XML is a popular multi-language system, and XHTML depends on it.

XML details languages
Many of the newer standards, including XHTML, are based on *XML = Extensible Markup Language*, so we will study that first.

XML is not really a language

Well, what is it then?

It is a system for inventing new languages
<course>
  <title>Web Technologies</title>
  <part>
    <title>Client Side</title>
    <chapter>
      <title>Overview</title>
      <slide>
        <title>Aims</title>
        <para>The aims are...</para>
        ...
      </slide>
      ...
    </chapter>
  </part>
</course>
I think I get it, but what good is it?

It is a compromise between human reading and writing, and processing by program.

Taking the human side first, you can store it in a plain text file and use an ordinary editor.

The downsides are that it is a bit verbose, and it has to be 100% precise, like a programming language.
That doesn't sound too good yet, what are the advantages on the programming side?

You don't have to write any new software. Usually, when you invent a new computer language, you have to write a parser, at least

With XML, there are already standard tools you can use to process your new language

And there are libraries simple enough for every day programming, small enough to fit in a credit card
A program reads XML into the simplest structure known for representing complex data, a tree, always drawn in computing with the root at the top!
A 'family tree' style is not convenient for display or interaction, so a 'navigation' style is usually used.
OK, so it's great for programs, but what about this human authoring problem?

You can use a standard XML editor for any XML-based language.

Most programmer's editors support XML directly, or with a plugin.
Advantages of XML

OK, I think I get the message, but I want to understand what the advantages of XML are.

There is a lot of hype about XML, but here are some genuine advantages:

- Stable
- Declarative
- Stylable
- Generic
- Interoperable
- Programmable
- Modular
XML is Stable

XML is defined by a simple, stable, open standard with test suites to make sure implementers agree.

This allows the long term storage of data, and avoids lock-in to proprietary or imprecise or under-documented tools or formats.

For instance, the storage formats for Open Office are in XML, and are much more stable than the storage formats for other office software (old versions of which are already effectively unreadable).
XML is Declarative

To *declare* means to *say what you mean* and XML encourages you to say what you want done, and not how to do it, e.g. `<h2>Heading</h2>` means *form a top level heading*, instead of *change font, change size, ... change back again*

The more you store the original intention and not low level detail, the more the tools can do for you

XML is a prime technology in the "Semantic Web" movement, which is trying to make web sites searchable and processable according to the meaning of their content and not their layout
Being declarative also means separating style from content, and XML stores just meaningful content.

Then the style, i.e. how data is displayed, can be added as re-usable style sheets in CSS or XSL.

This is encouraging the single source idea, where a document is stored once in XML (or translated into XML on the fly from databases), and then converted into web pages, or PDF or other formats.

One style sheet is used for each format, with each being used for all the documents in an archive.
XML is Generic

XML doesn't make any assumptions about what it is going to be used for, so it supports the widest possible variety of languages and applications.

In particular, it is international, with no English keywords, or biases towards Western cultures, it supports full Unicode, and strongly encourages the robust transport of text in every human language.

Although most applications concentrate on data, programs are also supported, in a declarative style, e.g. XSLT for transforming any XML document into another, or SMIL for animating any XML document.
XML is designed to be transported from tool to tool and program to program, as files, or as web data, or as bytestreams travelling over networks.

It is not biased towards any platform or media.

This has promoted the development of web services where XML acts as a portable interface between programs written in different languages and running on different platforms (providing a much looser binding between software modules than object models such as CORBA).
As well as being easy to read into a program and write out from a program, XML comes with a **DOM** = **Document Object Model**

This specifies how the document is stored in memory so that languages (XSL for stylesheet, JavaScript for scripting, SMIL for animation, Java for programming) can cooperate in accessing and modifying a document dynamically online.

In addition, the document, as text or DOM, can be inspected by human eye in order to find out what is going on behind the scenes, with nothing hidden.
XML is Modular

Different XML languages can be combined

In the past, web pages had separate bits in different formats, which don't integrate or interact

These days, a web page, or any data, can consist of an interwoven mixture of languages in an overall single XML framework, so that, for example, stylesheets and animations can be applied to everything
We will look at some of the details and jargon of XML.

XML is just text, but the `<` and `>` characters are special, surrounding case-sensitive *tag* names.

A *start tag* and *end tag* surround the *content* of an *element*, and elements nest to form a tree.

```
<slide>
  <title>Conventions</title>
  <para>These pages are...</para>
</slide>
```
Another way to make e.g. a slide title is to use an attribute with a value between single/double quotes

```xml
<slide title="Conventions">
  <para>These pages are...</para>
</slide>
```

This works because there is only one title, there is no issue with position in the slide, it is just one line of simple text. A language can often use either an element or an attribute, but an attribute is simpler to use, e.g. by stylesheets or animations
XML comments look like this 🔄 _beware minuses_

```xml
<!-- comment -->
```

An element with no content can be abbreviated 🔄 _space_

```xml
<title/> = <title></title>
```

You can insert raw text containing `<` and `>` etc. (see the polyglot standard to use it safely)

```xml
<![CDATA[ text with & or < or > in it ]]> 
```
Minuses in comments

An XML comment has double minuses near the start and end

```
<!-- comment -->
```

For historical reasons (SGML + old browsers) you should avoid having a double minus anywhere in between
Many authors write an empty element like this:

```xml
<title /></title>
```

This space was needed in XHTML to fool old browsers like Netscape 4, but those browsers are obsolete.

Some people may prefer the version with the space, as an added reminder that it is different from `</title>`.

It is a matter of taste.
To help make it international, XML uses *unicode*, which gives standard numbers (0 to 1114111) to characters in all languages (see *history of characters*).

An *encoding* specifies how to represent (some) unicode numbers as bytes, but the default XML one is *UTF-8*.

A UTF-8 file may be *unreadable* unless (a) you use tools which understand UTF-8 and (b) your computer has suitable *fonts* installed.
Characters

Characters include letters (ABC...abc...), digits (01234...), punctuation (! ? . , : ; " ' ...), operators (+− */%<>...) and controls (newline, tab, ...)

A and a are different characters, but A, A, A, A, A, A, A, A, A, A are all the same character, i.e. fonts, sizes and styles don't count

There are characters for Latin (e.g. English), Greek, Cyrillic (e.g. Russian), Hebrew, Arabic, Chinese, Japanese and so on, and it is very interesting to skim through the PDF list of all characters
The commonest encoding is \textit{plain text} = \textit{ASCII} where characters 0 to 127 (roughly the ones on an English keyboard) are stored in one byte each.

There are many incompatible \textit{national encodings} which extended plain text using 128 to 255 to represent one language group, e.g. Latin-1 = ISO-8859-1 for Western Europe or Cyrillic = ISO-8859-5 for Russian etc. You can still use them, but only by negotiation, not by guarantee.

The \textit{UTF-8} encoding covers the whole of unicode.
UTF-8 is compact and compatible with plain text, but covers the entire Unicode character set.

Each character is represented using 1, 2, 3 or 4 bytes, and single bytes with first bit 0 represent characters 0 to 127, as with plain text.

Bytes starting 110, 1110, 11110 are the first bytes of 2- or 3- or 4-byte characters.

A byte starting with 10 is a continuation byte carrying the next 6 bits of a character.
A **font** is a table of images, or drawing instructions, usually in the old 256-character national language style.

Different fonts for the same language represent different sizes, styles (grouped into **serif**, **sans-serif**, **monospace**, ...) and variations (bold, italic, ...)

UTF-8 tools use many fonts, but *only* if your computer has them installed; there are *no* fonts common to all major platforms !!!!!

If the exact font matters on your web site, you have to use web fonts, as we will see later when we look at CSS.
An XML entity is a 'variable name' between & and ; which stands for some other text (like a 'macro')

&<gt; &amp; &apos; &quot; are built in and stand for < > & ' " so that these can be inserted without their usual special meanings

XML also has an alternative to UTF-8 where, for example π (Greek pi) is written &amp;#960; (character 960) or &amp;#x3c0; (3C0 in hex, decimal hex) allowing everything to be plain text (don't memorise, look up, e.g. type unicode pi into Google)
Decimal or Hex?

Should you use decimal or hex for character codes?

Personally, I think it is completely absurd to force anyone, even programmers or computer scientists, to use binary or hex unless they happen to be working at the bit manipulation level (e.g. machine instructions or compression or protocols)

There are no normal occasions in which the bit pattern of a character code is of any interest or importance

Therefore I strongly prefer decimal, but unfortunately the world doesn't agree with me
Hex, short for hexadecimal, is base 16, a "compact form of binary", and is used more than necessary.

The 16 digits are 0123456789abcdef, so $3c0_{16}$ means $3 \times 16 \times 16 + 12 \times 16 + 0 = 960$

Each digit in hex matches up with 4 digits in binary, e.g. $0011\ 1100\ 0000_2 = 960$

Plain text has no subscripts, so there are different ways to show that a number is in hex, e.g. `\x3c0` in XML, `0x3c0` in C, `\u03c0` in Java.
**Newlines** in text files have never been standardized, with several common "reasonable" conventions, e.g.:

<table>
<thead>
<tr>
<th>System</th>
<th>Newline Characters</th>
<th>Equivalent Characters</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Old Mac</td>
<td>\r = #13 = ^M = CR = Carriage Return = Enter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unix</td>
<td>\n = #10 = ^J = LF = Line Feed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Windows</td>
<td>\r\n = #13#10 = CRLF = ^M^J (2 characters)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

All are standard: CR for keyboard input, LF for programs, and CRLF for network transmission.

What is not reasonable is the lack of agreement, but the XML fix is that all programs must accept all conventions (and convert internally to LF) ✅ **separate or terminate?**
Separate or Terminate?

Should newlines separate or terminate, i.e. should there be a newline at the end of the last line of a file?

Theoretically, there is no way to distinguish an empty file from a file with one blank line unless newlines are terminators.

Practically, the majority of software expects terminators.

Historically for output, and today for input, the last line needs a terminating newline.

So, make sure your editor adds a final newline when there isn't one.
Versions of XML

An XML file can declare its version with, e.g.

```xml
<?xml version="1.0" encoding="UTF-8"?>
```

Versions 1.0 and 1.1 are nearly the same: when writing, use version 1.0 for interoperability until 1.1 becomes common; when reading, accept both.

The default encoding is UTF-8 (recommended), and the whole line can sometimes be left out, but if it is there, `<?xml` must be the first characters in the file (to allow encoding recognition).
XML Languages

An XML language is described by specifying what tag names are allowed, whether an element contains text or inner elements, what tags can appear in the text, what inner elements are allowed, what attributes are allowed in the tags, and what values the attributes can take.

In a simple language (e.g. XHTML4) what is allowed in any context is determined just by the parent element, but in more complex languages (e.g. XHTML5) it may depend on other ancestor elements.
There are three ways to specify which XML language a document uses. One way is to use a DOCTYPE (or DTD = Document Type Definition):

```xml
<?xml version="1.0" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN">
<html>
  . . . .
</html>
```

This is bad HTML according to the decisions we have made, so don't copy-and-paste this.
The second way to define the language is to use attributes referring to a *Schema*. (This information is taken from a [W3C document](http://www.w3.org).)

```xml
<?xml version="1.0" ?>
<html lang="en-GB" xml:lang="en-GB"
    xmlns="http://www.w3.org/1999/xhtml"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/1999/xhtml
    http://www.w3.org/2002/08/xhtml/xhtml1-strict.xsd">
    ...
</html>
```

Again, don't copy-and-paste this
The third approach to language definition is to ignore XML-based approaches and have a custom language definition, implemented by custom tools.

This is the approach taken by HTML5 (the language is defined by the standard, implemented by browsers, and is too complex to be handled by a Doctype or Schema).

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

The dummy Doctype is for old HTML4 browsers, the xmlns attribute is for XML-aware browsers and tools.
```
The **html** element should specify the main human language in a web page, for the benefit of tools like Google translate:

```html
<html xmlns="http://www.w3.org/1999/xhtml" lang="en-GB" xml:lang="en-GB">
</html>
```

The **lang** attribute is for old browsers, the **xml:lang** attribute is for XML-aware browsers and tools. **en-GB** is for my language, which is real English (not American English or anybody else's English).
Error Checking and Handling

A program can read XML without a Doctype or Schema, and check it is *well-formed*, i.e. the tags match and nest

**Or** the program can check against a Doctype or Schema or custom rules to make sure the XML is *valid*

Loose error handling was the biggest mistake of early HTML standards, causing tag soup, but XML tools, and browsers if pages are delivered as XHTML, *must* check if the XML is well-formed, and *must* stop and tell the user if not

ℹ️ *must/should*  ℹ️ *namespaces*
Believe it or not, there is a standard, RFC2119, which specifies the meanings of words like must and must and should in standards documents!

must is an absolute requirement, but for should: ...there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course

So read and understand the standard, and find out how it is used, before you bend the rules
Namespaces

An XML document can switch between text, drawings, diagrams, equations, tables etc. using namespaces (xmlns) to change languages, e.g. here's how to include an SVG drawing directly in XHTML5:

```html
<html xmlns="http://www.w3.org/1999/xhtml">
  <svg xmlns="http://www.w3.org/2000/svg"
       width="100px" height="100px">
    <rect />
  </svg>
</html>
```