

LAB ASSIGNMENT-4.2


NAME:SATHWIK


H.NO:2403A51273

BATCH:12

Task-1:

- **Zero-shot:**
- Prompt AI with only the instruction — Write a Python function to generate the Fibonacci sequence up to n terms

```
 def fibonacci_sequence(n):  
    """Generates the Fibonacci sequence up to n terms."""  
    sequence = []  
    a, b = 0, 1  
    for _ in range(n):  
        sequence.append(a)  
        a, b = b, a + b  
    return sequence  
  
# Example usage:  
print(fibonacci_sequence(10))
```

```
 [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

•

Explanation:

- **def fibonacci_sequence(n):**: This line defines the function named fibonacci_sequence that takes one argument, n, which represents the number of terms in the sequence you want to generate.
- **"""Generates the Fibonacci sequence up to n terms."""**: This is a docstring, which explains what the function does.

- **sequence = []:** This initializes an empty list called sequence. This list will store the generated Fibonacci numbers.
- **a, b = 0, 1:** This initializes two variables, a and b, with the first two numbers of the Fibonacci sequence, which are 0 and 1.
- **for _ in range(n):** This starts a for loop that will iterate n times. The _ is used as a variable name when you don't need to use the loop counter within the loop.
- **sequence.append(a):** In each iteration, the current value of a (which represents the next Fibonacci number) is added to the sequence list.
- **a, b = b, a + b:** This is the core of the Fibonacci logic. It updates the values of a and b for the next iteration. The new a becomes the current b, and the new b becomes the sum of the current a and b.
- **return sequence:** After the loop finishes, the function returns the sequence list containing the generated Fibonacci numbers.

Task-2:

- **One-shot:**
- Provide one example: Input: 100, Output: 37.78 to help AI generate a function that converts Fahrenheit to Celsius

```
def fahrenheit_to_celsius(fahrenheit):
    """Converts Fahrenheit to Celsius."""
    celsius = (fahrenheit - 32) * 5/9
    return celsius

# Example usage based on the one-shot prompt:
# Input: 100, Output: 37.78
input_fahrenheit = 100
expected_celsius = 37.78
calculated_celsius = fahrenheit_to_celsius(input_fahrenheit)

print(f"Input Fahrenheit: {input_fahrenheit}")
print(f"Calculated Celsius: {calculated_celsius}")
print(f"Expected Celsius: {expected_celsius}")

# You can add an assertion to check if the output is close to the expected value
# assert abs(calculated_celsius - expected_celsius) < 0.01, "Calculated Celsius is not close to the expected value"
```

Input Fahrenheit: 100
Calculated Celsius: 37.77777777777778
Expected Celsius: 37.78

Explanation:

- **def fahrenheit_to_celsius(fahrenheit):**: This line defines a function named `fahrenheit_to_celsius` that takes one argument, `fahrenheit`, which is the temperature in degrees Fahrenheit that you want to convert.
- **"""Converts temperature from Fahrenheit to Celsius."""**: This is a docstring explaining what the function does.
- **celsius = (fahrenheit - 32) * 5/9**: This is the core of the conversion. It applies the standard formula to convert Fahrenheit to Celsius: subtract 32 from the Fahrenheit temperature and then multiply the result by 5/9. The result is stored in the `celsius` variable.
- **return celsius**: The function returns the calculated Celsius temperature.

Task-3:

- **Prompt: Few-shot:**
- Give 2–3 examples to create a function that extracts the domain name from an email address.
-
-

```

def extract_domain(email):
    """Extracts the domain name from an email address."""
    try:
        domain = email.split('@')[1]
        return domain
    except IndexError:
        return "Invalid email format"

# Example usage 1
email1 = "abc@gmail.com"
domain1 = extract_domain(email1)
print(f"The domain for {email1} is: {domain1}")

# Example usage 2
email2 = "user@example.org"
domain2 = extract_domain(email2)
print(f"The domain for {email2} is: {domain2}")

# Example usage 3
email3 = "another.user@sub.domain.co.uk"
domain3 = extract_domain(email3)
print(f"The domain for {email3} is: {domain3}")

```

The domain for [abc@gmail.com](#) is: gmail.com
The domain for [user@example.org](#) is: example.org
The domain for [another.user@sub.domain.co.uk](#) is: sub.domain.co.uk

Explanation:

- **def extract_domain(email)::** This line defines a function named `extract_domain` that takes one argument, `email`, which is the email address as a string.
- **"""Extracts the domain name from an email address.""":** This is a docstring explaining the function's purpose.
- **try::** This starts a try block, which is used for error handling. Code within this block is attempted, and if an error occurs, the code in the except block is executed.

- **domain = email.split('@')[1]:** This is the core logic for extracting the domain.
 - `email.split('@')` splits the email string into a list of substrings using the "@" symbol as the delimiter. For example, `"abc@gmail.com".split('@')` would result in `['abc', 'gmail.com']`.
 - `[1]` accesses the element at index 1 of the resulting list, which is the part after the "@" symbol (the domain name).
 - The extracted domain is stored in the domain variable.
- **return domain:** If the split and indexing are successful, the function returns the extracted domain.
- **except IndexError::** This is the start of the except block, which catches an `IndexError`. An `IndexError` will occur if the `split('@')` operation does not produce a list with at least two elements (i.e., if there is no "@" symbol in the email address).
- **return "Invalid email format":** If an `IndexError` occurs (meaning the email format is invalid because it lacks an "@" symbol), the function returns the string "Invalid email format"

Task-4

Prompt:

Compare zero-shot vs few-shot prompting for generating a function that checks whether a word is a palindrome, ignoring punctuation and case

Zero-shot:

```
✓ [6] import re
0s

def is_palindrome_zero_shot(word):
    """Checks if a word is a palindrome, ignoring punctuation and case."""
    # Remove punctuation and convert to lowercase
    cleaned_word = re.sub(r'^\w', '', word).lower()
    # Check if the cleaned word is equal to its reverse
    return cleaned_word == cleaned_word[::-1]

# Example usage (you can add examples here to test the function)
print(is_palindrome_zero_shot("Racecar"))
print(is_palindrome_zero_shot("A man, a plan, a canal: Panama"))
print(is_palindrome_zero_shot("hello"))
```

⇒ True
True
False

Few-shot:

```
import re

# Few-shot examples:
# Input: "Racecar", Output: True
# Input: "A man, a plan, a canal: Panama", Output: True
# Input: "hello", Output: False

def is_palindrome_few_shot(word):
    """Checks if a word is a palindrome, ignoring punctuation and case."""
    # Remove punctuation and convert to lowercase
    cleaned_word = re.sub(r'^\w', '', word).lower()
    # Check if the cleaned word is equal to its reverse
    return cleaned_word == cleaned_word[::-1]

# Example usage
print(is_palindrome_few_shot("Racecar"))
print(is_palindrome_few_shot("A man, a plan, a canal: Panama"))
print(is_palindrome_few_shot("hello"))
```

⇒ True
True
False

P few-shot prompting for generating a palindrome, ignoring punctuation and case.

✦ Certainly. I can generate a Python

Comparision:

Feature	Zero-Shot Prompting	Few-Shot Prompting
Examples Provided	No examples of input/output are given.	A few examples of input/output are given to guide the model.
Guidance Level	Relies solely on the model's pretraining and understanding of the task description.	Provides explicit examples to demonstrate the desired behavior and output format.
Complexity of Task	More suitable for simpler, well-defined tasks where the expected output is unambiguous.	Can be more effective for complex or nuanced tasks where examples help clarify the requirements.
Outcome (in this case)	Generated a correct function for checking palindromes, ignoring punctuation and case.	Generated the same correct function for checking palindromes, ignoring punctuation and case, as the task was relatively simple.
Code Generated	is_palindrome_zero_shot function	is_palindrome_few_shot function (identical code to zero-shot in this case)

Explanation:

- The two functions to check for palindromes, one using a zero-shot prompt (is_palindrome_zero_shot) and one using a few-shot prompt (is_palindrome_few_shot).
- In this particular case, the code generated for both functions and examples is the same

Task-5:

- **Prompt:** Use few-shot prompting with 3 sample inputs to generate a function that determines the maximum of three numbers without using the built-in `max()` function.

```

# Few-shot examples:
# Input: 5, 10, 3, Output: 10
# Input: -1, -5, 0, Output: 0
# Input: 7, 7, 7, Output: 7

def find_maximum(a, b, c):
    """Determines the maximum of three numbers without using the built-in max() function."""
    maximum = a
    if b > maximum:
        maximum = b
    if c > maximum:
        maximum = c
    return maximum

# Example usage 1
num1, num2, num3 = 5, 10, 3
max_num = find_maximum(num1, num2, num3)
print(f"The maximum of {num1}, {num2}, and {num3} is: {max_num}")

# Example usage 2
num4, num5, num6 = -1, -5, 0
max_num2 = find_maximum(num4, num5, num6)
print(f"The maximum of {num4}, {num5}, and {num6} is: {max_num2}")

# Example usage 3
num7, num8, num9 = 7, 7, 7
max_num3 = find_maximum(num7, num8, num9)
print(f"The maximum of {num7}, {num8}, and {num9} is: {max_num3}")

The maximum of 5, 10, and 3 is: 10
The maximum of -1, -5, and 0 is: 0
The maximum of 7, 7, and 7 is: 7

```

Explanation:

- **`def find_maximum(a, b, c):`**: This line defines a function named `find_maximum` that takes three arguments: `a`, `b`, and `c`, which are the three numbers you want to compare.
- **`"""Determines the maximum of three numbers without using the built-in max() function."""`**: This is a docstring explaining the function's purpose.
- **`maximum = a`**: This line initializes a variable called `maximum` and assumes that the first number `a` is the maximum.

- **if b > maximum::** This is a conditional statement. It checks if the second number b is greater than the current value of maximum.
- **maximum = b:** If b is indeed greater than maximum, the value of maximum is updated to b.
- **if c > maximum::** This is another conditional statement. It checks if the third number c is greater than the current value of maximum.
- **maximum = c:** If c is greater than the current value of maximum, the value of maximum is updated to c.
- **return maximum:** After checking both b and c against the current maximum, the function returns the final value of maximum, which will be the largest of the three input numbers.