

8

Conquer Forms with HTML5 and CSS3

Historically, forms have been a pain to style consistently cross-browser. They also require JavaScript to validate the inputs and lack specific input types to deal with everyday information like telephone numbers, e-mail addresses, and URLs.

The good news is that HTML5 largely solves these common problems. Let's get familiar with the new HTML5 form features and see how they alleviate our traditional form-building burden.

Using HTML5 to code our forms brings an additional benefit when used for responsive designs; it once more allows us to trim our code base to provide the leanest possible pages for our users. For the browsers that don't support these new features, we have tools to patch them up and bring them in line.

In this chapter, we will learn how to use HTML5 to:

- Easily insert placeholder text into relevant form fields
- Disable auto-completion of form fields where necessary
- Set certain fields to be required before submission
- Specify different input types such as e-mail, telephone number, and URL
- Create number range sliders for easy value selection
- Insert date and color pickers
- Learn how we can use a regular expression to define an allowed form value
- Add a polyfill to provide support for less capable browsers
- Use CSS3 to easily and flexibly style an HTML5 form

HTML5 forms

Here's the scenario: for our example *And the winner isn't...* responsive website. I've decided that I'd like people to be able to vent their own frustration at the turkeys that have been picking up the award gongs. We'll be adding a form that let's people tell us about the film they feel shouldn't have won, and the film they feel should have taken its place.

The following screenshot shows how our basic form looks, with just a little basic styling in Chrome (v16):

The screenshot shows a web browser window with the title "Oscar Redemption: And the...". The page content includes a sidebar on the left with two sections: "UNSUNG HEROES..." featuring movie posters for "Midnight Run" and "Nightmare on Elm Street", and "OVERHYPED NONSENSE..." featuring posters for "Moulin Rouge" and "King Kong". The main content area is titled "OSCAR REDEMPTION" and contains the following text: "HERE'S YOUR CHANCE TO SET THE RECORD STRAIGHT: TELL US WHAT YEAR THE WRONG FILM GOT NOMINATED, AND WHICH FILM SHOULD HAVE RECEIVED A NOD...".

The form is divided into three sections:

- About the offending film (part 1 of 3)**
 - The film in question?
 - Year Of Crime
 - Award Won
 - Tell us why that's wrong?
 - How you rate it (1 is woeful, 10 is awesomesauce)
- What should have won? (part 2 of 3)**
 - The film that should have won?
 - Tell us why it should have won?
 - How you rate it (1 is woeful, 10 is awesomesauce)
- About you? (part 3 of 3)**
 - Your Name
 - Telephone (so we can berate you if you're wrong)
 - Your Email address
 - Your Web address

A red "SUBMIT REDEMPTION" button is located at the bottom right of the form.

Besides standard form input fields and text areas, we have a number spinner, a range slider, and placeholder text for many of the fields. If we 'focus' (select) on that particular field the placeholder text is removed and if we lose focus without entering anything (by clicking outside of the input box again) the placeholder text re-appears. Furthermore, looking at this page in Google's Chrome browser, if we go ahead and submit the form without entering anything, the following happens:

The screenshot shows a web browser window with the title "Oscar Redemption: And the...". The page content includes a navigation menu with "UNsung HEROES..." and "OVERHYPED NONSENSE...". The main heading is "OSCAR REDEMPTION" with the sub-heading "HERE'S YOUR CHANCE TO SET THE RECORD STRAIGHT: TELL US WHAT YEAR THE WRONG FILM GOT NOMINATED, AND WHICH FILM SHOULD HAVE RECEIVED A NOD...".

The form is divided into three sections:

- About the offending film (part 1 of 3):**
 - The film in question?
 - Year Of Crime
 - Award Won
 - Tell us why that's wrong?
 - How you rate it (1 is woeful, 10 is awesomesauce)
- What should have won? (part 2 of 3):**
 - The film that should have won?
 - Tell us why it should have won?
 - How you rate it (1 is woeful, 10 is awesomesauce)
- About you? (part 3 of 3):**
 - Your Name
 - Telephone (so we can berate you if you're wrong)
 - Your Email address
 - Your Web address

A red "SUBMIT REDEMPTION" button is located at the bottom right of the form.

So besides a couple of visual flourishes (the slider and spinner) we have some client-side validation in place. As we've already noted, typically, to get a form working like this would require JavaScript of one sort or another.

However, the great news is that all these user interface elements (including the aforementioned slider, placeholder text, and spinner) and the form validation are all being handled natively with HTML5 and no JavaScript is being employed. Let's work through how the new form capabilities of HTML5 make this possible.

Understanding the component parts of HTML5 forms

There's a lot going on in our HTML5 powered form, so let's break it down. The form has been given an ID to aid styling and then an HTML5 hgroup for the title and introductory text:

```
<form id="redemption" method="post">
  <hgroup>
    <h1>Oscar Redemption</h1>
    <h2>Here's your chance to set the record straight: tell us what
      year the wrong film got nominated, and which film <b>should</b>
      have received a nod...</h2>
  </hgroup>
```

The three sections of the form are then wrapped in a fieldset with a legend:

```
<fieldset>
<legend>About the offending film (part 1 of 3)</legend>
<div>
  <label for="film">The film in question?</label>
  <input id="film" name="film" type="text" placeholder="e.g. King
    Kong" required aria-required="true" >
</div>
```

You can see from the previous code snippet that each input element of the form is also wrapped in a div with a label associated with each input. So far, so normal. However, within this first input we've just stumbled upon our first HTML5 form features. After common attributes of id, name, and type we have placeholder.

placeholder

The placeholder attribute looks similar to the following:

```
placeholder="e.g. King Kong"
```

Placeholder text within form fields is such a common requirement that the folks creating HTML5 decided it should be built into the markup and supported by browsers. Simply include the placeholder attribute within your input and the value will be displayed by default until the field gains focus. When it loses focus, if a value has not been entered, it will re-display the placeholder text.

After the placeholder attribute, in the previous code snippet, the next HTML5 form feature is the `required` attribute.

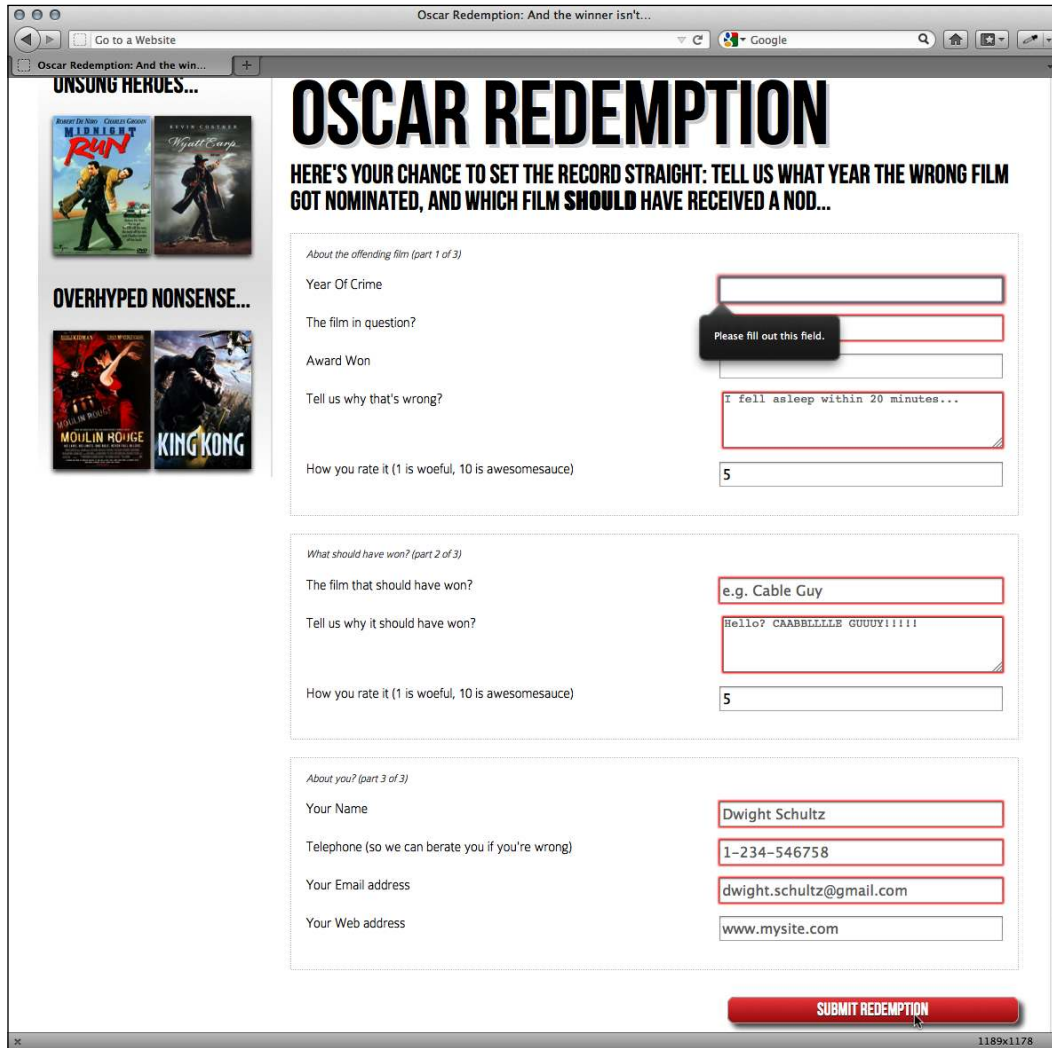
required

The required attribute looks similar to the following:

```
required aria-required="true"
```

In supporting HTML5 capable browsers, by adding the Boolean (meaning you simply include the attribute or not) attribute `required` within the input element, it indicates that a value is required. If the form is submitted without the field containing the requisite information, a warning message should be displayed. The message displayed is specific (both in content and styling) to both the browser and the input type used. In addition to the HTML5 `required` value, in our example we have also added the WAI-ARIA equivalent; `aria-required="true"`. Unless there is a good reason not to, include this WAI-ARIA version of the required attribute to assist those using screen readers (if you remember, we looked at WAI-ARIA back in *Chapter 4, HTML5 for Responsive Designs*).

We've already seen what the `required` field browser message looks like in Chrome. The following screenshot shows the same message in Firefox (9):



The `required` value can be used alongside many input types to ensure a value is entered. Notable exceptions are the `range`, `color`, `button`, and `hidden` input types as they almost always have a default value.

Another HTML5 form attribute that can be added to input fields is `autofocus`.

autofocus

The HTML5 `autofocus` attribute allows a form to be loaded with a field already focused (selected) ready for user input. The following code is an example of an input field wrapped in a `div` with the `autofocus` attribute added at the end:

```
<div>
  <label for="search">Search the site...</label>
  <input id="search" name="search" type="search" placeholder="Wyatt
    Earp" autofocus>
</div>
```

Be careful when using this attribute. Cross browser confusion can reign if multiple fields have the `autofocus` attribute added. For example, if multiple fields have `autofocus` added, in Chrome (v16) the last field with the `autofocus` attributed is focused on page load. However, Firefox (v9) does the opposite with the first `autofocus` field selected.

It's also worth considering that some users use the space bar to quickly skip down the content of a web page once it's loaded. On a page where a form has an autofocused input field, it prevents this capability; instead it adds a space into the focused input field. It's easy to see how that could be a source of frustration for users.

autocomplete

By default, most browsers aid user input by auto-completing the value of form fields where possible. Whilst the user can turn this preference on and off within the browser, we can now also indicate to the browser when we don't want a form or field to allow auto-completion. This is useful not just for sensitive data (for example bank account numbers) but also if you want to ensure users pay attention and enter something *by hand*. For example, for many forms I complete, if a telephone number is required, I enter a 'spooof' telephone number. I know I'm not the only one that does that (doesn't everyone?) but I can ensure that users don't enter an auto-completed spooof number by setting the `autocomplete` attribute to `off` on the relevant input field. The following is a code example of a field with the `autocomplete` attribute set to `off`:

```
<div>
  <label for="tel">Telephone (so we can berate you if you're
    wrong)</label>
  <input id="tel" name="tel" type="tel" placeholder="1-234-546758"
    autocomplete="off" required aria-required="true" >
</div>
```

We can also set entire forms (but not fieldsets) to not autocomplete by using the `autocomplete` attribute on the form itself. The following is a code example:

```
<form id="redemption" method="post" autocomplete="off">
```

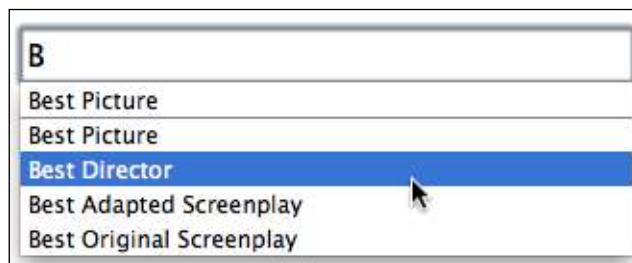
list (and the associated datalist element)

This `list` attribute and the associated `datalist` element allow a number of selections to be presented to a user once they start entering a value in the field. The following is a code example of the `list` attribute in use with an associated `datalist` wrapped in a `div`:

```
<div>
  <label for="awardWon">Award Won</label>
  <input id="awardWon" name="awardWon" type="text" list="awards">
  <datalist id="awards">
    <select>
      <option value="Best Picture"></option>
      <option value="Best Director"></option>
      <option value="Best Adapted Screenplay"></option>
      <option value="Best Original Screenplay"></option>
    </select>
  </datalist>
</div>
```

The value given in the `list` attribute (`awards`) refers to the `id` of the `datalist`. Doing this associates the `datalist` with the input field. Although wrapping the options with a `<select>` element isn't strictly necessary, it helps when applying polyfills for older browsers.

Whilst the input field seems to be just a normal text input field, when typing in the field, a selection box appears below it (in supporting browsers) with matching results from the `datalist`. In the following screenshot, we can see the list in action (Firefox v9). In this instance, as **B** is present in all options within the `datalist`, all values are shown to select from:



However, when typing **D** instead, only the matching suggestions appear as shown in the following screenshot:



This doesn't prevent a user entering anything else they want in the input box but it provides another great way of adding common functionality and user enhancement through markup alone.

HTML5 input types

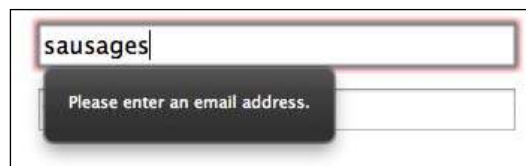
HTML5 adds a number of extra input types, which amongst other things, enable us to limit the data that users input without the need for extraneous JavaScript code. The most comforting thing about these new input types is that by default, where browsers don't support the feature, they degrade to a standard text input box. Furthermore, there are great polyfills available to bring older browsers up to speed. We will look at these shortly. In the meantime, let's look at these new HTML5 input types and the benefits they provide.

email

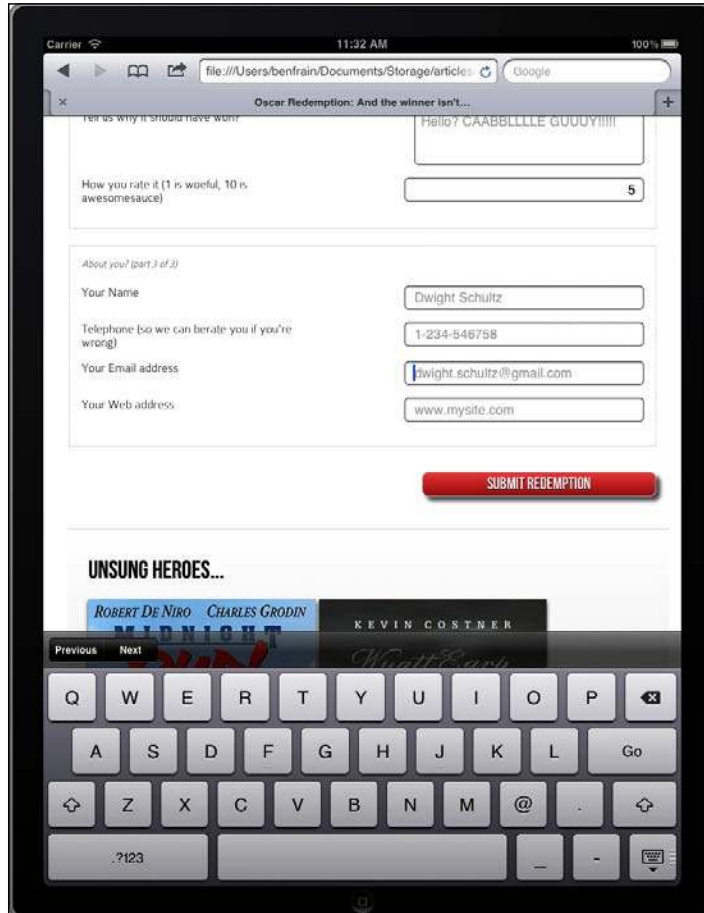
`type="email"` – supporting browsers will expect a user input that matches the syntax of an e-mail address. In the following code example `type="email"` is used alongside `required` and `placeholder`:

```
<div>
  <label for="email">Your Email address</label>
  <input id="email" name="email" type="email" placeholder=
    "dwight.schultz@gmail.com" required aria-required="true">
</div>
```

When used in conjunction with `required` submitting a non-conforming input will generate a warning message:



Furthermore, many touch screen devices (for example Android, iPhone and so on) change the input display based upon this input type. The following screenshot shows how an input `type="email"` screen looks on the iPad. Notice the '@' symbol for easy email address completion:



number

`type="number"` – supporting browsers expect a number to be entered in a number type input field. They also supply *spinner* controls by default, allowing users to easily click up or down to alter the value. The following is a code example:

```
<div>
  <label for="yearOfCrime">Year Of Crime</label>
  <input id="yearOfCrime" name="yearOfCrime" type="number" min="1929"
    max="2015" required aria-required="true" >
</div>
```

And the following screenshot shows how it looks in a supporting browser (Chrome v16):



Implementation of what happens if you don't enter a number varies. For example, Chrome (v16) clears the field as soon as it loses focus without providing any feedback whilst Firefox (v9) allows anything to be entered (defaulting to the standard text input type). You'll notice in the previous code example, we have also set a minimum and maximum allowed range similar to the following code:

```
type="number" min="1929" max="2015"
```

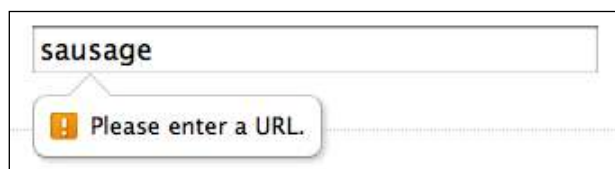
Numbers outside of this range (should) get special treatment. Browser implementation is varied. For example, Chrome (v16) displays a warning whilst Firefox (v9) does nothing.

url

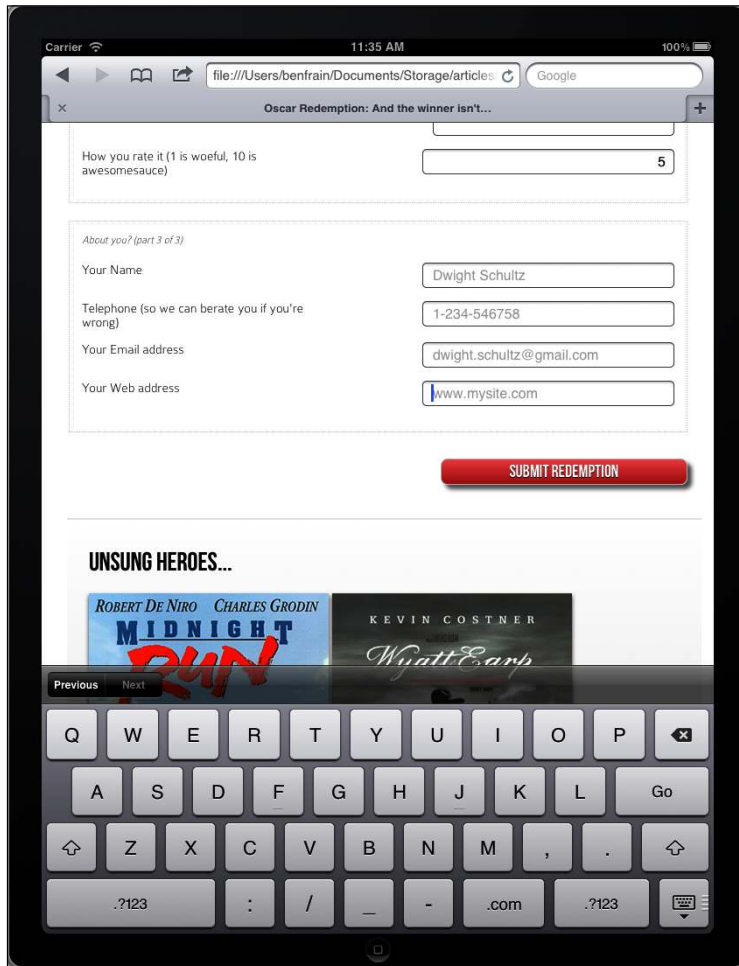
`type="url"` - as you might expect, the URL input type is for URL values. Similar to the `tel` and `email` input types, it behaves almost identically to a standard text input. However, some browsers add specific information to the warning message provided when submitted with incorrect values. The following is a code example including the `placeholder` attribute:

```
<div>
  <label for="web">Your Web address</label>
  <input id="web" name="web" type="url" placeholder="www.mysite.com">
</div>
```

The following screenshot shows what happens when an incorrectly entered URL field is submitted in Chrome (v16):



Like `type="email"`, touch screen devices often amend the input display based upon this input type. The following screenshot shows how an input `type="url"` screen looks on the iPad:



Notice the **Go**, forward slash (/), and **.com** keys? Because we've used a URL input type they are presented by the device for easy URL completion (unless you're not going to a .com site in which case, you know, thanks for nothing Apple).

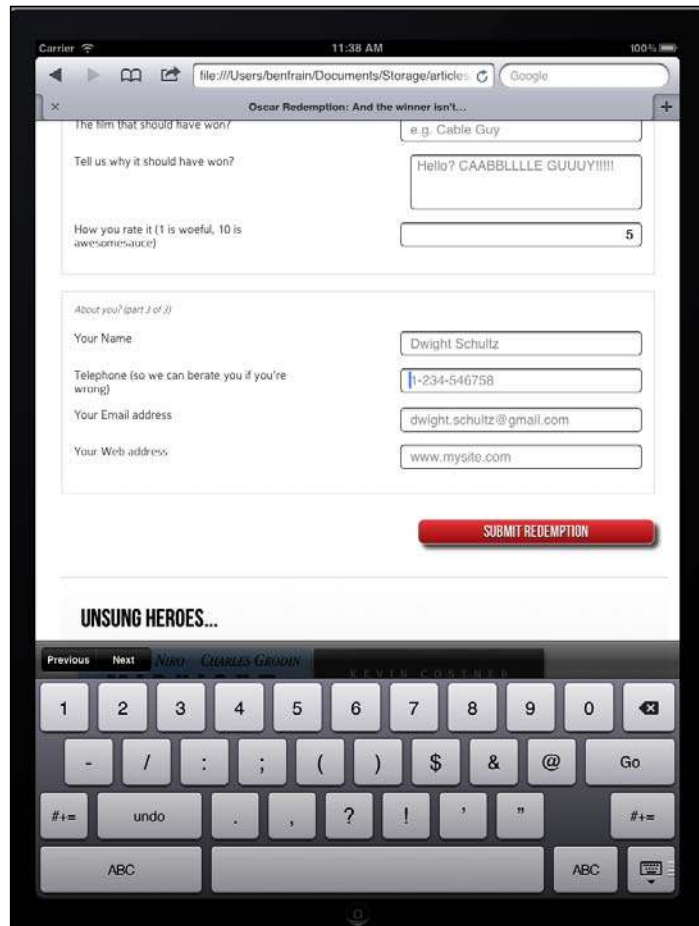
tel

`type="tel"` is another contact information specific input type. `tel` is used to signify to the browser that the form expects a telephone number entered within that field. The following code is an example:

```
<div>
  <label for="tel">Telephone (so we can berate you if you're
    wrong)</label>
  <input id="tel" name="tel" type="tel" placeholder="1-234-546758"
    autocomplete="off" required aria-required="true" >
</div>
```

Although, a number format is expected, on many browsers, even modern ones such as Chrome v16 and Firefox v9, it merely behaves like a text input field. They are currently failing to provide a suitable warning message on form submission when incorrect values are entered.

However, better news is that like the `email` and `url` input types, touch screen devices often thoughtfully accommodate this kind of input with an amended input display for easy completion; here's the `tel` input when accessed with an iPad (running iOS 5):



Notice the lack of alphabet characters in the keyboard area? This makes it much faster for users to enter a value in the correct format.

search

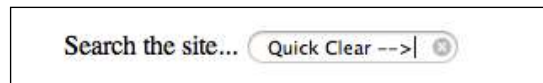
`type="search"` – although the `search` input type works in the same manner as a standard text input, some browsers render the code with some subtle differences. The following code is an example:

```
<div>
  <label for="search">Search the site...</label>
  <input id="search" name="search" type="search" placeholder=
    "Wyatt Earp">
</div>
```

The following screenshot shows how the previous code looks in Firefox (v9); notice the default styling of the input box is rectangular:



However, Chrome (v16) renders that same code differently by default with rounded edges and a quick clear button on the right:



pattern

`pattern=""` – *Be afraid, be very afraid* (remember what film that's the tagline from?) In my opinion, this tagline could just as easily be applied to **regular expressions**. If you don't know what regular expressions are, I dare say ignorance is bliss. If you do, and worse still, you understand them, the following section is for you.



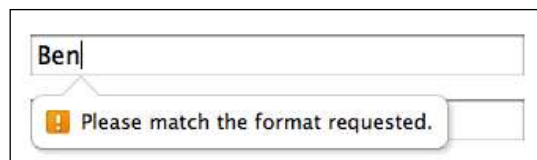
Learn about regular expressions

If you've watched 'The Exorcist' alone, in a graveyard, at midnight, on Halloween you're possibly ready to learn about regular expressions: http://en.wikipedia.org/wiki/Regular_expressions.

The `pattern` attribute allows you to specify, via a regular expression, the syntax of data that should be allowed in a given input field. The following code is an example:

```
<div>
  <label for="name">Your Name (first and last)</label>
  <input id="name" name="name" pattern="([a-zA-Z]{3,30}\s*)+[a-zA-Z]{3,30}" placeholder="Dwight Schultz" required aria-required="true" >
</div>
```

Such is my commitment to this book, I searched the Internet for approximately 458 seconds to find a regular expression that would match a first and last name syntax. By entering the regular expression value within the `pattern` attribute, it makes supporting browsers expect a matching input syntax. Then, when used in conjunction with the `required` attribute, incorrect entries get the following treatment in supporting browsers. In this instance I tried submitting the form without providing a last name:

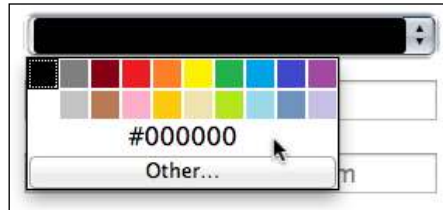


color

`type="color"` – the `color` input type produces a color picker in supporting browsers, allowing users to select a color value in a Hexadecimal value. The following code is an example:

```
<div>
  <label for="color">Your favorite color</label>
  <input id="color" name="color" type="color">
</div>
```

Sadly, at present, browser support is scant. Only Opera (v11) seems to provide the color picker. When the required color isn't initially shown, clicking the **Other...** button at the bottom launches the OS's default color picker:



Date and time inputs

The thinking behind the new `date` and `time` input types is to provide a consistent user experience for choosing dates and times. If you've ever bought tickets to an event online, chances are that you have used a date picker of one sort or another. This functionality is almost always provided via JavaScript (typically jQuery) but the hope is to make this common necessity possible merely with HTML5 markup.

date

The following code is an example:

```
<input id="date" type="date" name="date" />
```

Similar to the `color` input type, native browser support is thin on the ground at present, defaulting on most browsers to a standard text input box. Good ol' Opera has already implemented the functionality though and the following screenshot shows how that example code renders in Opera (v11):



There are a variety of different `date` and `time` input types available. What follows is a brief overview of the others.

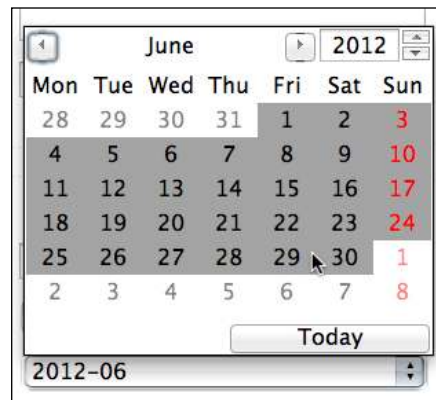
month

The following code is an example:

```
<input id="month" type="month" name="month">
```

The interface allows the user to select a single month and provides the input as a year and month for example **2012-06**.

The following screenshot shows how it looks in the browser:



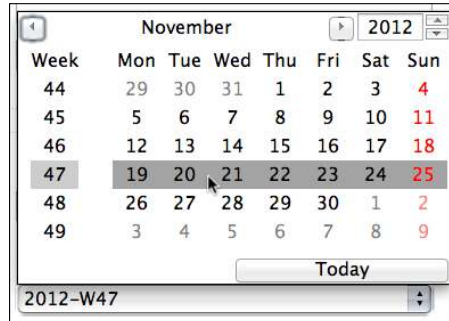
week

The following code is an example:

```
<input id="week" type="week" name="week">
```

When the `week` input type is used, the picker allows the user to select a single week within a year and provides the input in the **2012-W47** format.

The following screenshot shows how it looks in the browser:



time

The following code is an example:

```
<input id="time" type="time" name="time">
```

The `time` input type allows a value in the 24 hour format, for example **23:50**.

It displays in supporting browsers with spinner controls but only allows relevant time values:

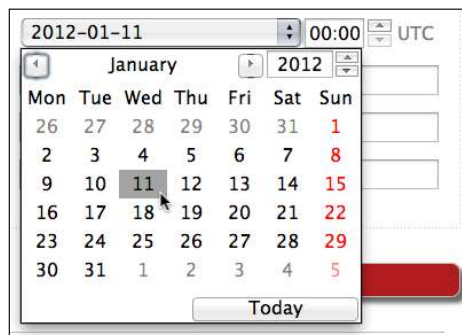


datetime and datetime-local

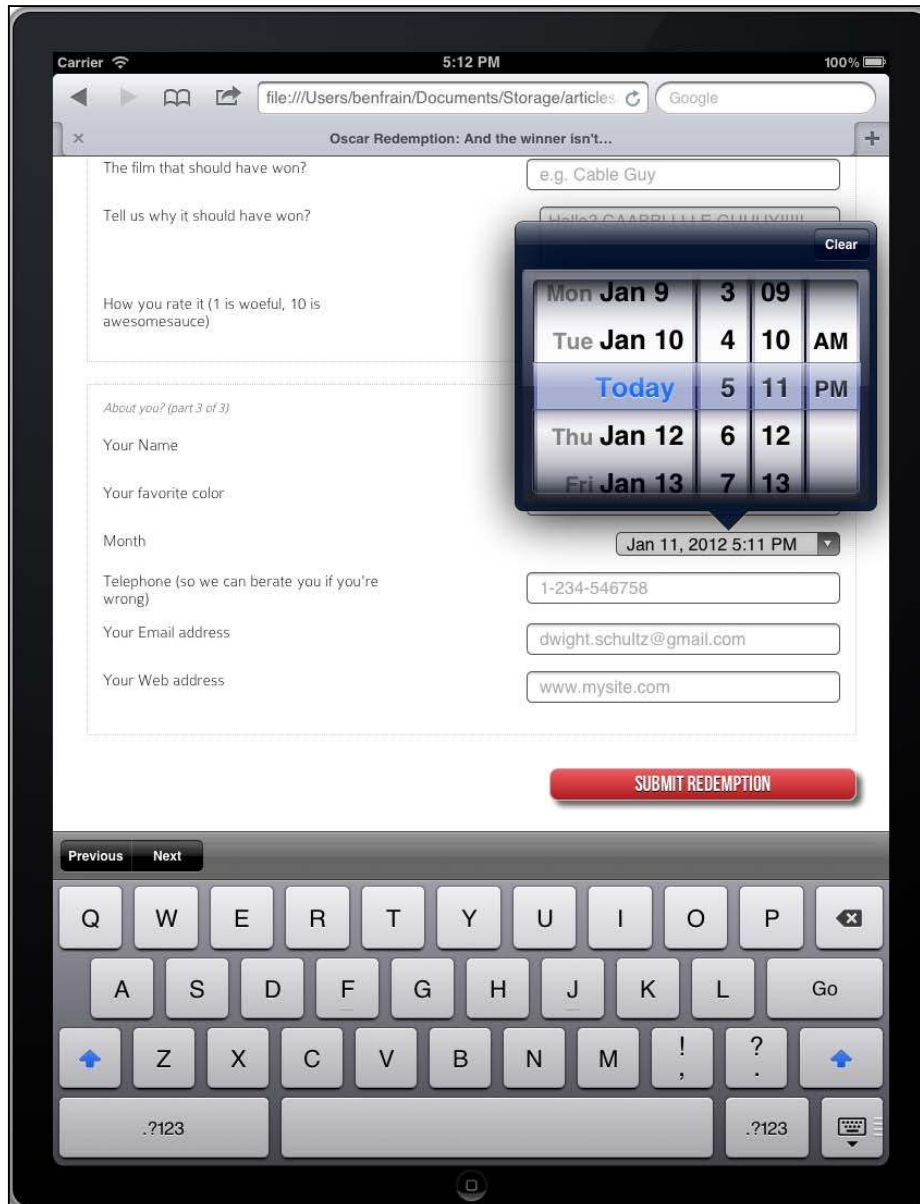
The following code is an example:

```
<input id="datetime" type="datetime" name="datetime">
```

It looks similar to the following screenshot in Opera (v11):



And looks even better on iOS devices as shown in the following screenshot:



This input type creates date and time values (separated by a `T`) and then the time zone (`Z` for UTC or a `+` or `-` for offset values). 25th October 2009 in UTC is shown as follows:

```
2009-10-25T05:05:00Z
```

As UTC is, for most practical purposes, equivalent to GMT, it's easy to understand offsets. For example, Pacific Standard Time (Los Angeles) is 8 hours behind GMT (UTC -8 hours). That would be reflected in the input value as shown:

```
2009-10-25T05:05:00-8:00
```

The `datetime-local` version works in exactly the same manner as `datetime` but omits the time zone information.



Changing the step increments

You can alter the step increments (granularity) for the spinner controls of various input types with the use of the `step` attribute. For example, to step 4 hours at a time, enter the value of 4 hours as 14400 seconds (60 (seconds), multiplied by 60 (minutes), multiplied by 4 (hours)). Following is the `datetime` example amended to use 4-hour steps in the time selector:

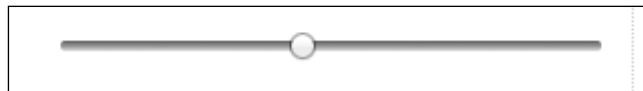
```
<input id="datetime" type="datetime" name="datetime" step="14400">
```

range

The `range` input type creates a slider interface element. The following code is an example:

```
<input id="howYouRateIt" name="howYouRateIt" type="range" min="1"
max="10" value="5" >
```

And the following screenshot shows how it looks in Safari (v5.1):



The default range is from 0 to 100. However, by specifying a `min` and `max` value in our example we have limited it to between 1 and 10.

One big problem I've encountered with the range input type is that the current value is never displayed to the user. Although the range slider is only intended for vague number selections, I've often wanted to display the value as it changes. Currently, there is no way to do this using HTML5. However, if you absolutely must display the current value of the slider, it can be achieved easily with some simple JavaScript. Amend the previous example to the following code:

```
<input id="howYouRateIt" name="howYouRateIt" type="range" min="1"
  max="10" value="5" onchange="showValue(this.value)"><span
  id="range">5</span>
```

We've added two things, an `onchange` attribute and also a `span` element with the `id` of `range`. Now, we'll add the following tiny piece of JavaScript somewhere in the page:

```
<script>
  function showValue(newValue)
  {
    document.getElementById("range").innerHTML=newValue;
  }
</script>
```

All this does is gets the current value of the range slider and display it in the element with an `id` of `range` (our `span` tag). With a tiny bit of CSS styling to make the value bigger and red, the following screenshot shows the effect—with the value updating as the slider is moved:



There are a few other form related features that are new in HTML5 but as they relate more to building applications and backend development they've not been featured here. To read the *W3C Editor's draft of the HTML5 form section* visit: <http://dev.w3.org/html5/spec-author-view/forms.html#forms>.

How to polyfill non-supporting browsers

All this HTML5 form malarkey is all well and good. There seems however, to be two things that put a serious dent in our ability to use them: disparity between how supporting browsers implement the features and how to deal with browsers that don't support the features at all. Thankfully, as ever, the web community has found a way.

Back in *Chapter 4, HTML5 for Responsive Designs* I mentioned Modernizr (<http://www.modernizr.com>), a fantastic JavaScript library that helps insert **polyfills** for browsers lacking the requisite HTML5/CSS3 features. "*Webshims Lib*", written by Alexander Farkas (<http://afarkas.github.com/webshim/demos/>) is built on top of this and the ubiquitous jQuery library to only load the form polyfills (it can handle poly-filling of other HTML5 features too) needed to make non-supporting browsers handle our HTML5 forms. What's particularly great is the fact that as it utilizes Modernizr's loading capabilities, the relevant polyfills are only added if needed. It adds very little flab to a web page if being viewed by a browser that supports these HTML5 features natively. Older browsers, although they need to load more code (as they are less capable by default), get a similar user experience, albeit with the relevant functionality created with the help of JavaScript.

But it isn't just older browsers that benefit. As we've seen, many modern browsers haven't implemented the HTML5 form features fully. Employing Webshims Lib to the page also fills any gaps in their capability. For example, Safari (5.1) doesn't offer any warning when a HTML5 form is submitted with any required fields empty. Whilst the form isn't actually submitted, no feedback is given to the user as to what the problem is: hardly ideal. With Webshims Lib added to the page, the following happens in the aforementioned scenario:



So when Firefox (v9) isn't able to provide a spinner for a `type="number"` attribute, Webshims Lib provides a suitable jQuery powered fallback. In short, it's a great tool, so let's get this beautiful little package installed and hooked up and then we can carry on writing forms with HTML5, safe in the knowledge all users will see what they need to use our form (except those two people using IE6 with JavaScript turned off — you know who you are — now pack it in!).

First download Webshims Lib (<http://github.com/aFarkas/webshim/downloads>) and extract the package. Now copy the `js-webshim` folder to a relevant section of your web page. For simplicity, for this example I've copied it into the website root.

Now add the following code into the `<head>` section of your page:

```
<script src="js/jquery-1.7.1.js"></script>
<script src="js-webshim/minified/extras/modernizr-
  custom.js"></script>
<script src="js-webshim/minified/polyfiller.js"></script>
<script>
  //load all defined
  $.webshims.polyfill();
</script>
```

Let's go through this a section at a time. First I've linked to a local copy of the jQuery library (get the latest version at www.jquery.com):

```
<script src="js/jquery-1.7.1.js"></script>
```

Next, I'm adding the versions of Modernizr and the polyfiller JavaScript files that are within Webshims Lib:

```
<script src="js-webshim/minified/extras/modernizr-custom.js"></script>
<script src="js-webshim/minified/polyfiller.js"></script>
```

Finally, I'm telling the script to load all needed polyfills:

```
<script>
  //load all defined
  $.webshims.polyfill();
</script>
```

And that's all there is to it. Now, missing functionality is automatically added by the relevant polyfill. Excellent!

Styling HTML5 forms with CSS3

Our form is now fully functional across all browsers and whilst we've got some very basic styling, you and I both know, with CSS3 we can do so much better. Let's apply some of the techniques we've already learned and used to spice up our form a little. So far, the following are all the form specific styles we have:

```
#redemption {
  width: 100%;
  font-family: 'ColaborateThinRegular';
  font-weight: 400;
```

```
}
#redemption hgroup {
  margin-bottom: 20px;
}
#redemption div {
  width: 100%;
  margin-bottom: 15px;
  float: left;
}
#redemption span#range {
  float: left;
  font-size: 3em;
  width: 100%;
  color: red;
  clear: both;
  text-align: center;
}
#howYouRateThis,#yearOfCrime {
  text-align: right;
}
#redemption legend {
  font-style: italic;
  color: #434242;
  font-size: 0.8em;
  margin-bottom: 20px;
  float: left;
  width: 100%;
}
#redemption fieldset {
  border: 1px dotted #cccccc;
  padding: 2%;
  margin-bottom: 20px;
}
#redemption label {
  width: 40%;
  float: left;
}
#redemption input {
  height: 20px;
  font-size: 1em;
  width: 40%;
  float: right;
}
#redemption textarea {
```



```

    height: 60px;
    font-size: 1em;
    width: 40%;
    float: right;
}
#redemption input#submit {
    text-decoration: none;
    height: 34px;
    font: 1.25em /* 36px + 16 */ 'BebasNeueRegular';
    background-color: #b01c20;
    border-radius: 8px;
    color: white;
    float: right;
    margin-bottom: 10px;
    background: linear-gradient(top, rgb(241,92,96) 0%, rgb(176,28,32)
        100%);
    margin-top: 10px;
    box-shadow: 5px 5px 5px hsla(0, 0%, 26.6667%, 0.8);
    text-shadow: 0px 1px black;
    border: 1px solid #bfbfbf;
}
.polyfill-important .input-range, .polyfill-important .step-controls {
    float: right;
}
.polyfill-important .step-controls {
    margin-right: -20px!important;
}

```

The only point worthy of note here is that the final two styles are only relevant when some of the polyfills are loaded.

So, first off, I want to make each `fieldset` stand out a little more with a subtle gradient background. The following is the amended CSS for the `fieldset`:

```

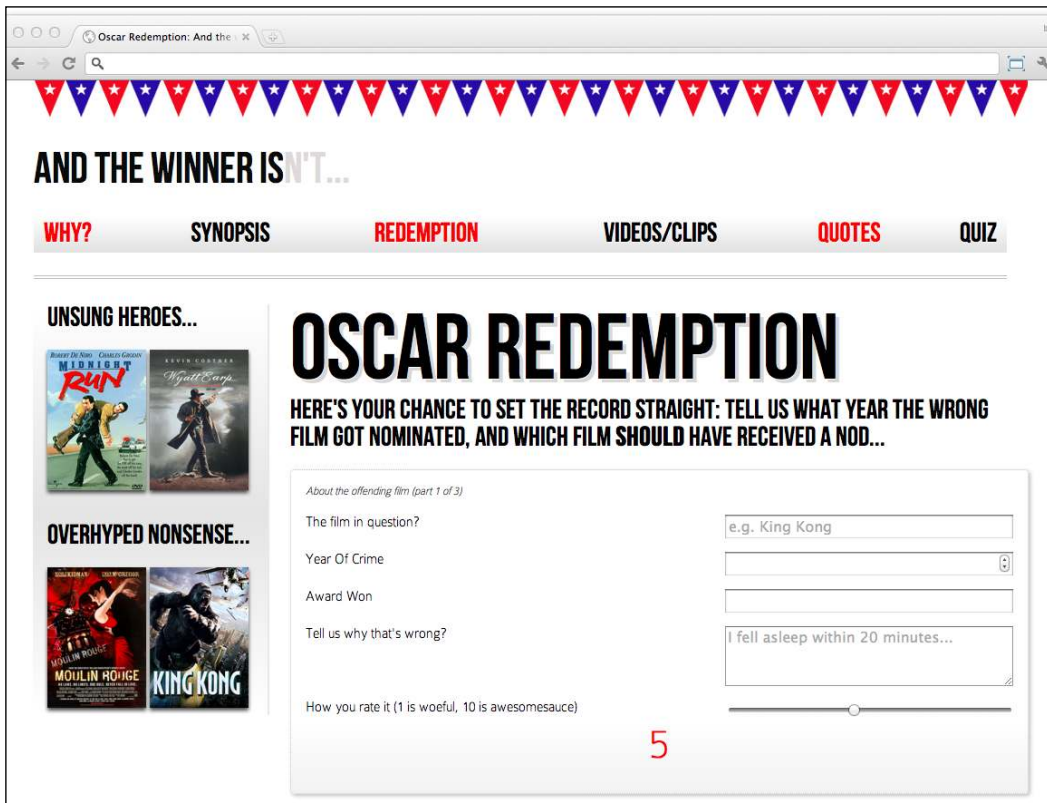
#redemption fieldset {
    border: 1px dotted #cccccc;
    padding: 2%;
    margin-bottom: 20px;
    background: #ffffff;
    background: linear-gradient(top, #ffffff 77%, #f2f2f2 100%);
    border-radius: 4px;
    box-shadow: 2px 2px 5px hsla(0, 0%, 16.6667%, 0.3);
}

```

Aside from the `border-radius`, and `background gradient`, the only other thing we have done is add a subtle `box-shadow` declaration.

As in many of the previous examples, I've omitted vendor-prefixed versions of the CSS3 declarations (`background gradient`, `border-radius`, and `box-shadow` in this case).

The following screenshot is the output shown in Chrome:



Mixing color values

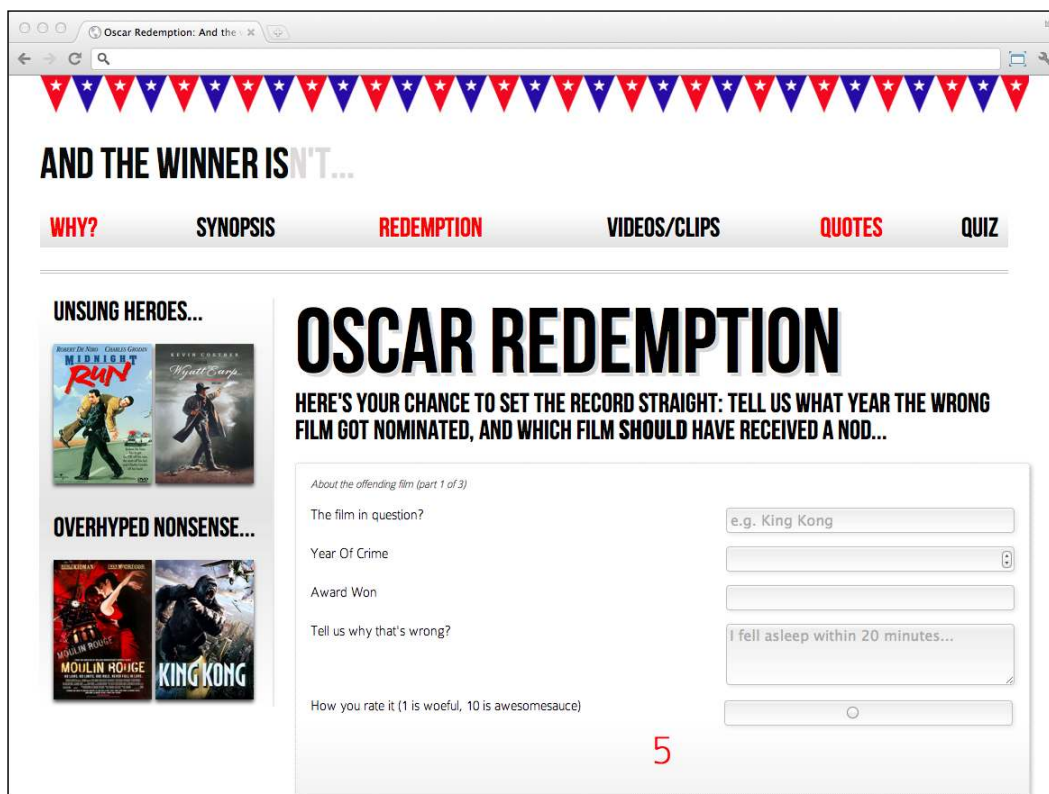


Throughout the examples you can see that I've mixed and matched how colors have been defined. In some instances I'm using values like `red` whilst I've also used `HEX`, `RGB` and `HSL` values too. In supporting browsers there is no penalty for doing so. In a production site however, you may choose to stick to one or two formats for consistency.

So far, so good. But those text input fields are still looking a little drab. Let's add a sprinkling of CSS3 there too using the following code:

```
input, textarea, select {
  border: 1px solid #bfbfbf;
  padding: 0.2em;
  font-size: 1.1em;
  line-height: 1.2em;
  background: #ffffff;
  background: linear-gradient(top, #ffffff 0%, #ededed 8%, #ffffff
    100%);
  border-radius: 4px;
  box-shadow: 2px 2px 5px hsla(0, 0%, 16.6667%, 0.1);
}
```

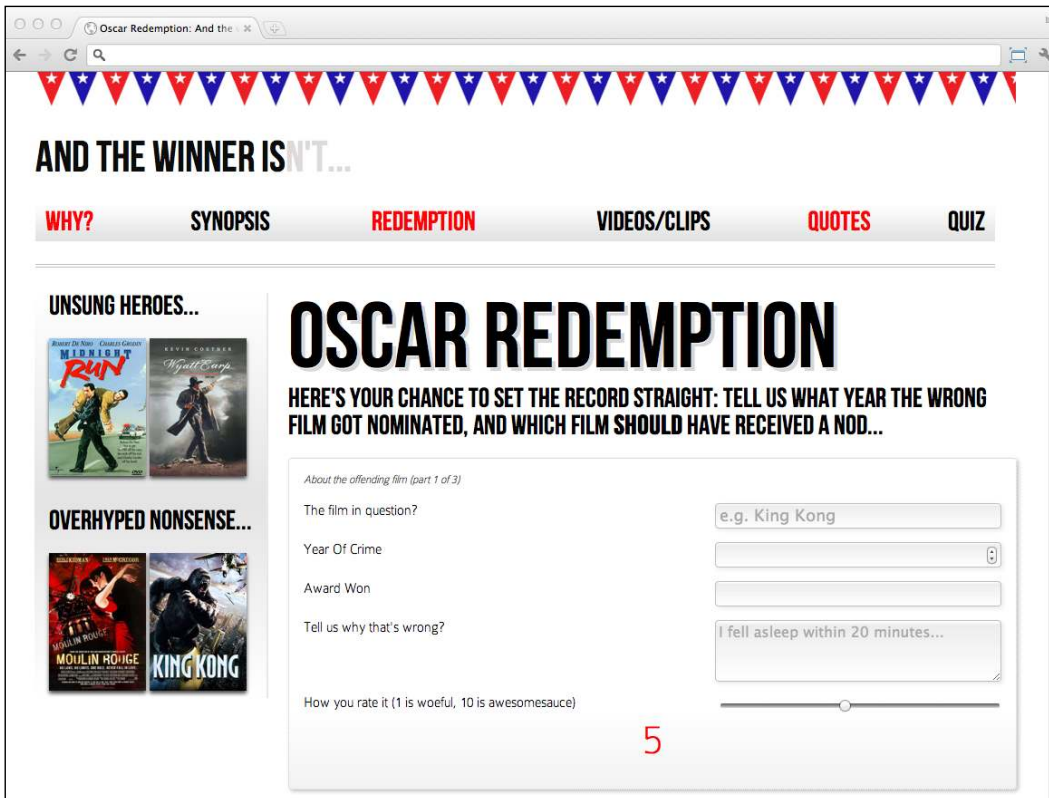
Again, we've got a background gradient there, a slight border-radius, and a subtle box-shadow. The following screenshot shows how it looks in Chrome:



I'm happy with that...Oh, hold on. Take a look at the slider at the bottom. That's not what I want. I don't want those rules to affect the range slider so I'll amend my selector and use one of the new CSS3 selectors to sort things out:

```
input:not([type="range"]), textarea, select{
  /* the styles */
}
```

I've used the `:not` pseudo selector to specify that I don't want the rule to apply to inputs with the attribute `type="range"`. Let's take another look in Chrome:



Excellent! That's what I was gunning for and CSS3 has made it easy to not only add the relevant styles, but also to prevent adding them to elements on which they're not wanted.

Form-specific CSS3 pseudo class selectors

Alongside all the fun CSS3 tools we already know about, there are also a few form-specific pseudo selectors:

- `input:required`: for required fields
- `input:focus:invalid`: for focused fields that have an invalid value
- `input:focus:valid`: for focused fields that have a valid value

So, let's use these to make three additional style rules as shown in the following code examples:

```
input:required {
  border: 1px solid rgba(253, 8, 8, 0.29);
}
input:focus:invalid {
  background: url('../img/cross.png') no-repeat right;
  padding-right: 3px;
}
input:focus:valid {
  background: url('../img/tick.png') no-repeat right;
  padding-right: 3px;
}
```

The first is a subtle border for required fields. The second adds a cross for when an incorrect value has been included as the user types and the final rule adds a green tick when a correct value has been entered.

The following screenshot shows how that works in the browser (Firefox v9) on page load:



The screenshot shows a web form with the following elements:

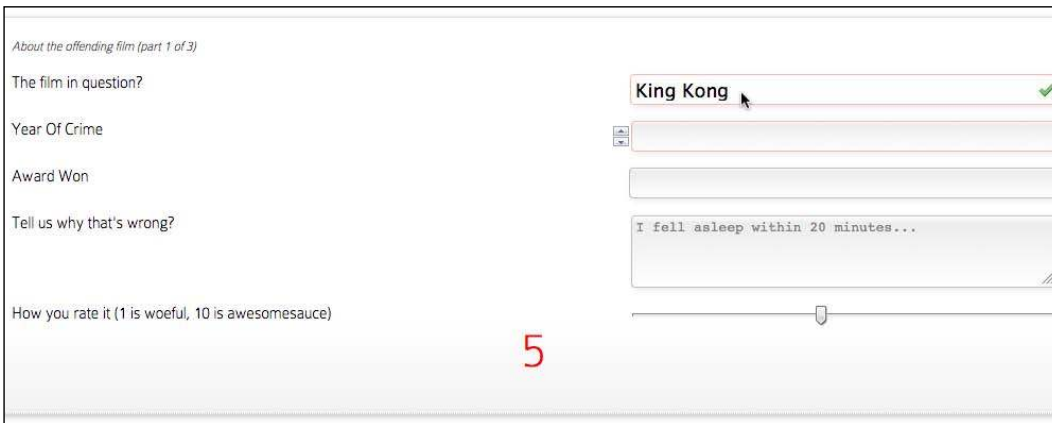
- About the offending film (part 1 of 3)**
- The film in question?** Input field containing "e.g. King Kong" with a red border and a red cross icon on the right.
- Year Of Crime** Dropdown menu.
- Award Won** Input field.
- Tell us why that's wrong?** Textarea containing "I fell asleep within 20 minutes...".
- How you rate it (1 is woeful, 10 is awesomesauce)** Slider control with a red "5" displayed below it.

Now, if we focus (click into) on one of the required input fields, a **red cross** appears (as we haven't yet entered a valid value):



A screenshot of a web form titled "About the offending film (part 1 of 3)". The form contains several input fields: "The film in question?", "Year Of Crime", "Award Won", "Tell us why that's wrong?", and "How you rate it (1 is woeful, 10 is awesomesauce)". The "The film in question?" field is currently empty and has a red border with a red 'X' icon in the top right corner. A yellow tooltip with the text "Please fill out this field." is visible over the field. A red number "5" is displayed in the bottom right corner of the form area.

If we go ahead and enter a valid value, the **red cross** image swaps out for our **green tick**:



A screenshot of the same web form as above. The "The film in question?" field now contains the text "King Kong" and has a green border with a green checkmark icon in the top right corner. The red 'X' and tooltip are no longer present. The red number "5" remains in the bottom right corner of the form area.

Using these new CSS3 pseudo class selectors makes for a nice, easy to implement, layer of enhancement that adds to the overall user experience when filling in the forms.

Summary

In this chapter, we have learned how to use a host of new HTML5 form attributes. They enable us to make forms more usable than ever before and the data they capture more relevant. Furthermore, we can future proof this new markup by using JavaScript feature detection and conditional loading of JavaScript polyfill scripts so that all users experience similar form features, regardless of their browsers capability.

We're nearing the end of our Responsive HTML5 and CSS3 journey. We've covered a lot of theory alongside our practical 'And the winner isn't' example website. However, implementing responsive designs in the real world often presents further challenges. How to handle a mass of navigation links on a small screen? How to only load additional files for the browsers that need them? In the final chapter we will be looking at some of these common issues (and their solutions) when implementing responsive designs built with HTML5 and CSS3. We'll also revisit how best to deal with some specific shortcomings of common older browsers.