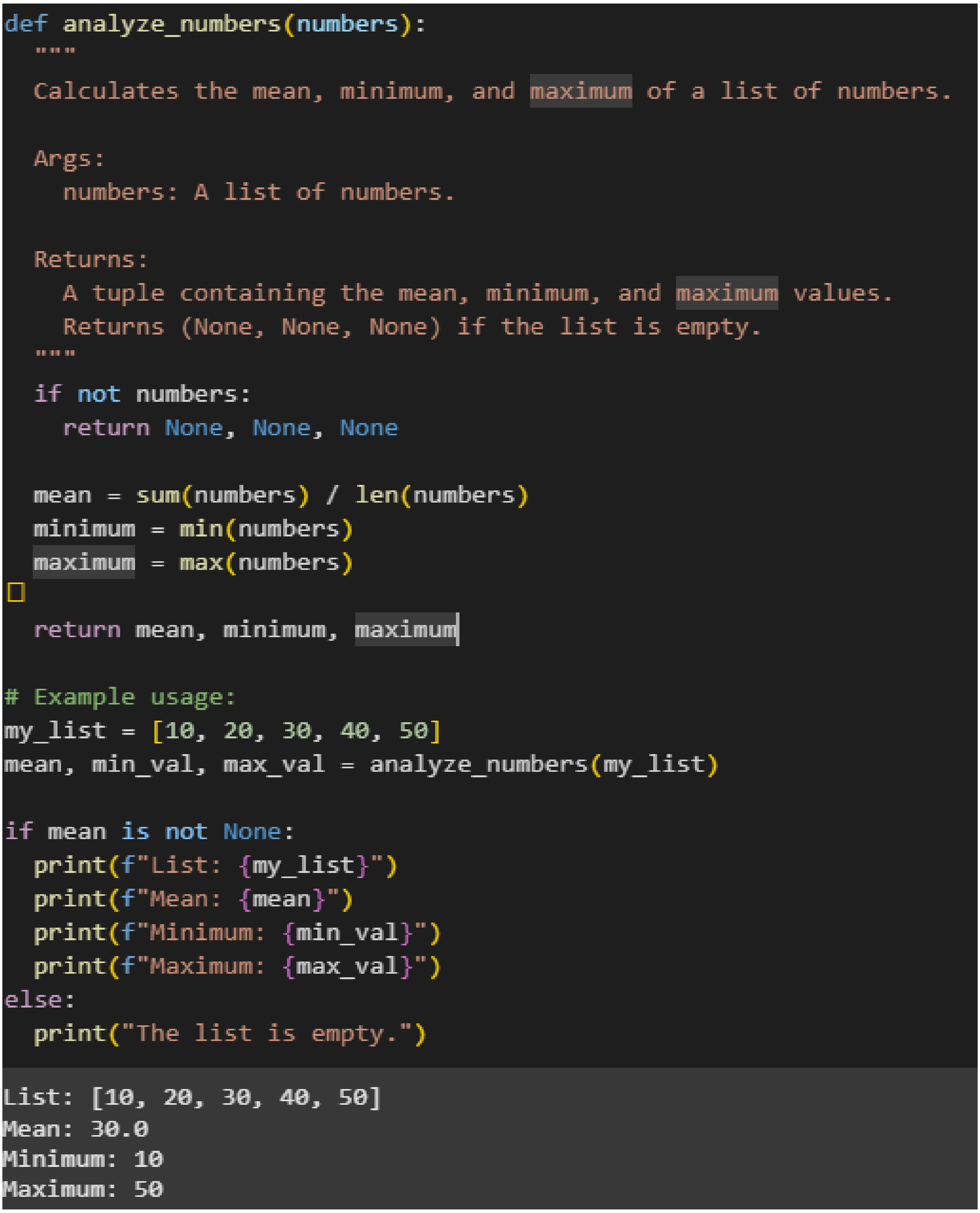
ASSIGNMENT 2

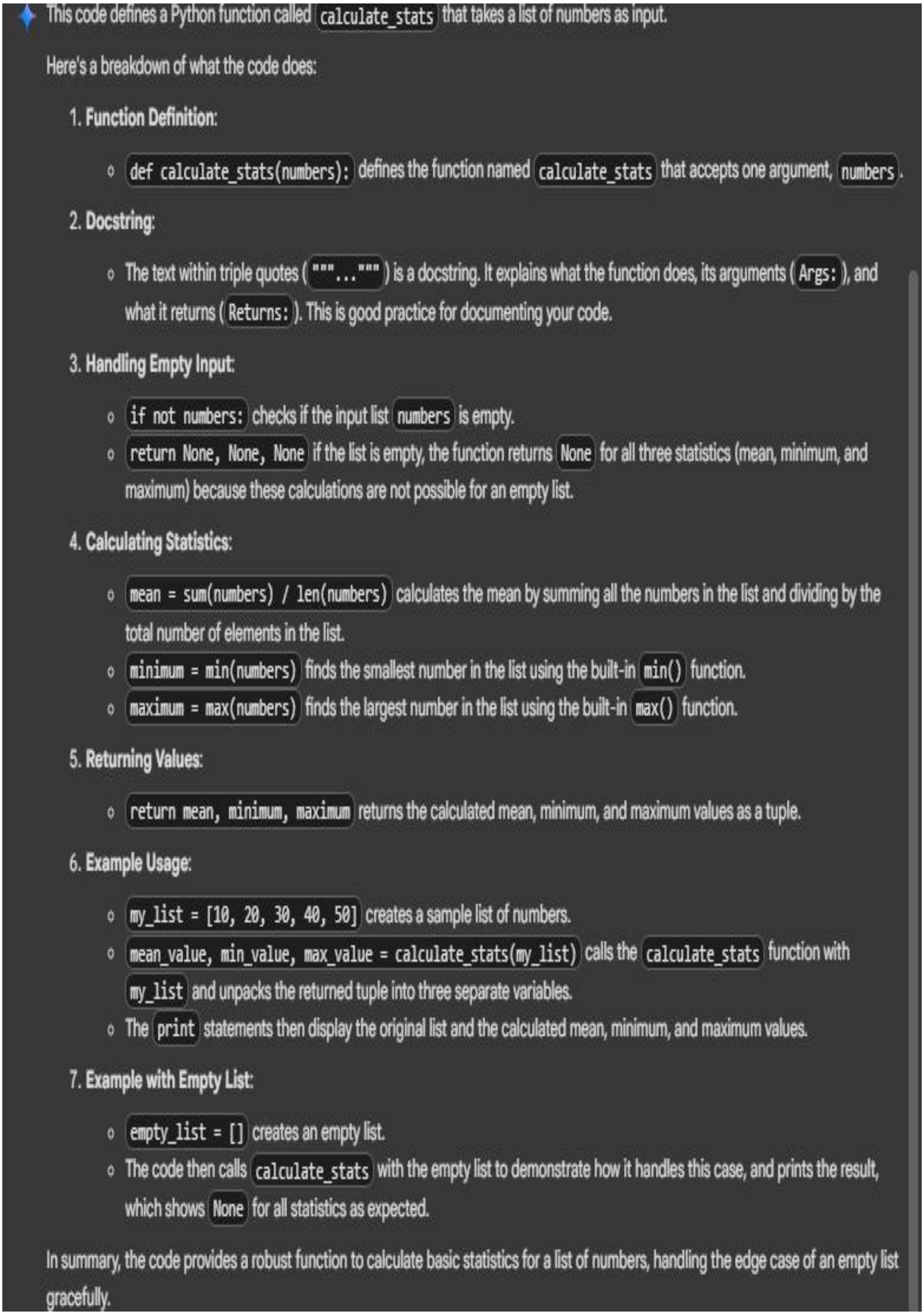
TASK 1 :



```python
def analyze_numbers(numbers):
  """
  Calculates the mean, minimum, and maximum of a list of numbers.

  Args:
    numbers: A list of numbers.

  Returns:
    A tuple containing the mean, minimum, and maximum values.
    Returns (None, None, None) if the list is empty.
  """
  if not numbers:
    return None, None, None

  mean = sum(numbers) / len(numbers)
  minimum = min(numbers)
  maximum = max(numbers)

  return mean, minimum, maximum

# Example usage:
my_list = [10, 20, 30, 40, 50]
mean, min_val, max_val = analyze_numbers(my_list)

if mean is not None:
  print(f"List: {my_list}")
  print(f"Mean: {mean}")
  print(f"Minimum: {min_val}")
  print(f"Maximum: {max_val}")
else:
  print("The list is empty.")
```

```
List: [10, 20, 30, 40, 50]
Mean: 30.0
Minimum: 10
Maximum: 50
```

EXPALNATION :



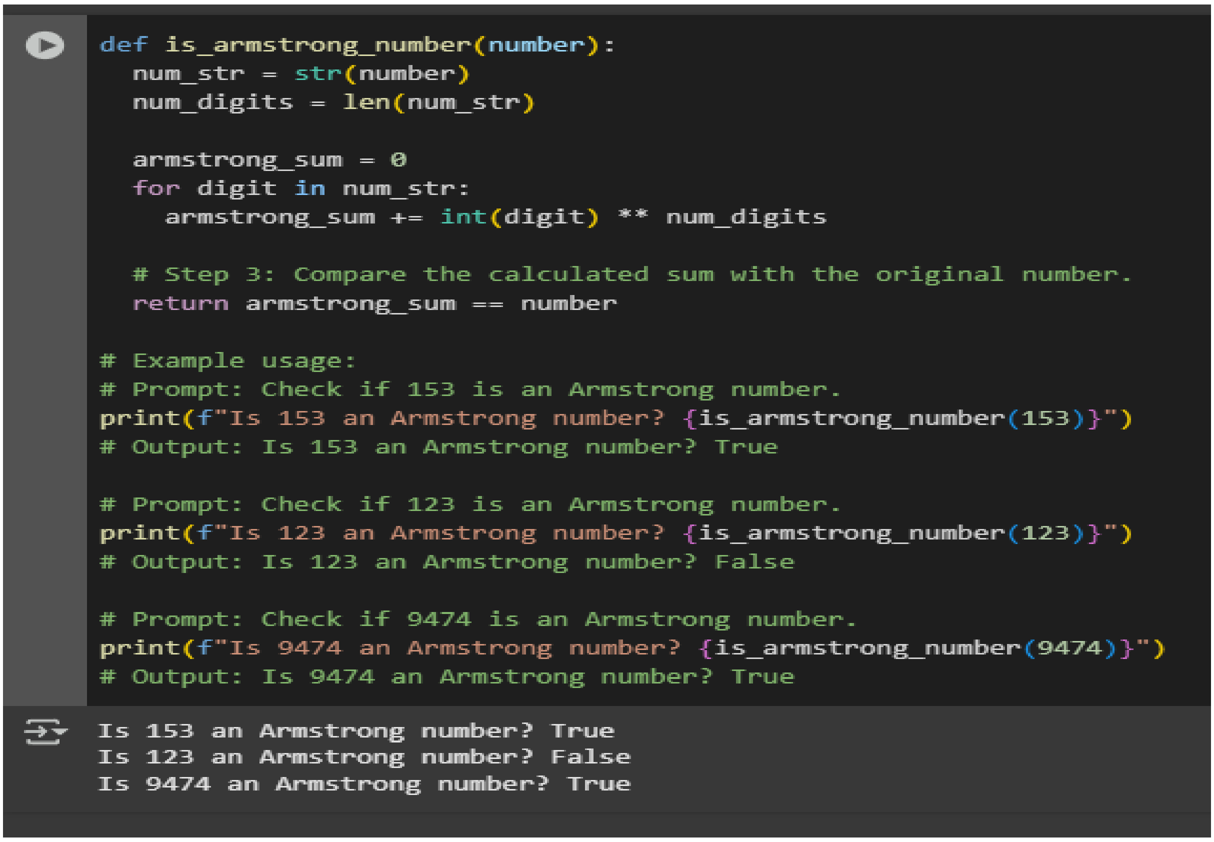
This code defines a Python function called `calculate_stats` that takes a list of numbers as input.

Here's a breakdown of what the code does:

1. **Function Definition:**
   - `def calculate_stats(numbers):` defines the function named `calculate_stats` that accepts one argument, `numbers`.
2. **Docstring:**
   - The text within triple quotes (`"""..."""`) is a docstring. It explains what the function does, its arguments (`Args:`), and what it returns (`Returns:`). This is good practice for documenting your code.
3. **Handling Empty Input:**
   - `if not numbers:` checks if the input list `numbers` is empty.
   - `return None, None, None` if the list is empty, the function returns `None` for all three statistics (mean, minimum, and maximum) because these calculations are not possible for an empty list.
4. **Calculating Statistics:**
   - `mean = sum(numbers) / len(numbers)` calculates the mean by summing all the numbers in the list and dividing by the total number of elements in the list.
   - `minimum = min(numbers)` finds the smallest number in the list using the built-in `min()` function.
   - `maximum = max(numbers)` finds the largest number in the list using the built-in `max()` function.
5. **Returning Values:**
   - `return mean, minimum, maximum` returns the calculated mean, minimum, and maximum values as a tuple.
6. **Example Usage:**
   - `my_list = [10, 20, 30, 40, 50]` creates a sample list of numbers.
   - `mean_value, min_value, max_value = calculate_stats(my_list)` calls the `calculate_stats` function with `my_list` and unpacks the returned tuple into three separate variables.
   - The `print` statements then display the original list and the calculated mean, minimum, and maximum values.
7. **Example with Empty List:**
   - `empty_list = []` creates an empty list.
   - The code then calls `calculate_stats` with the empty list to demonstrate how it handles this case, and prints the result, which shows `None` for all statistics as expected.

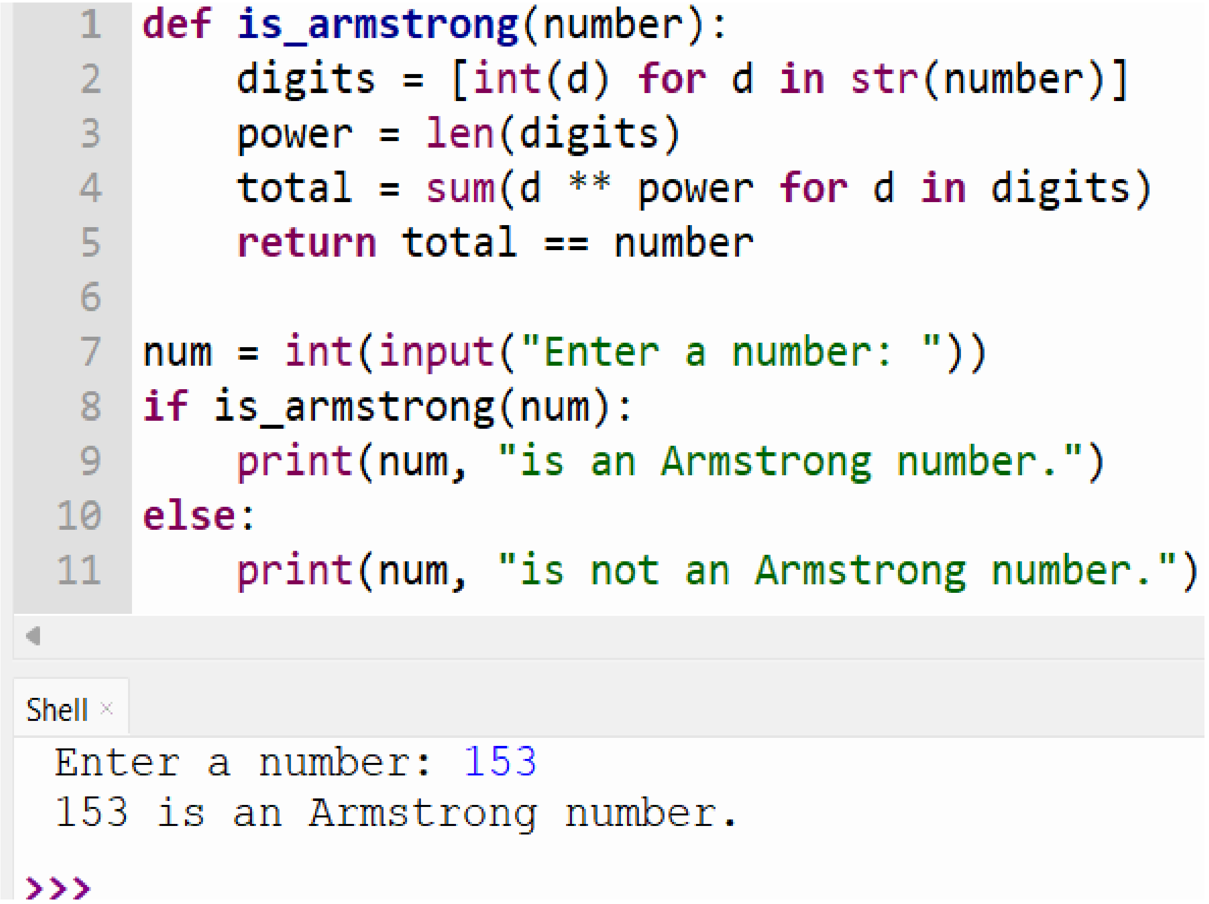
In summary, the code provides a robust function to calculate basic statistics for a list of numbers, handling the edge case of an empty list gracefully.

TASK 2 :



```python
def is_armstrong_number(number):
  num_str = str(number)
  num_digits = len(num_str)

  armstrong_sum = 0
  for digit in num_str:
    armstrong_sum += int(digit) ** num_digits

  # Step 3: Compare the calculated sum with the original number.
  return armstrong_sum == number

# Example usage:
# Prompt: Check if 153 is an Armstrong number.
print(f"Is 153 an Armstrong number? {is_armstrong_number(153)}")
# Output: Is 153 an Armstrong number? True

# Prompt: Check if 123 is an Armstrong number.
print(f"Is 123 an Armstrong number? {is_armstrong_number(123)}")
# Output: Is 123 an Armstrong number? False

# Prompt: Check if 9474 is an Armstrong number.
print(f"Is 9474 an Armstrong number? {is_armstrong_number(9474)}")
# Output: Is 9474 an Armstrong number? True
```

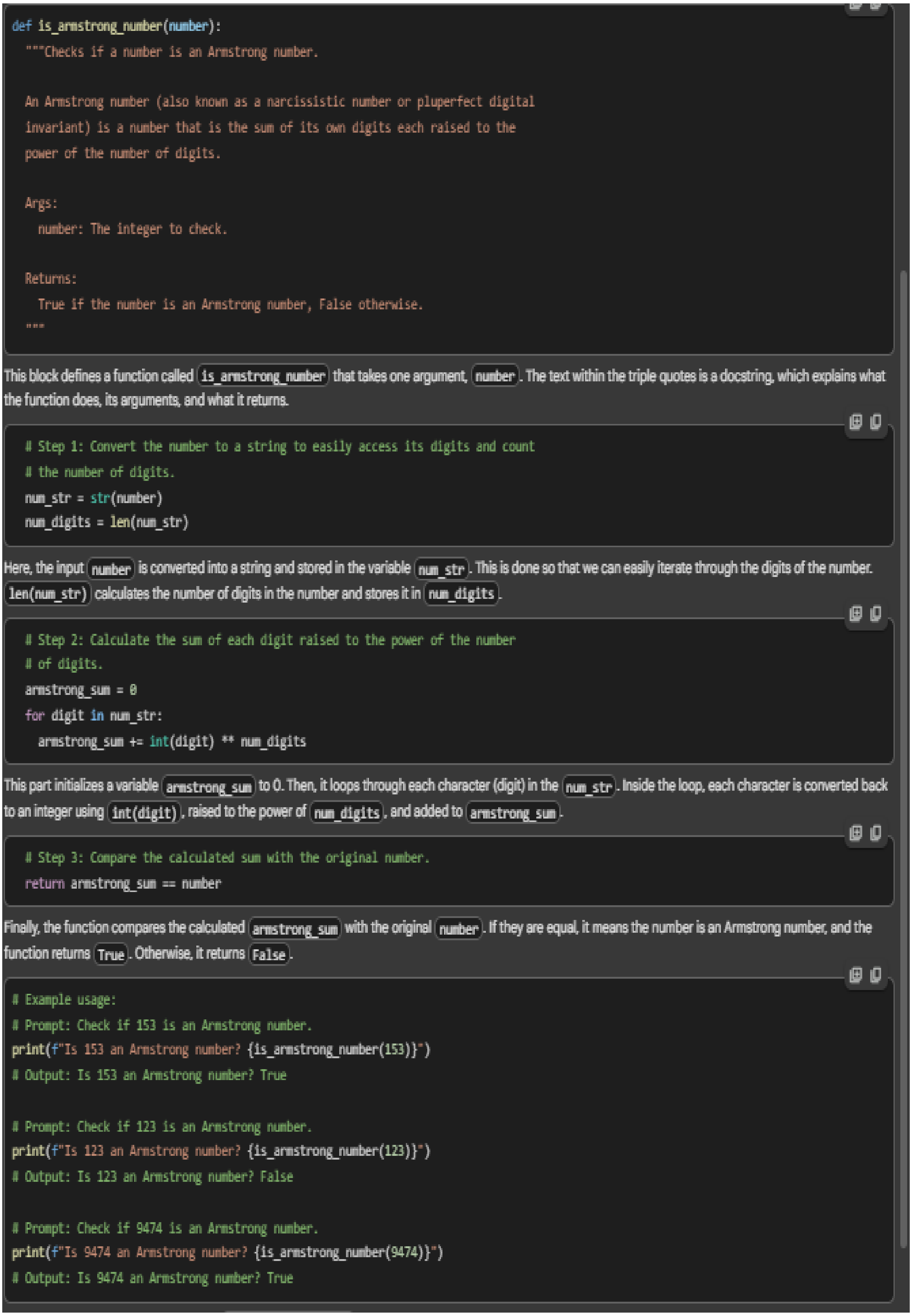
```
Is 153 an Armstrong number? True
Is 123 an Armstrong number? False
Is 9474 an Armstrong number? True
```



```python
def is_armstrong(number):
    digits = [int(d) for d in str(number)]
    power = len(digits)
    total = sum(d ** power for d in digits)
    return total == number

num = int(input("Enter a number: "))
if is_armstrong(num):
    print(num, "is an Armstrong number.")
else:
    print(num, "is not an Armstrong number.")
```

```
Enter a number: 153
153 is an Armstrong number.

>>>
```

EXPALNATION :



```
def is_armstrong_number(number):
  """Checks if a number is an Armstrong number.

  An Armstrong number (also known as a narcissistic number or pluperfect digital
  invariant) is a number that is the sum of its own digits each raised to the
  power of the number of digits.

  Args:
    number: The integer to check.

  Returns:
    True if the number is an Armstrong number, False otherwise.
  """
```

This block defines a function called `is_armstrong_number` that takes one argument, `number`. The text within the triple quotes is a docstring, which explains what the function does, its arguments, and what it returns.

```
  # Step 1: Convert the number to a string to easily access its digits and count
  # the number of digits.
  num_str = str(number)
  num_digits = len(num_str)
```

Here, the input `number` is converted into a string and stored in the variable `num_str`. This is done so that we can easily iterate through the digits of the number. `len(num_str)` calculates the number of digits in the number and stores it in `num_digits`.

```
  # Step 2: Calculate the sum of each digit raised to the power of the number
  # of digits.
  armstrong_sum = 0
  for digit in num_str:
    armstrong_sum += int(digit) ** num_digits
```

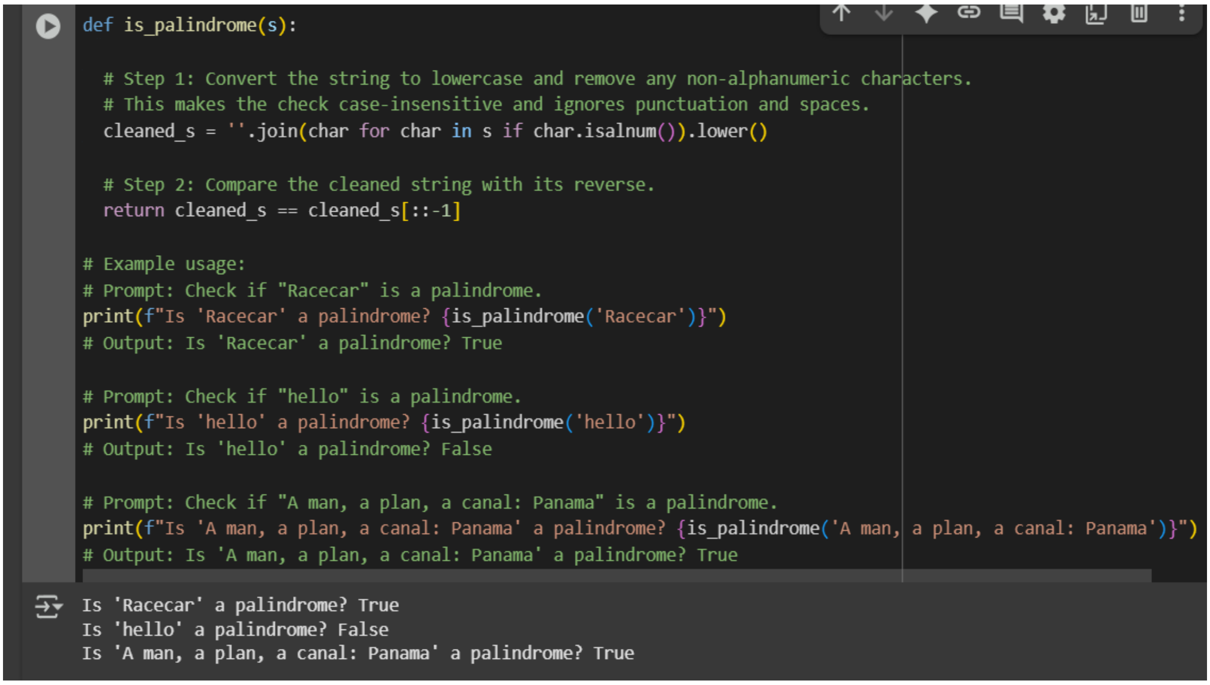
This part initializes a variable `armstrong_sum` to 0. Then, it loops through each character (digit) in the `num_str`. Inside the loop, each character is converted back to an integer using `int(digit)`, raised to the power of `num_digits`, and added to `armstrong_sum`.

```
  # Step 3: Compare the calculated sum with the original number.
  return armstrong_sum == number
```

Finally, the function compares the calculated `armstrong_sum` with the original `number`. If they are equal, it means the number is an Armstrong number, and the function returns `True`. Otherwise, it returns `False`.

```
# Example usage:
# Prompt: Check if 153 is an Armstrong number.
print(f"Is 153 an Armstrong number? {is_armstrong_number(153)}")
# Output: Is 153 an Armstrong number? True

# Prompt: Check if 123 is an Armstrong number.
print(f"Is 123 an Armstrong number? {is_armstrong_number(123)}")
# Output: Is 123 an Armstrong number? False

# Prompt: Check if 9474 is an Armstrong number.
print(f"Is 9474 an Armstrong number? {is_armstrong_number(9474)}")
# Output: Is 9474 an Armstrong number? True
```

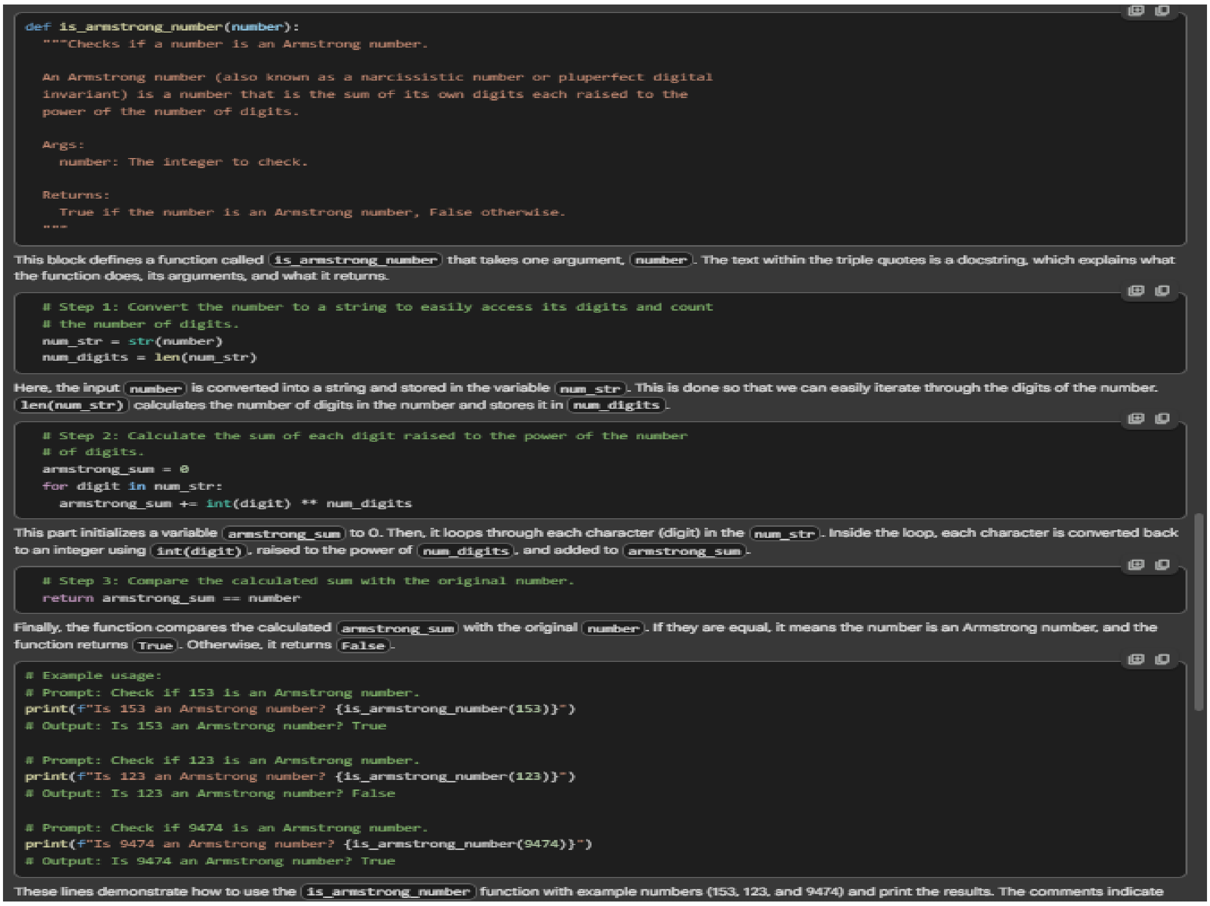
TASK 3 :



```
def is_palindrome(s):

  # Step 1: Convert the string to lowercase and remove any non-alphanumeric characters.
  # This makes the check case-insensitive and ignores punctuation and spaces.
  cleaned_s = ''.join(char for char in s if char.isalnum()).lower()

  # Step 2: Compare the cleaned string with its reverse.
  return cleaned_s == cleaned_s[::-1]

# Example usage:
# Prompt: Check if "Racecar" is a palindrome.
print(f"Is 'Racecar' a palindrome? {is_palindrome('Racecar')}")
# Output: Is 'Racecar' a palindrome? True

# Prompt: Check if "hello" is a palindrome.
print(f"Is 'hello' a palindrome? {is_palindrome('hello')}")
# Output: Is 'hello' a palindrome? False

# Prompt: Check if "A man, a plan, a canal: Panama" is a palindrome.
print(f"Is 'A man, a plan, a canal: Panama' a palindrome? {is_palindrome('A man, a plan, a canal: Panama')}")
# Output: Is 'A man, a plan, a canal: Panama' a palindrome? True
```

```
Is 'Racecar' a palindrome? True
Is 'hello' a palindrome? False
Is 'A man, a plan, a canal: Panama' a palindrome? True
```

EXPALNATION :



```
def is_armstrong_number(number):
  """Checks if a number is an Armstrong number.

  An Armstrong number (also known as a narcissistic number or pluperfect digital
  invariant) is a number that is the sum of its own digits each raised to the
  power of the number of digits.

  Args:
    number: The integer to check.

  Returns:
    True if the number is an Armstrong number, False otherwise.
  """
```

This block defines a function called `is_armstrong_number` that takes one argument, `number`. The text within the triple quotes is a docstring, which explains what the function does, its arguments, and what it returns.

```
  # Step 1: Convert the number to a string to easily access its digits and count
  # the number of digits.
  num_str = str(number)
  num_digits = len(num_str)
```

Here, the input `number` is converted into a string and stored in the variable `num_str`. This is done so that we can easily iterate through the digits of the number. `len(num_str)` calculates the number of digits in the number and stores it in `num_digits`.

```
  # Step 2: Calculate the sum of each digit raised to the power of the number
  # of digits.
  armstrong_sum = 0
  for digit in num_str:
    armstrong_sum += int(digit) ** num_digits
```

This part initializes a variable `armstrong_sum` to 0. Then, it loops through each character (digit) in the `num_str`. Inside the loop, each character is converted back to an integer using `int(digit)`, raised to the power of `num_digits`, and added to `armstrong_sum`.

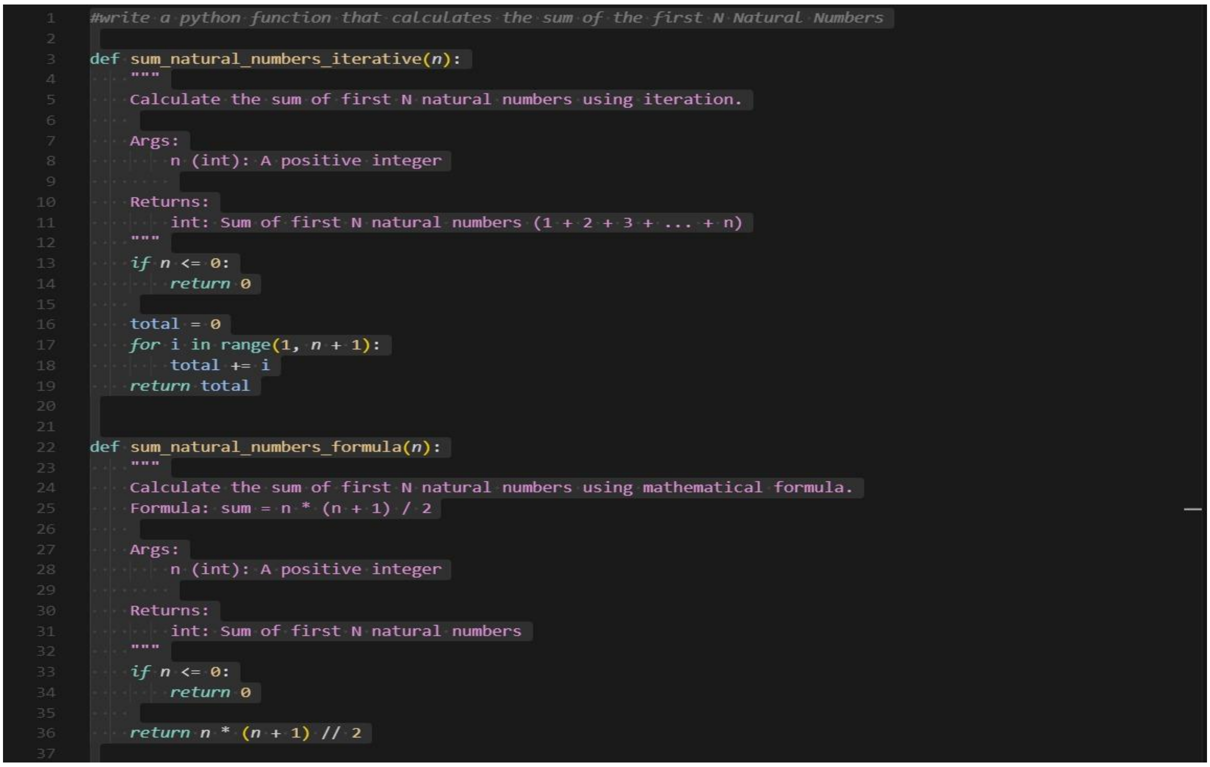
```
  # Step 3: Compare the calculated sum with the original number.
  return armstrong_sum == number
```

Finally, the function compares the calculated `armstrong_sum` with the original `number`. If they are equal, it means the number is an Armstrong number, and the function returns `True`. Otherwise, it returns `False`.

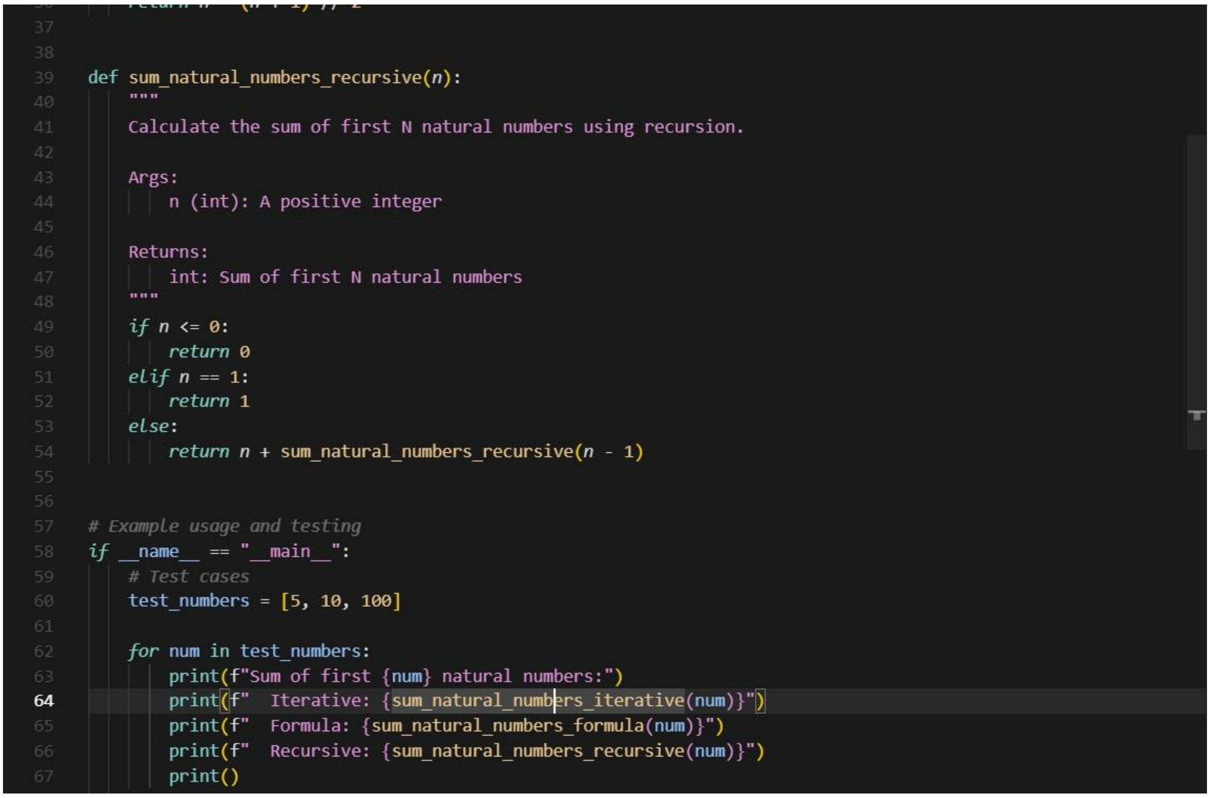
```
# Example usage:
# Prompt: Check if 153 is an Armstrong number.
print(f"Is 153 an Armstrong number? {is_armstrong_number(153)}")
# Output: Is 153 an Armstrong number? True

# Prompt: Check if 123 is an Armstrong number.
print(f"Is 123 an Armstrong number? {is_armstrong_number(123)}")
# Output: Is 123 an Armstrong number? False

# Prompt: Check if 9474 is an Armstrong number.
print(f"Is 9474 an Armstrong number? {is_armstrong_number(9474)}")
# Output: Is 9474 an Armstrong number? True
```

These lines demonstrate how to use the `is_armstrong_number` function with example numbers (153, 123, and 9474) and print the results. The comments indicate

TASK 4 :

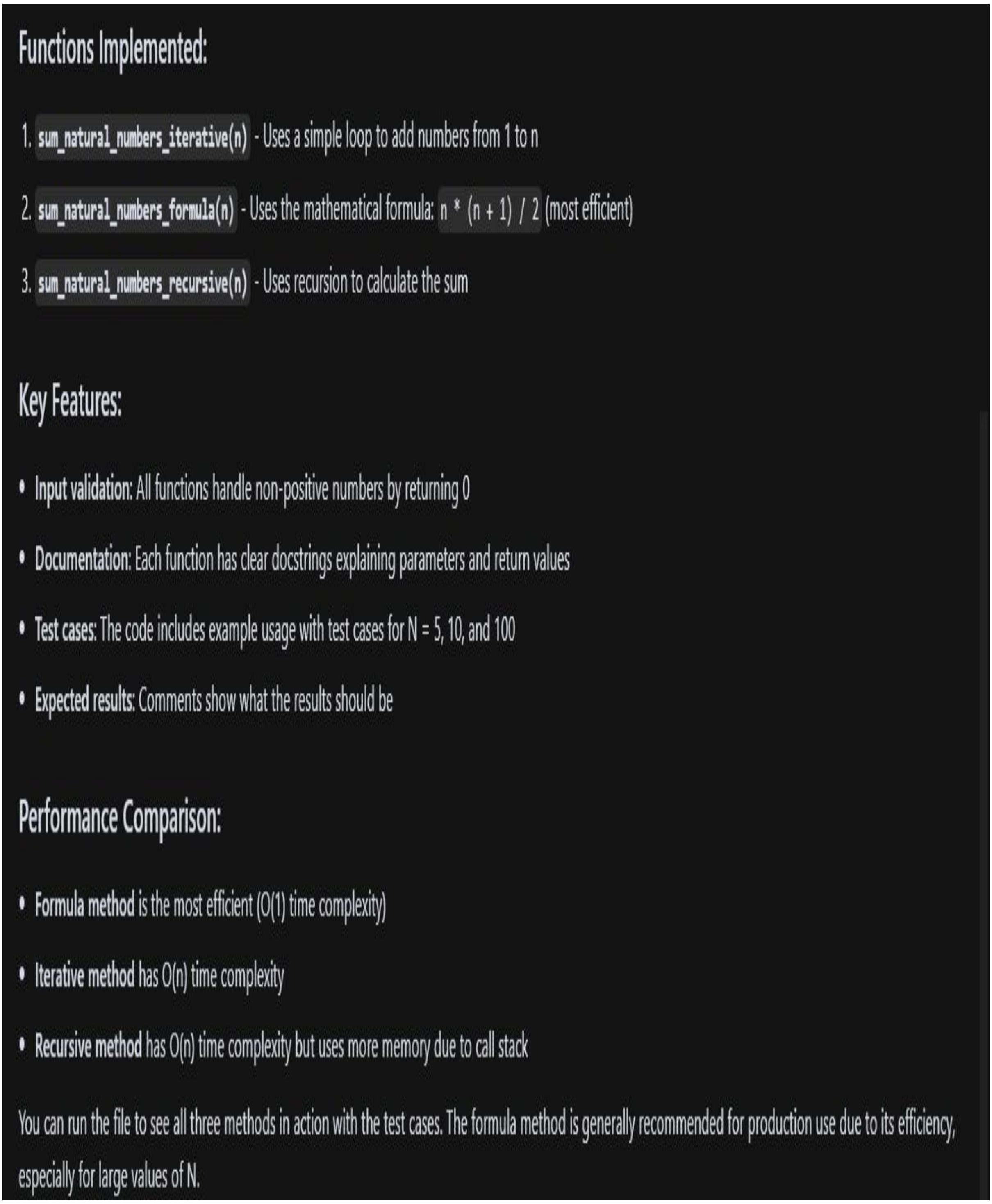


```python
#write a python function that calculates the sum of the first N Natural Numbers

def sum_natural_numbers_iterative(n):
    """
    Calculate the sum of first N natural numbers using iteration.
    
    Args:
        n (int): A positive integer
        
    Returns:
        int: Sum of first N natural numbers (1 + 2 + 3 + ... + n)
    """
    if n <= 0:
        return 0
    
    total = 0
    for i in range(1, n + 1):
        total += i
    return total


def sum_natural_numbers_formula(n):
    """
    Calculate the sum of first N natural numbers using mathematical formula.
    Formula: sum = n * (n + 1) / 2
    
    Args:
        n (int): A positive integer
        
    Returns:
        int: Sum of first N natural numbers
    """
    if n <= 0:
        return 0
    
    return n * (n + 1) // 2

```



```python


def sum_natural_numbers_recursive(n):
    """
    Calculate the sum of first N natural numbers using recursion.

    Args:
        n (int): A positive integer

    Returns:
        int: Sum of first N natural numbers
    """
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    else:
        return n + sum_natural_numbers_recursive(n - 1)


# Example usage and testing
if __name__ == "__main__":
    # Test cases
    test_numbers = [5, 10, 100]

    for num in test_numbers:
        print(f"Sum of first {num} natural numbers:")
        print(f"  Iterative: {sum_natural_numbers_iterative(num)}")
        print(f"  Formula: {sum_natural_numbers_formula(num)}")
        print(f"  Recursive: {sum_natural_numbers_recursive(num)}")
        print()
```

EXPALNATION :



Functions Implemented:

1. `sum_natural_numbers_iterative(n)` - Uses a simple loop to add numbers from 1 to n
2. `sum_natural_numbers_formula(n)` - Uses the mathematical formula: `n * (n + 1) / 2` (most efficient)
3. `sum_natural_numbers_recursive(n)` - Uses recursion to calculate the sum

Key Features:

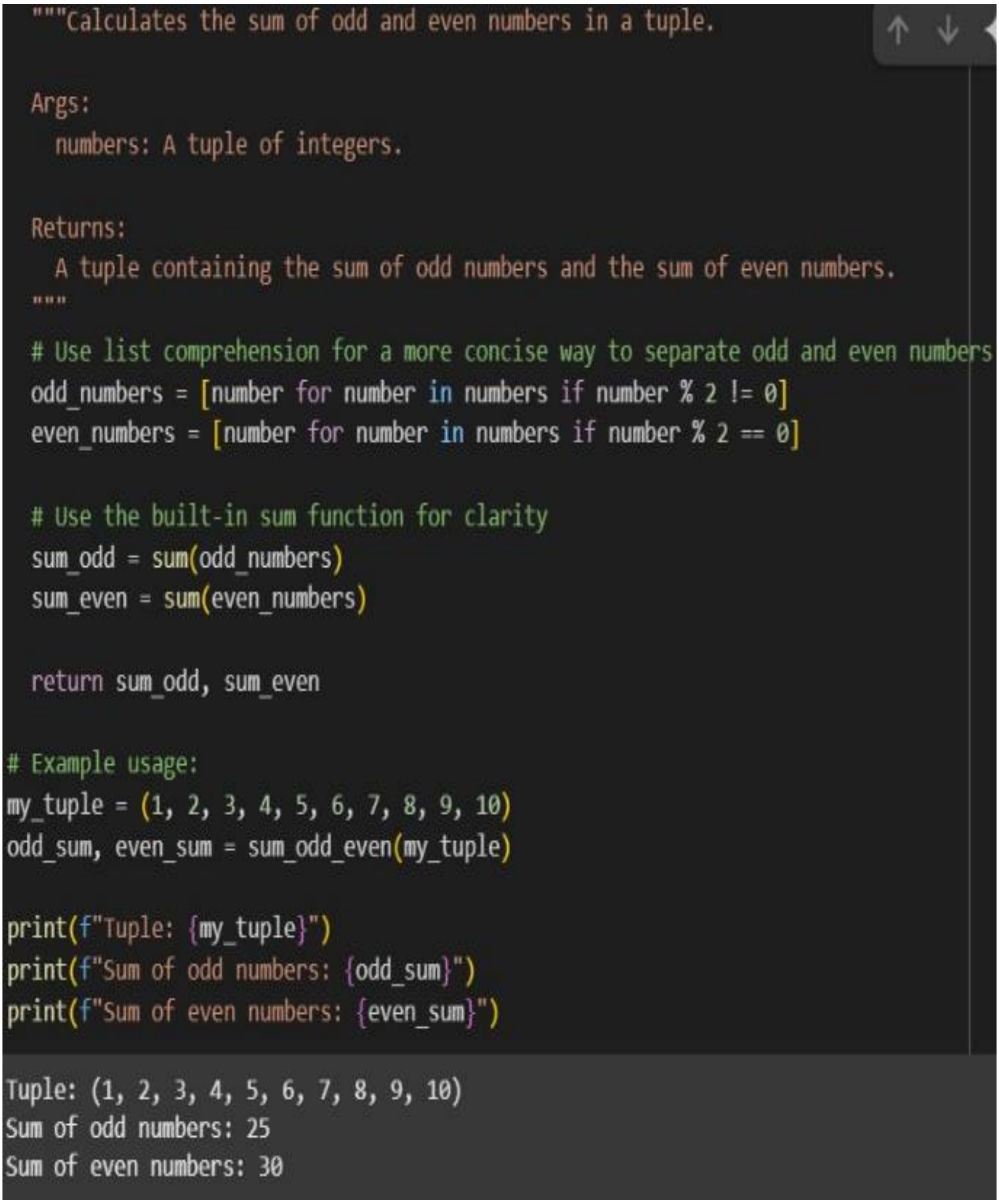
- **Input validation**: All functions handle non-positive numbers by returning 0
- **Documentation**: Each function has clear docstrings explaining parameters and return values
- **Test cases**: The code includes example usage with test cases for N = 5, 10, and 100
- **Expected results**: Comments show what the results should be

Performance Comparison:

- **Formula method** is the most efficient (O(1) time complexity)
- **Iterative method** has O(n) time complexity
- **Recursive method** has O(n) time complexity but uses more memory due to call stack

You can run the file to see all three methods in action with the test cases. The formula method is generally recommended for production use due to its efficiency, especially for large values of N.

TASK 5:



```
  """Calculates the sum of odd and even numbers in a tuple.

  Args:
    numbers: A tuple of integers.

  Returns:
    A tuple containing the sum of odd numbers and the sum of even numbers.
  """
  # Use list comprehension for a more concise way to separate odd and even numbers
  odd_numbers = [number for number in numbers if number % 2 != 0]
  even_numbers = [number for number in numbers if number % 2 == 0]

  # Use the built-in sum function for clarity
  sum_odd = sum(odd_numbers)
  sum_even = sum(even_numbers)

  return sum_odd, sum_even

# Example usage:
my_tuple = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
odd_sum, even_sum = sum_odd_even(my_tuple)

print(f"Tuple: {my_tuple}")
print(f"Sum of odd numbers: {odd_sum}")
print(f"Sum of even numbers: {even_sum}")
```

```
Tuple: (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
Sum of odd numbers: 25
Sum of even numbers: 30
```

EXPALNATION :

1. `def calculate_sum_even_odd(numbers):`
    - This line **defines a function** named `calculate_sum_even_odd`. This function takes one argument, `numbers`, which is expected to be a tuple.
2. `even_sum = 0`
    - This line **initializes a variable** named `even_sum` and sets its value to `0`. This variable will be used to store the sum of all even numbers found in the tuple.
3. `odd_sum = 0`
    - Similarly, this line **initializes a variable** named `odd_sum` to `0`. It will store the sum of all odd numbers.
4. `for number in numbers:`
    - This is the start of a `for` **loop**. It iterates through each element in the `numbers` tuple. In each iteration, the current element is assigned to the variable `number`.
5. `if number % 2 == 0:`
    - This is an `if` **statement** that checks a condition. The modulo operator (`%`) returns the remainder of a division. The condition `number % 2 == 0` is true if the number is **evenly divisible by 2** (i.e., it's an even number).
6. `even_sum += number`
    - If the condition in the `if` statement is true, this line **adds the current** `number` **to** `even_sum`. It's shorthand for `even_sum = even_sum + number`.
7. `else:`
    - This is the `else` **block** of the `if` statement. The code here is executed only if the condition `number % 2 == 0` is false.
8. `odd_sum += number`
    - This line is executed if the number is not even (i.e., it's an odd number). It **adds the current** `number` **to** `odd_sum`.
9. `return even_sum, odd_sum`
    - After the loop finishes, this line **returns the final values** of `even_sum` and `odd_sum` as a tuple.
10. `my_tuple = (1, 2, 3, 4, 5, 6, 7, 8, 9)`
    - This line **creates the tuple** of numbers that will be used as input for the function.
11. `even_sum, odd_sum = calculate_sum_even_odd(my_tuple)`
    - This line **calls the function** `calculate_sum_even_odd` **with** `my_tuple` as its argument. The returned tuple (e.g., `(20, 25)`) is then **unpacked** into the two variables, so `even_sum` becomes `20` and `odd_sum` becomes `25`.
12. `print(f"The given tuple is: {my_tuple}")`
    - This line **prints the original tuple** to the console using an f-string for clear formatting.
13. `print(f"Sum of even numbers: {even_sum}")`
    - This line **prints the final sum of the even numbers.**
14. `print(f"Sum of odd numbers: {odd_sum}")`
    - This line **prints the final sum of the odd numbers.**