Web Technologies
Overview

Background information, and one approach to the potentially huge task of learning about the Web

Aims  Search  Standards  Design  Development
Aims

• Survey the huge number of web technologies
• Organise them to form an overview
• Investigate some of them in depth
• Develop both technical and creative skills
• Learn how to avoid the major pitfalls
• Configure and program a web server
• Provide support for web-based projects
A Sea of Technologies
Professionalism

Any 'amateur' web developer can follow a tutorial and use a framework – your aim is to be a 'professional' web developer, who understands more deeply, e.g. about:

- protocols
- standards
- accessibility
- customization
- security

Your site should (a) not look like everybody else's and (b) behind the scenes, not behave like everybody else's
The most important classification of web technologies is the division into these two types:

**Client side technologies** are mostly browser related, and provide the Web's user interface, further divided into **writing, painting, drawing, animating**

**Server side technologies** are about protocols and programming and databases and provide the Web's functionality

*asides*  *kinds*  *is it Web or web?*
Asides are extra comments which are not part of the main discussion.

In this unit, asides are lecture slides which are not normally displayed, when viewing a chapter as an HTML page, though they do get printed. They are available only as links, and they may or may not be mentioned in lectures.
## Kinds of Technology

There are various kinds of web technologies:

<table>
<thead>
<tr>
<th>Protocols</th>
<th>Object Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platforms</td>
<td>File Formats</td>
</tr>
<tr>
<td>Meta Languages</td>
<td>Browsers</td>
</tr>
<tr>
<td>Data Languages</td>
<td>Search Engines</td>
</tr>
<tr>
<td>Scripting Languages</td>
<td>Web Servers</td>
</tr>
<tr>
<td>Animation Languages</td>
<td>Databases</td>
</tr>
<tr>
<td>Programming Languages</td>
<td>Validators</td>
</tr>
<tr>
<td>Libraries</td>
<td>Frameworks</td>
</tr>
</tbody>
</table>
Is it Web or web?

Should you write *Web* or *web*? This is not a very important issue, but some people may want to know.

There isn't a universal agreement, but a common convention is to use *Web* when it is a noun, and *web* when it is an adjective, e.g. "the Web is big", "my page is on the Web", "my web page", "the BBC web site", "web technologies".
This unit caters for

- experienced programmers who want to show off their programming skills on the Web
- beginning programmers who want to learn how to put things together on the Web with just a little programming glue

On the client side, this means *JavaScript* – no choice

On the server side, there is a wide choice, but we will also be using *JavaScript*
**Technical Tess:** You need technical skills for web work, to know how to do things

**Creative Kev:** But you also need creative skills, to know what looks and feels good

The ideal is to combine technical with creative skills. Technically, you need to know how to get things to work on all platforms and how to combine tricks of the trade, tools, techniques to get specific effects. Creatively, you need to know what will appeal to everyone, and what will cause minimal confusion.
The Web is our target, *and* our most important tool

Strangely, the Web is just old enough... to hold everything you need to know about the Web

The recommended key to the Web is [Google](http://www.google.com) which may be the most important invention since the Web

[www.google.com](http://www.google.com)  [www.google.co.uk](http://www.google.co.uk)

Another important general resource is Wikipedia at:

[en.wikipedia.org](http://en.wikipedia.org)  [Googling](http://www.google.com)  [Google](http://www.google.com)
You may *think* you are good at googling, but are you really?

Complex issues need tough Google research, not just accepting the first thing you find

Here are some tips...
What is the quickest way to find out what Wikipedia says about browsers? Answer, type:

\texttt{wiki browser}

into your google bar.
Reject the Rubbish

The web is **full** of junk about web design, e.g. obsolete techniques to do with old browsers or old standards, or just people who don't know what they are talking about.

You need to learn to recognize this rubbish and ignore it.
Find Things Fast

Don't plough through hundreds of hits from a search, instead, refine the search by changing the phrase

Learn to use 'advanced' Google features like "exact phrases" and site:www.cs.bris.ac.uk restrictions

Work quickly, covering as many sources as possible and don't get distracted from your main objective
At first sight, there seems no way to assess how truthful or accurate or unbiased a site is, so don't spend time on it (especially "flame wars")

Instead, work very rapidly, visiting as many sites as you can, to build up a kind of consensus view

Even though each individual opinion is suspect, an average over a lot of opinions can be quite solid

Google does some of this for you in the first place, by giving you pages in popularity order
Look for Lists

Suppose you are looking for a software tool or library package or any kind of "product". The site for each product is of limited use because it is weak on context, criticism or comparisons.

Look for sites which list, summarise or compare products, e.g. put several product names into a query.

Add words like "survey", "review", "article", "choose", "compare", "comparison", "evaluation", "advantages", to your queries to seek out list sites.
Catch the Cons

To evaluate something, you need to know its pros and cons, but nobody tells you the cons.

A product site will tell you the pros because it is advertising itself, and survey sites only tell you minor cons, because they compare products which *do* the job, and fail to discuss ones which *don't*.

Look for sites which criticize products.
Don't take what people say about themselves or their own products at face value. Look for other sites which recommend the main site.

For example, pay more attention to products that appear on several list sites.

Pay more attention to "independent" sites which describe or explain or criticise or compare.

Add words like "tutorial", "forum", "explanation", "discussion", "problems", "overview" to queries.
Bin your Bookmarks

Bookmark important sites? No: to find onelook.com type onelook or multiple dictionaries into Google

Bookmark wikipedia? No: to find out about worms, type wiki worms into Google

Bookmark obscure sites? No: e.g. to buy uni car park coupons, type bristol parking coupons into Google

Bookmark interesting but irrelevant pages that you find while looking for something else, to avoid distraction, so you can go back to them later
Tutorials or stack-overflow posts show you how, but hardly ever tell you why.

Don't completely believe or trust instructions, and don't just use the first ones you find.

Read around until you understand the issue.

Don't be like non-computer-scientists who have no idea what they are doing, and can only blindly follow tutorials.
The 'right' answer to an issue changes over time, either because your understanding improves, or because the technology gets updated.

So when you have an answer, don’t just leave it there.

Review the situation every so often.
The exact details of how Google works are a company secret. However, there is a research paper available describing how Google worked before it went commercial:

[Research Paper]

Here are some more facts and guesses...
Google supports a huge number of queries per day. So, it has a massive connection to the internet (a very "thick pipe", to use a water services analogy), and a massive amount of computing power:
What provides Google's computing power? Well, the cheapest computing power you can buy is the ordinary PC, and Google basically uses huge numbers of ordinary PCs. Actually, they are just PC boards, because all that is needed for each one is a main board, processor, memory, network connection, and perhaps an on-board disk.
As well as providing a lot of computing power for reasonable cost, this setup also provides excellent short term and long term maintenance. For short term maintenance, a PC or bank of PCs can be shut down for repair or whatever, and then started up again, without shutting down the overall service. For longer term maintenance, to keep up with ever increasing demand, the amount of computing power can be increased continually by adding or upgrading individual PCs.
Parallelism

How can you use lots of individual PCs, each of which is not all that powerful, to solve a massive problem? The answer is parallelism. The problem is one which can be dealt with by parallel processing, since it consists of a large number of small queries, which are essentially independent of each other. This is fortunate, because a lot of large problems cannot be broken down in this way. The networking hardware and the software are organised so that lots of queries can be dealt with simultaneously, by routing them to different PCs.
Although the processing required for each query is independent of other queries, the data the queries need is shared, so there are tough organisational problems to be solved. Some of the main techniques are layering (using primary "indexing" data to access secondary "bulk" data), compression, caching (holding a copy of disk data in memory), and replication (having lots of copies of data). On the other hand, the data is pretty static (a new version of the entire data is calculated offline every few days or weeks, and this then replaces the old version).
The main impressive feature of Google, beyond its efficiency, is its accuracy. When Google first emerged, it was stunningly accurate, compared to its nearest rival search engine, and it is still excellent.

This is because it retrieves results based on two different criteria. The first is relevance and the second is popularity. The results are the most popular and most relevant to the search phrase you type in.
The relevance calculation is complex, but not particularly exciting. Roughly speaking, a page is more relevant if it contains more words from your query phrase, if those words are in more prominent places such as titles, and if the occurrences of the words are closer together.
What is more interesting is the popularity calculation, which uses the PageRank™ algorithm. A page is more popular if it has more links pointing to it. But more than that, a page is more popular if those links come from popular pages! That sounds like a circular definition, and it is, so an iterative convergent algorithm is needed to calculate the result.
Google is an automatic index of the pages on the Web. Automatic indexes are never as good as indexes produced by intelligent human effort. For example, for many topics, looking them up on Wikipedia is much more effective than searching for them on Google. So why is Google still important?
If you order all the pages on the internet by popularity, the list has a long tail. There are quite a lot of popular pages near the front of the list which are better served by manually created indexes. However, there are also a lot of minority interest pages. These are not very popular, and will never be served well by manual indexes. Overall, these minority pages vastly outnumber the popular ones.
Technical answers

It is very common to want answers from long tail pages, especially when doing technical work. For example, if you come across an obscure error message from a tool you are using, you can cut and paste the message into Google, put quotes round it for an exact match, and find an obscure discussion site which explains its meaning and how to deal with it.

It is this long tail which Google deals with particularly well.
Standards

The Web consists of lots of browsers connecting to lots of pages via the Internet
First, how many browsers are there?

Let's ignore minor version differences, and different user preferences

Using the wikipedia entries List of web browsers, Mobile browser and others you quickly reach over 100

More importantly, with mobile browsers etc., the number and variety of browsers is rising all the time
The number of indexed pages on the Web is:

175647683452

See [wikipedia: WWW Statistics](https://en.wikipedia.org/wiki/WWW_statistics), but a Google search shows that there is a lot of dispute about this number.
The Need for Uniformity

There is a simple rule which has taken a very long time to emerge, and people *still* don't get it.

A visitor wants their browser to display every page exactly as the author intended.

An author wants their page to be displayed exactly as intended by every browser.

So the Web is *almost useless* unless *almost every* browser displays *almost every* page *identically*.
Testing or Standards?

To make sure all browsers and pages work properly together, could you use testing?

No: it is absurd to test a browser on every page and very impractical to test a page on every browser.

The only reasonable solution is to use standards.

The Web does not work unless there is a standard way of writing web pages, and a standard way for a browser to display web pages.
Testing a browser on every web page sounds absurd

But the Web is full of non-standard legacy web pages, and a browser writer need some confidence that the browser will handle them properly

The Chrome browser uses a huge database of sample pages for regression testing, with automated checking of the display, pixel for pixel, with what is expected

So testing a browser on all web pages is not quite as absurd as it sounds
Testing a new web page on 100 browsers sounds very impractical, especially as several platforms are involved.

But, for critical web sites, you need to be sure that all 100 browsers display the pages properly.

Web sites like browsershots provide a service where you can submit a page and get a snapshot of the display of that page from many browsers.

But the service takes an appreciable time to produce its results, and the results are images that need to be checked one-by-one by eye.
Open Standards

Open standards (at [W3C](https://w3.org)) are the only kind that work for the Web, and they are, on the whole:

- freely available (no cost or fuss)
- precise (and platform independent)
- incorruptible (no political motives)
- up to date (changing at the right pace)
- enforceable (by market power of users)
- well supported (by non-profit organisations)

ℹ️ **poor standards**  ℹ️ **features**  ℹ️ **enforcement**
Poor Standards

A standard could be enforced by law *but no law applies internationally to the Web*

A standard could be "de facto", i.e. decided by the market winner or agreed on by the major players, *but this leads to politically or commercially driven, technically poor, standards*

A standard could be endorsed by the International Standards Organisation or similar official body *but nobody takes any notice of ISO standards because they move too slowly and are not freely available*
A browser *ought* to obey the standards. Can it have extra features? Toolbars, tabs etc are not important, but extensions to HTML etc. are. The answer is yes, because that is the only way new things can develop.

But every so often, there must be a convergence between all browsers to a new version of the standards

Authors shouldn't use proprietary (browser-dependent) extensions for essential public pages

Authoring tools should have extensions switched off by default (to help authors stick to the rule)
Enforcement

There is still a big enforcement problem, e.g.:

comparison of web browsers

And that's just a quick summary!

It shows an incredible variation in *which* features browsers support

Detailed comparison sites illustrate *to what extent* different browsers support particular technologies

*Browser differences are the biggest nightmare which web developers face*
Open Source Software

The Web benefits enormously from the open source software movement, look it up in google and wikipedia if it is new to you

Open software has the same advantages as open standards, and it can be of exceptionally high quality if it has a "critical mass" of support

There are really good open products in almost every area of web work
Proprietary software can be OK, and is used extensively in industry, but there are several dangers:

- the lock-in effect
- the gloss effect
- the choice effect
- the advert effect
- the trust issue
- edge cases
The lock-in effect is where, once money has been paid for a product, it becomes nearly impossible to switch to anything different.

It is 'only' psychological, but it is very powerful.
The **gloss effect** is where a lot of effort is put in to make the software look good and easy to use for non-experts, but under the surface there is often quite poor handling of more technical issues, especially poor attention to standards

Summary: glossy on the outside, grotty on the inside
The choice effect

With open source products, you can download 10 products, check them out properly rather than relying on reviews by people who don't know what they are talking about, and choose the most suitable.

It is usually impossible to do that with proprietary software, without wasting a lot of money.
The *advert effect* is where a product gets a reputation for being the 'standard' choice, but in fact lags behind the open source equivalent, which may have gained more volunteer development support than a single company could afford.
The **trust issue** is where you can't be sure that a proprietary product is safe, e.g. it might gather data about you or push adverts at you, without your authorisation. That's why Ubuntu Linux insists that everything it installs automatically is open source, because otherwise it can't offer any guarantees. It doesn't stop you from installing e.g. a proprietary driver for your graphics card, or the full of version of Chrome or Firefox which contain proprietary components, but you have to do it explicitly yourself, so that it is clear that it is you taking the risk, and not Ubuntu.
Edge cases

There are a lot of edge cases in this area

For example proprietary products may have valuable free cut-down versions

There are cases where the open source equivalent hasn't caught up with the proprietary version yet

So, after this unit, you have to make up your own mind.
Open source tools

For this unit, you must use the right standards, and only use open source software:

<table>
<thead>
<tr>
<th>Proprietary</th>
<th>Open Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet Explorer</td>
<td>Chrome</td>
</tr>
<tr>
<td>Dreamweaver</td>
<td>several</td>
</tr>
<tr>
<td>Illustrator</td>
<td>Inkscape</td>
</tr>
<tr>
<td>Photoshop</td>
<td>Gimp</td>
</tr>
</tbody>
</table>

The main exceptions are Windows/MacOS for general development, and Internet Explorer as a secondary browser for testing.
Web Site Design

There is not enough time in this course to talk extensively about good web site design.

Web site design is based on a few obvious principles which are ignored by 99% of web sites:

- **make sure you know who is going to maintain it**
- **avoid well known bad practices**
- **pay attention to accessibility, and ARIA**
- **be clear what your aims are**
- **put yourself in the shoes of your visitors**
Design Links

top ten mistakes (and see others in the series)

- maintenance
- site aims
- visitors
- assumptions
- fast download
- clear links
- morality
- features
A web site that doesn't change will quickly die; stale information is *much worse* than no information.

It is common for an organisation to hire a contractor to build a site, which lasts a few months maybe, and then goes stale, because the organisation hasn't thought about maintenance.

It is common for a web developer to offer to help someone to start up a web site, only to find that they are continually pestered ever after with questions and requests, because they didn't ask the right questions about maintenance at the start.
Think carefully about your web site's aims - does it provide information, sell products, or what?

How will visitors navigate the site and find what they want? Will they want to see advertisements? (No.) Will they want to fill in forms? (No.) Will they want to be slowed down by images or videos or sound bites? (No.) Will they want the site to be complex or pretty? (No, provided it works well.) How can they compare with other sites or order or pay really easily?
Visitors

Three things about visitors stand out straight away

Putting yourself in their shoes is difficult so get other people to criticise your prototype site

Visitors use Google to find sites, so check some obvious queries to see if your site appears on the first page of results

Visitors surf quickly, so make sure you don't slow people down as they surf through your pages
Assumptions

You are never designing a site for yourself, or for people like you, but for a wide audience, and you just can't make any assumptions about the people visiting your site, you have to do your best to please everybody.
Interests

Avoid assuming that visitors are particular kinds of people, e.g. Creative Kevs or Technical Tesses.

Also, people may have either an expert or a casual interest in the subject matter of your site (and we are all expert in some areas and have only a casual interest in others).
You have a moral (and legal) obligation to make your site **accessible**, which means available to people with disabilities such as blindness, or just people with low-power hand-held devices
Screen size

Screen sizes vary from mobile devices up to wall-mounted screens.

Screen sizes such as "17 inch" are a completely non-standardized diagonal measurement where every manufacturer means something different.
A *pixel* is a dot on a screen, and pixels are measured in

\[ ppi = \text{pixels per inch} \quad \text{or} \quad mpp = \text{mm per pixel} \]

On normal monitors, pixel sizes can be 50 to 100 ppi (0.51 to 0.25 mpp) and this (and eyesight) affects people's size preferences.

Sometimes it is given in dpi = dots per inch, but beware, especially with printers, this may refer to the number of primary-colour dots, 3 or 4 per pixel.
The number of pixels on a screen, width x height, is usually called *screen resolution*

Common ones are 320x240, 800x600, 1024x768, 1280x1024, 1280x800 (phone ... wide screen)

The *aspect ratio* (width to height) always used to be 4x3, but can now be 3x4 or 5x4 or 8x5

Advice: set the resolution in your operating system to match the maximum your screen will support, and check that circles look round!
Old browsers are a pain, surely we can suggest to people that they upgrade.

No, many people only know how to upgrade by going back to the shop.

I suppose telling them off for an old browser, even politely, is condescending and insulting.

Yes, and so are IE-only sites. You can't support all browsers, but there is no point in saying anything to visitors, just do your best.
I use my browser maximised – landscape style

I use a half-width portrait window to compare, or copy-and-paste between, windows

Different platforms encourage different conventions, too:

Windows (personal and office, full screen windows), Macs (graphic design and style fans, full screen windows), Linux (servers and technical, but improving for personal or office, smaller multiple windows), Mobiles (tiny windows, different UI)
Fast download doesn't matter now that everyone has broadband access, does it?

After a very short time, people get used to faster surfing, then any page which takes twice as long as usual sticks out like a sore thumb and irritates people, especially if they just want to surf past it, so a top site design rule is still

*Make sure your pages download fast*
Clear Links

Visitors who are surfing past don't read text, so *make links prominent and self-describing*

This paragraph is a bad example because it refers to the Computer Science main page [here](#) with the link in the middle of the text where people miss it.

This paragraph is good because it refers to *Computer Science*

with the link spaced out and indented and with a title which tells people what they want to know.
Technology can be good or bad, e.g. bitcoins improve privacy, but are used to launder drug money, social media sites are popular, but are heavily infected with hidden adverts, scams and lies.

Many sites join ad schemes which make a small profit but you then have no control over the ads appearing on your page, e.g. one may contain a fake download button which gets confused with yours, or worse.

There isn't even a financial advantage from the ads if they reduce your reputation and therefore harm your main business.
Web pages can contain:

- text
- images
- tables
- links
- forms
- boxes
- video
- audio
- canvas
- WebGL
- capture

**jQuery Demos**
**HTML5 Rocks**
**WebGL**
The coursework is a single open-ended project, to be done in pairs, with preliminary submissions for feedback that doesn't count, and a final submission worth 100%

Both people in a pair get the same mark (UG/MSc pairs, COMS32500/COMSM0104 pairs, singles are allowed, and triples if you ask)

If a pair breaks up, each submits individually saying what was done together and what was added alone

The majority (60%) of the marks are self-estimated
Coursework rules

- both members of a pair must learn everything
- you must cover all the subject areas
- the server must be written in JavaScript (with exceptions)
- any frameworks or tools can be used
- publishing (with care) is entirely optional
- it is OK to develop commercial, organisational or charity sites
The biggest mistake that too many people make with the coursework is leaving it too late.

It isn't difficult, but the breadth means it takes time.

You *cannot* afford to wait until the lectures reach each topic; you *must* read ahead, and ask if you need help.

There are no lab sessions (because there are no lab helpers who can cover the breadth) – just visit my office.
Ideally, web pages should be developed and tested on the publishing site, but publishing sites have awkward rules, so you should *develop using a local server*

Do the exercise (individually) *now* to see why

First you need to choose some tools:

- *editors*
- *IDEs*
- *browsers*
- *IE*
- *tools*

Then you need to find a strategy for writing and testing web pages – *don't start without doing this*
Don't use a word processor (Word, WordPad, ...). You would have to remember to export as plain text every time you saved the file, to avoid having illegal formatting characters in the text. Never use an HTML export feature, in Word or any other tool! It produces terrible HTML.

Don't (regularly) use an HTML editor in this unit - the structure of your HTML tends to deteriorate, and you will probably learn nothing.

Using a plain text editor such as gedit or atom is fine, though something more web-aware like Brackets is best.
IDEs

You could use an IDE, especially if you do a lot of server-side programming. My favourite is IntelliJ, but there are others. Getting an IDE set up to your liking, and working out the structure that the IDE uses, and understanding which issues are general and which are just IDE conventions, is complex. Also, using an IDE tends to reduce your productivity, unless you use it all day every day. So my advice would be not to use an IDE until you are very confident and understand all the issues – learn everything first.
Browsers

It is important to choose a browser *for development* which supports modern standards, and which has good developer tools available.

You should use a mainstream browser, not a clever minority browser, because much more maintenance effort goes into mainstream browsers.

So, in priority order, choose Chrome or Firefox or Safari or Opera.

On Linux, install full Chrome to replace Chromium, and/or full Firefox to replace the cut-down Firefox.
Internet Explorer

Internet Explorer (IE/Edge) causes the most problems because old versions are still being used, whereas old versions of other browsers are not

IE stagnated for many years, then leapt forward from IE9 onward, but it is still the least standard

You shouldn't use it as your main development browser, but you should use it as a secondary testing browser, especially if you want to support old browsers
You need to become familiar with the developer tools in your browser

In most big-name browsers, the tools are built in - in Chrome or Firefox, right click on the page and choose "Inspect Element"

You can debug HTML, CSS, JS, test efficiency, and check network request and response headers (e.g. content type, character encoding, redirections, caching, refreshing)
Node

To develop pages, you **must** use the node server I am providing.

And later, for the server side, you will need node.

So install node (with npm) and learn to use it **now**.

Install it from [nodejs.org](http://nodejs.org)
You **should** learn to use [Version Control](https://git-scm.com) (git plus a cloud service) **now** to share and/or backup your work.

That means learning to use **git** and one of the cloud services **gitlab**/**github**/**bitbucket**.

I prefer **gitlab** for private repos and **github** for public ones, but use whichever you like (or all) and accept any free gifts you get.
I suggest this (notional) site structure for everybody, whether planning a raw or express-based server.
Node modules shouldn't be included when you submit, or included in a git repository. That's because some modules are platform dependent, so each person must install them separately.

Either put `node_modules` in an outer folder, or exclude it from the repository and zip files.

Then when you type `npm install xxx` the module `xxx` will be installed there. `require("xxx")` will work in the site folder.
If you create a `package.json` file (with `npm init`) and maintain it (with `npm install xxx -save`) then you need to keep that in the same folder as `node_modules`.

Including it in the zip file can be helpful for marking, but don't include `node_modules`.
Use **git** and an online service (gitlab or github) for backup, versioning and sharing

Either put your repository outside **site** or exclude it from zip files (it makes submissions too bulky)

Use a **.gitignore** file to exclude anything such as **node_modules** that shouldn't go in your repository
The site folder is the one you are going to zip up and submit.

It needs to contain everything you have written yourself, and everything you have downloaded.

But when you submit it for marking, it should not contain .git or node_modules and if you are working on a Mac, it is polite to get rid of .DS_Store folders too.
Using a command line version of zip gives you more control than using a GUI zipper.

With a GUI zipper, even if you can work out how to exclude things from the zip file, you have to remember to do it every time - you can't automate it.

A good zip command is:

```
zip -r site.zip site -x *.git* *node_modules* *.DS_Store*
```
The public folder (also sometimes called static) contains all the 'public static' files, i.e. the files visible directly to the browser (not necessarily to the general public) without being manipulated by the server first.

Other files in the site folder, e.g. server.js, are part of the site but not directly accessible to the browser.

The folder site/public is effectively the root of the site - the word public does not appear in URLs.
The file `server.js` (sometimes called `app.js` or `index.js`) is initially a provided server, but you will later replace it with your own.

The server is either 'raw' (written without any framework) or 'express' (based on `express` and other modules from the express framework).
The *index.html* file is the 'home page' of your website.

There is a convention that if you type in a URL which ends with a slash, representing a folder (e.g. `http://localhost:8080/` representing `.../site/public/`), then it refers to the file `index.html` within the folder.

There are other conventional names, but you should stick to this one.
Correct pages

It is **absolutely essential** that your HTML is correct.

If your web pages are not correct, you have **no chance** of making sure they work in all browsers.

Just viewing a page to see if it looks OK is **useless**.

There are only two known approaches to this:

- use a server which delivers pages as XHTML
- use a validation tool
XHTML is HTML with some tighter rules to make it simpler, and to make it XML compatible.

The main difference: you can't leave out any close tags.

Opinions differ - I prefer XHTML because otherwise, to understand my own HTML properly, I have to understand the complex rules about what happens when close tags are left out.

Others, especially amateurs, like the looseness of HTML.

But HTML **must** be 100% precise, like programming.
If a server *delivers* a page as XHTML (as well as it being written in XHTML), the browser gives error messages. It gives instant feedback if you leave out any tags.

Technically, it only checks *well-formedness* (tags nested correctly) not *validity* (tags used correctly) to allow for future development of the standards, so a validation tool should occasionally be used as well.

There are problems using it with some frameworks, but they can usually be overcome with a bit of grit.
A validation tool does full validity checking

Online validators, and browser tools which use online validation, are unsatisfactory because

• either you have to publish pages on a non-secure site (risking public access)
• or you have to upload pages as files (very inconvenient, especially if using a client-side framework)

A validating editor would work, but there aren't any good ones (2016)
The best open source validation tool is `vnu.jar` which can be extracted from a zip file downloaded from [github.com/validator/validator/tree/17.1.0](https://github.com/validator/validator/tree/17.1.0).

You use it by typing e.g.
```
java -jar vnu.jar index.html
```

The main problem with using a tool like this is the enormous psychological pressure to 'forget' or not to bother after editing, because it is not fully automatic, so you should create an automatic script to run it.
A development server is provided for you

Install an *up-to-date* version of Node, put the file in your site folder, run `node server` or `nodejs server`, then visit `http://localhost:8080`/ 

The *development* exercise covers the details

Beware: 🚸 security risks 🚸 localhost 🚸 caching
A server runs with your permissions and is used by other people, so think about security before you run it.

- You can limit what the server can do
- You can use a firewall-protected port
- You can encrypt the network traffic
- You can use some form of authentication

If a server just delivers pages, can a hacker (a) read a private (unlinked) file inside your site or (b) access a file outside your site?

By default, the provided server only accepts requests from a browser running on the same computer.
If you run your own web server, you can use localhost in URLs instead of the true computer name to mean "this computer"

It is handy if you move between workstations in the lab

It only works if the server and browser are running on the same computer

So don't use localhost in actual links on pages
If you update/republish/resubmit a new version of a web page or style sheet or script file, and then view it in your browser, you may not see the change

That may be because your browser is using an old cached version of the file(s)

Some browsers are not good at this, especially with style and script files

Try refresh or reload, or a more forceful refresh such as F5 or CTRL/F5 or whatever, or in desperation try clearing your browser's cache
The provided server handles the first three of these issues; the first two are to help with marking:

- **Delivering XHTML**
- **Case sensitivity of URLs and filenames**
- **Same origin policy**
- **URL validation**
- **Safe URLs**
- **Running on a secure site**
- **Extensions**
- **Index pages**
- **Location sensitive links**
Most servers deliver pages as HTML.

To test your pages properly while developing them, they must be delivered as XHTML, so that you see error messages for incorrect pages.

That means telling the browser that pages have content type `application/xhtml+xml`, not the usual `text/html`.

This *cannot* be done just by writing pages as XHTML – the server must add a Content-Type header on delivery.
Case sensitivity

The URL standard is terrible: it says URLs are case sensitive, but a server may treat both cases as equivalent. That means servers can (and do) pass the case sensitivity issue on to their file systems:

- servers running on Linux are case sensitive
- servers running on Windows are not

This is *intolerable* for marking - your submitted site must work on Windows and Linux.

The provided server converts URLs to lower case, then makes sure all your file names are lower case.
The same-origin policy

In Javascript programs, you often want to access other files besides the page that the script is running on.

For security, browsers insist that those files are on the same site as the page.

Also for security, browsers often prevent access when a page is being visited directly via a file: URL (otherwise hackers could offer you a page for download and tell you to visit it with your browser, and it would then have access to your whole file system).

The provided server is a genuine site, so that scripts work properly.
The provided server doesn't do URL validation, which is essential for a public site to avoid security loopholes.

URL validation means checking that a URL starts with / and doesn't contain // or ./ or ../../../

Express (and every other server framework in every language, as far as I can find out) fails to validate the URL properly, particularly not checking for //, so has a potential security loophole.
Safe URLs

As well as invalid URLs, there are other possible URL security issues

Spaces should be banned (they are technically allowed if escaped, but you can't expect non-experts to understand escaping, so it is best to avoid them)

Non-ascii characters should be banned for the same reason, and because there are security risks associated with them

URLs which are too long should be banned, because of security risks
If a page is published on a secure site, and there are links to external resources which are not secure, many browsers will give error messages, so for example you can link to a resource using any of

http://ajax.googleapis.com/...
https://ajax.googleapis.com/...
//ajax.googleapis.com/...

The first link will cause error messages on https, the second can cause problems on http, the third links to the appropriate one and is the best option
All files delivered on a web site should have extensions, because:

- browsers use the extension to decide what to do
- a server can't easily tell whether a URL like .../readme refers to a file or a folder

The simplest thing to do is to ban such a URL (if you treat it as a folder, you must redirect the browser to .../readme/, not deliver index.html, otherwise the browser thinks it is in the wrong folder)
Index pages

It is common for URLs like .../folder/ to refer automatically to a default index page such as .../folder/index.html

But the default *name* of the index page differs between sites

I suggest you stick with *index.html*
Location sensitivity

You could link to an image with any of:

```
file:///.../images/pic.png
/images/pic.png
../images/pic.png
```

The first obviously won't work when you publish, so never use it.

The second won't work if your site gets published as a sub-site of some other site, and the third won't work if you move folders around inside your own site.