Theming the web with CSS Custom Properties

When I first heard about CSS Custom Properties (CSS Variables) one of the things I found most intriguing was the possibility of creating themes and have multiple themes available for use. In this post we'll explore some new (or not so new) pseudo elements that will make working with CSS variables easier, we'll explore what are CSS Custom Properties and the acompanying Javascript API for working with these new variables.

> Note that, for this post, I'm using CSS custom properties and CSS variables interchangeably since both terms represent the same spec.

Syntax and the :root pseudo element

One of the first things we need to do is place the variable declarations as high up in the cascade as possible to make the cascade and inheritance work for use. Normally that would mean putting everything in the `html` element but there may be times when that is not specific enough. Enter the :root pseudo class. It is identical to the `html` element selector except that it has higher specificity (in a conflict between html and :root, the later wins).

Now that we know where we'll put them we can talk about the syntax for declaring variables and using in our code.

Declaring CSS variables is easy, just start the name of the attribute with two dashes (`--`) before the name.

To use a CSS variable put the name of the variable inside a `var()` declaration and use that as the value of the attribute. We'll look at more practical examples later on.

in the example below we define 2 variables in `:root`. One to define the color of our text and one to define an accent color. Later in the stylesheet we use `--main-text-color` as the value for the color of our `#navigation h1` element.

Variables are reusable. Once they are defined we can use them everywhere that we need that information. We are using colors so far but later we'll see variables used in media queries.

```
:root {
  /* colors */
  --main-text-color: rebeccapurple;
  --main-accent-color: #06c;
  /* doc characteristic */
  --main-line-height: 1.25;
  --main-font-size: 16px;
  --main-content-width: 42em;
}
/* The rest of the CSS file */
#navigation h1 {
  color: var(--main-text-color);
}
```

We can also work with Javascript to read and write CSS variables using Javascript. To do so we query the HTML element using `window.getComputedStyle` for `document.querySelector("html")` element.

To read a property use `get.PropertyValue` for the CSS variable that you want to know about.

```
var htmlStyles = window.getComputedStyle(document.querySelector("html"));
// returns "rebeccapurple"
var myColor = htmlStyles.getPropertyValue("--main-text-color");
```

To set a variable use `style.setProperty` with two parameters, the name of the variable (using `--` as the first two characters) and the value.

```
document.querySelector("html").style.setProperty("--main-color", "#06a");
```

Theming

When revisiting themes I discovered that you can do it in one of two ways. We can place all our variables in the `:root` pseudo element and make sure that we give them different names to qualify them (like I did below by prefixing all the variables for the amber theme with the name of the theme) or assign them to classes and use the class as a namespace for each individual group of variables.

I will go with the first option. Less headaches for me that way; it's less things to track and less places to make changes :-)

```
:root {
  --dark-primary-color: #1976D2;
  --default-primary-color: #2196F3;
  --light-primary-color: #BBDEFB;
  --text-primary-color: #FFFFFF;
  --accent-color: #448AFF;
  --primary-text-color: #212121;
  --secondary-text-color: #727272;
  --divider-color: #B6B6B6;
  /* amber- theme */
  --amber-dark-primary-color: #FFA000;
  --amber-default-primary-color: #FFC107;
  --amber-light-primary-color: #FFECB3;
  --amber-text-primary-color: #212121;
  --amber-accent-color: #FFC107;
  --amber-primary-text-color: #212121;
  --amber-secondary-text-color: #727272;
  --amber-divider-color: #B6B6B6;
}

.container {
  padding: 2em;
}


.card1 {
  background-color: var(--default-primary-color);
```

```
  padding: 1em;
}

.card1 p {
  color: var(--primary-text-color);
}

.amber {
  background-color: var(--amber-default-primary-color);
  border: 1px solid black;
  margin-top: 1em;
  padding: 1em;
}

.amber p {
  color: var(--amber-default-primary-text-color);
}
```

In the `:root` pseudo element I've added the colors for two themes, one based in blues and gray and an amber one based on yellow and orange. To illustrate how it works I created two different sections, each using a different theme.

for the adventurous you can create a theme switcher with javascript and use classes to define your themes. You can then let users define the look of your content.

For the full example, without the switcher, see http://codepen.io/caraya/full/JERYpL/

Media queries

We can also use variables to provide a central point to keep the sizes of our media queries. In the example below we have three breakpoints with their associated sizes in `em`. If we change our mind about the sizes later we only have to change the sizes of the breakpoint variables in one place rather than have to change each place where the value appears.

We can also add device specific or content specific breakpoints if we need them.

```
:root {
  --query-small-breakpoint: 25em;
  --query-medium-breakpoint: 35em;
  --query-large-breakpoint: 50em;
}

@media all and (max-width: var(--query-small-breakpoint)){
  /* small screen styles */
}

@media all and (min-width: var(--query-small-breakpoint))
           and (max-width: var(--query-medium-breakpoint)) {
  /* larger than small and smaller than medium*/
}

@media all and (min-width: var(--query-medium-breakpoint))
           and (max-width: var(--query-large-breakpoint)) {
  /* bigger than medium and smaller than larger*/
}

@media all and (min-width: var(--query-large-breakpoint)) {
  /* larger than large */
}
```

internationalization

The last efficiency of using CSS variables that I find particularly useful is for internationalization.

We can leverage the `:lang` pseudo class to generate localized text like the example below. We have three options or the language attribute:

- If there is no language defined on the document or the language is English (en) we provide an English translation
- If the language is German (de) we provide a german translation
- If the language is Spanish (es) we provide the Spanish language

Then for every external link we

```
:root,
:root:lang(en) {
  --message-external-link: "external link";
}

"external link":root:lang(de)external-link: "externer Link";
}

:root:lang(es) "externer Link":root:lang(es)"Link externo";
}

a[href^="http"]::after {conten"Link externo"a[href^="http"]::after {conte
```

Where to next?

These are some examples of what you can do using CSS variables. I will post more ideas as I think about them ;-)