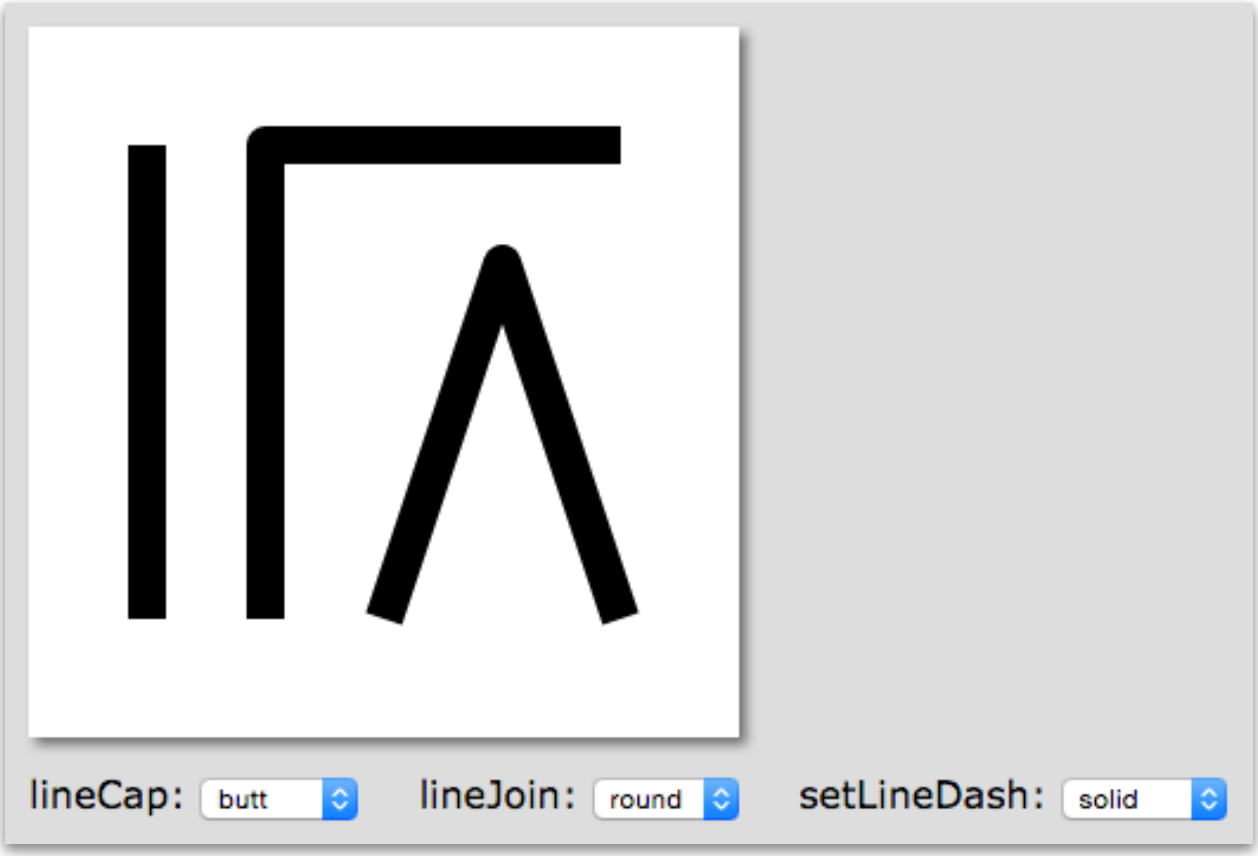
Study Guide 2
Due: In dropbox start of class, second meeting of week 3. Late submissions will not be accepted.

Our free canvas "textbook" is located here: https://developer.apple.com/library/safari/documentation/AudioVideo/Conceptual/HTML-canvas-guide/Introduction/Introduction.html

I. More drawing lines

1) Read the **Drawing Lines and Shapes** - "Drawing Lines" and "Setting Caps and Joins" sections.

Download **2-1-line-cap-and-join-start.html** file from mycourses. The completed version is a demo of using the `.lineCap` and `.lineJoin` properties, and the `.setLineDash()` method. It should look like this when it's done:



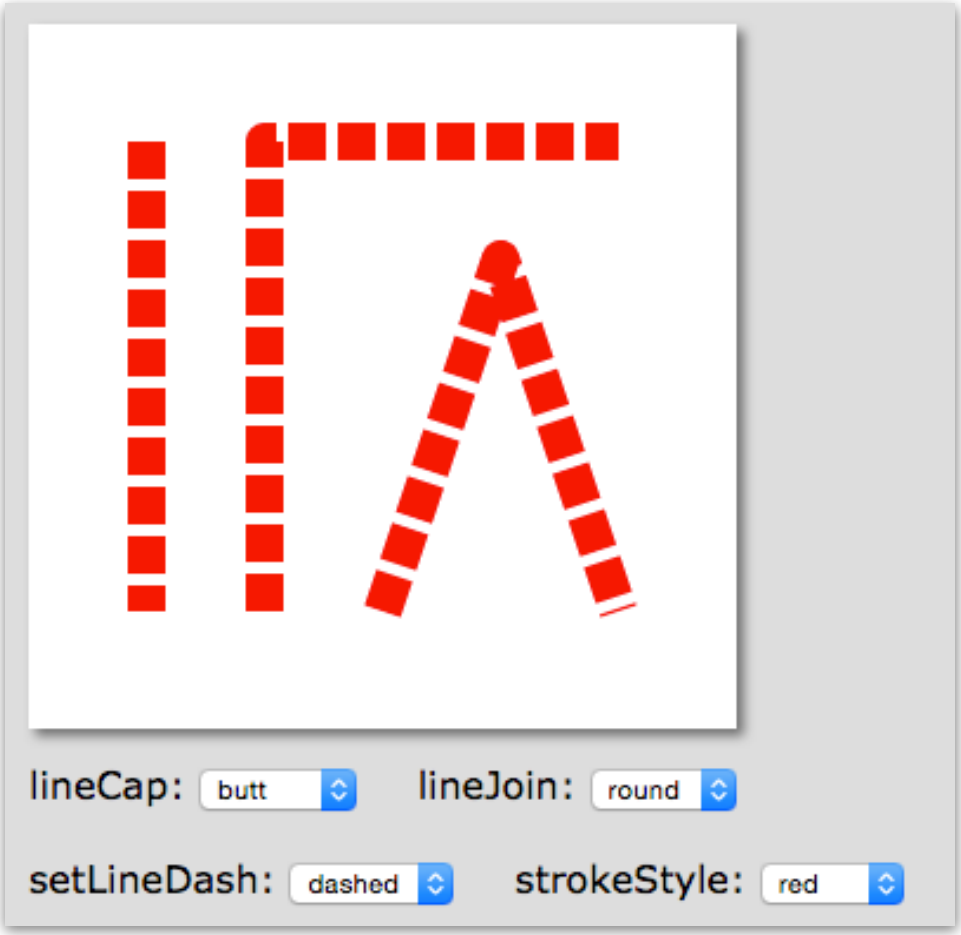
The code is all set for you, except for the implementation of the `drawLines()` function. Here it is:

```
function drawLines(){
    /* #4 - start drawing! */
    // clear screen
    ctx.clearRect(0,0,300,300);

    ctx.beginPath();
    // subpath for left line
    ctx.moveTo(50, 50);
    ctx.lineTo(50, 250);

    // subpath for middle line
    ctx.moveTo(100, 250);
    ctx.lineTo(100, 50);
    ctx.lineTo(250, 50);

    // subpath for right line
    ctx.moveTo(150, 250);
    ctx.lineTo(200, 100);
    ctx.lineTo(250, 250);

    // don't close the path or we'll get a triangle!
   // ctx.closePath();

    // we can't yet see the path, so stroke it.
    ctx.strokeStyle="black";
    ctx.lineCap = gLineCap;
    ctx.lineJoin = gLineJoin;
    ctx.setLineDash(gLineDash);
    ctx.lineWidth = 16;
    ctx.stroke();
}
```

2) Reference
More Dotted and Dashed lines demo using ctx.setLineDash():
http://www.rgraph.net/blog/2013/january/html5-canvas-dashed-lines.html

ctx.miterLimit demo:
https://developer.mozilla.org/samples/canvas-tutorial/4_8_canvas_miterlimit.html

3) HW deliverable - ***Line Demo A***
Oh, one more thing - add another `<select>` that lets the user choose the color of the lines - give them at least 4 choices including black (the default). ZIP and post to the dropbox - but don't do so yet, we're going to keep adding onto this.



Hint: name the global variable that holds the color info `gStrokeStyle`

II. Gradients

1) Read the **Gradients and Patterns** - “Linear Gradients” and “Radial Gradients” and “Animating Gradients” sections**.**

• When you stroke or fill something on the canvas, it’s drawn using the current stroke or fill style. The stroke or fill style can be set to a **color**, a **pattern**, or a **gradient**.

• A gradient specifies a starting color, an ending color, and an area over which color changes. A single gradient can encompass more than one color change.

• The 2D canvas drawing context supports two kinds of gradients: *linear* and *radial*.

• `createLinearGradient()`, `createRadialGradient()`, and `addColorStop()` are the canvas methods we use to create gradients.

2) Look over the 3-1,3-2, and 3-3 code listings and sample code (see mycourses). These illustrate how to create and apply both linear and radial gradients.

Figure 3–1 Linear gradient

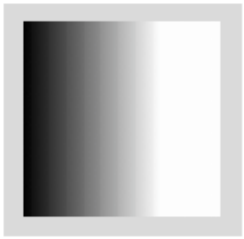
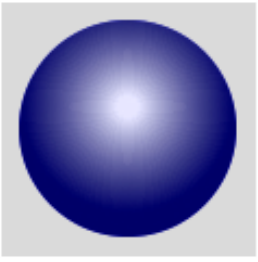


Figure 3–2 Linear gradients

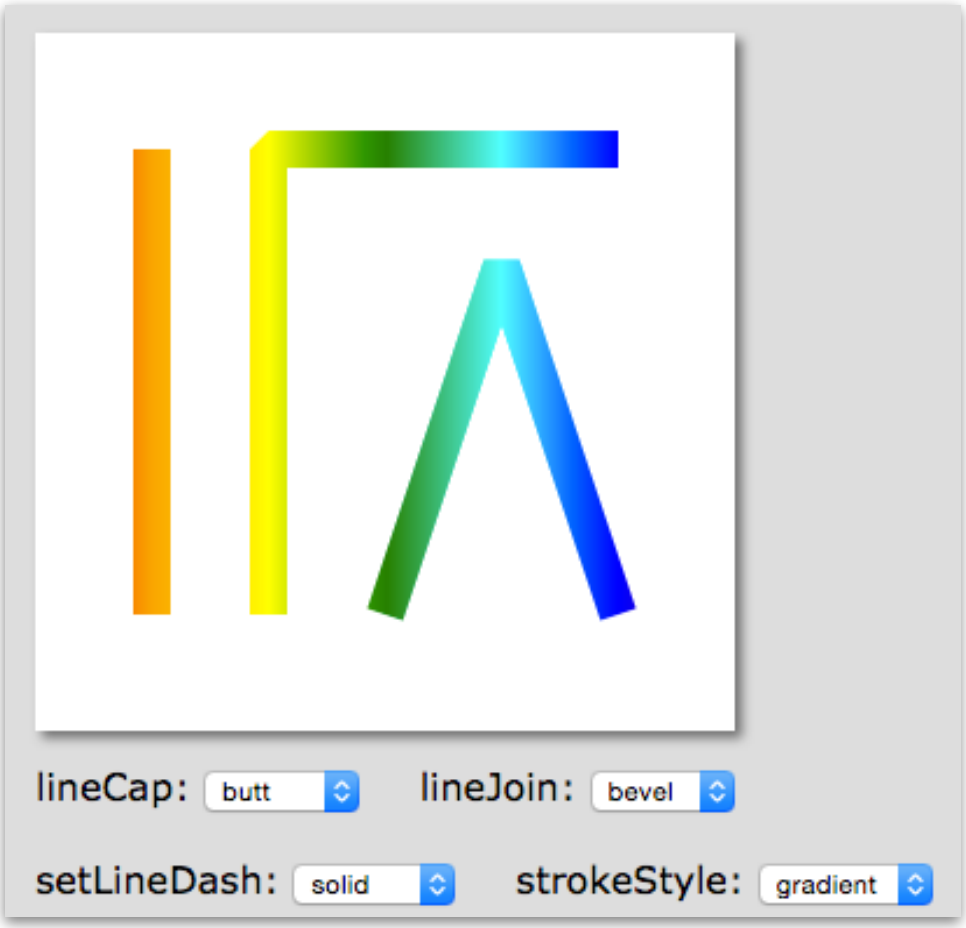
Figure 3–3 Rainbow gradient

Figure 3–5 Ball with gradient overlay



2) HW deliverable - ***LineDemo B***
Add an option for a linear gradient to your strokeStyle chooser - make the gradient run between at least 3 colors, and make it at least 200 pixels wide. Below I took the rainbow gradient from the sample code and made it 300 pixels wide (the width of the canvas).

Note that the red and the purple colors from opposite ends of this gradient don't show up because there is no path to apply the gradient to on the left or right of the canvas.



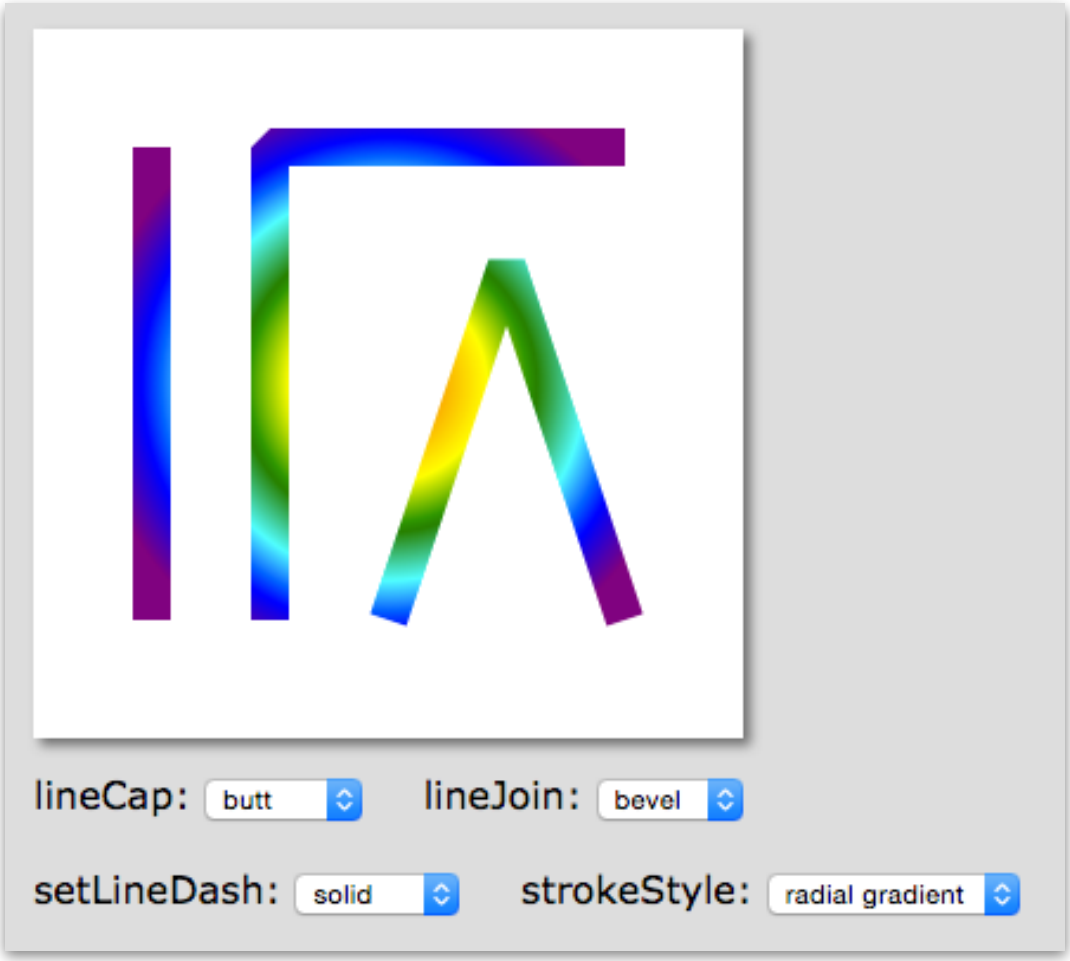
Hints:
Our gradient `<option>` looks like this:

```
<option value="gradient">gradient</option>
```

The event handling code looks like this:

```
if (e.target.value == "gradient"){
      // make a gradient here
      // more gradient code here
      gStrokeStyle = grad;
}else{
      gStrokeStyle = e.target.value;
}
drawLines();
```

3) HW deliverable - ***LineDemo C*** - add a strokeStyle option to ***LineDemo*** that lets the user choose a *radial* gradient.



Here we re-used the rainbow color stops, and made the radius of the outer circle 125 pixels.

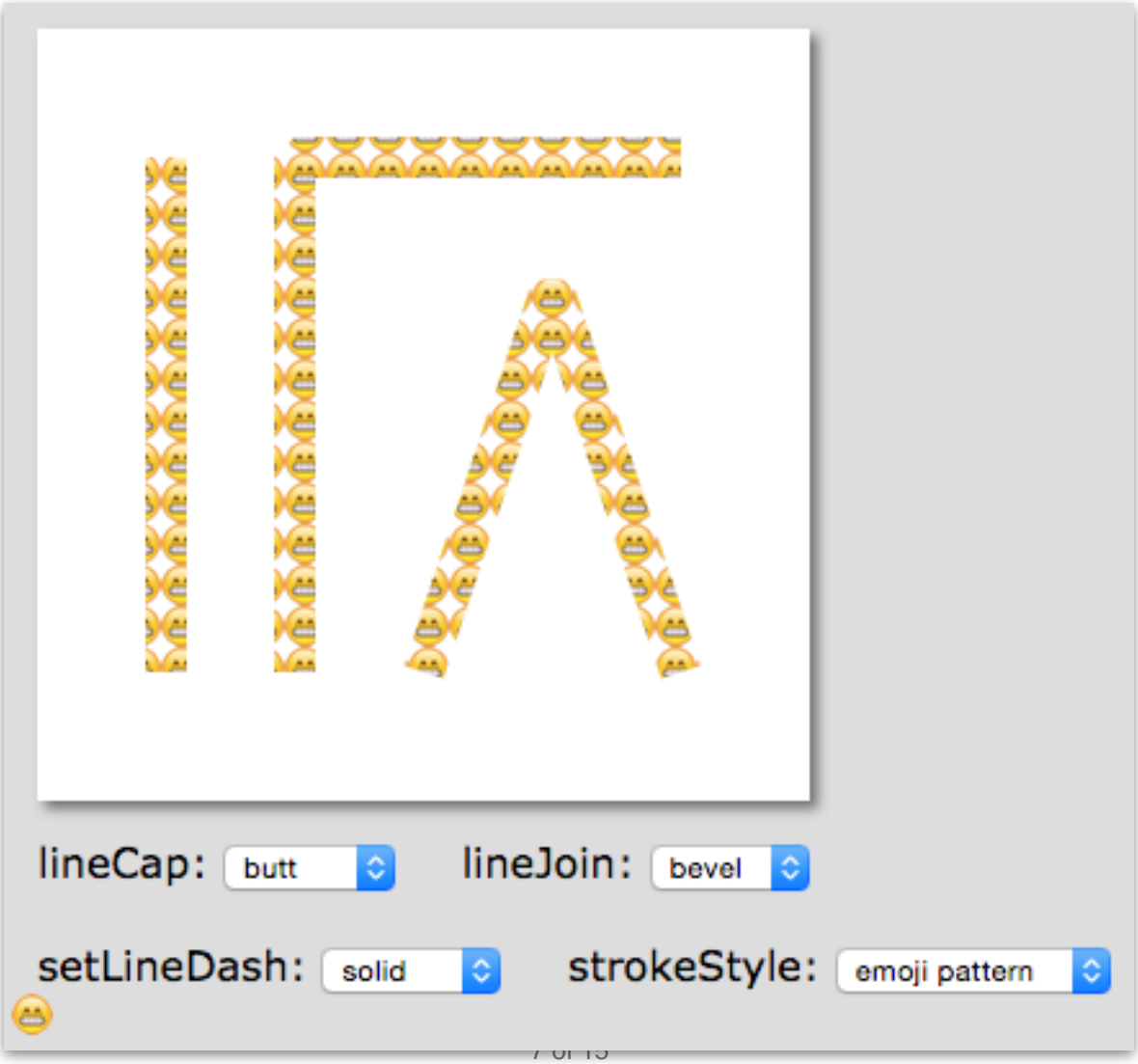
III. Patterns

• Read the **Gradients and Patterns** - “Patterns” section. Patterns are really easy — just call `createPattern()` and pass in an image (or a video, or another canvas). See some code examples here:

https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/createPattern

• Take a look at the sample code for **3-4-creating-patterns.html** - note that we need to preload an image before we can use it in the pattern. The easiest way to do that it to put it on the page in an `<img>` tag.

1) HW deliverable - ***LineDemo D*** - add a strokeStyle option to ***LineDemo*** that lets the user choose a pattern.



Hints:

- put the emoji image on the page (so that it's loaded by the browser) like this:

```
<img id="emoji" src="grimacing-face-emoji-small.png" />
```

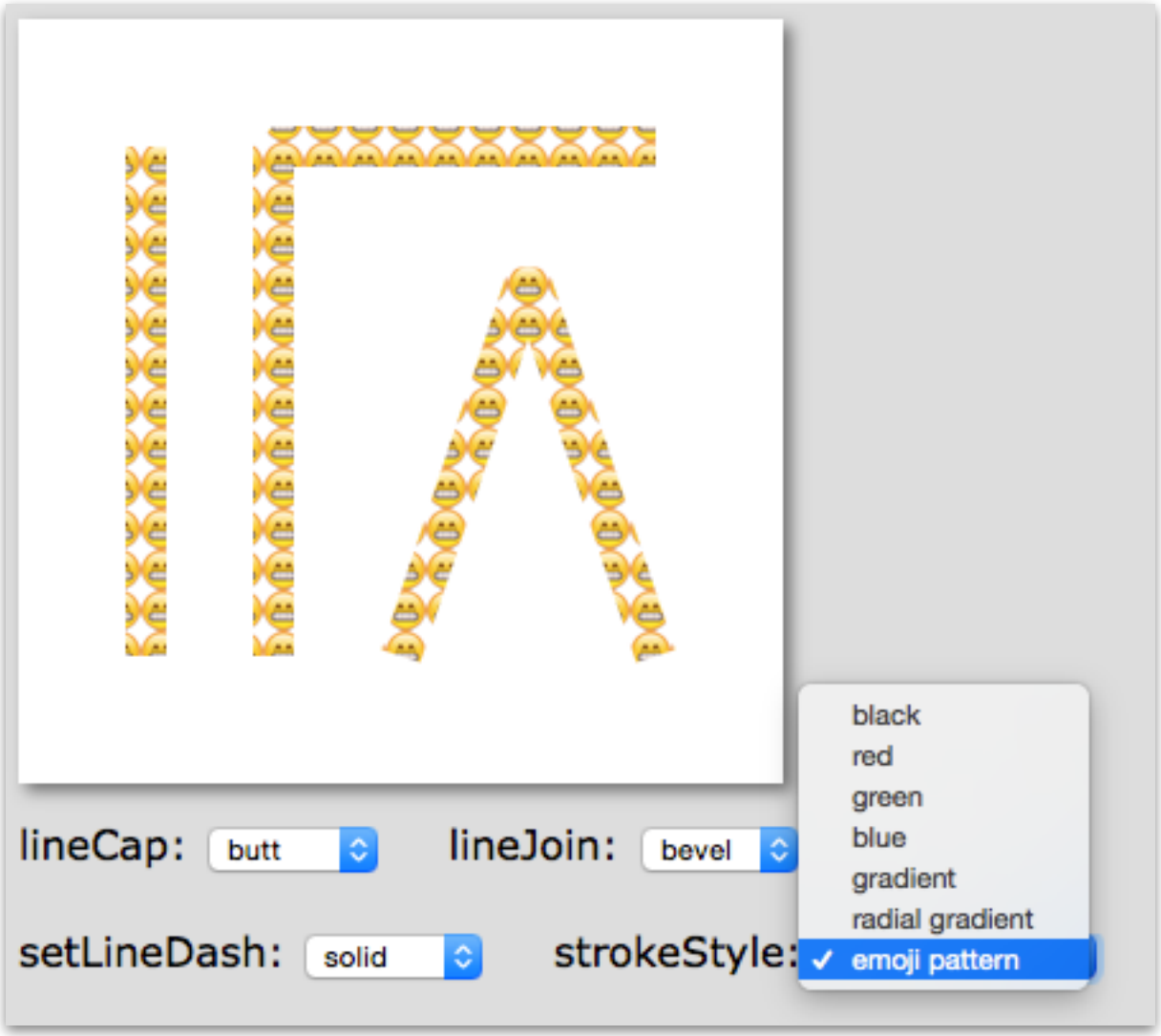
- If you want to hide the image on the page, here's the CSS:

```
#emoji{
     display:none;
}
```

- If your strokeStyleChooser `onchange` logic is getting ugly with a lot of if/else clauses, use a case statement or something like this:

```
document.querySelector('#strokeStyleChooser').onchange = function(e){
                if (e.target.value == "gradient"){
                    var grad = ctx.createLinearGradient(0, 0, 300, 0);
                    // add color stops
                    ////
                    gStrokeStyle = grad;
                    drawLines();
                    return; // bail out of function
                }
                if(e.target.value == "radialgradient"){
                    var grad = ctx.createRadialGradient(150,150, 0, 150, 150, 125);
                    // add color stops
                    ////
                    gStrokeStyle = grad;
                    drawLines();
                    return; // bail out of function
                }

                if(e.target.value == "emojipattern"){
                    // get image
                    // create pattern
                    gStrokeStyle = pat;
                    drawLines();
                    return; // bail out of function
                }

                // else just grab the color
                gStrokeStyle = e.target.value;
                drawLines();
            };
```

Your final version of ***LineDemo*** should look like this:



Submission:
ZIP and post LineDemo-D folder (with image files)

Rubric (10 total points)
A completed = 2.5 points
B completed = 2.5 points
C completed = 2.5 points
D completed = 2.5 points

IV. Drawing Bezier Curves

• Read the **Drawing Lines and Shapes** - “Drawing Bezier Curves” section.

Drawing Bezier Curves

You draw bezier curves in the same connect-the-dots manner as you draw lines, but instead of using the `lineTo()` method, you use either the `bezierCurveTo()` method, which connects the endpoints with a cubic bezier curve using a specified pair of control points, or the `quadraticCurveTo()` method, which connects the endpoints with a quadratic bezier curve using a single specified control point.

The following snippet draws two bezier curves, one cubic and one quadratic, from the upper left corner of the canvas to the lower right corner of the canvas.

```
function init() {
    can = document.getElementById("can");
    ctx = can.getContext("2d");
    var wide = can.width;
    var high = can.height;
}
function drawCurves() {
    ctx.strokeStyle = "black";
    var ctrlX = 5;
    var ctrlY = 150;
    var ctrlXa = 50;
    var ctrlYa = 300;
    ctx.beginPath();
    ctx.moveTo(0, 0);
    ctx.quadraticCurveTo(ctrlX, ctrlY, wide, high);
    ctx.stroke();
    ctx.beginPath();
    ctx.moveTo(0, 0);
    ctx.bezierCurveTo(ctrlX, ctrlY, ctrlXa, ctrlYa, wide, high);
    ctx.stroke();
}
```

`ctx.quadraticCurveTo(ctrlX, ctrlY, endX, endY)` draws bezier curves with 1 control point.

`ctx.bezierCurveTo(ctrlX, ctrlY, ctrlXa, ctrlYa, endX, endY)` draws cubic bezier curves with 2 control points.

More on bezier curves at these links:
https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/bezierCurveTo

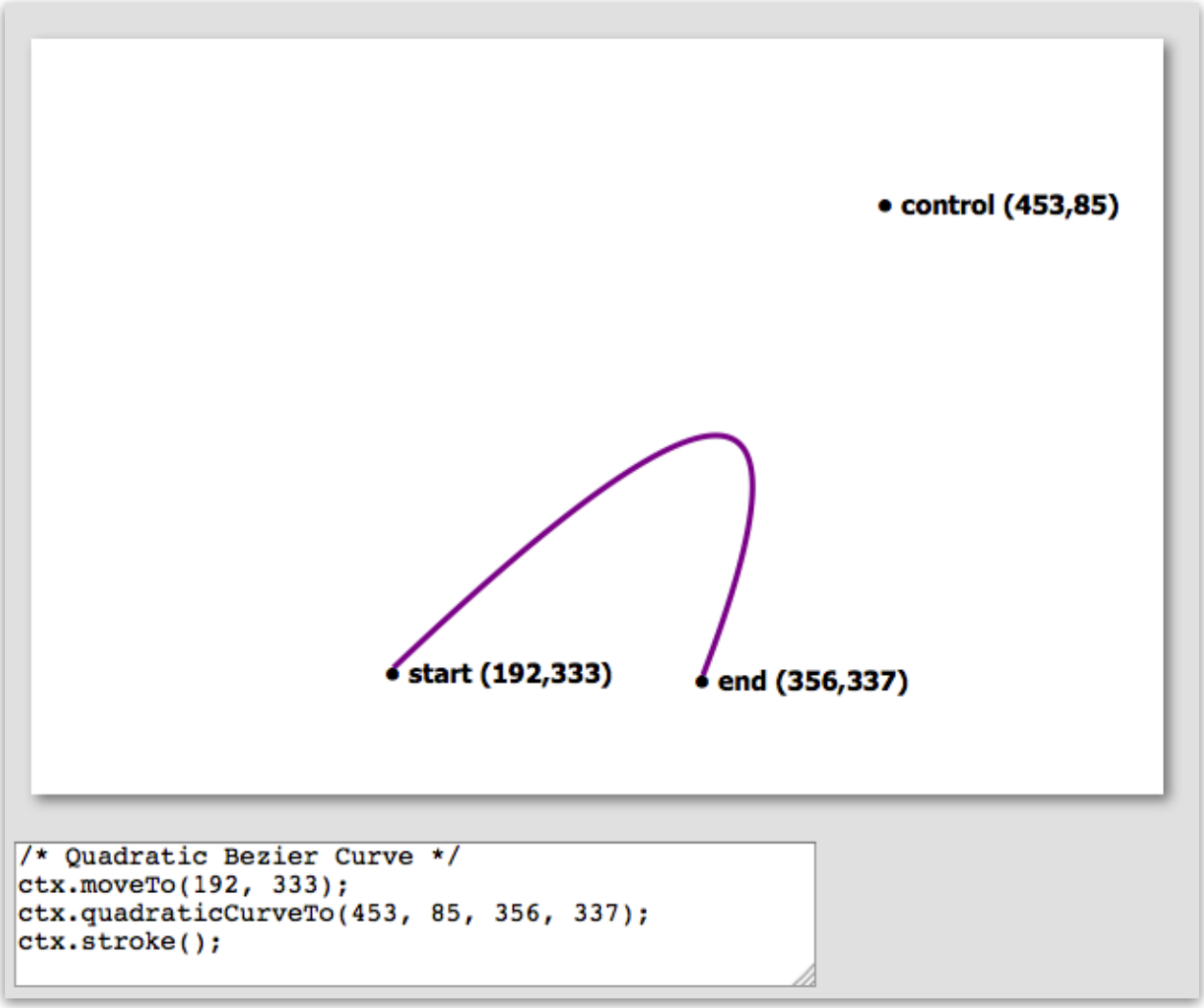
https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/quadraticCurveTo

http://en.wikipedia.org/wiki/Bézier_curve

1) Open up the **2-2-bezier-curves.html** file from mycourses. This gives the results of the previous code listings. The red line is a cubic bezier curve (2 control points), and the black line is a quadratic bezier curve (1 control point)

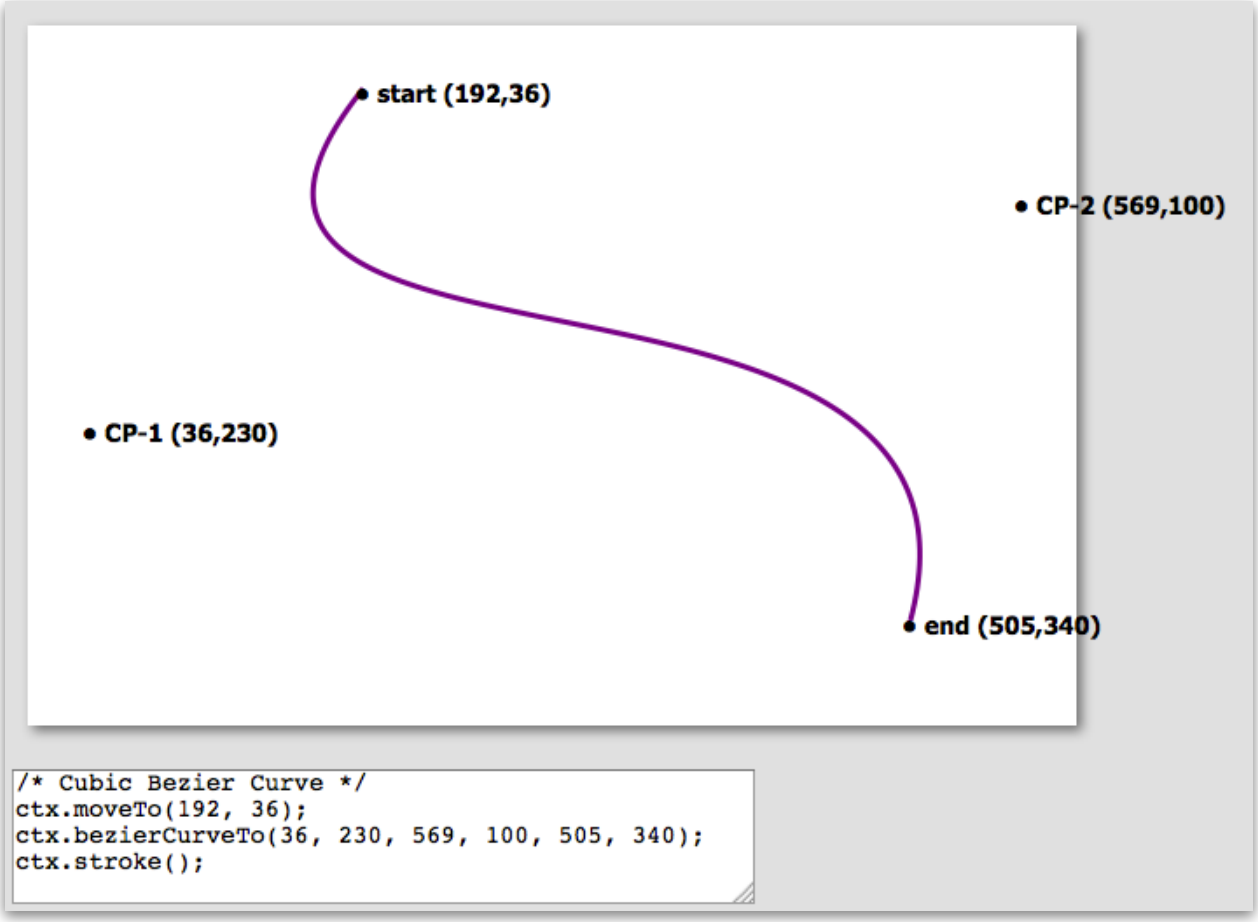
Go ahead and edit the file and play with the values a little bit.

2) Understanding and using bezier curves could really help you out in the upcoming *Audio Visualizer* assignment. To help you visualize how these curves work, check out the **quadratic-bezier-curves-playground.html** demo file we put together for you:



This is demoing how quadratic bezier curves work. The start, end, and control points can be dragged with the mouse, and the code you need to generate the curve will be presented in the `<textarea>` for your copy and paste pleasure (you can paste this canvas code into the previous example file to see it in action)

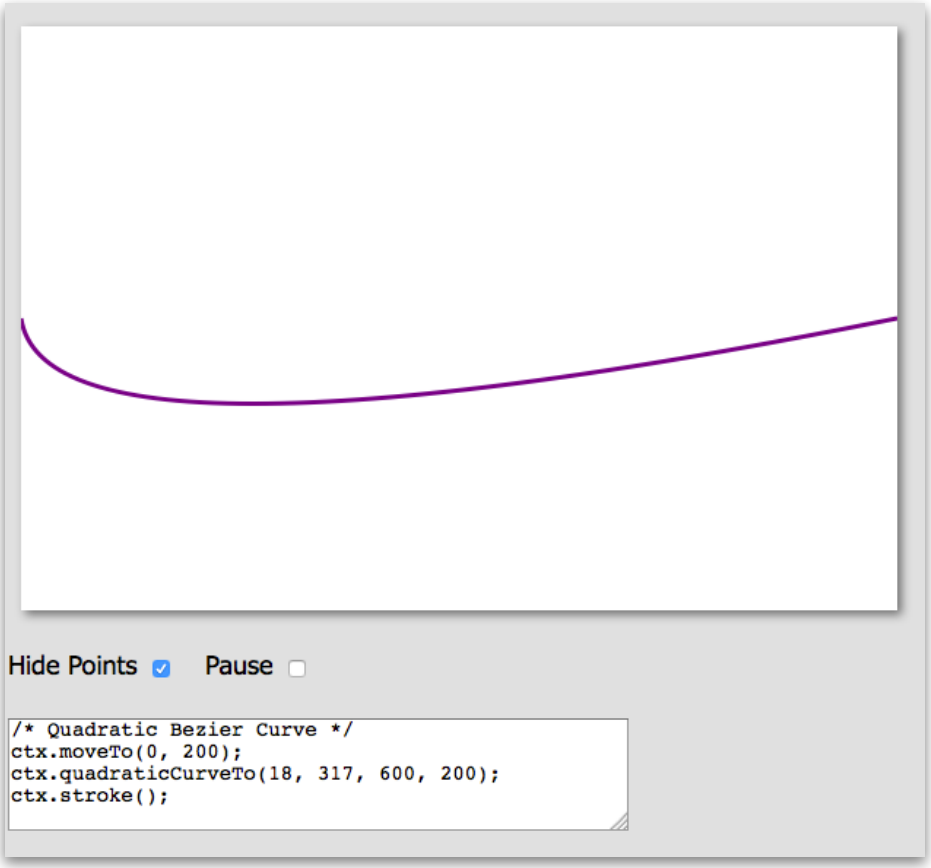
3) Similarly, to help you visualize how cubic bezier curves work, check out the **cubic-bezier-curves-playground.html** demo file:



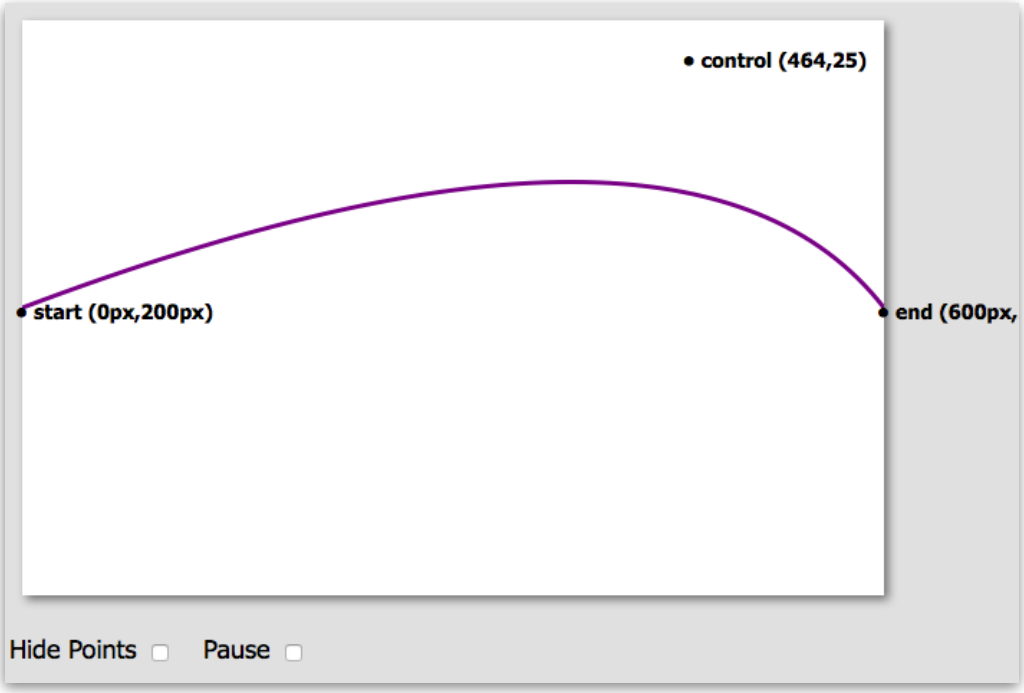
The start, end, and both control points can be dragged with the mouse, and the code you need to generate the cubic bezier curve will be generated.

Note about these last 2 examples: There's only three lines of actual drawing code here, but there's quite a bit more code needed to get the dragging of control and end points working. These points are actually `<p>` tags that have CSS `position:absolute` - and we move them with the `mousemove` event.

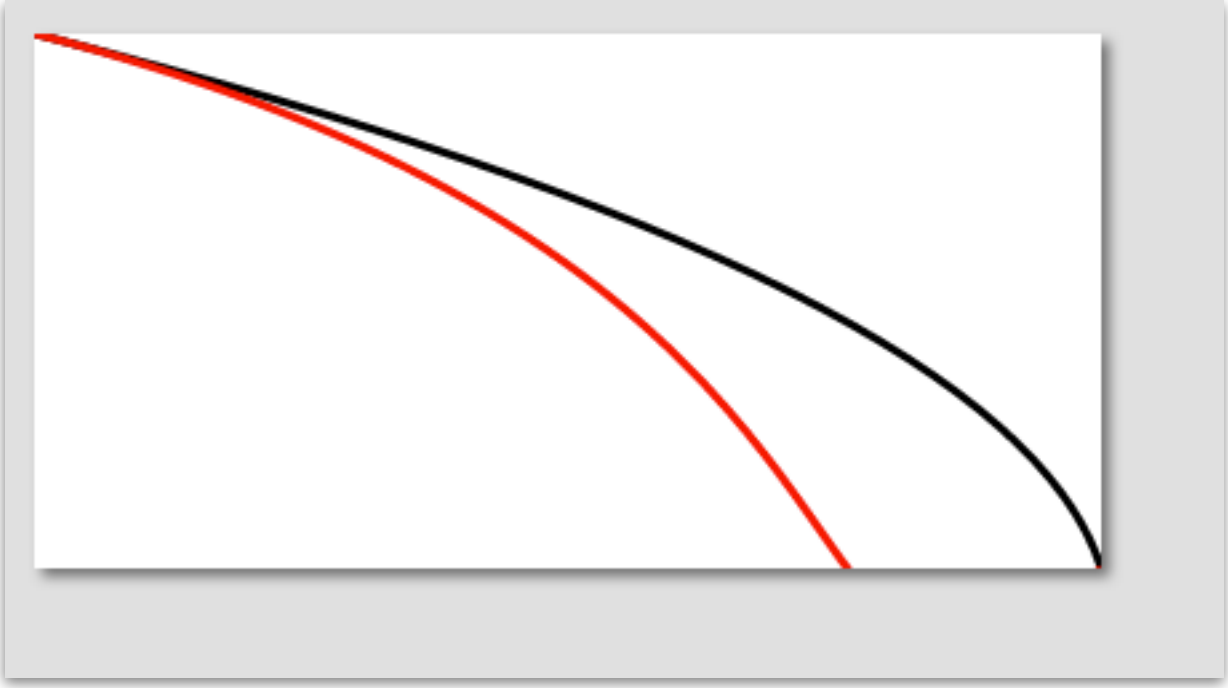
4) Lastly, we look at animating a curve - open the **animated-curves-playground.html** demo file. Here we're just moving the quadratic bezier curve control point around on the screen.



Unchecking **Hide Points** gives us a better look at what's going on.



5) More HW - make a copy of **2-2-bezier-curves.html** named **moving-curves.html** to use as a starting point. Animate at least one of the control points using `requestAnimationFrame()` - see the previous example for ideas.



Hints:

- Make `ctrlX`,`ctrlY`, `speed`, and `vector` global variables.

- If you decide to animate a second control point, you'll need a `vectora` global

- 600x400 is a nice width for your canvas

- you won't need `parseInt()` or `parseFloat()` or `toFixed()` from the previous example. We only needed those because we were animating DOM elements. This HW is much simpler in that everything is happening in canvas.

V. Submission

This SG is worth 2 HW assignments.

1) ZIP up your **LineDemo-D** folder and submit it to the mycourses dropbox - you can earn up to 10/10 points on this first assignment.

2) ZIP up your **moving-curves.html** file and submit it to the mycourses dropbox - this counts as the second assignment and is worth up to 10/10 points