WEEK 4 TASK

TASK 25: Find Missing Number

Steps:

1. Input a list of numbers from the user.
2. Calculate the expected sum of numbers from 1 to n+1 using the formula: Sum(n) = n(n+1)/2.
3. Calculate the actual sum of numbers in the given list.
4. The missing number is found by subtracting the actual sum from the expected sum.

Expected Input:

Enter numbers separated by space: 1 2 4 5 6

Expected Output:

Missing number is: 3

Code:

```
def find_missing_number():
    arr = list(map(int, input("Enter numbers separated by space: ").split()))
    n = len(arr) + 1
    expected_sum = n * (n + 1) // 2
    actual_sum = sum(arr)
    print("Missing number is:", expected_sum - actual_sum)

find_missing_number()
```

TASK 26: Check Balanced Parentheses

Steps:

1. Input a string of parentheses from the user.
2. Use a stack to keep track of opening parentheses.
3. Match each closing parenthesis with the top of the stack.
4. If all parentheses are matched correctly, return True, otherwise return False.

Expected Input:

Enter a string of parentheses: ({[]})

Expected Output:

True

Code:

```
def is_balanced_parentheses():
    s = input("Enter a string of parentheses: ")
    stack = []
    pairs = {')': '(', '}': '{', ']': '['}

    for char in s:
        if char in pairs.values():
            stack.append(char)
        elif char in pairs.keys():
            if not stack or stack.pop() != pairs[char]:
                print(False)
                return
    print(not stack)
```

```
is_balanced_parentheses()
```

TASK 27: Longest Word in a Sentence

Steps:

1. Input a sentence from the user.
2. Split the input sentence into words.
3. Find and return the word with the maximum length.

Expected Input:

```
Enter a sentence: The quick brown fox jumps over the lazy dog
```

Expected Output:

```
Longest word is: jumps
```

Code:

```
def longest_word():
    sentence = input("Enter a sentence: ")
    words = sentence.split()
    print("Longest word is:", max(words, key=len))

longest_word()
```

TASK 28: Count Words in a Sentence

Steps:

1. Input a sentence from the user.

2. Split the sentence using spaces.

3. Count and return the number of words.

Expected Input:

Enter a sentence: Hello world!

Expected Output:

Word count: 2

Code:

```
def count_words():
    sentence = input("Enter a sentence: ")
    print("Word count:", len(sentence.split()))

count_words()
```

TASK 29: Check Pythagorean Triplet

Steps:

1. Input three numbers from the user.

2. Sort the three numbers to ensure c is the largest.

3. Check if a^2 + b^2 = c^2.

Expected Input:

Enter three numbers separated by space: 3 4 5

Expected Output:

True

Code:

```
def is_pythagorean_triplet():
    a, b, c = map(int, input("Enter three numbers separated by space: ").split())
    a, b, c = sorted([a, b, c])
    print(a ** 2 + b ** 2 == c ** 2)

is_pythagorean_triplet()
```

TASK 30: Bubble Sort

Steps:

1. Input a list of numbers from the user.
2. Iterate through the list multiple times.
3. Swap adjacent elements if they are in the wrong order.
4. Repeat until the list is sorted.

Expected Input:

```
Enter numbers separated by space: 5 3 8 1 2
```

Expected Output:

```
Sorted list: [1, 2, 3, 5, 8]
```

Code:

```
def bubble_sort():
    arr = list(map(int, input("Enter numbers separated by space: ").split()))
    n = len(arr)
    for i in range(n):
        for j in range(0, n-i-1):
```

```
        if arr[j] > arr[j+1]:
            arr[j], arr[j+1] = arr[j+1], arr[j]
    print("Sorted list:", arr)

bubble_sort()
```

TASK 31: Binary Search

Steps:

1. Input a sorted list and a target number from the user.
2. Set left and right pointers to the start and end of the list.
3. Check the middle element.
4. Adjust the search range until the element is found or the search ends.

Expected Input:

```
Enter sorted numbers separated by space: 1 2 3 4 5 6 7 8 9
Enter target number: 5
```

Expected Output:

```
Target found at index: 4
```

Code:

```
def binary_search():
    arr = list(map(int, input("Enter sorted numbers separated by space: ").split()))
    target = int(input("Enter target number: "))
    left, right = 0, len(arr) - 1
    while left <= right:
        mid = (left + right) // 2
        if arr[mid] == target:
            print("Target found at index:", mid)
            return
        elif arr[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    print("Target not found")

binary_search()
```

TASK 32: Find Subarray with Given Sum

Expected Input:

```
Enter numbers separated by space: 1 2 3 7 5
Enter target sum: 12
```

Expected Output:

```
Subarray found between indices: (1, 3)
```

Code:

```
def find_subarray_with_sum():
    arr = list(map(int, input("Enter numbers separated by space: ").split()))
    target = int(input("Enter target sum: "))
    left, current_sum = 0, 0
    for right in range(len(arr)):
        current_sum += arr[right]
        while current_sum > target and left <= right:
            current_sum -= arr[left]
            left += 1
        if current_sum == target:
            print("Subarray found between indices:", (left, right))
            return
    print("No subarray found")

find_subarray_with_sum()
```

Log Analysis System

Steps:

1. Input a file path from the user.
2. Open the log file and read line by line.
3. Extract IP addresses, response codes, and accessed URLs.
4. Count occurrences and display the most frequent ones.

Expected Input:

```
Enter log file path: logs.txt
```

Expected Output:

```
Most Frequent IP Addresses: [('192.168.1.1', 10), ('192.168.1.2', 8)]
Most Common Response Codes: {'200': 15, '404': 3, '500': 2}
Most Accessed URLs: [('/home', 7), ('/login', 5)]
```

Code:

```
def analyze_logs():
    file_path = input("Enter log file path: ")
    ip_count = {}
    response_codes = {}
    accessed_urls = {}

    try:
        with open(file_path, 'r') as file:
            for line in file:
                parts = line.split()
                if len(parts) < 3:
                    continue
                ip, url, response_code = parts[0], parts[1], parts[2]
                ip_count[ip] = ip_count.get(ip, 0) + 1
                response_codes[response_code] = response_codes.get(response_code, 0) + 1
                accessed_urls[url] = accessed_urls.get(url, 0) + 1

        print("Most Frequent IP Addresses:", sorted(ip_count.items(), key=lambda x: x[1], reverse=True)[:5])
        print("Most Common Response Codes:", response_codes)
        print("Most Accessed URLs:", sorted(accessed_urls.items(), key=lambda x: x[1], reverse=True)[:5])
    except Exception as e:
        print("Error reading log file:", e)


analyze_logs()
```