

School of Computer Science and Artificial Intelligence

LabAssignment#4.2

Program :B.Tech(CSE)

Specialization :AIML

Course Title :AIAssistedCoding

Course Code : 23CS002PC304

Semester : VI

AcademicSession:2025-2026

Name of Student :P.Sushanthi

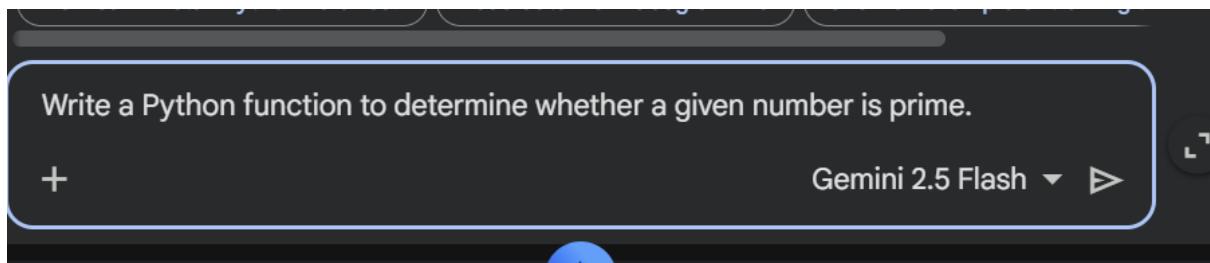
Enrollment No. : 2303A52102

Batch No. : 33

Date :19/01/26

Task-1:ZERO-SHOTPROMPTING

PromptUsed:



Code:

```
def is_prime(number):
```

```
    """
```

```
    Determines whether a given number is prime.
```

A prime number is a natural number greater than 1 that has no positive divisors other than 1 and itself.

Args:

number(int): The integer to check for primality.

Returns:

bool: True if the number is prime, False otherwise. """

```
if not isinstance(number, int):
```

```
    raise TypeError("Input must be an integer.") if
```

```
    number < 2:
```

```
    return False # Numbers less than 2 are not prime
```

```
for i in range(2, int(number**0.5) + 1):
```

```
    if number % i == 0:
```

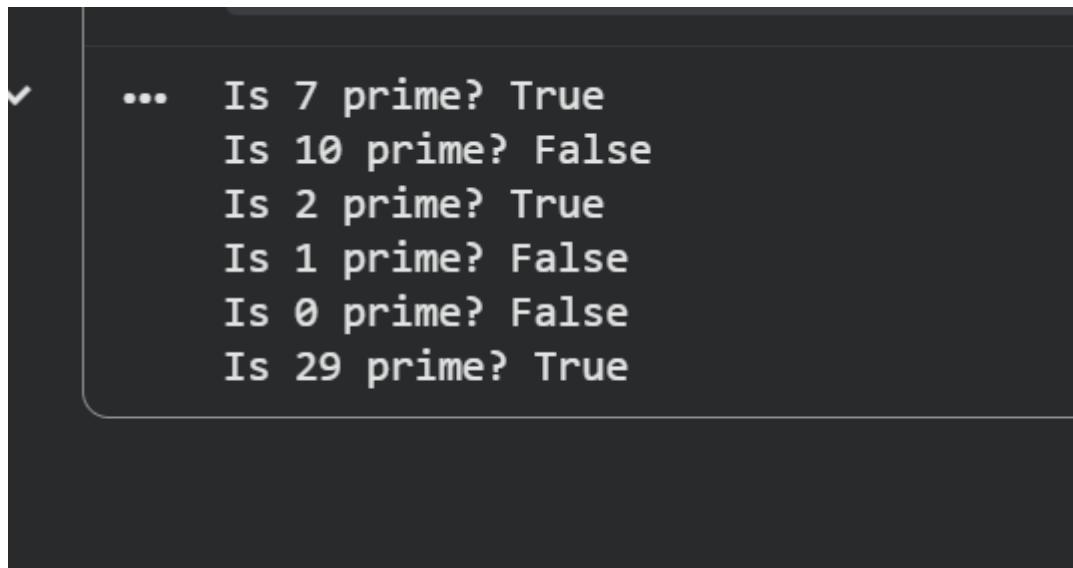
```
        return False # Found a divisor, so it's not prime
```

```
returnTrue#Nodivisorsfound,it's prime
```

#Example usage:

```
print(f'Is 7 prime? {is_prime(7)}')  
print(f'Is 10 prime? {is_prime(10)}')  
print(f'Is 2 prime? {is_prime(2)}')  
print(f'Is 1 prime? {is_prime(1)}')  
print(f'Is 0 prime? {is_prime(0)}')  
print(f'Is 29 prime? {is_prime(29)}')
```

Output:



A terminal window showing the output of the prime number check code. The output is as follows:

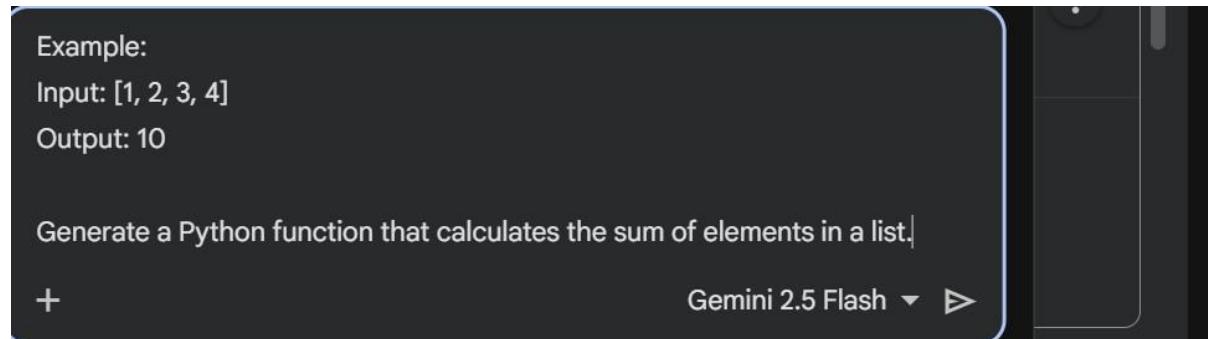
```
✓ ... Is 7 prime? True  
      Is 10 prime? False  
     Is 2 prime? True  
    Is 1 prime? False  
   Is 0 prime? False  
 Is 29 prime? True
```

Explanation

The function `is_prime()` checks whether a given number is prime. It first validates that the input is an integer and raises an error if not. Numbers less than 2 are immediately considered non-prime. The function then checks divisibility only up to the square root of the number for efficiency. If any divisor is found, it returns `False`, otherwise it returns `True`, confirming the number is prime. Example outputs demonstrate correct behavior for different inputs.

Task-2:ONE-SHOTPROMPTING

PromptUsed:



Code:

```
def calculate_list_sum(input_list):
```

```
    """
```

Calculatesthesumof allnumericelementsina list.

Args:

 input_list(list):Alistcontainingnumericelements.

Returns:

 (intorfloat):Thesumof theelementsinthe list.

Raises:

 TypeError:Iftheinputisnotalist. """

 ifnot isinstance(input_list, list):

 raiseTypeError("Inputmustbealist.")

total_sum =0

foritem in input_list:

 #Optional:Addcheckfornon-numericitemsifstrictnessisrequired

```

#if not isinstance(item,(int,float)):
#    raise TypeError(f"All elements in the list must be numbers. Found:{type(item)}")
    total_sum += item
return total_sum

#Example usage:
list1=[1, 2, 3, 4]
print(f'Input:{list1},Output:{calculate_list_sum(list1)}')

list2=[10, 20, 30]
print(f'Input:{list2},Output:{calculate_list_sum(list2)}')

list3 =[]
print(f'Input:{list3},Output:{calculate_list_sum(list3)}')

list4=[5.5, 2.5, 1.0]
print(f'Input:{list4},Output:{calculate_list_sum(list4)}')

```

Output:

```

• Input: [1, 2, 3, 4], Output: 10
  Input: [10, 20, 30], Output: 60
  Input: [], Output: 0
  Input: [5.5, 2.5, 1.0], Output: 9.0

```

Explanation

The function calculate_list_sum() computes the total sum of all elements in a given list. It first checks whether the input is a valid list and raises an error if not. The function then initializes a variable total_sum to zero and iterates through each element in the list, adding them to the total.

Finally, it returns the calculated sum. The example outputs show that the function works correctly for integer, float, and empty lists.

Task-3:FEW-SHOT PROMPTING

Prompt Used:

Examples:

Input: "a1b2c3" → Output: "123"

Input: "abc456" → Output: "456"

Input: "x9y8" → Output: "98"

Write a function to extract digits from a alphanumeric string.

Code:

```
def extract_digits(input_string):
```

```
    """
```

Extracts all digit characters from a alphanumeric string.

Args:

input_string(str): The alphanumeric string to process.

Returns:

str: A new string containing only the digits from the input_string.

Returns an empty string if no digits are found.

```
    """
```

```
if not isinstance(input_string, str):
```

```
    raise TypeError("Input must be a string.")
```

```
    digits = ""
```

```
for char in input_string:
    if char.isdigit():
        digits += char

return digits

# Example usage:
input1 = "a1b2c3"
print(f"Input: '{input1}' -> Output: '{extract_digits(input1)}'")

input2 = "abc456"
print(f"Input: '{input2}' -> Output: '{extract_digits(input2)}'")

input3 = "x9y8"
print(f"Input: '{input3}' -> Output: '{extract_digits(input3)}'")

input4 = "hello_world"
print(f"Input: '{input4}' -> Output: '{extract_digits(input4)}'")

input5 = "123testing456"
print(f"Input: '{input5}' -> Output: '{extract_digits(input5)}'")
```

Output:

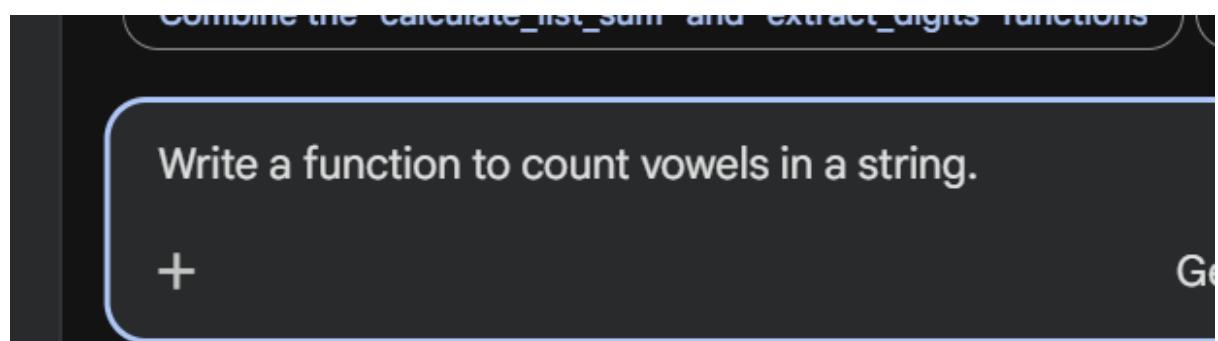
```
Input: "a1b2c3" -> Output: "123"
Input: "abc456" -> Output: "456"
Input: "x9y8" -> Output: "98"
Input: "hello_world" -> Output: ""
Input: "123testing456" -> Output: "123456"
```

Explanation:

The function extract_digits() extracts only the numeric characters from a given alphanumeric string. It first checks whether the input is a valid string and raises an error if not. The function then iterates through each character of the string and uses the isdigit() method to identify digits. All detected digits are concatenated into a new string, which is returned as the final result. If no digits are present, the function returns an empty string.

Task-4: ZERO-SHOT VS FEW-SHOT COMPARISON

A) Zero-shot Prompt



Code:

```
def count_vowels(input_string):
```

```
    """
```

Counts the number of vowels (a,e,i,o,u, case-insensitive) in a string.

Args:

 input_string(str): The string to analyze.

Returns:

 int: The total count of vowels in the string.

Raises:

```

TypeError: If the input is not a string. """
if not isinstance(input_string,str):
    raise TypeError("Input must be a string.")

vowels="aeiouAEIOU"
vowel_count = 0
for char in input_string:
    if char in vowels:
        vowel_count+=1
return vowel_count

```

#Example usage:

```

print(f"HelloWorld' has {count_vowels('Hello World')} vowels.")
print(f"PythonProgramming' has {count_vowels('PythonProgramming')} vowels.")
print(f"AEIOUaeiou' has {count_vowels('AEIOUaeiou')} vowels.")
print(f"Rhythm' has {count_vowels('Rhythm')} vowels.")
print(f"(emptystring) has {count_vowels("")} vowels.")
print(f"12345' has {count_vowels('12345')} vowels.")

```

Output:

```

--> 'Hello World' has 3 vowels.
    'Python Programming' has 4 vowels.
    'AEIOUaeiou' has 10 vowels.
    'Rhythm' has 0 vowels.
    '' (empty string) has 0 vowels.
    '12345' has 0 vowels.

```

Explanation:

The function `count_vowels()` calculates the total number of vowels in a given string. It first checks whether the input is a valid string and raises an error if not. A string containing all uppercase and lowercase vowels is defined, and the function iterates through each character of the input. If a character matches any vowel, the counter is increased. Finally, the function returns the total vowel count. The example outputs confirm correct behavior for different types of inputs.

B) Few-shot Prompt:

The screenshot shows the Gemini 2.5 Flash interface. In the input field, there is a few-shot prompt:
Examples:
Input: "hello" → Output: 2
Input: "education" → Output: 5

Generate function to count vowels

Below the input field is a plus sign (+). To the right of the input field, there is a "Gemini 2.5 Flash" dropdown and a "Run" button (represented by a right-pointing arrow). At the bottom of the interface, a note says: "Gemini can make mistakes, so double-check it and use code with caution. [Learn more](#)".

Code:

```
print(f"'hello' has {count_vowels('hello')} vowels.")  
print(f"'education' has {count_vowels('education')} vowels.")
```

Output:

```
'hello' has 2 vowels.  
'education' has 5 vowels.
```

Explanation:

This demonstrates how the function works with sample inputs. The word "*hello*" contains 2 vowels, and "*education*" contains 5 vowels, which confirms that the function correctly counts vowels in different strings.

Task-5:FEW-SHOT PROMPTING

Prompt Used:

```
Input: 10, 4, 6 → Output: 4
Input: 8, 2, 9 → Output: 2

Create a function to find minimum of three numbers WITHOUT using
min().
```

Code:

```
def find_minimum_of_three(num1, num2, num3): """
```

Findsthe minimumofthree given numbers without using the built-in min() function.

Args:

 num1 (int or float): The first number.

 num2(intorfloat):Thesecondnumber.

 num3 (int or float): The third number.

Returns:

 (intorfloat):The smallestofthe three numbers. """

```
if not all(isinstance(arg, (int, float)) for arg in [num1, num2, num3]): raise
```

 TypeError("All inputs must be numbers.")

```
if num1 <= num2 and num1 <= num3: return
```

 num1

```
elif num2<=num1 and num2<=num3: return  
    num2  
else:  
    return num3
```

#Example usage:

```
print(f"Input:3,7,5→Output:{find_minimum_of_three(3,7,5)}")  
print(f"Input:10,4,6→Output:{find_minimum_of_three(10,4,6)}")  
print(f"Input:8,2,9→Output:{find_minimum_of_three(8,2,9)}")  
print(f"Input:1.5,0.5,2.0→Output:{find_minimum_of_three(1.5,0.5,2.0)}")  
print(f"Input:5,5,5→Output:{find_minimum_of_three(5,5,5)}")
```

Output:

```
Input: 3, 7, 5 → Output: 3  
Input: 10, 4, 6 → Output: 4  
Input: 8, 2, 9 → Output: 2  
Input: 1.5, 0.5, 2.0 → Output: 0.5  
Input: 5, 5, 5 → Output: 5
```

Explanation:

The function `find_minimum_of_three()` determines the smallest value among three numbers without using the built-in `min()` function. It first verifies that all inputs are numeric values and raises an error if invalid data is provided. The function then compares the three numbers using conditional statements. If the first number is less than or equal to the others, it is returned as the minimum. Otherwise, the second or third number is returned based on comparisons. The example outputs confirm that the function correctly identifies the minimum value in different cases.

