CSC1026 Website Design and Construction *(Semester 2)*

Topics

- Background & Landscape (week 1)
  - How the Web Works
- Web Content Creation (week 2)
  - Key Principles: Structure, Presentation & Behaviour
- Web Content: Structure (week 3)
  - HTML & XHTML
  - Tags & Attributes
  - Standards & Validation
- Web Content: Presentation (weeks 4-5)
  - CSS Essentials
  - CSS Box Model & Positioning
  - CSS Browsers & Standards
- Web Content: Behaviour (weeks 7-11)
  - HTML Forms
  - JavaScript & the HTML DOM
  - Form Validation
  - Regular Expressions

Structure (CSC1026)

- 2 Lectures/week
- 2 practical sessions/week (starting from week 2)
  - Demonstrator support
  - Work on practical tasks and assignments

Assessments (CSC1026)

Coursework (35%)

- Two pieces of coursework… *three* deadlines…

Assignment 1: XHTML & CSS
8th March 2019 (W6) 4:00pm
- Specification and details next week

Assignment 2: Developing a web site
pt 1 29th March 2019 4:00pm
pt 2 10th May 2019 4:00pm

Exam (65%)

- Details later in module

Staff & materials…

Jennifer Warrender (module leader) - week 7-12
- Jennifer.warrender@ncl.ac.uk

Sara Fernstad - week 1-6
- Sara.fernstad@ncl.ac.uk

Please add the module code to e-mail subject line!

Materials, announcements, ReCap recordings etc….
- Blackboard

What is the Web ?

What is the Web?

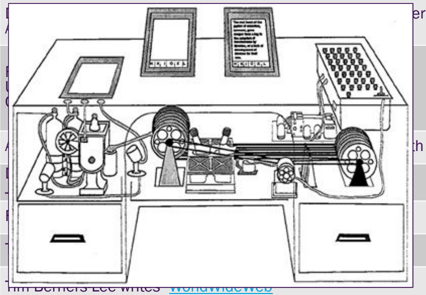
> *"The Web is an abstract (imaginary) space of information … On the Net, the connections are cables between computers; on the Web, connections are hypertext links … The Web made the net useful because people are really interested in information."*
> http://www.w3.org/People/Berners-Lee/FAQ.html#InternetWeb

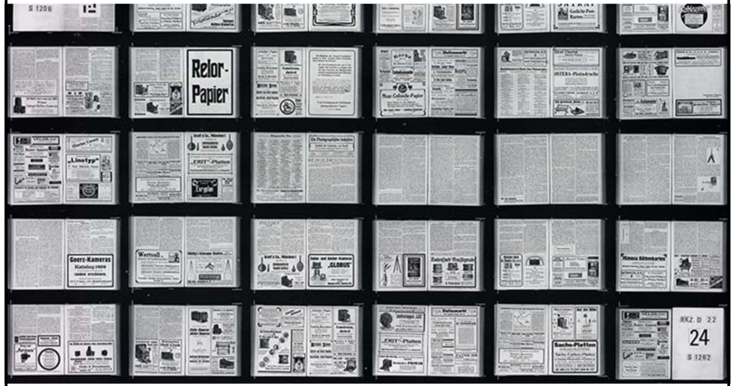
> *The Web "… is a system of interlinked hypertext documents accessed via the Internet"*
> http://en.wikipedia.org/wiki/World_Wide_Web

http://uk.youtube.com/watch?v=p4NKbJPZq2Q
http://uk.youtube.com/watch?v=24WqXehCueg
https://www.ted.com/talks/tim_berners_lee_on_the_next_web?language=en

Web past: Beginnings

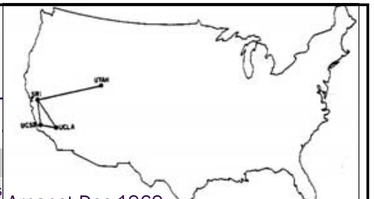
1930s	Vannevar Bush proposes the Memex hypertext device for microfiche
1960s	D ertext" A
1970s	F U C
1982	A th 235 hosts
1984	D J
1988	F
1989	T
1990	Tim Berners-Lee writes "WorldWideWeb" (with GUI hypertext browser and editor for NEXT workstation) First Web server (November) nxoc01.cern.ch (later info.cern.ch)



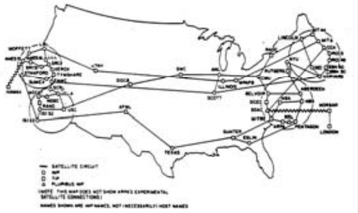


Web past: Beginnin

1930s	Vannevar Bush proposes the
1960s	Doug Engelbart and Ted Nels ARPANET is born
1970s	Ray Tomlinson (BBN) invents UCL first international connec Cerf and Kahn invent TCP (lat
1982	ARPANET adopts TCP/IP - the
1984	Domain Name System (DNS) JANET connects UK universiti
1988	First Internet worm affects 6,
1989	Tim Berners-Lee proposes hy
1990	Tim Berners-Lee writes "Worl (with GUI hypertext browser a First Web server (November) nxoc01.cern.ch (later info.cern.ch)



Arpanet Dec 1969



Arpanet July 1977

http://som.csudh.edu/fac/lpress/history/arpamaps/

Web past: Beginnings

1930s	Vannevar Bush proposes the Memex hypertext device for microfiche
1960s	Doug Engelbart and Ted Nelson independently propose "hypertext" ARPANET is born
1970s	Ray Tomlinson (BBN) invents email over ARPANET UCL first international connection to ARPANET Cerf and Kahn invent TCP (later split into TCP/IP)
1982	ARPANET adopts TCP/IP - the Internet of TCP/IP internets with 235 hosts
1984	Domain Name System (DNS) introduced JANET connects UK universities to Internet
1988	First Internet worm affects 6,000 of 60,000 hosts
1989	Tim Berners-Lee proposes hypertext system for CERN
1990	Tim Berners-Lee writes "WorldWideWeb" (with GUI hypertext browser and editor for NEXT workstation) First Web server (November) nxoc01.cern.ch (later info.cern.ch)

http://som.csudh.edu/fac/lpress/history/arpamaps/

Web past: Take off…

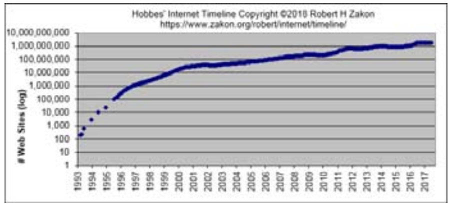
1991	CERN releases World Wide Web technology (WWW) Released to CERN (May)… …then files posted on public FTP (Aug)… http://bit.ly/QFBTIG
1992	26 HTTP servers Public project page in Nov 1992
1993	GUI browsers (including Mosaic) available for all platforms CERN declare WWW technology free to use with no fees The Guardian publishes its first page on Web Over 200 HTTP servers
1994	W3 Consortium (W3C) founded & 1st International WWW conference WWW 2nd most popular service on the Internet (behind FTP) AltaVista search engine and Yahoo launch First banner ads appear…
1995	WWW overtakes FTP as No 1 internet service Amazon and eBay launch
1996	"Browser wars" begin between Microsoft and Netscape
1998	Google launches
1999	Napster launches Forged Bloomberg financial Web page leads to rise of 31% in shares of small technology company

Web past: ...and present

2001	Baltimore train tunnel fire damages backbone fibre optics causing ripple effect across US CodeRed worm infiltrates 1000s of Web servers
2002	Blogging takes off DDoS attack knocks out 5 of 13 DNS root servers
2003	Nearly half of UK homes connected to Internet SQL slammer worm DDoS attack spreads worldwide in 10 minutes
2005	YouTube launches
2007	Google is “most valuable global brand” and most visited Web site
2012	Facebook reaches 1 billion monthly active users
2013	Netflix and YouTube account for over 50% of Internet traffic (in bytes)
2016	United Nations Human Rights Council adopts a resolution on the promotion, protection and enjoyment of human rights on the Internet
2017	Facebook and other social media services are found to have been used by foreign governments to influence elections in the U.S. and other countries

A people's history of the internet (Guardian Tech http://bit.ly/4a1zFQ)
Web history timeline project http://webdirections.org/history/

Growth of the Web



Sites = Number of web servers
(one host may have multiple sites by using different domains or port numbers)

http://www.zakon.org/robert/internet/timeline/
http://www.evolutionoftheweb.com/
http://news.netcraft.com/archives/2016/01/26/january-2016-web-server-survey.html#more-22414

Year	# Web Servers
1990	1
1991	10
1992	50
1993	603
1994	10,022
1995	30,500
1996	603,367
1997	1,681,868
1998	3,689,227
1999	9,560,866
2000	25,675,581
2001	36,276,252
2002	35,543,105
2003	45,980,112

Web present

- The academic/scientific web
  - Research centres, universities (early adopters)
- The commercial web
  - Business to Consumer (Amazon)
  - Consumer to Consumer (Ebay)
  - Business to business data exchange and services
- The social web
  - Blogs, Facebook, Wikipedia, Twitter, Instagram, etc
- The creative web
  - Mashups, Open APIs and free(er) access to data (e.g. Guardian datastore, **data.gov.uk**)

Web future?

- Barriers for change
  - (Virtually) no barrier to new devices to connect
  - Low barrier for new applications to run
  - (Increasingly) high barrier to infrastructure changes
- A semantic web of sorts?
  - Simplified meta information systems & smart agents
  - XML, RDFa and microformats
- (The web) as a service (cloud computing)
  - Infrastructure, Platforms, Software/Applications e.g. Amazon WS, Google AppEngine, Google Apps
- Browser/OS blurring e.g. Chrome OS?

How the Internet works

What is the Internet?

- Networking infrastructure that allows hardware devices to physically connect to one another
  - Servers, Desktop PCs, Mobile devices etc.
  - Based on common standards – notably the TCP/IP protocol suite
- ***Protocols*** specify how the transactions are carried out
  - How communication is initiated and maintained
  - Type, size and structure of the data sent/received
  - Enable the services running over the internet ("the web" is just one!)
- Software ***clients*** (and ***servers***) allow users and devices to access and use internet services
  - Using centralised servers or distributed peer-to-peer systems

Clients, Servers and Peers

- Most common internet services based on a ***client/server model*** using appropriate communication protocol(s)
- Usually ***local*** software ***clients*** access centralised ***remote servers***
  - Email via an Outlook client accessing an ISP's mail server
  - Web pages via a Firefox web client accessing an Apache web server
- ***Peer-to-peer*** technology can directly connect multiple, decentralised computers - each acting as both client and server
  - Responsibly implemented P2P can be very efficient for data-rich services
  - Distributed download, media on demand, VOIP/video calling etc.

Protocols to build services

- Clients and servers typically use a combination (*stack*) of protocols to provide specific services
  - Web access
  - Email
  - File transfer
  - Peer-to-peer data exchange
  - Chat/instant messaging
  - Shell/console access to computers
- Most common protocols in use today are those associated with web and email services
  - HTTP/HTTPS, SMTP/IMAP/POP etc.

Common internet protocols

Protocol		Notes
HTTP	**H**yper**T**ext **T**ransfer **P**rotocol	Core protocol for web transactions
SSL	**S**ecure **S**ockets **L**ayer	Data encryption
HTTPS	Secure HTTP	A combination of HTTP and SSL
SMTP **POP** **IMAP**	**S**imple **M**ail **T**ransfer **P**rotocol **P**ost **O**ffice **P**rotocol **I**nternet **M**essage **A**ccess Protocol	Used in combination to provide most email services
FTP	**F**ile **T**ransfer **P**rotocol	Used to exchange files between computers
SSH and SFTP	**S**ecure **SH**ell and **S**ecure/SSH **FTP**	Encrypted access to remote computers plus extension for encrypted file transfer
BitTorrent	…er… BitTorrent	Protocol for peer-to-peer data access and exchange

TCP/IP: The heart of the Internet

- TCP (**T**ransmission **C**ontrol **P**rotocol)
  - Handles direct connection on computer between client/server software and network interface(s)
  - Breaks outgoing messages from application into ***packets*** to send via IP
  - Assembles incoming packets into messages to pass back to application
- ***Packets*** have two parts:
  - **Header** (addressing/sequencing information and metadata)
  - **Body** (actual data payload)
- IP (**I**nternet **P**rotocol)
  - Handles delivery of packets to/from any device with an ***IP address***
  - a unique numeric identifier for a device on the internet e.g.

`128.240.233.249`

IP: Routing packets

- IP ***routes*** packets using address info in header to direct them to their destination
- IP routing is ***connection-less***
  - Does not require a fixed connection between endpoints
  - Enables data to travel via multiple routes to reach endpoint
  - One "wire" can handle multiple connections/services
  - Packets can be re-routed around points of failure
- But risks include…
  - Corruption of data/loss of packets/duplicate packets arriving
  - Packets delivered out of sequence (sent A -> B, received B -> A)
- As a result IP only promises ***best-effort-delivery***

Best effort delivery?

- Performance of IP routing can be affected by various factors
- Unpredictability means IP cannot guarantee that all packets will arrive…
  - within a fixed time
  - in correct order
  - at all!
- To mitigate, TCP on destination device checks and reassembles packets based on header information and can…
  - Request re-transmission of lost/corrupted packets
  - Correctly re-sequence packets
  - Refuse to accept/wait for any more (drop connection)

A (very) common analogy

Imagine a written message sent as a series of separate, sequenced postcards

- Sequence set by sender, each card addressed to same recipient and posted
- Cards take their own slightly different route through postal system and are delivered (over a period of time) to recipient
- Recipient assembles cards and acts accordingly
  - Put in order and read message?
  - Contact sender and ask for missing cards?
  - Stop waiting and ignore future deliveries?
  - Ask sender to stop sending cards?

TCP
↓
IP
↓
TCP

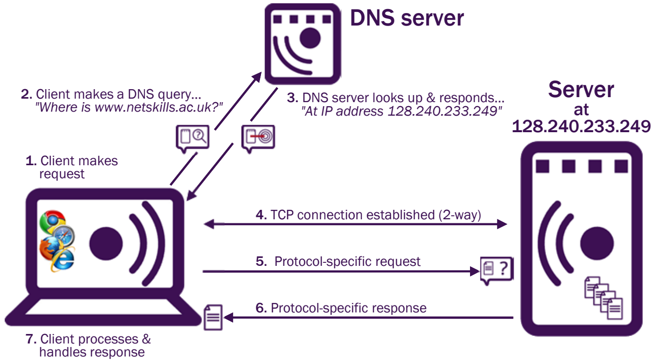
Domain names

- Domain names make internet easier (for humans) to use by mapping numeric IP addresses to text-based names e.g.

```
128.240.233.249 <=> www.netskills.ac.uk
```

- When device makes an request using a domain name it is resolved back into a numeric IP address by a DNS server
  - Thousands of DNS servers across internet
  - Your device/ISP will primarily point to one
- New domain registrations are automatically propagated across global DNS system

A simple request



DNS server

Server at 128.240.233.249

1. Client makes request

2. Client makes a DNS query... *"Where is www.netskills.ac.uk?"*

3. DNS server looks up & responds... *"At IP address 128.240.233.249"*

4. TCP connection established (2-way)

5. Protocol-specific request

6. Protocol-specific response

7. Client processes & handles response

Web clients

Web clients

- Software to create and send HTTP requests…and handle the responses
- Mosaic kick-started mainstream interest - followed by Internet Explorer & Netscape
  - Internet Explorer survives with: Firefox, Chrome, Safari, Opera… plus… Camino, Seamonkey, Flock, Konqueror, iCab, Web TV, MSN Explorer, AOL, Omniweb, Lynx… etc…
- "Under the hood" many share common layout engines e.g. Trident, Gecko, KHTML/Webkit etc.

http://www.upsdell.com/BrowserNews/overview.htm
http://en.wikipedia.org/wiki/Timeline_of_web_browsers

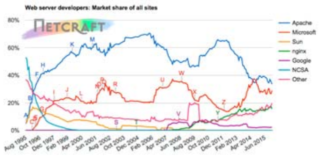
Typical Web client features

Request generation	Assemble URIs and initiate the request(s)
Response handling	Accept server response codes and act accordingly
Content parsing	Read/process HTML markup & CSS from response
Maintain state	Create and manage storage of local tokens & cookies
Client-side scripting	Run JavaScript supplied by server response
Encryption	Manage SSL encrypted transactions (over HTTPS)
Media objects	Display/play embedded media objects

Web servers

Web servers

- Term *web server* refers to software not a physical box… one box can run multiple servers
- Server *listens* on an open network port for incoming HTTP requests and responds accordingly
- Original specs from CERN released in 1991
  - Until 1995 dominant server was the **"NCSA server"**
  - NCSA spawned the **Apache project** (#1 by end of 1996)
  - Apache still dominates the sever market



http://news.netcraft.com/archives/2016/01/26/january-2016-web-server-survey.html#more-22414

Typical Web server features

Path translation	Locate resource specified in the URL
Virtual hosting	Run multiple websites from one server
Access restriction	Control who can access the resources
Log access & errors	Record what's going on (and what went wrong!)
Encrypted operation	Protect sensitive data in transit
Load balancing	Manage traffic and processing resources
Server scripting	Run external programs and applications

Web architecture:
HTTP, URIs and URLs

Key web architecture components

- **HTTP/1.1**: HyperText Transfer Protocol
  - Format and semantics of request/response messages
- **URI**: Uniform Resource Identifier
  - Formatted string that identifies a resource
- **HTML/XHTML**: HyperText Markup Language

Plus…

- **DNS**: Domain Name System
- **TCP/IP**: Internet Protocol Suite

HTTP: Request-response protocol

- HyperText Transfer Protocol makes the Web work!
  - ***Clients*** ask for resources from servers by assembling and sending an *HTTP* ***request*** message
  - ***Servers*** respond with the appropriate *HTTP* ***response*** message, including any content to be displayed
- HTTP requests & responses travel as TCP/IP packets:
  - Metadata in ***headers*** & content in an (optional) entity ***body***
- HTTP is ***stateless***
  - Each request/response pair is an independent exchange.
  - No protocol level maintenance of state (for scalability)

HTTP methods

- The operations carried out over HTTP
  - HEAD, GET, POST, PUT, DELETE etc…
- HEAD and GET are mandatory
  - Anything handling HTTP supports them
- GET returns current state and content of resource
  - This is the "default" method for HTTP
- HEAD just returns response metadata
  - i.e. a GET without the body (content)
- Other methods are optional

Common HTTP methods

Method	Use	Safe*	Idempotent**	Mandatory
HEAD	Exchange of request/response headers	Yes	Yes	**Yes**
GET	Request and return the current state and content of a resource e.g. access a web page	Yes	Yes	**Yes**
POST	Request uses entity body to update resource or as input for processing e.g. form input	No	No	No
PUT	Server stores entity body contents at request URI location e.g. file uploads	No	Yes	No
DELETE	Deletes identified resource i.e. opposite of PUT	No	Yes	No

* *Safe*… does not change state of resource

** *Idempotent*… the side effects of repeated, identical requests are the same as for a single request

HTTP headers

- *Headers* are the metadata for the request/response exchange
- Some are generic and apply to both request &response e.g.
  - ***Date*** -> Date/time stamp for the message
  - ***Cache-Control*** -> Instructions for en-route caching (or not)
- Understanding the reading, setting and manipulation HTTP headers is very useful for managing a web site
- Be aware that headers can be spoofed – not good!

More HTTP headers

- **Request** headers (4 classes)

Response preferences	->	`Accept`, `Accept-Charset` etc…
Additional request info	->	`Authorization`, `From` etc…
Conditional headers	->	`If-Modified-Since` etc…
Constrain server behaviour	->	`Max-Forwards` etc…

- **Response** headers (4 classes)

Redirection	->	`Location`
Additional server info	->	`Server`, `Retry-After` etc…
Authentication	->	`WWW-Authenticate`, `Proxy-Authenticate`
Caching	->	`Age` etc…

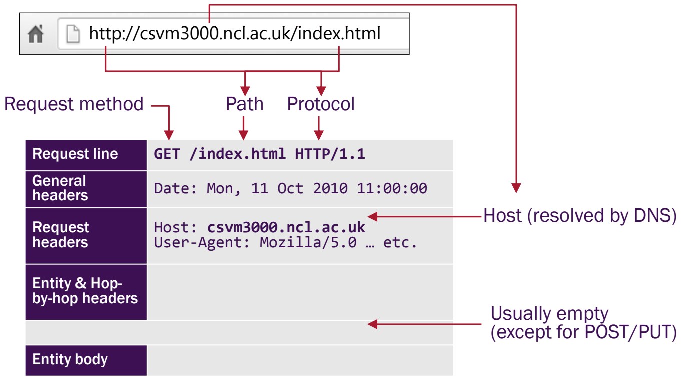
Making an HTTP request

- Any given web resource will have a specific **URL** (Uniform Resource Locator)

HTTP URI examples

Scheme	Authority	Path	Query	Fragment
`http:`	`//csvm3000.ncl.ac.uk`	`/`		
`http:`	`//csvm3000.ncl.ac.uk`	`/index.html`		
`http:`	`//csvm3000.ncl.ac.uk`	`/urls/index.php`	`?msg=hello`	`#demo`
`http:`	`//www.bbc.co.uk`	`/news/technology/`		
`https:`	`//www.google.co.uk`	`/search`	`?q=csc3422`	

Basic HTTP request structure



http://csvm3000.ncl.ac.uk/index.html

Request method

Path

Protocol

Request line	`GET /index.html HTTP/1.1`
General headers	`Date: Mon, 11 Oct 2010 11:00:00`

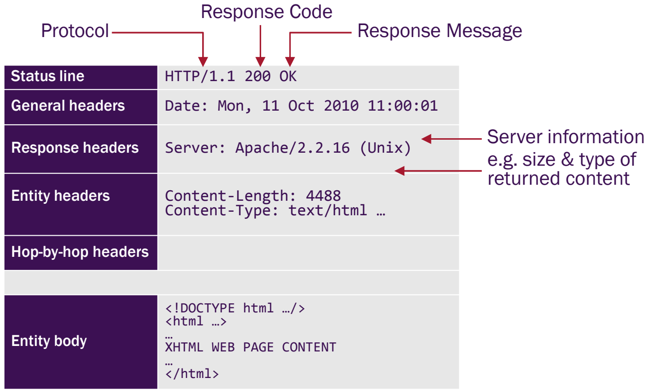
Request headers	`Host: csvm3000.ncl.ac.uk`
	`User-Agent: Mozilla/5.0 … etc.`

Entity & Hop-by-hop headers	
Entity body	

Host (resolved by DNS)

Usually empty
(except for POST/PUT)

Basic HTTP response structure



Protocol

Response Code

Response Message

Status line	`HTTP/1.1 200 OK`
General headers	`Date: Mon, 11 Oct 2010 11:00:01`
Response headers	`Server: Apache/2.2.16 (Unix)`

Entity headers	`Content-Length: 4488`
	`Content-Type: text/html …`

Hop-by-hop headers	

Entity body	`<!DOCTYPE html …/>`
	`<html …>`
	`…`
	`XHTML WEB PAGE CONTENT`
	`…`
	`</html>`

Server information

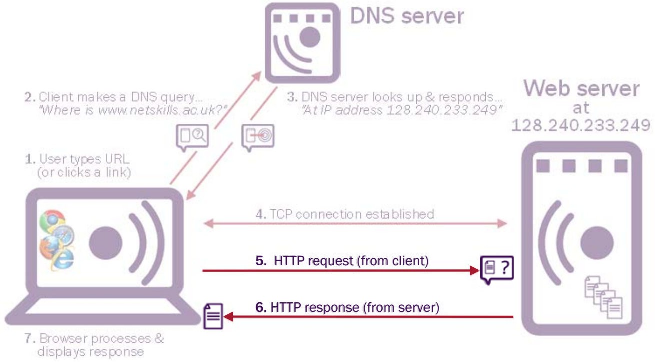
e.g. size & type of returned content

HTTP response codes

- Generated by the server tell a client the *status* of a request
  - 41 response codes in total (some you'll never see!)

Class	Range	Examples
Informational	1XX	100 *Continue* 101 *Switching* Protocols
Success	2XX	**200 *OK*** 201 *Created* 204 *No Content*
Redirection	3XX	300 *Multiple Choices* **301 *Moved Permanently***
Client Error	4XX	400 *Bad Request* **401 *Unauthorized*** **403 *Forbidden*** **404 *Not Found***
Server Error	5XX	**500 *Internal Server Error***

A simple *Web* request



DNS server

Web server
at
128.240.233.249

1. User types URL
(or clicks a link)

2. Client makes a DNS query…
"Where is www.netskills.ac.uk?"

3. DNS server looks up & responds…
"At IP address 128.240.233.249"

4. TCP connection established

5. HTTP request (from client)

6. HTTP response (from server)

7. Browser processes &
displays response

Web Content

Key Principles

Web content basics

- Source code delivered to client (browser)
  - Flat file, dynamic generation, via HTTP, via local file system etc.
- Browser processes document and renders display to user
  - Final appearance is a combination of source construction (by the author) and rendering capability (in web browser)
- All web documents use HTML – Hypertext Mark-up Language
- Other embedded technologies enhance display and behaviour

A web document typically has three concerns

- **Content** – the information conveyed by a page
  - Meaningful structure (section headings, paragraphs, emphasis etc.) and subject matter
- **Presentation** – the appearance of a page
  - Typefaces
  - Layout
  - Colour schemes & graphics
  - Eye-candy
- **Behaviour** – interactive or responsive functionality
  - Respond to user input
  - Manage external data manipulation
  - Handling browser inconsistencies/requirements
  - More eye-candy!

Separation of concerns

- Content structure and semantics

  HTML
- Presentation style and layout

  CSS
- Behaviour scripting and interactivity

  JavaScript (client-side)

  PHP, Perl, Java, Ruby etc. (server-side)

Benefits of separation

- Separation of content from presentation tells you (and your device) something meaningful about a page…

  …independently of its appearance…

  - Semantic markup versus stylistic instructions
  - Heading level versus large font
  - Strong emphasis versus bold
  - New paragraph versus line break
- Plus consistency, accessibility, ease of maintenance etc.

HTML: A simple view

- Content
  - Text (and images)
- Elements (Tags)
  - Core structure for the content
  - Focus on semantics and organisation
  - Default rendering conventions allow for basic (functional) display
- Attributes
  - Additional information/semantics about elements

CSS: A simple view

- Allows visual presentation to be applied to structured mark-up
- Pattern matching syntax identifies where to apply style
- Property: Value syntax specifies what to apply
- Allows reflowing/positioning of content
- Can include external images, animation and transformations

Client-side Scripting:<br>A simple view

- Programmes delivered alongside web content
  - Part of page
  - Linked to a page
- Compiled and executed on client-side
- Interacts via the web page Document Object Model (DOM) to read, write and manipulate
  - HTML elements and attributes
  - CSS rules
  - Client features and capabilities
- Allows rich interaction and dynamic functionality

A simple web page

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
 <title>A Simple (HTML) Document 1</title>
 <meta http-equiv="content-type"
   content="text/html;charset=iso-8859-1">
</head>
<body>
<div><img src="logo.gif" alt="My Logo"></div>
<h1>
 <font face="Georgia, Times New Roman, serif" color="green" size="5">
 Creating Web Pages</font>
</h1>
<p>
 <font face="Arial, Helvetica, sans-serif" color="red">
 A <strong>one-day workshop</strong> run by:<br>
 <a href="http://www.netskills.ac.uk/">Netskills</a></font>
</p>
</body>
</html>
```

One degree of separation

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>A Simple (HTML) Document</title>
 <meta http-equiv="content-type"
   content="text/html;charset=iso-8859-1">
 <style type="text/css">
  h1 { font-family: Georgia "Times New Roman", serif;
       color: green; font-size: 155%; }
  p  { font-family: Arial, Helvetica, sans-serif; color: red; }
 </style>
</head>
<body>
 <div><img src="logo.gif" alt="My Logo"></div>
 <h1>Creating Web Pages</h1>
 <p>A <strong>one-day workshop</strong> run by: <br>
 <a href="http://www.netskills.ac.uk/">Netskills</a></p>
</body>
</html>
```

Two degrees of separation

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>A Simple (HTML) Document</title>
 <meta http-equiv="content-type"
   content="text/html;charset=iso-8859-1">
 <link rel="stylesheet" type="text/css" href="style.css" >
</head>
<body>
 <div><img src="logo.gif" alt="My Logo"></div>
 <h1>Creating Web Pages</h1>
 <p>A <strong>one-day workshop</strong> run by: <br>
 <a href="http://www.netskills.ac.uk/">Netskills</a></p>
</body>
</html>
```

style.css

```
h1 { font-family: Georgia "Times New Roman", serif;
     color: green; font-size: 155%; }
p  { font-family: Arial, Helvetica, sans-serif;
     color: red; }
```

HTML, XHTML & HTML5

Web Pages: Content

HTML versions

- HTML was originally designed to be very simple and handle text-based documents
  - As the web became popular and browsers developed, new features were added to subsequent versions
- HTML 2.0 -> HTML3.2 (1996) standardised common features
  - Page structure, Forms, Images, Tables, Frames
- HTML 4.01 (1999) added enhancements for new technology
  - Support for CSS (Cascading Style Sheets)
  - Support for dynamic scripting with JavaScript
  - Accessibility features
  - Standardised support for multimedia and embedded objects

Problems with HTML

- HTML originally only targeted at Web browsers on PCs and workstations
  - Forgiving of syntax errors and variations in HTML markup (elements with and without end tags, uppercase and lowercase tag names etc.)
  - Requires more processing at browser than a markup language with stricter syntax
  - Causes problems for constrained devices and for machine-oriented processing of markup
- Led to development of **XHTML** 1.0 (2000/02)

XHTML

- The syntax for HTML was tightened up using XML mark-up rules
- The result... **XHTML**... *"HTML as an application of XML"*
- XHTML uses the same tags and attributes as HTML 4.01
- Differences are in structure and syntax
  - Tag/attribute names ***must*** be in ***lower*** case
  - Attributes ***must*** be name/value pairs
  - Tags ***must*** both open ***and*** close
  - Self-contained tags ***must*** be closed
  - Special characters ***must*** use correct entity values

XML?

- Extensible Markup Language
  - A standardised, but flexible specification for creating markup languages
  - Derived directly from SGML (as was HTML)
- Designed for online data description and exchange
  - Strict but simplified core rules for structuring documents
  - e.g. RSS, XHTML, SVG etc.
- A well-formed XML document can be by *any* XML parser
- Provides greater consistency
- Final context applied by client/application
  - e.g. web browser, news aggregator etc.

Advantages of XHTML

Good

- Designed for internationalization, accessibility, device-independence, usability and document structuring
- Lead developers towards enforced separation of concerns

Not so good

- Complex evolution
- Requires new/re-engineered renderers to benefit over existing HTML development
- Unwieldy in contemporary use cases

Q. What is "HTML5"...?
A. More than just HTML ...!

Logical extension of our three concerns:

- Markup – developed from HTML4.01/XHTML
- Presentation – extensions to CSS ... CSS3 ☺
- Behaviour – extensions to DOM

Some core principles:

- Improve semantic organisation
- Replace scripting with core markup where possible
- Reduce need for browser plugins
- Improved device independence

Why is it important?

- Provides a cross-platform, device independent option for rich application development – not just "web pages"
- Includes features such as:
  - Semantic content structure elements
  - Logical video and audio elements for media playback
  - New form controls – with baked in validation
  - Animation and drawing with CSS3, <canvas> and SVG
  - Drag and drop interfaces
  - Support for local offline storage

HTML, XHTML or HTML5?

- Simple answer is… "If you do it right…it shouldn't matter"
  - What are the requirements for your web site?
  - Are there any existing QA/workflow rules specifying one over the other?
- Whatever you produce, check it complies with the current standard for that version

  http://validator.w3.org

  http://validator.nu/
- In many cases you may not get a choice
  - A CMS may be making the decision for you
  - Design specifications from a client
  - Application restrictions

Reference URLs

- W3C HTML Home page
  - http://www.w3.org/html/
- W3Schools tutorials
  - http://www.w3schools.com/html/
- HTML 4.01 specification
  - http://www.w3.org/TR/html4/
- XHTML 1.0 specification
  - http://www.w3.org/TR/xhtml1/
- XHTML specifications
  - http://www.w3.org/TR/html5/
  - http://developers.whatwg.org/
  - http://www.whatwg.org/specs/web-apps/current-work/multipage/

Document Structure

Web Pages: Content

Why is structure important for HTML?

- You are creating materials to be processed at the point of use
  - Compare with printed material, you have less control over final output
- Correctly structured HTML…
  - Allows for consistent rendering in browsers
  - Means users can take advantage of accessibility features for display and navigation of pages
  - Makes editing and maintaining documents easier
  - Gives page authors full access to style sheet and scripting features

Web page specifications

- The specifications for HTML are provided by the W3C (World Wide Web Consortium)
- W3C publishes the Document Type Definitions (DTD) for each version of HTML
- A DTD contains the rules for a markup language e.g.
  - Which tags can be used
  - Where they can appear in the document
  - Which attributes they can hold
- Alongside the DTD are recommendations as to how *user-agents* (e.g. web browsers) should interpret and render the marked up content

HTML: Basic document structure

- Specification set by the W3C

An HTML document is composed of three parts:

1. *A line containing HTML version information*
2. *A declarative header section (delimited by the <head> element)*
3. *A body, which contains the document's actual content…implemented by the <body> element…*

Sections 2 and 3 should be delimited by the <html> element http://www.w3.org/TR/html401/struct/global.html

"A line containing HTML version information"

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
```

- The DTD (Document Type Definition) declaration
- Identifies the document version to the user-agent
  - The PUBLIC part identifies the version e.g. XHTML 1.0, HTML4
  - The URL specifies the location of the mark-up specification
- Common browsers do not validate pages against DTD but…
  - A DTD will allows a browser to process document correctly
  - Sometimes called *DOCTYPE switching*

http://hsivonen.iki.fi/doctype/

"A declarative header section"

```
<head>
 <title>A Simple Document</title>
 <meta http-equiv="content-type" content="text/html;charset=utf-8">
</head>
```

- Information *about* the document, used to help process it
  - Must include a `<title>` and character set info
- Optional declarations and references for:
  - Metadata via `<meta>`
  - Scripts via `<script>…</script>`
  - Style sheets via `<style>…</style>` and/or `<link>`
- NO content!

"A body, which contains the document's actual content"

```
<body>
 <div><img src="logo.gif" alt="My Logo"></div>
 <h1>Creating Web Pages</h1>
 <p>A <strong>one-day workshop</strong> run by: <br>
 <a href="http://www.netskills.ac.uk/">Netskills</a>
 </p>
</body>
```

- This is processed into the content users actually see!
- Final appearance may be influenced by information from the `<head>` (scripts, style sheets etc)

"Sections 2 & 3 should be delimited by the <html> element"

```
<html>
<head>
   …etc…
</head>
<body>
  …etc…
</body>
</html>
```

HTML 4

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  …etc…
</head>
<body>
  …etc…
</body>
</html>
```

XHTML

An HTML 4.01 document

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">

<html>
<head>
 <title>A Simple (HTML) Document</title>
 <meta http-equiv="content-type" content="text/html;charset=utf-8">
</head>
<body>
 <div><img src="logo.gif" alt="My Logo"></div>
 <h1>Creating Web Pages</h1>
 <p>A <strong>one-day workshop</strong> run by: <br>
 <a href="http://www.netskills.ac.uk/">Netskills</a></p>
</body>
</html>
```

An XHTML 1.0 document

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
 <title>A Simple (HTML) Document</title>
 <meta http-equiv="content-type" content="text/html;charset=utf-8" />
</head>
<body>
 <div><img src="logo.gif" alt="My Logo" /></div>
 <h1>Creating Web Pages</h1>
 <p>A <strong>one-day workshop</strong> run by: <br />
 <a href="http://www.netskills.ac.uk/">Netskills</a></p>
</body>
</html>
```

XHTML 1.0 DTD

Extra attribute in `<html>` tag

Self-closing non-paired tags e.g. `<br />` instead of `<br>`

An HTML5 Document

```
<!DOCTYPE html>
<html>

  <head>
    <title>Some HTML5</title>
    <meta charset="utf-8" />
  </head>

  <body>

    Page Content…

  </body>
</html>
```

Simplified DOCTYPE

XHTML syntax is OK
(as is HTML!)

Tags & Attributes

Web Pages: Content

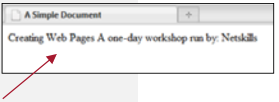
Basic document structure

DTD – version information

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>
 <title>A Simple Document</title>
 <meta http-equiv="content-type"
   content="text/html;charset=utf-8" />
</head>

<body>
 Creating Web Pages
 A one-day workshop run by:
 Netskills
</body>

</html>
```

`<html>` – wraps all markup

`<head>` – *information about* the page *not* for display



Creating Web Pages A one-day workshop run by: Netskills

`<body>` – content for display

Tags

- Usually wrapped around content in pairs i.e. begin/end

```
<h1>This is a Heading</h1>
```

- Some are not paired (never enclose any content)

```
<br /> <hr />
```

- Inline tags can be nested inside block level tags

```
<p>The <em>useful</em> bit of this paragraph</p>
```

- Some block level tags can be nested together

```
<div>
 <p>A paragraph</p>
 <p>Another paragraph</p>
 <p>Yet another paragraph</p>
</div>
```

Attributes

- Specify additional properties and/or behaviour for HTML tags as name/value pairs

```
<tag attribute="value">content</tag>
```

- Attributes are declared in the opening tag

```
<div id="content-block-1">content</div>
```

- Self-closing tags can contain attributes too

```
<img src="logo.gif" alt="My Logo" />
```

- All attributes in XHTML must have a value

```
checked becomes checked="checked"
```

Common attributes

- These attributes can be applied to any `<body>` tag
- Typically used to proved the framework for enhancements to the user experience

Attribute	Purpose
`class`	Associates an element with a CSS class
`id`	Uniquely identifies element for CSS/scripting
`style`	Provides inline CSS style rules for an element
`title`	Describes an element and its content. Creates tool-tips in browsers & used by screen readers

Body tags

- **Block-level**
  - Define blocks(!) of content
  - Browser will add new lines above and below
  - Default width/height handled automatically
- **Inline**
  - Semantics and organisation within block
  - Not associated with new lines
  - Must be nested inside a block-level element
- **Replaced**
  - Browser calculates dimensions and *replaces* with embedded or drawn objects
  - Nesting as for inline tags
  - Used for specific page elements such as **images** and **form controls**

HTML5 Element classification

- HTML5 reclassified tags into "more logical" groups based on *content model*
  - **"Heading"**, **"Flow"** and **"Sectioning"** elements… *mostly* block-level
  - **"Phrasing"** elements… *mostly* inline
  - **"Embedding"** elements… images and other included items
  - **"Interactive"** elements… including form controls
  - **"Metadata"** elements… about the document and linked code
- Organised by use in document not rendering rules
  - Some elements appear in multiple categories

Block-Level Tags

Web Pages: Content

Block-level tags

- Block-level tags provide basic content structure for a web page
- Text structure e.g.
  - Headings
  - Paragraphs
  - Horizontal rules
- List organisation
- Tables
  - Primarily for data presentation
  - Can also be used (with care) for simple content layout

Headings

- Most important structural component for a page
  - Sets reading patterns, allowing users to scan content
  - Provides core accessibility for screen readers etc.
- Can be set to six levels `<h1>` to `<h6>` (one `<h1>` per page)
  - Each heading is a separate block (cannot be nested)

`<h1>Welcome<h2>Part 1</h2></h1>` ✗

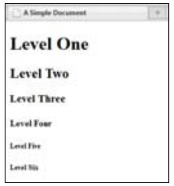
  - Levels should not be skipped

`<h1>-<h2>-<h3>` ✓

`<h1>-<h2>-<h3>-<h2>` ✓

`<h3>-<h3>-<h3>` ✗

`<h1>-<h3>-<h1>` ✗



Level One

Level Two

Level Three

Level Four

Level Five

Level Six

Paragraphs, breaks & rules

- Paragraph used as basic container for body text

```
<p>A one-day workshop run by Netskills</p>
```

- Line breaks can be forced (but try and avoid)

```
<p>A one-day workshop run by <br />Netskills</p>
```

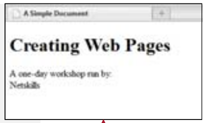
- Sections can be easily delineated using a horizontal rule

```
<p>A one-day workshop run by Netskills</p>
<hr />
<h2>About Netskills</h2>
```

Adding block-level structure

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
 <title>A Simple Document</title>
 <meta http-equiv="content-type"
   content="text/html;charset=utf-8" />
</head>
<body>
 <h1>Creating Web Pages</h1>
 <p>A one-day workshop run by:
 <br />
 Netskills
 </p>
</body>
</html>
```

Heading <h1>…</h1>, paragraph <p>…</p> and line breaks <br /> define the basic structure and layout of the page content


A Simple Document

Creating Web Pages

A one-day workshop run by:
Netskills

Lists

- HTML supports 3 types of list structure
  - Unordered (bullets)
  - Ordered (numbered)
  - Definition (descriptive terms)
- Focus is on semantics/organisation not visual display
  - Lists can be fine-tuned using CSS (Cascading Style Sheets)
- Well structured lists form the basis of many dynamic menu and tab effects
  - In combination with CSS and JavaScript

Unordered and ordered lists

```
<ul>
 <li>A bullet list starts with a ul tag</li>
 <li>Each item is surrounded by li tags</li>
 <li>The list ends with a closing ul tag</li>
</ul>
```

- A bullet list starts with a ul tag
- Each item is surrounded by li tags
- The list ends with a closing ul tag

```
<ol>
 <li>A numbered list starts with an ol tag</li>
 <li>Each item is surrounded by li tags</li>
 <li>The list ends with a closing ol tag</li>
</ol>
```

1. A numbered list starts with an ol tag
2. Each item is surrounded by li tags
3. The list ends with a closing ol tag

Definition lists

```
<dl>
 <dt>List</dt>
  <dd>A collection of related points</dd>
 <dt>Lost</dt>
  <dd>In need of a map</dd>
</dl>
```

→

List
   A collection of related points
Lost
   In need of a map

- Don't be tempted to (ab)use to create indents!

HTML5: Flow elements

- Introduced to reflect a logical structure for "modern" web documents
- Notable additions, typically used at block-level:
  `<article>`
  `<section>`
  `<nav>`
  `<aside>`
  `<header>`
  `<footer>`
  `<menu>`
- Reduces the need for extra common `id` and `class` attributes (see later)

Inline Tags

Web Pages: Content

Inline tags

- Must be nested inside a block-level tag e.g.
  `<p>Some content marked up <em>using inline tags</em></p>`
- **Logical tags** supply additional semantic meaning or *rendering intent* to content
- Allowing content to be consistently interpreted
  - Different user-agents might render them differently e.g. bold text on screen or stronger tone in speech synthesizer
- Some (older) **physical tags** still exist
  - Only really produce a visual effect
  - Try and avoid them as alternatives exist
- Tags used to create links are also inline

Common logical inline tags

- Provide semantic meaning – not visual style

Tag	Enclosed content is...	Typical visual feedback
`<em>`	...given extra emphasis	Text in *italics*
`<strong>`	... given strong emphasis	Text in **bold**
`<cite>`	...a citation or reference	Usually *italicised* – detail in a `title` attribute
`<code>`	...is program code	Text in `monospace` font
`<abbr>` and `<acronym>`	...is a shortened form	None – requires a `title` attribute

Common physical inline tags

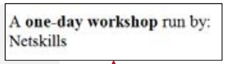
- Provide visual feedback, many *deprecated*... so use the alternative

Tag	Visual appearance	Alternative
`<b>`	Bold	`<strong>` or CSS
`<i>`	Italics	`<em>` or CSS
`<big>`	Big font size	CSS
`<small>`	Small font size	CSS
`<sub>`	$_{\text{subscript}}$Text	CSS
`<sup>`	$^{\text{superscript}}$Text	CSS
`<u>`	Underline	CSS
`<s>`	~~Strikethrough~~	CSS

Adding inline tags

```
<html>
<head>
 <title>A Simple Document</title>
 <meta http-equiv="content-type"
   content="text/html;charset=utf-8" />
</head>
<body>
 <h1>Creating Web Pages</h1>
 <p>A <strong>one-day workshop</strong> run by:
 <br />
 Netskills
 </p>
</body>
</html>
```

`<strong>` – enclosed text will be strongly emphasised. Most browsers render this as bold



A **one-day workshop** run by:
Netskills

HTML5: Phrasing elements

- Introduced to reflect a logical structure for "modern" web documents
- Notable additions typically used at inline-level:
  `<canvas>`
  `<progress>`
  `<output>`
  `<meter>`
  `<time>`
- Reduces the need for extra common `id` and `class` attributes (see later)

Links

Web Pages: Content

Making links

- (Hyper)links make the web work and follow a simple principle
  - As defined in the W3C specification…

> "A link is a connection from one Web resource to another"

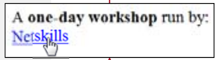
- Hyperlinks are often referred to as:
  - **Anchors** (particularly in the formal specifications)
  - Just plain old **links**
- Created in HTML using the anchor tag `<a>`
  - Usually (but not always!) with an `href` attribute

`<a href="destination">Trigger content</a>` → Trigger content

Adding a link

```
<html>
<head>
 <title>A Simple Document</title>
 <meta http-equiv="content-type"
   content="text/html;charset=utf-8" />
</head>
<body>
 <h1>Creating Web Pages</h1>
 <p>A <strong>one-day workshop</strong> run by:
 <br />
 <a href="http://www.netskills.ac.uk/">Netskills</a>
 </p>
</body>
</html>
```

`href` attribute specifies location to link to



A **one-day workshop** run by:
Netskills

Common link types

- To other websites

`href` specifies a full ***absolute*** URL to another location on the web

`<a href="http://www.netskills.ac.uk/">Netskills</a>`

- To other pages on your site

`href` specifies a file location in the same website ***relative*** to the current document

`<a href="workshops.html">Netskills Workshops</a>`

- To parts of pages

`href` points to a specific part (fragment), identified by a tag containing `id="2012"` in the destination document

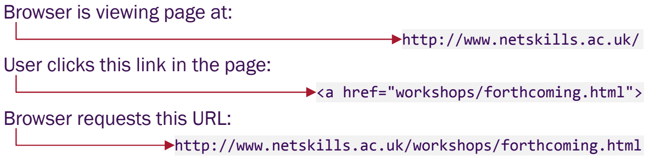
`<a href="workshops.html#2012">Netskills workshops in 2012</a>`

Linking within a web site

- Typically use **relative** links e.g.

```
<a href="forthcoming.html">
<a href="workshops/forthcoming.html">
<a href="../forthcoming.html">
```

- Browser works out full URL based on its current location i.e.



Browser is viewing page at:

`http://www.netskills.ac.uk/`

User clicks this link in the page:

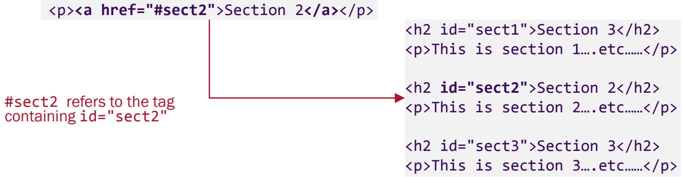
`<a href="workshops/forthcoming.html">`

Browser requests this URL:

`http://www.netskills.ac.uk/workshops/forthcoming.html`

Linking within a web page

- Link destination points to a fragment of the current document
- Identified with an `id` attribute
  - The `name` attribute was used in previous versions of HTML
  - Superseded by `id` (which can be placed in any tag)



```
<p><a href="#sect2">Section 2</a></p>
```

```
<h2 id="sect1">Section 3</h2>
<p>This is section 1….etc……</p>

<h2 id="sect2">Section 2</h2>
<p>This is section 2….etc……</p>

<h2 id="sect3">Section 3</h2>
<p>This is section 3….etc……</p>
```

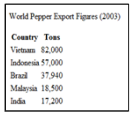
`#sect2` refers to the tag containing `id="sect2"`

Tables

Web Pages: Content

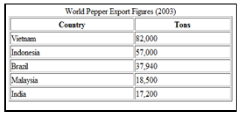
Tables

- Introduced into HTML to describe tabular data
- Have been (and still can be) used for content layout (with care)
- Tags and attributes provide basic structure and presentation
- Enhancements can be made using CSS



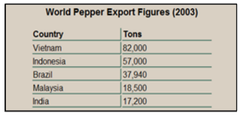
World Pepper Export Figures (2003)

Country	Tons
Vietnam	82,000
Indonesia	57,000
Brazil	37,940
Malaysia	18,500
India	17,200



World Pepper Export Figures (2003)

Country	Tons
Vietnam	82,000
Indonesia	57,000
Brazil	37,940
Malaysia	18,500
India	17,200



World Pepper Export Figures (2003)

Country	Tons
Vietnam	82,000
Indonesia	57,000
Brazil	37,940
Malaysia	18,500
India	17,200

Basic table structure

- Start with `<table>`… `</table>`
- Content inside cells `<td>` or headers `<th>`
- Cells and/or headers enclosed in rows `<tr>`

```
<p>World Pepper Export Figures (2003)</p>
<table>
  <tr>
    <th>Country</th><th>Tons</th>
  </tr>
  <tr>
    <td>Vietnam</td><td>82,000</td>
  </tr>
  <tr>
    <td>Indonesia</td><td>57,000</td>
  </tr>
</table>
```

Table attributes

Attribute	Effect	Notes
`border`	Controls outside border visibility and thickness	`0` = no gridlines `1` = gridlines `>1` = outside border only
`width`	Display width of table (default is "shrink-wrapped")	Numbers (pixels) for absolute width % for relative width
`cellspacing`	Controls distance between cells (i.e. internal gridlines)	Specify as number (of pixels)
`cellpadding`	Controls distance between cell content and cell edges	Specify as number (of pixels)

Table cell attributes

Attribute	Effect	Notes
`colspan`	Number of columns cell should span	Equiv of "merged cells" in a spreadsheet
`rowspan`	Number of rows cell should span	
`scope`	Direction in which a cell provides header information	Values are `row` or `col`
`headers`	The `id` values of the header(s) that apply to a cell	Once cell can have multiple headers
`summary`	Provides a text summary of table purpose/contents	Use for accessibility

ted

Table captions

- Use `<caption>` to associate a title with a table

```
<table border="1" width="40%>
 <caption>World Pepper Export Figures (2003)</caption>
  <tr>
    <th scope="col">Country</th><th scope="col">Tons</th>
  </tr>
  <tr>
    <td>Vietnam</td><td>82,000</td>
  </tr>
  <tr>
    <td>Indonesia</td><td>57,000</td>
  </tr>
</table>
```

- Other markup to group columns, rows, header and footer rows for fixed scrolling etc.

http://www.w3.org/TR/html4/struct/tables.html

Structural Grouping

Web Pages: Content

Structural grouping & association

- Why?
  - To group or associate related parts of the document together
  - To identify unique parts of the document
- How?
  - At document level, `<div>` and `<span>` tags used to enclose content
  - At tag level `class` and `id` attributes used to identify and/or group elements
- No associated visual styles/feedback
- Create an underlying framework for presentation and interactivity provided by CSS and JavaScript

Document-level grouping

- Block-level tags can be grouped together using `<div>`
  - Creates *page* ***div****isions*
  - Contents of a <div> can be treated as a single block

```
<div>
<h1>Section One</h1>
<p>The first bit of…</p>
</div>
```

- Inside blocks `<span>` used to select inline fragments of content
  - e.g. Lines of text, or even single characters

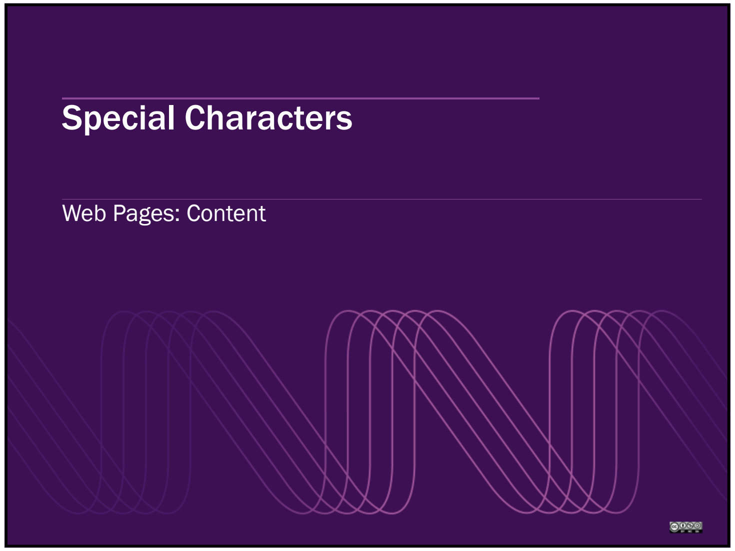
```
<p>The <span>first bit</span> of…</p>
```

- Use `class` and/or `id` attributes to attach style/script

Tag-level association

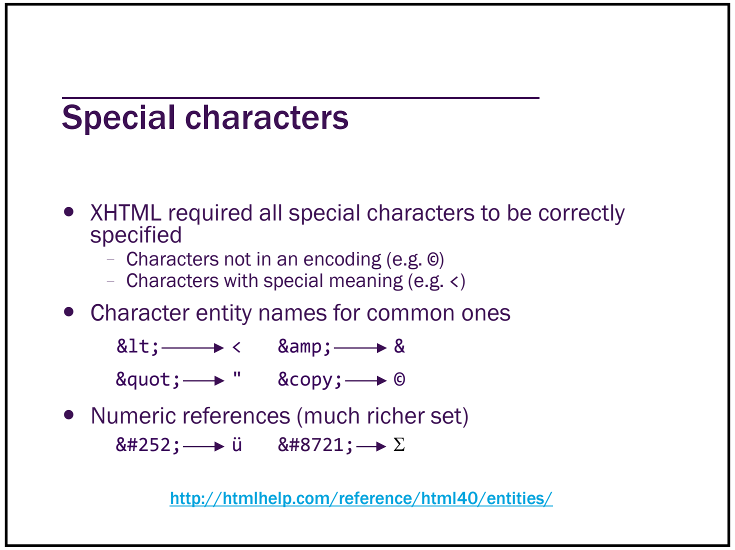
- Uses `class` and `id` attributes
- Tags can be grouped together by adding a `class` attribute
  - Tags with the same value for `class` can be associated together
  - A tag can be in multiple classes
  - Any individual class value can be reused anywhere within a page
- Any tag can be uniquely identified within a page by assigning it a specific `id` attribute
  - A tag can only have one `id` attribute
  - An individual id attribute value can only be used *once within a page*

```
<p class="flowers">Roses are red</p>

<p class="flowers" id="violets">Violets are blue</p>
```



Special Characters

Web Pages: Content



Special characters

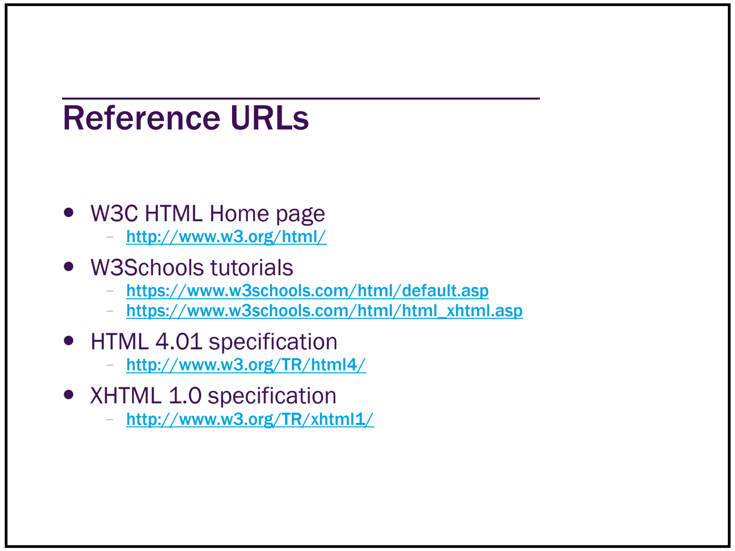
- XHTML required all special characters to be correctly specified
  - Characters not in an encoding (e.g. ©)
  - Characters with special meaning (e.g. <)
- Character entity names for common ones

```
&lt; → <      &amp; → &
&quot; → "    &copy; → ©
```

- Numeric references (much richer set)

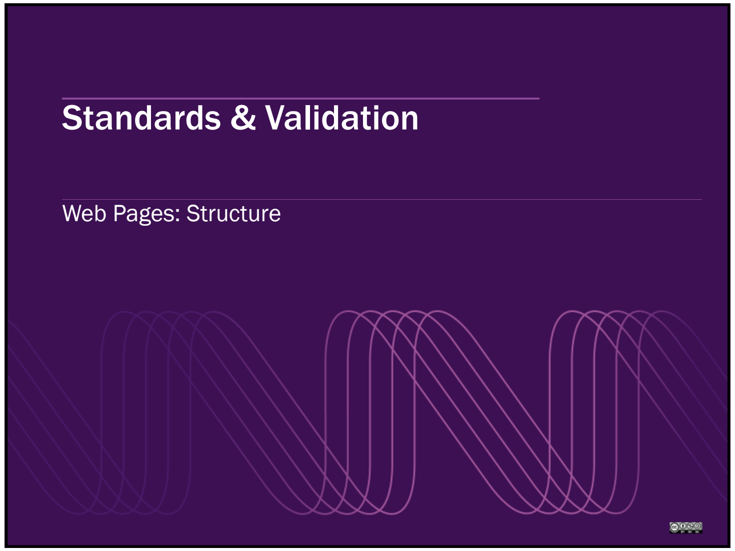
```
&#252; → ü    &#8721; → Σ
```

http://htmlhelp.com/reference/html40/entities/



Reference URLs

- W3C HTML Home page
  - http://www.w3.org/html/
- W3Schools tutorials
  - https://www.w3schools.com/html/default.asp
  - https://www.w3schools.com/html/html_xhtml.asp
- HTML 4.01 specification
  - http://www.w3.org/TR/html4/
- XHTML 1.0 specification
  - http://www.w3.org/TR/xhtml1/



Standards & Validation

Web Pages: Structure

What are web standards?

> *"Technologies [used] for creating and Interpreting web-based content [designed and developed to] ensure long-term viability of web publishing"*

http://www.webstandards.org/about/mission/

- Evolved with the help of (and in spite of) major software/hardware manufacturers
- Often thought (wrongly) to be solely connected with W3C
- Always important but now practical and usable
- Often part of legal requirements

Why are standards important?

- Historical expansion of new technology
  - Web technologies made publicly and freely available
  - Browsers supporting core standards plus proprietary extras
  - Users with wrong browser excluded/several versions of content needed
- Fixing software is not the only problem
- Web developers also hold the key
  - Frustrated or ignorant or both?
  - "Designers" who don't understand the web
  - Implications of third party content creation solutions
- Consistent application of standards can help solve these problems

(X)HTML, SGML and DTD

- HTML (and XHTML) ultimately derived from **SGML**
  - **Standard Generalized Markup Language**
  - An ISO standard meta-language for defining markup languages
- SGML markup languages all have a **DTD**
  - **Document Type Definition**
  - Contains the formal description of the language
  - Defines permitted tags, attributes and content types
- Marked up documents should contain a **DTD declaration**
- The W3C (World Wide Web Consortium) publishes DTDs for each version of (X)HTML
  - Along with recommendations as to how user-agents (e.g. web browsers) should interpret and render the marked up content

(X)HTML DTDs

- All versions of HTML/XHTML have at least two associated DTDs
- **Strict**
  - Contains only the current, valid tags and attributes
- **Transitional** (or loose)
  - Retains ***deprecated*** definitions for some older markup to allow backwards compatibility
- Always try to write to the **strict** specification wherever possible
  - HTML 4.01 Strict or XHTML 1.0 Strict are both good ☺

DTD declarations

- HTML 4.01 Strict

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
```

Transitional

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
```

- XHTML 1.0 Strict

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0//EN"
  ""http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"">
```

Transitional

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

DTD snippet

- A DTD is a text file
  - Designed to be machine-read
  - Used by a validator to check a document

```
<!-- multi-line text field -->
<!ELEMENT textarea (#PCDATA)>
<!ATTLIST textarea
  %attrs;
  %focus;
  name        CDATA         #IMPLIED
  rows        %Number;      #REQUIRED
  cols        %Number;      #REQUIRED
  disabled    (disabled)    #IMPLIED
  readonly    (readonly)    #IMPLIED
  onselect    %Script;      #IMPLIED
  onchange    %Script;      #IMPLIED
  >
```

Definition for a **<textarea>** tag

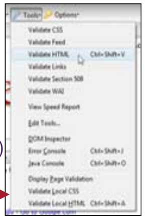
List of permitted attributes for **<textarea>**

Deprecated tags and attributes

- A number of tags and attributes were **deprecated** with the introduction of HTML 4.01
  - Support for deprecated code could be removed from browsers (theoretically at any time)
  - Developers therefore encouraged to remove it from their code
- Deprecated code is invalid for strict HTML 4.01 Strict (and therefore XHTML 1.0 Strict)
  - If it used, pages must use the Transitional DTD declaration
- Most deprecations are associated with style and layout e.g. `<font>…</font>`tags, `align` attributes etc.
  - Deprecated code will have a more up-to-date alternative, usually achieved with CSS (Cascading Style Sheets)

XHTML validation

- Validation parses source document and compares against a DTD
- Lots of tools available
  - Online/offline, standalone/integrated into other systems
- Get used to using the W3C Validation Service
  - Access directly at http://validator.w3.org
- Good browser tools now exist too e.g.
  - Firebug (Firefox extension)
  - Chrome and IE9 (built in)
  - Web Developer Toolbar (Firefox & Chrome extension)



Why validate pages?

- Browsers do not validate markup
  - A DTD might encourage the browser to use a "standards-compliant" rendering mode
  - Otherwise think "tag soup"
- Interoperability and standards compliance is a good thing…
  - Scalability
  - Accessibility
  - Extendibility
  - Manageability
- You should always make sure your code is valid

Who sets the standards?

- The World Wide Web Consortium (W3C)
  - Established in 1994, manages many core web technical specifications
  - Focus on consistency to allow long-term viability of web

  http://www.w3.org/Consortium/facts#history

  http://www.w3.org/Consortium/mission
- Now over 450 members including:
  - Microsoft, Apple, Google, Yahoo!, IBM, BBC, BT…
  - Mozilla Foundation, Apache foundation, Walt Disney Internet Group…

  http://www.w3.org/Consortium/Member/List
- The W3C do not define **all** the standards though…

(Some) W3C standards

- Core web content standards
  - HTML/XHTML/HTML5
  - CSS
  - XML
  - DOM
  - PNG
- Web architecture standards
  - HTTP
  - Identifiers (URI/URLs etc.)
- …and many more!

http://www.w3.org/standards/about.html

(Some) other standards

- Not all web technology standards are managed by the W3C
- ECMA-262
  - The technical standard for JavaScript
- ISO 8879:1986… SGML
  - An ISO standard for making markup languages
- TCP/IP
  - Describes the interconnection of the protocols which make the internet work
  - Managed by Internet Engineering Task Force (IETF)

http://www.webstandards.org/learn/external/orgs/

Reference URLs

- W3C HTML home page, specifications and FAQ
  - http://www.w3.org/html/
  - http://www.w3.org/TR/html4/
  - http://www.w3.org/TR/xhtml1/
  - http://www.w3.org/MarkUp/2004/xhtml-faq
- Web Standards Project (WaSP)
  - http://www.webstandards.org/
  - http://www.webstandards.org/learn/tutorials/common_ideas/
- W3C Validation service
  - http://validator.w3.org/

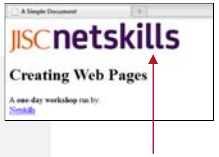
Images and Multimedia

Web Pages: Structure

Basic principles

- Any page content that isn't text requires the browser to be capable of handling it
- Some content will have *native* support
  - Built in to browser – no extra software needed
  - Core image formats supported by all browsers
  - Increasingly HTML5 allow native support for video/audio too
- Others will require plug-in/helper applications
  - Heavier duty work and proprietary objects
  - Flash player, Java VM, Media player, QuickTime etc
- A browser may include some plug-ins by default
  - Google Chrome includes Flash player and a PDF reader

Adding images to a web page


A Simple Document

JISC netskills

Creating Web Pages

A one-day workshop run by:
Netskills

```
<html>
<head>
 <title>A Simple Document</title>
 <meta http-equiv="content-type"
   content="text/html;charset=utf-8" />
</head>
<body>
 <div>
  <img src="netskills.png" alt="JISC Netskills" />
 </div>
 <h1>Creating Web Pages</h1>
 <p>A <strong>one-day workshop</strong> run by:
 <br />
 <a href="http://www.netskills.ac.uk/">Netskills</a>
 </p>
</body>
</html>
```

`src` attribute specifies location of image file

Image attributes

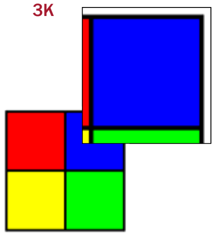
Attribute	Purpose	Notes
`src`	Specifies a URL location (relative or absolute) from which an image file can be retrieved	Images can be located anywhere the browser can access
`alt`	A meaningful description of an image to be used by screen-readers (or if image fails to load)	An `alt` attribute required by specification. Can be empty `alt=""` for purely decorative images
`width` & `height`	Pixel dimensions for the space the browser should allow to display the image	Changing these **does not** alter the size and shape of the image. Use an image editor to resize images
`title`	Provides a descriptive tag for the image	The text in a **title** attribute is usually rendered as a tool-tip when the mouse hovers over the image

Image file types

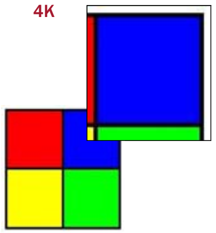
- GIF(`.gif`)
  - 256 colour palette – good for solid graphics
  - Can have transparent background
  - Can be used for simple animation
  - Loss-less compression
- JPEG (`.jpg`)
  - 16 million colour palette – good for photos
  - Lossy compression - take care
  - Careful use = small file size + high quality
- PNG(`.png`)
  - Open-source format with loss-less compression
  - Conceived as an improvement on/replacement for GIF
  - Handles higher resolution images, transparency etc.

GIF for block colours

Lossless GIF compression, plus small pallet, gives clean blocks of colour

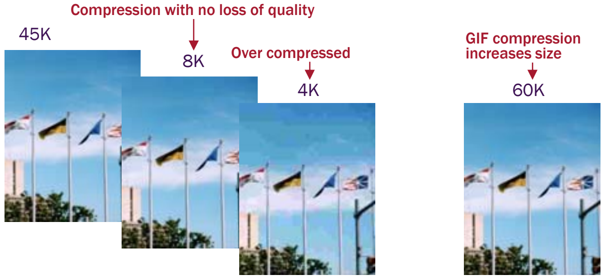


Lossy JPEG compression, introduces blur (and can make file bigger)

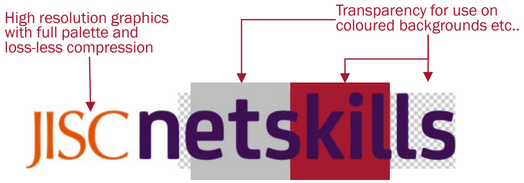


JPEG For photographic images

- JPEG produces high images at comparatively small file sizes


Compression with no loss of quality
45K
8K
Over compressed
4K
GIF compression increases size
60K

PNG – best of both worlds?



High resolution graphics with full palette and loss-less compression

Transparency for use on coloured backgrounds etc..

- PNG much better for illustrations and complex graphics
- Photographic images with complex colours usually better (and smaller) with carefully created JPEGs

http://www.turnkeylinux.org/blog/png-vs-jpg

Other media objects

- Web documents can contain more than just images
  - Audio content – sound files, streamed music etc.
  - Video content – movies, steamed live footage etc.
  - Embedded applications – Flash and Java apps etc.
- Most cannot be played natively in the browser and require O/S support and/or a specific plug-in
- Common formats
  - Java, Adobe Flash, Apple QuickTime, Windows Media, MP3 audio etc

Adding media objects

- Historically, most media formats were proprietary
  - Support depended on browser buy-in and licensing
  - Has taken a long time to approach anything like a "standard" for including media objects in web pages
- Originally proprietary tags added by browser manufacturers
- `<applet>` (Java applets) – became part of HTML spec
- `<embed>` (Netscape) – widely supported but **never** adopted in any standard HTML version
- `<object>` and `<param> are used by` HTML 4.01 and XHTML 1.0
- HTML 5 has other options

Using <embed>

- The most widely used method
  - Mostly due to support for older browsers (which don't exist now!)

```
<embed src="http://blip.tv/play/AYGJ7iIA"
       type="application/x-shockwave-flash"
       width="525" height="424"
       allowscriptaccess="always"
       allowfullscreen="true">
</embed>
```

- Still used by major content services
  - Newer browsers still support it
  - YouTube (and others) are phasing it out

Using <object>

- Introduced by the W3C in HTML 4.01 as a generic way to include any external object in a page
- Supported by all current browsers, with some *slight* differences

type – required by Internet Explorer

data – required by Firefox

```
<object width="525" height="424"
        type="application/x-shockwave-flash"
        data="http://blip.tv/play/AYGJ7iIA">
 <param name="movie" value="http://blip.tv/play/AYGJ7iIA" />
 <param name="allowFullScreen" value="true" />
 <param name="allowscriptaccess" value="always" />
</object>
```

HTML 5?

- The specification for HTML 5 includes improved support for media objects
- New logical tags e.g. `<video>` and `<audio>`
- Expectations that browsers will support playback of some formats without needing additional plugins
- Issues with choice of default video format
  - Focus on widely used (proprietary) **h.264** video format for MP4
  - Google Chrome announced a future removal support for h.264 (to focus on their own format)
  - Mozilla didn't want to include h.264 as it needs to be licensed
  - Most recent browsers support h.264

http://youtube-global.blogspot.com/2010/01/introducing-youtube-html5-supported.html
https://caniuse.com/#search=video%20format

An HTML 5 example

- The open source OGG Theora (currently) works in Firefox, Opera, Chrome & Edge

https://www.w3schools.com/html/html5_video.asp

```
<video width="640" height="480"
       src="microblog.ogg"
       type="video/ogg"
       controls="controls">
</video>
```

An HTML 5 capable browser will add controls and play the video.

HTML5 also includes...

`<iframe>`

... and...

`<embed>`

😐

Reference URLs

- W3Schools tutorials
  - https://www.w3schools.com/html/html_media.asp
- HTML 5 specification
  - http://www.w3.org/TR/html5/
- HTML 5 developments
  - http://en.wikipedia.org/wiki/HTML5
  - http://en.wikipedia.org/wiki/HTML5_video

CSS Essentials

Web Pages: Presentation

Evolution of HTML formatting

- (X)HTML only for structuring content
  - Specification only contains *guidelines* for visual browsers
- Some tags/attributes added for visual formatting

```
<font face="Arial" color="red">Hello</font>
```

→ Hello

- This mixes style and structure
  - Often using proprietary mark up with limitations on what can be applied

The Solution: CSS

- Cascading Style Sheets
  - Separation of style from structure
  - Control – potentially over every item in the page
  - Easier style management
- Strict XHTML & Strict HTML 4.01 both *deprecated* HTML formatting in favour of CSS

Same content… different view

http://www.csszengarden.com

http://www.mezzoblue.com/zengarden/alldesigns/

Why style?

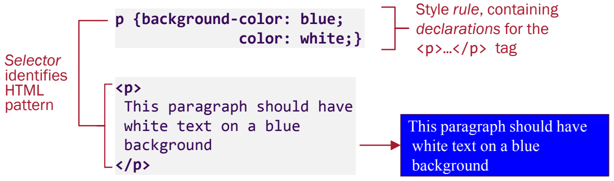
- Plain web pages are dull!
  - Additional meaning and aesthetics enhance (and influence) user experience
- An opportunity for creative expression
- Need to balance signal (information & purpose) with noise (distraction) where…

  Absence of style == monotone signal

  "Overstyled" == increased noise

CSS style sheets

- Style sheets specify formatting *rules*
- Rules consist of *selectors* and *declarations*



```
p {background-color: blue;
             color: white;}
```

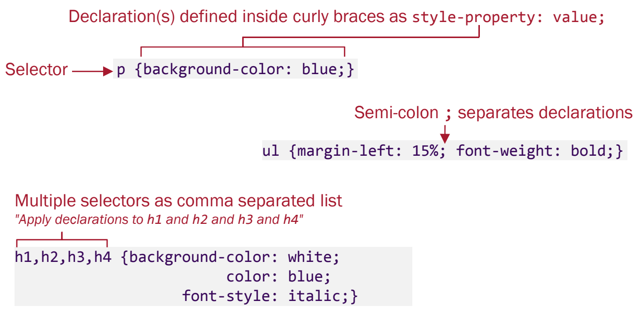
Style *rule*, containing *declarations* for the `<p>…</p>` tag

Selector identifies HTML pattern

```
<p>
 This paragraph should have
 white text on a blue
 background
</p>
```

This paragraph should have white text on a blue background

Basic style sheet syntax



Declaration(s) defined inside curly braces as `style-property: value;`

Selector

```
p {background-color: blue;}
```

Semi-colon `;` separates declarations

```
ul {margin-left: 15%; font-weight: bold;}
```

Multiple selectors as comma separated list

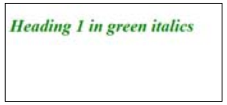
"Apply declarations to h1 and h2 and h3 and h4"

```
h1,h2,h3,h4 {background-color: white;
                        color: blue;
                   font-style: italic;}
```

Internal style sheets

- Rules set out in `<style>` tags in the `<head>` section of the page

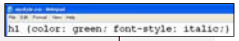
```
<html>
<head>
 <title>Internal Example</title>
 <style type="text/css">
  h1 {color: green; font-style: italic;}
 </style>
</head>
<body>
 <h1>Heading 1 in green italics</h1>
</body>
</html>
```



Heading 1 in green italics

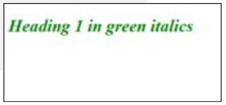
External style sheets

- Style sheets are stored in separate files
  - Linked to current document
  - Multiple style sheets can be linked to a single page



```
h1 {color: green; font-style: italic;}
```

```
<html>
<head>
 <title>CSS example</title>
 <link rel="stylesheet" type="text/css" href="mystyle.css" />
</head>
<body>
 <h1>Heading 1 in green italics</h1>
</body>
</html>
```



Heading 1 in green italics

Using @import rules

- Alternative way to include external style sheets

```
<style type="text/css">
 @import url("styles.css");
</style>
```

- No difference in effect or behaviour, but can be more convenient
  - Only need one hard-coded `<link>` in XHTML document
  - Style sheets can be edited/attached/renamed without touching XHTML document

```
<link rel="stylesheet" type="text/css" href="styles.css" />
```

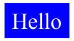
```
@import url("default.css");
@import url("navbar.css");
@import url("print.css");
```

Single linked style sheet used to import actual styles from separate files

Inline styles

- Style can also be added *inline*
  - Uses `style` attribute with CSS rule(s) as value

```
<p style="color:white; background-color: blue;">Hello</p>
```



Hello

- Try and avoid if possible – mixes style and structure back up
- Can be a useful option if needed to overcome a *specificity* issue

More on CSS selectors

- Three basic selector types define patterns to find in the mark-up

  Tag – match all instances of the tag e.g. every `<p>…</p>`

  Class – match tags containing this `class` attribute

  Id – match the unique tag containing this `id` attribute

- Can be combined for more *specific* matches
- Additional syntax and operators allow precise control
- Combine with `<div>` and `<span>` to build a *framework* for display

Classes as selectors

- Used to apply styles to specific sub-sets of HTML tags
  - Tags are grouped using a `class` attribute
  - Tags can be in more than one class

```
<h1 class="special">A heading</h1>
<p>This is a normal paragraph</p>
<p class="special">A different class of paragraph</p>
```

- Define style rule(s) in the style sheet

```
p {text-align: left; color: red;}
.special {text-transform: uppercase;}
p.special {text-align: right; color: green;}
h1.special {text-decoration: underline;}
```

Dot (.) in selector pattern indicates a class e.g. `p.special` matches `<p class="special">`

ID as a selector

- Used to identify *unique* elements in the page
  - Uses an `id` attribute in the tag
  - Each `id` value can only be used *once* in any page (same `id` can be used on multiple pages though)

```
<p>The <span id="oneoff">Important</span> bit of…</p>
```

`oneoff` now provides a unique id for a single element in this document

- Hash (#) in the CSS selector pattern indicates an `id`

```
#oneoff {font-style: italic; font-weight: bold;}
```

The ***Important*** bit of….

More selector syntax

Selector	Pattern matched
`p`	All `<p>`
`.special`	`<anytag class="special">`
`p.special`	All `<p class="special">`
`#thisBox`	The only `<anytag id="thisBox">`
`#thisBox p`	All `<p>` nested *anywhere* inside the only `<anytag id="thisBox">`
`#thisBox > p`	All `<p>` that are *direct children* of `<anytag id="thisBox">`
`#thisBox p.special`	All `<p class="special">` nested *anywhere* inside the *only* tag with the `id` of `thisBox`
`div#thisBox p`	All `<p>` nested *anywhere* inside the *only* `<div id="thisBox">`

http://www.w3.org/TR/CSS2/selector.html

Combining selectors

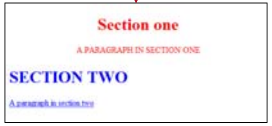
CSS Rules

```
#section1 {color:red; text-align:center;}
#section2 {color:blue;}
.caps {text-transform:uppercase;}
#section2 p {text-decoration:underline;}
```

HTML

```
<div id="section1">
 <h1>Section one</h1>
 <p class="caps">A paragraph in
   section one</p>
</div>
<div id="section2">
 <h1 class="caps">Section two</h1>
 <p>A paragraph in section two</p>
</div>
```

`#section2 p` styles only applied to `<p>…</p>` nested inside `#section2`



Section one

A PARAGRAPH IN SECTION ONE

SECTION TWO

A paragraph in section two

Cascading style sheets

- All available styles for a page are combined as it loads
  - Final appearance for each element is composite of all appropriate rules
- Conflicting property values resolved by simple rules
  1. **Source:** User-specified styles (in the browser) are more specific
  2. **Specificity:** Relative weighting of selector priority
  3. **Order declared:** If specificity value are the same then "last one wins" (means inline styles are always more specific)
- Specificity – a measure if importance
  - The more *specific* the rule is… the greater priority its declarations have
  - Easy to calculate…

Specificity calculator

Count the number of ID, class and tag names in each selector

Selector	IDs	Classes	Tags
`#thisbox`	1	0	0
`.special`	0	1	0
`p`	0	0	1
`p.special`	0	1	1
`#thisbox p`	1	0	1
`#thisBox > p`	1	0	1
`div#thisBox p`	1	0	2
`#thisBox p.special`	1	1	1

Basic values of:
ID = 100
Class = 10
Tag = 1

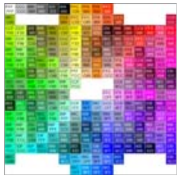
More *specific* combinations have higher values

CSS Units

- CSS supports many types of measurement unit
- **Absolute** units calculated independently of other page content and/or browser defaults
  - Useful for precise layout
  - Include Pixels (`px`), Points (`pt`), Millimetres (`mm`)
- **Relative** units calculated proportionally against other page content or a browser default
  - e.g. currently available width, default text size etc.
  - Include Percentages (`%`), Ems (`em`), Exes (`ex`)
  - Also special relative units for text e.g. `small`, `large`, `x-large` … etc.
- Good design uses a combination of both

CSS colours

- CSS allows rich control over colo(u)r
  - *Any* colour can be specified using RGB or Hex (hexadecimal) codes
  - Only 17 *names* are actually valid in CSS2



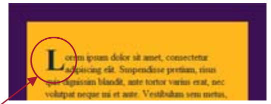
```
color: red;              ✓
color: magenta;          ✗
color: rgb(0,32,234);    ✓
color: #0000ff;          ✓
```

http://www.w3.org/MarkUp/Guide/Style.html http://colorschemedesigner.com/

Pseudo elements

Selectors for special parts of some elements

```
p.opening:first-letter {
        font-size: 300%;
        font-weight: bold;
        float: left; }
```



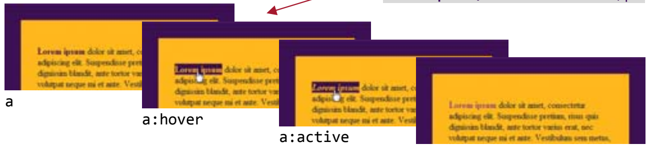
```
<p class="opening">Lorem ipsum dolor sit amet,
consectetur adipiscing elit. Suspendisse pretium,
risus quis dignissim blandit, ante tortor... etc
</p>
```

Pseudo classes

Selectors for special status of some elements

```
a          {text-decoration: none; font-weight: bold; color: #3c1053;}
a:hover    {background-color: #3c1053; color: #ffb81c;}
a:active   {font-style: italic;}
a:visited  {color: #8c4779;}
```

```
<p>
<a href="http://lipsum.com">
Lorem ipsum</a> dolor... etc </p>
```



a

a:hover

a:active

a:visited

The CSS box model

Web Pages: Presentation

The CSS box model

- Fundamental to CSS layout
  - Every page element represented as a box
  - Box properties not inherited from parent boxes
- Box properties can apply to whole box or individually to any of the 4 sides
  - Shorthand declarations make this easy
  - Depends on property

Border

- The outline of a box, made up of three sub-properties

  Width – a unit of measurement e.g.

  `border-width: 20px;`

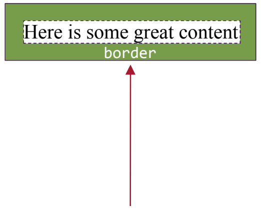
  Style –value from a preset list e.g.

  `border-style: solid|dashed|…`

  Color – a valid colour value e.g.

  `border-color: blue;`

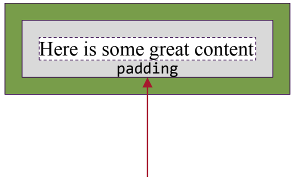
- Simple shorthand to set quickly



```
p {border: 20px solid green;} → <p>Here is some great content</p>
```

Padding

- The space between the *outside* edge of the content and the *inside* edge of its box
  - i.e. excluding any borders
- Creates space *inside* boxes
  - Leave empty (negative space)
  - Use background images to fill

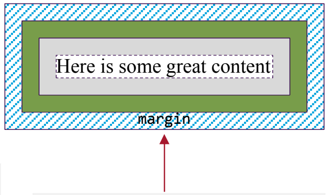


```
p {border: 20px solid green;
   padding: 20px;
}
```

→ `<p>Here is some great content</p>`

Margin

- The space around the outside edge of a box – including borders
- Creates space *between* boxes
  - Useful for spacing content



```
p {border: 20px solid green;
   padding: 20px;
   margin: 20px;
}
```

→ `<p>Here is some great content</p>`

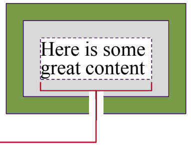
Shorthand declarations

- Useful for setting box properties quickly and/or precisely
  - Use wherever possible to reduce the amount of code needed to style a page

Declaration	Result
`border-width: 20px;`	Width `20px` on all 4 sides
`border-left-width: 20px;`	Width `20px` on left-hand side only
`margin: 20px 40px;`	Margin `20px` top/bottom, `40px` left/right
`padding: 20px 40px 10px 30px;`	Padding set for top/right/bottom/left
`border: 20px solid blue;`	`20px`, `solid`, `blue` border on all 4 sides

Width and Height

- By default, browser uses the max width available for each box
- Box property `width` used to impose a defined value for *content* width



```
#box1 {width: 200px;}
```

Total width (i.e. as drawn) = `width` (+ 2x `padding` + 2x `border`)

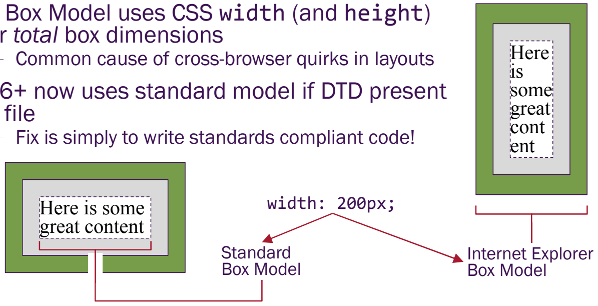
- Modern browsers also support more flexible min/max

```
#box1 {min-width: 200px; max-width: 800px;}
```

- Height can be set in the same way

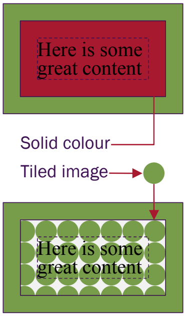
The Internet Explorer box model

- IE Box Model uses CSS `width` (and `height`) for *total* box dimensions
  - Common cause of cross-browser quirks in layouts
- IE6+ now uses standard model if DTD present in file
  - Fix is simply to write standards compliant code!


Here is some great content
Here is some great cont ent
width: 200px;
Standard Box Model
Internet Explorer Box Model

Background

- Background of all the visible space *inside* any borders i.e. (content + padding)
- Made up of several sub-properties
  - **Color** – a valid colour value (or `transparent`)
    `background-color: red;`
  - **Image** – Use an image file as box background
    `background-image: url(/images/bg1.png);`
  - **Position** – Control placement for initial image tile
    `background-position: top right;`
  - **Repeat** – Control direction of image tiling
    `background-repeat: repeat-x|y|no-repeat;`



Using background images

- Form the basis of most modern web designs
- Easy to work with once you have grasped the basic box model (and got the graphics!)
- Two important things to remember…

1. CSS2 only allows one image per box (CSS3 allows multiple images ☺)
2. Images delivered by CSS are a decorative part of the design **not** the content
   - Image *content* e.g. photos, diagrams etc. should be part of the XHTML (via the `<img />` tag)

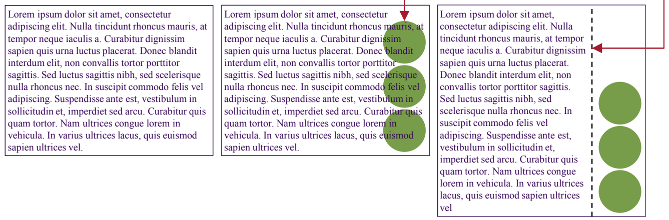
Placing a background image

```
background-image: url(bg1.png);
background-position: bottom right;
background-repeat: no-repeat;
```

Inserts a single instance of `bg1.png`

Create space for the image using `padding` – set to at least the image width

```
padding-right: 200px;
```



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla tincidunt rhoncus mauris, at tempor neque iaculis a. Curabitur dignissim sapien quis urna luctus placerat. Donec blandit interdum elit, non convallis tortor porttitor sagittis. Sed luctus sagittis nibh, sed scelerisque nulla rhoncus nec. In suscipit commodo felis vel adipiscing. Suspendisse ante est, vestibulum in sollicitudin et, imperdiet sed arcu. Curabitur quis quam tortor. Nam ultrices congue lorem in vehicula. In varius ultrices lacus, quis euismod sapien ultrices vel.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla tincidunt rhoncus mauris, at tempor neque iaculis a. Curabitur dignissim sapien quis urna luctus placerat. Donec blandit interdum elit, non convallis tortor porttitor sagittis. Sed luctus sagittis nibh, sed scelerisque nulla rhoncus nec. In suscipit commodo felis vel adipiscing. Suspendisse ante est, vestibulum in sollicitudin et, imperdiet sed arcu. Curabitur quis quam tortor. Nam ultrices congue lorem in vehicula. In varius ultrices lacus, quis euismod sapien ultrices vel.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla tincidunt rhoncus mauris, at tempor neque iaculis a. Curabitur dignissim sapien quis urna luctus placerat. Donec blandit interdum elit, non convallis tortor porttitor sagittis. Sed luctus sagittis nibh, sed scelerisque nulla rhoncus nec. In suscipit commodo felis vel adipiscing. Suspendisse ante est, vestibulum in sollicitudin et, imperdiet sed arcu. Curabitur quis quam tortor. Nam ultrices congue lorem in vehicula. In varius ultrices lacus, quis euismod sapien ultrices vel

Content > Box Size = Overflow

- Use box's `overflow` property to control

Shrinking a box creates an *overflow*

Allow overflow to spill out of box (default):

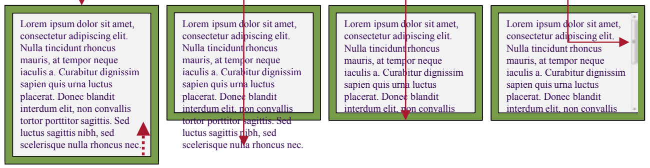
`overflow:visible;`

Overflow can be hidden (clipped) with:

`overflow:hidden;`

Add scrollbar to see clipped content with:

```
overflow:scroll;
or
overflow:auto;
```



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla tincidunt rhoncus mauris, at tempor neque iaculis a. Curabitur dignissim sapien quis urna luctus placerat. Donec blandit interdum elit, non convallis tortor porttitor sagittis. Sed luctus sagittis nibh, sed scelerisque nulla rhoncus nec.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla tincidunt rhoncus mauris, at tempor neque iaculis a. Curabitur dignissim sapien quis urna luctus placerat. Donec blandit interdum elit, non convallis tortor porttitor sagittis. Sed luctus sagittis nibh, sed scelerisque nulla rhoncus nec.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla tincidunt rhoncus mauris, at tempor neque iaculis a. Curabitur dignissim sapien quis urna luctus placerat. Donec blandit interdum elit, non convallis

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla tincidunt rhoncus mauris, at tempor neque iaculis a. Curabitur dignissim sapien quis urna luctus placerat. Donec blandit interdum elit, non convallis

Display and Visibility

- CSS `display` property controls the display *type* of an element
  - Can be used to override HTML default or even remove items from the styled page flow

```
display: block;
display: inline;
display: none;
```

- CSS `visibility` does exactly what it says on the tin!
  - Show/hide elements whilst leaving them in the page flow

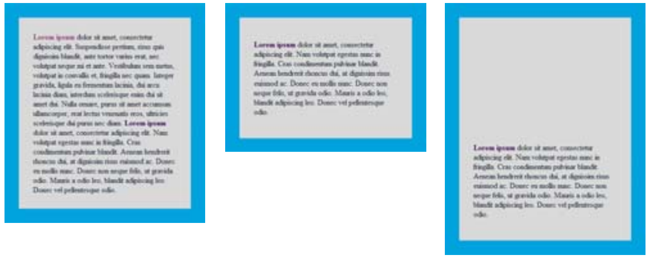
```
visibility: hidden;
visibility: visible;
visibility: collapse; (tables only)
```

Display and Visibility

```
p {display: inline;}
```

```
p.opening {
  display: none;}
```

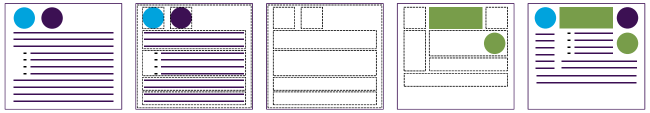
```
p.opening {
  visibility: hidden;}
```



Positioning possibilities

- Once content can be viewed as "just boxes", CSS positioning and layout simply involves moving, placing or rearranging them

Structured content provides the basis for box model framework → Consider page as just the boxes → Manipulate boxes (position, background, padding etc.) to achieve final layout



- Specific box properties control how positioning takes place
  - `position`, `float`, `top`, `left`, etc…
  - These are considered in more detail in *CSS Positioning*

CSS Positioning

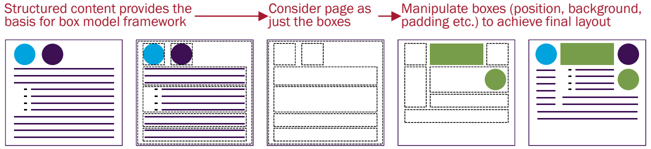
Web Pages: Presentation

First… clean (X)HTML

- Good CSS layout relies on good (X)HTML
- Positioning is easier to manage in a well structured document
  - Important to know which elements are contained within which others
  - Good use of `<div>`, `<span>`, `class` and `id` to create additional framework
- Choosing HTML or XHTML doesn't matter
- Choosing *strict* HTML or XHTML *is* important
  - Avoids temptation to use deprecated tags/attributes
  - Encourages the use of CSS instead

Second... the CSS box model

- The first step to mastering CSS positioning and layout is to understand the CSS Box Model
  - Easier with well structured XHTML
- Once content is viewed as "just boxes", CSS positioning and layout simply involves moving, placing or rearranging them



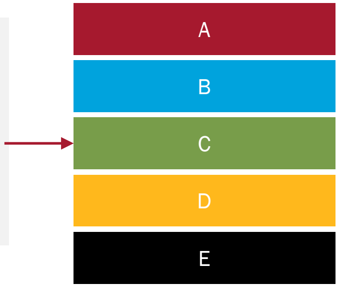
Structured content provides the basis for box model framework → Consider page as just the boxes → Manipulate boxes (position, background, padding etc.) to achieve final layout

Next... check natural page flow

- The *natural page flow* of a document is the *source order* display of the XHTML element boxes it contains
- The final rendered position of each block of content determines the starting point for the following ones
- With no CSS applied this may not look pretty – but should make logical sense to help non-visual browsers
  - e.g. screen reading text-to speech browsers

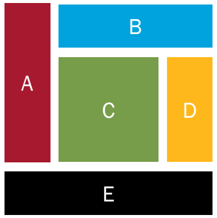
Natural page flow

```
<html>
<head>etc.</head>
<body>
<div id="A">A</div>
<div id="B">B</div>
<div id="C">C</div>
<div id="D">D</div>
<div id="E">E</div>
</body>
</html>
```



Finally...apply CSS and enjoy ☺

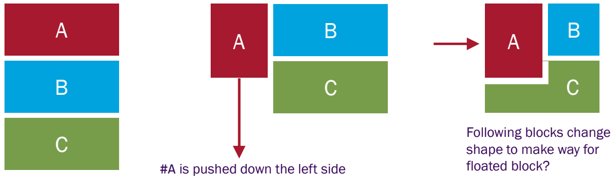
- Browser parses (reads) the XHTML and the CSS styles which apply to each box *before* drawing the page
- The final appearance is determined by combination of CSS properties and (if not explicitly set in CSS) browser defaults
- CSS can be used to re-position content...
  - within the page flow i.e. offset from natural location
  - contrary to the page flow i.e. removed from natural flow and placed elsewhere



Floats

- Potentially very neat way to position content, **but avoid** this important misconception…

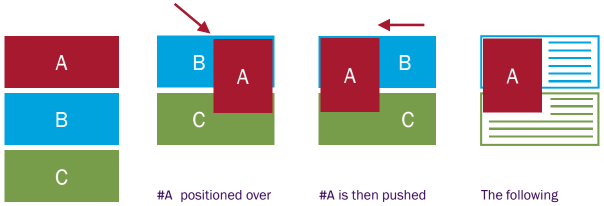
```
#A  {width: 200px;
    float: left;}
```



#A is pushed down the left side of the following content and then pushed over?

Following blocks change shape to make way for floated block?

Floats: What really happens



```
#A  {width: 200px;
    float: left;}
```

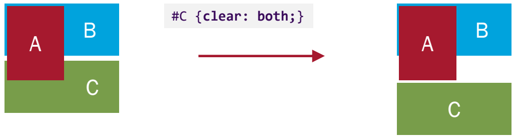
#A positioned over the following content as far over as possible *in the opposite direction to the float*

#A is then pushed back across as far as possible *in the direction of the float.*

The following blocks *do not* change shape! Content in them is allowed to wrap around the float

Clearing floats

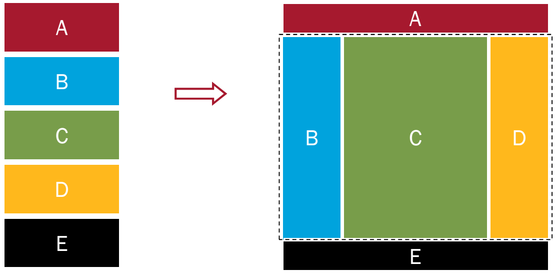
- The CSS `clear` property can be used to stop the float covering later content.
- Values `left`, `right` or `both` determine which side of the box should be "clear" of any floated elements



```
#C {clear: both;}
```

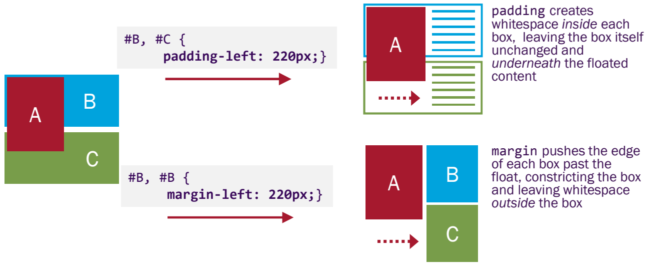
Columns

- CSS columns are (relatively) easy with floats



Floats and columns

- The non-floated boxes can be turned into columns using `margin` or `padding`



```
#B, #C {
      padding-left: 220px;}
```

`padding` creates whitespace *inside* each box, leaving the box itself unchanged and *underneath* the floated content

```
#B, #B {
      margin-left: 220px;}
```

`margin` pushes the edge of each box past the float, constricting the box and leaving whitespace *outside* the box

CSS `position` property

Four values explicitly re-position boxes using CSS

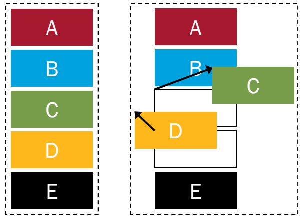
Value	Description
`static`	Default non-positioned value. Not normally set unless to specify an override of other positioning
`relative`	Leaves element in the page flow, but allows it to be displayed in an offset position
`absolute`	Removes element from page flow and allows it to be positioned anywhere
`fixed`	Removed element from page flow and fixes it to the browser viewport. Rest of page can now scroll behind it

Relative positioning

- Box is initially **positioned according to natural flow**
  - Offset is specified using *"from the..."* properties

```
#thisBox {position: relative;
          top: 25px;
          left: 100px; }
```

- Following XHTML retains original position

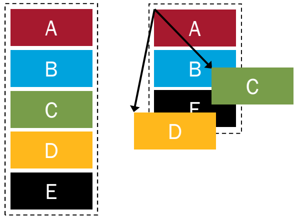


Absolute positioning

- Box *removed* from natural flow
  - New position is specified using *"from the..."* properties
  - Measured from ***nearest positioned parent* container** (default is `<body>`)

```
#thisBox {position: absolute;
          top: 25px;
          left: 100px; }
```

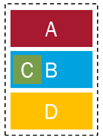
- Following XHTML behaves as if positioned block never existed!



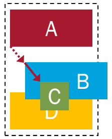
Positioned parents

- An element acting as an origin point for absolute positioning must itself *have position*
  - It does not have to have *moved* though ☺
- If no containers *with position* found, browser uses `<body>`

```
<body>
 <div id="A"></div>
  <div id="B">
   <div id="C"></div>
  </div>
<div id="D"></div>
</body>
```



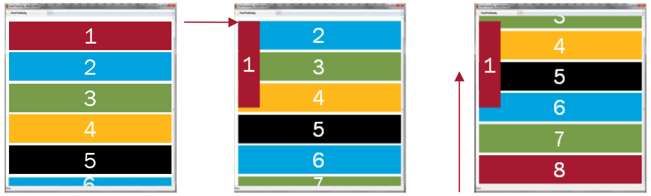
```
#B {position: relative;
    top: 50px;
    left: 100px;}

#C {position: absolute;
    top: 20px;
    left: 50px;
```



Fixed positioning

- Box *removed* from natural flow and fixed to the browser *viewport*
  - New position is specified relative to the viewport... following content scrolls underneath

```
#thisBox {position: fixed;
          top: 0px;
          left:0px; }
```



Stacking order

- Only works on content *with position*
  - Uses `z-index` property

```
#ThisBox {position:absolute;
          top: 25px;
          left: 100px;
          z-index: 1; }
```

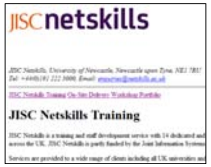
Red
Amber
Green

http://www.timkadlec.com/2008/01/detailed-look-at-stacking-in-css/

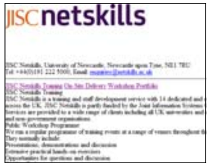
Precision layouts

You can choose to work with or against the browser defaults

- For ultimate precision some designers use a "reset style sheet"
  - Loaded first, typically resets/strips out all default style



```
body,div,ul,h1,h2,h3,p {
         margin: 0;
         padding: 0;
         border: 0;
         font-size: 100%;}
```



http://meyerweb.com/eric/tools/css/reset/

Media specific styles

Web Pages: Presentation

CSS: Media types

- CSS spec identifies a range of media types to which specific styles can be applied

Media	Intended device
`all`	Apply to all outputs (default)
`screen`	Computer screens
`print`	Printed media (and Print Preview)

CSS: Using media specific styles

- Specify via `media` attribute in `<link />`

```
<link rel="stylesheet" type="text/css" href="printer.css" media="print" />
```

- Styles only added to page when media "invoked"
  - Effect is cumulative (i.e. specificity/inheritance important!)
- Use with other media types for better control

```
<link rel="stylesheet" type="text/css" href="core.css" media="all" />
<link rel="stylesheet" type="text/css" href="layout.css" media="screen" />
<link rel="stylesheet" type="text/css" href="printer.css" media="print" />
```

CSS: Using media specific styles

- Specify inside style sheets using `@media` rules

```
@media print {
   body {background-color: #ffffff; color:#000000;}
   a {text-decoration: none; font-weight: normal; color: #000000;}
   #navbar {display: none;}
   h2 {page-break-before:always;}
}
```

CSS: Browser compatibility

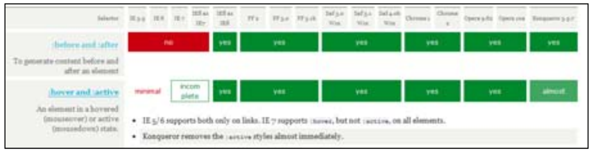
Web Pages: Presentation

CSS: Browser support

- Often seen as a big issue – mostly historical
  - Worst (mainstream) offenders Netscape 4 (ignore nowadays) & IE 4-6
- IE still seen by many as the "big problem" but…
  - IE 4 -5.5 can safely be ignored
  - IE7 & IE 8 much better
  - Main issue is slow (corporate) migration from IE 6
- Standards compliance is important for less reliance on "hacks"
- All browsers have some "quirks" – is perfection possible?

CSS: Browser support

- Tools to help
  - !Doctype switching
  - Compatibility tables
  - Clean code and good design practice!



Selector	IE 5.5	IE 6	IE 7	IE8 as IE7	IE8 as IE8	FF 2	FF 3.0	FF 3.5b	Saf 3.0 Win	Saf 3.1 Win	Saf 4.0b Win	Chrome 1	Chrome 2	Opera 9.62	Opera 10a	Konqueror 3.5.7
:before and :after To generate content before and after an element	no				yes	yes			yes			yes		yes		yes
:hover and :active An element in a hovered (mouseover) or active (mousedown) state.	minimal		incomplete		yes	yes			yes			yes		yes		almost

- IE 5/6 supports both only on links. IE 7 supports :hover, but not :active, on all elements.
- Konqueror removes the :active styles almost immediately.

http://www.quirksmode.org/css/contents.html

- er… test for yourself?

Dealing with IE

- Policies
  - Decide not to cater for IE differences?
  - Hacks – CSS tricks to hide/show specific rules to IE?
  - Conditional comments – regular HTML comments with IE specific syntax (not just CSS!)

```
<!--[if IE 6]>
<style type="text/css">
   p {property: value for IE6 only;}
</style>
<![endif]-->
```

http://www.quirksmode.org/css/condcom.html

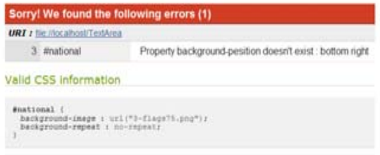
CSS: Standards and validation

Web Pages: Presentation

CSS: Validation

- CSS is error tolerant
  - Process all complete, valid rules, ignore "incorrect" rules
- W3C spec for CSS 1 & CSS 2.1 currently widely adopted (CSS3 creeping out)
- Validation service exists – use it!

http://jigsaw.w3.org/css-validator/


Sorry! We found the following errors (1)

URI : file://localhost/TextArea

3	#national	Property background-pesition doesn't exist : bottom right

Valid CSS information

```
#national {
  background-image : url("3-flags75.png");
  background-repeat : no-repeat;
}
```

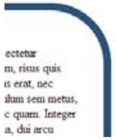
CSS3 you can use now

...BUT NOT IN ASSIGNMENT 1 !!!

Box model

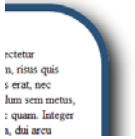
- Rounded corners with border-radius

```
#thisbox {border-radius: 10px;}
```



- Shadows with box-shadow

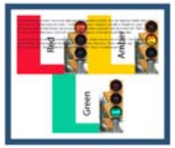
```
#thisbox {
  box-shadow: 10px 10px 40px 0px #000000;
}
```



Background enhancements

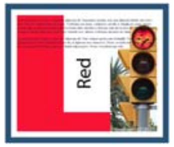
- Multiple background images

```
background-image: url('red-box.png'),
                  url('amber-box.png'),
                  url('green-box.png');
background-position: 30px 30px,
                     390px 30px,
                     210px 330px;
background-repeat: no-repeat;
```



- Background origin (includes/excludes the border)

```
background-origin: padding-box|content-box|border-box;
```



- Background sizing

```
background-size: 50% 50%;
```

Fonts

- Use the `@font-face` rule to define a font to be used in a document

```
@font-face {
    font-family: 'BLOKKNeue-Regular';
    src: url('BLOKKNeue-Regular.eot');
    src: url('BLOKKNeue-Regular.eot?#iefix') format('embedded-opentype'),
         url('BLOKKNeue-Regular.woff') format('woff'),
         url('BLOKKNeue-Regular.svg#BLOKKNeue-Regular') format('svg');
    font-weight: normal;
    font-style: normal;
}
```

http://www.css3files.com/font/

- Apply as normal

```
body { font-family: 'BLOKKNeue-Regular'; }
```

Text shadows

- If you really want to…

```
h1 {text-shadow: 1px 3px 5px gray; }
```

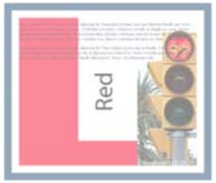
Lorem ipsum dolor

http://www.css3files.com/box-shadow/

Opacity

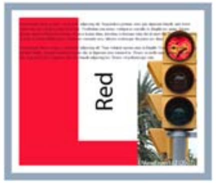
- For whole elements use `opacity`

```
#thisbox {opacity: 0.5;}
```



- For just the colour use the new `rgba` specification

```
#thisbox {border-color: rgba(43,82,119,0.5);}
```



Generated content

- Dynamically inserting content based on CSS patterns e.g.
  - text, images, counters etc.
- Uses better psuedo-element support in modern browsers

```
h1:before {content: "Banana: "}
                +
        <h1> A heading</h1>
```

= **Banana: A Heading**

http://www.westciv.com/style_master/academy/css_tutorial/advanced/generated_content.html

Media queries

- Cornerstone of "responsive design"
- Allow browser to adapt presentation based on browser dimensions

```
@media screen and (max-width: 980px) {
  /*CSS rules for viewports smaller than 980px*/
}
@media screen and (max-width: 650px) {
  /*CSS rules for viewports smaller than 650px*/
}
@media screen and (max-width: 480px) {
  /*CSS rules for viewports smaller than 480px*/
}
```

http://webdesignerwall.com/tutorials/responsive-design-with-css3-media-queries

References & look-up tables

http://caniuse.com/

http://www.css3files.com/

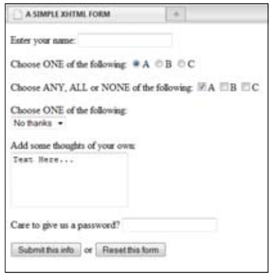
http://html5please.com/

Forms

Web Pages: Behaviour

What are web forms?

- Form tags and attributes create an interface for user interaction
- Core purpose of a form is to collect user input and return it to a server
  - Instruct browser to create one or more form *controls*
  - Provide a means to let user *submit* the form
  - Specify *how* browser should include form data in request to server
  - Specify *where* (URL) to send the input


A SIMPLE XHTML FORM

Enter your name:

Choose ONE of the following: ● A ○ B ○ C

Choose ANY, ALL or NONE of the following: ☑ A □ B □ C

Choose ONE of the following:
No thanks ▾

Add some thoughts of your own:
Text Here...

Care to give us a password?

Submit this info or Reset this form

Form structure

- Each form on page enclosed in `<form>` tags
  - wrapped around all controls for that form
- Each `<form>` tag has `method` and `action` attributes

`method` (optional) specifies how form data will be packaged and sent to server.

`action` (required) specifies the location (URL) that form data will be sent to

`<form>` tags enclose the whole form

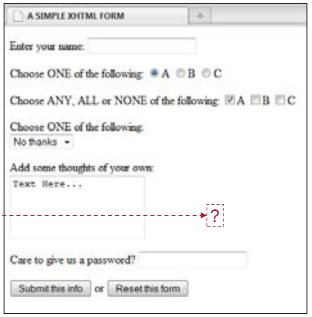
```
<body>
<form method="get" action="http://a.server.com/script">
<p>
 Enter your name: <input type="text" name="username" />
</p>
</form>
</body>
```

Form controls

- Form controls are replaced tags used to collect user input
  - The tag is ***replaced*** by the browser interface for that control type
  - W3C provide guidelines for consistent user experience
- Each form control will have a `name` attribute
- Most have an explicit `value` attribute or an implied value (based on the user action)
- On submission a ***successful*** form control will contain both a name and a value
- (Only) successful name/value pairs returned as a string to the server e.g.

```
username=Bob&age=35&email=bob@somewhere.com
```

Common form controls


A SIMPLE XHTML FORM

Enter your name:

Choose ONE of the following: ● A ○ B ○ C

Choose ANY, ALL or NONE of the following: ☑ A □ B □ C

Choose ONE of the following:
No thanks ▾

Add some thoughts of your own:
Text Here...

Care to give us a password?

Submit this info or Reset this form

One line text input →

Radio buttons →

Checkboxes →

Drop-down selections →

Multi-line line input →

Hidden fields → ?

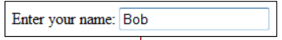
Password fields →

Submit & reset buttons →

One-line text input

- The basic (default) form control created by `<input>` tag
  - Additional attributes to control basic behaviour e.g.
  - Visible characters (width of box) with `size` e.g. `size="25"`
  - Number of characters to accept with `maxlength` e.g. `maxlength="8"`

```
<form method="get" action="http://a.server.com/script">
<p>
 Enter your name: <input type="text" name="username" />
</p>
</form>
```



Enter your name: Bob

```
username=Bob
```

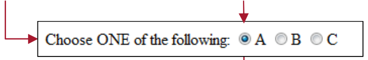
Radio buttons

- Create mutually exclusive controls – user can only select one
  - Returns fixed `name/value` – one option can (should?) be pre-selected

```
<p>Choose ONE of the following:
<input type="radio" name="choice" value="A" checked="checked" /> A
<input type="radio" name="choice" value="B" /> B
<input type="radio" name="choice" value="C" /> C
</p>
```

Same `name` attribute creates a single control

Pre-selected option



Choose ONE of the following: ● A ○ B ○ C

```
choice=A
```

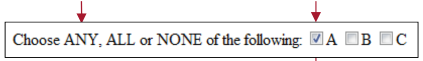
Checkboxes

- Create individual independent controls
  - *Should* return on if selected (better to supply a value though!)

```
<p>Choose ONE of the following:
<input type="checkbox" name="A" value="Yes" checked="checked" /> A
<input type="checkbox" name="B" value="Yes" /> B
<input type="checkbox" name="C" value="Yes" /> C
</p>
```

Different name attributes =independent controls

Pre-selected option



Choose ANY, ALL or NONE of the following: ☑ A □ B □ C

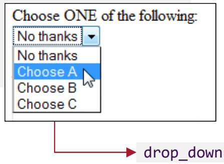
```
A=Yes
```

Drop down menus

- The `<select>` tag allows user to select from a list
  - Can pre-select a default with `selected` attribute

```
<p>Choose ONE of the following:<br />
<select name="drop_down">
  <option value="None" selected="selected">No thanks</option>
  <option value="A">Choose A</option>
  <option value="B">Choose B</option>
  <option value="C">Choose C</option>
</select>
</p>
```

Selected `<option>` returns a fixed value



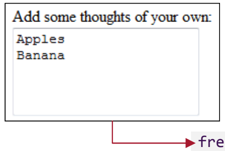
Choose ONE of the following:

```
drop_down=A
```

Multi-line input

- Collected using a `<textarea>` tag
  - Box contents (inc whitespace/newlines) returned as the `value`
  - Requires `rows` and `cols` attributes to set the size

```
<p>Add some thoughts of your own... <br />
 <textarea name="free_text" rows="4" cols="20"></textarea>
</p>
```


Add some thoughts of your own:

```
Apples
Banana
```

Initially displays 4 rows and 20 columns (chars) of text

Encoded newline characters

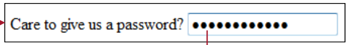
```
free_text=Apples%0D%0ABanana
```

Password fields

- Specific `type` of one-line text box which hides input from anyone looking at screen
  - Does **not** provide any other security or encryption!

```
<p>
 Care to give us a password?
 <input type="password" name="my_password" />
</p>
```

`password` type tells browser to mask input (however that browser chooses to do it)


Care to give us a password? ●●●●●●●●●●●●

```
my_password=WhateverYouTypedIn!
```

Hidden fields

- Do not appear on screen!
- Used to return fixed values alongside user input
  - Values usually pre-populated e.g. time/date stamps
  - Can be populated dynamically via client-side script e.g. returning calculated values alongside use input

```
<p>
  <input type="hidden" name="my_secret" value="hello" />
</p>
```

Both `name` and `value` must be supplied.

```
my_secret=hello
```

Form organisation

- Tags/attributes for organising forms
- Uses `<fieldset>`, `<legend>`, `<label>`
  - Plus `id` attributes
- Offers block-level structure
  - Reducing markup needed inside them e.g. `<p>`, `<div>` etc
- Provides visual cues to users
  - Logical blocks with descriptive names
- Enhanced usability/accessibility
  - Associating labels with controls
  - Increasing "hit area!" on small controls

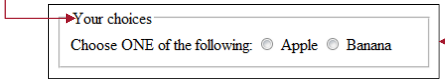
Using <fieldset> and <legend>

- Block-level organisation used to group form controls together
  - Default visual appearance can be altered using CSS

<legend> provides a label for group

```
<fieldset>
 <legend>Your choices</legend>
   Choose ONE of the following: <br />
   <input type="radio" name="choice" value="A" /> Apple
   <input type="radio" name="choice" value="B" /> Banana
</fieldset>
```

<fieldset> groups block of controls



Your choices

Choose ONE of the following: ○ Apple ○ Banana

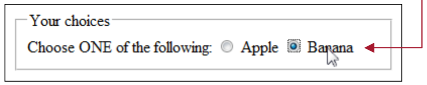
Using labels and id

- Allows text to be explicitly associated with a form control
  - Clicking on the label will activate the control

<label> can be explicitly linked to an id

```
<input type="radio" name="choice" value="A" id="A1" />
<label for="A1">Apple</label>

<label>
 <input type="radio" name="choice" value="B" /> Banana
</label>
```

<label> can also be wrapped around control & label content



Your choices

Choose ONE of the following: ○ Apple ● Banana

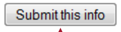
Form buttons

- 3 types of form button control
  - Set by `type` attribute in `<input>` tag
- **Submit** – most important
  - Invokes the browser's submit process and sends data to the server
- **Reset** – very useful!
  - Clears user input and returns form to its initial state ***without*** reloading the page
- **Button** – creates a generic button
  - Can *only* be used if attached to client-side scripted actions or events
- Submit and Reset buttons **do not** require any programming or additional scripting to work

Submit button

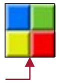
- Every good form should have one!
  - Can be a "regular" button (as drawn by the browser) or an image file
  - Images not used as often now as buttons can be styled with CSS

```
<input type="submit" value="Submit this info" />
```



`type="submit"` draws the browser's regular button `value` specifies the button text

```
<input type="image" src="button.gif" alt="submit" />
```



`type="image"` tells the browser to use an image **click** to trigger form submission `src` and `alt` specifies the image and text alternative

Submit process

- Virtually every form should have one!
- *Successful* name/value pairs are attached to a request for the URL specified by the `action` attribute in the `<form>` tag
- The `method` attribute specifies the type of request made (and therefore how the data is attached)

On submission… attach successful name/value pairs to an HTTP GET type request for this URL

```
<form method="get" action="http://a.server.com/script">
…etc…
</form>
```

The GET method

- When the user submits the form, browser makes an HTTP **GET** request for the processing URL
  - Successful form data is appended to the URL as a ***querystring***
  - The processing script will need to look in the querystring to extract data and use it

Querystring appended to URL

```
http://…/formscript.php?inputbox=Arnold&choice=A
```

- GET is the "default" method for all web requests (i.e. not actually restricted to use with forms)
  - Typed addresses in browser, links clicked in pages etc.
  - Input data can be hard-coded directly in URLs/bookmarked e.g.

http://www.google.com/search**?hl=en&q=netskills**

The POST method

- Using `method="post"` browser makes an HTTP **POST** request for the processing URL
  - Specific method for carrying data to a server in the *body* of the request
  - Form data sent in same format but in different part of request message
  - The processing script will need to look in the querystring to extract data and use it

`http://…/formscript.php` ← No querystring appended to URL

`inputbox=Arnold&choice=A` ← Form data sent as name/value pairs in request body

- POST needs to be specifically invoked – usually via a form
  - Can be done programmatically using client-side scripting too

GET or POST for your form?

GET	POST
Default method (used if no specific method supplied)	Needs to be explicitly invoked via a form or script
Data submitted (including passwords!) visible in the querystring	Submitted data not (easily) viewable as it sent in the request body
Results page can be bookmarked as data to re-run submission is stored in URL	Results pages cannot be bookmarked or returned to later as POST data is not stored
Refreshing a result page will repeat the submission with no warning	User warned (by browser) before repeat action
Not suitable for large amount of data as URL is often truncated by browser	Can handle large amounts of data as post body can be any size
Good for searches and simple applications where repeat submission not a problem	Always use for login/password submissions

Handling Form Data

Handling form data

- Once user has interacted with form, results must be processed
- Typically involves three stages:

1. **Validation**
   - Checking that the form has been completed and the correct type of data entered
2. **Processing**
   - Extracting the submitted data and doing something useful with it
3. **Feedback**
   - (Optionally) returning something to the user

Validation

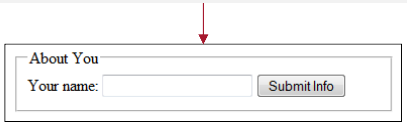
- Helps to ensure data submitted for processing is complete (required fields etc.) and consistent (format and type)
- Can be done **client-side** with JavaScript
  - Quick, efficient, better for usability
  - Can be worked/disabled around by a malicious user
- Can be done **server-side** by web server application
  - Robust, potentially more powerful, can be slower
  - Harder to work around if done correctly
- Is best done at **both** ends for data-sensitive applications
  - Specific database formats
  - Eliminating malicious input etc.

Processing (server side)

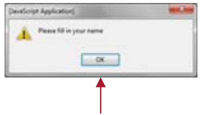
- Submitted data typically passed through web server to a server-side scripting application
  - Typically via CGI (Common Gateway Interface)
- Scripting application could be one of many
  - PHP, Perl, Python
  - ASP and/or .NET
  - Java
  - etc.
- Application may carry out some validation (previous slide) then functionality depends on the task in hand…

A simple form…

```
<form method="get" action="http://a.server.com/formscript.php">
 <fieldset><legend>About You</legend>
  Your name: <input type="text" name="username" />
  <input type="submit" value="Submit Info" />
 </fieldset>
</form>
```



…with some (very) simple client-side validation…



```
<script type="text/javascript">
function validate() {
 if (document.forms[0].username.value == ""){
   alert("Please fill in your name");
   return false; }
}
</script>
```

`validate()` runs and alerts user if box is empty then returns `false` – stopping submit process and allowing user to correct error *without* reloading the page

```
<form method="get" action="http://a.server.com/formscript.php"
      onsubmit="return validate();">
 <fieldset><legend>About You</legend>
  Your name: <input type="text" name="username" />
  <input type="submit" value="Submit Info" />
 </fieldset>
</form>
```

An *event handler* in the form forces a JavaScript function called `validate()` to run when submit button pressed but *before* data is sent to server

…and then some (very) simple server-side processing (PHP)

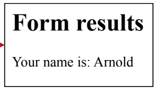
```
http://a.server.com/formscript.php?username=Arnold
```

Incoming request (with querystring)

```
<?php
if (isset($_GET["username"];)){
  $NAME = $_GET["username"];
}
?>
<html>
<head>
 <title>My Form Results</title></head>
<body>
<h1>Form results</h1>
<p>Your name is: <?=$NAME ?></p>
</body>
</html>
```

Simple *server-side* validation checks that querystring contains correct item (`username`)

If it does then value is extracted and used in results page (feedback to user)



Form results

Your name is: Arnold

Practicalities

- Creating a form-based user interface is very easy but…
- Adding the functionality for validation and processing will involve some (often complex) programming
- Creating your own client-side validation using JavaScript is usually easier (and under your control)
- In practice you may find that your host/organisation provides some standard server-based form processing/validating scripts
  - Usually tailored to the server systems they run
  - May not be possible to tailor, but may well be all you need

Reference URLs

- W3Schools tutorial
  - http://www.w3schools.com/html/html_forms.asp
- Web Standards Project guide to accessible forms
  - http://www.webstandards.org/learn/tutorials/accessible-forms/beginner/
  - There are also intermediate and advance linked from the same place
- W3C HTML forms specifications
  - http://www.w3.org/TR/html4/interact/forms.html

HTML5 Forms

Form controls in HTML5

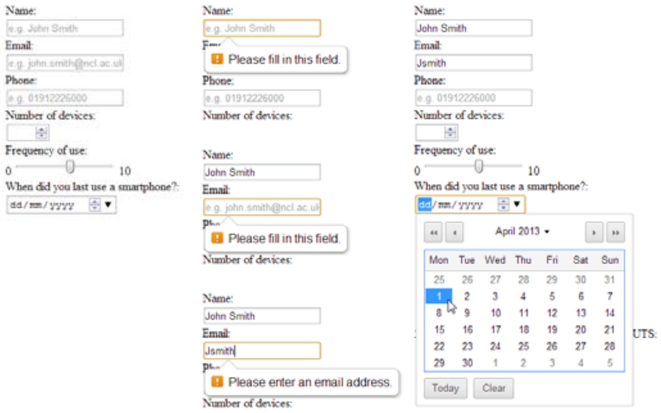
Some nice features

- Extension of `<input>` types
  - Date pickers, sliders etc.
- Baked in client-side validation
- Plus, progress, meter and output elements

…but still incomplete adoption ☹

- Need polyfill scripts and fall-backs to use new controls in most browsers

http://caniuse.com/#feat=forms



Name:
e.g. John Smith
Email:
e.g. john.smith@ncl.ac.uk
Phone:
e.g. 01912226000
Number of devices:
Frequency of use:
0 10
When did you last use a smartphone?:
dd/mm/yyyy

Name:
e.g. John Smith
Please fill in this field.
Phone:
e.g. 01912226000
Number of devices:

Name:
John Smith
Email:
e.g. john.smith@ncl.ac.uk
Please fill in this field.
Number of devices:

Name:
John Smith
Email:
Jsmith
Please enter an email address.
Number of devices:

Name:
John Smith
Email:
Jsmith
Phone:
e.g. 01912226000
Number of devices:
Frequency of use:
0 10
When did you last use a smartphone?:
dd/mm/yyyy

April 2013

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

JavaScript

Web Pages: Behaviour

JavaScript: Overview

- JavaScript is a scripting language that executes in the Web browser on the client machine
  - The browser has a JavaScript interpreter and is the script execution environment
- JavaScript is embedded in XHTML
  - i.e. the code to run arrives with the web page
- JavaScript is an imperative language with C-style syntax
  - With dynamic typing and runtime evaluation
- The only connection to Java is in the name and core C-style syntax…

JavaScript: Evolution

- Netscape introduced client scripting as *LiveScript*
  - Name changed to *JavaScript* after Netscape licensed Java support for embedded applets because…

> ***"(this) new (client-side) language should look like Java, but be a scripting language"***

- Microsoft developed a variant for Internet Explorer called *JScript*
- Both scripting languages were standardised as *ECMAScript* (as JavaScript ^{tm} belonged to Sun)

http://en.wikipedia.org/wiki/ECMAScript

JavaScript: Uses

- Interaction with user behaviour
  - Mouse movement, key presses etc.
- Interaction with browser environment
  - Browser driven events
  - Browser-type driven behaviour
- Interaction could be:
  - Validation form data
  - Creating and controlling browser windows
  - Dynamic style/presentation effects
  - Dynamic content creation/inclusion
  - Asynchronous data acquisition

JavaScript: Adding to web pages

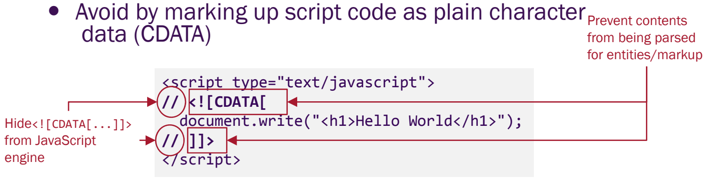
- Inline using `<script>` tags
  - The content of a script element is parsed and executed by the browser's JavaScript interpreter/engine

```
<script type="text/javascript">
  document.write("<h1>Hello World</h1>");
</script>
```

  - The `type` attribute is required
- Provides script content to be run in the current document only
  - Similar level of separation to internal `<style>` blocks

JavaScript: CDATA for internal scripts

- XHTML read as *parsed* character data (PCDATA)
  - Client should look for nested entities and markup
  - Special characters e.g. ampersands (&) in scripts can cause interpretation/validation problems
  - Special characters should be correctly defined as entities
- Avoid by marking up script code as plain character data (CDATA)



```
<script type="text/javascript">
// <![CDATA[
  document.write("<h1>Hello World</h1>");
// ]]>
</script>
```

Prevent contents from being parsed for entities/markup

Hide`<![CDATA[...]]>` from JavaScript engine

JavaScript: Using external scripts

- Include external scripts using `src` attribute

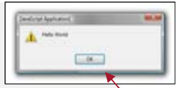
```
<script type="text/javascript" src="scripts/myscript.js"></script>
```

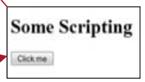
- Code between `<script>` tags will be ignored
  - No support for hybrid external/internal script blocks
  - Cannot self-close `<script>` to `<script />`
- Code can be reused across pages
  - Clean separation form content
  - Common functions can be shared
  - Shared code can be cached

JavaScript: Where in the page?

- Put as much in the `<head>` as possible (function definitions etc)
- Document `<body>` will contain direct output plus direct

```
<html>
<head>
<title>A page</title>
<script type="text/javascript">
 function hiya(){
  alert("Hello World");}
</script>
</head>
<body>
 <script type="text/javascript">
  document.write("<h1>Some Scripting</h1>");
 </script>
 <div><input type="button" value="Click me" onclick="hiya();" /></div>
</body>
</html>
```





JavaScript: Programming recap

- Common constructs
  - Variables, loops, arrays, conditionals etc.
- Interpreted (no need to pre compile)
- Loosely typed – no need to declare data types before use
- Feedback in browser error console
  - Better in Firefox (default & via extensions)

JavaScript: Data types, variables & operators

- Strings, numbers & booleans etc. (loosely typed)

```
myData = "Hello";
myData = 26;
myData = true;
myData = new Array();
```

- Good practice to initialise with `var` key word

```
var myString;
myString = "Hello";
```

- Full range of operators e.g
  - Data operators `+  -  *  /  ++`
  - Logical operators `&&  ||  !`
  - Comparison `>  <  <=  >=  ==  !=`

JavaScript: Arrays

- Arrays store related data – zero-based index
- Create and populate

```
var day = new Array("Mon","Tue","Wed","Thurs","Fri","Sat","Sun");
```

```
var day = new Array(7);
day[0] = "Mon";
day[1] = "Tue";
etc.
```

- Manipulate and access

```
day[0] = "Erp";
document.write(day[0]);

day.push("Erp");
document.write(day.length);
```

day[0]	Mon
day[1]	Tues
day[2]	Wed
day[3]	Thurs
day[4]	Fri
day[5]	Sat
day[6]	Sun

JavaScript: Loops

- Mechanisms for repeating a block of code e.g.
  - Fixed number of repetitions

```
for (i = 0; i < 10; i++) {
  document.write('<p>Hello World!</p>');
}
```

  - While a condition exists

```
do {
  name = window.prompt('You must enter your name','');
}
while (name == "");
```

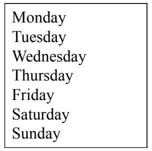
```
while(name == "") {
  name = window.prompt('You must enter your name','');
}
```

JavaScript: Loops with arrays

- V. useful dealing with dynamic content

Loop delimited by a dynamic value

```
for(i=0; i<day.length; i++) {
 document.write(day[i] + "<br />");
}
```



Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday

- Many browser/page objects return properties and values as arrays of data

JavaScript: Conditional statements

- If…Else

```
if (name == "Arnold"){
  document.write("<p>Hello Arnold</p>");
  }
else {
  document.write("<p>Who are you?</p>");
   }
```

- Switch…Case

```
switch (name){
 case "Arnold":
          document.write("<p>Hello Arnold</p> ");
          break;
 case "Dave":
          document.write("<p>Hello Dave</p>");
          break;
 default:
          document.write("<p>Who are you?</p>");
}
```

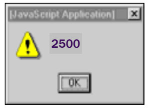
JavaScript: Functions

- Group statements for logical execution & flow control
- Define

```
function calculate(x,y) {
 answer = (x*y)/2 * 100;
 return answer;
}
```

- Call

```
thisNumber = calculate(5,10);
alert(thisNumber);
```



2500

- Can also be called by browser/page object events using event handlers

JavaScript: Comments

- Single lines

```
//circumference = 2 * radius * 3.1415;
```

- Multi-line

```
/*
 This is a
 multi-line
 comment
*/
```

JavaScript & HTML DOM

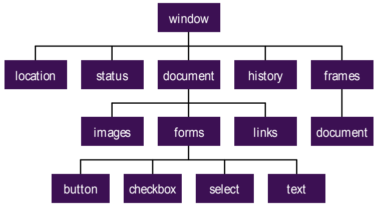
Web Pages: Behaviour

DOM: Principles

- DOM = Document Object Model
  - Language/platform neutral interface to a document
  - Defines document objects, the relationships between them & their methods and properties
- HTML DOM describes HTML documents and the browser window that contains them
- Allows access to *any* part of a document
  - As predefined (by standard/browser) objects
  - Via location in hierarchical document tree
  - As author-defined objects (using `id` attributes)

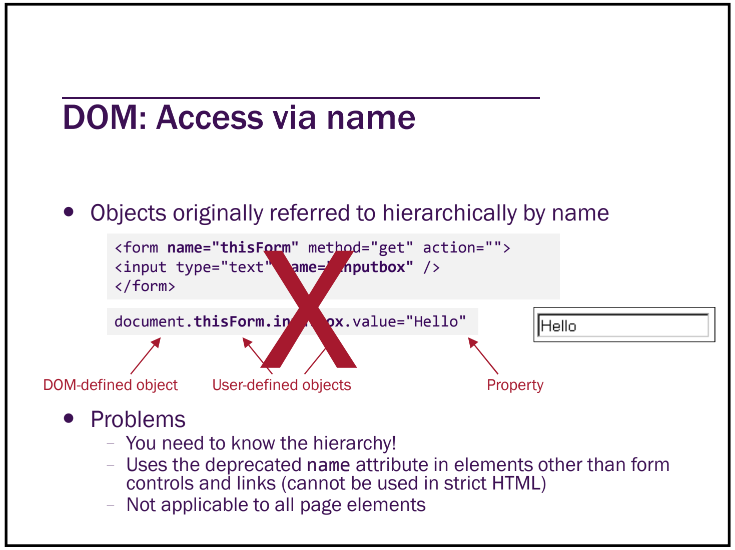
XHTML DOM: Basic hierarchy

- Browser window (or frame) is the `window` object
  - Provides global context for other objects.
- Current web page is represented by the `document` object
  - All HTML elements are components of document



DOM: Access & addressing

- DOM *nodes* accessed via range of methods
  - Some more specific than others
  - Some more restrictive than others
- Most standardised as W3C DOM 1.0 but…
  ….historically browsers developed different DOM(s)
  - Netscape (Layers) – old so completely ignore
  - Internet Explorer (All) – much less important now (but useful to test for)
  - W3C (ID) DOM – standard in use in most browsers



DOM: Access via name

- Objects originally referred to hierarchically by name

```
<form name="thisForm" method="get" action="">
<input type="text" name="inputbox" />
</form>
```

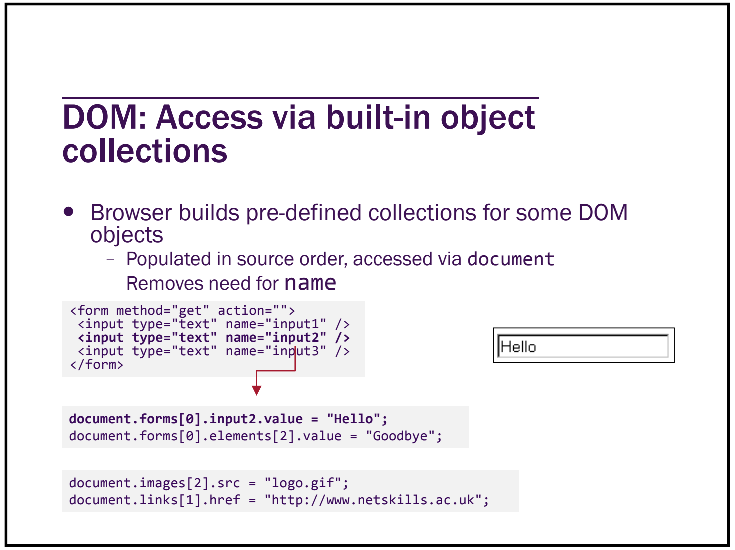
```
document.thisForm.inputbox.value="Hello"
```

DOM-defined object

User-defined objects

Property

- Problems
  - You need to know the hierarchy!
  - Uses the deprecated `name` attribute in elements other than form controls and links (cannot be used in strict HTML)
  - Not applicable to all page elements



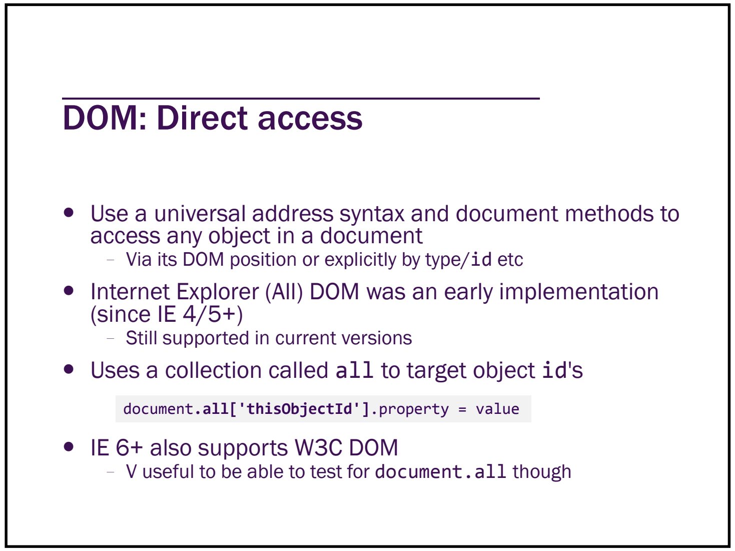
DOM: Access via built-in object collections

- Browser builds pre-defined collections for some DOM objects
  - Populated in source order, accessed via `document`
  - Removes need for `name`

```
<form method="get" action="">
 <input type="text" name="input1" />
 <input type="text" name="input2" />
 <input type="text" name="input3" />
</form>
```

```
document.forms[0].input2.value = "Hello";
document.forms[0].elements[2].value = "Goodbye";
```

```
document.images[2].src = "logo.gif";
document.links[1].href = "http://www.netskills.ac.uk";
```

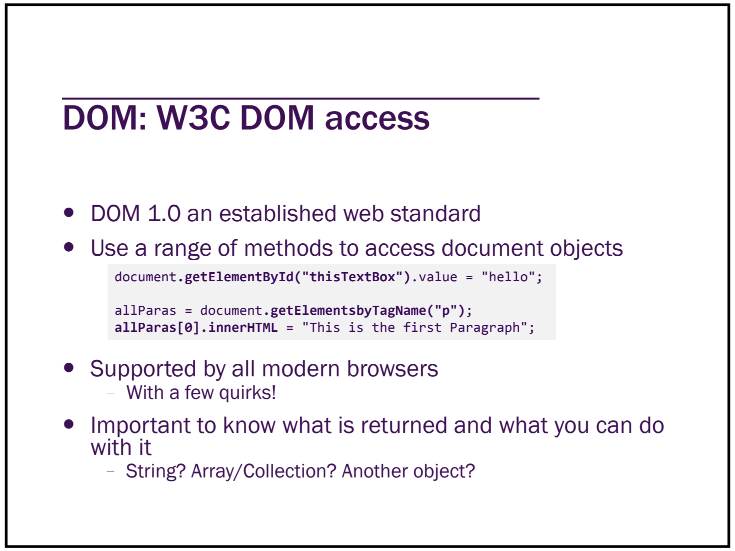


DOM: Direct access

- Use a universal address syntax and document methods to access any object in a document
  - Via its DOM position or explicitly by type/`id` etc
- Internet Explorer (All) DOM was an early implementation (since IE 4/5+)
  - Still supported in current versions
- Uses a collection called `all` to target object `id`'s

```
document.all['thisObjectId'].property = value
```

- IE 6+ also supports W3C DOM
  - V useful to be able to test for `document.all` though



DOM: W3C DOM access

- DOM 1.0 an established web standard
- Use a range of methods to access document objects

```
document.getElementById("thisTextBox").value = "hello";

allParas = document.getElementsbyTagName("p");
allParas[0].innerHTML = "This is the first Paragraph";
```

- Supported by all modern browsers
  - With a few quirks!
- Important to know what is returned and what you can do with it
  - String? Array/Collection? Another object?

DOM: Simple feature detection

- Usually a good idea!
  - Still some quirks in support for W3C DOM
- Feature sensing *not* browser detection (at best unreliable!)
  - Look for known/desired feature of intended DOM then use browser specific syntax if needed

```
if (document.all) {
  domALL = "OK";
  //Prob some version of IE
}
if (document.getElementById) {
 domW3C = "OK";
 //Prob support for W3C DOM
}
```

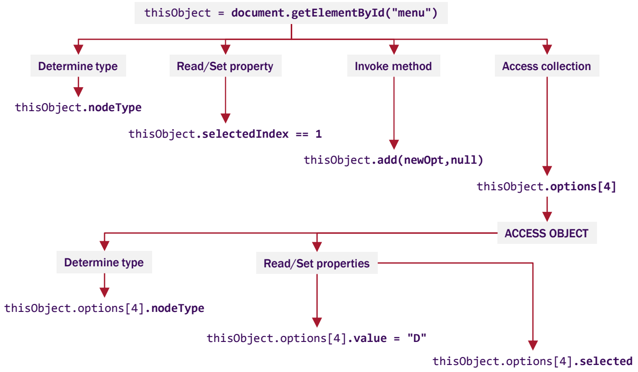
```
if ((domW3C == "OK") && (domAll == "OK")){
 alert("Prob IE 6+");
}
if ((domW3C != "OK") && (domAll != "OK")){
 alert("No nice DOM here!");
}
```

DOM: Script access

- JavaScript accesses page objects via the DOM
- Web page components inside document object will have….
  - A *Type* (radio button, link etc.)
  - *Properties* (read/write, single values)

  …and possibly…
  - *Collections* (of sub objects) (e.g. the `<option>` list for a `<select>` object)
  - *Methods* (invoked or handled when they occur)
- Uses dot operator (`.`) to access objects
- Range of ways to start the process
  - Explicitly identify object, evaluate from a collection etc.

DOM: Object access



thisObject = document.getElementById("menu")

Determine type

thisObject.nodeType

Read/Set property

thisObject.selectedIndex == 1

Invoke method

thisObject.add(newOpt,null)

Access collection

thisObject.options[4]

ACCESS OBJECT

Determine type

thisObject.options[4].nodeType

Read/Set properties

thisObject.options[4].value = "D"

thisObject.options[4].selected

DOM: Standard properties, methods & collections

http://www.w3schools.com/jsref/

- All the `HTMLElement` objects in the page support a set of basic properties, methods & collections

Collections	Description
`attributes[]`	Returns an array of attributes held by current tag
`childnodes[]`	Returns an array of the children of the current tag
Properties	**Description**
`tagName`	Returns the tag name (in UPPER case e.g. H1)
`nodeType`	Returns the object type
`innerHTML`	Set (or return) the HTML contents of a tag
Methods	**Description**
`focus()`	Gives focus to the element (e.g. a form field)
`click()`	"Click" the element (e.g. a link or button)
`setAttribute()`	Create a new attribute for the element

DOM Events

Web Pages: Behaviour

DOM: Events

- User or browser initiated
- Detected using *event handlers*
  - Inline with HTML
  - Registered in script
- Events can be programmatically invoked
  - Use object methods e.g.

```
document.getElementById('thisBox').focus()
```

...would cause a *focus* event, which could be handled by...

```
<input type="text" name="user" id="thisBox" onfocus="runThis();" />
```

DOM: Simple (inline) event handlers

- Part of HTML specification (not script)
  - Attribute of element in which the event will occur

```
<a href="somewhere.html" onclick="runThis();">Click me</a>
```

```
<h1 onmouseover="hilite('on');" onmouseout="hilite('off');">Hello</h1>
```

```
<form method="post" action="somescript" onsubmit="return validation();">
...etc...
</form>
```

- Remember XHTML is lowercase

`onclick=""` ***not*** `onClick=""` etc...

DOM: Event handler registration

- Event handlers can be registered within a script
  - No need for HTML event handler attributes
  - Several ways of doing this, linked to evolution of DOM
  - Can get confusing (i.e. prepare to run into some browser differences)

```
function clickAlert() {
  alert('You clicked me!');
}
```

"Traditional method"

```
document.getElementById("goClick").onclick = clickAlert
```

IE "Event attachment"

```
document.getElementById("goClick").attachEvent('onclick',clickAlert)
```

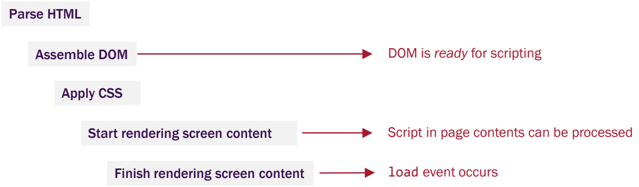
W3C "advanced/standard method"

```
document.getElementById("goClick").addEventListener('click',clickAlert,false)
```

http://www.quirksmode.org/js/introevents.html

DOM: Ready or loaded?

- Script *can* be triggered to run as soon as the page has been rendered on screen (using `onLoad`) to ensure objects required actually exist
- However script access to the DOM can occur *before* the page is rendered on screen



Parse HTML

Assemble DOM → DOM is *ready* for scripting

Apply CSS

Start rendering screen content → Script in page contents can be processed

Finish rendering screen content → `load` event occurs

DOM: Detecting "DOM ready"

- Standard method is to add an event listener for the `DOMContentLoaded` event occurring for the `document`

```
document.addEventListener("DOMContentLoaded", doSomething, false)
```

- Unfortunately IE doesn't support this
  - Both `addEventListener` and the detection of `DOMContentLoaded`
- Workaround is to get IE to load up a (dummy) script file and tell you when that has happened

Form validation

Web Pages: Behaviour

Forms: Validation

- Basic form operation is limited

  "...press submit, collect 'successful' data, send to server..."

- Robust applications validate input prior to processing to avoid errors/issues
  - Missing data, incorrect format/type etc.
- Validation can take place at the server
  - Allows request information to be checked (referrers etc)
- But server validation requires
  - Extra round trip via HTTP
  - Maintaining of state during submit > check > resubmit

Forms: Client-side validation

- Form data can be validated in the client
  - *Before* the HTTP request to submit occurs
- Can be used to check completeness, format, type, value etc.
- If validation succeeds then HTTP submission occurs
- If validation fails then HTTP request is never made and control drops back to the user
  - Without reloading the form/losing progress thus far

Forms: Triggering client-side validation

- Need to capture and handle the `submit` event as it occurs in the `form` element
- Use an event handler!

```
<form method="post" action="http://..." onsubmit="return validate();" >
```

- Event handler waits for the return value from `validate()`

Return value	Outcome
`true`	Allow submit event to continue and send data to server
`false`	Stop submit event and return control to the client

Validation: Simple checks

"...has something been entered/selected?"

- Access required form control objects and query appropriate state/value
  - Handle via conditional statement
  - Assemble/deliver user feedback
  - Return true/false as appropriate
- Best practice to provide one set of feedback
  - Not per question!
- Usually better to validate whole form
  - Rather than per control (e.g. via `onfocus`/`onblur` etc)

Validation: Data checks

- Can be done via data type
  - Is it a number? string? etc
- More powerful to accurately match patterns in the input
- Unless specifically looking for a fixed string/number you'll need to enter the world of...

Regular Expressions!

Regular expressions

Web Pages: Behaviour

Regular expressions

- A "standardised" pattern matching syntax for text
  - Define pattern – test against input
- Can appear baffling at first!
- Are actually pretty logical and (relatively) straightforward to use

```
/^[\w]+([\.\w-]*)?@[\w]+(\.[\w-]+)*(\.[a-z]{2,3})(\.[a-z]{2,3})*?$/i
```

```
something(.something)@something.xx(or.xxx)(.xx or .xxx)
```

Regular expressions: Building patterns

- Square brackets [ ]
  - Match any one of the characters or ranges in the brackets

```
[ae]  matches one of a or e
[a-z] matches any one of the lower case letters
[0-9] matches any one of the digits
```

- Caret ^ negates a range (match anything *but*…)

[^a-z] anything *but* the lower case letters
[^5-9] anything *but* the digits 5, 6, 7, 8, 9

- Escape special characters with \

[\[\]] matches opening or closing square bracket
[\.a-z] matches a dot (.) or a single lower case letter

Regular expressions: Meta-characters

- Shorthand for common ranges

Meta-character	Matches	Equivalent range
.	Any character	N/A
\d	A digit	[0-9]
\D	A non-digit	[^0-9]
\s	A whitespace character	[\t\n\x0B\f\r]
\S	A non-whitespace character	[^\s]
\w	A word character	[a-zA-Z0-9_]
\W	A non-word character	[^a-zA-Z0-9_]

Regular expressions: Quantifiers

Quantifier	Effect
`[a-z]?`	A letter, zero or one time
`[a-z]*`	A letter, zero or more times
`[a-z]+`	A letter, one or more times
`[a-z]{n}`	A letter, exactly `n` times
`[a-z]{n,}`	A letter, at least `n` times
`[a-z]{n,m}`	A letter, between `n` and `m` times

Regular expressions: Greediness

- Quantifiers are *greedy* by default
  - Matching as many times as possible until end of string before *backtracking* to conclude pattern
- Try matching the opening `<b>` tag in `some <b>bold</b> text`
- A simple pattern should work `/<.+>/`
- But the quantifier `+` is *greedy* and keeps matching until it reaches the end of the string to find…

  `<b>bold</b> text`
- Then *backtracks* to finally match…

  `<b>bold</b>`

Backtrack to find the end of the pattern i.e. the *last* >

Regular expressions: Laziness

- Append `?` to the quantifier to make it *lazy*
  - Match as few times as possible before backtracking to conclude pattern

  `/<.+?>/`
- Now it backtracks after each match to complete the pattern…meaning a match occurs after the first `>` character

  `<b>b`

Backtrack each time to find the end of the pattern i.e. the *first* >

http://www.regular-expressions.info/repeat.html

Regular expressions: Anchors & flags

- *Anchors* fix expression to start/end of string or boundaries between word/non-word characters

Anchor	Matches
`^`	The beginning of a string
`$`	The end of a string
`\b`	A word boundary
`\B`	A non-word boundary

- *Flags* are appended the end of an expression

Flag	Matches
`i`	Use case-insensitive matching
`g`	Global matching (instead of stopping at first match)
`m`	Multiline mode

Regular expressions: JavaScript

- JavaScript supports regular expressions in a couple of ways:
  - via the `RegExp` object (more powerful)
  - via the `String` object (simple but less options)
- The RegExp object is defined as a pattern to match
- RegExp object methods use/test/apply pattern where needed

```
var pattern = /^[a-z]+$/i;
```

Creates a RegExp object

```
if (pattern.test(someString)){
  alert("Yay")
}
else {
  alert("Boo");
}
```

Tests string against the expression

Regular Expressions: JavaScript RegExp methods

- JavaScript `regExp` object has two methods

Method	Purpose
`thisPattern.exec(someString);`	Return an array of info about the first match (or `null` if no match)
`thisPattern.test(someString);`	Return `true` or `false` if string contains a match

Regular Expressions: JavaScript string methods

- JavaScript `string` object can use regular expressions in three string matching methods

Method	Purpose
`someString.search(/^[a-z]+$/i);`	Return position of first substring match (-1 if no match)
`someString.replace(/^[a-z]+$/i,"X");`	Replace the text matched by expression with string in second parameter
`someString.match(/^[a-z]+$/i);`	Return and array containing all the matches for the expression

Regular Expressions: JavaScript form validation

Check for no input

Pattern for letters only

Check user input against pattern

Act on outcome

```
if (thisForm.inputbox.value != ""){

 var pattern = /^[a-z]+$/i;

 if (pattern.test(thisForm.inputbox.value)){
  //SUCCESS :-)
  }
 else {
  //FAILURE :-(
  }
}
```

Regular Expressions: testing tools

- Constructing regular expressions can be fiddly
  - Try and avoid doing it in live code!
- Online testing tools are very useful – copy/paste final expression
  - Try and use a test tool using the correct expression engine i.e. JavaScript, PHP, Perl etc.
- JavaScript-based

  http://regex101.com/#javascript

  http://www.regular-expressions.info/javascriptexample.html
- General purpose (PHP-based)

  http://www.phpliveregex.com/

Regular Expressions: Reference & tutorials

http://www.regular-expressions.info/tutorial.html

http://www.regular-expressions.info/examples.html

http://www.regular-expressions.info/reference.html

http://lawrence.ecorp.net/inet/samples/regexp-intro.php

Remember these...?

Form controls in HTML5

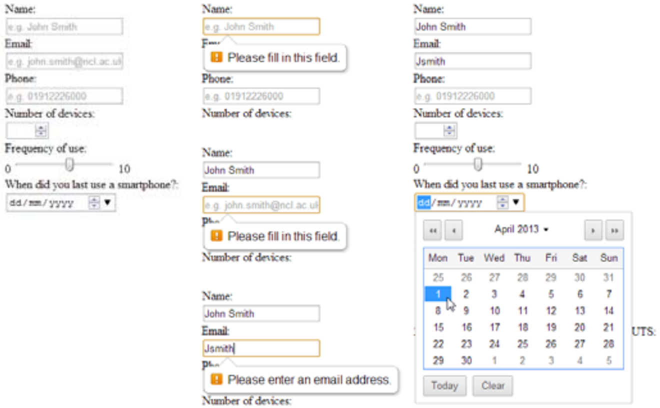
Some nice features

- Extension of `<input>` types
  - Date pickers, sliders etc.
- Baked in client-side validation
- Plus, progress, meter and output elements

...but still incomplete adoption ☹

- Need polyfill scripts and fall-backs to use new controls in most browsers

http://caniuse.com/#feat=forms



Name: e.g. John Smith
Email: e.g. john.smith@ncl.ac.uk
Phone: e.g. 01912226000
Number of devices:
Frequency of use: 0 10
When did you last use a smartphone?: dd/mm/yyyy

Name: e.g. John Smith
Please fill in this field.
Phone: e.g. 01912226000
Number of devices:

Name: John Smith
Email: e.g. john.smith@ncl.ac.uk
Please fill in this field.
Number of devices:

Name: John Smith
Email: Jsmith
Please enter an email address.
Number of devices:

Name: John Smith
Email: Jsmith
Phone: e.g. 01912226000
Number of devices:
Frequency of use: 0 10
When did you last use a smartphone?: dd/mm/yyyy

April 2013

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

Practical scripting

Web Pages: Behaviour

JavaScript in the wild

- Scripting support has stretched the horizon of what is possible in "just a web browser"
  - However poorly applied scripting can be a big contributor to poor website experiences
  - You can easily improve your scripting at two levels
- Design level
  - Use your tools wisely & add *appropriate* features
  - Practice unobtrusive scripting ☺
- Development level
  - Don't re-invent the wheel
  - Use JavaScript frameworks like **jQuery** to save time and cross-browser headaches

Unobtrusive JavaScript

- *"Unobtrusive JavaScript"* is the name given to a collection of techniques which aim to ensure that JavaScript is used in a way that is:
  - Beneficial (to both the content and the user experience)
  - Responsible (in its use of browser resource)
  - Scalable (or removable)
- Key aims are:
  - Keep JavaScript separate from XHTML markup (separate behaviour from content)
  - *Degrade gracefully* (enhance but make sure that content is available with or without JavaScript)
  - Do not limit accessibility (and ideally enhance it)

Gracefully degrade or progressively enhance?

- Largely a question of mindset for the developer
  - Net result should be broadly similar!

Progressive enhancement
Design for the core functionality and allow for *progressive addition* of features *if* supported by current browser environment

Graceful degradation
Design for the latest browsers and allow for *graceful removal* of features *if not* supported by current browser environment

What any given user & browser combination will actually see

JavaScript frameworks

- Pre-built libraries of common functionality
  - Save you from handling browser inconsistencies
  - Enable you to quickly and easily include complex interactions
- You do need to know something about the process at hand though!
  - Only really work if you have a clear understanding of:
    - Your XHTML structure
    - CSS selector syntax and properties
    - The desired effect (if it is appropriate for your content)

jQuery

http://jquery.com/

- Arguably the most popular, current JavaScript library
- Easy to get started with
  - Download library, include via `<script>` tags
  - Try basic tutorial, get cracking... ☺
- Also highly extensible
  - (Lots of) third-party "plugins" for specific effects/functionality
- jQuery plugins are just more JavaScript
  - i.e. include in page alongside jQuery and run

jQuery example

Include the core jQuery library

```
<script type="text/javascript" src="jquery-1.11.0.min.js"></script>
```

```
<script type="text/javascript">
 $(document).ready(function(){

   $("#fadeControl").mouseover(function(){
     $("#fadeAction").fadeOut('slow');
   });

   $("#fadeControl").mouseout(function(){
     $("#fadeAction").fadeIn('slow');
   });
 });
</script>
```

jQuery handles the DOM ready testing

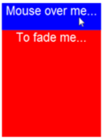
$ sign indicates these are jQuery instructions

jQuery handles event registration

Built-in jQuery function for content effect

Use CSS selector syntax to identify where code applies in page

```
<body>
  <div id="fadeControl">Mouse over me...</div>
  <div id="fadeAction">To fade me...</div>
</body>
```



References

https://magentaer.com/devopera-static-backup/http/dev.opera.com/articles/view/the-seven-rules-of-unobtrusive-javascrip/index.html

http://docs.jquery.com/Main_Page

http://docs.jquery.com/Tutorials

Other frameworks and templates...

Other frameworks and templates...

https://modernizr.com/

http://getbootstrap.com/

https://html5boilerplate.com/