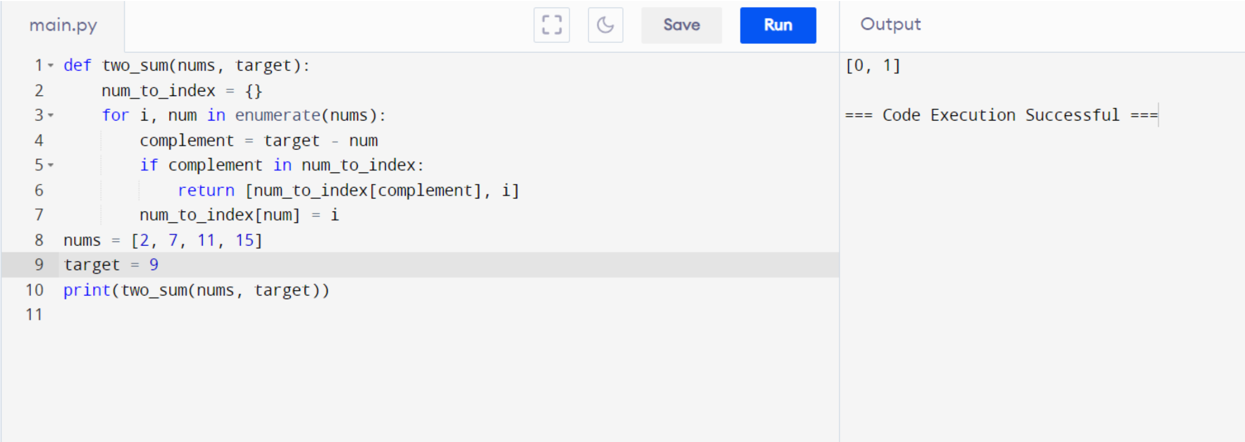
ASSIGNMENT-5

1.Two Sum

Given an array of integers nums and an integer target, return *indices of the two numbers such that they add up to target*.

You may assume that each input would have *exactly* one solution, and you may not use the *same* element twice.

You can return the answer in any order.



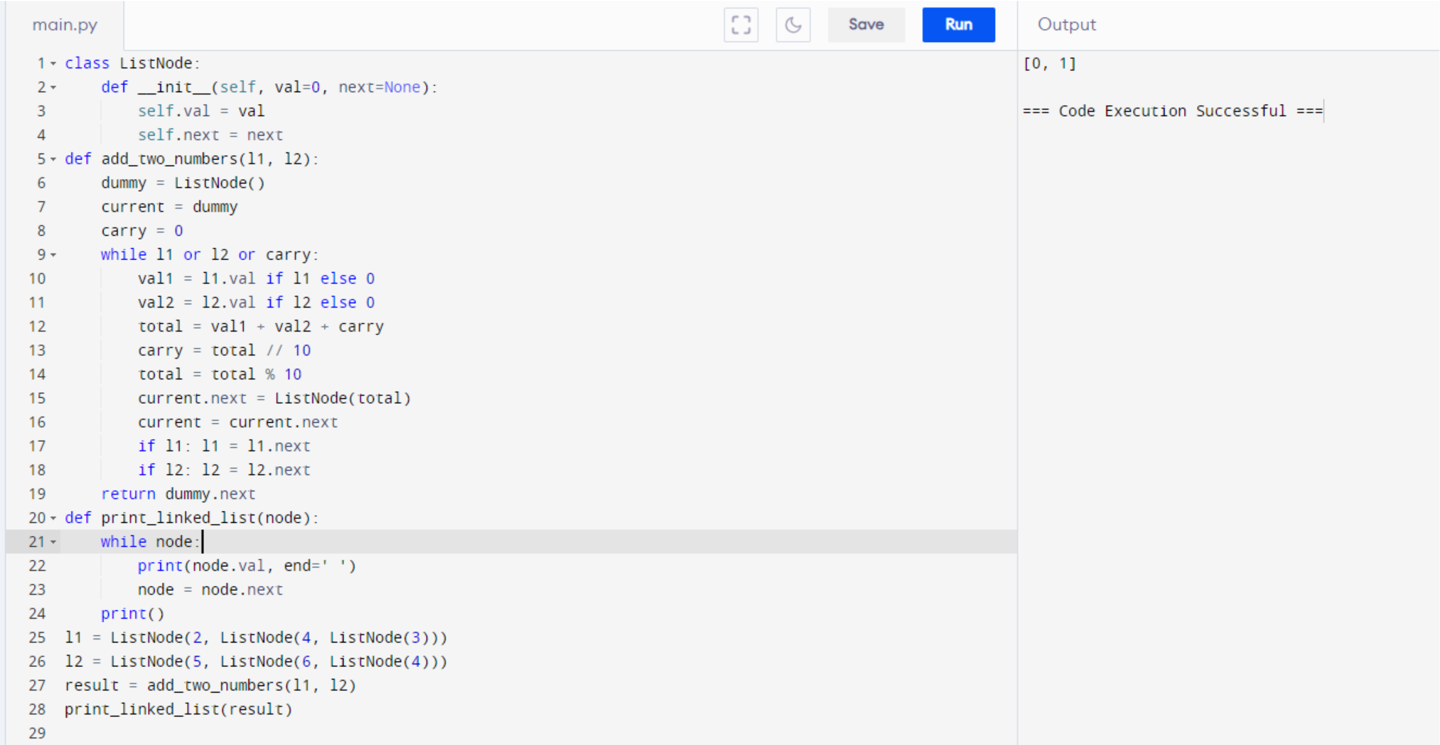
```python
def two_sum(nums, target):
    num_to_index = {}
    for i, num in enumerate(nums):
        complement = target - num
        if complement in num_to_index:
            return [num_to_index[complement], i]
        num_to_index[num] = i
nums = [2, 7, 11, 15]
target = 9
print(two_sum(nums, target))
```

```
[0, 1]

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

2.Add Two Numbers

You are given two non-empty linked lists representing two non-negative integers. The digits are stored in reverse order, and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list.

You may assume the two numbers do not contain any leading zero, except the number 0 itself.

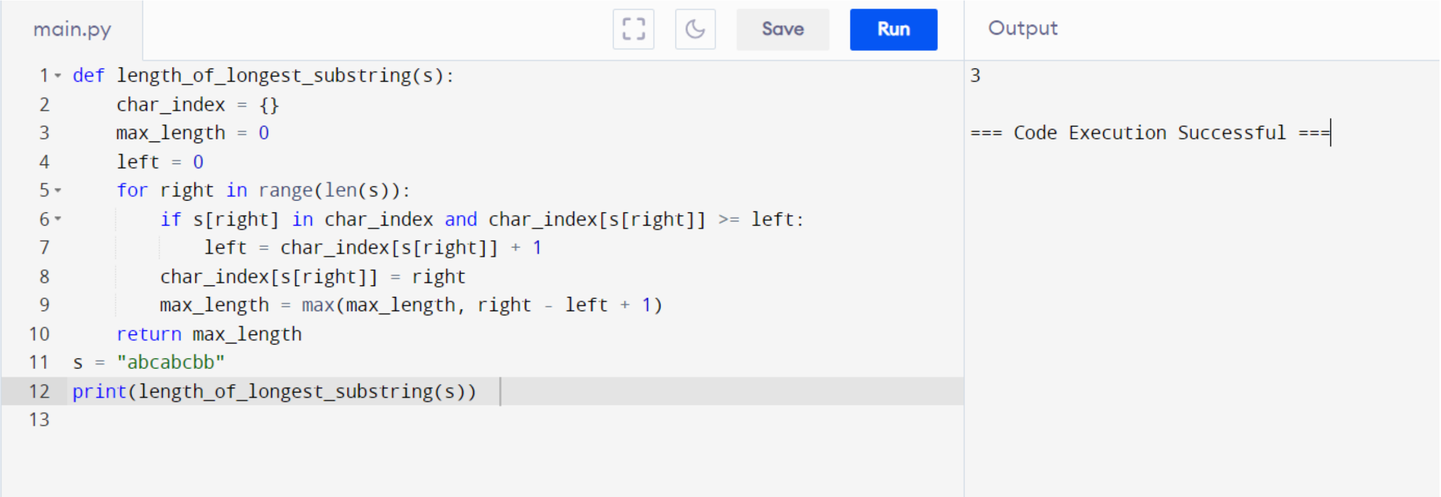


```python
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next
def add_two_numbers(l1, l2):
    dummy = ListNode()
    current = dummy
    carry = 0
    while l1 or l2 or carry:
        val1 = l1.val if l1 else 0
        val2 = l2.val if l2 else 0
        total = val1 + val2 + carry
        carry = total // 10
        total = total % 10
        current.next = ListNode(total)
        current = current.next
        if l1: l1 = l1.next
        if l2: l2 = l2.next
    return dummy.next
def print_linked_list(node):
    while node:
        print(node.val, end=' ')
        node = node.next
    print()
l1 = ListNode(2, ListNode(4, ListNode(3)))
l2 = ListNode(5, ListNode(6, ListNode(4)))
result = add_two_numbers(l1, l2)
print_linked_list(result)
```

```
[0, 1]

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

3. Longest Substring without Repeating Characters

Given a string s, find the length of the longest substring without repeating characters.



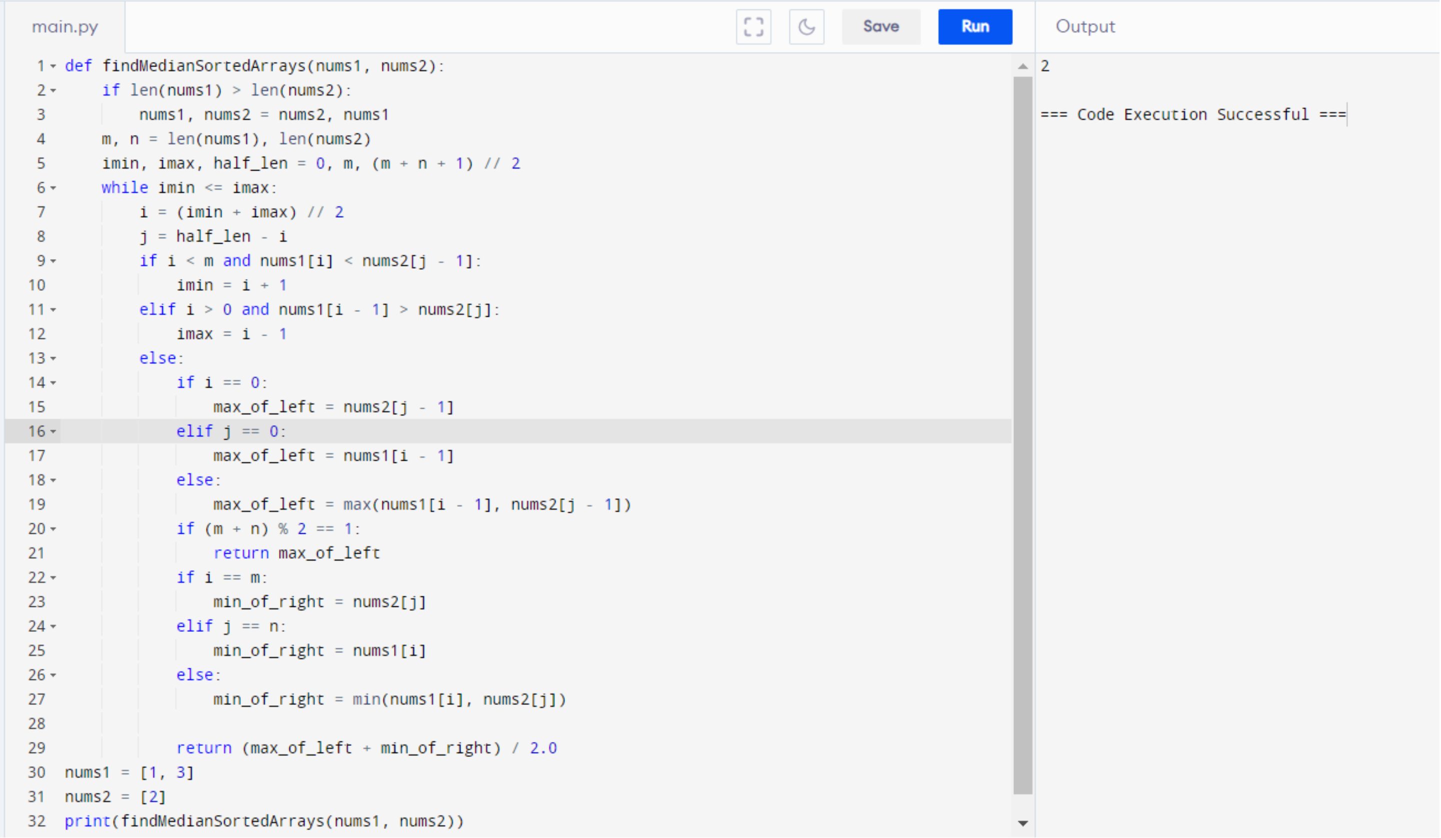
```
def length_of_longest_substring(s):
    char_index = {}
    max_length = 0
    left = 0
    for right in range(len(s)):
        if s[right] in char_index and char_index[s[right]] >= left:
            left = char_index[s[right]] + 1
        char_index[s[right]] = right
        max_length = max(max_length, right - left + 1)
    return max_length
s = "abcabcbb"
print(length_of_longest_substring(s))
```

```
3

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

4. Median of Two Sorted Arrays

Given two sorted arrays nums1 and nums2 of size m and n respectively, return the median of the two sorted arrays.

The overall run time complexity should be O(log (m+n)) .

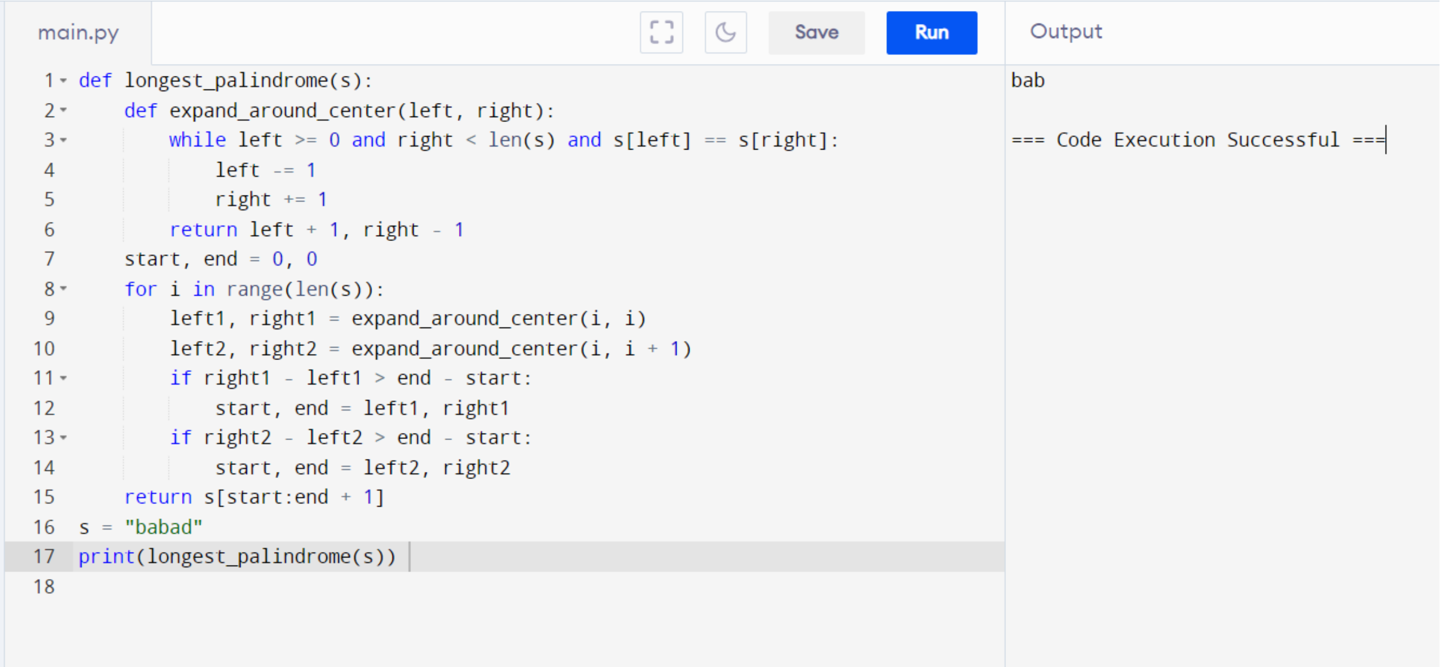


```
def findMedianSortedArrays(nums1, nums2):
    if len(nums1) > len(nums2):
        nums1, nums2 = nums2, nums1
    m, n = len(nums1), len(nums2)
    imin, imax, half_len = 0, m, (m + n + 1) // 2
    while imin <= imax:
        i = (imin + imax) // 2
        j = half_len - i
        if i < m and nums1[i] < nums2[j - 1]:
            imin = i + 1
        elif i > 0 and nums1[i - 1] > nums2[j]:
            imax = i - 1
        else:
            if i == 0:
                max_of_left = nums2[j - 1]
            elif j == 0:
                max_of_left = nums1[i - 1]
            else:
                max_of_left = max(nums1[i - 1], nums2[j - 1])
            if (m + n) % 2 == 1:
                return max_of_left
            if i == m:
                min_of_right = nums2[j]
            elif j == n:
                min_of_right = nums1[i]
            else:
                min_of_right = min(nums1[i], nums2[j])

            return (max_of_left + min_of_right) / 2.0
nums1 = [1, 3]
nums2 = [2]
print(findMedianSortedArrays(nums1, nums2))
```

```
2

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

5. Longest Palindromic Substring

Given a string s, return *the longest palindromic substring* in s.



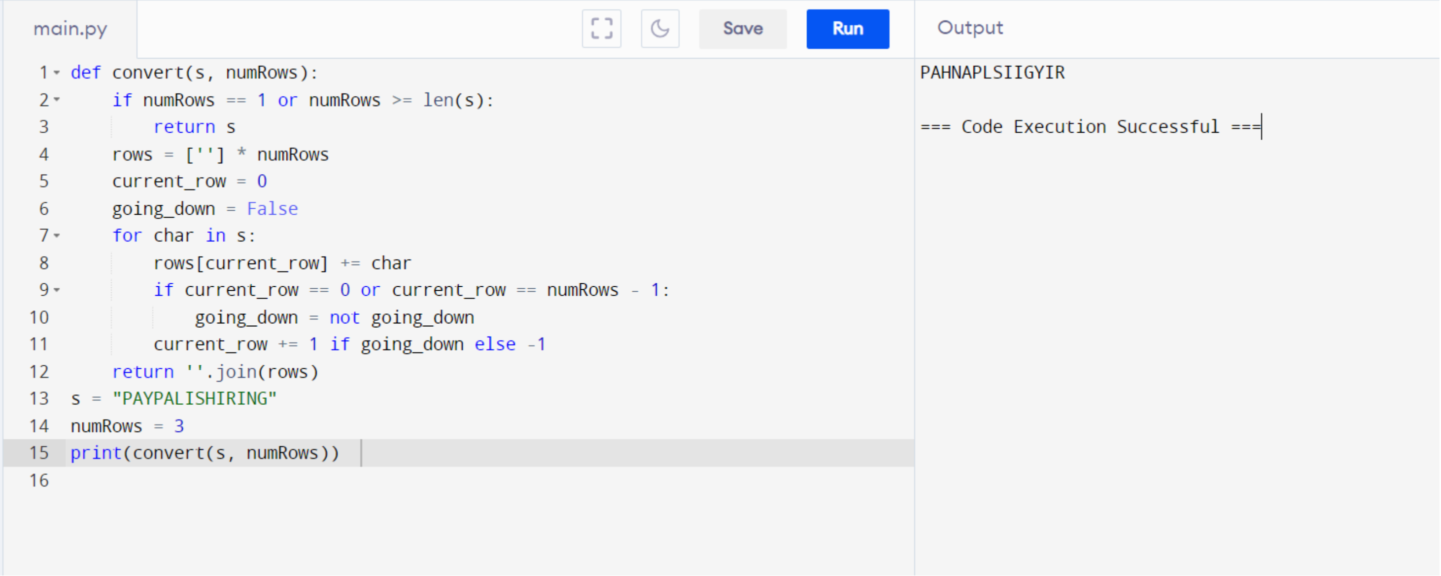
```python
def longest_palindrome(s):
    def expand_around_center(left, right):
        while left >= 0 and right < len(s) and s[left] == s[right]:
            left -= 1
            right += 1
        return left + 1, right - 1
    start, end = 0, 0
    for i in range(len(s)):
        left1, right1 = expand_around_center(i, i)
        left2, right2 = expand_around_center(i, i + 1)
        if right1 - left1 > end - start:
            start, end = left1, right1
        if right2 - left2 > end - start:
            start, end = left2, right2
    return s[start:end + 1]
s = "babad"
print(longest_palindrome(s))
```

```
bab

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

6. **Zigzag Conversion**

The string "PAYPALISHIRING" is written in a zigzag pattern on a given number of rows like this: (you may want to display this pattern in a fixed font for better legibility) P   A   H   N
A P L S I I G
Y   I   R

And then read line by line: "PAHNAPLSIIGYIR"
Write the code that will take a string and make this conversion given a number of rows:
string convert(string s, int numRows);

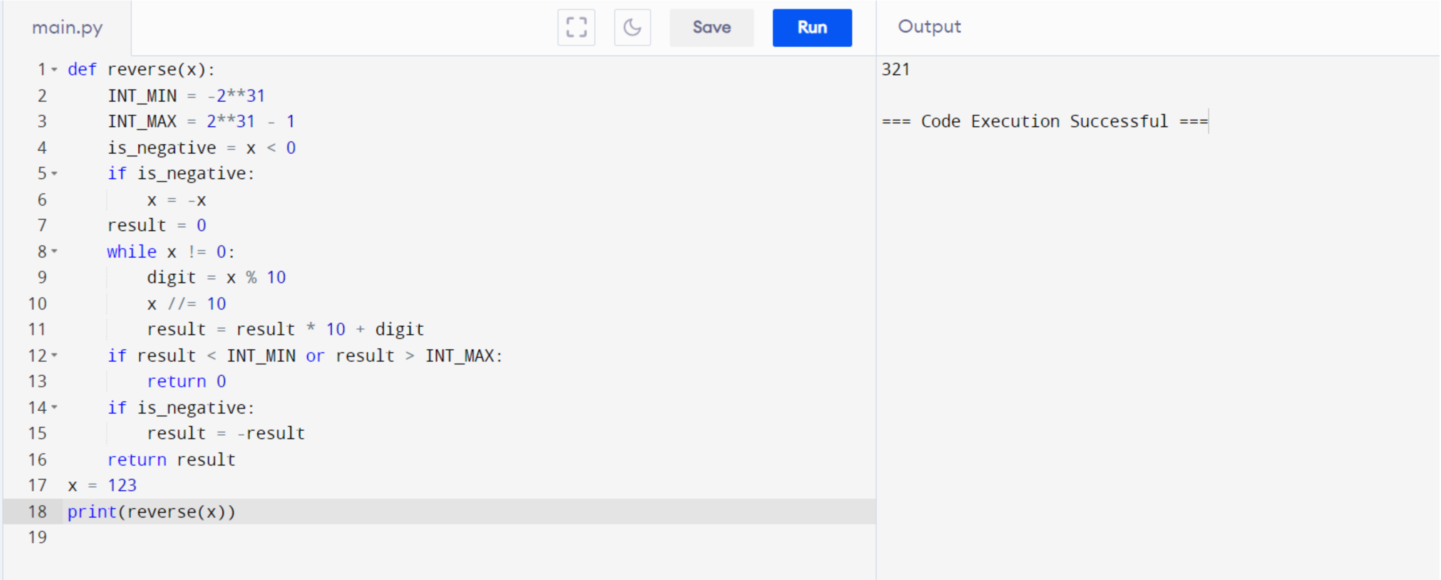


```python
def convert(s, numRows):
    if numRows == 1 or numRows >= len(s):
        return s
    rows = [''] * numRows
    current_row = 0
    going_down = False
    for char in s:
        rows[current_row] += char
        if current_row == 0 or current_row == numRows - 1:
            going_down = not going_down
        current_row += 1 if going_down else -1
    return ''.join(rows)
s = "PAYPALISHIRING"
numRows = 3
print(convert(s, numRows))
```

```
PAHNAPLSIIGYIR

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

7. **Reverse Integer**

Given a signed 32-bit integer x, return x *with its digits reversed*. If reversing x causes the value to go outside the signed 32-bit integer range [-231, 231 - 1], then return 0. Assume the environment does not allow you to store 64-bit integers (signed or unsigned).



```python
def reverse(x):
    INT_MIN = -2**31
    INT_MAX = 2**31 - 1
    is_negative = x < 0
    if is_negative:
        x = -x
    result = 0
    while x != 0:
        digit = x % 10
        x //= 10
        result = result * 10 + digit
    if result < INT_MIN or result > INT_MAX:
        return 0
    if is_negative:
        result = -result
    return result
x = 123
print(reverse(x))

```

```
321

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

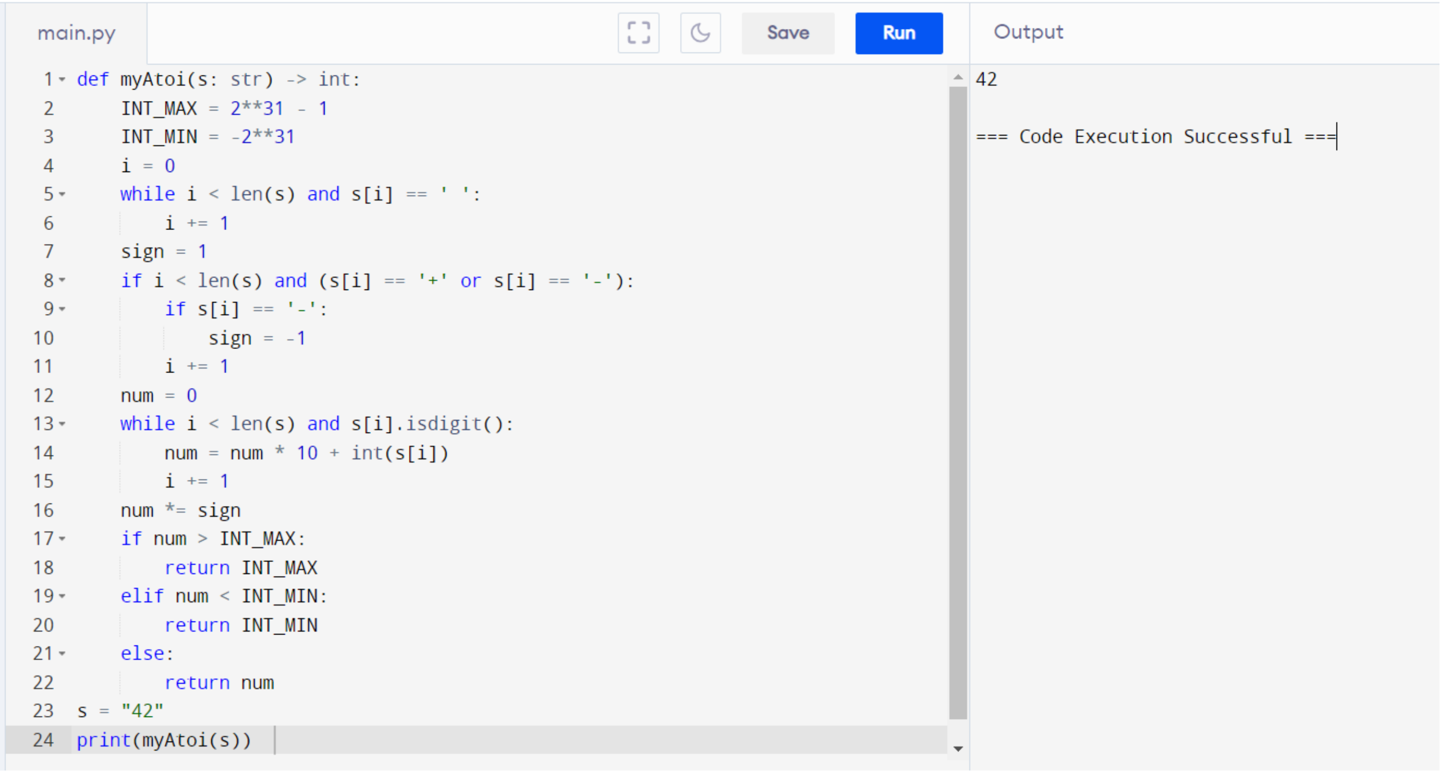
8. String to Integer (atoi)

Implement the myAtoi(string s) function, which converts a string to a 32-bit signed integer (similar to C/C++'s atoi function).

The algorithm for myAtoi(string s) is as follows:

1. Read in and ignore any leading whitespace.
2. Check if the next character (if not already at the end of the string) is '-' or '+'. Read this character in if it is either. This determines if the final result is negative or positive respectively. Assume the result is positive if neither is present.
3. Read in next the characters until the next non-digit character or the end of the input is reached. The rest of the string is ignored.
4. Convert these digits into an integer (i.e. "123" -> 123, "0032" -> 32). If no digits were read, then the integer is 0. Change the sign as necessary (from step 2).
5. If the integer is out of the 32-bit signed integer range [-231, 231 - 1], then clamp the integer so that it remains in the range. Specifically, integers less than -231 should be clamped to -231, and integers greater than 231 - 1 should be clamped to 231 - 1.
6. Return the integer as the final result. Note:

- Only the space character ' ' is considered a whitespace character.
- Do not ignore any characters other than the leading whitespace or the rest of the string after the digits.

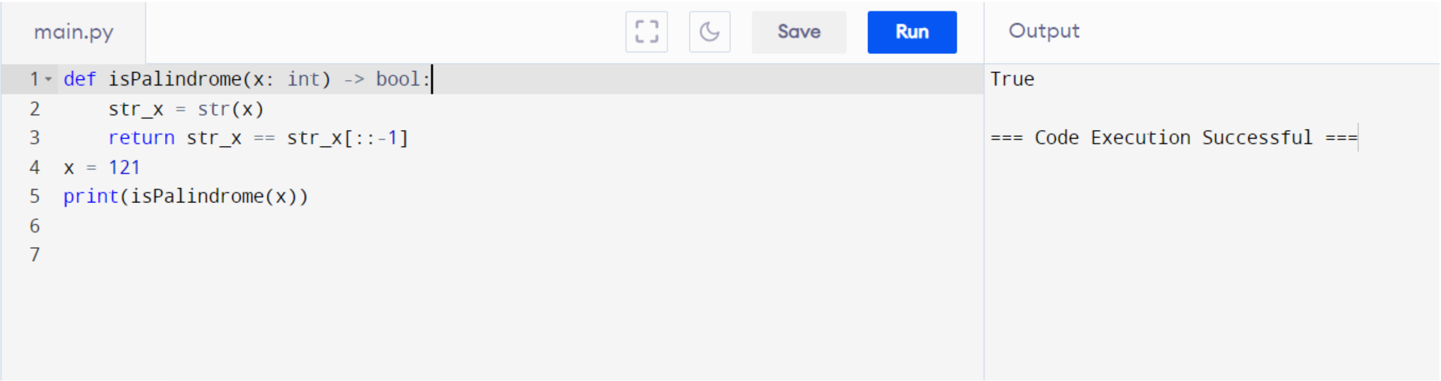


```python
def myAtoi(s: str) -> int:
    INT_MAX = 2**31 - 1
    INT_MIN = -2**31
    i = 0
    while i < len(s) and s[i] == ' ':
        i += 1
    sign = 1
    if i < len(s) and (s[i] == '+' or s[i] == '-'):
        if s[i] == '-':
            sign = -1
        i += 1
    num = 0
    while i < len(s) and s[i].isdigit():
        num = num * 10 + int(s[i])
        i += 1
    num *= sign
    if num > INT_MAX:
        return INT_MAX
    elif num < INT_MIN:
        return INT_MIN
    else:
        return num
s = "42"
print(myAtoi(s))
```

```
42

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

9. Palindrome Number

Given an integer x, return true *if* x *is a palindrome, and* false *otherwise.*



```python
def isPalindrome(x: int) -> bool:
    str_x = str(x)
    return str_x == str_x[::-1]
x = 121
print(isPalindrome(x))
```

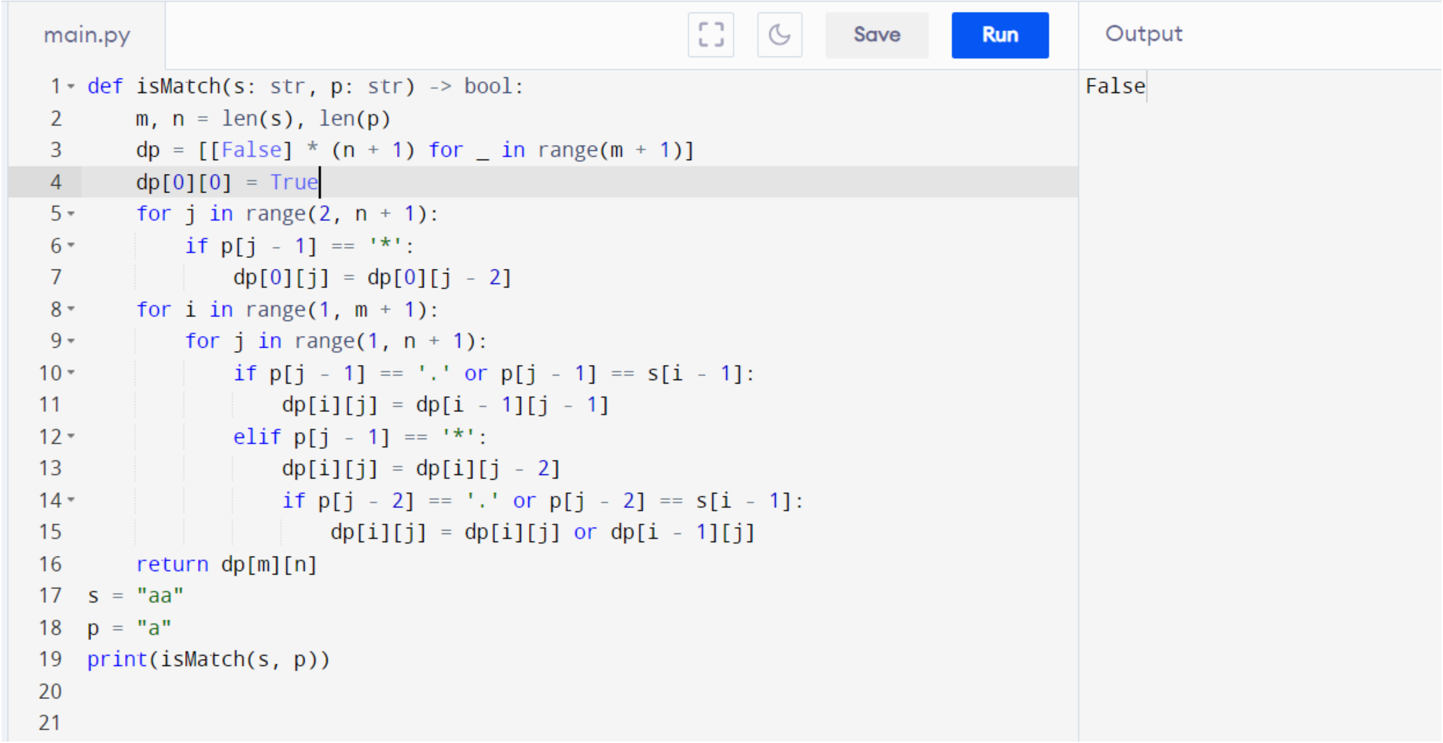
```
True

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

10. Regular Expression Matching

Given an input string s and a pattern p, implement regular expression matching with support for '.' and '*' where:

- '.' Matches any single character.
- '*' Matches zero or more of the preceding element.

The matching should cover the entire input string (not partial).



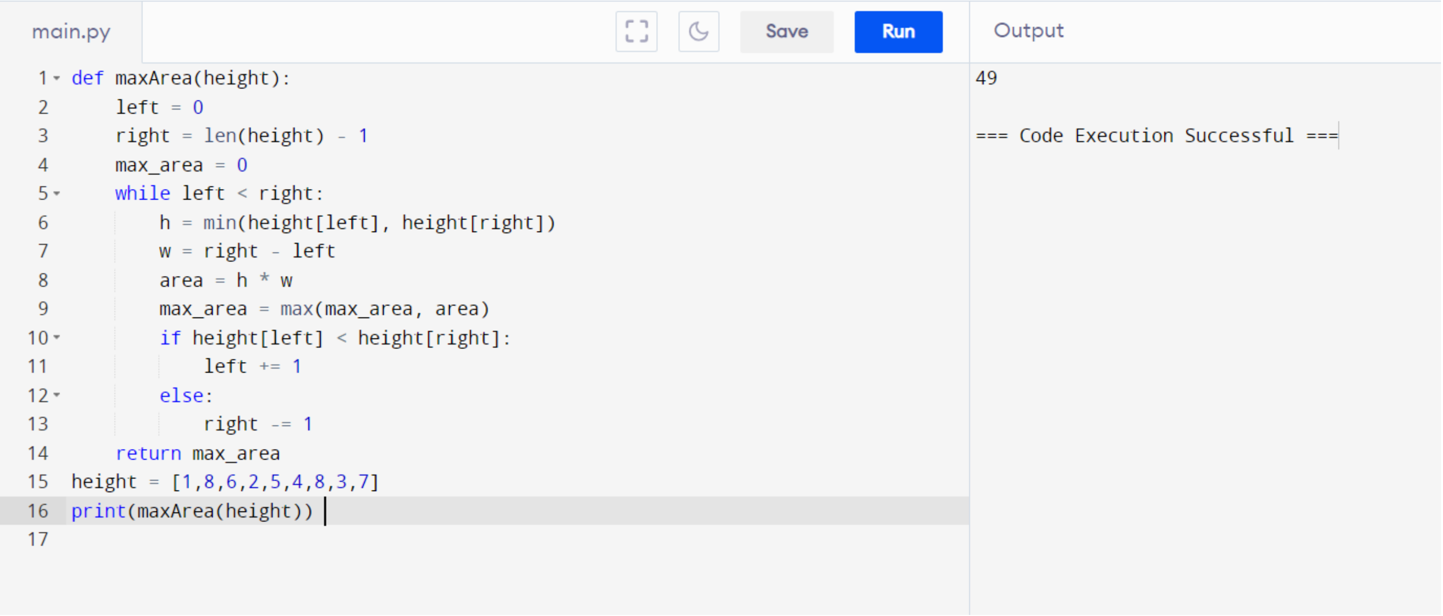
```python
def isMatch(s: str, p: str) -> bool:
    m, n = len(s), len(p)
    dp = [[False] * (n + 1) for _ in range(m + 1)]
    dp[0][0] = True
    for j in range(2, n + 1):
        if p[j - 1] == '*':
            dp[0][j] = dp[0][j - 2]
    for i in range(1, m + 1):
        for j in range(1, n + 1):
            if p[j - 1] == '.' or p[j - 1] == s[i - 1]:
                dp[i][j] = dp[i - 1][j - 1]
            elif p[j - 1] == '*':
                dp[i][j] = dp[i][j - 2]
                if p[j - 2] == '.' or p[j - 2] == s[i - 1]:
                    dp[i][j] = dp[i][j] or dp[i - 1][j]
    return dp[m][n]
s = "aa"
p = "a"
print(isMatch(s, p))
```

```
False
```

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.



```python
def maxArea(height):
    left = 0
    right = len(height) - 1
    max_area = 0
    while left < right:
        h = min(height[left], height[right])
        w = right - left
        area = h * w
        max_area = max(max_area, area)
        if height[left] < height[right]:
            left += 1
        else:
            right -= 1
    return max_area
height = [1,8,6,2,5,4,8,3,7]
print(maxArea(height))
```

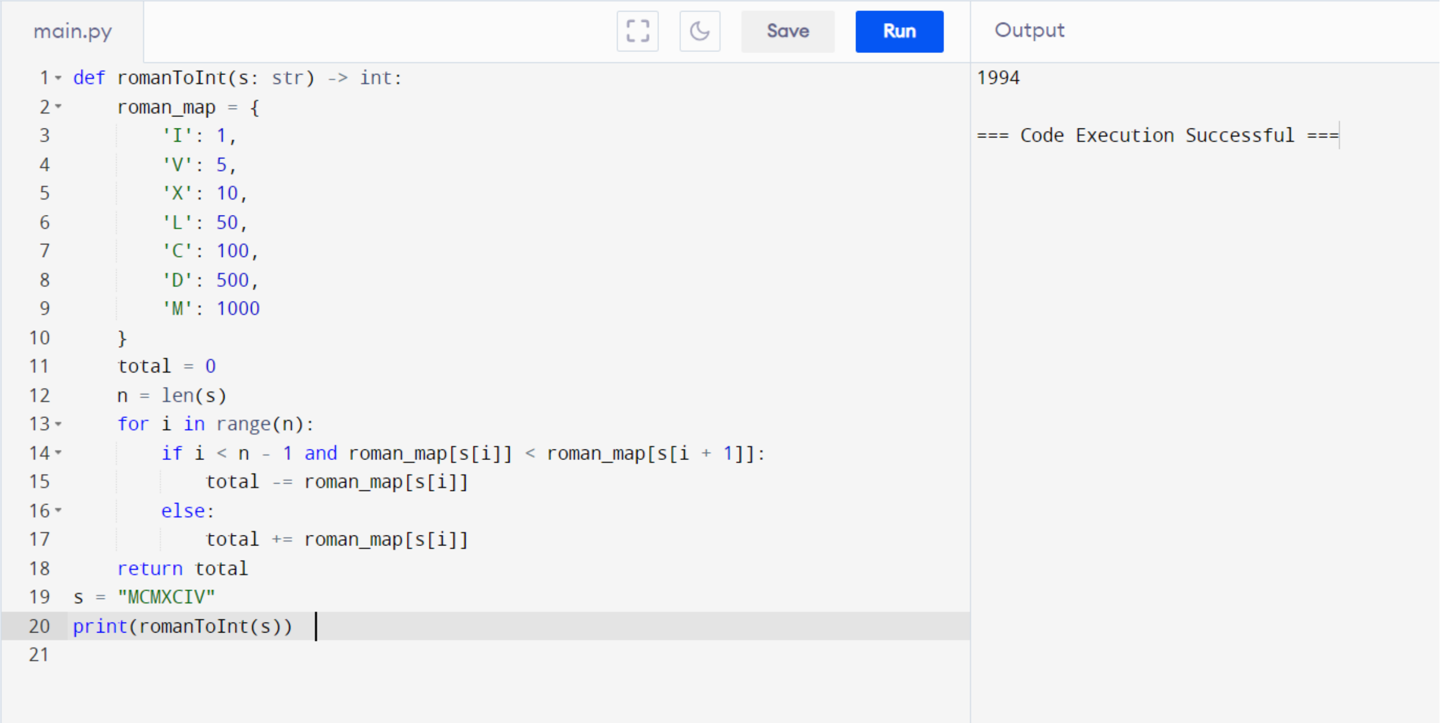
```
49

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

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



```python
def romanToInt(s: str) -> int:
    roman_map = {
        'I': 1,
        'V': 5,
        'X': 10,
        'L': 50,
        'C': 100,
        'D': 500,
        'M': 1000
    }
    total = 0
    n = len(s)
    for i in range(n):
        if i < n - 1 and roman_map[s[i]] < roman_map[s[i + 1]]:
            total -= roman_map[s[i]]
        else:
            total += roman_map[s[i]]
    return total
s = "MCMXCIV"
print(romanToInt(s))
```

```
1994

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

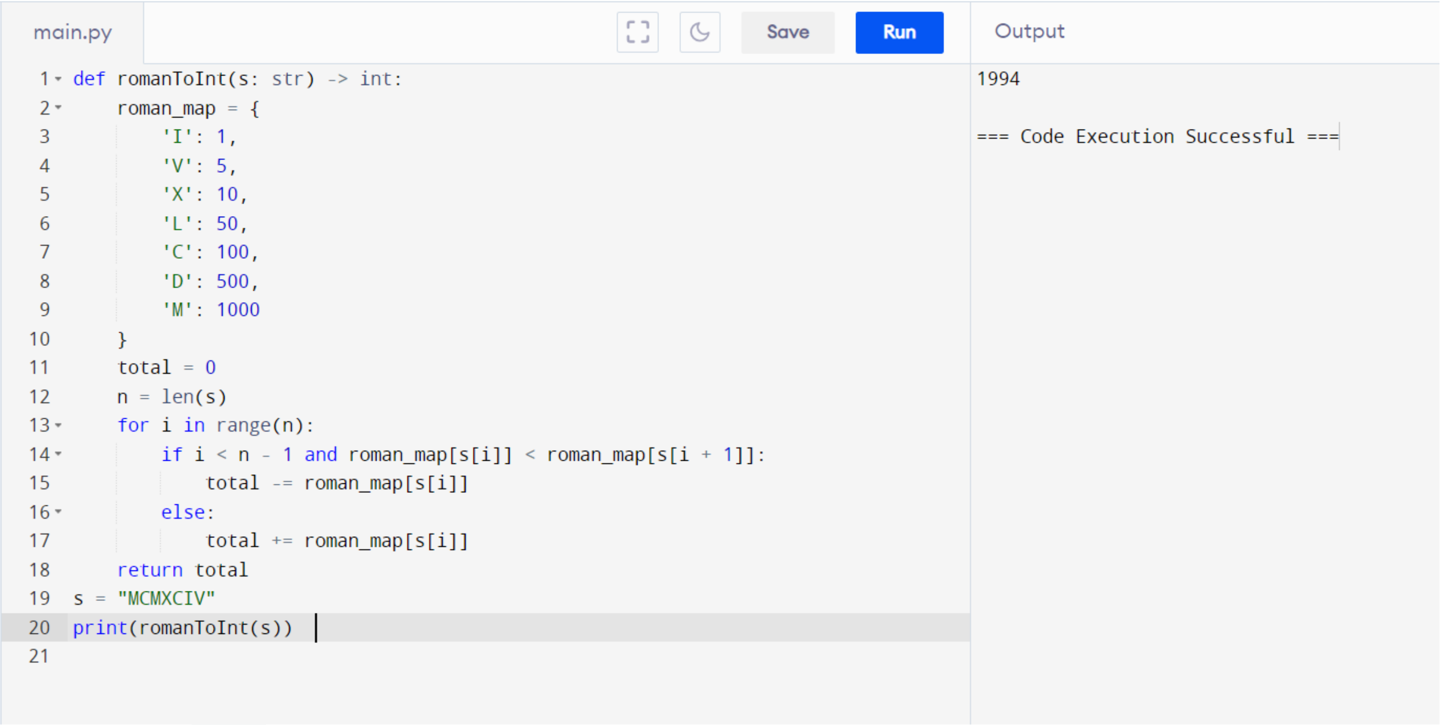
13. Roman to Integer

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

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 . ""



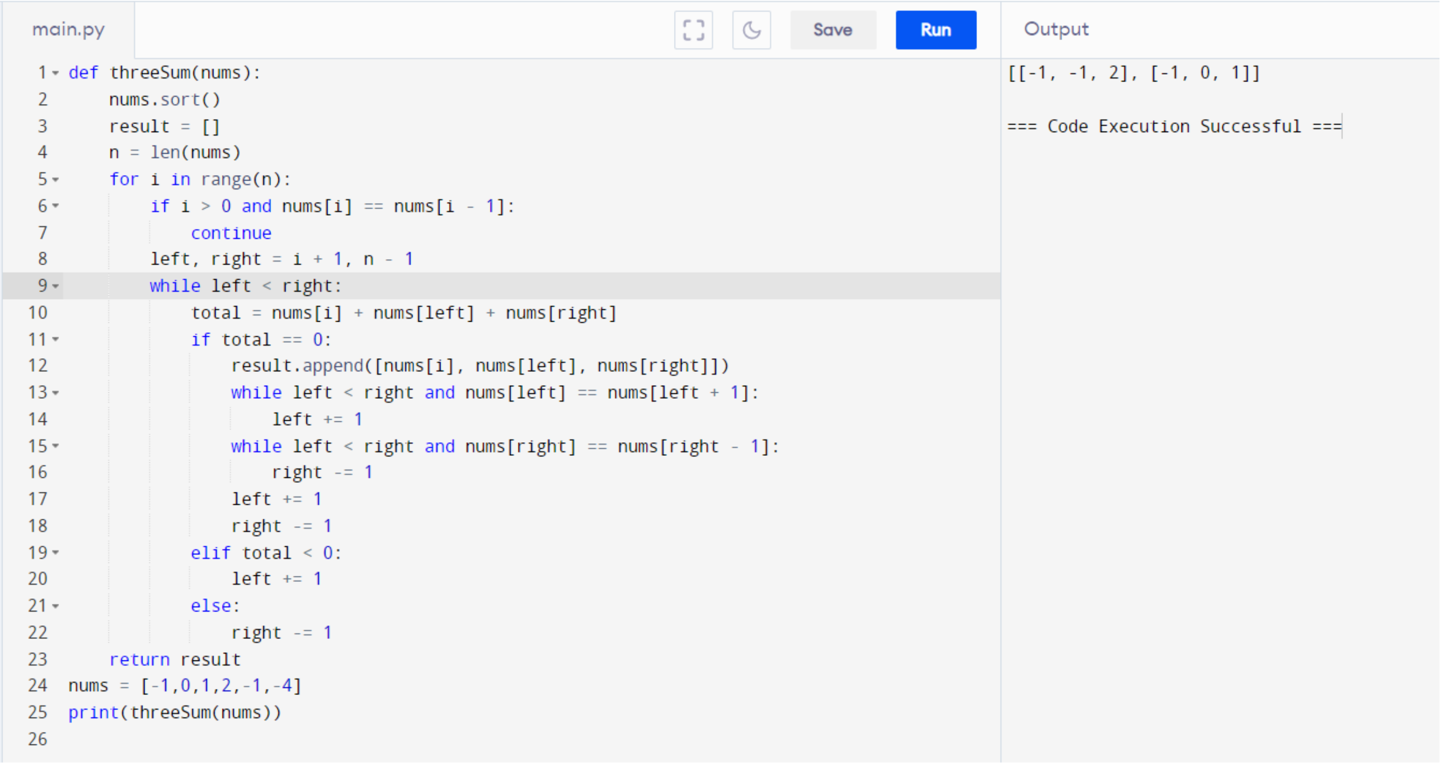
```python
def romanToInt(s: str) -> int:
    roman_map = {
        'I': 1,
        'V': 5,
        'X': 10,
        'L': 50,
        'C': 100,
        'D': 500,
        'M': 1000
    }
    total = 0
    n = len(s)
    for i in range(n):
        if i < n - 1 and roman_map[s[i]] < roman_map[s[i + 1]]:
            total -= roman_map[s[i]]
        else:
            total += roman_map[s[i]]
    return total
s = "MCMXCIV"
print(romanToInt(s))
```

```
1994

=== 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.



```python
def threeSum(nums):
    nums.sort()
    result = []
    n = len(nums)
    for i in range(n):
        if i > 0 and nums[i] == nums[i - 1]:
            continue
        left, right = i + 1, n - 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
nums = [-1,0,1,2,-1,-4]
print(threeSum(nums))
```

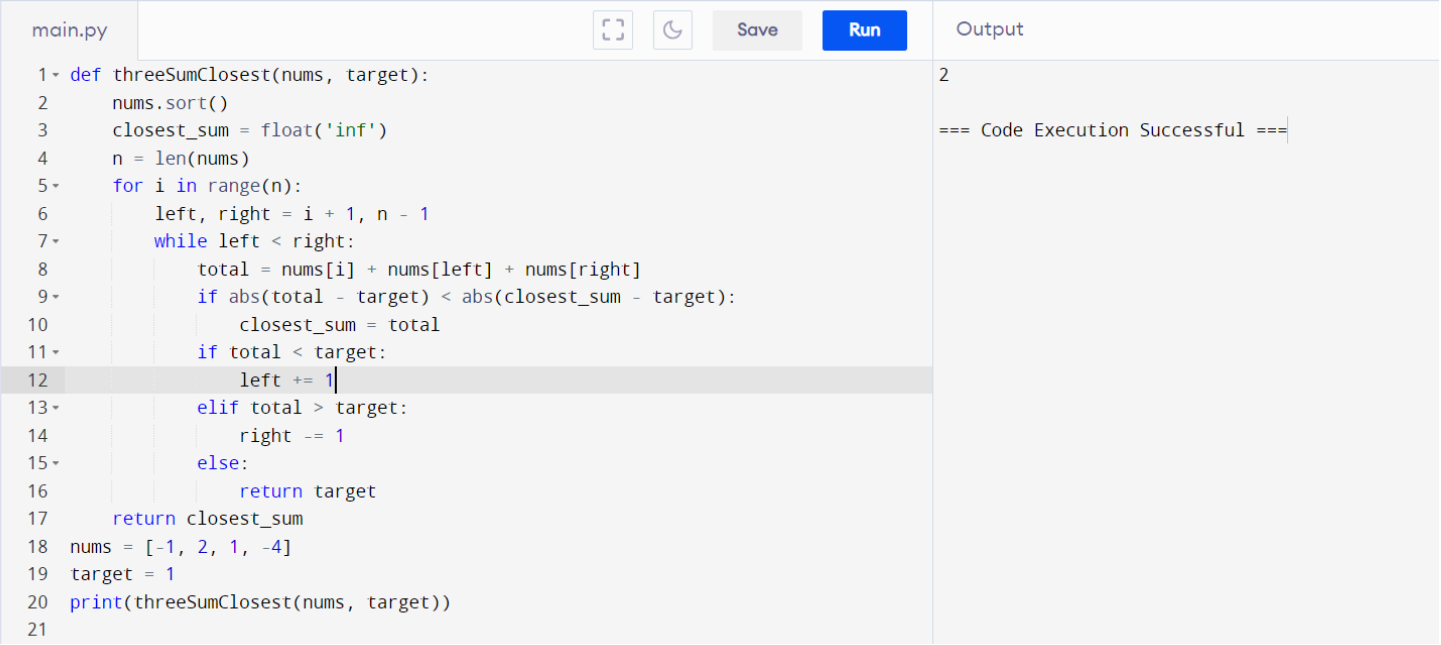
```
[[-1, -1, 2], [-1, 0, 1]]

=== 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.



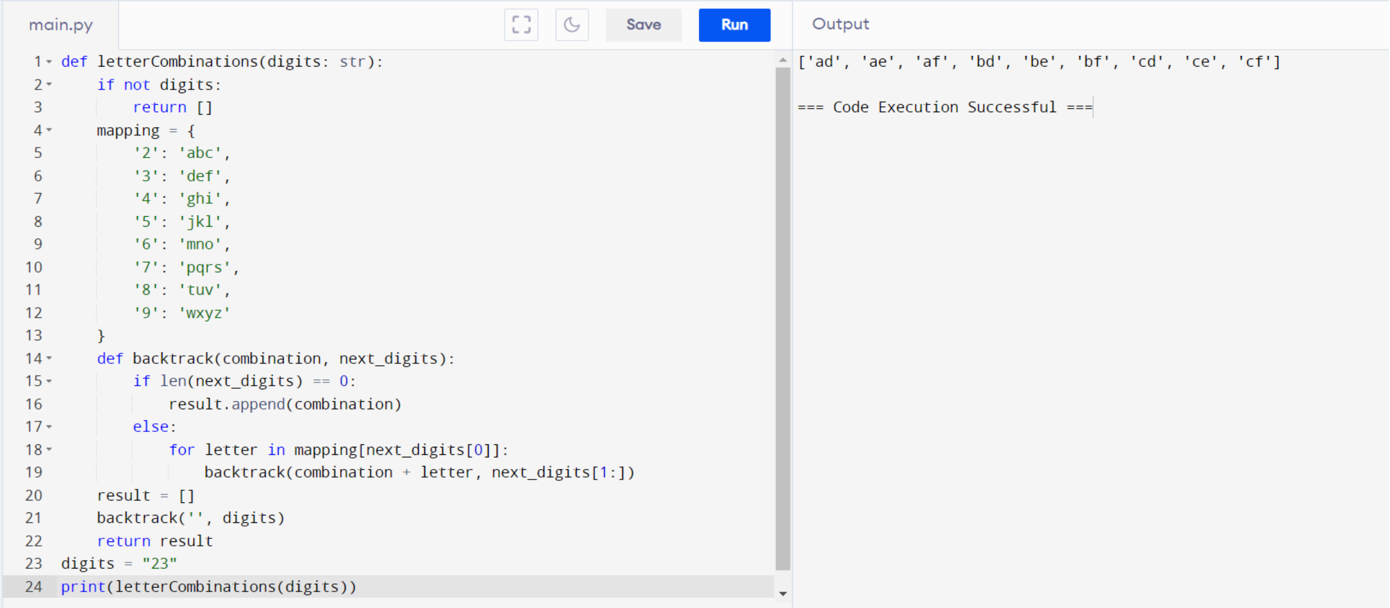
```python
def threeSumClosest(nums, target):
    nums.sort()
    closest_sum = float('inf')
    n = len(nums)
    for i in range(n):
        left, right = i + 1, n - 1
        while left < right:
            total = nums[i] + nums[left] + nums[right]
            if abs(total - target) < abs(closest_sum - target):
                closest_sum = total
            if total < target:
                left += 1
            elif total > target:
                right -= 1
            else:
                return target
    return closest_sum
nums = [-1, 2, 1, -4]
target = 1
print(threeSumClosest(nums, target))
```

```
2

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

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.



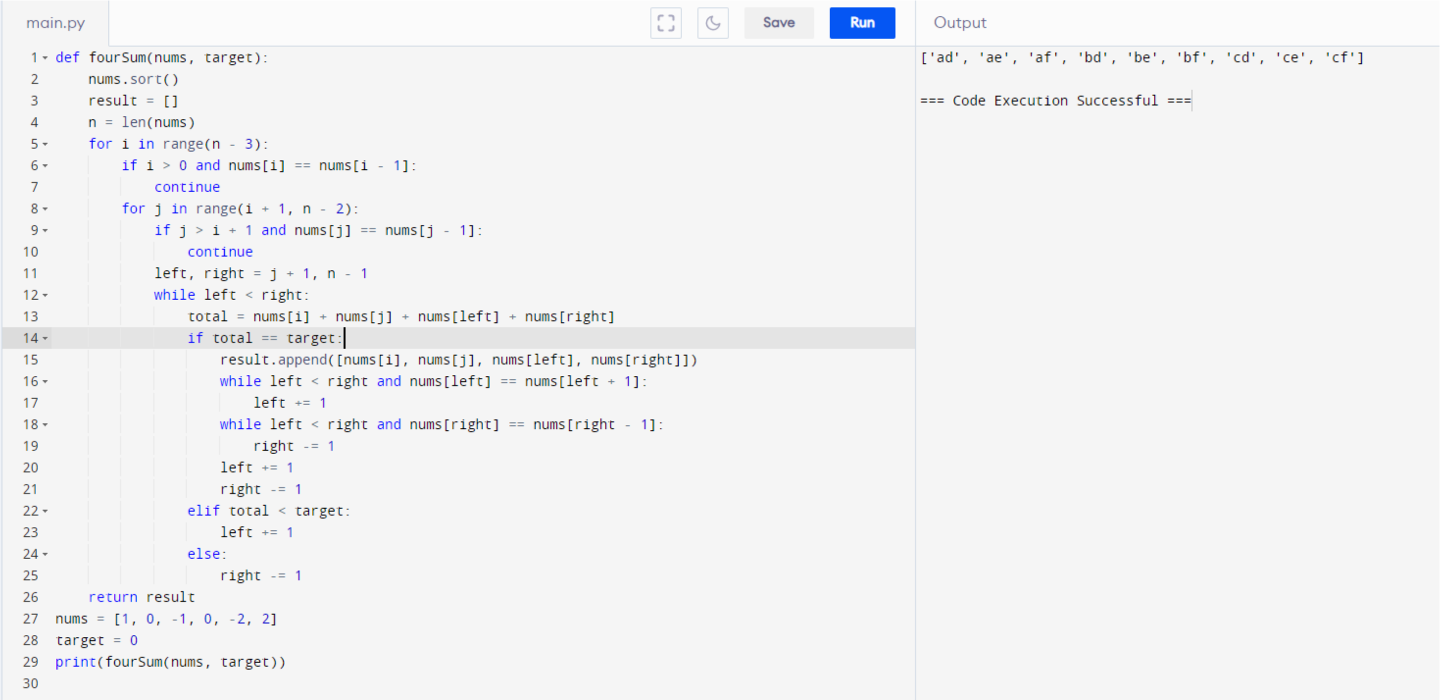
```python
def letterCombinations(digits: str):
    if not digits:
        return []
    mapping = {
        '2': 'abc',
        '3': 'def',
        '4': 'ghi',
        '5': 'jkl',
        '6': 'mno',
        '7': 'pqrs',
        '8': 'tuv',
        '9': 'wxyz'
    }
    def backtrack(combination, next_digits):
        if len(next_digits) == 0:
            result.append(combination)
        else:
            for letter in mapping[next_digits[0]]:
                backtrack(combination + letter, next_digits[1:])
    result = []
    backtrack('', digits)
    return result
digits = "23"
print(letterCombinations(digits))
```

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

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

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.

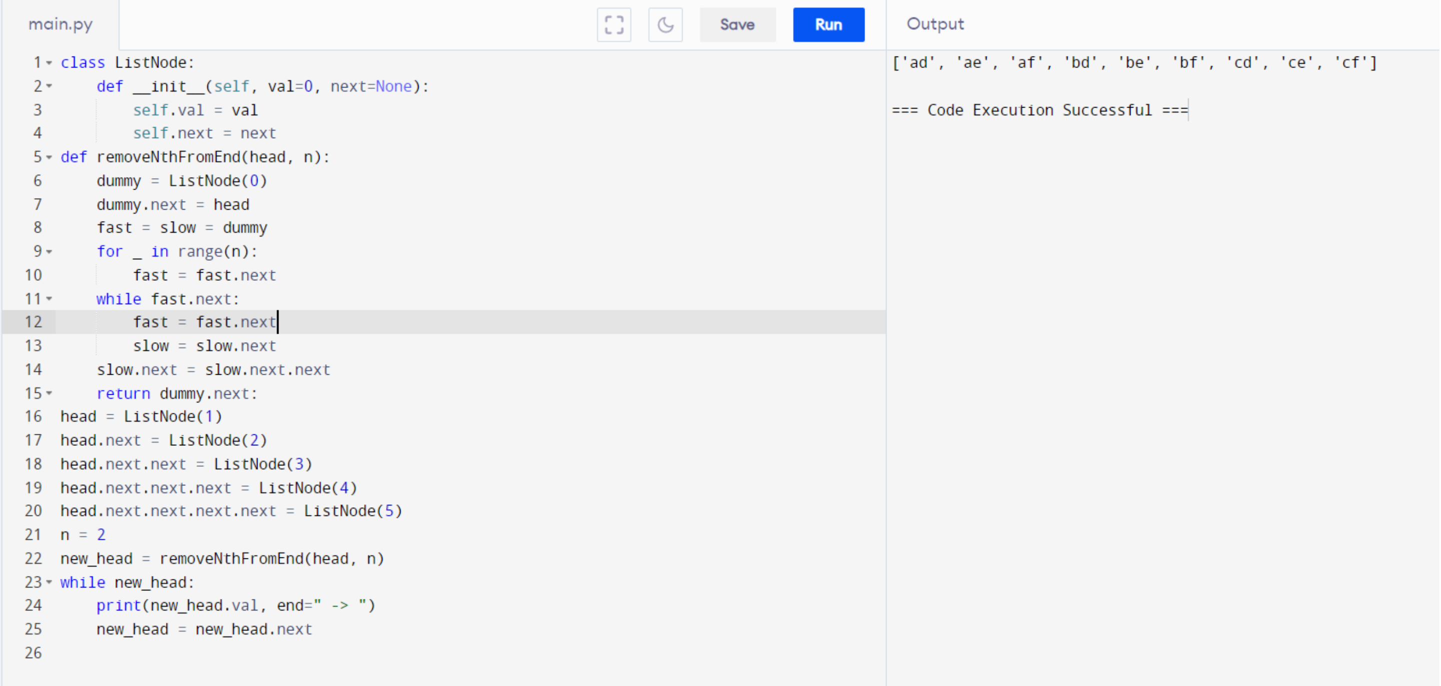


```python
def fourSum(nums, target):
    nums.sort()
    result = []
    n = len(nums)
    for i in range(n - 3):
        if i > 0 and nums[i] == nums[i - 1]:
            continue
        for j in range(i + 1, n - 2):
            if j > i + 1 and nums[j] == nums[j - 1]:
                continue
            left, right = j + 1, n - 1
            while left < right:
                total = nums[i] + nums[j] + nums[left] + nums[right]
                if total == target:
                    result.append([nums[i], nums[j], 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 result
nums = [1, 0, -1, 0, -2, 2]
target = 0
print(fourSum(nums, target))
```

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

=== 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.



```python
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next
def removeNthFromEnd(head, n):
    dummy = ListNode(0)
    dummy.next = head
    fast = slow = dummy
    for _ in range(n):
        fast = fast.next
    while fast.next:
        fast = fast.next
        slow = slow.next
    slow.next = slow.next.next
    return dummy.next:
head = ListNode(1)
head.next = ListNode(2)
head.next.next = ListNode(3)
head.next.next.next = ListNode(4)
head.next.next.next.next = ListNode(5)
n = 2
new_head = removeNthFromEnd(head, n)
while new_head:
    print(new_head.val, end=" -> ")
    new_head = new_head.next
```

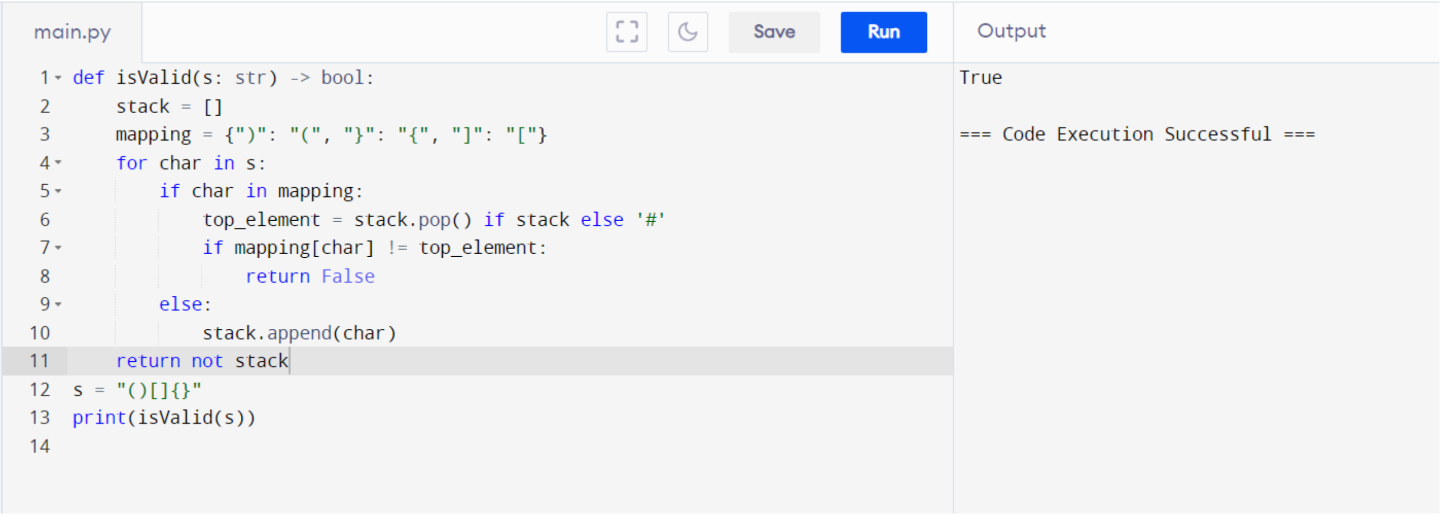
```
['ad', 'ae', 'af', 'bd', 'be', 'bf', 'cd', 'ce', 'cf']

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

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.



```python
def isValid(s: str) -> bool:
    stack = []
    mapping = {")": "(", "}": "{", "]": "["}
    for char in s:
        if char in mapping:
            top_element = stack.pop() if stack else '#'
            if mapping[char] != top_element:
                return False
        else:
            stack.append(char)
    return not stack
s = "()[]{}"
print(isValid(s))
```

```
True

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