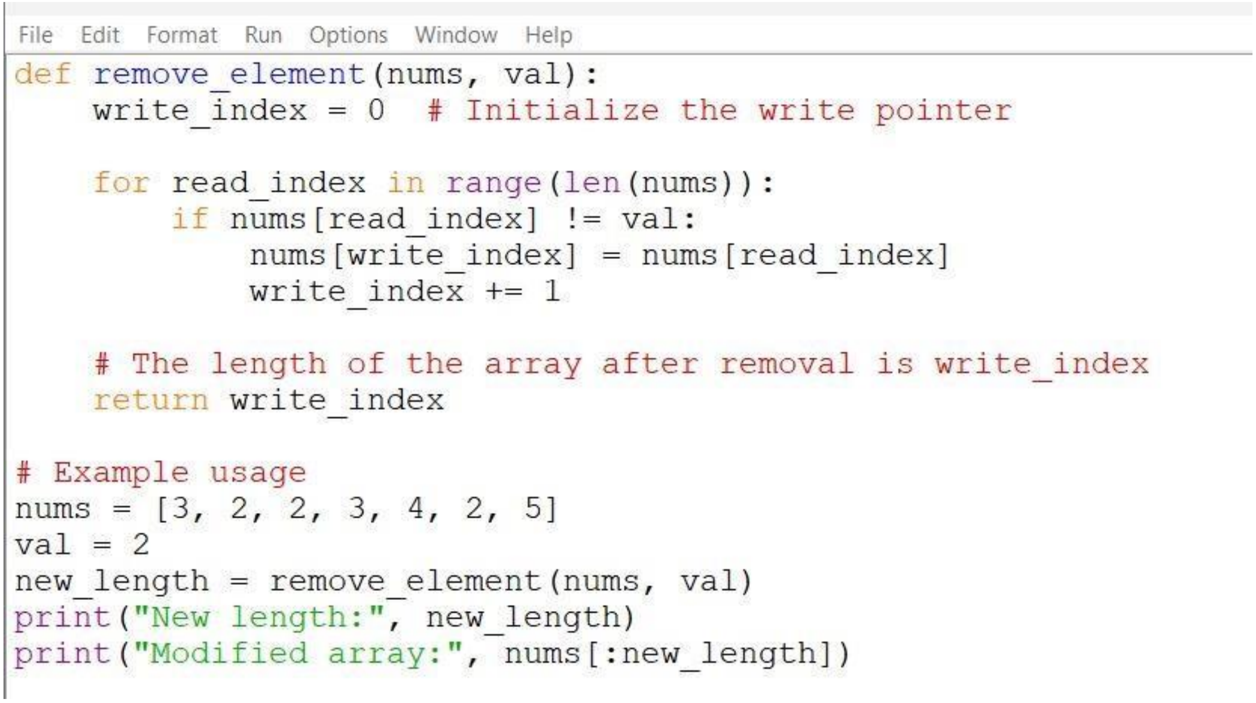
1. Remove Element

Given an integer array nums and an integer val, remove all occurrences of val in nums inplace. The relative order of the elements may be changed.

Since it is impossible to change the length of the array in some languages, you must instead have the result be placed in the first part of thenums . More formally, if there are k array
elements after removing the duplicates, then the elements of k nums should hold the final result. It does not matter what you leave bey nums . ements first

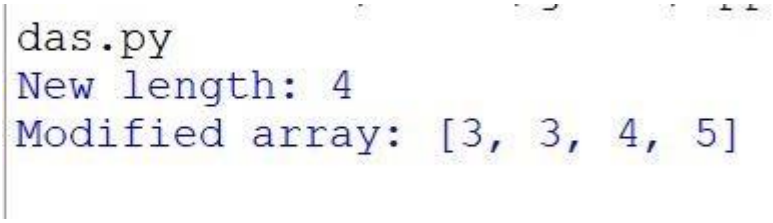
Return k *after placing the final result in the first* k *slots of*

Do not allocate extra space for another array. You input
array in-place with O(1) extra memory.



```python
def remove_element(nums, val):
    write_index = 0  # Initialize the write pointer

    for read_index in range(len(nums)):
        if nums[read_index] != val:
            nums[write_index] = nums[read_index]
            write_index += 1

    # The length of the array after removal is write_index
    return write_index

# Example usage
nums = [3, 2, 2, 3, 4, 2, 5]
val = 2
new_length = remove_element(nums, val)
print("New length:", new_length)
print("Modified array:", nums[:new_length])
```

OUTPUT:



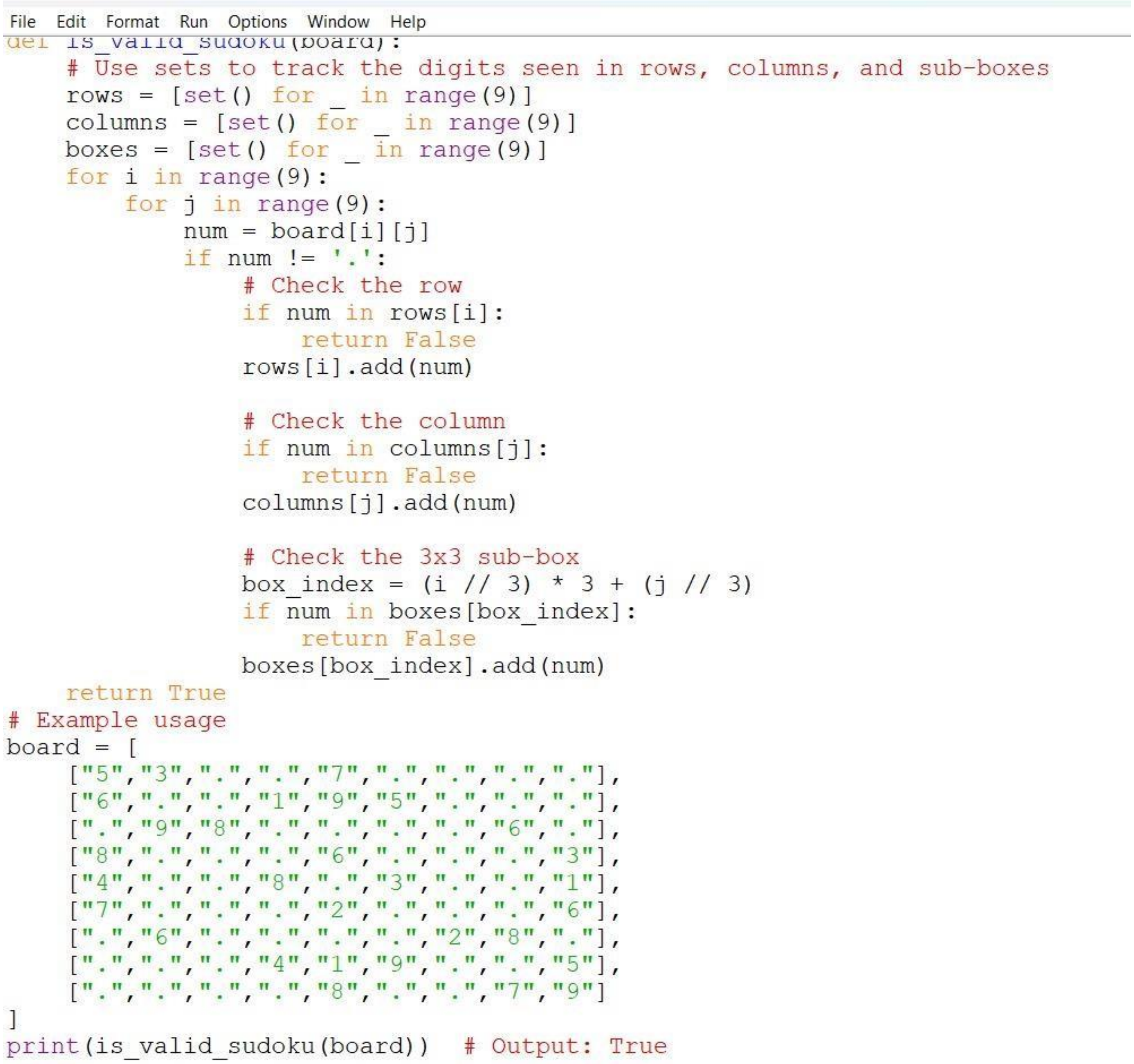
```
das.py
New length: 4
Modified array: [3, 3, 4, 5]
```

Determine if a 9 x 9 Sudoku board is valid. Only the filled cells need to be validated according to the following rules:

1. **Each row must contain the digits 1-9 without repetition.**
2. **Each column must contain the digits 1-9 without repetition.**
3. **Each of the nine 3 x 3 sub-boxes of the grid must contain the digits 1-9 without repetition.**

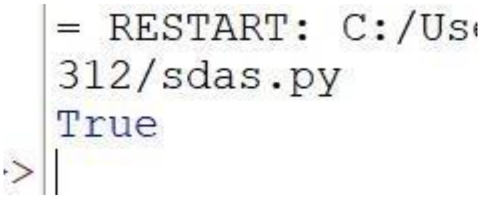
Note:

- **A Sudoku board (partially filled) could be valid but is not necessarily solvable.**
- **Only the filled cells need to be validated according to the mentioned rules.**



```
def is_valid_sudoku(board):
    # Use sets to track the digits seen in rows, columns, and sub-boxes
    rows = [set() for _ in range(9)]
    columns = [set() for _ in range(9)]
    boxes = [set() for _ in range(9)]
    for i in range(9):
        for j in range(9):
            num = board[i][j]
            if num != '.':
                # Check the row
                if num in rows[i]:
                    return False
                rows[i].add(num)

                # Check the column
                if num in columns[j]:
                    return False
                columns[j].add(num)

                # Check the 3x3 sub-box
                box_index = (i // 3) * 3 + (j // 3)
                if num in boxes[box_index]:
                    return False
                boxes[box_index].add(num)
    return True
# Example usage
board = [
    ["5","3",".",".","7",".",".",".","."],
    ["6",".",".","1","9","5",".",".","."],
    [".","9","8",".",".",".",".","6","."],
    ["8",".",".",".","6",".",".",".","3"],
    ["4",".",".","8",".","3",".",".","1"],
    ["7",".",".",".","2",".",".",".","6"],
    [".","6",".",".",".",".","2","8","."],
    [".",".",".","4","1","9",".",".","5"],
    [".",".",".",".","8",".",".","7","9"]
]
print(is_valid_sudoku(board))  # Output: True
```

-
- **OUTPUT:**

- 

```
= RESTART: C:/Us
312/sdas.py
True
>
```

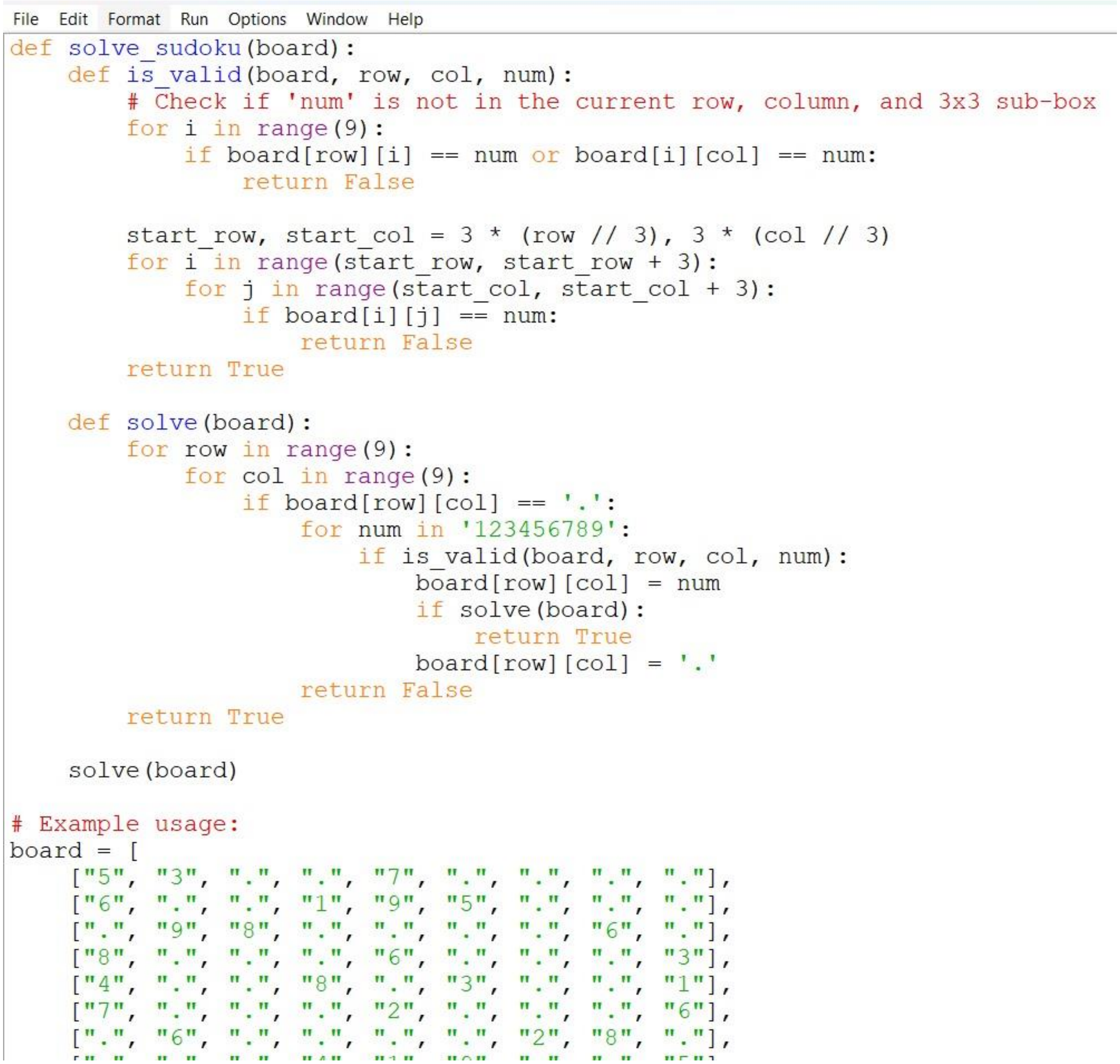
37. Sudoku Solver

Write a program to solve a Sudoku puzzle by filling the empty cells.

A sudoku solution must satisfy all of the following rules:

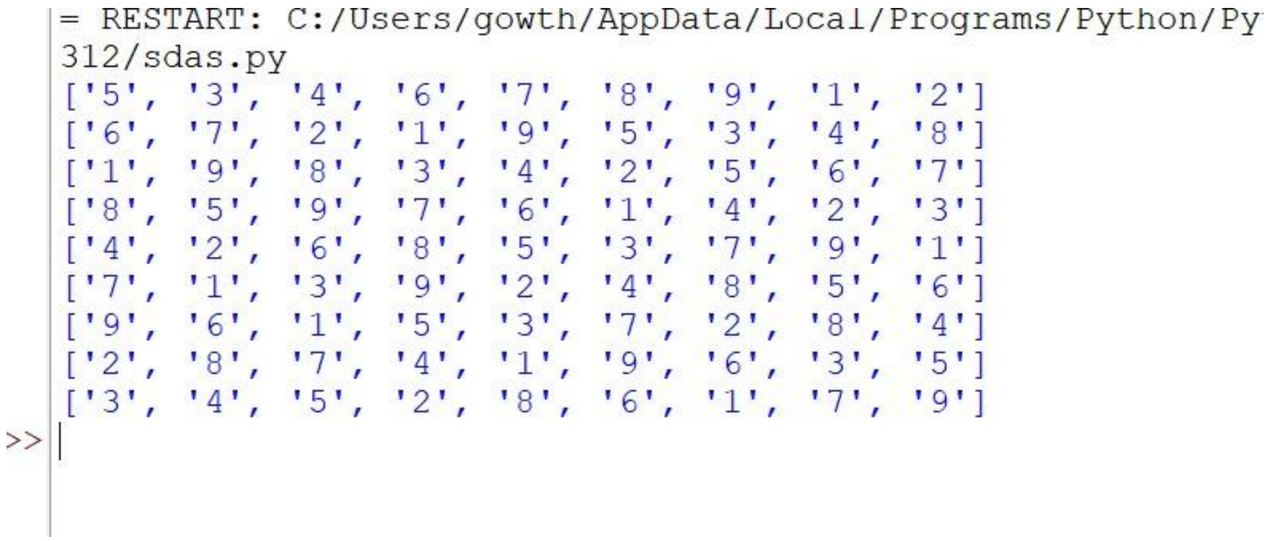
1. **Each of the digits 1-9 must occur exactly once in each row.**
2. **Each of the digits 1-9 must occur exactly once in each column.**
3. **Each of the digits 1-9 must occur exactly once in each of the 9 3x3 sub-boxes of the grid.**

The '.' character indicates empty cells.



```
def solve_sudoku(board):
    def is_valid(board, row, col, num):
        # Check if 'num' is not in the current row, column, and 3x3 sub-box
        for i in range(9):
            if board[row][i] == num or board[i][col] == num:
                return False

        start_row, start_col = 3 * (row // 3), 3 * (col // 3)
        for i in range(start_row, start_row + 3):
            for j in range(start_col, start_col + 3):
                if board[i][j] == num:
                    return False
        return True

    def solve(board):
        for row in range(9):
            for col in range(9):
                if board[row][col] == '.':
                    for num in '123456789':
                        if is_valid(board, row, col, num):
                            board[row][col] = num
                            if solve(board):
                                return True
                            board[row][col] = '.'
                    return False
        return True

    solve(board)

# Example usage:
board = [
    ["5", "3", ".", ".", "7", ".", ".", ".", "."],
    ["6", ".", ".", "1", "9", "5", ".", ".", "."],
    [".", "9", "8", ".", ".", ".", ".", "6", "."],
    ["8", ".", ".", ".", "6", ".", ".", ".", "3"],
    ["4", ".", ".", "8", ".", "3", ".", ".", "1"],
    ["7", ".", ".", ".", "2", ".", ".", ".", "6"],
    [".", "6", ".", ".", ".", ".", "2", "8", "."],
    [".", ".", ".", "4", "1", "9", ".", ".", "5"],
```

Output:



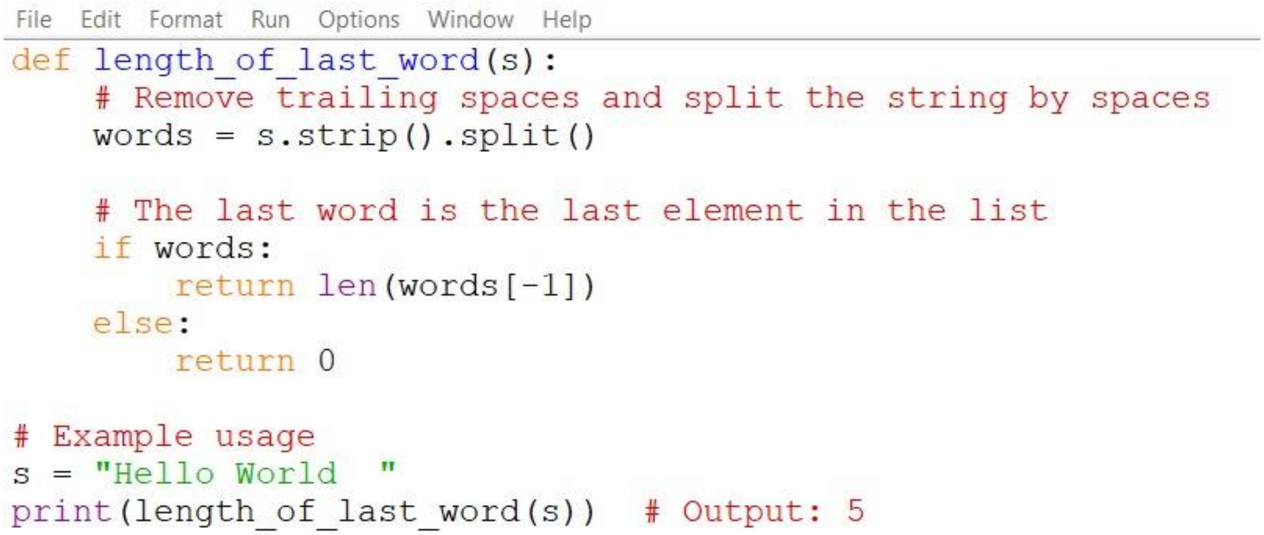
```
= RESTART: C:/Users/gowth/AppData/Local/Programs/Python/Py
312/sdas.py
['5', '3', '4', '6', '7', '8', '9', '1', '2']
['6', '7', '2', '1', '9', '5', '3', '4', '8']
['1', '9', '8', '3', '4', '2', '5', '6', '7']
['8', '5', '9', '7', '6', '1', '4', '2', '3']
['4', '2', '6', '8', '5', '3', '7', '9', '1']
['7', '1', '3', '9', '2', '4', '8', '5', '6']
['9', '6', '1', '5', '3', '7', '2', '8', '4']
['2', '8', '7', '4', '1', '9', '6', '3', '5']
['3', '4', '5', '2', '8', '6', '1', '7', '9']
>>
```

OUTPUT:

3.Count and Say

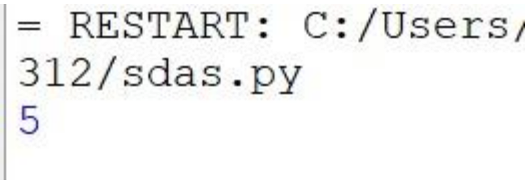
The count-and-say sequence is a sequence of digit strings defined by the recursive formula:

- **countAndSay(1) = "1"**
- **countAndSay(n) is the way you would "say" the digit string from countAndSay(n1), which is then converted into a different digit string.**



```
def length_of_last_word(s):
    # Remove trailing spaces and split the string by spaces
    words = s.strip().split()

    # The last word is the last element in the list
    if words:
        return len(words[-1])
    else:
        return 0

# Example usage
s = "Hello World  "
print(length_of_last_word(s))  # Output: 5
```

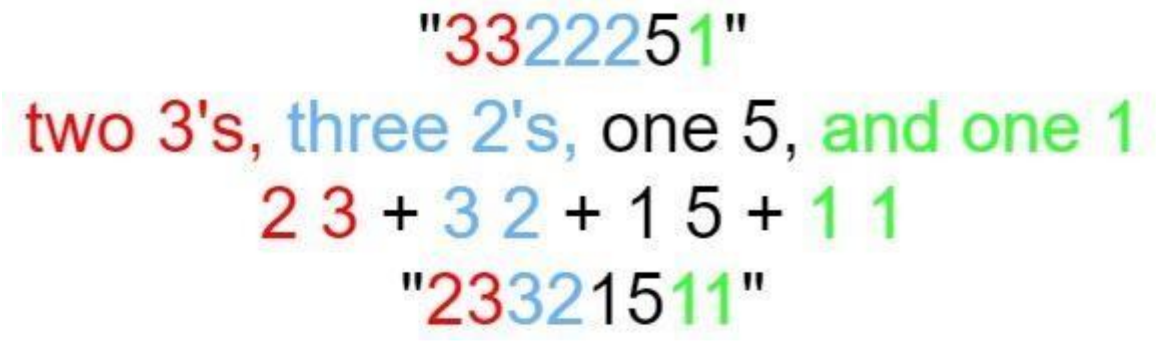
-
- **Output:**

- 

```
= RESTART: C:/Users/
312/sdas.py
5
```

To determine how you "say" a digit string, split it into the minimal number of substrings such that each substring contains exactly one unique digit. Then for each substring, say the number of digits, then say the digit. Finally, concatenate every said digit.

For example, the saying and conversion for digit string `"3322251"`:

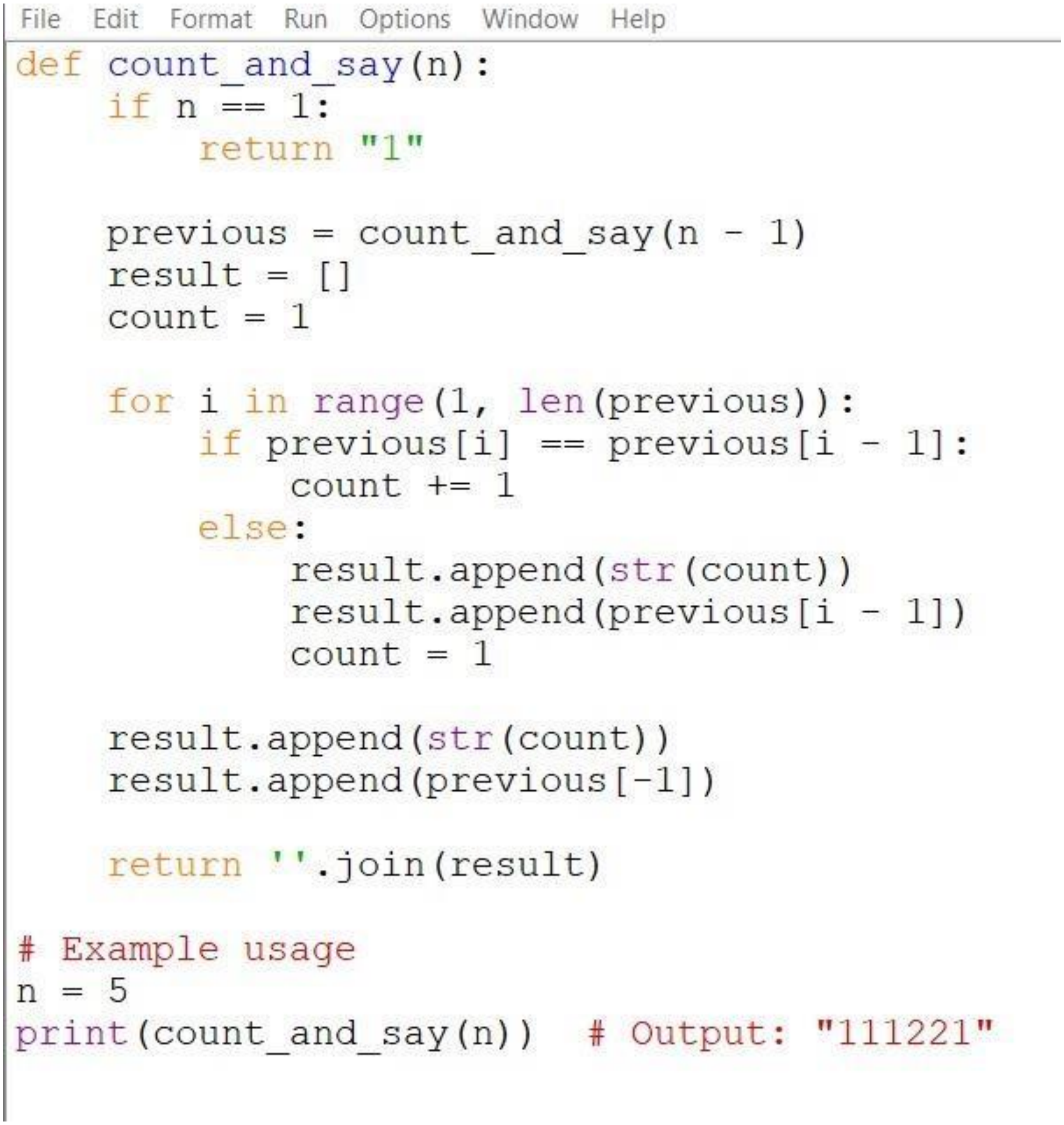


"3322251"

two 3's, three 2's, one 5, and one 1

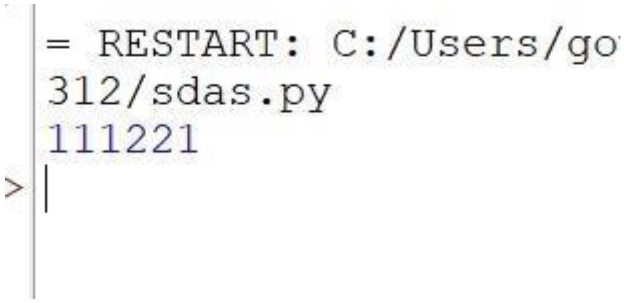
2 3 + 3 2 + 1 5 + 1 1

"23321511"



```
def count_and_say(n):
    if n == 1:
        return "1"

    previous = count_and_say(n - 1)
    result = []
    count = 1

    for i in range(1, len(previous)):
        if previous[i] == previous[i - 1]:
            count += 1
        else:
            result.append(str(count))
            result.append(previous[i - 1])
            count = 1

    result.append(str(count))
    result.append(previous[-1])

    return ''.join(result)

# Example usage
n = 5
print(count_and_say(n))  # Output: "111221"
```

OUTPUT:

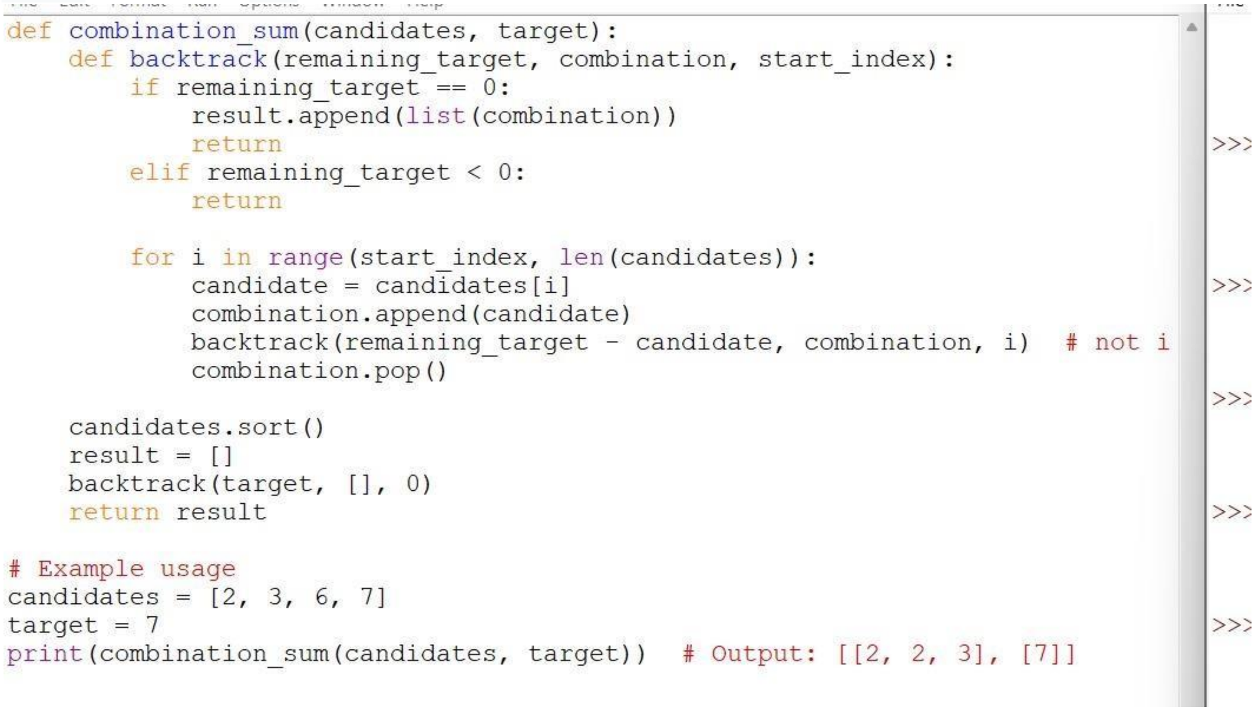


```
= RESTART: C:/Users/go
312/sdas.py
111221
>
```

39. Combination Sum

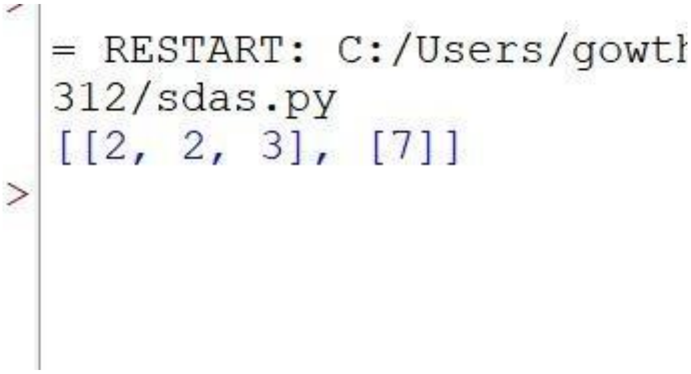
Given an array of distinct integers candidates and a target integer target, return *a list of all unique combinations of* candidates *where the chosen numbers sum to* target. You may return the combinations in any order.

The same number may be chosen from candidates an unlimited number of times. Two combinations are unique if the frequency of at least one of the chosen numbers is different. The test cases are generated such that the number of unique combinations that sum up to target is less than 150 combinations for the given input.



```python
def combination_sum(candidates, target):
    def backtrack(remaining_target, combination, start_index):
        if remaining_target == 0:
            result.append(list(combination))
            return
        elif remaining_target < 0:
            return

        for i in range(start_index, len(candidates)):
            candidate = candidates[i]
            combination.append(candidate)
            backtrack(remaining_target - candidate, combination, i)  # not i
            combination.pop()

    candidates.sort()
    result = []
    backtrack(target, [], 0)
    return result

# Example usage
candidates = [2, 3, 6, 7]
target = 7
print(combination_sum(candidates, target))  # Output: [[2, 2, 3], [7]]
```

OUTPUT:



```
= RESTART: C:/Users/gowtl
312/sdas.py
[[2, 2, 3], [7]]
>
```

40. Combination Sum II

Given a collection of candidate numbers (candidates) and a target number (target), find all unique combinations in candidates where the candidate numbers sum to target.

Each number in candidates may only be used once in the combination.

Note: The solution set must not contain duplicate combinations.

Example 1:

Input: candidates = [10,1,2,7,6,1,5], target = 8

Output:

[

[1,1,6],

[1,2,5],

[1,7],

[2,6]

]

Example 2:

Input: candidates = [2,5,2,1,2], target = 5 Output:

[

[1,2,2],

[5]

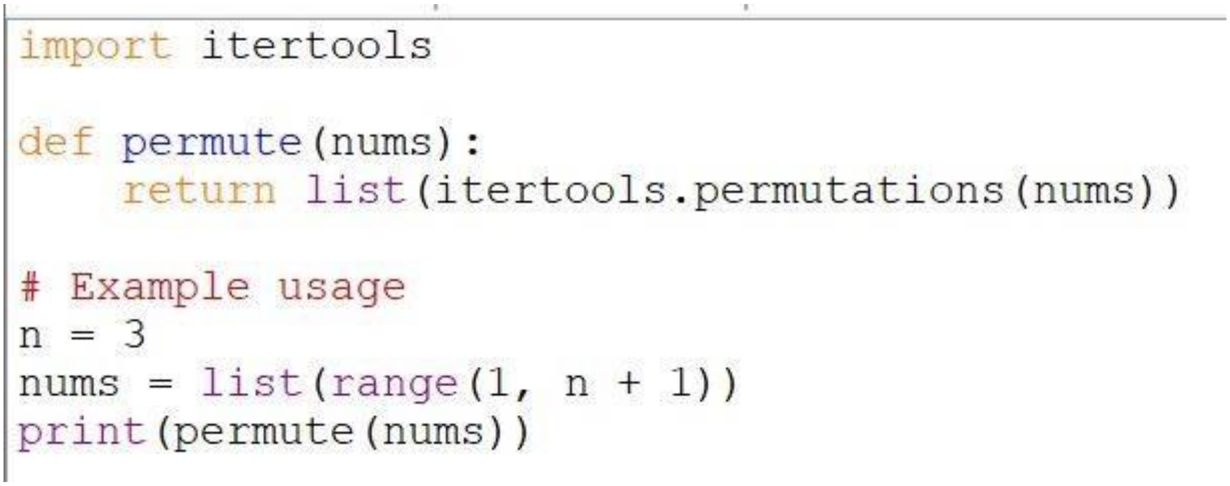
]

Constraints:

- **1 <= candidates.length <= 100**
- **1 <= candidates[i] <= 50**
- **1 <= target <= 30**

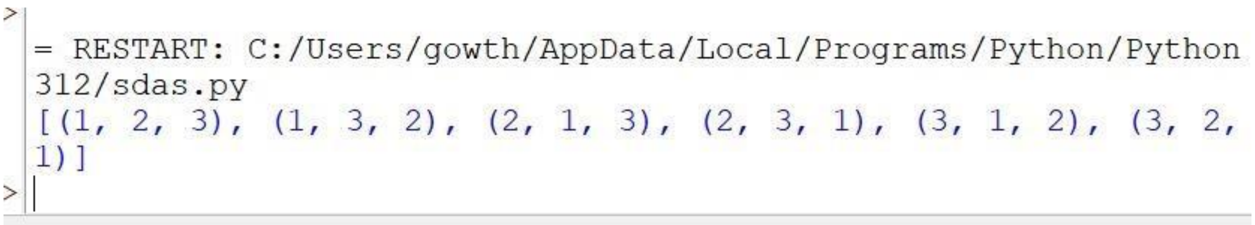
Permutations II

Given a collection of numbers, nums, that might contain duplicates, return *all possible unique permutations in any order.*



```
import itertools

def permute(nums):
    return list(itertools.permutations(nums))

# Example usage
n = 3
nums = list(range(1, n + 1))
print(permute(nums))
```

OUTPUT:



```
>
  = RESTART: C:/Users/gowth/AppData/Local/Programs/Python/Python
  312/sdas.py
  [(1, 2, 3), (1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 1, 2), (3, 2,
  1)]
>
```

Length of Last Word

Given a string s consisting of words and spaces, return *the length of the last word in the string.*

A word is a maximal substring consisting of non-space characters only.

Permutation Sequence

The set [1, 2, 3, ..., n] contains a total of n! unique permutations.

By listing and labeling all of the permutations in order, we get the following sequence for n = 3:

1. "123"
2. "132"
3. "213"
4. "231"
5. "312"
6. "321"

Given n and k , the returnkth permutation sequence.

Example 1:

Input: n = 3, k = 3

Output: "213"

Example 2:

Input: n = 4, k = 9

Output: "2314"

Example 3:

Input: n = 3, k = 1

Output: "123"

Constraints:

- **1 <= n <= 9**
- **1 <= k <= n!**