

AI Assisted Coding

Week2 – Monday

Name: Lalu prasad Aroori

Batch: 07

Roll No.: 2303A51948

Experiment – Prompt Engineering Techniques

Task Description

Design and refine prompts using different prompting strategies to generate Python programs for basic computational problems.

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.

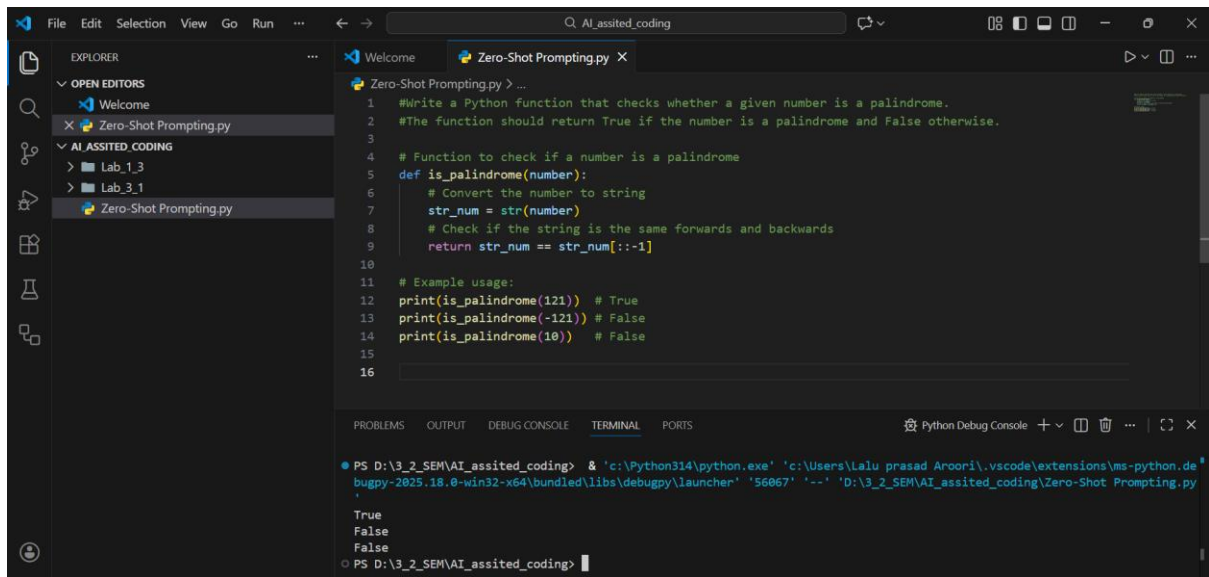
Task:

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

Prompt:

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

The function should return True if the number is a palindrome and False otherwise.



```
1 #Write a Python function that checks whether a given number is a palindrome.
2 #The function should return True if the number is a palindrome and False otherwise.
3
4 # Function to check if a number is a palindrome
5 def is_palindrome(number):
6     # Convert the number to string
7     str_num = str(number)
8     # Check if the string is the same forwards and backwards
9     return str_num == str_num[::-1]
10
11 # Example usage:
12 print(is_palindrome(121)) # True
13 print(is_palindrome(-121)) # False
14 print(is_palindrome(10)) # False
15
16
```

```
PS D:\3_2_SEM\AI_assited_coding> & 'c:\Python314\python.exe' 'c:\Users\Lalu prasad Aroori\vscode\extensions\ms-python.de
bugpy-2025.18.0-win32-x64\bundle\libs\debugpy\launcher' '56067' '--' 'D:\3_2_SEM\AI_assited_coding\Zero-Shot Prompting.py'
True
False
False
PS D:\3_2_SEM\AI_assited_coding>
```

Remarks

- The code correctly identifies palindrome numbers.
- **Limitation:** Does not handle negative numbers (e.g., -121).
- Zero-shot prompting works but may miss edge-case handling.
- Logic is clear but could be improved with input validation.

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.

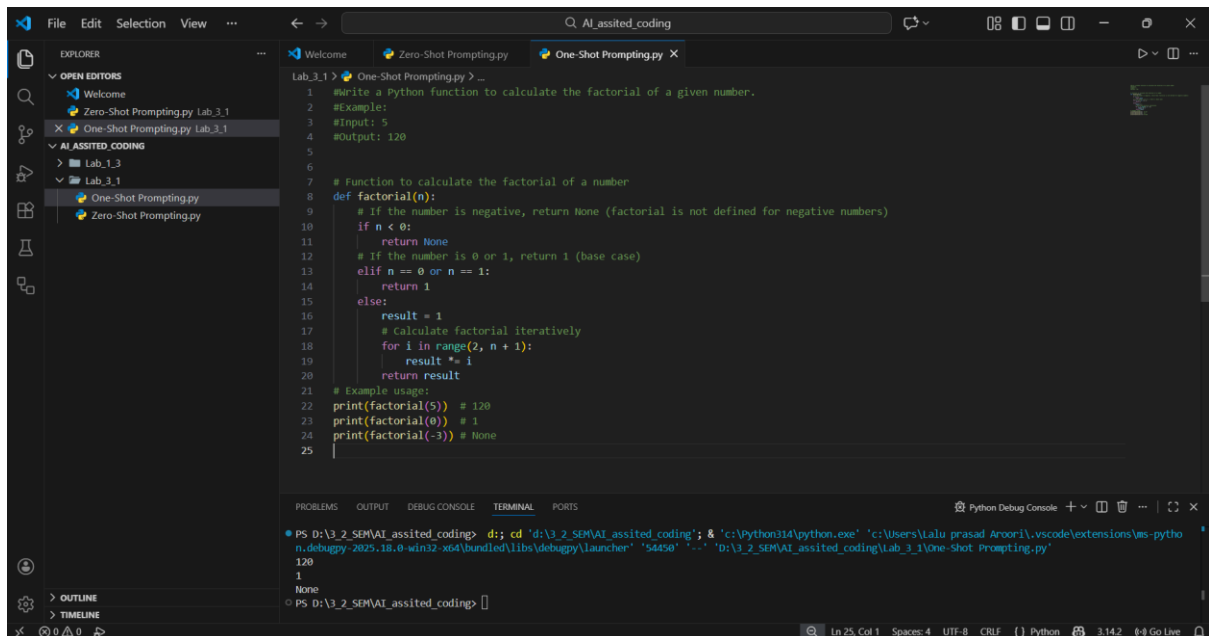
Prompt:

#Write a Python function to calculate the factorial of a given number.

Example:

Input: 5

Output: 120



```
1 #Write a Python function to calculate the factorial of a given number.
2 #Example:
3 #Input: 5
4 #Output: 120
5
6
7 # Function to calculate the factorial of a number
8 def factorial(n):
9     # If the number is negative, return None (factorial is not defined for negative numbers)
10    if n < 0:
11        return None
12    # If the number is 0 or 1, return 1 (base case)
13    elif n == 0 or n == 1:
14        return 1
15    else:
16        result = 1
17        # calculate factorial iteratively
18        for i in range(2, n + 1):
19            result *= i
20        return result
21
22 # Example usage:
23 print(factorial(5)) # 120
24 print(factorial(0)) # 1
25 print(factorial(-3)) # None
```

```
PS D:\3_2_SEM\AI_assited_coding> cd 'D:\3_2_SEM\AI_assited_coding'; & 'c:\Python314\python.exe' 'c:\Users\Lalu prasad Aroor\vscode\extensions\ms-python-debugpy-2025.18.0-win32-x64\bundle\libs\debugpy\launcher' '54450' '-...' 'D:\3_2_SEM\AI_assited_coding\Lab_3_1\One-Shot Prompting.py'
1
120
None
PS D:\3_2_SEM\AI_assited_coding>
```

Remarks

- One-shot prompting improved clarity and correctness.
- Handles negative numbers properly.
- Compared to zero-shot, the logic is more structured.
- Demonstrates how **examples guide AI behavior**.

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.

Prompt:

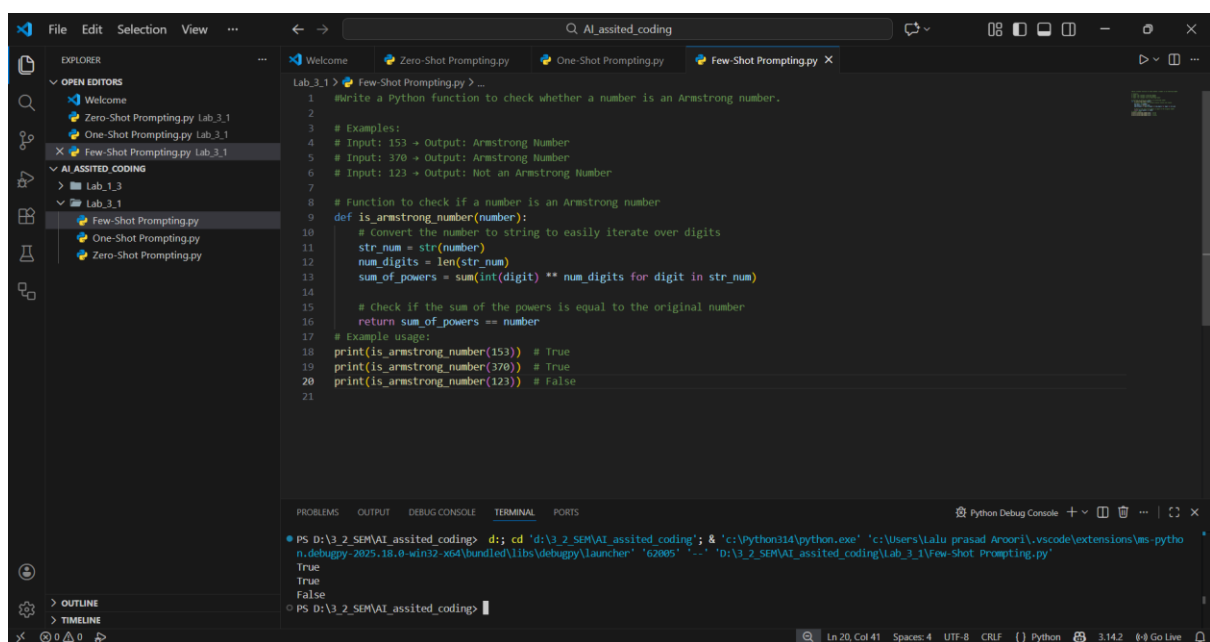
#Write a Python function to check whether a number is an Armstrong number.

Examples:

Input: 153 → Output: Armstrong Number

Input: 370 → Output: Armstrong Number

Input: 123 → Output: Not an Armstrong Number



```
File Edit Selection View ... < -> AI_assited_coding
EXPLORER
  OPEN EDITORS
    Welcome
    Zero-Shot Prompting.py Lab_3.1
    One-Shot Prompting.py Lab_3.1
    Few-Shot Prompting.py Lab_3.1
  AI ASSITED CODING
    Lab_1.3
    Lab_3.1
      Few-Shot Prompting.py
      One-Shot Prompting.py
      Zero-Shot Prompting.py
  OUTLINE
  TIMELINE

Lab_3.1 > Few-Shot Prompting.py > ...
1 #write a Python function to check whether a number is an Armstrong number.
2
3 # Examples:
4 # Input: 153 → Output: Armstrong Number
5 # Input: 370 → Output: Armstrong Number
6 # Input: 123 → Output: Not an Armstrong Number
7
8 # Function to check if a number is an Armstrong number
9 def is_armstrong_number(number):
10     # Convert the number to string to easily iterate over digits
11     str_num = str(number)
12     num_digits = len(str_num)
13     sum_of_powers = sum(int(digit) ** num_digits for digit in str_num)
14
15     # Check if the sum of the powers is equal to the original number
16     return sum_of_powers == number
17
18 # Example usage:
19 print(is_armstrong_number(153)) # True
20 print(is_armstrong_number(370)) # True
21 print(is_armstrong_number(123)) # False
22

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python Debug Console
PS D:\3_2_SEMVAI_assited_coding> d; cd 'd:\3_2_SEMVAI_assited_coding'; & 'c:\Python314\python.exe' 'c:\Users\Lalu prasad Aroori\vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundle\libs\debugpy\launcher' '62805' '-' 'D:\3_2_SEMVAI_assited_coding\Lab_3_1\Few-Shot Prompting.py'
True
True
False
PS D:\3_2_SEMVAI_assited_coding> |
```

Remarks

- Few-shot prompting resulted in clean and accurate logic.
- Works for boundary values like 0.
- Code readability and correctness improved.
- Demonstrates the effectiveness of multiple examples.

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:

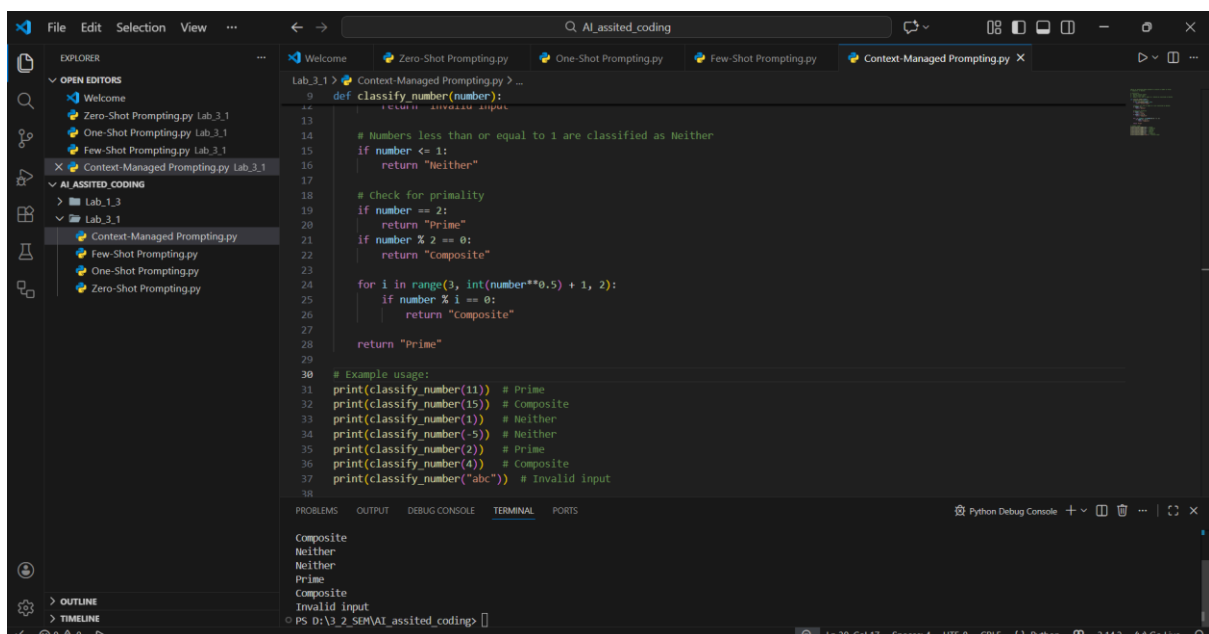
- Ensure proper input validation.
- Optimize the logic for efficiency.
- Compare the output with earlier prompting strategies.

Prompt:

#Write an optimized Python program to classify a number as Prime, Composite, or Neither.

Constraints:

- Handle invalid inputs
- Use efficient logic
- Numbers less than or equal to 1 should be classified as Neither



The screenshot shows a VS Code editor with a file named 'Context-Managed Prompting.py' open. The code defines a function 'classify_number(number)' that classifies a number as 'Prime', 'Composite', or 'Neither'. The function includes input validation for numbers less than or equal to 1 and an efficient primality check using a loop from 3 to the square root of the number. Below the function, there is an example usage section with test cases. The terminal at the bottom shows the output of the program for various inputs.

```
def classify_number(number):  
    # Numbers less than or equal to 1 are classified as Neither  
    if number <= 1:  
        return "Neither"  
  
    # Check for primality  
    if number == 2:  
        return "Prime"  
    if number % 2 == 0:  
        return "Composite"  
  
    for i in range(3, int(number**0.5) + 1, 2):  
        if number % i == 0:  
            return "Composite"  
  
    return "Prime"  
  
# Example usage:  
print(classify_number(11)) # Prime  
print(classify_number(15)) # Composite  
print(classify_number(1)) # Neither  
print(classify_number(-5)) # Neither  
print(classify_number(2)) # Prime  
print(classify_number(4)) # Composite  
print(classify_number("abc")) # Invalid input
```

Terminal Output:

```
Composite  
Neither  
Neither  
Prime  
Composite  
Invalid input
```

Remarks

- Context-managed prompting produces optimized and robust code.
- Efficient logic using square-root optimization.
- Proper input validation included.
- Best quality among all prompting techniques.

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

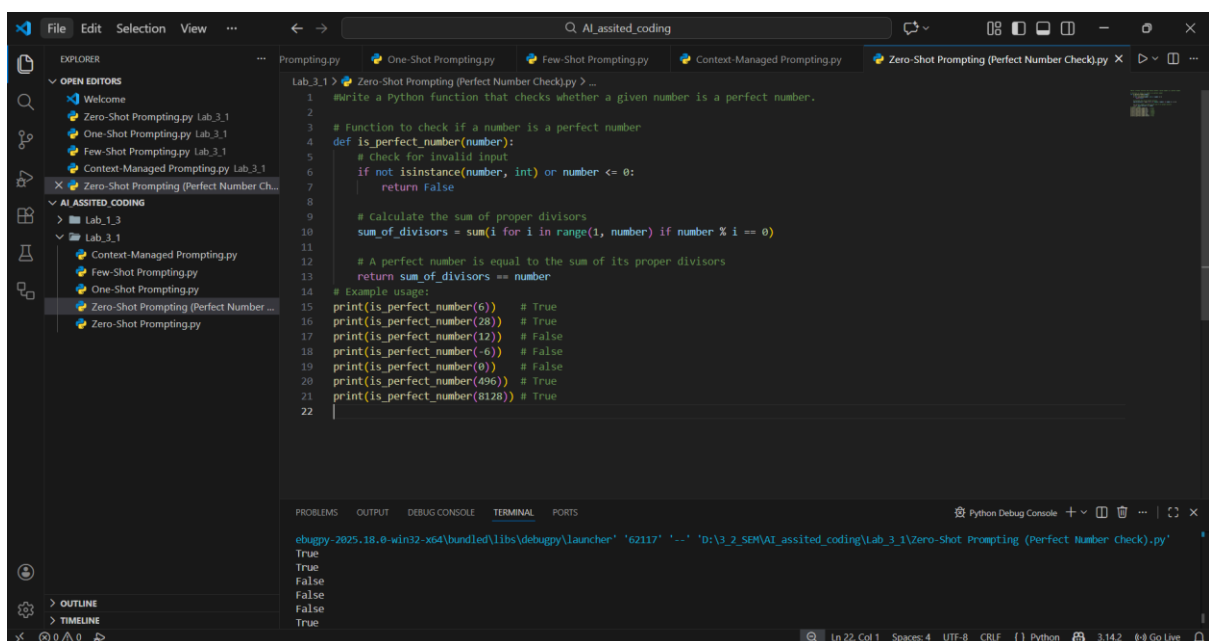
Write a zero-shot prompt (without providing any examples) to generate a Python function that checks whether a given number is a perfect number.

Task:

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

Prompt:

#Write a Python function that checks whether a given number is a perfect number.



```
1 #write a Python function that checks whether a given number is a perfect number.
2
3 # Function to check if a number is a perfect number
4 def is_perfect_number(number):
5     # Check for invalid input
6     if not isinstance(number, int) or number <= 0:
7         return False
8
9     # calculate the sum of proper divisors
10    sum_of_divisors = sum(i for i in range(1, number) if number % i == 0)
11
12    # A perfect number is equal to the sum of its proper divisors
13    return sum_of_divisors == number
14
15    # Example usage:
16    print(is_perfect_number(6))    # True
17    print(is_perfect_number(28))  # True
18    print(is_perfect_number(12))  # False
19    print(is_perfect_number(-6))  # False
20    print(is_perfect_number(0))   # False
21    print(is_perfect_number(496)) # True
22    print(is_perfect_number(8128)) # True
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Python Debug Console

```
ebuggy-2025.18.0-win32-x64\bundle\libs\debugpy\launcher '62117' '-' 'D:\3_2_SEM\AI_assisted_coding\Lab_3_1\Zero-Shot Prompting (Perfect Number Check).py'
True
True
False
False
False
True
```

Ln 22, Col 1 Spaces: 4 UTF-8 CRLF Python 3.142 0-0 Go Live

Remarks

- Works correctly for perfect numbers.
- **Inefficient** for large numbers (checks all values till n).
- Zero-shot lacks optimization like checking till $n/2$.
- Improvement possible with better constraints.

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.

Prompt:

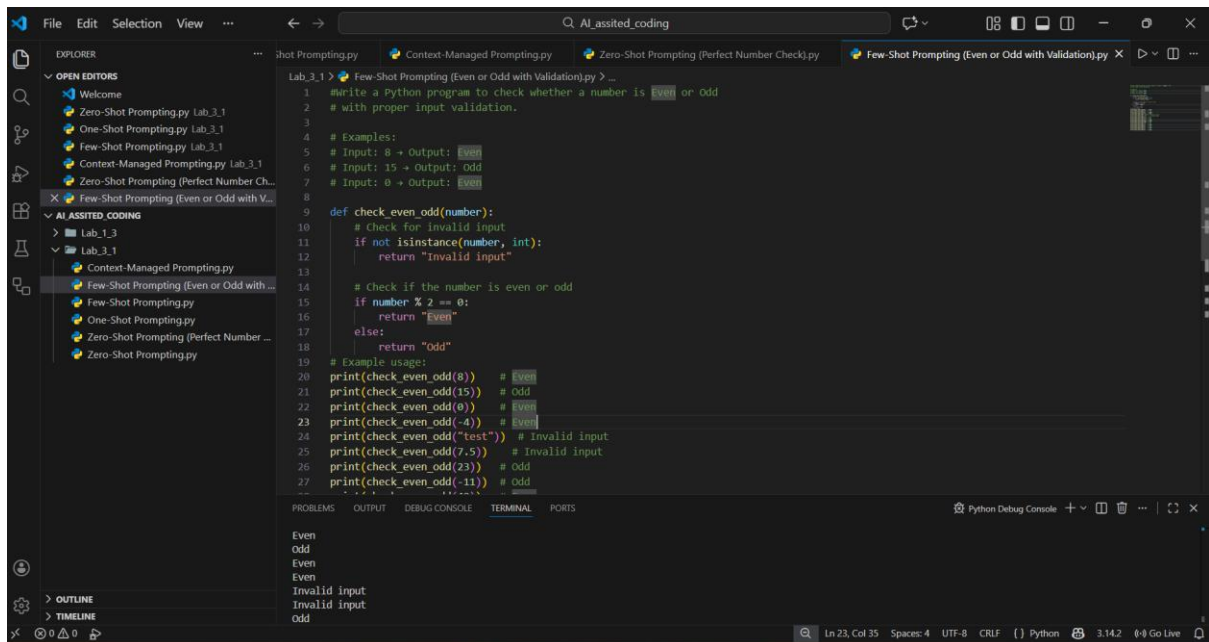
#Write a Python program to check whether a number is Even or Odd with proper input validation.

Examples:

Input: 8 → Output: Even

Input: 15 → Output: Odd

Input: 0 → Output: Even



```
1 # Write a Python program to check whether a number is Even or Odd
2 # with proper input validation.
3
4 # Examples:
5 # Input: 8 → Output: Even
6 # Input: 15 → Output: Odd
7 # Input: 0 → Output: Even
8
9 def check_even_odd(number):
10     # Check for invalid input
11     if not isinstance(number, int):
12         return "Invalid input"
13
14     # Check if the number is even or odd
15     if number % 2 == 0:
16         return "Even"
17     else:
18         return "Odd"
19
20 # Example usage:
21 print(check_even_odd(8)) # Even
22 print(check_even_odd(15)) # Odd
23 print(check_even_odd(0)) # Even
24 print(check_even_odd(-4)) # Even
25 print(check_even_odd("test")) # Invalid input
26 print(check_even_odd(7.5)) # Invalid input
27 print(check_even_odd(23)) # Odd
28 print(check_even_odd(-11)) # Odd
```

Even
Odd
Even
Even
Invalid input
Invalid input
Odd

Remarks

- Few-shot prompting significantly improved input validation.
- Handles negative numbers and non-integers correctly.
- Output clarity is high.
- Demonstrates strong influence of examples on AI output.