Programmatic Access

This lesson discusses how to have programmatic access to all the data we define in CSS.

What is programmatic access?

As developers who use web technologies, we're used to working across three different but cooperative languages: HTML (the layout), JS (the script), CSS (the style).

Sometimes, it's beneficial to be able to access what we define in CSS from the script. This is what we mean by gaining programmatic access to the values we define in our CSS/SCSS.

Let's look at a use case. Assuming we have 3 different themes in our app, we'll have those defined somewhere in our CSS/SCSS files. It's a common need to display some kind of page where the end user can select what theme they want. In such a case, we'll want a way for our code to get the names of those themes. We *could* put the theme names in our code file in addition to their definitions in CSS/SCSS, but that would be redundant and will double the places we'll have to edit if we ever change anything.

What we need is a way to obtain the different CSS values we're defining in CSS programmatically.

Explanation

In JavaScript, we do this by obtaining the computed styles of an element:

```
const style = getComputedStyle(element);
const value = style.getPropertyValue(prop);
```

The `prop` above can be any CSS prop or CSS variables. Our strategy here is to somehow put the theme names in a CSS variable.

Let’s assume we have our themes defined as a SCSS `map`:

```
$themes: (
  'light-theme': (...),
  'dark-theme': (...)
);
```

To get a list of the theme names we’ll map the keys to obtain a list of the names:

```
$theme-names: map-keys($themes);
```

Then, put the result in a CSS variable so that we can obtain it later in code:

```
body {
  --theme-names: #{$theme-names};
}
```

The `--theme-names` variable will contain a comma delimited list of the names now.

Then, obtaining those names in code becomes easy:

```
const style = getComputedStyle(document.body);
const names = style.getPropertyValue('--theme-names');
const themeNames = names.split(',').join(', ');
```

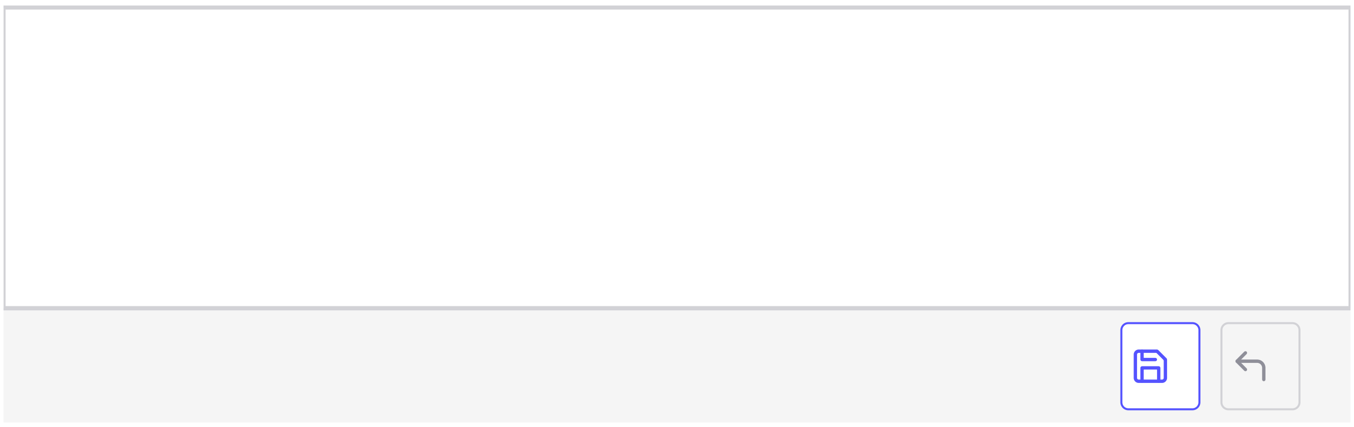
Putting all of this together:

Output

JavaScript

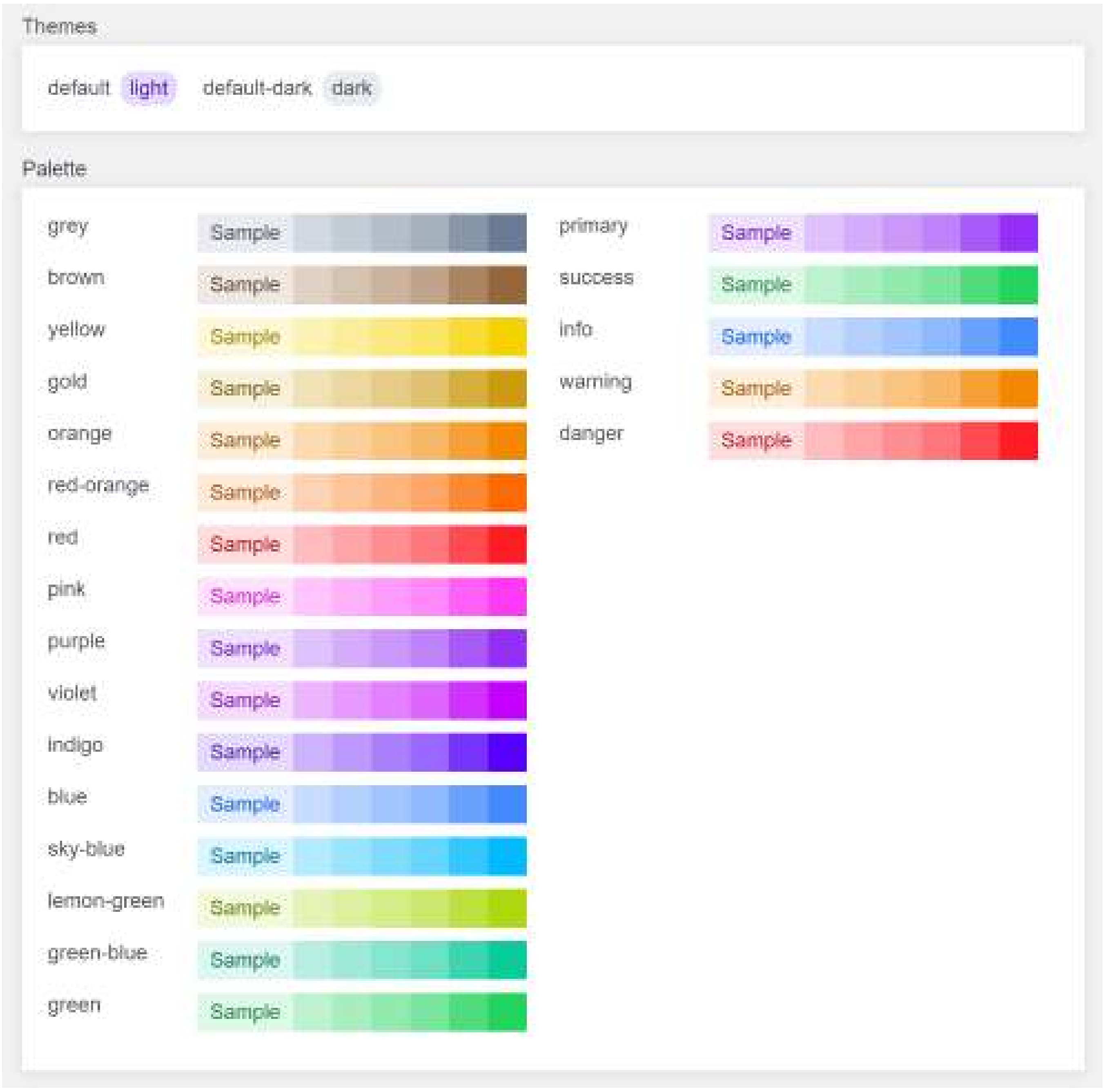
HTML

CSS (SCSS)



Using this method, we’ll be able to gain access to most things we define in CSS. So that even when we get to a point where we have a platform that’s fully customizable, we’ll easily be able to obtain all the values dynamically and keep a single source of truth. We’ll be able to create a page that’s dynamic and changes with the definitions in our CSS.

Here’s an example of such a page, taken from the basic sample in `css-theming`:



Themes

default light   default-dark dark

Palette

grey	Sample	primary	Sample
brown	Sample	success	Sample
yellow	Sample	info	Sample
gold	Sample	warning	Sample
orange	Sample	danger	Sample
red-orange	Sample		
red	Sample		
pink	Sample		
purple	Sample		
violet	Sample		
indigo	Sample		
blue	Sample		
sky-blue	Sample		
lemon-green	Sample		
green-blue	Sample		
green	Sample		

With this, having access to all the design values programmatically is possible without any redundancy. Now, let’s move on to discuss the different types of themes.