Abbreviation and Introduction

HyperText Markup Language (HTML)--how you want different parts of your content to be displayed
Cascading Style Sheets (CSS)
"uniform resource locators" (URLs)

- HTML is for adding meaning to raw content by marking it up.
- CSS is for formatting that marked up content.
- JavaScript is for making that content and formatting interactive.
- HTML as the abstract text and images behind a web page, CSS as the page that actually gets displayed, and JavaScript as the behaviors that can manipulate both HTML and CSS.

Web Development: build tools, domain name, web server
Frameworks: abstract away some of the redundant aspects of creating web pages from scratch (e.g., Bootstrap, ZURB foundation, and Pure CSS)
Mozilla Developer Network-- "MDN <some-element>"

ATOM

Cmd+N-- create another **untitled** tab
Cmd+T—search files
Split Right (click right)--open that file in a new pane, useful for examining a CSS file and its related HTML file at the same time.

HTML

Elements (Attributes-element-content: attribute adds meaning to the element it's attached to)

- *Metadata*

<head>: page title, CSS links, other abstract things
<title>: page title
<html lang='en'>: language (http://www.iana.org/assignments/language-subtag-registry/language-subtag-registry)
<meta charset='UTF-8'/>: character type

- *block-level elements* (flow content): always drawn on a new line

<body>: headings, paragraphs, other things you can see
<p>: paragraph text
<h>: six levels (h1, h2…)
: unordered list, <ul> elements should only contain <li>, otherwise wrap other elements within
: ordered list
: items in the list

- *inline elements* (phrasing content): can affect sections of text anywhere within a line

: emphasized italicized text
: bold

<a href=''>: link
Absolute: start with the "scheme" (typically http:// or https://), followed by the domain name of the website, then the path of the target web page
Relative: point to another file in your website from the vantage point of the file you're editing. the scheme and domain name are the same as the current page, so the only thing you need to supply is the path. Each folder and file in a path is separated by a forward slash (/).
From mother to son: <a href='misc/extras.html'>
From son to mother: <a href='../links.html'>, multiple: ../../elsewhere.html
Root-relative links: home page, they're relative to the "root" of the entire website, you can add more folders and files to the path after that initial slash
P.S. Spaces aren't allowed in URLs

: image
Attribute: src (points to the image file you want to display); width; height; alt (defines a "text alternative" to the image being displayed).
image format: pixel-based image formats need to be twice as big as you want them to appear on retina displays.
-jpg: large color palettes without exorbitantly increasing file size, don't allow for transparent pixels, good for photos
-gif: simple animations, limited for color palette, Transparent pixels are a binary option for GIFs, can't have semi-opaque pixels, make it difficult to get high levels of detail on a transparent background
-png: great for anything that's not a photo or animated, excellent fit for icons, technical diagrams, logos
-svg: it can scale up or down to any dimension without loss of quality, a wonderful tool for responsive design. for them to display consistently across browsers, you need to convert any text fields to outlines using your image editor, If your images contain a lot of text (like the fancy screenshots in this tutorial), this can have a big impact on file size.

- *Empty elements* (" / " in all empty HTML elements is entirely optional)

</br>: a hard line break
<hr/>: a "horizontal rule", which represents a thematic break.

HTML entity: a special character that can't be represented as plain text in an HTML document. either means it's a reserved character in HTML or you don't have a key on your keyboard for it. Entities always begin with an ampersand (&) and end with a semicolon (;).
(reserved characters: <, >, & they aren't allowed to be inserted into an HTML document without being encoded. syntax: < begins a new tag, > ends a tag; & sets off an HTML entity)
quotes: “ ” ‘ ’
Example:

```
<p>If you&rsquo;re into &ldquo;web typography,&rdquo; you&rsquo;ll also find
  yourself using curly quotes quite a bit.</p>
```

!-- --!: note

Code example:

```
<!DOCTYPE html>
<html>
 <head>
  <title>Interneting Is Easy!</title>
 </head>
 <body>
  <h1>Interneting Is Easy!</h1>
  <p>First, we need to learn some basic HTML.</p>

  <h2>Headings</h2>
  <p>Headings define the outline of your site. There are six levels of
  headings.</p>
 </body>
</html>
```

CSS

Reference: https://developer.mozilla.org/en-US/docs/Web/CSS/Reference

The CSS hierarchy for every web page looks like this:

- The browser's default stylesheet
- User-defined stylesheets
- External stylesheets (that's us)
- Page-specific styles (that's also us)
- Inline styles (that could be us, but it never should be)

This is ordered from least to most precedence, which means styles defined in each subsequent step *override* previous ones.

Type Selectors(example)

```
body {
 color: #FF0000;
}
```

CSS basic properties

In html <link rel='stylesheet' href='styles.css'/>

- color
- background-color
- font-size (px and em-defining sizes relative to some base font)
- font-family

```
h1, h2, h3, h4, h5, h6 {
 font-family: "Helvetica", "Arial", sans-serif;
```

```
}
```
(load the left-most one first (Helvetica), falls back to Arial if the user doesn't have it, and finally chooses the system's default sans serif font
Nowadays, system fonts have been largely superseded by web fonts.

- font-weight : boldness
- font-style : italized or not
- list-style-type (circle, lower-roman)
- text-decoration: whether text is underlined or not

none-remove underline
line-through= <del> (<ins>) in html

- text-align: left, right, center, or justify

Note: /* */

Page-specific styles

<style> (in html): used to add page-specific CSS rules to individual HTML documents, always lives in the <head> of a web page

Page-specific styles occasionally come in handy when you're in a rush, but it's almost always better to store all your CSS in external stylesheets opposed to <style> elements.

inline styles

```
<p>Want to try crossing out an <a href='nowhere.html'
  style='color: #990000; text-decoration: line-through;'>obsolete link</a>?
  This is your chance!</p>
```
the most specific way to define CSS
should be avoided at all costs because they make it impossible to alter styles from an external stylesheet. Use CSS classes instead

multiple stylesheets

CSS rules can be spread across several external stylesheets by adding multiple <link/> elements to the same page. A common use case is to separate out styles for different sections of your site.

```
<!-- All product pages have this -->
<head>
  <link rel='stylesheet' href='styles.css'/>
  <link rel='stylesheet' href='product.css'/>
</head>
<!-- While all blog posts have this -->
<head>
  <link rel='stylesheet' href='styles.css'/>
  <link rel='stylesheet' href='blog.css'/>
</head>
```
Stylesheets that come later will override styles in earlier ones.

CSS box model: padding, borders, margins, block boxes, and inline boxes.

CSS treats each element in your HTML document as a "box" with a bunch of different properties that determine where it appears on the page.

- Block boxes always appear below the previous block element.
- The **width of block boxes** is set automatically based on **the width of its parent container**. In this case, our blocks are always the width of the browser window.
- The default **height of block boxes** is based on **the content it contains**.
- **Inline boxes** don't affect **vertical spacing**. They're not for determining layout—they're for styling stuff *inside* of a block.
- The **width of inline boxes** is based on **the content it contains**, not the width of the parent element.

display: We can override the default box type of HTML elements with the CSS display property. (blocks instead of inline)

useful for turning <a> elements into buttons or format <img/> elements

Four properties of CSS box model

- Content - The text, image, or other media content in the element.
- Padding - The space between the box's content and its border. (whole or one side like left, top, etc)

two values to the padding property, it's interpreted as the vertical and horizontal padding values, respectively

```
p {
  padding: 20px 10px;  /* Vertical  Horizontal */
}
```

The values are interpreted clockwise, starting at the top

- Border - The line between the box's padding and margin. (size, style, color)

border-radius : 5px

- Margin - The space between the box and surrounding boxes.

Padding or margin

The padding of a box has a background, while margins are always transparent.
Padding is included in the click area of an element, while margins aren't.
Margins collapse vertically, while padding doesn't

Margin on inline elements

Inline boxes completely ignore the top and bottom margins of an element.
If you want to play with the vertical space of a page, you must be working with block-level elements

Vertical margin collapse

When you have two boxes with vertical margins sitting right next to each other, only the biggest one is displayed.

Preventing margin collapse

Only consecutive elements can collapse into each other. Putting an element with non-zero height (hence the padding-top) between our paragraphs forces them to display both the 25px top margin and the 50px bottom margin.

```
<div style='padding-top: 1px'></div>
```

an alternative solution would be to use padding to space out our paragraphs, but this only works if you're not using the padding for anything else

A third option to avoid margin collapse is to stick to a bottom-only or top-only margin convention.

Generic boxes

Both and are "container" elements that don't have any affect on the semantic structure of an HTML document.

 block level

Width, height (define the box's content)

 inline level

box-sizing: change how the width of a box is calculated.

- content-box: padding and border are both added on top of whatever explicit dimensions width and height you set.
- border-box (best practice): This forces the actual width of the box to be 200px—including padding and borders. Of course, this means that the content width is now determined automatically

Aligning boxes

"auto-margins" for center alignment, "floats" for left/right alignment, and "flexbox" for complete control over alignment.

Note that this only works on blocks that have an explicit width defined on them. Remove that width: 200px line, and our button will be the full width of the browser, making "center alignment" meaningless.

Resetting styles: "universal" CSS selector (*)

Class selectors

Class selectors require two things:

- A class attribute on the HTML element in question.
- A matching CSS class selector in your stylesheet.

In html: <p class='synopsis'>

In CSS: .synopsis

When there's two conflicting class properties in a CSS file, the last one is always the one that gets applied.

BUT!! The order of the class attribute in HTML element has no effect on override behavior.

Descendant selectors

target only those elements that are inside of another element

```
.synopsis em {
  font-style: normal;
}
```

Pseudo-classes

class selectors that you don't have to write on your own because they're built into the browser.

- :link – A link the user has never visited.
- :visited – A link the user has visited before.
- :hover – A link with the user's mouse over it.

- :active – A link that's being pressed down by a mouse (or finger).
- :last-of-type – selects the final element of a particular type in its parent element. (also for the first-of-type, but this will be applied to every div that wraps the content) you would need to limit its scope using a child selector, like so:

```
.page > p:first-of-type {
  color: #7E8184;
  font-style: italic;
}

a:link {
  color: blue; !! use directly the color
  text-decoration: none;
}

a:visited:hover {
  color: orange;
}

.button:active,
.button:visited:active {
  color: #FFF;
  background-color: #5995DA;  /* Blue */
}
```

ID selectors

You can only have *one* element with the same ID per page
They require an id attribute on whatever HTML element you're trying to select, if we wanted to share this style with another button, we'd have to give it another unique id attribute.
The corresponding CSS selector must begin with a hash sign (#) opposed to a dot.

URL fragments

id attributes need to be unique because they serve as the target for "URL fragments"
Fragments are how you point the user to a specific part of a web page (scheme-domain-path-fragment)
we can omit the URL entirely if we're linking to a different section on the same page:

```
<!-- From the same page -->
<a href='#button-2'>Go to Button Two</a>

<!-- From a different page -->
<a href='selectors.html#button-2'>Go to Button Two</a>
```

CSS specificity

the weight given to different categories of selectors
ID selectors have higher specificity than class selectors, so the whole "order matters" concept only works when all your rules have the same specificity.
Specificity from greatest to smallest:

- #button-2

- .button:link
- a:link and .synopsis em (they're equal)
- .button
- a

BEM: attempts to make CSS rules more reusable by making everything a class selector.

Floats

Left align—float: left
Right align--float: right; /* Right-aligned */
float: none;   /* Revert to default flow */
center align: margin: 0 auto
(only applies to block boxes. Inline boxes are aligned with the text-align property)
Floated boxes always align to the left or right of their parent element
When you float multiple elements in the same direction, they'll stack horizontally, much like the default vertical layout algorithm, except rotated 90 degrees.

Clearing floats
clear: both/right/left (if there's other elements than floats in the box)
tell a block to ignore any floats that appear before it. A cleared element always appears after any floats.
Floated elements don't count towards its height.
Overflow: hidden (if there're only floats in the box), combined with a background color

Full-bleed layouts
centering requires an explicit width property

```
<div class='container'>            <!-- Add this -->
   <div class='page'>
     <div class='sidebar'>Sidebar</div>
     <div class='content'>Content</div>
   </div>
 </div>
```

Add corresponding settings in CSS

Equal-width columns

```
<div class='footer'>
  <div class='column'></div>
  <div class='column'></div>
  <div class='column'></div>
</div>
```

```
width: 31%;
```

Percentages in CSS are relative to the width of the parent element. The result is three columns that automatically resize to one-third of the browser window.

Floats for content

Overflow: hidden trick. Sticking it on our .comment box made sure that the text “horizontally cleared” (that’s not a technical term) the floated image. Without it, the last line of the .comment text would hang underneath the image.

Flexbox

alignment, direction, order, and size
reserving floats for when you need text to flow around a box (i.e., a magazine-style layout) or when you need to support legacy web browsers

flex containers and flex items
Every HTML element that’s a direct child of a flex container is an “item”.
For the most part, it’s up to the container to determine their layout. The main purpose of flex items are to let their container know how many things it needs to position.
display: flex;
justify-content: center (define the horizontal alignment of its items, same effect as adding a margin: 0 auto)
other values—center, flex-start, flex-end, space-around, space-between
Vertical alignment: adding an align-items property to a flex container.
align-items: center; (center, flex-start(top), flex-end(bottom), stretch-lets you display the background of each element., baseline)
stretch useful for creating equal-height columns with a variable amount of content in each one
flex-wrap: wrap (create a grid)

transform rows into columns
flex-direction: column; (row) (!When you rotate the direction of a container, you also rotate the direction of the justify-content property.)
The flex-direction property also offers you control over the order in which items appear via the row-reverse and column-reverse properties.

flex item order
adding an order property to a flex item defines its order in the container without affecting surrounding items. Its default value is 0, and increasing or decreasing it from there moves the item to the right or left, order works across row/column boundaries.
align-self (align a single item without influencing others)

Flex items are flexible: they can shrink and stretch to match the width of their containers.
flex allows individual items in a flex container to have flexible widths (It works as a weight that tells the flex container how to distribute extra space to each item. Ex. an item with a flex value of 2 will grow twice as fast as items with the default value of 1.)
flex: initial: falls back to the item’s explicit width property.
many websites have a fixed-width sidebar (or multiple sidebars) and a flexible content block containing the main text of the page.

flex items and auto-margins
auto-margins as a “divider” for flex items in the same container.

- Use display: flex; to create a flex container.
- Use justify-content to define the horizontal alignment of items.
- Use align-items to define the vertical alignment of items.
- Use flex-direction if you need columns instead of rows.
- Use the row-reverse or column-reverse values to flip item order.
- Use order to customize the order of individual elements.
- Use align-self to vertically align individual items.
- Use flex to create flexible boxes that can stretch and shrink.

Responsive design

- your website should display equally well in everything from widescreen monitors to mobile phones

```
@media+media type + media feature
@media only screen and (min-width: 401px) and (max-width: 960px) {
  body {
    background-color: #F5CF8E; /* Yellow */
  }
}
```

- A “fluid” layout is one that stretches and shrinks to fill the width of the screen, just like the flexible boxes we covered a few chapters ago.
- A “fixed-width” layout is the opposite: it has the same width regardless of the screen dimensions
- fluid layouts for mobile/tablet devices and fixed-width layouts for wider screens.

Set features between * and media, this will be applied to all devices

Disabling viewport zooming

```
<meta name='viewport'
    content='width=device-width, initial-scale=1.0, maximum-scale=1.0' />
```

Sectioning elements

- headers
- footers
- inline semantic html

<article>: independent article in a web page (ex can be grabbed by search engine)
<article>’s are essentially mini web pages in your HTML document. They have their own headers, footers, and document outline that are completely isolated from the rest of your site.
<section>: it doesn’t need to make sense outside the context of the document. an explicit way to define the sections in a document outline.
each <section> can have its own set of <h1> through <h6> headings that are independent of the rest of the page.
using <section> only as a replacement for container ’s when appropriate.
!! each <section> element should contain at least one heading, otherwise it will add an “untitled section” to your document outline.
only use <section> as a more descriptive wrapper for the implicitly defined sections of your page.

<nav>: main site navigation, links to related pages in a sidebar, tables of content, and pretty much any group of links

<header>: It denotes introductory content for a section, article, or entire web page. "Introductory content" can be anything from your company's logo to navigational aids or author information.

It's a best practice to wrap a website's name/logo and main navigation in a <header>

associated with the nearest sectioning element

<footer>: like header but at the end of the article, associated with the nearest sectioning element

<aside>: remove information from an article, good for advertisement, highlighting definitions, stats, or quotations.Ideal for a sidebar when used outside an <article>

<time>: a time of day or a calendar date

<time datetime='2017-1-3 15:00-0800' >

<address>: It defines contact information for the author of the article or web page in question. <address> should not be used for arbitrary physical addresses.

<figure>: a self-contained "figure", like a diagram, illustration, or even a code snippet.

<figcaption>: optional, and it associates a caption with its parent <figure> element.

Common use: add visible descriptions to the <img/> elements in an article

```
section, article, aside, footer, header, nav {
  display: block;
}
```

This makes the new semantic elements behave like elements (which are block boxes, not inline boxes) in legacy browsers.

divs for layout

when none of the semantic HTML elements we just covered would make sense

relevant for flexbox, as it requires lots of 's to group flex items correctly.

- Schema.org microdata lets you alter the appearance of your site in search engine results.
- Twitter cards define how your web page is displayed in tweets.
- Open Graph metadata changes how Facebook shares your content.

HTML forms

collect input from your website's visitors.

two aspects of a functional HTML form: the frontend user interface and the backend server.

<form>: action attribute defines the URL that processes the form.

Common backend technologies for processing forms include Node.js, PHP, and Ruby on Rails.

The method attribute can be either post or get, use post when you're changing data on the server, reserving get for when you're only getting data.

- **text**

```
<form action='' method='get' class='speaker-form'>
<div class='form-row'>
  <label for='full-name'>Name</label>
  <input id='full-name' name='full-name' type='text'/>
</div>
</form>
```

A label's for attribute must match the id attribute of its associated <input/> element.

```
.form-row input[type='text'] {
}
```

"attribute selector": It only matches <input/> elements that have a type attribute equal to text.

```
.form-row label {
}
```

- **email**

placeholder='joe@example.com'—display some default text when the <input/> element is empty.

Other built-in validation options: required, minlength, maxlength, and pattern

- **The :invalid and :valid**

```
.form-row input[type='text']:invalid,
.form-row input[type='email']:invalid {}
```

:focus--selects the element the user is currently filling out.

- **Radio**

Every radio button group you create should:

- Be wrapped in a <fieldset>, which is labeled with a <legend>.
- Associate a <label> element with each radio button.
- Use the **same name attribute** for each radio button in the group.
- Use different value attributes for each radio button.

```
<fieldset class='legacy-form-row'>
 <legend>Type of Talk</legend>
 <input id='talk-type-1'
      name='talk-type'
      type='radio'
      value='main-stage' />
 <label for='talk-type-1' class='radio-label'>Main Stage</label>
 <input id='talk-type-2'
      name='talk-type'
      type='radio'
      value='workshop'
      checked />
 <label for='talk-type-2' class='radio-label'>Workshop</label>
</fieldset>
```

- **dropdown menus**

The <select> element represents the dropdown menu, and it contains a bunch of <option> elements that represent each item.

```
<div class='form-row'>
 <label for='t-shirt'>T-Shirt Size</label>
 <select id='t-shirt' name='t-shirt'>
  <option value='xs'>Extra Small</option>
  <option value='s'>Small</option>
  <option value='m'>Medium</option>
```

```
   <option value='l'>Large</option>
  </select>
</div>
```

Change style:

```
.form-row select {
  width: 100%;
  padding: 5px;
  font-size: 14px;          /* This won't work in Chrome or Safari */
  -webkit-appearance: none;   /* This will make it work */
}
```

The -webkit prefix will only apply to Chrome and Safari (which are powered by the WebKit rendering engine), while Firefox will remain unaffected.

- **<textarea>:** multi-line text

```
<div class='form-row'>
  <label for='abstract'>Abstract</label>
  <textarea id='abstract' name='abstract'></textarea>
  <div class='instructions'>Describe your talk in 500 words or less</div>
</div>
```

Note that this isn't self-closing like the <input/> element, so you always need a closing </textarea> tag.

```
  resize: none; --don't allow user resize <textarea> elements to whatever dimensions they want.
```

- **Checkboxes**

```
<div class='form-row'>
  <label class='checkbox-label' for='available'>
  <input id='available'
        name='available'
        type='checkbox'
        value='is-available'/>
  <span>I'm actually available the date of the talk</span>
  </label>
</div>
```

- **Submit buttons**

```
<div class='form-row'>
  <button>Submit</button>
</div>
```