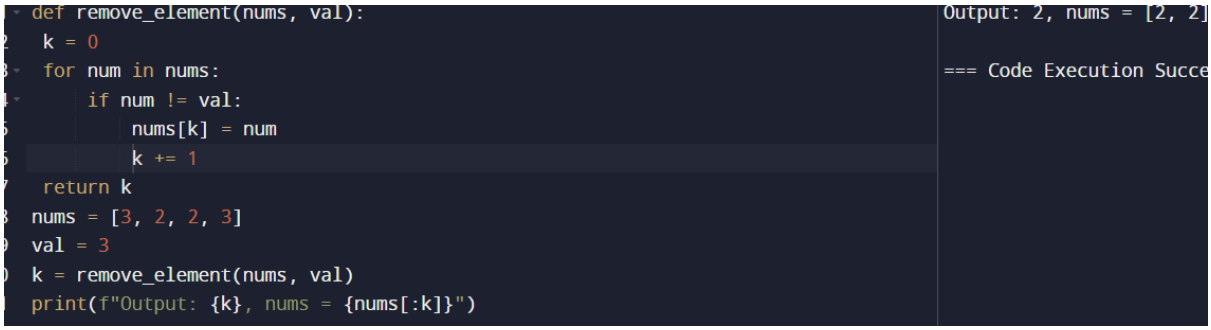
# LAB EXPERIMENT (TEST-1)

1)REMOVE ELEMENT.

CODE AND OUTPUT:



```
def remove_element(nums, val):
    k = 0
    for num in nums:
        if num != val:
            nums[k] = num
            k += 1
    return k
nums = [3, 2, 2, 3]
val = 3
k = remove_element(nums, val)
print(f"Output: {k}, nums = {nums[:k]}")
```

```
Output: 2, nums = [2, 2]

=== Code Execution Succe
```

2) Determine if a 9 x 9 Sudoku board is valid. Only the filled cells need to be validated according to the following rules:

CODE AND OUTPUT:

```
def is_valid_sudoku(board):

    rows = [set() for _ in range(9)]
    cols = [set() for _ in range(9)]
    boxes = [set() for _ in range(9)]

    for i in range(9):
        for j in range(9):
            cell = board[i][j]
            if cell != ".":

                box_index = (i // 3) * 3 + j // 3

                if cell in rows[i] or cell in cols[j] or cell in boxes[box_index]:
                    return False

                rows[i].add(cell)
                cols[j].add(cell)
                boxes[box_index].add(cell)

    return True

# Example usage
board = [
    ["5","3",".",".","7",".",".",".","."],
    ["6",".",".","1","9","5",".",".","."],
    [".","9","8",".",".",".",".","6","."],
    ["8",".",".",".","6",".",".",".","3"],
    ["4",".",".","8",".","3",".",".","1"],
    ["7",".",".",".","2",".",".",".","6"],
    [".","6",".",".",".",".","2","8","."],
    [".",".",".","4","1","9",".",".","5"],
    [".",".",".",".","8",".",".","7","9"]
]

print(is_valid_sudoku(board))
```

Output:

```
== RESTART: C:/Users
True
```

3) Sudoku Solver

Code:

```
def solve_sudoku(board):
    def is_valid(num, row, col):
        for i in range(9):
            if board[row][i] == num or board[i][col] == num:
                return False
        box_row, box_col = 3 * (row // 3), 3 * (col // 3)
        for i in range(box_row, box_row + 3):
            for j in range(box_col, box_col + 3):
                if board[i][j] == num:
                    return False
        return True
    def backtrack():
        for row in range(9):
            for col in range(9):
                if board[row][col] == ".":
                    for num in map(str, range(1, 10)):
                        if is_valid(num, row, col):
                            board[row][col] = num
                            if backtrack():
                                return True
                            board[row][col] = "."
                    return False
        return True
    backtrack()
# Example usage
board = [
    ["5","3",".",".","7",".",".",".","."],
    ["6",".",".","1","9","5",".",".","."],
    [".","9","8",".",".",".",".","6","."],
    ["8",".",".",".","6",".",".",".","3"],
    ["4",".",".","8",".","3",".",".","1"],
    ["7",".",".",".","2",".",".",".","6"],
    [".","6",".",".",".",".","2","8","."],
    [".",".",".","4","1","9",".",".","5"],
    [".",".",".",".","8",".",".","7","9"]
]
solve_sudoku(board)
for row in board:
    print(" ".join(row))
```

Output:

```
5 3 4 6 7 8 9 1 2
6 7 2 1 9 5 3 4 8
1 9 8 3 4 2 5 6 7
8 5 9 7 6 1 4 2 3
4 2 6 8 5 3 7 9 1
7 1 3 9 2 4 8 5 6
9 6 1 5 3 7 2 8 4
2 8 7 4 1 9 6 3 5
3 4 5 2 8 6 1 7 9
```

4)count and say

Code:

```
def count_and_say(n):
    def say(s):
        result = []
        i = 0
        while i < len(s):
            count = 1
            while i + 1 < len(s) and s[i] == s[i + 1]:
                count += 1
                i += 1
            result.append(str(count) + s[i])
            i += 1
        return "".join(result)

    sequence = "1"
    for _ in range(n - 1):
        sequence = say(sequence)

    return sequence

# Example usage
n = 1
print(count_and_say(n))
```

Output:

```
== RESTART: C:/
1
```

5)combination sum:

Code:

```python
def combination_sum(candidates, target):
    def backtrack(start, target, path):
        if target == 0:
            result.append(path[:])
            return
        for i in range(start, len(candidates)):
            if candidates[i] <= target:
                path.append(candidates[i])
                backtrack(i, target - candidates[i], path)
                path.pop()

    result = []
    candidates.sort()
    backtrack(0, target, [])
    return result

# Example usage
candidates = [2, 3, 6, 7]
target = 7
print(combination_sum(candidates, target))
```

Output:

```
== RESTART: C:/Users/s
[[2, 2, 3], [7]]
```

6) combination sum 2:

Code:

```python
def combination_sum2(candidates, target):
    def backtrack(start, target, path):
        if target == 0:
            result.append(path[:])
            return
        for i in range(start, len(candidates)):
            if i > start and candidates[i] == candidates[i - 1]:
                continue
            if candidates[i] <= target:
                path.append(candidates[i])
                backtrack(i + 1, target - candidates[i], path)
                path.pop()

    result = []
    candidates.sort()
    backtrack(0, target, [])
    return result

# Example usage
candidates = [10, 1, 2, 7, 6, 1, 5]
target = 8
print(combination_sum2(candidates, target))
```

Output:

```
-- RESTART: C:/Users/Sai17/AppData/Local/P
[[1, 1, 6], [1, 2, 5], [1, 7], [2, 6]]
```

7) permutations 2:

Code:

```
def permute_unique(nums):
    def backtrack(start):
        if start == len(nums):
            result.append(nums[:])
            return
        for i in range(start, len(nums)):
            if i > start and nums[i] == nums[start]:
                continue
            nums[start], nums[i] = nums[i], nums[start]
            backtrack(start + 1)
            nums[start], nums[i] = nums[i], nums[start]

    result = []
    nums.sort()
    backtrack(0)
    return result

# Example usage
nums = [1, 1, 2]
print(permute_unique(nums))
```

Output:

```
[[1, 1, 2], [1, 2, 1], [2, 1, 1]]
```

8)maximum subarray:

Code:

```
def max_subarray_sum(nums):
    max_sum = float('-inf')
    current_sum = 0

    for num in nums:
        current_sum = max(num, current_sum + num)
        max_sum = max(max_sum, current_sum)

    return max_sum

# Example usage
nums = [-2, 1, -3, 4, -1, 2, 1, -5, 4]
print(max_subarray_sum(nums))
```

Output:

```
6
```