JavaScript – theory and tasks

Scoping

Scope is a big idea in programming, so let's start at a high level.

Scope refers to where in a program a variable can be accessed. The idea is that some variables are unable to be accessed everywhere within a program.

Think of it like a an apartment building. Everyone who lives in the apartment building is under the global scope of the building and its manager. So, if there are rats in the shared laundry room, everyone has access to the laundry machines, and the rats.

If you write a variable outside of a function in JavaScript, it's in the global scope and can be used by any other part of the program, just like the laundry room can be used by everyone in an apartment.

Let's practice this by making some global variables.

***Instructions***

*Write your code in the scope.js file.*

1. Write two variables, one named laundryRoom set equal to 'Basement', the other named mailRoom set equal to 'Room 1A'.
2. Below these variables, use console.log to print both variables to the console, like this:

console.log('Laundry: ' + laundryRoom + ', Mail: ' + mailRoom);

In our theoretical apartment building, you have your own apartment. It has stuff in it that is yours. Other people in the building can't access it. This is like functional scope. You have access to your stuff inside your apartment, and in the building – but not anyone else's apartment.

When we write variables inside a function, only that function has access to its own variables. Therefore, they are in the functional scope.

In addition to a function having access to its own variables, it also has access to variables in the global scope.

In the last exercise we created both variables in the global scope. That is, laundryRoom and mailRoom are accessible from anywhere in our program.

Now, let's make two variables within a functional scope.

***Instructions***

1. Write a function named myApartment before the console.logs from the last exercise.
2. Inside of the function, write a variable named mailBoxNumber and set it equal to Box 3.

Also, you're lucky enough to have in-unit laundry, so let's re-assign laundryRoom inside our function to: laundryRoom = 'In-unit'.

1. Inside the function, use console.log to print out both variables, like this:

console.log('Mail box: ' + mailBoxNumber + ', Laundry:' + laundryRoom);

1. Now, let's try to see the mailBoxNumber outside the function. On the last line of the program, write:

console.log(mailBoxNumber);

1. Nothing showed up in the console! That's right, and it means that JavaScript does not have access to the variable, since it is hidden away in the myApartment function.

To see the hidden variables inside the function, delete the console.log on mailBoxNumber in the global scope, and call the myApartment function instead.

Nice work! Now that we know how global and functional scope contain variables, let's organize some code to practice.

To the right, we provided you with incorrect sample code. If you try to run it as is, you will get an error because the console.log on line 20 references variables that are within the functional scope of myApartment.

Let's fix our scoping issues!

***Instructions***

*Write your code in the apartment.js file.*

1. Within the myApartment function, move the variables with apartment building information to the global scope, so that the console.log on line 20 can run.(apartment.js)

***Note:*** Do not edit the variables or their values.

This unit introduced you to scope.

* *Scope* is the idea in programming that some variables are acessible/inaccessible from other parts of the program.
* *Global* *Scope* refers to variables that are accessible to every part of the program.
* *Functional* *Scope* refers to variables created inside functions, which are not accessible outside of its block.

Keep up the awesome job in the next lesson where we'll learn how to make lists and how to program JavaScript to do repetitive tasks for us with loops!