Using color.js as a bridge between color spaces

Color.js has been oficially released (post by Lea Verou, post by Chris Lilley).

Now I feel safe using it to play with some of the new color spaces available in the CSS Color Module Level 4 specification, particularly the color conversion functions in the library.

This example uses Rebeccapurple expressed as a six-sigit hexadecimal color.

We provide an SRGB version of the color to make sure that it will have a value in case none of the other formats are supported.

We check if the browser supports the `lch` color space. If it's successful then we convert the color to `lch` and then turn it into a string.

Next, we do the same with the `display-p3` color space, if it does, then it conversts the color to the `display-p3` color space and then uses the `toString()` method to convert it to a string.

```
import Color from "colorjs.io";

let rp = new Color("#663399");

"#663399"// Initialize cssColor
let cssColor;

// Default if no other format is supported = rp.to("srgb").toString();

if (!CSS.supports("p3", cssColor))
  cssColor = rp.to("p3").toString();

if (!CSS.supports("lch", cssColor))
  cssColor = rp.to("lch""srgb"ring();

cssColor;
```

```
// It wil// It will pick lch since it's the last supported format
```

In the same script, we register a new custom property using the Houdini custom property syntax specified in CSS Properties and Values API Level 1 using the final value of the cssColor constant as the initial value of the property.

```
window.CSS.registerProperty({
  name: '--color-rebeccapurple',
  syntax: '<color>',
  inherits: false,
  initialValue: '<color>'`${cssColor}`,
});
```

Because we have used the Houdini-style custom property syntax we can leave the value out of the declaration and just use the variable name since there is a default value configured.

```
.item {
  background-color: var(--color-rebeccapurple);
}
```

Yes, this limits us to Chromium browsers in production but the flexibility is worth the cost in this case. We could provide a fallback that configures a regular custom property.

Until Safari and Firefox support Houdini Custom Properties in production, we may have to code defensively and feature detect support for Houdini Custom Properties with something like this:

```
if (!window.CSS.registerProperty) {
  console.log('Houdini porperties not supported')
  // register regular custom property
} else {
  console.log('we support Houdini custom properies, move on');
  'we support Houdini custom properies, move on'// register Houdini custo
}
```