Assignment 3.4

2303A510A1 – Bhanu

Batch - 14

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

Task 1: Zero-shot Prompt – Fibonacci Series Generator

Prompt:

Write a Python function that takes an integer N as input and prints the first N Fibonacci numbers.

Code & Output:

```
# Write a Python function that takes an integer N as input
# and prints the first N Fibonacci numbers.
def print_fibonacci_numbers(N):
    if N <= 0:
        print("Please enter a positive integer for N.")
        return
    a, b = 0, 1
    count = 0
    if N == 1:
        print(a)
    else:
        while count < N:
            print(a, end=' ')
            nth = a + b
            a = b
            b = nth
            count += 1
        print()
num_fibonacci = int(input())
print(f"The first {num_fibonacci} Fibonacci numbers are:")
print_fibonacci_numbers(num_fibonacci)
```

```
10
The first 10 Fibonacci numbers are:
0 1 1 2 3 5 8 13 21 34
```

Explanation:

- It first checks if N is less than or equal to 0. If it is, it prints an error message "Please enter a positive integer for N." and exits, as the Fibonacci sequence is typically defined for positive integers.
- It initializes two variables, a and b, to the first two Fibonacci numbers: 0 and 1. These will be used to generate subsequent numbers in the sequence.
- A count variable is set to 0. This will track how many Fibonacci numbers have been printed.
- If N is 1, meaning only the first Fibonacci number is requested, it simply prints a (which is 0).
- For N greater than 1, it enters a while loop.
- The loop continues until count reaches N, ensuring N numbers are printed.
- It prints the current value of a (which is a Fibonacci number). end=' ' ensures that numbers are printed on the same line, separated by a space.
- Calculates the next Fibonacci number by adding the current two (a and b).
- Updates a to the value of b (shifting the sequence forward).
- Updates b to the newly calculated nth Fibonacci number.
- Increments the counter.
- After the loop finishes, this prints a newline character to ensure subsequent output starts on a new line.

Task 2: One-shot Prompt – List Reversal Function

Prompt:

Write a Python function to reverse a list.

Example:

Input: [1, 2, 3]

Output: [3, 2, 1]

Code & Output

```python
# Write a Python function to reverse a list.
# Example:
# Input: [1, 2, 3]
# Output: [3, 2, 1]

def reverse_list(input_list):
    return input_list[::-1]

# Example usage:
my_list = list(map(int,input().split()))
reversed_my_list = reverse_list(my_list)
print(f"Original list: {my_list}")
print(f"Reversed list: {reversed_my_list}")
```

```
1 2 3
Original list: [1, 2, 3]
Reversed list: [3, 2, 1]
```

Explanation:

- This is the core of the function and uses Python's slicing feature to reverse the list.
- [::-1] creates a reversed copy of the input_list. The first colon means to start from the beginning, the second colon means to go until the end, and -1 indicates a step of -1, effectively reversing the order of elements. This method creates a new list with the elements in reverse order, leaving the original list unchanged.

Task 3: Few-shot Prompt – String Pattern Matching

Prompt:

Write a function is_valid(s) that returns True if:

- The string starts with a capital letter

- The string ends with a period

Examples:

"Hello." → True

"hello." → False

"Hello" → False

Code & Output:

```
#Write a function is_valid(s) that returns True if:
#- The string starts with a capital letter
#- The string ends with a period
#Examples:
#"Hello." → True
#"hello." → False
#"Hello" → False
def is_valid(s):
    if not s:
        return False  # Handle empty string case
    starts_with_capital = s[0].isupper()
    ends_with_period = s.endswith('.')
    return starts_with_capital and ends_with_period
# Examples:
s=input()
print(f"{is_valid(s)}")
```

```
Hello.
True
```

Explanation:

- This line first checks if the input string s is empty. If s is an empty string, it immediately returns False because an empty string cannot start with a capital letter or end with a period.
- This line checks the first character of the string (s[0]). The .isupper() method returns True if the character is an uppercase letter, and False otherwise. The result is stored in the starts_with_capital boolean variable.
- This line uses the .endswith('.') string method to check if the string s ends with a period character. It returns True if it does, and False otherwise. The result is stored in the ends_with_period boolean variable.
- Finally, the function returns the logical AND of the two boolean variables. This means True is returned only if both conditions (starts with a capital and ends with a period) are True; otherwise, False is returned.

Task 4: Zero-shot vs Few-shot – Email Validator

Prompt:

Write a Python function to validate an email address.

Code & Output:

```python
#Write a Python function to validate an email address.
import re
def is_valid_email(email_address):
    email_regex = re.compile(r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$')

    if email_regex.match(email_address):
        return True
    else:
        return False

# Examples:
email_address = input()
print(f"{is_valid_email(email_address)}")
```

```
bhanu@gmail.com
True
```

Explanation:

- This just brings in Python's tool for working with these patterns.
- This creates our pattern matcher with a specific set of rules for emails.
- This line tries to see if the email_address perfectly fits the email_regex pattern from the beginning to the end.
- If it fits the pattern, the function says True (it's valid); otherwise, it says False (it's not valid).

Task 5: Prompt Tuning – Summing Digits of a Number

Style 1: Generic task prompt

Prompt:

Write a Python function that returns the sum of digits of a number.

Code & Output:

```python
# Write a Python function that returns the sum of digits of a number.
def sum_digits(number):
    s = 0
    while number > 0:
        s += number % 10
        number //= 10
    return s
num = int(input())
result = sum_digits(num)
print(f"The sum of the digits of {num} is: {result}")
```

```
98765
The sum of the digits of 98765 is: 35
```

Explanation:

- A variable s is initialized to 0. This variable will accumulate the sum of the digits.
- This loop continues as long as the number is greater than 0. Each iteration processes one digit.
- The modulo operator (%) gives the remainder when number is divided by 10. This effectively extracts the last digit of the number (e.g., 123 % 10 is 3). This extracted digit is then added to s.
- The floor division operator (//) divides number by 10 and rounds down to the nearest whole number. This effectively removes the last digit from the number (e.g., 123 // 10 is 12).
- Once the while loop finishes (meaning number has become 0 and all digits have been processed), the function returns the final sum stored in s.

Style 2: Task + Input/Output example

Prompt:

Write a Python function that returns the sum of digits of a number.

Example:

Input: 123

Output: 6

Code & Output:

```python
# Write a Python function that returns the sum of digits of a number.
# Example:
# Input: 123
# Output: 6
def sum_digits(number):
    s = 0
    str_num = str(abs(number))
    for digit in str_num:
        s += int(digit)
    return s

# Example:
input_number = 123
output_sum = sum_digits(input_number)
print(f"Input: {input_number}")
print(f"Output: {output_sum}")

# Another example:
input_number_2 = int(input("Enter a number: "))
output_sum_2 = sum_digits(input_number_2)
print(f"Input: {input_number_2}")
print(f"Output: {output_sum_2}")
```

```
Input: 123
Output: 6
Enter a number: 98765
Input: 98765
Output: 35
```

Explanation:

- Start with a total of zero. We need a place to keep track of the sum, so we begin with s = 0.
- Turn the number into text. Imagine you have the number 123. The easiest way to get each digit separately is to treat it like a word: "1", "2", "3". That's what str(abs(number)) does. abs() just makes sure it works if you accidentally give it a negative number, by making it positive first.
- Go through each character. The code then looks at each character in that "text" one by one.
- Convert and add. For each character ("1", "2", "3"), it turns it back into a number (an integer) and adds it to our running total s.
- Give back the total. Once it has gone through all the characters, it returns the final sum s.