Drawing Things

WE'LL COVER THE FOLLOWING

In the previous section, we basically defined our `canvas` element in markup. While important, what we've just done is a very VERY minor part of working with the `canvas` overall. The real work is about to happen in this section when we write the JavaScript that interacts with the `canvas` element to get pixels to show up on the screen.

Adding the Script Tags

Before we can write JavaScript, we need a `script` tag that will house the JavaScript we write. You can use an external JavaScript file or keep all of the code in the same document. It will look something as follows:

HTML

html

```html
<!DOCTYPE html>
<html>

<head>
  <title>Simple Example</title>
  <style>
    canvas {
      border: #333 10px solid;
    }
  </style>
</head>

<body>
  <canvas id="myCanvas" width="550px" height="350px"></canvas>
```

```

    <script>
    </script>

</body>

</html>
```

With this minor addition, you are now set to write some JavaScript that will interact with our `canvas` element!

Accessing our Canvas Element

When it comes to working with the `canvas`, the first line of JavaScript you will almost always write will involve getting a reference to the `canvas` element in your HTML. This will allow you to start doing all sorts of canvas-ey things using JavaScript.

Now to get the reference, inside your `script` tag, go ahead and add the following line:

```
var canvas = document.querySelector("#myCanvas");
```

All we are doing here is initializing our `canvas` variable to our **myCanvas** element we defined in HTML earlier. We get a reference to the **myCanvas** element by relying on the querySelector function. This function is the cooler way of finding elements in HTML compared to the older `getElementById` and `getElementsByClass` functions you might be familiar with.

Getting the Rendering Context

Here is some useful trivia. Our canvas element has two modes of operation. One mode is designed for drawing things in 2D, and this is the mode we care about for now. The other mode is all about supporting drawing in 3D. These modes are more formally known as **rendering contexts**. To be able to draw to our `canvas` element, we need to first specify the rendering context we want to use. That is done by calling the `getContext` method on our `canvas` object and passing in the argument for the 2D rendering context we want:

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

The `context` variable now stores a reference to our `canvas` element's 2d rendering context and all the sweet drawing-related properties and functions that go along with it. To put all of this more simply, with a hook into the rendering context, you now have a pipeline through which you can issue commands to get pixels to show up inside your canvas!

Issuing Draw Commands

All that remains at this point is to use the various tools JavaScript provides to get things drawn on the screen. We've covered a lot of ground here already, so I won't overwhelm you with what all the various draw commands are and how to use them. Instead, I am going to first provide some code for you to copy/paste/write to see our `canvas` element in action.

Go ahead and add the following lines to your code:

HTML JavaScript

```javascript
var canvas = document.querySelector("#myCanvas");
```

javascript

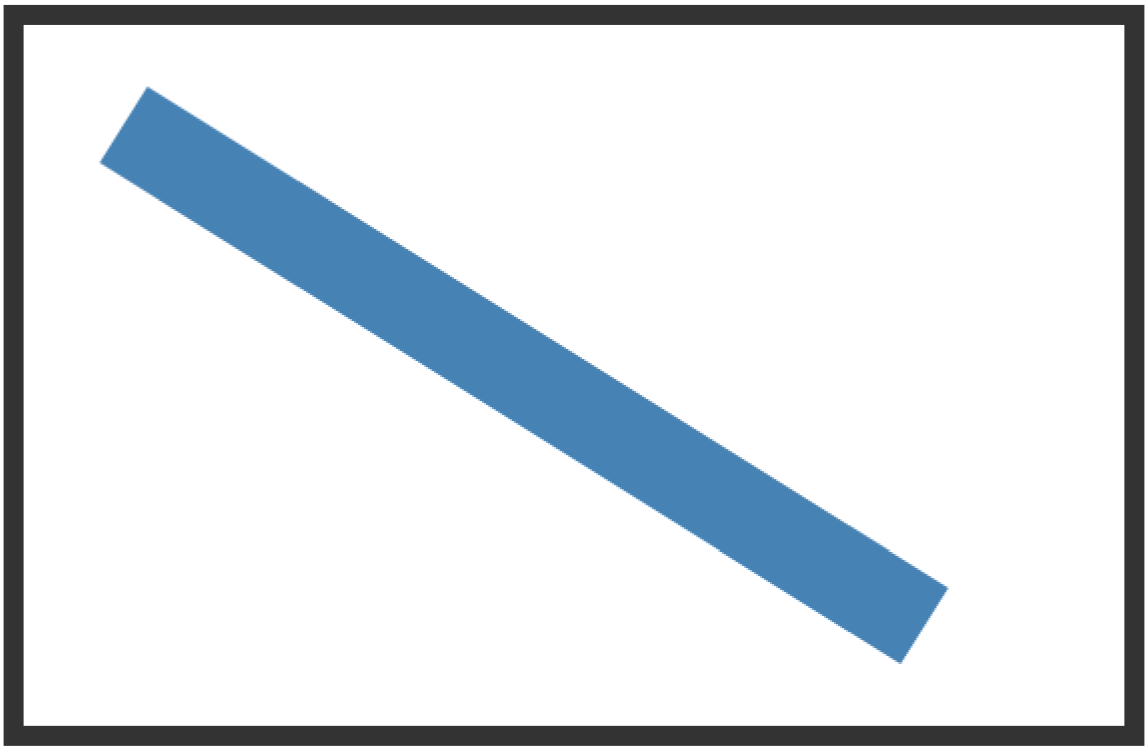
```
var context = canvas.getContext("2d");

// draw a diagonal line
context.moveTo(50, 50);
context.lineTo(450, 300);

// close the path
context.closePath();

// specify what our line looks like
context.lineWidth = 45;
context.strokeStyle = "steelblue";

// get the line drawn to the canvas
context.stroke();
```



If everything went well, this time you will actually see something drawn with a thick blue diagonal line showing up:

All of the code you just pasted is responsible for getting this line to show up. Take a few moments to look at what each line of code does. Don't worry if it doesn't fully make sense. For this introduction, just notice that we are doing what looks like moving a virtual pen around and specifying the starting and ending co-ordinates:

```
// draw a diagonal line
```

```
context.moveTo(50, 50);
context.lineTo(450, 300);
```

There is even some code that seems to specify the thickness and color of the line that gets drawn:

```
// specify what our line looks like
context.lineWidth = 45;
context.strokeStyle = "steelblue";
```

In future articles, we will dive much deeper into what each of these commands do and learn precisely how they work. We will also look at the supporting code that is very important…despite me ignoring them in this lightning-fast overview haha. For now, just be proud of the massive amount of progress you made!