ASSIGNMENT -2

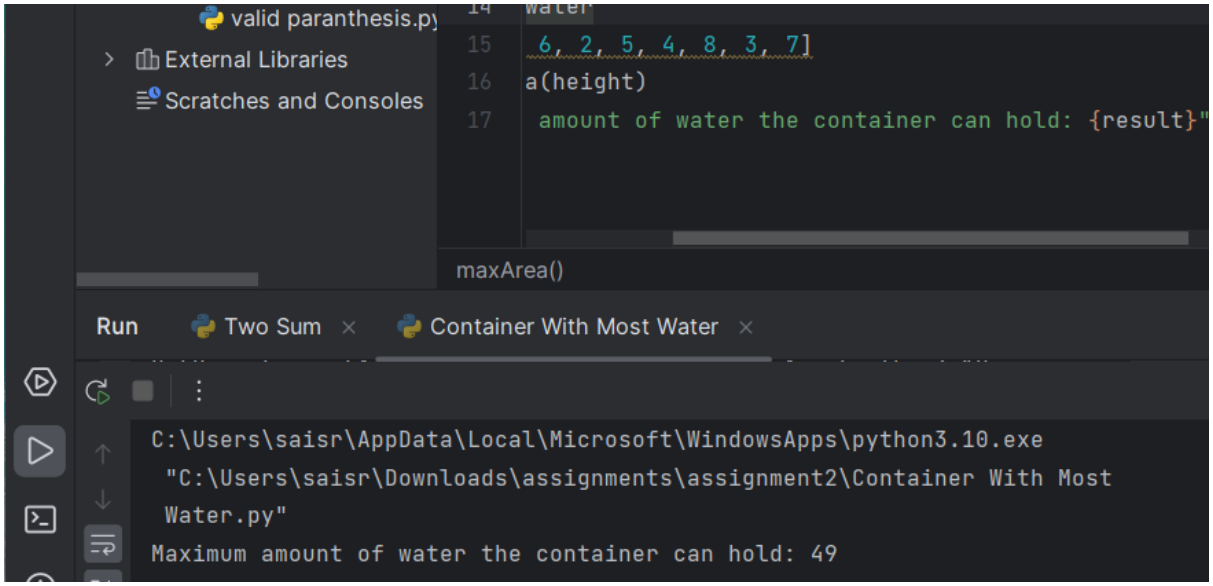
<u>11. Container With Most Water</u>

You are given an integer array height of length n. There are n vertical lines drawn such that thetwo 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 themost water.
Return *the maximum amount of water a container can store*.
Notice that you may not slant the container.

Coding:

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

    while left < right:
        area = min(height[left], height[right]) * (right - left)
        max_water = max(max_water, area)

        if height[left] < height[right]:
            left += 1
        else:
            right -= 1

    return max_water
height = [1, 8, 6, 2, 5, 4, 8, 3, 7]
result = maxArea(height)
print(f"Maximum amount of water the container can hold: {result}")
```

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 fivewe 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:

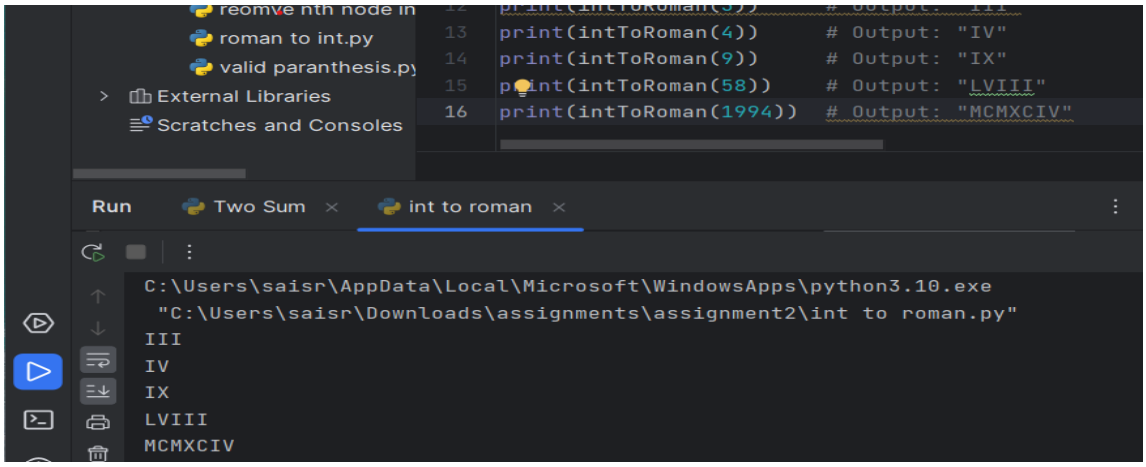
- **I can be placed before V (5) and X (10) to make 4 and 9.**
- **X can be placed before L (50) and C (100) to make 40 and 90.**
- **C can be placed before D (500) and M (1000) to make 400 and 900.**

Given an integer, convert it to a roman numeral.

Coding:

```
def intToRoman(num):
    values = [1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1]
    numerals = ["M", "CM", "D", "CD", "C", "XC", "L", "XL", "X", "IX", "V",
"IV", "I"]
    result = ""
    for i, v in enumerate(values):
        count = int(num / v)
        result += (numerals[i] * count)
        num -= v * count
    return result

# Example usage
print(intToRoman(3))      # Output: "III"
print(intToRoman(4))      # Output: "IV"
print(intToRoman(9))      # Output: "IX"
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.**

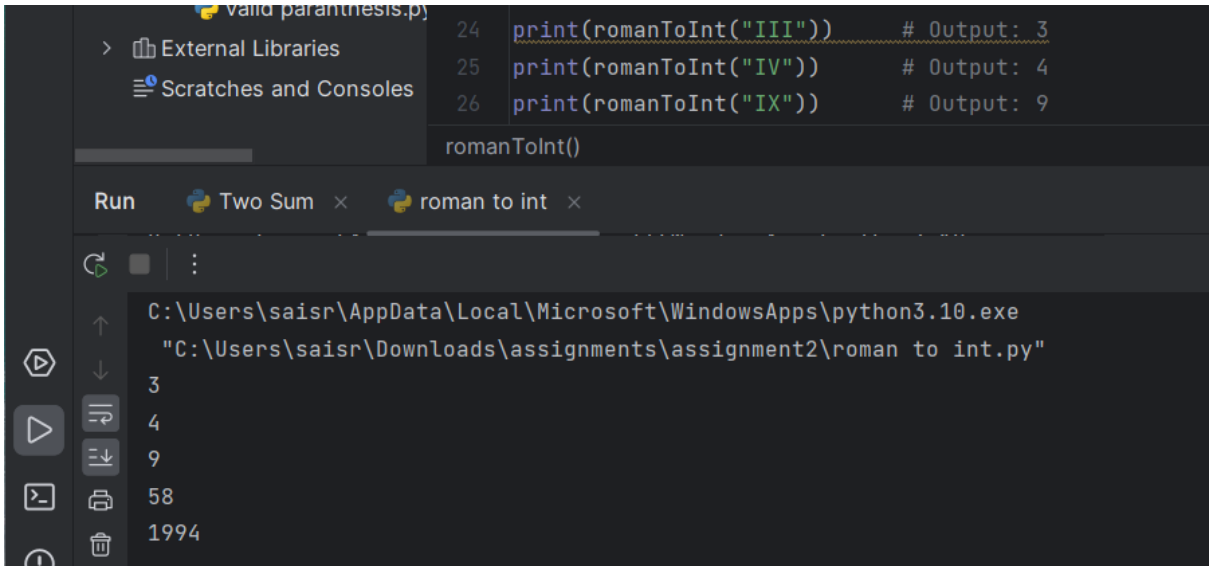
Given a roman numeral, convert it to an integer.

Coding:

```
def romanToInt(s):
    roman_values = {
        "I": 1,
        "V": 5,
        "X": 10,
        "L": 50,
        "C": 100,
        "D": 500,
```

```python
        "M": 1000
    }
    total = 0
    prev_value = 0

    for char in s[::-1]:
        current_value = roman_values[char]
        if current_value < prev_value:
            total -= current_value
        else:
            total += current_value
        prev_value = current_value

    return total

print(romanToInt("III"))      # Output: 3
print(romanToInt("IV"))       # Output: 4
print(romanToInt("IX"))       # Output: 9
print(romanToInt("LVIII"))    # Output: 58
print(romanToInt("MCMXCIV")) # Output: 1994
```

Output:



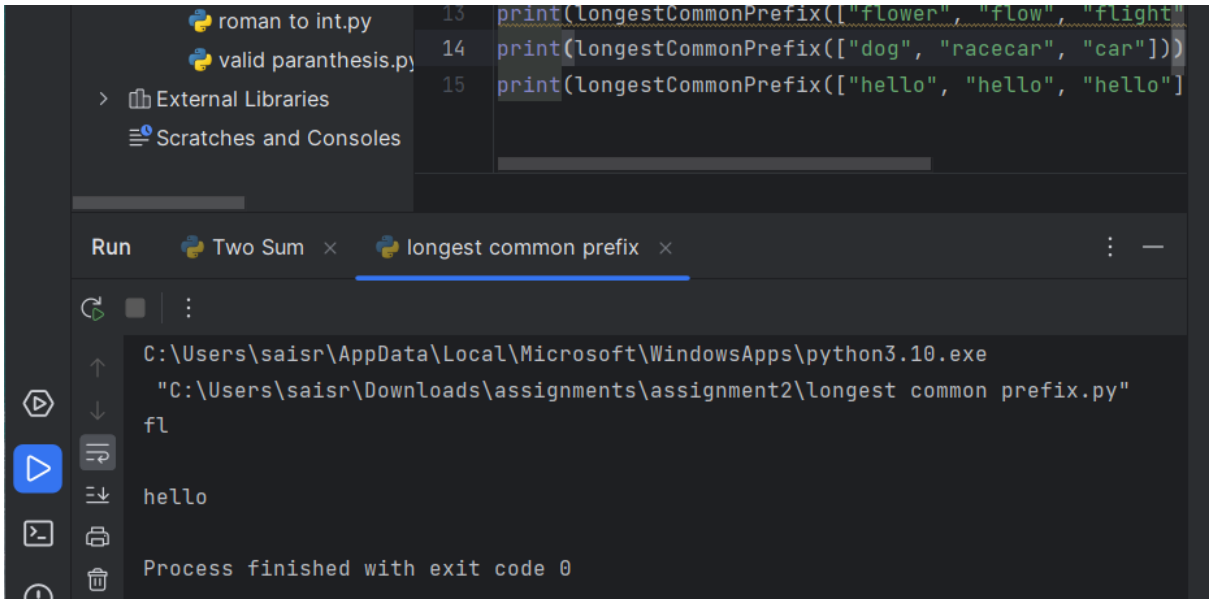
14. Longest Common Prefix

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 `""`.

Coding:

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

    shortest = min(strs, key=len)

    for i, char in enumerate(shortest):
        for s in strs:
            if s[i] != char:
                return shortest[:i]

    return shortest
print(longestCommonPrefix(["flower", "flow", "flight"]))  # Output: "fl"
print(longestCommonPrefix(["dog", "racecar", "car"]))     # Output: ""
print(longestCommonPrefix(["hello", "hello", "hello"]))   # Output: "hello"
```

Output:



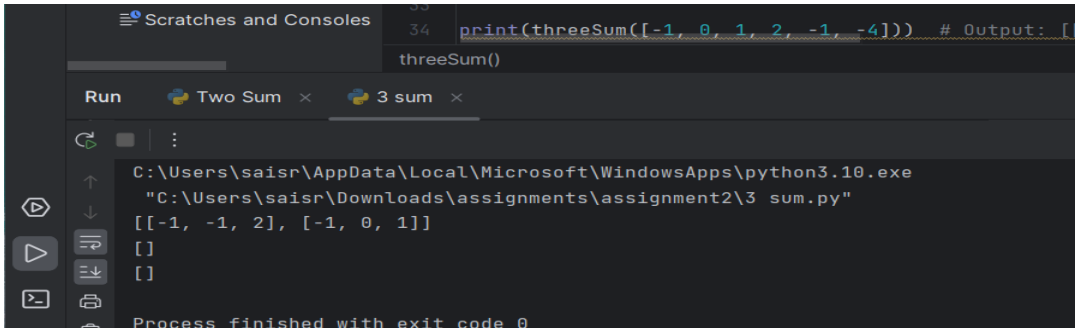
15. 3Sum

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.

Coding:

```python
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: []
```

Ouput:



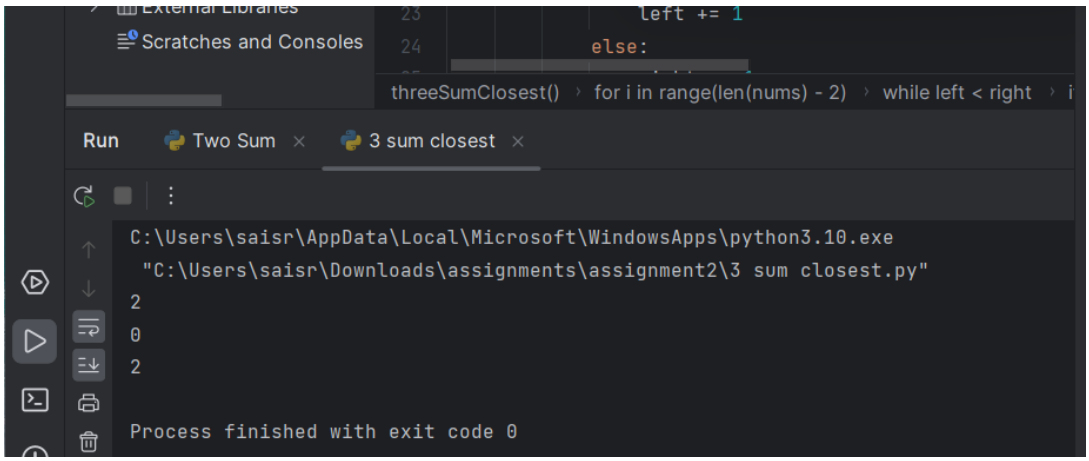
16. 3Sum Closest

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:



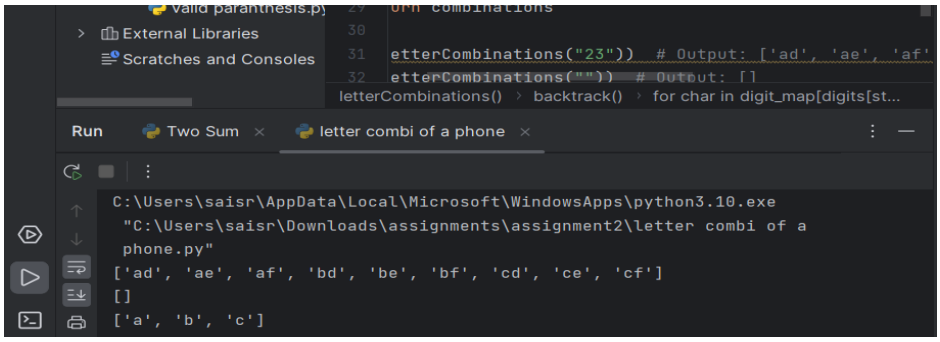
17. Letter Combinations of a Phone Number

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:

```python
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:



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.

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
```

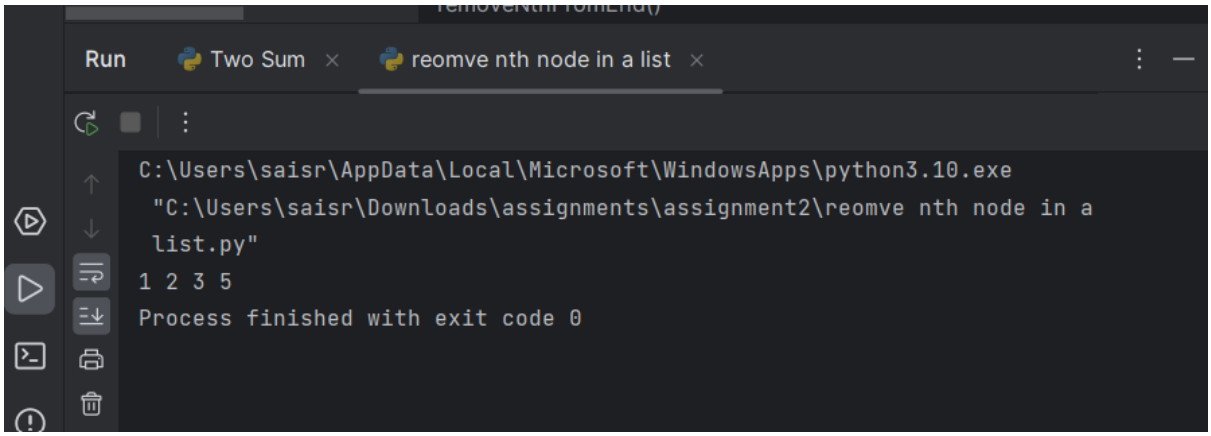
19.Remove Nth Node From End of List

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:



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 Two Sum valid paranthesis

```
C:\Users\saisr\AppData\Local\Microsoft\WindowsApps\python3.10.exe
 "C:\Users\saisr\Downloads\assignments\assignment2\valid paranthesis.py"
True
True
False
False
True

Process finished with exit code 0
```