AI Assistant lab 1

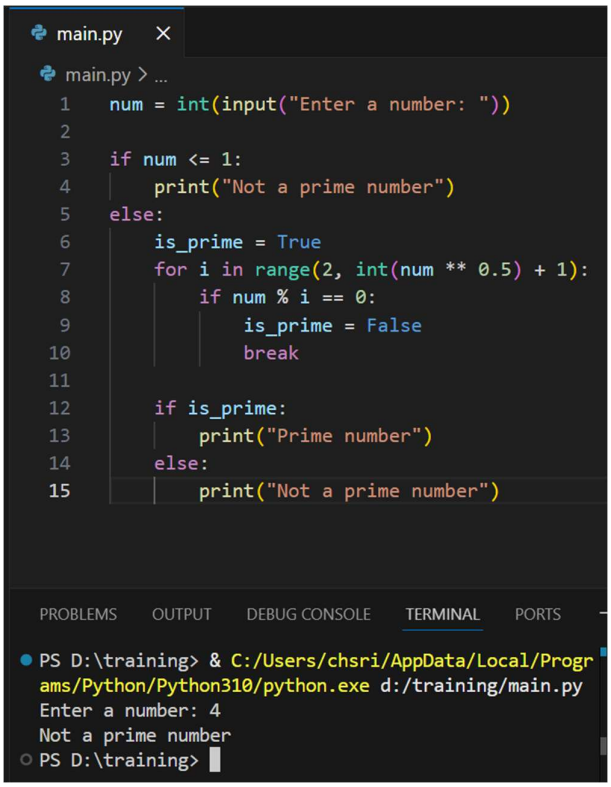
Name: ch srinidh rao

HallTicket:2303A52398

Assignment: 1.4

Task1: AI-Generated Logic Without Modularization (Prime Number Check Without Functions

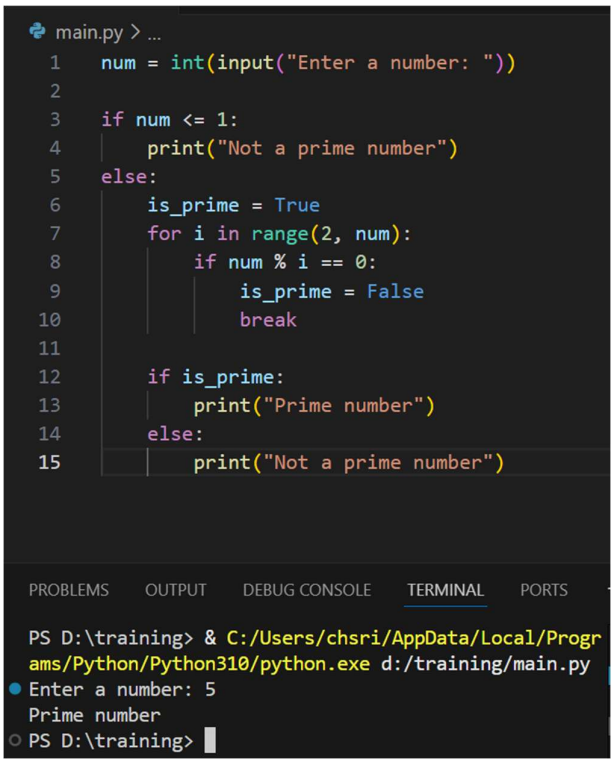
Prompt: write a python code to check the given number is prime or not without using functions



Observations: The program effectively checks whether a given number is prime by implementing the entire logic directly in the main code without using any user defined functions. It accepts user input and applies conditional statements and loops to test divisibility. This approach makes the code simple and easy to understand, especially for beginners learning basic programming concepts. While the absence of modularization limits reusability and scalability, it helps in clearly demonstrating the core logic. Overall, the task fulfills its objective of basic validation in a learning-focused application

Task2: Efficiency & Logic Optimization (Cleanup)

Prompt: optimize the code and simplify the logic to check to get the optimal solution for prime number checking



Observations: The original code checks divisibility from 2 to num-1, resulting in O(n) time complexity, which is inefficient for large numbers. The optimized version reduces unnecessary iterations by checking divisors only up to the square root of the number, since a larger factor would have a corresponding smaller factor already checked. This reduces the time complexity to O(√n). Early termination using break further improves performance. Additionally, clearer variable names and simplified logic enhance readability and maintainability

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

Prompt :give me a code for to check the prime number using the functions

```python
num = int(input("Enter a number: "))

if num <= 1:
    print("Not a prime number")
else:
    is_prime = True
    for i in range(2, int(num ** 0.5) + 1):
        if num % i == 0:
            is_prime = False
            break

    if is_prime:
        print("Prime number")
    else:
        print("Not a prime number")
```

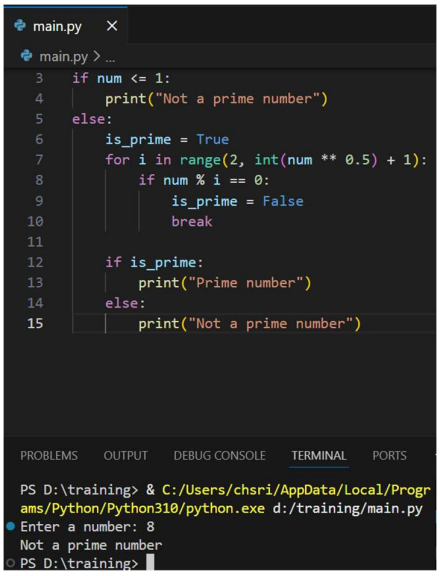
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS D:\training> & C:/Users/chsri/AppData/Local/Programs/Python/Python310/python.exe d:/training/main.py
Enter a number: 4
Not a prime number
PS D:\training>
```

Observations: The modular, function-based implementation successfully encapsulates the prime-checking logic into a reusable user-defined function that returns a Boolean value. This design improves code reusability across multiple modules and enhances maintainability. The use of meaningful, AI-assisted comments increases code clarity and helps in understanding the logic flow. Returning a Boolean value allows the function to be easily integrated into different applications or validation pipelines. Overall, the modular approach results in cleaner, more scalable, and professional-quality code.

Task 4: Comparative Analysis –With vs Without Functions

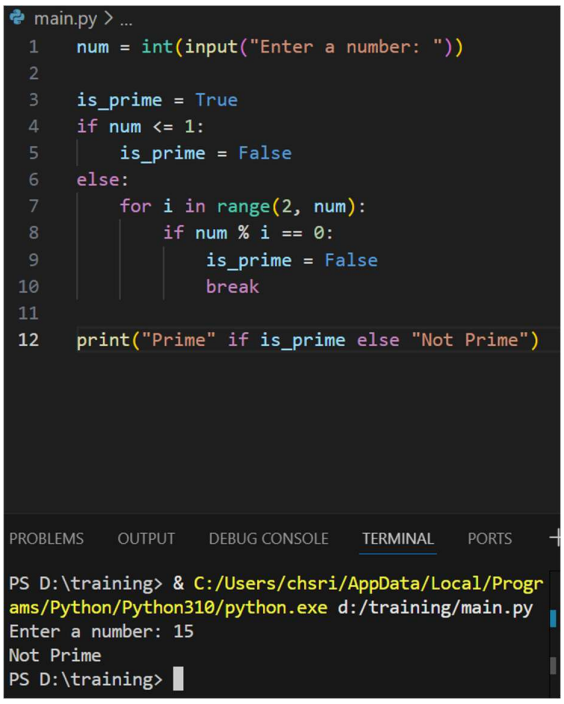
Prompt :give me the comparative analysis for the task1 task2 task 3



Observation: The version without functions is straightforward and easy to follow for beginners, but the logic is tightly coupled to the main code, reducing overall clarity as the program grows. In contrast, the function-based implementation presents clearer structure by separating logic from execution, making the code easier to read and understand. Reusability is minimal in the non-modular version, whereas the function-based approach allows the prime-checking logic to be reused across multiple modules. Debugging is simpler with functions since errors can be isolated within a single unit of code, unlike the monolithic structure where issues are harder to trace. Overall, the function-based design is far more suitable for large-scale and maintainable applications, while the non-function approach is best limited to small or educational scripts.

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

Prompt: itetrative vs recursive Fibbanaci give me the code



Observations: The basic divisibility approach is simple and easy to understand, but it performs many unnecessary checks, making it inefficient for larger input values. The optimized approach improves performance by limiting the loop to the square root of the number, reducing the number of iterations significantly. Execution flow in the optimized version is shorter and more efficient due to early termination. For large inputs, the optimized method provides much better performance compared to the naive approach. Overall, the basic method is suitable for learning purposes, while the optimized approach is more appropriate for real world and large-scale applications.