Functional JavaScript Lab

In this lab 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.js. Include it in your web page.
2. At the top create three functions called "expression", "statement", and "arrow". Use all three different types of creating functions. All three should just console.log() its name.
3. After defining the functions, try to run each of them. They should run just fine.
4. Now that you've proven that they run at the top, move all of them to the bottom of functions.js and re-run your test. Do they pass? They may or may not depending on how you created them. Think about why they worked or didn't.
5. Get them running when you're finished.
6. Go back through all of your functions 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 all passed parameters

7. Write a function named makePerson(). It should do nothing for now.
8. Write a testing function called canCreateAPerson. It will call makePerson, passing it a first name, a last name, an email, and a phone number. makePerson should create an object using property shorthand with all of those values and return it.
9. Run your test function, validating that makePerson indeed returns a person object.

Using default parameters

10. Write a testing function called willThrowIfNoNameIsPassedIn. It will call makePerson, passing it no parameters. makePerson() should throw an Error.
11. Change it to pass a first name only (no last name). makePerson() should throw an Error in this case also.
12. Change it to pass a first and a last name. It should not fail and it should set the email address and phone number to some default value.

Once you've proven that makePerson() uses default parameters for email and phone but throws when first name and last name are blank, you can move on.

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 create a special function that calculates factorials.

13. At the top of your file, write a function called factorial() that does nothing.

For the next few steps, you'll write each testing function. Then you'll change the factorial() function in such a way to pass the requirements below.

14. Write a testing function called CanCalculateAFactorial. It should call factorial(), passing in the number 10 and asserting that factorial returns 3628800. If so, print "success" and if not, throw an Error.
15. Write a testing function called canCalculateThreeFactorials. It should call factorial(), passing in 3 numbers and getting the correct factorials in an output array of size 3. If so, print "success" and if not, throw an Error.

16. Write a testing function called canCalculate10Factorials. It should call factorial(), passing in 10 numbers and gettubg the correct factorials in an output array of size 10. . If so, print "success" and if not, throw an Error.
17. Write a testing function called willReturn1IfGivenNoParameters. It should call factorial(), passing in no parameters and getting back the number 1. (Because that is the correct return for the factorial of 0).

Once you have a factorial function that meets all these requirements, you can be finished.