AI Assisted Coding

Assignment – 4

Name: K.Chaitanya

Batch: 22

HtNo: 2303A51677

Task Description-1

• Zero-shot: Prompt AI with only the instruction. Write a Python function to determine whether a given number is prime

Code:

```python
#write a python program to find a number is prime or not

def is_prime(number):
    if not isinstance(number, int):
        return "Invalid input, please provide a number"
    if number <= 1:
        return "Provide a number greater than 1"
    for i in range(2, int(number**0.5) + 1):
        if number % i == 0:
            return f"{number} is not a prime number"
    return f"{number} is a prime number"
# Take user input
try:
    num = int(input("Enter a number to check if it is prime: "))
    result = is_prime(num)
    print(result)
except ValueError:
    print("Invalid input, please provide a number")
```

Output:

A basic Python function to check if a number is prime, demonstrating correct logical conditions without relying on examples or additional context.

```
PS C:\Users\chaitanya\New folder\AI assisted Coding> & C:/Users/chaitanya/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/chaitanya/New folder/AI assisted Coding/Assignment-4"
Enter a number to check if it is prime: 3
3 is a prime number
PS C:\Users\chaitanya\New folder\AI assisted Coding>
```

Final Observation:

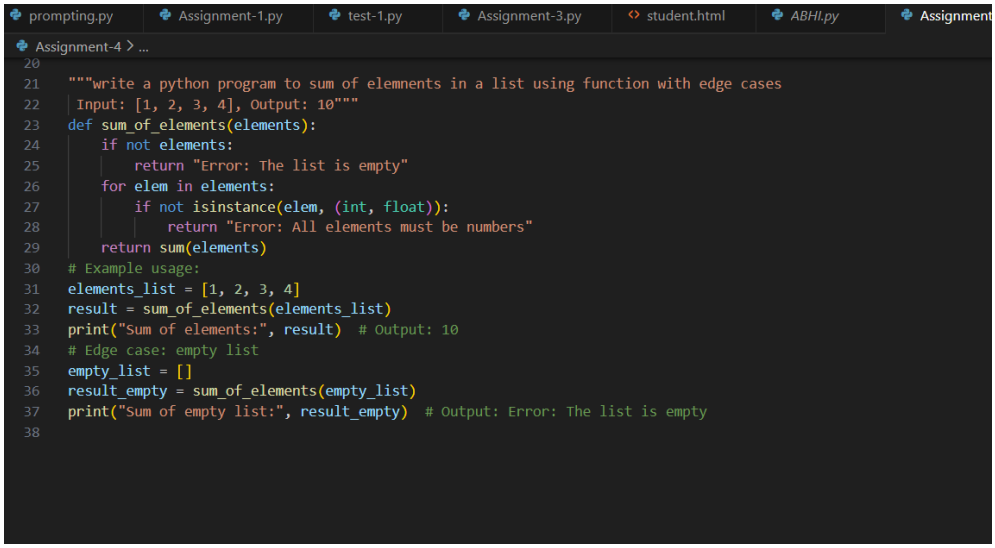
is_prime(number):
function first checks whether the number is greater than 1, since prime numbers cannot be 0, 1, or negative.
It then tests divisibility only up to √n instead of n−1, improving efficiency.
If any number in this range divides n exactly, the function returns False.
If no divisor is found, it returns True, confirming the number is prime.
This zero-shot–generated logic shows the AI's ability to apply mathematical reasoning without examples.

Task Description-2

• One-shot: Provide one example: Input: [1, 2, 3, 4], Output: 10 to help AI generate a

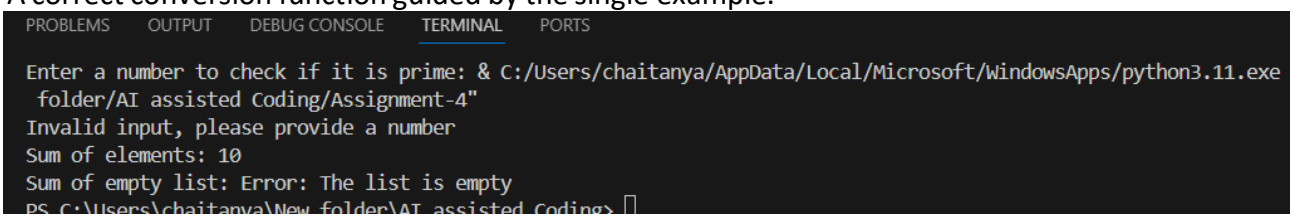
function that calculates the sum of elements in a list.

Code:



Output:

A correct conversion function guided by the single example.



Final Observation:

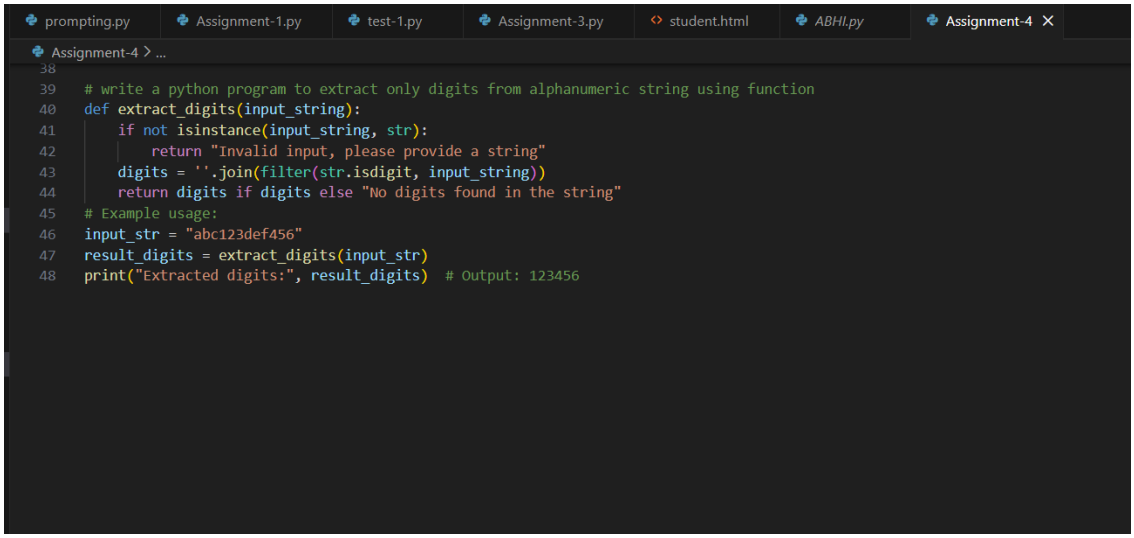
The sum_of_elements(arr) function computes the sum of all values in a list by initializing an accumulator variable to zero.
It then iterates through each element of the list, adding every value to the accumulator step by step.
Once all elements have been processed, the accumulated value represents the total sum of the list.
The function finally returns this value as the output.
Because it was created using **one-shot prompting**, the single example provided helped the AI

accurately grasp the task and generate a clear and correct solution.

Task Description-3

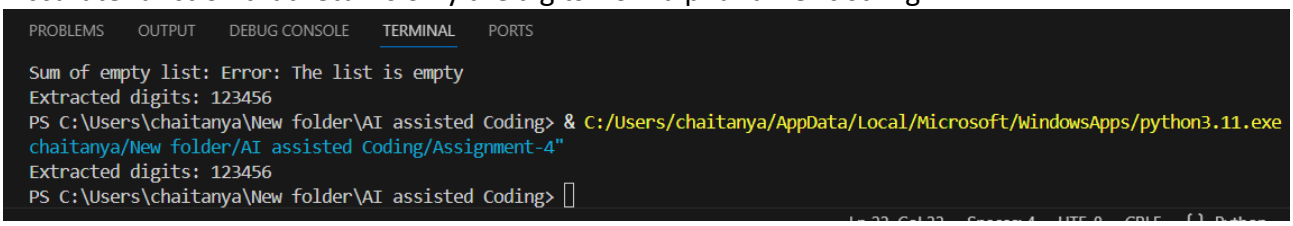
• Few-shot: Give 2–3 examples to create a function that extracts digits from an alphanumeric string.

Code:



Output:

Accurate function that returns only the digits from alphanumeric string.



Final Observation:

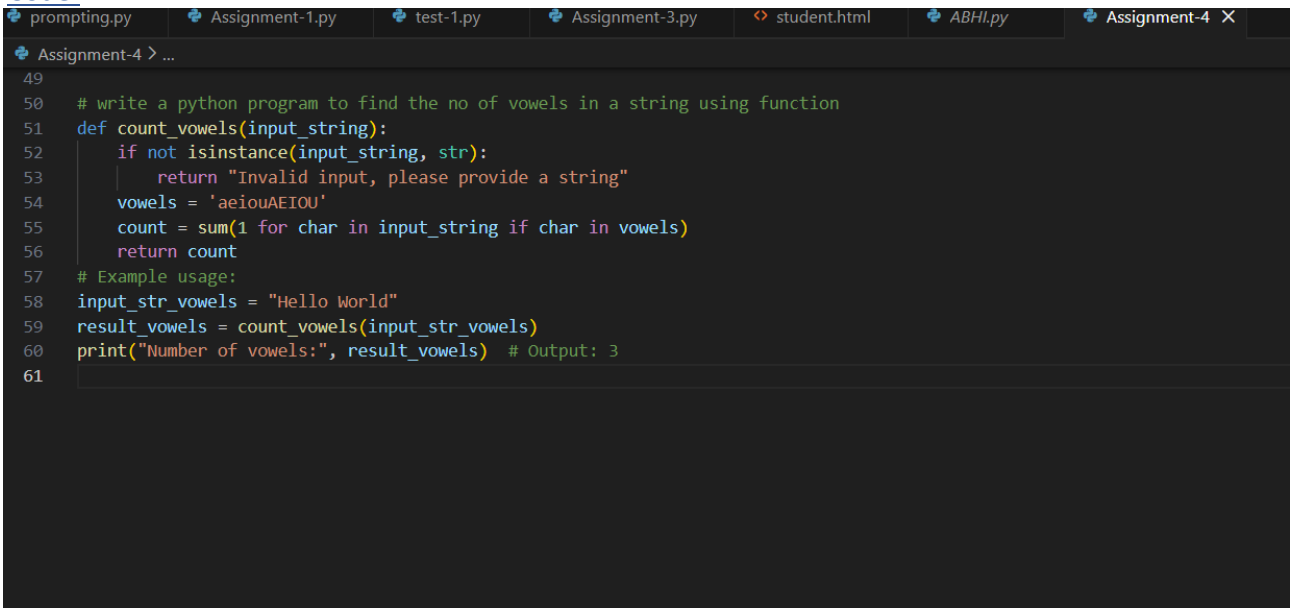
The extract_digits(text) function iterates through every character in the input string and determines whether it is a numeric character by using the isdigit() method. Whenever a digit is found, it is appended to the result string. Once all characters have been processed, the function returns a new string containing only the digits extracted from the original input. Since this function was created using few-shot prompting, the multiple examples helped the AI clearly identify the pattern of filtering out letters while retaining numbers, leading to a precise and dependable implementation.

Task Description-4

• Compare zero-shot vs few-shot prompting for generating a function that counts the number of vowels in a string.

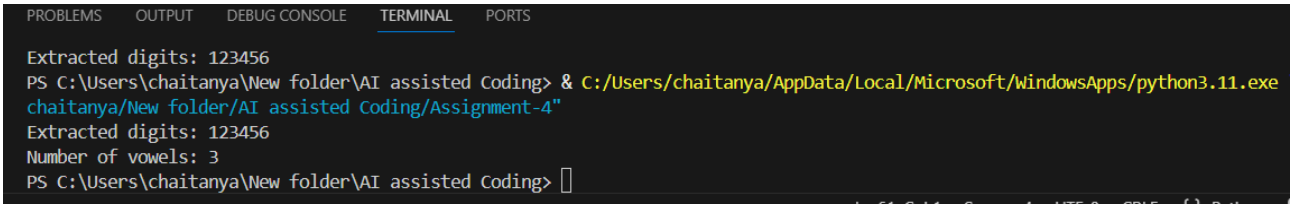
Zero Shot:

Code:



Output:

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



Output Comparison:

Few-shot version:

Guided by the provided examples, the model clearly understood that vowels in both uppercase and lowercase needed to be counted. This resulted in a more confident, optimized, and compact implementation of the function.

Zero-shot version:

The function remains correct and easy to understand, relying on a straightforward loop and a counter variable to achieve the desired result.

Final Observation:

In the zero-shot approach, the AI produced a simple vowel-counting function by relying only on the instruction, resulting in a basic solution that used a loop. In contrast, the few-shot approach included clear examples that demonstrated which characters should be treated as vowels and what the expected output should be. These examples helped the AI understand the pattern more effectively, leading to a cleaner, better-structured, and slightly optimized solution. This comparison highlights how providing examples improves clarity and often results in more accurate and reliable outputs.

Task Description-5

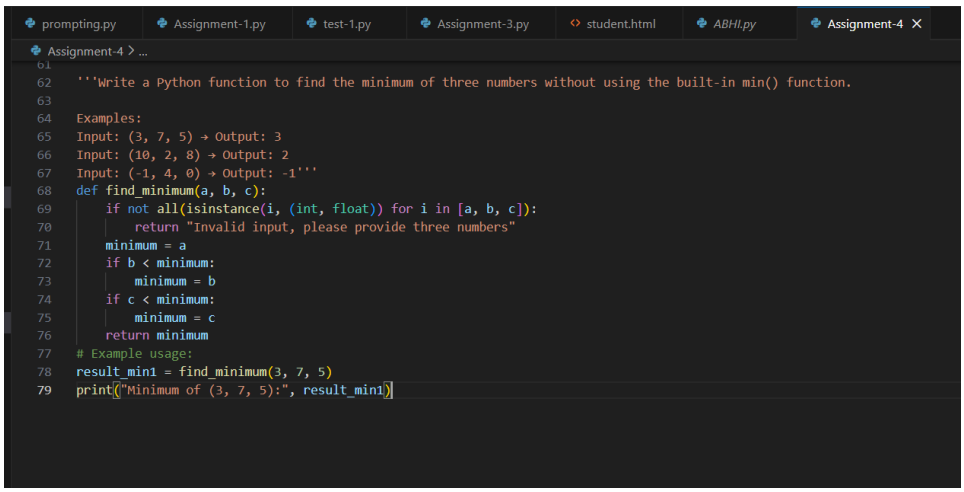
• Use few-shot prompting with 3 sample inputs to generate a function that determines

the minimum of three numbers without using the built-in min() function.

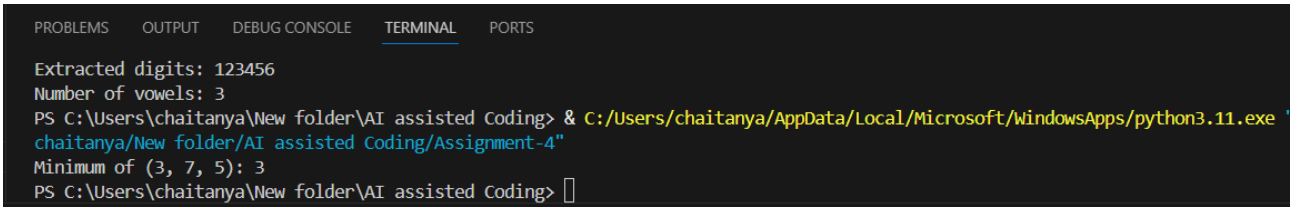
Expected Output-5

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

Code:



Output:



Final Observation:

find_minimum(a, b, c) function determines the smallest of three numbers by using conditional statements. It first verifies whether a is less than or equal to both b and c. If that condition fails, it then checks whether b is less than or equal to the remaining values. If neither condition is satisfied, the function returns c as the minimum. Because this function was created using few-shot prompting, the provided examples clearly illustrated the comparison pattern and expected output, enabling the AI to design logic that works correctly for all cases, including negative numbers.