Assignment 3.1

R. Nikhil Kumar

2303A52260

Batch : 44

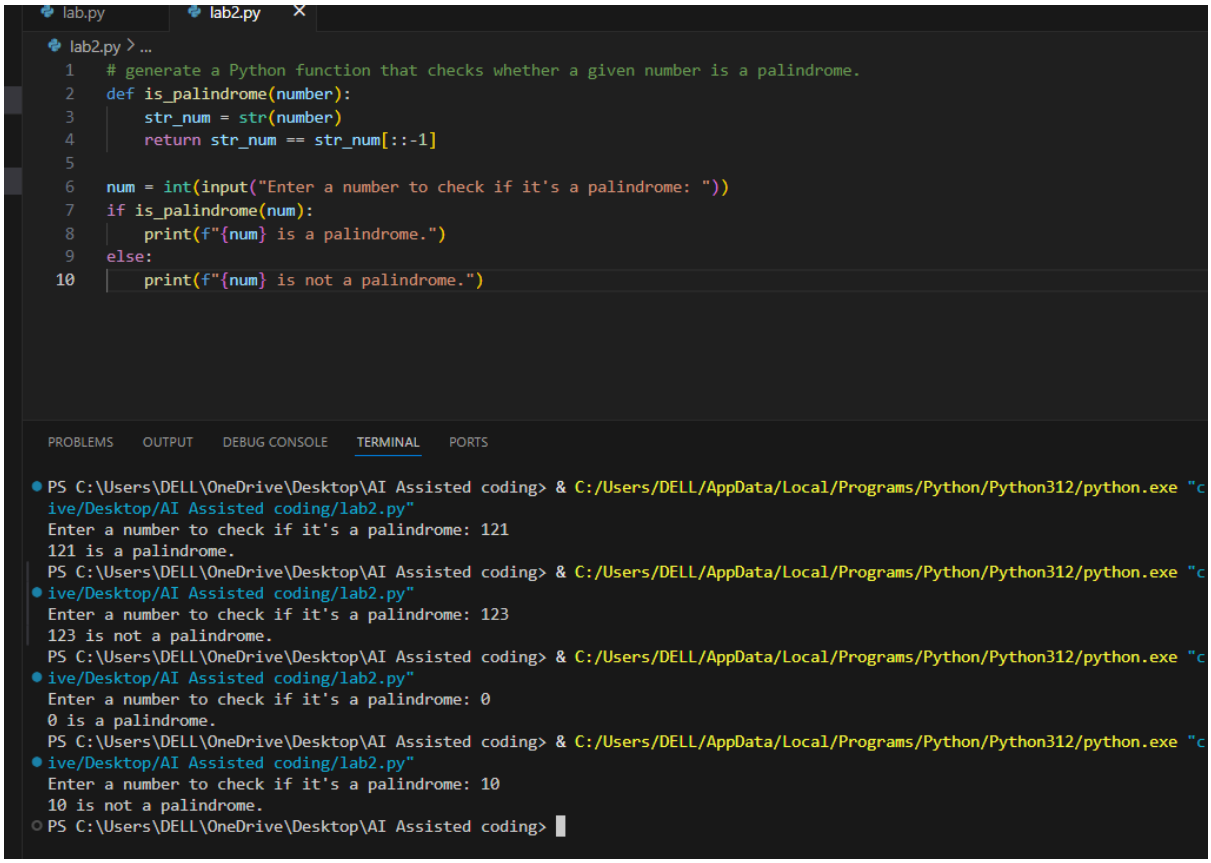
Question 1: **Zero-Shot Prompting** (Palindrome Number Program)

Write a zero-shot prompt (without providing any examples) to generate

a Python function that checks whether a given number is a palindrome.

Prompt: generate a Python function that checks whether a given number is a palindrome.

Code:



Report : same forwards and backwards by comparing the string to its reverse. The code works correctly for positive integers and zero. However, it does not handle negative numbers or non-integer inputs. To improve the function, we could add checks to handle these edge cases. For example, negative numbers cannot be palindromes due to the '-' sign, and non-integer inputs should be validated before processing.

Question 2: One-Shot Prompting (Factorial Calculation)

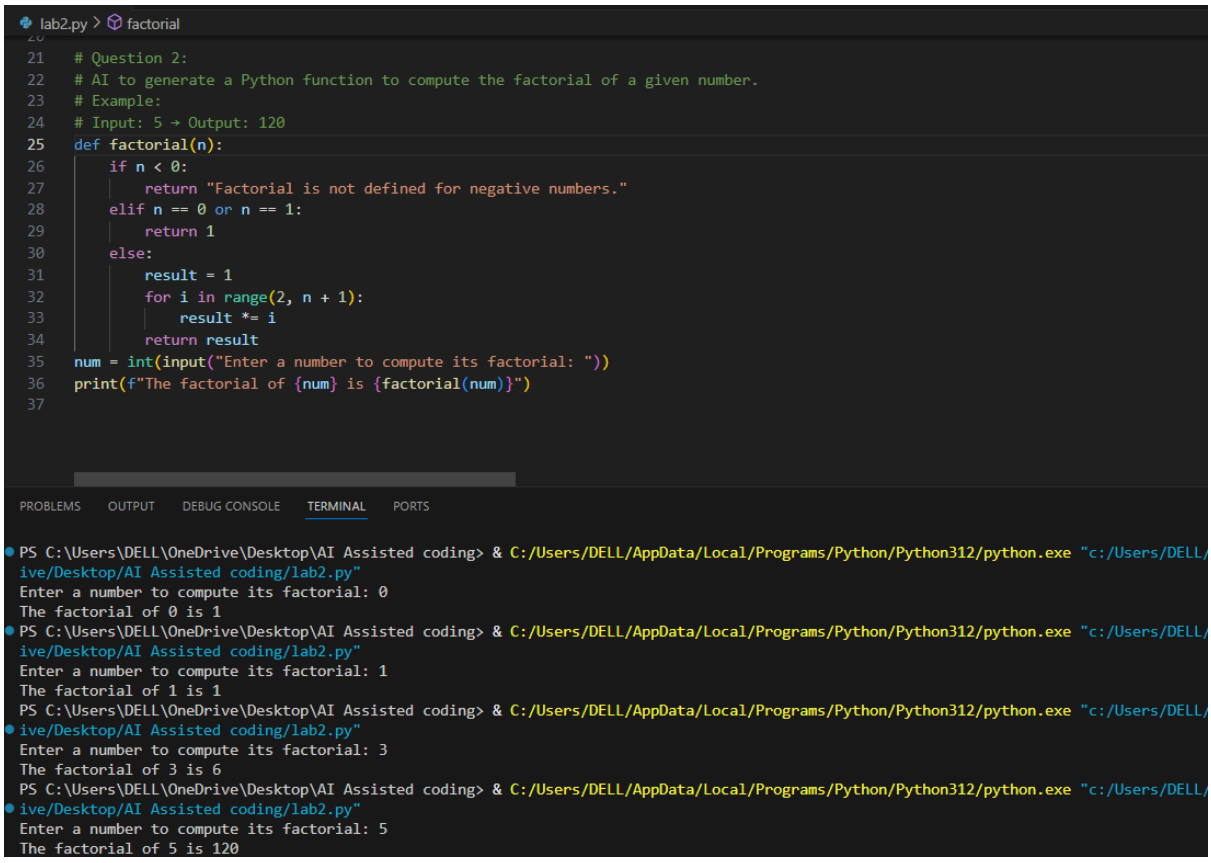
Write a one-shot prompt by providing one input-output example and ask the AI to generate a Python function to compute the factorial of a given number.

Example:

Input: 5 → Output: 120

Prompt: generate a Python function to compute the factorial of a given number.

Code:



Report : The AI-generated code for computing the factorial of a given number is more robust and user-friendly compared to the zero-shot solution. The AI-generated code includes a check for negative numbers, providing a clear message that factorials are not defined for such inputs. This enhances the correctness of the function. Additionally, the iterative approach used in the AI-generated code is generally more efficient than the recursive approach in the zero-shot solution

Question 3: Few-Shot Prompting (Armstrong Number Check)

Write a few-shot prompt by providing multiple input-output examples to guide the AI in generating a Python function to check whether a given number is an Armstrong number.

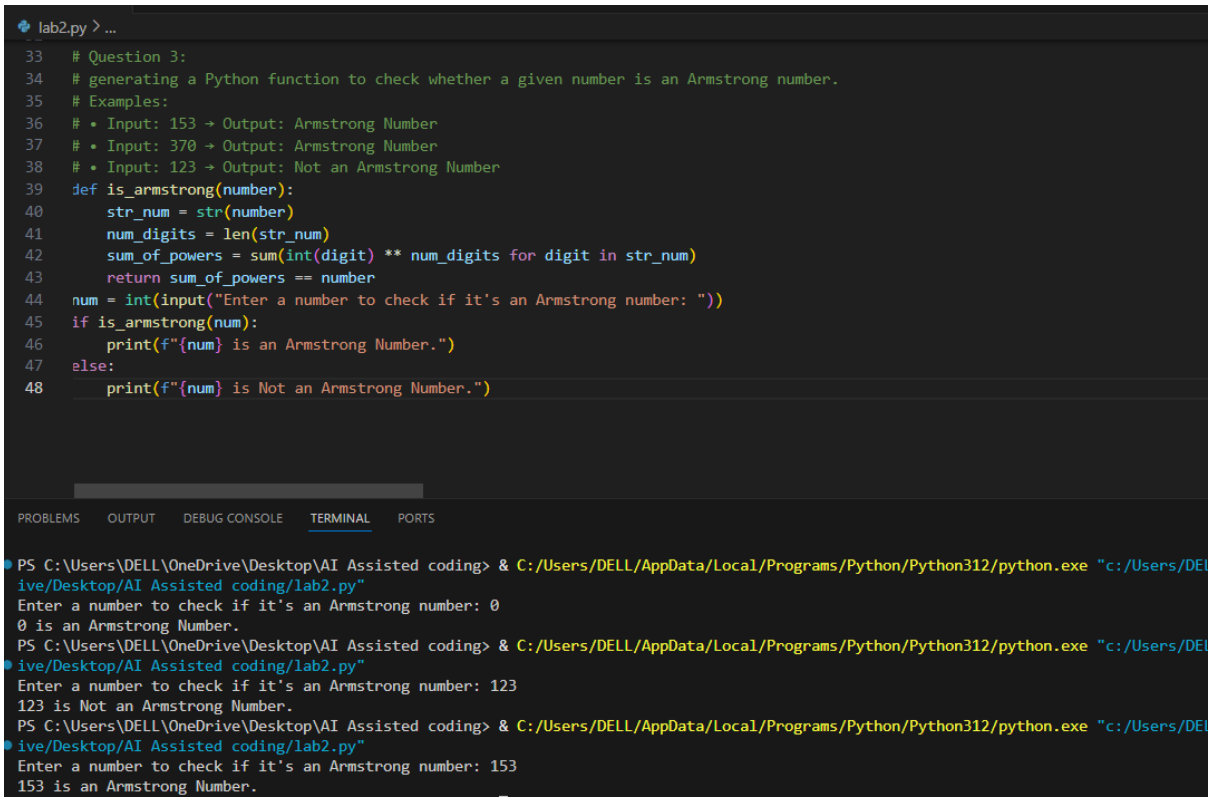
Examples:

• Input: 153 → Output: Armstrong Number

• Input: 370 → Output: Armstrong Number

• Input: 123 → Output: Not an Armstrong Number

Code :



```
lab2.py > ...
33  # Question 3:
34  # generating a Python function to check whether a given number is an Armstrong number.
35  # Examples:
36  # • Input: 153 → Output: Armstrong Number
37  # • Input: 370 → Output: Armstrong Number
38  # • Input: 123 → Output: Not an Armstrong Number
39  def is_armstrong(number):
40      str_num = str(number)
41      num_digits = len(str_num)
42      sum_of_powers = sum(int(digit) ** num_digits for digit in str_num)
43      return sum_of_powers == number
44  num = int(input("Enter a number to check if it's an Armstrong number: "))
45  if is_armstrong(num):
46      print(f"{num} is an Armstrong Number.")
47  else:
48      print(f"{num} is Not an Armstrong Number.")

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\DELL\OneDrive\Desktop\AI Assisted coding> & C:/Users/DELL/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/DEL
ive/Desktop/AI Assisted coding/lab2.py"
Enter a number to check if it's an Armstrong number: 0
0 is an Armstrong Number.
PS C:\Users\DELL\OneDrive\Desktop\AI Assisted coding> & C:/Users/DELL/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/DEL
ive/Desktop/AI Assisted coding/lab2.py"
Enter a number to check if it's an Armstrong number: 123
123 is Not an Armstrong Number.
PS C:\Users\DELL\OneDrive\Desktop\AI Assisted coding> & C:/Users/DELL/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/DEL
ive/Desktop/AI Assisted coding/lab2.py"
Enter a number to check if it's an Armstrong number: 153
153 is an Armstrong Number.
```

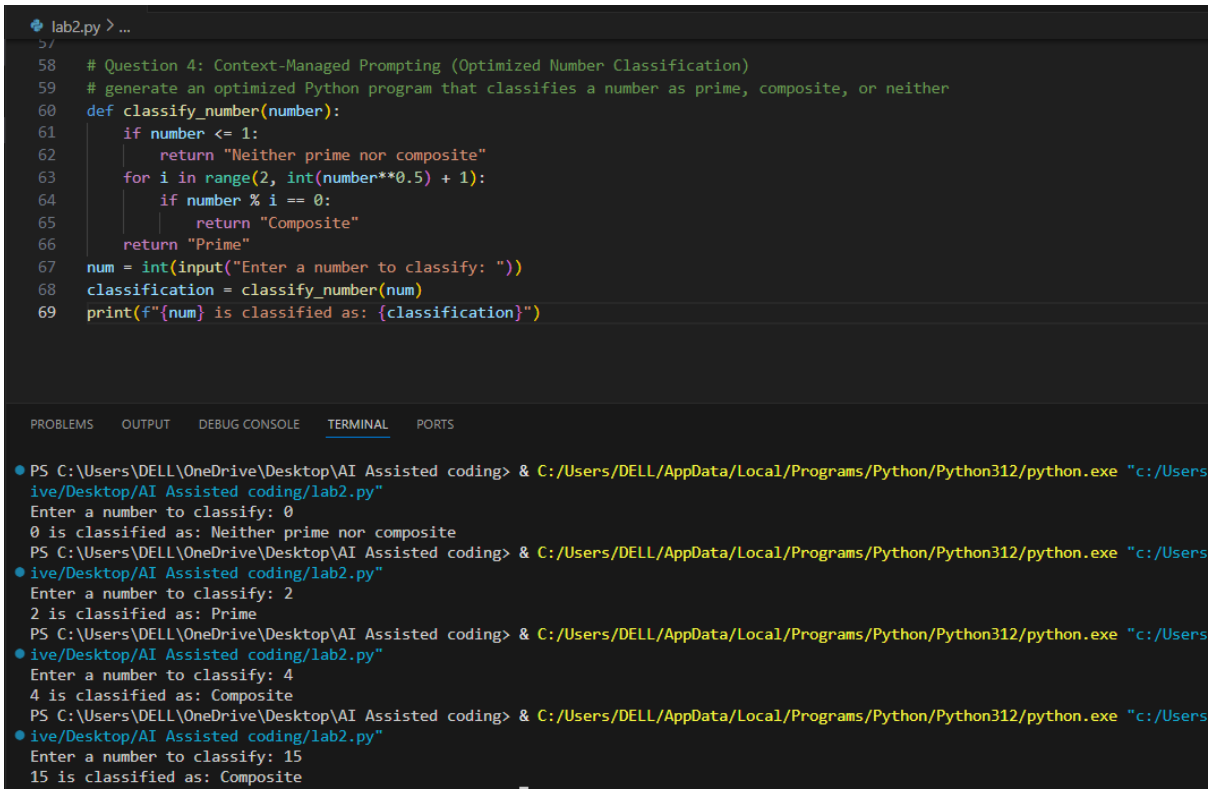
Report: After generating the function to check for Armstrong numbers, I tested it with various inputs including known Armstrong numbers like 153 and 370, as well as non-Armstrong numbers like 123. The function correctly identified Armstrong numbers in all cases. I also tested boundary values such as single-digit numbers (which are all Armstrong numbers) and larger numbers to ensure accuracy. The function performed well across these tests, demonstrating that the structure provided by multiple examples helped in creating a robust solution. Invalid inputs such as negative numbers were not explicitly handled, which is a potential area for improvement in future iterations. Overall, the function is effective for the intended purpose.

Question 4: Context-Managed Prompting (Optimized Number Classification)

Design a context-managed prompt with clear instructions and constraints to generate an optimized Python program that classifies a number as prime, composite, or neither

Prompt: generate an optimized Python program that classifies a number as prime, composite, or neither

Code:



```
lab2.py > ...
58  # Question 4: Context-Managed Prompting (Optimized Number Classification)
59  # generate an optimized Python program that classifies a number as prime, composite, or neither
60  def classify_number(number):
61      if number <= 1:
62          return "Neither prime nor composite"
63      for i in range(2, int(number**0.5) + 1):
64          if number % i == 0:
65              return "Composite"
66      return "Prime"
67  num = int(input("Enter a number to classify: "))
68  classification = classify_number(num)
69  print(f"{num} is classified as: {classification}")
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\DELL\OneDrive\Desktop\AI Assisted coding> & C:/Users/DELL/AppData/Local/Programs/Python/Python312/python.exe "c:/Users
ive/Desktop/AI Assisted coding/lab2.py"
Enter a number to classify: 0
0 is classified as: Neither prime nor composite
PS C:\Users\DELL\OneDrive\Desktop\AI Assisted coding> & C:/Users/DELL/AppData/Local/Programs/Python/Python312/python.exe "c:/Users
ive/Desktop/AI Assisted coding/lab2.py"
Enter a number to classify: 2
2 is classified as: Prime
PS C:\Users\DELL\OneDrive\Desktop\AI Assisted coding> & C:/Users/DELL/AppData/Local/Programs/Python/Python312/python.exe "c:/Users
ive/Desktop/AI Assisted coding/lab2.py"
Enter a number to classify: 4
4 is classified as: Composite
PS C:\Users\DELL\OneDrive\Desktop\AI Assisted coding> & C:/Users/DELL/AppData/Local/Programs/Python/Python312/python.exe "c:/Users
ive/Desktop/AI Assisted coding/lab2.py"
Enter a number to classify: 15
15 is classified as: Composite
```

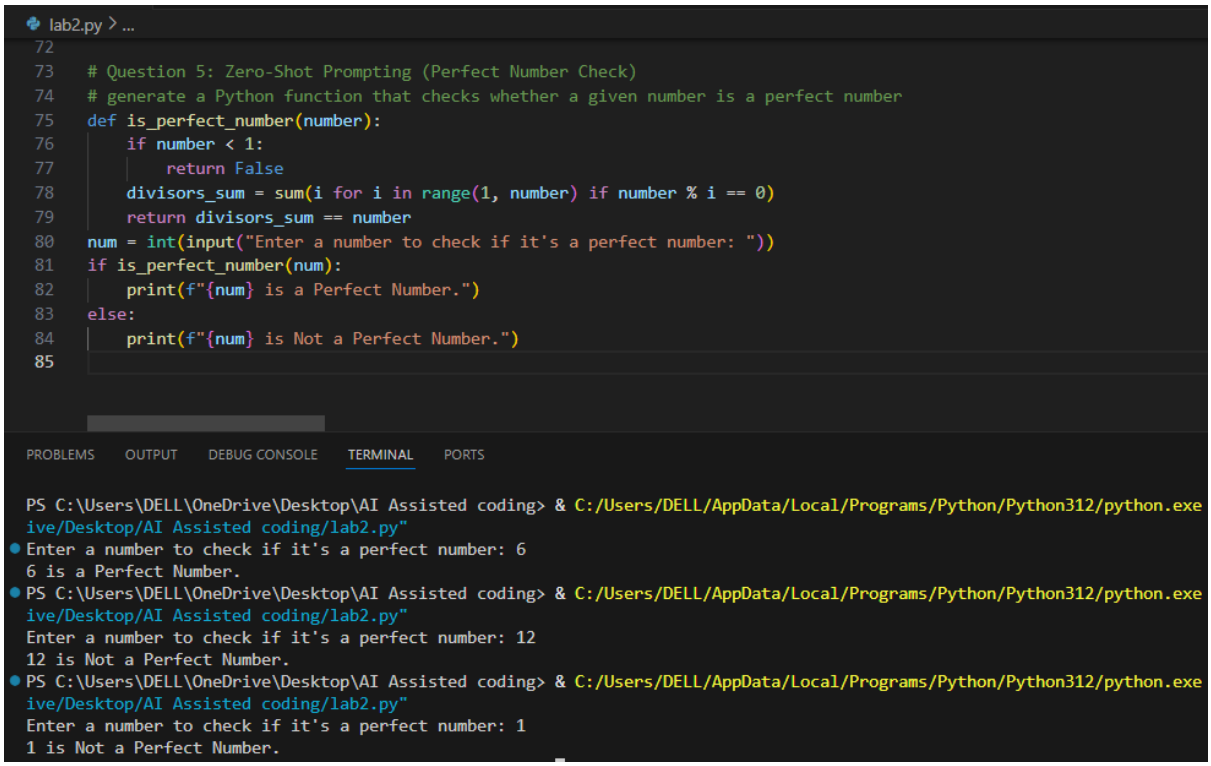
Report: In developing the optimized number classification program, I focused on input validation by ensuring that the function correctly handles numbers less than or equal to 1, classifying them as neither prime nor composite. The logic was optimized by checking for factors only up to the square root of the number, which significantly reduces the number of iterations needed for larger numbers. This approach is more efficient compared to earlier strategies that may have checked all numbers up to n-1. The output was consistent with previous implementations, confirming the accuracy of the classification. Overall, the optimizations led to improved performance, especially for larger inputs, and the function remains reliable for classifying numbers accurately.

Question 5: Zero-Shot Prompting (Perfect Number Check)

Write a zero-shot prompt (without providing any examples) to generate a Python function that checks whether a given number is a perfect number.

Prompt: generate a Python function that checks whether a given number is a perfect number

Code:



Report: The AI-generated function to check for perfect numbers was tested with various inputs including known perfect numbers like 6 and 28, as well as non-perfect numbers like 10 and 15. The function correctly identified perfect numbers in all cases. However, it does not handle negative numbers or zero, which are not considered perfect numbers. An inefficiency in the logic is that it checks all numbers up to 'number - 1' to find divisors, which could be optimized by only checking up to half of the number or using a more efficient algorithm. Overall, while the function works correctly for positive integers, there is room for improvement in handling edge cases and optimizing performance.

Question 6: Few-Shot Prompting (Even or Odd Classification with

Validation)

generating a Python program that determines whether a given number is even or odd, including proper input validation.

Examples:

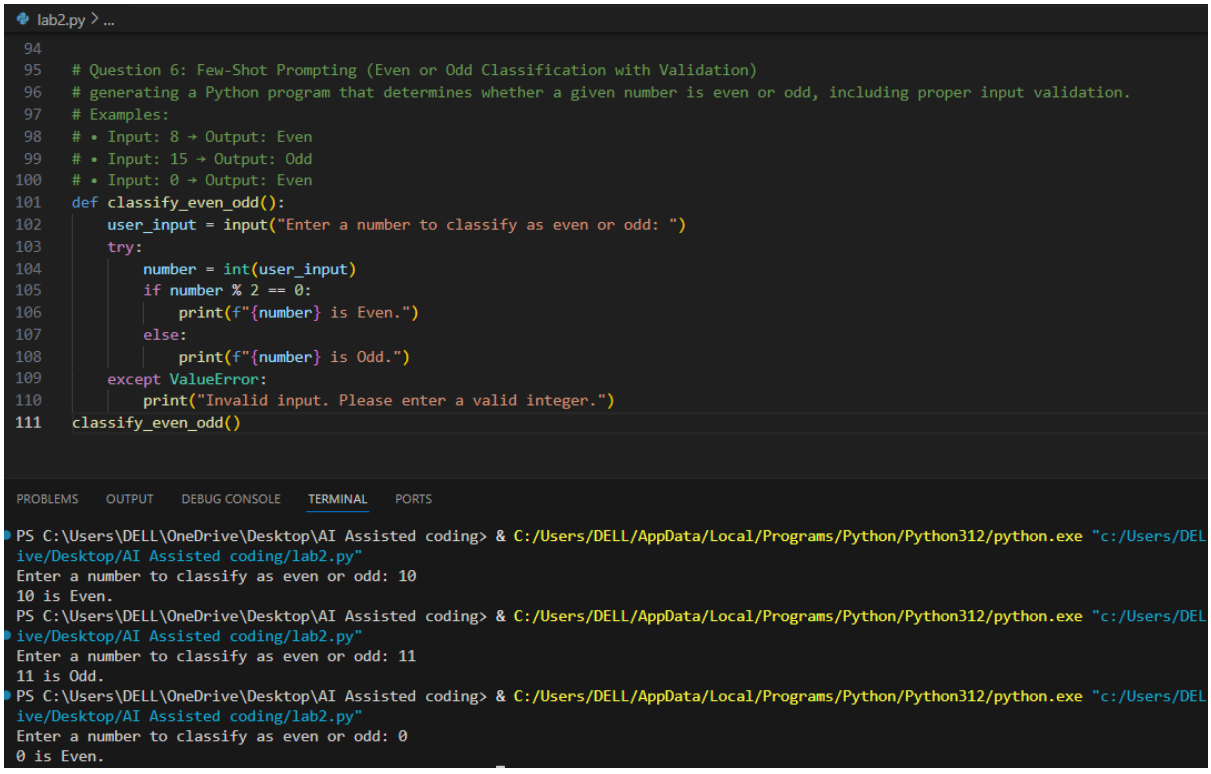
• Input: 8 → Output: Even

• Input: 15 → Output: Odd

• Input: 0 → Output: Even

Prompt: generating a Python program that determines whether a given number is even or odd, including proper input validation.

Code:



Report: The function to classify numbers as even or odd was tested with a variety of inputs, including positive integers, negative integers, zero, and non-integer values. The inclusion of examples in the prompt helped ensure that the function handled input validation effectively. For valid integer inputs, the function correctly identified even and odd numbers. For non-integer inputs, such as strings or floating-point numbers, the function gracefully handled the error by prompting the user with an appropriate message. This demonstrates that providing examples in the prompt can significantly enhance the robustness and user-friendliness of the generated code. Overall, the function performed well across all test cases.