FRONT-END WEB DEVELOPER VERSION 2.1

Handout 3A: HTML Forms – The Basics

(Used in conjunction with Lesson 12 and 14 of Version 2.1)

Table of Contents

This handout is used in conjunction with Lesson 12 and Lesson 14 supported by the following videos.

Lesson 12 videos:

- 08.08 Create a basic form
- 08.09 Use HTML5 input
- 08.10 Add in other form input fields
- 08.11 Validate forms
- 08.12 Submit form data
- 08.13 Summary

Lesson 14 videos:

- R.09 Use tables to present tabular information or to align sections of text
- R.10 Use forms to send data to servers
- R.11 Collect user data with form input elements
- R.12 Apply a few miscellaneous elements
- R.13 Summary

Basic HTML Forms

Developing HTML forms is a very important part of web page development. Forms are used to gather information from others who read and interact with your web pages. Web forms allow you to receive feedback, orders, or other information the users who visit your web pages. For example, a browser search tool is a single-entry form with one button that gives you the information you are looking for when you press the button. Another example is a product order form to purchase an item, such as from Amazon.com.

This section discusses how to create forms from the front-end. The server-side handling of the forms requires knowledge of programming language beyond the scope of this course.

An HTML form is part of a web page that includes areas where users can enter information to be sent back to you, to another email address that you specify, to a databased that you manage, or to another system altogether, such as a third-party management system for your company's lead generation form (like Salesforce.com). The behind-the-scenes (server0side or backend) process requires knowledge of a programming language or you must be able to follow specific instructions when using someone else's server side script to handle the form input or data you receive. Many web hosting providers have several back-end scripts that you can use to minimize the programming required.

Form Structure

Forms include a button you click to submit forms. This button can be an image you create yourself or a standard HTML form button that is created when a form <input? Element is created and give a type value of submit.

When a person clicks a form submission button, all the information typed in the form is sent to a URL that you specify in the action attribute of the <form? Element. That URL should point to a specific script that will process your form, sending the form contents via email or performing another step in an interactive process (such as requesting results from a search engine or placing items in an online shopping cart.)

Creating a Form

Every form must begin with an opening <form? Tag. This can be located anywhere in the body of the HTML document. The <form" tag has three attributes: name, method, and action.

```
<form name="my_form" method="post" action="myprocessingscript.php">
```

The most common method is post, which sends the form entry results as a document. In some situations, you need to use method="get" which submits the results as part of the URL query string instead. Get is sometimes used when queries are submitted to search engines from a web form.

The action attribute specifies the address for sending the form data.

There are two options:

1. You can type the location of a form-processing program or script on a web server, and the form data will then be sent to that program. This is the most common scenario.
2. You can type mailto: followed by your email address, and the form data will be sent directly to you whenever someone fills out the form. This approach is dependent on the user's computer being properly configured with an email client, which may or may not be the case.

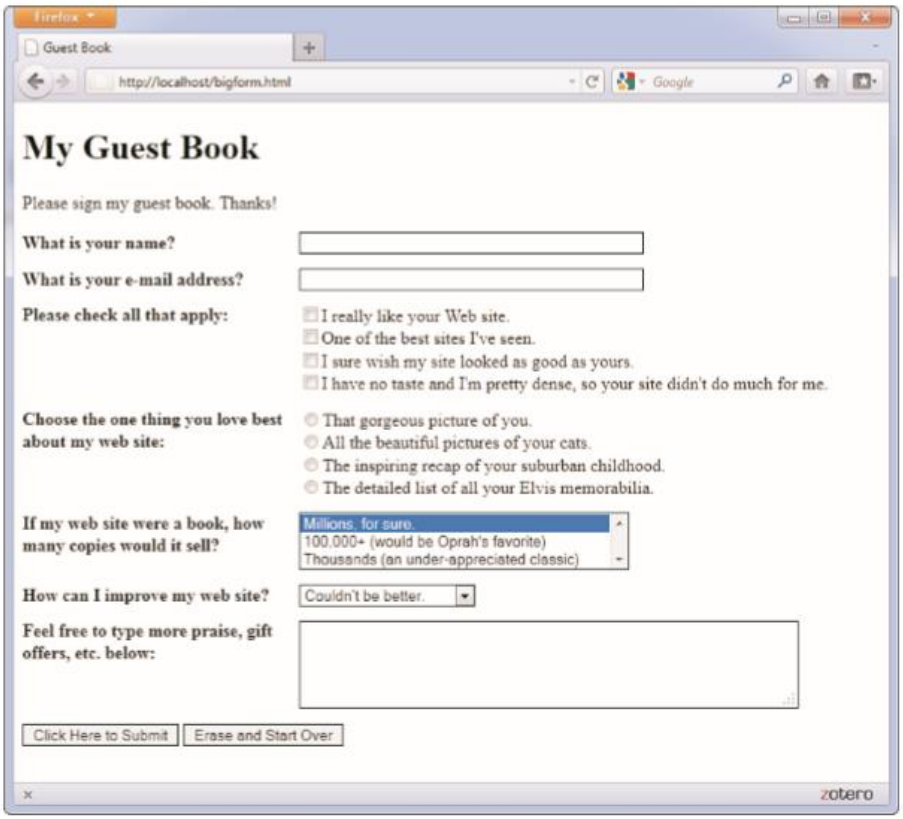
Refer to the HTML Code Listing and the following figure to see the output.

Listing 1 A Form with Various User-Input Components

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
  <head>
    <title>Guest Book</title>

    <style type="text/css">
      .formlabel {
         font-weight:bold;
         width: 250px;
         margin-bottom: 12px;
         float: left;
         text-align: left;
         clear: left;
      }
      .formfield {
         font-weight:normal;
         margin-bottom: 12px;
         float: left;
         text-align: left;
      }

      input, textarea, select {
         border: 1px solid black;
      }
    </style>
  </head>
  <body>
    <h1>My Guest Book</h1>
          <p>Please sign my guest book. Thanks!</p>
    <form name="gbForm" method="post" action="URL_to_script">

    <div class="formlabel">What is your name?</div>
    <div  class="formfield"><input type="text" name="name"
       size="50" /></div>

    <div class="formlabel">What is your e-mail address?</div>
    <div class="formfield"><input type="text" name="email"
```

```
   size="50" /></div>

<div class="formlabel">Please check all that apply:</div>
<div class="formfield">
   <input type="checkbox" name="website_response[]" value="I
   really like your Web site." />I really like your Web site.<br />
   <input type="checkbox" name="website_response[]" value="One
   of the best sites I've seen." />One of the best sites I've
   seen.<br />
   <input type="checkbox" name="website_response[]" value="I sure
   wish my site looked as good as yours." />I sure wish my site
   looked as good as yours.<br />
   <input type="checkbox" name="website_response[]" value="I have
   no taste and I'm pretty dense, so your site didn't do much for
   me." />I  have no taste and I'm pretty dense, so your site
   didn't do much for me.<br />
</div>

<div  class="formlabel">Choose the one thing you love best about my
web site:</div>
<div class="formfield">
   <input type="radio" name="lovebest" value="me" />That gorgeous
   picture of you.<br />
   <input type="radio" name="lovebest" value="cats" />All the
   beautiful pictures of your cats.<br />
   <input type="radio" name="lovebest" value="childhood" />The
   inspiring recap of your suburban childhood.<br />
   <input type="radio" name="lovebest" value="treasures" />The
   detailed list of all your Elvis memorabilia.<br />
</div>

<div class="formlabel">If my web site were a book, how many copies
would it sell?</div>
<div class="formfield">
  <select size="3" name="sales">
    <option value="Millions, for sure." selected="selected">Millions,
       for sure.</option>
    <option value="100,000+ (would be Oprah's favorite)">100,000+
       (would be Oprah's favorite)</option>
    <option value="Thousands (an under-appreciated classic)">
       Thousands (an under-appreciated classic)</option>
    <option value="Very few: not banal enough for today's
       public">Very few: not banal enough for today's
       public.</option>
    <option value="Sell? None. Everyone will download it
       for free.">Sell? None. Everyone will download it for
       free.</option>
  </select>
</div>


<div class="formlabel">How can I improve my web site?</div>
<div class="formfield">
  <select name="suggestion">
    <option value="Couldn't be better." selected="selected">Couldn't
```

```
          be better.</option>
        <option value="More about the cats.">More about the
          cats.</option>
        <option value="More about the family.">More about the
          family.</option>
        <option value="More about Elvis.">More about Elvis.</option>
      </select>
    </div>

    <div class="formlabel">Feel free to type more praise,
        gift offers, etc. below:</div>
    <div class="formfield">
      <textarea name="comments" rows="4" cols="55"></textarea>
    </div>

    <div style="float:left;">
       <input type="submit" value="Click Here to Submit" />
       <input type="reset" value="Erase and Start Over" />
    </div>
    </form>
  </body>
</html>
```



The code uses a <form> element that contains quite a few <input /> tags. Each <input /> tag corresponds to a specific user input component, such as a checkbox or radio button. The input, select, and text area elements contain borders in the stylesheet, so it is easy to see the outline of the elements in the form. You can apply all sorts of CSS to those elements.

Let's look more at the tags.

Accepting Text Input

To ask the user for a specific piece of information within a form, use the `<input />` tag. This tag must fall between the `<form>` and `</form>` tags, but it can be anywhere on the page in relation to text, images, and other HTML tags. For example, to ask for someone's name, you could type the following text followed immediately by an `<input />` field:

```
<p>What's your name? <input type="text" size="50"
        maxlength="100" name="user_name" /></p>
```

The `type` attribute indicates what type of form element to display—a simple, one-line text entry box in this case. (Each element type is discussed individually in this chapter.)

The `size` attribute indicates approximately how many characters wide the text input box should be. If you are using a proportionally spaced font, the width of the input will vary depending on what the user enters. If the input is too long to fit in the box, most web browsers will automatically scroll the text to the left.

The `maxlength` attribute determines the number of characters the user is allowed to type into the text box. If a user tries to type beyond the specified length, the extra characters won't appear. You can specify a length that is longer, shorter, or the same as the physical size of the text box. The `size` and `maxlength` attributes are used only for `type="text"` because other input types (check boxes, radio buttons, and so on) have fixed sizes.

Naming Each Piece of Form Data

No matter what type an input element is, you must give a name to the data it gathers. You can use any name you like for each input item, as long as each one on the form is different (except in the case of radio buttons and check boxes, which are discussed later in this chapter). When the form is processed by a back-end script, each data item is identified by name. This name becomes a variable, which is filled with a value. The value is either what the user typed in the form or the value associated with the element the user selected.

For example, if a user enters **Jane Doe** in the text box defined previously, a variable is sent to the form processing script; the variable is `user_name` and the value of the variable is `Jane Doe`. Form-processing scripts work with these types of variable names and values.

To use this (or other) text fields in JavaScript, remember that the text object uses the `name` attribute; you would refer to the value of the field in the previous snippet as:

```
document.formname.user_name.value
```

Including Hidden Data in Forms

Want to send certain data items to the server script that processes a form but don't want the user to see those data items? Use an `<input />` tag with a `type="hidden"` attribute. This attribute has no effect on the display; it just adds any name and value you specify to the form results when they are submitted.

If you are using a form-processing script provided by your web-hosting provider, you might use this attribute to tell a script where to email the form results. For example, including the following code will email the results to `me@mysite.com` after the form has been submitted:

```
<input type="hidden" name="mailto" value="me@mysite.com" />
```

You might sometimes see scripts using hidden input elements to carry additional data that might be useful when you receive the results of the form submission; some examples of hidden form fields include an email address and a subject for the email. If you are using a script provided by your web hosting provider, consult the documentation provided with that script for additional details about potential required hidden fields.

Form Input Controls

Check Boxes

The simplest input type is a *check box*, which appears as a small square. Users can click checkboxes to select or deselect one or more items in a group. For example, the checkboxes listed in Listing 26.1 display after a label that reads "Please check all that apply," implying that the user could indeed check all that apply.

The HTML for the check boxes in Listing 26.1 shows that the value of the `name` attribute is the same for all of them: `website_response[]`.

```
<input type="checkbox" name="website_response[]" value="I
   really like your Web site." />I really like your Web site.<br />
<input type="checkbox" name="website_response[]" value="One
   of the best sites I've seen." />One of the best sites I've
   seen.<br />
<input type="checkbox" name="website_response[]" value="I sure
   wish my site looked as good as yours." />I sure wish my site
   looked as good as yours.<br />
<input type="checkbox" name="website_response[]" value="I have
   no taste and I'm pretty dense, so your site didn't do much for
   me." />I  have no taste and I'm pretty dense, so your site
   didn't do much for me.<br />
```

The use of the brackets in the `name` attribute (`[]`) indicates to the processing script that a series of values will be placed into this one variable, instead of just one value (well, it might just be one value if the user only selects one check box). If a user selects the first check box, the text string "I really like your Web site." will be placed in the `website_response[]` bucket. If the user selects the third checkbox, the text string "I sure wish my site looked as good as yours." will also be put into the `website_response[]` bucket. The processing script will then work with that variable as an array of data rather just a single entry.

However, you might see groups of check boxes that do use individual names for the variables in the group. For example, the following is another way of writing the check box group:

```
<input type="checkbox" name="liked_site" value="yes" /> I really like
   your Web site.<br />
<input type="checkbox" name="best_site" value="yes" /> One of the best
   Sites I've seen.<br />
<input type="checkbox" name="my_site_sucks" value="yes" />I sure wish my
   site looked as good as yours.<br />
<input type="checkbox" name="am_dense" value="yes" />I have no taste and
   I'm pretty dense, so your site didn't do much for me.<br />
```

In the previous check boxes, the variable name of the first check box is `"liked_site"` and the value (if checked) is `"yes"`.

If you want a check box to be checked by default when the form is rendered by the web browser, include the `checked` attribute. For example, the following code creates two check boxes and the first is checked by default:

```
<input type="checkbox" name="website_response[]" value="I
   really like your Web site." checked="checked" />I really like
   your Web site.<br />
<input type="checkbox" name="website_response[]" value="One
   of the best sites I've seen." />One of the best sites I've
   seen.<br />
```

The check box labeled "I really like your site." is checked by default in this example. The user would have to click the check box to indicate they had another opinion of your site. The check box marked "One of the best I've seen." would be unchecked to begin with, so the user would have to click it to turn it on. Check boxes that are not selected do not appear in the form output at all.

If you want to handle values from the `checkbox` object in JavaScript, the object has the following four properties:

- `name` is the name of the check box and also the object name.
- `value` is the "true" value for the check box—usually `on`. This value is used by server-side programs to indicate whether the check box was checked. In JavaScript, you should use the `checked` property instead.
- `defaultChecked` is the default status of the check box, assigned by the `checked` attribute in HTML.
- `checked` is the current value. This is a Boolean value: `true` for checked and `false` for unchecked.

To manipulate the check box or use its value, you use the `checked` property. For example, this statement turns on a check box called `same_address` in a form named `order`:

```
document.order.same.checked = true;
```

The check box has a single method: `click()`. This method simulates a click on the box. It also has a single event, `onClick`, which occurs whenever the check box is clicked. This happens whether the box was turned on or off, so you'll need to examine the `checked` property via JavaScript to see what action really happened.

Radio Buttons

Radio buttons, for which only one choice can be selected at a time, are almost as simple to implement as check boxes. The simplest use of a radio button is for yes/no questions or for voting when only one candidate can be selected.

To create a radio button, just use `type="radio"` and give each option its own `<input />` tag. Use the same `name` for all the radio buttons in a group, but don't use the `[]` that you used with the check box:

```
<input type="radio" name="vote" value="yes" checked="checked" /> Yes<br />
<input type="radio" name="vote" value="no" /> No <br/>
```

The `value` can be any name or code you choose. If you include the `checked` attribute, that button is selected by default. No more than one radio button with the same `name` can be checked.

When designing your form and choosing between checkboxes and radio buttons, ask yourself, "Is the question being asked one that could be answered only one way?" If so, use a radio button.

As for scripting, radio buttons are similar to check boxes, except that an entire group of them shares a single name and a single object. You can refer to the following properties of the `radio` object:

- `name` is the name common to the radio buttons.
- `length` is the number of radio buttons in the group.

To access the individual buttons in JavaScript, you treat the `radio` object as an array. The buttons are indexed, starting with `0`. Each individual button has the following properties:

- `value` is the value assigned to the button.
- `defaultChecked` indicates the value of the `checked` attribute and the default state of the button.
- `checked` is the current state.

For example, you can check the first radio button in the `radio1` group on the `form1` form with this statement:

```
document.form1.radio1[0].checked = true;
```

However, if you do this, be sure you set the other values to `false` as needed. This is not done automatically. You can use the `click()` method to do both of these in one step.

Like a check box, radio buttons have a `click()` method and an `onClick` event handler. Each radio button can have a separate statement for this event.

Selection Lists

Both *scrolling lists* and *pull-down pick lists* are created with the `<select>` tag. You use this tag together with the `<option>` tag, as the following example shows (taken from Listing 26.1):

```
<select size="3" name="sales">
   <option value="Millions, for sure." selected="selected">Millions,
      for sure.</option>
   <option value="100,000+ (would be Oprah's favorite)">100,000+
      (would be Oprah's favorite)</option>
   <option value="Thousands (an under-appreciated classic)">Thousands
      (an under-appreciated classic)</option>
         <option value="Very few: not banal enough for today's public">Very
      few: not banal enough for today's public.</option>
   <option value="Sell? None. Everyone will download it for free.">Sell?
      None. Everyone will download it for free.</option>
</select>
```

No HTML tags other than `<option>` and `</option>` should appear between the `<select>` and `</select>` tags.

Unlike the `text` input type, the `size` attribute here determines how many items show at once on the selection list. If `size="2"` were used in the preceding code, only the first two options would be visible and a scrollbar would appear next to the list so the user could scroll down to see the third option.

Including the `multiple` attribute enables users to select more than one option at a time; the `selected` attribute makes an option initially selected by default. When the form is submitted, the text specified in the `value` attribute for each option accompanies the selected option.

The object for selection lists is the `select` object. The object itself has the following properties:

- `name` is the name of the selection list.
- `length` is the number of options in the list.
- `options` is the array of options. Each selectable option has an entry in this array.
- `selectedIndex` returns the index value of the currently selected item. You can use this to check the value easily. In a multiple-selection list, this indicates the first selected item.

The options array has a single property of its own, length, which indicates the number of selections. In addition, each item in the options array has the following properties:

- index is the index into the array.
- defaultSelected indicates the state of the selected attribute.
- selected is the current state of the option. Setting this property to true selects the option. The user can select multiple options if the multiple attribute is included in the <select> tag.
- name is the value of the name attribute. This is used by the server.
- text is the text that is displayed in the option.

The select object has two methods—blur() and focus()—which perform the same purposes as the corresponding methods for text objects. The event handlers are onBlur, onFocus, and onChange, also similar to other objects.

Reading the value of a selected item is a two-step process. You first use the selectedIndex property, and then use the value property to find the value of the selected choice. Here's an example:

```
ind = document.mvform.choice.selectedIndex;
val = document.mvform.choice.options[ind].value;
```

This uses the ind variable to store the selected index, and then assigns the val variable to the value of the selected choice. Things are a bit more complicated with a multiple selection; you have to test each option's selected attribute separately.

Text Fields and Text Areas

The <input type="text"> attribute mentioned earlier this chapter allows the user to enter only a single line of text. When you want to allow multiple lines of text in a single input item, use the <textarea> and </textarea> tags to create a text area instead of just a text field. Any text you include between these two tags is displayed as the default entry. Here's an example:

```
<textarea name="comments" rows="4" cols="20">Your
      message here.</textarea>
```

As you probably guessed, the `rows` and `cols` attributes control the number of rows and columns of text that fit in the input box. The `cols` attribute is a little less exact than `rows` and approximates the number of characters that fit in a row of text. Text area boxes do have a scrollbar, however, so the user can enter more text than what fits in the display area.

The `text` and `textarea` objects also have a few methods you can use:

- `focus()` sets the focus to the field. This positions the cursor in the field and makes it the current field.
- `blur()` is the opposite; it removes the focus from the field.
- `select()` selects the text in the field, just as a user can do with the mouse. All of the text is selected; there is no way to select part of the text.

You can also use event handlers to detect when the value of a text field changes. The `text` and `textarea` objects support the following event handlers:

- The `onFocus` event happens when the text field gains focus.
- The `onBlur` event happens when the text field loses focus.
- The `onChange` event happens when the user changes the text in the field and then moves out of it.
- The `onSelect` event happens when the user selects some or all of the text in the field. Unfortunately, there's no way to tell exactly which part of the text was selected. (If the text is selected with the `select()` method described previously, this event is not triggered.)

If used, these event handlers should be included in the `<input>` tag declaration. For example, the following is a text field including an `onChange` event that displays an alert:

```
<input type="text" name="text1" onChange="window.alert('Changed.');" />
```

Submitting Form Data

Forms typically include a button that submits the form data to a script on the server or invokes a JavaScript action. You can put any label you like on the submit button with the `value` attribute:

```
<input type="submit" value="Place My Order Now!" />
```

A gray button will be sized to fit the label you put in the `value` attribute. When the user clicks it, all data items on the form are sent to the email address or script specified in the form's `action` attribute.

You can also include a Reset button that clears all entries on the form so users can start over if they change their minds or make mistakes. Use the following:

```
<input type="reset" value="Clear This Form and Start Over" />
```

If the standard Submit and Reset buttons look a little bland to you, remember that you can style them using CSS. If that's not good enough, you'll be glad to know that there is an easy way to substitute your own graphics for these buttons. To use an image of your choice for a Submit button, use the following:

```
<input type="image" src="button.gif" alt="Order Now!" />
```

The `button.gif` image will display on the page and the form will be submitted when a user clicks the `button.gif` image. You can also include any attributes normally used with the `<img />` tag, such as `alt` and `style`.

The form element also includes a generic button type. When using `type="button"` in the `<input />` tag, you will get a button that performs no action on its own but can have an action assigned to it using a JavaScript event handler.

Sending Form Data by Email

One easy way to use a form is to send the results by email. To send a form's results by email, you use the "mailto" action in the form's action attribute. See the following modified version of the name and address form that sends the results by email.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
  <head>
    <title>Form Submit Example</title>
  </head>
  <body>
    <h1>Form Submit Example</h1>
        <p>Enter the following information. When you press the Send
   button, the data you entered will be send via e-mail.</p>
    <form name="form1" method="post" action="mailto:user@domain.com"
          enctype="text/plain">

    <p>NAME: <input type="text" name="name" size="50" /></p>
    <p>ADDRESS: <input type="text" name="address" size="50" /></p>
    <p>PHONE: <input type="text" name="phone" size="50" /></p>
    <p><input type="submit" value="Submit Form" /></p>
    </form>
  </body>
</html>
```

To use this form, change `user@domain.com` in the `action` attribute of the `<form>` tag to your email address. Notice the `enctype="text/plain"` attribute in the `<form>` tag. This ensures that the information in the email message will be in a readable plain-text format rather than encoded.

Although this provides a quick and dirty way of retrieving data from a form, the disadvantage of this technique is that it is highly browser dependent. Whether it will work for each user of your page depends on the configuration of her browser and email client.

Validating a Form

Talk to your hosting provider to see if they have a script. Or, use JavaScript.

HTML Tags for Forms

Tag/Attribute	Function
`<form>...</form>`	Indicates an input form.
Attributes	
`action="scripturl"`	The address of the script to process this form input.
`method="post/get"`	How the form input will be sent to the server. Normally set to `post`, rather than `get`.
`<input />`	An input element for a form.
Attributes	
`type="controltype"`	The type for this input widget. Possible values are `checkbox`, `hidden`, `radio`, `reset`, `submit`, `text`, and `image`.
`name="name"`	The unique name of this item, as passed to the script.
`value="value"`	The default value for a text or hidden item. For a check box or radio button, it's the value to be submitted with the form. For reset or submit buttons, it's the label for the button itself.
`src="imageurl"`	The source file for an image.
`checked="checked"`	For check boxes and radio buttons. Indicates that this item is checked.
`size="width"`	The width, in characters, of a text input region.
`maxlength="maxlength"`	The maximum number of characters that can be entered into a text region.
`<textarea>...</textarea>`	Indicates a multiline text entry form element. Default text can be included.
Attributes	
`name="name"`	The name to be passed to the script.
`rows="numrows"`	The number of rows this text area displays.
`cols="numchars"`	The number of columns (characters) this text area displays.
`<select>...</select>`	Creates a menu or scrolling list of possible items.
Attributes	
`name="name"`	The name that is passed to the script.
`size="numelements"`	The number of elements to display. If `size` is indicated, the selection becomes a scrolling list. If no `size` is given, the selection is a drop-down pick list.
`multiple="multiple"`	Allows multiple selections from the list.
`<option>...</option>`	Indicates a possible item within a `<select>` element.
Attributes	
`selected="selected"`	With this attribute included, the `<option>` will be selected by default in the list.
`value="value"`	The value to submit if this `<option>` is selected when the form is submitted.