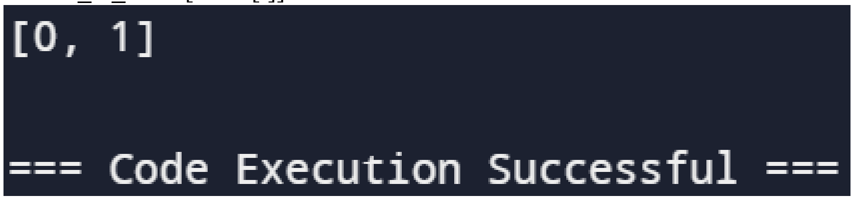
1. **Two Sum Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to target. You may assume that each input would have exactly one solution, and you may not use the same element twice. You can return the answer in any order.**
   Example 1: Input: nums = [2,7,11,15], target = 9 Output: [0,1]

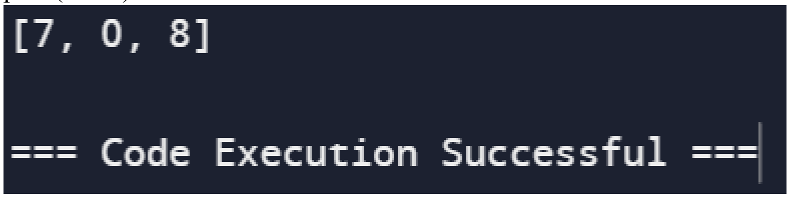
```
nums = [2, 7, 11, 15]
target = 9
num_to_index = {}
for i in range(len(nums)):
    complement = target - nums[i]
    if complement in num_to_index:
        print([num_to_index[complement], i])
        break
    num_to_index[nums[i]] = i
```



```
[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.**
   Example 1: Input: l1 = [2,4,3], l2 = [5,6,4] Output: [7,0,8]

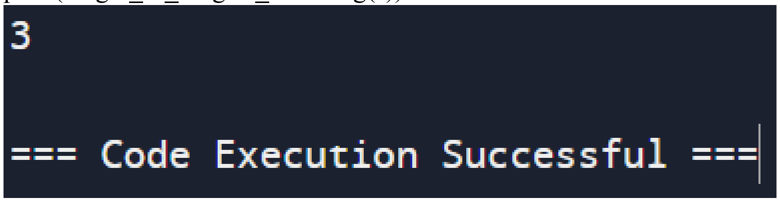
```
def add(a, b):
    a.reverse()
    b.reverse()
    anum = int(''.join(map(str, a)))
    bnum = int(''.join(map(str, b)))
    d = anum + bnum
    if d == 0:
        return [0]
    c = []
    while d > 0:
        r = d % 10
        c.append(r)
        d = d // 10
    return c
a = [2, 4, 3]
b = [5, 6, 4]
result = add(a, b)
print(result)
```



```
[7, 0, 8]

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

3. **Longest Substring without Repeating Characters Given a string s, find the length of the longest substring without repeating characters.**
   Example 1: Input: s = "abcabcbb" Output: 3

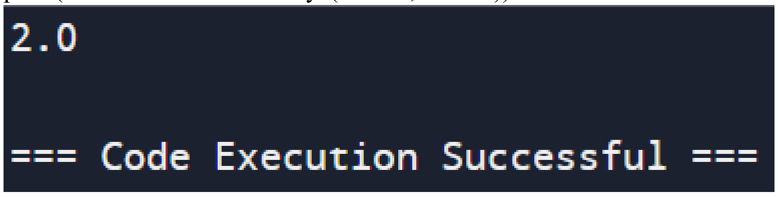
```
def length_of_longest_substring(s):
    start = maxLength = 0
    usedChars = {}
    for i in range(len(s)):
        if s[i] in usedChars and start <= usedChars[s[i]]:
            start = usedChars[s[i]] + 1
        else:
            maxLength = max(maxLength, i - start + 1)
        usedChars[s[i]] = i
    return maxLength
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)).**
   Example 1:   Input: nums1 = [1,3], nums2 = [2]   Output: 2.00000

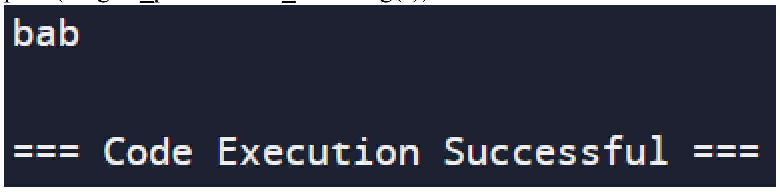
```
def findMedianSortedArrays(nums1, nums2):
    nums = sorted(nums1 + nums2)
    n = len(nums)
    if n % 2 == 0:
        return (nums[n // 2 - 1] + nums[n // 2]) / 2
    else:
        return nums[n // 2]
nums1 = [1, 3]
nums2 = [2]
print(findMedianSortedArrays(nums1, nums2))
```



```
2.0

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

5. **Longest Palindromic Substring Given a string s, return the longest palindromic substring in s.**
   Example 1:   Input: s = "babad"   Output: "bab"

```
def longest_palindromic_substring(s):
    def is_palindrome(s):
        return s == s[::-1]
    longest_palindrome = ""
    for i in range(len(s)):
        for j in range(i, len(s)):
            substring = s[i:j+1]
            if is_palindrome(substring) and len(substring) > len(longest_palindrome):
                longest_palindrome = substring
    return longest_palindrome
s = "babad"
print(longest_palindromic_substring(s))
```



```
bab

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

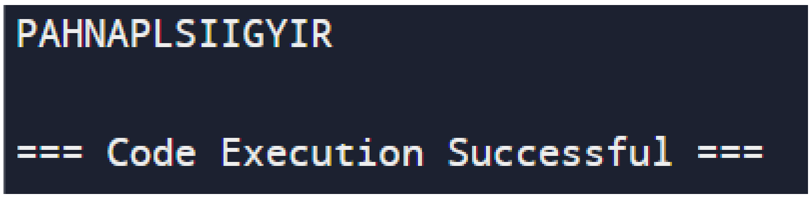
6. **Zigzag Conversion The string "PAYPALISHIRING" is written in a zigzag pattern on a given number of rows like this: (you may want to display this pattern in a fixed font for better legibility) P A H N A P L S I I G Y I R And then read line by line: "PAHNAPLSIIGYIR" Write**

the code that will take a string and make this conversion given a number of rows: string convert(string s, int numRows);

Example 1: Input: s = "PAYPALISHIRING", numRows = 3

Output: "PAHNAPLSIIGYIR"

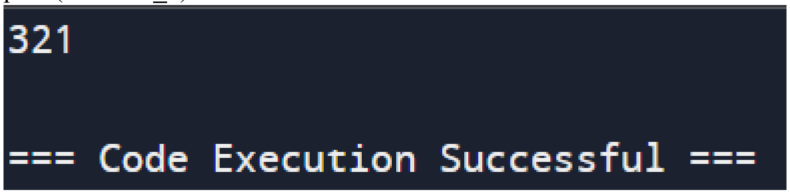
```
def convert(s, numRows):
    if numRows == 1 or numRows >= len(s):
        return s
    rows = [''] * numRows
    index, step = 0, 1
    for char in s:
        rows[index] += char
        if index == 0:
            step = 1
        elif index == numRows - 1:
            step = -1
        index += step
    return ''.join(rows)
s = "PAYPALISHIRING"
numRows = 3
output = convert(s, numRows)
print(output)  # Output: "PAHNAPLSIIGYIR"
```



```
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).**
   Example 1: Input: x = 123 Output: 321

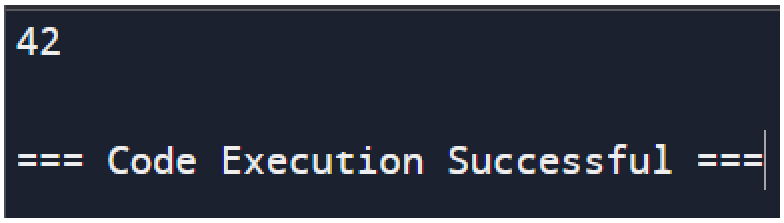
```
def reverse_integer(x):
    INT_MIN = -2**31  # Minimum value of a 32-bit signed integer
    INT_MAX = 2**31 - 1  # Maximum value of a 32-bit signed integer
    reversed_num = 0
    is_negative = x < 0
    x = abs(x)
    while x > 0:
        digit = x % 10
        if reversed_num > (INT_MAX - digit) // 10:
            return 0  # Overflow occurred, return 0
        reversed_num = (reversed_num * 10) + digit
        x //= 10
    if is_negative:
        reversed_num = -reversed_num
    return reversed_num
x = 123
reversed_x = reverse_integer(x)
print(reversed_x)
```

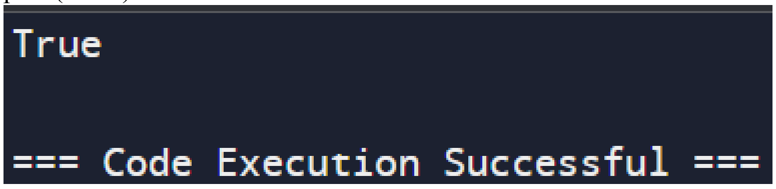


```
321

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

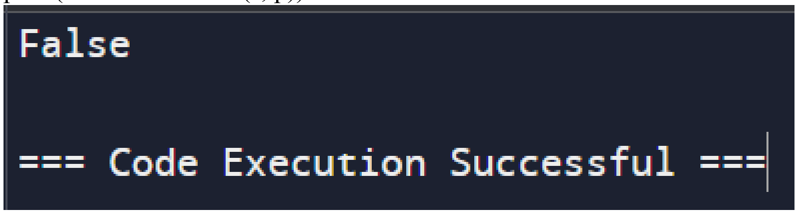
8. **String to Integer (atoi) Implement the myAtoi(string s) function, which converts a string to a 32-bit signed integer (similar to C/C++'s atoi function). The algorithm for myAtoi(string s) is as follows:**
1. Read in and ignore any leading whitespace.
2. Check if the next character (if not already 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.
Example 1: Input: s = "42" Output: 42

```
def myAtoi(s):
    INT_MIN = -2**31  # Minimum value of a 32-bit signed integer
    INT_MAX = 2**31 - 1  # Maximum value of a 32-bit signed integer
    i = 0
    while i < len(s) and s[i] == ' ':
        i += 1
    sign = 1
    if i < len(s) and (s[i] == '+' or s[i] == '-'):
        sign = -1 if s[i] == '-' else 1
        i += 1
    num = 0
    while i < len(s) and s[i].isdigit():
        digit = int(s[i])
        if num > (INT_MAX - digit) // 10:
            return INT_MAX if sign == 1 else INT_MIN
        num = num * 10 + digit
        i += 1
    return min(max(sign * num, INT_MIN), INT_MAX)
s = "42"
result = myAtoi(s)
print(result)
```


```
42

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

9. **Palindrome Number Given an integer x, return true if x is a palindrome, and false otherwise.**
Example 1:   **Input: x = 121**
```
def is_palindrome(x):
    if x < 0:
        return False
    reversed_num = 0
    original_num = x
    while x > 0:
        digit = x % 10
        reversed_num = (reversed_num * 10) + digit
        x //= 10
    return reversed_num == original_num
x = 121
result = is_palindrome(x)
print(result)
```

```
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).**

Example 1:   **Input: s = "aa", p = "a"**   **Output: false**
```
class Solution:
    def isMatch(self, s: str, p: str) -> bool:
        if not p:
            return not s
        first_match = bool(s) and p[0] in {s[0], '.'}
        if len(p) >= 2 and p[1] == '*':
            return (self.isMatch(s, p[2:]) or
                    first_match and self.isMatch(s[1:], p))
        else:
            return first_match and self.isMatch(s[1:], p[1:])
solution = Solution()
s = "aa"
p = "a"
print(solution.isMatch(s, p))
```

```
False

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