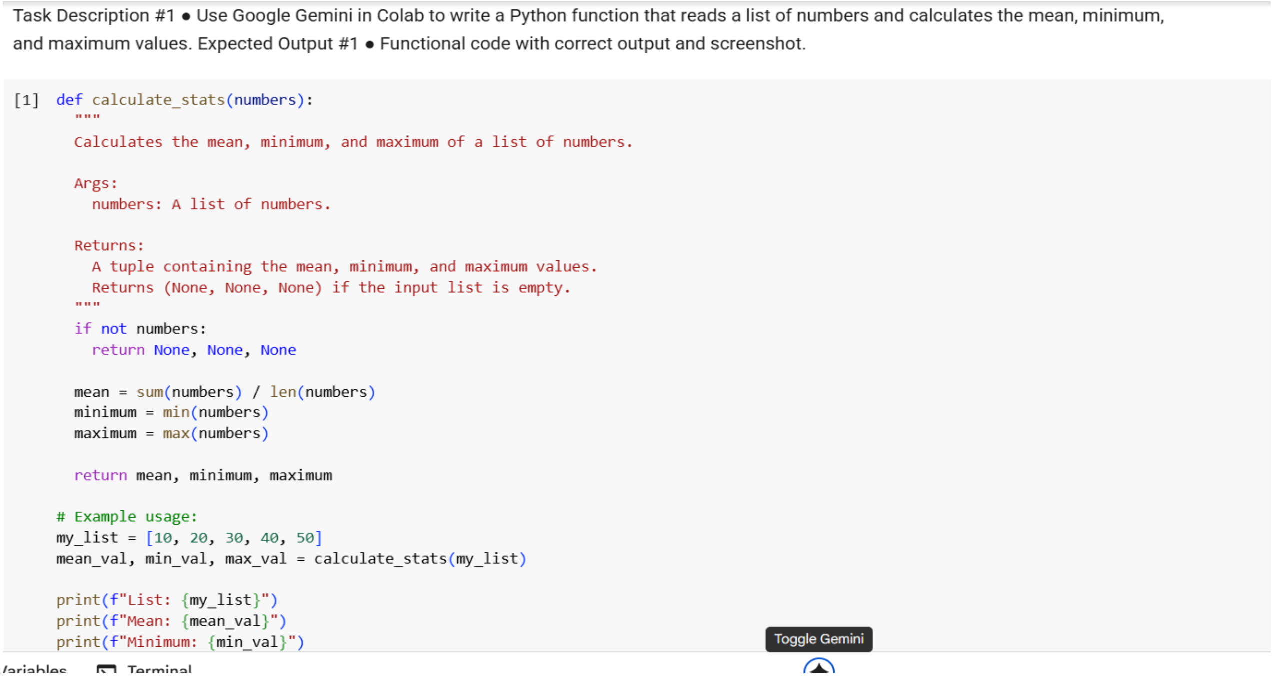
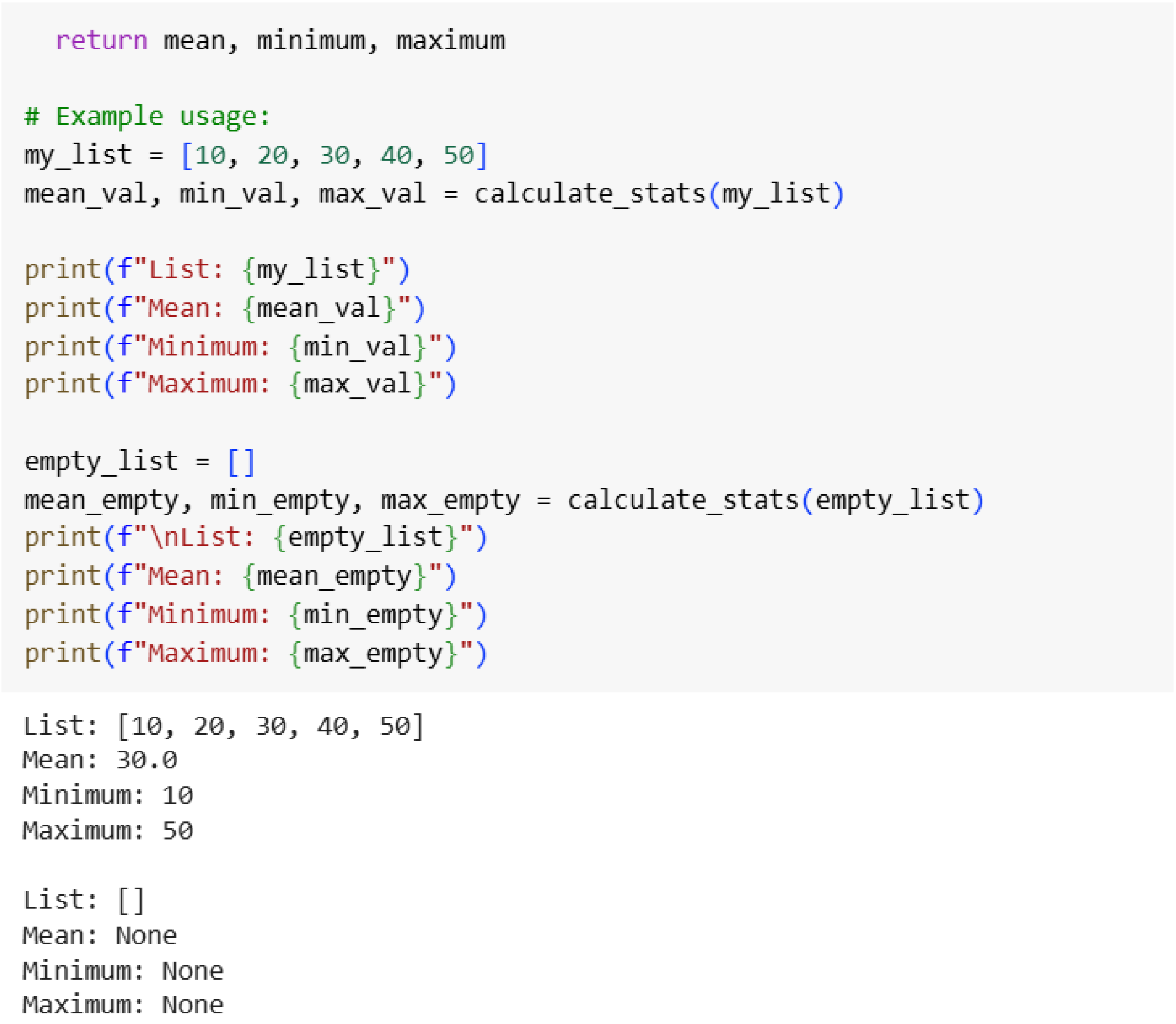


Task Description #1 ● Use Google Gemini in Colab to write a Python function that reads a list of numbers and calculates the mean, minimum, and maximum values. Expected Output #1 ● Functional code with correct output and screenshot.

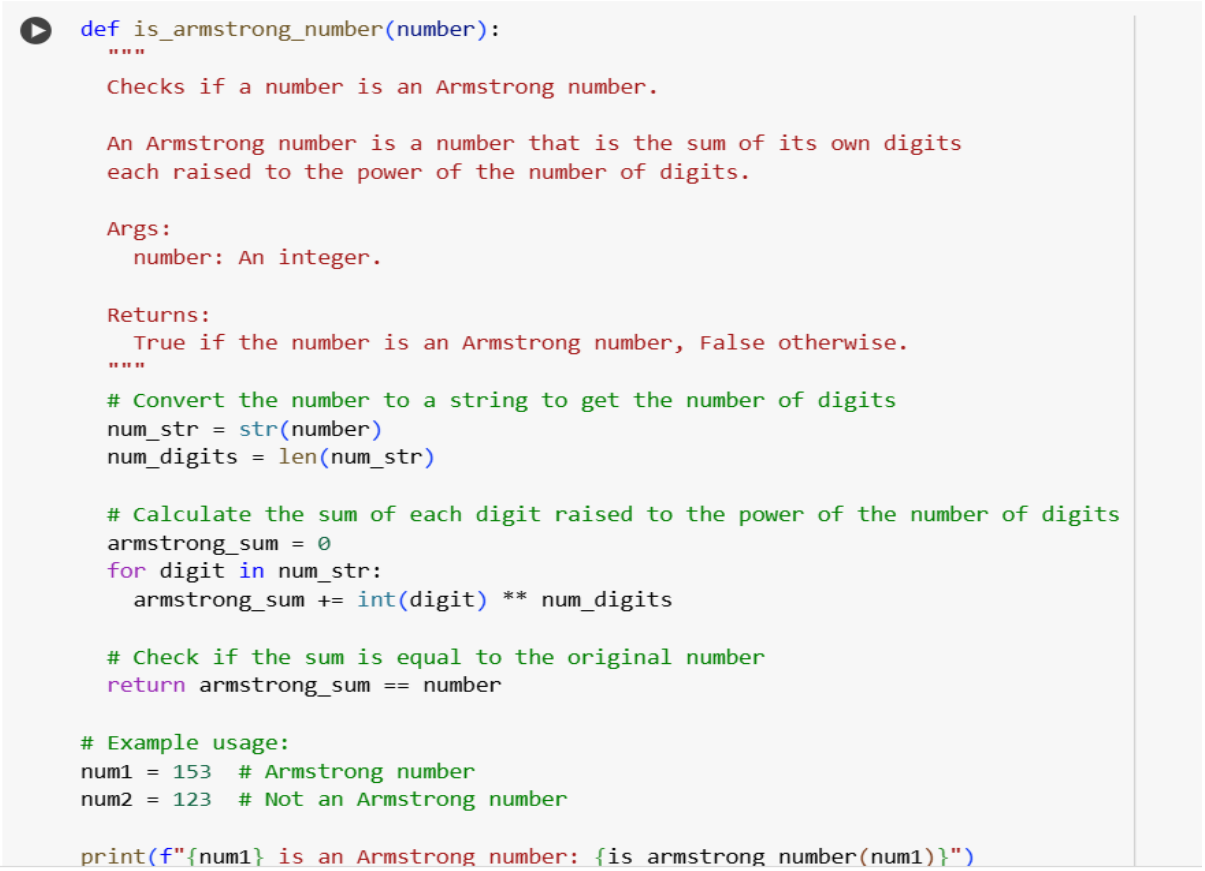
```
[1] def calculate_stats(numbers):
      """
      Calculates the mean, minimum, and maximum of a list of numbers.

      Args:
        numbers: A list of numbers.

      Returns:
        A tuple containing the mean, minimum, and maximum values.
        Returns (None, None, None) if the input 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_val, min_val, max_val = calculate_stats(my_list)

    print(f"List: {my_list}")
    print(f"Mean: {mean_val}")
    print(f"Minimum: {min_val}")
```

m



```
  return mean, minimum, maximum

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

print(f"List: {my_list}")
print(f"Mean: {mean_val}")
print(f"Minimum: {min_val}")
print(f"Maximum: {max_val}")

empty_list = []
mean_empty, min_empty, max_empty = calculate_stats(empty_list)
print(f"\nList: {empty_list}")
print(f"Mean: {mean_empty}")
print(f"Minimum: {min_empty}")
print(f"Maximum: {max_empty}")
```

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

List: []
Mean: None
Minimum: None
Maximum: None
```



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

  An Armstrong number is a number that is the sum of its own digits
  each raised to the power of the number of digits.

  Args:
    number: An integer.

  Returns:
    True if the number is an Armstrong number, False otherwise.
  """
  # Convert the number to a string to get the number of digits
  num_str = str(number)
  num_digits = len(num_str)

  # 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

  # Check if the sum is equal to the original number
  return armstrong_sum == number

# Example usage:
num1 = 153  # Armstrong number
num2 = 123  # Not an Armstrong number

print(f"{num1} is an Armstrong number: {is_armstrong_number(num1)}")
```

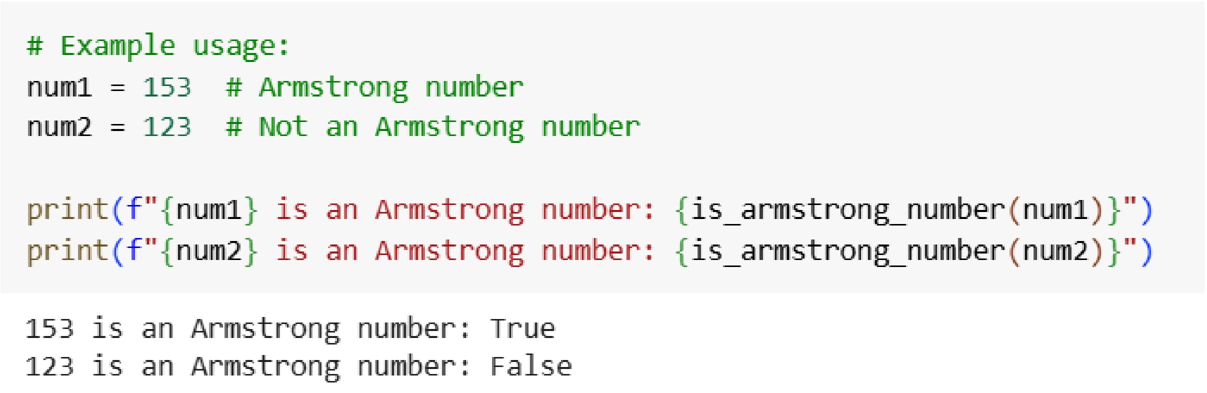
ask Description #2 ● Compare Gemini and Copilot outputs for a Python function that checks whether a number is an Armstrong number. Document the steps, prompts, and outputs. Expected Output #2 ● Side-by-side comparison table with observations and screenshots

Here's a Python function to check if a number is an Armstrong number, along with the steps and prompts used.

Prompt: "Write a Python function that checks if a number is an Armstrong number."

Steps:

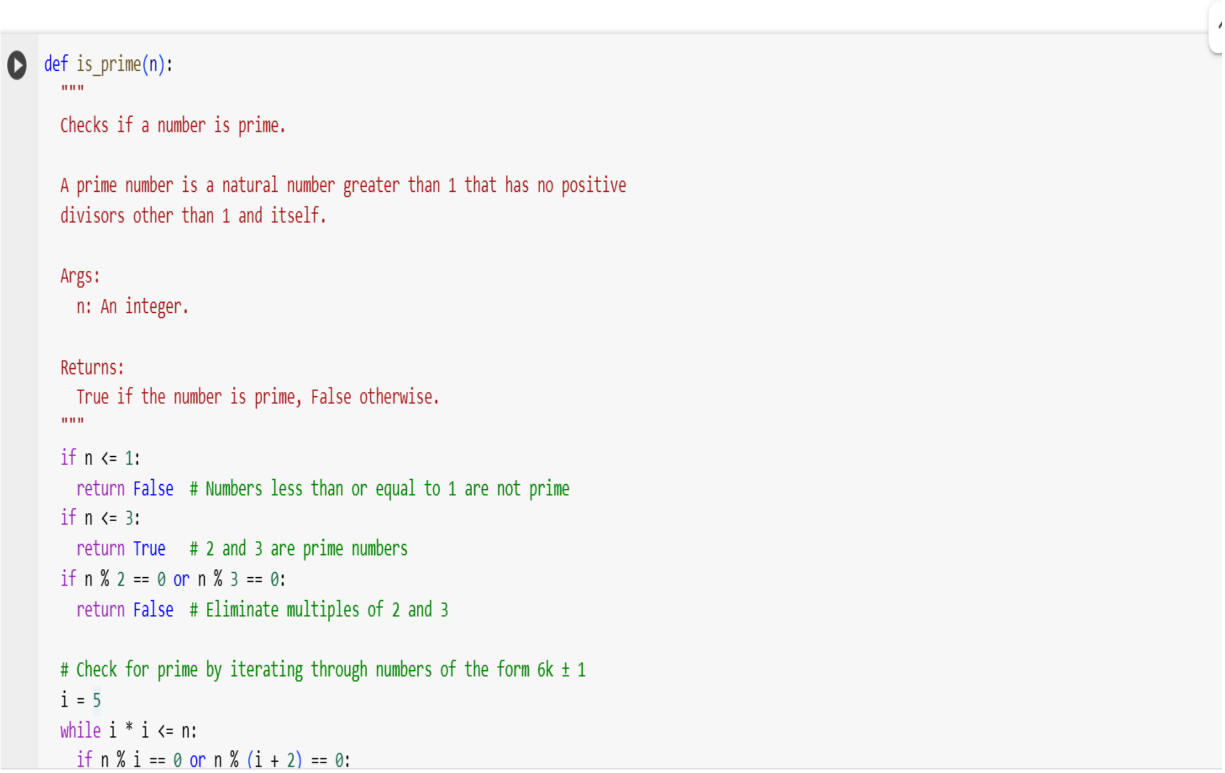
1. Define a function that takes an integer as input.
2. Convert the number to a string to find the number of digits.
3. Initialize a variable to store the sum of the digits raised to the power of the number of digits.
4. Iterate through each digit of the number.
5. For each digit, convert it back to an integer, raise it to the power of the number of digits, and add it to the sum.
6. Compare the calculated sum with the original number.
7. Return `True` if they are equal (it's an Armstrong number), otherwise return `False`.



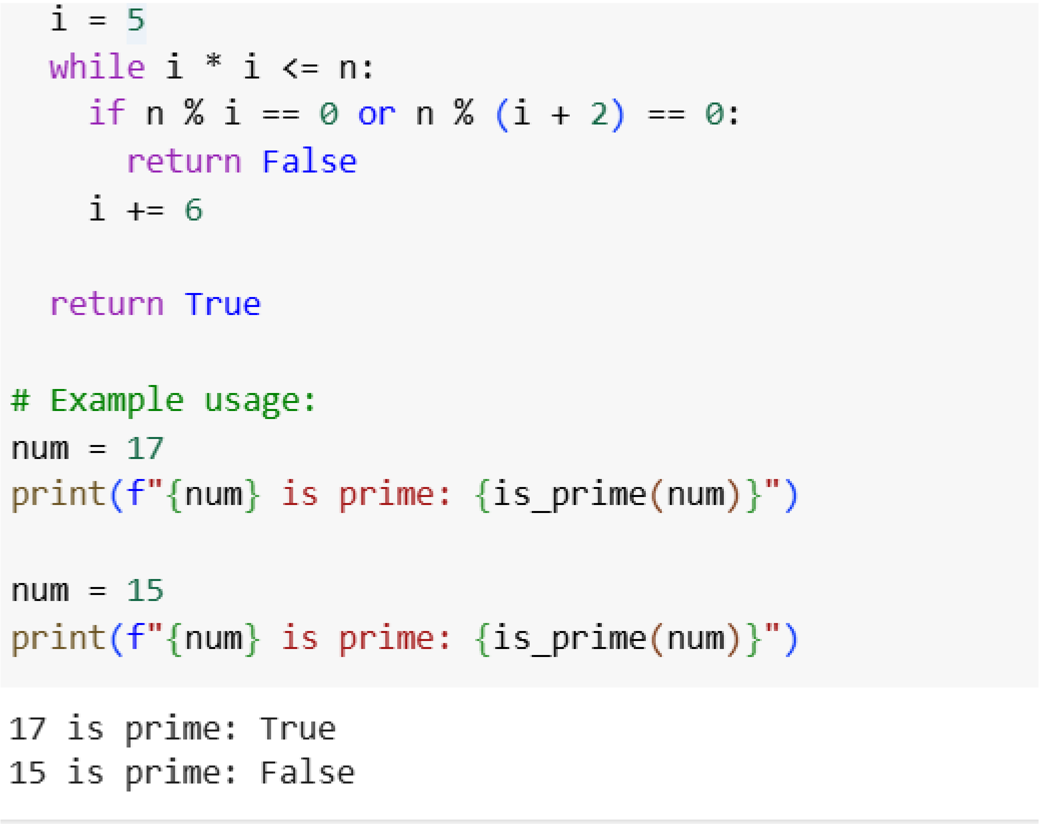
```
# Example usage:
num1 = 153  # Armstrong number
num2 = 123  # Not an Armstrong number

print(f"{num1} is an Armstrong number: {is_armstrong_number(num1)}")
print(f"{num2} is an Armstrong number: {is_armstrong_number(num2)}")
```

```
153 is an Armstrong number: True
123 is an Armstrong number: False
```

Task Description #3 • Ask Gemini to explain a Python function (e.g., is_prime(n) or is_palindrome(s)) line by line. • Choose either a prime-checking or palindrome-checking function and document the explanation provided by Gemini. Expected Output #3 • Detailed explanation with the code snippet and Gemini's responseexplain



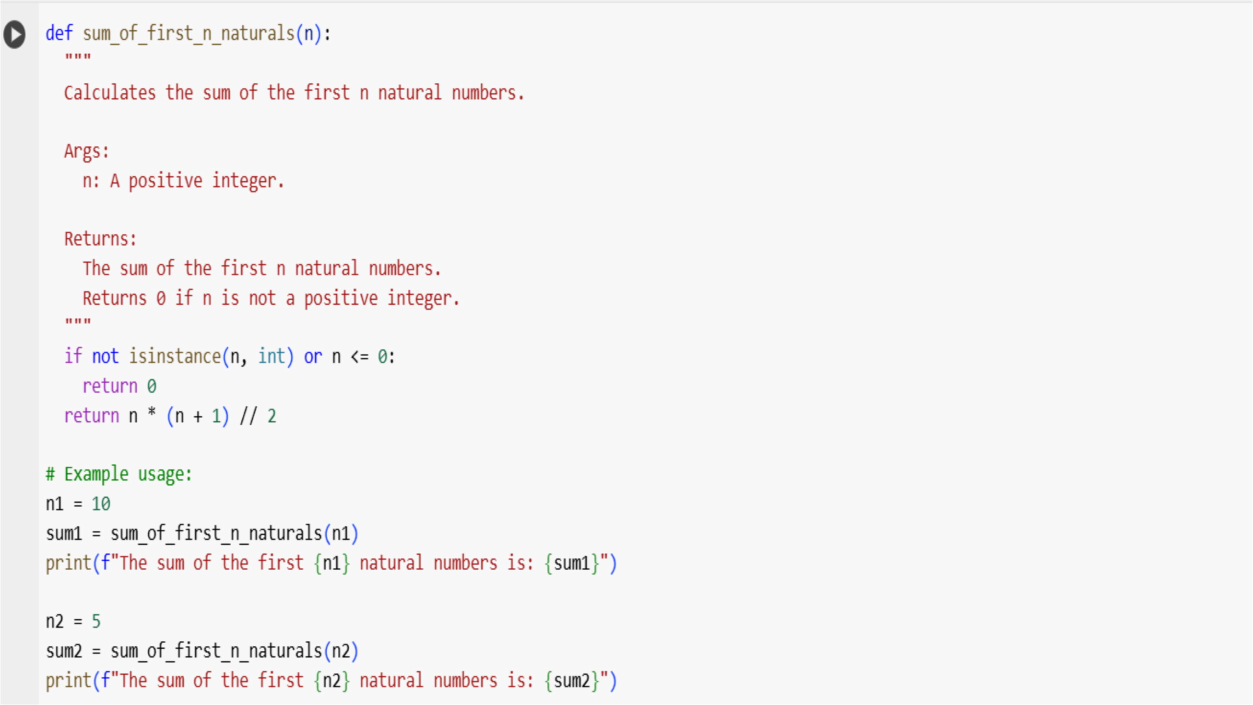
```
def is_prime(n):
  """
  Checks if a number is prime.

  A prime number is a natural number greater than 1 that has no positive
  divisors other than 1 and itself.

  Args:
    n: An integer.

  Returns:
    True if the number is prime, False otherwise.
  """
  if n <= 1:
    return False  # Numbers less than or equal to 1 are not prime
  if n <= 3:
    return True   # 2 and 3 are prime numbers
  if n % 2 == 0 or n % 3 == 0:
    return False  # Eliminate multiples of 2 and 3

  # Check for prime by iterating through numbers of the form 6k ± 1
  i = 5
  while i * i <= n:
    if n % i == 0 or n % (i + 2) == 0:
```



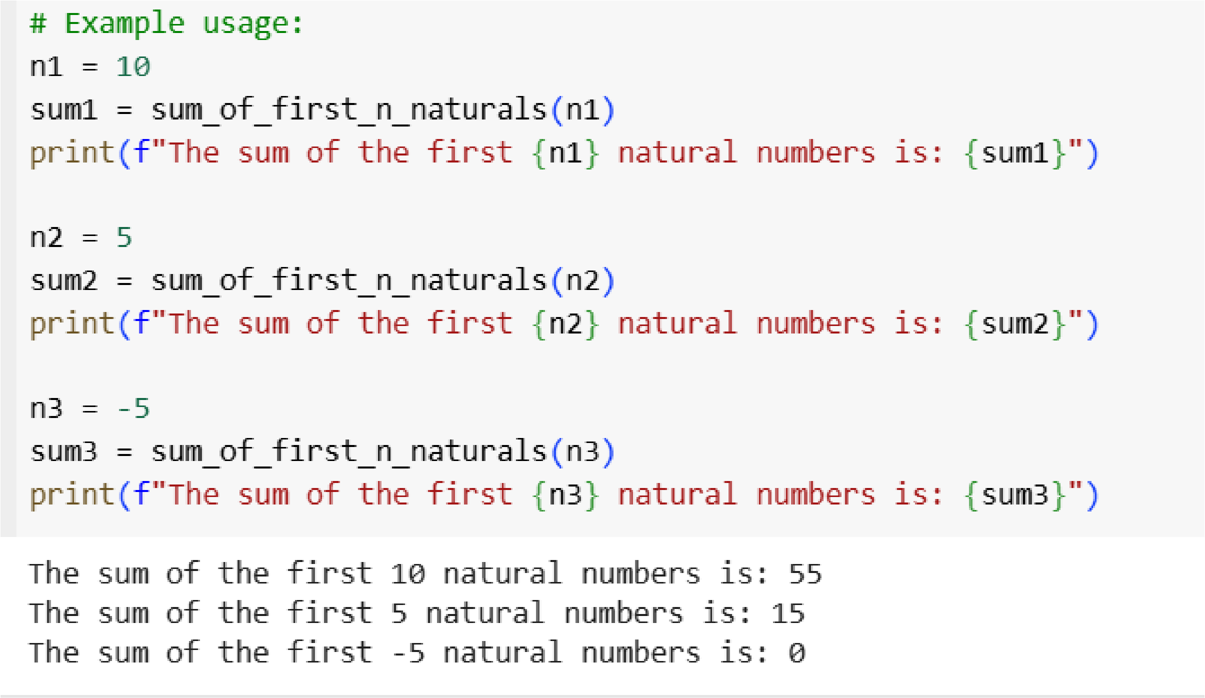
```
  i = 5
  while i * i <= n:
    if n % i == 0 or n % (i + 2) == 0:
      return False
    i += 6

  return True

# Example usage:
num = 17
print(f"{num} is prime: {is_prime(num)}")

num = 15
print(f"{num} is prime: {is_prime(num)}")
```

```
17 is prime: True
15 is prime: False
```

Task Description #4 ● Install and configure Cursor AI. Use it to generate a Python function (e.g., sum of the first N natural numbers) and test its output. ● Optionally, compare Cursor AI's generated code with Gemini's output. Expected Output #4 ● Screenshots of Cursor AI setup, prompts used, and generated code with output



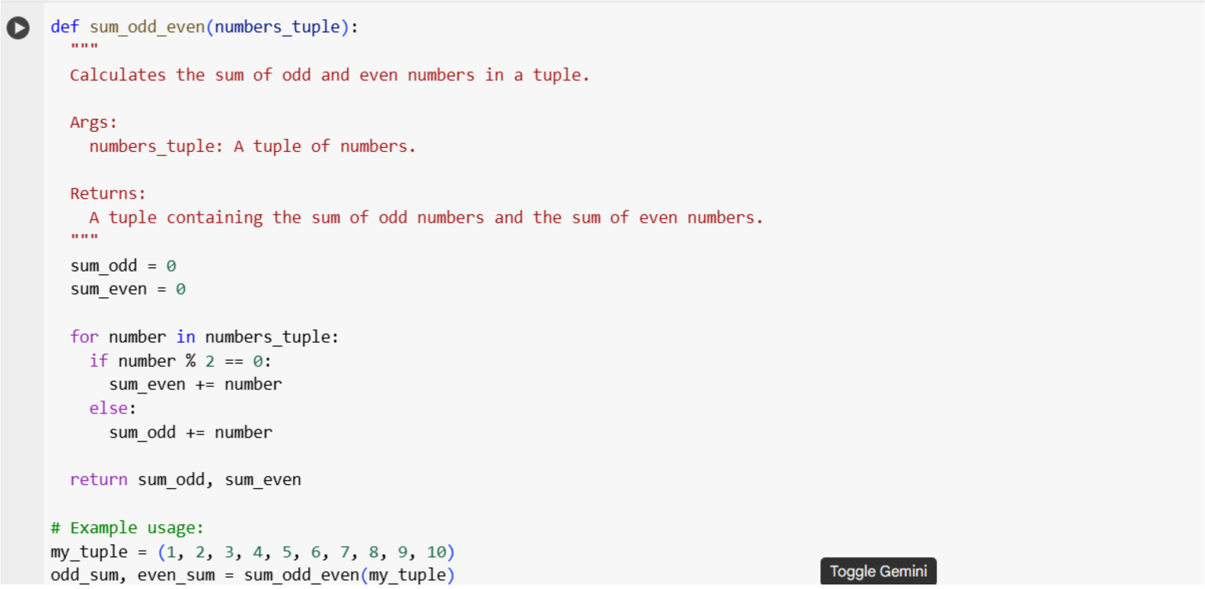
```python
def sum_of_first_n_naturals(n):
  """
  Calculates the sum of the first n natural numbers.

  Args:
    n: A positive integer.

  Returns:
    The sum of the first n natural numbers.
    Returns 0 if n is not a positive integer.
  """
  if not isinstance(n, int) or n <= 0:
    return 0
  return n * (n + 1) // 2

# Example usage:
n1 = 10
sum1 = sum_of_first_n_naturals(n1)
print(f"The sum of the first {n1} natural numbers is: {sum1}")

n2 = 5
sum2 = sum_of_first_n_naturals(n2)
print(f"The sum of the first {n2} natural numbers is: {sum2}")
```



```python
# Example usage:
n1 = 10
sum1 = sum_of_first_n_naturals(n1)
print(f"The sum of the first {n1} natural numbers is: {sum1}")

n2 = 5
sum2 = sum_of_first_n_naturals(n2)
print(f"The sum of the first {n2} natural numbers is: {sum2}")

n3 = -5
sum3 = sum_of_first_n_naturals(n3)
print(f"The sum of the first {n3} natural numbers is: {sum3}")
```

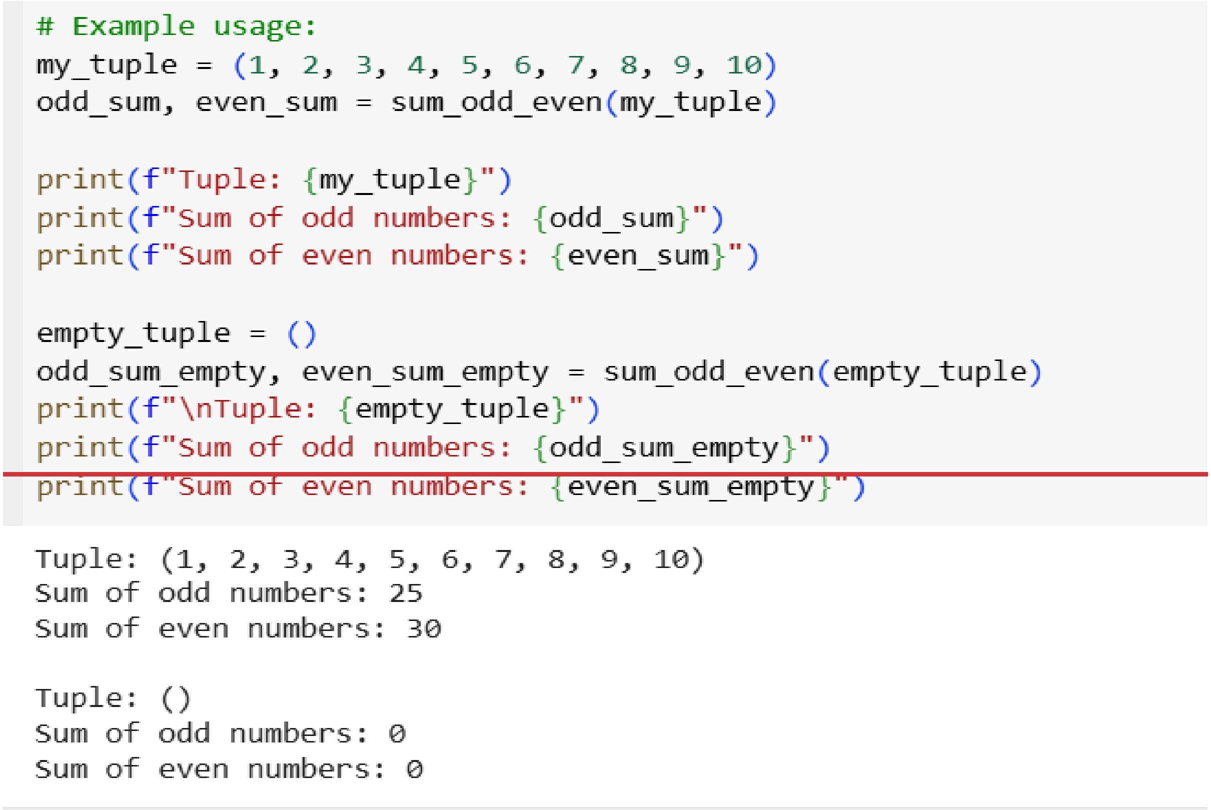
```
The sum of the first 10 natural numbers is: 55
The sum of the first 5 natural numbers is: 15
The sum of the first -5 natural numbers is: 0
```

Task Description #5 ● Students need to write a Python program to calculate the sum of odd numbers and even numbers in a given tuple Refactor the code to improve logic and readability. Expected Output #5 ● Student-written refactored code with explanations and output screenshots.



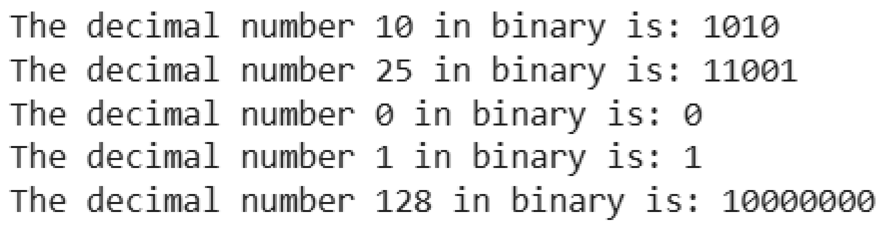
```
def sum_odd_even(numbers_tuple):
  """
  Calculates the sum of odd and even numbers in a tuple.

  Args:
    numbers_tuple: A tuple of numbers.

  Returns:
    A tuple containing the sum of odd numbers and the sum of even numbers.
  """
  sum_odd = 0
  sum_even = 0

  for number in numbers_tuple:
    if number % 2 == 0:
      sum_even += number
    else:
      sum_odd += number

  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)
```

~~BHG~~



```
# 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}")

empty_tuple = ()
odd_sum_empty, even_sum_empty = sum_odd_even(empty_tuple)
print(f"\nTuple: {empty_tuple}")
print(f"Sum of odd numbers: {odd_sum_empty}")
print(f"Sum of even numbers: {even_sum_empty}")
```

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

Tuple: ()
Sum of odd numbers: 0
Sum of even numbers: 0
```



```
The decimal number 10 in binary is: 1010
The decimal number 25 in binary is: 11001
The decimal number 0 in binary is: 0
The decimal number 1 in binary is: 1
The decimal number 128 in binary is: 10000000
```