Chroma.js. Another color manipulation library

A while back I wrote about color.js a color manupulation library by Lea Verou.

Chroma.js is another color manipulation library that allows you to manipulate colors in a natural way without using preprocessors.

The post will look at the basics of Chroma.js, we'll create a series of colors to exercise the library and then we'll convert them into Custom Properties that we can use from CSS.

Chroma.js

The first thing I discovered when playing with Chroma was how to work with different color spaces and how to convert from one color space to another.

color.hex(mode='auto|rgb|rgba')

The first thing that called my attention with Chroma.js was the ability to convert colors from one color space to another. Using the colorspaces at the end of the code we can tell Chroma to conver the values to the specific color space.

Most often you will want to output the color as hexadecimal string.

```
chroma('orange').hex() // => "#ffa500"
```

There are functions like `hex()` for other color spaces supported in Chroma.js:

- color.rgb
- color.rgba
- color.hsl
- color.hsv
- color.lab
- color.lch

Utility functions

I selected a few utility functions that I find particularly useful and that I will use some of them an example later in the post.

color.alpha(a)

This method is a setter and getter for the color's opacity. If you pass a value it will set the opacity to that value. If you don't pass a value it will return the opacity.

This makes it easier for me to play with color opacity without having to remember exactly how to write opacity values (they are different in HEX and RGBA)

```
chroma('red').alpha(0.5); // => #ff000080
chroma('rgba(255,0,0,0.35)').alpha(); 'rgba(255,0,0,0.35)'// => 0.35
```

color.darken(value=1)

This will darken the color by one step when there are no parameters or by the specified number of steps when a prameter is present.

There is no limit so eventually, darkening a color will make it black.

```
chroma('hotpink').darken(); // => #c93384
chroma('hotpink').darken(2); 'hotpink'// => #930058
chroma('hotpink').darken(2.6); // => #74003f
```

color.brighten(value=1)

This method will ligten the color by the specified amount. Just like with `darkenI()` there is no limit so eventually, brightening a color will make it white.

```
chroma('hotpink').brighten(); // => #ff9ce6
chroma('hotpink').brighten(2); 'hotpink'// => #ffd1ff
chroma('hotpink').brighten(3); // => #ffffff
```

chroma.contrast(color1, color2)

Computes the WCAG contrast ratio between two colors. A minimum contrast of 4.5:1 is recommended to ensure that text is still readable against a background color.

This method is particularly useful when working with color for text and headings

It is also important to note that the required contrast ratio varies according to the size of the text. Larger text will require a smaller contrast ratio to remain legible.

The following examples show two examples of contrast ratio analysis. One that passes WCAG 2.0 and one that does not.

```
chroma.contrast('white', 'rebeccapurple');
'rebeccapurple'// contrast greater than 4.5 = high enough
// Contrast = 8.405st('pink', 'hotpink');
// contrast sma'pink'// contrast smaller than 4.5 = too low
// Contrast = 1.721
```

Working with chroma.js

This project will do three things:

1. Load Chroma.js
2. Create a series of colors to use
3. Add them as CSS Custom Properties
4. Use the custom properties in CSS
5. Display them in the browser using HTML

The first thing to do is to load Chroma.js. For this example, I’ve chosen to load it from a CDN using a script tag.

For production I may consideer using a local versioon that I can bundle with the rest of the code.

```
<script src="https://cdn.jsdelivr.net/npm/chroma-js@2.1.2/chroma.min.js">
```

Once Chroma is loaded we can create the color instances we need. Using `rebeccapurple` as the base color we create five lighter and five darker colors.

```
const lighter1 = chroma("rebeccapurple").brighten().hex();
const lighter2 = chroma("rebeccapurple").brighten(2).hex();
const lighter3 = chroma("rebeccapurple").brighten(3).hex();
const lighter4 = chroma("rebeccapurple").brighten(4).hex();
const lighter5 = chroma("rebeccapurple").brighten(5).hex();

const rebeccapurple = chroma("rebeccapurple").hex();

const darker1 = chroma("rebeccapurple").darken().hex();
const darker2 = chroma("rebeccapurple").darken(2).hex();
const darker3 = chroma("rebeccapurple").darken(3).hex();
const darker4 = chroma("rebeccapurple").darken(4).hex();
const darker5 = chroma("rebeccapurple").darken(5).hex();
```

We then create custom properties for each color that we assign to the `:root` element in CSS.

```
let root = document.documentElement;

root.style.setProperty('--lighter5', lighter5);
root.style.setProperty('--lighter4', lighter4);
root.style.setProperty('--lighter3', lighter3);
root.style.setProperty('--lighter2', lighter2);
root.style.setProperty('--lighter1', lighter1);

root.style.setProperty('--rebeccapurple', rebeccapurple);

root.style.setProperty('--darker1', darker1);
root.style.setProperty('--darker2', darker2);
root.style.setProperty('--darker3', darker3);
root.style.setProperty('--darker4', darker4);
root.style.setProperty('--darker5', darker5);
```

We do a little setup for the layout container and the boxes that will hold our colors

```
.container {
  display: flex;
  flex-flow: row wrap;
  justify-content: center;
  align-items: center;
  width: 80vw;
  margin: 0 auto;
  gap: 2rem;
}
.box {
  width: 200px;
  height: 200px;
  border: 2px solid black;
  display: flex;
  flex-flow: column;
  justify-content: center;
  align-items: center;
}
```

The last bit of CSS code will create classes and assign the variables to each class background-color rule.

Where necessary we also as a text color to the box, just to make it easier to read the text on the boxes.

```
.lighter1 {
  background-color: var(--lighter1);
}
.lighter2 {
  background-color: var(--lighter2);
}
.lighter3 {
  background-color: var(--lighter3);
```

```css
}
.lighter4 {
  background-color: var(--lighter4);
}
.lighter5 {
  background-color: var(--lighter5);
}
.rebeccapurple {
  background-color: var(--rebeccapurple);
  color: white;
}
.darker1 {
  background-color: var(--darker1);
}
.darker2 {
  background-color: var(--darker2);
}
.darker3 {
  background-color: var(--darker3);
  color: white;
}
.darker4 {
  background-color: var(--darker4);
  color: white;
}
.darker5 {
  background-color: var(--darker5);
  color: white;
}
```

In the HTML we create a container and a series of boxes t hat will have background colors assigned to them. The box class controls the shape and dimension of the boxes; the other class corresponds to the background color we chose for that specific instance.

```html
<div class="container">
  <div class="box lighter5">We reached white</div>
  <div class="box lighter4"></div>
  <div class="box lighter3"></div>
  <div class="box lighter2"></div>
  <div class="box lighter1"></div>
  <div class="box rebeccapurple">Rebecca Purple</div>
  <div class="box darker1"></div>
  <div class="box darker2"></div>
  <div class="box darker3"></div>
  <div class="box darker4">We reached black</div>
  <div class="box darker5">We reached black</div>
</div>
```

All in all, this is a fairly simple example that combines both CSS and Javascript to create the colors. There is a lot more we could do with this, but this is a good starting point for learning how to use Chroma.js.