# 3. Web Languages

---



---

## 3.1. The World Wide Web

- The *World Wide Web* (*WWW*, or simply *Web*) is an information space accessible over the Internet
- items of interest, called *resources* are identified by global identifiers called *Uniform Resource Identifiers* (URIs), e.g., `http://www.dcs.bbk.ac.uk/~ptw/`
- conceived by Tim Berners-Lee and Robert Cailliau; launched at CERN in 1991:
  - using *HyperText Markup Language* (HTML) for representation of resources
  - using *HyperText Transfer Protocol* (HTTP) for transport over the Internet
- hypertext and markup languages already existed, but HTML was simpler

- Web now comprises billions of resources, many represented using HTML
- ongoing development of Web specifications is overseen by the *World Wide Web Consortium* (W3C)
- W3C *recommendations* are de facto Web standards

## 3.2. Web Languages

- there are many languages used for *representing information* on the Web
- we will consider only
  - HTML (just a short overview)
  - XML (in some detail)
  - XHTML (the "XML version" of HTML)
  - JSON (more concise representation for data)
- later on, we will also consider *programming languages* used on the Web, such as
  - Javascript (used in the browser)
  - PHP (used on the server)

## 3.3. HyperText Markup Language (HTML)

- document structure and hypertext specification language
- specified by the World Wide Web Consortium (W3C)
- latest *recommendation* is HTML5.2 (December 2017)
- previous *recommendation* was HTML 4.01 (December 1999)
- designed to specify *logical structure* of information
- intended for presentation as *Web pages*
- text marked up with *tags* defining document's logical units, e.g.
  - title
  - headings
  - paragraphs
  - lists
- displayed properties of logical units determined by browser (and stylesheet, if present)

## 3.4. Elements and tags

- the logical units of documents are called *elements*
- an element is delimited by a *start tag* and an *end tag*
- tags are delimited by `<` and `>` characters
- example HTML `head` element:

```
<head>
   <title>Elements and tags</title>
</head>
```

  - `<head>` is the *start tag* of the element
  - `</head>` is the *end tag* of the element
  - `head` is the *element name*
  - `<title>Elements and tags</title>` is the element *contents*
  - "Elements and tags" will be displayed in the title bar of the browser
- extra *whitespace characters* (e.g. space, tab, carriage return, line feed) are usually not significant

# 3.5. HTML Document Structure

- a document can be viewed as elements nested inside one another
- whole document is an `html` element, with a `head` element followed by a `body` element

```
<html>
    <head>
        <title>Document title</title>
    </head>
    <body>
        <h1>First level heading</h1>
        <p>
            First paragraph
        </p>
        <p>
            Second paragraph
        </p>
        ...
    </body>
</html>
```

- what is between < and > inclusive (i.e., in `red` on slides) is *markup*;
  what is between > and < exclusive (i.e., in `blue` on slides) is *character data*

# 3.6. Some Block-Level HTML Elements

| Element name | Description |
|---|---|
| `h1...h6` | headings |
| `p` | paragraph |
| `table` | table |
| `tr` | table row |
| `ol` | ordered list |
| `ul` | unordered list |
| `li` | list item |
| `form` | form to fill in (see later) |
| `hr` | horizontal rule |
| `pre` | preformatted text |
| `div` | division (often used for formatting with stylesheets) |

- block-level elements are displayed with a line break before and after them
- they form the larger structural components of a document
- these are called *flow* elements in HTML5

# 3.7. Some In-line HTML Elements

| Element name | Description |
|---|---|
| `code` | sample code |
| `em` | emphasis |
| `a` | anchor (for hyperlinks) |
| `img` | inline image |
| `td` | table data item |
| `span` | often used for formatting with stylesheets |

- in-line elements are displayed in the current line
- can generally only contain other in-line elements and text
- these are called *phrasing* elements in HTML5

# 3.8. Example

The HTML source code for the previous slide looks as follows:

```
<h1>Some In-line HTML Elements</h1>

<table>
  <tr>
    <th>Element name</th>
    <th>Description</th>
  </tr>
  <tr>
    <td><code>code</code></td>
    <td>sample code</td>
  </tr>
  ...
  <tr>
    <td><code>span</code></td>
    <td>often used for formatting with stylesheets</td>
  </tr>
</table>

<ul>
  <li>in-line elements are displayed in the current line</li>
  ...
</ul>
```

# 3.9. Attributes of Elements

- optional name-value pairs that are included in a start tag
  `<tagname attribute="value">`
- used to provide information associated with an element
- examples:
  `<a href="http://example.org/index.html">`
  `<img src="example.jpg" alt="An example image">`
- order of attributes is *not* significant, e.g.,
  `<img src="example.jpg" alt="An example image">`

is equivalent to

```
<img alt="An example image" src="example.jpg">
```

## 3.10. HTML, XHTML and XML

- we have in fact been covering *XHTML* (a W3C Recommendation)
- XHTML uses the syntax of *XML* (*Extensible Markup Language*)
- XML is a language for creating other languages, usually called *XML vocabularies*
- this is done using something called a *Document Type Definition* (*DTD*)
- so XHTML corresponds to a particular XML vocabulary or *document type*
- this is specified at the start of an XHTML document, e.g.:

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

  the above syntax will be explained later
- for HTML5, one can simply use

```
<!DOCTYPE html>
```

- HTML is essentially a syntactically less strict form of XHTML

## 3.11. Permissive Syntax of HTML

- HTML4 and HTML5 are both less strict than XHTML
- not all elements need *start* tags: e.g. `head`, `body`
- not all elements need *end* tags: e.g. `head`, `body`, `p`, `li`
- some elements (*empty* elements) must *not* have end tags: e.g. `hr`, `br`, `img`
- tag names are case insensitive
- quotes are not needed around most attribute values
- none of the above is true for XHTML

## 3.12. Limitations of (X)HTML

- (X)HTML defines a *fixed set* of elements (XHTML is *one* XML vocabulary)
- elements have *document* structuring semantics
- for presentation to human readers
- organisations want to be able to define their own elements
- applications need to exchange structured *data* too
- applications cannot consume (X)HTML easily
- move to XML (and JSON) for *data* exchange and (X)HTML for document representation

## 3.13. XML Overview

- based on *Standard Generalized Markup Language* (SGML)
- W3C Recommendation (Fifth Edition 2008, originally 1998)
- design of XML motivated by
  - *inflexibility* of HTML
  - *complexity* of SGML
- XML allows *application-specific* document types (vocabularies)
- it has become one of the dominant ways to represent information exchanged among applications on the Internet

- e.g., the *RSS* XML vocabulary used for news feeds such as from the BBC

## 3.14. XML Example

Representing information about a collection of classical CDs: cd-collection.xml

```
<CDcollection>
  <CD>
    <soloist>Martha Argerich</soloist>
    <orchestra>London Symphony Orchestra</orchestra>
    <conductor>Claudio Abbado</conductor>
    <date>1968</date>
    <performance>
      <composer>Frederic Chopin</composer>
      <composition>Piano Concerto No. 1</composition>
    </performance>
    <performance>
      <composer>Franz Liszt</composer>
      <composition>Piano Concerto No. 1</composition>
    </performance>
  </CD>

  <CD>
    <composer>Antonin Dvorak</composer>
    <performance>
      <composition>Symphony No. 9</composition>
      <orchestra>Vienna Philharmonic</orchestra>
      <conductor>Kirill Kondrashin</conductor>
      <date>1980</date>
    </performance>
    <performance>
      <composition>American Suite</composition>
      <orchestra>Royal Philharmonic</orchestra>
      <conductor>Antal Dorati</conductor>
      <date>1984</date>
    </performance>
  </CD>
</CDcollection>
```

## 3.15. Some XML Vocabularies

- XML is becoming ubiquitous as both
  - a storage format
  - a data interchange format
- XML vocabularies are used for
  - expressing mathematical formulas (MathML)
  - drawing 2D pictures (SVG)
  - documents: OpenDocument format (ODF) and Microsoft Office Open XML
  - news feeds (Atom and RSS)
  - defining multimedia presentations (SMIL)
  - defining contents of web resources (RDF)
  - modelling chemical structures (CML)
  - electronic commerce (ebXML, cXML, ...)
  - voice-activated services (voiceXML)

Meaning of acronyms:

- SVG: scalable vector graphics
- SMIL: synchronous multimedia integration language
- RDF: resource description framework
- RSS:
  - Really Simple Syndication (RSS 2.0)
  - Rich Site Summary (RSS 0.91, RSS 1.0)
  - RDF Site Summary (RSS 0.9 and 1.0)
- CML: chemical markup language
- cXML: commerce XML
- ebXML: electronic business XML

# 3.16. SVG

- most browsers now have built-in support for some subset of SVG
  - Firefox, Opera, Safari, Chrome, IE9, ...
  - IE8 and earlier versions needed a (free) Adobe plug-in
  - also included in HTML5
- allows drawing of various 2D shapes (and animation)
- example of 3 circles: svg-example.xml

```
<svg>
 <g style="fill-opacity:0.7; stroke:black; stroke-width:0.1cm;">
  <circle cx="6cm" cy="2cm" r="100" style="fill:red;"
          transform="translate(0,50)" />
  <circle cx="6cm" cy="2cm" r="100" style="fill:blue;"
          transform="translate(70,150)" />
  <circle cx="6cm" cy="2cm" r="100" style="fill:green;"
          transform="translate(-70,150)"/>
 </g>
</svg>
```

# 3.17. MathML

- some browsers have built-in support
  - Firefox, Safari
  - supported in Chrome 24 and in IE8 using (commercial) plug-in, but not subsequently
  - also included in HTML5
- MathJax is a Javascript library for MathML (and LaTeX)
- in MathML, the cube root of (1 minus (x over 2)) can be written as

```
<math>
  <mroot>
    <mrow>
      <mn>1</mn>
      <mo>-</mo>
      <mfrac>
        <mi>x</mi>
        <mn>2</mn>
      </mfrac>
    </mrow>
    <mn>3</mn>
  </mroot>
</math>
```

and looks like this when viewed with a MathML-capable browser, and this when using MathJax

## 3.18. RSS

- RSS is a simple XML vocabulary for use in news feeds
- RSS stands for *Really Simple Syndication*, among other things
- an example is BBC News (view the source to see the XML, since a stylesheet has been applied)
- you can subscribe to news feeds using a news reader
- or they can be used by developers to create other pages, such as news plotted on a map of the world

## 3.19. RSS Example

- a fragment of old news has been stored in `rss-fragment.xml`
- the basic structure is given below

```
<rss>
  <channel>
    <title> ... </title>
      ...
    <item>
      <title> ... </title>
      <description> ... </description>
      <link> ... </link>
      <pubDate> ... </pubDate>
    </item>
      ...
    <item>
      <title> ... </title>
      <description> ... </description>
      <link> ... </link>
      <pubDate> ... </pubDate>
    </item>
  <channel>
</rss>
```

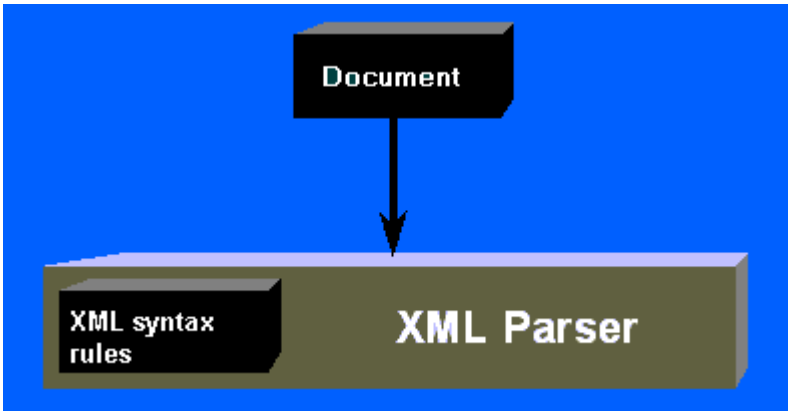
## 3.20. XML Syntax Rules

- a root element is required
  - e.g., `html`
- end tags are required
- elements must be properly nested
  - if start tag of `A` precedes start tag of `B`, end tag of `B` must precede end tag of `A`
- case is significant:
  - e.g. `<A>` is different to `<a>`
- attribute values must be enclosed in quotes
- entity references, other than those to built-in entities, must be declared (see later)

Attribute values can be enclosed in either single quotes or double quotes. This allows one to use quotes inside attribute values, such as `height='72"'`.

# 3.21. Well-Formed XML

- a *well-formed* document observes the syntax rules of XML
- which means the document can be correctly *parsed*




- parsing transforms a stream of characters into a *logical* structure
- the logical structure represents a collection (or hierarchy) of conceptual objects, e.g., elements

# 3.22. More XML syntax

- document should start with the *XML declaration* which gives, e.g., XML version being used
  - e.g., `<?xml version="1.0"?>`
- other tags which begin with `<?` and end with `?>` are called *processing instructions*, e.g. to specify a stylesheet
- special shorthand allowed for empty elements
  - e.g., `<hr/>` for `<hr></hr>`
- note that *whitespace* characters (space, tab, carriage return and line feed) do not constitute empty content
- comments are enclosed between `<!--` and `-->`
  - e.g., `<!--changed 10/1/01-->`
- characters to be interpreted as themselves rather than XML can be placed in a *CDATA section* which starts with `<![CDATA[` and ends with `]]>`
  - e.g., comment above produced using `<![CDATA[<!--changed 10/1/01-->]]>`

The XML declaration can also specify the character encoding used, using the `encoding` attribute, and whether or not the document is dependent on others, using the `standalone` attribute. Putting this information inside the document makes it *self-identifying*, which is preferable to using file extensions, for example.

Processing instructions contain instructions which are supposed to be passed on to applications processing the document.

# 3.23. Character Encodings

- a minority of web users speak English
- W3C Internationalization (I18n) Activity tries to ensure that formats and protocols are usable worldwide in all languages and in all writing systems
- original ASCII character encoding allows only 128 characters

    - letters of the English alphabet
    - numbers
    - some common symbols and special characters (e.g. line feed)
- Unicode is designed to be a universal system for encoding all the characters in all the world's languages
- the form of Unicode used in XML is called UTF-8
    - uses 1 to 4 bytes to represent each character

## 3.24. Character and built-in entities

- *character entity* references can be used represent characters not otherwise accessible
    - start with `&` and `#` characters; end with `;` character
    - refer to character's encoding either in decimal: e.g. `&#8804;` for ≤ (less than or equal to)
    - or in hexadecimal: e.g. `&#x05D0;` for א (Hebrew aleph character)
- *entities* also used to "escape" the meaning of XML special (markup) symbols
- only 5 entities are built-in
    - `&` (&), `<` (<), `>` (>), `&quot;` ("), `&apos;` (')
- so <, e.g., can be obtained using `<`, `&#60;` or `&#x3C;`
- see DTDs for more general uses of entities

> Single quotes and double quotes have special meaning in XML in that they are used to delimit attribute values. If, for example, we want to use a double quote to represent inches inside an attribute value delimited using double quotes, we can use the `quot` entity to do so as follows: `height="72&quot;"`. Now the value of the `height` attribute returned by an XML parser will be `72"`.

## 3.25. Namespaces

- say we want to use XHTML, MathML and SVG in a single XML document
- how does a browser know which element is from which vocabulary?
- e.g., both SVG and MathML define a `set` element
- we shouldn't have to worry about potential name clashes
- should be able to switch between different *namespaces*, one for each of XHTML, MathML and SVG

```
<html xmlns="http://www.w3.org/1999/xhtml" ... >
  ...
  <svg xmlns="http://www.w3.org/2000/svg" ... >
    ...
  </svg>
  ...
  <math xmlns="http://www.w3.org/1998/Math/MathML">
    ...
  </math>
  ...
</html>
```

- in fact, HTML5 allows MathML and SVG to be used without namespaces, as in this example

# 3.26. The namespaces solution

- solution is to *qualify* element names with *URIs*
- since URI is a unique identifier
- *qualified name* then consists of two parts

```
namespace:local-name
```

- e.g.

```
<http://www.w3.org/2000/svg:circle ... />
```

- where `http://www.w3.org/2000/svg` is a URI and namespace
- URI doesn't have to reference real Web resource
- URIs only disambiguate names, don't have to define them
- in this case, the browser knows the SVG namespace and behaves accordingly

# 3.27. Namespace declarations

- using URIs everywhere is very cumbersome
- so namespaces are used indirectly using
  - namespace *declarations* and
  - associated *prefixes* (user-specified)

```
<... xmlns:svg="http://www.w3.org/2000/svg">
   <p>A circle looks like this
   ...
      <svg:circle ... />
   ...
</...>
```

- `xmlns:svg` attribute
  - declares the namespace `http://www.w3.org/2000/svg`
  - associates it with prefix `svg`

# 3.28. Scope of namespace declarations

- the *scope* of a namespace declaration is
  - the element containing the declaration
  - and all its *descendants* (those elements nested inside the element)
  - can be overridden by *nested* declarations
- both elements and attributes can be qualified with namespaces
- unprefixed element names are assigned a *default* namespace
- default namespace declaration: `xmlns="URI"`

## 3.29. Namespaces example

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:svg="http://www.w3.org/2000/svg">
   ...
   <p>A circle looks like this
      <svg:svg ... >
         ...
         <svg:circle ... />
         ...
      </svg:svg>
      and has
      ...
   </p>
</html>
```

- `html` and `p` are in the *default* namespace (`http://www.w3.org/1999/xhtml`)
- `svg` and `circle` are in the `http://www.w3.org/2000/svg` namespace
- note that `svg` is used both as a prefix and as an element name
- what if we left out the *prefix* `svg` in front of the element name `svg`?

## 3.30. Overriding namespaces

- consider again the example where the default namespace is overridden

```
<html xmlns="http://www.w3.org/1999/xhtml" ... >
  ...
  <svg xmlns="http://www.w3.org/2000/svg" ... >
    ...
  </svg>
  ...
  <math xmlns="http://www.w3.org/1998/Math/MathML">
    ...
  </math>
  ...
</html>
```

- namespace for
  - `html` element and its descendants is `http://www.w3.org/1999/xhtml`
  - `svg` element and its descendants is `http://www.w3.org/2000/svg`
  - `math` element and its descendants is `http://www.w3.org/1998/Math/MathML`
- note the entity reference to `pi` built in to MathML

## 3.31. JSON

- JSON stands for JavaScript Object Notation
- no longer actually tied to any particular programming language
- now widely used for data interchange
- more concise than XML

## 3.32. JSON Data Types

- number: e.g. `23.45`
- string: e.g. `"abc"`
- Boolean: e.g. `true`
- `null`
- array

    - an ordered list of zero or more values, each of which may be of any type
    - uses square bracket notation, with elements being comma-separated
- object
    - an unordered associative array (key/value pairs)
    - objects are delimited with curly brackets and use commas to separate each pair
    - within each pair, the colon ':' character separates the key from its value
    - keys must be strings and should be distinct within the object

## 3.33. JSON example

XML `CDCollection` example from earlier, represented in JSON

```
{
  "CDs": [
    {
      "soloist": "Martha Argerich",
      "orchestra": "London Symphony Orchestra",
      "conductor": "Claudio Abbado",
      "date": 1968,
      "performances": [
        {
        "composer": "Frederic Chopin",
        "composition": "Piano Concerto No. 1"
        },
        {
        "composer": "Franz Liszt",
        "composition": "Piano Concerto No. 1"
        }
      ]
    },
    {
      "composer": "Antonin Dvorak",
      "performances": [
        {
        "composition": "Symphony No. 9",
        "orchestra": "Vienna Philharmonic",
        "conductor": "Kirill Kondrashin",
        "date": 1980
        },
        {
        "composition": "American Suite",
        "orchestra": "Royal Philharmonic",
        "conductor": "Antal Dorati",
        "date": 1984
        }
      ]
    }
  ]
}
```

## 3.34. Exercises

1. Consider using an XML document to represent information about students on an MSc programme. All information should be represented using elements rather than attributes. The root element of the document is `programme`. A `programme` has a `degree`, whose value in this case is "MSc", and a `year`, whose value in this case is "2018/19". These elements are followed by the `results` for the programme. The `results` are partitioned into `distinction`, `merit`, `pass` and `fail`. Within each is a sequence of `name` elements, each containing the name of a person having achieved the corresponding `result` for the `programme`. The actual results are as follows: Jemima Puddle-Duck and Peter Rabbit obtained distinctions, Tom Kitten obtained a merit, and

Samuel Whiskers failed (nobody obtained only a pass).

2. Consider a relational database containing a relation `teaches` with attributes `course` and `lecturer`, representing the relationship between courses taught on an MSc programme and the lecturers who teach them. Give an XML document which represents a relation instance containing two tuples. (You are free to make up your own course and lecturer names.)

3. Now represent the information in both the above exercises in JSON rather than XML.

## 3.35. Links to more information

- some history relating to hypertext and HTML
- Architecture of the World Wide Web, Volume One (W3C Recommendation)
- HTML tutorial
- HTML 4.01 (W3C Recommendation)
- the XML 1.0 specification
- the Annotated XML Specification by Tim Bray
- the XHTML 1.0 specification
- XML information
- XML tutorial from W3Schools
- list of free XML tools (no longer maintained)
- W3C MathML home page
- W3C SVG home page
- W3C tutorial on Character sets & encodings in XHTML, HTML and CSS
- Web Accessibility Initiative (WAI) is concerned with making the Web accessible to people with disabilities
- the W3C XML Namespace Recommendation
- an explanation of the namespace recommendation by James Clark
- Official JSON web site
- ECMA-404 - The JSON Data Interchange Format

HTML is covered in Chapter 1 and XML in Chapter 2 of [Moller and Schwartzbach]. XML is covered in Chapter 1 of [Jacobs]. HTML and XML are mentioned in passing in Chapter 4 of [Comer].