Assignment – 3.1

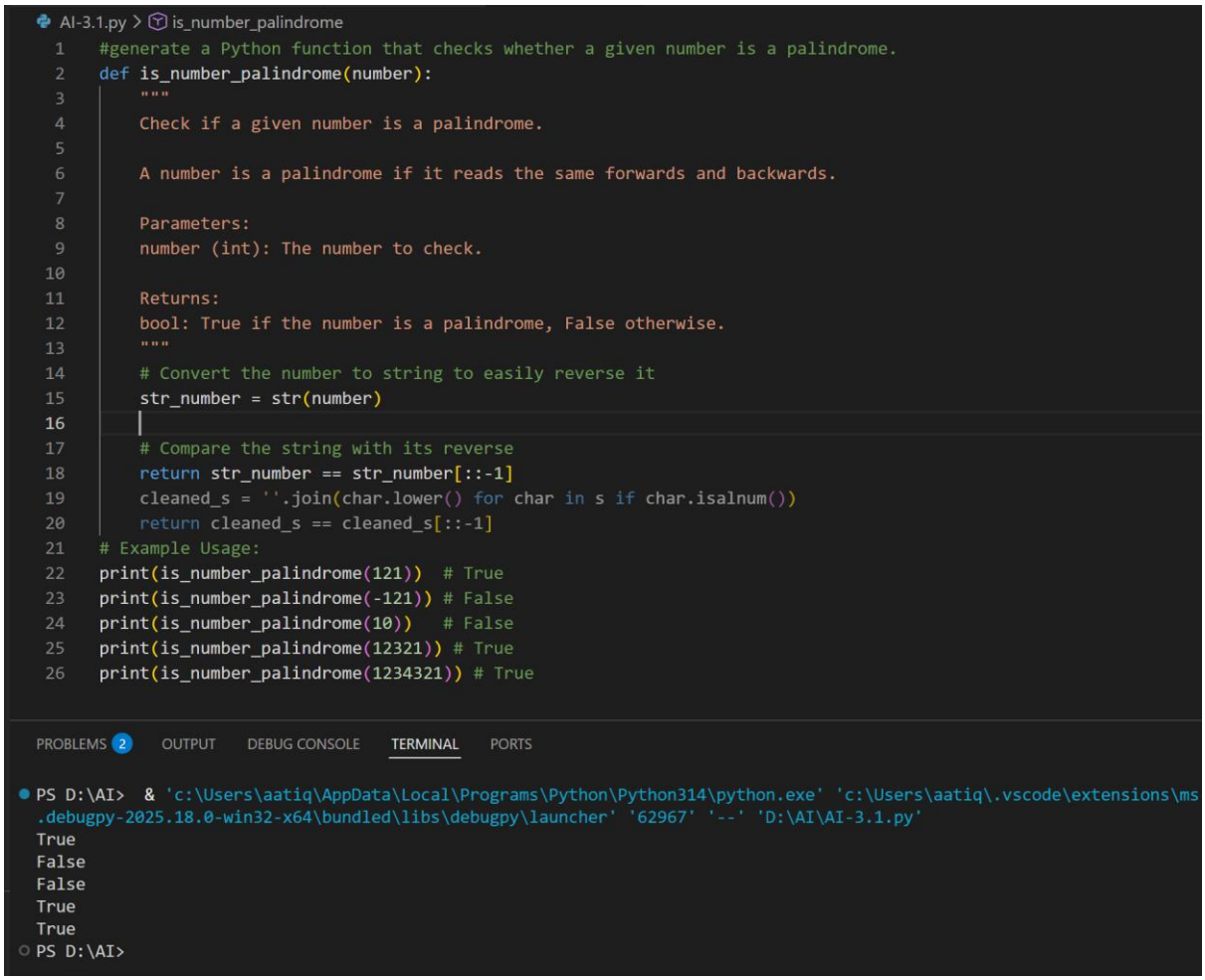
2303A51860

Task 1: Zero-Shot Prompting (Palindrome Number Program)

PROMPT :

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

CODE and OUTPUT :



```
AI-3.1.py > is_number_palindrome
#generate a Python function that checks whether a given number is a palindrome.
def is_number_palindrome(number):
    """
    Check if a given number is a palindrome.

    A number is a palindrome if it reads the same forwards and backwards.

    Parameters:
    number (int): The number to check.

    Returns:
    bool: True if the number is a palindrome, False otherwise.
    """
    # Convert the number to string to easily reverse it
    str_number = str(number)

    # Compare the string with its reverse
    return str_number == str_number[::-1]
    cleaned_s = ''.join(char.lower() for char in s if char.isalnum())
    return cleaned_s == cleaned_s[::-1]
# Example Usage:
print(is_number_palindrome(121))  # True
print(is_number_palindrome(-121)) # False
print(is_number_palindrome(10))   # False
print(is_number_palindrome(12321)) # True
print(is_number_palindrome(1234321)) # True
```

```
PROBLEMS 2  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS D:\AI>  & 'c:\Users\aatiq\AppData\Local\Programs\Python\Python314\python.exe' 'c:\Users\aatiq\.vscode\extensions\ms
.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '62967' '--' 'D:\AI\AI-3.1.py'
True
False
False
True
True
PS D:\AI>
```

Justification :

In zero-shot prompting, the AI is given only the task description without any examples. This tests the AI's general understanding of the problem. When asked to generate a palindrome-checking function, the AI usually produces a basic solution by reversing the number or converting it to a string. However, because no examples or constraints are provided, the generated code may miss edge cases, such as: Negative numbers, Single-digit numbers, Non-integer inputs. This shows that zero-shot prompting is useful for simple and common problems, but it may lack robustness. The experiment demonstrates that without context or

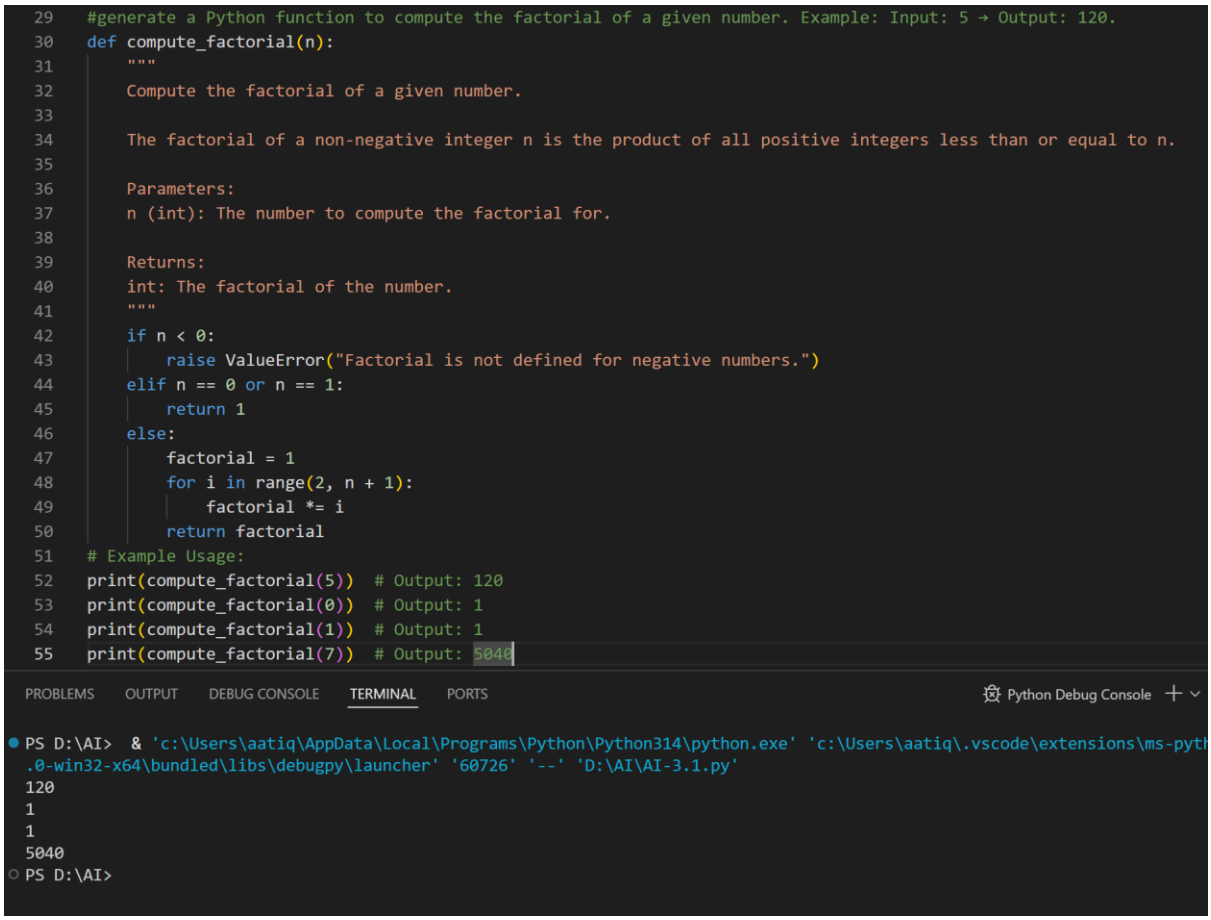
examples, AI-generated code may be logically correct but not fully reliable for real-world use.

Task 2: One-Shot Prompting (Factorial Calculation)

PROMPT :

Generate a Python function to compute the factorial of a given number. Example: Input: 5 → Output: 120.

CODE and OUTPUT :



```python
#generate a Python function to compute the factorial of a given number. Example: Input: 5 → Output: 120.
def compute_factorial(n):
    """
    Compute the factorial of a given number.

    The factorial of a non-negative integer n is the product of all positive integers less than or equal to n.

    Parameters:
    n (int): The number to compute the factorial for.

    Returns:
    int: The factorial of the number.
    """
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers.")
    elif n == 0 or n == 1:
        return 1
    else:
        factorial = 1
        for i in range(2, n + 1):
            factorial *= i
        return factorial
# Example Usage:
print(compute_factorial(5))  # Output: 120
print(compute_factorial(0))  # Output: 1
print(compute_factorial(1))  # Output: 1
print(compute_factorial(7))  # Output: 5040
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                Python Debug Console

PS D:\AI>  & 'c:\Users\aatiq\AppData\Local\Programs\Python\Python314\python.exe' 'c:\Users\aatiq\.vscode\extensions\ms-pyth
.0-win32-x64\bundled\libs\debugpy\launcher' '60726' '--' 'D:\AI\AI-3.1.py'
120
1
1
5040
PS D:\AI>
```

Justification :

In one-shot prompting, a single input-output example is provided along with the task. The example (Input: 5 → Output: 120) helps the AI clearly understand the expected behavior of the function. Compared to zero-shot prompting, the one-shot approach results in: Clearer logic, Better handling of loops or recursion, Improved correctness. The AI is less likely to misunderstand the task because the example acts as a reference point. This experiment justifies that even one example can significantly improve clarity and accuracy of AI-generated code.

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

PROMPT :

Generating a Python function to check whether a given number is an Armstrong number.

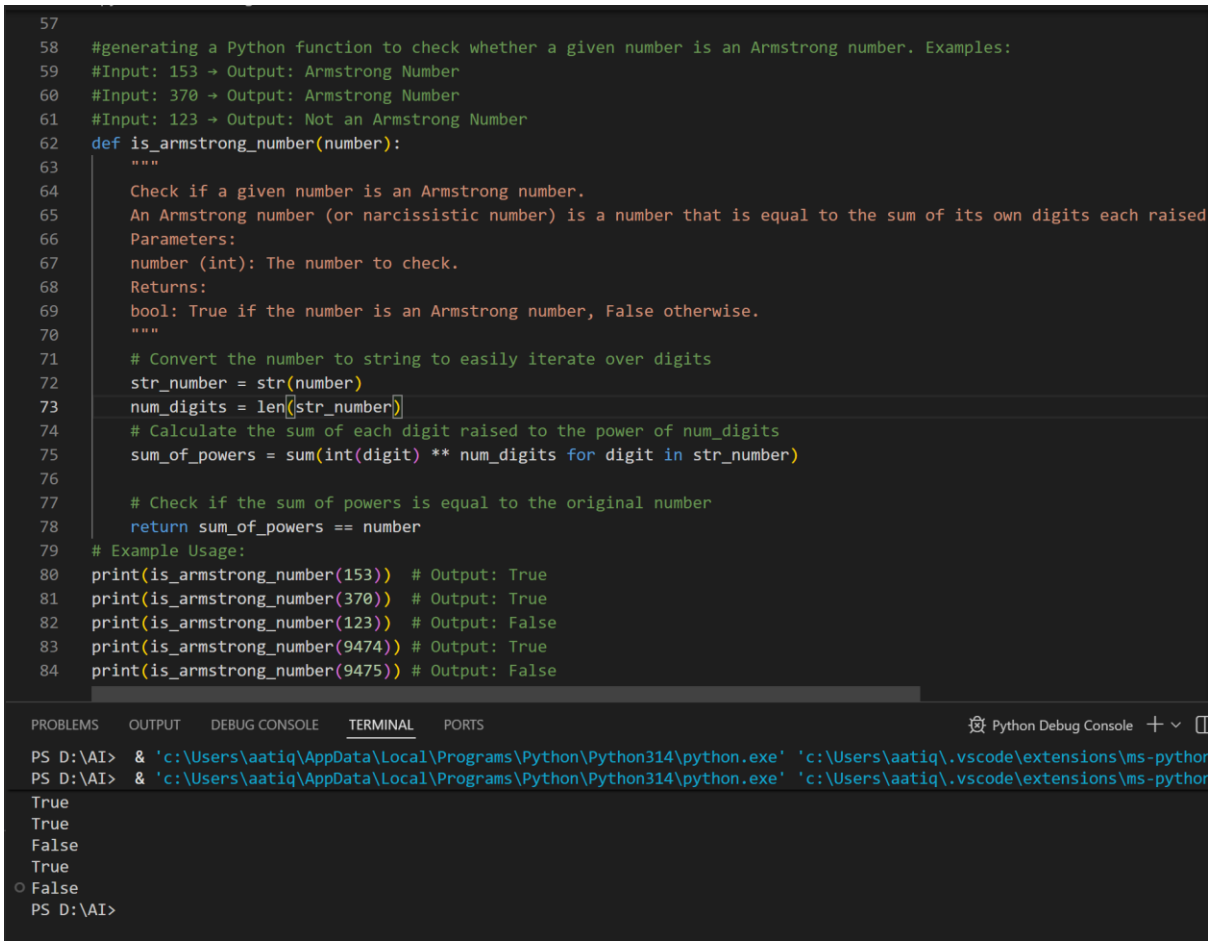
Examples:

• Input: 153 → Output: Armstrong Number

• Input: 370 → Output: Armstrong Number

• Input: 123 → Output: Not an Armstrong Number

CODE and OUTPUT :



```
#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
def is_armstrong_number(number):
    """
    Check if a given number is an Armstrong number.
    An Armstrong number (or narcissistic number) is a number that is equal to the sum of its own digits each raised
    Parameters:
    number (int): The number to check.
    Returns:
    bool: True if the number is an Armstrong number, False otherwise.
    """
    # Convert the number to string to easily iterate over digits
    str_number = str(number)
    num_digits = len(str_number)
    # Calculate the sum of each digit raised to the power of num_digits
    sum_of_powers = sum(int(digit) ** num_digits for digit in str_number)

    # Check if the sum of powers is equal to the original number
    return sum_of_powers == number
# Example Usage:
print(is_armstrong_number(153))  # Output: True
print(is_armstrong_number(370))  # Output: True
print(is_armstrong_number(123))  # Output: False
print(is_armstrong_number(9474)) # Output: True
print(is_armstrong_number(9475)) # Output: False
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS        Python Debug Console
PS D:\AI>  & 'c:\Users\aatiq\AppData\Local\Programs\Python\Python314\python.exe' 'c:\Users\aatiq\.vscode\extensions\ms-python
PS D:\AI>  & 'c:\Users\aatiq\AppData\Local\Programs\Python\Python314\python.exe' 'c:\Users\aatiq\.vscode\extensions\ms-python
True
True
False
True
False
PS D:\AI>
```

Justification :

Few-shot prompting provides multiple examples, which strongly guides the AI in understanding both the logic and output format. With several Armstrong and non-Armstrong examples, the AI can correctly infer: How digits are processed, How powers are calculated, How decisions are made As a result, the generated code is usually: Well-structured, More accurate, Consistent with mathematical definitions Testing with boundary values and invalid inputs further shows that few-shot prompting improves generalization and reduces logical errors. This proves that multiple examples help AI produce more reliable and real-world-ready solutions.

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

PROMPT :

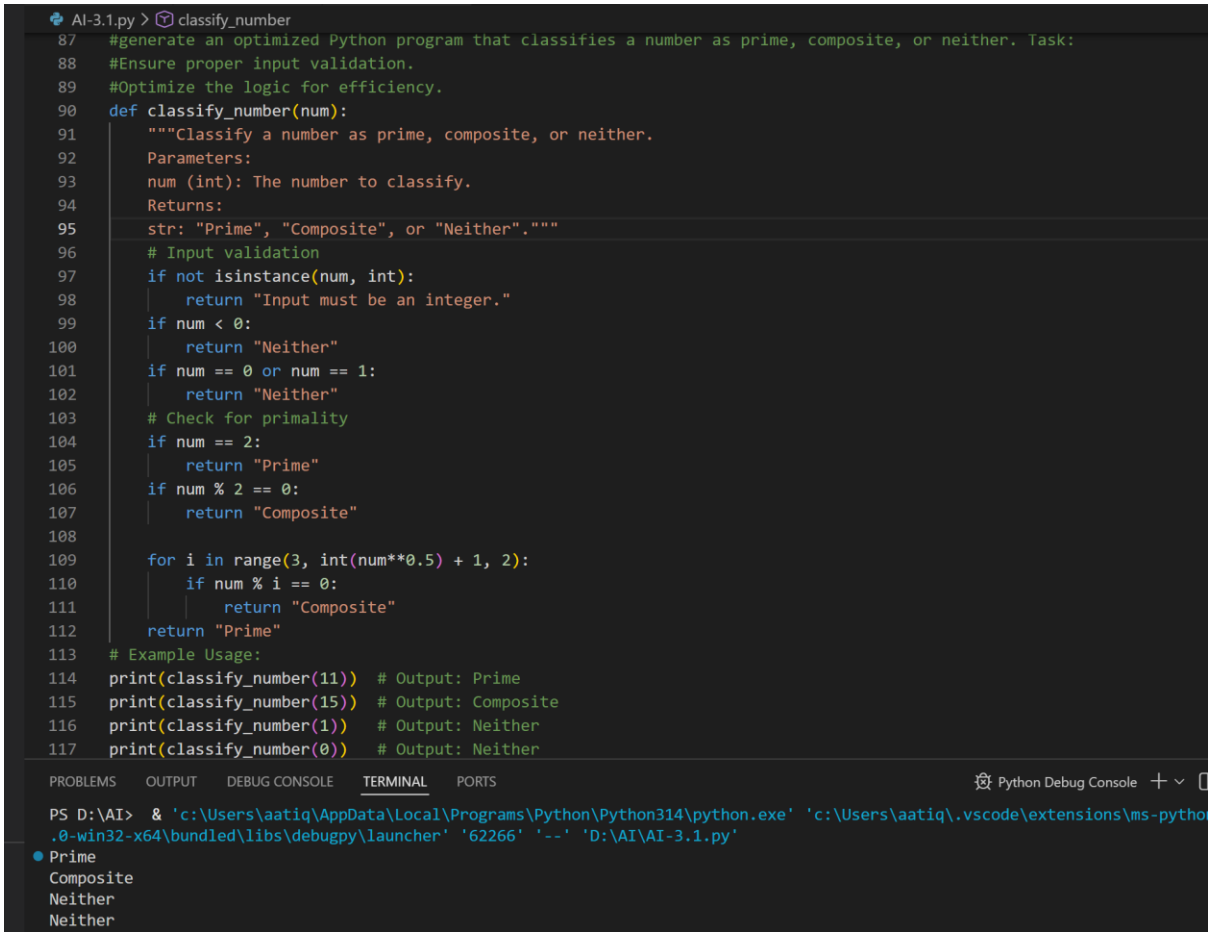
Generate an optimized Python program that classifies a number as prime, composite, or neither.

Task:

• Ensure proper input validation.

• Optimize the logic for efficiency.

CODE and OUTPUT :



Justification :

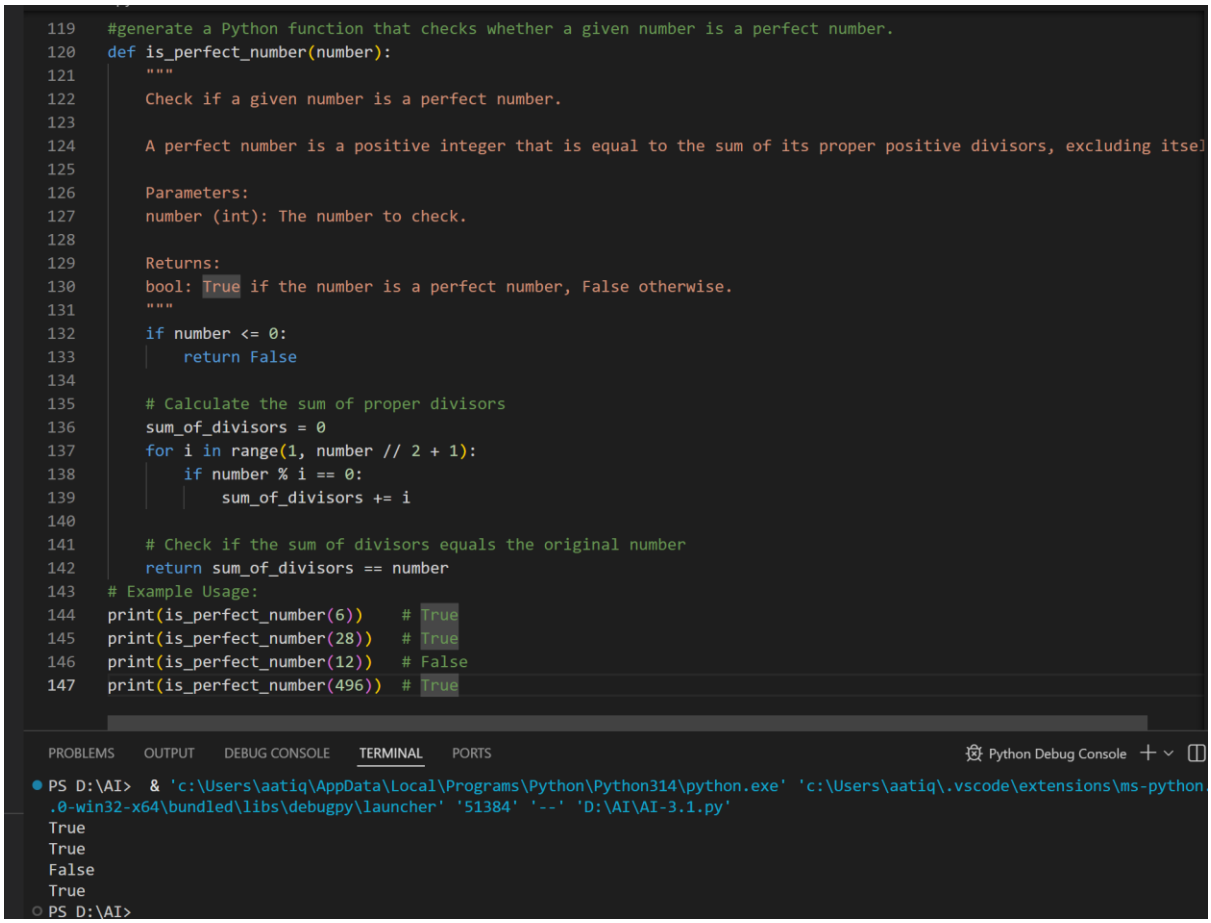
Context-managed prompting includes clear instructions, constraints, and expectations. By specifying input validation, optimization, and classification rules, the AI generates a more efficient and complete program. This approach ensures: Proper handling of edge cases (e.g., 0, 1, negative numbers) Optimized logic (checking divisibility up to √n) Clear classification output Compared to earlier prompting strategies, context-managed prompting produces high-quality, optimized, and professional-level code. This justifies that providing context and constraints is essential for complex or performance-sensitive tasks.

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

PROMPT :

Generate a Python function that checks whether a given number is a perfect number.

CODE and OUTPUT :



```python
#generate a Python function that checks whether a given number is a perfect number.
def is_perfect_number(number):
    """
    Check if a given number is a perfect number.

    A perfect number is a positive integer that is equal to the sum of its proper positive divisors, excluding itsel

    Parameters:
    number (int): The number to check.

    Returns:
    bool: True if the number is a perfect number, False otherwise.
    """
    if number <= 0:
        return False

    # Calculate the sum of proper divisors
    sum_of_divisors = 0
    for i in range(1, number // 2 + 1):
        if number % i == 0:
            sum_of_divisors += i

    # Check if the sum of divisors equals the original number
    return sum_of_divisors == number
# Example Usage:
print(is_perfect_number(6))    # True
print(is_perfect_number(28))   # True
print(is_perfect_number(12))   # False
print(is_perfect_number(496))  # True
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS        Python Debug Console
PS D:\AI>  & 'c:\Users\aatiq\AppData\Local\Programs\Python\Python314\python.exe' 'c:\Users\aatiq\.vscode\extensions\ms-python.
.0-win32-x64\bundled\libs\debugpy\launcher' '51384' '--' 'D:\AI\AI-3.1.py'
True
True
False
True
PS D:\AI>
```

Justification :

Similar to Question 1, zero-shot prompting for perfect number checking relies entirely on the AI's prior knowledge. The generated code often correctly checks divisors and sums them. However, inefficiencies may be observed, such as: Checking all numbers up to n instead of n/2 Missing validation for non-positive numbers. This experiment highlights that while zero-shot prompting can generate a working solution, it may not be optimized or fully correct. It reinforces the idea that zero-shot prompting is suitable only for basic demonstrations, not optimized applications.

Task 6: Few-Shot Prompting (Even or Odd Classification with Validation)

PROMPT :

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

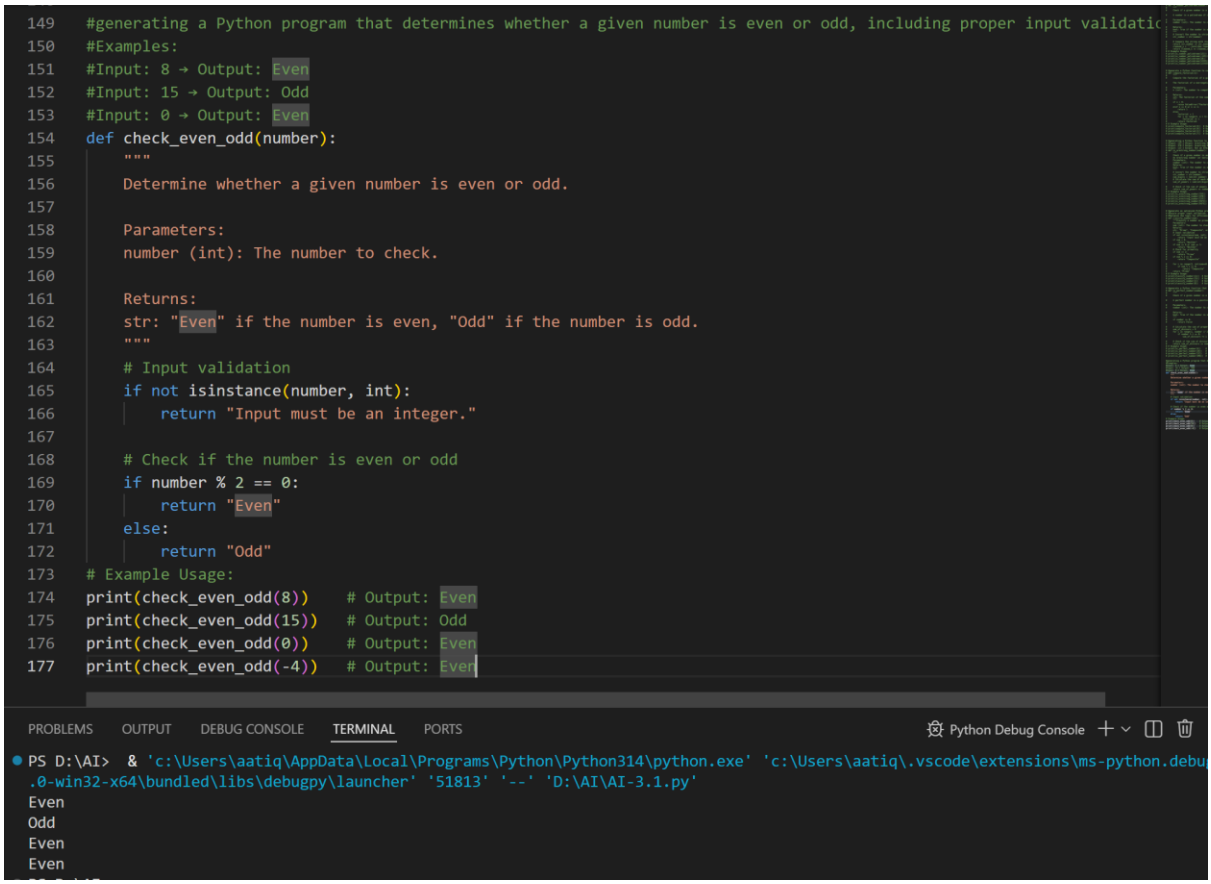
Examples:

• Input: 8 → Output: Even

• Input: 15 → Output: Odd

• Input: 0 → Output: Even

CODE and OUTPUT :



```
#generating a Python program that determines whether a given number is even or odd, including proper input validatic
#Examples:
#Input: 8 → Output: Even
#Input: 15 → Output: Odd
#Input: 0 → Output: Even
def check_even_odd(number):
    """
    Determine whether a given number is even or odd.

    Parameters:
    number (int): The number to check.

    Returns:
    str: "Even" if the number is even, "Odd" if the number is odd.
    """
    # Input validation
    if not isinstance(number, int):
        return "Input must be an integer."

    # Check if the number is even or odd
    if number % 2 == 0:
        return "Even"
    else:
        return "Odd"
# Example Usage:
print(check_even_odd(8))    # Output: Even
print(check_even_odd(15))   # Output: Odd
print(check_even_odd(0))    # Output: Even
print(check_even_odd(-4))   # Output: Even
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS        Python Debug Console
PS D:\AI>  & 'c:\Users\aatiq\AppData\Local\Programs\Python\Python314\python.exe' 'c:\Users\aatiq\.vscode\extensions\ms-python.debu
.0-win32-x64\bundled\libs\debugpy\launcher' '51813' '--' 'D:\AI\AI-3.1.py'
Even
Odd
Even
Even
```

Justification :

By providing multiple examples, including edge cases like zero, few-shot prompting helps the AI understand: Expected outputs Input validation requirements, Handling of different numerical cases. The generated program usually includes: Clear conditional checks, Proper output messages, Improved handling of negative numbers. When tested with non-integer inputs, the AI-generated code often performs better than zero-shot solutions. This proves that few-shot prompting significantly improves input handling, output clarity, and robustness.