Mathsoc Programming Workshop: Week 2

Introduction

This week we will be moving on to the basics of 2d drawing.

For each Program, you have been provided with some incomplete code to get you started, and working through the exercises should tell you how to develop this into a fully working interactive web page. If at any stage you get stuck you are advised to ask others, the workshop helpers, or look for relevant materials online.

Getting the code and using c9.io

In order to complete this workshop, you will first need to create an account at `https://c9.io`, an online development environment which will allow you to edit and preview your programs in your web browser.

Once you have created your account, you should login and click on *Create a new workspace*. Here you should fill in the *Clone from Git or Mercurial URL* field as `https://github.com/twright/mathsoc-programming-week2` which will allow you to get a copy of the workshop code, and under *Choose a template* select HTML5. Now you should click *Create workspace*, and once this is done, you will be ready to start coding.

Program 3: Crayon Houses

The goal of these exercises is to draw some pretty houses.

For these exercises you will need to edit the files `houses.html` and `houses.js`.

1. Have a look at `houses.html` and `houses.js` and try to understand the code.

   You should look at some of the following online tutorials to learn a bit about HTML canvas, and fully understand the code:

   - `http://www.w3schools.com/js/default.asp`
   - `https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial`
   - `http://www.w3schools.com/canvas/`

   One thing to note especially is that in the canvas coordinate system, the vertical axis points downwards (so $(0, 0)$ refers to the top left corner of the canvas).

2. Preview the page and see what shape is displayed.

   You will notice that it is only half complete. Draw the other half (the resulting shape should be symmetric).

3. At present the `drawHouse` function is not very flexible, as it always draws the house in the same position.

   Rewrite this function so it takes two extra arguments `x` and `y` which specify the centre of the base of the house, and draws the house at this position.

   Hint: You will need to change the coordinates given in the drawing commands so they are relative to this position.

4. Use this new command to draw two houses at positions $(125, 300)$ and $(300, 175)$.

5. Now we have two houses, it just remains to give them a groovy colour scheme.

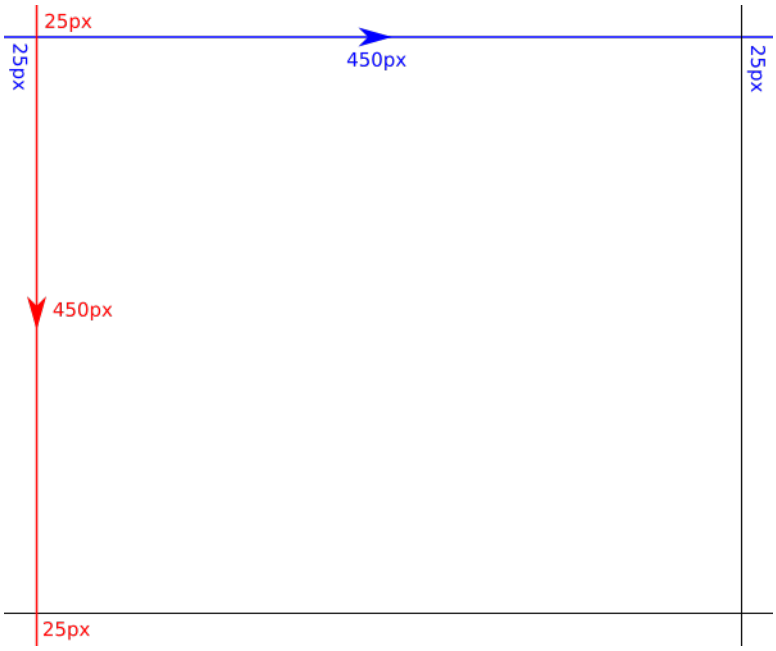
   Modify the `drawHouse` function to make it possible to specify the fill colour and stroke colour of the houses. You should make the first house red and blue, and the second house yellow and purple.

Program 4: Function Plotting

The goal of these exercises is to plot the graphs of mathematical functions.

For these exercises you will need to edit the files `function-plotting.html` and `function-plotting.js`.

For this exercise we are using a 500×600 pixel canvas, divided to include a plotting area, axes, and margins according to the following diagram.



1. Look at the code in `function-plotting.js`. You will see there is already code to draw the x-axis using the function `drawXAxis`.

   By writing a new function called `drawYAxis`, draw a y-axis as well.

2. In order to plot functions over the interval $[0, 10]$ which are assumed to take values in the interval $[0, 10]$ on the canvas, we have to convert from coordinates in $(x, y) \in [0, 10] \times [0, 10]$, to the coordinate system of the canvas. Write functions `xToPx` which converts $x \in [0, 10]$ to the corresponding x position in the canvas coordinate system, and similarly, and `yToPx` which converts $y \in [0, 10]$ to the corresponding y position.

   Use the browser console to check you get `xToPx(0) == 25`, `xToPx(5) == 300`, `xToPx(10) == 575`, `yToPx(0) == 475`, `yToPx(5) == 250`, and `yToPx(10) == 25`.

3. The following function plots the graph of the function $f(x) = x$ (that is, the line from $(0, 0)$ to $(10, 10)$):

```
function plotExample(ctx) {
    ctx.beginPath();

    ctx.strokeStyle = "red";
    ctx.lineWidth = "1.5";

    ctx.moveTo(xToPx(0), yToPx(0));
    ctx.lineTo(xToPx(10), yToPx(10));

    ctx.stroke();
```

```
    ctx.closePath();
```

add this function to your program, call it, and preview the results. Modify it to plot a piecewise linear function of your choice (make it interesting!).

4. We want to proceed to plotting arbitrary functions. In Javascript we can define a function which takes other functions as arguments, for example, the following function approximates the integral of `f` over the interval $[a, b]$ using the Trapezium Rule:

```
function trapeziumRule(f, a, b) {
    return 0.5 * (f(a) + f(b))/(b - a);
}
```

When we want to call this with a specific function we can either use an existing function defined in the normal way, or define an *anonymous function* (that is, a function without a name) inline, and call it like

```
trapeziumRule(function(x){ return x*x }, 0, 10)
```

Use these ideas to write a function `plotFunction(ctx, f)` that plots a rough approximation of the function, by plotting lines connecting its values at $x = 0, 2, 4, 6, 8, 10$.

You should try your plotting function on simple examples like $f(x) = x^2$, and more complicated like $f(x) = -0.1x^3 + x^2 + 9$.

5. This is all very nice, but it would be alot of work to plot the function to any reasonable precision in this manner. Instead we need to master another key programming concept, loops, which will make this easy.

To get you started, read this page on loops in Javascript: `http://www.w3schools.com/js/js_loop_for.asp`.

Your task is then to extend the previous `plotFunction` function to include an extra parameter, `interval`, and plots the function using its values at each of the points x_j where $x_0 = 0$ and $x_{j+1} = x_j +$ `interval` which lie inside our interval $[0, 10]$. Once you have done this, calling `plotFunction` with a small enough `interval` (say 0.1) will yield a seamless plot of the function.

Congratulations, if you have got this far this means have got through some of the basics of 2D drawing and should now have a working function plotter.

If you want to go further, you should try to extend this program, either by making the plot look nicer (some labels on the axes would be nice), or by making it more flexible, for example, by allowing the interval we plot the function over to be configured, or by automatically scaling the graph to the height of the function.

Next Week

In next week's workshop, we will combine our newly acquired 2d graphics skills with interactive elements, to make an online drawing program.