The right way to handle arrays

In this lab, let's work with arrays. We'll keep using our testing structure, but we're going to focus on arrays instead of our roman numeral convertor.

1. Decide which of you will be partner 1 and which will be partner 2. Follow the instructions below.

Partner 1

2. Create a new test file called arrays.spec.js. In it, create a *describe* (aka. test group) with one test that just returns successfully.
3. Run your test to make sure it works.
4. Write a new test for "can be created". It should verify the existence of an array called people and that the array has five members.
5. Verify that the test fails.

Partner 2

6. Make the test pass by creating a people variable that has five person objects in it. You can make the person look any way you like but they should have a first and last name at minimum.
7. Write a new test for "can be altered". It should look like this:

```
const testPerson = { first: "Jo", last:"Bennett"};
// Add a person to the end of the array here
expect(people.length).toEqual(6);
expect(people[5].first).toEqual("Jo");
// Add a person to the front of the array here
expect(people.length).toEqual(7);
expect(people[0].last).toEqual("Bennett");
// Remove a person from the end of the array here
expect(people.length).toEqual(6);
expect(people[5].first).not.toEqual("Jo");
// Remove a person to the front of the array here
expect(people.length).toEqual(5);
expect(people[0].last).not.toEqual("Bennett");
```

8. Verify that the test fails

Partner 1

9. Make the test pass
10. Write a new test called "can loop two ways"
11. It should pass immediately but should have these two comments in the test:

```
// loop through the persons array using "for in", console.logging each.
// loop through the persons array using "for of", console.logging each.
```

Partner 2

12. Make the tests console.log as they should.
13. Write a test called "can destructure". It should ...

```
// Create five variables called p1, p2, p3, p4, and p5. Each is a
person
// from your array of persons. Use destructuring to do that.
expect(p1).toEqual(people[0]);
expect(p2).toEqual(people[1]);
expect(p3).toEqual(people[2]);
expect(p4).toEqual(people[3]);
expect(p5).toEqual(people[4]);
```

This, of course, will fail.

Partner 1

14. Make the test pass by using destructuring to create those five variables from the people array.
15. WrIte a failing test called "can convert your array of people into an array of strings"
16. // Convert it here

```
expect(arrayOfStrings[0])
  .toEqual(`name: ${people[0].first} ${people[0].last}`);
expect(arrayOfStrings[1])
  .toEqual(`name: ${people[1].first} ${people[1].last}`);
expect(arrayOfStrings[2])
  .toEqual(`name: ${people[2].first} ${people[2].last}`);
expect(arrayOfStrings[3])
  .toEqual(`name: ${people[3].first} ${people[3].last}`);
```

Partner 2

17. Make the test pass

Bonus! Once you're finished, you can continue working on your roman numeral converter. Maybe you can use arrays to enhance it.