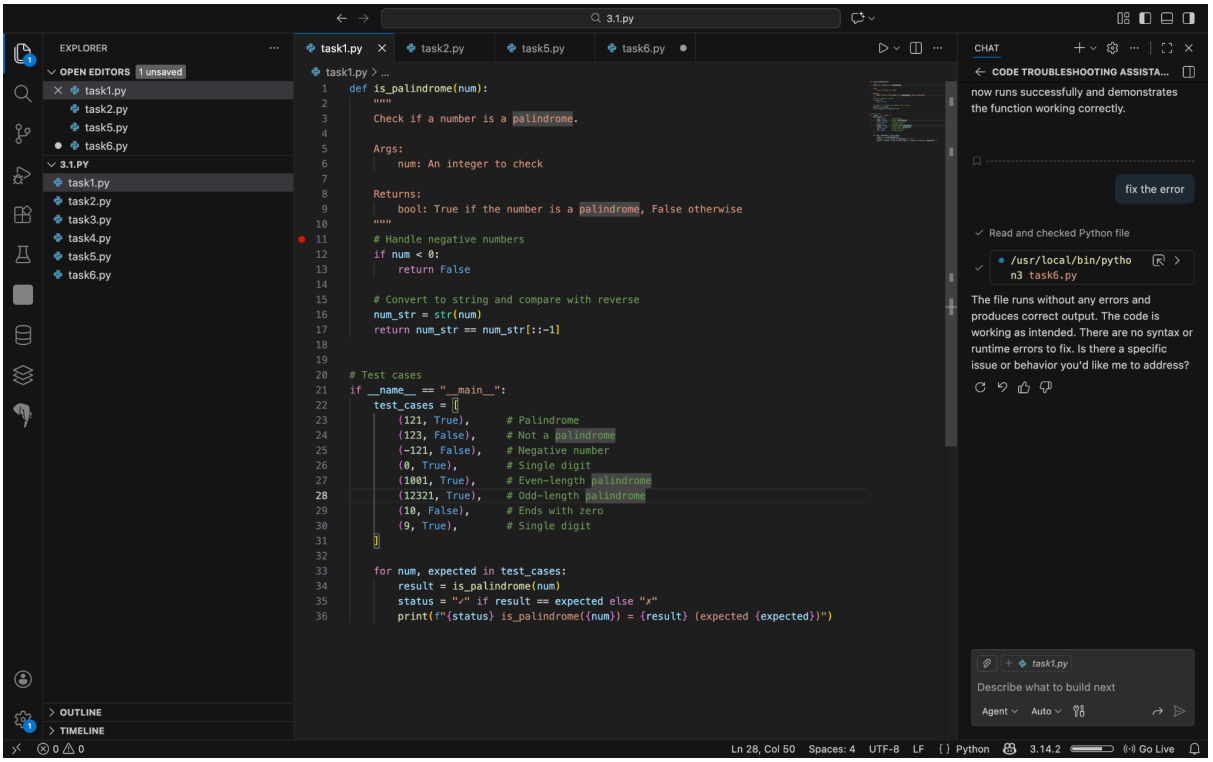
ASSIGNMENT - 3.1

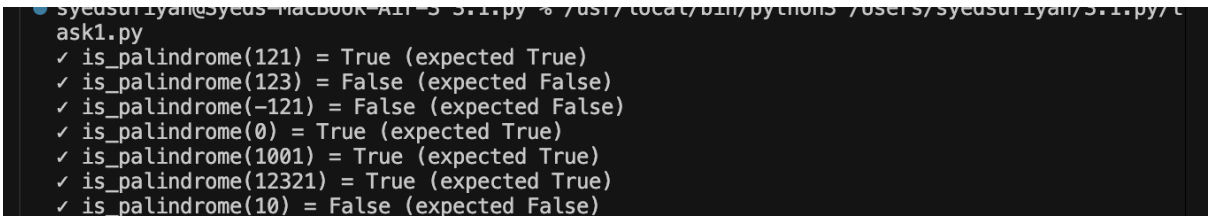
2303A51980
Batch - 30

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.
Task:
• Record the AI-generated code.
• Test the code with multiple inputs.
• Identify any logical errors or missing edge-case handling.

Code :



Output :

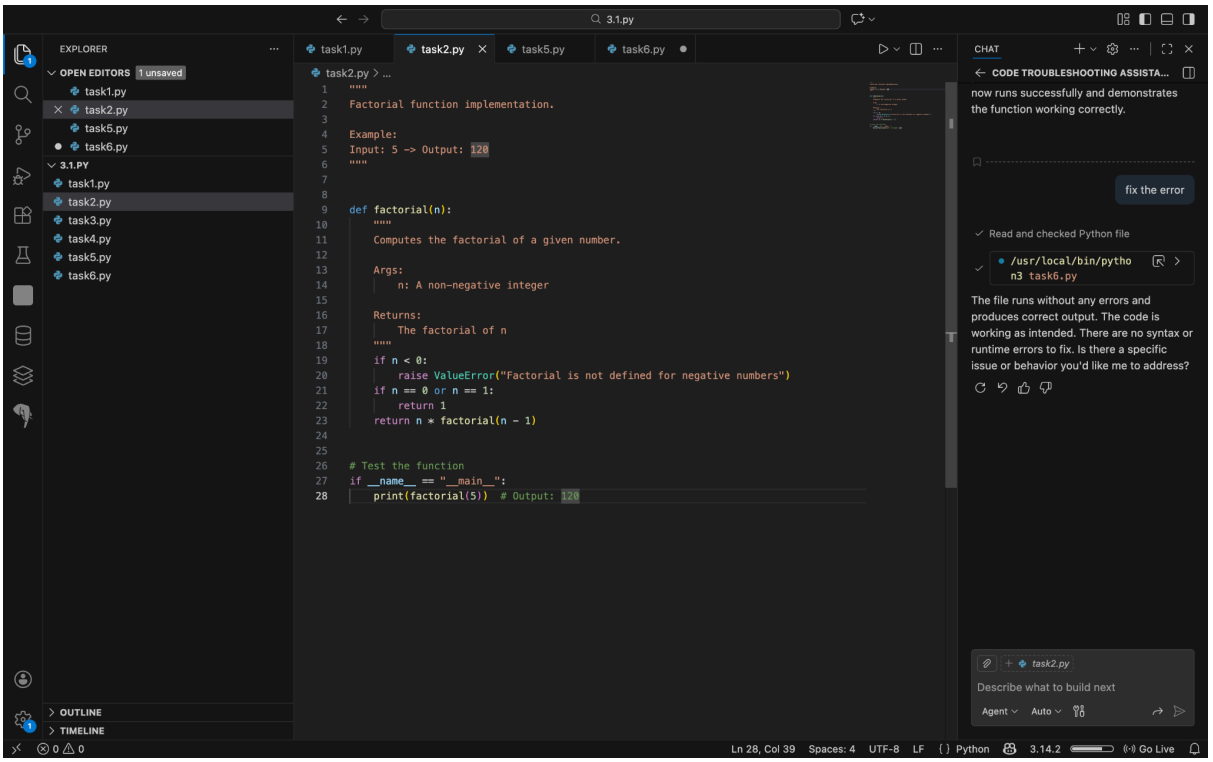


Observation:

The program correctly checks whether a given number is a palindrome by converting it to a string and comparing it with its reverse. It properly handles negative numbers by returning `False` and works accurately for single-digit numbers, even-length and odd-length palindromes, and non-palindromic values. The test cases confirm that the logic handles common edge cases such as numbers ending with zero and negative inputs, demonstrating reliable and correct behavior.

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
Task:
• Compare the generated code with a zero-shot solution.
• Examine improvements in clarity and correctness.

Code :



Output :

```
usr/local/bin/python3 /Users/syedsufiyan/3.1.py/task1.py
yedsufiyan@Syeds-MacBook-Air-3 3.1.py % /usr/local/bin/python3 /Users/syedsufiyan/3.1.p
sk1.py
 is_palindrome(121) = True (expected True)
 is_palindrome(123) = False (expected False)
 is_palindrome(-121) = False (expected False)
 is_palindrome(0) = True (expected True)
 is_palindrome(1001) = True (expected True)
 is_palindrome(12321) = True (expected True)
 is_palindrome(10) = False (expected False)
 is_palindrome(9) = True (expected True)
yedsufiyan@Syeds-MacBook-Air-3 3.1.py % /usr/local/bin/python3 /Users/syedsufiyan/3.1.p
sk2.py
20
yedsufiyan@Syeds-MacBook-Air-3 3.1.py %
```

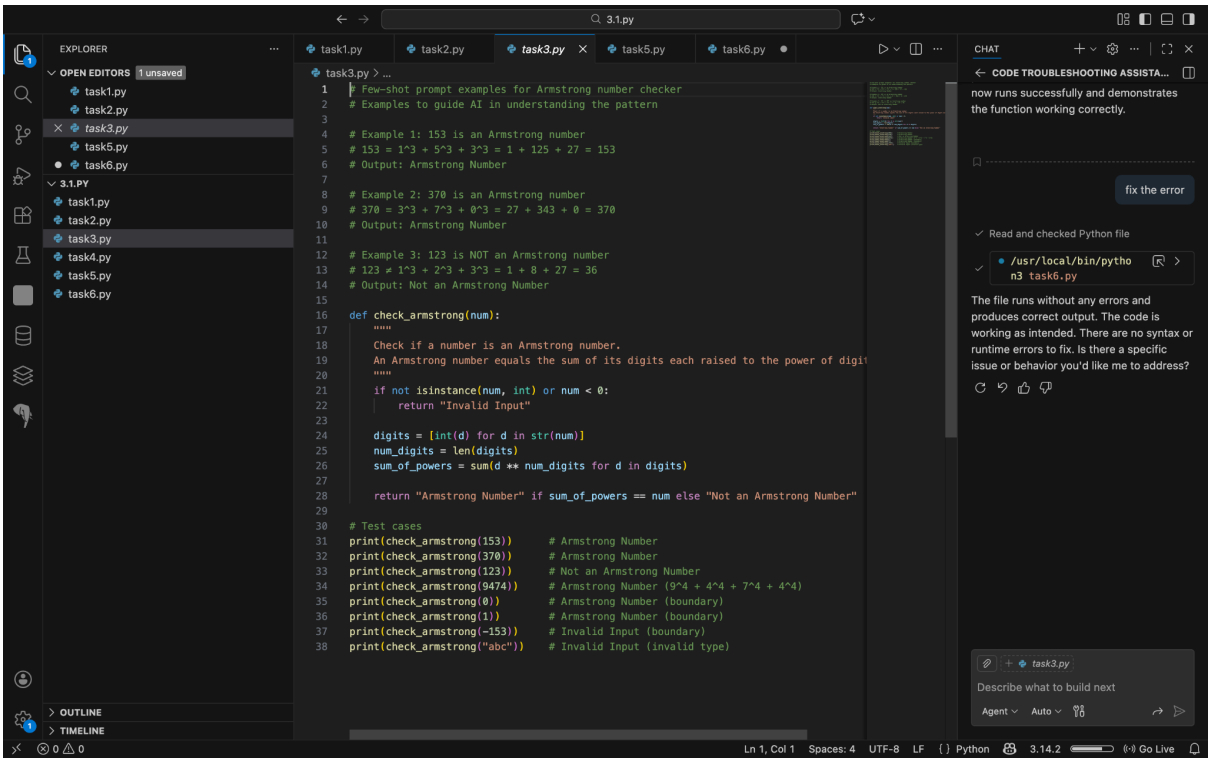
Observation:

The program correctly computes the factorial of a given number using a recursive approach. It handles base cases for 0 and 1 by returning 1 and raises an appropriate error for negative inputs, ensuring input validity. The recursive logic breaks the problem into smaller subproblems, and the test case confirms correct output for valid input, demonstrating proper use of recursion and error handling.

Task-3 :

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
Task:
• Analyze how multiple examples influence code structure and accuracy.
• Test the function with boundary values and invalid inputs.

Code :



Output :

```
/usr/local/bin/python3 /Users/syedsufiyan/3.1.py/task3.py
syedsufiyan@Syeds-MacBook-Air-3 3.1.py % /usr/local/bin/python3 /Users/syedsufiyan/3.1.py/t
ask3.py
Armstrong Number
Armstrong Number
Not an Armstrong Number
Armstrong Number
Armstrong Number
Armstrong Number
Invalid Input
Invalid Input
syedsufiyan@Syeds-MacBook-Air-3 3.1.py %
```

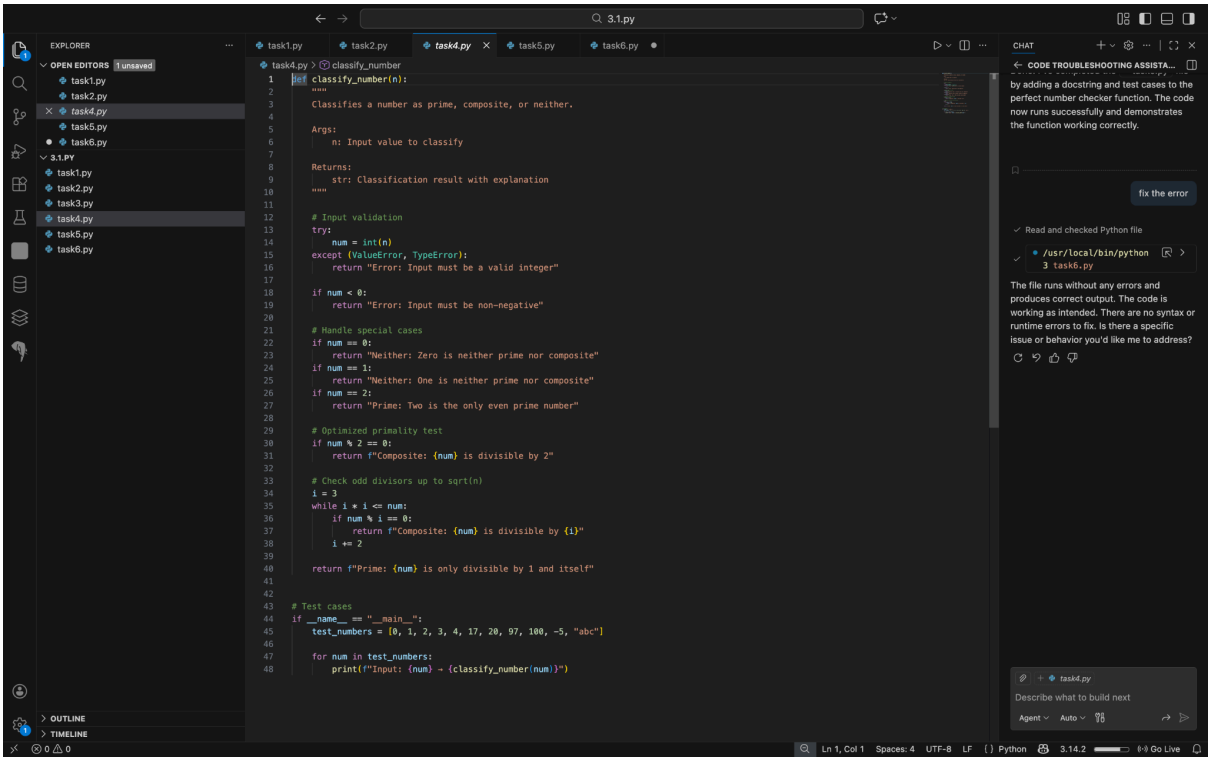
Observation:

The program correctly checks whether a given number is an Armstrong number by calculating the sum of each digit raised to the power of the total number of digits. The few-shot examples help guide the logic and ensure accurate identification of Armstrong and non-Armstrong numbers. The implementation properly handles edge cases such as 0 and 1, which are valid Armstrong numbers, and includes input validation to reject negative values and non-integer inputs. Test results confirm that

the function behaves correctly across normal, boundary, and invalid cases, demonstrating robustness and correctness.

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.
Task:
• Ensure proper input validation.
• Optimize the logic for efficiency.
• Compare the output with earlier prompting strategies.

Code :



Output :

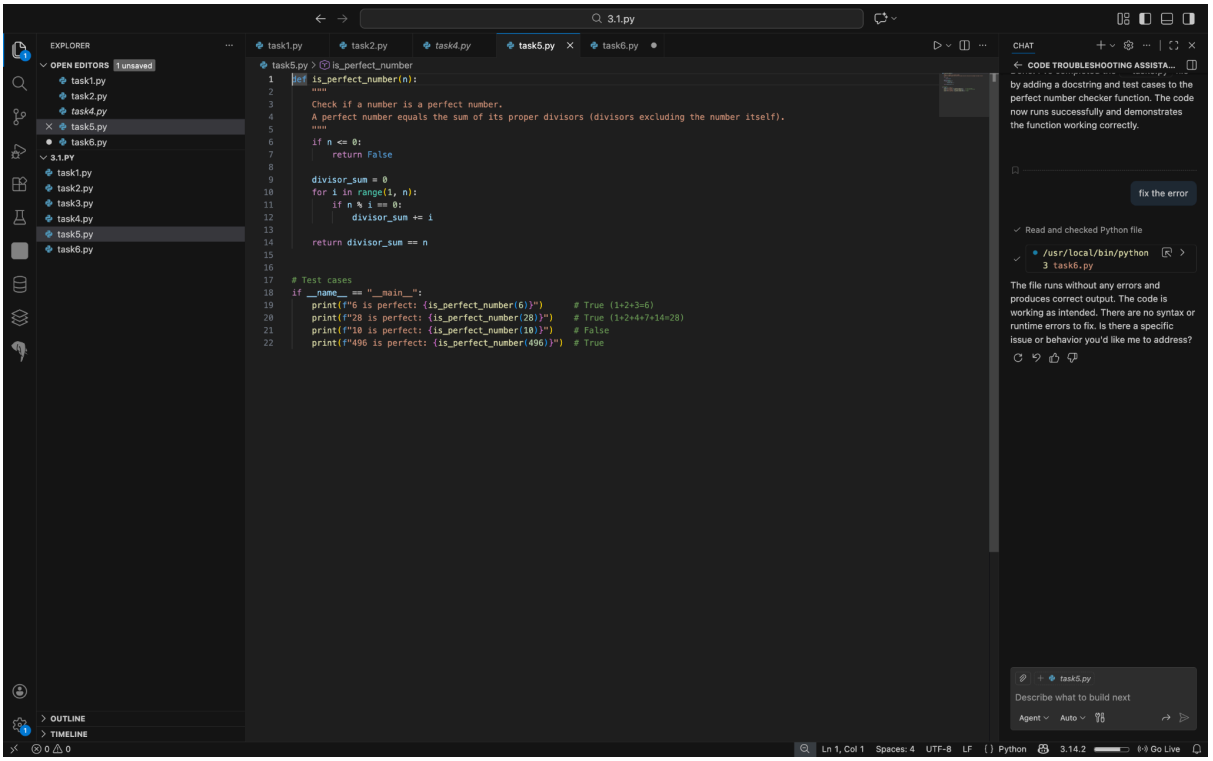
```
/usr/local/bin/python3 /Users/syedsufiyan/3.1.py/task4.py
syedsufiyan@Syeds-MacBook-Air-3 3.1.py % /usr/local/bin/python3 /Users/syedsufiyan/3.1.py/ta
Input: 0 → Neither: Zero is neither prime nor composite
Input: 1 → Neither: One is neither prime nor composite
Input: 2 → Prime: Two is the only even prime number
Input: 3 → Prime: 3 is only divisible by 1 and itself
Input: 4 → Composite: 4 is divisible by 2
Input: 17 → Prime: 17 is only divisible by 1 and itself
Input: 20 → Composite: 20 is divisible by 2
Input: 97 → Prime: 97 is only divisible by 1 and itself
Input: 100 → Composite: 100 is divisible by 2
Input: -5 → Error: Input must be non-negative
Input: abc → Error: Input must be a valid integer
syedsufiyan@Syeds-MacBook-Air-3 3.1.py %
```

Observation:

The program accurately classifies numbers as prime, composite, or neither by applying proper input validation and an optimized primality test. It correctly handles special cases such as 0 and 1, identifies 2 as the only even prime number, and efficiently checks divisibility for other numbers up to their square root. Invalid and negative inputs are handled gracefully with meaningful error messages. The test results confirm that the program produces correct and explanatory classifications for a wide range of inputs.

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.
Task:
• Record the AI-generated code.
• Test the program with multiple inputs.
• Identify any missing conditions or inefficiencies in the logic

Code :



Output :

```
/usr/local/bin/python3 /Users/syedsufiyan/3.1.py/task5.py
syedsufiyan@Syeds-MacBook-Air-3 3.1.py % /usr/local/bin/python3 /Users/syedsufiyan/3.1.py/task5.py
6 is perfect: True
28 is perfect: True
10 is perfect: False
496 is perfect: True
syedsufiyan@Syeds-MacBook-Air-3 3.1.py %
```

Observation:

The program correctly determines whether a given number is a perfect number by calculating the sum of its proper divisors and comparing it with the original number. It properly excludes non-positive values and accurately identifies known perfect numbers such as 6, 28, and 496, while correctly rejecting non-perfect numbers like 10. The test cases confirm that the divisor-based logic works as expected, demonstrating correct use of loops and conditional statements.

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

Validation)

Write a few-shot prompt by providing multiple input-output

examples to guide the AI in 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

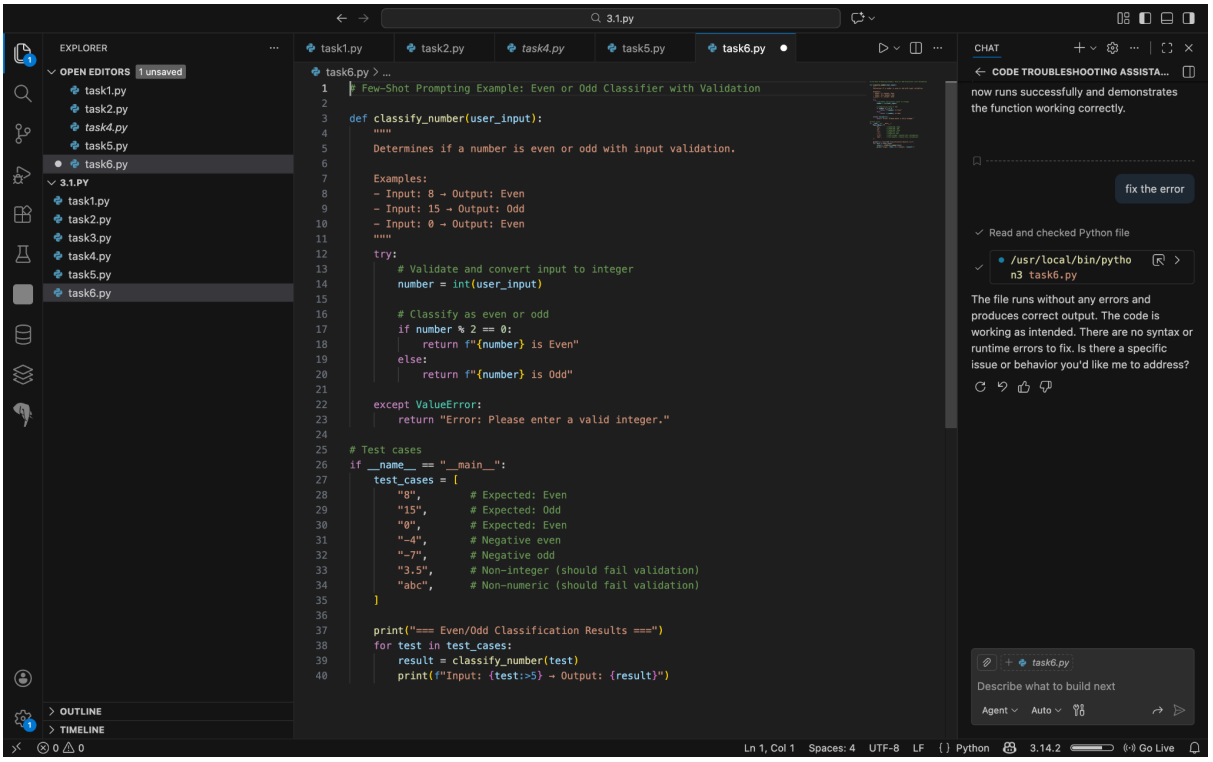
• Input: 0 → Output: Even

Task:

• Analyze how examples improve input handling and output clarity.

• Test the program with negative numbers and non-integer inputs.

Code :



Output :

```
/usr/local/bin/python3 /Users/syedsufiyan/3.1.py/task6.py
syedsufiyan@Syeds-MacBook-Air-3 3.1.py % /usr/local/bin/python3 /Users/syedsufiyan/3.1.py/t
ask6.py
=== Even/Odd Classification Results ===
Input:     8 → Output: 8 is Even
Input:    15 → Output: 15 is Odd
Input:     0 → Output: 0 is Even
Input:    -4 → Output: -4 is Even
Input:    -7 → Output: -7 is Odd
Input:   3.5 → Output: Error: Please enter a valid integer.
Input:   abc → Output: Error: Please enter a valid integer.
syedsufiyan@Syeds-MacBook-Air-3 3.1.py %
```

Observation :

The program correctly identifies even and odd numbers using few-shot prompting while validating user input. It handles positive numbers, negative numbers, and zero accurately. Invalid inputs such as floating-point values and non-numeric strings are safely rejected using exception handling, ensuring robust and error-free execution.