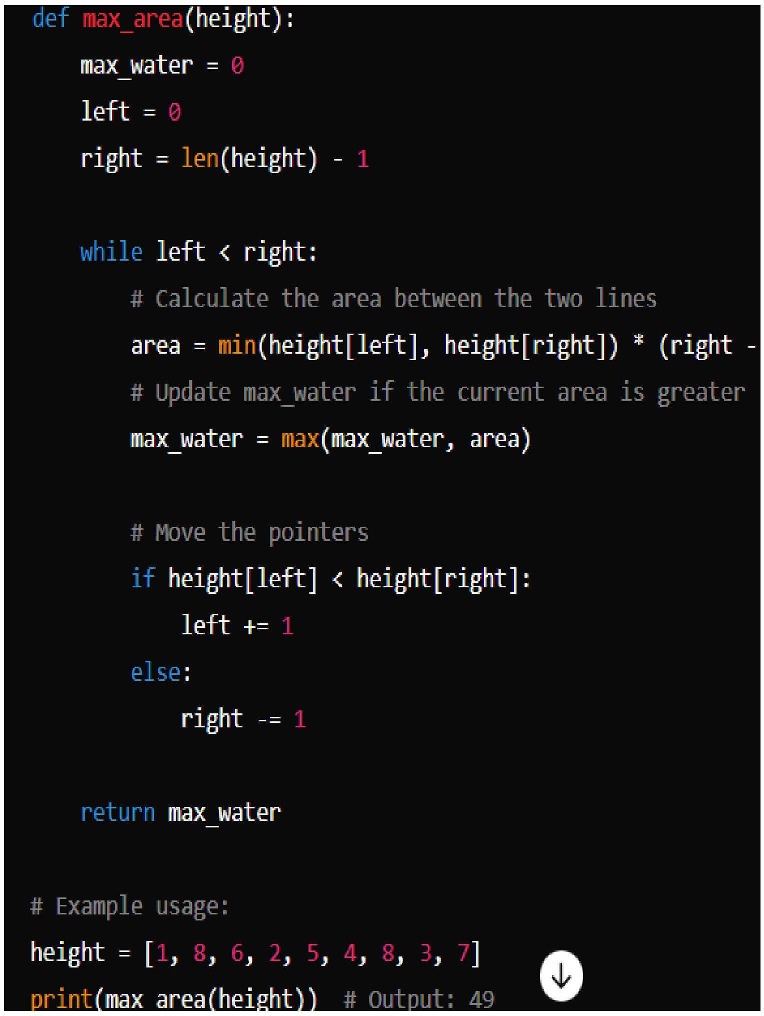
1. Container With Most Water (Two Pointers)

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.

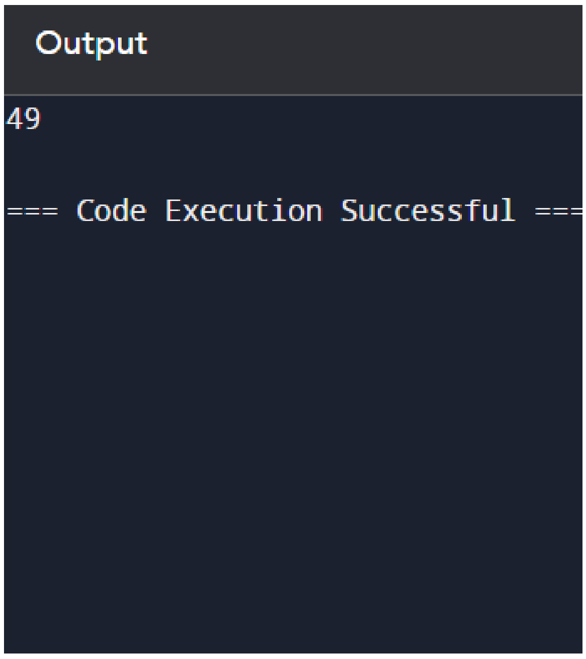
Input: height = [1,8,6,2,5,4,8,3,7] Output: 49

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.

Program:



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

    while left < right:
        # Calculate the area between the two lines
        area = min(height[left], height[right]) * (right -
        # Update max_water if the current area is greater
        max_water = max(max_water, area)

        # Move the pointers
        if height[left] < height[right]:
            left += 1
        else:
            right -= 1

    return max_water

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

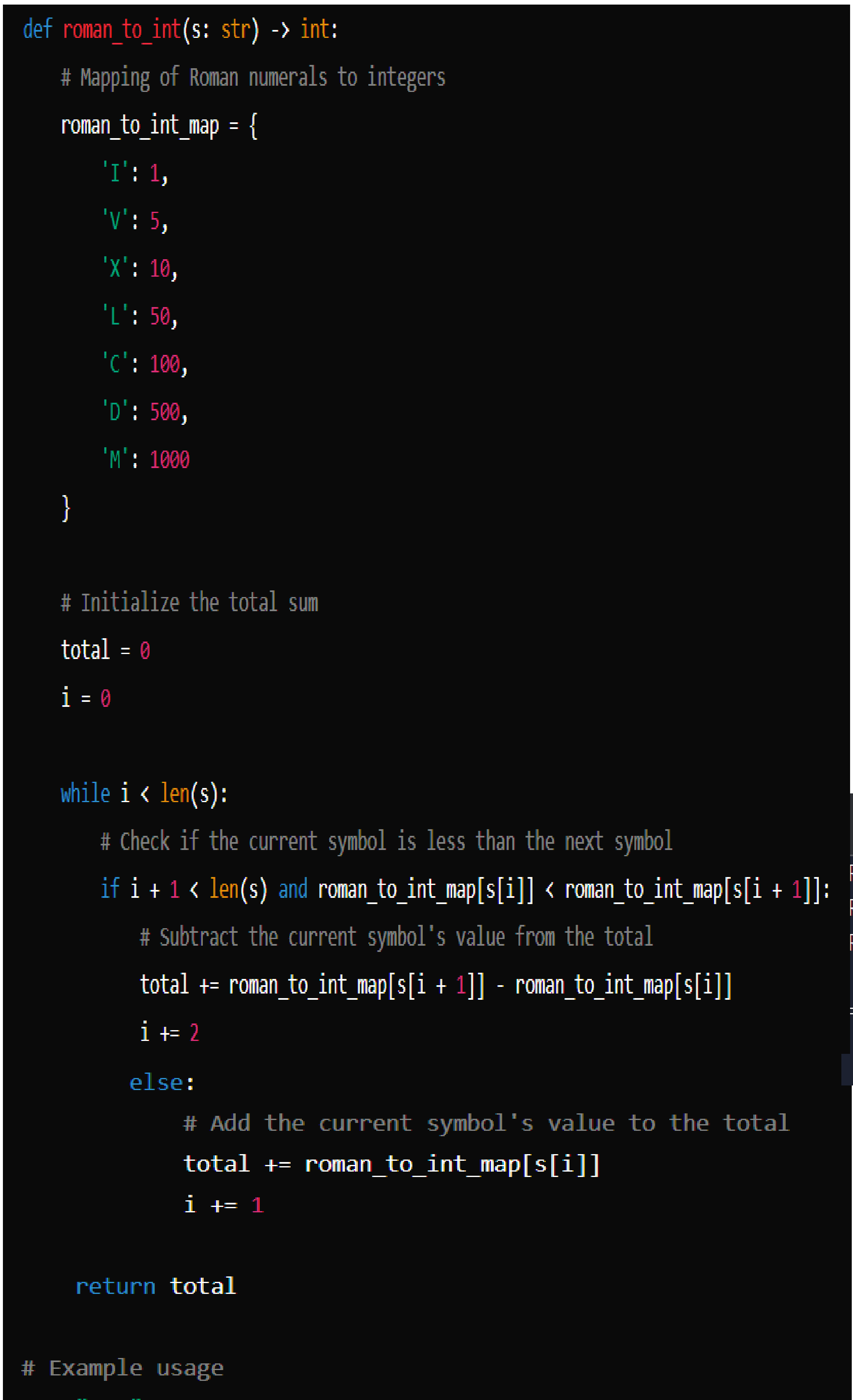


```
49

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

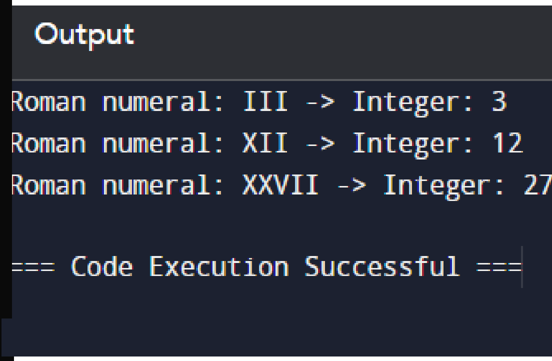
2.Roman to Integer

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
Example 1:
Input: s = "III" Output: 3
Explanation: III = 3. .

Program:



```python
def roman_to_int(s: str) -> int:
    # Mapping of Roman numerals to integers
    roman_to_int_map = {
        'I': 1,
        'V': 5,
        'X': 10,
        'L': 50,
        'C': 100,
        'D': 500,
        'M': 1000
    }

    # Initialize the total sum
    total = 0
    i = 0

    while i < len(s):
        # Check if the current symbol is less than the next symbol
        if i + 1 < len(s) and roman_to_int_map[s[i]] < roman_to_int_map[s[i + 1]]:
            # Subtract the current symbol's value from the total
            total += roman_to_int_map[s[i + 1]] - roman_to_int_map[s[i]]
            i += 2
        else:
            # Add the current symbol's value to the total
            total += roman_to_int_map[s[i]]
            i += 1

    return total

# Example usage
```



Output

```
Roman numeral: III -> Integer: 3
Roman numeral: XII -> Integer: 12
Roman numeral: XXVII -> Integer: 27

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

```python
s = "XII"
print(f"Roman numeral: {s} -> Integer: {roman_to_int(s)}")

s = "XXVII"
print(f"Roman numeral: {s} -> Integer: {roman_to_int(s)}")
```

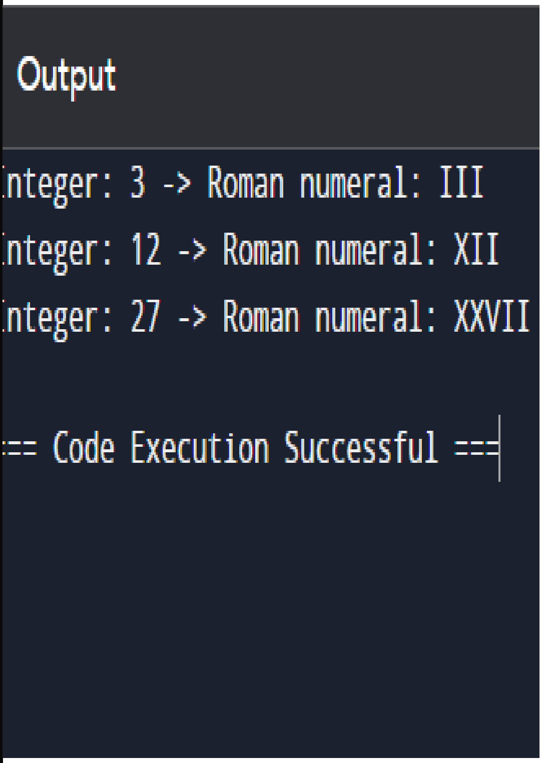
3. Integer to Roman

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

Example 1: Input: num = 3 Output: "III"

Explanation: 3 is represented as 3 ones.

```python
def int_to_roman(num: int) -> str:
    # Mapping of integer values to Roman numeral symbols
    int_to_roman_map = [
        (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')
    ]

    # Initialize the result string
    result = []
    for value, symbol in int_to_roman_map:
        while num >= value:
            result.append(symbol)
            num -= value

    return ''.join(result)

# Example usage
num = 3
print(f"Integer: {num} -> Roman numeral: {int_to_roman(num)}")

num = 12
print(f"Integer: {num} -> Roman numeral: {int_to_roman(num)}")

num = 27
print(f"Integer: {num} -> Roman numeral: {int_to_roman(num)}")
```



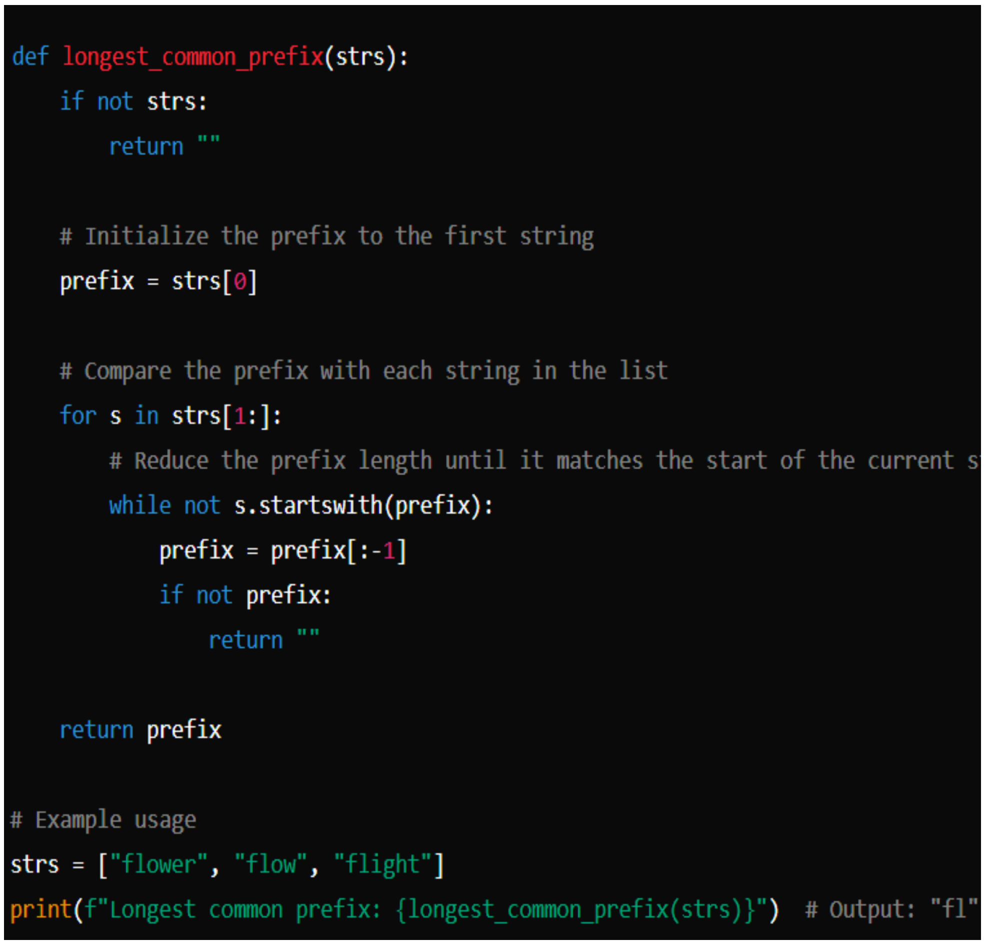
Output

```
nteger: 3 -> Roman numeral: III
nteger: 12 -> Roman numeral: XII
nteger: 27 -> Roman numeral: XXVII

== Code Execution Successful ===
```

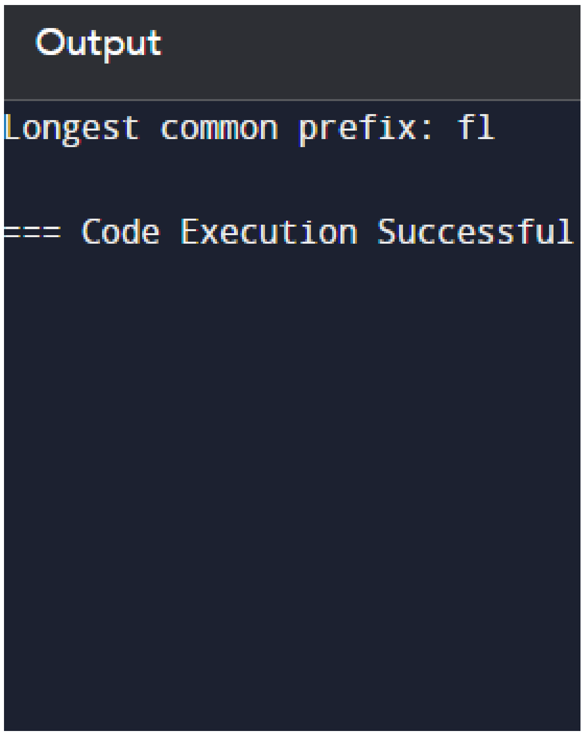
4. Longest Common Prefix

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:



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

    # Initialize the prefix to the first string
    prefix = strs[0]

    # Compare the prefix with each string in the list
    for s in strs[1:]:
        # Reduce the prefix length until it matches the start of the current s
        while not s.startswith(prefix):
            prefix = prefix[:-1]
            if not prefix:
                return ""

    return prefix

# Example usage
strs = ["flower", "flow", "flight"]
print(f"Longest common prefix: {longest_common_prefix(strs)}")  # Output: "fl"
```



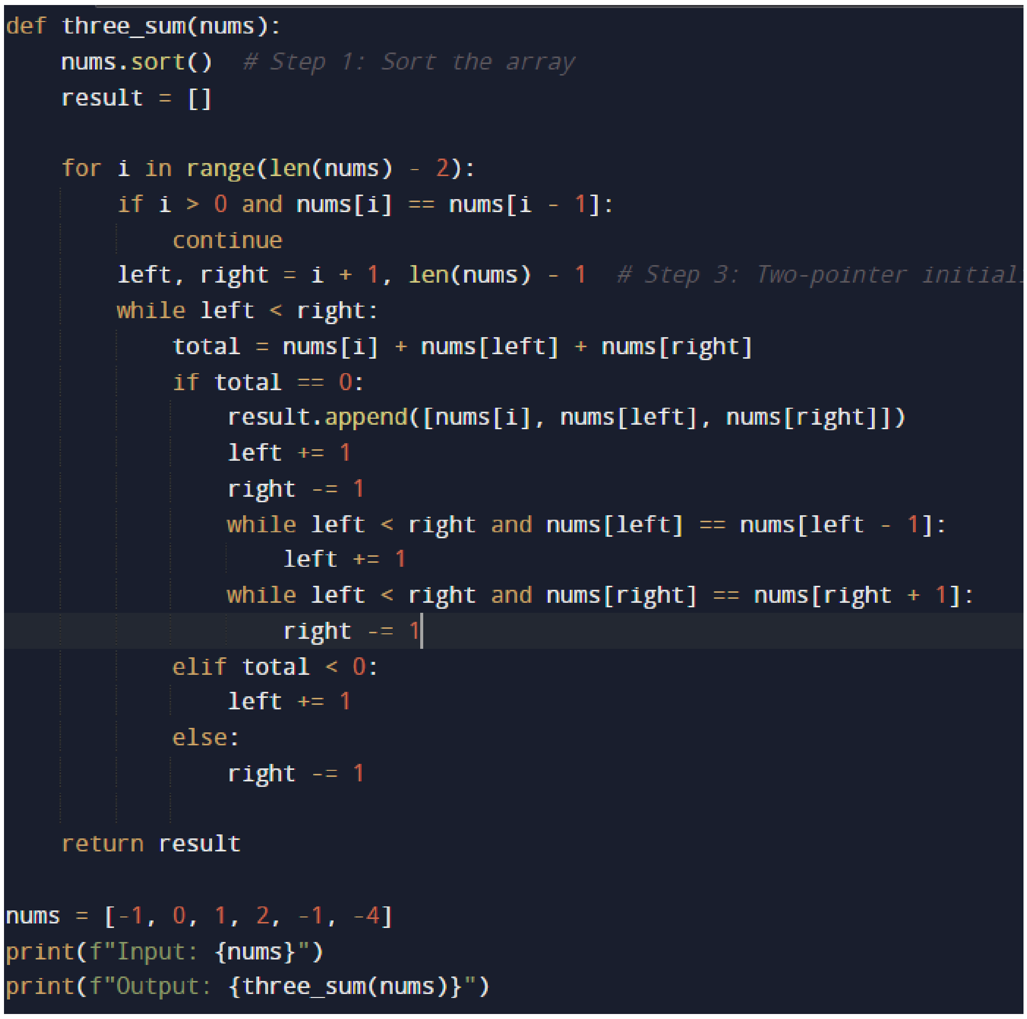
Output

```
Longest common prefix: fl

=== Code Execution Successful
```

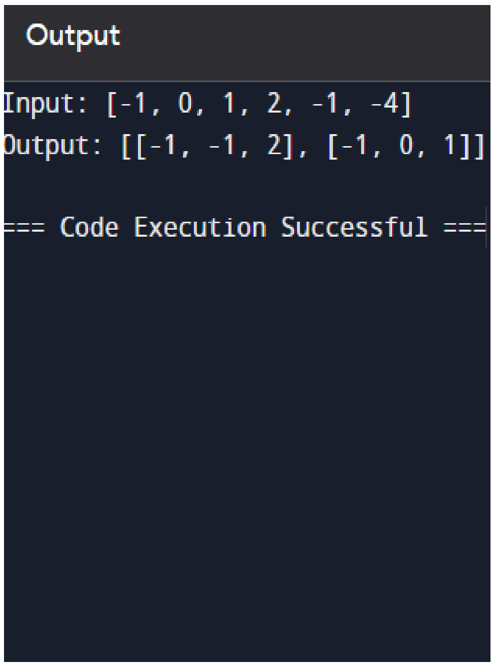
5. 3Sum

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:



```python
def three_sum(nums):
    nums.sort()  # Step 1: Sort the array
    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  # Step 3: Two-pointer initial
        while left < right:
            total = nums[i] + nums[left] + nums[right]
            if total == 0:
                result.append([nums[i], nums[left], nums[right]])
                left += 1
                right -= 1
                while left < right and nums[left] == nums[left - 1]:
                    left += 1
                while left < right and nums[right] == nums[right + 1]:
                    right -= 1
            elif total < 0:
                left += 1
            else:
                right -= 1

    return result

nums = [-1, 0, 1, 2, -1, -4]
print(f"Input: {nums}")
print(f"Output: {three_sum(nums)}")
```



Output

```
Input: [-1, 0, 1, 2, -1, -4]
Output: [[-1, -1, 2], [-1, 0, 1]]

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

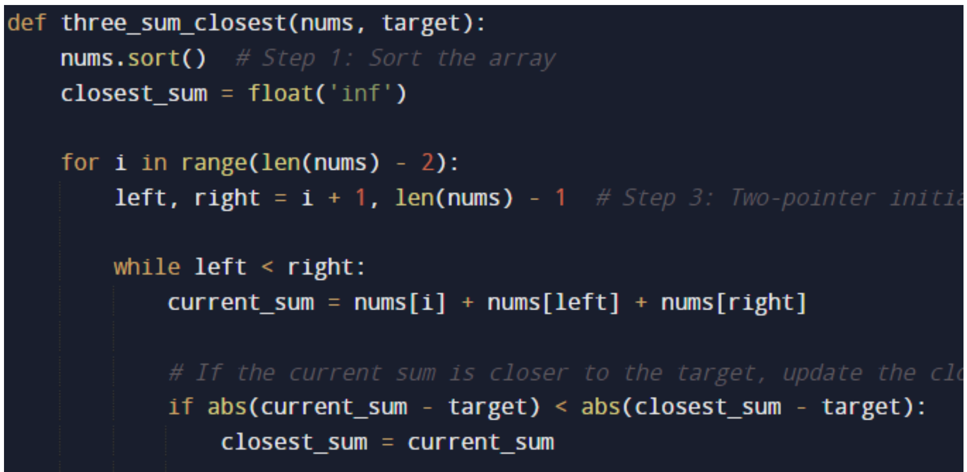
6. 3Sum Closest

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.

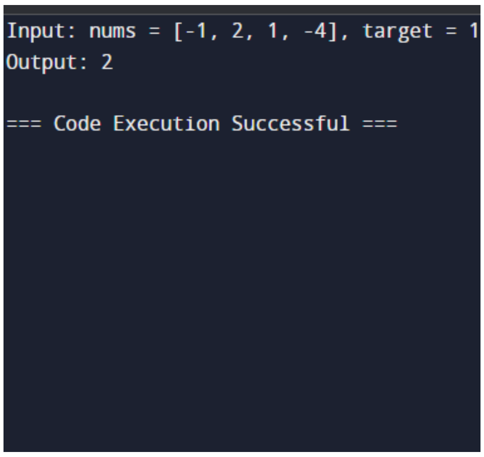
Example 1: Input: nums = [-1,2,1,-4], target = 1 Output: 2

Explanation: The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).

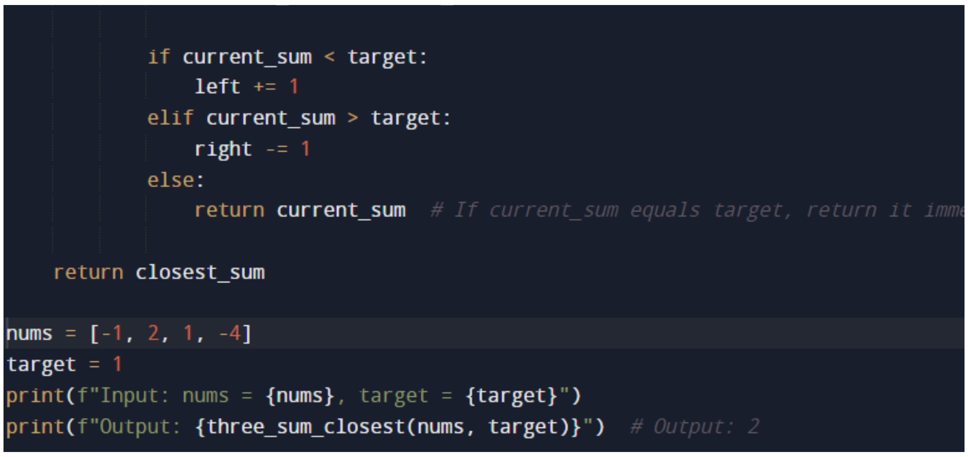
Program:



```python
def three_sum_closest(nums, target):
    nums.sort()  # Step 1: Sort the array
    closest_sum = float('inf')

    for i in range(len(nums) - 2):
        left, right = i + 1, len(nums) - 1  # Step 3: Two-pointer initia

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

            # If the current sum is closer to the target, update the clo
            if abs(current_sum - target) < abs(closest_sum - target):
                closest_sum = current_sum
```



```
Input: nums = [-1, 2, 1, -4], target = 1
Output: 2

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



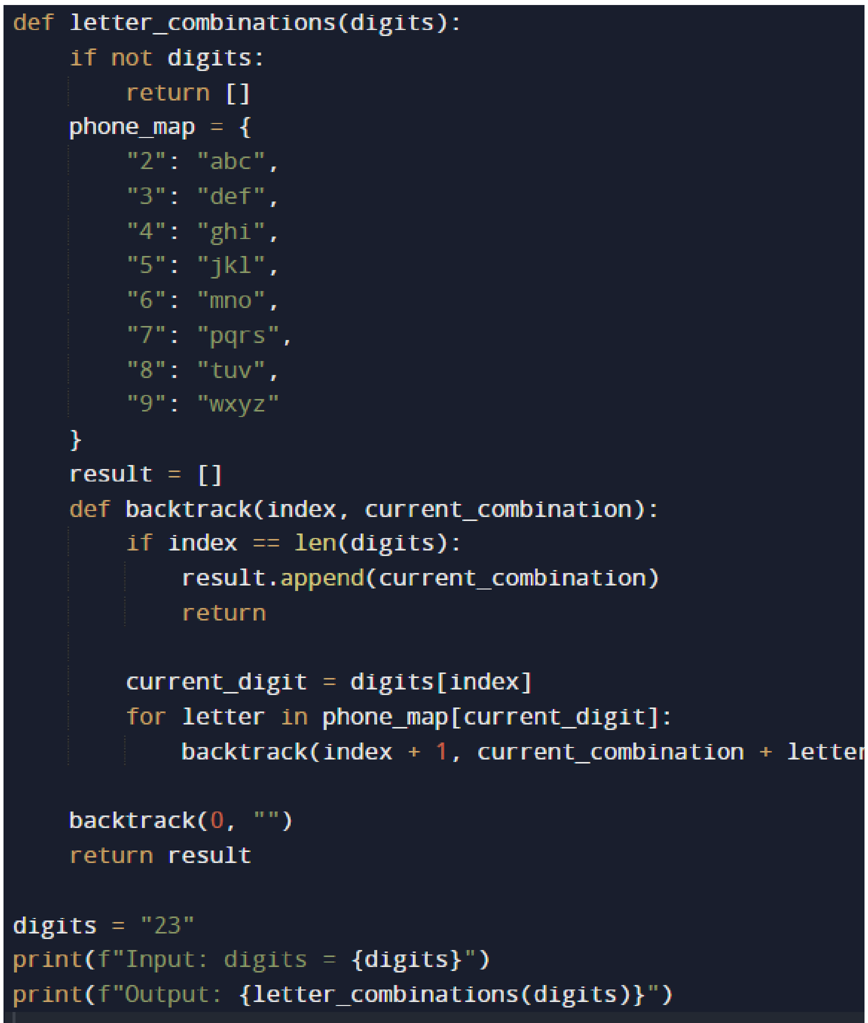
```python
            if current_sum < target:
                left += 1
            elif current_sum > target:
                right -= 1
            else:
                return current_sum  # If current_sum equals target, return it imme

    return closest_sum

nums = [-1, 2, 1, -4]
target = 1
print(f"Input: nums = {nums}, target = {target}")
print(f"Output: {three_sum_closest(nums, target)}")  # Output: 2
```

7. Letter Combinations of a Phone Number

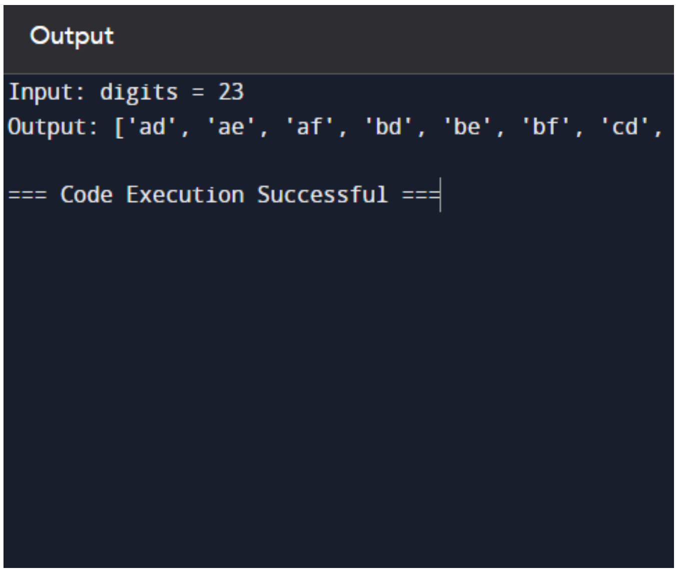
Given a string containing digits from 2-9 inclusive, return all possible letter combinations that the number could represent. Return the answer in any order. A mapping of digits to letters (just like on the telephone buttons) is given below. Note that 1 does not map to any letters..

Example 1: Input: digits = "23" Output: ["ad","ae","af","bd","be","bf","cd","ce","cf"]

Program:



```python
def letter_combinations(digits):
    if not digits:
        return []
    phone_map = {
        "2": "abc",
        "3": "def",
        "4": "ghi",
        "5": "jkl",
        "6": "mno",
        "7": "pqrs",
        "8": "tuv",
        "9": "wxyz"
    }
    result = []
    def backtrack(index, current_combination):
        if index == len(digits):
            result.append(current_combination)
            return

        current_digit = digits[index]
        for letter in phone_map[current_digit]:
            backtrack(index + 1, current_combination + letter

    backtrack(0, "")
    return result

digits = "23"
print(f"Input: digits = {digits}")
print(f"Output: {letter_combinations(digits)}")
```



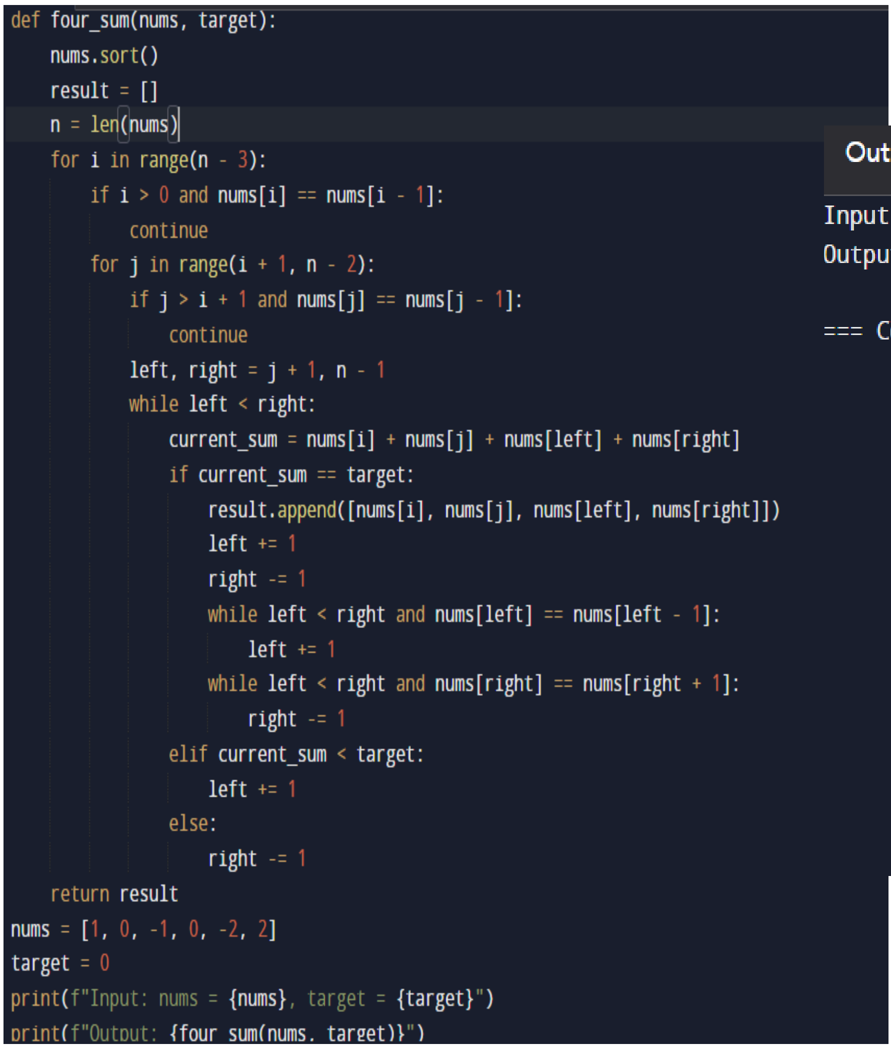
Output

```
Input: digits = 23
Output: ['ad', 'ae', 'af', 'bd', 'be', 'bf', 'cd',

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

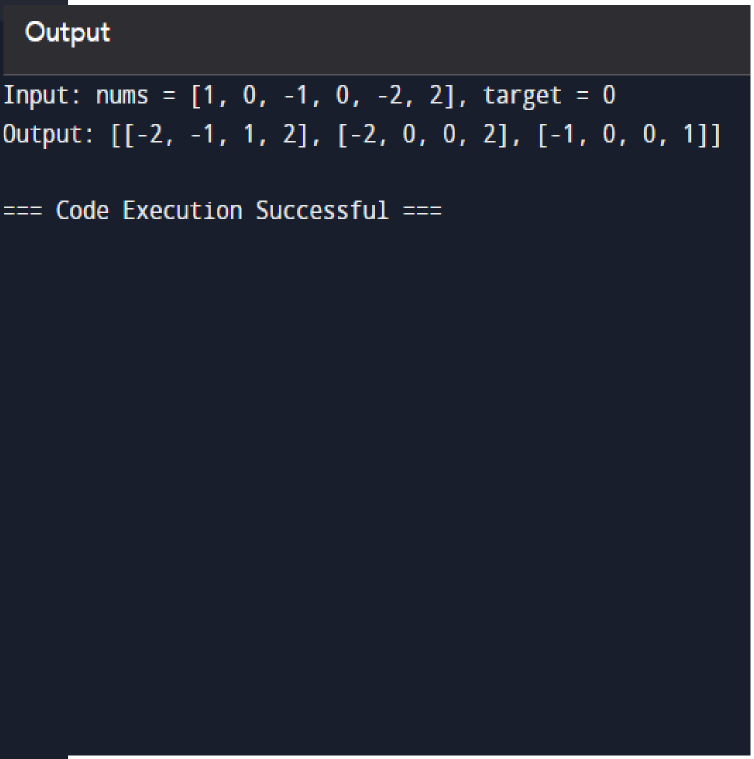
8. 4Sum

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:
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:



```python
def four_sum(nums, target):
    nums.sort()
    result = []
    n = len(nums)
    for i in range(n - 3):
        if i > 0 and nums[i] == nums[i - 1]:
            continue
        for j in range(i + 1, n - 2):
            if j > i + 1 and nums[j] == nums[j - 1]:
                continue
            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]])
                    left += 1
                    right -= 1
                    while left < right and nums[left] == nums[left - 1]:
                        left += 1
                    while left < right and nums[right] == nums[right + 1]:
                        right -= 1
                elif current_sum < target:
                    left += 1
                else:
                    right -= 1
    return result
nums = [1, 0, -1, 0, -2, 2]
target = 0
print(f"Input: nums = {nums}, target = {target}")
print(f"Output: {four_sum(nums, target)}")
```



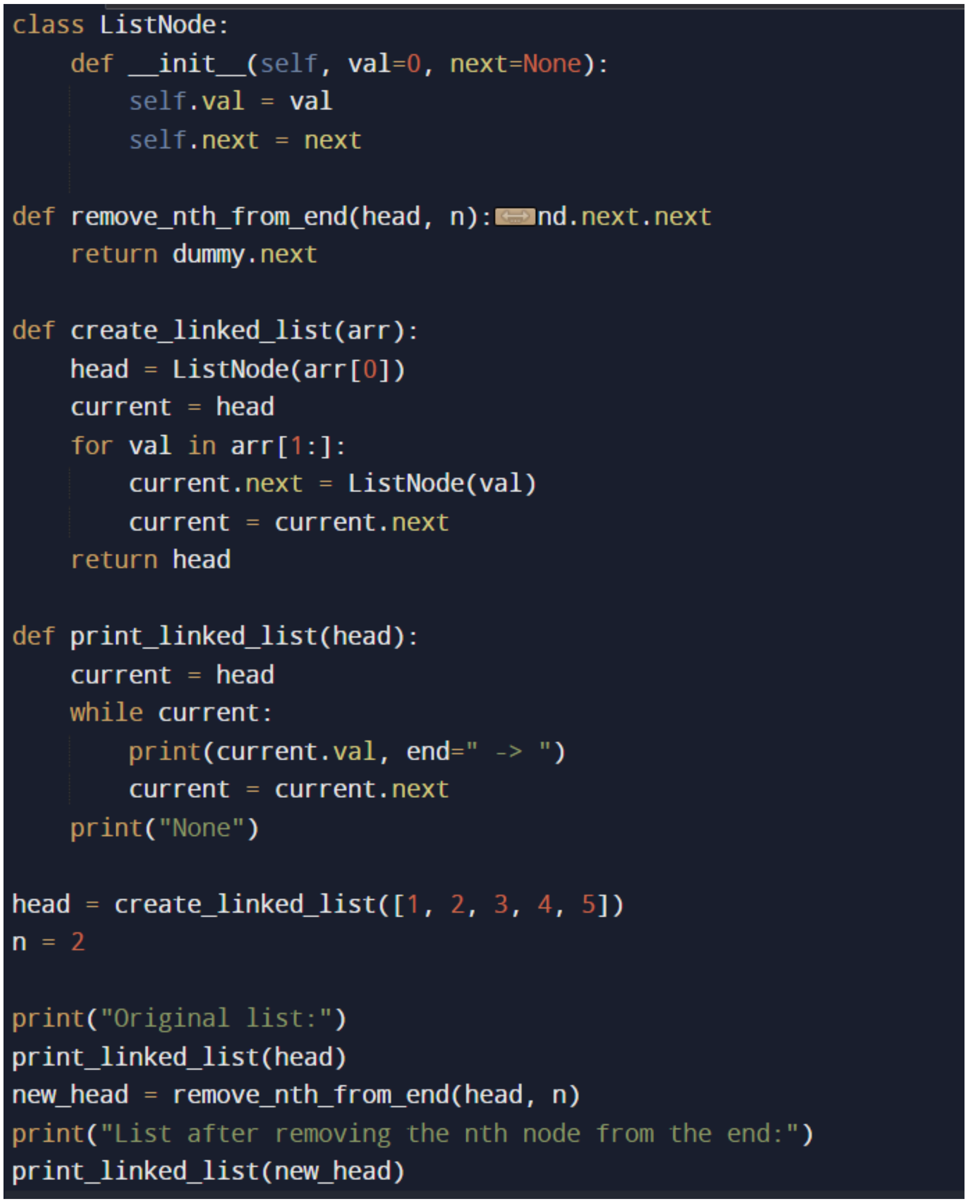
Output

```
Input: nums = [1, 0, -1, 0, -2, 2], target = 0
Output: [[-2, -1, 1, 2], [-2, 0, 0, 2], [-1, 0, 0, 1]]

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

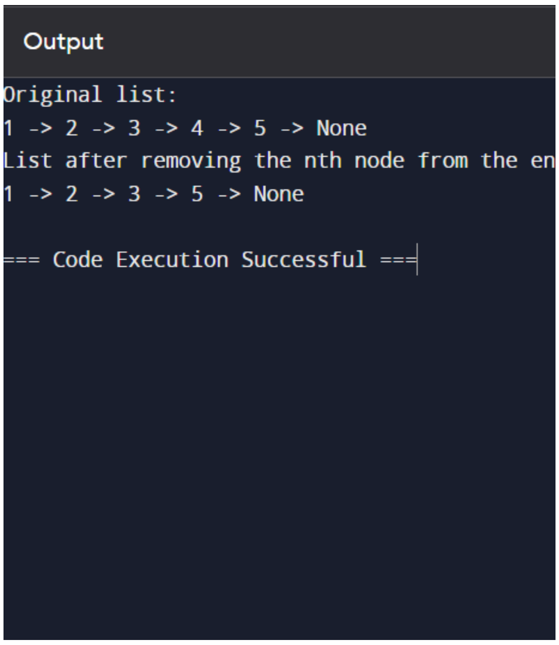
9. Remove Nth Node From End of List

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:



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

def remove_nth_from_end(head, n):⟷nd.next.next
    return dummy.next

def create_linked_list(arr):
    head = ListNode(arr[0])
    current = head
    for val in arr[1:]:
        current.next = ListNode(val)
        current = current.next
    return head

def print_linked_list(head):
    current = head
    while current:
        print(current.val, end=" -> ")
        current = current.next
    print("None")

head = create_linked_list([1, 2, 3, 4, 5])
n = 2

print("Original list:")
print_linked_list(head)
new_head = remove_nth_from_end(head, n)
print("List after removing the nth node from the end:")
print_linked_list(new_head)
```

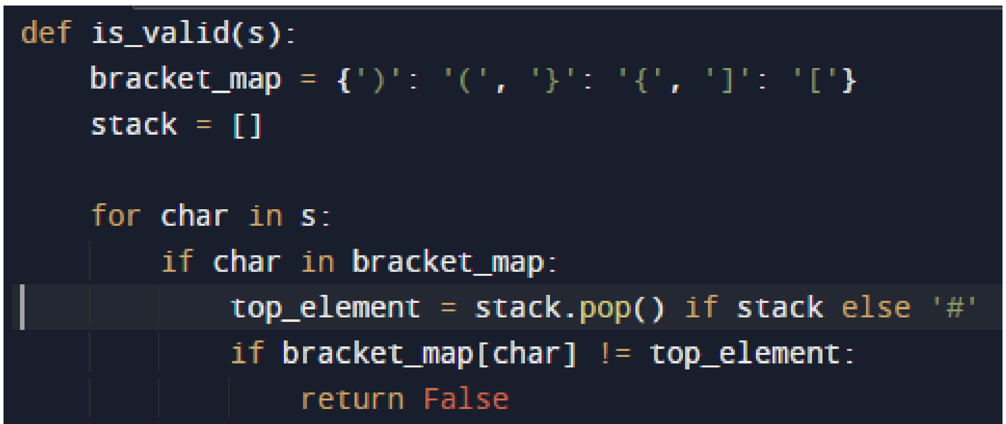


```
Original list:
1 -> 2 -> 3 -> 4 -> 5 -> None
List after removing the nth node from the en
1 -> 2 -> 3 -> 5 -> None

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

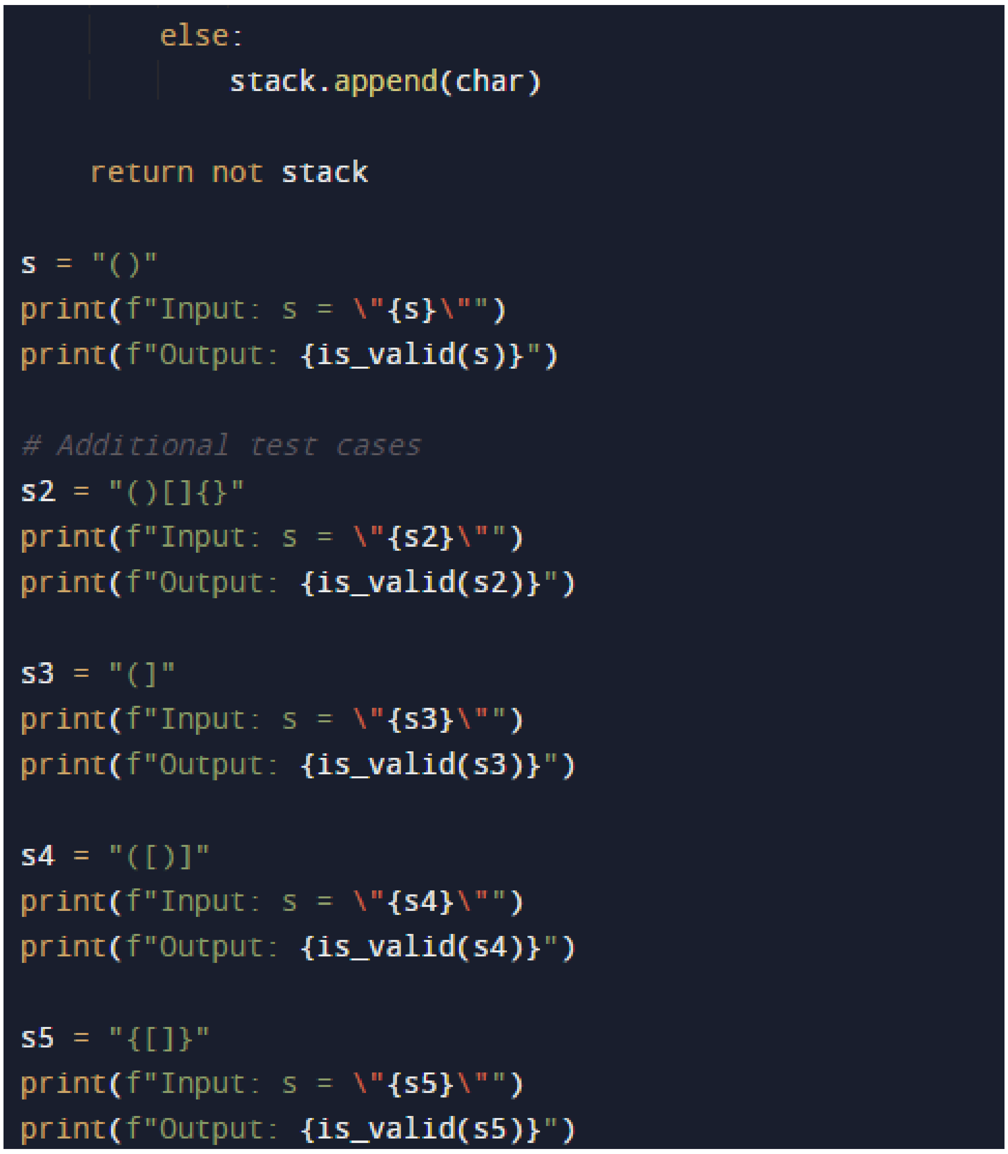
10. Valid Parentheses

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

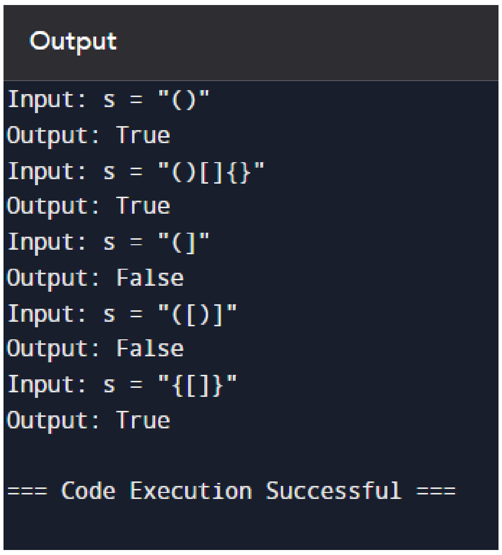
Program:



```python
def is_valid(s):
    bracket_map = {')': '(', '}': '{', ']': '['}
    stack = []

    for char in s:
        if char in bracket_map:
            top_element = stack.pop() if stack else '#'
            if bracket_map[char] != top_element:
                return False
```



```
        else:
            stack.append(char)

    return not stack

s = "()"
print(f"Input: s = \"{s}\"")
print(f"Output: {is_valid(s)}")

# Additional test cases
s2 = "()[]{}"
print(f"Input: s = \"{s2}\"")
print(f"Output: {is_valid(s2)}")

s3 = "(]"
print(f"Input: s = \"{s3}\"")
print(f"Output: {is_valid(s3)}")

s4 = "([)]"
print(f"Input: s = \"{s4}\"")
print(f"Output: {is_valid(s4)}")

s5 = "{[]}"
print(f"Input: s = \"{s5}\"")
print(f"Output: {is_valid(s5)}")
```



Output

```
Input: s = "()"
Output: True
Input: s = "()[]{}"
Output: True
Input: s = "(]"
Output: False
Input: s = "([)]"
Output: False
Input: s = "{[]}"
Output: True

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