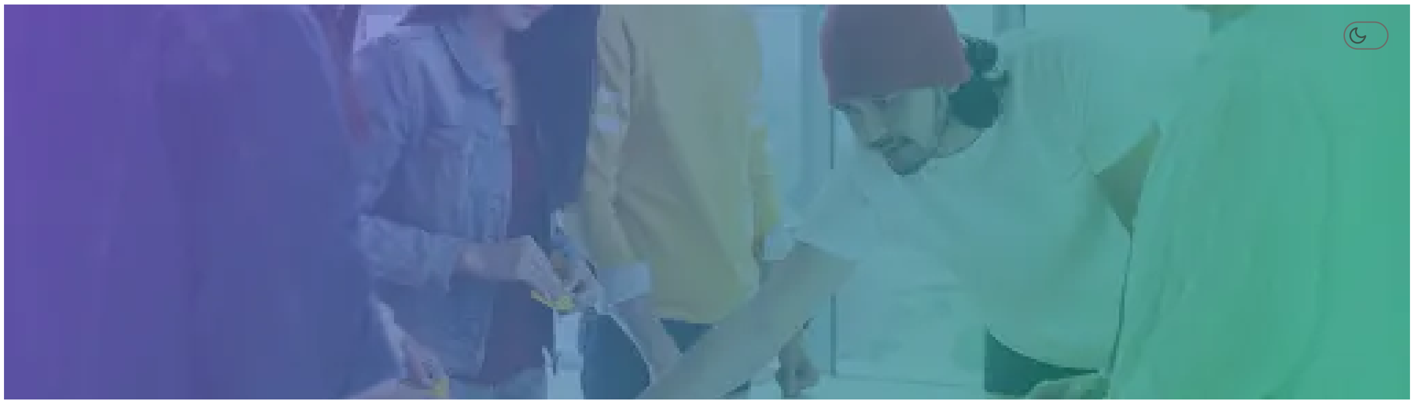
Springboard



JS Functions

functions-intro-demo.zip 6574.9KB

Goals

- Describe what functions are, and why they're useful
- Explain the syntax for creating functions
- Explain what the return keyword does
- Define functions that accept parameters
- Explain how scope works in JavaScript

Functions

What Is A Function?

- A way to bundle up code we may want to use over and over
- Accepts inputs and can return an output
- A helpful way to structure our code

Why Should I Care?

- Functions help you avoid duplication in your code
- You can name your functions to make your code more readable

Function Structure In Javascript

```
function functionName() { // here is the function body }
```

- function keyword
- name of function
- body of function

Our First Function

```
function soExcited() { console.log("OMG FUNCTIONZ WOW!!!!!!!"); } soExcited();
```

Saving The Message

```
function soExcited() { console.log("OMG FUNCTIONZ WOW!!!!!!!"); } let message =
soExcited(); message; // undefined :(
```

Saving The Message Now?

```
function soExcited() { "OMG FUNCTIONZ WOW!!!!!!!"; } soExcited(); // undefined - again! :
(
```

How can we get access to the message inside of the function?

Accessing Values Inside Of Functions

```
function soExcited() { return "OMG FUNCTIONZ WOW!!!!!!!"; } soExcited(); // "OMG
FUNCTIONZ WOW!!!!!!!"
```

The Return Keyword

- Immediately exits out of the function
- Sets the output of the function equal to the value returned

Immediately exits out of the function

```
function eatFood() { return "nom nom nom"; console.log("This line will never run!") }
```

- Sets the output of the function equal to the value returned
- In JavaScript, every function returns a value.
- If no return statement is present, the function returns undefined.

```
function returnsAValue() { return "hi!"; } returnsAValue(); // "hi!" function
returnsAValue() { console.log("hi!"); } returnsAValue(); // logs "hi", // but returns
undefined
```

Function Parameters

- So far, all of our functions return the same thing every time.
- It would be nice if our functions could produce different outputs if we give them different inputs.
- But how can we give inputs to our functions?

Here’s another function!

```
function order(food) { return `I'll have the ${food}, please.`; } order("salad"); //
"I'll have the salad, please." order("pizza"); // "I'll have the pizza, please."
order("tacos"); // "I'll have the tacos, please."
```

food is a parameter to our function!

- Parameters represent values that will be passed to your function.
- You can think of parameters as variables that only your function knows about.
- Like with variables, you can call your parameters whatever you want.

Like with function names and variable names, think about what you want to name your parameters!

```
// pretty bad function favorite(firstVariableName, secondVariableName) { return `My
favorite ${firstVariableName} is the ${secondVariableName}.` }
```

```
// not much better function favorite(x, y) { return `My favorite ${x} is the ${y}.` }
```

```
// much better (notice the name of the function is more descriptive as well!) function
displayFavorite (typeOfThing, favoriteThing) { return `My favorite ${typeOfThing} is the
${favoriteThing}.` }
```

Parameters Vs Arguments

- You'll sometimes see the words parameter and argument used interchangeably. But there's a subtle difference.
- A parameter is a variable described in the definition of a function.
- An argument is the value of a parameter when a function is called.

```
function isItEven(number) { return number % 2 === 0; } isItEven(4); // true isItEven(7);
// false // number is a parameter, 4 and 7 are arguments when we invoke the function
```

Watch Out!

You only have access to arguments inside of a function. Outside of it, there's no automatic arguments variable!

```
function showMeTheArguments() { console.log(arguments); } showMeTheArguments(1,2,3); //
Arguments object console.log(arguments); // Uncaught ReferenceError: arguments is not
defined
```

Too Few Arguments

If you supply fewer arguments than the function expects, the values of the remaining parameters will be undefined.

```
function topThreeMusicians(musicianOne, musicianTwo, musicianThree) { return ` My top three favor
${musicianOne}, ${musicianTwo}, and ${musicianThree}.`; } topThreeMusicians("Miley Cyrus", "Hanso
three favorite musicians are Miley Cyrus, Hanson, and undefined."
```

Global Scope

By default, variables remain in the scope where they were originally declared. This is called lexical scope .

When a variable is declared outside of a function, it is in the global scope .

```
let hello = 'hello'; // I'm a variable on the global scope
```

Global variables can be used or modified anywhere in your program, so you usually want to avoid them and have local variables instead.

Function Scope

Functions create their own layer of scope (where variables are remembered/live), called function scope.

In general, inner scopes have access to variables in their parent scopes, but parents cannot access variables in child scopes.

From inside a function, you have access to all variables in the function scope, as well as all variables in any scope the function sits inside.

Scope Example

```
let count = 0; function counter() { let myName = "Colt"; count++; return `${myName} has
called the counter function ${count} times!` } counter(); // "Colt has called the counter
function 1 times!" counter(); // "Colt has called the counter function 2 times!" count;
// 2 myName; // Uncaught ReferenceError: myName is not defined
```