Jerry Gensiejewski

Html Notes

Things to Learn

**Html**

* Media queries Media queries let you adapt your site or app depending on the presence or value of various device characteristics and parameters.
* Algorithmic challenged codewarrior
* [Unsplash](http://www.unsplash.com/), [Stocksnap](http://www.stocksnap.io/), [Pexels](http://www.pexels.com/), [Picjumbo](http://www.picjumbo.com/), [Death To The Stock Photo](http://www.deathtothestockphoto.com/), [Gratisography](http://www.gratisography.com/) (silly photos), [Negative Space](http://www.negativespace.co/), [Splitshire](http://www.splitshire.com/) (now also featuring videos), [Freestocks](http://www.freestocks.org/), [TookAPic](https://stock.tookapic.com/photos?filter=free), [Picography](http://www.picography.co/), [MMT Stock](http://www.mmtstock.com/), [Kaboom Pics](http://www.kaboompics.com/), and [Graphic Burger](http://www.graphicburger.com/).

**Links**

* Flaticon.com for social media icons
* <https://www.google.com/fonts>
* validators
* <https://jigsaw.w3.org/css-validator/>
* <https://validator.w3.org/>
* Fonts
* <http://www.cssfontstack.com/>

<meta name="viewport" content="width=device-width,initial-scale=1">

<meta charset="utf-8">

<meta name="viewport" content="width=device-width,initial-scale=1">

**Html Information**

So what should you do? If you need interactivity, HTML and [JavaScript](https://developer.mozilla.org/en-US/docs/Glossary/JavaScript) can readily get the job done for you with no need for Java applets or outdated ActiveX/BHO technology. Instead of relying on Adobe Flash, you should use [HTML5 video](https://developer.mozilla.org/en-US/docs/Learn/HTML/Howto/Add_audio_or_video_content_to_a_webpage) for your media needs, [SVG](https://developer.mozilla.org/en-US/docs/Learn/HTML/Howto/Add_vector_image_to_a_webpage) for vector graphics, and [Canvas](https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial) for complex images and animations. [Peter Elst was already writing some years ago](https://plus.google.com/+PeterElst/posts/P5t4pFhptvp) that Adobe Flash is rarely the right tool for the job. As for ActiveX, even Microsoft's [Edge](https://developer.mozilla.org/en-US/docs/Glossary/Microsoft_Edge) browser no longer supports it.

All HTML documents must start with a document type declaration: <!DOCTYPE html>.

<html lang="en-US"> to set html to English to optimize search engines

The HTML document itself begins with <html> and ends with </html>.

The visible part of the HTML document is between <body> and </body>.

Html paragraphs <p>

Html links <a>

HTML buttons are defined with the <button> tag:

HTML lists are defined with the <ul> (unordered/bullet list) or the <ol> (ordered/numbered list) tag, followed by <li> tags (list items):

<ul>

<li>Coffee</li>

<li>Tea</li>

<li>Milk</li>

</ul>

<ol>

<li>Coffee</li>

<li>Tea</li>

<li>Milk</li>

</ol>

The <div> tag defines a division or a section in an HTML document.

The <div> element is often used as a container for other HTML elements to style them with CSS or to perform certain tasks with JavaScript.

<br> ends the line of text to make it more neat

Font awesome icons

Margin: auto;

This CSS shorthand for the margin property would set the top and bottom margins to a value of 0, while the left and right would use "auto." This essentially takes any space that is available and divides it evenly between the two sides of the viewport window, effectively centering the element on the page.

A block-level element always starts on a new line and takes up the full width available (stretches out to the left and right as far as it can).

An inline element does not start on a new line and only takes up as much width as necessary.

An RGB color value is specified with: rgb( RED , GREEN , BLUE ).

Each parameter defines the intensity of the color as an integer between 0 and 255.

For example, rgb(0,0,255) is rendered as blue, because the blue parameter is set to its highest value (255) and the others are set to 0.

To get color matching use paletton.com

**HTML Elements**

* <sup> superscript
* <sub> subscript
* <br> line break
* <hr> to create a break between themes such as a change of topic in a book
* <strong> bold
* <em> italics
* <blockquote> is used for longer quotes that take up an entire paragraph
* <q> is used for shorter quotes that sit within a paragraph
* <abbr> for abbreviations
* <cite> When you are referencing a piece of work such as a book, the <cite> element can be used to indicate where the citation is from.
* <dfn> the <dfn> element id used to indicate the defining instance of a new term.
* <address> the <address> element has quite a specific use: to contain contact details for the author of the page
* <ins>
* <del> the <ins> element can be used to show content that has been inserted into a document ,while the <del> element can show text that has been deleted from it.
* <s> the <s> element indicates something that is no longer accurate or relevant(but that should not be deleted). Visually the <s> content will be shown with a line through the center of it.
* <dl> definition list
* <dt> this is used to contain the term being defined
* <dd> this is used to contain the definition
* <a> links
* mailto: for Email links <a href=”mailto:jerryg2212@gmail.com>Jerry’s email</a>
* target of you want a link to open on a new window, you can use the target attribute on the opening <a> tag. The value of this attribute should be \_blank.
* <a href=<http://www.google.com> target=”\_blank”> Google’s website</a>
* <figure> contains images and their captions
* <figurecaption> captions an image
* <thread> the heading of the table should sit inside the <thread> element
* <tbody> the body should sit inside the <tbody> element
* <tfloat> the footer belongs inside the <tfoot> element
* <form> form controls live inside this element. This element should always carry the action attribute and will usually have a method and id attribute to.
* <input> is used to create several different form controls. The value of the type attribute determines what kind of input they will be creating.
* <textarea> is used to create a multiline input. Any text that appears between the opening <textarea> and closing </textarea> tags will appear in the text box when the page loads.
* <button> elements in side appear as a button
* <label> Each form control should have its own <label> element as this makes the form accessible to vision-impaired users.
* <fieldset> You can group related form controls together inside the <fieldset> element. This is particularly helpful for linger forms.
* <legend> can come directly after the opening <fieldset> tag and contains a caption which helps identify the purpose of that group of form controls.
* <iframe> is like a little window that has been cut into your page it can be used to shoe other
* <video> this element has a number of attributes which allow you to control video playback
* <source> To specify the location of the file to be played, you can use the <source> element inside the <video> element(This should replace the attribute on the opening <video> tag.)
* <audio> to display audio
* <source> to have multiple sources instead on just one src attribute
* <code>: For marking up generic pieces of computer code.
* <pre>: For retaining whitespace (generally code blocks) — if you use indentation or excess whitespace inside your text, browsers will ignore it and you will not see it on your rendered page. If you wrap the text in <pre></pre> tags however, your whitespace will be rendered identically to how you see it in your text editor.
* <var>: For specifically marking up variable names.
* <kbd>: For marking up keyboard (and other types of) input entered into the computer.
* <samp>: For marking up the output of a computer program.
* <time> HTML also provides the [<time>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/time) element for marking up times and dates in a machine-readable format. For example:
* **Layout Elements**
  + <header> <footer> <aside> <article> <nav> <content> <section>
  + <hgroup> - the purpose of the <hgroup> element is to group together a set of one or more <h1> through <h6> elements so that they are treated as one single heading.

**End Layout Elements**

* HTML5 allows web page authors to place an <a> element around a block level element that contains child elements. This allows you to turn an entire block into a link.
* Older browsers that do not know the new HTML5 elements will automatically treat them as inline elements. Therefore, to help older browsers , you should include the line of CSS on the left which indicates that the elements should be block level elements.

**Tables**

* <table> - for a table
* <tr> for table row
* <td> for a cell in a table
* <th> table headers — special cells that go at the start of a row or column and define the type of data that row or column contains
* <col> - see under Notes on tables
* <colgroup> - see under Notes on tables
* <span> The HTML <span> element is a generic inline container for phrasing content, which does not inherently represent anything. It can be used to group elements for styling purposes (using the class or id attributes), or because they share attribute values, such as lang. It should be used only when no other semantic element is appropriate. <span> is very much like a <div> element, but <div> is a block-level element whereas a <span> is an inline element.
* <em> There are times when you will want something to look different based on where it is in the document. There are a number of selectors that can help you here, but for now we will look at just a couple. In our document are two <em> elements — one inside a paragraph and the other inside a list item. To select only an <em> that is nested inside an <li> element I can use a selector called the descendant combinator, which simply takes the form of a space between two other selectors.
* <thread> the heading of the table should sit inside the <thread> element
* <tbody> the body should sit inside the <tbody> element
* <tfloat> the footer belongs inside the <tfoot> element
* collapse
* Adjacent cells have shared borders (the collapsed-border table rendering model).
* separate
* Adjacent cells have distinct borders (the separated-border table rendering model).

**End HTML Elements**

There are many [open source](https://en.wikipedia.org/wiki/Open_source) tools for web development such as [BerkeleyDB](https://en.wikipedia.org/wiki/Berkeley_DB), [GlassFish](https://en.wikipedia.org/wiki/GlassFish), [LAMP](https://en.wikipedia.org/wiki/LAMP_(software_bundle)) ([Linux](https://en.wikipedia.org/wiki/Linux), [Apache](https://en.wikipedia.org/wiki/Apache_HTTP_Server), [MySQL](https://en.wikipedia.org/wiki/MySQL), [PHP](https://en.wikipedia.org/wiki/PHP)) stack and [Perl/Plack](https://en.wikipedia.org/wiki/Plack_(software)). This has kept the cost of learning web development to a minimum.

See also

Mozilla developer newsletter

Our newsletter for web developers, which is a great resource for all levels of experience.

Learn JavaScript

An excellent resource for aspiring web developers — Learn JavaScript in an interactive environment, with short lessons and interactive tests, guided by automated assessment. The first 40 lessons are free, and the complete course is available for a small one-time payment.

Web demystified

A great series of videos explaining web fundamentals, aimed at complete beginners to web development. Created by Jérémie Patonnier.

Codecademy

A great interactive site for learning programming languages from scratch.

BitDegree

Basic coding theory with a gamified learning process. Mainly focused on beginners.

Code.org

Basic coding theory and practice, mainly aimed at children/complete beginners.

EXLskills

Free and open courses for learning tech skills, with mentorship and project-based learning.

freeCodeCamp.org

Interactive site with tutorials and projects to learn web development.

Web literacy map

A framework for entry-level web literacy and 21st century skills, which also provides access to teaching activities sorted by category.

Teaching activities

A series of teaching activities for teaching (and learning) created by the Mozilla Foundation, covering everything from basic web literacy and privacy to JavaScript and hacking Minecraft.

Edabit

Thousands of interactive JavaScript challenges.

**A graphics editor, like GIMP, Paint.NET, Photoshop, or XD, to make images or** graphics for your web pages.

This is ok. It contains the content you need, and is nicely stylable using CSS. But there is a problem here: there is nothing that semantically links the image to its caption, which can cause problems for screen readers. For example, when you have 50 images and captions, which caption goes with which image?

A better solution, is to use the HTML5 <figure> and <figcaption> elements. These are created for exactly this purpose: to provide a semantic container for figures, and to clearly link the figure to the caption. Our above example could be rewritten like this:

A figure doesn't have to be an image. It is an independent unit of content that:

Expresses your meaning in a compact, easy-to-grasp way.

Could go in several places in the page's linear flow.

Provides essential information supporting the main text.

A figure could be several images, a code snippet, audio, video, equations, a table, or something else.

**Notes on Tables**

We need a way to get "Animals", "Hippopotamus", and "Crocodile" to span across two columns, and "Horse" and "Chicken" to span downwards over two rows. Fortunately, table headers and cells have the colspan and rowspan attributes, which allow us to do just those things. Both accept a unitless number value, which equals the number of rows or columns you want spanned. For example, colspan="2" makes a cell span two columns.

There is one last feature we'll tell you about in this article before we move on. HTML has a method of defining styling information for an entire column of data all in one place — the <col> and <colgroup> elements. These exist because it can be a bit annoying and inefficient having to specify styling on columns — you generally have to specify your styling information on every <td> or <th> in the column, or use a complex selector such as :nth-child().

<table>

<colgroup>

<col>

<col style="background-color: yellow">

</colgroup>

<tr>

<th>Data 1</th>

<th>Data 2</th>

</tr>

<tr>

<td>Calcutta</td>

<td>Pizza</td>

</tr>

<tr>

<td>Robots</td>

<td>Jazz</td>

</tr>

</table>

collapse

Adjacent cells have shared borders (the collapsed-border table rendering model).

separate

Adjacent cells have distinct borders (the separated-border table rendering model).

• Empty-cells: specify whether or not borders are shown fir empty cells. Value= show, hide, or inherit

* A table-layout value of fixed is generally a good idea to set on your table, as it makes the table behave a bit more predictably by default. Normally, table columns tend to be sized according to how much content they contain, which produces some strange results. With table-layout: fixed, you can size your columns according to the width of their headings, and then deal with their content as appropriate. This is why we've selected the four different headings with the thead th:nth-child(n) (:nth-child) selector ("Select the nth child that is a <th> element in a sequence, inside a <thead> element") and given them set percentage widths. The entire column width follows the width of its heading, making for a nice way to size your table columns. Chris Coyier discusses this technique in more detail in Fixed Table Layouts.
* There is nothing remarkable here, except for the caption-side property, which has been given a value of bottom. This causes the caption to be positioned on the bottom of the table,

**End notes on tables**

Media queries let you adapt your site or app depending on the presence or value of various device characteristics and parameters.

They are a key component of responsive design. For example, a media query can shrink the font size on small devices, increase the padding between paragraphs when a page is viewed in portrait mode, or bump up the size of buttons on touchscreens.

In CSS, use the @media at-rule to conditionally apply part of a style sheet based on the result of a media query. Use @import to conditionally apply an entire style sheet.

**CSS Information**

**CSS Properties**

* transform: rotate(0.8turn) The transform CSS property lets you rotate, scale, skew, or translate an element. It modifies the coordinate space of the CSS visual formatting model. <transform-function> = <matrix()> | <translate()> | <translateX()> | <translateY()> | <scale()> | <scaleX()> | <scaleY()> | <rotate()> | <skew()> | <skewX()> | <skewY()> | <matrix3d()> | <translate3d()> | <translateZ()> | <scale3d()> | <scaleZ()> | <rotate3d()> | <rotateX()> | <rotateY()> | <rotateZ()> | <perspective()>
* :last-child
* :only-child The :only-child CSS pseudo-class represents an element without any siblings. This is the same as :first-child:last-child or :nth-child(1):nth-last-child(1), but with a lower specificity.
* :invalid The :invalid CSS pseudo-class represents any <input> or other <form> element whose contents fail to validate. /\* Selects any invalid <input> \*/ input:invalid {
* background-color: pink;
* }
* This pseudo-class is useful for highlighting field errors for the user.
* :hover — mentioned above; this only applies if the user moves their pointer over an element, typically a link.
* :focus — only applies if the user focuses the element using keyboard controls.
* The ::first-line pseudo-element selector will do this for you reliably — if the number of words increases and decreases it will still only select the first line.
* You could use these to insert a string of text, such as in the live example below. Try changing the text value of the content property and see it change in the output. You could also change the ::before pseudo-element to ::after and see the text inserted at the end of the element instead of the beginning. .box::before {
* content: "This should show before the other content."
* }
* Box-sizing: border-box By default, browsers use the standard box model. If you want to turn on the alternative model for an element you do so by setting box-sizing: border-box on it. By doing this you are telling the browser to take the border box as the area defined by any size you set.
* Border: inset
* Displays a border that makes the element appear embedded. It is the opposite of outset. When applied to a table cell with border-collapse set to collapsed, this value behaves like groove.
* Border: outset
* Displays a border that makes the element appear embossed. It is the opposite of inset. When applied to a table cell with border-collapse set to collapsed, this value behaves like ridge.
* Controlling background-repeat
* The background-repeat property is used to control the tiling behavior of images. The available values are:
* no-repeat — stop the background from repeating altogether.
* repeat-x — repeat horizontally.
* repeat-y — repeat vertically.
* repeat — the default; repeat in both directions.
* Background: size background-size property, which can take length or percentage values, to size the image to fit inside the background.
* cover — the browser will make the image just large enough so that it completely covers the box area while still retaining its aspect ratio. In this case some of the image is likely to end up outside the box.
* contain — the browser will make the image the right size to fit inside the box. In this case you may end up with gaps on either side or on the top and bottom of the image, if the aspect ratio of the image is different to that of the box.
* The background-position property allows you to choose the position in which the background image appears on the box it is applied to. This uses a coordinate system in which the top-left-hand corner of the box is (0,0), and the box is positioned along the horizontal (x) and vertical (y) axes.
* A <position>. A position defines an x/y coordinate, to place an item relative to the edges of an element's box. It can be defined using one to four values. If two non-keyword values are used, the first value represents the horizontal position and the second represents the vertical position. If only one value is specified, the second value is assumed to be center. If three or four values are used, the length-percentage values are offsets for the preceding keyword value(s).
* background-position is a shorthand for background-position-x and background-position-y, which allow you to set the different axis position values individually.
* The <gradient> CSS data type is a special type of <image> that consists of a progressive transition between two or more colors.
* Radial gradients transition colors progressively from a center point (origin). They are generated with the radial-gradient() function.
* Repeating gradients duplicate a gradient as much as necessary to fill a given area. They are generated with the repeating-linear-gradient() and repeating-radial-gradient() functions.
* Another option we have available for backgrounds is specifying how they scroll when the content scrolls. This is controlled using the background-attachment property, which can take the following values:
* scroll: Causes the element's background to scroll when the page is scrolled. If the element content is scrolled, the background does not move. In effect, the background is fixed to the same position on the page, so it scrolls as the page scrolls.
* fixed: Causes an element's background to be fixed to the viewport, so that it doesn't scroll when the page or element content is scrolled. It will always remain in the same position on the screen.
* local: This value was added later on (it is only supported in Internet Explorer 9+, whereas the others are supported in IE4+) because the scroll value is rather confusing and doesn't really do what you want in many cases. The local value fixes the background to the element it is set on, so when you scroll the element, the background scrolls with it.
* Collapse: Adjacent cells have shared borders (the collapsed-border table rendering model).
* Separate: Adjacent cells have distinct borders (the separated-border table rendering model).
* The :nth-child() CSS pseudo-class matches elements based on their position in a group of siblings.
* The nth-child pseudo-class is specified with a single argument that describes a pattern for matching element indices in a list of sibblings. Element indices are 1-based.
* Keyword values
* odd
* Represents elements whose numeric position in a series of siblings is odd: 1, 3, 5, etc.
* even
* Represents elements whose numeric position in a series of siblings is even: 2, 4, 6, etc.
* Font-style: either italic, normal, or oblique
* Text-tranform: either lowercase, uppercase, or capitalize
* Text-decoration: none, underline, overline, line-through, or blink
* Line-height: controls the height of the line of text ( a good size is 1.4 to 1.5 em) (em is better than pixels in this because the size of the gap will be relative to the size of the text
* Letter-spacing:
* word-spacing: both should be measured in ems
* text-align: left, right, center, or justify (This indicates that every line in a paragraph, except the last line, should be set to take up the full width of the containing box.
* Vertical-align: it is used for <td> <th> <image>
  + The values it can take are
    - Baseline
    - Sub
    - Super
    - Top
    - Text-top
    - Middle
    - Bottom
    - Text-bottom
* Text-indent: allows you to indent the first line of text within the element
* Text-shadow:
* Min-width: minimum width the box can be
* Max-width:
* Min-height
* Max-height
* Overflow: - property that tells the browser what to do if the content within a box is larger than the box itself it can have two values Hidden or Scroll.
* Display: inline, block, inline-block, and none
* Visibility: allows you to hide boxes from users but it leaves a space where the element would have been it had two values hidden, visible
* Border-image: applies an image to the border of any box.
* Border-radius: allows you to create rounded corners
* List-style-type: property allows you to control the shape or style of a bullet point, None, disc, circle, and square
* List-style-image: property allows you to put an image as the bullet
* list-style-image: url(“images/star.png”)
* list-style-position: inside or outside.
* Empty-cells: specify whether or not borders are shown fir empty cells. Value= show, hide, or inherit
* Border-collapse: collapse and separate
* Border-spacing: to separate spacing in between cells-
* Cursor: values= auto, crosshair, default, pointer, move, text, wait, help, url (“cursor.gif”)
* Position: static – this indicates that elements are in normal flow
* Position: relative – relative positioning moves an element in relation to where it would have been in normal flow.
* Position: absolute – when the position property is given a value of absolute, the box is taken out of normal flow and no longer affects the position of other elements on the page.(they act like it is not there).
* Position:fixed – is a type of absolute positioning that requires the position property to have a value of fixed. It positions the element in relation to the browser window. Therefore, when a user scrolls down the page, it stays in the exact same place.
* Z-index: When you use relative, fixed, or absolute positioning, boxes can overlap, the elements that appear later in the html code sit on top of those that are earlier in the page. If you want to control which element sits in top, you can use the z-index property. Its value is a number, and the higher the number the closer that element is to the front, 10.
* Float: (right or left) The float property allows you to take an element in normal flow and place it as far to the right or left of the containing element as possible. Anything else that sits inside the containing element will flow around the element that is floated. When you first use the float property, you should also use the width property to indicate how wide the floated element should be. A lot of layouts place boxes next to each other. The float property is commonly used to achieve this.
* Clear: left or right – the clear property allows you to say that no element (within the same containing element) should touch the left or right-hand sides of a box.
* If a containing element only contains floated elements, some browsers will treat it as if it is zero pixels tall. The solution for this is to set the element to overflow: auto, and width:100%
* Background-position: - this property can be used to specify where in the browser window the background image should be placed.
* Background-attachment: specifies whether a background image should stay in one position or move as the user scrolls up and down the page.(values= fixed, or scroll)
* Flex-direction (values= row, column, row-reverse, column-reverse) Flexbox provides a property called flex-direction that specifies what direction the main axis runs in (what direction the flexbox children are laid out in) — by default this is set to row, which causes them to be laid out in a row in the direction your browser's default language works in (left to right, in the case of an English browser).
* Flex-wrap: wrap
* Flex: 200px
* Flex-flow: At this point it is worth noting that a shorthand exists for flex-direction and flex-wrap — flex-flow. So for example, you can replace
* align-items controls where the flex items sit on the cross axis.
* By default, the value is stretch, which stretches all flex items to fill the parent in the direction of the cross axis. If the parent hasn't got a fixed width in the cross axis direction, then all flex items will become as long as the longest flex items. This is how our first example got equal height columns by default.
* The center value that we used in our above code causes the items to maintain their intrinsic dimensions, but be centered along the cross axis. This is why our current example's buttons are centered vertically.
* You can also have values like flex-start and flex-end, which will align all items at the start and end of the cross axis respectively. See align-items for the full details.
* You can override the align-items behavior for individual flex items by applying the align-self property to them. For example, try adding the following to your CSS:
* button:first-child {
* align-self: flex-end;
* }
* Have a look at what effect this has, and remove it again when you've finished.
* justify-content controls where the flex items sit on the main axis.
* The default value is flex-start, which makes all the items sit at the start of the main axis.
* You can use flex-end to make them sit at the end.
* center is also a value for justify-content, and will make the flex items sit in the center of the main axis.
* The value we've used above, space-around, is useful — it distributes all the items evenly along the main axis, with a bit of space left at either end.
* There is another value, space-between, which is very similar to space-around except that it doesn't leave any space at either end.
* Order: By default, all flex items have an order value of 0.
* Flex items with higher order values set on them will appear later in the display order than items with lower order values.
* Flex items with the same order value will appear in their source order. So if you have four items with order values of 2, 1, 1, and 0 set on them respectively, their display order would be 4th, 2nd, 3rd, then 1st.
* Flex: 1 auto Finally, we set some sizing on the button, but more interestingly we give it a flex value of 1 auto. This has a very interesting effect, which you'll see if you try resizing your browser window width. The buttons will take up as much space as they can and sit as many on the same line as they can, but when they can no longer fit comfortably on the same line, they'll drop down to create new lines.
* Writing-mode: A writing mode in CSS refers to whether the text is running horizontally or vertically. The [writing-mode](https://developer.mozilla.org/en-US/docs/Web/CSS/writing-mode) property lets us switch from one writing mode to another.
* The three possible values for the writing-mode property are:

horizontal-tb: Top-to-bottom block flow direction. Sentences run horizontally.

vertical-rl: Right-to-left block flow direction. Sentences run vertically.

vertical-lr: Left-to-right block flow direction. Sentences run vertically.

Overflow: The  [overflow](https://developer.mozilla.org/en-US/docs/Web/CSS/overflow) property is how you take control of an element's overflow and tell the browser how you want it to behave. The default value of overflow is visible, which is why by default we can see our content when it overflows.

If you want to crop the content when it overflows you can set overflow: hidden on your box. This will do exactly what it says — hide the overflow. This may well cause things to vanish so you should only ever do this if hiding content is not going to cause a problem.

* Overflow: scroll;
* In the above example we only need to scroll on the y axis, however we get scrollbars in both axes. You could instead use the overflow-y property, setting overflow-y: scroll to only scroll on the y axis.
* Overflow: auto; If you only want scrollbars to appear if there is more content than can fit in the box, then use overflow: auto. In this case it is left up to the browser to decide whether to display scrollbars. Desktop browsers will typically only do so once there is enough content to cause overflow.
* vw
* 1% of the viewport's width.
* vh
* 1% of the viewport's height.
* The viewport — which is the visible area of your page in the browser you are using to view a site — also has a size. In CSS we have units which relate to the size of the viewport — the vw unit for viewport width, and vh for viewport height. Using these units you can size something relative to the viewport of the user.
* 1vh is equal to 1% of the viewport height, and 1vw is equal to 1% of the viewport width. You can use these units to size boxes, but also text. In the example below we have a box which is sized as 20vh and 20vw. The box contains a letter A, which has been given a font-size of 10vh.
* Table-layout: fixed A table-layout value of fixed is generally a good idea to set on your table, as it makes the table behave a bit more predictably by default. Normally, table columns tend to be sized according to how much content they contain, which produces some strange results. With table-layout: fixed, you can size your columns according to the width of their headings, and then deal with their content as appropriate. This is why we've selected the four different headings with the thead th:nth-child(n) (:nth-child) selector ("Select the nth child that is a <th> element in a sequence, inside a <thead> element") and given them set percentage widths. The entire column width follows the width of its heading, making for a nice way to size your table columns. Chris Coyier discusses this technique in more detail in Fixed Table Layouts.
* Caption-side: bottom; There is nothing remarkable here, except for the caption-side property, which has been given a value of bottom. This causes the caption to be positioned on the bottom of the table,
* Display: flow-root; The modern way of solving this problem is to use the value flow-root of the display property. This exists only to create a BFC without using hacks — there will be no unintended consequences when you use it. Remove overflow: auto from your .wrapper rule and add display: flow-root. Assuming you have a [supporting browser](https://developer.mozilla.org/en-US/docs/Web/CSS/display#Browser_compatibility), the box will clear.
* Column-count:
* Column-width: The [<div>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/div) with a class of container will become our multicol container. We switch on multicol by using one of two properties [column-count](https://developer.mozilla.org/en-US/docs/Web/CSS/column-count) or [column-width](https://developer.mozilla.org/en-US/docs/Web/CSS/column-width). The column-count property will create as many columns as the value you give it, so if you add the following CSS to your stylesheet and reload the page, you will get three columns:
* Column-gap:

Column-rule: The columns created by multicol cannot be styled individually. There is no way to make one column bigger than other columns, or to change the background or text color of a single column. You have two opportunities to change the way that columns display:

* Changing the size of the gap between columns using the [column-gap](https://developer.mozilla.org/en-US/docs/Web/CSS/column-gap).
* Adding a rule between columns with [column-rule](https://developer.mozilla.org/en-US/docs/Web/CSS/column-rule).
* Column-rule: You can play around with different values — the property accepts any length unit. Now add a rule between the columns, with column-rule. In a similar way to the [border](https://developer.mozilla.org/en-US/docs/Web/CSS/border) property that you encountered in previous lessons, column-rule is a shorthand for [column-rule-color](https://developer.mozilla.org/en-US/docs/Web/CSS/column-rule-color), [column-rule-style](https://developer.mozilla.org/en-US/docs/Web/CSS/column-rule-style), and [column-rule-width](https://developer.mozilla.org/en-US/docs/Web/CSS/column-rule-width), and accepts the same values as border.

Break-inside: avoid; To control this behavior we can use properties from the [CSS Fragmentation](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Fragmentation) specification. This specification gives us properties to control breaking of content in multicol and in paged media. For example, add the property [break-inside](https://developer.mozilla.org/en-US/docs/Web/CSS/break-inside) with a value of avoid to the rules for .card. This is the container of the heading and text, and therefore we do not want to fragment this box.

At the present time it is also worth adding the older property page-break-inside: avoid for best browser support.

**End Properties**

**Pseudo-elements**

* :first-letter
* First-line

**End Pseudo-elements**

**Pseudo-Classes**

* :link – this allows you to set styles for links that have not been clicked on
* :visited – this allows you to set styles for links that have not been clicked on
* :hover – this is applied when a user hover over an element with a mouse

Active: - this is applied when an element is being activated by a user; for example, when a button or link is being clicked

**End Pseudo-Classes**

**Normal Information**

CSS provides four special universal property values for controlling inheritance. Every CSS property accepts these values.

inherit

Sets the property value applied to a selected element to be the same as that of its parent element. Effectively, this "turns on inheritance".

initial

Sets the property value applied to a selected element to be the same as the value set for that property on that element in the browser's default style sheet. If no value is set by the browser's default style sheet and the property is naturally inherited, then the property value is set to inherit instead.

unset

Resets the property to its natural value, which means that if the property is naturally inherited it acts like inherit, otherwise it acts like initial.

**Attribute Selectors**

* As you know from your study of HTML, elements can have attributes that give further detail about the element being marked up. In CSS you can use attribute selectors to target elements with certain attributes. This lesson will show you how to use these very useful selectors.

**Pseudo-Classes**

* A pseudo-class is a selector that selects elements that are in a specific state, e.g. they are the first element of their type, or they are being hovered over by the mouse pointer. They tend to act as if you had applied a class to some part of your document, often helping you cut down on excess classes in your markup, and giving you more flexible, maintainable code.
* Pseudo-classes are keywords that start with a colon:
* :pseudo-class-name
* Instead of adding the class, we could use the :first-child pseudo-class selector — this will always target the first child element in the article, and we will no longer need to edit the HTML
* **User-action pseudo classes**
* Some pseudo-classes only apply when the user interacts with the document in some way. These user-action pseudo-classes, sometimes referred to as dynamic pseudo classes, act as if a class had been added to the element when the user interacts with it. Examples include:
* :hover — mentioned above; this only applies if the user moves their pointer over an element, typically a link.
* :focus — only applies if the user focuses the element using keyboard controls.
* Pseudo-elements behave in a similar way, however they act as if you had added a whole new HTML element into the markup, rather than applying a class to existing elements. Pseudo-elements start with a double colon ::.
* ::p to select only <p> elements that are direct children of <article> elements:
* article > pseudo-element-name
* The adjacent sibling selector (+) is used to select something if it is right next to another element at the same level of the hierarchy. For example, to select all <img> elements that come right after <p> elements:
* If you want to select siblings of an element even if they are not directly adjacent, then you can use the general sibling combinator (~). To select all <img> elements that come anywhere after <p> elements, we'd do this:

**Boxes**

* In CSS we broadly have two types of boxes — block boxes and inline boxes. These characteristics refer to how the box behaves in terms of page flow, and in relation to other boxes on the page:
* If a box is defined as a block, it will behave in the following ways:
* The box will extend in the inline direction to fill the space available in its container. In most cases this means that the box will become as wide as its container, filling up 100% of the space available.
* The box will break onto a new line.
* The width and height properties are respected.
* Padding, margin and border will cause other elements to be pushed away from the box
* The type of box applied to an element is defined by display property values such as block and inline, and relates to the outer value of display.
* If a box is defined as a block, it will behave in the following ways:
* The box will extend in the inline direction to fill the space available in its container. In most cases this means that the box will become as wide as its container, filling up 100% of the space available.
* The box will break onto a new line.
* The width and height properties are respected.
* Padding, margin and border will cause other elements to be pushed away from the box
* Unless we decide to change the display type to inline, elements such as headings (e.g. <h1>) and <p> all use block as their outer display type by default.
* If a box has an outer display type of inline, then:
* The box will not break onto a new line.
* The width and height properties will not apply.
* Padding, margin and borders will apply but will not cause other inline boxes to move away from the box.
* The <a> element, use
* Examples of different display types
* Let's move on and have a look at some examples. Below we have three different HTML elements, all of which have an outer display type of block. The first is a paragraph, which has a border added in CSS. The browser renders this as a block box, so the paragraph begins on a new line, and expands to the full width available to it.
* The second is a list, which is laid out using display: flex. This establishes flex layout for the items inside the container, however, the list itself is a block box and — like the paragraph — expands to the full container width and breaks onto a new line.
* Below this, we have a block-level paragraph, inside which are two <span> elements. These elements would normally be inline, however, one of the elements has a class of block, and we have set it to display: block.
* What is the CSS box model?
* The full CSS box model applies to block boxes, inline boxes only use some of the behavior defined in the box model. The model defines how the different parts of a box — margin, border, padding, and content — work together to create a box that you can see on the page. To add some additional complexity, there is a standard and an alternate box model.
* Parts of a box
* Making up a block box in CSS we have the:
* Content box: The area where your content is displayed, which can be sized using properties like width and height.
* Padding box: The padding sits around the content as white space; its size can be controlled using padding and related properties.
* Border box: The border box wraps the content and any padding. Its size and style can be controlled using border and related properties.
* Margin box: The margin is the outermost layer, wrapping the content, padding and border as whitespace between this box and other elements. Its size can be controlled using margin and related properties.
* By default, browsers use the standard box model. If you want to turn on the alternative model for an element you do so by setting box-sizing: border-box on it. By doing this you are telling the browser to take the border box as the area defined by any size you set.
* Margin
* The margin is an invisible space around your box. It pushes other elements away from the box. Margins can have positive or negative values. Setting a negative margin on one side of your box can cause it to overlap other things on the page. Whether you are using the standard or alternative box model, the margin is always added after the size of the visible box has been calculated.
* Margin collapsing
* A key thing to understand about margins is the concept of margin collapsing. If you have two elements whose margins touch, and both margins are positive, those margins will combine to become one margin, which is the size of the largest individual margin. If one or both margins are negative, the amount of negative value will subtract from the total.
* In the example below, we have two paragraphs. The top paragraph has a margin-bottom of 50 pixels. The second paragraph has a margin-top of 30 pixels. The margins have collapsed together so the actual margin between the boxes is 50 pixels and not the total of the two margins.
* The box model and inline boxes
* All of the above applies fully to block boxes. Some of the properties can apply to inline boxes too, such as those created by a <span> element.
* In the example below, we have a <span> inside a paragraph and have applied a width, height, margin, border, and padding to it. You can see that the width and height are ignored. The margin, padding, and border are respected but they do not change the relationship of other content to our inline box and so the padding and border overlaps other words in the paragraph.
* Using display: inline-block
* There is a special value of display, which provides a middle ground between inline and block. This is useful for situations where you do not want an item to break onto a new line, but do want it to respect width and height and avoid the overlapping seen above.
* An element with display: inline-block does a subset of the block things we already know about:
* The width and height properties are respected.

padding, margin, and border will cause other elements to be pushed away from the box.

* It does not, however, break onto a new line, and will only become larger than its content if you explicitly add width and height properties.

**End Boxes**

**FlexBox**

* Flexbox provides a property called flex-direction that specifies what direction the main axis runs in (what direction the flexbox children are laid out in) — by default this is set to row, which causes them to be laid out in a row in the direction your browser's default language works in (left to right, in the case of an English browser).
* Here we see that the children are indeed breaking out of their container. One way in which you can fix this is to add the following declaration to your <section> rule:
* flex-wrap: wrap;
* Also, add the following declaration to your <article> rule:
* flex: 200px;
* Try this now; you'll see that the layout looks much better with this included:
* At this point it is worth noting that a shorthand exists for flex-direction and flex-wrap — flex-flow. So for example, you can replace
* This is a unitless proportion value that dictates how much of the available space along the main axis each flex item will take up. In this case, we are giving each <article> element a value of 1, which means they will all take up an equal amount of the spare space left after things like padding and margin have been set. It is a proportion, meaning that giving each flex item a value of 400000 would have exactly the same effect.
* You can also specify a minimum size value inside the flex value. Try updating your existing article rules like so:
* article {
* flex: 1 200px;
* }
* article:nth-of-type(3) {
* flex: 2 200px;
* }
* This basically states "Each flex item will first be given 200px of the available space. After that, the rest of the available space will be shared out according to the proportion units." Try refreshing and you'll see a difference in how the space is shared out.
* **align-items controls where the flex items sit on the cross axis.**
* By default, the value is stretch, which stretches all flex items to fill the parent in the direction of the cross axis. If the parent hasn't got a fixed width in the cross axis direction, then all flex items will become as long as the longest flex items. This is how our first example got equal height columns by default.
* The center value that we used in our above code causes the items to maintain their intrinsic dimensions, but be centered along the cross axis. This is why our current example's buttons are centered vertically.
* You can also have values like flex-start and flex-end, which will align all items at the start and end of the cross axis respectively. See align-items for the full details.
* You can override the align-items behavior for individual flex items by applying the align-self property to them. For example, try adding the following to your CSS:
* button:first-child {
* align-self: flex-end;
* }
* Have a look at what effect this has, and remove it again when you've finished.
* justify-content controls where the flex items sit on the main axis.
* The default value is flex-start, which makes all the items sit at the start of the main axis.
* You can use flex-end to make them sit at the end.
* center is also a value for justify-content, and will make the flex items sit in the center of the main axis.
* The value we've used above, space-around, is useful — it distributes all the items evenly along the main axis, with a bit of space left at either end.
* There is another value, space-between, which is very similar to space-around except that it doesn't leave any space at either end.
* By default, all flex items have an order value of 0.
* Flex items with higher order values set on them will appear later in the display order than items with lower order values.
* Flex items with the same order value will appear in their source order. So if you have four items with order values of 2, 1, 1, and 0 set on them respectively, their display order would be 4th, 2nd, 3rd, then 1st.
* It is possible to create some pretty complex layouts with flexbox. It is perfectly ok to set a flex item to also be a flex container, so that its children are also laid out like flexible boxes. Have a look at complex-flexbox.html (see it live also).
* Finally, we set some sizing on the button, but more interestingly we give it a flex value of 1 auto. This has a very interesting effect, which you'll see if you try resizing your browser window width. The buttons will take up as much space as they can and sit as many on the same line as they can, but when they can no longer fit comfortably on the same line, they'll drop down to create new lines.

**End FlexBox**

**Grids**

* Unlike flexbox, the items will not immediately look any different. Declaring display: grid gives you a one column grid, so your items will continue to display one below the other as they do in normal flow.
* To see something that looks more grid-like, we will need to add some columns to the grid. Let's add three 200-pixel columns here. You can use any length unit, or percentages to create these column tracks.
* .container {
* display: grid;
* grid-template-columns: 200px 200px 200px;
* }
* In addition to creating grids using lengths and percentages, we can use the fr unit to flexibly size grid rows and columns. This unit represents one fraction of the available space in the grid container.
* To create gaps between tracks we use the properties grid-column-gap for gaps between columns, grid-row-gap for gaps between rows, and grid-gap to set both at once.
* We have only specified column tracks so far, and yet rows are being created to hold our content. This is an example of the explicit versus the implicit grid. The explicit grid is the one that you create using grid-template-columns or grid-template-rows. The implicit grid is created when content is placed outside of that grid — such as into our rows. The explicit and implicit grids are analogous to the main and cross flexbox axes.
* By default, tracks created in the implicit grid are auto sized, which in general means that they are large enough to fit their content. If you wish to give implicit grid tracks a size you can use the grid-auto-rows and grid-auto-columns properties. If you add grid-auto-rows with a value of 100px to your CSS, you will see that those created rows are now 100 pixels tall.
* **Our 100-pixel tall tracks won’t be very useful if we add content into those tracks that is taller than 100 pixels, in which case it would cause an overflow. It might be better to have tracks that are at least 100 pixels tall and can still expand if more content gets into them. A fairly basic fact about the web is that you never really know how tall something is going to be; additional content or larger font sizes can cause problems with designs that attempt to be pixel perfect in every dimension.**
* **The minmax function lets us set a minimum and maximum size for a track, for example minmax(100px, auto). The minimum size is 100 pixels, but the maximum is auto, which will expand to fit the content. Try changing grid-auto-rows to use a minmax value:**
* We can combine some of the things we have learned about track listing, repeat notation and minmax() to create a useful pattern. Sometimes it is helpful to be able to ask grid to create as many columns as will fit into the container. We do this by setting the value of grid-template-columns using repeat() notation, but instead of passing in a number, pass in the keyword auto-fill. For the second parameter of the function we use minmax(), with a minimum value equal to the minimum track size that we would like to have, and a maximum of 1fr.
* An alternative way to place items on your grid is to use the [grid-template-areas](https://developer.mozilla.org/en-US/docs/Web/CSS/grid-template-areas) property and giving the various elements of your design a name.
* Remove the line-based positioning from the last example (or re-download the file to have a fresh starting point), and add the following CSS.

**End Grids**

**Columns Layout**

* The <div> with a class of container will become our multicol container. We switch on multicol by using one of two properties column-count or column-width. The column-count property will create as many columns as the value you give it, so if you add the following CSS to your stylesheet and reload the page, you will get three columns:
* You can play around with different values — the property accepts any length unit. Now add a rule between the columns, with column-rule. In a similar way to the [border](https://developer.mozilla.org/en-US/docs/Web/CSS/border) property that you encountered in previous lessons, column-rule is a shorthand for [column-rule-color](https://developer.mozilla.org/en-US/docs/Web/CSS/column-rule-color), [column-rule-style](https://developer.mozilla.org/en-US/docs/Web/CSS/column-rule-style), and [column-rule-width](https://developer.mozilla.org/en-US/docs/Web/CSS/column-rule-width), and accepts the same values as border.
* To control this behavior we can use properties from the [CSS Fragmentation](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Fragmentation) specification. This specification gives us properties to control breaking of content in multicol and in paged media. For example, add the property [break-inside](https://developer.mozilla.org/en-US/docs/Web/CSS/break-inside) with a value of avoid to the rules for .card. This is the container of the heading and text, and therefore we do not want to fragment this box.
* At the present time it is also worth adding the older property page-break-inside: avoid for best browser support.

**End Columns Layout**

**Centering Content**

* If you want to center a box on the page (or center it inside the element that it sits in), you can set the left-margin and right-margin to auto
* In order to center a box on the page, you need to set a width for the box (otherwise it will take up the full width of the page.)

**End Centering Content**

**Background and Images**

* Controlling background-repeat
* The background-repeat property is used to control the tiling behavior of images. The available values are:
* no-repeat — stop the background from repeating altogether.
* repeat-x — repeat horizontally.
* repeat-y — repeat vertically.
* repeat — the default; repeat in both directions.
* Sizing the background image
* In the example above, we have a large image that has ended up being cropped as it is larger than the element it is a background of. In this case we could use the background-size property, which can take length or percentage values, to size the image to fit inside the background.
* You can also use keywords:
* cover — the browser will make the image just large enough so that it completely covers the box area while still retaining its aspect ratio. In this case some of the image is likely to end up outside the box.
* contain — the browser will make the image the right size to fit inside the box. In this case you may end up with gaps on either side or on the top and bottom of the image, if the aspect ratio of the image is different to that of the box.
* The background-position property allows you to choose the position in which the background image appears on the box it is applied to. This uses a coordinate system in which the top-left-hand corner of the box is (0,0), and the box is positioned along the horizontal (x) and vertical (y) axes.
* A <position>. A position defines an x/y coordinate, to place an item relative to the edges of an element's box. It can be defined using one to four values. If two non-keyword values are used, the first value represents the horizontal position and the second represents the vertical position. If only one value is specified, the second value is assumed to be center. If three or four values are used, the length-percentage values are offsets for the preceding keyword value(s).
* background-position is a shorthand for background-position-x and background-position-y, which allow you to set the different axis position values individually.
* The <gradient> CSS data type is a special type of <image> that consists of a progressive transition between two or more colors.
* Radial gradients transition colors progressively from a center point (origin). They are generated with the radial-gradient() function.
* Repeating gradients duplicate a gradient as much as necessary to fill a given area. They are generated with the repeating-linear-gradient() and repeating-radial-gradient() functions.
* Another option we have available for backgrounds is specifying how they scroll when the content scrolls. This is controlled using the background-attachment property, which can take the following values:
* scroll: Causes the element's background to scroll when the page is scrolled. If the element content is scrolled, the background does not move. In effect, the background is fixed to the same position on the page, so it scrolls as the page scrolls.
* fixed: Causes an element's background to be fixed to the viewport, so that it doesn't scroll when the page or element content is scrolled. It will always remain in the same position on the screen.
* local: This value was added later on (it is only supported in Internet Explorer 9+, whereas the others are supported in IE4+) because the scroll value is rather confusing and doesn't really do what you want in many cases. The local value fixes the background to the element it is set on, so when you scroll the element, the background scrolls with it.
* Background-position: - this property can be used to specify where in the browser window the background image should be placed.
* Background-attachment: specifies whether a background image should stay in one position or move as the user scrolls up and down the page.(values= fixed, or scroll)
* Using CSS it is possible to create a link or button that changes to a second style when a user moves their mouse over it (known as a **rollover**) and a third style when they click on it.

**End Background and Images**

**Developer Tools**

* How do you pull it up? Three ways:
* Keyboard: Ctrl + Shift + I, except
* Internet Explorer and Edge: F12
* macOS: ⌘ + ⌥ + I
* Menu bar:
* Firefox: Menu ➤ Web Developer ➤ Toggle Tools, or Tools ➤ Web Developer ➤ Toggle Tools
* Chrome: More tools ➤ Developer tools
* Safari: Develop ➤ Show Web Inspector. If you can't see the Develop menu, go to Safari ➤ Preferences ➤ Advanced, and check the Show Develop menu in menu bar checkbox.
* Opera: Developer ➤ Developer tools
* Context menu: Press-and-hold/right-click an item on a webpage (Ctrl-click on the Mac), and choose Inspect Element from the context menu that appears. (An added bonus: this method straight-away highlights the code of the element you right-clicked.)

**End Developer Tools**

**Text**

* @font-face allows you to use a font, even if it is not installed on the computer of the person browsing, by allowing you to specify a path to a copy of the font, which will be downloaded if it is not on the user’s machine.
* You can find free fonts using
  + [www.fontsquirrel.com](http://www.fontsquirrel.com)
  + [www.fontex.org](http://www.fontex.org)
  + [www.openfontlibrary.org](http://www.openfontlibrary.org)
* Fonts for a one time fee
  + [www.typekit.com](http://www.typekit.com)
  + [www.kernest.com](http://www.kernest.com)
  + [www.fontspring.com](http://www.fontspring.com)
* Different browsers support different formats for fonts, so you will need to supply the font in several variations to reach all browsers. If you do not have all of these formats for your font, you can upload the font to a website called Font Squirrel where they will convert it for you.
  + [www.fontsquirrel.com/fontface/generator](http://www.fontsquirrel.com/fontface/generator)

**End Text**

**Layout**

* Multiple columns is achieved by width, float, and margin. Width controls the width of each column, float positions the columns next to each other, and margin creates a gap between the columns
* 960 pixel grid.
  + Every margin that doesn’t border the outside has a value of 20pixels shared.
  + Each outside margin on the sides has a margin of 10 so total of 20.
  + It has a total of 12 columns each column 60px
* There is also the 960.GS Framework

**End Layout**

**Styling Text**

* serif
* Fonts that have serifs (the flourishes and other small details you see at the ends of the strokes in some typefaces)
* My big red elephant
* sans-serif
* Fonts that don't have serifs.
* My big red elephant
* monospace
* Fonts where every character has the same width, typically used in code listings.
* My big red elephant
* cursive
* Fonts that are intended to emulate handwriting, with flowing, connected strokes.
* My big red elephant
* fantasy
* Fonts that are intended to be decorative.
* My big red elephant
* Since you can't guarantee the availability of the fonts you want to use on your webpages (even a web font could fail for some reason), you can supply a font stack so that the browser has multiple fonts it can choose from. This simply involves a font-family value consisting of multiple font names separated by commas, e.g.
* ems: 1 em is equal to the font size set on the parent element of the current element we are styling (more specifically, the width of a capital letter M contained inside the parent element.) This can become tricky to work out if you have a lot of nested elements with different font sizes set, but it is doable, as you'll see below. Why bother? It is quite natural once you get used to it, and you can use em to size everything, not just text. You can have an entire website sized using em, which makes maintenance easy.
* rems: These work just like em, except that 1 rem is equal to the font size set on the root element of the document (i.e. [<html>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/html)), not the parent element. This makes doing the maths to work out your font sizes much easier, although if you want to support really old browsers, you might struggle — rem is not supported in Internet Explorer 8 and below.
* ems: 1 em is equal to the font size set on the parent element of the current element we are styling (more specifically, the width of a capital letter M contained inside the parent element.) This can become tricky to work out if you have a lot of nested elements with different font sizes set, but it is doable, as you'll see below. Why bother? It is quite natural once you get used to it, and you can use em to size everything, not just text. You can have an entire website sized using em, which makes maintenance easy.
* rems: These work just like em, except that 1 rem is equal to the font size set on the root element of the document (i.e. <html>), not the parent element. This makes doing the maths to work out your font sizes much easier, although if you want to support really old browsers, you might struggle — rem is not supported in Internet Explorer 8 and below.
* ems: 1 em is equal to the font size set on the parent element of the current element we are styling (more specifically, the width of a capital letter M contained inside the parent element.) This can become tricky to work out if you have a lot of nested elements with different font sizes set, but it is doable, as you'll see below. Why bother? It is quite natural once you get used to it, and you can use em to size everything, not just text. You can have an entire website sized using em, which makes maintenance easy.
* rems: These work just like em, except that 1 rem is equal to the font size set on the root element of the document (i.e. <html>), not the parent element. This makes doing the maths to work out your font sizes much easier, although if you want to support really old browsers, you might struggle — rem is not supported in Internet Explorer 8 and below.
* ems: 1 em is equal to the font size set on the parent element of the current element we are styling (more specifically, the width of a capital letter M contained inside the parent element.) This can become tricky to work out if you have a lot of nested elements with different font sizes set, but it is doable, as you'll see below. Why bother? It is quite natural once you get used to it, and you can use em to size everything, not just text. You can have an entire website sized using em, which makes maintenance easy.
* rems: These work just like em, except that 1 rem is equal to the font size set on the root element of the document (i.e. <html>), not the parent element. This makes doing the maths to work out your font sizes much easier, although if you want to support really old browsers, you might struggle — rem is not supported in Internet Explorer 8 and below.
* The [start](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/ol#attr-start) attribute allows you to start the list counting from a number other than 1. The following example <ul start=”4”>
* The [value](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/ol#attr-value) attribute allows you to set your list items to specific numerical values. The following example:
* The [reversed](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/ol#attr-reversed) attribute will start the list counting down instead of up. The following example:
* A final word — how did we select just external links? Well, if you are writing your HTML links properly, you should only be using absolute URLs for external links — it is more efficient to use relative links to link to other parts of your own site. The text "http" should therefore only appear in external links, and we can select this with an attribute selector: a[href\*="http"] selects <a> elements, but only if they have an href attribute with a value that contains "http" somewhere inside it.
* But there is an alternative, which works very well, right back to IE version 6. Web fonts are a CSS feature that allows you to specify font files to be downloaded along with your website as it is accessed, meaning that any browser that supports web fonts can have exactly the fonts you specify available to it. Amazing! The syntax required looks something like this:
* First of all, you have a [@font-face](https://developer.mozilla.org/en-US/docs/Web/CSS/@font-face) block at the start of the CSS, which specifies the font file(s) to download:

**End Styling Text**

**Responsive Design**

**Media Queries**

* A common approach when using Media Queries is to create a simple single-column layout for narrow-screen devices (e.g mobile phones), then check for larger screens and implement a multiple-column layout when you know that you have enough screen width to handle it. This is often described as **mobile first** design.

The simplest media query syntax looks like this:

@media *media-type* and (*media-feature-rule*) {

/\* CSS rules go here \*/

}

It consists of:

* A media type, which tells the browser what kind of media this code is for (e.g. print, or screen).
* A media expression, which is a rule, or test that must be passed for the contained CSS to be applied.
* A set of CSS rules that will be applied if the test passes and the media type is correct.

Media types

The possible types of media you can specify are:

* all
* print
* screen
* speech

The following media query will only set the body to 12pt if the page is printed. It will not apply when the page is loaded in a browser.

* One well-supported media feature is orientation, which allows us to test for portrait or landscape mode. To change the body text color if the device is in landscape orientation, use the following media query.

**End Media Queries**

**Note**: In the [<head>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/head) of the example linked above, you'll find the line <meta name="viewport" content="width=device-width">: this forces mobile browsers to adopt their real viewport width for loading web pages (some mobile browsers lie about their viewport width, and instead load pages at a larger viewport width then shrink the loaded page down, which is not very helpful for responsive images or design).

**End Responsive Design**

<https://guides.library.scranton.edu/researchbasics>