HTML forms - the basics

From Web Education Community Group

Contents

- 1 Introduction
- 2 Step one: The basic code
  - 2.1 The <form> element
  - 2.2 The <input> element
  - 2.3 The type attribute
  - 2.4 The name attribute
  - 2.5 The value attribute
  - 2.6 the <textarea> element
  - 2.7 <input type="submit">
- 3 Step two: Adding structure and behaviour
  - 3.1 Giving the form an id
  - 3.2 The method attribute: telling the data how to travel
  - 3.3 The action attribute: telling the data where to go
  - 3.4 hidden form inputs
  - 3.5 Structuring with an unordered list
  - 3.6 Labels for accessibility
- 4 Step three: Adding some more complex form elements
  - 4.1 Checkboxes: providing multiple yes/no choices
  - 4.2 Radio buttons: a multiple choice option
  - 4.3 The checked attribute
  - 4.4 Uploading a file
  - 4.5 Select and option: creating a multi-line dropdown menu
- 5 Step four: further structuring with fieldsets and legends
  - 5.1 Explaining fieldset and legend
  - 5.2 A little bit of style...
- 6 Summary
- 7 Exercise questions

Introduction

Everyone has seen a form. Everyone has used one. But have you coded one?

Most areas where you can input information into web pages are created using HTML forms, for example entering text or numbers into a text box, checking a tick box, causing a radio button to “fill in”, or selecting an option from a list. The form is then usually submitted to the web site when you submit the form, and the data is used in some way, for example customer details being stored in a database for later use, or a web site being instantly updated after a buyer chooses a product to look at or comments on a blog post.

It is easy to create a form, but what about a usable form that your users can fill in painless, rather than getting frustrated and fed up? In this article we’ll cover all the basics you need to know to create elegant, accessible form structures with HTML, starting with the real basics, and working up to some more complex features. We'll then look at the new form features in HTML5 separately in the next article: HTML5 form additions. We have done it like this because it is useful to see where the distinction is, plus as of the time of this writing (18th August 2011), HTML5 form support is not quite there across all browsers.

Step one: The basic code

Let’s start building up a comment form, the sort of form you would use on a web site to allow people to give you feedback on something such as an article you’ve written, or a product you've sold. We'll start it off really basic:

```
<form>
  Name: <input type="text" name="name" id="name" value="" />
  Email: <input type="text" name="email" id="email" value="" />
  Comments: <textarea name="comments" id="comments" cols="25" rows="3"></textarea>
  <input type="submit" value="submit" />
</form>
```

If you enter this into an HTML document and then open that document in a browser, the code is rendered as shown in Figure 1.

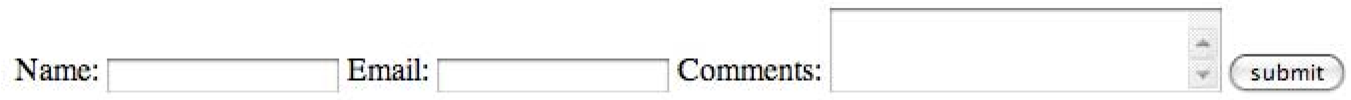


Figure 1: The first, basic form example.

Try it for yourself — enter the above code into your own sample HTML document and load it in a browser, or click here to navigate to the form in a separate page (http://dev.opera.com/articles/view/20-html-forms-the-basics/step-1-form.html) . Try playing around with the different form controls to see what you can do with them.

As you read the code, you’ll see an opening `<form>` tag, a `</form>` closing tag, and some bits in between the two. The element contains three inputs in which you can enter information: two single line text inputs, and a multiline text input.

What have we got here?

The `<form>` element

The <form>element is needed to wrap the form — without it you don’t have a web form. The `<form>` tag can have a few attributes, which will be explained in Step Two, but please do note that you can’t nest forms inside each other.

The `<input>` element

`<input>`: This element defines an area where you can insert information in some way.

The `type` attribute

There are different `types` of `<input>` element that we'll look at later, but in this case we are using `<input type="text">` to specify that we want single line text input fields (the type attribute is mandatory). We have also put "name" and "email" labels next to them, to give our site visitors an indication of what data we'd like them to enter there.

The `name` attribute

Every `<input>` element must also have a `name` attribute that you the developer can decide on. The only exceptions to this rule are special cases where the `value` attribute is always set to the same value as the `type` attribute, eg `type="submit"` or `type="reset"`; in those cases the `name` attribute is not required. The `name` attribute is needed for the database or other data destination to uniquely identify that piece of data.

When the form is submitted, most scripts use the `name` attribute to place the form data into a database or into an email that can be read by a person. Thus, if the `<input>` element is for the site visitor to enter their name into, then the `name` attribute would be `name="name"` or `name="first-name"`, etc.

The `value` attribute

Every `<input>` element should also have a `value` attribute. The value this attribute takes depends on the element it is being used on:

- In the case where the value is whatever the user types in, the `value` attribute can be set to blank — `value=""` — this will tell the processing script to just insert whatever the site visitor types into the box. If you do enter a value into the `value` attribute, it will appear in the text input as an initial value that the user can then overwrite.
- In the case of a more complex input with a specific choice of options such as a checkbox or radio button you can set the value to equal what you want the initial value to be, out of the choices.
- In cases where there is only one value so the user doesn't enter anything, such as submit or hidden, you set the value to equal the final input. For example, you can set `value="yes"` to have the initial choice of a yes/no radiobutton pair set to yes. And you'd use `value="submit"` for a submit button

Let's talk through a more in depth example of how the `value` attribute is used:

- For a start, let's think about a blank value attribute, which the user input determines the value of.
  - The code says: `<input type="text" name="first-name" id="first-name" value="" />`
  - The user inputs: Jenifer
  - The value of `first-name` is sent as `Jenifer` when the form is submitted.
- Now let's think about a predetermined value:
  - The code says: `<input type="checkbox" name="mailing-list" id="mailing-list" value="no" />`
  - The user checks the box as they wish to join the website's mailing list.
  - The value of `mailing-list` is sent as “yes” when the form is submitted.

the `<textarea>` element

After the two `<input>` elements, you can see something a bit different — the `<textarea>` element. This provides a multiple line text input area, and you can even define how many lines are available to enter text into. Note the `cols` and `rows` attributes — these are required for every `textarea` element, and specify how many columns tall and rows wide to make the text area. The values are measured in characters.

`<input type="submit">`

Last but not least, you have a special `<input>` element with the attribute `value="submit"`. Instead of rendering a one line text box for input, the `submit` input will render a button that, when clicked, submits the form to whichever target the form has specified to send its data to.

Step two: Adding structure and behaviour

But wait! When you run the first example, fill it out and click Submit, it doesn't do anything! Why? And why does it look so bad and all in one line? The answer is that we haven't structured it yet, or defined a place for the data the form is collecting to be submitted to.

Let's go back to the drawing board, with a new form:

```
<form id="contact-form" action="script.php" method="post">
    <input type="hidden" name="redirect" value="http://www.opera.com" />
    <ul>
        <li>
            <label for="name">Name:</label>
            <input type="text" name="name" id="name" value="" />
        </li>
        <li>
            <label for="email">Email:</label>
            <input type="text" name="email" id="email" value="" />
        </li>
        <li>
            <label for="comments">Comments:</label>
            <textarea name="comments" id="comments" cols="25" rows="3"></textarea>
        </li>
        <li>
            <input type="submit" value="submit" />
            <input type="reset" value="reset" />
        </li>
    </ul>
</form>
```

This form looks like Figure 2 when rendered in a browser:

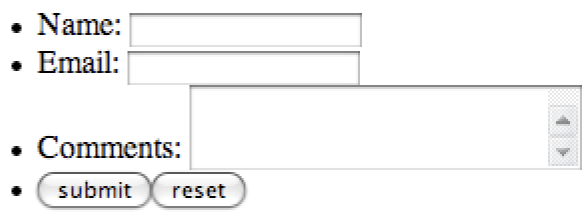


Figure 2: The second form example — looking better.

You can play with the improved form on a separate page by clicking here (http://dev.opera.com/articles /view/20-html-forms-the-basics/step-2-form.html) .

Here I have made a few additions to the basic, simple form. Let's break it down so you know what I did:

Giving the form an `id`

I added an `id` attribute to provide a unique identifier for the form so it can be more easily styled using CSS or manipulated using JavaScript if required. You can only have one of each `id` per page; in this case I called it `contact-form`.

The `method` attribute: telling the data how to travel

Lights, camera, action! When you pressed the submit button in the first form and it did not do anything, this was because it had no action or method. The `method` attribute specifies how the data is sent to the script that will process it. The two most common methods are “GET” & “POST”.

- The “GET” method will send the data in the page’s URL (you will sometimes see URLs along the lines of `http://www.example.com/page.php?data1=value1&data2=value2...`; these are bits of data being transported using the “GET” method). Unless you have a specific reason to use “GET”, it is probably best to not use it if you are trying to send secure information as anyone can see the information as it is transmitted via the URL.
- “POST” sends the data via the script that powers the form, either to an email that is sent to the site’s administrator, or a database to be stored and accessed later, rather than in the “GET” URL. "POST" is the more secure and usually the better option (http://www.w3.org/2001/tag/doc/whenToUseGet.html) .

If you are very concerned about the security of the data in the form, for example if you are submitting a credit card number to a shopping site, then you should use the https protocol with a secure socket layer (SSL). Basically, this means that data will be sent over the https protocol, not the http protocol. Have a look at the URLs next time you are paying for something on a shopping site, or using online banking — you’ll probably see `https://` in your address bar, not `http://`. The difference is that an https connection is a bit slower to transmit than http, but the data is encrypted, so anyone hacking into the data connection can’t make any sense out of it while it is in transit. Talk to your web host for information on how they can provide you with https and SSL.

The `action` attribute: telling the data where to go

The `action` attribute specifies what script file the form data should be sent to for processing. Many web hosts will have a generic send mail script or other form scripts available for use (see your host’s documentation for information) that they have customized to their servers. On the other hand, you could use a server-side script that you or someone else has created to power your form. Most of the time, folks use languages such as PHP, Perl or Ruby to create a script that will process the form — you could for example send an email containing the form information, or input the form information into a database to be stored for later use. It is outside of the scope of this part of the course to write up a server-side script for you, or teach you how to write server-side code yourself — please inquire with your host to find out what they offer, or find a nice programmer to befriend.

hidden form inputs

The second line that’s been added to our Step Two form is the “hidden” input field — this is a redirect. What? Under the goal of separating markup structure from presentation and behaviour, it is ideal to use the script that will power the form to also redirect the user once the form is submitted. You don’t want your users to be left sitting there looking at the form page, wondering what the heck to do next after they’ve submitted the form; we’re sure you’ll agree that it is a much better user experience to instead redirect your users to a thank you page featuring “what to do next” links, after a successful form submission. This particular example specifies that after this form is submitted, the user will be redirected to the Opera homepage.

Structuring with an unordered list

To improve the look of the form, we have put all the form elements into an unordered list so that we can use CSS to hook into the HTML structure and make the form look better, by lining it up and giving it some polish. Some folk would argue that you should not use an unordered list to markup a form, but use set of `<div>` elements instead. To be perfectly honest, we think either way is fine, so pick the one that suits you best.

Labels for accessibility

Last but not least in step two, we’ve labeled the form elements. It is best to give all the form elements corresponding `<label>` elements - these labels are tied to their respective form elements by giving the `<input>` and `<textarea>` elements `id`s that have the same value as the labels' `for` attributes. This is great because it not only gives a visual indicator of the purpose of each form field on the screen, but it also gives the form fields more meaning semantically. In addition, a visually impaired person using the page with a screen reader can now see which label goes with which form element and have more of a clue what data to fill into what input. The `id`s can also be used for targeting individual form fields with CSS styles.

By now you are probably wondering why `id` attributes are included as identifiers in form elements as well as `name` attributes. The answer is that `<input>` elements without `name` attributes are not submitted to the server, so those are definitely needed. `id` attributes are needed to associate form elements with their corresponding `<label>` elements. You should therefore use both.

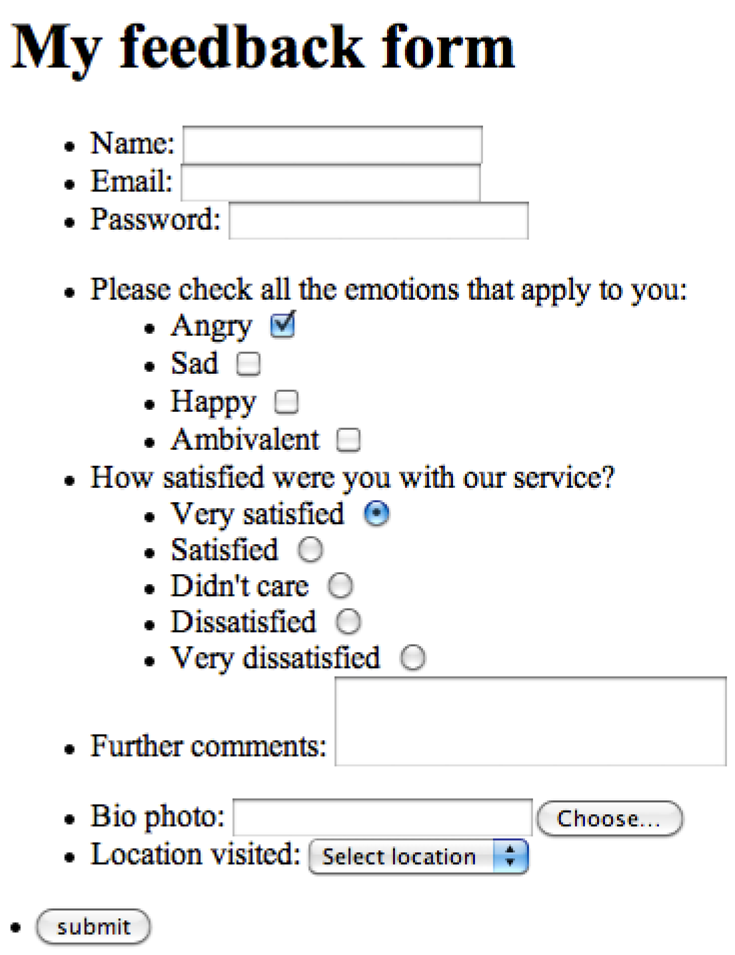
The 2nd form displays a bit better, but we could add more. Time to improve the structure further!

Step three: Adding some more complex form elements

The next version of the form looks like this:

```
<form id="contact-form" action="script.php" method="post">

<ul>
<li><label for="name">Name:</label> <input type="text" name="name" id="name" value="" /></li>
<li><label for="email">Email:</label> <input type="text" name="email" id="email" value="" /></li>
<li><label for="pwd">Password:</label> <input type="password" name="pwd" id="pwd" value="" /></li>
</ul>

<ul>
  <li>Please check all the emotions that apply to you:
    <ul>
      <li><label for="angry">Angry</label> <input type="checkbox" name="angry" id="angry" value="angry"></li
      <li><label for="sad">Sad</label> <input type="checkbox" name="sad" id="sad" value="sad"></li>
      <li><label for="happy">Happy</label> <input type="checkbox" name="happy" id="happy" value="happy"></li
      <li><label for="ambivalent">Ambivalent</label> <input type="checkbox" name="ambivalent" id="ambivalent"
    </ul>
  </li>
  <li>How satisfied were you with our service?
    <ul>
      <li><label for="vsat">Very satisfied</label> <input type="radio" name="satisfaction" id="vsat" value="v
      <li><label for="sat">Satisfied</label> <input type="radio" name="satisfaction" id="sat" value="sat"></l
      <li><label for="dcare">Didn't care</label> <input type="radio" name="satisfaction" id="dcare" value="dc
      <li><label for="disat">Dissatisfied</label> <input type="radio" name="satisfaction" id="disat" value="d
      <li><label for="vdisat">Very dissatisfied</label> <input type="radio" name="satisfaction" id="vdisat" v
    </ul>
  </li>
  <li><label for="comments">Further comments:</label> <textarea name="comments" id="comments" cols="25" rows=
</ul>

<ul>
  <li><label for="photo">Bio photo:</label> <input type="file" name="photo" id="photo" value="" /></li>
  <li><label for="location">Location visited:</label>
  <select name="location" id="location">
    <option value="">Select location</option>
    <option value="nyork">New York</option>
    <option value="vancouver">Vancouver</option>
    <option value="atlantis">Atlantis</option>
    <option value="alpha">Alpha Centauri</option>
    <option value="blackpool">Blackpool</option>
    <option value="bognor">Bognor Regis</option>
  </select></li>
</ul>

<li><input type="submit" value="submit" /></li>

</form>
```

When rendered in a browser, this form looks as shown in Figure 3.



My feedback form

- Name:
- Email:
- Password:
- Please check all the emotions that apply to you:
  - Angry ✓
  - Sad
  - Happy
  - Ambivalent
- How satisfied were you with our service?
  - Very satisfied ●
  - Satisfied
  - Didn't care
  - Dissatisfied
  - Very dissatisfied
- Further comments:
- Bio photo: Choose...
- Location visited: Select location

- submit

Figure 3: Some new, more complex form elements!

To see this form live in a browser and play with it, follow this link (http://dev.opera.com/articles/view/20-html-forms-the-basics/step-3-form.html) .

What have we added here? Let's have a look.

Checkboxes: providing multiple yes/no choices

The first thing we added is a set of checkboxes:

```
<li>Please check all the emotions that apply to you:
  <ul>
    <li><label for="angry">Angry</label> <input type="checkbox" name="angry" id="angry" value="angry" checked
    <li><label for="sad">Sad</label> <input type="checkbox" name="sad" id="sad" value="sad"></li>
    <li><label for="happy">Happy</label> <input type="checkbox" name="happy" id="happy" value="happy"></li>
    <li><label for="ambivalent">Ambivalent</label> <input type="checkbox" name="ambivalent" id="ambivalent" v
  </ul>
</li>
```

There is nothing special to know here: each separate element creates a checkbox that can be checked on and off by the site visitor. They are identified by IDs, and their data identified by their `name` attributes. When the form is submitted, any data items not checked are submitted as off/no.

Radio buttons: a multiple choice option

Next up, we have a set of radio buttons:

```
<li>How satisfied were you with our service?
```

```
  <ul>
    <li><label for="vsat">Very satisfied</label> <input type="radio" name="satisfaction" id="vsat" value="vsa
    <li><label for="sat">Satisfied</label> <input type="radio" name="satisfaction" id="sat" value="sat"></li>
    <li><label for="dcare">Didn't care</label> <input type="radio" name="satisfaction" id="dcare" value="dcar
    <li><label for="disat">Dissatisfied</label> <input type="radio" name="satisfaction" id="disat" value="dis
    <li><label for="vdisat">Very dissatisfied</label> <input type="radio" name="satisfaction" id="vdisat" val
  </ul>
</li>
```

The only thing here that is different from the checkboxes (apart from the `type` attribute value of course) is the `name` values — notice that they are all the same. This is because whereas with checkboxes you are dealing with separate items of data with on/off values, with sets of radio buttons you are dealing with a single item of data that cane take one of several values. Having the `name` attributes all set to the same value makes all these radio buttons part of the same set, and you can only select one of them at once.

The checked attribute

Note the `checked` attribute on the above two bits of code - this makes the element it is set on selected by default when the page first loads.

Uploading a file

```
<li><label for="photo">Bio photo:</label> <input type="file" name="photo" id="photo" value="" /></li>
```

This line of code creates a file uploader, allowing you to upload for example a photo.

Select and option: creating a multi-line dropdown menu

The last new bit of code looks like this:

```
<li><label for="location">Location visited:</label>
<select name="location" id="location">
  <option value="">Select location</option>
  <option value="nyork">New York</option>
  <option value="vancouver">Vancouver</option>
  <option value="atlantis">Atlantis</option>
  <option value="alpha">Alpha Centauri</option>
  <option value="blackpool">Blackpool</option>
  <option value="bognor">Bognor Regis</option>
</select></li>
```

The `<select>` element is quite different to the others you've seen: it creates a single line form control that when activated, drops down to reveal the multiple options available, as defined inside the `<option>` elements. the `value` attribute contains the actual data that is submitted when you submit the form (the only one submitted is the one you select before pressing "Submit"), and the values the user sees are the text content inside the elements.

Step four: further structuring with fieldsets and legends

The final form example (http://dev.opera.com/articles/view/20-html-forms-the-basics/step-4-form.html) is identical to the last one, except that we have wrapped the different major parts of the form in `<fieldset>` elements, and given each one its own `<legend>` element. For example:

```
<fieldset>
```

```
<legend>Login details</legend>

<ul>
<li><label for="name">Name:</label> <input type="text" name="name" id="name" value="" /></li>

  ...

</fieldset>
```

When rendered in a browser, these look as shown in Figure 4.

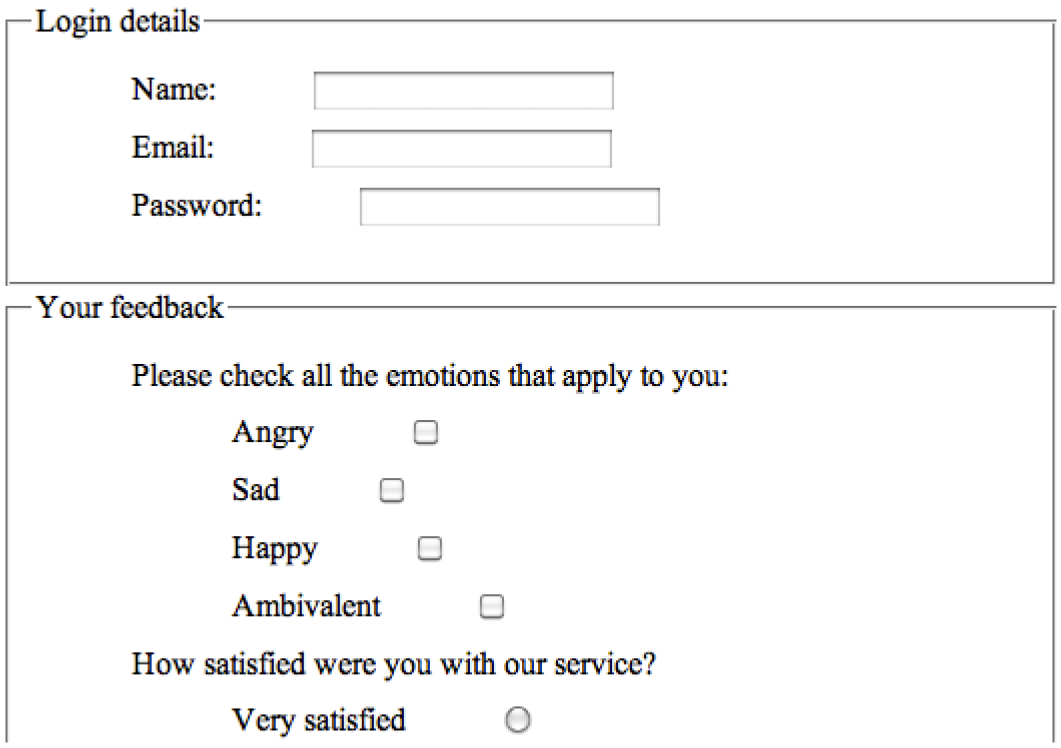

Login details
Name:
Email:
Password:
Your feedback
Please check all the emotions that apply to you:
Angry
Sad
Happy
Ambivalent
How satisfied were you with our service?
Very satisfied

Figure 4: More structuring with fieldset and legend.

Explaining fieldset and legend

`<fieldset>` and `<legend>` are not mandatory, but they are very useful for more complex forms and for presentation.

The `<fieldset>` element allows you to organize the form into semantic modules. In a more complex form, you could for example use different `<fieldset>`s to contain address information, billing information, customer preference information, and so on. The `<legend>` element allows you to add a label to each `<fieldset>` section.

A little bit of style...

I’ve also applied a little bit of CSS to this form, to style the structural markup. This is applied to the third form example using an external stylesheet — click on this link to see the styles (http://dev.opera.com/articles/view/20-html-forms-the-basics/form.css) . The two most important tasks I wanted the basic CSS to do is add margins to line up the labels and input boxes, and get rid of the unordered list’s bullet points. Here is the CSS that resides in the external stylesheet:

```
#contact-form fieldset {width:40%;}
#contact-form li {margin:10px; list-style: none;}
#contact-form input  {margin-left:45px; text-align: left;}
#contact-form textarea {margin-left:10px; text-align: left;}
```

What does it do? The first line styles the fieldset border to not take up the whole page; you could also set the border to none using `{border: none;}` if you didn’t want one. The second line puts a margin of 10 pixels on the `<li>` elements to help give a little visual room between each list item, and gets rid of the bullet points. The third and fourth lines set a left margin on the `<input>` and `<textarea>` elements so that they don’t crowd the labels and line up slightly better.

You can see that this little bit of CSS make it look better, but there is still a lot to do before it will really look nice. If you would like more information on the styling of a form please consult Styling forms or Nick Rigby’s A List Apart article on “Prettier Accessible Forms” (http://alistapart.com/articles/prettyaccessibleforms) . You will also be able to find more information on margins and borders later on in this course.

Summary

In this article, we have covered the basics of creating a standards compliant, best practice form!

Exercise questions

It's time to code your own contact form.

1. Create a simple contact form that asks the user for their name, email address, and a comment.
2. Add a checkbox asking if the reader would like to join your mailing list.
3. Use some CSS to style your form: set a width to the form, align the labels to the left, put a background colour on to your page, etc.
4. Find out what the `<optgroup>` element does, and use it in your form.