

# Python Developer Task - 4

Submitted by: Khushi Maurya

Company: Main Flow Services and Technologies Pvt. Ltd.

## Objective

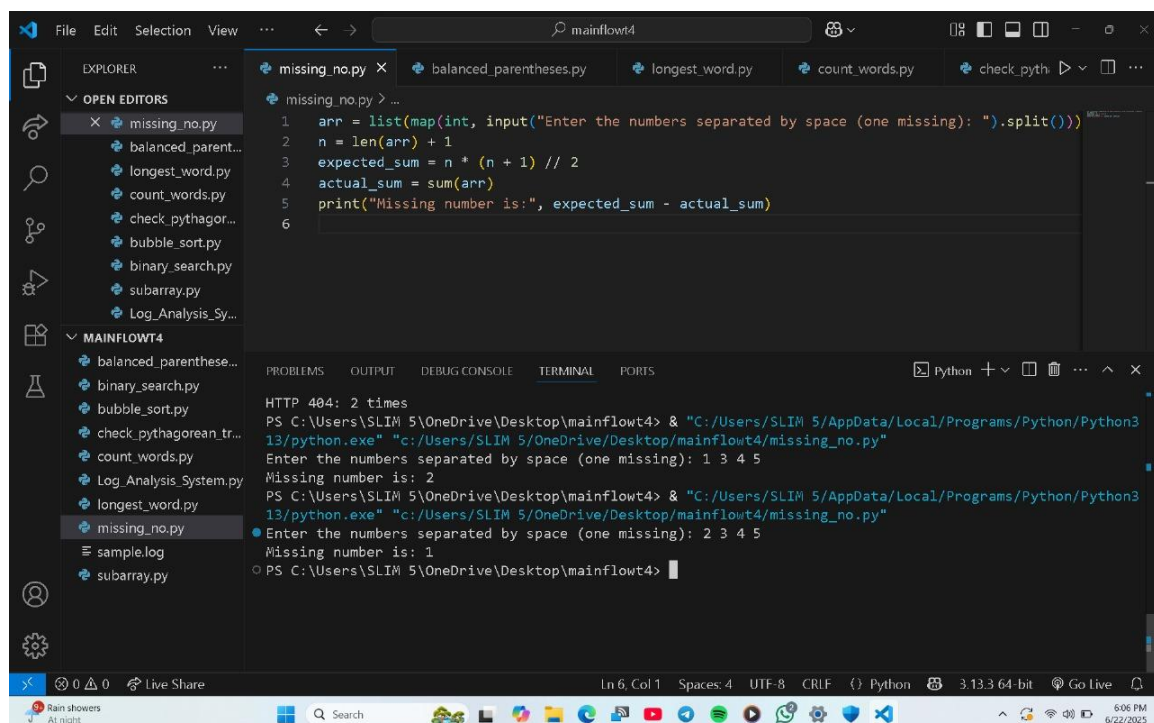
The objective of Task 4 was to implement Python programs focusing on logic development, data structures, and real-world applications like log file analysis. It covered problems involving missing number detection, sorting and searching, string analysis, and an efficient system to analyze log data.

## Task-wise Approach

### 1. Find Missing Number

Calculated the expected sum of numbers from 1 to  $n+1$  and subtracted the sum of the actual list to find the missing number.

Screenshot:



The screenshot displays a Python IDE with the file explorer on the left showing a project named 'mainflowt4'. The 'missing\_no.py' file is open in the editor, containing the following code:

```
1 arr = list(map(int, input("Enter the numbers separated by space (one missing): ").split()))
2 n = len(arr) + 1
3 expected_sum = n * (n + 1) // 2
4 actual_sum = sum(arr)
5 print("Missing number is:", expected_sum - actual_sum)
6
```

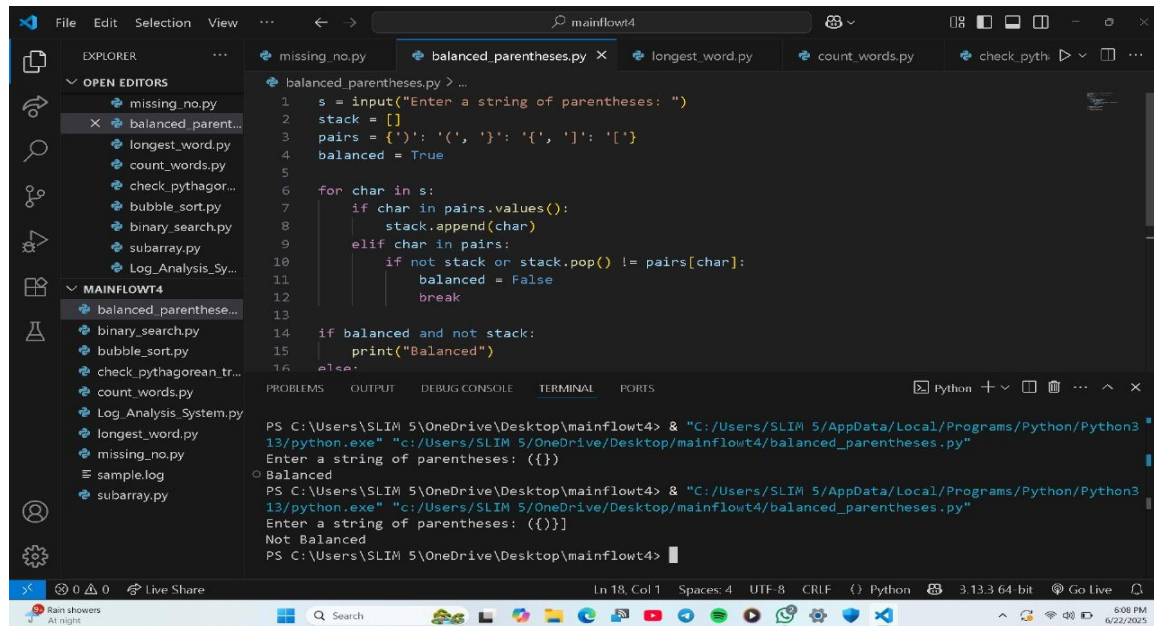
The terminal at the bottom shows the execution of the script. It prompts the user to enter numbers, and two examples are shown: one with the input '1 3 4 5' resulting in a missing number of 2, and another with the input '2 3 4 5' resulting in a missing number of 1.

```
HTTP 404: 2 times
PS C:\Users\SLIM 5\OneDrive\Desktop\mainflowt4> & "C:/Users/SLIM 5/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/SLIM 5/OneDrive/Desktop/mainflowt4/missing_no.py"
Enter the numbers separated by space (one missing): 1 3 4 5
Missing number is: 2
PS C:\Users\SLIM 5\OneDrive\Desktop\mainflowt4> & "C:/Users/SLIM 5/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/SLIM 5/OneDrive/Desktop/mainflowt4/missing_no.py"
Enter the numbers separated by space (one missing): 2 3 4 5
Missing number is: 1
PS C:\Users\SLIM 5\OneDrive\Desktop\mainflowt4>
```

## 2. Check Balanced Parentheses

Used a stack to ensure each opening bracket has a correct matching closing bracket.

Screenshot:



The screenshot shows a Visual Studio Code editor window with the file `balanced_parentheses.py` open. The code uses a stack to check if parentheses in a string are balanced. The terminal at the bottom shows the execution of the script, which prompts the user to enter a string of parentheses. The first input is `()()`, which is correctly identified as balanced. The second input is `(())`, which is also correctly identified as balanced. The third input is `(())`, which is incorrectly identified as not balanced (likely a typo in the original image for `(())`).

```
1 s = input("Enter a string of parentheses: ")
2 stack = []
3 pairs = {'(': ')', '[': ']', '{': '}'}
4 balanced = True
5
6 for char in s:
7     if char in pairs.values():
8         stack.append(char)
9     elif char in pairs:
10        if not stack or stack.pop() != pairs[char]:
11            balanced = False
12            break
13
14 if balanced and not stack:
15     print("Balanced")
16 else:
17     print("Not Balanced")
```

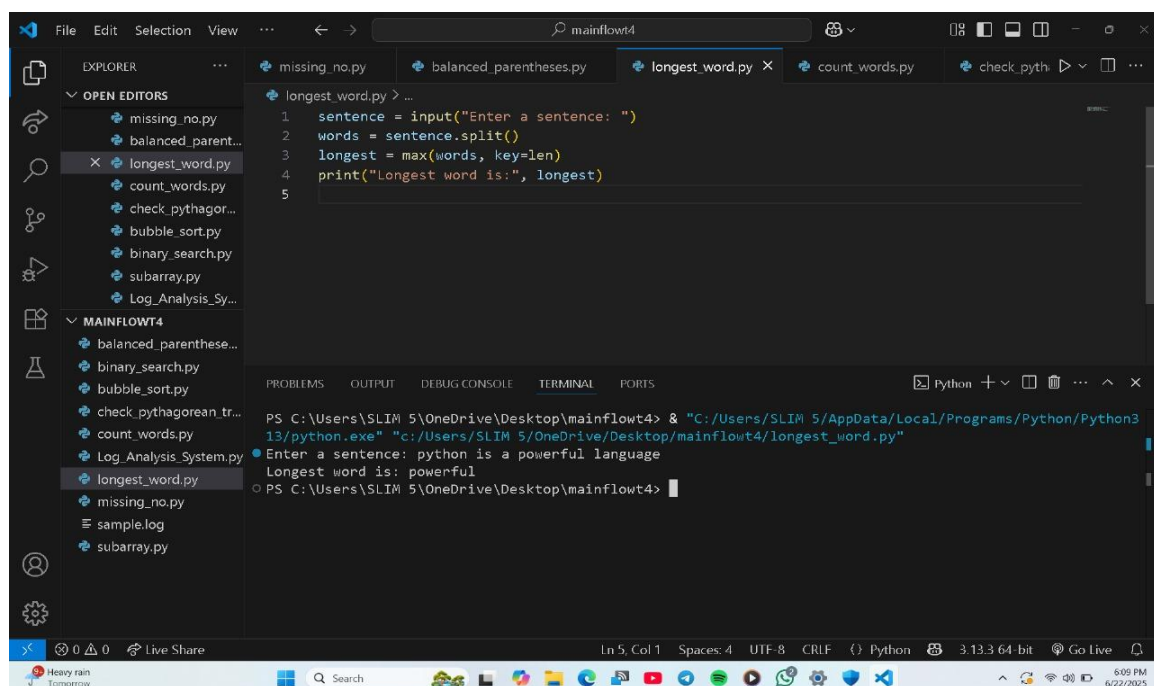
Terminal output:

```
PS C:\Users\SLIM 5\OneDrive\Desktop\mainflowt4> & "C:/Users/SLIM 5/AppData/Local/Programs/Python/Python3
13/python.exe" "c:/Users/SLIM 5/OneDrive/Desktop/mainflowt4/balanced_parentheses.py"
Enter a string of parentheses: ()()
Balanced
PS C:\Users\SLIM 5\OneDrive\Desktop\mainflowt4> & "C:/Users/SLIM 5/AppData/Local/Programs/Python/Python3
13/python.exe" "c:/Users/SLIM 5/OneDrive/Desktop/mainflowt4/balanced_parentheses.py"
Enter a string of parentheses: (())
Not Balanced
PS C:\Users\SLIM 5\OneDrive\Desktop\mainflowt4>
```

## 3. Longest Word in Sentence

Split the sentence using spaces and identified the longest word based on length.

Screenshot:



The screenshot shows a Visual Studio Code editor window with the file `longest_word.py` open. The code prompts the user to enter a sentence, splits it into words, and prints the longest word. The terminal at the bottom shows the execution of the script, which prompts the user to enter a sentence. The input is `python is a powerful language`, and the output is `Longest word is: powerful`.

```
1 sentence = input("Enter a sentence: ")
2 words = sentence.split()
3 longest = max(words, key=len)
4 print("Longest word is:", longest)
5
```

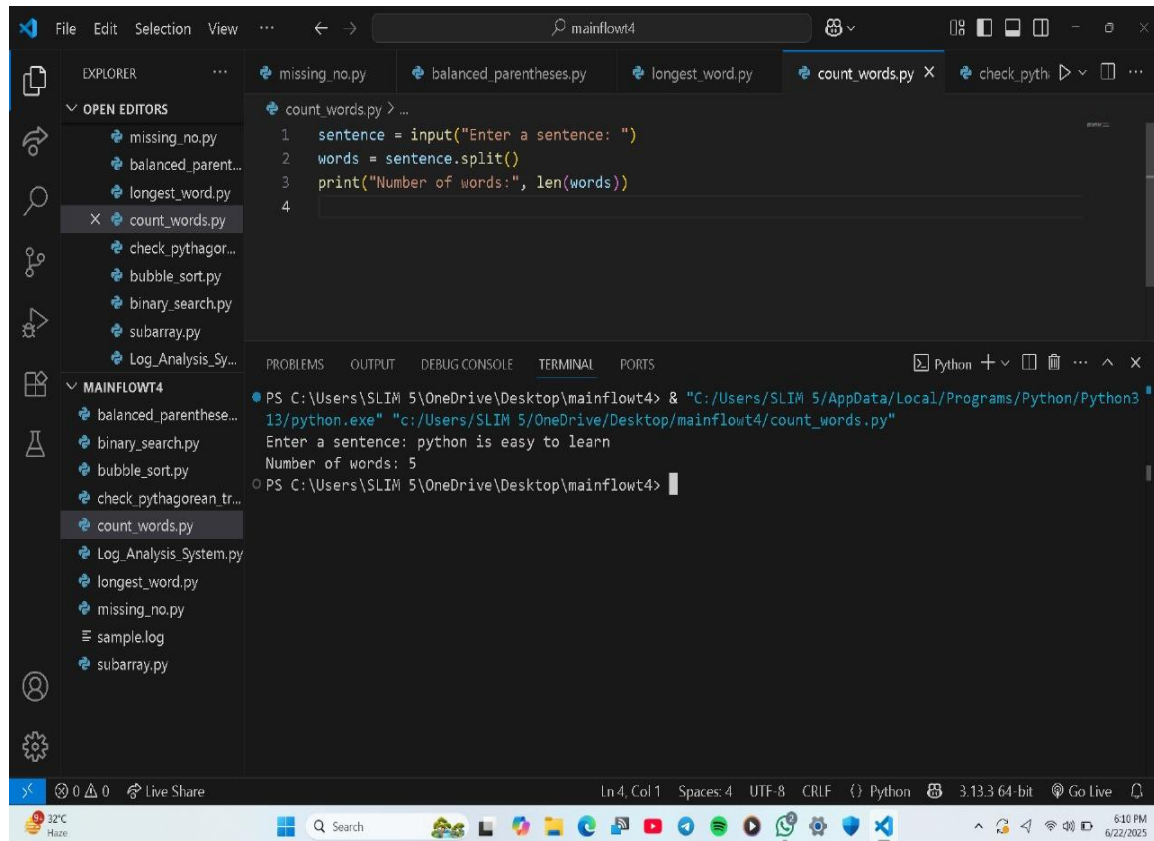
Terminal output:

```
PS C:\Users\SLIM 5\OneDrive\Desktop\mainflowt4> & "C:/Users/SLIM 5/AppData/Local/Programs/Python/Python3
13/python.exe" "c:/Users/SLIM 5/OneDrive/Desktop/mainflowt4/longest_word.py"
Enter a sentence: python is a powerful language
Longest word is: powerful
PS C:\Users\SLIM 5\OneDrive\Desktop\mainflowt4>
```

#### 4. Count Words in Sentence

Used the split() function to divide the sentence into words and counted them.

Screenshot:



The screenshot shows a Visual Studio Code editor window with a project named 'mainflowt4'. The Explorer sidebar on the left lists several Python files, with 'count\_words.py' selected. The Editor pane displays the code for 'count\_words.py':

```
1 sentence = input("Enter a sentence: ")
2 words = sentence.split()
3 print("Number of words:", len(words))
4
```

The TERMINAL pane at the bottom shows the command prompt output:

```
PS C:\Users\SLIM 5\OneDrive\Desktop\mainflowt4> & "C:/Users/SLIM 5/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/SLIM 5/OneDrive/Desktop/mainflowt4/count_words.py"
Enter a sentence: python is easy to learn
Number of words: 5
PS C:\Users\SLIM 5\OneDrive\Desktop\mainflowt4>
```

The status bar at the bottom indicates the current line is 4, column 1, with 4 spaces, UTF-8 encoding, CRLF line endings, and Python 3.13.3 64-bit.

#### 5. Check Pythagorean Triplet

Took three numbers, sorted them, and checked if the square of the largest equals the sum of squares of the other two.

Screenshot:

The screenshot shows a Visual Studio Code editor window with the file `check_pythagorean_triplet.py` open. The code is as follows:

```
1 a, b, c = map(int, input("Enter three numbers: ").split())
2 x, y, z = sorted([a, b, c])
3 if x**2 + y**2 == z**2:
4     print("Pythagorean Triplet")
5 else:
6     print("Not a Pythagorean Triplet")
7
```

The terminal at the bottom shows the execution of the script:

```
PS C:\Users\SLIM 5\OneDrive\Desktop\mainflowt4> & "C:/Users/SLIM 5/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/SLIM 5/OneDrive/Desktop/mainflowt4/check_pythagorean_triplet.py"
Enter three numbers: 5 9 8
Not a Pythagorean Triplet
PS C:\Users\SLIM 5\OneDrive\Desktop\mainflowt4> & "C:/Users/SLIM 5/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/SLIM 5/OneDrive/Desktop/mainflowt4/check_pythagorean_triplet.py"
Enter three numbers: 3 4 5
Pythagorean Triplet
PS C:\Users\SLIM 5\OneDrive\Desktop\mainflowt4>
```

## 6. Bubble Sort

Used nested loops to repeatedly swap adjacent elements if they were in the wrong order.

Screenshot:

The screenshot shows a Visual Studio Code editor window with the file `bubble_sort.py` open. The code is as follows:

```
1 arr = list(map(int, input("Enter numbers to sort: ").split()))
2 n = len(arr)
3
4 for i in range(n):
5     for j in range(0, n - i - 1):
6         if arr[j] > arr[j + 1]:
7             arr[j], arr[j + 1] = arr[j + 1], arr[j]
8
9 print("Sorted array:", arr)
10
```

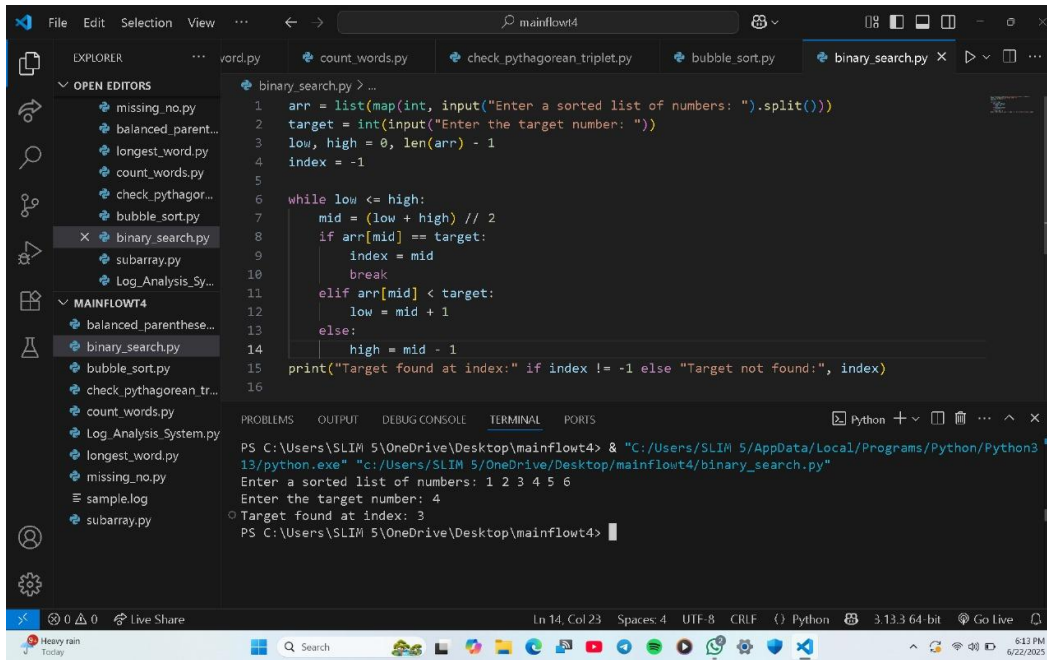
The terminal at the bottom shows the execution of the script:

```
PS C:\Users\SLIM 5\OneDrive\Desktop\mainflowt4> & "C:/Users/SLIM 5/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/SLIM 5/OneDrive/Desktop/mainflowt4/bubble_sort.py"
Enter numbers to sort: 5 8 7 6 3 4 2 1
Sorted array: [1, 2, 3, 4, 5, 6, 7, 8]
PS C:\Users\SLIM 5\OneDrive\Desktop\mainflowt4>
```

## 7. Binary Search

Implemented divide-and-conquer logic to search a target in a sorted list and return its index.

Screenshot:



```
File Edit Selection View ... mainflowt4
EXPLORER
  OPEN EDITORS
    missing_no.py
    balanced_parent...
    longest_word.py
    count_words.py
    check_pythagor...
    bubble_sort.py
    X binary_search.py
    subarray.py
    Log_Analysis_Sy...
  MAINFLOWT4
    balanced_parentese...
    binary_search.py
    bubble_sort.py
    check_pythagorean_tr...
    count_words.py
    Log_Analysis_System.py
    longest_word.py
    missing_no.py
    sample.log
    subarray.py

1 arr = list(map(int, input("Enter a sorted list of numbers: ").split()))
2 target = int(input("Enter the target number: "))
3 low, high = 0, len(arr) - 1
4 index = -1
5
6 while low <= high:
7     mid = (low + high) // 2
8     if arr[mid] == target:
9         index = mid
10        break
11    elif arr[mid] < target:
12        low = mid + 1
13    else:
14        high = mid - 1
15 print("Target found at index:" if index != -1 else "Target not found:", index)
16

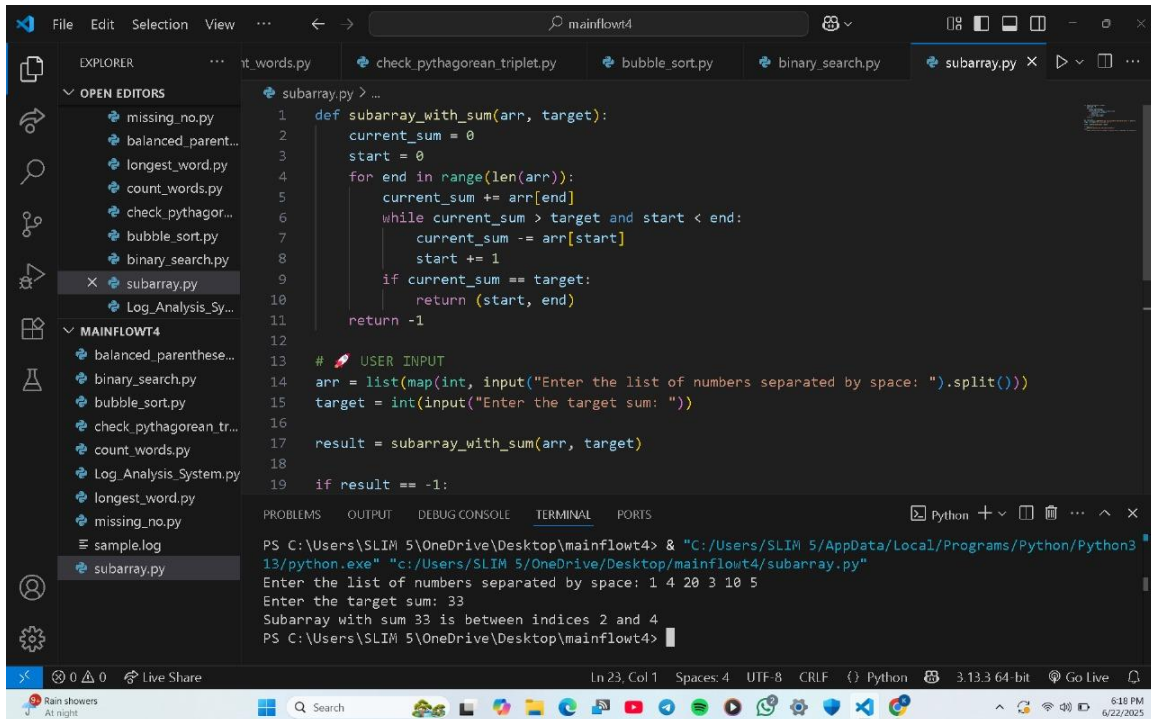
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\SLIM 5\OneDrive\Desktop\mainflowt4> & "C:/Users/SLIM 5/AppData/Local/Programs/Python/Python3
13/python.exe" "c:/Users/SLIM 5/OneDrive/Desktop/mainflowt4/binary_search.py"
Enter a sorted list of numbers: 1 2 3 4 5 6
Enter the target number: 4
Target found at index: 3
PS C:\Users\SLIM 5\OneDrive\Desktop\mainflowt4>
```

## 8. Find Subarray with Given Sum

Used sliding window technique to find the start and end indices of a subarray whose sum equals the target.

Screenshot:





## 9. Log Analysis System

Objective: Find a contiguous subarray in a list whose sum equals a target number.

- Approach: Sliding Window Technique
- Input: List of integers and a target sum (from user)
- Output: Start and end index of the subarray if found, otherwise message saying no subarray found
- Pros: Fast and memory efficient ( $O(n)$  time and  $O(1)$  space)
- Cons: Works only with non-negative numbers, returns only first match

Parsed a sample log file, extracted useful statistics like frequent IP addresses and most accessed URLs, handled possible formatting errors, and summarized data using dictionaries. Followed file-size constraint under 100 MB.

Code:

```

def subarray_with_sum(arr, target):
    current_sum = 0
    start = 0
    for end in range(len(arr)):
        current_sum += arr[end]

```

```

while current_sum > target and start < end:
    current_sum -= arr[start]
    start += 1
if current_sum == target:
    return (start, end)
return -1

```

```

arr = list(map(int, input("Enter the list of numbers separated by space: ").split()))
target = int(input("Enter the target sum: "))
result = subarray_with_sum(arr, target)

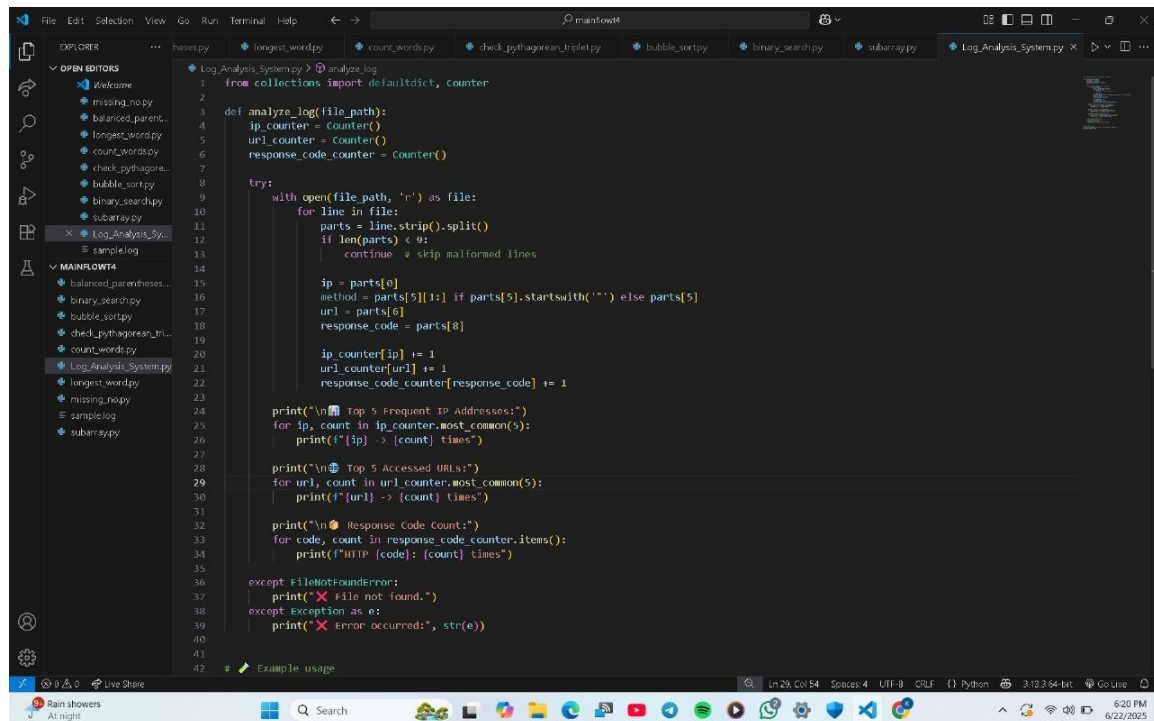
```

```

if result == -1:
    print("No subarray with the given sum found.")
else:
    print(f"Subarray with sum {target} is between indices {result[0]} and {result[1]}")

```

Screenshot:



```

Log_Analysis_System.py > analyze_log
1 from collections import defaultdict, Counter
2
3 def analyze_log(file_path):
4     ip_counter = Counter()
5     url_counter = Counter()
6     response_code_counter = Counter()
7
8     try:
9         with open(file_path, 'r') as file:
10             for line in file:
11                 parts = line.strip().split()
12                 if len(parts) < 9:
13                     continue # skip malformed lines
14
15                 ip = parts[0]
16                 method = parts[5][1:] if parts[5].startswith('(') else parts[5]
17                 url = parts[6]
18                 response_code = parts[8]
19
20                 ip_counter[ip] += 1
21                 url_counter[url] += 1
22                 response_code_counter[response_code] += 1
23
24     print("\n Top 5 Frequent IP Addresses:")
25     for ip, count in ip_counter.most_common(5):
26         print(f"{ip} -> {count} times")
27
28     print("\n Top 5 Accessed URLs:")
29     for url, count in url_counter.most_common(5):
30         print(f"{url} -> {count} times")
31
32     print("\n Response Code Count:")
33     for code, count in response_code_counter.items():
34         print(f"HTTP {code}: {count} times")
35
36 except FileNotFoundError:
37     print("❌ File not found.")
38 except Exception as e:
39     print("❌ Error occurred:", str(e))
40
41
42 # Example usage

```

The screenshot shows a VS Code editor with a Python script named `Log_Analysis_System.py` open. The script defines a function `analyze_log(file_path)` that uses `collections.defaultdict` and `Counter` to analyze a log file. The terminal output shows the results of running the script on `sample.log`.

```
1 from collections import defaultdict, Counter
2
3 def analyze_log(file_path):
4     ip_counter = Counter()
5     url_counter = Counter()
6     response_code_counter = Counter()
7
8     try:
9         with open(file_path, 'r') as file:
10             for line in file:
11                 parts = line.strip().split()
12                 if len(parts) < 9:
13                     continue # skip malformed lines
```

Terminal Output:

```
PS C:\Users\SLIM 5\OneDrive\Desktop\mainflowt4> & "C:/Users/SLIM 5/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/SLIM 5/OneDrive/Desktop/mainflowt4/Log_Analysis_System.py"
Enter path to log file (max 100MB): sample.log

Top 5 Frequent IP Addresses:
192.168.0.1 -> 2 times
192.168.0.2 -> 2 times
192.168.0.3 -> 1 times

Top 5 Accessed URLs:
/home.html -> 3 times
/login -> 1 times
/contact -> 1 times

Response Code Count:
HTTP 200: 3 times
HTTP 404: 2 times
PS C:\Users\SLIM 5\OneDrive\Desktop\mainflowt4>
```

## Challenges Faced

Parsing a log file efficiently without using external libraries required careful use of file I/O and built-in data structures. Ensuring edge cases like unbalanced brackets, or invalid log entries, were handled without crashing the program was also a key challenge.

## Learning Outcomes

This task enhanced my confidence in string processing, algorithmic problem solving, and working with file data. I also learned how to apply theoretical concepts like binary search and sorting in real-world use cases like log analysis.

## Conclusion

Successfully completing Task 4 helped solidify my Python fundamentals. The log analysis system was a particularly impactful challenge, pushing me to think about real-world file constraints and efficient data parsing.