AI Assisted Coding

Assignment – 4.2

Name: **JILLELA AKSHAYA**

Batch: **22**

HtNo: **2303A51629**

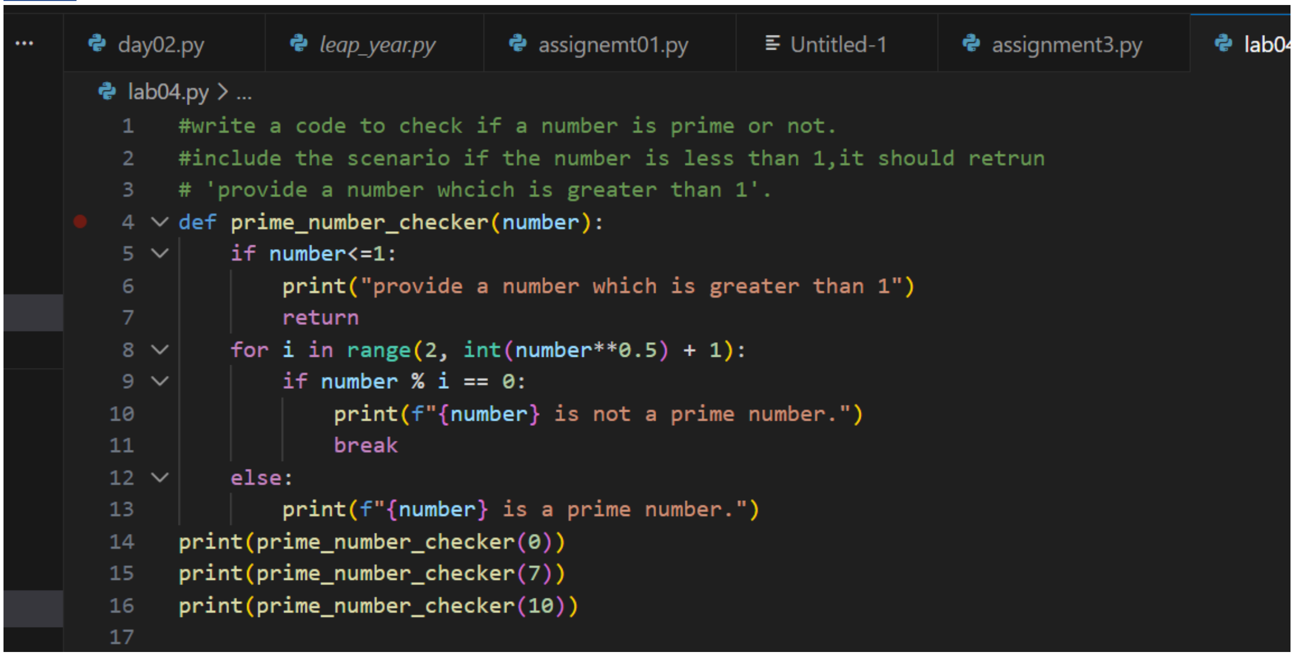
Task Description-1

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

Expected Output-1

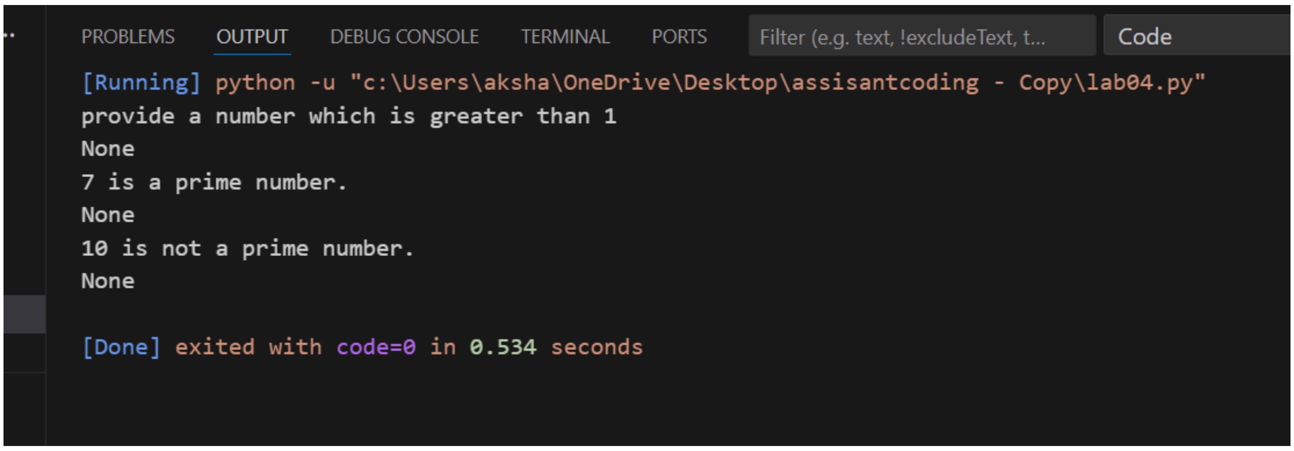
• A basic Python function to check if a number is prime, demonstrating correct logical conditions without relying on examples or additional context.

Code:



```
#write a code to check if a number is prime or not.
#include the scenario if the number is less than 1,it should retrun
# 'provide a number whcich is greater than 1'.
def prime_number_checker(number):
    if number<=1:
        print("provide a number which is greater than 1")
        return
    for i in range(2, int(number**0.5) + 1):
        if number % i == 0:
            print(f"{number} is not a prime number.")
            break
    else:
        print(f"{number} is a prime number.")
print(prime_number_checker(0))
print(prime_number_checker(7))
print(prime_number_checker(10))
```

Output:



```
[Running] python -u "c:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy\lab04.py"
provide a number which is greater than 1
None
7 is a prime number.
None
10 is not a prime number.
None

[Done] exited with code=0 in 0.534 seconds
```

Final Observation:

The is_prime(n) function checks whether a number is prime by first making sure the number is greater than 1, since 0, 1, and negative numbers are not prime. It then goes through the numbers from 2 to n−1 to see if any of them divide the given number exactly. If it finds even one such number, it returns False, meaning the number is not prime. If no divisor is found, it returns True, confirming that the number is prime. Because this solution was generated using zero-shot prompting, without giving any examples, it shows that the AI can understand the concept and produce a correct but basic prime-checking logic.

Task Description-2

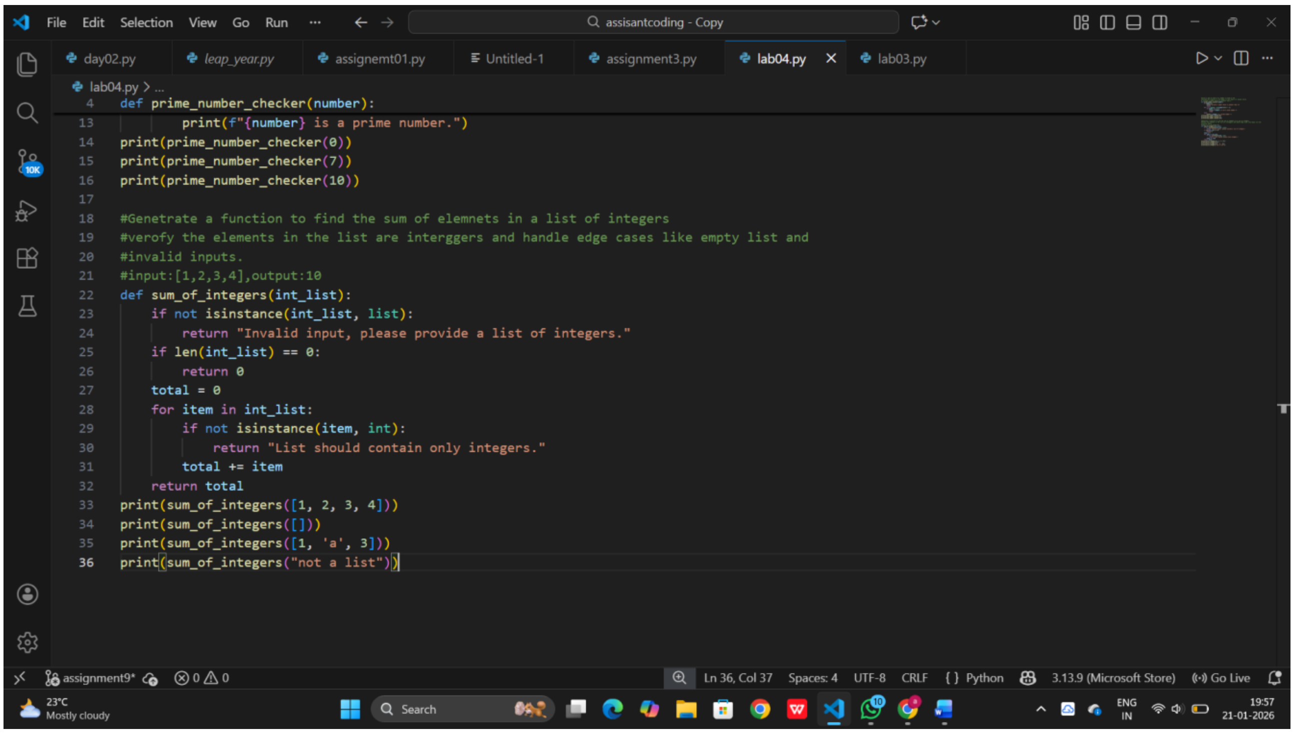
• One-shot: Provide one example: Input: [1, 2, 3, 4], Output: 10 to help AI generate a

function that calculates the sum of elements in a list.

Expected Output-2

• A correct conversion function guided by the single example.

Code:

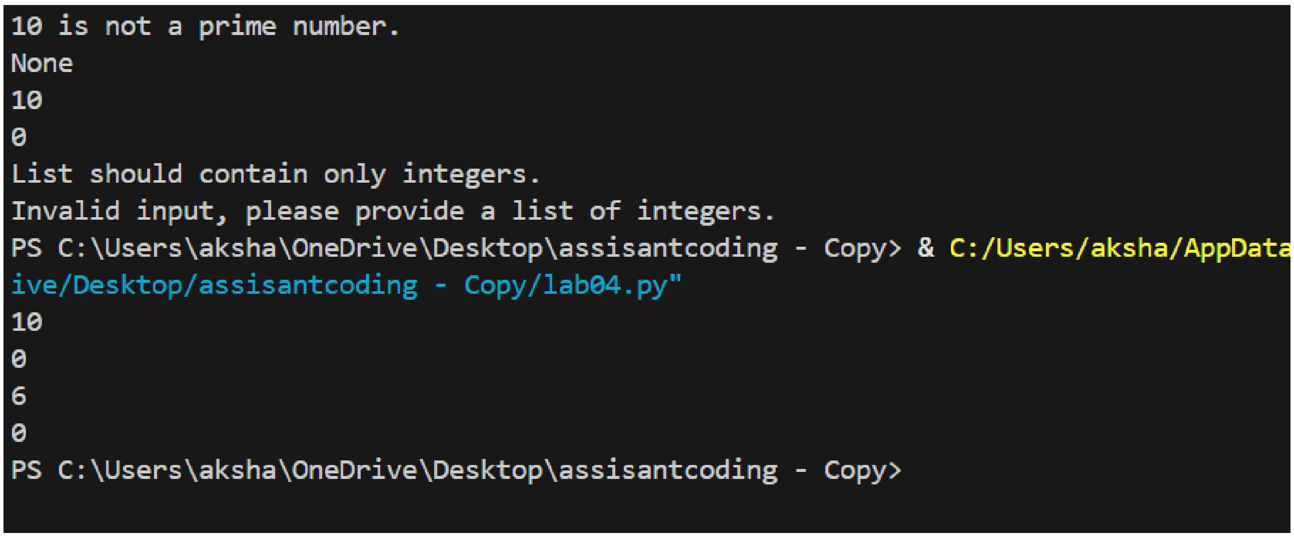


```python
def prime_number_checker(number):
            print(f"{number} is a prime number.")
print(prime_number_checker(0))
print(prime_number_checker(7))
print(prime_number_checker(10))

#Genetrate a function to find the sum of elemnets in a list of integers
#verofy the elements in the list are intergers and handle edge cases like empty list and
#invalid inputs.
#input:[1,2,3,4],output:10
def sum_of_integers(int_list):
    if not isinstance(int_list, list):
        return "Invalid input, please provide a list of integers."
    if len(int_list) == 0:
        return 0
    total = 0
    for item in int_list:
        if not isinstance(item, int):
            return "List should contain only integers."
        total += item
    return total
print(sum_of_integers([1, 2, 3, 4]))
print(sum_of_integers([]))
print(sum_of_integers([1, 'a', 3]))
print(sum_of_integers("not a list"))
```

•

Output:



```
10 is not a prime number.
None
10
0
List should contain only integers.
Invalid input, please provide a list of integers.
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy> & C:/Users/aksha/AppData
ive/Desktop/assisantcoding - Copy/lab04.py"
10
0
6
0
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy>
```

Final Observation:

The sum_of_elements(arr) function calculates the total sum of all elements in a list by starting with a variable total set to zero and then adding each number in the list one by one. After the loop finishes, the final value of total is returned as the sum of the list. Since this function was generated using one-shot prompting, where a single input-output example was provided, the example helped guide the AI to clearly understand the task and produce a correct and simple function for summing list elements.

Task Description-3

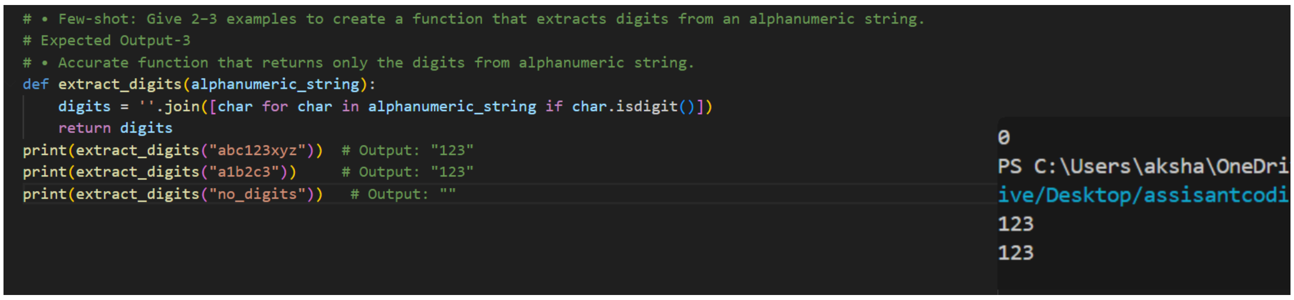
• Few-shot: Give 2–3 examples to create a function that extracts digits from an alphanumeric string.

Expected Output-3

• Accurate function that returns only the digits from alphanumeric string.

Code:

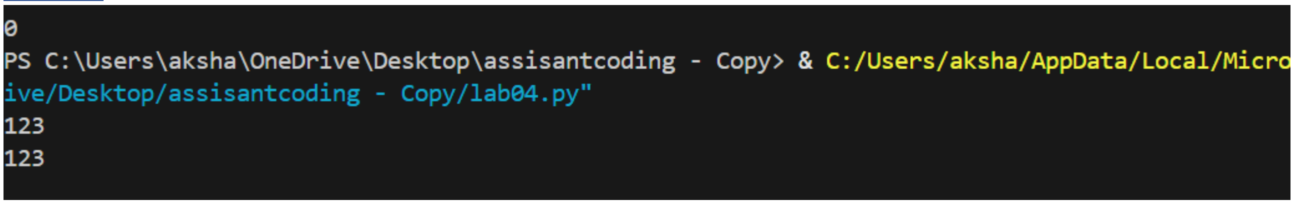
•



```python
# • Few-shot: Give 2-3 examples to create a function that extracts digits from an alphanumeric string.
# Expected Output-3
# • Accurate function that returns only the digits from alphanumeric string.
def extract_digits(alphanumeric_string):
    digits = ''.join([char for char in alphanumeric_string if char.isdigit()])
    return digits
print(extract_digits("abc123xyz"))  # Output: "123"
print(extract_digits("a1b2c3"))     # Output: "123"
print(extract_digits("no_digits"))   # Output: ""
```

```
0
PS C:\Users\aksha\OneDri
ive/Desktop/assisantcodi
123
123
```

Output:



```
0
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy> & C:/Users/aksha/AppData/Local/Micro
ive/Desktop/assisantcoding - Copy/lab04.py"
123
123
```

Final Observation:

The extract_digits(text) function goes through each character in the given string and checks whether it is a digit using the isdigit() method. If the character is a digit, it is added to the result string. After the loop finishes, the function returns a string that contains only the digits from the original input. Since this was generated using few-shot prompting, the multiple examples clearly guided the AI to recognize the pattern of removing letters and keeping only numbers, resulting in an accurate and reliable function.

Task Description-4

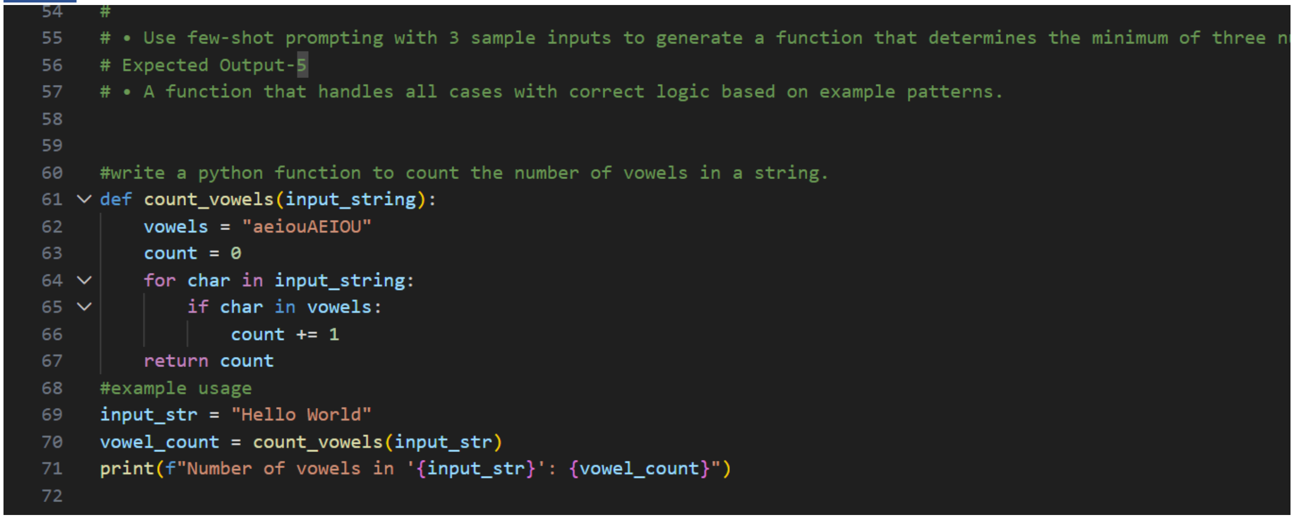
• Compare zero-shot vs few-shot prompting for generating a function that counts the number of vowels in a string.

Expected Output-4

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

Zero Shot:

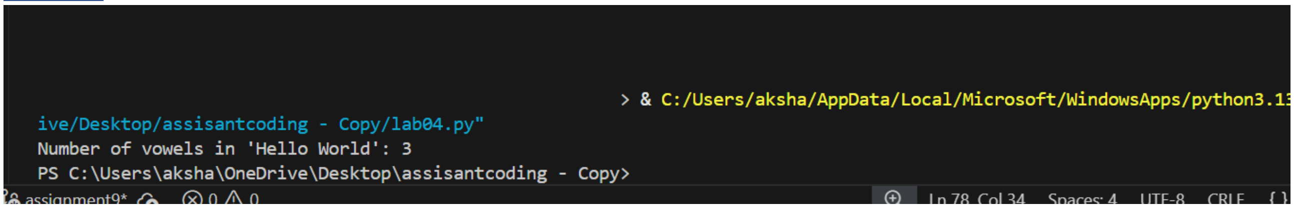
Code:



```python
#
# • Use few-shot prompting with 3 sample inputs to generate a function that determines the minimum of three n
# Expected Output-5
# • A function that handles all cases with correct logic based on example patterns.


#write a python function to count the number of vowels in a string.
def count_vowels(input_string):
    vowels = "aeiouAEIOU"
    count = 0
    for char in input_string:
        if char in vowels:
            count += 1
    return count
#example usage
input_str = "Hello World"
vowel_count = count_vowels(input_str)
print(f"Number of vowels in '{input_str}': {vowel_count}")
```

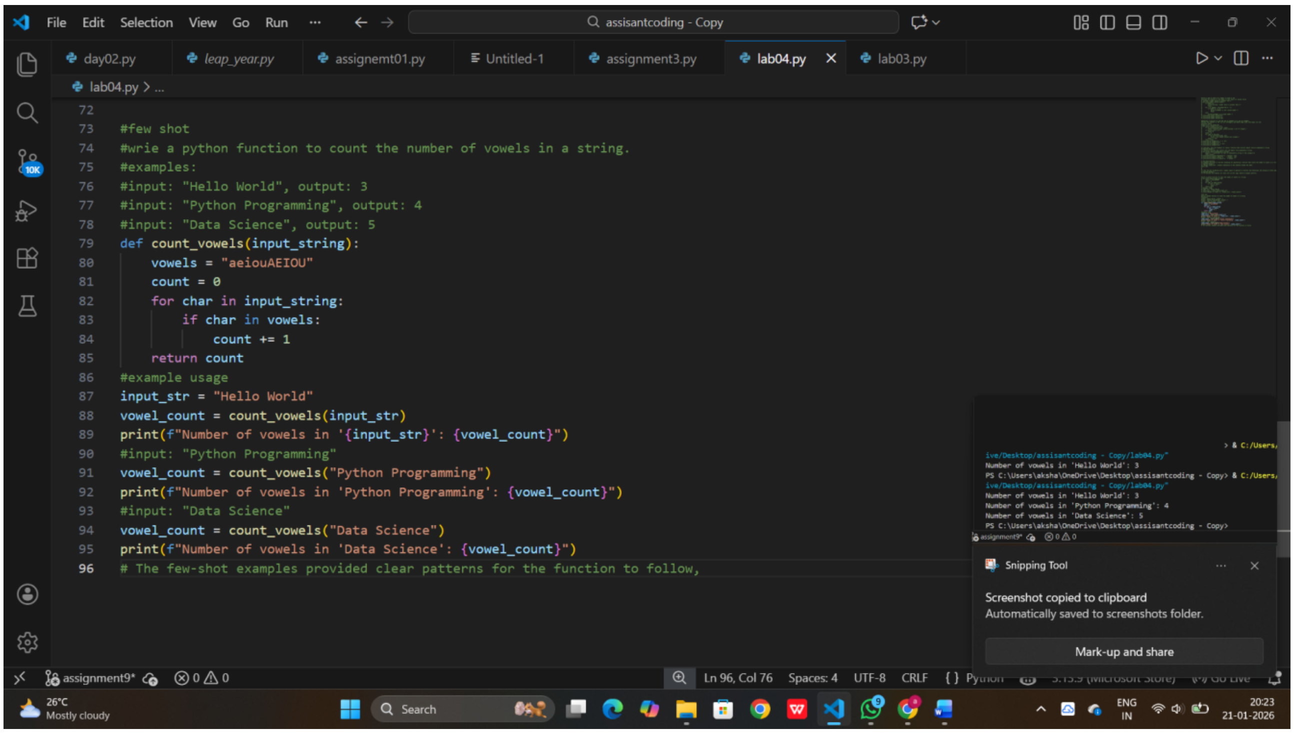
Output:



```
                                                > & C:/Users/aksha/AppData/Local/Microsoft/WindowsApps/python3.13
ive/Desktop/assisantcoding - Copy/lab04.py"
Number of vowels in 'Hello World': 3
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy>
```

Few Shot:

Code:

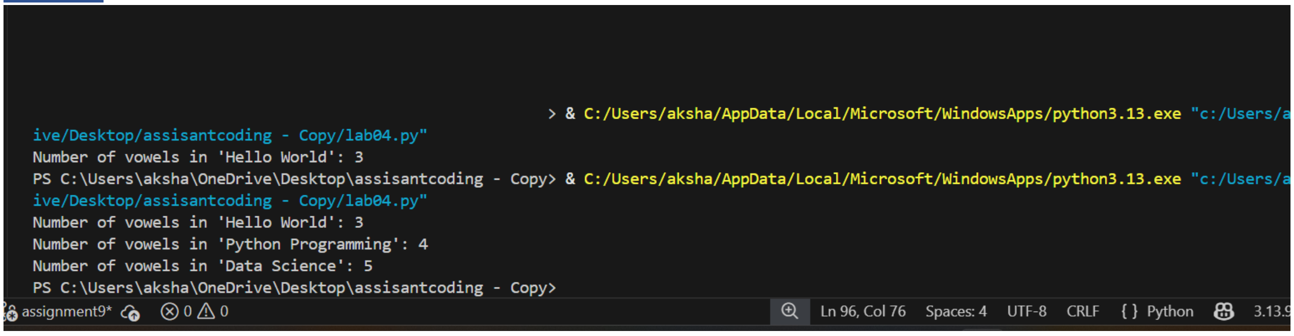


```python

#few shot
#wrie a python function to count the number of vowels in a string.
#examples:
#input: "Hello World", output: 3
#input: "Python Programming", output: 4
#input: "Data Science", output: 5
def count_vowels(input_string):
    vowels = "aeiouAEIOU"
    count = 0
    for char in input_string:
        if char in vowels:
            count += 1
    return count
#example usage
input_str = "Hello World"
vowel_count = count_vowels(input_str)
print(f"Number of vowels in '{input_str}': {vowel_count}")
#input: "Python Programming"
vowel_count = count_vowels("Python Programming")
print(f"Number of vowels in 'Python Programming': {vowel_count}")
#input: "Data Science"
vowel_count = count_vowels("Data Science")
print(f"Number of vowels in 'Data Science': {vowel_count}")
# The few-shot examples provided clear patterns for the function to follow,
```

```
                                                          > & C:/Users,
ive/Desktop/assisantcoding - Copy/lab04.py"
Number of vowels in 'Hello World': 3
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy> & C:/Users,
ive/Desktop/assisantcoding - Copy/lab04.py"
Number of vowels in 'Hello World': 3
Number of vowels in 'Python Programming': 4
Number of vowels in 'Data Science': 5
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy>
```

Output:



```
                                                   > & C:/Users/aksha/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/a
ive/Desktop/assisantcoding - Copy/lab04.py"
Number of vowels in 'Hello World': 3
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy> & C:/Users/aksha/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/a
ive/Desktop/assisantcoding - Copy/lab04.py"
Number of vowels in 'Hello World': 3
Number of vowels in 'Python Programming': 4
Number of vowels in 'Data Science': 5
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy>
```

Output Comparison:

Zero-shot version:

The function is clear, simple, and correct, using a loop and counter variable.

Few-shot version:

The function is more refined and compact. The examples helped the model clearly understand that both uppercase and lowercase vowels must be counted, leading to a more confident and optimized implementation.

Final Observation:

In the zero-shot approach, the AI generated a basic vowel-counting function based only on the instruction, which resulted in a straightforward loop-based solution. In the few-shot approach, the given examples clearly showed what should be considered as vowels and how the output should look. These examples helped the AI better understand the pattern and expectations, which led to a cleaner and slightly optimized solution. This comparison shows that providing examples improves clarity and often results in better-structured and more accurate outputs.

Task Description-5

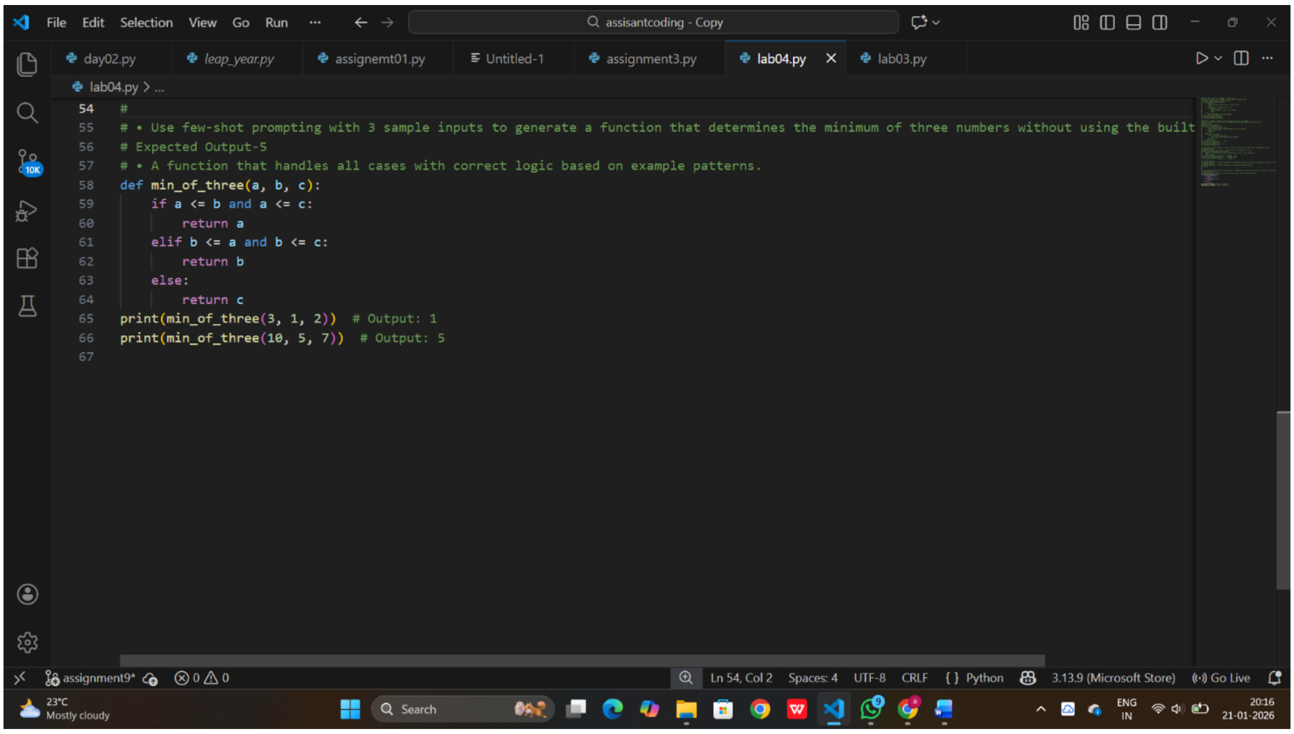
• Use few-shot prompting with 3 sample inputs to generate a function that determines

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

Expected Output-5

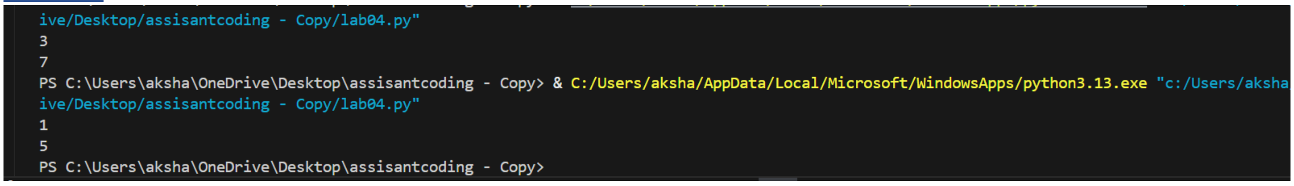
• A function that handles all cases with correct logic based on example patterns.

Code:

- 

```python
#
# • Use few-shot prompting with 3 sample inputs to generate a function that determines the minimum of three numbers without using the built
# Expected Output-5
# • A function that handles all cases with correct logic based on example patterns.
def min_of_three(a, b, c):
    if a <= b and a <= c:
        return a
    elif b <= a and b <= c:
        return b
    else:
        return c
print(min_of_three(3, 1, 2))  # Output: 1
print(min_of_three(10, 5, 7))  # Output: 5
```

Output:



```
ive/Desktop/assisantcoding - Copy/lab04.py"
3
7
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy> & C:/Users/aksha/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/aksha
ive/Desktop/assisantcoding - Copy/lab04.py"
1
5
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy>
```

Final Observation:

The find_minimum(a, b, c) function compares the three given numbers using conditional statements. It first checks whether a is smaller than or equal to both b and c. If not, it then checks whether b is smaller than or equal to the other two. If neither of these conditions is true, the function returns c as the minimum. Since this function was generated using few-shot prompting, the three examples clearly showed the expected pattern of comparing values and returning the smallest one. These examples helped the AI design logic that correctly handles all cases, including negative numbers.