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:

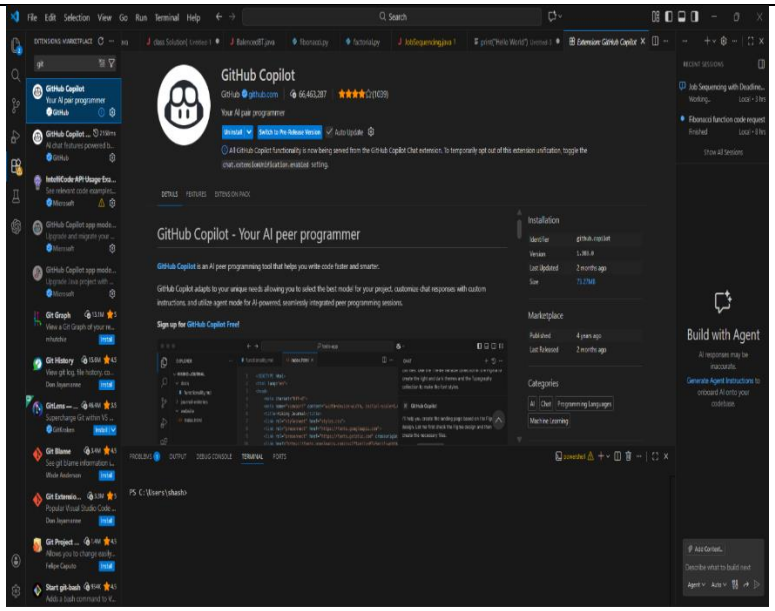
- Set up GitHub Copilot in VS Code successfully.
- Use inline comments and context to generate code with Copilot.
- Evaluate AI-generated code for correctness and readability.
- Compare code suggestions based on different prompts and programming styles.

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**

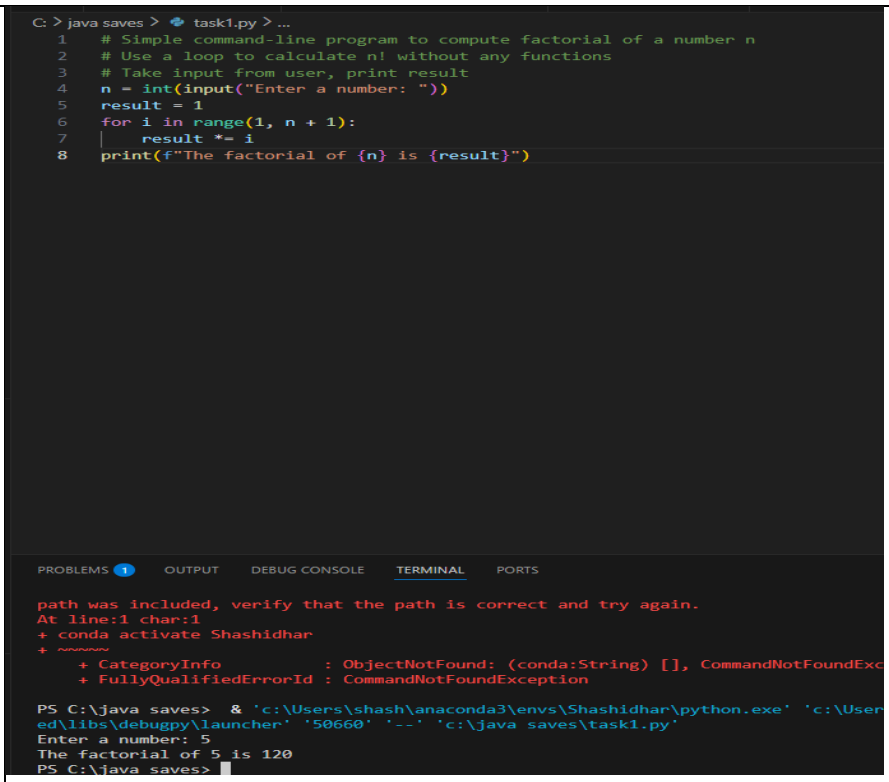
Use GitHub Copilot to generate a Python program that computes a mathematical product-based value (factorial-like logic) directly in the main execution flow, without using any user-defined functions.

- **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

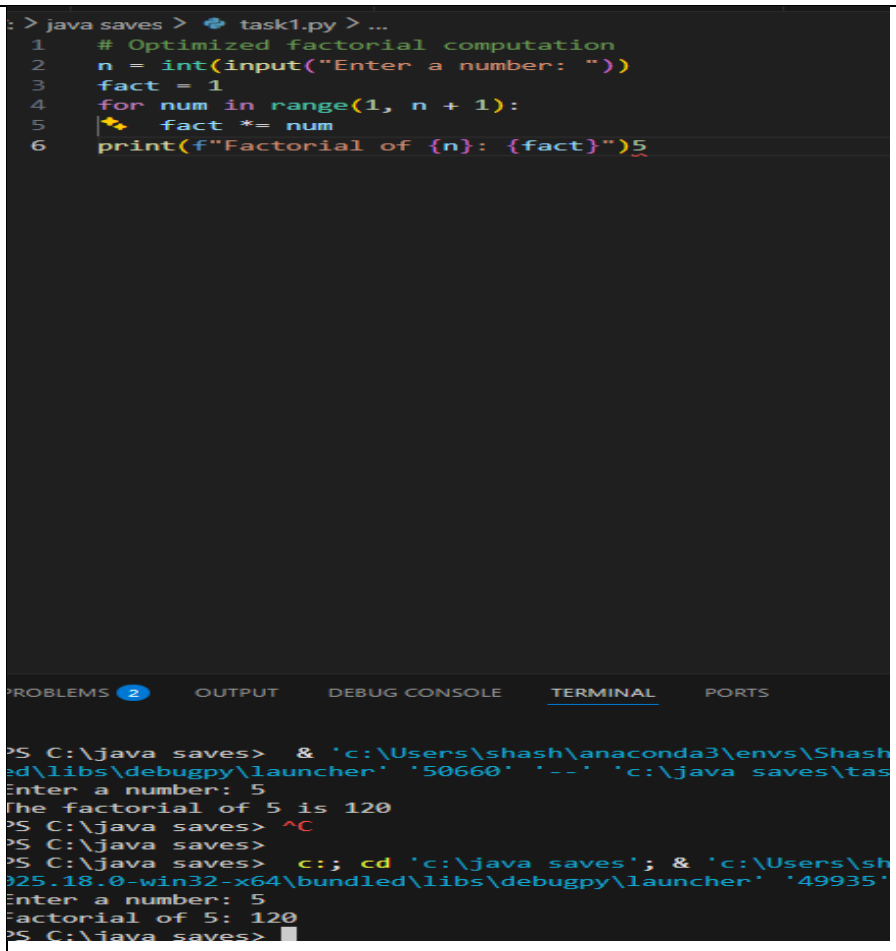
Hint:

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

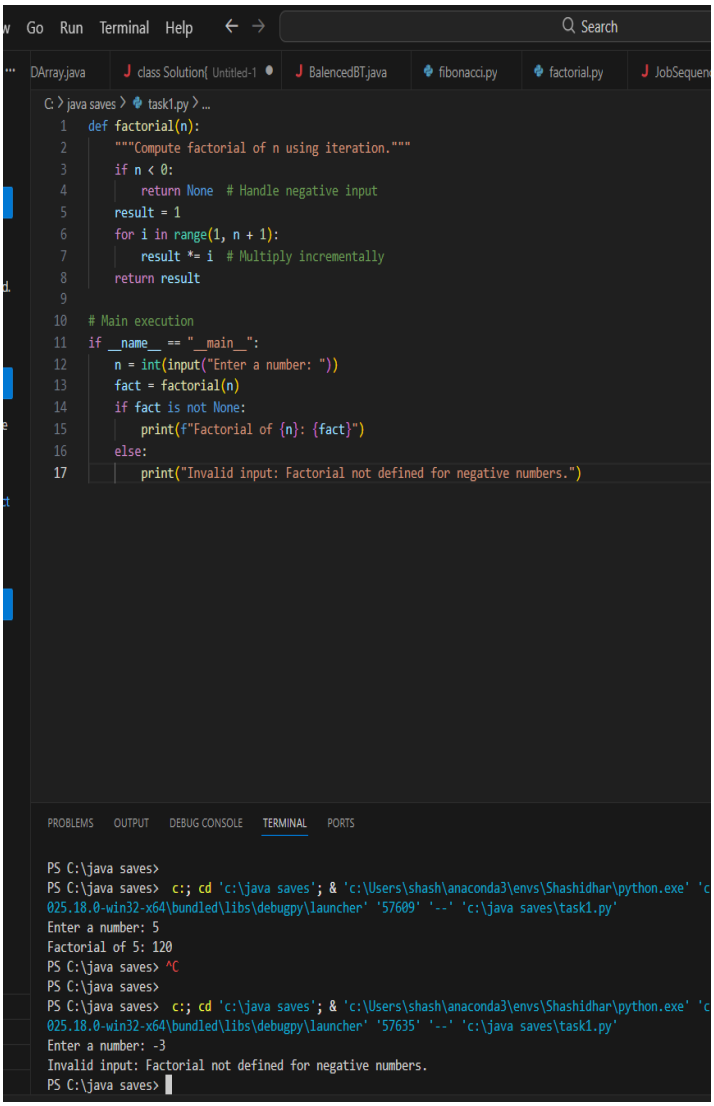
❖ **Constraints**

- 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

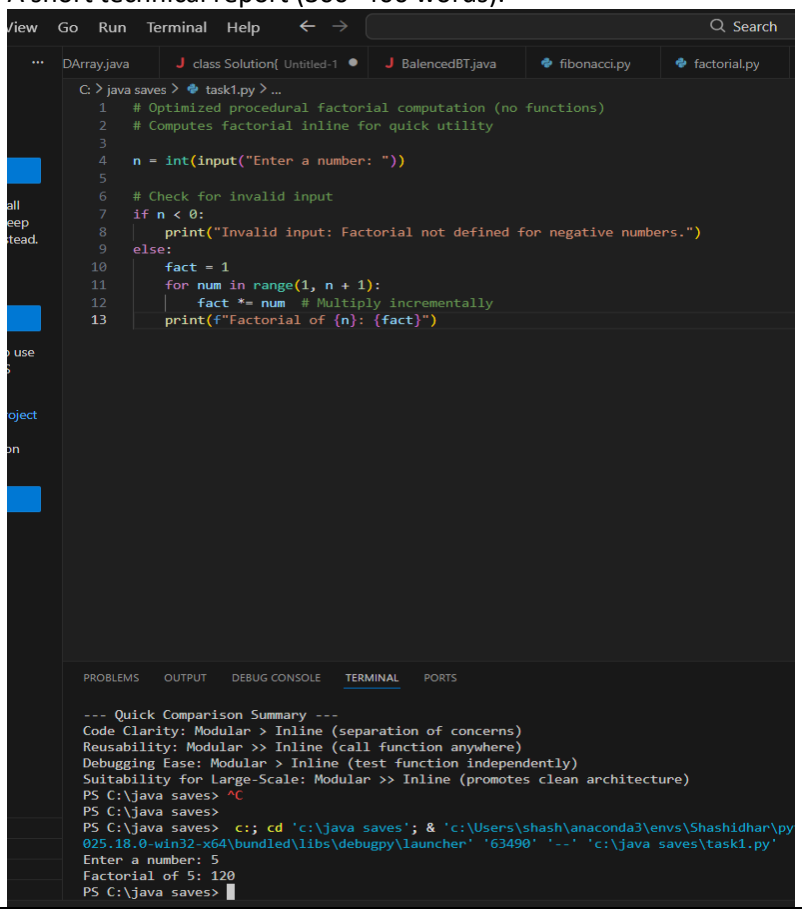
❖ **Expected Deliverables**

Choose **one**:

- A comparison table

  OR

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



DArray.java | class Solution{ Untitled-1 | BalencedBT.java | fibonacci.py | factorial.p

C: > java saves > task1.py > ...

```python
# Modular factorial program using a function for reusability

def factorial(n):
    """
    Compute factorial of n using iteration.
    Handles negative inputs gracefully.
    """
    if n < 0:
        return None  # Handle negative input
    result = 1
    for i in range(1, n + 1):
        result *= i  # Multiply incrementally
    return result

# Main execution block
if __name__ == "__main__":
    n = int(input("Enter a number: "))
    fact = factorial(n)
    if fact is not None:
        print(f"Factorial of {n}: {fact}")
    else:
        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\Shashid
025.18.0-win32-x64\bundled\libs\debugpy\launcher' '63490' '--' 'c:\java saves\task1
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\Shashid
025.18.0-win32-x64\bundled\libs\debugpy\launcher' '63554' '--' 'c:\java saves\task1
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**

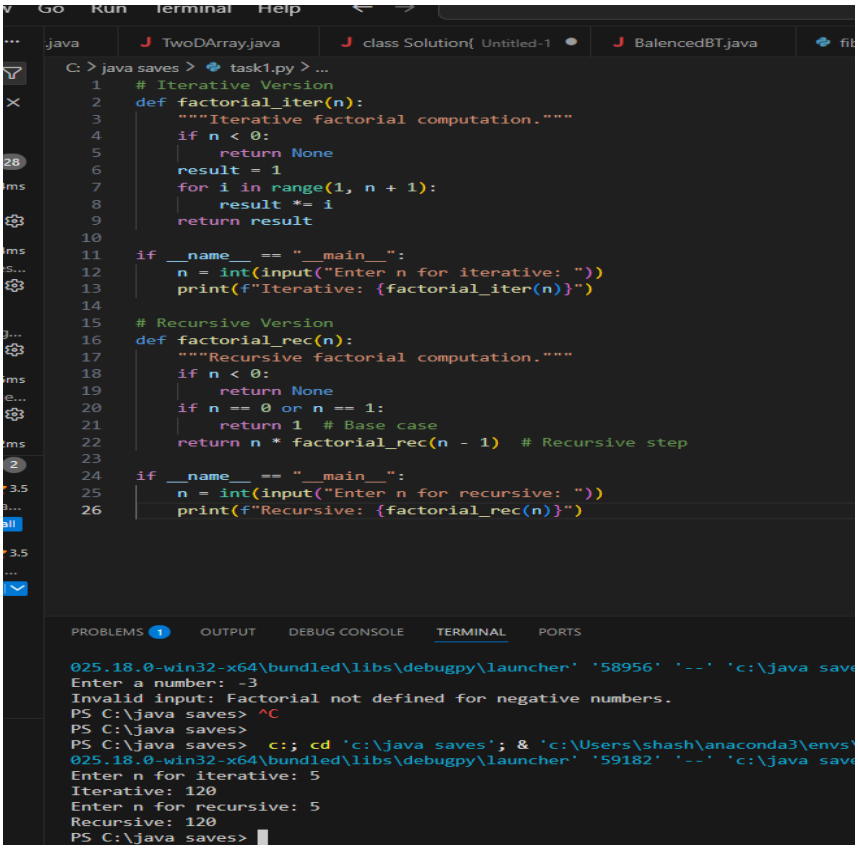
Two AI-generated implementations

Execution flow explanation (in your own words)

Comparison covering:

- ➢ Readability
- ➢ Stack usage
- ➢ Performance implications

- When recursion is *not* recommended.
-



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:

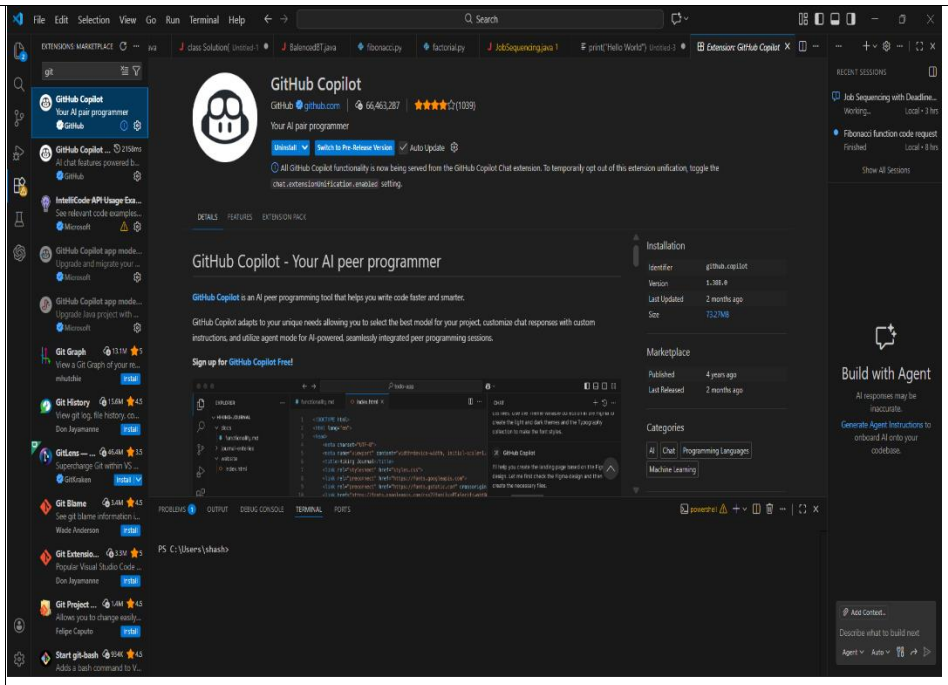
- Set up GitHub Copilot in VS Code successfully.
- Use inline comments and context to generate code with Copilot.
- Evaluate AI-generated code for correctness and readability.
- Compare code suggestions based on different prompts and programming styles.

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.

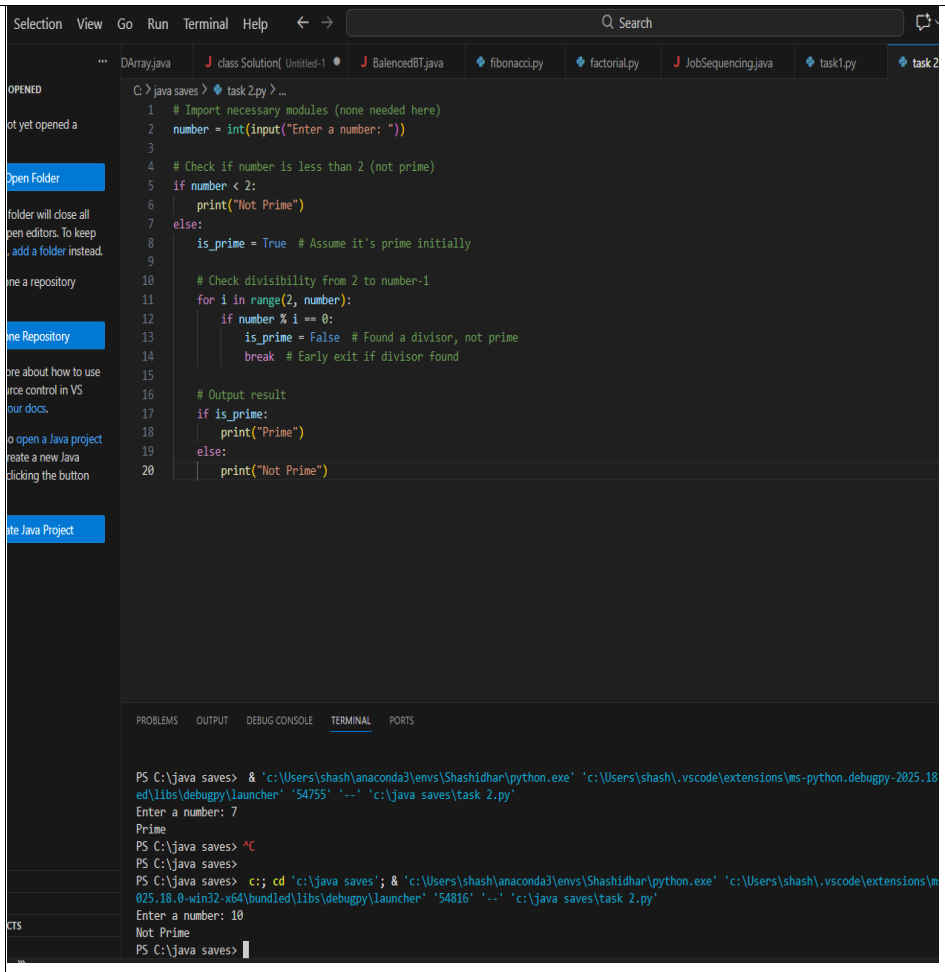
❖ **Task Description**

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:

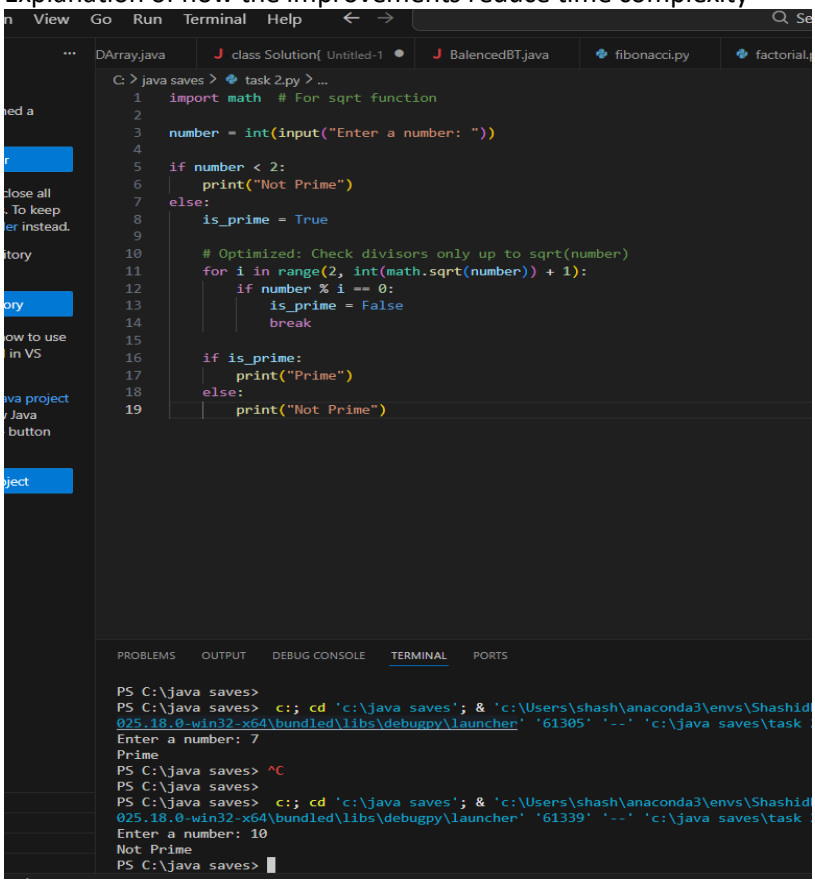
- Reducing unnecessary iterations
- Optimizing the loop range (e.g., early termination)
- Improving readability
- Use Copilot prompts like:
  - *"Optimize prime number checking logic"*
  - *"Improve efficiency of this code"*

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.

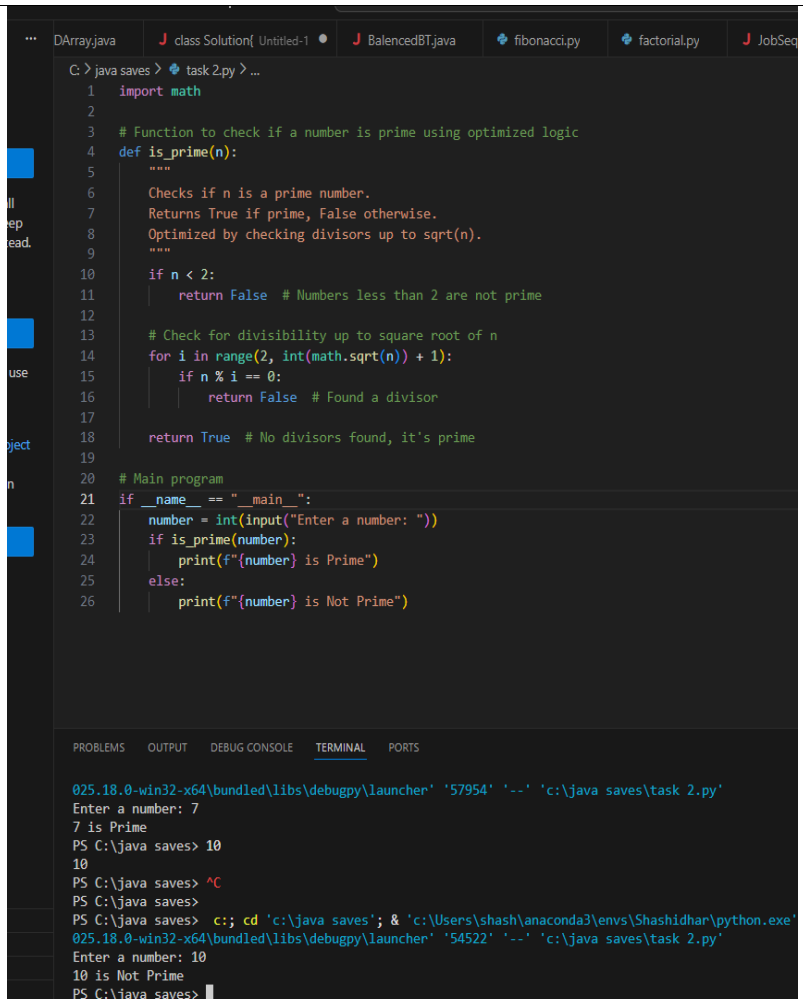
❖ **Task Description**

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.

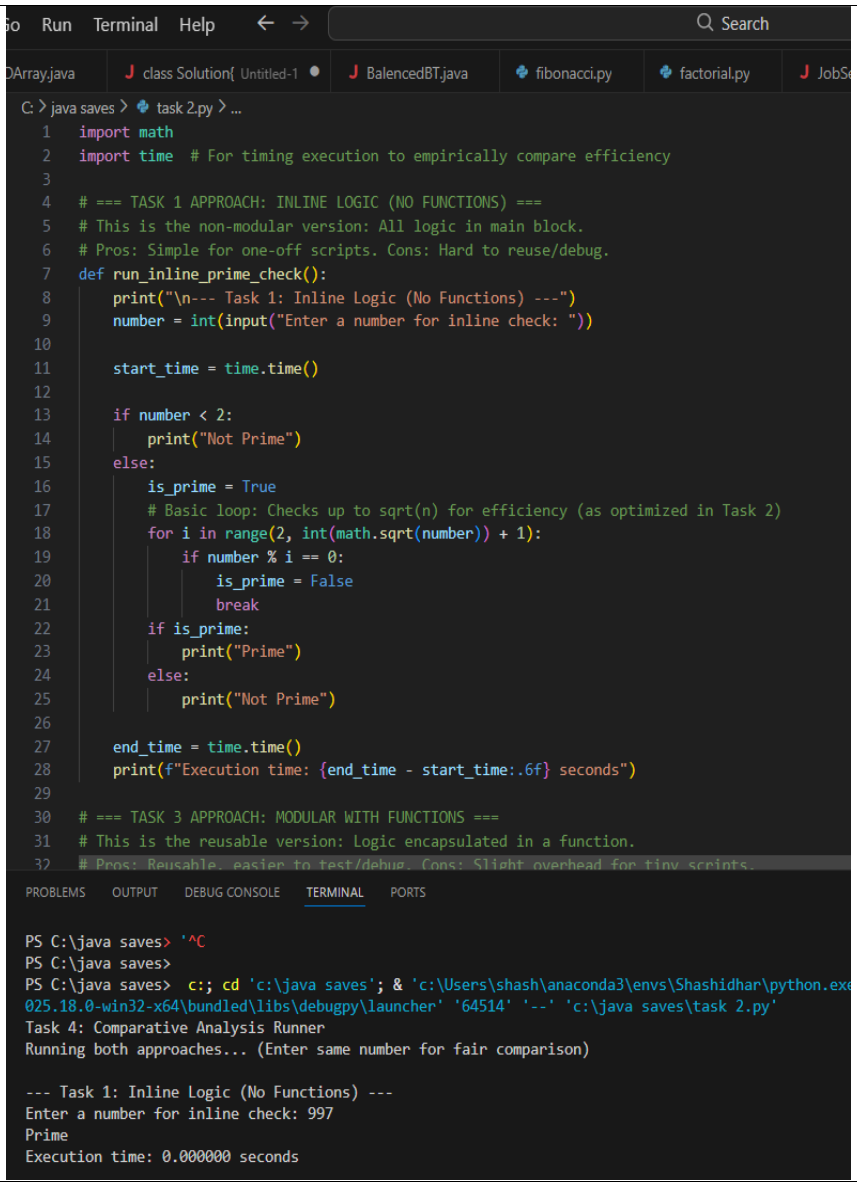
❖ **Task Description**

Compare the Copilot-generated programs:

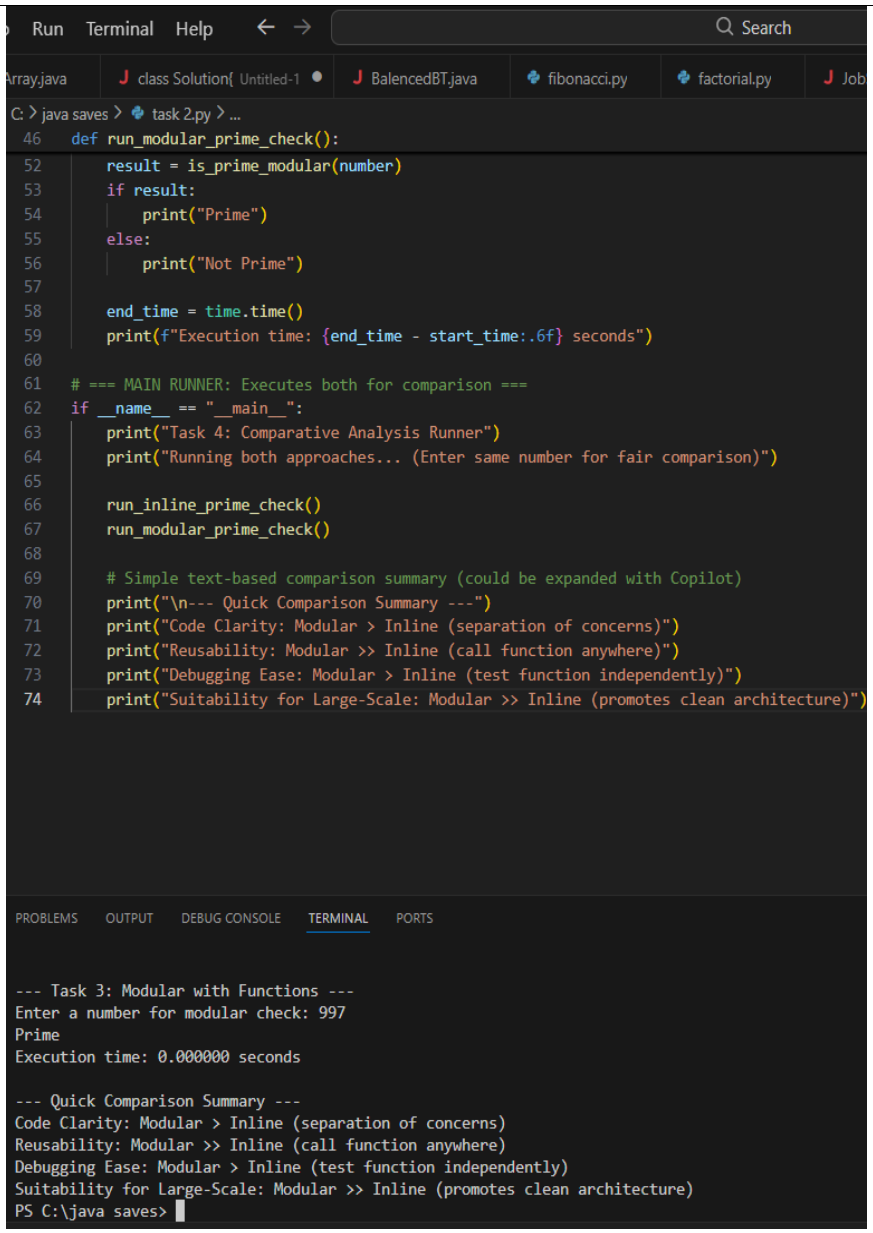
- ➢ Without functions (Task 1)
- ➢ With functions (Task 3)
- ➢ Analyze them based on:
- ➢ Code clarity
- ➢ Reusability
- ➢ Debugging ease
- ➢ Suitability for large-scale applications

❖ **Expected Output**

Comparison table or short analytical report


Go Run Terminal Help
Search
DArray.java
class Solution{ Untitled-1
BalencedBT.java
fibonacci.py
factorial.py
C: > java saves > task 2.py > ...
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.
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\java saves> '^C
PS C:\java saves>
PS C:\java saves> c:; cd 'c:\java saves'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe
025.18.0-win32-x64\bundled\libs\debugpy\launcher' '64514' '--' 'c:\java saves\task 2.py'
Task 4: Comparative Analysis Runner
Running both approaches... (Enter same number for fair comparison)
--- Task 1: Inline Logic (No Functions) ---
Enter a number for inline check: 997
Prime
Execution time: 0.000000 seconds


DArray.java
class Solution{ Untitled-1
BalencedBT.java
fibonacci.py
factorial.py
JobSequenc
C: > java saves > task 2.py > ...
# === 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")
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

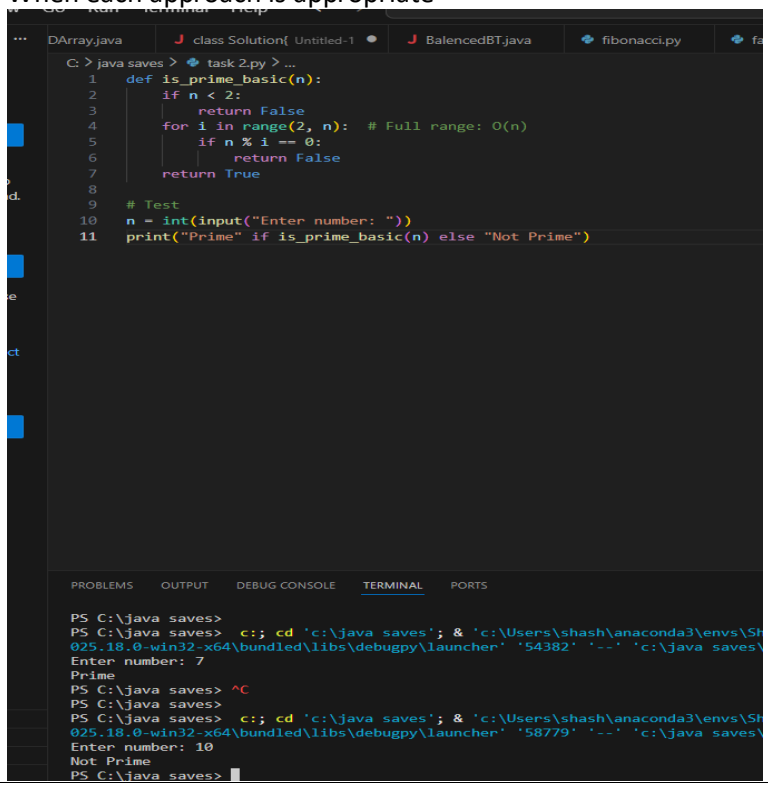


Task 5: AI-Generated Iterative vs Recursive Fibonacci Approaches (Different Algorithmic Approaches to Prime Checking)

❖ **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 ▪ 	

DArray.java | class Solution{ Untitled-1 | BalencedBT.java | fibonacci.py

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

```python
import math
def is_prime_optimized(n):
    if n < 2:
        return False
    for i in range(2, int(math.sqrt(n)) + 1):  # Up to √n: O(√n)
        if n % i == 0:
            return False
    return True

# Test
n = int(input("Enter number: "))
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			**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 C assisted coding within Google Colab.
- To understand and use Cursor AI for code ger 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:

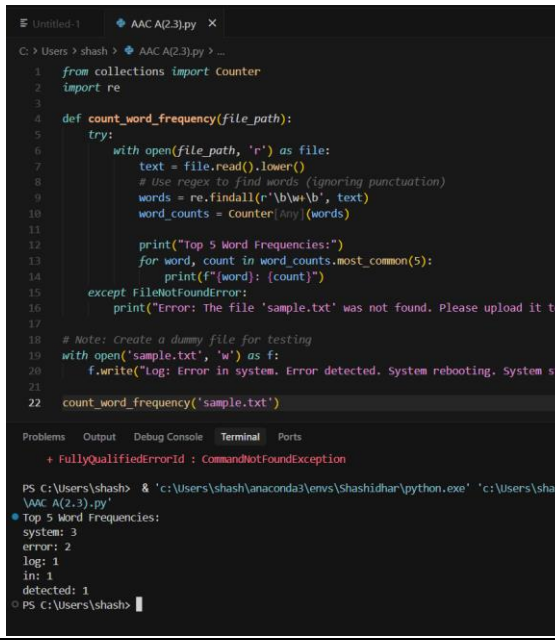
- Generate Python code using Google Gemini in
- Analyze the effectiveness of code explanations Gemini.
- Set up and use Cursor AI for AI-powered codin
- Evaluate and refactor code using Cursor AI fea
- Compare AI tool behavior and code quality acr

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

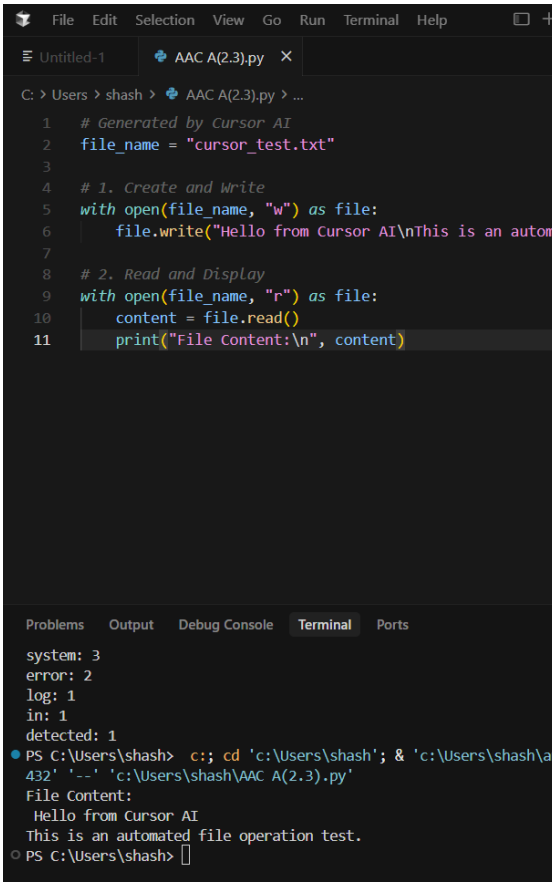


Task 2: File Operations Using Cursor AI

❖ **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



➢

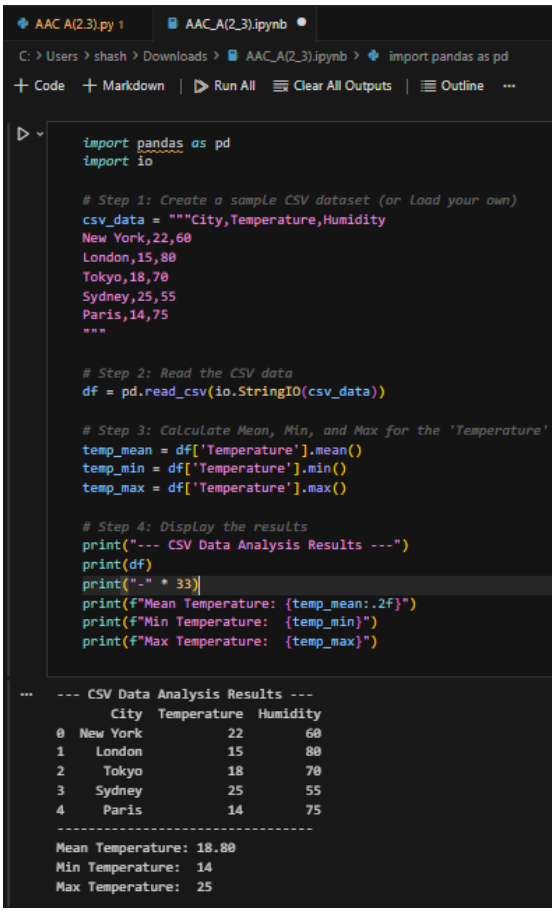
Task 3: CSV Data Analysis

❖ **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



➢

Task 4: Sorting Lists – Manual vs Built-in

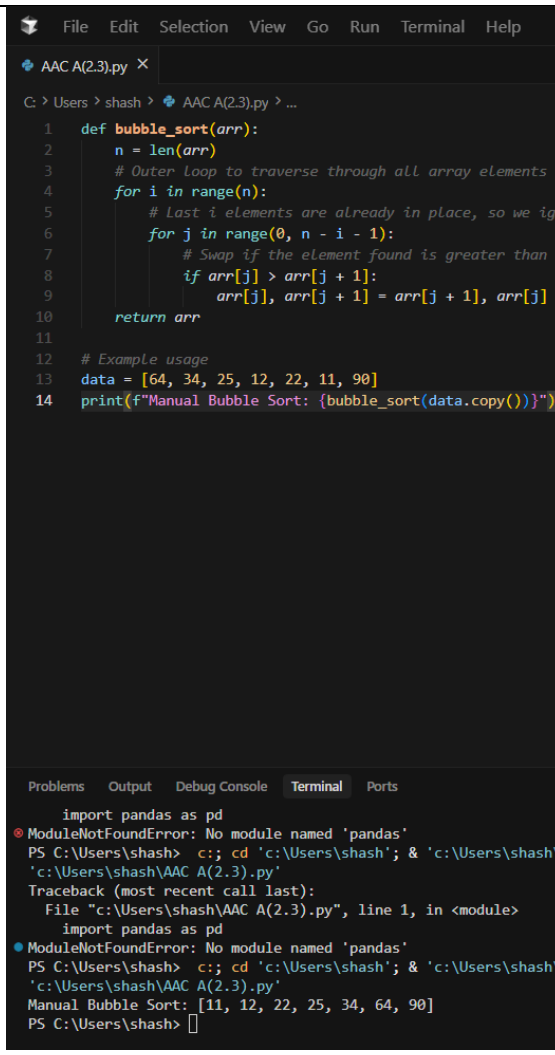
❖ **Scenario:**
You are reviewing algorithm choices for efficiency.

❖ **Task:**
Use **Gemini** to generate:

- Bubble sort
- Python's built-in sort()
- Compare both implementations.

❖ **Expected Output:**

- Two versions of code
- Short comparison



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.	

```
C: > Users > shash > AAC A(3.4).py > ...
def print_fibonacci(n):
    if n <= 0:
        return
    a, b = 0, 1
    print(a, end=" ")
    if n > 1:
        print(b, end=" ")
    for i in range(2, n):
        a, b = b, a + b
        print(b, end=" ")
    print()

# Test with input N = 7
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.

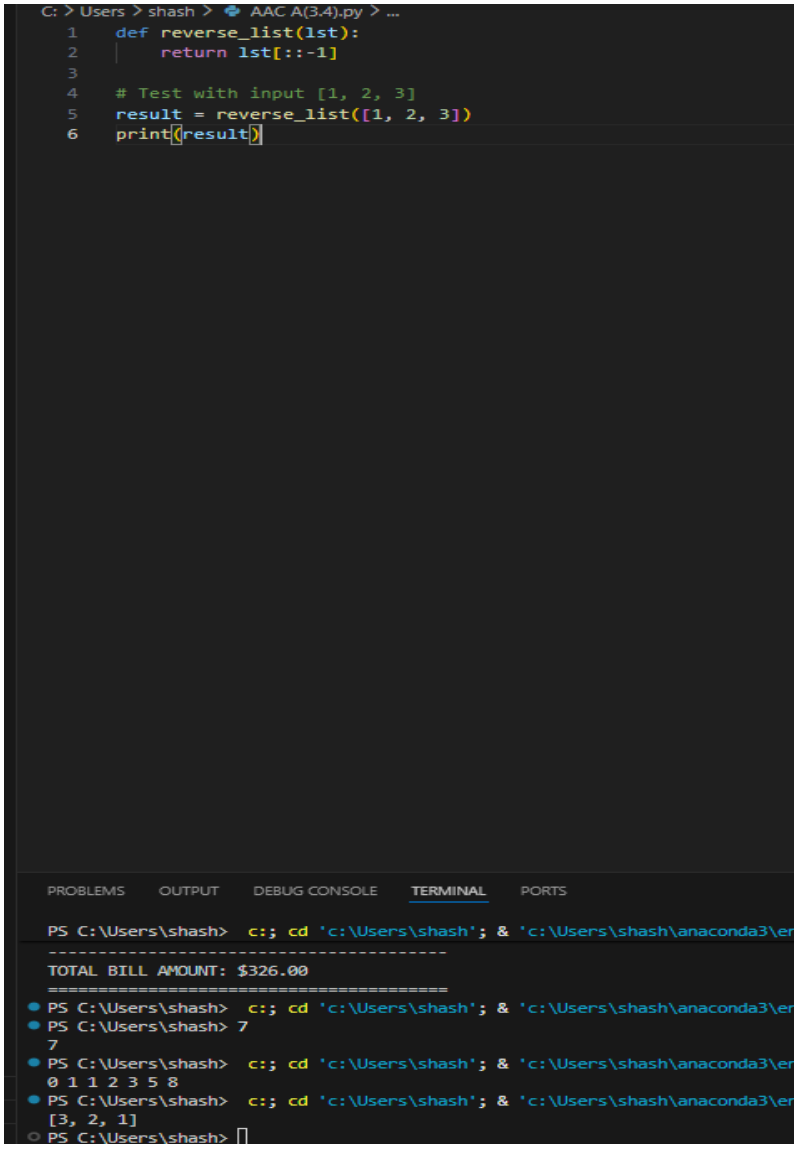
Expected Output #2

• 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.



Task 3: Few-shot Prompt – String Pattern Matching

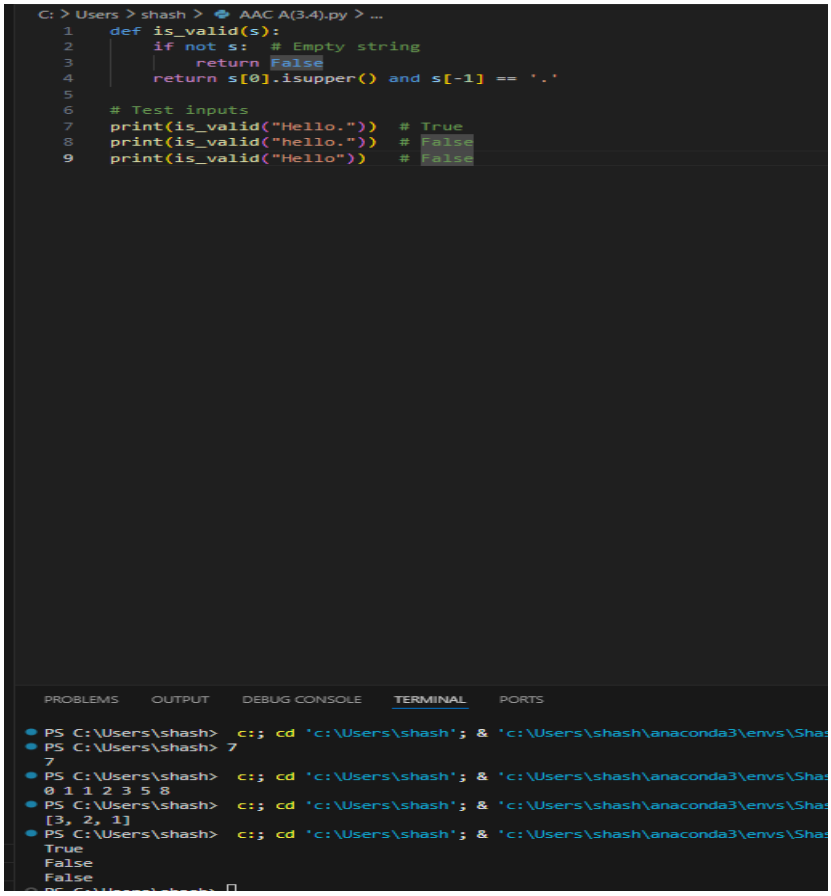
Task Description #3

• 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.

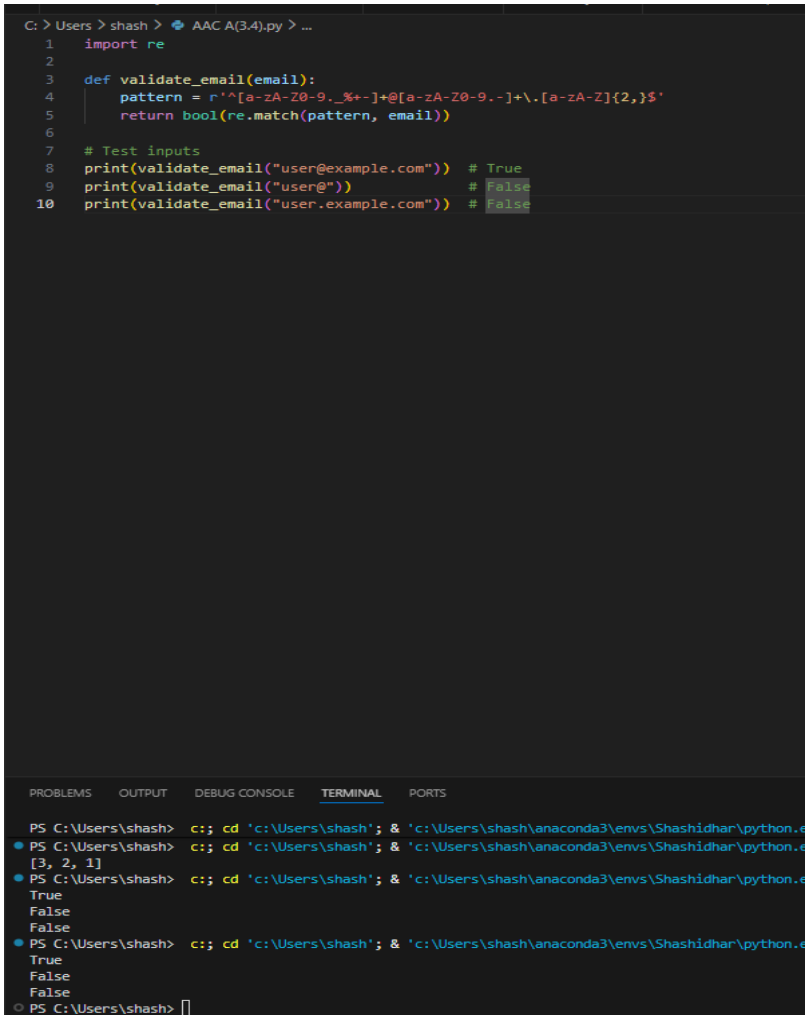
Expected Output #3

• 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

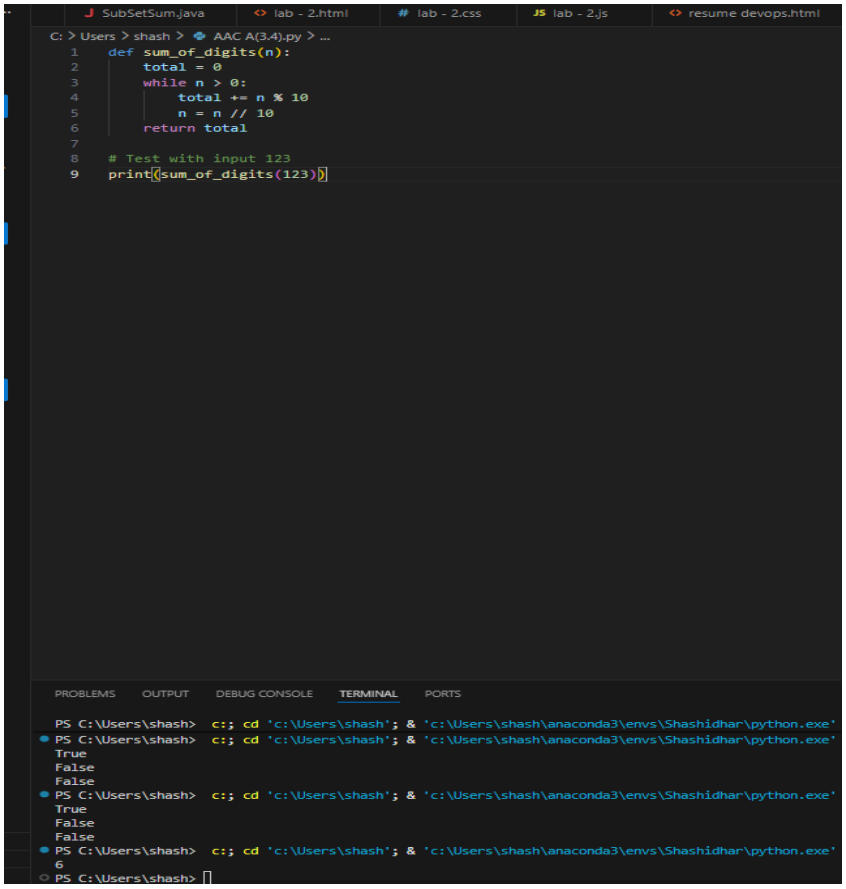
Expected Output #5

• 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

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

ASSIGNMENT-3.3

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		
Course Code	23CS002PC304	**Course Title**	AI Assisted Coding
Year/Sem	III/I	**Regulation**	R23
Date and Day of Assignment	Week 2 - Wednesday	**Time(s)**	23CSBTB01 To 23CSBTB52
Duration	2 Hours	**Applicable to Batches**	All batches
Assignment Number: 3.3(Present assignment number)/**24**(Total number of assignments)			

Q.No.	Question	*Expected Time to complete*
1	**Lab 3: Application for TGNPDCL – Electricity Bill Generation Using Python & AI Tools** **Lab Objectives** • To design a real-world electricity billing application using Python • To use AI-assisted coding tools for logic generation and optimization • To understand conditional logic and arithmetic operations • To generate structured billing output similar to utility bills **Lab Outcomes (LOs)** After completing this lab, students will be able to:	Week2 - Wednesday

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

ASSIGNMENT-3.3

- Read and validate user input in Python
- Apply conditional logic for tariff-based billing
- Use AI tools to assist in program development
- Calculate and display electricity bill components
- Build a complete real-time application

Task 1: AI-Generated Logic for Reading Consumer Details

Scenario

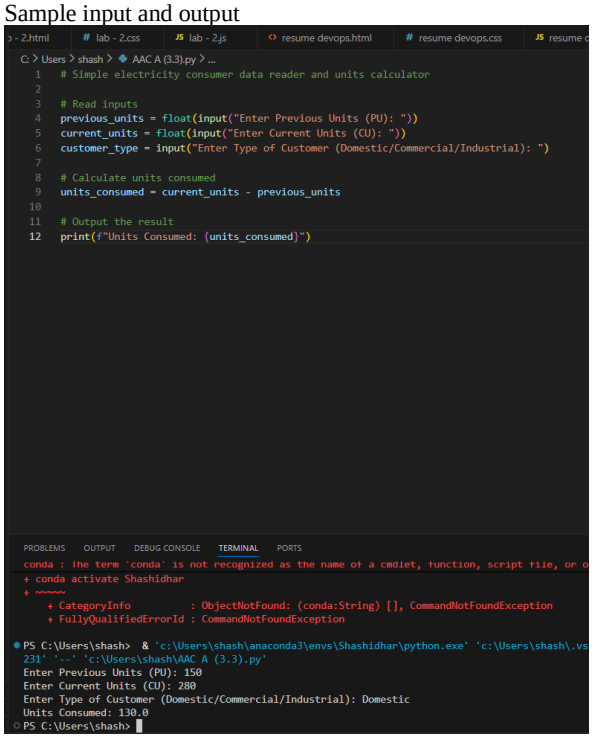
An electricity billing system must collect accurate consumer data.

Task Description

Use an AI tool (GitHub Copilot / Gemini) to generate a Python program that:

- Reads:
  - Previous Units (PU)
  - Current Units (CU)
  - Type of Customer
- Calculates units consumed
- Implements logic directly in the main program (no functions)

Expected Output

- Correct input reading
- Units consumed calculation
- Screenshot showing AI-generated code
- Sample input and output
- 

Task 2: Energy Charges Calculation Based on Units Consumed

Scenario

Energy charges depend on the number of units consumed and customer type.

Task Description

Review the AI-generated code from Task 1 and extend it to:

- Calculate **Energy Charges (EC)**
- Use conditional statements based on:
  - Domestic
  - Commercial
  - Industrial consumers
- Improve readability using AI prompts such as:

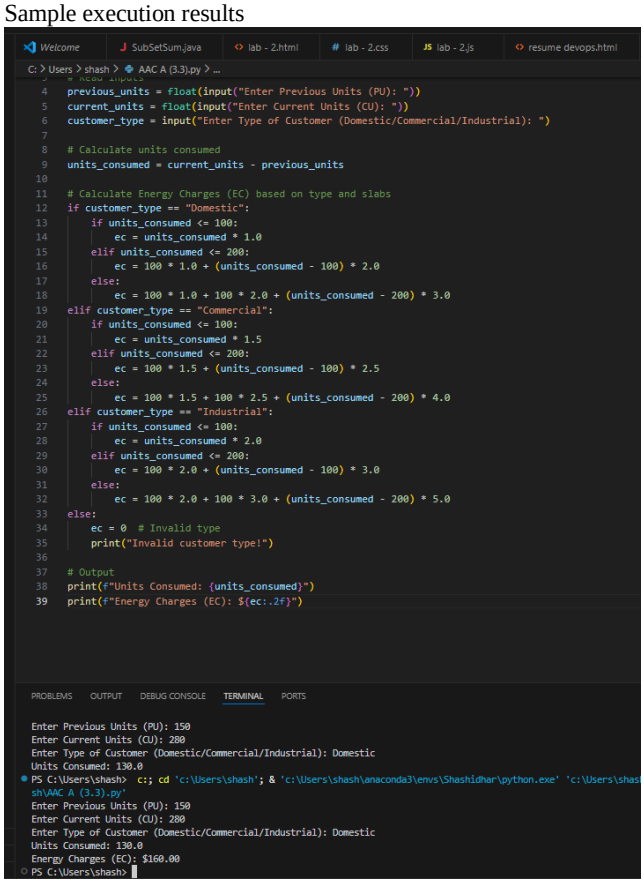
NAME:G.BHAGATH H.NO: 2303A51807 BATCH:26

ASSIGNMENT-3.3

- *"Simplify energy charge calculation logic"*
- *"Optimize conditional statements"*

Expected Output

- Correct EC calculation
- Clear conditional logic
- Original and improved versions (optional)
- Sample execution results



-

Task 3: Modular Design Using AI Assistance (Using Functions)

Scenario

Billing logic must be reusable for multiple consumers.

Task Description

Use AI assistance to generate a Python program that:

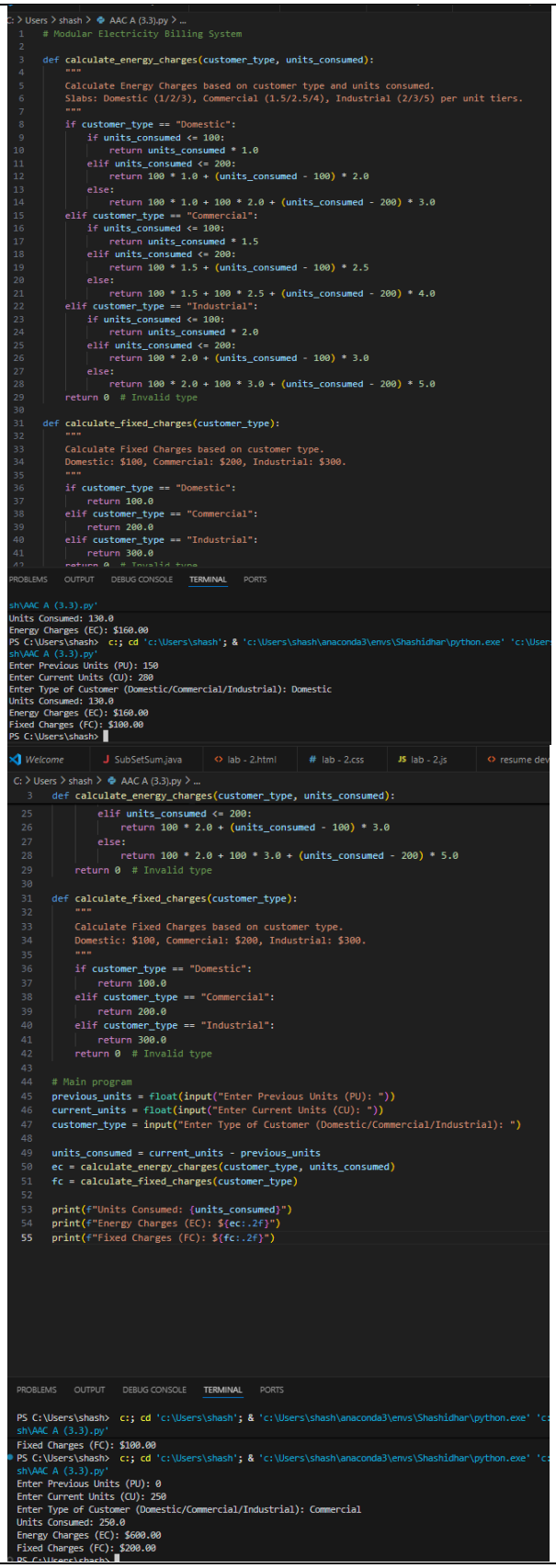
- Uses user-defined functions to:
  - Calculate Energy Charges
  - Calculate Fixed Charges
- Returns calculated values
- Includes meaningful comments

Expected Output

- Function-based Python program
- Correct EC and FC values
- Screenshots of AI-assisted function generation
- Test cases with outputs

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

ASSIGNMENT-3.3

- 

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

ASSIGNMENT-3.3

Task 4: Calculation of Additional Charges

Scenario

Electricity bills include multiple additional charges.

Task Description

Extend the program to calculate:

- **FC –** Fixed Charges
- **CC –** Customer Charges
- **ED –** Electricity Duty (percentage of EC)

Use AI prompts like:

- *"Add electricity duty calculation"*
- *"Improve billing accuracy"*

Expected Output

- Individual charge values printed
- Correct duty calculation
- Well-structured output
- Verified intermediate results

```
C: > Users > shash > AAC A (3.3).py > ...
# Extended Electricity Billing with Additional Charges

def calculate_energy_charges(customer_type, units_consumed):
    """
    Calculate Energy Charges based on customer type and units consumed.
    Slabs: Domestic (1/2/3), Commercial (1.5/2.5/4), Industrial (2/3/5) per unit tiers.
    """
    if customer_type == "Domestic":
        if units_consumed <= 100:
            return units_consumed * 1.0
        elif units_consumed <= 200:
            return 100 * 1.0 + (units_consumed - 100) * 2.0
        else:
            return 100 * 1.0 + 100 * 2.0 + (units_consumed - 200) * 3.0
    elif customer_type == "Commercial":
        if units_consumed <= 100:
            return units_consumed * 1.5
        elif units_consumed <= 200:
            return 100 * 1.5 + (units_consumed - 100) * 2.5
        else:
            return 100 * 1.5 + 100 * 2.5 + (units_consumed - 200) * 4.0
    elif customer_type == "Industrial":
        if units_consumed <= 100:
            return units_consumed * 2.0
        elif units_consumed <= 200:
            return 100 * 2.0 + (units_consumed - 100) * 3.0
        else:
            return 100 * 2.0 + 100 * 3.0 + (units_consumed - 200) * 5.0
    return 0  # Invalid type

def calculate_fixed_charges(customer_type):
    """
    Calculate Fixed Charges based on customer type.
    Domestic: $100, Commercial: $200, Industrial: $300.
    """
    if customer_type == "Domestic":
        return 100.0
    elif customer_type == "Commercial":
        return 200.0
    elif customer_type == "Industrial":
        return 300.0
    return 0  # Invalid type

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
conda : The term 'conda' is not recognized as the name of a cmdlet, function, script file, or operable progra
    + FullyQualifiedErrorId : CommandNotFoundException

PS C:\Users\shash>  & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' 'c:\Users\shash\.vscode\extension
Enter Previous Units (PU): 150
Enter Current Units (CU): 280
Enter Type of Customer (Domestic/Commercial/Industrial): Domestic
Units Consumed: 130.0
Energy Charges (EC): $160.00
Fixed Charges (FC): $100.00
Customer Charges (CC): $50.00
Electricity Duty (ED): $16.00
PS C:\Users\shash>
```

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

ASSIGNMENT-3.3

```python
def calculate_fixed_charges(customer_type):
        return 100.0
    elif customer_type == "Commercial":
        return 200.0
    elif customer_type == "Industrial":
        return 300.0
    return 0  # Invalid type

def calculate_customer_charges():
    """Fixed Customer Charges: $50 for all types."""
    return 50.0

def calculate_electricity_duty(ec):
    """Electricity Duty: 10% of Energy Charges."""
    return 0.10 * ec

# Main program
previous_units = float(input("Enter Previous Units (PU): "))
current_units = float(input("Enter Current Units (CU): "))
customer_type = input("Enter Type of Customer (Domestic/Commercial/Industrial): ").strip(

if customer_type not in ["Domestic", "Commercial", "Industrial"]:
    print("Invalid type! Defaulting to Domestic.")
    customer_type = "Domestic"

units_consumed = current_units - previous_units
ec = calculate_energy_charges(customer_type, units_consumed)
fc = calculate_fixed_charges(customer_type)
cc = calculate_customer_charges()
ed = calculate_electricity_duty(ec)

# Print individual charges
print(f"Units Consumed: {units_consumed}")
print(f"Energy Charges (EC): ${ec:.2f}")
print(f"Fixed Charges (FC): ${fc:.2f}")
print(f"Customer Charges (CC): ${cc:.2f}")
print(f"Electricity Duty (ED): ${ed:.2f}")
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
conda : The term 'conda' is not recognized as the name of a cmdlet, function, script file, or operable prog
    + FullyQualifiedErrorId : CommandNotFoundException

PS C:\Users\shash> & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' 'c:\Users\shash\.vscode\extens
Enter Previous Units (PU): 150
Enter Current Units (CU): 280
Enter Type of Customer (Domestic/Commercial/Industrial): Domestic
Units Consumed: 130.0
Energy Charges (EC): $160.00
Fixed Charges (FC): $100.00
Customer Charges (CC): $50.00
Electricity Duty (ED): $16.00
PS C:\Users\shash>
```

Task 5: Final Bill Generation and Output Analysis

Scenario

The final electricity bill must present all values clearly.

Task Description

Develop the final Python application to:

- Calculate total bill:
- Total Bill = EC + FC + CC + ED
- Display:
  - Energy Charges (EC)
  - Fixed Charges (FC)
  - Customer Charges (CC)
  - Electricity Duty (ED)
  - Total Bill Amount
- Analyze the program based on:
  - Accuracy
  - Readability
  - Real-world applicability

Expected Output

- Complete electricity bill output
- Neatly formatted display
- Sample input/output
- Short analysis paragraph

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

ASSIGNMENT-3.3

C: > Users > shash > AAC A (3.3).py > ...

```python
# Final Electricity Bill Generator

def calculate_energy_charges(customer_type, units_consumed):
    """
    Calculate Energy Charges based on customer type and units consumed.
    Slabs: Domestic (1/2/3), Commercial (1.5/2.5/4), Industrial (2/3/5) per unit tiers.
    """
    if customer_type == "Domestic":
        if units_consumed <= 100:
            return units_consumed * 1.0
        elif units_consumed <= 200:
            return 100 * 1.0 + (units_consumed - 100) * 2.0
        else:
            return 100 * 1.0 + 100 * 2.0 + (units_consumed - 200) * 3.0
    elif customer_type == "Commercial":
        if units_consumed <= 100:
            return units_consumed * 1.5
        elif units_consumed <= 200:
            return 100 * 1.5 + (units_consumed - 100) * 2.5
        else:
            return 100 * 1.5 + 100 * 2.5 + (units_consumed - 200) * 4.0
    elif customer_type == "Industrial":
        if units_consumed <= 100:
            return units_consumed * 2.0
        elif units_consumed <= 200:
            return 100 * 2.0 + (units_consumed - 100) * 3.0
        else:
            return 100 * 2.0 + 100 * 3.0 + (units_consumed - 200) * 5.0
    return 0

def calculate_fixed_charges(customer_type):
    """Fixed Charges: Domestic $100, Commercial $200, Industrial $300."""
    if customer_type == "Domestic":
        return 100.0
    elif customer_type == "Commercial":
        return 200.0
    elif customer_type == "Industrial":
        return 300.0
    return 0

def calculate_customer_charges():
    """Fixed Customer Charges: $50 """
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\shash>  c:; cd 'c:\Users\shash'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' 'c:\
Enter Previous Units (PU): 150
Enter Current Units (CU): 280
Enter Type of Customer (Domestic/Commercial/Industrial): Domestic

========================================
ELECTRICITY BILL SUMMARY
========================================
Customer Type: Domestic
Units Consumed: 130.0
Energy Charges (EC): $160.00
Fixed Charges (FC): $100.00
```

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

ASSIGNMENT-3.3

```python
def calculate_fixed_charges(customer_type):
    return 0

def calculate_customer_charges():
    """Fixed Customer Charges: $50."""
    return 50.0

def calculate_electricity_duty(ec):
    """Electricity Duty: 10% of EC."""
    return 0.10 * ec

# Main program
previous_units = float(input("Enter Previous Units (PU): "))
current_units = float(input("Enter Current Units (CU): "))
customer_type = input("Enter Type of Customer (Domestic/Commercial/Industrial): ").strip()

if customer_type not in ["Domestic", "Commercial", "Industrial"]:
    print("Invalid type! Defaulting to Domestic.")
    customer_type = "Domestic"

units_consumed = current_units - previous_units
ec = calculate_energy_charges(customer_type, units_consumed)
fc = calculate_fixed_charges(customer_type)
cc = calculate_customer_charges()
ed = calculate_electricity_duty(ec)
total_bill = ec + fc + cc + ed

# Neatly formatted bill display
print("\n" + "="*40)
print("ELECTRICITY BILL SUMMARY")
print("="*40)
print(f"Customer Type: {customer_type}")
print(f"Units Consumed: {units_consumed}")
print(f"Energy Charges (EC): ${ec:.2f}")
print(f"Fixed Charges (FC): ${fc:.2f}")
print(f"Customer Charges (CC): ${cc:.2f}")
print(f"Electricity Duty (ED): ${ed:.2f}")
print("-"*40)
print(f"TOTAL BILL AMOUNT: ${total_bill:.2f}")
print("="*40)
```

```
PS C:\Users\shash> c:; cd 'c:\Users\shash'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' 'c:\Users\sha
========================================
Customer Type: Domestic
Units Consumed: 130.0
Energy Charges (EC): $160.00
Fixed Charges (FC): $100.00
Customer Charges (CC): $50.00
Electricity Duty (ED): $16.00
----------------------------------------
TOTAL BILL AMOUNT: $326.00
========================================
```

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.