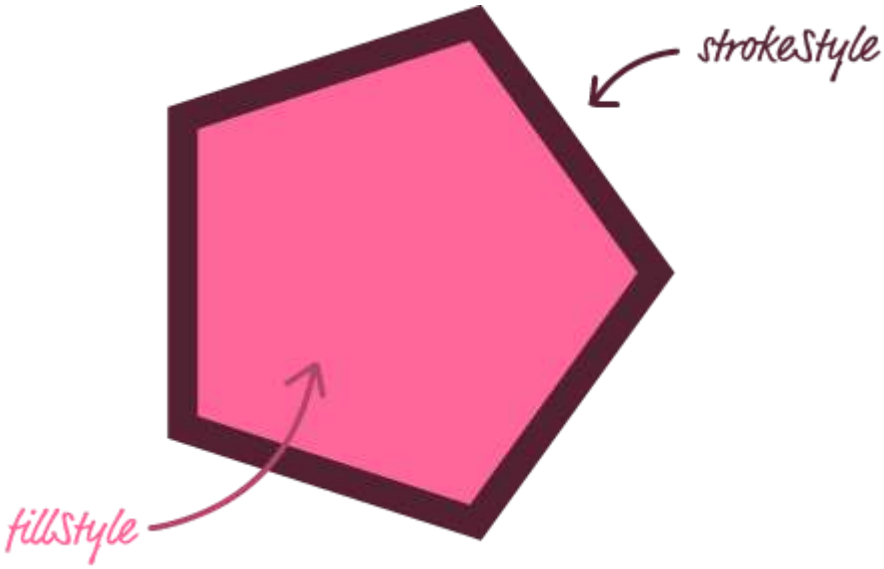
Colors, strokeStyle, and fillStyle!

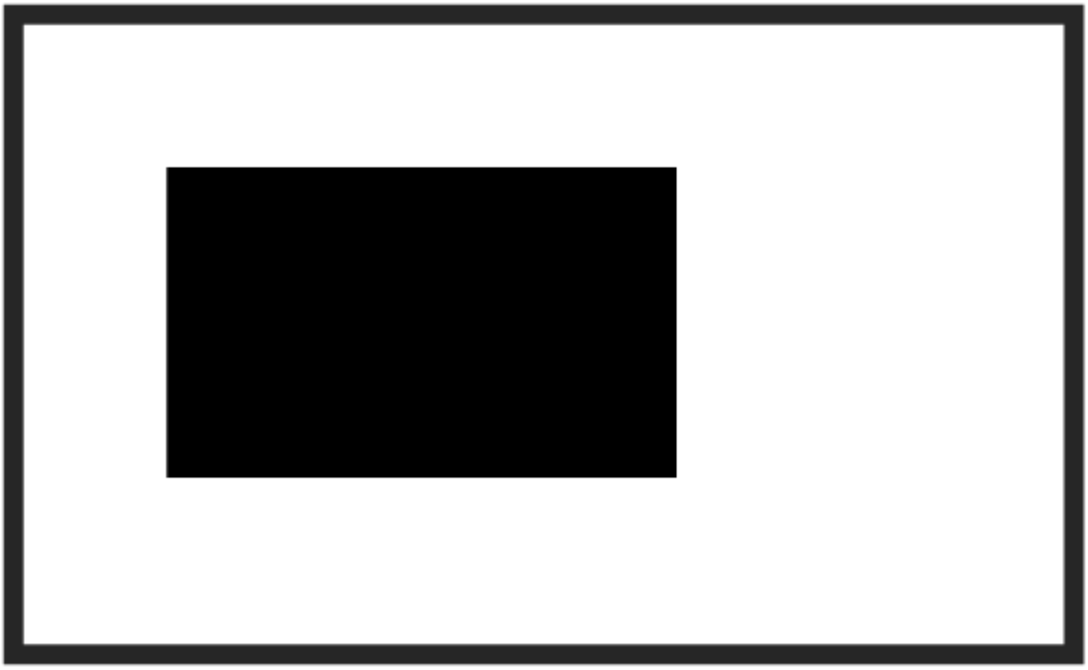
WE'LL COVER THE FOLLOWING

The primary way you colorize content is by setting the `strokeStyle` property for shape outlines and the `fillStyle` property for the shape insides:



Between these two properties, you can color everything from lines to geometric shapes to text.

Let's say we have a rectangle that looks as follows:



The code responsible for this work of art looks like this:

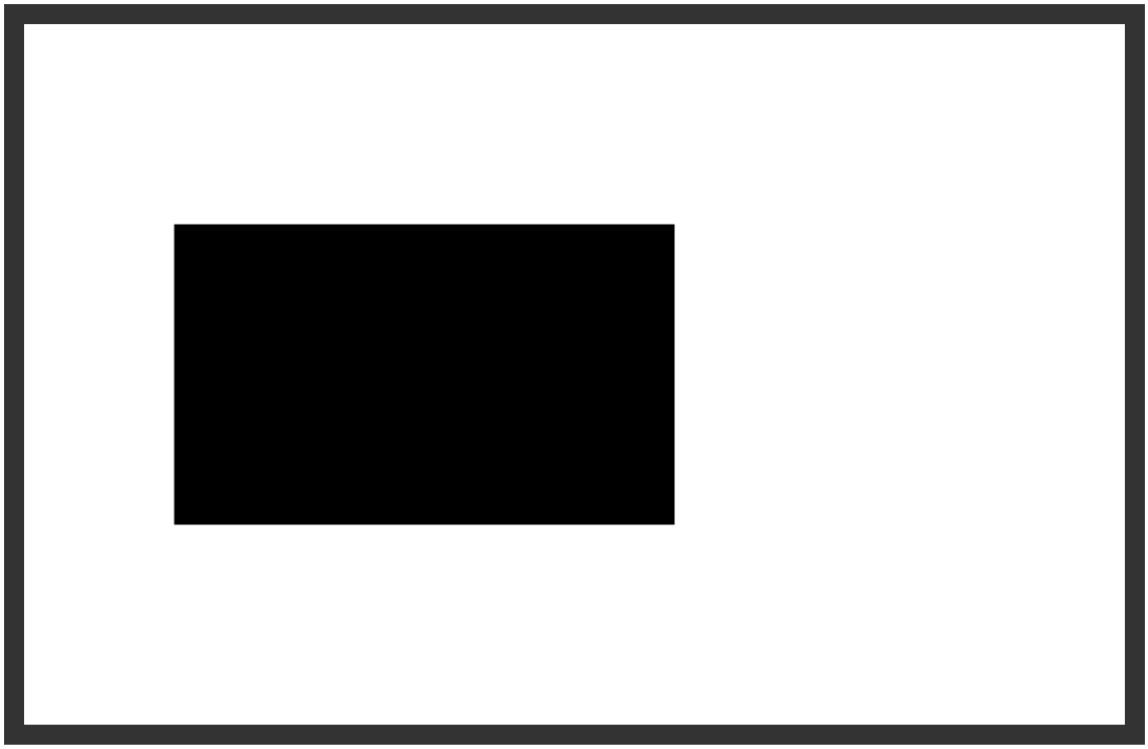
HTML JavaScript

javascript

```
var canvasElement = document.querySelector("#myCanvas");
var context = canvasElement.getContext("2d");

context.beginPath();
context.rect(75, 100, 250, 150);
context.fill();
```

output



To set the fill color of this element, add line 3 that specifies the `fillStyle` property:

```
context.beginPath();
context.rect(75, 100, 250, 150);
context.fillStyle = "#FFCC00";
context.fill();
```

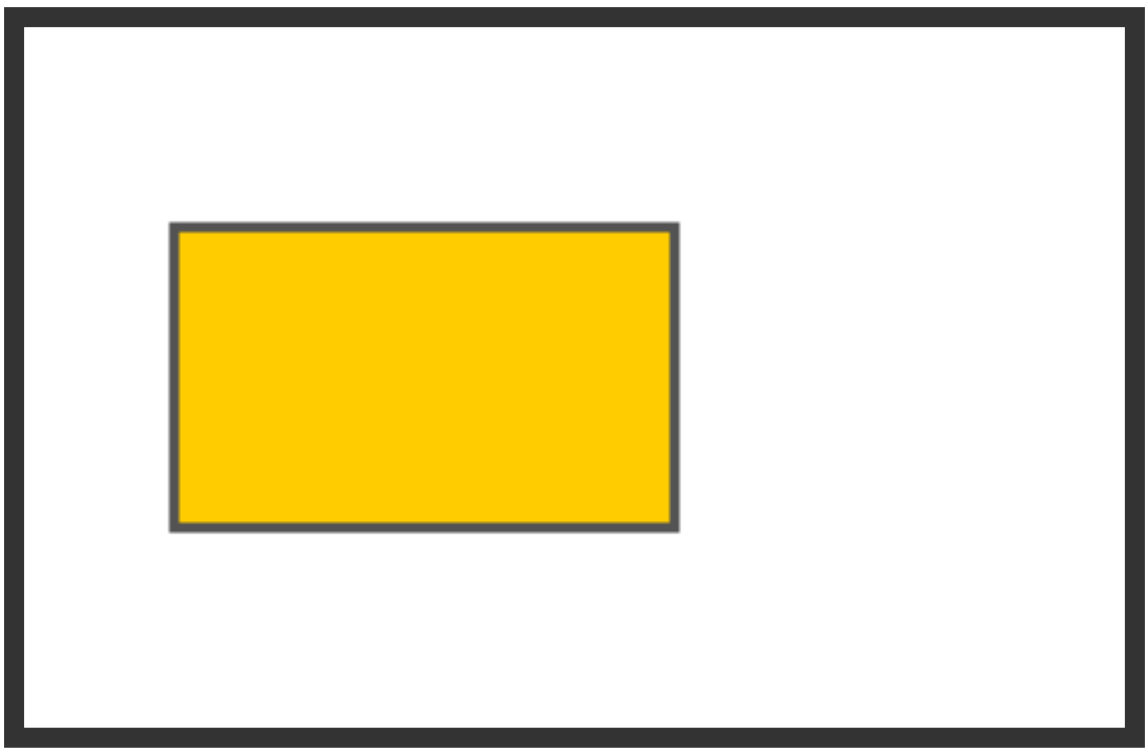
Let's not stop with just the fill. Since we are already here, we are going to next add a thick outline and give that a color via the `strokeStyle` property. Add lines 9-11 to your code:

HTML JavaScript

javascript

```
1  var canvasElement = document.querySelector("#myCanvas");
2  var context = canvasElement.getContext("2d");
3
4  context.beginPath();
5  context.rect(75, 100, 250, 150);
6  context.fillStyle = "#FFCC00";
7  context.fill();
8
9  context.lineWidth = 5;
10 context.strokeStyle = "#535353";
11 context.stroke();
```

output



The two lines of code that took our pretty drab looking rectangle and helped make it a bit more lively are the `fillStyle` and `strokeStyle` properties. You can argue that the `fillStyle` probably had more to do with it since the outline is still a pretty dull shade of gray, but anyway...but what we are going to do from here on out is take a look at the various ways you have for specifying colors. While I will be focusing only on the `fillStyle` property, everything you see will apply equally to `strokeStyle` as well. Me ignoring `strokeStyle` is purely done for aesthetic reasons. After all, changing the fill color of a rectangle looks a lot more impressive than changing just the outline color!

Hex Color Values

Let us start our exploration by looking at what we have already specified for the `fillStyle` property in our code:

```
context.fillStyle = "#FFCC00";
```

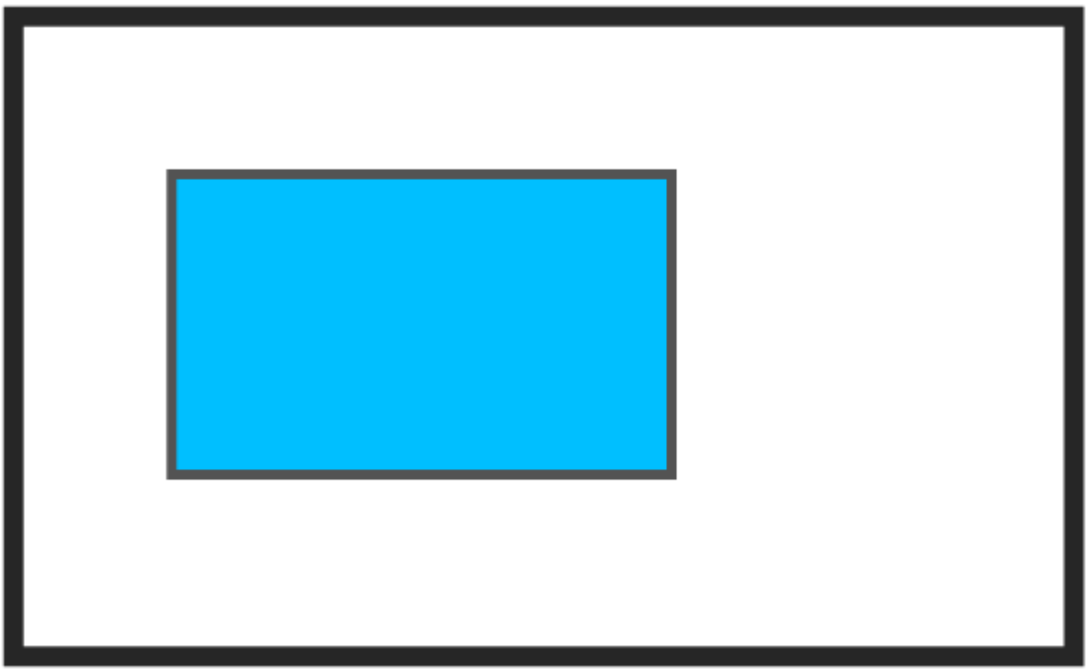
Here, we've assigned a **hex color value** that corresponds to a nice shade of yellow. This hex value corresponds to the Red, Green, and Blue components of the color we are trying to represent. Chances are, if you are copying color values defined in an image editor, you are going to be seeing values in this format.

CSS Color Keywords

You can also specify CSS color keywords such as **yellow, navy, firebrick**, and a whole bunch of colors with really cool names:

```
context.fillStyle = "deepskyblue";
```

Have you ever wondered what deepskyblue looks like? Well, wonder no more:



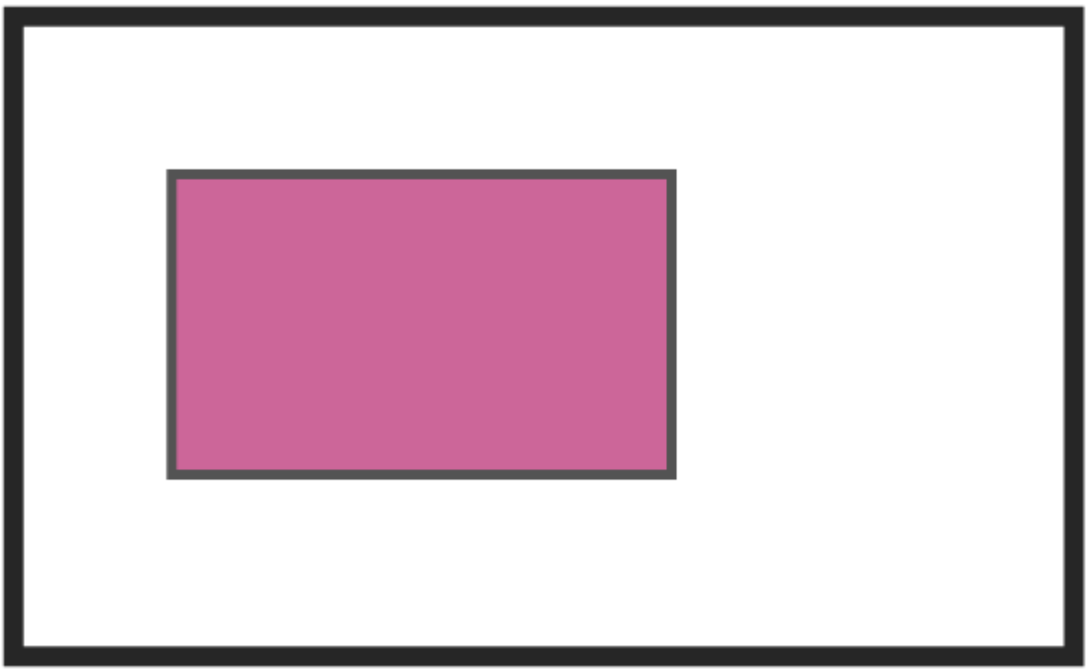
You can see the full range of color keywords in this MDN article. For the most part, you probably won't be using color keywords much. The reason is that these named colors are extremely web-specific, and most image editing tools have no idea to either accept or export color keywords. Instead, many image editing tools prefer good old RGB. Speaking of which...

RGB Values

As it turns, you can also specify colors to the `fillStyle` and `strokeStyle` properties as an RGB value by using the **rgb** function:

```
context.fillStyle = "rgb(204, 102, 153)";
```

Replacing our rectangle's `fillStyle` value with this RGB entry looks as follows:



Your RGB function and the values are expected to be in the form of strings. That is why you see them wrapped inside quotation marks. With that said, we are inside JavaScript. Nothing prevents you from parameterizing the values by doing some good, old-fashioned string magic:

```
context.fillStyle = "rgb(" + r + ", " + g + ", " + b +")";
```

For many interactive scenarios, you will find yourself doing this quite often, so just be aware that you can totally do this.

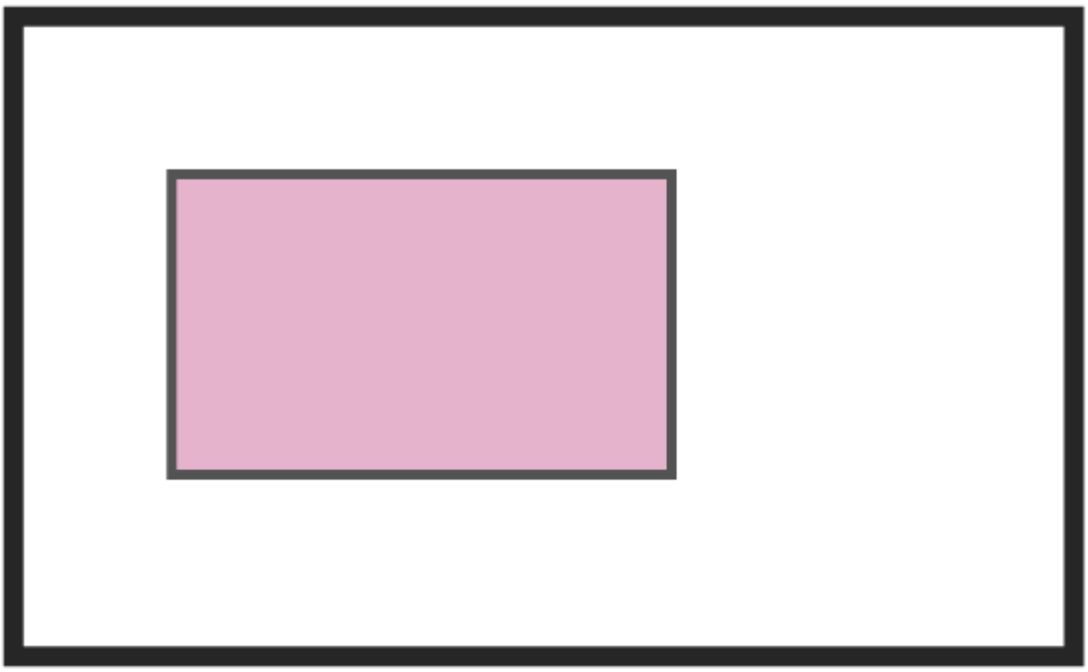
Transparency and RGBA

So far, we've looked at specifying a color as a hex value, CSS color keyword, and a RGB value. There is just one more variant for specifying a single color, and that is RGBA. The "A" stands for alpha…or transparency, and you specify that value as a number between 0.0 (fully transparent) to 1.0 (fully opaque).

If we to extend the earlier example by giving our fill color a 50% transparency, here is what the rgba declaration would look like:

```
context.fillStyle = "rgba(204, 102, 153, .5)";
```

This will translate to a more muted version of the pink color we saw earlier:



You do have one more way for setting the transparency, and that way is by setting it globally via the `globalAlpha` property:

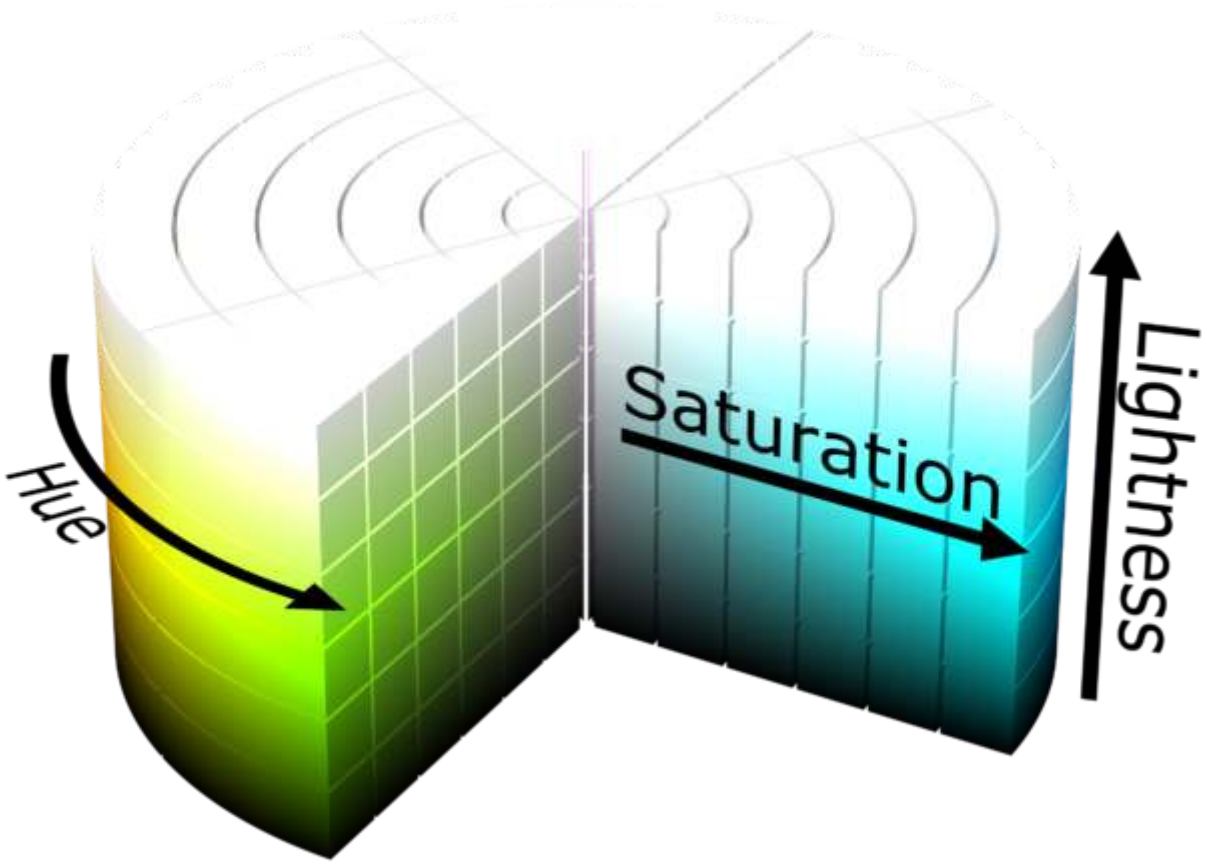
```
context.globalAlpha = .3;
```

With this line, we are setting the opacity of **everything** in our `canvas` to be just 30 percent. Your strokes and fills will be impacted, and the `globalAlpha` property overrides any alpha value you may have specified inside your **rgba** function. Use this property at your own peril.

Using HSL Values

We've spent a lot of time taking about RGB values. While a lot of the colors you will use will be in that format, you aren't limited to just RGB, though. You can also specify colors in the HSL space. In HSL, you define a color by specifing values for **Hue, Saturation**, and **Lightness**.

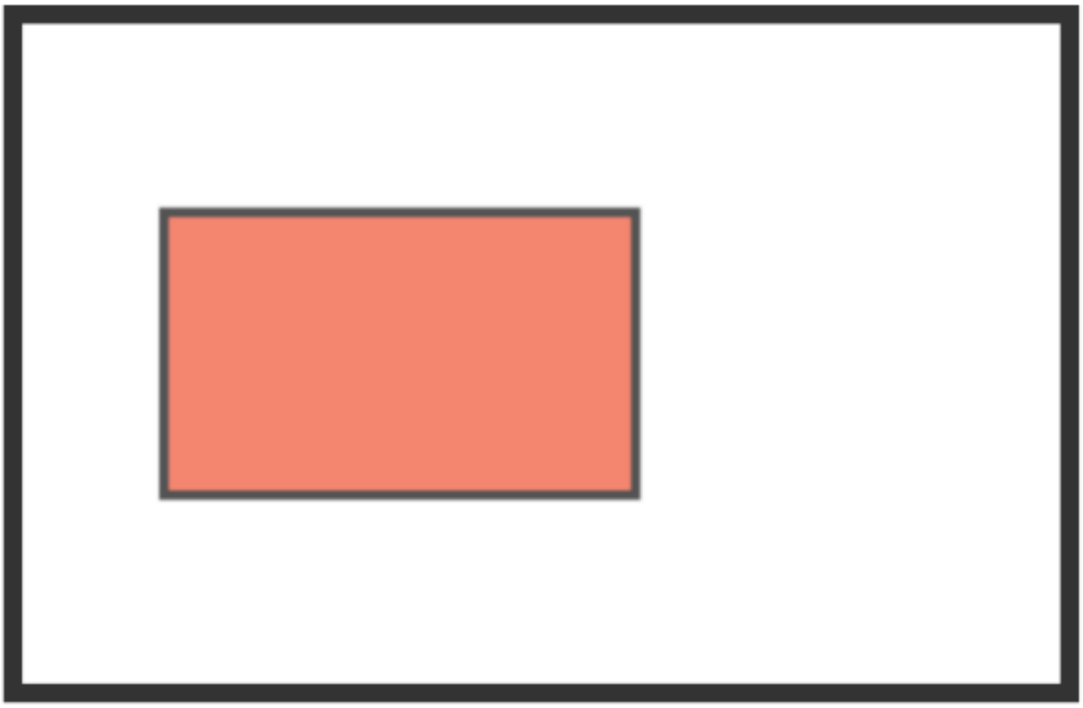
You can think of HSL colors modeled as shown in the following three-dimensional cylinder (created by SharkD):



The value for Hue is specified in degrees going from 0 to 360. Both Saturation and Lightness are percentages that you specify as as percentage values. Below is what the `fillStyle` property set to a HSL value looks like:

```
// a salmon-ish color specified in HSL
context.fillStyle = "hsl(9, 83%, 70%)";
```

This translates into a rectangle that looks like the following:



Just like what we saw with RGB values, you can tackle on a value for alpha to define a HSLA color. Below is an example of an HSLA color that is green-ish looking with an opacity of 50%:

```
context.fillStyle = "hsla(100, 83%, 70%, .5)";
```

HSL values are very common when working with colors from design tools, but they are great to use if you want to manipulate colors programmatically. Instead of dealing with three independent values like you have with RGB, in the HSL world, you can go through a range of colors by just modifying one of the H, S, or L values. That's a convenience you should totally not overlook if you ever find yourself manipulating color values in JavaScript.