Transforming XML Data with XSLT

- Eventually, the web-page content should be displayed in a web browser
  - the web-page content is encoded in XML and needs to be rendered
  - standard task in web-information systems built on top of XML databases

- Web-pages are more than just content, they also comprise:
  - a URI to address them
  - navigation links to other web-pages
  - layout and style options (presentation)
  - operations (functionality)
  - adjustment mechanisms (adaptivity)
  - etc.

- The *XSLT* language can be used to specify the presentation of XML data
  - to modifies the layout
  - to add style options
  - to add navigation links to other web-pages

XSLT

- The XSLT language is a *query language* that can be used to transform XML data

- XSLT has a mother language: the *Extensible Stylesheet Language (XSL)*
  - XSLT stands for *XSL Transformations*
  - the other daughter of XSL is XSL Formating Objects (XSLFO)
  - XSLFO is suitable for specifying physical layout

- Originally, the XSLT language was developed for creating stylesheets
  - XSLT can be used to transform XML data into HTML documents
  - but this is only one possible application of XSLT

- we will use XSLT to transform web-page content into XHTML documents
  - *XHTML* is the XMLification of HTML
  - XHTML documents are well-formed XML documents
  - XHTML documents can then be rendered in a web browser
  - thus, they may serve as user interfaces of a web-information system

XSLT documents for Generating XHTML

- For convenience, XSLT transformations are stored in XSLT documents

- In case of XSLT transformations for generating XHTML:
  - the XSLT document is an XML document with root element html
  - the XSLT document starts as follows:

```
⟨html xsl:version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                        xmlns="http://www.w3.org/1999/XHTML"⟩
```

  - *xsl* stands for the *namespace* of the XSLT language (containing all keywords)
  - we also link the XSLT document to the XHTML web-site

- The *transformation* is a valid expression from the XSLT language
  - when applying the XSLT transformation, this XSLT expression will be *evaluated*
  - this evaluation is usually done against an input XML document

- When applying the transformation, an *output XHTML document* is generated

XSLT Expressions

- The XSLT language is a *W3C Recommendation* since 16 November 1999
  - XSLT 2.0 is a W3C Recommendation since 23 January 2007
  - XSLT uses the XPath language which is a W3C Recommendation, too

- The XSLT language uses the following kinds of expressions:
  - paths expressions
  - value-extraction expressions
  - node constructors
  - repetition expressions
  - conditional expressions
  - sorting expressions
  - copy expression

- The XSLT language includes the standard XHTML language
  - XHTML expressions are valid XSLT expressions
  - XHTML may be used to construct XHTML nodes (elements, attributes, text)
  - recall that XHTML is part of XML

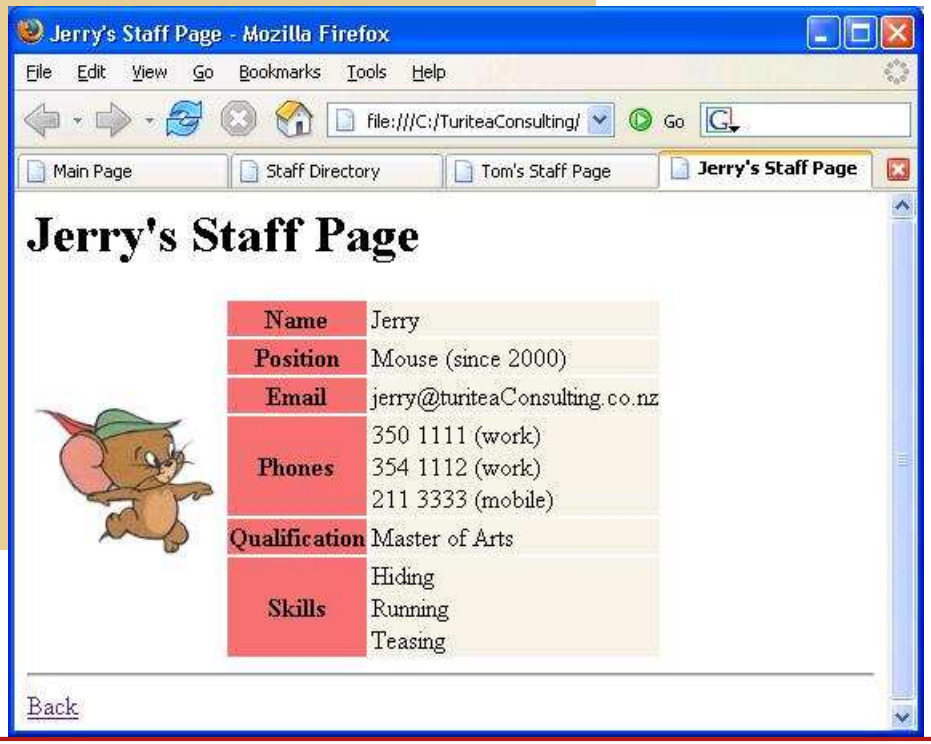
Using XSLT to Display Web-Page Content

- Our next step:
  - Create a transformation that generates an XHTML document for a staff page

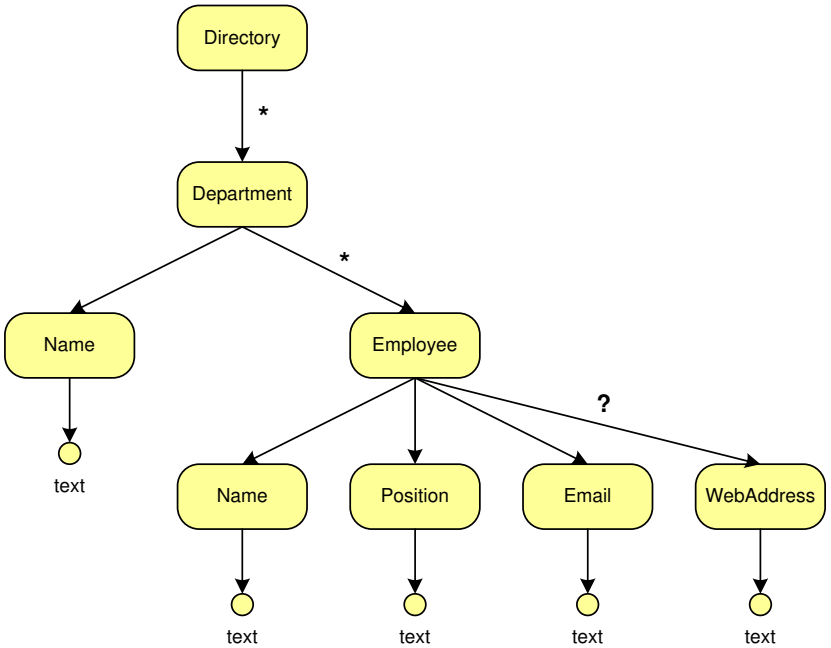
```
⟨html xsl:version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                        xmlns="http://www.w3.org/1999/XHTML"⟩
  ⟨head⟩
    ⟨title⟩⟨xsl:value-of select="Employee/Name" /⟩'s Staff Page⟨/title⟩
  ⟨/head⟩
  ⟨body⟩
    ⟨h1⟩⟨xsl:value-of select="Employee/Name" /⟩'s Staff Page⟨/h1⟩

    ⟨table⟩
    ... here go the table rows ...
    ⟨/table⟩

    ⟨hr/⟩
    ⟨a href="staffdirectory.html"⟩Back⟨/a⟩
  ⟨/body⟩
⟨/html⟩
```

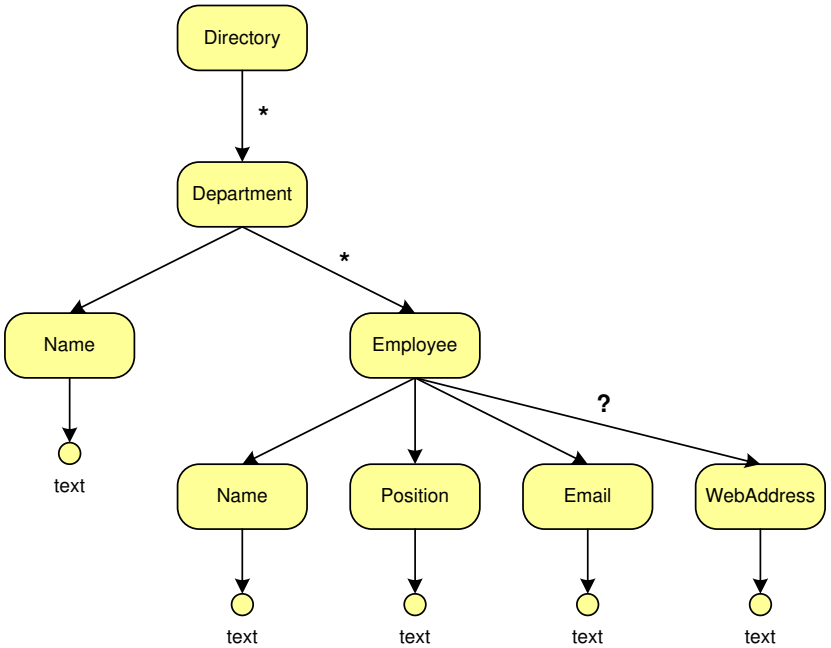


Selecting XML Nodes in XML Trees



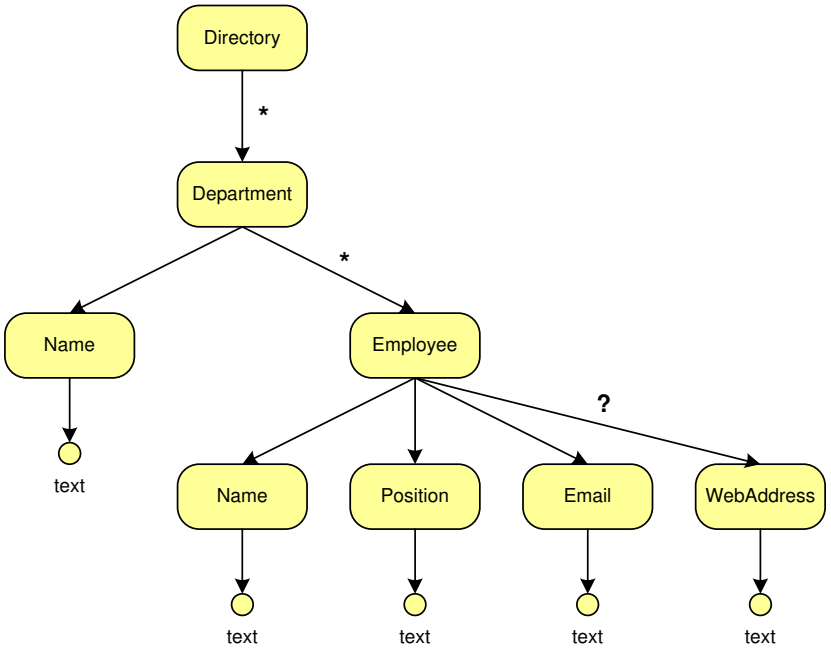
- Here is an example of a *location path*: Department / Employee / Name
  - Location paths are used to select (a sequence of) nodes in XML trees
- Which nodes are selected by the example path?
  - Let us assume that we sit in the Directory-node
    - The starting point for the evaluation is called the *context node*
  - To answer this question, we need to *evaluate* the location path
  - The example path selects just a single node: the Name-node under the Employee-node

Choosing the Context Node



- The choice of the context node matters: the path `Name` selects
  - the first Name-node, if the Department-node is the context node
  - the second Name-node, if the Employee-node is the context node
  - nothing, if we choose any other node as the context node
- If we want the document node to be the context node, then we put a slash in front of the location path, e.g. `/ Directory / Department / Name`
  - the *document node* is an additional virtual node on top of the entire tree

Selecting Text and Attribute Nodes



- We can also specify paths for selecting text nodes: `Employee / Position / text()`
  - text() is used to select the text content of XML elements
- And for selecting attribute nodes: `Employee / Position / @Since`
  - the symbol @ indicates attributes (to distinguish them from XML elements)
- To be more flexible (or lazy) we can skip some nodes: `Directory // Email`
  - the double slash // is used as a wildcard for any sequence of nodes

Selecting XML Nodes and Value Extraction

- XSLT uses location paths to select nodes in XML documents
  - to begin with, the context node is the document node of the XML document used as input for the XSLT transformation

- The `xsl:value-of` instruction generates text data from the XML nodes *selected* by a location path
  - extracts the value of attribute nodes
  - extracts the pure text content of element nodes
  - better apply the instruction only to text, attribute, or element nodes with pure text content

```
⟨tr⟩
  ⟨th rowspan="6"⟩⟨img src="{Employee/Photo/text()}" /⟩⟨/th⟩
  ⟨th⟩Name⟨/th⟩
  ⟨td⟩⟨xsl:value-of select="Employee/Name" /⟩⟨/td⟩
⟨/tr⟩
```

- Enclosed expressions can be used inside a ""-environment
  - the content has to be *computed* first
  - can be used for value extraction (when inside a ""-environment)

Repetition Expressions

```
⟨tr⟩
  ⟨th⟩Position⟨/th⟩
  ⟨td⟩⟨xsl:value-of select="Employee/Position" /⟩
      (since ⟨xsl:value-of select="Employee/Position/@Since" /⟩)⟨/td⟩
⟨/tr⟩
```

```
⟨tr⟩
  ⟨th⟩Email⟨/th⟩
  ⟨td⟩⟨xsl:value-of select="Employee/Email" /⟩⟨/td⟩
⟨/tr⟩
```

- The `xsl:for-each` instruction iterates through the *selected* XML nodes
  - each time, it evaluates the XSLT expression *inside*, and adds to the overall result
  - observe, the change of the context node for the location paths inside

```
⟨tr⟩
  ⟨th⟩Phones⟨/th⟩
  ⟨td⟩
    ⟨xsl:for-each select="Employee/Phones/Phone"⟩
      ⟨xsl:value-of select="." /⟩ (⟨xsl:value-of select="@Kind" /⟩) ⟨br/⟩
    ⟨/xsl:for-each⟩
  ⟨/td⟩
⟨/tr⟩
```

Conditional Expressions

- The `xsl:if` instruction evaluates the XSLT expression *inside* only if the *tested* location path is valid
  - if the employee has no qualification, then we skip this row
  - be careful: here the context node for the location path inside does not change

```
⟨xsl:if test="Employee/Qualification"⟩
  ⟨tr⟩
    ⟨th⟩Qualification⟨/th⟩
    ⟨td⟩⟨xsl:value-of select="Employee/Qualification" /⟩⟨/td⟩
  ⟨/tr⟩
⟨/xsl:if⟩
```

```
⟨tr⟩
  ⟨th⟩Skills⟨/th⟩
  ⟨td⟩
  ⟨xsl:for-each select="Employee/Skills/Skill"⟩
    ⟨xsl:value-of select="." /⟩ ⟨br/⟩
  ⟨/xsl:for-each⟩
  ⟨/td⟩
⟨/tr⟩
```

- This completes the rows of our table, and thus the entire XSLT transformation for the staff web-page

Using XSLT to Display Web-Page Content

- Our next step:
  - Create a transformation that generates an XHTML document for a staff directory

```
⟨html xsl:version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                        xmlns="http://www.w3.org/1999/XHTML"⟩
  ⟨head⟩
    ⟨title⟩Staff Directory⟨/title⟩
  ⟨/head⟩
  ⟨body⟩
    ⟨h1⟩Staff Directory⟨/h1⟩
    ⟨xsl:for-each select="Directory/Department"⟩
      ⟨h2⟩⟨xsl:value-of select="Name" /⟩⟨/h2⟩
      ⟨table⟩
        ... here go the table rows ...
      ⟨/table⟩
    ⟨/xsl:for-each⟩
    ⟨hr/⟩
    ⟨a href="main.html"⟩Back⟨/a⟩
  ⟨/body⟩
⟨/html⟩
```

Node Constructors

- XSLT may be used to construct XHTML nodes
  - we can use *node constructors* for create new nodes (elements, attributes, text)
  - direct node constructors use the standard XHTML language

```
⟨tr⟩
  ⟨th⟩Name⟨/th⟩
  ⟨th⟩Position⟨/th⟩
  ⟨th⟩Email⟨/th⟩
⟨/tr⟩
```

  - alternatively, they may be used to create wrappers around computed content

```
⟨xsl:for-each select="Employee"⟩
  ⟨tr⟩
    ⟨xsl:if test="WebAddress"⟩
      ⟨td⟩⟨a href="{WebAddress}"⟩⟨xsl:value-of select="Name" /⟩⟨/a⟩⟨/td⟩
    ⟨/xsl:if⟩
    ⟨td⟩⟨xsl:value-of select="Position" /⟩⟨/td⟩
    ⟨td⟩⟨xsl:value-of select="Email" /⟩⟨/td⟩
  ⟨/tr⟩
⟨/xsl:for-each⟩
```

- XSLT instructions can be nested into one another
  - recall that the WebAddress is optional, so we test whether it exists

Sorting Expressions

- The `xsl:sort` instruction can be used to sort the XML nodes according to the *selected key field*
  - the *order* can be ascending and descending
  - values may be compared as strings or as numbers
    - this is important: 250 versus 1000
  - to sort according to multiple key fields, sort instructions may be nested

```
⟨xsl:for-each select="Employee"⟩
  ⟨xsl:sort select="Name" order="ascending" data-type="string"⟩
  ⟨tr⟩
    ⟨xsl:if test="WebAddress"⟩
      ⟨td⟩⟨a href="{WebAddress}"⟩⟨xsl:value-of select="Name" /⟩⟨/a⟩⟨/td⟩
    ⟨/xsl:if⟩
    ⟨td⟩⟨xsl:value-of select="Position" /⟩⟨/td⟩
    ⟨td⟩⟨xsl:value-of select="Email" /⟩⟨/td⟩
  ⟨/tr⟩
⟨/xsl:for-each⟩
```

XSLT documents for Generating XML

- XSLT transformations may also be used to generate other XML documents
  - they are not restricted to generating XHTML
  - the XSLT language is a powerful query language

- XSLT transformations are again stored in XSLT documents
  - the XSLT document is an XML document with a root element
  - the XSLT document starts, for example, as follows:

```
⟨Results xsl:version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"⟩
```

  - *xsl* stands for the *namespace* of the XSLT language (containing all keywords)

- When applying the transformation, an *output XML document* is generated

Copying Nodes and Creating Nodes

- The `xsl:copy-of` instruction can be used to copy the *selected* XML nodes into the output document
  - the XSLT language includes the entire XML language
  - the alternative `xsl:copy` instruction eliminates child elements and attributes

- To copy all employees with a staff page, we can use:

```
⟨Results xsl:version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"⟩
  ⟨xsl:copy-of select="//Employee[WebAddress]"⟩
⟨/Results⟩
```

- The `xsl:attribute` instruction can be used to create new attribute nodes

```
⟨Results xsl:version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"⟩
  ⟨xsl:for-each select="//Employee"⟩
    ⟨Staff⟩
      ⟨xsl:attribute name="Salary"⟩confidential⟨/xsl:attribute⟩
      ⟨xsl:attribute name="Position"⟩⟨xsl:value-of select="Position/text()"⟩⟨/xsl:attribute⟩
    ⟨/Staff⟩
  ⟨/xsl:for-each⟩
⟨/Results⟩
```

Creating Nodes with Computed Names

- The `xsl:element` instruction can be used to create new element nodes

```
⟨Results xsl:version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"⟩
  ⟨xsl:for-each select="//Employee"⟩
    ⟨xsl:element name="Position/text()"⟩
      ⟨xsl:value-of select="Name" /⟩
    ⟨/xsl:element⟩
  ⟨/xsl:for-each⟩
⟨/Results⟩
```

  - here the element names have to be computed first:

```
⟨Results⟩
  ⟨Cat⟩Tom⟨/Cat⟩
  ⟨Mouse⟩Jerry⟨/Mouse⟩
⟨/Results⟩
```

- Note that this transforms data into metadata (the element tags)
  - similarly, one can transform attribute values as attribute names

XSLT documents revisited

- There is an alternative format for XSLT documents
  - rather than

```
⟨html xsl:version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                        xmlns="http://www.w3.org/1999/XHTML"⟩
... here go the instructions ...
⟨/html⟩
```

  - we can use the following for generating XHTML:

```
⟨xsl:transform xsl:version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                        xmlns="http://www.w3.org/1999/XHTML"⟩
  ⟨xsl:template match="/"⟩
    ⟨html⟩
    ... here go the instructions ...
    ⟨/html⟩
  ⟨/xsl:template⟩
⟨/xsl:transform⟩
```

  - the root element can also `xsl:stylesheet` instead of `xsl:transform`

- The `xsl:template` instruction defines a *template* for the root element

XSLT documents revisited

- There is an alternative format for other XML documents
  - rather than

```
⟨Results xsl:version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"⟩
... here go the instructions ...
⟨/Results⟩
```

  - we can use the following for generating XML:

```
⟨xsl:transform xsl:version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  ⟨xsl:template match="/"⟩
    ⟨Results⟩
    ... here go the instructions ...
    ⟨/Results⟩
  ⟨/xsl:template⟩
⟨/xsl:transform⟩
```

  - the root element can also `xsl:stylesheet` instead of `xsl:transform`

- The `xsl:template` instruction defines a *template* for the root element

Applying Template Rules

- The `xsl:apply-template` instruction can be used to apply other templates
  - the template will be applied to all *selected* XML nodes
  - of course, the template has to be defined

```
⟨xsl:transform xsl:version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
               xmlns="http://www.w3.org/1999/XHTML"⟩
  ⟨xsl:template match="/"⟩
    ⟨html⟩
      ⟨head⟩
        ⟨title⟩Staff Directory⟨/title⟩
      ⟨/head⟩
      ⟨body⟩
        ⟨h1⟩Staff Directory⟨/h1⟩
        ⟨xsl:apply-templates select="Directory/Department" /⟩
        ⟨hr/⟩
        ⟨a href="main.html"⟩Back⟨/a⟩
      ⟨/body⟩
    ⟨/html⟩
  ⟨/xsl:template⟩
⟨/xsl:transform⟩
```

Defining Template Rules

- The `xsl:template` instruction can be used to define other templates
  - a template can be used for all XML nodes that *match* the specified location path (here also called *pattern*)

```
⟨xsl:template match="//Department"⟩
  ⟨h2⟩⟨xsl:value-of select="Name" /⟩⟨/h2⟩
  ⟨table⟩
    ⟨xsl:apply-templates select="Employee" /⟩
  ⟨/table⟩
⟨/xsl:template⟩
```

```
⟨xsl:template match="//Employee"⟩
  ⟨tr⟩
    ⟨xsl:if test="WebAddress"⟩
      ⟨td⟩⟨a href="{WebAddress}"⟩⟨xsl:value-of select="Name" /⟩⟨/a⟩⟨/td⟩
    ⟨/xsl:if⟩
    ⟨td⟩⟨xsl:value-of select="Position" /⟩⟨/td⟩
    ⟨td⟩⟨xsl:value-of select="Email" /⟩⟨/td⟩
  ⟨/tr⟩
⟨/xsl:template⟩
```

- Templates allow the modularisation of XSLT transformation, and motivate reuse

Presenting Web Content with XHTML and CSS

- Recall that we want to display web-page content in a web browser
  - the web-page content is encoded in XML and needs to be rendered
  - we used the XSLT language to generate an XHTML document
  - the XSLT transformation specifies the structural layout for the web-page

- There are other aspects of presentation such as *style* (colours, fonts, sizes, etc.)
  - we could have used the XSLT transformation to include style information, too
  - it is recommended to separate structural layout and style

- The presentation of web-pages is not only an artistic, but also a management problem
  - one needs to maintain uniform appearance over the web-information system
  - nuances between different areas of the web-information system should be introduced in a controlled manner
  - at the same time, it should be possible to change the appearance in a consistent way without re-implementing the web-information system

- The *CSS language* can be used to specify style information for web-pages

CSS Rules

- The *Cascading Style Sheet language (CSS)* can be used to specify style information for XHTML (and other XML) documents
  - *rules* are statements about stylistic aspects of one or more nodes
  - a *style sheet* is a collection of rules

- A *rule* has the general form `selector {property-declarations}`
  - the *selector* specifies which nodes are affected by the rule
  - the property declarations set forth what the effect will be
  - the individual property declarations in the list are separated by semicolons
  - each *property declaration* has the form `property: value`
  - the property is a stylistic attribute that the affected nodes possess

- Examples:
  - `body {color: blue; background: white;}`
  - `h1 {color: green; font-size: 24pt; font-style: italic; text-align: center;}`
  - `Name {color: red}`

CSS Selectors

- A CSS rule applies to all nodes that *match* the selector
  - unfortunately, the CSS language does not use XPath selection paths
  - rather it uses *CSS patterns*
  - here are some common examples of CSS patterns
    (E and F are element names, and A is an attribute name)

Pattern	Meaning
*	matches any element node
E	matches any E node
E F	matches any F node that is a descendant of an E node
E>F	matches any F node that is a child of an E node
E[A]	matches any E node that has an A attribute
E[A="v"]	matches any E node that has an A attribute with value v
$\#i$	matches the node whose id attribute has the value i

- Examples:
  - `p {color: black}`
  - `h1 p {color: green}`
  - `Employee Name {color: red}`

Some common Properties of XHTML elements

- Some common tasks of style sheets
  - specifying colors (for rendering text)
  - specifying fonts (for rendering text)
  - specifying margins (for rendering blocks)

Property	Some sample values
color	red, yellow, rgb(255,204,204), #ffcccc
font-style	normal, italic, oblique
font-weight	normal, bold
font-size	12pt, larger, 150%, 1.5em
font-family	serif, Arial
font	italic bold 2em Arial
margin-top	2em
margin-right	5em, 10%
margin-bottom	2em
margin-left	5em, 10%
margin	2em 5em 2em 5em

- We note:
  - the properties font and margin are shorthand properties for setting several related properties at once

Visual Formatting

- Web browsers render XHTML elements either inline or as blocks
- *Block-level elements* are those elements that are formatted visually as blocks
  - their pure text content is displayed in a box
  - by default, the following elements are rendered as blocks:
    paragraphs (p), headers (h1, . . . , h6), tables (table, tr, td, th), lists (ul, ol, li)
- *Inline-level elements* are those elements that do not form new blocks
  - their pure text content is distributed in lines
  - usually, these are the emphasised pieces of text within a paragraph, etc.
  - by default, the following elements are rendered inline: b, em, i
- The property *display* specifies whether an element is inline-level or block-level
  - for XHTML elements this property is automatically set by the web browser
  - but not for other XML elements
- Examples:
  - Department Name {display: block}
  - Employee Name {display: inline}

Presentation Experiments

- The XHTML language provides a range of elements that have their own "typical" appearance
  - web browsers render them using their default CSS rules
  - unless we change the default presentation
  - the CSS language is powerful enough to change the presentation of any XHTML element into virtually any other
  - in general, however, we do not recommend to do this

- The XHTML language provides two special elements that designers can use for "presentation experiments"
  - *div* is an all-purpose block-level element
  - *span* is an all-purpose inline-level element
  - there are no default values for presenting these elements (apart from the display property)

- Example: to have a means for rendering text in red and centering it, we
  - declare the CSS rule `div.myRedCenter {color: red; text-align: center;}`
  - and use `⟨div class="myRedCenter"⟩Hello World⟨/div⟩` in the XHTML document

Classifying XHTML elements

- We can declare CSS rules
  - for all elements of some type, e.g., `p {color: green}`
  - or for individual elements, e.g., `#p26 {color: green}`
  - the latter CSS rule only applies to the unique paragraph with id "p26"

    `⟨p id="p26"⟩This is a very important paragraph.⟨/p⟩`
  - What if there are several important paragraphs?

- The XHTML language provides the *class* attribute that can be used in the selector
  - we can declare the CSS rule `p.important {color: green}`
  - this rule applies to all paragraphs that are *classified* as "important"

    `⟨p class="important"⟩This is a very important paragraph.⟨/p⟩`
  - there may be several paragraphs that are classified as "important"
  - there may be other paragraphs that are classified as something else

- Note: the selector `p.important` is actually a shortcut of `p[class="important"]`

Linking a Style Sheet to XHTML documents

- For a style sheet to affect the presentation of web-pages, it must be combined with the respective XHTML documents
  - usually, there are many XHTML documents that use the same style sheet
  - then, we should store the style sheet in a CSS document
  - the CSS document must be linked to the respective XHTML documents
- There are several ways to link a CSS document to an XHTML document:
  - we can include a link element into the head of the XHTML document

    ⟨link href="turiteaConsulting.css" rel="stylesheet" type="text/css" /⟩

  - alternatively, we can use a processing instruction (this works for other XML documents, too)

    ⟨?xml:stylesheet href="turiteaConsulting.css" type="text/css" ?⟩

- It is good habit to tell the web browser which style sheet language is used
  - the type attribute specifies that we used the CSS language
  - potentially, a range of style sheet languages could be used, but at present only CSS is widely supported by web browsers

Merging Style Sheets for XHTML documents

- We can also embed a style sheet into an XHTML document:
  - we can include a style element into the head of the XHTML document

```
⟨style type="text/css"⟩
   ... here go the CSS rules...
⟨/style⟩
```

- Style information may even be kept in several style sheet which can be merged

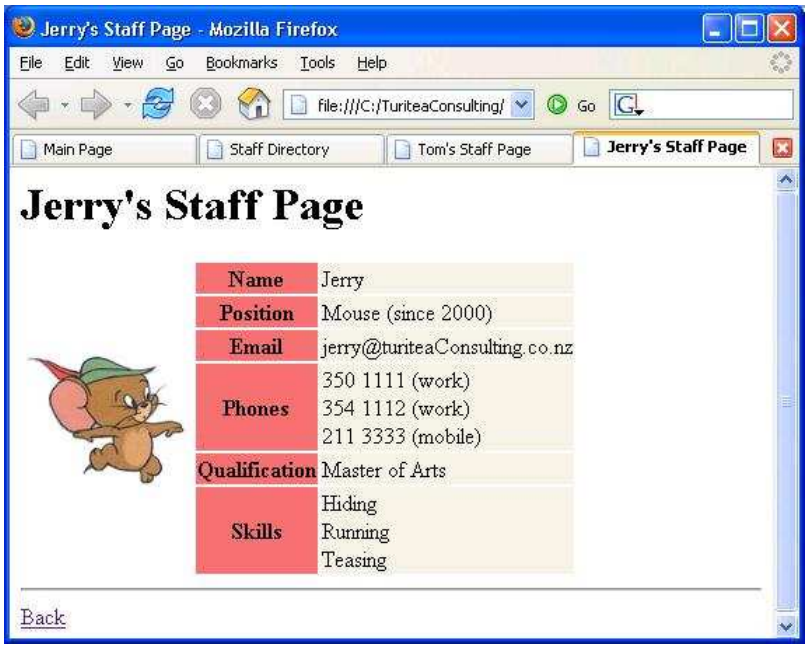
```
⟨link href="turiteaConsulting.css" rel="stylesheet" type="text/css" /⟩
⟨link href="staffpage.css" rel="stylesheet" type="text/css" /⟩
⟨style type="text/css"⟩
   ... here go the internal CSS rules...
⟨/style⟩
```

- Conflicts are resolved by the web browser:
  - the different style sheets are thought of as coming in a series
  - rules in the second CSS document will override rules in the first CSS document
  - internal rules will override external rules
  - this approach is known as *cascading*
  - potential sources of style sheets: the browser, one or more designers, the user

CSS rules for Displaying Web-pages

- Our next step:
  - Create a style sheet **turiteaConsulting.css** that contains CSS rules for rendering the staff pages and the staff directory

```
th {background-color: #f57276;}
h1 {font: bold 2em;}
```



  - Insert ⟨link href="turiteaConsulting.css" rel="stylesheet" type="text/css" /⟩ as a child of the head element in the XSLT documents for the staff pages and staff directory

Generating Web-pages - Summary