Run | Debug | Stop | Share | Save | { } Beautify

Language Python 3

main.py

```python
def count_elements(arr):
    return sum(1 for x in arr if x + 1 in arr)

arr = [1, 2, 3]
print(count_elements(arr))

```

Input

```
2


...Program finished with exit code 0
Press ENTER to exit console.
```

Run | Debug | Stop | Share | Save | { } Beautify | Language Python 3

main.py

```python
def string_shift(s, shift):
    for direction, amount in shift:
        if direction == 0:
            s = s[amount:] + s[:amount]
        else:
            s = s[-amount:] + s[:-amount]
    return s

s = "abc"
shift = [(0, 1), (1, 2)]
result = string_shift(s, shift)
print(result)

```

Input

```
cab


...Program finished with exit code 0
Press ENTER to exit console.
```

Run | Debug | Stop | Share | Save | { } Beautify

Language Python 3

main.py

```python
def leftMostColumnWithOne(binaryMatrix: 'BinaryMatrix') -> int:
    rows, cols = binaryMatrix.dimensions()
    current_row = 0
    current_col = cols - 1
    leftmost_col = -1

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

    return leftmost_col

```

Input

```
...Program finished with exit code 0
Press ENTER to exit console.
```

Run | Debug | Stop | Share | Save | { } Beautify | Language Python 3

main.py

```python
from collections import deque, OrderedDict

class FirstUnique:
    def __init__(self, nums):
        self.queue = deque()
        self.count = {}
        self.unique = OrderedDict()

        for num in nums:
            self.add(num)

    def showFirstUnique(self):

        for key in self.unique:
            if self.count[key] == 1:
                return key
        return -1

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

        self.queue.append(value)

        if self.count[value] == 1:
            self.unique[value] = None
        elif value in self.unique:
```

Input

```
...Program finished with exit code 0
Press ENTER to exit console.
```

Run | Debug | Stop | Share | Save | { } Beautify | Language Python 3

main.py

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

def isValidSequence(root: TreeNode, arr: [int]) -> bool:
    def dfs(node, index):
        if not node:
            return False

        if index >= len(arr) or node.val != arr[index]:
            return False

        if not node.left and not node.right and index == len(arr) - 1:
            return True

        return dfs(node.left, index + 1) or dfs(node.right, index + 1)

    return dfs(root, 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.left.right.left = TreeNode(0)
root.left.right.right = TreeNode(1)
```

input

```
False
False


...Program finished with exit code 0
Press ENTER to exit console.
```

Run | Debug | Stop | Share | Save | { } Beautify | Language Python 3

main.py

```python
def kids_with_candies(candies, extra_candies):
    """
    Returns a boolean array where each element represents whether the kid at that index
    can have the greatest number of candies after receiving all the extra candies.

    Args:
    candies (list): A list of integers representing the number of candies each kid has.
    extra_candies (int): The number of extra candies available.

    Returns:
    list: A boolean list where each element is True if the kid at that index can have
          the greatest number of candies, False otherwise.
    """
    max_candies = max(candies)

    result = [False] * len(candies)

    # Iterate over the kids
    for i in range(len(candies)):
        # If the kid's candies plus the extra candies is greater than or equal to the max candies,
        # they can have the greatest number of candies
        if candies[i] + extra_candies >= max_candies:
            result[i] = True

    return result

print(kids_with_candies([2,3,5,1,3], 3))
```

input

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


...Program finished with exit code 0
Press ENTER to exit console.
```

Run | Debug | Stop | Share | Save | { } Beautify | Language Python 3

main.py

```python
def maximumGap(num: int) -> int:
    num_str = str(num)
    max_num = num
    min_num = num

    for i in range(10):
        for j in range(10):
            new_str = ''.join([str(j) if c == str(i) else c for c in num_str])
            if new_str[0] == '0' and len(new_str) > 1:
                continue
            new_num = int(new_str)
            if new_num != num:
                max_num = max(max_num, new_num)
                min_num = min(min_num, new_num)

    return max_num - min_num
print(maximumGap(555))
```

input

```
888


...Program finished with exit code 0
Press ENTER to exit console.
```

Run | Debug | Stop | Share | Save | { } Beautify | Language Python 3

main.py

```python
def checkIfCanBreak(s1, s2):
    s1_sorted = sorted(s1)
    s2_sorted = sorted(s2)

    can_break_s2 = all(x >= y for x, y in zip(s1_sorted, s2_sorted))
    can_break_s1 = all(x >= y for x, y in zip(s2_sorted, s1_sorted))

    return can_break_s2 or can_break_s1
print(checkIfCanBreak("abc", "xya"))
```

input

```
True


...Program finished with exit code 0
Press ENTER to exit console.
```

Run | Debug | Stop | Share | Save | { } Beautify | Language Python 3

main.py

```python
import sys
from typing import List

MOD = int(1e9) + 7

def numberWays(hats: List[List[int]]) -> int:
    n = len(hats)
    dp = [[0] * 41 for _ in range(n + 1)]
    dp[0][0] = 1

    for i in range(1, n + 1):
        for j in range(41):
            dp[i][j] = dp[i - 1][j]
            for k in hats[i - 1]:
                if j >= k:
                    dp[i][j] += dp[i - 1][j - k]
                    dp[i][j] %= MOD

    return dp[n][0]
print(numberWays([[3,4],[4,5],[5]]))
```

input

```
1


...Program finished with exit code 0
Press ENTER to exit console.
```

Run | Debug | Stop | Share | Save | { } Beautify | Language Python 3

main.py

```python
def nextPermutation(nums):
    i = len(nums) - 2
    while i >= 0 and nums[i + 1] <= nums[i]:
        i -= 1
    if i >= 0:
        j = len(nums) - 1
        while j > i and nums[j] <= nums[i]:
            j -= 1
        nums[i], nums[j] = nums[j], nums[i]
    left, right = i + 1, len(nums) - 1
    while left < right:
        nums[left], nums[right] = nums[right], nums[left]
        left += 1
        right -= 1
nums = [1, 2, 3]
nextPermutation(nums)
print(nums)
```

input

```
[1, 3, 2]


...Program finished with exit code 0
Press ENTER to exit console.
```