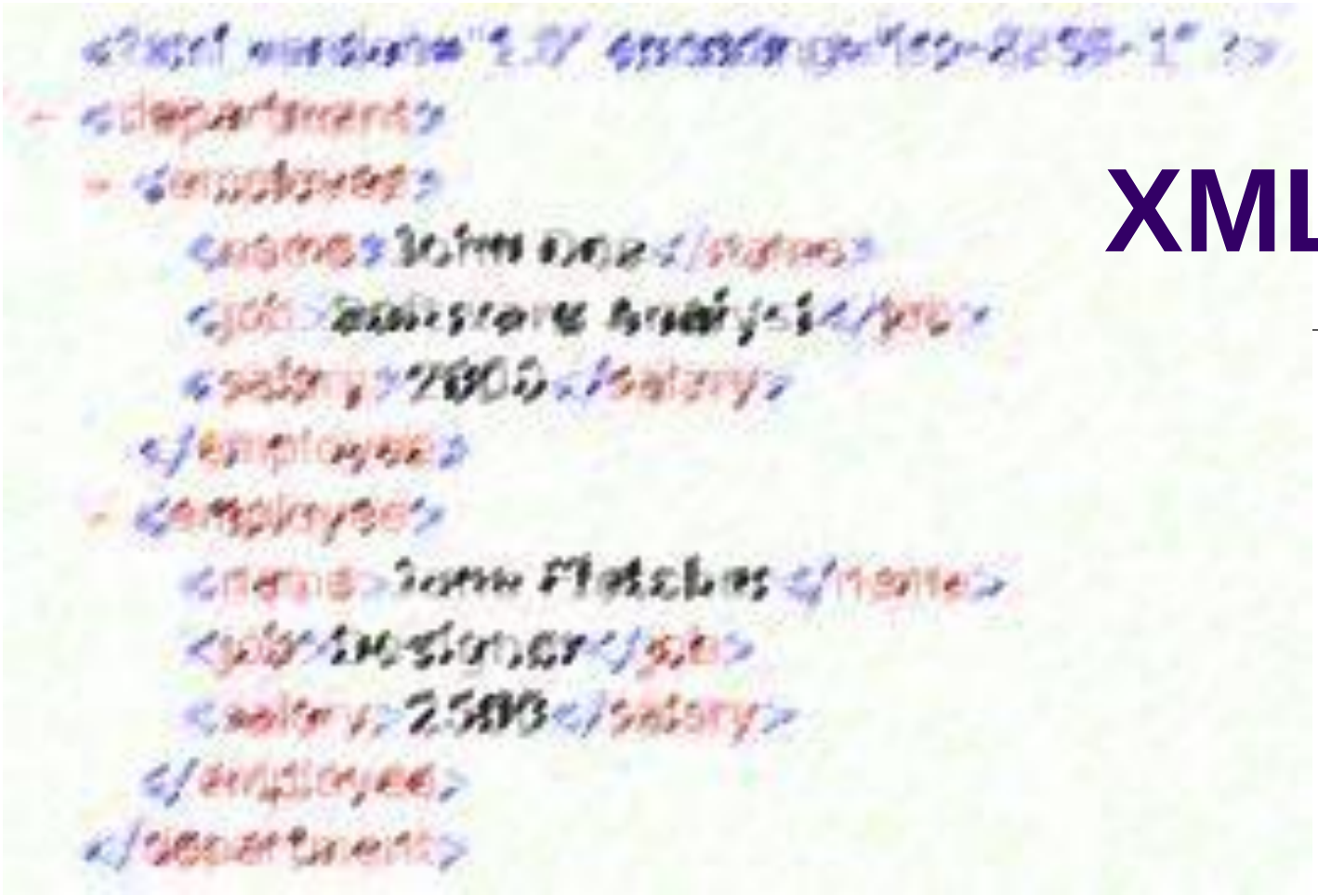
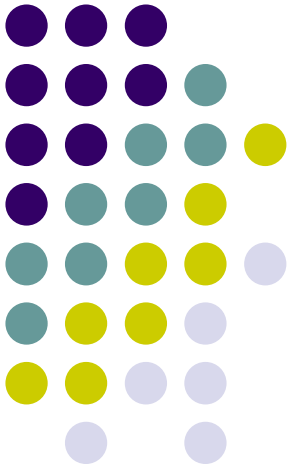
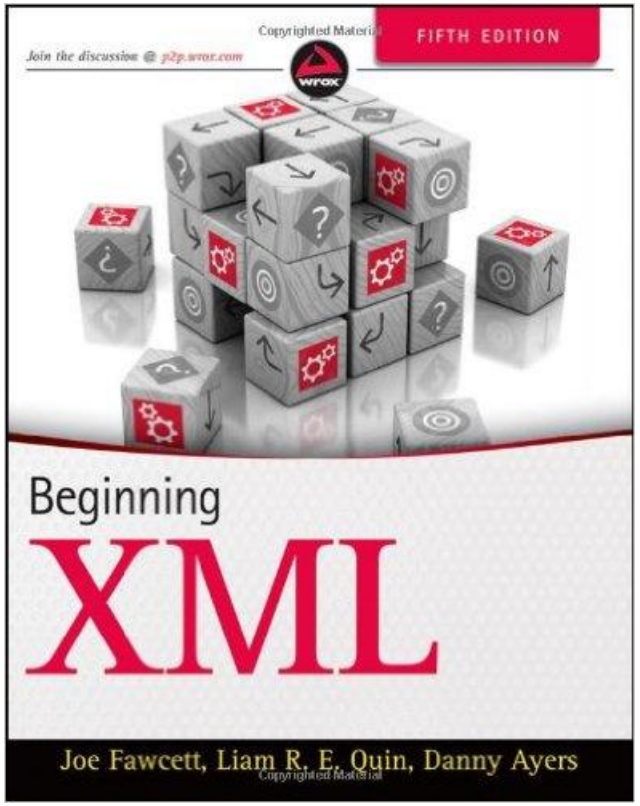


XML



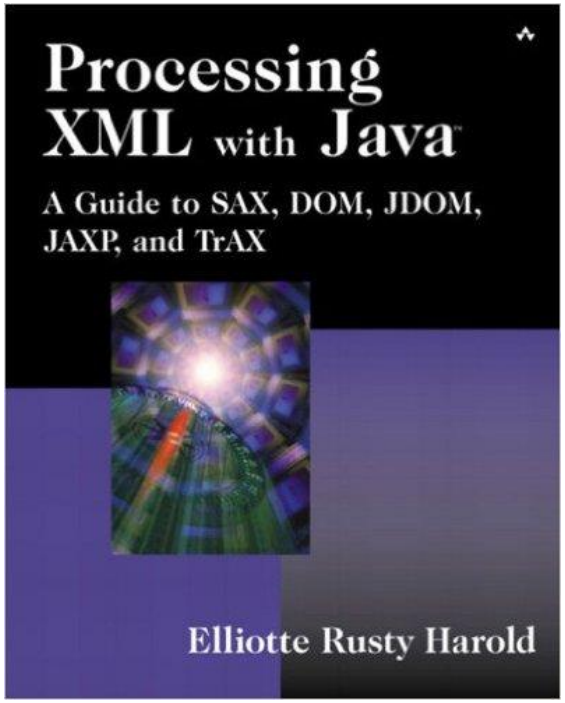
Book

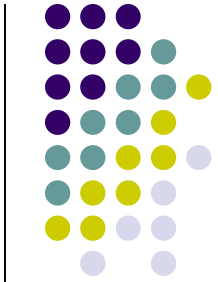
- **Beginning XML**, 5th Ed. by Joe Fawcett, Danny Ayers and Liam R. E. Quin (Jul 2012) **Wrox**



Book, optional for programmers

- Processing XML with Java: A Guide to SAX, DOM, JDOM, JAXP, and TrAX
  Nov 15, 2002 by Elliotte Rusty Harold



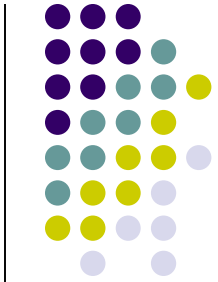


GETTING STARTED

XML

- e**X**tensible **M**arkup **L**anguage
  - Uses meaningful tags to ”mark up” data
- XML is *meta-language.* This means that you use its rules for creating languages.
  - XML languages are rules for creating ”XML” documents
- Designed for
  - data storage
  - transportation

History

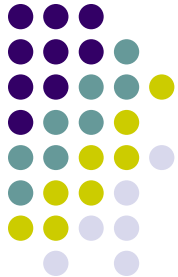


- W3C released in February 1998 and followed with
  - XML Schema - XML structures in terms of elements, attributes and constraints.
  - eXtensible Stylesheet Language (XSL) - presentation logic
  - XSL Transformation (XSLT) - mappings/transformations programs
  - XML Query (XQuery) - queries on XML data.
  - XML Path (XPath) - matching text syntax

Data Models

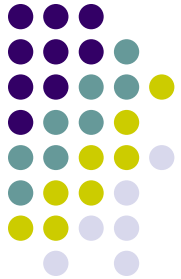
- **Relational** – primary & foreign keys to eliminate data duplication
- **Object** – encapsulation & inheritance to eliminate behavioral duplication
- Key / value - NoSQL
- Files – proprietary
- **XML** – markup for meaning, non-repudiation

Data models



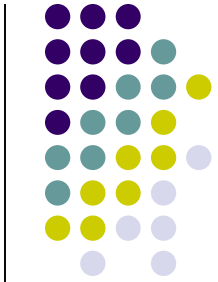
Type	file extensions / in-memory types	significant qualities	query language	template / validating structure
flat file	doc, txt, xls, pdf, csv	ASCII or binary text+formatting	proprietary	none or proprietary
	gif, jpg, png	binary	none	public standard or proprietary
XML	htm, html, rss, xml, … / document	parent-child relationships	XPath	DTD, schema (.xsd)
objects	ser / array, collection	references, behavior, inheritance	language methods	class, interface
relational	all proprietary	hierarchical (FK → PK)	SQL and proprietary SQL	schema of tables

Markup



- Used to add meaning to text
  - semantic
  - SEO = Google can understand and use that meaning
  - <H1> are titles
  - means it should be emphasized, not italicized

Markup choices



- XML
- HTML with data- attributes
- schema.org
- Markdown
  - http://daringfireball.net/projects/markdown/
  - Simplified HTML syntax

.docx – Office Open XML

Windows
MS Word 2007

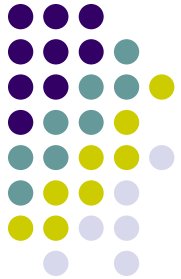
Document.docx

Mac OS X

```
<xml>
<heading1>title</heading1>
.
.
</xml>
```

Now the format is open and it's much easier to access

Well-formed XML



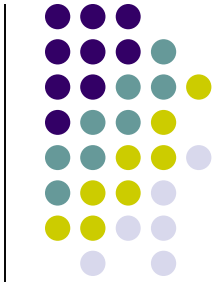
- Elements are closed
  - Tag names must match if using
- All elements must be nested in root element
- Child tags are closed before parent tags are
- Values in quote marks

- It’s not (valid) XML if it’s not well-formed!

XML languages

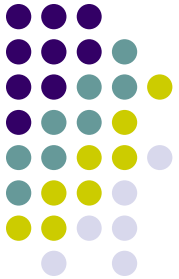
- Sets of tags with rules of cardinality, type and order.
- A set is named by a namespace
  - <namespace:tagName>
- Described in a document used to validate the marked up data
  - DTD
  - Schema (XML Schema Document or XSD)

Correctness



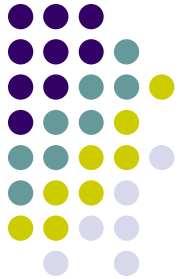
- There are two levels of correctness of an XML document:
  1. **Well-formed -** conforms to basic XML rules.
  2. **Valid** - additionally conforms to some specific XML language rule set (RSS, SOAP, etc.)
- Validation (checking correctness) is enforced by the tool that reads the XML data.

XML languages



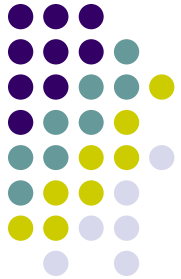
- Examples
  - XHTML
  - ACORD - https://www.acord.org/
  - RSS
  - SVG
  - SOAP
  - WSDL
  - MathML...
- List:
  - http://en.wikipedia.org/wiki/List_of_XML_markup_languages

XHTML - Example



```
<?xml version="1.0"?>
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
   lang="en">
  <head>
    <title>Minimal XHTML 1.0 Document</title>
  </head>
  <body>
    <p>This is a minimal <a
   href="http://www.w3.org/TR/xhtml1/">XHTML 1.0</a>
    document.</p>
  </body>
</html>
```

SVG - Example

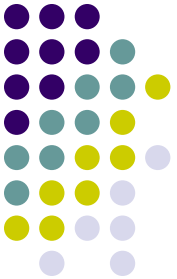


```
<?xml version="1.0"?>
<!DOCTYPE svg
     PUBLIC "-//W3C//DTD SVG 1.1//EN"
     "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">

<svg width="100%" height="100%" version="1.1"
xmlns="http://www.w3.org/2000/svg">

<circle cx="100" cy="50" r="40" stroke="black"
stroke-width="2" fill="red"/>

</svg>
```

MathML (Open Office)

```
<?xml version="1.0"?>
<!DOCTYPE math:math PUBLIC "-//OpenOffice.org//DTD Modified W3C
   MathML 1.01//EN" "math.dtd">
<math:math xmlns:math="http://www.w3.org/1998/Math/MathML">
 <math:semantics>
  <math:mrow>
   <math:mi>x</math:mi>
   <math:mo math:stretchy="false">=</math:mo>
   <math:mfrac>
    <math:mrow>
        ...
    </math:mrow>
  <math:annotation math:encoding="StarMath 5.0">x = {-b +-
   sqrt{b^{2}-4{ac}}   } over {2 {a}} </math:annotation>
 </math:semantics>
</math:math>
```

RSS 2.0 - Example

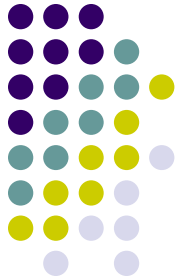


```
<?xml version="1.0"?>
<rss version="2.0">
<channel>
  <title>W3Schools Home Page</title>
  <link>http://www.w3schools.com</link>
  <description>Free web building tutorials</description>
  <item>
    <title>RSS Tutorial</title>
    <link>http://www.w3schools.com/rss</link>
    <description>New RSS tutorial on W3Schools</description>
  </item>
  <item>
    <title>XML Tutorial</title>
    <link>http://www.w3schools.com/xml</link>
    <description>New XML tutorial on W3Schools</description>
  </item>
</channel>
</rss>
```

XML Editors

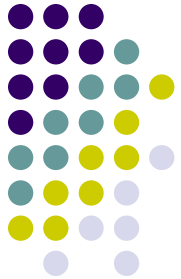
- $$$
  - XML Spy
  - <oXygen/>
  - EditiX - requires Java
- Free
  - ***XML Copy Editor**
  - **Microsoft Visual Studio Code + XML Tools (Josh Johnson)**
  - **Microsoft XML Notepad 2007**

Definitions



- Tag
  - A markup identifier with a name
    - <OpenCloseTagName>
    - </OpenCloseTagName>
- Body
  - Text (including white space) between open and close tags
    - <OpenTagName></CloseTagName>

Definitions



- Element
  - A grouping of text including an open tag, all of the body, and the matching closing tag
- Empty element
  - An element without body. Either <EmptyElement /> or <elementTag></elementTag>
- Root element
  - The element that is not nested in anything.

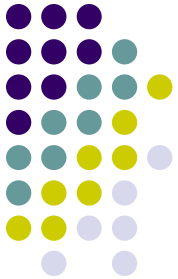
Definitions

- attribute
  - a key value pair of descriptive data placed in the open tag. Single or double quotes delimit the value.
- namespace
  - the source of the group of element names being used abbreviated and prefixed to element names with a colon.
  - <namespace:tagName>

Valid XML

- Follows rules defined in another document
  - Element order
  - Element names
  - Data types of elements
  - Cardinality of elements
- Allows non-repudiation

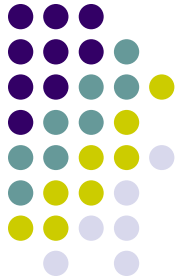
DTDs



- Older validation document
  - Most well-known XML languages like XHTML use a DTD
- Does not use XML rules
- Unable to define too much
  - Name and order of elements
  - Attributes of elements

  - **`<!ELEMENT elementname rule >`**

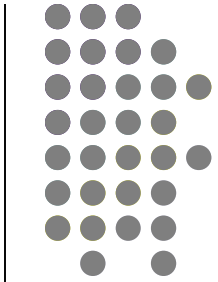
XML Schema document



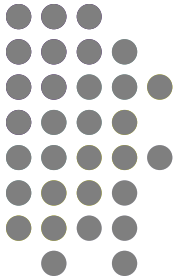
- Defined by another XML based language
- More like a database schema
  - Complex rules
  - Allows aggregation of rules into complex types
- File suffix is usually .xsd

Exercise

- Set up XML Copy Editor



Exercise



- Create a classes.xml data document with several
  - class
    - name
    - teacher
    - room – use only numbers
    - date – use several on one
    - comment – use on one
    - isRemote – use values of true or false

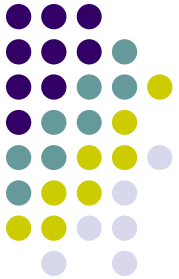
Exercise

- Save and generate an XSD.
  - Generate an XSD online
  - Generate an XSD through XML copy editor
- Validate your data.

XSL & XSLT

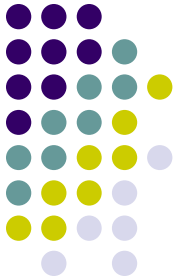
- Used to produce
  - styled documents from raw XML data
  - XML data in a different structure
  - documents on-the-fly for web sites from raw data
- Using XSL and XSL-FO will produce
  - an Adobe PDF file to view in Acrobat
  - a Postscript file to print

Presentation style



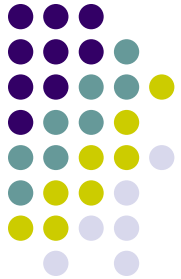
- Can be displayed in a browser
- Can be styled with
  - CSS
  - XSL to output text, html, any text format

Exercise



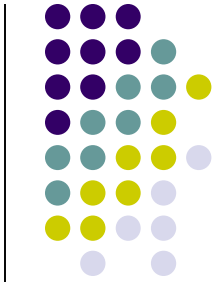
- Create CSS stylesheet
  - * { display: block;}
  - name {font-weight: bold; color: red; }
  - teacher {font-size: 1.5em }

  - name::before {
  - content: "Name: ";
  - }
- In XML, associate with XML stylesheet and change text/XSL to text/CSS.



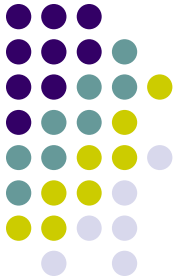
XML IN APPLICATIONS

Reasons and Places for XML



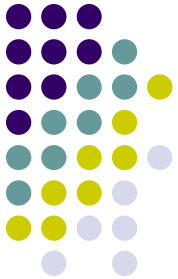
- Data exchange
  - update to EDI
  - common format
  - non-repudiation
- Data formatting
  - use of new schema
  - use of web services

SAX parsers



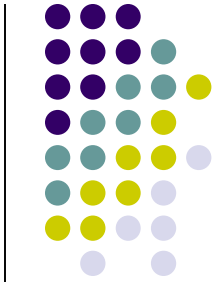
- Runs through XML data but does not remember anything
- Calls functions as it sees matches
- Very fast
- Good for large data files
- Can not start in the middle of file
- Validation error stops processing

DOM parsers



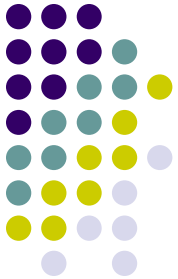
- Allows in memory processing of XML document
  - JavaScript updates HTML in browser
  - random access manipulation in document
- Uses a SAX parser
- Very memory intensive
  - Don’t use files bigger than 2Mb to process
  - Would require 2x2Mb of memory

Web services

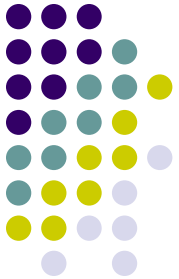


- Transfer of data made easier with standard format
- SOAP is XML compliant document for requesting functions and formatting results
- JavaScript uses XML documents or fragments in JSON protocol
- AJAX returns fragments
- Web services are not practical to look up
- SOAP is declining (WS-*), REST is increasing

Java

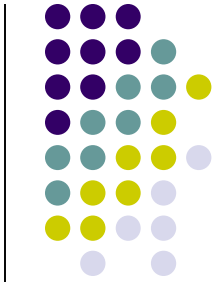


- The Java API for XML Processing (JAXP)
  - the Simple API for XML parsing interface or SAX interface
  - the Document Object Model parsing interface or DOM interface
  - the Streaming API for XML or StAX interface (JDK 6)
- JAXB - Java Architecture for XML Binding (JAXB)
- JAX-WS - Java API for XML Web Services



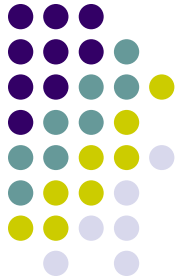
WELL-FORMED XML

Tools / validation



- XML Notepad does not validate with DTDs.
- XML Copy Editor does.
- Download RXP (Windows exe & man page)
  - http://www.cogsci.ed.ac.uk/~richard/rxp.html
  - Place in directory where lab files are.
  - Copy address from Explorer
  - Click: Start/Run…
  - Type: cmd
  - Type: cd <right click – paste>
  - Type: RXP –V <file to validate>

XML document structure



- Two sections
  - Prolog
  - Data section
- Element
  - Open tag with optional attributes
  - Body
    - if only elements – relational style (row+fields)
    - if both text and elements – mixed model (html)
  - Close tag

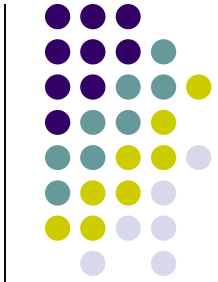
Simple Generic XML Example

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<presentation>
  <slide number="1">
    <name>Introduction to XML</name>
    <contents>XML is ...</contents>
  </slide>
</presentation>
```

Attributes

- Inside of open tag
- Does not affect being well-formed
- Used for metadata for the tool
- Examples
  - <form method=“get”>
  - <person gender ='female'>
  - <gangster name='George "Shotgun" Ziegler'>
  - <gangster name="George "Shotgun" Ziegler">
- Why not this?
  - <person>
    - <gender>female</gender>

Naming rules



- Name start characters
  - A through Z, a through z
  - Underscore
- Following characters
  - Any above
  - 0 through 9
  - Dash, period, colon (not recommended for you)
- Don’t use xml in any case
- No spaces
- Case-sensitive

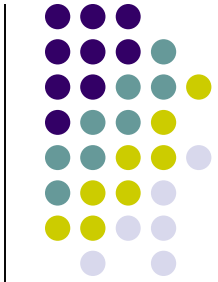
Element vs. Tag vs. Attribute

- **Element** consists of *start tag, optional content* and an *end tag:*
  - <name>Introduction to XML</name>
- **Start tag**
  - <name>
- **Content**
  - Introduction to XML
- **End tag**
  - </name>
- **Start tag** may have **attribute**
  - <slide **number="1"**>

Empty elements

- <aTag> white space here <aTag>
- Empty elements do not contain:
  - Text
  - White space
    - Tabs
    - Returns
    - Spaces
- <tag></tag> = <tag/>

Nesting tags



- Usually referred to as parent-child hierarchies
  - Descendant
  - Ancestor
  - Siblings
- Parents can not overlap
  - Last opened is first closed.
- Top level parent is the root element

XML processing instructions

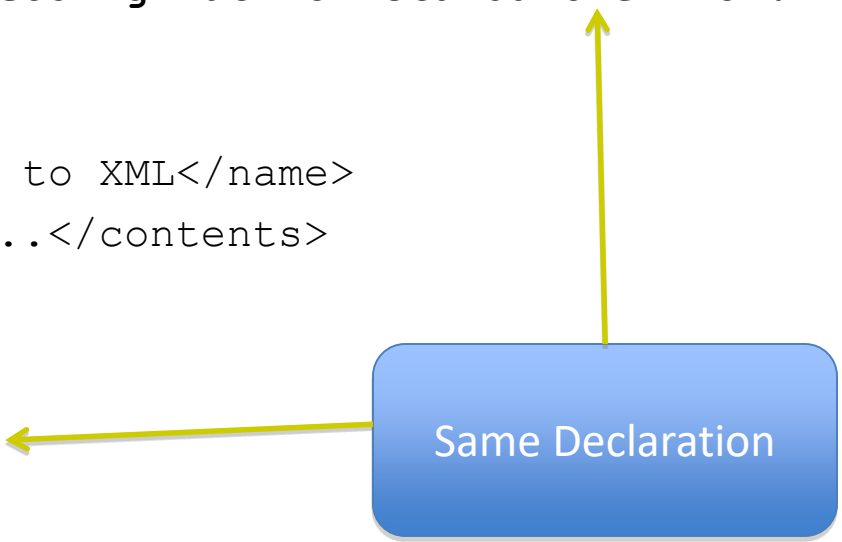
- A PI is not part of the doc. <? Instruction ?>
- XML declaration is a PI
  - <?xml … must be first in doc
- Only a recommendation/assist for metadata
- <?php code in php; ?>

XML Declaration

- XML-declaration is **optional in XML 1.0, mandatory in 1.1**
  - Recommendation: *use it. But don't use 1.1*
- **Version:** 1.0 or 1.1
- **Encoding:** character encoding**,** default utf-8
- **Standalone:**
  - is the xml-document linked to external markup declaration
  - yes: no external markup declarations
  - no: **can have** external markup declaration (open issue..)
  - default: "no"

Comparing Declarations

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<presentation>
  <slide>
    <name>Introduction to XML</name>
    <contents>XML is ...</contents>
  </slide>
</presentation>
```




```
<?xml version="1.0"?>
<presentation>
  <slide>
    <name>Introduction to XML</name>
    <contents>XML is ...</contents>
  </slide>
</presentation>
```

Other XML constructs

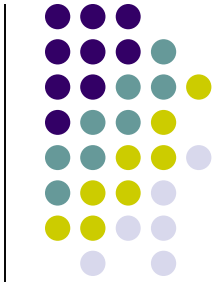
- <!-- Comments -->

- CDATA sections (character data)
  - Turns off XML parsing
  - odd syntax - <![CDATA[data]]>
  - Has been used for JavaScript in older browsers
  - Use when entities should not be used

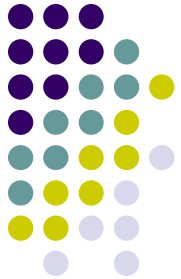
Entity references

- Entity
  - defined in schema
  - &code;
  - XHTML defines nbsp, eacute, etc.
- XML defaults
  - <, >, ", &apo;, &

Character references

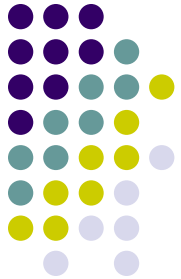


- Character
  - defined in schema
  - &#value;
  - Decimal values= ##
  - Hex values = x#####



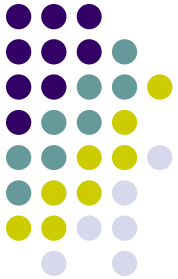
NAMESPACES

Why namespaces?



- Use another library of tag names
- Create unique tag names

Namespace prefixes

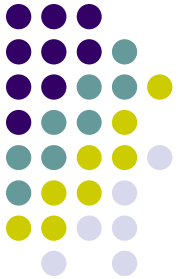


- <namespace:tagname>
  body
  </ namespace:tagname>
- Also prefix:local name
- Qualified name = tag using namespace

Namespace declaration

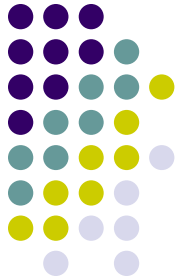
- 1st use of namespace or before must have a attribute definition.
- xmlns:<your namespace name>="some URI that usually looks like a web site address"
- Validating parser will **not** search for file. But you can see if it's there.
  - XML Copy Editor will search for it!
- URI = unique name
- URL = URI that locates a file over internet

Namespace declaration - multiple



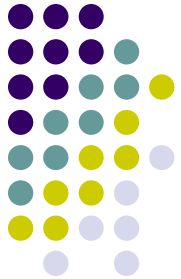
- Namespaces can be used whenever you start using those tags.
- Mixing is OK.
- Use xmlns attribute as needed.

Namespace declaration – root element



- **Best practice** – declare all namespaces in root element for document
- Root element does not have to be from a namespace library.

Namespace declaration - default



- Any tag name without a namespace can be qualified if a default is declared

- Instead of

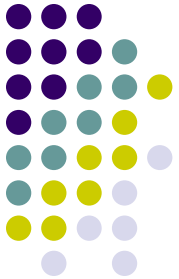
  xmlns:namespace="URI"

  use

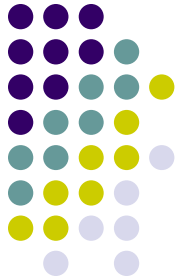
  xmlns="URI"

- Why? For validating parsers.

Exercise

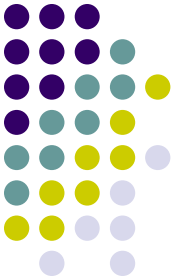


- Find a web site that has some namespace usage



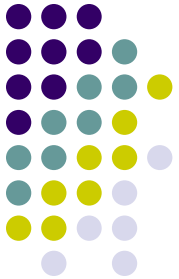
DTDs

DTD writing



- One element definition in any order
- Starts with <!ELEMENT name (
- Uses any one of
  - #PCDATA – for text
  - elementName, elementName, …– for child elements
  - ANY – text and children (mixed model)
  - EMPTY – no body

DTD writing

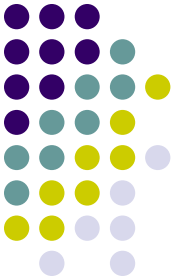


- Document ends with .dtd
- XML doc is linked with <!DOCTYPE …

DTD linking

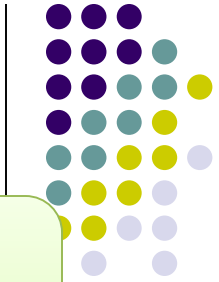
- Internal spec– not practical
- External uses <!DOCTYPE … declaration
  - First declaration is root element name
  - SYSTEM tells the parser to get the file via URI
  - PUBLIC tells the parser to get a file by an id it knows or via URI if necessary
    - A text string is used as a unique id for the DTD
    - An optional quoted file URI is used if necessary

Local external link



- <?xml version="1.0" standalone="no" ?>
- <!DOCTYPE rootElement SYSTEM "howItShouldBeDone.dtd">

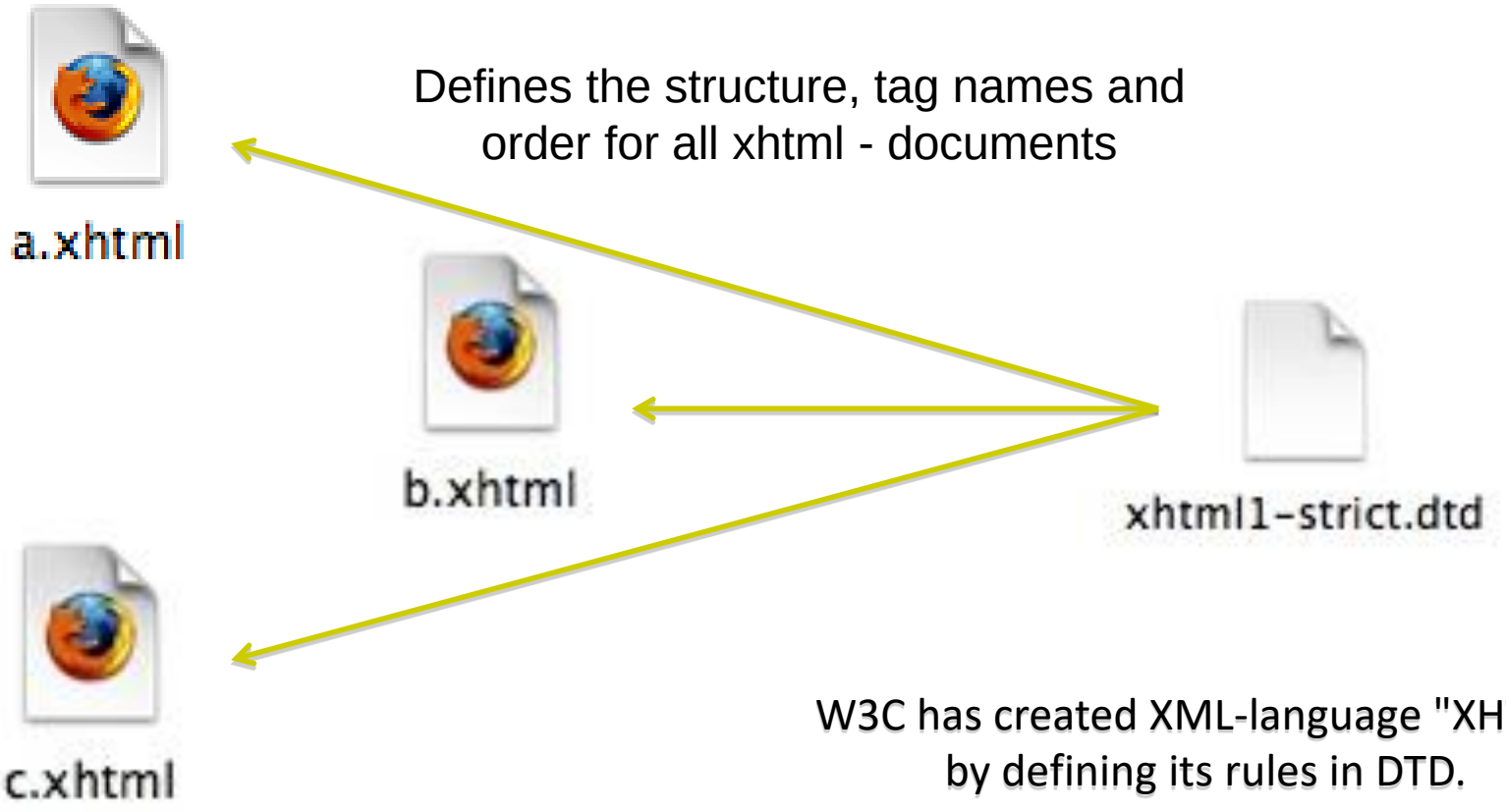
DTD Linking



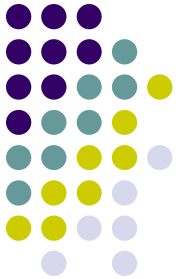
Rules for XHTML elements (order, names, etc)

```
<?xml version="1.0"?>
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
   lang="en">
  <head>
    <title>Minimal XHTML 1.0 Document</title>
  </head>
  <body>
    <p>This is a minimal <a
   href="http://www.w3.org/TR/xhtml1/">XHTML 1.0</a>
    document.</p>
  </body>
</html>
```

DTD Linking - XHTML



Exercise

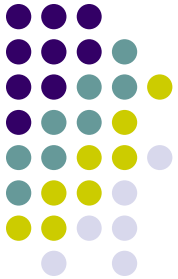


- Grab an RSS feed file
  - http://www.kmbc.com/rss
- Check well-formedness
- Generate a DTD from it
- Associate the DTD with the XML file
- Validate it.
  - What are the problems? Why?
  - Eliminate the problems. Revalidate.

DTD element declaration - !ELEMENT

- Child elements are declared in order
  - <!ELEMENT elementName (1st q, 2nd q, 3rd q) q >
  - Comma separates ordered items
  - Pipe separates choice of items
  - Parentheses help group choices
- Quantifiers
  - ? – zero or one
  - * - zero, one, or more
  - + - one or more
  - Used after element or group

DTD attribute declaration - !ATTLIST



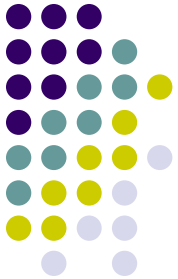
- To describe attributes of an element
  - <!ATTLIST elementName
    attributeName1 type rule [default]
    attributeName2 type rule [default]
    …
    >
- Type = CDATA or a choice from a group
  - (1st | 2nd |3rd)
- Rule – cardinality (action)
  - #REQUIRED, #IMPLIED, #FIXED
- Default = optional quoted text or quoted choice to be used when data is not present
  - “1st”

DTD attribute declaration – predefined types

- ID – primary key
- IDREF – foreign key

- NMTOKEN – not used

DTD attribute declaration – action rules or cardinality



- #REQUIRED
- #IMPLIED – optional attribute
- #FIXED -

DTD entity declaration – !ENTITY general

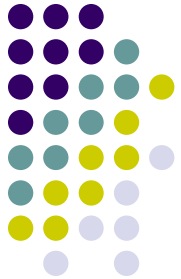
- Defines an abbreviation for XML data like &abbreviation;
- Internal entity – small chunk
  - <!ENTITY abbreviation “expanded text”>
- External entity – large chunk (include)
  - <!ENTITY abbreviation SYSTEM “URI”>

DTD entity declaration – !ENTITY parsed parameterized

- Defines an abbreviation for DTD declarations like %abbreviation;

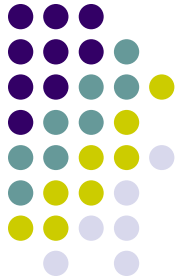
- Internal entity – small chunk
  - <!ENTITY % abbreviation “expanded text”>
- External entity – large chunk (include)
  - <!ENTITY % abbreviation SYSTEM “URI”>

DTDs and namespaces

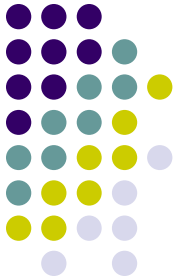


- DTDs preceded namespaces and were not updated.
- Namespaces are seen as attributes
- Use an xsd

Exercise



- Generate DTD
- Add association to XML for DTD
- Validate
- Remove a required field and validate again.

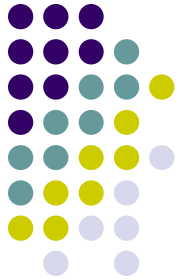


XSD VALIDATION

XSD overview

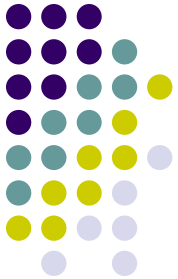
- Newer validation rule document from W3C
- More datatypes
- Better quantifiers
- Element defaults
- Element enums (restricted choices)
- XML document
- Namespace aware
- No entities

XSD writing



- XML syntax
- Namespace – xsd commonly used
  - <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" >

XSD writing



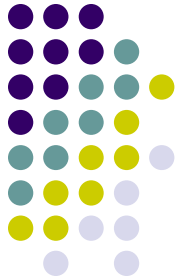
- Root element – schema
- Schema child elements - globals
  - element - predefined data types
  - simpleType – modified element
  - complexType – groups of elements or simpleTypes

Linking XML data to XSD

- Requires attributes on root element instead of !DOCTYPE declaration

- No namespaces used
  - <rootElement nameSpaceDeclaration noNameSpaceSchemaLocationDeclaration />

Built in element types



- string, normalizedString, token
- base64Binary, hexBinary
- integer, positiveInteger, negativeInteger, nonNegativeInteger, nonPositiveInteger, long, unsignedLong, int, unsignedInt, short, unsignedShort, byte, unsignedByte
- decimal, float, double
- boolean
- duration, dateTime, date, time
- gYear, gYearMonth, gMonth, gMonthDay, gDay
- Name, Qname, NCName
- anyURI
- language

Simple element types - declaration

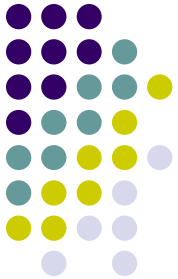
- Top level element – name the type
  - <xsd:simpleType name=“stName”>
- Child element of type – declare rule type
  - <xsd:restriction base=“xsd:int”>
  - <xsd:restriction base="xsd:string">
- Child element of rule – declare facet
  - <xsd:minInclusive value="10000"/>
  - <xsd:minLength value=“1” />
  - <xsd:enumeration value="AK"/>

Complex element types – ordered declaration

- Groups simple and complex types in to one named type (think class or table)

- Element name – **xsd:complexType**
- Child element – **xsd:sequence**
  - must be in the order declared
- Grandchild elements – **xsd:element**
  - Attributes
    - name="attributeName"
    - type="primitiveTypeNameOrComplexTypeName"
- Last child – **xsd:attribute**

Complex element types – unordered declaration



- Element name – **xsd:complexType**
- Child element – **xsd:all**
  - permits any order of elements
- Grandchild elements – **xsd:element**
  - Attributes
    - name="attributeName"
    - type="primitiveTypeNameOrComplexTypeName"
- Last child – **xsd:attribute**

Complex element types – choice of one declaration

- Element name – **xsd:complexType**
- Child element – **choice**
  - permits only one of descendant elements to be a child
- Grandchild elements – **xsd:element**
  - Attributes
    - name="attributeName"
    - type="primitiveTypeNameOrComplexTypeName"
- Last child – **xsd:attribute**

Element attribute declarations

- Element name – **xsd:attribute**
- Attributes
  - name="attribute name"
  - type = "xsd:simpleType"
  - use = "required | optional | prohibited"
  - default = "textString" (use="optional")
- Position is last child in complexType element
- Enumerated choices requires definition of a simpleType

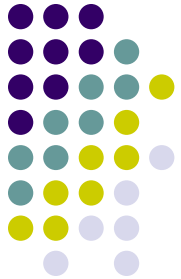
Enumerated choices type

- Element name - **xsd:simpleType**
  - Attribute: name="choicesName"
- Child element – **xsd:restriction**
  - Attribute: base="xsd:primitiveDataType"
- Grandchild elements – **xsd:enumeration**
  - Attribute: value="itemValue"

Named and unnamed types

- Named types
  - have a **name and type** attribute
  - may not need a name because they are used just once.
- Unnamed or anonymous types
  - lack name and type attribute
  - create a more readable xsd
  - are declared where they are needed as child elements

Anonymous type declaration



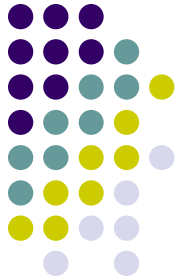
- <xsd:attribute type="referencedType" … />

- <xsd:simpleType name="referencedType">
  - children
- </xsd:simpleType />

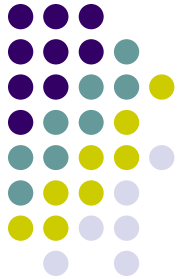
- **is the same as**

- <xsd:attribute …>
  - <xsd:simpleType>
    - children
  - </xsd:simpleType>
- </xsd:attribute>

Generate xsd from xml



- Use Microsoft's xsd.exe
  - Google - XML Schema Definition Tool
  - Locations
    - C:\Program Files (x86)\Microsoft SDKs\Windows\v8.0A\bin\NETFX 4.0 Tools
      C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0\Bin\xsd.exe
- Generate xsd from xml
  - Type: xsd <fileName>.xml
  - File created will be <fileName>.xsd
- Not as good as online versions

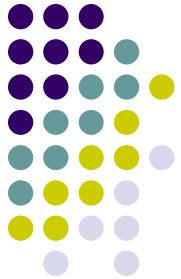


XSD WITH NAMESPACES

Qualified XML

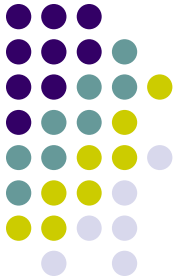
- Schemas (xsd files) validate xml data.
- Data is qualified if a namespace if used.
  - Prefixed elements refer back to the prefix declaration
  - Unprefixed elements rely on a default namespace declaration.
  - Attributes don’t pick up the default namespace
- Data is unqualified when no default namespace is declared and elements are not prefixed.
  - Unqualified data needing validation use the noNameSpaceSchemaLocation attribute

XML data linking to XSL with namespace



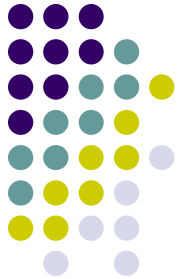
- Add attributes to root element
- Use the schema namespace
  - xmlns:xsi=http://www.w3.org/2001/XMLSchema
  - xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance

XML data linking to XSL with namespace



- Locate the schema for this document
  - xsi:schemaLocation="<validatingFile>.xsd">
- Declare (or not) a document namespace
  - <rootElement xmlns="URI" …
  - <xyz:rootElement xmlns:xyz ="URI" …

Namespace URIs – URL or URN



- URLs
  - Universal Resource Locator
  - Looks like a web address
  - People expect a file at this address
  - Doesn’t require a file to be present
  - Parser doesn’t look up file
- URN
  - Universal Resource Name
  - Doesn’t look like a file
  - Needs corporate support to be resolved

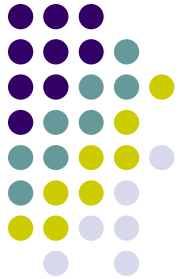
XSL namespace declarations 1

- Root element - xs:schema
  - xs is the short convention, use any you want
- Declare namespace the same as xml data
  - Default - xmlns=“URI"
  - Explicit - xmlns:xyz=“URI”
- Declare schema namespace
  - Default – xmlns=“URI"
  - Explicit - xmlns:xs=“http://www.w3.org/2001/XMLSchema”

XSL namespace declarations 2

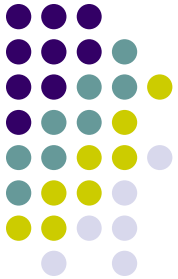
- Bind XSD to namespace
  - targetNamespace="URI"
  - namespace of xml elements will use same URI
  - xml data file will use same URI
- Choose combination of namespace declarations
  - one default, one explicit (more common)
  - or both explicit

Controlling element and attribute qualification

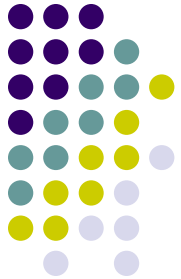


- xsd decides if you must use namespace in data for non-global elements
- Globals (direct children of xs:schema) must always be qualified
- Entire xsd
  - elementFormDefault = “qualified” (common)
  - attributeFormDefault = “qualified”
  - defaults are “unqualified”
- One element
  - use attribute form=“qualified” or “unqualified”

Splitting an xsd into multiple files



- Better for team development
- All schemas use same target namespace
- <xs:include… will merge files
  - must follow root element
- <xs:import… will merge files with namespaces



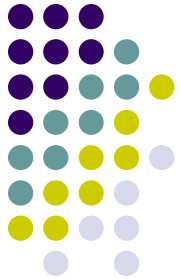
Scripting with XML

XSLT

XSL

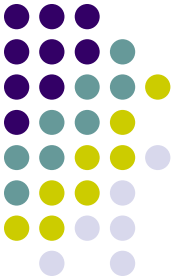
- e**X**tensible **S**tylesheet **L**anguage
  - XSLT - transformations
  - XSL-FO - formatting objects
- A complete programming language to transform an XML document to any other format

XSL files



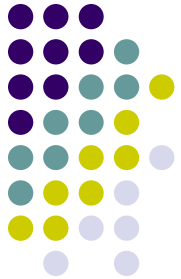
- The data
  - an XML compliant file
- The program (stylesheet)
  - what processing to do written in either XSLT or XSL-FO
- The result
  - a name of the processed version
- An FO processor takes XSLT pre-processed data and produces an Acrobat (pdf) document. It’s a typesetting engine.

Demo



- https://github.com/doughoff/JV-394-XML
  - garage.xml
  - garage.xslt
- Place both files in the same directory and process with XML Copy Editor

XSLT processors



- An XSLT processor is a scripting language engine.
- XSLT processors
  - exist in most languages
  - are built in to some browsers
  - Saxon, Xalan, MSXML, Sablotron, xsltproc, XT, 4XSLT

Browsers

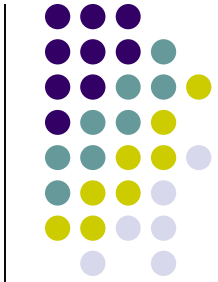
- Chrome removed XSLT/Xpath processing from Blink
  - web authors prefer DOM manipulation with JavaScript.
  - used more in a pipeline on server
- IE
- Firefox

XSLT syntax - <stylesheet>

- Root element is <stylesheet>
- Namespace convention is xsl

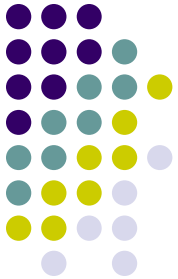
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform”
- Version attribute is required.
  - version=“2.0”
  - XSLT 2.0 is used with XPath 2.0

XSLT syntax - <template>



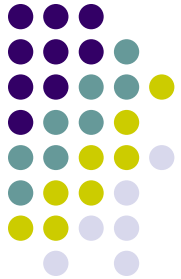
- A rule in XPath about position of processing
- Define rule in attribute match
  - match = “XPath expression”
- Body of template element is copied to result document
- Processing starts with match=“/”

XSLT syntax - <value-of>



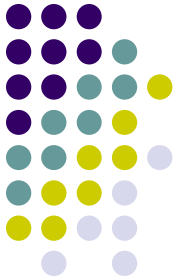
- Retrieves xml body of elements
  - concatenates bodies of child elements
- Select attribute limits body
  - select = “XPath expression”

XSLT syntax - <apply-templates>



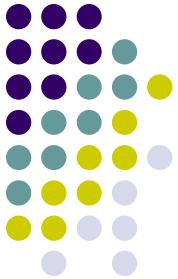
- A way to make a function call
- Child of <template>
- Attribute select is input to function
- Template selected to run will match nodes with attribute match
- Each matched item will run in the template
  - XPath is relative based on matched node

XSLT syntax - <output>



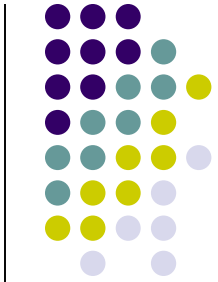
- Type of document being created
- Child of <stylesheet> root element
- Types
  - xml – default
  - html – default only if result starts with <html>
    - to create XHTML, use xml output type
    - HTML can be non-compliant

Exercise



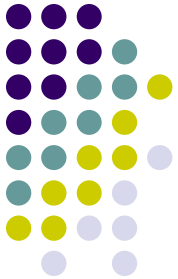
- https://github.com/doughoff/JV-394-XML
  - people.xml
  - people.xslt
- Associate the XSL
  - only needs the relative path, will not work well with full file:// path
- Transform and save the result
- View in browser

Exercise



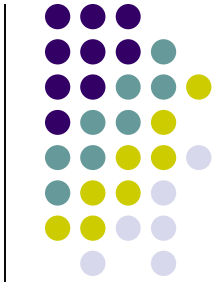
- garage1.xslt
- garage2.xslt
- garage3.xslt

Tutorial

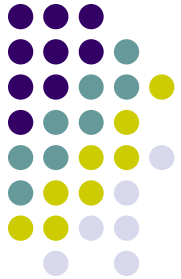


- https://www.w3schools.com/xml/xsl_transformation.asp

XSL-FO and PDFs



- Apache FOP
  - http://xmlgraphics.apache.org/fop/
- JavaScript
  - https://parall.ax/products/jspdf (70k ?)



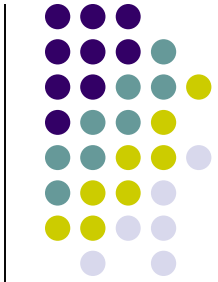
Query language for XML

XPATH

XPath

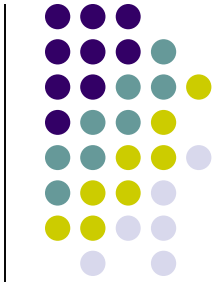
- A DSL for searching and selecting elements
- Elements are nodes when in memory
  - Many other parts of XML are nodes also (attributes, PIs, etc.)
  - Whitespace can optionally be a node – can be controlled
- Used in JavaScript and commonly supported in major languages
  - jQuery selectors with compatibility plugin - http://plugins.jquery.com/?s=xpath

Basic syntax



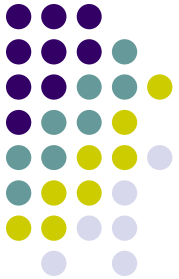
- Used like hrefs in a link
- Absolute references start with /
  - All other references are relative to current position of processing
  - /root/branch/twig
  - twig/fruit (assuming context node of branch)
- Wildcard * will select all elements
  - /* selects all elements

Basic syntax



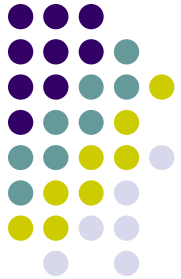
- // means anywhere in document
  - //dog finds any dog element
- Attributes are marked with @
  - //@name finds any element with a name attribute
  - @* finds all attributes of the current node
- Filters or predicates follow node description in brackets
  - /[only if this]

Exercise



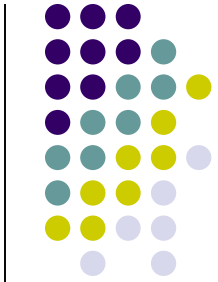
- Garage
  - //car
  - //car/maintenance
  - //make
  - //@miles

XPath Functions



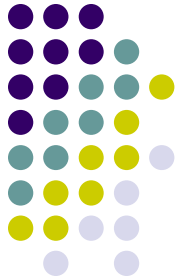
- Apply additional logic to node sets
  - position( ) = #
  - last( )
  - count( )
  - format-number( )
  - round( ), ceiling( ), floor( )
  - substring-after( ), substring-before( )
  - translate( )
  - sum( )

Exercise



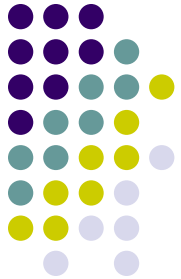
- Garage.xml
  - //car[last()]
  - count(/garage/*)
  - sum(//@miles)

XPath Definitions



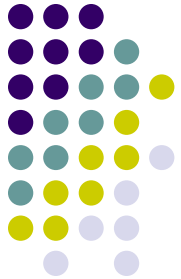
- node
  - a discrete part that can be addressed.
  - A root node (with prolog, root element and its children, comments, and PIs following the root element), an element, an attribute, a comment, a PI, a namespace, or text nodes.
  - The root node can be represented as one slash.

XPath Definitions



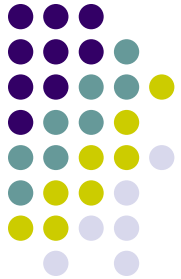
- node-set
  - a group of nodes returned by an XPath expression.
- context node
  - the node from where future directions are taken from. Similar to relative addressing from where you are currently.

XPath Definitions



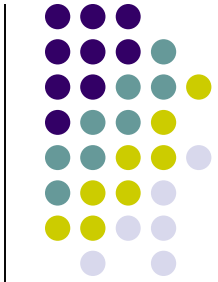
- location path
  - the set of instructions in XPath of how to get somewhere. If it's relative, starting with the context node, it has steps, separated by slashes.
  - A location step has the syntax of axis::nodetest[predicate]

XPath Definitions



- axis
  - This says in which direction you want to move or look at from the context node.
  - Choices are child, parent, attribute, ancestor, ancestor-or-self, descendant, descendant-or-self, following, following-sibling, preceding, preceding-sibling, namespace, and self.
  - In XSLT <xsl:template match> expressions you use child and attribute axes only.

Axis shortcuts

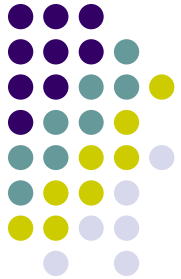


- self::node()
  - .
- parent::node()
  - ..
- child::
  - default when no axis is declared
- descendant-or-self::node()
  - //
- attribute::
  - @

XPath Definitions

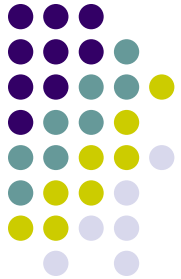
- **nodetest**
  - where you want to stop looking along the axis you have specified. Can be a name, a node type followed by parentheses, or the syntax *processing-instruction(“some name”)*, where you put the name in “some name” of the PI that you want or an asterisk to represent any node. If given, the string value will be set to the first found node’s string values.
  - Functions are node( ), text( ), comment( ), and processing-instruction( ).

XPath Definitions



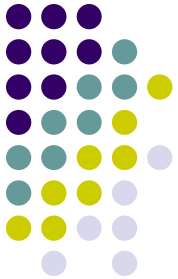
- **predicate**
  - a Boolean test when applied to each node in the currently selected node-set.. Operators include = , !=, <, >, <=, or >=.

General functions



- Boolean functions
  - not( ), true( ), false( ), boolean( ), lang( )
  - or
  - and
- String functions
  - string( ), concat( ), starts-with( ), contains( ),
  - substring( ), substring-before( ), substring-after( ),
  - string-length( ), normalize-space( )
  - translate( )

Exercise

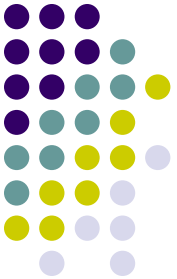


- Test using two_gent.xml from https://github.com/doughoff/JV-394-XML
- **the play**
  - /PLAY
  - //PLAY
  - ./PLAY
  - PLAY
  - /descendant::PLAY
  - descendant::PLAY
  - /descendant-or-self::PLAY

Exercise

- **the entire description of people in the play – enclosing element(s) only**
  - /child::PLAY/child::PERSONAE
  - /PLAY/PERSONAE
- **the entire description of people in the play – all child elements**
  - /PLAY/PERSONAE/*
- **the minor characters in the play**
  - /PLAY/PERSONAE/PERSONA

Exercise

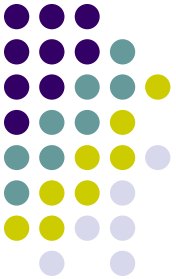


- **all of the characters in the play**
  - //PERSONA
  - /PLAY//PERSONA
- **the speeches of any speaker**
  - //SPEECH/SPEAKER/..
- **the speeches of Valentine**
  - //SPEAKER[self::node()="VALENTINE"]/..
  - //SPEAKER[.="VALENTINE"]/..
- **find any speeches not by a speaker**
  - //SPEECH[not(./SPEAKER)]

Exercise

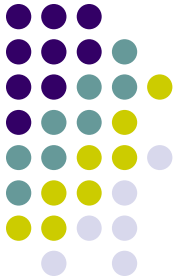
- **how many times does Valentine speak?**
  - count(//SPEAKER[.="VALENTINE"])
- **how many times does Valentine or Proteus speak?**
  - count(//SPEAKER[.="VALENTINE"] | //SPEAKER[.="PROTEUS"])
- **subtopics of the play (any element nested two deep)**
  - /*/*

Exercise



- **everything (any element starting from anywhere)**
  - //*
- **the first character of the play**
  - //PERSONAE[1]/PERSONA[1]
- **the last character of the play**
  - //PERSONAE[last( )]/PERSONA[last( )]
- **any element that has an attribute of type (none in this example – add one to test)**
  - //*[@type]

Exercise



- **any element that has an attribute of type set to data**
  - //*[@type=”data”]
- **any element that has an attribute of type set to data that may include spaces before or after the data**
  - //*[normalize-space(@type)=”data”]
- **any element that has any attribute (none in this example)**
  - //*[@*]

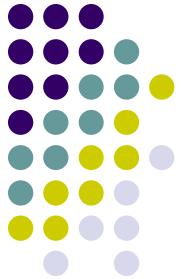
Exercise

- **any element that does not have an attribute**
  - //*[not(@*)]
- **any attribute of type**
  - //@type
- **any attribute of any kind**
  - //@*

Scalable Vector Graphics

SVG

SVG

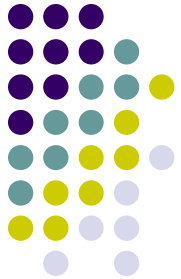


- Vector, not bitmap
  - Similar to PostScript
- Declarative
  - <svg version="1.0" xmlns="http://www.w3.org/2000/svg">
  - <circle cx="100" cy="75" r="75" fill="blue" stroke="#FF0000" stroke-width="1px"></circle>
  - </svg>

HTML/CSS

- Browser renderable
- Uses CSS rules
  - color, fill, fill-opacity, font, kerning, letter-spacing, opacity, stroke, stroke-dasharray, stroke-opacity, stroke-width, transform, word-spacing

CSS

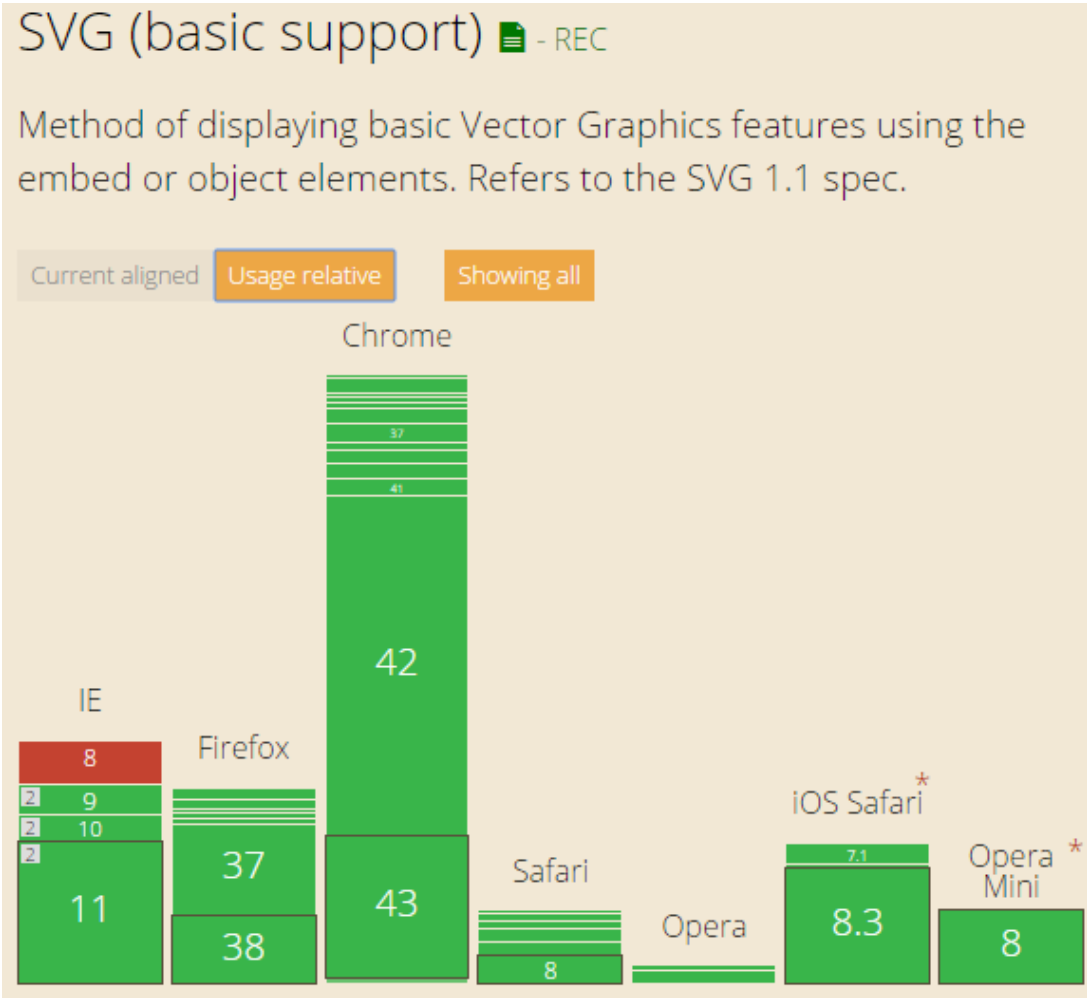


- <circle cy="100" cx="200" r="30" class='redfill stroked'/>
- .redfill{ fill: red; }
- .stroked {stroke: darkgray; stroke-width: 4px; }

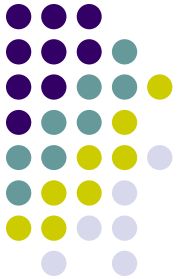
SVG support

- http://caniuse.com/#search=svg

- <svg style="width:500px;height:500px;border:1px lightgray solid;">
- <path d="M 10,60 40,30 50,50 60,30 70,80“ style="fill:black;stroke:gray;stroke-width:4px;" />
- <polygon style="fill:gray;“ points="80,400 120,400 160,440 120,480 60,460" />
- <g>
- <line x1="200" y1="100" x2="450" y2="225“ style="stroke:black;stroke-width:2px;"/>
- <circle cy="100" cx="200" r="30"/>
- <rect x="410" y="200" width="100" height="50“ style="fill:pink;stroke:black;stroke-width:1px;" />
- </g>
- </svg>



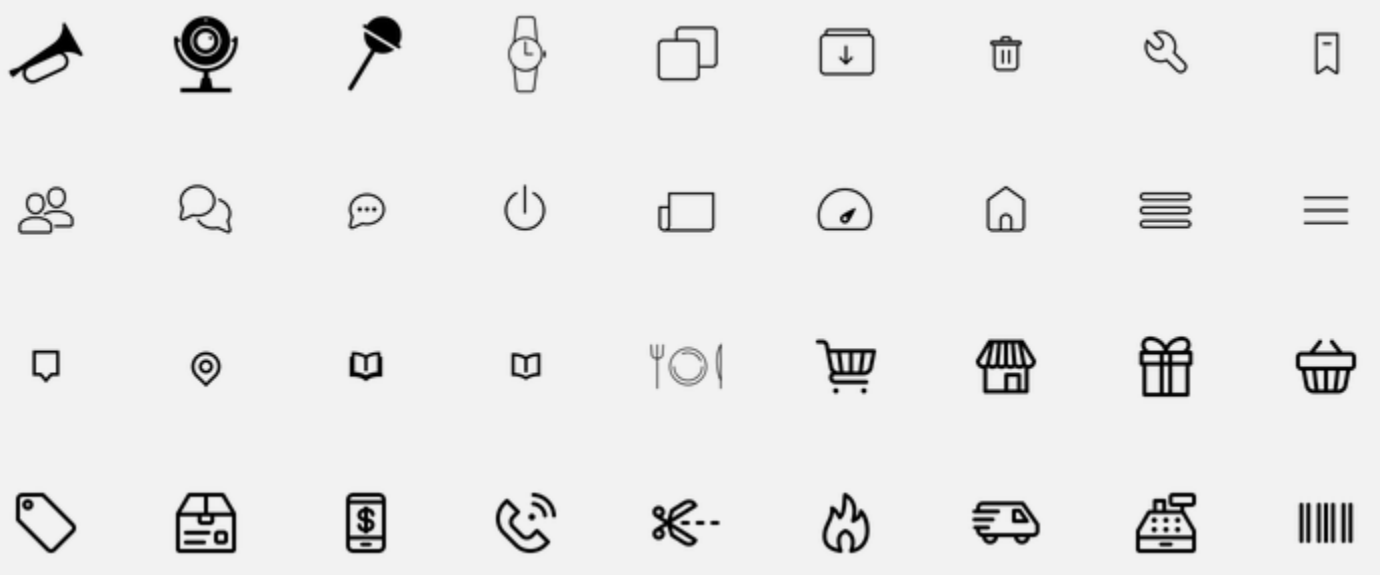
SVG vs Canvas



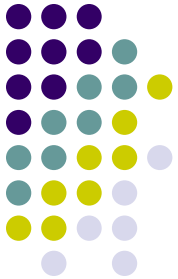
- SVG
  - XML based with selectable nodes
  - use CSS to style
  - use events, filters, animation, etc.
  - vector based
- Canvas
  - requires JavaScript code
  - is faster to render
  - 3D
  - bit-mapped, poor text rendering

SVG Images

- https://thenounproject.com/
  - PNG and SVG
  - requires account

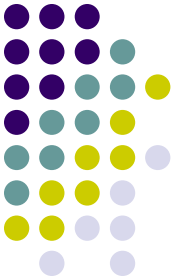


SVG tools



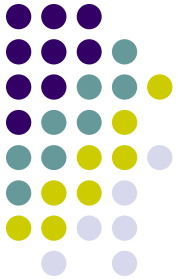
- D3 – library to create SVG for charts, not a charting package. https://d3js.org/
  - New York Times examples
    - https://bost.ocks.org/mike/
- Editors
  - Google SVG-edit - https://code.google.com/p/svg-edit/
  - Inkscape - https://inkscape.org/en/
  - Snap - http://snapsvg.io/
    - a simpler way to create graphics
    - newer version of Raphael, IE9+

Resources



- https://developer.mozilla.org/en-US/docs/Web/SVG
- SVG on the Web - A Practical Guide - https://svgontheweb.com/
- CSS reference - http://tympanus.net/codrops/css_reference
- http://projects.fivethirtyeight.com/flights/
- Quartz Chartbuilder - http://quartz.github.io/Chartbuilder/

Resources



- http://tutorials.jenkov.com/svg/index.html
- Codrops examples
  - https://tympanus.net/codrops/?s=svg&search-type=posts
- Free icons
  - https://icomoon.io
  - http://www.flaticon.com/