

ASSIGNMENT-6

Name: Shaik jabbar basha

1. Convert the Temperature You are given a non-negative floating point number rounded to two decimal places celsius, that denotes the temperature in Celsius. You should convert Celsius into Kelvin and Fahrenheit and return it as an array `ans = [kelvin, fahrenheit]`. Return the array `ans`. Answers within 10⁻⁵ of the actual answer will be accepted. Note that: • Kelvin = Celsius + 273.15 • Fahrenheit = Celsius * 1.80 + 32.00 Example 1: Input: celsius = 36.50 Output: [309.65000, 97.70000] Explanation: Temperature at 36.50 Celsius converted in Kelvin is 309.65 and converted in Fahrenheit is 97.70. Example 2: Input: celsius = 122.11 Output: [395.26000, 251.79800] Explanation: Temperature at 122.11 Celsius converted in Kelvin is 395.26 and converted in Fahrenheit is 251.798. Constraints: 0

PROGRAM:

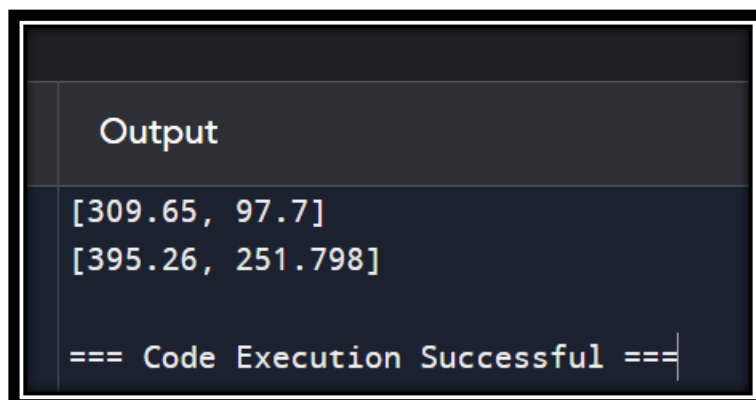


```
Python Online Compiler

main.py

1 def convert_temperature(celsius):
2     kelvin = celsius + 273.15
3     fahrenheit = celsius * 1.80 + 32.00
4     return [round(kelvin, 5), round(fahrenheit, 5)]
5
6 # Example usage:
7 print(convert_temperature(36.50)) # Output: [309.65000, 97.70000]
8 print(convert_temperature(122.11)) # Output: [395.26000, 251.79800]
9
```

Output:



```
Output

[309.65, 97.7]
[395.26, 251.798]

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

2. Number of Subarrays With LCM Equal to K Given an integer array nums and an integer k, return the number of subarrays of nums where the least common multiple of the subarray's elements is k. A subarray is a contiguous non-empty sequence of elements within an array. The least common multiple of an array is the smallest positive integer that is divisible by all the array elements. Example 1: Input: nums = [3,6,2,7,1], k = 6 Output: 4 Explanation: The subarrays of nums where 6 is the least common multiple of all the subarray's elements are: - [3,6,2,7,1] - [3,6,2,7,1] - [3,6,2,7,1] - [3,6,2,7,1] Example 2: Input: nums = [3], k = 2 Output: 0 Explanation: There are no subarrays of nums where 2 is the least common multiple of all the subarray's elements. Constraints: • $1 \leq \text{nums.length} \leq 1000$ • $1 \leq \text{nums}[i]$,

Program:

main.py

```
1  from math import gcd
2  from functools import reduce
3  def lcm(a, b):
4      return abs(a * b) // gcd(a, b)
5  def lcm_of_list(lst):
6      return reduce(lcm, lst)
7  def subarrays_with_lcm_k(nums, k):
8      n = len(nums)
9      count = 0
10     for i in range(n):
11         for j in range(i, n):
12             subarray = nums[i:j+1]
13             subarray_lcm = lcm_of_list(subarray)
14             if subarray_lcm == k:
15                 count += 1
16
17     return count
18
19  # Example usage:
20  print(subarrays_with_lcm_k([3, 6, 2, 7, 1], 6)) # Output: 4
21  print(subarrays_with_lcm_k([3], 2))           # Output: 0
22
```

Output:

```
Output
4
0
=== Code Execution Successful ===
```

3. Minimum Number of Operations to Sort a Binary Tree by Level You are given the root of a binary tree with unique values. In one operation, you can choose any two nodes at the same level and swap their values. Return the minimum number of operations needed to make the values at each level sorted in a strictly increasing order. The level of a node is the number of edges along the path between it and the root node. Example 1: Input: root = [1,4,3,7,6,8,5,null,null,null,null,9,null,10] Output: 3 Explanation: - Swap 4 and 3. The 2nd level becomes [3,4]. - Swap 7 and 5. The 3rd level becomes [5,6,8,7]. - Swap 8 and 7. The 3rd level becomes [5,6,7,8]. We used 3 operations so return 3. It can be proven that 3 is the minimum number of operations needed. Example 2: Input: root = [1,3,2,7,6,5,4] Output: 3 Explanation: - Swap 3 and 2. The 2nd level becomes [2,3]. - Swap 7 and 4. The 3rd level becomes [4,6,5,7]. - Swap 6 and 5. The 3rd level becomes [4,5,6,7]. We used 3 operations so return 3. It can be proven that 3 is the minimum number of operations needed. Example 3: Input: root = [1,2,3,4,5,6] Output: 0 Explanation: Each level is already sorted in increasing order so return 0. Constraints: • The number of nodes in the tree is in the range [1, 105]. • 1 <= Node.val <= 105 • All the values of the tree

Program:

File Edit Format Run Options Windows Help

```
from collections import deque
```

```
# Definition for a binary tree node.
```

```
class TreeNode:
```

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

```
def minSwaps(arr):
```

```
    n = len(arr)  
    # Create a list of tuples where each tuple is (value, original_index)  
    arrpos = list(enumerate(arr))  
    # Sort the array by the value  
    arrpos.sort(key=lambda it: it[1])  
    # To keep track of visited elements  
    visited = {i: False for i in range(n)}  
    # Initialize result  
    swaps = 0  
    for i in range(n):  
        # If the element is already visited or already in the correct position  
        if visited[i] or arrpos[i][0] == i:  
            continue  
        # Compute the size of the cycle  
        cycle_size = 0  
        x = i  
        while not visited[x]:  
            visited[x] = True  
            x = arrpos[x][0]  
            cycle_size += 1  
        # If there is more than one element in the cycle, add the number of swaps  
        if cycle_size > 1:  
            swaps += cycle_size - 1  
    return swaps
```

```
def minOperationsToSortByLevel(root):
```

```
    if not root:  
        return 0  
  
    queue = deque([root])  
    total_swaps = 0  
  
    while queue:  
        level_size = len(queue)  
        level_nodes = []  
        for _ in range(level_size):  
            node = queue.popleft()
```

```

        level_size = len(queue)
        level_nodes = []
        for _ in range(level_size):
            node = queue.popleft()
            level_nodes.append(node.val)
            if node.left:
                queue.append(node.left)
            if node.right:
                queue.append(node.right)

        # Calculate the minimum number of swaps to sort this
        total_swaps += minSwaps(level_nodes)

    return total_swaps

# Example usage:
root1 = TreeNode(1)
root1.left = TreeNode(4)
root1.right = TreeNode(3)
root1.left.left = TreeNode(7)
root1.left.right = TreeNode(6)
root1.right.left = TreeNode(8)
root1.right.right = TreeNode(5)
root1.left.left.left = TreeNode(9)
root1.right.right.left = TreeNode(10)

print(minOperationsToSortByLevel(root1)) # Output: 3

root2 = TreeNode(1)
root2.left = TreeNode(3)
root2.right = TreeNode(2)
root2.left.left = TreeNode(7)
root2.left.right = TreeNode(6)
root2.right.left = TreeNode(5)
root2.right.right = TreeNode(4)

print(minOperationsToSortByLevel(root2)) # Output: 3

root3 = TreeNode(1)
root3.left = TreeNode(2)
root3.right = TreeNode(3)
root3.left.left = TreeNode(4)
root3.left.right = TreeNode(5)
root3.right.left = TreeNode(6)

print(minOperationsToSortByLevel(root3)) # Output: 0

```

1. 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 Example 2: Input: ["FirstUnique", "showFirstUnique", "add", "add", "add", "add", "add", "showFirstUnique"] [[[7,7,7,7,7,7]], [], [7], [3], [3], [7], [17], []] Output: [null,-1,null,null,null,null,null,17] Explanation: FirstUnique firstUnique = new FirstUnique([7,7,7,7,7,7]); firstUnique.showFirstUnique(); // return -1 firstUnique.add(7); // the queue is now [7,7,7,7,7,7,7] firstUnique.add(3); // the queue is now [7,7,7,7,7,7,3] firstUnique.add(3); // the queue is now [7,7,7,7,7,7,3,3] firstUnique.add(7); // the queue is now [7,7,7,7,7,7,3,3,7] firstUnique.add(17); // the queue is now [7,7,7,7,7,7,3,3,7,17] firstUnique.showFirstUnique(); // return 17 Example 3: Input: ["FirstUnique", "showFirstUnique", "add", "showFirstUnique"] [[[809]], [], [809], []] Output: [null,809,null,-1] Explanation: FirstUnique firstUnique = new FirstUnique([809]); firstUnique.showFirstUnique(); // return 809 firstUnique.add(809); // the queue is now [809,809] firstUnique.showFirstUnique(); // return -1

Program:

```
76 *Untitled*
File Edit Format Run Options Windows Help
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):
        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):
        if value in self.count:
            self.count[value] += 1
        else:
            self.count[value] = 1
            self.queue.append(value)

# Example usage:
firstUnique = FirstUnique([2, 3, 5])
print(firstUnique.showFirstUnique()) # return 2
firstUnique.add(5) # the queue is now [2, 3, 5, 5]
print(firstUnique.showFirstUnique()) # return 2
firstUnique.add(2) # the queue is now [2, 3, 5, 5, 2]
print(firstUnique.showFirstUnique()) # return 3
firstUnique.add(3) # the queue is now [2, 3, 5, 5, 2, 3]
print(firstUnique.showFirstUnique()) # return -1

# More example usage:
firstUnique = FirstUnique([7, 7, 7, 7, 7, 7])
print(firstUnique.showFirstUnique()) # return -1
firstUnique.add(7) # the queue is now [7, 7, 7, 7, 7, 7, 7]
firstUnique.add(3)
```

Output:

```
Output
2
2
3
-1
-1
17

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

2. 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 Example 2: Input: root = [0,1,0,0,1,0,null,null,1,0,0], arr = [0,0,1] Output: false Explanation: The path 0 -> 0 -> 1 does not exist, therefore it is not even a sequence. Example 3: Input: root = [0,1,0,0,1,0,null,null,1,0,0], arr = [0,1,1] Output: false Explanation: The path 0 -> 1 -> 1 is a sequence, but it is not a valid sequence.

Program:

```
File Edit Format Run Options Windows Help
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 not node or index == len(arr):
            return False
        if 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.right.left.right = TreeNode(1)
root.right.right = TreeNode(0)
arr1 = [0, 1, 0, 1]
arr2 = [0, 0, 1]
arr3 = [0, 1, 1]
print(isValidSequence(root, arr1)) # Output: True
print(isValidSequence(root, arr2)) # Output: False
print(isValidSequence(root, arr3)) # Output: False
```


Output:

```
Output
False
False
True

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

3. 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. Example 2: Input: `candies = [4,2,1,1,2]`, `extraCandies = 1` Output: `[true,false,false,false,false]` Explanation: There is only 1 extra candy. Kid 1 will always have the greatest number of candies, even if a different kid is given the extra candy. Example 3: Input: `candies = [12,1,12]`, `extraCandies = 10` Output: `[true,false,true]`

Output:

```
Output

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

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

Program:

```
main.py [ ] [ ] Save Run

1 def kidsWithCandies(candies, extraCandies):
2     max_candies = max(candies)
3     result = []
4     for candy in candies:
5         result.append(candy + extraCandies >= max_candies)
6     return result
7
8 # Example usage:
9 candies1 = [2, 3, 5, 1, 3]
10 extraCandies1 = 3
11 print(kidsWithCandies(candies1, extraCandies1)) # Output: [True, True
    , True, False, True]
12
13 candies2 = [4, 2, 1, 1, 2]
14 extraCandies2 = 1
15 print(kidsWithCandies(candies2, extraCandies2)) # Output: [True,
    False, False, False, False]
16
17 candies3 = [12, 1, 12]
18 extraCandies3 = 10
19 print(kidsWithCandies(candies3, extraCandies3)) # Output: [True,
    False, True]
```

4. 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 Example 2: Input: num = 9 Output: 8 Explanation: The first time pick x = 9 and y = 9 and store the new integer in a. The second time pick x = 9 and y = 1 and store the new integer in b. We have now a = 9 and b = 1 and max difference = 8

Program:

```
main.py  [ ] [ ] Save Run

1 def maxDiff(num):
2     num_str = str(num)
3     a = b = num_str
4     for x in num_str:
5         if x != '9':
6             a = num_str.replace(x, '9')
7             break
8     if a == num_str:
9         for x in num_str:
10            if x != '1' and x != '0':
11                a = num_str.replace(x, '1')
12                break
13    b = num_str.replace(num_str[0], '1') if num_str[0] != '1' else
        num_str.replace(num_str[0], '0')
14
15    return int(a) - int(b)
16 num1 = 555
17 print(maxDiff(num1)) # Output: 888
18
19 num2 = 9
20 print(maxDiff(num2)) # Output: 8
21
```



Output:

```
Output
888
0

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

5. 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". Example 2: Input: s1 = "abe", s2 = "acd" Output: false Explanation: All permutations for s1="abe" are: "abe", "aeb", "bae", "bea", "eab" and "eba" and all permutation for s2="acd" are: "acd", "adc", "cad", "cda", "dac" and "dca". However, there is not any permutation from s1 which can break some permutation from s2 and vice-versa. Example 3: Input: s1 = "leetcodee", s2 = "interview"

Program:

```
main.py   Save Run

1 def canBreak(s1, s2):
2     s1_sorted = sorted(s1)
3     s2_sorted = sorted(s2)
4     s1_breaks_s2 = True
5     for i in range(len(s1)):
6         if s1_sorted[i] < s2_sorted[i]:
7             s1_breaks_s2 = False
8             break
9     s2_breaks_s1 = True
10    for i in range(len(s2)):
11        if s2_sorted[i] < s1_sorted[i]:
12            s2_breaks_s1 = False
13            break
14    return s1_breaks_s2 or s2_breaks_s1
15 s1_1, s2_1 = "abc", "xya"
16 print(canBreak(s1_1, s2_1)) # Output: True
17
18 s1_2, s2_2 = "abe", "acd"
19 print(canBreak(s1_2, s2_2)) # Output: False
20
21 s1_3, s2_3 = "leetcodee", "interview"
22 print(canBreak(s1_3, s2_3)) # Output: True
23
```

Output:

```
Output
True
False
True

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

6. 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 i th 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. Example 2: Input: `hats = [[3,5,1],[3,5]]` Output: 4 Explanation: There are 4 ways to choose hats: (3,5), (5,3), (1,3) and (1,5) Example 3: Input: `hats = [[1,2,3,4],[1,2,3,4],[1,2,3,4],[1,2,3,4]]` Output: 24 Explanation: Each person can choose hats labeled from 1 to 4. Number of Permutations of (1,2,3,4) = 24.

Program:

```
main.py
1 def numberWays(hats):
2     MOD = 10**9 + 7
3     n = len(hats)
4     MAX_MASK = (1 << 40)
5     dp = [0] * MAX_MASK
6     dp[0] = 1 # Base case: No one has chosen any hat yet
7     person_hats = [set(h) for h in hats]
8     for h in person_hats:
9         for mask in range(MAX_MASK - 1, -1, -1):
10             for hat in h:
11                 if (mask & (1 << hat)) == 0: # If the hat is not
                    taken
12                     new_mask = mask | (1 << hat)
13                     dp[new_mask] = (dp[new_mask] + dp[mask]) % MOD
14     return dp[MAX_MASK - 1]
15 hats1 = [[3, 4], [4, 5], [5]]
16 print(numberWays(hats1)) # Output: 1
17 hats2 = [[3, 5, 1], [3, 5]]
18 print(numberWays(hats2)) # Output: 4
19 hats3 = [[1, 2, 3, 4], [1, 2, 3, 4], [1, 2, 3, 4], [1, 2, 3, 4]]
20 print(numberWays(hats3)) # Output: 24
21
```

7. 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. Example 1: Input: nums = [1,2,3] Output: [1,3,2] Example 2: Input: nums = [3,2,1] Output: [1,2,3] Example 3: Input: nums = [1,1,5] Output: [1,5,1] Constraints:

Program:

```
def nextPermutation(nums):
    k = len(nums) - 2
    while k >= 0 and nums[k] >= nums[k + 1]:
        k -= 1
    if k == -1:
        nums.reverse()
        return nums
    l = len(nums) - 1
    while l > k and nums[l] <= nums[k]:
        l -= 1
    nums[k], nums[l] = nums[l], nums[k]
    nums[k + 1:] = nums[k + 1:][::-1]

    return nums

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

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

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

Output:

```
Output
[1, 3, 2]
[1, 2, 3]
[1, 5, 1]

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