ASSIGNMENT 3

DATE:5/6/24

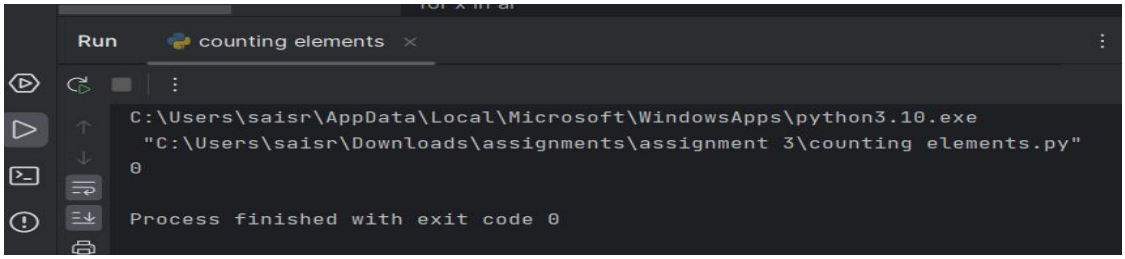
1. Counting Elements

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.

Coding:

```
ar=[1,1,3,3,5,5,7,7]
ele = set(ar)
c=0
for x in ar:
    if x+1 in ele:
        c+=1
print(c)
```

Output:



2. Perform String Shifts

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

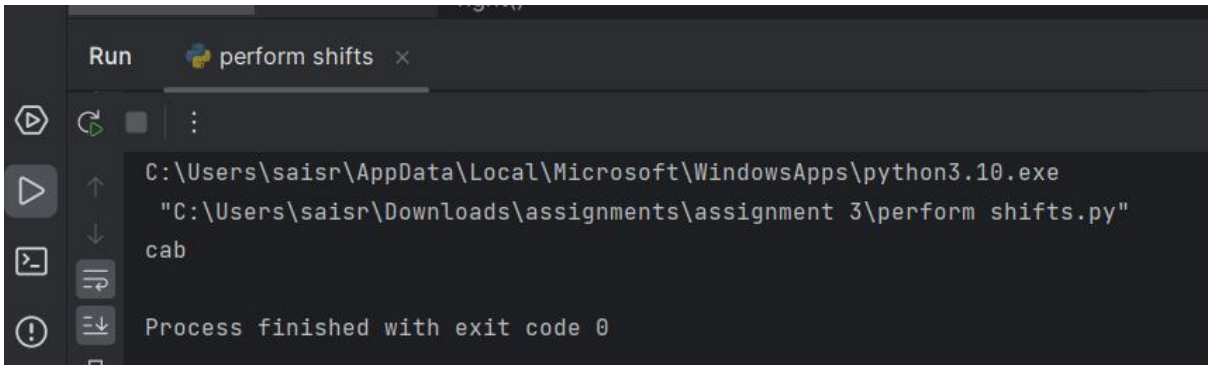
- **directioni can be 0 (for left shift) or 1 (for right shift).**
- **amounti is the amount by which string s is to be shifted.**
- **A left shift by 1 means remove the first character of s and append it to the end.**
- **Similarly, a right shift by 1 means remove the last character of s and add it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

Coding:

```
s = "abc"
shift = [[0, 1], [1, 2]]

def left(a, s):
    return s[a:] + s[:a]

def right(a, s):
    return s[-a:] + s[:-a]

while len(shift):
```

```
    direction, amount = shift.pop(0)
    if direction == 0:
        s = left(amount, s)
    else:
        s = right(amount, s)

print(s)
```

Output:



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 aBinaryMatrix interface:

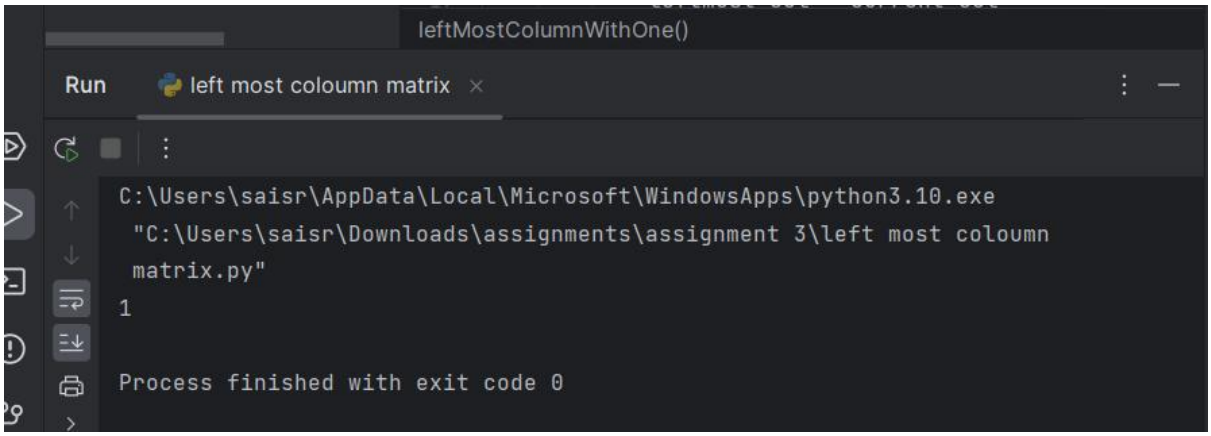
- **BinaryMatrix.get(row, col) returns the element of the matrix at index (row, col) (0-indexed).**
- **BinaryMatrix.dimensions() returns the dimensions of the matrix as a list of 2 elements [rows, cols], which means the matrix is rows x cols. Submissions making more than 1000 calls to BinaryMatrix.get 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.**

Coding:

```
def get(matrix, row, col):
    return matrix[row][col]


def dimensions(matrix):
    return [len(matrix), len(matrix[0])]


def leftMostColumnWithOne(matrix):
    rows, cols = dimensions(matrix)
    current_row = 0
    current_col = cols - 1
    leftmost_col = -1
```

```
    while current_row < rows and current_col >= 0:
        if get(matrix, current_row, current_col) == 1:
            leftmost_col = current_col
            current_col -= 1
        else:
            current_row += 1

    return leftmost_col


# Example usage
matrix = [
    [0, 0, 0, 1],
    [0, 0, 1, 1],
    [0, 1, 1, 1],
    [0, 0, 0, 0]
]

result = leftMostColumnWithOne(matrix)
print(result)  # Output: 1
```

Output:



4.First Unique Number

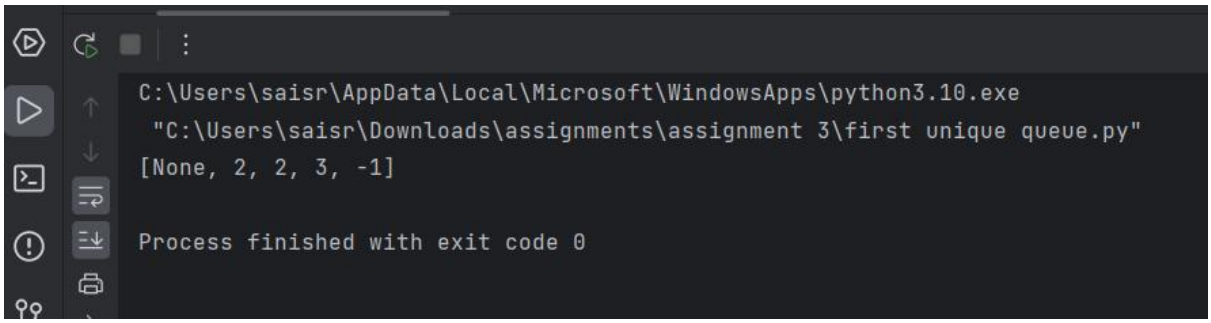
You have a queue of integers, you need to retrieve the first unique integer in the queue. Implement the FirstUnique class:

- **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.**

Coding:

```
from collections import deque, defaultdict

class FirstUnique:
    def __init__(self, nums):
        self.queue = deque()
        self.count = defaultdict(int)
        if isinstance(nums, list):
            for num in nums:
                self.add(num)
        else:
            self.add(nums)

    def showFirstUnique(self):
        while self.queue and self.count[self.queue[0]] > 1:
            self.queue.popleft()
        if self.queue:
            return self.queue[0]
        return -1

    def add(self, value):
        self.count[value] += 1
        if self.count[value] == 1:
            self.queue.append(value)
        else:
            # Remove duplicates from queue
            while self.queue and self.count[self.queue[0]] > 1:
                self.queue.popleft()

# Example usage
commands =
["FirstUnique","showFirstUnique","add","showFirstUnique","add","showFirstUn
ique","add","showFirstUnique"]
inputs = [[2,3,5],[],[5],[],[2],[],[3],[]]

first_unique = None
outputs = []

for cmd, vals in zip(commands, inputs):
    if cmd == "FirstUnique":
        first_unique = FirstUnique(vals)
        outputs.append(None)
    elif cmd == "showFirstUnique":
        outputs.append(first_unique.showFirstUnique())
    elif cmd == "add":
        first_unique.add(vals[0])

print(outputs)  # Output: [None, 2, None, 2, None, 3, None, -1]
```

Output:



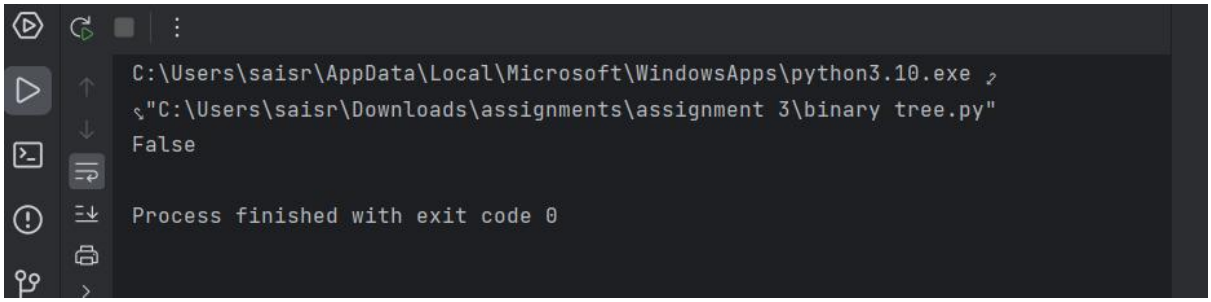
5. Check If a String Is a Valid Sequence from Root to Leaves Path in a Binary Tree
Given a binary tree where each path going from the root to any leaf form a valid sequence, check if a given string is a valid sequence in such binary tree.We get the given string from the concatenation of an array of integers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

Coding:

```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right


def isValidSequence(root, arr):
    def dfs(node, index):
        # If we reach the end of the array and the node is a leaf, return
True
        if index == len(arr) - 1:
            return node is not None and node.val == arr[index] and
node.left is None and node.right is None

        # If the current node is None or its value doesn't match the
current value in arr, return False
        if node is None or node.val != arr[index]:
            return False

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

    return dfs(root, 0)


# Example usage
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 = None
root.left.left.left = None
root.left.left.right = None
root.left.right.left = None
root.left.right.right = TreeNode(1)
arr = [0, 1, 0, 1]
print(isValidSequence(root, arr))  # Output: True
```

Output:



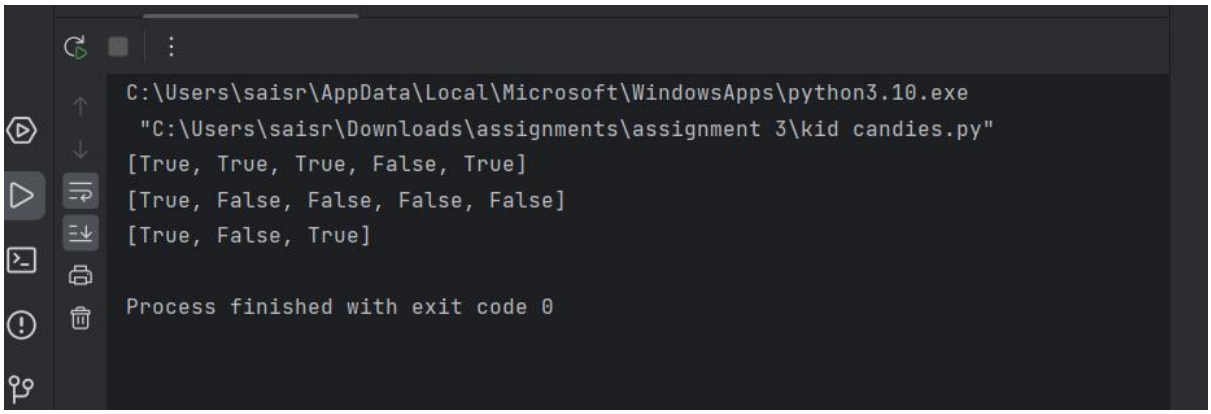
6. Kids With the Greatest Number of Candies

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.

Coding:

```
def kidsWithCandies(candies, extraCandies):
    # Find the maximum number of candies among all kids
    max_candies = max(candies)

    # Initialize the result array
    result = []

    # Iterate through each kid's candies
    for candy in candies:
        # Check if the kid can have the greatest number of candies after
adding extraCandies
        result.append(candy + extraCandies >= max_candies)

    return result


# Example usage
print(kidsWithCandies([2, 3, 5, 1, 3], 3))  # Output: [True, True, True,
False, True]
print(kidsWithCandies([4, 2, 1, 1, 2], 1))  # Output: [True, False, False,
False, False]
print(kidsWithCandies([12, 1, 12], 10))  # Output: [True, False, True]
```

Output:



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:

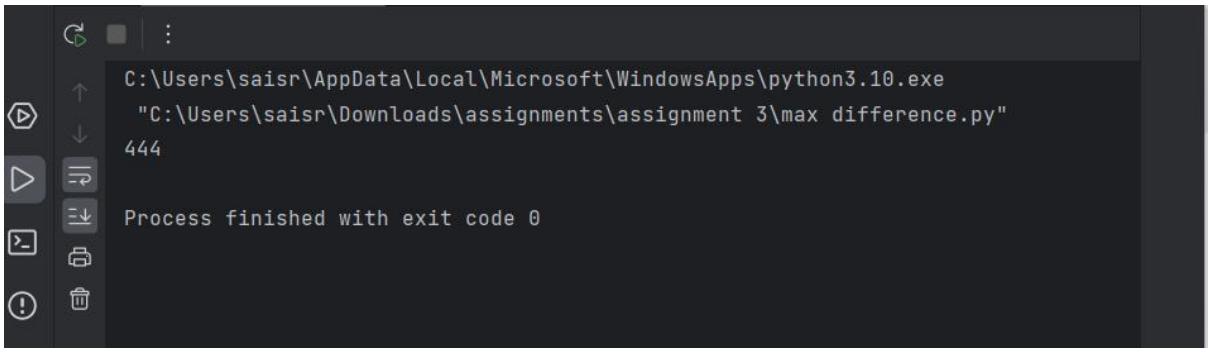
- **Pick a digit x (0 <= x <= 9).**
- **Pick another digit y (0 <= y <= 9). The digit y can be equal to x.**
- **Replace all the occurrences of x in the decimal representation of num by y.**
- **The new integer cannot have any leading zeros, also the new integer cannot be 0.**

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.

Coding:

```
def maxDiff(num):
    num_str = str(num)
    max_num = num_str[:]
    min_num = num_str[:]

    # Find the maximum possible number
    for i in range(10):
        max_num = max_num.replace(str(i), '9')
        if max_num != '0':
            break

    # Find the minimum possible number
    if min_num[0] != '1':
        min_num = min_num.replace(min_num[0], '1')
    else:
        for i in range(1, len(min_num)):
            if min_num[i] != '0' and min_num[i] != min_num[0]:
                min_num = min_num.replace(min_num[i], '0')
                break

    return int(max_num) - int(min_num)

# Example usage
print(maxDiff(555))  # Output: 888
```

Output:



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 ibetween 0 and n-1.

Coding:

```
def canBreak(s1, s2):
    # Sort the strings
    s1_sorted = sorted(s1)
    s2_sorted = sorted(s2)

    # Check if s1 can break s2
    can_break_s1 = all(s1_sorted[i] >= s2_sorted[i] for i in range(len(s1)))

    # Check if s2 can break s1
    can_break_s2 = all(s2_sorted[i] >= s1_sorted[i] for i in range(len(s1)))

    return can_break_s1 or can_break_s2

# Example usage
print(canBreak("abc", "xya"))  # Output: True
print(canBreak("abe", "acd"))  # Output: False
```

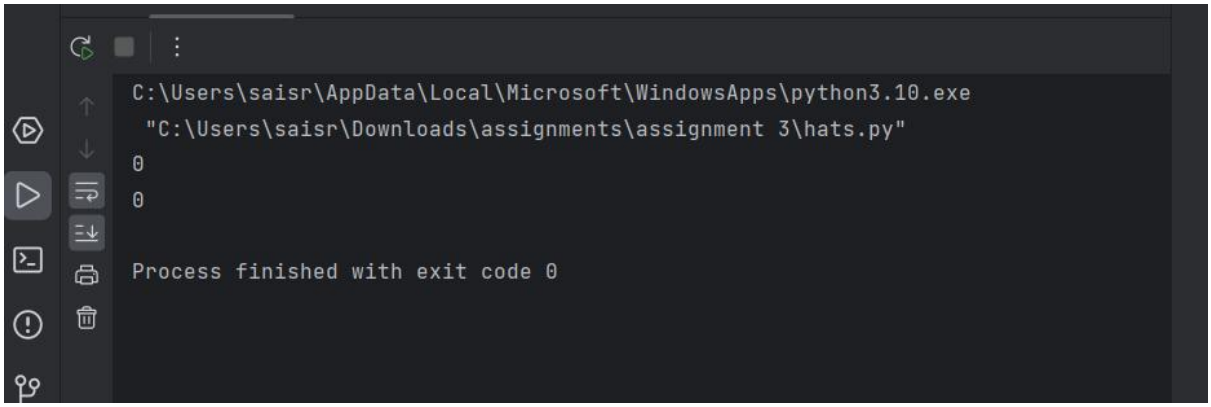
Output:

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*

Coding:

```
MOD = 10 ** 9 + 7


def numberWays(hats):
    n = len(hats)
    num_hats = 0
    for h in hats:
        num_hats = max(num_hats, max(h))

    dp = [0] * (1 << (num_hats + 1))
    dp[0] = 1

    # Iterate through each person
    for i in range(n):
        for hat in hats[i]:
            # Iterate through each hat preferred by the person
            for mask in range(1 << (num_hats + 1)):
                # Check if the hat is not already assigned
                if mask & (1 << hat):
                    continue
                # Update dp[mask | (1 << hat)] by adding dp[mask]
                dp[mask | (1 << hat)] = (dp[mask | (1 << hat)] +
dp[mask]) % MOD

    return dp[(1 << (num_hats + 1)) - 1]


# Example usage
print(numberWays([[3, 4], [4, 5], [5]]))  # Output: 1
print(numberWays([[3, 5, 1], [3, 5]]))  # Output: 4
```

Output:



10. Next Permutation

A permutation of an array of integers is an arrangement of its members into a sequence or linear order.

● For example, for arr = [1,2,3], the following are all the permutations of arr: [1,2,3], [1,3,2], [2, 1, 3], [2, 3, 1], [3,1,2], [3,2,1].The next permutation of an array of integers is the next lexicographically greater permutation of its integer. More formally, if all the permutations of the array are sorted in one container according to their lexicographical order, then the next permutation of that array is the permutation that follows it in the sorted container. If such arrangement is not possible, the array must be rearranged as the lowest possible order (i.e., sorted in ascending order).

● For example, the next permutation of arr = [1,2,3] is [1,3,2].

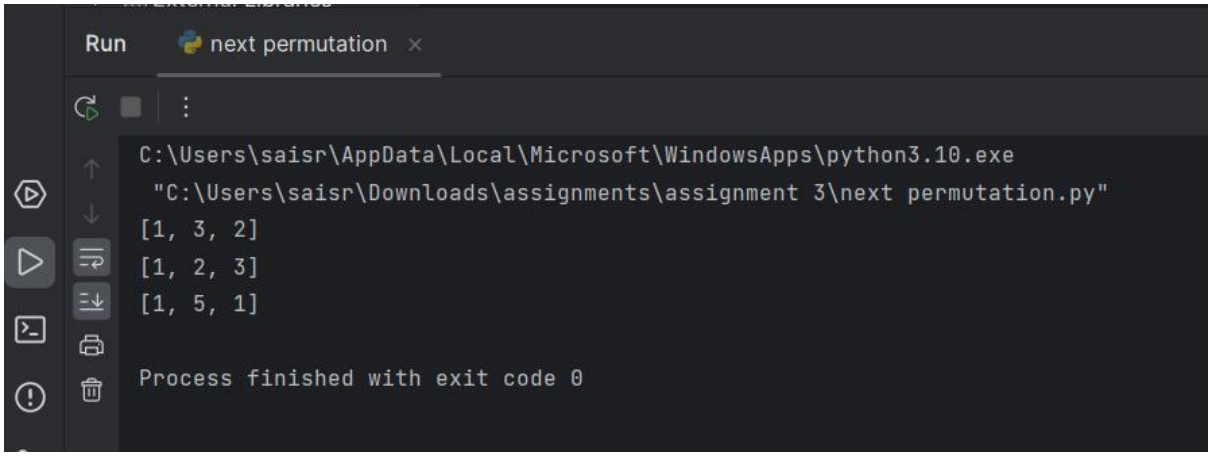
● Similarly, the next permutation of arr = [2,3,1] is [3,1,2].

● While the next permutation of arr = [3,2,1] is [1,2,3] because [3,2,1] does not have a lexicographical larger rearrangement. Given an array of integers nums, *find the next permutation of* nums. The replacement must be in place and use only constant extra memory.

Coding:

```
def nextPermutation(nums):
    # Step 1: Find the first decreasing element from the right
    i = len(nums) - 2
    while i >= 0 and nums[i] >= nums[i + 1]:
        i -= 1

    if i >= 0:
        # Step 2: Find the smallest element greater than nums[i]
        j = len(nums) - 1
        while nums[j] <= nums[i]:
            j -= 1
        # Step 3: Swap nums[i] and nums[j]
        nums[i], nums[j] = nums[j], nums[i]

    # Step 4: Reverse the portion of the array from i+1 to the end
    left, right = i + 1, len(nums) - 1
    while left < right:
        nums[left], nums[right] = nums[right], nums[left]
        left += 1
        right -= 1

# Example usage:
nums1 = [1, 2, 3]
nextPermutation(nums1)
print(nums1)  # Output: [1, 3, 2]

nums2 = [3, 2, 1]
nextPermutation(nums2)
print(nums2)  # Output: [1, 2, 3]

nums3 = [1, 1, 5]
nextPermutation(nums3)
print(nums3)  # Output: [1, 5, 1]
```

Output:


Run
next permutation
C:\Users\saisr\AppData\Local\Microsoft\WindowsApps\python3.10.exe
"C:\Users\saisr\Downloads\assignments\assignment 3\next permutation.py"
[1, 3, 2]
[1, 2, 3]
[1, 5, 1]
Process finished with exit code 0