SMACSS

Hi, there!
I’m Luciano Battagliero
@battaglr

Scalable and Modular Architecture for CSS

MEANING

Jonathan Snook

WHO

~2011

WHEN

A little bit of context

BEM ~2009

OOCSS ~2010

SUIT CSS ~2012

What it’s not

A framework, a boilerplate
or a library

There’s no **code**

What is it, then?

A series of guidelines, recommendations and advices

It’s a **methodology**

What’s the goal?

Author **organized** and
structured code

Easier to **build**, **maintain**
and **scale**

Okay, got it! Let's dive into that "smacks" thing

Categoriza-tion

Every **code-base** needs
some **organization**

One of the **core** aspects
is **categorization**

There are **five categories**

Base, **layout**, **module**,
state and **theme**

Base

The **low-level defaults**, or the **global scope** if you prefer it

```
body {
  margin: 0;
}
```

```
b,
strong {
  font-weight: bold;
}
```

```
*,
*::before,
*::after {
  box-sizing: border-box;
}
```

Using things like `reset.css` or `normalize.css`
It’s okay to use them as long as you know what they do, and don’t remain as a black-box kind of stuff

Layout

The **main** building **blocks**
of a page

Generally defining **structure**-related styles

Based on reuse, we can divide them into **major** and **minor**

Major layouts

```
#header { … }
```

```
#sidebar { … }
```

```
#footer { … }
```

To `id`, or not to `id`

There's no real advantage on using an `id` over a `class`, and you will be avoiding a potential specificity-related headache

Minor layouts

```
#header { … }

.l-fixed #header { … }
```

```
.l-grid { … }
```

Namespacing

With the exception of `id` selectors, the *recommendation* is to namespace layouts
using an `l-` prefix

Nesting

Layouts will mainly contain modules, although they can contain other layouts too

Module

The **reusable blocks**
of a page

Should be built to work as **stand-alone** blocks

Providing **flexibility** and context **independence**

Building modules

Avoid element selectors

```
<div class="folder">
  <span>42 KB</span>
</div>
```

```
.folder span { … }
```

```
/* This may be a better idea */
.folder > span { … }
```

```
<div class="folder">
  <span>42 KB</span>
  <span>01/01/2021</span>
</div>
```

```
.folder > span { … }

.folder > span + span { … }
```

```
<div class="folder">
  <span>42 KB</span>
  <span>01/01/2021</span>
  <span>John Doe</span>
</div>
```

```
.folder > span { … }

.folder > span + span { … }

.folder > span:last-child { … }
```

```
<div class="folder">
  <span class="folder-size">42 KB</span>
  <span class="folder-date">01/01/2021</span>
  <span class="folder-owner">John Doe</span>
</div>
```

```
.folder-size { … }

.folder-date { … }

.folder-owner { … }
```

Kind of a bold statement

You can use element selectors, but only if they can and will be predictable; also avoid going more than one level deeper

Namespacing

Descendant elements of a module use its name as a prefix

Subclass modules

Modifies certain **styles**
of a module

```
.nav { … }

.nav > li { … }
```

```
.nav { … }

.nav > li { … }

#sidebar .nav { … }

#sidebar .nav > li { … }
```

```
.nav { … }

.nav > li { … }

.nav-stacked { … }

.nav-stacked > li { … }
```

```
<ul class="nav">...</ul>

<ul class="nav nav-stacked">...</ul>
```

Namespacing

Subclass modules use as a prefix the name of the base module

Nesting

Modules will be inside layouts, although they can be inside other modules, too

State

Describes modules or layouts
in a **particular state**

Subclass modules are applied at **render-time** and will not change on run-time

States may be applied at render or run-time and are likely to be modified at **run-time**

States are most likely triggered by **user's actions**

```
<a class="button is-hidden">...</a>
```

```
.is-hidden {
  display: none !important;
}
```

```
<a class="tab is-tab-active">...</a>
```

```
.tab { … }

.is-tab-active { … }
```

Namespacing
Global states are prefixed with `is-`, while scoped states are prefixed with `is-` followed by the module or layout name

Theme

The different **variations** of
look and feel

Can **affect** rules of **any category**

```
/* On buttons.css */

.button {
  border: 1px solid;
}
```

```
/* On theme-navy.css */

.button {
  border-color: #001f3f;
}
```

Depth of applicability

The number of **generations** that are **affected** by a given **rule**

```
section#contact ul.details li { … }
```

```
#contact .details li { … }
```

```
/* Depth of 3 */
section#contact ul.details li { … }

/* Depth of 3 */
#contact .details li { … }
```

Heads up!
Don’t confuse depth of applicability with specificity

Keeping the **depth** as **low** as possible

Increases the **flexibility** and
makes the code **easier** to **reuse**

Avoids relying heavily on a **specific** HTML **structure**

Lowers the **specificity**

```
#footer form {
  padding: .625em;
}

#footer form > input {
  display: inline-block;
}
```

```
#footer form,
#sidebar form {
  padding: .625em;
}

#footer form > input,
#sidebar form > input {
  display: inline-block;
}
```

```
.search {
  padding: .625em;
}

.search > input {
  display: inline-block;
}
```

```
.search {
  padding: .625em;
}

.search-input {
  display: inline-block;
}
```

```
.panel > div,
.panel > ul,
.panel > ol {
  padding: .75em;
}
```

```
.panel-body {
  padding: .75em;
}
```

That’s not just it!

There's a lot **more** about **SMACSS**. Go to smacss.com and find out!

Questions?

Thanks!