

## AI ASSISTED CODING

**Name:** P.Swaran Raj

**Batch:** 37

**Roll.No:** 2303A52247

**Assignment- 4**

### **Task Description 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."**

The screenshot shows a terminal window on the left and a Gemini AI interface on the right. The terminal window displays the following code:

```
[1]  def is_prime(number):
    """
    Determines whether a given number is prime.

    Args:
        number: An integer.

    Returns:
        True if the number is prime, False otherwise.
    """
    if number < 2:
        return False
    for i in range(2, int(number**0.5) + 1):
        if number % i == 0:
            return False
    return True

    # Example usage:
    print("Is 7 prime? " + str(is_prime(7)))
    print("Is 10 prime? " + str(is_prime(10)))
    print("Is 2 prime? " + str(is_prime(2)))
    print("Is 1 prime? " + str(is_prime(1)))
    print("Is 0 prime? " + str(is_prime(0)))

...
Is 7 prime? True
Is 10 prime? False
Is 2 prime? True
Is 1 prime? False
Is 0 prime? False
```

The Gemini interface shows the prompt and the generated function. It also displays the output of the function for various numbers and a note that it correctly identifies prime and non-prime numbers.

### **Explanation:**

The AI generated a prime-checking function using only instructions. It checks numbers less than or equal to 1 and verifies divisibility up to the square root of the number.

### **Output:**

Is 7 prime? True  
Is 10 prime? False

Is 2 prime? True  
Is 1 prime? False  
Is 0 prime? False

## Observation:

The AI correctly generated logic without any examples.  
Zero-shot prompting works well for simple and well-defined problems.

## Task Description 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:

**"#Create a Python function to calculate the sum of elements in a list. Example: Input: [1, 2, 3, 4] → Output: 10"**

The screenshot shows a code editor on the left and a Gemini AI interface on the right. The code editor contains a Python script with a function definition for calculating the sum of elements in a list. The Gemini interface shows the user prompt and the AI's generated response, which includes the function code and its execution results.

```
[2] ✓ 0s def sum_list_elements(input_list):
    """
    Calculates the sum of all numerical elements in a list.

    Args:
        input_list: A list of numbers.

    Returns:
        The sum of the elements in the list.
    """
    total_sum = 0
    for item in input_list:
        total_sum += item
    return total_sum

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

another_list = [10, 20, 30]
print(f"Input: {another_list} -> Output: {sum_list_elements(another_list)}")
```

Gemini

Create a Python function to calculate the sum of elements in a list.  
Example: Input: [1, 2, 3, 4] → Output: 10

Certainly! Here's a Python function `sum_list_elements` that calculates the sum of all elements in a given list, along with an example:

The `sum_list_elements` function has executed successfully. The output demonstrates that the function correctly calculates the sum of elements for both `[1, 2, 3, 4]` (resulting in `10`) and `[10, 20, 30]` (resulting in `60`).

Add error handling to `sum_list_elements`.  
Explain the `is_prime` function's logic.  
Write a function to find the maximum element in a list.

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](#)

## Explanation:

The function initializes a total variable and iterates through the list. Each element is added to the total and the final sum is returned.

## Output:

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

Input: [10, 20, 30] -> Output: 60

## Observation:

Providing one example clearly guided the AI. One-shot prompting improved clarity and correctness.

## Task Description 3:

Few-shot: Give 2–3 examples to create a function that extracts digits from an alphanumeric string.

## Prompt used:

"#Create a function to extract digits from an alphanumeric string.

### Examples:

Input: 'a1b2c3' → Output: '123'

Input: 'x9y8z' → Output: '98'

Input: 'abc123' → Output: '123'"

The screenshot shows a code editor on the left and a Gemini AI interface on the right. The code editor contains the following Python code:

```
[3]  def extract_digits(alphanumeric_string):
    """
    Extracts all digits from an alphanumeric string.

    Args:
        alphanumeric_string: The input string containing letters and digits.

    Returns:
        A string containing only the digits found in the input string.

    digits = ''
    for char in alphanumeric_string:
        if char.isdigit():
            digits += char
    return digits

    # Example usage:
    print(f"Input: 'a1b2c3' -> Output: '{extract_digits('a1b2c3')}'")
    print(f"Input: 'x9y8z' -> Output: '{extract_digits('x9y8z')}'")
    print(f"Input: 'abc123' -> Output: '{extract_digits('abc123')}'")
    print(f"Input: 'no_digits_here' -> Output: '{extract_digits('no_digits_here')}'")
    """

    ... Input: 'a1b2c3' -> Output: '123'
    Input: 'x9y8z' -> Output: '98'
    Input: 'abc123' -> Output: '123'
    Input: 'no_digits_here' -> Output: ''
```

The Gemini interface shows the prompt and the generated Python function. It also displays the function's execution results for the provided examples.

## Explanation:

The AI generated a function that loops through the string and checks digits using `isdigit()`. Digits are appended to the result string and returned.

## Output:

Input: 'a1b2c3' -> Output: '123'

Input: 'x9y8z' -> Output: '98'

Input: 'abc123' -> Output: '123'

## Observation:

Multiple examples improved accuracy and logic understanding.

Few-shot prompting produced more reliable code.

## Task Description 4:

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

**Problem:** Generate a function to count the number of vowels in a string.

**Zero-Shot prompting for generating a function that counts the number of vowels in a string.:**

**"#Write a Python function to count the number of vowels in a string."**

The screenshot shows a Jupyter Notebook cell on the left and a Gemini AI interface on the right. The Jupyter cell contains the following Python code:

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

    Args:
        input_string: The string to analyze.

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

# Example usage:
print(f"'Hello World' has {count_vowels('Hello World')} vowels.")
print(f"'Python' has {count_vowels('Python')} vowels.")
print(f"'AEIOU' has {count_vowels('AEIOU')} vowels.")
print(f"'rhythm' has {count_vowels('rhythm')} vowels.")
print(f"'bcdgfh' has {count_vowels('bcdgfh')} vowels.")

'Hello World' has 3 vowels.
'Python' has 1 vowels.
'AEIOU' has 5 vowels.
'rhythm' has 0 vowels.
'bcdgfh' has 0 vowels.
```

The Gemini interface shows the prompt: "P Write a Python function to count the number of vowels in a string". It also displays the generated function and some example outputs:

- Certainly! Here's a Python function `count_vowels` that counts the number of vowels (a, e, i, o, u, case-insensitive) in a given string, along with some examples:
- The `count_vowels` function has executed successfully. The output demonstrates that it correctly counts the number of vowels in the given strings:
  - 'Hello World' has 3 vowels.
  - 'Python' has 1 vowels.
  - 'AEIOU' has 5 vowels.
  - 'rhythm' has 0 vowels.
  - 'bcdgfh' has 0 vowels.

Below the examples are buttons for "Apply the 'count\_vowels' function to a new string", "Explain the 'extract\_digits' function", and "Write a Python function to reverse a string". A "What can I help you build?" input field and a "Gemini 2.5 Flash" button are at the bottom.

## Output:

'Hello World' has 3 vowels.

'Python' has 1 vowels.  
'AEIOU' has 5 vowels.  
'rhythm' has 0 vowels.  
'bcdgfgh' has 0 vowels.

## Few-shot prompting for generating a function that counts the number of vowels in a string:

**“Create a function to count vowels.**

**Examples:**

**Input: ‘apple’ → Output: 2**

**Input: ‘education’ → Output: 5**

**Input: ‘sky’ → Output: 0”**

The screenshot shows a Jupyter Notebook cell containing Python code for a `count_vowels` function. The code includes docstrings and examples. To the right, the Gemini AI interface is shown, with a prompt asking to create a vowel-counting function and a history of previous examples and the resulting output.

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

    Args:
        input_string: The string to analyze.

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

# Example usage:
print("Hello World" has {count_vowels('Hello World')} vowels.)
print("Python" has {count_vowels('Python')} vowels.)
print("AEIOU" has {count_vowels('AEIOU')} vowels.)
print("rhythm" has {count_vowels('rhythm')} vowels.)
print("bcdgfgh" has {count_vowels('bcdgfgh')} vowels.)

# New examples:
print("apple" has {count_vowels('apple')} vowels.)
print("education" has {count_vowels('education')} vowels.)
print("sky" has {count_vowels('sky')} vowels.)
```

Gemini

P Create a function to count vowels. Examples: Input: ‘apple’ → Output: 2 Input: ‘education’ → Output: 5 Input: ‘sky’ → Output: 0

◆ You've asked for a function to count vowels, and I already have a `count_vowels` function in the notebook. I've updated the example usage in cell 58232831 to include the new examples you provided. The `count_vowels` function has executed successfully with the updated examples. Here's the complete output:

‘Hello World’ has 3 vowels.  
‘Python’ has 1 vowels.  
‘AEIOU’ has 5 vowels.  
‘rhythm’ has 0 vowels.

Explain the `count_vowels` function logic  
Modify `count_vowels` to count consonants  
Write a function to find the longest word in a string

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](#)

## Output:

'Hello World' has 3 vowels.  
'Python' has 1 vowels.  
'AEIOU' has 5 vowels.  
'rhythm' has 0 vowels.  
'bcdgfgh' has 0 vowels.  
'apple' has 2 vowels.  
'education' has 5 vowels.  
'sky' has 0 vowels.

## **Comparison & Explanation:**

<b>Aspect</b>	<b>Zero-Shot</b>	<b>Few-Shot</b>
Clarity	Basic logic	More optimized
Accuracy	Correct	Correct

## **Observation:**

Few-shot prompting produced clearer and optimized logic.  
Examples significantly improved AI understanding.

## **Task Description 5:**

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

## **Prompt used:**

**"#Create a function to find the minimum of three numbers without using min()."**

### **Examples:**

**Input: (3, 5, 1) → Output: 1  
Input: (10, 7, 8) → Output: 7  
Input: (-2, -5, 0) → Output: -5"**

```
[6] | 0s  def find_minimum_of_three(num1, num2, num3):
    """
    Finds the minimum of three given numbers without using the min() function.

    Args:
        num1: The first number.
        num2: The second number.
        num3: The third number.

    Returns:
        The smallest of the three numbers.
    """
    minimum = num1
    if num2 < minimum:
        minimum = num2
    if num3 < minimum:
        minimum = num3
    return minimum

# Example usage:
print(f"Input: (3, 5, 1) -> Output: {find_minimum_of_three(3, 5, 1)}")
print(f"Input: (10, 7, 8) -> Output: {find_minimum_of_three(10, 7, 8)}")
print(f"Input: (-2, -5, 0) -> Output: {find_minimum_of_three(-2, -5, 0)}")
```

Gemini

P Create a function to find the minimum of three numbers without using min(). Examples: Input: (3, 5, 1) → Output: 1 Input: (10, 7, 8) → Output: 7 Input: (-2, -5, 0) → Output: -5

◆ Certainly! Here's a Python function `find_minimum_of_three` that determines the smallest of three given numbers without using the built-in `min()` function, along with example usages:

The `find_minimum_of_three` function has executed successfully. The output demonstrates that it correctly identifies the minimum of the three given numbers:

Input: (3, 5, 1) -> Output: 1  
Input: (10, 7, 8) -> Output: 7  
Write a function to find the maximum of three numbers  
Add more test cases for `find_minimum_of_three`  
Explain the logic of `find_minimum_of_three`

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](#)

## Explanation:

The function assumes the first number is smallest and compares it with others. The smallest value is updated and returned.

## Output:

Input: (3, 5, 1) -> Output: 1  
Input: (10, 7, 8) -> Output: 7  
Input: (-2, -5, 0) -> Output: -5

## Observation:

Few-shot examples helped the AI infer comparison logic correctly. The solution avoided built-in functions as instructed.

## Conclusion:

- Zero-shot prompting works well for simple tasks but may lack optimization.
- One-shot prompting improves clarity using examples.
- Few-shot prompting provides the best accuracy, structure, and optimized solutions.