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. It is not a prescriptive guide telling you what you must do but the product of 20 years of craating web content

History

The world wide web, or WWW, was created as a method to navigate the now extensive system of connected computers. Tim Berners-Lee, a contractor with the European Organization for Nuclear Research (CERN), developed a rudimentary hypertext program called ENQUIRE.

The program was designed to make information readily available to users, and to allow a user to explore relationships between different pages (i.e. clicking to get to a different section of a website). By 1990 Berners-Lee developed the skeletal outline of the internet, including a web browser and web server. This coincided with companies like AOL, Compuserve and GEnie opening their networks to the Internet.

[Screenshot of the updated line-mode browser at CERN]

The fundamental technologies for the web were (and still are):

HTML: HyperText Markup Language. The markup (formatting) language for the web.

URI: Uniform Resource Identifier. A kind of "address" that is unique and used to identify to each resource on the web. It is also commonly called a **URL** (Uniform Resource Locator)

HTTP: Hypertext Transfer Protocol. Allows for the retrieval of linked resources from across the web.

In the beginning the web was a series of text documents, difficult to navigate, and inaccessible to most people (the early ARPANET was not public and it had few nodes at military and academic institutions). But all that changed in 1993, with the release of the Mosaic web browser, which allowed users to explore multimedia online. This was also the time when companies like Compuserve, AOL and GEnie opened their services to the Internet and began providing web access

[Mosaic screenshot] [Early Netscape screenshot]

1993 also saw the introduction of the first modern search engines. Though early search engines were primitive, manual, and primarily only indexed titles and headers, in 1994 WebCrawler began to “crawl” the net, indexing entire pages of active websites. This technology opened the door for more powerful search engines, and made it possible to search through large amounts of connected information.

[Screenshots of search engines]

In this same year, Berners-Lee founded the world wide web Consortium (W3C) to help further develop ease of use and accessibility of the web, and made it a standard that the web should be available to the public for free and with no patent.

Basic Components of the web

We group these componentstogether 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 sue the saame structure and presentation tags.

We'll use the following example throughout our discussion of HTML and its components.

```
<!DOCTYPE html>
<html>
  <head><html>title>Hello!</title>
    <meta charset="utf-8">
 </title> rel="stylesheet" href="styles.css">
    <script src="/script.js" defer><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 non-empty URL potentially surrounded by spaces. 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
- 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 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 does not represent content for 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**

Presentational Markup

These element 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 it's 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 structual element in HTML is the <p> element that represents a paragraph of information.

Preformated 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 comes 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 orginal 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>
```

Note that `cite`, even when

- ol
- ul
- li

- dl
- dt
- dd

- figure
- figcaption
- img

- video
- audio
- track

- div
- span

- a

- em
- strong
- small
- dfn
- abbr
- sub and sup

- i
- b
- u

- br
- wbr

Class and ID attributes

- Class Attribute

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

When specified on HTML elements, the class attribute must have a value that is a set of space-separated tokens representing the various classes that the element belongs to.

Assigning classes to an element affects class matching in selectors in CSS, the getElementsByClassName() method in the DOM, and other such features.

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.

- ID attribute

When specified on HTML elements, the id attribute value must be unique amongst all the IDs in the element's tree and must contain at least one character. The value must not contain any ASCII whitespace.

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.

Accessibility and Assistive Technology acommodations

alt attribute for images

Integrating CSS and JS to our web pages

CSS

- 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)

Accessibility

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 are particularly suited to custom elements and components we create using Polymer or React-based applications.

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

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

Building a simple web page

- Apply everything we've learned so far

Going forward

Progressive Enhancement: No, it doesn’t have to look the same in all browsers

- What it is?
- Why it’s important

Javascript: Beyond the Basics

- Programmatically adding accessibility attributes to your markup
- Where to look for errors
- How to debug
    - Use the console, Luke
    - Introducing dev tools
- Where to look for help

Choosing a stack. How to decide what to study next

- What’s popular
- By Technology
    - 3D / Games
    - Data Visualization

Tooling Best Practices

- Build Systems and Task Runners
    - WebPack
    - Gulp
- Directory and file structure

More HTML Tips, Tricks and Best Practices

- Micro Data and structured Markup

Advanced Topics

Performance

- Move scripts to the bottom of the page
- Compress your images
- Minimize and concatenate your stylesheets and scripts

Advanced examples using Gulp

Advanced examples using WebPack

Pattern Libraries

- What they are?
- What they're good for?

React, Preact, Angular, Vue, oh my! (libraries and frameworks)

- What they are?
- What they're good for?

Web Components

- What they are?
- What they're good for?