

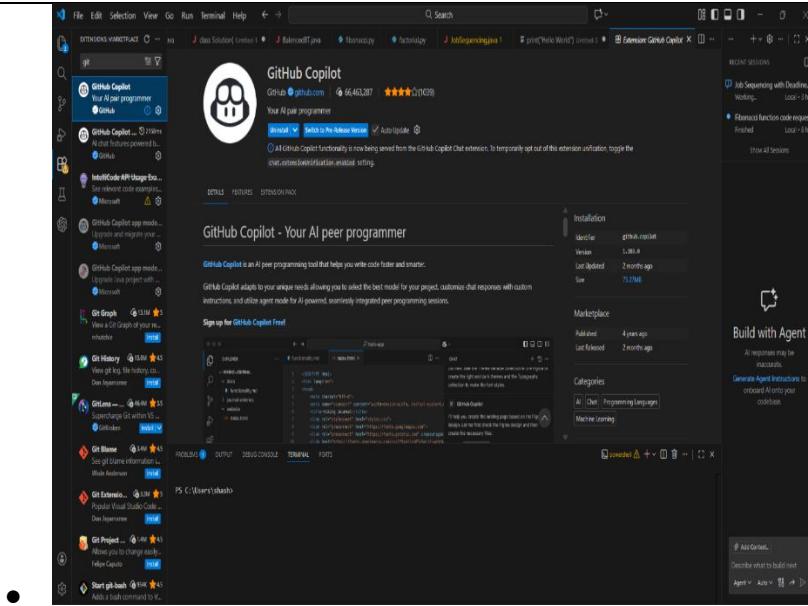
Name:G.Bhagath

H.No:2303A51807

Batch: 26

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech		Assignment Type: Lab	
Course Coordinator Name		Dr. Rishabh Mittal	
Instructor(s) Name		Mr. S Naresh Kumar Ms. B. Swathi Dr. Sasanko Shekhar Gantayat Mr. Md Sallauddin Dr. Mathivanan Mr. Y Srikanth Ms. N Shilpa Dr. Rishabh Mittal (Coordinator) Dr. R. Prashant Kumar Mr. Ankushavali MD Mr. B Viswanath Ms. 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

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



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
ed\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
925.18.0-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

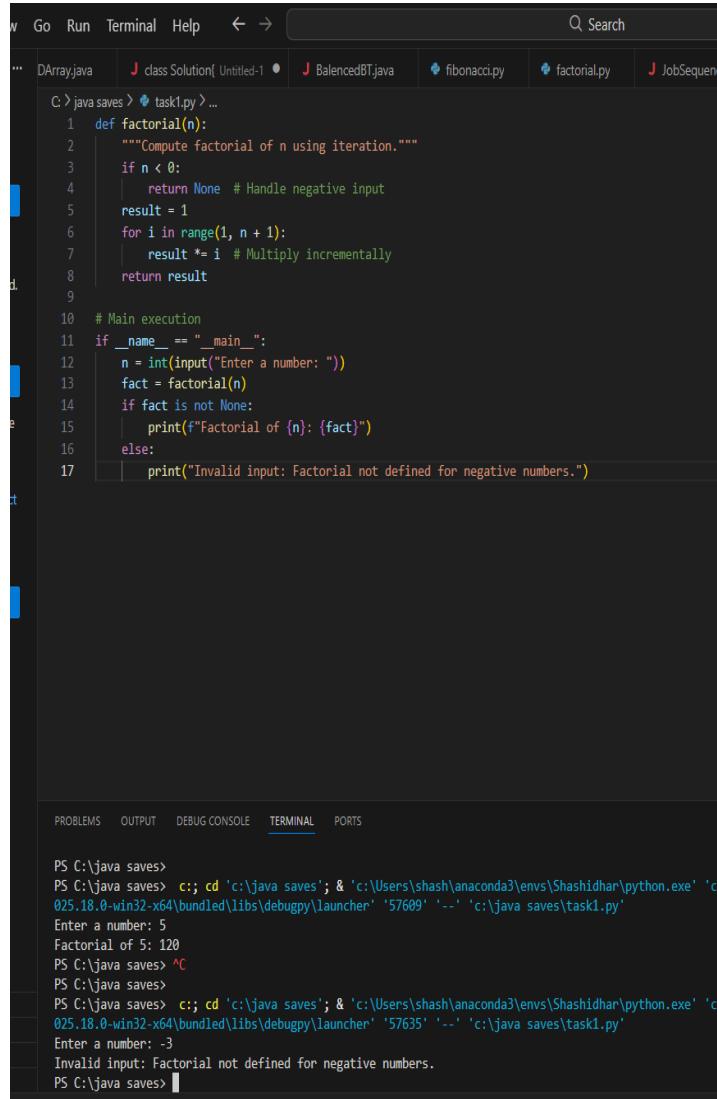
- Use meaningful function and variable names
- Include inline comments (preferably suggested by Copilot)

❖ 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> cd 'c:\java\saves'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' 'c:\java\saves\task1.py'
025.18.0-win32-x64\bundled\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> cd 'c:\java\saves'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' 'c:\java\saves\task1.py'
025.18.0-win32-x64\bundled\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

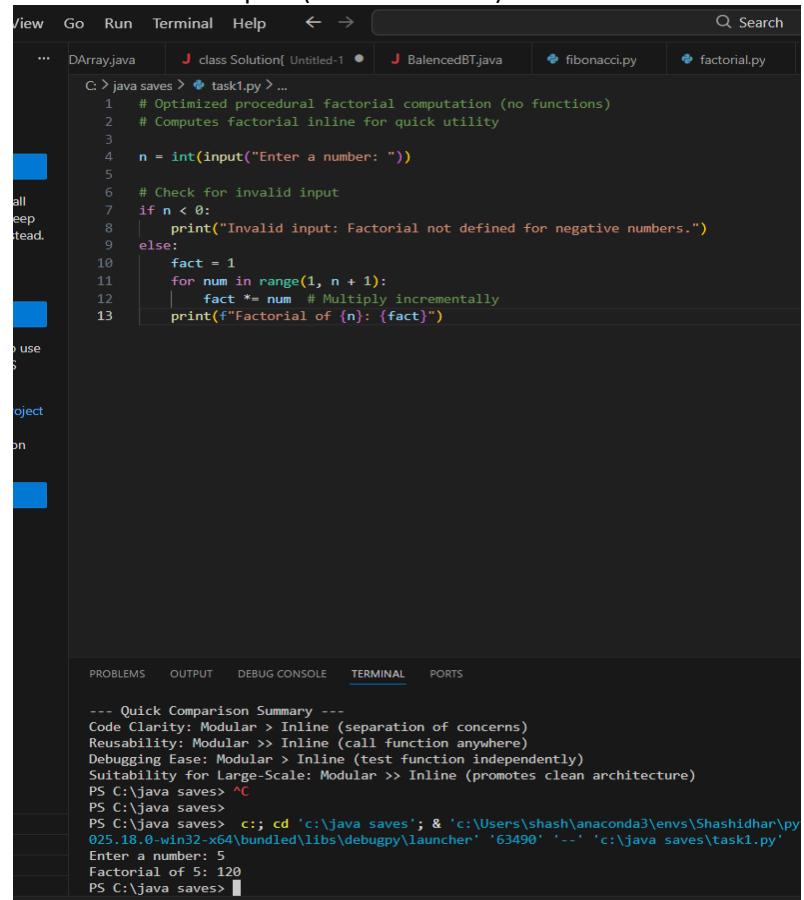
Compare the **non-function** and **function-based** Copilot-generated programs on the following criteria:

- 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).



The screenshot shows a code editor interface with a terminal tab at the bottom. The terminal tab displays a Python script named `task1.py` and its execution output.

```
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>
```

The image shows a code editor interface with a terminal window below it. The code editor has tabs for 'DArray.java', 'Solution.py', 'Untitled-1', 'BalancedBT.java', 'fibonacci.py', and 'factorial.py'. The 'factorial.py' tab is active, displaying the following Python code:

```

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.")

```

The terminal window below shows the execution of the code:

```

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\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\Shashidh
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)

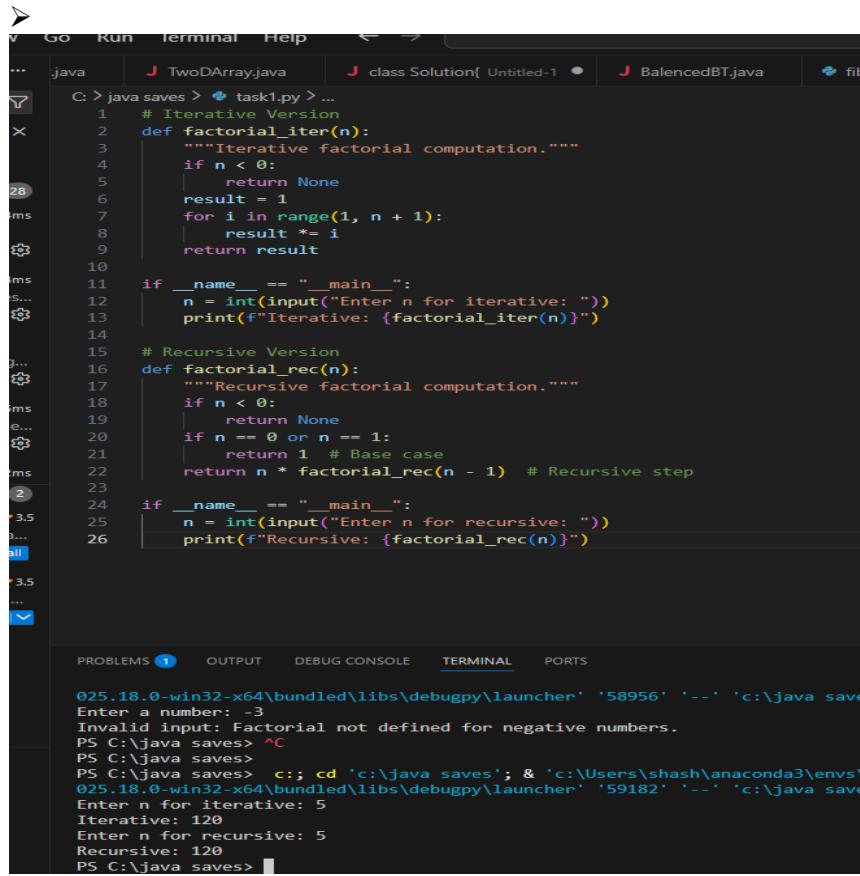
Comparison covering:

➤ Readability

➤ Stack usage

➤ Performance implications

- When recursion is *not* recommended.



Submission Requirements

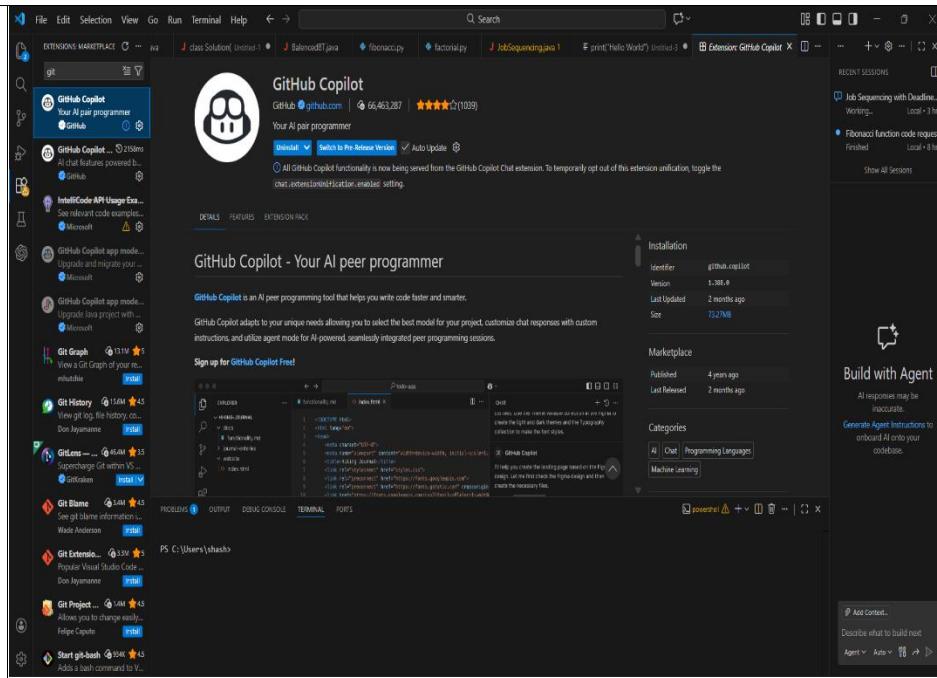
1. Generate code for each task with comments.
 2. Screenshots of Copilot suggestions.
 3. Comparative analysis reports (Task 4 and Task 5).
 4. Sample inputs/outputs demonstrating correctness.

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



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

❖ Scenario

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

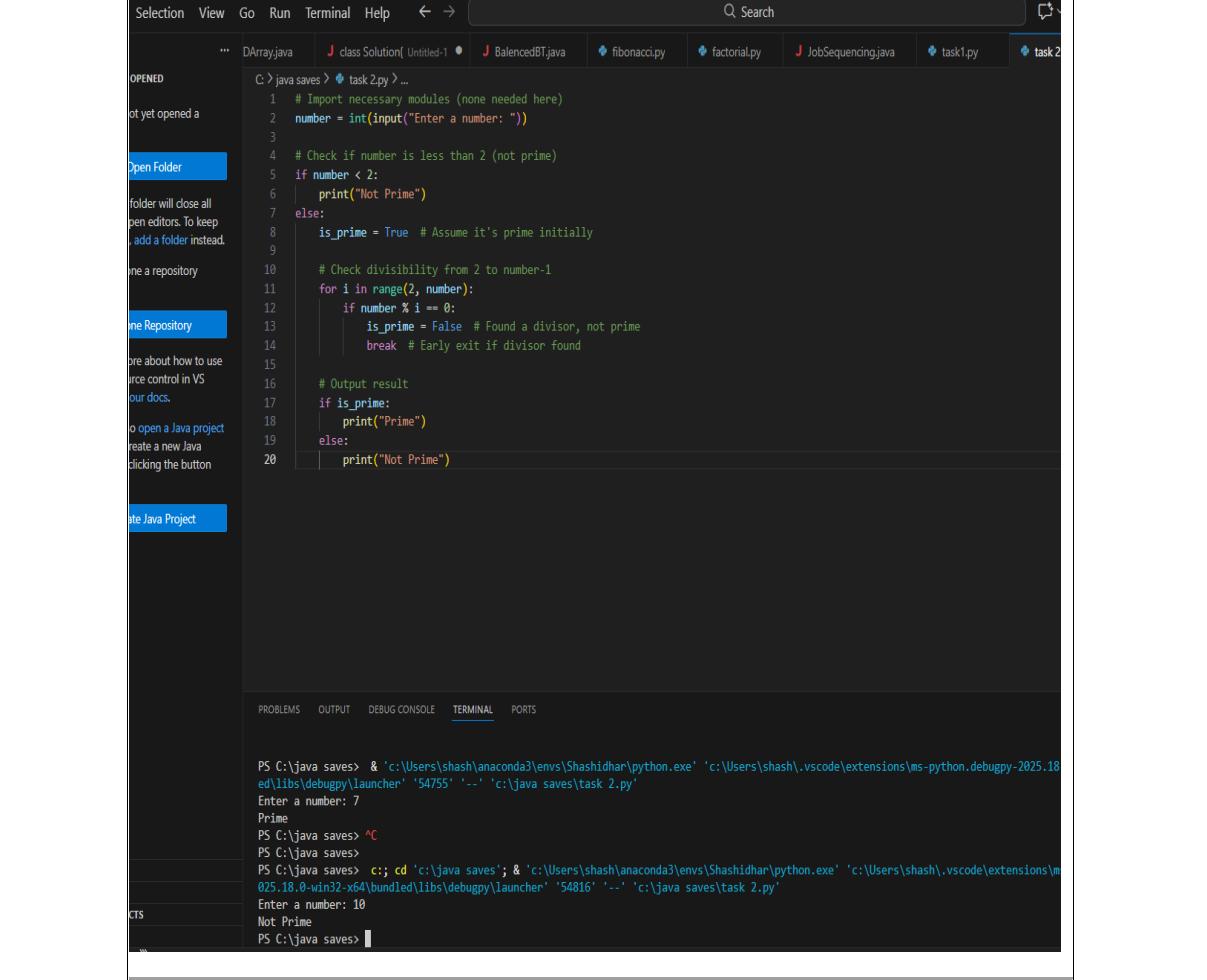
❖ Task Description

Use GitHub Copilot to generate a Python program that:

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

❖ Expected Output

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



The screenshot shows the Visual Studio Code interface with a Python script named `task 2.py` open in the editor. The script contains code to check if a number is prime. The terminal below shows the script being run and a user inputting a number to check.

```

C:\> java saves & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' 'c:\Users\shash\.vscode\extensions\ms-python.debugpy-2025.18
ed\libs\debugpy\launcher' '54755' '--' 'c:\java saves\task 2.py'
Enter a number: 7
Prime
PS C:\java saves> ^
PS C:\java saves>
PS C:\java saves> c; cd 'c:\java saves'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' 'c:\Users\shash\.vscode\extensions\m
025.18.0-win32-x64\libs\debugpy\launcher' '54816' '--' 'c:\java saves\task 2.py'
Enter a number: 10
Not Prime
PS C:\java saves>

```

Task 2: Efficiency & Logic Optimization (Cleanup)

❖ **Scenario**
The script must handle larger input values efficiently.

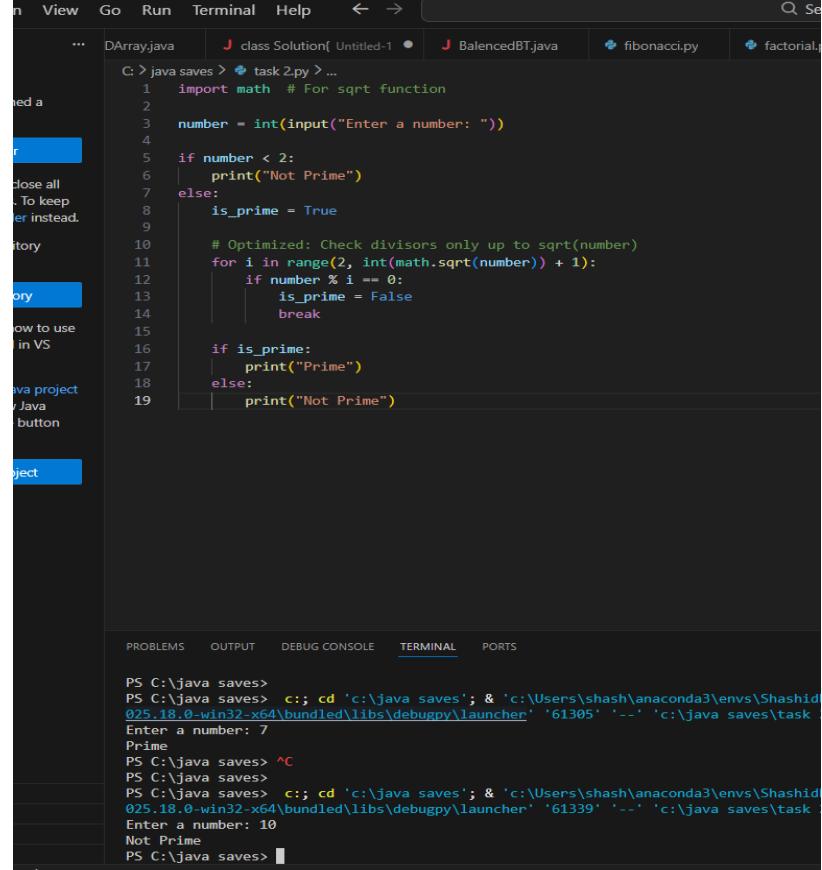
❖ **Task Description**
Review the Copilot-generated code from Task 1 and improve it by:

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

Hint:
Prompt Copilot with phrases like
“optimize this code”, *“simplify logic”*, or *“make it more readable”*

❖ **Expected Output**

- Original and optimized code versions
- Explanation of how the improvements reduce time complexity



```

1  import math # For sqrt function
2
3  number = int(input("Enter a number: "))
4
5  if number < 2:
6      print("Not Prime")
7  else:
8      is_prime = True
9
10     # Optimized: Check divisors only up to sqrt(number)
11     for i in range(2, int(math.sqrt(number)) + 1):
12         if number % i == 0:
13             is_prime = False
14             break
15
16         if is_prime:
17             print("Prime")
18         else:
19             print("Not Prime")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

PS C:\java saves>
PS C:\java saves> c;; cd 'c:\java saves'; & 'c:\Users\shash\anaconda3\envs\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.

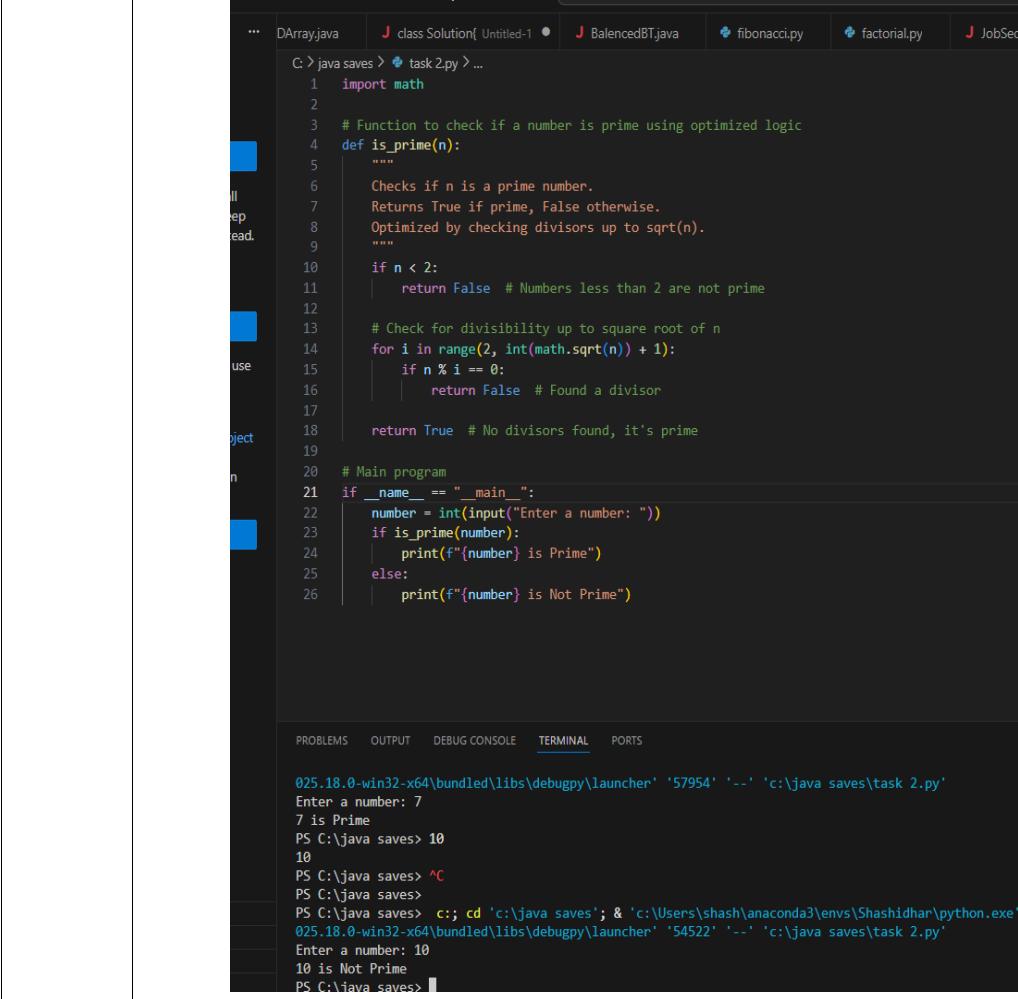
❖ Task Description

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

```

Task 4: Comparative Analysis –With vs Without Functions

❖ Scenario

You are participating in a technical review discussion.

❖ Task Description

Compare the Copilot-generated programs:

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

❖ Expected Output

Comparison table or short analytical report

The image shows a code editor interface with a terminal window below it. The code editor has tabs for 'Array.java', 'Solution', 'Untitled-1', 'BalencedBT.java', 'fibonacci.py', 'factorial.py', and 'JobSe'. The main editor area contains Python code for prime number checking, divided into two sections: 'INLINE LOGIC (NO FUNCTIONS)' and 'MODULAR WITH FUNCTIONS'. The terminal window below shows the execution of the code in a Windows environment, with the command 'java saves' and the output of the prime check for the number 997.

```
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.

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

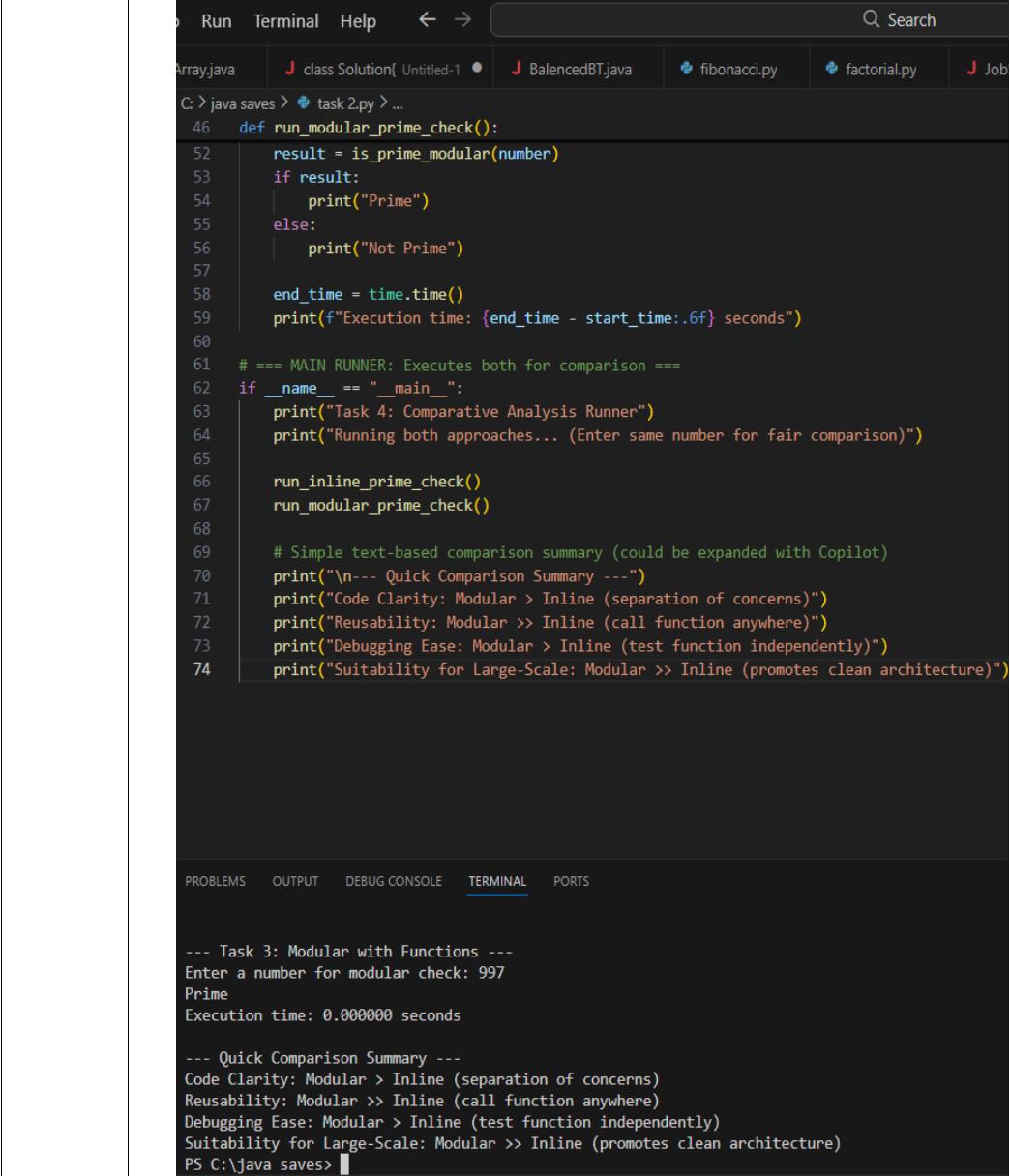
PS C:\java saves> ^C
PS C:\java saves>
PS C:\java saves> c;; cd 'c:\java saves'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' '025.18.0-win32-x64\bundled\libs\debugpy\launcher' '64514' '--' 'c:\java saves\task 2.py'
Task 4: Comparative Analysis Runner
Running both approaches... (Enter same number for fair comparison)

--- Task 1: Inline Logic (No Functions) ---
Enter a number for inline check: 997
Prime
Execution time: 0.000000 seconds
```

The screenshot shows a code editor interface with a terminal tab at the bottom. The terminal tab displays two sets of command-line interactions. The first interaction, labeled '--- Task 1: Inline Logic (No Functions) ---', shows the user entering '997' and the program outputting 'Prime' with an execution time of '0.000000 seconds'. The second interaction, labeled '--- Task 3: Modular with Functions ---', shows the user entering '997' and the program outputting 'Prime' with an execution time of '0.000000 seconds'. The code in the editor is as follows:

```
C:\> java saves > task 2.py > ...
30  # === TASK 3 APPROACH: MODULAR WITH FUNCTIONS ===
31  # This is the reusable version: Logic encapsulated in a function.
32  # Pros: Reusable, easier to test/debug. Cons: Slight overhead for tiny scripts.
33  def is_prime_modular(n):
34      """
35          Checks if n is a prime number.
36          Returns True if prime, False otherwise.
37          Optimized by checking divisors up to sqrt(n).
38      """
39      if n < 2:
40          return False
41      for i in range(2, int(math.sqrt(n)) + 1):
42          if n % i == 0:
43              return False
44      return True
45
46 def run_modular_prime_check():
47     print("\n--- Task 3: Modular with Functions ---")
48     number = int(input("Enter a number for modular check: "))
49
50     start_time = time.time()
51
52     result = is_prime_modular(number)
53     if result:
54         print("Prime")
55     else:
56         print("Not Prime")
57
58     end_time = time.time()
59     print(f"Execution time: {end_time - start_time:.6f} seconds")
60
```

Below the code editor, the terminal tab has tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The TERMINAL tab is currently selected.



The image shows a terminal window with the following content:

```

Run Terminal Help ← → Search
Array.java J class Solution( Untitled-1 ● J BalencedBT.java J fibonacci.py J 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   J class Solution{ Untitled-1  J BalencedBT.java  fibonaci.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")

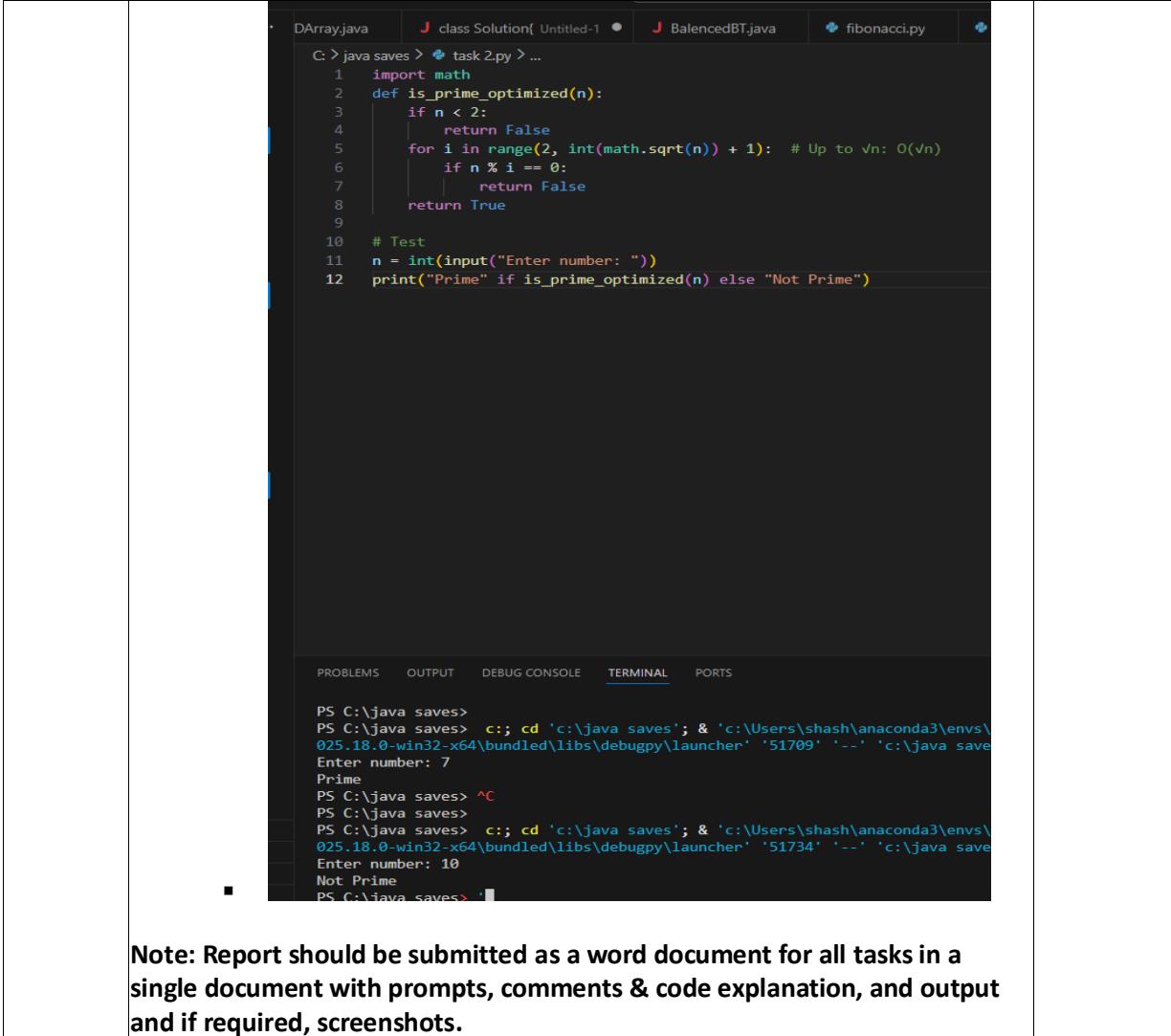
```

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 image shows a code editor interface with a terminal window below it. The code editor has several tabs: DArray.java, Solution (Untitled-1), BalancedBT.java, and fibonacci.py. The terminal window is titled 'TERMINAL' and shows the following session:

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

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

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 Gantayya 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
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 no. of assignments)		
Q.No.			Question	
1			Lab 2: Exploring Additional AI Coding Tools beyond Cursor AI and Cursor AI Lab Objectives:	

- ❖ To explore and evaluate the functionality of Google Gemini assisted coding within Google Colab.
- ❖ To understand and use Cursor AI for code generation and refactoring.
- ❖ To compare outputs and usability between Google Gemini and Cursor AI.
- ❖ To perform code optimization and documentation.

Lab Outcomes (LOs):

After completing this lab, students will be able to:

- ❖ Generate Python code using Google Gemini in Google Colab.
- ❖ Analyze the effectiveness of code explanations provided by Google Gemini.
- ❖ Set up and use Cursor AI for AI-powered coding and refactoring.
- ❖ Evaluate and refactor code using Cursor AI features.
- ❖ Compare AI tool behavior and code quality across different tools.

Task 1: Word Frequency from Text File

❖ Scenario:

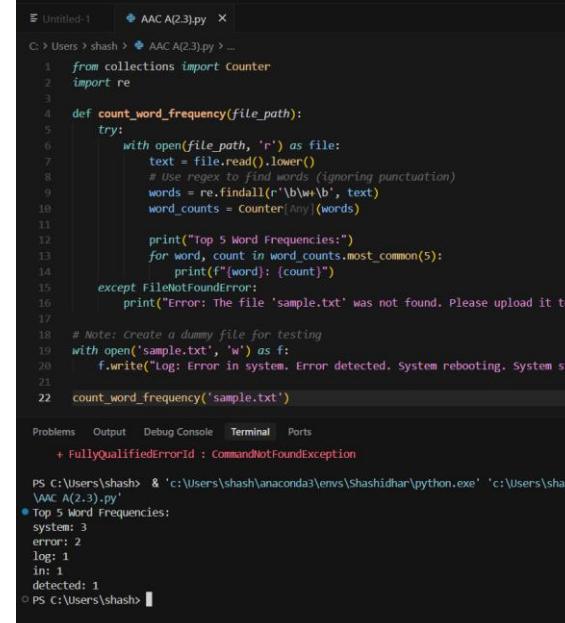
You are analyzing log files for keyword frequency.

❖ Task:

Use Gemini to generate Python code that reads a text file for word frequency, then explains the code.

❖ Expected Output:

- Working code
- Explanation
- Screenshot



The screenshot shows a Jupyter Notebook cell with the following code:

```

1  #!/usr/bin/env python3
2  # This script reads a text file and prints the top 5 word frequencies.
3  # Usage: python3 word_frequency.py <file>
4
5  import re
6  from collections import Counter
7
8  def count_word_frequency(file_path):
9      try:
10          with open(file_path, 'r') as file:
11              text = file.read().lower()
12              # Use regex to find words (ignoring punctuation)
13              words = re.findall(r'\b\w+\b', text)
14              word_counts = Counter(words)
15
16              print("Top 5 Word Frequencies:")
17              for word, count in word_counts.most_common(5):
18                  print(f"({word}): {count}")
19
20      except FileNotFoundError:
21          print("Error: The file 'sample.txt' was not found. Please upload it to the workspace and try again.")
22
23  if __name__ == "__main__":
24      count_word_frequency('sample.txt')

```

The terminal output shows the code running and printing the top 5 word frequencies from a file named 'sample.txt'.

```

PS C:\Users\shash> & 'c:\Users\shash\anaconda\envs\shashidhar\python.exe' 'c:\Users\shash\word_frequency.py'
Top 5 Word Frequencies:
error: 3
log: 2
in: 1
detected: 1

```

Task 2: File Operations Using Cursor AI

❖ **Scenario:**

You are automating basic file operations.

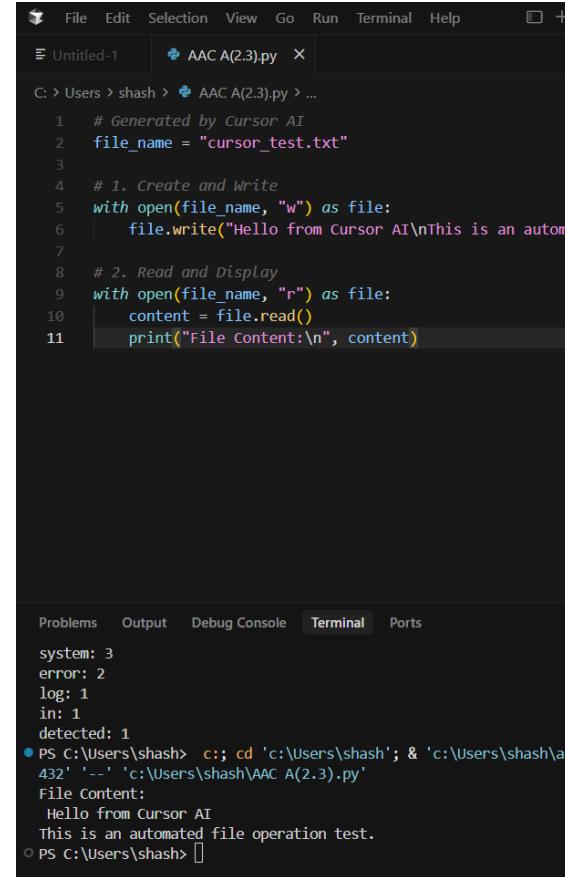
❖ **Task:**

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



The image shows a code editor and a terminal window. The code editor has a tab for 'Untitled-1' and a file named 'AAC A(2.3).py'. The Python code creates a file named 'cursor_test.txt', writes 'Hello from Cursor AI\nThis is an automated file operation test.', and then reads and prints the content. The terminal window shows the command 'PS C:\Users\shash> c;; cd 'c:\Users\shash'; & 'c:\Users\shash\432' --- 'c:\Users\shash\AAC A(2.3).py' File Content: Hello from Cursor AI This is an automated file operation test.

```
1 # Generated by Cursor AI
2 file_name = "cursor_test.txt"
3
4 # 1. Create and Write
5 with open(file_name, "w") as file:
6     file.write("Hello from Cursor AI\nThis is an automated file operation test.")
7
8 # 2. Read and Display
9 with open(file_name, "r") as file:
10     content = file.read()
11     print("File Content:\n", content)
```

Problems Output Debug Console Terminal Ports

system: 3
error: 2
log: 1
ini: 1
detected: 1
● PS C:\Users\shash> c;; cd 'c:\Users\shash'; & 'c:\Users\shash\432' --- 'c:\Users\shash\AAC A(2.3).py'
File Content:
Hello from Cursor AI
This is an automated file operation test.

Task 3: CSV Data Analysis

❖ **Scenario:**

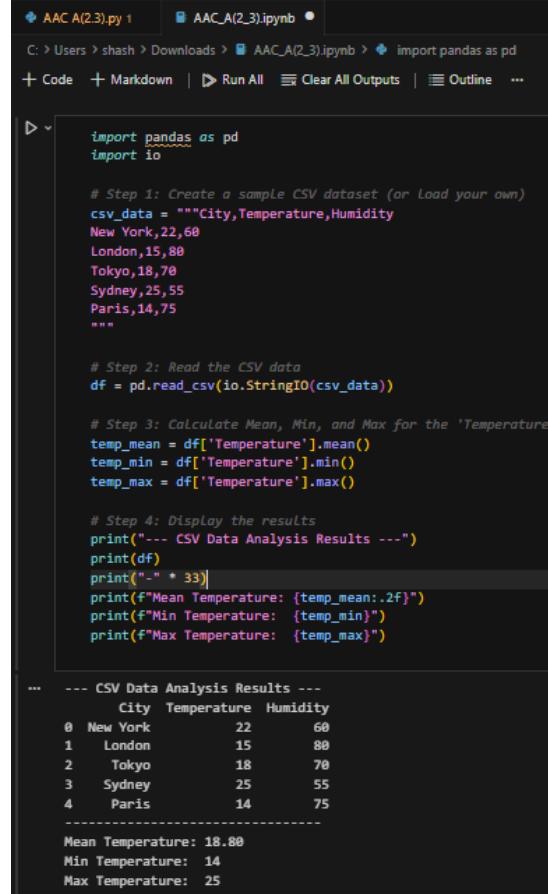
You are processing structured data from a CSV file.

❖ **Task:**

Use Gemini in Colab to read a CSV file and calculate

❖ **Expected Output:**

- Correct output
- Screenshot



The screenshot shows a Jupyter Notebook cell with the following code:

```
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.

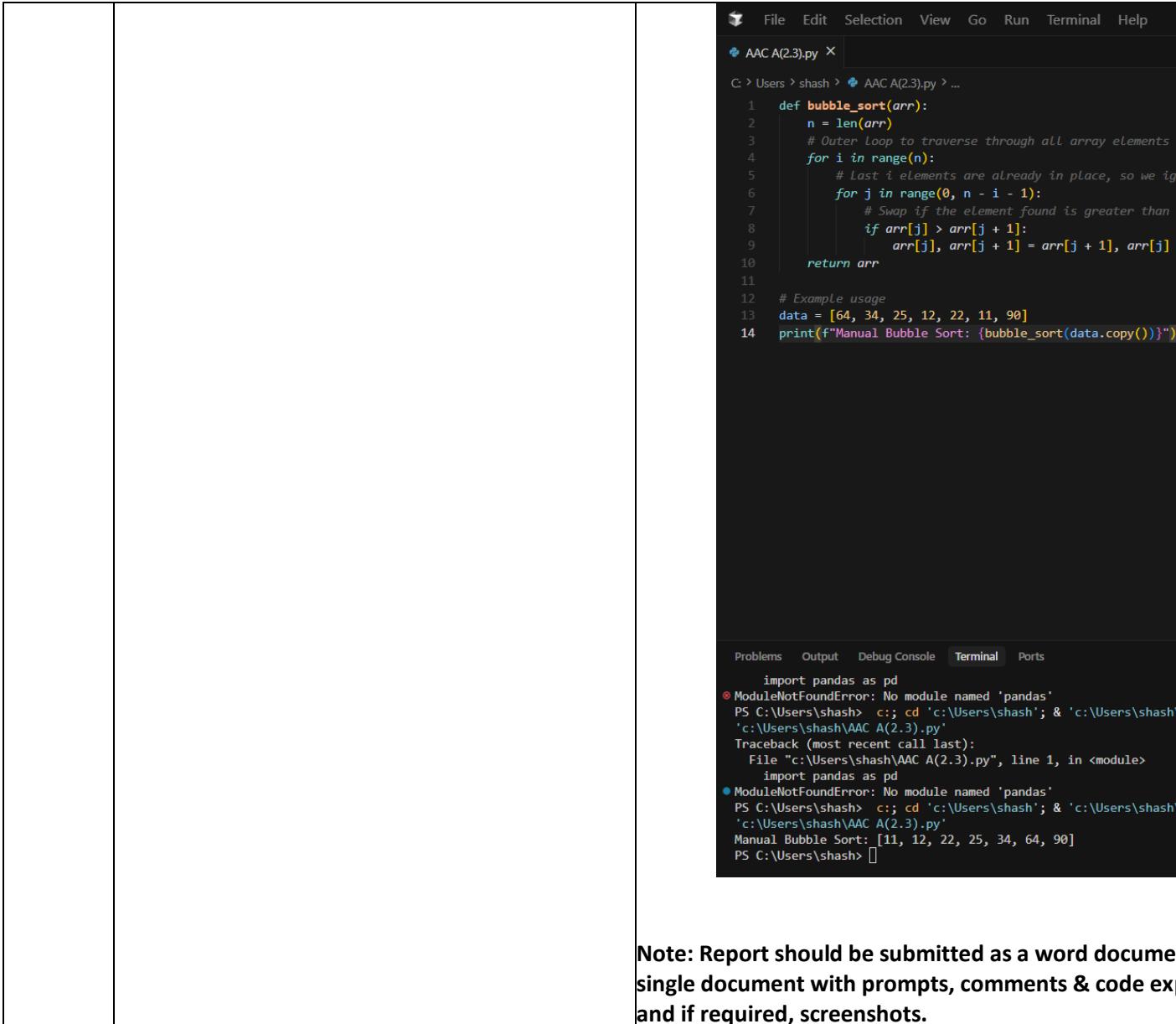
❖ **Task:**

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 X
C > Users > shash > AAC A(2.3).py > ...
1 def bubble_sort(arr):
2     n = len(arr)
3     # Outer Loop to traverse through all array elements
4     for i in range(n):
5         # Last i elements are already in place, so we ignore them
6         for j in range(0, n - i - 1):
7             # Swap if the element found is greater than the next element
8             if arr[j] > arr[j + 1]:
9                 arr[j], arr[j + 1] = arr[j + 1], arr[j]
10
11
12 # Example usage
13 data = [64, 34, 25, 12, 22, 11, 90]
14 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\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\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 document. The report should be a single document with prompts, comments & code examples 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	
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 $N = 7 \rightarrow 0 1 1 2 3 5 8$
- Observation on how Copilot understood the instruction with zero context.

```
C: > Users > shash > AAC A(3.4).py > ...
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\Shash' & python AAC A(3.4).py
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\Shash' & python AAC A(3.4).py
● PS C:\Users\shash> 7
● PS C:\Users\shash> c;; cd 'c:\Users\shash'; & 'c:\Users\shash\anaconda3\envs\Shash' & python AAC A(3.4).py
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\envs\py37\python' AAC A(3.4).py
-----
TOTAL BILL AMOUNT: $326.00
=====
● PS C:\Users\shash> c;; cd 'c:\Users\shash'; & 'c:\Users\shash\anaconda3\envs\py37\python' AAC A(3.4).py
● PS C:\Users\shash> 7
7
● PS C:\Users\shash> c;; cd 'c:\Users\shash'; & 'c:\Users\shash\anaconda3\envs\py37\python' 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\py37\python' AAC A(3.4).py
[3, 2, 1]
○ PS C:\Users\shash> []
```

Task 3: Few-shot Prompt – String Pattern Matching

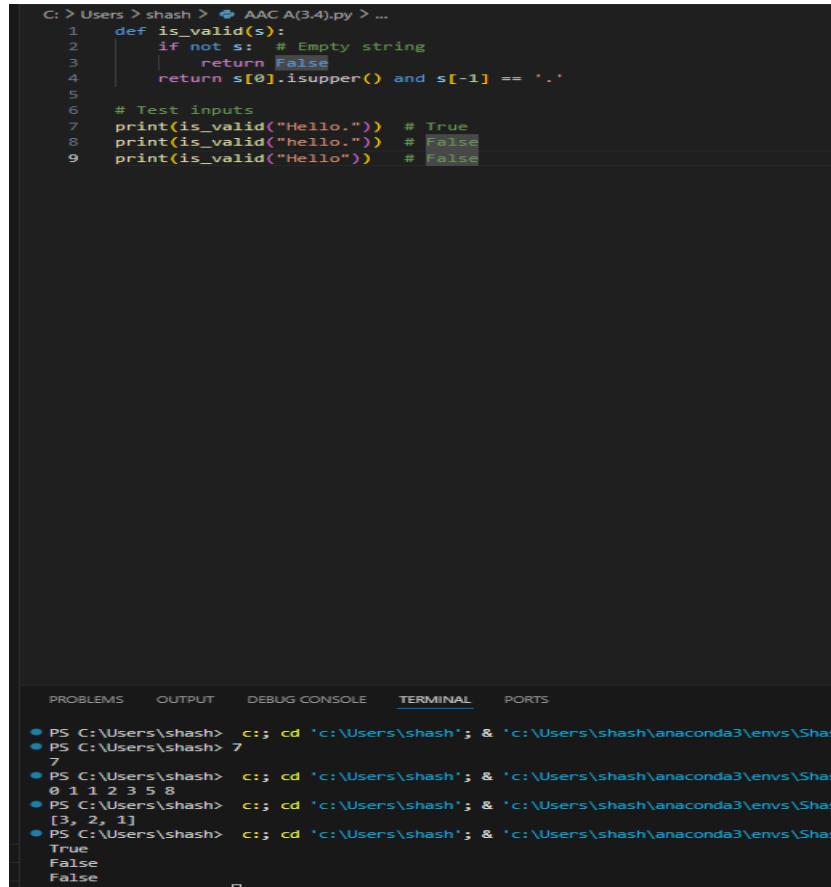
Task Description #3

- 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."))
8  print(is_valid("hello."))
9  print(is_valid("Hello"))

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

● PS C:\Users\shash> c:; cd 'c:\Users\shash'; & 'c:\Users\shash\anaconda3\envs\Shash\Scripts\python.exe' AAC A(3.4).py
7
● PS C:\Users\shash> c:; cd 'c:\Users\shash'; & 'c:\Users\shash\anaconda3\envs\Shash\Scripts\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\Shash\Scripts\python.exe' AAC A(3.4).py
[3, 2, 1]
● PS C:\Users\shash> c:; cd 'c:\Users\shash'; & 'c:\Users\shash\anaconda3\envs\Shash\Scripts\python.exe' AAC A(3.4).py
True
False
False

```

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

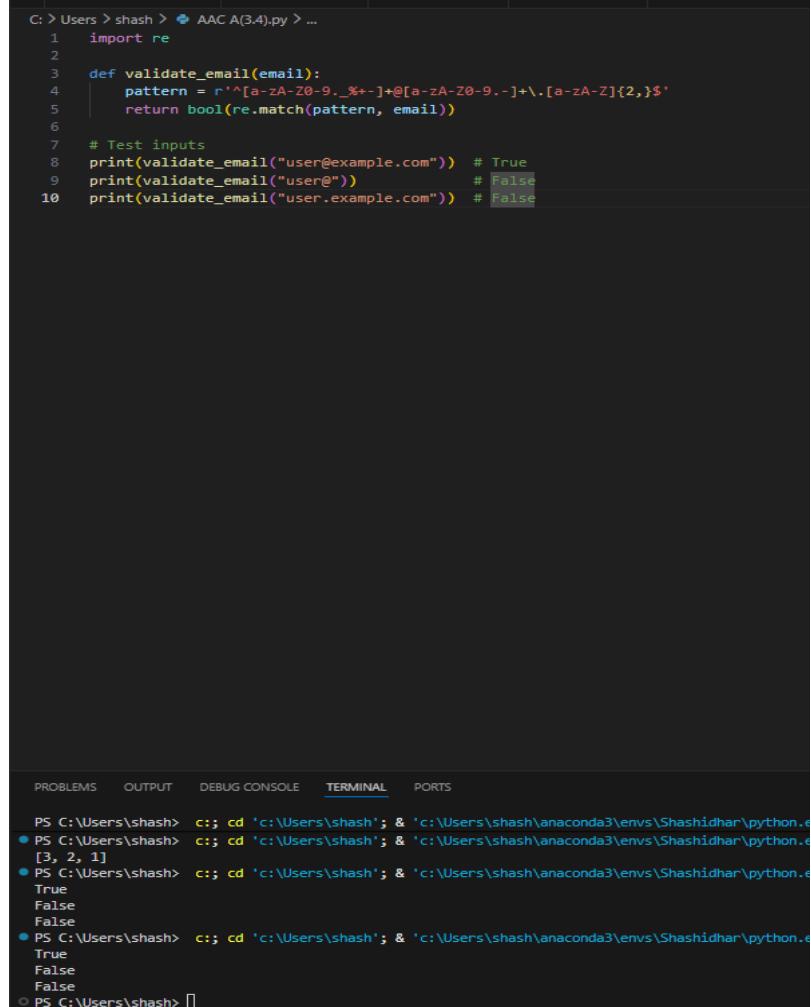
- Compare both outputs:

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.exe' AAC A(3.4).py
● PS C:\Users\shash> c:; cd 'c:\Users\shash'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' [3, 2, 1]
● PS C:\Users\shash> c:; cd 'c:\Users\shash'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' True
True
False
False
● PS C:\Users\shash> c:; cd 'c:\Users\shash'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' False
True
False
False
● PS C:\Users\shash> c:; cd 'c:\Users\shash'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' []

```

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.

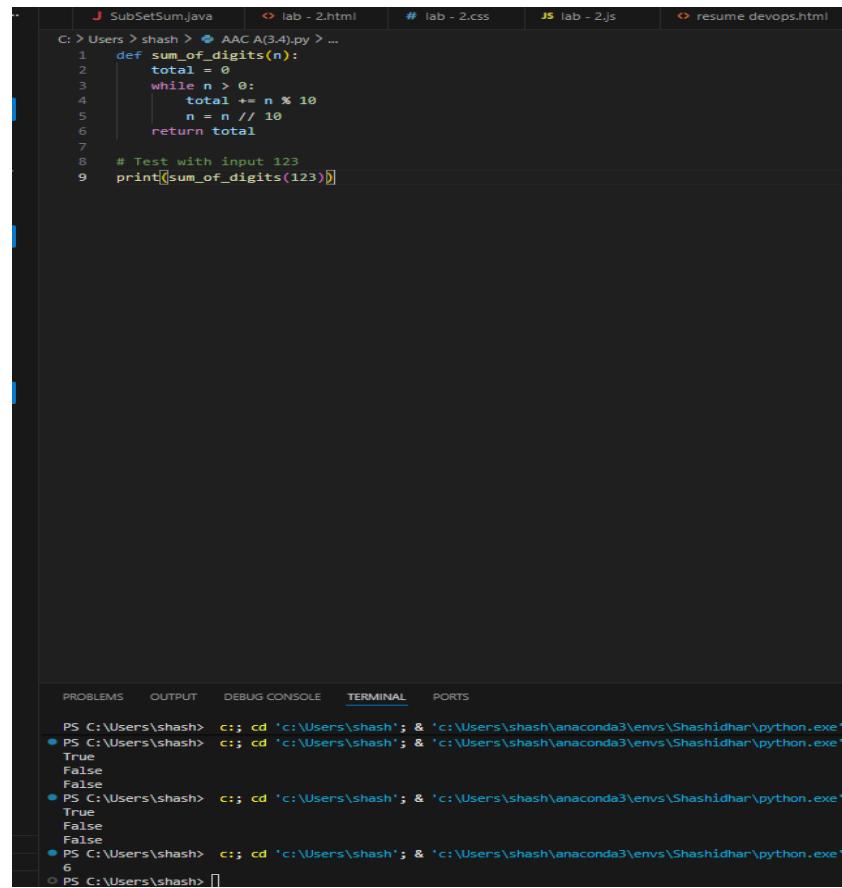
Style 1: Generic task prompt

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?



The screenshot shows a terminal window within a code editor interface. The terminal tab is active, displaying the following Python code:

```
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))
```

Below the code, the terminal shows the execution of the script and its output:

```
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

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	
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	<p>Lab 3: Application for TGNPDCL – Electricity Bill Generation Using Python & AI Tools</p> <p>Lab Objectives</p> <ul style="list-style-type: none"> • 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 <p>Lab Outcomes (LOs) After completing this lab, students will be able to:</p>	Week2 - Wednesday

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.

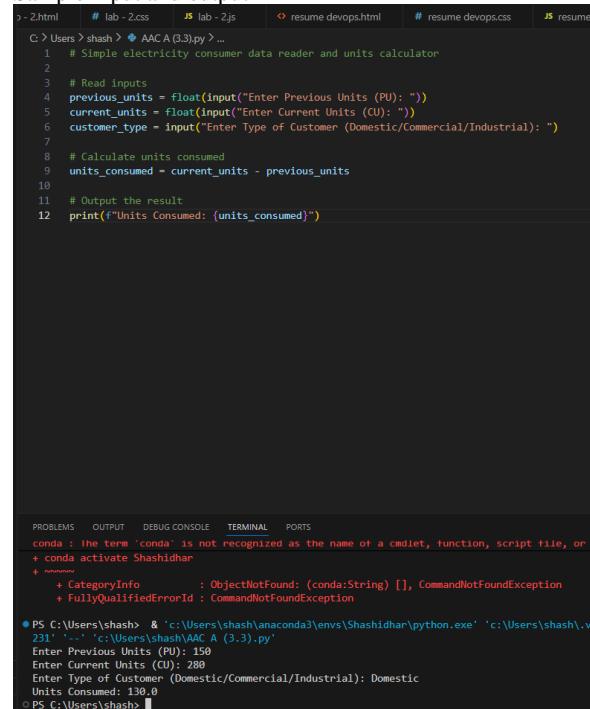
Task Description

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



```

1 # Simple electricity consumer data reader and units calculator
2
3 # Read inputs
4 previous_units = float(input("Enter Previous Units (PU): "))
5 current_units = float(input("Enter Current Units (CU): "))
6 customer_type = input("Enter Type of Customer (Domestic/Commercial/Industrial): ")
7
8 # Calculate units consumed
9 units_consumed = current_units - previous_units
10
11 # Output the result
12 print(f"Units Consumed: {units_consumed}")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

conda : The term 'conda' is not recognized as the name of a cmdlet, function, script file, or oper
+ conda activate Shashidhar
+ ~~~~~
+ CategoryInfo          : ObjectNotFound: (conda:String) [], CommandNotFoundException
+ FullyQualifiedErrorId : CommandNotFoundException

PS C:\Users\shash> & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' 'c:\Users\shash\vs
231' 'c:\Users\shash\AMC A (3.3).py'
Enter Previous Units (PU): 150
Enter Current Units (CU): 280
Enter Type of Customer (Domestic/Commercial/Industrial): Domestic
Units Consumed: 130.0

```

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:

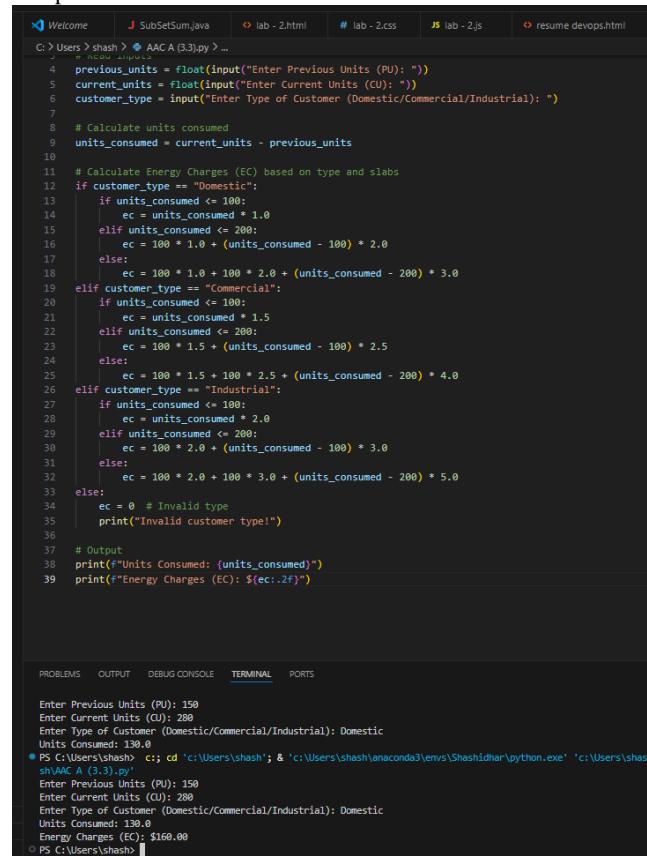
- Calculate **Energy Charges (EC)**
- Use conditional statements based on:
 - Domestic
 - Commercial
 - Industrial consumers
- Improve readability using AI prompts such as:

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



```

1  Welcome          J SubSetSum.java    O lab - 2.html    # lab - 2.css    JS lab - 2.js    O resume devops.html
2  AAC A (3.3).py
3
4  previous_units = float(input("Enter Previous Units (PU): "))
5  current_units = float(input("Enter Current Units (CU): "))
6  customer_type = input("Enter Type of Customer (Domestic/Commercial/Industrial): ")
7
8  # Calculate units consumed
9  units_consumed = current_units - previous_units
10
11 # Calculate Energy Charges (EC) based on type and slabs
12 if customer_type == "Domestic":
13     if units_consumed <= 100:
14         ec = units_consumed * 1.0
15     elif units_consumed <= 200:
16         ec = 100 * 1.0 + (units_consumed - 100) * 2.0
17     else:
18         ec = 100 * 1.0 + 100 * 2.0 + (units_consumed - 200) * 3.0
19 elif customer_type == "Commercial":
20     if units_consumed <= 100:
21         ec = units_consumed * 1.5
22     elif units_consumed <= 200:
23         ec = 100 * 1.5 + (units_consumed - 100) * 2.5
24     else:
25         ec = 100 * 1.5 + 100 * 2.5 + (units_consumed - 200) * 4.0
26 elif customer_type == "Industrial":
27     if units_consumed <= 100:
28         ec = units_consumed * 2.0
29     elif units_consumed <= 200:
30         ec = 100 * 2.0 + (units_consumed - 100) * 3.0
31     else:
32         ec = 100 * 2.0 + 100 * 3.0 + (units_consumed - 200) * 5.0
33 else:
34     ec = 0 # Invalid type
35     print("Invalid customer type!")
36
37 # Output
38 print("Units Consumed: {units_consumed}")
39 print("Energy Charges (EC): ${ec:.2f}")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

Enter Previous Units (PU): 150
Enter Current Units (CU): 200
Enter Type of Customer (Domestic/Commercial/Industrial): Domestic
Units Consumed: 150.0
PS C:\Users\shash> c:\; cd 'c:\Users\shash'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' 'c:\Users\shash\AAC A (3.3).py'
Enter Previous Units (PU): 150
Enter Current Units (CU): 200
Enter Type of Customer (Domestic/Commercial/Industrial): Domestic
Units Consumed: 150.0
Energy Charges (EC): $100.00
PS C:\Users\shash>

```

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:

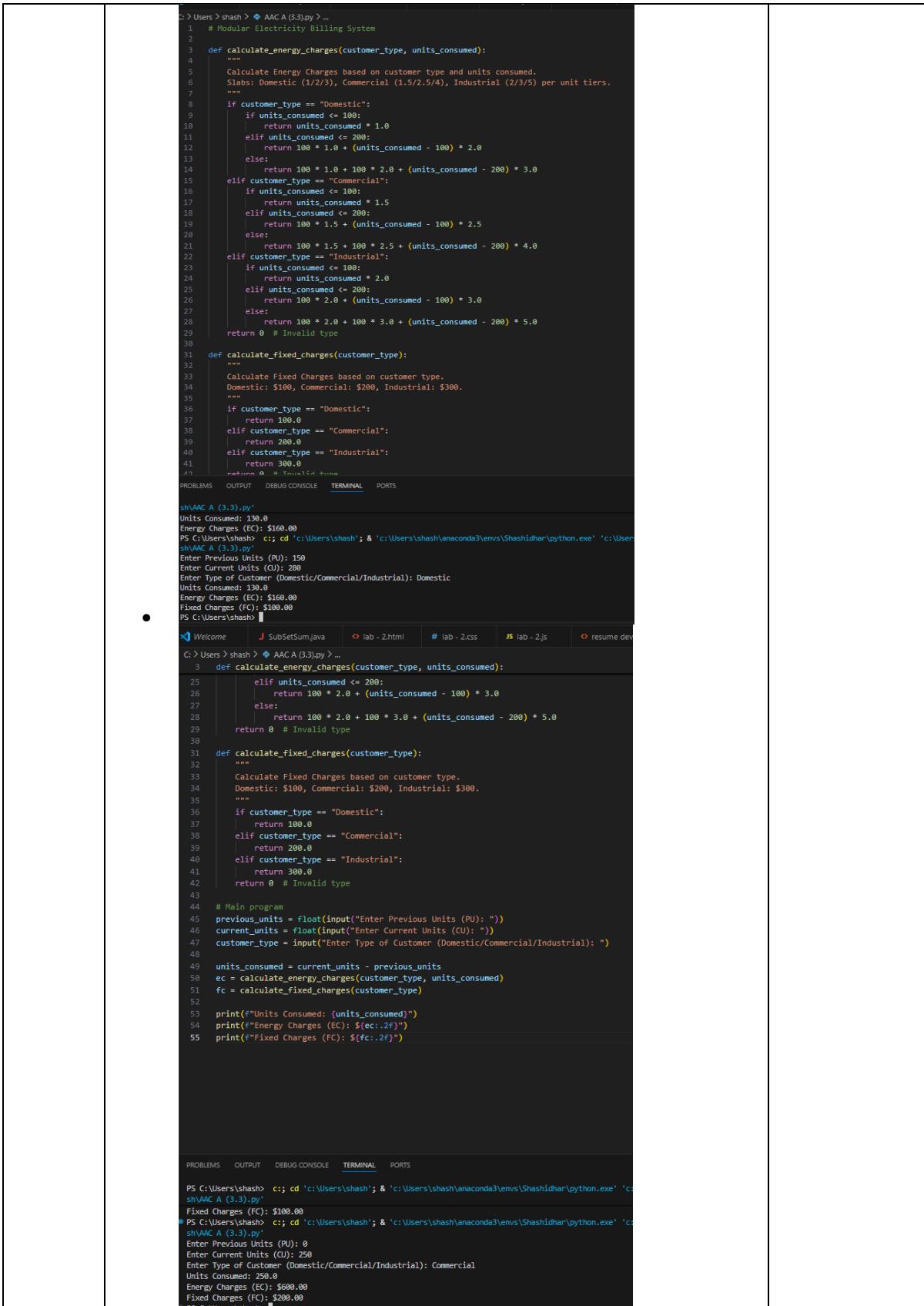
- Uses user-defined functions to:
 - Calculate Energy Charges
 - Calculate Fixed Charges
- Returns calculated values
- Includes meaningful comments

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



```
C:\> Users > shash > AAC A (3.3).py > ...
 1  # Modular Electricity Billing System
 2
 3  def calculate_energy_charges(customer_type, units_consumed):
 4      """
 5          Calculate Energy Charges based on customer type and units consumed.
 6          Slabs: Domestic (1/2/3), Commercial (1.5/2.5/4), Industrial (2/3/5) per unit tiers.
 7      """
 8      if customer_type == "Domestic":
 9          if units_consumed <= 100:
10              return units_consumed * 1.0
11          elif units_consumed <= 200:
12              return 100 * 1.0 + (units_consumed - 100) * 2.0
13          else:
14              return 100 * 1.0 + 100 * 2.0 + (units_consumed - 200) * 3.0
15      elif customer_type == "Commercial":
16          if units_consumed <= 100:
17              return units_consumed * 1.5
18          elif units_consumed <= 200:
19              return 100 * 1.5 + (units_consumed - 100) * 2.5
20          else:
21              return 100 * 1.5 + 100 * 2.5 + (units_consumed - 200) * 4.0
22      elif customer_type == "Industrial":
23          if units_consumed <= 100:
24              return units_consumed * 2.0
25          elif units_consumed <= 200:
26              return 100 * 2.0 + (units_consumed - 100) * 3.0
27          else:
28              return 100 * 2.0 + 100 * 3.0 + (units_consumed - 200) * 5.0
29
30      return 0 # Invalid type
31
32  def calculate_fixed_charges(customer_type):
33      """
34          Calculate Fixed Charges based on customer type.
35          Domestic: $100, Commercial: $200, Industrial: $300.
36      """
37      if customer_type == "Domestic":
38          return 100.0
39      elif customer_type == "Commercial":
40          return 200.0
41      elif customer_type == "Industrial":
42          return 300.0
43      return 0 # Invalid type
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
99
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
shVAC A (3.3).py'
Units Consumed: 130.0
Energy Charges (EC): $160.00
PS C:\Users\shash> c; cd 'c:\Users\shash'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' 'c:\Users\shash\shVAC A (3.3).py'
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
PS C:\Users\shash> 
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
C:\> Users > shash > AAC A (3.3).py > ...
 1  # Modular Electricity Billing System
 2
 3  def calculate_energy_charges(customer_type, units_consumed):
 4      """
 5          Calculate Energy Charges based on customer type.
 6          Domestic: $100, Commercial: $200, Industrial: $300.
 7      """
 8      if customer_type == "Domestic":
 9          if units_consumed <= 100:
10              return 100 * 1.0 + (units_consumed - 100) * 2.0
11          elif units_consumed <= 200:
12              return 100 * 1.0 + 100 * 2.0 + (units_consumed - 200) * 3.0
13          else:
14              return 100 * 1.0 + 100 * 2.0 + 100 * 3.0 + (units_consumed - 200) * 4.0
15      elif customer_type == "Commercial":
16          if units_consumed <= 100:
17              return 100 * 1.5 + (units_consumed - 100) * 2.5
18          elif units_consumed <= 200:
19              return 100 * 1.5 + 100 * 2.5 + (units_consumed - 200) * 4.0
20          else:
21              return 100 * 1.5 + 100 * 2.5 + 100 * 4.0 + (units_consumed - 200) * 5.0
22      elif customer_type == "Industrial":
23          if units_consumed <= 100:
24              return 100 * 2.0 + (units_consumed - 100) * 3.0
25          elif units_consumed <= 200:
26              return 100 * 2.0 + 100 * 3.0 + (units_consumed - 200) * 4.0
27          else:
28              return 100 * 2.0 + 100 * 3.0 + 100 * 4.0 + (units_consumed - 200) * 5.0
29
30      return 0 # Invalid type
31
32  def calculate_fixed_charges(customer_type):
33      """
34          Calculate Fixed Charges based on customer type.
35          Domestic: $100, Commercial: $200, Industrial: $300.
36      """
37      if customer_type == "Domestic":
38          return 100.0
39      elif customer_type == "Commercial":
40          return 200.0
41      elif customer_type == "Industrial":
42          return 300.0
43      return 0 # Invalid type
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
79
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\shash> c; cd 'c:\Users\shash'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' 'c:\Users\shash\shVAC A (3.3).py'
Fixed Charges (FC): $100.00
PS C:\Users\shash> c; cd 'c:\Users\shash'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' 'c:\Users\shash\shVAC A (3.3).py'
Enter Previous Units (PU): 0
Enter Current Units (CU): 250
Enter Type of Customer (Domestic/Commercial/Industrial): Commercial
Units Consumed: 250.0
Energy Charges (EC): $600.00
Fixed Charges (FC): $200.00
PS C:\Users\shash> 
```

ASSIGNMENT-3.3

Task 4: Calculation of Additional Charges**Scenario**

Electricity bills include multiple additional charges.

Task Description

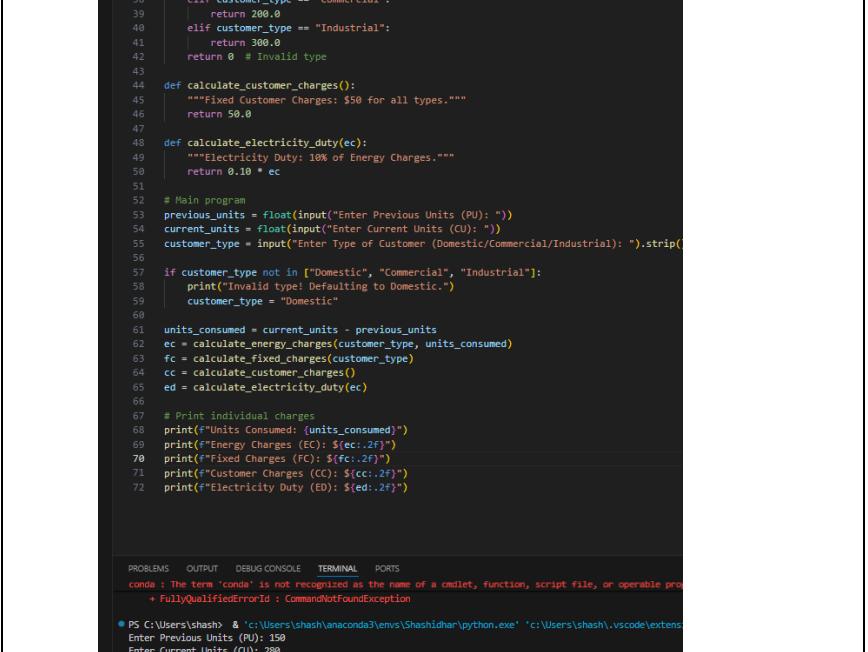
Extend the program to calculate:

- **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 ...
1  # Extended Electricity Billing with Additional Charges
2
3  def calculate_energy_charges(customer_type, units_consumed):
4      """
5          Calculate Energy Charges based on customer type and units consumed.
6          Slabs: Domestic (1/2/3), Commercial (1.5/2.5/4), Industrial (2/3/5) per unit tiers.
7      """
8      if customer_type == "Domestic":
9          if units_consumed <= 100:
10              return units_consumed * 1.0
11          elif units_consumed <= 200:
12              return 100 * 1.0 + (units_consumed - 100) * 2.0
13          else:
14              return 100 * 1.0 + 100 * 2.0 + (units_consumed - 200) * 3.0
15      elif customer_type == "Commercial":
16          if units_consumed <= 100:
17              return units_consumed * 1.5
18          elif units_consumed <= 200:
19              return 100 * 1.5 + (units_consumed - 100) * 2.5
20          else:
21              return 100 * 1.5 + 100 * 2.5 + (units_consumed - 200) * 4.0
22      elif customer_type == "Industrial":
23          if units_consumed <= 100:
24              return units_consumed * 2.0
25          elif units_consumed <= 200:
26              return 100 * 2.0 + (units_consumed - 100) * 3.0
27          else:
28              return 100 * 2.0 + 100 * 3.0 + (units_consumed - 200) * 5.0
29      return 0 # Invalid type
30
31  def calculate_fixed_charges(customer_type):
32      """
33          Calculate Fixed Charges based on customer type.
34          Domestic: $100, Commercial: $200, Industrial: $300.
35      """
36      if customer_type == "Domestic":
37          return 100.0
38      elif customer_type == "Commercial":
39          return 200.0
40      elif customer_type == "Industrial":
41          return 300.0
42      return 0 # Invalid type
43
44
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
conda : The term 'conda' is not recognized as the name of a cmdlet, function, script file, or operable program
+ FullyQualifiedErrorId : CommandNotFound
Exception
● 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): 200
Enter Type of Customer (Domestic/Commercial/Industrial): Domestic
Units Consumed: 150.0
Energy Charges (EC): $150.00
Fixed Charges (FC): $100.00
Customer Charges (CC): $50.00
Electricity Duty (ED): $16.00
○ PS C:\Users\shash>
```



```
PS C:\Users\shashu> & 'C:\Users\shashu\anaconda3\envs\shashidhar\python.exe' 'c:\Users\shashu\vscode\extensions\ms-vscode\python\python.exe' 'C:\Users\shashu\PycharmProjects\Python\lab\lab-2.py'
C: > Users > shashu > AAC A (3.3).py > ...
31 def calculate_fixed_charges(customer_type):
32     if customer_type == "Residential":
33         return 100.0
34     elif customer_type == "Commercial":
35         return 200.0
36     elif customer_type == "Industrial":
37         return 300.0
38     return 0 # Invalid type
39
40 def calculate_customer_charges():
41     """Fixed Customer Charges: $50 for all types."""
42     return 50.0
43
44 def calculate_electricity_duty(ec):
45     """Electricity Duty: 10% of Energy Charges."""
46     return 0.10 * ec
47
48 # Main program
49 previous_units = float(input("Enter Previous Units (PU): "))
50 current_units = float(input("Enter Current Units (CU): "))
51 customer_type = input("Enter Type of Customer (Domestic/Commercial/Industrial): ").strip()
52
53 if customer_type not in ["Domestic", "Commercial", "Industrial"]:
54     print("Invalid type! Defaulting to Domestic.")
55     customer_type = "Domestic"
56
57 units_consumed = current_units - previous_units
58 ec = calculate_energy_charges(customer_type, units_consumed)
59 fc = calculate_fixed_charges(customer_type)
60 cc = calculate_customer_charges()
61 ed = calculate_electricity_duty(ec)
62
63 # Print individual charges
64 print(f"Units Consumed: {units_consumed}")
65 print(f"Energy Charges (EC): ${ec:.2f}")
66 print(f"Fixed Charges (FC): ${fc:.2f}")
67 print(f"Customer Charges (CC): ${cc:.2f}")
68 print(f"Electricity Duty (ED): ${ed:.2f}")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
console : The term 'conda' is not recognized as the name of a cmdlet, function, script file, or operable program.
+ FullyQualifiedCmdletName : CommandNotFoundException
PS C:\Users\shashu> & 'C:\Users\shashu\anaconda3\envs\shashidhar\python.exe' 'c:\Users\shashu\vscode\extensions\ms-vscode\python\python.exe' 'C:\Users\shashu\PycharmProjects\Python\lab\lab-2.py'
Enter Previous Units (PU): 150
Enter Current Units (CU): 200
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\shashu>
```

Task 5: Final Bill Generation and Output Analysis

Scenario

The final electricity bill must present all values clearly.

Task Description

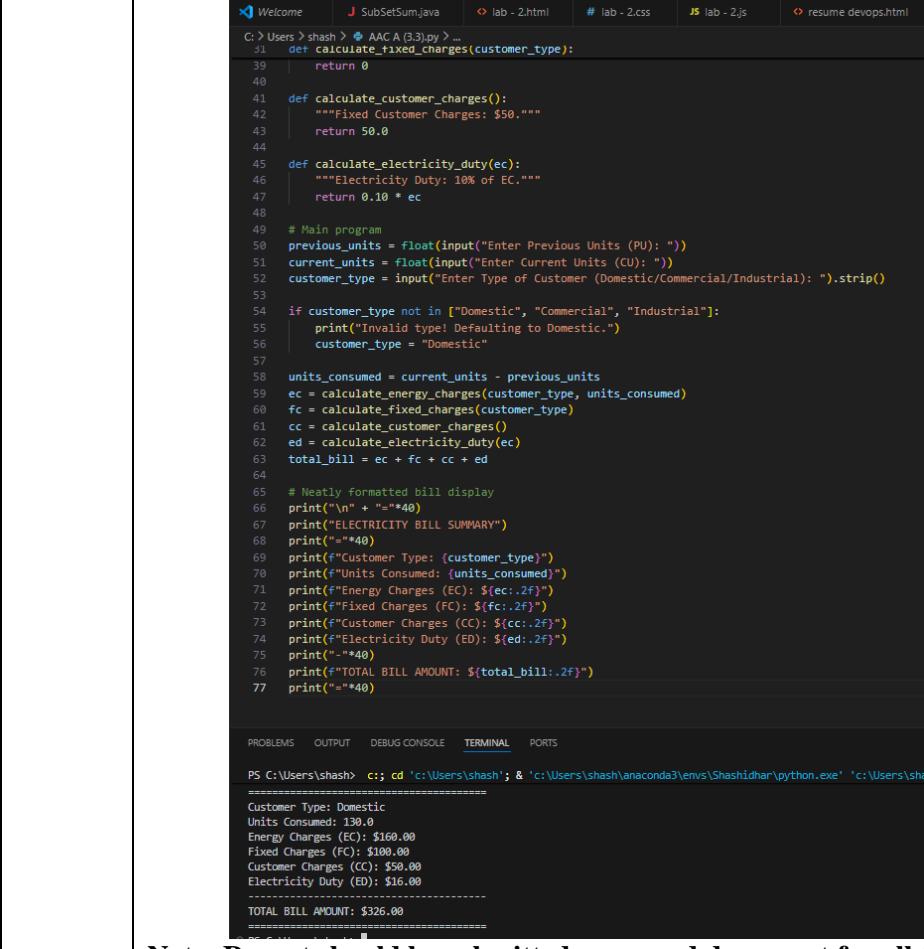
Develop the final Python application to:

- 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

ASSIGNMENT-3.3



```

  Welcome          AAC A (3.3).py      lab - 2.html      lab - 2.css      lab - 2.js      resume devops.html
C:\Users>shash > AAC A (3.3).py >_>
31     def calculate_fixed_charges(customer_type):
32         return 0
33
34     def calculate_customer_charges():
35         """Fixed Customer Charges: $50.00"""
36         return 50.0
37
38     def calculate_electricity_duty(ec):
39         """Electricity Duty: 10% of EC.00"""
40         return 0.10 * ec
41
42     # Main program
43     previous_units = float(input("Enter Previous Units (PU): "))
44     current_units = float(input("Enter Current Units (CU): "))
45     customer_type = input("Enter Type of Customer (Domestic/Commercial/Industrial): ").strip()
46
47     if customer_type not in ["Domestic", "Commercial", "Industrial"]:
48         print("Invalid type! Defaulting to Domestic.")
49     customer_type = "Domestic"
50
51     units_consumed = current_units - previous_units
52     ec = calculate_energy_charges(customer_type, units_consumed)
53     fc = calculate_fixed_charges(customer_type)
54     cc = calculate_customer_charges()
55     ed = calculate_electricity_duty(ec)
56     total_bill = ec + fc + cc + ed
57
58     # Nicely formatted bill display
59     print("\n" + "="*40)
60     print("ELECTRICITY BILL SUMMARY")
61     print("="*40)
62     print(f"Customer Type: {customer_type}")
63     print(f"Units Consumed: {units_consumed}")
64     print(f"Energy Charges (EC): ${ec:.2f}")
65     print(f"Fixed Charges (FC): ${fc:.2f}")
66     print(f"Customer Charges (CC): ${cc:.2f}")
67     print(f"Electricity Duty (ED): ${ed:.2f}")
68     print("-"*40)
69     print(f"TOTAL BILL AMOUNT: ${total_bill:.2f}")
70     print("-"*40)
71
72
73
74
75
76
77

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

PS C:\Users\shash> c:; cd "c:\Users\shash"; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' 'c:\Users\shash\OneDrive\Desktop\Assignment 3.3\AAC A (3.3).py'
=====
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.