Program 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 Constraints: ● n == height.length ● 2 <= n <= 105 ● 0 <= height[i] <= 104

Program:

```
def maxArea(height):
    left, right = 0, len(height) - 1
    max_area = 0
      while left < right:
       width = right - left
       h = min(height[left], height[right])
       max_area = max(max_area, width * h)
       if height[left] < height[right]:
          left += 1
       else:
          right -= 1

    return max_area
# Example usage:
height = [1, 8, 6, 2, 5, 4, 8, 3, 7]
print(maxArea(height))  # Output: 49

height = [1, 1]
print(maxArea(height))  # Output: 1
```

```
"C:\Program Files\Python312\python.exe" "C:\Work Space\DAA COADS.PYTHON\assignment 2\ass1.py"
49
1

Process finished with exit code 0
```

PROGAR12: 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

PROGRAM:-

```
def intToRoman(num):
    val = [
        (1000, 'M'), (900, 'CM'), (500, 'D'), (400, 'CD'),
        (100, 'C'), (90, 'XC'), (50, 'L'), (40, 'XL'),
        (10, 'X'), (9, 'IX'), (5, 'V'), (4, 'IV'), (1, 'I')
    ]
    roman = ""
    for value, symbol in val:
        while num >= value:
            roman += symbol
            num -= value
    return roman
# Example usage:
num = 3
print(intToRoman(num))  # Output: "III"
num = 27
print(intToRoman(num))  # Output: "XXVII"
num = 58
print(intToRoman(num))  # Output: "LVIII"
```

num = 1994

print(intToRoman(num)) # Output: "MCMXCIV"

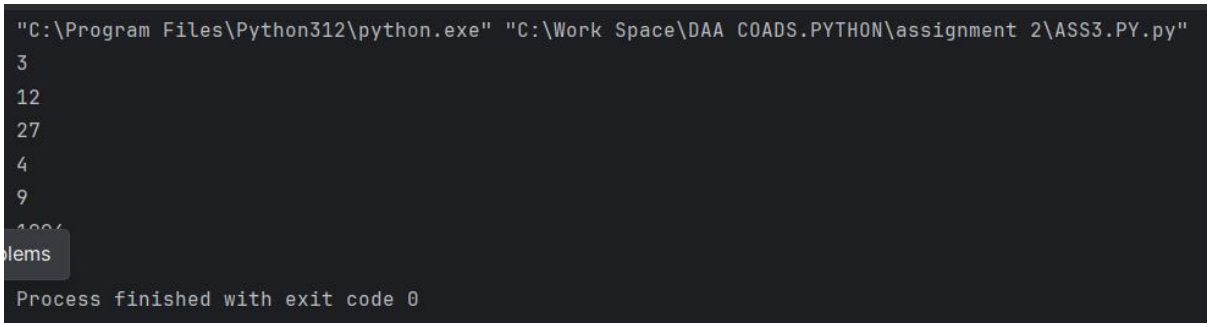
```
"C:\Program Files\Python312\python.exe" "C:\Work Space\DAA COADS.PYTHON\assignment 2\ass2.py"
III
XXVII
LVIII
MCMXCIV

Process finished with exit code 0
```

PROGRAM13: 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.

PROGRAM:

```
def romanToInt(s: str) -> int:
    # Define the Roman numeral values
    roman_values = {
        'I': 1,
        'V': 5,
        'X': 10,
        'L': 50,
        'C': 100,
        'D': 500,
        'M': 1000
    }
    total = 0
    prev_value = 0
    # Iterate over the characters in the string from right to left
    for char in reversed(s):
        value = roman_values[char]
        # If the current value is less than the previous one, subtract it from total
        if value < prev_value:
            total -= value
```

```
        else:
            total += value
        prev_value = value
    return total
# Example usage:
s = "III"
print(romanToInt(s))  # Output: 3
s = "XII"
print(romanToInt(s))  # Output: 12
s = "XXVII"
print(romanToInt(s))  # Output: 27
s = "IV"
print(romanToInt(s))  # Output: 4
s = "IX"
print(romanToInt(s))  # Output: 9
s = "MCMXCIV"
print(romanToInt(s))  # Output: 1994
```

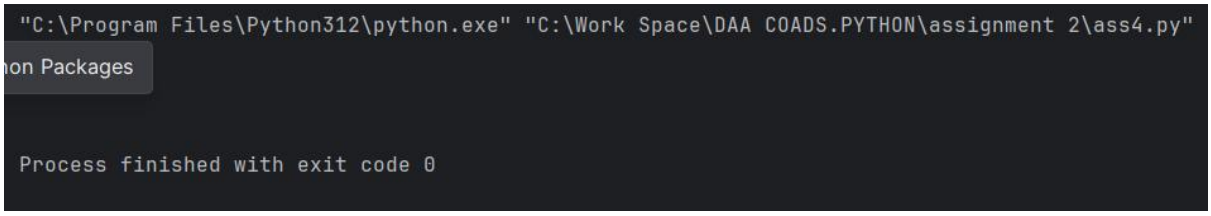


14. Longest Common Prefix Write a function to find the longest common prefix string amongst an array of strings. If there is no common prefix, return an empty string "". Example 1: Input: strs = ["flower","flow","flight"] Output: "fl"

PROGRAM:-

```
def longestCommonPrefix(strs):
    if not strs:
        return ""
    # Find the minimum length string in the list
    min_length = min(len(s) for s in strs)
   # Initialize the prefix to an empty string
     prefix = ""
    for i in range(min_length):
```

```
        # Take the current character from the first string
        current_char = strs[0][i]
        # Check if this character is the same in all strings at the current position
        if all(s[i] == current_char for s in strs):
            prefix += current_char
        else:
            break
    return prefix
# Example usage:
strs = ["flower", "flow", "flight"]
print(longestCommonPrefix(strs))  # Output: "fl"
strs = ["dog", "racecar", "car"]
print(longestCommonPrefix(strs))  # Output: ""
```



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.

Program:

```
def threeSum(nums):
    # Sort the array to make it easier to avoid duplicates
    nums.sort()
    result = []
    for i in range(len(nums) - 2):
        # Skip the same element to avoid duplicates
        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]])
            # Move left and right to the next different numbers
            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
# Example usage:
nums = [-1, 0, 1, 2, -1, -4]
print(threeSum(nums))  # Output: [[-1, -1, 2], [-1, 0, 1]]
```

```
"C:\Program Files\Python312\python.exe" "C:\Work Space\DAA COADS.PYTHON\assignment 2\ass15.py"
[[-1, -1, 2], [-1, 0, 1]]

Process finished with exit code 0
```

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

Program:

```
def threeSumClosest(nums, target):
    nums.sort()
    closest_sum = float('inf')
    for i in range(len(nums) - 2):
        left, right = i + 1, len(nums) - 1
        while left < right:
            current_sum = nums[i] + nums[left] + nums[right]
            # Update the closest sum if the current one is closer to the target
            if abs(current_sum - target) < abs(closest_sum - target):
                closest_sum = current_sum
```

```
        if current_sum < target:
            left += 1
        elif current_sum > target:
            right -= 1
        else:
            # If the current sum is exactly equal to the target, return it immediately
            return current_sum
    return closest_sum
# Example usage:
nums = [-1, 2, 1, -4]
target = 1
print(threeSumClosest(nums, target))  # Output: 2
```

```
"C:\Program Files\Python312\python.exe" "C:\Work Space\DAA COADS.PYTHON\assignment 2\ass16.py"
2

Process finished with exit code 0
```

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"]

Program:

```
def letterCombinations(digits):
    if not digits:
        return []
    # Mapping of digits to letters
    phone_map = {
        '2': 'abc',
        '3': 'def',
        '4': 'ghi',
        '5': 'jkl',
        '6': 'mno',
        '7': 'pqrs',
        '8': 'tuv',
      '9': 'wxyz'
    }
```

```
    def backtrack(index, path):
        # If the path length is equal to the digits length, add to combinations
        if index == len(digits):
            combinations.append("".join(path))
            return
        # Get the letters that the current digit maps to, and recurse
        possible_letters = phone_map[digits[index]]
        for letter in possible_letters:
            path.append(letter)
            backtrack(index + 1, path)
            path.pop()
    combinations = []
    backtrack(0, [])
    return combinations
# Example usage:
digits = "23"
print(letterCombinations(digits))  # Output: ["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"]
```

```
"C:\Program Files\Python312\python.exe" "C:\Work Space\DAA COADS.PYTHON\assignment 2\ass17.py"
['ad', 'ae', 'af', 'bd', 'be', 'bf', 'cd', 'ce', 'cf']

Process finished with exit code 0
```

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]

Program:

```
def fourSum(nums, target):
    nums.sort()
    result = []
    length = len(nums)
    for i in range(length - 3):
        # Avoid duplicates for the first number
        if i > 0 and nums[i] == nums[i - 1]:
            continue


        for j in range(i + 1, length - 2):
```

```
            # Avoid duplicates for the second number
            if j > i + 1 and nums[j] == nums[j - 1]:
                continue
            left, right = j + 1, length - 1
            while left < right:
                total = nums[i] + nums[j] + nums[left] + nums[right]
                if total == target:
                    result.append([nums[i], nums[j], nums[left], nums[right]])
                    # Avoid duplicates for the third and fourth numbers
                    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 < target:
                    left += 1
                else:
                    right -= 1
    return result
# Example usage:
nums = [1, 0, -1, 0, -2, 2]
target = 0
print(fourSum(nums, target))  # Output: [[-2, -1, 1, 2], [-2, 0, 0, 2], [-1, 0, 0, 1]]
```

```
"C:\Program Files\Python312\python.exe" "C:\Work Space\DAA COADS.PYTHON\assignment 2\ass18.py"
[[-2, -1, 1, 2], [-2, 0, 0, 2], [-1, 0, 0, 1]]

Process finished with exit code 0
```

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]

Program:

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

```
        self.val = val
        self.next = next
def removeNthFromEnd(head, n):
    # Create a dummy node which points to the head of the list
    dummy = ListNode(0)
    dummy.next = head
    first = dummy
    second = dummy
    # Move first pointer n+1 steps ahead
    for _ in range(n + 1):
        first = first.next
    # Move both pointers until first pointer reaches the end
    while first:
        first = first.next
        second = second.next
    # Remove the nth node from the end
    second.next = second.next.next
    return dummy.next
# Example usage:
def print_list(node):
    while node:
        print(node.val, end=" -> ")
        node = node.next
    print("None")
head = ListNode(1, ListNode(2, ListNode(3, ListNode(4, ListNode(5)))))
n = 2
new_head = removeNthFromEnd(head, n)
print_list(new_head)  # Output: 1 -> 2 -> 3 -> 5 -> None
```

```
"C:\Program Files\Python312\python.exe" "C:\Work Space\DAA COADS.PYTHON\assignment 2\ass19.py"
1 -> 2 -> 3 -> 5 -> None

Process finished with exit code 0
```

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

Program:

```
def isValid(s):
    # Dictionary to hold the mapping of closing and opening brackets
    bracket_map = {')': '(', '}': '{', ']': '['}
    # Stack to keep track of opening brackets
    stack = []
    for char in s:
        # If the character is a closing bracket
        if char in bracket_map:
            # Pop the top element from the stack if it is not empty, otherwise assign 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:
            # If it's an opening bracket, push it onto the stack
            stack.append(char)
    # If the stack is empty, all brackets were matched correctly
    return not stack
# Example usage:
s = "()"
print(isValid(s))  # Output: True
s = "()[]{}"
print(isValid(s))  # Output: True
s = "(]"
print(isValid(s))  # Output: False
s = "([)]"
print(isValid(s))  # Output: False
s = "{[]}"
print(isValid(s))  # Output: True
```

```
"C:\Program Files\Python312\python.exe" "C:\Work Space\DAA COADS.PYTHON\assignment 2\ass20.py"
True
True
False
False
True

Process finished with exit code 0
```