

## LAB ASSSIGMENT-4.2

NAME : KURAPATI AKSHITHA

BRANCH : CSE/AIML

ENROLL NO : 2403A52021

COURSE : AI ASSISTED CODING

DATE : 26-08-2025

BATCH : 02

### TASK DESCRIPTION #1

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

#### **PROMPT:**

Develop a code to generate Zero-shot: Prompt AI with only the instruction – Write a Python function to generate the Fibonacci sequence up to n terms with user input.

#### **EXPECTED OUTPUT:**

A working function without using any sample input/output

```
def fibonacci_sequence(n):  
    """  
    Generates the Fibonacci sequence up to n terms.  
  
    Args:  
        n: The number of terms in the sequence.  
  
    Returns:  
        A list containing the Fibonacci sequence up to n terms.  
    """  
    if n <= 0:  
        return []  
    elif n == 1:  
        return [0]  
    else:  
        sequence = [0, 1]  
        while len(sequence) < n:  
            next_term = sequence[-1] + sequence[-2]  
            sequence.append(next_term)  
        return sequence  
  
# Example usage:  
n_terms = 10  
fib_sequence = fibonacci_sequence(n_terms)  
print(fib_sequence)
```

→ [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]

## EXPLANATION:

- `def fibonacci_sequence(n):` : This line defines the function `fibonacci_sequence` that takes one argument, `n`, which represents the desired number of terms in the sequence.
- `if n <= 0:` : This checks if the input `n` is less than or equal to 0. If it is, an empty list `[]` is returned, as the Fibonacci sequence is typically defined for positive integers.
- `elif n == 1:` : If `n` is not less than or equal to 0, this checks if `n` is equal to 1. If it is, the function returns a list containing only the first term of the Fibonacci sequence, which is `[0]`.
- `else:` : If `n` is greater than 1, the code enters this block.
- `sequence = [0, 1]` : A list called `sequence` is initialized with the first two terms of the Fibonacci sequence, 0 and 1.
- `while len(sequence) < n:` : This loop continues as long as the length of the `sequence` list is less than the desired number of terms `n`.
- `next_term = sequence[-1] + sequence[-2]` : Inside the loop, the next term in the sequence is calculated by adding the last two terms in the `sequence` list (`sequence[-1]` and `sequence[-2]`).
- `sequence.append(next_term)` : The calculated `next_term` is added to the end of the `sequence` list.
- `return sequence` : After the loop finishes (when the `sequence` list

What can I help you build?



## TASK DESCRIPTION#2

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

## PROMPT:

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

## EXPECTED OUTPUT:

A correct conversion function guided by the single example



```
def fahrenheit_to_celsius(fahrenheit):  
    """  
    Converts Fahrenheit to Celsius.  
  
    Args:  
        fahrenheit: The temperature in Fahrenheit.  
  
    Returns:  
        The temperature in Celsius.  
    """  
    celsius = (fahrenheit - 32) * 5/9  
    return celsius  
  
# Example usage:  
fahrenheit_temp = 100  
celsius_temp = fahrenheit_to_celsius(fahrenheit_temp)  
print(f"{fahrenheit_temp} Fahrenheit is equal to {celsius_temp:.2f} Celsius")
```

100 Fahrenheit is equal to 37.78 Celsius

## EXPLANATION:

- `def fahrenheit_to_celsius(fahrenheit):`: This defines the function named `fahrenheit_to_celsius` which takes one argument, `fahrenheit`, representing the temperature in Fahrenheit.
- `"""Docstring"""`: This is a docstring explaining what the function does, its arguments, and what it returns.
- `celsius = (fahrenheit - 32) * 5/9`: This is the core of the conversion. It subtracts 32 from the Fahrenheit temperature and then multiplies the result by 5/9 to get the equivalent Celsius temperature.
- `return celsius`: The function returns the calculated Celsius temperature.
- `fahrenheit_temp = 100`: This line sets a variable `fahrenheit_temp` to 100, which is the temperature we want to convert.
- `celsius_temp = fahrenheit_to_celsius(fahrenheit_temp)`: This line calls the `fahrenheit_to_celsius` function with `fahrenheit_temp` (which is 100) and stores the returned Celsius temperature in the variable `celsius_temp`.
- `print(f"{fahrenheit_temp} Fahrenheit is equal to {celsius_temp:.2f} Celsius")`: This line prints the original Fahrenheit temperature and the converted Celsius temperature. The `:.2f` formats the Celsius temperature to two decimal places.

## TASK DESCRIPTION#3

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

PROMPT:

Generate a code that develop Few-shot: Give 2–3 examples to create a function that extracts the domain name from an email address

## EXPECTED OUTPUT:

Accurate function that returns only the domain portion of an email (e.g.@gmail.com)

```
def extract_domain(email):
    """
    Extracts the domain name from an email address.

    Args:
        email: The email address as a string.

    Returns:
        The domain name as a string, or None if the email format is invalid.
    """
    if "@" in email:
        return email.split("@")[-1]
    else:
        return None

# Example usage:
email1 = "testuser@example.com"
email2 = "another.user@sub.domain.org"
email3 = "invalid-email"

domain1 = extract_domain(email1)
domain2 = extract_domain(email2)
domain3 = extract_domain(email3)

print(f"Domain for '{email1}': {domain1}")
print(f"Domain for '{email2}': {domain2}")
print(f"Domain for '{email3}': {domain3}")
```

Domain for 'testuser@example.com': example.com  
Domain for 'another.user@sub.domain.org': sub.domain.org  
Domain for 'invalid-email': None

## EXPLANATION:

- `def extract_domain(email):` : This line defines the function `extract_domain` that takes one argument, `email`, which is expected to be a string representing an email address.
- `"""Docstring"""` : This is a docstring that explains the function's purpose, arguments, and return value.
- `if "@" in email:` : This checks if the "@" symbol is present in the `email` string. A valid email address should contain an "@" symbol separating the local part from the domain.
- `return email.split("@")[-1]` : If the "@" symbol is found, the code splits the `email` string into a list of substrings using "@" as the delimiter. `email.split("@")` would produce a list like `['testuser', 'example.com']` for the email "[redacted link]". `[-1]` then accesses the last element of this list, which is the domain name. This domain name is returned by the function.
- `else:` : If the "@" symbol is not found in the `email` string, it's considered an invalid email format for this function's purpose.
- `return None` : In the case of an invalid format (no "@" symbol), the function returns `None`.
- `email1 = "testuser@example.com", email2 = "another.user@sub.domain.org", email3 = "invalid-email":`

## TASK DESCRIPTION#4

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

PROMPT:

Generate a code that develops Compare zero-shot vs few-shot prompting for generating a function that

checks whether a word is a palindrome, ignoring punctuation and case

## Expected Output#4

• Output comparison + student explanation on how examples helped the model

```
import re

def is_palindrome_zero_shot(word):
    """
    Checks if a word is a palindrome, ignoring punctuation and case (Zero-shot).

    Args:
        word: The input string.

    Returns:
        True if the word is a palindrome, False otherwise.
    """
    # 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 (will be expanded later for comprehensive testing)
print(f"'Racecar' is a palindrome: {is_palindrome_zero_shot('Racecar')}")
print(f"'Hello' is a palindrome: {is_palindrome_zero_shot('Hello')}")
print(f"'A man, a plan, a canal: Panama.' is a palindrome: {is_palindrome_zero_shot('A man, a plan, a canal: Panama.')}")
```

```
'Racecar' is a palindrome: True
'Hello' is a palindrome: False
'A man, a plan, a canal: Panama.' is a palindrome: True
```

EXPLANATION:



- 
- `import re`: This line imports the `re` module, which provides regular expression operations. This module is used here to easily remove punctuation.
  - `def is_palindrome_zero_shot(word):`: This defines the function `is_palindrome_zero_shot` that takes one argument, `word`, which is the string to be checked.
  - `"""Docstring"""`: This is a docstring explaining the function's purpose, arguments, and return value. It also notes that this function was generated with a zero-shot prompt.
  - `cleaned_word = re.sub(r'^\w', '', word).lower()`: This is a crucial line for cleaning the input string:
    - `re.sub(r'^\w', '', word)`: This uses a regular expression to substitute any character that is NOT a word character (`\w` matches alphanumeric characters and underscore) with an empty string (`''`). This effectively removes punctuation and spaces.
    - `.lower()`: This converts the resulting string to lowercase, ensuring that the comparison is case-insensitive.
  - `return cleaned_word == cleaned_word[::-1]`: This line performs the palindrome check:

## TASK DESCRIPTION#5

- 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

PROMPT:

Develop a code to generate 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

### Expected Output#5

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

```
def find_maximum_few_shot(a, b, c):  
    """  
    Finds the maximum of three numbers without using the built-in max() function (Few-shot).  
  
    Args:  
        a: The first number.  
        b: The second number.  
        c: The third number.  
  
    Returns:  
        The maximum of the three numbers.  
    """  
    # Example 1: Input: 5, 2, 8 Output: 8  
    # Example 2: Input: 10, 10, 10 Output: 10  
    # Example 3: Input: 3, 7, 1 Output: 7  
  
    maximum = a  
    if b > maximum:  
        maximum = b  
    if c > maximum:  
        maximum = c  
    return maximum  
  
# Example usage:  
print(f"Maximum of 5, 2, 8: {find_maximum_few_shot(5, 2, 8)}")  
print(f"Maximum of 10, 10, 10: {find_maximum_few_shot(10, 10, 10)}")  
print(f"Maximum of 3, 7, 1: {find_maximum_few_shot(3, 7, 1)}")
```

Maximum of 5, 2, 8: 8  
Maximum of 10, 10, 10: 10  
Maximum of 3, 7, 1: 7

### EXPLANATION:

- `def find_maximum_few_shot(a, b, c):`: This defines the function named `find_maximum_few_shot` that accepts three arguments: `a`, `b`, and `c`.
- `"""Docstring"""`: This is a docstring explaining the function's purpose, arguments, and return value. It also indicates that this function was developed with a few-shot prompt.
- `# Example 1: Input: 5, 2, 8 Output: 8, # Example 2: Input: 10, 10, 10 Output: 10, # Example 3: Input: 3, 7, 1 Output: 7`: These lines are comments providing the few-shot examples that were used to guide the generation of this function.
- `maximum = a`: This line initializes a variable `maximum` with the value of the first input number, `a`. This assumes `a` is the maximum until proven otherwise.
- `if b > maximum:`: This checks if the second number, `b`, is greater than the current `maximum`.
- `maximum = b`: If `b` is indeed greater than `maximum`, the value of `maximum` is updated to `b`.
- `if c > maximum:`: This checks if the third number, `c`, is greater than the current `maximum`.
- `maximum = c`: If `c` is greater than the current `maximum`, the value of `maximum` is updated to `c`.