ISTA 230

Forms

HTML

Forms

Forms are one of the most critical HTML elements in use today. They provide the backbone for most communication on the web and, as such, should be designed with the greatest of care.

Forms allow users to enter data to be processed and operated on by the server.

- Examples:
    - Contact/Email Form
    - Login Page
    - Facebook Status
    - Etc.

HTML

Forms

The <form> element is a block element. This element serves as the container for all of the individual pieces of a form. It is worth noting that all children of a form element must be block elements (though grand-children and other descendants can be inline elements).

HTML

```
<form>
   <div>

   </div>
</form>
```

HTML

Forms - Input Element

The <input> element provides us with a number of different form inputs. is a block element. Note that it is a self-closing HTML tag, much like the
 and <img /> tags.

The <input> element requires at least three attributes: name, value, and type.

The 'name' attribute is used to specify the name of the data in the form field. This attribute shouldn't contain any spaces or special characters (just like class names). Best practice is to use the same value for the 'name' attribute as you use for the 'id' attribute.

The 'value' attribute is displayed differently depending on the value of the 'type' attribute. In some cases, the 'value' attribute can be left empty (but still must be specified).

The 'type' attribute determines the appearance and behavior of the <input> element.

HTML

```
<input name="" value="" type="" />
```

There are a number of options for the 'type' attribute, each of which will be described in the following pages.

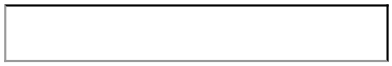
HTML

Forms - Text Input

HTML

```
<input name="username" value="" type="text" />
```

Browser



- Allows users to enter text **in a single line**.
- If 'value' is not empty, then that text is displayed by default.

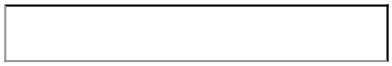
HTML

Forms - Password Input

HTML

```
<input name="pass" value="" type="password" />
```

Browser



- Similar to 'text' but characters are masked.
- Can specify a 'value' but not a great idea.

  Because users can view the HTML source of your webpage, providing a default value for a password field isn't secure, though it might seem so on the surface.

HTML

Forms - Hidden Input

HTML

```
<input name="userID" value="1" type="hidden" />
```

Browser

- Doesn't appear on the page
- Useful for storing values that are used by the website but not input by the user

HTML

Forms - Checkbox Input

HTML

```
<input name="terms" value="1" type="checkbox" />  I agree to...
```

Browser

I agree to the terms of service

HTML

Forms - Checkbox Input

- Good for yes/no responses
- 'Value' is what is submitted to the website
- Best practice: Put the checkbox before the related text.
- Can also include a 'checked' attribute

HTML

Forms - Checkbox Input

When provided, the 'checked' attribute (with a value of 'checked') makes the checkbox marked by default.

HTML

```
<input name="terms" checked="checked" value="1" type="checkbox" />
```

Browser

I agree to...

HTML

Forms - Radio Input

Using 'radio' for the type attribute, we end up with a 'radio button'. The name is derived from old-fashioned radio sets that had buttons which could only be depressed one at a time.

When creating a set of radio buttons, each input element should have a single shared 'name' attribute.

HTML

```
<input name="veggie" value="1" type="radio" /> Brocolli<br />
<input name="veggie" value="2" type="radio" /> Carrots<br />
<input name="veggie" value="3" type="radio" /> Spinach
```

Browser

Best practice is to put the radio button before the related text or label.

It is worth noting that radio buttons can also have a checked attribute which, When provided, the 'checked' attribute (with a value of 'checked'), makes the given option selected by default.

HTML

Forms - Submit Button

By specifying 'submit' for an input element's 'type' attribute, we can include a button for submitting the contents of our form to the web server.

HTML

```
<input name="submit" type="submit" value="Save Data" />
```

Browser

- When clicked, submits the form.
- 'value' is displayed as the button text

While adding a 'submit' button allows us to submit the form, it doesn't specify where the form data should be sent.

HTML

Forms - Action Attribute

When we introduced the <form> element, we failed to mention that there are two required attributes: action & method.

HTML

```
<form action="" method="">
```

The 'action' attribute is used to specify where the data should be sent. This is always a URL and can be either a relative or absolute URL, though it is typically a relative URL.

HTML

Forms - Method Attribute

HTML

```
<form action="" method="">
```

The 'method' attribute is used for specifying how the form data should be sent. This attribute is always one of two values: 'get' or 'post'.

HTML

GET method

- Sends form data as part of the URL
    - E.g., search.php?query=widgets

- Should only be used for 'safe' actions

    Safe actions are those that don't involve any sensitive data and that can be repeated without having any adverse effects.

- Limited amount of data can be passed via GET

    While there isn't a standard specification as to how long a URL can be, Microsoft Internet Explorer limits URL length to 2,048 characters.

- 'Bookmark-able'

    Because the form data is submitted as part of the URL, it can be stored by the browser as a bookmark. This is useful for actions such as search results.

HTML

POST method

- Sends form data apart from the URL

  When sent via POST, form data is sent with the HTTP headers, meaning that it is not sent as part of the URL.

- Should be used for 'unsafe' actions

  "Unsafe actions" are those that could have adverse side effects if repeated. Because browsers are required to warn users when they accidentally re-submit form data via POST, this method is good for ensuring that users don't accidentally submit purchase orders more than once.

- No size limitation

HTML

Forms - Reset Button

HTML

```
<input name="reset" type="reset" value="Reset Form" />
```

Browser

- When clicked, resets each field in the form **to their original values**.
- In other words, if 'value' attributes are set for a form field, this is what it is reset to

HTML

Forms - Textarea

Text area elements can be used to provide users with a way to submit multiple lines of text.

HTML

```
Comment:<br />
<textarea name="username" rows="5" cols="30"></textarea>
```

Browser

Comment:

Text area elements are not the same as the <input> element in that it is not self closing. There are also two required attributes: rows and cols. These attributes take an integer value, representing the number of characters wide/tall the element should be.

While the 'rows' & 'cols' attributes seemingly violates our rule of using HTML only for structure, organization, and meaning, they are required for your text area elements to be valid XHTML 1.0 Strict.

It is worth noting that the <textarea> element doesn't have a value attribute. Instead, the default value can be placed between the opening and closing <textarea> tags.

HTML

Forms - Select Menus

Select menus are created using a combination of the <select> element and one or more <option> elements.

<select> elements should have a name attribute and, when applicable, a matching id attribute.

<option> elements should have a value attribute. This value is the data that is actually sent to the server when the form is submitted, but isn't visible by the user. The text between the opening and closing <option> tags is the text that will be displayed to the user.

HTML

```
<select name="movie">
   <option value="1">Toy Story</option>



</select>
```

HTML

Forms - Select Menus

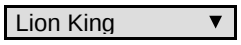
Below is a full example of a <select> element with multiple options. You'll note that we also included a 'selected' attribute for one of our options. This indicates which option should be selected by default.

While not always necessary, using the 'selected' attribute is useful if you have a large number of options but one that is selected more often than others. For example, if you have a dropdown list of states but primarily sell to customers in Arizona, you might want to make 'AZ' the default option.

HTML

```
<select name="movie">
   <option value="1">Toy Story</option>
   <option value="2" selected='selected'>Lion King</option>
   <option value="3">The Jungle Book</option>
   <option value="4">Finding Nemo</option>
</select>
```

Browser



HTML

Forms - Option Groups

In some instances, we might want to add some additional organization to our option elements. We can do so using the <optgroup> tag. Using <optgroup> allows us to group our elements into smaller sections. The <optgroup> requires a 'label' attribute. This attribute is used as a heading for the option group but is not selectable.

HTML

```
<select name="movie">
   <optgroup label="Disney Movies">
      <option value="1">Toy Story</option>
      <option value="2">Lion King</option>
      <option value="3">The Jungle Book</option>
      <option value="4" selected="selected">Finding Nemo</option>
   </optgroup>
   <optgroup label="Scary Movies">
      <option value="5">Poltergeist</option>
      <option value="6">Psycho</option>
      <option value="7">The Thing</option>
      <option value="8">Jaws</option>
   </optgroup>
</select>
```

Best practice is to only use <optgroup> when you have a large number of options and only a few option groups.

HTML

Forms - Fieldsets

As previously stated, all children of our <form> element must be block elements. While we can use elements to achieve this, this is a semantically generic element that doesn't tell us much about what it is being used for.

Instead, we can use a <fieldset> element to serve as the block container for all of our form elements. Fieldsets are useful for grouping related fields. You can have multiple fieldsets in a single form and they can also be nested, allowing for subgrouping.

HTML

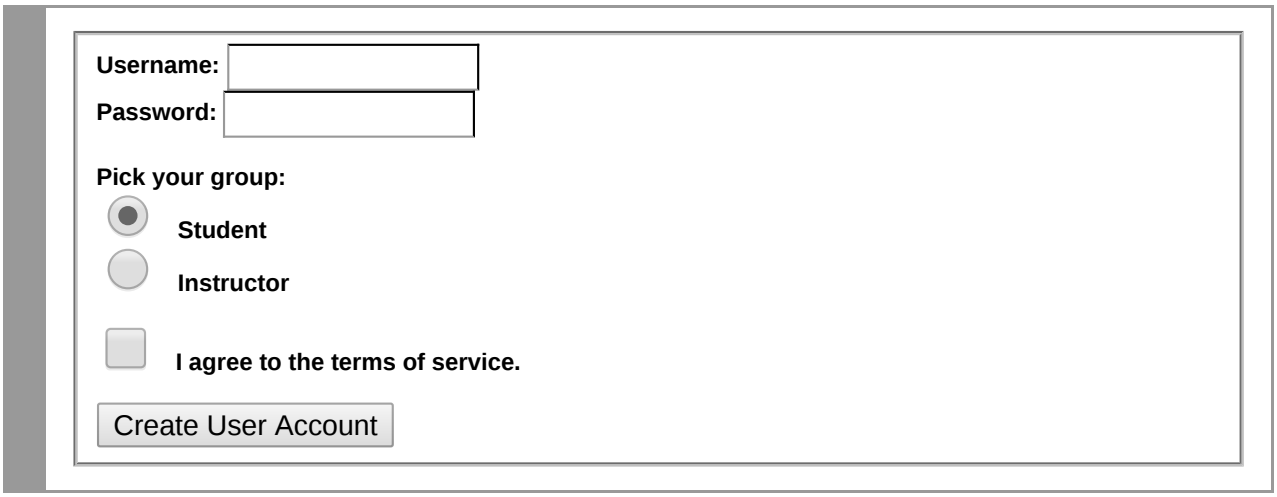
```
<form action="somePage.html" method="post" >
<fieldset>
Username: <input name="username" value="" type="text" /><br />
Password: <input name="password" value="" type="password" /><br /><br />

Pick your group:<br />
<input name="group" value="1" type="radio" checked="checked" /> Student<br />
<input name="group" value="2" type="radio" /> Instructor<br /><br />

<input name="terms" value="1" type="checkbox" />
I agree to the terms of service.<br /><br />

<input name="submit" type="submit" value="Create User Account" />
</fieldset>
</form>
```

HTML

Forms - Fieldsets

When displayed on the screen, fieldsets have a solid border by default. However, this can be changed with CSS.

Browser



HTML

Forms - Fieldsets

Fieldsets are particularly important when it comes to radio buttons. By grouping radio buttons in a single fieldset, we use HTML to tell the screen readers that the options listed are all related.

HTML

```
<form action="somePage.html" method="post" >
<fieldset>
Username: <input name="username" value="" type="text" /><br />
Password: <input name="password" value="" type="password" /><br /><br />

   <fieldset>
   Pick your group:<br />
   <input name="group" value="1" type="radio" checked="checked" /> Student<br />
   <input name="group" value="2" type="radio" /> Instructor<br /><br />
   </fieldset>

<input name="terms" value="1" type="checkbox" />
I agree to the terms of service.<br /><br />

<input name="submit" type="submit" value="Create User Account" />
</fieldset>
</form>
```

HTML

Forms - Legend

The <legend> element allows us to provide a caption for our fieldset. When included, the <legend> must be the first child of a <fieldset> element and can only be specified once per fieldset.

HTML

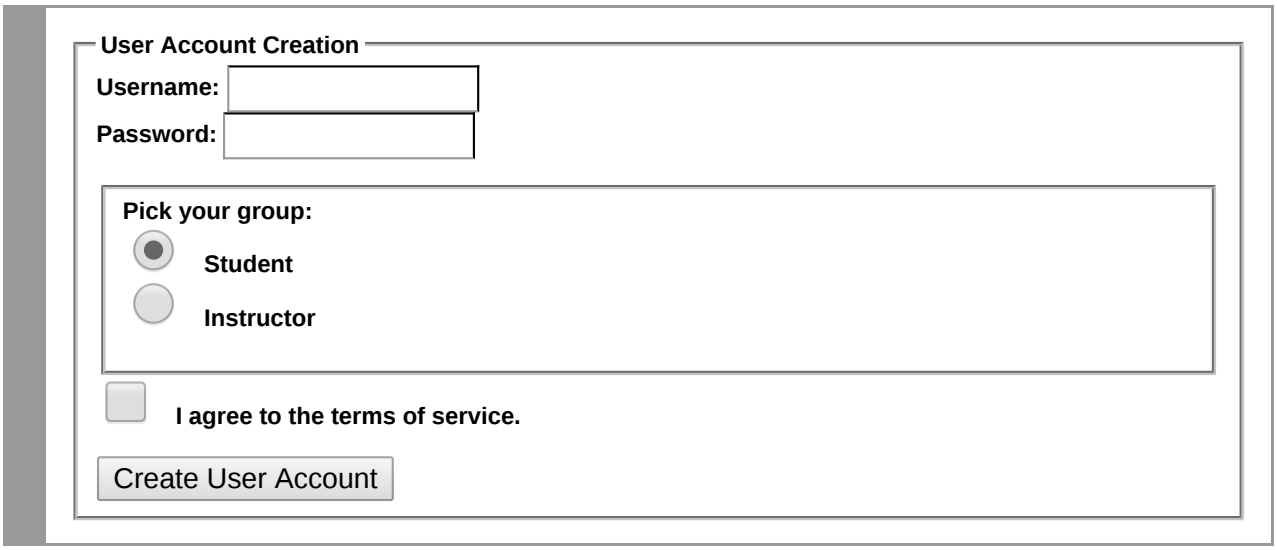
```
<form action="somePage.html" method="post" >
<fieldset>
<legend>User Account Creation</legend>
Username: <input name="username" value="" type="text" /><br />
Password: <input name="password" value="" type="password" /><br /><br />

   <fieldset>
   Pick your group:<br />
   <input name="group" value="1" type="radio" checked="checked" /> Student<br />
   <input name="group" value="2" type="radio" /> Instructor<br /><br />
   </fieldset>

<input name="terms" value="1" type="checkbox" />
I agree to the terms of service.<br /><br />

<input name="submit" type="submit" value="Create User Account" />
</fieldset>
</form>
```

HTML

Forms - Legend

When displayed on the screen, <legend> elements sit on top of the border of the parent <fieldset>.

Browser



HTML

Forms - Label

The <label> element allows us to associate the text label of a form element with that element. <label> elements should always have a 'for' attribute, the value of which should always match the corresponding 'id' attribute from the form element.

In the example below, we use the value 'subscribe' for our 'for' attribute, matching the 'id' attribute of the associated checkbox.

HTML

```
<input name="subscribe" id="subscribe" value="1" type="checkbox" />
<label for="subscribe">Subscribe to newsletter</label>
```

Browser

Subscribe to newsletter

One of the benefits of using labels is that users can click on the label for an element and the associated field will be focused on. For checkboxes and radio buttons, this means that the user can click on the text of the label to toggle the checkbox (or select the radio button).

For text and password input elements, clicking on the label will move the focus to the input field, allowing the user to start typing.

HTML

Forms - Uploading Files

In order to allow users to upload files, we need to use 'file' for the type attribute of our input element. Additionally, we need to adjust the 'enctype' attribute of our form element, setting it to 'multipart/form-data'. This attribute tells the browser that we're sending the form data as well as an attached file.

HTML

```
<form action="somePage.html" method="post"
enctype="multipart/form-data" >
   ...
      Logo: <input name="logoFile" type="file" /><br />
   ...
</form>
```

Browser

Most of the time, you won't need to specify the 'enctype' attribute for your form elements.

CSS

Styling Form Elements

Because forms are composed from a number of diverse elements, it is worth discussing how each of these elements can be styled.

The <form> element is a block element and can be styled the same way you would any element.

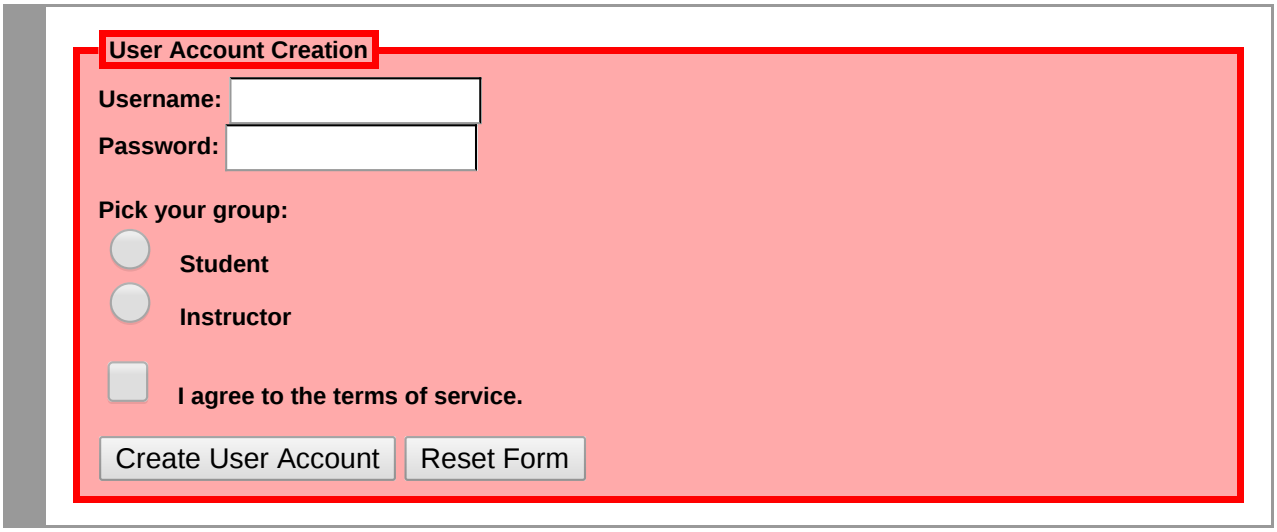
<fieldset> elements are also block elements and can be styled fairly easily.

<legend> elements are also block elements but have some quirks. In particulary, legends are displayed so that they sit on top of the border of their parent fieldset, as seen in the example on the next page.

CSS

Styling Form Elements

Browser



CSS

Styling Fieldsets & Legends

If we wanted the legend to display above the fieldset border (rather than on top of it), we might try something like the following:

CSS

```
legend
{
   margin-top: -.5em;
}
```

However, this would have no effect. In most browsers, it would seem that legend elements are impervious to any margin adjustments.

CSS

Styling Fieldsets & Legends

While you can't adjust the margins for your legend elements, doing something like the following will have a similar effect.

CSS

```
legend
{
   position: relative;
   top: -.7em;
}
```

While relative positioning is a valid approach, it is important to remember that changing the position value for an element can have unintended consequences. Be sure to test your solution thoroughly if taking this approach!

CSS

Styling Input Elements

Because the <input> element is so versatile, styling it can be challenging.

- Text inputs, password inputs, form buttons (submit/reset), and text areas can all be styled easily
- Select inputs can be styled, options can also be styled but typically are not
- Some elements of file inputs can be styled
- Styling radio buttons and checkboxes can have mixed (or no) results and is generally not recommended

CSS

Styling Input Elements

Below, we've specified that all of the input elements on our page should have a solid border and background color (see output on the next page).

CSS

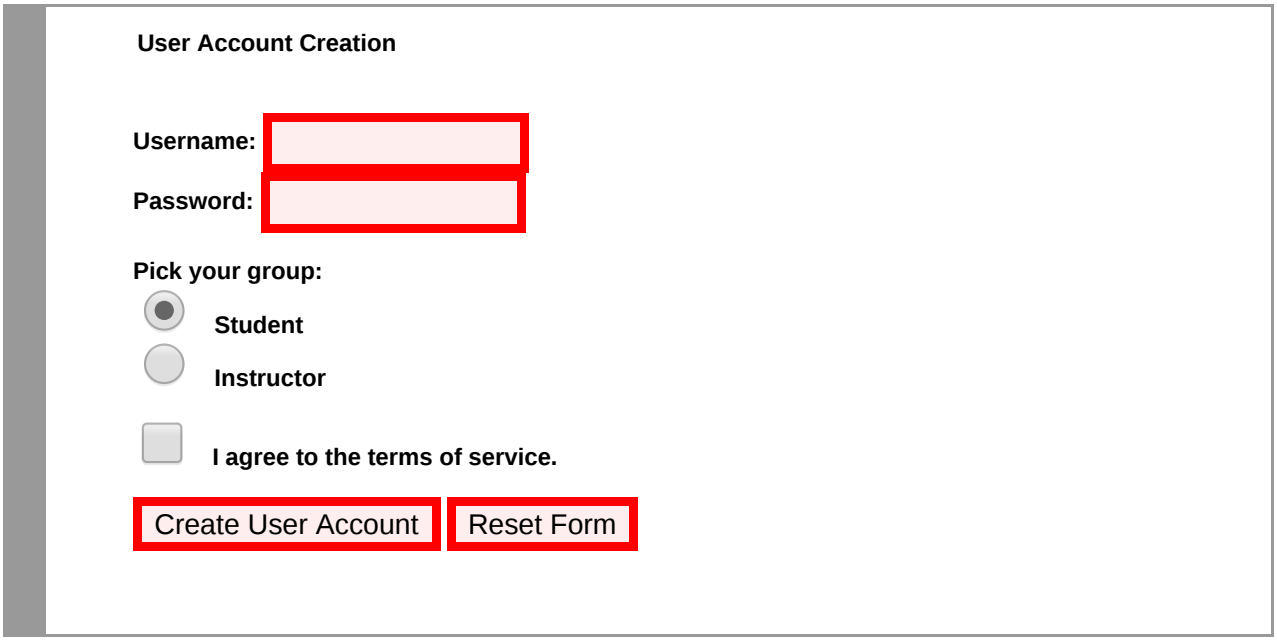
```
input
{
   border:3px solid #f00;
   background-color:#faa;

}
```

CSS

Styling Input Elements

Our text, password, submit, and reset buttons all display as expected, with a red border and a pink background color. However, our checkboxes and radio buttons don't have either of these styles applied to them.

Browser



CSS

Styling Input Elements

If we were to adjust the width of our input elements, we'd find that it has similarly mixed results (see next page).

CSS

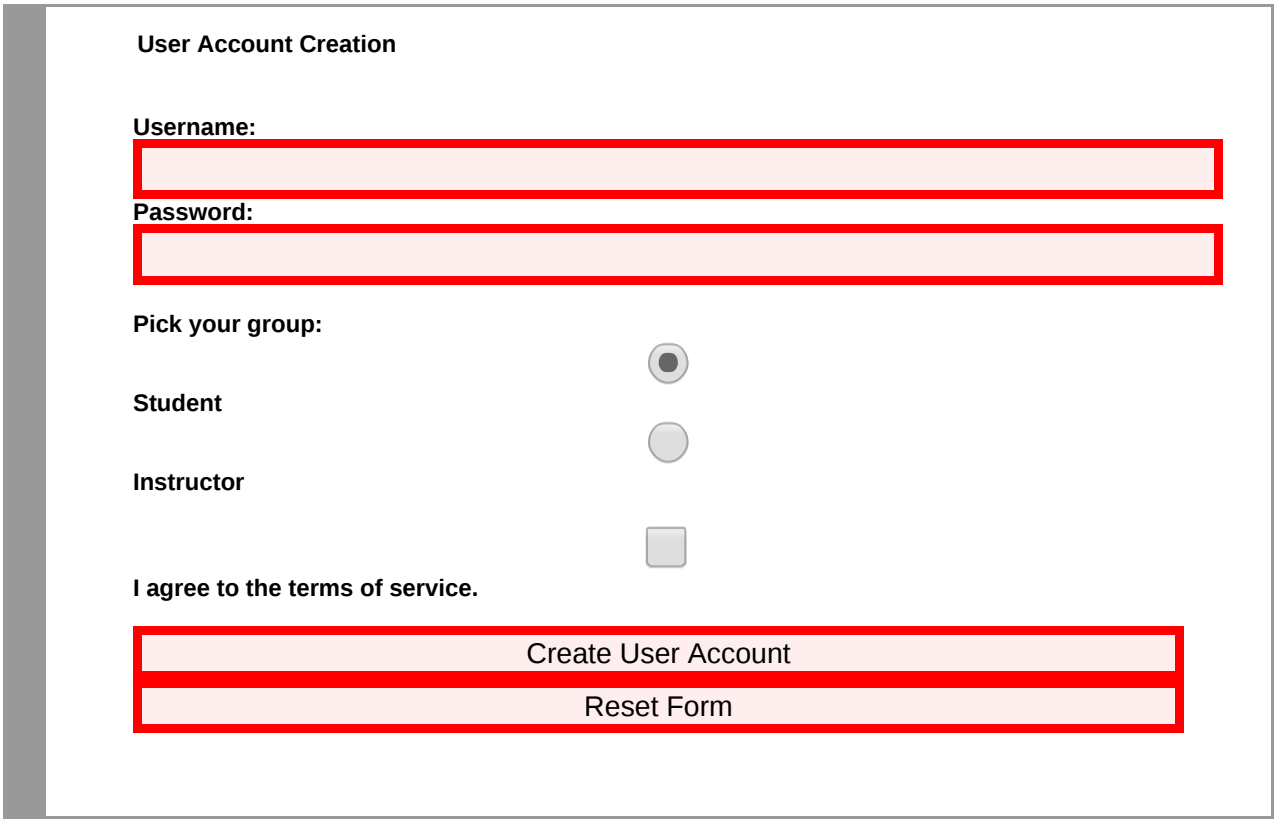
```
input
{
   border:3px solid #f00;
   background-color:#faa;
   width:100%;
}
```

CSS

Styling Input Elements

Each of our input elements does in fact take up 100% of the width of the containing element. However, the actual checkbox and radio buttons display looks the same, only centered. While the display doesn't look any different, the 'clickable' area of these input elements does in fact take up the full 100% width.

Browser



CSS

Styling Input Elements

Because the various input elements respond to our CSS rules differently, it is recommended that you make your CSS selectors more specific, targeting a specific type of input element rather than simply styling the <input> element.

- Helpful to add a class for each type of form element
    - E.g., <input type="text" class="textBox"... />
- Alternatively, you can use CSS attribute selectors...

CSS

CSS Attribute Selectors

E[attr] - Matches all elements that have the specified attribute

CSS

```
input { width: 100% }
input[checked] { width:20px; }
```

CSS

CSS Attribute Selectors

E[attr="val"] - Targets all elements with an attribute value **exactly** equal to 'val'

CSS

```
input[type="text"]
{
   color: red;
}
```

CSS

Styling Other Form Elements

<select> and <textarea> elements can both by styled as though they were block elements. Depending on the browser, the styles applied to your <select> element may or may not cascade to your <option> elements.

CSS

```
select,
textarea
{
   border:3px solid #f00;
   background-color:#faa;
}
```

While all form elements can be styled to a certain extent, doing so may have a negative effect on your user experience. Most users are familiar with the default styling of form fields. Changing these styles may lead to confusing for some.

CSS

Painless Forms

Forms are vital to online communication, gathing data from websites, and e-commerce. However, forms tend to be painful for most users. Poorly designed forms are, at best, tedious to fill out. In the worst case scenario, a poorly designed form can actually cause potential customers to go to another website to find what they're looking for.

How can we make the process less painful?

CSS

Painless Forms

To provide our users with well-designed forms, we want to consider three goals when thinking about our form layout:

- Facilitate scanning
- Reduce visual noise
- Provide a clear path to completion

CSS

Painless - Label Alignment

As we've discussed before, users tend to scan a web page before they actually take time to read or interact with it. This holds true for web forms as well.

Users will scan the labels on our forms to identify what information is being requested and to make a decision about whether or not they want to fill out our form. Our goal is to design our form so that our labels are easily scanned while flexible enough for our needs.

There are four approaches to label placement:

<labels>
- Left-aligned
- Right-aligned
- Top-aligned
- Inline Faux-labels

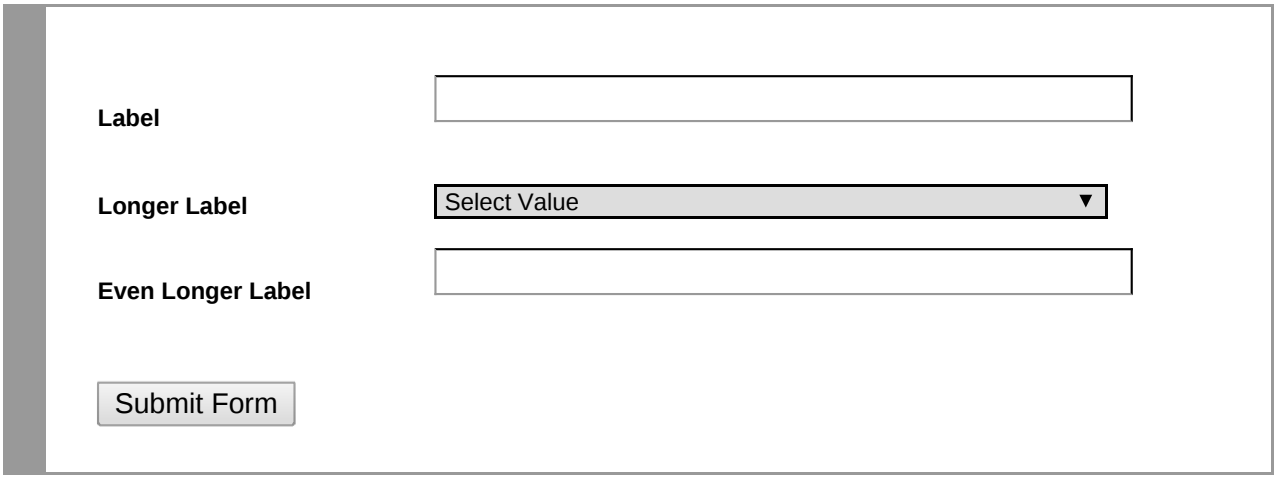
We'll explore each of these on the following pages.

CSS

Painless Forms - Left-aligned Labels

Left-aligned labels provide a flush left edge, allowing users to easily scan the labels. However, this approach also leaves a relatively large amount of space between our labels and the corresponding form elements, making it harder for the user to associate one with the other. Laying out the labels horizontally next to the form field also gives us a limited amount of space for our labels before they require a second line of text. Ideally, we'd like to keep our labels to one line so that they line up with the form fields.

Browser



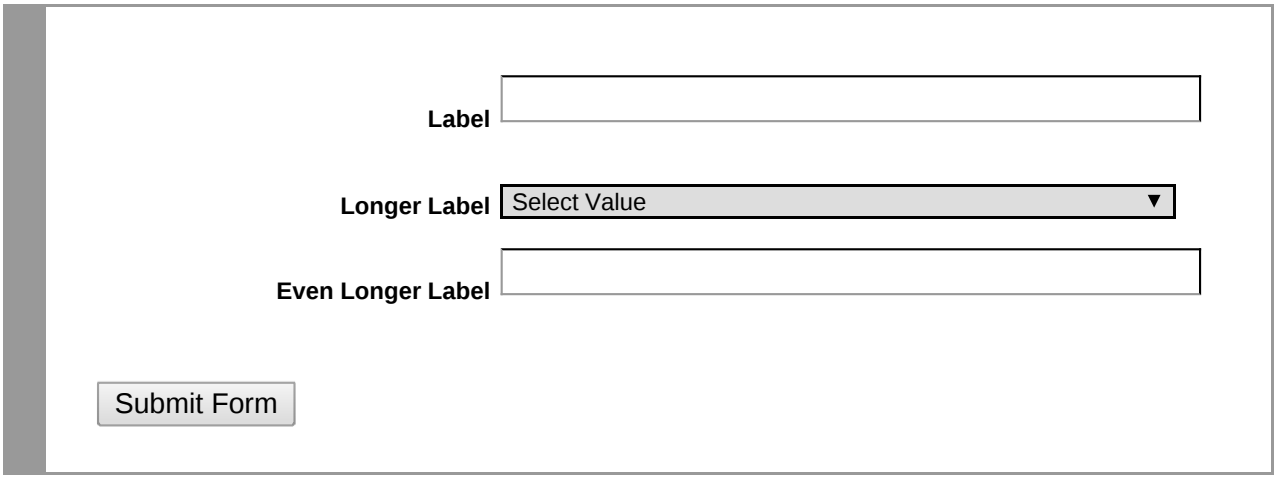
Studies have shown that when viewing a form with left-aligned labels, users require more time to scan and complete the form (as compared to right-aligned or top-aligned labels).

CSS

Painless Forms - Right-aligned Labels

Right-aligned labels provide a flush right edge but leaves a jagged left edge. While this creates a clearer association between labels and the corresponding form elements, it makes it more difficult for users to read labels. This approach also shares the drawback of giving us a limited amount of space for our labels before they require a second line of text.

Browser



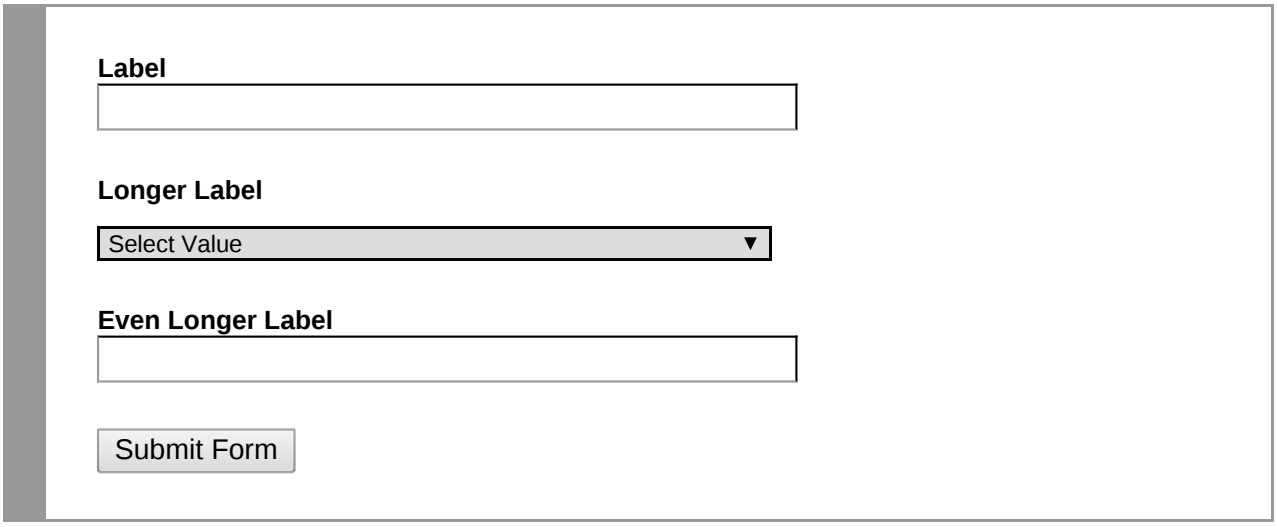
Studies have shown that when viewing a form with right-aligned labels, users require less time to scan and complete the form as compared with left-aligned, particularly if the form is asking for common information, such as name, address, etc.

CSS

Painless Forms - Top-aligned Labels

Top-aligned labels provide a clear association when combined with good spacing between form elements. Additionally, there is no jagged edge to worry about since both labels and form elements are aligned on the left side. This approach also provides us with more space for our labels to grow, even allowing for a second line of text without disrupting the flow of the form.

Browser



While this approach has a number of benefits, it can be problematic if the spacing between labels and form elements isn't managed properly. Too much spacing can create a disconnect between the two elements. Additionally, not enough space between labels and unassociated elements can create confusion for users. It's also worth noting that this approach requires more vertical space and can easily push form elements 'below the fold'.

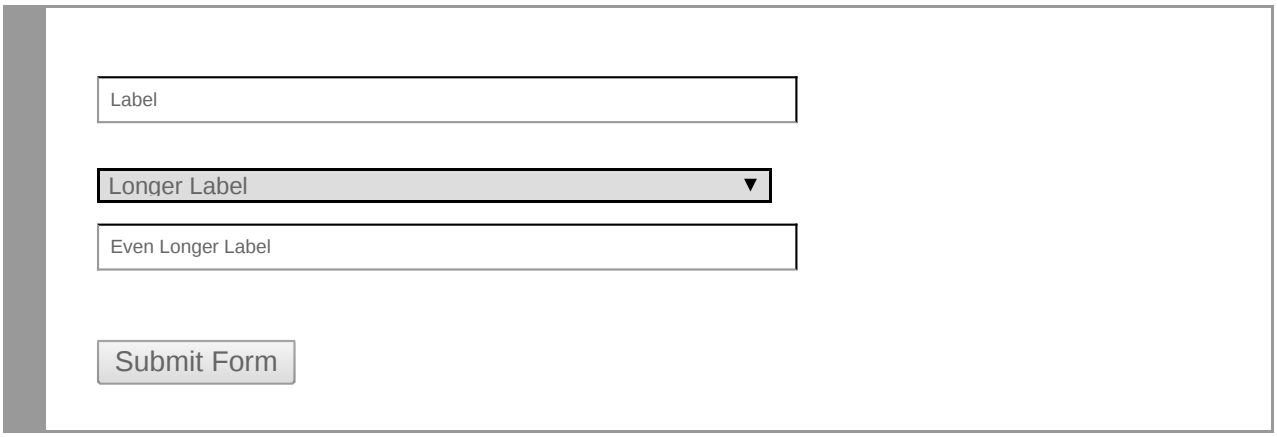
All that being said, top-aligned labels have proven themselves to be the best option for providing users with a form that can be easily scanned and quickly completed.

CSS

Painless Forms - Inline Faux-Labels

Inline faux labels are actually not labels at all. This approach uses the 'value' attribute of input elements to provide a 'label', indicating what the form field should be used for.

Browser



While this approach is useful when you're working with a limited amount of vertical and horizontal space, it has some significant issues that you must consider before using. First, this approach uses less semantic markup than using an actual <label> element, making it less accessible for screen-reaaders. Additionally, if a user enters data into the form field, the 'label' is no longer visible. Were they to delete the entered data, they would be left with a form without any labels and would have to refresh the page or reset the form.

If you use this approach, it is important to be sure and provide a reset button to allow users to easily reset the form and re-display the label information.

CSS

Painless Forms - Facilitate scanning

You are advised to use the following approaches for your form labels, in order from most preferred to least preferred:

- Top-aligned for reduced completion times and familiar data input
- Right-aligned when vertical space is constrained
- Left-aligned for unfamiliar or complex data entry
- Inline faux-labels when horizontal and vertical space is constrained (rare)

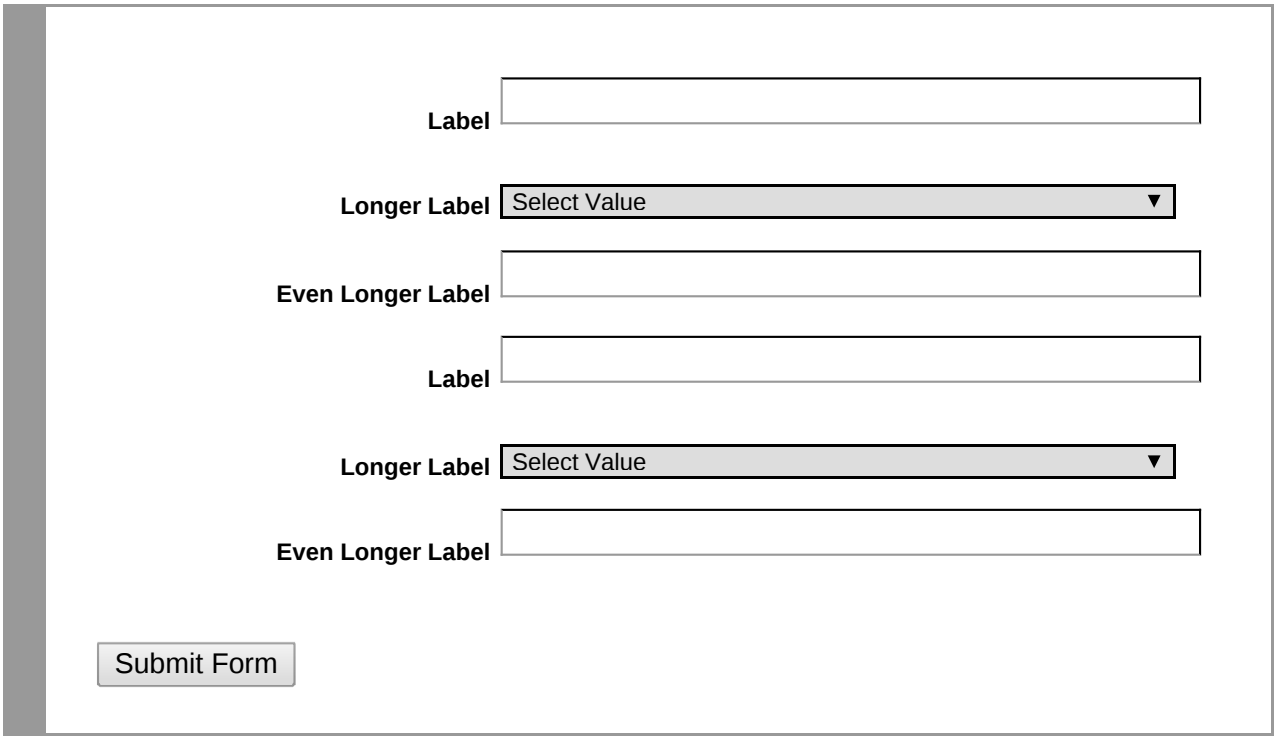
CSS

Painless Forms - Content Grouping

Another factor in making it easier for users to scan our forms is the grouping of similar or related form elements.

The form below has a number of elements. For users, looking at a long form like this can be a little intimidating. Additionally, as the eye scans down the page, the repeating elements can cause the users eyes to tire and the form elements to begin to blur together.

Browser

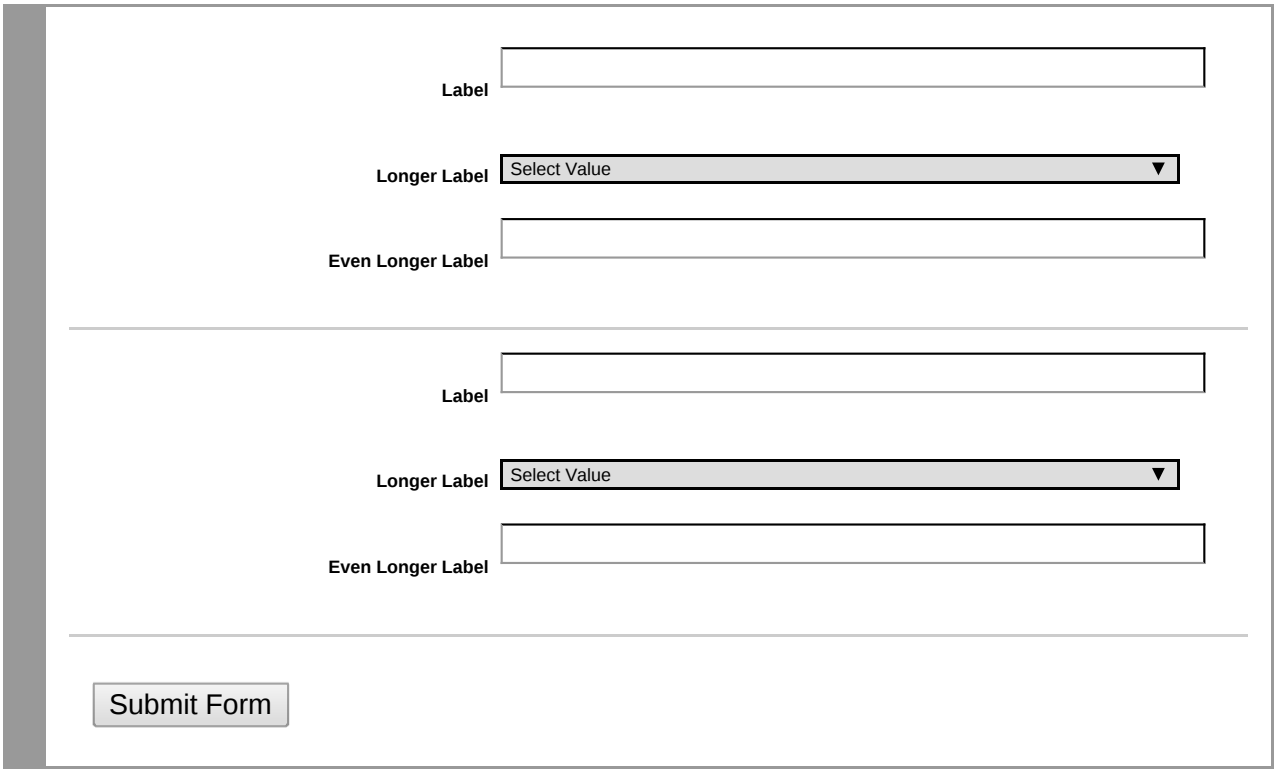


CSS

Painless Forms - Content Grouping

To help the user scan more easily, we can provide small breaks in the form, logically placed between form elements so that related or similar elements are grouped together.

Browser



These small breaks allow the users eye to rest, making the scanning process easier and also providing some additional structure for your form elements.

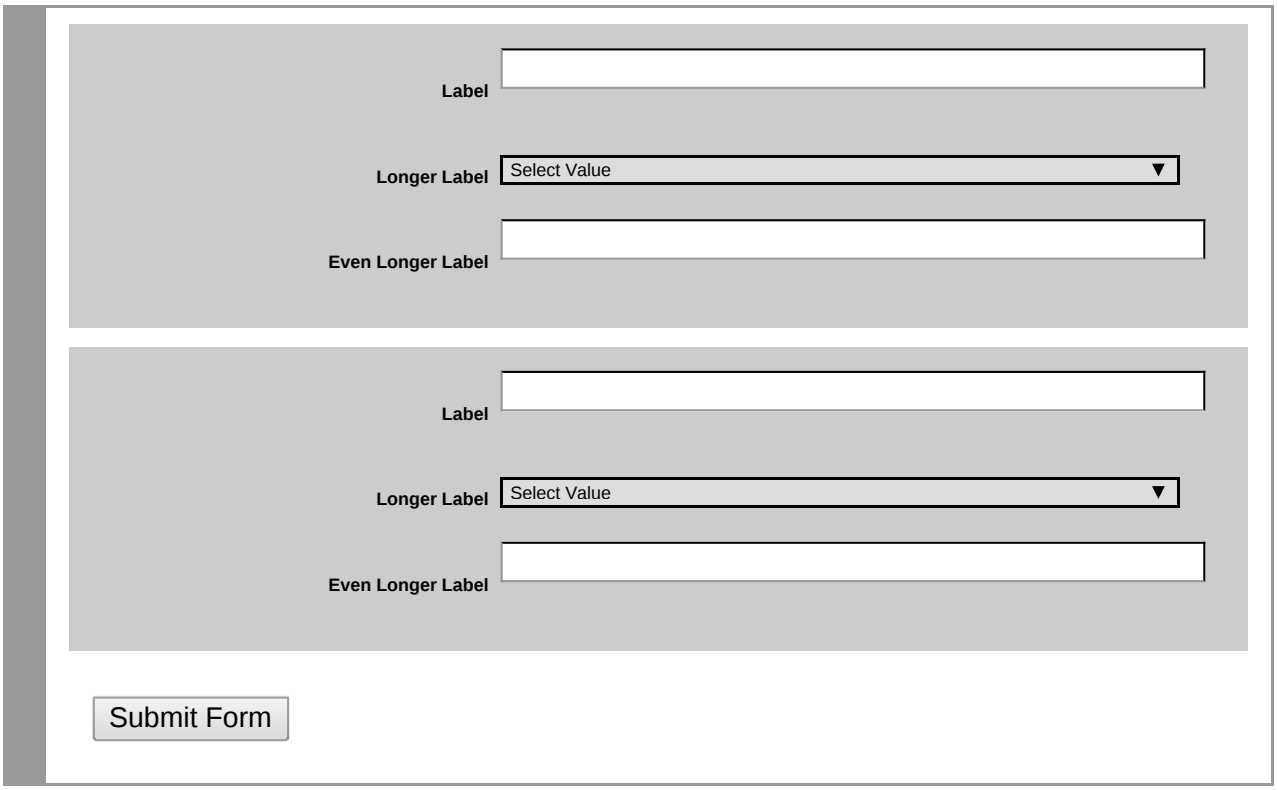
Visually, we've used a small horizontal line between form elements to create groupings. Structurally, this can easily be done using a combination of <fieldset> elements and the 'border' property in CSS.

CSS

Painless Forms - Content Grouping

Instead of a small border between our form elements, consider the following example using background color to group our content.

Browser



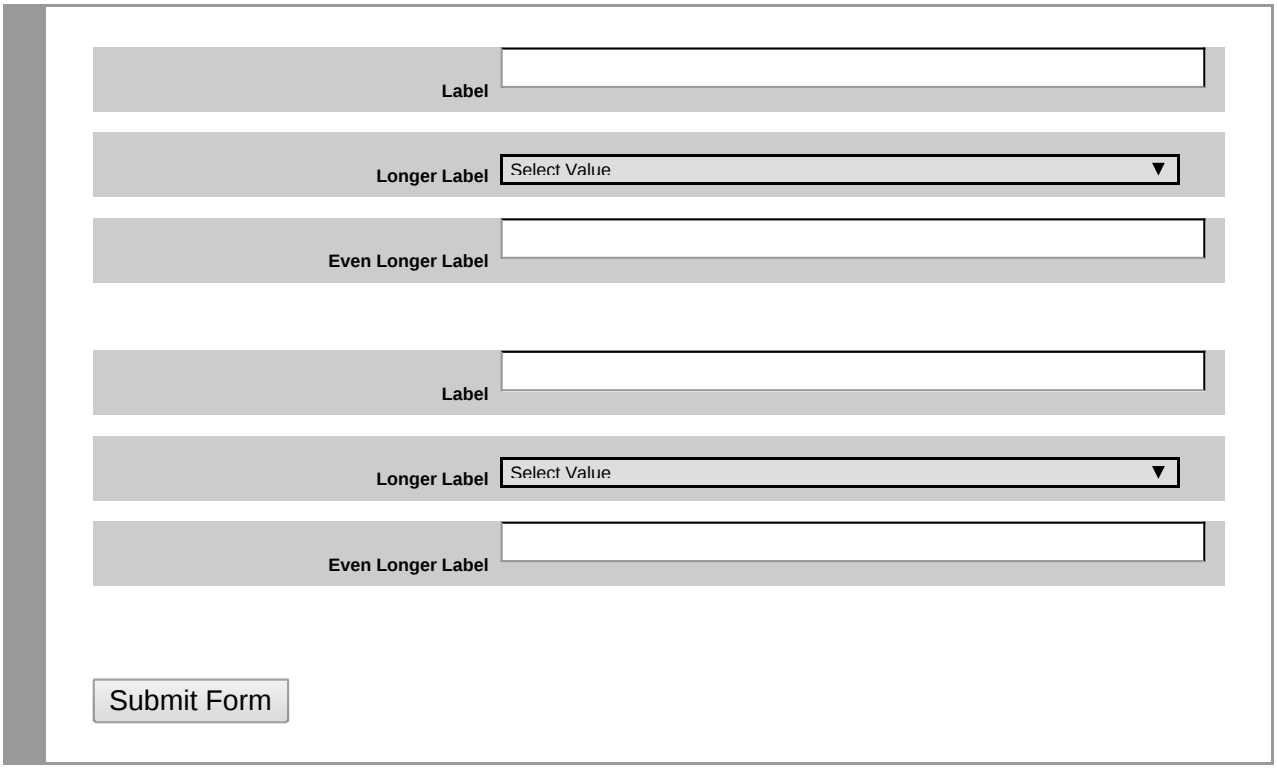
While the use of background color does provide a means for grouping content, it adds visual weight to the page, often leading to a more cluttered look and feel.

CSS

Painless Forms - Content Grouping

Additionally, the combination of poor spacing with the use of a background color can lead to a very unpleasant user experience (as seen below).

Browser



Because they can easily have the opposite effect, the use of borders and background colors to facilitate scanning should be limited and well thought out!

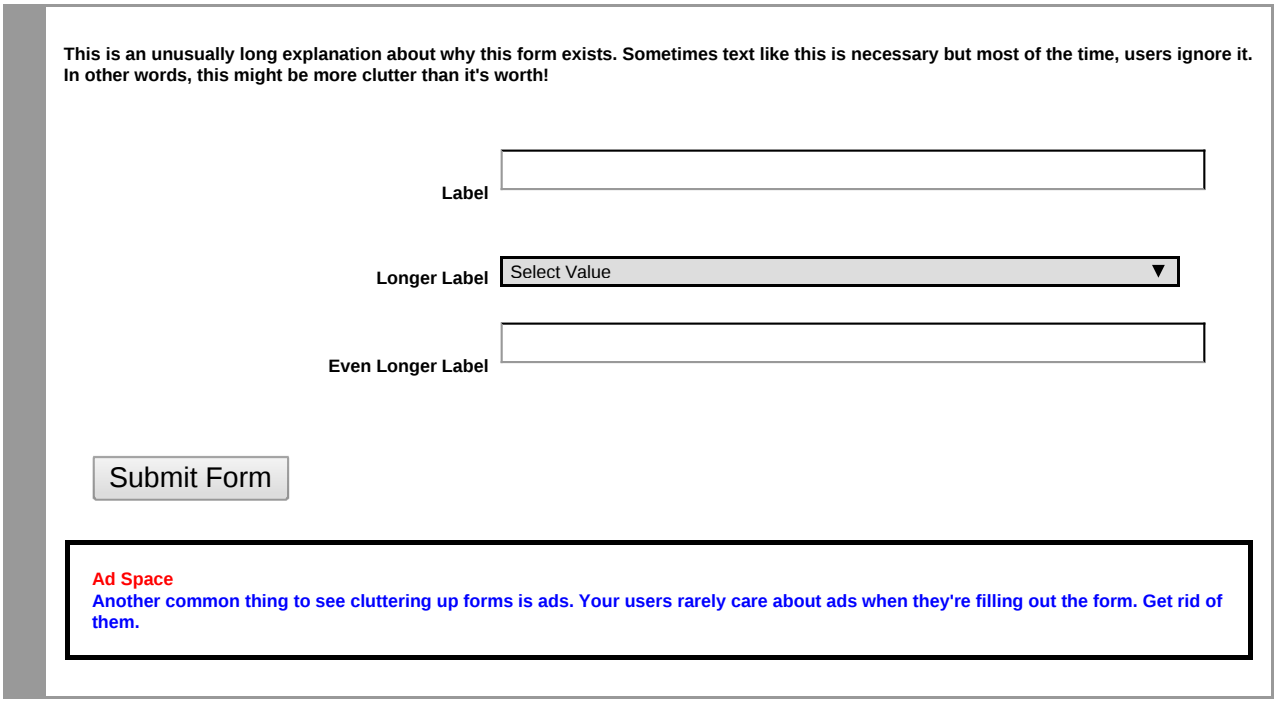
CSS

Painless Forms - Unnecessary Elements

The second goal for creating painless forms is to reduce visual clutter, things that would draw the user's eye away from the form elements. The first, and easiest step, towards this goal is to remove any unnecessary elements.

Consider the sample page below:

Browser



The form is accompanied with a set of instructions at the top of the page (which most users are unlikely to read unless it is a complicated and/or unfamiliar form). There is also space for advertising underneath the form. It could be argued that both of these elements aren't necessary on this page. If an element isn't necessary on a form page, the recommendation would be to remove it.

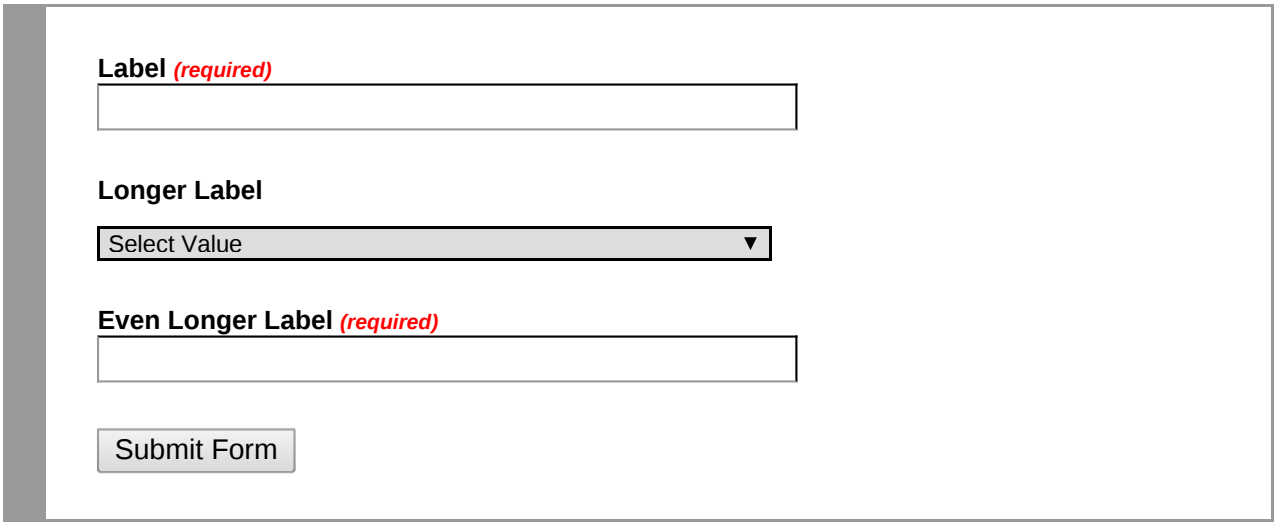
CSS

Painless Forms - Unnecessary Elements

A common practice with forms is to add indicators on required form fields so that users know they have to fill it out before proceeding.

While we could actually spell out the word 'required', it could be argued that this adds visual clutter to the form.

Browser

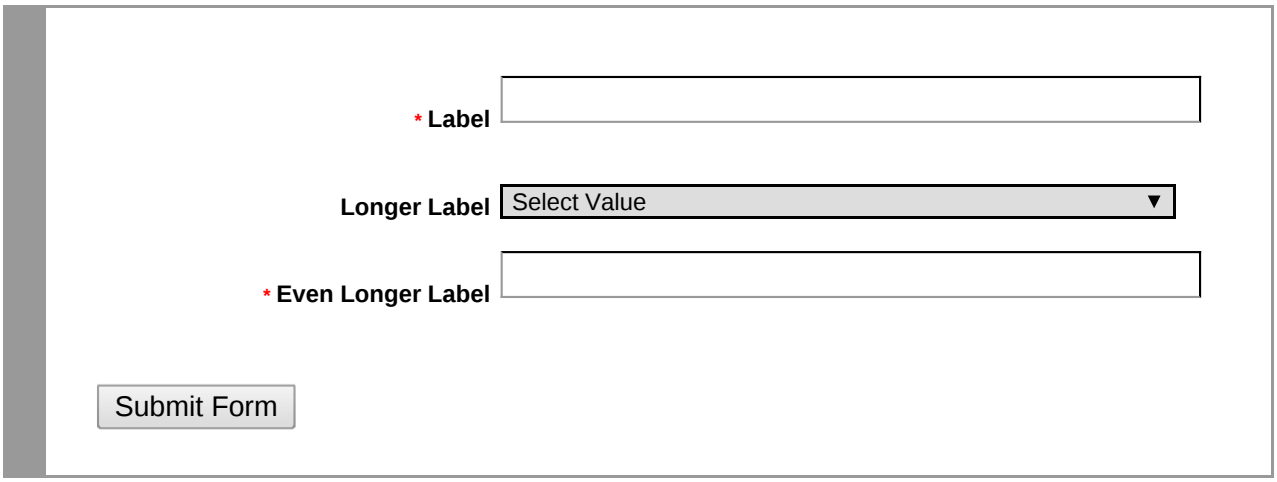


CSS

Painless Forms - Unnecessary Elements

Instead of using the word 'required', we could use an asterisk instead. The form could have a footnote noting that the asterisk indicates a required field. That said, most users know that the asterisk is a common marker for required fields, as it has become an accepted convention for a number of years now.

Browser



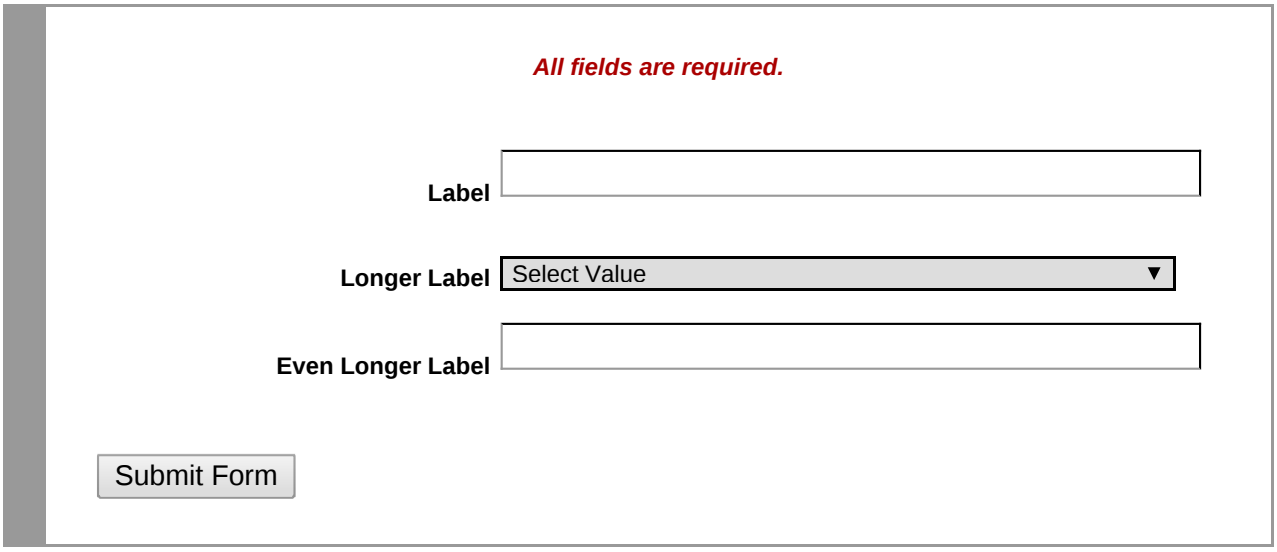
It's worth noting that for right-aligned labels, we would want to place the asterisk before the text of the label, so as to not create a jagged right edge. For left-aligned, top-aligned, and inline faux-labels, you would want to place the asterisk after the text of the label.

CSS

Painless Forms - Unnecessary Elements

If all fields on a form are required, it could be argued that adding an asterisk to each label is unnecessary. Simply leaving all labels as they are would suffice. You could also add a short note at the top of the page stating that all fields are required.

Browser

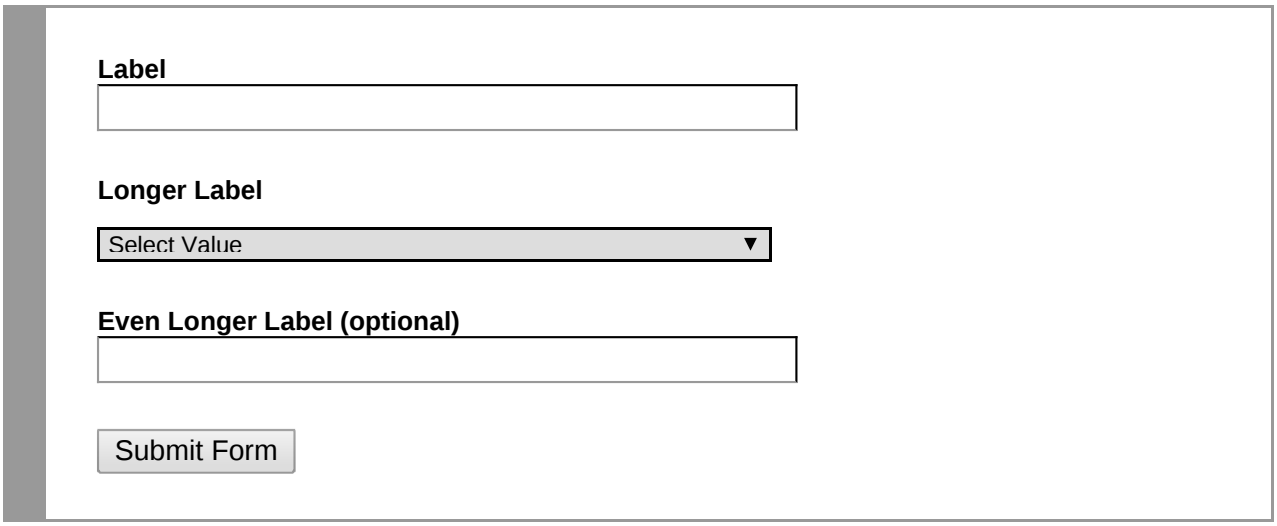


CSS

Painless Forms - Unnecessary Elements

If most (but not all) fields on a form are required, another common approach to reducing clutter is to add the word 'optional' to the labels of optional form fields.

Browser



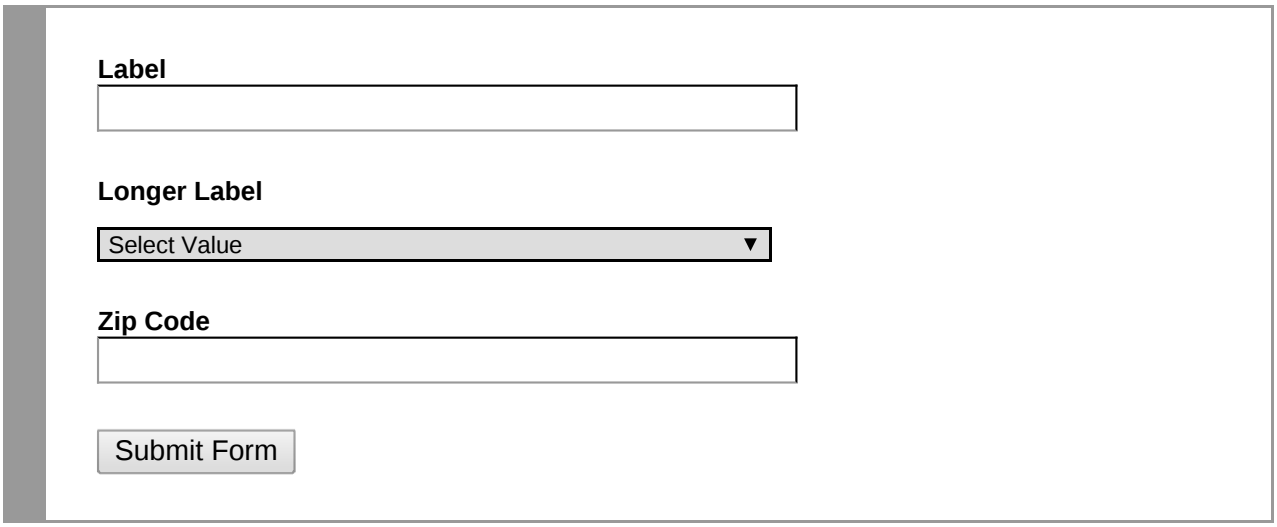
While this is a valid approach, you might also consider removing the optional field, as most users won't fill it out.

CSS

Painless Forms - Field Length

Another way we can reduce visual clutter is to reduce the size of some form fields. For example, consider the form field for 'Zip Code', displayed below:

Browser



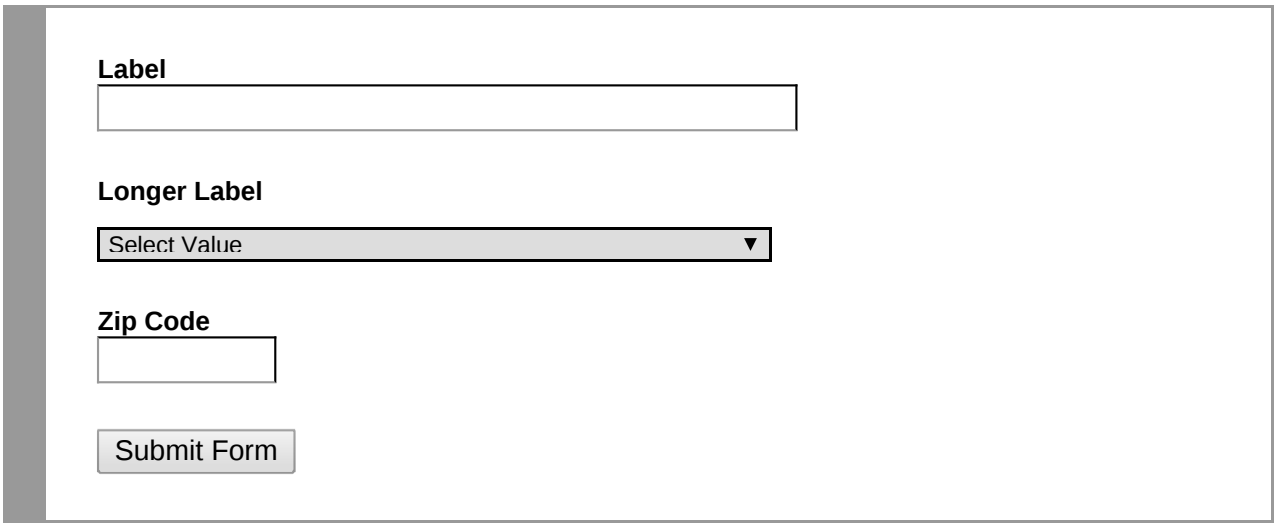
The field stretches the entire width of the screen, even though we know US zip codes are typically five digits (Note: Other countries do not have the same five-digit limitation on postal codes!)

CSS

Painless Forms - Field Length

Instead of having the zip code field with a width of 100%, we can specify that it should be '5em'. This would ensure that it is large enough for users to see all of the data they've typed in while reducing the amount of space required by the field.

Browser

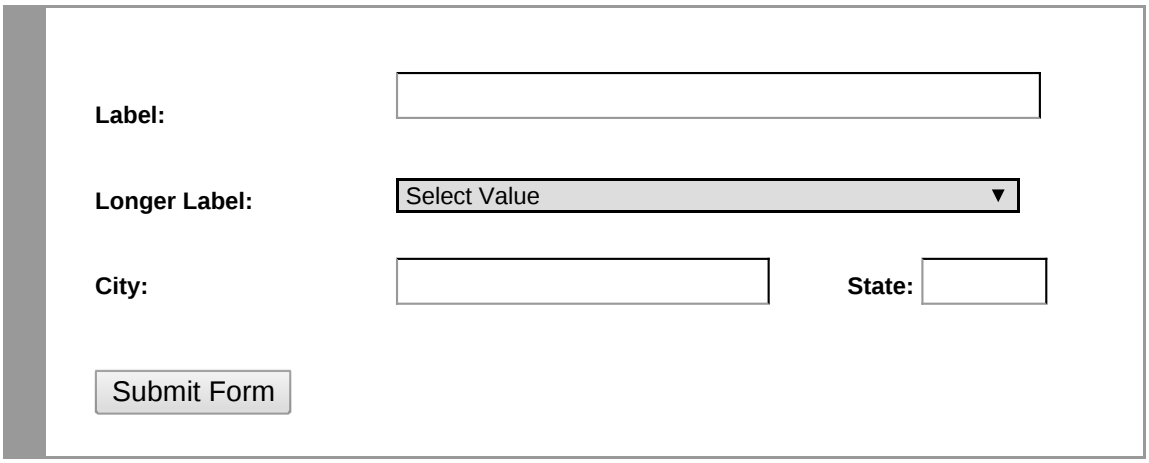


CSS

Painless Forms - Field Length

Similarly, consider the form displayed below. Instead of using a full row for city and state, we can move these both to a single line by reducing the size of the city and state fields. This also gives the user a visual indication that we only expect them to put in the state abbreviation, rather than the full name of their state.

Browser



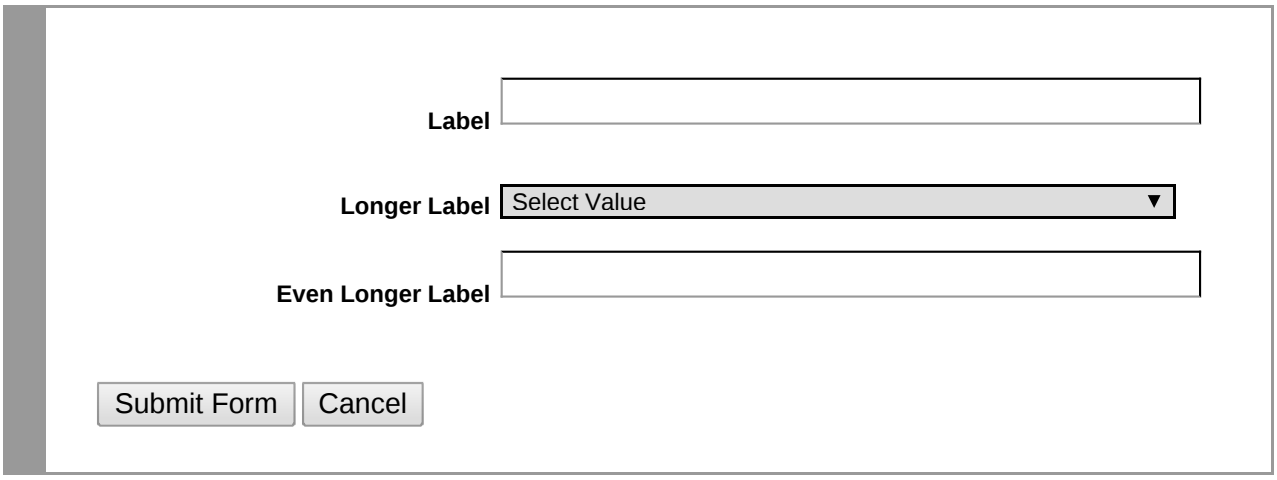
CSS

Painless Forms - Primary Actions vs. Secondary Actions

The last step in reducing visual clutter is to clearly distinguish between primary actions and secondary actions. Primary actions are those that are responsible for form completion (e.g., submit buttons). Secondary actions are those that are rarely (or never) used (e.g., reset buttons).

Consider the form displayed below:

Browser



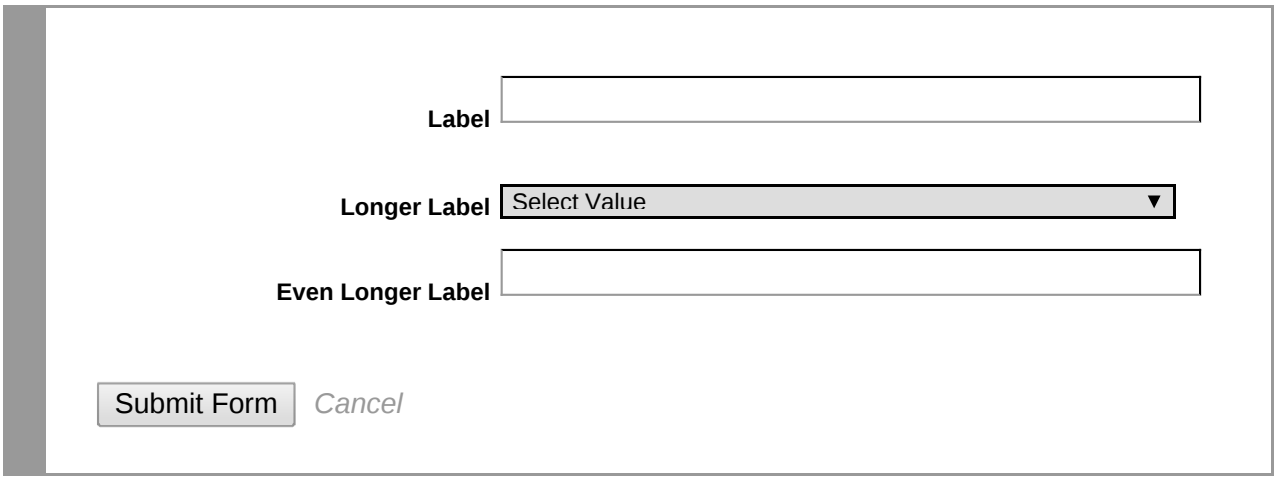
Both the submit button and the cancel button look exactly the same. With this being the case, it is not too hard to imagine a scenario where a user accidentally clicks the wrong button.

CSS

Painless Forms - Primary Actions vs. Secondary Actions

By using our understanding of color theory and design principles, we can reduce the visual weight of our secondary action by removing the background color & borders and making the text color a light gray.

Browser

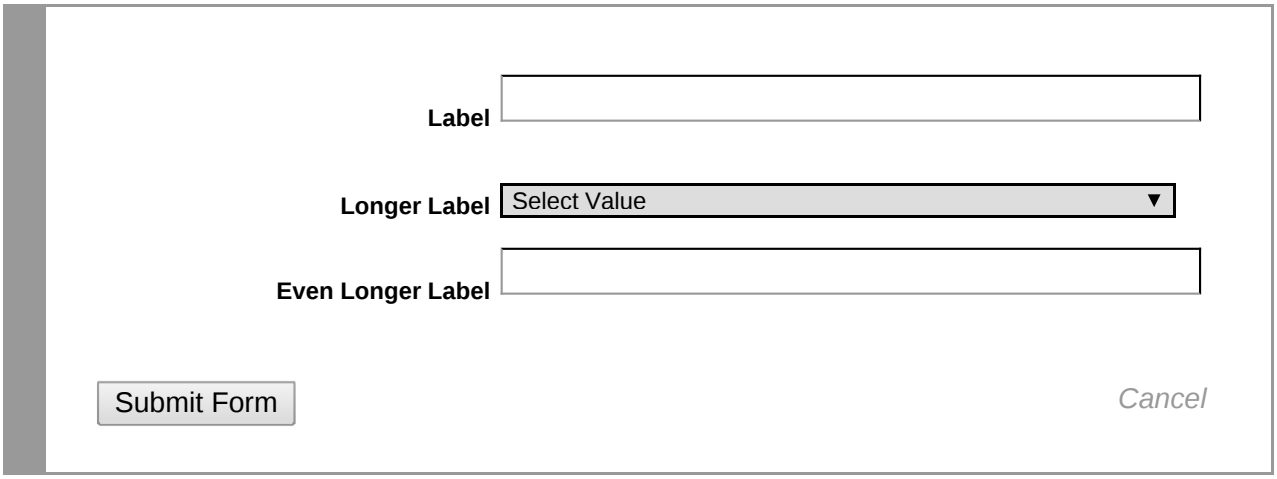


CSS

Painless Forms - Primary Actions vs. Secondary Actions

Even more visual distinction can be made by adding white space between the two buttons.

Browser



Depending on the use of the secondary action, a better option might be to remove it all together.

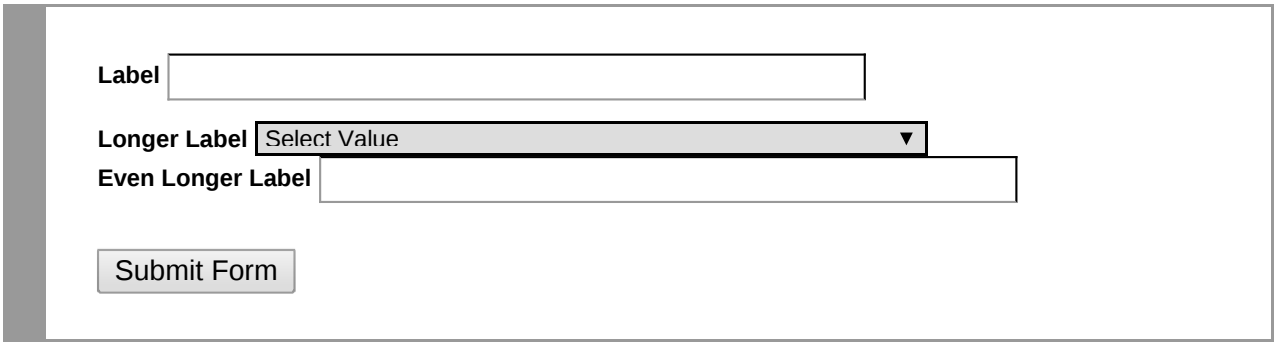
CSS

Painless Forms - Completion Path

The last step in creating a painless web form is to provide the user with a clear path to completion. We want our users to be able to follow an imaginary vertical line from the start of the first form field, past each of the following form fields, all the way to the submit button.

The following form does just the opposite, requiring the user's eye to bounce around across the page from one field to the next.

Browser

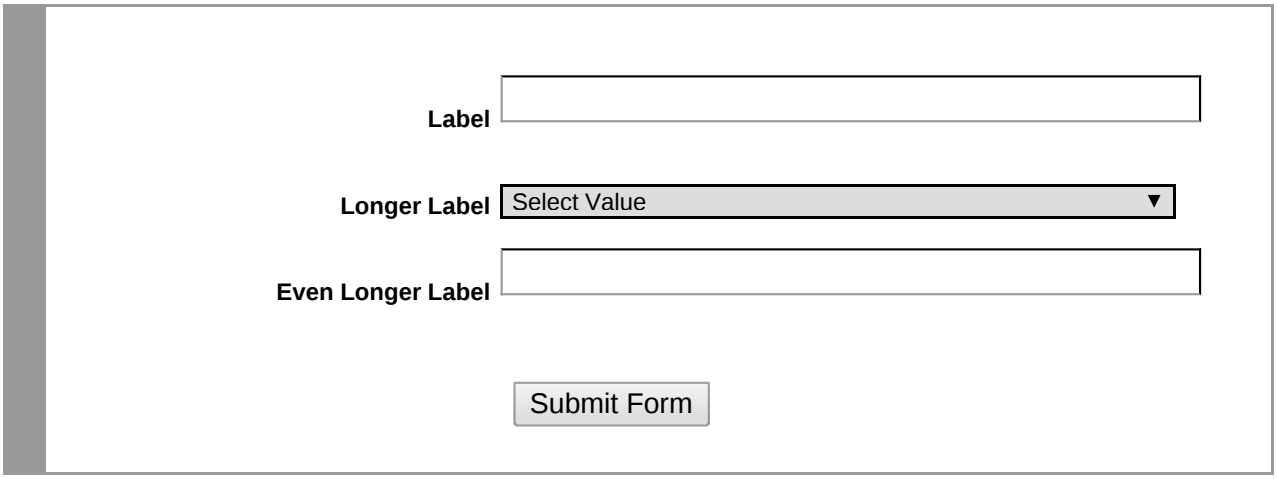


CSS

Painless Forms - Completion Path

A better approach would be to provide consistent placement of all of the form fields, **including the submit button**.

Browser



While this is more challenging when using left-aligned or right-aligned labels, this is a fairly trivial task when using a top-aligned approach.