ASSIGNMENT-2

Name: Shaik Jabbar Basha

11. Container With Most Water You are given an integer array height of length n. There are n vertical lines drawn such that the two endpoints of the ith line are (i, 0) and (i, height[i]). Find two lines that together with the x-axis form a container, such that the container contains the most water. Return the maximum amount of water a container can store. Notice that you may not slant the container.

Example 1: Input: height = [1,8,6,2,5,4,8,3,7]

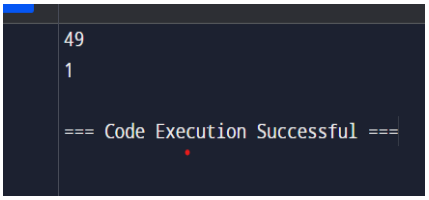
Output: 49 Explanation: The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container can contain is 49. Example 2: Input: height = [1,1] Output: 1

Program:

main.py

```python
def maxArea(height):
    left, right = 0, len(height) - 1
    max_area = 0

    while left < right:
        width = right - left
        min_height = min(height[left], height[right])
        current_area = width * min_height
        max_area = max(max_area, current_area)
        if height[left] < height[right]:
            left += 1
        else:
            right -= 1

    return max_area
height1 = [1, 8, 6, 2, 5, 4, 8, 3, 7]
height2 = [1, 1]
print(maxArea(height1))  # Output: 49
print(maxArea(height2))  # Output: 1
```

Output:



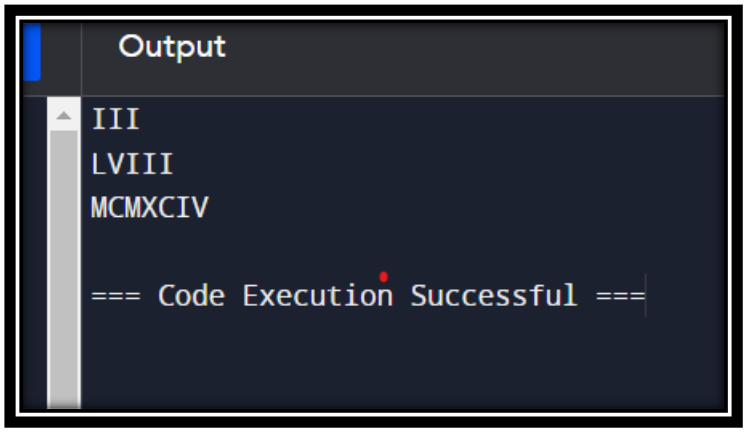
12. Integer to Roman Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M. Symbol Value I 1 V 5 X 10 L 50 C 100 D 500 M 1000 For example, 2 is written as II in Roman numeral, just two one's added together. 12 is written as XII, which is simply X + II. The number 27 is written as XXVII, which is XX + V + II. Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not IIII. Instead, the number four is written as IV. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as IX. There are six instances where subtraction is used: ● I can be placed before V (5) and X (10) to make 4 and 9. ● X can be placed before L (50) and C (100) to make 40 and 90. ● C can be placed before D (500) and M (1000) to make 400 and 900. Given an integer, convert it to a roman numeral. Example 1: Input: num = 3 Output: "III" Explanation: 3 is represented as 3 ones. Example 2: Input: num = 58 Output: "LVIII" Explanation: L = 50, V = 5, III = 3. Example 3: Input: num = 1994 Output: "MCMXCIV" Explanation: M = 1000, CM = 900, XC = 90 and IV = 4.

Program:

main.py

```python
def intToRoman(num):
    val = [
        1000, 900, 500, 400,
        100, 90, 50, 40,
        10, 9, 5, 4,
        1
    ]
    syms = [
        "M", "CM", "D", "CD",
        "C", "XC", "L", "XL",
        "X", "IX", "V", "IV",
        "I"
    ]

    roman_numeral = ""
    for i in range(len(val)):
        while num >= val[i]:
            num -= val[i]
            roman_numeral += syms[i]

    return roman_numeral

# Example usage
print(intToRoman(3))    # Output: "III"
print(intToRoman(58))   # Output: "LVIII"
print(intToRoman(1994)) # Output: "MCMXCIV"
```

Output:



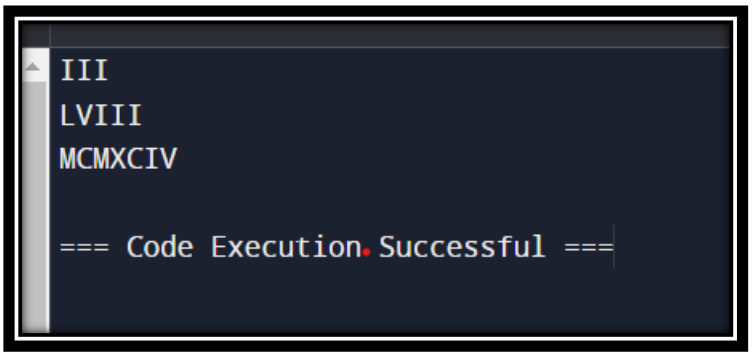
13. Roman to Integer Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M. Symbol Value I 1 V 5 X 10 L 50 C 100 D 500 M 1000 For example, 2 is written as II in Roman numeral, just two ones added together. 12 is written as XII, which is simply X + II. The number 27 is written as XXVII, which is XX + V + II. Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not IIII. Instead, the number four is written as IV. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as IX. There are six instances where subtraction is used: ● I can be placed before V (5) and X (10) to make 4 and 9. ● X can be placed before L (50) and C (100) to make 40 and 90. ● C can be placed before D (500) and M (1000) to make 400 and 900. Given a roman numeral, convert it to an integer. Example 1: Input: s = "III" Output: 3 Explanation: III = 3. Example 2: Input: s = "LVIII" Output: 58 Explanation: L = 50, V= 5, III = 3. Example 3: Input: s = "MCMXCIV" Output: 1994

Program:

main.py

```python
def romanToInt(s):
    # Define the Roman numeral symbols and their corresponding values
    roman_to_int = {
        'I': 1, 'V': 5, 'X': 10, 'L': 50,
        'C': 100, 'D': 500, 'M': 1000
    }

    # Initialize the integer value
    total = 0
    i = 0

    while i < len(s):
        # Check if this is a subtractive combination
        if i + 1 < len(s) and roman_to_int[s[i]] < roman_to_int[s[i + 1]]:
            total += roman_to_int[s[i + 1]] - roman_to_int[s[i]]
            i += 2  # Move past this pair of characters
        else:
            total += roman_to_int[s[i]]
            i += 1

    return total

# Example usage
print(romanToInt("III"))     # Output: 3
print(romanToInt("LVIII"))   # Output: 58
print(romanToInt("MCMXCIV")) # Output: 1994
```

Output:



14. Longest Common Prefix Write a function to find the longest common prefix string amongst an array of strings. If there is no common prefix, return an empty string "". Example 1: Input: strs = ["flower","flow","flight"] Output: "fl" Example 2: Input: strs = ["dog","racecar","car"] Output: "" Explanation: There is no common prefix among the input strings

Program:

```python
def longestCommonPrefix(strs):
    if not strs:
        return ""

    # The first string is taken as the reference
    prefix = strs[0]

    # Iterate over the other strings
    for s in strs[1:]:
        # Compare the prefix with each string and reduce it if necessary
        while not s.startswith(prefix):
            prefix = prefix[:-1]
            if not prefix:
                return ""

    return prefix

# Example usage
print(longestCommonPrefix(["flower", "flow", "flight"]))  # Output: "fl"
print(longestCommonPrefix(["dog", "racecar", "car"]))     # Output: ""
```

Output:

```
fl


=== Code Execution Successful ===
```

15. 3Sum Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that i != j, i != k, and j != k, and nums[i] + nums[j] + nums[k] == 0. Notice that the solution set must not contain duplicate triplets. Example 1: Input: nums = [-1,0,1,2,-1,-4]

Output: [[-1,-1,2],[-1,0,1]] Explanation: nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0. nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0. nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0. The distinct triplets are [-1,0,1] and [-1,-1,2]. Notice that the order of the output and the order of the triplets does not matter. Example 2: Input: nums = [0,1,1] Output: [] Explanation: The only possible triplet does not sum up to 0. Example 3: Input: nums = [0,0,0] Output: [[0,0,0]] Explanation: The only possible triplet sums up to 0

Program:

```
def threeSum(nums):
    nums.sort()
    result = []
    for i in range(len(nums) - 2):
        if i > 0 and nums[i] == nums[i - 1]:
            continue
        left, right = i + 1, len(nums) - 1
        while left < right:
            total = nums[i] + nums[left] + nums[right]

            if total == 0:
                result.append([nums[i], nums[left], nums[right]])
                while left < right and nums[left] == nums[left + 1]:
                    left += 1
                while left < right and nums[right] == nums[right - 1]:
                    right -= 1
                left += 1
                right -= 1
            elif total < 0:
                left += 1
            else:
                right -= 1

    return result

```

Output:

```
fl


=== Code Execution Successful ===
```

16. 3Sum Closest Given an integer array nums of length n and an integer target, find three integers in nums such that the sum is closest to target. Return the sum of the three integers.

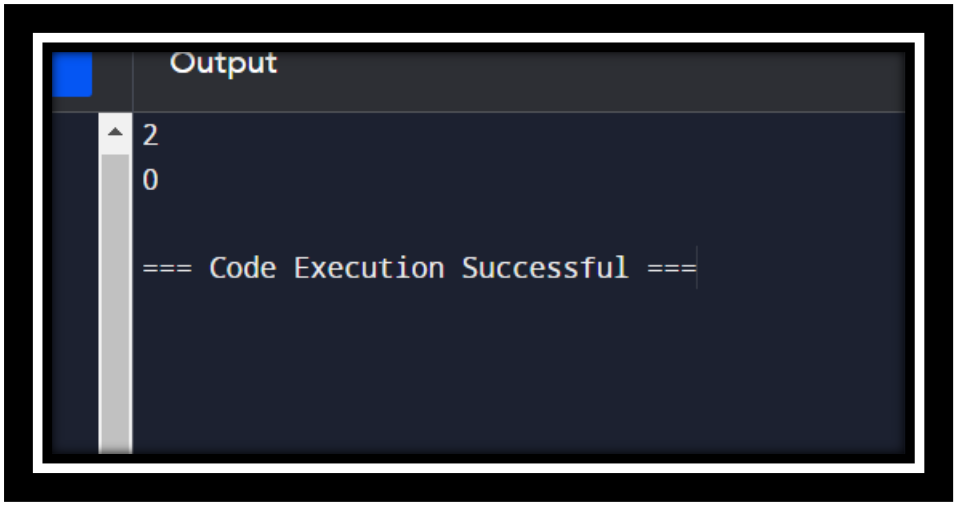
You may assume that each input would have exactly one solution. Example 1: Input: nums = [-1,2,1,-4], target = 1 Output: 2 Explanation: The sum that is closest to the target is 2. (-1 + 2 + 1 = 2). Example 2: Input: nums = [0,0,0], target = 1 Output: 0 Explanation: The sum that is closest to the target is 0. (0 + 0 + 0 = 0).

Program:

main.py Save Run

```
def threeSumClosest(nums, target):
    # Sort the array to use the two-pointer technique
    nums.sort()
    # Initialize the closest sum with a large number
    closest_sum = float('inf')

    for i in range(len(nums) - 2):
        # Use two pointers to find the closest sum for the current nums[i]
        left, right = i + 1, len(nums) - 1

        while left < right:
            current_sum = nums[i] + nums[left] + nums[right]
            if current_sum == target:
                return current_sum
            if abs(current_sum - target) < abs(closest_sum - target):
                closest_sum = current_sum

            # Move the pointers based on the comparison with the target
            if current_sum < target:
                left += 1
            else:
                right -= 1
    return closest_sum
print(threeSumClosest([-1, 2, 1, -4], 1))  # Output: 2
print(threeSumClosest([0, 0, 0], 1))       # Output: 0

```

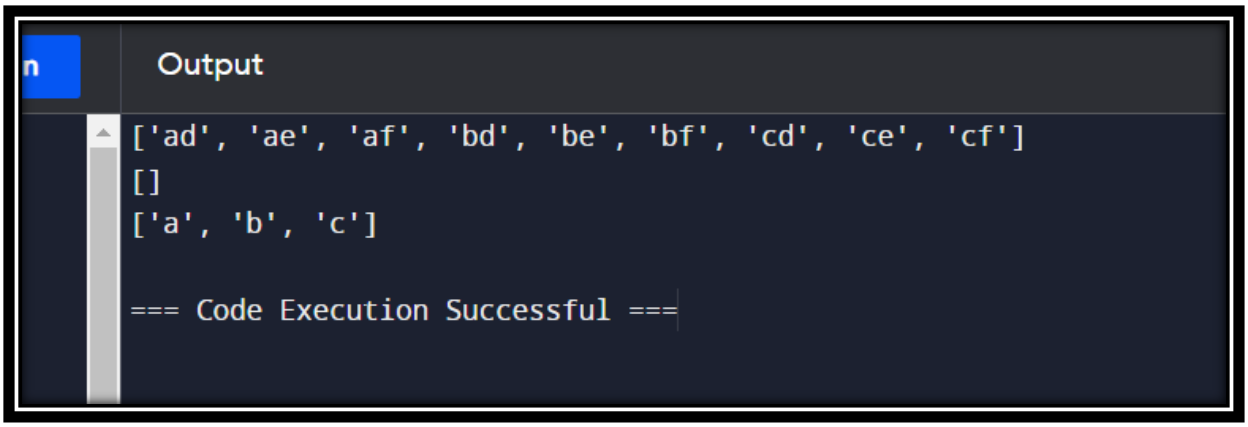
Output:



17. Letter Combinations of a Phone Number Given a string containing digits from 2-9 inclusive, return all possible letter combinations that the number could represent. Return the answer in any order. A mapping of digits to letters (just like on the telephone buttons) is given below. Note that 1 does not map to any letters. Example 1: Input: digits = "23" Output: ["ad","ae","af","bd","be","bf","cd","ce","cf"] Example 2: Input: digits = "" Output: [] Example 3: Input: digits = "2" Output: ["a","b","c"]

Program:

Output:



18. 4Sum Given an array nums of n integers, return an array of all the unique quadruplets [nums[a], nums[b], nums[c], nums[d]] such that: ● 0 <= a, b, c, d < n ● a, b, c, and d are distinct. ● nums[a] + nums[b] + nums[c] + nums[d] == target You may return the answer in any order. Example 1: Input: nums = [1,0,-1,0,-2,2], target = 0 Output: [[-2,-1,1,2],[-2,0,0,2],[-1,0,0,1]] Example 2: Input: nums = [2,2,2,2,2], target = 8 Output: [[2,2,2,2]]

Program:

```
def fourSum(nums, target):
    # Sort the array to facilitate the two-pointer approach
    nums.sort()
    result = []
    n = len(nums)

    # Outer two loops to fix the first two numbers
    for i in range(n - 3):
        # Skip duplicate elements for the first number
        if i > 0 and nums[i] == nums[i - 1]:
            continue

        for j in range(i + 1, n - 2):
            # Skip duplicate elements for the second number
            if j > i + 1 and nums[j] == nums[j - 1]:
                continue

            # Two-pointer approach for the remaining two numbers
            left, right = j + 1, n - 1
            while left < right:
                current_sum = nums[i] + nums[j] + nums[left] + nums[right]
                if current_sum == target:
                    result.append([nums[i], nums[j], nums[left], nums[right]])

                    # Move left and right pointers to the next unique elements
                    while left < right and nums[left] == nums[left + 1]:
                        left += 1
                    while left < right and nums[right] == nums[right - 1]:
                        right -= 1

                    left += 1
                    right -= 1
                elif current_sum < target:
                    left += 1
                else:
                    right -= 1

    return result

# Example usage
print(fourSum([1, 0, -1, 0, -2, 2], 0))  # Output: [[-2, -1, 1, 2], [-2, 0, 0, 2], [-1, 0, 0, 1]]
print(fourSum([2, 2, 2, 2, 2], 8))       # Output: [[2, 2, 2, 2]]
```

Output:

```
['ad', 'ae', 'af', 'bd', 'be', 'bf', 'cd', 'ce', 'cf']
[]
['a', 'b', 'c']

=== Code Execution Successful ===
```

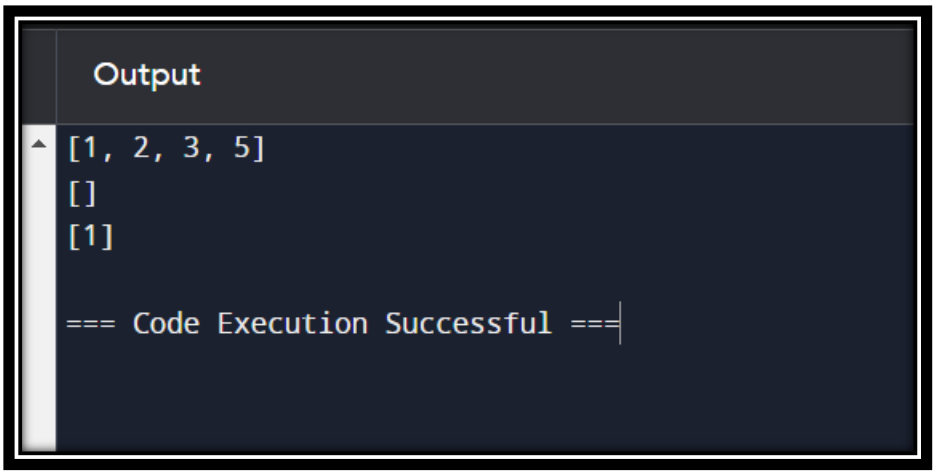
19. Remove Nth Node From End of List Given the head of a linked list, remove the nth node from the end of the list and return its head. Example 1: Input: head = [1,2,3,4,5], n = 2 Output: [1,2,3,5] Example 2: Input: head = [1], n = 1 Output: [] Example 3: Input: head = [1,2], n = 1 Output: [1]

Program:

File Edit Format Run Options Windows Help

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

def removeNthFromEnd(head, n):
    # Create a dummy node to handle edge cases smoothly
    dummy = ListNode(0)
    dummy.next = head
    first = dummy
    second = dummy

    # Move the first pointer n+1 steps ahead
    for _ in range(n + 1):
        first = first.next

    # Move both first and second pointers until first reaches the end
    while first is not None:
        first = first.next
        second = second.next

    # Remove the nth node from the end
    second.next = second.next.next

    # Return the new head of the list
    return dummy.next

# Helper function to create a linked list from a list
def create_linked_list(lst):
    if not lst:
        return None
    head = ListNode(lst[0])
    current = head
    for value in lst[1:]:
        current.next = ListNode(value)
        current = current.next
    return head

# Helper function to convert linked list to a list
def linked_list_to_list(head):
    result = []
    current = head
    while current:
        result.append(current.val)
        current = current.next
    return result
```

Output:



20. Valid Parentheses Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid. An input string is valid if: 1. Open brackets must be closed by the same type of brackets. 2. Open brackets must be closed in the correct order. 3. Every close bracket has a corresponding open bracket of the same type. Example 1: Input: s = "()" Output: true Example 2: Input: s = "()[]{}" Output: true Example 3: Input: s = "(]" Output: false

Program:

```python
def isValid(s):
    # Dictionary to hold matching pairs of brackets
    bracket_map = {')': '(', '}': '{', ']': '['}
    # Stack to keep track of opening brackets
    stack = []

    for char in s:
        if char in bracket_map:
            # Pop the topmost element from the stack if it's not empty,
                otherwise use a dummy value
            top_element = stack.pop() if stack else '#'
            # Check if the top element matches the corresponding opening
                bracket
            if bracket_map[char] != top_element:
                return False
        else:
            # Push the opening bracket onto the stack
            stack.append(char)

    # If the stack is empty, all brackets were matched correctly
    return not stack

# Example usage
print(isValid("()"))      # Output: true
print(isValid("()[]{}"))  # Output: true
print(isValid("(]"))      # Output: false
```

Output: