

HTML (Hypertext Markup Language) is the most basic of all things that you must learn to make something visual for the World Wide Web. HTML is a very easy language to learn and super simple to write. You start off with

```
<html>
  <head>
    <title>An HTML Document with No Body</title>
  </head>
  <body>
  </body>
</html>
```

As you can see, the HTML language uses words contained inside angle brackets: `<` and `>`, known as tags or elements. You also might notice that every tag is almost duplicated, but the second tag has a `/` before it. **Open tags** are the ones without the slashes, like `<html>` and `<body>`. **Close tags** are the ones with the slashes, like `</html>` and `</body>`. Everything between an open tag and its corresponding close tag is the **tag content**.

In the above HTML snippet, the `html` tag wraps the entire document. All other content should be within the `html` open and close tags.

The `head` tag contains metadata about the document. In this case, it contains the `title` tag which specifies the title of the HTML document which, in this case, is "An HTML Document with No Body".

*Remember, HTML was invented so scientists could share their results. Phrases like "HTML document" hearken back to that time when the "document" was a research paper and HTML was the format it was written in.*

Between the open and close `body` tag should be the place where you put all of the visible content of your HTML document.

The phrase **HTML element** is sometimes used interchangeably with the concept of the everything from the open tag, all the content, all the way to the close tag.

Some elements have *no* close tag. These are called "empty" in that they have no content.

## Attributes

---

All HTML elements can have "attributes". These add more information to the HTML element. The three attributes in the following list can be applied to any HTML element.

- **id** – gives each element a unique identifier to be selected with CSS
- **class** – assigns an element to a class to be selected with CSS or JavaScript
- **title** – gives a cool hover feature that displays text over an element

Here's an example of those attributes on a paragraph tag.

```
<p id="paragraph-1" class="fancy-paragraph" title="I am a title">
  Hover over me to see the title
</p>
```

Each value for the "id" attribute must be unique across the *entire* HTML document. That gives each element its own unique identifier that you can use as a CSS selector to apply styling to one specific element.

The "class" attribute is another way for CSS to be able to select HTML elements. Classes are meant to be applied to more than one element. That means that there's nothing wrong with the following HTML even though the paragraph tags have the same class value. (Note that the ids are different, though.)

```
<p id="paragraph-1" class="fancy-paragraph" title="I am a title">
  Hover over me to see the title
</p>
<p id="paragraph-2" class="fancy-paragraph">
  I have not title. I am sad.
</p>
```

If an HTML element needs to have more than one class, you write a *space-delimited list* in the class attribute's value like this.

```
<p id="paragraph-1" class="fancy-paragraph error-message">
  Hover over me to see the title
</p>
```

This way the paragraph would have the CSS rules applied to it for all paragraph tags, any rules for the class "fancy-paragraph", any rules for the class "error-message", and any rules for the HTML element with the id of "paragraph-1".

## Hyperlink

---

In itself, an HTML hyperlink is one of the most useful distinctions between the World Wide Web and a standard text file. Linking is the foundation of the World Wide Web. You can employ links to provide navigation to other content. Remember that hyperlinks are created in HTML with **anchor tags**, that is, the <a> tag.

### Relative and absolute links

Absolute links are generally used to send readers of your Web site to another Web site. For example, <https://appacademy.io> is an absolute link because it starts with "http://" or "<https://>".

Relative links do not start with "http://" or "<https://>". Instead, they're like relative paths with respect to the file system.

```
<a href="https://appacademy.io"> Absolute Link</a>  
<a href="subdirectory/another-page.html"> Relative Link</a>
```

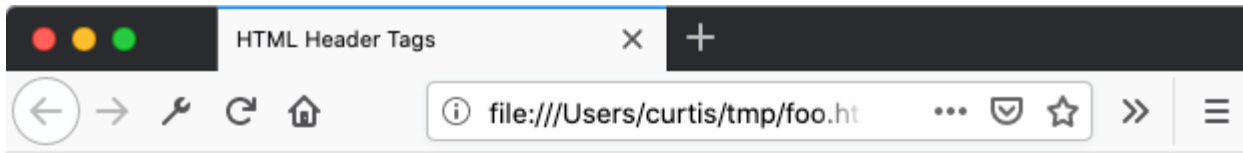
## Headings

---

Remember that HTML was originally designed to let scientists publish their results so that others could read them on their computers? Well, any good scientific paper must have headers. You have them in Microsoft Word and Google Docs. You might as well have them in HTML, too!

There are six levels of headers, each represented by a different tag: h1, h2, h3, h4, h5, and h6.

```
<html>  
  <head>  
    <title>HTML Head Tag - Tutorial and Reference</title>  
  </head>  
  <body>  
    <h1>I am very very important.</h1>  
    <h2>I am very important.</h2>  
    <h3>I am important.</h3>  
    <h4>I am kind of important.</h4>  
    <h5>I am somewhere below important.</h5>  
    <h6>I am near the importance and size of a paragraph tag.</h6>  
  </body>  
</html>
```



**I am very very important.**

**I am very important.**

**I am important.**

**I am kind of important.**

**I am somewhere below important.**

**I am near the importance and size of a paragraph tag.**

A nice rule to observe is that you should appropriately stack your headings. You should never have an `h3` directly following an `h1`. While it doesn't break the HTML, it defeats the intent of the *structure* of the document. You shouldn't want to have a hole in your hierarchy.

Visually impaired users that interact with the World Wide Web with a screen reader cannot see the tag following the "hole". Always structure your tags to have the next larger tag before it.

## Div and span tags

---

Div tags and span tags are very common HTML elements used to create additional structure for the use of styling or grouping. Because `div` tags are "block" tags, they were primarily used to create sections on a Web page before HTML 5 came along and introduced the `section` tag.

Span tags primarily used to style text within a larger body of text.

For example, in the code blocks on this page, the code block itself is a `div`. Each differently-colored element is wrapped in a `span` tag.

# Image tag

---

Unlike most elements, the image tag allows you to show an image in your Web page. The HTML image tag instructs the browser to retrieve the image and displays it in your Web page. Like an anchor tag which needs an `href` attribute to know where to send the user when they click on the link, the image tag needs a `src` attribute to provide the URL (absolute or relative) of where its located. Image tags are *empty tags* in that they have no content and, therefore, no end tag.

```

```

A required attribute for the image tag is the "alt" attribute. The value of the "alt" attribute is displayed if an image fails to load. Also, visually impaired people using a screen reader will hear the "alt" text as a description of the image.

```

```

## Fieldsets

---

One of the most overlooked tags in HTML is the fieldset tag. There are a few reasons why the fieldset tag is so great. One of the reasons is that it creates a clear division of content. You'll see the use of the fieldset in the "Building Forms in HTML" reading in this module.

## Reference for Common HTML Elements

This page lists all the common HTML elements. They are grouped by function to help you find what you have in mind easily. This is a curated list from Mozilla Developer's Network for your use in this prep work.

## Main sectioning elements

---

Element	Description
html	The HTML <code>html</code> element represents the root (top-level element) of an HTML document, so it is also referred to as the root element. All other elements must be descendants of this element.
head	The HTML <code>head</code> element contains machine-readable information (metadata) about the document, like its title, scripts, and style sheets.
body	The HTML <code>body</code> Element represents the content of an HTML document. There can be only one <code>body</code> element in a document.

## Document metadata

---

Element	Description
link	The HTML External Resource Link element ( <code>link</code> ) specifies relationships between the current document and an external resource. This element is most commonly used to link to stylesheets, but is also used to establish site icons (both "favicon" style icons and icons for the home screen and apps on mobile devices) among other things.
title	The HTML Title element ( <code>title</code> ) defines the document's title that is shown in a browser's title bar or a page's tab.

## Content sections

---

Element	Description
footer	The HTML <code>footer</code> element represents a footer for its nearest sectioning content or sectioning root element. A footer typically contains information about the author of the section, copyright data or links to related documents.
header	The HTML <code>header</code> element represents introductory content, typically a group of introductory or navigational aids. It may contain some heading elements but also a logo, a search form, an author name, and other elements.
h1, h2, etc.	The HTML <code>h1</code> – <code>h6</code> elements represent six levels of section headings. <code>h1</code> is the highest section level and <code>h6</code> is the lowest.
main	The HTML <code>main</code> element represents the dominant content of the <code>body</code> of a document. The main content area consists of content that is directly related to or expands upon the central topic of a document, or the central functionality of an application.
nav	The HTML <code>nav</code> element represents a section of a page whose purpose is to provide navigation links, either within the current document or to other documents. Common examples of navigation sections are menus, tables of contents, and indexes.
section	The HTML <code>section</code> element represents a standalone section — which doesn't have a more specific semantic element to represent it — contained within an HTML document.

## Text content

---

Element	Description
div	The HTML Content Division element ( <code>div</code> ) is the generic container for flow content. It has no effect on the content or layout until styled using CSS.
li	The HTML <code>li</code> element is used to represent an item in a list.
ol	The HTML <code>ol</code> element represents an ordered list of items — typically rendered as a numbered list.
p	The HTML <code>p</code> element represents a paragraph.
ul	The HTML <code>ul</code> element represents an unordered list of items, typically rendered as a bulleted list.

## Inline text semantics

---

Element	Description
a	The HTML <code>a</code> element (or anchor element), with its <code>href</code> attribute, creates a hyperlink to web pages, files, email addresses, locations in the same page, or anything else a URL can address.
br	The HTML <code>br</code> element produces a line break in text (carriage-return). It is useful for writing a poem or an address, where the division of lines is significant.
em	The HTML <code>em</code> element marks text that has stress emphasis. The <code>em</code> element can be nested, with each level of nesting indicating a greater degree of emphasis.
span	The HTML <code>span</code> element is a generic inline container for phrasing content, which does not inherently represent anything. It can be



Element	Description
	used to group elements for styling purposes (using the class or id attributes), or because they share attribute values, such as lang.
strong	The HTML Strong Importance Element ( <code>strong</code> ) indicates that its contents have strong importance, seriousness, or urgency. Browsers typically render the contents in bold type.

## Image and multimedia

---

Element	Description
img	The HTML <code>img</code> element embeds an image into the document.

## Table content

---

Element	Description
table	The HTML <code>table</code> element represents tabular data — that is, information presented in a two-dimensional table comprised of rows and columns of cells containing data.
tbody	The HTML Table Body element ( <code>tbody</code> ) encapsulates a set of table rows ( <code>tr</code> elements), indicating that they comprise the body of the table ( <code>table</code> ).
td	The HTML <code>td</code> element defines a cell of a table that contains data. It participates in the table model.
tfoot	The HTML <code>tfoot</code> element defines a set of rows summarizing the columns of the table.

Element	Description
th	The HTML <code>th</code> element defines a cell as header of a group of table cells. The exact nature of this group is defined by the <code>scope</code> and <code>headers</code> attributes.
thead	The HTML <code>thead</code> element defines a set of rows defining the head of the columns of the table.
tr	The HTML <code>tr</code> element defines a row of cells in a table. The row's cells can then be established using a mix of <code>td</code> (data cell) and <code>th</code> (header cell) elements.

## Forms

---

Element	Description
button	The HTML <code>button</code> element represents a clickable button, which can be used in forms or anywhere in a document that needs simple, standard button functionality.
fieldset	The HTML <code>fieldset</code> element is used to group several controls as well as labels ( <code>label</code> ) within a web form.
form	The HTML <code>form</code> element represents a document section that contains interactive controls for submitting information to a web server.
input	The HTML <code>input</code> element is used to create interactive controls for web-based forms in order to accept data from the user; a wide variety of types of input data and control widgets are available, depending on the device and user agent.
label	The HTML <code>label</code> element represents a caption for an item in a user interface.

Element	Description
legend	The HTML <code>legend</code> element represents a caption for the content of its parent <code>fieldset</code> .
option	The HTML <code>option</code> element is used to define an item contained in a <code>select</code> , an <code>optgroup</code> , or a <code>datalist</code> element. As such, <code>option</code> can represent menu items in popups and other lists of items in an HTML document.
select	The HTML <code>select</code> element represents a control that provides a menu of options
textarea	The HTML <code>textarea</code> element represents a multi-line plain-text editing control, useful when you want to allow users to enter a sizable amount of free-form text, for example a comment on a review or feedback form.

## Building Forms In HTML

Eventually, you'll want to have people send information to our applications so that you can send them stuff or get their opinions. You will do this by creating an HTML form. In this, you will design a simple form, implement it using the right HTML form controls and other HTML elements, add some styling via CSS, and cover how data is sent to a server.

**IMPORTANT:** Do not copy and paste the code from this page into your text editor. Type the code. Typing the code will give you better time to think about what's happening. That way, you can reflect on it. That way you can build durable knowledge.

## Getting started

---

1. If it's not open, open up your terminal. Change the present working directory to your home directory.
2. Make a new directory in your home directory named "html-form".
3. Change the present working directory into "html-form".
4. Start Visual Studio Code by running the command `code .` from the terminal in the "html-form" directory.
5. Create a new file.
6. Type the following HTML code into it.
7. Save the file as "add-user.html".

The code to *TYPE* in "add-user.html" is:

```
<!DOCTYPE html>
<html lang="en-US">
  <head>
    <meta charset="utf-8">
    <title>Learning the Forms</title>
  </head>
  <body>
    <h1>Hello, Forms!</h1>
  </body>
</html>
```

Open this in a browser. If you're on macOS, type `open add-user.html`. That will open the HTML file in your default browser. If you're on Windows, type `explorer.exe` in your Ubuntu command line. That will open Windows File Explorer. Double-click the "index" file that you see. That should open your file in your default browser.

## What are HTML forms?

---

HTML Forms are one of the oldest types of interaction between a person and an application. Forms allow users to enter data. Then, your application can decide to send it somewhere or do something with it locally.

An HTML Form is made of one or more input element types. Those types can be buttons, checkboxes, drop-downs, multi-selects, multi-line text fields, radio buttons, and single-line text fields. The single-line text fields can even require data entered to be of a specific format or value. Each form element can and should have a corresponding label to describe what the form elements expects. This helps sighted and blind people to interact with forms.

The main difference between a HTML form and a regular HTML document is that most of the time, the data collected by the form is sent to a web server. In that case, you need to set up a web server to receive and process the data. You'll get to that in the actual software engineering class.

## The form's design

---

Before starting to code anything with a visual element, it's always better to step back and take the time to think about what it will look like, how people will use your stuff. Designing a quick mock-up will help you to define the right set of data you want to ask people. From an interactive experience point of view, it's important to remember that the bigger your form, the more you risk losing users. Keep it simple and stay focused: ask only for that data you absolutely need.

In this, you will build a simple "add a user" form. It will look like this.

This form has four single-line text, one drop-down, and one multi-line text elements.

## Implementing your form HTML

Ok, now we're ready to go to HTML and code our form. To build our contact form, we will use the following HTML elements: `<form>`, `<label>`, `<input>`, `<textarea>`, and `<button>`.

### The `<form>` element

All HTML forms start with a `<form>` element like this:

```
<form action="/form-handling-url" method="post">

</form>
```

This element formally defines a form. It's a container element like a `<div>` or `<p>` element, but it also supports some specific attributes to configure the way the form behaves. Technically, all of its attributes are optional. It's standard practice to always set at least the `action` attribute and the `method` attribute:

- The `action` attribute defines the location (URL) where the form's collected data should be sent when it is submitted.
- The `method` attribute defines which HTTP method to send the data with. Browsers support only two values for this attribute: "get" and "post". You will use "post" 99% of the time.

For now, add the above `<form>` element into the HTML body after the `<h1>` element that reads "Hello, Forms!".

## The <label>, <input>, and <textarea> elements

The add user form is not complex. The form contains five text fields and one drop-down, each with a <label>. The input field for the first name is a single-line text field. The input field for the last name is a single-line text field. The input field for the email is an input of type *email*, a special text field that accepts only email addresses. The input field for expiration is an input of type *date*, a special text field that accepts only dates. The input field for the message is a multiline text field. The element for the role is a drop-down of predefined items.

In terms of HTML code we need something like the following to implement these form elements. Update your form code to look like the code below.

*Don't copy and paste this code. Type it out. Get used to typing HTML. Get used to indentation. Get used to the way that it feels to write code. This is one of the methods through which you learn. Don't cheat yourself.*

```
<form action="/form-handling-url" method="post">
  <fieldset>
    <legend>Add user</legend>

    <div>
      <label for="firstName">First name</label>
      <input type="text" id="firstName" name="first_name">
    </div>

    <div>
      <label for="lastName">Last name</label>
      <input type="text" id="lastName" name="last_name">
    </div>

    <div>
      <label for="email">Email</label>
      <input type="email" id="email" name="email_address">
    </div>

    <div>
      <label for="role">Role</label>
      <select id="role" name="user_role_name">
        <option value="admin">Admin</option>
        <option value="user">User</option>
        <option value="guest">Guest</option>
      </select>
    </div>

    <div>
      <label for="expiration">Expiration</label>
      <input type="date" id="expiration" name="expiration">
    </div>

    <div>
      <label for="bio">Bio</label>
      <textarea id="bio" name="bio"></textarea>
    </div>

    <div>
      <button type="submit">Add this person</button>
    </div>
  </fieldset>
</form>
```

```
</div>

</fieldset>
</form>
```

The `<div>` elements are there to provide some structure to your code, to group the `<label>` and inputs together. For usability and accessibility, each form element has an associated label. Each label uses the `for` attribute to refer to the value of the `id` attribute on its corresponding form element. This associates the label to the form control enabling touching and clicking the label to activate the corresponding form element.

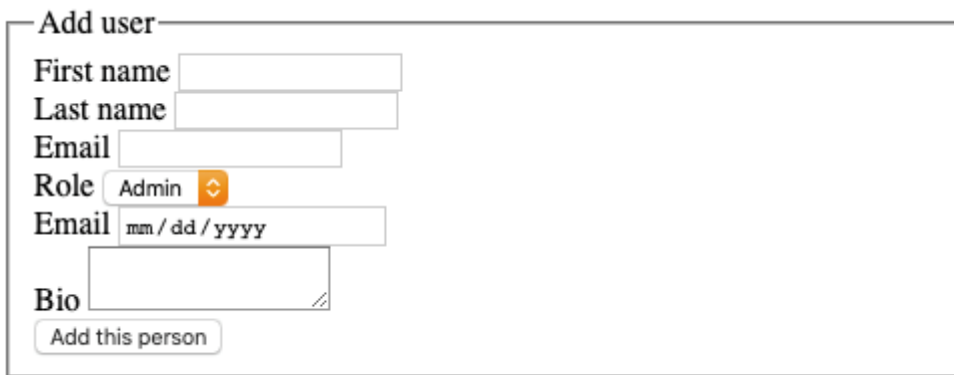
On the `<input>` element, the most important attribute is the `type` attribute. The `type` attribute tells the browser what kind of input it expects. In your form, you have three different types of inputs: `text`, `email`, and `date`. This will constrain the input to valid values. You won't, for example, be able to enter "Umbrella stand" for the `<input>` with the `date` value for its `type` attribute. It is not valid HTML to have a closing "input" tag. You should never write `<input type="text"></input>`.

The `<select>` element contains zero or more `<option>` elements. Having zero `<option>` elements makes very little sense, but it would be valid HTML. The content between the `<option>` and the `</option>` is what the browser shows in the drop-down. The option that the person selects, the contents of its `value` attribute becomes the value of the `<select>`. So, if a person chose "Guest" in the drop-down, the value of the `<select>` would be "guest" because that's the content of the `value` attribute of the `<option>` with the text "Guest". The `<select>` element should have a closing `</select>`.

A `<textarea>` is a multi-line text input. It must have a closing tag like `</textarea>`.

Finally, the `<button>` element represents a button for the form. By default, if a button is inside a form, that is between a `<form>` and a `</form>` tag, then it is a *submit* button. That means that the form will send the content of each of the form fields to a server for processing. The content between the button tag and its closing tag `</button>` is the content that will appear on the button's face.

A completely unstyled form looks like this.



## Styling Your Form

Create a new file in Visual Studio Code and save it as "style.css" in the same directory as the "add-user.html" file above. In the `<head>` element, add the following HTML code.

```
<link rel="stylesheet" href="style.css">
```

The `link` element tells the browser that it wants to link the specified file to this specific page. The `rel` attribute reads "stylesheet" which means that you want the rules in that stylesheet to apply to this page. Finally, the `href` attribute contains a URL to the file that you want it to use. This is a *relative* URL since it does not begin with "http://" or "<https://>". It tells the browser to get the file in the same directory as the HTML file that contains this.

Styling HTML forms is hard. Type the following CSS into the *style.css* file. After every rule, save the *style.css* file and refresh the Web page to see the changes that it makes. Read each of the comments to understand how it affects the

```
fieldset {
  /* Get rid of the border on the fieldset. */
  border: 0;
}

form {
  /* Center the form on the page. */
  margin: 0 auto;
  width: 600px;

  /* Curved border around the form. */
  padding: 1em;
  border: 1px solid #CCC;
  border-radius: 1em;

  /* Make all the fonts in the form the same. */
  font-family: Arial, Helvetica, sans-serif;
}

/* Every div after the first div in the form. */
form div + div {
  margin-top: 1em;
}

label {
  /* Uniform size & alignment for labels. */
  display: inline-block;
  width: 90px;
  text-align: right;
}

legend {
  /* Add a bottom border to the legend. */
  border-bottom: 1px solid black;
}

input,
textarea {
  /* You must explicitly override the fonts in textareas and inputs. */
  font-family: Arial, Helvetica, sans-serif;

  /* Uniform text field and textarea sizes. */
  width: 425px;
  box-sizing: border-box;

  /* Match form field borders */
  border: 1px solid #999;
}

input:focus,
textarea:focus {
  /* Additional highlight for focused elements */
  border-color: #000;
}
```



```

}

textarea {
  /* Align multiline text fields with their labels */
  vertical-align: top;

  /* Provide space to type some text */
  height: 5em;
}

select {
  /* You must explicitly override the font in a select. */
  font-family: Arial, Helvetica, sans-serif;

  /* Make the select as wide as the other form elements. */
  width: 425px;
}

button {
  /* Space the button to the right. */
  margin-left: .5em;
}

```

Now that you've done that, your form should look like the following. Note how the styling allows the form to look nearly identical on Windows and macOS. This is the real strength of CSS. You can allow people using any kind of computer and device to experience your form in the way that makes sense to them.

**Add user**

First name

Last name

Email

Role

Admin

⌵

Email

Bio

Add this person

ON A MAC

**Add user**

First name

Last name

Email

Role

Admin

▼

Email

Bio

Add this person

ON WINDOWS

## Media queries for narrow devices

The form is 600px wide no matter what and has padding of 16px all around it and a border of 1px all around it, which makes it about 635px wide. If the screen that the person is using to look at your form is less than 635px wide, they will have to scroll horizontally to use your form. That is considered bad design. To fix this, you can use a media query to adjust the way the form is laid out when the width is less than 650px, just to add some extra space in there to make sure everything appears to flow smoothly. Type the following at the end of your CSS file after the last rule (the `button` rule).

```
@media screen and (max-width: 650px) {

    /* remove the width from the form so that
       it is just as wide as the screen. */
    form {
        width: auto;
    }

    /* Turn labels into blocks so they can stretch
       across the entire screen. Also, get rid of
       the margin so that the text is left aligned.
       Finally, set the width to auto so that it
       isn't stuck at 90px. */
    label {
        display: block;
        margin: 0;
        text-align: left;
        width: auto;
    }

    /* Let the inputs, textareas, and selects span
       the entire screen, too. */
    input,
    select,
    textarea {
        width: 100%;
    }
}
```

Now, you can resize your screen and see the form change its layout based on the width of the device.

### Add user

---

First name

Last name

Email

Role

Admin

Email

Bio

Add this person

## Using CSS To Layout HTML Elements

This teaches the fundamentals of CSS that you will use to create any modern Web site. Now that you've become familiar with selectors, properties, and values, it's time to dive into all of the ways that you can use CSS to layout elements on a Web page.

## The "display" property

The "display" property is the most important property for controlling layout in CSS. Every element has a default display value. For most elements, that value is either "block" or "inline". In this material, "block" elements are sometimes referred to as "block-level" elements. You will come across both of those terms in many sources.

The `div` element is the standard block-level element. A block-level element starts on a new line and stretches out to the left and right as far as it can. Other common block-level elements are `p`, `form`, `header`, `footer`, and `section`.

The `span` element is the standard inline element. An inline element can wrap some text inside a paragraph without disrupting the flow of that paragraph. The `a` element is the most common inline element that has extra functionality, since you use them for links. Another common display value is "none". Some specialized elements such as `script` use this as their default. The `script` element is commonly used with JavaScript to hide and show elements without really deleting and recreating them. This is something that you will learn about in the online class.

A display value of "none" is different from visibility. Setting display to "none" will render the page as though the element does not exist. Setting visibility to "hidden" will hide the element, but the element will still take up the space it would if it was fully visible.

There are plenty of more exotic display values, such as "list-item" and "table" which are special types of display for list items and tables. Here is an [exhaustive list](#).

## The "margin" property set to "auto"

---

```
#main {  
  width: 600px;  
  margin: 0 auto;  
}
```

Setting the width of a block-level element will prevent it from stretching out to the edges of its container to the left and right. Then, you can set the left and right margins to auto to horizontally center that element within its container. The element will take up the width you specify, then the remaining space will be split evenly between the two margins.

The only problem occurs when the browser window is narrower than the width of your element. The browser resolves this by creating a horizontal scrollbar on the page.

## Using "max-width" rather than "width"

---

```
#main {  
  max-width: 600px;  
  margin: 0 auto;  
}
```

```
}
```

Using max-width instead of width in this situation will improve the browser's handling of small windows. This is important when making a site usable on mobile. Resize this page to check it out!

By the way, max-width is supported by all major browsers including IE7+ so you shouldn't be afraid of using it.

## Understanding the "box model" of layout

While diving into width, you should learn about width's big caveat: the box model. When you set the width of an element, the element can actually appear bigger than what you set it. Sometimes the element's border and padding will stretch out the element beyond the width that you specified. Look at the following example, where two elements with the same width value end up different sizes in the result.

```
.simple {  
  width: 500px;  
  margin: 20px auto;  
}  
  
.fancy {  
  width: 500px;  
  margin: 20px auto;  
  padding: 50px;  
  border-width: 10px;  
}
```

For generations, the solution to this problem has been extra math. CSS authors have always just written a smaller width value than what they wanted, subtracting out the padding and border. This was hard. So a new CSS property called "box-sizing" was created. When you set "box-sizing" to the value of "border-box" on an element, the padding and border of that element no longer increase its width. Here are the improved selectors.

```
.simple {  
  width: 500px;  
  margin: 20px auto;  
  box-sizing: border-box;  
}  
  
.fancy {  
  width: 500px;  
  margin: 20px auto;  
  padding: 50px;  
}
```

```
border-width: 10px;  
box-sizing: border-box;  
}
```

Many authors just set "border-box" on all elements on all their pages.

```
* { box-sizing: border-box; }
```

## Both yoga and CSS have positions

In order to make more complex layouts, you will need to use the "position" property. It has a bunch of possible values, and their names make no sense to people new to CSS which makes them hard to remember unless you use them.

```
.static {  
  position: static;  
}
```

For all HTML elements, "static" is the default value. An element with the CSS rule `position: static;` is not positioned in any special way and is said to be "unpositioned". If an element has any other value for its "position", then it is said to be "positioned".

```
.relative-like-static {  
  position: relative;  
}
```

By itself, "relative" acts just like "static". So, this does not really do anything.

```
.relative-and-obvious {  
  position: relative;  
  top: -20px;  
  left: 20px;  
}
```

Setting the "top", "right", "bottom", and "left" properties of a "relatively"-positioned element will cause it to be adjusted away from its normal position. Other content will not be adjusted to fit into any gap left by the element.

```
.fixed {  
  position: fixed;  
  bottom: 0;  
  right: 0;  
  width: 200px;  
}
```

A "fixed" element is positioned relative to the viewport, which means it always stays in the same place even if the page is scrolled. As with relative, the "top", "right", "bottom", and "left" properties are used.

A "fixed" element does not leave a gap in the page where it would normally have been located.

```
.absolute {
  position: absolute;
  top: 120px;
  right: 0;
  width: 300px;
  height: 200px;
}
```

Elements with the "display" property set to "absolute" is the trickiest position value. The value "absolute" behaves like "fixed" except relative to the nearest positioned ancestor instead of relative to the viewport. If an absolutely-positioned element has no positioned ancestors, it uses the document body, and still moves along with page scrolling. Remember, a "positioned" element is one whose position is anything except static.

## Responding to different devices

---

"Responsive Design" is the strategy of making a site that "responds" to the browser and device that it is being shown on by changing the layout to adapt to the available space. Media queries are the most powerful tool for doing this.

```
@media screen and (min-width: 600px) {
  nav {
    position: absolute;
    top: 0;
    width: 100%;
    height: 50px;
  }
  main {
    margin-top: 50px;
  }
}
@media screen and (max-width: 599px) {
  nav {
    position: static;
    height: auto;
  }
}
```

When the width of the screen falls below 600px, then the nav changes to be inline rather than stuck to the top because it changes the "position" and "height" elements.

## Hybrid display "inline-block"

---

Web developers got sick of trying to hack the best of the "inline" and "block" display styles together. Elements with "inline-block" displays are like "inline" elements that can have widths and heights. You can use "inline-block" for total page layout.

- Elements with "inline-block" are affected by the "vertical-align" property, which you probably want set to top.
- You need to set the width of each column
- There will be a gap between the columns if there is any whitespace between them in the HTML

```
nav {  
  display: inline-block;  
  vertical-align: top;  
  width: 25%;  
}
```

## Get rid of blocks, use flexible box!

---

The "flexbox" layout mode redefines how we do layouts in CSS. You have an assignment to do *Flexbox Froggy* and *Flexbox Defense*. These teach you well how to use "flexbox". Conceptually, though, it provides a way to build rows and columns and align the elements within those rows or columns.

## The all powerful grid layout

---

The "grid" layout mode is the newest of CSS layouts. You can use it to break an element's layout space into a grid and assign child elements to sectors in the grid. You have an assignment to complete the *CSS Grid Garden* to learn how to use it.



# CSS Transitions

CSS transitions provide a way to control animation speed when changing CSS properties. Instead of having property changes take effect immediately, you can cause the changes in a property to take place over a period of time. For example, if you change the color of an element from white to black, usually the change is instantaneous. With CSS transitions enabled, changes occur at time intervals that follow an acceleration curve, all of which can be customized.

Animations that involve transitioning between two states are often called implicit transitions as the states in between the start and final states are implicitly defined by the browser.

CSS transitions let you decide which properties to animate (by listing them explicitly), when the animation will start (by setting a delay), how long the transition will last (by setting a duration), and how the transition will run (by defining a timing function, e.g. linearly or quick at the beginning, slow at the end).

## Defining transitions

---

CSS Transitions are controlled using the shorthand transition property. This is the best way to configure transitions, as it makes it easier to avoid out of sync parameters, which can be very frustrating to have to spend lots of time debugging in CSS.

You can control the individual components of the transition with the following sub-properties:

Sub-property	Definition
transition-property	Specifies the name or names of the CSS properties to which transitions should be applied. Only properties listed here are animated during transitions; changes to all other properties occur instantaneously as usual.
transition-duration	Specifies the duration over which transitions should occur. You can specify a single duration that applies to all properties

Sub-property	Definition
	during the transition, or multiple values to allow each property to transition over a different period of time.
transition-delay	Defines how long to wait between the time a property is changed and the transition actually begins.

## Examples

---

This example performs a four-second font size transition with a two-second delay between the time the user mouses over the element and the beginning of the animation effect:

```
#delay {  
  font-size: 14px;  
  transition-property: font-size;  
  transition-duration: 4s;  
  transition-delay: 2s;  
}  
  
#delay:hover {  
  font-size: 36px;  
}
```

When the mouse hovers over it, after a delay of two seconds, a four-second transition begins which changes the font size of the text from its normal size to 36px.

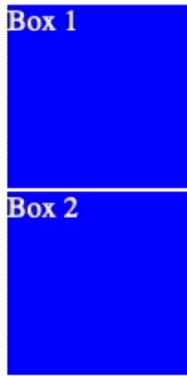
Hover over this



In the following, any element with the "box" class will have combined transitions for: width, height, background-color, transform.

```
.box {  
  border-style: solid;  
  border-width: 1px;  
  display: block;  
  width: 100px;  
  height: 100px;  
  background-color: #0000FF;  
  transition: width 2s, height 2s, background-color 2s, transform 2s;  
}  
  
.box:hover {  
  background-color: #FFCCCC;  
  width: 200px;  
  height: 200px;  
  transform: rotate(180deg);  
}
```

When the mouse hovers over a box, it spins due to the rotate transform. Its width and height change. Its background color changes.



You can check out both of these demos at the [Transition Examples](#) on CodePen.

## What can you affect with this?

---

You can't apply transitions to every CSS property there is. Here is the [list of animatable CSS properties](#). If it's not in that list, then you can't animate it.

Glaringly absent from that list are any CSS properties that allow you to position elements on a Web page, properties like `left` or `bottom`. The work around is to animate the *margin* of the element.