CSS:
Venturing Beyond ‘It Works’

Writing Sane and Flexible CSS that Makes Sense to Future You

State of the CSS Address

Current State

- 3 Style Guides/Themes
    1. **myUSCIS Style Guide**
    2. **US Style Guide**
    3. **SVS Style Guide**
- Selectors with same purpose (i.e., .float-right and .to-right)
- Selectors that are too specific (i.e., .p-no-margin-top)

Current State

- Confusing file structure, resulting in uncertainty over what's already been written
- Things work, but it's at the expense of repeated code
- Lots of unused code pulling from myUSCIS

We Need...

1. A **sane environment** that is accessible to lots of people
2. To **tame and manage** source order
3. To create a **place for everything** to live (new and old)
4. To **reduce** waste and redundancy **(code bloat)**

Our Focus

1. Who's Fault: CSS or Us?
2. Keeping Things Tidy with the 7-1 Pattern
3. Class Naming: Specificity
4. Nesting: Keeping Things Readable
5. Syntax, Commenting, and the Death of !important

Problems with CSS:

CSS' Fault vs. Our Fault

CSS' Fault

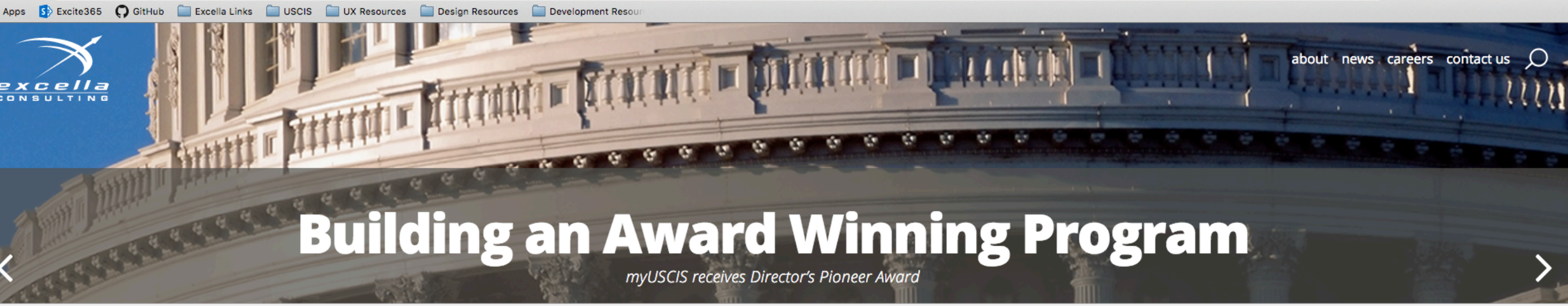
- The cascade and **inheritance**
- Highly dependent on **source order**
- Lots of **gotchas**
- **Specificity**

Our Fault

- Lack of **documentation**
- Mixture of **abilities**
- Different styles, preferences, **ways of working**
- Not looking to see/being aware of **what already exists**
- Adding new styles to the **end of stylesheets**

Inheritance and source order

Each piece of CSS needs a knowledge of what came before it and what might come after it – a.k.a **dependencies**



Elements Console Sources Network Timeline Profiles Resources Security Audits

```
<div class="mobile-menu">…</div>
<div class="site-container">
  ::before
<header class="site-header" itemscope itemtype="http://schema.org/WPHeader">
  ::before
<div class="wrap">…</div>
  ::after
</header>
<div class="homepage-slider">
<ul class="cycle-slideshow" data-cycle-fx="scrollHorz" data-cycle-slides="li" data-cycle-timeout="20000" data-cycle-swipe="true" data-cycle-auto-height="calc" data-cycle-prev=".prev" data-cycle-next=".next" style="position: relative; overflow: hidden;">
<li class="slide cycle-slide cycle-sentinel" style="position: static; top: 0px; left: 0px; z-index: 100; opacity: 1; display: block; visibility: hidden; background-image: url("https://www.excella.com/wp-content/uploads/2015/03/Salesforce-Training-Banner-Darker.png"); background-size: cover; background-repeat: no-repeat;">…</li>
<li class="slide cycle-slide" style="position: absolute; top: 0px; left: 0px; z-index: 95; opacity: 1; display: block; visibility: hidden; background-image: url("https://www.excella.com/wp-content/uploads/2015/03/Salesforce-Training-Banner-Darker.png"); background-size: cover; background-repeat: no-repeat;">…</li>
<li class="slide cycle-slide" style="position: absolute; top: 0px; left: 0px; z-index: 96; visibility: hidden; opacity: 1; display: block; background-image: url("https://www.excella.com/wp-content/uploads/2015/03/Data-Intetegration-Tool-Comparison-Banner.jpg"); background-size: cover; background-repeat: no-repeat;">…</li>
<li class="slide cycle-slide cycle-slide-active" style="position: absolute; top: 0px; left: 0px; z-index: 99; visibility: visible; opacity: 1; display: block; background-image: url("https://www.excella.com/wp-content/uploads/2016/03/USCIS-award-banner.jpg"); background-size: cover; background-repeat: no-repeat;">…</li>
<li class="slide cycle-slide" style="position: absolute; top: 0px; left: 0px; z-index: 97; visibility: hidden; background-image: url("https://www.excella.com/wp-content/uploads/2015/03/Homepage-Hero-3.jpg"); background-size: cover; background-repeat: no-repeat;">…</li>
<li class="slide cycle-slide" style="position: absolute; top: 0px; left: 0px; z-index: 96; visibility: hidden; background-image: url("https://www.excella.com/wp-content/uploads/2015/05/myUSCIS_carousel.jpg"); background-size: cover; background-repeat: no-repeat;">…</li>
</ul>
<div class="slide-controls clearfix">
  ::before
<span class="prev">…</span>
<span class="next">…</span>
  ::after
</div>
</div>
<div class="mobile-subpage-menus clearfix">
<aside class="widget-area"></aside>
</div>
<nav class="nav-primary" itemscope itemtype="http://schema.org/SiteNavigationElement">
  ::before
<div class="wrap">…</div>
  ::after
</nav>
<div class="site-inner">
```

Styles Computed Event Listeners DOM Breakpoints Properties Accessibility Properties

Filter .cls

```
-ms-transition: left 0.25s ease-in-out;
-o-transition: left 0.25s ease-in-out;
transition: left 0.25s ease-in-out;
}
*, input[type="search"] {                                   style.css?ver=0.1.0:2787
-webkit-box-sizing: border-box;
-moz-box-sizing: border-box;
box-sizing: border-box;
}
d  {                                                        user agent stylesheet
splay: block;
}
Inherited from body.home.page.page-id-5.page-template.page-template-front-page.page-template-front-page-php.tribe-bar-is-disabled.header-image.header-f…
body {                                                      style.css?ver=0.1.0:2909
background-color: white;
color: #22221f;
font-family: "Open Sans", sans-serif;
font-size: 15px;
font-weight: 400;
line-height: 1.5;
margin: 0;
}
Inherited from html
html {                                                      style.css?ver=0.1.0:2641
font-family: sans-serif;
-ms-text-size-adjust: 100%;
-webkit-text-size-adjust: 100%;
}
Pseudo ::before element
.author-box:before, .clearfix:before, .entry:before, .entry-content:before, .footer-widgets:before, .nav-primary:before, .nav-secondary:before, .pagination:before, .site-container:before, .site-footer:before, .site-header:before, .site-inner:before, .wrap:before {   style.css?ver=0.1.0:2795
content: " ";
display: table;
}
Pseudo ::after element
.author-box:after, .clearfix:after, .entry:after, .entry-content:after, .footer-widgets:after, .nav-primary:after, .nav-secondary:after, .pagination:after, .site-container:after, .site-footer:after, .site-header:after, .site-inner:after, .wrap:after {   style.css?ver=0.1.0:2811
clear: both;
content: " ";
display: table;
}
```

CSS is **one giant dependency tree**. We need a way to **manage this dependency** at a very low level.

Ways of ordering stylesheets

1. **Mirror the web page**
2. **Thematic chunks** – typography, forms, buttons, etc
3. **Just stick it at the end of the stylesheet** (and regret it later)

SVS Method: Thematic Chunks (more to come on this)

Ordering Stylesheets (Gone Bad)

- **Undoing the CSS:** writing more CSS in order to undo the other CSS
- Poor source order coupled with inherited/inheriting styles can lead to a **lot of waste** and/or redundancy (**code bloat**)

Keep things tidy. Use **thematic chunks.**

Break out like-styles and @import them into a main file. Think of the main file as purely a table of contents.

SVS Adaptation: Our main file is **all.scss**

ONE FILE TO **RULE** THEM ALL,
ONE FILE TO **FIND** THEM,
ONE FILE TO **BRING** THEM ALL,
AND IN THE SASS WAY **MERGE** THEM.

- J.R.R. TOLKIEN (PARAPHRASED)

File Structure: 7-1 Pattern

File Structure: 7-1 Pattern

- 7 Folders, 1 File
  - All partials stuffed into 7 different folders
  - Single file at root level (i.e., **all.scss**)
- Documentation
  - http://sass-guidelin.es/#the-7-1-pattern

7-1 Pattern

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

```
// Import Themes ----- //

    // US Web Style Guide --//

        // Vendor -------------- //
            @import '_scss/themes/usstyleguide/lib/bourbon/bourbon';
            @import '_scss/themes/usstyleguide/lib/neat/neat';
            @import '_scss/themes/usstyleguide/lib/normalize';

        // Core/Grid ----------- //

            @import '_scss/themes/usstyleguide/core/grid-settings';

    // VARIABLES ----------- //
        @import '_scss/variables/variables';

        // Core -------------- //
            @import '_scss/themes/usstyleguide/core/base';
            @import '_scss/themes/usstyleguide/core/grid';
            @import '_scss/themes/usstyleguide/core/utilities';

        // Elements ---------- //
        // Styles basic html elements
            @import '_scss/themes/usstyleguide/elements/typography';
            @import '_scss/themes/usstyleguide/elements/list';
            @import '_scss/themes/usstyleguide/elements/inputs';
            @import '_scss/themes/usstyleguide/elements/buttons';
            @import '_scss/themes/usstyleguide/elements/table';
            @import '_scss/themes/usstyleguide/elements/figure';
            @import '_scss/themes/usstyleguide/elements/labels';


        // Components ---------- //
            @import '_scss/themes/usstyleguide/components/skipnav';
            // @import '_scss/themes/usstyleguide/components/disclaimer';
            // @import '_scss/themes/usstyleguide/components/sidenav';
            // @import '_scss/themes/usstyleguide/components/footer';
            // @import '_scss/themes/usstyleguide/components/forms';
            // @import '_scss/themes/usstyleguide/components/search';
            @import '_scss/themes/usstyleguide/components/alerts';
            @import '_scss/themes/usstyleguide/components/accordions';


// SVS Homegrown ----- //

    // Vendor -------------- //
        @import '_scss/vendor/vendor_overrides';
        @import '_scss/vendor/fontawesome/font-awesome';

    // Abstracts ----------- //
        @import '_scss/abstracts/utils/utils';
        @import '_scss/abstracts/mixins';
        @import '_scss/abstracts/classes';

    // Core ---------------- //
        @import '_scss/core/global';
```

SVS Adaption: We don’t use **pages/** and the US Style Guide is located in **themes/**

Class Naming

Be descriptive – not prescriptive – to keep the focus on **modularization**

Prescriptive:

.list-no-styles

.box-with-shadow

Descriptive:

.main-menu

.callout-box

Descriptive class names and thoughtful comments go hand-in-hand. More on this later…

Context-dependent children should inherit the parent's naming as a prefix to **contextualize context-specific classes**

Contextualize Content-Specific Classes

```
.thing {   }

.thing-heading {   }

.thing-heading-button {   }
```

Think Modularly: Think about what aspects of a thing might be (or become) variable, and style them as such

Recognize Variations

```
/* It's always padded nicely, and bold... */

.button {

      display: inline-block;

      padding: 0.5em 1em;

      font-weight: bold;

}
```

Recognize Variations

```
/* ...but sometimes it's red... */

.button-red {

      background: red;

      color: white;

}
```

Recognize Variations

```
/* ...and sometimes blue. */

.button-blue {

      background: lightblue;

      color: darkblue;

}
```

Style Basic Elements First: CSS provides exceptions to rules, not the rule itself

Style Basic Elements First: **Bad Example**

```
ul {
    list-style: none;
    margin: 0;
    padding: 0;
    /* Yay, now my nav menu doesn't need this! */
}
```

Style Basic Elements First: **Bad Example**

```
.main-section ul {
    list-style: disc;
    margin: 1em 0;
    padding: 0 0 0 1em;
    /* Oh. Now I'll undo that initial reset so these lists
    actually look like lists. And remember to put a .main-section div
    on every page where I want lists to look like lists. */
}
```

Style Basic Elements First: **Good Example**

```
ul {
    list-style: disc;
    margin: 1em 0;
    padding: 0 0 0 1em;
    /* Lists are lists */
}
```

Style Basic Elements First: **Good Example**

```
.nav-menu {
    list-style: none;
    margin: 0;
    padding: 0;
    /* The thing that's not really a list gets the
    special reset styling */
}
```

Keeping key selectors specific.

The more general your key selector, the more likely it is that future code changes will be erroneously affected.

Specific Selectors: **Bad Example**

```
.thing span {
    color: orange;
    /* Is it orange because it's in a span? */
}
```

Specific Selectors: **Good Example**

```
.thing-accent {
    color: orange;
    /* Or is it orange because it’s an accent in
    .thing? */
}
```

Nesting

Avoid unnecessary nesting. The more specific your key selector, the more likely it is that it can stand on its own.

Nesting: **Bad Example**

```
.thing {
      .thing-heading {
            .thing-heading-accent {
                  /* . . . */
            }
      }
}
```

Nesting: **Good Example**

```
.thing {   }

.thing-heading {   }

.thing-accent {   }
```

Mind the depth. Try to keep things no more than 4 levels deep, including pseudo-selectors (:before, :hover, etc.)

```
.nav-menu {
        > li {
                > a {
                        &:hover {
                        }
                }
        }
        ul {
                li {
                }
                ul {
                        li {
                                /* Where am I? Who are you? */
                        }
                }
        }
}
```

Syntax

Use **new lines** to keep code readable.

New Lines: **Bad Example**

```
.thing, .other-thing {
    background: blue; color: red;
}
```

New Lines: **Good Example**

```
.thing,

.other-thing {

    background: blue;

    color: red;

}
```

Keep all element rules before nested children to keep properties close to their selector.

Ordering: **Bad Example**

```
.thing {
        position: absolute;
        top: 7px;
        left: 0;

        .thing-child {
                /* . . . */
        }
        background: red;
}
```

Ordering: **Good Example**

```
.thing {
        position: absolute;
        top: 7px;
        left: 0;
        background: red;

        .thing-child {
                /* . . . */
        }
}
```

Make sure your future self knows what the heck your current self was thinking when you made that z-index: 743.

But, future-you knows how to read code too.

Comments: **Bad Example**

```
.thing {
    color: blue; // Make it blue
    font-weight: bold; // Bold
    z-index: 743;
}
```

Comments: **Good Example**

```
.thing {
    color: blue;
    font-weight: bold;
    z-index: 743; // Tuck between sticky header at
740 and overlay at 745
}
```

The code should tell you how, the comments should tell you why.

Never use **!important**

Think long and hard before using **!important**. It can easily wreak havoc on a codebase, and make debugging CSS issues a nightmare.

Once there’s one !important, you’ll inevitably have to add more.

Going Beyond ‘It Works’

Beyond ‘It Works’

1. Keep things **DRY**
   - Check to see if a style has already been written
   - Utilize the power of SASS (specifically variables and mixins)
2. Think twice about the importance of using **!important**
3. Code tells you how, **comments tell you why**

Beyond ‘It Works’

4. Name Things for **Reuse**
5. Consistency: Naming Selectors
    - Choose either .camelCase, .hyphened-selectors, or .underscored_selectors (limit a mixture of more than two)
    - For **SVS**, we’ll stick to **.hyphened-selectors** as much as possible (use discretion when using underscores, but don’t use camelCases)
        - **Reason:** US Style Guide primarily hyphenates their selectors.

Resources

- CSS Tricks (https://css-tricks.com/)
- Sass Guidelines (http://sass-guidelin.es/)
- Can I Use? (http://caniuse.com/)
- SitePoint (http://www.sitepoint.com/html-css/)

Questions?