

Course Title : AI Assisted Coding

Name : M Pravalika

Batch : 34

Hall No : 2303a52347

Lab 4: Advanced Prompt Engineering – Zero-shot, One-shot, and Few-shot Techniques

Task Description-1

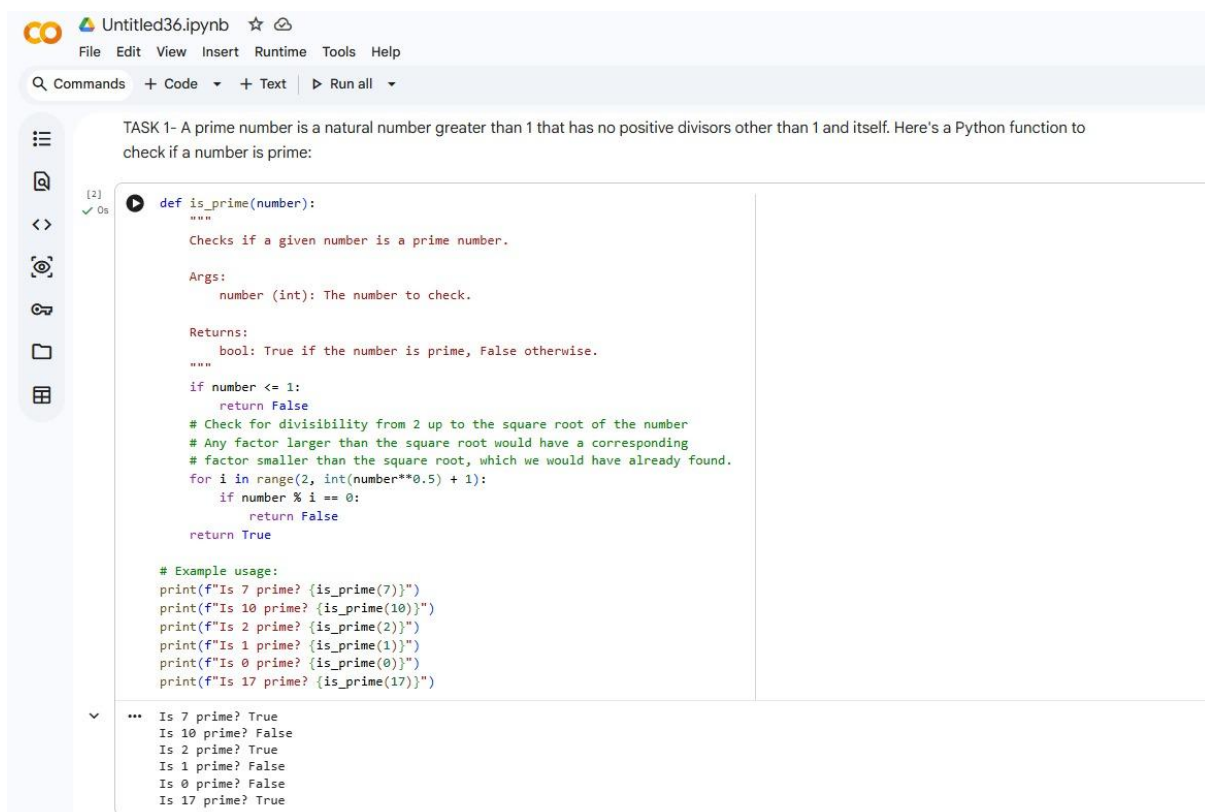
- **Zero-shot: Prompt AI with only the instruction. Write a Python function to determine**

whether a given number is prime

Expected Output-1

- **A basic Python function to check if a number is prime, demonstrating correct logical**

conditions without relying on examples or additional context



The screenshot shows a Jupyter Notebook titled 'Untitled36.ipynb'. The interface includes a menu bar (File, Edit, View, Insert, Runtime, Tools, Help) and a toolbar with icons for commands, code, text, and running all cells. The notebook content is as follows:

TASK 1- A prime number is a natural number greater than 1 that has no positive divisors other than 1 and itself. Here's a Python function to check if a number is prime:

```
[2] ✓ Os
def is_prime(number):
    """
    Checks if a given number is a prime number.

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

    Returns:
        bool: True if the number is prime, False otherwise.
    """
    if number <= 1:
        return False
    # Check for divisibility from 2 up to the square root of the number
    # Any factor larger than the square root would have a corresponding
    # factor smaller than the square root, which we would have already found.
    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)}")
print(f"Is 17 prime? {is_prime(17)}")
```

The output of the notebook execution is shown at the bottom:

```
... Is 7 prime? True
Is 10 prime? False
Is 2 prime? True
Is 1 prime? False
Is 0 prime? False
Is 17 prime? True
```

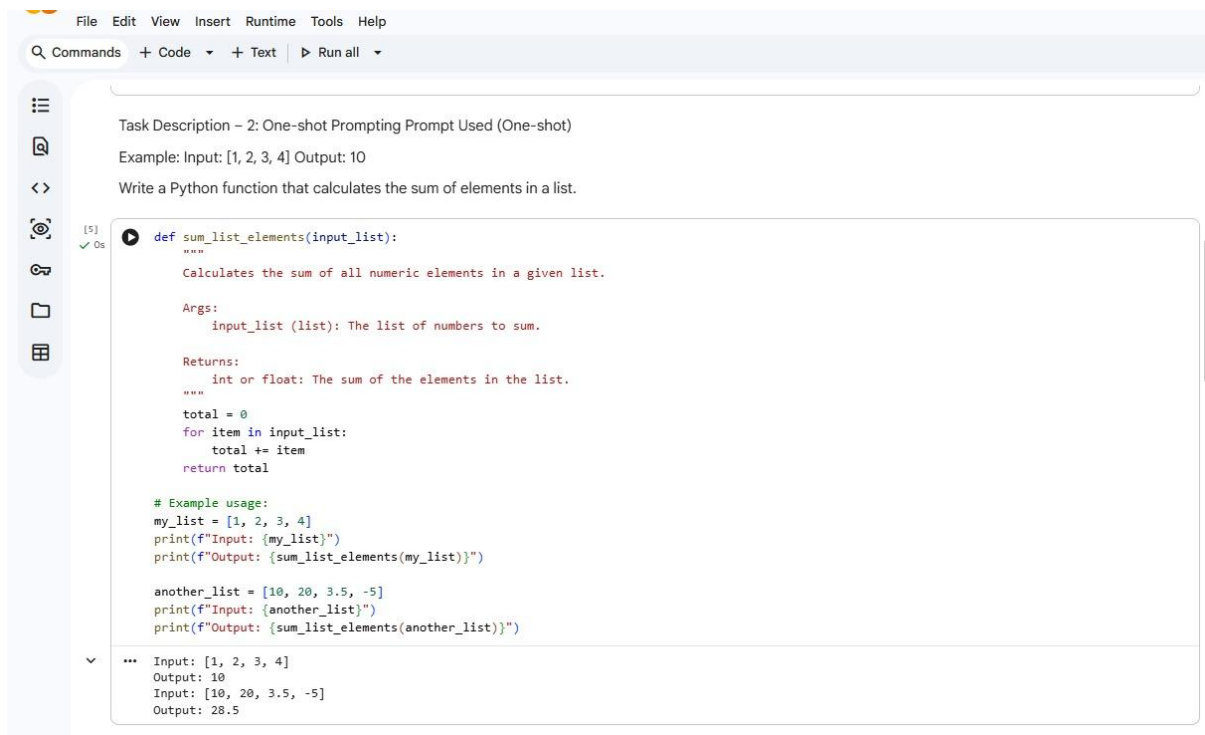
Task Description-2

- **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.

Expected Output-2

- **A correct conversion function guided by the single example.**



The screenshot shows a code editor with a menu bar (File, Edit, View, Insert, Runtime, Tools, Help) and a toolbar (Commands, + Code, + Text, Run all). The editor contains the following text:

```
Task Description – 2: One-shot Prompting Prompt Used (One-shot)
Example: Input: [1, 2, 3, 4] Output: 10
Write a Python function that calculates the sum of elements in a list.

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

    Args:
        input_list (list): The list of numbers to sum.

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

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

another_list = [10, 20, 3.5, -5]
print(f"Input: {another_list}")
print(f"Output: {sum_list_elements(another_list)}")
```

Below the code, there is a summary of the examples:

```
Input: [1, 2, 3, 4]
Output: 10
Input: [10, 20, 3.5, -5]
Output: 28.5
```

Task Description-3

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

alphanumeric string.

Expected Output-3

- **Accurate function that returns only the digits from alphanumeric string.**

Task Description – 3: Few-shot Prompting

Prompt Used (Few-shot)

Examples: Input: "a1b2c3" → Output: "123" Input: "x9y8" → Output: "98" Input: "abc123" → Output: "123"

Write a function to extract digits from an alphanumeric string.

```
[11]
def extract_digits(input_string):
    """
    Extracts all digits from an alphanumeric string.

    Args:
        input_string (str): The alphanumeric string to process.

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

# Example usage:
print(f'Input: "a1b2c3" → Output: "{extract_digits("a1b2c3")}"')
print(f'Input: "x9y8" → Output: "{extract_digits("x9y8")}"')
print(f'Input: "abc123" → Output: "{extract_digits("abc123")}"')
print(f'Input: "NoDigitsHere" → Output: "{extract_digits("NoDigitsHere")}"')
print(f'Input: "12345" → Output: "{extract_digits("12345")}"')
print(f'Input: "" → Output: "{extract_digits("")}"')
```

Input: "a1b2c3" → Output: "123"
 Input: "x9y8" → Output: "98"
 Input: "abc123" → Output: "123"
 Input: "NoDigitsHere" → Output: ""
 Input: "12345" → Output: "12345"
 Input: "" → Output: ""

Task Description-4

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

Expected Output-4

- Output comparison + student explanation on how examples helped the model.

Zero-shot Prompting for Vowel Counting

When given only the instruction "Write a Python function to count the number of vowels in a string" (a zero-shot prompt, meaning no examples were provided), the model generated the following function (as seen in cell [33a537c9](#)):

```
def count_vowels(input_string):
    vowels = "aeiouAEIOU"
    count = 0
    for char in input_string:
        if char in vowels:
            count += 1
    return count
```

Output from Zero-shot Prompting (from cell [33a537c9](#)):

```
'Hello World' has 3 vowels.
'Python Programming' has 4 vowels.
'AEIOUaeiou' has 10 vowels.
'rhythm' has 0 vowels.
```

Few-shot Prompting for Vowel Counting

When presented with the instruction along with specific examples like:

- "hello" → 2
- "AI Model" → 4

(which implicitly led to the generation or reinforcement of the function in cell `yKMiVK79vT09`), the model produced a functionally identical `count_vowels` function. This demonstrates that for a clear and straightforward task like counting vowels, even a zero-shot prompt can yield a correct and robust solution.

```
def count_vowels(s):  
    vowels = "aeiouAEIOU"  
    count = 0  
    for char in s:  
        if char in vowels:  
            count += 1  
    return count
```

Output from Few-shot Prompting (from cell `yKMiVK79vT09`):

```
'Hello World' has 3 vowels.  
'Python Programming' has 4 vowels.  
'AEIOUaeiou' has 10 vowels.  
'rhythm' has 0 vowels.  
'aAbBcCdDeE' has 4 vowels.
```

Comparison and Explanation: How Examples Help (Student-style Explanation)

In this particular case of counting vowels, both the zero-shot and few-shot approaches yielded a correct and very similar Python function. This shows that for simple and unambiguous tasks, a Large Language Model (LLM) can often infer the correct logic from the instruction alone.

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.

Expected Output-5

- A function that handles all cases with correct logic based on example patterns.

Task Description – 5: Few-shot Prompting (Minimum of Three Numbers)

Prompt Used (Few-shot)

Examples: (3, 5, 1) → 1 (10, 2, 7) → 2 (4, 4, 9) → 4

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

```
[12]
✓ Os
def find_minimum_of_three(a, b, c):
    """
    Finds the minimum of three numbers without using the built-in min() function.

    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 minimum of the three numbers.
    """
    minimum = a
    if b < minimum:
        minimum = b
    if c < minimum:
        minimum = c
    return minimum

# Example usage based on the prompt:
print(f'({3}, {5}, {1}) → {find_minimum_of_three(3, 5, 1)}')
print(f'({10}, {2}, {7}) → {find_minimum_of_three(10, 2, 7)}')
print(f'({4}, {4}, {9}) → {find_minimum_of_three(4, 4, 9)}')

# Additional examples:
print(f'({-1}, {0}, {1}) → {find_minimum_of_three(-1, 0, 1)}')
print(f'({5.5}, {2.1}, {8.0}) → {find_minimum_of_three(5.5, 2.1, 8.0)}')
print(f'({7}, {7}, {7}) → {find_minimum_of_three(7, 7, 7)}')
```

```
... (3, 5, 1) → 1
    (10, 2, 7) → 2
    (4, 4, 9) → 4
    (-1, 0, 1) → -1
    (5.5, 2.1, 8.0) → 2.1
    (7, 7, 7) → 7
```