

# 28 HTML5 Features, Tips, and Techniques you Must Know

Jeffrey Way on Jul 14th 2011 with 420 Comments and 0 Reactions

## Tutorial Details

- **Topic:** HTML5
- **Estimated Difficulty:** Basix

This entry is part 2 of 14 in the [HTML5 and You](#) Session - [Show All](#)

[« Previous](#)[Next »](#)

This industry moves fast — really fast! If you're not careful, you'll be left in its dust. So, if you're feeling a bit overwhelmed with the coming changes/updates in HTML5, use this as a primer of the things you must know.

## 1. New Doctype

Still using that pesky, impossible-to-memorize XHTML doctype?

[view plaincopy to clipboardprint?](#)

1. `<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"`
2. `"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">`

If so, why? Switch to the new HTML5 doctype. You'll live longer — as Douglas Quaid might say.

[view plaincopy to clipboardprint?](#)

1. `<!DOCTYPE html>`

In fact, did you know that it truthfully isn't even really necessary for HTML5? However, it's used for current, and older browsers that require a specified doctype. Browsers that do not understand this doctype will simply render the contained markup in standards mode. So, without worry, feel free to throw caution to the wind, and embrace the new HTML5 doctype.

## 2. The Figure Element

Consider the following mark-up for an image:

[view plaincopy to clipboardprint?](#)

1. ``
2. `<p>Image of Mars. </p>`

There unfortunately isn't any easy or semantic way to associate the caption, wrapped in a paragraph tag, with the image element itself. HTML5 rectifies this, with the introduction of the `<figure>` element. When combined with the `<figcaption>` element, we can now semantically associate captions with their image counterparts.

[view plaincopy to clipboardprint?](#)

```
1. <figure>
2.   
3.   <figcaption>
4.     <p>This is an image of something interesting. </p>
5.   </figcaption>
6. </figure>
```

---

## 3. <small> Redefined

Not long ago, I utilized the `<small>` element to create subheadings that are closely related to the logo. It's a useful presentational element; however, now, that would be an incorrect usage. The `small` element has been redefined, more appropriately, to refer to small print. Imagine a copyright statement in the footer of your site; according to the new HTML5 definition of this element; the `<small>` would be the correct wrapper for this information.

*The `small` element now refers to "small print."*

---

## 4. No More Types for Scripts and Links

You possibly still add the `type` attribute to your `link` and `script` tags.

[view plaincopy to clipboardprint?](#)

```
1. <link rel="stylesheet" href="path/to/stylesheet.css" type="text/css" />
2. <script type="text/javascript" src="path/to/script.js"></script>
```

This is no longer necessary. It's implied that both of these tags refer to stylesheets and scripts, respectively. As such, we can remove the `type` attribute all together.

[view plaincopy to clipboardprint?](#)

```
1. <link rel="stylesheet" href="path/to/stylesheet.css" />
2. <script src="path/to/script.js"></script>
```

---

## 5. To Quote or Not to Quote.

...That is the question. Remember, HTML5 is not XHTML. You don't have to wrap your attributes in quotation marks if you don't want to. You don't have to close your elements. With that said, there's nothing wrong with doing so, if it makes you feel more comfortable. I find that this is true for myself.

[view plaincopy to clipboardprint?](#)

```
1. <p class=myClass id=someId> Start the reactor.
```

Make up your own mind on this one. If you prefer a more structured document, by all means, stick with the quotes.

---

## 6. Make your Content Editable

### To-Do List

- Break mechanical cab driver.
- Drive to abandoned factory
- Watc

The new browsers have a nifty new attribute that can be applied to elements, called `contenteditable`. As the name implies, this allows the user to edit any of the text contained within the element, including its children. There are a variety of uses for something like this, including an app as simple as a to-do list, which also takes advantage of local storage.

[view plaincopy to clipboardprint?](#)

```
1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4.   <meta charset="utf-8">
5.   <title>untitled</title>
6. </head>
7. <body>
8.   <h2> To-Do List </h2>
9.   <ul contenteditable="true">
10.    <li> Break mechanical cab driver. </li>
11.    <li> Drive to abandoned factory
12.    <li> Watch video of self </li>
13.  </ul>
14. </body>
15. </html>
```

Or, as we learned in the previous tip, we could write it as:

[view plaincopy to clipboardprint?](#)

```
1. <ul contenteditable=true>
```

---

## 7. Email Inputs

If we apply a `type` of “email” to form inputs, we can instruct the browser to *only* allow strings that conform to a valid email address structure. That’s right; built-in form validation will soon be here! We can’t 100% rely on this just yet, for obvious reasons. In older browsers that do not understand this “email” type, they’ll simply fall back to a regular textbox.

[view plaincopy to clipboardprint?](#)

```
1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4.   <meta charset="utf-8">
5.   <title>untitled</title>
6. </head>
```

```

7. <body>
8.   <form action="" method="get">
9.     <label for="email">Email:</label>
10.    <input id="email" name="email" type="email" />
11.    <button type="submit"> Submit Form </button>
12.  </form>
13. </body>
14. </html>

```

The screenshot shows a web form with a label 'Email:' followed by a text input field containing 'nettutsplus.com'. To the right of the input field is a button labeled 'Submit Form'. Below the input field, a red-bordered error message box displays the text 'nettutsplus.com is not a legal email address'.

*At this time, we cannot depend on browser validation. A server/client side solution must still be implemented.*

It should also be noted that all the current browsers are a bit wonky when it comes to what elements and attributes they do and don't support. For example, Opera seems to support email validation, just as long as the name attribute is specified. However, it does not support the placeholder attribute, which we'll learn about in the next tip. Bottom line, don't depend on this form of validation just yet...but you can still use it!

## 8. Placeholders

Before, we had to utilize a bit of JavaScript to create placeholders for textboxes. Sure, you can initially set the value attribute how you see fit, but, as soon as the user deletes that text and clicks away, the input will be left blank again. The placeholder attribute remedies this.

[view plaincopy to clipboardprint?](#)

```

1. <input name="email" type="email" placeholder="doug@givethesepeopleair.com" />

```

Again, support is shady at best across browsers, however, this will continue to improve with every new release. Besides, if the browser, like Firefox and Opera, don't currently support the placeholder attribute, no harm done.

## Chrome

Email:

## Firefox

Email:

## Opera

Email:

## Safari

Email:

---

## 9. Local Storage

Thanks to local storage (not officially HTML5, but grouped in for convenience's sake), we can make advanced browsers “remember” what we type, even after the browser is closed or is refreshed.

*[Subscribe to our YouTube page to watch all of the video tutorials!](#)*

*Or, watch this video on [Screenr.com](#).*

*“localStorage sets fields on the domain. Even when you close the browser, reopen it, and go back to the site, it remembers all fields in localStorage.”*

*-[QuirksBlog](#)*

While obviously not supported across all browsers, we can expect this method to work, most notably, in Internet Explorer 8, Safari 4, and Firefox 3.5. Note that, to compensate for older browsers that won't recognize local storage, you should first test to determine whether window.localStorage exists.

	MAC						
							
	CHROME	FIREFOX	OPERA	SAFARI	CHROME	FIREFOX	
	5	3.6	10.1	4	4	3.6	3
Local Storage	✓	✓	✗	✓	✓	✓	✗

via <http://www.findmebyip.com/litmus/>

## 10. The Semantic Header and Footer

Gone are the days of:

[view plaincopy to clipboardprint?](#)

1. `<div id="header">`
2. `...`
3. `</div>`
4. `<div id="footer">`
5. `...`
6. `</div>`

Divs, by nature, have no semantic structure — even after an `id` is applied. Now, with HTML5, we have access to the `<header>` and `<footer>` elements. The mark-up above can now be replaced with:

[view plaincopy to clipboardprint?](#)

1. `<header>`
2. `...`
3. `</header>`
4. `<footer>`
5. `...`
6. `</footer>`

*It's fully appropriate to have multiple headers and footers in your projects.*

Try not to confuse these elements with the “header” and “footer” of your website. They simply refer to their container. As such, it makes sense to place, for example, meta information at the bottom of a blog post within the `footer` element. The same holds true for the `header`.

## 11. More HTML5 Form Features

Learn about more helpful HTML5 form features in this quick video tip.

[Subscribe to our YouTube page to watch all of the video tutorials!](#)

---

## 12. Internet Explorer and HTML5

Unfortunately, that dang Internet Explorer requires a bit of wrangling in order to understand the new HTML5 elements.

*All elements, by default, have a display of inline.*

In order to ensure that the new HTML5 elements render correctly as block level elements, it's necessary at this time to style them as such.

[view plaincopy to clipboardprint?](#)

```
1. header, footer, article, section, nav, menu, hgroup {
2.     display: block;
3. }
```

Unfortunately, Internet Explorer will still ignore these stylings, because it has no clue what, as an example, the `header` element even is. Luckily, there is an easy fix:

[view plaincopy to clipboardprint?](#)

```
1. document.createElement("article");
2. document.createElement("footer");
3. document.createElement("header");
4. document.createElement("hgroup");
5. document.createElement("nav");
6. document.createElement("menu");
```

Strangely enough, this code seems to trigger Internet Explorer. To simplify this process for each new application, Remy Sharp [created a script](#), commonly referred to as the HTML5 shiv. This script also fixes some printing issues as well.

[view plaincopy to clipboardprint?](#)

```
1. <!--[if IE]>
2. <script src="http://html5shiv.googlecode.com/svn/trunk/html5.js"></script>
3. <![endif]-->
```

---

## 13. hgroup

Imagine that, in my site's header, I had the name of my site, immediately followed by a subheading. While we can use an `<h1>` and `<h2>` tag, respectively, to create the mark-up, there still wasn't, as of HTML4, an easy way to semantically illustrate the relationship between the two. Additionally, the use of an `h2` tag presents more problems, in terms of hierarchy, when it comes to displaying other headings on the page. By using the `hgroup` element, we can group these headings together, without affecting the flow of the document's outline.

[view plaincopy to clipboardprint?](#)

```
1. <header>
2.   <hgroup>
3.     <h1> Recall Fan Page </h1>
4.     <h2> Only for people who want the memory of a lifetime. </h2>
5.   </hgroup>
6. </header>
```

---

## 14. Required Attribute

Forms allow for a new `required` attribute, which specifies, naturally, whether a particular input is required. Dependent upon your coding preference, you can declare this attribute in one of two ways:

[view plaincopy to clipboardprint?](#)

1. `<input type="text" name="someInput" required>`

Or, with a more structured approach.

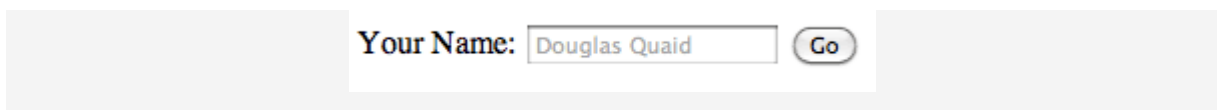
[view plaincopy to clipboardprint?](#)

1. `<input type="text" name="someInput" required="required">`

Either method will do. With this code, and within browsers that support this attribute, a form cannot be submitted if that “someInput” input is blank. Here’s a quick example; we’ll also add the placeholder attribute as well, as there’s no reason not to.

[view plaincopy to clipboardprint?](#)

1. `<form method="post" action="">`
2. `<label for="someInput"> Your Name: </label>`
3. `<input type="text" id="someInput" name="someInput" placeholder="Douglas Quaid" required>`
4. `<button type="submit">Go</button>`
5. `</form>`



The screenshot shows a web form rendered in a browser. It consists of a label "Your Name:" followed by a text input field. The input field contains the text "Douglas Quaid". To the right of the input field is a button labeled "Go". The entire form is enclosed in a light gray border.

If the input is left blank, and the form is submitted, the textbox will be highlighted.

---

## 15. Autofocus Attribute

Again, HTML5 removes the need for JavaScript solutions. If a particular input should be “selected,” or focused, by default, we can now utilize the `autofocus` attribute.

[view plaincopy to clipboardprint?](#)

1. `<input type="text" name="someInput" placeholder="Douglas Quaid" required autofocus>`

Interestingly enough, while I personally tend to prefer a more XHTML approach (using quotation marks, etc.), writing `"autofocus=autofocus"` feels odd. As such, we’ll stick with the single keyword approach.

---

## 16. Audio Support

No longer do we have to rely upon third party plugins in order to render audio. HTML5 now offers the `<audio>` element. Well, at least, ultimately, we won’t have to worry about these plugins. For the time



being, only the most recent of browsers offer support for HTML5 audio. At this time, it's still a good practice to offer some form of backward compatibility.

[view plaincopy to clipboardprint?](#)

1. `<audio autoplay="autoplay" controls="controls">`
2. `<source src="file.ogg" />`
3. `<source src="file.mp3" />`
4. `<a href="file.mp3">Download this file.</a>`
5. `</audio>`

Mozilla and Webkit don't fully get along just yet, when it comes to the audio format. Firefox will want to see an .ogg file, while Webkit browsers will work just fine with the common .mp3 extension. This means that, at least for now, you should create two versions of the audio.

When Safari loads the page, it won't recognize that .ogg format, and will skip it and move on to the mp3 version, accordingly. Please note that IE, per usual, doesn't support this, and Opera 10 and lower can only work with .wav files.

---

## 17. Video Support

Much like the `<audio>` element, we also, of course, have HTML5 video as well in the new browsers! In fact, just recently, [YouTube announced a new HTML5 video embed](#) for their videos, for browsers which support it. Unfortunately, again, because the HTML5 spec doesn't specify a specific codec for video, it's left to the browsers to decide. While Safari, and Internet Explorer 9 can be expected to support video in the H.264 format (which Flash players can play), Firefox and Opera are sticking with the open source Theora and Vorbis formats. As such, when displaying HTML5 video, you must offer both formats.

[view plaincopy to clipboardprint?](#)

1. `<video controls preload>`
2. `<source src="cohagenPhoneCall.ogv" type="video/ogg; codecs='vorbis, theora' />`
3. `<source src="cohagenPhoneCall.mp4" type="video/mp4; 'codecs='avc1.42E01E, mp4a.40.2'" />`
4. `<p> Your browser is old. <a href="cohagenPhoneCall.mp4">Download this video instead.</a> </p>`
5. `</video>`

*Chrome can correctly display video that is encoded in both the “ogg” and “mp4” formats.*

There are a few things worth noting here.

1. We aren't technically required to set the `type` attribute; however, if we don't, the browser has to figure out the type itself. Save some bandwidth, and declare it yourself.
2. Not all browsers understand HTML5 video. Below the `source` elements, we can either offer a download link, or embed a Flash version of the video instead. It's up to you.
3. The `controls` and `preload` attributes will be discussed in the next two tips.

---

## 18. Preload Videos

The `preload` attribute does exactly what you'd guess. Though, with that said, you should first decide whether or not you want the browser to preload the video. Is it necessary? Perhaps, if the visitor accesses a page, which is specifically made to display a video, you should definitely preload the video, and save the visitor a bit of waiting time. Videos can be preloaded by setting `preload="preload"`, or by simply adding `preload`. I prefer the latter solution; it's a bit less redundant.

[view plaincopy to clipboardprint?](#)

1. `<video preload>`

---

## 19. Display Controls

If you're working along with each of these tips and techniques, you might have noticed that, with the code above, the video above appears to be only an image, without any controls. To render these play controls, we must specify the `controls` attribute within the `video` element.

[view plaincopy to clipboardprint?](#)

1. `<video controls preload>`



Please note that each browser renders its player differently from one another.

---

## 20. Regular Expressions

How often have you found yourself writing some quickie regular expression to verify a particular textbox.

Thanks to the new `pattern` attribute, we can insert a regular expression directly into our markup.

[view plaincopy to clipboardprint?](#)

1. `<form action="" method="post">`
2. `<label for="username">Create a Username: </label>`
3. `<input type="text"`
4. `name="username"`
5. `id="username"`
6. `placeholder="4 <> 10"`

```
7.     pattern="[A-Za-z]{4,10}"
8.     autofocus
9.     required>
10.    <button type="submit">Go </button>
11. </form>
```

If you're moderately familiar with regular expressions, you'll be aware that this pattern: `[A-Za-z]{4,10}` accepts only upper and lowercase letters. This string must also have a minimum of four characters, and a maximum of ten.

Notice that we're beginning to combine all of these new awesome attributes!

*If regular expressions are foreign to you, [refer here](#).*

---

## 21. Detect Support for Attributes

What good are these attributes if we have no way of determining whether the browser recognizes them? Well, good point; but there are several ways to figure this out. We'll discuss two. The first option is to utilize the excellent [Modernizr](#) library. Alternatively, we can create and dissect these elements to determine what the browsers are capable of. For instance, in our previous example, if we want to determine if the browser can implement the `pattern` attribute, we could add a bit of JavaScript to our page:

[view plaincopy to clipboardprint?](#)

```
1. alert( 'pattern' in document.createElement('input') ) // boolean;
```

In fact, this is a popular method of determining browser compatibility. The jQuery library utilizes this trick. Above, we're creating a new `input` element, and determining whether the `pattern` attribute is recognized within. If it is, the browser supports this functionality. Otherwise, it of course does not.

[view plaincopy to clipboardprint?](#)

```
1. <script>
2.   if (!'pattern' in document.createElement('input')) {
3.     // do client/server side validation
4.   }
5. </script>
```

*Keep in mind that this relies on JavaScript!*

---

## 22. Mark Element

Think of the `<mark>` element as a highlighter. A string wrapped within this tag should be relevant to the current actions of the user. For example, if I searched for "Open your Mind" on some blog, I could then utilize some JavaScript to wrap each occurrence of this string within `<mark>` tags.

[view plaincopy to clipboardprint?](#)

```
1. <h3> Search Results </h3>
2. <p> They were interrupted, just after Quato said, <mark>"Open your Mind"</mark>. </p>
```

---

## 23. When to Use a <div>

Some of us initially questioned when we should use plain-ole `div`s. Now that we have access to `headers`, `articles`, `sections`, and `footers`, is there ever a time to use a...`div`? Absolutely. *Divs should be utilized when there's no better element for the job.*

For example, if you find that you need to wrap a block of code within a wrapper element specifically for the purpose of positioning the content, a `<div>` makes perfect sense. However, if you're instead wrapping a new blog post, or, perhaps, a list of links in your footer, consider using the `<article>` and `<nav>` elements, respectively. They're more semantic.

---

## 24. What to Immediately Begin Using

With all this talk about HTML5 [not being complete until 2022](#), many people disregard it entirely – which is a big mistake. In fact, there are a handful of HTML5 features that we can use in all our projects right now! Simpler, cleaner code is always a good thing. In today's video quick tip, I'll show you a handful of options.

[Subscribe to our YouTube page to watch all of the video tutorials!](#)

---

## 25. What is Not HTML5

People can be forgiven for assuming that awesome JavaScript-less transitions are grouped into the all-encompassing HTML5. Hey — even Apple has inadvertently promoted this idea. For non-developers, who cares; it's an easy way to refer to modern web standards. However, for us, though it may just be semantics, it's important to understand exactly what **is not** HTML5.

1. **SVG:** Not HTML5. It's at least five years old.
2. **CSS3:** Not HTML5. It's...CSS.
3. **Geolocation:** Not HTML5.
4. **Client Storage:** Not HTML5. It was at one point, but was removed from the spec, due to the fact that many worried that it, as a whole, was becoming too complicated. It now has its own specification.
5. **Web Sockets:** Not HTML5. Again, was exported to its own specification.

Regardless of how much distinction you require, all of these technologies can be grouped into the modern web stack. In fact, many of these branched specifications are still managed by the same people.

---

## 26. The Data Attribute

We now officially have support for custom attributes within all HTML elements. While, before, we could still get away with things like:

[view plaincopy to clipboardprint?](#)

1. `<h1 id=someId customAttribute=value> Thank you, Tony. </h1>`

...the validators would kick up a fuss! But now, as long as we preface our custom attribute with “data,” we can officially use this method. If you’ve ever found yourself attaching important data to something like a `class` attribute, probably for JavaScript usage, this will come as a big help!

### HTML Snippet

[view plaincopy to clipboardprint?](#)

1. `<div id="myDiv" data-custom-attr="My Value"> Bla Bla </div>`

### Retrieve Value of the Custom Attribute

[view plaincopy to clipboardprint?](#)

1. `var theDiv = document.getElementById('myDiv');`
2. `var attr = theDiv.getAttribute('data-custom-attr');`
3. `alert(attr); // My Val`

It can also even be used in your CSS, like for this silly and lame CSS text changing example.

[view plaincopy to clipboardprint?](#)

1. `<!DOCTYPE html>`
2. `<html lang="en">`
3. `<head>`
4. `<meta charset="utf-8">`
5. `<title>Sort of Lame CSS Text Changing</title>`
6. `<style>`
7. `h1 { position: relative; }`
8. `h1:hover { color: transparent; }`
9. `h1:hover:after {`
10. `content: attr(data-hover-response);`
11. `color: black;`
12. `position: absolute;`
13. `left: 0;`
14. `}`
15. `</style>`
16. `</head>`
17. `<body>`
18. `<h1 data-hover-response="I Said Don't Touch Me!"> Don't Touch Me </h1>`
19. `</body>`
20. `</html>`

You can view a demo of the effect above [on JSBIN](#).

---

## 27. The Output Element

As you probably have guessed, the `output` element is used to display some sort of calculation. For example, if you’d like to display the coordinates of a mouse position, or the sum of a series of numbers, this data should be inserted into the `output` element.

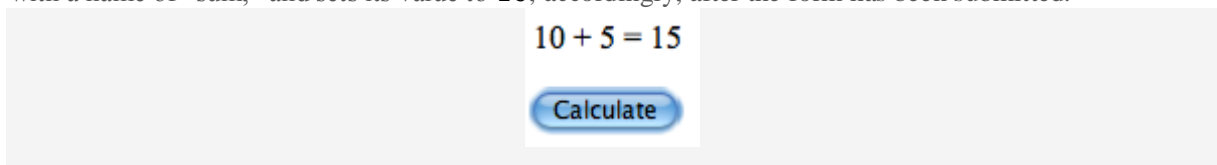
As a simple example, let's insert the sum of two numbers into an empty output with JavaScript, when a `submit` button is pressed.

[view plain](#)[copy to clipboard](#)[print?](#)

```
1. <form action="" method="get">
2.   <p>
3.     10 + 5 = <output name="sum"></output>
4.   </p>
5.   <button type="submit"> Calculate </button>
6. </form>
7. <script>
8. (function() {
9.   var f = document.forms[0];
10.  if ( typeof f['sum'] !== 'undefined' ) {
11.    f.addEventListener('submit', function(e) {
12.      f['sum'].value = 15;
13.      e.preventDefault();
14.    }, false);
15.  }
16.  else { alert('Your browser is not ready yet. '); }
17. })();
18. </script>
```

[Try it out for yourself.](#)

Please note that support for the `output` element is still a bit wonky. At the time of this writing, I was only able to get Opera to play nice. This is reflected in the code above. If the browser does not recognize the element, the browser will simply alert a notice informing you of as much. Otherwise, it finds the output with a name of “sum,” and sets its value to 15, accordingly, after the form has been submitted.



This element can also receive a `for` attribute, which reflects the name of the element that the `output` relates to, similar to the way that a `label` works.

---

## 28. Create Sliders with the Range Input

HTML5 introduces the new `range` type of input.

[view plain](#)[copy to clipboard](#)[print?](#)

```
1. <input type="range">
```

Most notably, it can receive `min`, `max`, `step`, and `value` attributes, among others. Though only Opera seems to support this type of input right now fully, it'll be fantastic when we can actually use this!

For a quick demonstration, let's build a gauge that will allow users to decide how awesome “Total Recall” is. We won't build a real-world polling solution, but we'll review how it could be done quite easily.

### Step 1: Mark-up

First, we create our mark-up.

[view plain](#)[copy to clipboard](#)[print?](#)

```

1. <form method="post">
2.   <h1> Total Recall Awesomness Gauge </h1>
3.   <input type="range" name="range" min="0" max="10" step="1" value="">
4.   <output name="result"> </output>
5. </form>

```

## Total Recall Awesomness Gauge



Notice that, in addition to setting `min` and `max` values, we can always specify what the `step` for each transition will be. If the `step` is set to 1, there will then be 10 values to choose. We also take advantage of the new `output` element that we learned about in the previous tip.

## Step 2: CSS

Next, we'll style it just a bit. We'll also utilize `:before` and `:after` to inform the user what our specified `min` and `max` values are. *Thanks so much to Remy and Bruce for teaching me this trick, via "Introducing HTML5."*

[view plaincopy to clipboardprint?](#)

```

1. body {
2.   font-family: 'Myriad-Pro', 'myriad', helvetica, arial, sans-serif;
3.   text-align: center;
4. }
5. input { font-size: 14px; font-weight: bold; }
6. input[type=range]:before { content: attr(min); padding-right: 5px; }
7. input[type=range]:after { content: attr(max); padding-left: 5px; }
8. output {
9.   display: block;
10.  font-size: 5.5em;
11.  font-weight: bold;
12. }

```

Above, we create content before and after the range input, and make their values equal to the `min` and `max` values, respectively.

## Total Recall Awesomness Gauge



## Step 3: The JavaScript

Lastly, we:

- Determine if the current browser knows what the range input is. If not, we alert the user that the demo won't work.
- Update the `output` element dynamically, as the user moves the slider.
- Listen for when the user mouses off the slider, grab the value, and save it to local storage.
- Then, the next time the user refreshes the page, the range and output will automatically be set to what they last selected.

[view plaincopy to clipboardprint?](#)

```

1. (function() {

```

```

2.  var f = document.forms[0],
3.      range = f['range'],
4.      result = f['result'],
5.      cachedRangeValue = localStorage.rangeValue ? localStorage.rangeValue : 5;
6.  // Determine if browser is one of the cool kids that
7.  // recognizes the range input.
8.  var o = document.createElement('input');
9.  o.type = 'range';
10. if ( o.type === 'text' ) alert('Sorry. Your browser is not cool enough yet. Try the latest Opera. ');
11. // Set initial values of the input and output elements to
12. // either what's stored locally, or the number 5.
13. range.value = cachedRangeValue;
14. result.value = cachedRangeValue;
15. // When the user makes a selection, update local storage.
16. range.addEventListener("mouseup", function() {
17.     alert("The selected value was " + range.value + ". I am using local storage to remember the value. Refresh and check on a modern browser.");
18.     localStorage ? (localStorage.rangeValue = range.value) : alert("Save data to database or something instead.");
19. }, false);
20. // Display chosen value when sliding.
21. range.addEventListener("change", function() {
22.     result.value = range.value;
23. }, false);
24. })();

```

## Total Recall Awesomness Gauge

0 ————— 10

9

Ready for the real world? Probably not yet; but it's still fun to play with and prep for!

[Download the source code](#), and try it out for yourself. But use Opera.

---

Thanks for reading! We've covered a lot, but have still only scratched the surface of what's possible with HTML5. I hope this served as a helpful primer!