Week 4 HTML.

Exercises 9 & 10 from JavaScript lab should be completed with appropriate GUI's. SO if you've only done the JavaScript go back and build a front end for those two.

Working with HTML5 Canvas:

Probably one of the most exciting new features of HTML5 is the canvas. You can use it to create drawings anywhere on a web page. The only way to do this previously was by using some other technology such as Flash, SVG, or some other browser plugin.

The HTML5 canvas is both an element and an API. The <canvas> element defines a rectangular area of a web page where graphics can be drawn. The canvas API works with a <canvas> element to provide the JavaScript interface to draw on the canvas. It is a low-level set of functions for drawing lines, rectangle, circles, and other graphic primitives.

The <canvas> element itself is very simple. You must set the width and height attributes to specify its size. You can optionally put content inside the <canvas> element to be displayed for browsers that don't support it. The good news is that the HTML5 <canvas> element is widely supported by nearly every modern browser.

Getting a context.
The canvas API is accessed via a canvas context object. You get the context by calling the getContext() method of the <canvas> element, passing in a string parameter that defines the type of the context you want:

```
var context = $("canvas")[0].getContext("2d");
```

The only valid context type parameter you can pass into getContext() at this time is "2d" . This begs the question, "Is there a 3D context?" The answer is no, there is not. But we can always hope for one in the future.

Canvas basics
In this section we will learn some of the basics of using the canvas API. Now that we have a context, we can call its methods to draw lines and shapes. The API has a whole host of methods that let you draw everything from the most basic lines, to shapes, and even bitmap images.

Line Methods
These are methods to allow you to draw lines on the canvas:

Sr.No.	Method and Description
1	**beginPath()** This method resets the current path.
2	**moveTo(x, y)** This method creates a new subpath with the given point.
3	**closePath()** This method marks the current subpath as closed, and starts a new subpath with a point the same as the start and end of the newly closed subpath.
4	**fill()** This method fills the subpaths with the current fill style.
5	**stroke()** This method strokes the subpaths with the current stroke style.
6	**lineTo(x, y)** This method adds the given point to the current subpath, connected to the previous one by a straight line.

Example

Following is a simple example which makes use of above mentioned methods to draw a triangle.

```
<!DOCTYPE HTML>
<html>
   <head>
   
      <style>
         #test {
            width: 100px;
            height:100px;
            margin: 0px auto;
         }
      </style>
      
      <script type="text/javascript">
         function drawShapes(){
            // get the canvas element using the DOM
            var canvas = document.getElementById('mycanvas');
            
            // Make sure we don't execute when canvas isn't supported
            if (canvas.getContext){
            
               // use getContext to use the canvas for drawing
               var ctx = canvas.getContext('2d');

               // Filled triangle
               ctx.beginPath();
               ctx.moveTo(25,25);
               ctx.lineTo(105,25);
               ctx.lineTo(25,105);
               ctx.fill();

               // Stroked triangle
               ctx.beginPath();
               ctx.moveTo(125,125);
               ctx.lineTo(125,45);
               ctx.lineTo(45,125);
               ctx.closePath();
               ctx.stroke();
            }

            else {
               alert(Update your browser, eg Safari or Firefox 1.5+ to see
this demo.');
            }
         }
      </script>
   </head>

   <body id="test" onload="drawShapes();">
      <canvas id="mycanvas"></canvas>
   </body>

</html>
```

Ex1.html - The above example would draw following shape –

Line Properties

There are several properties which allow us to style lines.

Sr.No.	Property and Description
1	**lineWidth [= value]** This property returns the current line width and can be set, to change the line width.
2	**lineCap [= value]** This property returns the current line cap style and can be set, to change the line cap style. The possible line cap styles are *butt*, *round*, and *square*
3	**lineJoin [= value]** This property returns the current line join style and can be set, to change the line join style. The possible line join styles are *bevel*, *round*, and *miter*.

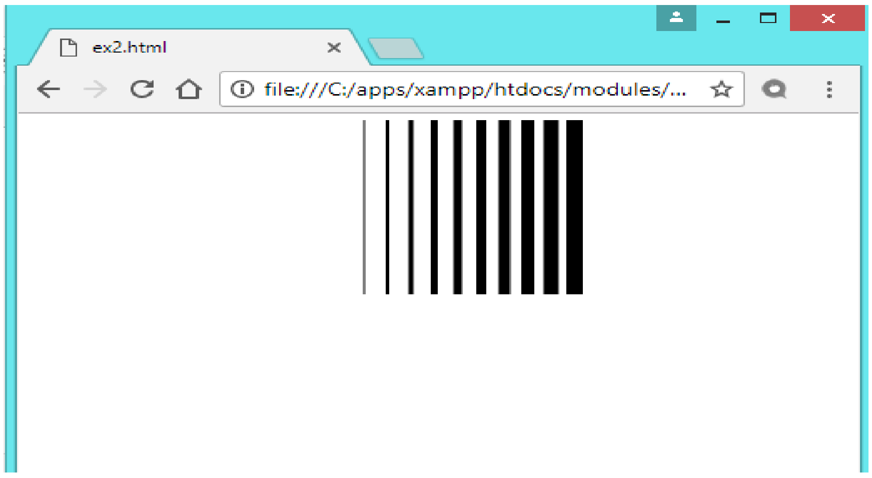
4	**miterLimit [= value]** This property returns the current miter limit ratio and can be set, to change the miter limit ratio.

Another Example

Following is a simple example which makes use of *lineWidth*property to draw lines of different width.

```
<!DOCTYPE HTML>
<html>
   <head>
   
      <style>
         #test {
            width: 100px;
            height:100px;
            margin: 0px auto;
         }
      </style>
      
      <script type="text/javascript">
         function drawShape(){
            // get the canvas element using the DOM
            var canvas = document.getElementById('mycanvas');
            
            // Make sure we don't execute when canvas isn't supported
            if (canvas.getContext){
            
               // use getContext to use the canvas for drawing
               var ctx = canvas.getContext('2d');
               
               for (i=0;i<10;i++){
                  ctx.lineWidth = 1+i;
                  ctx.beginPath();
                  ctx.moveTo(5+i*14,5);
                  ctx.lineTo(5+i*14,140);
                  ctx.stroke();
               }
            }
            
            else
            {
               alert('You need Safari or Firefox 1.5+ to see this demo.');
            }
         }
      </script>
   </head>
   
   <body id="test" onload="drawShape();">
      <canvas id="mycanvas"></canvas>
   </body>
</html>
```

Ex2.html - The above example would draw following shape –



Clearing the canvas
The background of the canvas is transparent. Whatever background color you specify for the canvas element in your CSS will show through. You can clear the canvas, or a portion of it, using the context's clearRect() method. It takes x, y, width, and height parameters and clears that part of the canvas.

```
context.clearRect(0, 0, canvas.width, canvas.height);
```

Exercise 1: Make sure you practice with the various methods in the tables above to draw a variety of shapes eg triangle, square, rectangle, circle etc
For the rectangle you could draw a border round your canvas.

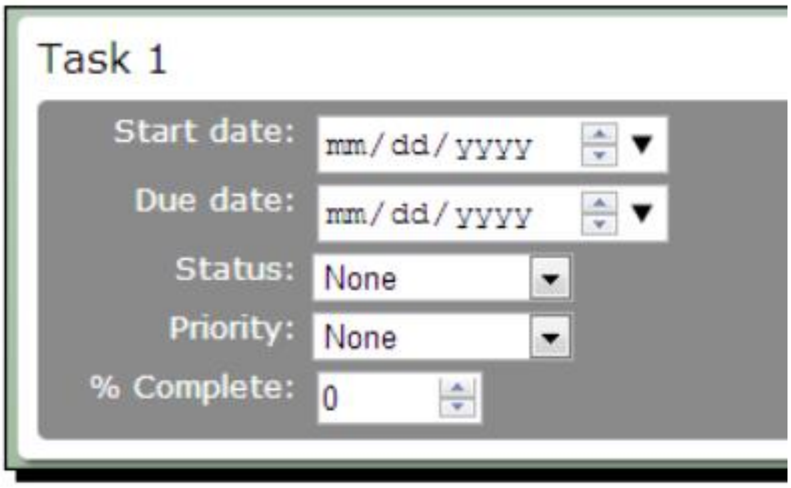
HTML5 input types

HTML5 comes with a whole host of new input types. These new types are designed to provide formatting, validation, and in some cases, selectors. For touch devices some of them provide a different set of keys for the keyboard. Not all of the new input types are supported by all browsers yet. Fortunately for us, if a browser doesn't support a type it will just display it as a normal text field. Unfortunately for us, you can't depend on the browser to provide the correct formatted data if the unsupported types are only shown as text fields. So make sure you have a backup plan if you are going to use them.

Here are a few of the more useful new input types with images of the ones that are supported by Chrome.

Color, Date, Email, Number, Range, Time and Datalist.

Exercise 2: Build an HTML form that uses some of the new tags to collect data from a user that defines a task that needs to be scheduled for completion e.g.


Task 1

Start date:	mm/dd/yyyy
Due date:	mm/dd/yyyy
Status:	None
Priority:	None
% Complete:	0

HTML5 Geolocation API

Geolocation is widely supported by nearly every modern browser. The accuracy of the location depends on the capabilities of the user's device. Devices that have GPS will give you a very accurate location, while those that don't will try to determine the user's location as close as they can by some other means, such as by IP address.

The Geolocation API is accessed by using the navigator.geolocation object. To get the user's location you call the `getCurrentPosition()` method. It takes two parameters- a callback function if it succeeds and a callback function if it fails:

```
navigator.geolocation.getCurrentPosition(
   function(position) { alert("call was successful"); },
   function(error) { alert("call failed"); }
);
```

The function that is called on success gets an object passed into it that contains another object named coords . The following is a list of some of the more useful fields the cords object contains:

- ‹latitude : This is the user's latitude in decimal degrees (for example, 44.6770429).
- ‹longitude : This is the user's longitude in decimal degrees (for example, -85.60261659).
- ‹accuracy : This is the accuracy of the position in meters.
- ‹speed : This is the speed the user is moving in meters per second. This is available for devices with GPS.
- ‹heading : This is the heading degrees that the user is moving in. Like speed this is for devices with GPS.

For example, if you wanted to get the user's location you would do the following:

```
var loc = position.coords.latitude + ", " +
          position.coords.longitude);
```

The user must allow your page to use the Geolocation API. If they reject your request, the call to `getCurrentPosition()` will fail, and depending on the browser your error handler may get called or fail silently.

The error handler gets passed an error object that contains two fields, code and message. The code field is an integer error code and message is the error message string. There are three possible error codes: permission denied , position unavailable , or timeout .

The Geolocation API also has a `watchPosition()` method. It works the same as `getCurrentPosition()` , except that your callback function gets called whenever the user moves. This way you can track the user and update their position in your application in real time.

Exercise 3: Build an HTML5 form that displays your current position.

That’s all for now but keep practicing.