

ASSIGNMENT-2

DATE:04/06/2024

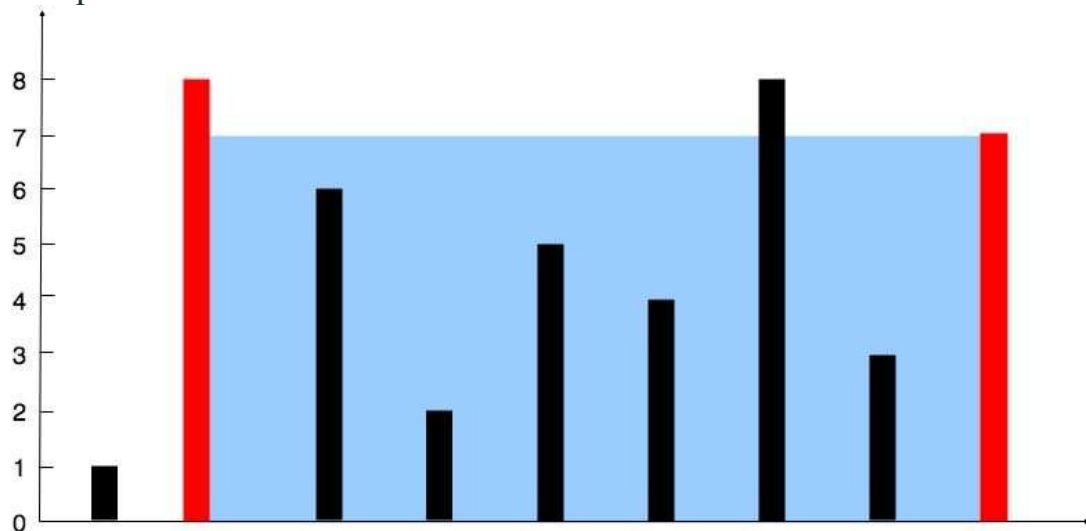
11. Container With Most Water

You are given an integer array `height` of length n . There are n vertical lines drawn such that the two endpoints of the i th line are $(i, 0)$ and $(i, \text{height}[i])$. 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

CODING:

```
2 usages
1 def maxArea(height):
2     left, right = 0, len(height) - 1
3     max_area = 0
4     while left < right:
5         area = min(height[left], height[right]) * (right - left)
6         max_area = max(max_area, area)
7         if height[left] < height[right]:
8             left += 1
9         else:
10            right -= 1
11    return max_area
12 height = [1,8,6,2,5,4,8,3,7]
13 print(maxArea(height))
14 height = [1,1]
15 print(maxArea(height))
```

OUTPUT:

```
C:\Users\vinot\PycharmProjects\pythonProject3\.venv\Scripts\python.exe C:\Users\
49
1
Process finished with exit code 0
```

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.

Constraints:

- $1 \leq \text{num} \leq 3999$

Coding:

```
3 usages
1  def intToRoman(num):
2      val = [
3          1000, 900, 500, 400,
4          100, 90, 50, 40,
5          10, 9, 5, 4,
6          1
7      ]
8      syb = [
9          "M", "CM", "D", "CD",
10         "C", "XC", "L", "XL",
11         "X", "IX", "V", "IV",
12         "I"
13     ]
14     roman_num = ''
15     i = 0
16     while num > 0:
```

```

5     i = 0
6     while num > 0:
7         for _ in range(num // val[i]):
8             roman_num += syb[i]
9             num -= val[i]
10        i += 1
11    return roman_num
12    print(intToRoman(3))
13    print(intToRoman(58))
14    print(intToRoman(1994))

```

OUTPUT:

```

C:\Users\vinot\PycharmProjects\pythonP
III
LVIII
MCMXCIV

Process finished with exit code 0

```

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

Explanation: M = 1000, CM = 900, XC = 90 and IV = 4.

Constraints:

- $1 \leq s.length \leq 15$
- s contains only the characters ('I', 'V', 'X', 'L', 'C', 'D', 'M').
- It is guaranteed that s is a valid roman numeral in the range [1, 3999]

Coding:

```
3 usages
1 def romanToInt(s: str) -> int:
2     roman_dict = {'I': 1, 'V': 5, 'X': 10, 'L': 50, 'C': 100, 'D': 500, 'M': 1000}
3     total = 0
4     for i in range(len(s) - 1):
5         if roman_dict[s[i]] < roman_dict[s[i + 1]]:
6             total -= roman_dict[s[i]]
7         else:
8             total += roman_dict[s[i]]
9     return total + roman_dict[s[-1]]
10 print(romanToInt("III")) # Output: 3
11 print(romanToInt("LVIII")) # Output: 58
12 print(romanToInt("MCMXCIV")) # Output: 1994
13
```

Output:

```
C:\Users\vinot\PycharmProjects\pythonProject3\.venv>
3
58
1994

Process finished with exit code 0
```

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.

Constraints:

- $1 \leq \text{strs.length} \leq 200$
- $0 \leq \text{strs}[i].\text{length} \leq 200$
- `strs[i]` consists of only lowercase English letters.

Coding:

```
1  def longestCommonPrefix(strs: List[str]) -> str:
2      if not strs:
3          return ""
4      prefix = strs[0]
5      for i in range(1, len(strs)):
6          while strs[i].find(prefix) != 0:
7              prefix = prefix[:-1]
8              if not prefix:
9                  return ""
10     return prefix
11     print(longestCommonPrefix(["flower", "flow", "flight"])) # Output: "fl"
12     print(longestCommonPrefix(["dog", "racecar", "car"])) # Output: ""
```

Output:

```
C:\Users\vinot\PycharmProjects\pythonProject
fl

Process finished with exit code 0
```

```
C:\Users\vinot\PycharmProjects\pythonProject
fl

Process finished with exit code 0
```

15. 3Sum

Given an integer array `nums`, return all the triplets `[nums[i], nums[j], nums[k]]` such that $i \neq j$, $i \neq k$, and $j \neq k$, and $\text{nums}[i] + \text{nums}[j] + \text{nums}[k] = 0$.

Notice that the solution set must not contain duplicate triplets.

Coding:

```
def threeSum(nums):
    result = []
    nums.sort() # Sort the input list

    for i in range(len(nums) - 2):
        # Skip duplicates for the first element
        if i > 0 and nums[i] == nums[i - 1]:
            continue

        left = i + 1
        right = len(nums) - 1

        while left < right:
            total = nums[i] + nums[left] + nums[right]

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

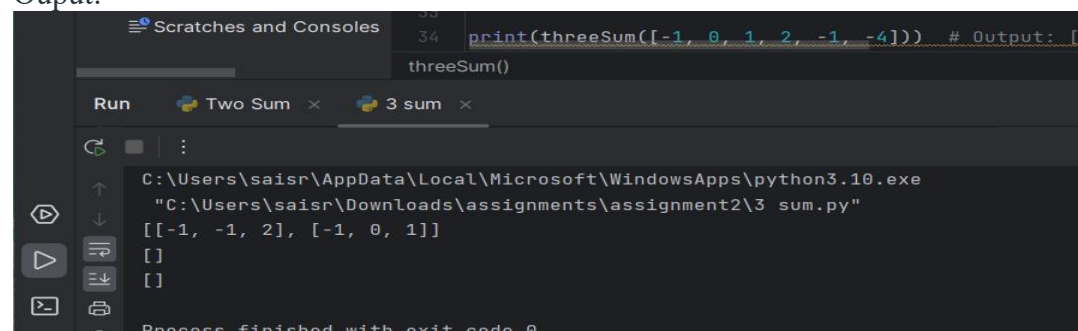
                # Skip duplicates for the second and third elements
                while left < right and nums[left] == nums[left + 1]:
                    left += 1
                while left < right and nums[right] == nums[right - 1]:
                    right -= 1

                left += 1
                right -= 1

    return result

print(threeSum([-1, 0, 1, 2, -1, -4])) # Output: [[-1, -1, 2], [-1, 0, 1]]
print(threeSum([])) # Output: []
print(threeSum([0])) # Output: []
```

Output:

A screenshot of a Python IDE window titled "Scratches and Consoles". The console shows the execution of the code. The first line of output is "[[-1, -1, 2], [-1, 0, 1]]", followed by two empty lists "[]" on separate lines. At the bottom, it says "Process finished with exit code 0". The IDE interface includes a "Run" button and tabs for "Two Sum" and "3 sum".

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.

Coding:

```
def threeSumClosest(nums, target):
    nums.sort() # Sort the input list
    closest_sum = nums[0] + nums[1] + nums[2] # Initialize with the
sum of the first three elements

    for i in range(len(nums) - 2):
        # Skip duplicates for the first element
        if i > 0 and nums[i] == nums[i - 1]:
            continue

        left = i + 1
        right = len(nums) - 1

        while left < right:
            total = nums[i] + nums[left] + nums[right]

            if total == target:
                return target # If the sum equals the target, return
it

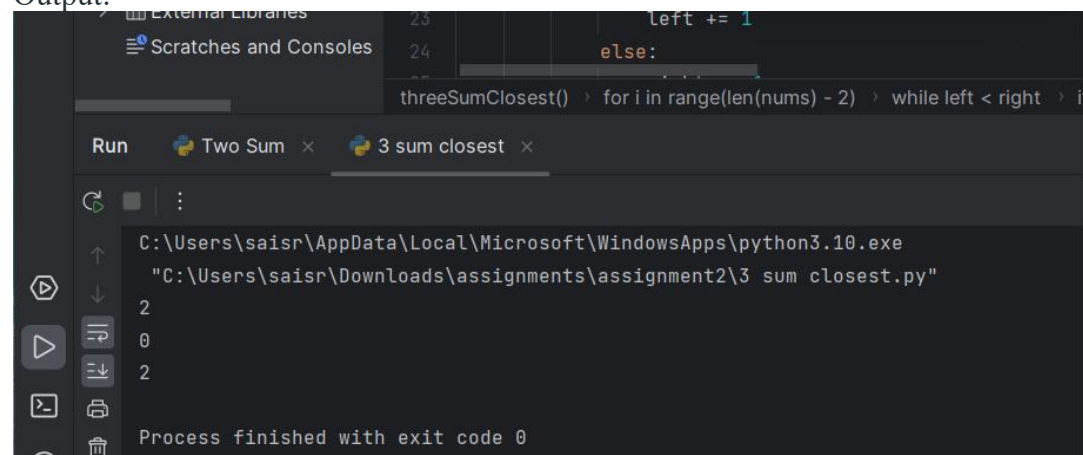
            if abs(total - target) < abs(closest_sum - target):
                closest_sum = total # Update the closest sum if the
current sum is closer

            if total < 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
print(threeSumClosest([1, 1, 1, 0], -100)) # Output: 2
```

Output:



The screenshot shows a code editor with a dark theme. The console output is visible at the bottom, showing the results of the threeSumClosest function for three test cases: 2, 0, and 2. The process finished with exit code 0. The code editor also shows the function definition and the test cases being run.

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.

Coding:

```
def letterCombinations(digits):
    if not digits:
        return []

    digit_map = {
        '2': 'abc',
        '3': 'def',
        '4': 'ghi',
        '5': 'jkl',
        '6': 'mno',
        '7': 'pqrs',
        '8': 'tuv',
        '9': 'wxyz'
    }

    combinations = []

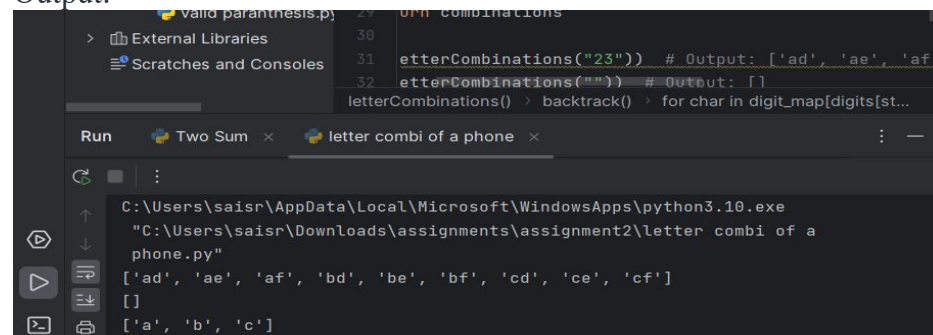
    def backtrack(combination, start):
        if len(combination) == len(digits):
            combinations.append(''.join(combination))
            return

        for char in digit_map[digits[start]]:
            combination.append(char)
            backtrack(combination, start + 1)
            combination.pop()

    backtrack([], 0)
    return combinations

print(letterCombinations("23")) # Output: ['ad', 'ae', 'af', 'bd',
'be', 'bf', 'cd', 'ce', 'cf']
print(letterCombinations("")) # Output: []
print(letterCombinations("2")) # Output: ['a', 'b', 'c']
```

Output:



```
letterCombinations("23") # Output: ['ad', 'ae', 'af', 'bd', 'be', 'bf', 'cd', 'ce', 'cf']
letterCombinations("") # Output: []
letterCombinations("2") # Output: ['a', 'b', 'c']
```

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:

- $0 \leq a, b, c, d < n$
- `a, b, c, and d` are distinct.
- `nums[a] + nums[b] + nums[c] + nums[d] == target`

You may return the answer in any order.

Coding:

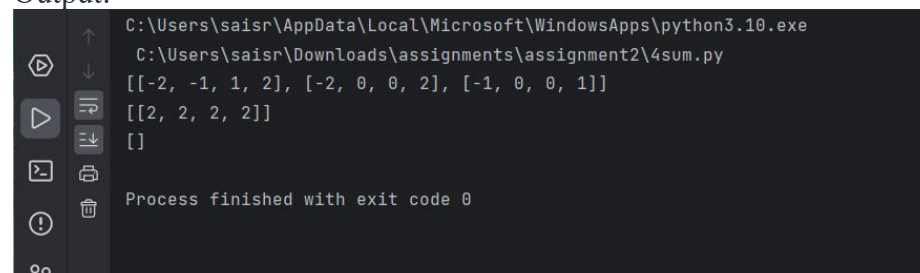
```
def fourSum(nums, target):
    nums.sort()
    n = len(nums)
    result = []

    def kSum(start, k, curr_sum, curr_nums):
        if k == 2:
            left, right = start, n - 1
            while left < right:
                total = curr_sum + nums[left] + nums[right]
                if total == target:
                    result.append(curr_nums + [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 < target:
                    left += 1
                else:
                    right -= 1
            return

        for i in range(start, n - k + 1):
            if i > start and nums[i] == nums[i - 1]:
                continue
            curr_nums.append(nums[i])
            kSum(i + 1, k - 1, curr_sum + nums[i], curr_nums)
            curr_nums.pop()

    kSum(0, 4, 0, [])
    return result
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]]
print(fourSum([], 0)) # Output: []
```

Output:



```
C:\Users\saisr\AppData\Local\Microsoft\WindowsApps\python3.10.exe
C:\Users\saisr\Downloads\assignments\assignment2\4sum.py
[[-2, -1, 1, 2], [-2, 0, 0, 2], [-1, 0, 0, 1]]
[[2, 2, 2, 2]]
[]
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.

Coding:

```
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 the case when we need to remove
    the head
    dummy = ListNode(0)
    dummy.next = head

    # Initialize two pointers
    first = dummy
    second = dummy

    # Move the second pointer n+1 steps ahead
    for _ in range(n + 1):
        if not second.next:
            # If the list has fewer than n+1 nodes, return the
original list
            return head
        second = second.next

    # Move both pointers until the second pointer reaches the end
    while second:
        first = first.next
        second = second.next

    # Remove the nth node from the end by skipping its next pointer
    first.next = first.next.next

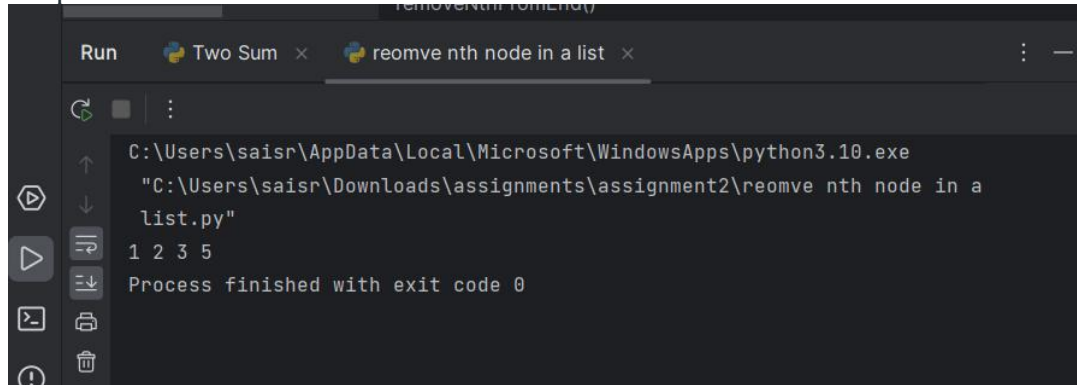
    return dummy.next

# Create the linked list: 1 -> 2 -> 3 -> 4 -> 5
node5 = ListNode(5)
node4 = ListNode(4, node5)
node3 = ListNode(3, node4)
node2 = ListNode(2, node3)
node1 = ListNode(1, node2)

# Remove the 2nd node from the end
new_head = removeNthFromEnd(node1, 2)

# Print the modified linked list
current = new_head
while current:
    print(current.val, end=" ")
    current = current.next
# Output: 1 2 3 5
```

Output:



```
Run Two Sum x remove nth node in a list x
C:\Users\saisr\AppData\Local\Microsoft\WindowsApps\python3.10.exe
"C:\Users\saisr\Downloads\assignments\assignment2\remove nth node in a list.py"
1 2 3 5
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.

Coding:

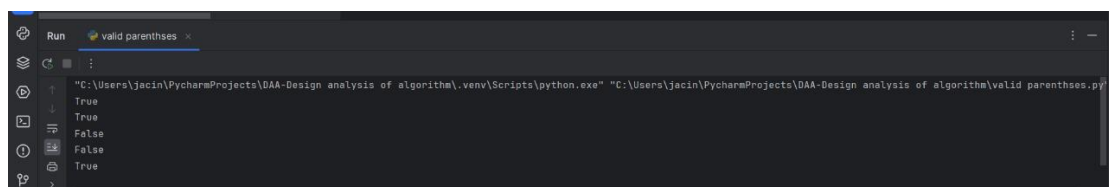
```
def isValid(s):
    stack = []
    mapping = {")": "(", "}": "{", "]": "["}

    for char in s:
        if char in mapping.values():
            stack.append(char)
        else:
            if not stack or stack.pop() != mapping[char]:
                return False

    return not stack

print(isValid("()")) # Output: True
print(isValid("()[]{}")) # Output: True
print(isValid("(]")) # Output: False
print(isValid("([)]")) # Output: False
print(isValid("{[]}")) # Output: True
```

Output:



```
Run valid parentheses x
True
True
False
False
True
```