1. Intro to Functions

That pizza did not look too stellar. Pizza reheat button is a function.

1. Function Example

Reversing a string as a function is easier than reversing each string.

1. Declaring Functions

function reheatPizza(numSlices) {

// code that figures out reheat settings!

}

// declares the sayHello function

function sayHello() {

var message = "Hello!"

return message; // returns value instead of printing it

}

A parameter is always going to be a variable name and appears in the function declaration. On the other hand, an argument is always going to be a value (i.e. any of the JavaScript data types - a number, a string, a boolean, etc.)

1. Function Recap

Functions package up code so you can easily use (and reuse) a block of code. Parameters are variables that are used to store the data that's passed into a function for the function to use. Arguments are the actual data that's passed into a function when it is invoked.

1. Quiz: Laugh it Off 1 (5-1)

function laugh() {

return "hahahahahahahahahaha!";

}

console.log(laugh());

1. Quiz: Laugh it Off 2 (5-2)

function laugh(num) {

laughString = "";

for (var index = 0; index < num; index++) {

laughString += "ha";

}

laughString += "!";

return laughString;

}

console.log(laugh());

1. Return Values

Console.log used to print a value to the JS console. Undefined is being returned back from the console.log function. Return used to stop the execution of a function and return a value back to the caller.

1. Using Return Values

A function's return value can be stored in a variable or reused throughout your program as a function argument.

1. Scope

James will find more books in the library than in the halls.

1. Scope Example

Global and function scope.

1. Shadowing

Scope overriding or shadowing can reassign values out of scope. Just use new variables in inner scopes.

1. Global Variables

Minimize global variable use when possible.

1. Scope Recap
2. If an identifier is declared in global scope, it's available everywhere.
3. If an identifier is declared in function scope, it's available in the function it was declared in (even in functions declared inside the function).
4. When trying to access an identifier, the JavaScript Engine will first look in the current function. If it doesn't find anything, it will continue to the next outer function to see if it can find the identifier there. It will keep doing this until it reaches the global scope.
5. Global identifiers are a bad idea. They can lead to bad variable names, conflicting variable names, and messy code.
6. Hoisting

Before any JS is executed, all function declarations are “hoisted” to the top of their current scope.

1. Hoisting Recap
2. JavaScript hoists function declarations and variable declarations to the top of the current scope.
3. Variable assignments are not hoisted.
4. Declare functions and variables at the top of your scripts, so the syntax and behavior are consistent with each other.
5. Quiz: Build a Triangle (5-3)

function buildTriangle(length) {

triangle = "";

for (var index = 1; index <= length; index++) {

triangle += makeLine(index);

}

return triangle;

}

1. Function Expressions

You can store a function as a variable.

var catSays = function(max) {

var catMessage = "";

for (var i = 0; i < max; i++) {

catMessage += "meow ";

}

return catMessage;

};

It's an anonymous function, a function with no name, and you've stored it in a variable called catSays.

1. Patterns with Function Expressions

Being able to store a function in a variable makes it really simple to pass the function into another function. A function that is passed into another function is called a callback.

Anonymous inline function expressions are often used with function callbacks that are probably not going to be reused elsewhere. Yes, you could store the function in a variable, give it a name, and pass it in like you saw in the examples above. However, when you know the function is not going to be reused, it could save you many lines of code to just define it inline.

1. Function Expression Recap

You can even pass a function into another function inline. This pattern is commonly used in JavaScript, and can be helpful streamlining your code.

// function declaration that takes in two arguments: a function for displaying

// a message, along with a name of a movie

function movies(messageFunction, name) {

messageFunction(name);

}

// call the movies function, pass in the function and name of movie

movies(function displayFavorite(movieName) {

console.log("My favorite movie is " + movieName);

}, "Finding Nemo");

1. Quiz: Laugh (5-4)

var laugh = function(num) {

var laughs = "";

for (var i = 0; i < num; i++) {

laughs += "ha";

}

laughs += "!";

return laughs;

};

console.log(laugh(10));

1. Quiz: Cry (5-5)

var cry = function cry() {

return "boohoo!";

};

console.log(cry());

1. Quiz: Inline (5-6)

var laugh = function(num) {

var laughs = "";

for (var i = 0; i < num; i++) {

laughs += "ha";

}

laughs += "!";

return laughs;

};

emotions("happy", function(num) {

return (laugh(num));

});

1. Lesson 5 Summary

Grats! We made it! Arrays are next.