

This Seems Simple, No?

Earlier, we looked at an example where we drew a simple four-sided shape. The `canvas` code for that looked as follows:

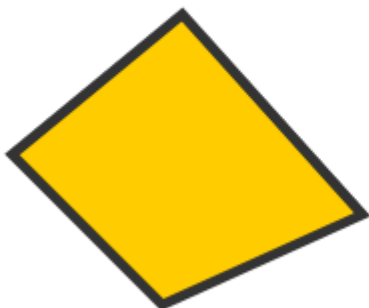
HTML

JavaScript

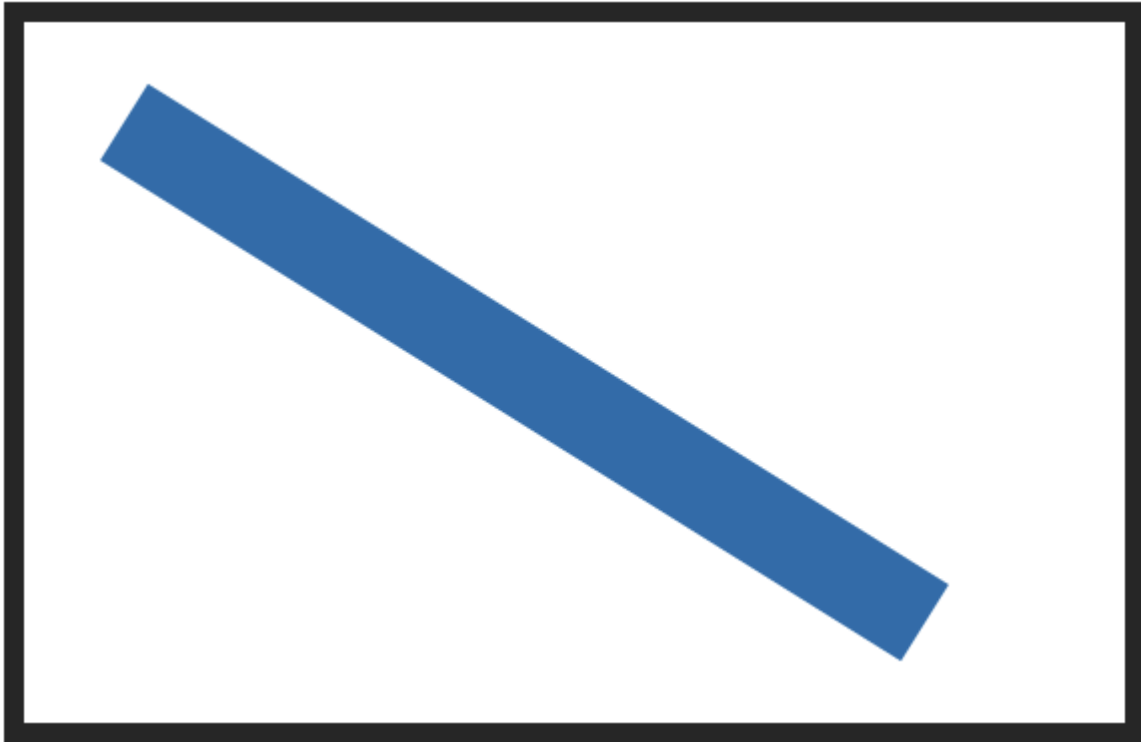
```
1 var canvas = document.querySelector("#myCanvas");
2 var context = canvas.getContext("2d");
3
4 context.moveTo(160, 130);
5 context.lineTo(75, 200);
6 context.lineTo(150, 275);
7 context.lineTo(250, 230);
8 context.closePath();
9
10 context.lineWidth = 5;
11 context.strokeStyle = "#333333";
12 context.fillStyle = "#FFCC00";
13
14 context.fill();
15 context.stroke();
```

javascript

output



Now, let's say that we want to add another shape to what we have here. The shape we want looks something like this:



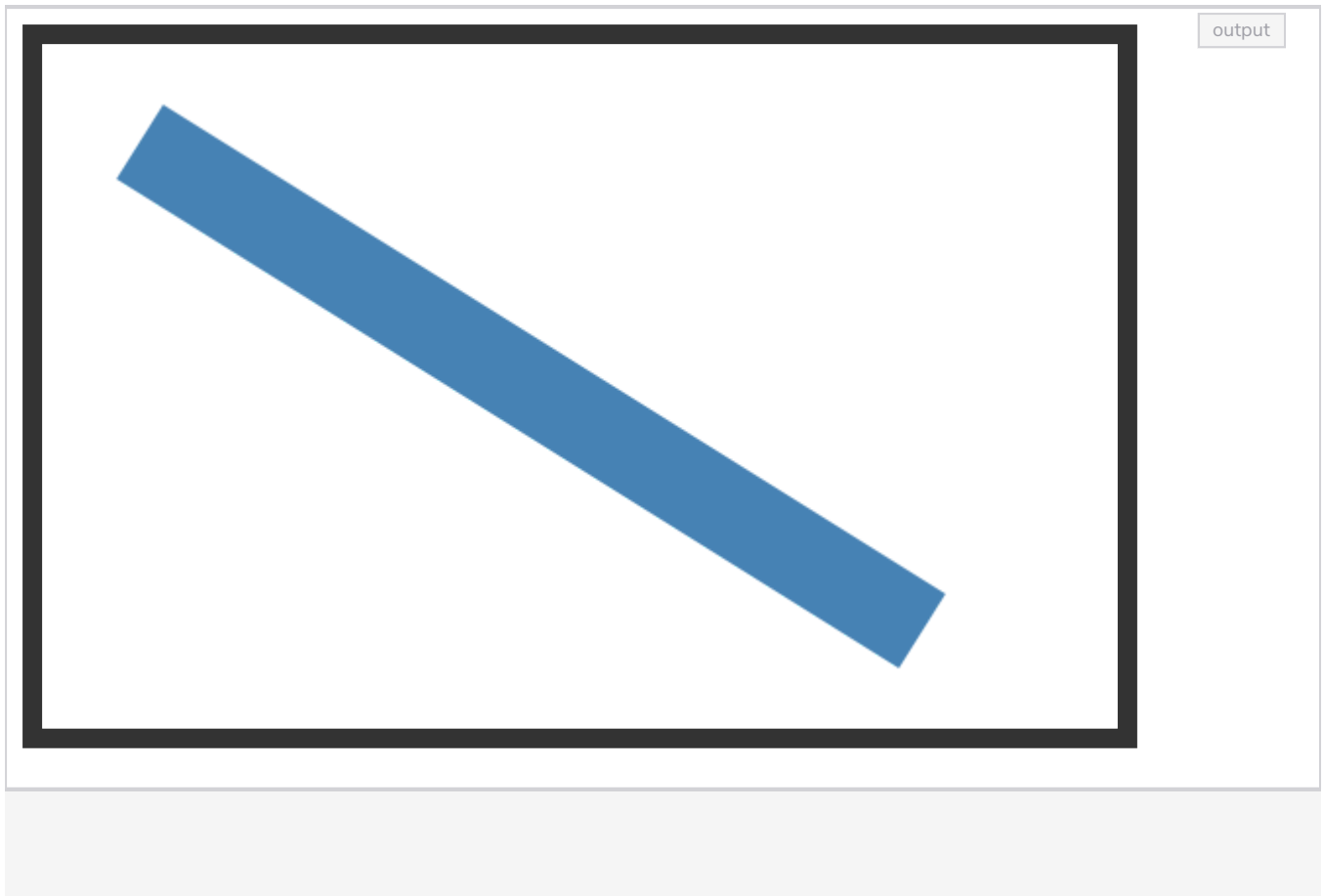
In code, it is represented as follows:

HTML

JavaScript

```
1 var canvas = document.querySelector("#myCanvas");
2 var context = canvas.getContext("2d");
3
4 context.moveTo(50, 50);
5 context.lineTo(450, 300);
6 context.closePath();
7
8 context.lineWidth = 45;
9 context.strokeStyle = "steelblue";
10
11 context.fill();
12 context.stroke();
13
```

javascript



What do you think we should do to combine these shapes? One thing we learned from earlier is that the `stroke()` and `fill()` methods act as the big red button you push to get things displayed on our screen. One reasonable attempt might be to keep the common `stroke()` and `fill()` methods from earlier and merge in the lines of code that represent our diagonal line.

On the surface, this seems like a reasonable thing to do. The first part of the code deals with the four-sided shape and what it looks like. The second (and highlighted) part deals with our diagonal line and what it looks like. In my mind, this seems like a solid solution!

When we preview this in our browser, this is what you will see:

HTML JavaScript

```
1 var canvas = document.querySelector("#myCanvas");
2 var context = canvas.getContext("2d");
3
4 context.moveTo(160, 130);
5 context.lineTo(75, 200);
6 context.lineTo(150, 275);
7 context.lineTo(250, 230);
8 context.closePath();
9
```

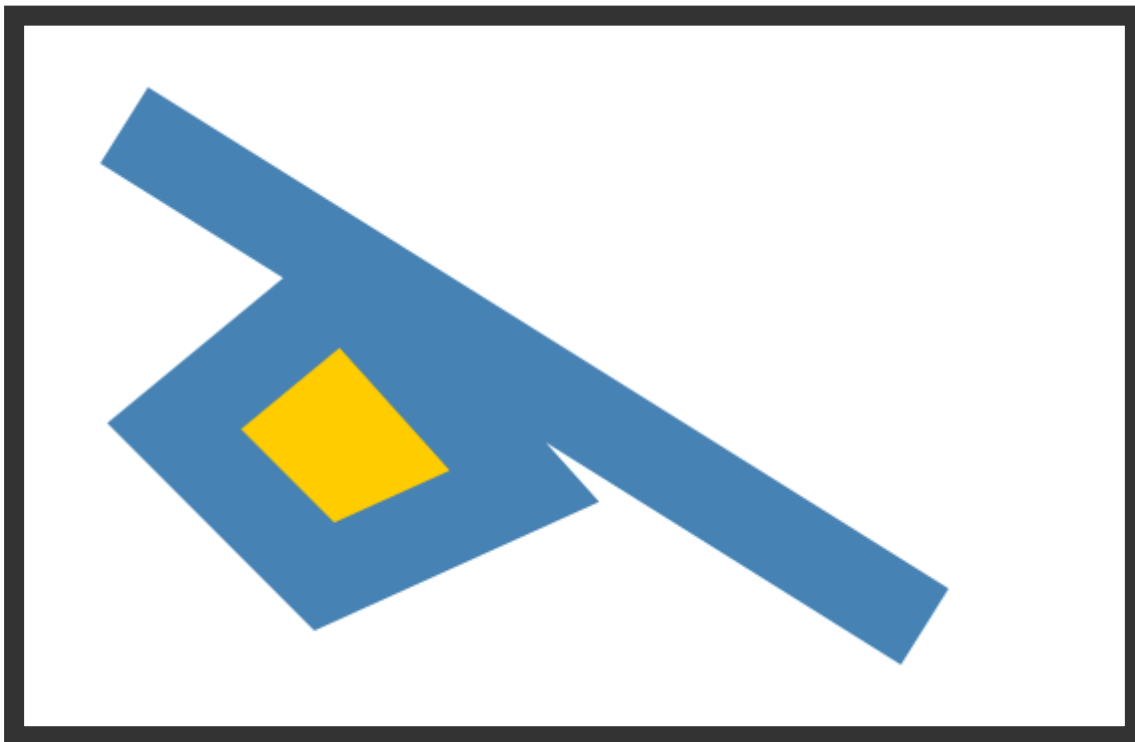
javascript

```

9
10 context.lineWidth = 5;
11 context.strokeStyle = "#333";
12 context.fillStyle = "#FFCC00";
13
14 context.moveTo(50, 50);
15 context.lineTo(450, 300);
16 context.closePath();
17
18 context.lineWidth = 45;
19 context.strokeStyle = "steelblue";
20
21 context.fill();
22 context.stroke();
23

```

output



Not quite what we had in mind, right? Ok, so it turns out merging the relevant lines of code under one single `stroke` and `fill` call didn't work out properly. What if we decide to keep the code for these shapes separate and have duplicate stroke/fill calls for each shape?

The code for that would look like this:

HTML JavaScript

```

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

```

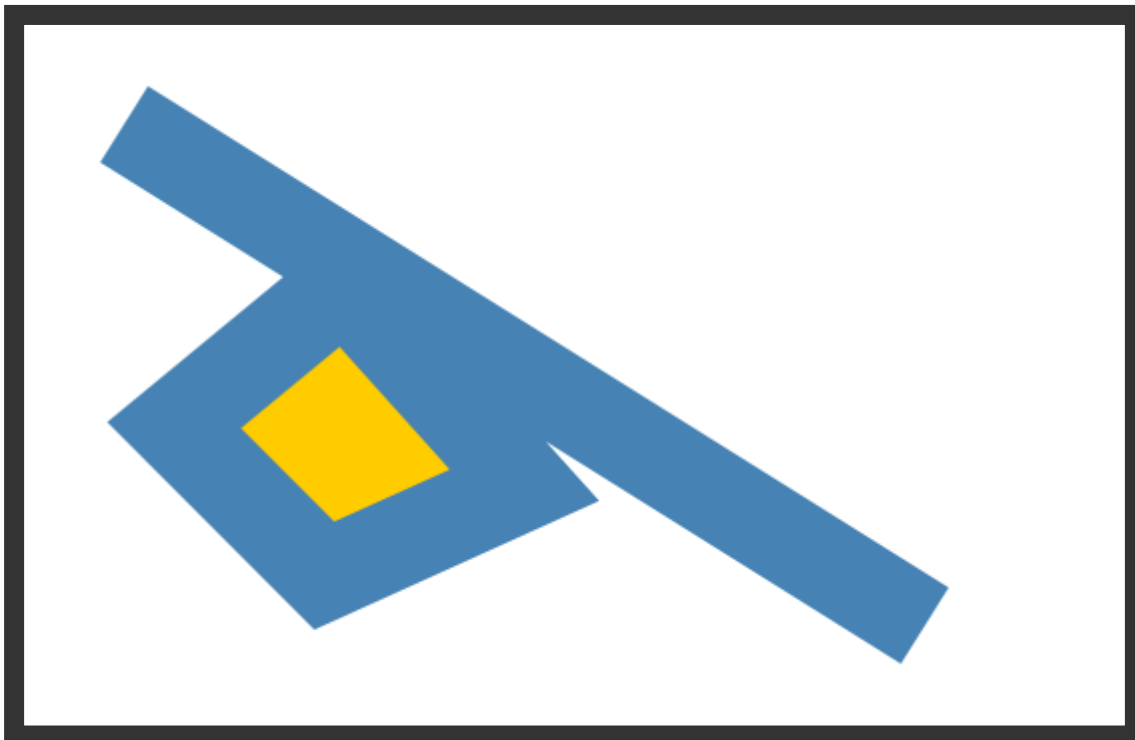
javascript

```

2 var context = canvas.getContext("2d");
3
4 // first shape
5 context.moveTo(160, 130);
6 context.lineTo(75, 200);
7 context.lineTo(150, 275);
8 context.lineTo(250, 230);
9 context.closePath();
10
11 context.lineWidth = 5;
12 context.strokeStyle = "#333";
13 context.fillStyle = "#FFCC00";
14
15 context.fill();
16 context.stroke();
17
18 // second shape
19 context.moveTo(50, 50);
20 context.lineTo(450, 300);
21 context.closePath();
22
23 context.lineWidth = 45;
24 context.strokeStyle = "steelblue";
25
26 context.stroke();
27

```

output



If you try this arrangement and preview in your browser, what do you see?
It's exactly the same thing as the weird jumble of shapes you saw earlier.
 What do you think is going on? Let's figure this out in the next section.

What do you think is going on? Let's figure this out in the next section