Assignment(8)(19/6/24)

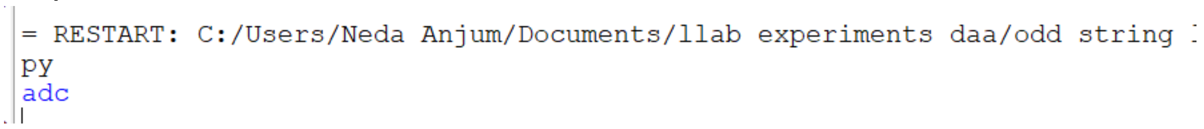
1. Odd String Difference You are given an array of equal-length strings words. Assume that the length of each string is n.Each string words[i] can be converted into a difference integer array difference[i] of length n- 1 where difference[i][j] = words[i][j+1]- words[i][j] where 0 <= j <= n- 2. Note that the difference between two letters is the difference between their positions in the alphabet i.e. the position of 'a' is 0, 'b' is 1, and 'z' is 25. For example, for the string "acb", the difference integer array is [2- 0, 1- 2] = [2,-1].All the strings in words have the same difference integer array, except one. You should find that string. Return the string in words that has different difference integer array.

Example 1: Input: words = ["adc","wzy","abc"]

Output: "abc"
Code:

```
def find_odd_string(words):
    def get_difference_array(word):
        return [ord(word[i+1]) - ord(word[i]) for i in range(len(word) - 1)]
    difference_arrays = [get_difference_array(word) for word in words]
    diff_count = {}
    for i, diff in enumerate(difference_arrays):
        diff_tuple = tuple(diff)
        if diff_tuple in diff_count:
            diff_count[diff_tuple].append(i)
        else:
            diff_count[diff_tuple] = [i]
            for diff, indices in diff_count.items():
                if len(indices) == 1:
                    return words[indices[0]]
words = ["adc", "wzy", "abc"]
print(find_odd_string(words))
```

output:



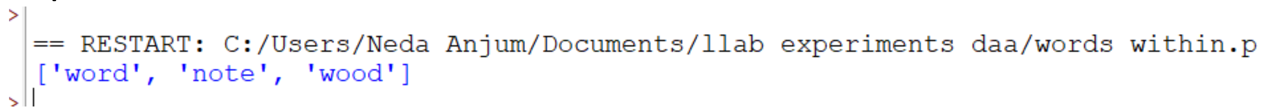
```
= RESTART: C:/Users/Neda Anjum/Documents/llab experiments daa/odd string
py
adc
```

2.Words Within Two Edits of Dictionary You are given two string arrays, queries and dictionary. All words in each array comprise of lowercase English letters and have the same length.In one edit you can take a word from queries, and change any letter in it to any other letter. Find all words from queries that, after a maximum of two edits, equal some word from dictionary. Return a list of all words from queries, that match with some word from dictionary after a maximum of two edits. Return the words in the same order they appear in queries.

Example 1: Input: queries = ["word","note","ants","wood"], dictionary = ["wood","joke","moat"] Output: ["word","note","wood"]
Code:

```
def words_within_two_edits(queries, dictionary):
   def within_two_edits(word1, word2):
      if len(word1) != len(word2):
         return False
      edits = sum(1 for a, b in zip(word1, word2) if a != b)
      return edits <= 2

   result = []
   for query in queries:
      if any(within_two_edits(query, dict_word) for dict_word in dictionary):
         result.append(query)

   return result
queries = ["word", "note", "ants", "wood"]
dictionary = ["wood", "joke", "moat"]
print(words_within_two_edits(queries, dictionary))
```

output:



```
== RESTART: C:/Users/Neda Anjum/Documents/llab experiments daa/words within.p
['word', 'note', 'wood']
```

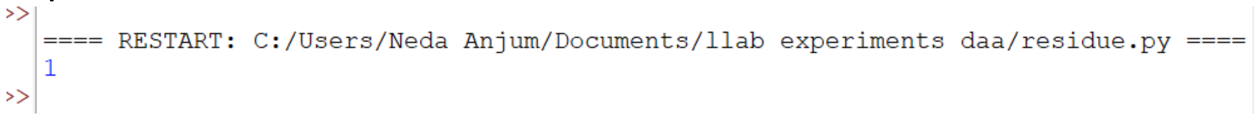
3.Destroy Sequential Targets You are given a 0-indexed array nums consisting of positive integers, representing targets on a number line. You are also given an integer space. You have a machine which can destroy targets. Seeding the machine with some nums[i] allows it to destroy all targets with values that can be represented as nums[i] + c * space, where c is any non-negative integer. You want to destroy the maximum number of targets in nums. Return the minimum value of nums[i] you can seed the machine with to destroy the maximum number of targets.

Example 1: Input: nums = [3,7,8,1,1,5], space = 2
Output: 1
Explanation: If we seed the machine with nums[3], then we destroy all targets equal to 1,3,5,7,9,... In this case, we would destroy 5 total targets (all except for nums[2])
Code:

```
def destroy_sequential_targets(nums, space):
   from collections import defaultdict
   residue_count = defaultdict(int)
   num_to_residue = {}

   for num in nums:
      residue = num % space
      residue_count[residue] += 1
      if residue not in num_to_residue:
         num_to_residue[residue] = num
      else:
         num_to_residue[residue] = min(num_to_residue[residue], num)
```

```
    max_targets = max(residue_count.values())
    min_seed_value = float('inf')

    for residue, count in residue_count.items():
        if count == max_targets:
            min_seed_value = min(min_seed_value, num_to_residue[residue])

    return min_seed_value
nums = [3, 7, 8, 1, 1, 5]
space = 2
print(destroy_sequential_targets(nums, space))
```

output:



```
>>
==== RESTART: C:/Users/Neda Anjum/Documents/llab experiments daa/residue.py ====
1
>>
```

4. Next Greater Element IV You are given a 0-indexed array of non-negative integers nums. For each integer in nums, you must find its respective second greater integer. The second greater integer of nums[i] is nums[j] such that: j > i

● nums[j] > nums[i] There exists exactly one index k such that nums[k] > nums[i] and i < k < j. If there is no such nums[j], the second greater integer is considered to be-1.

● For example, in the array [1, 2, 4, 3], the second greater integer of 1 is 4, 2 is 3, and that of 3 and 4 is-1. Return an integer array answer, where answer[i] is the second greater integer of nums[i].

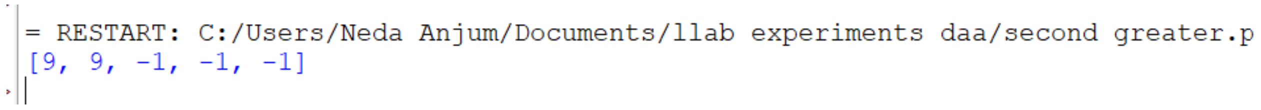
Example 1: Input: nums = [2,4,0,9,6]

Output: [9,6,6,-1,-1]

Code:

```
def second_greater(nums):
    n = len(nums)
    first_greater = [-1] * n
    second_greater = [-1] * n
    stack1 = []
    stack2 = []
    for i in range(n - 1, -1, -1):
        while stack1 and nums[stack1[-1]] <= nums[i]:
            stack1.pop()
        if stack1:
            first_greater[i] = stack1[-1]
        stack1.append(i)
    for i in range(n - 1, -1, -1):
        while stack2 and nums[stack2[-1]] <= nums[i]:
            stack2.pop()
        if stack2:
            second_greater[i] = nums[stack2[-1]]
            if first_greater[i] != -1:
                stack2.append(first_greater[i])

        return second_greater
    nums = [2, 4, 0, 9, 6]
    print(second_greater(nums))
```

output:



```
= RESTART: C:/Users/Neda Anjum/Documents/llab experiments daa/second greater.p
[9, 9, -1, -1, -1]
```

5..Average Value of Even Numbers That Are Divisible by Three Given an integer array nums of positive integers, return the average value of all even integers that are divisible by 3. Note that the average of n elements is the sum of the n elements divided by n and rounded down to the nearest integer.

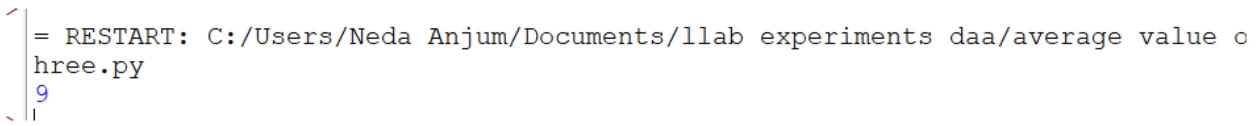
Example 1: Input: nums = [1,3,6,10,12,15]

Output: 9 Explanation: 6 and 12 are even numbers that are divisible by 3. (6 + 12) / 2 = 9

Code:

```
def average_value_of_even_divisible_by_three(nums):

    even_divisible_by_three = [num for num in nums if num % 6 == 0]

     if not even_divisible_by_three:

        return 0

      total_sum = sum(even_divisible_by_three)

    count = len(even_divisible_by_three)

    return total_sum // count

nums = [1, 3, 6, 10, 12, 15]

print(average_value_of_even_divisible_by_three(nums))
```

output:



```
= RESTART: C:/Users/Neda Anjum/Documents/llab experiments daa/average value o
hree.py
9
```

6. Most Popular Video Creator You are given two string arrays creators and ids, and an integer array views, all of length n. The ith video on a platform was created by creator[i], has an id of ids[i], and has views[i] views. The popularity of a creator is the sum of the number of views on all of the creator's videos. Find the creator with the highest popularity and the id of their most viewed video. ● Ifmultiple creators have the highest popularity, find all of them.

● Ifmultiple videos have the highest view count for a creator, find the lexicographically smallest id. Return a 2D array of strings answer where answer[i] = [creatori, idi] means that creatori has the highest popularity and idi is the id of their most popular video. The answer can be returned in any order.

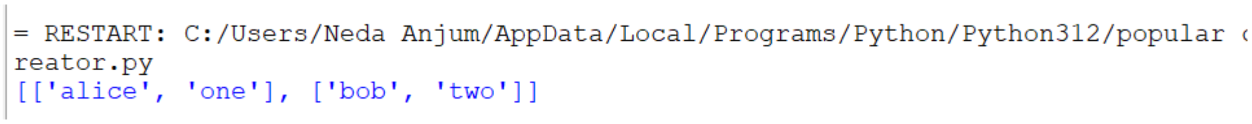
Example 1: Input: creators = ["alice","bob","alice","chris"], ids = ["one","two","three","four"], views = [5,10,5,4]

Output: [["alice","one"],["bob","two"]]

Code:

```
def mostPopularCreator(creators, ids, views):
    from collections import defaultdict
    creator_views = defaultdict(int)
    creator_top_video = {}

    for creator, id, view in zip(creators, ids, views):
        creator_views[creator] += view

        if creator not in creator_top_video:
            creator_top_video[creator] = (view, id)
        else:
            if view > creator_top_video[creator][0]:
                creator_top_video[creator] = (view, id)
            elif view == creator_top_video[creator][0] and id < creator_top_video[creator][1]:
                creator_top_video[creator] = (view, id)
    max_views = max(creator_views.values())
    result = []
    for creator in creator_views:
        if creator_views[creator] == max_views:
            result.append([creator, creator_top_video[creator][1]])
    return result

creators = ["alice", "bob", "alice", "chris"]

ids = ["one", "two", "three", "four"]

views = [5, 10, 5, 4]


print(mostPopularCreator(creators, ids, views))
```

output:



```
= RESTART: C:/Users/Neda Anjum/AppData/Local/Programs/Python/Python312/popular c
reator.py
[['alice', 'one'], ['bob', 'two']]
```

7. Minimum Addition to Make Integer Beautiful You are given two positive integers n and target. An integer is considered beautiful if the sum of its digits is less than or equal to target. Return the minimum non-negative integer x such that n + x is beautiful. The input will be generated such that it is always possible to make n beautiful.

Example 1: Input: n = 16, target = 6

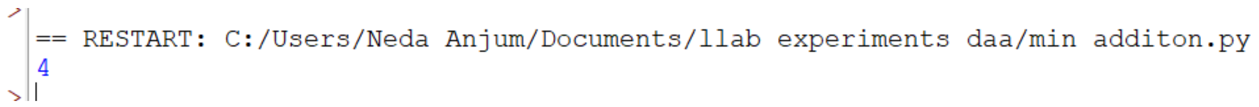
Output: 4 Explanation: Initially n is 16 and its digit sum is 1 + 6 = 7. After adding 4, n becomes 20 and digit sum becomes 2 + 0 = 2. It can be shown that we can not make n beautiful with adding non-negative integer less than 4.

Code:

```
def minAdditionToMakeBeautiful(n, target):

    def digit_sum(num):

        return sum(int(digit) for digit in str(num))


    x = 0

    while digit_sum(n + x) > target:

        increment = 10 ** len(str(x))

        x += increment - (n + x) % increment


    return x

n = 16

target = 6
```

```
print(minAdditionToMakeBeautiful(n, target))
```

output:



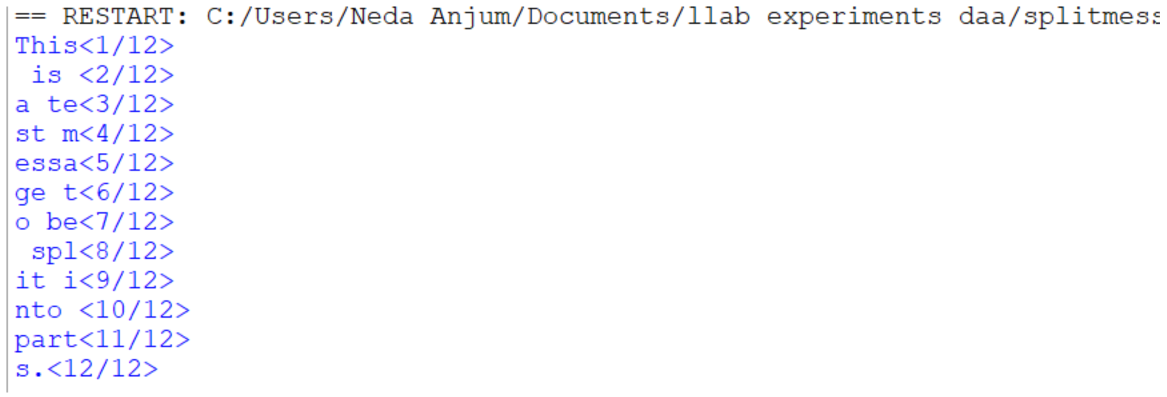
```
== RESTART: C:/Users/Neda Anjum/Documents/llab experiments daa/min additon.py
4
```

8. Split Message Based on Limit You are given a string, message, and a positive integer, limit. You must split message into one or more parts based on limit. Each resulting part should have the suffix "", where "b" is to be replaced with the total number of parts and "a" is to be replaced with the index of the part, starting from 1 and going up to b. Additionally, the length of each resulting part (including its suffix) should be equal to limit, except for the last part whose length can be at most limit. The resulting parts should be formed such that when their suffixes are removed and they are all concatenated in order, they should be equal to message. Also, the result should contain as few parts as possible

Code:

```
def splitMessage(message, limit):

    def suffix_length(parts):
        return len(f"<1/{parts}>")

    n = len(message)
    parts = 1
    while True:
        suffix_len = suffix_length(parts)
        if parts * (limit - suffix_len) >= n:
            break
        parts += 1

    result = []
    suffix_len = suffix_length(parts)
    i = 0
    for part in range(1, parts + 1):
        part_limit = limit - suffix_len

        part_message = message[i:i + part_limit]

        result.append(f"{part_message}<{part}/{parts}>")

        i += part_limit


    return result

message = "This is a test message to be split into parts."

limit = 10

result = splitMessage(message, limit)

for part in result:

    print(part)
```

output:



```
== RESTART: C:/Users/Neda Anjum/Documents/llab experiments daa/splitmess
This<1/12>
 is <2/12>
a te<3/12>
st m<4/12>
essa<5/12>
ge t<6/12>
o be<7/12>
 spl<8/12>
it i<9/12>
nto <10/12>
part<11/12>
s.<12/12>
```