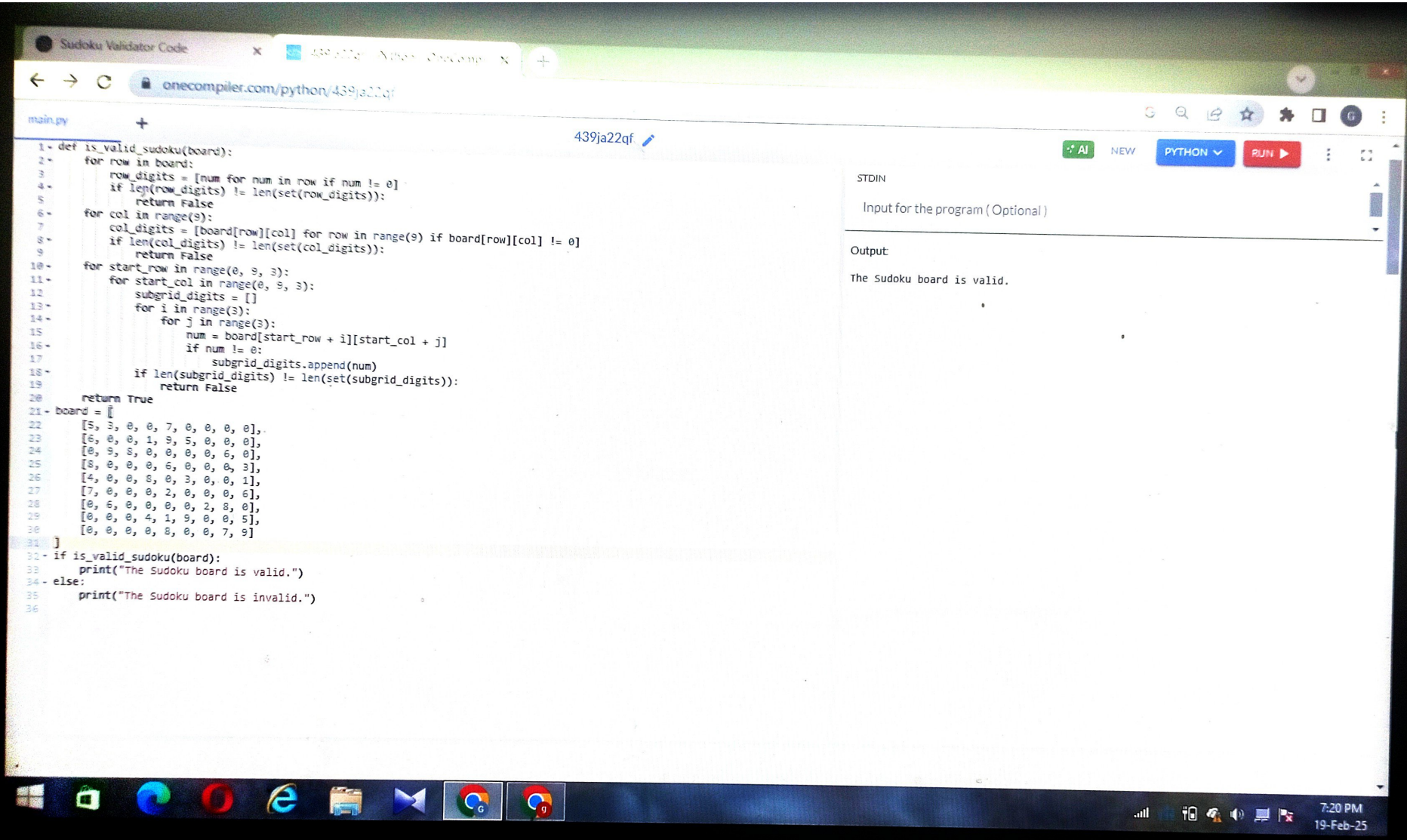
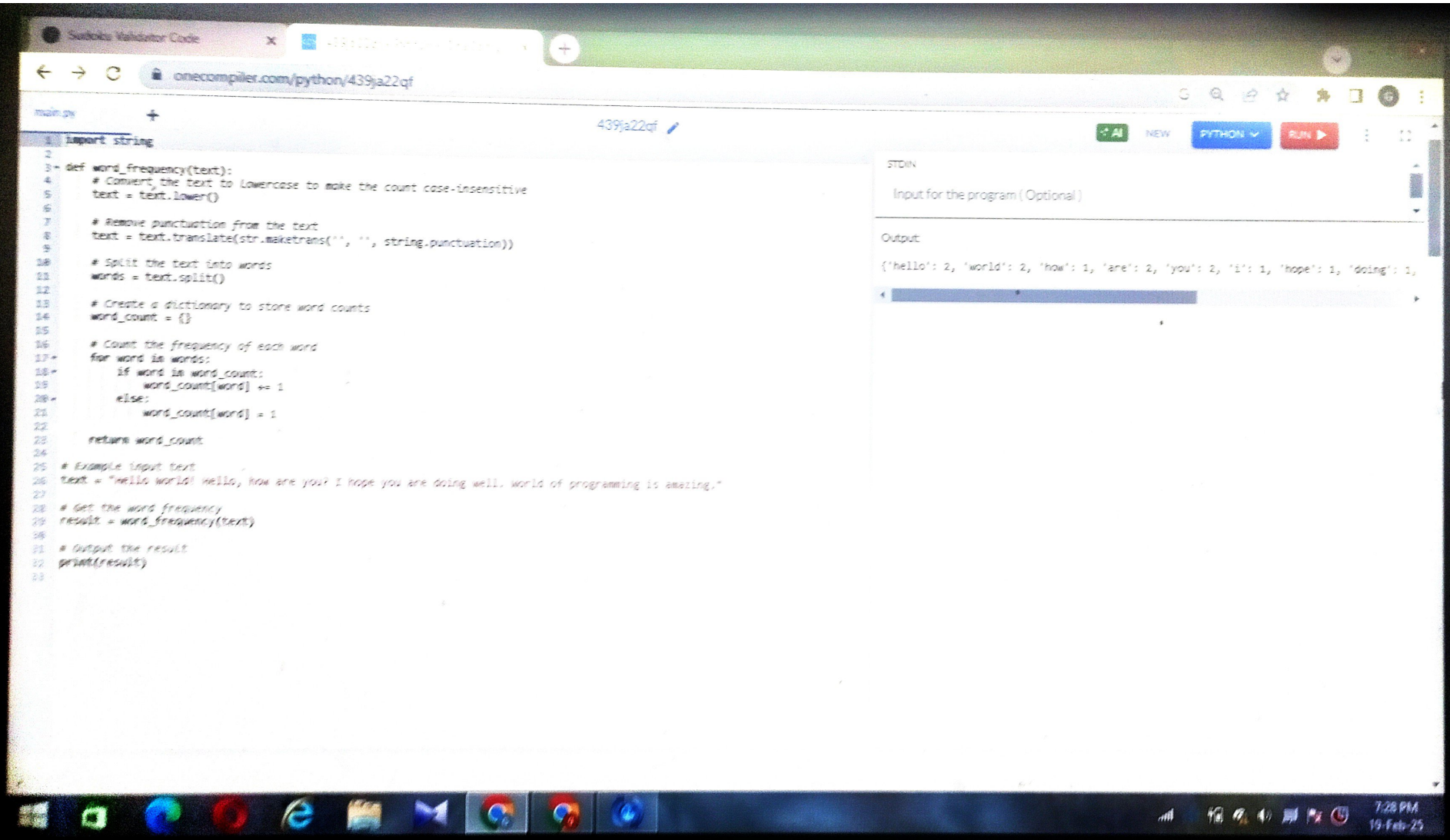



```python
def is_valid_sudoku(board):
    for row in board:
        row_digits = [num for num in row if num != 0]
        if len(row_digits) != len(set(row_digits)):
            return False
    for col in range(9):
        col_digits = [board[row][col] for row in range(9) if board[row][col] != 0]
        if len(col_digits) != len(set(col_digits)):
            return False
    for start_row in range(0, 9, 3):
        for start_col in range(0, 9, 3):
            subgrid_digits = []
            for i in range(3):
                for j in range(3):
                    num = board[start_row + i][start_col + j]
                    if num != 0:
                        subgrid_digits.append(num)
            if len(subgrid_digits) != len(set(subgrid_digits)):
                return False
    return True
board = [
    [5, 3, 0, 0, 7, 0, 0, 0, 0],
    [6, 0, 0, 1, 9, 5, 0, 0, 0],
    [0, 9, 8, 0, 0, 0, 0, 6, 0],
    [8, 0, 0, 0, 6, 0, 0, 0, 3],
    [4, 0, 0, 8, 0, 3, 0, 0, 1],
    [7, 0, 0, 0, 2, 0, 0, 0, 6],
    [0, 6, 0, 0, 0, 0, 2, 8, 0],
    [0, 0, 0, 4, 1, 9, 0, 0, 5],
    [0, 0, 0, 0, 8, 0, 0, 7, 9]
]
if is_valid_sudoku(board):
    print("The Sudoku board is valid.")
else:
    print("The Sudoku board is invalid.")
```

19/02/2025 19:20




```python
import string

def word_frequency(text):
    # Convert the text to lowercase to make the count case-insensitive
    text = text.lower()

    # Remove punctuation from the text
    text = text.translate(str.maketrans('', '', string.punctuation))

    # Split the text into words
    words = text.split()

    # Create a dictionary to store word counts
    word_count = {}

    # Count the frequency of each word
    for word in words:
        if word in word_count:
            word_count[word] += 1
        else:
            word_count[word] = 1

    return word_count

# Example input text
text = "Hello world! Hello, how are you? I hope you are doing well. World of programming is amazing."

# Get the word frequency
result = word_frequency(text)

# Output the result
print(result)
```

19/02/2025 19:28

Sudoku Validator Code | 439ja22qf - Python - OneCompil

onecompiler.com/python/439ja22qf

main.py

439ja22qf

AI NEW PYTHON RUN

```python
def knapsack(weights, values, capacity):
    # Number of items
    n = len(weights)

    # Create a 2D DP table with (n+1) rows and (capacity+1) columns
    dp = [[0 for _ in range(capacity + 1)] for _ in range(n + 1)]

    # Fill the DP table
    for i in range(1, n + 1):
        for w in range(1, capacity + 1):
            # If weight of the current item is greater than the current capacity,
            # we can't include it in the knapsack
            if weights[i - 1] <= w:
                # Max value by either including or excluding the current item
                dp[i][w] = max(dp[i - 1][w], dp[i - 1][w - weights[i - 1]] + values[i - 1])
            else:
                # If current item can't be included, take the value without it
                dp[i][w] = dp[i - 1][w]

    # The maximum value will be in the bottom-right cell of the DP table
    return dp[n][capacity]

# Example input:
weights = [2, 3, 4, 5]  # Weights of items
values = [3, 4, 5, 6]   # Values of items
capacity = 5            # Maximum capacity of the knapsack

# Call the function to solve the problem
max_value = knapsack(weights, values, capacity)

# Output the result
print("Maximum value that can be carried:", max_value)

```

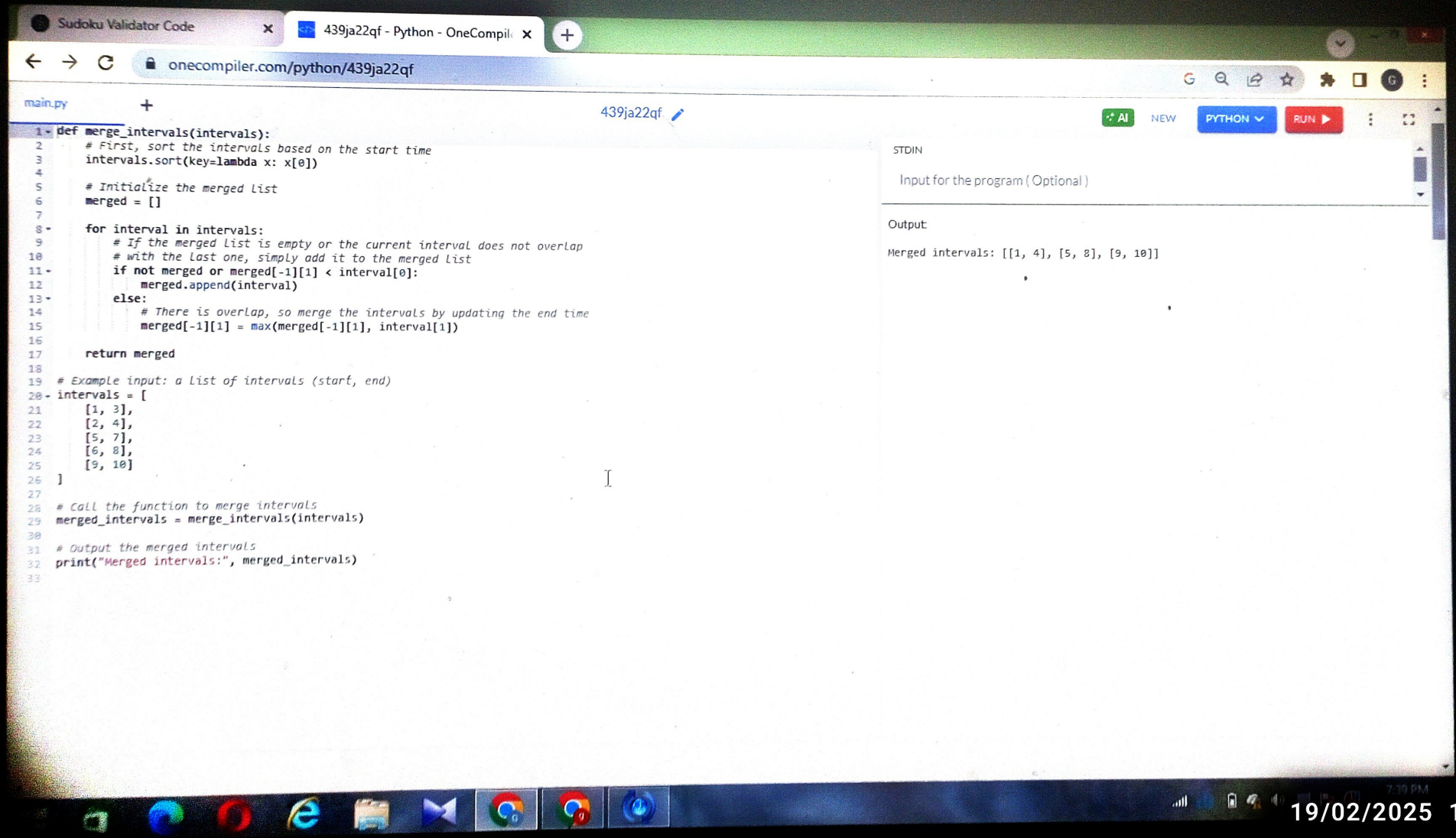
STDIN

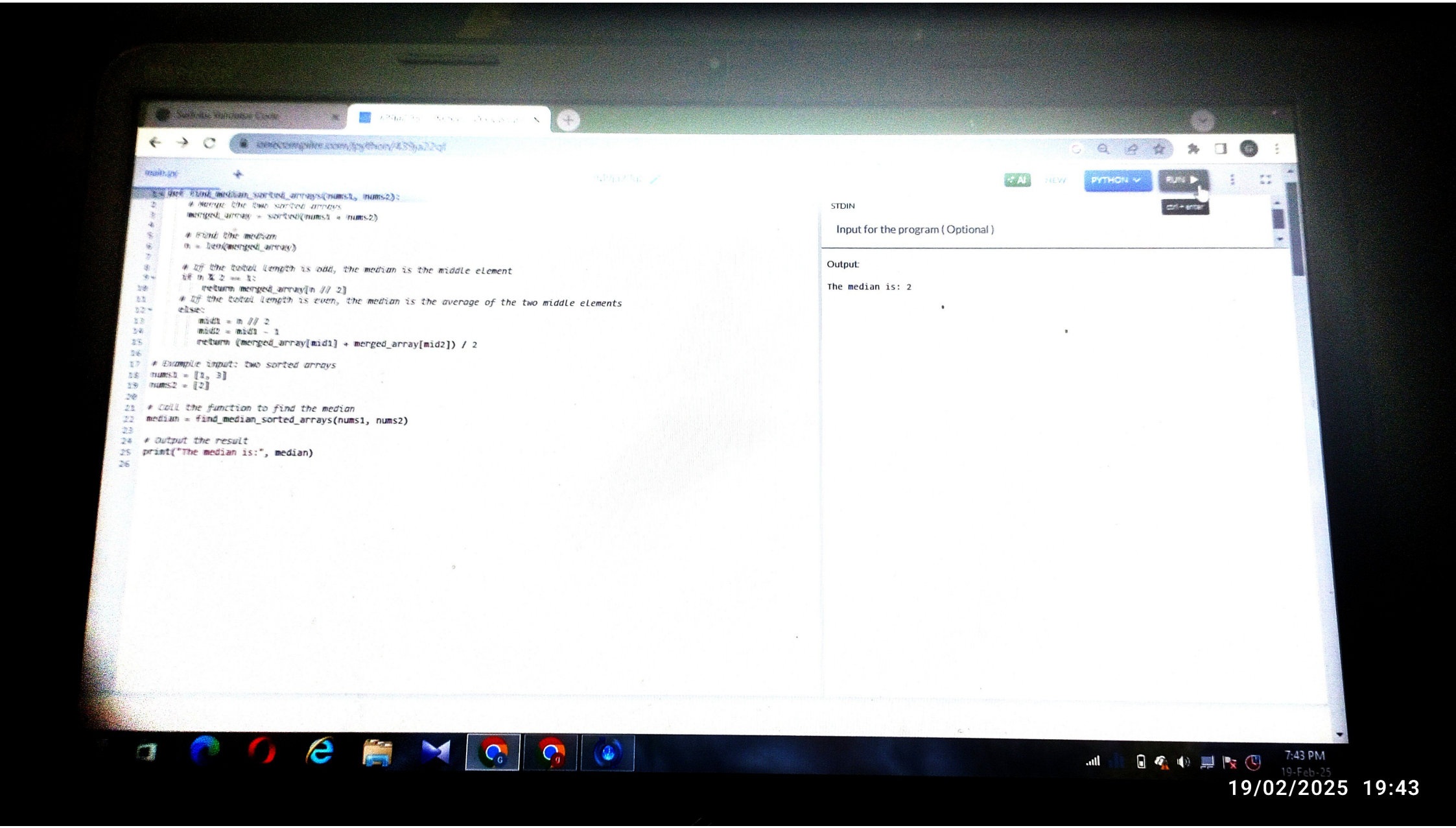
Input for the program (Optional)

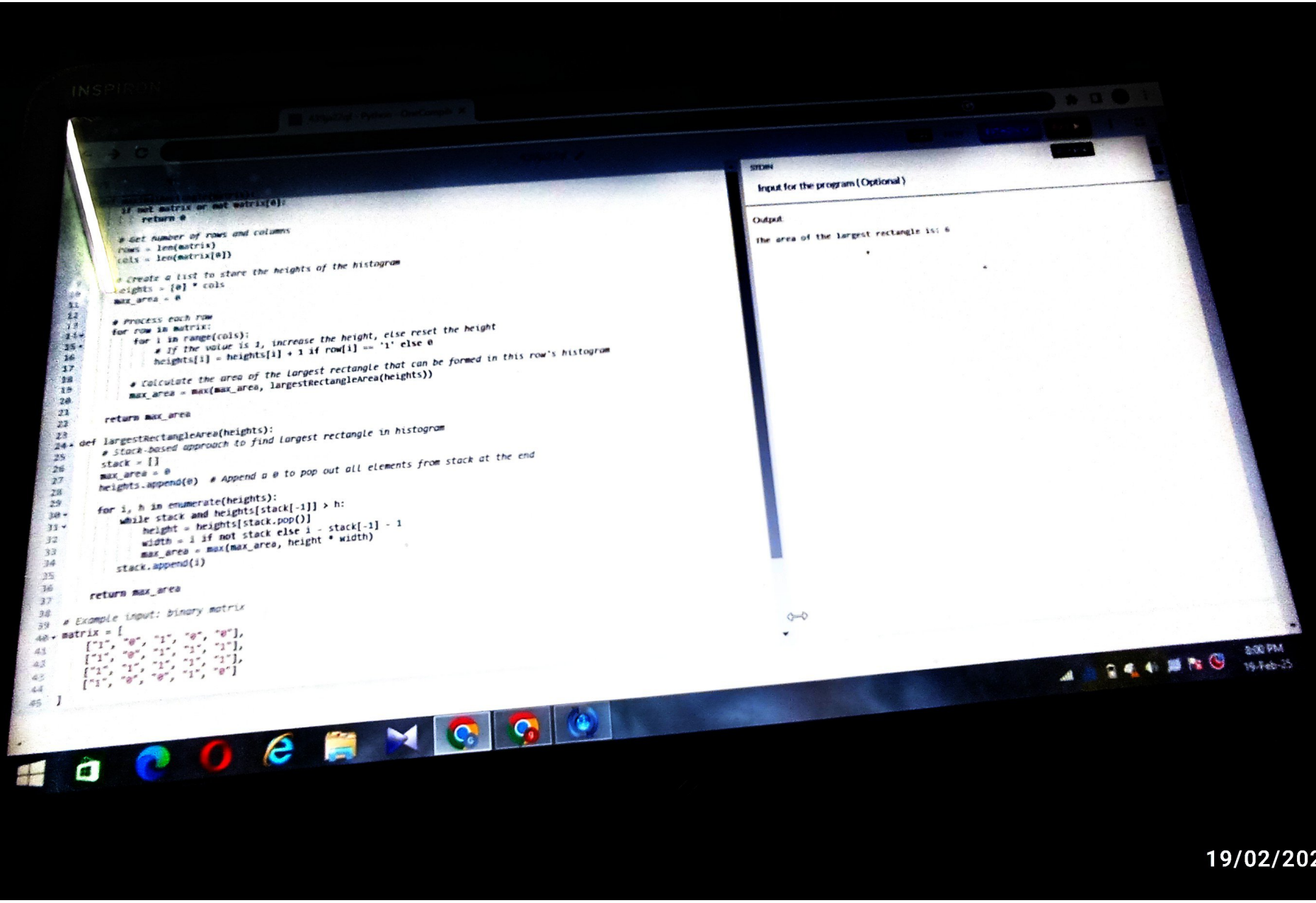
Output:

```
Maximum value that can be carried: 7
```

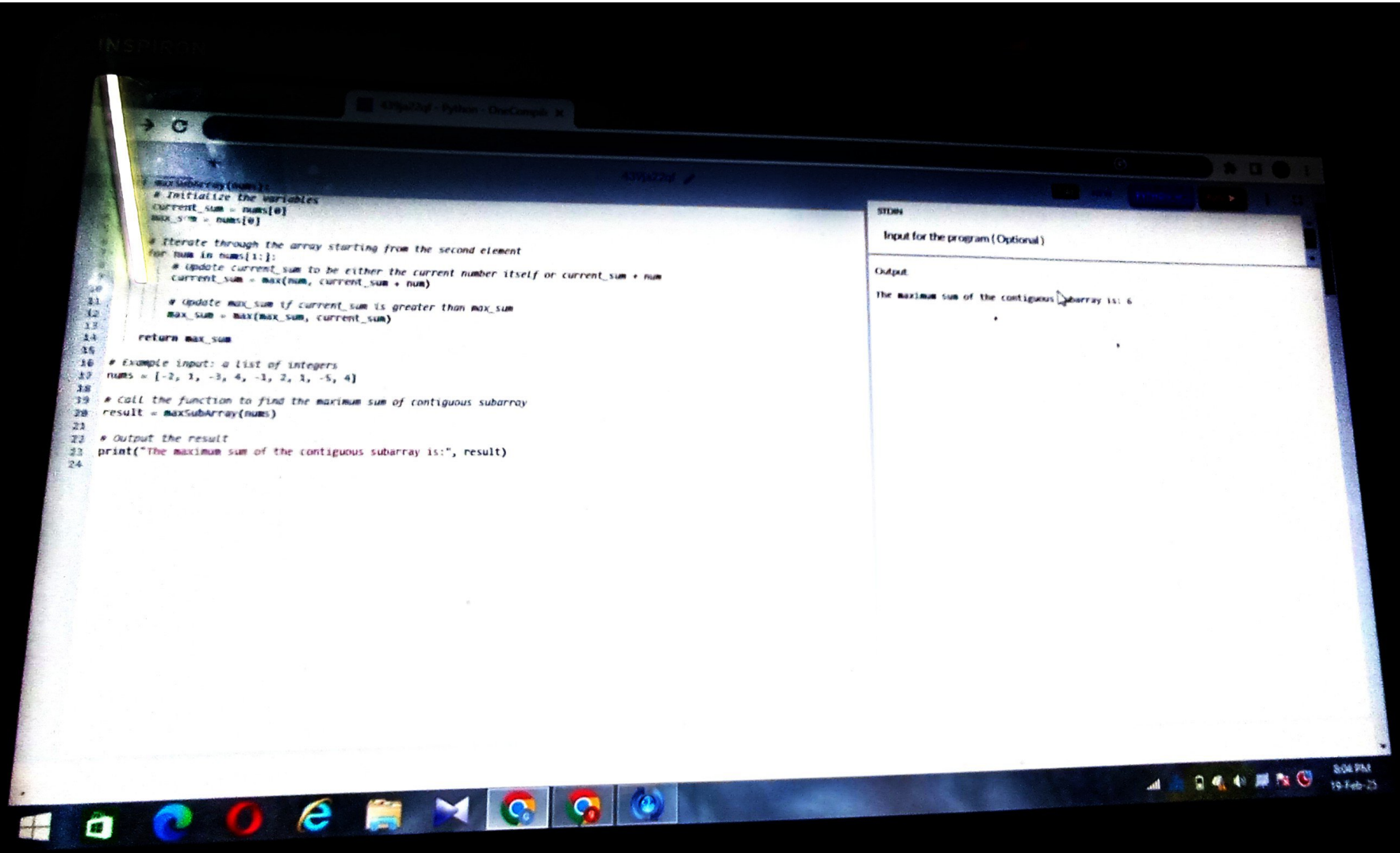
19/02/2025 19:35


Sudoku Validator Code
439ja22qf - Python - OneCompil
onecompiler.com/python/439ja22qf
main.py
439ja22qf
AI
NEW
PYTHON
RUN
STDIN
Input for the program (Optional)
Output:
Merged intervals: [[1, 4], [5, 8], [9, 10]]
def merge_intervals(intervals):
# First, sort the intervals based on the start time
intervals.sort(key=lambda x: x[0])
# Initialize the merged list
merged = []
for interval in intervals:
# If the merged list is empty or the current interval does not overlap
# with the last one, simply add it to the merged list
if not merged or merged[-1][1] < interval[0]:
merged.append(interval)
else:
# There is overlap, so merge the intervals by updating the end time
merged[-1][1] = max(merged[-1][1], interval[1])
return merged
# Example input: a list of intervals (start, end)
intervals = [
[1, 3],
[2, 4],
[5, 7],
[6, 8],
[9, 10]
]
# Call the function to merge intervals
merged_intervals = merge_intervals(intervals)
# Output the merged intervals
print("Merged intervals:", merged_intervals)
19/02/2025 19:39

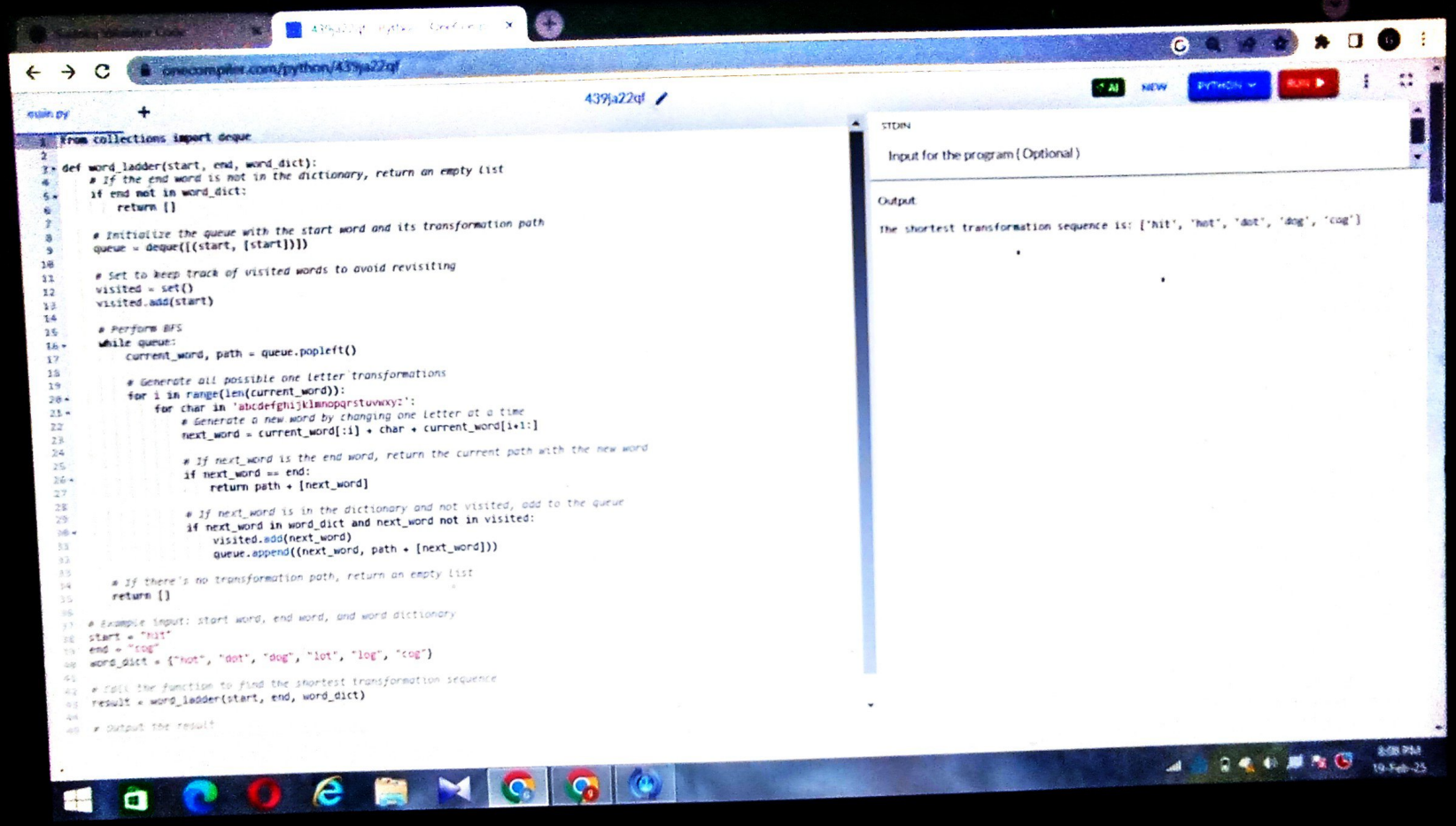

main.py
AI
NEW
PYTHON
RUN
def find_median_sorted_arrays(nums1, nums2):
# Merge the two sorted arrays
merged_array = sorted(nums1 + nums2)
# Find the median
n = len(merged_array)
# If the total length is odd, the median is the middle element
if n % 2 == 1:
return merged_array[n // 2]
# If the total length is even, the median is the average of the two middle elements
else:
mid1 = n // 2
mid2 = mid1 - 1
return (merged_array[mid1] + merged_array[mid2]) / 2
# Example input: two sorted arrays
nums1 = [1, 3]
nums2 = [2]
# Call the function to find the median
median = find_median_sorted_arrays(nums1, nums2)
# Output the result
print("The median is:", median)
STDIN
Input for the program (Optional)
Output:
The median is: 2
7:43 PM
19-Feb-25
19/02/2025 19:43



19/02/2025 19:59



19/02/2025 20:03



19/02/2025 20:08