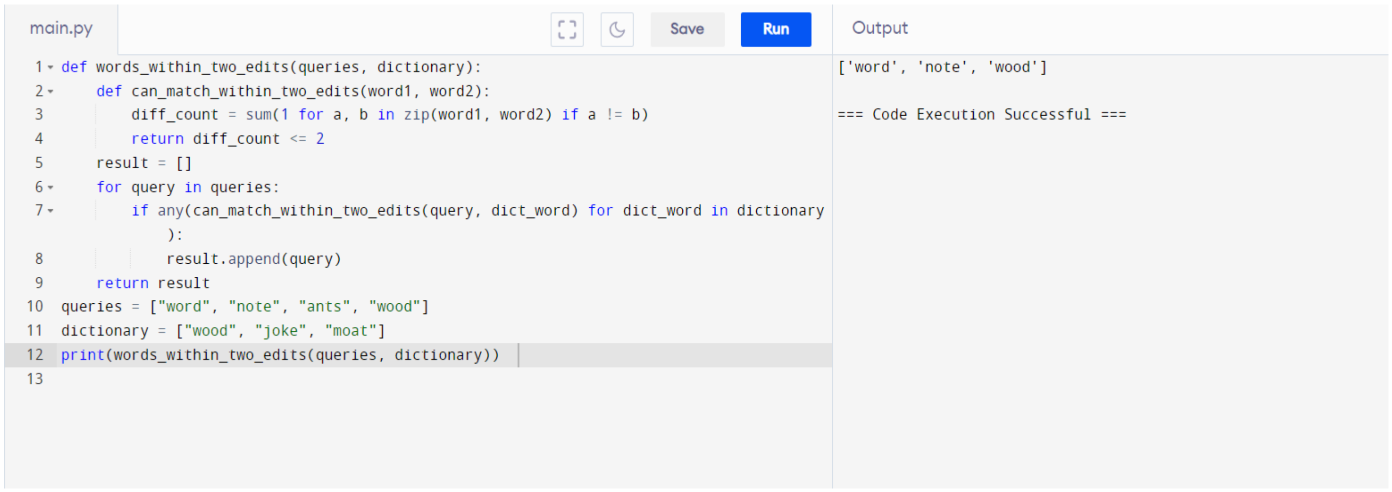
ASSIGNMENT-4

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.

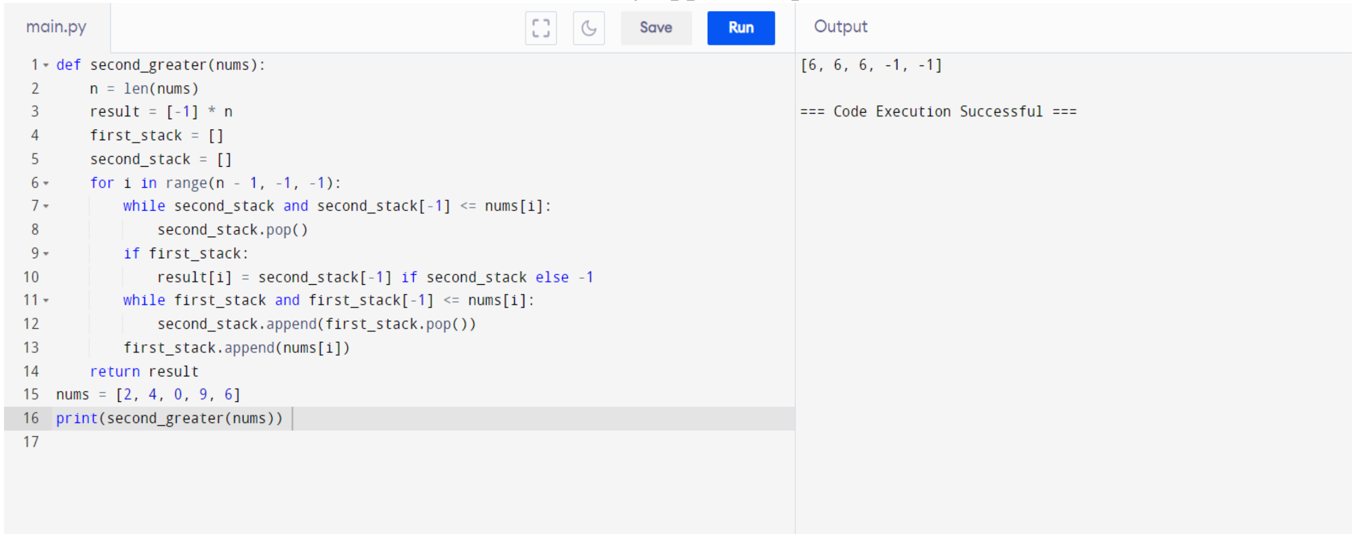


```python
def words_within_two_edits(queries, dictionary):
    def can_match_within_two_edits(word1, word2):
        diff_count = sum(1 for a, b in zip(word1, word2) if a != b)
        return diff_count <= 2
    result = []
    for query in queries:
        if any(can_match_within_two_edits(query, dict_word) 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))
```

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

=== 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.



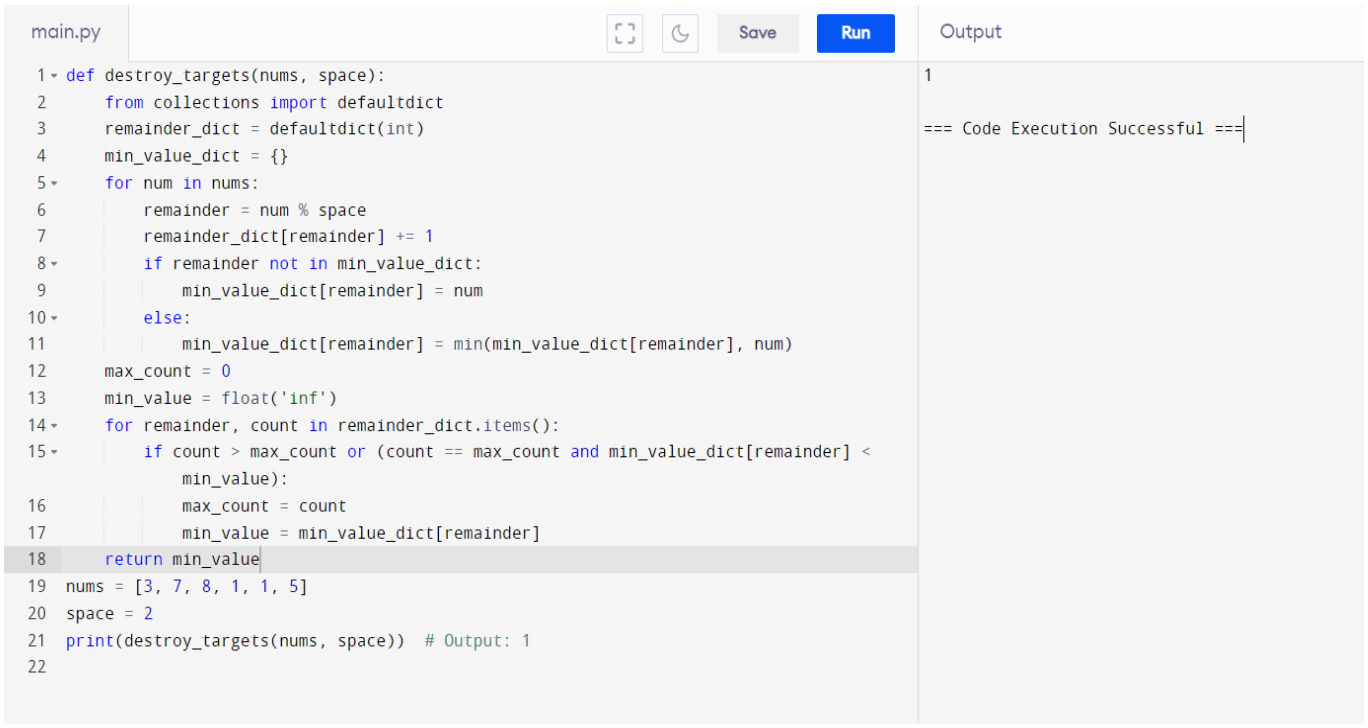
```python
def second_greater(nums):
    n = len(nums)
    result = [-1] * n
    first_stack = []
    second_stack = []
    for i in range(n - 1, -1, -1):
        while second_stack and second_stack[-1] <= nums[i]:
            second_stack.pop()
        if first_stack:
            result[i] = second_stack[-1] if second_stack else -1
        while first_stack and first_stack[-1] <= nums[i]:
            second_stack.append(first_stack.pop())
        first_stack.append(nums[i])
    return result
nums = [2, 4, 0, 9, 6]
print(second_greater(nums))
```

```
[6, 6, 6, -1, -1]

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

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.

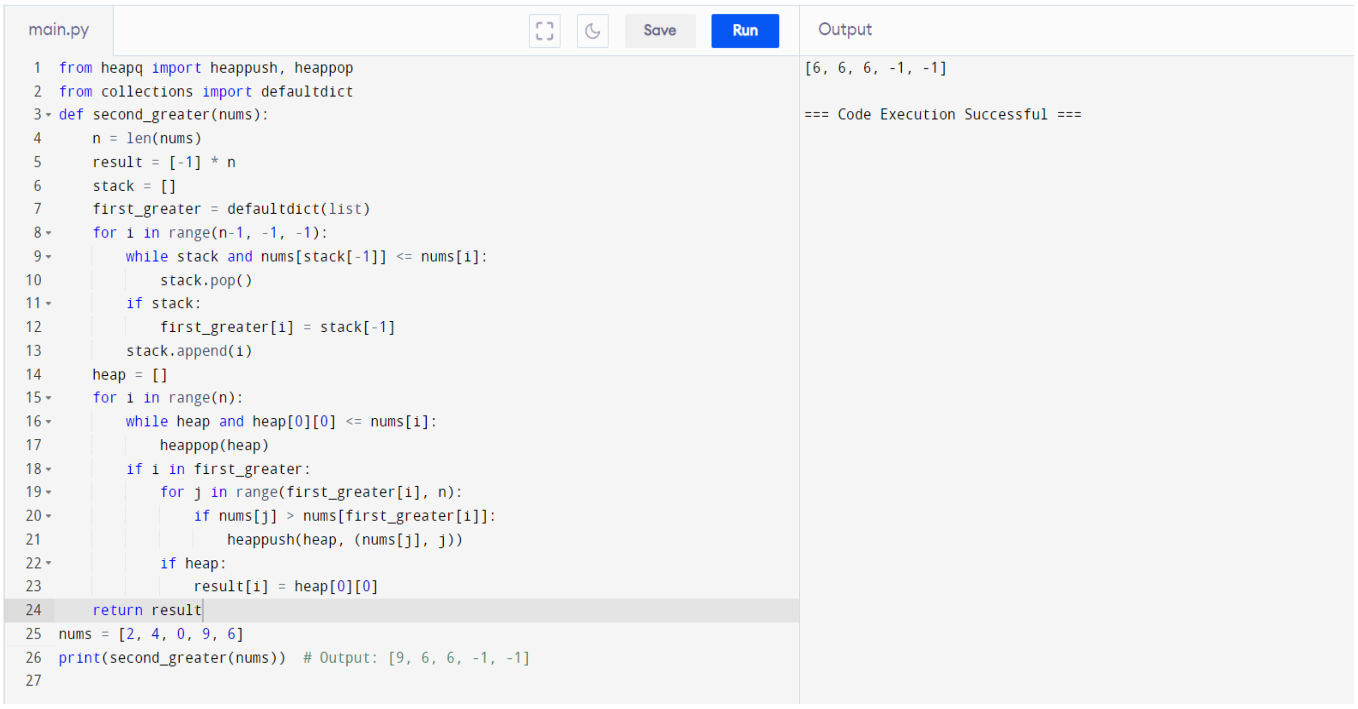
You have a machine which can destroy targets. Seeding the machine with some nums[i] allows it to destroy all targets with values that can be represented as nums[i] + c * space, where c is any non-negative integer. You want to destroy the maximum number of targets in nums.
Return the minimum value of nums[i] you can seed the machine with to destroy the maximum number of targets.



```python
def destroy_targets(nums, space):
    from collections import defaultdict
    remainder_dict = defaultdict(int)
    min_value_dict = {}
    for num in nums:
        remainder = num % space
        remainder_dict[remainder] += 1
        if remainder not in min_value_dict:
            min_value_dict[remainder] = num
        else:
            min_value_dict[remainder] = min(min_value_dict[remainder], num)
    max_count = 0
    min_value = float('inf')
    for remainder, count in remainder_dict.items():
        if count > max_count or (count == max_count and min_value_dict[remainder] <
            min_value):
            max_count = count
            min_value = min_value_dict[remainder]
    return min_value
nums = [3, 7, 8, 1, 1, 5]
space = 2
print(destroy_targets(nums, space))  # Output: 1
```

```
1

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

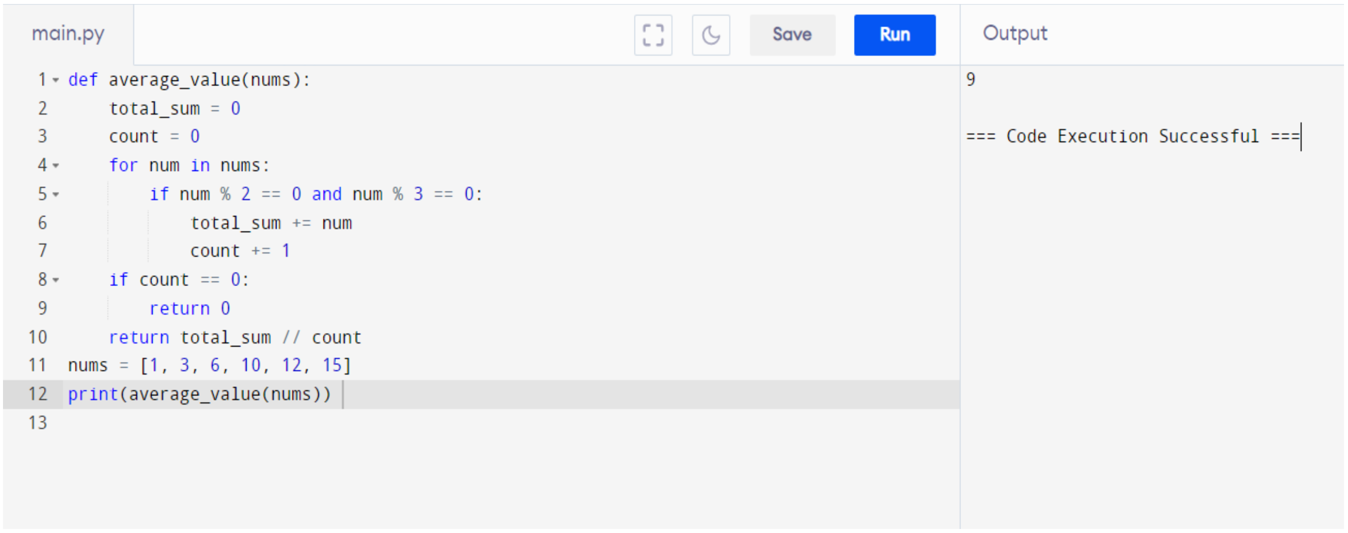
4. **Next Greater Element IV**
   You are given a 0-indexed array of non-negative integers nums. For each integer in nums, you must find its respective second greater integer. The second greater integer of nums[i] is nums[j] such that: j > i
   nums[j] > nums[i]
   There exists exactly one index k such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



```python
from heapq import heappush, heappop
from collections import defaultdict
def second_greater(nums):
    n = len(nums)
    result = [-1] * n
    stack = []
    first_greater = defaultdict(list)
    for i in range(n-1, -1, -1):
        while stack and nums[stack[-1]] <= nums[i]:
            stack.pop()
        if stack:
            first_greater[i] = stack[-1]
        stack.append(i)
    heap = []
    for i in range(n):
        while heap and heap[0][0] <= nums[i]:
            heappop(heap)
        if i in first_greater:
            for j in range(first_greater[i], n):
                if nums[j] > nums[first_greater[i]]:
                    heappush(heap, (nums[j], j))
            if heap:
                result[i] = heap[0][0]
    return result
nums = [2, 4, 0, 9, 6]
print(second_greater(nums))  # Output: [9, 6, 6, -1, -1]
```

```
[6, 6, 6, -1, -1]

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

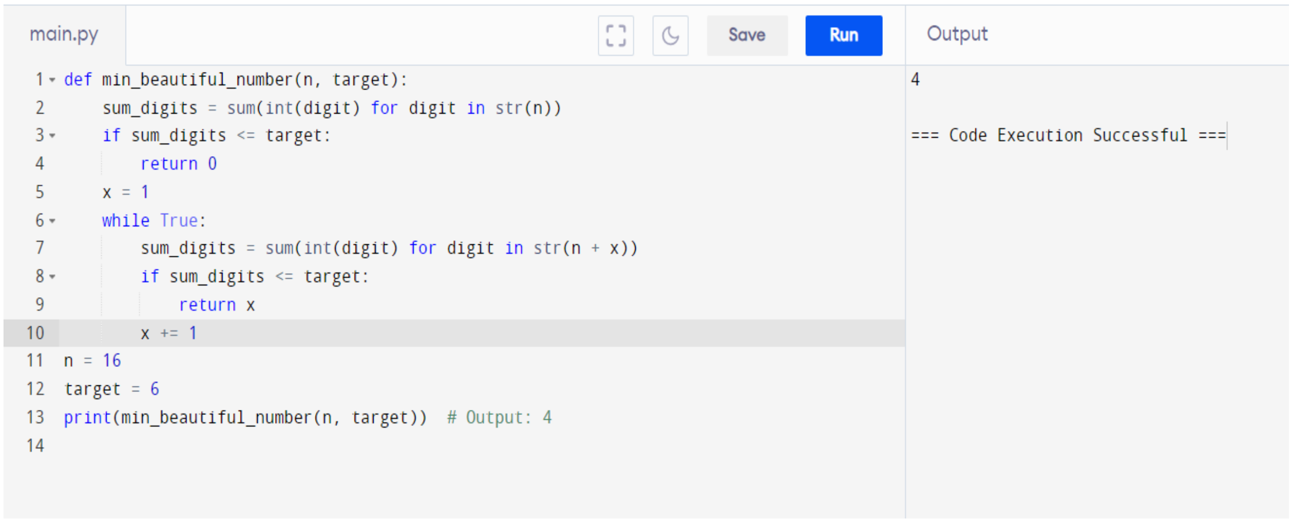
5. **Average Value of Even Numbers That Are Divisible by Three**
   Given an integer array nums of positive integers, return the average value of all even integers that are divisible by 3.
   Note that the average of n elements is the sum of the n elements divided by n and rounded down to the nearest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



```python
def average_value(nums):
    total_sum = 0
    count = 0
    for num in nums:
        if num % 2 == 0 and num % 3 == 0:
            total_sum += num
            count += 1
    if count == 0:
        return 0
    return total_sum // count
nums = [1, 3, 6, 10, 12, 15]
print(average_value(nums))
```

```
9

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

6. **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.

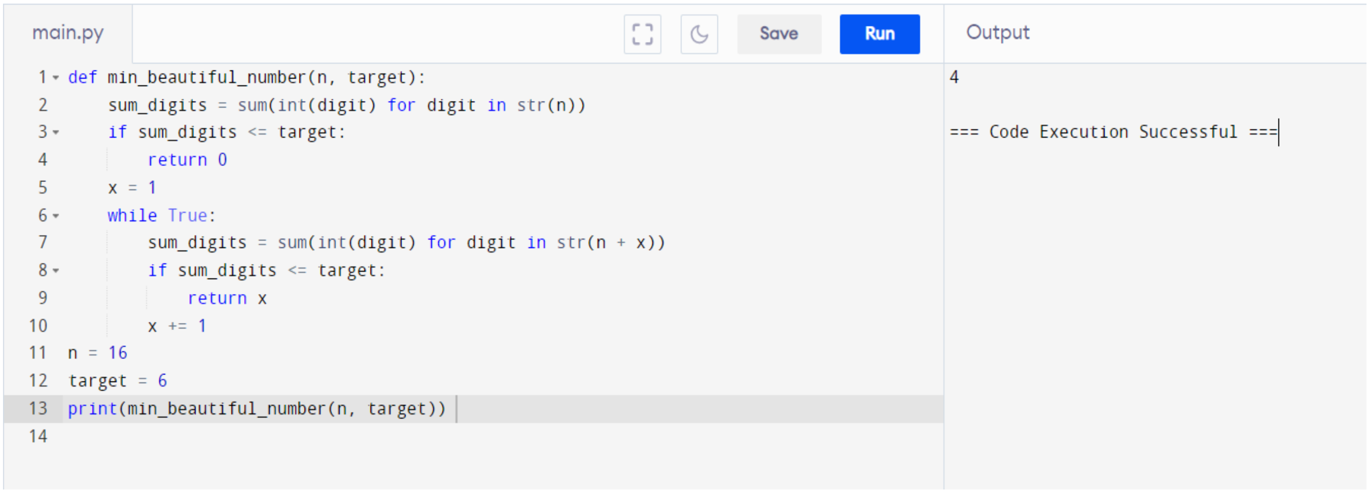


```python
def min_beautiful_number(n, target):
    sum_digits = sum(int(digit) for digit in str(n))
    if sum_digits <= target:
        return 0
    x = 1
    while True:
        sum_digits = sum(int(digit) for digit in str(n + x))
        if sum_digits <= target:
            return x
        x += 1
n = 16
target = 6
print(min_beautiful_number(n, target))  # Output: 4

```

```
4

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

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.



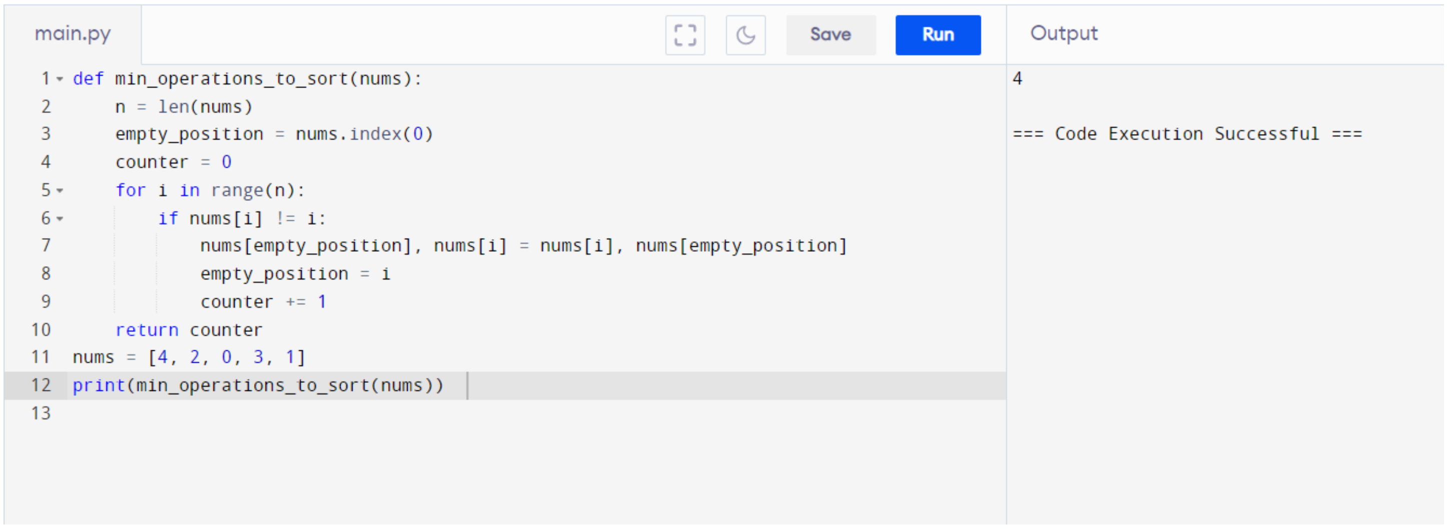
```python
def min_beautiful_number(n, target):
    sum_digits = sum(int(digit) for digit in str(n))
    if sum_digits <= target:
        return 0
    x = 1
    while True:
        sum_digits = sum(int(digit) for digit in str(n + x))
        if sum_digits <= target:
            return x
        x += 1
n = 16
target = 6
print(min_beautiful_number(n, target))

```

```
4

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

8. **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.



```python
def min_operations_to_sort(nums):
    n = len(nums)
    empty_position = nums.index(0)
    counter = 0
    for i in range(n):
        if nums[i] != i:
            nums[empty_position], nums[i] = nums[i], nums[empty_position]
            empty_position = i
            counter += 1
    return counter
nums = [4, 2, 0, 3, 1]
print(min_operations_to_sort(nums))
```

```
4

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

9. **Apply Operations to an Array**

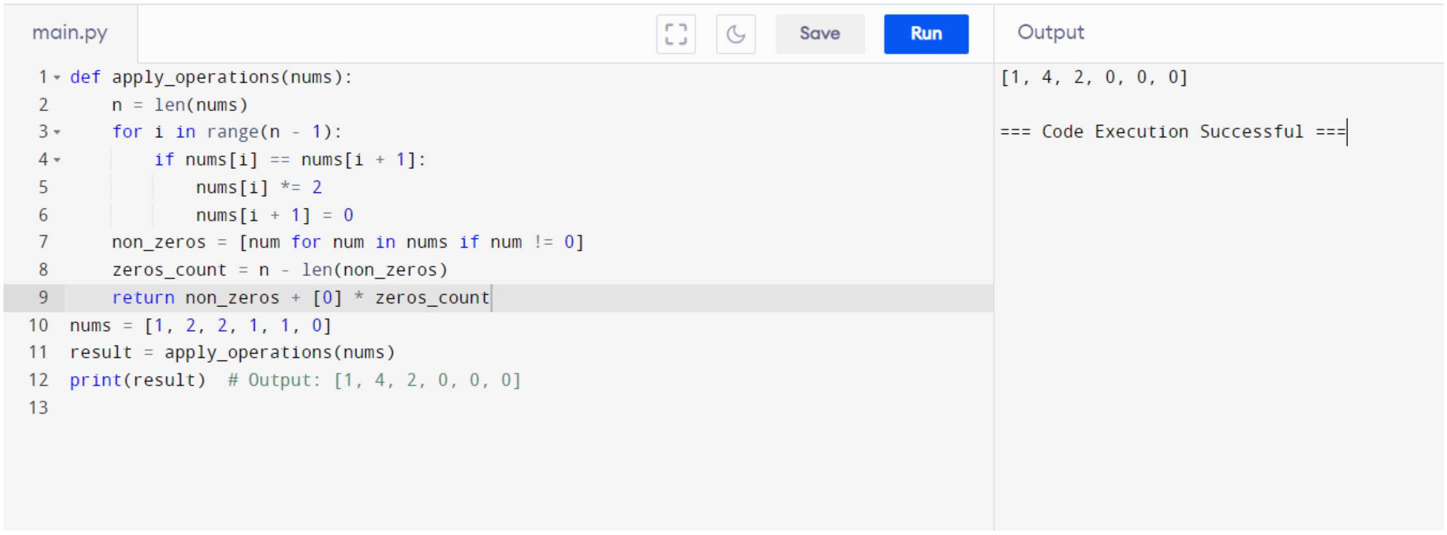
You are given a 0-indexed array nums of size n consisting of non-negative integers.

You need to apply n - 1 operations to this array where, in the ith operation (0indexed), 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.



```python
def apply_operations(nums):
    n = len(nums)
    for i in range(n - 1):
        if nums[i] == nums[i + 1]:
            nums[i] *= 2
            nums[i + 1] = 0
    non_zeros = [num for num in nums if num != 0]
    zeros_count = n - len(non_zeros)
    return non_zeros + [0] * zeros_count
nums = [1, 2, 2, 1, 1, 0]
result = apply_operations(nums)
print(result)  # Output: [1, 4, 2, 0, 0, 0]
```

```
[1, 4, 2, 0, 0, 0]

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