

1. Remove Element

```
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]}")
```

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

```
def is_valid_sudoku(board):  
    seen = set()  
    for i in range(9):  
        for j in range(9):  
            if board[i][j] != '.':  
                current_num = board[i][j]  
                if (i, current_num) in seen or (current_num, j) in seen or (i // 3, j // 3, current_num)  
in seen:  
                    return False  
                seen.add((i, current_num))  
                seen.add((current_num, j))  
                seen.add((i // 3, j // 3, current_num))  
    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: True

37. Sudoku Solver

```
def solveSudoku(board):
```

```
    def is_valid(num, row, col):
```

```
        for i in range(9):
```

```
            if board[i][col] == num or board[row][i] == num or board[3 * (row // 3) + i // 3][3 *
(col // 3) + i % 3] == num:
```

```
                return False
```

```
        return True
```

```
def solve():
```

```
    for i in range(9):
```

```
        for j in range(9):
```

```
            if board[i][j] == '.':
```

```
                for num in '123456789':
```

```
                    if is_valid(num, i, j):
```

```
                        board[i][j] = num
```

```
                        if solve():
```

```
                            return True
```

```
                        board[i][j] = '.'
```

```
                return False
```

```
    return True
```

```
solve()
```

```
# 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"]]
```

```
solveSudoku(board)
```

```
print(board)
```

3.Count and Say

```
def countAndSay(n):
```

```
    if n == 1:
```

```
        return "1"
```

```
    prev = countAndSay(n - 1)
```

```
    result = ""
```

```
    count = 1
```

```
    for i in range(len(prev)):
```

```
        if i + 1 < len(prev) and prev[i] == prev[i + 1]:
```

```
            count += 1
```

```
        else:
```

```
            result += str(count) + prev[i]
```

```
            count = 1
```

```
    return result
```

```
# Test the function
```

```
n = 1
```

```
print(countAndSay(n)) # Output: "1"
```

39. Combination Sum

```
def combinationSum(candidates, target):
```

```
    def backtrack(start, path, target):
```

```
        if target == 0:
```

```

        result.append(path[:])
        return

    for i in range(start, len(candidates)):
        if candidates[i] > target:
            continue
        path.append(candidates[i])
        backtrack(i, path, target - candidates[i])
        path.pop()

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

```

Test the function with the provided example

```
candidates = [2, 3, 6, 7]
```

```
target = 7
```

```
print(combinationSum(candidates, target)) # Output: [[2, 2, 3], [7]]
```

40. Combination Sum II

```

def combinationSum2(candidates, target):
    def backtrack(start, path, target):
        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:
                break
            path.append(candidates[i])
            backtrack(i + 1, path, target - candidates[i])

```

```
path.pop()
```

```
candidates.sort()
```

```
result = []
```

```
backtrack(0, [], target)
```

```
return result
```

```
# Example
```

```
candidates = [10, 1, 2, 7, 6, 1, 5]
```

```
target = 8
```

```
print(combinationSum2(candidates, target))
```

Permutations II

```
from itertools import permutations
```

```
def unique_permutations(nums):
```

```
    return list(set(permutations(nums)))
```

```
# Test the function with the given example
```

```
nums = [1, 1, 2]
```

```
print(unique_permutations(nums))
```

53. Maximum Subarray

```
def max_subarray_sum(nums):
```

```
    max_sum = float('-inf')
```

```
    for i in range(len(nums)):
```

```
        current_sum = 0
```

```
        for j in range(i, len(nums)):
```

```
            current_sum += nums[j]
```

```
            max_sum = max(max_sum, current_sum)
```

```
    return max_sum
```

```
# Test the function
```

```
nums = [-2, 1, -3, 4, -1, 2, 1, -5, 4]
print(max_subarray_sum(nums)) # Output: 6
```

Permutation Sequence

```
import math

def getPermutation(n, k):
    numbers = list(range(1, n+1))
    k -= 1
    factorial = math.factorial(n)
    result = []

    for i in range(n, 0, -1):
        factorial //= i
        index = k // factorial
        k %= factorial
        result.append(str(numbers[index]))
        numbers.pop(index)

    return "".join(result)
```

```
# Example
n = 3
k = 3
output = getPermutation(n, k)
print(output) # Output: "213"
```

Length of Last Word

```
def length_of_last_word(s):
    words = s.split()
    if len(words) == 0:
        return 0
    return len(words[-1])
```

```
# Test the function
```

```
s = "Hello World"
```

```
print(length_of_last_word(s)) # Output: 5
```