1. Interactive Bubble Sort Visualization Challenge (30 Points)

Objective

Create an interactive visualization system that shows how Bubble Sort works step by step.

Requirements

1. User Input Features

- Let users:
  - Enter their own numbers
  - Generate random numbers
  - Choose how many numbers (max 100)
  - Control sorting speed

2. Visual Display

Show:

- Current state of numbers
- Which numbers are being compared
- Swapping animation
- Which part is already sorted
- Which pass number we're on
- How far along we are

3. Basic Metrics

Count and show:

- How many comparisons made
- How many swaps done
- How long it took
- Current progress percentage

Required Functions

```
class BubbleSortVisualizer:
    def __init__(self):
        self.numbers = []
        self.comparisons = 0
        self.swaps = 0
        self.time_taken = 0
        self.current_step = 0

    def get_random_numbers(self, size):
        """Make random numbers"""
        pass

    def set_user_numbers(self, numbers):
        """Use numbers from user"""
        pass

    def show_current_step(self):
        """Show what's happening now"""
        pass

    def do_sorting(self):
        """Sort and show each step"""
        pass

    def show_stats(self):
        """Show how many swaps and comparisons"""
        pass

    def animate_swap(self, pos1, pos2):
        """Show numbers swapping places"""
        Pass
```

Example Output

```
Numbers now: [5, 3, 8, 4, 2]
Step 1: Looking at 5 and 3
[5*, 3*, 8, 4, 2] -> Need to swap
[3, 5, 8, 4, 2]   -> After swap
--------------------
Stats:
- Compared: 1 times
- Swapped: 1 times
- Time: 0.001s
Done: 5%
```

2. Selection Sort Challenge: Student Performance Analyzer (30 Points)

Problem Description

You are tasked with creating a student performance analysis system for a school. The system should sort and analyze student records using the Selection Sort algorithm.

Requirements

1. Create Student Class

```
class Student:
    def __init__(self, id, name, score, attendance):
        self.id = id          # Student ID (string)
        self.name = name      # Student name (string)
        self.score = score    # Test score (float)
        self.attendance = attendance  # Attendance percentage (float)
```

2. Implement Modified Selection Sort

Your task is to implement a Selection Sort that can:

- Sort students by different criteria (score, attendance, name)
- Sort in both ascending and descending order
- Track and display the sorting process
- Count comparisons and swaps

```
def selection_sort_students(students, sort_by, order='ascending'):
    """
    Sort student records using selection sort

    Parameters:
    students: List of Student objects
    sort_by: 'score', 'attendance', or 'name'
    order: 'ascending' or 'descending'

    Returns:
    sorted_students: Sorted list of Student objects
    stats: Dictionary containing number of comparisons and swaps
    """
```

3. Required Functions

```
def selection_sort_students(students, sort_by, order='ascending'):
    """
    Sort student records using selection sort

    Parameters:
    students: List of Student objects
    sort_by: 'score', 'attendance', or 'name'
    order: 'ascending' or 'descending'

    Returns:
    sorted_students: Sorted list of Student objects
    stats: Dictionary containing number of comparisons and swaps
    """
```

Specific Tasks

1. Basic Implementation:
    - Implement Selection Sort for the Student class
    - Add visualization of each step
    - Count and display operations
2. Multiple Sorting Criteria:
    - Score (highest to lowest)
    - Attendance (highest to lowest)
    - Name (alphabetical order)

Example Test Data

```
test_data = [
    Student("A001", "John", 85.5, 92.0),
    Student("A002", "Alice", 92.0, 88.5),
    Student("A003", "Bob", 78.5, 95.0),
    Student("A004", "Mary", 90.0, 91.5),
    Student("A005", "David", 88.0, 85.0)
]
```