# Lecture #10: Intro to HTML

*CS106E Spring 2018, Young*

*In this lecture we learn the basics of HyperText Markup Language (HTML). We begin by seeing how HTML fits in with the rest of web technologies. We then learn about the key constructs in HTML: elements, tags, and attributes.*

*We see that on modern webpages HTML largely focuses on semantics, whereas Cascading Style Sheets (a second language we will study in depth next lecture) is used to provide presentation information. We also explore why WYSIWYG (What You See is What You Get) doesn't work well for the web.*

*We review versions and variants of HTML and along the way, we discover that politics has led to competing versions of the HTML specification. We briefly review XML, a meta-language for defining HTML-like languages, which has also lead to some variations in how HTML is written.*

*We end with a discussion on the mechanics of creating and testing webpages and a quick look at how to actually get a webpage up on the web.*

**How the Web Works**

What happens when I try to read a webpage?

- I enter a ***URL*** (Uniform Resource Locator) into my web browser such as:

  `http://web.stanford.edu/class/cs106e/`

- The web uses a ***Client-Server Architecture***
  - Applications using this architecture include two distinct types of computers. A set of server computers that wait for and fulfill requests and a set of client computers that make requests of the servers.
  - In technical terms, my computer, which requests webpages, is called the client computer or web client. We can also refer to the web browser program as a web client.

- My computer uses the ***HTTP (HyperText Transport Protocol)*** to communicate with the web server identified in the URL.
- The web server responds with the resource I've requested.
- The resource returned may include references to other resources, and my client computer may make additional requests for different resources to the server using HTTP.
- My web browser will use the resources returned in order to determine what to display to me.

- HTTP is a general-purpose protocol that may be used to send any type of digital data over the Internet. Common file types sent via HTTP include:

HTML (HyperText Markup Language) files – these files contain text along with semantic information for the web browser to use. We'll be studying HTML in much more detail for the rest of this week.

CSS (Cascading Style Sheet) files – these files contain formatting information telling the web browser how to display the webpage. We'll be looking at CSS in detail starting next lecture.

Image Files – these include the JPEG, PNG, and GIF formats we talked about earlier in the quarter.

- We can think of web browsers as actually handling several different tasks:
  - They understand the HTTP protocol and can make requests to web servers for resources.
  - They understand HTML and CSS and can display webpages using the formatting information contained in these files.
  - We'll also discover in a few weeks that they contain a JavaScript interpreter which they use to execute client-side programs.

**HyperText Markup Language (HTML)**
- The most important type of files sent using HTTP are HTML files.
- Let's first break down what the words in the name HyperText Markup Language mean:

***Hypertext*** is the concept that we can link text and other resources into a web of information.

- ▪ With a traditional book, a page is preceded by one page and succeeded by another page. This leads to a single linear ordering of pages.
- ▪ With computers, we are no longer limited to a single linear ordering — we can instead link any piece of information to any other piece of information.
- ▪ Probably the first widespread consumer use of hypertext was when encyclopedias began appearing on CD-ROM discs for use on personal computers.
- ▪ You can think of think of the World Wide Web as combining the concept of Hypertext with the Internet.
- ▪ The ability to link to non-text items, such as sounds and videos, is sometimes referred to as Hypermedia.

The term ***Markup*** comes from publishing. A journalist might write a story for a newspaper and hand that story to their editor. The editor might then *markup* the original article by adding notes on what fonts to use, when to use bold or italic, or provide other formatting information. The article would then be passed to a typesetter who would use those markup notes in order to create the actual printing blocks used for the printer.

Finally, we can think of HTML as a ***Language***. As with human languages, there are specific syntax rules and a given vocabulary.

- HTML allows us to take an unformatted text file and provide markup information for the web browser, telling it how to display our text for the user.

**Tags and Elements**

- A human editor might simply scribble some markup notes on a journalist's original typewriter-written article and expect that a human typesetter would understand the notes. We can't do that with a computer.
- We need a formal way to markup our text that is easy for the computer to understand.

- In HTML we do this by adding tags to our text.
    - o Suppose we have the text:

      ```
      Go Stanford Cardinal
      ```

      and we want the word Stanford to appear in bold and Cardinal to appear in italics.

    - o Using HTML we tell the web browser how we want the text to appear like this:

      ```
      Go <b>Stanford</b> <i>Cardinal</i>
      ```

- Tags are indicated by using the '<' and '>' characters.
- Typically (but not always) they will work in pairs with a *start tag* and a corresponding *end tag*, where the end tag is indicated by using the back slash '/'.

- We can either talk about tags or elements.
    - o Here I've used the <b> bold tag and the <i> italic tag to tell the web browser how I want my text to appear.
    - o An element consists of the start tag, the end tag, and whatever is contained between them. So instead of talking about the tags I could alternatively say, I have a b bold element and an i italic element.
    - o You can think of an element as being a container, starting with the start tag and ending with the end tag. This analogy will prove fruitful both when we look at some of the rules for tags and when we study CSS next lecture.

**Attributes**

- Sometimes we need to provide additional information to a tag in order for the web browser to know exactly what to do with a tag.

- For example, the <a> anchor tag can be used to link two different webpages. But it's not enough to simply use an <a> by itself:

  ```
  Go <a>Stanford</a> Cardinal
  ```

  indicates that Stanford is linked, but doesn't say where it is linked to.
- We provide this additional information by adding attributes and values to the start tag. Here we've linked the text Stanford to the actual Stanford homepage.

  ```
  Go <a href="http://www.stanford.edu/">Stanford</a>
  Cardinal
  ```

  here href is an ***attribute*** (href stands for Hypertext Reference). The <a> anchor tag takes a href attribute, where the ***value*** of the href attribute is the URL we are linking to.

**Learning HTML**

- Most of learning HTML consists of learning the different tags that are available and then finding out which of them have attributes and what those attributes do.
- You can find quite a number of general HTML tag references online. Here's a reasonable one:

  https://www.w3schools.com/tags/default.asp

- We'll assume that you look up tags as needed for the HW assignments, as we will not necessarily be explicitly covering in lecture all the tags you'll need for your HW.
- Here are a few tags to get you started:

  <h1>, <h2>, <h3>, …, </h6> – These six different tags are used to create headings in your webpage. <h1> should indicate your largest and most important heading and <h6> should be used to indicate headings for your least important subsections.

  <p> – You'll need to explicitly indicate paragraphs in HTML. Any carriage returns or blank line in your HTML source file will not be shown when your webpage is displayed in a web browser.

**HTML and CSS / Semantics vs. Presentation**

- Originally, HTML provided all formatting information for a webpage. However, on a modern webpage, HTML works in conjunction with a second language called CSS (Cascading Style Sheets).
- HTML focuses on providing ***semantic*** information about what's on a webpage. For example, which text is a paragraph vs. which text is actually a topic heading.
- CSS focuses on the ***presentation*** of information. For example, should a paragraph be displayed in 12-point Helvetica font or 10-point Times font; or should a topic heading be displayed in bold or should it be underlined.

- Most non-semantic concepts in HTML have been removed over the last two versions (HTML4 and HTML5). However, arguably some of the current elements, including the bold and italic elements I just used above, provide presentation instructions rather than giving semantic information.
  - Using HTML that is strictly semantic is sometimes referred to as *Plain-Old Semantic HTML* or *POSH*. The simpler term Semantic HTML is also sometimes used.
  - If you want your HTML to focus on being Semantic, but you still need to emphasize something HTML provides the <em> (for emphasis) tag, which fulfills a similar purpose to the <i> tag, but doesn't actually tell the web browser that it must be italicized.
  - Similarly, there is a <strong> tag that matches with the <b> bold tag.
  - On default settings and with no additional instructions from the webpage author, <em> will act exactly the same as <i> and <strong> will do the same thing as the <b> tag.

**Some HTML Rules**

Let's go over some basic HTML rules.

- HTML elements must be completely contained within each other.

```
<b><i>Stanford</i> Cardinal<b>
```

  Here the i element is completely contained within the b element, this is legal. In this next example however, neither the i element contains the b element, nor the b element contains the i element, this makes it illegal:

```
<i><b>Stanford</i> Cardinal</b>
```

- Attributes may appear in any order. In addition, in the variant of HTML I am using (more on that later), HTML tag names and attribute names should appear in lower-case and attribute values should be enclosed in quotes. Here is an example using the <img> image tag (we'll take a closer look at how this tag works next lecture:

  ```
  <img src="quad.jpg" alt="The Stanford Quad" />
  ```

  This can be rewritten with the order of the attributes reversed:

  ```
  <img alt="The Stanford Quad" src="quad.jpg" />
  ```

- In the variant of HTML I use, tags which do not have matching end tags such as the <img> tag and the line break tag <br> are identified with a closing backslash. It doesn't make sense to have a line break start tag and a line break end tag – what would be contained between the start of a line break and the end of the line break? I write these tags like this:

  ```
  <br />
  ```

- With HTML5 the rules for which tags can contain which other tags is actually rather complex and can only be determined by reading the official specifications. However, some rules from earlier versions of HTML can provide some good guidelines.

  - We can divide HTML tags into several categories. These include:

    *text-level tags* that are designed to affect individual words or short phrases. The <b> and <i> tags are examples of text-level tags.

    *block-level tags* that are designed to create large blocks within our document. The <h1>, … , <h6> heading tags and the <p> paragraph tags are examples of block-level tags.

  - Block-level tags may contain text-level tags, but text-level tags may not contain block-level tags. So:

    ```
    <p>I go to <b>Stanford</b>.</p>
    ```

    is legal. However:

    ```
    <b><p>I go to Stanford.</p></b>
    ```

    is not, because the text-level b bold element cannot contain the block-level p paragraph element.

  - Block-level elements *may or may not* contain other block-level elements, you'll need to learn these on a case-by-case basis. For example:
    - The heading elements may not contain other block-level elements, so you can't put a paragraph inside a heading, nor may you place an h2 heading within an h1 heading.
    - The paragraph element cannot contain other block-level elements, so you couldn't put a paragraph inside another paragraph.
    - In contrast the td table data element which defines the cell of a table can contain block-level elements. So a table cell could contain several paragraphs or even a heading such as h3 along with some paragraphs.

**Overall File Structure**
HTML files are divided into two distinct parts:

The *head* contains information about the webpage. This includes the character encoding to use (such as Unicode utf-8 or Japanese SHIFT-JIS), Cascading Style Sheet CSS formatting information, and the title used to label the web browser tab.

The *body* contains the actual contents that will be displayed on the webpage. Everything we've gone over in this handout prior to this point goes into the body.

Here's what the overall structure of your HTML file should look like:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<title> … </title>
…
</head>
<body>
…
</body>
</html>
```

- The DOCTYPE identifies our document as HTML and allows us to validate our webpage (see section on validation below). Note that this tag does not follow the normal rules on how tags work.
- The lang="en" attribute value pair in the html tag specifies that our webpage is in English.
- The meta tag identifies the character set the web browser should use when displaying the webpage. It's listed first, as the other elements such as the title might include foreign-language characters. This tag can be used for other purposes as well, for example, forcing the webpage to reload after a set amount of time (as is done on many news websites).
- The text between the title start and end tags will appear on the web browser tab and will also be the default text used if the user creates a bookmark for the webpage.
- We'll see some additional tags that belong in the head section in future lectures.
- As I've previously mentioned, the body contains the actual contents to display in the webpage.

**What You See is Not Necessarily What You Get**
One very important innovation in computer applications was the development of ***WYSIWYG*** "What You See is What You Get" applications. Prior to this, creating a document with a word processor might actually look rather like writing HTML, with specific character codes used to indicate formatting. Now, when we're editing something in Word, we're actually seeing almost exactly what will appear on the printed page. So why don't we use Word to create our webpages?

When we're creating a document in Word, we generally know exactly what sort of media we intend to output to. I can choose to format a Word document for 8.5" by 11" paper printed in portrait mode, or I can choose to format my document to print out in Legal 8.5" by 14" paper printed in landscape mode. Regardless of the actual paper used, it is my choice as the author of the document.

On the web, I don't control the device my webpage will be displayed on. It could be a 24" 2560x1440 pixel monitor. It could be an Amazon 7" 1024x600 Kids tablet. It could be a massive 72" 3840x2160 pixel TV supporting HDR10 colors. What's more, even on the same computer, different web browsers will display HTML differently – how a webpage appears in Safari and Chrome might be different even on the same computer.

To quote the official W3C (World Wide Web Consortium) validation website: WYSINWOG "What You See Is Not What Others Get".[1] In other words, what we're seeing on our screen when we view our own webpage is not necessarily what others are going to see when they view our webpage.

So what do we do? We choose a specific range of screen resolutions and color depths we are going to support. We think about what sort of devices we're going to support (web designs that work well with desktops and laptops, for example, don't necessarily work well for touch screen devices without a mouse). In addition, we think about which versions of which web browsers we want to make sure our website works with. Then we test the heck out of everything on multiple platforms and multiple web browsers.

In addition, we make sure our webpage validates.

**Validation**

- What happens if we violate the rules of HTML? When we load our illegal webpage in the web browser, do we get a big red flashing "WARNING" indicator?
- Web browsers are primarily designed for consumers to view webpages that other people have written. As such, unless the browser happens to see something that might cause serious security concerns, it just ignores any problems and attempts to render the webpage as best it can.

- As webpage authors we want to make sure our HTML is legal. If we do something illegal, one web browser (say for example Safari) might try to correct our error in one way and another web browser (say perhaps Chrome) might try to correct our error in another way. So illegal HTML leads to lots of potential problems that are hard to identify.

- An HTML validator is a program that checks our HTML to make sure that it follows the rules. You should make sure to run all your HTML files through the official W3C validator which can be found here:

  https://validator.w3.org/

**HTML5?**

- The current version of HTML is called ***HTML5***.
  - In addition to including what we typically think of as HTML (namely the actual markup language and accompanying tags and attributes) HTML5 also includes specification for some programming mechanisms web browsers are asked to implement. These include:

    Geolocation – The ability to ask a web browser for its location including its latitude and longitude and optionally additional properties such as the altitude, heading, and velocity.

    Canvas – The ability to create a canvas for drawing along with a specification of functions to drawing on the canvas.

    Web Storage – A specification for storing data locally on the web browser that can be accessed by client-side programs running within the web browser (this provides similar functionality to cookies – a topic we will cover next week).

- In casual speech, you may hear people refer to HTML5 meaning not specifically just the latest version of the HTML standard, but also the whole set of technologies that are typically used in

[1] I've never seen this acronym used elsewhere, but it really does emphasize that on the web the traditional WYSIWYG "What You See is What You Get" doesn't hold.

conjunction with HTML5 such as CSS and JavaScript. If someone says they're building an HTML5 website, they don't just mean that they are using HTML5, they mean they're building a modern website with the latest in web technology.

- Unfortunately there are two competing groups responsible for defining HTML5
  - The ***World Wide Web Consortium (W3C)*** is responsible for web-related specifications. Their HTML committee has completed specification of HTML5.2 and is working on HTML5.3.
  - The *WHATWG* (Web Hypertext Application Technology Working Group) is primarily composed of web browser developers. While the W3C and WHATWG both agreed on the HTML5 specification they have different design philosophies.
    - As web browser developers, WHATWG is not happy with the concept of specific versions of HTML that their web browsers must support, instead they prefer to think of the HTML specification as "A Living Standard" that will change over time.
    - WHATWG philosophy is that HTML will no longer be numbered or have specific versions.

**XML**

- The concept of creating a language that has tags and elements, attributes and values, has utility outside of just HTML.
- In order to capitalize on the popularity of this technique ***XML*** (Extensible Markup Language) was defined. XML is really less of a language and more of a meta-language specifying rules that markup languages should follow in order to be supported by XML tools.

- Following the specification of XML, a number of XML-based markup languages have been developed. These include CML (Chemical Markup Language) in which tags and attributes define atoms, molecules, and bonds, MathML (Math Markup Language) used for defining mathematical equations, and ***SVG (Scalable Vector Graphics)***.

- SVG allows us to define object-based drawings and is widely supported by modern web browsers. Here is an example of a diagram defined in SVG:

```
<circle cx="200" cy="200" r="100" fill="red" />
<rect x="100" y="100" width="200" height="200"
           stroke="blue" stroke-width="5"
        fill="none" />
<text x="20" y="30" fill="red"
       font-weight="bold" font-size="24pt">
                 Go Stanford</text>
```

- XML follows similar rules to HTML, but it has more restrictive syntax rules. These include:
  - tag and attribute names must be in lower-case
  - attribute values must be enclosed in quotes
  - all tags must work in pairs, if there are no contents between a start and end tag, you may just write the start tag, but end it with a '/' (see for example the <circle> tag in the SVG snippet above).

- If these rules sound very familiar, it's because the variant of HTML I use is based on XHTML, which is a variant of HTML designed to follow XML's more restrictive rules.
  - In standard HTML, some tags don't need end tags. If an end tag can be deduced it can be skipped, for example, a <p> paragraph ends when the next <p> tag is encountered, no </p> is necessary.
  - Similarly, if an attribute value is simply a single word or number, no quotes are necessary.

    - Also the case of tags is not specified, I could write <body>, <BODY>, or even <BoDy>

**How to Create a Webpage**

- You'll need to get a text editor to write webpages in HTML. Note that a text editor is different from a Word Processor such as Microsoft Word. Text editors are designed to work with raw unformatted text.
    - I use Notepad++ on Windows. The standard Notepad will work, but Notepad++ has additional features such as coloring tag names and attributes differently than other text.
    - While TextEdit on the Macintosh can work as a text editor, it also has a WYSIWYG (What You See Is What You Get) mode that can cause problems for unwary users trying to create HTML files with it. Therefore, I don't recommend using it.
    - TextWrangler and BBEdit are good choices for text editors on the Macintosh.
    - Sublime is a popular text editor available on Windows, Mac, and Linux.
    - Atom is another popular text editor that works on Windows, Mac, and Linux.

- Rather than creating a new HTML file from scratch, just copy an existing template. We'll provide you with a "starter.html" file that is essentially the same as the HTML shown in the "Overall File Structure" section of these notes. Start with that and fill in the details for your webpage.
- Go to the W3C's validator website and see if your HTML is correct:

  https://validator.w3.org/

- Once you've gotten rid of any errors identified by the validator, we're ready to see what our webpage looks like in a web browser.
    - You can load HTML files directly from the SSD or Hard Drive on your personal computer into a web browser running on your computer. You do not have to place your HTML file onto an actual web server to practice writing HTML.
    - I generally just drag and drop html files from my folders in Windows into the web browser windows. However, most web browsers also have "File" menu items allowing you to choose files to open from your drive.
- If your HTML file does not appear correctly in the web browser, think about what's actually appearing in the web browser and go back to your HTML source file and find what might be causing problems.

**Putting a Webpage on the Web**

If you want other people to be able to see your webpage, you'll need to put it on an actual server. Do make sure you've thoroughly validated and tested it locally (see previous section) before sticking it up where the public can see it.

At Stanford, you have a special directory called WWW in your online Stanford disk space. If you copy a file into the WWW directory it is now on the web. The URL for a file named example.html placed in your WWW directory will be:

`http://web.stanford.edu/~`*`yourSUNetID`*`/example.html`

where yourSUNetID is replaced by your 3-8 letter Stanford University Network ID. Do make sure to include the tilde in the actual URL though '~'.

To prevent problems, I recommend that you name all files you plan to use on the web using lower-case letters, numbers, and the dash '-'. Don't use upper-case letters and definitely don't use any symbols other than the dash and of course the period used for .html or .css.