Functional JavaScript Lab

In this lab we'll be working completely within a single new spec/test file. We're going to practice with creating functions in a number of ways, working with default parameters, and using rest parameters.

Writing functions in all three ways

- 1. Create a new test file called functions.spec.is.
- 2. At the top create three functions called "expression", "statement", and "arrow". Use all three different types of creating functions. All three should just return true for now.
- 3. Write three tests/specs to make sure that they all work okay.
- 4. Run those tests and fix any problems until they're passing.
- 5. Move all of them to the bottom of functions.spec.js and re-run your tests. Do they pass? They may or may not depending on how you created them and the functions. Discuss with your partner why the worked or didn't.
- 6. Get your tests passing again.
- 7. Go back through all of your specs including those from previous labs and change them all to use arrow functions. Re-run your tests to make sure you haven't broken anything.

Using default parameters

- 8. Write a function called makePerson(). It should do nothing for now.
- 9. Write these tests:
 - "can create a person". It will pass in a first name, a last name, an email, and a phone number. It should return an object with all of those values.
 - "will throw if no name is passed in". It will pass if the function throws when it is given an email address and a phone number, but either first name or last name is not provided. (Hint: you can pass in undefined to simulate that).
 - "will set email to 'no email' if not provided". It will pass in a first and last name, but no email address. Assert that the returned person's email property to 'no email'.
 - "will set phone to 'no phone' if not provided". Same as above but for the phone number.
- 10. Run those tests. They should fail.
- 11. Each partner takes turns making these tests pass.

Using rest parameters

In mathematics, a factorial on an integer n is the product of all positive numbers less than n. In this section, we're going to use TDD to create a special function that calculates factorials.

- 12. At the top of your file, write a function called factorial() does nothing.
- 13. Write a few tests:
 - "can calculate a factorial". It should pass in the number 10 and assert that factorial returns 3628800
 - "can calculate three factorials". It should pass in 3 numbers and get the correct factorials in an output array of size 3.
 - "can calculate 10 factorials". It should pass in 10 numbers and get the correct factorials in an output array of size 10.
 - "will return 1 if given no parameters". It should pass in no parameters and get back the number 1. (Because that is the correct return for the factorial of 0).
- 14. Run those tests. They should all fail.
- 15. Each partner should take turns making these tests pass.