***HTML***

* **H**yper**T**ext **M**arkup **L**anguage
* A *markup* language is a computer language that defines the structure and presentation of raw text. Markup languages work by surrounding raw text with information the computer can interpret, "marking it up" to be processed.
* In HTML, the computer can interpret raw text that is wrapped in HTML *elements*. These elements are often nested inside one another, with each containing information about the type and structure of the information to be displayed in the browser.
* *HyperText* is text displayed on a computer or device that provides access to other text through links, also known as “hyperlinks.” In fact, you probably clicked on many, many hyperlinks on your path to this Codecademy course!
* You can let web browsers know that you are using HTML by starting your document with a document type declaration; The declaration looks like this: <!DOCTYPE html>. This declaration is an instruction. It tells the browser what type of document to expect, along with what version of HTML is being used in the document.
* <!DOCTYPE html> must be the first line of code in all of your HTML documents. If you don't use the declaration, your HTML code will likely still work, however, it's risky. For now, the browser will correctly assume that the html in <!DOCTYPE html> is referring to HTML5, as it is the current standard.
* To create HTML structure and content, we must add opening and closing <html> *tags*, like so:
* <!DOCTYPE html> <html> </html>
* Anything between the opening <html> and closing </html> tags will be interpreted as HTML code. Without these tags, it's possible that browsers could incorrectly interpret your HTML code.
* The <head> element contains the metadata for a web page. Metadata is information about the page that isn't displayed directly on the web page.
* <title></tile> moze ici u head tag odmah -> pojavljuje se gore na tabu (recimo kao naslov taba)
* Before we can add content that a browser will display, we have to add a bodyto the HTML file. Only content inside the opening and closing body tags can be displayed to the screen
* The line break element <br /> (ili samo <br>) is one example of a self-closing tag. You can use it anywhere within your HTML code. The result is a line break in the browser. == \n
* <h1> do <h6> tagovi se koriste za naslove razlicitih velicina i fontova
* *Paragraphs* (<p>) simply contain a block of plain text.
* <div>s can contain any text or other HTML elements. They are primarily used to *divide* HTML documents into sections.
* <span>s contain short pieces of text or other HTML. They are primarily used to wrap small pieces of content that are on the same line as other content and do not break text into different sections.
* Tags provided by HTML exist to organize and describe the content of web pages. Two of these HTML tags are <em> and <strong>. They are used to signal that the text within them should be "emphasized" or "strong."
* The <em> tag will generally render as *italic* emphasis.
* The <strong> will generally render as **bold** emphasis.
* In HTML, you can use an unordered list tag (<ul>) to create a list of items in no particular order. An unordered list outlines individual list items with a bullet point.
* The <ul> element cannot hold raw text and cannot automatically format raw text into an unordered list of items. Individual list items must be added to the unordered list using the <li> tag. The <li> or list item tag is used to describe an item in a list.
* Ordered lists are like unordered lists, except that each list item is numbered. You can create the ordered list with the <ol> tag and then add individual list items to the list using <li> tags.
* The <img> tag allows you to add an image to a web page. This is another example of a self-closing tag.
* The <img> tag has a required attribute called src. The src attribute must be set to the image's source, or the location of the image. In this case, the value of src must be the uniform resource locator (URL) of the image. A URL is the web address or local address where a file is stored. Note that the end of the <img> tag has a forward slash /. Self-closing tags may include or omit the final slash — both will render properly.
* Attributes provide more information about an element's content. They live directly inside of the opening tag of an element. Attributes are made up of the following two parts:
* The *name* of the attribute
* The *value* of the attribute
* The alt attribute is applied specifically to the <img> element. The value of alt should be a description of the image.
* In addition to images, HTML also supports displaying videos. Like the <img> tag, the <video> tag requires a src attribute with a link to the video source. Unlike the <img> tag however, the <video> element requires an opening and a closing tag.
* In this example, the video source (src=) is "myVideo.mp4." The source must link to a video file, not to a video on another site. After the src attribute, the width and height attributes are used to set the size of the video displayed in the browser. The controls attribute instructs the browser to include basic video controls: pause, play and skip. The text, Video not supported, between the opening and closing video tags will only be displayed if the browser is unable to load the video.
* You can add links to a web page by adding an anchor element <a> and including the text of the link in between the opening and closing tags.
* The anchor element in the example above is incomplete without the href attribute. This attribute stands for hyperlink reference and is used to link to a file path, or the address to where a file is located (whether it is on your computer or another location).
* Thankfully, HTML allows you to turn nearly any element into a link by wrapping that element with an anchor element. With this technique, it's possible to turn images into links by simply wrapping the <img> element with an <a> element.
* Some of the tags we have used, such as <div>, are called non-semantic tags. This means that they do not describe the content that is inside of them. However, many tags are used to describe the content that they surround, which helps us modify and style our content later. These are called semantic tags and <nav> is one of them!
* The <table> element will contain all of the tabular data you plan on displaying.
* The first step in entering data into the table is to add rows using the table row element: <tr>.
* Rows aren't sufficient to add data to a table. Each cell element must also be defined. In HTML, you can add data using the table data element: <td>
* To add titles to rows and columns, you can use the table heading element: <th>.
* In the last exercise, the table's headings were kept inside of the table's body. When a table's body is sectioned off, however, it also makes sense to section off the table's headings using the <thead> element.

***CSS***

CSS can select HTML elements by their tag, class, and ID. CSS classes and IDs have different purposes, which can affect which one you use to style HTML elements.

CSS classes are meant to be reused over many elements. By writing CSS classes, you can style elements in a variety of ways by mixing classes on HTML elements.

For instance, imagine a page with two headlines. One headline needs to be bold and blue, and the other needs to be bold and green. Instead of writing separate CSS rules for each headline that repeat each other's code, it's better to write a .boldCSS rule, a .green CSS rule, and a .blue CSS rule. Then you can give one headline the bold green classes, and the other the bold blue classes.

While classes are meant to be used many times, an ID is meant to style only one element. As we'll learn in the next exercise, IDs override the styles of tags and classes. Since IDs override class and tag styles, they should be used sparingly and only on elements that need to always appear the same.

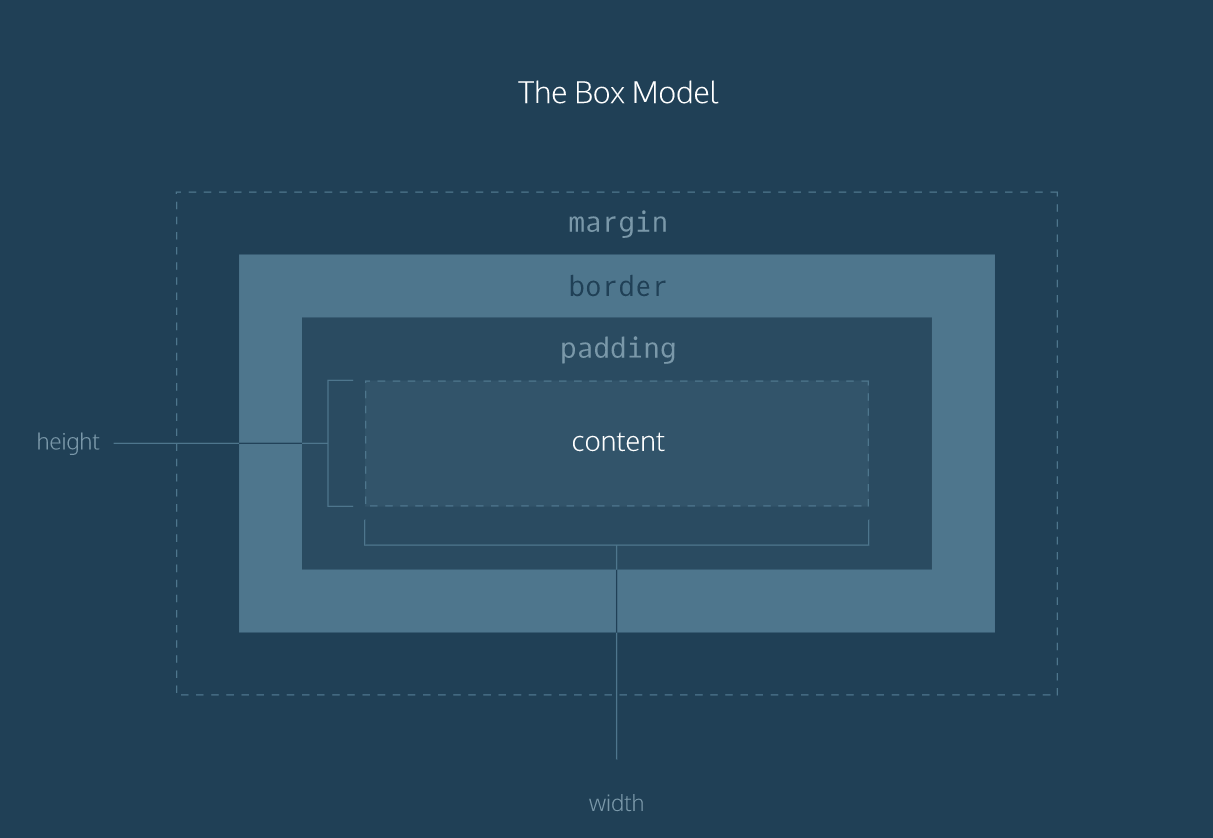
<ul class='main-list'> <li> ... </li> <li> ... </li> <li> ... </li> </ul>

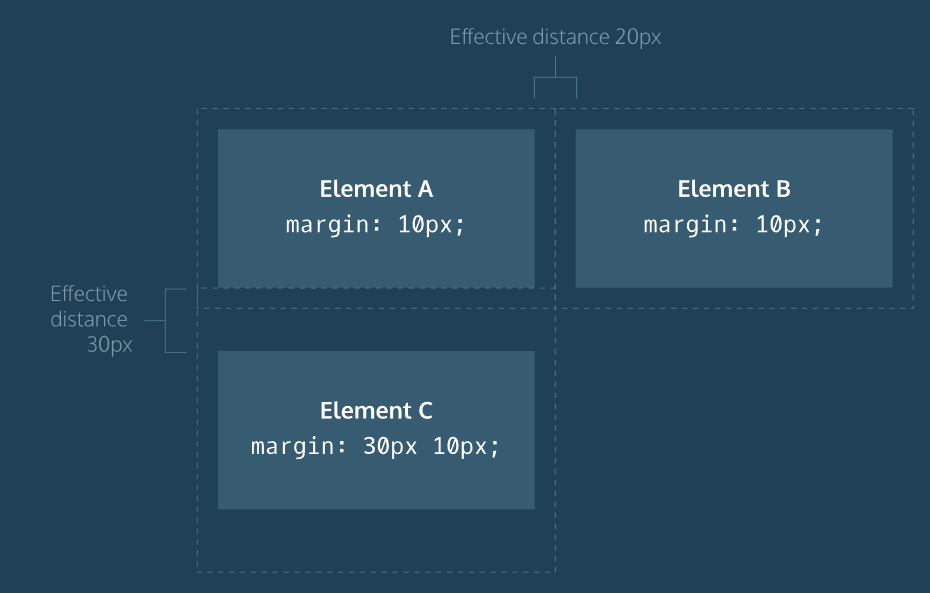
The nested <li> elements are selected with the following CSS:

.main-list li { }

Opacity is the measure of how transparent an element is. It's measured from 0 to 1, with 1 representing 100%, or fully visible and opaque, and 0 representing 0%, or fully invisible.

Opacity can be used to make elements fade into others for a nice overlay effect.





The overflow property controls what happens to content that spills, or overflows, outside its box. It can be set to one of the following values:

* hidden - when set to this value, any content that overflows will be hidden from view.
* scroll - when set to this value, a scrollbar will be added to the element's box so that the rest of the content can be viewed by scrolling.
* visible - when set to this value, the overflow content will be displayed outside of the containing element. Note, this is the default valu

Resetting Defaults

All major web browsers have a default stylesheet they use in the absence of an external stylesheet. These default stylesheets are known as *user agent stylesheets*. In this case, the term "[user agent](https://en.wikipedia.org/wiki/User_agent)" is a technical term for the browser.

User agent stylesheets often have default CSS rules that set default values for padding and margin. This affects how the browser displays HTML elements, which can make it difficult for a developer to design or style a web page.

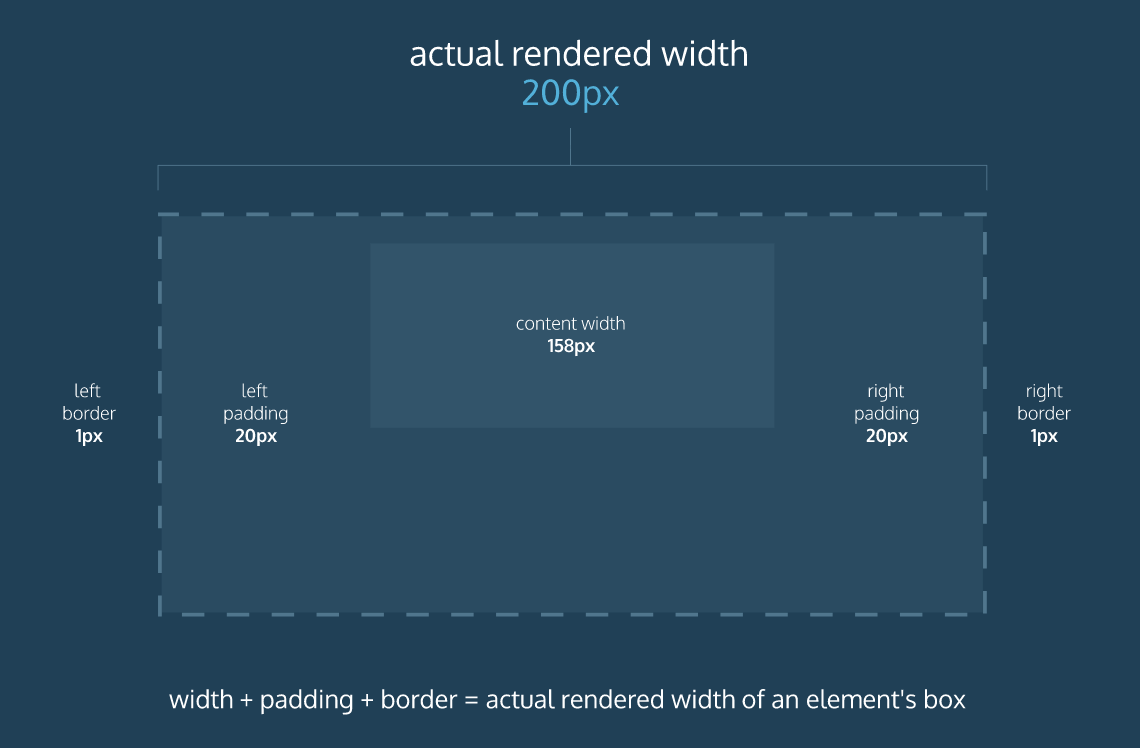
Many developers choose to reset these default values so that they can truly work with a clean slate.

\* { margin: 0; padding: 0; }

The code in the example above resets the default margin and padding values of all HTML elements. It is often the first CSS rule in an external stylesheet.

Note that both properties are both set to 0. When these properties are set to 0, they do not require a unit of measurement.

**Note:** What's the difference between display: none and visibility: hidden? An element with display: none will be completely removed from the web page. An element with visibility: hidden, however, will not be visible on the web page, but the space reserved for it will.



Fortunately, we can reset the entire box model and specify a new one: border-box.

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

The code in the example above resets the box model to border-box for all HTML elements. This new box model avoids the dimensional issues that exist in the former box model you learned about.

In this box model, the height and width of the box will remain fixed. The border thickness and padding will be included inside of the box, which means the overall dimensions of the box do not change.

<h1>Hello World</h1>

When an element's position is set to absolute all other elements on the page will ignore the element and act like it is not present on the page.

Inline Display

Every HTML element has a default display value that dictates if it can share horizontal space with other elements. Some elements fill the entire browser from left to right regardless of the size of their content. Other elements only take up as much horizontal space as their content requires and can be directly next to other elements.

In this lesson, we’ll cover three values for the display property: inline, block, and inline-block.

The default display for some tags, such as <em>, <strong>, and <a>, is called *inline*. Inline elements have a box that wraps tightly around their content, only taking up the amount of space necessary to display their content and not requiring a new line after each element. The height and width of these elements cannot be specified in the CSS document. For example, the text of an anchor tag will, by default, be displayed on the same line as the text inside of an emphasize element. Each of these will only be as wide as necessary to contain their content.

To learn more about <em>inline</em> elements, click <a href="#">here</a>.

In the example above, the <em> element is inline, because it displays its content on the same line as the content surrounding it, including the anchor tag. This example will display:

To learn more about *inline* elements, click [here](https://developer.mozilla.org/en-US/docs/Web/HTML/Inline_elements).

The CSS display property provides the ability to make any element an inline element. This includes elements that are not inline by default such as paragraphs, divs, and headings.

h1 { display: inline; }

The CSS in the example above will change the display of all <h1> elements to inline. The browser will render <h1>elements on the same line as other inline elements immediately before or after them (if there are any).

Block Display

Some elements are not displayed in the same line as the content around them. These are called *block-level* elements. These elements fill the entire width of the page and, unless specified, are the height necessary to accommodate the content inside of them.

Elements that are block-level by default include all levels of heading elements (<h1> through <h6>), <p>, <div> and <footer>. For a complete list of block level elements, visit [the MDN documentation](https://developer.mozilla.org/en-US/docs/Web/HTML/Block-level_elements).

strong { display: block; }

In the example above, all <strong> elements will be displayed on their own line, with no content directly on either side of them even though their contents may not fill the width of most computer screens.

# Clear

The float property can also be used to float multiple elements at once. However, when multiple floated elements have different heights, it can affect their layout on the page. Specifically, elements can "bump" into each other and not allow other elements to properly move to the left or right.

The clear property specifies how elements should behave when they bump into each other on the page. It can take on one of the following values:

1. left — the left side of the element will not touch any other element within the same containing element.
2. right — the right side of the element will not touch any other element within the same containing element.
3. both — neither side of the element will touch any other element within the same containing element.
4. none — the element can touch either side.

# Hexadecimal

One syntax that we can use to specify colors is called hexadecimal. Colors specified using this system are called hex colors. A hex color begins with a hash character (#) which is followed by three or six characters. The characters represent values for red, blue and green.

# RGB Colors

There is another syntax for representing RGB values that uses decimal numbers. It looks like this: color: rgb(23, 45, 23); Here, each of the three values represents a color component, and each can have a decimal number value from 0 to 255. The first number represents the amount of red, the second is green, and the third is blue. These colors are exactly the same as hex, but with a different syntax and a different number system.

In general, hex and decimal color representations are equivalent. Which you choose is a matter of personal taste. That said, it's good to choose one and be consistent throughout your CSS, because it's easier to compare hex to hex and decimal to decimal.

# Hue, Saturation, and Lightness

The RGB color scheme is convenient because it's very close to how computers represent colors internally. There's another equally powerful system in CSS called the hue-saturation-lightness color scheme, abbreviated as HSL.

The syntax for HSL is similar to the decimal form of RGB, though it differs in important ways. The first number represents the degree of the hue, and can be between 0 and 360. The second and third numbers are percentages representing saturation and lightness respectively. Here is an example:

color: hsl(120, 60%, 70%);

Hue is the first number. It refers to an angle on a color wheel. Red is 0 degrees, Green is 120 degrees, Blue is 240 degrees, and then back to Red at 360. You can see an example of this [color wheel here](http://dba.med.sc.edu/price/irf/Adobe_tg/models/images/hsl_top.JPG).

Saturation refers to the intensity or purity of the color. If you imagine a line segment drawn from the center of the color wheel to the perimeter, the saturation is a point on that line segment. If you spin that line segment to different angles, you'll see how that saturation looks for different hues. The saturation increases towards 100% as the point gets closer to the edge (the color becomes more rich). The saturation decreases towards 0% as the point gets closer to the center (the color becomes more gray).

Lightness refers to how light or dark the color is. Halfway, or 50%, is normal lightness. Imagine a sliding dimmer on a light switch that starts halfway. Sliding the dimmer up towards 100% makes the color lighter, closer to white. Sliding the dimmer down towards 0% makes the color darker, closer to black.

HSL is convenient for adjusting colors. In RGB, making the color a little darker may affect all three color components. In HSL, that's as easy as changing the lightness value. HSL is also useful for making a set of colors that work well together by selecting various colors that have the same lightness and saturation but different hues.

# Opacity and Alpha

All of the colors we've seen so far have been opaque, or non-transparent. When we overlap two opaque elements, nothing from the bottom element shows through the top element. In this exercise, we'll change the opacity, or the amount of transparency, of some colors so that some or all of the bottom elements are visible through a covering element.

To use opacity in the HSL color scheme, use hsla instead of hsl, and four values instead of three. For example:

color: hsla(34, 100%, 50%, 0.1);

The first three values work the same as hsl. The fourth value (which we have not seen before) is the alpha. This last value is sometimes called the opacity.

Alpha is a decimal number from zero to one. If alpha is zero, the color will be completely transparent. If alpha is one, the color will be opaque. The value for half transparent would be 0.5.

You can think of the alpha value as, "the amount of the background to mix with the foreground". When a color's alpha is below one, any color behind it will be blended in. The blending happens for each pixel; no blurring occurs.

The RGB color scheme has a similar syntax for opacity, rgba. Again, the first three values work the same as rgb and the last value is the alpha. Here's an example:

color: rgba(234, 45, 98, 0.33);

Alpha can only be used with HSL and RGB colors; we cannot add the alpha value to color: green color: #FFFFF.

There is, however, a named color keyword for zero opacity, transparent. It's equivalent to rgba(0, 0, 0, 0). It's used like any other color keyword:

color: transparent;

When setting typefaces on a web page, keep the following points in mind:

1. The font specified in a stylesheet must be installed on a user's computer in order for that font to display when a user visit the web page. We'll learn how to work around this issue in a later exercise.
2. You've probably noticed that we haven't been specifying a typeface in previous exercises of this course. How exactly does the browser know what typeface to use when displaying the web page? The default typeface for all HTML elements is Times New Roman. You may be familiar with this typeface if you have ever used a formatted word processor.
3. It's a good practice to limit the number of typefaces used on a web page to 2 or 3.
4. When the name of a typeface consists of more than one word, it must be enclosed in double quotes (otherwise it will not be recognized), like so:

h1 { font-family: "Courier New"; }

If we want to bold text in a web page, we can set the font-weight to bold.

p { font-weight: bold; }

If we want to ensure that text is not bold, we can set the font-weight to normal.

p { font-weight: normal; }

# Font Style

You can also italicize text with the font-style property.

h3 { font-style: italic; }

The italic value causes text to appear in italics. The font-style property also has a normal value which is the default.

# Word Spacing

You can also increase the spacing between words in a body of text, technically known as word spacing.

To do so, you can use the word-spacing property:

h1 { word-spacing: 0.3em; }

The default amount of space between words is usually 0.25em. In the example above, the word spacing is set to 0.3em, which represents an increase of only .05em in word spacing.

It's not common to increase the spacing between words, but it may help enhance the readability of bolded or enlarged text. Note, again, that the preferred unit is ems.



Line Height

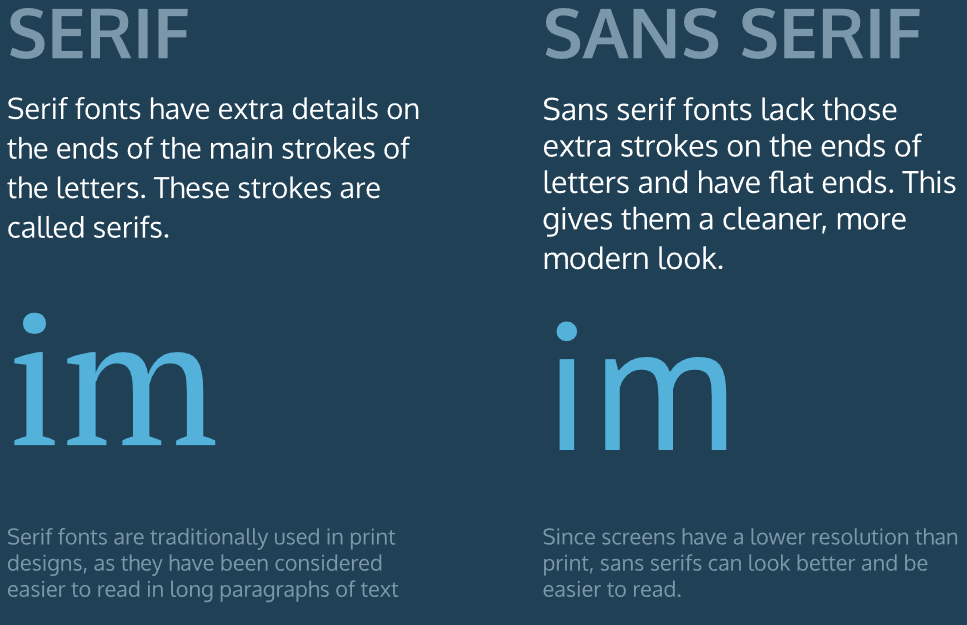
We often modify line-height to make text on a web page easier to read. When text is styled to appear larger, the vertical spacing between lines of text can decrease, creating text that is difficult to read, particularly in paragraphs.

We can use the line-height property to set how tall we want the line containing our text to be, regardless of the height of the text. Line heights can take one of several values:

1. A unitless number, such as 1.2. This number is an absolute value that will compute the line height as a ratio of the font size.
2. A number specified by unit, such as 12px. This number can be any valid CSS unit, such as pixels, percents, ems, or rems.

Generally, the unitless ratio value is the preferred method, since it is responsive and based exclusively on the current font size. In other words, if we change the font size, a unitless line-height would automatically readjust, whereas the pixel value would remain static.

p { line-height: 1.4; }



//https://www.codecademy.com/courses/learn-css-typography/lessons/css-typography/exercises/linking-fonts-i?action=lesson\_resume&course\_redirect=learn-css

When we have the link to the font of our choice, we can add the font to the <head> section of the HTML document, using the <link> tag and the href.

Let's take a look at a few examples:

1. A single linked font, using Droid Serif as an example:

<head> <link href="https://fonts.googleapis.com/css?family=Droid+Serif" type="text/css" rel="stylesheet"> </head>

2. Multiple linked fonts, using the Droid Serif and Playfair Display fonts as an example:

<head> <link href="https://fonts.googleapis.com/css?family=Droid+Serif|Playfair+Display" type="text/css" rel="stylesheet"> </head>

3. Multiple linked fonts, along with weights and styles. Here Droid Serif has font weights of 400, 700, and 700i, while Playfair Display has font weights of 400, 700, and 900i:

<head> <link href="https://fonts.googleapis.com/css?family=Droid+Serif:400,700,700i|Playfair+Display:400,700,900i" rel="stylesheet"> </head>

Once a font is linked, we can create CSS selectors to target elements, just as we do with other fonts.

# Font-Face III

While Google Fonts and other resources can broaden font selection, you may wish to use an entirely different font or abstain from using a font from an external service.

We can modify our @font-face rule to use local font files as well. We can supply the user with the desired font family and host it along with our site instead of depending on a different site.

@font-face { font-family: "Roboto"; src: url(fonts/Roboto.woff2) format('woff2'), url(fonts/Roboto.woff) format('woff'), url(fonts/Roboto.tff) format('truetype'); }

Here, you'll notice:

1. The main difference is the use of a relative filepath instead of a web URL.
2. We add a format for each file to specify which font to use. Different browsers support different font types, so providing multiple font file options will support more browsers.

As of now .woff2 appears to be the way of the future, due to greatly reduced file sizes and improved performance, but many browsers still don’t support it. There are lots of great sources to find fonts to use locally, such as [Font Squirrel](https://www.fontsquirrel.com/).