HTML5

In this module we'll take a look at some of the new features in HTML5

Topics for this module

In this module going to take a look at HTML5. Topics include the following:

- Introduction to HTML5
- Progressive Enhancement & Browser Support
- HTML5 Semantic Elements
- HTML5 Form Elements
- HTML5 Video Basics
- HTML5 Canvas

Introduction to HTML5

HTML5 is a cooperation between the World Wide Web Consortium (W3C) and the Web Hypertext Application Technology Working Group (WHATWG).

WHATWG had been working with web forms and applications

W3C had been working with XHTML 2.0.

In 2006 they decided to cooperate on a new version of HTML – HTML5.

Introduction to HTML5

HTML5 adds a range of features, including:

- New semantic elements
- New form elements
- Local storage
- Geolocation
- Embedded audio & video
- Canvas

Note: Not all features are supported by all browsers. In particular, IE8 and earlier versions have limited support. We'll discuss how features can be detected as we progress through the module.

Progressive Enhancement

Progressive enhancement is a strategy that applies web technologies using a layered approach:

- Basic content & functionality is made available to all browsers.
- Enhanced features are made available to more advanced browsers.
- Enhanced layout is implemented via external CSS.
- Enhanced functionality is implemented via external JavaScript/jQuery files.
- Libraries such as Modernizr allow browser feature detection, and optional loading of JS/CSS resources, to add missing HTML5 functionality.

Backward Compatibility

Although HTML5 adds a lot of new functionality, some older browsers do not support it.

Even amongst newer browsers there are differing levels of support.

Polyfills provide a way of implementing new functionality in older browsers:

- The name comes from the 'Polyfilla' brand of products used for filling in cracks when home decorating.
- Polyfill = "A shim that mimics a future API, providing fallback functionality to older browsers" (Wikipedia).

Backward Compatibility

More information on polyfills, feature detection and browser compatibility can be found at the following addresses:

- https://github.com/Modernizr/Modernizr/wiki/HTML5-Cross-Browser-Polyfills
- http://shims-and-shivs.heroku.com
- http://caniuse.com/

Browser support for HTML5 features

The Modernizr library can be used to detect support for HTML5 features.

Once the library is loaded, the Modernizr object exposes properties that can be examined to determine whether a particular feature is supported:

```
<script type= "text/javascript" src="Modernizr.js">


<script>
    if (Modernizr.canvas) {
        console.log("HTML5 canvas is supported");
    } else {
        console.log("HTML5 canvas not supported");
    }
</script>
```

Browser support for HTML5 features

Modernizr includes an optional tool called Modernizr.load. *

This provides a way of loading external CSS and JavaScript files conditionally, depending on whether a specific feature is available or not:

```
Modernizr.load({
    test: Modernizr.canvas,
    yep: 'my-canvas-library.js',
    nope: 'http://flashcanvas.net/bin/flashcanvas.js'
});
```

* Modernizr.load is also available as a standalone library ‘yepnope.js’:

http://yepnopejs.com

New HTML5 Elements

HTML5 introduces a number of new elements. These can be grouped in the following categories:

- **Semantic** & structural elements
- **Form** elements
- **Media** elements (audio & video)
- The **<canvas>** element
- **Web app** features (geolocation, local storage)

Note: geolocation and local storage have been covered previously, and are not included in this module.

HTML5 Semantic Elements

In the context of an HTML document, term ‘semantic’ relates to the use of markup elements to give meaning to content.

For example, <code> represents some kind of computer code, and <cite> implies the title of a work, such as a book.

Prior to HTML5, there were a limited number of semantic elements.

Developers used <div> elements with ids and classes to identify content such as headers and footers, navigation lists, articles, etc.:

```
<div class="article">Article content...</div>
```

HTML5 introduces new elements for common content types:

```
<article>Article content...</article>
```

HTML5 Semantic Elements

HTML5 has a number of new semantic and structural elements, including:

- <section>
- <nav>
- <article>
- <aside>
- <header>
- <footer>

A full list is available at:

http://www.w3schools.com/html/html5_new_elements.asp

HTML5 <section> element

The <section> element represent a logical section of content in a page.

Each section can have its own <h1> element.

```
<section id="about">
    <h1>About us</h1>
    <!-- content -->
</section>
<section id="news">
    <h1>Latest news</h1>
   <!-- content -->
</section>
```

Matt Cutts of Google explains use of multiple <h1> elements:

http://www.youtube.com/watch?v=GIn5qJKU8VM

HTML5 <nav> element

The <nav> element is used to identify a section of a page that contains the primary navigation links to other pages.

It would typically be contained within a <header> section:

```
<section>
    <header>
        <nav><!-- main navigation menu --></nav>
        <h1>Welcome to our site...</h1>
        <p>Hey, thanks for stopping by...</p>
    </header>
</section>
```

HTML5 <article> element

The <article> element is used to identify a part of a page that represents a self-contained piece of content such as a blog post, user-comment or article.

It would usually be a distinct piece of content that could be published separately via a RSS feed. For example:

```
<section>
    <article>
        <h2>Article title</h2>
        <p>Article content...</p>
        <p>etc...</p>
    </article>
</section>
```

HTML5 <aside> element

The <aside> element is used to identify content that is related to content around it but is considered to be separate to that content.

It would typically be used for effects such as sidebars or pull-quotes.

```
<section>
    <article>
        <h2>Article title</h2>
        <p>Article content...</p>
        <p>etc...</p>
        <aside>
            <!-- Quote from article -->
        </aside>
    </article>
</section>
```

HTML5 <header> element

The <header> element is used to identify and wrap a group of introductory or navigation elements as well as elements such as search forms and logos.

A page might just have one <header> element or a separate <header> for each <section>.

```
<section>
    <header>
        <!-- header content -->
    </header>
</section>
<section>
    <header>
        <!-- header content -->
    </header>
</section>
```

HTML5 <footer> element

The <footer> element identifies a footer section – either for the page as a whole or for its closest ancestor element.

It would normally contain copyright details, the author's name, links to related pages, etc.

```
<section>
    <header>
        <nav><!-- main navigation menu --></nav>
        <h1>Welcome to our site...</h1>
        <p>Hey, thanks for stopping by...</p>
    </header>
    <article><!-- article content --></article>
    <article><!-- article content --></article>
    <footer><!-- footer content --></footer>
</section>
```

HTML5 Form Elements & Input Types

HTML5's new form elements and input types include:

- **<datalist>** – provides 'autocomplete' on <input> elements
- **<output>** – for the results of a calculation
- <input type="**color**"> – displays a color-picker
- <input type="**date**"> – displays a date control
- <input type="**email**"> – automatic email format validation
- <input type="**range**"> – input for range of values (slider)

A full list is available at:

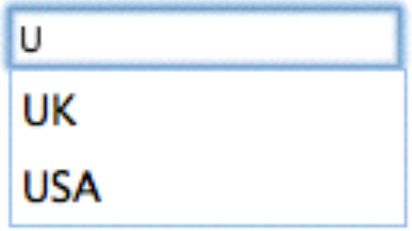
http://www.w3schools.com/html/html5_form_input_types.asp

HTML5 Form Elements & Input Types

The <datalist> elements allows you to specify pre-defined ‘auto-complete’ options for an input element.

It is currently only supported in IE10+, Chrome, Opera and Firefox.

```
<input list=“countries”>

<datalist>
    <option value=“UK”>
    <option value=“USA”>
    <option value=“Canada”>
    <option value=“Spain”>
    <option value=“Italy”>
</datalist>
```

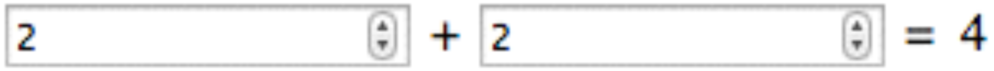


HTML5 Form Elements & Input Types

The <output> element can be used to contain the result of a calculation.

It is not currently supported in IE.

```
<form oninput="sum.value=parseInt(num1.value)
+parseInt(num2.value)">

    <input type="number" id="num1" value="2"> +
    <input type="number" id="num2" value="2"> =
    <output name="sum" for="num1 num2">4</output>


</form>
```

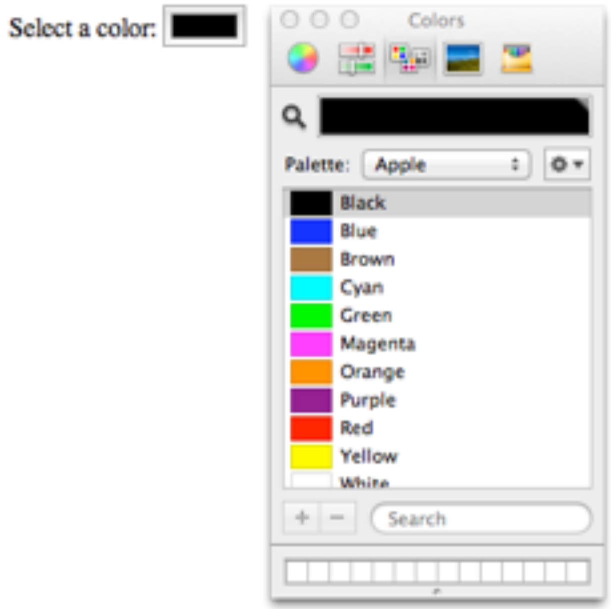


HTML5 Form Elements & Input Types

The type="color" attribute can be used to display a color-picker.

It is currently supported in Chrome and Opera.

```
<form>
    <label for="myColor">Select a color:</label>
    <input type="color" id="myColor" name="myColor">
  </form>
```

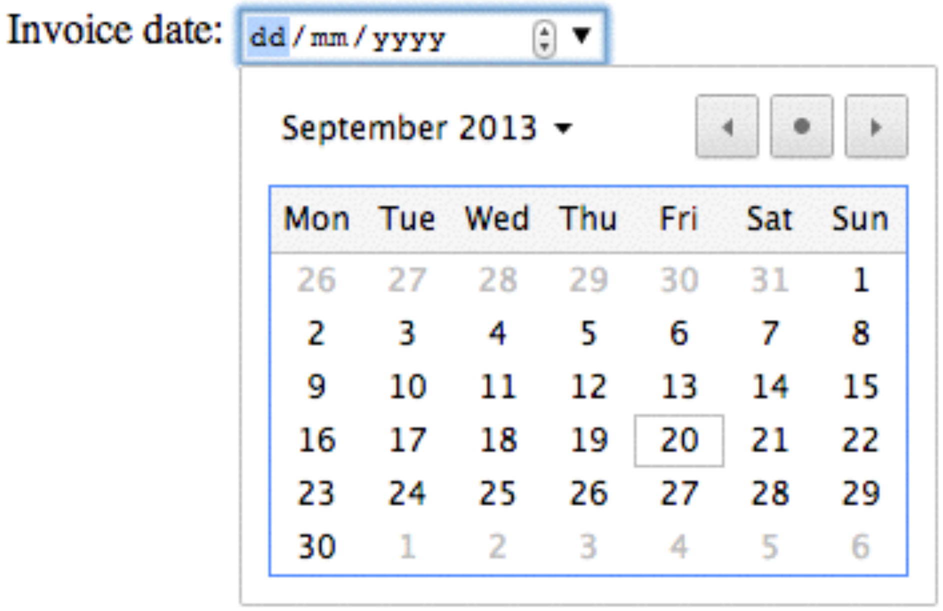


HTML5 Form Elements & Input Types

The type="date" attribute can be used to display a date-picker.

It is currently supported in Chrome, Safari and Opera.

```
<form>
  Invoice date: <input type="date" name="invDate">
</form>
```



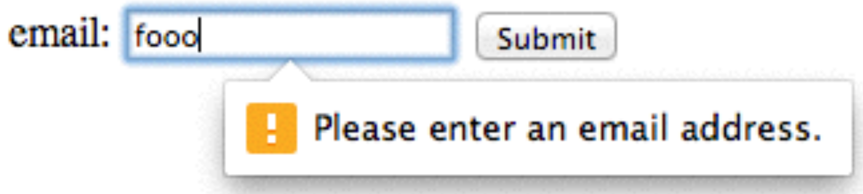
Mon	Tue	Wed	Thu	Fri	Sat	Sun
26	27	28	29	30	31	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	1	2	3	4	5	6

HTML5 Form Elements & Input Types

The type="email" attribute can be used to create an input field that will validate an email address when the form is submitted.

It is currently supported in Firefox, Chrome and Opera.

```
<form>
    <label for="email" >email: </label>
    <input type="email" id="email" name="email">
    <input type="submit">
</form>
```



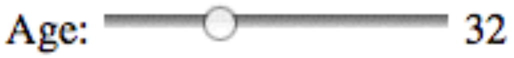
Please enter an email address.

HTML5 Form Elements & Input Types

The type="range" attribute can be used to create an slider type input field for setting a numeric value.

It is currently supported in Firefox, Chrome and Opera.

```
<form oninput="myAge.value=age.value;">
    <label for="age">Age: </label>
    <input type="range" name="age" id="age" min="1" max="100">
    <output name="myAge"></output>
</form>
```



HTML5 Form Element Attributes

HTML5 introduces new attributes for form elements, including:

- **autocomplete**
- **novalidate**
- **autofocus**
- **list**
- **min**
- **max**
- **pattern**
- **placeholder**
- **required**

A full list can be found at:

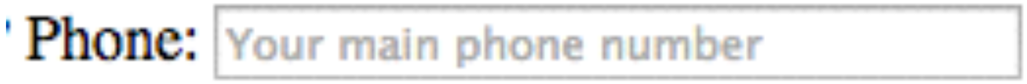
http://www.w3schools.com/html/html5_form_attributes.asp

HTML5 Form Elements & Input Types

The **placeholder** attribute can be used to provide a hint or prompt for input fields.

It is currently supported in Firefox, Chrome, Opera, Safari and IE10.

```
<form>
    <label for "phone">Phone: </label>
    <input type="text" id="phone" name= "phone"
     placeholder="Your main phone number">
</form>
```


Phone: Your main phone number

Video in HTML5

Over recent years, video content has become more and more popular on website pages.

Prior to HTML5, there was no standard way of embedding a video into a page. Video had to be delivered via a third-party plugin, such as RealPlayer or Quicktime.

HTML5 provides a standard method to embed video, by using the <video> element.

Before we discuss at the way in which the <video> element is used, let's look at how video files themselves work...

Video in HTML5

Video file formats, such as MP4 or AVI, act as containers for a number of tracks that are used to deliver video content.

This is similar in concept to a zip file, which is a container for other files.

The tracks that are used to generate video content will include both video and audio tracks.

The tracks contain markers that allow them to relate to each other, so that the audio is in sync with the the video for example.

Tracks can also have meta data. For example, an audio track may contain meta data that identifies the language of the track.

The video container itself can also have meta data, such as the title of the video, release date, etc.

There are a number of container formats in use, some of which are listed on the next slide...

Video in HTML5

Examples of video container formats:

- Audio Video Interleave (AVI)
- ASF
- Flash Video
- WebM
- MPEG 4 (MP4)
- Ogg

The container format defines how the individual tracks are stored in a single file.

Video in HTML5

When a video (with sound) is played, the following processes take place:

- The container (AVI, MP4, etc.) is examined to determine what audio and video tracks are available, and how they are stored in the file.
- The video track is decoded and displayed as a series of images.
- The audio stream is decoded and played through the audio output device (speakers, headset, etc.).

The method that is used to encode and decode a video or audio stream is known as a **codec** (a combination of the works ‘coder’ and ‘decoder’).

There are a number of different video and audio codecs in use, some of which are listed on the next slides…

Video in HTML5

Examples of video codecs:

- VP8
- Theora
- H.264

The H.264 codec is designed for use with a variety of devices, such as desktops, tablets and mobile phones.

These devices will vary in terms of the bandwidth that they can use, and the power of their CPUs.

The H.264 codec uses profiles to configure the options for different device types.

Video in HTML5

Examples of audio codecs:

- Advanced Audio Coding (AAC)
- MPEG-1 Audio Layer 3 (MP3)
- Vorbis

In an ideal world, there would be a standard combination of container formats, codecs and profiles that was supported across all browsers.

However, as you might expect, that is not actually the case, and different browsers support different combinations.

The next slide shows which combinations of container and codecs are supported by the main browsers...

Video in HTML5

Browser support for video:

Container & codecs	IE	FF	Safari	Chrome	Opera	iPhone	Android
H.264+ AAC + MP4	9.0+	n/a	3.0+	5.0+	n/a	3.0+	2.0+
Theora + Vorbis + Ogg	n/a	3.5+	n/a **	5.0+	10.5+	n/a	n/a
WebM *	9.0+	4.0+	n/a **	6.0+	10.6+	n/a	2.3+

Unfortunately, there is no single combination of container and codec types that is supported by all browsers.

Notes:

** The WebM container format uses the VP8 video codec and Vorbis audio codec.*

*** Safari doesn't have native support for Theora or WebM containers, only via third-party plugins.*

Video in HTML5

The fact that there is no combination of video containers and codecs that is supported across all browsers could present a problem.

Fortunately, HTML5 allows more than one video file to be defined for a <video> element.

The browser will then use the first file format that it can play.

This does mean however, that you will need to encode your video in a range of formats for modern browsers. *

A Flash version can also be provided as a fallback for older browsers.

** Note: Video encoding is outside the scope of this module, but a useful encoding tool can be found here:*

http://www.mirovideoconverter.com

Video in HTML5

If you can be absolutely sure that only one format is needed for your requirements (maybe an iPhone or Android App) then the video file can be specified by using a src attribute in the opening <video> tag:

```
<video src="myvideo.webm" width="400" height="250"></video>
```

A default set of player controls can be added with a **controls** attribute:

```
<video src="myvideo.webm" controls></video>
```

The video can be downloaded once the page has finished loading by including a **preload** attribute. This makes it ready for immediate play:

```
<video src="myvideo.webm" preload></video>
```

To have the browser start playing the video automatically when the page has loaded, include an **autoplay** attribute:

```
<video src="myvideo.webm" autoplay></video>
```

Video in HTML5

For public-facing website and browser based applications, it will most likely be the case that a number of possible video file formats need to be specified.

This is done by using separate <source> elements inside a parent <video> element:

```
<video width="400" height="250" controls preload>
  <source src="vid1.webm" type="video/webm">
  <source src="vid1.mp4" type="video/mp4">
  <source src="vid1.ogv" type="video/ogg">
</video>
```

The src attribute is now defined in each <source> element, rather than in the <video> element.

A type attribute is also included. This is known as the file's MIME Type.

Video in HTML5

The video file's data type (aka MIME type) should be specified both in the HTML markup (as the first part of the type attribute) and in the HTTP header sent by the server when it returns the video as a resource representation.

If Apache is being used, then the MIME type can be set in one of two ways:

- By adding directives to the Apache configuration file.
- By adding directives to a .htaccess file saved in the same directory as the video files.

In either case, the directives would be as follows:

```
AddType video/mp4  .mp4
AddType video/ogg  .ogv
AddType video/webm  .webm
```

Note: other commonly used MIME types include:

- text/css - for CSS files
- text/javascript - for JS files
- text/html - for HTML files

The HTML5 Canvas

The <canvas> element is new to HTML5. It allows you to use JavaScript to render or ‘draw’ graphic content.

The content could be a simple drawing, a graph, or graphics for a game.

The first step is to define the dimensions of a rectangular area in which the graphic content will be rendered:

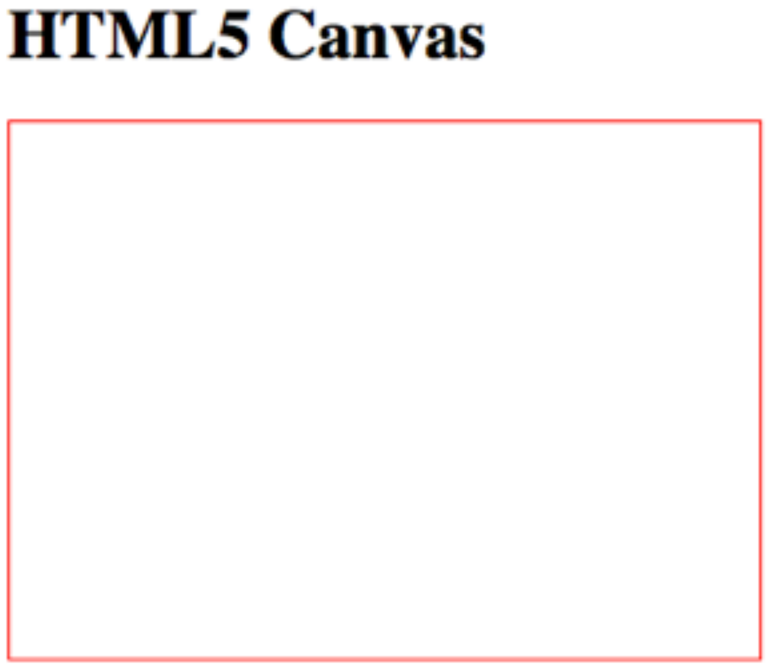
```
<h1>HTML5 Canvas</h1>
<canvas width="350" height="250"></canvas>
```

This doesn’t actually make anything visible on the page. We can use some CSS to add a border:

```
canvas {
    border: 1px solid red;
}
```

The HTML5 Canvas

Now that a border has been added, the canvas area can be seen.


HTML5 Canvas

The HTML5 Canvas

Now that a border has been added, the canvas area can be seen.

The next step is to use JavaScript to draw something on the canvas .

The first step is to select the canvas. Let's add an id attribute to make that simple:

```
<canvas id="canvas1" width="350" height="250"></canvas>
```

The canvas can then be selected with JavaScript or jQuery:

```
//JavaScript syntax
var canvas = document.getElementById("canvas1");
//jQuery syntax
var canvas = $("canvas1")[0];
```

Note: the jQuery selector returns an extended object. The canvas itself is referenced via the first element.

The HTML5 Canvas

Every canvas element has a context. The context is where any drawing is rendered.

Once the canvas object has been selected, its **`getContext()`** method is called:

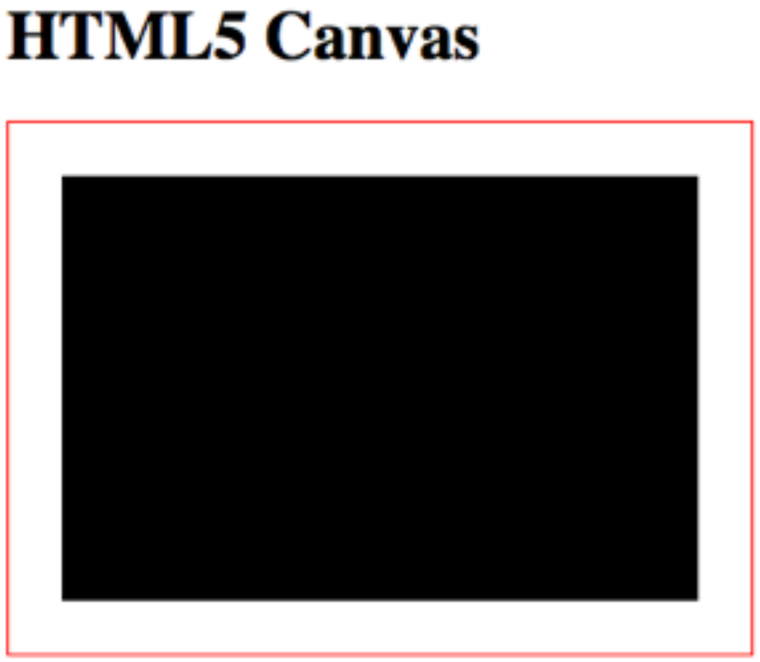
```
var canvas = document.getElementById("canvas1"),
    context = canvas.getContext();
```

Now we can actually draw something on the canvas. For example, to draw a rectangle:

```
var canvas = document.getElementById("canvas1"),
    context = canvas.getContext();
context.fillRect(25, 25, 300, 200);
```

The HTML5 Canvas

A solid black rectangle has now been added to the canvas:


HTML5 Canvas

A rectangle is added by calling the context's `fillRect()` method and passing 4 values:

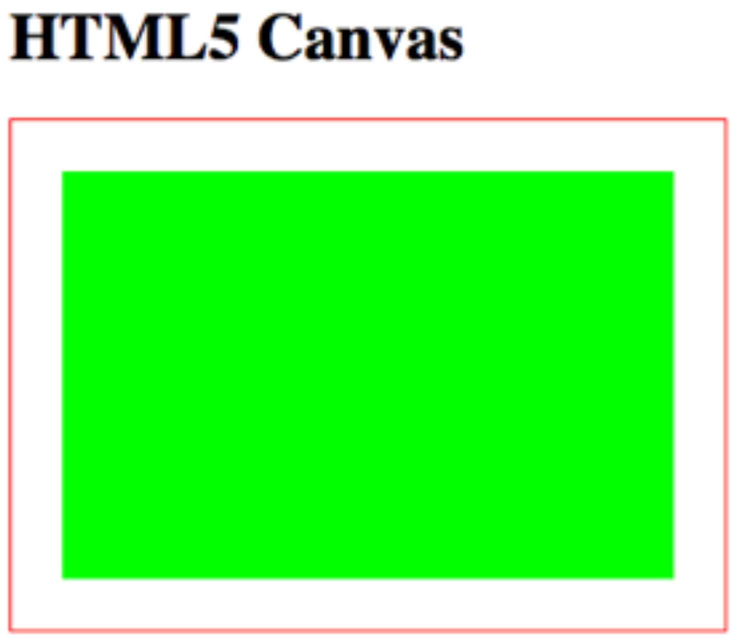
```
context.fillRect(25, 25, 300, 200);
```

- The first 2 values are the x and y coordinates of the top left corner.
- The next 2 values are the width and height.
- All values are in pixels.
- Coordinates are given as offsets from the top-left corner of the canvas, which has coordinates of (0,0).

The HTML5 Canvas

The default color for a rectangle is black. The context's **`fillStyle`** property can be modified to change that:

```
context.fillStyle = "#0f0"; // CSS color value for green
context.fillRect(25, 25, 300, 200);
```



HTML5 Canvas

The HTML5 Canvas

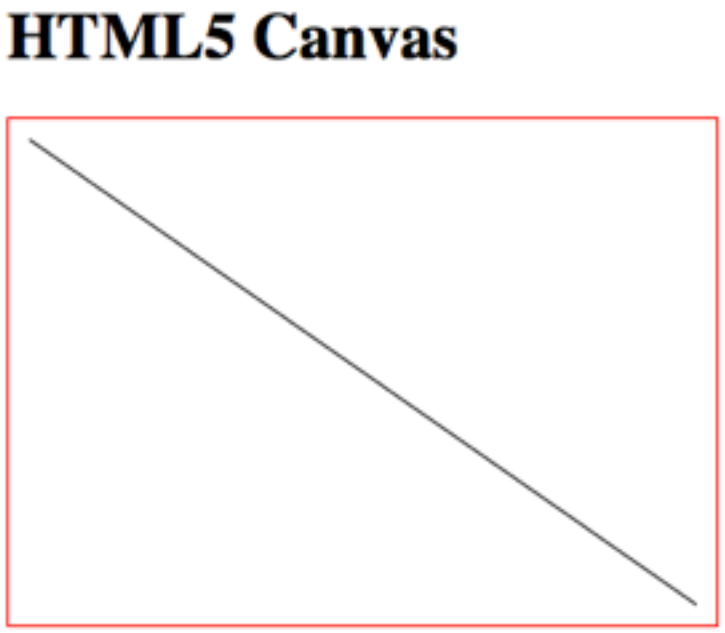
As we've seen, drawing a rectangle with the `fillRect()` function is fairly straightforward.

Drawing lines requires a little more work.

- First of all, the path of the line is defined by using the **`moveTo()`** and **`lineTo()`** methods. This doesn't actually draw the line, it simply defines the coordinates where the line will start and end.
- The path can be for a simple line between two points, or a more complex path that passes through a number of points.
- Once the line's path has been defined, the line itself is drawn with the **`stroke()`** method.
- We'll look at an example on the next page...

The HTML5 Canvas

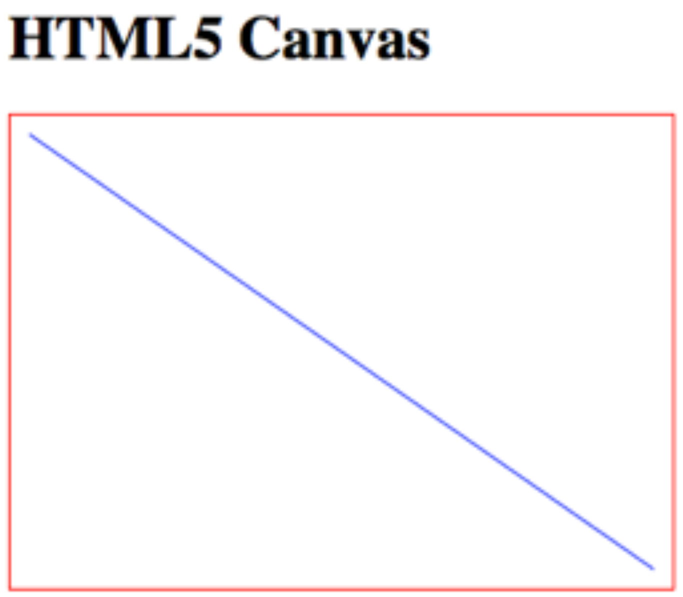
```
var canvas = document.getElementById("canvas1"),
    context = canvas.getContext();

context.moveTo(10, 10); // Coordinates for start of the line
context.lineTo(340, 240); // Coordinates for end of the line
context.stroke(); // Draw the line
```



The HTML5 Canvas

The line’s color can be modified via the **`strokeStyle`** property.

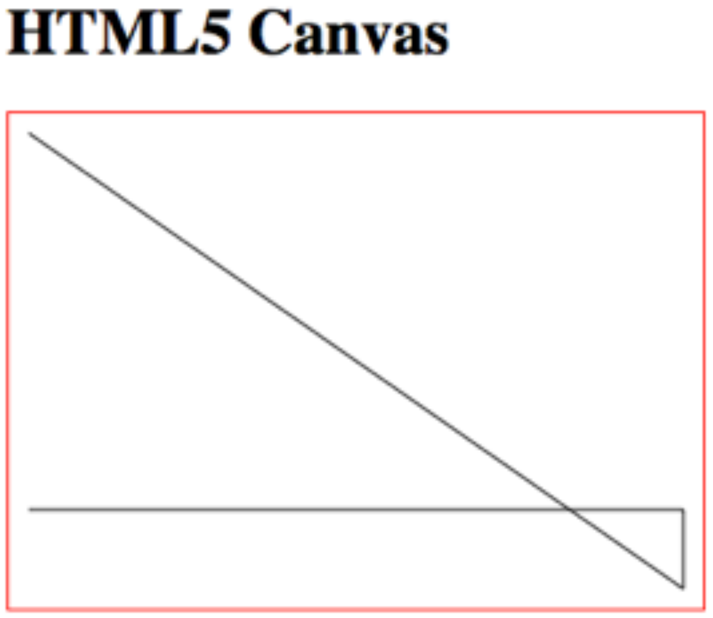
```
context.moveTo(10, 10); // Coordinates for start of the line
context.lineTo(340, 240); // Coordinates for end of the line
context.strokeStyle = "#00f"; // CSS color value for blue
context.stroke(); // Draw the line
```



The HTML5 Canvas

More complex lines can be drawn by calling `lineTo()` multiple times:

```
context.moveTo(10, 10); // Coordinates for start of the line
context.lineTo(340, 240); // Next point on line
context.lineTo(340, 200); // Next point on line
context.lineTo(10, 200); // Next point on line
context.stroke(); // Draw the line
```



HTML5 Canvas

The HTML5 Canvas

The previous slides have covered the basic concepts of working with the canvas and drawing some basic rectangles and lines.

There is much more than can be done with the canvas element, and an excellent series of examples and tutorials can be found here:

http://www.html5canvastutorials.com/

Questions

Do you have any questions before we finish the course

?

Don't worry, if you think of something later then just ask!