# Patterns in JavaScript

## Overview

In this lab, you'll refactor a web page that draws quadratic equations such as the following:

y = 3x2 + 6x - 3

All of the code is complete, but the structure is monolithic and tangled. You'll refactor the code so that it makes appropriate use of namespaces and other techniques, to improve the modularity of the code and to avoid polluting the global namespace. These are vital ingredients in creating robust and maintainable JavaScript code.

## Source folders

* C:\AngularDev\Student\AppxB-ManagingScope
* C:\AngularDev\Solutions\AppxB-ManagingScope

## Roadmap

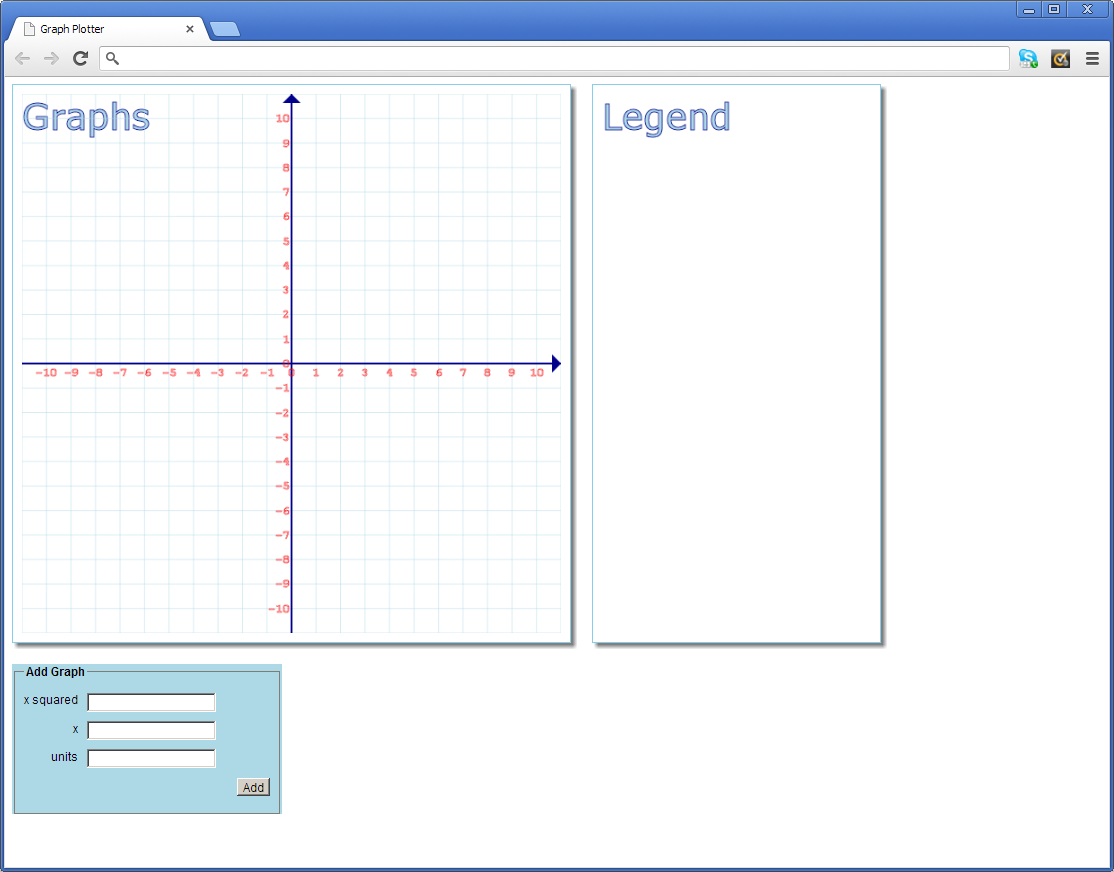
There are 3 exercises in this lab, of which the last exercise is "if time permits". Here is a brief summary of the tasks you will perform in each exercise; more detailed instructions follow later:

1. Refactor the JavaScript code into separate files
2. Creating and using namespaces
3. Protecting the global scope

## Getting started with the Student project

Go to the *Student* folder, which contains 3 complete files as follows:

* GraphPlotter.html is a web page that plots quadratic equations, based on values entered by the user.
* GraphPlotter.js is a rather complicated JavaScript file that does all the work for the web page.
* Styles.css is a simple CSS style sheet to make the web page look professional.

Open GraphPlotter.html in a browser. The web page appears as follows:

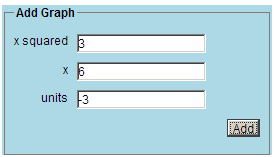
The web page contains two <canvas> elements and a <form> element:

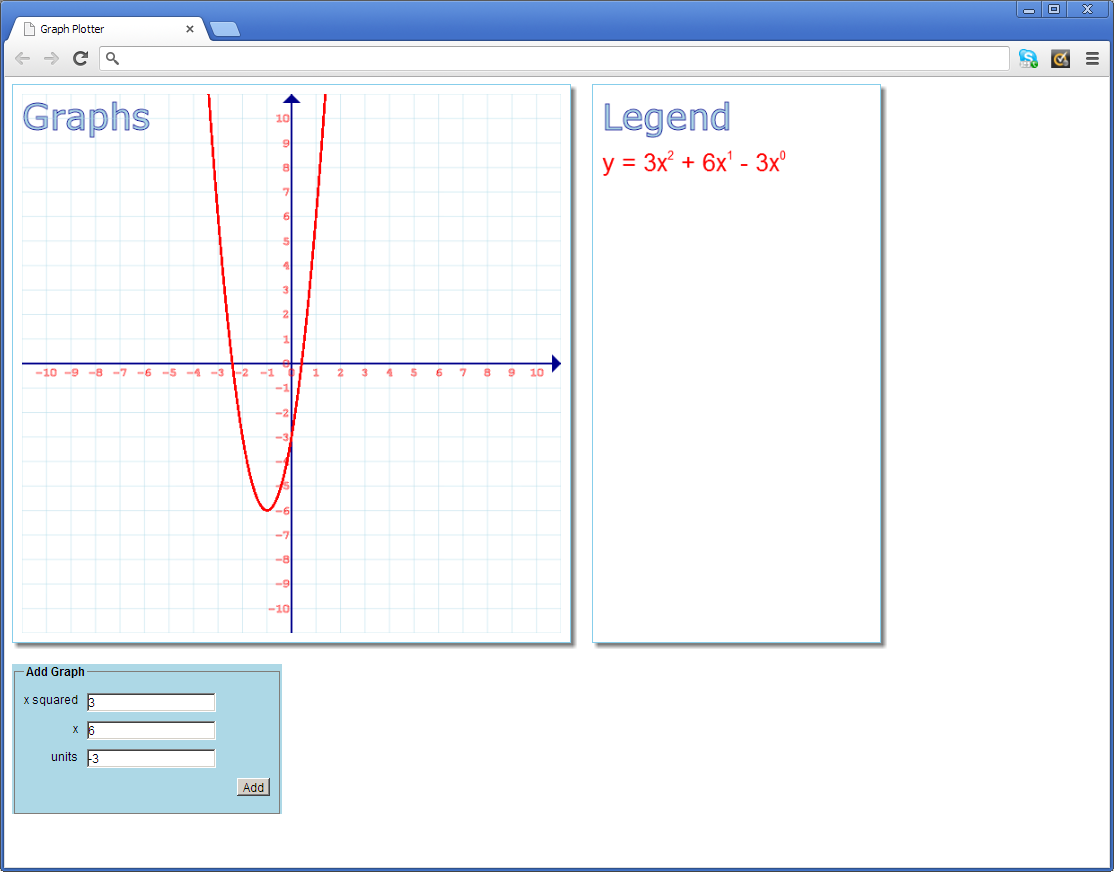
* The first canvas displays graphs for quadratic equations.
* The second canvas displays the quadratic equations textually in a legend.
* The form at the bottom allows the user to add another quadratic equation.

The following page describes how to use the web page.

The form at the bottom of the web page invites you to enter the coefficients for a quadratic equation. For example, imagine you want to plot the following equation:

y = 3x2 + 6x - 3

To plot this quadratic equation, enter the following values in the form:

When you click *Add*, the web page plots the graph in the first canvas and displays the equation textually in the second canvas:

You can add any number of quadratic equations. The first canvas plots all the equations graphically, and the second canvas shows all the equations textually. Each equation is displayed in a different colour (red, orange, gold, green, blue, indigo, and violet). If you add more than 7 equations, the colour scheme wraps around to red again ☺.

## Exercise 1: Refactor the JavaScript code into separate files

At the moment, all the JavaScript code is located in GraphPlotter.js. Take another look at this file – it's not important that you understand all the code, but note there are 3 distinct sections in the file:

* First, we define a Graph constructor to represent a graph for one equation.
* Next, we define an object named state that contains various properties describing the current application state (e.g. the width and height of the graph context, a collection of Graph objects, etc.)
* The rest of the file is a series of functions that handle events (i.e. window loaded and "add" button clicked) and that perform various drawing operations.

Use any of the patterns that we have been introduced to if you think they would be beneficial.

Split GraphPlotter.js into 3 separate JavaScript files, with the following suggested names:

* GraphModel.js - Should contain the Graph constructor definition.
* GraphState.js - Should define the state object.
* GraphUI.js - Should contain the event handler hook-ups, plus all the functions.

Then modify GraphPlotter.html so that it includes these 3 separate JavaScript files, rather than just including GraphPlotter.js (in fact, you can delete GraphPlotter.js if you like).

Test that GraphPlotter.html still works… in an ideal world you'd have test suite to run to verify everything is still OK, but in the absence of these tests the next best way is to actually open the web page in a browser and try it out for yourself.

## Exercise 2: Creating and using namespaces

The code in GraphModel.js and GraphState.js defines global identifiers (i.e. the Graph constructor and the state object respectively). It would be much better to enclose these definitions in a namespace, a) to emphasize their purpose, and b) to avoid possible name clashes with items in other libraries.

You can make use of the createNamespace() function that we showed during the chapter – so copy CreateNamespace.js in from the *Demo* folder, and include this JavaScript file in your web page, GraphPlotter.html.

Edit GraphModel.js so that it defines a namespace as follows:

* At the top of the file, call createNamespace() to create a namespace such as "osl.graphplotter" (for example).
* Modify the definition of the Graph variable, so that rather than being a free global variable, it is actually a property in the osl.graphplotter namespace. Here's the kind of code you need:

osl.graphplotter.Graph = function … … …

Now edit GraphState.js so that it also defines a namespace in a similar kind of way:

* Call createNamespace() to create a namespace named "osl.graphplotter" here too (remember, createNamespace() is resilient enough to cope with namespaces that span multiple JavaScript files).
* Modify the definition of the state variable so that it's part of the osl.graphplotter namespace.

Now that Graph and state are in the osl.graphplotter namespace, much of the code in GraphUI.js is broken (this code is expecting global Graph and state items). How do you fix this problem?

* One approach would be to make a bunch of edits in GraphUI.js so that every reference to Graph is modified to osl.graphplotter.Graph, and every reference to state is modified to osl.graphplotter.state. This would be a nightmare!
* A much better approach is to simply define aliases named Graph and state, which represent osl.graphplotter.Graph and osl.graphplotter.state respectively. Then you don't need to edit the rest of the code at all!

Save all your work and refresh GraphPlotter.html in the browser. It should all still work fine…

## Exercise 3 (If time permits): Protecting the global scope

There's a lot of code in GraphUI.js. Worse still, all these functions are currently in the global namespace, so they are ripe for name clashes with any other functions you might use in other libraries.

Think about the best strategy for protecting the global scope, so that these functions are localized and will not conflict with other functions of the same name elsewhere in the application.