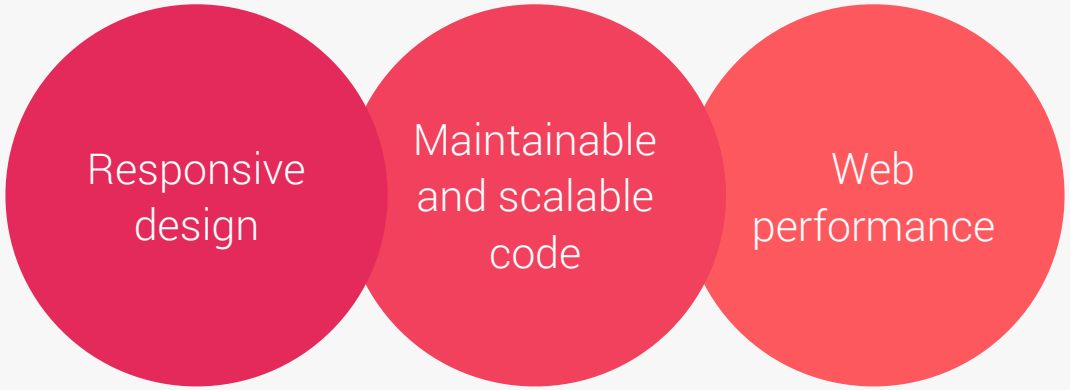
THREE PILLARS TO WRITE GOOD HTML AND CSS... AND BUILD GOOD WEBSITES



- Fluid layouts
- Media queries
- Responsive images
- Correct units
- Desktop-first vs mobile-first

- Clean
- Easy-to-understand
- Growth
- Reusable
- How to organize files
- How to name classes
- How to structure HTML

- Less HTTP requests
- Less code
- Compress code
- Use a CSS preprocessor
- Less images
- Compress images

JONAS.IO
SCHMEDTMANN

ADVANCED CSS AND SASS

TAKE YOUR CSS TO THE NEXT LEVEL!

@JONASSCHMEDTMAN

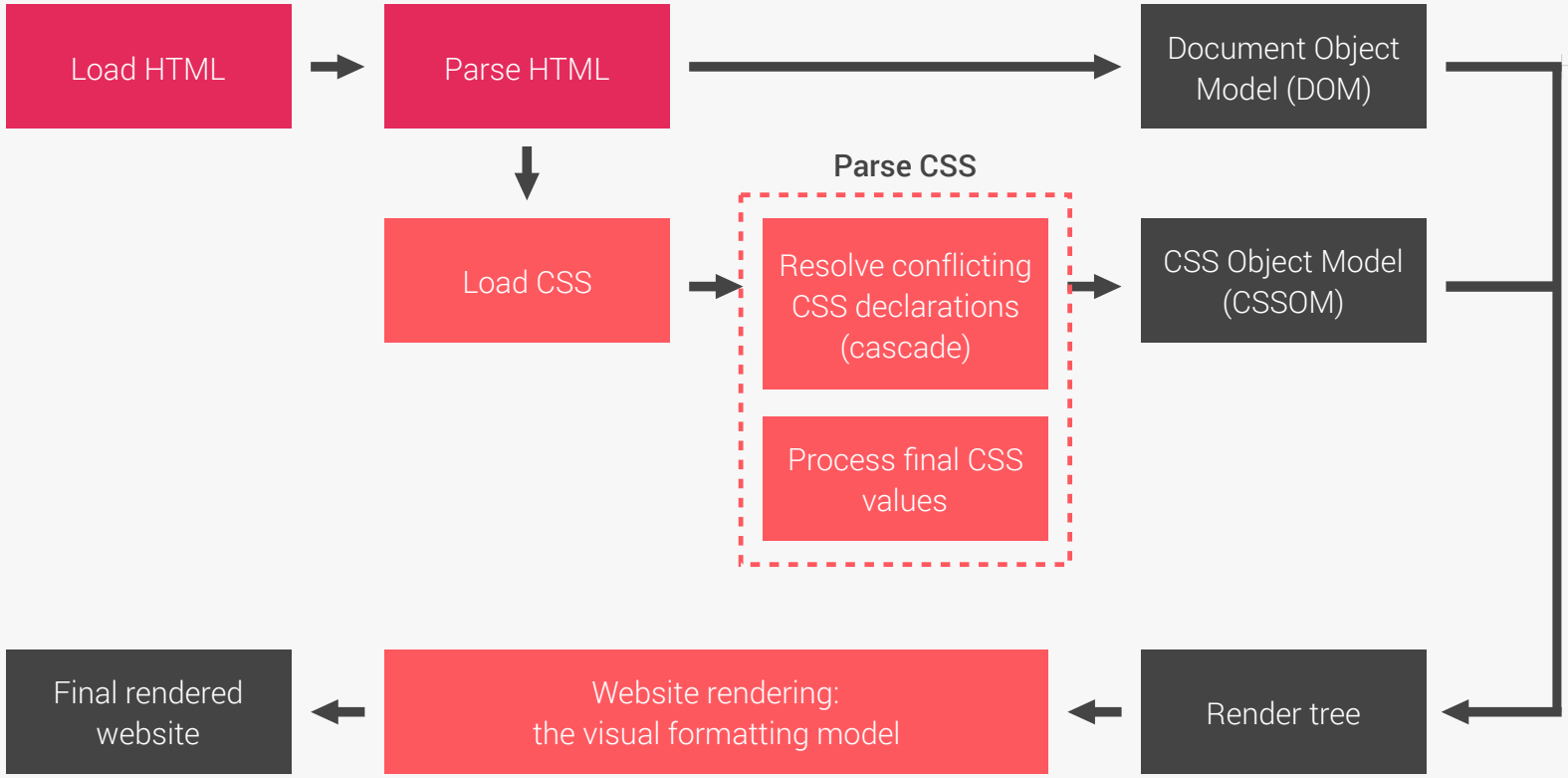
SECTION

HOW CSS WORKS: A LOOK BEHIND THE SCENES

LECTURE

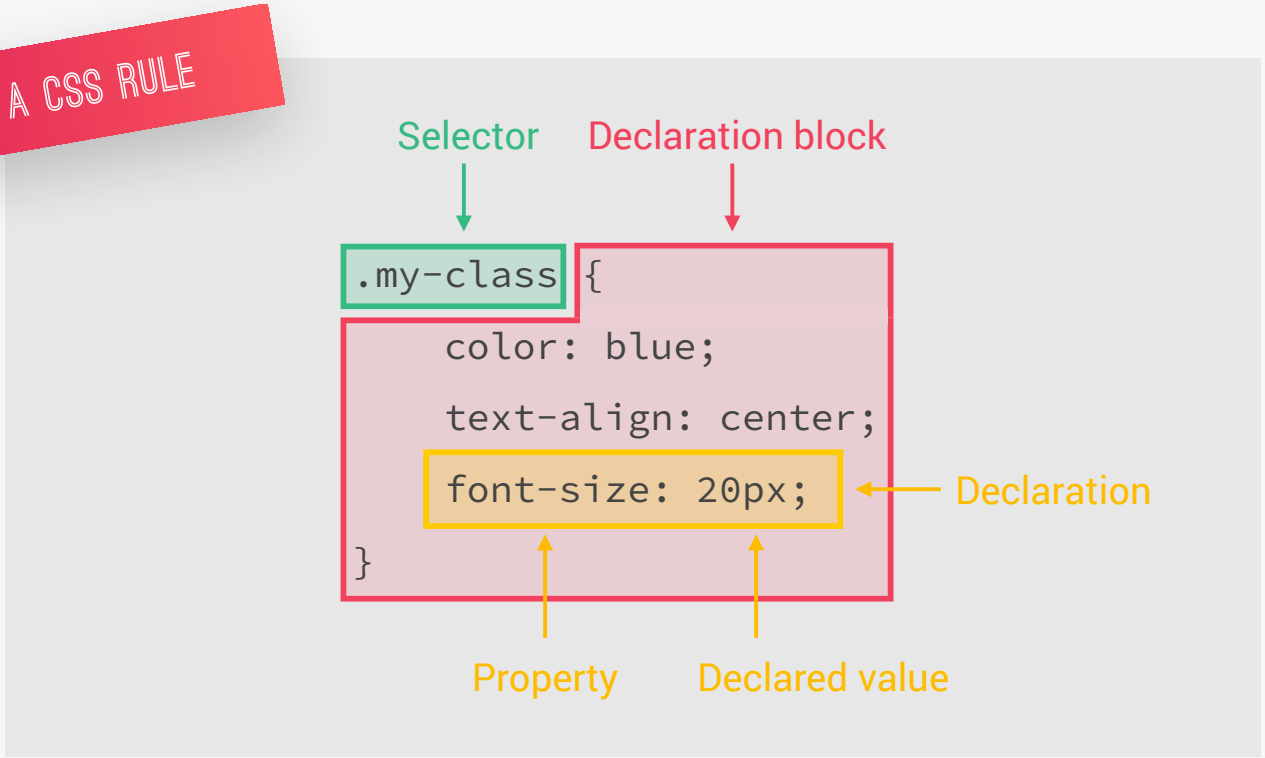
HOW CSS WORKS BEHIND THE SCENES: AN OVERVIEW

WHAT HAPPENS TO CSS WHEN WE LOAD UP A WEBPAGE?



QUICK REVIEW: CSS TERMINOLOGY

A CSS RULE



THE CASCADE (THE "C" IN CSS)

CASCADE

Process of combining different stylesheets and resolving conflicts between different CSS rules and declarations, when more than one rule applies to a certain element.

CSS can come from difference sources:

- Author
- User
- Browser (user agent)

User agent css: Eg anchor tag where the link is blue and underline

IMPORTANCE (WEIGHT) > SPECIFICITY > SOURCE ORDER

IMPORTANCE > SPECIFICITY > SOURCE ORDER

1. User `!important` declarations
2. Author `!important` declarations
3. Author declarations
4. User declarations
5. Default browser declarations

Same importance?

When we have conflicting declaration with same importance we calculate the specificity

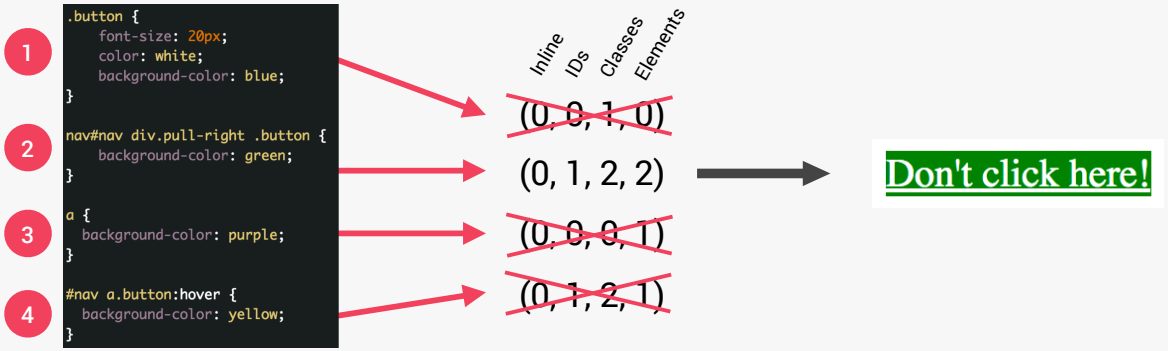
1. Inline styles
2. IDs
3. Classes, pseudo-classes, attribute
4. Elements, pseudo-elements

Same specificity?

The last declaration in the code will override all other declarations and will be applied.

Important keyword:
We have two rules for background-color

But the blue background color has ! important, therefore it gets precedence



The specificity is actually not just one number, but for each category.
For each of this, we count the number of occurrences in the selector.

CASCADE AND SPECIFICITY: WHAT YOU NEED TO KNOW

- CSS declarations marked with `!important` have the highest priority;
- But, only use `!important` as a last resource. It's better to use correct specificities — **more maintainable code**!
- Inline styles will always have priority over styles in external stylesheets;
- A selector that contains **1** ID is more specific than one with **1000** classes;
- A selector that contains **1** class is more specific than one with **1000** elements;
- The universal selector `*` has no specificity value (0, 0, 0, 0);
- Rely more on **specificity** than on the **order** of selectors;
- But, rely on order when using 3rd-party stylesheets — always put your author stylesheet last.

Each time you use a unit other than PX, it will get converted to PX

Declared value/Cascaded value:
140px and 66% are actually conflicting, so the cascade is used to figure out which rule will apply.
The `.amazing` class selector will be applied as its more specific then the `p` element, so thats the Cascaded Value

Specified value:
This one is the default value of a certain CSS property. In this case it is not really relevant because we have a cascaded value already

Computed value:
Values with relative units are converted to PX, so they can be inherited. Also, CSS treats orange, oral, bolder..etc are computed and replaced at this step In this case the %, which is not a unit, so nothing happens

Used value/Actual Value:
The CSS engine uses the rendered layout to figure out the remaining value. For eg the % value that depends on the layout. the 66% is in relation to its parent element. So the parser needs to know that width in order to calculate the paragraph width. 66% of 280px is 184.8.px. Browser cannot render too specific 184.8, so it is rounded to 185

HOW CSS VALUES ARE PROCESSED

	width (paragraph)	padding (paragraph)	font-size (root)	font-size (section)	font-size (paragraph)
1. Declared value (author declarations)	140px 66%	–	–	1.5rem	–
2. Cascaded value (after the cascade)	66%	–	16px (Browser default)	1.5rem	–
3. Specified value (defaulting, if there is no cascaded value)	66%	0px (Initial value)	16px	1.5rem	24px
4. Computed value (converting relative values to absolute)	66%	0px	16px	24px (1.5 * 16px)	24px
5. Used value (final calculations, based on layout)	184.8px	0px	16px	24px	24px
6. Actual value (browser and device restrictions)	185px	0px	16px	24px	24px

(66% of 280px)

INHERITANCE

```
<div class="section">
  <p class="amazing">CSS is absolutely amazing</p>
</div>
```

```
.section {
  font-size: 1.5rem;
  width: 280px;
  background-color: orangered;
}

p {
  width: 140px;
  background-color: green;
}

.amazing {
  width: 66%;
}
```

CSS is absolutely amazing

(Let's analyse the green paragraph)

Padding: But each css property has an initial value, even if we don't declare it. Different property has different initial value, eg; Padding is 0px

Font-Size(root):
The overall fontSize of the document.No declared value, but browser has a default value 16px, so this is our cascaded value. The fotSize is a user-agent declaration.

Section:
Now we actually have a declared value of 1.5rem. Since its a relative unit it needs to be converted by the engine. The cascaded and specified value but the computed value is 24px The rem unit is always relative to the root fontSize which is 16px.

Font-Size (P):
There is declared value, no default or initial value. Mechanism called inheritance. Some properties (related to text such as font-size) inherited the computed value of their parent element.

HOW UNITS ARE CONVERTED FROM RELATIVE TO ABSOLUTE (PX)

		Example (x)	How to convert to pixels	Result in pixels
	% (fonts)	150%	x% * parent's computed font-size	24px
	% (lengths)	10%	x% * parent's computed **width**	100px
Font-based	em (font)	3em	x * **parent** computed font-size	72px (3 * 24)
Font-based	em (lengths)	2em	x * **current element** computed font-size	48px
Font-based	rem	10rem	x * **root** computed font-size	160px
Viewport-based	vh	90vh	x * 1% of viewport height	90% of the current viewport height
Viewport-based	vw	80vw	x * 1% of viewport width	80% of the current viewport width

4. Computed value
(converting relative values to absolute)

```
html, body {
  font-size: 16px;
  width: 80vw;
}

header {
  font-size: 150%;
  padding: 2em;
  margin-bottom: 10rem;
  height: 90vh;
  width: 1000px;
}

.header-child {
  font-size: 3em;
  padding: 10%;
}
```

Percentages:
There is a difference between using % for fonts or for length/ distance/measurement.
Body has 16px, so its 150% x 16 = 24px

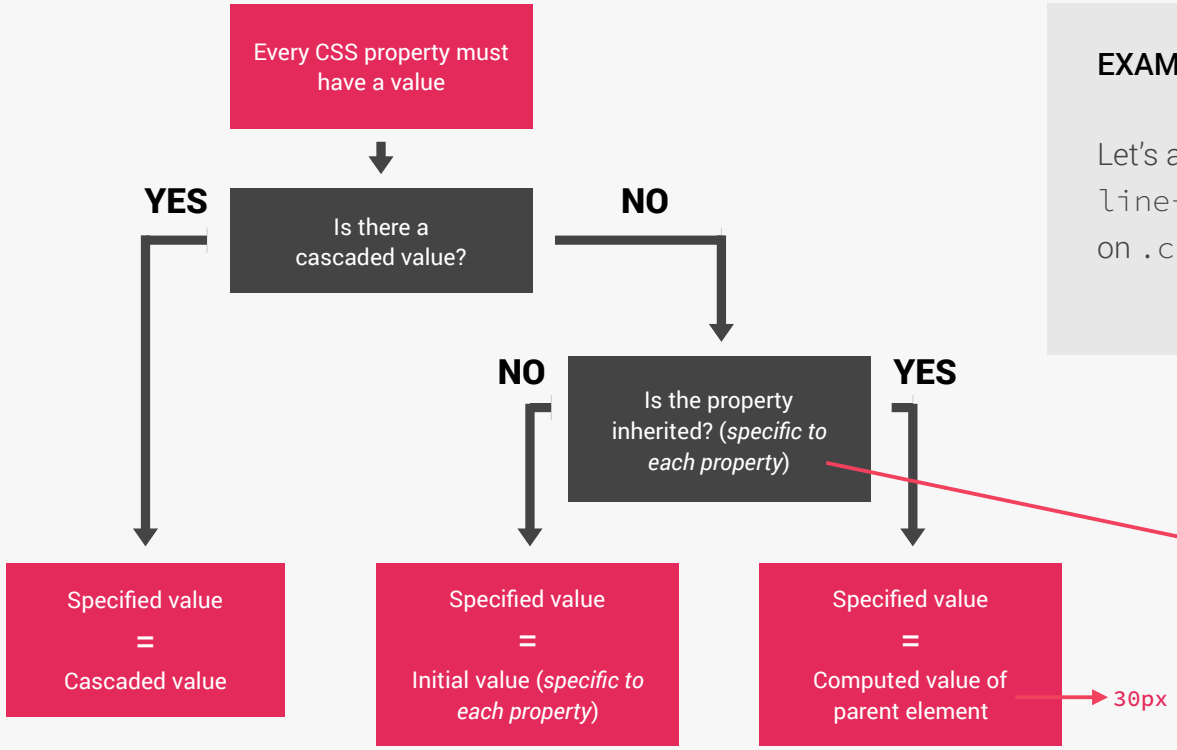
Length %:
When we express a length in % such as a height, padding, margin.. the reference is always the parent's element width.
So the padding of 10% in the header-child results to 100px due to the header width 100px

em and rems:
Different when we use ems for font and for length.
Both em/rem are font based. The difference is em used the parent/current element as reference while rem uses the root fontSize as reference.
For em length, 2 x 24px, as the current element fontSize is 24px.
rems: Works the same for font/length, so its 10x24 = 16ppx

CSS VALUE PROCESSING: WHAT YOU NEED TO KNOW

- Each property has an initial value, used if nothing is declared (and if there is no inheritance — see next lecture);
- Browsers specify a **root `font-size`** for each page (usually `16px`);
- Percentages and relative values are always converted to pixels;
- Percentages are measured relative to their parent's **`font-size`**, if used to specify `font-size`;
- Percentages are measured relative to their parent's **`width`**, if used to specify lengths;
- `em` are measured relative to their **parent** `font-size`, if used to specify `font-size`;
- `em` are measured relative to the **current** `font-size`, if used to specify lengths;
- `rem` are always measured relative to the **document's root** `font-size`;
- `vh` and `vw` are simply percentage measurements of the viewport's `height` and `width`.

INHERITANCE IN CSS



THIS IS INHERITANCE!

EXAMPLE

Let's analyse `line-height` on `.child`

```
.parent {
  font-size: 20px;
  line-height: 150%;
}

.child {
  font-size: 25px;
}
```

`line-height` property specification

Initial value	normal
Applies to	all elements. It also applies to `::first-letter` and `::first-line`.
Inherited	yes
Percentages	refer to the font size of the element itself
Media	visual
Computed value	for percentage and length values, the absolute length, otherwise as specified
Animation type	either number or length
Canonical order	the unique non-ambiguous order defined by the formal grammar

Source: https://developer.mozilla.org/en/docs/Web/CSS/line-height

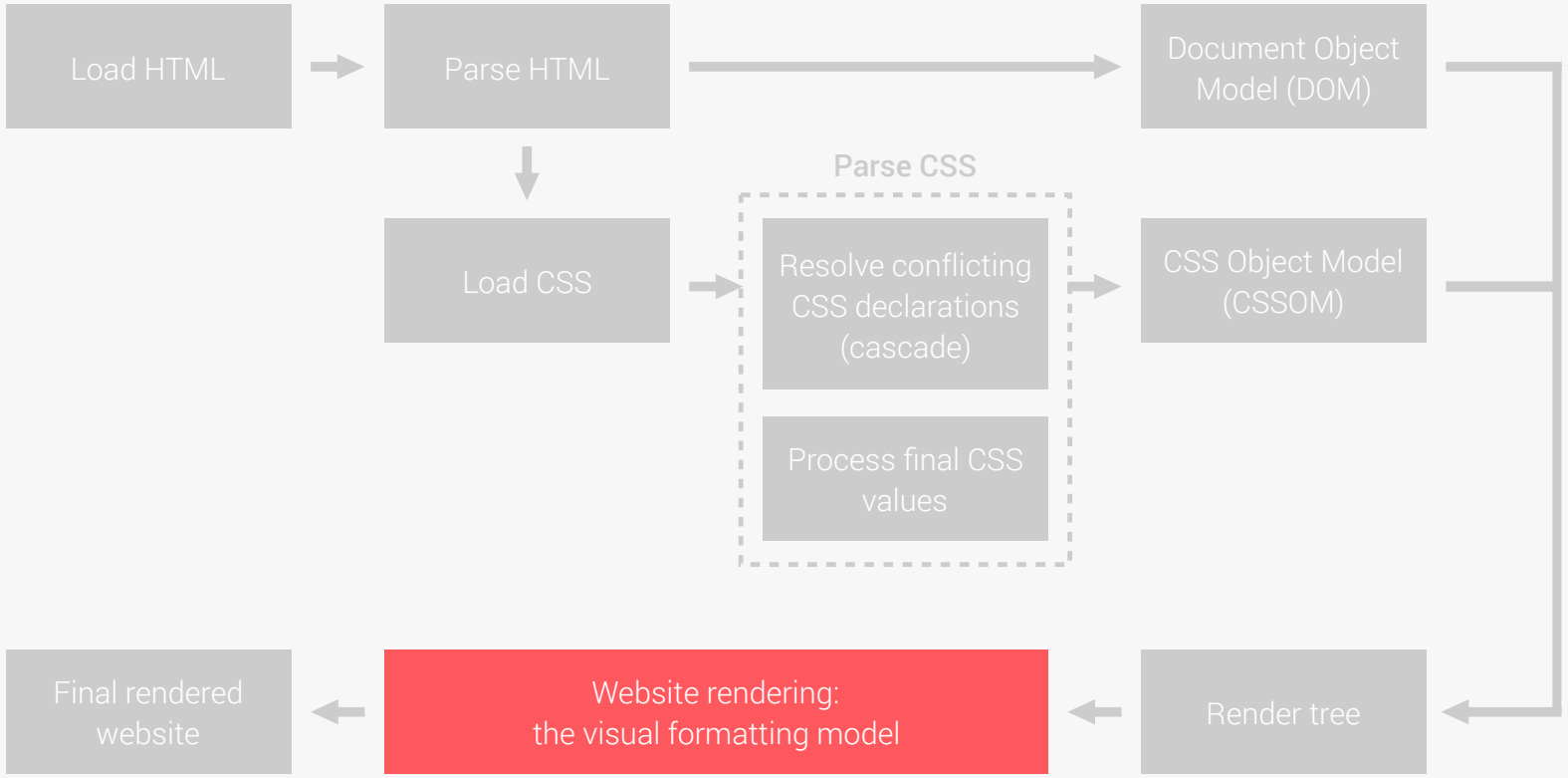
30px because its the line-height 150 % times the .parent font-size 20px.

150% x 20 = 30px

INHERITANCE: WHAT YOU NEED TO KNOW

- Inheritance passes the values for some specific properties from parents to children — **more maintainable code**;
- Properties related to text are inherited: `font-family`, `font-size`, `color`, etc;
- The computed value of a property is what gets inherited, **not** the declared value.
- Inheritance of a property only works if no one declares a value for that property;
- The `inherit` keyword forces inheritance on a certain property;
- The `initial` keyword resets a property to its initial value.

REMEMBER...?

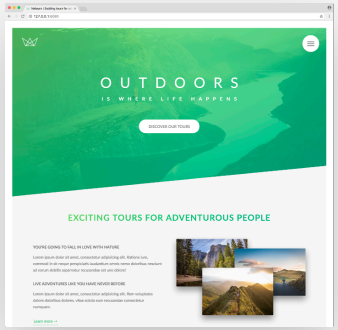

Load HTML
Parse HTML
Document Object Model (DOM)
Parse CSS
Load CSS
Resolve conflicting CSS declarations (cascade)
CSS Object Model (CSSOM)
Process final CSS values
Final rendered website
Website rendering: the visual formatting model
Render tree

THE VISUAL FORMATTING MODEL

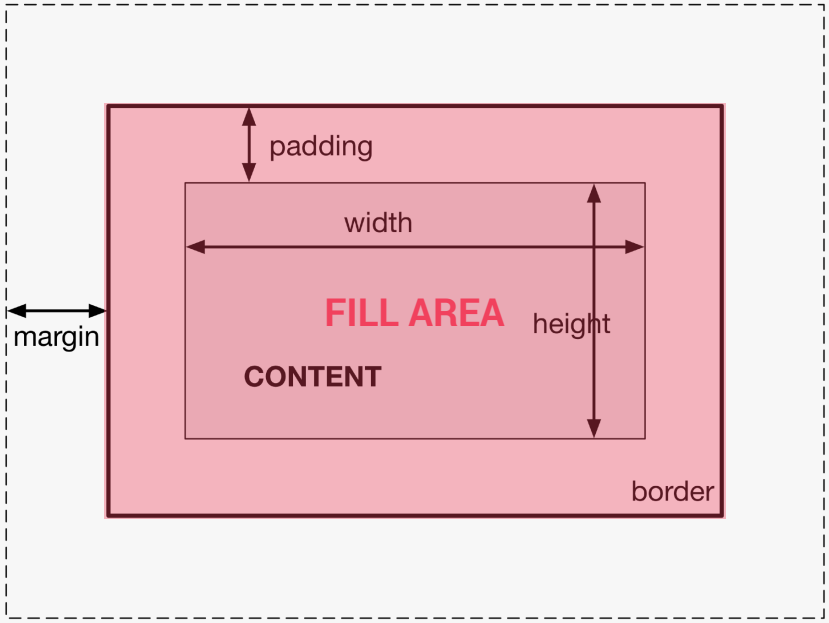
DEFINITION

Algorithm that calculates boxes and determines the layout of theses boxes, for each element in the render tree, in order to determine the final layout of the page.

- **Dimensions of boxes**: the box model;
- **Box type**: inline, block and inline-block;
- **Positioning scheme**: floats and positioning;
- **Stacking contexts**;
- Other elements in the render tree;
- Viewport size, dimensions of images, etc.



1. THE BOX MODEL



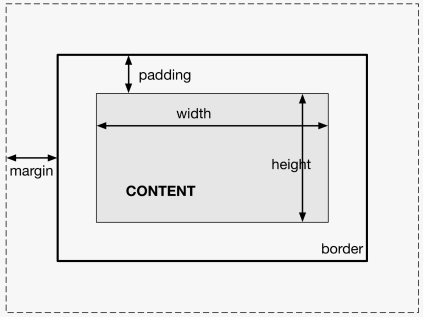
- **Content**: text, images, etc;
- **Padding**: transparent area around the content, inside of the box;
- **Border**: goes around the padding and the content;
- **Margin**: space between boxes;
- **Fill area**: area that gets filled with background color or background image.

1. THE BOX MODEL: HEIGHTS AND WIDTHS

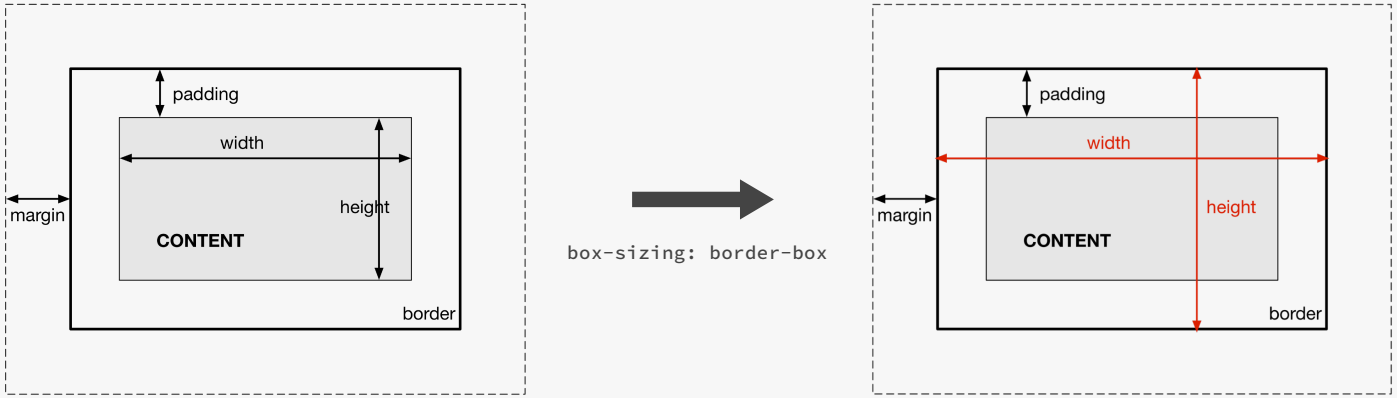
total width = right border + right padding + specified width + left padding + left border

total height = top border + top padding + specified height + bottom padding + bottom border

Example: height = 0 + 20px + 100px + 20px + 0 = 140px



1. THE BOX MODEL WITH BOX-SIZING: BORDER-BOX



total width = ~~right border~~ + ~~right padding~~ + specified width + ~~left padding~~ + ~~left border~~

total height = ~~top border~~ + ~~top padding~~ + specified height + ~~bottom padding~~ + ~~bottom border~~

Example: height = 0 + ~~20px~~ + 100px + ~~20px~~ + 0 = 100px

2. BOX TYPES: INLINE, BLOCK-LEVEL AND INLINE-BLOCK

Block-level boxes

- Elements formatted visually as blocks
- 100% of parent's width
- Vertically, one after another
- Box-model applies as showed

```
display: block
(display: flex)
(display: list-item)
(display: table)
```

Type of a box is always defined by a display property.
All HTML elements has a default display property.
Elements such as paragraph, div..etc are block by default

Inline-block boxes

- A mix of block and inline
- Occupies only content's space
- No line-breaks
- Box-model applies as showed

```
display: inline-block
```

Inline-block boxes are technically inline boxes, but which simply work as a block-level box on the inside.

Inline boxes

- Content is distributed in lines
- Occupies only content's space
- No line-breaks
- No heights and widths
- Paddings and margins only horizontal (left and right)

```
display: inline
```

Opposite of Block-Level boxes

3. POSITIONING SCHEMES: NORMAL FLOW, ABSOLUTE POSITIONING AND FLOATS

Normal flow

- Default positioning scheme;
- **NOT** floated;
- **NOT** absolutely positioned;
- Elements laid out according to their source order.

```
Default
position: relative
```

Floats

- **Element is removed from the normal flow;**
- Text and inline elements will wrap around the floated element;
- The container will not adjust its height to the element.

```
float: left
float: right
```

Absolute positioning

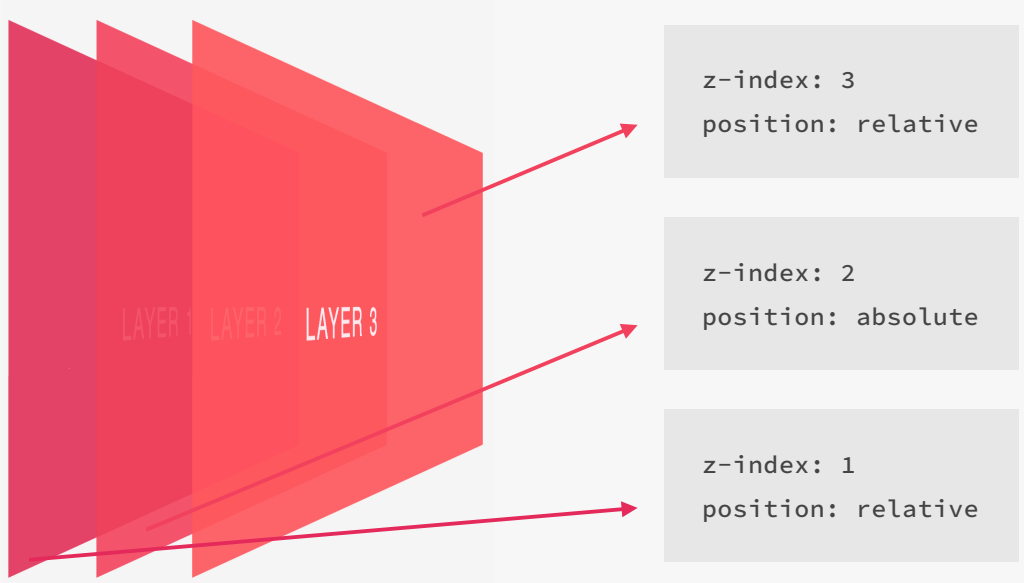
- **Element is removed from the normal flow**
- No impact on surrounding content or elements;
- We use `top`, `bottom`, `left` and `right` to offset the element from its relatively positioned container.

```
position: absolute
position: fixed
```

Floats:
Causes an element to be completely taken out of the normal flow and shifted to the left or right as far as possible until it touches the edge of its containing box, or another floated elements.
When this happens, text and inline elements will wrap around the floated element.
Also when an element is floated, its container will not adjust its height to the element - Solution is `clear-fix`

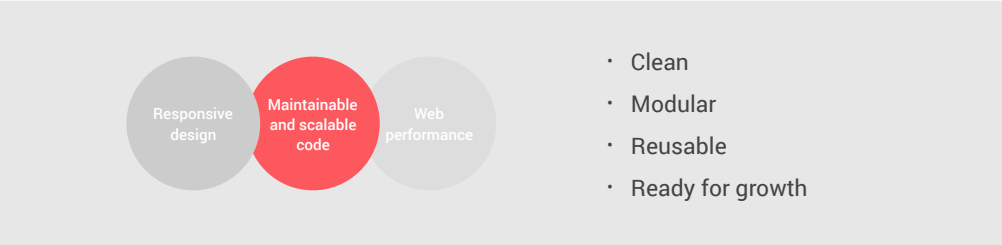
Absolute Positioning:
The element is taken out of the normal flow, the element has no impact on surrounding content or elements, It can even overlap them.
If we want to position an absolutely positioned element` on a page, we use the CSS properties `top`.etc to offset it to its relatively positioned container

4. STACKING CONTEXTS



Stacking contexts are what determine in which order elements are rendered on the page.
A new stacking context can be created by a different CSS properties, (Z index).
Stacking contexts are like layers,

THE THINK - BUILD - ARCHITECT MINDSET



THINK

Think about the layout of your webpage or web app before writing code.

BUILD

Build your layout in HTML and CSS with a consistent structure for naming classes.

ARCHITECT

Create a logical **architecture** for your CSS with files and folders.

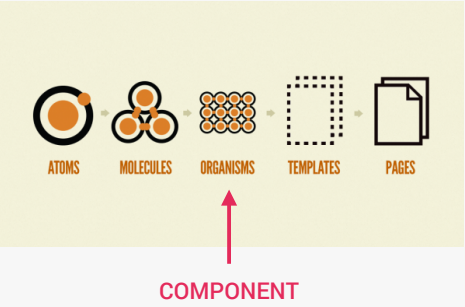
THINKING ABOUT THE LAYOUT

THINK → BUILD → ARCHITECT

COMPONENT-DRIVEN DESIGN

- **Modular building blocks** that make up interfaces;
- Held together by the **layout** of the page;
- **Re-usable** across a project, and between different projects;
- **Independent**, allowing us to use them anywhere on the page.

ATOMIC DESIGN



[Image taken from bradfrost.com]

BUILDING WITH MEANINGFUL CLASS NAMES

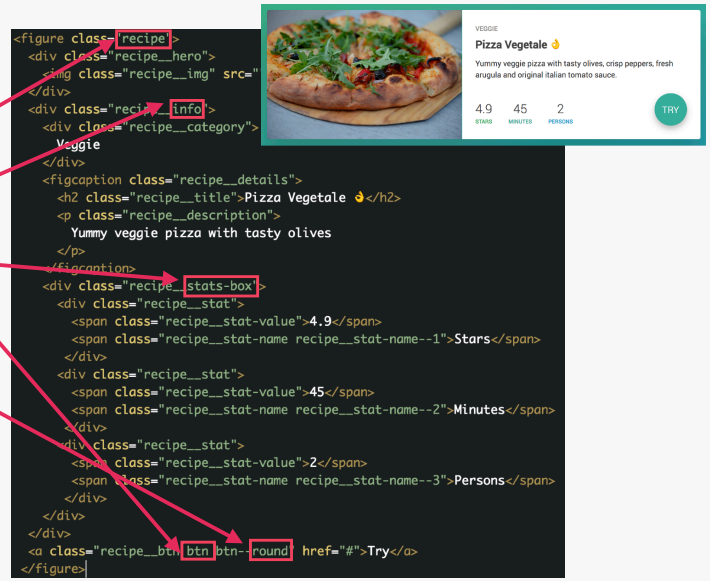
THINK → **BUILD** → ARCHITECT

BEM

- **B**lock **E**lement **M**odifier
- **BLOCK**: standalone component that is meaningful on its own.
- **ELEMENT**: part of a block that has no standalone meaning.
- **MODIFIER**: a different version of a block or an element.

```
.block {}
.block__element {}
.block__element--modifier {}
```

Low-specificity BEM selectors



ARCHITECTING WITH FILES AND FOLDERS

THINK → BUILD → ARCHITECT

THE 7-1 PATTERN

7 different folders for partial Sass files, and
1 main Sass file to import all other files into
a compiled CSS stylesheet.

THE 7 FOLDERS

- base/
- components/
- layout/
- pages/
- themes/
- abstracts/
- vendors/