

AI ASSISTED CODING

Name: P.Swaran Raj
Batch: 37

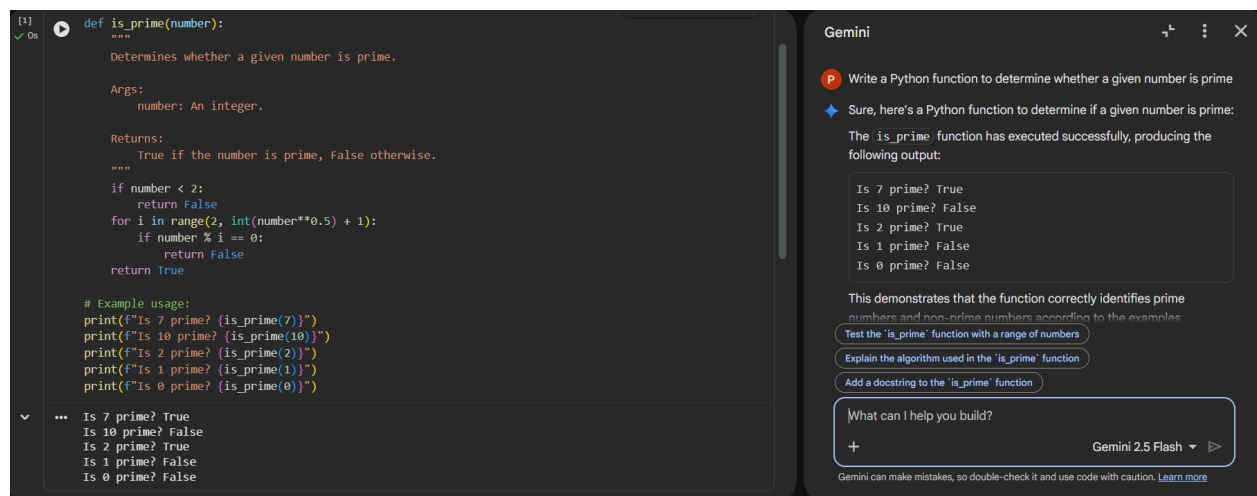
Roll.No: 2303A52247
Assignment- 4

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 displays a code editor on the left and a Gemini chat interface on the right. The code editor shows a Python function `is_prime` that checks if a number is prime. The function includes a docstring, arguments, and returns a boolean value. It uses a loop to check divisibility up to the square root of the number. Example usage is provided at the bottom of the function definition. The Gemini chat interface shows the prompt "Write a Python function to determine whether a given number is prime" and the response, which includes the function code and its output for various inputs: 7 (True), 10 (False), 2 (True), 1 (False), and 0 (False). The chat interface also includes buttons for testing the function, explaining the algorithm, and adding a docstring.

```
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(f"Is 7 prime? {is_prime(7)}")  
print(f"Is 10 prime? {is_prime(10)}")  
print(f"Is 2 prime? {is_prime(2)}")  
print(f"Is 1 prime? {is_prime(1)}")  
print(f"Is 0 prime? {is_prime(0)}")
```

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

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:

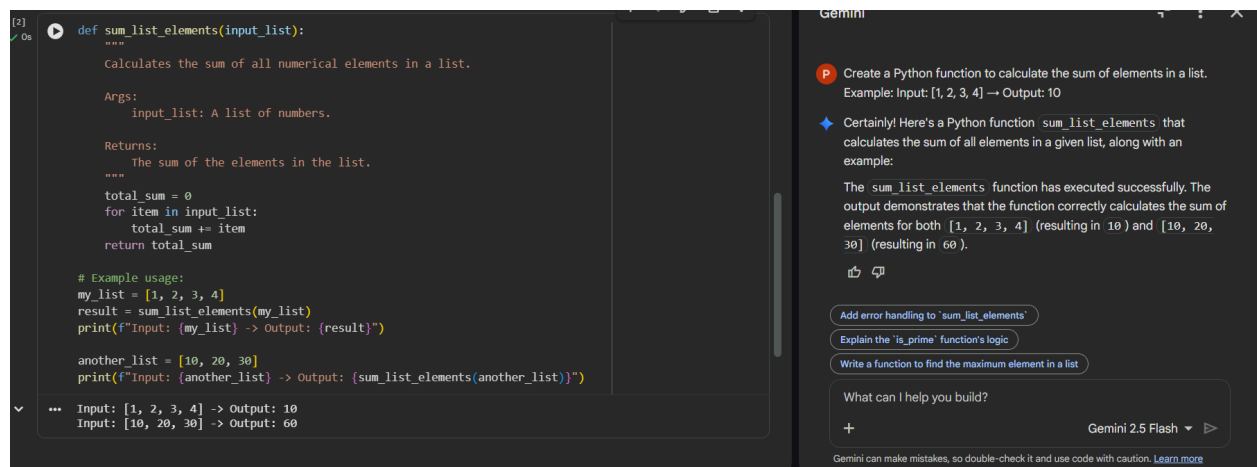
The AI correctly generated logic without any examples.
Zero-shot prompting works well for simple and well-defined problems.

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 displays a code editor on the left and a Gemini chat interface on the right. The code editor shows a Python function `sum_list_elements` that calculates the sum of all numerical elements in a list. The function includes docstrings for its purpose, arguments, and return value, as well as example usage code. The Gemini chat interface shows the prompt: "Create a Python function to calculate the sum of elements in a list. Example: Input: [1, 2, 3, 4] → Output: 10". The response confirms the function's execution and provides the output for the example input: 10. Below the response, there are suggestions for further actions: "Add error handling to 'sum_list_elements'", "Explain the 'is_prime' function's logic", and "Write a function to find the maximum element in a list". The chat interface also includes a search bar and a Gemini 2.5 Flash logo.

```
[2] 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)}")  
... Input: [1, 2, 3, 4] -> Output: 10  
Input: [10, 20, 30] -> Output: 60
```

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:

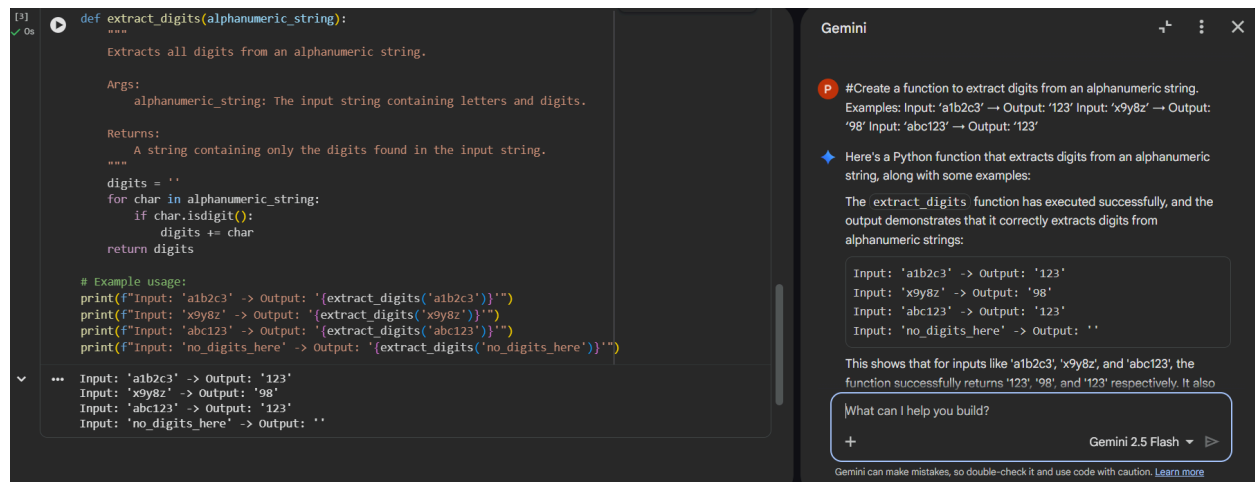
“#Create a function to extract digits from an alphanumeric string.

Examples:

Input: ‘a1b2c3’ → Output: ‘123’

Input: ‘x9y8z’ → Output: ‘98’

Input: ‘abc123’ → Output: ‘123’”



The screenshot displays a code editor on the left and a Gemini chat interface on the right. The code editor shows a Python function named `extract_digits` that takes an alphanumeric string and returns a string of its digits. The function uses a loop with `char.isdigit()` to identify digits. Below the function, example usage is demonstrated with print statements. The Gemini chat interface on the right shows the prompt: "#Create a function to extract digits from an alphanumeric string. Examples: Input: 'a1b2c3' -> Output: '123' Input: 'x9y8z' -> Output: '98' Input: 'abc123' -> Output: '123'". The chat response includes a Python function that matches the one in the code editor, followed by a summary of the function's behavior and a confirmation that it correctly extracts digits from the provided examples.

```
[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: ''

What can I help you build?

Gemini 2.5 Flash

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.

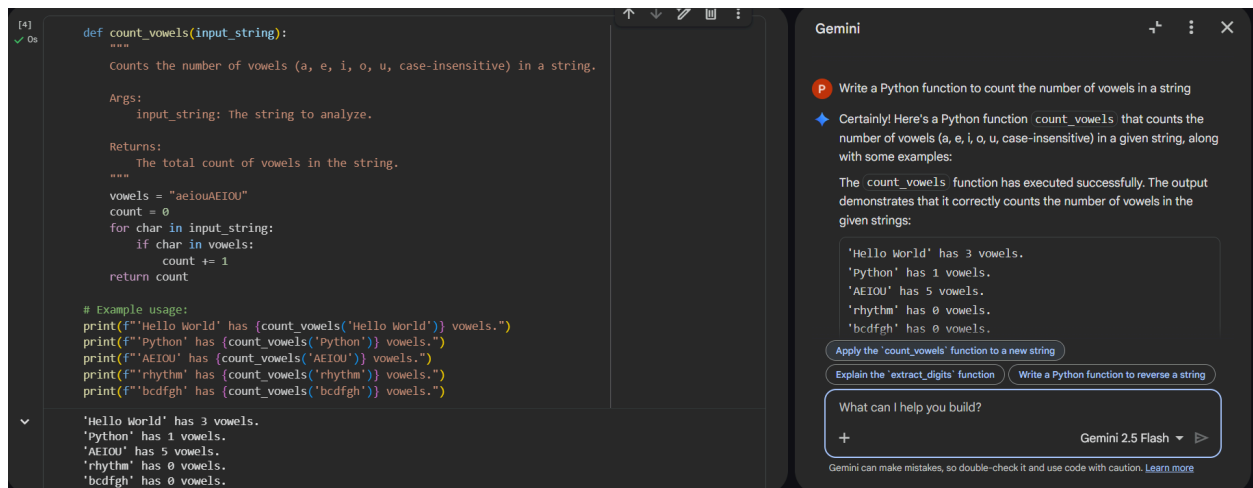
Task Description 4:

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 image shows a side-by-side comparison of a code editor and a Gemini chat interface. On the left, a code editor displays a Python function `count_vowels` that counts vowels in a string. The function includes docstrings, arguments, and example usage. On the right, the Gemini chat interface shows a prompt to write a Python function to count vowels, followed by a response that includes the same function code and example outputs.

```
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"bcd fgh' has {count_vowels('bcd fgh')} vowels.")
```

Output:

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

Output:

'Hello World' has 3 vowels.

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

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

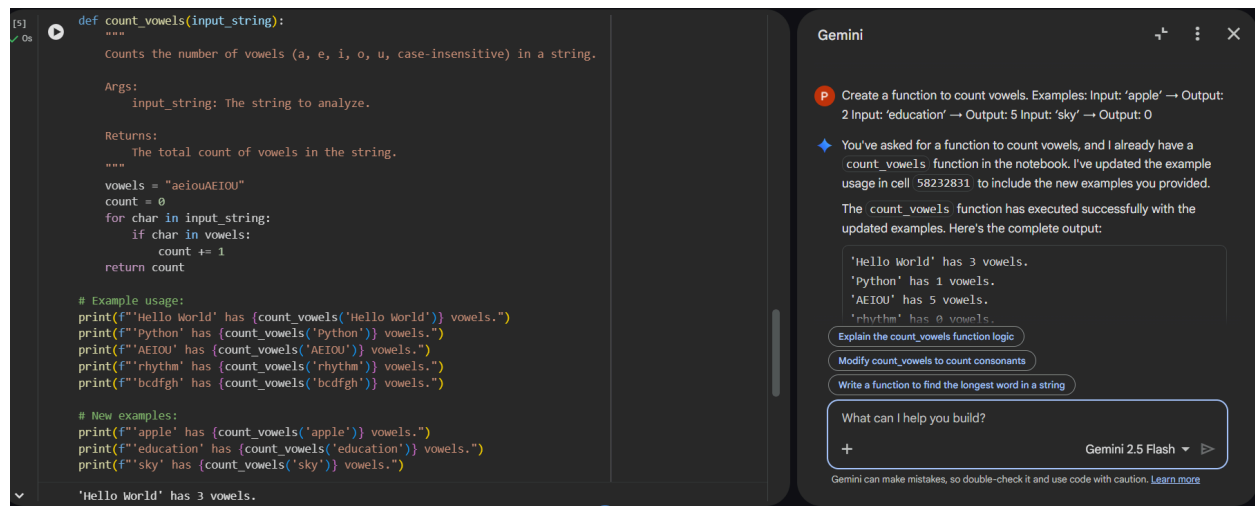
“Create a function to count vowels.

Examples:

Input: ‘apple’ → Output: 2

Input: ‘education’ → Output: 5

Input: ‘sky’ → Output: 0”



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

Output:

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

Output:

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

Comparison & Explanation:

Aspect	Zero-Shot	Few-Shot
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:

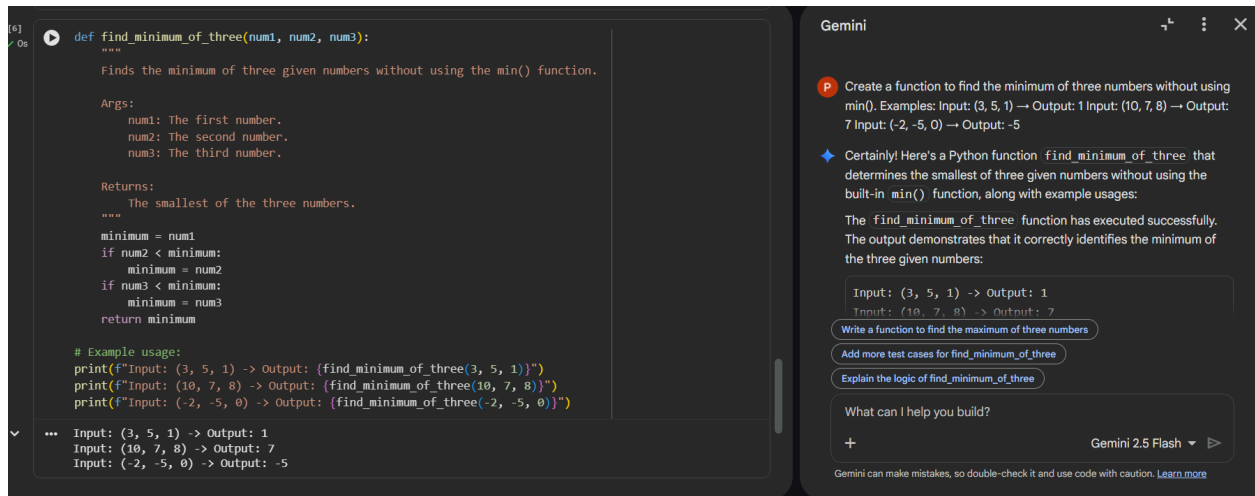
“#Create a function to find the minimum of three numbers without using `min()`.

Examples:

Input: (3, 5, 1) → Output: 1

Input: (10, 7, 8) → Output: 7

Input: (-2, -5, 0) → Output: -5”



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.