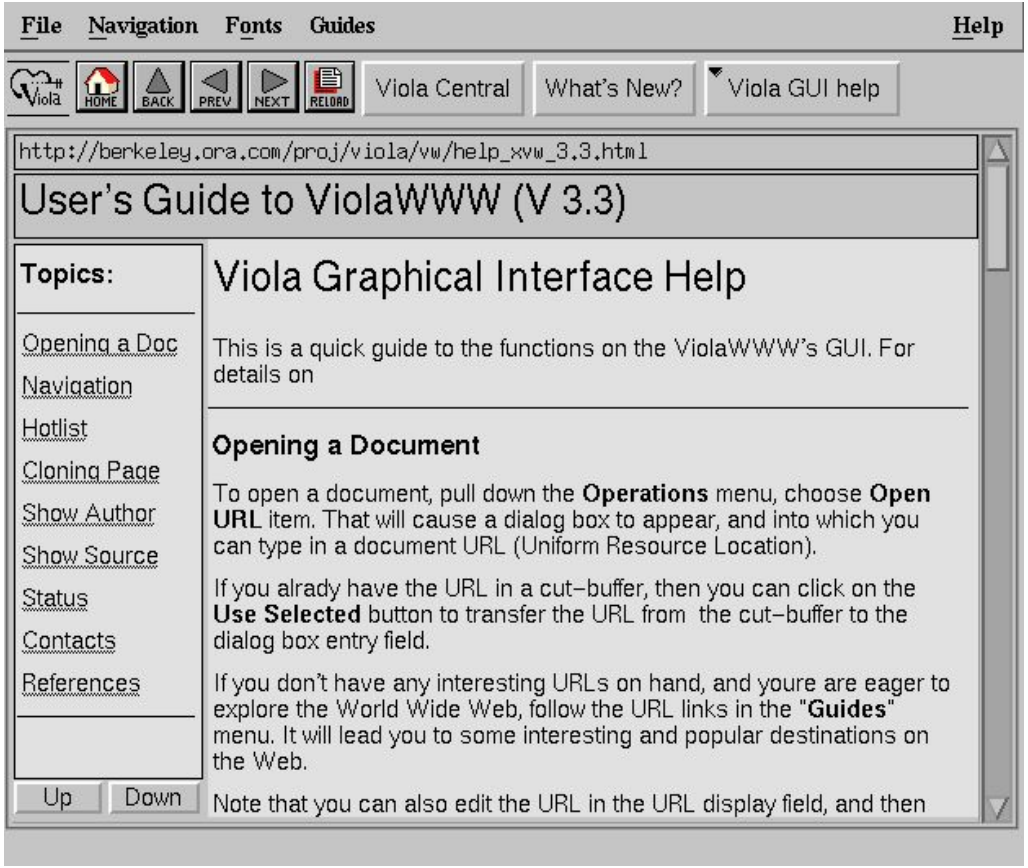
How to structure, scale and maintain CSS

Methodologies that helps with styling applications

11. 03. 2022

cloudtalk

A little bit of history



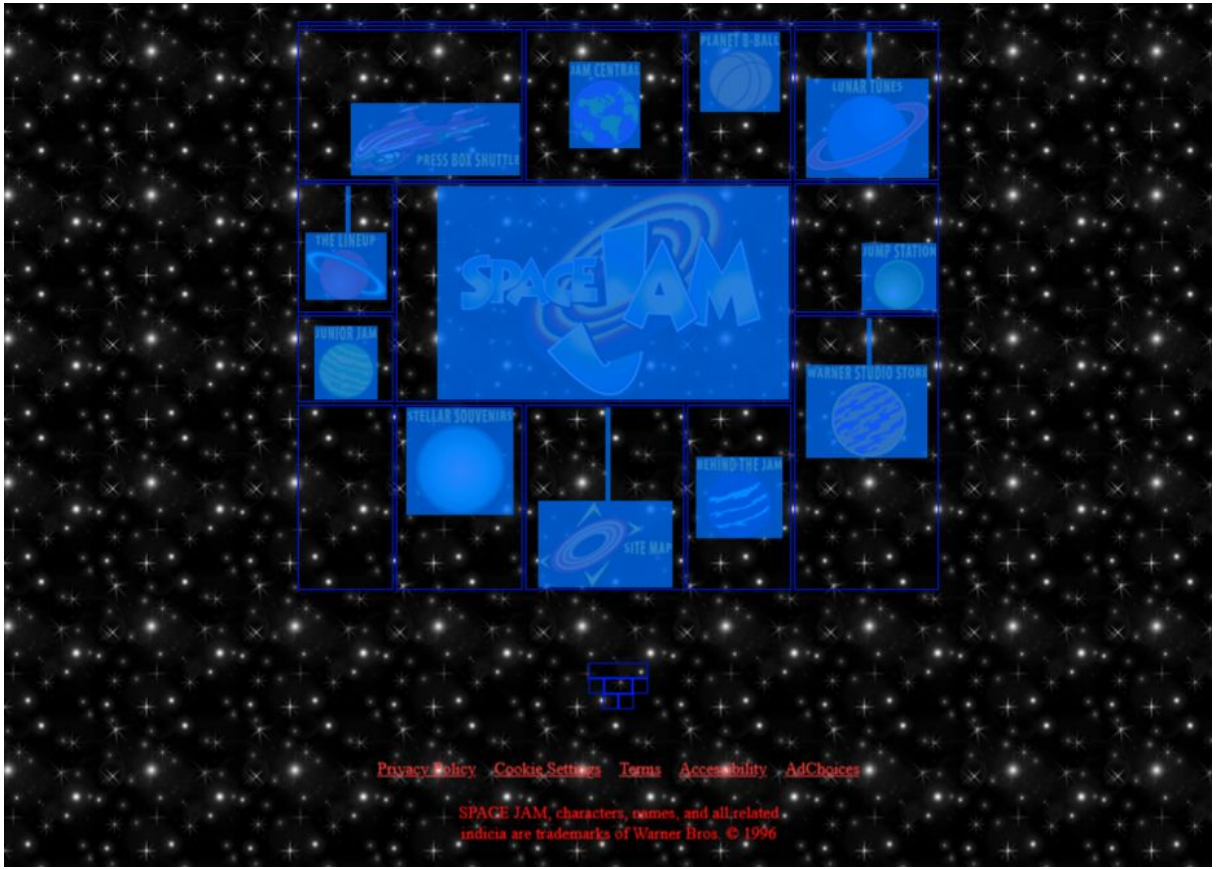
Space Jam


PLANET B-BALL
JAM CENTRAL
LUNAR TUNES
PRESS BOX SHUTTLE
THE LINEUP
SPACE JAM
JUMP STATION
JUNIOR JAM
WARNER STUDIO STORE
STELLAR SOUVENIRS
BEHIND THE JAM
SITE MAP
Privacy Policy
Cookie Settings
Terms
Accessibility
AdChoices
SPACE JAM, characters, names, and all related indicia are trademarks of Warner Bros. © 1996

How we style web sites

```
<body bgcolor="#000000" text="#ff0000" link="#ff4c4c" vlink="#ff4c4c"
alink="#ff4c4c">
```

Layout


PLANET B-BALL
JAM CENTRAL
LUNAR TUNES
PRESS BOX SHUTTLE
THE LINEUP
SPACE JAM
JUMP STATION
JUNIOR JAM
WARNER STUDIO STORE
STELLAR SOUVENIRS
BEHIND THE JAM
SITE MAP
Privacy Policy
Cookie Settings
Terms
Accessibility
AdChoices
SPACE JAM, characters, names, and all related indicia are trademarks of Warner Bros. © 1996

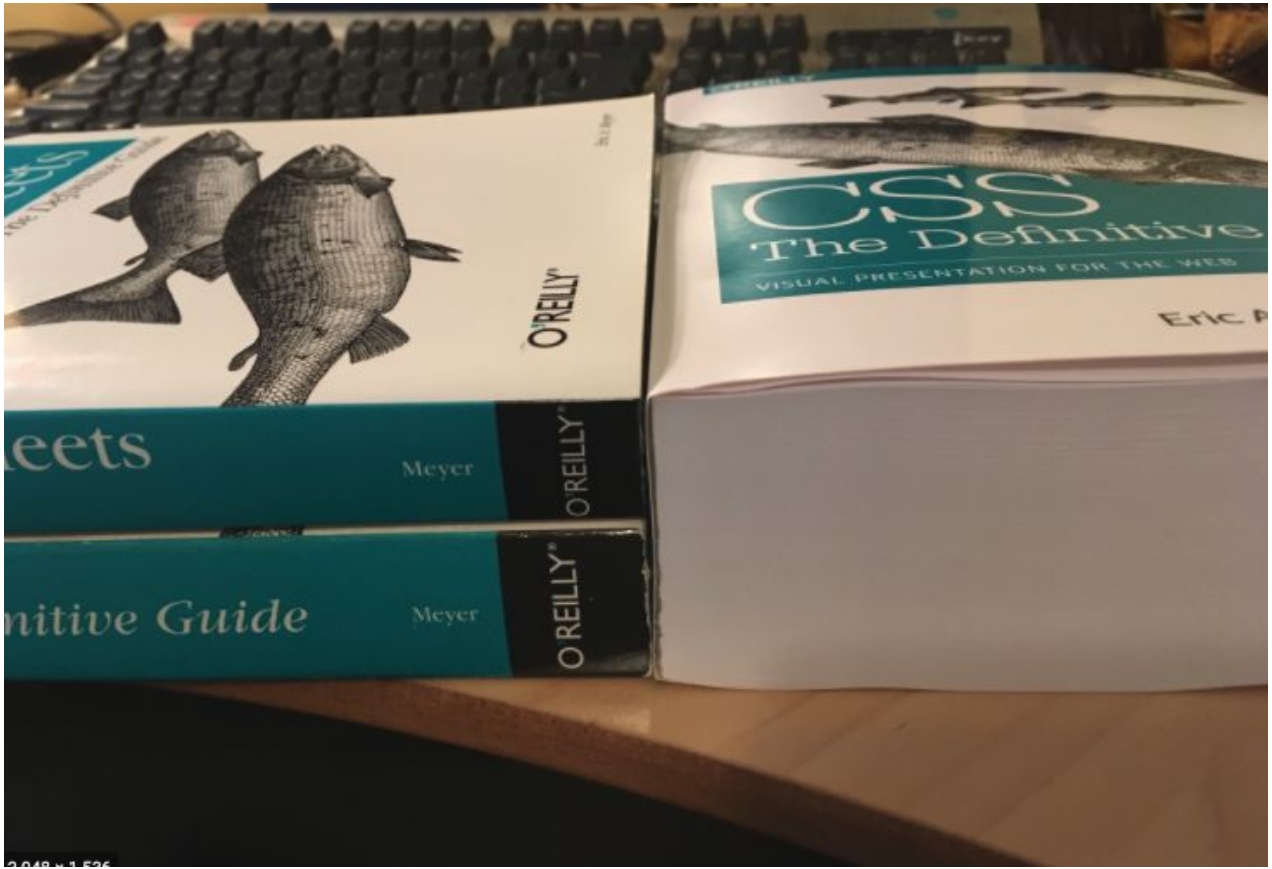
Tables for everything

```
<table width=500 border=0>
<TR>
<TD colspan=5 align=right valign=top>
</td></tr>
<tr>
<td colspan=2 align=right valign=middle>
<br>
<br>
<br>
<center>
<a href="cmp/pressbox/pressboxframes.html"><img src="img/p-
pressbox.gif" height=56 width=131 alt="Press Box Shuttle" border=0>
</a>
</center>
</td>
...
</tr>
...
</table>
```

Today, we have layout properties

- ➔ Float (before flexbox come into game)
- ➔ Flexbox
- ➔ Grid
- ➔ Subgird
- ➔ etc.

So, CSS is improving every day



Writing good CSS is hard

“CSS: the only language that is both so easy it’s not worth learning but also so hard that it’s not worth learning.”

So why is CSS so hard to maintain?

Why?

- ➔ Deep nesting of selectors and high specificity
- ➔ Nesting Hell
- ➔ Cascade
- ➔ Inheritance
- ➔ **Misunderstanding of how CSS works**

Let's take a quick simplest example

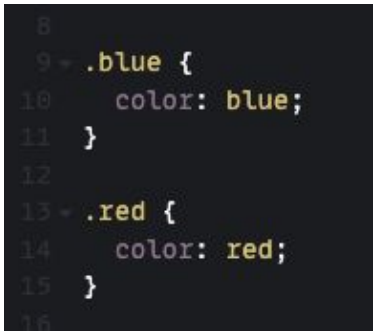
```css
.red {
  color: red;
}

.blue {
  color: blue;
}
```

```html
<div class="red blue">Hello</div>
<div class="blue red">World</div>
```

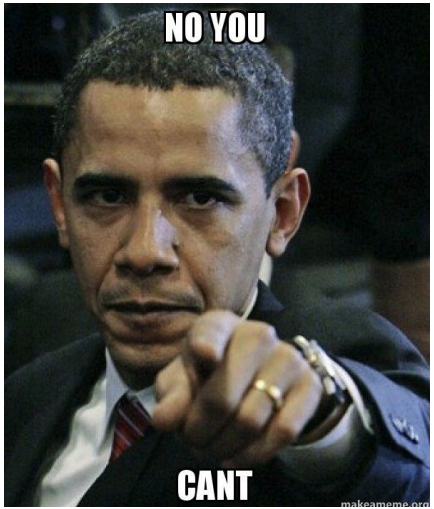
https://codepen.io/ondrejko/pen/xxpKRyo

That means that order in the resulting file matters!

≠

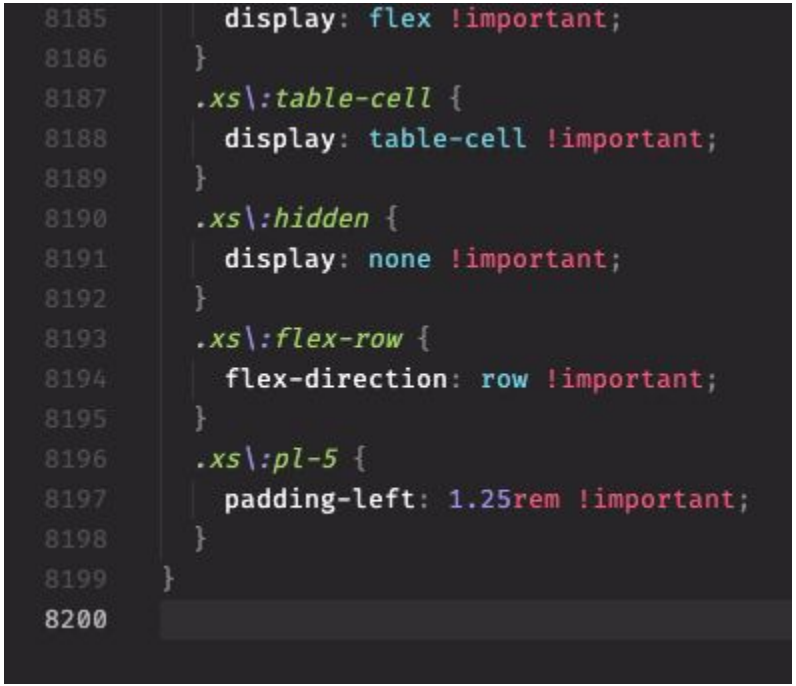


Can you check for e.g. selector order in the resulting CSS file?



Because there is thousands of lines

Current Dashboard CSS file with 8200 line of code:



Deep nesting of selectors and high specificity

HTML:

```
<ul class="seznam">
    <li><a href="prvni.html">Odkaz 1</a></li>
    <li><a href="druhy.html">Odkaz 2</a></li>
    <li><a href="treti.html">Odkaz 3</a></li>
</ul>
```

CSS:

```
.seznam li a {
    font-size: 20px;
    color: blue;
}
```

The link color has not changed

What is in CSS and what blocks me?

```
body #content .page ul li a {
    font-size: 16px;
    color: red;
    font-weight: bold;
}
```

What are the options?

```
body #content .page ul li a {
    font-size: 16px;
    color: red;
    font-weight: bold;
}
```

Selector overload (or refactor code or use !important)

```
body #content .page ul.seznam li a {
    font-size: 16px;
    color: red;
    font-weight: bold;
}
```

Such a selector can be really hard to modified

Nesting Hell

```
.Checkbox--toggle {
  padding: $chekcbox-toggle-diameter / 10 0;
    .Checkbox {
      &-input {
        &:checked {
          & + .Checkbox-label {
            @extend .Checkbox-toggle - active;
          }
        }
      &:not(:checked) {
        &:focus {
          & + .Checkbox-label {
            &::before {
              background-color: $checkbox-toggle-active-handle-bg;
            }
          }
        }
      & + .Checkbox-label {
        background-color: rgba($checkbox-toggle-bg, 0.46);
      }
    &[disabled], &[readonly] {
      & + .Checkbox-label {
        @extend .Checkbox-toggle - disabled;
      }
    }
  }
  &-label {
    @extend .Checkbox-toggle;
  }
}
```

You would say, that is not so bad, but..

Output in style.css:

```
.Checkbox-toggle--active,
.Checkbox--image .Checkbox-input:checked + .Checkbox-image .Checkbox-toggle,
.Checkbox--image
  .Checkbox-input:checked
  + .Checkbox-image
  .Checkbox--toggle
  .Checkbox-label,
.Checkbox--toggle
  .Checkbox--image
  .Checkbox-input:checked
  + .Checkbox-image
  .Checkbox-label,
.Checkbox--toggle .Checkbox-input:checked + .Checkbox-label {
  background-color: #71c5e8;
}
.Checkbox-toggle--active::before,
.Checkbox--image
  .Checkbox-input:checked
  + .Checkbox-image
  .Checkbox-toggle::before,
.Checkbox--image
  .Checkbox-input:checked
  + .Checkbox-image
  .Checkbox--toggle
  .Checkbox-label::before,
.Checkbox--toggle
  .Checkbox--image
  .Checkbox-input:checked
  + .Checkbox-image
  .Checkbox-label::before,
.Checkbox--toggle .Checkbox-input:checked + .Checkbox-label::before {
  left: calc(100% - 20px);
  background-color: #0284ff;
}

.Checkbox-toggle--disabled,
.Checkbox--image .Checkbox-input[disabled] + .Checkbox-image .Checkbox-toggle,
.Checkbox--image
  .Checkbox-input[disabled]
  + .Checkbox-image
  .Checkbox--toggle
  .Checkbox-label,
.Checkbox--toggle
  .Checkbox--image
  .Checkbox-input[disabled]
  + .Checkbox-image
  .Checkbox-label,
.Checkbox--image .Checkbox-input[readonly] + .Checkbox-image .Checkbox-toggle,
.Checkbox--image
  .Checkbox-input[readonly]
  + .Checkbox-image
  .Checkbox--toggle
  .Checkbox-label,
```

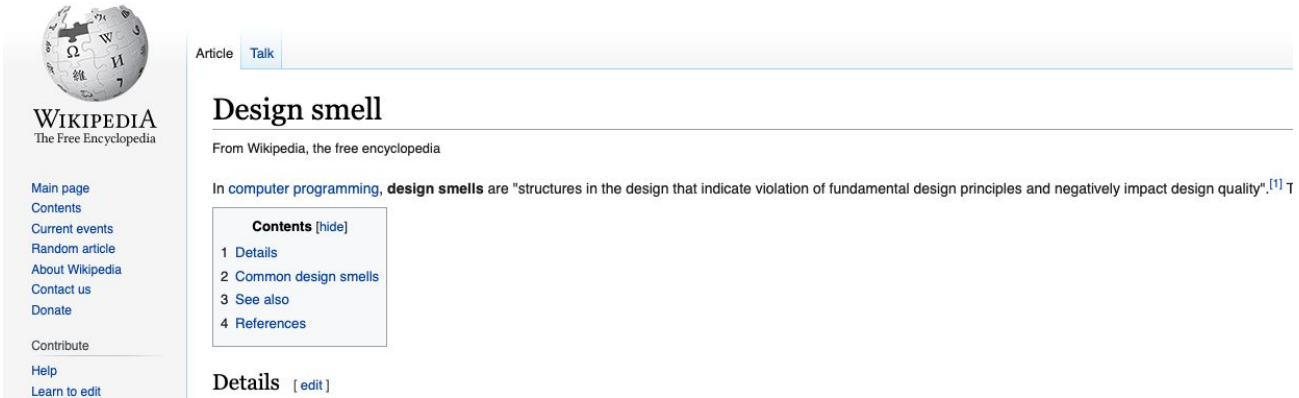
“Nesting selectors” is a good friend but a bad lord

It is alway good approach to stay simple as possible

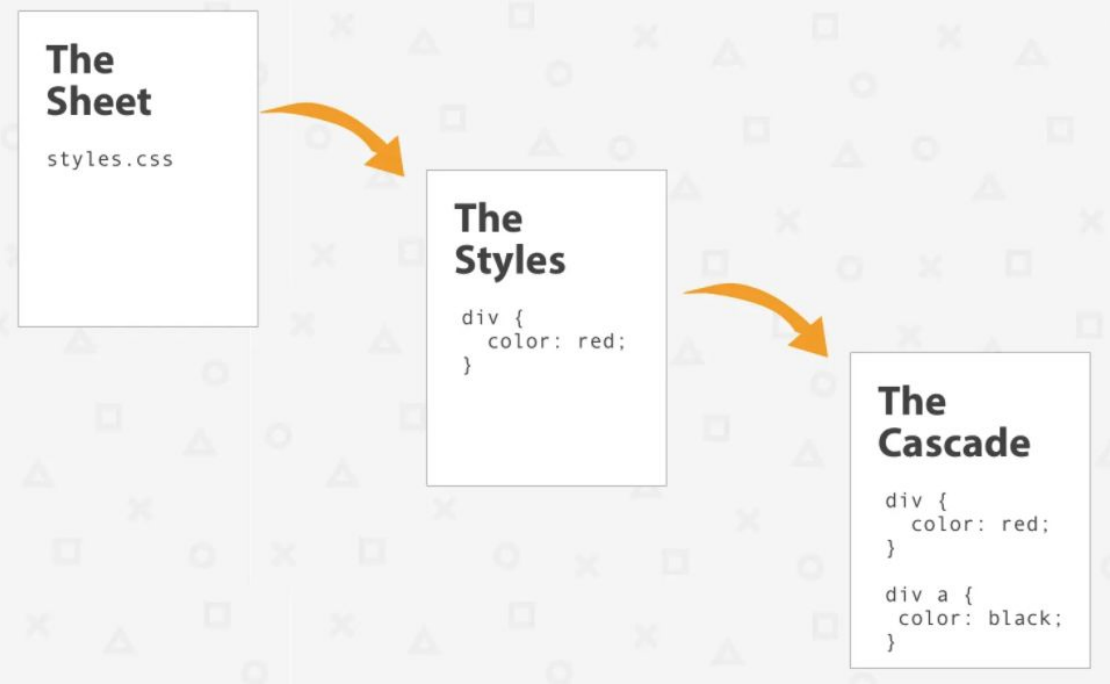
```
.button {
  padding: 10px;

  @include breakpoint(tablet) {
    padding: 8px;
  }

  &:hover {
    background: blue;
  }

  &.is-active {
    color: red;
  }

  &-icon {
    max-width: 16px;
  }

  &-text {
    font-size: 0.875rem;
  }
}
```

Takeaway: Avoid nesting as much as possible

If you need to nest selectors in the third level, something is wrong with design/usage of the component and we call it “**Design smell**”.



Cascade

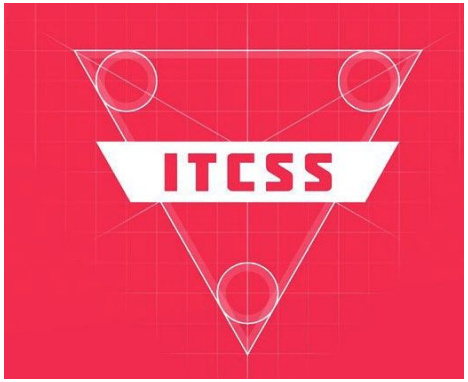

The Sheet
styles.css
The Styles
div {
color: red;
}
The Cascade
div {
color: red;
}
div a {
color: black;
}

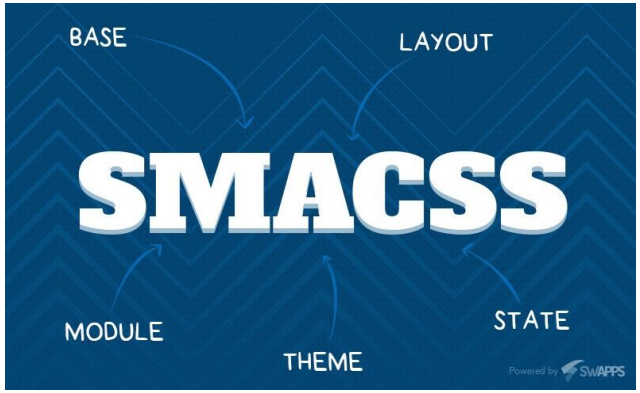
We often end up like this

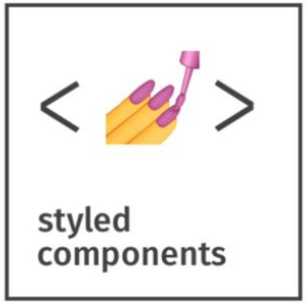
Our current Dashboard

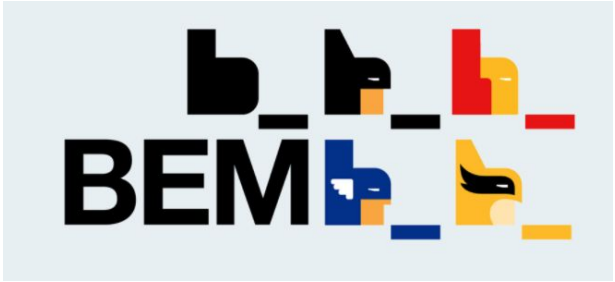
```
body {                                                    styles.0510…d7b7f.css:5
    color: #3d3e40;
    background-size: 220px 220px;
}
body, html {                                              styles.0510…d7b7f.css:5
    font-family: Roboto,sans-serif;
    -webkit-font-smoothing: antialiased;
    font-size: 12px;
}
body {                                                    styles.0510…d7b7f.css:5
    font-family: Helvetica Neue,Helvetica,Arial,sans-serif;
    font-size: 14px;
    line-height: 1.428571429;
    color: #333;
    background-color: #fff;
}
body {                                                    styles.0510…d7b7f.css:1
    font-family: Roboto,sans-serif;
}
body {                                                    <style>
    color: #3d3e40;
    background-size: 220px 220px;
}
body, html {                                              <style>
    font-family: Roboto,sans-serif;
    -webkit-font-smoothing: antialiased;
    font-size: 12px;
}
body {                                                    <style>
    font-family: Helvetica Neue,Helvetica,Arial,sans-serif;
    font-size: 14px;
    line-height: 1.428571429;
    color: #333;
    background-color: #fff;
}
```

What we can do? Use some good approaches



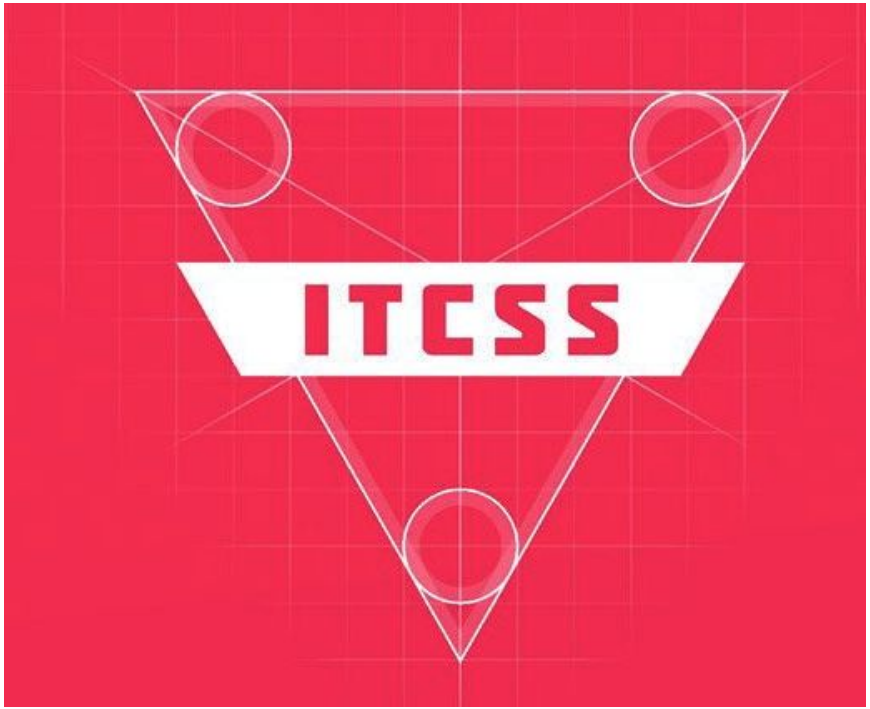




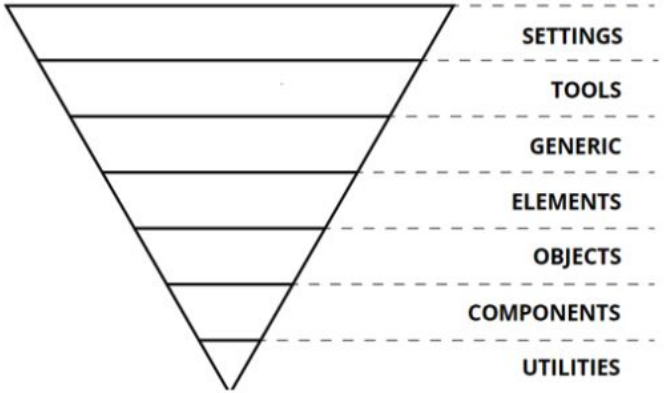


SUIT
CSS

ITCSS



ITCSS


SETTINGS
TOOLS
GENERIC
ELEMENTS
OBJECTS
COMPONENTS
UTILITIES

ITCSS

Settings — Space for preprocessors with variables such as colors, design tokens, typography, grid.
Tools — Layer with mixins, functions, media queries.

Generic — Here we insert styles for third party libraries such as normalize, reset or any others

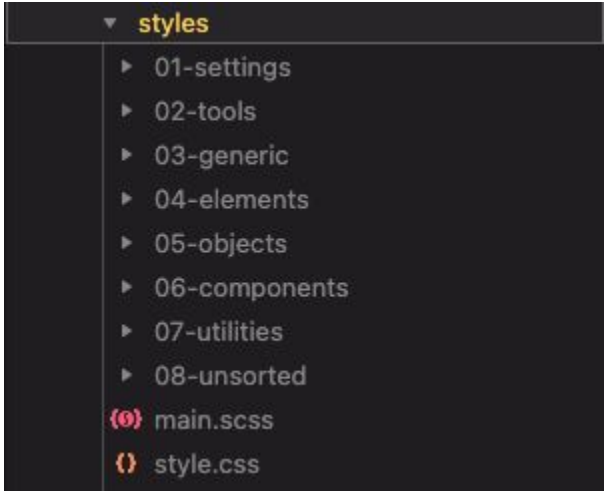
Elements — Selectors for bare HTML elements such as h1, p, article, a

Objects — Class definitions for layout, grid, indentation - reusable non-decorative styles.

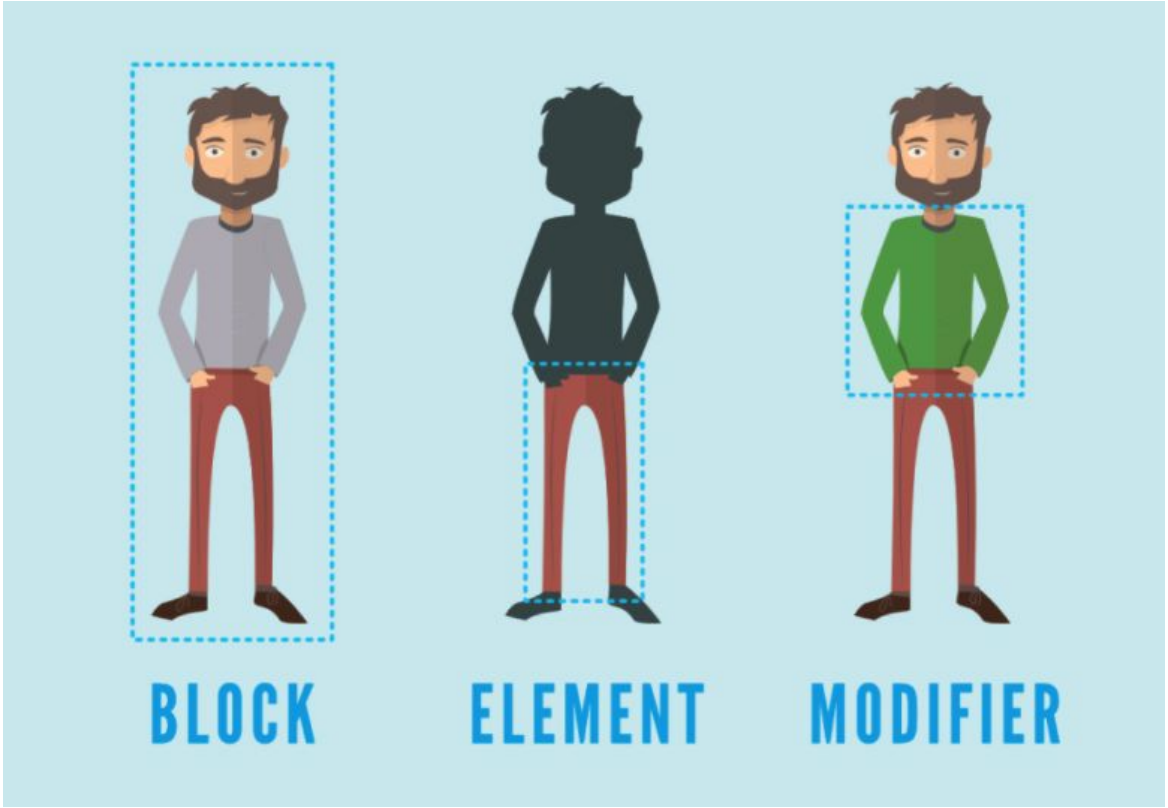
Components — Specific components across the project - accordion, buttons, breadcrumbs, tooltip.

Utilities — Class utilities that are designed to affect one particular CSS property and are in most cases written with the utmost importance. Utilities and helper classes with ability to override anything which goes before in the triangle.

ITCSS structure is great for any project and it is easy to use



BEM



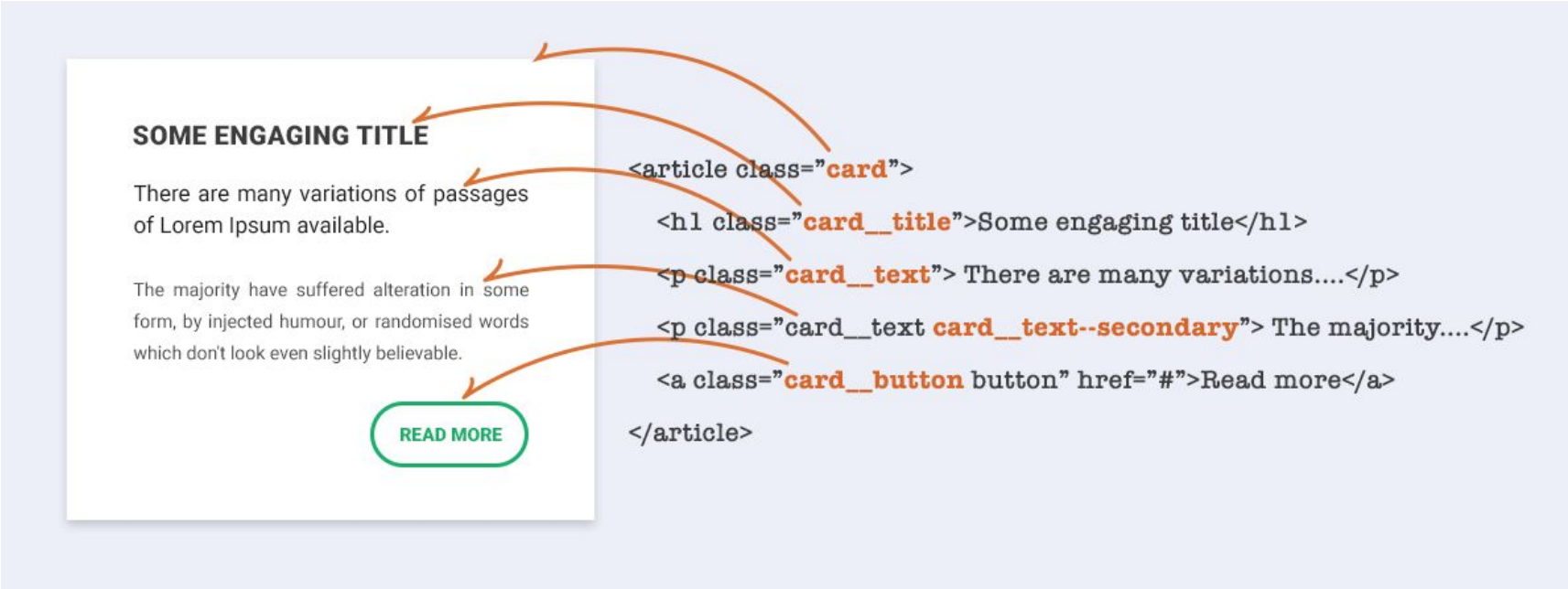
What is important to realize?

“The more experienced developer you are, the more you prefer **code readability** to efficiency”

Why BEM?

- ➔ Find and write CSS rules in a large project is easy.
- ➔ Organize rules for media queries and reusable libraries.
- ➔ **Reduce the complexity and nesting of your CSS selectors**.
- ➔ Have a consistent approach to positioning elements on the page.
- ➔ Have a consistent approach to changing the look of HTML.
- ➔ **Have a consistent approach to composing larger components from smaller components**.
- ➔ A unified approach that is easy to explain to newcomers
- ➔ It keeps the world of CSS safe from mess and clutter.

BEM is G. R. E. A. T


SOME ENGAGING TITLE
There are many variations of passages of Lorem Ipsum available.
The majority have suffered alteration in some form, by injected humour, or randomised words which don't look even slightly believable.
READ MORE
<article class="card">
<h1 class="card__title">Some engaging title</h1>
 There are many variations....
 The majority....
Read more
</article>

G. R. E. A. T

G for Global

BEM is one of the most recognized naming conventions out there. So if you are introducing a new team member to your BEM project, there’s a good chance they already know the convention, which reduces initial friction and allows them to be productive since day 1.

G. **R**. E. A. T

R for Readable

Thanks to descriptive class names given to basically every element, the stylesheet is easy to read on its own. Not only selectors look better, they also work faster than deeply nested ones.

G. R. **E**. A. T

E for Expandable

As the specificity of CSS selectors is minimal, adding another variation is very simple. Single modifier class should be enough — no more ‘at least equal selector weight’ toil.

```
.menu{
  ...
  &__list { ... }
  &__link { ... }
  &__icon { ... }
}
.menu--secondary {
  .menu__icon{...}
}

...

.menu--secondary .menu__icon{ ... }
```

```
nav {
  ...
  ul {
    li {
      i { ... }
      ul {
        i { ... }
      }
    }
  }
}

...

nav ul li ul i { ... }
```

G. R. E. **A**. T

A for Adaptable

Sharing the philosophy of modularity, BEM naturally works fine with frameworks. Also, styling is independent of elements type and nesting, making it less prone to break when tackling with document structure.

HTML:
```
<article class="card">
  <a class="card__button">click me</a>
</article>
```

CSS:
```
.card__button { ... }
```

HTML changed:
```
<article class="card">
  <div>
    <button class="card__button">click me</button>
  </div>
</article>
```

STILL WORKS

HTML:
```
<article>
  <a>click me</a>
</article>
```

CSS:
```
article a { .... }
```

HTML changed:
```
<article>
  <div>
    <button>click me</button>
  </div>
</article>
```

NO LONGER WORKS

G. R. E. A. **T**

T for Tough

There are only two hard things in Computer Science: cache invalidation and naming things.

When you start following BEM (fully and honestly), you'll probably find yourself struggling with it constantly. Paradoxically, it's a good thing:

- ➔ finding proper block names makes the code clean and legible to others (your future self included)
- ➔ reusing existing blocks
- ➔ avoiding multi-level nesting makes you rethink document structure

Tailwind, and why I would consider not using it

Tailwind is good for

- Quickly prototyping
- Safety - there is nothing in design that is not in config
- Small projects like personal sites, blog sites etc.

Tailwind is (IMHO) not good for large scaled projects

- Styling and HTML are Mixed
- It Takes Time to Learn (not necessarily a disadvantage)
- Lack of Important Components (not so much components)
- Components aren’t provided by default
- Unreadable class names
- It's Inconsistent (items-*: align or justify? content-*: align or justify?)
- It's Difficult to Read
- Really hard to do code reviews
- You Can't Chain Selectors
- Tailwind Locks You Into the Utility CSS Paradigm
- Tailwind Is an Unnecessary Abstraction
- Tailwind, Dev Tools, and Developer Experience (Imposible create variants)

It's Inconsistent

items-*: align or justify?

content-*: align or justify?

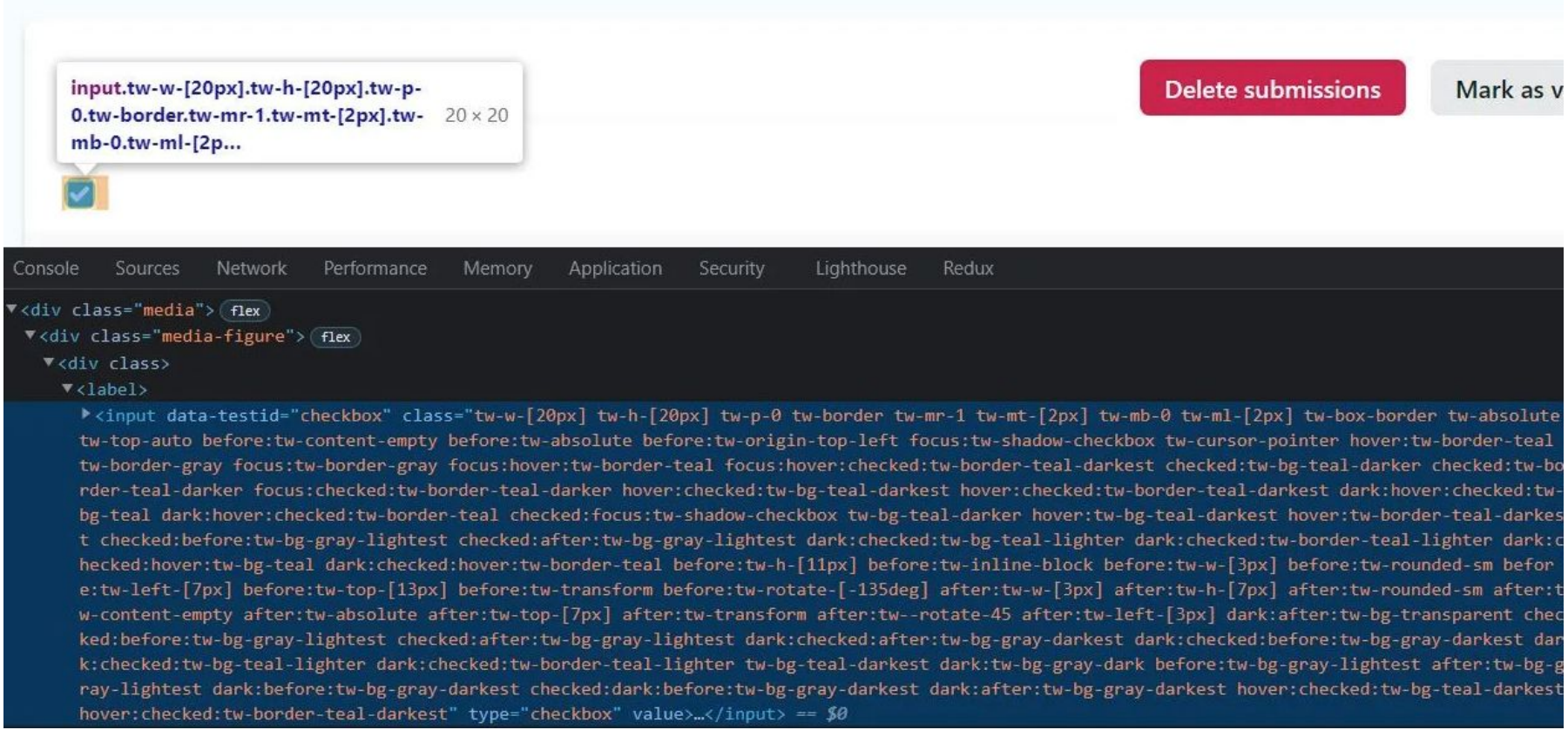
justify-*: content or items?

align-*: content or items?

It's hard to read

 Yikes.

It's hard to read



```css
.thing {
  width: 16px;
  height: 16px;
  color: white;
  background-color: black;
  padding: 0.25rem 0.5rem;
  margin: 0.25rem;
  border-radius: 0.25rem;
  font-size: 0.875rem;
  line-height: 1.25rem;
}

@media screen and (min-width: 768px) {
  .thing {
    width: 32px;
    height: 32px;
    border-radius: 0.375rem;
    font-size: 1rem;
    line-height: 1.5rem;
  }
}

@media screen and (min-width: 1024px) {
  .thing {
    width: 48px;
    height: 48px;
    border-radius: 0.5rem;
    font-size: 1.125rem;
    line-height: 1.75rem;
  }
}
```

More pleasure for eyes

You Can't Chain Selectors

```
.nav-link:focus,
.nav-link:hover,
.nav-link[aria-current="page"] {
  /* CSS goes here */
}
```

Or better yet, this:

```
.nav-link:is(:focus, :hover, [aria-current="page"]) {
  /* CSS goes here */
}
```

It's Harder to Tweak CSS in Dev Tools

- ➔ It is hard to simulate styling in DevTools
- ➔ It's Harder to Find Components in Dev Tools
- ➔ Recompiling HTML Is Slower Than Recompiling CSS

Tailwind Is Still Missing Some Key Features of CSS

What will be in CSS specification soon? What is in working drafts?

- ➔ Container Queries
- ➔ :has() selector
- ➔ @when/@else rules
- ➔ Cascade Layers
- ➔ Subgrid
- ➔ Nesting