6.MITx: Day 3 PM Project

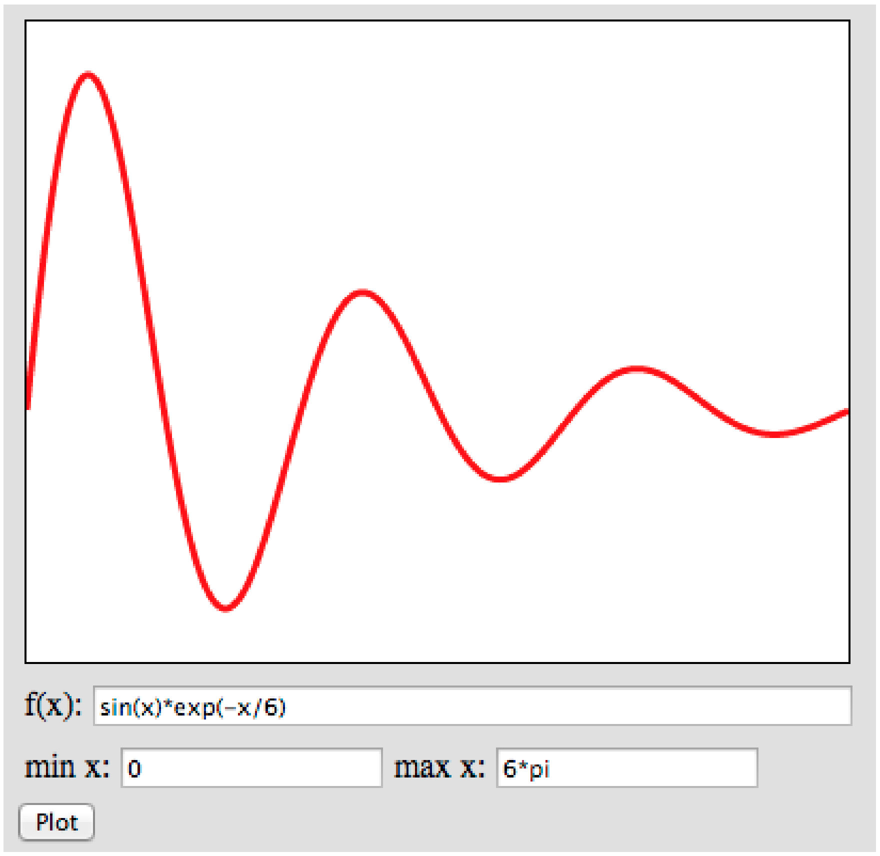
For this afternoon's project, we're asking you to hone your `<canvas>` skills by creating a graphing calculator widget. Users of your widget would add the following script line to their html:

```
<script src=”graphcalc.js”></script>
```

and then place the following div in their HTML wherever they'd like a graphic calculator widget:

```
<div class=”graphcalc”></div>
```

Your script would fill in the body of the graphcalc div with the appropriate HTML components to produce something like the following (I'm sure yours will look much more professional!):



The widget includes a canvas for displaying the plot, three expression input fields to capture the function of the variable x to be plotted, a minimum value for x and a maximum value for x, and finally a plot button to make the plot appear. The widget should calculate the value of the expression for values of x between min x and max x, then determine how to scale by the x and y values to the appropriate canvas coordinates. Note that the y scale has been chosen so the max and min points of the plot don't quite hit the edges of the canvas.

As a bit of a head start we've provided a Javascript calculator script that will evaluate arithmetic expressions with support for variables and built-in functions. To use the script include

```
<script src="http://web.mit.edu/6.mitx/www/calculator.js"></script>
```

in your HTML. This will define a `calculator` object with the following functions:

`tree = calculator.parse(string);`

returns an array representing the parse tree when the string argument is interpreted as an arithmetic expression. The function may throw an error message during parsing; you can catch the error and display the message appropriately.

`value = calculator.evaluate(tree,{name: value, …});`

`tree` is the parse tree returned by calculator.parse. The second argument is an associative array mapping variable names to numeric values, e.g., the argument `{e: Math.E, pi: Math.PI, x: 17}` would define values for the variables e, pi and x. This function returns the numeric value of the expression represented by *`tree`* or it may throw an error message, which can be caught and displayed appropriately.

Here's a rough outline of the necessary steps (see the file templates at the end of the handout):

1. Write a `setup` function that fills in a div of class graphcalc with the appropriate HTML components (canvas, labels, input fields and a Plot button). It should define a click handler function for the Plot button that calls `graph` (see below) with the canvas and three expression strings as arguments; `graph` is responsible for producing the actual plot. You should use a separate .css file to set most of the display attributes for the components.
2. Write a `graph` function, which when given a canvas and three expression strings (the function to be plotted, the min x value and the max x value) draws the appropriate curve on the canvas. This entails
   a. parsing the expression string for the function, displaying any thrown errors as text message in the center of the canvas.
   b. parsing and evaluating the expression strings for min x and max x, displaying any thrown errors as a text message in the center of the canvas.
   c. evaluating the parse tree from (a) for a range of x values, ranging from min x to max x. If the canvas is w pixels wide, a good choice would be to evaluate the parse tree for w different values for x. Save both the x and y values in arrays.
   d. determining the appropriate the formula to use for mapping the x values to horizontal canvas coordinates, i.e., mapping values in the range [xmin..xmax] to the range [0..w] where w is the width of the canvas in pixels. Do the same for the y values, remembering to expand the range slightly so that the plot won't touch the edge of the canvas (it just looks better that way!).
   e. clearing the canvas, creating a path using `moveTo` and `lineTo` that connects all the x,y points, then stroking the path with a thick red line.

Once this is working, spend your remaining time tackling the following extensions:

1. Add a tastefully chosen grid to the plot. "Tastefully chosen" means that there are a modest number of grid lines in each direction at "interesting" values. Most engineering and scientific plots have grid lines at multiples of 1, 2 or 5, scaled by some appropriate power of 10. In the example plot on the first page where the y values vary between roughly -1 and +1, the horizontal grid lines might be at y = -1, -0.5, 0, 0.5, 1. Consider adding labels in the plot for the grid lines.

2. Add support for taking measurements from the plot using the mouse. One way of doing this is to display a vertical cursor line that moves back and forth across the plot following the mouse. Then near the intersection of the vertical cursor with the plotted curve, display text giving the (x,y) value at the intersection. Note that you should use the technique described in the morning where a background image is used to hold the grid and plot and then, as the mouse moves, the canvas is repainted from the background image, then the vertical cursor and text display. This avoids recomputing/replotting the grid and plot with each mouse movement.

Templates for the .html and .js files can be found on the following page.

Good luck! Call us over and give a quick demo as you get the various pieces working...

Template for graphcalc.html

```
<html>
<head>
<title>Graphing Calculator</title>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.8.3/jquery.min.js">
</script>
<script src="http://web.mit.edu/6.mitx/www/calculator.js"></script>
<script src="graphcalc.js"></script>
<link rel="stylesheet" type="text/css" href="graphcalc.css">
</head>
<body>
<div class="graphcalc"></div>
</body>
</html>
```

Template for graphcalc.js

```
var graphcalc = (function () {
    var exports = {};  // functions,vars accessible from outside

    function graph(canvas,expression,x1,x2) {
        … your code to plot the value of expression as x varies from x1 to x2 …
    }

    function setup(div) {
        … your code to fill the div with the appropriate HTML components and
         to call graph() with the appropriate arguments in response to a click
         of the Plot button …
    }
    exports.setup = setup;

    return exports;
}());

// setup all the graphcalc divs in the document
$(document).ready(function() {
    $('.graphcalc').each(function() {
        graphcalc.setup(this);
    });
});
```