Specifying Gradients

WE'LL COVER THE FOLLOWING

There is just one more thing we need to look at before our look into working with colors is wrapped up. That thing is **gradients**! So far, we've only specified a single color value for the various things we've been coloring. Gradients change all that. With gradients, you have the ability to specify multiple colors and how those colors blend in with each other. In the following sections, let's learn all about them.

The Linear Gradient

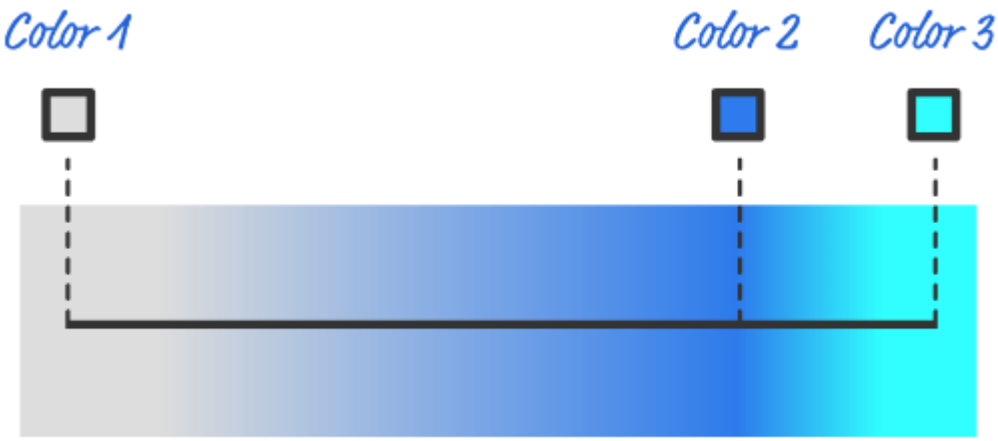
The first gradient variation we'll learn about is the linear one. A linear gradient is one where you have two or more colors on a virtual (invisible) straight line. These colors blend evenly as you move from one color to another on this line. That definition is probably more complicated than it needs to be, but hopefully the following image makes more sense:

We have two colors one two ends of a rectangle. On one end is a light gray. On

the other end is cyan. Between those two points, the colors gradually blend

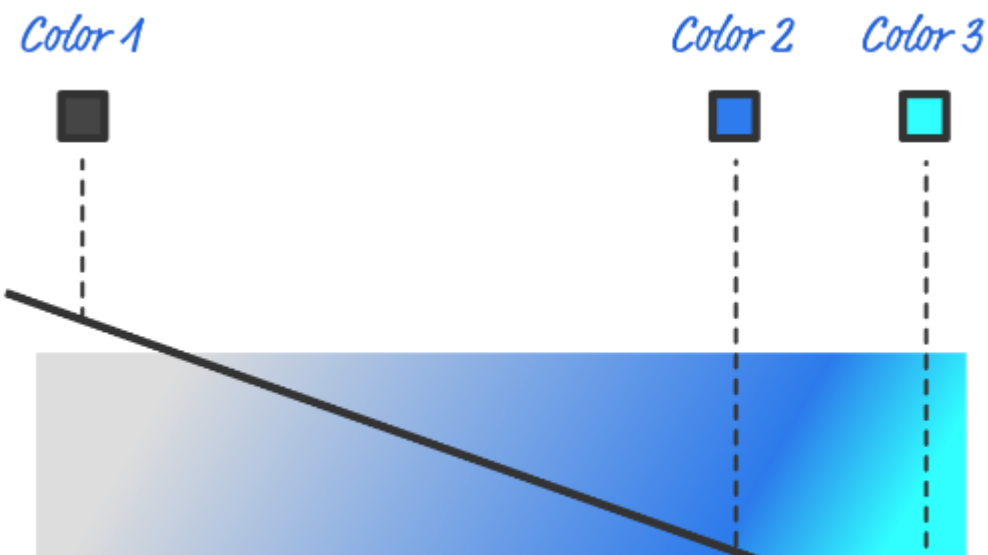
between the gray and cyan to give you this smooth look. That blending is one of the hallmark features of a gradient. Now, let's go a bit deeper.

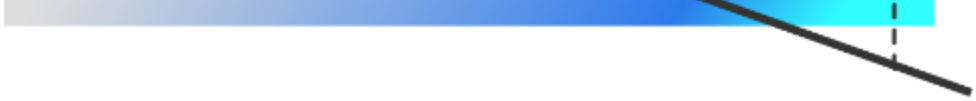
These two points on either end of our gradient have a more formal name. They are known as **gradient color stops** or just **color stops**. You can have as many color stops as you want, and each color stop defines its own color and position on the virtual straight line:



In this version, we modified our earlier example by adding a slightly darker blue gradient stop near the right. Notice that how our colors blend has changed to accommodate this new color that has been thrown in. By adding (or removing) gradient stops and positioning them in various places on our virtual straight line, you can create a variety of different gradient effects.

There is just one more thing to cover before we start looking at how to use this linear gradient to colorize what we draw on the `canvas`. This virtual line all of our gradient stops live on can be rotated in all sorts of ways:





This rotation ends up changing the angle our colors transition from one value to another. By rotating the straight line, you can create gradients whose colors change horizontally (like we saw earlier) to ones where the colors are stacked like pancakes to ones that change in any other angle in-between.

Now that we have seen all of this, let's learn how to use linear gradiens in our `canvas`! The way you specify a linear gradient is by using the `createLinearGradient` method. This method takes four arguments that map to the starting and ending X and Y points of our virtual straight line:

```
context.createLinearGradient(x_0, y_0, x_1, y_1);
```

What this method returns is a `CanvasGradient` object that you can then add your color stops to using `addColorStop`. Let's re-create the gray/blue/cyan gradient that we've been seeing so far.

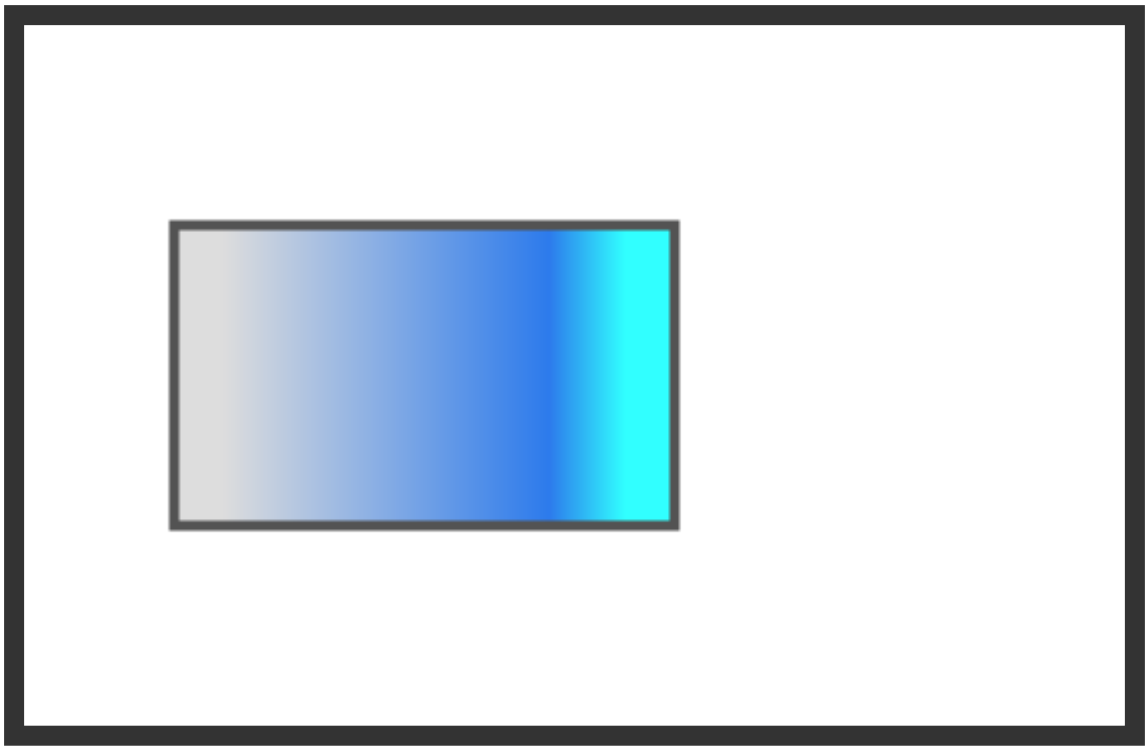
Take a look a the following code where anything gradient-related is between lines 4-13:

HTML JavaScript

javascript

```
1  var canvas = document.querySelector("#myCanvas");
2  var context = canvas.getContext("2d");
3
4  context.beginPath();
5  context.rect(75, 100, 250, 150);
6
7  // virtual line length matches our rectangle's dimensions
8  var gradient = context.createLinearGradient(75, 0, 325, 0);
9
10 // our three color stops
11 gradient.addColorStop(0.1, "#DDDDDD");
12 gradient.addColorStop(0.75,"#2D7BEC");
13 gradient.addColorStop(0.9, "#31FFFF");
14
15 // assigning the gradient
16 context.fillStyle = gradient;
17 context.fill();
18
19 context.lineWidth = 5;
20 context.strokeStyle = "#535353";
21 context.stroke();
```

output



Notice how we go from defining our `CanvasGradient` object to specifying our `colorStops` to ultimately assigning our `CanvasGradient` object to the `fillStyle` property.

Our rectangle is filled with the linear gradient that we specified earlier. Before we move on to the next section, let's look at our gradient code in greater detail to truly understand what is going on.

The first line is where we define our `CanvasGradient` object very cleverly called `gradient`:

```
var gradient = context.createLinearGradient(75, 0, 325, 0);
```

The position of our virtual line matches the size of the rectangle we are trying to paint. Because our virtual line isn't going to be rotated, I left the Y value for both of the points to just be 0. The size of a gradient is infinite after all, so every Y position (in our unrotated virtual straight line) will have the same gradient value applied. Anyway, the X value maps to the starting and ending points of our rectangle, and it is important to get this value correct. If these

values are off, our color stops will be placed at points you probably don't want. Speaking of color stops, gaze your eyes to the next chunk of code:

```
// our three color stops
gradient.addColorStop(0.1, "#DDDDDD");
gradient.addColorStop(0.75,"#2D7BEC");
gradient.addColorStop(0.9, "#31FFFF");
```

We call the `addColorStop` method on `gradient`. The `addColorStop` method takes two arguments. The first argument specifies where the color stop will be located. You specify the location in terms of a decimal value between 0 and 1 with 0 being the start of our virtual line and 1 being the end of our virtual line. The second argument takes a color value.

You put all of this together, we specify our three color stops at at the 0.1, 0.75, and 0.9 locations with a color value of #DDDDDD, #2D7BEC, and #31FFFF respectively.

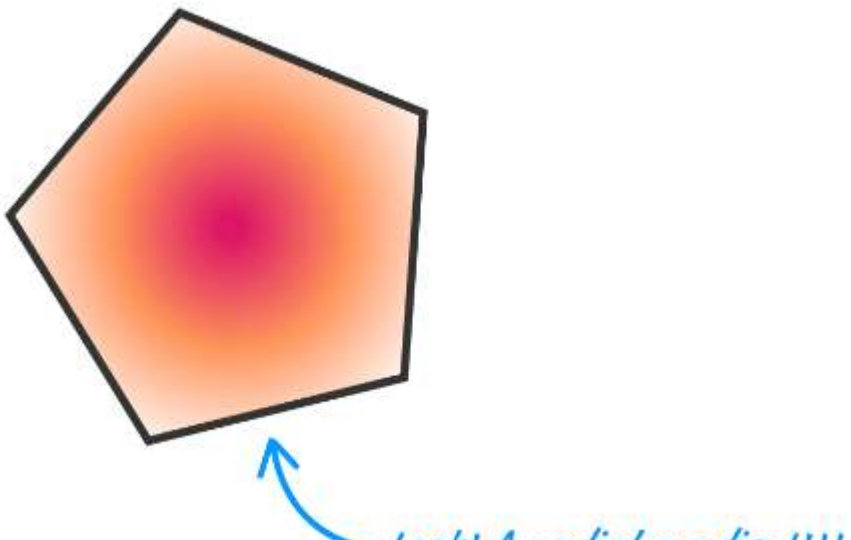
All that remains is to now assign our `CanvasGradient` object to whatever any of the `canvas` properties that deal with color. The lucky winner in our example is `fillStyle`...again:

```
context.fillStyle = gradient;
```

This line and the subsequent call to the `fill` method ensure our linear gradient gets applied to our rectangle!

The Radial Gradient

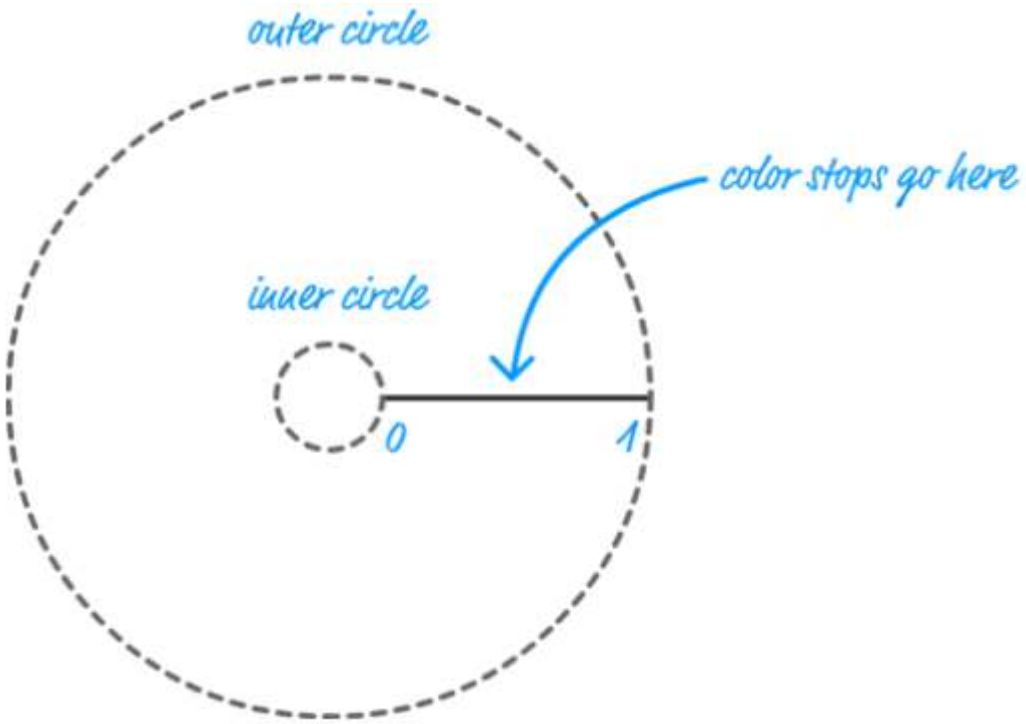
The next (and last!) type of gradient we will look at is the radial gradient:



Look! A radial gradient!!!

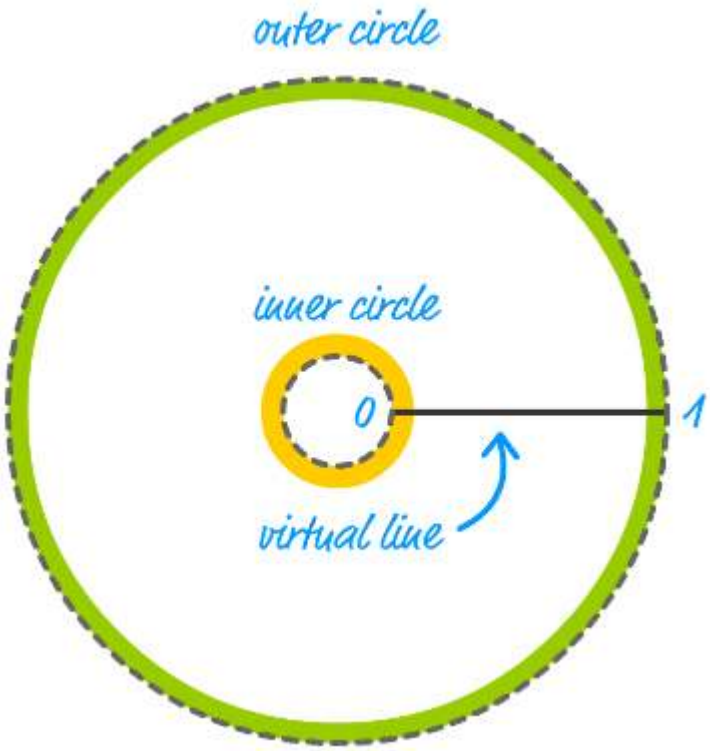
Radial gradients allow you to define a gradient that is all round and oval shaped. That sounds simple, but the way you visualize and actually use them is a whole another story. We will start at the very beginning and gradually ratchet up the complexity until we fully understand it.

With linear gradients, we had a virtual straight line that we specified our color stops on. That was easy to quickly understand. With radial gradients, we still have a virtual straight line. The difference is that this virtual straight line is bounded by two virtual circles. I know that last sentence makes no sense right now, so take a look at the following diagram:



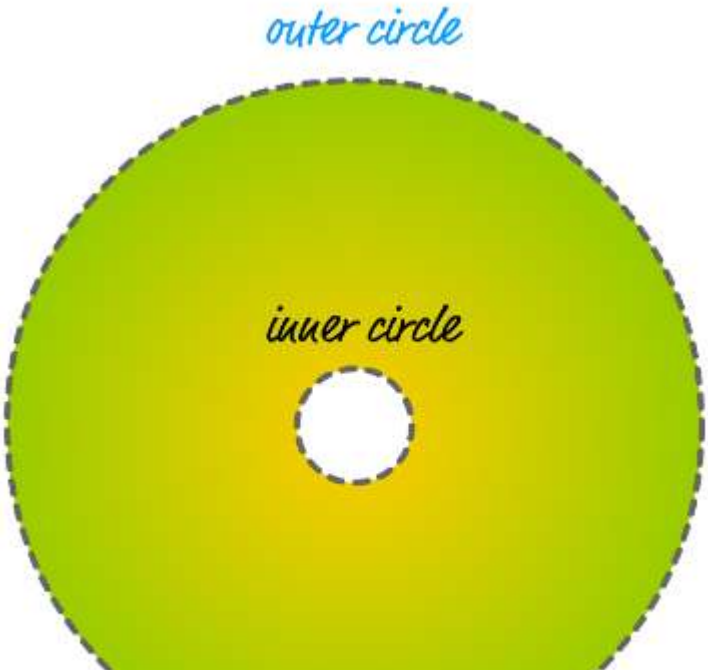
What we have here is a very literal representation of what a radial gradient looks like with no colors specified. We have an inner circle with a position and radius. We have an outer circle with a position and radius. Let's assume both of these circles are centered at the same location. If we drew a straight line from the center points of these two circles to the outer circle's edge, we have our virtual straight line that determines where our color stops will go. So far so good, right?

Let's throw some colors into the mix. At the 0 end of our virtual line, we are going to specify a yellow color. At the 1 end of our virtual line, we are going to specify a green color. Without doing any gradienting, given the two virtual circles we have, the two colors will look as follows:

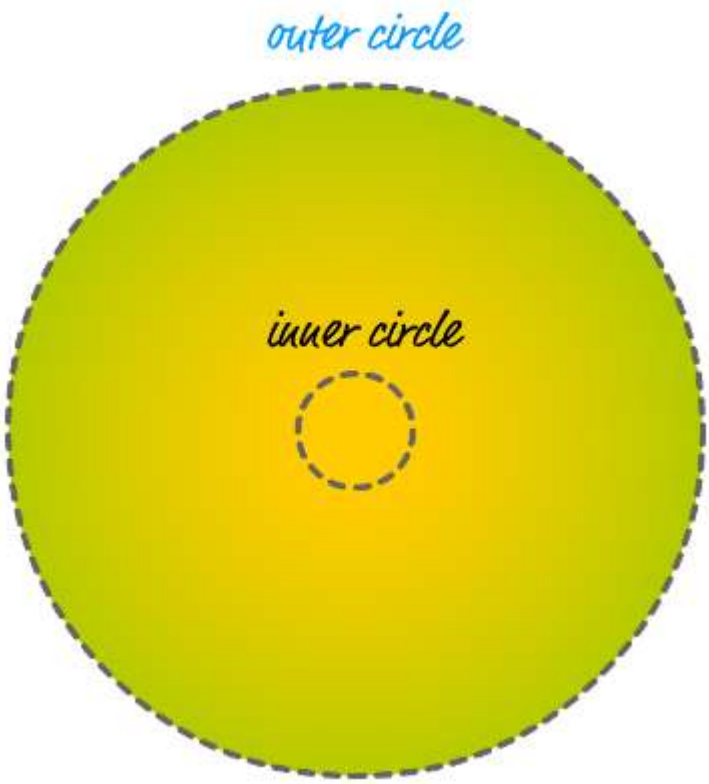


The way the colors move in a radial gradient is interesting and a little confusing at first. The easiest way to make sense of everything is to start by figuring out the coloring behavior for the inner circle and the outer circle.

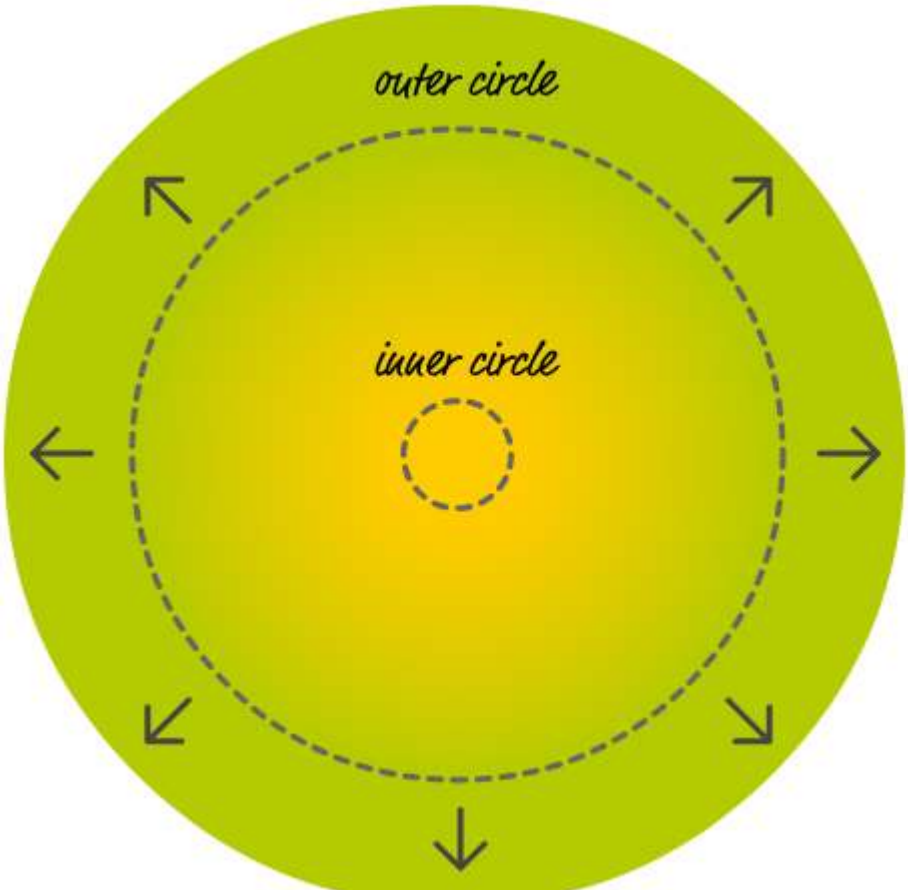
The inner circle is affected by the first color stop. In our case, that is the yellow color stop at the 0 position. The outer circle is affected by the last color stop. That is the green color at the 1 position. Between the two color stops, the colors blend just like they had in the linear case. The yellow will slowly make way to the green:



Inside the inner circle, the color will be the same as whatever the first color stop was. That means, the inside of our circle is going to be yellow:



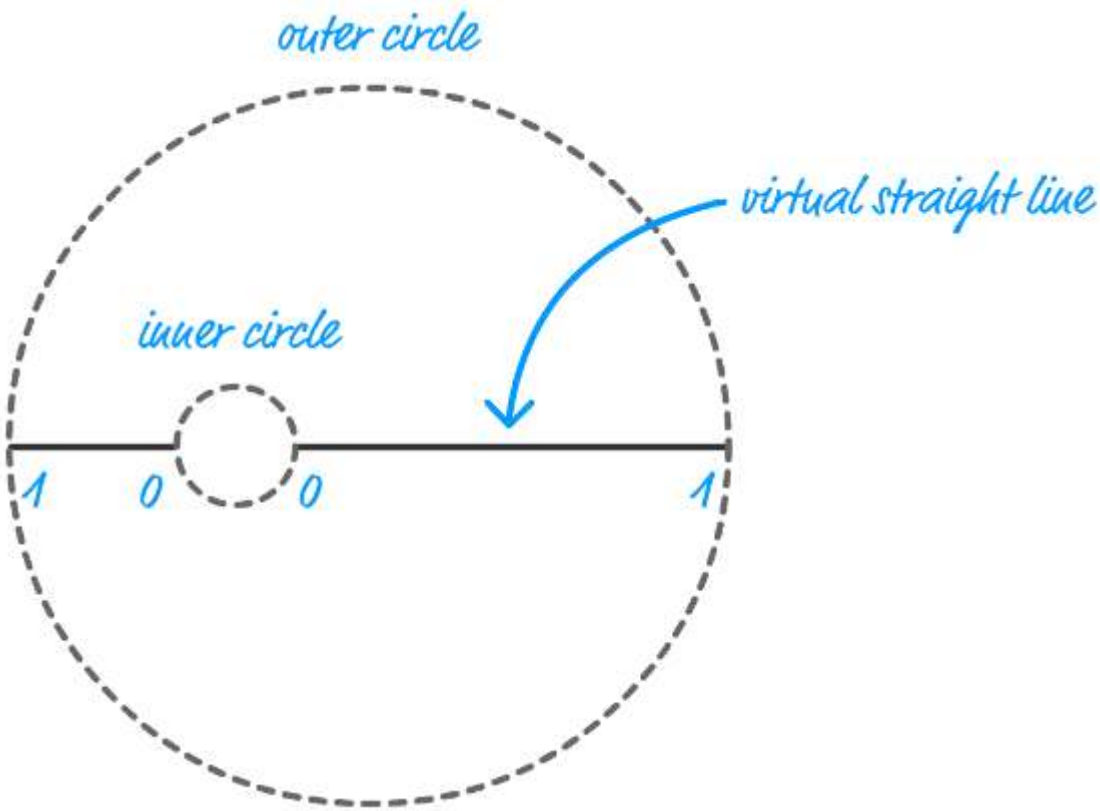
Outside of the outer circle, everything is going to be the same color as the last color stop. That means, everything is going to be green:



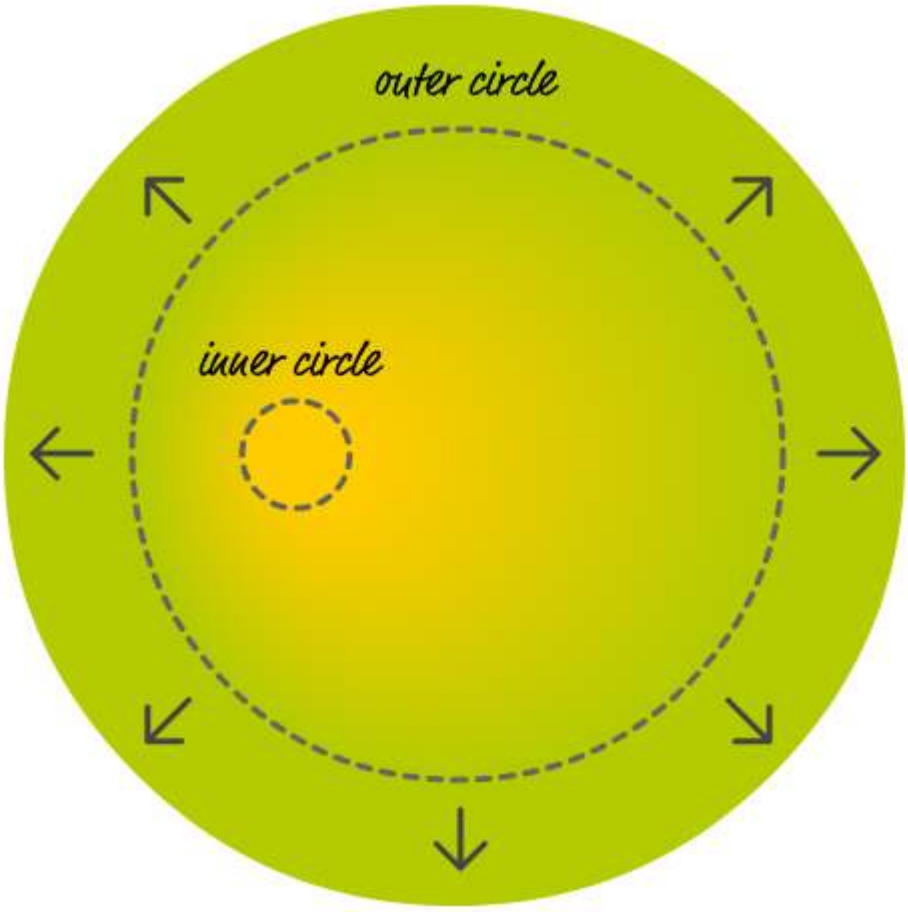
This green will go on forever in all directions. Any additional colors you specify only affect how the gradient behaves within the virtual straight line. The inside of the inner circle and the outside of the outer circle will always be a solid color that matches the color stop they are closest to.

> Ok. Pause for a moment. Did you know that we just fully explained how one variation of a radial gradient works?!!

You think we are done with this, right? Not yet! What we looked as is the ideal case where both the inner circle and the outer circle share the same center point. You can define a radial gradient where the inner and outer circles have different center points:



The behavior that we've seen so far is entirely identical even in this situation. The only difference is that your colors will look condensed on one side and more spread out on the other. Using the same yellow and green from before, the following is what you will see when the inner circle's center point is further left than our outer circle's center point:



There are a few more quirks beyond the big ones we've seen so far, but enumerating all of those will take too much time. The hard part was learning the basics that you just saw, so let's next look at how to implement a radial gradient.

To create a radial gradient, you are going to use the `createRadialGradient` method. This method takes six arguments:

```
context.createRadialGradient(x_i, y_i, r_i, x_o, y_o, r_o);
```

The first three arguments stand for the inner circle's x position, y position, and radius. The last three arguments stand for the outer circle's x position, y position, and radius. The `createRadialGradient` method returns a `CanvasGradient` object that you can then add color stops to and assign to a property like `fillStyle` or `strokeStyle`.

Here is us applying our yellow and green colored radial gradient to our rectangle:

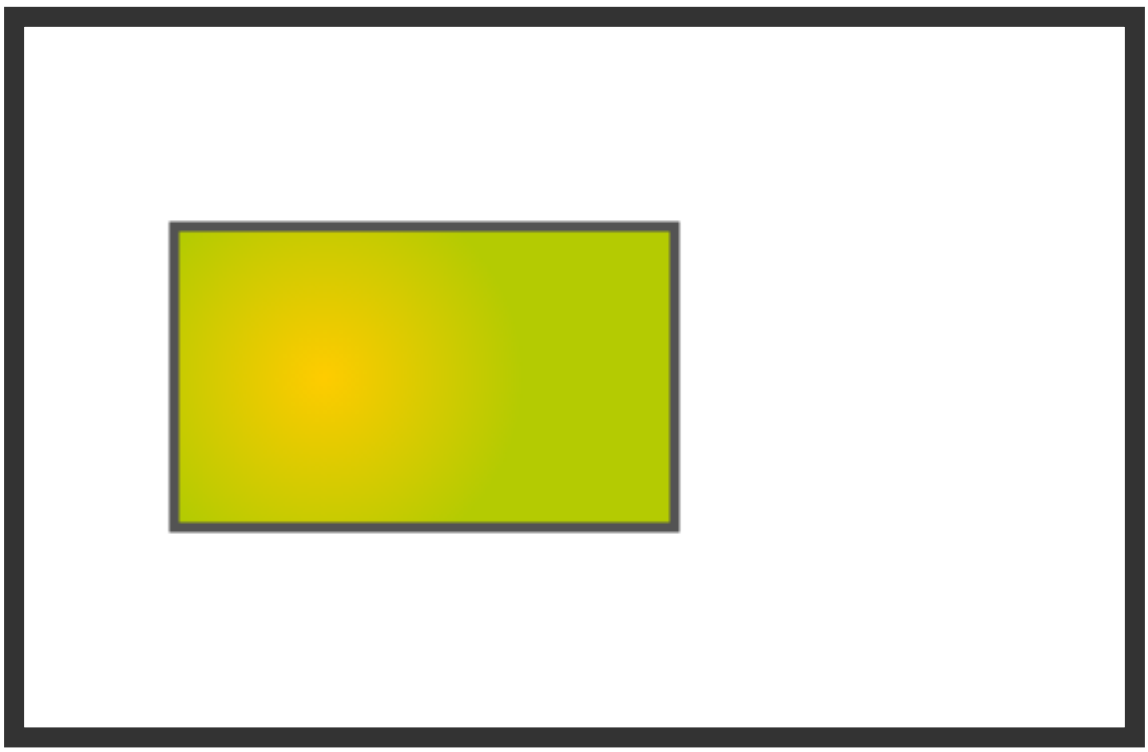
HTML JavaScript

HTML JavaScript

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

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

// our radial gradient!
var gradient = context.createRadialGradient(150, 175, 0, 150, 175, 100);

// our two color stops
gradient.addColorStop(0, "#FFCC00");
gradient.addColorStop(1, "#B4CB02");

// assigning the gradient
context.fillStyle = gradient;
context.fill();

context.lineWidth = 5;
context.strokeStyle = "#535353";
context.stroke();
```

javascript

output



Once you understand the basics of how to work with radial gradients, the only tricky part is positioning the center points at the right location with regards to the thing you are painting. You can see where we positioned our inner and outer circles with respect to the rectangle - both in the code as well as visually.