```python
#39. Sudoku Validator
def print_sudoku_board(board):
    print("\nSudoku Board:")
    for i, row in enumerate(board):
        # Add horizontal box separators
        if i % 3 == 0 and i != 0:
            print("-" * 21)

        for j, value in enumerate(row):
            # Add vertical box separators
            if j % 3 == 0 and j != 0:
                print("|", end=" ")
            print(value, end=" ")
        print()

def is_valid_sudoku(board):
    def is_valid_unit(unit):
        unit = [i for i in unit if i != '.']
        return len(unit) == len(set(unit))

    # Check rows, columns, and 3x3 grids
    for row in board:
        if not is_valid_unit(row):
            return False

    for col in zip(*board):
        if not is_valid_unit(col):
            return False

    for i in (0, 3, 6):
        for j in (0, 3, 6):
            grid = [board[x][y] for x in range(i, i+3) for y in
range(j, j+3)]
            if not is_valid_unit(grid):
                return False

    return True

# Input
# Input with validation
board = []
print("Enter the Sudoku board row by row (use '.' for empty cells):")
for i in range(9):
    row = input(f"Row {i+1}: ").split()
    if len(row) != 9:
        print("Error: Each row must have exactly 9 elements. Please
re-enter the row.")
        exit()
    board.append(row)
```

```python
# Display Board
print_sudoku_board(board)

# Validation Result
result = is_valid_sudoku(board)

if result:
    print("\nThe given Sudoku board is valid.")
else:
    print("\nThe given Sudoku board is invalid.")
```

```
Enter the Sudoku board row by row (use '.' for empty cells):

Row 1:  1 . . . . . . . .
Row 2:  3 . . . . . . . 1
Row 3:  . . . . . . . . .
Row 4:  . . . . . 2 . . 9
Row 5:  . . 8 . . . . . .
Row 6:  2 . . . . . 5 .
Row 7:  . . . . . . . . .
Row 8:  9 . . . . . . . .
Row 9:  . . . . 1 . . . .


Sudoku Board:
1 . . | . . . | . . .
3 . . | . . . | . . 1
. . . | . . . | . . .
---------------------
. . . | . . 2 | . . 9
. . 8 | . . . | . . .
2 . . | . . . | . 5 .
---------------------
. . . | . . . | . . .
9 . . | . . . | . . .
. . . | . 1 . | . . .

The given Sudoku board is valid.
```

```python
#40.Word Frequency In a Test
from collections import Counter

def word_frequency(text):
    words=text.lower().split()
    return dict(Counter(words))

text="Mainflow services and tecnolgy pvt.limited"
print(word_frequency(text))
```

```
{'mainflow': 1, 'services': 1, 'and': 1, 'tecnolgy': 1, 'pvt.limited':
1}
```

```
#41.Knapsack Problem
def knapsack(weights,values,capacity):
    n=len(weights)
    dp=[[0 for i in range(capacity+1)] for j in range(n+1)]
    for k in range (1,n+1):
        for w in range(1,capacity+1):
            if weights[k-1]<=w:
                dp[k][w]=max(values[k-1]+dp[k-1][w-weights[k-1]],dp[k-
1][w])
            else:
                dp[k][w]=dp[k-1][w]
    return dp[n][capacity]
weights=[1,2,3,4]
values=[3,4,5,6]
capacity=5
print(knapsack(weights,values,capacity))
```

```
9
```

```
#42.Merge Intervals
def merge_intervals(intervals):
    intervals.sort()
    merged=[intervals[0]]
    for start,end in intervals[1:]:
        if start<=merged[-1][1]:
            merged[-1][1]=max(merged[-1][1],end)
        else:
            merged.append([start,end])
    return merged

intervals=[[3,5],[5,7],[6,7],[2,3]]
print(merge_intervals(intervals))
```

```
[[2, 7]]
```

```
#43.Find the median of two arrays
import statistics
def find_median(arr1,arr2):
    merged=sorted(arr1+arr2)
    return statistics.median(merged)

arr1=[3,5,7]
arr2=[2,6,8,9]
print(find_median(arr1,arr2))
```

```
6
```

```python
#44.Maximal Rectangle in Binary Matrix
def maximal_rectangle(matrix):
    if not matrix:
        return 0
    max_area=0
    heights=[0]*len(matrix[0])
    for row in matrix:
        for i in range(len(row)):
            heights[i]=heights[i]+ 1 if row[i]=="1" else 0
        max_area=max(max_area,max_histogram_area(heights))
    return max_area

def max_histogram_area(heights):
    stack,max_area=[],0
    for i,h in enumerate(heights+[0]):
        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

matrix=[
    ["1","1","0","1","0"],
    ["0","1","0","1","0"],
    ["1","1","0","1","0"],
    ["1","1","0","1","1"]
]
print(maximal_rectangle(matrix))
```

```
4
```

```python
#45.Larget sum contigious subarray(Kadane's Algorithm)
def max_subarray_sum(arr):
    max_sum=curr_sum=arr[0]
    for num in arr[1:]:
        curr_sum=max(num,curr_sum+num)
        max_sum=max(max_sum,curr_sum)
    return max_sum

arr=[-3,-4,1,2,3,4,5]
print(max_subarray_sum(arr))
```

```
15
```

```python
#46.word Ladder Problem

#6.Command-Line RPG Game(Project-6)
import random
class Character:
    def __init__(self,name,health,attack):
```

```
        self.name=name
        self.health=health
        self.attack=attack

    def attack_enemy(self,enemy):
        damage=random.randint(1,self.attack)
        enemy.health-=damage
        print(f"{self.name} attacks {enemy.name} for {damage}
damage!")

player=Character("Hero",100,15)
enemy=Character("goblin",50,10)

while enemy.health>0 and player.health>0:
    player.attack_enemy(enemy)
    if enemy.health<=0:
        print(f"{enemy.name} is defeated!")
        break
    enemy.attack_enemy(player)

print("Game over!")
```

```
Hero attacks goblin for 14 damage!
goblin attacks Hero for 9 damage!
Hero attacks goblin for 2 damage!
goblin attacks Hero for 4 damage!
Hero attacks goblin for 7 damage!
goblin attacks Hero for 4 damage!
Hero attacks goblin for 15 damage!
goblin attacks Hero for 2 damage!
Hero attacks goblin for 13 damage!
goblin is defeated!
Game over!
```