

Python Developer Internship – Task 7 Report

Name: Khushi Maurya

Internship Title: Python Developer

Company: Main Flow Services and Technologies Pvt. Ltd.

Task: Task 7 – Problem Solving and Web Development

1. Task Overview

This task focused on solving classic algorithmic and data structure problems in Python, along with developing a real-world Flask web app: **Personal Budget Advisor**.

2. List of Problems Solved

47. Count Inversions

- **Objective:** Count how many pairs (i, j) exist where $i < j$ and $a[i] > a[j]$.
- **Technologies:** Python
- **Approach:** Modified Merge Sort
- **Input:** List of integers
- **Output:** Integer (number of inversions)

Example:

Input: [2, 4, 1, 3, 5] → Output: 3

```
def count_inversions(arr):
    if len(arr) <= 1:
        return arr, 0
    mid = len(arr) // 2
    left, inv_left = merge_sort(arr[:mid])
    right, inv_right = merge_sort(arr[mid:])
    merged, inv_split = merge_and_count(left, right)
    return merged, inv_left + inv_right + inv_split

def merge_and_count(left, right):
    result, i, j, inv = [], 0, 0, 0
    while i < len(left) and j < len(right):
        if left[i] <= right[j]:
            result.append(left[i])
            i += 1
        else:
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\SLIM 5\OneDrive\Desktop\all tasks> & "C:/Users/SLIM 5/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/SLIM 5/OneDrive/Desktop/all tasks/mainflow_task7/Count Inversions_in_an_Array.py"
Enter integers separated by space: 2 4 7 8 9 4
Number of inversions: 3
PS C:\Users\SLIM 5\OneDrive\Desktop\all tasks>

Ln 30, Col 1 Spaces: 4 UTF-8 CRLF {} Python 3.13.5 64-bit Go Live

48. Longest Palindromic Substring

- Objective:** Find the longest palindromic substring in a given string.
- Approach:** Expand around center
- Input:** A string
- Output:** Longest palindrome substring

Example:

Input: babad → Output: bab or aba

The screenshot shows the Visual Studio Code interface with the following details:

- File Menu:** File, Edit, Selection, View, ...
- Search Bar:** all tasks
- Explorer:** Shows a tree view of files and tasks. Under "OPEN EDITORS", "Longest Palindromic Substring.py" is selected. Under "ALL TASKS", "mainflow_task7" is expanded, showing its contents.
- Editor:** Displays the Python code for "Longest Palindromic Substring.py". The code defines a function `longest_palindrome` that uses a helper function `expand_center` to find the longest palindromic substring in a given string `s` by expanding around each character.
- Terminal:** Shows the command-line output of running the script. It prompts for a string input ("Enter a string: ") and prints the longest palindromic substring found ("Longest Palindromic Substring: bab").
- Status Bar:** Shows the current file is "Python", line 18, column 1, spaces: 4, encoding: UTF-8, CRLF, Python 3.13.5 64-bit, and Go Live.

49. Traveling Salesman Problem (TSP)

- **Objective:** Find the shortest path visiting each city once and returning to the start.
- **Technologies:** Python
- **Approach:** Dynamic programming
- **Input:** Distance matrix
- **Output:** Shortest route and distance

✓ Example Output:

Best Route: [0, 1, 3, 2]

Minimum Distance: 80

The screenshot shows a code editor interface with several tabs open. The main tab displays Python code for solving the Traveling Salesman Problem using brute force permutations:

```

1  from itertools import permutations
2
3  def tsp_brute_force(distances):
4      cities = list(range(len(distances)))
5      min_distance = float('inf')
6      best_route = []
7
8      for perm in permutations(cities):
9          distance = 0
10         for i in range(len(perm)):
11             distance += distances[perm[i]][perm[(i + 1) % len(cities)]]
12             if distance < min_distance:
13                 min_distance = distance
14                 best_route = perm
15
16     return list(best_route), min_distance

```

Below the code editor, the terminal window shows the execution of the script:

```

PS C:\Users\SLIM 5\OneDrive\Desktop\all tasks> & "C:/Users/SLIM 5/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/SLIM 5/OneDrive/Desktop/all tasks/mainflow_task7/Traveling Salesman Problem.py"
Enter number of cities: 3
Enter distances from city 0 (space separated): 4 5 6
Enter distances from city 1 (space separated): 4 7 5
Enter distances from city 2 (space separated): 1 2 5
Best Route: [0, 1, 2]
Minimum Distance: 11
PS C:\Users\SLIM 5\OneDrive\Desktop\all tasks>

```

50. Graph Cycle Detection

- Objective:** Detect if an undirected graph contains a cycle.
- Approach:** Depth-First Search (DFS)
- Input:** Adjacency list
- Output:** True / False

Example:

Graph with cycle → True

Graph without cycle → False

The screenshot shows the Microsoft Visual Studio Code interface. In the Explorer sidebar, there are several projects and files listed under 'OPEN EDITORS' and 'ALL TASKS'. The 'Cycle Detection in Graph.py' file is currently active in the editor. The code implements a Depth-First Search (DFS) algorithm to detect cycles in a graph represented as a dictionary of neighbors. The terminal window at the bottom shows the execution of the script, where the user enters the number of nodes (3), edges (3), and specific edges (0 1, 1 2, 2 0). The output indicates that the graph contains a cycle.

```
mainflow_task7 > Cycle Detection in Graph.py > ...
1 def has_cycle(graph):
2     visited = set()
3
4     def dfs(v, parent):
5         visited.add(v)
6         for neighbor in graph[v]:
7             if neighbor not in visited:
8                 if dfs(neighbor, v):
9                     return True
10            elif neighbor != parent:
11                return True
12        return False
13
14    for node in graph:
15        if node not in visited:
16            if dfs(node, None):
17                print("Graph contains cycle: True")
18
19 PS C:\Users\SLIM 5\OneDrive\Desktop\all tasks> & "C:/Users/SLIM 5/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/SLIM 5/OneDrive/Desktop/all tasks/mainflow_task7/Cycle Detection in Graph.py"
Enter number of nodes: 3
Enter number of edges: 3
Enter edge (u v): 0 1
Enter edge (u v): 1 2
Enter edge (u v): 2 0
Graph contains cycle: True
PS C:\Users\SLIM 5\OneDrive\Desktop\all tasks>
```

51. Longest Substring Without Repeating Characters

- **Objective:** Find the length of the longest substring without repeating characters.
- **Approach:** Sliding Window
- **Input:** A string
- **Output:** Integer length

✓ Example:

Input: abcabcbb → Output: 3 (abc)

The screenshot shows a code editor interface with several tabs open. The tabs include 'Long Salesman Problem.py', 'Cycle Detection in Graph.py', and 'Longest Substring Without Repeating Characters.py'. The 'Longest Substring Without Repeating Characters.py' tab is active, displaying Python code for finding the length of the longest substring without repeating characters. The code uses a sliding window approach with a dictionary to track character positions. The terminal window below shows the execution of the script and its output for the input 'abcabbbg'.

```

def length_of_longest_substring(s):
    start = 0
    max_len = 0
    seen = {}
    for i, char in enumerate(s):
        if char in seen and seen[char] >= start:
            start = seen[char] + 1
        seen[char] = i
        max_len = max(max_len, i - start + 1)
    return max_len

# User Input
s = input("Enter a string: ")
print("Length of longest substring without repeating characters:", length_of_longest_substring(s))

```

```

PS C:\Users\SLIM 5\OneDrive\Desktop\all tasks> & "C:/Users/SLIM 5/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/SLIM 5/OneDrive/Desktop/all tasks/mainflow_task7/Longest Substring Without Repeating Characters.py"
Enter a string: abcabbbg
Length of longest substring without repeating characters: 3
PS C:\Users\SLIM 5\OneDrive\Desktop\all tasks>

```

52. Valid Parentheses Combinations

- Objective:** Generate all valid combinations of n pairs of parentheses.
- Approach:** Backtracking
- Input:** Integer n
- Output:** List of valid strings

Example:

Input: n = 3 → Output: ['((()))', '(()())', '(())()', '()()()', '()()()']

```
def generate_parentheses(n):
    result = []

    def backtrack(current, open_count, close_count):
        if len(current) == 2 * n:
            result.append(current)
            return
        if open_count < n:
            backtrack(current + "(", open_count + 1, close_count)
        if close_count < open_count:
            backtrack(current + ")", open_count, close_count + 1)

    backtrack("", 0, 0)
    return result

# User Input
n = int(input("Enter number of parentheses pairs: "))
print("Valid combinations:", generate_parentheses(n))
```

3/python.exe" "c:/Users/SLIM 5/OneDrive/Desktop/all tasks/mainflow_task7/Valid_Parentheses_Combinations.py"
Enter number of parentheses pairs: 3
Valid combinations: [']([())', '(()()', '(()()()', '(()())', '(()())()']
PS C:\Users\SLIM 5\OneDrive\Desktop\all tasks>

53. Zigzag Level Order Traversal of Binary Tree

- **Objective:** Traverse a binary tree in zigzag (left-right then right-left) level order.
- **Approach:** Breadth-first traversal using two stacks
- **Input:** Binary tree
- **Output:** List of levels in zigzag

Output Example:

[[1], [3, 2], [4, 5, 6]]

The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, ..., all tasks
- Editor:** Zigzag Level Order Traversal.py
- Explorer:** Shows various Python files under 'OPEN EDITORS' and 'ALL TASKS'.
- Terminal:** Shows the execution of the script and its output. The script defines a function `build_tree` that reads root values and creates a tree structure. The terminal shows the user interacting with the script to build a tree with root value 3, left child 6, and right child 6.
- Status Bar:** Ln 38, Col 1, Spaces: 4, UTF-8, CRLF, Python 3.13.5 64-bit, Go Live

```
def build_tree():
    val = input("Enter root value (or empty to stop): ")
    if not val:
        return None
    node = TreeNode(int(val))
    print(f"Enter left child of {val}:")
    node.left = build_tree()
    print(f"Enter right child of {val}:")
    node.right = build_tree()
    return node
```

```
Enter root value (or empty to stop): 3
Enter left child of 3:
Enter root value (or empty to stop):
Enter right child of 3:
Enter root value (or empty to stop): 6
Enter left child of 6:
Enter root value (or empty to stop): 6
Enter root value (or empty to stop): 6
Enter root value (or empty to stop): 6
Enter left child of 6:
Enter root value (or empty to stop):
Enter right child of 6:
Enter root value (or empty to stop):
Zigzag Traversal: [[1], [3, 2], [4, 5, 6]]
```

54. Palindrome Partitioning

- Objective:** Partition a string such that each substring is a palindrome.
- Approach:** Backtracking
- Input:** A string
- Output:** List of lists

✓ Example:

Input: aab → Output: [['a', 'a', 'b'], ['aa', 'b']]

```

File Edit Selection View ... ← → all tasks
EXPLORER ... Valid_Parentheses_Combinations.py Zigzag Level Order Traversal.py Palindrome Partitioning.py ...
OPEN EDITORS
Count Inversions... Longest Palindro... Traveling Salesm... Cycle Detection i... Longest Substrin... Valid_Parenthes... Zigzag Level Ord...
X Palindrome Part... Personal Budget ...
ALL TASKS
MainFlow_task1
mainflow_task7
Count Inversions.in_... Cycle Detection in G... Longest Palindromic... Longest Substring ... Palindrome Partition...
Personal Budget Ad... Traveling Salesman ... Valid_Parentheses_C... Zigzag Level Order T...
mainflow4
mainflowtask2
mainflowtask3
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python + ... ×
PS C:\Users\SLIM 5\OneDrive\Desktop\all tasks> & "C:/Users/SLIM 5/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/SLIM 5/OneDrive/Desktop/all tasks/mainflow_task7/Palindrome Partitioning.py"
Enter a string: aab
Palindrome Partitions: [['a', 'a', 'b'], ['aa', 'b']]
PS C:\Users\SLIM 5\OneDrive\Desktop\all tasks>

```

3. Personal Budget Advisor Web App (Flask Project)

◆ **Objective:**

To build a web-based budget advisor tool using Flask that allows users to enter income and expenses, calculates total spending, shows savings, and gives suggestions.

◆ **Technologies Used:**

- Python 3.13
- Flask Framework
- HTML, CSS (via static/style.css)
- Jinja2 Templates

◆ **Features:**

- Input form for income and expenses
- Calculates expense breakdown (amount + %)
- Computes total savings
- Gives suggestions: save more / good job
- Clean and modern UI

Screenshots:

The screenshot shows the Visual Studio Code interface. The left sidebar displays a tree view of open editors and tasks. The main editor area shows the content of index.html, which is part of a project structure under personal_budget_advisor/templates. The code is an HTML form for a Personal Budget Advisor, with CSS styles applied via a static file.

```

<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}>
<title>Personal Budget Advisor</title>
<style>
    body { font-family: Arial; background: #f3f3f3; padding: 30px; }
    .container { max-width: 500px; margin: auto; background: white; padding: 20px; border: 1px solid #ccc; }
    input[type=number] { width: 100%; padding: 10px; margin: 8px 0; }
    input[type=submit] { background-color: #4CAF50; color: white; padding: 10px; border: none; }
</style>
</head>
<body>
    <div class="container">
        <h2>Personal Budget Advisor</h2>
        <form action="/result" method="post">
            <label>Total Monthly Income:</label>
            <input type="number" name="income" step="0.01" required>
            <label>Rent:</label>
            <input type="number" name="rent" step="0.01" required>
            <label>Food:</label>
            <input type="number" name="food" step="0.01" required>
            <label>Transport:</label>
            <input type="number" name="transport" step="0.01" required>
            <label>Entertainment:</label>
            <input type="number" name="entertainment" step="0.01" required>
        </form>
    </div>
</body>

```

The image contains two side-by-side screenshots of a web application titled "Personal Budget Advisor". Both screenshots show the same form with different values entered in the input fields.

Left Screenshot (Initial State):

- Total Monthly Income: (empty input field)
- Rent: (empty input field)
- Food: (empty input field)
- Transport: (empty input field)
- Entertainment: (empty input field)
- Others: (empty input field)
- Analyze Budget button (green button at the bottom)

Right Screenshot (After Input):

- Total Monthly Income: 100000
- Rent: 20000
- Food: 10000
- Transport: 3000
- Entertainment: 2000
- Others: 5000 (highlighted with an orange border)
- Analyze Budget button (green button at the bottom)

Budget Summary

Total Monthly Income: ₹100000.0

Category	Amount (₹)	Percent (%)
Rent	20000.0	20.0
Food	10000.0	10.0
Transport	3000.0	3.0
Entertainment	2000.0	2.0
Others	5000.0	5.0

Total Expenses: ₹40000.0

Total Savings: ₹60000.0

 **Good job on your savings!**

Personal Budget Advisor

Total Monthly Income:

100000

Rent:

40000

Food:

20000

Transport:

10000

Entertainment:

10000

Others:

40000

Analyze Budget

Budget Summary

Total Monthly Income: ₹100000.0

Category	Amount (₹)	Percent (%)
Rent	40000.0	40.0
Food	20000.0	20.0
Transport	10000.0	10.0
Entertainment	10000.0	10.0
Others	40000.0	40.0

Total Expenses: ₹120000.0

Total Savings: ₹-20000.0

 Try to cut down your expenses by at least 10%.

5. Pros and Cons

Pros:

- All solutions implemented manually using core Python (no pandas/numpy).

- Efficient logic using optimal algorithms (merge sort, DFS, DP).
- Clean and interactive Flask UI.
- Great real-world project for resume and portfolio.

Cons:

- No persistent database for saving data (can be improved).
 - No charts or data visualization (could be added).
 - Budget app doesn't store history.
-

6. Conclusion

This task helped in mastering:

- Real-world problem solving with Python
- Web development with Flask and HTML/CSS
- Core concepts of algorithms, backtracking, dynamic programming
- Frontend/backend integration

It also gave a practical project (budget advisor) that can be extended into a complete finance management app.