Introduction

Using **functions** to give a name to a block of statements so that you can execute them whenever you want to, adds incredible power to your programming. You can also pass values to a function using function arguments.

Passing Arguments to Functions

Function arguments let you take a value that you want to use inside of that function, and pass it in so that you can manipulate it. Variables to be used by a function can be passed as arguments between the parentheses that you use when you're calling and executing a function.

```
var color = "red";

function show(item) {
  console.log(item);
}

show(color);
// "red"
```

When you define a function, you don't define it around the specific variables that exist. You define it in the abstract, using it's own internal set of variables.

What we've done is we've created a function that's looking for a variable, and then it's going to perform a `console.log` on that variable. This function uses the name, `item` for the parameter that we're passing in. It executes it's statements on the local `item` variable. And as a result, when we do a `showItem`, and we pass in the variable `color`, it treats `color` as `item`. Knowing how to pass arguments to a function,makes functions even more powerful.

Pass Multiple Named Arguments

You can pass multiple named arguments to any function. The arguments that you pass to a function can be of any variable type. You refer to them by their parameter name when you're writing the definition of the function. When you're writing the statements that go in the code block that the function executes.

```
var word = "hello";
var list = ["dog", "cat", "bird"];

greeter(word, list); // arguments
// "hello dog"
// "hello cat"
// "hello bird"

function greeter(str, arr) { // parameters
  var counter;
  for (counter = 0; counter < arr.length; counter++) {
    console.log(str + " " + arr[counter]);
  }
}
```

It's worth experimenting a little bit with some of these just so that you get familiar with the concept that the arguments that you're creating when you

define greater function, and the parameters you're passing in when you call the greater function, relate to each other directly.

Design and Name for Reuse

When creating a function, it's important to design and name your function so that it can be reusable in multiple contexts. Ideally a function should be very flexible. It should be reusable, and it should be independent of a specific context. It should be able to be picked up and used in different places, conveniently. Name functions so that they can support flexible applications.

```
var greeting = "hello";
var animals = ["dog", "cat", "bird"];
var salutation = "goodbye";
var target = ["moon", "sun"];

addresser(greeting, animals);
// "hello dog"
// "hello cat"
// "hello bird"

addresser(salutation, target);
// "goodbye moon"
// "goodbye sun"

addresser(greeting, target);
// "hello moon"
// "hello sun"

function addresser(str, arr) {
  var counter;
  for (counter = 0; counter < arr.length; counter++) {
    console.log(str + " " + arr[counter]);
  }
}
```

Functions Can Call Functions

Functions can also call other functions, and this is convenient because you can treat your functions just like native functions in JavaScript, and use them just like any other JavaScript keyword. Imagine we wanted a JavaScript function that could take a string with multiple words in it, and return it to us in title case. That would be each word in the string, with the first letter capitalized. And otherwise the string remains the same.

We have to do a couple of things to that string. The first thing we'd have to do is split the string into separate substrings for each word. Then we'd have to go through each word, capitalize the first letter of that word and then join the string back together with spaces, and return the value. Now, we've seen how to do all of those different pieces but with functions, we can actually make useful little tools that will help us make that more efficient.

```
console.log(titleCase("hello world."));
// "Hello World."

function titleCase(str) {
  var strArray = str.split(" ");
  var counter;
  for (counter = 0; counter < strArray.length; counter++) {
    strArray[counter] = capitalize(strArray[counter]);
  }
  return strArray.join(" ");
}

function capitalize(str) {
  var result = [];
  result[0] = str.charAt(0).toUpperCase();
  result[1] = str.substring(1);
  return (result.join(""));
}
```

What we've shown here is that we can create small modular functions that perform very specific tasks named appropriately. We can use one function to call another function, and get a more complex result out of much simpler modular pieces.