

1. Counting Elements

```
main.py  Save Run Output Clear
1- def count_elements(arr):
2-     return sum(1 for x in arr if x + 1 in arr)
3
4- # Example 1
5- arr1 = [1, 2, 3]
6- print(count_elements(arr1)) # Output: 2
7
8- # Example 2
9- arr2 = [1, 1, 3, 3, 5, 5, 7, 7]
10- print(count_elements(arr2)) # Output: 4
```

```
2
0

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

2. Perform String Shifts

```
main.py  Save Run Output
1- def stringShift(s, shift):
2-     total_shift = 0
3-     for sh in shift:
4-         if sh[0] == 0:
5-             total_shift -= sh[1]
6-         else:
7-             total_shift += sh[1]
8
9-     total_shift %= len(s)
10-    return s[-total_shift:] + s[:-total_shift]
11- s1 = "abc"
12- shift1 = [[0,1],[1,2]]
13- print(stringShift(s1, shift1))
14- s2 = "abcdefg"
15- shift2 = [[1,1],[1,1],[0,2],[1,3]]
16- print(stringShift(s2, shift2))
```

```
cab
efgabcd

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

3. Leftmost Column with at Least a One

```
main.py  Save Run Output
1- def generate_row_sorted_binary_matrix(rows, cols):
2-     import random
3-     matrix = [[random.randint(0, 1) for _ in range(cols)] for _ in range(rows)]
4-     for row in matrix:
5-         row.sort()
6-     return matrix
7
8- # Example Usage
9- rows = 2
10- cols = 2
11- binary_matrix = generate_row_sorted_binary_matrix(rows, cols)
12- print(binary_matrix)
```

```
[[1, 1], [1, 1]]

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

4. First Unique Number

```
main.py | Save | Run | Output
1 from collections import deque
2
3 class FirstUnique:
4     def __init__(self, nums):
5         self.queue = deque()
6         self.count = {}
7         for num in nums:
8             self.add(num)
9     def showFirstUnique(self):
10        while self.queue and self.count[self.queue[0]] > 1:
11            self.queue.popleft()
12        if self.queue:
13            return self.queue[0]
14        return -1
15    def add(self, value):
16        if value in self.count:
17            self.count[value] += 1
18        else:
19            self.count[value] = 1
20            self.queue.append(value)
21 firstUnique = FirstUnique([2, 3, 5])
22 print(firstUnique.showFirstUnique())
23 firstUnique.add(5)
24 print(firstUnique.showFirstUnique())
25 firstUnique.add(2)
26 print(firstUnique.showFirstUnique())
```

```
2
2
3
-1
=== Code Execution Successful ===
```

5. Check If a String Is a Valid Sequence from Root to Leaves Path in a Binary Tree

```
main.py | Save | Run | Output
1 class TreeNode:
2     def __init__(self, x):
3         self.val = x
4         self.left = None
5         self.right = None
6 def isValidSequence(root, sequence):
7     def dfs(node, index):
8         if not node or index >= len(sequence) or node.val != sequence[index]:
9             return False
10        if not node.left and not node.right and index == len(sequence) - 1:
11            return True
12        return dfs(node.left, index + 1) or dfs(node.right, index + 1)
13    return dfs(root, 0)
14 root = TreeNode(0)
15 root.left = TreeNode(1)
16 root.right = TreeNode(0)
17 root.left.left = TreeNode(0)
18 root.left.right = TreeNode(1)
19 root.right.left = TreeNode(0)
20 root.left.left.right = TreeNode(1)
21 root.left.right.left = TreeNode(0)
22 root.left.right.right = TreeNode(0)
23 sequence = [0, 1, 1]
24 print(isValidSequence(root, sequence))
```

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

6. Kids With the Greatest Number of Candies

```
main.py | Save | Run | Output
1 def kidsWithCandies(candies, extraCandies):
2     max_candies = max(candies)
3     result = [candy + extraCandies >= max_candies for candy in candies]
4     return result
5 candies = [2, 3, 5, 1, 3]
6 extraCandies = 3
7 print(kidsWithCandies(candies, extraCandies))
```

```
[True, True, True, False, True]
=== Code Execution Successful ===
```

7. Max Difference You Can Get From Changing an Integer

main.py	Output
<pre>1 def max_diff(num): 2 num_str = str(num) 3 max_num = int('9' + num_str[1:].replace('9','8')) 4 min_num = int(('1' if num_str[0] != '1' else '0') + num_str[1:].replace(num_str[0], '1')) 5 return max_num - min_num 6 print(max_diff(9))</pre>	<pre>8 === Code Execution Successful ===</pre>

8. Check If a String Can Break Another String

main.py	Output
<pre>1 def canBreak(s1, s2): 2 sorted_s1 = sorted(s1) 3 sorted_s2 = sorted(s2) 4 s1_breaks_s2 = True 5 s2_breaks_s1 = True 6 7 for c1, c2 in zip(sorted_s1, sorted_s2): 8 if c1 < c2: 9 s1_breaks_s2 = False 10 if c2 < c1: 11 s2_breaks_s1 = False 12 return s1_breaks_s2 or s2_breaks_s1 13 s1 = "abc" 14 s2 = "xya" 15 print(canBreak(s1, s2)) 16 s1 = "abe" 17 s2 = "acd" 18 print(canBreak(s1, s2))</pre>	<pre>True False === Code Execution Successful ===</pre>

9. Number of Ways to Wear Different Hats to Each Other

main.py	Output
<pre>1 from collections import defaultdict 2 def numberWays(hats): 3 MOD = 10**9 + 7 4 n = len(hats) 5 hat_to_people = defaultdict(list) 6 for person, hat_list in enumerate(hats): 7 for hat in hat_list: 8 hat_to_people[hat].append(person) 9 memo = {} 10 def dp(hat, mask): 11 if mask == (1 << n) - 1: 12 return 1 13 if hat > 40: 14 return 0 15 if (hat, mask) in memo: 16 return memo[(hat, mask)] 17 ways = dp(hat + 1, mask) 18 for person in hat_to_people[hat]: 19 if not (mask & (1 << person)): 20 ways += dp(hat + 1, mask (1 << person)) 21 ways %= MOD 22 memo[(hat, mask)] = ways 23 return dp(1, 0) 24 hats = [25 [2, 6, 1]</pre>	<pre>4 === Code Execution Successful ===</pre>

10. Destination City

main.py	Output
<pre>1 def destCity(paths): 2 start_cities = set(cityA for cityA, cityB in paths) 3 for cityA, cityB in paths: 4 if cityB not in start_cities: 5 return cityB 6 paths = [["London", "New York"], ["New York", "Lima"], ["Lima", "Sao Paulo"]] 7 print(destCity(paths))</pre>	<pre>Sao Paulo === Code Execution Successful ===</pre>