Tutorial for JavaScript Functions

In this tutorial, you'll practice writing some JavaScript functions. These are a lot like the methods you wrote in C# or Java, but they're different in some very subtle ways. You'll also practice documenting your functions so other developers know what's going on and how to use them.

You can run the Live Server on VS Code on either the `ui.html` file or the `test.html` file, whichever you prefer. The `ui.html` file looks blank, but it'll fill in with values when you start writing your functions. The `test.html` file shows a series of unit tests that pass as you complete the tutorial.

Then open `tutorial.js` in VS Code. You'll see several comments and half-finished functions in here. You'll fill those in now.

turnOn()

The first comment at the top asks you to write a function called `turnOn` that returns `true`. You first need to write the function signature in JavaScript:

```javascript
function turnOn() {

}
```

You then need to return the value `true`.

```javascript
function turnOn() {
  return true;
}
```

When you save that, the unit test for that function should now pass and the UI appears. If that didn't work, make sure you spelled everything correctly and make sure your function is after the `*/` of the comments and not before it.

Remember that you can't specify the return type of a function in JavaScript. The only way a programmer knows the return type is by reading the code or reading the JSDoc comments before the code.

returnsName()

The next function is another simple function that returns your name. Try to fill this out yourself without copying from the example below.

This is a function that takes no arguments and returns a string that contains your name. The finished function would look something like this:

```
function returnsName() {
  return "Jane Doe";
}
```

returnGivenParameter()

Functions in JavaScript can also take parameters, just like methods in C# and Java. Write a function that takes a single parameter and then return it. Try to fill this out yourself without copying from the example below.

Again, in JavaScript, you can't specify types for the parameters, so the function signature won't include types, just the name of the parameter. In this case, the parameter is called `thing`, but you can call it anything you want.

```
function returnGivenParameter(thing) {
  return thing;
}
```

takeOptionalParameter()

Functions in JavaScript can't be overloaded as in C# or Java. In those languages, if you want different parameters for the same method, you can create a different method with the same name with those different parameters.

In JavaScript, if you define a function with the same function as another, the first is overwritten by the second and ceases to exist. So to have a function be able to take different numbers of parameters, you can give parameters a default value. If you don't give a default value for a parameter and that parameter is not given, the value is `undefined`. An `undefined` value might not be what you want and might break your function but with a default value, you get to define what a missing parameter means.

Now you'll create a function that takes a single parameter and returns it. However, this time, if you don't get a parameter, you want to return `0`, and you want to do it without using an if statement.

The solution to this looks something this:

```
function takeOptionalParameter(thing = 0) {
  return thing;
}
```

filterArrayToOnlySingleDigitNumbers()

Now you're going to look at writing a function that you can pass to another function. This is a common activity in JavaScript, and you'll use a built-in JavaScript function to perform this task: the Array `filter()` function.

When creating functions to pass to other functions, they are typically written as anonymous functions using the "fat arrow" or "rocket" syntax.

The main part of the function has already been written for you. You need need to fill in the anonymous function.

```
function filterArrayToOnlySingleDigitNumbers(arrayToFilter) {
  return arrayToFilter.filter(
    // Your code goes here
  );
}
```

Remember that filter works by taking a function, running that function on every element in the array, and returning in a new array every element that returned `true`. So what you need to write is a function that returns `true` if the element is only a single digit.

So the anonymous function needs to take one parameter—the element from the array—and return true if the element is a single digit. That could look something like this:

```
(element) => {
  return element < 10;
}
```

Take the element, check to make sure it's less than ten, and then return the result of that check (either `true` or `false`).

In the tests, you can see that there is a part of this that is failing. Can you fix this function to make that pass as well? Try to fill this out yourself without copying from the example below.

If the numbers are negative, the anonymous function doesn't properly filter out double-digit negative numbers. You can fix that by changing the anonymous function to this:

```
(element) => {
  return element < 10 && element > -10;
}
```

mapArrayToDoubleAllNumbers()

There is also an array function called `map()` that lets you perform a calculation on all elements in an array and return a new array with the results. This is used in `mapArrayToDoubleAllNumbers()` to double every number in an array.

The main function is already written for you:

```
function mapArrayToDoubleAllNumbers(arrayToDouble) {
  return arrayToDouble.map(

  );
}
```

Here, whatever you return from your anonymous function is the value added to the new array that is returned from `map()`. That means you can return the calculation result. Try to fill this out yourself without copying from the example below.

Again, you're going to write an anonymous function that takes one parameter and returns the value you want. You could write the function like this:

```
(element) => { return element * 2; }
```

You can see that you can write anonymous functions all on one line if you want to. For those functions, it makes sense to keep it compact.

reduceArrayToFindProduct()

The `reduce()` function takes an array and collapses it down to a single value, essentially accumulating the elements in the array using whatever calculation you define. This could be used to find an average of a collection of values, the sum of all the values or, like you will do, find the product of all the values.

The current function looks like this:

```
function reduceArrayToFindProduct(arrayToMultiply) {
  return arrayToMultiply.reduce(

  );
}
```

Write an anonymous function that multiplies all the values of an array together. One thing to pay attention to is that anonymous functions passed to `reduce()` get two parameters. The first is the current running result of the calculation from previous elements, and the second is the current element you're looking at.

Knowing that, how would you write an anonymous function that could multiply all the values in a given array? Try to fill this out yourself without copying from the example below.

Since you want to multiply all the values together, and you don't care what order you do that in, you can just multiply the current element by the result of all the others before it:

```
(currentResult, element) => {
  return currentResult * element;
}
```

filterStringArrayForSon()

These functions aren't limited to just numbers. You can use arrays of any type with these functions.

```
function filterStringArrayForSon(arrayToFilter) {
  return arrayToFilter.filter(

  );
}
```

Write a function that can take an array of names and returns only those names that contain son. Try to fill this out yourself without copying from the example below.

Knowing that all the elements of this array are strings—or at least hoping they are because it says so in the comments for the function—you can use the string methods to check that the string contains son. In fact, VS Code tells you what methods the parameter has because it reads the JSDoc and checks whether you're dealing with strings or not.

```
(element) => { return element.includes('son'); }
```

makeNamesAllCaps()

If you want to change every string in an array, you can use the map() method to do that.

```
function makeNamesAllCaps(arrayToCapitalize) {
  return arrayToCapitalize.map(

  );
}
```

What function would you write here to convert the names to all upper case? Try to fill this out yourself without copying from the example below.

Again, VS Code knows what functions are available on strings. You can use toUpperCase() to convert the array:

```
(element) => { return element.toUpperCase(); }
```

convertTemperature()

The convertTemperature() method is already written, but it's missing comments. Using the other examples, try writing some comments for this function. To get a template ready for you to fill out, go to the line right

before the function and type /** and hit enter. VS Code fills in a lot of what you need to write. Remember to document parameter types, return types, and optional parameters that have default values.