HTML5

Learning Sources
MDN Web Docs – HTML5 ← Primary
HTML 5.2 Docs – W3 ← Reference

How the Net Works & Intro - Odin
https://www.theodinproject.com/courses/web-development-101/lessons/how-does-the-web-work

-Teach others as a way to make learned knowledge more concrete. Teach a pillow, even (tfw).

-Protocols – Ex. TCP (Transmission Control Protocol) determine how computers on the net communicate with one another
-Packets – contain data (split up into many packets), metadata, where it is coming from, going to, etc.
-Routers – computers connect to. Router guides packets through the net via ip addr, DNS, etc. from sender to receiver
-Modem – converts data coming from/to computers into signals transferable via telephone, coax, etc.
-ISP – Takes in data from clients and sends it to other routers, which reply with data that is sent back to the client (end user)
-IP – unique identifier for computer on the net (ex. Website server). Ex. 173.194.121.32
-Domain name – linked to IP (ex. Qwant.com) to make better for human memory & branding
-DNS – Domain Name Servers – Look up domain name and connect to IP so it can properly send http messages
-HTTP – Hypertext Transfer Protocol – defines language to allow clients and servers to communicate
-Component Files – code files (HTML, JS, etc.) and assets (images, music, video, pdf, etc.)

Website vs web page
-web page = document that can be displayed on browser. Website = collection of web pages grouped together with a unique domain name.

Basic web browser comm steps
-Browser connects to dns server for IP, sends HTTP request to server IP using TCP/IP. Server sends a “200 OK” message, then sends response packets. Browser assembles packages and displays for client. ISP in between client & servers site is hosted on.

Elements
-Hypertext Markup Language

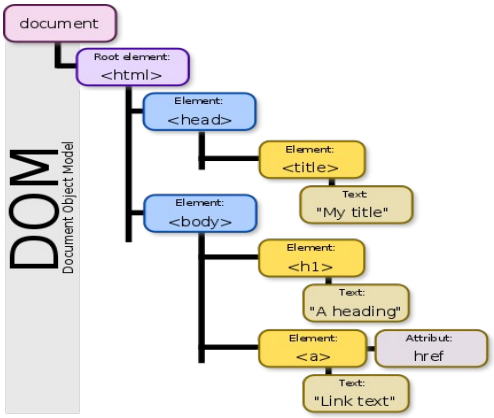
-Enclose, wrap, or *mark up* different parts of the content in HTML doc to make it appear or act a certain way
    -ex. * * - paragraph element with opening (* *) and closing (* *) tags

-Elements can be nested, but nested inner must close before outer closes
    -ex * My cat is very grumpy. *

-Elements create hierarchy trees where nested element(s) are children, grandchildren, etc. of

parent(s), grandparent(s), etc.. ex. <*ul*> inside <*p*> inside <*body*>. Children can inherit behavior & styling from parents, grandparents, etc.

-Structure of HTML file creates the **DOM** (Document Object Model, a language-independent interface for how to treat the structure of HTML, XML, and XHTML documents. In the DOM the document is treated as a tree structure.



Two types of elements
a) block-level – form a visible block on a page appearing after new line, and followed by new line (paragraphs, lists, etc.).

b) inline – contained w/i block level (ex. hyperlink, italics, etc.)
-Self-closing elements = elements which can open and close is one tag
-ex. <*img src = url* >

Attributes
-Expand on an element by adding further markup, identification, semantics, etc
-ex. <p *class=”editor-note”*></p> -ex. <*img src="url"*>

-Syntax: <*element attributeName=”value”*>

<a> element and attributes
-<*a*> - anchor element – makes a piece of text wrap around, usually to make a hyperlink
-*href* – attribute that links to specified web addr
-*title* – attribute that creates title to show more info on when hovered over
-*target* – browsing context for new link (ex. open in new tab: target=”_blank”)

-<*h1*> to <*h6*> - bold, large font header elements, in various sizes w/ <*h1*> being largest font

-Element can have multiple attributes: <*elmX attributeName=”value”, attributeName=”value”, ...> </element*>
-ex. <*a href="https://4chan.org" title=4chan target=_blank">4chan</a*>

-boolean attributes can exist without attribute value following attr name
-ex. <*input type=”text” disabled*>

Note on Naming Conventions
-In CSS, class names are seperated by dashes (*class-name*) so for consistancy, use this in HTML identifiers as well. Consider just lowercase ids as well, for typing speed and uniformity with common standards.

Anatomy of a HTML Doc
-*<!DOCTYPE html>* - at top of page & declares file version

-*<html></html>* - open & close HTML doc with all html code goes inside

-*<head></head>* - contains metadata. All the content not showing to viewers (keywords, page description for search results, CSS to style content, etc.) for SEO, etc. use.

-*<meta charset="utf-8">* - sets to use massive utf-8 character set, which contains most human lang chars. Nested in head.

-*<title></title>* - Title of page displayed in browser tab & used when bookmarking. Nested in *<head>*.

-*<body></body>* - Content users see on site. Text images, videos, games, audio, etc.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>My test page</title>
  </head>
  <body>
    <p>This is my page</p>
  </body>
</html>
```

Special Chars
-Some common special chars

<	<
>	>
"	"
'	'
&	&

-Also: *(* is *(* *)* is *)*
-ex. * In HTML, you define a paragraph using the <p> element. *

Comments
<!-- comment -->
-ex. <!-- this is a comment -->

Head Elements
-*<title>pageTitle</title>* - title for the html doc. Displays in browser tab.

Meta elements – metadata elements
-*<meta charset="utf-8">* - specifies doc encoding (utf-8 covers all langs)

-Many meta elements contain *name* and *content attributes*, where *name* describes what the metadata

is and the *content* lists the actual metadata content. The following ex should be included in all pages:

-<*meta name=”author” content=”Dick Leiptz”*>

-<*meta name=”description”=”Website is for buying bulk dragon dildos.”*>

-Facebook created the “Open Graph Data” metadata protocol for what data to provide. See “The Open Graph protocol.pdf” for details.

-favicon – adds favorites icon to you site. .ico format. Many different types, with different size, etc.

-ex. <*link rel=”shortcut icon” href=”icoName.ico” type=”image/x-icon”*>

-Primary language - <*html lang=”en-US”*> - set in opening <html> element

Javascript and CSS also applied to page in the head

-<*link*> - element allows to link document to outside sources (ex. .css stylsheet)

-syntax: <*link rel=”stylesheet” href=”cssFilname.css”*>

-<*script*> - element to run an external script. Should usually be placed just before closing <*body*>, to ensure all html data has been read prior to running

-syntax: <*script src=”jsFilename.js”></script*>

Lists

-Unordered – have dots - <*ul></ul*>
-Ordered – numbered - <*ol></ol*>
-List items - <*li></li*>

-ex. <*ul*>
<*li*>*eggs*</*li*>
<*li*>*milk*</*li*>
</*ul*>

-Can nest lists

Bold & Italics

-emphasis (italics) - <*em></em*> - use on stressed words
-strong word (bold) - - use on words that bear strong importance

-bold - <*b></b*> - key words, product names, lead scentence
-italics - <*i></i*> - foreign words, taxonomic design, tech terms, a thought
-underline - <*u></u*>- proper name, misspelling
-<b><i><u> often not used anymore as replaced by styling in CSS

Hyperlink Basics

-<*a href=”linkURL” title=”hoverTitle*> *Text you are linking* </*a*>

-Block level elements can also be turned into links

-ex
```
<a href=”linkURL”>
    <img src=”imageLocation” alt=”hoverTitle”>
</a>
```

-Local file link ex. (<*a href=contacts.html*>)
-Follow same directory path structure as in unix, including ., .., /, ./, etc.

-ex. *<a href="subdir1/page.html"* -ex. *<a href= ../subdir2/page1.html">*

Linking to Document Fragments
-Document fragments are named portions of a site that can be linked to

-Done through *id* attribute, which gives an element an unique id
-ex. *<h2 id="Mailing_address">Mailing address</h2>*
-Then, once *id* created, reference via *docName.html#idName*
-ex. * Contact us on our Address Pg *

Absolute vs Relative URLs
-Absolute – Always points to same location. Ex. https://example.com/files/pdfs/pdf-read.pdf

-Relative – Don't need to include the full filepath. Depends on where linked file is located in relation to file linking from.
Example: pdf-list.html is in folder /example/files/pdfs. Relative link calling pdf-read.pdf in same folder would be: *href=pdf-read.pdf*, not *href=example/files/pdfs/pdf-read.pdf*

Link Best Practices
-Linked text should be short, concise, and include main keywords only (ex. *Download PDF*)

-Prefer relative links over absolute as they are shorter and more server efficient.

-Leave details to know when linking to external files (ex. "Download sales report (PDF, 10mb)," "Play car game (requires flash)"

-Use *download* attribute when linking to external file for default save name. ex. *download="firefox-installer-64bit-v56.exe"*

Email Links
-URL formatting to allow outgoing mail links to pop up in email managers, etc.
-href="mailto:email@email.com
-Can also set other inputs like cc, bcc, subject, and body.
-Format: *"mailto:email@email.com?cc=xyz&bcc=xyz&subject=some%20words&body=the %20words*
-Can also mail to multiple emails, as in first example, separating emails with a comma

Description Lists
-Used for a set of items and their associated descriptions (ex. word and definition)

-ex. **aside**
In drama, where a character speaks their thoughts out loud to share them with the...
soliloquy
In drama, where a character speaks to themselves, representing their inner thoughts…

<dl>
<dt>description term</dt>
<dd>description definition</dd>
<dd>description definition</dd> //optional n more definitions
<dl>

Citations

-Used when a block element (ex. paragraph) or part of that block, via inline element (ex. sentence) is quoted from another site, cite using quotations, for reference in source code

-If citing a block element, wrap it in a blockquote. Blockquote will indent the block.

```
<blockquote cite="https://citation-url">
    //block element here
</blockquote>
```

-If citing an inline element, wrap in <q> (quote) element. <q> will add quotes around.

```
<p>The xyz element does <q cite="https://citation-url">list of things </q></p>
```

Abbreviations

-Inline element that creates a hover tool-tip over what it wraps

-ex. *<abbr title="Hypertext Markup Language">HTML</abbr>*

Subscript & Superscript

-numbers smaller or larger than the rest of the text (ex. $C_8H_{10}N_4O_2$, 25^{th})

-subscript - **

-superscript - **

Representing Computer Code

-*<code></code>* - general pieces of code

-*<pre></pre>* - for preserving whitespace (indentation, etc.). Should wrap *<code>*.

-*<var></var>* - mark variable names

-<kbd></kbd> - mark keyboard input

-*<samp></samp>* - output

Marking Dates & Times

-*<time datetime="yyyy-mm-dd">The date in english</time>*

-ex. *<time datetime="2016-01-20">20 January 2016</time>*

-Provides machine readable version of the marked date time, for scripting, etc. purposes

-Can set other attribute values besides *"yyyy-mm-dd"* such as *"mm-dd" "hh:mm:ss.ms"* etc.

Basics Structure Sections of a Doc

-Header – *<header>* - Usually a big strip across the top w/ big heading/logo. Stays consistent across pages.

-Navigation bar – *<nav>* - Under header. Horizontal. Links to site's main sections via menu, links, tabs, etc. Consistent across pages.

-Main content – *<main>* - Center. Bulk of site. Subjections via *<article>*, *<section>*, & * *. Unique to page. Inside *<body>*. Prefer don't nest w/i other elements.

-Sidebar – *<aside>* (often placed inside *<main>*) - Some links, quotes, ads, secondary nav, etc.. Content related to main content.

-Footer – *<footer>* - Strip on bottom. Fine prints, contact info, etc. End content.

-*<article>* - a block of content that makes sense on its own (ex. a single blog post). Can be sub-dived into different sections/articles. Should have header. Can be sub-dived further.

-*<section>* - similar to *<article>*, but intended for single functionality (ex. mini map, set of headlines). Can be sub-dived further.

-Non-semenatic wrappers: * & * - Used when content doesn't fit into above wrappers. Use with class attribute to ID them via *class="class-name"*.

-<*span*> - inline element for ambiguous content that doesn't fit into above. Ex. editor's note.
-<*div*> - block level. ex. shopping cart widget. User semantic wrappers if possible.

-<*br*> - Break. line break in paragraph
-<*hr*> - horizontal rule (line). Denotes thematic change in text (ex. in topic or scene)

Basic Design
1) Write common to every page elements (ex. header w/ logo & title, footer w/ contact, terms & conditions, site lang, accessibility policy).
2) Draw rough sketch of page layout as would appear on screen
3) Brainstorm content for pages across site, then sort into groups to determine pages
4) Sketch rough sitemap. Ex. Bubbles w/ arrows connecting to show workflow. Main purpose & features of each page, etc.

Debugging
-Can debug in Firefox via "Inspect element"
-Inspector allows you to read code for individual elements, edit the code, and see live changes. Right clicking on element gives you options such as "delete element."

-If unable to find error manually can run through WC3 Markup Validation Service site to display errors in more detail

Images
-Never host an image by linking to URL people will say your code is trash & is not stable
-Syntax: <*img src="imageName.jpg" alt="image description"*>

Attributes
-*alt="description"* - attribute for text to display if image cannot load. Read by screenreader for blind. Uses by SEO (search engine optimizer).
-*width="#OfPixels" height="#OfPixels"*- Avoid altering size if possible as may making pixelated, etc.
-*title="hover title description"*- avoid using due to accessibility issues
-*align= ""* - to align left, center, etc.

-Images are linked in html docs and a space is held open for them. An image can be turned into part of a self contained element by turning into a *figure*, which does not affect the main flow of the doc. Generally used with *figcaptions.*

-<*figure*> - block element. <*img*> would go inside.
-<*figcaption*>*Caption Text*</*figurecaption*> - goes inside <*figure*>. Adds caption at bottom of image. Ex. use: in textbook. By placing inside <*figure*>, link to it and provide better structure

Audio & Video
-Function through <*audio*> and <*video*> elements which are controlled by JavaScript

-<*video src="VideoName.filetype" controls*> - Must either include *controls* or or build own JS API to handle>

-Fallback content – Include <*p*> inside video (<*video*><*p*></*p*></*video*>) with <*a*> to link for video, in case browser does not support HTML5 video, along with description

-Video format support – WebM mostly Firefox & chrome. MP4 mostly IE & Safarit.

-Can specify multiple filetypes via <*source*> element & *type* attribute

```
<video controls>
    <source  src="video.mp4" type="video/mp4">
    <source src="video.webm" type="video/webm>
</video>
```

<*video*> *attributes*
-binary (no definition required) = autoplay, loop, & muted
-width="#OfPixels" height="#OfPixels"
-poster="imageFile.png/jpg/etc" - displays image as splash screen before user clicks play
-preload="option" - Used for buffering. Three options: *"none"* (no buffering), *"auto"* (buffers file), & *"metadata"* (buffers metadata only)

-<*audio*> - Pretty much the same as <*video*>, except no *poster* support.

-Restarting Media Playback – Can restart media playback at any time (including unique <*source*> if multi sources) by creating media variable based on media *id*, then calling *load()* method on it:

```
var mediaElem = document.getElementById("myMediaElement")}}
mediaElem.load();
```

-Detecting Track additional/removal – Can monitor when tracks added or removed to identified elements:

```
var mediaElem = document.querySelector("video");
mediaElem.audioTracks.onaddtrack = function(event) {
  audioTrackAdded(event.track);
}
```

-Subtitles & Video Text – Nest following element inside <video> element
<*track kind="subtitles" src="filename.vtt" srclang=en"*> Same for as .srt files.

Embedding Technologies
-Embedded Java applets and Flash popular in early '00s but not anymore. Many problems with.

-<*iframe*> - inline element allows nested browsing, effectively a html page within the existing html page. Many sites (ex. Gmaps, Youtube, etc.) allow you to "share" a feature via an iframe and will provide you code to embed.

Attributes
-src="urlOfEmbeddedContent"
-allowfullscreen - binary attribute. Allows fullscreen via full screen API
-height="#pixels" width="#pixesls"
-sandbox – binary. Works in more modern browsers. Sandboxes for added security.
-frameborder="0 or 1" - default is 1 which adds border. 0 is (no border). Avoid for css instead.

-Security Issues with iframes – If you embed an iframe in your site and it turns out there is malicious code in it (ex. tracking logins on your site), well...yeah. Hackers also hack sites, then embed iframes in them that don't show on the page (but can see in source) that do malicious actions.

-iframe security best practices - Only embed when necessary. Host via HTTPS. Always use the *sandbox* attribute & if site needs less sandboxing, modify via sandbox="options". Never set *sandbox=""* to both *allow-scripts* and *allow-same-origin* as this could let an attacker us JS to turn

sandboxing off. Set content security policy in header via content security policy (CSP) headers seen in: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy

-<*embed*> and <*object*> - for embedding plugins like Java applets, Flash, etc. Avoid uses since require extensions. Add security risks (Flash is a nightmare)

Vector Graphics
-Raster images – typical images (jpg, png) w/ info showing where each pixel is placed
-Vector images – algorithms showing shape & path defs, allowing scalability without pixalization. Less size than rasters.

-SVG – XML-based lang for creating & describing vector images. Can define shapes (ex. circle), transform colors, animate, etc.

ex. //creates blue circle

```
<svg version="1.1"
    baseProfile="full"
    width="300" height="200"
    mlns="http://www.w3.org/2000/svg">
<circle cx="150" cy="100" r="90" fill="blue" />
</svg>
```

-Usually don't handcode & instead use vector graphics creator (ex. *Adobe Illustrator*)

-Other benefits of: Text remains visible in (SEO can find it). Easily scripted.
-Downsides: Can use more processing power

-Adding via <*img*> - add via *src=”filename.svg”* specifying height & width attr.
-Con: cannot manipulate with JS & requires more work and code inclusion for CSS styling.

-Adding via inline <*svg*> - After create svg file, open in text editor, then paste code in between *<svg width=”#” height=”#”><code /></svg>*.
-Pros: Reduces loading time. Allows classes and ids. Allow CSS interation. Can hyperlink by wrapping in <*a*>.
-Cons: Too heavy for load if many SVGs. Increases HTML file size. No browser cache support.

Responsive Images
-Necessary for sites due to vast difference in screen sizes with smartphone popularity. Having images of various sizes, cropped, etc. in line with screen size changes for both viewability & bandwith.

-Note: CSS has better tools for responsive images

-Always include *<meta name="viewport" content="width=device-width, initial-scale=1.0">* in header to force browsers to adopt their actual viewport width.

Resoultion switching: Different Sizes
-<*srcset*> - provides set of different image sizes for browser to choose from and what img size is
-syntax: *srcset = “filename.jpg 280w, filename2.jpg 480w, filename3 800w”*
//where nums are set to imgs’ inherent widths in pixels

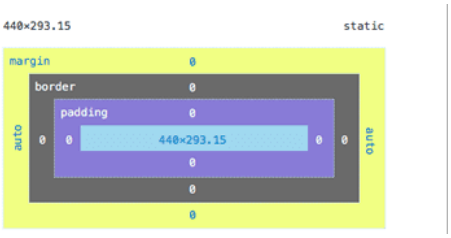
-<*sizes*> - defines set of media conditions (ex. screen width) and which img to use when conditions

met
-syntax: *sizes=”(max-width: 320px) 280px, (max-width: 480px) 440px, 800px”*
//where *max-width* is a media condition (possible state of a screen), the #px value is the width the media will fill when the condition is met, and last px value is “else”

-Ex. *<img srcset=”clouds320.jpg 320w, clouds480.jpg 480w, clouds800.jpg 800w”*
sizes=”(max-width: 320px) 280px, (max-width: 480px) 440px, 800px”
src=”clouds800.jpg” alt=”clouds in the sky”>

Developer Tools
-In firefox: Tools > Web Developer > Responsive Design View – shows web page in various screen sizes. Then can go view Box Model Viewer details to determine sizes to use.

-Padding = area before border
-Margin area = empty space surrounding the border



-Can test if *<srcset>* is working by setting viewport (screen size) width in Responsive View design, then turning on Tools > Web Developer > Network to open net inspector which will list assets page download so you can check that right size img was downloaded.

<picture> - similar how *<video>* can host multi vid formats, *<picture>* allows you to host different images & condition select which is displayed
-syntax: *<picture>*
<source media=”(max-width: 799px)” srcset=”croppedImage.jpg”>
<source media=”(min-width: 800px” srcset=”fullImage.jpg”>
<img src=”fullimage.jpg” alt=”A full image”>
</picture>
<!--If viewpoint smaller than 800px, shows cropped, if >= 800px, shows full-→

-Always compress images to save disk space: https://tinypng.com/

HTML Tables
-HTML tables are for tabular data, not styling. HTML table styling causes numerous problems & CSS has advanced past the need for HTML table styling.

-Syntax:
```
<table>
    <tr>                    <!--table row. First row is usually label-->
        <td> </td>     <!--non breaking space char. ie empty entry-->
        <th>dataX</td>      <!--table header. Stylizes & marks as <th> element-->
        <td>dataY</td>      <!--table data-->
    </tr>
    <tr>
        <th>dataF</td>
        <td>dataJ</td>
<table>
```

-*<th scope="col">* and *<th scope="row">* - define is a header is a row or column. Useful for screenreaders (used by blind).
-If heading sits atop multiple columns or rows (ex. "clothes" heading over "pants" and "shirts"

columns can also set to *="colgroup"* or *="rowgroup")*

-*colspan=”#”* and *rowspan=”#”* - attributes to make row or column span multiple columns/rows

-*<colgroup>* - specifies how to style columns via nested *<col> elements*

-Syntax:

```
<table>
    <colgroup>
        <col>          <!--first column unstyled-->
        <col style=”background-color: yellow>
    </colgroup>
    <tr>
    …
```

-Can apply to # multiple sucessive columns via one *<col>* element with *span=”#”* attr

-*<caption>Text</caption>* - adds caption title above table. Place under *<table>* tag.

Table Structure

-Can break table into unique sections (header, body, footer) to allow easy CSS styling, etc.

-*<thead>* - wraps around part of table to be header (typically first row column headings).
-*<tfoot>* - wraps around part of table to be footer. Ex. a final row with above rows summed
-*<tbody>* - wraps around table that is not head or footer

-Can nest tables by placing new *<table>* inside *<tr>*. Make sure to give tables unique ids to ref.

-Can give unique ids to table entries via *<th id="uniqueID">* and *<td headers="uniqueID">*

Forms 101

-One of basic points of interaction between site and user
-Made of one or more widgets (text fields, select boxes, buttons, check-boxes, radio buttons)
-Widget - a module that allows users to access information or perform a function
-Usually paired with label describing purpose
-Form data usually sent to server

-Suggested to design form layout same way as site layout by sketching, etc. Design Guides:
An Extensive Guide To Web Form Usability
Buttons Best Practices
-Generally include label, input field, actions (buttons, etc), help, messages (feedback based on input...ex. “form submitted”), and validation, to ensure data submitted is acceptable (ex. “email addr not valid)

<form> Element

-Container element (like * *, etc.) that contains a form. Generally has an *action* and *method* attrs
-*action* - defines url for form data to be sent when submitted
-*method* - which HTTP method to send data with (*post* or *get)*

-ex. *<form action=”/my-handling-form-page” method=”post”>*

<label>, <input>, and <textarea> elements

-Contained within *<form>* element

-*<label>* - caption for form element, such as check-box, text box, etc.

-has *for* attribute, which is et to equal *id* of element it is labeling. Ex. *for=”emailBox”*
-<*input*> - specifies type of data being input, min/max length of data being entered, etc.
    -ex. <*input type="text" id="name" name="user_name"*>
-<*textarea*> - multi-line text entry box for user entry. Has # rows/columns to define size.

-Often contain *label* and element label inside <*div*> to make styling easier

<*button*> Element
-<*button type=”buttonType”>Button Label</button*>
-*type* values: *submit, reset* (bad practice), *button* (does nothing...customizable)

Sending Form Data to Server
-Where and how defined by *action* and *method* attributes of form, but each piece of data also has a *name* attribute (ex. *name=”user_email”*) to uniquely id data being sent to server from form, where data is sent as key/value pair

Structuring an HTML Form

<*form*> Element
-Container element (like <*div*>, etc.) that contains a form. Defines attr that determine behavior of form.
-Cannot nest *form* inside *form*

<*fieldset*> and <*legend*> Elements
-<*fieldset*> - wrapper to group together widgets that share the same purpose
-<*legend*> - describes the *fieldset.* Generally right under *fieldset*.
    -Syntax: <*legend*>*labelText*</*legend*>

-ex. <*form*> contains <*fieldset*> w/ <*legend*> label at top, then three <*div*>, where each <*div*> has a *radio* type <*input*> and a <*label*>

<label> element
-defines a label for a form widget
-Syntax: <*label for=”widgetName”*>*labelTex*</*lable*>

-assign to widget via *for* attr, which ref widget *id* (ex. *for=”nameBox”* //where *nameBox* is text input widget*)*
-Clicking on label will activate widget (ex. click on label for *checkbox* will toggle *checkbox*)

-Can nest widget within label (common):
    <*label for=”xyz”*>
        <*span*>*labelText*</*span*>
        <*widgetHere*>*...*</*widgetHere*> *//ex.*<*input*> *w/ type=”email”*
    </*label*>

HTML Form Structure
-Common practice to wrap paired form elements (ex. *label* and *input*) in <*div*> or <*p*>
-Also use common <*section*>, <*h1*>, <*h2*>, etc.

Native Form Widgets

Global Form Widget attr

-*autofocus* - boolean. Gives element auto-focus when page loads. Only one element per page should have this. *False* by default.
-*disabled* - boolean. Data not sent to server. *False* by default.
-*form* - set to *id* of *form* elem widget associated with. Generally not used since widget usually wrapped in *form*.
-*name* - name of element. Unique id for server.
-*value* - elem default value

Text Input Fields
-<*input*> - boxes for user data input.
-Can be general or set *type* attr (ex. *type="email"*) to make specialized

-Common Attr:
    -*readonly* - boolean. user cannot interact with.
    -*placeholder="text"*- shows as semi-transparent text in input box before user enters text
    -Can set size (width and length) with either html or css

-Single line types: *text, password, search, url, tel* (phone num)

-*text* - general text entry
-*email* - error displayed if not valid email format. Can include multi emails seperated w/ comma by adding *multiple* boolean attr to widget.
-*password* - obscures values
-*search* - often given different styling by browser (ex. rounded corners with "x" to clear search). Adds autosave/completion features.
-*tel* - no entry constraints since many phone num formats. Semantic.
-*url* - only accepts valid url format

Multi-Line Text Field
-Syntax: *<textarea cols="#" rows="#"></textarea>*
-*cols* based on char width size
-*rows* based on font size
-*wrap* - attr. Takes *hard* or *soft* value.

-would put default val (if desired) between open/close tag
-plain text input only

Drop-down Content
-Select box: *<select id="..." name="...>* Contains elements: *<option>optionText</option>*
-*selected* – boolean attribute for <*option*>. Sets <*option*> as default.
-Can group options by wrapping in *<optgroup label="groupName"></optgroup>*
-*multiple* – boolean attribute for <*select*>. Lets user select multi options by ctrl + clicking, etc.

-Autocomplete box: text field w/ preset options.
1) create *<input type="text" … list="dataListID">* //Give it a label. Also, note the *list* attr
2) create *datalist: <datalist id="..."> optionsHere </datalist>*

Checkboxes
-*<input type="checkbox" … value="...">* - make sure to set *value*, so value exits for when checked.
-*checked* – boolean attr to have check by default

Radio Buttons
-<*input type=”radio” … “value”*> - make sure val input. Takes boolean *checked* attr.
-Can give multiple radios same *name*, which puts them in a name group. When in group, only one can be selected at a time.

Buttons
-<*button type=”...”>buttonText</button*> - three types: *submit, reset, anonymous*
-types: *submit* to server, *reset* form to default val, *anonymous* no default action & require JS
-button text can be HTML styled (<*em*>, etc.) where <*input*> text cannot

Advanced Native Form Widgets

Numbers
-<*input type=”number”*> - Text box but only allows numbers. Can take *min=”#”, max=”#”*, and *step=”#”* (step places arrows on side of box that increase/decrease # by x *step)*

Sliders
-<*input type=”range”*> - Same attributes as *number*, but displays as slider instead
-Sliders do not show value, so need to implement via JS. See **HERE** for JS implementation

Date & Time Picker
-<*input type=”timeValHere”*> - Displays box to pick date/time values and only takes x format.
-Time values: *date, datetime-local* (date & time), *month* (month & year), *time, week* (week & year)
-Can take *min* and *max* values for date/time range (ex. *min="2013-06-01" max="2013-08-31"*)

File Picker
-Creates a “Browse” button” that when clicked opens up a file browser to select files.
-Syntax: <*input type="file" name="file" id="file" accept="image/*" multiple*> - *accept* and *multiple* optional.

Hidden Content
-Data sent with form, but not shown to users.
-Syntax: <*input type="hidden" … name="..." value="..."*> - *name* & *value* requited
-Does not create any space for in box model, exists as if floating above

Image Button
-Displayed like an image but when clicked behaves like *submit button*
-<*input type=”image*> - takes global *input* attributes, plus all attributes *img* can take

Progress Meter & Bar

Bar
-For showing progress of task. Content inside element is fallback for if cannot display bar
-Syntax: <*progress max="100" value="75">75/100</progress*>

Meter
-Threat meter functionality
-Usually shown as a bar, where current point on bar is a fixed value in a range defined by *min* and *max* attr
-Also takes *low* and *high* attr, which creates three subranges (min to low, low to high, high to max), useful for styling
-*optimum* attr takes a number, and depending on which subrange it falls in, the bar turns red, yellow,

or green colored (red if in worst…)

-Example of use: create meter for disk space, with min=”0%”, max=”100%, low=”50%” high=”75”%”, optimum set by Javascript as current disk usage. When disk < 50% full, bar is green. When 75% > disk > 50% full bar is yellow. When disk > 75% full, bar is red.

-Content inside <*meter*> is fallback for it *meter* cannot display

Sending Form Data

Client/Server Architecture
-Client (ex. browser) sends request to server using HTTP protocol. Server responds using http protocol.
-Attributes of <*form*> determine where and how data gets sent

action attribute
-Syntax: *action=”urlOrFileLocation”*
-Where data is sent to. If no *action* specified, sent to page for is hosted on.

method attribute
-Syntax: *method=”...”* //value can be *get*, *put*, or other less common method type
-HTTP request = two parts: header w/ browser capability metadata, body w/ info for server to process request

-*get* – *form* attribute. Used to request resource from server. Body is empty and *value*s of form elements (ex. <*input*>) appended in name/value pairs onto URL of sever specified in *action.* Empty body. Do not use for sending password or sensative info.

-*post* – *form* attribute. Used to request a resource from the server with request depending on content of sent body. No data appended to URL and sent in body instead. ex. of POST

```
POST / HTTP/1.1                                    //header
Host: foo.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 13

say=Hi&to=Mom                                      //body
```

-Can view *post*, *get*, and other methods’ client/server interaction via Network tool in Firefox

-Once send to server, server handles data with server side language (ex. PHP, Python), including processing and choosing what data to respond with, which it then sends back via HTTP methods

Sending Files
-HTTP is text data, while files are binary. Thus require special handling.

-*enctype=”multipart/form-data”* – attribute/value pair for *form*. Specifies *Content-Type* HTTP header as file. Multipart as file split into parts as sent. Send using *post*.

Security Concerns
-Cross-Site Scripting (XSS) – inject client-side script into page that is sent to back to user or other users. To prevent, filter data sent from users before processing.

-Cross-Site Request Forgery (CSRF) - inject client-side script into page that tries to escalate user privilege. To prevent, filter data sent from users before processing.
-SQL Injection – Sends SQL data to host, hoping DB will exe. To prevent, anitize data before storing in DB.
-HTTP Header injection – attacker uses data about user sent in header to build other attacks (phising, etc.)

FORMS NOTE
The following sections on forms still need to reviewed, after learning some JS and python

- -Form Data Validation
- -How to Build Custom Form Widgets
- -Sending Forms through JS
- -HTML Forms in Legacy
- -Property compatibility table for form widgets
- -Styling HTML Forms
- -Advanced Styling for HTML Forms