Study Guide 1 - Intro to Canvas

Name:___________________________

(non-) submission Instructions: We are not collecting this document or the answers, but you might see some of this on an exam, so be sure to fill it out and keep it handy.

Origins of canvas:

• Apple invented the canvas 2D drawing API in 2004 as a way to add customized dashboard widgets to OS X. It has since been adopted by all major desktop and mobile browsers and is supported by the current versions of Chrome, Firefox, Edge, Opera and even Internet Explorer (8>)

- The `<canvas>` tag defines an area on the page that we can draw into using a procedural drawing API. This bitmap drawing API has methods for drawing lines, paths, rectangles, arcs, circles, curves, and text.

• The canvas drawing API allows a developer to create Flash-like (do you remember Adobe Flash?) games and experiences without using the Flash SDK and with no need for a browser plug-in.

• The canvas API is focused on drawing, and is fairly lightweight. The full specification is here: http://www.w3.org/TR/2dcontext/

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

- Another great online reference is here: https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial

I. About Canvas

1) Read the “At a glance” section under *About Canvas*

A. **You can add a Canvas element with a few lines of code**
(*no answer required* - we’ll do this in the next section)

B. **There are Methods for Drawing** _______

You draw shapes other than rectangles by creating a path, adding line segments, curves, or arcs, and closing the path. Begin a path using `beginPath()`. Set the starting point, or start a discontinuous subpath, by calling the `moveTo(x, y)` method. The `closePath()` method draws a line from the current endpoint to the starting point of the path, creating a closed shape.

The path is not actually drawn until you _______

Canvas supports matrix transforms—anything you draw can be _______

C. **It’s Easy To Include** _______

D. **You Can Also Render** _______

E. **Canvas is Great for InfoGraphics** - give examples:

F. **Canvas Can Create Fast, Lightweight Animations**.
(*no answer required* - note: Apple's examples use `setInterval()` for animation - we'll instead be using `requestAnimationFrame()` in our examples)

G. **You Can Manipulate Pixels Directly for Image Processing** - give examples:

H. **Make Games That Play on Desktop and iOS Devices**.
(or Android, Blackberry, Windows, … - *no answer required*)

I. **Export to Canvas is Possible from Adobe Illustrator**

no answer required

II. Setting Up the Canvas

1) Read over the “Setting Up the Canvas” section - the following steps are listed:

- Start by Adding a `<canvas>` Tag

- Specify the Fallback Behavior

- Create a Drawing Context

- Support Retina Displays from the Start
(a good practice, but we’re not going to worry about this in the course)

- Save and Restore the Context
(we’ll get into this soon)

2) Your first canvas App:

The canvas drawing API allows a developer to create interactive experiences and games without the need for a browser plug-in.

You need to do the following to get started drawing in canvas on an HTML page:

i. Wait for the HTML page to load

ii. Get a reference to the `<canvas>` tag on the page

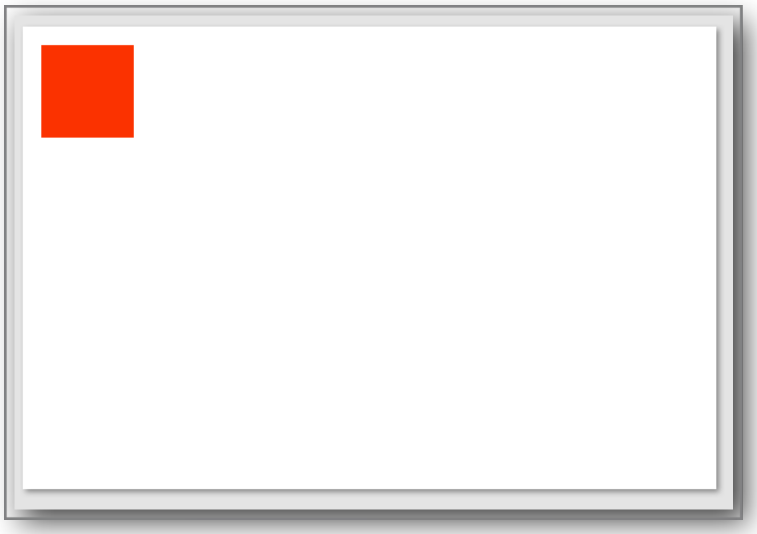
iii. Get a reference to the “drawing context”, which is the 2D drawing API we use to do the actual drawing.

iv. Start issuing drawing commands!

Here is a simple example (see mycourses for **sg-1.html**):

```
<!DOCTYPE html>
<html lang="en">
<head>
      <meta charset="utf-8" />
      <title>Canvas 1</title>
      <script>

            "use strict";
            // #1
            window.onload = init;

            function init(){
                  // #2
                  let canvas = document.querySelector('canvas');

                  // #3
                  let ctx = canvas.getContext('2d');

                  /* #4 - start drawing! */
                  // draw filled square
                  ctx.fillStyle = "red";
                  // ctx.fillRect(x,y,width,height);
                  ctx.fillRect(20,20,100,100);

            }

      </script>
</head>
<body>
      <canvas id="canvas" width="750" height="500">
         Get a browser that supports Canvas!
      </canvas>
</body>
</html>
```

Go ahead and load this example in the browser so you can see the exciting red square on the screen. Note that (0,0) is in the upper-left corner of the screen. x-values get larger as you move to the right, and y-values get larger as you move down the screen.



The **sg-1.html** example that's posted to mycourses has additional JS code that you can uncomment and see more drawing examples.

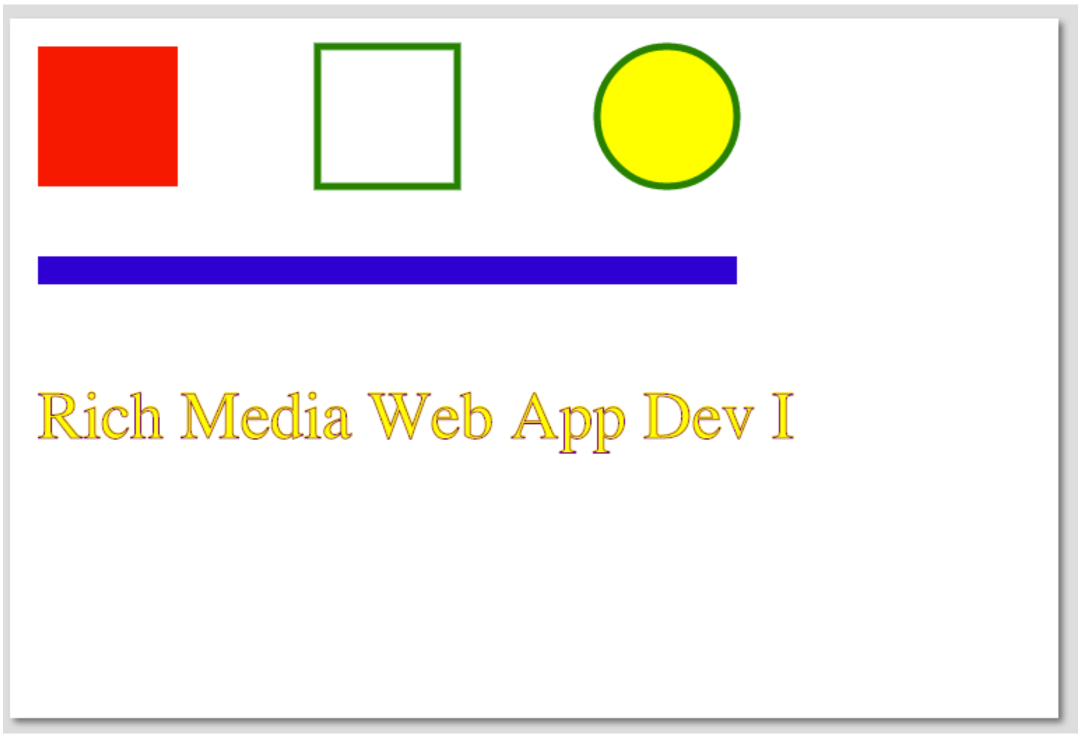
Before you move on, go get this file and play with it now. Uncomment the remaining code section-by-section and experiment:

- change the `x/y/width/height` values,
- change the `.fillStyle` and `.strokeStyle` values
- change the `startAngle` and `endAngle` values

to see what happens.

Also note that `.strokeText()` strokes the outline of the text with the current `.strokeStyle`, and `.fillText()` fills the text with the current `.fillStyle` You can use these together to get some nice effects.

Here's the default version:



3) For the following, modify the **sg-1.html** example file to get the answer (first comment out all of the drawing code under step #4).

For your answers, assume that you already have a variable named `ctx` that points at the drawing context. Note: The squares only require 1 or 2 lines of code, the circles will require a few more.

A. Write code that will **fill** a black 50x50 square at (0,0)

B. Write code that will **fill** a purple 50x50 square at (300,300)

C. Write code that will **stroke** a purple 50x50 square at (300,300)

D. Write code that will **stroke** a 5 pixel thick green line between (0,100) and (100,500)

E. Write code to **fill** a black circle at (0,0) that has a radius of 50 pixels.

F. Write code to **fill** a yellow semi-circle (180 degrees ie. `Math.PI` radians) at (200,200) that has a radius of 100 pixels.

G. What are the default colors for `.fillStyle` and `.strokeStyle` if you don't specify them?

H. Try giving `.fillStyle` a non-existent CSS value for a color like "pukegreen" - what color do you get instead?

I. What is the default value of `.lineWidth` if you don't specify a value?

J. What is the default `height` and `width` of the `<canvas>` tag if you don't specify a value (use google or the Web Inspector to find out)

III. More above the Canvas API

1) Read over **Set the Stroke and Fill Styles**:

A. Color can be specified in all of the usual CSS ways - give the `.fillStyle` color for white in each format below :

i) CSS Keyword: `ctx.fillStyle = "white";`

ii) Longhand hexadecimal:

iii) Shorthand hexadecimal:

iv) RGB percentage:

v) RGB value:

vi) RGBa value:

Note that a gradient or pattern can also be used for `.fillStyle` *and* `.strokeStyle` *values. HSL and HSLA values are also supported: http://www.w3schools.com/cssref/css_colors_legal.asp*

2) Read over **Drawing Rectangles**:

List the 3 rectangle methods below:

There's also a 4th one, `rect()`*, that creates a rectangle path without filling or stroking it. We'll get into that one later.*

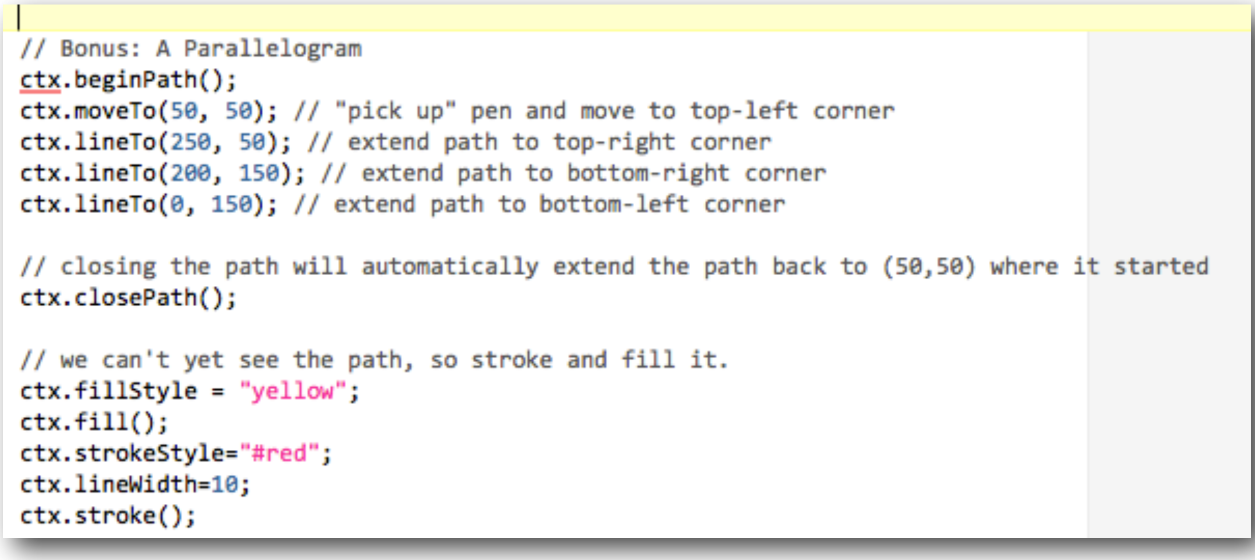
3) Read over **Paths and Subpaths**

A. You draw shapes other than rectangles by _______________

B. Calling `stroke()` or `fill()` ____________________

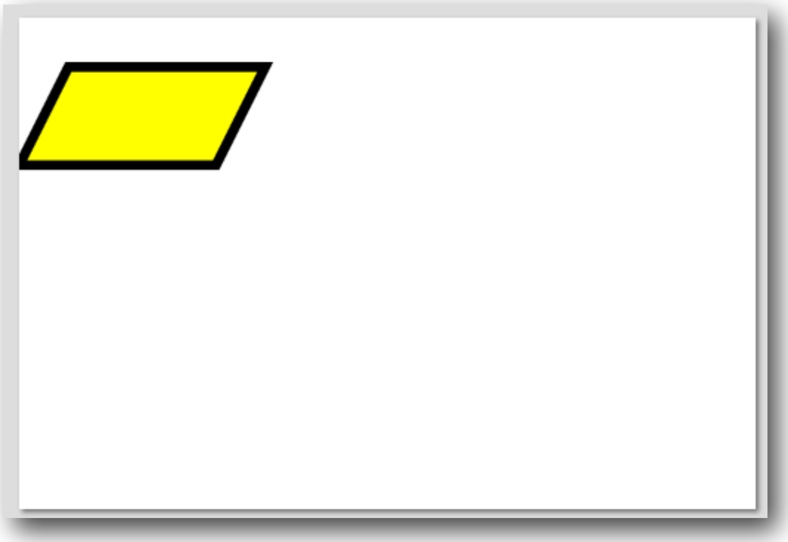
C. A path ends when you call `closePath()` or __________________________

Apple mysteriously left out illustrative example code, so here's some:



```
// Bonus: A Parallelogram
ctx.beginPath();
ctx.moveTo(50, 50); // "pick up" pen and move to top-left corner
ctx.lineTo(250, 50); // extend path to top-right corner
ctx.lineTo(200, 150); // extend path to bottom-right corner
ctx.lineTo(0, 150); // extend path to bottom-left corner

// closing the path will automatically extend the path back to (50,50) where it started
ctx.closePath();

// we can't yet see the path, so stroke and fill it.
ctx.fillStyle = "yellow";
ctx.fill();
ctx.strokeStyle="#red";
ctx.lineWidth=10;
ctx.stroke();
```

This code drew a parallelogram with side lengths of 200 and 100 units. Note that the stroke goes outside the boundaries of the path. Also note that if you change the order of the `.fill()` and `.stroke()` calls, you will get different results.



4) Skip down to **Drawing Arcs and Circles**:

`arc(x,y,radius,startAngle,endAngle)`

A) What do `x` and `y` represent? (note that this is different from the rectangle methods)

B) Are angle values specified in *degrees* or *radians*?

C) By default, which way is the arc drawn, *clockwise* or *counter-clockwise*?

5. A little review - write a line of code:

A. that sets the current fill color to green

B. that sets the current stroke color to purple

C. that sets the stroke width to 20

D. that sets the current font to `courier`

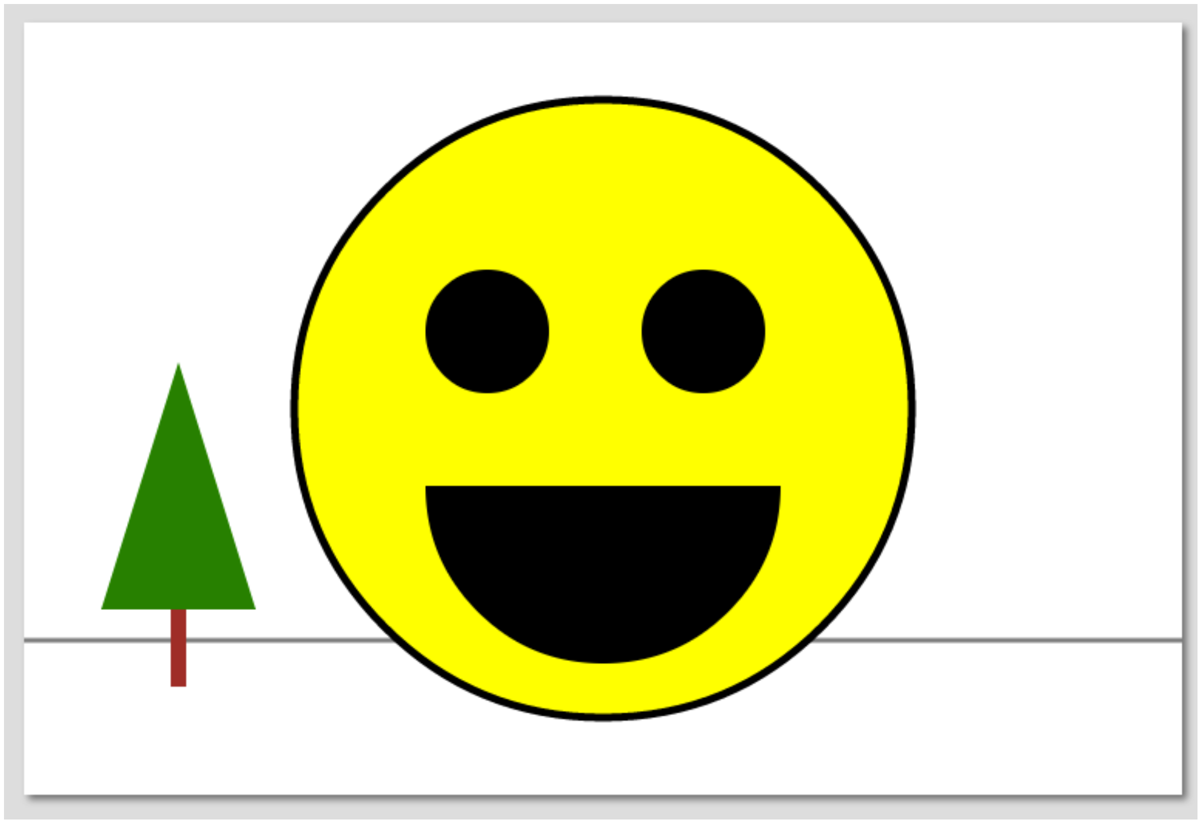
E. that clears an entire `<canvas>` that is 640 x 480

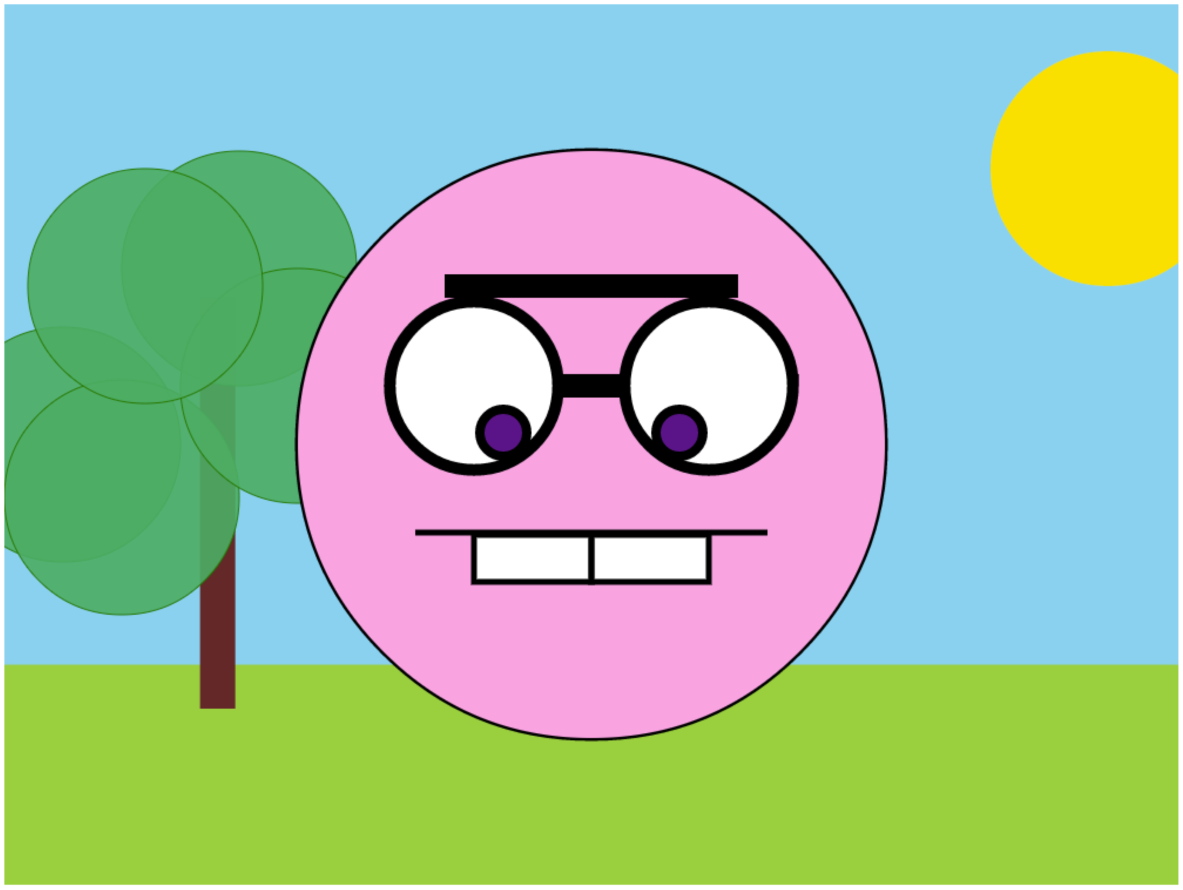
F. Lastly, where is 0,0 located in the canvas coordinate system?

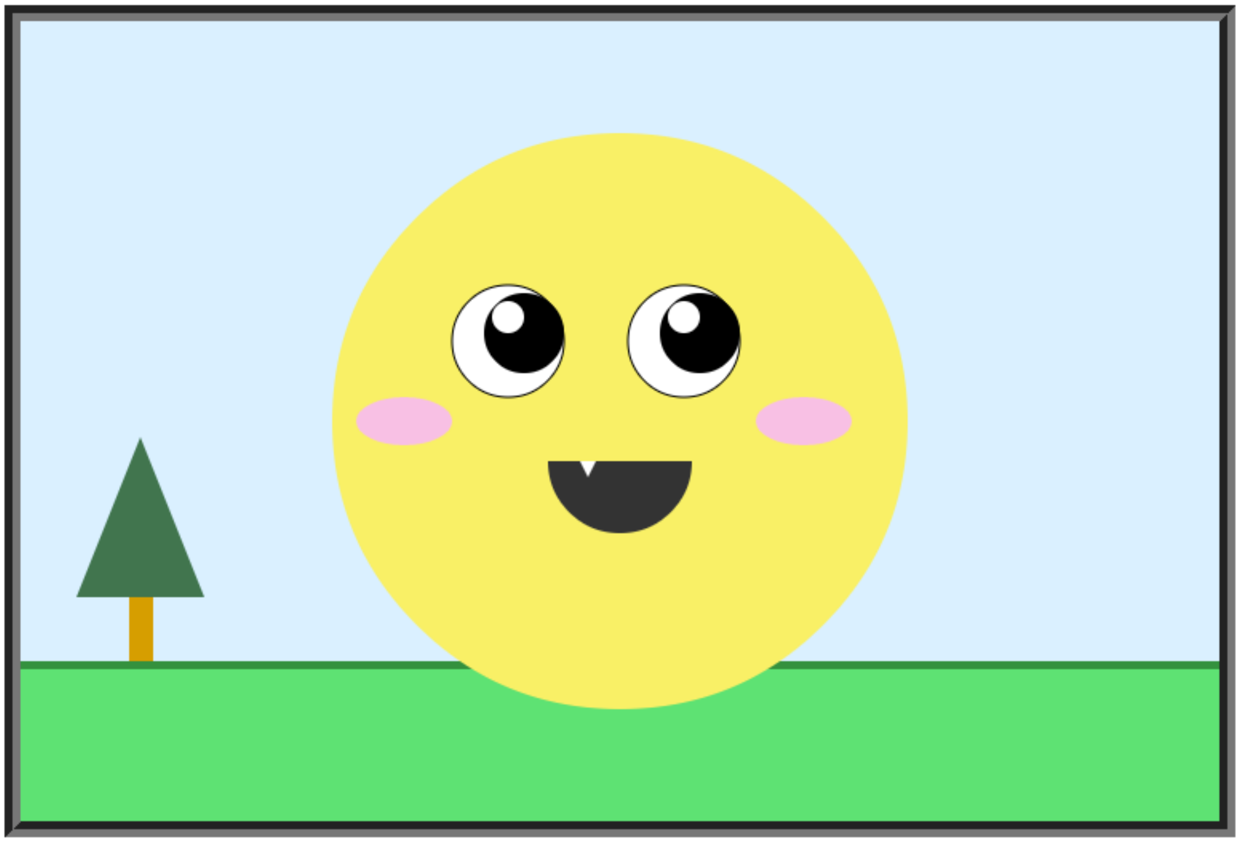
IV. Make something!

Create a scene with a “smiley face”:

- 4 arcs
- a horizon line (a line)
- and a tree (a path with 3 points plus a line for a trunk).
- The Smiley face is black and yellow. The tree is brown and green. See the first example and hints below - it doesn’t have to look exactly like mine.
- Be sure things are drawn in the right order (ex. The horizon line should be behind everything else). Canvas uses the “painters model” where new drawing is drawn on top of and obscures (or partially obscures) old drawing.







Hint I: My example canvas is 750 x 500

```
<canvas width="750" height="500">
         Get a browser that supports Canvas!
</canvas>
```

Hint II: Here's the code for the oval of the face:

```
let canvas = document.querySelector('canvas');

// get pointer to "drawing context" and drawing API
let ctx = canvas.getContext('2d');

// set state variables
ctx.fillStyle = "yellow";
ctx.strokeStyle = “black”;
ctx.lineWidth = 5;

// start drawing
ctx.beginPath();
//  ctx.arc(x,y,radius,startAngle,endAngle, clockwise)
ctx.arc(375, 250, 200, 0, Math.PI*2, false);
ctx.closePath();

// fill inside of arc with yellow
ctx.fill();

// stroke outside of arc with black
ctx.stroke();
```

Hint III: If nothing is getting drawn, check the debugger console for errors!

Other ideas:

- Make the “sky” a skyish color - - use `ctx.fillRect();`
- Make the “ground” a groundish color - - use `ctx.fillRect();`
- Other elements? see examples above for ideas