Chapter 17 Notes

**SVG** (Scalable Vector Graphics) - A document-markup dialect that focuses on shapes rather than text.

**Canvas** is a single DOM element that encapsulates a picture.

* An **SVG** picture the original description of the shapes is preserved so that they can be moved or resized at any time.
* A **Canvas** converts the shapes to pixels as soon as they are drawn and does not remember what these pixels represent.

Example of HTML document with SVG picture:

<p>Normal HTML here . </p> 
2 <svg xmlns—"http://www.w3. 
<circle r—" 59" cx " 50" cy " 50" fiLL '[red" /> 
<rect x "12@" y " 5" width— "90" height—" 90" 
stroke—"blue" fill "none" /> 
6 </svg> 
Normal HTML here. 

Canvas graphics can be drawn onto a <canvas> element. There are two supported graphic styles, "2d" and "webgl" for 3D graphics.

Create a context with *the* **getContext** method on the <canvas> DOM element.

<p>Before canvas . 
2 <canvas width— "120" height "60"></canvas> 
3 canvas . 
4 <script> 
let canvas document . querySeLectorC canvas" ) ; 
let context 
canvas . getContext( " 2d" ) ; 
context . fil I Style 
= "red" , 
context . fill RectC10, 10, lee, 50); 
9 </script> 
Before canvas _ 
After canvas. 

**Lines and Surfaces**

A shape can be filled in the canvas interface which means that its area is given a certain color or pattern

A shape can be stroked, which means a line is drawn along its edge.

**fillRect**

* Fills a rectangle with x and y coordinates

**strokeRect**

* Draws the outline of the rectangle

\*\* neither method takes any further parameters. The color, thickness, etc, are not determined by an argument to the method but by the properties of the context object \*\*

**fillStyle**

* Controls the way shapes are filled

**strokeStyle**

* Works similar to fillstyle but determines the color used for a stroked line.

<canvas></canvas> 
2 <script> 
let cx document . querySelector( canvas . getContext( " 2d") ; 
cx . strokeStyle 
= '[blue" , 
cx. strokeRect(5, 5, 50, 50); 
cx. LineWidth 5 ; 
cx. strokeRect(135, 5, 50, 50); 
8 </script> 
VVhen no width or height attribute is specified, as in the example, a canvas 
element gets a default width of 300 pixels and height of 150 pixels. 

**Paths**

A Path is a sequence of lines. They are not values that can be stored and passed around. You have to make a sequence of method calls to describe its shape

<canvas></canvas> 
2 <script> 
let cx document . querySelector( canvas . getContext( " 2d") ; 
cx. beginPath() ; 
for (let y 10; y < 100; y += 10) { 
cx . moveTo(10, Y); 
cx . lineTo(90, Y); 
cx . stroke() ; 
10 </script> 
This example creates a path with a number of horizontal line segments and 
then strokes it using the stroke method. Each segment created with I ineTo 
starts at the path's current position. That position is usually the end of the last 
segment, unless moveTo was called. In that case, the next segment would start 
at the position passed to moveTo. 

**Curves**

The **quadraticCurveTo** method draws a curve to a given point. To determine the curvature of the line, the method is given a control point as well as a destination point.

<canvas></canvas> 
2 <script> 
let cx document . querySelector( canvas . getContext( " 2d") ; 
cx. beginPath() ; 
cx. moveTo(1Ø, 90); 
// control—C60, 10) 
cx. quadraticCurveTo(6Ø, 10, 90, 90) ; 
cx. LineTo(6Ø, 10); 
cx. cLosePath() ; 
cx . stroke() ; 
11 </script> 
We draw a quadratic curve from the left to the right, with (60,10) as control 
point, and then draw two line segments going through that control point and 
back to the start of the line. The result somewhat resembles a Star Trek 
insignia. You can see the effect of the control point: the lines leaving the lower 
corners start off in the direction of the control point and then curve toward 
their target. 

**bezierCurve**

The bezierCurveTo method draws a similar kind of curve. Instead of a single 
control point, this one has two one for each of the line's endpoints. Here is a 
similar sketch to illustrate the behavior of such a curve: 
<canvas></canvas> 
2 <script> 
let cx document . querySelector( canvas . getContext( " 2d") ; 
cx. beginPath() ; 
cx. moveTo(1Ø, 90); 
// 19) control 90) 
cx. bezierCurveToC10, 10, 90, 10, 50, 90); 
cx. LineTo(9Ø, 10); 
cx. LineTo(1Ø, 10); 
cx. closePath() ; 
cx . stroke() ; 
12 </script> 
The two control points specify the direction at both ends of the curve. The 
farther they are away from their corresponding point, the more the curve will 
"bulge" in that direction. 

**Arc**

Machine generated alternative text:
The arc method is a way to draw a line that curves along the edge of a circle. It 
takes a pair of coordinates for the arc's center, a radius, and then a start angle 
and end angle. 
Those last two parameters make it possible to draw only part of the circle. The 
angles are measured in radians, not degrees. This means a full circle has an 
angle of 231, or 2 * Math . PI, which is about 6.28. The angle starts counting at 
the point to the right of the circle's center and goes clockwise from there. You 
can use a start of o and an end bigger than 231 (say, 7) to draw a full circle. 
<canvas></canvas> 
2 <script> 
let cx document . querySelector( canvas . getContext( " 2d") ; 
cx. beginPath() ; 
// center=C50 , 50) radius=4Ø angle=0 to 7 
cx.arcC50, 5e, 4e, o, 7); 
// center=C150 , 50) radius=40 angle=Ø to lhT1 
cx.arcC150, 50, 4e, O, 0.5 * Math.P1); 
cx . stroke() ; 
10 </script> 
The resulting picture contains a line from the right of the full circle (first call to 
arc) to the right of the quarter-circle (second call). Like other path-drawing 
methods, a line drawn with arc is connected to the previous path segment. 
You can call moveTo or start a new path to avoid this. 