



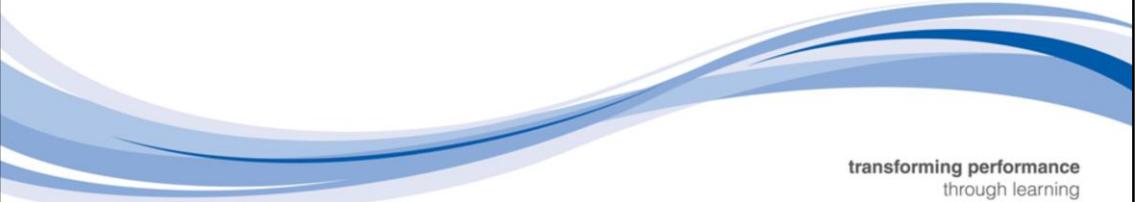
# Developing Web Applications Using HTML5

## Delegate Guide



# Introduction

Developing Web Applications using HTML5



transforming performance  
through learning

## Chapter overview

- **Objectives**
  - To explain the aims and objectives of the course
- **Contents**
  - Course administration
  - Course objectives and assumptions
  - Introductions
  - Any questions?
- **Exercise**
  - Locate the exercises

At the end of this session you will be able to:

- Explain the aims and objectives of the course

Contents:

- Course administration
- Course objectives and assumptions
- Introductions

## Administration

- **Front door security**
- **Name card**
- **Chairs**
  
- **Fire exits**
- **Toilets**
- **Smoking**
- **Coffee Room**
  
- **Timing**
- **Breaks**
- **Lunch**
  
- **Downloads & Viruses**
- **Admin support**
- **Messages**
  
- **Taxis**
- **Trains/Coaches**
- **Hotels**
  
- **First Aid**
  
- **Telephones/Mobiles**

We need to deal with practical matters right at the beginning.

Above all, please ask if you have any problems regarding the course or practical arrangements. If we know early on that something is wrong, we have the chance to fix it. If you tell us after the course, it's too late! We ask you to fill in an evaluation form at the end of the course. If you alert us a problem for the first time on the feedback form at the end of the course then we have not had the opportunity to put it right.

If this course is being held at your company's site, much of this will not apply or will be outside our control.

## Course delivery



### Lecture material

*Hear and Forget  
See and Remember  
Do and Understand*



### Questions and exercises

### Course workbooks



### Practical sessions

The course will be made up of lecture material coupled with the course workbook, informal questions and exercises, and structured practical sessions. Together, these different teaching techniques will help you to absorb and understand the material in the most effective way.

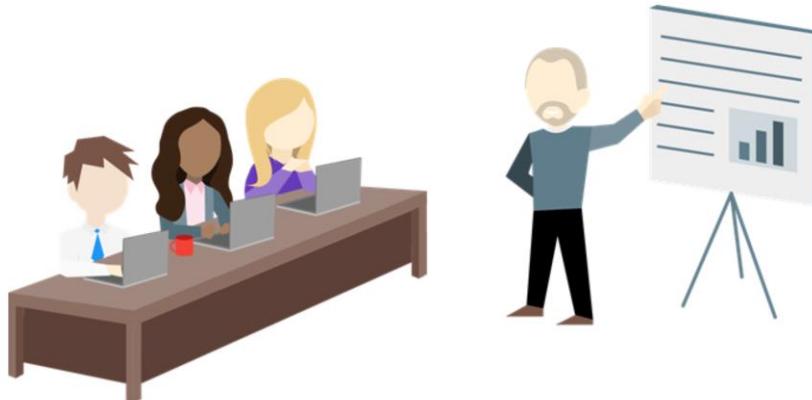
The course notebooks contain all the overhead slides that will be shown, so you do not need to copy them. In addition, there are extra textual comments (like these) below the slides, which are there to amplify the slides and provide further information. Hopefully these notes mean you will not need to write too much and can listen and observe during the lectures. There is, however, space to make your own annotations too.

In the practical exercise sessions, you will be given the opportunity to experiment and consolidate what has been taught during the lecture sessions. Please tell the instructor if you are having difficulty in these sessions. It is sometimes difficult to see that someone is struggling, so please be direct.

## The training experience

### A course should be

- A two-way process
- A group process
- An individual experience



The best courses are not those in which the instructor spends all his or her time pontificating at the front of the class. Things get more interesting if there is dialogue, so please feel free to make comments or ask questions. At the same time, the instructor has to think of the whole group, so if you have many queries, he or she may ask to deal with them off-line.

Work with other people during practical exercise sessions. The person next to you may have the answer, or you may know the remedy for them. Obviously do not simply 'copy from' or 'jump-in on' your neighbour, but group collaboration can help with the enjoyment of a course.

We are also individuals. We work at different paces and may have special interests in particular topics. The aim of the course is to provide a broad picture for all. Do not be dismayed if you do not appear to complete exercises as fast as the next person. The practical exercises are there to give plenty of practical opportunities. They do not have to be finished and you may even choose to focus for a long period on the topic that most interests you. Indeed there will be parts labelled 'if time allows' that you may wish to save until later to give yourself time to read and absorb the course notes. If you have finished early, there is a great deal to investigate. Such "hacking" time is valuable. You may not get the opportunity to do it back in the office!

## Course aims and objectives

**By the end of the course you will be able to:**

- **Understand the purpose of HTML5**
- **Work with the HTML5 skeleton**
- **Use the HTML5 structural elements**
- **Consider the benefits of CSS3, especially selectors**
- **Create a HTML5 form and understand its limitations**
- **Work with video and audio**
- **Use the Canvas API**
- **Understand SVG**
- **Pinpoint yourself with Geolocation**
- **Wave goodbye to cookies with local storage**

This is a three day course focusing on the emerging HTML5 and CSS3 standards.

## Assumptions

- **This course assumes**
  - You are familiar with XHTML or HTML4
  - You have experience using CSS
  - You have JavaScript programming experience
- **This course focuses on skill migration**
  - It is not a course for developers new to web technologies

## Introductions

**Please say a few words about yourself:**

- **What is your name & job?**
- **What is your current experience of...**
  - Web Development?
  - HTML
  - CSS?
  - JavaScript?
- **What is your main objective for attending the course?**

One of the great benefits of attending these courses is meeting other people. They may have similar interests, have encountered similar problems and may even have found the solution to yours. The contacts made on the course can be very useful.

It is useful for us all to be aware of levels of experience. It will help the instructor judge the level of depth to go into and the analogies to make to help you understand a topic. People in the group may have specialised experience that will be helpful to others.

## Any questions?

- **Golden Rule**
  - “There is no such thing as a stupid question”
- **First amendment to the Golden Rule**
  - “...even when asked by an instructor”
- **Corollary to the Golden Rule**
  - “A question never resides in a single mind”

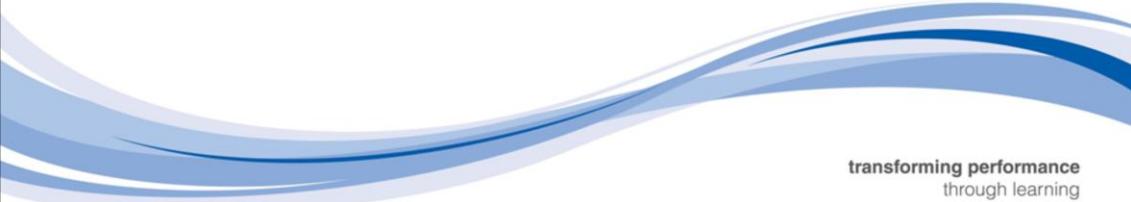
Please feel free to ask questions.

Teaching is a much more enjoyable and productive process if it is interactive. You will no doubt think of questions during the course; if so, ask them!



# Introducing HTML5

Developing Web Applications using HTML5



A decorative graphic consisting of several overlapping, curved blue and white bands that resemble waves or flowing water.

transforming performance  
through learning

## This chapter covers

- **What is HTML?**
- **Major HTML versions**
- **HTML5 and the basics of using it**
- **HTML5 browser support**
- **New JavaScript APIs and closely related web specifications**

## In this chapter, you'll learn

- About what HTML is and what the major HTML versions are
- Why you should use HTML5 and its advantages
- The wide range of JavaScript APIs available in HTML5 itself, as well as a number of closely related specifications you can use in your application

## What is HTML?

- **The well-known acronym ‘HTML’ stands for Hypertext Markup Language**
  - Primary language for world wide web
  - Used to create web documents
- **Utilises CSS (Cascading Style Sheets)**
  - Visual appearance of content
  - Layout, colours and fonts
- **Creation of interactive forms, embedding of images, video, audio, and other objects**
- **Can embed scripts**
  - JavaScript
  - Dynamic behaviour of web pages

The well-known acronym ‘HTML’ stands for Hypertext Markup Language.

- Primary language for world wide web
- Capable of creating web document by specifying content structure including headings, paragraphs, tables, footer and other elements

HTML markup also typically utilises CSS (Cascading Style Sheets) to describe the visual appearance of content.

- CSS enables the separation of document HTML content from document
- Visual presentation such as the layout, colours and fonts

HTML allows for the creation of interactive forms, embedding of images, video, audio, and other objects. HTML code can embed scripts, such as JavaScript, which contribute to dynamic behaviour of web pages.

## Major HTML versions

- **First HTML document**
  - “HTML Tags”
  - Published by Tim Berners-Lee in 1991
- **HTML 4.0**
  - W3C recommendation in 1997
  - Transitional, strict, and frameset
- **XHTML 1.0**
  - In 2000-2002.
  - Conforms to XML syntax requirements
- **XHTML 2.0 working drafts**
  - Released in 2002-2006
  - Make a more radical break from past version
    - Sacrificed backward compatibility
  - W3C halt any further development in favour of more flexible HTML5 standard

The first HTML document called “HTML Tags”, was published by Tim Berners-Lee in 1991.

HTML 4.0 was published as a World Wide Web Consortium (W3C) recommendation in 1997, offering three variations: transitional, strict and frameset.

XHTML 1.0 a more restrictive subset of HTML markup, was published in 2000-2002. It conforms to XML syntax requirements.

XHTML 2.0 working drafts were released in 2002-2006.

The proposed standard attempted to make a more radical break from past version, but sacrificed a backward compatibility.

Later W3C decided to halt any further development of the draft into standard, in favour of more flexible HTML5 standard.

## Major HTML versions

- **HTML5 first public draft**
  - In 2008
  - Completed in 2014
- **HTML 5.1 is currently being developed**
  - Various newly proposed HTML and CSS3 building blocks
  - 7 elements, 22 attributes, 18 events, 32 CSS properties and more

The HTML5 first public draft was released by the W3C in 2008 and it was completed in 2014.

HTML 5.1 is currently being developed.

- It includes various newly proposed HTML and CSS3 building blocks
- Including an additional: 7 elements, 22 attributes, 18 events, 32 CSS properties and more

## Major HTML versions

This is an estimated, combined timeline for HTML 5.0, HTML 5.1 and HTML 5.2

	2013	2014	2015	2016	2017
HTML 5.0	Call for Review	Recommendation			
HTML 5.1	Working Draft		Last Call	Candidate Rec	Recommendation
HTML 5.2			1 <sup>st</sup> Working Draft		Last Call

## HTML 5

- **HTML5 development**
  - Began in 2004
  - WHATWG group (Web Hypertext Application Technology Working Group)
    - Informal group of experts from Apple, the Mozilla Foundation and Opera Software
  - Ian Hickson, lead author
    - Google, Inc
- **HTML5 markup**
  - Backward compatible with HTML 4, XHTML 1.0, XHTML 2.0
- **HTML5**
  - Introduces many new elements
  - Semantic replacements for generic HTML elements
  - Many HTML4 elements were deprecated

HTML5 development began in 2004 by an informal group of experts from Apple computer, the Mozilla Foundation, and Opera Software forming the WHATWG group (Web Hypertext Application Technology Working Group). Ian Hickson of Google, Inc is the lead author on the HTML 5 specification.

HTML5 markup is more backward compatible with HTML 4 and XHTML 1.0 vs. XHTML 2.0.

HTML5 introduces many new elements including semantic replacements for generic HTML elements.

- New semantic elements such as <header>, <footer>, <section>, <nav>, <article> were created
- Many HTML4 elements were deprecated

## HTML 5

- **Introduces many additional plugin-in free capabilities**
  - Standardised video and audio interface
  - Local databases
  - Multi-threaded JavaScript
- **Unifies everything in a single specification**
- **XHTML5 is the XML serialisation of HTML5**

HTML5 also introduces many additional plugin-in free capabilities such as standardised video and audio interface, local databases, more efficient multi-threaded JavaScript and more.

HTML5 unifies everything in a single specification by allowing both HTML and XML serialisation; that is, the specification provides a vocabulary that can be expressed in with HTML or XHTML.

XHTML5 is the XML serialisation of HTML5. XHTML5 document is served with an XML MIME type, e.g. application/xhtml+xml.

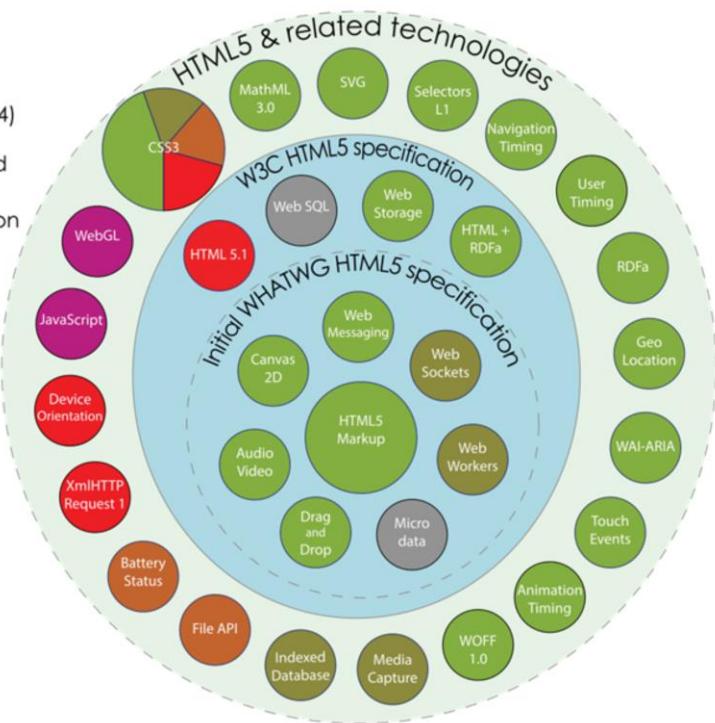
- Also XHTML5 requires stricter, well-formed syntax
- The HTML5 document type declaration is optional and may be omitted
- It may be utilised to extend HTML5 to some XML-based technology, such MathML

## HTML5

# HTML5

Taxonomy & Status (October 2014)

- Recommendation/Proposed
- Candidate Recommendation
- Last Call
- Working Draft
- Non-W3C Specifications
- Deprecated or inactive



If we take a look at Sergey Mavrody's Creative Common image we can see that HTML5 is a huge set of interrelated technologies and that some are already deprecated! It most certainly is not just markup.

## Browser Support

- **Browsers are updating rapidly to utilise HTML5**
  - This will mean mainstream HTML5 support across the board
  - The implementation is not universal, features are missing
  - A lot of work arounds are needed right now
  - IE8 and below have serious issues
  - Even early HTML5 browsers need support
- **Online Resources**
  - Your browser HTML5/CSS3 support test
    - <http://www.findmebyip.com>
  - Your browser HTML5 support test
    - <http://html5test.com>

When your browser renders a web page, it constructs a Document Object Model (*DOM*), a collection of objects that represent the HTML elements on the page.

Every element – every `<p>`, every `<div>`, every `<span>` – is represented in the DOM by a different object.

All DOM objects share a set of common properties, but some objects have more than others. In browsers that support HTML5 features, certain objects will have unique properties. A quick peek at the DOM will tell you which features are supported.

Testing for these and checking is a good way of checking compliancy, be aware that Internet Explorer 8 or less will be the big bugbear to your HTML5 development lifecycle right now.

## Why use HTML5 – advantages

- **Backward compatibility**
- **Simpler syntax**
- **New elements and attributes**
  - Make design and development more flexible
- **Plugin-free video and audio**
  - Timed media playback
- **Smart Web Forms 2.0 functionality**
- **In-line SVG**
- **MathML with text/html MIME Type**
- **Over 20 new plugin-in free scripting APIs**
  - Canvas 2D element graphics
  - Document Editing
  - Drag-and-drop
  - Geolocation, Local offline storage and Media Capture
- **Bottom line**
  - Easier development
  - Enhanced user experience

### HTML5 advantages

- Backward compatibility: HTML5 is wrapping up all previous doctypes
- Simpler Syntax: improved semantics, more productive coding and smaller document size
- New elements and attributes make design and development more flexible
- Plugin-free video and audio and timed media playback
- Smart Web Forms 2.0 functionality (HTML5 supersedes Web Forms 2.0)
- Ability to use inline SVG and MathML with text/html MIME Type
- Over 20 new plugin-in free scripting APIs (application programming interfaces), including: Canvas 2D element graphics, Document Editing, Drag-and-drop, Geolocation, Local offline storage and Media Capture
- The bottom line: easier development and enhanced user experience

## Getting started with HTML5

- HTML5 is not case sensitive, all the below code is valid

```
<META CHARSET=UTF-8>
<META CHARSET=UTF-8/> ← Case does not matter!
<META CHARSET="UTF-8"/> ← Quotes are optional
<META CHARSET=UTF-8> ← End tags are optional
<meta charset=utf-8>
```

- As a developer you will want to adopt a style and keep it
- Develop house style documents as an organisation now!
  - Flexibility is back but we don't want to make 1990s mistakes
- It is recommended that you keep the XHTML 'style'
  - Yet understand you are working with HTML and not XML

At first look we have taken a massive retrograde step into lax standards of HTML4 and below after all those years of being good XHTML developers. It needs to be reinforced that your browser has never cared about the missing slash from your line break if the data was sent as a **text/html** content type. Only the XHTML validator most developers would run pages through before going live checked this.

QA strongly recommends adopting a house style now and ensuring your organisation sticks to it in its own QA. For instance this course is mostly written as lower case tags with attributes in quotations, but default attributes have been omitted.

## The HTML4/XHTML problem

- **HTML4/XHTML are primarily text layout tools**
  - No context to the data
  - Often just <div> tags
- **Take the following example of a <div> based blog entry**

```
<div class="post"> <
    <h2>My HTML4 Blog</h2>
    <small>15th March 2010</small>
    <div class="entry"> <
        <p>
            The problem with this type of HTML
            is 'divitius'
        </p>
        <p>(That is a technical term)</p>
        <p class="metadata"> <
            <a href="#">Posted in</a>
        </p>
    </div>
</div>
```

You end up  
with a lot of  
DIV tags

Separated only  
by class or id

Formatting  
blocks like P are  
used for layout  
structure

This mark-up is valid, but shows the limitation of HTML4/XHTML; it presents very little real meaning to the data. This is one of the key issues with XHTML: you end up with an attribute-driven set of tags that only make sense when examined alongside the CSS files.

## Why HTML5 markup changes the rules

- **Before HTML5 we had two categories of elements**
  - Inline
  - Block
- **HTML5 provides a more fine-grained set of categories:**
  - Text level semantics - previously *inline tags*, e.g. *strong*
  - Grouping Content - *block level elements*, e.g. *<p> <div>*
  - Forms - *everything inside a form tag*
  - Embedded content - *images, video and canvas*
  - Sectioning content - *the new structural tags*

In HTML4 and XHTML blocks such as a paragraph belong to the heading they follow in the outline. To create a hierachal outline of our content we use a set of h1-h6 tags. They work well in most situations but without sectioning elements, such as *<div>*, they can break down.

Creating document outlines prior to HTML5 was simple. You had six heading elements, *<h1>* through *<h6>*. Lower-numbered headings were of a higher rank of higher-numbered ones, i.e. *<h1>* was ranked higher than *<h2>*:

The concepts behind HTML5 document outlines are actually older than you might think! Tim Berners-Lee posted to the www-talk mailing list back in 1991, suggesting something quite close to what is demonstrated in this article.

The sectioning elements *<section>*, *<article>*, *<aside>* and *<nav>* can all help to create a more logical structure in the document outline. The sectioning elements act quite literally as their name suggests: they define sections of the parent element. These sections can be thought of as child nodes whose headings fall under their parent heading, regardless of their rank.

This is the intent; browsers, search engines and other tools do not yet fully implement these structures yet but HTML5 today is about preparing for the future.

## Understanding the new semantic elements

These tags create **sectioning** elements and are different from <div>

- Sections form part of the document outline when used correctly
- They provide a semantic hierarchy for the document
  - It becomes very powerful when combined with CSS
- All sectioning elements behave the same way in the document outline
  - But provide semantic meaning to the document

```
<section>
    <h2>Your links</h2>
    <ul>
        <li><a href="#">My Friend</a></li>
        <li><a href="#">My Other Friend</a></li>
    </ul>
</section>
```

In the next few slides we are going to examine the new structural elements HTML5 introduces. As we mentioned earlier in the chapter HTML4/XHTML only has two types of elements – block and inline. HTML5 introduces new types, including sectioning blocks.

Over the next few slides we will examine these new tags, but an important concept needs to be raised here. These are not replacements for <div> tags. In effect they provide a closer relationship to <hn> elements. With very few exceptions the first tag after a <section> type is a <hn> element. You will explore these concepts in a later exercise and examine these concepts.

## The HTML5 skeleton

- DOCTYPE is always the first tag, but no URL required

```
<!doctype html>
```

- Required by browsers to trigger a standards mode
- Character encoding is optional but should be present
  - Could expose you to some security vulnerabilities

```
<meta charset="utf-8" />
```

- This is all you need to set encoding
  - content and http-equiv allowed but not required
- Reality check – one day it will be this simple
  - Not until legacy browsers leave active support

First of all, note the simplicity of the doctype code: no need to add in the URL that most HTML authors praise their code editor for supplying. It is an essential tag and most browsers needs it to switch into the right document processing mode.

The encoding is also required. The page will process without it but there are security vulnerabilities that could exploit character encoding vulnerabilities without it including the UTF-7 vulnerability

<http://code.google.com/p/doctype/wiki/ArticleUtf7>

You may be very used to adding the attributes content="text/html" and http-equiv="content-type" these are entirely optional here and QA have opted to take the low attribute usage approach to HTML5.

## Demonstration – the HTML5 skeleton

- **<html> , <head> and <body> are mandatory**
  - <head> has a mandatory child of <title>
- **Open the document with a <html> tag**
  - No xmlns attribute required
  - lang attribute preferred as it helps screen readers
- **Place a <body> tag within as normal**
  - Your favourite structural tags are still here
  - Many new ones to come

The quick exercise proved to us we did not need to add the building blocks of HTML for the page to process normally. Simply the browser can infer this mark-up in HTML. However this is not the case in valid XHTML! So, if they are optional, why add them at all? The answer to this is quite simply maintainability.

Most HTML editors would be rather freaked out by the missing tags and more importantly it makes the page more difficult to understand from a human reader perspective. HTML in any version should always aim for semantic mark syntax. Moving into the looser HTML-based standard will likely cause interesting challenges in coming years and once again the good practice we picked up in XHTML applies a more rigorous approach to testing.

## CSS3

- **New version of CSS**
  - Introduced and approved in modules
  - Allows for more flexibility to be released
  - Extensive new features
- **Selectors**
  - More specific way of selecting elements
  - Matching on attributes and attributes values
  - Structural pseudo-classes
- **Text effects and layout**
  - Hyphenation, “whitespace” and justification of text
- **Paged media and generated content,**
  - More options in pages media
    - Running headers, footers, page numbering, footnotes and cross reference
- **Multi-column layout properties**
- **Ruby modules**
  - Add small annotation on top or next to words
  - Used in Asian scripts

The new version of CSS is introduced and approved in modules which allow for more flexibility to be released. New features of CSS3 are quite extensive:

- Selectors offers a much more specific way of selecting elements, including matching on attributes and attributes values, structural pseudo-classes, target pseudo-class to style only elements that are targeted in the URL, a checked pseudo-class to style any element that is checked such as radio or checkbox element
- Text effects and layout, including hyphenation, “whitespace”, and justification of text
- Paged media and generated content, supporting more options in pages media, such as running headers, footers, page numbering, footnotes and cross reference
- Multi-column layout properties allow for multiple column layouts
- Ruby modules offers ability to add small annotation on top or next to words, used in Asian scripts

## CSS3

### Online resources

- **CSS selector test**
  - <http://tools.css3.info/selectors-test/test.html>
- **Your browser CSS3 support test**
  - <http://fmbip.com/>

After starting the test suite it will automatically run a large number of small tests which will determine if your browser is compatible with a large number of CSS selectors. If it is not compatible with a particular selector it is marked as such. You can click on each selector to see the results, including a small example and explanation for each of the tests.

It is technically not possible to simulate certain user interactions. The test is limited to selectors that are not dependant on user interactions. So this test suite does not include tests for the following selectors: :hover, :active, :focus and :selection.

## JavaScript and the DOM

- **JavaScript and the Document Object Model (DOM)**
  - They play an important role in modern web applications
- **Ability to dynamically interact with elements**
  - Provide rich functionality and interactivity previously found only in desktop applications
- **AJAX (Asynchronous JavaScript and XML)**
  - Removed burden of page refreshes
  - Allow server-side actions to be updated inline
  - Provide a much-improved user experience
- **JSON (JavaScript Object Notation)**
  - Data interchange format for web applications
- **Range of powerful JavaScript framework and libraries**
  - Abstraction of JavaScript that allows developers to worry less about cross-browser consistencies

JavaScript and the Document Object Model (DOM) play an important role in modern web applications.

The ability to dynamically interact with elements enabled developers to provide rich functionality and interactivity previously found only in desktop application.

AJAX (Asynchronous JavaScript and XML) removed burden of page refreshes:

- Allow server-side actions to be updated inline
- Provide a much-improved user experience

JSON (JavaScript Object Notation) has become the de facto data interchange format for web applications.

A range of powerful JavaScript framework and libraries has risen to provide an abstraction of JavaScript that allow developers to worry less about cross-browser consistencies.

## JavaScript and the DOM

### First-class citizen in HTML5

Each section of the specification detailing what DOM API methods and properties are available for any given elements.

HTML5 also defines advanced APIs that allow you to develop applications that use audio and video, store data locally on the client, and do much more.

## The HTML5 DOM APIs

- **DOM APIs exist for nearly everything in HTML5**
- **Many have been around for a long time**
  - But have never been defined in the HTML specification
- **The new DOM APIs in HTML5/HTML5.1:**
  - 2D Canvas
  - Audio and video
  - Drag and drop
  - Web messaging (cross-document and channel messaging)
  - Server-sent events
  - WebSockets
  - Document editing
  - Web storage
  - XML HTTP Request Object (AJAX)
  - Service workers (replace deprecated Offline Application)

## Popular non-HTML5 technologies

- **HTML5 family doesn't stop at the HTML5 specification itself**
- **A host of other technologies and specifications**
- **These APIs include:**
  - Geolocation API
  - Indexed database (IndexedDB API)
  - File, File Reader, File Writer
    - File System APIs (no longer maintained)
  - Scalable Vector Graphics (SVG)
  - Web Graphics Library (WebGL)
- **Other technologies**
  - CSS3
  - Node.js
  - jQuery and other JavaScript Libraries

Some popular specifications and technologies are commonly mistaken for HTML5 because of their intriguing features. Although these new technologies began to emerge around the same time that HTML5 was becoming established and frequently featured in HTML5 Showcase sites and HTML5 books (they're not HTML5 by the definition given earlier). One good way to describe this group of web development technologies, suggested by Bruce Lawson, is "HTML5 and friends."

The best way to keep up with the main HTML specification is to follow The WHATWG Blog (<http://blog.whatwg.org/>). Reading the specification in its raw form can be tedious, to say the least. We find it much easier to read the spec using the edition for web authors, which is available at <http://developers.whatwg.org/>.

For the rest of the specifications there's no central source. Each individual W3C working group has its own blog and/or mailing list. One approach is to keep an eye on the development blogs for the major browsers to find out what new features they're experimenting with:

- *Mozilla Hacks*: <https://hacks.mozilla.org/>
- *Google Chrome Blog*: <http://chrome.blogspot.co.uk/>
- *IEBlog*: <http://blogs.msdn.com/b/ie/>
- *Surfin' Safari*: <https://www.webkit.org/blog/>
- *Opera Desktop Team*: <http://my.opera.com/desktopteam/blog/>
- *Opera Mobile*: <http://my.opera.com/mobile/blog/>

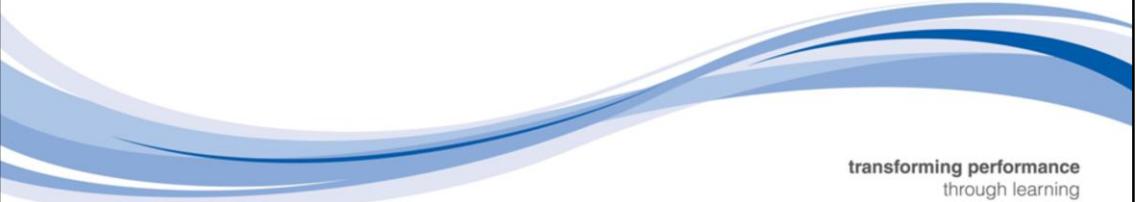
## Review Questions

- 1. What is HTML and what are the major versions?**
- 2. What are some of the advantages of using HTML5?**
- 3. What are some of the new categories in HTML5 elements?**
- 4. How do JavaScript and the DOM play a hugely important role in modern web application?**
- 5. What are some of the new HTML5 DOM APIs and related specifications?**



## HTML5 Markup

Developing Web Applications using HTML5



transforming performance  
through learning

## This chapter covers

- **The new HTML5 semantic structural markup**
  - When to use what
- **Providing fallback for unsupported browsers**
  - Using Modernizer
- **Deprecated HTML Tags and Attributes**
- **Changes to existing HTML elements**
- **New HTML5.0/5.1 elements and attributes**

## In this chapter, you'll learn

- How to use the new semantics elements to lay out a page
- How Modernizr simplifies detection of browser support for various features of HTML5 and conditionally loads fallback
- How to plug gaps in browser support with polyfills, a JavaScript fallback, that will only deploy if the browser lacks native support
- What are new in HTML 5.0 and HTML5.1

## The new HTML5 structural elements

- **HTML5 has a series of new structural elements**
  - To create a more semantically structured page
- **The main building blocks of HTML5 are:**
  - <header>
  - <nav>
  - <section>
  - <article>
  - <aside>
  - <footer>



If you look at some of the more obscure HTML 4 tags like <kbd>, <samp> and <var> you can see the scientific roots of HTML 4. HTML5 is a child of its time for a mainstream, information-based web. If you consider your own pages they are probably littered with <div> tags with semantically useful id attributes such as header, foot, nav and content. To save us from the swarm of <div> tags more suitable structural elements have been added.

As previously discussed, HTML5 introduces a new sectioning set of tags. These allow us to provide semantic description to a page and place the document into a more human readable and better structured document. Over the next pages we will examine these elements and understand how to use them effectively and appropriately.

## Headings and sectioning elements

- **HTML5 documents have an outlining algorithm**
  - Generates a table of content based on the section and heading you've used
  - Important for assistive technology and search engine optimisation
- **Sectioning elements create a logical structure in the document outline**

```
<body>
  <h1>Main heading</h1>
  <p>Some text</p>
  <h2>Level 2 heading</h2>
  <p>Some more text</p>
  <h3>Level 3 heading</h3>
  <p>A bit more text</p>
  <h2>Another level 2 heading</h2>
  <p>The last bit of text</p>
</body>
```



The document outline is the structure of a document, generated by the document's headings, form titles, table titles, and any other appropriate landmarks to map out the document. The user agent can apply this information to generate a table of contents, for example. This table of contents could then be used by assistive technology to help the user, or be parsed by a machine like a search engine to improve search results.

The sectioning elements `<section>`, `<article>`, `<aside>` and `<nav>` can all help to create a more logical structure in the document outline.

## The `<header>` element

- **Normally the first element of the document**
  - Should act as container logos, links back to home etc.
  - Will usually contain a `<h1>` to `<h6>` to denote level of header

```
<header>
  <a href="/">
    
  </a>
  <h1>My Main Title</h1>
  <h2>My Sub title</h2>
</header>
```

- **There can be one `<header>` per sectioning block**

Assuming we are using the `<hn>` tags as is appropriate in modern mark-up development, the main heading will normally have a `<h1>` element within.

The specification says it should be used to represent a group of introductory or navigational aids. It can contain an optional `hgroup` element. It can also be used to wrap a section's table of content, search form or similar.

Essentially when a `<hn>` element on its own is not suitable, there is other related information that makes up the block you should use a `<header>`.

Each block element, be that the `<html>` root or an `<article>` element for example can contain a `<header>` element. You may not have more than one `<heading>` per sectioning block.

The optional `<hgroup>` may only contain `<hn>` elements its purpose is to take a heading element, which would normally appear in the outline, out of it!

## The `<nav>` element

- **`<nav>` is used to mark up navigation**
- **Should be limited to links within the page and site**
  - Not sponsored links for instance
- **Links normally surrounded by `<li>` within a `<ul>`**
- **Multiple `<nav>` allowed, each should contain a related category**

```
<nav>
  <h2>Main site navigation</h2>
  <ul>
    <li><a href="/">Home</a></li>
    <li>
      <a href="/aboutsus.html">About Us</a>
    </li>
  </ul>
</nav>
```

Not all groups of links on a page need to be in a nav element – the element is primarily intended for sections that consist of major navigation blocks.

It is common for footers to have a short list of links to various pages of a site, such as the terms of service, the home page, and a copyright page. The footer element alone is sufficient for such cases; while a nav element can be used in such cases, it is usually unnecessary.

`<nav>` elements can be nested inside other sectioning elements, `<footer>` and `<header>` most commonly.

## The <footer> element

- Many <footer> elements may occur on a page
- Appear at the end of a sectioning element
  - blockquote, body, div etc.
- Contains information about the section or document, e.g. the author

```
<footer>
    <small>This tag has been redefined</small>
</footer>
```

- The <footer> element requires no heading element
  - Unique in the sectioning element
  - It is optional and can be added
- The above code is a 'fat footer'
  - The <small> tag represents small print in HTML5
  - Add a <nav> element within if links required

The footer element represents a footer for its nearest ancestor sectioning content or sectioning root element. A footer typically contains information about its section, such as who wrote it, links to related documents, copyright data, and the like.

When the footer element contains entire sections, they represent appendices, indexes, long colophons, verbose license agreements, and other such content.

A common use of the <footer> element is to create a so-called 'fat footer'. "Fat Footer" are footers that contain a lot of material, including images, links to other articles, links to pages for sending feedback, special offers... in some ways, a whole "front page" in the footer.

## The <article> element

- **Represents a self-contained composition on the page**
  - A blog entry
  - Comic strip
  - Video
- **Articles represent indivisible units of work**

```
<article>
    <h2>Yesterday</h2>
    <p>Some stuff goes here</p>
</article>
<article>
    <h2>Today</h2>
    <p>Some more stuff goes here</p>
</article>
```

The <article> element represents a unit of work that makes up a significant section of data. This may include a <video> element, a blog entry or news story, or results from a dynamic server page.

As is common with all of our sectioning elements the first tag inside an <article> must be a heading.

## The `<aside>` element

- **`<aside>` provides tangential information to a block**
  - It should assist but not be essential to the main document
- **For example nested within an `<article>`**

```
<article>
  <h1>My Blog Post</h1>
  <p>...</p>
  <aside>
    <h1>Glossary</h1>
    <dl> ... </dl>
  </aside>
</article>
```

- **Also used at a page level to denote 'sidebar' content**

```
<aside>
  <h2>Blog roll</h2>
  <ul>
    <li><a href="#">My Friend</a></li>
    <li><a href="#">My Other Friend</a></li>
  </ul>
</aside>
```

The `<aside>` element is an often misused tag in HTML5. Its purpose is to provide data that is tangential to the main block to assist with related but non-essential information. In the first example above it is used to enclose a glossary.

With the new definition of aside, it is important to remain aware of its context. When used within an article element, the contents should be specifically related to that article (e.g. a glossary).

While in the second example which would exist as a child of the `<body>`, the contents relate to the site (e.g. a blogroll, groups of additional navigation, and even advertising if that content is related to the page).

## The `<section>` element

- **<section> is used to break up semantic elements**
  - Different parts to a news story or a group of links
- **Should be used in conjunction with a heading**
  - Otherwise the <section> is untitled in the HTML outline
  - Most generic and easiest to abuse semantic element
  - Do not use as a pure stylistic container

```
<body>
...
<section>
<h2>level of heading = section nesting level</h2>
rest of the content
</section>
...
</body>
```

The section is used to break semantic articles like the `<article>` or `<nav>` into smaller chunks. There are a few simple rules in their use:

- Do not use it as a container for styling to scripting the `<div>` element should be used
- Most generic and least meaningful of the other sectioning elements use them when more appropriate
- The section must be followed by a `<hn>` element

`<sections>` unlike `<div>` provide a child level in the document outline, they are a structural sub level in the hierarchy. A common mistake is new developers to HTML5 think the `<div>` tag is obsolete. As we will discuss shortly when you are using a `<Tag>` as a container presentational rather than informational structure the `<section>` element should not be used.

## The <section> vs. <article> element



ON OUR SAMPLE WEBSITES, WHAT  
WOULD BE MARKED UP AS A SECTION?



LOOK AT THE BBC HOME PAGE. EACH  
COMPONENT IS FROM A DIFFERENT PART  
OF THE SITE: NEWS, SPORTS, AND SO ON.

OK SO EACH OF THOSE IS A SECTION.  
IS THERE AN EXAMPLE THAT SHOWS THE  
DIFFERENCE BETWEEN A SECTION  
AND AN ARTICLE?



LOOK AT THE COMMENTS SECTION ON A  
TYPICAL WORDPRESS BLOG: IT'S A SECTION  
WITH EACH COMMENT AN ARTICLE.



## <div> is still valid HTML5

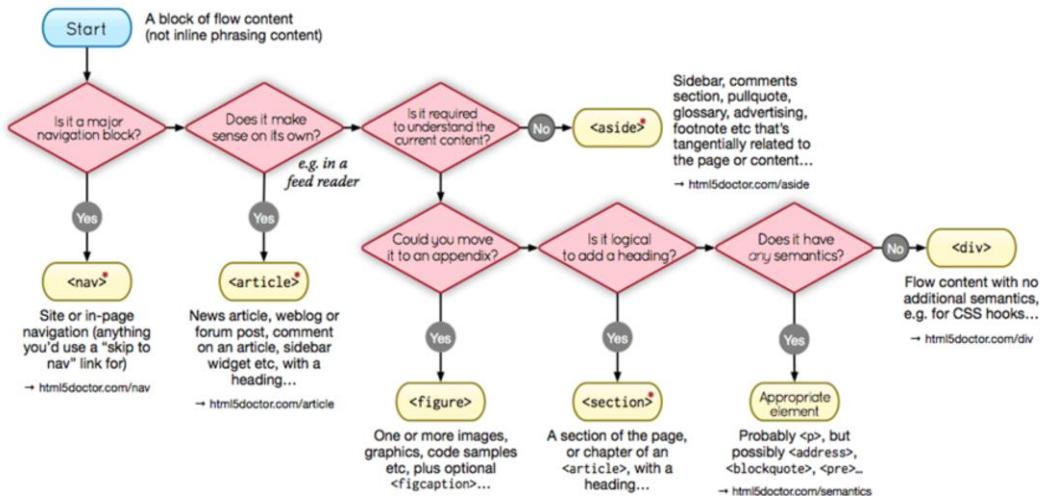
- You can still use the <div> tag no change to it in HTML5
  - I.e. A generic element for structuring a page
  - Has no semantic meaning except via attributes
- Used if no semantic alternative or as a CSS wrapper

```
<body>
    <div id="wrapper">
        <header>...</header>
        <nav>...</nav>
        ...
    </div>
</body>
```

The new semantic elements allow us to replace a lot of <div>'s old functionality. It is not a tag to discount in this new semantic world. The use of <div> is perfectly appropriate if no suitable semantic definition can be found. Its most common purpose is for stylistic functionality where we want the rendered mark-up to look different but not apply meaning.

The same is true of the <span> element. There are may more suitable semantic definitions such as <mark> but its use as a generic content wrapper is still very useful.

## When to use what



## When to use what

Element	Typical Content	Typical Parent and	Child Element
<header>	Title, logo, banner, Introductory information	Body, Section, Article	Nav, Section
<nav>	Primary navigation content	Body	Section, Nav
<section>	Generic page section	Body	Article, Header, Footer, Aside, Nav
<article>	Story, subsection, blog post	Body, Section	Section, Header, Footer
<aside>	Sidebar content, tip, quotation	Body	Section, Article
<footer>	Footer, summary, copyright, info, secondary navigation	Body, Section, Article	Nav, Section

## Restructuring a blog with HTML5

- Each blog entry would be an article

```
<article>
  <header>
    <h2>My HTML5 Blog</h2>
    <p>March 15th 2010</p>
  </header>

  <p>Much less divitius occurs!</p>

  <footer>
    <address>
      <a href=". .">Posted in</a>
    </address>
  </footer>
</article>
```

- Providing clear semantic intent for the page

Here we have implied some intent meaning that the correct semantic tag is an `<article>` specifying a single unit of information. The introductory mater is encapsulated in a header and the footer contains closing navigation. While the main body of the article can go back to using `<p>` for what it was intended!

With the right use of CSS selectors we can easily style these specific articles (more on this later).

Still, there are issues with the article. The data is in a `<p>` for example which does not emphasize the intent of the data being displayed.

## Styling HTML5 with CSS

- **Applying CSS rules to the elements is straightforward**
  - Browsers that understand HTML5 process them as block elements

```
header,nav,footer, article {display:block;}  
nav { float:left; width:25%;}  
article {float:right; width:78%;}  
footer {clear:both;}
```
- **There are a series of legacy issues to be aware of**
  - IE8 or less does not recognise the new semantic elements
  - CSS assumes elements `display:inline`
  - Just setting the widths will not work
  - Must explicitly set `display:block`
- **We need any HTML5 page to be stable in a legacy browser**

Since HTML4, browsers have been able to style any explicitly named element. In fact, at one point the idea was to drop HTML all together and move to XML as the client language. This failed for a number of reasons but until we get fully factored HTML5 rendering engines in our browsers the basis of this technology is very useful.

By default CSS assumes all elements `display:inline` so switching the heights and widths will not work correctly. We must explicitly tell the browsers to display these elements as blocks. This is because the current crop of browsers have a rudimentary style sheet of 'standard' behaviour. If any browser does change their default rules since the release of this courseware the code would be redundant but harmless.

As always seems to be the case, Internet Explorer causes us problems. Currently these elements will not style the way we expect; we will resolve this in the exercise for this chapter.

## Providing fallbacks for unsupported browsers

- **Main drawbacks to HTML5's new feature**
  - Browsers support isn't uniform
- **Use Feature Detection**
  - Testing to see if the browser support a given feature
  - Unfortunately, the approaches vary widely
  - Difficult to remember how to test for each individual feature
  - May wish to load certain external resources
    - Only if the user's browser support (or doesn't support) a given feature

52

One of the main drawbacks to using HTML5's new features is that browsers support isn't uniform.

An important concept when you're working with HTML5's new APIs is that of Feature Detection – testing to see if the browser support a given feature

Unfortunately, the approaches for detecting feature support vary widely, making it difficult to remember how to test for each individual features.

Also, you may wish to load certain external resources only if the user's browser supports (or lacks support of) a given feature.

What's the point in loading a large WebGL support framework if the user's browser doesn't support WebGL.

Or why should we load a colour-picker widget if the user's browser includes a native widget that will be used instead.

## Detecting features and loading resources with Modernizr

### Modernizr

- Purpose-built JavaScript library
- Perform bulletproof feature detection and dynamic loading
- Can detect support for a feature using a much easier syntax
- E.g. to check if the user's browser supports the **Canvas element**

```
if (Modernizr.canvas) {  
    //Canvas is supported, fire one up!  
} else {  
    //Canvas is not supported, use a fallback  
}
```

To detect Canvas support without Modernizr

```
if (!!document.createElement('canvas').getContext) {  
    //Canvas is supported, fire one up!  
} else {  
    //Canvas is not supported, use a fallback  
}
```

## Detecting features and loading resources with Modernizr

### Simple to use Modernizr

- Dynamically load resources (either .js or .css files) based on a feature
- E.g. to determine if the browser support the localStorage API

```
Modernizr.load ({
  test: Modernizr.localstorage,
  yep: 'localStorage.js',
  nope: 'localStorage-polyfill.js'
});
```

- To detect Canvas support without Modernizr

## Using polyfills and Modernizr to plug the gap

### Polyfills (coined by Remy Sharp)

- Implement missing parts of an API specification
- Fill the gaps or crack in various web browsers support for HTML5
- E.g. use Modernizr to load a month-picker polyfill into those browser without a built-in month-picker
  - If the user's browser doesn't support the month input type, load the monthpicker.js

```
Modernizr.load ({
  test: Modernizr.inputtypes.month,
  nope: 'monthpicker.js'
});
```

- Modernizr site
  - "HTML5-Cross-browser-Polyfills"
  - Collection of useful links for many various solutions
  - <https://github.com/Modernizr/Modernizr/wiki/HTML5-Cross-Browser-Polyfills>

55

Polyfills (coined by Remy Sharp) refers to a piece of code that aims to implement missing parts of an API specification.

Use polyfills to fill the gaps or crack in various web browsers support for HTML5.

For example, use Modernizr to load a month-picker polyfill into those browser without a built-in month-picker; if the user's browser doesn't support the month input type, load the monthpicker.js.

Modernizr site has a page "HTML5-Cross-browser-Polyfills, which is a collection of useful links for many various solutions of this kind.

<https://github.com/Modernizr/Modernizr/wiki/HTML5-Cross-Browser-Polyfills>

## Using Polyfills and Modernizr to plug the gap

- **Versions of IE8 and below do not recognise new-in-HTML5 elements**
  - By default, you have to fix this with some JavaScript
  - Or you might use the html5shiv script which includes all the new elements
    - <https://github.com/afarkas/html5shiv>
- **Modernizr does all this for you, so you don't need to include the Shiv**

## HTML5 Boilerplates

- **HTML5 polyfills and developing cross browser CSS is a vast topic**
  - Almost a separate course in its own right!
- **Modernizr helps but we need more, thankfully we have some help**
  - The HTML5 community has created open source boilerplates
    - Mitigates the majority of issues
    - Provides a template to start developing
    - Works in legacy browsers
- **There are a few online resources:**
  - HTML5 Boilerplate - <http://html5boilerplate.com/>
  - Initializr <http://www.initializr.com/>

## List of deprecated HTML tags and attributes

- The following elements are not available in HTML5 and their function is better handled by CSS

<acronym>	<applet>	<basefont>
<big>	<center>	<dir>
<font>	<frame>	<frameset>
<isindex>	<noframes>	<s>
<strike>	<tt>	<u>

- Online resources:**

- [http://www.tutorialspoint.com/html/html\\_deprecated\\_tags.htm](http://www.tutorialspoint.com/html/html_deprecated_tags.htm)

The spec aims to remove browser specific tags and presentational elements which should not be part of the standard.

HTML5 does not support the above tags. Most were deprecated or marked obsolete in XHTML so avoid their usage at all costs.

More importantly and celebratory dance-worthy is the news it will not support frameset!

Some tags we will look at such as <h1>, <b> and <strong> have default appearance within your browser. Remember that appearance must come from the CSS and not the browser default.

### Quick exercise:

- Add <center>Obsolete</center> to a new HTML page.
- View the page in Chrome and right click on the centred text chose inspect element.
- Look to the Computed CSS styles on the right hand side of the inspector note that Chrome is implementing the appearance through CSS anyway.

## Changes to existing elements - <ol>

- In HTML 4 the useful start attribute was deprecated**

- Considered to be presentational
- HTML5 reinstates it

```
<ol start="5">
    <li>Mars
    <li>Jupiter
    <li>Saturn
</ol>
```

- There is also a reversed attribute in HTML5**

- Except IE

```
<ol start="12"reversed="reversed" >
    <li>Coffee</li>
    <li>Tea</li>
    <li>Milk</li>
</ol>
```

This useful little tribute is now back in core HTML5 its usage denotes it is a continuation of a list. It should only be used within a single sectioning block for maximum semantic importance.

## Changes to existing elements - <dl>

- <dl> has been refined in HTML5 as description list
- <dl> can be used to mark-up a glossary of terms, although you must remember to use <dfn> to indicate that the word is defined here
- Definition term <dt>, Definition data <dd>
- <dfn> will italicise the text

```

<aside>
  <h2>Glossary</h2>
  <dl>
    <dt><dfn>RSS</dfn></dt>
    <dd>
      An XML format for aggregating information from websites
      whose content is frequently updated.
    </dd>
    <dt><dfn>Weblog</dfn></dt>
    <dd>
      Contraction of the term "web log", a weblog is a website
      that is periodically updated, like a journal
    </dd>
  </dl>
</aside>

```

The forgotten and unloved definition list makes a revamped return. In HTML4, <dl> was considered a “definition list”, containing groups of terms and their definitions. The terms and definitions were a many-to-many relationship: one or more terms to one or more definitions. The element was often misunderstood and therefore misused or not used at all in favour of more widely used and (perhaps) less semantic mark-up.

To address these issues, <dl>’s definition has been refined in HTML5 as a description list. From the spec:

*The dl element represents an association list consisting of zero or more name-value groups (a description list). Each group must consist of one or more names (dt elements) followed by one or more values (dd elements). Within a single dl element, there should not be more than one dt element for each name.*

It maintains the many-to-many relationship between names and values. These groupings use <dt> to represent the term or name and <dd> for the description. Also note the last line of the quote, stating that a name should not be used more than once within a single <dl>.

## Changes to existing elements <address>

### <address>

- Provides contact information for a document or part of a document
- Includes the names of the document's maintainers, links to the maintainers' Web pages, e-mail addresses for feedback, postal addresses, phone numbers, and so on

```
<section>
    This site is run by the following group of
    volunteers:
        <address>
            <a href="http://html5.com/author/jackc">Jack
            Chan</a>,
            <a href="http://html.com/author/richc">Rich
            Clark</a>,
            <a href="http://html5.com/author/mikew">Mike Wu
        </a>,
        </address>
</section>
```

## Changes to existing elements – fontography

- **Presentational information should come from CSS**
- **Common tags have been redefined in their usage**

Tag	Usage
<i>	Now for text in an “alternate voice,” such as technical terms and typographically italicized text
<b>	For “stylistically offset” text such as keywords and typographically emboldened text
<em>	For stress emphasis i.e., something you’d pronounce differently
<strong>	Now for strong importance

## HTML5 element - <figure>

### <figure>

- **Used in conjunction with the <figcaption> element**
  - Mark up diagrams, illustrations, photos, and code examples
- **Semantically marking up this sort of content directly in our HTML, instead resorting to CSS class names**

```
<figure>
  
  <figcaption>Slightly worn time machine &copy;
    <a href="http://bbc.co.uk">The BBC</a>
  </figcaption>
</figure>
```

The <figure> element is intended to be used in conjunction with the <figcaption> element to mark up diagrams, illustrations, photos, and code examples (among other things). The spec says:

*The figure element represents a unit of content, optionally with a caption, that is self-contained, that is typically referenced as a single unit from the main flow of the document, and that can be moved away from the main flow of the document without affecting the document's meaning.*

The above example has been used on an img tag but this could equally be used on video, canvas or to give comments to a <code> tag.

## HTML5 element - <time>

### <time>

- **Defines a human-readable date/time**
- **Used to encode dates and times in a machine-readable way**
  - User agents can offer to add birthday reminders or scheduled events to the user's calendar
  - Search engines produce smarter search results
- **Has a datetime attribute**
  - Uses a YYYY-MM-DD format

```
<p>I have a date on <time datetime="2016-02-14  
20:00">Valentines day</time>.</p>
```

Time, as defined in the HTML5 specification, represents a time on a 24 hour clock or a date in the Gregorian calendar. The value of time could be displayed as a machine-readable value. For example, HH:MM should be interpreted just fine. More likely the time tag will be accompanied with a **datetime** attribute which has a precise formatting method and must be supplied.

YYYY-MM-DD

A time element can be added by appending a T to the date stream. Then it must be presented on a t2 hour clock format:

2010-03-15T19:02

Optionally you can append seconds by adding a further colon. Adding date and time together requires the use of a time zone using Z for UTC with the time difference so

2010-03-15T12:00:00.001-05:00

Represents noon in New York (UTC -5) March 15<sup>th</sup> 2010.

## HTML5 element - <mark>

- The mark up equivalent of a highlighter pen
- Combine with CSS or JavaScript to highlight content
- Indicates relevance to users activity or request

```
<mark>'divitius'</mark>
```

The mark element represents a run of text in one document marked or highlighted for reference purposes, due to its relevance in another context. When used in a quotation or another block of text referred to from the prose, it indicates a highlight that was not originally present but which has been added to bring the reader's attention to a part of the text that might not have been considered important by the original author when the block was originally written, but which is now under previously unexpected scrutiny. When used in the main prose of a document, it indicates a part of the document that has been highlighted due to its likely relevance to the user's current activity.

Previously em and strong may have been used for adding emphasis or importance to portions of text. This was arguably valid at the time due to the lack of a better element, but the introduction of mark simply means their use will be more strict. Use strong when you need to indicate the importance of a piece of text, such as an error or warning message, and em should be for adding emphasis to text, stressing words to adapt the meaning of a sentence.

## HTML5 element - <ruby>

- Provides internationalised support for Asian language

```
<ruby>日本</ruby>
```

- By adding a <rt> tag inside the <ruby> phonetic alternative is used

```
<ruby>日本<rt>にほん</rt></ruby>
```

- For non-supporting browsers use an optional <rp> tag

```
<ruby>日本<rp>(</rp><rt>にほん</rt></rp>)</rp></ruby>
```

<ruby> provides a useful internationalisation support for Asian languages. Japanese text for example uses a combination of kanji, hiragana and katakana writing symbols. In certain cases the kanji can have more than one pronunciation. To help the reader sometimes the pronunciation is written above the kanji with the phonetic hiragana alphabet. This is called furigana in Japanese and ruby in English from the name of a small font used in British printing.

By applying a ruby text <rt> tag within the <ruby> you tell a supporting browser to display the phonetic above the Kanji. In a non-supporting browser this will not work and the ruby text just appears alongside the kanji. To deal with non-complaint browsers an optional ruby parenthesis tag <rp> exists. This means we can add in parenthesis around the hiragana which would display in a non-complaint browser and display as per the HTML5 spec on a complaint one.

## New HTML5.1 elements

New	Deprecated
1. details	• hgroup (removed in html5)
2. dialog	• keygen
3. main	• tt (removed in html5)
4. menu	
5. menuitem	
6. picture	
7. summary	

## New HTML5.1 elements - <details> & <summary>

- **<details>**
  - A disclosure widget, users can obtain additional information or controls
- **<summary> element child**
  - Represents the summary or legend of the details

```
<summary>MacBook Pro Specification</summary>
<details>
  <ul>
    <li><strong>13.3-inch LED-backlit glossy widescreen display</strong> with edge-to-edge, uninterrupted glass (1280 x 800-pixel resolution).</li>
    <li><strong>2.4 GHz Intel Core i5 dual-core processor</strong> with 3 MB shared L3 cache for excellent multitasking.</li>
    <li><strong>Intel HD Graphics 3000</strong> with 384 MB of DDR3 SDRAM shared with main memory.</li>
    <li><strong>500 GB Serial ATA hard drive</strong> (5400 RPM)</li>
    <li><strong>4 GB installed RAM</strong> (1333 MHz DDR3; supports up to 8 GB)</li>
  </ul>
</details>
```

## New HTML5.1 Elements - <dialog>

- **<dialog>**
  - Part of an application that a user interacts
  - To perform a task, for example a dialog box, inspector, or window
- **“open” attribute**
  - The dialog is active and available for interaction
  - Not set, it shouldn't be shown to the user

```
<dialog open>
    <p>Greetings, one and all!</p>
</dialog>
```

<dialog> represents a part of an application that a user interacts with to perform a task, for example a dialog box, inspector, or window.

“open” attribute - indicates that the dialog is active and available for interaction. When the open attribute is not set, it shouldn't be shown to the user.

## 1.29 New HTML5.1 elements - <main>

### <main> semantic element

- Main content of the body of a document or application
- Can only be used once per page

```
<main>
    <h1>Heading </h1>
    <article>Article 1</article>
    <article>Article 2</article>
</main>
```

The inclusion of a main element (or similar) has long been debated in the working groups with authors and others often questioning why we had new elements such as <header>, <article>, and <footer> but no element to accurately describe the primary content of a page. The proposal was accepted in November 2012, and <main> was then rolled into the HTML5.1 specification. Recently, it was added to the HTML5 specification following no objections.

The primary purpose of <main> is to map ARIA's landmark role main to an element in HTML. This will help screen readers and other assistive technologies understand where the main content begins. The W3C spec describes <main> as representing:

“The main content of the body of a document or application. The main content area consists of content that is directly related to or expands upon the central topic of a document or central functionality of an application.”

Since the <main> element is now included in the HTML specification, the <body> element has reverted to its HTML4 definition:

“The body element represents the content of the document.”

One important facet of <main> is that it can only be used once per page.

## New HTML5.1 elements - <menu> & <menuitem>

- **<menu> element**
  - A list of commands
- **<menuitem> element**
  - Command that the user can invoke from a popup menu (either a context menu or the menu of a menu button)
  - Partial support in Firefox refers to being limited to context menus, not toolbar menus

### **<menu>element**

- A list of commands

### **<menuitem> element**

- Command that the user can invoke from a popup menu (either a context menu or the menu of a menu button)
- Partial support in Firefox refers to being limited to context menus, not toolbar menus

## New HTML5.1 elements - <menu> & <menuitem>

```
<!-- A <div> element with a context menu -->
<div contextmenu="popup-menu">
    Right-click to see the adjusted context menu
</div>

<menu type="context" id="popup-menu">

    <menuitem type="checkbox" onclick="toggleOption()" checked="true">Checkbox</menuitem>

    <menuitem type="command" label="Command" icon="icon.png" onclick="doSomething()">Checkbox</menuitem>

    <menuitem type="radio" name="group1" onclick="option()" checked="true">Radio button 1</menuitem>

    <menuitem type="radio" name="group1" onclick="option()">Radio button 2</menuitem>

</menu>
```

## 1.31 New HTML5.1 elements - <picture>

- <picture>
  - Does not display anything itself
  - Provides a container which provides multiple sources
  - Allows a specific image resource
  - Based on the screen pixel density
    - Viewport size, image format, and other factors
  - Accepts *global* attribute
- Different from the similar *video* and *audio* elements
  - *src* attribute has no meaning when the element is nested within a *picture* element
  - Resource selection algorithm is different

```
<picture>
    <source media="(min-width: 45em)" srcset="large.jpg">
    <source media="(min-width: 32em)" srcset="med.jpg">
    
</picture>
```

<picture> element itself does not display anything; it merely provides a container which provides multiple sources to its contained “*img*” elements to allow a specific image resource, based on the screen pixel density, viewport size, image format, and other factors. It accepts *global* attribute.

<picture> element is somewhat different from the similar *video* and *audio* elements; the source element’s *src* attribute has no meaning when the element is nested within a *picture* element, and the resource selection algorithm is different.

## 1.32 HTML 5.0 Global attributes

These are the new 5.0 global attributes

Name	Purpose
contenteditable	Sets whether the user can edit the content or not.
contextmenu	Used with the currently unsupported <menu> and <command> tags
hidden	Tells the browser not to render content
aria-*	Supports the WAI-ARIA accessibility attributes
spellcheck	Tells the browser to check the element content for spelling and grammar
data-*	Create your own xml based data attributes

Some of these global attributes, such as spellcheck, are so simple to use they are no brainers. If the Browser/OS is capable of running a dictionary then adding the spellcheck attribute to a form or a field makes the UI experience for that user infinitely better. For the legacy browser user it has no effect and the attribute is ignored.

## 1.33 HTML 5.1 attributes

There are also 22 new HTML 5.1 attributes

New attribute	Element
allowfullscreen	Any element
lang	
itemid	
itemprop	
itemref	
itemscope	
itemtype	
translate	
contenteditable	
contextmenu	

## 1.34 HTML 5.1 attributes

There are also 22 new HTML 5.1 attributes

New attribute	Element
command	menuitem
default	
radiogroup	
title	
type	
type	menu
inputmode	input; textarea
menu	button
open	details, dialog
scoped	style
seamless	iframe
sortable	table
sorted	th

## Summary

- **HTML5 provides new semantic way to layout a web page**
- **Use Modernizr to detect missing HTML5 features**
- **Use Polyfills to implement missing part of the specification**

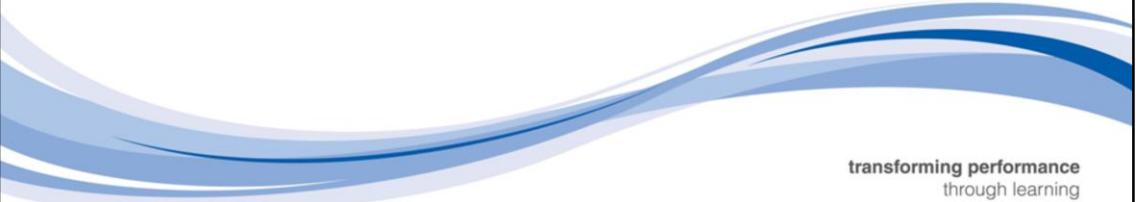
## Review Questions

- **What are the new HTML5.0 structural elements?**
- **How do we provide fallback for unsupported browser?**
- **What are some of the changes to existing elements in HTML4.01?**
- **What are some of the new HTML5.0 elements and when do we use them?**
- **What are some of the new HTML5.0 and HTML5.1 attributes?**



## Introducing CSS3

Developing Web Applications using HTML5



transforming performance  
through learning

## This chapter covers

- **What is CSS3?**
- **CSS Basic**
- **New CSS3 Selectors**
- **The Box Model**
- **CSS Display Mode**
- **Positioning and Layout**
- **New CSS3 Layout Features**

## This chapter you'll learn

- A refresher of the basics of CSS
- How to use new CSS3 selectors
- How to layout a webpage using new CSS3 layout features

## What is CSS3?

**CSS3 is the third major revision of W3C CSS specification**

- **Currently divided into modules**
- **More than 30 individual documents**
- **Progress and mature at their own rates**
- **Many features considered to be CSS3 are actually in CSS2.1**
- **CSS4 currently in early draft stage**

82

CSS3 is the third major revision of the W3C CSS specification. Unlike the previous two revisions, CSS3 is divided into modules. Instead of being one, long document like CSS1 and CSS2/2.1, there are currently more than 30 individual documents that are part of CSS3. These are all allowed to progress and mature at their own rates. Depending on the level of interest, some modules will progress to level 4 before the level-3 work is completed.

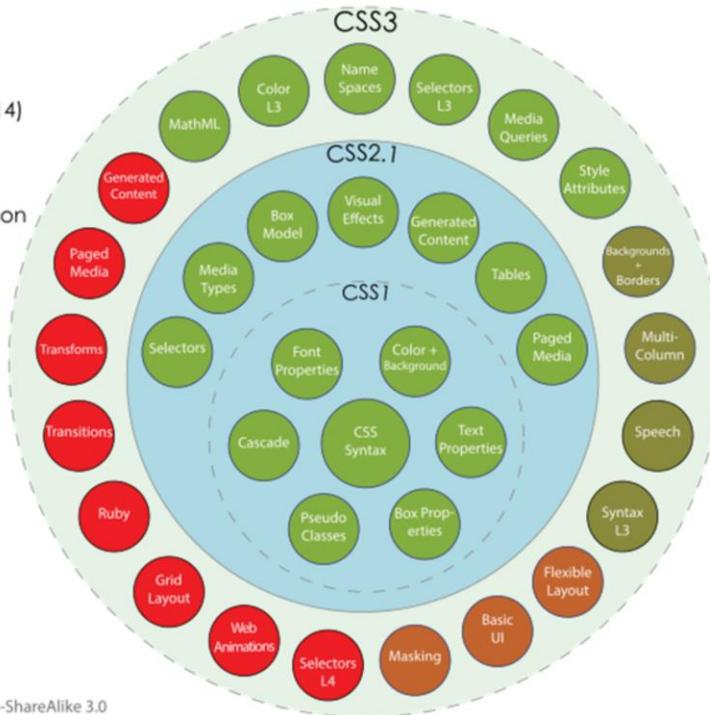
Like the term *HTML5*, the term *CSS3* is often given a wider definition than just the specifications. Many of the features people consider to be CSS3 are actually in CSS2.1. Each generation of CSS builds upon the last, adding new features. CSS4 is currently in early draft stage.

## CSS3 Module Progress

# CSS3

Taxonomy & Status (October 2014)

- W3C Recommendation
- Candidate Recommendation
- Last Call
- Working Draft
- Obsolete or inactive



By Sergey Mavrody 2011-14 | CC Attribution-ShareAlike 3.0

83

## Browser specific rules

- **CSS3 like many parts of the living standard is unstable**
  - May be different ways of doing the same thing
  - May not be supported in all browsers
  - May be legacy issues
- **Vendor specific prefixes introduced to CSS**
  - Intended as a way of testing experimental features
  - To be removed when CSS is ratified and legacy issues resolved

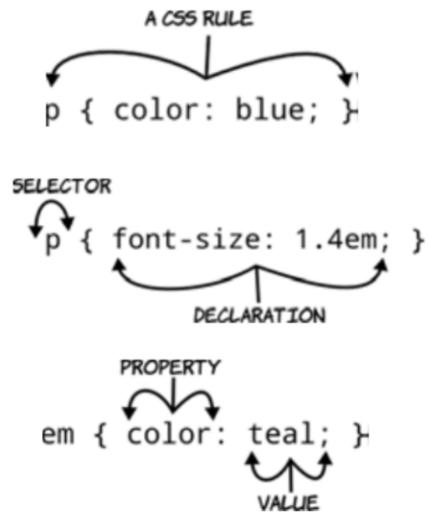
Prefix	Organization
-ms	IE
-o	Opera
-webkit	Safari, Chrome and others
-moz	Firefox and others

Reference: [shouldiprefix.com](http://shouldiprefix.com)

## CSS Basic

### Rules, selectors, properties, and values

- A CSS style sheet is made up of rules
- Here are three examples CSS rules:



A CSS rule is made up of a selector and a semicolon-separated list of declarations inside brackets. Each declaration has a property and a value separated by a colon. If an element in an associated HTML document matches a selector in the style sheet, then the declarations will be applied to that element.

## Adding style sheet to HTML

- **Four ways to include CSS in HTML**
- **Inline style**
  - Apply directly to individual elements with style attribute

```
<p style="color: red;">Another paragraph</p>
```

- **Embedded style**
  - Using <style> element
  - Should appear in the <head> element
  - Rules apply to everything in that page

```
<style> p { color: red; } </style>
```

There are four ways to include CSS in HTML. At the lowest level, you can apply it directly to individual elements with the style attribute. This is known as an *inline style*:

```
<p style="color: red;">Another paragraph</p>
```

Slightly more useful is the <style> element. The <style> element should appear in the <head> element of the HTML document, although all popular browsers will use the styles if they're added to the body instead. Rules in a <style> element apply to everything in that page:

```
<style> p { color: red; } </style>
```

The main benefit of this approach over inline styles is that you can control the styles of multiple elements from a single rule, but they still only affect the page on which they're placed.

## Adding Style Sheet to HTML

- **External Style Sheet**

- Put all CSS in separate file and then link it from each page
- <link> element references an external style sheet
- Should appear in the head of the document

```
<link href="styles.css" rel="style sheet">
```

- **Link from within existing CSS**

- Requires some CSS to already be included
- Use @import
- Should almost never be used; prevent stylesheet from downloading concurrently

```
<style> @import url('styles.css'); </style>
```

On a multiple-page site, the rules would have to be included on every page, so it's far more common to put all the CSS in a separate file and then link to it from each page. This <link> element references an external style sheet:

```
<link href="styles.css" rel="style sheet">
```

Like the <style> element, the link should appear in the head of the document. This same CSS file can be used with every page on the site that links to it. Most browsers download style.css only once and then reuse it, saving bandwidth and page-rendering speed.

The final way to include CSS is to link from within existing CSS. This requires that some CSS has already been included, via a link or a <style> element:

```
<style> @import url('styles.css'); </style>
```

In this example, the style sheet is imported from a <style> element in the head section of the document. From a page speed standpoint, @import from a CSS file should almost never be used, as it can prevent stylesheets from being downloaded concurrently.

## Inheritance

### Key properties of CSS

- **Style inherited down the document tree**
- **You don't have to write style rules for every element in the document**
- **Setting a font or a colour on the <body> element, all text will be that font and colour**
- **CSS can 'cascade': it applies properties/values in order of priority**
- **To style specific elements, you need to learn to write selectors**

```
<style>
body {font-family: "Komika Hand"; font-size: 250%;}
p {color: blue; font-size: 1.4em;}
em {color: teal; }
</style>
```

One of the key properties of CSS is that styles are inherited down the document tree. In the following example, it's already possible to see this feature in action. The font-family property is specified only for the <body> element, but the elements in the document are displayed with the Komika Hand font from that rule. This is because all the other elements are children of the <body> element, so they inherit the font-family property.

Inheritance means you don't have to write style rules for every element in the document. Setting a font or a colour on the <body> element usually means that all text in the document will be that font and colour. To style specific elements, you need to learn how to write selectors; these will be covered in the next section.

## Selecting elements to style

### ID selectors

- Choose an element based on the id attribute
- Should be a unique value; apply only to a single element and its descendants

```
#myelement {  
    color: white;  
    background-color: black;  
}
```

### Class selectors

- Choose elements based on the class attribute
- Doesn't have to be unique; apply to a selection of elements in a document

```
.myclass {  
    color: white;  
    background-color: black;  
}
```

An ID selector chooses an element based on the id attribute. This attribute should have a unique value in any given document, so an ID selector will only ever apply to a single element and its descendants. An ID selector consists of a hash character followed by the id.

By themselves, ID selectors aren't very useful. You certainly wouldn't want to add an ID to every element you needed to style. They're normally used to pick out particular landmarks on a page.

The class selector chooses elements based on the class attribute. Unlike the id attribute, values in the class attribute don't have to be unique throughout the document, so rules based on class selectors usually apply to a selection of elements in a document. A class selector consists of a period (.) followed by the class name and selects any element with the value *myclass* in the class attribute.

## Choosing elements through their relationships

- **Combinators allow you to chain simple selectors together**
- **Descendant Combinator**

```
article p { background-color: #000; }
```

- **Child Combinator**

```
article > p { background-color: #000; }
```

- **Adjacent-Sibling Combinator**

```
h1 + p { background-color: #000; }
```

- **General-Sibling Combinator**

```
h1 ~ p { background-color: #000; }
```

Combinators allow you to chain simple selectors together. They're the workhorses of CSS.

A space is the descendant combinator, selecting any p that is a descendant of an article element.

The greater-than symbol is the child combinator, selecting only the p elements that are direct children of an article.

The + is known as the *adjacent-sibling combinator*. This rule selects any paragraph elements that immediately follows a level-one heading element.

The ~ is known as the *general-sibling combinator*. This rule selects any paragraph elements that follow a level-one heading element.

## Selecting sets of elements with pseudo-classes

- **Selecting first and last element**

```
ul li:first-child { background-color: #000; }

ul li:last-child { background-color: #000; }
```

- **Selecting an element by its ordering**

```
li:nth-child(3), li:nth-child(5) {
    background-color: #000;
}

li:nth-child(2n + 1) { background-color: #000; }

li:nth-child(odd) { background-color: #f00; }

li:nth-child(even) { background-color: #006; }
```

CSS pseudo-classes allow you to select the first element, the last element, or a subset of the elements according to a pattern. They remove the need to add classes to your markup for purely stylistic purposes.

`ul li:first-child { }` This selector is saying, “Select the `<li>` elements that are the first child of their respective `<ul>` elements.”

`ul li:last-child { }` This selector is saying, “Select the `<li>` elements that are the last child of their `<ul>` parent elements.”

With `:nth-child` you can easily select specific elements by specifying a numeric parameter.

## Selecting sets of elements with pseudo classes

- **More selection patterns**

```
ul:last-child { background-color: #000; }
```

```
ul:first-child:last-child { background-color: #000; }
```

- **Selecting by types of element**

```
article:first-of-type { background-color: #000; }
```

```
p:last-of-type { background-color: #000; }
```

- **Choosing what isn't**

```
input:not([type=checkbox]):not([type=radio]) {  
display: block; width: 12em;  
}
```

ul:last-child { background-color: #000; } Without specifying that only <li> elements that were last children should be styled, the rule now selects any element that is a last child.

ul :first-child:last-child { background-color: #000; } This rule selects elements that are both a first and a last child and are descendants of the <ul> element.

article:first-of-type { background-color: #000; } This selector will always apply to the first article element on the page, as well as any other first article elements further down the tree.

p:last-of-type { background-color: #000; } The last-of-type pseudo-class works in the same way, except in reverse.

CSS3 also gives us the ability to select elements that aren't the first child or don't have particular attributes, with the :not pseudo-class.

## Choosing elements by their attribute

- **CSS3 defines three new attribute selectors**
- **`^=` operator finds attributes starting with a value**

```
a[href^="http:"] { color: blue; }
```
- **`$=` operator finds attributes ending with a value**

```
a[src$=".png"] { color: red; }
```
- **`*=` operator finds attributes containing the value**

```
div[id*="stuff"] { color: red; }
```

CSS3 makes it possible to write selectors that target elements based on the values in their attribute. These are called *attribute selectors*.

## Pseudo-elements

- **::first-line**

```
::first-line { background-color: #000; }
```

- **::first-letter**

```
::first-letter { background-color: #000; }
```

- **::selection**

```
::selection {background-color: #b3d4fc; }
```

- In CSS3, pseudo-elements are distinguished by a double colon (::) rather than the single colon of a pseudo-class.

*Pseudo-elements* are CSS selectors that allow you to style certain page elements as if an element existed in your markup.

CSS3 has the ::first-line and ::first-letter pseudo-elements.

::first-line pseudo-element selects the first line of text on the browser.

The ::first-letter pseudo-element is similar, except that it only selects the first letter:

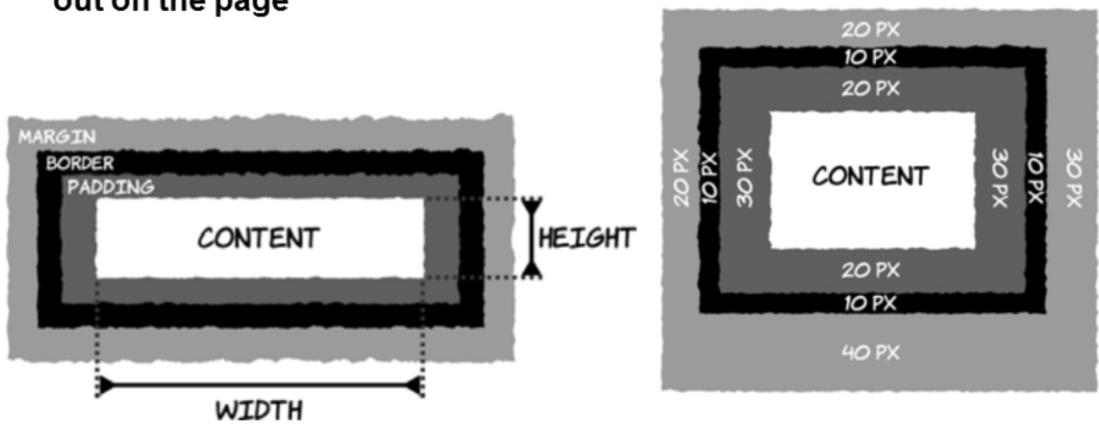
The ::selection CSS pseudo-element applies rules to the portion of a document that has been highlighted (e.g. selected with the mouse or another pointing device) by the user. Only a small subset of CSS properties can be used in a rule using ::selection in its selector: color, background-color, cursor, outline, text-decoration, text-emphasis-color and text-shadow. Note that, in particular, background-image is ignored, like any other property.,

## Pseudo-elements vs. pseudo-classes

Pseudo-elements create virtual elements within your document, as opposed to pseudo-classes, which rely on properties of the document entered by the author. In CSS3, pseudo-elements are distinguished by a double colon (::) rather than the single colon of a pseudo-class. This differs from CSS2, where both used a single colon.

## The box model

- CSS box model defines the dimensions of element as they're laid out on the page



- The width and height correspond to the content box
  - The outer edge defines what is known as total width/height
  - Total width = width + 2\*padding width + 2\*border width + 2\*margin width

The CSS box model defines the dimensions of elements as they're laid out on the page. In order to do page layout with CSS, it's important to know how elements are sized.

Elements have a width and height, padding, a border, and a margin. The element's width is either defined explicitly or determined automatically by the browser based on the content and display mode. Between the content and the border is the padding; then you have the border, and finally the margin, which is the space between this element and the next one.

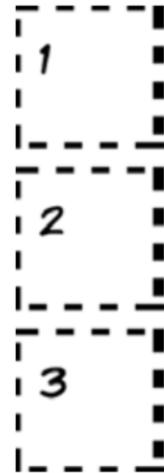
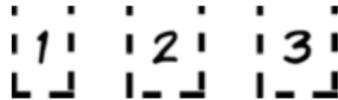
The padding, border, and margin have associated collections of properties in CSS.

## Display modes: inline and block

### ▪ **Inline elements**

- Sit in the flow of text
- Can't have width, height, padding or margin.

```
div { display: inline; }
```



### ▪ **Block elements**

- Cause line breaks
- Have defined width, height, padding and margin

```
div { display: block; }
```

A `<block>` element can have a defined width, height, padding, and margin and causes a break in the text. An `<inline>` element sits on the line of text, but can't have a width, height, padding, or margin.

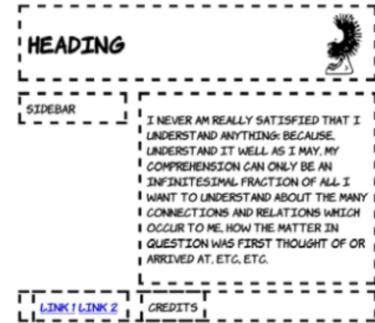
Inline elements sit in the flow of text, whereas block elements cause line breaks before and after. This visual presentation can also be controlled by CSS through the `display` property by setting the value to either `inline` or `block`.

## Positioning and layout

### Floated element

- Element that is outside normal flow of text
- Two CSS properties: float and clear
- Float: determine which side the element float
- Clear: determine how element behaves with respect to other floated element

```
#header img {float: right;}  
#header h1 { margin-right: 150px; }  
#main {float: right; width: 60%;}  
#sidebar {float: left; width: 25%;}  
#footer { clear: both; }  
#footer > div {display: inline;}
```



In the last 10 years, most CSS layouts have been built around floated elements, commonly referred to as *floats*. A floated element is one that's outside of the normal flow of text, like a cut-out. The text flows around these floated elements as long as there's room.

Floats rely on two CSS properties: float and clear. The float property determines which side the element floats to, whereas the clear property determines how the element behaves with respect to other floated elements. Values for float are left, right, and none; values for clear are left, right, both, and none.

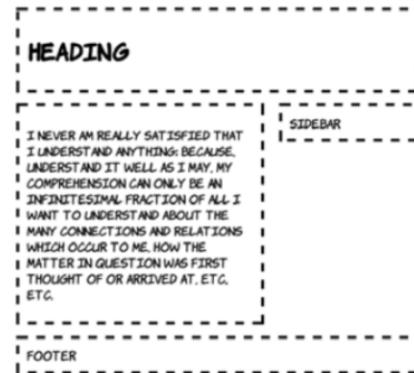
If two consecutive elements are floated the same way and not cleared, then as long as there's available width, they sit alongside each other. But if the second has clear set, it drops below the first element.

## CSS 2.0 display mode – inline-block

### Inline-block element

- **Sits on the line of text**
- **Can be given a specific width, height, padding and margin**

```
header, section, aside, footer{
margin: 2%;
padding: 2%;
outline: 4px dashed black;
vertical-align: top;
}
section, aside {
display: inline-block;
width: 54.5%;
}
aside {
margin-right: 0;
width: 28.5%;
}
```



inline-block is a compromise between `<block>` elements and `<inline>` elements.

An inline-block element combines features of both inline and block element, it sits on the line of text, but it can be given a specific width, height, padding, and margin.

The first benefit of inline-block over floats is that the elements aren't removed from the normal document flow. This means that in a basic two-column layout, it doesn't matter which of the columns is the longest. Any full-width element will automatically be pushed below both columns—no clearing is required.

inline-block also makes the behaviour of grids of items more consistent when there are elements of different sizes. Elements that are inline-block are aligned with the normal character grid, just like lines of text on a page.

## CSS3 improvements to CSS2 layout approaches

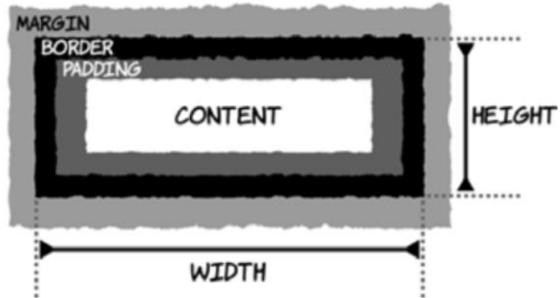
- **calc() function**

- Allows construction of widths from multiple unit

```
section, aside {  
width: calc(70% - 4.665em);  
}  
aside {  
width: calc(30% - 3.665em);  
}
```

- **box-sizing property**

```
#two {  
box-sizing: border-box;  
width: 200px;  
height: 200px;  
border: 25px solid black;  
}
```



One of the major pain points that arise with the CSS layout approaches mentioned in the previous sections is the control of combined width, particularly when mixing percentage and pixel units as you saw with inline-block. CSS3 offers two new features that alleviate this pain:

- A calc() function
- The box-sizing property

The calc function allows the construction of widths from multiple units: for example, 25%—4px. This is useful if several elements need to fit exactly in a percentage width, but each needs to have a border or margin of a certain number of pixels.

box-sizing gives the web author control over the problematic CSS box model: content-box and border-box.

## Using media queries for print

### Media Queries

- Ability to apply different styles based on the output device, whether it's a PC screen, a handheld device, or a printer
- Any CSS rules placed inside the squiggly brackets will only be applied if the conditions are met
- To create your print styles

```
@media print {  
    /* All your print styles go here */  
    #header, #footer, #nav { display: none !important; }  
}
```

CSS has long had the ability to apply different styles based on the output device, whether it's a PC screen, a handheld device, or a printer. These are known as *media queries*. In CSS2 you could also restrict to screen, aural, braille, handheld, or speech, among others. The default, if you don't specify anything, is all the styles will apply no matter what the output device is.

## Using media queries for flexible layout

- **CSS3 extend numbers of properties used**
- **Can choose between different device feature such as screen resolution and window size**
- **Selects based on the browser window size**

```
@media screen and (max-width: 800px) { }
```

- **Select based on display size**

```
@media screen and (max-device-width: 800px) { }
```

Media queries avoid browser detection by letting the browser itself determine what support it has. If a new browser or device comes along that you hadn't anticipated, as long as you've used media queries, it should still select the most appropriate set of CSS rules. CSS3 dramatically extends the number of properties that can be used in media queries.

The most common distinguishing features of different devices are screen resolution and window size. Most desktop users have a browser window at least 800 pixels wide, whereas most mobile browsers are less than 800 pixels wide. Media queries let you choose between the two situations.

The basic syntax for creating a set of rules for a window 800 pixels wide is this:

```
@media screen and (max-width: 800px) { }
```

```
@media screen and (max-device-width: 800px) { }
```

Any CSS rules placed inside the squiggly brackets will only be applied if the conditions are met. The first rule selects based on the browser window size, and the second one selects based on display size—the browser window doesn't have to fill the entire width of the display for this rule to match.

## Changing layout based on orientation and aspect ratio

- **CSS3 provides an orientation media query**

```
@media screen and (min-width: 640px) and orientation:  
portrait) { }  
@media screen and (min-width: 640px) and orientation:  
landscape) { }
```

- **Other features for detection in the spec**
  - Colour, resolution, touch-enabled, device-pixel-ratio
- **Mobile first**
  - Design for the small screen and use media queries to adapt for larger devices

Media queries were used to adapt to lower resolutions. For practical uses, it's often better to do things the other way around. When IE9 is released all the major desktop browsers will support media queries. But IE on Windows Mobile 7 won't, and neither will browsers on older feature phones. If you expect lots of these visitors, design for the small screen and use media queries to adapt for larger devices.

Maybe you want to do different things on a display that's 640 pixels wide and 480 pixels tall compared to one that is 640 pixels wide but 800 pixels tall—you want to know whether the aspect ratio is landscape or portrait for a given width. CSS3 provides an orientation media query for just this situation.

Media queries can be used for more than just screen sizes. There are several other features for detection in the spec, and various browser vendors are introducing more as they add functionality to their browsers.

color – select rules based on the number of bits available per color channel, where 8 bits is 255 levels per color.

resolution – select rules based on the dots per inch (dpi) of the display.

touch-enabled – this is currently a Mobile Firefox–only feature. Select rules based on whether the display is a touch input device, perhaps to give buttons and links more finger space.

device-pixel-ratio – currently a Mobile Safari–only feature. Select rules based on the zoom level, perhaps to provide a higher-resolution background image as the user zooms in so the image remains crisp and sharp.

## The future of CSS layout – flexboxes

### Flexboxes

- Layout approach developed in Firefox
- To produce five equal-size boxes

```
div { width: 90%; display: box; }
div div { box-flex: 1; }
```

- A larger box-flex causes element to take increasing proportion of the spare space

```
div div:nth-child(4) { box-flex: 3; }
```

- Flexboxes allow elements to be displayed in a different order to their position in the markup

```
div div { box-ordinal-group: 2; }
div div:nth-child(5) { box-ordinal-group: 1; }
```

*Flexible boxes*, commonly referred to as *flexboxes*, are a layout approach developed in Firefox to be used for laying out various elements of the user interface.

To produce five equal-size boxes, set the parent element to display:box and set equal box-flex values on the child elements:

```
div { width: 90%; display: box; }
```

```
div div { box-flex: 1; }
```

Setting a larger box-flex value on certain elements causes them to take up an increasing proportion of the spare space:

```
div div:nth-child(4) { box-flex: 3; }
```

Flexboxes allow elements to be displayed in a different order than their position in the markup:

```
div div { box-ordinal-group: 2; }
```

```
div div:nth-child(5) { box-ordinal-group: 1; }
```

Because the fifth child is set to be ordinal-group: 1, it appears before all the elements that are ordinal-group: 2.

## The future of CSS layout – flexboxes

- Flexboxes are horizontal by default, they can also be set to be vertical

```
div { width: 5em; height: 600px; box-orient: vertical; }
```

- Note that in both horizontal and vertical cases, you need to specify a length in that direction in order to get the flex to appear

Flexboxes are horizontal by default, they can also be set to be vertical:

```
div { width: 5em; height: 600px; box-orient: vertical; }
```

Note that in both horizontal and vertical cases, you need to specify a length in that direction in order to get the flex to appear. This is because the flex distributed among the elements comes from the leftover space after the intrinsic size of the element is taken away. This can lead to some counter-intuitive results when the elements with flex don't have a well-defined intrinsic width.

## The future of CSS layout – multi-column layouts

### CSS multi-column

- Allow easy definition of multiple columns of text
- Just as newspapers
- column-count property - sets the number of columns to a particular number

```
#col { -moz-column-count: 2; -webkit-column-count: 2;  
column-count: 2; }
```

- column-width property - sets the minimum desired column width

```
#wid { -moz-column-width: 100px; -webkit-column-width:  
100px; column-width: 100px; }
```

- column-gap property - gap between columns

```
#column_gap { -webkit-column-count: 5; -moz-column-  
count: 5; column-count: 5; -moz-column-gap: 2em; -  
webkit-column-gap: 2em; column-gap: 2em; }
```

The CSS multi-column layout extends the *block layout mode* to allow the easy definition of multiple columns of text. People have trouble reading text if lines are too long; if it takes too long for the eyes to move from the end of the one line to the beginning of the next, they lose track of which line they were on. Therefore, to make maximum use of a large screen, authors should have limited-width columns of text placed side by side, just as newspapers do.

Two CSS properties control whether and how many columns will appear: column-count and column-width.

The column-count property sets the number of columns to a particular number.

```
#col { -moz-column-count: 2; -webkit-column-count: 2; column-count: 2; }
```

The column-width property sets the minimum desired column width. If column-count is not also set, then the browser will automatically make as many columns as fit in the available width.

```
#wid { -moz-column-width: 100px; -webkit-column-width: 100px; column-width:  
100px; }
```

It is often convenient to use the shorthand columns. The two CSS declarations column-width:8em and column-count:12 can be replaced by:

```
#columns_12 { -moz-columns: 12 8em; -webkit-columns: 12 8em; columns: 12  
8em; }
```

## The future of CSS layout – multi-column layouts

### Shorthand

The two CSS declarations `column-width:8em` and `column-count:12` can be replaced by.

```
#columns_12 { -moz-columns: 12 8em; -webkit-columns: 12  
8em; columns: 12 8em; }
```

There is a gap between columns. The recommended default is 1em. This size can be changed by applying the `column-gap` property to the multi-column block:

```
#column_gap { -webkit-column-count: 5; -moz-column-count: 5; column-count:  
5; -moz-column-gap: 2em; -webkit-column-gap: 2em; column-gap: 2em; }
```

## Summary

In this chapter, you've have learned

- Many new features in CSS3 for selecting elements based on their relationship with their parent
- Pseduo-classes for selecting elements based on their relationship with their sibling
- Attribute selectors that reduce your dependence on class and id attribute
- Some of the murky past of CSS layout
- New CSS layout feature such box-model, calc, media queries, flexboxes, and multi-column layouts

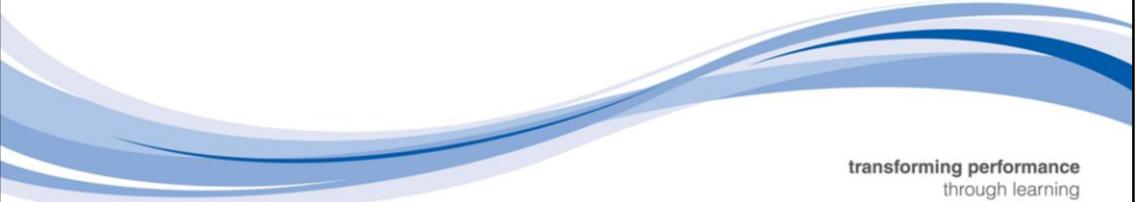
## Review Questions

- **What are the different browsers specific rules?**
- **What are the different ways to add style sheets to HTML?**
- **How do we choose elements through their relationships?**
- **What is the difference between inline and block level elements?**
- **What are some of the new CSS3 features to layout a web page?**



## Learning CSS3 Features

Developing Web Applications using HTML5



transforming performance  
through learning

A decorative graphic at the bottom right of the slide features several overlapping, curved bands in shades of blue and white, resembling waves or flowing lines.

## This chapter covers

- **CSS3 in motion and colour**
- **Borders and backgrounds with CSS3**
- **CSS3 for text and fonts**

## In this chapter you will learn

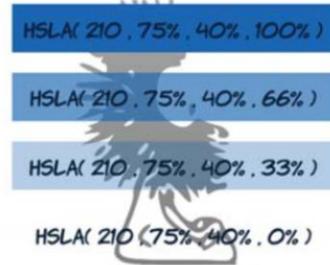
- How to make elements and colours semi-transparent with opacity property, RGBA and HSLA
- How to create natural user interaction with transitions and animation
- How to manipulate borders with CSS3
- How to use new features in web font

## Colour and opacity

- CSS3 provides several ways of specifying colour values that have a level of transparency
- **rgba()**
  - RGB expressed as values between 0 and 255.
  - Alpha is the last parameter and has a value between 0 and 1

background: rgba(102, 102, 102, 0.5);

- **hsla()**
  - Hue – basic color
  - Saturation – intensity of colour
  - Luminosity – how light or dark the colour
  - Alpha – opacity



112

CSS3 includes several new features for colours, RGBA, HSL, and HSLA.

rgba() – Red, Green, Blue, Alpha(Opacity)

HSL stands for *hue*, *saturation*, and *luminosity* in the same way that RGB stands for *red*, *green*, and *blue*. The basic colour is provided by the hue,

and the saturation determines the intensity of the colour – lower saturation means more grey. The luminosity determines how light or dark the

colour is. HSL has a semi-transparent equivalent: HSLA.

## CSS transforms

- **CSS3 transform property**
  - Scale, rotate, translate and/or skew any element on the page
- **Scale**

```
-webkit-transform: scale(1.5);  
-ms-transform: scale(1.5);  
transform: scale(1.5);
```



- **Rotate**

```
-webkit-transform: rotate(16.5deg);  
-ms-transform: rotate(16.5deg);  
transform: rotate(16.5deg);
```



- **Translate (prefix required)**

```
transform: translateX(50px);  
transform: translateY(100px);  
transform: translate(50px, 100px);
```



The CSS3 transform property lets you lets you translate, rotate, scale, and/or skew any element on the page.

Transforms require vendor prefixing for IE9, Android up to 4.4.3, iOS8, and Blackberry 10.

## CSS3 Transform

### Skew (prefix required)

```
transform: skewX(16.5deg);  
transform: skewY(33deg);  
transform: skew(16.5deg, 33deg);
```



## CSS Transformations

- Latest browsers support 3D transformations
- A nice taste of the future however!

Property	Description
<code>translate3d(x, y, z)</code> <code>translateZ(z)</code>	Move the element in x, y and z
<code>scale3d(sx, sy, sz)</code> <code>scaleZ(sz)</code>	Scale the element in x, y and z
<code>rotateX(angle)</code> , <code>rotateY(angle)</code> , <code>rotate3d(x, y, z, angle)</code>	The first two forms simply rotate the element about the horizontal and vertical axes
<code>perspective(p)</code>	This function allows you to put some perspective
<code>matrix3d(...)</code>	This function allows you to specify the raw 4×4 homogeneous transformation

WebKit now has support for CSS 3D transforms, which allows you to position elements on the page in three-dimensional space using CSS. This is included in iOS from version 2.0.

There are several new transform functions available for use in the -webkit-transform property:

### **translate3d(x, y, z), translateZ(z)**

Move the element in x, y and z, and just move the element in z. Positive z is towards the viewer. Unlike x and y, the z value cannot be a percentage.

### **scale3d(sx, sy, sz), scaleZ(sz)**

Scale the element in x, y and z. The z scale affects the scaling along the z axis in transformed children.

### **rotateX(angle), rotateY(angle), rotate3d(x, y, z, angle),**

The first two forms simply rotate the element about the horizontal and vertical axes. Angle units can be degrees (deg) radians (rad) or gradians (grad). The last form allows you to rotate the element around an arbitrary vector in 3D space; x, y and z.

### **perspective(p)**

This function allows you to put some perspective into the transformation matrix. For an explanation of p, see below.

### **matrix3d(...)**

Free hand 3d transforms, one for the 3D maths gurus

## CSS Transitions

- **CSS transforms can be scaled with new transitions**
  - No prefix required
- **Transitions gradually change from one style to another**
  - To do this, you must specify two things:
    - Specify the CSS property you want to add an effect to
    - Specify the duration of the effect.

```
transition: width 2s;
```

- **Pick out individual CSS properties to transform**

```
transition: width 2s, height 2s, transform 2s;
```

- Or use the `all` keyword to change all simultaneously

```
transition: all 2s linear 1s;
```

CSS transitions are an exciting way of transforming appearance without using JavaScript. The two key parts of the transition are the style you are transforming and the duration marked in either seconds (s) or milliseconds (m). It is essential to enter a duration as the default 0 seconds.

A full list of the properties that can be transformed can be located here:

<http://www.w3.org/TR/css3-transitions/#properties-from-css->

The third optional parameter is the timing-function value.

- `ease`
- `linear`
- `ease-in`
- `ease-out`
- `ease-in-out`
- `cubic-bezier` (which allows you to define your own timing curve)

The transition can be tied to selectors such as `:hover` and `:active` for quite impressive results.

## CSS Transitions

**There are additional transition properties available:**

Property	Description
transition	A shorthand property for setting the four transition properties into a single property
transition-property	Specifies the name of the CSS property to which the transition is applied
transition-duration	Defines the length of time that a transition takes. Default 0
transition-timing-function	Describes how the speed during a transition will be calculated. Default "ease"
transition-delay	Defines when the transition will start. Default 0

For example the below code changes the width, but the mouse would need to hold over the attached object for 2 seconds

```
transition-property: width;
transition-duration: 1s;
transition-timing-function: linear;
transition-delay: 2s;
```

```
/* Safari and Chrome */
-webkit-transition-property:width;
-webkit-transition-duration:1s;
-webkit-transition-timing-function:linear;
-webkit-transition-delay:2s;
```

Alternatively the transition could be added as a 4<sup>th</sup> parameter to a transition:

```
transition: width 1s linear 2s;
/* Firefox 4 */
-moz-transition:width 1s linear 2s;
/* Safari and Chrome */
-webkit-transition:width 1s linear 2s;
/* Opera */
-o-transition:width 1s linear 2s;
```

## Key frame animation

- **@keyframes allows more sophisticated animation**
  - First define the animation

```
@keyframes YOUR-ANIMATION-NAME
{
    0%   { opacity: 0; }
    50%  { opacity: 0.5; }
    100% { opacity: 1; }
}
```

- **Then apply the animation against a selector:**

```
#box {
    animation: YOUR-ANIMATION-NAME 5s infinite;
}
```

- **Using @-webkit-keyframes and –webkit-animation for chrome and safari**

The first unusual thing you'll notice about any CSS3 animation code is the keyframes @ rule. According to the spec, this specialised CSS @ rule is followed by an identifier (chosen by the developer) that is referred to in another part of the CSS.

The @ rule and its identifier are then followed by a number of rule sets (i.e. style rules with declaration blocks, as in normal CSS code). This chunk of rule sets is delimited by curly braces, which nest the rule sets inside the @ rule.

Then you use the animation-name property for the CSS block you wish to animate. The value of this property must match an identifier in an existing @keyframes rule, otherwise no animation will occur. In some circumstances, you can use JavaScript to set its value to none, the only keyword that has been reserved for this property, to prevent an animation from occurring.

## Key frame animation

The animation can be refined using additional properties:

```
#box.animate {  
    animation-name: YOUR-ANIMATION-NAME;  
    animation-duration: 10s;  
    animation-timing-function: ease;  
    animation-iteration-count: 1;  
    animation-direction: normal;  
    animation-delay: 5s;  
    animation-play-state: running;  
}
```

The animation-iteration-count property is set to 1, meaning that the animation will play only once. This property accepts an integer value of infinite.

The animation-direction property is set to normal (the default), which means that the animation will play in the same direction (from start to finish) each time it runs. In our example, the animation is set to run only once, so the property is unnecessary. The other value we could specify here is alternate, which makes the animation play in reverse on every other iteration. For the alternate value to take effect, the iteration count needs to have a value of 2 or higher.

The animation-delay property, which does exactly what you would think: it allows you to delay the animation by a set amount of time.

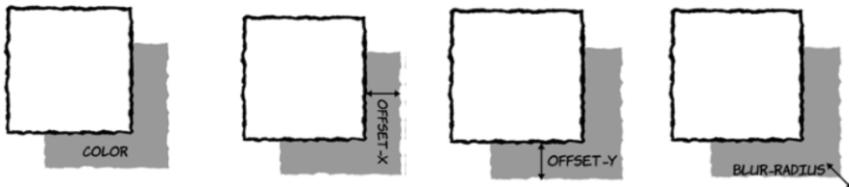
The animation-play-state property, which might be removed from the spec, accepts one of two possible values: running and paused. This value has limited practical use. The default is running, and the value paused simply makes the animation start off in a paused state, until it is manually played. You can't specify a paused state in the CSS for an individual keyframe; the real benefit of this property becomes apparent when you use JavaScript to change it in response to user input or something else.

## Drop shadows with CSS3

CSS3 defines two types of shadow: **box and text**

- A basic shadow, in either case, is defined by four values:
- `<color> <offset-x> <offset-y> <blur-radius>`

```
box-shadow: rgb(0,0,0) 3px 3px 3px;
```



```
text-shadow: rgb(51,51,51) 0px 6px 6px;
```

The diagram shows the text "Text Shadow" inside a dashed rectangular frame, demonstrating the effect of a text shadow.

CSS3 defines two types of shadow: **box and text**. They use a similar syntax:

`text-shadow: rgb(0,0,0) 3px 3px 3px;`

`box-shadow: rgb(0,0,0) 3px 3px 3px;`

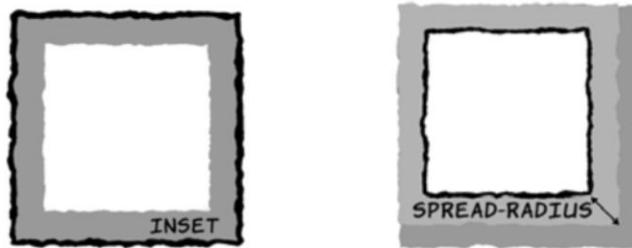
A basic shadow, in either case, is defined by four values:

`<color> <offset-x> <offset-y> <blur-radius>`

## Drop shadows with CSS3

The full box-shadow definition includes two additional, optional elements:

- `<inset> <color> <offset-x> <offset-y> <blur-radius> <spread-radius>`
- inset, if present, puts the shadow inside the element instead of outside
- spread-radius is a CSS length: it causes the shadow to grow (positive values) or shrink (negative values) relative to the size of the element



The full box-shadow definition includes two additional, optional elements:

`<inset> <color> <offset-x> <offset-y> <blur-radius> <spread-radius>`

## Easy rounder corners

### border-radius property

- Can accept up to four values
- Apply starting from the upper-left corner and proceed clockwise around the element

```
#myBox {  
background-color: #999;  
border: 6px solid #000;  
border-radius: 40px;  
}
```



Border radius

```
#myBox {  
background-color: #999;  
border: 6px solid #000;  
border-radius: 40px 160px  
80px 120px;  
}
```



Border radius

## Creating gradient with CSS

- A simple linear gradient – specify the direction and two colours

```
background: linear-gradient(  
  to bottom, #000, #fff  
) ;
```



- Diagonal gradients

```
background: linear-gradient(  
  to bottom left, #000, #fff  
) ;
```



- Direction can also be specified in degrees

```
background: linear-gradient(  
  315deg, #000, #fff  
) ;
```

A simple gradient is easy – specify the direction and two colours. The browser calculates a gradient across the entire background, treating the first colour as the starting colour and the second as the end colour:

```
background: linear-gradient(  
  to bottom, #000, #fff  
) ;
```

Maybe you want something other than up and down or left and right. You can combine the two to get diagonal gradients:

```
background: linear-gradient(  
  to bottom left, #000, #fff  
) ;
```

The direction can also be specified in degrees.

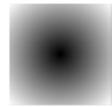
```
background: linear-gradient(  
  315deg, #000, #fff  
) ;
```

## Creating gradient with CSS

### Radial Gradients

- Supply a start and an end color

```
background: radial-gradient(  
#000, #fff  
) ;
```



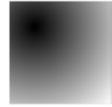
- "at" keyword is used to specify the centre point

```
background: radial-gradient(  
at top, #000, #fff  
) ;
```



- The gradient center can be positioned anywhere

```
background: radial-gradient(  
at 25% 25%, #000, #fff  
) ;
```



Radial gradients are as simple as linear gradients. You supply a start colour and an end colour:

```
background: radial-gradient(  
#000, #fff  
) ;
```

You can achieve more interesting effects by positioning the centre of the gradient:

```
background: radial-gradient(  
at top, #000, #fff  
) ;
```

The at keyword is used to specify the centre point.

The gradient centre can be positioned anywhere:

```
background: radial-gradient(  
at 25% 25%, #000, #fff  
) ;
```



## Gaining control of fonts with the @font-face rule

### @font-face rule

- Allows you to specify a font to be downloaded with the web page

```
@font-face {  
    font-family: "Liberation Sans Bold";  
    src: url(LiberationSans-Bold.ttf)  
        format("truetype");  
}
```

- Reference them in CSS rules like any other font

```
h1 { font-size: 32px; font-weight: normal; }  
div:nth-child(1) { font-family: "Liberation  
Sans Bold"; }
```

- Different browsers support several different font file formats

- EOT, TTF/OTF, WOFF, and SVG.
- For widest support across browsers, provide all four formats

The @font-face rule allows you to specify a font to be downloaded with the web page in the same way as markup, images, and script. Here's a basic declaration to download the Liberation Sans Bold font:

```
@font-face {  
  
    font-family: "Liberation Sans Bold";  
  
    src: url(LiberationSans-Bold.ttf) format("truetype");  
  
}
```

Now that the downloadable fonts have been defined, you can reference them in CSS rules like any other font:

```
h1 { font-size: 32px; font-weight: normal; }  
  
div:nth-child(1) { font-family: "Liberation Sans Bold"; }
```

Different browsers support several different font file formats, among them EOT, TTF/OTF, WOFF, and SVG. For widest support across browsers, you need to provide your font in four different formats.

Many online services have appeared in recent years to simplify getting the fonts you want on your website.

- <https://www.fontsquirrel.com/>
- <https://fonts.google.com/>

## Online resource for CSS3

- **CSS3 generator**
  - <http://css3generator.com/>
  - <http://www.css3.me/>
- **CSS3 gradient**
  - <http://www.colorzilla.com/gradient-editor/>
  - <http://www.cssmatic.com/gradient-generator>
- **CSS3 transformation and transition**
  - <http://css3gen.com/css-3d-transform/>
- **Font face builder**
  - <https://www.fontsquirrel.com/>
- **CSS prefix check**
  - <http://shouldiprefix.com/>

## Summary

- **CSS3 provides several ways of specifying colour values that have a level of transparency**
  - `rgba()`
  - `hsla()`
- **CSS3 transform property**
  - Scale, rotate, translate and/or skew any element on the page
- **CSS transforms can be scaled with new transitions**
- **@keyframes allows more sophisticated animation**
- **CSS3 defines two types of shadow: box and text**
- **border-radius property for rounder corner**
- **Create gradient with linear-gradient and radial-gradient**
- **@font-face rule allows you to specify a font to be downloaded with the web page**

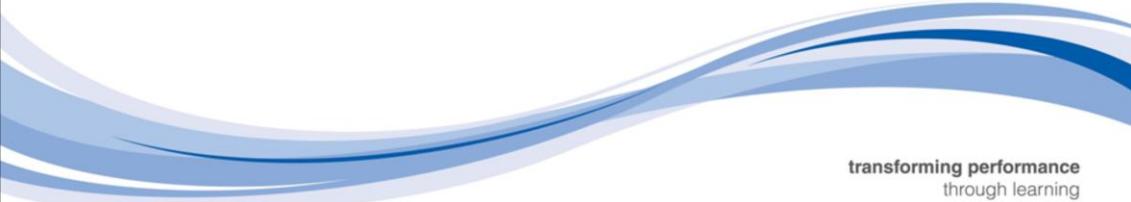
## Review questions

- How do you specify colours with CSS that have a level of transparency?
- What are the different CSS transform property?
- What are the two things you have to specify CSS transitions?
- What kind of gradient can you create with CSS?
- What is the purpose of @font-face?



# JavaScript For Developer

Developing Web Applications using HTML5



transforming performance  
through learning

A decorative graphic consisting of several overlapping, curved blue and white lines that create a sense of motion and depth.

## This chapter covers

- **Arrays**
  - What are arrays
  - Creating arrays
  - Accessing arrays
- **Functions**
  - What are functions
  - Creating functions
  - Calling functions
- **Scope**
  - What is scope
  - Functions and scope
- **Objects**
  - Creating objects
  - Accessing objects

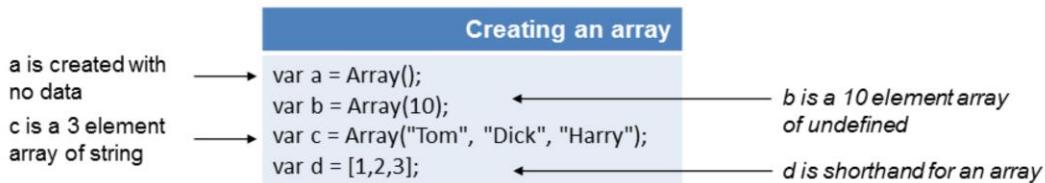
## In this chapter you will learn

- How to create and use arrays and functions in JavaScript
- The important of scope in JavaScript
- How to create and use objects in JavaScript

## Creating arrays



- Arrays hold a set of related data, e.g. **students in a class**
  - The default approach is accessed by a numeric index



- Arrays in javascript have some idiosyncrasies

- They can be resized at any time
- They index at 0
  - So Array(3) would have element 0, 1 and 2
- They can be sparsely filled
  - Unassigned parts of an array are undefined
- They can be created in shorthand using just square brackets

132

As with the rest of JavaScript, the elements in arrays are loosely-typed. This means that each separate element of an array can contain a different type. This is useful and dangerous at once and brings us back to the consideration that type matters.

Array indices in JavaScript start at 0. Arrays have a `length` property equal to the length of the array; note a array of length 6 will have elements 0 to 5.

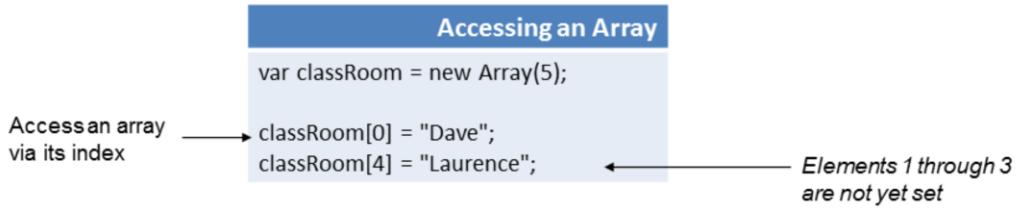
JavaScript arrays are *sparse* – this means that there can be holes in an array. The example above shows an array which initially has values assigned to elements number 0 and 5. This results in an array which contains only two elements. Elements numbered 1 to 4 exist, but if referenced will yield the special value `undefined`.

Note that once a value has been assigned to an element of array, that array element is now defined, and cannot be removed. Assigning `null` to it changes the value, but does not remove the array element. In fact, simply *referring* to the element is enough to create it.

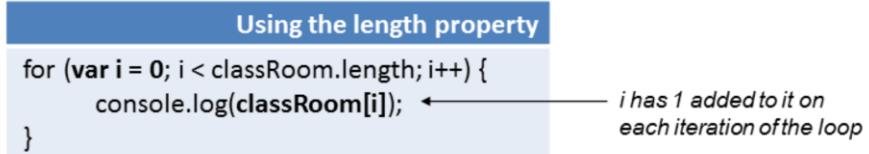
Arrays in JavaScript will grow dynamically from its initial size. It is also possible to shrink an array by assigning a smaller value to the `length` property than it currently holds. Any elements with indexes greater than or equal to the new value should no longer be assumed to exist.

## Accessing arrays

- Arrays are accessed with a square bracket notation



- Arrays have a **length** property that is useful in loops:



133

Arrays allow us to store a related set of data. Arrays store data in a list of elements; we access a particular elements by specifying the name of the array and the element index within square brackets, e.g.

```
myArray[0];
```

The first element of the array is always accessed through [0].

In the above example, a 5 element array is created and elements 1 through 3 are not set. In this situation they will have a value of **undefined**. If you remember what we have learnt from the module on types this makes perfect sense. They have been created but not initialised.

Arrays also have a length property. This will always reflect the exact number of elements an array contains. This is immensely useful as you can see as it allows us to create loops that will always run as many times as is needed.

## Functions – about

- **Functions are one of the most important concepts in JavaScript**
  - This is our first pass, we have a full module on them later
- **Functions allow us to block out code for execution when we want**
  - Instead of it running as soon as the browser processes it
  - It also allows us to reuse the same operations repeatedly
    - Like `console.log()`;
  - Functions are first-class objects and are actually a type of built-in type
    - The keyword function actually creates a new object
      - of type `Function`

134

In JavaScript a Function is a type. It is a first-class object. Whenever we declare a function we are actually declaring a variable of type Function with the name of the function becoming the name of the variable.

When we invoke a function we are actually calling a method of that function (a function on the function) called call; the invocation syntax is simply sugar.

## Functions – creating

- The function keyword is used to create JavaScript functions

Function is a language keyword



- Parameters may be passed into a function

**Passing a parameter**

```
function sayHelloToSomeone(name){
    alert("Hi there " + name + "!");
}
```

- It may optionally return a value

**Returning a value**

```
function returnAGreetingToSomeone(name){
    return "Hi there " + name + "!";
}
```

135

Functions are created with the reserved word function. The newly created function block has mandatory curled braces and a function name. In the first example above, a simple function has been created that calls an alert and takes no input.

In example two we are passing in a value that is known as a parameter or argument. It is at this point an optional parameter. If it was passed in blank it would be undefined as a type. In the exercise we will consider the idea of passing in the parameter in and checking its value.

The first two functions just go off and do 'something' they would often be known as sub routines. They are effectively a diversion from the main program allowing us to reuse code.

The third example uses the return keyword, it calculates or achieves some form of result and then returns this result to the program. It would normally be used in conjunction with a variable as we will see on the next slide.

## Functions – calling

### Functions once created can be called

- Use the function name
- Pass in any parameters, ensuring the order
- If the function returns, pass back result

#### Calling functions

```
sayHelloToSomeone("Dave");

var r = returnAGreetingToSomeone("Adrian");
```

- Parameters are passed in as value based
  - The parameter copies the value of the variable
  - For a primitive this is the value itself
  - For an object this a memory address

136

The function must be declared before it is called. It must be declared in the current block, a previously rendered block or an external script file you have referenced. Else the function call will raise an error.

Parameters need to be passed in in the correct order and remember that JavaScript has no type checking externally. We would need to be sure what is going into the function call is not rubbish. Any parameters passed in are local copies and do not refer to the original variable.

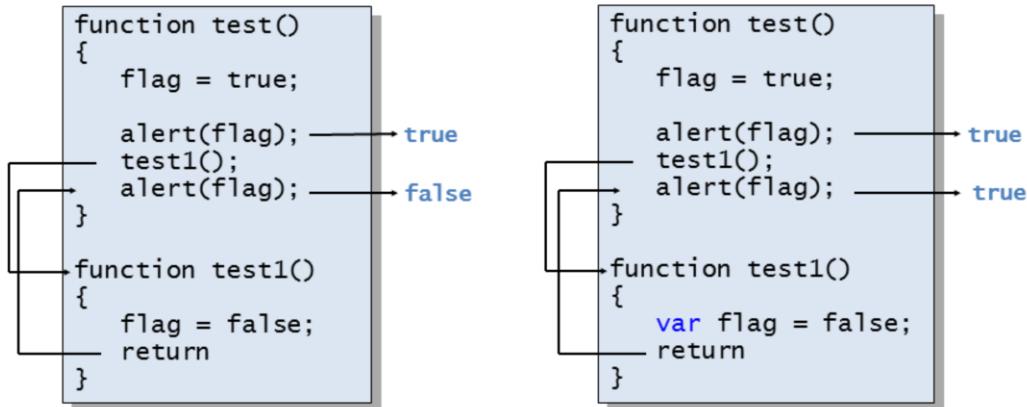
The return statement has two purposes in a function. The first is as a method of flow control: when a return statement is encountered, the function exits immediately, without executing any code which follows the return statement. The second, and more important, use of the return statement is to return a value to the caller, by giving the function a value.

If a function contains no return statement, or the return statement does not specify a value, then its return value is the special undefined value, and attempting to use it in an arithmetic expression will cause an error.

## Functions – scope(1)

**Scope defines where variables can be seen**

- **Use var keyword to specify scope**
- **Sets scope to the current block**
- **If you don't use var then variable has 'global' scope**



137

The **var** keyword has additional semantics: it provides ‘scoping’ of variables. Variables declared outside the body of a function are always global as expected. Less obvious is that variables created inside a function body are also global. However, variables created explicitly using by the **var** keyword have scope limited to the enclosing function (no matter how deep within the function they are created).

Variables created outside of a function body using the **var** keyword are global. Note that this means global to the html page, not just the current SCRIPT block.

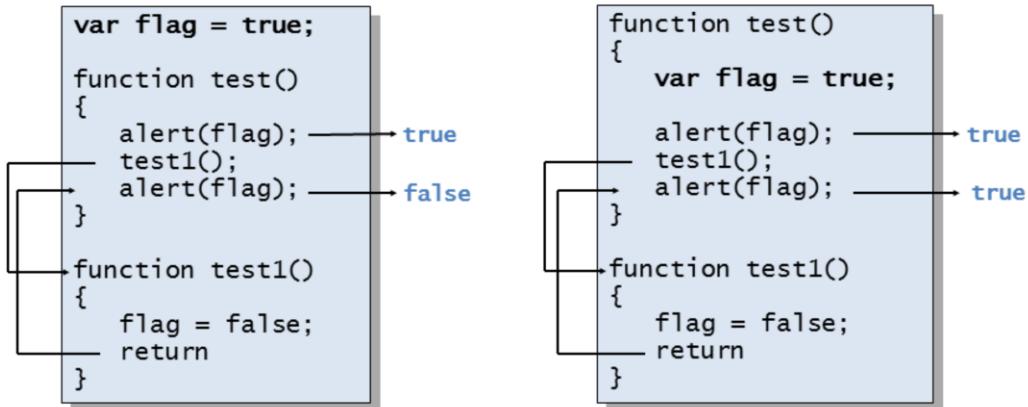
In the first example in the slide, setting the value of `flag` to `false` in function `test1()` has the effect of changing the value of the global variable and hence `flag` in function `test()`.

In the second example, marking the variable `flag` as **var** in function `test1()` restricts its scope to that function only. This means that setting the value of the local `flag` variable does not effect the value of `flag` set in function `test()`.

Improper used of scoping can be a source of major debugging headaches.

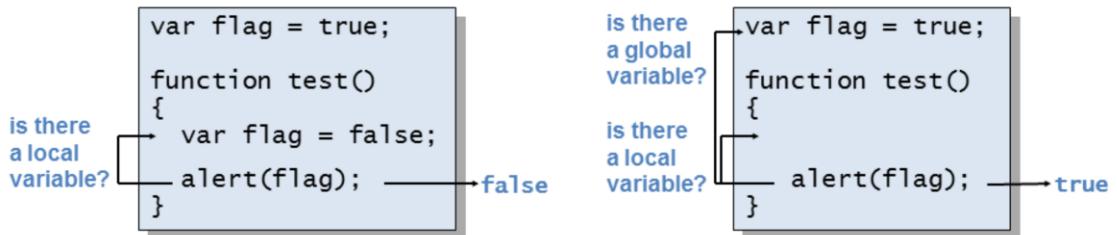
## Functions – scope(2)

- In the code sample on the left:
  - `flag` variable is not explicitly defined at global level
- In the code sample on the right it is declared in the scope of `test`
  - Can `test1` see it?



## Functions – local vs. global scope

- Scope chains define how an identifier is looked up
  - Start from inside and work out



- What happens if there is not a local or global variable?
  - One is added to global scope!

139

The scoping rules and the way in which identifiers are resolved follow an inside to outside flow in JavaScript.

If this code is executing in a function, then the first object to be checked is the function itself: local variables and parameters. After that the 'global' store of variables is checked.

We discuss functions more later, but while we are talking about scope, variables declared in a formal parameter list are implicitly local.

```
function test(flag)      // This flag has local scope
{
  flag = false
}
```

## The global object

- **Global object for client-side JavaScript is called window**
  - Global variables and functions belong to the window object
  - Global variables are actually properties of the current window
  - Global functions are actually methods of the current window
  - Current window reference is implicit
- **Unless you create a variable within a function it is of global scope**
  - The scope chain in JavaScript is interesting
  - JavaScript looks up the object hierarchy, not the call stack
    - This is not the case in many other languages
  - If a variable is not seen in scope it can be accidentally added to global
    - Like the example in the previous slide

```
var a = 7;  
alert(b);
```

```
window.a = 7;  
window.alert(b);
```

These are equivalent

## Objects – data structures

- **Objects in JavaScript are key – value pairs**
  - Where standard arrays are index – value pairs
  - Keys are very useful for providing semantic data
  - Known as associative arrays or object literals

### Creating an Associative Array

```
var student = new Object();  
  
student["name"] = "Caroline";  
student["id"] = 1234;  
student["courseCode"] = "LGJAVSC3";
```

- **The object can have new properties added at any time**
  - Known as an expando property

### Using an expando property

```
student.email = "caroline@somewhere.com";
```

141

JavaScript is an object oriented language. It provides a series of input objects we will shortly be examining that include window and document and their numerous offspring are very important, but they are defined by the browser, not by the programmer. We can define our own objects.

Objects are principal objects in JavaScript. If the variable does not refer to a primitive type it is an object of some kind. In principal they are very similar to the concepts of arrays, but instead of an indexed identifier a string based key is used.

This is one of JavaScript's secrets! In JavaScript, objects are also associative arrays (or hashes). That is, the property

**student["name"]**

...can also be read or written by calling

**student.name**

Thus, you can access each property by entering the name of the property as a string into this array. Such an array associates each key with a value (in this case the key Home is associated with the value normal).

## Objects – accessing properties

- The key part of an object is often referred to as a property
  - It can be directly accessed

### Accessing a property

```
student.email ;  
student["email"];
```

- When working with objects the `for in` loop is very useful
  - `key` holds the string value of the key
  - `student` is the object
  - So it loops for each property in the object

### The for in loop

```
for (var key in student) {  
    console.log(key + " : " + student[key]);  
}
```

142

Objects are collections of properties and every property gets its own standard set of internal properties. (We can think of these as abstract properties – they are used by the JavaScript engine but they aren't directly accessible to the user. ECMAScript uses the `[[property]]` format to denote internal properties.)

One of these properties is `[[Enumerable]]`. The `for-in` statement will iterate over every property for which the value of `[[Enumerable]]` is true. This includes enumerable properties inherited via the prototype chain. Properties with an `[[Enumerable]]` value of false, as well as *shadowed* properties (i.e. properties which are overridden by same-name properties of descendant objects) will not be iterated.

## Objects – literal notation

- There is an alternative syntactic approach to defining objects

### Object Literal Notation

```
var student2 = { name: "David", id: 1235, courseCode: "LGJAVSC3" };
```

- This can be combined into more complex arrays:

- Below is an indexed array containing two associative objects
- Note the comma separator

### Object Literal Notation

```
var classRoom = [
    { name: "David", id: 1235, courseCode: "LGJAVSC3" },
    { name: "Caroline", id: 1234, courseCode: "LGJAVSC3" }
]
```

143

The above code is a quick implementation for JavaScript object. It initialises the object and sets three properties.

The second example creates an indexed array of associative objects.

## Summary

- **Arrays, functions and objects are very important concepts**
  - Most real programming involves reuse of code
    - With minimum repetition
  - Functions and arrays allow this to happen
- **JavaScript is an object-based language**
  - Everything is an object behind the scenes
  - Many very useful objects built into JavaScript
- **We will revisit all three concepts through the course**

## Review questions

- **How do you create Array in JavaScript?**
- **What is the purpose of Function in JavaScript?**
- **How do you specify Scope in JavaScript?**
- **Why is key-value pair in Object useful?**
- **How do you loop through an Object?**



## Advanced JavaScript

Developing Web Applications using HTML5



transforming performance  
through learning

## This chapter covers

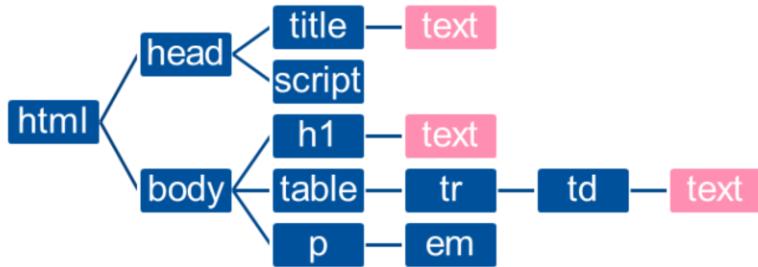
- **What is the DOM**
  - The DOM and HTML tree
- **Selecting elements**
  - Basic selectors
  - CSS3 Selector patterns
- **Arrays of selected objects**
- **Advanced function design**
  - Closures
    - What are closures and how do they work?
    - Using closures to simplify development
  - Self-executing functions
- **Robust JavaScript design**

## This chapter you'll learn

- How to access the DOM using JavaScript
- How to create new element in JavaScript
- How to do use Closure and create Self-executing function in JavaScript

## What is the Document Object Model

- HTML documents have a hierarchical structure that form the DOM
  - Every element, except <html> is contained within another
  - Creating a parent/child relationship



- A DOM tree contains two types of elements
  - Nodes
  - Text

149

HTML documents have what is called a hierarchical structure. Each element (or tag) except the top <html> tag is contained in another element, its parent. This element can in turn contain child elements. You can visualise this as a kind of organisational chart or family tree.

HTML has valid rules for how this DOM fits together and renders the mark-up delivered from the server into a client side DOM. For all intents and purposes the DOM is a complex array.

JavaScript provides a series of inbuilt functions to manipulate the structure of the HTML in your browser. It is fair to say that nothing rendered in the page is safe from your grubby little mitts as you start to traverse the page with JavaScript. You can manipulate existing items and create new ones.

## HTML markup to DOM object (1)

- Consider the following HTML

```

```

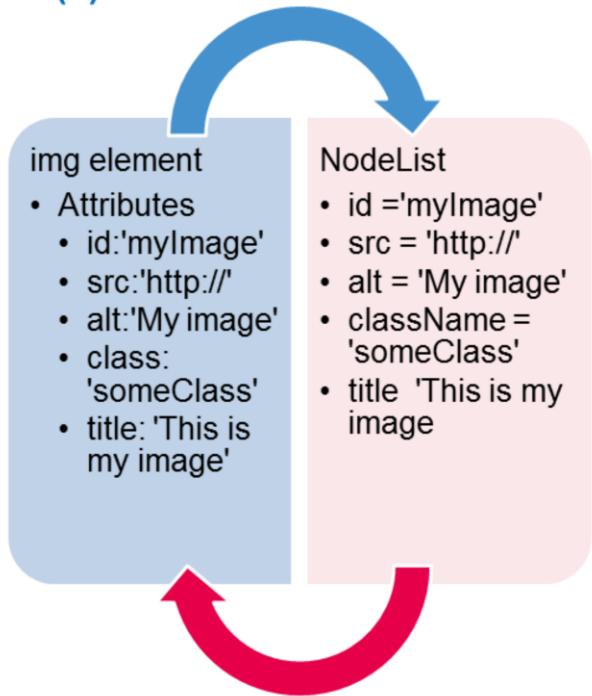
- The tag has a type of `<img>` and four attributes
  - `id`
  - `src`
  - `alt`
  - `title`
- The element is read and interpreted by the browser into a DOM
  - Each element becomes a `NodeList` object
  - Assigned a property based on the `html` attribute

HTML gives the browser instructions on how to render the order and set the properties of the objects. Each element in HTML can have zero or more attributes assigned to it. So in the example above, each element has properties and attributes assigned to it.

These properties and attributes are initially assigned to the DOM elements as a result of parsing their HTML mark-up and can be changed dynamically under script control.

## HTML markup to DOM object (2)

**HTML is translated into DOM elements, including the attributes of the tag and the properties created from them.**



For the most part, the name of a JavaScript attribute property matches that of any corresponding attribute, but there are some cases where they differ. For example, the class attribute in this example is represented by the className attribute property.

Every element mapped into the DOM is in fact an object literal of key value pairs as we discussed in the previous chapter. If we can select the element we want from the DOM we can manipulate it in exactly the same way.

## Selecting elements

**HTML DOM elements can be selected via JavaScript**

- Single elements can be selected in the following ways:

**document.getElementById()**

```
var x = document.getElementById('id');
```

**document.querySelector()**

```
var y = document.querySelector('#id');
```

- Multiple elements can be selected using the following approaches:

**document.getElementsByTagName()**

```
var allP = document.getElementsByTagName('p');
```

**document.querySelectorAll()**

```
var allA = document.querySelectorAll('div > a');
```

To manipulate a DOM object you need to gain access to it. To achieve this we create a variable that points to the required object. Up until recently this was achieved with a set of elements of the document object starting with getElement...

ECMAScript introduces a new way of doing things using the **querySelector** approach. These methods leverage the CSS3 selector engine. The power and flexibility of these selectors are astounding. However, legacy browsers (IE7<) do not support these methods and they will not work. We will look at strategies to deal with legacy browsers later in the course.

## Attribute selectors example

- **Use attribute selectors with care as they can be expensive**
  - A complex search pattern

- **$^=$  operator finds attributes starting with a value**

```
document.querySelectorAll('a[href^="http"]');
```

- **$$=$  operator finds attributes ending with a value**

```
document.querySelectorAll('a[href$=".doc"]');
```

- **$*$ = operator finds attributes containing the value**

```
document.querySelectorAll('a[href*="name"]');
```

## Creating new content – innerHTML and innerText

- In the olden days IE broke the DOM programing standard
  - All browsers now support `innerHTML` and `innerText`
- These functions allow us to add to the DOM in a quick and dirty way
  - The entire JavaScript string is parsed into a HTML element
  - Beware that older browsers can face injection attacks!

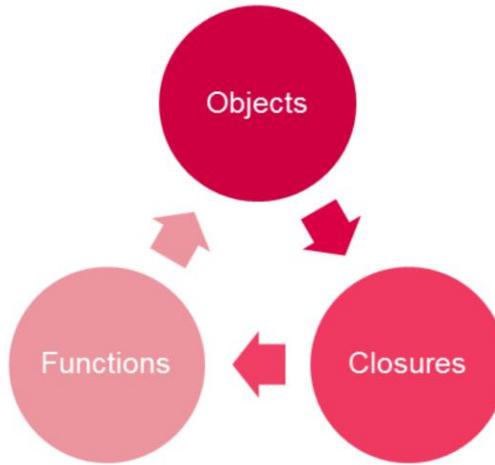
### innerHTML and innerText

```
var el = document.querySelector('#id');
el.innerHTML = "<em>cool</em>";
```

## Closing in on closures

JavaScript has a close relationship between objects and functions

- Along with arrays, they are first-class elements of the language
- Understanding this relationship improves our JS programming ability



155

JavaScript consists of a close relationship between objects, functions – which, in JavaScript, are first-class elements – and closures. Understanding the strong relationship between these three concepts vastly improves our JavaScript programming ability, giving us a strong foundation for any type of application development.

Traditionally, closures have been a feature of purely functional programming languages. Having them cross over into mainstream development has been particularly encouraging, and enlightening. It's not uncommon to find closures permeating JavaScript libraries, along with other advanced code bases, due to their ability to drastically simplify complex operations.

## What are closures?

- A closure is the scope created when a function is declared
  - It allows the function to access variables external to itself
  - It allows a function to use variables as if they were in the same scope
- This may seem rather intuitive until you remember
  - A declared function can be called at any time
  - Even after the scope that declared it has gone out of scope

A simple closure

```
var outerValue = 'stuff';
function outerFunction() {
    Assert is true → console.assert(outerValue == "stuff", "I
        can see stuff");
}
outerFunction();
```

- Function and variable are global objects added to `window`

156

In this code example, we declare a variable `outerValue` and a function `outerFunction` in the same scope. Afterwards, we cause the function to execute. The assert condition returns true as we are able to see the `outerValue` variable.

Closures allow a function to access all the variables as well as other functions that are declared in the same scope that the function itself is declared.

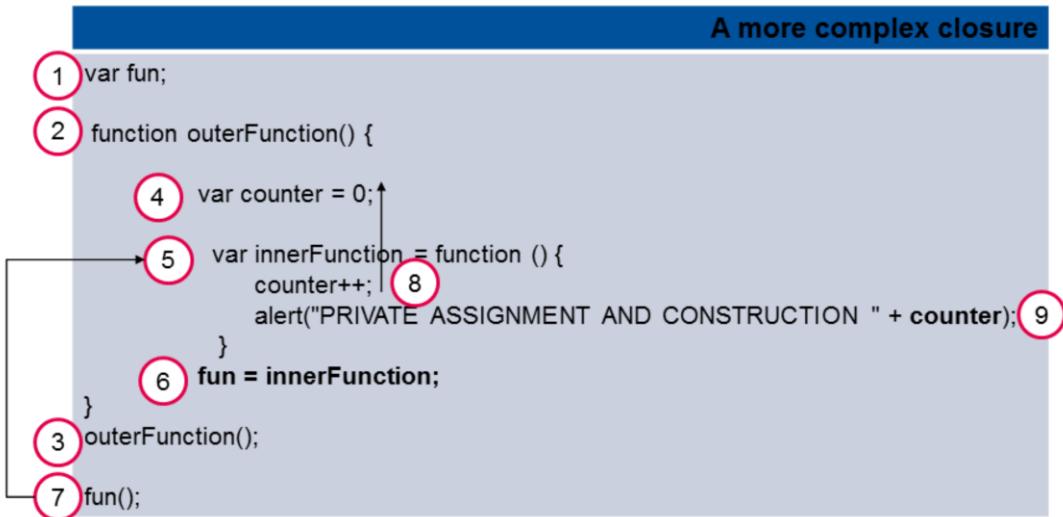
That may seem rather intuitive until you remember that a declared function can be called at any later time, even after the scope in which it was declared has gone away.

Remembering that every variable, object and function is contained within an object. Both the outer value and the outer function are declared in global scope (window object); that, as long as the page is loaded, that scope never goes away.

So, in this scenario we have created two global resources. What would be the outcome of the code `var outerFunction = "hello";` ?

## Walkthrough – more complex closures

- A more complex closure – what happens?



- What is the value of `counter` if we call `fun()` again?

157

We are going to explore closures in a little more depth here:

1. We declare a variable called `fun` – what is its current value?
2. We declare a function called `outerFunction` – remember that functions are objects.
3. **`outerFunction`** is called by the global stack.
4. Within `outerFunction`'s body, a number primitive is created with an id of **`counter`**.
5. A function called `innerFunction` is created and a reference to the heap object is passed to the `innerFunction` variable.
6. `Fun` copies the value of `innerFunction`, that value is a memory pointer to the `innerFunction` function.
7. The function now exits in most scenarios ‘`counter`’, and ‘`innerFunction`’ goes out of scope and would be destroyed as they are no longer needed. In this instance, that is not the case. `fun` now contains the address of the `innerFunction` object. For as long as `fun` is in scope the object created on the heap has a valid memory reference. As a result, it remains alive.
8. So the call stack now moves to the function we have created and invokes it. In turn, this function adds one to the value of `counter`. This is where we see the real power of closures. `Counter` was not released either because `innerFunction` holds a reference to it.
9. As a result, we have an enclosed counter. `Fun` knows about `innerFunction` and `innerFunction` knows about `counter`. While `fun` remains in scope so do `innerFunction` and `counter`.

## Why do we need closures?

- Until now, we have declared everything globally
  - This is potentially dangerous as we could destroy data
  - Consider how many times we have used `x` and `i` as variables
  - Were you sure they were out of scope when you assigned to them?

### Oops there goes the neighbourhood

```
function everythingImportant(params) {  
    .... lots of lines of code  
}  
  
var everythingImportant = [];
```

- In the above example, the function is wiped out and replaced
  - There is no recovery other than to reload the page

158

The case above talks about the danger of accidentally overriding a function or variable and causing a massive amount of damage to your code. We are also being very wasteful. Consider how many times you have used `document.querySelectorAll` to create an array of objects ready to hook up to events as an example. What do we use that array for as long as our page is live? What an amazing waste of memory!

Closures are part of the answer, but we would still have a vulnerability as the function that encloses it could still be wiped out. We can solve that problem with self-executing functions.

## Self-executing functions

- Some of closure's most amazing features are self-executing functions

### The self-executing function pattern

```
(function(params) { alert("Do Stuff") ; }) (params)
```

- This single pattern of code is incredibly versatile, but odd!
  - Consider how `(3 + 4) * 5` works
  - The first brackets are doing the same – it is a precedence delimiter
    - They contain a named or anonymous function
  - The second brackets optionally contain parameters
- This pattern creates, executes and discards a function in one block
  - It helps to keep our global scope uncluttered
  - Limiting scope chain and parameter availability issues

159

This single pattern of code is incredibly versatile and ends up giving the JavaScript language a huge improvement in power and performance. Its syntax, with all those braces and parentheses, may seem a little strange, so let's deconstruct what's going on step by step.

First, let's ignore the contexts of the first set of parentheses, and examine the construct:

`(...())`

We know that we can call any function using the `functionName()` syntax, but in place of the function name, we can use any expression that references a function instance. That's why we could call a function from a variable that refers to a function using the `variableName()` syntax. As in other expressions, if we want an operator (in this case the function call operator `()`) to be applied to an entire expression, we'd enclose that expression in a set of parentheses.

So in `(...())`, the first set of parentheses is merely a set of delimiters enclosing an expression while the second set is an operator.

The result of this code is an expression that creates, executes, and discards a function all in the same statement. Additionally, since we're dealing with a function that can have a closure like any other, we also have access to all outside variables in the same scope as the statement.

## Using self-executing functions

Using self-executing, we can build up interesting enclosures

- Examine the following code

```
self-executing function  
(function () {  
    var numClicks = 0;  
    document.addEventListener("click", function () {  
        alert(++numClicks);  
    }, false);  
})();
```

- The function is executed immediately
- The click handler is bound right away
- numClicks is enclosed in a document level event listener
- It is now a private variable

160

Using immediate functions, we can start to build up interesting enclosures for our work. Because the function is executed immediately, and all the variables inside of it are confined to its inner scope, we can use it to create a temporary scope within which our state can be maintained.

Remember variables in JavaScript are scoped to the function within which they were defined. By creating a temporary function, we can use this to our advantage and create a temporary scope for our variables to live in.

As the immediate function is executed immediately (hence its name), the click handler is also bound right away. Additionally, note that a closure is created allowing the **numClicks** variable to persist with the handler but nowhere else. This is the most common way in which immediate functions are used, just as a simple self-contained wrapper for functionality.

## Parameterising self-executing functions

- The last example was only ever good for the document object

- We should parameterise it so the pattern can be reused
- Use the second set of brackets in the function to do this

### Parameterised self-executing function

```
(function (el) {
    var numClicks = 0;
    el.addEventListener("click", function () {
        alert(++numClicks);
    }, false);
}) (document.getElementById('d1'));
```

- We are passing the result of `getElementById` into the function
  - No reference to anything pollutes the global scope

161

The anonymous function now takes parameters. Use the second set of brackets to pass parameters into the global function as you normally would.

## Summary

- **HTML documents have a hierarchical structure that form the DOM**
- **JavaScript provides several ways to select the DOM**
- **Closures**
  - Closures give us a great way of managing scope
  - Avoids extending the window object unnecessarily
- **Self-executing functions**
  - A great side effect of the closure pattern
  - The key to advanced JavaScript

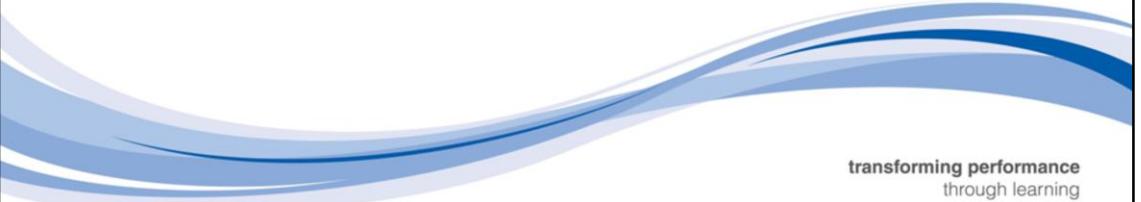
## Review Question

- **What is the DOM?**
- **How do you select single and multiple elements in JavaScript?**
- **What are the quick and dirty ways to create elements in JavaScript?**
- **What is the construct of a self-executing function?**
- **How do you parameterised a self-executing function?**



## Introducing jQuery

Developing Web Applications using HTML5



transforming performance  
through learning

## This chapter covers

- **What is jQuery?**
- **Selecting elements with jQuery**
- **Manipulating properties with jQuery**
- **Events with jQuery**

## In this chapter you will learn

- **What jQuery is**
- **How to use jQuery to simplify client-side coding**

## What is jQuery?

- **A freely available JavaScript library**
  - <http://www.jquery.com>
  - Developer and release formats
- **Cross-browser compatibility functionality**
  - Works almost everywhere (providing JavaScript is enabled)
- **Key features include**
  - CSS3 selector patterns
  - Compatibility checking functions
  - jQuery UI
  - A plethora of plugins!

JavaScript is a notoriously tricky language often because of the different and non-standardised way that browsers process the scripting language. jQuery is a library built by a team that understand the way that browsers work, extracting us from many of the cross and legacy browser issues. jQuery allows us to write JavaScript once and benefit many times. Just in terms of dealing with the legacy event handling of IE6 vs. the rest of the world is a mammoth time saving functionality.

It has a series of key concepts we will be exploring during the course that include CSS3 type selectors. These selectors can be used in jQuery even for browsers that do not support CSS3.

We will explore the jQuery UI: a sister project put together to provide rapid development of modern WEB2.0 style applications.

Plus we will explore the growing number of third party plugins and the principles of how to build your own.

## Adding jQuery to a web page

- **jQuery is simply a JavaScript file**
  - Add it to the HTML

```
<script type="text/javascript" src="jquery-1.x.x.min.js"></script>
```
- **There will be at least two versions of the file**
  - jquery-1.x.x.js - The development version
  - jquery-1.x.x.min.js - The release version
- **The development file is uncompressed and commented**
  - Easier for development
- **The release version is compressed**
  - All whitespace removed
  - Much smaller for delivery

Add the jQuery library as you would add any external javascript file. It is important to note that the jQuery reference must be added before you use any functionality from the library.

A standard download involves two files, an uncompressed and commented development version. This offers comments against each function and many development IDE pick up these comments and offer some sort of developer prompts.

The release version is compressed and has been minified with all whitespace and comments removed making it almost unreadable by human eyes.

## What does jQuery do?

- **jQuery is a JavaScript library**
  - Designed to make common client-side tasks trivial
  - Example of zebra-striped tables
    - Each even row being a different css class
    - Trivial with jQuery...
- **jQuery focuses on retrieving elements from our HTML pages**
  - And performing operations on them
  - Using selector-like syntax
- **High-priority given to cross-browser compatibility**

jQuery is a JavaScript library, but it is a powerful library that is specifically designed to make common client-side tasks easy if not trivial.

jQuery concentrates on retrieving sets of elements from our HTML pages and performing operations on them using CSS-selector like syntax.

The operations performed are cross-browser compliant.

## The jQuery wrapper

### jQuery uses selector notation

- Similar to CSS

- Makes it easier to pick up

```
$ (selector);  
jQuery(selector);
```

- In CSS we may define the selector

```
div p a
```

- Meaning all anchors within a paragraph within a div

- In jQuery we can select the same items using

```
$ ("div p a")
```

- Retrieves a JavaScript object containing an array of the DOM elements that match the selector
  - Lots of useful functions are attached to the object return from the call

As jQuery uses selector syntax it is relatively simple to select any DOM elements we would like. jQuery is by default aliased to \$. One of the ways we can use this method call is by supplying the selector as a string as the parameter. The example above of "div p a" simply means all anchors that are within paragraphs that are within divisions of the page. When we retrieve the set we can apply many useful functions.

## jQuery chaining

- If we want to fade out all the div elements with class of finished

```
$("div.finished").fadeOut();
```

- Special feature of jQuery sets

- When a function (like fadeOut) is called it returns the same set once it has completed allowing another operation to be executed
- So we can chain the calls together

```
$("div.finished").fadeOut().addClass("gone");
```

- If using an id selector, we can treat it as an array if we like

```
$("#someElement").html("Some text");
```

```
$("#someElement")[0].innerHTML = "Some text";
```

If we want to fade out all of the div elements with a class of finished we can use the line of code shown in the first example above.

One of the interesting and powerful features of jQuery is that when a function like fadeOut is called it will return the same set as the return type which means we can chain calls, as in the second example.

The third example shows how we can use the return as an array if we like – the two code lines are equivalent, but I'll let you decide which is more readable.

## jQuery - first look at some special selectors

As well as the usual CSS selectors that we know

- jQuery introduces some powerful custom selectors

<code>\$("p:even");</code>	Returns all the even paragraph elements
<code>\$("tr:nth-child(1)");</code>	Returns the first row of every table
<code>\$("body &gt; div");</code>	Returns divs that are direct descendants of the body
<code>\$("a[href\$=docx]");</code>	Returns all anchors where the href attribute ends in docx (i.e. anchors to Word 2007 documents)
<code>\$("body &gt; div:has(a)");</code>	Returns divs that are direct descendants of the body that contain an anchor

- For more information on selectors

- <http://docs.jquery.com>Selectors>

As well as the usual CSS selectors that we know and love, jQuery introduces many more, some of which are shown above.

A full exhaustive list of selectors can be found at the jQuery site as can all other documentation. We will spend a lot of time in the next chapter looking at them.

Many of these selector options are part of the emerging CSS3 standard. If we were to use them as pure CSS3 we would find they would not work in anything but the most recent crop of browsers,

However, jQuery has been designed to understand these selectors, allowing you the ability to use them in any browser that supports the jQuery library. Yes that is right even IE6!

This is important because jQuery's power starts with its selector arsenal. For those of you looking to engage in HTML5, now you can use jQuery as your fallback strategy.

## Events with jQuery (1)

**jQuery provides a nice event architecture**

- **on or bind method**
- **To attach an event handler**
  - To all elements returned by a selector

```
$("img").on('click', clickHandler);
```

- **To attach many event handlers (multicast)**
  - Chain the calls to bind...

```
$("img").on('click', clickHandler).bind('click', myOtherHandler);
```

- **Single call events that unregister when fired once**

```
$("img").one('click', clickHandler);
```

- **Common events are aliased as set methods**
  - E.g. click

```
$("img").click(fn); //Pass the handler function in
```

Event handling is the cross browser horror that turns many a good man from JavaScript. There are significant differences in legacy browsers and the current ECMA and W3C complaint browsers and how they respond.

Event binding/unbinding is made very easy with jQuery. We can use the bind method (equivalent of \$addHandler) as shown above, but we can bind many elements to a single event handler with one statement. We can also bind many handlers at once by chaining calls to bind.

To sum up the code above, we use selectors to locate the elements we want to work with, e.g. all images ("img"). Then jQuery executes a bind function call on every matched element. So the click event of each img is bound to a function. How the bind occurs and what method it utilises beneath the covers is abstracted from us. All we need to know is that it works efficiently and reliably across the board.

There are situations where we may want to react to an event once and then not react again. In this case we can use the one method which is a once only event binding.

To make things even easier, common events are aliased as methods; for example click, shown above, and mouseover.

## Events with jQuery (2)

- **Removing events**
  - Use `unbind` or `off` method
- **To remove all events from set**

```
$("img").unbind();
```

- **To remove all events of a particular type**

```
$("img").off("click");
```

- **To remove a handler for a particular type**

```
$("img").unbind("click", clickHandler);
```

Removing events is just as simple. We use the `unbind` method.

It is important to release event handlers from the document when a user navigates from or closes a page. On terminated JavaScript is a terrible resource sink.

In JavaScript, functions that are defined inline (such as our call-backs and event handlers) are called anonymous functions. They are referred to as “anonymous” simply because they don’t have a name! You use anonymous functions when you only require the function to be run from one particular location.

In any situation where we’re using anonymous functions, it’s also possible to pass a function name yet define the function elsewhere. This is best done when the same function needs to be called in several different places.

## Are you ready?

- jQuery provides a `ready()` function which hooks into an event
  - jQuery code will execute as soon as is possible

### The ready function()

```
$(document).ready(function() {
    alert('Are you ready?!');
});
```

- This ensures a DOM object is there before you attempt to access it
  - Only needs to be run once per page
  - Should be used on most pages

### The ready function() in short hand

```
$(function() {
    alert('Are you ready?!');
});
```

Almost everything you do in jQuery will need to be done *after* the document is ready – so we'll be using this action a lot. It will be referred to as the document-ready event from now on. Every example unless otherwise stated, needs to be run from inside the document-ready event. You should only need to declare it once per page though.

It is often a good idea to add a similar event handler when a page unloads. This means any global resources you have created can be disposed of or any updates to cookie or state can occur now.

```
$(document).unload(function () {
    //clear up your code here
});
```

## Just the beginning

- **More animations**
- **Effects**
- **Utility functions**
  - User agent
  - Box model
  - Detecting float styles
- **Manipulating JavaScript objects and collections**
- **Dynamically loading scripts**
- **Custom plug-ins...**

## Summary

- **What is jQuery?**
- **Selecting elements with jQuery**
  - jQuery chaining
- **Manipulating properties with jQuery**
- **Events with jQuery**
  - on, bind, unbind and off

## Summary

**jQuery is a freely available JavaScript library, which is cross browser compatible that help to simplify Client side coding.**

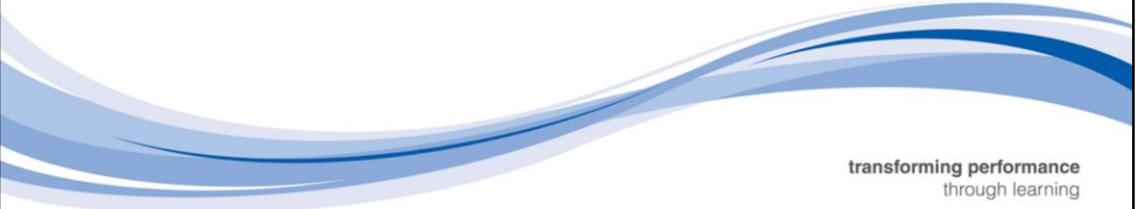
## Review Question

- **What are the versions of jQuery library?**
- **How do you add jQuery onto a web page?**
- **How do you ensure that the DOM object is there before you attempt to access it in jQuery?**



## HTML5 Forms

Developing Web Applications using HTML5



transforming performance  
through learning

## This chapter covers

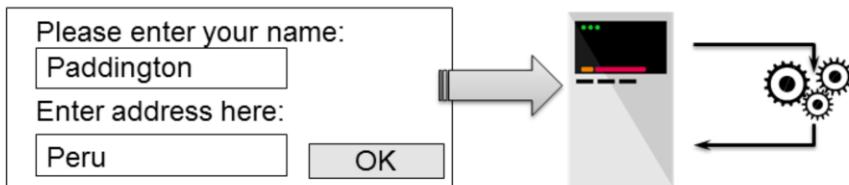
- **What are forms and its limitations**
- **Using fieldset and legend**
- **HTML5 form controls**
- **New attributes for input elements**
- **HTML5 Validation**
- **Extending forms with JavaScript**

## In this chapter you will learn

- How to create a form with new HTML5 forms controls and new attributes for input elements
- How to validate form using HTML5 new attributes, CSS3 pseudo-classes, and JavaScript

## Understanding forms – what are forms?

- Forms allow us to send data to the server for processing



```
<form action="/folder/enrol.aspx" method="POST" >
  <input type="text" name="username" />
  <input type="text" name="address" />
  <input type="submit" value="OK" />
</form>
```

- Define:
  - Form – action & method
  - Inputs – name, type & value

183

Users can interact with programs running on the server by using HTML forms; in this case the web server acts as nothing more than a gateway to forward the form's data to the back-end application and to return any generated response. Forms allow the web to be used for a variety of purposes including user inquiries, ordering and booking systems, and front ends to databases.

A form is described using the `<form>` tag; this will contain other elements and text, including the elements describing the form's input fields. The `<form>` tag takes two attributes: the `method` states how the data is to be sent to the Web server, and the `action` gives the URL of a resource used to process the form. This is normally a server-side program or script, but can be a `mailto:` URL.

The values for the `method` attribute determine how the data is sent to the server and takes the value POST or GET. Depending which you use, the server application needs to be programmed slightly differently, but the details of this are beyond the scope of this course.

## The limitation of HTML4 form inputs

### Text boxes / areas

```
<input type="text" name="username" value="" />
<input type="password" name="password" value="" />
<textarea name="comment" rows="10" cols="40"></textarea>
```

### Checkboxes and radio buttons

```
<input type="checkbox" name="milk" value="CHECKED"/>Milk?<br />
<input type="radio" name="drink" value="tea" />Tea<br />
<input type="radio" name="drink" value="coffee" />Coffee<br />
<input type="radio" name="drink" value="choc" />Chocolate<br />
```

### Selections

```
<select name="title">
  <option value="Dr">Dr</option>
  <option value="Ms">Ms</option>
  <option value="Mr">Mr</option>
</select>
```

```
<select name="prod" multiple>
  <option value="a">Apples</option>
  <option value="p">Pears</option>
  <option value="g">grapes</option>
</select>
```

184

HTML4 has a paltry selection of input types: three ways of entering text and three ways of selecting from a predefined list of options.

Most form fields are defined using an `<input>` tag; the `type` attribute specifies whether this is a checkbox, radio button, etc. A few field types have their own specific tags, for example `textarea` and `select`. These are shown above.

Each input element has a name and a value. The name is specified in the `name` attribute, the value may be specified in the `value` attribute (except for text areas when it is specified by placing text within the `<textarea>...</textarea>` tags) or left for the user to enter (e.g. with text box input). The name-value pairs are how back-end code accesses the data from the form.

## Using fieldset and legend element

### fieldset and legend element

- **Method to group and label form control**
- **Makes documents more accessible**

```
<fieldset>
  <legend>Your details</legend>
  <ol>
    <li>
      <label for="name">Name</label>
      <input id="name" name="name" type="text"/>
    </li>
    <li>
      <label for="email">Email</label>
      <input id="email" name="email" type="text"/>
    </li>
  </ol>
</fieldset>
```

185

The [HTML 4.1 specification](#) provides a method for grouping and labelling related form controls. This capability is provided by the use of the `fieldset` and `legend` elements:

The `fieldset` element allows authors to group thematically related controls and labels. Grouping controls makes it easier for users to understand their purpose while simultaneously facilitating tabbing navigation for visual user agents and speech navigation for speech-oriented user agents. The proper use of this element makes documents more accessible.

The `legend` element allows authors to assign a caption to a `fieldset`. The `legend` improves accessibility when the `fieldset` is rendered non-visually.

Whenever a label is needed to provide information about a related set of controls consider using the `legend` and `fieldset` elements. They can be used to group any thematically related controls in a form, such as address details, date of birth and sets of radio buttons or check boxes.

Note: it is required that the `fieldset` and `legend` are used in conjunction. A `fieldset` cannot be used without a `legend` and vice versa.

## HTML5 Form controls

### Number input type

- Provides a spinbox
- Attributes: min, max, value (default), step (increment by)

```
<input type="number" min="1" value="4" step="2">
```

### Range input type

- Render as a slider
- Attributes: min, max, value, step

```
<input type="range" min="1" max="10" value="2" step="2">
```



186

HTML5 introduces 13 form controls that didn't exist in HTML4; they give you more precise control over how you gather user input.

The number input types, as you might expect, is used for specifying a numerical value. IE, Safari and Chrome render the input as a spinbox control whereby you can click the arrows to move up or down. With the additional attributes min, max and step we can change the default step value of this spinbox control as well as set minimum, maximum and starting values (using the standard HTML value attribute). Each of the attributes are optional, with defaults being set if they aren't used.

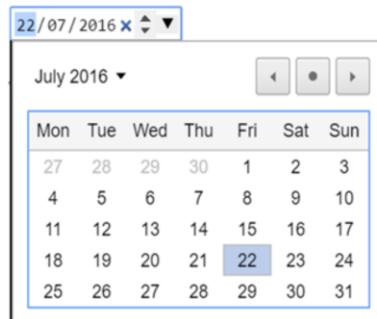
The range input type is similar to number, but more specific. In Opera, Safari, Internet Explorer 10 and Chrome, type="range" renders as a slider. When moving the slider in IE 10 a tooltip appears showing the current value. You can set the min and max attributes, and also set the starting point for range using the value attribute.

## HTML5 Form controls

### Date input type

- Render as datepicker
- Six date time types: date, month, week, time, datetime, datetime-local
- Limited support, Opera lead the way, other browsers varied

```
<input id="startdate" name="startdate" min="2016-01-01"  
max="2020-01-01" type="date">
```



187

If you've ever booked tickets online, you will have come across date pickers to help you quickly and easily choose the date you require. In HTML5 we get that functionality baked into the browser. Not only that, but we don't have to stop at just selecting a single date; we can select a week, month, time, date and time, and even date and time with a time zone using the different input types.

You can use the min and max attributes to ensure the user can choose from a specified date range. As with many of the other form implementations, Opera leads the way (support in other browsers is varied).

## HTML Form controls

### Search Input Type

- Like search in e-commerce site
- Provide a semantic definition

```
<input type="search" name="search">
```

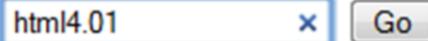


Figure 1. type="search" as displayed in Safari for Windows



Figure 2. type="search" on the iPhone

188

Search input type is like the search you find on that e-commerce site you just made a purchase from. Desktop browsers will render this in a similar way to a standard text field – until you start typing, that is. At this point, a small cross appears on the right side of the field. This lets you quickly clear the field, just like Safari's built-in search field.

On a mobile device, however, things start to get interesting. Take the example iPhone shown in Figure 2; when you focus on an input using type="search", notice the keyboard, specifically the action button on the keyboard (bottom right). Did you spot that it says "Search" rather than the regular "Go"? It's a subtle difference that most users won't even notice, but those who do will afford themselves a wry smile.

As you've seen with the new attributes, browsers that don't understand them will simply degrade gracefully. The same applies to all of the new input types discussed here. If a browser doesn't understand type="search", it will default to type="text". This means you're not losing anything. In fact, you're using progressive enhancement and helping users to have a better experience. Let's face it, filling out web forms isn't very fun, so anything you can add to ensure a smoother experience, the better.

## HTML5 Form controls

### Email input type

- Just a standard text input type
- Combined with “required” attribute, implement basic validation
- Mobile device load custom keyboard

```
<input type="email" name="email" required>
```

The screenshot shows an Opera browser window. In the top left, there's a status bar with signal strength, 02-UK, 08:59, and 91%. Below it is a form with several fields:

- An "Email" field containing "gordo".
- A "Date of Birth" field.
- A "Name" field containing "nail".
- A "URL" field containing "http://mysite.com".

A red callout box points to the "Name" field, which has a red border. Inside the box is the text "Please enter a valid email address".

Figure 1. Opera e-mail address error messaging



Figure 2. The iPhone displays a custom keyboard when using type="email".

189

In rendering terms, the email input type is no different than a standard text input type and allows for one or more e-mail addresses to be entered.

Combined with the required attribute, the browser is then able to look for patterns to ensure a valid e-mail address has been entered. Naturally, this checking is rudimentary, perhaps looking for an @ character or a period (.) and not allowing spaces. Opera 9.5+, Firefox 4+, Internet Explorer 10 and Chrome 5+ have already implemented this basic validation.

The iPhone displays a custom keyboard when using type="email". There are dedicated keys for the @ and . characters to help you complete the field more efficiently. As we discussed with type="search", there is no downside to using type="email" right now. If a browser doesn't support it, it will degrade to type="text". And in some browsers, users will get a helping hand.

## HTML5 Form controls

### url input type

- For web address
- Simple validation with “required” attribute
- Mobile device load custom keyboard

```
<input type="url" name="url" required>
```



Figure 1. type="url" activates a URL-specific keyboard on the iPhone.

190

The url input type, as you might expect, is for web addresses. You can use the multiple attribute to enter more than one URL. Like type="email", a browser will carry out simple validation on these fields and present an error message on form submission. This is likely to include looking for forward slashes, periods, and spaces, and possibly detecting a valid top-level domain (such as .com or .co.uk).

Again, we'll take a look at how the iPhone renders type="url". As you can see in Figure1 above, it has again updated the onscreen keyboard to ensure that completing the field is as simple as possible for the user by swapping the default space key for period, forward slash, and .com keys. (To access more endings like .org and .net, tap and hold the .com key.)

## HTML5 Form controls

### tel input type

- No particular syntax is enforced, Phone numbers differ around the world
- Display numeric keyboard

```
<input type="tel" name="tel" id="tel" required>
```

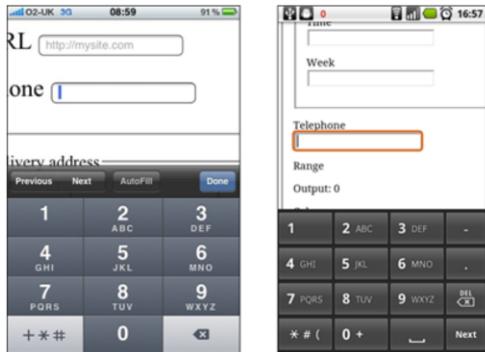


Figure 6. type="tel" on the iPhone and some Android devices dynamically changes the keyboard to a numeric keypad.

191

tel differs from email and url in that no particular syntax is enforced. Phone numbers differ around the world, making it difficult to guarantee any type of specific notation except for allowing only numbers and perhaps a + symbol to be entered. It's possible that you can validate specific phone numbers (if you can guarantee the format) using client-side validation.

Once more, the iPhone recognises type="tel", only this time it goes one step further and completely changes the keyboard to the standard phone keyboard, as shown in Figure 1. In addition to the iPhone, some Android devices (such as HTC Desire, shown on the right in Figure 2) also display a numeric keyboard for type="tel".

## HTML5 Form controls

### Colour input type

- Simple colour picker
- Allows the user to select a colour
- Returns the hex value for that colour

```
<input id="color" name="color" type="color">
```

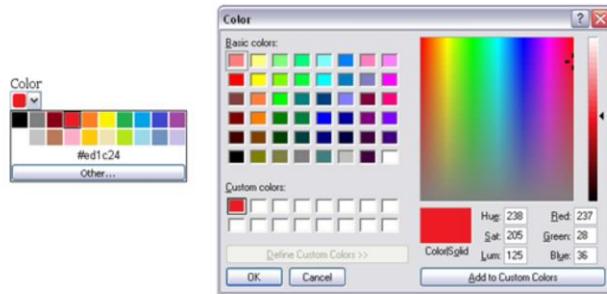


Figure 1. type="color" in Opera on the left and the result of clicking Other shown on the right

192

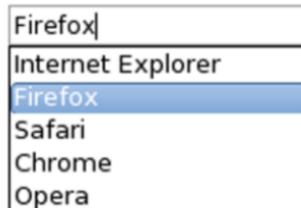
The color input type is pretty self-explanatory: it allows the user to select a colour and returns the hex value for that colour. It is anticipated that users will either be able to type the value or select from a colour picker, which will either be native to the operating system or a browser's own implementation. Opera 11 has implemented type="color" with a simple colour picker that offers a number of standard colour choices or the option to select Other, which brings up the operating system colour picker.

## HTML5 form controls

### Creating combo boxes with <datalist>

- Combination of a text box and a select list
- User can select from a list or free-type a value that isn't on the list

```
<input type="text" name="browser" list="browsers">
<datalist id="browsers">
<option value="Internet Explorer">
<option value="Firefox">
<option value="Safari">
<option value="Chrome">
<option value="Opera">
</datalist>
```



193

One type of form control that's common in desktop applications but not available in HTML4 is the *combo box*. so named because it's a combination of a text box and a select list – the user can select from a list or free-type a value that isn't on the list.

HTML5 adds support for this directly into the markup with the `<datalist>` element. A *datalist* is a named list of options, similar to the list of options in a `<select>` element, which can then be associated with one or more `<input>` elements using the `list` attribute.

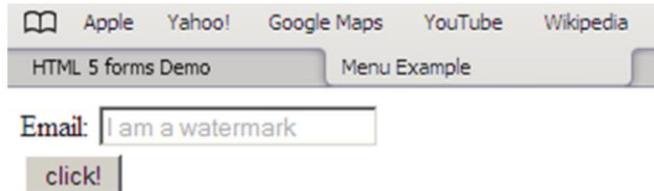
In the example, with a screenshot taken in Firefox, the input has been associated with a `<datalist>` with the id value "browsers". When a user selects the input, the list of options pops up. The user can pick one of the options using the cursor keys or type their own.

## New attributes for the input element

### placeholder attribute

- Offers default text
- Give the user examples or instructions for the field
- Sometime called a watermark
- Can only be used for text value, not default value

```
<input type="text" placeholder="I am a watermark" />
```



194

In addition to the new form controls in HTML5, existing HTML4 form controls have been extended with new attributes.

Several others can be applied to most `<input>` elements: `placeholder`, `autofocus`, and `autocomplete`.

A popular technique in recent years has been to put a suggestion for a field's required user input in the field by default. This is called *placeholder text*. This approach is popular with designers because using field names as placeholders doesn't take up any extra space, but provides the user with useful information about the expected input. It's also a compact way of indicating the desired input format.

## New attributes for the input element

### autofocus attribute

- It is common to have the first field of a form to focus
  - To have the cursor flashing ready to type
- Previously achieved with JavaScript
  - The markup representation is faster
    - Part of the page rendering rather than code execution
- Supported in all browsers other than IE9 and older:
  - Use JavaScript to support legacy browsers

```
<form>
    <input name="q" autofocus="true">
    <input type="submit" value="Search">
</form>
```

## New attributes for the input element

### Autocomplete attribute

- Provide a hint to the browser that the values entered into a field shouldn't be remembered for future use by the browser's auto-form-filling functionality
- Used for confidential information like PIN number or password

```
<label>Account: <input type="text" name="ac" autocomplete="off"></label>
<label>PIN: <input type="password" name="pin" autocomplete="off"></label>
```

196

The autocomplete attribute allows you to provide a hint to the browser that the values entered into a field shouldn't be remembered for future use by the browser's auto-form-filling functionality. This could be because the field accepts information that's supposed to be secret (for instance, a PIN number) or because the field expects a one-off value where past values entered are likely to be irrelevant (such as a password-reset code).

## HTML5 Validation

### required attribute

- Can add to any type of input
- Force a field to be mandatory
- Message appears differently in each browser

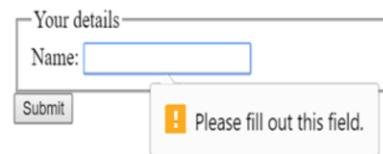
```
<input id="name" name="name"  
type="text" required/>
```

Your details

Name:

Submit

Please fill out this field.



```
<input type="email" name="email"  
required>
```

Email

Please enter a valid email address

<http://mysite.com>



197

Validation is often a crucial issue on the web, both for security and for general smooth operation of apps. HTML5 has built-in form-validation features.

The simplest form of validation is to mark a field as required. In HTML5 this is done by adding the required attribute. If an input is marked as required, the browser shouldn't allow the form to be submitted until the user has provided a value.

Figure1 shows what happens when a text input is marked as required and the user tries to submit the form without entering a value. You can add the required attribute to any type of input.

## HTML5 Validation

### ■ min and max attribute

- Number input type, trigger an error

```
<input type="number" min="1"  
max="10" name="exnumber">
```

Number: 20

! Value must be less than or equal to 10.

### ■ Pattern attribute

- Use a regular expression to validate a field
- Work with text, search, url, tel, email and password input types
- Validation fails, browser display value of title attribute

```
<input type="text" name="partno"  
pattern="[0-9] [A-Z] {3}"  
title="A part number is a digit  
followed by three uppercase  
letters.">
```

a

! Please match the requested format.  
A part number is a digit followed by three uppercase letters.

198

In HTML5, you can use the min and max attributes on the number input type. You already saw these attributes on the range control, where they specify the limits of the slider. On the number control, they trigger an error when the user tries to submit the form. The min attribute works exactly the same way.

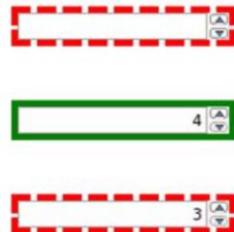
You can also use the pattern attribute to supply a regular expression that is then used to validate the field. If the validation fails, the browser displays the value of the title attribute, so you should include some information there that will help the user in filling out the form.

## HTML5 Validation with CSS3

- **CSS3 pseudo-classes**

- - :valid
  - :invalid

```
input:valid {  
    outline: 5px solid green;  
}  
input:invalid {  
    outline: 5px dashed red;  
}
```



- **The images show the result of applying this CSS to these three number controls**

```
<input type="number" required>  
<input type="number" min="4" value="4">  
<input type="number" min="4" value="3">
```

199

HTML5 provides behind-the-scenes hooks for CSS and JavaScript. These let you provide immediate visual feedback. CSS has two pseudo-classes that allow you to provide different styles based on whether they're currently valid or invalid.

The :valid CSS pseudo-class represents any <input> or <form> element whose content validates correctly according to the input's type setting and the :invalid CSS pseudo-class represents any <input> or <form> element whose content fails to validate according to the input's type setting. These allow you to easily have fields adopt an appearance that helps the user identify and correct errors.

The figure show a simple pair of CSS rules to put a green outline around valid controls and a dotted red outline around invalid controls.

## HTML5 Validation with CSS3

- **CSS3 pseudo classes**

- :required
- :optional

```
input:required {  
outline: 5px dashed blue;  
}  
input:optional {  
outline: 5px solid green;  
}
```



- **The images show the result of this CSS applied to these two inputs**

```
<input type="number" required>  
<input type="number">
```

200

There's also CSS support for styling required controls differently through pseudo-classes.

## Turning off validation

- **To submit the form without triggering validation**
  - Two new attributes: *novalidate* and *formnovalidate*
  - *novalidate* attribute can be applied to the <form> element itself

```
<form novalidate>
  <input type="email" id="addr">
  <input type="submit" value="Subscribe">
</form>
```

- **formnovalidate attribute affects the enter form but should be applied only to a Submit button:**

```
<input type="submit" value="Save for Later"
formnovalidate>
```

201

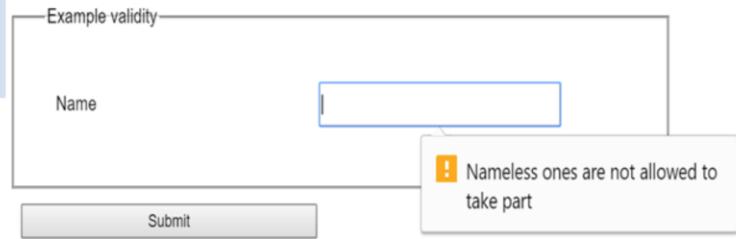
Sometimes you want the user to be able to submit the form without triggering validation. For example, if a form is long and has many sections, you might want to let the user save their progress and come back and complete the form later. To do this, HTML5 provides two new attributes: *novalidate* and *formnovalidate*.

The *novalidate* attribute can be applied to the <form> element itself, whereas the *formnovalidate* attribute affects the enter form but should be applied only to a Submit button.

## Extending forms with JavaScript

- **Customising the validation messages**
  - Use `setCustomValidity` property and `oninvalid` event

```
var fldName =  
document.getElementById('fullname');  
fldName.oninvalid = function () {  
fldName.setCustomValidity("");  
if (!fldName.validity.valid) {  
    fldName.setCustomValidity(  
        "Nameless ones are not " +  
        "allowed to take part"  
    );  
}  
};
```



202

You can supply a custom validation message using JavaScript. Use the `setCustomValidity` property of the `<input>` element in the DOM.

As you can see in the example, your message is displayed if the field fails the validity check. In this case the required attribute was set on the text field.

Also, for the `oninvalid` event to fire, the form must be submitted.

## Summary

- **HTML4 form is limited**
- **Use fieldset and legend to create accessible form**
- **There are 13 new HTML5 form controls**
- **There are also various new attributes for HTML5 input elements**
- **HTML5 validation is possible with new attributes, CSS3 pseudo-classes and JavaScript**

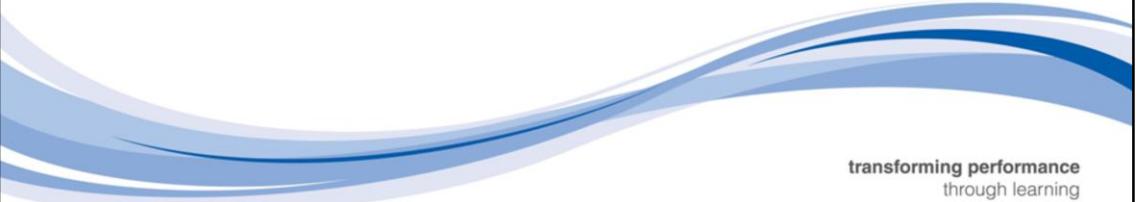
## Review Questions

- How do we create more accessible HTML5 form?
- How many new HTML5 form control are there? Name some of them
- What are some of the useful new attributes for input elements?
- How do you create validation using new HTML attribute?
- What are the properties and event we can use to customise validation messages?



# Geolocation

Developing Web Applications using HTML5



transforming performance  
through learning

## This chapter covers

- **What is Geolocation API**
- **Google map provider**
- **The Position object**
- **The Coordinate object**
- **The Position Options object**

## In this chapter you will learn

- **What are the various objects in Geolocation API**
- **How to use Google Map API**

## What is Geolocation API

- **Geolocation API**
  - Exposes Global Positioning System and other positioning service to your web page
  - Never been part of the HTML Living Standard or the HTML5 family of specification at W3C
  - Popular technologies associated with HTML5
  - Supported in most modern browsers
- **There are numerous browser specific adaptions**
  - Extends the navigator object with a new API
  - Must check for support or else get a silent fail
- **Much greater accuracy when using wireless and mobile devices**
  - Some browsers may not work at all when attached to a wired network

Browser support quick check: geolocation		Standard
	5.0	
	3.5	
	9.0	
	10.6	
	5.0	

Location-aware services and applications are a hot topic at the moment. Most of us are now familiar with navigation devices in our cars that constantly update position information using the Global Positioning System (GPS) network. These days, many mobile phones and other portable devices come with built-in GPS technology, as well as other positioning services, and the HTML5 Geolocation API exposes these to your web pages.

The Geolocation API lets you share your location with trusted web sites. The latitude and longitude are available to JavaScript on the page, which in turn can send it back to the remote web server and do location-aware things like finding local businesses or showing your location on a map.

A lot of early HTML5 demos featured the Geolocation API. But this API has never been a part of the HTML Living Standard or the HTML5 family of specifications at W3C. The Geolocation API has its own specification at the W3C.

## Where am I and do I want you to know?

- **Geolocation is always optional and a user must allow it**
  - User is sharing exact location
  - Will always be prompted
- **A mapping provider is needed (e.g. Google maps).**
  - Google map required API key

```
<script  
src="https://maps.googleapis.com/maps/api/js?key=YOUR_A  
PI_KEY_HERE"></script>
```

- **Then get hold of the `getCurrentPosition` method**
- **Mandatory success callback optional error method**

```
navigator.geolocation.getCurrentPosition(success,  
error);
```

Geolocation on its own does not do anything, but there are a number of mapping providers out there, Google maps and Bing maps for example. Both functionalities allow you to make use of their own JavaScript APIs for connecting to their service.

It is important to note that Geolocation is not supported everywhere and a user may choose not to opt into the service. Geolocation opt-in means your browser will never force you to reveal your current physical location to a remote server. The user experience differs from browser to browser.

As the end user you are told:

- A website wants to know your location
- Which website wants to know your location

The user can then choose:

- To share your location or not
- In most browsers you can tell your browser to remember your choice

The call to `getCurrentPosition()` will return immediately, but that doesn't mean that you have access to the user's location.

You can obtain google map API key from the following URL:

<https://developers.google.com/maps/documentation/geolocation/get-api-key>

## The position object

- When user agrees pass a position object to method

```
function success(position) {  
    //work with the position object here  
}
```

- Position object has two properties
  - timestamp - date and time when location was calculated
  - coords - list of properties about location (next slide)
- A failure returns a PositionError object

Property	Type	Description
Code	Short	An enumerated value
Message	String	Not intended for UI

The callback function will be called with a single parameter, an object with two properties: coords and timestamp. The timestamp is just that, the date and time when the location was calculated.

Since this is all happening asynchronously, you can't really know when that will happen in advance. It might take some time for the user to receive the data.

The coords object has properties like latitude and longitude which are exactly what they sound like: the user's physical location in the world.

If anything goes wrong, your error callback function will be called with a PositionError object. The code property will be one of:

- PERMISSION\_DENIED (1) if the user clicks that "Don't Share" button or otherwise denies you access to their location
- POSITION\_UNAVAILABLE (2) if the network is down or the positioning satellites can't be contacted
- TIMEOUT (3) if the network is up but it takes too long to calculate the user's position
- UNKNOWN\_ERROR (0) if anything else goes wrong

## The coordinates object

The `coord` object has the properties listed below

- Only latitude, longitude and accuracy are guaranteed
- Depends on device whether others are supported

Property	Type	Description
latitude	double	Decimal degrees
longitude	double	Decimal degrees
altitude	double or null	Meters above the referenced ellipsoid
accuracy	double	Meters
altitudeAccuracy	double or null	Meters
heading	double or null	Degrees clockwise from true north
speed	double or null	Meters/second

Only three of the properties are guaranteed to be there (`coords.latitude`, `coords.longitude`, and `coords.accuracy`). The rest might come back null, depending on the capabilities of your device and the backend positioning server that it talks to. The heading and speed properties are calculated based on the user's previous position, if possible.

## The position options object

- **Mobile devices often offer two methods for geolocation**
  - Cell tower triangulation - fast but inaccurate
  - GPS positioning - power hungry and slower to connect
- `getCurrentPosition` supports a third optional parameter

Property	Type	Default	Description
enableHighAccuracy	Boolean	false	True will usually be slower
timeout	long	(no default)	Measured in milliseconds
maximumAge	long	0	Measured in milliseconds

If the enableHighAccuracy property is true, and the device can support it, then the device will try to provide it. Both iPhones and Android phones have separate permissions for low- and high-accuracy positioning, so it is possible that calling `getCurrentPosition()` with `enableHighAccuracy:true` will fail, but calling with `enableHighAccuracy:false` would succeed.

The timeout property is the number of milliseconds your web application is willing to wait for a position. This timer does not start counting down until *after* the user gives permission to even try to calculate their position.

The maximumAge property allows the device to answer immediately with a cached position. For example if you call `getCurrentPosition()` for the first time, the user consents, and your success callback function is called with a position that was calculated at exactly 10:00 AM. Exactly one minute later, at 10:01 AM, you call `getCurrentPosition()` again with a maximumAge property of 75000.

## Summary

- **Geolocation API exposes Global Positioning System and other positioning service to your web page**
- **Geolocation is always optional and a user must allow it**
- **A mapping provide like Google Maps is needed**

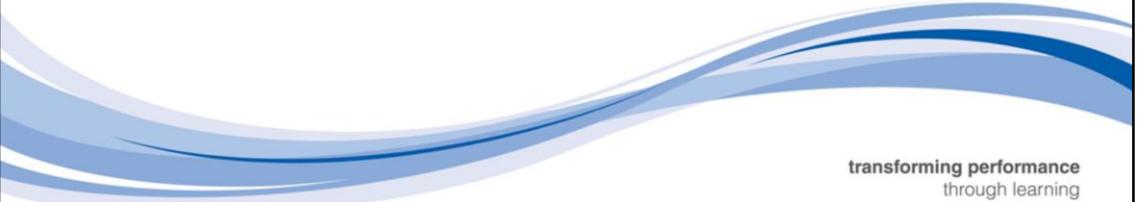
## Review Questions

- What is one of the most popular mapping provider?
- What method allow us to get the current position of the device?
- What are the main two properties of the position object?
- What properties are guaranteed in the coordinates object?
- What are the properties in the third optional parameter in getCurrentPosition object?



## Drag and Drop

Developing Web Applications using HTML5



transforming performance  
through learning

## This chapter covers

- **What is the Drag and Drop API**
- **dataTransfer object**
- **dataTransfer item**
- **Drag-and-drop events**
- **Drag and drop demo**

## In this chapter you will learn

- How to use the drag and drop API to drag an item and drop into a container
- The various attributes and methods in the API
- How to use the events from the API

## What is the drag and drop API

- **HTML5 drag and drop interfaces**
  - Enable application to use drag and drop feature in Firefox and other browsers
  - User can select draggable element with a mouse
  - Drag the element to a droppable element
  - Drop those elements by releasing the mouse button
- **Drag and Drop API**
  - Reverse engineering from IE implementation
  - Three main parts:
    - dataTransfer object
    - dataTransfer item
    - A collection of events
- **Drag operation create dataTransfer object**
  - This contains one or more dataTransfer items
  - Access both by listening to the events

HTML5 drag and drop interfaces enable applications to use drag and drop features in Firefox and other browsers. For example, with these features, the user can select *draggable* elements with a mouse, drag the elements to a *droppable* element, and drop those elements by releasing the mouse button. A translucent representation of the *draggable* elements follows the mouse pointer during the drag operation.

The drag and drop API is another API that's reverse engineered from the IE implementation. The API has three main parts: the dataTransfer object, the dataTransfer item, and a collection of events. Some of those are covered in tables respectively. A drag operation will create a dataTransfer object; this will contain one or more dataTransfer items in the items attribute, and you can gain access to both by listening to the events.

## dataTransfer object

Attribute/Method	Description
dropEffect	This is the type of operation taking place (copy, link, move, none)
effectAllowed	Contains the type of operations allowed (copy, copyLink, copyMove, link, linkMove, move, all, uninitialised, none).
Items	Returns a list of dataTransfer items with the drag data
setDragImage(element, x, y)	Updates the drag feedback image with the given element and coordinates
setData(format, data)	Sets the data being dragged
getData(format)	Returns the data being dragged

## dataTransfer item

Attribute/Method	Description
Kind	This is the kind of item being dragged (string or file).
Type	This is the data item type string.

data-Transfer item defines an object being dragged to the drop zone.

## Drag and drop events

Event Name	Description
dragstart	Fires on the source element when the user starts to drag the source element
drag	Fires on the source element as the user is dragging the source element
dragenter	Fires on the target element when the user drags the source element into it
dragleave	Fires on the target element when the user drags the source element out of it
dragover	Fires on the target element as the user is dragging the source element over it
drop	Fires on the target element when the user drops the source element on it
dragend	Fires on the source element when the user stops dragging the source element

Table lists the drag-and-drop events. When the application listens for these events, it can use the event object to gain access to the dataTransfer object or dataTransfer items. To access the dataTransfer object, use e.dataTransfer, where e is an event object. To access dataTransfer items, use e.dataTransfer.items, where items is a list of dataTransfer items.

## Drag and drop demo

- **Step1: to make an element draggable, set draggable attribute to be true**
- **Step2: what to drag**
  - ondragstart attribute calls a function, drag(event), that specifies what data to be dragged

```

```

## Drag and drop demo

### Step 3: create drag(event) function

- **dataTransfer.setData() method sets the data type and the value of the dragged data**
- **The target event property returns the element that triggered the event**
- **Images are usually dragged only by their URLs**
  - Use the text type as with other URL links

```
function drag(event) {  
    event.dataTransfer.setData("text", event.target.id)  
}
```

## Drag and drop demo

### Step 4: ondragover event

- Specifies where the dragged data can be dropped

```
<div id="box1" ondragover="allowDrop(event)"></div>
```

- By default, data/elements cannot be dropped in other elements
- To allow a drop, we must prevent the default handling of the element

```
function allowDrop(event) {  
    event.preventDefault();  
}
```

## Drag and drop demo

### Step 5: do the drop

When the dragged data is dropped, a drop event occurs.

```
function drop(event) {  
  
    //prevent the browser default handling of the data  
    //(default is open as link on drop)  
    event.preventDefault();  
  
    //Get the dragged data  
    var data = event.dataTransfer.getData("text");  
  
    //Append the dragged element into the drop element  
    event.target.appendChild(document.getElementById(data));  
}
```

## Summary

- **Drag and drop API is reverse engineered from the IE implementation**
- **The API has three main parts: the dataTransfer object, the dataTransfer item, and a collection of events**

## Review questions

- **What are the three main parts in the drag and drop API?**
- **How do you make an element draggable?**
- **What method sets the data type and the value of the dragged data?**
- **Which event specifies where the dragged data can be dropped?**
- **What method is used to prevent the browser default handling of the data?**



# AJAX

Developing Web Applications using HTML5



transforming performance  
through learning

A decorative graphic consisting of several blue and white curved lines that flow from left to right across the page.

## This chapter covers

- **What is AJAX?**
- **Four principles of Ajax**
- **XMLHttpRequest object**
- **JSON**
- **Fetch API**

## In this chapter you will learn

- How to use the enable AJAX using XMLHttpRequest object and Fetch API

## What is Ajax?

- **A technology for creating rich internet applications**
  - Used to create highly-responsive applications
  - Rich content and interactions
- **A client-focused model**
  - Uses client-side technologies - JavaScript, CSS, DHTML
- **A user-focused model**
  - Ajax behaviour based on user interactions
  - “User-first” development model
- **An asynchronous model**
  - Communications with the server are made asynchronously
  - User activity is not interrupted

Users of web applications increasingly expect a user experience that provides a high level of interactivity on as close a level to a desktop application as possible.

The typical server-bound web application cannot easily provide this due to having to defer to the server for any significant updates to the user interface.

Ajax enables us to create a highly-responsive, rich user interface. It does this by enabling several design options through standards-based technologies.

Key to this are the client-side technologies: DHTML with JavaScript and CSS for the styling of our pages. Due to developments and standardisation of these technologies, we can change the focus of our application from server-centric to client-centric.

We focus our development efforts on user interactions with our application and have a user-driven, event-based model so we can concentrate as much as possible on the user experience.

Through asynchronous communication with the server we are able to retrieve data or HTML fragments that we can then insert into the DOM programmatically. The main benefits of asynchronous communication in this scenario are that we are only transmitting a small amount of information and are semi-coupled with the server, and more importantly, user activity and interactions with our application are not interrupted during this request process.

## Four principles of Ajax

### 1. The browser hosts an application, not content

- A richer document is sent to the browser
- JavaScript manages the client-side interaction with the user

### 2. The server delivers data, not content

- Requests for data, not content, are sent to the server
- Less network traffic and greater responsiveness

### 3. User interaction can be continuous and fluid

- The client is able to process simple user requests
- Near instantaneous response to the user

### 4. Real programming

- This is real coding and requires discipline!

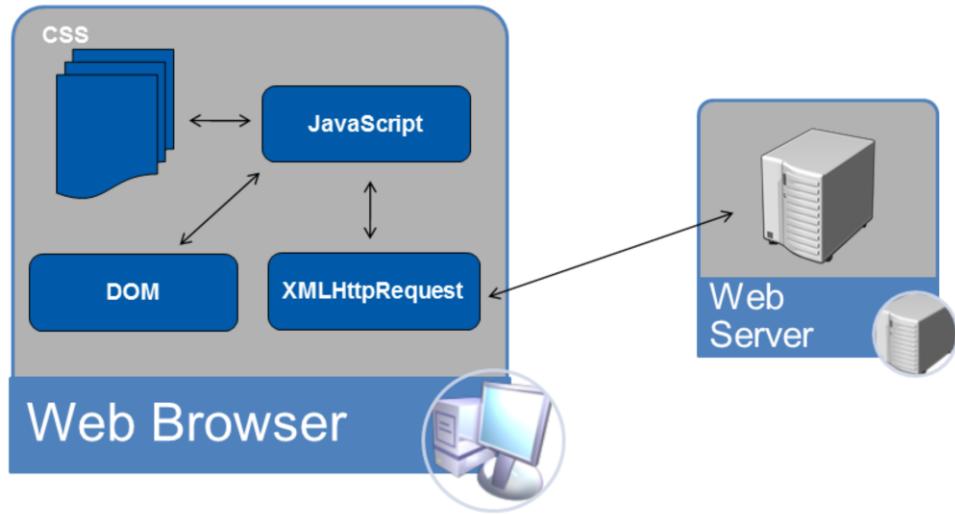
When we work in an Ajax environment, we have to think differently to how we may usually think with a server-bound application. We need to change our idea of the application running from the server to the client. When we add Ajax functionality to a web application, we will be adding a certain amount of code that we would not add normally to a typical server-bound web application. This means that the browser will essentially be hosting an application and not simply content. We will be sending a richer document to the browser including JavaScript files and then we will manage interaction from the client by making requests for data (not content) and using this to update the DOM within the browser. This mechanism creates less network traffic and therefore allows us to show greater responsiveness within our application.

User interaction is simplified in a similar way to event-driven desktop applications in that the client browser using our JavaScript code is able to process the simple (or even relatively complex) requests and make an asynchronous request for data (if necessary) and provide a near instantaneous response to the user due to the decreased traffic and potentially lower processing overhead on the server.

However, we will be writing a lot more code and constructing an application rather than just a series of effects on the client and so we are in the realm of real coding and we need to take a disciplined approach to this.

## Ajax enabling technologies

**CSS, DOM, JavaScript and XMLHttpRequest.**



The diagram above shows the main enabling technologies of Ajax.

At the client end we have the web browser which will host the document that is requested. This document is made available via a Document Object Model (DOM) so that we can programmatically manipulate the document using JavaScript and also hook into events of both the browser, document and constituent elements.

In addition, we can use the XMLHttpRequest object to initiate requests to a server in order to retrieve data, then use that data through JavaScript to update the document through the DOM.

## XMLHttpRequest – overview

- **Handles the request process**
  - W3C specification
  - See <http://www.w3.org/TR/XMLHttpRequest/>
  - Defines an API that provides scripted client functionality for transferring data between a client and a server
- **Benefits**
  - Simple to use
  - Can be used for any request type e.g. GET, POST
  - Can be used synchronously or asynchronously
  - Request headers can be added
  - Response headers can be read
  - Support in all modern browsers

The W3C Working Draft document referenced above defines an interface that is implemented by the XMLHttpRequest object within conformant browsers. This includes all modern browsers.

The object contains a simple API for creating most types of request (GET, POST, HEAD, PUT, DELETE, OPTIONS) and can be used over HTTP or HTTPS. It can be used for making synchronous or asynchronous requests.

Like any typical HTTP request, we can manipulate the headers by adding extra entries to the request and we can read the response headers.

## XMLHttpRequest – requests

- **open method**
  - Sets up the XMLHttpRequest object for communications

```
request.open(sendMethod, sendUrl[, booleanAsync, stringUser, stringPwd]);
```

- **send method**
  - Initiates the request

```
request.send([varData]);
```

- **abort method**
  - Cancels a request currently in process

- **setRequestHeader method**
  - Adds custom HTTP headers to the request
    - Used mainly to set *Content-Type*

```
request.setRequestHeader(sName, sValue);
```

There are four named request methods; open, send, abort and setRequestHeader.

The open method takes a string indicating the method (GET, POST etc...). It also requires the url as a string. The other parameters are optional, however to make an asynchronous call you will need to pass true as the bAsync parameter. You can also, optionally, provide a username and password to use for authentication.

The send method initiates the request and has three ways of invocation. You can send the request without any data (e.g. when invoking for a GET request). In addition, the varData parameter can contain either a string containing name/value pairs as would be the body of the request, or it can contain an XML Document.

The abort method allows us to cancel a request that is currently processing.

The setRequestHeader method is used to add headers to the request and is used mainly to set the content-type when we want to POST data. Both parameters are strings.

## XMLHttpRequest – responses

- **readystatechange event**
  - Fires for each stage in the request cycle
  - Handle to be able to retrieve the content of the response
- **readyState property – progress indicator (0 to 4)**
  - Most important is 4 (Loaded); you can access the data
- **responseXXX property – retrieves the response**
  - responseText – as a string
  - responseBody – as an array of unsigned bytes
- **status property, statusText property**
  - Return the HTTP response code or friendly text respectively

The key to retrieving or handling the response is to hook into the readyystatechange event of the XMLHttpRequest object.

This event fires for each stage in the request lifecycle and can be used to retrieve the content of the response. The readyState property indicates what stage the request/response is at:

readyState value	Description
0	Unsent
1	Opened
2	Headers Received
3	Loading
4	Loaded (Done) – data is fully loaded

You can check the HTTP status code using the status property and obtain a friendly text description from the statusText property. To retrieve the data you use the responseText, responseXML methods. You can interrogate the response headers by using getResponseHeader or getAllResponseHeaders.

## XMLHttpRequest – example

### Using XMLHttpRequest

- **Create a new XMLHttpRequest object**
- **Set the request details using the open method**
- **Hook-up the readystatechange event to a callback function**
  - Easiest way is to use an anonymous function
- **send the request**

```
function loadAjaxTxt() {  
    var xmlhttp = new XMLHttpRequest();  
    xmlhttp.onreadystatechange = function () {  
        if (xmlhttp.readyState == 4 && xmlhttp.status == 200)  
        {  
            document.getElementById('content').innerHTML =  
  
            xmlhttp.responseText;  
        }  
        //set up communication  
        xmlhttp.open("GET", "ajax_info.txt", true);  
        //initiates the request  
        xmlhttp.send();  
    }  
}
```

In the example above, we are making a request to a text file.

First we instantiate an XMLHttpRequest object by calling its constructor.

We then invoke open with our GET method and the url and state that the request will be asynchronous.

Next, we attach a handler to the readystatechange event (available via the onreadystatechange event listener). You can provide a function name here, but in the example above we have used an anonymous function.

Within the function we first check the readyState (we only want to respond when the response has been fully loaded.) We then check to see that the request completed successfully (HTTP response code 200 – OK).

We can then do something with the responseText/responseXML.

Finally we initiate the request by invoking the send method.

## XMLHttpRequest – important information

### XMLHttpRequest support

- **Supported in all modern browsers**
  - As a native JavaScript object XMLHttpRequest
- **IE 5.x/6 use through an ActiveXObject**
- **Use *object detection* to get the right object**

#### A cross browser compliant request

```
var xmlhttp = null;
function ensureXMLHttpRequest() {
    if (window.XMLHttpRequest) {
        xmlhttp = new XMLHttpRequest();
    } else if (window.ActiveXObject) {
        try {
            xmlhttp = new
ActiveXObject("Microsoft.XMLHTTP");
        }
        catch(e) {}
    }
}
```

The XMLHttpRequest object is supported in all modern browsers, however this has not always been the case. In IE 5.x and 6, the request object provided was a custom ActiveX object.

The code above shows some script that enables us to return the correct object using object detection to see whether the XMLHttpRequest object is available. If not then we check if we can create an ActiveXObject. If we can then we return an instance of Microsoft.XMLHTTP. This is what is used in IE 5.x and 6.

## JavaScript Object Notation (JSON)

- **Lightweight data-interchange format**
  - Compared to XML
- **Simple format**
  - Easy for humans to read and write
  - Easy for machines to parse and generate
- **JSON is a text format**
  - Programming language independent
  - Conventions familiar to programmers of the C-family of languages, including C# and JavaScript

Transferring data can be a cumbersome task. XML is all well and good, however it requires a DOM parser in order to read/write and is not easily realised into object format.

JavaScript Object Notation (or JSON) is a lightweight data interchange format which is easy to read and write and, more importantly, easy for machines to parse and to generate.

JSON is a text format that is programming language independent and uses conventions familiar to C-family programmers. To JavaScript, JSON looks and behaves as an associative array and so can be parsed (using eval) and turned into a fully functioning object which is very easily consumed.

## JSON Structures

- Universal data structures supported by most modern programming languages
- A collection of name/value pairs
  - Realised as an object (associative array)
- An ordered list of values
  - Realised as an array
- JSON object
  - Unordered set of name/value pairs
  - Begins with { (left brace) and ends with } (right brace)
  - Each name followed by a : (colon)
  - Name/value pairs separated by a , (comma)

JSON consists of structures that are supported by most modern programming languages and so is immediately accessible to most.

It is an associative array (name/value pairs). It can contain an ordered list of values as an array.

The overall JSON object consists of an unordered list of name/value pairs contained within curly braces with each name and value pair separated by a colon and the name/value pairs separated by a comma.

## JSON and JavaScript

**JSON is a subset of the object literal notation of JavaScript**

- Can be used in the JavaScript language with no problems

```
var myJSONObject = { "searchResults": [
    { "productName" : "Aniseed Syrup", "unitPrice" : 10 },
    { "productName" : "Alice Mutton", "unitPrice" : 39 }
];
}
```

- Object created by this example

- Single member `searchResults`
- Contains two objects each containing `productName` and `unitPrice` members
- Can use dot or subscript operators

```
alert(myJSONObject.searchResults[0].productName); // alerts
"Aniseed Syrup"
```

JSON is a subset of the object literal notation of JavaScript and so can be used (as shown above) in JavaScript with no problems.

The object realised in the above example can be accessed using either dot or subscript operators as shown in the second example.

## Fetch API

**fetch()** allows you to make network requests similar to XMLHttpRequest (XHR)

- **Uses promises**
- **Simpler and cleaner**

```
// url (required), options (optional)
fetch('https://qa.com/some/url', {
    method: 'get'
}).then(function(response) {

}).catch(function(err) {
    // Error :(
});
```

**fetch()** allows you to make network requests similar to XMLHttpRequest (XHR). The main difference is that the Fetch API uses promises, which enables a simpler and cleaner API, avoiding callback hell and having to remember the complex API of XMLHttpRequest.

### Basic fetch usage

A fetch function is now provided in the global window scope, with the first argument being the URL.

## A fetch request example

### Parse response as JSON

```
fetch('./api/some.json')
  .then(
    function(response) {
      if (response.status !== 200) {
        console.log('Looks like there was a problem. Status Code: ' +
          response.status);
        return;
      }

      // Examine the text in the response
      response.json().then(function(data) {
        console.log(data);
      });
    }
  )
  .catch(function(err) {
    console.log('Fetch Error :-S', err);
  });
}
```

We start by checking that the response status is 200 before parsing the response as JSON.

The response of a `fetch()` request is a [Stream](#) object, which means that when we call the `json()` method, a promise is returned since the reading of the stream will happen asynchronously.

## Response metadata

Other metadata we may want to access, like headers, are illustrated below.

```
fetch('users.json').then(function(response) {  
    console.log(response.headers.get('Content-Type'));  
    console.log(response.headers.get('Date'));  
  
    console.log(response.status);  
    console.log(response.statusText);  
    console.log(response.type);  
    console.log(response.url);  
});
```

## Chaining promises

- **Great features of promises**
  - Chain them together
- **For fetch**
  - This allows you to share logic across fetch requests

```
fetch('users.json')
  .then(status)
  .then(json)
  .then(function(data) {
    console.log('Request succeeded with JSON response',
data);
  }).catch(function(error) {
    console.log('Request failed', error);
 });
```

One of the great features of promises is the ability to chain them together. For `fetch`, this allows you to share logic across `fetch` requests.

If you are working with a JSON API, you'll need to check the status and parse the JSON for each response. You can simplify your code by defining the status and JSON parsing in separate functions which return promises, freeing you to only worry about handling the final data and the error case.

```
function status(response) {
  if (response.status >= 200 && response.status < 300) {
    return Promise.resolve(response)
  } else {
    return Promise.reject(new Error(response.statusText))
  }
}

function json(response) {
  return response.json()
}
```

## POST Request

To call an API with a POST Method. Set method and body parameters in the `fetch()` option.

```
fetch(url, {
  method: 'post',
  headers: {
    "Content-type": "application/x-www-form-urlencoded; charset=UTF-8"
  },
  body: 'foo=bar&lorem=ipsum'
})
.then(json)
.then(function (data) {
  console.log('Request succeeded with JSON response', data);
})
.catch(function (error) {
  console.log('Request failed', error);
});
```

It's not uncommon for web apps to want to call an API with a POST method and supply some parameters in the body of the request.

To do this we can set the **method** and **body** parameters in the **fetch()** options.

## Summary

- **AJAX is...**
  - A technology for creating rich internet applications
  - A client and user-focused model
  - A technology that enables asynchronous requests
- **XMLHttpRequest is used for making requests from the client to the server**
- **fetch() allows you to make network requests similar to XMLHttpRequest (XHR), but is simpler and cleaner**

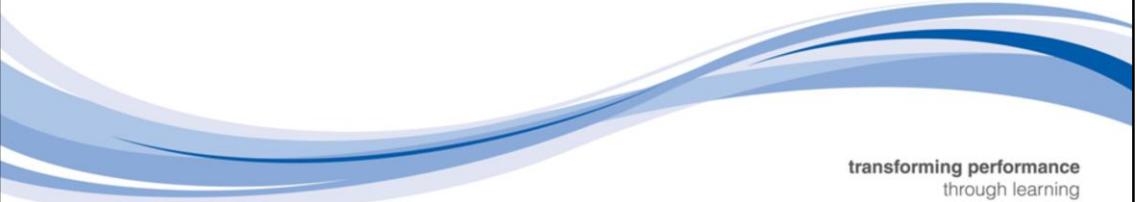
## Review Questions

- What method sets up the XMLHttpRequest object for communications?
- What event handle to be able to retrieve the content of the response?
- Which two property check whether the data is ready to be accessed?
- What does fetch() API use to make network request?



## Web Worker

Developing Web Applications using HTML5



transforming performance  
through learning

## This chapter covers

- **Key Concepts**
- **How Web Workers Work**
- **Creating Web Workers**
- **Setting up event handlers**
- **Communicating with Web Workers**
- **Implementing a Web Worker**

## In this chapter you will learn

- **How to use the Web Worker to bring background processing capabilities to web applications**

## Key concepts

**HTML5 Web Workers bring background processing capabilities to web applications**

- Run on separate threads, so JavaScript apps using HTML5 Web Workers can take advantage of multi-core CPUs
- Separating long-running tasks into HTML5 Web Workers also avoids the "slowscript" warnings

252

Web Workers provide a simple means for web content to run scripts in background threads. The worker thread can perform tasks without interfering with the user interface. In addition, they can perform I/O using [XMLHttpRequest](#) (although the responseXML and channel attributes are always null). Once created, a worker can send messages to the JavaScript code that created it by posting messages to an event handler specified by that code (and vice versa.)

## How Web Workers work

- **To use Web Workers in a web page:**
  - Create a Web Worker object, and pass in a JavaScript file to be executed
  - In the page, set up an event listener to process incoming messages and errors posted by the Web Worker
  - To communicate from a Web page to a Web Worker, call **postMessage()** to pass in the required data
- **To implement a Web Worker:**
  - Create a JavaScript file
  - In the JavaScript file, set up an event listener to process incoming messages and errors posted by the web page
  - To communicate from a Web Worker to a web page, call **postMessage()** to pass in the required data

253

This slide describes the overall mechanisms that underpin the web worker model in HTML5.

## Creating Web Workers

- **Web Workers are Worker JavaScript objects**
  - When you create a **Worker** object, you must specify the url of the JavaScript file
  - The code in the JavaScript file will run in a separate thread
- **The url can be relative or absolute**
  - If relative, it will assume the same scheme, host, and port as the main page

```
var aWorker = new Worker("MyWorker.js");
```

254

You can check whether your browser supports Web Workers as follows:

```
function testForWebWorkers() {  
    if (typeof(Worker) !== "undefined") {  
        alert("Your browser supports Web Workers.");  
    }  
    else {  
        alert("Your browser doesn't support Web Workers.");  
    }  
}
```

## Setting up Event Handlers

Before you tell the Web worker to do anything, you should set up two event handlers on the Web worker...

- **message event**

```
aWorker.addEventListener("message",
                        myMessageHandler, true);
...
function myMessageHandler(e) {
    alert("Worker response data: " + e.data);
}
```

- **error event**

```
aWorker.addEventListener("error",
                        myErrorHandler, true);
...
function myErrorHandler(e) {
    alert("Error [" + e.filename +
          ", line " + e.lineno + "] " + e.message);
}
```

255

There are two events to handle:

- The **message** event handles messages posted back from the worker (e.g. results)
- The **error** event handles errors that occur (e.g. script-loading errors)

## Communicating with a Web Worker

- **To communicate from a Web page to a Web Worker:**
  - Call `postMessage()` to pass in the required data
- **Example 1**
  - Posting a simple string message:

```
aWorker.postMessage("Some data for the Worker to use");
```

- **Example 2**
  - Posting a JSON object (e.g. a person) as a message:

```
aWorker.postMessage({ 'name': 'Andy',
                      'age': 21,
                      'welsh': true,
                      'canSing': false});
```

256

The host web page can call the `postMessage()` method on the `Worker` object, to post any data to the background thread.

## Implementing a Web Worker

- A Web Worker can listen for incoming messages sent from the Web page:

- Define an event listener to listen for **message** events

```
addEventListener("message",
    myWorkerMessageHandler, true);
```

- A Web Worker can communicate back to a web page:

- Call **postMessage()** to pass in the required data
  - You can pass anything (e.g. a string or a JSON object)

```
postMessage("Some data for the page to display");
```

257

The Web Worker can implement a method to handle "message" events from the web page, as follows:

```
function myWorkerMessageHandler(e) {
    // Access the message from main page via e.data.
    // E.g. e.data accesses the whole message.
    // E.g. e.data.name accesses part of the message.
}
```

The Web Worker can send data to the web page by calling the **postMessage()** function, as shown in the lower code fragment on the slide.

## Summary

**Web Workers are a simple API that enables a web page to do background work, whilst the browser continues to be updated on the main user interface thread.**

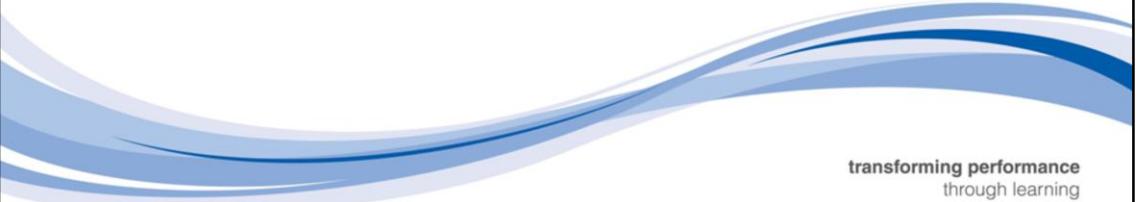
## Review Questions

- **Why do we use Web Worker?**
- **What else do you need to specify after we created a Worker object?**
- **What method is use to communicate a web page to a web worker?**
- **Where do you normally implement a Web Worker?**



## Web Socket

Developing Web Applications using HTML5



transforming performance  
through learning

## This chapter covers

### What are web sockets?

- The need for web sockets
- Understanding web sockets
- Implementing a web sockets client

## In this chapter you will learn

**How to use the Web Worker to bring background processing capabilities to web applications.**

## The need for web sockets

- Traditional HTTP Communication
- Polling
- Long polling
- Limitations of HTTP

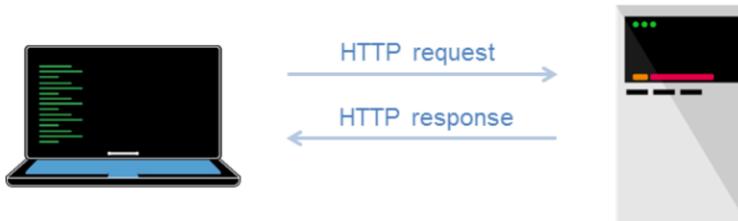
263

This section describes the need for Web Sockets. Or to put it another way, what's the problem that Web Sockets addresses?

## Traditional HTTP communication

- **In traditional HTTP:**

- A browser sends an HTTP request to a web server
  - The web server sends an HTTP response, e.g. an HTML page



- **The Web page is an expression of data at one moment in time**
    - E.g. stock prices, news reports, ticket sales, etc.
  - **There are several techniques for keeping the page up to date:**
    - Polling
    - Long polling

264

In traditional HTTP, a web browser sends an HTTP request to a web server and receives an HTTP response in return. The communication is instigated by the browser and is synchronous.

HTTP has served us well for many years, but it has some important limitations:

- It's a verbose protocol. Each HTTP request and response has a series of headers that have to be set up at one end and processed at the other end
- It's a client-driven protocol and requires the client to request content when it needs it. This means the content displayed by the client (i.e. the contents of the Web page) is a representation of data at a moment in time

## Polling

- **This is how polling works in traditional HTTP:**
  - The browser sends AJAX requests to the server at regular intervals
  - The server responds immediately, possibly with updated data
  - The process repeats every few seconds
- **Remarks:**
  - Real-time data is often not that predictable, so the browser will probably make unnecessary requests
  - HTTP connections will probably be opened and closed needlessly in low-message-rate situations
  - Each AJAX request is a separate HTTP request, and HTTP is quite an inefficient protocol

265

One way for a browser to ensure its data is (reasonably) up-to-date is to make regular AJAX calls to the server. Each AJAX call is an asynchronous call to get the latest data from the server. You write a JavaScript callback function in the web page to receive the data and update the user interface accordingly.

This is a popular technique in contemporary web applications, but it's not a perfect solution. Consider these observations:

- How often should the client request data from the server? Every 5 seconds? Every 30 seconds? Every 10 minutes?
- What if the client requests data now, and then 5 milliseconds later the data is updated at the server. The client missed this update, and will have to wait until the next scheduled request to see the new data. Is this important...?
- As mentioned on the previous slide, HTTP is verbose. If you're making lots of requests to the server, each one incurs the overhead imposed by the HTTP protocol

## Long polling

- **This is how long polling works in traditional HTTP:**
  - The browser connects to the server, setting a connection timeout to a very big value (e.g. hours)
  - The browser then sends a request to the server, to get new data
  - The server returns new data when available, or connection times out
  - The process then repeats
- **Remarks:**
  - Better than polling, because fewer connections opened/closed
  - But there is an overhead in maintaining an open connection
  - Also the server might only support a limited number of connections

266

Another way for a browser to ensure its data is up-to-date is by using a technique known as "long polling".

With long polling, the client connects to the server and specifies a long timeout (possibly several hours). The client then requests data from the server, and waits until data is returned. The server doesn't have to reply immediately; it can hold off until new data is available, and then return that data. The connection will be closed when the server finally returns data, or when the connection times out. Either way, the client can then create a new connection, and the process repeats.

This technique allows the server to send bang-up-to-date data to the client, but it means the server might potentially have to maintain lots of open connections in order to service all the clients. In practice, there will be a limit on the number of concurrent open connections supported by the server.

## Limitations of HTTP

- **HTTP was not designed for real-time full-duplex communication**
  - The methods described earlier involve lots of HTTP request/response headers, which causes high latency
- **Furthermore, in an attempt to simulate full-duplex communication over half-duplex HTTP...**
  - The traditional approaches have typically used two separate connections (one for upstream, one for downstream)
  - Maintaining and coordinating these two connections consumes extra resources and adds complexity

267

As described on the previous slides, polling and long polling gives some support for overcoming the inherent one-way, synchronous limitations of traditional HTTP. However, neither of these techniques allows us to create true two-way asynchronous communications between a client and a server.

This is exactly the problem that Web Sockets is designed to address.

## Overview of web sockets

**Web Sockets defines a full-duplex, socket-based communication channel between browser and server**

- **Simultaneous two-way data exchange between client and server**
- **A large advance in HTTP capabilities**
- **Extremely useful for real-time, event-driven web applications**



268

The Web Sockets specification is a parallel initiative to HTML5, and defines a lightweight, two-way, real-time communications protocol between browsers and servers.

Web Sockets provide a dramatic reduction of unnecessary network traffic compared to polling:

- Estimated 500:1 reduction in unnecessary HTTP header traffic
- Estimated 3:1 reduction in latency

These are huge improvements, and offer a major step forward in the scalability of real-time web applications.

## How web sockets work

- **To establish a Web Socket connection between a client and a server:**
  - The client sends an HTTP or HTTPS request, containing an "upgrade to WebSocket" HTTP header
  - If the server supports Web Sockets, it returns an HTTP response containing an "OK to upgrade to WebSocket" HTTP header
  - A persistent two-way socket connection is established
- **Once a Web Socket connection has been established:**
  - The client and server can send and receive messages over the open connection simultaneously
  - The format of the data can be anything that both parties are happy with

269

The Web Sockets protocol is defined in the RFC6455 specification from the Internet Engineering Task Force (IETF). See <http://tools.ietf.org/html/rfc6455> for full details.

This is how Web Sockets work:

1. Initially, the client sends a normal HTTP or HTTPS request to the server. The request includes an "upgrade to WebSocket" header, which informs the server that the client would like to upgrade to the Web Sockets protocol rather than keep talking HTTP.
2. If the server is Web Sockets-enabled, it will understand the request from the client and it will return an HTTP response that contains "OK to upgrade to WebSocket" header.
3. A Web Socket connection has now been established between the client and the server, so they can now communicate bidirectionally, asynchronously, in full-duplex mode. The client and the server can exchange data in any format they like – there is no need for any HTTP headers any more.

## Web socket enabled servers

- **Microsoft**
  - Internet Information Services 8 and later
- **Node.js**
  - <http://nodejs.org>
- **PHP**
  - <http://code.google.com/p/phpwebsocket/>
- **Apache HTTP Server extension**
  - <http://code.google.com/p/pywebsocket/>
- **Java**
  - <http://jwebsocket.org/>
- **Ruby**
  - <https://github.com/gimite/web-socket-ruby>

270

To use Web Sockets, you need a server that understands the Web Sockets protocol. The slide above lists some possibilities.

Note that you also need a browser that supports the client-side Web Sockets API. For example, Microsoft Internet Explorer 10 supports Web Sockets.

## The web socket interface

**The W3C defines a client-side WebSocket interface.**

```
interface WebSocket {  
  
    readonly attribute DOMString URL;  
  
    const unsigned short CONNECTING = 0;  
    const unsigned short OPEN = 1;  
    const unsigned short CLOSED = 2;  
    readonly attribute unsigned short readyState;  
    readonly attribute unsigned long bufferedAmount;  
  
    attribute Function onopen;  
    attribute Function onmessage;  
    attribute Function onclose;  
    attribute Function onerror;  
  
    void close();  
};
```

To use Web Sockets in a web page in the browser, you make use of the `WebSocket` object which is presented in the slide above. We'll take a detailed look at its various properties, methods and events in the following slides.

## Checking for web sockets support

- The first step in a client application is to check whether the browser supports web sockets
  - Test for the existence of the `WebSocket` property on the `window` object
- Example:

```
if (window.WebSocket) {  
    // Web Sockets are supported ...  
}
```

272

Web Sockets are a new feature, and the majority of browsers in current usage do not support them. Therefore, your first step should always be to test whether the browser supports Web Sockets. To do this, test the `window` object to see if it has a `WebSocket` property as shown in the slide above.

## Opening a connection

- **To open Web Sockets connection a server:**
  - Create a WebSocket object, specifying the URL to connect to
- **For an unsecure Web Socket connection:**
  - Use the ws:// protocol
  - Default port is 80
- **For a secure Web Socket connection:**
  - Use the wss:// protocol
  - Default port is 443
- **Example:**

```
var aSocket = new WebSocket ("ws://myserver.com/myapp");
```

273

The client opens a Web Socket connection by creating a **WebSocket** object and specifying a "**ws://**" or "**wss://**" url.

The task of opening a Web Socket connection is asynchronous (as in fact are all Web Sockets operations), because the handshake for establishing a Web Sockets connection is relatively time-consuming.

You can't start sending or receiving data until you know the connection is open. To deal with this situation, you must handle the **open** event on the **WebSocket** object. The following slide describes how to handle all the events on a **WebSocket** object.

## Handling web sockets events

- **The Web Sockets API is asynchronous, because:**
  - Establishing a connection is relatively time-consuming
  - Once a connection has been established, data can arrive at any time over the connection
- **The client should handle events as follows:**

```
aSocket.onopen      = function(e) { ... };  
aSocket.onmessage  = function(e) { ... };  
aSocket.onclose    = function(e) { ... };  
aSocket.onerror   = function(e) { ... };
```

274

The **WebSocket** object has the following properties for handling Web Sockets events:

- **onopen** – indicates the connection has been opened, so you can start sending data to the server
- **onmessage** – indicates a message has arrived from the server
- **onclose** – indicates the connection has been closed
- **onerror** – indicates an error has occurred

## Sending data to the server

- **The client can send data to the server at any time**
  - Invoke send() on the WebSocket object
  - The data is buffered and sent asynchronously

```
aSocket.send(someData);
```

- **Types of data supported:**
  - Text
  - Binary data
  - An array

275

Here's an example of sending text data (e.g. JSON) to the server:

```
var obj = ... ;  
aSocket.send(JSON.stringify(obj));
```

Here's an example of sending binary data (e.g. a file) to the server:

```
var file = document.getElementById("myFile").files[0];  
aSocket.send(file);
```

Here's an example of sending an array to the server:

```
var lotteryNumbers = new Uint16Array([3,12,19,1,2,7]);  
socket.send(lotteryNumbers.buffer);
```

## Receiving data from the server

- **The client can receive data from the server at any time**
  - Handle the message event
- **The event argument has two properties: type and data**
  - If the type is "binary", the web socket object has a binaryType property that indicates whether the data is a "blob" or "arrayBuffer"

```
aSocket.onmessage = function(e) {  
    if (e.type == "text") {  
        ... handle text data ...  
    }  
    else {  
        if (aSocket.binaryType == "blob") ...  
        else if (aSocket.binaryType == "arrayBuffer") ...  
    }  
};
```

276

When the server sends data to the client, the client receives a **message** event. To handle this event, set the **onmessage** property on your **WebSocket** object.

In the event handler function for the **message** event, the event argument has two properties:

- **type** – indicates the type of the data, i.e. either "**text**" or "**binary**". For binary data, the **WebSocket** object has a **binaryType** property that is either "**blob**" or "**arrayBuffer**", so you know how to process the data at the client
- **data** – this is the data itself

## Closing the connection

- The client should close the Web Sockets connection when it wants to terminate its conversation with the server

- Invoke close() on the WebSocket object
- You can specify optional parameters: code and reason

```
aSocket.close(42, "Client closed connection normally");
```

- When the connection has been closed, the close event occurs

- The event argument has three properties: wasClean, code, reason

```
aSocket.onclose = function(e) {  
    if (!e.wasClean) {  
        alert("Connection closed with code " + e.code +  
              " [reason: " + e.reason + "]");  
    }  
};
```

277

The client should close the Web Sockets connection when it wants to terminate its conversation with the server. Note that the **close()** method is asynchronous, so you must handle the **close** event if you want to do any additional work after the connection has been closed.

## Summary

**The Web Sockets specification defines a lightweight, two-way, real-time communications protocol between browsers and servers.**

## Review Questions

- **What is the purpose of Web Socket?**
- **How does the client open a Web Socket connection?**
- **What are the events that handle Web Socket?**
- **What types of data are supported in Web Socket?**



## Client Side Storage

Developing Web Applications using HTML5



transforming performance  
through learning

## This chapter covers

- **Why local storage?**
- **The competing visions**
  - Web Storage
  - IndexedDB

## In this chapter you will learn

- How to use Web Storage API to store data on the client side
- How to use IndexDB to create web applications with rich query abilities

## Goodbye cookies

- **Local storage means the end to cookie abuse!**
  - Often used to maintain persistent client state
- **This will ultimately be a good thing for web applications**
  - Cookies are limited to 4KB of data and 20 per domain
  - Cookies are included in every HTTP request
    - Wasteful plus potential bandwidth issues
    - Potential security issues
- **What modern web apps need is:**
  - More storage space
  - Client based
  - That persists



283

Cookies have been with us for a long time, they are venerable soldiers in the campaign to make a stateless protocol stateful. With HTML5 we have the chance to see farewell to these brave soldiers and move on to a new and improved local storage facility.

## Web Storage

- **HTML5 is a volatile part of the spec**
- **Web storage is the most mature**
  - Has very good browser support
  - Allows you to create a key value pair on the client
  - Create up to a 5MB sql based client datastore

IE	FF	Safari	Chrome	Opera	iOS	Android
8+	3.5+	4.0+	4.0	10.5+	2.0+	2.0+

- **indexedDB is less supported currently and is still experimental**
  - indexedDB moving towards a consensus
  - Browser support an issue

284

The Web Storage API provides mechanisms by which browsers can store key/value pairs, in a much more intuitive fashion than using cookies.

The following page will test whether your browser supports localStorage, sessionStorage or globalStorage.

The tests will also determine if there is a storage limit for any storage system that is supported.

<http://dev-test.nemikor.com/web-storage/support-test/>

## What is web storage?

- **Web storage is based on named key/value pairs**
  - Retrieve data based upon the key
  - Key is a string value
- **Value can be any JavaScript type, but is held as string**
  - Must be parsed into correct type
  - Using parseInt() or similar
- **Local storage can be treated like an associative array**
  - Method based or array based approach to manipulating data
  - Can be used for more complex JSON structures
- **Can be held as sessionStorage or localStorage**

285

Web storage is based on named key/value pairs. Data is stored based on a named key. The key name itself is held as a string but the data can be any type supported by JavaScript, including strings, Booleans, integers, or floats.

The data is actually stored as a string. If you are storing and retrieving anything other than strings, you will need to use functions like parseInt() or parseFloat() to coerce your retrieved data into the expected JavaScript datatype.

The storage API offers two modes sessionStorage and localStorage. Session Storage is available only to the window until the window is closed. Use this for shopping basket type activities where you would not want the date to leak from one window to another.

localStorage is based around a domain and spans all windows that are open on that domain. The data is cross-session durable and will last until you want to be rid of it.

## Using web storage

- Use `setItem` method to save the key value pair

```
localStorage.setItem("thing", 5);
//or
sessionStorage.setItem("stuff", "value");
```

- Read with `getItem` method parsing where necessary

```
var x = parseInt(localStorage.getItem("thing"));
```

- Or use expando properties or array syntax

```
alert(localStorage.thing); //use to add or read
alert(localStorage.key(0)); //access properties by indexer
```

286

This is probably the most beautiful and simple mechanism you have seen on the course so far and should change the way you create applications today. Web storage is simple to use and here today.

Both storage methods use `setItem` to add to the local storage state as a OO type setter. There is then a `getItem` as a suitable retrieval method.

However, you may have come across the concept of expando properties in your JavaScript programming lifetime. You could set and retrieve by simply adding a property to the storage object without invoking the methods. It still holds the values as strings in the underlying type, but may save you a lot of method calling.

Finally you can use the `key` method if you wish to iterate over a storage object using a `for` loop for instance. What I have just said here may set an alarm bell off in your hind brain, what if I have a property called `key`? Depends on the browser, but ultimately not good things. Webkit root browsers have a known bug with the issue so avoid the key of '`key`'.

## Deleting state

- You can delete by item

```
localStorage.removeItem("stuff");
```

- Or in entirety

```
localStorage.clear();
```

- You can also retrieve the length of a storage object

```
localStorage.length;
```

- **sessionStorage** is automatically wiped when a window closes
  - **localStorage** persists until it is overwritten or cleared

287

Remember local storage persists indefinitely so you will want to be able to clear it up by key or give a user the choice to wipe out all data.

If the key does not exist the delete will do nothing.

Be careful in the development lifecycle to ensure you do not use the same key names or objects elsewhere in your application to avoid overwriting values accidentally.

## Dealing with complex objects

- AJAX applications often hold JSON data structures

```
var courseDetails = {  
    title : "Introduction to HTML5 and CSS3",  
    code: "QAHHTML5",  
    duration: 3  
}
```

- Using a JSON serialisation we can add it to storage

```
setItem("courseDetails", JSON.stringify(courseDetails));  
  
var o = JSON.parse(sessionStorage.getItem("courseDetails"));
```

288

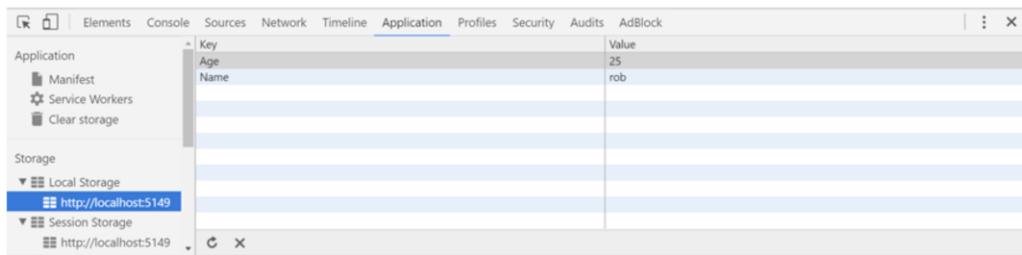
In a modern AJAX application JSON objects are likely being thrown back and forth between client and server on a regular basis. One of the key problems in AJAX application development has been holding state on the client. Web storage makes this a much more simple.

All a developer needs is a way to serialise the JSON object into a string. There are ways to do this in all the major JavaScript libraries (jQuery, Dojo, AJAX.NET to name a few). To hold away from leveraging into a framework the example we are using Douglas Crockford's JSON library (<http://www.json.org/json2.js>).

The benefit of using JSON is when the string is returned to an object state the data inside the object properties will be of the correct type, saving you a whole slew of parse calls.

## Managing state

You will want to debug and explore web storage state. Many of the browsers have a way of doing this.



The screenshot shows the Chrome DevTools interface with the Application tab selected. On the left, under the Storage section, the Local Storage tab is selected, showing data for the URL `http://localhost:5149`. Two items are listed: 'Age' with value '25' and 'Name' with value 'rob'. The Session Storage tab is also visible below it.

Key	Value
Age	25
Name	rob

## Introducing indexedDB

- **indexedDB has been in W3C recommendation since Jan 2015**
  - Mozilla led, Chrome and Microsoft backed

IE	FF	Safari	Chrome	Opera	iOS	Android
10+	4.0+	6.0+	11.0+	12.0+	8	4.4

- **indexedDB had a synchronous and asynchronous API**
  - Most browsers will only offer asynchronous option
- **It has a benefit over the much more simple to use local storage API**
  - Capable of holding a significant amount of data as objects
  - Able to search and index more effectively

290

On the 18<sup>th</sup> of November, 2010, the W3C announced that Web SQL database is a deprecated specification. This is a recommendation for web developers to no longer use the technology as effectively the spec will receive no new updates and browser vendors aren't encouraged to support this technology.

The replacement, **indexed** has been in W3C recommendation since Jan 2015.

Many major browsers including Chrome, Safari, Opera and nearly all Webkit based mobile devices support WebSQL and will likely maintain support for the foreseeable future.

In the majority of cases where you are using Indexed Database you will be using the asynchronous API. The asynchronous API is a non-blocking system and as such will not get data through return values, but rather will get data delivered to a defined callback function.

The IndexedDB support through HTML is transactional. It is not possible to execute commands or open cursors outside of a transaction. There are several types of transactions: read/write transactions, read only and snapshot. In this tutorial, we will be using read/write transactions.

## IndexedDB

- **Data is stored in object stores, which are:**
  - Collections of JavaScript objects
  - Whose attributes contain individual values
- **Each object, in an object store has a common attribute**
  - Key values uniquely identify records within an object store
  - An index organises objects based upon the attribute value
- **Indexes return sets of key values**
  - Used to obtain the individual records from the object store
- **A cursor represents a set of values**
  - When an index defines a cursor
  - The cursor represents the set of key values returned by the index

291

IndexedDB is new in HTML5. Web databases are hosted and persisted inside a user's browser. By allowing developers to create applications with rich query abilities it is envisioned that a new breed of web applications will emerge that have the ability to work online and off-line.

IndexedDB is an Object Store. It is not the same as a Relational Database, which has tables, with collections rows and columns. It is an important and fundamental difference and affects the way that you design and build your applications.

In a traditional relational data store we would have a table of "to do" items that store a collection of the users to do data in rows. With columns of named types of data. To insert data, the semantics normally follow: `INSERT INTO Todo(id, data, update_time) VALUES (1, "Test", "01/01/2010");`

IndexedDB differs in that you create an Object Store for a type of data and simply persist Javascript Objects to that store. Each Object Store can have a collection of Indexes that make it efficient to query and iterate across.

IndexedDB also does away with the notion of a Standard Query Language (SQL), instead it is replaced with a query against an index which produces a cursor that you use to iterate across the result set.

## Creating an indexedDB

### First you must create the datastore

- Different browsers implement in different ways
- So you need to use an alias

```
var indexExample= {};
var indexedDB = window.indexedDB || window.webkitIndexedDB ||
    window.mozIndexedDB;
```

- Currently we also need to alias the key and transaction

- Chrome supports the API but in an experimental capacity

```
if ('webkitIndexedDB' in window) {
    window.IDBTransaction = window.webkitIDBTransaction;
    window.IDBKeyRange = window.webkitIDBKeyRange;
}
```

292

Data is stored in object stores, which are collections of JavaScript objects whose attributes contain individual values.

Each JavaScript object, sometimes called a record, in an object store has a common attribute called a key path; the value of this attribute is called a key value (or key). Key values uniquely identify individual records within an object store.

An index organises objects according to the value of a common attribute. Indexes return sets of key values that can be used to obtain the individual records from the original object store.

A cursor represents a set of values. When an index defines a cursor, the cursor represents the set of key values returned by the index. When an object store defines a cursor, the cursor represents a set of records stored in the cursor.

A keyRange defines a range of values for an index or a set of records in an object store; key ranges allow you to filter cursor results.

A database contains the object stores and indexes; databases also manage transactions.

A request represents individual actions taken against objects in a database. For example, opening a database leads to a request object and you define event handlers on the request object to react to the results of the request.

A transaction manages the context of operations and is used to maintain the integrity of database activities. For example, you can create object stores only in the context of a version change transaction. If a transaction is aborted, all operations within the transaction are cancelled.

## Opening the indexedDB datastore

We then need to open the database, setting up a function for success and failure.

```
indexExample.indexedDB.db = null;

indexExample.indexedDB.open = function () {
    var request = indexedDB.open("dbName");

    request.onsuccess = function (e) {
        indexExample.indexedDB.db = e.target.result;
        // what to do with the dataStore
    };

    request.onerror = indexExample.indexedDB.onerror;
};
```

293

In the above example the asynchronous pattern for indexedDB is being used. First you assign a function to the open event of the indexedDB. If the request to open the database is successful the onsuccess event is fired where the success passes a reference to the database we wish to work with.

If any error is received we trap it with the onfailure event assigning it to the onerror event of the indexedDB object.

## Creating the DataStore

To create the database we need to set the version, then set the index key.

```
/* We can only create Object stores in a setVersion transaction; */
var v = "1.99";
if (v!= db.version) {
    var setVrequest = db.setVersion(v);

    /* onsuccess is the only place we
       can create Object Stores */
    setVrequest.onerror = indexExample.indexedDB.onerror;
    setVrequest.onsuccess = function(e) {
        if(db.objectStoreNames.contains("dbName")) {
            db.deleteObjectStore("dbName");
        }

        var store = db.createObjectStore("dbName",
            {keyPath: "timeStamp"});
    }
}
```

294

The above code actually does quite a lot. We define an "open" method in our API, which when called will open the database "**dbName**". The open request isn't executed straight away. Instead an IDBRequest is returned.

The indexedDB.open method will be called when the current function exits. This is a little different to how we normally assign asynchronous callbacks, but we get the chance to attach our own listeners to the IDBRequest object before the callbacks are executed.

If the open request is successful, our onsuccess callback is executed. In this callback we check the database version and if it is not the same as the number we expect, we call "setVersion".

SetVersion is the only place in our code that we can alter the structure of the database. In it we can create and delete Object Stores and build and remove indexes. A call to setVersion returns an IDBRequest object where we can attach our callbacks. When successful, we start to create our Object Stores.

Object Stores are created with a single call to createObjectStore. The method takes a name of the store, and an parameter object. The parameter object is very important, it lets you define important optional properties. In our case, we define a keyPath that is the property that makes an individual object in the store unique. That property in this example is "timeStamp". "timeStamp" must be present on every object that is stored in the objectStore.

## Adding data to the DataStore

To add to the DataStore we need to do the following:

- Get hold of the indexedDB object
- Create a transaction object
- Create an object literal data object
  - Note the timeStamp key in the below example
- Call the put method of the transaction object
- Handle the response (see notes)

```
var db = indexExample.indexedDB.db;
var trans = db.transaction(["todo"],
IDBTransaction.READ_WRITE);
var store = trans.objectStore("todo");
var data = {
    "text": todoText,
    "timeStamp": new Date().getTime()
};
var request = store.put(data);
```

295

Now that the application has access to the Object Store, we can issue a simple put command with a basic JSON object. Notice that there is a timeStamp property, that is our unique key for the object and is used as the "keyPath". When the call to put is successful, our onsuccess event is triggered and we are able to render the contents to the screen. Note the event subscription below that handle the success or failure of the asynchronous put.

```
request.onsuccess = function(e) {
    //code to update ui
};

request.onerror = function(e) {
    //error handeling code
};
```

## Reading from the DataStore

### To access the DataStore

- Access the db, create a transaction and begin a transaction
  - As shown on the previous slide
- Instantiate a key range
- Open a cursor passing it the required range

```
// Get everything in the store;
var keyRange = IDBKeyRange.lowerBound(0);
var cursorRequest = store.openCursor(keyRange);

cursorRequest.onsuccess = function(e) {
  var result = e.target.result;
  if (!!result == false)
    return;
  // Do something with the data
  result.continue();
```

296

The code makes a transaction and instantiates a keyRange search over the data. The keyRange defines a simple subset of the data we want to query from the store. Given that the keyPath for the store is the numeric timestamp, we set the lowest value of the search to be 0 (anything since the epoch) which just so happens to return all of our data.

There is now a transaction, a reference to the store we want to query and a range that we wish to iterate over. All that remains is to open the cursor and attach an "onsuccess" event.

The results are passed through to the success callback on the cursor, where we render the result. The callback is only fired once per result, so to ensure you keep iterating across the data you need to ensure you call "continue" on your result object.

### IDBTransaction options:

Constant	Value	Description
READ_ONLY	0	Allows data to be read but not changed.
READ_WRITE	1	Allows reading and writing of data in existing data stores to be changed.
VERSION_CHANGE	2	Allows any operation to be performed, including ones that delete and create object stores and indexes.

## Summary

- **Web Storage is the most mature of the technologies**
  - May not be suitable for large sets of data
  - Overhead of serialisation for larger objects
- **indexedDB is likely the future**
  - Still a working draft, so a few years to go
  - Then the legacy browser issues to contend with

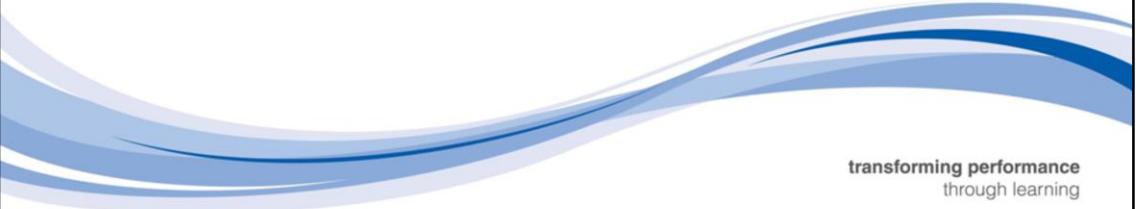
## Review question

- **What is Web Storage based on?**
- **What is the difference between local storage and session storage?**
- **What methods allow you to save the key/value pair in web storage?**



# SVG

Developing Web Applications using HTML5



transforming performance  
through learning

## This chapter covers

- **What is SVG?**
- **SVG shape elements**
- **Applying style to SVG**
- **Drawing common shapes**
- **Drawing with path**
- **Images and text**
- **Textpath**
- **Grouping and transform**

## In this chapter you will learn

- How to use **SVG** element to draw various shapes
- How to apply styles to **SVG**
- How to insert images and create text element
- How to group element and transformation
- What the difference is between **Canvas** and **SVG**

## What is <svg>

- **SVG is a vector based API**
  - Solid browser support, plugin for earlier IE browser
  - Some IE issues
    - Advanced effects do not render in IE

IE	FF	Safari	Chrome	Opera	iOS	Android
9+	4.0+	5.1+	7.0+	11.6+	5.0+	3.0+

- **Mature W3C spec first ratified in 2003**
  - XML API for describing 2D-graphics and graphical applications
  - XML is then rendered by an SVG viewer
- **HTML5 supports inline SVG using the <svg> tag**

302

SVG stands for Scalable Vector Graphics and it is a language for describing 2D-graphics and graphical applications in XML. The XML is then rendered by an SVG viewer. SVG is mostly useful for vector type diagrams like pie charts, two-dimensional graphs in an X,Y coordinate system etc. SVG became a W3C recommendation the 14<sup>th</sup> January 2003 and you can check latest version of SVG specification at [www.w3.org/TR/SVG](http://www.w3.org/TR/SVG).

## Viewing SVG Files

Most of the web browsers can display SVG just like they can display PNG, GIF, and JPG. Internet Explorer users may have to install the Adobe SVG Viewer to be able to view SVG in the browser. IE9-11 desktop & mobile don't properly scale SVG files. Adding height, width, viewBox, and CSS rules seem to be the best workaround.

## Embedding SVG in HTML5

HTML5 allows embedding SVG directly using <svg>...</svg> tag which has following simple syntax:

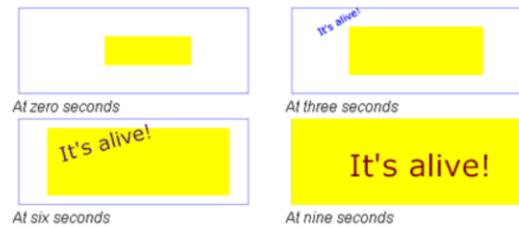
```
<svg>
```

```
...
```

```
</svg>
```

## Scalable Vector Graphics (SVG)

- **SVG is a language for describing two-dimensional graphics in XML**
- **Vector graphics contain geometric objects such as lines and curves**
  - Gives greater flexibility compared to raster-only formats (such as PNG and JPEG)
- **SVG graphics are scalable to different display resolutions**
  - The same SVG graphic can be placed at different sizes on the same Web page, and re-used at different sizes on different pages. SVG graphics can be magnified to see fine detail or to aid those with low vision
- **SVG drawings can be interactive and dynamic**
  - SVG supports the ability to change vector graphics over time



303

To be scalable means to increase or decrease uniformly. In terms of graphics, scalable means not being limited to a single, fixed, pixel size. On the web, scalable means that a particular technology can grow to a large number of files, a large number of users, a wide variety of applications. SVG, being a graphics technology for the web, is scalable in both senses of the word.

SVG graphics are scalable to different display resolutions, so that for example printed output uses the full resolution of the printer and can be displayed at the same size on screens of different resolutions. The same SVG graphic can be placed at different sizes on the same web page, and re-used at different sizes on different pages. SVG graphics can be magnified to see fine detail, or to aid those with low vision.

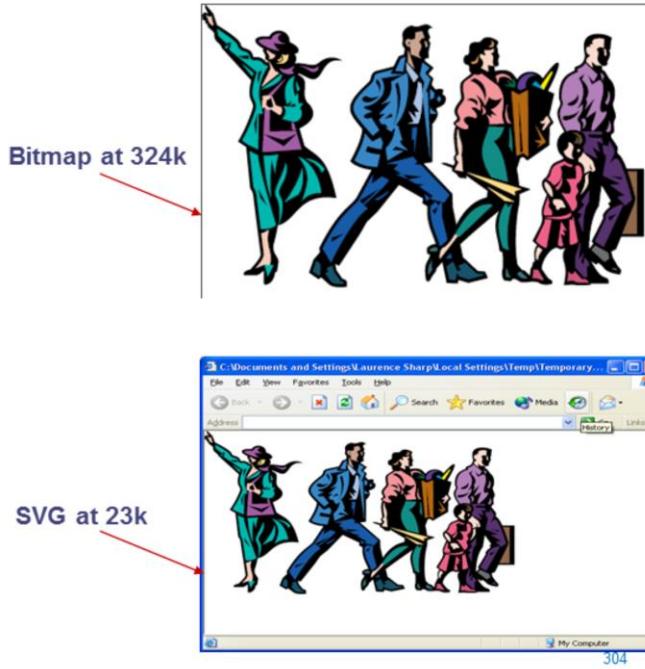
Vector graphics contain geometric objects such as lines and curves. This gives greater flexibility compared to raster-only formats (such as PNG and JPEG) which have to store information for every pixel of the graphic. Typically, vector formats can also integrate raster images and can combine them with vector information such as clipping paths to produce a complete illustration; SVG is no exception.

Since all modern displays are raster-oriented, the difference between raster-only and vector graphics comes down to where they are rasterized; client side in the case of vector graphics, as opposed to already rasterized on the server. SVG gives control over the rasterization process, for example to allow anti-aliased artwork without the ugly aliasing typical of low quality vector implementations.

Specification available from <http://www.w3.org/TR/SVG/>

## SVG Shape Elements

- **SVG contains the following set of basic shape elements:**
  - Rectangle (rectangle, including optional rounded corners)
  - Circles
  - Ellipses
  - Lines
  - Polylines
  - Polygons
- **Complex images can be created using these basic shapes**



With any XML grammar, consideration has to be given to what exactly is being modelled. For textual formats, modelling is typically at the level of paragraphs and phrases, rather than individual nouns, adverbs, or phonemes. Similarly, SVG models graphics at the level of graphical objects rather than individual points.

SVG provides a general path element, which can be used to create a huge variety of graphical objects, and also provides common basic shapes such as rectangles and ellipses. These are convenient for hand coding and may be used in the same ways as the more general path element. SVG provides fine control over the coordinate system in which graphical objects are defined and the transformations that will be applied during rendering.

## Getting started with SVG

- <svg> element

```
<svg id="mysvg" viewBox="0 0 320 240"
style="outline: 1px solid #999; width: 320px; height:
240px;">
    <rect x="50" y="50" width="100" height="100"
    style="fill: rgb(255,0,0)">
    </rect>
    <line x1="50" y1="50" x2="150" y2="150"
    style="stroke: rgb(0,127,127); stroke-width: 5;">
    </line>
</svg>
```



- viewBox

- Mapping between physical element
- And logical coordinates of everything within

**viewBox="0 0 640 480"**

- Graphic renders into larger viewport



305

First, note that the size of the element on the example is determined by CSS in the style attribute, but you also define a viewBox with the same values. Because SVG is a vector format, pixels aren't as significant; you can use viewBox to define a mapping between the physical dimensions of the element, defined in CSS, and the logical coordinates of everything displayed within.

Look what happens if you use these values: viewBox="0 0 640 480". It's the same SVG graphic as before, but rendered into a larger viewport.

## Adding styles to SVG

- **Inline style**

```
<rect x="50" y="50" width="100" height="100"  
style="fill: rgb(255,0,0)">  
</rect>
```

- **Apply directly to element**

```
<rect x="50" y="50" width="100" height="100"  
fill="rgb(255,0,0)"></rect>
```

306

The previous examples used an inline style to apply colours and stroke thicknesses. Those properties can also be applied directly to the elements in question. But you can alternatively leave off the style and inline attributes and use in your CSS file, and achieve the same results.

## Drawing common shapes

### Rectangle

```
rect x="50" y="50" width="100" height="100"  
fill="rgb(255,0,0)"></rect>
```



### Line

```
<line x1="50" y1="50" x2="150" y2="150"  
stroke="rgb(0,127,127)" stroke-width="5"/>
```



### Circle

```
<circle cx="250" cy="100" r="50" fill="rgb(255,0,0)">
```



## Drawing common shape

### ■ **Polygon**

- Closed shape

```
<polygon points="265,50 315,150 215,150"
stroke="rgb(0,127,127)" fill="rgb(255,0,0)" stroke-
width="5"/>
```



### ■ **Polyline**

- Open shape

```
<polyline points="265,50 315,150 215,150"
stroke="rgb(0,127,127)" fill="rgb(255,0,0)" stroke-width="5"
/>
```



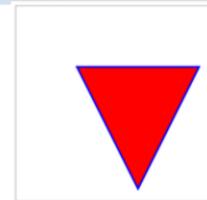
308

## Drawing with path

### Path

```
<path d="M 100 100 L 300 100 L 200 300 Z" fill="red"  
stroke="blue" stroke-width="3"/>
```

- **M : move to**
- **L : line to**
- **Z: close path**



309

```
<path d="(path data)">/>
```

The M indicates a move to, the Ls indicate line to, and the Z indicates a close path.

Uppercase is for absolute coordinate and lowercase is for relative coordinate

## Images and text

### Images

```
<image x="10" y="10" width="236" height="260"  
xlink:href="macaque.jpg"/>
```



### Text

```
<text x="10" y="10" font-family="arial" font-size="15" font-  
weight="bold">  
    <tspan x="10">Happiness is</tspan>  
    <tspan x="10" dy="20">when what you think,</tspan>  
    <tspan x="10" dy="20"> what you say,</tspan>  
    <tspan x="10" dy="20">and what you do are in  
harmony.</tspan>  
</text>
```

Happiness is  
when what you think,  
  
what you say,  
  
and what you do are in harmony.

310

Images are easy to embed within your SVG drawing. The syntax is similar to that of HTML, and the only additional information you need to provide over and above the `<image>` element are the coordinates of the upper-left corner. You use an `xlink:href` to link to the image. The `xlink` is a namespace, a legacy of SVG's XML heritage that leaks through to HTML5.

Text is handled a little differently in SVG compared to HTML. In SVG, text has to be explicitly wrapped within a containing element. Line breaks also have to be explicitly coded using the `<tspan>` element.

The `dx` attribute indicates a shift along the x-axis on the position of an element or its content and the `dy` attribute indicates a shift along the y-axis on the position of an element or its content.

## Textpath

### Textpath

```
<defs>
  <path id="myTextPath"
        d="M215,120
            a55,55 0 1 1 -110,0 55,55 0 1 1 110,0">
  </path>
</defs>
```

```
<text>
  <textPath xlink:href="#myTextPath">
    Be happy for this moment. This moment is your life.
  </textPath>
</text>
```



311

A nice effect you can achieve on short runs of text is to make the text follow a path. If you extract the circle part of the path from the earlier example, you can spread the text along it with the `<textpath>` element. The path is created in the `<defs>` element, and then you link to it using an `xlink:href` like you used for the image earlier. The link works like other web content, so you could refer to the path in a separate file if you wanted to.

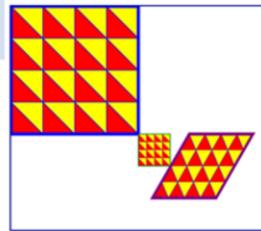
This draws an elliptical arc from the current point to  $(x, y)$ .

The size and orientation of the ellipse are defined by two radii ( $rx, ry$ ) and an x-axis-rotation, which indicates how the ellipse as a whole is rotated relative to the current coordinate system. The centre  $(cx, cy)$  of the ellipse is calculated automatically to satisfy the constraints imposed by the other parameters. large-arc-flag and sweep-flag contribute to the automatic calculations and help determine how the arc is drawn.

## Grouping and transform

- **<g> element for grouping**
- **Transform property: translate, scale, rotate, skewX, skewY**

```
<g transform="translate(200, 200)">
  <g transform="scale(0.25)">
    <rect x="0" y="0" width="200" height="200"
fill="url(#trianglePattern)" stroke="green" stroke-width="5">
  <g transform="translate(320, 0), scale(2), skewX(-30)">
    <rect x="0" y="0" width="200" height="200"
fill="url(#trianglePattern)" stroke="purple" stroke-
width="10" />
  </g>
</g>
</g>
```



312

When you want to apply an effect to a collection of elements, you use the grouping element, `<g>`. Grouping is also useful for other purposes, for example, if you want to move several elements at the same time.

The transform attribute accepts a space-separated list of commands that are applied in order.

You can set the transform property to "translate", "scale", "rotate", "skewX", or "skewY".

## Canvas vs. SVG

- **SVG is an alternative W3C spec for creating graphics**
- **Vectors vs. pixels**
  - SVG is vector based
  - Canvas offers pixel operations
- **Document vs. script**
  - SVG images are defined in XML
    - They are part of the DOM and can be accessed with JS and CSS
  - Canvas is script-based and requires JavaScript to be on
    - All canvas images are built using a series of API drawing commands
    - A single entity unreachable by the DOM
- **Browser support**
  - Canvas is near universal in support
  - SVG is a plug in for IE8 or less

313

## Which Should You Choose?

In general, SVG is best suited to scalable and interactive graphics. Canvas is the best option for fast games or animations where hundreds of elements are being rendered. There will be situations where either format could be used, such as data charts.

More cautious developers will probably avoid SVG and canvas until a large majority of users have one or both enabled. However, they are viable technologies and there's little reason why you shouldn't investigate them further. They're certainly an option for progressive enhancement techniques — for example, IE8 and earlier versions show a table of data whereas supported browsers show an animated pie chart.

Are you using SVG or canvas within your projects, or is it too early to adopt these technologies?

## Summary

- **SVG has good support across browser except IE8 and below**
- **SVG graphics are scalable to different display resolution**
- **You can draw common shapes, insert images, and create text with path**
- **You can also group elements and create transformation with them**

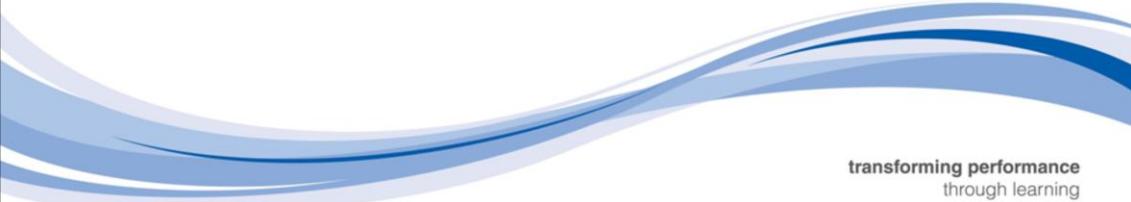
## Review questions

- **What are the different ways to apply style to SVG elements?**
- **What are some of the common shapes you can draw with SVG?**
- **How do you insert images in SVG?**
- **What element allows you to group SVG?**
- **What are some of the differences between SVG and Canvas?**



# Canvas

Developing Web Applications using HTML5



A decorative graphic consisting of several overlapping, curved blue lines of varying shades, creating a sense of motion and depth.

transforming performance  
through learning

## This chapter covers

- **What is <canvas>?**
- **The rectangle methods**
- **Adding paths**
- **Drawing text**
- **Gradients**
- **Images**

## In this chapter you will learn

- How to create HTML5 Canvas and draw various shapes
- How to add paths and draw text in Canvas
- How to add gradients and images to Canvas

## What is <canvas>

- **Canvas is a API for 2D drawing**
  - No plug in required and supported heavily now

IE	FF	Safari	Chrome	Opera	iOS	Android
9+	3.0+	3.0+	3.0+	10.0+	1.0+	1.0+

- **Topic large enough to have its own spec document**
  - New tags
  - New set of JavaScript objects
  - Ability to create RIA with no plugins

319

Canvas is the tool creating the greatest buzz on the web right now and could truly see competition to Flash and Silverlight once the feature set and IDE designers become more adept at developing the mark up.

The spec defines that canvas is a resolution independent bitmap canvas which can be used for rendering graphs, game graphics or other visual graphics on the fly.

Though the support is strong across the board for Canvas (and its SVG roots) it has a few issues. Internet Explorer uses the third-party canvas library plugin to make it work for instance.

## A blank <canvas>

- The <canvas> element is a container

```
<canvas width=300 height=225 id=a></canvas>
```

- Write to it by first getting a reference to the object

```
var aCanvas = document.getElementById("a");
```

- Every canvas has a drawing context

```
var aContext = aCanvas.getContext("2d");
```

- Then use this reference to draw

```
aContext.fillRect(50, 35, 150, 100);
```

320

Adding the canvas is easy, the problem is it is an empty object so you will want to give it an ID so you can modify it. Once you have the canvas you need to reference it via JavaScript.

This simple DOM reference is the gateway to far more interesting things. Next you need to call the **getContext()** method to begin drawing. This method has a mandatory value of **2d**.

With this reference you can begin drawing to the canvas. The canvas is a **2d** grid where **0x** and **0y** are top left . The example above creates a simple rectangle to start off with. It will be black to begin with which is the default **fillStyle** for any shape which we will learn to control shortly.

Any drawing added to the canvas will remember its own properties, its context in fact, for as long as the page is open or you do something to reset the canvas.

## The rectangle methods

- **There are methods for drawing rectangular shapes**
- `fillRect(x, y, width, height)`
  - Draws a rectangle filled with the current fill style
- `strokeRect(x, y, width, height)`
  - Draws an rectangle with the current stroke style
  - StrokeRect only draws the outline
- `clearRect(x, y, width, height)`
  - Clears the pixels in the specified rectangle

321

The canvas is a two-dimensional grid. The coordinate (0, 0) is at the upper-left corner of the canvas. Along the X-axis, values increase towards the right edge of the canvas. Along the Y-axis, values increase towards the bottom edge of the canvas.

Calling the `fillRect()` method draws the rectangle and fills it with the current fill style, which is black until you change it. The rectangle is bounded by its upper-left corner (50, 25), its width (150), and its height (100).

## Adding paths

- There are two key methods to drawing paths

```
context.moveTo(x, y);           //moves the 'pencil' to the starting point  
context.lineTo(x, y);          //draws the line
```

- These just '*mark out*' the lines nothing is drawn yet

- Set a strokeStyle
- Then draw with the stroke() method

```
context.strokeStyle = "#bbb";    //set the colour  
context.stroke();                //draw the line
```

- To change line style start a new path

```
context.beginPath();            //only needed for a new path
```

## Drawing text

- **If you add text there is no box model**
  - Positioning can be challenging
  - Elements need to be positioned within the canvas
- **Instead the following attributes are available**
  - `font` can be anything you would put in a CSS font rule
    - That includes font style, font variant and font family
  - `textAlign` controls text alignment
    - Similar to a CSS text-align rule
  - `textBaseline` where text is drawn relative to the starting point
    - top, hanging, middle, alphabetic, ideographic, or bottom

323

font can be anything you wound like to add to a CSS rule.

`textAlign` is used for text alignment; its possible values are `start`, `end`, `left`, `right`, and `center`.

`textBaseline` is tricky, because text is tricky. The HTML5 specification explains the issue:

*"The top of the em square is roughly at the top of the glyphs in a font, the hanging baseline is where some glyphs like 3π are anchored, the middle is half-way between the top of the em square and the bottom of the em square, the alphabetic baseline is where characters like Á, ÿ, f, and Ω are anchored, the ideographic baseline is where glyphs like 私 and 達 are anchored, and the bottom of the em square is roughly at the bottom of the glyphs in a font. The top and bottom of the bounding box can be far from these baselines, due to glyphs extending far outside the em square."*

## The baseline problem

Consider the following illustration:



324

The following graphic, reproduced under Creative Commons 3.0, demonstrates the different baseline options.

## Adding text

- Text inherits the font style of the canvas

- You can override

```
context.font = "bold 12px sans-serif"; //much like CSS font settings
```

- The `textAlign` and `textBaseline` properties position

```
context.textAlign = "right";
context.textBaseline = "bottom";
```

- The `fillText` method adds the text

```
context.fillText("string", x , y);
```

- All properties must be set before `fillText()` is called

- The `fillText` function draws the text object
  - Its settings are preserved until they are changed

## Gradients

- Linear and radial gradients are supported

	IE	Firefox	Safari	Chrome	Opera	iOS	Android
linear gradients	9.0+	3.0+	3.0+	3.0+	10.0+	1.0+	1.0+
radial gradients	9.0+	3.0+	3.0+	3.0+	10.0+	1.0+	1.0+

- Linear gradient paints along a line

- From  $x_0$  to  $y_0$  to  $x_1$ ,  $y_1$

```
context.createLinearGradient(x0, y0, x1, y1);
```

- Radial gradient paints a cone between two circles

```
context.createRadialGradient(x0, y0, r0, x1, y1, r1);
```

326

Two types of gradients are supported in HTML5, linear and gradient, note that gradient support only reaches Internet Explorer in version 9.

The two methods to paint the gradient are:

### createLinearGradient

Linear gradient paints a line from the first two coordinates to the last pair

### createRadialGradient

Paints along a cone between two circles. The first of the parameters represents the starting circle ( $x$ ,  $y$  and radius), the last three the outer one.

## Gradient colour stops

- Once a gradient is declared you can add ColorStop

- There must be two or more

```
var myGradient = context.createLinearGradient(0, 0, 300, 0);  
  
myGradient.addColorStop(0, "black");  
myGradient.addColorStop(1, "white");
```

- The gradient is a fill style and must be applied

```
context.fillStyle = myGradient;  
context.fillRect(0, 0, 300, 225);
```

- Able to create vertical, horizontal and diagonal gradients

327

### Examples:

```
// diagonal  
  
var myGradient = context.createLinearGradient(0, 0, 300,  
225);  
  
myGradient.addColorStop(0, "black");  
myGradient.addColorStop(1, "white");  
  
context.fillStyle = myGradient;  
context.fillRect(0, 0, 300, 225);  
  
//radial  
  
var myGradient = context.createRadialGradient(0, 0, 10,  
0, 225, 300);  
  
myGradient.addColorStop(0, "black");  
myGradient.addColorStop(1, "white");  
  
context.fillStyle = myGradient;  
context.fillRect(0, 0, 300, 225);
```

## Images

- **The canvas has several overloaded drawImage methods**
  - Discussed in the notes
- **Provide an image object and define canvas x and y**

```
drawImage (image, dx, dy);
```

- Image represents an `img` object
- `dx` represents the x position on the canvas
- `dy` represents the y position on the canvas
  - `dx` and `dy` used as starting positions for the image draw
- **Image can be a clone of an existing image or dynamic**
  - Copy the `src` of an `<img>` tag
  - Or load a new graphic from the server

328

The HTML5 specification explains the parameter as the following:

The source rectangle is the rectangle [within the source image] whose corners are the four points  $(sx, sy)$ ,  $(sx+sw, sy)$ ,  $(sx+sw, sy+sh)$ ,  $(sx, sy+sh)$ .

The destination rectangle is the rectangle [within the canvas] whose corners are the four points  $(dx, dy)$ ,  $(dx+dw, dy)$ ,  $(dx+dw, dy+dh)$ ,  $(dx, dy+dh)$ .

The canvas drawing context defines a `drawImage()` method for drawing an image on a canvas. The method can take three, five, or nine arguments.

**drawImage(image, dx, dy)** takes an image and draws it on the canvas. The given coordinates  $(dx, dy)$  will be the upper-left corner of the image. Coordinates  $(0, 0)$  would draw the image at the upper-left corner of the canvas.

**drawImage(image, dx, dy, dw, dh)** takes an image, scales it to a width of `dw` and a height of `dh`, and draws it on the canvas at coordinates  $(dx, dy)$ .

**drawImage(image, sx, sy, sw, sh, dx, dy, dw, dh)** takes an image, clips it to the rectangle  $(sx, sy, sw, sh)$ , scales it to dimensions  $(dw, dh)$ , and draws it on the canvas at coordinates  $(dx, dy)$ .

## Adding an image

- Dynamically adding a graphic:

```
var context = canvas.getContext("2d");
var logo = new Image();
logo.onload = function () {
    context.drawImage(logo, 0, 0);
}
```

- Or a series of graphics:

```
logo.onload = function () {
    for (var x = 0, y = 0; x < 500 && y < 375; x += 50, y
+= 75)
    {
        context.drawImage(logo, x, y, 50, 50);           Sets the width and
                                                       height of the graphic
    }
}
```

## Summary

- **HTML5 Canvas is widely supported in modern browser except IE8**
- **The API provides various method to add shapes, path, text and images**

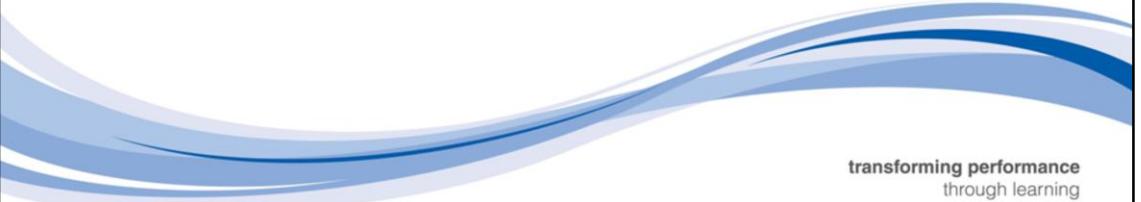
## Review questions

- What is the method needed to begin drawing in Canvas?
- What are the various methods in the rectangle methods?
- When drawing a path, what is the method that will move the pencil to the starting point?
- What are two types of gradients that you can create HTML5 Canvas?
- What methods do you use to add graphic in HTML5 Canvas?



## Video and Audio

Developing Web Applications using HTML5



transforming performance  
through learning

## This chapter covers

- Why `<video>` element?
- Dos and don'ts of `<video>` and `<audio>` element
- The `<video>` element and attributes
- The codec issue
- Browser support of video formats
- The `<source>` element
- Encoding resources
- Custom playback controls
- The `<audio>` element
- Support for legacy browsers

## In this chapter you will learn

- How to add video and audio to your web pages
- How to use create custom controls for video player
- How to encode audio and video files for the web

## Why the <video> element?

- **Multimedia support has always been a delivery issue**
- **Commonly the <object> tag provides the support**
  - With a type `param` to denote the plugin required
  - The non-standard `<embed>` may also be nested
    - For legacy browser support
- **Execution is then passed from the browser to a plugin**
  - User must have correct version of plugin
  - More obscure the plugin the higher likelihood of failure
- **Mobile devices increasingly restrict the media that can be played**
  - Flash and Silverlight are a dying architecture
  - Not supported on Windows 8, iOS and being retired from Android

Lights! Camera! Plugin! Has been the way it has worked up to the arrival of HTML5. The browser passes off the execution to a suitably registered plugin that supports the type supplied by the embed tag, with some collection of parameters to force it render and control as needed. The `<video>` element allows native embedding of video types.

## What <video> and <audio> dos (and don'ts)

- **Fully fledged and recognised parts of the DOM**
  - Can be styled with JavaScript and CSS
  - Fitted to a <canvas> element
  - Controlled by JavaScript
- **The limitations...**
  - No DRM, this is effectively a multimedia <img> tag
    - Many browsers offer the chance to save the video
      - Not Safari interestingly
    - Recently browsers have allowed full-screen playback
  - Older browsers do not support native video
    - Flash and its ilk will be with us for many years yet
    - We can develop a fall-back approach

The video and audio tags do wonderful things, browser independent support of multimedia has been a dream for many years and now it is here. It is also an element that is here now! YouTube were early adopters of this provision as it supported their future ambitions for mobile development and support and is a prime example that the HTML5 living standard lives and dies by adoption. It should be noted that by the time of writing, video and audio support is now almost universal in browser support.

Some companies, such as Apple, promote that HTML5 and the video element will be the death of Flash.

## The <video> element

- **The <video> element has a `src` attribute**
  - By default shows first frame of video but does not start
  - Add an `autoplay` attribute to initialise
  - Add `controls` attribute for user control

```
<video src="video.ogg" autoplay controls>
  <p>your browser fights the future!</p>
</video>
```

- **The browser must support the video type**
  - No video format supported on all devices
  - Different video formats must be supported
  - `src` is only useful for prototyping
  - HTML5 must be all inclusive!

In principle the use of the video element is simple, but the demon of codecs are waiting right around the corner to catch us unawares! Lets focus on the mark up first.

There is a mandatory `src` attribute which points to the video like an `img src` to a picture. This sets the `video` attribute in place but will not start the movie playing. There are two markup driven approaches to this. You can add an `autoplay` attribute which will cause the video to play automatically. Avoid this in the real world it is a bandwidth sink and poor for accessibility.

Instead add `controls` which allows the user to determine what happens next. The appearance of these controls will differ between browsers. Choosing to use JavaScript instead will achieve a more universal appearance.

If you are not auto-starting the video you may want to tantalise people in with an image from the movie. Use the `poster` attribute for this, pointing to a suitable screen shot.

## The <video> attributes

There are a series of optional attributes available.

Attribute	Usage
<code>poster</code>	Screen shot displayed before video is played. Default is first frame of the movie.
<code>height/width</code>	Very necessary, set the size of the video element on the page.
<code>loop</code>	Boolean value false by default.
<code>preload</code>	Set whether the browser should automatically buffer the video for play.

In the same way you would never countenance an image tag without width, you need to set attributes for a video. If you fail to add them the browser should try to access the width from the video resource. If it can not define the width it will use the width of the poster frame. If that fails it will default to 300px. This is especially important if the browser fails to understand the video type.

Preload has three options according to the spec:

auto	Suggests the browser should download the movie in preparation to play. A user agent's settings can override this locally
none	Suggests the user agent does not load the movie until it is started
metadata	Suggests metadata about the movie should be requests, such as frame rate, first frame etc.

## The codec issue

- **The HTML5 spec does not mandate a specific format**
  - It was Ogg Vorbis for audio and Ogg Theora for video
- **Apple and Nokia objected to the inclusion**
  - Looked like Apple's H.264 format was in
  - However it is proprietary
- **Google have provided an open source alternative**
  - VP8 (an Ogg type)
  - Now called webM
  - Broad support
  - Not from Apple



This is where the 'all in it together' ethos of HTML5 gets ugly. Video on the web is a big deal. For a long time video on the web has been done via Adobe Flash. YouTube has already moved to a HTML5 video support based on Apple's H.264 codec.

It looked as though the H.264 codec (MPEG4 Part 10) would become the web standard, especially after Google-owned YouTube started using it to play videos using HTML5 instead of Adobe Flash. Now Google has come out against H.264 and said it will drop support from its Chrome browser. Instead, the company wants everyone to use the VP8 codec, which it bought with On2 Technologies and made open source as WebM. In other words, Google's trying to standardise web video in a format that the iPod, iPhone and iPad can't play. War is declared!

Both Firefox and Opera have refused to use H.264 because it's heavily patented and expensive. Google is likely in saying that HTML5 web video is still an emerging platform. However H.264 is deeply embedded in the rest of the technology industry.

Even if these have software to play WebM videos, it will take more processor power and use more battery power than playing H.264 videos. The trend is for graphics chips to help decode more than one type of video, and several companies are providing hardware acceleration for WebM, but it will be a long time before WebM matches H.264's level of support.

The war continues!

## Browser support video format

Browser support quick check: video formats		MPEG-4	Ogg/Theora	WebM
		5*	5	8
		~	3.5	4
		9	~	**
		~	10.5	11.1
		4	***	***

Google has announced that Chrome will stop supporting MP4 in a future release.

IE9 will support WebM if the user downloads an additional codec.

Safari will support anything that can be played by QuickTime. Users have to download additional codecs.

## The <source> element

- **Media must be supported in multiple formats**
- **The <source> element gives multiple chances**
  - Use instead of src attribute on video tag

```
<video poster="arctic_giant.gif" controls width="512"  
height="385">  
  <source src="arctic_giant.mp4" type="video/mp4"/>  
  <source src="arctic_giant.ogv" type="video/ogg" />  
  <source src="arctic_giant.webm" type="video/webm" />  
  <p>Your Browser does not fully support HTML 5  
video.</p>  
</video>
```

- **Type attribute**
  - Mime type

This is where things get messy. Until we have a universal video format on the web we need to provide three different video sources:

H.264 MP4

VP8 webM

Theora OGG

It is important that they are in this order. There is a 'bug' in iPads which mean they only use the first source and Apple only support H.264. WebM should be second as it is supported in Firefox, Chrome and Opera. OGG should be supported for legacy browsers and Linux systems which may not support the other encoding types. This means unfortunately the proprietary format has to go first which will skew usage stats on the web.

At the date of authoring H.264 is supported in all major browsers, but Google claim to be pulling it soon. WebM is supported in all major browsers except Safari.

There are two attributes required a src as before and a two part type containing the mime type of the video and the codecs used. The codec itself is a compound value consisting of the video codec first and the audio codec second.

The code above is an example of how to use this in most situations .

## MIME and the server

- **Video files must be served with the proper MIME types**
  - Most servers will already do .mov and .avi
  - Likely .ogg and .webM must be added
- **Just adding it to the source tag will not achieve this**
  - The server must include a suitable MIME type in the HTTP response
    - Else the browser may not play the video
- **Beyond the scope of this course to explore all options**
  - Refer to your technical documentation
    - IIS7 – add through the GUI
    - IIS6 – add through the app config dialog
    - Apache – use AddType in your httpd.conf file

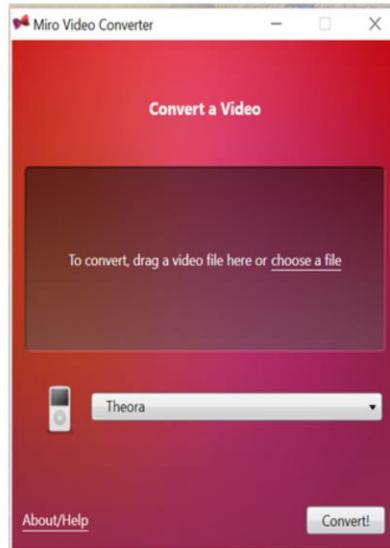
Your server must support the mime type your mark up will request. Failure to do this will cause a crunching failure in the way the server works.

It is beyond the scope of this course to support all potential options that could arise here.

## Encoding resources

Professional tools out there, but there are a few good free ones

- Miro video Convertor
- Handbrake



So we have established that video initially looked really simple and, unfortunately, it is not quite so clear cut. As a result you are going to need to take your video file and export it into a number of different files. Much like image compression it is important you keep your original files uncompressed and untouched as compression into one of these formats results in data loss in the same way as jpeg compression would.

The elephant in the corner in this discussion has been whether we should be building open source standards of propriety formats. It is very important that you provide an H.264 compressed format as many of the hacks for legacy support (you will see some of them in the next exercise) require the format. Even when this legacy format goes away it is the author's informed opinion we will end up with the Apple and Google standards competing.

To summarise, a codec is effectively a container format in the same way as a cab or zip format. It contains a video track and one or more audio tracks. An audio track normally contains markers within it to synch with the video.

## Custom playback controls

- **<video> element can be controlled through JavaScript**
  - Used to give a consistent cross-browser feel
  - Or dynamic functionality

```
function playMovie(video) {  
    var v = document.getElementById(video);  
    v.play();  
}
```

- **This also includes element specific events, for example:**
  - loadstart
  - play
  - seeking
  - timeupdate

As different browsers may render a different controls rendering you may decide you want to create a universal and consistent playback . Or you may want more advanced functionality like chapter point navigation within the movie or a slider to scrub back and forward.

You will explore this in the video exercise.

## The <audio> element

- **Almost identical to the video element**
  - source element needs to be used

```
<audio controls>
  <source src="AJugOfWineAndYou.mp3" type="audio/mp3"/>
  <source src="AJugOfWineAndYou.ogv" type="audio/ogg" />
</audio>
```

Currently, the HTML5 spec defines five attributes for the <audio> element:

- src – a valid <abbr>URL</abbr> specifying the content source
- autoplay – a Boolean specifying whether the file should play as soon as it can
- loop — a Boolean specifying whether the file should be repeatedly played
- controls – a Boolean specifying whether the browser should display its default media controls
- preload – none / metadata / auto – where 'metadata' means preload just the metadata and 'auto' leaves the browser to decide whether to preload the whole file

To create our own controls, we can use the API methods defined by the spec:

- play() – plays the audio
- pause() – pauses the audio
- canPlayType() – interrogates the browser to establish whether the given mime type can be played
- buffered() – attribute that specifies the start and end time of the buffered part of the file

## Browser support audio codec

Browser support quick check: audio codecs		WAV	OGG	MP3	AAC	WebM
	8	5	5	5	5	8
	3.5	3.5	~	~	~	4
	~	~	9	9	9	*
	10.5	10.5	~	~	~	11.1
	4	**	4	4	4	**

IE9 will support WebM if the user downloads an additional codec.

Safari will support anything that can be played by QuickTime. Users have to download additional codecs.

## Video for legacy browsers

- Any markup can occur within the <video> tag
  - A legacy browser will ignore video
  - A HTML5 video-enabled browser will not
    - This can cause a gotcha
- Add an <embed> beneath any <source> elements
  - Resolves legacy browser issues
  - A number of JavaScript/Flash hacks also exist
    - Video For Everyone!
    - FlowPlayer
    - Video.js



Video is extremely powerful and very useful in what it can do but for the time being legacy browsers in the IE family will be a problem in its adoption. So we need to support them.

Remember that a browser ignores tags it does not recognise, so the easiest way to support browsers that do not recognise the video tag is to add an embed tag within the video tag. This approach can cause issue when the browser does understand video, but does not support the video you want it to display. Unfortunately it does not fall through to the legacy tags. There is a javascript approach design by Video For Everybody! You will look at that in the exercise that supports this material.

## Video for everyone

### Video for everyone is dependable markup for video

```
<video width="640" height="360" controls>
  <source src="__VIDEO__.MP4" type="video/mp4" /> 1
  <source src="__VIDEO__.OGV" type="video/ogg" /> 2
  <object width="640" height="360"
    type="application/x-shockwave-flash"
    data="__FLASH__.SWF"> 3
    <param name="movie" value="__FLASH__.SWF" />
    <param name="flashvars"
    value="controlbar=over&image=
      __POSTER__.JPG&file=__VIDEO__.MP4" /> 4
    
  </object>
</video>
```

Video for Everybody is simply a chunk of HTML code that embeds a video into a website using the HTML5 `<video>` element, falling back to Flash automatically without the use of JavaScript or browser-sniffing. It therefore works in RSS readers (no JavaScript), on the iPhone / iPad (which don't support Flash) and on many browsers and platforms.

Thanks to the rapid adoption of HTML5 video happening right now, Video for Everybody isn't the only solution around. It is not a neatly packaged, fully-featured solution for those unfamiliar with HTML. VfE is for developers who either want something really simple they can quickly use on their blog or websites, or as a good starting point to develop their own custom solution. It does not use JavaScript. Because of this, it does not work on Android versions prior to 2.3 (Gingerbread).

[http://camendesign.com/code/video\\_for\\_everybody](http://camendesign.com/code/video_for_everybody)

## Summary

- **HTML5 has make it easy to add video and audio on the web pages**
- **You can easily use JavaScript to create custom controls**
- **Different formats are required for various browser**

## Review questions

- **How do you add video and controls on to web page?**
- **What is the attribute to use to display a screen shot before the video is played?**
- **What are the different formats supported for video elements?**



**Contact us:**

0845 757 3888

[info@qa.com](mailto:info@qa.com)

[www.qa.com](http://www.qa.com)

