Globant

we are ready

Front End Engineer

CSS, Selectors & SMACSS

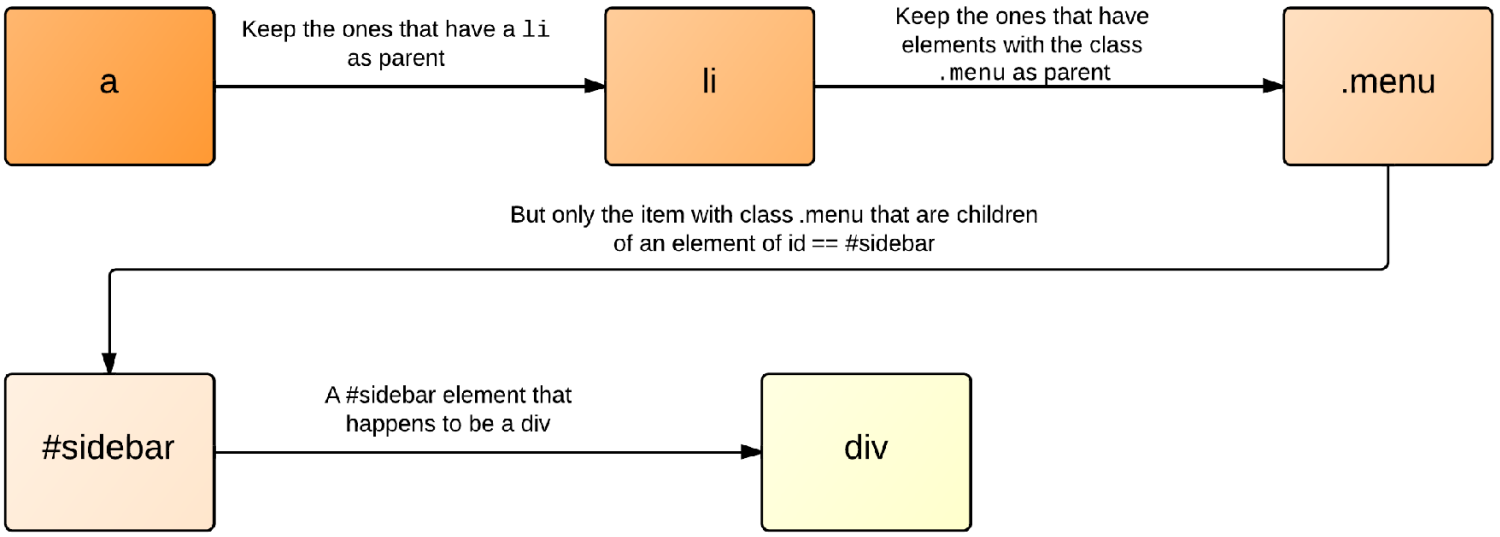
Topics

Selectors
Performance
Box Model
Cross-Browser
Advanced selectors.
Good practices.

HOW RULE ENGINES WORK

```
div#sidebar .menu li a
```

Get all a elements



SELECTORS

Attribute Selectors

```
1  input[type="text"] /* Matches all input type == "text" */
2
3  div[id~="some"] /* Matches "some guy" */
4
5  div[id|="en"] /* Matches "en" or "en-US" */
6
7  div[title="container"][id="content"] /* Matches a div
8                                           with title == container
9                                           & id == content */
10
11 a[href$=".html"] /* Matches hrefs ending in ".html" */
12
13 p[title*="hello"] /* Matches titles containing "hello" */
14
15 p[title^="hello"] /* Matches titles starting with "hello" */
16
```

Selectors also can be defined based on attributes and attribute values associated with elements

Two attributes, id and class, are often key in targeting styles to a specific element or group of elements.

Attribute Selectors

Selector	Description	Example	Matches
`#id`	The element with the id value, *id*	#intro	The element with the id *intro*
`.class`	All elements with the class value, *class*	.main	All elements belonging to the *main* class
`elem.class`	All *elem* elements with the class value *class*	p.main	All paragraphs belonging to the *main* class
`elem[att]`	All *elem* elements containing the *att* attribute	a[href]	All hypertext elements containing the href attribute
`elem[att="text"]`	All *elem* elements whose *att* attribute equals *text*	a[href="gloss.htm"]	All hypertext elements whose href attribute equals *gloss.htm*
`elem[att~="text"]`	All *elem* elements whose *att* attribute contains the word *text*	a[rel~="glossary"]	All hypertext elements whose rel attribute contains the word *glossary*
`elem[att|="text"]`	All *elem* elements whose *att* attribute value is a hyphen-separated list of words beginning with *text*	p[id|="first"]	All paragraphs whose id attribute starts with the word *first* in a hyphen-separated list of words
`elem[att^="text"]`	All *elem* elements whose *att* attribute begins with *text* (CSS3)	a[rel^="prev"]	All hypertext elements whose rel attribute begins with *prev*
`elem[att$="text"]`	All *elem* elements whose *att* attribute ends with *text* (CSS3)	a[href$="org"]	All hypertext elements whose href attribute ends with *org*
`elem[att*="text"]`	All *elem* elements whose *att* attribute contains the value *text* (CSS3)	a[href*="faq"]	All hypertext elements whose href attribute contains the text string *faq*

Now, you can see:

01 CSS Attribute and Value Selectors

http://youtu.be/ni-58OKlJAs

PSEUDO CLASSES SELECTORS

CSS pseudo-classes are used to add special effects to some selectors.

All CSS Pseudo Classes/Elements

Selector	Example	Example description
:link	a:link	Selects all unvisited links
:visited	a:visited	Selects all visited links
:active	a:active	Selects the active link
:hover	a:hover	Selects links on mouse over
:focus	input:focus	Selects the input element which has focus
:first-letter	p:first-letter	Selects the first letter of every <p> element
:first-line	p:first-line	Selects the first line of every <p> element
:first-child	p:first-child	Selects every <p> elements that is the first child of its parent
:before	p:before	Insert content before every <p> element
:after	p:after	Insert content after every <p> element
:lang(*language*)	p:lang(it)	Selects every <p> element with a lang attribute value starting

Now, you can see:

02 CSS Pseudo Class Selectors

http://youtu.be/dyHTz93M8mM

PSEUDO CLASSES SELECTORS

```
1  a:link    { color: red; }    /* unvisited links */
2
3  a:visited { color: blue; }   /* visited links   */
4
5  a:hover   { color: yellow; } /* user hovers     */
6
7  a:active  { color: lime; }   /* active links    */
8
9  html:lang(fr-ca) { quotes: '« ' ' »'; }
10
11 p:before {content: "Hello! "; }
12
13 p:after {content: "Goodbye! "; }
```

Now, you can see:

03 Before and After Pseudo Elements

http://youtu.be/JQiQ40ww5Ko

PSEUDO CLASSES SELECTORS

```
1  div > p:first-child { text-indent: 0; }
2
3  div > p::last-child { text-indent: 0; }
4
5  div > p:nth-child(odd) { color: red;}
6
7  div > p:nth-last-child(-n+2) { color: red;}
8
9  h2:nth-of-type(n+2) { color: blue; }
10
11 h2:nth-last-of-type(n+2) {color: red; }
12
13 p:first-of-type { color: red;}
14
15 p:last-of-type { color: red;}
16
17 p:first-letter { font-size: 3em; font-weight: normal; }
18
19 p:first-line { text-transform: uppercase; }
20
21 p:only-child { color: red; }
22
23 p:only-of-type { color: blue; }
24
25 p:empty { color: yellow; }
```

Now, you can see:

04 Child Selectors

http://youtu.be/4QcUPJM3NSU

HIGH PERFORMANCE CSS

AVOID UNNECESSARY TAG IDENTIFIERS

```
1 ul#navigation,
2 ul.menu {
3   margin-left: 0;
4 }
```

Bad

```
1 #navigation,
2 .menu {
3   margin-left: 0;
4 }
```

Good

AVOID CHAINING CLASSES

```
1 .small.private.blue.icon {
2   width: 30px;
3 }
4
```

Bad

Use subclassing instead

```
1 .small-private-icon {
2   width: 30px;
3 }
4
```

Good

AVOID HIGH NESTING DEPTH

```
1 html div table tr td {
2   font-weight: bold;
3 }
4
```

```
1 html div table tr td {
2   font-weight: bold;
3 }
4
```

WHAT?!?!

AVOID THE UNIVERSAL SELECTOR

```
1 * {
2     display: block;
3 }
```

CROSS BROWSER TECHNIQUES

Target IE with conditional comments

```
1 <!--[if IE]>
2     <link href="ie.css" rel="stylesheet" type="text/css" />
3 <![endif]-->
4
```

Target a special version of IE with conditional Comments

```
1 <!--[if IE6]>
2     <link href="ie.css" rel="stylesheet" type="text/css" />
3 <![endif]-->
4
```

CROSS BROWSER TECHNIQUES

Targeting IE with CSS bugs

```
.class {
  width: 200px; /* All browsers */
  *width: 250px; /* IE7 & Below */
  _width: 300px; /* IE6 */
  width: 200px\9; /* IE8 & Below */
  width /*\**/: 200px\9; /* IE8 Standards Mode */
}

```

Now, you can see:

05 True Grit Debugging CSS

http://youtu.be/aUhubnGvYK8

SMACSS

STRUCTURE

Base — These are your defaults (html, body, h1, ul, etc).

Layout — These divide the page into major sections.

Module — These are the reusable modular components of a design.

State — These describe how things look when in a particular state (hidden or expanded, active/inactive).

Theme — These define things like a color scheme or typographic treatment across a site.

BASE STYLES

- Reset.css
- Default typography
- Default colours
- Usually just single element selectors

BASE STYLES

A Base rule is applied to an element using an element selector, a descendent selector, or a child selector, along with any pseudo-classes.

It doesn't include any class or ID selectors.

It is defining the default styling for how that element should look in all occurrences on the page.

Base styles include setting heading sizes, default link styles, default font styles, and body backgrounds. There should be no need to use in a Base style.

BASE STYLES

```
/* body */
body {
  margin: 0;
  font-size: 14px;
}

/* headings */
h1 {
  text-transform: uppercase;
}

h1, h2, h3 {
  font-weight: bold;
  color: orange;
}

/* link styles */
a {
  color: teal;
}

a:hover {
  color: navy;
}

```

LAYOUT

Layout styles can also be divided into major and minor styles based on reuse.

Major layout styles such as header and footer are traditionally styled using ID selectors but take the time to think about the elements that are common across all components of the page and use class selectors where appropriate.

LAYOUT

Layout declarations

```
#header, #article, #footer {
    width: 960px;
    margin: auto;
}

#article {
    border: solid #CCC;
    border-width: 1px 0 0;
}
```

LAYOUT

Generally, a Layout style only has a single selector: a single ID or class name. However, there are times when a Layout needs to respond to different factors.

For example, you may have different layouts based on user preference. This layout preference would still be declared as a Layout style and used in combination with other Layout styles.

LAYOUT

Use of a higher level Layout style affecting other Layout styles.

```
#article {
    float: left;
}

#sidebar {
    float: right;
}

.l-flipped #article {
    float: right;
}

.l-flipped #sidebar {
    float: left;
}
```

LAYOUT

Theory is one thing but application is another.

Let’s take a look at an actual web site and consider what is part of the layout and what is a module.

LAYOUT

In this example, the .l-fixed class modifies the design to change the layout from fluid (using percentages) to fixed (using pixels).

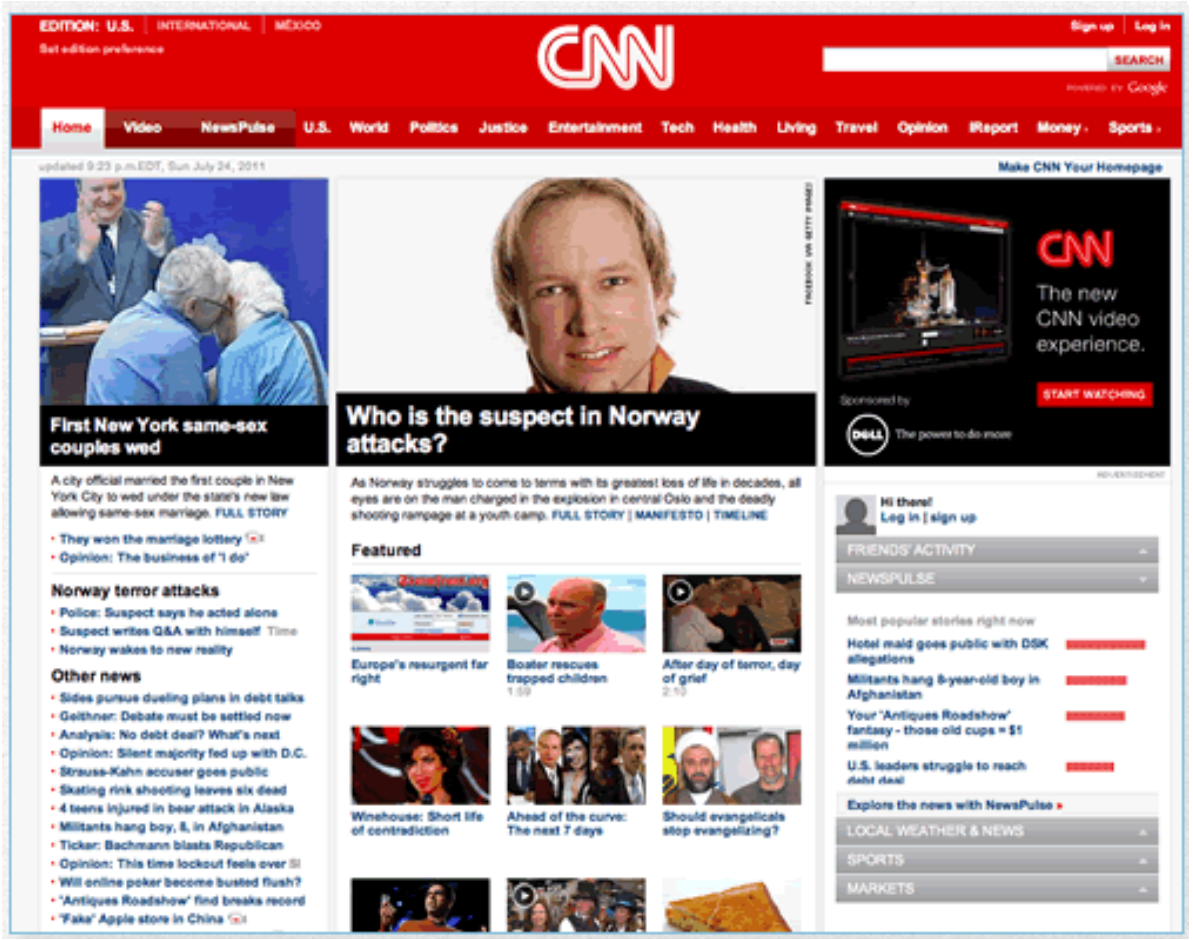
Using two Layout styles together to switch from fluid to fixed layout.

```
#article {
    width: 80%;
    float: left;
}

#sidebar {
    width: 20%;
    float: right;
}

.l-fixed #article {
    width: 600px;
}

.l-fixed #sidebar {
    width: 200px;
}
```

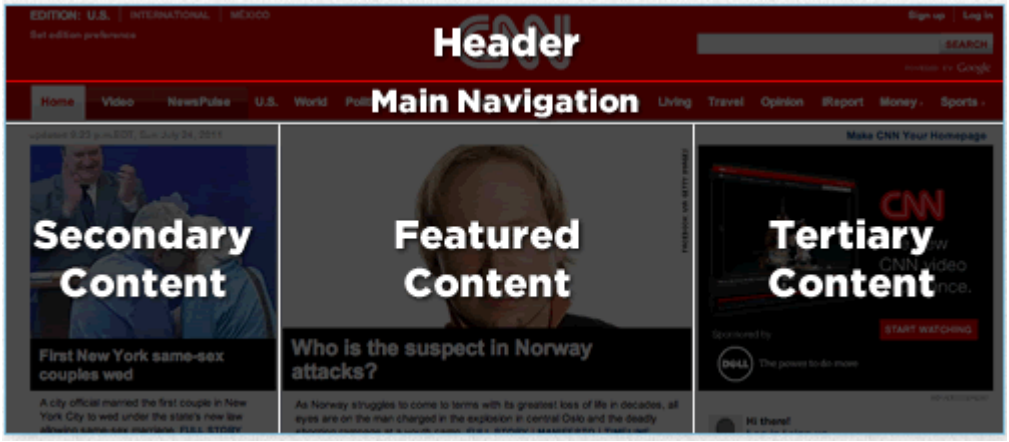
LAYOUT

In taking a look at the CNN web site, there are a number of patterns that occur in plenty of web sites.

For example, there is a header, a navigation bar and a content area.



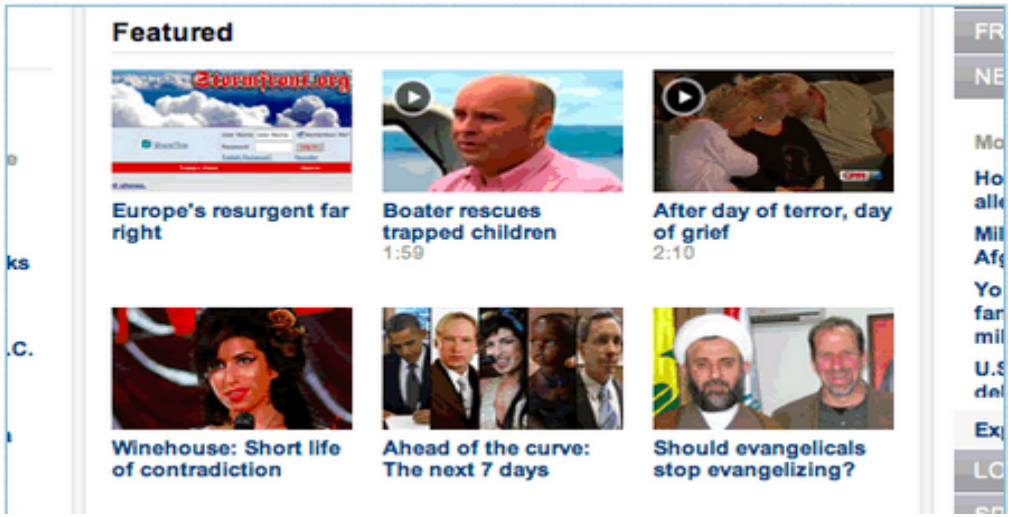
LAYOUT



Our CSS structure might look something like this:

```
#header { … }
#primarynav { … }
#maincontent { … }

<div id="header"></div>
<div id="primarynav"></div>
<div id="maincontent"></div>
```

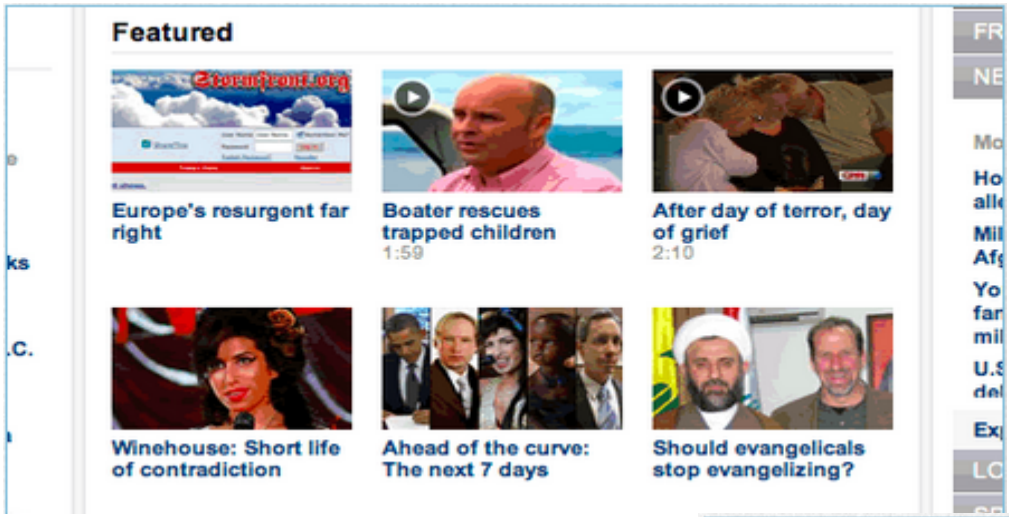
LAYOUT



Example HTML code for the Featured section layout

```
<div>
<h2>Featured</h2>
<ul>
    <li><a href="…">…</a></li>
    <li><a href="…">…</a></li>
    …
</ul>
</div>
```

LAYOUT



A possible approach to styling the list of featured items

```
div#featured ul {
    margin: 0;
    padding: 0;
    list-style-type: none;
}

div#featured li {
    float: left;
    height: 100px;
    margin-left: 10px;
}
```

LAYOUT

Familiar stuff and probably similar to html and css you've written before. There are 3 assumptions with the above code.

- there will only ever be one featured section on page
- list items in the section are always floated to the left
- list items in the section always have a height of 100px

These assumptions might be reasonable for small sites, but become less reasonable as sites grow larger and more and different people work on them.

LAYOUT

SMACSS guidelines would instead suggest the following css after adding the l-grid class to the container div.

```
.l-grid > li {
    display: inline-block;
    margin: 0 0 10px 10px;

    /* IE7 hack to mimic inline-block on block elements */
    *display: inline;
    *zoom: 1;
}
```

LAYOUT

- The grid layout can now be applied to any container to create a float-style layout
- The depth of applicability has been decreased by 1 (more on this below)
- The specificity of the selectors has been reduced
- The height requirement has been removed allowing each row to grow to the height of its tallest item.

MODULE

A Module is a more discrete component of the page. It is your navigation bars and your carousels and your dialogs and your widgets and so on.

This is the meat of the page. Modules sit inside Layout components.

Modules can sometimes sit within other Modules, too.

Each Module should be designed to exist as a standalone component. In doing so, the page will be more flexible. If done right, Modules can easily be moved to different parts of the layout without breaking.

MODULE

When defining the rule set for a module, avoid using IDs and element selectors, sticking only to class names.
A module will likely contain a number of elements and there is likely to be a desire to use descendent or child selectors to target those elements.

Module example

```
.module > h2 {
    padding: 5px;
}

.module span {
    padding: 5px;
}
```

MODULE

Use child or descendant selectors with element selectors if the element selectors will and can be predictable. Using .module span is great if a span will predictably be used and styled the same way every time while within that module.

Styling with generic element

```
<div class="fld">
    <span>Folder Name</span>
</div>

/* The Folder Module */
.fld > span {
    padding-left: 20px;
    background: url(icon.png);
}
```

MODULE

The problem is that as a project grows in complexity, the more likely that you will need to expand a component's functionality and the more limited you will be in having used such a generic element within your rule.

Styling with generic element

```
<div class="fld">
    <span>Folder Name</span>
    <span>(32 items)</span>
</div>
```

MODULE

Only include a selector that includes semantics.
A span or div holds none. A heading has some. A class defined on an element has plenty.

Styling with generic element

```
<div class="fld">
    <span class="fld-name">Folder Name</span>
    <span class="fld-items">(32 items)</span>
</div>
```

SUBCLASSING MODULES

When we have the same module in different sections, the first instinct is to use a parent element to style that module differently.

The problem with this approach is that you can run into specificity issues that require adding even more selectors to battle against it or to quickly fall back to using

SUBCLASSING MODULES

We have an input with two different widths.
Throughout the site, the input has a label beside it and therefore the field should only be half the width.
In the sidebar, however, the field would be too small so we increase it to 100% and have the label on top.
All looks well and good.

Subclassing

```
.pod {
    width: 100%;
}
.pod input[type=text] {
    width: 50%;
}
#sidebar .pod input[type=text] {
    width: 100%;
}
```

SUBCLASSING MODULES

Now, we need to add a new component to our page. It uses most of the same styling as a .pod and so we re-use that class.

However, this pod is special and has a constrained width no matter where it is on the site. It is a little different, though, and needs a width of 180px.

Battling against specificity

```
.pod {
    width: 100%;
}
.pod input[type=text] {
    width: 50%;
}
#sidebar .pod input[type=text] {
    width: 100%;
}

.pod-callout {
    width: 200px;
}
#sidebar .pod-callout input[type=text],
.pod-callout input[type=text] {
    width: 180px;
}
```

SUBCLASSING MODULES

We are doubling up on our selectors to be able to override the specificity of #sidebar.

What we should do instead is recognize that the constrained layout in the sidebar is a subclass of the pod and style it accordingly.

Battling against specificity

```
.pod {
    width: 100%;
}
.pod input[type=text] {
    width: 50%;
}
.pod-constrained input[type=text] {
    width: 100%;
}

.pod-callout {
    width: 200px;
}
.pod-callout input[type=text] {
    width: 180px;
}
```

SUBCLASSING MODULES

With sub-classing the module, both the base module and the sub-module class names get applied to the HTML element.

Sub-module class name in HTML

```
<div class="pod pod-constrained">...</div>
<div class="pod pod-callout">...</div>
```

To help battle against specificity (and if IE6 isn't a concern), then you can double up on your class names.

Subclassing

```
.pod.pod-callout { }

<!-- In the HTML -->
<div class="pod pod-callout"> ... </div>
```

STATES

A state is something that augments and overrides all other styles.

For example, an accordion section may be in a collapsed or expanded state. A message may be in a success or error state.
States are generally applied to the same element as a layout rule or applied to the same element as a base module class.

STATES

State applied to an element

```
<div id="header" class="is-collapsed">
    <form>
        <div class="msg is-error">
            There is an error!
        </div>
        <label for="searchbox" class="is-hidden">Search</labe
        <input type="search" id="searchbox">
    </form>
</div>
```

STATES

The header element just has an ID. As such we can assume that any styles, if there are any, on this element are for layout purposes and that the is-collapsed represents a collapsed state. One might presume that without this state rule, the default is an expanded state.

The msg module is simple enough and has an error state applied to it. One could imagine a success state could be applied to the message, alternatively.

Finally, the field label has a hidden state applied to hide it from sight but still keep it for screen readers. In this case, we are actually applying the state to a base element and not overriding a layout or module.

STATES

There is plenty of similarity between a sub-module style and a state style. They both modify the existing look of an element. However, they differ in two key ways:

1. State styles can apply to layout and/or module styles; 2. State styles indicate a JavaScript dependency.

For example, clicking on a tab will activate that tab. Therefore, an is-active or is-tab-active class is appropriate. Clicking on a dialog close button will hide the dialog. Therefore, an is-hidden class is appropriate.

STATES

In a case where a state rule is made for a specific module, the state class name should include the module name in it.

The state rule should also reside with the module rules and not with the rest of the global state rules.

The styles for a particular module won't need to be loaded until that particular module is loaded.

State rules for modules

```
.tab {
    background-color: purple;
    color: white;
}

.is-tab-active {
    background-color: white;
    color: black;
}
```

THEMES

A theme defines colors and images that give your application or site its look and feel.
Separating the theme out into its own set of styles allows for those styles to be easily redefined for alternate themes.
Themes can affect any of the primary types. It could override base styles like default link colors.
It could change module elements such as chrome colors and borders.
It could affect layout with different arrangements. It could also alter how states look.

THEMES

Let’s say you have a dialog module that needs to have a border color of blue, the border itself would be initially defined in the module and then the theme defines the color:

Module Theming

```
// in module-name.css
.mod {
    border: 1px solid;
}

// in theme.css
.mod {
    border-color: blue;
}
```

THEMES

Last but not least, there are font rules.
Similar to themes, there are times when you need to redefine the fonts that are being used on a wholesale basis, such as with internationalization.
Locales such as China and Korea have complex ideograms that are difficult to read at smaller font sizes.
As a result, we create separate font files for each locale that redefine the font size for those components.

Now, you can see:

06 Scalable and modular Architecture for CSS

http://youtu.be/QmSL-U8oifY

Thank you!

Globant
we are ready