Assignment – 3.1

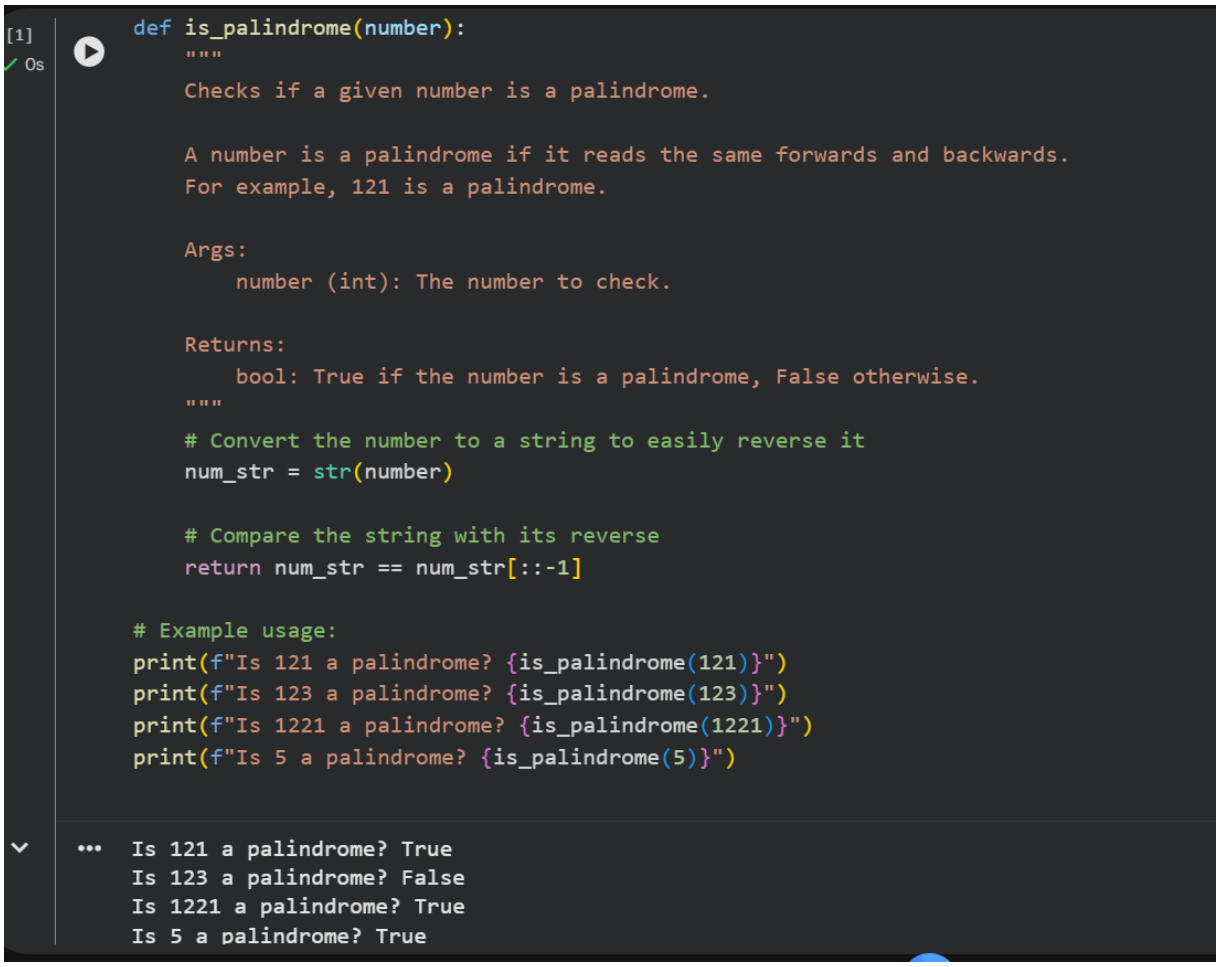
2303A510i0

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 :



```
[1]
✓ 0s
def is_palindrome(number):
    """
    Checks if a given number is a palindrome.

    A number is a palindrome if it reads the same forwards and backwards.
    For example, 121 is a palindrome.

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

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

    # Compare the string with its reverse
    return num_str == num_str[::-1]

# Example usage:
print(f"Is 121 a palindrome? {is_palindrome(121)}")
print(f"Is 123 a palindrome? {is_palindrome(123)}")
print(f"Is 1221 a palindrome? {is_palindrome(1221)}")
print(f"Is 5 a palindrome? {is_palindrome(5)}")

...
Is 121 a palindrome? True
Is 123 a palindrome? False
Is 1221 a palindrome? True
Is 5 a palindrome? True
```

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 palindromechecking 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, Noninteger 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 :

```
def factorial(n):
    """
    Computes the factorial of a non-negative integer.

    The factorial of a non-negative integer n, denoted by n!, is the
    product of all positive integers less than or equal to n.
    By convention, 0! = 1.

    Args:
        n (int): The non-negative integer for which to compute the factorial.

    Returns:
        int: The factorial of n.

    Raises:
        ValueError: If n is a negative number.
    """
    if not isinstance(n, int) or n < 0:
        raise ValueError("Factorial is only defined for non-negative integers.")

    if n == 0:
        return 1
    else:
        result = 1
        for i in range(1, n + 1):
            result *= i
        return result

# Example usage:
print(f"Factorial of 5 is: {factorial(5)}")
print(f"Factorial of 0 is: {factorial(0)}")
print(f"Factorial of 7 is: {factorial(7)}")
```

```
Factorial of 5 is: 120
Factorial of 0 is: 1
Factorial of 7 is: 5040
```

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 AIgenerated code.

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

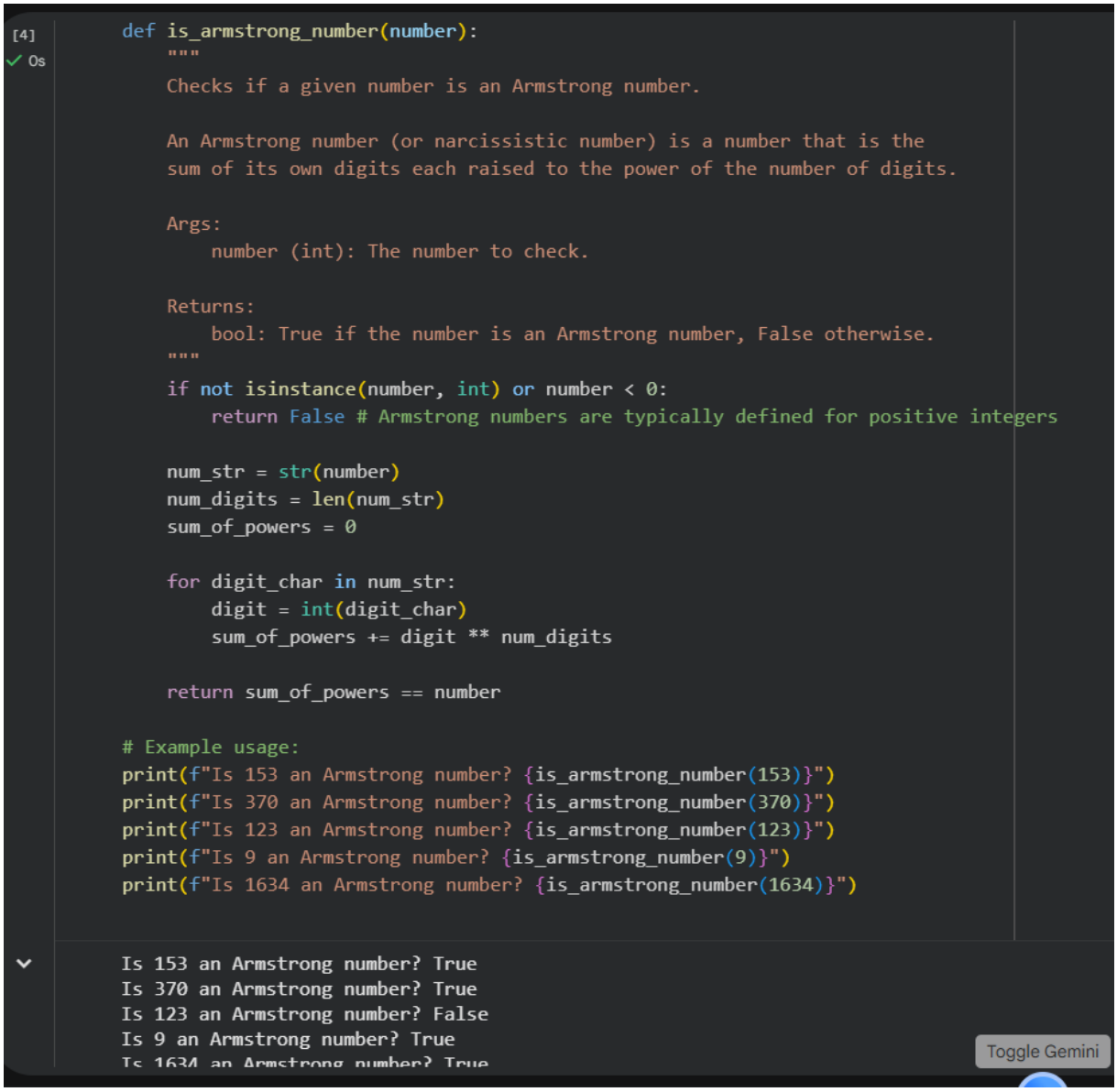
PROMPT :

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 :**



Justification :

Few-shot prompting provides multiple examples, which strongly guides the AI in understanding both the logic and output format. With several Armstrong and nonArmstrong 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: Wellstructured, 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)

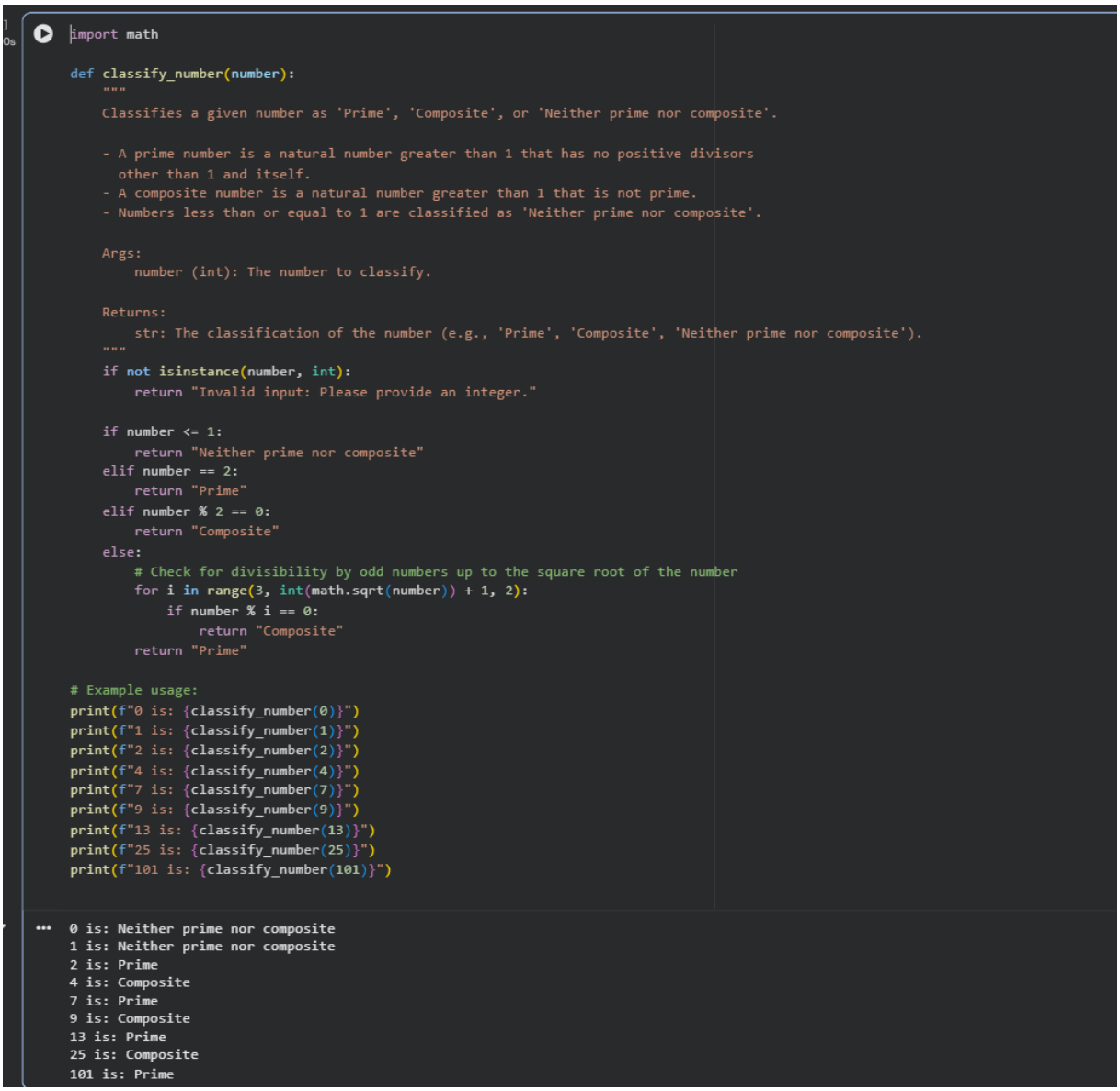
PROMPT :

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 :



```python
import math

def classify_number(number):
    """
    Classifies a given number as 'Prime', 'Composite', or 'Neither prime nor composite'.

    - A prime number is a natural number greater than 1 that has no positive divisors
      other than 1 and itself.
    - A composite number is a natural number greater than 1 that is not prime.
    - Numbers less than or equal to 1 are classified as 'Neither prime nor composite'.

    Args:
        number (int): The number to classify.

    Returns:
        str: The classification of the number (e.g., 'Prime', 'Composite', 'Neither prime nor composite').
    """
    if not isinstance(number, int):
        return "Invalid input: Please provide an integer."

    if number <= 1:
        return "Neither prime nor composite"
    elif number == 2:
        return "Prime"
    elif number % 2 == 0:
        return "Composite"
    else:
        # Check for divisibility by odd numbers up to the square root of the number
        for i in range(3, int(math.sqrt(number)) + 1, 2):
            if number % i == 0:
                return "Composite"
        return "Prime"

# Example usage:
print(f"0 is: {classify_number(0)}")
print(f"1 is: {classify_number(1)}")
print(f"2 is: {classify_number(2)}")
print(f"4 is: {classify_number(4)}")
print(f"7 is: {classify_number(7)}")
print(f"9 is: {classify_number(9)}")
print(f"13 is: {classify_number(13)}")
print(f"25 is: {classify_number(25)}")
print(f"101 is: {classify_number(101)}")
```

```
0 is: Neither prime nor composite
1 is: Neither prime nor composite
2 is: Prime
4 is: Composite
7 is: Prime
9 is: Composite
13 is: Prime
25 is: Composite
101 is: Prime
```

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 :

```
def is_perfect_number(number):
    """
    Checks if a given number is a perfect number.

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

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

    Returns:
        bool: True if the number is a perfect number, False otherwise.
    """
    if not isinstance(number, int) or number <= 0:
        return False # Perfect numbers are positive integers

    divisors_sum = 0
    for i in range(1, number):
        if number % i == 0:
            divisors_sum += i

    return divisors_sum == number

# Example usage:
print(f"Is 6 a perfect number? {is_perfect_number(6)}")
print(f"Is 28 a perfect number? {is_perfect_number(28)}")
print(f"Is 12 a perfect number? {is_perfect_number(12)}")
print(f"Is 1 a perfect number? {is_perfect_number(1)}")
print(f"Is 496 a perfect number? {is_perfect_number(496)}")
```

```
Is 6 a perfect number? True
Is 28 a perfect number? True
Is 12 a perfect number? False
Is 1 a perfect number? False
Is 496 a perfect number? True
```

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 zeroshot 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 :**

```
def is_even_or_odd(number):
    """
    Determines whether a given number is even or odd.

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

    Returns:
        str: 'Even' if the number is even, 'Odd' if the number is odd.
             Returns an error message if the input is not a valid integer.
    """
    if not isinstance(number, int):
        return "Invalid input: Please provide an integer."

    if number % 2 == 0:
        return "Even"
    else:
        return "Odd"

# Example usage:
print(f"8 is: {is_even_or_odd(8)}")
print(f"15 is: {is_even_or_odd(15)}")
print(f"0 is: {is_even_or_odd(0)}")
print(f"-4 is: {is_even_or_odd(-4)}")
print(f"3.5 is: {is_even_or_odd(3.5)}")
print(f"'hello' is: {is_even_or_odd('hello')}")
```

```
8 is: Even
15 is: Odd
0 is: Even
-4 is: Even
3.5 is: Invalid input: Please provide an integer.
'hello' is: Invalid input: Please provide an integer.
```

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.