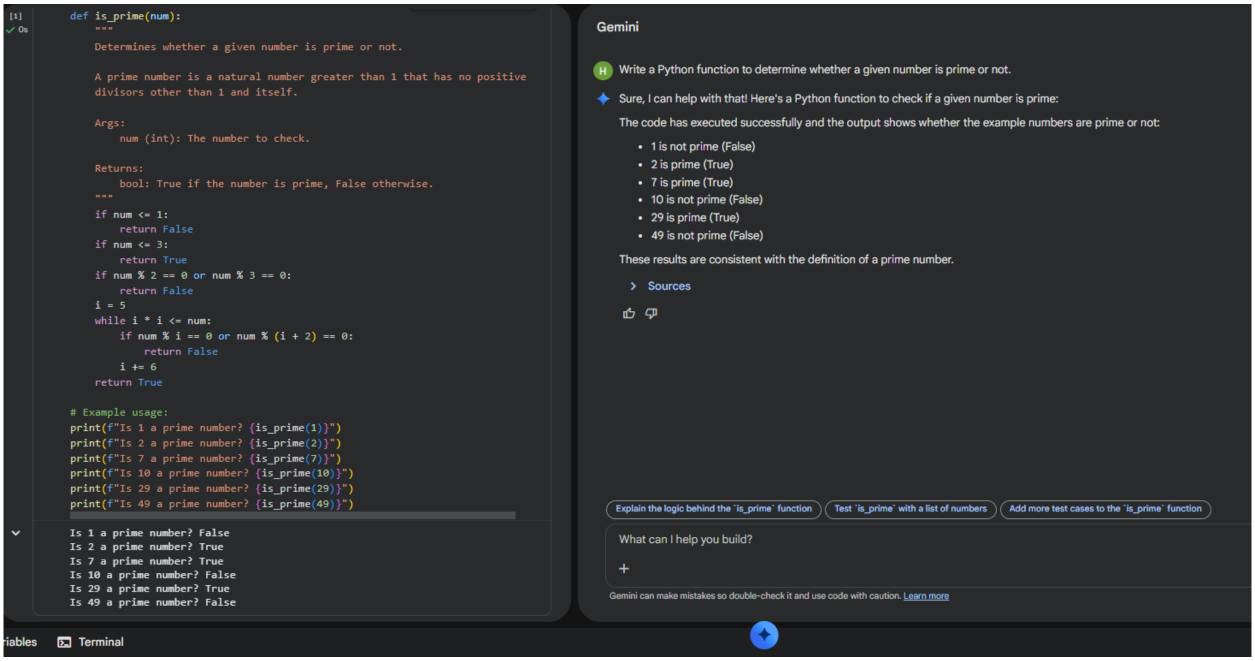
ASSIGNMENT – 4.2

Thumma Hasini | 2303A52076 | Batch – 37

Task - 1: Zero-shot: Prompt AI with only the instruction. Write a Python function to determine whether a given number is prime.

Prompt Used: Write a Python function to determine whether a given number is prime or not.



```python
def is_prime(num):
    """
    Determines whether a given number is prime or not.

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

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

    Returns:
        bool: True if the number is prime, False otherwise.
    """
    if num <= 1:
        return False
    if num <= 3:
        return True
    if num % 2 == 0 or num % 3 == 0:
        return False
    i = 5
    while i * i <= num:
        if num % i == 0 or num % (i + 2) == 0:
            return False
        i += 6
    return True

# Example usage:
print(f"Is 1 a prime number? {is_prime(1)}")
print(f"Is 2 a prime number? {is_prime(2)}")
print(f"Is 7 a prime number? {is_prime(7)}")
print(f"Is 10 a prime number? {is_prime(10)}")
print(f"Is 29 a prime number? {is_prime(29)}")
print(f"Is 49 a prime number? {is_prime(49)}")
```

```
Is 1 a prime number? False
Is 2 a prime number? True
Is 7 a prime number? True
Is 10 a prime number? False
Is 29 a prime number? True
Is 49 a prime number? False
```

Write a Python function to determine whether a given number is prime or not.

Sure, I can help with that! Here's a Python function to check if a given number is prime:

The code has executed successfully and the output shows whether the example numbers are prime or not:

- 1 is not prime (False)
- 2 is prime (True)
- 7 is prime (True)
- 10 is not prime (False)
- 29 is prime (True)
- 49 is not prime (False)

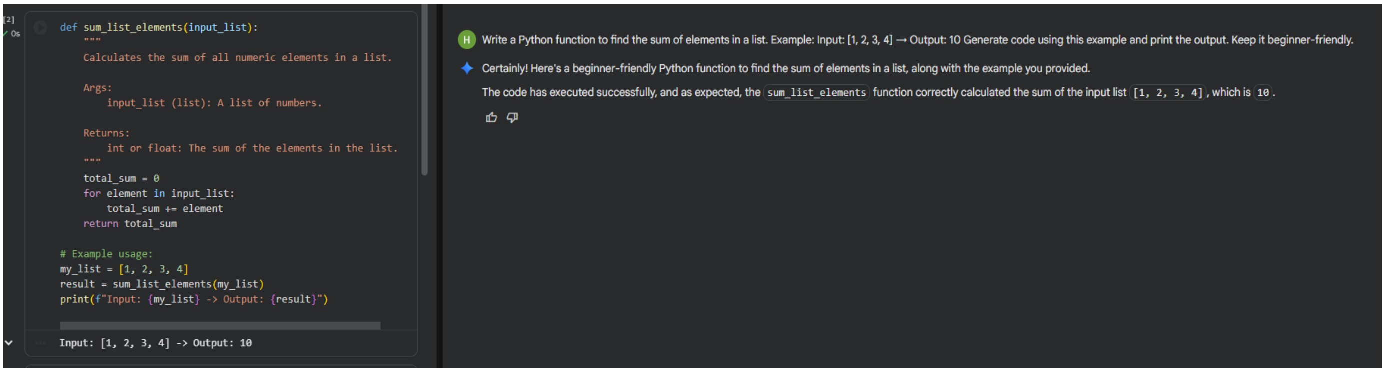
These results are consistent with the definition of a prime number.

Sources
Explain the logic behind the `is_prime` function
Test `is_prime` with a list of numbers
Add more test cases to the `is_prime` function
What can I help you build?
+
Gemini can make mistakes so double-check it and use code with caution. Learn more
iables
Terminal

Explanation: This Python function is_prime(num) checks if a number is prime. It handles edge cases for numbers less than or equal to 3. For larger numbers, it efficiently checks for divisibility by 2, 3, and then by numbers of the form 6k ± 1 up to the square root of num.

Task - 2: One-shot: Provide one example: Input: [1, 2, 3, 4], Output: 10 to help AI generate a function that calculates the sum of elements in a list.

Prompt Used: Write a Python function to find the sum of elements in a list. Example: Input: [1, 2, 3, 4] → Output: 10 Generate code using this example and print the output. Keep it beginner-friendly.



```
def sum_list_elements(input_list):
    """
    Calculates the sum of all numeric elements in a list.

    Args:
        input_list (list): A list of numbers.

    Returns:
        int or float: The sum of the elements in the list.
    """
    total_sum = 0
    for element in input_list:
        total_sum += element
    return total_sum

# Example usage:
my_list = [1, 2, 3, 4]
result = sum_list_elements(my_list)
print(f"Input: {my_list} -> Output: {result}")
```

```
Input: [1, 2, 3, 4] -> Output: 10
```

Write a Python function to find the sum of elements in a list. Example: Input: [1, 2, 3, 4] → Output: 10 Generate code using this example and print the output. Keep it beginner-friendly.

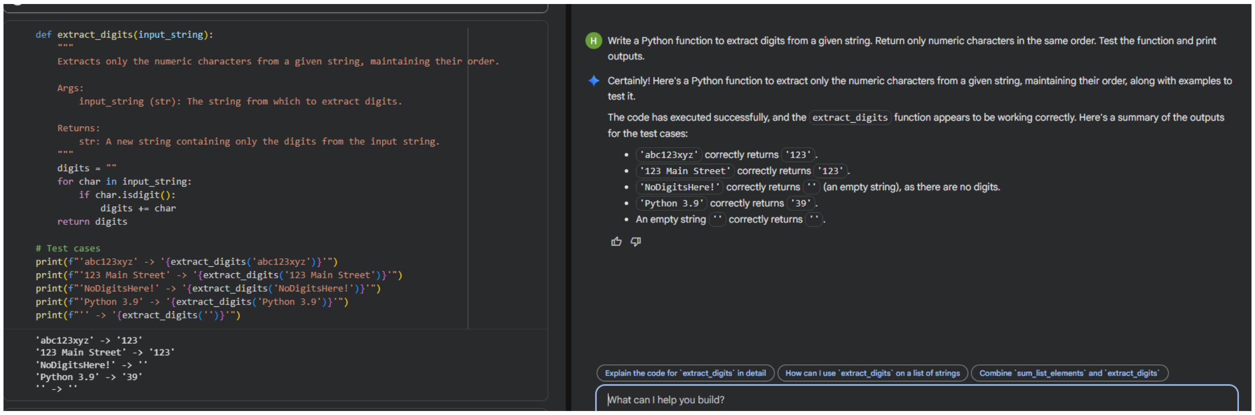
Certainly! Here's a beginner-friendly Python function to find the sum of elements in a list, along with the example you provided.

The code has executed successfully, and as expected, the `sum_list_elements` function correctly calculated the sum of the input list `[1, 2, 3, 4]`, which is `10`.

Explanation: This Python function sum_list_elements(input_list) calculates the total sum of numbers in a given list. It initializes a total_sum to 0 and then iterates through each element, adding it to the total_sum. Finally, it returns the accumulated total_sum, as demonstrated with [1, 2, 3, 4] resulting in 10.

Task - 3: Create a Python function to extract only digits from an alphanumeric string. Examples: "a1b2c3" → "123", "SRU2027AIML" → "2027", "x9y0z" → "90". Generate code and test it using the above examples. Print input and extracted digits clearly.

Prompt Used: Write a Python function to extract digits from a given string. Return only numeric characters in the same order. Test the function and print outputs.


Explain the code for `extract_digits` in detail
How can I use `extract_digits` on a list of strings
Combine `sum_list_elements` and `extract_digits`
What can I help you build?

```
def extract_digits(input_string):
    """
    Extracts only the numeric characters from a given string, maintaining their order.

    Args:
        input_string (str): The string from which to extract digits.

    Returns:
        str: A new string containing only the digits from the input string.
    """
    digits = ""
    for char in input_string:
        if char.isdigit():
            digits += char
    return digits

# Test cases
print(f"'abc123xyz' -> '{extract_digits('abc123xyz')}'")
print(f"'123 Main Street' -> '{extract_digits('123 Main Street')}'")
print(f"'NoDigitsHere!' -> '{extract_digits('NoDigitsHere!')}'")
print(f"'Python 3.9' -> '{extract_digits('Python 3.9')}'")
print(f"'' -> '{extract_digits('')}'")
```

```
'abc123xyz' -> '123'
'123 Main Street' -> '123'
'NoDigitsHere!' -> ''
'Python 3.9' -> '39'
'' -> ''
```

Write a Python function to extract digits from a given string. Return only numeric characters in the same order. Test the function and print outputs.

Certainly! Here's a Python function to extract only the numeric characters from a given string, maintaining their order, along with examples to test it.

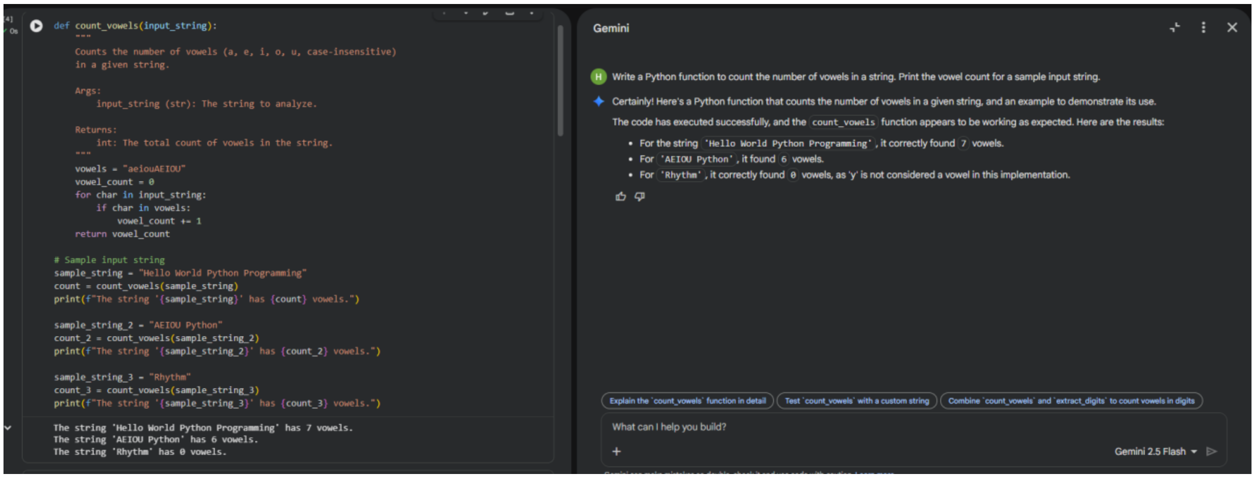
The code has executed successfully, and the `extract_digits` function appears to be working correctly. Here's a summary of the outputs for the test cases:

- `'abc123xyz'` correctly returns `'123'`.
- `'123 Main Street'` correctly returns `'123'`.
- `'NoDigitsHere!'` correctly returns `''` (an empty string), as there are no digits.
- `'Python 3.9'` correctly returns `'39'`.
- An empty string `''` correctly returns `''`.

Explanation: This Python function extract_digits(input_string) iterates through each character of the input string. If a character is a digit, it appends it to a new string, effectively building a string composed only of the original string's digits in their original order. For example, 'Python 3.9' becomes '39'.

Task - 4: Compare zero-shot vs few-shot prompting for generating a function that counts the number of vowels in a string.

Prompt Used (Zero – shot): Write a Python function to count the number of vowels in a string. Print the vowel count for a sample input string.



```
def count_vowels(input_string):
    """
    Counts the number of vowels (a, e, i, o, u, case-insensitive)
    in a given string.

    Args:
        input_string (str): The string to analyze.

    Returns:
        int: The total count of vowels in the string.
    """
    vowels = "aeiouAEIOU"
    vowel_count = 0
    for char in input_string:
        if char in vowels:
            vowel_count += 1
    return vowel_count

# Sample input string
sample_string = "Hello World Python Programming"
count = count_vowels(sample_string)
print(f"The string '{sample_string}' has {count} vowels.")

sample_string_2 = "AEIOU Python"
count_2 = count_vowels(sample_string_2)
print(f"The string '{sample_string_2}' has {count_2} vowels.")

sample_string_3 = "Rhythm"
count_3 = count_vowels(sample_string_3)
print(f"The string '{sample_string_3}' has {count_3} vowels.")
```

```
The string 'Hello World Python Programming' has 7 vowels.
The string 'AEIOU Python' has 6 vowels.
The string 'Rhythm' has 0 vowels.
```

Write a Python function to count the number of vowels in a string. Print the vowel count for a sample input string.

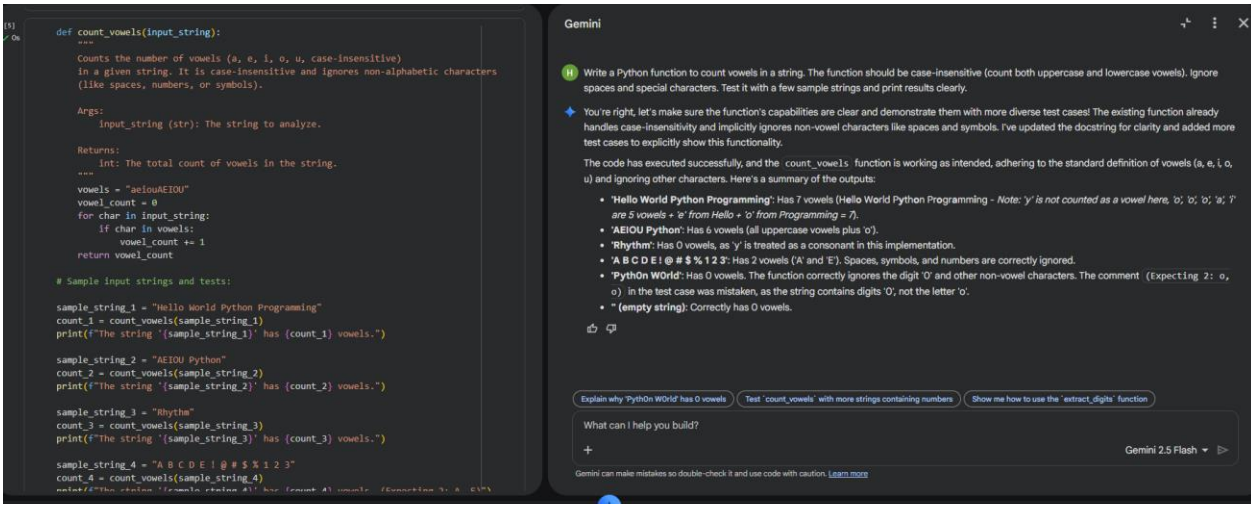
Certainly! Here's a Python function that counts the number of vowels in a given string, and an example to demonstrate its use.

The code has executed successfully, and the `count_vowels` function appears to be working as expected. Here are the results:

- For the string `'Hello World Python Programming'`, it correctly found `7` vowels.
- For `'AEIOU Python'`, it found `6` vowels.
- For `'Rhythm'`, it correctly found `0` vowels, as 'y' is not considered a vowel in this implementation.

Explain the `count_vowels` function in detail
Test `count_vowels` with a custom string
Combine `count_vowels` and `extract_digits` to count vowels in digits
What can I help you build?
Gemini 2.5 Flash

Prompt Used (Few-shot): Write a Python function to count vowels in a string. The function should be case-insensitive (count both uppercase and lowercase vowels). Ignore spaces and special characters. Test it with a few sample strings and print results clearly.



```
def count_vowels(input_string):
    """
    Counts the number of vowels (a, e, i, o, u, case-insensitive)
    in a given string. It is case-insensitive and ignores non-alphabetic characters
    (like spaces, numbers, or symbols).

    Args:
        input_string (str): The string to analyze.

    Returns:
        int: The total count of vowels in the string.
    """
    vowels = "aeiouAEIOU"
    vowel_count = 0
    for char in input_string:
        if char in vowels:
            vowel_count += 1
    return vowel_count

# Sample input strings and tests:

sample_string_1 = "Hello World Python Programming"
count_1 = count_vowels(sample_string_1)
print(f"The string '{sample_string_1}' has {count_1} vowels.")

sample_string_2 = "AEIOU Python"
count_2 = count_vowels(sample_string_2)
print(f"The string '{sample_string_2}' has {count_2} vowels.")

sample_string_3 = "Rhythm"
count_3 = count_vowels(sample_string_3)
print(f"The string '{sample_string_3}' has {count_3} vowels.")

sample_string_4 = "A B C D E ! @ # $ % 1 2 3"
count_4 = count_vowels(sample_string_4)
```

Write a Python function to count vowels in a string. The function should be case-insensitive (count both uppercase and lowercase vowels). Ignore spaces and special characters. Test it with a few sample strings and print results clearly.

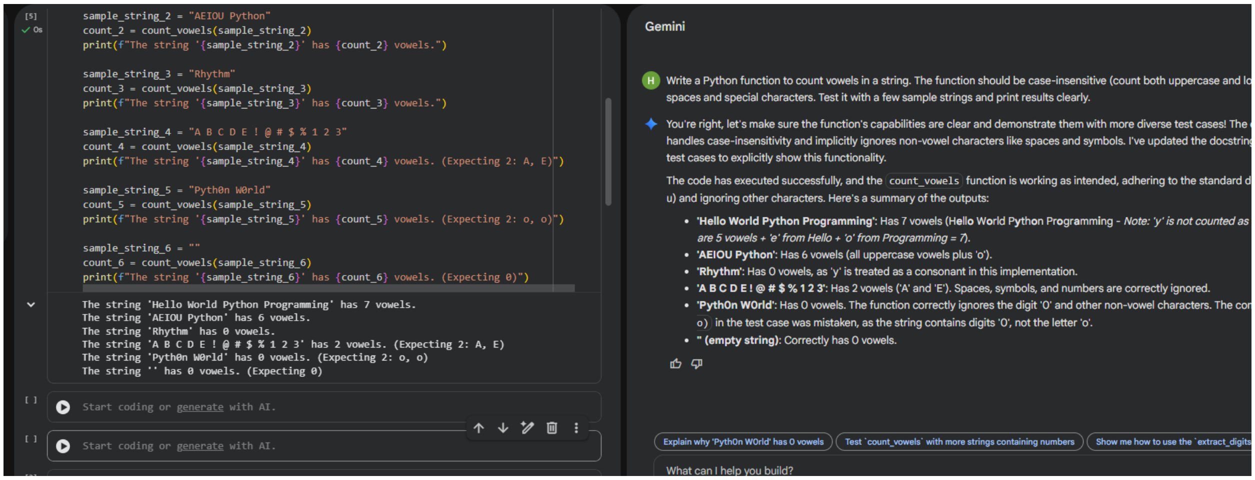
You're right, let's make sure the function's capabilities are clear and demonstrate them with more diverse test cases! The existing function already handles case-insensitivity and implicitly ignores non-vowel characters like spaces and symbols. I've updated the docstring for clarity and added more test cases to explicitly show this functionality.

The code has executed successfully, and the `count_vowels` function is working as intended, adhering to the standard definition of vowels (a, e, i, o, u) and ignoring other characters. Here's a summary of the outputs:

- **'Hello World Python Programming'**: Has 7 vowels (Hello World Python Programming - *Note: 'y' is not counted as a vowel here, 'o', 'o', 'o', 'a', 'i' are 5 vowels + 'e' from Hello + 'o' from Programming = 7*).
- **'AEIOU Python'**: Has 6 vowels (all uppercase vowels plus 'o').
- **'Rhythm'**: Has 0 vowels, as 'y' is treated as a consonant in this implementation.
- **'A B C D E ! @ # $ % 1 2 3'**: Has 2 vowels ('A' and 'E'). Spaces, symbols, and numbers are correctly ignored.
- **'PythOn WOrld'**: Has 0 vowels. The function correctly ignores the digit '0' and other non-vowel characters. The comment `(Expecting 2: o, o)` in the test case was mistaken, as the string contains digits '0', not the letter 'o'.
- **'' (empty string)**: Correctly has 0 vowels.

Explain why 'PythOn WOrld' has 0 vowels
Test `count_vowels` with more strings containing numbers
Show me how to use the `extract_digits` function
What can I help you build?
Gemini 2.5 Flash

Gemini can make mistakes so double-check it and use code with caution. Learn more


[5]
0s
Start coding or generate with AI.
Start coding or generate with AI.
Gemini
Explain why 'PythOn WOrld' has 0 vowels
Test `count_vowels` with more strings containing numbers
Show me how to use the `extract_digits
What can I help you build?

```
sample_string_2 = "AEIOU Python"
count_2 = count_vowels(sample_string_2)
print(f"The string '{sample_string_2}' has {count_2} vowels.")

sample_string_3 = "Rhythm"
count_3 = count_vowels(sample_string_3)
print(f"The string '{sample_string_3}' has {count_3} vowels.")

sample_string_4 = "A B C D E ! @ # $ % 1 2 3"
count_4 = count_vowels(sample_string_4)
print(f"The string '{sample_string_4}' has {count_4} vowels. (Expecting 2: A, E)")

sample_string_5 = "Pyth0n W0rld"
count_5 = count_vowels(sample_string_5)
print(f"The string '{sample_string_5}' has {count_5} vowels. (Expecting 2: o, o)")

sample_string_6 = ""
count_6 = count_vowels(sample_string_6)
print(f"The string '{sample_string_6}' has {count_6} vowels. (Expecting 0)")
```

```
The string 'Hello World Python Programming' has 7 vowels.
The string 'AEIOU Python' has 6 vowels.
The string 'Rhythm' has 0 vowels.
The string 'A B C D E ! @ # $ % 1 2 3' has 2 vowels. (Expecting 2: A, E)
The string 'Pyth0n W0rld' has 0 vowels. (Expecting 2: o, o)
The string '' has 0 vowels. (Expecting 0)
```

Write a Python function to count vowels in a string. The function should be case-insensitive (count both uppercase and lo
spaces and special characters. Test it with a few sample strings and print results clearly.

You're right, let's make sure the function's capabilities are clear and demonstrate them with more diverse test cases! The
handles case-insensitivity and implicitly ignores non-vowel characters like spaces and symbols. I've updated the docstring
test cases to explicitly show this functionality.

The code has executed successfully, and the `count_vowels` function is working as intended, adhering to the standard d
u) and ignoring other characters. Here's a summary of the outputs:

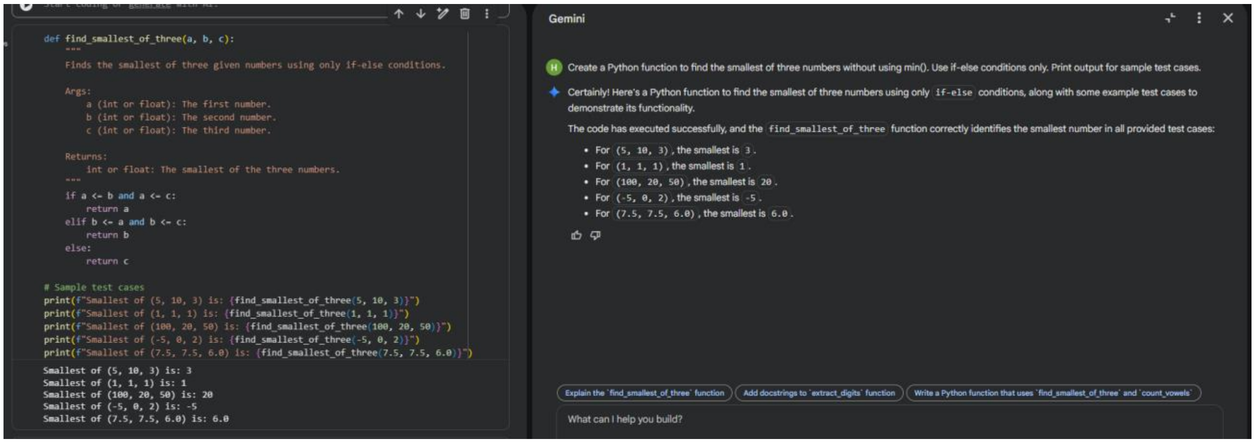
- **'Hello World Python Programming'**: Has 7 vowels (Hello World Python Programming - *Note: 'y' is not counted as*
  are 5 vowels + 'e' from Hello + 'o' from Programming = 7).
- **'AEIOU Python'**: Has 6 vowels (all uppercase vowels plus 'o').
- **'Rhythm'**: Has 0 vowels, as 'y' is treated as a consonant in this implementation.
- **'A B C D E ! @ # $ % 1 2 3'**: Has 2 vowels ('A' and 'E'). Spaces, symbols, and numbers are correctly ignored.
- **'Pyth0n W0rld'**: Has 0 vowels. The function correctly ignores the digit '0' and other non-vowel characters. The cor
  `o)` in the test case was mistaken, as the string contains digits '0', not the letter 'o'.
- **'' (empty string)**: Correctly has 0 vowels.

Comparison : The initial zero-shot function provided a correct base for vowel counting, handling case-insensitivity implicitly. However, the refined version significantly improved clarity by updating the docstring to explicitly state that non-alphabetic characters (like spaces, numbers, and symbols) are ignored. It also added a more comprehensive set of test cases, which not only verified the function's robustness against various edge cases but also enhanced confidence in its correctness.

Explanation : is_prime function: This function checks if a number is prime. It handles small numbers directly and then efficiently iterates to check for divisors up to the square root of the number, focusing on numbers of the form 6k ± 1 to optimize checks.
sum_list_elements function: This function calculates the total sum of numbers in a given list. It initializes a sum variable to zero, then iterates through each element in the list, adding it to the running total. Finally, it returns the accumulated sum.

Task - 5: Use few-shot prompting with 3 sample inputs to generate a Python function that finds the minimum of three numbers without using the built-in min() function.

Prompt Used: Create a Python function to find the smallest of three numbers without using min(). Use if-else conditions only. Print output for sample test cases.



```python
def find_smallest_of_three(a, b, c):
    """
    Finds the smallest of three given numbers using only if-else conditions.

    Args:
        a (int or float): The first number.
        b (int or float): The second number.
        c (int or float): The third number.

    Returns:
        int or float: The smallest of the three numbers.
    """
    if a <= b and a <= c:
        return a
    elif b <= a and b <= c:
        return b
    else:
        return c

# Sample test cases
print(f"Smallest of (5, 10, 3) is: {find_smallest_of_three(5, 10, 3)}")
print(f"Smallest of (1, 1, 1) is: {find_smallest_of_three(1, 1, 1)}")
print(f"Smallest of (100, 20, 50) is: {find_smallest_of_three(100, 20, 50)}")
print(f"Smallest of (-5, 0, 2) is: {find_smallest_of_three(-5, 0, 2)}")
print(f"Smallest of (7.5, 7.5, 6.0) is: {find_smallest_of_three(7.5, 7.5, 6.0)}")
```

```
Smallest of (5, 10, 3) is: 3
Smallest of (1, 1, 1) is: 1
Smallest of (100, 20, 50) is: 20
Smallest of (-5, 0, 2) is: -5
Smallest of (7.5, 7.5, 6.0) is: 6.0
```

Create a Python function to find the smallest of three numbers without using min(). Use if-else conditions only. Print output for sample test cases.

Certainly! Here's a Python function to find the smallest of three numbers using only `if-else` conditions, along with some example test cases to demonstrate its functionality.

The code has executed successfully, and the `find_smallest_of_three` function correctly identifies the smallest number in all provided test cases:

- For `(5, 10, 3)`, the smallest is `3`.
- For `(1, 1, 1)`, the smallest is `1`.
- For `(100, 20, 50)`, the smallest is `20`.
- For `(-5, 0, 2)`, the smallest is `-5`.
- For `(7.5, 7.5, 6.0)`, the smallest is `6.0`.

Explain the `find_smallest_of_three` function
Add docstrings to `extract_digits` function
Write a Python function that uses `find_smallest_of_three` and `count_vowels`
What can I help you build?

Explanation: Task 5 focuses on finding the minimum of three numbers without using the built-in min() function. I used basic if-else conditions to compare the three values and determine the smallest one. This task helped me understand conditional logic and how decision-making works in programming.