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"

Explanation: - The difference integer array of "adc" is [3 - 0, 2 - 3] = [3, -1]. - The difference integer array of "wzy" is [25 - 22, 24 - 25]= [3, -1]. - The difference integer array of "abc" is [1 - 0, 2 - 1] = [1, 1]. The odd array out is [1, 1], so we return the corresponding string, "abc".

main.py

Save Run

Output

```
def oddString(words):
    def get_diff_array(word):
        return [ord(word[i+1]) - ord(word[i]) for i in range(len(word) - 1)]

    # Create a dictionary to count the occurrences of each difference array
    diff_counts = {}
    for word in words:
        diff_array = tuple(get_diff_array(word))
        if diff_array in diff_counts:
            diff_counts[diff_array].append(word)
        else:
            diff_counts[diff_array] = [word]

    # Find the difference array that has only one word associated with it
    for diff_array, word_list in diff_counts.items():
        if len(word_list) == 1:
            return word_list[0]

# Example usage:
words = ["adc", "wzy", "abc"]
print(oddString(words))  # Output: "abc"

```

```
abc

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

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"]

Explanation: - Changing the 'r' in "word" to 'o' allows it to equal the dictionary word "wood". - Changing the 'n' to 'j' and the 't' to 'k' in "note" changes it to "joke". - It would take more than 2 edits for "ants" to

equal a dictionary word. - "wood" can remain unchanged (0 edits) and match the corresponding dictionary word. Thus, we return ["word","note","wood"].

```python
def within_two_edits(queries, dictionary):
    # Helper function to calculate the edit distance between two words
    def edit_distance(word1, word2):
        # Since all words have the same length, we only count the differing positions
        return sum(1 for a, b in zip(word1, word2) if a != b)

    result = []
    for query in queries:
        for dict_word in dictionary:
            if edit_distance(query, dict_word) <= 2:
                result.append(query)
                break  # No need to check other dictionary words for this query

    return result

# Example usage:
queries = ["word", "note", "ants", "wood"]
dictionary = ["wood", "joke", "moat"]
print(within_two_edits(queries, dictionary))  # Output: ["word", "note", "wood"]
```

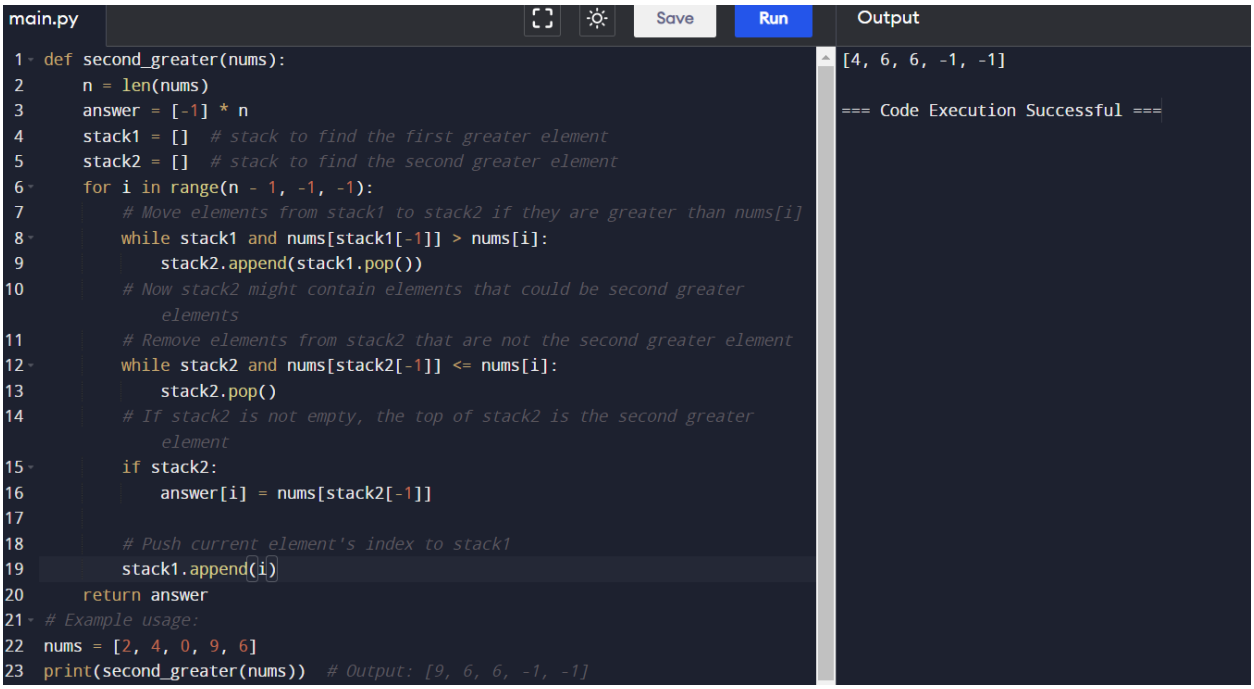
```
['word', 'note', 'wood']

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

3. 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]

Explanation: 0th index: 4 is the first integer greater than 2, and 9 is the second integer greater than 2, to the right of 2. 1st index: 9 is the first, and 6 is the second integer greater than 4, to the right of 4. 2nd index: 9 is the first, and 6 is the second integer greater than 0, to the right of 0. 3rd index: There is no integer greater than 9 to its right, so the second greater integer is considered to be -1. 4th index: There is no integer greater than 6 to its right, so the second greater integer is considered to be -1. Thus, we return [9,6,6,-1,-1].

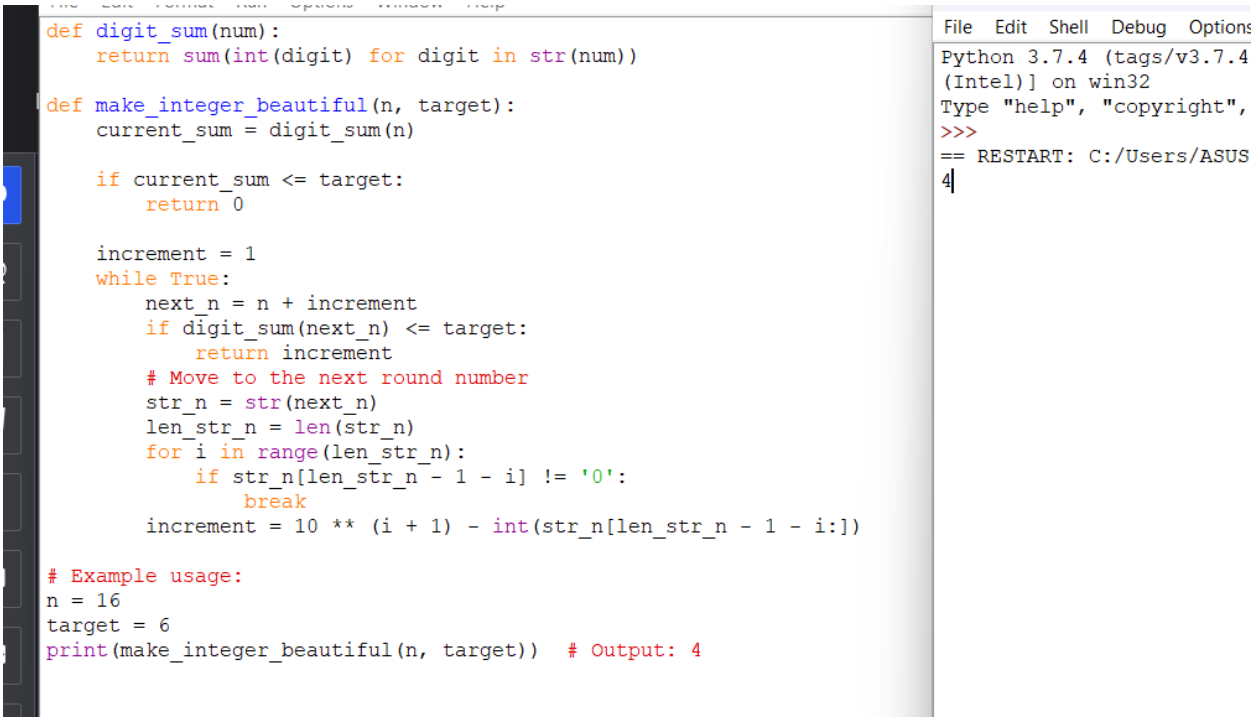


4. Minimum Addition to Make Integer Beautiful You are given two positive integers n and target. An integer is considered beautiful if the sum of its digits is less than or equal to target. Return the minimum non-negative integer x such that n + x is beautiful. The input will be generated such that it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



```python
def digit_sum(num):
    return sum(int(digit) for digit in str(num))

def make_integer_beautiful(n, target):
    current_sum = digit_sum(n)

    if current_sum <= target:
        return 0

    increment = 1
    while True:
        next_n = n + increment
        if digit_sum(next_n) <= target:
            return increment
        # Move to the next round number
        str_n = str(next_n)
        len_str_n = len(str_n)
        for i in range(len_str_n):
            if str_n[len_str_n - 1 - i] != '0':
                break
        increment = 10 ** (i + 1) - int(str_n[len_str_n - 1 - i:])

# Example usage:
n = 16
target = 6
print(make_integer_beautiful(n, target))  # Output: 4
```

```
File  Edit  Shell  Debug  Options
Python 3.7.4 (tags/v3.7.4
(Intel)] on win32
Type "help", "copyright",
>>>
== RESTART: C:/Users/ASUS
4
```

5. Sort Array by Moving Items to Empty Space You are given an integer array nums of size n containing each element from 0 to n - 1 (inclusive). Each of the elements from 1 to n - 1 represents an item, and the element 0 represents an empty space. In one operation, you can move any item to the empty space. nums is considered to be sorted if the numbers of all the items are in ascending order and the empty space is either at the beginning or at the end of the array. For example, if n = 4, nums is sorted if: ● nums = [0,1,2,3] or ● nums = [1,2,3,0] ...and considered to be unsorted otherwise. Return the minimum number of operations needed to sort nums.

Example 1: Input: nums = [4,2,0,3,1]

Output: 3

Explanation: - Move item 2 to the empty space. Now, nums = [4,0,2,3,1]. - Move item 1 to the empty space. Now, nums = [4,1,2,3,0]. - Move item 4 to the empty space. Now, nums = [0,1,2,3,4]. It can be proven that 3 is the minimum number of operations needed.

```python
def min_operations_to_sort(nums):
    n = len(nums)
    pos = {num: i for i, num in enumerate(nums)}  # Position of each number in
        the array
    empty_pos = pos[0]  # Position of the empty space (0)
    moves = 0

    for i in range(n):
        while nums[i] != i:
            # Move the item currently in the correct place to the empty spot
            nums[empty_pos], nums[pos[i]] = nums[pos[i]], nums[empty_pos]
            pos[nums[empty_pos]], pos[nums[pos[i]]] = empty_pos, pos[i]
            empty_pos = pos[0]  # Update the position of the empty space (0)
            moves += 1

    return moves

# Example usage:
nums = [4, 2, 0, 3, 1]
print(min_operations_to_sort(nums))  # Output: 3
```

6.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]). It is impossible to destroy more than 5 targets, so we return nums[3].

```python
def min_seeding_value(nums, space):
    nums.sort()
    max_targets = 0
    min_value = float('inf')

    for num in nums:
        targets = sum((num + c * space in nums) for c in range(1001))
        if targets > max_targets or (targets == max_targets and num < min_value)
            max_targets = targets
            min_value = num

    return min_value

# Example
nums = [3, 7, 8, 1, 1, 5]
space = 2
print(min_seeding_value(nums, space))  # Output: 1
```

```
File  Edit  Shell  Debug
Python 3.7.4 (tags/
(Intel)] on win32
Type "help", "copyr
>>>
= RESTART: C:/Users

= RESTART: C:/Users
1
>>>
```

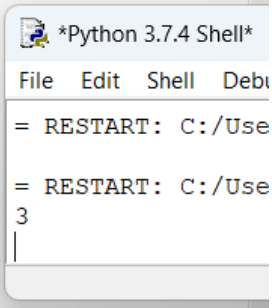
7. . Apply Operations to an Array You are given a 0-indexed array nums of size n consisting of non-negative integers. You need to apply n - 1 operations to this array where, in the ith operation (0-indexed), you will apply the following on the ith element of nums: ● If nums[i] == nums[i + 1], then multiply nums[i] by 2 and set nums[i + 1] to 0. Otherwise, you skip this operation. After performing all the operations, shift all the 0's to the end of the array. ● For example, the array [1,0,2,0,0,1] after shifting all its 0's to the end, is [1,2,1,0,0,0]. Return the resulting array. Note that the operations are applied sequentially, not all at once.

Example 1: Input: nums = [1,2,2,1,1,0]

Output: [1,4,2,0,0,0]

Explanation: We do the following operations: - i = 0: nums[0] and nums[1] are not equal, so we skip this operation. - i = 1: nums[1] and nums[2] are equal, we multiply nums[1] by 2 and change nums[2] to 0. The array becomes [1,4,0,1,1,0]. - i = 2: nums[2] and nums[3] are not equal, so we skip this operation. - i = 3: nums[3] and nums[4] are equal, we multiply nums[3] by 2 and change nums[4] to 0. The array becomes [1,4,0,2,0,0]. - i = 4: nums[4] and nums[5] are equal, we multiply nums[4] by 2 and change nums[5] to 0. The array becomes [1,4,0,2,0,0]. After that, we shift the 0's to the end, which gives the array [1,4,2,0,0,0].

```python
def min_operations_to_sort(nums):
    n = len(nums)
    moves = 0

    def swap(i, j):
        nums[i], nums[j] = nums[j], nums[i]

    # Find cycles and break them by swapping elements to the empty space (0)
    for i in range(n):
        while nums[i] != i:
            empty_pos = nums.index(0)
            if empty_pos == i:
                for j in range(i + 1, n):
                    if nums[j] != j:
                        swap(empty_pos, j)
                        moves += 1
                        break
            swap(i, empty_pos)
            moves += 1

    return moves

# Example usage:
nums = [4, 2, 0, 3, 1]
print(min_operations_to_sort(nums))  # Output: 3
```



Ln: 24 Col: 22