# 19 April 2021

* (github) source control - where u can push ur code to save it
* repository (just a chunk o code under a single label)
  + DOM - document object model: machine-readable document, which is what HTML is converted into for the machine to read and hold in meory
  + server side: PHP, Java, Ruby, Python, C#
  + HTML = hypertext markup language
    - gives the idea of howit should flo
    - it's "marked up" using the tags
  + self-closing tag: <br/>
  + html Element: objct in the memory of thebrowser -> what the tag creates in the browser
  + HTTP = hypertext transfer protocol
* BASIC STRUCTURE
* <!DOCTYPE html>
* <html lang="en">
* <head>
  + <meta charset="utf-8">
  + <title>what</title>
  + <meta name="description" content="description of ur website">
  + <meta name="author" content="ur name">
  + <link rel="stylesheet" href="styles.css">
* </head>
* <body>
  + <h1>what</h1>
* </body>
* </html>

# 21 April 2021

* deprecated elements: things that were used \*once\* but no longer
* HOMEWORK: first 16 things on freecodecamp (due Monday)
* element hierarchy and trees!
* root, node, sub-tree, leaf node, siblings
* branches: relationships between nodes
* nodes: any point on a tree where there is a bifurcation of two things
* components: a sub-tree within the larger tree
* leaf nodes: a terminal node, where the branch stops
* nesting: another way to describe hierarchical parent/child relationships
* perfect binary tree = even number of nodes between all the levels, of the two branches
* perfect: balanced, sorted, full
* complete: balanced, not sorted, not full
  + code is always trees, because it needs to be a finite thing that a machine can process in a finite amount of time!
* semantics!
* use the right HTML element for the purpose!!
* accessibility!
* accessible websites ebnefit from SEO!
* and ofc it's better for everyone
* make sure you arit alt for images (b/c that's what the screen reader reads)
* CHECKLIST
* page title
* image descriptions
* headings
* contrast ratio
* resize text! ensuring visability
* keyboard access & visual focus!
* forms, labels & errors - have proper labels and feedback on forms
* multimedia alternatives - having ARIA elements and transcripts
* ARIA spec = Accessible Rich Internet Applications Specification, add "aria-role" attribute
* role="button" etc.
* WCAG!!!! will tell you EVERYTHING you need to be completely compliant
* w3.org/WAI/WCAG21/quickref/
* 22 April, 2021: HTML attributes and forms
  + ARIA role from yesterday: need a tabindex="0"
  + aria-role, class, id, etc.
  + "config-data" that modifies the element's behavior
  + key/value pair: the key is how you get access to the value, and each attribute is a pair
  + a few exceptions are Boolean and don't need an explicit value (take on a value of true if present)
  + so: <input id="first-name" class="signup" required/>
  + so we have ID, class, and the "required" attribute
  + so: <details open/>
  + "open" is an attribute, toggling an accordion menu between open and closed
  + Global vs specific attributes!
* https://developer.mozilla.org/en-US/docs/Web/HTML/Global\_attributes
  + \*\*\* title="little tooltip title"
* FORMS AND USER input
  + semantic form elements! -- form, fieldset, legend, label, input, textarea, button, outpit, option, potgroup, select, meter, keygen, datalist
  + default behavior of an action is a "get" request!
  + action = "the URL that processes the form submission"
  + method = post / get
  + get: form data appended to the action URL with a ? eparator, use when the form has no side-effects, getting data sent back to us
  + post: sends a post request, which sends data to the server, which can do something with the data
  + name = name of the form!
* 26 April 2021
  + BUTTONS VS LINKS
* buttons are for executing javascript code; links are for navigating to a new view (usually a new page URL)
  + IMAGE semantics
  + <img src> vs background-image! --> <img> tag when the image is actual page contents, but background-image for non-content (patterns, gradients) which isn't accessible
  + HTML boilerplate!
  + including meta tags that are machine-readable, such as a <meta property="og:title" content="Hello!"/> giving you a title when sharing on twitter
  + robots.txt to un-index your website
  + evergreen browser = a browser that seeks to add releases on a very regular schedule
  + a11y.coffee = an intro to web accessibility!
  + C S S
  + you can select by element attribute!!!! --> [attribute-name]
  + and in combination it's div.wrapper (div with a class of "wrapper"), NO SPACE IN THERE
  + nesting!!!!
    - ul li{} --> two different elements, refering to ALL li that are descendants of a ul
    - ul a{} --> all <a> that are descendants of ul
    - ul > li {} --> has to be a DIRECT descendant of the first
    - ul > li > a {} --> the <a> has to be a child of the <li> which is a child of <ul>
    - ul > a {} --> an <a> that is a direct child of the ul
  + sibling selecters!!! (but no back-traversal -- pass over a tree once)
    - h1 + li {} --> selects first subsequent li after h1
    - h1 ~ li {} --> selects all subsequent li elements after h1
  + CSS percentage -- always in relation to the parent element
  + 1rem = 16px
  + em = a relative unit, scales based on how big the parent is
  + vh = view height
  + vw = view width
  + vmin = viewport minimum, measure both the current viewport's width and height, and give the smaller one
  + vmax = viewport maxiumum, measure both the current viewport's width and height, and give the larger one
  + box model: content, then padding, then border, then margin
  + box-sizing: content-box or border-box --> how the browser calculates the size of the element on the page

# 28 April 2021

* + the javascript needs to be the last element in the body, OR you can put "defer" attribute in the script element in the head
  + the "cascade" refers to: the pattern of the rule applying to all the children of the parent, a rule cascading down an HTML tree to affect all of the children the style starts with
  + CSS specificity!!!!
  + can have many of the same rule applied to the same browser element, so which should it apply??
  + THE RULE WITH THE HIGHEST SPECIFICITY HAS THE RULE APPLIED TO IT
* inline styles
* IDs
* classes, attributes, and pseudo-classes
* elements,a nd pseudo-elements
  + inline | ID | classes | elements --> each of each adds a digit to contribute to the prioritization math
  + so: #myID .class3 div = 0111 specificity
  + try not to style using IDs, because it's SO HARD to overwrite an ID
  + https://wjv.io/deck/03-css/img/css-specificity-wars.png
  + the !important is only to be used as a last resort, and is considered bad practice, because nothing else can override it and breaks the cascade
  + things written last will overwrite what gets written before, if they refer to the same thing
  + CSS resets & the useragent
  + THE USERAGENT
  + browsers all come with their built-in CSS defaults
  + useragent = the default css styles of the browser
  + so the resets are there to just reset everything to a neutral value
  + Normalize or Reset are the least opinionated of the offerings in the deck
  + WEB DESIGN SYSTEMS
  + just about creating a system of web design where you use consistent rules
  + also called pattern libraries! = a consistent & modular library of reuseable themes, components, motifs & rules for coherently styling and assembling web pages
  + colors!!!
  + HEX SUPPORTS AN APLHA CHANNEL FOR TRANSPARENCY!!! --> #ff9f4f to #ff9f4f99 (about 50% opacity)

# 29 April 2021

* + CSS FILTERS: https://developer.mozilla.org/en-US/docs/Web/CSS/filter
  + affordances --> how you communicate to the user how to use the thing, how it interacts with their mouse cursor, how it interacts with the world
  + HOW BROWSERS RENDER STUFF
  + browser parses the markup, and then "Recalculate Style" to figure out what styles to apply
  + next: "layout" all the DOM elements
  + then "Paint" the pixel data for elements
  + finally "Composite" will combine and render the painted layers onto the screen
  + CPU vs GPU
  + computer processing unit vs graphics processing unit

# 3 May 2021

* css variables are CUSTOM PROPERTIES
* HOW THE BROWSER RENDERS STUFF
  + CALCULATE style
  + takes HTML & CSS and parses it to create the DOM tree and the CSSOM, converting the tags to html elements and making a tree of styles, mapping our styles into a tree format on top of the DOM tree
  + the "Attachment" step creates the Render Tree, where everything is a node (element with style), ready for layout computation
  + Layout the nodes on the page
  + Paint - CSS properties that affect the colors and sizes of stuff gets applied to the pixels on the page
  + Composite - do some final layout adjustments, with performance imporvements, like saving html nodes into memory
* if there are animations on the page, then this calculation has to be done all over again, recalculating the entire render tree
* Bytes -> Characters (the code we write) -> Tokens -> Nodes -> the DOM
  + text nodes: terminal nodes that just contain text content
* CSSOM --> only worried about mapping the CSS style rules, only using the selectors that we give it
  + then it gets mapped onto the DOM with the selectors we give it
* so with every animation, it repaints the page each time, and the more elements have to change then performance will suffer
* if you change something at the Layout, then the Layout, Paint, and Composite will all be changed; but if you just change the Composite layer, then the bottom layers remain the same
* **LAYOUT!!!!**
  + generates geometry and position of each element, very expensive b/c engine has to reflow --> it's a cascading calculation!
  + occurs when DOM tree is manipulated, or a style chich affects layout is changed, when a browser window size is changed, when className property is changed
  + DOCUMENT FLOW!!!!!
    - or the layout flow
    - regular document flow: determined by the display property! which determines if the element is block or inline, and how its children are treated
    - BLOCK = tells the browser that this element takes up the whole width of the page
    - INLINE = tells the browser that they can be next to each other
    - FLEX =
    - GRID =
    - you can affect the outside of the element (block or inline) or the inside of the element (flow-root, table, flex, grid)
  + properties that trigger layout! don't change these!
    - width, positin, font-family, height. padding, margin, float, align, margin, display, border, white-space, clear
* **PAINT!!!!**
  + where all the pixels get their color and opacity! making something visible or hidden, adding outline and changing color: expensive b/c engine searches through all elements to determine what's visible and what isn't
  + browsers try to hand off as much as possible to the GPU, but we'r enot quite there yet
  + properties! okay to change, but fluid animation can be costly with lots of these
    - color, border-radius, background, box-shadow, text-decoration, image, visibility, outline
* **COMPOSITE!!!**
  + Only Opacity and Transform, because Blink and Webkit creates a new ayer for elements with CSS transforms!
  + TranslateZ(0) is used to force layer creation, which ensures the layer is painted as soon as the animation starts 🡪 it hands off the processing to the GPU instead of the CPU
* Compositing is faster b/c the GPU takes care of it instead of the CPU
* HOMEWORK 🡪 all CSS stuff, by Thursday, and then applied visual design assigned Thursday due Monday
* MEDIA OBJECTS
  + A card (fantastic for portfolios! Like an image with a caption text!)
  + A box for multimedia (not just text, not just imagery, a composite of 2+ things)
* Component: a re-useable chunk of code that can be slotted in anywhere on a page, filled with any kind of content
* Can this motif show up again and again? Then code it up as an isolated object