AI Assisted Coding

Assignment – 4.2

Name: MD SHAHID

Batch: 21

HtNo: 2303A51421

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.

Code:

```python
#Write a Python function to determine whether a given number is prime
def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True

dynamic_input = int(input("Enter a number to check if it's prime: "))
if is_prime(dynamic_input):
    print(f"{dynamic_input} is a prime number.")
else:
    print(f"{dynamic_input} is not a prime number.")

```

Output:

```
Enter a number to check if it's prime: 5
5 is a prime number.
```

Final Observation:

The `is prime(n)` function determines whether a number is prime by first verifying that the input is greater than 1, since 0, 1, and negative numbers cannot be prime. It then checks for divisibility by testing integers from 2 up to $n - 1$. If any value divides the number exactly, the function immediately returns `False`, indicating the number is not prime. If no divisors are found, it returns `True`, confirming that the number is prime. Since this solution was produced using zero-shot prompting without any examples, it demonstrates that the AI can grasp the basic concept and generate a correct, though simple, implementation

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.

Code:

```
"""
generate a function that calculates the sum of elements in a list.
Input: [1, 2, 3, 4],
Output: 10
"""
def sum_of_elements(input_list):
    return sum(input_list)
# Example usage
dynamic_input_list = [int(x) for x in input("Enter numbers separated by spaces: ").split()]
result = sum_of_elements(dynamic_input_list)
print("Sum of elements:", result)
```

Output:

```
Enter numbers separated by spaces: 4 3 8 5
Sum of elements: 20
```

Final Observation:

The is_prime(n) function checks whether a number is prime by first ensuring the input is greater than 1, as 0, 1, and negative numbers are not prime. It then tests divisibility by iterating from 2 to $n - 1$. If any number divides the input exactly, the function returns False; otherwise, it returns True. Because this solution was generated using zero-shot prompting without examples, it shows that the AI can understand the core concept and produce a correct, albeit basic, implementation.

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.

Code:

```
'''Write a Python function to extract only digits from an alphanumeric string.
Examples:
Input: "a1b2c3" -> Output: "123"
Input: "abc2026xyz" -> Output: "2026"
Input: "9a8b7" -> Output: "987"'''
def extract_digits(input_string):
    return ''.join(filter(str.isdigit, input_string))
# Example usage
dynamic_input_string = input("Enter an alphanumeric string: ")
result = extract_digits(dynamic_input_string)
print("Extracted digits:", result)
```

Output:

```
Enter an alphanumeric string: aszdfs2345df,ang,564fklj
Extracted digits: 2345564
```

Final Observation:

The extract_digits(text) function iterates through each character in the input string and uses the isdigit() method to identify numeric characters. Any digit encountered is appended to the result, while non-numeric characters are ignored. After processing the entire string, the function returns a new string containing only the extracted digits. Because this solution was generated using few-shot prompting with multiple examples, the AI was able to clearly learn the pattern of filtering out letters and retaining numbers, resulting in a more accurate and reliable implementation.

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:

Code:

```python
#write a python function to count the number of vowels in a given string.
def count_vowels(input_string):
    vowels = 'aeiouAEIOU'
    count = sum(1 for char in input_string if char in vowels)
    return count
# Example usage
dynamic_input = input("Enter a string: ")
result = count_vowels(dynamic_input)
print("Number of vowels:", result)
```

Output:

```
Enter a string: python
Number of vowels: 1
```

Few Shot:

Code:

```python
'''
Write a Python function to count the number of vowels in a string.
Examples:
Input: "hello" → Output: 2
Input: "ChatGPT" → Output: 2
Input: "AEIOU" → Output: 5
'''
def count_vowels(input_string):
    vowels = 'aeiouAEIOU'
    count = sum(1 for char in input_string if char in vowels)
    return count
# Example usage
dynamic_input = input("Enter a string: ")
result = count_vowels(dynamic_input)
print("Number of vowels:", result)
```

Output:

```
Enter a string: Dhanush
Number of vowels: 2
```

Output Comparison:

Zero-shot version:

The function is simple and functional, relying on a basic loop and counter to count vowels, but the logic is straightforward and minimally optimized

Few-shot version:

The function is more concise and refined. The provided examples clarified that both uppercase and lowercase vowels must be included, resulting in a more confident, compact, and efficient implementation.

Final Observation:

With zero-shot prompting, the AI produced a basic vowel-counting function using only the given instruction, which led to a standard loop-based approach. In contrast, the few-shot prompt supplied explicit examples that defined the expected behavior and output format. These examples reduced ambiguity and guided the AI toward a cleaner, better-structured, and slightly optimized solution. This comparison highlights that adding examples improves clarity and typically leads to more accurate and polished results.

• 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.

Code:

```
'''
Write a Python function to find the minimum of three numbers without using the built-in min() function.
Examples:
Input: (3, 7, 5) ➤ Output: 3
Input: (10, 2, 8) ➤ Output: 2
Input: (-1, 4, 0) ➤ Output: -1
'''
def find_minimum(a, b, c):
    if a <= b and a <= c:
        return a
    elif b <= a and b <= c:
        return b
    else:
        return c
# Example usage
num1 = float(input("Enter first number: "))
num2 = float(input("Enter second number: "))
num3 = float(input("Enter third number: "))
minimum = find_minimum(num1, num2, num3)
print("The minimum of the three numbers is:", minimum)

```

Output:

```
Enter first number: 3
Enter second number: 7
Enter third number: 2
The minimum of the three numbers is: 2.0
```

Final Observation:

The find_minimum(a, b, c) function determines the smallest of three numbers using conditional comparisons. It first checks whether a is less than or equal to both b and c. If not, it evaluates whether b is the smallest. If neither condition is satisfied, the function returns c as the minimum. Because this implementation was generated using few-shot prompting, the provided examples clearly demonstrated the comparison pattern and expected output. As a result, the AI produced logic that reliably handles all scenarios, including negative and equal values