Learn web development

JavaScript basics

Previous Overview: Getting started with the web 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.

JavaScript is incredibly versatile. You can start small, with carousels, image galleries, fluctuating layouts, and responses to button clicks. With more experience you'll be able to create games, animated 2D and 3D graphics, comprehensive database-driven apps, and much more!

JavaScript itself is fairly compact yet very flexible. Developers have written a large variety of tools on top of the core JavaScript language, unlocking a vast amount of extra functionality with minimum effort. These include:

- Browser Application Programming Interfaces (APIs) — APIs built into web browsers, providing functionality like dynamically creating HTML and setting CSS styles, collecting 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 content providers, 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.

Because this article is only supposed to be a light introduction to JavaScript, we are not going to confuse you at this stage by talking in detail about what the difference is between the core JavaScript language and the different tools listed above. You can learn all that in detail later on, in our JavaScript learning area, and in the rest of MDN.

Below we will introduce you some aspects of the core language, and you'll also play with a few browser API features too. Have fun!

A "hello world" example

The above section might sound really exciting, and so it should — JavaScript is one of the most lively 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 little more complex to become comfortable with than HTML and CSS. You may have to start small, and keep working in small consistent steps. To start, we'll show how to add some basic JavaScript to your page, creating 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 folder named 'scripts' (without the quotes). Then, within the new scripts folder you just created, create a new file called `main.js`. Save it in your `scripts` folder.
2. Next, in your `index.html` file enter the following element on a new line just before the closing `</body>` tag:

```
<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 (along with the CSS, and anything else on the page).
4. Now add the following code to the `main.js` file:

```
var myHeading = document.querySelector('h1');
myHeading.textContent = 'Hello world!';
```

5. Finally, 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?

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

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

Note: Both of the features you used above are parts of the Document Object Model (DOM) API, which allows you to manipulate documents.

Language basics crash course

Let's explain some of the core features of the JavaScript language, to give you greater understanding of how it all works. Better yet, these features are common to all programming languages. If you master these fundamentals, you're on your way to being able to program 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: A semi-colon at the end of a line indicates where a statement ends; it is only absolutely required when you need to separate statements on a single line, but it is a good practice to put them in at the end of each statement. There are other rules for when you should and shouldn't use them — see Your Guide to Semicolons in JavaScript for more details.

Note: You can name a variable nearly anything, but there are some name restrictions (see this article on variable naming rules.) If you are unsure, you can check your variable name to see if it is valid.

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:

```
myVariable = 'Bob';
```

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

```
var myVariable = 'Bob';
```

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

```
myVariable;
```

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

```
var myVariable = 'Bob';
myVariable = 'Steve';
```

Note that variables have different data types:

Variable	Explanation	Example
String	A sequence of text known as a string. To signify that the variable is a string, you should enclose it in quote marks.	`var myVariable = 'Bob';`
Number	A number. Numbers don't have quotes around them.	`var myVariable = 10;`

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

```
/*
Everything in between is a comment.
*/
```

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

```
// This is a comment
```

Operators

An `operator` is a mathematical symbol which 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.	+	`6 + 9;` `"Hello " + "world!";`

Operator	Explanation	Symbol(s)	Example

subtract, multiply, divide	These do what you'd expect them to do in basic math.	`-`, `*`, `/`	`9 - 3;`
			`8 * 2; // multiply in JS is an asterisk`
			`9 / 3;`

assignment operator	You've seen this already: it assigns a value to a variable.	`=`	`var myVariable = 'Bob';`
Identity operator	Does a test to see if two values are equal to one another, and returns a `true`/`false` (Boolean) result.	`===`	`var myVariable = 3;` `myVariable === 4;`

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 *not* equal.	`!`, `!==`	The basic expression is `true`, but the comparison returns `false` because we've negated it:
			`var myVariable = 3;`
			`!(myVariable === 3);`
			Here we are testing "is `myVariable` NOT equal to 3". This returns `false` because `myVariable` IS equal to 3.
			`var myVariable = 3;`
			`myVariable !== 3;`

There are a lot more operators to explore, but this is enough 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 which allow you to test if an expression returns true or not, running alternative code revealed by its result. The most common form of conditional is called `if ... else`.

So for example:

```
var iceCream = 'chocolate';
if (iceCream === 'chocolate') {
  alert('Yay, I love chocolate ice cream!');
} else {
  alert('Awwww, but chocolate is my favorite...');
}
```

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`, the first block of code is run. If the comparison is not true, the first block is skipped and the second code block, after the `else` statement, is run instead.

Functions

Functions are a way of packaging functionality that you wish to reuse. When you need the procedure you can call a function, with the function name, instead of rewriting the entire code each time. You have already seen some uses of functions above, for example:

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

2.
```
alert('hello!');
```

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

If you see something which looks like a variable name, but has brackets — ( ) — after it, it is likely 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 argument.

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

```
function multiply(num1,num2) {
  var result = num1 * num2;

```

```
    return result;
}
```

Try running the above function in the console, then test with several arguments. For example:

```
multiply(4,7);
multiply(20,20);
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

Real interactivity on a website needs events. These are code structures which listen for things happening in browser, running code in response. The most obvious example is the click event, which is fired by the browser when you click on something with your mouse. To demonstrate this, enter the following into your console, then click on the current webpage:

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

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

Note that

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

is equivalent to

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

It's just shorter.

Supercharging our example website

Now we've gone over a few JavaScript basics, let's add a few cool features to our example site to see what is possible.

Adding an image changer

In this section, we'll add an additional image to our site using some more DOM API features, using some JavaScript to switch between the two when the image is clicked.

1. First of all, find another image you'd like to feature on your site. Be sure it is the same size as the first image, or as close as possible.
2. Save this image in your `images` folder.
3. Rename this image 'firefox2.png' (without quotes).
4. Go to your `main.js` file, and enter the following JavaScript. (If your "hello world" JavaScript is still there, delete it.)

```
var myImage = document.querySelector('img');

myImage.onclick = function() {
    var mySrc = myImage.getAttribute('src');
    if(mySrc === 'images/firefox-icon.png') {
      myImage.setAttribute ('src','images/firefox2.png');
    } else {
      myImage.setAttribute ('src','images/firefox-icon.png');
    }
}
```

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

You store a reference to your image element in the `myImage` variable. Next, you 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. You retrieve the value of the image's `src` attribute.
2. You use a conditional to check whether the `src` value is equal to the path to the original image:
    1. If it is, you 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), the `src` value swaps back to the original image path, to the original state.

Adding a personalized welcome message

Next we will add another bit of code, changing the page's title to a personalized welcome message when the user first navigates to the site. This welcome message will persist, should the user leave the site and later return — we will save it using the Web Storage API. 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:

```
<button>Change user</button>
```

2. In `main.js`, place the following code at the bottom of the file, exactly as written — this takes references to the new button and the heading, storing them inside variables:

```
var myButton = document.querySelector('button');
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 fix this in a moment:

```
function setUserName() {
  var myName = prompt('Please enter your name.');
  localStorage.setItem('name', myName);
  myHeading.textContent = 'Mozilla is cool, ' + myName;
}
```

   This function contains a `prompt()` function, which brings up a dialog box, a bit like `alert()`. This `prompt()`, however, asks the user to enter some data, storing it 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 later retrieve it. We use localStorage's `setItem()` function to create and store a data item called `'name'`, setting its value to the `myName` variable which contains the data the user entered. Finally, we set the `textContent` of the heading to a string, plus the user's newly stored name.

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

```
if(!localStorage.getItem('name')) {
  setUserName();
} else {
  var storedName = localStorage.getItem('name');
  myHeading.textContent = 'Mozilla is cool, ' + storedName;
}
```

   This block first uses the negation operator (logical NOT, represented by the !) to check whether

the `name` data exists. If not, the `setUserName()` function is run to create it. If it exists (that is, the user set it during a previous visit), we retrieve the stored name using `getItem()` and set the `textContent` of the heading to a string, plus the user's name, as we did inside `setUserName()`.

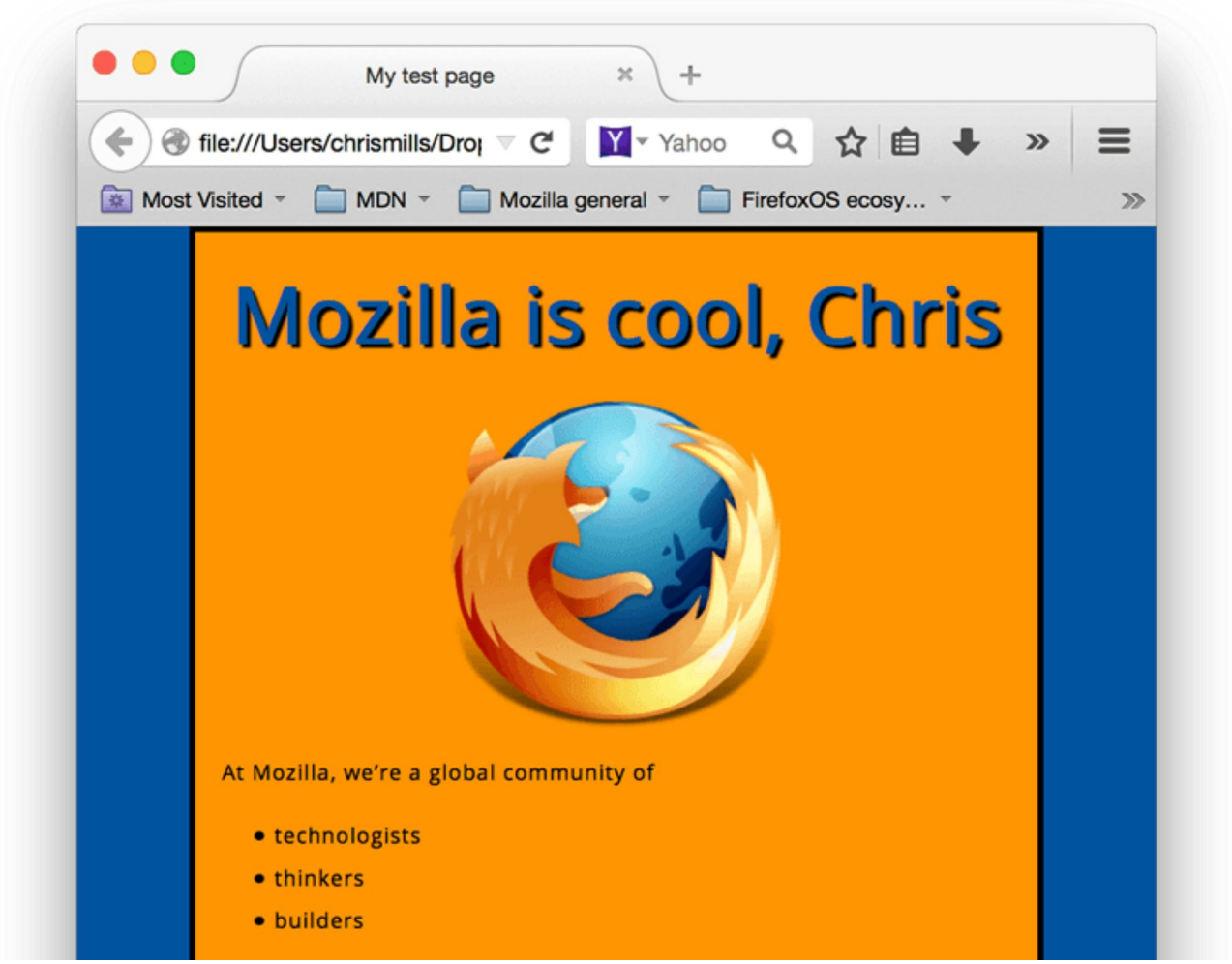
5. Finally, put the below `onclick` event handler on the button. When clicked, the `setUserName()` function is run. This allows the user to set a new name, when they wish, by pressing the button:

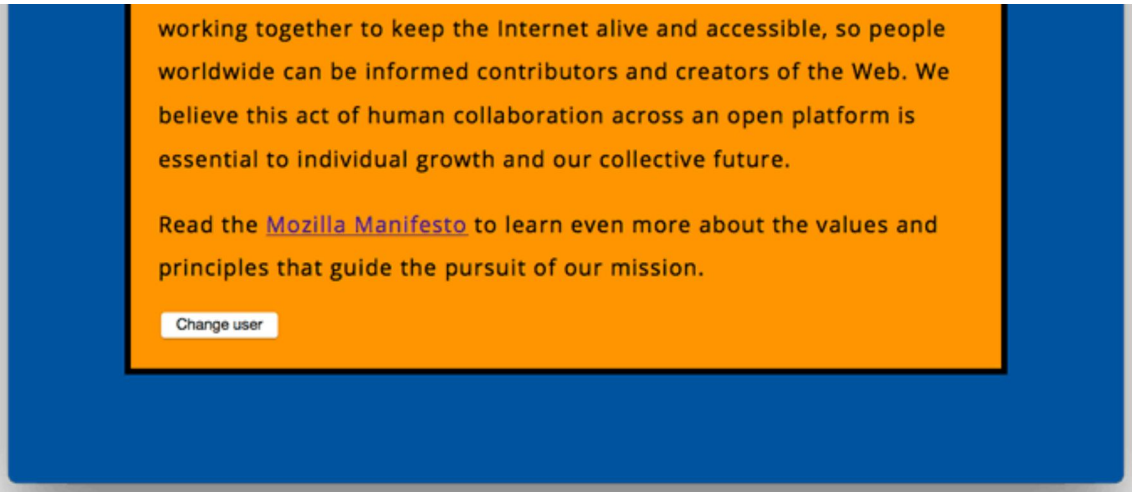
```
myButton.onclick = function() {
  setUserName();
}
```

Now when you first visit the site, it will ask you for your username, then give you a personalized message. You can 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, keeping the personalized message there when you next open the site!

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 compare your work with our finished example code on Github.

We have barely scratched the surface of JavaScript. If you have enjoyed playing, and wish to advance even further, head to our JavaScript learning topic.

Previous Overview: Getting started with the web Next