

ASSIGNMENT-3

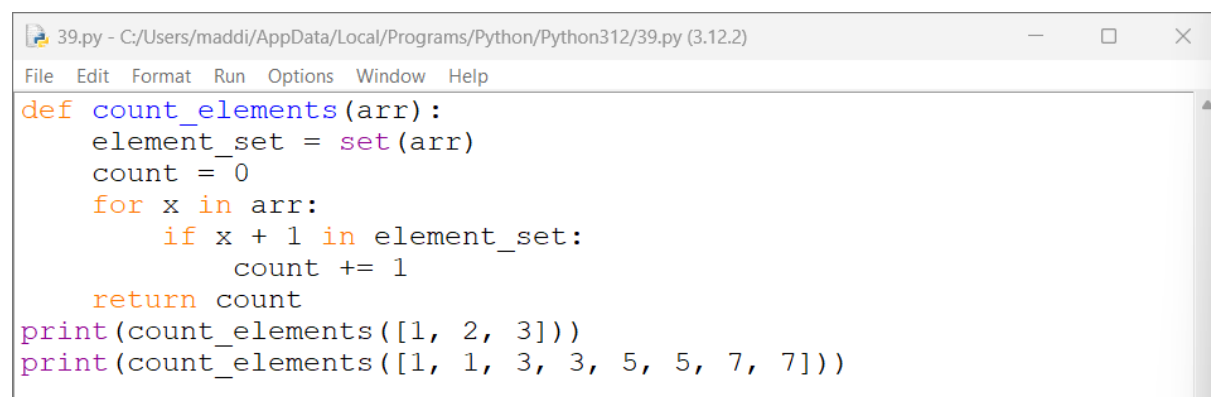
NAME : M.Mahathi

REGISTER NUMBER : 192324098

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.

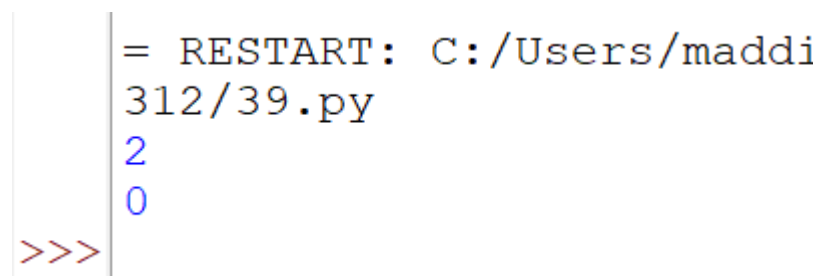
Example Input: arr = [1,2,3] Output: 2 Explanation: 1 and 2 are counted cause 2 and 3 are in arr.

PROGRAM:



```
39.py - C:/Users/maddi/AppData/Local/Programs/Python/Python312/39.py (3.12.2)
File Edit Format Run Options Window Help
def count_elements(arr):
    element_set = set(arr)
    count = 0
    for x in arr:
        if x + 1 in element_set:
            count += 1
    return count
print(count_elements([1, 2, 3]))
print(count_elements([1, 1, 3, 3, 5, 5, 7, 7]))
```

OUTPUT:



```
= RESTART: C:/Users/maddi
312/39.py
2
0
>>>
```

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.

Example 1: Input: s = "abc", shift = [[0,1],[1,2]] Output: "cab" Explanation: [0,1] means shift to left by 1. "abc" -> "bca" [1,2] means shift to right by 2. "bca" -> "cab".

PROGRAM:

```

39.py - C:/Users/maddi/AppData/Local/Programs/Python/Python312/39.py (3.12.2)
File Edit Format Run Options Window Help
def string_shift(s, shift):
    net_shift = 0
    for direction, amount in shift:
        if direction == 0:
            net_shift -= amount
        else:
            net_shift += amount
    net_shift %= len(s)
    return s[-net_shift:] + s[:-net_shift] if net_shift != 0 else s
print(string_shift("abc", [[0, 1], [1, 2]]))
print(string_shift("abcdefg", [[1, 1], [1, 1], [0, 2], [1, 3]]))

```

OUTPUT:

```

= RESTART: C:,
312/39.py
cab
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.

Example 1: Input: `mat = [[0,0],[1,1]]` Output: 0

PROGRAM:

```

39.py - C:/Users/maddi/AppData/Local/Programs/Python/Python312/39.py (3.12.2)
File Edit Format Run Options Window Help
class BinaryMatrix:
    def __init__(self, mat):
        self.mat = mat
    def get(self, row, col):
        return self.mat[row][col]
    def dimensions(self):
        return [len(self.mat), len(self.mat[0])]
def leftmost_column_with_one(binaryMatrix):
    rows, cols = binaryMatrix.dimensions()
    current_row, current_col = 0, cols - 1
    leftmost = -1
    while current_row < rows and current_col >= 0:
        if binaryMatrix.get(current_row, current_col) == 1:
            leftmost = current_col
            current_col -= 1
        else:
            current_row += 1
    return leftmost
print(leftmost_column_with_one(BinaryMatrix([[0, 0], [1, 1]])))
print(leftmost_column_with_one(BinaryMatrix([[0, 0], [0, 1]])))
print(leftmost_column_with_one(BinaryMatrix([[0, 0], [0, 0]])))

```

OUTPUT:

```

= RESTART: C:/Users.
0
1
-1
>>>

```

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.

Example 1: Input:

```
["FirstUnique","showFirstUnique","add","showFirstUnique","add","showFirstUnique","add","showFirstUnique"]
[[[2,3,5]],[5],[2],[3],[]]
```

Output: [null,2,null,2,null,3,null,-1] Explanation: FirstUnique firstUnique = new FirstUnique([2,3,5]);
firstUnique.showFirstUnique(); // return 2 firstUnique.add(5); // the queue is now [2,3,5,5]
firstUnique.showFirstUnique(); // return 2 firstUnique.add(2); // the queue is now [2,3,5,5,2]
firstUnique.showFirstUnique(); // return 3 firstUnique.add(3); // the queue is now [2,3,5,5,2,3]
firstUnique.showFirstUnique(); // return -1

PROGRAM:

```
39.py - C:/Users/maddi/AppData/Local/Programs/Python/Python312/39.py (3.12.2)
File Edit Format Run Options Window Help
from collections import deque, Counter
class FirstUnique:
    def __init__(self, nums):
        self.queue = deque(nums)
        self.count = Counter(nums)
    def showFirstUnique(self):
        while self.queue and self.count[self.queue[0]] > 1:
            self.queue.popleft()
        return self.queue[0] if self.queue else -1
    def add(self, value):
        self.queue.append(value)
        self.count[value] += 1
firstUnique = FirstUnique([2, 3, 5])
print(firstUnique.showFirstUnique())
firstUnique.add(5)
print(firstUnique.showFirstUnique())
firstUnique.add(2)
print(firstUnique.showFirstUnique())
firstUnique.add(3)
print(firstUnique.showFirstUnique())
```

OUTPUT:

```
= RESTART: C:/Use
312/39.py
2
2
3
-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.

Example 1: Input: root = [0,1,0,0,1,0,null,null,1,0,0], arr = [0,1,0,1]

Output: true Explanation: The path 0 -> 1 -> 0 -> 1 is a valid sequence (green color in the figure).

Other valid sequences are: 0 -> 1 -> 1 -> 0 0 -> 0 -> 0

PROGRAM:

```

*39.py - C:/Users/maddi/AppData/Local/Programs/Python/Python312/39.py (3.12.2)*
File Edit Format Run Options Window Help

class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
def is_valid_sequence(root, arr):
    def dfs(node, arr, index):
        if not node or 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, arr, index + 1) or dfs(node.right, arr, index + 1)
    return dfs(root, arr, 0)
root = TreeNode(0, TreeNode(1, TreeNode(0, None, TreeNode(1)), TreeNode(1, TreeNode(0))), TreeNode(0, TreeNode(0)))
print(is_valid_sequence(root, [0, 1, 0, 1]))
print(is_valid_sequence(root, [0, 0, 1]))
print(is_valid_sequence(root, [0, 1, 1]))

```

OUTPUT:

```

= RESTART: C:/Users
312/39.py
True
False
False
>>>

```

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 i th 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 i th 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.

Example 1: Input: `candies = [2,3,5,1,3]`, `extraCandies = 3`

Output: `[true,true,true,false,true]`

Explanation: If you give all `extraCandies` to: - Kid 1, they will have $2 + 3 = 5$ candies, which is the greatest among the kids. - Kid 2, they will have $3 + 3 = 6$ candies, which is the greatest among the kids. - Kid 3, they will have $5 + 3 = 8$ candies, which is the greatest among the kids. - Kid 4, they will have $1 + 3 = 4$ candies, which is not the greatest among the kids. - Kid 5, they will have $3 + 3 = 6$ candies, which is the greatest among the kids.

PROGRAM:

```

*39.py - C:/Users/maddi/AppData/Local/Programs/Python/Python312/39.py (3.12.2)*
File Edit Format Run Options Window Help

def kids_with_candies(candies, extraCandies):
    max_candies = max(candies)
    return [candy + extraCandies >= max_candies for candy in candies]
print(kids_with_candies([2, 3, 5, 1, 3], 3))
print(kids_with_candies([4, 2, 1, 1, 2], 1))
print(kids_with_candies([12, 1, 12], 10))

```

OUTPUT:

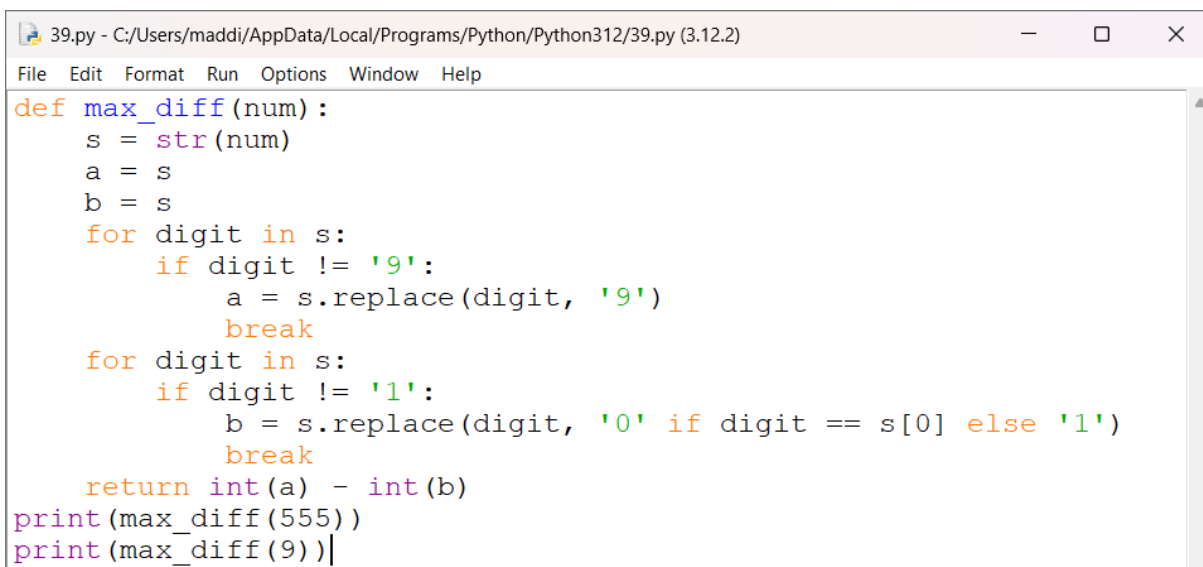
```
= RESTART: C:/Users/maddi/AppData/Local
[True, True, True, False, True]
[True, False, False, False, False]
[True, False, True]
>>>
```

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 ($0 \leq x \leq 9$). • Pick another digit y ($0 \leq y \leq 9$). 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.

Example 1: Input: num = 555 Output: 888

Explanation: The first time pick x = 5 and y = 9 and store the new integer in a. The second time pick x = 5 and y = 1 and store the new integer in b. We have now a = 999 and b = 111 and max difference = 888

PROGRAM:



```
39.py - C:/Users/maddi/AppData/Local/Programs/Python/Python312/39.py (3.12.2)
File Edit Format Run Options Window Help
def max_diff(num):
    s = str(num)
    a = s
    b = s
    for digit in s:
        if digit != '9':
            a = s.replace(digit, '9')
            break
    for digit in s:
        if digit != '1':
            b = s.replace(digit, '0' if digit == s[0] else '1')
            break
    return int(a) - int(b)
print(max_diff(555))
print(max_diff(9))
```

OUTPUT:

```
= RESTART: C:/Users:
312/39.py
999
9
>>>
```

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 $x[i] \geq y[i]$ (in alphabetical order) for all i between 0 and n-1.

Example 1: Input: s1 = "abc", s2 = "xya" Output: true

Explanation: "ayx" is a permutation of s2="xya" which can break to string "abc" which is a permutation of s1="abc".

PROGRAM:

```
39.py - C:/Users/maddi/AppData/Local/Programs/Python/Python312/39.py (3.12.2)
File Edit Format Run Options Window Help
def check_if_can_break(s1, s2):
    s1, s2 = sorted(s1), sorted(s2)
    return all(c1 >= c2 for c1, c2 in zip(s1, s2)) or all(c2 >= c1 for c1, c2 in zip(s1, s2))
print(check_if_can_break("abc", "xya"))
print(check_if_can_break("abe", "acd"))
print(check_if_can_break("leetcode", "interview"))
```

OUTPUT:

```
= RESTART: C:/User
True
False
True
>>>
```

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 $10^9 + 7$.

Example 1: Input: hats = [[3,4],[4,5],[5]] Output: 1

Explanation: There is only one way to choose hats given the conditions. First person choose hat 3, Second person choose hat 4 and last one hat 5.

PROGRAM:

```

39.py - C:/Users/maddi/AppData/Local/Programs/Python/Python312/39.py (3.12.2)
File Edit Format Run Options Window Help
MOD = 10**9 + 7

def number_ways(hats):
    from collections import defaultdict
    dp = defaultdict(int)
    dp[0] = 1
    all_mask = (1 << len(hats)) - 1
    hat_to_people = defaultdict(list)

    for i, hat_list in enumerate(hats):
        for hat in hat_list:
            hat_to_people[hat].append(i)
    for hat in range(1, 41):
        new_dp = dp.copy()
        for mask, ways in dp.items():
            for person in hat_to_people[hat]:
                if mask & (1 << person) == 0:
                    new_dp[mask | (1 << person)] = (new_dp[mask | (1 << person)] + ways) % MOD
        dp = new_dp
    return dp[all_mask]
print(number_ways([[3, 4], [4, 5], [5]]))
print(number_ways([[3, 5, 1], [3, 5]]))
print(number_ways([[1, 2, 3, 4], [1, 2, 3, 4], [1, 2, 3, 4], [1, 2, 3, 4]]))

```

OUTPUT:

```

= RESTART: C:/User
1
4
24
>>> |

```

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.

Example 1: Input: paths = [["London","New York"],["New York","Lima"],["Lima","Sao Paulo"]]

Output: "Sao Paulo" Explanation: Starting at "London" city you will reach "Sao Paulo" city which is the destination city. Your trip consist of: "London" -> "New York" -> "Lima" -> "Sao Paulo".

PROGRAM:


```
39.py - C:/Users/maddi/AppData/Local/Programs/Python/Python312/39.py (3.12.2)
File Edit Format Run Options Window Help
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
def bst_from_preorder(preorder):
    def helper(lower=float('-inf'), upper=float('inf')):
        nonlocal idx
        if idx == len(preorder):
            return None
        val = preorder[idx]
        if val < lower or val > upper:
            return None
        idx += 1
        root = TreeNode(val)
        root.left = helper(lower, val)
        root.right = helper(val, upper)
        return root
    idx = 0
    return helper()
def print_inorder(node):
    if node:
        print_inorder(node.left)
        print(node.val, end=' ')
        print_inorder(node.right)
preorder1 = [8, 5, 1, 7, 10, 12]
preorder2 = [10, 5, 1, 7, 40, 50]
bst1 = bst_from_preorder(preorder1)
bst2 = bst_from_preorder(preorder2)
print("Inorder traversal of BST from preorder1:")
print_inorder(bst1)
print("\nInorder traversal of BST from preorder2:")
print_inorder(bst2)
```

OUTPUT:

```
= RESTART: C:/Users/maddi/AppData/Local/Python/Python312/39.py
Inorder traversal of BST from preorder1:
1 5 7 8 10 12
Inorder traversal of BST from preorder2:
1 5 7 10 40 50
>>>
```