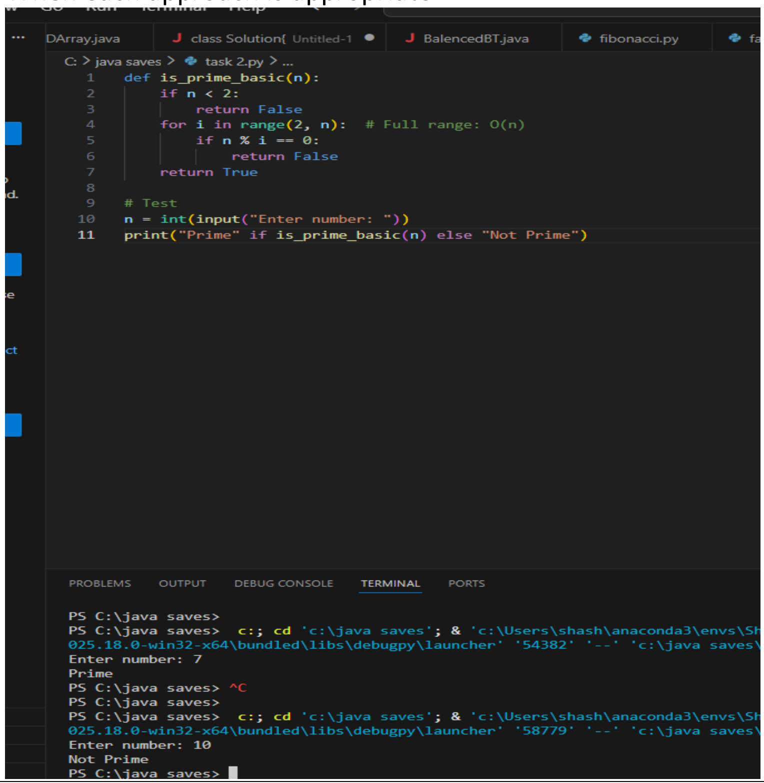
Your mentor wants to evaluate how AI handles **alternative logical strategies**.

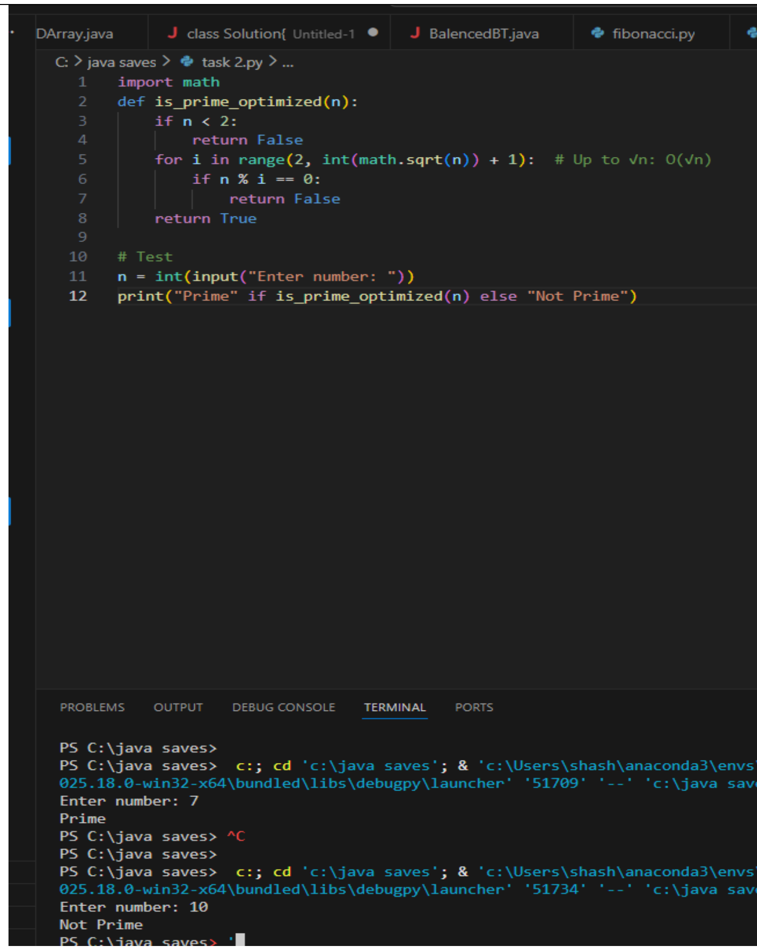
❖ **Task Description**

Prompt GitHub Copilot to generate:

- A **basic divisibility check** approach
- An **optimized approach** (e.g., checking up to √n)

❖ **Expected Output**

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

- 

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		DEPAR

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 batche

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 **Lab Objectives:**

- To explore and evaluate the functionality of G
  assisted coding within Google Colab.
- To understand and use Cursor AI for code ge
  and refactoring.
- To compare outputs and usability between G
  and Cursor AI.
- To perform code optimization and document

Lab Outcomes (LOs):

After completing this lab, students will be able to:

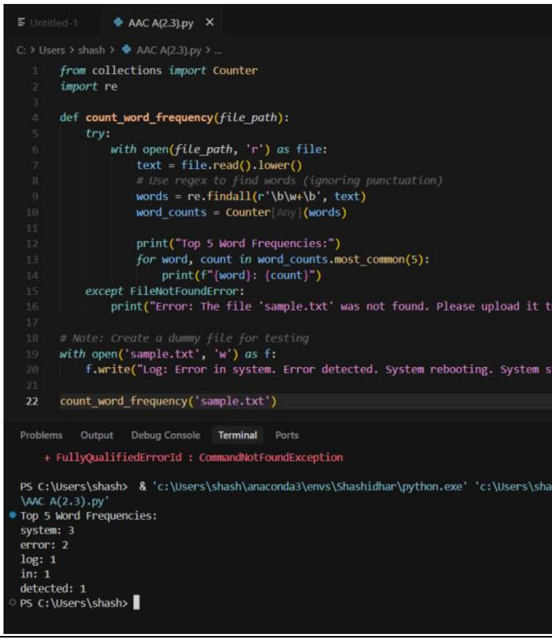
- Generate Python code using Google Gemini in
- Analyze the effectiveness of code explanation
  Gemini.
- Set up and use Cursor AI for AI-powered codin
- Evaluate and refactor code using Cursor AI fea
- Compare AI tool behavior and code quality acr

Task 1: Word Frequency from Text File

- **Scenario:**
  You are analyzing log files for keyword frequency.

- **Task:**
  Use Gemini to generate Python code that reads a t
  frequency, then explains the code.

- **Expected Output:**
  - Working code
  - Explanation
  - Screenshot



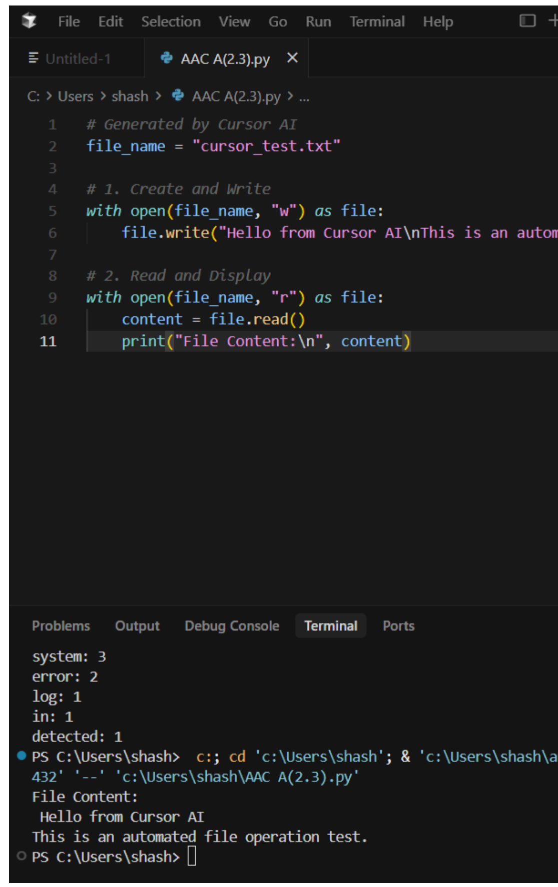
-

Task 2: File Operations Using Cursor AI

❖ **Scenario:**
You are automating basic file operations.

❖ **Task:**
Use Cursor AI to generate a program that:
- ➢ Creates a text file
- ➢ Writes sample text
- ➢ Reads and displays the content

❖ **Expected Output:**
- ➢ Functional code
- ➢ Cursor AI screenshots



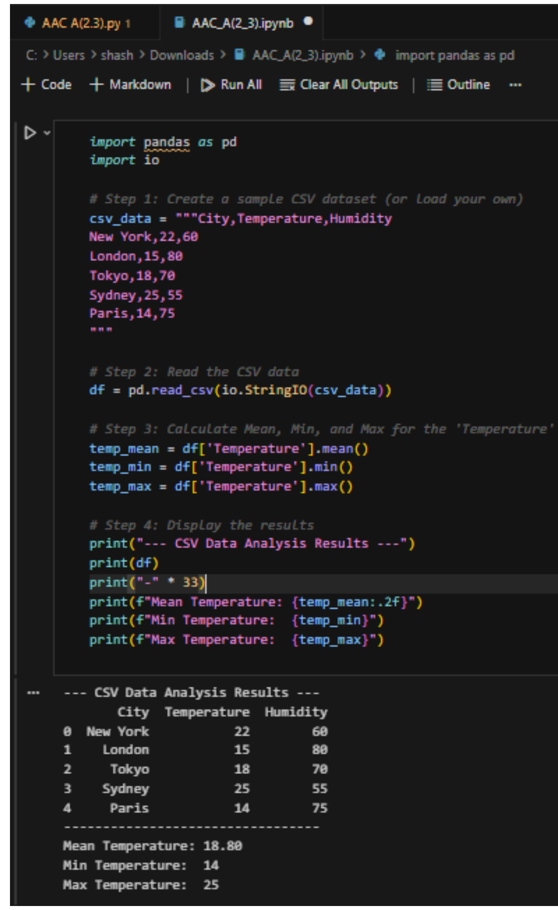
- ➢

Task 3: CSV Data Analysis

❖ **Scenario:**
You are processing structured data from a CSV file

❖ **Task:**
Use Gemini in Colab to read a CSV file and calculat

- ❖ **Expected Output:**
  - ➢ Correct output
  - ➢ Screenshot



AAC A(2.3).py 1 | AAC_A(2_3).ipynb

C: > Users > shash > Downloads > AAC_A(2_3).ipynb > import pandas as pd

+ Code + Markdown | Run All | Clear All Outputs | Outline ...

```python
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("-" * 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
```

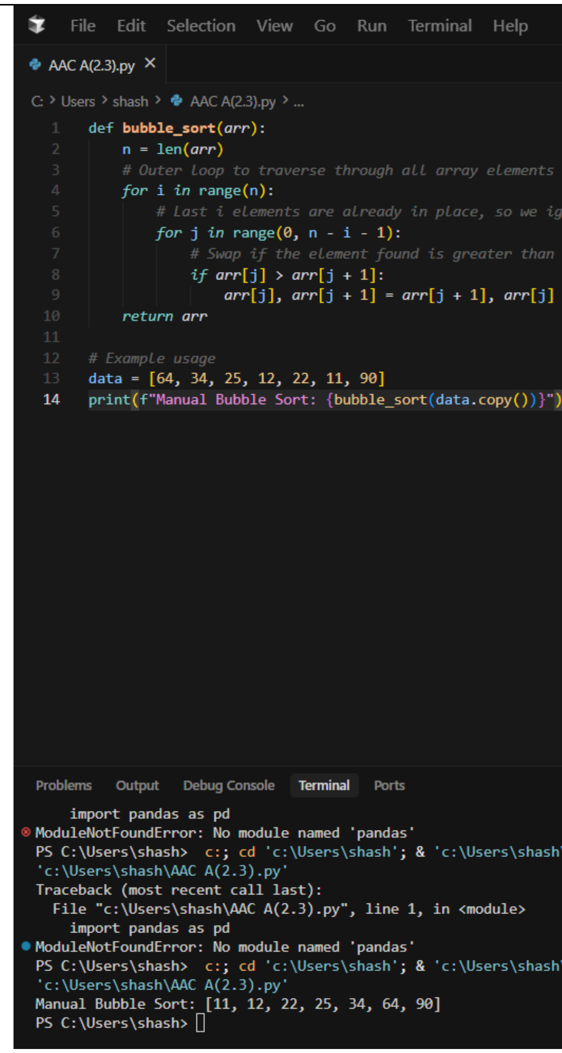
- ➢

Task 4: Sorting Lists – Manual vs Built-in

- ❖ **Scenario:**
  You are reviewing algorithm choices for efficiency.

- ❖ **Task:**
  Use **Gemini** to generate:
  - ➢ Bubble sort
  - ➢ Python's built-in sort()
  - ➢ Compare both implementations.

- ❖ **Expected Output:**
  - ➢ Two versions of code
  - ➢ Short comparison



```
File Edit Selection View Go Run Terminal Help
AAC A(2.3).py ×
C: > Users > shash > AAC A(2.3).py > ...
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 ig
        for j in range(0, n - i - 1):
            # Swap if the element found is greater than
            if arr[j] > 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())}")
```

```
Problems   Output   Debug Console   Terminal   Ports
    import pandas as pd
ModuleNotFoundError: No module named 'pandas'
PS C:\Users\shash>  c:; cd 'c:\Users\shash'; & 'c:\Users\shash
'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 <module>
    import pandas as pd
ModuleNotFoundError: No module named 'pandas'
PS C:\Users\shash>  c:; cd 'c:\Users\shash'; & 'c:\Users\shash
'c:\Users\shash\AAC A(2.3).py'
Manual Bubble Sort: [11, 12, 22, 25, 34, 64, 90]
PS C:\Users\shash>
```

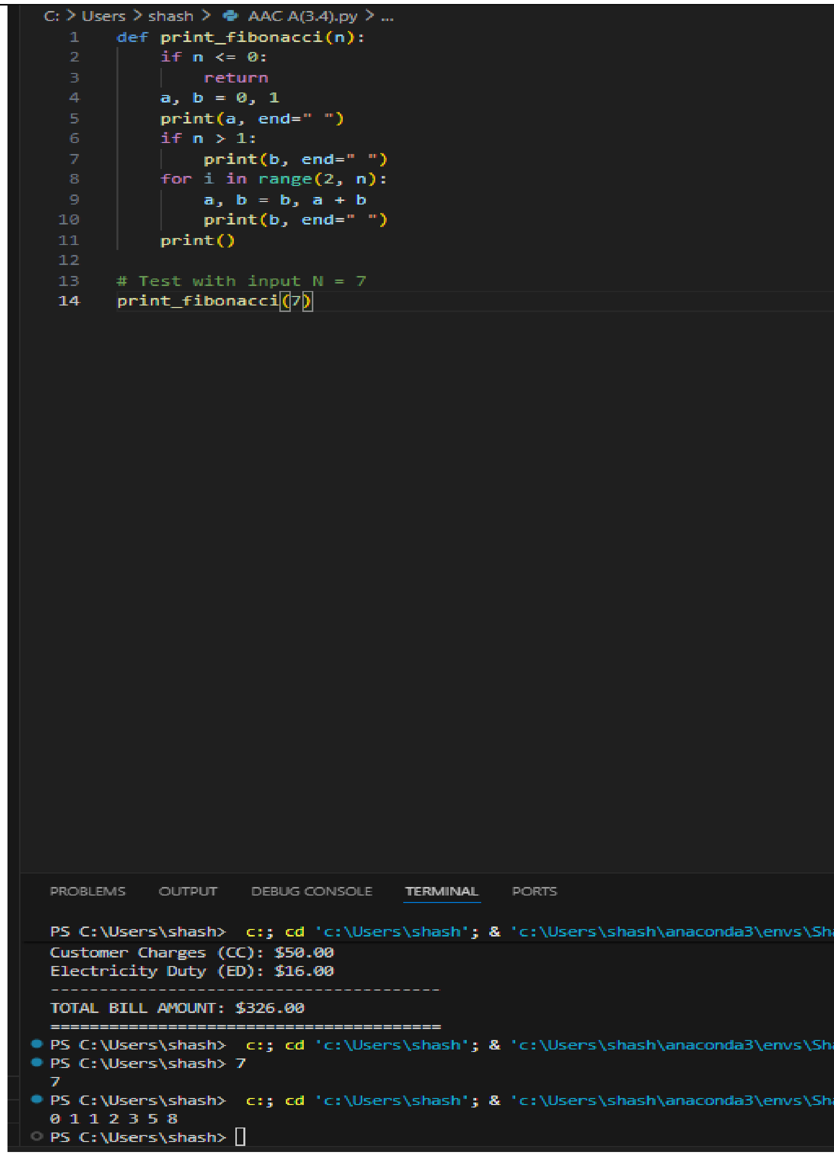
Note: Report should be submitted as a word docume single document with prompts, comments & code ex and if required, screenshots.

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	**Week2**	**Time(s)**	23CSBTB01 To 23CSBTB52
Duration	2 Hours	**Applicable to Batches**	All batches
Assignment Number: 3.4 (Present assignment number)/**24**(Total number of assignments)			

Q.No.	**Question**	***Expected Time to complete***
1	Lab 4: Advanced Prompt Engineering – Zero-shot, One-shot, and Few-shot Techniques	Week2

	Task 1: Zero-shot Prompt – Fibonacci Series Generator **Task Description #1** • Without giving an example, write a single comment prompt asking GitHub Copilot to generate a Python function to print the first N Fibonacci numbers. **Expected Output #1** • A complete Python function generated by Copilot without any example provided. • Correct output for sample input N = 7 → 0 1 1 2 3 5 8 • Observation on how Copilot understood the instruction with zero context.	



Task 2: One-shot Prompt – List Reversal Function

Task Description #2

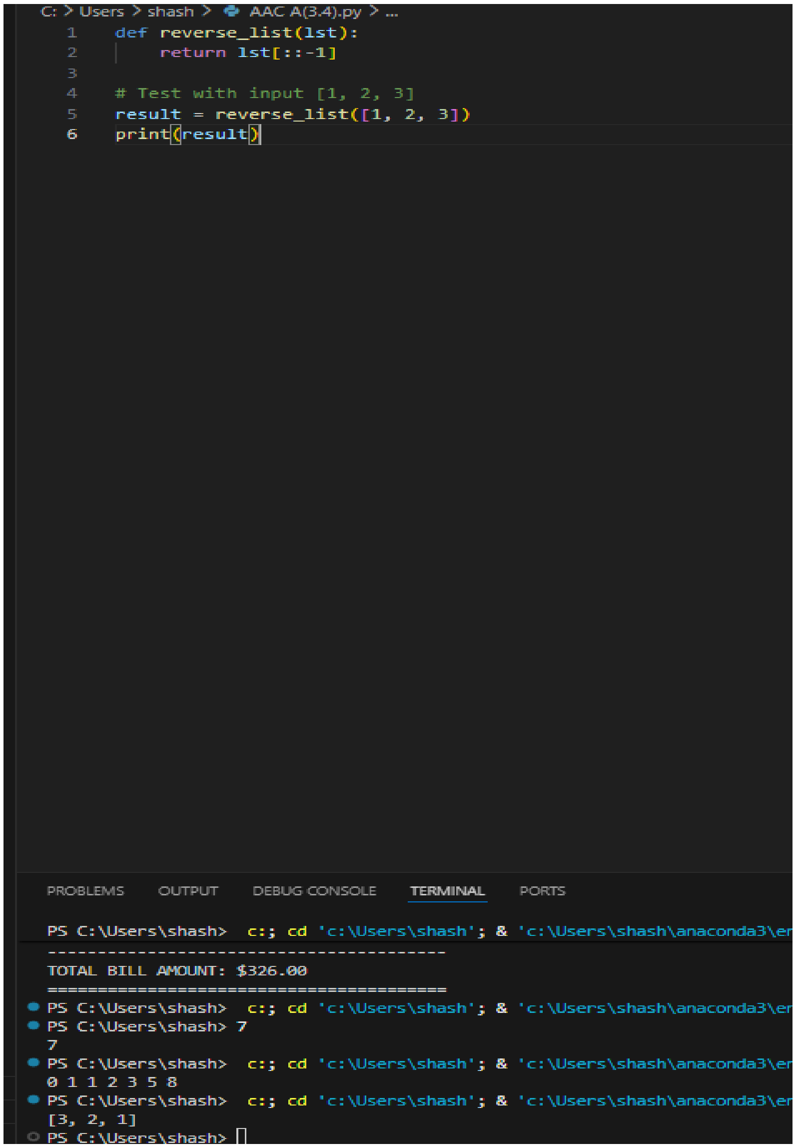
• Write a comment prompt to reverse a list and provide one example below the comment to guide Copilot.

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.



Task 3: Few-shot Prompt – String Pattern Matching

Task Description #3

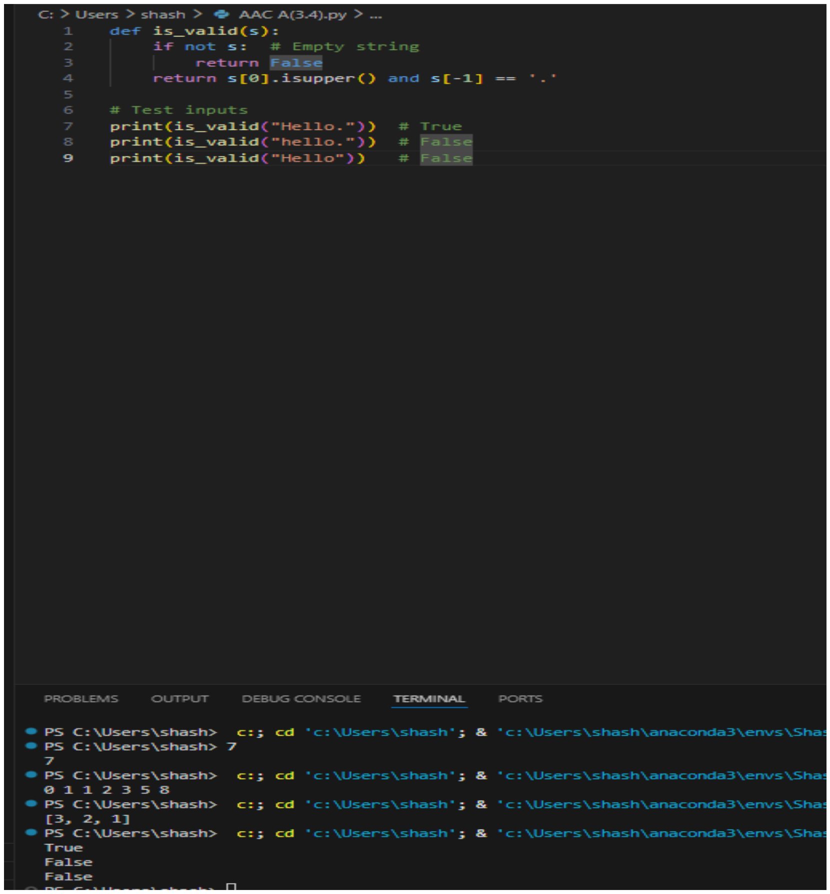
• Write a comment with 2–3 examples to help Copilot understand how to check if a string starts with a capital letter and ends with a period.

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.



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.

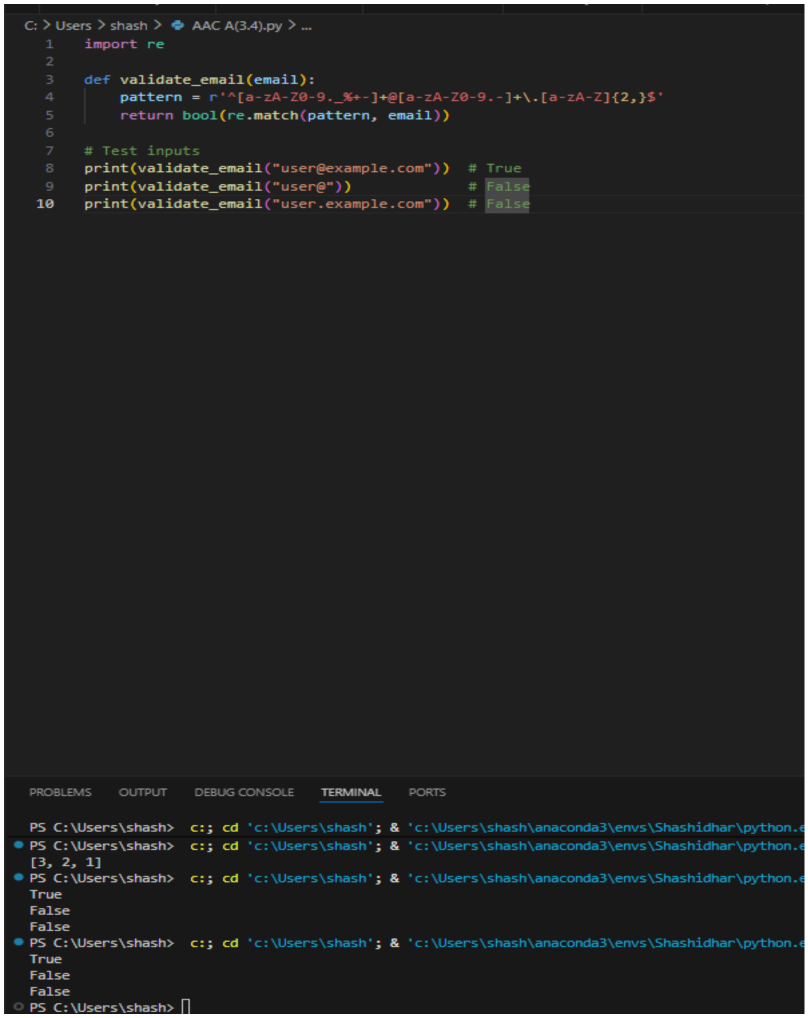
Expected Output #4

• 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.



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

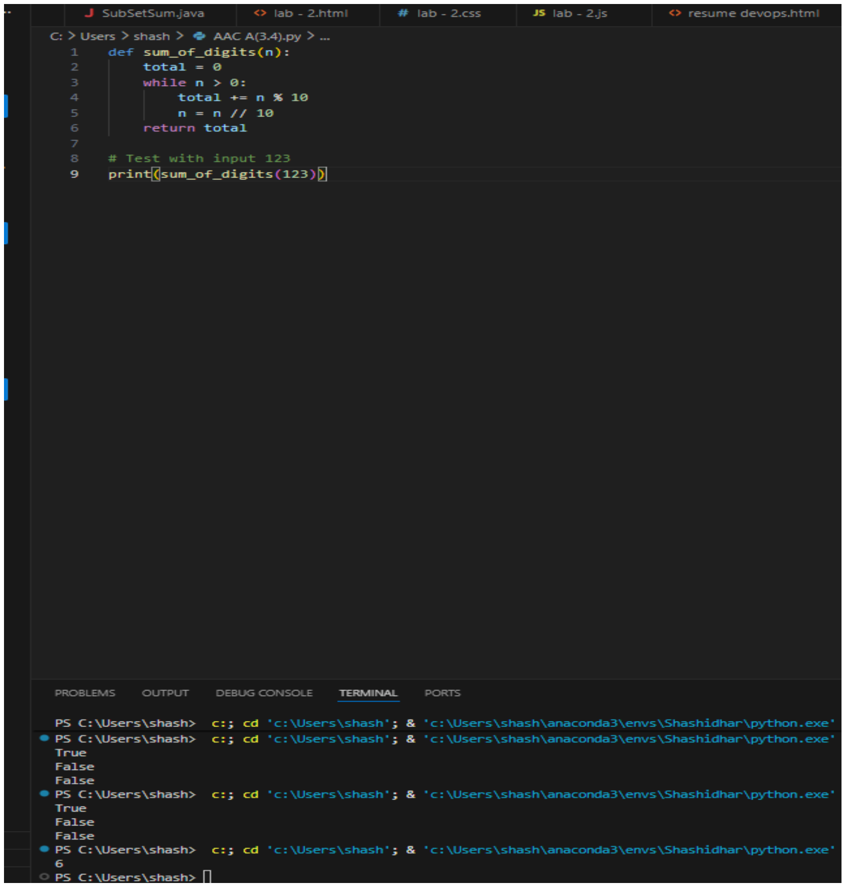
Style 2: Task + Input/Output example

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?



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