

Sudoku Validator Code

onecompiler.com/python/439ja22qf

```
main.py + 439ja22qf
```

```
1- def is_valid_sudoku(board):
2-     for row in board:
3-         row_digits = [num for num in row if num != 0]
4-         if len(row_digits) != len(set(row_digits)):
5-             return False
6-     for col in range(9):
7-         col_digits = [board[row][col] for row in range(9) if board[row][col] != 0]
8-         if len(col_digits) != len(set(col_digits)):
9-             return False
10-    for start_row in range(0, 9, 3):
11-        for start_col in range(0, 9, 3):
12-            subgrid_digits = []
13-            for i in range(3):
14-                for j in range(3):
15-                    num = board[start_row + i][start_col + j]
16-                    if num != 0:
17-                        subgrid_digits.append(num)
18-            if len(subgrid_digits) != len(set(subgrid_digits)):
19-                return False
20-    return True
21- board = [
22-     [5, 3, 0, 0, 0, 7, 0, 0, 0],
23-     [6, 0, 0, 1, 9, 5, 0, 0, 0],
24-     [0, 9, 0, 0, 0, 0, 0, 0, 0],
25-     [0, 0, 0, 0, 0, 0, 0, 0, 0],
26-     [0, 0, 0, 0, 0, 0, 0, 0, 0],
27-     [0, 0, 0, 0, 0, 0, 0, 0, 0],
28-     [0, 0, 0, 0, 0, 0, 0, 0, 0],
29-     [0, 0, 0, 0, 0, 0, 0, 0, 0],
30-     [0, 0, 0, 0, 0, 0, 0, 0, 0]
31- ]
32- if is_valid_sudoku(board):
33-     print("The Sudoku board is valid.")
34- else:
35-     print("The Sudoku board is invalid.")
36-
```

STDIN

Input for the program (Optional)

Output

The Sudoku board is valid.

19/02/2025 19:20

Syntax Validator Code

onecompiler.com/python/439ja22qf

main.py

439ja22qf

NEW PYTHON RUN

```
1 import string
2
3 def word_frequency(text):
4     # Convert the text to lowercase to make the count case-insensitive
5     text = text.lower()
6
7     # Remove punctuation from the text
8     text = text.translate(str.maketrans('', '', string.punctuation))
9
10    # Split the text into words
11    words = text.split()
12
13    # Create a dictionary to store word counts
14    word_count = {}
15
16    # Count the frequency of each word
17    for word in words:
18        if word in word_count:
19            word_count[word] += 1
20        else:
21            word_count[word] = 1
22
23    return word_count
24
25 # Example input text
26 text = "Hello world! Hello, how are you? I hope you are doing well. World of programming is amazing."
27
28 # Get the word frequency
29 result = word_frequency(text)
30
31 # Output the result
32 print(result)
```

STDIN

Input for the program (Optional)

Output

{'hello': 2, 'world': 2, 'how': 1, 'are': 2, 'you': 1, 'i': 1, 'hope': 1, 'doing': 1, 'the': 1, 'of': 1, 'programming': 1, 'is': 1, 'amazing': 1}

Sudoku Validator Code 439ja22qf - Python - OneCompiler

onecompiler.com/python/439ja22qf

main.py + 439ja22qf

```
1 def knapsack(weights, values, capacity):
2     # Number of items
3     n = len(weights)
4
5     # Create a 2D DP table with (n+1) rows and (capacity+1) columns
6     dp = [[0 for _ in range(capacity + 1)] for _ in range(n + 1)]
7
8     # Fill the DP table
9     for i in range(1, n + 1):
10         for w in range(1, capacity + 1):
11             # If weight of the current item is greater than the current capacity,
12             # we can't include it in the knapsack
13             if weights[i - 1] <= w:
14                 # Max value by either including or excluding the current item
15                 dp[i][w] = max(dp[i - 1][w], dp[i - 1][w - weights[i - 1]] + values[i - 1])
16             else:
17                 # If current item can't be included, take the value without it
18                 dp[i][w] = dp[i - 1][w]
19
20     # The maximum value will be in the bottom-right cell of the DP table
21     return dp[n][capacity]
22
23 # Example input:
24 weights = [2, 3, 4, 5] # Weights of items
25 values = [3, 4, 5, 6] # Values of items
26 capacity = 5          # Maximum capacity of the knapsack
27
28 # Call the function to solve the problem
29 max_value = knapsack(weights, values, capacity)
30
31 # Output the result
32 print("Maximum value that can be carried:", max_value)
33
```

STDIN

Input for the program (Optional)

Output:

Maximum value that can be carried: 7

19/02/2025 19:35

Sudoku Validator Code 439ja22qf - Python - OneCompiler

onecompiler.com/python/439ja22qf

main.py

```
1 def merge_intervals(intervals):
2     # First, sort the intervals based on the start time
3     intervals.sort(key=lambda x: x[0])
4
5     # Initialize the merged list
6     merged = []
7
8     for interval in intervals:
9         # If the merged list is empty or the current interval does not overlap
10        # with the last one, simply add it to the merged list
11        if not merged or merged[-1][1] < interval[0]:
12            merged.append(interval)
13        else:
14            # There is overlap, so merge the intervals by updating the end time
15            merged[-1][1] = max(merged[-1][1], interval[1])
16
17    return merged
18
19 # Example input: a list of intervals (start, end)
20 intervals = [
21     [1, 3],
22     [2, 4],
23     [5, 7],
24     [6, 8],
25     [9, 10]
26 ]
27
28 # Call the function to merge intervals
29 merged_intervals = merge_intervals(intervals)
30
31 # Output the merged intervals
32 print("Merged intervals:", merged_intervals)
```

439ja22qf

AI NEW PYTHON RUN

STDIN

Input for the program (Optional)

Output

Merged intervals: [[1, 4], [5, 8], [9, 10]]

7:39 PM 19/02/2025 19:39

A screenshot of a computer monitor displaying a code editor and a terminal window.

The code editor on the left shows a Python script named `main.py` with the following content:

```
1 #!/usr/bin/env python3
2
3 def find_median_sorted_arrays(nums1, nums2):
4     # Merge the two sorted arrays
5     merged_array = sorted(nums1 + nums2)
6
7     # Find the median
8     n = len(merged_array)
9
10    # If the total length is odd, the median is the middle element
11    if n % 2 == 1:
12        return merged_array[n // 2]
13    # If the total length is even, the median is the average of the two middle elements
14    else:
15        mid1 = n // 2
16        mid2 = mid1 - 1
17        return (merged_array[mid1] + merged_array[mid2]) / 2
18
19    # Example input: two sorted arrays
20    nums1 = [1, 3]
21    nums2 = [2]
22
23    # Call the function to find the median
24    median = find_median_sorted_arrays(nums1, nums2)
25
26    # Output the result
27    print("The median is:", median)
28
```

The terminal window on the right has the following interface:

- Buttons: AI, NEW, PYTHON ▾, RUN ▶, DOWNLOAD
- Text input field: Input for the program (Optional)
- Text output field: STDIN, Output:
The median is: 2



7:43 PM
19-Feb-25

19/02/2025 19:43

INSPIRON

Python - OneCompiler X

```
1 if not matrix or not matrix[0]:
2     return 0
3
4 # Get number of rows and columns
5 rows = len(matrix)
6 cols = len(matrix[0])
7
8 # Create a list to store the heights of the histogram
9 heights = [0] * cols
10 max_area = 0
11
12 # Process each row
13 for row in matrix:
14     for i in range(cols):
15         # If the value is 1, increase the height, else reset the height
16         if row[i] == '1':
17             heights[i] = heights[i] + 1 if row[i] == '1' else 0
18         # Calculate the area of the largest rectangle that can be formed in this row's histogram
19         max_area = max(max_area, largestRectangleArea(heights))
20
21 return max_area
22
23 def largestRectangleArea(heights):
24     # Stack-based approach to find largest rectangle in histogram
25     stack = []
26     max_area = 0
27     heights.append(0) # Append a 0 to pop out all elements from stack at the end
28
29     for i, h in enumerate(heights):
30         while stack and heights[stack[-1]] > h:
31             height = heights[stack.pop()]
32             width = i if not stack else i - stack[-1] - 1
33             max_area = max(max_area, height * width)
34
35     stack.append(i)
36
37 return max_area
38
39 # Example input: binary matrix
40 matrix = [
41     ["0", "0", "1", "0", "0"],
42     ["1", "0", "1", "1", "1"],
43     ["1", "0", "1", "1", "1"],
44     ["1", "1", "1", "1", "1"],
45     ["1", "0", "0", "1", "0"]
46 ]
```

STDRN

Input for the program (Optional)

Output:

the area of the largest rectangle is: 6

8:00 PM
19-Feb-25

19/02/2025 19:59

INSPIRATION

A screenshot of a Windows desktop environment. On the left, there is a code editor window titled "439ja22pl - Python - OneCompiler" containing Python code for finding the maximum sum of contiguous subarray. On the right, there is a terminal window titled "439ja22pl" showing the output of the program. The taskbar at the bottom shows various application icons.

```
1 def maxSubarray(nums):
2     # Initialize the variables
3     current_sum = nums[0]
4     max_sum = nums[0]
5
6     # Iterate through the array starting from the second element
7     for num in nums[1:]:
8         # Update current_sum to be either the current number itself or current_sum + num
9         current_sum = max(num, current_sum + num)
10
11         # Update max_sum if current_sum is greater than max_sum
12         max_sum = max(max_sum, current_sum)
13
14
15     return max_sum
16
17
18     # Example input: a list of integers
19     nums = [-2, 1, -3, 4, -1, 2, 1, -5, 4]
20
21     # Call the function to find the maximum sum of contiguous subarray
22     result = maxSubarray(nums)
23
24     # Output the result
25     print("The maximum sum of the contiguous subarray is:", result)
```

STDIN
Input for the program (Optional)
Output
The maximum sum of the contiguous subarray is: 6

806 PM
19-Feb-25

19/02/2025 20:03

```
439ja22qf
from collections import deque

def word_ladder(start, end, word_dict):
    # If the end word is not in the dictionary, return an empty list
    if end not in word_dict:
        return []

    # Initialize the queue with the start word and its transformation path
    queue = deque([(start, [start])])

    # Set to keep track of visited words to avoid revisiting
    visited = set()
    visited.add(start)

    # Perform BFS
    while queue:
        current_word, path = queue.popleft()

        # Generate all possible one letter transformations
        for i in range(len(current_word)):
            for char in 'abcdefghijklmnopqrstuvwxyz':
                # Generate a new word by changing one letter at a time
                next_word = current_word[:i] + char + current_word[i+1:]

                # If next_word is the end word, return the current path with the new word
                if next_word == end:
                    return path + [next_word]

                # If next_word is in the dictionary and not visited, add to the queue
                if next_word in word_dict and next_word not in visited:
                    visited.add(next_word)
                    queue.append((next_word, path + [next_word]))

    # If there's no transformation path, return an empty list
    return []

# Example input: start word, end word, and word dictionary
start = "hit"
end = "cog"
word_dict = {"hot", "dot", "dog", "lot", "log", "cog"}

# Edit the function to find the shortest transformation sequence
result = word_ladder(start, end, word_dict)
print(result)
```

19/02/2025 20:08