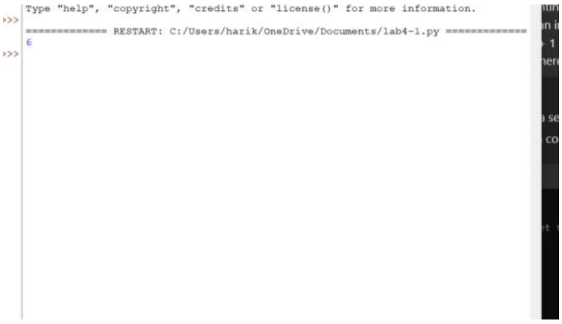
1. Counting Elements
Given an integer array arr, count how many elements x there are, such that x + 1 is also in arr. If there are duplicates in arr, count them separately.

Code:

```
def count_elements(arr):

    element_set = set(arr)

        count = 0    for x in arr:

        if x + 1 in element_set:

            count += 1


    return count

arr = [1, 2, 3, 4, 5, 5, 6, 6]

print(count_elements(arr))
```

output:



```
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: C:/Users/harik/OneDrive/Documents/lab4-1.py =============
6
>>>
```

2. Perform String Shifts
You are given a string s containing lowercase English letters, and a matrix shift, where shift[i] = [directioni, amounti]:

- directioni can be 0 (for left shift) or 1 (for right shift).
- amounti is the amount by which string s is to be shifted.
- A left shift by 1 means remove the first character of s and append it to the end.
- Similarly, a right shift by 1 means remove the last character of s and add it to the beginning.

Return the final string after all operations.

CODE:

```
def perform_string_shifts(s, shift):

    net_shift = 0


    for direction, amount in shift:
        if direction == 0:
            net_shift -= amount
        else:
            net_shift += amount

    # Reduce the net shift to within the bounds of the string length
    net_shift %= len(s)

    # Perform the final shift
    if net_shift > 0:
        # Right shift
        s = s[-net_shift:] + s[:-net_shift]
    elif net_shift < 0:
        # Left shift
        s = s[-net_shift:] + s[:-net_shift]

    return s

s = "abcdefg"
shift = [[1, 1], [1, 1], [0, 2], [1, 3]]
print(perform_string_shifts(s, shift))
```

```
efgabcd
>>> |
```

3. Leftmost Column with at Least a One

A row-sorted binary matrix means that all elements are 0 or 1 and each row of the matrix is sorted in non-decreasing order.

Given a row-sorted binary matrix binaryMatrix, return the index (0-indexed) of the leftmost column with a 1 in it. If such an index does not exist, return -1.

You can't access the Binary Matrix directly. You may only access the matrix using a BinaryMatrix interface:

- BinaryMatrix.get(row, col) returns the element of the matrix at index (row, col) (0-indexed).

- BinaryMatrix.dimensions() returns the dimensions of the matrix as a list of 2 elements [rows, cols], which means the matrix is rows x cols.

Submissions making more than 1000 calls to BinaryMatrix.get will be judged Wrong Answer. Also, any solutions that attempt to circumvent the judge will result in disqualification.

For custom testing purposes, the input will be the entire binary matrix mat. You will not have access to the binary matrix directly.

CODE:

```
class BinaryMatrix:
    def get(self, row, col):
        # This method would be provided by the problem environment
        pass

    def dimensions(self):
        # This method would be provided by the problem environment
        pass

def leftMostColumnWithOne(binaryMatrix: 'BinaryMatrix') -> int:
    # Get the dimensions of the matrix
    rows, cols = binaryMatrix.dimensions()

    # Initialize the starting position at the top-right corner
    current_row = 0
    current_col = cols - 1

    # Initialize the result as -1 (default if no 1 is found)
    leftmost_col = -1

    # Traverse the matrix
    while current_row < rows and current_col >= 0:
        if binaryMatrix.get(current_row, current_col) == 1:
            leftmost_col = current_col
            current_col -= 1  # Move left
        else:
            current_row += 1  # Move down

    return leftmost_col
```

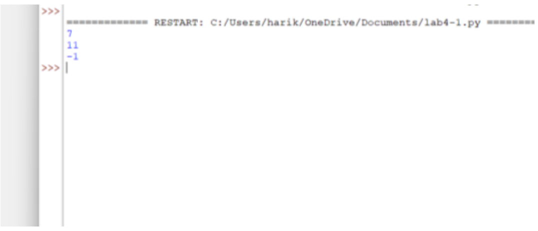
4. First Unique Number

You have a queue of integers, you need to retrieve the first unique integer in the queue.
Implement the FirstUnique class:

- FirstUnique(int[] nums) Initializes the object with the numbers in the queue.
- int showFirstUnique() returns the value of the first unique integer of the queue, and returns -1 if there is no such integer.
- void add(int value) insert value to the queue.

CODE:

```
from collections import deque
class FirstUnique:

    def __init__(self, nums):
        self.queue = deque()
        self.count = {}
        for num in nums:
            self.add(num)

    def showFirstUnique(self):
        # Keep removing elements from the front of the queue if they are not unique
        while self.queue and self.count[self.queue[0]] > 1:
            self.queue.popleft()
        # Return the first unique element
        if self.queue:
            return self.queue[0]
        else:
            return -1

    def add(self, value):
        if value in self.count:
            self.count[value] += 1
        else:
            self.count[value] = 1
            self.queue.append(value)


nums = [2, 3, 5, 2, 3, 7, 5, 11]
first_unique = FirstUnique(nums)
print(first_unique.showFirstUnique())
first_unique.add(7)
print(first_unique.showFirstUnique())
first_unique.add(11)
print(first_unique.showFirstUnique())
```

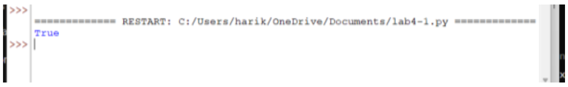


```
>>>
============= RESTART: C:/Users/harik/OneDrive/Documents/lab4-1.py ========
7
11
-1
>>> |
```

5. Check If a String Is a Valid Sequence from Root to Leaves Path in a Binary Tree
Given a binary tree where each path going from the root to any leaf form a valid sequence, check if a given string is a valid sequence in such binary tree.
We get the given string from the concatenation of an array of integers arr and the concatenation of all values of the nodes along a path results in a sequence in the given binary tree.

CODE:

```
def isValidSequence(root, arr):
    def dfs(node, index):
        # If node is None, return False
        if not node:
            return False

        # Check if the current node's value matches the current element in arr
        if node.val != arr[index]:
            return False

        # If it's the last element in arr, check if the node is a leaf node
        if index == len(arr) - 1:
            return not node.left and not node.right

        # Recursively check the left and right subtrees
        return (dfs(node.left, index + 1) or dfs(node.right, index + 1))

    # Start the DFS from the root and the first index of arr
    return dfs(root, 0)

# Example usage:
# Construct a binary tree
#      0
#     / \
#    1   0
#   /|   | \
#  0 1   0  1
# /|  \  \
# 0 1   0   0

root = TreeNode(0)
root.left = TreeNode(1)
root.right = TreeNode(0)
root.left.left = TreeNode(0)
root.left.right = TreeNode(1)
root.right.left = TreeNode(0)
root.right.right = TreeNode(1)
root.left.left.left = TreeNode(0)
root.left.left.right = TreeNode(1)
root.left.right.left = TreeNode(0)

root.right.right.right = TreeNode(0)


# Given sequence to check

arr = [0, 1, 0, 1]


print(isValidSequence(root, arr))
```



```
>>> ============= RESTART: C:/Users/harik/OneDrive/Documents/lab4-1.py =============
True
>>> |
```

6. Kids With the Greatest Number of Candies

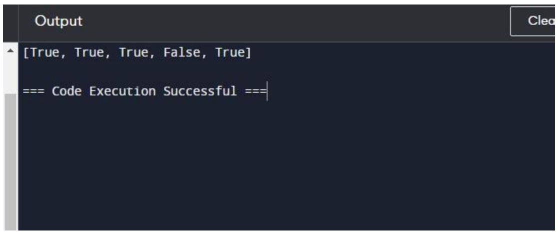
There are n kids with candies. You are given an integer array candies, where each candies[i] represents the number of candies the ith kid has, and an integer extraCandies, denoting the number of extra candies that you have.

Return a boolean array result of length n, where result[i] is true if, after giving the ith kid all the extraCandies, they will have the greatest number of candies among all the kids, or false otherwise.

Note that multiple kids can have the greatest number of candies.

CODE:

```
def kidsWithCandies(candies, extraCandies):

    # Step 1: Determine the maximum number of candies currently held by any kid

    max_candies = max(candies)


    # Step 2: Initialize the result array

    result = []


    # Step 3: For each kid, check if giving them all the extra candies will make them have the most candies

    for candy in candies:

        if candy + extraCandies >= max_candies:

            result.append(True)

        else:

            result.append(False)
    return result


candies = [2, 3, 5, 1, 3]

extraCandies = 3

print(kidsWithCandies(candies, extraCandies))
```



```
[True, True, True, False, True]

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

7. Max Difference You Can Get From Changing an Integer

You are given an integer num. You will apply the following steps exactly two times:

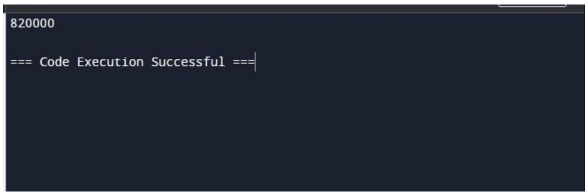
- Pick a digit x (0 <= x <= 9).
- Pick another digit y (0 <= y <= 9). The digit y can be equal to x.
- Replace all the occurrences of x in the decimal representation of num by y.
- The new integer cannot have any leading zeros, also the new integer cannot be 0.

Let a and b be the results of applying the operations to num the first and second times, respectively.

Return the max difference between a and b.

CODE:

```
def maxDiff(num):

    # Convert the number to string to facilitate replacement

    num_str = str(num)


    def replace_digit(n_str, x, y):

        # Replace digit x with y in the string representation of the number

        return n_str.replace(x, y)


    max_result = num

    min_result = num


    # To find the maximum number, we need to replace a digit with 9
    for digit in num_str:
        if digit != '9':
            max_str = replace_digit(num_str, digit, '9')
            max_result = max(max_result, int(max_str))
            break  # Only need to replace the first non-9 digit

    # To find the minimum number, we need to replace the first digit (if it's not '1') with '1'
    # or any other digit with '0' but ensuring it does not become zero
    if num_str[0] != '1':
        min_str = replace_digit(num_str, num_str[0], '1')
        min_result = min(min_result, int(min_str))
    else:
        for digit in num_str[1:]:
            if digit != '0' and digit != num_str[0]:
                min_str = replace_digit(num_str, digit, '0')
                min_result = min(min_result, int(min_str))
                break  # Only need to replace the first non-0 digit in the rest

    return max_result - min_result

num = 123456
print(maxDiff(num))
```

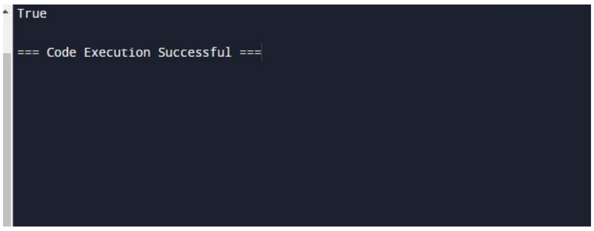


```
820000

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

8. Check If a String Can Break Another String
Given two strings: s1 and s2 with the same size, check if some permutation of string s1 can break some permutation of string s2 or vice-versa. In other words s2 can break s1 or vice-versa.
A string x can break string y (both of size n) if x[i] >= y[i] (in alphabetical order) for all i between 0 and n-1.

CODE:

```
def checkIfCanBreak(s1, s2):
    # Step 1: Sort both strings
    s1_sorted = sorted(s1)
    s2_sorted = sorted(s2)

    # Step 2: Check if s1 can break s2
    can_s1_break_s2 = True
    can_s2_break_s1 = True

    for i in range(len(s1_sorted)):
        if s1_sorted[i] < s2_sorted[i]:
            can_s1_break_s2 = False
        if s2_sorted[i] < s1_sorted[i]:
            can_s2_break_s1 = False

    # Step 3: Return True if either condition is satisfied
    return can_s1_break_s2 or can_s2_break_s1

# Example usage:
s1 = "abc"
s2 = "xya"
print(checkIfCanBreak(s1, s2))
```



```
True

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

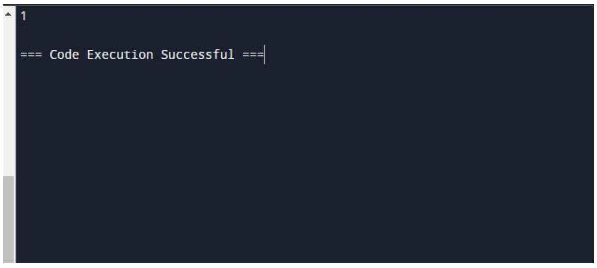
9. Number of Ways to Wear Different Hats to Each Other
There are n people and 40 types of hats labeled from 1 to 40.
Given a 2D integer array hats, where hats[i] is a list of all hats preferred by the ith person.
Return the number of ways that the n people wear different hats to each other.

Since the answer may be too large, return it modulo 109 + 7.

CODE:

```
def numberWays(hats):
    MOD = 10**9 + 7
    n = len(hats)

    # Create a list of people preferring each hat
    hat_to_people = [[] for _ in range(41)]
    for person in range(n):
        for hat in hats[person]:
            hat_to_people[hat].append(person)

    # Initialize DP table
    dp = [0] * (1 << n)
    dp[0] = 1

    # Iterate over each hat
    for hat in range(1, 41):
        # Update DP table from back to front to avoid overwriting
        for mask in range((1 << n) - 1, -1, -1):
            for person in hat_to_people[hat]:
                if mask & (1 << person) == 0:  # Check if the person has not been assigned a hat
yet
                    dp[mask | (1 << person)] += dp[mask]
                    dp[mask | (1 << person)] %= MOD

    # The answer is the number of ways to assign hats when all people have been assigned hats
    return dp[(1 << n) - 1]

# Example usage:
hats = [[3, 4], [4, 5], [5]]
```

```
print(numberWays(hats))
```

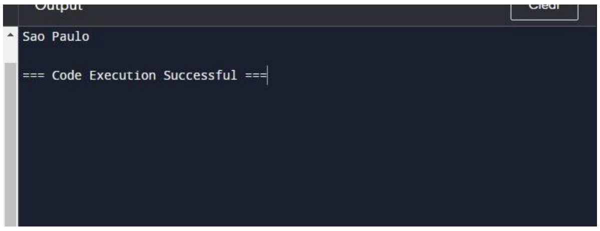


```
1

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

10. Destination City
You are given the array paths, where paths[i] = [cityAi, cityBi] means there exists a direct path going from cityAi to cityBi. Return the destination city, that is, the city without any path outgoing to another city.
It is guaranteed that the graph of paths forms a line without any loop, therefore, there will be exactly one destination city.

CODE:

```
def destCity(paths):
    # Set to track cities with outgoing paths
    outgoing = set()

    # Add each cityA (starting point of a path) to the outgoing set
    for path in paths:
        outgoing.add(path[0])

    # The destination city will be the one cityB not in the outgoing set
    for path in paths:
        if path[1] not in outgoing:
            return path[1]

# Example usage:
paths = [["London", "New York"], ["New York", "Lima"], ["Lima", "Sao Paulo"]]
print(destCity(paths))
```



```
Sao Paulo

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