

Learning the Web

JavaScript basics

[« Previous](#)[Next »](#)

JavaScript is a programming language that adds interactivity to your website (for example: games, responses when buttons are pressed or data entered in forms, dynamic styling, animation). This article helps you get started with this exciting language and gives you an idea of what is possible.

What is JavaScript, really?

JavaScript ("JS" for short) is a full-fledged dynamic programming language that, when applied to an HTML document, can provide dynamic interactivity on websites. It was invented by Brendan Eich, co-founder of the Mozilla project, the Mozilla Foundation, and the Mozilla Corporation.

You can do pretty much anything with JavaScript. You'll start small with simple features such as carousels, image galleries, fluctuating layouts, and responses to button clicks. Eventually as you get more experienced with the language, you'll be able to create games, animated 2D and 3D graphics, full blown database-driven apps, and more!

JavaScript itself is fairly compact but very flexible, and developers have written a lot of tools on top of the core JavaScript language, to unlock a huge amount of extra functionality with very little effort. These include:

- Application Programming Interfaces (APIs) built into web browsers, providing various functionality like dynamically creating HTML and setting CSS styles, grabbing and manipulating a video stream from the user's webcam, or generating 3D graphics and audio samples.
- Third-party APIs to allow developers to incorporate functionality in their sites from other properties, such as Twitter or Facebook.

- Third-party frameworks and libraries you can apply to your HTML to allow you to rapidly build up sites and applications.

A "hello world" example

The above section sounds really exciting, and so it should — JavaScript is one of the most exciting Web technologies, and as you start to get good at using it, your websites will enter a new dimension of power and creativity.

However, JavaScript is a bit more complex to get comfortable with than HTML and CSS, and you'll have to start small, and keep working at it in tiny regular steps. To start off with, we'll show you how to add some really basic JavaScript to your page, to create a "hello world!" example ([the standard in basic programming examples](#)).

! Important: If you haven't been following along with the rest of our course, [download this example code](#) and use it as a starting point.

1. First, go to your test site and create a new file called `main.js`. Save it in your scripts folder.
2. Next, go to your `index.html` file and enter the following element on a new line just before the closing `</body>` tag:

```
1 | <script src="scripts/main.js"></script>
```

3. This is basically doing the same job as the `<link>` element for CSS — it applies the JavaScript to the page, so it can have an effect on the HTML (and CSS, and anything else on the page).
4. Now add the following code to the `main.js` file:

```
1 | var myHeading = document.querySelector('h1');  
2 | myHeading.innerHTML = 'Hello world!';
```

5. Now make sure the HTML and JavaScript files are saved, and load `index.html` in the browser. You should see something like the following:



Note: The reason we've put the `<script>` element near the bottom of the HTML file is that HTML is loaded by the browser in the order it appears in the file. If the JavaScript is loaded first and it is supposed to affect the HTML below it, it might not work, as the JavaScript would be loaded before the HTML it is supposed to work on. Therefore, putting JavaScript near the bottom of the HTML page is often the best strategy.

What happened?

So, your heading text has been changed to "Hello world!" using JavaScript. We did this by first using a function called `querySelector()` to grab a reference to our heading, and store it in a variable called `myHeading`. This is very similar to what we did in CSS using selectors. When you want to do something to an element, first you need to select it.

After that, we set the value of the `myHeading` variable's `innerHTML` property (which represents the content of the heading) to "Hello world!".

Language basics crash course




Let's explain just some of the basic features of the JavaScript language, to give you some more understanding of how it all works. Better yet, these features are common to all programming languages. If you can understand these fundamentals, you should be able to start programming just about anything!

Important: In this article, try entering the example code lines into your JavaScript console to see what happens. For more details on JavaScript consoles, see [Discover browser developer tools](#).

Variables

Variables are containers that you can store values in. You start by declaring a variable with the `var` keyword, followed by any name you want to call it:

```
1 | var myVariable;
```

-  **Note:** All statements in JavaScript must end with a semi-colon, to indicate that this is where the statement ends. If you don't include these, you can get unexpected results.
-  **Note:** You can name a variable nearly anything, but there are some name restrictions (see [this article on variable naming rules](#).)
-  **Note:** JavaScript is case sensitive — `myVariable` is a different variable to `myvariable`. If you are getting problems in your code, check the casing!

After declaring a variable, you can give it a value:

```
1 | myVariable = 'Bob';
```

You can do both these operations on the same line if you wish:

```
1 | var myVariable = 'Bob';
```

You can retrieve the value by just calling the variable by name:

```
1 | myVariable;
```

After giving a variable a value, you can later choose to change it:

```
1 | var myVariable = 'Bob';  
2 | myVariable = 'Steve';
```

Note that variables have different [data types](#):

Variable	Explanation	Example
String	A string of text. To signify that the variable is a string, you should enclose it in quote marks.	<code>var myVariable = 'Bob';</code>
Number	A number. Numbers don't have quotes around them.	<code>var myVariable = 10;</code>
	A True/False value. The words <code>true</code> and <code>false</code> are used.	

Boolean	false are special keywords in JS, and don't need quotes.	<code>var myVariable = true;</code>
Array	A structure that allows you to store multiple values in one single reference.	<code>var myVariable =</code> <code>[1, 'Bob', 'Steve', 10];</code> Refer to each member of the array like this: <code>myVariable[0], myVariable[1],</code> etc.
Object	Basically, anything. Everything in JavaScript is an object, and can be stored in a variable. Keep this in mind as you learn.	<code>var myVariable =</code> <code>document.querySelector('h1');</code> All of the above examples too.

So why do we need variables? Well, variables are needed to do anything interesting in programming. If values couldn't change, then you couldn't do anything dynamic, like personalize a greeting message or change the image displayed in an image gallery.

Comments

You can put comments into JavaScript code, just like you can in CSS:

```

1  /*
2  Everything in between is a comment.
3  */

```

If your comment contains no line breaks, it's often easier to put it behind two slashes like this:

```

1  // This is a comment

```

Operators

An operator is a mathematical symbol that produces a result based on two values (or variables). In the following table you can see some of the simplest operators, along with some examples to try out in the JavaScript console.


Operator	Explanation	Symbol(s)	Example
add/concatenation	Used to add two numbers together, or glue two strings together.	+	<code>6 + 9;</code> <code>"Hello " +</code> <code>"world!";</code>

subtract, multiply, divide	These do what you'd expect them to do in basic math.	-, *, /	<pre> 9 - 3; 8 * 2; // multiply in JS is an asterisk 9 / 3; </pre>
assignment operator	You've seen this already: it assigns a value to a variable.	=	<pre> var myVariable = 'Bob'; </pre>
Identity operator	Does a test to see if two values are equal to one another, and returns a true/false (Boolean) result.	===	<pre> var myVariable = 3; myVariable === 4; </pre>
Negation, not equal	Returns the logically opposite value of what it precedes; it turns a true into a false, etc. When it is used alongside the Equality operator, the negation operator tests whether two values are <i>not</i> equal.	!, !==	<p>The basic expression is true, but the comparison returns false because we've negated it:</p> <pre> var myVariable = 3; ! (myVariable === 3); </pre> <p>Here we are testing "is myVariable NOT equal to 3". This returns false because</p>

```
myVariable  
IS equal to 3.
```

```
var  
myVariable  
= 3;  
myVariable  
!== 3;
```

There are a lot more operators to explore, but this will do for now. See [Expressions and operators](#) for a complete list.

 **Note:** Mixing data types can lead to some strange results when performing calculations, so be careful that you are referring to your variables correctly, and getting the results you expect. For example, enter "35" + "25" into your console. Why don't you get the result you expected? Because the quote marks turn the numbers into strings, so you've ended up concatenating strings rather than adding numbers. If you enter, 35 + 25 you'll get the correct result.

Conditionals

Conditionals are code structures that allow you to test whether an expression returns true or not, and then run different code depending on the result. The most common form of conditional is called `if ... else`. So for example:

```
1 | var iceCream = 'chocolate';  
2 | if (iceCream === 'chocolate') {  
3 |     alert('Yay, I love chocolate ice cream!');  
4 | } else {  
5 |     alert('Awww, but chocolate is my favorite...');  
6 | }
```

The expression inside the `if (...)` is the test — this uses the identity operator (as described above) to compare the variable `iceCream` with the string `chocolate` to see if the two are equal. If this comparison returns `true`, run the first block of code. If not, skip that code and run the second block of code after the `else` statement.

Functions

Functions are a way of packaging functionality that you want to reuse, so that whenever you want

the functionality you can call the function with the function name rather than constantly rewriting the entire code. You have already seen some uses of functions above, for example:

```
1 | var myVariable = document.querySelector('h1');
```

```
1 | alert('hello!');
```

These functions, `document.querySelector` and `alert`, are built into the browser for you to use whenever you like.

If you see something that looks like a variable name, but has brackets — `()` — after it, it is probably a function. Functions often take arguments — bits of data they need to do their job. These go inside the brackets, separated by commas if there is more than one item.

For example, the `alert()` function makes a pop-up box appear inside the browser window, but we need to give it a string as an argument to tell the function what to write in the pop-up box.

The good news is that you can define your own functions — in this next example we write a simple function that takes two numbers as arguments and multiplies them:

```
1 | function multiply(num1,num2) {  
2 |     var result = num1 * num2;  
3 |     return result;  
4 | }
```

Try running the above function in the console, then try using your new function a few times, e.g.:

```
1 | multiply(4,7);  
2 | multiply(20,20);  
3 | multiply(0.5,3);
```



Note: The `return` statement tells the browser to return the `result` variable out of the function so it is available to use. This is necessary because variables defined inside functions are only available inside those functions. This is called variable scoping. (Read [more on variable scoping](#).)

Events

To create real interactivity on a website, you need events — these are code structures that listen out for things happening to the browser, and then allow you to run code in response to those things. The most obvious example is the [click event](#), which is fired by the browser when the mouse clicks on something. To demonstrate this, try entering the following into your console, then clicking on the current webpage:

```
1 | document.querySelector('html').onclick = function() {  
2 |     alert('Ouch! Stop poking me!');  
3 | }
```

There are many ways to attach an event to an element. Here we are selecting the HTML element and setting its `onclick` handler property equal to an anonymous (i.e. nameless) function that contains the code we want to run when the click event occurs.

Note that

```
1 | document.querySelector('html').onclick = function() {};
```

is equivalent to

```
1 | var myHTML = document.querySelector('html');  
2 | myHTML.onclick = function() {};
```

It's just shorter.

Supercharging our example website

Now that we've gone over a few JavaScript basics, let's add a few cool basic features to our example site to give you some first idea of what is possible.

Adding an image changer

In this section, we'll add another image to our site, and add some simple JavaScript to change between the two when the image is clicked on.

1. First of all, find another image that you'd like to feature on your site. Make sure it is the same size as your first image, or as close as possible.
2. Save the image in your `images` folder.

3. Go to your `main.js` file, and enter the following JavaScript. (If your "hello world" JavaScript is still there, delete it.)

```
1 | var myImage = document.querySelector('img');
2 |
3 | myImage.onclick = function() {
4 |     var mySrc = myImage.getAttribute('src');
5 |     if(mySrc === 'images/firefox-icon.png') {
6 |         myImage.setAttribute ('src','images/firefox2.png');
7 |     } else {
8 |         myImage.setAttribute ('src','images/firefox-icon.png');
9 |     }
10 | }
```

4. Save all files and load `index.html` in the browser. Now when you click the image, it should change to the other one!

So here, we are storing a reference to our image element in the `myImage` variable. Next, we make this variable's `onclick` event handler property equal to a function with no name (an "anonymous" function). Now, every time this image element is clicked:

1. We retrieve the value of the image's `src` attribute.
2. We use a conditional to check whether the `src` value is equal to the path to the original image:
 1. If it is, we change the `src` value to the path to the 2nd image, forcing the other image to be loaded inside the `<image>` element.
 2. If it isn't (meaning it must already have changed), we change the `src` value back to the original image path, to flip it back to how it was originally.

Adding a personalized welcome message

Next we will add another bit of code, to change the page's title to include a personalized welcome message when the user first navigates to the site. This welcome message will persist when the user leaves the site and then comes back. We will also include an option to change the user and, therefore, the welcome message anytime it is required.

1. In `index.html`, add the following line just before the `<script>` element:

```
1 | <button>Change user</button>
```

2. In `main.js`, add the following code at the bottom of the file, exactly as written — this grabs

references to the new button we added and the heading, and stores them in variables:

```
1 | var myButton = document.querySelector('button');  
2 | var myHeading = document.querySelector('h1');
```

3. Now add the following function to set the personalized greeting — this won't do anything yet, but we'll use it later on:

```
1 | function setUsername() {  
2 |     var myName = prompt('Please enter your name.');
```

```
3 |     localStorage.setItem('name', myName);  
4 |     myHeading.innerHTML = 'Mozilla is cool, ' + myName;  
5 | }
```

This function contains a `prompt()` function, which brings up a dialog box, a bit like `alert()` except that `prompt()` asks the user to enter some data, and stores that data in a variable after the user presses **OK**. In this case, we are asking the user to enter their name. Next, we call on an API called `localStorage`, which allows us to store data in the browser and retrieve it later on. We use `localStorage`'s `setItem()` function to create and store a data item called `'name'`, and set its value to the `myName` variable that contains the name the user entered. Finally, we set the `innerHTML` of the heading to a string, plus the user's name.

4. Next, add this `if ... else` block — we could call this the initialization code, as it sets up the app when it first loads:

```
1 | if(!localStorage.getItem('name')) {  
2 |     setUsername();  
3 | } else {  
4 |     var storedName = localStorage.getItem('name');
```

```
5 |     myHeading.innerHTML = 'Mozilla is cool, ' + storedName;  
6 | }
```

This block first uses the negation operator (logical NOT) to check whether the name data item exists. If not, the `setUsername()` function is run to create it. If so (that is, the user set it during a previous visit), we retrieve the stored name using `getItem()` and set the `innerHTML` of the heading to a string, plus the user's name, the same as we did inside `setUsername()`.

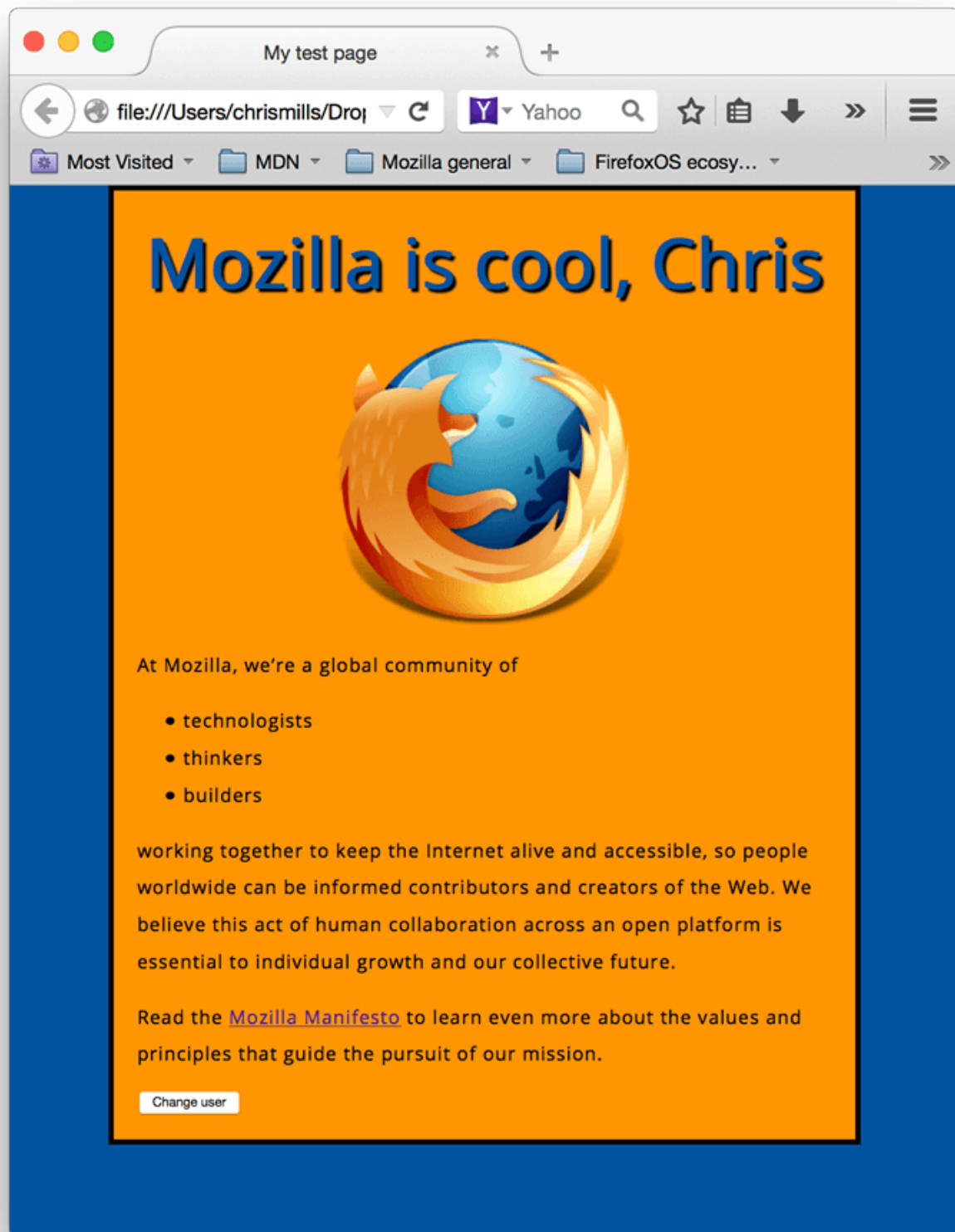
5. Finally, put the below `onclick` event handler on the button, so that when clicked the `setUsername()` function is run. This allows the user to set a new name whenever they want by pressing the button:

```
1 | myButton.onclick = function() {  
2 |   setUsername();  
3 | }
```

Now when you first visit the site, it'll ask you for your username then give you a personalized message. You can then change the name any time you like by pressing the button. As an added bonus, because the name is stored inside `localStorage`, it persists after the site is closed down, so the personalized message will still be there when you open the site up again!

Conclusion

If you have followed all the instructions in this article, you should end up with a page that looks something like this (you can also [view our version here](#)):



If you get stuck, you can always compare your work with our [finished example code on Github](#).

Here, we have only really scratched the surface of JavaScript. If you have enjoyed learning about it and want to go deeper with your studies, go to our [JavaScript Guide](#).

