Chapter 7: Forms for Interactivity

As we saw in Chapter 2, the hypertext technology underlying the web was originally designed for publishing information. However, the web extends the original publishing capabilities of hypertext by allowing the user to both read and enter information. Information provided by the user can be used to create much more interactive websites.

In order to create these interactive websites, we'll need to do two things. First, we'll need to provide a means for users to enter information into our webpage. Second we'll need to define how our website will respond to the information entered.

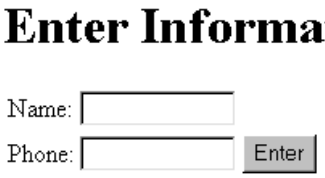
HTML *forms* allow users to enter information into a webpage. Using forms, our webpage can include most of the standard user-interface elements found in traditional computer programs. User-interface elements supported by HTML include buttons, pull-down menus, and text boxes. In computer science, user-interface elements which allow interaction between the user and the computer are often referred to as *controls*. In this textbook we'll use both the web terminology—*form elements* or *input elements*—and the more general computer science name—*controls*. In this chapter, we'll study in detail the various controls supported by HTML.

Once a user enters information onto our webpage, our webpage needs to respond to the information provided. Two general techniques are used to respond to data. The webpage itself may include a computer program along with its HTML source code. In this case, the included program responds directly as the user enters information. As an alternative, the information entered in the form may be sent back to the webserver where a program analyzes the data and creates a new webpage in response to the information received. We will look more closely at these two types of web programming in Chapter 9 and we will be studying the former method extensively from Chapter 9 onward. Both techniques require a solid understanding of programming.

In this chapter we will focus on how to create user interface elements on a webpage, deferring programming issues for now. At the end of this chapter, we'll learn how to create a webpage which sends you an e-mail message containing information a user has entered into it.

Forms

Readers enter information into a web browser through the use of HTML forms. Each page may contain one or more forms. Each form contains individual form elements, such as buttons and text fields, allowing readers to enter information. The form itself is created using the `<form>` starting and `</form>` ending tags. Between the `<form>` start and end tags, we can place any text and general HTML formatting information desired. The most important items within a `<form>`, however, are the form elements. These define the various user-interface controls which allow viewers to enter information into the webpage. Let's take a look at a webpage with a very simple form.



<form>

The <form> tag is used to group together user-interface elements on a web page. All user-interface elements on a webpage must be contained between a <form> start tag and </form> end tag. In addition to the user-interface elements described in the next section, you can put any other legal HTML inside of a form. The HTML <table> tag, for example, is often used to format the user-input elements contained in a form.

name=*string*—You can provide a form with a name. This helps identify the purpose of the form and can make access to the element easier from JavaScript.

action=*URL*—This attribute determines the action taken by the webserver when the form's contents are submitted. Forms which are not working in conjunction with a webserver should skip this attribute.

This form allows the user to enter a name and a phone number into the webpage and to click on the "Enter" button. Here is the actual HTML used to create this webpage:

```
1.  <!DOCTYPE html>
2.  <head>
3.  <meta charset="UTF-8" />
4.  <title>Simple Form Example</title>
5.  </head>
6.  <body>
7.  <h1>Enter Information</h1>
8.
9.  <form name="info">
10. <table>
11.   <tr><td>Name:</td>
12.       <td><input type="text" name="name" size="12" /></td>
13.       <td></td></tr>
14.   <tr><td>Phone:</td>
15.       <td><input type="text" name="phone" size="12" /></td>
16.       <td><input type="submit" value="Enter" /></td></tr>
17. </table>
18. </form>
19.
20. </body>
21. </html>
```

The actual text fields which allow the user to enter a name and phone number and the "Enter" button are created by <input> tags (lines 12, 15, and 16) and are laid out using an HTML table (lines 10-17). We'll be exploring the various attributes which control the type and appearance of input elements in a moment. First, let's take a brief look at the <form> tag (line 9-18).

HTML input controls are generally contained within a form. When used in conjunction with a webserver, information entered in input controls associated with a given form will be sent to the webserver in a single message. In our example webpage, both the name and phone number would ship to a webserver together, because both are contained within the same form. If we wanted different sets of information submitted to the webserver independently, we could create multiple forms on a webpage.

For example, we might create a webpage to publicize an event. Our webpage might provide one set of input controls allowing the user to request information on the event and a second set of input controls allowing the user to purchase tickets. If the webpage contained only a single form, request information and ticket purchase information would be sent to the webserver together all at once. Creating two separate forms on the webpage allows the user to independently send either an information request or a ticket purchase request.

In any case, all HTML tags associated with a specific webserver interaction should be placed between the form's `<form>` start and `</form>` end tags. Notice that all three `<input>` tags in our brief example are enclosed within the `<form>` and `</form>` start and end tags.

In addition to the actual input elements, forms may include text and formatting HTML including block formatting tags. In fact, table-based layout is commonly used to layout the various form elements in a clear and attractive manner. As you can see in our example, the `<table>`, `<tr>`, and `<td>` tags have been used to layout both the text labels (e.g., "Name:" and "Phone:") and the actual `<input>` tags which create text fields on the webpage.

Form tags only include a few attributes. The `name` attribute allows us to provide a descriptive identifying name for the form. For example we might give our information request form the name `info` and our ticket purchase form the identifying name `purchase`.

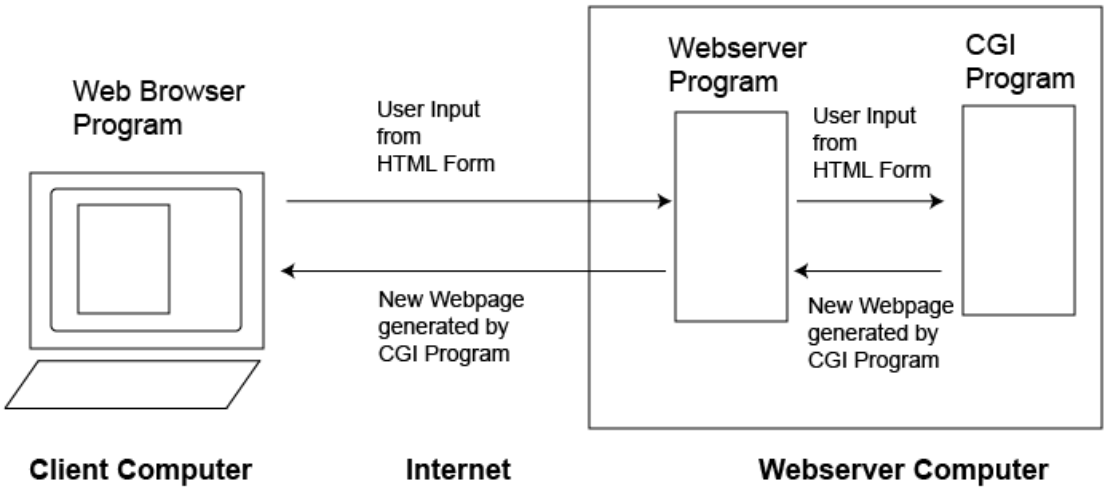
We'll be using the value of the `name` attribute later in the book to allow access to the form from JavaScript. For now, I recommend giving your forms a simple descriptive one-word identifying `name` attribute.

In Depth

CGI Programming

What happens when the user submits information to the webserver? Let's take a closer look at how the webserver might respond to a form submitted by the user.

The webserver program by itself cannot respond directly to most form submissions. The webserver program is designed to send a web browser static webpage only. This means that the webserver program can send an HTML file to a web browser, but the webserver program does not have a means of creating new HTML files. From the webserver's standpoint, the HTML files are static, they are defined by the webpage author, and once the webpage author has written them and placed them on the webserver, all the webserver program does is send them to the various web browsers on request.



The Webserver program by itself can only serve static HTML webpages. However, using the Common Gateway Interface (CGI) it can call a separate CGI program. The CGI program generates a new webpage. The Webserver program passes the webpage on to the user.

While sending static webpages works well for many purposes, it is insufficient for others. In particular when working with forms, we'll often want to create a specialized webpage just in response to the user's request. Suppose for example, we have an e-commerce webpage where the user is marking items she would like to purchase. When she completes selecting items and filling in the quantities of each item she would like to buy, she clicks on the submit button. Now, we would like her to see a shopping cart webpage which lists all the items she has chosen to purchase. Unfortunately the webserver program by itself can't generate such a webpage. Remember, the webserver program can only send webpages which have already been created. Because we had no way of anticipating the exact purchases of our user, we don't have a pre-created webpage which matches the contents of her shopping cart. What we need is a way to dynamically create webpages.

While the webserver program itself can't create dynamic webpages, it is able to call programs which can by using a special interface called the Common Gateway Interface (CGI). What CGI does is it provides an interface between the webserver program and other programs which are also running on the webserver computer. These programs, called CGI programs, generate new webpages in response to the user's form inputs.

Here's how this all works (see accompanying figure):

1. The user enters information into a form on the webpage and submits it to the webserver program.

2. The webserver program takes the information from the form and uses the action attribute on the form to determine which CGI program to call. If necessary, the webserver program starts up the CGI program. It then passes the user's form inputs to the CGI program.

3. The CGI program reads the user's form inputs and using them creates a new webpage. The new webpage is in HTML, but is created specifically in response to the user's request. This new webpage is passed to the webserver program.

4. The webserver program passes the new HTML webpage created by the CGI program on to the user's web browser. The browser displays the new webpage to the user as the results of the form submission.

CGI programs are full computer programs. They can be written in different computer languages, and can access any resources available on the webserver computer. For example, if we have a database application running on the webserver, a CGI program can look information up in the database or can store information into the database in response to the user's request.

CGI works well for the purposes for which it was designed. However, it is not always the most efficient method for responding to a user's form inputs. CGI form submission requires two trips through the Internet—one to send the form inputs to the webserver and a second to send back the HTML webpage created. In Chapter 9, we'll discover an alternative method for working with forms which eliminates these two trips.

End In Depth

Input Elements

Most of the interesting work in a form comes from the input elements contained within the form. These elements define the various text fields, pulldown menus, and buttons that actually show up on the webpage. In the next few sections, we'll be going over the different elements that can be created on a webpage. Let's use an extended example in order to place the elements in the context of their real world use.

Suppose the campus Ski Club has asked you to enhance their website. They are planning a ski trip for Winter break. They would like you to create a new webpage which allows users to sign up for the ski trip. Students will need to select from a variety of different options. For example, do they want to share a room, or have a single? How many lift tickets do they want to buy? Are they planning to drive up on their own, or would they like to reserve space on the Ski Club bus? If we design our page using forms, users can provide us with their option choices directly on the webpage.

Let's take a look at the available input elements and see how we might use them on our Ski Club webpage.

Input Tag

The `<input>` tag is a multi-purpose tag used to create most of the input elements which are found in webpage forms. We'll be using two general-purpose attributes with the `<input>` tag and along with quite a number of special purpose attributes. Our two general-purpose attributes are the `name` attribute and the `type` attribute.

The `name` attribute on an `<input>` tag serves several purposes. In many respects it works similar to the `name` attribute on `<form>` tags. Like the `name` attributes on `<form>` tags, the `name` attribute on an `<input>` tag can be used to document the purpose of the input element. Also, again as with the `<form>` tag, we'll eventually use the `<input>` tag's `name` to access the input element from JavaScript. We'll be discussing use of the `name` attribute for JavaScript in much more detail later in the quarter. The `<input>` tag's `name` attribute is also important when a form is used in conjunction with a webserver. When a user submits information in a form, each input element's name and value are sent together to the webserver. We'll take a closer look at exactly how this works at the end of this chapter.

<input />

The <input> tag is used to create single-line text boxes, buttons, check boxes, and radio buttons. There is no corresponding end tag, so all <input> tags should end with a " />". All <input> tags must be contained within a <form>.

name=*string*—You can provide an input element with a name. The name is used when the form containing the <input> element is submitted. This attribute also makes access to the element easier from JavaScript.

type=*input-type*—The type attribute determines what kind of user-interface element is displayed. Legal types are text, password, checkbox, radio, button, submit, and reset.

value=*string*—The purpose of the value attribute varies depending on the setting of the type attribute. value provides the initial text-field contents for text fields. It provides the label for button, submit, and reset buttons. It can also be used to provide values for transmission to the webserver for both checkbox and radio input elements.

Additional attributes for text input items are:

size=*integer*—The size attribute determines the length of the text field in characters. This is actually just the number of characters which will be visible at any given time. The actual total number of characters which can be typed is determined by the maxlength attribute.

maxlength=*integer*—The maxlength attribute determines the maximum number of characters the user can type into the text field.

Both checkbox and radio input items also support the checked attribute:

checked—If the checked attribute is provided the radio button or checkbox is initially marked as checked when the web page is first displayed. The checked attribute has no corresponding value so use checked="checked".

The <input> tag supports onclick and onchange attributes for JavaScript. As with the other attributes, usability of these attributes will depend on the type setting of the <input> tag. For more information, see Chapter 15 or Appendix A.

The most important attribute on an <input> tag is its type attribute. An <input> tag can be used to create a wide-range of different input controls on a webpage. Text fields, buttons, and checkboxes are all created using the <input> tag. An <input> tag's type attribute determines exactly which kind of control is represented by the tag. For example, if type="text", a text field appears on the webpage, whereas, if type="button", a button appears on the webpage.

We will now survey the various settings available for the type attribute. **Note:** while our examples will show the <input> tag only, don't forget, each HTML <input> tag must ultimately be enclosed within a <form> start and </form> end tag.

type="text"

Our first input element is the text field. The text field is created using the `<input>` tag with the `type` attribute set to `text`. The text field provides us with a simple one-line text box in which the user can enter a line of text. We can use text fields any time we need the user to enter words or numbers. For example, on our Ski Club webpage, we might provide a text field allowing the reader to specify his or her name. The HTML source code

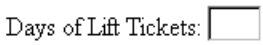
```
Name: <input name="username" type="text" />
```

will show up on the webpage as

We can control the width of the text field using the `size` attribute[1] and the maximum number of characters which may be entered using the `maxlength` attribute. If the user tries to enter more characters than the `size`, the text box won't have enough space to show all the characters. Instead previously entered text will scroll off-screen towards the left as new characters are entered on the right. All characters typed will be entered, they just will not all be visible at once. In contrast, if the user tries to enter more characters than the `maxlength`, the browser will ignore any additional characters typed. Here is an input element using both these attributes:

```
Days of Lift Tickets: <input name="days" type="text"
                            size="3" maxlength="1" />
```

In this example, I'm only allowing the user to enter a single character since `maxlength="1"`. However, setting `size="3"` makes the text field a bit wider than it would if I had set `size="1"`. If I set `maxlength="1"`, but don't set a `size`, even though the user will only be able to type one character, the field will appear on the webpage using the standard text field width (which on most web browsers is approximately large enough to fit 20 characters). With `size="3"` and `maxlength="1"` our text field looks something like this:



In some cases, we'll want to start our text field out with an initial value. For example, we might want to place an initial value in our credit card text field to show users the format we want them to use. We can do this using the `value` attribute.

```
Credit Card Number: <input name="credit" type="text" size="19"
                           value="0000 0000 0000 0000" />
```

will be displayed as

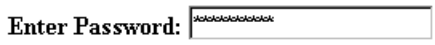
[1]Internet Explorer uses a variable spaced font for its text fields. The actual width of characters will depend on the characters typed—for example the three characters "WWW" will take up much more space than the characters "lll". Because of this, depending on the actual characters entered, the size of an Internet Explorer text field may appear to be larger or smaller than the requested size.

type="password"

Creating an <input> tag with type="password" works exactly the same as creating one with type="text", except that as the user types text into the text field, the text field displays asterisks '*' instead of showing the actual text. Thus if someone is watching over the user's shoulder they will not be able to see the actual information entered into the password field. If we define our password field using:

```
Enter Password: <input name="password" type="password" />
```

Here is an example of what our password input element would look like if someone typed in the word "JavaScript":



Using the password <input> element only prevents someone from reading the information from the screen, it does not change how the information is sent to the webserver. Unless your webpage and webserver are setup for encrypted transmissions, the information entered in a password field, like the information entered in a normal text field, is not encrypted as it is sent over the Internet. While unlikely, you should be aware that it is possible for someone to intercept and read the information as it passes from client to server.[2] See the exploring "Cryptography and Network Security" discussion at the end of this chapter for more information on encryption.

type="checkbox"

We can create checkboxes by setting type to checkbox. Checkboxes are generally used on a webpage to provide users with a number of different options where as many or as few of the options can be chosen as desired. For example, the Ski Club might give participants the option of either eating with the Ski Club or providing their own dinner on each night of the ski trip. Participants can choose any combination of meals. Here's our HTML source code:

```
<b>Meals desired ($8 each):</b>
<table>
<tr><td>Friday Dinner</td>
    <td><input name="fridayDinner" type="checkbox" /></td></tr>
<tr><td>Saturday Dinner</td>
    <td><input name="saturdayDinner" type="checkbox" /></td></tr>
<tr><td>Sunday Dinner</td>
    <td><input name="sundayDinner" type="checkbox" /></td></tr>
</table>
```

Note that in this case we're using an HTML table to align the various checkboxes. Here's what the actual webpage looks like:

[2] Most commercial websites use encryption to send and transmit confidential information such as credit card numbers. Use of encryption requires special webservers and use of the HTTPS protocol. Check with your university or ISP to see if they provide this capability.

Meals desired ($8 each):

Friday Dinner ☐

Saturday Dinner ☐

Sunday Dinner ☐

In some cases, you'll want to start off with a checkbox already checked. The user will need to explicitly uncheck the box if they don't want the associated option. We can do this by using the `checked` attribute. This attribute works a bit differently from previous attributes we've used because there is no corresponding value—if the checked attribute is present the checkbox is checked. However, in XHMTL all attributes must have a value, when we have an attribute without an actual value we instead use the attribute's name as a value. In this case we use `checked="checked"`. Here's an example using `checked`:

```
<input name="insurance" type="checkbox" checked="checked" />
Insurance ($5) <em>Highly Recommended</em>
```

which will show up as

☑ Insurance ($5) *Highly Recommended*

type="radio"

Radio buttons are used when a user may check one and only one of several different choices. For example, our ski trip participants may decide that they want a single room or that they want to save money by sharing a double room—obviously it doesn't make sense for them to say that they want both a single room and a double room at the same time. Here's what our Ski Club radio buttons might look like:

```
<input name="room" type="radio" checked="checked" /> Single ($80/night)
<input name="room" type="radio" /> Double ($40/night)
```

which will be displayed as:

◉ Single ($80/night) ○ Double ($40/night)

Radio buttons are named after the preset station buttons on a car radio. While your car stereo may allow you to store five different stations as presets, only one of the stations may be active at once. If you're listening to one station, and press another station's button, the previous selection stops playing and the new selection begins playing. Computer radio buttons work similarly. If the user has selected the Single room option and presses on the Double option, the Double option becomes active and the Single room option is automatically cleared. Only one of the two options can be chosen. If we were to use checkboxes instead of radio buttons, the user could check both options—single *and* double—at once.

The `checked` attribute determines which radio button is initially checked off. If you don't set any of your radio buttons to checked, none of them will initially be checked and your webpage will look like this:

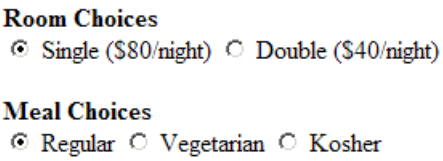
○ Single ($80/night) ○ Double ($40/night)

This is almost never what you want, as it gives your reader the opportunity to inadvertently forget to fill out part of your form—do they want a single or a double, what do we do if they forget to check of either option? So don't forget to start off one of your options as checked by adding a `checked` attribute.

The web browser uses the `name` attribute on each `<input>` tag to determine which sets of radio buttons represent mutually exclusive choices. Consider for example the following HTML code:

```
<p>
<b>Room Choices</b><br />
<input name="room" type="radio"
       checked="checked" /> Single ($80/night)
<input name="room" type="radio" /> Double ($40/night)
</p>
<p>
<b>Meal Choices</b><br />
<input name="meal" type="radio" checked="checked" /> Regular
<input name="meal" type="radio" /> Vegetarian
<input name="meal" type="radio" /> Kosher
</p>
```

which is displayed as:



Here the user is presented with two sets of mutually exclusive choices. They can choose one type of room assignment *and* they can choose one type of meal. How does the computer distinguish between the different sets of options? It checks the `name` attribute on each tag and notices that the first two radio buttons have the name "room" and that the last three have the name "meal". Therefore it determines that one of the first two buttons may be pressed and one of the last three may be pressed. If we gave all five radio button elements the *same* name, the user would be able to choose either a type of room or one of our three meal types—clicking on any one of the five options would clear away any previous room or meal choices. In contrast if we didn't give any of the elements names, the user could click and select all five options at once.

If you're using the form in conjunction with a webserver, you'll need to set the `value` attribute on each of your radio buttons. When the user submits the form the `name` and `value` will be sent to the webserver. For example if we write:

```
<input name="room" type="radio" value="single" /> Single ($80/night)
<input name="room" type="radio" value="double" /> Double ($40/night)
```

the webserver will be told either that the value of room is single or double. With no `value` attribute set, your webserver won't be able to determine the user's selection for a particular set of radio buttons. We'll take a closer look at this when we consider e-mail form submission at the end of this chapter.

type="button"

The button type allows us to create a pushbutton on the webpage. We'll need to use the value attribute to place a label on the button. As we will see in Chapter 9, buttons are typically used to begin execution of a JavaScript function. Our Ski Club webpage might include a button which, when clicked, informs the user of the total trip cost based on the options selected. We can create the button as follows:

```
<input type="button" value="Total Cost" />
```

which will be displayed as:

In general our buttons won't need a `name` attribute. We will defer a discussion on how to connect a button to a JavaScript program until Chapter 9.

type="submit"

The submit type creates a special button which is used to submit the information the user has entered in the form to a webserver computer. As with the standard button type, the `value` attribute can be used to place a label on the submit button. If no `value` is provided, the browser will attempt to come up with a suitable button label (typically "Submit" or "Submit Query").

Our Ski Club webpage will need a submit button to allow users to submit their trip information. Here is the HTML for our submission button:

```
<input type="submit" value="Submit Order" />
```

which is displayed on the webpage as:

type="reset"

The reset type creates another special button. This button clears all information entered by the user and returns the form to its initial state. Like the submit button, the browser will attempt to come up with a suitable button label, or the `value` attribute may be used to provide a button label. Here is the HTML for our Ski Club's reset button:

```
<input type="reset" value="Reset Order" />
```

which will be displayed as:

<textarea>

While the <input> tag's text input element is used to create a single-line text input item, the <textarea> tag creates a multi-line text area.

Attributes for <textarea> include:

name=*string*—As with the <input> tag, providing a name makes the textarea's data easier to access from JavaScript.

rows=*integer*—This attribute determines the number of rows of text visible in the <textarea>. This sets the height of the text area. *In XHTML this attribute is required.*

cols=*integer*—This attribute determines the number of columns of text visible in the <textarea>. This sets the width of the text area. *In XHTML this attribute is required.*

The rows and cols attributes only determine the amount of text displayed at once. The <textarea> provides scrollbar support in case the user decides to provide more information than will fit in the allotted space.

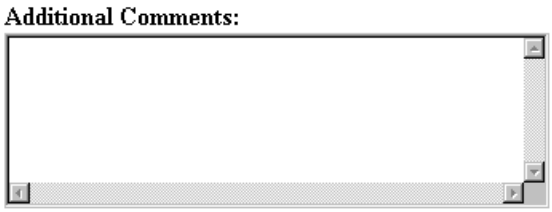
Like the <input> tag with type="text", the <textarea> has an onchange attribute which can be used for JavaScript programming.

Text Area Tag

The <input> tag with type="text" creates a single-line text entry field. In some cases, however, the user will need to enter more text than can fit easily in a single-line text field. In this case, we'll need to use the <textarea> tag instead of the <input> tag. In contrast to the <input> tag, there is a <textarea> start tag and a </textarea> end tag, and the end tag is required. In XHTML you must also specify the size of the text area by providing rows and cols attributes. These attributes determine the width and height of the text area. Their measurements are in characters. In addition, the <textarea> tag, like the <input> tag will typically include a name attribute. Here is an example of a 40-character wide by 5-character high text area created using the <textarea> tag:

```
<b>Additional Comments:</b><br />
<textarea name="comments" cols="40" rows="5">
</textarea>
```

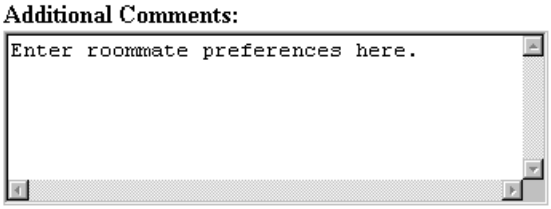
This HTML will be displayed on the webpage as:



If you want the text area to start with text area entered, you can enter the initial text desired between the `<textarea>` start and `</textarea>` end tags. For example:

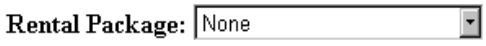
```
<b>Additional Comments:</b><br />
<textarea name="comments" cols="40" rows="5">
Enter roommate preferences here.
</textarea>
```

will start with initial text as follows:

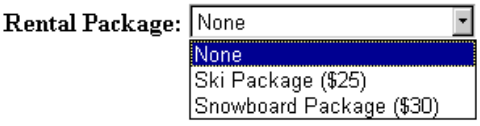


Select and Option Tags

The `<select>` tag in conjunction with the `<option>` tag is used to create *pulldown menus*. For example, we might want our website to include a pulldown menu of different ski package options. The menu will look like this



until the user moves the mouse cursor on top of it and clicks on the down arrow button. Then the menu will deploy displaying the ski package options as follows



The menu itself is created using the `<select>` tag. Individual menu items are created using the `<option>` tag. In this case, our pulldown menu was created using the following HTML code:

```
<select name="rentalPackage">
  <option>None</option>
  <option>Ski Package ($25)</option>
  <option>Snowboard Package ($30)</option>
</select>
```

The Select Tag

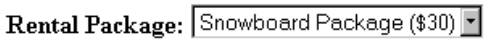
The `<select>` tag instruct the web browser to create a pulldown menu. Like other form elements, the `<select>` tag has a `name` attribute which may be used to access information on the element from JavaScript or in conjunction with a webserver.

The Option Tag

Each individual menu item is created using the `<option>` tag. The `selected` attribute may be added to an `<option>` to make it the initial menu selection. As with the `<input>` tag's `checked` attribute the `selected` attribute has no real corresponding value—because XHTML requires all attributes to have values, we write `selected="selected"`. Here is an example of the `selected` attribute in actual usage. Suppose our Ski Club has lots of snowboarders, we could create our pulldown menu using:

```
<select name="rentalPackage">
  <option>None</option>
  <option>Ski Package ($25)</option>
  <option selected="selected">Snowboard Package ($30)</option>
</select>
```

Now our pulldown menu is initially displayed with the Snowboard Package selected:



The `<option>` tag also has an optional `value` attribute which may be useful if the pulldown menu is used in conjunction with a webserver. If the chosen `<option>` tag includes a `value` attribute, the `value` will be sent to the webserver. For example if we define our pulldown menu as follows:

```
<select name="rentalPackage">
  <option value="none">None</option>
  <option value="ski">Ski Package ($25)</option>
  <option value="snowboard">Snowboard Package ($30)</option>
</select>
```

and the user selects the snowboard package, the webserver will be informed that "`rentalPackage`" element is set to "`snowboard`". If values are not supplied, the text of the option will be used as the option's value—for example the webserver might be informed that the "`rentalPackage`" is set to "`Snowboard Package ($30)` ".

<select>

The <select> tag can be used to present a list of items for selection to the user. Items in the list are specified by placing <option> tags (described below) between the <select> tag and its corresponding </select> end tag. The list can be presented in a variety of ways depending on the attributes provided.

Attributes include:

name=*string*—Providing the <select> tag with a name makes it easier to access the selection from JavaScript.

size=*integer*—The value of the size attribute determines how many option items are displayed at once. If size is one, the select input element will show up as an in-place pull-down menu. Otherwise the select input element will show up as a scrollable list of options.

multiple—Providing this tag notifies the browser that the user may select more than one item from the options provided. If multiple is set, the select input element will always show up as a scrollable list. The multiple attribute has no corresponding value so use multiple="multiple".

The <select> element includes an onchange attribute for use with JavaScript.

<option>

The <option> tag is used to provide options for a <select> user-input element.

selected—If selected is set, the option will show up as initially selected in the <select> user-input element. The selected attribute has no corresponding value so use selected="selected".

value=*string*—The value attribute may be used to define the value sent to a webserver when the option is selected. If no value is provided, the text enclosed in the <option> tag will be used. This attribute may also be used in conjunction with JavaScript.

The <option> tag also supports an onselect tag which we may use for JavaScript.

List Boxes

The <select> and <option> tags can be used to create a list box of items instead of a pulldown menu. If the <select> tag includes a size attribute with size set to 2 or more, the browser will display a list of items instead of a pulldown menu. Naturally, the size will determine how many items are displayed in the list, with a scroll bar displayed as needed.

Here is an example of our same set of rental options, displayed using a list box:

```
<select name="rentalPackage" size="3">
  <option>None</option>
  <option>Ski Package ($25)</option>
  <option>Snowboard Package ($30)</option>
</select>
```

This is displayed on the webpage as

Multiple Selections

With a pulldown menu we can only choose one item at a time. Once our selection's options are displayed as a list box instead of as a pulldown menu, we can allow the user to select more than one option at once. This won't always make sense—it's unlikely for example, that a Ski Club member would want to rent both a ski package and a snowboard package. The default behavior for both pulldown menus and list boxes is one selection only. By adding the `multiple` attribute to our `<select>` tag, we allow the user to select multiple list items. The `multiple` attribute doesn't have a corresponding value, so we write it as `multiple="multiple"`.

Let's take a look at an example:

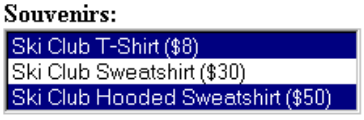
```
<b>Souvenirs:</b><br />
<select name="souvenirs" size="3" multiple="multiple">
  <option>Ski Club T-Shirt ($8)</option>
  <option>Ski Club Sweatshirt ($30)</option>
  <option>Ski Club Hooded Sweatshirt ($50)</option>
</select>
```

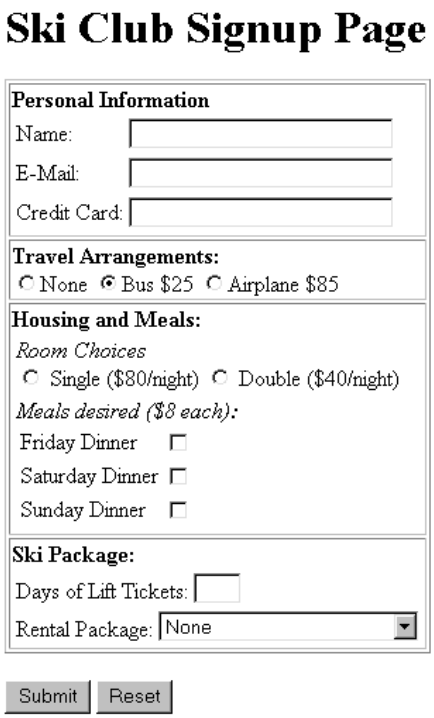
will create the following list box:

The user can select multiple items using a combination of the keyboard and the mouse. Unfortunately, the selection technique varies from computer to computer. Generally, clicking on an item with the SHIFT key down or with the CTRL or Apple Command key down will select more than one item. Here is the same list after our user has decided to purchase both a T-Shirt and a nice warm Hooded Sweatshirt:

Ski Club Example

Now that we've covered the most important input elements, let's take a look at a complete form for the Ski Club. Here's a screenshot of the form (we haven't used all the previous example elements to keep our example to a manageable size):





The actual code for this example is shown in Figure 5-1. As you can see, the webpage is laid out using a series of nested tables. The outer table consists of four rows and one column. The outer table is responsible for the border drawn around the form. Inner tables are used to align individual labels and input elements.

Notice that all input elements are contained within a `<form>` which begins before the outer table and ends after the "Submit" and "Reset" elements—remember all input elements must be contained within a `<form>` tag.

Begin In Depth

E-Mail Submissions

Typically forms on a webpage are used for one of two purposes:

- Providing a means for the user to enter information which will be processed by a local JavaScript program.
- Giving the user a method for entering information which will be submitted to a webserver.

```
<h1>Ski Club Signup Page</h1>
<form name="signup">
<table border="1">
<tr><td><b>Personal Information</b><br />
  <table>
    <tr><td>Name:</td>
        <td><input name="name" type="text" size="25" /></td></tr>
    <tr><td>E-Mail:</td>
        <td><input name="e-mail" type="text" size="25" /></td></tr>
    <tr><td>Credit Card:</td>
        <td><input name="credit" type="text" size="25" /></td></tr>
  </table>
</td></tr>

<tr><td><b>Travel Arrangements:</b><br />
  <input name="travel" type="radio" value="none" />None
  <input name="travel" type="radio" value="bus" checked="checked" />Bus $25
  <input name="travel" type="radio" value="air" />Airplane $85</td></tr>

<tr><td><b>Housing and Meals:</b><br />
  <table>
    <tr><td><i>Room Choices</i><br />
    <input name="room" type="radio" value="single" />
        Single ($80/night)
    <input name="room" type="radio" value="double" />
        Double ($40/night)
    </td></tr>
    <tr><td><i>Meals desired ($8 each):</i><br />
    <table>
      <tr><td>Friday Dinner</td>
          <td><input name="fridayDinner" type="checkbox" /></td></tr>
      <tr><td>Saturday Dinner</td>
          <td><input name="saturdayDinner" type="checkbox" /></td></tr>
      <tr><td>Sunday Dinner</td>
          <td><input name="sundayDinner" type="checkbox" /></td></tr>
    </table></td></tr>
  </table>
</td></tr>

<tr><td><b>Ski Package:</b><br />
  <table>
    <tr><td>Days of Lift Tickets:
        <input name="days" type="text" size="3" maxlength="1" /></td></tr>
    <tr><td>Rental Package:
        <select name="rentalPackage">
          <option>None</option>
          <option>Ski Package ($25)</option>
          <option>Snowboard Package ($30)</option>
        </select></td></tr>
  </table>
</td></tr>
</table>
<input value="Submit" type="submit" />
<input value="Reset" type="reset" />
</form>
```

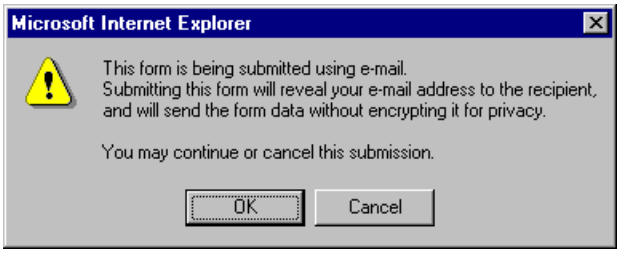
Figure 5-1: Ski Club Form

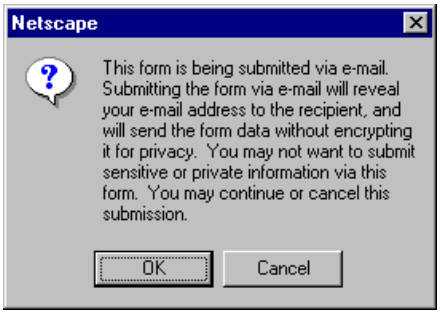
We'll be discussing how to write local JavaScript programs extensively starting in Chapter 9. Our Ski Club form is an example of this second purpose—a webpage which gathers information on the user's ski trip preferences and sends it to a webserver.

What happens to the information in our Ski Club form once the user presses the submission button? The information will be sent over the Internet to a webserver. Once the information is received, the webserver will run a program which will enter the information into a database. Unfortunately, while almost every *web browser* uses the HTML and JavaScript languages we teach in this book, the environment available on *webservers* varies widely. Some webservers use a variant of the JavaScript language which we teach in the second half of this book. Other webservers use the PERL programming language. For some purposes, you may need to use the Java programming language. Some webservers will include a sophisticated Oracle database, while others will require you to store information using basic text files.

In this book, rather than attempt to cover all possible approaches you may need on the webserver, we'll teach you a simple method which will allow your webpage to e-mail you the information entered into the form. If you find this approach too limiting and you need access to the flexibility and power provided by a real program running on the webserver, you'll need to make sure that you have a solid understanding of programming before attempting to work on the webserver. The JavaScript programming taught in the second half of this text should provide you with a strong foundation to build on.

Okay let's take a quick look at how e-mail submission works. Suppose we have the "Ski Club Signup" webpage from the previous section setup for e-mail submission. If the user fills in the form on the webpage and clicks on the submit button, the web browser will warn him that he is about to submit the information via e-mail. Here's what the warning looks like in Internet Explorer and in Netscape:





Assuming the user agrees, we will shortly receive an e-mail message from the user providing us the information entered in the form. Our e-mail message will look something like this:

```
To: psy@statecollege.edu
From: smaturin@statecollege.edu
Subject: Form Posted from Internet Explorer
----------------------
name=Stephen Maturin
e-mail=smaturin@statecollege.edu
credit=1234 5678 9876 5432
travel=bus
room=double
fridayDinner=on
saturdayDinner=on
days=3
rentalPackage=Ski Package ($25)
```

We'll need to store all of our e-mail messages in order to determine how many people are going on the ski trip and what options they've chosen.

Each e-mail message specifies all the information which the user entered on the webpage. The message lists the `name` of each input element followed by the `value` of each element. For text fields, such as the "`name`" and "`e-mail`" fields, it uses the current contents of the element as the value. Checkboxes will return either "`on`" or "`off`". Sets of radio buttons will return the `value` attribute associated with the currently selected button—the message above, for example, indicates that of the radio buttons with `name="room"`, the button with `value="double"` was chosen.

In order to get our webpage setup for e-mail submission, we'll need to modify our `<form>` tag by setting values on several new attributes.

- First of all, we'll add a `method="post"` attribute value pair. The `method` attribute controls how information passes between a client web browser and a server computer. While we don't actually have a web server computer, we do want to set the `method` to `post` for e-mail form handling.

- Next we set a value for the `action` attribute. The `action` attribute controls what actually happens when the user presses on the `submit` button. On a typical commercial website, the `<form>` tag's `action` value will be the URL for a program on the webserver computer. When the user presses the `submit` button, the information in the form is passed to the webserver program specified by the `action` attribute. The program will process the user's inputs and generate a new webpage.

  Since we don't have a webserver computer program, instead we'll use the `mailto` URL. As you may recall from Chapter 4, when the user clicks on a link using the `mailto` URL, the mail program starts up and a new message is created, using the mail address specified in the `mailto` URL. We'll be doing something similar, with the `action` attribute. When the user clicks on the `submit` button, we don't want a program to run on the webserver, instead we want the user to send us an e-mail message. We set the value of the `action` attribute to a `mailto` URL using the address where we want the information sent. For example, if we want to send the information to the e-mail address `psy@statecollege.edu`, we'll set our `action` as follows:

  ```
  action="mailto:psy@statecollege.edu"
  ```

- The web browser can send the information in a variety of formats. Most of these formats are designed to be read by a computer. Since we'll be reading the e-mail submissions manually, we'll want the information sent in as simple a manner as possible. We can set the format by setting the `enctype` (short for encoding type) attribute to `"text/plain"`. The value of this attribute is case-sensitive, so make sure you use all lower-case letters. Here's our `enctype` setting:

  ```
  enctype="text/plain"
  ```

Once we've made these three changes to our `<form>` our webpage is ready for actual use. Here's what the actual `<form>` should look like in the context of our Ski Club webpage:

```
<form name="signup"
      method="post"
      action="mailto:psy@statecollege.edu"
      enctype="text/plain">

<table border="1">

<tr><td><b>Personal Information</b><br />
```

```
  <table>
    <tr><td>Name:</td>
        <td><input name="name" type="text" size="25" /></td></tr>
    <tr><td>E-Mail:</td>
        <td><input name="e-mail" type="text" size="25" /></td></tr>
    <tr><td>Credit Card:</td>
        <td><input name="credit" type="text" size="25" /></td></tr>
  </table>
</td></tr>

…

</table>

<input value="Submit" type="submit" />
<input value="Reset" type="reset" />

</form>
```

E-mail form submission provides us with a handy method for receiving information from a form on our webpage, even when we don't have programming access to the webserver. There is one important limitation to keep in mind, however. E-mail submission requires that the user's web browser is setup to interact with the e-mail program. Depending on where the user is, this may or may not be true. The web browsers in many public access computers, such as those found in libraries and schools may not be setup to send e-mail.

End In Depth

Debugging Forms

Okay, let's go over a few hints on debugging forms before we end the chapter.

- If your form items aren't showing up at all check for either missing `</form>` or `</table>` end tags. In some browsers, missing either one will result in none of the relevant items appearing in the webpage.
- If you're working with checkboxes and radio buttons, remember, text labels telling the user the purpose of the button must be added manually. In contrast text labels for pushbuttons are provided using the `value` attribute.
- If you're working with multiple sets of radio buttons, remember that the `name` attribute on the buttons determines which sets of buttons are mutually exclusive.
- If you're having problems with a form with a `<textarea>` make sure that your `<textarea>` has a start and an end tag. In this respect, `<textarea>` does not work the same as an `<input>` tag.

Summary

In this chapter we studied HTML forms.

- HTML forms allow users to enter information into a webpage. Information entered into a form may be sent to a webserver or processed locally with a JavaScript program.

- HTML provides a variety of different input elements which may be placed on a webpage. These elements are also called controls in general computer science terminology.

- The `<input>` tag is a multi-purpose tag which can be used to create a variety of different input elements, depending on the value of its `type` attribute.

- The `<input>` tag with `type="text"` is used to create a single-line text entry element. In contrast the `<textarea>` tag is used to create a multi-line text entry element.

- The `<input>` tag can be used to create both radio buttons and checkboxes. These tags have different semantic meaning in addition to their different physical appearances. The checkbox should be use when the user can check as many or as few options as desired. In contrast the radio button is used when the user can check *only one* of the available options.

- The `<select>` tag can be used to create either a pulldown menu or a list box displaying a list of options. Individual options within the pulldown menu or list box added using the `<option>` tag.

- Most forms on the web work in conjunction with a program on the web server. However, a form can be setup to send e-mail when the user submits information on the webpage.

Review Questions

1. What is the general purpose of forms in HTML?

2. Why might you want to have multiple forms on the same webpage?

3. What is a control?

4. What is the difference between a checkbox and a radio button?

5. If we create radio buttons to represent multiple sets of mutually exclusive choices, how can we tell the web browser which radio buttons go with which sets of mutually exclusive choices?

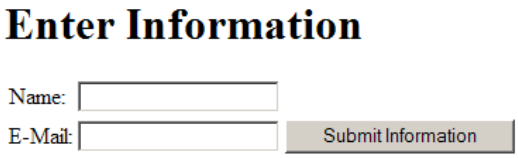
6. What is the difference between a text field and a text area?

7. What is a pulldown menu? What is a list box?

8. How do we control whether a `<select>` tag creates a pulldown menu or a list box?

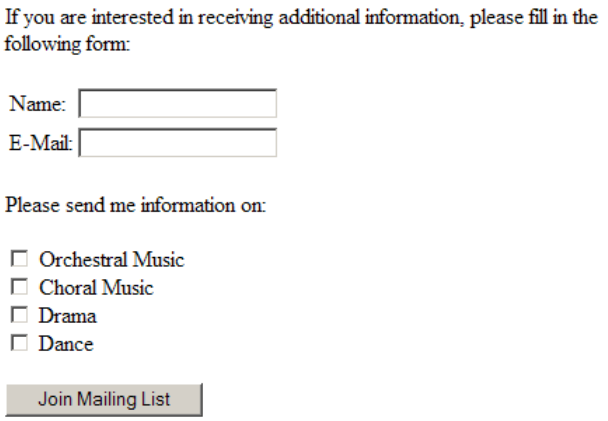
Problems

1. For our first problem we create a simple form. Here's what our form should look like:



Give the form the id and name “info”. Give the two text fields the names “name” and “eMail”. Include a submit button labeled “Submit Information”.

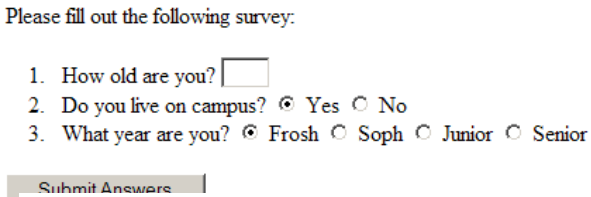
2. This next problem is very similar to the previous one, except for the addition of some checkboxes. Here is the new form:



Give this form the id and name “interest”. Give the two text fields the names “name” and “eMail” and the four checkboxes the names “orchestral”, “choral”, “drama”, and “dance”.

3. Our next form uses radio buttons. Do you remember how to make these work? Here’s our form:

I’ll let you come up with your own names for the form and form elements on this problem. Make



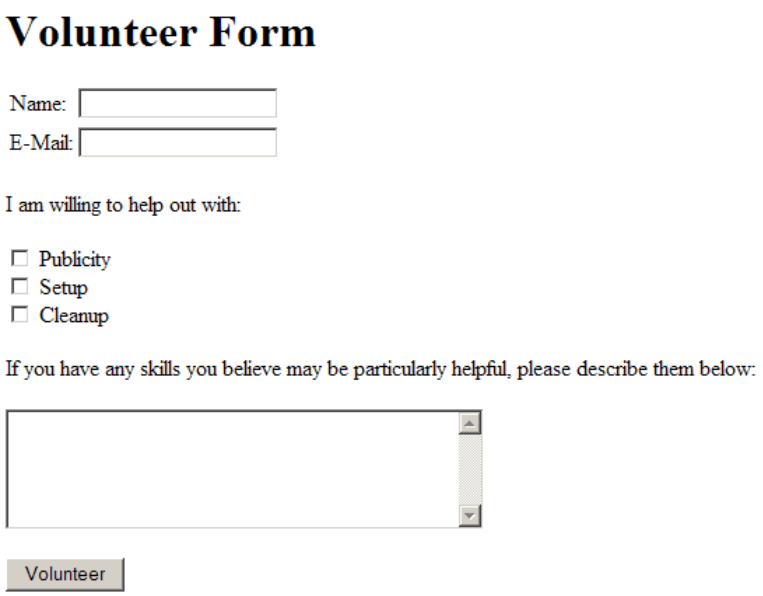
sure that the “Living on Campus” radio buttons operate independently from the “Year in School” radio buttons. Also make sure the “Yes” and “Frosh” radio buttons start of as checked.

4. This next form takes reservations. Here it is:

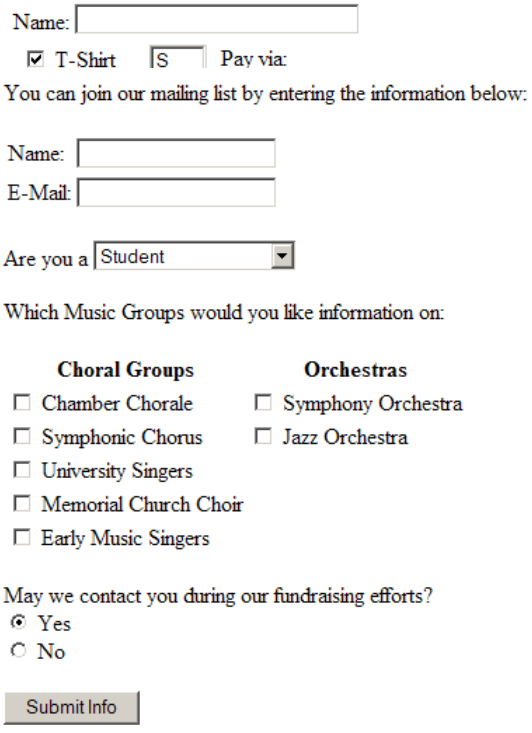
The time is specified using a pair of select tags. Allow the user to set the hours to any hour between 5 and 9, allow them to set the minutes to 00, 15, 30, or 45. As with the last problem I’ll let you come up with your own names for the form and input elements.

5. Our next form includes a text area. Set the text area to 40 columns and 5 rows. Here is the form:

6. This next form gives you an opportunity to work on both HTML forms and HTML layout. Set the



two checkboxes as initially checked. Set the list boxes initially to "L". Set the "Pay via" button initially to "Cash or Check". Set the credit card text field to have both a size and a maximum length of 16.



7. On this next form the pulldown menu provides the options "Student", "Faculty", "Staff", and "Community Member". Start the fundraising radio buttons with "Yes" marked.

8. Modify the previous problem so that it submits the form contents to you via e-mail.