Michelle Dinan // Blog - Design & front end dev blog (http://blog.michelledinan.com)

HTML5 Canvas: Intro and drawing simple shapes

21st February 2012

I thought it was time to look into **HTML5 Canvas** as it was a big grey area in my knowledge and therefore something I should investigate, so started out slow with exploring how to draw shapes. In this post, I'll just cover drawing simple shapes, and move onto paths (http://blog.michelledinan.com/03/2012/html5-canvas-paths/) and curves (http://blog.michelledinan.com/03/2012/html5-canvas-bezier-curves/) in future posts.

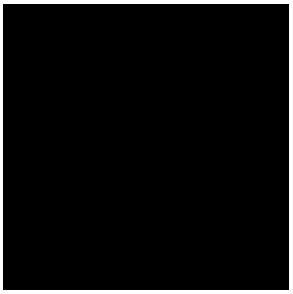
Quick links

Rectangles

Filled rectangle

Here is an example of probably the simplest shape we can create – a square:

Categories



View demo

(http://blog.michelledinan.com/demos/canvas/simple-shapes#filledRect)

The markup

To display a shape in a Canvas, I need to add a javascript portion to the `head` section of the HTML document which will control what will be displayed in the Canvas element, and then create the `canvas` element in the `body` of the page which will reference the javascript portion and pull in any settings set from there.

A filled rectangle can be drawn easily by using the `fillRect` context and set X and Y position parameters, as well as width and height.

View markup

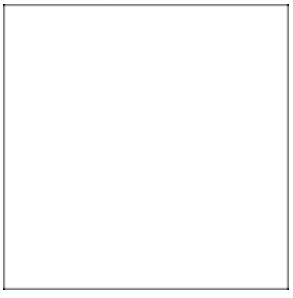
As you can see, there are **four numbers** as the parameters for `fillRect`. These mean as follows:

1. Horizontal (X) position (from left)
2. Vertical (Y) position (from top)
3. Width value
4. Height value

In my example, I have set the X and Y positions to **0** so the shape is butted up to the top left corner of the Canvas, and I have set both the width and height to **150** (px) to create the square shape.

Outlined rectangle

As well as a filled rectangle, I can also create one which just has an outline:



View demo

(http://blog.michelledinan.com/demos/canvas/simple-shapes#outlinedRect)

The markup

View markup

The markup is exactly the same, apart from that `fillRect` is replaced with `strokeRect` in the javascript portion of the markup. Don't forget to ensure

web based projects (http://blog.michelledinan.com/category/work/)
- Branding List of posts relating to branding work (http://blog.michelledinan.com/category/work/branding/)
- Illustration List of posts relating to vector illustration (http://blog.michelledinan.com/category/work/illustration/)

- Photoshop List of posts relating to Adobe Photoshop (http://blog.michelledinan.com/category/photoshop/)
- Search List of posts relating to web search (http://blog.michelledinan.com/category/search/)
- Site news List of posts relating to the blog (http://blog.michelledinan.com/category/site-news/)

Author

Michelle Dinan - smiley designer and front end developer from Bournemouth, Dorset.

- Bio (http://blog.michelledinan.com/biography/)
- Facebook page (https://www.facebook.com/MsMichelleDinan)

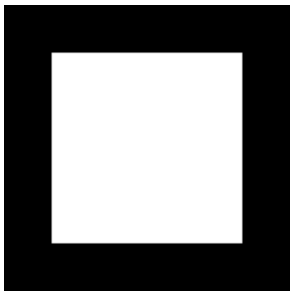
Archives

- May 2013 (http://blog.michelledinan.com/2013/05/)
- February 2013 (http://blog.michelledinan.com/2013/02/)
- January 2013 (http://blog.michelledinan.com

that the IDs relating to the Canvas also match up, or it'll remain blank. All parameters remain the same to create the same sized and positioned shape. By default, a 1px line is drawn around the shape; we'll look at styling strokes and fills (http://blog.michelledinan.com/04/2012/html5-canvas-styles/) in a future post.

Clearing rectangle

A rectangle can also be used as an eraser of sorts. When the `clearRect` context is used, anything that is *underneath* the clearing rectangle gets erased. In my example below, I've placed a smaller clearing rectangle on top of a larger black rectangle, and the background colour shows through where the clearing rectangle was placed. To see how other shapes and lines can be 'erased', check out my post on erasing on an HTML5 Canvas (http://blog.michelledinan.com/03/2012/html5-canvas-erasing-shapes/).



View demo

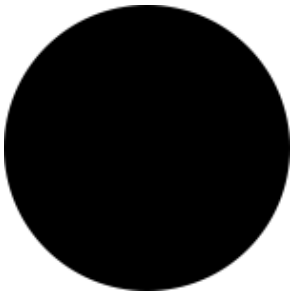
(http://blog.michelledinan.com/demos/canvas/simple-shapes#clearingRect)

The markup

View markup

Circles

Full circle



View demo

(http://blog.michelledinan.com/demos/canvas/simple-shapes#fullCircle)

The markup

Circles use the `arc` context, and variables are added as parameters to it.

View markup

Setting up the variables

- /2013/01/)
- November 2012 (http://blog.michelledinan.com/2012/11/)
- October 2012 (http://blog.michelledinan.com/2012/10/)
- September 2012 (http://blog.michelledinan.com/2012/09/)
- August 2012 (http://blog.michelledinan.com/2012/08/)
- July 2012 (http://blog.michelledinan.com/2012/07/)
- May 2012 (http://blog.michelledinan.com/2012/05/)
- April 2012 (http://blog.michelledinan.com/2012/04/)
- March 2012 (http://blog.michelledinan.com/2012/03/)
- February 2012 (http://blog.michelledinan.com/2012/02/)
- January 2012 (http://blog.michelledinan.com/2012/01/)
- December 2011 (http://blog.michelledinan.com/2011/12/)

Blogroll

- Andria Dinan (http://andriadinan.blogspot.co.uk/)
- Drop That Muffin (http://dropthatmuffin.com)
- Scott Reed (http://scottreed.eu/)

As you can see, before any drawing is carried out, I have set the variables for:

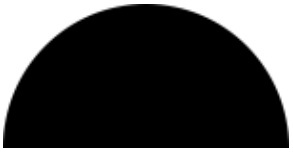
1. The X and Y position of the shape (the default origin for the circle will be top left of the Canvas, so these have to be set so the shape is actually visible)
2. The radius of the circle
3. The start angle for the circle. **0** means that the circle starts at the **3 O'clock position**.
4. The end angle for the circle. If we want to specify other angles on a circle, we use multiples of **PI**, so we have to do a small calculation. **PI*1 (or just PI) would be at the 9 O'clock position**, **PI*0.5 would be at the 6 O'clock position**, **PI*1.5 would be at the 12 O'clock position** and **PI*2 would be back at the 3 O'clock position** again, so to get a full circle, we need to use this one.
5. Whether the shape is drawn clockwise or anticlockwise. Mine is drawn clockwise, by setting `antiClockwise` to **false**.

Drawing the shape

The shape then has to be drawn on our Canvas. Before when I was drawing rectangles, there was a specialised context set up for them already which would create a filled or outlined rectangle depending on if `fillRect` or `strokeRect` was used. With other shapes however, I will have to draw them in a more manual way.

1. The first thing I need to do is to indicate that I want to start drawing a shape path. I will do this by adding the context `createPath` (it takes no parameters).
2. I then want to draw the circle. This is achieved by using the `arc` context, then calling in all the variables previously set as parameters **(X, Y, radius, startAngle, endAngle, antiClockwise)**.
3. Then I close the path by using the `closePath` context (this also takes no parameters).
4. The last thing I need to do is decide how I want the circle shape to appear, as it's just a transparent path at the moment. I have just added a context of **fill** to very simply fill the shape in with the default fill colour. I'll look at more exciting ways of styling shapes (http://blog.michelledinan.com/04/2012/html5-canvas-styles/) in a future post.

Half circle



View demo

(http://blog.michelledinan.com/demos/canvas/simple-shapes#halfCircle)

The markup

View markup

To create the **half circle**, a 180 degree arc needs to be drawn instead of a

full 360, so I just need to change the **PI** values. My circle above starts at **0** (3 O'clock position as before) and draws 180 degrees around anti-clockwise to the 9 O'clock position (PI). The path is closed and filled.

Third of a circle

View demo

(http://blog.michelledinan.com/demos/canvas/simple-shapes#thirdCircle)

The markup

View markup

Javascript portion

```
    function draw() {
            // Third of a circle
            canvas = document.getElementById("circleThird");
            if (canvas.getContext) {
 5.                 var ctx = canvas.getContext("2d");
                    var x = 75;
                    var y = 75;
                    var radius = 75;
                    var startAngle = PI;
10.                 var endAngle = Math.PI*1.75;
                    var antiClockwise = false;

                    ctx.beginPath();
                    ctx.arc(x, y, radius, startAngle, endAngle, an
    tiClockwise);
15.                 ctx.lineTo(75, 75);
                    ctx.closePath();

                    ctx.fill();
            }
20. }
```

HTML portion

```
    <body onLoad="draw();">
            <canvas id="circleThird" width="150" height="75"></can
    vas>
    </body>
```

For my third of a circle, I have a `startAngle` of **PI**, and an `endAngle` of **PI*1.75**. This by itself does not make the shape we desire, as the two end points will join up together when a fill is added and there won't be any lines going to the centre of the circle to create that 'slice' shape. So, I shall add an extra context of `lineTo` with **75, 75** (these are X and Y values) set as the parameters after the arc. This will draw a line to that location (which is the

centre of the would-be circle shape) before the path is closed and filled.

Quarter circle

View demo

(http://blog.michelledinan.com/demos/canvas/simple-shapes#quarterCircle)

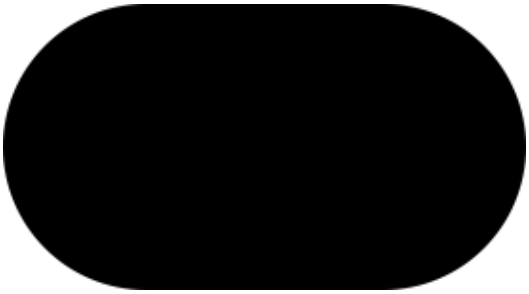
The markup

View markup

The quarter circle is created in exactly the same way as the third of a circle, but the `endAngle` is set to **PI*1.5** instead (12 O'clock position).

Other shapes

Pill shape



View demo

(http://blog.michelledinan.com/demos/canvas/simple-shapes#pill)

The markup

View markup

To create the pill shape, I will need two 180 degree arcs to be drawn. My pill shape example above starts with the first arc (on the left hand side); its start angle is **PI*0.5** (6 O'clock position) and draws 180 degrees around clockwise to the 12 O'clock position **(PI*1.5)**. A horizontal line is then drawn from there to the start of the next arc (right hand side), which runs 180 degrees clockwise through from **PI*1.5** (12 O'clock) back to **PI*0.5** (6 O'clock). The shape is then closed, so there is no need to draw another line connecting the two arcs again.