

ASSIGNMENT-4.2

HT.NO:2303A510H6

Batch.No:30

Task 1:

Prompt:

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

Code& Output:

The screenshot shows a code editor window with a Python script. The script defines a function `is_prime` that checks if a given number is prime. It includes a usage example where the user is prompted to enter a number, and the program prints whether it is prime or not.

```
ass4.5 > ass7.5 > ass4.2 > task1 > ...
1 #Write a Python function to determine whether a given number is prime.
2 def is_prime(n):
3     if n <= 1:
4         return False
5     for i in range(2, int(n**0.5) + 1):
6         if n % i == 0:
7             return False
8     return True
9 # Example usage:
10 number = int(input("Enter a number: "))
11 if is_prime(number):
12     print(f"{number} is a prime number.")
13 else:
14     print(f"{number} is not a prime number.")
```

Below the code editor is a terminal window showing the execution of the script. The user enters the number 67, and the script correctly identifies it as a prime number.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
● PS C:\Users\ANUSHA\OneDrive\Desktop\AI-CODING> & C:/Users/ANUSHA/AppData/Local/Programs/Python/Python312\ass4.2/task1
Enter a number: 67
67 is a prime number.
○ PS C:\Users\ANUSHA\OneDrive\Desktop\AI-CODING>
```

Observation:

Zero-shot prompting worked well because the instruction was clear. The model generated correct logic without needing examples. However, we had to explicitly call the function to see output.

Task 2:

Prompt:

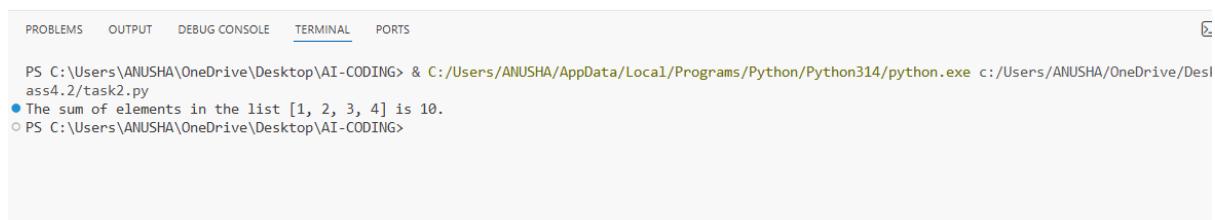
Input: [1, 2, 3, 4]

Output: 10

Write a Python function that calculates the sum of elements in a list.

Code & Output:

```
ass4.5 > ass7.5 > ass4.2 > task2.py > ...
1 | 
2 #Input: [1, 2, 3, 4] Output: 10 Write a Python function that calculates the sum of elements in a list.
3 def sum_of_elements(lst):
4     total = 0
5     for num in lst:
6         total += num
7     return total
8
9 # Example usage:
10 numbers = [1, 2, 3, 4]
11 result = sum_of_elements(numbers)
12 print(f"The sum of elements in the list {numbers} is {result}.")
```



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\ANUSHA\OneDrive\Desktop\AI-CODING> & C:/Users/ANUSHA/AppData/Local/Programs/Python/Python314/python.exe c:/Users/ANUSHA/OneDrive/Desktop/ass4.2/task2.py
● The sum of elements in the list [1, 2, 3, 4] is 10.
○ PS C:\Users\ANUSHA\OneDrive\Desktop\AI-CODING>
```

Observation:

Providing one example helped the model clearly understand that all elements in the list must be added. The single example was enough to guide the correct logic.

Task 3:

Prompt:

Example 1:

Input: "abc123"

Output: "123"

Example 2:

Input: "a1b2c3"

Output: "123"

Example 3:

Input: "hello2026world"

Output: "2026"

Write a Python function that extracts only the digits from an alphanumeric string.

Code & Output:

```
ass4.b > ass/.b > tas3.py > ...
1
2 #Example 1:
3 #Input: "abc123"
4 #Output: "123"
5 #Example 2:
6 # Input: "a1b2c3"
7 #Output: "123"
8 #Example 3:
9 #Input: "hello2026world"
10 #Output: "2026"
11 #Write a Python function that extracts only the digits from an alphanumeric string.
12 def extract_digits(s):
13     digits = ""
14     for char in s:
15         if char.isdigit():
16             digits += char
17     return digits
18 # Example usage:
19 input_string = input("Enter an alphanumeric string: ")
20 result = extract_digits(input_string)
21 print(f"The digits extracted from the string '{input_string}' are: {result}.")
```

The screenshot shows a terminal window with the following content:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python
● PS C:\Users\ANUSHA\OneDrive\Desktop\AI-CODING> & C:/Users/ANUSHA/AppData/Local/Programs/Python/Python314/python.exe c:/Users/ANUSHA/OneDrive/Desktop/AI
tas3.py
Enter an alphanumeric string: a4tfr
The digits extracted from the string 'a4tfr' are: 4.
○ PS C:\Users\ANUSHA\OneDrive\Desktop\AI-CODING>
```

Observation:

Multiple examples made the pattern very clear. The model understood that only numeric characters should be kept and everything else ignored. Few-shot improved clarity and accuracy.

Task 4:

Prompt:

Zero-shot:

Write a Python function that counts the number of vowels in a string.

Few-shot with examples:

Example 1:

Input: "hello"

Output: 2

Example 2:

Input: "AEIOU"

Output: 5

Example 3:

Input: "Python"

Output: 1

Write a Python function that counts the number of vowels in a string.

Code & Output:

```

ass4.5 > ass7.5 > task4.4.2.py > ...
1
2 #Zero-shot: Write a Python function that counts the number of vowels in a string.
3 def count_vowels(s):
4     vowels = "aeiouAEIOU"
5     count = 0
6     for char in s:
7         if char in vowels:
8             count += 1
9     return count
10 # Example usage:
11 input_string = input("Enter a string: ")
12 result = count_vowels(input_string)
13 print(f"The number of vowels in the string '{input_string}' is: {result}.")
14
15 #Write a Python function that counts the number of vowels in a string.
16 def count_vowels(s):
17     vowels = "aeiouAEIOU"
18     count = 0
19     for char in s:
20         if char in vowels:
21             count += 1
22     return count
23 # Example usage:
24 input_string = input("Enter a string: ")
25 result = count_vowels(input_string)
26 print(f"The number of vowels in the string '{input_string}' is: {result}.")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\ANUSHA\OneDrive\Desktop\AI-CODING> & C:/Users/ANUSHA/AppData/Local/Programs/Python/Python314/python.exe c:/Users/ANUSHA/OneDrive/D task4.4.2.py
Enter a string: asdf
The number of vowels in the string 'asdf' is: 1.
Enter a string: ■

Observation:

Zero-shot generated correct logic, but few-shot was more reliable. The examples helped ensure uppercase and lowercase vowels were handled properly. Few-shot reduced ambiguity.

Task 5:

Prompt:

Example 1:

Input: (3, 7, 5)

Output: 3

Example 2:

Input: (10, 2, 8)

Output: 2

Example 3:

Input: (4, 4, 9)

Output: 4

Write a Python function that determines the minimum of three numbers **without using the built-in min() function**.

Code & Output:

```
ass4.5 > ass7.5 > task5.4.2.py > ...
1  #Write a Python function that determines the minimum of three numbers without using the built-in min() function.
2  def find_minimum(a, b, c):
3      minimum = a # Assume a is the minimum
4      if b < minimum:
5          minimum = b # Update minimum if b is smaller
6      if c < minimum:
7          minimum = c # Update minimum if c is smaller
8      return minimum
9  # Example usage:
10 num1 = float(input("Enter the first number: "))
11 num2 = float(input("Enter the second number: "))
12 num3 = float(input("Enter the third number: "))
13 result = find_minimum(num1, num2, num3)
14 print(f"The minimum of the numbers {num1}, {num2}, and {num3} is: {result}.")
```

```
PS C:\Users\ANUSHA\OneDrive\Desktop\AI-CODING> & C:/Users/ANUSHA/AppData/Local/Programs/Python/Python314/python.exe c:/Users/ANUSHA/OneDrive/Desktop/task5.4.2.py
Enter the first number: 23
Enter the second number: 45
Enter the third number: 56
The minimum of the numbers 23.0, 45.0, and 56.0 is: 23.0.
PS C:\Users\ANUSHA\OneDrive\Desktop\AI-CODING>
```

Observation:

The three examples helped the model understand comparison logic, including equal values. The function correctly compares numbers without using the built-in min() function.