An HTML file has both a head and a body. The head is where you put information about your HTML file, like its title.

The body is where you put your content, such as text, images, and links. The content in the body is what will be visible on the actual page.

The body goes inside the <html> tags, right after the <head> tags,

Html tag

Head tag

Title tag

Close Head tag

Body tag

<!DOCTYPE html>

<html>

<head>

<title></title>

</head>

<body>

<a href="http://www.codeacademy.com">My Favorite Site </a>

</body>

</html>

///////////////////////////////////////////////////////////////////////////////////////

Image as a link:

<!DOCTYPE html>

<html>

<head>

<title></title>

</head>

<body>

<img src="http://www.nps.gov/pwr/deva/images/00542D46-FA04-DAE3-3B63820EE4116240.jpg" />

<a href="codeacademy.com"><img src= http://www.nps.gov/pwr/deva/images/00542D46-FA04-DAE3-3B63820EE4116240.jpg > </a>

</body>

</html>

/////////////////////////////////////////////////////////////////////////////////

<!DOCTYPE html>

<html>

<head>

<title></title>

</head>

<body>

<a href="htt://www.google.com"> <img src="http://www.nps.gov/pwr/noca/images/2C5DAAAF-1DD8-B71C-0ED1D7722A073EF1.jpg" /></a>

<img src="http://www.nps.gov/imr/grte/images/E9D84AD5-1DD8-B71C-0EBF9FABFFB7CFFE.jpg"/>

<a href="http://www.nps.gov/ner/images/170A0D1A-1DD8-B71C-0E3B12E4F199DE9A.jpg">National Parks></a>

<img src="http://www.sense-lang.org/img/inconTyping.png"> <!-- image only --> <p>

<a href="http://game.giveawayoftheday.com/" ><img src="http://www.sense-lang.org/img/inconGame.png"></a><!--image with link to pg -->

</body>

</html>

Link is an image

Plain image

Link to an image

////////////////////////////////////////////////////////////////////////////////////////////////////

**Ordered lists**= a list that is numbered

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

Attributes in the opening tags:

<p style="font-size: 12px">

<h2 style="color:red">

<h2 style="color: green; font-size:12px">

Background:

<p style="background-color: red;">Hello!</p>

Moving text around:

<h1 style="text-align:center">

Bold; emphasize:

Surround those words with opening tag<strong> and closing tag </strong>.

Surround the word or words with the opening tag <em> and closing tag </em>.

Tables, able rows, table data:

<table> </table>

<tr></tr>

<td></td>

Head and body of table:

<thead> </thead>

<tbody>  </tbody>

<html>

<head>

<title>Table Time</title>

</head>

<body>

<table border="1px">

<thead>

<tr>

<th>Famous Monster

</th>

<th>Birth Year

</th>

</tr>

</thead>

<tbody>

<tr>

<td>King Kong</td>

<td>1933</td>

</tr>

<tr>

<td>Dracula</td>

<td>1897</td>

</tr>

<tr>

<td>Bride of Frankenstein</td>

<td>1935</td>

</tr>

</tbody>

</table>

</body>

</html>

Title of a table: use the **colspan** attribute for the <th> tag. By default, table cells take up 1 column. If we want a table cell to take up the space of 3 columns instead of 1, we can set the colspan attribute to 3.

It looks like this:

<th colspan="3">3 columns across!</th>

////////////////////////////////////////////////////////////////////////////////////////////////////////////////

<!DOCTYPE html>

<html>

<head>

<title>Result</title>

</head>

<body>

<div style="width:50px; height:50px; background-color:red"></div>

<div style="width:50px; height:50px; background-color:blue"></div>

<div style="width:50px; height:50px; background-color:green"></div>

<a href="http://www.google.com"><div style="width:50px; height:50px; background-color:yellow"></div>

</body>

</html>

While <div> allows you to divide your webpage up into pieces you can style individually, <span> allows you to control styling for smaller parts of your page, such as text. For example, if you always want the first word of your paragraphs to be red, you can wrap each first word in <span></span> tags and make them red using CSS!

<!DOCTYPE html>

<html>

<head>

<title>Ye Olde Fancye Booke</title>

<link type="text/css" rel="stylesheet" href="stylesheet.css"/>

</head>

<body>

<h3>Ye Olde Storye</h3>

<p><span>A</span> long time ago there was an intrepid young student who wanted to learn CSS...</p>

</body>

</html>

///////////////////////////////////////////////////////////////////////////////////////////////

CSS

<style></style> or CSS tags go inside the<head></head> of your webpage

Because nothing ever goes between <link></link> tags, it's okay to use a single set of<>s to be the opening *and* closing tags.

<link type="text/css" rel="stylesheet" href="CS

<link type="text/css" rel="stylesheet" href="stylesheet.css"/>

**hexadecimal values**: Hexadecimal counting is **base-16**. Each digit can be the numbers 0 through 9 **or the letters a through f**!

The font-size unit **em** is a **relative** measure: one em is equal to the default font size on whatever screen the user is using. That makes it great for smartphone screens.

//////////////////////////////////////////////////////////////////////////////////////////////////////////////

Buttons and links

<!DOCTYPE html>

<html>

<head>

<link type="text/css" rel="stylesheet" href="stylesheet.css"/>

<title>About Me</title>

</head>

<body>

<img src="http://s3.amazonaws.com/codecademy-blog/assets/46838757.png"/>

<p>We're Codecademy! We're here to help you learn to code.</p><br/><br/>

<div>

<a href= "http://www.google.com">Goggle it!</a>

</div>

</body>

</html>

img {

display: block;

height: 100px;

width: 300px;

margin: auto;

}

p {

text-align: center;

font-family: Garamond, serif;

font-size: 18px;

}

/\*Start adding your CSS below!\*/

div {

height: 50px;

width: 120px;

border-color: #6495ed;

background-color: #bcd2ee;

border-style:solid;

border-width: 2px;

border-radius: 5px;

margin:auto;

text-align:center;

}

a {

text-decoration: none;

font-family: Verdana;

}

///////////////////////////////////////////////////////////////////////////////////////////

<h2>CatPhotoApp</h2>

h2{

color:blue

}

<style>h2 {color: blue;}</style>.

<p>Kitty ipsum dolor sit amet, shed everywhere shed everywhere stretching attack your ankles chase the red dot, hairball run catnip eat the grass sniff.</p>

/////////////////////////////////////////////////////////////////////////////

Create an "urgently-red" class that gives an element the font-color of red, but add !important to the class to ensure the element is rendered as being red. Immediately below your "urgently-red" class declaration, create a "blue-text" class that gives an element the font-color of blue. Apply both classes to your h2 element.

You can add more than one class to an element by separating the class declarations with a space, like this: <h2 class='green-text giant-text'>This will be giant green text</h2>.

Sometimes HTML elements will receive conflicting information from CSS classes as to how they should be styled.

If there's a conflict in the CSS, the browser will use whichever style declaration is closest to the bottom of the CSS document (whichever declaration comes last). Note that in-line style declarations are the final authority in how an HTML element will be rendered.

There's one way to ensure that an element is rendered with a certain style, regardless of where that declaration is located. That one way is to use !important.

In case you're curious, this is the priority hierarchy for element styles: !important beats inline styles, which beats CSS class selectors, which beats CSS selector. That is, !important trumps all other styles, and inline styles trump style tag declarations.

Here's an example of a CSS style that uses !important:<style> .urgently-blue { color: blue !important; } </style>.

Now see if you can make sure the h2 element is rendered in the color red without removing the "blue-text" class, doing an in-line styling, and without changing the sequence of CSS class declarations.

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

Multiple selectors:

<!DOCTYPE html>

<html>

<head>

<link type="text/css" rel="stylesheet" href="stylesheet.css"/>

<title>Result</title>

</head>

<body>

<h3>I'm plain old font!</h3>

<div>

<h3>Me, too!</h3>

<div>

<h3>Me three!</h3>

<div>

<h3>Forget you guys. I'm about to be red!</h3>

</div>

</div>

</div>

</body>

</html>

div div div h3{

color:red;

}

//////////////////////////////////////////////////////////////////////////////

There's also a very special selector you can use to apply CSS styling to *every element* on the page: the \* selector. For example, if you type

\* {

border: 2px solid black;

}

Reach an element that is a child of another element like this:

div div p { /\* Some CSS \*/ }

Certain selectors will "override" others if they have a greater**specificity value**. ul li p { is more specific CSS than just p {, so when CSS sees tags that are *both* <p> tags *and* happen to be inside unordered lists, it will apply the more specific styling (ul li p {) to the text inside the lists.

There are two selectors that are even more specific than nested selectors like the ones above: **classes** and **IDs**. Check them out in the editor to the right.

There are two important selectors you can use in addition to the universal selector and HTML elements: **classes** and **IDs**.

Classes are useful when you have a bunch of elements that should all receive the same styling. Rather than applying the same rules to several selectors, you can simply apply the same class to all those HTML elements, then define the styling for that class in the CSS tab.

Classes are assigned to HTML elements with the word class and an equals sign, like so:

<div class="square"></div>

<img class="square"/>

<td class="square"></td>

Classes are identified in CSS with a dot (.), like so:

.square {

height: 100px;

width: 100px;

}

This allows you to take elements of different types and give them the same styling.

IDs, on the other hand, are great for when you have exactly *one* element that should receive a certain kind of styling.

IDs are assigned to HTML elements with the word id and an equals sign:

<div id="first"></div>

<div id="second"></div>

<p id="intro"></p>

IDs are identified in CSS with a pound sign (#):

#first {

height: 50px;

}

#second {

height: 100px;

}

#intro {

color: #FF0000;

}

This allows you to apply style to a single instance of a selector, rather than *all*instances.

.class

#ID

///////////////////////////////////////////////////////////////////////////////////////////

A **pseudo-class selector** is a way of accessing HTML items that aren't part of the document tree (remember the tree structure we talked about earlier?). For instance, it's very easy to see where a link is in the tree. But where would you find information about whether a link had been clicked on or not? It isn't there!

Pseudo-class selectors let us style these kinds of changes in our HTML document. For example, we saw we could change a link's text-decoration property to make it something other than blue and underlined. Using pseudo selectors, you can control the appearance of unvisited and visited links—even links the user is hovering over but hasn't clicked!

The CSS syntax for pseudo selectors is

selector:pseudo-class\_selector {

property: value;

}

It's just that little extra colon (:).

a:link: An unvisited link.  
a:visited: A visited link.  
a:hover: A link you're hovering your mouse over.

Another useful pseudo-class selector is first-child. It's used to apply styling to *only* the elements that are the first children of their parents. For instance:

p:first-child {

color: red;

}

Would make all paragraphs that are the first children of their parent elements red.

Well done! You can actually select *any* child of an element after the first child with the pseudo-class selector nth-child; you just add the child's number in parentheses after the pseudo-class selector. For example,

p:nth-child(2) {

color: red;

}

Would turn every paragraph that is the*second* child of its parent element red.

The element that **is the child** goes before:nth-child; its parent element is the element that contains it.

body :nth-child(4) {

font-size: 26px;

}

!!!Make sure to leave a space btw. body and :nth child

Cricles (divs):

div {

display: inline-block;

margin-left: 5px;

width: 100px;

height: 100px;

border-radius:100%;

border-width:2px;

border-style:solid;

border-color:black;

}

////////////////////////////////////////////////////////////////////////

Positioning

Elements populate the page in what's known as the **CSS box model**. Each HTML element is like a tiny box or container that holds the pictures and text you specify.

We can change all this with the first positioning property we'll learn: the **display**property. We'll learn about four possible values.

**block**: This makes the element a block box. It won't let anything sit next to it on the page! It takes up the full width.

**inline-block**: This makes the element a block box, but will allow other elements to sit next to it on the same line. The inline-block value allows you to put several block elements on the same line. Theinline value places all your elements next to one another, but not as blocks: they don't keep their dimensions. The inline display value is better suited for HTML elements that are blocks by default, such as headers and paragraphs.

**inline**: This makes the element sit on the same line as another element, but without formatting it like a block. It only takes up as much width as it needs (not the whole line).

**none**: This makes the element and its content disappear from the page entirely!

The **margin** is the space around the element. The larger the margin, the more space between our element and the elements around it. We can adjust the margin to move our HTML elements closer to or farther from each other. Set an element's margins all at once: you just start from the top margin and go around clockwise (going from top to right to bottom to left). For instance,

margin: 1px 2px 3px 4px;

The **border** is the edge of the element. It's what we've been making visible every time we set the border property.

The **padding** is the spacing between the content and the border. We can adjust this value with CSS to move the border closer to or farther from the content. Padding can be set in two ways, just like your margins. You can either select them individually, like this:

padding-top: */\*some value\*/*

padding-right: */\*some value\*/*

padding-bottom: */\*some value\*/*

padding-left: */\*some-value\*/*

Or select them all in one declaration, like this:

padding: value value value value;

The **content** is the actual "stuff" in the box. If we're talking about a <p> element, the "stuff" is the text of the paragraph.

One way is to use **floats**. When you float an element on the page, you're telling the webpage: "I'm about to tell you where to put this element, but you have to put it into the **flow** of other elements." This means that if you have several elements all floating, they all know the others are there and don't land on top of each other.

You can think of the HTML page as sort of like a sea, and floating elements as boats on it: all the boats have positions on the sea, and they all see and steer clear of each other.

(Some of the positioning methods we'll learn in upcoming sections *can* accidentally drop elements on top of each other.)

If you tell an element to clear: left, it will immediately move below any floating elements on the left side of the page; it can also clear elements on the right. If you tell it to clear: both, it will get out of the way of elements floating on the left *and* right!

The syntax is what you're used to:

element {

clear: */\*right, left, or both\*/*

}

The first type of positioning is **absolute** positioning. When an element is set to position: absolute, it's then positioned in relation to the first parent element it has that *doesn't* have position: static. If there's no such element, the element with position: absolute gets positioned relative to <html>.

To show you how this works, we've set the#outer div to have absolute positioning. This means that when you position the #inner div, it will be relative to #outer. (If #outer had the default positioning of static, then #innerwould get positioned relative to the entire HTML document.)

Finally, **fixed** positioning anchors an element to the browser window—you can think of it as gluing the element to the screen. If you scroll up and down, the fixed element stays put even as other elements scroll past.