08240 2D Computer Graphics: Lab1

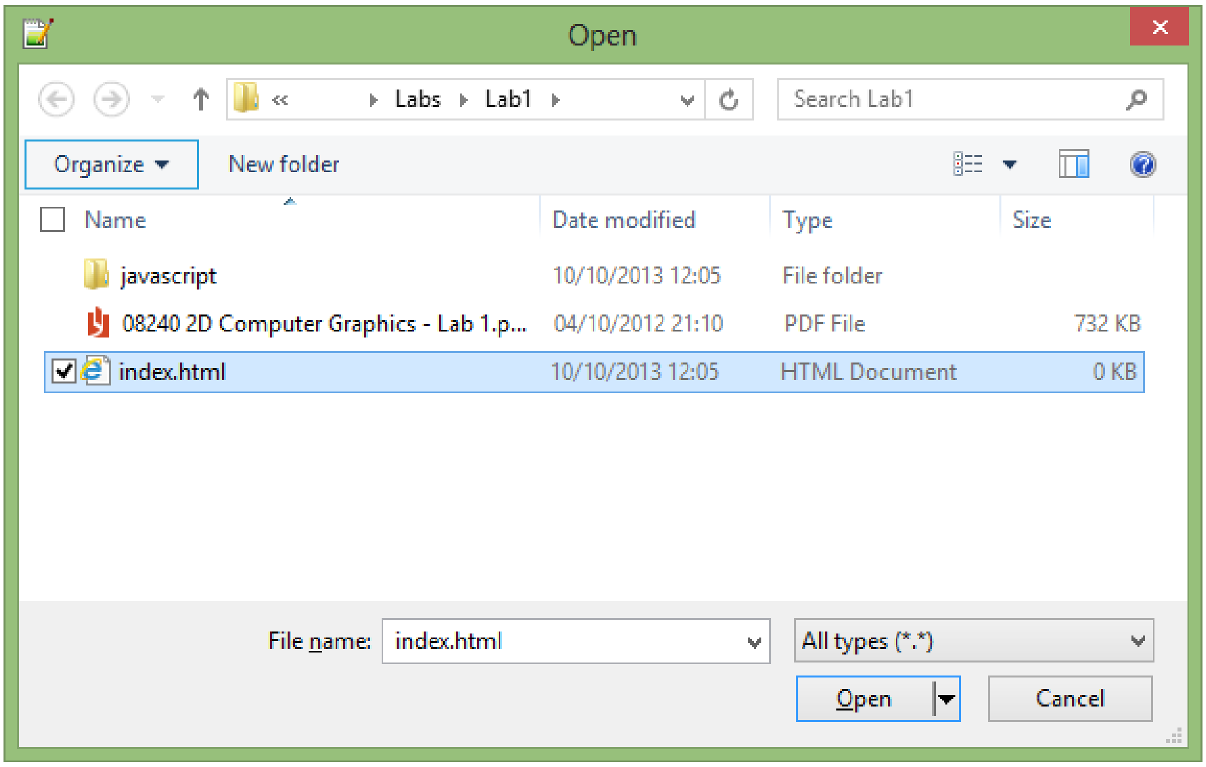
Aim

The aim of this lab is to give a brief introduction to drawing on the HTML5 Canvas as well as the development environment and source control systems that you will be using. If you haven't already done so, then please follow the instructions to checkout your repository folder (08240 2D Computer Graphics - SVN checkout.pdf).

Task 1 Editing your HTML file

1) Locate and run the Notepad++ application.

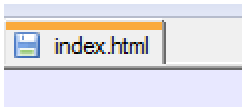
2) Click on the 'Open' icon. This opens up a file dialog.
3) Use the dialog to navigate to the new Lab1 folder that was created as part of the SVN export process you did earlier. Select the 'index.html' file and click the 'Open' button.

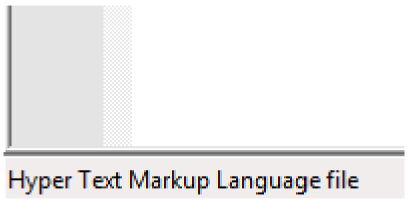


☐ Name	Date modified	Type	Size
javascript	10/10/2013 12:05	File folder	
08240 2D Computer Graphics - Lab 1.p...	04/10/2012 21:10	PDF File	732 KB
☑ index.html	10/10/2013 12:05	HTML Document	0 KB

File name: index.html
All types (*.*)

Notice that the file tab has changed to 'index.html' and the file type shown in the bottom left has changed to 'Hyper Text Markup Language file'.





This is nice because if the file type is set correctly the Notepad++ will provide some nice text formatting to make writing our code easier.

Dealing with Internet Explorer

So now we have an empty HTML file. The first thing that we want to do is side step a quirk in Internet Explorer that will allow it to render our HTML5 tags correctly.

1) Add the following tag to the top of the file:

```
<!doctype html>
```

Setting up the Canvas

Before we can create our canvas we need somewhere to put it.

1) Start with the top level <html> tags.

```
<!doctype html>
<html>
</html>
```

2) Add the <head> tags.

```
<!doctype html>
<html>
    <head>
    </head>
</html>
```

3) We want our page to have a title, so we need to add <title> tags. Everything between the open <title> and close</title> tags will appear as the title of our page. In this case use 'Lab1'.

```
<!doctype html>
<html>
    <head>
        <title>Lab1</title>
    </head>
</html>
```

4) The main content of our page exists between the <body> tags.

```
<!doctype html>
<html>
    <head>
        <title>Lab1</title>
    </head>
    <body>
    </body>
</html>
```

5) At last it is time for the <canvas> tag.

```
<!doctype html>
<html>
    <head>
        <title>Lab1</title>
    </head>
    <body>
        <canvas id="canvas" width="800" height="600">
            Your browser does not support the canvas tag
        </canvas>
    </body>
</html>
```

There are several important elements to this tag. The first are the attributes that we set in the opening tag (id, width, and height). We have set `id="canvas"` so that we have an identifier that we can use later to play with the canvas. `width="800" height="600"` is fairly self-explanatory and determines the dimensions of the canvas at load time.

The other is the text that we have entered between the opening <canvas> and closing </canvas> tags. This text will appear if the HTML page is viewed in a browser that does not support the canvas tag. For example, if you forget to set the doctype then Internet Explorer will ignore the canvas tags and display the text instead.

6) Next we will add a little bit of text below the canvas so you can give a description of the lab. Type the description between paragraph <p> tags.

```
<!doctype html>
<html>
    <head>
        <title>Lab1</title>
    </head>
    <body>
        <canvas id="canvas" width="800" height="600">
            Your browser does not support the canvas tag
        </canvas>
        <p>This is a simple 'Hello World' lab tutorial 
using the HTML5 canvas.</p>
    </body>
</html>
```

We are almost finished with the HTML file now. The last thing we need to add is a link to the JavaScript file that will contain the code for drawing on our canvas.

7) Add a <script> tag under the <title> tag in the <head> section.

```
<!doctype html>
<html>
    <head>
        <title>Lab1</title>
        <script type="text/javascript"
src="javascript/canvas.js"></script>
    </head>
    <body>
        <canvas id="canvas" width="800" height="600">
            Your browser does not support the canvas tag
        </canvas>
        <p>This is a simple 'Hello World' lab tutorial
using the HTML5 canvas.</p>
    </body>
</html>
```

Notice the <script> attributes. `type="text/JavaScript"` tells us that the script language is JavaScript and the `src="canvas.js"` attribute gives the relative path to the source file which we will create next in the JavaScript folder that we created earlier.

Commit to SVN

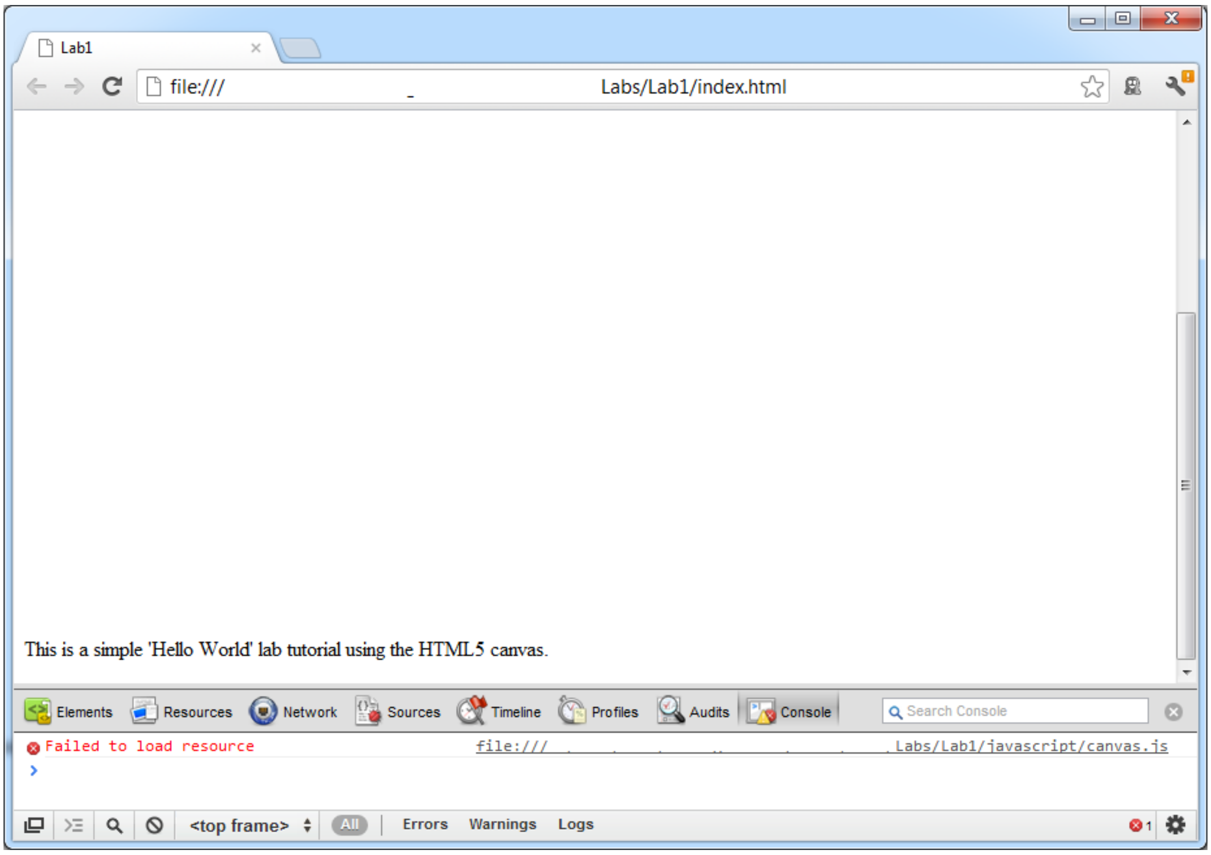
Now that you have reached a milestone in your lab (creating the HTML file) it is a good time to commit your work to SVN. Remember to select your new ‘index.html’ file and then commit it with a meaningful log message that describes the changes you made. **Please start the commit log message with “L1T1” on the first line to note that it is the completion of Lab1 Task1.**

The browser debugger

This would be a good point to introduce the browser debugger. First, save your work and then use the Windows file explorer to locate the index.html file that you created in the previous section and open it in Chrome (you may need to use the ‘Open with’ context menu if it is not the default browser.

You should see the description paragraph text from your HTML file some way down the page. This is because it is below the empty canvas area. If you press the ‘F12’ function key then this opens the debugging pane. There are a lot of useful functions hidden away in the debugger but want we want for the moment is the ‘Console’ tab.

The ‘Console’ tab will show any errors in our code. There should be one in there as we specified that we would be using ‘JavaScript/canvas.js’ as a script source but it doesn’t exist yet. In the screenshot below it shows as ‘Failed to load resource’ in red text along with the resource path it was trying to load in grey on the right.



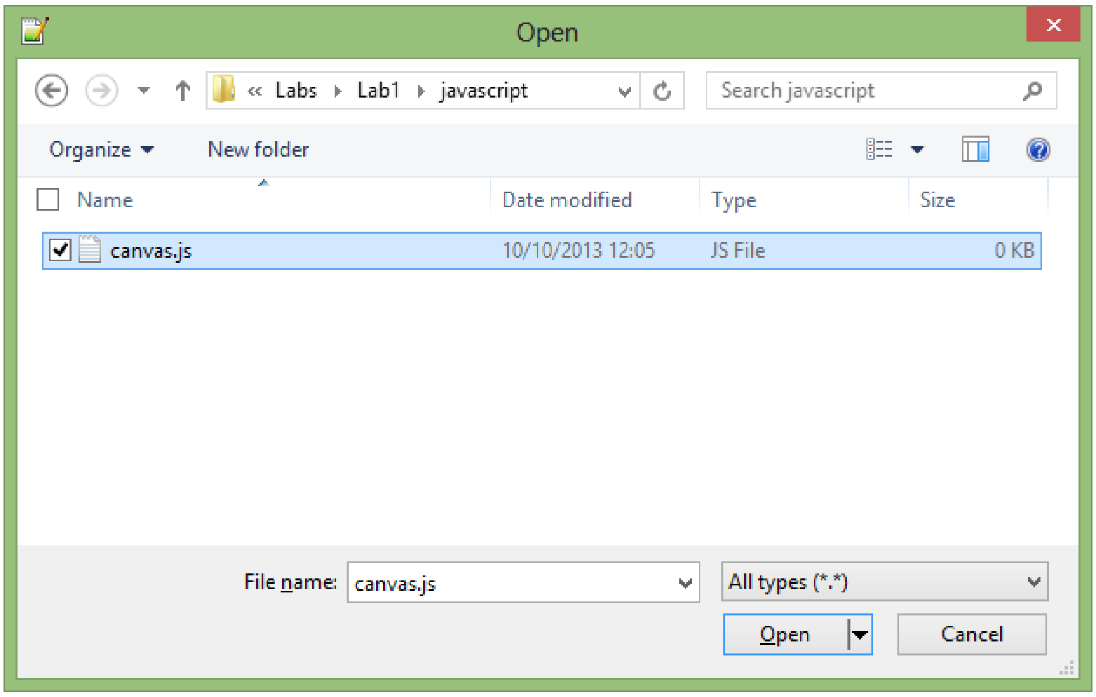
This is a simple 'Hello World' lab tutorial using the HTML5 canvas.

```
Failed to load resource                    file:///                              Labs/Lab1/javascript/canvas.js
```

We will be using the debugger a lot more once we start writing the JavaScript code and drawing on the canvas.

Task 2 Initialising our canvas.js file

Follow the same procedure as before for opening a file in Notepad++, but this time open the 'canvas.js' file from the 'javascript' folder in your 'Lab1' folder.

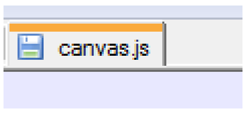


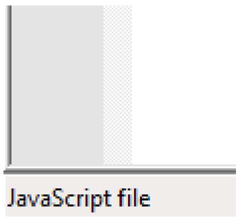
Name	Date modified	Type	Size
canvas.js	10/10/2013 12:05	JS File	0 KB

File name: canvas.js

All types (*.*)

You will notice the tab file name and the file type in the lower left of the Notepad++ window:

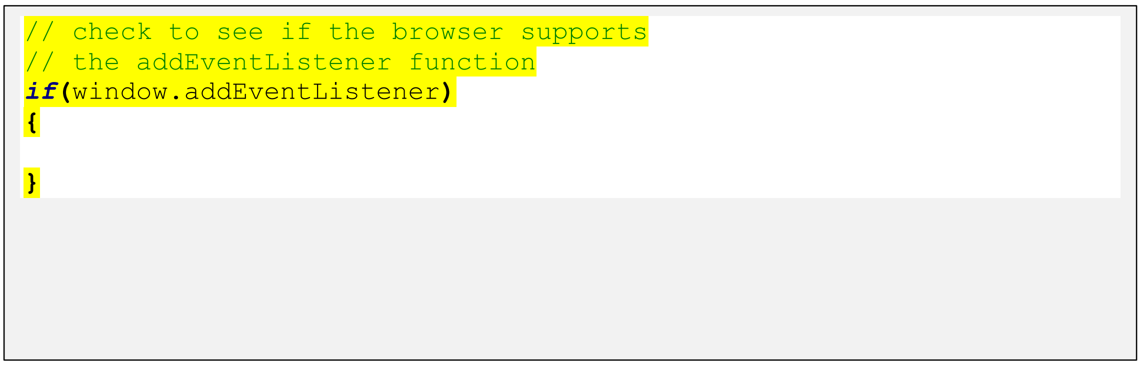




Coding the JavaScript

Now we have an empty JavaScript file it is time to start coding.

1) First we will check to see if the window object supports the addEventListener function



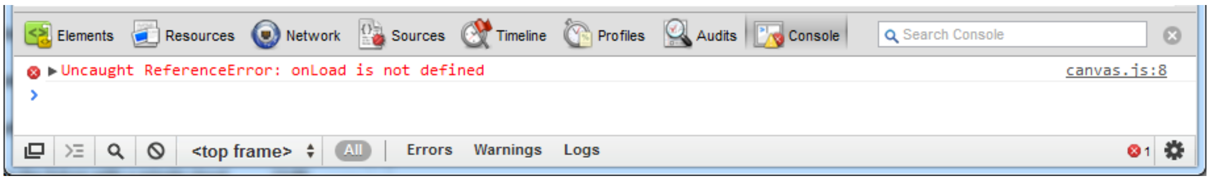
```
// check to see if the browser supports
// the addEventListener function
if(window.addEventListener)
{

}
```

2) Having determined that the function does exist, we call it with three parameters. The first parameter is the name of the event that we are listening for, in this case the 'load' event

that will fire when the page loads. The second is the name of the event handler function that will be called when the event is detected; we will have to write this function next. The final parameter is a Boolean value that determines whether the event is dealt with during the capture phase or the bubbling phase of the event propagation. For our purposes this is not very important, but if you want to know more check out this link: http://www.w3.org/TR/2003/NOTE-DOM-Level-3-Events-20031107/events.html#Events-phases

```
// check to see if the browser supports
// the addEventListener function
if(window.addEventListener)
{
    window.addEventListener
    (
        'load', // this is the load event
        onLoad, // this is the event handler we are going to write
        false   // useCapture boolean value
    );
}
```

3) Now save your work and open the HTML file in your browser again with the debugger. You should notice that there is an error because the event handler 'onLoad' that we specified as the second parameter of our addEventListener function has not been written yet. It also shows you on what line of the JavaScript file the error was caught (Line 8).



```
▶Uncaught ReferenceError: onLoad is not defined                    canvas.js:8
```

4) So let's add that event handler function. All of the rest of the code is going to go in this function.

```
// check to see if the browser supports
// the addEventListener function
if(window.addEventListener)
{
    window.addEventListener
    (
        'load', // this is the load event
        onLoad, // this is the event handler we are going to write
        false   // useCapture boolean value
    );
}

// the window load event handler
function onLoad()
{

}
```

5) Next we are going to define two variables so we can easily refer to the canvas and its drawing context. Variables in JavaScript are weakly typed, which means that you just define a variable with a name and you can put whatever you want in it.

```
// the window load event handler
function onLoad()
{
    var canvas;
    var context;
}
```

6) Now we need a function to initialise those variables.

```
// the window load event handler
function onLoad()
{
    var canvas;
    var context;

    // this function will initialise our variables
    function initialise()
    {

    }
}
```

7) We are going to query the HTML page's document object using the 'getElementById' function to get hold of the canvas object. If the element with the ID 'canvas' exists then the canvas variable will now point to it. Otherwise a message box will pop up to alert us that it could not be found.

```
    // this function will initialise our variables
    function initialise ()
    {
        // Find the canvas element using its id attribute.
        canvas = document.getElementById('canvas');
        // if it couldn't be found
        if (!canvas)
        {
            // make a message box pop up with the error.
            alert('Error: I cannot find the canvas element!');
            return;
        }
    }
```

8) Now we need the canvas' context. First we check if the canvas object has a 'getContext' function and throw an error message box if it does not.

```
// this function will initialise our variables
function initialise ()
{
    // Find the canvas element using its id attribute.
    canvas = document.getElementById('canvas');
    // if it couldn't be found
    if (!canvas)
    {
        // make a message box pop up with the error.
        alert('Error: I cannot find the canvas element!');
        return;
    }
    // check if there is a getContext function
    if (!canvas.getContext)
    {
        // make a message box pop up with the error.
        alert('Error: no canvas.getContext!');
        return;
    }
}
```

9) That worked so now we use the function to set the context variable to the canvas' context object. If that did not work then pop up a message box saying so.

```
// this function will initialise our variables
function initialise ()
{
    // Find the canvas element using its id attribute.
    canvas = document.getElementById('canvas');
    // if it couldn't be found
    if (!canvas)
    {
        // make a message box pop up with the error.
        alert('Error: I cannot find the canvas element!');
        return;
    }
    // check if there is a getContext function
    if (!canvas.getContext)
    {
        // make a message box pop up with the error.
        alert('Error: no canvas.getContext!');
        return;
    }
    // Get the 2D canvas context.
    context = canvas.getContext('2d');
    if (!context)
    {
        alert('Error: failed to getContext!');
        return;
    }
}
```

10) We have reached another milestone. Commit your changes to the 'canvas.js' file with an appropriate log message starting with "L1T2" on the first line.

Task 3 Adding the draw function

11) Almost at the end now. Next we will define the ‘draw’ function that will actually draw on the canvas.

```
// the window load event handler
function onLoad()
{
    var canvas;
    var context;

    // this function will initialise our variables
    function initialise ()
    {
        // Find the canvas element using its id attribute.
        canvas = document.getElementById('canvas');
        // if it couldn't be found
        if (!canvas)
        {
            // make a message box pop up with the error.
            alert('Error: I cannot find the canvas element!');
            return;
        }
        // check if there is a getContext function
        if (!canvas.getContext)
        {
            // make a message box pop up with the error.
            alert('Error: no canvas.getContext!');
            return;
        }
        // Get the 2D canvas context.
        context = canvas.getContext('2d');
        if (!context)
        {
            alert('Error: failed to getContext!');
            return;
        }
    }

    // this function will actually draw on the canvas
    function draw()
    {

    }
}
```

12) First we are going to set the colour of our drawing fill style to black. It is worth noting here that we are defining the colour in RGB hex. The hash character ‘#’ denotes that we are specifying hex and the hex value itself is three pairs of values, one for each of Red, Green, and Blue. By specifying ‘#000000’ we are saying that we want our colour to have no red, no green, and no blue, i.e. black. More information can be found here: http://www.w3schools.com/html/html_colors.asp

```
    // this function will actually draw on the canvas
    function draw()
    {
        // set the draw fill style colour to black
        context.fillStyle = "#000000";
    }
```

13) Now we have a colour we should do something with it. I see a canvas and I want it painted black. Use the 'fillRect' function of the context object to paint a rectangle of the previously specified colour. Its parameters are an X, Y starting position coordinate pair and a width, height pair to define the rectangle's dimensions. As we want to fill the canvas start the rectangle at 0,0 and make it the width and height of the canvas object. It should be noted that the X, Y position is relative to the top left of the canvas.

```
    // this function will actually draw on the canvas
    function draw()
    {
        // set the draw fill style colour to black
        context.fillStyle = "#000000";
        // fill the canvas with black
        context.fillRect(0,0,canvas.width,canvas.height);
    }
```

14) That gave us a nice black background for our canvas. Now we want to write our message. First choose a font.

```
    // this function will actually draw on the canvas
    function draw()
    {
        // set the draw fill style colour to black
        context.fillStyle = "#000000";
        // fill the canvas with black
        context.fillRect(0,0,canvas.width,canvas.height);
        // choose a font for our message
        context.font = "40pt Calibri";
    }
```

15) Set the draw fill style to white (#ffffff).

```
    // this function will actually draw on the canvas
    function draw()
    {
        // set the draw fill style colour to black
        context.fillStyle = "#000000";
        // fill the canvas with black
        context.fillRect(0,0,canvas.width,canvas.height);
        // choose a font for our message
        context.font = "40pt Calibri";
        // set the draw fill colour to white
        context.fillStyle = "#ffffff";
    }
```

16) Use the 'fillText' function of the context object. It has 3 parameters: the string that you want to write "Hello World!", and the X and Y coordinate pair position that you want to write it.

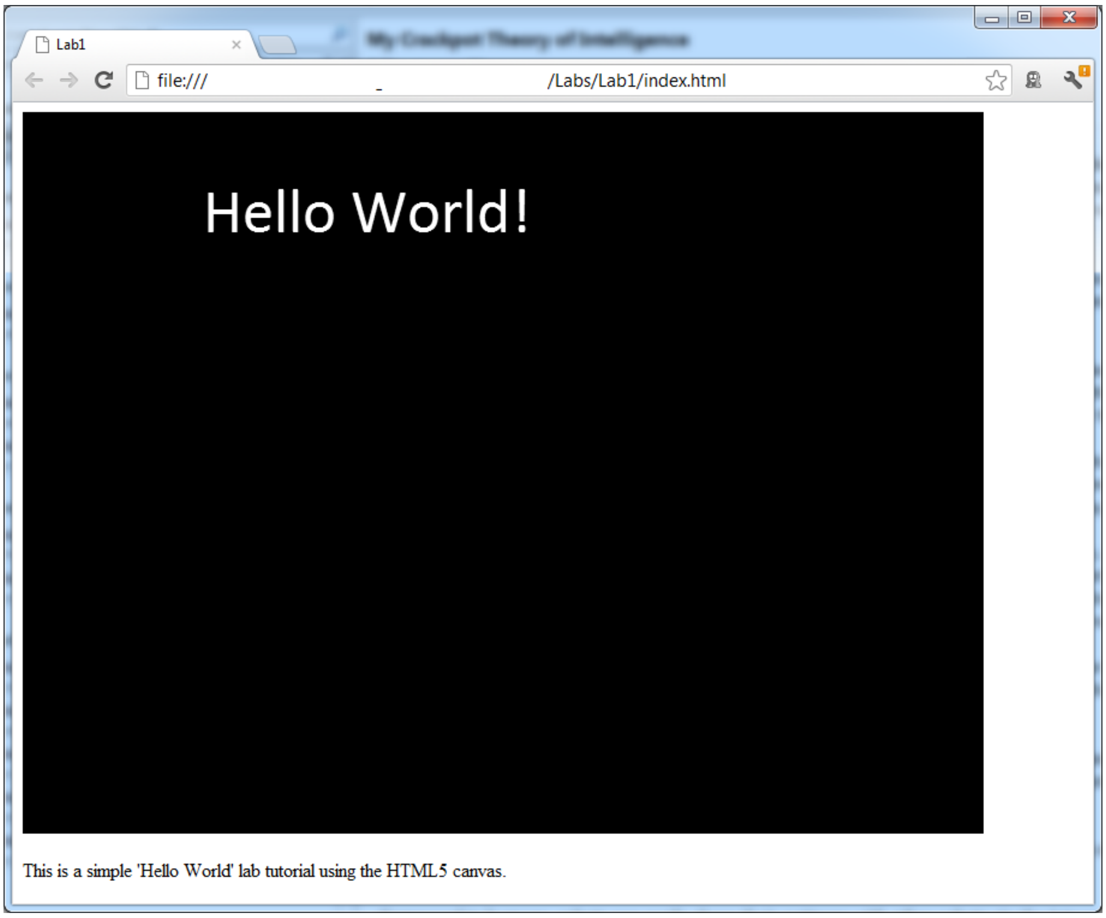
The position specifies the bottom left corner of the text that will appear on the canvas as opposed to the top left of the canvas for the ‘fillRect’ function.

```
    // this function will actually draw on the canvas
    function draw()
    {
        // set the draw fill style colour to black
        context.fillStyle = "#000000";
        // fill the canvas with black
        context.fillRect(0,0,canvas.width,canvas.height);
        // choose a font for our message
        context.font = "40pt Calibri";
        // set the draw fill colour to white
        context.fillStyle = "#ffffff";
        // draw the text at the specified position
        context.fillText("Hello World!", 150, 100);
    }
```

17) Just one more thing. We need to call the ‘initialise’ and ‘draw’ functions that we just finished writing.

```
// the window load event handler
function onLoad()
{
    var canvas;
    var context;

    // this function will initialise our variables
    function initialise ()
    {
        // Find the canvas element using its id attribute.
        canvas = document.getElementById('canvas');
        // if it couldn't be found
        if (!canvas)
        {
            // make a message box pop up with the error.
            alert('Error: I cannot find the canvas element!');
            return;
        }
        // check if there is a getContext function
        if (!canvas.getContext)
        {
            // make a message box pop up with the error.
            alert('Error: no canvas.getContext!');
            return;
        }
        // Get the 2D canvas context.
        context = canvas.getContext('2d');
        if (!context)
        {
            alert('Error: failed to getContext!');
            return;
        }
    }

    // this function will actually draw on the canvas
    function draw()
    {
        // set the draw fill style colour to black
        context.fillStyle = "#000000";
        // fill the canvas with black
        context.fillRect(0,0,canvas.width,canvas.height);
        // choose a font for our message
        context.font = "40pt Calibri";
        // set the draw fill colour to white
        context.fillStyle = "#ffffff";
        // draw the text at the specified position
        context.fillText("Hello World!", 150, canvas.height);
    }

    // call the initialise and draw functions
    initialise();
    draw();
}
```

18) Ta-daaaa!



Hello World!

This is a simple 'Hello World' lab tutorial using the HTML5 canvas.

Commit to SVN

Now that you have reached another milestone in your lab it is a good time to commit your work to SVN. **Commit your changes with a meaningful log message starting with "L1T3" on the first line.**

Hang on, can't I use those cool vectors we talked about?

Now that you mention it, yes we can. In fact that is what we are going to do next; starting with creating our vector object.

Task 4 – Set up the 'vector.js' reference in the 'index.html' file

Since we are going to be calling code from this file we need to add it to the list of scripts in the head of the html file.

```
<head>
   <title>Lab1</title>
   <script type="text/javascript" src="javascript/canvas.js"></script>
   <script type="text/javascript" src="javascript/vector.js"></script>
</head>
```

We have finished with the html file now so go ahead and commit it.

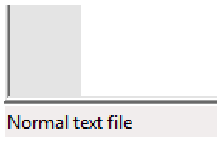
Commit to SVN

Now that you have reached another milestone in your lab (adding the vector.js reference) it is a good time to commit your work to SVN. Don't forget to add a meaningful log message **starting with the tag 'L2T4' on the first line on the log entry**.

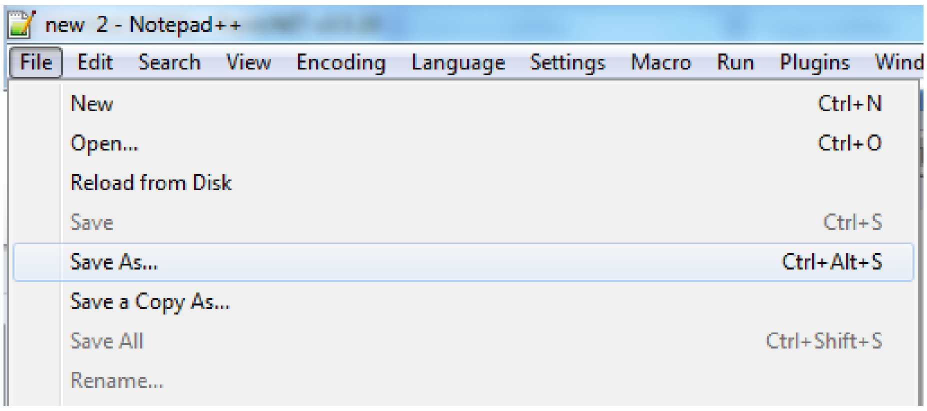
Task 5 – Create the 'vector.js' file

1) Click on the 'New file' icon. This opens up a blank page.

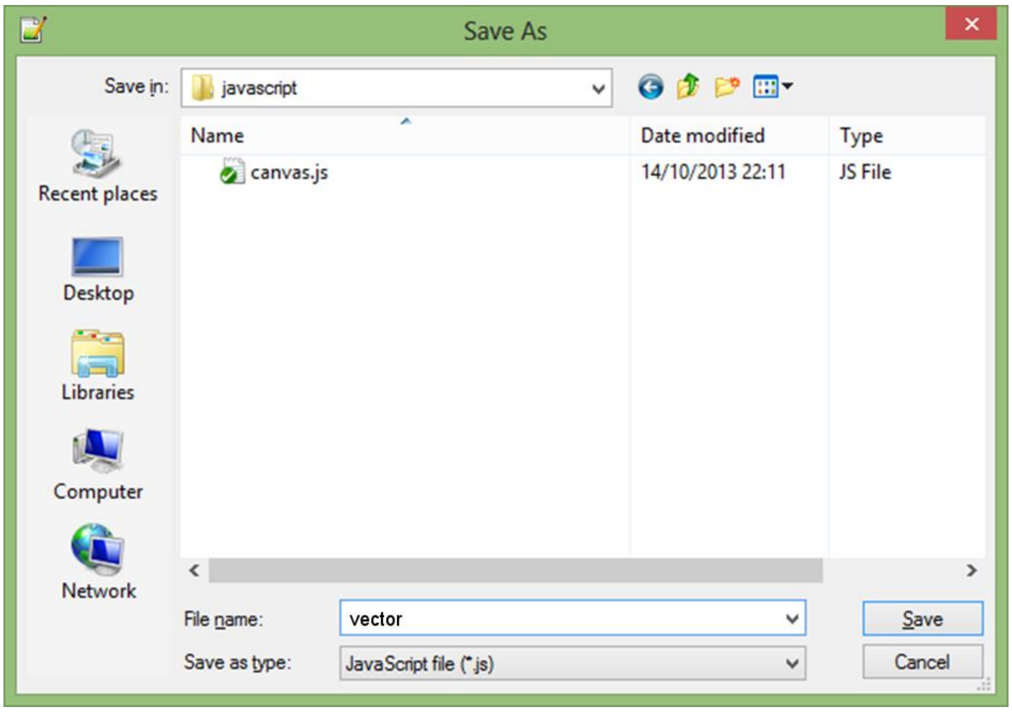
   At the moment, by default, this is a normal text file; you can see this in the bottom left of the Notepad++ window. What we want though is a JavaScript (*.js) file.



2) Using the menu, click File > Save As...



3) Select the JavaScript folder in your Lab1 folder and set the 'Save as type' to 'JavaScript file'. Type 'vector' in to the 'File name' field and click on the 'Save' button.

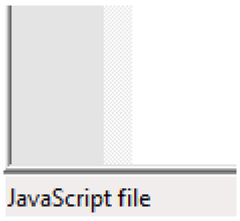


Name	Date modified	Type
canvas.js	14/10/2013 22:11	JS File

File name: vector

Save as type: JavaScript file (*.js)

You will notice the file type in the lower left of the Notepad++ window:



Commit to SVN

Now that you have reached another milestone in your lab (creating the vector.js file) it is a good time to commit your work to SVN. You will need to 'add' the vector.js file to SVN first. Don't forget to add a meaningful log message **starting with the tag 'L2T5' on the first line on the log entry**.

Task 6 – Writing our vector object definition

Next we are going to define the Vector object in the vector.js file starting with a constructor.

```
var Vector = (function () {
    function Vector(pX, pY) {

    };

    return Vector;
})();
```

In the above code we have defined an empty constructor that returns a Vector object.

Commit to SVN

Now that you have reached another milestone in your lab (defining our vector object) it is a good time to commit your work to SVN. Don't forget to add a meaningful log message **starting with the tag 'L2T6' on the first line on the log entry**.

Task 7 – Adding getters and setters to our vector object

We also need some functions so we can actually use our vector, starting with a getter and setter for the x coordinate of our vector.

JavaScript uses a concept called prototypes to allow for inheritance between objects. It is not important to understand the details of this now, but if you are interested then you can try Googling 'javascript prototype'. For now just know that all objects have prototypes and you can add functions to them like below.

Note that I am using a 'p' at the start of my variables where they are a parameter (p for parameter) and an 'm' where the variable is a member variable of the object (m for member). This just makes it easier to distinguish them.

```
var Vector = (function () {
    function Vector(pX, pY) {

    };
    Vector.prototype.getX = function() {

        return this.mX;
    };
    Vector.prototype.setX = function (pX) {
        this.mX = pX;
    };

    return Vector;
})();
```

Now that we have a get and set function for x we can use it in the constructor to store the x coordinate parameter.

```
function Vector(pX, pY) {
   this.setX(pX);
};
```

It is important to note the use of 'this' when calling the functions of the vector object. In C# the use of the 'this' keyword can be implied, meaning that you can omit it. Ie. 'setX(pX)' and 'this.setX(pX)' are the same thing. This is not the case in JavaScript so don't forget your 'this'.

Repeat the process for the 'pY' parameter.

Commit to SVN

Now that you have reached another milestone in your lab (added getter and setter functions) it is a good time to commit your work to SVN. Don't forget to add a meaningful log message **starting with the tag 'L2T7' on the first line on the log entry**.

Task 8 – Using our vector for the position of the text

To use our vector in the draw function we want to add a parameter for the position of the text. Add the parameter 'pPosition'. This is going to be a vector so we can use its getX and getY functions to extract the x and y coordinates in our fillText method.

```
    // this function will actually draw on the canvas
    function draw(pPosition)
    {
      // set the draw fill style colour to black
      context.fillStyle = "#000000";
      // fill the canvas with black
      context.fillRect(0,0,canvas.width,canvas.height);
      // choose a font for our message
      context.font = "40pt Calibri";
      // set the draw fill colour to white
      context.fillStyle = "#ffffff";
      // draw the text at the specified position
      context.fillText("Hello World!", pPosition.getX(), pPosition.getY());
    }
```

Commit to SVN

Now that you have reached another milestone in your lab (used a vector to position the text) it is a good time to commit your work to SVN. Don't forget to add a meaningful log message **starting with the tag 'L2T8' on the first line on the log entry**.

Task 9 – Instantiating and using our position vector

The draw function can now use a vector, but it is undefined so we need to actually instantiate one to use when we call the function. Back in our onLoad function, add a variable to store the 'textPosition'.

```
// the window load event handler
function onLoad()
{
    var canvas;
    var context;
    var textPosition;

    // this function will initialise our variables
    function initialise ()
    {
```

Then in the initialise function, make a new vector object with the coordinates you want the text to appear at.

```
        context = canvas.getContext('2d');
        if (!context)
        {
            alert('Error: failed to getContext!');
            return;
        }
        textPosition = new Vector(150,100);
    }
```

Finally add textPosition to the call of the draw function.

```
    // call the initialise and draw functions
    initialise();
    draw(textPosition);
}
```

Commit to SVN

Now that you have reached another milestone in your lab (instantiating and using a vector to position your text) it is a good time to commit your work to SVN. **Commit your changes with a meaningful log message starting with "L1T9" on the first line.**

Challenge

Try to get the text to be positioned centrally in the canvas area (both vertically and horizontally). Also, experiment with changing the colours of the text and background, along with the font type and size.

Commit to SVN

Now that you have reached another milestone in your lab (completing it) it is a good time to commit your work to SVN. **Commit your changes with a meaningful log message starting with "L1T8 Challenge" on the first line.**

Summary

In this lab you have created an HTML canvas and some accompanying JavaScript behaviour to draw a Hello World message to it. You should also have committed all your work to your 08240 module SVN repository along the way.