Browsers form the Web's user interface, with web versions of everything graphical

Browsers  Tag Soup  HTML  Development
Browsers

The official term for a browser is *user agent* i.e. any device or program which visits web pages.

The main desktop/laptop browsers are *Chrome*, *Firefox*, *Edge* (IE), *Safari*, *Opera*, *Vivaldi*.

The main mobile/tablet browsers are *Chrome*\textsuperscript{m}, *UC*\textsuperscript{m}, *Safari*\textsuperscript{m}, *Opera*\textsuperscript{m}, *Android*\textsuperscript{m}.

You also need to be aware of minority browsers, version differences, platform differences, accessibility for the disabled, web crawlers, and see [list of browsers](#).
Standards

We said before that *standards* are vital, so where are they? They are looked after by a group called \( W3C = \text{WWW}C = \text{World Wide Web Consortium} \)

[W3C website](#)

Standards are called *recommendations*  \( W3C \text{ notation} \)

We are particularly interested in the standards for HTML, CSS, JavaScript, PNG, SVG

If you write pages according to the standards, they should work in all standard-conforming browsers, and they should be *future-proof*
W3C Notation

A standard starts life as a living standard or discussion document, then it becomes, in turn, a draft recommendation, candidate recommendation, proposed recommendation, and finally a recommendation.

The name is presumably because it has no legal force.

The final stage guarantees two 100% interoperable implementations (i.e. two browsers support it).

Living standards are produced by the breakaway group WHATWG which produced HTML5, but it is not (yet) clear that they have enough stability for authoring.
Checking Browsers

A critical question is how well browsers conform to the standards, and there are some test and info sites: HTML5, CSS3 etc., comparisons, details, tables

The overall picture is: old browsers are a pain, things have improved enormously recently, the latest versions of the 'big' browsers support the standards well, but there are still worrying differences of detail

In theory, up-to-date browsers have 100% HTML5 support, but in practice, there are dusty corners
Ignoring Browsers

It is common for web authors to ignore any browser version with less than 1% of the usage share.

Important browsers which can now be ignored are Netscape 4, IE5, **IE6** (died 2015), IE7, IE8 (died 2016), so *all* browsers now accept HTML5!

*Beware:* lots of web sites still have obsolete info relating to how to support these dead browsers, which affects your Google strategy when trying to find techniques – you *must* try to avoid obsolete techniques – add HTML5 to your Google queries.
The Best Browser

What is the best browser for users? Answer: big-name browsers are better, but it doesn't matter much, and people choose according to unimportant (UI) features.

What is best for developers? Answer: (a) Chrome (b) Firefox (c) Safari, Edge, Opera: keep it up-to-date!

Why are older browsers or older versions OK for users? Answer: because web developers put a lot of effort into making sure their sites work in all browsers.
Developer tools

Good development browsers (Chrome, Firefox, etc.) have tools to help examine and debug pages.

As well as the menu entry, you can right-click on a page and choose "inspect" (or similar).

You can view the objects making up the page, the style, the JavaScript debugging messages (in the console tab) and the requests/responses.

Get used to using these tools.
Old browsers

Roughly speaking, 'old browsers' means pre-HTML5 browsers, up to and including IE8

All old browsers are individually below the 1% level, but as a group they are worth paying a little attention to, where the effort is small (by using standards, polyfills, feature detection)
Tag Soup

When HTML was (eventually) standardized, the standard allowed browsers a lot of choice, including how to handle incorrect HTML.

During the browser wars, browsers competed for the 'cleverest' (different) ways of accepting bad HTML to make things 'easy' for 'non-experts'.

Pages were tag soup, a mess of invalid, misused, unstructured, bloated, meaningless tags.

Unfortunately, many authors (and authoring tools) still produce tag soup.
Invalid HTML

The page has no doctype, no html or body tags, no head, and no title. The heading tags don't match. The first paragraph has no tags round it to say what it is. (According to the standard, text is not allowed directly in the body). The second paragraph has a start tag, but no end tag. The end-bold and end-italic tags are in the wrong order. (In any case, the b and i tags don't say why the text should be bold or italic, see later).
The `br` tag and `&nbsp;` entity are wrongly being used to generate vertical and horizontal space. The `blockquote` tag is being used to indent a paragraph. These things should be done using a stylesheet.
Web-authoring tools often produce rubbish like this. This is how a computer-illiterate person might use a word-processor. How do you keep all headings consistent? How do you change all the headings later? How do you remember all the properties, or prevent mistakes? The right thing to do is to use a tag such as h2, set its style, and perhaps make small exceptions using a class. And by the way, it might be worth automating the section number.
Bloated HTML

Here is a pinboard of photos:

```
<table> ... </table>
```

Assuming that the table uses a fine grid for random placement, this produces unreadable, bloated, slow HTML code. Some web-authoring tools used to use this fine-grid table technique for laying out all web pages. The right technique is to place each photo at explicit (x,y) coordinates using a stylesheet or script.
Meaningless HTML

A *statement* is a line of text such as `<font face="Courier">n = n+1</font>` which *does* something.

The *i*, **b**, `font` tags don't say *why* the styles are being changed. These days, you are supposed to try to say what you mean, and then make it look right with a stylesheet. An improvement might be:

A **dfn** statement is a line of text such as `<code>n = n+1</code>` which **strong** does something.

This helps with global changes, glossaries, scripts etc.
Every browser has several modes

If the page meets a standard, and its web site says so, then the browser behaves according to that standard.

Otherwise, it uses **quirks mode** for old tag soup pages, where it uses old display conventions.

It is **vital** for pages/sites to (a) trigger a standard mode and (b) follow the standard, otherwise the pages will be completely unpredictable between browsers.
Avoiding Tag Soup

We **must** have a strategy for developing web pages which avoids tag soup, for two reasons:

- To produce standard pages which work predictably in most browsers, and which are machine readable.
- To avoid spending endless hours trying to get a web page to work in multiple browsers.

You will not escape from the "endless hours" problem entirely, but starting out with a good strategy will help to keep it under control.
The Three Aspects

Besides following standards, tag soup can be avoided by dividing web page design into three aspects:

• **structure** deals with what pages mean, the different types of content, and their nesting

• **style** deals with how pages look, e.g. layout and sizing and colour and decoration

• **behaviour** deals with what pages do, animation and user interaction

Each of these has a specific standardised language, and each is in its own file
The Three Languages

- XHTML
- CSS
- JavaScript
Integrated Technologies

The three languages XHTML, CSS, JavaScript only cover writing and animating, and more technologies are needed for painting, drawing, multimedia etc.

Plugins and proprietary formats (GIF, JPEG, QuickTime, Flash, applets) should no longer be used.

Instead, HTML5 uses integrated technologies, defined by open standards, which are built into "all" browsers (PNG, SVG, MathML, canvas, audio, video)
Integration

What does it mean for a browser technology to be *integrated*?

Take SVG, an open and integrated replacement for Flash, for creating vector drawings.

Text in an SVG drawing can be copied-and-pasted, and searched using Google.

JavaScript code can reach items in the drawing to change them, and style can be applied to things in the drawing, e.g. the background colour.
Versions of HTML

A decision is needed about what version of HTML to use for this unit

In October 2014, HTML5 became an official standard, guaranteeing two 100% implementations

Since then, support for HTML5 has improved

The only issue is whether or not HTML4 makes it easier to support old browsers (if we care)
Why not HTML4?

But using HTML4 does not make it significantly easier to support old browsers.

HTML4 consists of many standards (HTML4.01, XHTML1.0, DOM, WOM).

HTML4 is an "80% standard"; 20% of the time it allowed browsers to do whatever they like (which they did).

HTML4 contains deprecated features.

Browser compliance with HTML4 was very patchy.

So, the HTML4 standard doesn't help.
Deprecation

Note: it's *deprecation* not *depreciation*

If something is deprecated in a web standard:

- it *must not* be used in new web pages
- it is allowed only so that old pages will still work
- it will be removed in the next standard

One problem is that validators typically don't check the first point, so self-discipline is needed
Why HTML5?

HTML5 includes XHTML, DOM, WOM all-in-one

The HTML5 syntax rules are carefully designed so that a valid HTML5 page stands an excellent chance of being handled well by old browsers.

Most new features involve new tags which are just ignored as not-understood by old browsers.

Validation, polyfills, and feature detection help make HTML5 pages work on old browsers.

All this makes HTML5 the right choice for us.
XHTML

XHTML is an XML-compatible version of HTML, basically just following tighter rules. The main advantage is that it is currently the easiest way to make sure that your web pages are valid.

Also, it allows a web page to be processed automatically more easily e.g. to produce PDF, or make a glossary, or spell-check, or translate, or just do syntax highlighting in an editor!

The disadvantages are that it is more verbose and fussier, e.g. all close tags have to be included explicitly.
XHTML History

XHTML 1.0 was successful, making HTML4 compatible with XML by tightening up the syntax.

XHTML 1.1 and 2.0 started breaking up HTML into separate XML-based languages, but this was incompatible with old browsers, and too fussy, and not popular, and the approach has been abandoned.

XHTML5 is part of the HTML5 standard, roughly like XHTML 1.0, with the proposed sublanguages now integrated, retaining compatibility with old browsers.
Using XHTML is a matter of opinion - mine is that avoiding XHTML causes more problems than it solves.

In this example, the open tag `<b>` has no close tag, HTML5 (not XHTML5) will insert it for you, but where?

As well as learning the HTML5 syntax, you have to learn the complicated rules about implicit close tags go, and even then making mistakes is easy, even in valid pages.

Advice: use XHTML and an editor that inserts close tags!
Avoiding XHTML

If you choose to use loose HTML5 instead of tight XHTML, *read this page carefully*

You **must** make sure your pages are valid (and it is likely to be awkward and time-consuming, because you get no instant feedback)

So you **must** build validity checks into your process, e.g. by downloading and using `vnu.jar`

If you use a framework, create complete samples of your pages for validity checking, e.g. with `wget`
The aim must be to write a single version of every page, and have it work in both old and new browsers.

That means page names must end .html, not .xhtml.

Polyglot XHTML5 is a further tightening of the rules so that pages are both XHTML5 (for new browsers) and HTML4-compatible (for old browsers) at the same time, and then your site can use content negotiation to deliver the right content type.

This is worth doing, if it doesn't cause much extra work.
Content types

A web server uses response **headers** to describe the content of web pages (and other files)

The header `Content-Type=text/html` tells a browser to interpret the page as loose HTML5

`Content-Type=application/xhtml+xml` tells a browser to interpret the page as XHTML5
A server can check whether a browser understands XHTML by looking at the Accepts header on the request that comes from the browser.

Then it can deliver the page with Content-Type set to text/html for old browsers, and application/xhtml+xml for new browsers.

Those 3 or 4 lines of code in the server are called *content negotiation*. 
Headers

A *request* from a browser to a server, or a *response* from the server to the browser, is a text file.

As with an email, the file has headers and a body.

The **Accept** header on a request says what types of content a browser is prepared to accept in the response.

There is a **Content-Type** header on a response (similar to the MIME-type of an email attachment).

The body of the response is the HTML text – the headers are extra, added by the server.
Why negotiate content?

For a valid XHTML5 page, the result is the same, whether delivered to a new browser as `text/html` or as `application/xhtml+xml`.

So why bother with content negotiation, why not always deliver as `text/html`?

For users, *if* pages have been validated somehow, it is not important.

For developers, `application/xhtml+xml` delivery gives *instant feedback for invalid pages* (or at least not well-formed pages).
Development

As with all issues in Web Tech, a strategy is needed to cut through the complexities:

- Use Chrome or other full-HTML5 browser with good development tools as the primary development browser
- Use a local development server, configured to be platform independent and to deliver pages as application/xhtml+xml
• Avoid IDEs for now; to make sure you understand HTML, just use a programmer's editor with syntax highlighting etc. (maybe have a look at brackets)
• Start with a blank page, add a tiny amount at a time
• Use the browser's XHTML5 instant error messages to do most of the validity checking
• Occasionally use a validation tool such as vnu.jar for more detailed validity checking
Development 3

• Add one feature at a time, keep structure, style and behaviour separate, in html, css and js files
• For each feature, use Google, find the most standard and up-to-date technique, use HTML5 in queries to avoid obsolete stuff
• Check compatibility tables to check how well browsers support the feature
• Get it working in the development browser
• Go back to Google to find out how to fix things up for old browsers, if you care