Extensible Markup Language

XML is quite simple, we will cover it quickly.
Most of our time will be spent on XSL, the stylesheet equivalent, which is much more complicated than CSS.

Background

HTML is not Extensible

In HTML we can't define things like poem or verse or line, for example – we can't define whatever we want. We instead have to adapt existing structures, e.g. `<h1>` and `<ol>`.

If we do, how would other people know what conventions we used? We want a way of defining purpose-specific tags.

SGML

There is an older language SGML (Standard Generalised Markup Language), which was invented by IBM and used for marking up electronic documents. SGML is a meta-language, a language for describing other languages.

It is used to define markup languages for specific applications, for example: poetry, legal documents.

HTML was defined using SGML.

Why not SGML?

- SGML is more complex than is needed.
- Many of its features are only rarely used.
- XML is a version of SGML which omits rarely-used features and simplifies the tasks of:
    - defining document types
    - writing programs which process documents
- Software for processing SGML can still be used to process XML files.

XML

XML is a meta-language, a simpler version of SGML, adequate for web purposes.

With XML you can:

- define new tags
- use these to define platform-independent languages and data structures

With XML **you can't specify rendering** (appearance e.g. italics, bold). To control appearance there are other technologies.

What is XML Good For?

- Defining new markup languages
- Data-exchange
    - Companies need to exchange electronic data with one another.
    - XML allows you to use HTTP connections instead of dedicated channels.

XML

Defining New Tags

We might define `<poem>`, `<title>`, `<poet>`, `<verse>`, `<line>` tags in order to mark up poems.

Even though `<title>` already exists in some markup languages, we will see ways to prevent problems with that.

Example: Marking Up Rhyme

We might define a **rhyme** attribute for the `<line>` tag:

```
<line rhyme="owd">I wandered lonely as a cloud</line>
```

Alternatively, we might define a pair of new tags `<rhyme>` and `</rhyme>`.

```
<line>I wandered lonely as a cloud <rhyme>owd</rhyme></line>
```

We use other technology to control rendering, which we could use so we don't print the content of rhyme tags, or print it in a special way.

A third way would be use `<rhyme>` tags with a **sound** attribute:

```
<line>I wandered lonely as a <rhyme sound="owd">cloud</rhyme></line>
```

Fourth option – define the tag `<sound>` for tagging a child element of a `<rhyme>` tag:

```
<line>I wandered lonely as a <rhyme><sound>owd</sound>cloud</rhyme></line>
```

Whether you use attributes or tags is generally a matter of taste, though sometimes one may be better.

Document Type Definitions and Schemata

There are many ways to markup whatever you want to markup. Once we've invented a new set of tags and attributes, XML provides us with a way of documenting the relationship between these tags/attributes so that other people can understand our approach.

XML provides two ways:

- The original way is to write a **Document Type Definition (DTD)**
- A newer way is to write an **XML Schema**

If we publish our DTD or schema, we are publishing a new XML-based language that others can use.

Tags in XML

XML elements are tagged like HTML elements, but there are some differences:

- XML tags are case-sensitive.
    - `<Verse>` is different from `<VERSE>`.
- Every start tag must have a corresponding closing tag.
- This implies that XML tags must be nested properly.
- XML also allows empty tags (a start tag which does not have a corresponding closing tag):
    - Empty tags must be written like this: `<someEmptyTag/>`, which is regarded as shorthand for `<someEmptyTag></someEmptyTag>`.

Content of XML Elements

The content is what lies between an element's start and closing tags.

This `<trustworthy>` element has "empty content":

```
<trustworthy/>
```

These `<name>` and `<age>` content have "character content":

```
<name>Fred</name>
<age>25</age>
```

This `<person>` element has "element content":

```
<person>
    <age>25</age>
    <name>Fred</name>
    <trustworthy/>
</person>
```

This `<person>` element has "mixed content":

```
<person>
    Fred
    <age>25</age>
    <trustworthy/>
</person>
```

Attributes in XML

- As in HTML, start tags can have attributes.
- The value of every attribute in XML must be quote, even if it is a number:
    - `<person age="25"></person>`

Comments in XML

Comments are the same as in HTML.

```
<!-- This is a comment -->
```

XML Version Declarations (check this slide)

The first line of an XML document *may* define the XML version.

```
<?xml version="1.0"?>
```

Syntactic Acceptability of an XML Document

XML documents can have 1 of 3 different levels of syntactic acceptability. (We're used to two in Python: acceptable or unacceptable.)

There are three:

- Not acceptable – syntactically **ill-formed**
- Obeys basic rules of XML – syntactically **well-formed**
- Obeys syntax rules of a special language based on XML – syntactically **valid**

Well-formed XML Documents (check this slide)

- The document contains exactly one root element
- All other elements are descendents of the root element
- start and closing tags for an element have the same spelling
- the start tag …

Valid XML Documents

- Must be well-formed.
- Must satisfy the syntax rules for an application-specific language based on XML.

A valid XML document must declare the XML-based language according to whose rules the XML document is claimed to be valid (done using a document type declaration which must appear before the start tag of the root element)

Document type declaration general format: `<!DOCTYPE nameOfRootElement someDTDspec>`

someDTDspec would either:

- Directly contain an internal document type definition (DTD)
- Contain a reference to an external file that contains a DTD

DOCTYPEs Referring to External DTDs

Two formats:

- Same machine – `<!DOCTYPE nameOfRootElement SYSTEM "fileNameAndPath">`
- Somewhere online – `<!DOCTYPE nameOfRootElement PUBLIC "fpi" "url">`

The *fpi* (Formal Public Identifier) gives the formal name of the organisation which produced the set of rules.

Document Type Definitions (DTDs)

A DTD defines the syntax rules for an aplication-specific type of XML documents.

It describes the root element that must appear in a valid document, and also describes the kinds of child elements that it may possess

It must contain **element type declarations**, statements with the following general format:

- `<!ELEMENT elementName contentModel>`

In addition, a DTD may contain a range of other types of statement, one of which is attribute declarations.

Element Type Declarations

An element type declaration has the following general format:

`<!ELEMENT elementName (contentModel)>`

Empty Elements

- `<!ELEMENT elementName (EMPTY)>`

Elements with Character Content

- `<!ELEMENT elementName (#PCDATA)>`

(#PCDATA stands for "parsed character data")

Elements with Element Content

- `<!ELEMENT elementName (child-name1, child-name2, …)>`

This specifies that every element following this format must contain child-name1, then child-name2, and so on.

Example:

- `<!ELEMENT memorandum (sender, receipient, message)>`

This specifies a memorandum element that must contain a sender element followed by a recipient element followed by a message element.

Full Declaration

When an element is declared to have element content, the children element types must also be declared:

```
<!ELEMENT memorandum (sender, recipient, message)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT recipient (#PCDATA)>
<!ELEMENT message(#PCDATA)>
```

However, children may also be optional or may occur more than once:

- `<!ELEMENT elementName (childName+)>`

The plus symbol means "one or more".

You can also define zero or more occurrences:

- `<!ELEMENT elementName (childName*)>`

And zero or one:

- `<!ELEMENT elementName (childName?)>`

Parentheses can be used to group sequences of child elements:

- `<!ELEMENT song (author?, verse, (chorus, verse)*)>`

An element can also have alternative children:

- `<!ELEMENT elementName (option1|option2|option3)>`

Alternatives can also be combined with the (+, *, ?) quantifiers:

- `<!ELEMENT elementName (father? mother? (male|female)*)>`
- This denotes a family that could contain a father, could contain a mother, could contain any number of children.

Elements with Mixed Content

Example:

- `<!ELEMENT person (#PCDATA|name)* >`
- This says a person element contains any amount of elements, each of which could be either #PCDATA or a name element.

Elements with Arbitrary Content

- `<!ELEMENT elementName (ANY)>`

Attribute Declarations

The attributes for an element are declared in a statement which has the following general format:

- `<!ATTLIST elementName attributeDefinition, attributeDefinition … >

Each Attribute Definition has the following format:

- `attributeName` `attributeType` `attributeDefault`
- The name, what value it can take, the default value.

Example:

```
<!ATTLIST person name  CDATA          #IMPLIED
                 id    ID             #REQUIRED
                 sex   (male|female)  "male">
```

This would be satisfied by these XML examples:

```
<person name="Fred" id="p001" sex="male">
<person id="p001">
```

Defaults

- `#REQUIRED` means that this attribute is required.
- `#IMPLIED` means that this attribute is optional.
- `"male"` means that if the sex isn't specified, the sex can be assumed to be male.

Attribute Types

- `CDATA` denotes a string-valued attribute that contains any character apart from `<`, `>`, `&`, `'`, `"`.
- `ID` is one of a set of types with predefined *validity constraints*, described later.
- `(male|female)` is an enumerated type, a type whose exact set of members is listed.

The ID type has the following requirements:

- An element can have only one attribute of type ID.
- The value of an ID attribute must start with a letter, underscore or colon which may be followed by a sequence containing any of these or digits.
- The value of an ID attribute must uniquely identify the element which bears it. No other element may have the same value for an ID attribute.
- An ID attribute must have a default of either `#REQUIRED` or `#IMPLIED`.