Scratch Programming: 10. Algorithm

Task: To order a set of numbers

Lesson Objectives:

To learn how to use data structures and algorithms. Here is an explanation of your task:
Computers often need to present a list of values in a sorted order. For example, grades in descending order, names in ascending order, blog entries in chronological order, files in order of megabytes, etc.

Sorting data is something that needs to be done frequently. Let's suppose we have the following list of numbers:

25	63	74	15	45

The list sorted in ascending order would be:

15	25	45	63	74

We will look at a simple algorithm for sorting: **Selection Sort**. It is inefficient but relatively easy to understand and implement. We will also see how we might animate the algorithm to help students visualize how it works.

We will color the elements in the unsorted portion of the list red. In the beginning, the entire list is unsorted so the whole list is red.

25	63	74	15	45

1st pass: find the smallest element in unsorted portion of list (from index 1 to the list length), swap it with the 1st element in the list.
Since 15 is the smallest element from index 1 to 5, we swap it with the 1st element. Now the blue portion of the list is sorted, and the red portion remains to be sorted.

15	63	74	25	45

2nd pass: find the smallest element in the unsorted portion of the list (from index 2 to list length), swap it with the 2nd element in the list. The smallest element is 25, we will swap it with the 63 in the second position:

15	25	74	63	45

3rd pass: find the smallest element in the unsorted portion of the list (from index 3 to list length), swap it with the 3rd element in the list. The smallest element is 45, we will swap it with the 74 in the third position:

15	25	45	63	74

4th pass: find the smallest element in the unsorted portion of the list (from index 4 to list length), swap it with the 4th element in the list. The smallest element is 63. Since it is already in place we do not need to swap:

15	25	45	63	74

When the size of the unsorted portion of the list is 1, we can stop. So, the number of passes is **list length - 1**

What do you need to do?

Follow the instructions below. Always remember to save your work (Save as) in a place you can remember and test your program. Regular saving of your programs will help you if you make mistakes or if there is problem and you lose your work.

Instructions

1. Create a new Scratch program. Delete the Cat and add the Wizard sprite. We will use the wizard to magically sort the numbers. Use only the third costume wizard-c.

2. We will create a new List variable called "grade list". The list will store 5 randomly generated grades. We will sort the list of grades in ascending order. So, in the variable category, click on the "make a list" button and name the list "grade list". You should see the list, which is presently empty, show up on the stage.

We will add 5 random numbers to the list when the green flag is clicked. Note that there is a block "add thing to grade list". If you drag this into the script area, and double click on it, you will see the text "thing" inserted into your list in position 1. If you modify the block to be "add blah to gradelist" and double click on it, you will see "thing" in position 1 and "blah" in position 2. So, we can add elements to the list by using the add block. Each new element will be added to the end of the list.

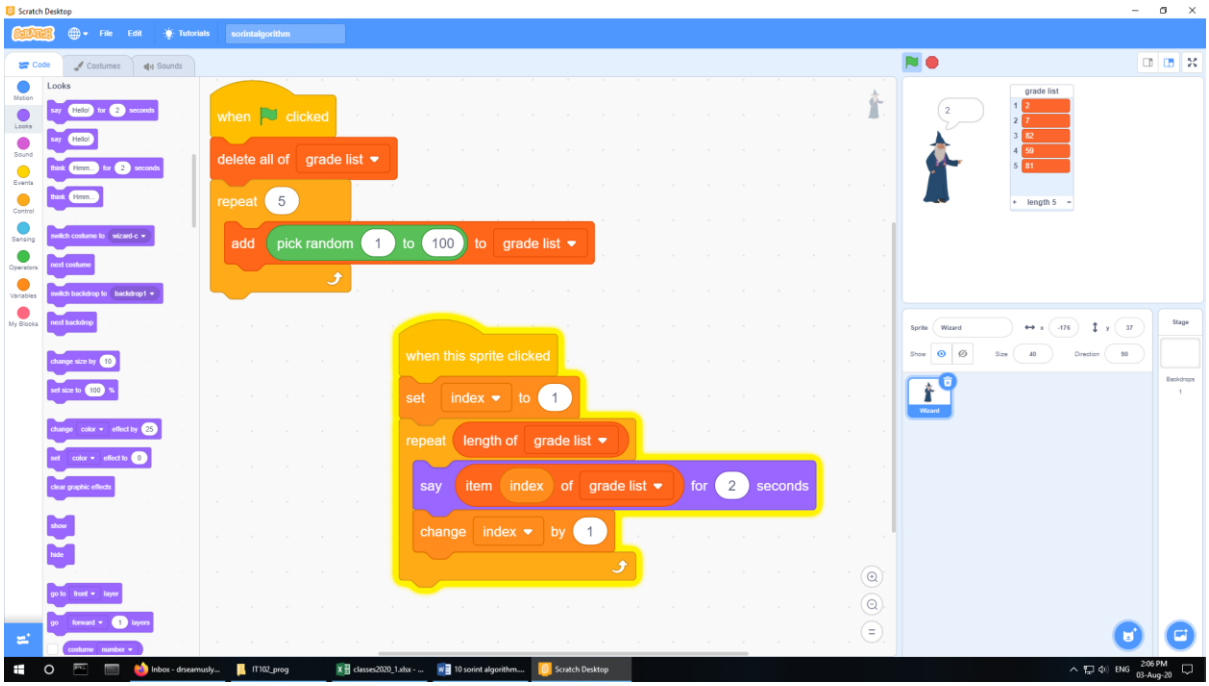
We want to add 5 random numbers to the list each time the green flag is clicked. Use the pick random green block and change to 1 to 100. Note that right now if we added 5 numbers, they would be added to the existing list (which contains "thing" and "blah"), so we first need to remove all of the list elements, then we can add in the 5 random numbers.

Drag in the "delete all of grade list" block, which would delete the first element in the list. Alter it to "delete all of grade list". Press the green flag, you should see the list randomly populated with 5 numbers.

3. We will need to loop through the list, comparing elements to each order and potentially swapping them, in order to sort the list.

First let's just loop through the list, having the wizard say the value of each element. He should say the value of the first element, then the value of the second, etc. We need a variable to keep track of the wizard's position in the list so we can ask for the value at that position.

In many programming languages, the position of an element in a list is often referred to using a variable called index. So, create a new variable called index. Note that the grade list element positions (indices) go from 1 to 5. So, we will initialize the index variable to 1, and then loop 5 times, saying the value of the element in the grade list at the specified index, and increasing the index by 1 each time through the loop.

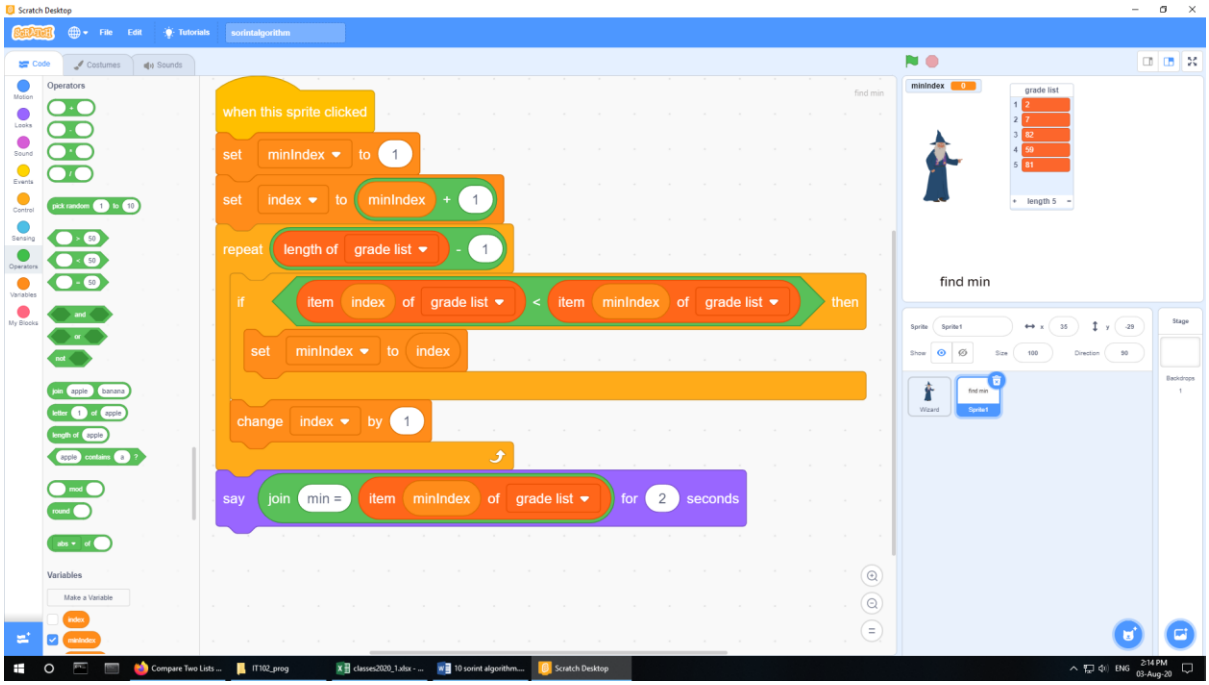


What if we decide to change how many elements are in the list? Our code will break if we hardcode the number of times to loop as 5. We can ask the list for its length, and then use that value to control the repeat loop:

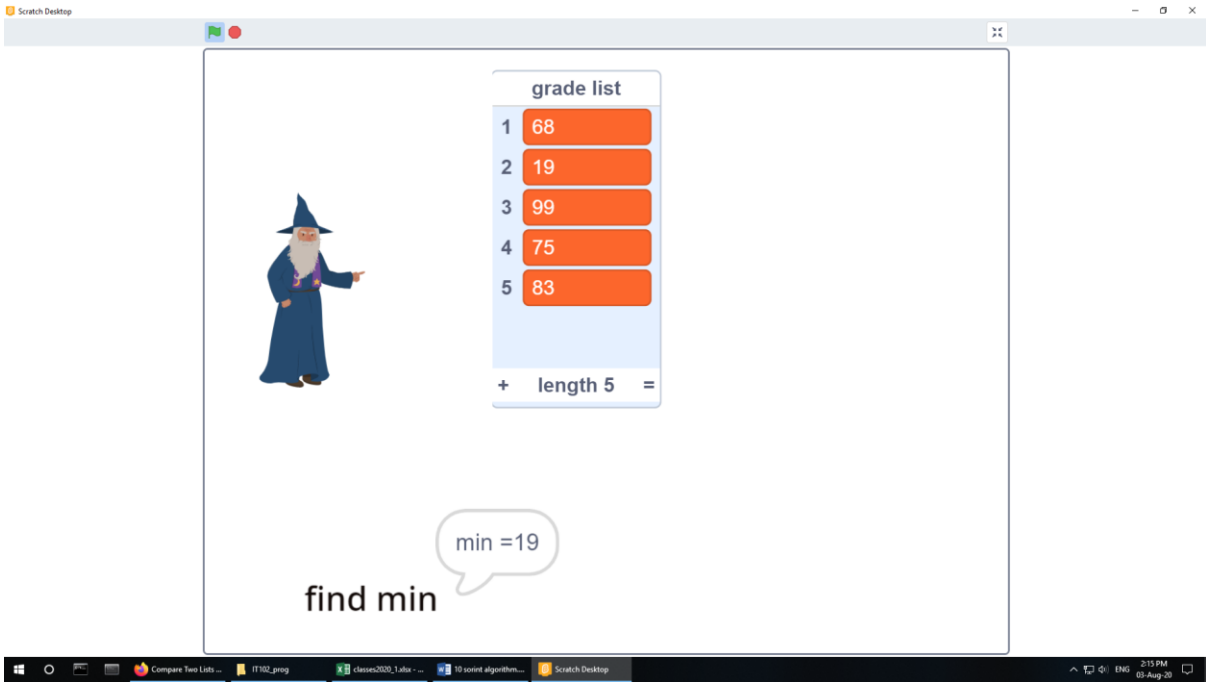
We will implement selection sort by first implementing an algorithm to find the smallest element in the list. What is important is not just the value of the smallest element, but the POSITION/Index of the smallest value. This is critical because we need to place the smallest value in the 1st position in the list, and the value that had been in the first position needs to go into the position of the smallest element (we are doing a swap)

4. Let's first add a "**find minimum**" button. When pressed, the button should broadcast "find minimum". The wizard should have a "when I receive *find minimum*" script that loops through the elements to find the index of the smallest value. Let's have the wizard will tell us the value of the smallest element in the list.

We will use the **index** variable to loop through the list. We need another variable called "**minIndex**" that will keep track of the *position of the smallest element* that we've encountered during the loop. **MinIndex** should be initialized to 1. Since we don't need to compare the first element with itself, we will start the comparison loop **index** at ***minIndex+1***. Each time we encounter an element smaller than the element at **minIndex**, we will update **minIndex** to the position of that element (i.e. **index**).



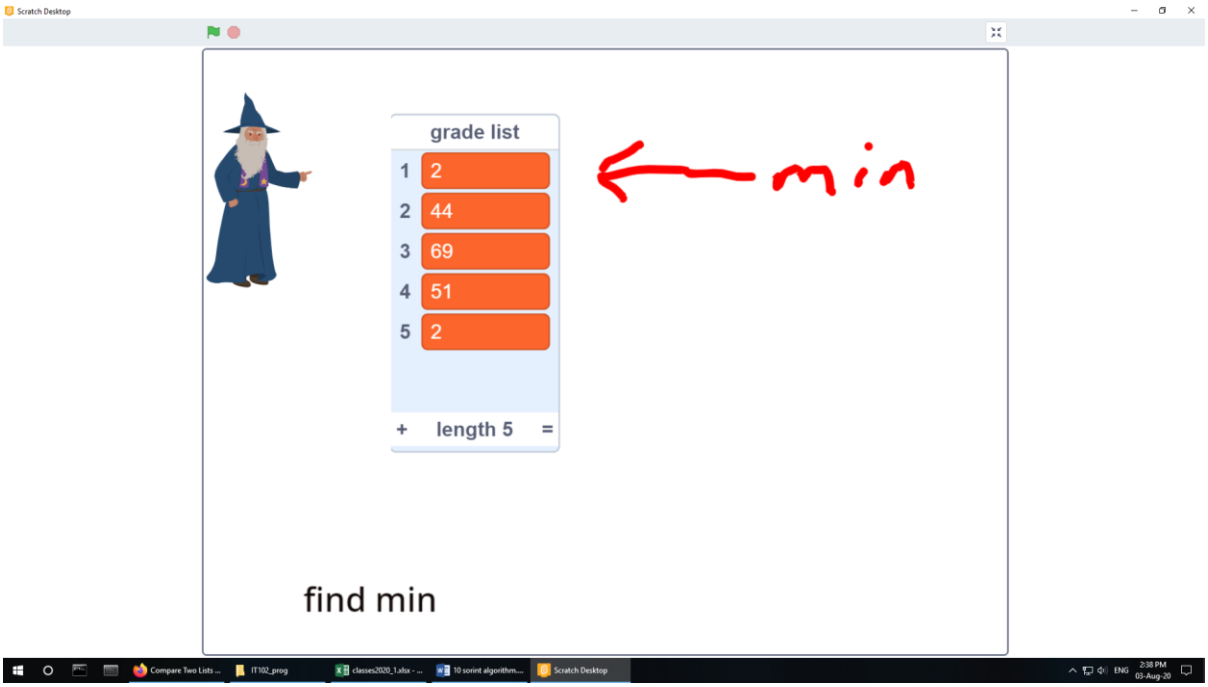
Press the green flag to generate a new list. Press the "minimum" button and test whether the wizard correctly finds the minimum. Repeat this process several times to ensure the minimum is correctly identified.



ANIMATING THE MINIMUM

5. Let's build a visual animation of the sorting algorithm in order to help the student understand how the algorithm works. We will create a sprite that will point to the smallest element in the grade list.

Create a new sprite called **"min animator"**. Use the paint editor to draw an arrow, with the word "min" alongside it. Position the sprite to the right of the grade list. You might want to move the grade list to the left of the stage:



After the wizard has found the smallest element in the list, we should move the "min animator" sprite to position it next to the smallest element.

Have the wizard broadcast "animate min" after the minimum index has been found.

The min-animator sprite should react to the broadcast by changing its y position so that it is in line with the smallest element in the list. Can you think of a way to do this? If we assume the gradelist has a fixed position on the screen, and the vertical spacing between the list elements is fixed, then the only information we need is the "minIndex" value!

Use your mouse to detect how much vertical space there is between the list elements (i.e. space between middle of index 1 and middle of index 2). It looks like it is about 20 pixels. Create a new variable called ***verticalSpacing*** and initialize it when the green flag is clicked.

Use your mouse to detect the vertical position of "grade list". Mine is about 150. Create a new variable called ***verticalStartY*** and initialize it.

Now when the min-animator sprite can be positioned next to the smallest element in the list:

Note that I am hiding the sprite when the green flag is clicked, and waiting until the minimum has been found to show it.

SWAPPING ELEMENTS IN A LIST:

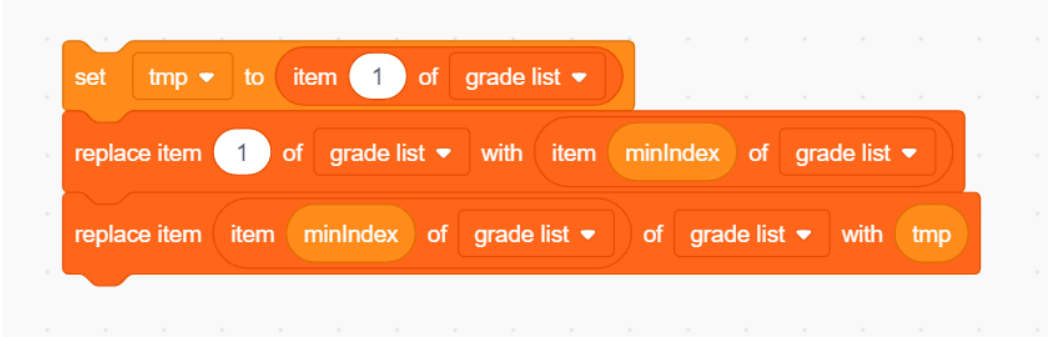
6. We are almost ready to implement the complete sort. Recall the selection sort algorithm:

1st pass: find smallest (in positions 1 . 5), swap smallest with 1st
2nd pass: find smallest (in positions 2 . 5), swap smallest with 2nd
etc.

Repeat the process for (list length - 1) passes.

Let's just build the code for the first pass so we can test the swap, which is the critical step in sorting.
Add a **sort button** sprite, and have it broadcast "sort". The wizard should respond by doing the 1st pass of the sort.
Have him say "pass #1", and then find the minimum. If the minIndex is 1, then the element is in place and we don't need to swap. We will only swap if the minIndex is greater than 1.

How you exchange the values of 2 elements? We need a third memory location for the swap. Create a new variable called "tmp". The swap algorithm is:
tmp = gradelist[1]
gradelist[1] = gradelist[minIndex]
gradelist[minIndex] = tmp

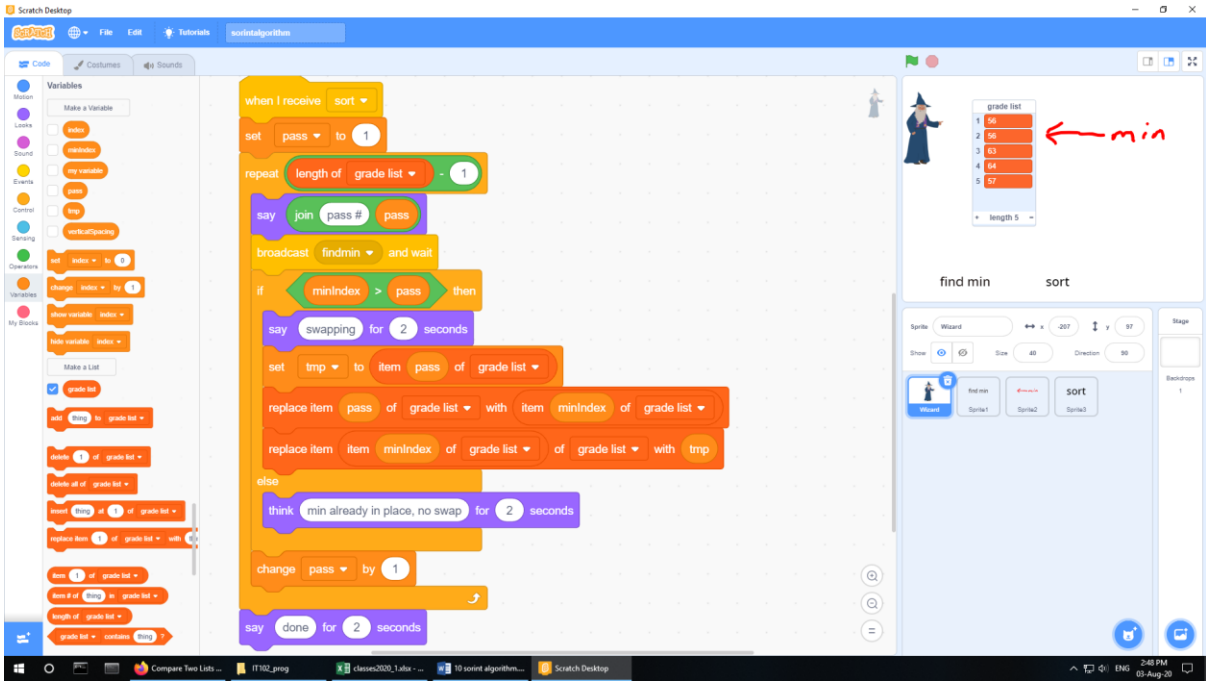


Press the sort button after pressing the green flag. Test that the code correctly swaps (if 1st is not smallest) or skips the swap (if 1st is smallest).

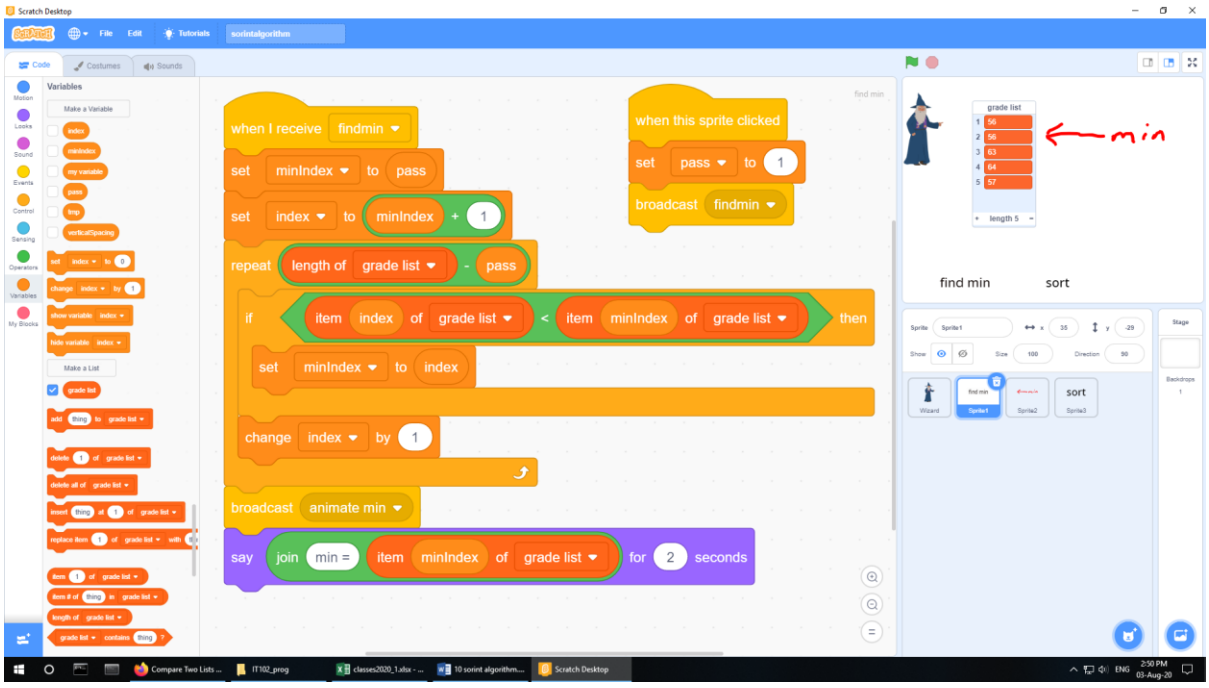
Multiple passes.

We need to loop (list length - 1) times. Use a new loop variable called **pass,** initialized to 1. For each pass, the wizard should say which pass we are on, we will find the minimum, and then do a swap (if necessary). The **minIndex** should be compared to the **pass** (rather than 1). The swap should use the **pass** value to perform the exchange of list elements.

Run the program, what happens? The first pass seems to work, but subsequent passes always find the minimum in position 1. Can you figure out how to fix the code?



The minIndex variable needs to be initialized to the pass value. We also need to update how many times the comparison loop happens, decreasing with each pass.
Note that the "min button" script should also set pass = 1 prior to broadcasting.



Well done you have completed Task 10

Well done you have finished!!!