

LAB ASSIGNMENT 4.2

KIRANMAI VENDI

HTNO: 2303A51124

Batch – 15

Task 1: Zero-Shot Prompting

Prompt Used: Write a Python function to determine whether a given number is prime.

CODE:

```
<untitled> * *
1 def is_prime(n):
2     # Prime numbers are greater than 1
3     if n <= 1:
4         return False
5     # Check divisibility up to sqrt(n)
6     for i in range(2, int(n ** 0.5) + 1):
7         if n % i == 0:
8             return False
9
10    return True
11 num = 29
12 if is_prime(num):
13     print(f"{num} is a prime number")
14 else:
15     print(f"{num} is not a prime number")
16
```

OUTPUT:

```
Shell ✘
>>> %Run -c $EDITOR_CONTENT
4 is not a prime number
>>>
```

Code Explanation:

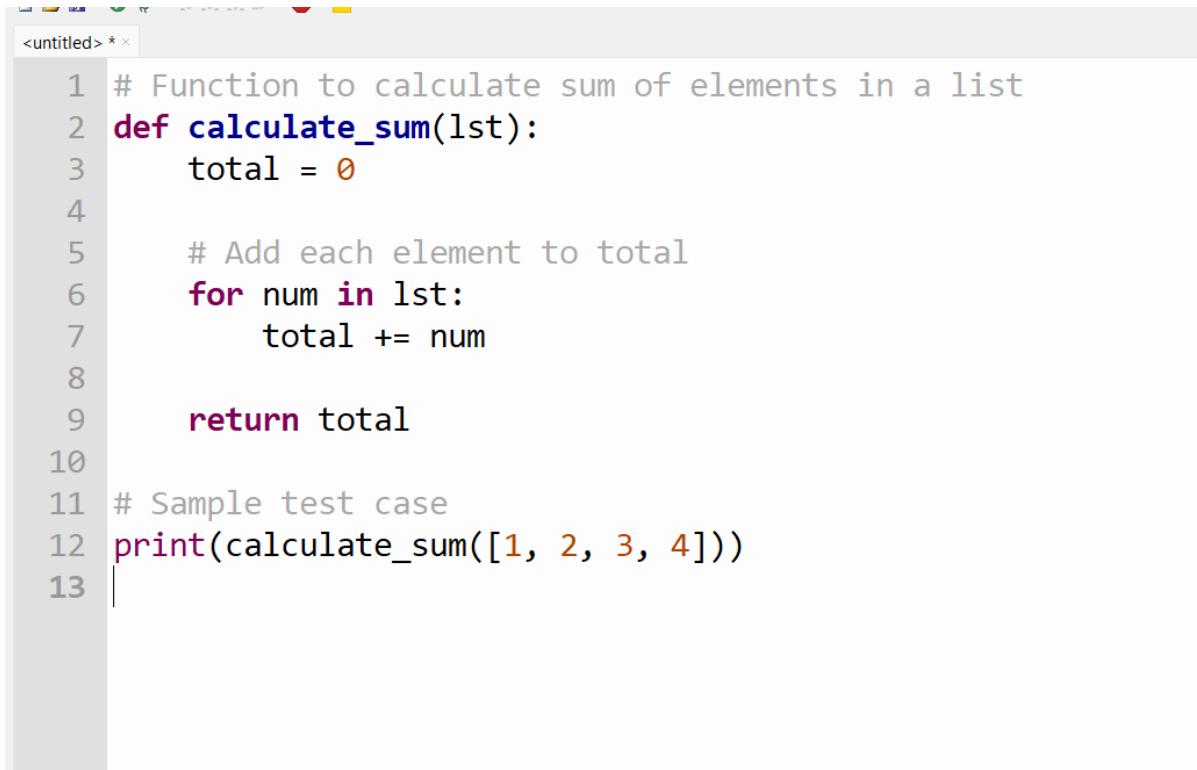
- The function is `_prime(n)` checks whether a number is prime.
- If the number is less than or equal to 1, it returns False because prime numbers start from 2.
- A loop runs from 2 up to the square root of the number to check divisibility.
- If the number is divisible by any value in this range, it is not prime.
- If no divisor is found, the function returns True, confirming the number is prime.

Task 2: One-Shot Prompting Prompt Used:

Input: [1, 2, 3, 4]

Output: 10

Write a Python function to calculate the sum of elements in a list. CODE:



```
<untitled> * 
1 # Function to calculate sum of elements in a list
2 def calculate_sum(lst):
3     total = 0
4
5     # Add each element to total
6     for num in lst:
7         total += num
8
9     return total
10
11 # Sample test case
12 print(calculate_sum([1, 2, 3, 4]))
13 |
```

OUTPUT:

```
Shell ×

>>> %Run -c $EDITOR_CONTENT
10
>>>
```

Code Explanation:

- The function initializes a variable to store the sum.
 - A loop iterates through the list.
 - Each element is added to the total.
 - The final sum is returned.
-

Task 3: Few-Shot Prompting Prompt Used:

Example 1:

Input: "a1b2c3"

Output: "123"

Example 2:

Input: "abc456"

Output: "456"

Example 3:

Input: "9x8y7"

Output: "987"

Write a Python function to extract digits from an alphanumeric string.

CODE:

```
<untitled> * 
1 # Function to extract digits from an alphanumeric string
2 def extract_digits(text):
3     digits = ""
4
5     # Loop through each character
6     for ch in text:
7         if ch.isdigit():
8             digits += ch
9
10    return digits
11
12 # Sample test cases
13 print(extract_digits("a1b2c3"))
14 print(extract_digits("abc456"))
15 |
```

OUTPUT:

```
Shell ×
>>> %Run -c $EDITOR_CONTENT
123
456
>>>
```

Code Explanation:

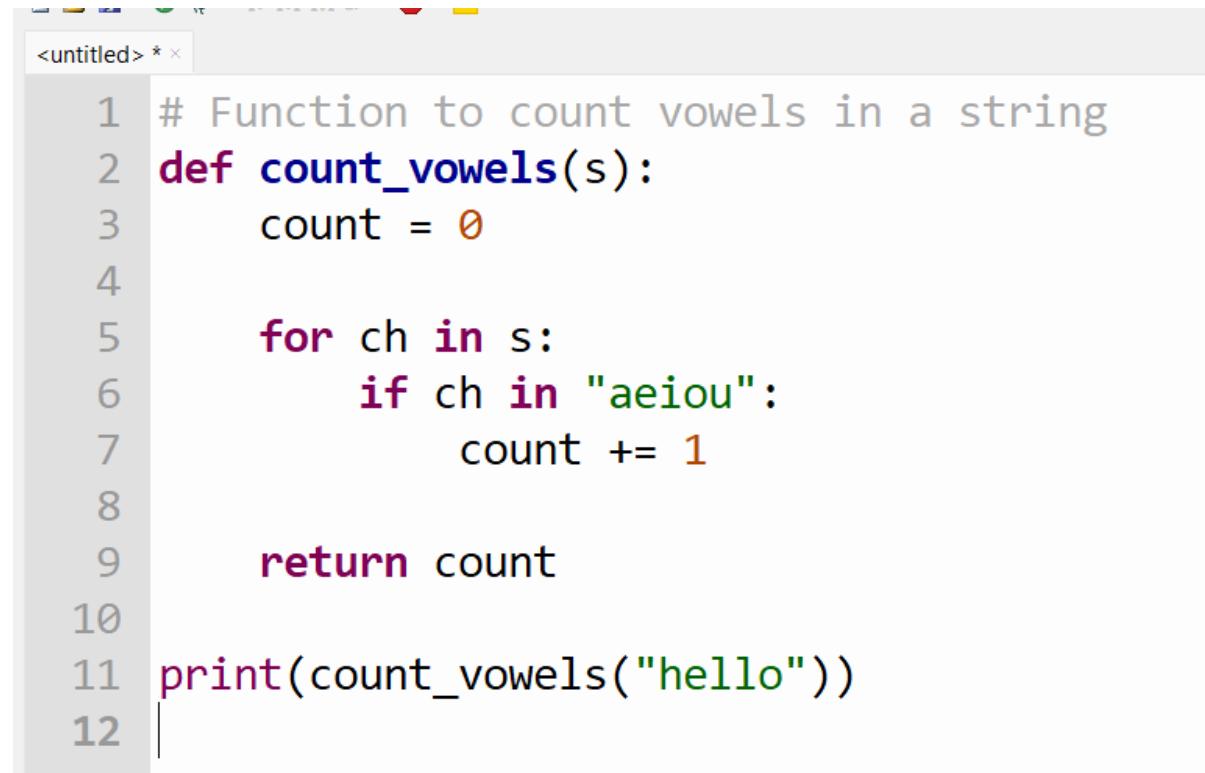
- An empty string stores digits.
- Each character is checked using `isdigit()`.
- Digits are appended to the result.
- The function returns only numeric characters.

Task 4: Zero-Shot vs Few-Shot Prompting

Zero-Shot Prompt:

Write a Python function to count the number of vowels in a string.

Zero-Shot Code:



```
<untitled> * ×
1 # Function to count vowels in a string
2 def count_vowels(s):
3     count = 0
4
5     for ch in s:
6         if ch in "aeiou":
7             count += 1
8
9     return count
10
11 print(count_vowels("hello"))
12 |
```

OUTPUT:

```
>>> %Run -c $EDITOR_CONTENT
2
>>>
```

Few-Shot Prompt:

Input: "education" Output: 5

Write a Python function to count vowels in a string.

Few-Shot Code:

```
<untitled> * 
1 # Improved function to count vowels (both cases)
2 def count_vowels(s):
3     vowels = "aeiouAEIOU"
4     return sum(1 for ch in s if ch in vowels)
5
6 print(count_vowels("Education"))
7
```

OUTPUT:

```
>>> %Run -c $EDITOR_CON
5
>>>
```

Code Explanation:

- Zero-shot handles only lowercase vowels.
- Few-shot handles both uppercase and lowercase.
- Examples improved correctness and optimisation.

Task 5: Few-Shot Prompting Prompt Used:

Example 1:

Input: 3, 5, 7

Output: 3

Example 2:

Input: 10, 2, 8

Output: 2

Example 3:

Input: 4, 4, 6

Output: 4

Write a Python function to find the minimum of three numbers without using min().

CODE:

The screenshot shows a Python code editor interface. The top window is titled 'untitled' and contains the following Python code:

```
1 # Function to find minimum of three numbers
2 def find_min(a, b, c):
3     smallest = a
4
5     if b < smallest:
6         smallest = b
7
8     if c < smallest:
9         smallest = c
10
11    return smallest
12
13 # Sample test cases
14 print(find_min(3, 5, 7))
15 print(find_min(10, 2, 8))
16
```

The bottom window is titled 'Shell' and shows the output of running the script:

```
>>> %Run -c $EDITOR_CONTENT
3
2
>>>
```

Code Explanation:

- First value is assumed as the minimum.
- It is compared with the second and third values.
- The smallest value is updated accordingly.
- Final minimum value is returned.

