Name: Karabi Mandal **Assignment:** 1.4 **Batch**: 25
Hallticket Number: 2303a51620

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* **Lab Objectives:**	Week1 - Monday

- 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.

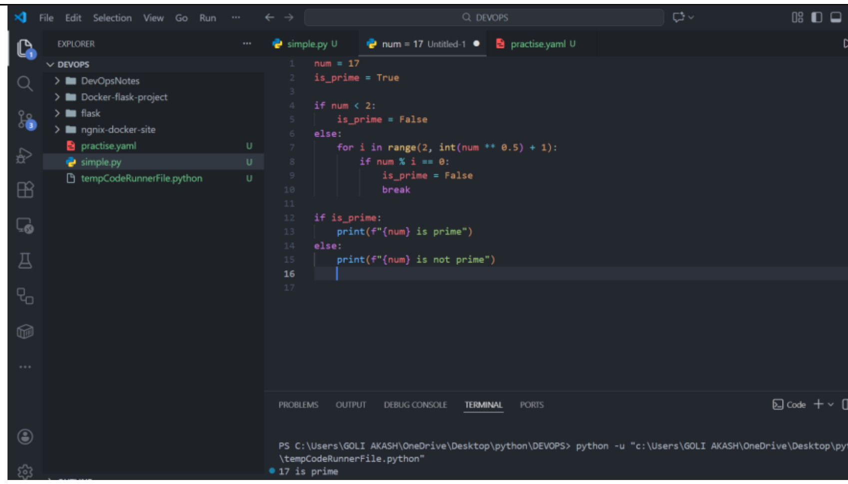
Expected Output
- Install and configure GitHub Copilot in VS Code. Take screenshots of each step.

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



Prompt:- Write a simple Python program to check whether a given number is a prime number **without using functions**. Use basic conditional statements and loops, assume the number is prime initially, and change the result only if a divisor is found. Keep the logic clear, beginner-friendly, and easy to understand.

Explaination:- This code checks whether a number (num = 17) is a prime number by first assuming it is prime using the variable is_prime = True. If the number is less than 2, it is immediately marked as not prime. Otherwise, the code checks whether the number is divisible by any value from 2 up to the square root of the number; if any divisor is found, is_prime is set to False and the loop stops. Finally, based on the value of is_prime, the program prints whether the number is prime or not.

Task 2: Efficiency & Logic Optimization (Cleanup)

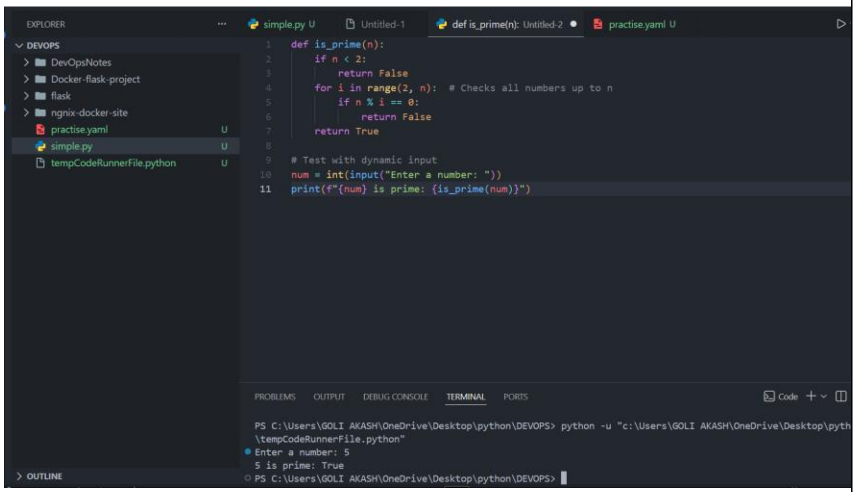
- **Scenario**
  The script must handle larger input values efficiently.

- **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



Prompt:- Optimize this prime number checking code to make it faster and cleaner by reducing unnecessary iterations and improving readability.

Explaination:-
This code defines a function that checks whether a number is prime by returning False for numbers less than 2 and testing divisibility from 2 up to the number itself. If any number divides it evenly, the function returns False; otherwise, it returns True. The program then takes a number from the user and prints whether it is prime

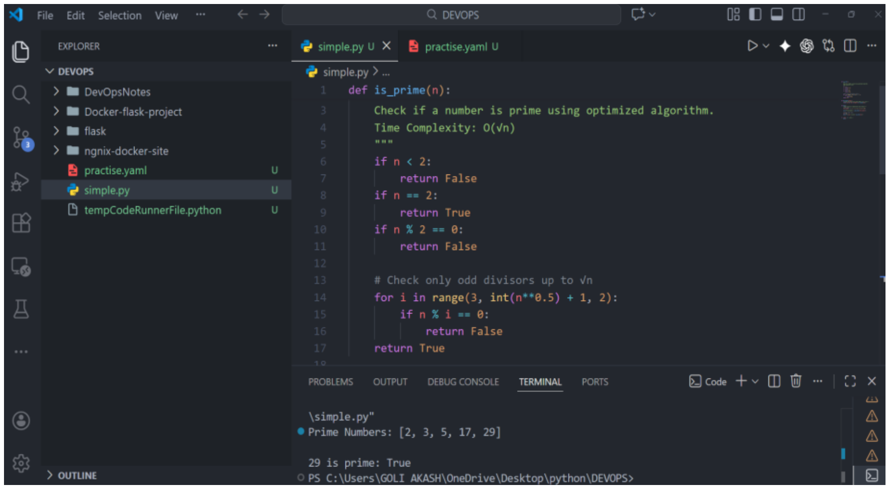
---

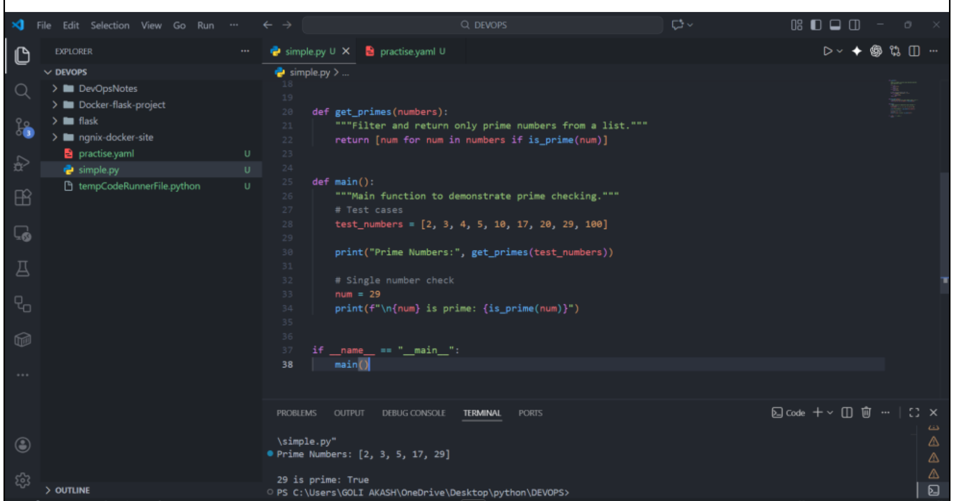
Task 3: Modular Design Using AI Assistance (Prime Number Check Using Functions)

- **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





Prompt: Rewrite the prime number checking logic using functions to make the code modular, reusable, and easier to read. Keep it simple and efficiency.

Explaination:

This program checks prime numbers in a **smart and efficient way**.

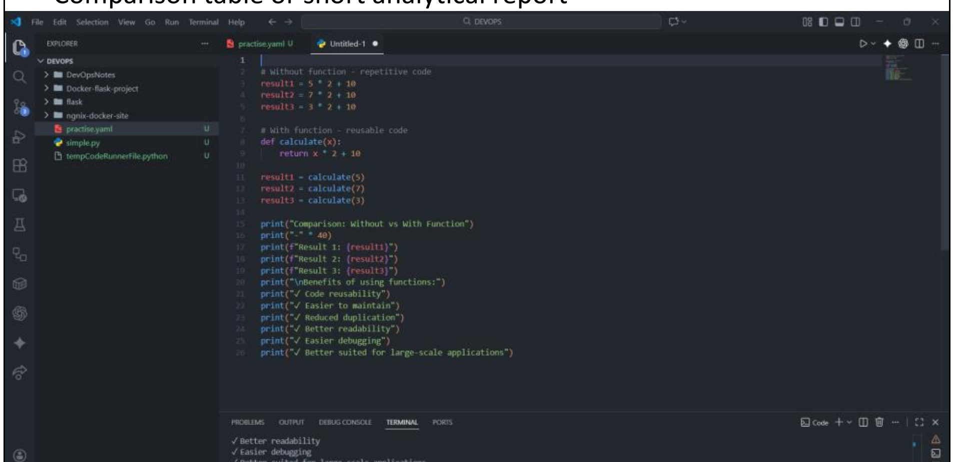
The is_prime function first removes obvious non-prime cases like numbers less than 2 and even numbers (except 2). Then, instead of checking all numbers, it checks only **odd divisors up to the square root** of the number, which makes it much faster. The get_primes function uses this logic to filter out only prime numbers from a list. The main function demonstrates both checking a list of numbers and testing a single number, and the if __name__ == "__main__" line ensures the program runs only when the file is executed directly.

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



Prompt: Compare the prime number checking code written **with functions** and **without functions**. Analyze them in terms of code clarity, reusability, ease of debugging, and how suitable they are for large-scale applications.

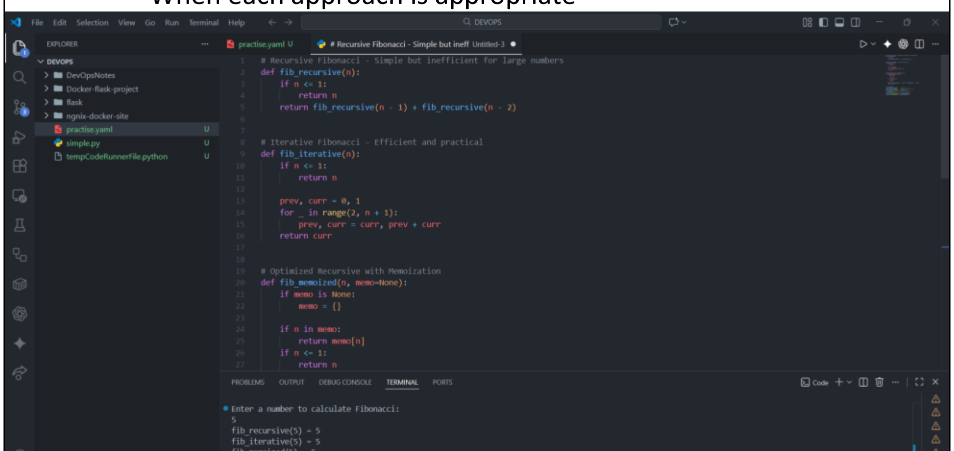
Explaination:
This code shows **why functions are useful** . In the first part, the same calculation (x * 2 + 10) is written again and again for different values, which works but creates repetitive and messy code. In the second part, the calculation is placed inside a function called calculate, so the same logic can be reused just by passing different values. This makes the code cleaner, easier to understand, and easier to update—because if the formula changes, it only needs to be changed in one place. The printed points at the end highlight these advantages, showing how functions help with reusability, readability, maintenance, debugging, and building larger programs.

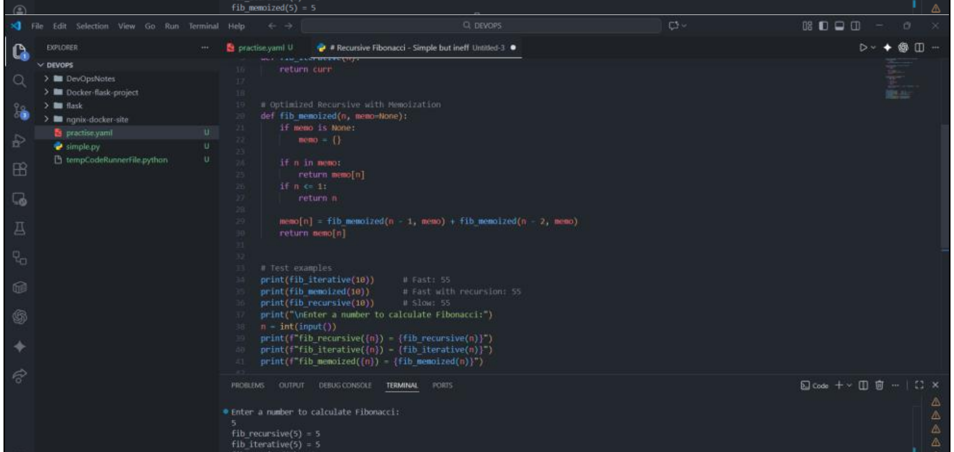
Task 5: AI-Generated Iterative vs Recursive Fibonacci Approaches (Different Algorithmic Approaches to Prime Checking)

- **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





	Prompt: Generate two different prime number checking approaches: one using a basic divisibility method that checks all possible factors, and another optimized version that checks divisibility only up to the square root of the number. Explain the difference in logic and efficiency. **Explaination:** This code calculates Fibonacci numbers in three ways. The recursive method is simple but slow because it repeats the same calculations. The iterative method is fast and efficient by using a loop. The memorized method improves recursion by storing previous results, making it much faster. All three give the same output, but their performance is different.	