

Drawing Triangles

First, make sure you have a `canvas` element defined in your HTML page, and give it an `id` of **myCanvas**. To help you out with this, here is what my HTML looks like:

```
<!DOCTYPE html>
<html>
<head>
  <title>Triangle Canvas Example</title>
</head>
<body>
  <canvas id="myCanvas" width="500" height="500"></canvas>

  <script>

  </script>

</body>
</html>
```

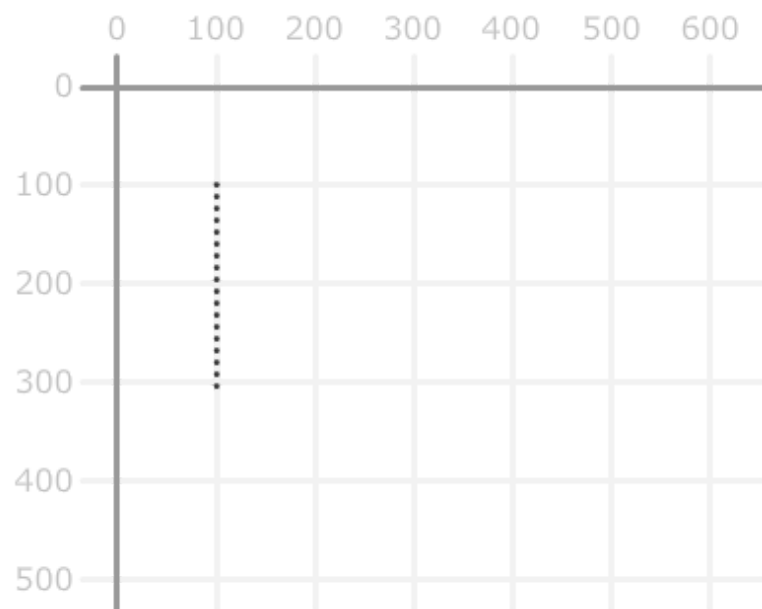
There isn't much going on here except for the `canvas` element whose `id` value is **myCanvas** with a `width` and `height` of 500 pixels. It is inside this canvas element we will draw our triangle. Now that we got this boring stuff out of the way...

The way you draw a triangle is by putting into code the following steps:

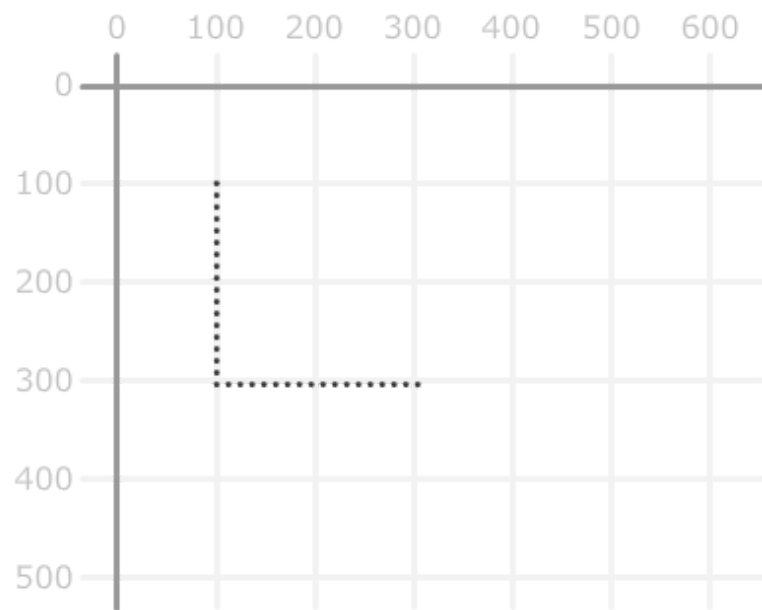
1. Declare your intent to draw lines so that the canvas knows what to expect
2. Move your virtual pen to to the x and y co-ordinate where you wish to start drawing the triangle
3. With your virtual pen at the starting point, use the `lineTo` method to draw lines between two points.
4. Specify the fill color, line color / thickness, etc. to adjust how your triangle looks



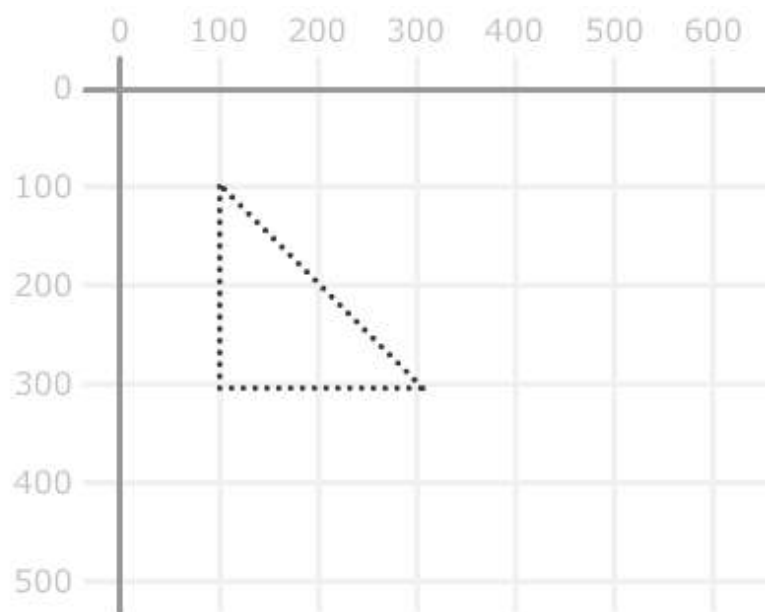
1 of 6



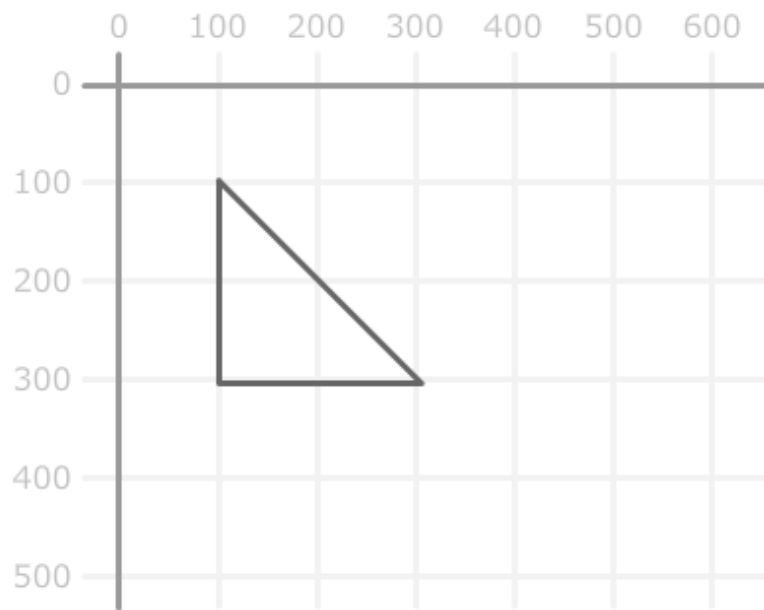
2 of 6



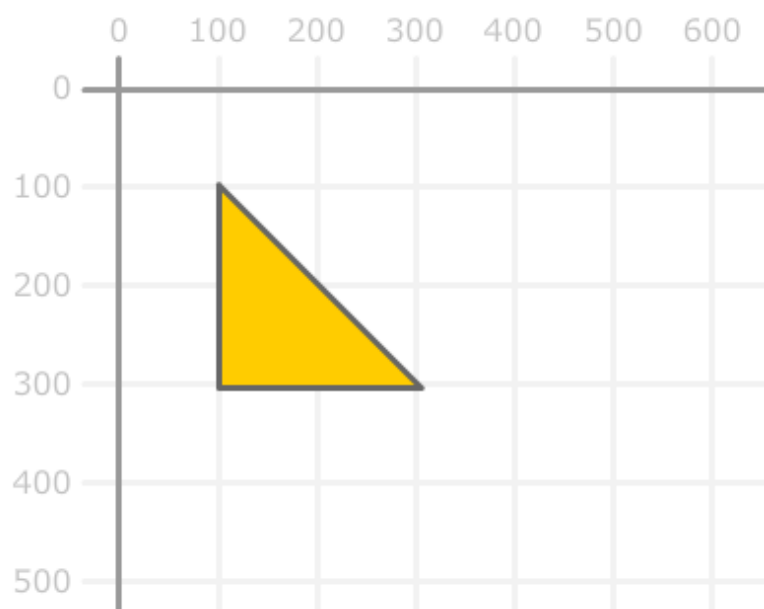
3 of 6



4 of 6



5 of 6



6 of 6



These steps are deliberately pretty hand-wavy to not overwhelm you at this point. The overwhelming will take place next, so let's look at the code for

drawing a simple rectangle first. We will weave these four steps in as part of explaining how the code works.


Inside the `script` tag, add the following lines of code:

HTML

JavaScript

```
1 var canvasElement = document.querySelector("#myCanvas");
2 var context = canvasElement.getContext("2d");
3
4 // the triangle
5 context.beginPath();
6 context.moveTo(100, 100);
7 context.lineTo(100, 300);
8 context.lineTo(300, 300);
9 context.closePath();
10
11 // the outline
12 context.lineWidth = 10;
13 context.strokeStyle = '#666666';
14 context.stroke();
15
16 // the fill color
17 context.fillStyle = "#FFCC00";
18 context.fill();
19
```

output

A yellow triangle with a dark gray outline is displayed on a white background. The triangle is oriented with its right angle at the bottom-left corner. The top vertex is at approximately (100, 100), the bottom-left vertex is at (100, 300), and the bottom-right vertex is at (300, 300). The triangle is filled with a bright yellow color, and its edges are outlined with a thick, dark gray stroke.

You can now see a yellow triangle appear.

Let's look at how the lines of code you've written map to the triangle that you

see on the screen. Starting at the top...

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



These two lines are a mainstay at the top of almost every canvas-related code you will be dealing with. The first line gets a pointer to the `canvas` element in our HTML. The second line gets you access to the `canvas` element's context object that allows you to actually draw things into the canvas.

Now, we get to the interesting stuff:

```
// the triangle  
context.beginPath();  
context.moveTo(100, 100);  
context.lineTo(100, 300);  
context.lineTo(300, 300);  
context.closePath();
```



Because these lines are crucial to drawing our triangle, I'm going to slow down and dive into greater detail on what each line does.

The first line sets it all up:

```
context.beginPath();
```



The `beginPath` method signals to the canvas that you intend to draw a path. I know that is a very unsatisfying explanation, but I never said this line of code was deep and full of meaning. It will hopefully make more sense as we look at the rest of the lines :P

The next line defines where to start our path from:

```
context.moveTo(100, 100);
```



That is handled by the `moveTo` function which takes an x and y co-ordinate value. In our case, we are going to be starting the path from a horizontal and vertical position of **100**:

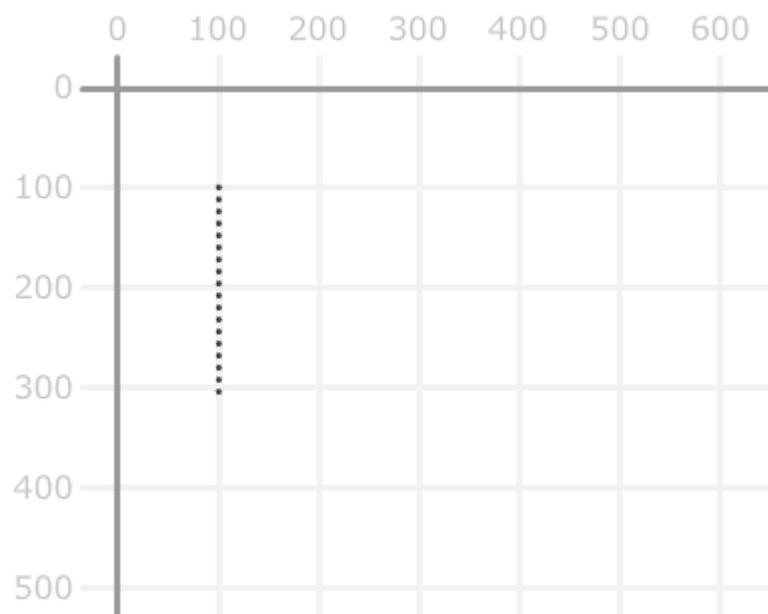


With the starting point set, it's time to start drawing the lines that make up our triangle:

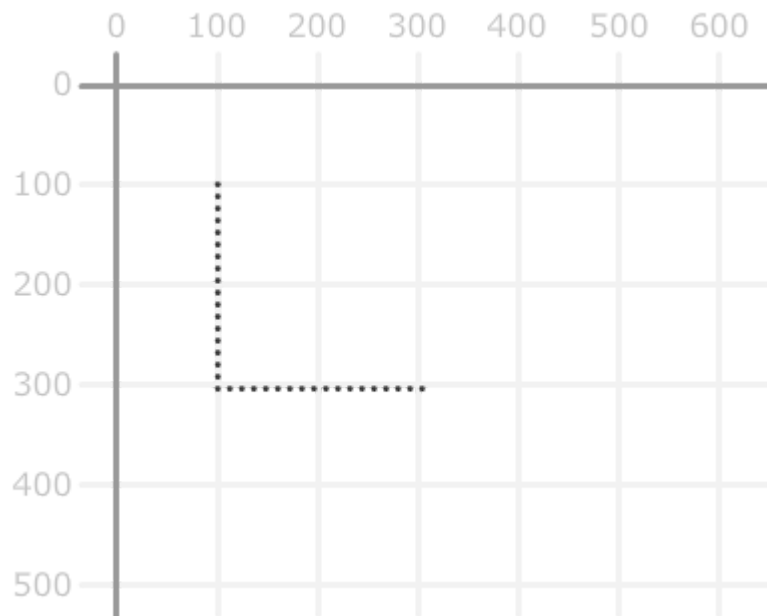
```
context.lineTo(100, 300);  
context.lineTo(300, 300);
```



The first `lineTo` function draws a line from our starting point of (100, 100) to (100, 300):



The second `lineTo` function picks up from where the pen currently is and draws a line from (100, 300) to (300, 300):



Right now, we have an L shape that isn't quite a triangle. To make this a triangle, we need to close this path by drawing a straight line from where we are back to the beginning. There are two ways to accomplish this:

1. One way is by specifying another `lineTo` function that looks like `context.lineTo(100, 100)`. This will draw a line from your current (300, 300) position to (100, 100).
2. The other way that you see in our code is by calling `context.closePath()`. The `closePath()` method tells your pen to draw a line back to the starting point.

Regardless of which approach you take, the end result is that you will now have a triangle:



At this point, you may be wondering why our triangle looks dotted and ghostly. If you haven't been wondering that, take a few moments and wonder.

Now that you are done wondering, here is why I visualize the lines in the way that I have. The five lines of code you've seen so far don't actually help you to see the triangle. What you've basically done is draw something that is **entirely invisible**:

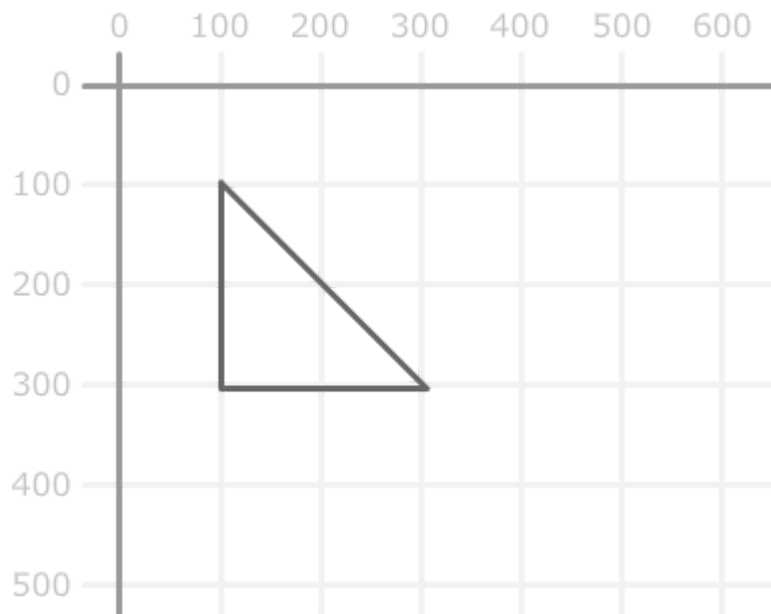


The next two chunks of code fix that up. The first chunk is where we define the triangle's outline:

```
// the outline
context.lineWidth = 10;
context.strokeStyle = '#666666';
context.stroke();
```



The `lineWidth` and `strokeStyle` properties specify the thickness and color of the line we want to draw. The actual drawing of the line is handled by calling the `stroke` method. At this point, you will see a triangle whose outlines are actually visible:

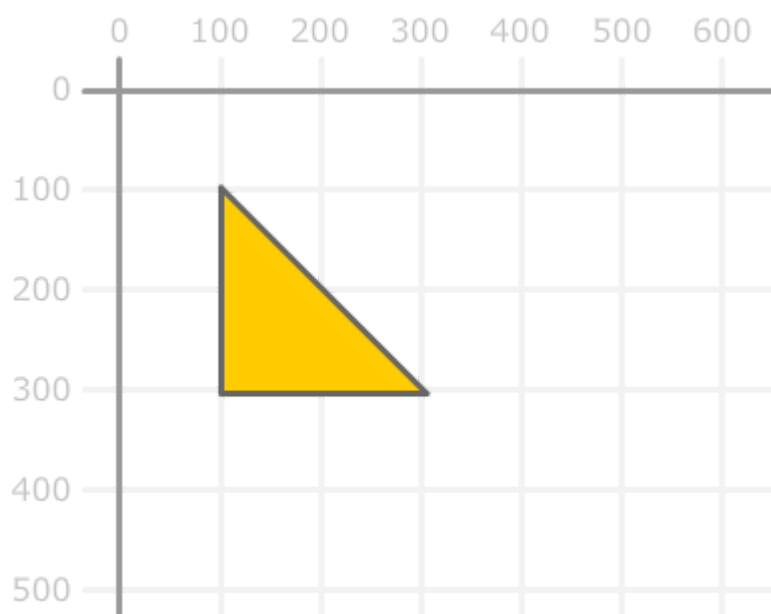


The second chunk of code fills our triangle up with a color:

```
// the fill color
context.fillStyle = "#FFCC00";
context.fill();
```



The `fillStyle` property allows you to define the color. The `fill` method tells your canvas to go ahead and fill up the insides of our closed path with that color. After this line of code has executed, you will end up with the triangle in its final form:



You now have a triangle that has an outline and a fill. It is, as some wise people say, complete.

