Learn web development

Function return values

Previous Overview: Building blocks Next

There's one last essential concept for us to discuss in this course, to close our look at functions — return values. Some functions don't return a significant value after completion, but others **do, and it's important to understand what their values are, how to make use of them in your code, and how to make your own custom functions return useful values. We'll cover all of these below.**

Prerequisites:	Basic computer literacy, a basic understanding of HTML and CSS, JavaScript first steps, Functions — reusable blocks of code.
Objective:	To understand function return values, and how to make use of them.

What are return values?

Return values are just what they sound like — values returned by the function when it completes. You've already met return values a number of times, although you may not have thought about them explicitly. Let's return to some familiar code:

```
var myText = 'I am a string';
var newString = myText.replace('string', 'sausage');
console.log(newString);
// the replace() string function takes a string,
// replaces one substring with another, and returns
// a new string with the replacement made
```

We saw exactly this block of code in our first function article. We are invoking the replace() function on the `myText` string, and passing it two parameters — the substring to find, and the substring to replace it with. When this function completes (finishes running), it returns a value, which is a new string with the replacement made. In the code above, we are saving this return value as the value of the `newString` variable.

If you look at the replace function MDN reference page, you'll see a section called Return value. It is very useful to know and understand what values are returned by functions, so we try to include this information wherever possible.

Some functions don't return a return value as such (in our reference pages, the return value is listed as `void` or `undefined` in such cases). For example, in the displayMessage() function we built in the previous article, no specific value is returned as a result of the function being invoked. It just makes a box appear somewhere on the screen — that's it!

Generally, a return value is used where the function is an intermediate step in a calculation of some kind. You want to get to a final result, which involves some values. Those values need to be calculated by a function, which then returns the results so they can be used in the next stage of the calculation.

Using return values in your own functions

To return a value from a custom function, you need to use ... wait for it ... the `return` keyword. We saw this in action recently in our random-canvas-circles.html example. Our `draw()` function draws 100 random circles somewhere on an HTML `<canvas>`:

```
function draw() {
  ctx.clearRect(0,0,WIDTH,HEIGHT);
  for (var i = 0; i < 100; i++) {
    ctx.beginPath();
    ctx.fillStyle = 'rgba(255,0,0,0.5)';
    ctx.arc(random(WIDTH), random(HEIGHT), random(50), 0, 2 * Math.PI);
    ctx.fill();
  }
}
```

Inside each loop iteration, three calls are made to the `random()` function, to generate a random value for the current circle's x coordinate, y coordinate, and radius, respectively. The `random()` function takes one parameter — a whole number — and it returns a whole random number between 0 and that number. It looks like this:

```
function random(number) {
  return Math.floor(Math.random()*number);

```

```
    }
```

This could be written as follows:

```
function random(number) {
  var result = Math.floor(Math.random()*number);
  return result;
}
```

But the first version is quicker to write, and more compact.

We are returning the result of the calculation `Math.floor(Math.random()*number)` each time the function is called. This return value appears at the point the function was called, and the code continues. So for example, if we ran the following line:

```
ctx.arc(random(WIDTH), random(HEIGHT), random(50), 0, 2 * Math.PI);
```

and the three `random()` calls returned the values 500, 200, and 35, respectively, the line would actually be run as if it were this:

```
ctx.arc(500, 200, 35, 0, 2 * Math.PI);
```

The function calls on the line are run first and their return values substituted for the function calls, before the line itself is then executed.

Active learning: our own return value function

Let's have a go at writing our own functions featuring return values.

1. First of all, make a local copy of the function-library.html file from GitHub. This is a simple HTML page containing a text `<input>` field and a paragraph. There's also a `<script>` element in which we have stored a reference to both HTML elements in two variables. This little page will allow you to enter a number into the text box, and display different numbers related to it in the paragraph below.
2. Let's add some useful functions to this `<script>` element. Below the existing two lines of JavaScript, add the following function definitions:

```
function squared(num) {
  return num * num;

```

```
}

function cubed(num) {
  return num * num * num;
}

function factorial(num) {
  var x = num;
  while (x > 1) {
    num *= x-1;
    x--;
  }
  return num;
}
```

The `squared()` and `cubed()` functions are fairly obvious — they return the square or cube of the number given as a parameter. The `factorial()` function returns the factorial of the given number.

3. Next we're going to include a way to print out information about the number entered into the text input. Enter the following event handler below the existing functions:

```
input.onchange = function() {
  var num = input.value;
  if (isNaN(num)) {
    para.textContent = 'You need to enter a number!';
  } else {
    para.textContent = num + ' squared is ' + squared(num) + '. ' +
                       num + ' cubed is ' + cubed(num) + '. ' +
                       num + ' factorial is ' + factorial(num) + '.';
  }
}
```

Here we are creating an `onchange` event handler that runs whenever the change event fires on the text input — that is, when a new value is entered into the text input, and submitted (enter a value then press tab for example). When this anonymous function runs, the existing value entered into the input is stored in the `num` variable.

Next, we do a conditional test — if the entered value is not a number, we print an error message into the paragraph. The test looks at whether the expression `isNaN(num)` returns true. We use the isNaN() function to test whether the num value is not a number — if so, it returns `true`, and if not, `false`.

If the test returns `false`, the `num` value is a number, so we print out a sentence inside the

paragraph element stating what the square, cube, and factorial of the number are. The sentence calls the `squared()`, `cubed()`, and `factorial()` functions to get the required values.

4. Save your code, load it in a browser, and try it out.

> **Note**: If you have trouble getting the example to work, feel free to check your code against the finished version on GitHub (see it running live also), or ask us for help.

At this point, we'd like you to have a go at writing out a couple of functions of your own and adding them to the library. How about the square or cube root of the number, or the circumference of a circle with a radius of length `num`?

This exercise has brought up a couple of important points besides being a study on how to use the `return` statement. In addition, we have:

- Looked at another example of writing error handling into our functions. It is generally a good idea to check that any necessary parameters have been provided, and in the right datatype, and if they are optional, that some kind of default value is provided to allow for that. This way, your program will be less likely to throw errors.
- Thought about the idea of creating a function library. As you go further into your programming career, you'll start to do the same kinds of things over and over again. It is a good idea to start keeping your own library of utility functions that you use very often — you can then copy them over to your new code, or even just apply it to any HTML pages where you need it.

Conclusion

So there we have it — functions are fun, very useful and, although there's a lot to talk about in regards to their syntax and functionality, fairly understandable given the right articles to study.

If there is anything you didn't understand, feel free to read through the article again, or contact us to ask for help.

See also

- Functions in-depth — a detailed guide covering more advanced functions-related information.
- Callback functions in JavaScript — a common JavaScript pattern is to pass a function into another function as an argument, which is then called inside the first function. This is a little beyond the scope of this course, but worth studying before too long.

← Previous ↑ Overview: Building blocks Next →

Was this article helpful?