



Learning to build the web

This is by no means an exhaustive treatment of the content to be covered and, like the outline before it, it's meant as a starting point and will definitely get further elaboration over time.

What is the web

The web is many things. It's a development platform; it delivers applications, it delivers content, it plays media (encrypted or not). There is very few things that the web can't display on its own. Rather than address the latest and greatest framework we'll cover the building blocks of the web how to use them and where to go once you've learned the basics.

History

Basic Components of the web

We group these components together without implying one is more important than the other. Developers need to use all these technologies together to make the web work (well). This is most important when we discuss accessibility as this will permeate the other areas.

HTML

HTML (HyperText Markup Language) is the language we use to write web content. Whether it's a simple page or a complex application, they all use the same structure and presentation tags.

This is a basic example of an HTML document. It shows the basic structure that all pages should have (doctype, html root element, and head and body as the children).

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello!</title>
    <meta charset="utf-8">
    <link rel="stylesheet" href="styles.css">
    <script src="/script.js" defer></script>
  </head>
  <body>
    <h1>Hi there!</h1>

    <p>I'm your cool new webpage.</p>

  </body>
</html>
```

In these sections we'll cover the following topics:

- Structural Markup
- Presentational Markup
- Elements, Attributes and Accessibility
- Integrating CSS and JS to our web pages

Structural Markup

The first set of elements we'll discuss are structural elements or tags. These are the elements that represent the organization of the content in the page.

The first group of tags are the ones that organize the page: `<html>`, `<head>` and `<body>`.

The `<html>` element is our main container for the page.

The `<head>` of the document is where we place information about the page

These are some of the metadata elements that can go inside the `<head>` of an HTML document

- [link](#) allows authors to link their document to other resources. The destination of the link(s) is given by the href attribute, which must be present and must contain a valid URL. If the href attribute is absent, then the element does not define a link
- [meta](#) represents various kinds of metadata that cannot be expressed using other elements in this list. The most often used is the charset attribute to indicate the character set used in the document
- [noscript](#) represents nothing if scripting is enabled, and represents its children if scripting is disabled. It is used to present different markup to user agents that support scripting and those that don't support scripting, by affecting how the document is parsed
- [script](#) allows authors to include dynamic script and data blocks in their documents. The element is invisible to the user
- [style](#) embeds CSS style sheets in their documents. The style element is one of several inputs to the styling processing model. The element is also invisible to the user
- [title](#) represents the document's title or name. Authors should use titles that identify their documents even when they are used out of context, for example in a user's history or bookmarks, or in search results. The document's title is often different from its first heading, since the first heading does not have to stand alone when taken out of context

The <body> holds the content of the page. This is where we will place the majority of our content.

A second group of structural tags will help further organize the page content. These elements tell the browser to create a new hierarchy of elements inside them and are different than the third group we'll discuss later that are logical containers that will not change the document outline.

- [article](#) represents a complete, or self-contained, composition in a document, page, application, or site and that is, in principle, independently distributable or reusable, e.g. in syndication. This could be a forum post, a magazine or newspaper article, a blog entry, a user-submitted comment, an interactive widget or gadget, or any other independent item of content
- [aside](#) represents a section of a page that consists of content that is tangentially related to the content around the aside element, and which could be considered separate from that content. Such sections are often represented as sidebars in printed publications
- [nav](#) represents a section of a page that links to other pages or to parts within the page: a section with navigation links

- [section](#) represents a generic section of a document or application. A section, in this context, is a thematic grouping of content, typically with a heading

The third, and last, group of structural elements are logical containers for other elements and elements that represent the content hierarchy (headings, subheadings and heading groups) of the page's content.

- [header](#) represents a group of introductory or navigational aids
- [footer](#) represents a footer for its nearest ancestor sectioning content or sectioning root element. A footer typically contains information about its section such as who wrote it, links to related documents, copyright data, and the like
- [h1-h6 and hgroup](#) The first element of heading content in an element of sectioning content represents the heading for that section. Subsequent headings of equal or higher rank start new (implied) sections, headings of lower rank start implied subsections that are part of the previous one. In both cases, the element represents the heading of the implied section.

Tag Soup Markup

While we strive to create good web content and to show you how you do it, the wider web doesn't always play by the rules. You may find demos and pages that have markup like this:

```
<html>
  <body>
    <h1>Page title</h1>
    <p>Content</p>
  </body>
</html>
```

While this is technically valid HTML (in the sense that a browser will still display the content of a page like the example above) it is not correct HTML.

Browsers must render old content, some of it 20+ years old, as faithfully as possible. This include working with tags and practices that have been deprecated by the groups that recommends standards for the web or removed by one or more browser vendors.

So it boils down to this: **Do things the right way from the start**

Presentation Markup

These elements change the way your content looks and/or behaves in visual browsers. Some of these elements also support accessibility attributes beyond. You can customize the looks of these elements beyond its default presentation using CSS. We'll talk about attributes in the next section and take a deeper look at accessibility in a later section of this tutorial.

I've classed these elements by function. It may not be the correct grouping.

Paragraphs

The most basic structural element in HTML is the `<p>` element that represents a paragraph of information.

Preformatted text and code samples

By default HTML paragraphs ignore extra spacing. There are times, however, when we want to preserve the spacing on our text when we're working with computer code or poetry. This is where the `<pre>` element comes in handy.

The following example shows a Pascal program that we want to show our readers.

```
<pre>var i: Integer;  
begin  
    i := 1;  
end.</pre>
```

By wrapping it with `<pre>` tags we make sure that the spacing of the code will be preserved. It will look like this:

```
var i: Integer;  
begin  
    i := 1;  
end.
```

The `code` works in a different situation. Say for example that you're writing code documentation and want to highlight the name of a file or a shell command. This is where you'd use this element; it represents a fragment of computer code. This could be an HTML element name, a file name, a command to run in the command line, a computer program, or any other string that a computer would recognize.

```
<p>Install NPM by running <code>npm i -g npm</code> from  
your terminal</p>
```

And the code will look like this:

```
Install NPM by running npm i -g npm from your terminal
```

We can combine the two elements to give a semantically accurate structure to the markup surrounding our programs. We can re-write the program example like this:

```
<pre><code>var i: Integer;  
begin  
    i := 1;  
end.</code></pre>
```

and it should look the same as it did earlier.

```
var i: Integer;  
begin  
    i := 1;  
end.
```

Citing content

Because of the original use for the web as a document sharing system, we have support for block quotations and sourcing where appropriate. The [blockquote](#) element acts as a container one or more elements from a different document and source.

The [cite](#) elements indicates the name of the source the quotation is from. This is an inline element. In the example below the cite element is inside a paragraph, which is one way I would normally cite content in blocks.

```
<section>
  <blockquote>
    <p>The truth may be puzzling. It may take some work to
    grapple with.
    It may be counterintuitive. It may contradict deeply held
    prejudices. It may not be consonant with what we
    desperately want to
    be true. But our preferences do not determine what's true.
    We have a
    method, and that method helps us to reach not absolute
    truth, only
    asymptotic approaches to the truth – never there, just
    closer
    and closer, always finding vast new oceans of undiscovered
    possibilities. Cleverly designed experiments are the
    key.</p>
  </blockquote>
  <p>Carl Sagan, in "<cite>Wonder and Skepticism</cite>",
  from
  the <cite>Skeptical Inquirer</cite> Volume 19, Issue 1
  (January-February
  1995)</p>
</div>
```

Blockquote works with large blocks of content but there are times when we need to quote smaller fragments inside a paragraph. That's the intended use of the [q](#) element. Here we also use the cite element as an attribute and provide a URL to point to the resource cited in the parent element.

```
<p>The W3C page <cite>About W3C</cite> says the W3C's mission is <q cite="https://www.w3.org/Consortium/">To lead the World Wide Web to its full potential by developing protocols and guidelines that ensure long-term growth for the Web</q>.</p>
```

Listing content

- [ol](#)
- [ul](#)
- [li](#)

HTML provides 3 ways of listing content. The first two will generate default lists. `ol` will generate numbered lists (defaulting to Roman numerals) and `ul` generates bulleted lists (defaulting to filled circles). We'll use CSS to change the defaults when we cover CSS.

Example of `ol` numbered list

```
<ol>
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
</ol>
```

That produces the following result:

1. Item 1
2. Item 2
3. Item 3

Example of `ul` bulleted lists

```
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
</ul>
```


With the following results:

- Item 1
- Item 2
- Item 3

You can nest and mix the two types of lists, like in the example below:

```
<ol>
  <li>Item 1</li>
  <ul>
    <li>Item 1.1</li>
    <li>Item 1.2</li>
    <li>Item 1.3</li>
  </ul>
  <li>Item 2</li>
  <li>Item 3</li>
</ol>
```

That will produce this display:

1. Item 1
 1. Item 1.1
 2. Item 1.2
 3. Item 1.3
4. Item 2
5. Item 3

Description Lists

There is a third way to list content, the description list. A description list represents an association list consisting of zero or more name-value groups. A name-value group consists of one or more names (dt elements) followed by one or more values (dd elements). Within a single dl element, there should not be more than one dt element for each name.

- [dl](#)
- [dt](#)
- [dd](#)

```
<dl>
  <dt>Term 1</dt>
  <dd>This is the definition of the first term.</dd>
  <dt>Term 2</dt>
  <dd>This is the definition of the second term.</dd>
</dl>
```

It will produce the following code

Term 1

This is the definition of the first term.

Term 2

This is the definition of the second term.

You can have multiple terms and definitions grouped together.

```
<dl>
  <dt>Term 1</dt>
  <dd>This is the definition of the first term.</dd>
  <dd>This is another definition of the first term.</dd>
  <dt>Term 2</dt>
  <dt>Term 3 is related to term 2</dt>
  <dd>This is the definition of the second term.</dd>
</dl>
```

That looks like this:

Term 1

This is the definition of the first term.

This is another definition of the first term.

Term 2

Term 3 is related to term 2

This is the definition of the second term.

Figures and images

Images are an important part of the web. Since the early days of HTML ([1993](#)) We've had a way to embed images directly on our HTML documents using the `img` tag.

The tag only required attribute is a source for the image using the `src` attribute. There are two ways to reference images. The first one is to use a URL to a remote resource, like this:

```

```



And the second one is to use a path (absolute or relative) to the image in the same server as where the page is:

```

```

We can also provide accessibility cues for screen readers using the `alt` attribute. Screen readers will read the value of the attribute along with the other content of the page. Our next iteration of the image looks like this:

```

```

But the result doesn't change... the image looks the same as it did without the `alt` attribute. However, if you hold your mouse over the image, most browsers will display the value of the `alt` attribute as a tooltip. Assistive technology devices will use the `alt` attribute as the text to read when it hits the image.



As we've seen the `img` element by itself only renders the image and provides accessibility accommodations. In order to display more information about the images we must use a different element; the `figure` element.

In its most basic form, the `figure` element is a container for an `img` element. There shouldn't be any difference

```
<figure>
  
</figure>
```



You can add a caption for the image using the `figcaption` child element. This is in addition to the `alt` attribute in the `img` element itself, not a replacement.

```
<figure>
  
  <figcaption>People in a bus looking at the
  camera</figcaption>
</figure>
```



Figure 2: People in a bus looking at the camera

Although `figure` is most often used with images, you can use it with other elements that need caption such as video, code listings and others.

- [img](#)
- [figure](#)
- [figcaption](#)

Multimedia

Multimedia on the web has followed a progression.

In the beginning the web was a text-only medium, designed to exchange academic and technical documentation among researches and scientists.

Then we had images. The first time we could embed content other than text on the web.

Around the same time images became a thing we also got the `embed` element as an entry point for external content. We could finally embed audio and video directly on our web content. The example below would embed a QuickTime movie at the point where the element was inserted.

```
<embed type="video/quicktime" src="movie.mov" width="640" height="480">
```

The problem with `embed` and its coupling object is that browser makers and content developers have no say on implementation and security of the third party plugins needed to play embeddable content. One of the best known plugins (and a great target for hackers) is Macromedia/Adobe Flash.

HTML5 seeks to address security and bloat issues with plugins by providing ways to play the multimedia audio and video without using plugins. Enter `video` and `audio`.

These tags provide media playback functionality without plugins, it is all baked into the platform and the browsers. The simplest way to use video is to specify a single source for the video (either local or remote).

The poster attribute provides a placeholder image that will be visible until the video begins playing.

src indicates the source of the video.

type indicates what kind of video we are playing. This will become important when we have multiple sources.

```
<video controls src="https://www.html5rocks.com/en/tutorials/
video/basics/devstories.webm"
  type='video/webm'
  poster="https://www.html5rocks.com/en/tutorials/video/
basics/poster/poster.png">
</video>
```

It will produce the result below:

If the browser doesn't support the video format you will only see the poster image until you hit play and then you'll see a white or black rectangle where the video would be placed.

The downside of HTML5 video is that vendors were never able to come to an agreement as to what format to use as default. As a result we will have to work with multiple versions of our content to make sure we have as much support as possible.

Instead of using a single src attribute we use one or more source children elements to specify location and format for each of the video formats we want to make available to our users.

```
<video controls poster="https://www.html5rocks.com/en/
tutorials/video/basics/poster.png">
  <source src="https://www.html5rocks.com/en/tutorials/video/
basics/devstories.webm"
    type='video/webm' />
  <source
    src="https://www.html5rocks.com/en/tutorials/video/
basics/devstories.mp4"
```

```
type='video/mp4' />
</video>
```

Order does matter. Browsers will play the first video format they support so it's important to put the most likely candidate first. In this case the browser will test if it can play mp4 video and, if it can't, then it'll check if it can play WebM.

The audio tag works similarly. We have the option of making a single source available, in this case an MP3 file.

```
<audio controls
  src="http://www.sample-videos.com/audio/mp3/
crowd-cheering.mp3">
  Your browser does not support the <code>audio</code>
  element.
</audio>
```

Where the result looks like this:

Your browser does not support the audio element.

We can also use the audio element with one or more source child elements specifying different formats for the same media.

```
<audio controls="">
  Your browser does not support the <code>audio</code>
  element.
  <source src="http://www.sample-videos.com/audio/mp3/
crowd-cheering.mp3">
</audio>
```

Note that we also provide a textual fallback in case the browser doesn't support any of the provided formats.

Your browser does not support the audio element.

- [video](#)
- [audio](#)
- [source](#)
- [track](#)

Both audio and video elements have accessibility implications. We'll discuss these implications when we talk about accessibility.

Special Containers

There are some elements that can be used to group their children and add further meaning to their children

`div` can be used with the `class`, `lang`, and `title` attributes to mark up semantics and styles for the `div` and its child elements.

```
<div class="paragraph1">
<p>Spicy jalapeno bacon ipsum dolor amet short ribs drumstick
burgdoggen, strip steak pig frankfurter leberkas turducken.
Bresaola meatloaf pork tongue salami shankle biltong
turducken
kevin. Filet mignon ribeye bresaola pastrami corned beef
short
loin capicola. Pork loin venison tail shoulder bacon brisket
boudin meatball burgdoggen. Turducken hamburger landjaeger
jerky short ribs ball tip. Shankle flank meatloaf porchetta,
bresaola venison strip steak pork chop tongue kielbasa ribeye
ham hock beef rump. Beef ribs short loin jowl, ball tip tail
capicola leberkas doner fatback flank sausage meatball.</p>

<p>Pig ball tip picanha, drumstick strip steak ribeye venison
andouille bacon. Turducken drumstick salami, pancetta beef
ribs
corned beef landjaeger brisket ham hock meatball picanha
swine.
Cow bresaola corned beef, sausage tail pork short loin spare
ribs beef ribs salami. Beef ribs pork loin drumstick sirloin.
Venison landjaeger fatback, tri-tip tongue beef ham hock
shankle
beef ribs tenderloin shoulder brisket alcatra. Corned beef
cupim
ribeye jowl picanha, shoulder short loin tri-tip short ribs
frankfurter alcatra shank beef meatloaf. Corned beef sausage
```

```
chuck shank landjaeger beef.</p>
</div>
```

span is the inline version of div and serves the same purpose. They are normally used to add microformat semantics and styles for its text and child elements.

```
<p>Pig ball tip picanha, drumstick strip steak ribeye venison
andouille bacon. Turducken drumstick salami, pancetta beef
ribs corned beef landjaeger brisket ham hock meatball picanha
swine. Cow bresaola corned beef, sausage tail pork short loin
spare ribs beef ribs salami. Beef ribs pork loin drumstick
sirloin. Venison landjaeger fatback, <span
class="wide">tri-tip
tongue beef ham hock shankle beef ribs tenderloin shoulder
brisket
alcatra</span>. Corned beef cupim ribeye jowl picanha,
shoulder
short loin tri-tip short ribs frankfurter alcatra shank beef
meatloaf. Corned beef sausage chuck shank landjaeger
beef.</p>
```

- [div](#)
- [span](#)

Hyperlinks

The web is made of links. The name World Wide **Web** implies relationships between resources. The a element creates relationships between resources both within a document and to external resources.

The first example sends people to Google search while the second shows a navigation menu with different

```
<p>Let me <a href="https://www.google.com">Google</a> it for
you.</p>
```

```
<nav>
  <ul>
    <li><a href="/">Home</a></li>
    <li><a href="about.html">About</a></li>
    <li><a href="content.html">Content</a></li>
  </ul>
</nav>
```

- [a](#)

Typographical styles

The most basic styles we see on the web are bold (represented by `strong`) and italics (represented by `em`). The actual definitions are a little more complex.

The `em` element represents stress emphasis of its contents.

```
<p>This is the <em>first</em> step of many.</p>
```

This is the *first* step of many.

The `strong` element represents strong importance, seriousness, or urgency for its contents.

- **Importance:** the `strong` element can be used to distinguish the part that really matters from other parts that might be more detailed, more jovial, or merely boilerplate.
- **Seriousness:** the `strong` element can be used to mark up a warning or caution notice.
- **Urgency:** the `strong` element can be used to denote contents that the user needs to see sooner than other parts of the document.

```
<p>Beware of the <strong>dark</strong> side</p>
```

Beware of the **dark** side

sup and sub provide superscript and subscript.

```
<p>Water: h<sup>2</sup>o</p>
```

h²o

```
<p> This is the formula for Carbon 12: 12<sub>N</sub></p>
```

This is the formula for Carbon 12: 12_N

- [em](#)
- [strong](#)
- [sub and sup](#)

Semantic Inflections

The `i` element represents a span of text in an alternate voice or mood, or otherwise offset from the normal prose in a manner indicating a different quality of text in Western texts.

In this example the content in French is a different voice than the default language so we mark it as such using the `i` element and a `lang` attribute to indicate the language for the fragment (French in this case).

```
<p>There is a certain <i lang="fr">je ne sais quoi</i> in  
the air.</p>
```

There is a certain *je ne sais quoi* in the air.

The **b** element represents a span of text to which attention is being drawn for utilitarian purposes without conveying any extra importance and with no implication of an alternate voice or mood.

```
<p>You enter a small room. Your <b>sword</b> glows brighter.  
A <b>rat</b> scurries past the corner wall.</p>
```

You enter a small room. Your **sword** glows brighter. A **rat** scurries past the corner wall.

The **s** element represents contents that are no longer accurate or no longer relevant. This element is not meant to indicate editorial changes for the document, there are other elements better used for it.

```
<p>The web was create in <s>1899</s> 1990.</p>
```

The web was create in 1899 1990.

The **dfn** element represents the defining instance of a term. The paragraph, description list group, or section that is the nearest ancestor of the **dfn** element must also contain the definition(s) for the term given by the **dfn** element.

In the following fragment, the term "Garage Door Opener" is defined in the first paragraph and used later in the document. In both cases, its abbreviation is what is displayed as a tooltip when you hover over the definition.

```
<p>The <dfn><abbr title="Garage Door Opener">GDO</abbr></dfn>  
is a device that allows off-world teams to open the iris.</p>  
<!-- ... later in the document: -->  
<p>Teal'c activated his <abbr title="Garage Door
```

```
Opener">GDO</abbr>
and so Hammond ordered the iris to be opened.</p>
```

With the addition of an `a` element, the reference can be made explicit by pointing later uses of the abbreviation to where it's defined:

```
<p>The <dfn id=gdo><abbr title="Garage Door
Opener">GDO</abbr></dfn>
is a device that allows off-world teams to open the iris.</p>
<!-- ... later in the document: -->
<p>Teal'c activated his <a href=#gdo><abbr title="Garage
Door Opener">GDO</abbr></a>
and so Hammond ordered the iris to be opened.</p>
```

The GDO (Garage Door Opener) is a device that allows off-world teams to open the iris.

Teal'c activated his GDO and so Hammond ordered the iris to be opened.

- [i](#)
- [b](#)
- [u](#)
- [s](#)
- [dfn](#)

Class and ID attributes

The class and id attributes may be specified on all HTML elements.

The attribute, if specified, must have a value that is a set of one or more space-separated tokens representing the various classes that the element belongs to.

The classes that an HTML element has assigned to it consists of all the classes returned when the value of the class attribute is **split on spaces** with duplicates removed.

There are no additional restrictions on the tokens authors can use in the class attribute, but authors are encouraged to use values that describe the nature of the content, rather than values that describe the desired presentation of the content.

In this example we use two classes, the first one, message, indicate purpose and the second one, info, tells what kind of message it is. We can have more than one type of message available.

```
<aside class="message info">
  <p>This is informational content for you to consider</p>
</aside>
```

This is informational content for you to consider

When specified on HTML elements, the id attribute value must be unique amongst all the IDs in the document and must contain at least one character other than spaces or space equivalent characters.

The id attribute specifies its element's unique identifier (ID).

There are no other restrictions on what form an ID can take; in particular, IDs can consist of just digits, start with a digit, start with an underscore, consist of just punctuation, etc.

An element's unique identifier can be used for a variety of purposes, most notably as a way to link to specific parts of a document using fragments, as a way to target an element when scripting, and as a way to style a specific element from CSS.

```
<h1 id="doc-title">This is the document title</h1>

<!-- later in the document -->

<p><a href="#doc-title">Go to the top</a></p>
```

This is the document title

[Go to the top](#)

- [class attribute](#)
- [id attribute](#)

Character Entities (part 1)

The web can accommodate pretty much all characters in all languages around the world. However browsers are dependent on the fonts available.

If you know how your computer can create special characters you can use that to generate the Unicode characters for those characters.

However older browsers need a different way to write characters to display: HTML entities. An HTML entity is a string of text that begins with an ampersand (&) and ends with a semicolon (;). Entities are frequently used to display reserved characters (which would otherwise be interpreted as HTML code), and invisible characters (like non-breaking spaces). You can also use them in place of other characters that are difficult to type with a standard keyboard.

HTML defines some entities as part of the HTML specification. The four reserved entities are:

| Character | Entity | Note |
|-----------|--------|---|
| & | & | Interpreted as the beginning of an entity or character reference. |
| < | < | Interpreted as the beginning of a tag |
| > | > | Interpreted as the ending of a tag |
| " | " | Interpreted as the beginning and end of an attribute's value. |

CSS

Cascading style sheets (CSS) provide a way to style the content of our HTML documents without adding elements and attributes beyond classes and IDs.

The basic syntax for CSS looks like this:


```
elementName {  
  ruleName: RuleValue;  
}
```

Given that basic syntax there are three basic ways to associate CSS to elements in our web page

Targeting specific elements.

The easiest way to use CSS is to target specific HTML elements and style them as needed. In the example below we style paragraphs (represented by the p element) with font family, size and line height information.

We also style the blockquote elements left and right margin.

```
p {  
  font-family: Roboto, Verdana, Arial;  
  font-size: 1em;  
  line-height: 1.25;  
}  
  
blockquote {  
  margin-left: 2em;  
  margin-right: 2em;  
}
```

Using class and id attributes from our HTML content.

The first example takes a class indicated by .myClass and gives it a green color.

The second example uses an ID, defined as #myID and assigns a different color.

The difference between classes and IDs is that IDs are unique to a document while classes can match multiple elements in the same document.

```
.myClass {  
  color: #00ff00;  
}  
  
#myID {  
  color: #0000ff;  
}
```

We've figured out how to write HTML but that introduces the following issue. Let's say that we have two different rules defined for the same element.

```
p {  
  font-family: Roboto, Verdana, Arial;  
  font-size: 1em;  
  line-height: 1.25;  
}  
  
p {  
  font-family: Roboto, Verdana, Arial;  
  font-size: 0.75em;  
  line-height: 1.25;  
}
```

Which definition will the browser use?

To answer this question we need to work with two concepts

description of the CSS cascade using marine life and analogies

Figure 3: CSS speciFISHity from [Estelle Weyl](#)

- How do you write CSS
- The cascade and specificity
- naming conventions
- accessibility considerations
- DRY

Javascript

- naming conventions
- commenting

- DRY
- Some things that you may have heard before
 - Frameworks
 - ES6 or ES2015
 - Progressive Web Applications (PWAs)

How do we use each of the basic components

- HTML for structure and semantics (incorporating accessibility)
- CSS for styling the content
- Javascript for interactivity and behaviors
- Accessibility should not be an afterthought

Accessibility

Semantics

Assistive Technology Affordances

[alt attribute](#) for images

VTT Transcripts and track element for video

Using ARIA

Accessibility is one of the most important aspects of development and one that we don't pay as much attention as we should.

We will look at ARIA (Accessible Rich Internet Applications), what it is and how we can use it in our content to help improve the accessibility of web applications and pages.

We'll also look at ARIA best practices and authoring guidelines. These practices to didn't .

As a last step we'll take page we've retrofitted and have it read by Voice Over, the screen reader bundled as part of MacOS X. This will give us an idea of what a user with visual disabilities experiences when they read the content. Yes, I know... This is all Mac. In my defense, that's what I have and what I use the most. If anyone wants to collaborate with a Windows Version, please ping me on Twitter.

- ARIA
 - [Using ARIA](#)
 - [WAI-ARIA Authoring Practices 1.1](#)
 - [MIND Patterns: Accessibility Patterns for the Web](#)
- Inert Polyfill
 - [Github Repository](#)
- Voice Over
 - [Voice Over Getting Started Guide](#)
 - [Voice Over Commands](#)
 - Standard HTML attributes and how they impact accessibility