

## LAB ASSSIGMENT-4.2

NAME : KURAPATI ASHRITHA  
BRANCH : CSE/AIML  
ENROLL NO : 2403A52023  
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

#### EXPECTED OUTPUT:

A working function without using any sample inputs/outputS

## CODE:

```
def generate_fibonacci(n_terms):  
    """  
    Generates the Fibonacci sequence up to n terms.  
  
    Args:  
        n_terms: The number of terms in the Fibonacci sequence.  
  
    Returns:  
        A list containing the Fibonacci sequence.  
    """  
    if n_terms <= 0:  
        return []  
    elif n_terms == 1:  
        return [0]  
    else:  
        fib_sequence = [0, 1]  
        while len(fib_sequence) < n_terms:  
            next_term = fib_sequence[-1] + fib_sequence[-2]  
            fib_sequence.append(next_term)  
        return fib_sequence  
  
# Get the number of terms from user input  
try:  
    num_terms = int(input("Enter the number of terms for the Fibonacci sequence: "))  
    if num_terms < 0:  
        print("Please enter a non-negative integer.")  
    else:  
        fib_result = generate_fibonacci(num_terms)  
        print("Fibonacci sequence:", fib_result)  
except ValueError:  
    print("Invalid input. Please enter an integer.")
```

## OUTPUT:

```
Enter the number of terms for the Fibonacci sequence: 12  
Fibonacci sequence: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

## EXPLANATION:

Here's a breakdown:

- `def generate_fibonacci(n_terms):` : This line defines a function named `generate_fibonacci` that takes one argument, `n_terms`, which represents the desired number of terms in the sequence.
- `if n_terms <= 0:` : This checks if the input `n_terms` is less than or equal to 0. If it is, an empty list `[]` is returned because a Fibonacci sequence needs at least one term.
- `elif n_terms == 1:` : This checks if `n_terms` is 1. If it is, a list containing just `[0]` is returned, as the first term of the Fibonacci sequence is 0.
- `else:` : This block is executed if `n_terms` is greater than 1.
  - `fib_sequence = [0, 1]` : Initializes a list called `fib_sequence` with the first two terms of the Fibonacci sequence (0 and 1).
  - `while len(fib_sequence) < n_terms:` : This loop continues as long as the number of terms in `fib_sequence` is less than the desired `n_terms`.
  - `next_term = fib_sequence[-1] + fib_sequence[-2]` : Calculates the next term in the sequence by adding the last two terms in the `fib_sequence` list.
  - `fib_sequence.append(next_term)` : Appends the newly calculated `next_term` to the `fib_sequence` list.
  - `return fib_sequence` : After the loop finishes, the function returns the complete Fibonacci sequence as a list.
- The code then prompts the user to enter the number of terms, handles potential `ValueError` if the input is not an integer, and calls the `generate_fibonacci` function to print the resulting sequence.

## 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

## CODE:

```
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  
  
# Get input from the user  
try:  
    f_temp = float(input("Enter temperature in Fahrenheit: "))  
    c_temp = fahrenheit_to_celsius(f_temp)  
    print(f"{f_temp}°F is equal to {c_temp:.2f}°C")  
except ValueError:  
    print("Invalid input. Please enter a number.")
```

## OUTPUT:

```
➞ Enter temperature in Fahrenheit: 200  
200.0°F is equal to 93.33°C
```

## EXPLANATION:

```
<> def fahrenheit_to_celsius(fahrenheit):
```

Absolutely! This code defines a function to convert temperatures from Fahrenheit to Celsius.

Here's a breakdown:

- **def fahrenheit\_to\_celsius(fahrenheit):**: This defines a function named `fahrenheit_to_celsius` that takes one argument, `fahrenheit`, which is the temperature in Fahrenheit.
- **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 by 5/9. The result is stored in the `celsius` variable.
- **return celsius**: The function returns the calculated Celsius temperature.
- The code then prompts the user to enter a temperature in Fahrenheit, converts the input to a floating-point number, calls the `fahrenheit_to_celsius` function to perform the conversion, and finally prints the original Fahrenheit temperature and the calculated Celsius temperature, formatted to two decimal places. It also includes error handling for invalid input.



## 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 developsFew-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)

## CODE:

```
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.
    """
    try:
        domain = "@" + email.split('@')[1]
        return domain
    except IndexError:
        return "Invalid email format"

# Get input from the user
email_address = input("Enter an email address: ")
domain_name = extract_domain(email_address)

if domain_name:
    print(f"The domain name is: {domain_name}")
else:
    print("Could not extract domain name.")
```

## OUTPUT:

```
➞ Enter an email address: 2403a52033@sru.edu.in
The domain name is: @sru.edu.in
```

## EXPLANATION:

- `def extract_domain(email):`: This defines a function named `extract_domain` that takes one argument, `email`, which is the email address as a string.
- `try:`: This starts a `try` block, which is used for error handling. It attempts to execute the code within this block.
  - `domain = "@" + email.split('@')[1]`: This is the main logic for extracting the domain.
    - `email.split('@')`: This splits the email string into a list of substrings using the '@' symbol as the delimiter. For example, "[redacted link]" would become `['ashritha', 'gmail.com']`.
    - `[1]`: This accesses the second element of the list (index 1), which should be the domain part of the email address.
    - `"@" + ...`: This prepends the '@' symbol back to the extracted domain name.
    - The result is stored in the `domain` variable.
  - `return domain`: If the extraction is successful, the function returns the `domain` name.
- `except IndexError:`: This is the error handling part. If an `IndexError` occurs (which would happen if `email.split('@')` doesn't produce a list with at least two elements, meaning the email format is invalid and doesn't contain an '@' symbol), the code inside this block is executed.
  - `return "Invalid email format"`: If an `IndexError` occurs, the function returns the string "Invalid email format".
- The code then prompts the user to enter an email address, calls the `extract_domain` function, and prints the extracted domain name or an error message if the format was invalid.

## 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

CODE(ZERO-SHOT):

```
import re

def is_palindrome(word):
    """
    Checks if a word is a palindrome, ignoring punctuation and case.

    Args:
        word: The word to check as a string.

    Returns:
        True if the word is a palindrome, False otherwise.
    """
    # Remove punctuation and convert to lowercase
    cleaned_word = re.sub(r'^\w\s', '', word).lower()
    # Remove spaces
    cleaned_word = cleaned_word.replace(" ", "")
    # Check if the cleaned word is equal to its reverse
    return cleaned_word == cleaned_word[::-1]

# Get input from the user
word_input = input("Enter a word to check if it's a palindrome: ")

if is_palindrome(word_input):
    print(f"'{word_input}' is a palindrome.")
else:
    print(f"'{word_input}' is not a palindrome.")
```

OUTPUT:

```
⇒ Enter a word to check if it's a palindrome: madam
'madam' is a palindrome.
```

## CODE(FEW-SHOT):

```
def is_palindrome_v2(word):  
    """  
    Checks if a word is a palindrome, ignoring punctuation and case.  
  
    Args:  
        word: The word to check as a string.  
  
    Returns:  
        True if the word is a palindrome, False otherwise.  
    """  
    # Step 1: Remove punctuation and convert to lowercase  
    # re.sub() is used to replace characters matching the pattern with an empty string.  
    # r'^\w\s' matches any character that is NOT a word character (\w) or a whitespace character (\s).  
    cleaned_word = re.sub(r'^\w\s', '', word).lower()  
  
    # Step 2: Remove spaces  
    cleaned_word = cleaned_word.replace(" ", "")  
  
    # Step 3: Check if the cleaned word is equal to its reverse  
    # slicing[::-1] creates a reversed version of the string.  
    return cleaned_word == cleaned_word[::-1]  
  
# Get input from the user  
word_input_v2 = input("Enter a word to check if it's a palindrome (different code version): ")  
  
# Check and print the result  
if is_palindrome_v2(word_input_v2):  
    print(f"'{word_input_v2}' is a palindrome.")  
else:  
    print(f"'{word_input_v2}' is not a palindrome.")
```

Toggle Gemini

## OUTPUT:

```
Enter a word to check if it's a palindrome (different code version): hello  
'hello' is not a palindrome.
```

## EXPLANATION:

## ZERO-SHOT:



- `import re`: This line imports the `re` module, which provides regular expression operations. This module is used here to help clean the input string.
- `def is_palindrome(word):`: This defines a function named `is_palindrome` that takes one argument, `word`, which is the string to be checked.
- `cleaned_word = re.sub(r'^\w\s', '', word).lower()`: This line cleans the input word.
  - `re.sub(r'^\w\s', '', word)`: This uses a regular expression substitution. `r'^\w\s'` is the pattern that matches any character that is *not* a word character (`\w`) or a whitespace character (`\s`). These matched characters are replaced with an empty string (`''`), effectively removing punctuation.
  - `.lower()`: This converts the resulting string to lowercase, ensuring that the check is case-insensitive.
  - The cleaned string is stored in the `cleaned_word` variable.
- `cleaned_word = cleaned_word.replace(" ", "")`: This line removes any spaces from the `cleaned_word`.
- `return cleaned_word == cleaned_word[::-1]`: This is the core palindrome check.
  - `cleaned_word[::-1]` creates a reversed version of the `cleaned_word` string using slicing.
  - The code then compares the original `cleaned_word` with its reversed version. If they are equal, the function returns `True` (indicating it's a palindrome); otherwise, it returns `False`.
- The code then prompts the user to enter a word or phrase, calls the `is_palindrome` function, and prints whether the input is a palindrome or not.

## FEW-SHOT:

- `import re`: This line imports the `re` module for regular expression operations.
- `def is_palindrome_v2(word):`: This defines a function named `is_palindrome_v2` that takes one argument, `word`, which is the string to be checked. The `_v2` in the name indicates it's a second version or alternative implementation.
- `cleaned_word = re.sub(r'^\w\s', '', word).lower()`: This step is identical to the previous palindrome function. It uses `re.sub` to remove any characters that are not word characters or whitespace, and then converts the result to lowercase.
- `cleaned_word = cleaned_word.replace(" ", "")`: This step is also identical to the previous function. It removes any spaces from the `cleaned_word`.
- `return cleaned_word == cleaned_word[::-1]`: This is the same core palindrome check as before. It compares the `cleaned_word` with its reversed version (`cleaned_word[::-1]`) and returns `True` if they are the same, and `False` otherwise.
- The code then prompts the user for input using a different message ("Enter a word to check if it's a palindrome (different code version): "), calls the `is_palindrome_v2` function, and prints the result.

In essence, `is_palindrome_v2` is functionally the same as `is_palindrome`, just presented as a separate version.



## STUDENT EXPLANATION:

The first code (in cell 35893f4a) was generated based on a more general request ("zero-shot"). I provided a standard implementation.

The second code (in cell 201b2990) was generated after you provided examples ("few-shot"). While the core logic is the same, I included more detailed comments to explain the steps, which can be helpful when working with examples.

Both codes are functionally the same for checking palindromes while ignoring punctuation and case. The difference lies in the level of detail and explanation provided in the code itself, influenced by the presence of examples in the prompt.

## 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

### CODE:

```

def find_maximum_of_three(num1, num2, num3):
    """
    Determines the maximum of three numbers without using the built-in max() function.

    Args:
        num1: The first number.
        num2: The second number.
        num3: The third number.

    Returns:
        The maximum of the three numbers.
    """
    maximum = num1
    if num2 > maximum:
        maximum = num2
    if num3 > maximum:
        maximum = num3
    return maximum

# Get input from the user
try:
    n1 = float(input("Enter the first number: "))
    n2 = float(input("Enter the second number: "))
    n3 = float(input("Enter the third number: "))

    max_number = find_maximum_of_three(n1, n2, n3)
    print(f"The maximum of {n1}, {n2}, and {n3} is: {max_number}")

except ValueError:
    print("Invalid input. Please enter valid numbers.")

```

## OUTPUT:

```

Enter the first number: 3
Enter the second number: -5
Enter the third number: 9
The maximum of 3.0, -5.0, and 9.0 is: 9.0

```

## EXPLANATION:

Sure! This code snippet defines a function to find the maximum of three numbers without using the built-in `max()` function.

Here's a breakdown:

- `def find_maximum_of_three(num1, num2, num3):`: This defines a function named `find_maximum_of_three` that takes three arguments: `num1`, `num2`, and `num3`.
- `maximum = num1`: It initializes a variable `maximum` with the value of the first number, `num1`. This assumes `num1` is the maximum until proven otherwise.
- `if num2 > maximum:`: This checks if the second number, `num2`, is greater than the current `maximum`.
  - `maximum = num2`: If `num2` is greater, it updates the `maximum` to the value of `num2`.
- `if num3 > maximum:`: This checks if the third number, `num3`, is greater than the current `maximum` (which is either `num1` or `num2` at this point).
  - `maximum = num3`: If `num3` is greater, it updates the `maximum` to the value of `num3`.
- `return maximum`: After checking all three numbers, the function returns the final `maximum` value.
- The code then prompts the user to enter three numbers, converts them to floating-point numbers, calls the `find_maximum_of_three` function, and prints the maximum number found. It also includes error handling for invalid input.

👍 🗨

THANK YOU