

HTML5 Training for Web Developers

HTML5 Canvas

Lesson 1, Activity 2: Getting Started with Canvas

The `canvas` element takes two attributes: `height` and `width`. Between the open and close tags, you can place fallback content for browsers that do not support canvas, like this:

```
<canvas id="my-canvas" width="500" height="500">Your browser doesn't support canvas.</canvas>
```

If you are using canvas to represent something that can be represented with an image as well, you could include an `img` element as your fallback content.

Context

The `<canvas>` tag only creates a drawing surface. To actually do any drawing, you need to use JavaScript. The first step is getting the `context`. Although eventually other contexts will be supported (including **3d**), the only currently supported context is **2d**.

```
var canvas=document.getElementById("my-canvas");
context = canvas.getContext("2d");
```

Since many browsers don't support canvas, they won't be able to get the context, so you should test for context support before beginning to draw:

```
var canvas=document.getElementById("my-canvas");
if (canvas.getContext) {
    context = canvas.getContext("2d");
    //draw here
}
```

Lesson 1, Activity 3: Drawing Lines

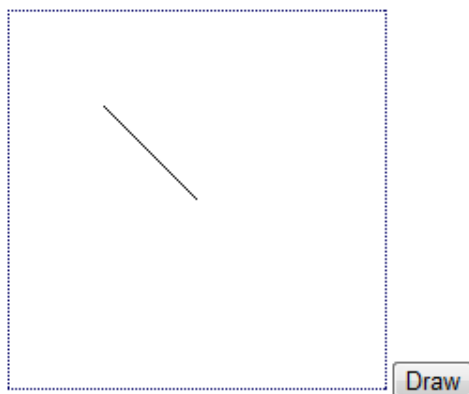
Let's start by drawing a simple line. Take a look at the following sample:

Code Sample:

<html5-canvas/Demos/path-simple.html>

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
<title>Path - Simple</title>
<link href="style.css" rel="stylesheet" type="text/css">
<script>
function drawPath() {
  var canvas=document.getElementById("my-canvas");
  if (canvas.getContext) {
    context = canvas.getContext("2d");
    context.beginPath();
    context.moveTo(50,50);
    context.lineTo(100,100);
    context.stroke();
  }
}
</script>
</head>
<body>
<canvas id="my-canvas" height="200" width="200">Your browser doesn't support canvas.</canvas>
<button onclick="drawPath();">Draw</button>
</body>
</html>
```

After clicking on the **Line** button, a line will get drawn as shown below:



Note that by default the `canvas` element has no border. We've used CSS to add a dotted border.

Let's look at the code:

1. `context.beginPath()` ; - tells canvas to forget any existing path being worked on and start a brand new path.
2. `context.moveTo(50,50)` ; - tells canvas to move to the x,y position of the `moveTo(x,y)` method. No sub-path is created.
3. `context.lineTo(100,100)` ; - tells canvas to create a sub-path from the current point to the x,y position of the `lineTo(x,y)` method. Note that this does not draw the sub-path. It just stores it in memory. Each call to `lineTo()` will create an additional sub-path.
4. `context.stroke()` ; - tells canvas to go ahead and draw the path, which is stored as a list of sub-paths making up one single path. The `stroke()` method is what gets your path on the canvas.

Multiple Sub-Paths

The demo below shows how to use multiple sub-paths to create a triangle:

Code Sample:

<html5-canvas/Demos/path-multiple-subpaths.html>

```

---- C O D E   O M I T T E D ----

if (canvas.getContext) {
    context = canvas.getContext("2d");
    context.beginPath();
    context.moveTo(50,50);
    context.lineTo(100,100);
    context.lineTo(100,50);
    context.lineTo(50,50);
    context.stroke();
}
---- C O D E   O M I T T E D ----

```

We can replace the final `lineTo()` call, which is used to close the triangle, with a call to `closePath()`, which attempts to create a sub-path from the current location to the starting point - the point at which the first sub-path started:

Code Sample:

<html5-canvas/Demos/path-closepath.html>

```

---- C O D E   O M I T T E D ----

if (canvas.getContext) {
    context = canvas.getContext("2d");
    context.beginPath();
    context.moveTo(50,50);
    context.lineTo(100,100);
    context.lineTo(100,50);
    context.closePath();
}

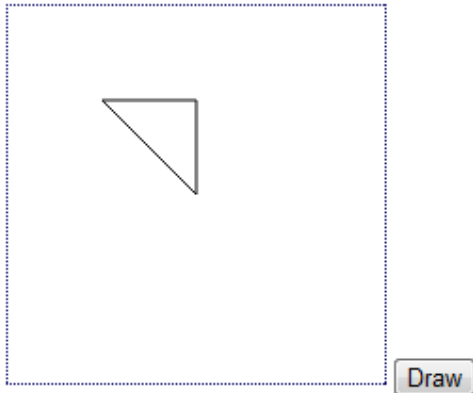
```

```

    context.stroke();
}
---- C O D E    O M I T T E D ----

```

Both this and the previous example render as follows:



Note that you can also hop around using `context.moveTo(x,y)` to create disconnected sub-paths:

Code Sample:

<html5-canvas/Demos/path-disconnected.html>

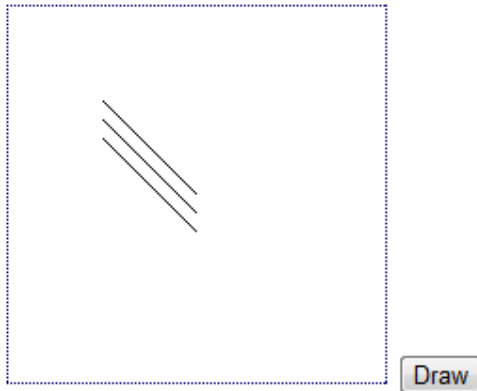
```

---- C O D E    O M I T T E D ----

if (canvas.getContext) {
    context = canvas.getContext("2d");
    context.beginPath();
    context.moveTo(50,50);
    context.lineTo(100,100);
    context.moveTo(50,60);
    context.lineTo(100,110);
    context.moveTo(50,70);
    context.lineTo(100,120);
    context.stroke();
}
---- C O D E    O M I T T E D ----

```

This will render as follows:



The Path Drawing Process

So, the process for drawing a basic path is as follows:

1. Begin the path: `context.beginPath()`
2. Create the sub-paths: one or more calls to `context.moveTo(x, y)` and `context.lineTo(x, y)`
3. Optionally close the path: `context.closePath()`
4. Draw the path: `context.stroke()`

The fill() Method

We saw the `stroke()` method for drawing the path. In cases where you have multiple connected sub-paths you can use the `fill()` method to fill the area with the current fill color (black by default):

Code Sample:

<html5-canvas/Demos/path-fill.html>

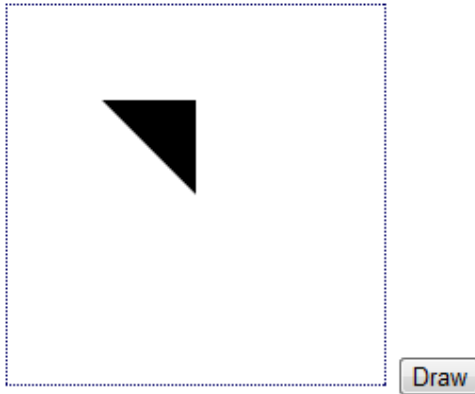
```

---- C O D E   O M I T T E D ----

if (canvas.getContext) {
    context = canvas.getContext("2d");
    context.beginPath();
    context.moveTo(50,50);
    context.lineTo(100,100);
    context.lineTo(100,50);
    context.fill();
}
---- C O D E   O M I T T E D ----

```

This will render as follows:



Note that in this example, the path was not closed (with `closePath()`) or drawn (with `stroke()`). The example below uses both `stroke()` and `fill()`. To see where the stroke starts and the fill ends, we have introduced the `fillStyle` and `strokeStyle` properties, which are used to set colors, and the `lineWidth` property, which sets the weight of the stroke.

Code Sample:

<html5-canvas/Demos/path-fill-stroke.html>

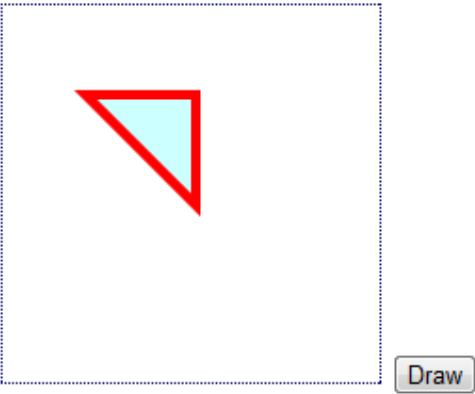
```

---- C O D E   O M I T T E D ----

if (canvas.getContext) {
  context = canvas.getContext("2d");
  context.lineWidth = 10;
  context.strokeStyle = "rgb(255,0,0)";
  context.fillStyle = "rgb(204,255,255)";
  context.beginPath();
  context.moveTo(50,50);
  context.lineTo(100,100);
  context.lineTo(100,50);
  context.closePath();
  context.stroke();
  context.fill();
}
---- C O D E   O M I T T E D ----

```

This will render as follows:



Lesson 1, Activity 5: Color and Transparency

As we saw in the previous presentation, we can set colors using the `fillStyle` and `strokeStyle` properties. For best support, when naming colors, choose from the following:

1. Color names (e.g., red, green, deeppink)
2. Hex colors (e.g., #ff0000, #008000, #ff1493)
3. Shortened hex colors (e.g., #f00, #080, #f19)
4. RGB functional notation (e.g., rgb(255,0,0), rgb(0,128,0), rgb(255,20,147))

Transparency

It is possible to make our strokes and fills semi-transparent using the `rgba(r, g, b, a)` (a for alpha) functional syntax:

- A value of 1 for a means fully opaque.
- A value of 0 for a means fully transparent.

Code Sample:

<html5-canvas/Demos/transparency.html>

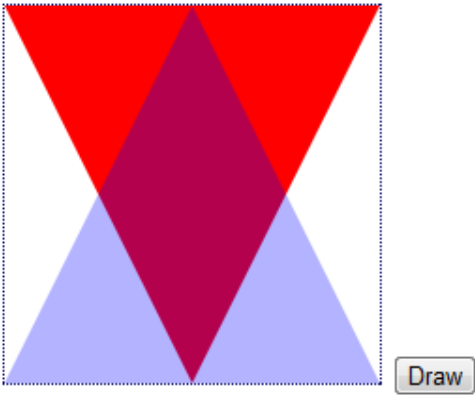
```

---- C O D E   O M I T T E D ----

if (canvas.getContext) {
    context = canvas.getContext("2d");
    context.fillStyle = "rgba(255,0,0,1)";
    context.beginPath();
    context.moveTo(0,0);
    context.lineTo(200,0);
    context.lineTo(100,200);
    context.fill();
    context.fillStyle = "rgba(0,0,255,.3)";
    context.beginPath();
    context.moveTo(0,200);
    context.lineTo(200,200);
    context.lineTo(100,0);
    context.fill();
}
---- C O D E   O M I T T E D ----

```

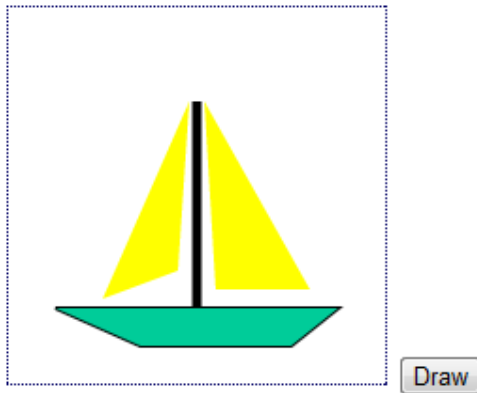
This will render as follows:



Lesson 1, Activity 6: Drawing a Sailboat

Duration: 20 to 30 minutes.

In this exercise, you will use HTML5 canvas to draw a simple sailboat like the one shown below:



1. Open html5-canvas/Exercises/sailboat.html in your editor.
2. Add the JavaScript code necessary to draw the sailboat pictured above.

Challenge

Have the left sail blink between different colors. You'll need to use some JavaScript skills to make this happen. If JavaScript isn't your thing, try adding a little person on your sailboat.

Solution:

html5-canvas/Solutions/sailboat.html

```

---- C O D E   O M I T T E D ----

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

//boat
context.fillStyle = "rgb(0,204,153)";
context.lineWidth = 2;
context.beginPath();
context.moveTo(25,160);
context.lineTo(70,180);
context.lineTo(150,180);
context.lineTo(175,160);
context.closePath();
context.stroke();
context.fill();

//pole
context.beginPath();
context.lineWidth = 5;
context.moveTo(100,160);
context.lineTo(100,50);

```

```

context.stroke();

//left sail
context.beginPath();
context.fillStyle = "rgb(255,255,0)";
context.moveTo(96,50);
context.lineTo(50,155);
context.lineTo(90,140);
context.fill();

//right sail
context.beginPath();
context.moveTo(104,50);
context.lineTo(160,150);
context.lineTo(110,150);
context.fill();
}
---- C O D E   O M I T T E D ----

```

Challenge Solution:

<html5-canvas/Solutions/sailboat-challenge.html>

```

---- C O D E   O M I T T E D ----

if (canvas.getContext) {
  context = canvas.getContext("2d");
  ---- C O D E   O M I T T E D ----
  blink(context,"rgb(255,255,0)");
  addSailor(context);
}
}

function blink(context,color) {
  context.fillStyle=color;
  context.beginPath();
  context.moveTo(96,50);
  context.lineTo(50,155);
  context.lineTo(90,140);
  context.fill();
  if (color == "rgb(255,255,0)") {
    color = "rgb(255,204,0)";
  } else {
    color = "rgb(255,255,0)";
  }
  setTimeout(function() {blink(context,color);},250);
}

function addSailor(context) {
  context.strokeStyle = "rgb(0,0,0)";
  context.lineWidth = 1;
  context.beginPath();
  //left leg

```

```
context.moveTo(130,160);
context.lineTo(135,152);
//right leg
context.lineTo(140,160);
//body
context.moveTo(135,152);
context.lineTo(135,140);
//arms
context.moveTo(130,145);
context.lineTo(140,145);
context.stroke();
//square head
context.beginPath();
context.lineWidth = 5;
context.moveTo(135,140);
context.lineTo(135,135);
context.stroke();
}
---- C O D E    O M I T T E D ----
```

Lesson 1, Activity 7: Rectangles

To create a rectangle, you specify the top-left starting position (x,y) and the width and height of the rectangle. There are three methods for creating rectangular shapes:

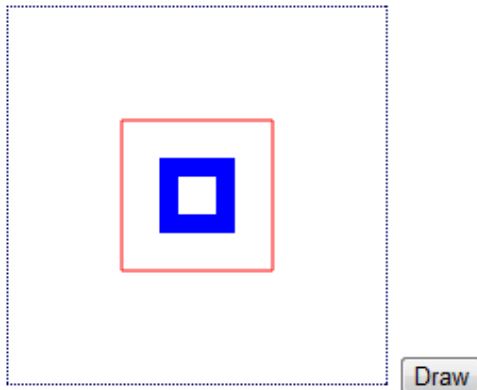
1. `fillRect(x,y,width,height)` - draws a filled rectangle.
2. `strokeRect(x,y,width,height)` - outlines a rectangle.
3. `clearRect(x,y,width,height)` - clears a rectangular area.

Code Sample:

html5-canvas/Demos/rect.html

```
---- C O D E   O M I T T E D ----  
if (canvas.getContext) {  
    context = canvas.getContext("2d");  
    context.strokeStyle="red";  
    context.fillStyle="blue";  
    context.fillRect(80,80,40,40);  
    context.strokeRect(60,60,80,80);  
    context.clearRect(90,90,20,20);  
}  
---- C O D E   O M I T T E D ----
```

The above code will render the following:



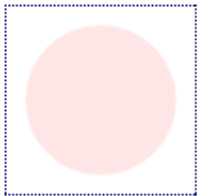
Lesson 1, Activity 8: Circles and Arcs

Note that a circle is just an arc that keeps going around until it gets to its starting point.

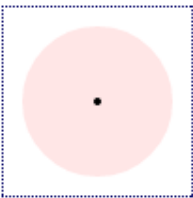
Circles and arcs are created by identifying the center of the circle (x,y) and then choosing a radius. You then set the starting and ending points on the circle and indicate whether to connect them going clockwise or counter-clockwise. The method looks like this:

```
arc(x,y,radius,startDegree,endDegree,counterclockwise);
```

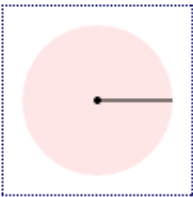
Let's break this apart. Imagine the following circle:



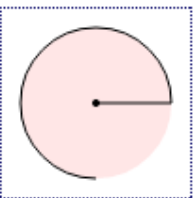
1. Identify the center point:

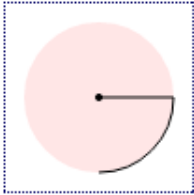


2. Choose a radius (e.g., 40). We draw the radius directly out to the right. Its end marks the 0 degree/radians point of the circle/arc:



3. Choose a starting point on the circle (usually 0) and an ending point (e.g., 90 degrees). The sixth and final argument indicates whether you want to connect these two points by moving along the circle counter-clockwise (true) or clockwise (false). The first image below shows the counter-clockwise result and the second shows the clockwise result:





The code for creating the above graphics is shown below:

Code Sample:

html5-canvas/Demos/arc-explained.html

```

---- C O D E   O M I T T E D ----
window.addEventListener("load",function() {
  var canvas=document.getElementById("my-canvas");
  if (canvas.getContext) {
    context = canvas.getContext("2d");
    context.beginPath();
    context.moveTo(50,50);
    context.arc(50,50,40,0,degreesToRadians(360),true);
    context.fillStyle="rgba(255,0,0,.1)";
    context.fill();
  }
}, false);

function centerPoint() {
  var canvas=document.getElementById("my-canvas");
  if (canvas.getContext) {
    context.fillStyle="rgba(0,0,0,1)";
    context = canvas.getContext("2d");
    context.beginPath();
    context.arc(50,50,2,0,degreesToRadians(360),true);
    context.fill();
  }
}

function radius() {
  var canvas=document.getElementById("my-canvas");
  if (canvas.getContext) {
    context = canvas.getContext("2d");
    context.beginPath();
    context.moveTo(50,50);
    context.lineTo(50+40,50);
    context.stroke();
  }
}

function showArc(ccw) {
  var canvas=document.getElementById("my-canvas");
  if (canvas.getContext) {
    context = canvas.getContext("2d");
    context.beginPath();
  }
}

```



```

    context.arc(50,50,40,0,degreesToRadians(90),ccw);
    context.stroke();
}
}

function degreesToRadians(degrees) {
    return (Math.PI/180)*degrees;
}
---- C O D E   O M I T T E D ----

```

Radians

Usually when we (meaning most humans) think of angles, we think in terms of degrees (e.g., a 90-degree angle). Others (meaning some math folks, including the people behind canvas) think in terms of radians. So, the fourth and fifth arguments of the `arc()` method are not in degrees, but in radians.

- A full circle is 2π radians, which is the same as 360 degrees.
- A semi-circle is π radians, or 180 degrees.
- A quarter-circle is $\pi/2$ radians, or 90 degrees.

For those of you who would prefer to continue to think in degrees, you can use this simple conversion function:

```

function degreesToRadians(degrees) {
    return (Math.PI/180)*degrees;
}

```

Without the `degreesToRadians()` function, a circle could be created like this:

```

context.arc(100,100,50,0,2*Math.PI,true);

```

Using the function, you would create the same circle like this:

```

context.arc(100,100,50,0,degreesToRadians(360),true);

```

To get a better understanding of radians and degrees and how the `counterclockwise` parameter works, open html5-canvas/Demos/arc-radians.html and html5-canvas/Demos/arc-degrees.html in your browser.

Lesson 1, Activity 10: Drawing a Snowman

Duration: 20 to 30 minutes.

In this exercise, you will use circles and squares to create a snowman like the one pictured below:



1. Open html5-canvas/Exercises/snowman.html in your editor.
2. Add the JavaScript code necessary to draw the snowman pictured above. You will need to add:
 1. A layer of snow on the ground.
 2. Three balls for the body and head.
 3. Eyes, mouth and nose.
 4. A hat.
 5. Arms.
 6. Buttons.
 7. A sun.

Challenge

Make the sun gradually disappear and the night grow darker.

Solution:

html5-canvas/Solutions/snowman.html

```
---- C O D E   O M I T T E D ----
```

```
if (canvas.getContext) {
```

```

context = canvas.getContext("2d");
//day
context.canvas.style.backgroundColor = "rgb(153,153,255)";

//ground snow
context.beginPath();
context.fillStyle = "rgb(255,255,255)";
context.fillRect(0,360,400,40);

//bottom ball
context.beginPath();
context.fillStyle = "rgb(255,255,255)";
context.arc(200,300,70,0,degreesToRadians(360),true);
context.fill();

//middle ball
context.beginPath();
context.fillStyle = "rgb(255,255,255)";
context.arc(200,200,50,0,degreesToRadians(360),true);
context.fill();

//head
context.beginPath();
context.fillStyle = "rgb(255,255,255)";
context.arc(200,125,35,0,degreesToRadians(360),true);
context.fill();

//right eye
context.beginPath();
context.fillStyle = "rgb(0,0,0)";
context.arc(190,115,4,0,degreesToRadians(360),true);
context.fill();

//left eye
context.beginPath();
context.fillStyle = "rgb(0,0,0)";
context.arc(210,115,4,0,degreesToRadians(360),true);
context.fill();

//mouth
context.beginPath();
context.fillStyle = "rgb(255,0,0)";
context.arc(200,125,15,degreesToRadians(45),degreesToRadians(135),false);
context.fill();

//nose
context.beginPath();
context.fillStyle = "rgb(255,102,0)";
context.moveTo(200,122);
context.lineTo(220,126);
context.lineTo(200,130);
context.fill();

//hat
context.beginPath();

```

```

context.fillStyle = "rgb(0,0,0)";
context.arc(200,100,25,0,degreesToRadians(180),true);
context.fill();
context.fillRect(163,94,75,8);

//left arm
context.beginPath();
context.lineWidth=2;
context.strokeStyle = "rgb(155,85,0)";
context.moveTo(220,180);
context.lineTo(300,160);
context.lineTo(315,165);
context.moveTo(300,160);
context.lineTo(315,158);
context.moveTo(300,160);
context.lineTo(315,150);
context.stroke();

//right arm
context.beginPath();
context.moveTo(170,180);
context.lineTo(110,240);
context.lineTo(115,255);
context.moveTo(110,240);
context.lineTo(95,255);
context.moveTo(110,240);
context.lineTo(85,255);
context.stroke();

//buttons
context.beginPath();
context.fillStyle = "rgb(0,0,0)";
context.arc(200,180,5,0,degreesToRadians(360),true);
context.arc(200,220,5,0,degreesToRadians(360),true);
context.arc(200,260,5,0,degreesToRadians(360),true);
context.fill();

//sun
context.beginPath();
context.fillStyle = "rgb(255,255,0)";
context.arc(400,0,70,0,degreesToRadians(360),true);
context.fill();
}
---- C O D E   O M I T T E D ----

```

Challenge Solution:

<html5-canvas/Solutions/snowman-challenge.html>

```

---- C O D E   O M I T T E D ----

var timer = null;
function drawPath() {

```

```

var canvas=document.getElementById("my-canvas");
if (canvas.getContext) {
---- C O D E   O M I T T E D ----

    darken(context,153,153,255,1);
}
---- C O D E   O M I T T E D ----
function darken(context,r,g,b,sunchop) {
    context.canvas.style.backgroundColor = "rgb("+r+","+g+","+b+")";
    context.beginPath();
    context.strokeStyle = "rgb("+r+","+g+","+b+")";
    context.lineWidth = sunchop;
    context.arc(400,0,70,0,degreesToRadians(360),true);
    context.stroke();
    if (r>0) r-=3;
    if (g>0) g-=3;
    if (b>0) b-=3;
    sunchop+=2;
    if (r>0 || g>0 || b>0) {
        timer = setTimeout(function() {darken(context,r,g,b,sunchop)},100);
    }
}
---- C O D E   O M I T T E D ----

```

Lesson 1, Activity 11: Quadratic and Bézier Curves

Canvas also includes functions for creating Quadratic and Bézier curves. These methods are shown below:

```
quadraticCurveTo(cp1x, cp1y, x, y)
bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y)
```

If you are new to Quadratic and Bézier curves, they can be difficult to get used to. For both:

1. Start with a line with a stated start point and end point.
 - The start point is the current position, which you can set using the `moveTo(x, y)` method.
 - The end point is set with the last two arguments of the method: `x` and `y`.
2. Add control points.
 - Just one for a Quadratic curve.
 - Two for a Bézier curve.
3. Imagine that these control points are tugging at the straight line created by the start and end points. By doing so, they create a curve.

The best way to see how they work is to practice with them a little. We've built small HTML5 applications for doing so.

Practice

We'll start with Quadratic curves:

1. Open html5-canvas/Demos/quadratic.html in your browser. You will see:
 1. A 500x500 Canvas.
 2. A menu in the upper right with a **Start Over** button for clearing the canvas and controls for setting the color.
 3. A `textarea` immediately below the Canvas, which holds the generated code, in case you create a drawing you want to use somewhere else.
 4. A list of the current curve point values to the right of the `textarea`.

To use this application:

1. Click anywhere on the Canvas to set a **start point**.
2. Click a second time to set an **end point**.
3. Click a third time to set the **control point**.
4. Click a fourth time to start over; that is, to set a **start point** for a new curve.

You may change the color any time before the fourth click to set a new color for the curve.

Play around with this for a little and then open html5-canvas/Demos/bezier.html in your browser to play with

creating Bézier curves. This application is the same, except that the fourth click sets a second control point.

If you generate a drawing you like:

1. Open html5-canvas/Demos/test-bed.html in your editor.
2. Copy the code from the textarea in the curve application.
3. Paste it between the start paste and end paste comments in test-bed.html and save.
4. Open test-bed.html in your browser. You should see your drawing there.

Lesson 1, Activity 12: When we were writing this course, we put together a video showing how you can use our Bézier Curve Application to build a simple "HTML5 heart." Review that [video here](#) and see if you can repeat the steps shown to build an HTML5 heart.

Lesson 1, Activity 13: Images

You can add existing images to your drawing with the `drawImage()` method, which is overloaded and has two signatures:

```
drawImage(image, x, y, width, height) //basic
drawImage(image, sx, sy, sWidth, sHeight, dx, dy, dWidth, dHeight) //sprites
```

In both cases, you need an image object as the first parameter. You can grab an image from the page (e.g., using `document.getElementById('my-image')`) or you can create an image object using JavaScript:

```
var img = new Image();
img.src = 'images/my-image.gif';
```

It is generally a good idea to hold off trying to display the image in your drawing until you are sure that it has loaded. So, you should wrap your drawing code as shown below:

```
img.onload = function() {
    context.drawImage(/*signature*/);
}
```

Let's look at the different signatures now.

drawImage() - Basic

The basic signature simply places the image at a given x,y position on the canvas. The `width` and `height` parameters are optional and are used for scaling or distorting the original image (generally a bad idea).

drawImage() Parameters (basic)

Parameter	Description
image	Image object
x	x position on canvas
y	y position on canvas
width	width of canvas
height	height of canvas

Here is a simple example:

Code Sample:

<html5-canvas/Demos/image-basic.html>

```

---- C O D E   O M I T T E D ----
if (canvas.getContext) {
    var context = canvas.getContext("2d");
    var logo = new Image();
    logo.src = 'Images/logo.png';
    logo.onload = function() {
        context.drawImage(logo,50,50);
    }
}
---- C O D E   O M I T T E D ----

```

This will render as follows:



drawImage() - Sprites

A sprite is an image file that contains several graphics used on a web page. By showing different parts of the sprite in different locations, it appears that there are several different images, but they are all contained in a single file, which translates to a single (faster) download.

To get this to work, we have to specify which part of the image (the source) we want to show and where (and how large) we want it to appear on the canvas. The table below shows the parameters the `drawImage()` method takes to create sprites.

drawImage() Parameters (sprites)

Parameter	Description
image	Image object
sx	X position of image (source)

sy	Y position of image (source)
sWidth	width of source (from X pos)
sHeight	height of source (from Y pos)
dx	X position of canvas (destination)
dy	Y position of canvas (destination)
dWidth	width of destination (for scaling)
dHeight	height of destination (for scaling)

Take a look at the following example:

Code Sample:

<html5-canvas/Demos/image-sprite.html>

```

---- C O D E   O M I T T E D ----
var img;
window.addEventListener("load",function() {
    img = new Image();
    img.src = 'Images/evolution.gif';
    draw();
},false);

function draw() {
    var canvas=document.getElementById("my-canvas");
    if (canvas.getContext) {
        var context = canvas.getContext("2d");
        img.onload = function() {
            context.drawImage(img, 10, 10, 60, 140, 60, 20, 60, 140);
            document.getElementById("btnEvolve").disabled=false;
        }
    }
}

function evolve(pic) {
    var pics = [
        {
            "sx" : 10, "w" : 60,
        },
        {
            "sx" : 74, "w" : 66
        },
    ]
}

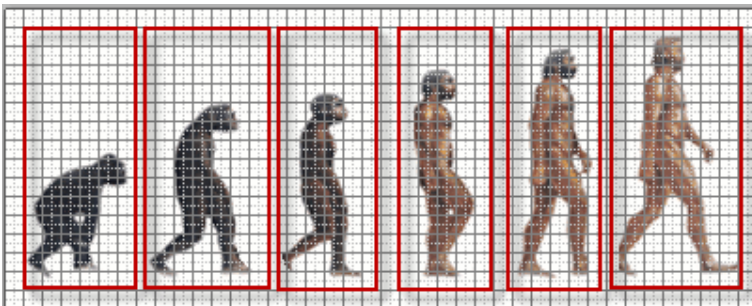
```

```

    "sx" : 143, "w" : 60
  },
  {
    "sx" : 217, "w" : 55
  },
  {
    "sx" : 270, "w" : 55
  },
  {
    "sx" : 324, "w" : 70
  }
];
var canvas=document.getElementById("my-canvas");
if (canvas.getContext) {
  var context = canvas.getContext("2d");
  context.clearRect(0,0,200,200);
  context.drawImage(img, pics[pic].sx, 10, pics[pic].w, 140, 60, 20, pics[pic].w, 140);
}
if (pic < 5) {
  setTimeout(function() {pic++; evolve(pic); }, 250);
}
}
---- C O D E   O M I T T E D ----

```

This page loads a single image (shown below with a grid overlay):



When the image loads, we show the first character in the image and enable the evolve button, so we can call the `evolve()` function, which recurses (calls itself) every 250 milliseconds passing on the next index of the `pics` array, which stores the X position of the image (`sx`) and the width (`w`) for both the source and destination.

Lesson 1, Activity 14: Text

There are two methods for adding text to the canvas:

1. `fillText(text, x, y)` - adds "solid" text at the x,y position.
2. `strokeText(text, x, y)` - adds "hollow" text at the x,y position.

Text Properties

You can set the following text properties:

1. `font` - uses the same syntax as the CSS font property: `font-style font-variant font-weight font-size/line-height font-family`
2. `textAlign` - possible values are "start," "end," "left," "right," and "center."
3. `textBaseline` - possible values are "top," "hanging," "middle," "alphabetic," "ideographic," and "bottom."

The `measureText(text)` method returns an object containing text metrics. Presumably, metrics will be added, but currently it only has one property: `width`. So, to see how wide some text would be in the current font, you would do this:

```
var width = context.measureText(text).width
```

Take a look at the following example:

Code Sample:

<html5-canvas/Demos/text.html>

```
---- C O D E   O M I T T E D ----
if (canvas.getContext) {
  var context = canvas.getContext("2d");
  context.fillStyle="red";
  context.strokeStyle="blue";

  context.font = "italic small-caps bold 44pt 'Comic Sans MS'";
  context.textAlign = "left";
  context.strokeText("Hello",10,100);
  context.fillText("World!",10+context.measureText("Hello ").width,100);
}
---- C O D E   O M I T T E D ----
```

Lesson 1, Activity 15: Images and Text

Duration: 30 to 40 minutes.

In this exercise, you will start with two images found in the [canvas/Exercises/Images](#) folder:

1. [south-america.gif](#) - a map of South America.
2. [flags.png](#) - a picture containing small graphics of country flags.

You will create the following drawing:



Notice the text under the graphics.

1. Open [html5-canvas/Exercises/south-america](#) in your editor.
2. Add the JavaScript code necessary to:
 1. Create the image objects and set their source values.
 2. Draw the backdrop (the map).
 3. Place the flags using the sprite method shown earlier. Each flag is 18 pixels wide and 13 pixels high. The source and destination positions are shown in the table below.
 4. Add the country names.

Country	Source X	Source Y	Destination X	Destination Y
---------	----------	----------	---------------	---------------

Chile	283	88	100	250
Argentina	255	4	130	300
Brazil	171	60	200	170
Paraguay	59	452	170	250
Uruguay	59	564	185	310
Bolivia	59	200	135	210
Peru	31	424	75	170

Solution:

<html5-canvas/Solutions/south-america.html>

```

---- C O D E   O M I T T E D ----
if (canvas.getContext) {
    var context = canvas.getContext("2d");
    var backdrop = new Image();
    var flags = new Image();
    backdrop.src = 'Images/south-america.gif';
    flags.src = 'Images/flags.png';
    backdrop.onload = function() {
        context.drawImage(backdrop,0,0);
    }
    flags.onload = function() {
        context.drawImage(flags, 283, 88, 18, 13, 100, 250, 18, 13);
        context.fillText("Chile",100, 250);
        context.drawImage(flags, 255, 4, 18, 13, 130, 300, 18, 13);
        context.fillText("Argentina",130, 300);
        context.drawImage(flags, 171, 60, 18, 13, 200, 170, 18, 13);
        context.fillText("Brazil",200, 170);
        context.drawImage(flags, 59, 452, 18, 13, 170, 250, 18, 13);
        context.fillText("Paraguay",170, 250);
        context.drawImage(flags, 59, 564, 18, 13, 185, 310, 18, 13);
        context.fillText("Uruguay",185, 310);
        context.drawImage(flags, 59, 200, 18, 13, 135, 210, 18, 13);
        context.fillText("Bolivia",135, 210);
        context.drawImage(flags, 31, 424, 18, 13, 75, 170, 18, 13);
        context.fillText("Peru",75, 170);
    }
}
---- C O D E   O M I T T E D ----

```