1. 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. Can be described by recursive rules
        ii. Stylesheet parsing -- 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: browser will tag error and not display intended style
        iv. Example
    c. Cascade: style sheets can "stack up" (cascade) until sum of calculated styles are reflected in the presentation
        i. CSS rules are in the global scope
            1. Applying styles too broadly can result in unintended consequences
                a. Eg. `div { background: red; }` applies to EVERY div in the document, even divs from 3rd party libraries, etc.
            2. Be cautious about far-ranging overrides that can cascade through doc
        ii. Browser stylesheets
        iii. Best practice: "reset"/"normalize" stylesheet
            1. https://necolas.github.io/normalize.css/8.0.1/normalize.css
    d. Specificity
        i. CSS rules win their way to visibility based on their specificity
        ii. Rules browser uses to calculate precedence of styling
        iii. If two or more selectors apply to the same element, the one with higher specificity wins
        iv. Four distinct categories which define the specificity level of a given selector
            1. inline styles
            2. IDs & classes
            3. Attributes
            4. Tag elements
        v. Rules:
            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
            6. Add them all up to get the specificity value
            7. If selectors have an equal specificity value, the latest rule is the one that counts
            8. The inline stylesheet has a greater specificity than other rules
        vi. !important
            1. Add to a style rule to apply style rule regardless of specificity of others
                a. If 2 conflicting rules have !important, specificity decides
            2. If everything is important, nothing is
            3. Best practice: add ids instead of !important
2. Stylesheet locations

   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
   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
   e. Example
3. Stylesheet rules
   a. What does a rule look like?
   b. [selector] { [style rule] }
      i. Eg `body { background-color: white; }`
   c. Can group many style rules between braces, all will apply to the selector
   d. Can have multiple duplicate selectors, all style rules will be applied
      i. Eg
         1. `body { background-color: white; }`
         2. `body { padding: 0; }`
   e. Rules can override each other if they apply to the same selector and either
      i. Come later (farther down) in the style sheet
      ii. Have a higher specificity
   f. Naming
      i. Give your style names (classes & ids) meaningful names that are easy to reuse
      ii. Try not to reference how the class/id is styled within the name
         1. Eg. `.blue-underline`
         2. That may be how it's styled now, but what happens if that changes in the future?
         3. You don't want your codebase full of `.blue-underline` that are actually green background colors
         4. Better name might be the purpose, like .content-highlight
            a. Reusable and not tied to specific implementation
            b. Has semantic meaning for people not familiar why the blue underline is used
4. Styling HTML
   a. Stylesheet rule is associated with selector, combination of ids, classes, tags, combinators, etc.
      i. https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Selectors
      ii. IDs
         1. HTML tag is given id attribute
         2. Id must be unique, 1 ID per document
         3. Stylesheet rule begins with #
      iii. Classes
         1. HTML tag is given class attribute, React = className attribute

        2. Classes do not need to be unique, can have same class several times per document
        3. Many classes can be applied to the same tag
    iv. Pseudo selectors/pseudo classes
        1. Used when you want to style a selected element but only when it is in a certain state
        2. Eg `:hover, :selected, :checked, :first-child, :nth-of-type`
        3. https://developer.mozilla.org/en-US/docs/Learn/CSS/Introduction_to_CSS/Pseudo-classes_and_pseudo-elements#Pseudo-classes
    v. Pseudo elements
        1. keywords preceded by a colons (:) added to the end of selectors to select a certain part of an element
        2. Different than the pseudo elements above!
        3. Only 6: `:after, :before, :first-letter, :first-line, :selection, :backdrop`
        4. :first-letter example
        5. :after example
    vi. Attribute Selectors
        1. Styling is applied to HTML tag based on characteristics inherent to the tag
            a. Eg `a[href=”http://www.google.com”] { color: green }`
        2. Brittle, will probably break styles if HTML structure is changed
    vii. Combinators
        1. Ways of combining classes and ids
        2. Descendent: element is a descendent of previous element
            a. Use a space
            b. Eg `section p` matches `<section><div><p>text</p></div></section>`
        3. Child: element is a direct descendent of previous element
            a. Use a >
            b. Eg `section > div` matches, `section > p` does not `<section><div><p>text</p></div></section>`
        4. Sibling: element is a sibling of previous element (they have the same parent but don’t necessarily follow directly)
            a. Use a ~
            b. Eg `div ~ button` matches `<section><div><p>text</p></div><img src=”foo.jpg” /><button>bar</button></section>`
        5. Adjacent: element is an adjacent sibling of previous element
            a. Use a +
            b. Eg `div + img` matches, `div + button` does not `<section><div><p>text</p></div><img src=”foo.jpg” /><button>bar</button></section>`
b. Units
    i. Px
        1. “Magic” unit of CSS
        2. not related to the current font
        3. 'reference' pixel, not a device pixel.
        4. px is an abstract unit where a ratio controls

            a. How it maps to actual device pixel
            b. How it maps to physical units (in a fixed way, the ratio is always 96 CSS px to an inch)
         5. designed to be roughly equivalent across devices
      ii. Percentage (relative to parent container)
      iii. Em (relative to current font size)
         1. 1em = current font size of element to style
      iv. Rem (relative to current font size)
         1. 1rem = current font size of root em (html font-size)
         2. Inherited font sizes have no effect
      v. Vh = 100th height of viewport
      vi. Vw = 100th width of viewport
      vii. For screen, recommended using em/rem, px, %
5. visibility
   a. Hidden: hide the element but leave the space it occupied (almost like making it transparent)
   b. Visible (default): show the element
6. Color
   a. Named colors: https://htmlcolorcodes.com/color-names/
   b. Hexadecimal colors
      i. = #[0-9a-f]{6}
      ii. # + 6 digits/3 tuples: #[rr][gg][bb], 0-255 rgb value
         1. 0-255 → 0-9, A-F
         2. No need to calculate yourself
            a. Designer will give you RGB/hex values
            b. https://www.google.com/search?q=rgb+to+hex
      iii. #ffffff = [255][255][255] = pure white
      iv. #000000 = [0][0][0] = pure black
      v. Can use hexadecimal shorthand notation to save space
         1. Eg `.dark-yellow {color:#ffcc00;} → .dark-yellow {color:#fc0;}`
         2. This only works if all 3 tuples are matching (ie cannot shorthand `#ccfeff` to `#cfef`)
   c. Rgb/rgba
      i. Use full RGB values as rgb(R, G, B)
         1. Eg `rgb(255, 255, 255) = #ffffff = pure white`
         2. Eg `rgb(0, 0, 0) = #000000 = pure black`
      ii. Rgba
         1. adds opacity value at some decimal value between 0 and 1
         2. 0 = full transparency
         3. 1 = full opacity
         4. Eg `rgba(255, 255, 255, .5) = #ffffff` at .5 transparency
      iii. example
7. Background
   a. Change the background of any element, ie what paints underneath the content in that element
   b. Lots of background images: we'll cover during images in week 6
   c. Background-color
      i. applies solid colors as background on an element
8. Box model

a. Each HTML element is rendered as a box
    i. Block box
        1. Always appear below each other in default browser display
        2. “Static” flow
        3. Width is based on the width of its parent container
        4. Height is based on the content it contains
    ii. Inline box
        1. Not for determining layout but for styling inside blocks
        2. Width is based on the content it contains
        3. Adding block styling like margins, height, width don’t have any effect
    iii. Can override behavior, ie block → inline + inline → block
    iv. Display
        1. Inline = default value for elements
            a. Browser stylesheets reset many to “block”
            b. Inline within a block container
            c. Accepts margin and padding but still sits inline within text
            d. Does not accept height/width
        2. Inline-block
            a. Similar to inline but will accept height/width
        3. Block = creates its own bounding box
        4. Flex = defines a flex container
        5. Grid = defines a grid container
        6. Table, et. al = force non-tabular elements to behave like a table
            a. https://css-tricks.com/almanac/properties/d/display/#display-table
        7. None
            a. Not displayed,
            b. Still in the DOM, removed visually and ignored by screen readers (unlike visibility: hidden)

b. border
    i. Line at boundary of box of content
    ii. Border-width
        1. thickness of the border
        2. Named:
            a. `Thick = 5px`
            b. `Medium = 3px`
            c. `Thin = 1px`
        3. Length
            a. 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
                a. can be used any time you want the border radius to be directly correlated with the elements width
                b. `Border-radius: 50%`
            4. rounding doesn't have to be perfectly circular, it can be elliptical
                a. Can specify the radiuses in which the corner is rounded by
                b. `border-radius: 10px/30px`
    c. padding
        i. Spacing inside box of content between content and border
        ii. Cannot be negative
    d. margin
        i. Spacing outside border
        ii. Independent values
        iii. Can be negative
            1. top/left: pulls element in that direction
            2. bottom/right: pulls other elements into overlapping element
    a. 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
2. Lists
    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. Disc: A filled circle (default value)
        iii. Circle: A hollow circle
        iv. Square: A filled square
        v. Decimal: numbers
        vi. Upper-roman: roman numerals
        vii. 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.
    d.
3. Typography
    a. Explanation of typography

        i. Baseline
        ii. ascenders/descenders
        iii. Line-height
        iv. https://builttoadapt.io/8-point-grid-vertical-rhythm-90d05ad95032

b. 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
    ii. Serif, sans serif, monospace, cursive, fantasy, system-ui
    iii. Best practice: always include at least one generic font family
        1. Cannot count on what fonts a user has installed
    iv. Importing a font to use
        1. <link>
        2. @import
        3. Example

c. Font-size
    i. Named: `Xx-small, x-small, small, medium, large, x-large, xx-large`
    ii. Relative: `smaller, larger` (roughly corresponding to named values)
    iii. Length: em, rem, px, etc.
    iv. Percentage: relative to parent's font size

d. 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

e. 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
        3. Somewhat spotty support
        4. Not supported by all browsers, IE notably (Edge supports)

f. Color
    i. Change text color of content box

g. 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

h. 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`
        vii. Example
    i. Transform
        i. Takes language specific cases into account
        ii. Capitalize
        iii. Uppercase
        iv. lowercase
    j. HTML entities
        i. special character that can’t be represented as plain text in an HTML document
        ii. Reserved = <, >, and &
        iii. Quotation marks
        iv. https://dev.w3.org/html5/html-author/charref
    k. Example
4. Positioning with CSS
    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 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.
        ii. Static
            1. every element has static position by default
            2. will conform to normal page flow.
        iii. Relative
            1. Continues to appear in normal page flow
            2. left/right/top/bottom can now be applied
            3. Element will be nudged in that direction
            4. Example
        iv. Absolute
            1. element is removed from the flow of the document
            2. other elements will behave as if it’s not there
            3. Positional properties will work on it
            4. If no other positioning is set on parent, child positioning will be relative to the document.
                a. To make positioning relative to parent, set position: relative on parent
            5. Example
        v. Fixed
            1. Similar to absolute
            2. Position relative to document
            3. Not affected by scrolling
            4. Example
    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 elements not statically positioned (ie, position absolute, relative, etc.)
        iv. Without z-index value
            1. elements stack in the order that they appear in the DOM
            2. Ie the lowest one down at the same hierarchy level appears on top

5. CSS shorthand
    a. Way of collapsing some number of style rules that act on a certain set of values into a single rule
    b. Shorthand properties try not to force a specific order for the values of the properties they replace
        i. If values could all be the same and would be difficult to determine which is which, certain orders are followed
            1. Shorthands handling properties related to edges of a box, like `border-style, margin or padding` use 1-to-4-value syntax:
                a. `border-width: 1em` — value = all edges
                b. `border-width: 1em 2em`
                    i. first value = top and bottom
                    ii. second value = left and right
                c. `border-width: 1em 2em 3em`
                    i. first value = top
                    ii. second value = left and right
                    iii. third value = bottom
                d. `border-width: 1em 2em 3em 4em`
                    i. first value = top
                    ii. second value = right
                    iii. third value = bottom
                    iv. fourth value = left
            2. Shorthands handling properties related to corners of a box, like `border-radius` use 1-to-4-value syntax:
                a. `border-radius: 1em` — value = all corners
                b. `border-radius: 1em 2em`
                    i. first value = top left and bottom right
                    ii. second value = top right and bottom left
                c. `border-radius: 1em 2em 3em`
                    i. first value = top left
                    ii. second value = top right and bottom left
                    iii. third value = bottom right
                d. `border-radius: 1em 2em 3em 4em`
                    i. first value = top left
                    ii. second value = top right
                    iii. third value = bottom right
                    iv. fourth value = bottom left
                    v. Clockwise from top left
    c. Inherit = take computed value of property from parent element
    d. Initial = apply initial/default value of property
6. CSS modules
    a. CSS files where class names are scoped locally by default
        i. not an official spec or an implementation in the browser

  ii. a process in a build step (w/ the help of Webpack or Browserify)
  iii. changes class names and selectors to be namespaced
  iv. identifier is guaranteed to be globally unique

b. Key benefits
  i. Step towards modular and reusable components that will not have side effects
  ii. Cleaner CSS
  iii. Avoidance of monolithic CSS files (each component will have its own file)

c. Disadvantages
  i. not as human-readable DOM
  ii. Need special webpack setup

d. 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
  iv. This transforms the CSS rules, namespacing all classes
  v. css-loader injects stylesheet into the document
  vi. value returned from the import is an object mapping of local CSS class names to their namespaced versions
    1. 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"`.