You will be using this as an event listener for the sortButton. Because buttons associated with a form element submit by default, you need to prevent that behavior. Call event.preventDefault() in your function to do this.

Step 5

Back in your sortInputArray function, you need to get the values from your select elements. Since they all have the class values-dropdown, you can query them all at once.

Use document.getElementsByClassName() to get all the elements with this class by passing in the argument "values-dropdown". Assign that to an inputValues variable with const

Remember that .getElementsByClassName() method returns an *HTMLCollection*, which is an array-like object of all the elements that have a matching class name. You can use the spread operator to convert it into an array.

Convert the document.getElementsByClassName() call to an array with the spread operator and assign it to a variable called inputValues.

You should use console.log() to print out the result of inputValues. Write the code for this inside the sortInputArray function.

To see the logged inputValues array, click on the sort button and open up the console. You should see an array of strings like this:

Example Code

```
[ "8", "2", "4", "1", "3" ]
```

Before going further, make sure you observe the data type of the printed result in the console.

In the next step, you will convert those strings into numbers.

Step 11

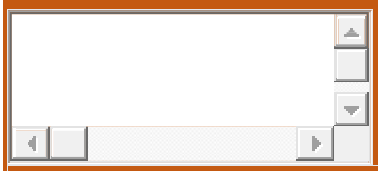
You need a function to update the display with the sorted numbers. Start by using arrow syntax to declare an updateUI function that takes a single array parameter.

Because you will be writing algorithms that won't immediately have a return value, set a fallback value for array to be an empty array. Here is an example of a function that has a fallback value:

Example Code

```
const myFunction = (string = "") => {

}
```



Check Your Code (Ctrl + Enter)

Reset

Navigated to Step 11

The last sorting algorithm you will implement is the *insertion sort*. This algorithm works by building up a sorted array at the beginning of the list. It begins the sorted array with the first element. Then it inspects the next element and swaps it backward into the sorted array until it is in a sorted position, and so on.

Start by declaring an `insertionSort` variable and assigning it an arrow function which takes an `array` parameter.

To sort the elements of an array, you can use the built-in method called `.sort()`. Therefore, you can update the `sortedValues` variable by assigning it the result of calling `.sort()` on the `inputValues` array.

Notice how the number `10` is placed at the beginning of the array. This is because the default behavior of `.sort()` is to convert the numbers values to strings, and sort them alphabetically. And `"10"` comes before `"2"` alphabetically.

To fix this, you can pass a callback function to the `.sort()` method. The callback function has two parameters - for yours, use `a` and `b`. The parameters of `a` and `b` represent the number values in the array that will be sorted.

Leave the function empty for now.

Step 44

The callback to `.sort()` should return a number. That number determines how to sort the elements `a` and `b`:

- If the number is negative, sort `a` before `b`.
- If the number is positive, sort `b` before `a`.
- If the number is zero, do not change the order of `a` and `b`.

Keeping in mind that you want the numbers to be sorted in ascending order (smallest to largest), return a single subtraction calculation using `a` and `b` that will correctly sort the numbers with the above logic.