

## A. CSS = Cascading Style Sheets

### a. Purpose

- i. Visual appearance of HTML elements
- ii. Positioning in relationship to one another
- iii. Interaction between elements
- iv. Transition/animation of elements

### b. Context-free language

- i. Definition: can be described by recursive rules
- ii. Stylesheet parsing is very strict
- iii. Unlike HTML, parsers can and will generate syntax errors based on incorrect syntax usage
  1. Style pre-processors, SASS: syntax error
  2. Vanilla CSS, react app: browser will tag error and not display intended style
  3. Example

### c. CSS rules are in the global scope

- i. Applying styles too broadly can result in unintended consequences
- ii. Eg. `div { background: red; }`
  1. applies to EVERY div in the document, even divs from 3rd party libraries, etc.
- iii. Be cautious about far-ranging overrides

## B. Stylesheet rules

### a. What does a rule look like?

### b. `[selector] { [style rule] }`

- i. Eg `body { background-color: white; }`

### c. We can group many style rules between braces and all will apply to the selector

### d. We can have multiple duplicate selectors

- i. all style rules will be applied according to their specificity
- ii. Best practice is to group all rules for the same selector together so they don't collide unnecessarily

### e. Rules can override each other if they apply to the same selector and either

- i. Are located farther down in the style sheet
- ii. Have a higher specificity

## C. Selectors

### a. Stylesheet rules are made up with selectors

- i. A combination of ids, classes, tags, combinators, etc.
- ii. [https://developer.mozilla.org/en-US/docs/Web/CSS/CSS\\_Selectors](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Selectors)

### b. IDs

- i. Targets an HTML tag with a certain ID attribute
- ii. Id must be unique, 1 ID per document
  1. HTML will allow you to tag a document with more than 1 ID, and some browsers will apply the styles properly
  2. Multiple IDs in a document are not valid HTML and will throw a validation error
- iii. Stylesheet rule begins with #

### c. Classes

- i. Targets an HTML tag with a certain string as part of the class attribute, React = `className` attribute
- ii. Classes do not need to be unique
  1. We can have the same class several times per document

- a. Classes are the workhorse of CSS
  - b. We want most of our styles to be classes so that they can be reused/reapplied
- 2. Many classes can be applied to the same tag and will stack up
- d. HTML tags
  - i. Style rules are applied to a plain HTML tag
  - ii. Applies broadly to any of that tag in the document
  - iii. Eg `div { background: red }`
- e. Pseudo selectors/pseudo classes
  - i. Used when you want to style a selected element but only when it is in a certain state
  - ii. Eg `:hover`, `:selected`, `:checked`, `:first-child`, `:nth-of-type`
  - iii. [https://developer.mozilla.org/en-US/docs/Learn/CSS/Introduction\\_to\\_CSS/Pseudo-classes\\_and\\_pseudo-elements#Pseudo-classes](https://developer.mozilla.org/en-US/docs/Learn/CSS/Introduction_to_CSS/Pseudo-classes_and_pseudo-elements#Pseudo-classes)
- f. Pseudo elements
  - i. keywords preceded by a colons (`::`) added to the end of selectors to select a certain part of an element
  - ii. Different than the pseudo elements above!
  - iii. Only 6: `::after`, `::before`, `::first-letter`, `::first-line`, `::selection`, `::backdrop`
  - iv. Eg. `::first-letter` example
  - v. Eg. `::first-line` example
- g. Attribute Selectors
  - i. Styling is applied to HTML tag based on characteristics inherent to the tag
  - ii. Eg `a[href="http://www.google.com"] { color: green }`
  - iii. Brittle, will probably break styles if HTML structure is changed
- h. Combinators
  - i. Ways of combining classes and ids
    - 1. Descendent: element is a descendent of previous element
      - a. Use a space
    - 2. Child: element is a direct descendent of previous element
      - a. Use a `>`
    - 3. Sibling: element is a sibling of previous element (they have the same parent but don't necessarily follow directly)
      - a. Use a `~`
    - 4. Adjacent: element is an adjacent sibling of previous element
      - a. Use a `+`

## D. Naming

- a. Give your style names (classes & ids) meaningful names that are easy to reuse
- b. Try not to reference how the class/id is styled within the name
  - i. `.blue-underline`
  - ii. That may be how it's styled now, but what happens if that changes in the future?
  - iii. You don't want your codebase full of `.blue-underline` that are actually green background colors
    - 1. Better name might be the purpose, like `.content-highlight`
    - 2. Reusable and not tied to specific implementation
    - 3. Has semantic meaning for people not familiar why the blue underline is used

## E. Cascading

- a. style sheets can "stack up" (cascade) until sum of calculated styles are reflected in the presentation

- b. The cascade is an algorithm that defines how to combine property values originating from different sources, including the default browser stylesheets
- c. Specificity
  - i. CSS rules win their way to visibility based on a calculated score from cascade algorithm
  - ii. Final score is called specificity and determines what style is displayed
  - iii. If two or more selectors apply to the same element, the one with higher specificity wins
  - iv. There are four distinct categories of selector which can combine define the specificity level of a given selector
    - 1. inline styles
    - 2. IDs & classes
    - 3. Attributes
    - 4. Tag elements
  - v. Calculating specificity
    - 1. Give every id selector ("foo") a value of 100
    - 2. Give every class selector (".bar") a value of 10
    - 3. Give every pseudo selector (":hover", ":selected") a value of 10
    - 4. Give every HTML selector ("div") a value of 1
    - 5. Give every pseudo element (":before", ":first-letter") a value of 1
  - vi. Add them all up to get the specificity value
    - 1. If selectors have an equal specificity value, the latest rule is the one that counts
    - 2. The inline stylesheet has a greater specificity than other rules
- d. !important
  - i. Add to a style rule to apply style rule regardless of specificity of others
  - ii. If 2 conflicting rules have !important, specificity decides
  - iii. If everything is important, nothing is
  - iv. Best practice: add ids instead of !important

#### F. How to add styles to a document

- a. Inline head = <style> tag within <head> tag
  - i. Advantage = visibility within doc
  - ii. Disadvantage = only applies to current doc, not scalable
- b. Inline tag = style attribute on tag
  - i. Advantage = only applies to current tag, high specificity
  - ii. Disadvantage = only applies to current tag, not scalable
- c. External = linked stylesheet <link href="stylesheet"> within <head> tag
  - i. Advantage = defined styles can be applied to any doc where stylesheet is linked, most common way of organizing styles
  - ii. Disadvantage = single stylesheet (or bundled) can be unwieldy, can import styles not present on page, wasteful
- d. CSS modules = CSS styles are included as js module via css class names created by webpack when building web app like React
  - i. Advantage = styles are namespaced within context of individual modules, no longer globally scoped = no unintentional style collisions

#### G. Units

- a. Px
  - i. "Magic" unit of CSS
  - ii. not related to the current font
  - iii. px is an abstract unit where a ratio controls how the pixel maps to actual device pixel and how it maps to physical units

- iv. Px is designed to be roughly equivalent across devices
- v. the goal is for 96px to equal about 1 inch on the screen, regardless of the actual pixel density of the screen
  - 1. if the user is on a phone with a pixel density of 266 DPI (dots per inch), and an element is placed on the screen with a width of 96px, the actual width of the element would be 266 device pixels (in other words, 1 inch)
- b. Percentage (relative to parent container)
- c. Em (relative to current font size)
  - i. 1em = current font size of element being styled
- d. Rem (relative to current font size)
  - i. 1rem = current font size of root em (html font-size)
  - ii. Inherited font sizes have no effect
- e. Vh = 100th height of viewport
- f. Vw = 100th width of viewport

#### H. Box model

- a. Each HTML element is rendered as a box
  - i. Content
  - ii. Padding inside box
  - iii. Border
  - iv. Margin outside box
- b. Block box
  - i. Always appear below each other in default browser display
  - ii. "Static" flow
  - iii. Width is based on the width of its parent container
  - iv. Height is based on the content it contains
- c. Inline box
  - i. Not for determining layout but for styling inside blocks
  - ii. Width is based on the content it contains
  - iii. Adding block styling like margins, height, width don't have any effect

#### I. Basic CSS attributes

- a. These attributes can be applied to just about any element via style rules
- b. Visibility
  - i. Hidden: hide the element but leave the space it occupied (almost like making it transparent)
    - 1. Not ignored by screen readers
  - ii. Visible (default): show the element
- c. Color
  - i. Named colors: <https://htmlcolorcodes.com/color-names/>
    - 1. Eg `div { background: red; }`
  - ii. Hexadecimal colors
    - 1. `#[0-9a-f]{6}`
    - 2. `#` + 6 digits/3 tuples: `#[rr][gg][bb]`, 0-255 rgb value
    - 3. 0-255 → 0-9, A-F
    - 4. <https://www.google.com/search?q=rgb+to+hex>
    - 5. `ffffff` = `[255][255][255]` = pure white
    - 6. `000000` = `[0][0][0]` = pure black
    - 7. Can use hexadecimal shorthand notation to save space

- a. Eg `.dark-yellow {color:#ffcc00;} → .dark-yellow {color:#fc0;}`
  - b. This only works if all 3 tuples are matching (ie cannot shorthand `#ccfeff` to `#cfef`)
- iii. Rgb/rgba
  1. Use full RGB values as `rgb(R, G, B)`
  2. Eg `rgb(255, 255, 255) = #ffffff` = pure white
  3. Eg `rgb(0, 0, 0) = #000000` = pure black
  4. Rgba
    - a. adds opacity value at some decimal value between 0 and 1
    - b. 0 = full transparency
    - c. 1 = full opacity
    - d. Eg `rgba(255, 255, 255, .5) = #ffffff` at .5 transparency
- d. Background
  - i. Change the background of any element, ie what paints underneath the content in that element
  - ii. Lots of background images: we'll cover during images in week 6
  - iii. Background-color
    1. applies solid colors as background on an element
- e. Display
  - i. behavior, ie `block → inline + inline → block`
  - ii. Inline
    1. within a block container
    2. Accepts margin and padding but still sits inline within text
    3. Does not accept height/width
  - iii. Inline-block
    1. Similar to inline but will accept height/width
  - iv. Block = creates its own bounding box
  - v. Flex = defines a flex container
  - vi. Grid = defines a grid container
  - vii. None = Not displayed, Still in the DOM, removed visually and ignored by screen readers (unlike visibility: hidden)
- f. Border
  - i. Line at boundary of box of content
  - ii. Border-width
    1. thickness of the border
      - a. Named: Thick = 5px b. Medium = 3px c. Thin = 1px
      - b. Length: px, em, rem, vh and vw units
  - iii. Border-style
    1. Specifies the type of line drawn around the element
    2. <https://developer.mozilla.org/en-US/docs/Web/CSS/border-style>
    3. solid: A solid, continuous line
    4. none (default): No line is drawn
    5. dashed: A line that consists of dashes
    6. dotted: A line that consists of dots
  - iv. Border-color
    1. Specifies the color of the border
  - v. Border-radius

1. give any element "rounded corners"
2. Eg border-radius: 4px
3. Can specify the value of border-radius in percentages to create a circle or ellipse shape
4. % can be used any time you want the border radius to be directly correlated with the elements width
5. Border-radius: 50%

g. Padding

- i. Spacing inside box of content between content and border
- ii. Cannot be negative

h. Margin

- i. Spacing outside border
- ii. Can be negative
  1. top/left: pulls element in that direction
  2. bottom/right: pulls other elements into overlapping element

i. Box-sizing

- i. Allows you to change how the width of the box is calculated
- ii. Content-box
  1. Built up from content box
  2. Eg 600px container, 3 boxes 200px wide
  3. Boxes are actually 202px wide with border
- iii. Border-box
  1. Built down from external width
  2. Forces actual width of entire box to "width", accounting for padding/border
  3. Best practice: set your blocks to use border-box

J. CSS List specific attributes

- a. list-style-type: Sets the type of bullets to use for the list, for example, square or circle bullets for an unordered list, or numbers, letters or roman numerals for an ordered list
  - i. <https://developer.mozilla.org/en-US/docs/Web/CSS/list-style-type#Values>
  - ii. None: removes the bullets from the list
- b. list-style-position: Sets whether the bullets appear inside the list items, or outside them before the start of each item
  - i. Outside (default): outside the bounds of the list item
  - ii. Inside: inside the bounds of the list item
- c. list-style-image: Allows you to use a custom image for the bullet, rather than a simple square or circle

K. CSS Typography attributes

- a. <https://builttoadapt.io/8-point-grid-vertical-rhythm-90d05ad95032>
- b. Explanation of typography
  - i. Baseline
  - ii. ascenders/descenders
  - iii. Line-height
- c. Font-family = specifies a prioritized list of one or more font family names and/or generic family names for the selected element
  - i. Font stack
    1. Serif, sans serif, monospace, cursive, fantasy, system-ui
    2. Best practice: always include at least one generic font family
    3. Cannot count on what fonts a user has installed

- d. Importing a font to use
  - i. <link>
  - ii. @import
- e. Font-size
  - i. <https://css-tricks.com/css-font-size/>
  - ii. Named: Xx-small, x-small, small, medium, large, x-large, xx-large
  - iii. Relative: smaller, larger (roughly corresponding to named values)
  - iv. Length: em, rem, px, etc.
  - v. Percentage: relative to parent's font size
- f. Line-height
  - i. sets the height of a line box
  - ii. amount of space between lines in the same block
  - iii. Example: <https://developer.mozilla.org/en-US/docs/Web/CSS/line-height>
- g. Font-weight
  - i. Named weights: normal, bold
  - ii. Relative: lighter, bolder
  - iii. Numeric: between 1 and 1000, inclusive
    - 1. Numeric weights 100,200,etc. map to typical font weights like extra light, bold, black, etc.
    - 2. 1-1000 supports finer grained control from fonts
      - a. Somewhat spotty support
      - b. Not supported by all browsers, IE notably (Edge supports)
- h. Color
  - i. Change text color of content box
- i. Font-style
  - i. Normal
  - ii. Italic
  - iii. Oblique
  - iv. Italic vs. oblique = oblique is usually just sloped, italic is usually a different font style, often cursive
- j. Text-decoration
  - i. Shorthand for text-decoration-line, text-decoration-color, and text-decoration-style
  - ii. Appearance of decorative lines used on text
  - iii. Best practice: Do not use underlining except on links
  - iv. Style: dashed, dotted, wavy, solid, double
  - v. Line: underline, overline, line-through
  - vi. Eg. `text-decoration: green dashed underline`
- k. Transform
  - i. Takes language specific cases into account
  - ii. CAPITALIZE
  - iii. Uppercase
  - iv. Lowercase

## L. CSS Positioning

- a. Float
  - i. Concept comes from print design
  - ii. Images/elements set into layout so that text wraps ("flows") around them
  - iii. Removed from the flow of the page, but remain part of it to affect other elements

- iv. Floating an element usually changes display: attribute to "block"
- b. Position
  - i. Can help you manipulate the location of an element in the page or relative to the other other elements around it
    - 1. Static
      - a. every element has static position by default
      - b. will conform to normal page flow
    - 2. Relative
      - a. Continues to appear in normal page flow
      - b. left/right/top/bottom can now be applied
      - c. Element will be nudged in that direction
    - 3. Absolute
      - a. element is removed from the flow of the document
      - b. other elements will behave as if it's not there
      - c. Positional properties will work on it
      - d. If no other positioning is set on parent, child positioning will be relative to the document
      - e. To make positioning relative to parent, set position: relative on parent
    - 4. Fixed
      - a. Similar to absolute
      - b. Position relative to document
      - c. Not affected by scrolling
- c. Z-index
  - i. controls the vertical stacking order of elements that overlap
    - 1. relative positioning has nudged it over something else
    - 2. negative margin has pulled the element over another
    - 3. absolutely positioned elements overlap each other
  - ii. ie, which one appears as if it is physically closer to you
  - iii. z-index only affects positioned elements: absolute, relative, etc.
  - iv. Without z-index value, elements stack in the order that they appear in the DOM, ie the lowest one down at the same hierarchy level appears on top

## M. CSS modules

- a. <http://localhost:3000/?demo=cssmodules>
- b. CSS files where class names are scoped locally by default
- c. not an official spec or an implementation in the browser
- d. a process in a build step (w/ the help of Webpack or Browserify)
- e. changes class names and selectors to be namespaced
- f. identifier is guaranteed to be globally unique
  - i. Advantages
    - 1. Step towards modular and reusable components that will not have side effects
    - 2. Cleaner CSS
    - 3. Avoidance of monolithic CSS files (each component will have its own file)
  - ii. Disadvantages
    - 1. not as human-readable DOM
    - 2. Need special webpack setup
- g. How does it work?
  - i. Normal css = styles are linked into the page and available globally



- ii. Tries to solve inadvertent collisions/cascades in disparate components, esp in larger apps
- iii. With modules, we import the styles like JS import, which transforms the CSS rules, namespacing all classes
- iv. css-loader injects stylesheet into the document
- v. value returned from the import is an object mapping of local CSS class names to their namespaced versions
- vi. Eg, `{ foo:"foo_foo_abcde", bar:"foo_bar_abcde" }`
- vii. Setting `class={style.foo}` in HTML = setting it to the local version of that named class, `class="foo_foo_abcde"`