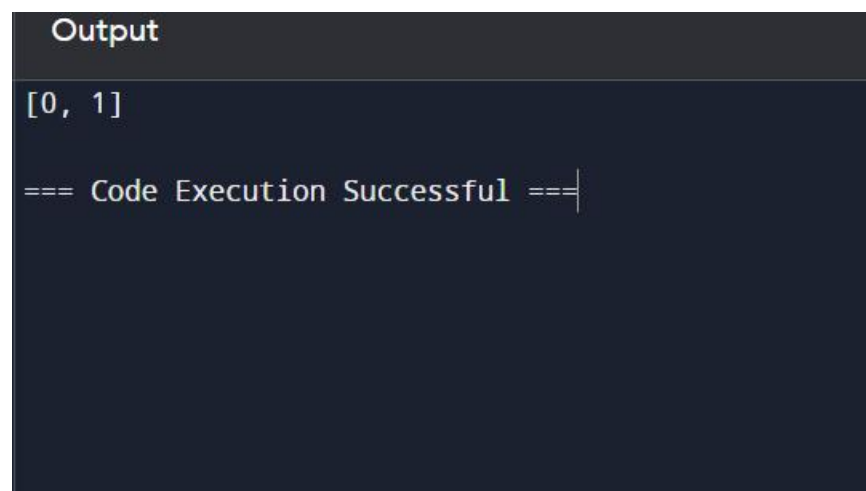


**QUESTION 1:**

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.

**CODE:**

```
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))
```

**RESULT:**

the program is executed successfully.

**QUESTION 2:**

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.

**CODE:**

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

    def addTwoNumbers(l1, l2):
        dummy = ListNode()
        current, carry = dummy, 0
        while l1 or l2 or carry:
            x = l1.val if l1 else 0
            y = l2.val if l2 else 0
            carry, out = divmod(x + y + carry, 10)
            current.next = ListNode(out)
            current = current.next
            l1 = l1.next if l1 else None
            l2 = l2.next if l2 else None
        return dummy.next

    def print_linked_list(node):
        while node:
            print(node.val, end=" -> " if node.next else "\n")
            node = node.next

l1 = ListNode(2, ListNode(4, ListNode(3)))
l2 = ListNode(5, ListNode(6, ListNode(4)))
result = addTwoNumbers(l1, l2)
print_linked_list(result)
```

### Output

7 -> 0 -> 8

=== Code Execution Successful ===

### RESULT:

the program is executed successfully.

### QUESTION 3:

Longest Substring without Repeating Characters Given a string s, find the length of the longest substring without repeating characters.

### CODE:

```
def lengthOfLongestSubstring(s: str) -> int:
    char_set = set()
    left = 0
    max_length = 0
    for right in range(len(s)):
        while s[right] in char_set:
            char_set.remove(s[left])
            left += 1
        char_set.add(s[right])
        max_length = max(max_length, right - left + 1)
    return max_length

s = "abcabcbb"
print(lengthOfLongestSubstring(s))
```

```
Output
3
=== Code Execution Successful ===
```

**RESULT:**

the program is executed successfully.

**QUESTION 4:**

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  $O(\log(m+n))$ .

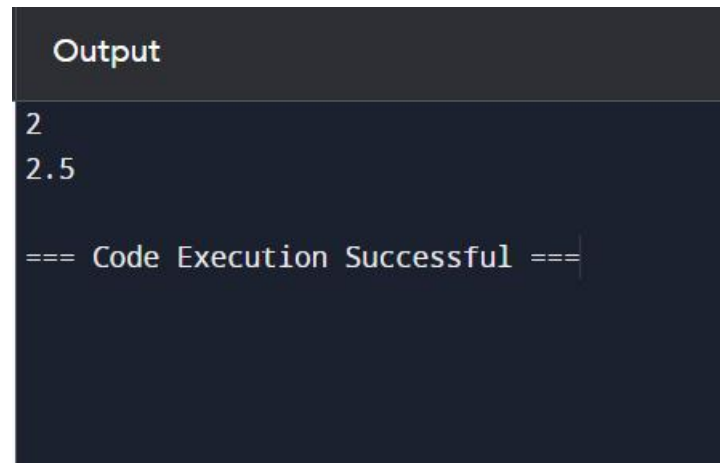
**CODE:**

```
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:
            max_of_left = max(nums1[i - 1] if i > 0 else float('-inf'),
                               nums2[j - 1] if j > 0 else float('-inf'))
```

```

if (m + n) % 2 == 1:
    return max_of_left
min_of_right = min(nums1[i] if i < m else float('inf'),
nums2[j] if j < n else float('inf'))
return (max_of_left + min_of_right) / 2.0
nums1 = [1, 3]
nums2 = [2]
print(findMedianSortedArrays(nums1, nums2)) # Output: 2.0
nums1 = [1, 2]
nums2 = [3, 4]
print(findMedianSortedArrays(nums1, nums2))

```



```

Output
2
2.5

=== Code Execution Successful ===

```

# **RESULT:**

the program is executed successfully.

# **QUESTION 5:**

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

# **CODE:**

```

def longestPalindrome(s: str) -> str:
    if not s:
        return ""

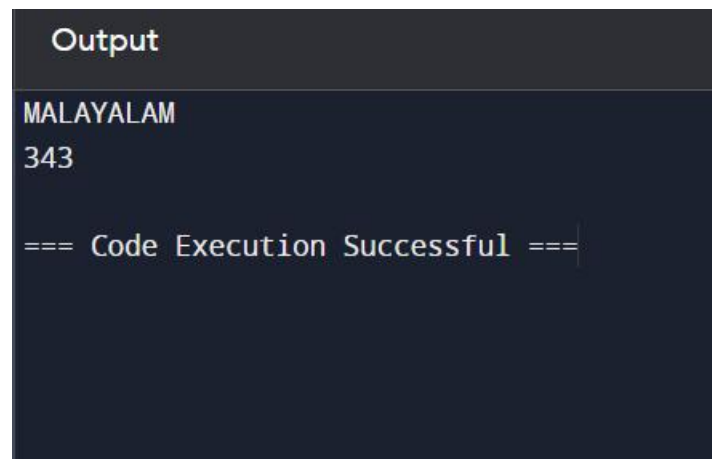
```

```
def expandAroundCenter(left: int, right: int) -> str:
    while left >= 0 and right < len(s) and s[left] == s[right]:
        left -= 1
        right += 1
    return s[left + 1:right]

longest = ""
for i in range(len(s)):
    longest = max(longest,
        expandAroundCenter(i, i),
        expandAroundCenter(i, i + 1),
        key=len)
return longest

s1 = "MALAYALAM"
print(longestPalindrome(s1))

s2 = "12343"
print(longestPalindrome(s2))
```



The screenshot shows a dark-themed output window with the following text:

```
Output
MALAYALAM
343

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

**RESULT:**

the program is executed successfully.

### QUESTION 6:

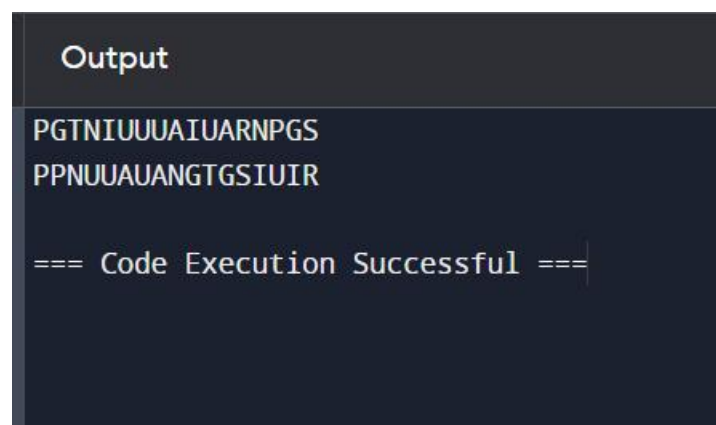
The string "PAYPALISHIRING" is written in a zigzag pattern on a given number of rows like this: (you may want to display this pattern in a fixed font for better legibility)

### CODE:

```
def convert(s: str, numRows: int) -> str:
    if numRows == 1 or numRows >= len(s):
        return s
    rows = [''] * numRows
    current_row, going_down = 0, 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 = "PUNUGUPATIGUNASRI"
numRows = 3
print(convert(s, numRows))

numRows = 4
print(convert(s, numRows))
```



The screenshot shows a dark-themed output window. At the top, the word "Output" is displayed in a light blue font. Below it, the results of the code execution are shown in a light green monospace font. The first result is for 3 rows, showing two lines of text: "PGTNIUUUAIUARNPGS" and "PPNUJUAUANGTGSUIR". The second result is for 4 rows, showing a single line of text: "=== Code Execution Successful ===".

### RESULT:

the program is executed successfully.

**QUESTION 7:**

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  $[-2^{31}, 2^{31} - 1]$ , then return 0. Assume the environment does not allow you to store 64-bit integers (signed or unsigned).

**CODE:**

```
def reverse(x: int) -> int:

    INT_MAX = 2**31 - 1

    INT_MIN = -2**31

    sign = 1 if x >= 0 else -1

    x = abs(x)

    result = 0

    while x:

        result = result * 10 + x % 10

        x //= 10

    if result > INT_MAX:

        return 0

    return sign * result

x = 123

print(reverse(x))

x = -123

print(reverse(x))

x = 120

print(reverse(x))

x = 1534236469

print(reverse(x))
```



```
Output
321
-321
21
0

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

**RESULT:**

the program is executed successfully.

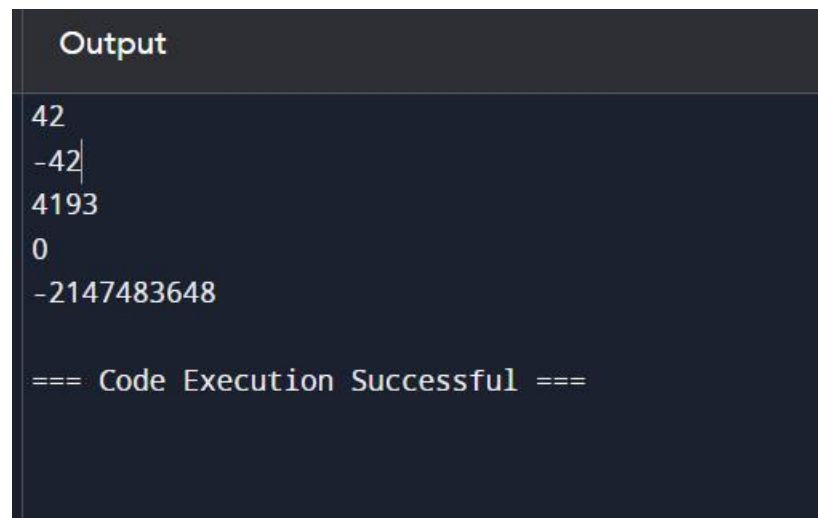
**QUESTION 8:**

Implement the myAtoi(string s) function, which converts a string to a 32-bit signed integer.

**CODE:**

```
def myAtoi(s: str) -> int:
    INT_MAX = 2**31 - 1
    INT_MIN = -2**31
    s = s.strip()
    if not s:
        return 0
    sign = 1
    if s[0] in ('+', '-'):
        sign = -1 if s[0] == '-' else 1
    s = s[1:]
    result = 0
    for char in s:
        if not char.isdigit():
            break
```

```
result = result * 10 + int(char)
if result > INT_MAX:
return INT_MAX if sign == 1 else INT_MIN
return sign * result
s1 = "42"
print(myAtoi(s1))
s2 = " -42"
print(myAtoi(s2))
s3 = "4193 with words"
print(myAtoi(s3))
s4 = "words and 987"
print(myAtoi(s4))
s5 = "-91283472332"
print(myAtoi(s5))
```



The screenshot shows a dark-themed window titled "Output". It displays the results of the myAtoi function for five different inputs. The outputs are: 42, -42, 4193, 0, and -2147483648. At the bottom, it says "=== Code Execution Successful ===".

```
Output
42
-42
4193
0
-2147483648
=== Code Execution Successful ===
```

**RESULT:**

the program is executed successfully.

**QUESTION 9:**

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

**CODE:**

```
def isPalindrome(x: int) -> bool:
```

```
    str_x = str(x)
```

```
    return str_x == str_x[::-1]
```

```
x1 = 121
```

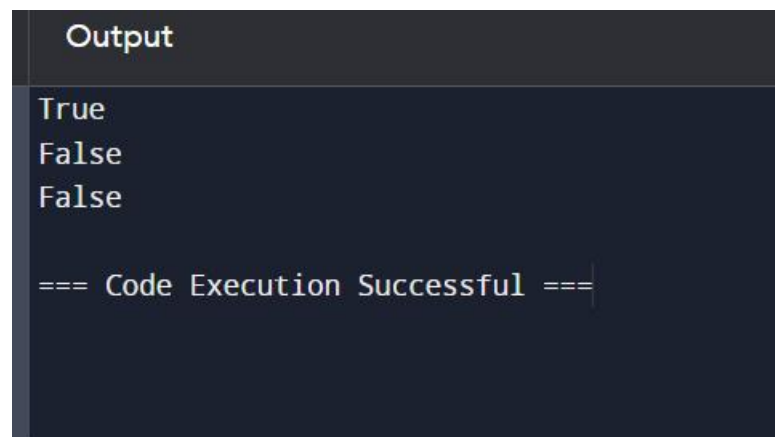
```
print(isPalindrome(x1))
```

```
x2 = -121
```

```
print(isPalindrome(x2))
```

```
x3 = 10
```

```
print(isPalindrome(x3))
```



The screenshot shows a dark-themed output window with a title bar labeled "Output". The output text is as follows:

```
True
False
False

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

**RESULT:**

the program is executed successfully.

**QUESTION 10:**

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).

**CODE:**

```
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(1, 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] or (dp[i - 1][j] and (p[j - 2] == s[i - 1] or p[j - 2] == '.'))
    return dp[m][n]

s1 = "aa"
p1 = "a"
print(isMatch(s1, p1))

s2 = "aa"
p2 = "a*"
print(isMatch(s2, p2))

s3 = "ab"
p3 = ".*"
print(isMatch(s3, p3))

s4 = "aab"
p4 = "c*a*b"
print(isMatch(s4, p4))
```

```
Output
False
True
True
True

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

**RESULT:**

the program is executed successfully.