

THE FORM ELEMENT

Forms are added to web pages with (no surprise here) the form element. The form element is a container for all the content of the form, including some number of form controls, such as text-entry fields and buttons. It may also contain block elements (**h1**, **p**, and lists, for example). However, it may not contain another **form** element.

This sample source document contains a form similar to the one shown in Fig. 1.1 below. FIGURE 9-1:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Mailing List Signup</title>
    <meta charset="utf-8">
  </head>
  <body>
    <h1>Mailing List Signup</h1>
    <form action="/mailinglist.php" method="POST">
      <fieldset>
        <legend>Join our email list</legend>
        <p>Get news about the band such as tour dates and special MP3
          releases sent to your own in-box.</p>
        <ol>
          <li><label for="firstlast">Name:</label>
            <input type="text" name="fullname" id="firstlast"></li>
          <li><label for="email">Email:</label>
            <input type="text" name="email" id="email"></li>
        </ol>
        <input type="submit" value="Submit">
      </fieldset>
    </form>
  </body>
</html>
```

NB: You should never nest form elements or allow them to overlap. A form element must be closed before the next one begins.

In addition to being a container for form control elements, the form element has some attributes that are necessary for interacting with the form processing program on the server. Let's take a look at each.

The action Attribute

The action attribute provides the location (URL) of the application or script that will be used to process the form. The action attribute in this example sends the data to a script called mailinglist.php: `<form action="/mailinglist.php" method="POST">...</form>`

The .php suffix indicates that this form is processed by a script written in the PHP scripting language, but web forms may be processed by any of the following technologies:

- PHP (.php) is an open source scripting language most commonly used with the Apache web server. It is the most popular and widely supported forms processing option.
- Microsoft ASP (Active Server Pages; .asp) is a programming environment for the Microsoft Internet Information Server (IIS).
- Microsoft's ASP.NET (Active Server Page; .aspx) is a newer Microsoft language that was designed to compete with PHP.

- Ruby on Rails. Ruby is the programming language that is used with the Rails platform. Many popular web applications are built with it.
- JavaServer Pages (.jsp) is a Java-based technology similar to ASP.
- Python is a popular scripting language for web and server applications.

There are other form-processing options that may have their own suffixes or none at all (as is the case for the Ruby on Rails platform).

Sometimes there is form processing code such as PHP embedded right in the HTML file. In that case, leave the action empty, and the form will post to the page itself.

The method Attribute

The method attribute specifies how the information should be sent to the server. Let's use fullname and email as data gathered from a form.

```
fullname = Jane Kibe
email = janekibe@example.com
```

When the browser encodes that information for its trip to the server, it looks like this:

```
fullname=Jane+Kibe&email=janekibe%40example.com
```

There are only two methods for sending this encoded data to the server:

POST or **GET**, indicated by the method attribute in the form element. The method is optional and will default to GET if omitted. The example uses the POST method, as shown here:

```
<form action="/mailinglist.php" method="POST">...</form>
```

The GET method

With the GET method, the encoded form data gets tacked right onto the URL sent to the server. A question mark character separates the URL from the following data, as shown here:

```
Get http://www.bandname.com/mailinglist.php?fullname=Jane+Kibe&email=janekibe%40example.com
```

GET is inappropriate if the form submission performs an action, such as deleting something or adding data to a database, because if the user goes back, it gets submitted again.

The POST method

When the form's method is set to POST, the browser sends a separate server request containing some special headers followed by the data. In theory, only the server sees the content of this request, and thus it is the best method for sending secure information such as a home address or other personal information. In practice, make sure HTTPS is enabled on your server so the user's data is encrypted and inaccessible in transit.

The POST method is also preferable for sending a lot of data, such as a lengthy text entry, because there is no character limit as there is for GET. The GET method is appropriate if you want users to be able to bookmark the results of a form submission (such as a list of search results). Because the content of the form is in plain sight, GET is not appropriate for forms with private personal or financial information. In addition, GET **may not** be used when the form is used to upload a file.

VARIABLES AND CONTENT

Web forms use a variety of controls that allow users to enter information or choose between options. Control types include various text-entry fields, buttons, menus, and a few controls with special

functions. They are added to the document with a collection of form control elements that we shall be examining next.

As a web designer, you need to be familiar with control options to make your forms easy and intuitive to use. It is also useful to have an idea of what form controls are doing behind the scenes.

The name Attribute

All form controls (except submit and reset buttons) must include a name attribute. The job of each form control is to collect one bit of information from a user. In the form example above, text-entry fields collect the visitor's name and email address. To use the technical term, "fullname" and "email" are two variables collected by the form. The data entered by the user ("Jane Kibe" and "janekibe@example.com") is the value or content of the variables. The name attribute provides the variable name for the control. In this example, the text gathered by a `textarea` element is defined as the "comment" variable:

```
<textarea name="comment" rows="4" cols="45" placeholder="Leave us a comment."></textarea>
```

When a user enters a comment in the field ("This is the best band ever!"), it would be passed to the server as a name/value (variable/content) pair like this:
`comment=This+is+the+best+band+ever%21`

All form control elements must include a name attribute so the form processing application can sort the information. You may include a name attribute for submit and reset button elements, but they are not required, because they have special functions (submitting or resetting the form) not related to data collection.

FORM CONTROLS

We now introduce the elements used to create the following:

- Text-entry controls
- Specialized text-entry controls
- Submit and reset buttons
- Radio and checkbox buttons
- Pull-down and scrolling menus
- File selection and upload control
- Hidden controls
- Dates and times
- Numerical controls
- Color picker control

The majority of controls are added to a form via the input element. The functionality and appearance of the input element changes based on the value of the type attribute in the tag. In HTML5.2, there are twenty-two types of input controls. We now discuss each in turn.

A. TEXT ENTRY CONTROLS

One of the most common web form tasks is entering text information. Which element you use to collect text input depends on whether users are asked to enter a single line of text (input) or multiple

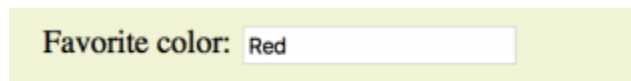
lines (textarea).

i. Single-line text field

`<input type="text">` Single-line text-entry control

One of the most straightforward form input types is the text-entry field for entering a single word or line of text. In fact, it is the default input type, which means it is what you'll get if you forget to include the type attribute or include an unrecognized value. Add a text input field to a form by inserting an input element with its type attribute set to **text**, as shown below.

```
<li><label>Favorite color: <input type="text" name="favcolor"
value="Red" maxlength="50"></label></li>
```

A screenshot of a web form. It features a label "Favorite color:" in a dark font. To the right of the label is a single-line text input field. The input field has a light gray border and contains the text "Red" in a dark font. The entire form is set against a light yellow background.

The attributes for the input tag include name, value, maxlength, minlength and size.

name

The name attribute is required for indicating the variable name.

value

The value attribute specifies default text that appears in the field when the form is loaded. When you reset a form, it returns to this value. The value of the value attribute gets submitted to the server, so in this example, the value "Red" will be sent with the form unless the user changes it.

As an alternative, you could use the **placeholder** attribute to provide a hint of what to type in the field, such as "My favorite color". The value of placeholder is not submitted with the form, and is purely a user interface enhancement.

maxlength, minlength

By default, users can type an unlimited number of characters in a text field regardless of its size (the display scrolls to the right if the text exceeds the character width of the box). You can set a maximum character limit using the maxlength attribute if the form-processing program you are using requires it. The minlength attribute specifies the minimum number of characters.

size

The size attribute specifies the length of the input field in number of visible characters. It is more common, however, to use style sheets to set the size of the input area. By default, a text input widget displays at a size that accommodates 20 characters.

ii. Multiline text-entry field

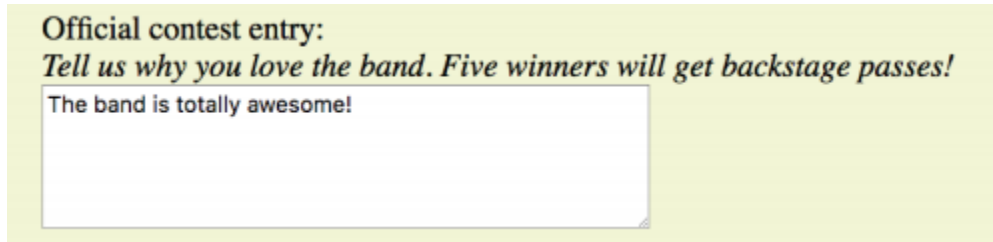
`<textarea>...</textarea>` Multiline text-entry control

At times, you'll want your users to be able to enter more than just one line of text. For these instances, use the **textarea** element, which is replaced by a multiline, scrollable text entry box when displayed by the browser.

Unlike the empty input element, you can put content between the opening and closing tags in the textarea element. The content of the textarea element shows up in the text box

when the form is displayed in the browser. It also gets sent to the server when the form is submitted, so carefully consider what you include here.

```
<p><label>Official contest entry: <br>
<em>Tell us why you love the band. Five winners will get
backstage passes!</em><br>
<textarea name="contest_entry" rows="5" cols="50">The band is
totally awesome!</textarea></label></p>
```

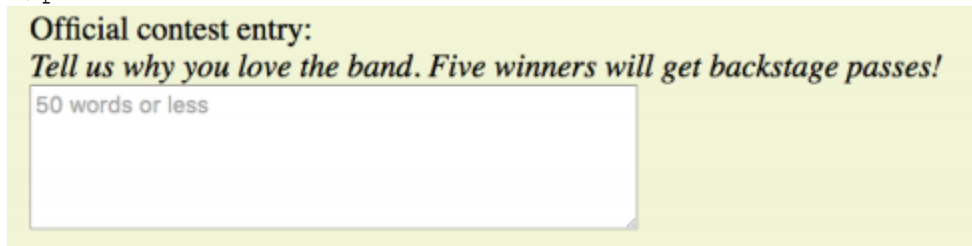


The **rows** and **cols** attributes provide a way to specify the size of the **textarea** with markup. **rows** specifies the number of lines the text area should display, and **cols** specifies the width in number of characters (although it is more common to use CSS to specify the width of the field). Scrollbars will be provided if the user types more text than fits in the allotted space.

The **maxlength** and **minlength** attributes set the maximum and minimum number of characters that can be typed into the field.

It is common for developers to put nothing between the opening and closing tags, and provide a hint of what should go there with a placeholder attribute instead. Placeholder text, unlike textarea content, is not sent to the server when the form is submitted. Examples of textarea content and placeholder text are shown below:

```
<p>Official contest entry:<br>
<em>Tell us why you love the band. Five winners will get
backstage passes!</em><br>
<textarea name="contest_entry" placeholder="50 words or less"
rows="5" cols="50"></textarea>
</p>
```



iii. Password entry field

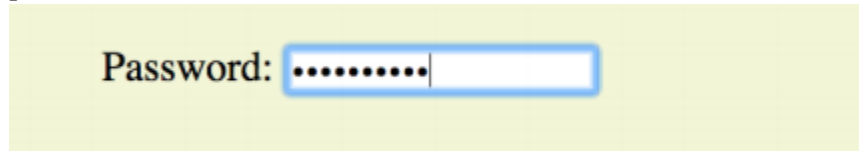
```
<input type="password"> Password text control
```

A password field works just like a text-entry field, except the characters are obscured from view by asterisk (*) or bullet (•) characters, or another character determined by the browser.

It's important to note that although the characters entered in the password field are not visible to casual onlookers, the form does not encrypt the information, so it should not be considered a real security measure.

Here is an example of the markup for a password field.

```
<li><label for="form-pswd">Password:</label><br>
  <input type="password" name="pswd" maxlength="12" id="form-
pswd"></li>
```



iv. Search, email, telephone numbers, and URLs

```
<input type="search"> Search field
<input type="email"> Email address
<input type="tel"> Telephone number
<input type="url"> Location (URL)
```

Until HTML5, the only way to collect email addresses, telephone numbers, URLs, or search terms was to insert a generic text input field. In HTML5, the **email**, **tel**, **url**, and **search** input types give the browser a heads-up as to what type of information to expect in the field. These input types use the same attributes as the generic text input type described earlier (**name**, **maxlength**, **minlength**, **size**, and **value**), as well as a number of other attributes.

All of these input types are typically displayed as single-line text inputs. But browsers that support them can do some interesting things with the extra semantic information. For example most modern mobile browsers use the input type to provide a keyboard well suited to the entry task, such as the keyboard featuring a Search button for the search input type or a “.com” button when the input type is set to url. Browsers usually add a one-click “clear field” icon (usually a little X) in search fields. A supporting browser could check the user’s input to see that it is valid—for example, by making sure text entered in an email input follows the standard email address structure (in the past, you needed JavaScript for validation). For example, the Opera and Chrome browsers display a warning if the input does not match the expected format.

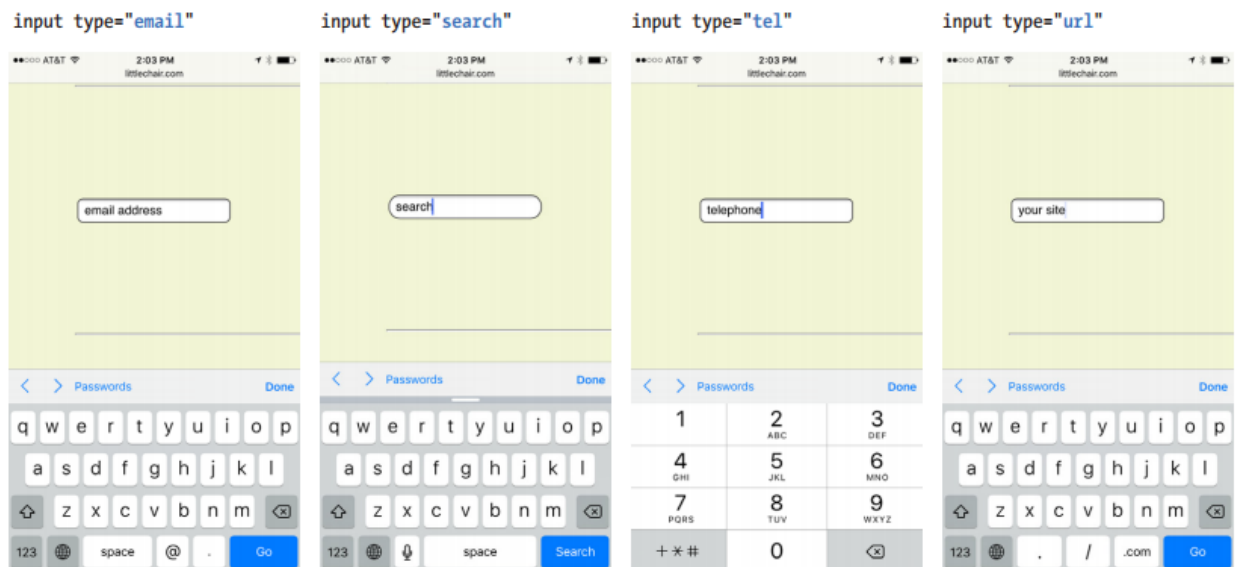


Fig. Safari on iOS provides custom keyboards based on the input type.

v. Drop-Down Suggestions

`<datalist>...</datalist>` Drop-down menu input

The **datalist** element allows the author to provide a drop-down menu of suggested values for any type of text input. It gives the user some shortcuts to select from, but if none are selected, the user can still type in their own text. Within the **datalist** element, suggested values are marked up as **option** elements. Use the list attribute in the input element to associate it with the **id** of its respective **datalist**. In the following example, a **datalist** suggests several education level options for a text input:

```
<p>Education completed: <input type="text" list="edulevel"
name="education">
<datalist id="edulevel">
  <option value="High School">
  <option value="Bachelors Degree">
  <option value="Masters Degree">
  <option value="PhD">
</datalist>
```

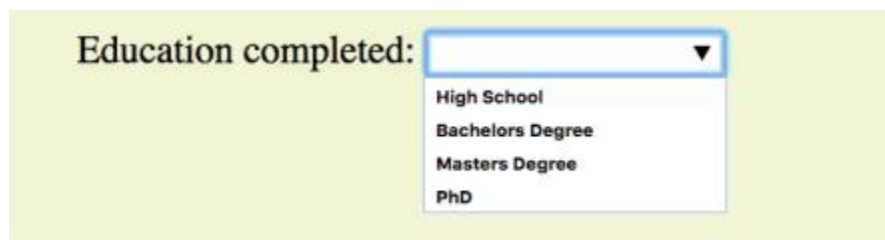


Fig. A **datalist** creates a pop-up menu of suggested values for a text-entry field.

vi. Submit and Reset Buttons

`<input type="submit">` Submits the form data to the server

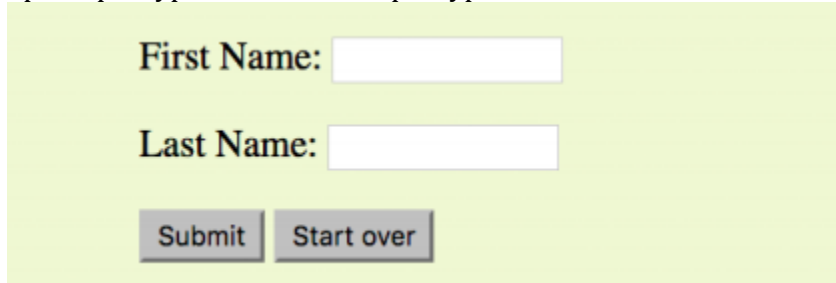
`<input type="reset">` Resets the form controls to their default settings

There are several kinds of buttons that can be added to web forms. The most fundamental is the submit button. When clicked or tapped, the submit button immediately sends the collected form data to the server for processing. A reset button returns the form controls to the state they were in when the form initially loaded. In other words, resetting the form doesn't simply clear all the fields.

Both submit and reset buttons are added via the input element. As mentioned earlier, because these buttons have specific functions that do not include the entry of data, they are the only form control elements that do not require the name attribute, although it is OK to add one if you need it.

Submit and reset buttons are straightforward to use. Just place them in the appropriate place in the form, which in most cases is at the very end. By default, the submit button displays with the label "Submit" or "Submit Query," and the reset button is labeled "Reset." You can change the text on the button by using the value attribute, as shown in the reset button in the example below?

`<p><input type="submit"> <input type="reset" value="Start over"></p>`



vii. **Radio and Checkbox Buttons**

Both checkbox and radio buttons make it simple for your visitors to choose from a number of provided options. They are similar in that they function like little on/off switches that can be toggled by the user and are added with the input element. They serve distinct functions, however.

A form control made up of a collection of radio buttons is appropriate when only one option from the group is permitted—in other words, when the selections are mutually exclusive (such as "Yes or No," or "Pick-up or Delivery"). When one radio button is "on," all of the others must be "off," sort of the way buttons used to work on old radios: press one button in, and the rest pop out.

When checkboxes are grouped together, however, it is possible to select as many or as few from the group as desired. This makes them the right choice for lists in which more than one selection is OK.

a. **Radio Buttons**

`<input type="radio">` Radio button

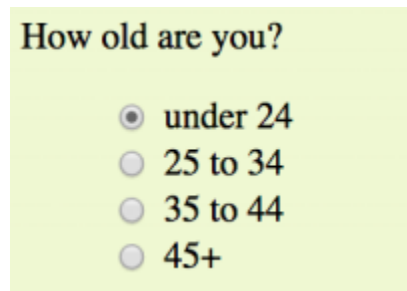
Radio buttons are added to a form via the input element with the type attribute set to "radio." Here is the syntax for a minimal radio button:

`<input type="radio" name="variable" value="value">`

The name attribute is required and plays an important role in binding multiple radio inputs into a set. When you give a number of radio button inputs the same name value (“age” in the following example), they create a group of mutually exclusive options.

In this example, radio buttons are used as an interface for users to enter their age group. A person can’t belong to more than one age group, so radio buttons are the right choice.

```
<p>How old are you?</p>
<ol>
  <li><input type="radio" name="age" value="under24" checked>
under
24</li>
  <li><input type="radio" name="age" value="25-34"> 25 to
34</li>
  <li><input type="radio" name="age" value="35-44"> 35 to
44</li>
  <li><input type="radio" name="age" value="over45"> 45+</li>
</ol>
```

A screenshot of a web form with a light green background. The title "How old are you?" is at the top. Below it are four radio button options: "under 24", "25 to 34", "35 to 44", and "45+". The "under 24" option is selected, indicated by a filled circle next to the text.

Notice that all of the input elements have the same variable name (“age”), but their values are different. Because these are radio buttons, only one button can be checked at a time, and therefore, only one value will be sent to the server for processing when the form is submitted.

You can decide which button is checked when the form loads by adding the checked attribute to the input element. In this example, the button next to “under 24” will be checked when the page loads.

b. Checkbox buttons

```
<input type="checkbox"> Checkbox button
```

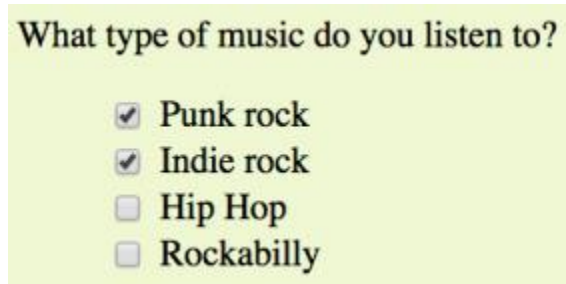
Checkboxes are added via the input element with its type set to checkbox. As with radio buttons, you create groups of checkboxes by assigning them the same name value. The difference, as we’ve already noted, is that more than one checkbox may be checked at a time. The value of every checked button will be sent to the server when the form is submitted. Here’s an example of a group of checkbox buttons used to indicate musical interests; FIGURE 9-11 shows how they look in the browser:

```
<p>What type of music do you listen to?</p>
<ul>
  <li><input type="checkbox" name="genre" value="punk"
checked> Punk
rock</li>
```

```

    <li><input type="checkbox" name="genre" value="indie"
checked> Indie
rock</li>
    <li><input type="checkbox" name="genre" value="hiphop"> Hip
Hop</li>
    <li><input type="checkbox" name="genre" value="rockabilly">
Rockabilly</li>
</ul>

```



Checkboxes don't necessarily need to be used in groups, of course. In this example, a single checkbox is used to allow visitors to opt in to special promotions. The value of the control will be passed along to the server only if the user checks the box.

```

<p><input type="checkbox" name="OptIn" value="yes"> Yes, send
me news
and special promotions by email.</p>

```

Checkbox buttons also use the **checked** attribute to make them preselected when the form loads.

viii. Menus

```

<select>...</select> Menu control
<option>...</option> An option within a menu
<optgroup>...</optgroup> A logical grouping of options within a menu

```

Another way to provide a list of choices is to put them in a drop-down or scrolling menu. Menus tend to be more compact than groups of buttons and checkboxes.

You add both drop-down and scrolling menus to a form with the select element. Whether the menu pulls down or scrolls is the result of how you specify its size and whether you allow more than one option to be selected.

Let's take a look at both menu types.

a. Drop-down menus

The select element displays as a **drop-down menu** (also called a pull-down menu) by default when no **size** is specified or if the size attribute is set to 1. In pull-down menus, only one item may be selected.

```

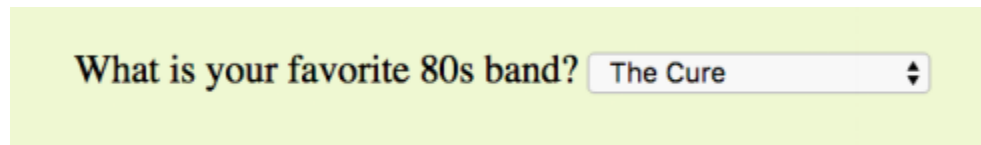
<p>What is your favorite 80s band?
<select name="EightiesFave">
  <option>The Cure</option>

```

```

    <option>Cocteau Twins</option>
    <option>Tears for Fears</option>
    <option>Thompson Twins</option>
    <option value="EBTG">Everything But the Girl</option>
    <option>Depeche Mode</option>
    <option>The Smiths</option>
    <option>New Order</option>
  </select>
</p>

```



A screenshot of a web form on a light green background. The label 'What is your favorite 80s band?' is in a bold, black, serif font. To its right is a dropdown menu with a white background and a grey border. The menu displays 'The Cure' in a black serif font, and a small upward-pointing arrow is visible on the right side of the menu box.

You can see that the **select** element is just a container for a number of **option** elements. The content of the chosen option element is what gets passed to the web application when the form is submitted. If, for some reason, you want to send a different value than what appears in the menu, use the value attribute to provide an overriding value. For example, if someone selects “Everything But the Girl” from the sample menu, the form submits the value “EBTG” for the “EightiesFave” variable. For the others, the content between the option tags will be sent as the value.

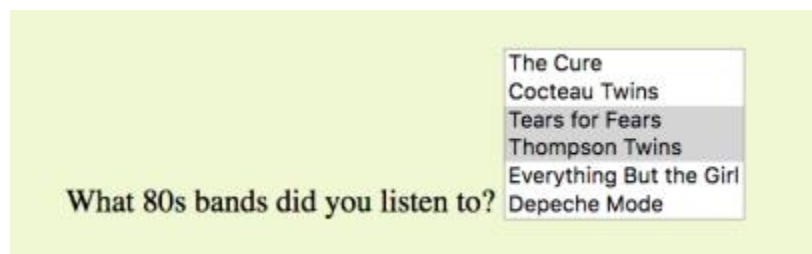
b. Scrolling menus

To make the menu display as a scrolling list, simply specify the number of lines you’d like to be visible using the size attribute. This example menu has the same options as the previous one, except it has been set to display as a scrolling list that is six lines tall.

```

<p>What 80s bands did you listen to?
  <select name="EightiesBands" size="6" multiple>
    <option>The Cure</option>
    <option>Cocteau Twins</option>
    <option selected>Tears for Fears</option>
    <option selected>Thompson Twins</option>
    <option value="EBTG">Everything But the Girl</option>
    <option>Depeche Mode</option>
    <option>The Smiths</option>
    <option>New Order</option>
  </select>
</p>

```



A screenshot of a web form on a light green background. The label 'What 80s bands did you listen to?' is in a bold, black, serif font. To its right is a scrolling list with a white background and a grey border. The list displays six options: 'The Cure', 'Cocteau Twins', 'Tears for Fears', 'Thompson Twins', 'Everything But the Girl', and 'Depeche Mode'. The options 'Tears for Fears' and 'Thompson Twins' are highlighted with a grey background, indicating they are selected.

You may notice a few minimized attributes. The **multiple** attribute allows users to make more than one selection from the scrolling list. Note that pull-down menus do not allow multiple selections; when the

browser detects the multiple attribute, it displays a small scrolling menu automatically by default.

Use the **selected** attribute in an **option** element to make it the default value for the menu control. Selected options are highlighted when the form loads. The selected attribute can be used with pull-down menus as well.

Grouping menu options

You can use the **optgroup** element to create conceptual groups of options. The required label attribute provides the heading for the group

```
<select name="icecream" size="7" multiple>
  <optgroup label="traditional">
    <option>vanilla</option>
    <option>chocolate</option>
  </optgroup>
  <optgroup label="fancy">
    <option>Super praline</option>
    <option>Nut surprise</option>
    <option>Candy corn</option>
  </optgroup>
</select>
```



ix. Hidden Controls

`<input type="hidden">` Hidden control field

There may be times when you need to send information to the form processing application that does not come from the user. In these instances, you can use a hidden form control that sends data when the form is submitted, but is not visible when the form is displayed in a browser.

Hidden controls are added via the input element with the type set to hidden. Its sole purpose is to pass a name/value pair to the server when the form is submitted. In this example, a hidden form element is used to provide the location of the appropriate thank-you document to display when the transaction is complete:

```
<input type="hidden" name="success-link"
value="http://www.example.com/thankyou.html">
```

x. Date and Time Controls

`<input type="date">` Date input control

`<input type="time">` Time input control

`<input type="datetime-local">` Date/time control

`<input type="month">` Specifies a month in a year
`<input type="week">` Specifies a particular week in a year

HTML5 introduced six new input types that make date and time selection widgets part of a browser's standard built-in display capabilities, just as they can display checkboxes, pop-up menus, and other widgets today. Date and time pickers are implemented on most current browsers.

The new date- and time-related input types are as follows:

`<input type="date" name="name" value="2017-01-14">`

Creates a date input control, such as a pop-up calendar, for specifying a date (year, month, day). The initial value must be provided in ISO date format (YYYY-MM-DD).

`<input type="time" name="name" value="03:13:00">`

Creates a time input control for specifying a time (hour, minute, seconds, fractional sections) with no time zone indicated. The value is provided as hh:mm:ss.

`<input type="datetime-local" name="name" value="2017-01-14T03:13:00">`

Creates a combined date/time input control with no time zone information (YYYY-MM-DDThh:mm:ss).

`<input type="month" name="name" value="2017-01">`

Creates a date input control that specifies a particular month in a year (YYYY-MM).

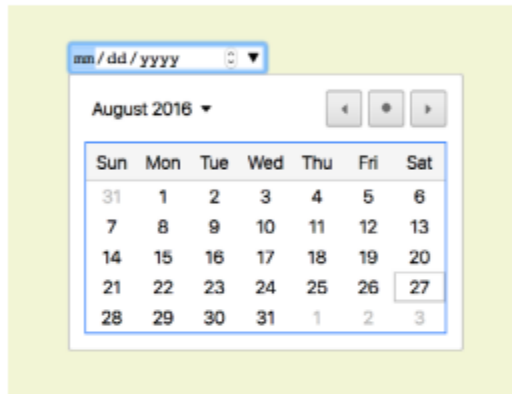
`<input type="week" name="name" value="2017-W2">`

Creates a date input control for specifying a particular week in a year using an ISO week numbering format (YYYY-W#).

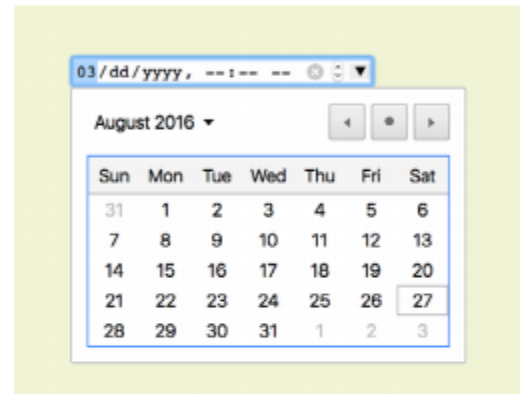
input type="time"



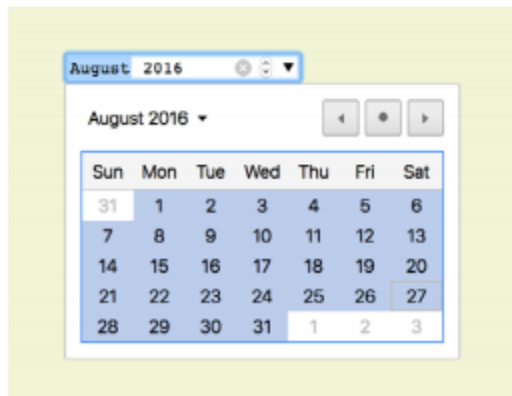
input type="date"



input type="datetime-local"



input type="month"



input type="week"

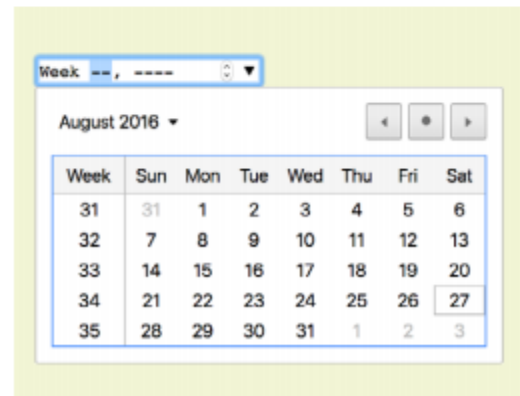


Fig. Date and time picker inputs (shown in Chrome on macOS).

xi. Numerical Inputs

<input type="number"> Number input

<input type="range"> Slider input

The number and range input types collect numerical data. For the number input, the browser may supply a spinner widget with up and down arrows for selecting a specific numerical value (a text input may display in user agents that don't support the input type). The range input is typically displayed as a slider that allows the user to select a value within a specified range:

```
<label>Number of guests <input type="number" name="guests"
min="1" max="6"></label>
```

```
<label>Satisfaction (0 to 10) <input type="range"
name="satisfaction" min="0" max="10" step="1"></label>
```

```
input type="number"
```



```
input type="range"
```

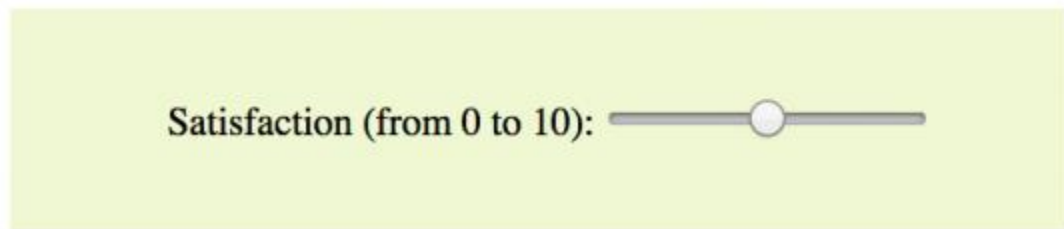


Fig. The number and range input types (shown in Chrome on macOS).

Both the number and range input types accept the min and max attributes for specifying the minimum and maximum values allowed for the input (again, the browser could check that the user input complies with the constraint). Both min and max are optional, and you can also set one without the other. Negative values are allowed. When the element is selected, the value can be increased or decreased with the number keys on a computer keyboard, in addition to being moved with the mouse or a finger.

xii. Color Selector

```
<input type="color"> Color picker
```

The intent of the color control type is to create a pop-up color picker for visually selecting a color value similar to those used in operating systems or image-editing programs. Values are provided in hexadecimal RGB values (#RRGGBB). Non-supporting browsers display the default text input instead.

```
<label>Your favorite color: <input type="color"
name="favorite"> </label>
```



Fig. The color input type (shown in Chrome on macOS).

xiii. File Selection Control

`<input type="file">` File selection field

Web forms can collect more than just data. They can also be used to transmit external documents from a user's hard drive. For example, a printing company could use a web form to upload artwork for a business card order. A magazine could use a form to collect digital photos for a photo contest.

The file selection control makes it possible for users to select a document from the hard drive to be submitted with the form data. We add it to the form by using the input element, with its **type** set to **file**.

The markup sample here (FIGURE 9-15) shows a file selection control used for photo submissions:

```
<form action="/client.php" method="POST"
  enctype="multipart/form-data">
  <label>Send a photo to be used as your online icon
  <em>(optional)</em><br>
    <input type="file" name="photo"></label>
</form>
```

The file upload widget varies slightly by browser and operating system, but it is generally a button that allows you to access the file organization system on your computer.

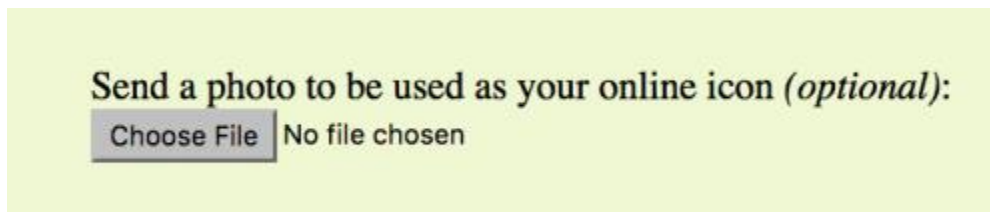


Fig. A file selection form field.

Homework:

Read on the following form elements:

progress:

percent downloaded: `<progress max="100" id="fave">0</progress>`

meter:

`<meter min="0" max="100" name="volume" value="60">60%</meter>`

output:

FORM ACCESSIBILITY FEATURES

It is essential to consider how users without the benefit of visual browsers will be able to understand and navigate through your web forms. The label, fieldset, and legend form elements improve accessibility by making the semantic connections between the components of a form clear. Not only is the resulting markup more semantically rich, but there are also more elements available to act as “hooks” for style sheet rules. Everybody wins!

Labels

Although we may see the label “Address” right next to a text field for entering an address in a visual browser, in the source, the label and field input may be separated. The **label** element associates descriptive text with its respective form field. This provides important context for users with speech-based browsers. Another advantage to using labels is that users can click or tap anywhere on them to select or focus the form control. Users with touch devices will appreciate the larger tap target.

Each **label** element is associated with exactly one form control. There are two ways to use it. One method, called implicit association, nests the control and its description within a label element. In the following example, labels are assigned to individual checkboxes and their related text descriptions. This is in fact the way to label radio buttons and checkboxes. You can’t assign a label to the entire group. Keep this in mind.

```
<ul>
  <li><label><input type="checkbox" name="genre" value="punk"> Punk
  rock</label></li>
  <li><label><input type="checkbox" name="genre" value="indie"> Indie
  rock</label></li>
  <li><label><input type="checkbox" name="genre" value="hiphop"> Hip
  Hop</label></li>
  <li><label><input type="checkbox" name="genre" value="rockabilly">
  Rockabilly</label></li>
</ul>
```

The other method, called **explicit association**, matches the label with the control’s **id** reference. The **for** attribute says which control the label is for. This approach is useful when the control is not directly next to its descriptive text in the source. It also offers the potential advantage of keeping the label and the control as two distinct elements, which you may find handy when aligning them with style sheets.

```
<label for="form-login-username">Login account</label>
<input type="text" name="login" id="form-login-username">
```

```
<label for="form-login-password">Password</label>
```

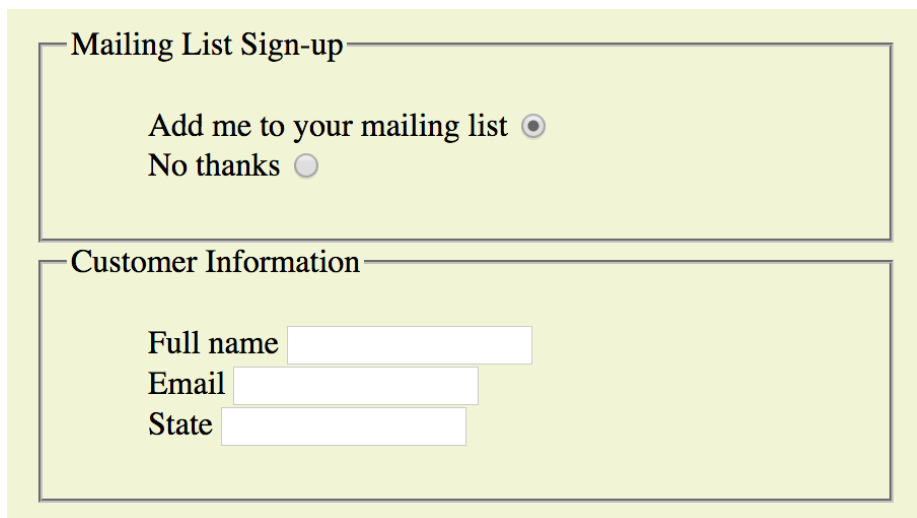
```
<input type="password" name="password" id="form-login-password">
```

fieldset and legend

The **fieldset** element indicates a logical group of form controls. A **fieldset** may also include a **legend** element that provides a caption for the enclosed fields.

The figure below shows the default rendering of the following example, but you could use style sheets to change the way the **fieldset** and **legend** appear.

```
<fieldset>
  <legend>Mailing List Sign-up</legend>
  <ul>
    <li><label>Add me to your mailing list <input type="radio"
      name="list" value="yes" checked</label></li>
    <li><label>No thanks <input type="radio" name="list" value="no">
</label></li>
  </ul>
</fieldset>
<fieldset>
  <legend>Customer Information</legend>
  <ul>
    <li><label>Full name: <input type="text" name="fullname"></label>
</li>
    <li><label>Email: <input type="text" name="email"></label></li>
    <li><label>State: <input type="text" name="state"></label></li>
  </ul>
</fieldset>
```



Mailing List Sign-up

Add me to your mailing list ☒

No thanks ☐

Customer Information

Full name

Email

State