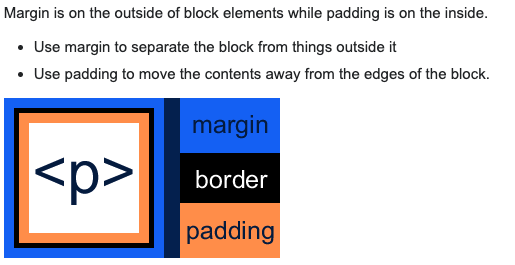
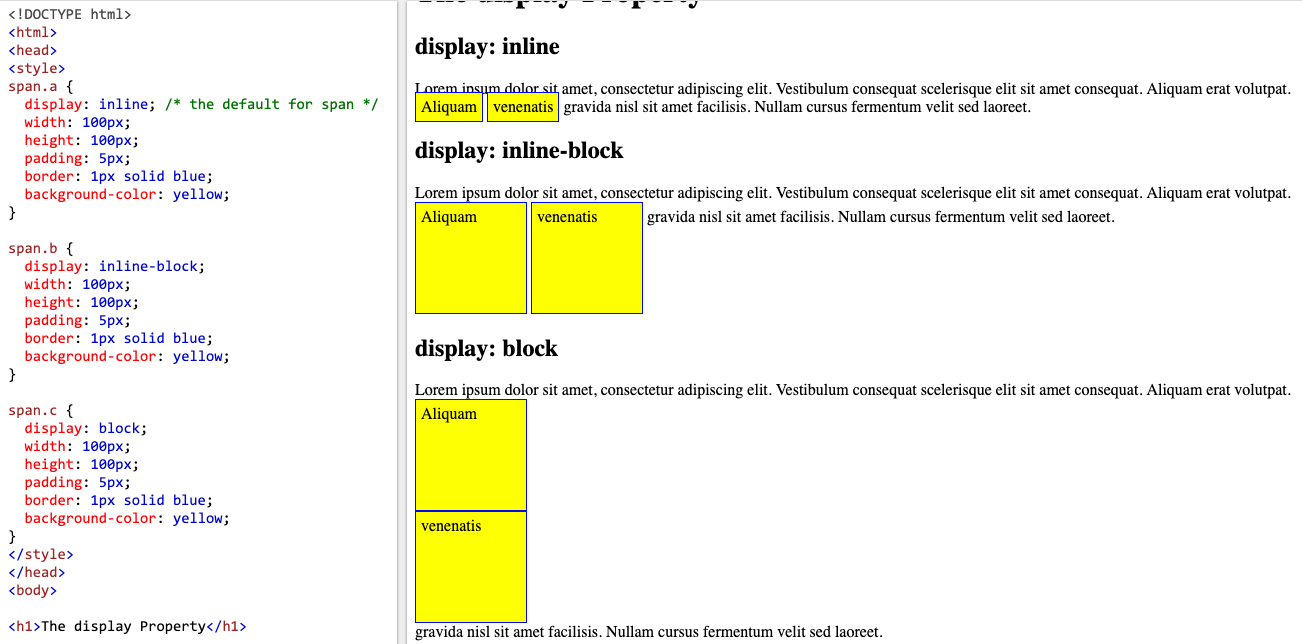
CSS:

* margin VS padding
  + vertical margins auto-collapse, but padding doesn’t



* **Display property**: inline VS block VS inline-block
  + **inline**: **NO** line breaker (same line); **NOT** allow set element width & height
  + **inline-block**: **NO** line breaker (same line); allow set element width & height
  + **block**: has line breaker (different lines); allow set element width & height



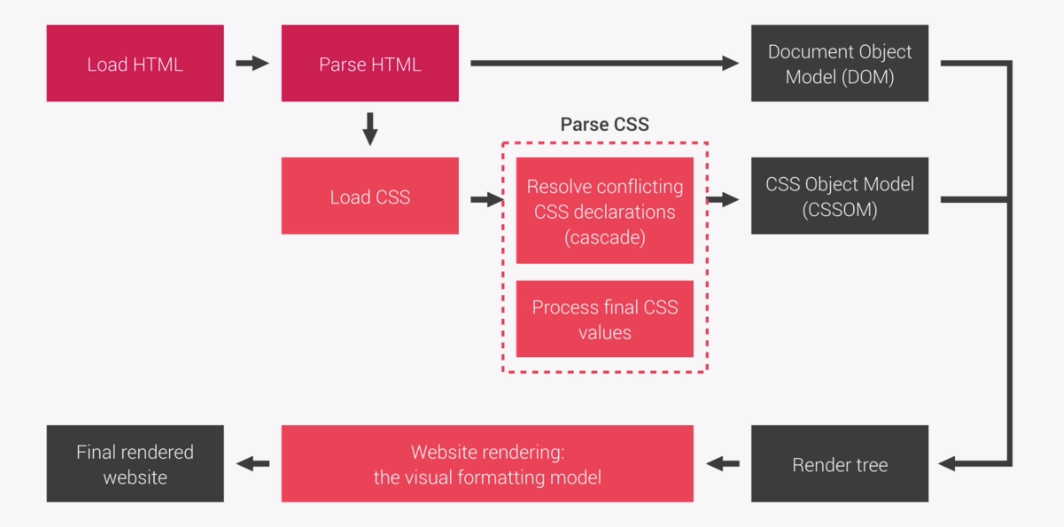
* **Position property:** <https://www.w3schools.com/css/css_positioning.asp>
  + **static (default): NOT positioned element**
    - not affected by top, bottom, left, right properties
    - according to the normal flow of page
    - eg: **<div>**
  + **relative:** 
    - set left, right, top, bottom properties (from its original position)
    - according to its normal position
  + **fixed**
    - relative to the viewpoint (stays still when scrolling the page)
  + **absolute**
    - relative to the **nearest positioned ancestor**
    - if no positioned ancestor, it used document body
      * **\*Positioned:** any position property **except static**
  + **sticky (like float??)**
    - according to the user’s **scroll** position
  + **Overlapping** Elements:
    - z-index: -1 # placed behind other elements
  + **transform & translate** (basing on the element itself)
    - eg: transform: translate (-50%, -50%) # shift to left and top by 50% of the element itself
* **<span> VS <div>**
  + **<span>** element is **inline;** usually used for a small chunk if HTML **inside a line**
  + **<div>** element is **block-line** (having line breakers before and after it)**;** used for larger chunk of code**; <div> is static (position)**

**CSS crucial concepts: 3 pillars for professional developer**

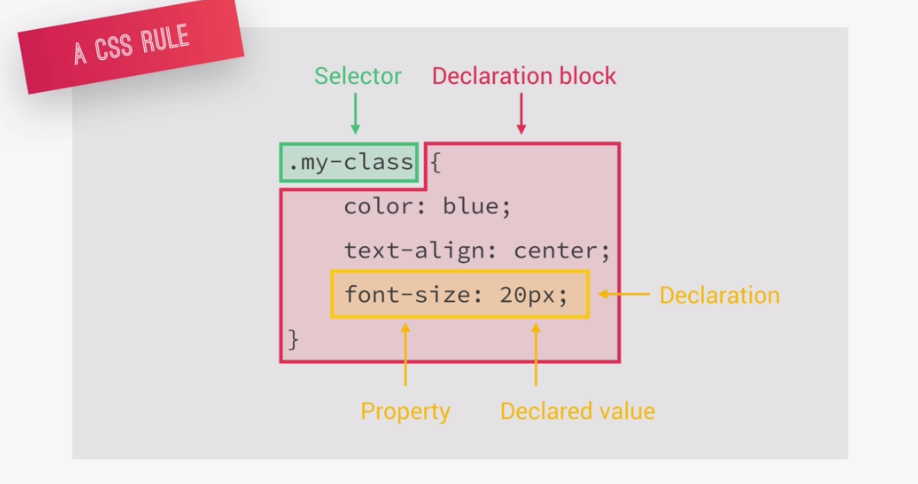
* responsive design: works for all screen size and devices
  + responsive img
  + correct unit for font size or element dimensions
  + desktop first VS mobile first
* maintainable & scalable code
  + clean, easy, reusable code
  + name the classes
  + structure the mark up in HTML
* web performance
  + as little as HTTP request (less files) as possible
  + using preprocessor (Sass)
  + use less images; compress img 🡪 less bandwidth for user end

**CSS + HTML** process how?

1. parsing: page 3-7 B. rendering: page 8-10



CSS rule & format



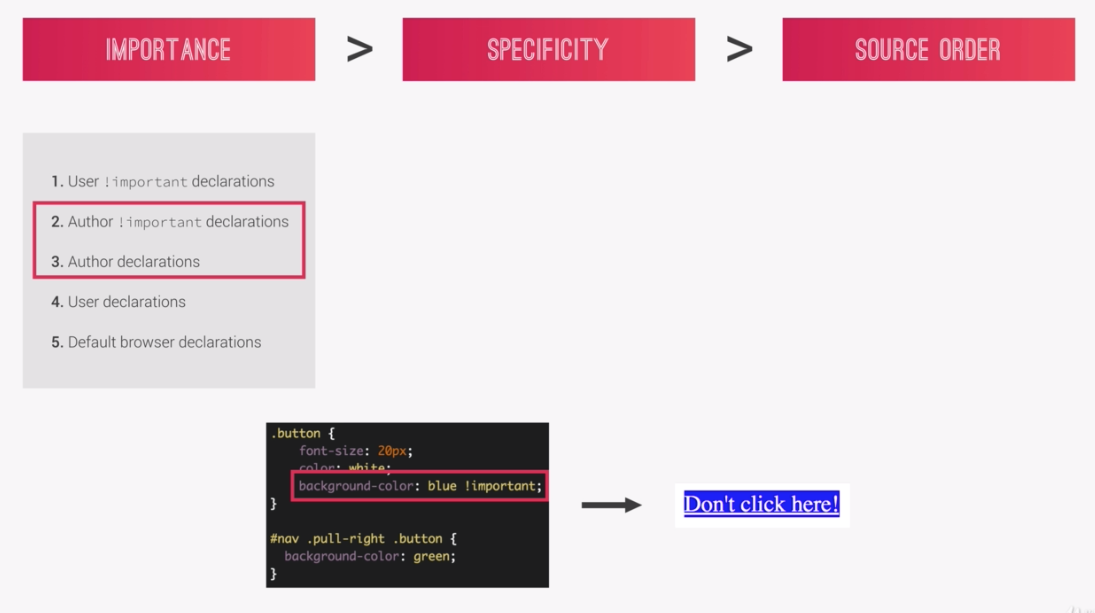
ID🡪 #nav

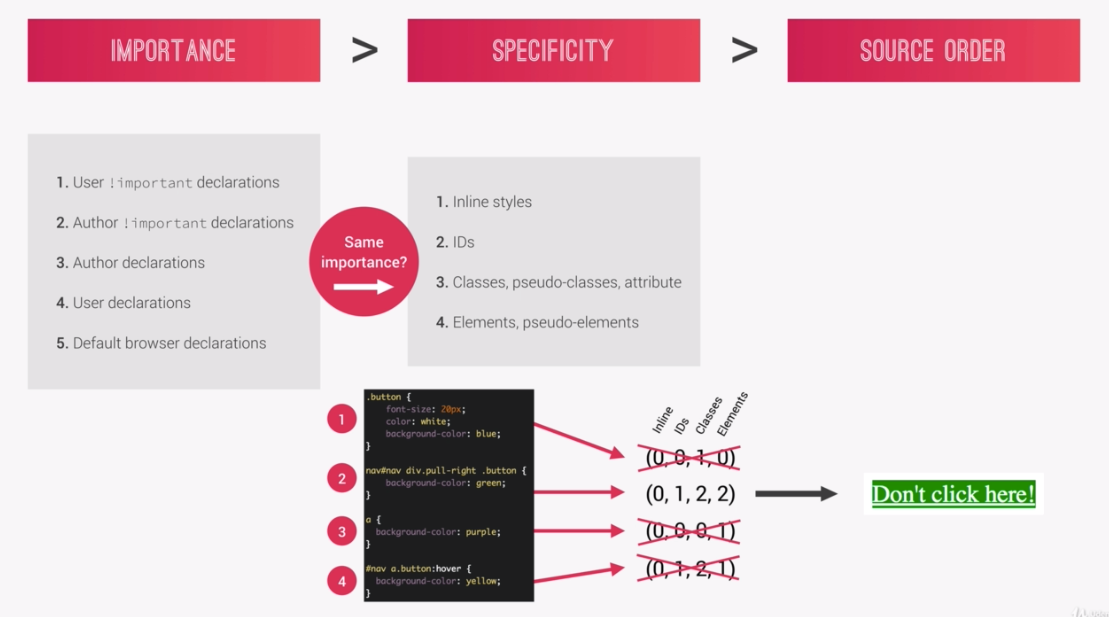
class🡪 .button :hover (pseudo-class)

element🡪 <a> ::after (pseudo-element)

**CSS parsing, how?**

* **cascade**: process of combining different stylesheets & resolving conflicts between different CSS rules & declarations (决定级别)
  1. author > user > browser (user agent)
  2. importance > specificity > source order



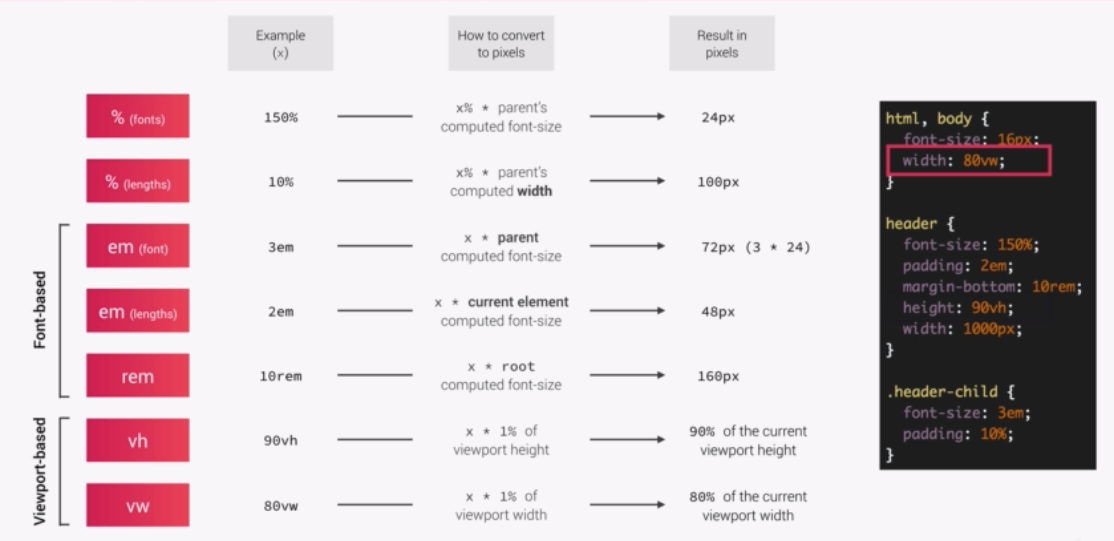


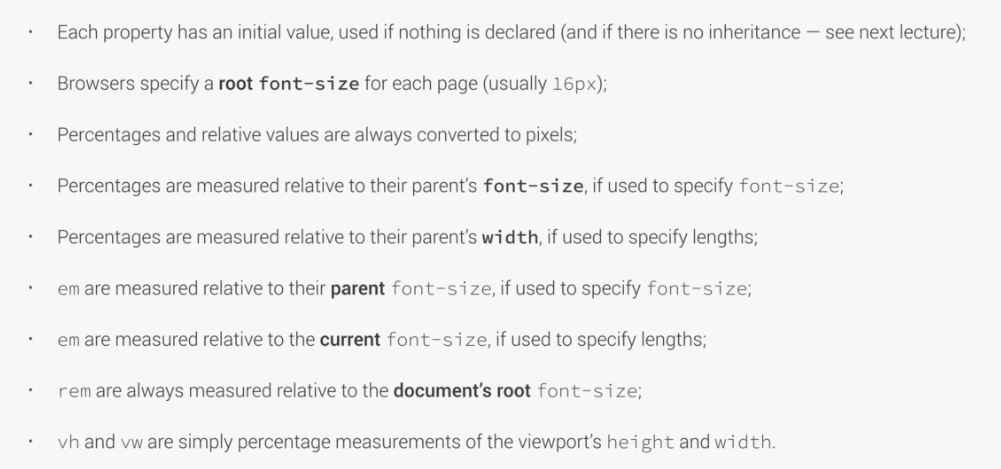


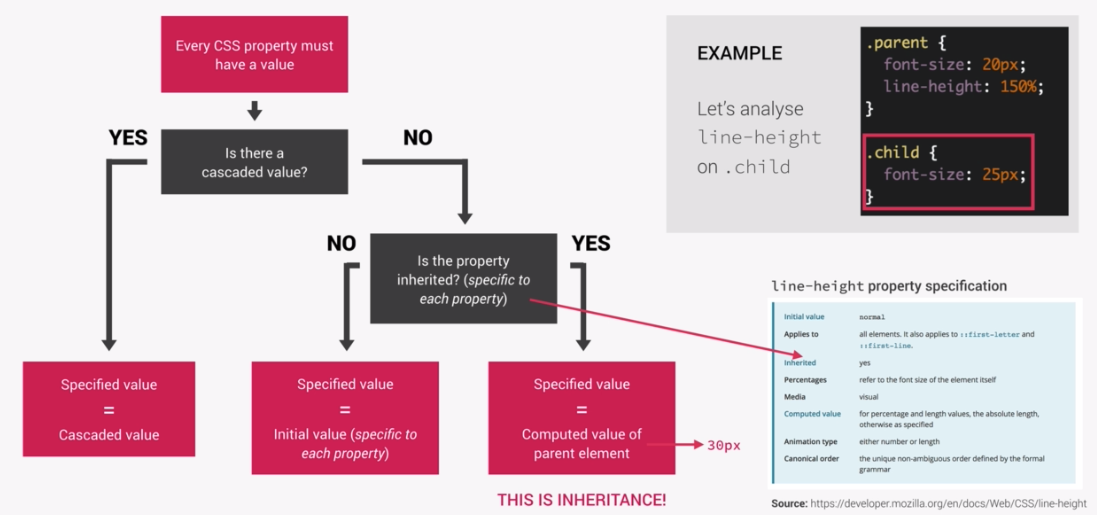
**CSS value parsing, how?**

****

**CSS value processing**

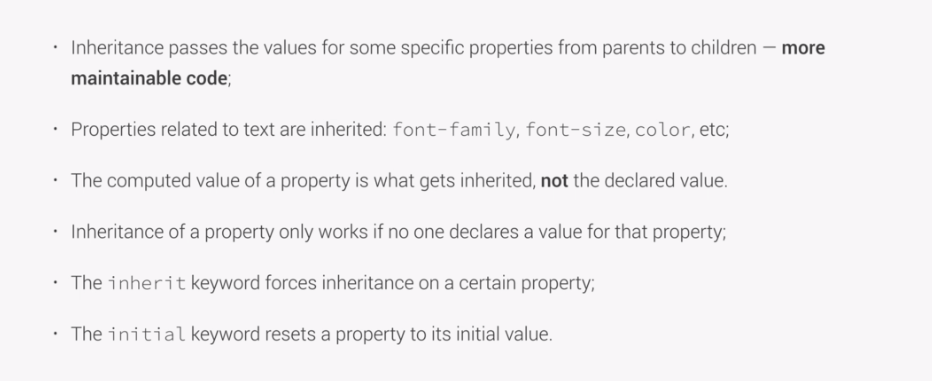
****

****

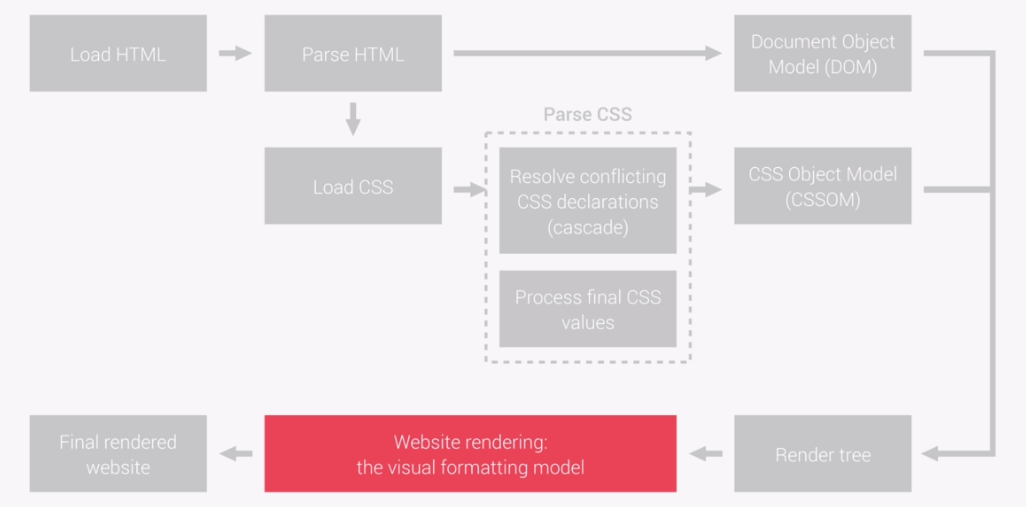
**CSS inheritance**

* line-height is inherited
* .child will use the **computed value** of its parent
* computed value: **150% \* 20px 🡪 30px**
* .child { font-size: 25px;

line-height: 30px; }

****

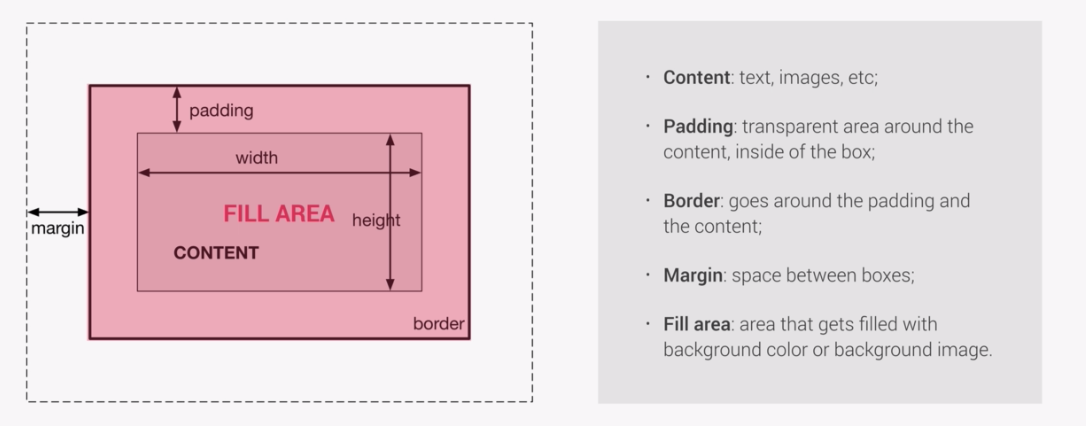
**not inherited: margin, padding, box-sizing**

**Website Rendering**

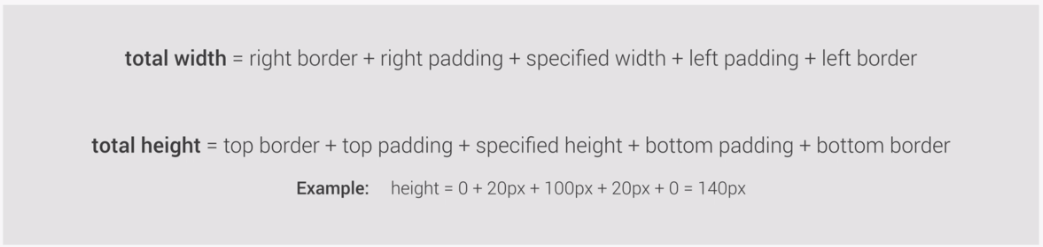
CSS formatting model (foundamental): **algorithm** calculate **boxes** and determinate the **layout** of the **boxes**

* dimensions of boxes: the box model
* box-type (display property):
  + inline
  + block
  + inline-block
* positioning scheme: floats and positioning
* stacking contexts

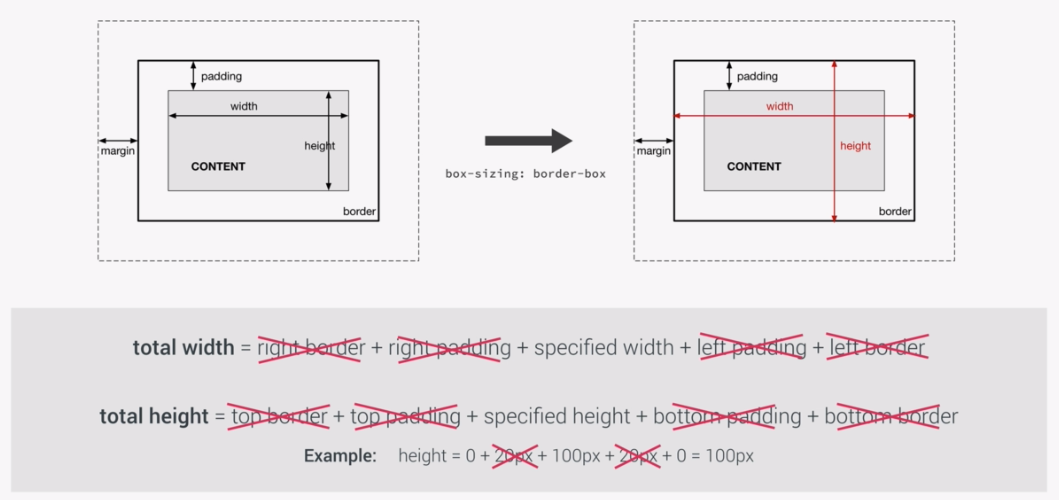
1. The box model (block level)



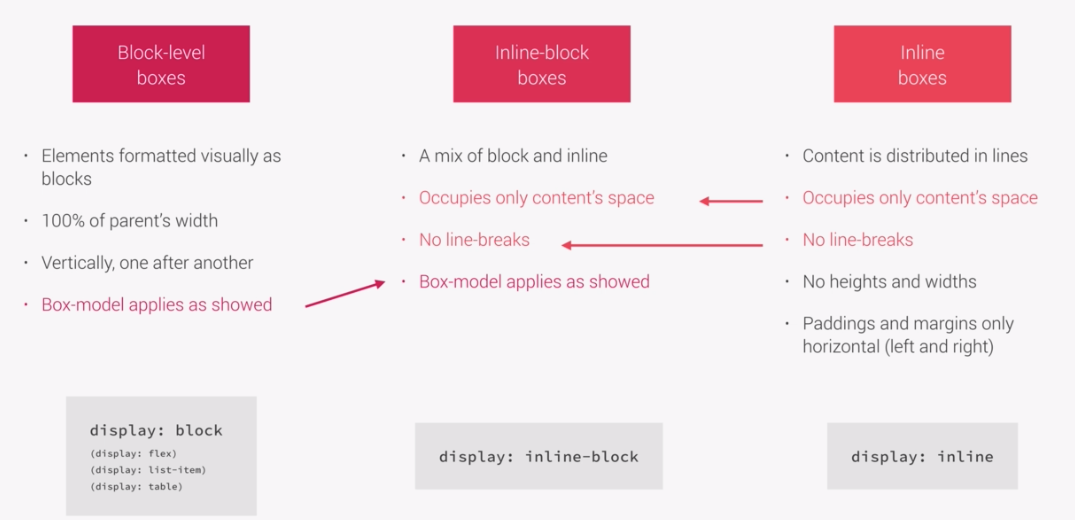
* 1. calculate the height and width: content-box



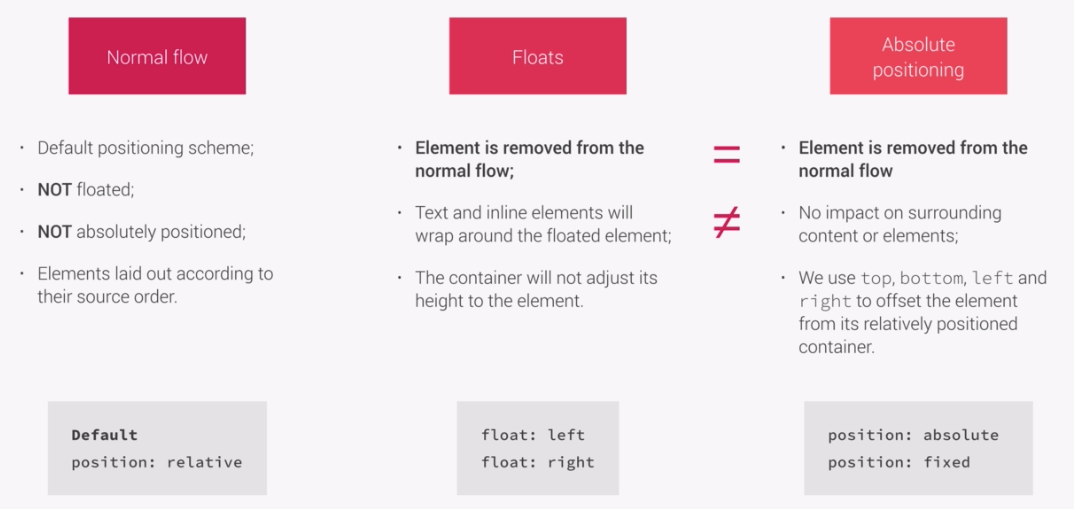
1. calculate the height and width: border-box!!! EASIER!!!



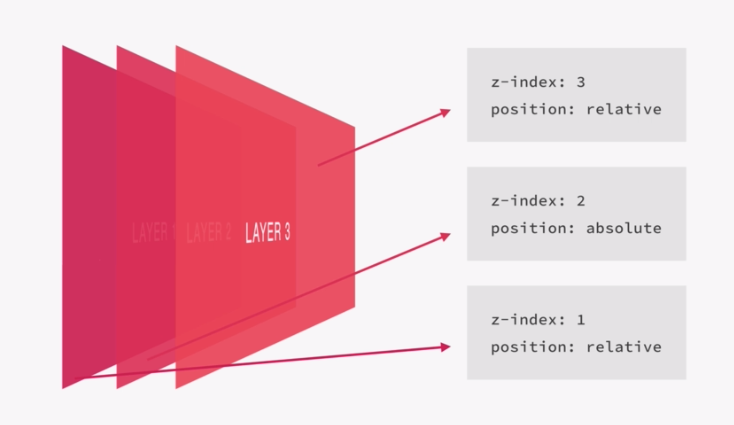
1. Box-type



1. Positioning Scheme



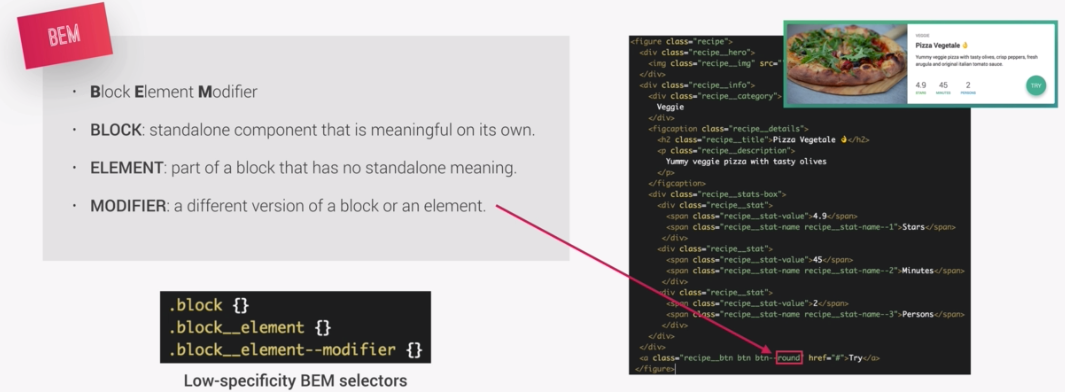
1. Stacking Contexts (z-index)



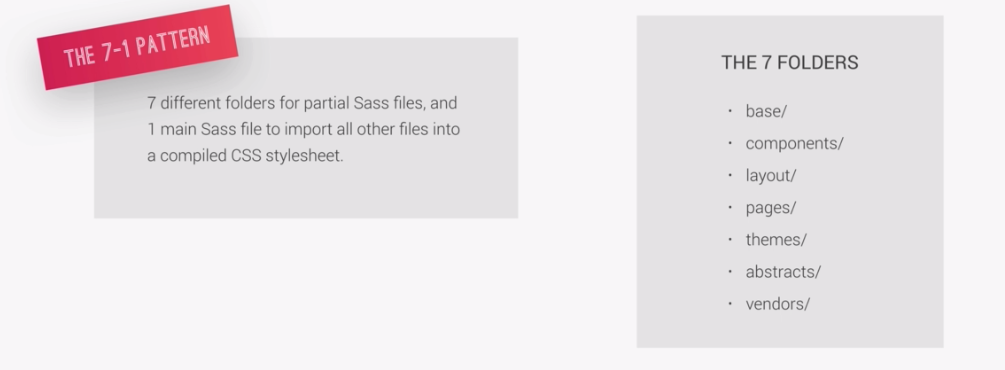
CSS Architecture, components and BEM

think 🡪 build 🡪 architect

* think
* build: **class** **naming rules**
  + BEM: Block Element Modifier



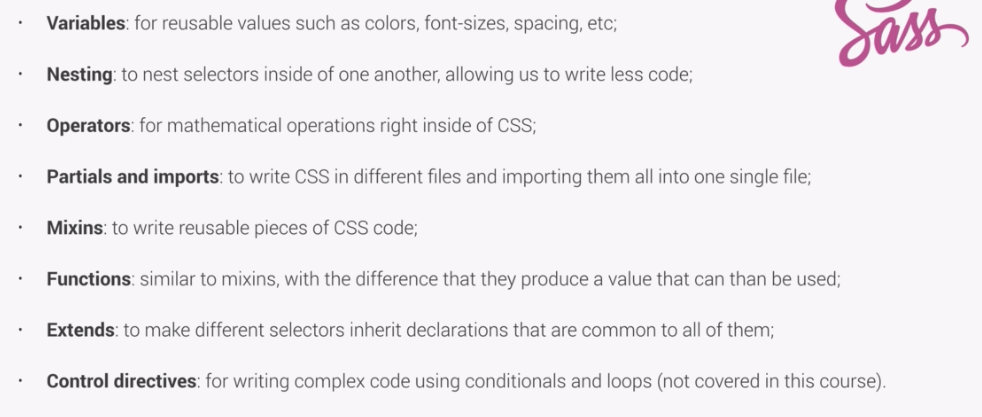
* architect
  + 7-1 pattern



SASS 🡪 CSS preprocessor

SASS source code 🡪 SASS compiler 🡪 compiled CSS code

SASS VS. SCSS: different syntax; we use **SCSS**



lec24,25: Variable, Nesting, Mixin, Extends, Functions

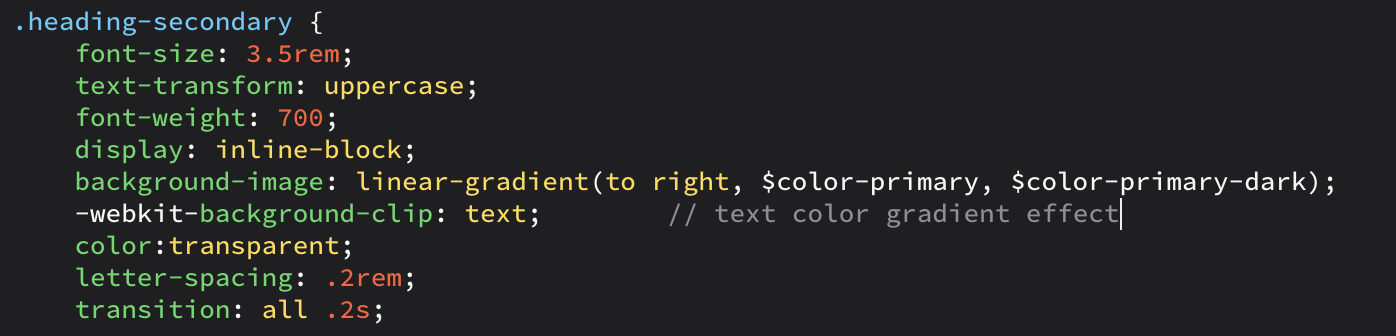
LEC34: How to architect and build simple Grid system (selector + :not pseudo-class + calc())

* Grid sys: consistent design system
* calc(): mix units and do calculation
* selector: []
  + [class^="col-"] # start with col-
  + [class\*="col-"] # contains col-
  + [class$="col-"] # end with col-

**Always define the width of pictures!! in percentage!!**

**text/font gradient effect:**

****

****

**Selector: direct children**

& > \* {

transform: skewY(7deg); // transform only direct children back

}

Position: transform VS. margin

* float type: transform: translateX(-3rem)
* block/inline block: margin: 3rem auto

background Pic/Video fit:

* pic: background-size: cover OR background-size: 100%
* video: height: 100% + width:100% + object-fit: cover

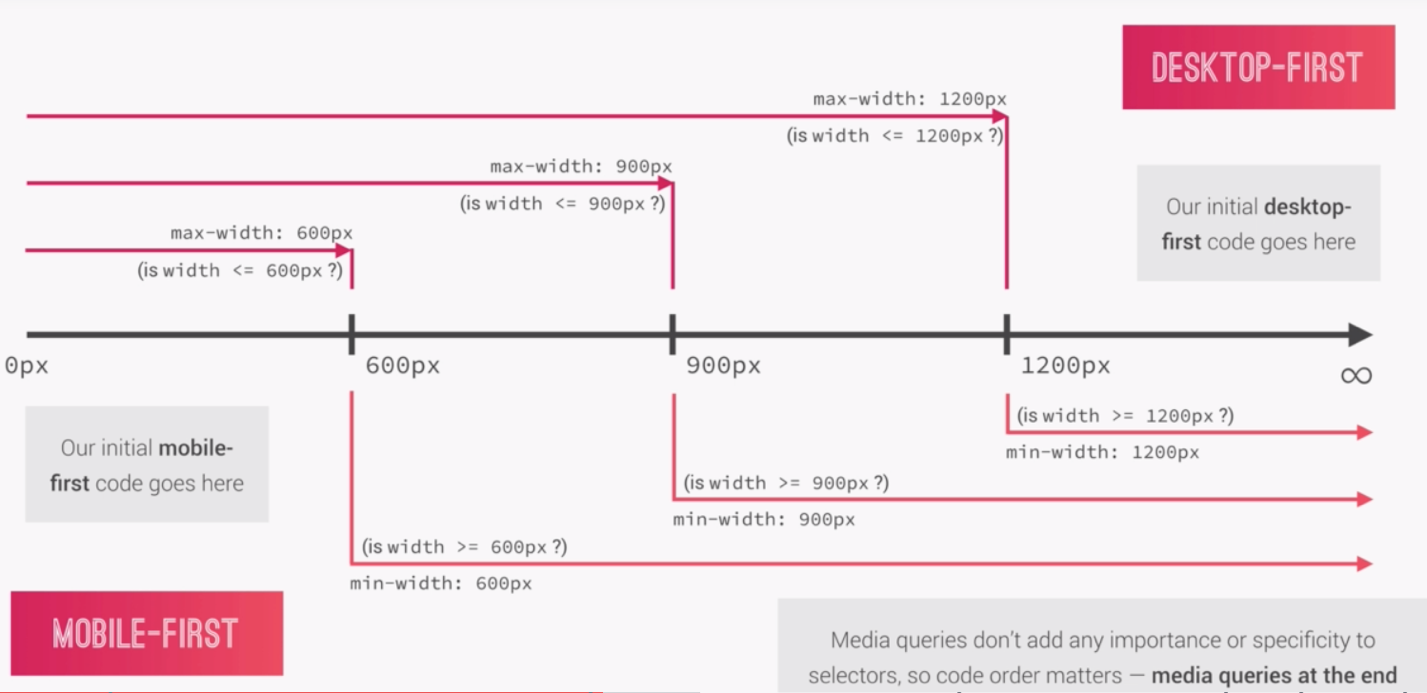
CSS sibling selector:

+ adjacent sibling selector

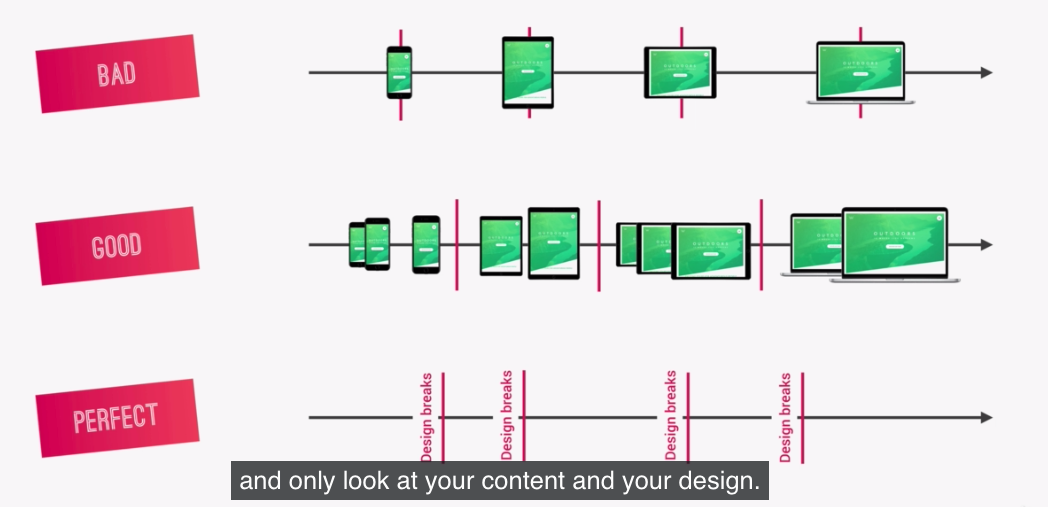
~ general sibling selector

**8.18 Responsive Web Design**

* mobile-first VS. desktop-first



* selecting **breakpoints**:
  1. good way: look at all most-used devices and choose breakpoint from that
  2. perfect way: ignore devices parameter, only look at your content and design



* ORDER:
  1. order matters a lot: the **last rule** meets the requirement will apply; not the closest one
* How to write media mixins:
  1. use em for media query, not rem!!!
  2. order: Base + typography > layout + grid > page layout > components