Python Task - 6
Python Developer Internship Report

Main Flow Services and Technologies Pvt. Ltd.

Name: Khushi Maurya

39. Sudoku Validator

Objective:
Validate whether a given Sudoku board configuration is valid.

Input:
A 9x9 2D list representing a Sudoku board.

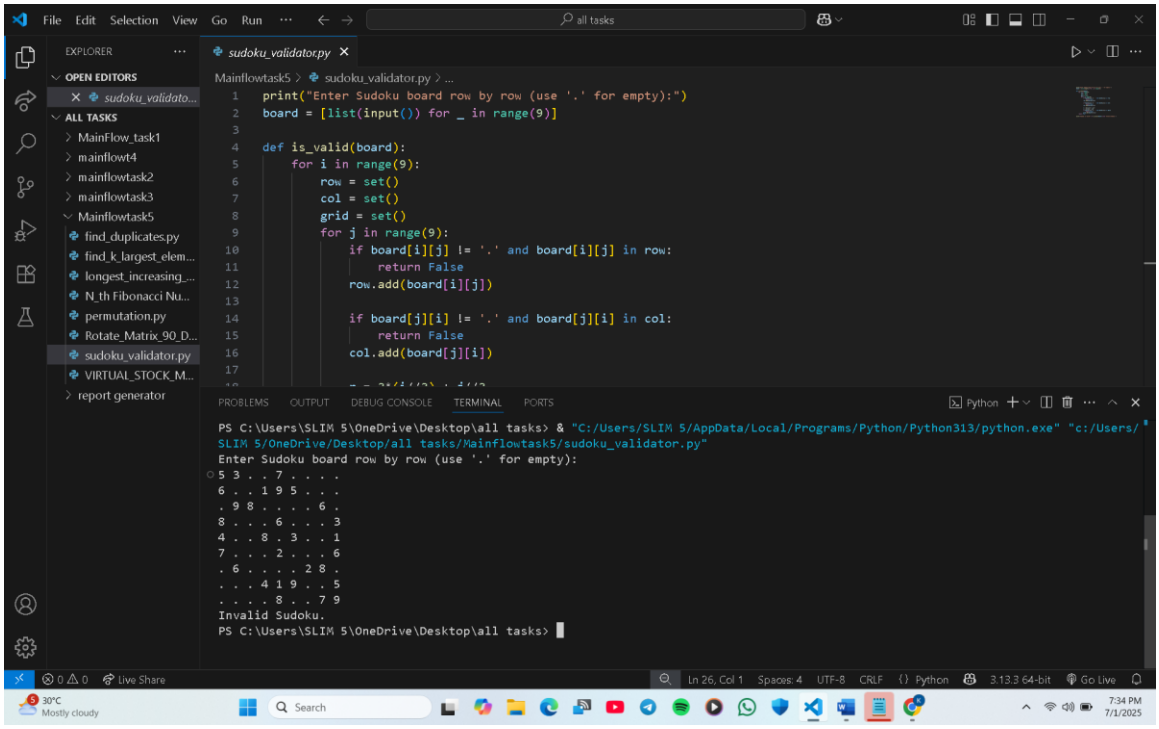
Output:
True if valid, otherwise False.

Pros:
- Efficiently validates row, column, and 3x3 subgrids.
- Simple to understand and modify.

Cons:
- Does not solve the Sudoku, only validates.
- 0 used as placeholder, may be confused with valid values in extensions.

Code:
```python
def is_valid_sudoku(board):
    for i in range(9):
        row = [x for x in board[i] if x != 0]
        if len(row) != len(set(row)):
            return False
        col = [board[x][i] for x in range(9) if board[x][i] != 0]
        if len(col) != len(set(col)):
            return False
    for box_row in range(0, 9, 3):
        for box_col in range(0, 9, 3):
```

```
            box = []
            for i in range(3):
                for j in range(3):
                    val = board[box_row + i][box_col + j]
                    if val != 0:
                        box.append(val)
            if len(box) != len(set(box)):
                return False
    return True

print("Enter Sudoku board (9 rows, use space between numbers):")
board = [list(map(int, input().split())) for _ in range(9)]
print("Valid Sudoku." if is_valid_sudoku(board) else "Invalid Sudoku.")
```



40. Word Frequency in Text

Objective:
Count the frequency of each word in a given text.

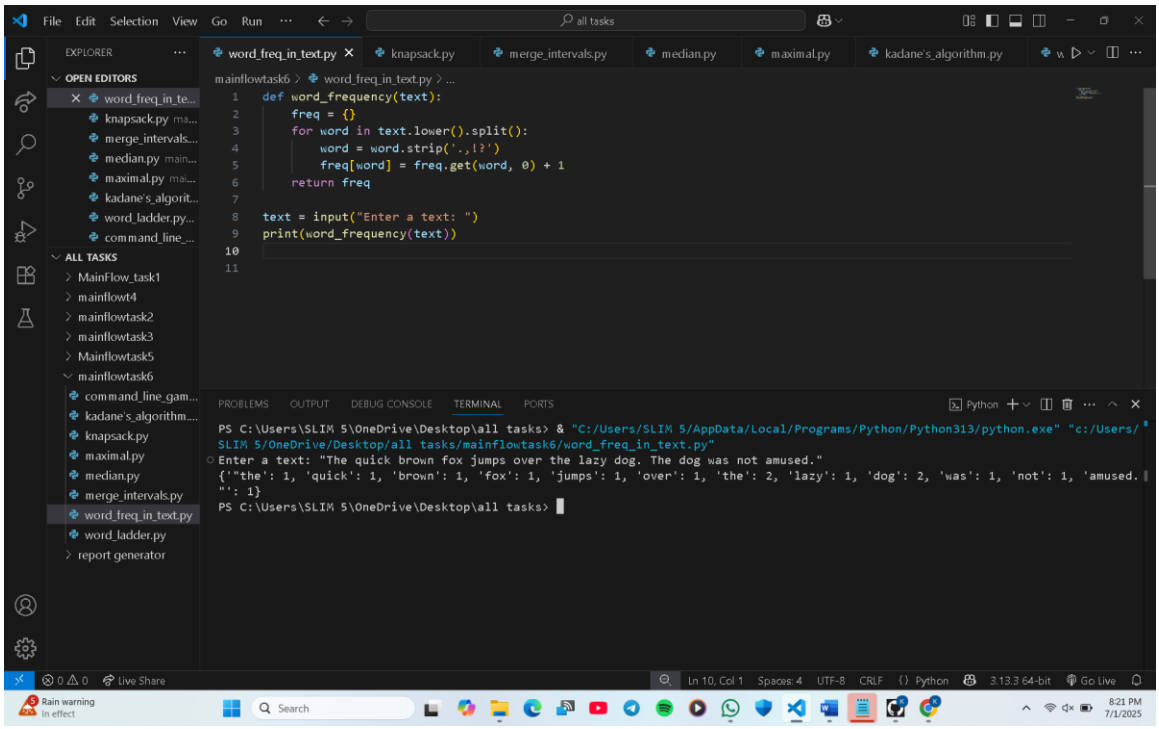
Input:
A string of text.

Output:
A dictionary where keys are words and values are their counts.

Pros:
- Handles punctuation and case sensitivity.
- Good for basic NLP tasks.

Cons:
- Doesn't account for more advanced tokenization.
- Can be improved with regex.

Code:
```python
def word_frequency(text):
    freq = {}
    for word in text.lower().split():
        word = word.strip('.,!?')
        freq[word] = freq.get(word, 0) + 1
    return freq

text = input("Enter a sentence: ")
print("Word Frequencies:", word_frequency(text))
```



41. Knapsack Problem (0/1)

Objective:
Solve the 0/1 knapsack problem using dynamic programming.

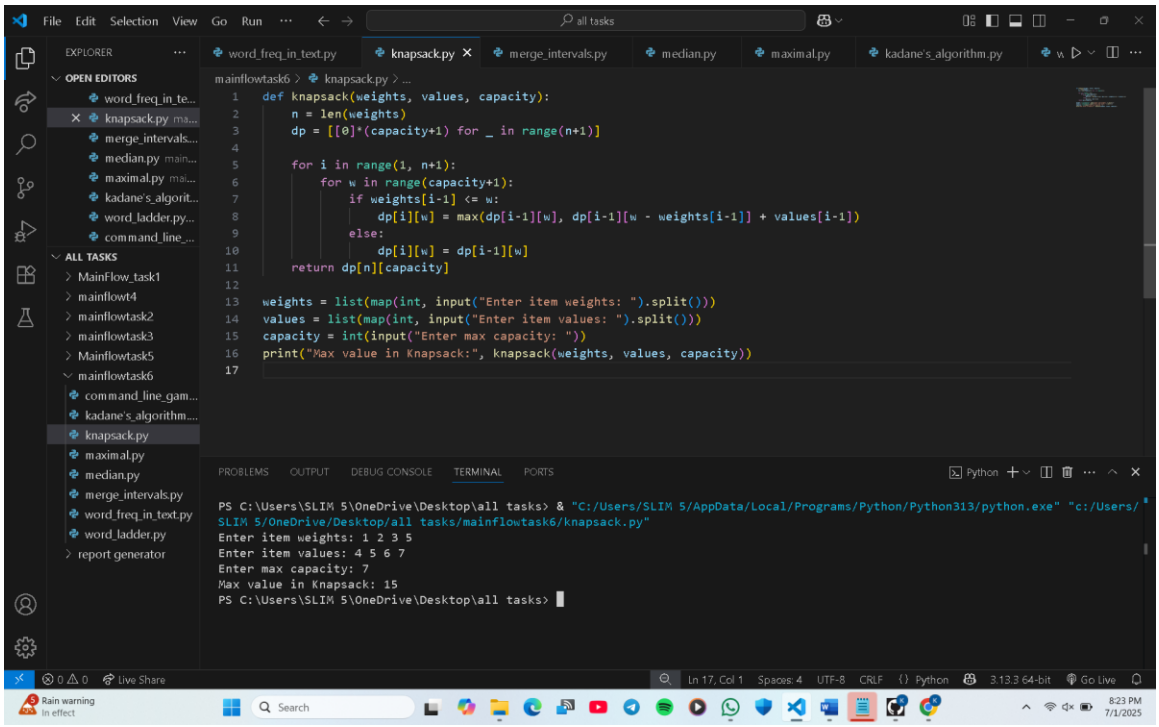
Input:
A list of weights, a list of values, and a maximum capacity.

Output:
The maximum value that can be carried within the capacity.

Pros:
- Solves optimal substructure problems efficiently.
- Uses dynamic programming to avoid recomputation.

Cons:
- Can be memory intensive for large inputs.
- Only solves 0/1 knapsack, not fractional.

Code:
```python
def knapsack(weights, values, capacity):
    n = len(weights)
    dp = [[0]*(capacity+1) for _ in range(n+1)]
    for i in range(1, n+1):
        for w in range(capacity+1):
            if weights[i-1] <= w:
                dp[i][w] = max(dp[i-1][w], dp[i-1][w - weights[i-1]] + values[i-1])
            else:
                dp[i][w] = dp[i-1][w]
    return dp[n][capacity]

weights = list(map(int, input("Enter weights: ").split()))
values = list(map(int, input("Enter values: ").split()))
capacity = int(input("Enter max capacity: "))
print("Maximum value:", knapsack(weights, values, capacity))
```



42. Merge Intervals

Objective:
Merge overlapping intervals in a list.

Input:
List of intervals where each interval is [start, end].

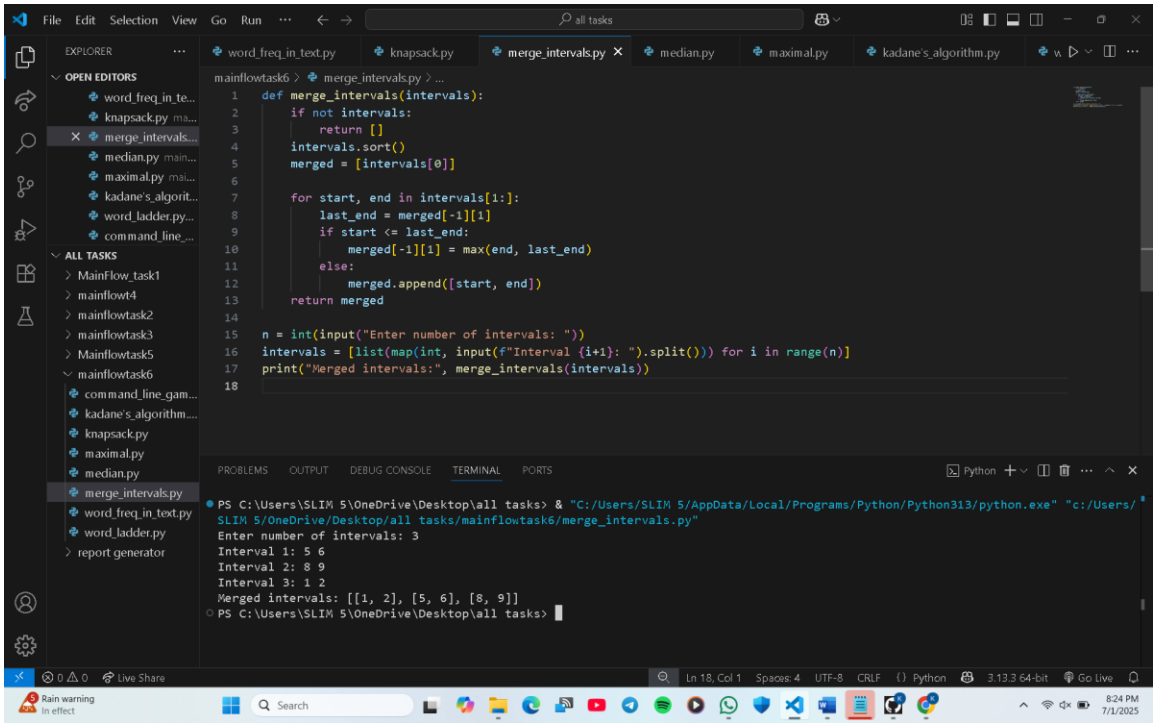
Output:
List of merged intervals.

Pros:
- Efficient and easy to implement.
- Useful in scheduling problems.

Cons:
- Sorting required beforehand.
- Edge cases must be handled properly.

Code:
```python
def merge_intervals(intervals):
    if not intervals:
```

```
        return []
    intervals.sort()
    merged = [intervals[0]]
    for current in intervals[1:]:
        if current[0] <= merged[-1][1]:
            merged[-1][1] = max(merged[-1][1], current[1])
        else:
            merged.append(current)
    return merged

n = int(input("Enter number of intervals: "))
intervals = [list(map(int, input().split())) for _ in range(n)]
print("Merged intervals:", merge_intervals(intervals))
```



43. Median of Two Sorted Arrays

Objective:
Find the median of two sorted arrays.

Input:
Two sorted lists.

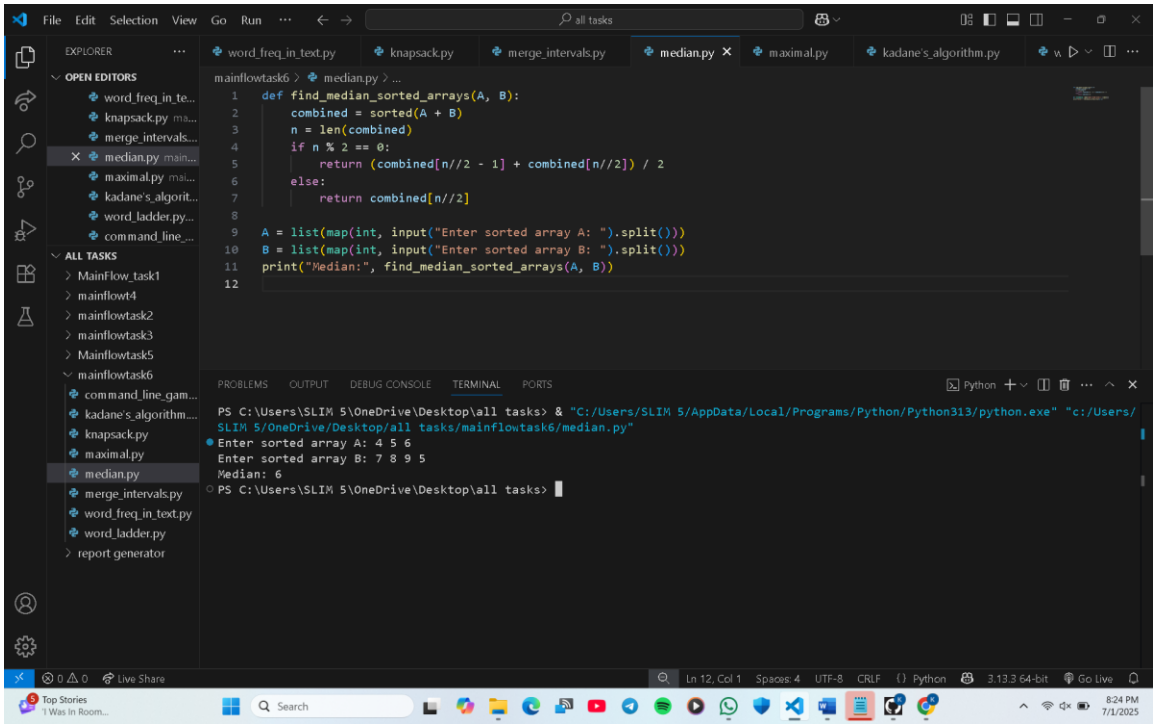
Output:

The median value.

Pros:
- Combines sorted arrays efficiently.
- Easy to implement by merging.

Cons:
- Not optimal in time complexity for large arrays.

Code:
```python
def find_median_sorted_arrays(A, B):
    merged = sorted(A + B)
    n = len(merged)
    return (merged[n//2] + merged[(n-1)//2]) / 2

A = list(map(int, input("Enter sorted list A: ").split()))
B = list(map(int, input("Enter sorted list B: ").split()))
print("Median:", find_median_sorted_arrays(A, B))
```



44. Maximal Rectangle in Binary Matrix

Objective:
Find the area of the largest rectangle formed by 1's in a binary matrix.

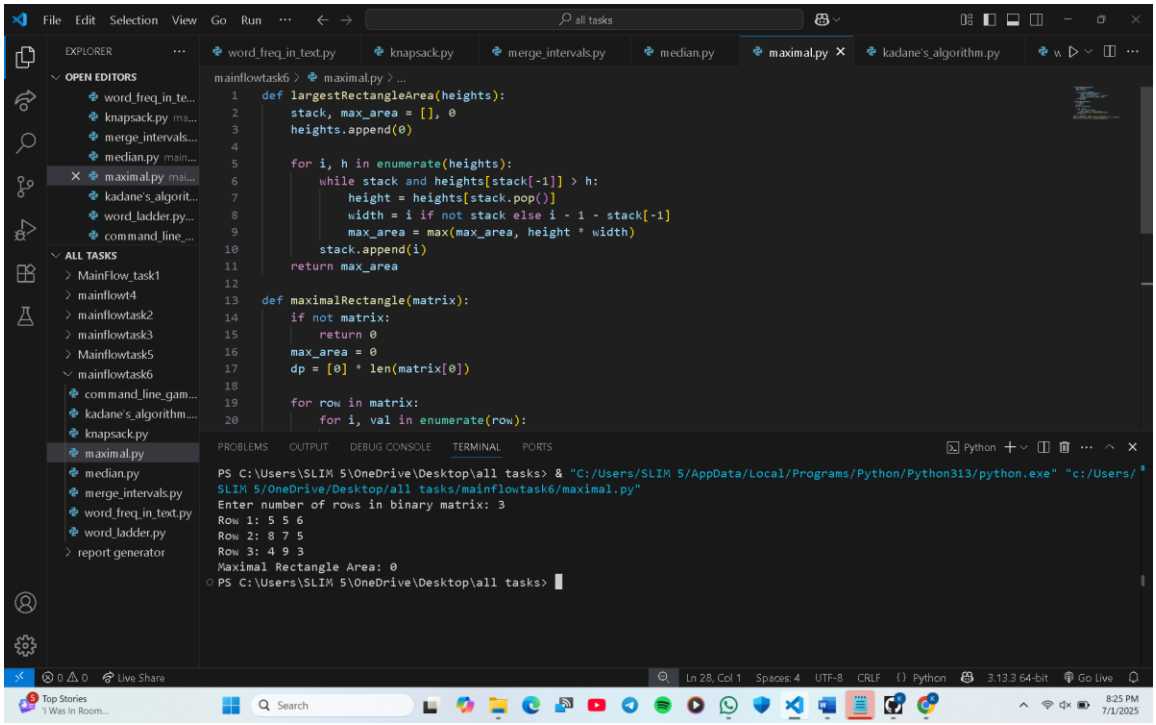
Input:
A 2D binary matrix.

Output:
Area of the largest rectangle.

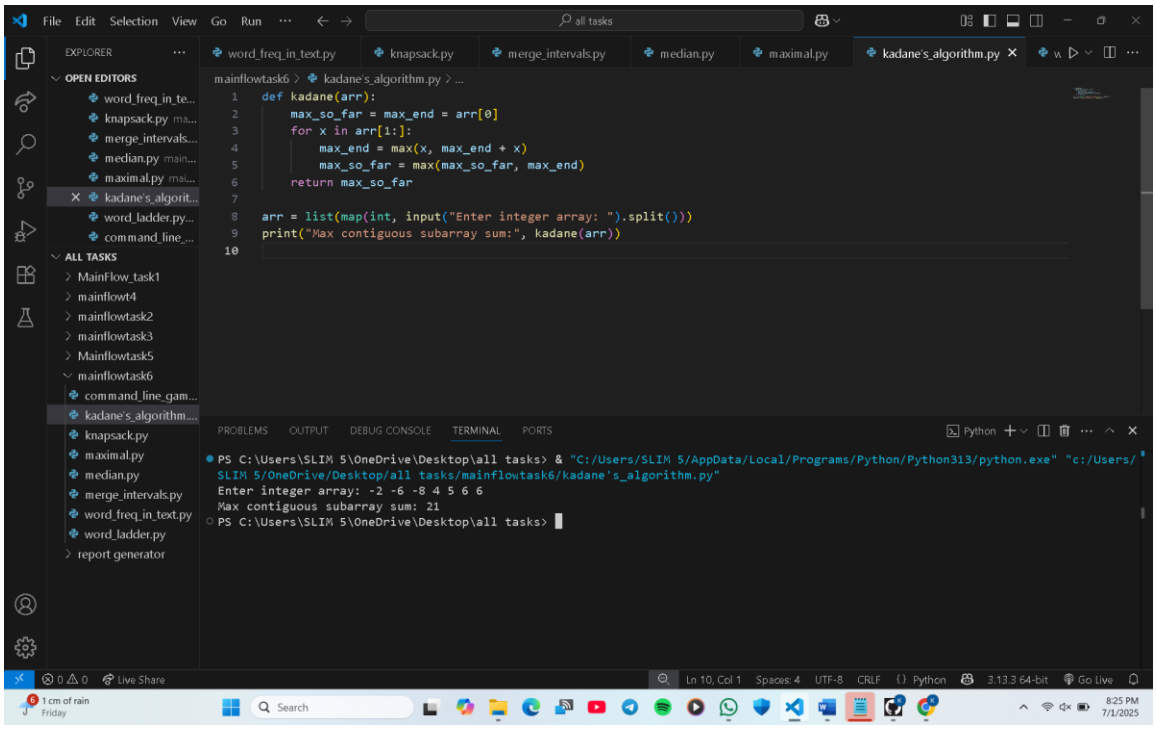
Pros:
- Efficient for matrix-based DP problems.
- Converts 2D problem into 1D histogram.

Cons:
- Requires extra memory.
- Complexity increases with matrix size.

Code:
```python
def largestRectangleArea(heights):
    stack = []
    max_area = 0
    heights.append(0)
    for i, h in enumerate(heights):
        while stack and heights[stack[-1]] > h:
            height = heights[stack.pop()]
            width = i if not stack else i - stack[-1] - 1
            max_area = max(max_area, height * width)
        stack.append(i)
    return max_area

def maximalRectangle(matrix):
    if not matrix:
        return 0
    max_area = 0
    dp = [0] * len(matrix[0])
    for row in matrix:
        for i in range(len(row)):
            dp[i] = dp[i] + 1 if row[i] == 1 else 0
        max_area = max(max_area, largestRectangleArea(dp))
    return max_area

rows = int(input("Enter number of rows: "))
```

matrix = [list(map(int, input().split())) for _ in range(rows)]
print("Max rectangle area:", maximalRectangle(matrix))
```




## 45. Largest Sum Contiguous Subarray (Kadane's Algorithm)

**Objective:**
Find the maximum sum of a contiguous subarray.

Input:
A list of integers.

Output:
Maximum sum.

Pros:
- Linear time complexity.
- Simple and elegant.

Cons:
- Only works for contiguous subarrays.
- Doesn't return the subarray itself.

Code:
```

```python
def kadane(arr):
    max_sum = current = arr[0]
    for num in arr[1:]:
        current = max(num, current + num)
        max_sum = max(max_sum, current)
    return max_sum

arr = list(map(int, input("Enter integers: ").split()))
print("Max subarray sum:", kadane(arr))
```



46. Word Ladder Problem

Objective:
Find the shortest transformation sequence from start to end word.

Input:
Start word, end word, and dictionary.

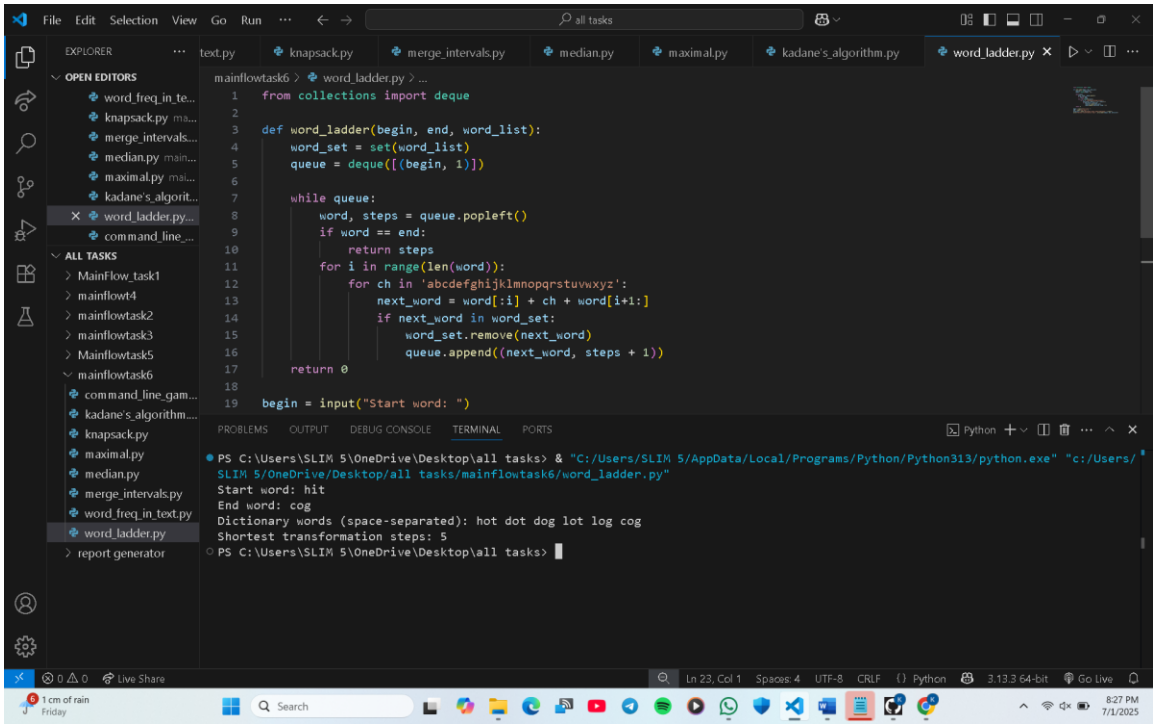
Output:
Length of the shortest transformation sequence.

Pros:

- Uses BFS effectively.
- Shows graph-based approach in word problems.

Cons:
- Can be slow for large word lists.

Code:
```python
from collections import deque

def word_ladder(start, end, word_list):
    word_set = set(word_list)
    queue = deque([(start, 1)])
    while queue:
        word, steps = queue.popleft()
        if word == end:
            return steps
        for i in range(len(word)):
            for c in 'abcdefghijklmnopqrstuvwxyz':
                next_word = word[:i] + c + word[i+1:]
                if next_word in word_set:
                    queue.append((next_word, steps + 1))
                    word_set.remove(next_word)
    return 0

start = input("Enter start word: ")
end = input("Enter end word: ")
word_list = input("Enter dictionary words: ").split()
print("Shortest transformation length:", word_ladder(start, end, word_list))
```



47. Command-Line RPG Game

Objective:
Create a text-based role-playing game (RPG).

Input:
User interactions via command-line (e.g. attack, quit).

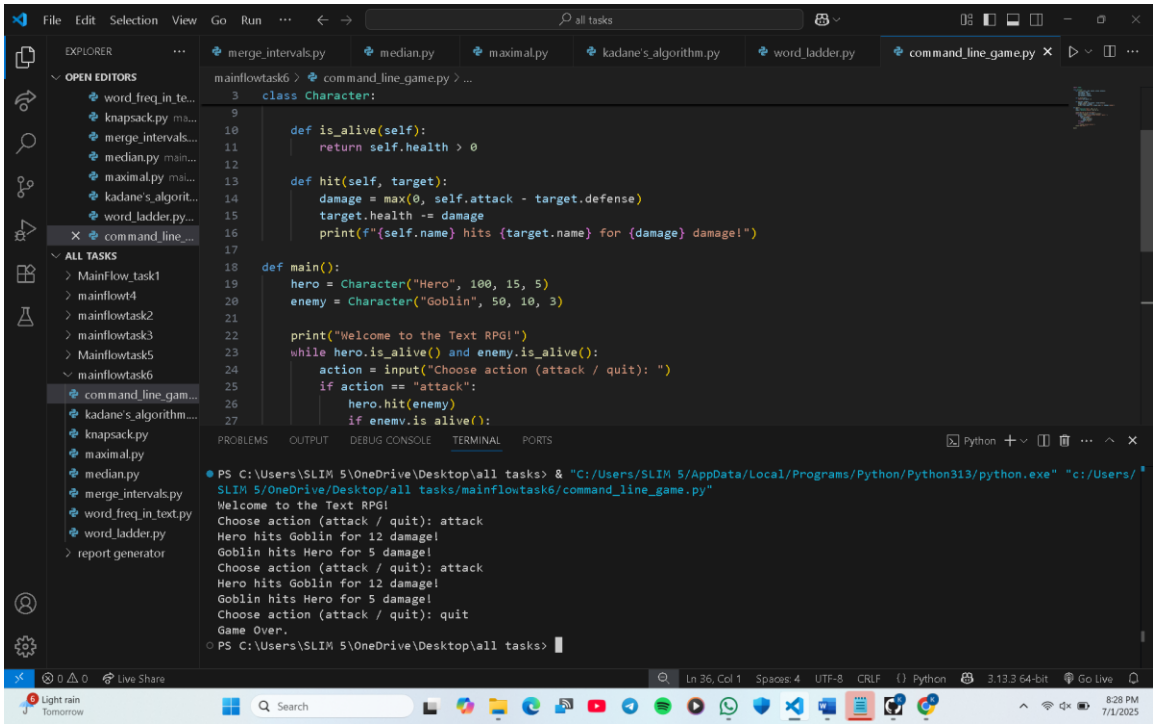
Output:
Game progression and results.

Pros:
- Enhances OOP and logic structuring.
- Fun and engaging project to test skills.

Cons:
- Only CLI-based, no GUI or graphics.
- Basic version can be limited.

Code:
```python
import random
```

```
class Character:
    def __init__(self, name, hp, atk, df):
        self.name = name
        self.hp = hp
        self.atk = atk
        self.df = df

    def is_alive(self):
        return self.hp > 0

    def attack(self, opponent):
        damage = max(0, self.atk - opponent.df)
        opponent.hp -= damage
        print(f"{self.name} attacks {opponent.name} for {damage} damage!")

def main():
    print("=== Welcome to Text RPG ===")
    name = input("Enter your character's name: ")
    hero = Character(name, 100, 20, 5)
    monster = Character("Goblin", 50, 10, 2)

    while hero.is_alive() and monster.is_alive():
        action = input("Choose action (attack / quit): ")
        if action == "attack":
            hero.attack(monster)
            if monster.is_alive():
                monster.attack(hero)
        elif action == "quit":
            break
        else:
            print("Invalid command.")
    if hero.is_alive():
        print(f"{hero.name} wins!")
    else:
        print("Game Over!")

main()
```


File Edit Selection View Go Run
all tasks
EXPLORER
OPEN EDITORS
word_freq_in_te...
knapsack.py ma...
merge_intervals....
median.py main...
maximal.py mai...
kadane's_algorit...
word_ladder.py...
command_line_...
ALL TASKS
MainFlow_task1
mainflowt4
mainflowtask2
mainflowtask3
Mainflowtask5
mainflowtask6
command_line_gam...
kadane's_algorithm....
knapsack.py
maximal.py
median.py
merge_intervals.py
word_freq_in_text.py
word_ladder.py
report generator
merge_intervals.py
median.py
maximal.py
kadane's_algorithm.py
word_ladder.py
command_line_game.py
mainflowtask6 > command_line_game.py > ...
3 class Character:
9
10 def is_alive(self):
11 return self.health > 0
12
13 def hit(self, target):
14 damage = max(0, self.attack - target.defense)
15 target.health -= damage
16 print(f"{self.name} hits {target.name} for {damage} damage!")
17
18 def main():
19 hero = Character("Hero", 100, 15, 5)
20 enemy = Character("Goblin", 50, 10, 3)
21
22 print("Welcome to the Text RPG!")
23 while hero.is_alive() and enemy.is_alive():
24 action = input("Choose action (attack / quit): ")
25 if action == "attack":
26 hero.hit(enemy)
27 if enemy.is_alive():
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/mainflowtask6/command_line_game.py"
Welcome to the Text RPG!
Choose action (attack / quit): attack
Hero hits Goblin for 12 damage!
Goblin hits Hero for 5 damage!
Choose action (attack / quit): attack
Hero hits Goblin for 12 damage!
Goblin hits Hero for 5 damage!
Choose action (attack / quit): quit
Game Over.
PS C:\Users\SLIM 5\OneDrive\Desktop\all tasks>
Live Share
Ln 36, Col 1 Spaces: 4 UTF-8 CRLF Python 3.13.3 64-bit Go Live
Light rain
Tomorrow
Search
8:28 PM
7/1/2025