XML and JSON Programming (Part 1)

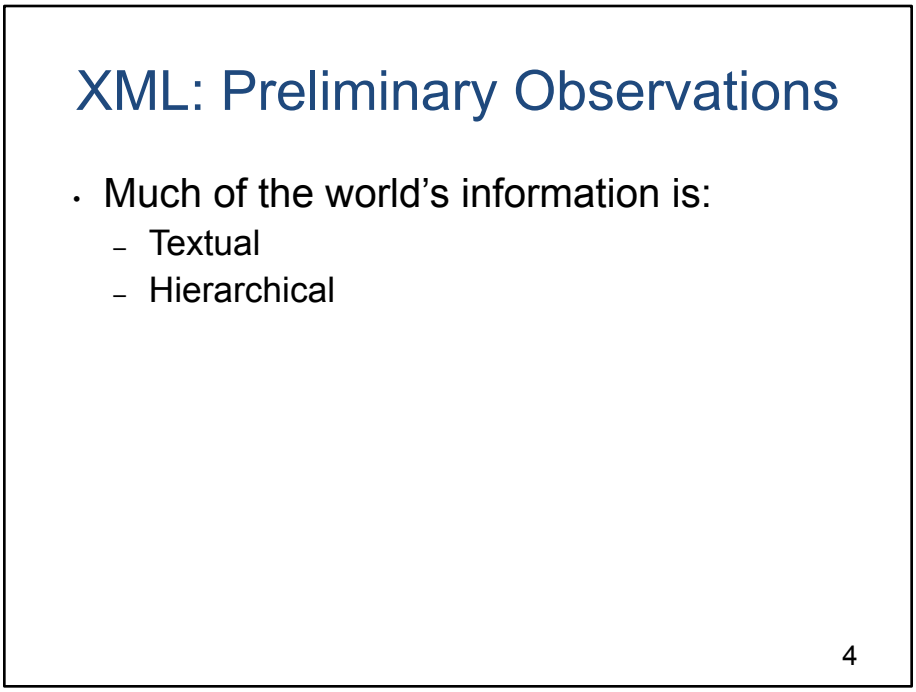
Robert M. Dondero, Ph.D.
Princeton University

Objectives

- We will cover:
  - XML
  - XML programming: DOM
  - XML programming: SAX
  - JSON
  - JSON programming
  - XML/JSON and AJAX

Agenda

- **XML**
- XML programming: DOM
- XML programming: SAX



XML: Preliminary Observations

Much of the world's information is:

- Textual
  - Books (mostly)
  - Web pages (mostly)
  - Email (mostly)
- Hierarchical
  - Recall Library of Congress and Dewey Decimal book classification systems

XML: Preliminary Observations

- And, by the way...
  - See *The Sciences of the Artificial* by Herbert Simon
  - Many natural systems are structured as ***nearly decomposable hierarchies***

XML

- ***XML (Extensible Markup Language)***
  - A language for expressing **textual** documents in **hierarchical** (tree-structured) form
  - Worldwide Web Consortium (W3C) specification
  - Similar to HTML…

XML

- Major differences between XML and HTML
  - In XML ***empty elements*** must be expressed using self-closing tags
    - Empty element: has start tag and no end tag
    - `<hr>`
      - Well formed in HTML
      - Malformed in XML
    - `<hr />`
      - Well formed in HTML
      - Well formed in XML

XML

- Major differences between XML and HTML
  - In XML element nesting must be proper
    - `<strong>…<em>…</strong>…</em>`
      - Well formed (or at least acceptable) in HTML
      - Malformed in XML
    - `<strong>…<em>…</em>…</strong>`
      - Well formed in HTML
      - Well formed in XML

XML

- Major differences (cont.):
  - In XML attribute values must be quoted
    - `<span id=myspan>Something</span>`
      - Well formed in HTML
      - Malformed in XML
    - `<span id="myspan">Something</span>`
      - Well formed in HTML
      - Well formed in XML

XML

- Major differences (cont.):
  - XML allows processing instructions
    - `<? ProcessingInstruction ?>`
    - Provides information to XML processor about how to handle the XML document

XML

- Major differences (cont.):
  - In HTML, the set of tags is predefined
  - **In XML, you define the tags!**

XML: Example

- See **books.html**
  - Tag set is predefined
- See **books.xml**
  - Tag set is **not** predefined

XML: Elements vs. Attributes

Design decision: When to use elements vs. attributes

```
<book>
   <author>Kernighan</author>
   <title>The C Programming Language</title>
   <price>
      <currency>dollars</currency>
      <quantity>24.99</quantity>
   </price>
</book>
```

Element-heavy

```
<book>
   <author>Kernighan</author>
   <title>The C Programming Language</title>
   <price currency="dollars" quantity="24.99" />
</book>
```

Attribute-heavy

XML: Elements vs. Attributes

- Heuristic:
  - **Elements** should **contain** data
  - **Attributes** should **describe** data

```
<book>
   <author>Kernighan</author>
   <title>The C Programming Language</title>
   <price currency="dollars">24.99</price>
</book>
```

XML: Elements vs. Attributes

- Heuristic:
  - Favor elements over attributes
    - Elements are easier to read, and so...
    - Elements are easier to maintain

XML: vs. HTML

- Some theory:
  - ***Content***: The document's raw data
  - ***Semantic structure***: The **organization** of the content
  - ***Presentation***: The **rendering** of the content

XML: vs. HTML

- HTML
  - Tags define **presentation**
  - **Semantic structure** is absent
- XML
  - Tags define **semantic structure**
  - **Presentation** is absent

XML: vs. HTML

- books.html
  - Tags define **presentation**
    - The book list should be rendered as a table with three columns…
  - **Semantic structure** is absent
    - What is Kernighan? Kernighan is a *td* within a *tr* within a *table* within a *body* within a *html*???

XML: vs. HTML

- books.xml
  - Tags define **semantic structure**
    - What is Kernighan? Kernighan is an *author* of a *book* within a list of *books*
  - **Presentation** is absent
    - No indication of how to render the book list

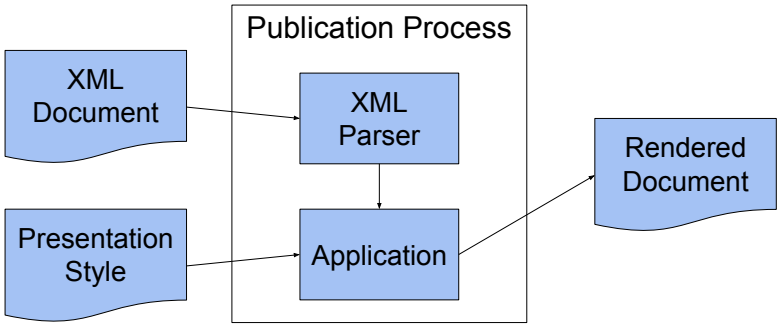
XML: Advantages

- Advantages of XML over HTML
  - Semantic structure is present
    - Document is easier to analyze mechanically
    - Document yields more useful information
    - Document has more potential uses
  - Presentation is absent
    - Same document can be presented in multiple ways

XML: Applications

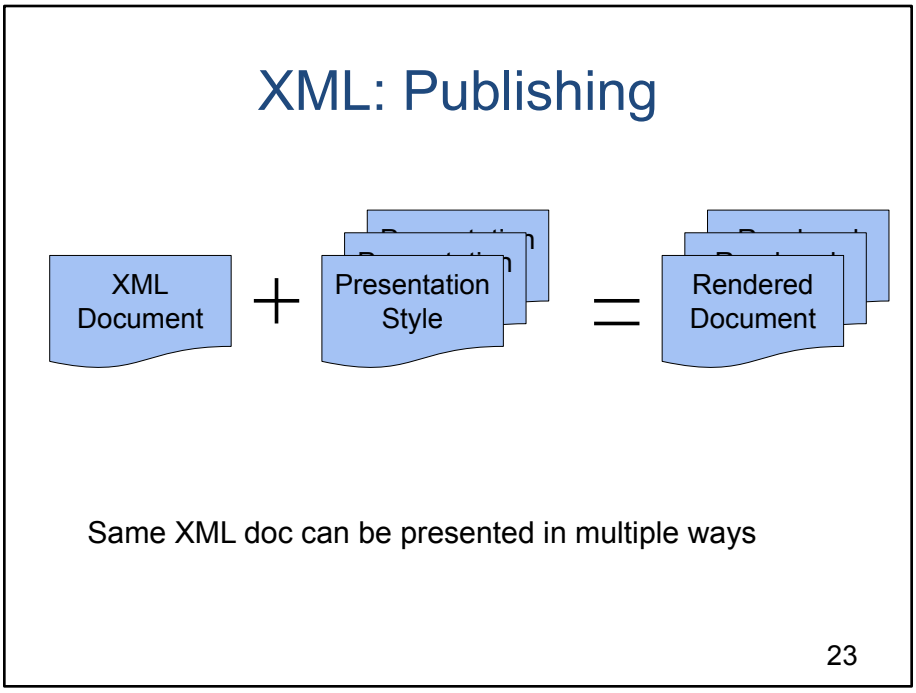
- Applications of XML:
  - Publishing
  - Data communication

XML: Publishing



Publisher can store XML docs separate from presentation styles

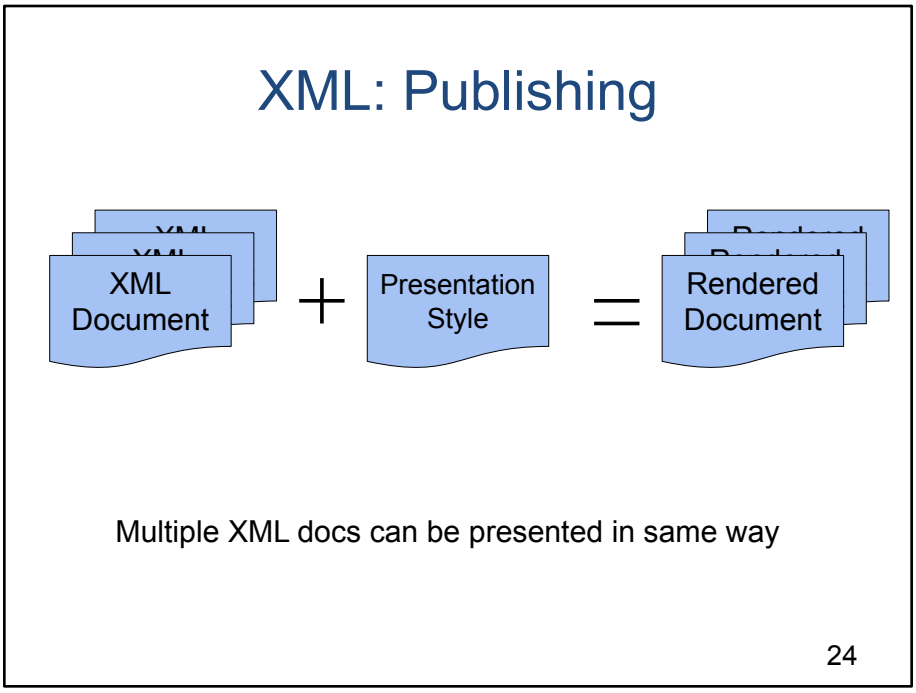
(Presentation styles can be represented using **XSL** scripts)



XML: Publishing

Example: Publisher can present same document as
- HTML (for browsers)
- Latex (for print media),
- ePub (for e-readers)

Same XML doc can be presented in multiple ways
Benefit: Eliminate content redundancy



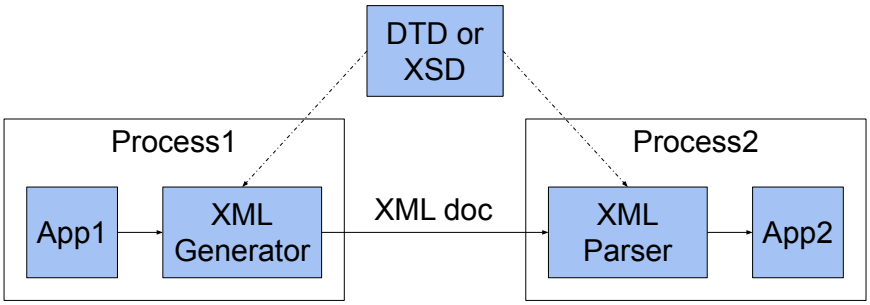
XML: Publishing

Example: Publisher can present book submitted by author1 using same format as book submitted by author2

Multiple XML docs can be presented in same way

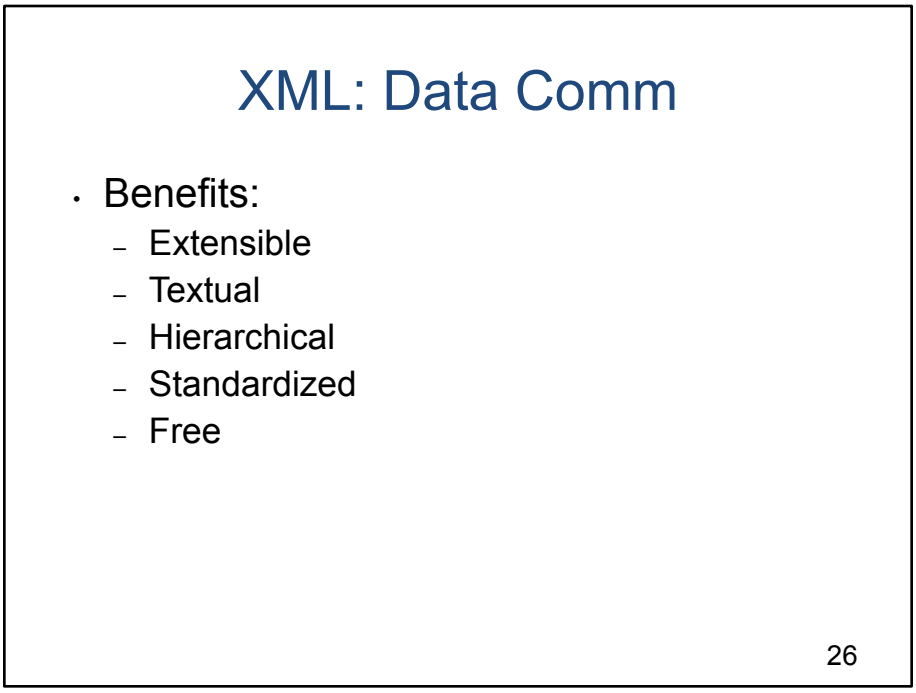
Benefit: Automated branding (look-and-field)

XML: Data Comm



XML is convenient for data comm

Can use a **DTD** (**D**ocument **T**ype **D**efinition) or **XSD** (**X**ML **S**chema **D**efinition) to define the comm protocol



XML: Data Com

Benefits

Extensible

Using DTD or XSD you define valid XML docs, and thereby you define the comm protocol

Textual

Human readable; easy to debug

Hierarchical

Natural for many domains

Standardized

The mechanisms for defining protocols (DTDs and XSDs) are standardized

Business communities define standard DTDs and XSDs (publishing, pharm, …)

Free

XML generators and parsers are available publicly

XML: Example Programs

- Examples in this lecture:
  - In Python
    - Appropriate for XML programming in the **server-side** of a Web app
  - In JavaScript
    - Appropriate for XML programming in the **client-side** of a Web app (i.e., in a browser)

XML: Example Programs

- To run the example **JavaScript** programs in this lecture:
  - `npm install xmldom`
  - `npm install xmlserializer`
  - `npm install sax-parser`

Agenda

- XML
- **XML programming: DOM**
- XML programming: SAX

DOM Programming

- **World Wide Web Consortium (W3C)** defines two standard APIs:
  - ***DOM***: The Document Object Model
  - ***SAX***: The Simple API for XML
- (W3C also defines DOM API for HTML)
- Programs use the APIs to create/use XML documents

DOM Programming

- A DOM parser:
  - Parses given XML doc in its entirety
  - Builds an in-memory tree of objects/nodes
    - Each XML element is represented by an object/node in the DOM tree

DOM Pgmming: Examples

- **writedom** programs
  - The job:
    - Write entire DOM of any given XML doc

DOM Pgmming: Examples

- See **writedom.py**
  - Recursive pre-order traversal
  - Run on books.xml…

DOM Pgmming: Examples

```
$ python writedom.py books.xml
Document: #document=None
   Comment: #comment=
============================================================
   Comment: #comment= books.xml
   Comment: #comment= Author: Bob Dondero
   Comment: #comment=
============================================================
   Element: books=None
      Text: #text=WHITESPACE
      Element: book=None
         Text: #text=WHITESPACE
         Element: author=None
            Text: #text=Kernighan
         Text: #text=WHITESPACE
         Element: title=None
            Text: #text=The Practice of Programming
         Text: #text=WHITESPACE
         Element: price=None (Attribute: currency=dollars)
            Text: #text=40.74
         Text: #text=WHITESPACE
      Text: #text=WHITESPACE
```

Continued on the next page

DOM Pgmming: Examples

```
      Element: book=None
         Text: #text=WHITESPACE
         Element: author=None
            Text: #text=Kernighan
         Text: #text=WHITESPACE
         Element: title=None
            Text: #text=The C Programming Language
         Text: #text=WHITESPACE
         Element: price=None (Attribute: currency=dollars)
            Text: #text=24.99
         Text: #text=WHITESPACE
      Text: #text=WHITESPACE
      Element: book=None
         Text: #text=WHITESPACE
         Element: author=None
            Text: #text=Sedgewick
         Text: #text=WHITESPACE
         Element: title=None
            Text: #text=Algorithms in C
         Text: #text=WHITESPACE
         Element: price=None (Attribute: currency=dollars)
            Text: #text=61.59
         Text: #text=WHITESPACE
      Text: #text=WHITESPACE
$
```

Note: DOM contains many nodes that you might not expect

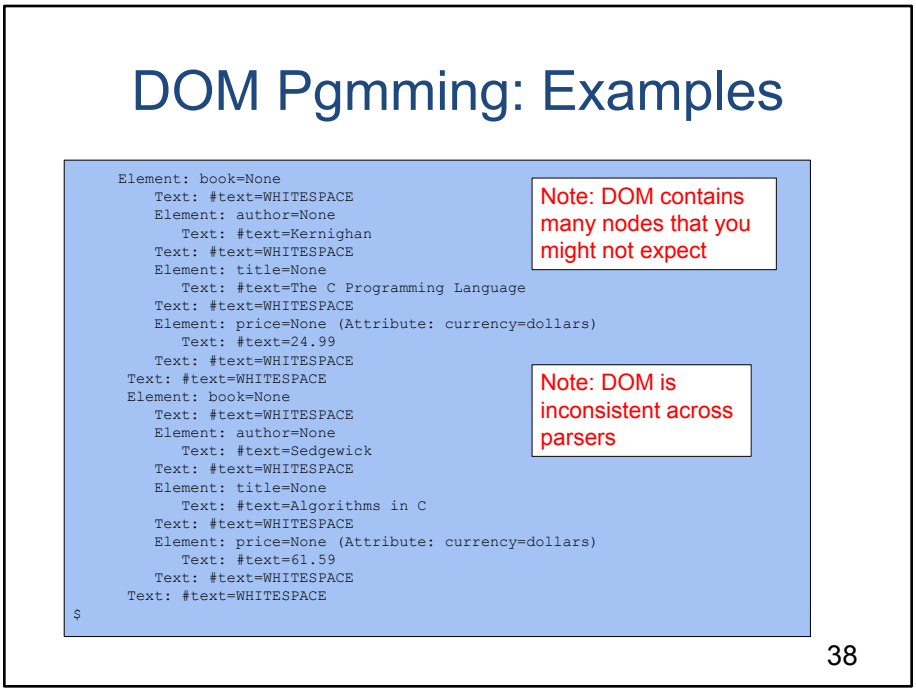
DOM Pgmming: Examples

- See **writedom.js**
  - Recursive pre-order traversal
  - Run on books.xml…

DOM Pgmming: Examples

```
$ node writedom.js books.xml
Document: #document=None
   Processing Instruction: undefined=version="1.0"
   Text: #text=WHITESPACE
   Comment: #comment=
============================================================
   Text: #text=WHITESPACE
   Comment: #comment= books.xml
   Text: #text=WHITESPACE
   Comment: #comment= Author: Bob Dondero
   Text: #text=WHITESPACE
   Comment: #comment=
============================================================
   Text: #text=WHITESPACE
   Element: books=None
      Text: #text=WHITESPACE
      Element: book=None
         Text: #text=WHITESPACE
         Element: author=None
            Text: #text=Kernighan
         Text: #text=WHITESPACE
         Element: title=None
            Text: #text=The Practice of Programming
         Text: #text=WHITESPACE
         Element: price=None (Attribute: currency=dollars)
            Text: #text=40.74
         Text: #text=WHITESPACE
      Text: #text=WHITESPACE
```

Continued on the next page



DOM Pgmming: Examples

DOM is inconsistent across parsers:

Beginning of Python DOM:

```
Document: #document=None
  Comment: #comment=
============================================================
===
  Comment: #comment= books.xml
  Comment: #comment= Author: Bob Dondero
  Comment: #comment=
============================================================
===
  Element: books=None
```

Beginning of JavaScript DOM:

```
Document: #document=None
  Processing Instruction: undefined=version="1.0"
  Text: #text=WHITESPACE
  Comment: #comment=
============================================================
```

```
===
   Text: #text=WHITESPACE
   Comment: #comment= books.xml
   Text: #text=WHITESPACE
   Comment: #comment= Author: Bob Dondero
   Text: #text=WHITESPACE
   Comment: #comment=
============================================================
===
   Text: #text=WHITESPACE
   Element: books=None
```

DOM Pgmming: Examples

- **writebooksdom** programs
  - The job:
    - Write all books in books.xml

DOM Pgmming: Examples

- See **writebooksdom.py**
  - Note: Awkwardness of traversing DOM

```
$ python writebooksdom.py
Author: Kernighan
Title: The Practice of Programming
Price: 40.74 dollars

Author: Kernighan
Title: The C Programming Language
Price: 24.99 dollars

Author: Sedgewick
Title: Algorithms in C
Price: 61.59 dollars

$
```

DOM Pgmming: Examples

- See **writebooksdom.js**
  - Note: Awkwardness of traversing DOM

```
$ node writebooksdom.js
Author: Kernighan
Title: The Practice of Programming
Price: 40.74 dollars

Author: Kernighan
Title: The C Programming Language
Price: 24.99 dollars

Author: Sedgewick
Title: Algorithms in C
Price: 61.59 dollars

$
```

DOM Pgmming: Examples

- **writebooksdomshort** programs
  - The job:
    - Same as writebooksdom programs

DOM Pgmming: Examples

- See **writebooksdomshort.py**
  - Note: Use of `getElementsByTagName()` as shortcut

```
$ python writebooksdomshort.py
Author: Kernighan
Title: The Practice of Programming
Price: 40.74 dollars

Author: Kernighan
Title: The C Programming Language
Price: 24.99 dollars

Author: Sedgewick
Title: Algorithms in C
Price: 61.59 dollars

$
```

DOM Pgmming: Examples

- See **writebooksdomshort.js**
  - Notes:
    - Use of `getElementsByTagName()` as shortcut
    - Returned value is not an array!

```
$ node writebooksdomshort.js
Author: Kernighan
Title: The Practice of Programming
Price: 40.74 dollars

Author: Kernighan
Title: The C Programming Language
Price: 24.99 dollars

Author: Sedgewick
Title: Algorithms in C
Price: 61.59 dollars

$
```

DOM Pgmming: Examples

- **roundtripxml** programs
  - The job:
    - **Parse**: XML doc → DOM tree
      - Common
    - **Generate**: DOM tree → XML doc
      - Less common

DOM Pgmming: Examples

- See **roundtripxml.py**
  - Note:
    - In Python, converting DOM tree to XML doc is easy:
      - `xmlStr = xmlDoc.toxml()`

DOM Pgmming: Examples

```
$ python roundtripxml.py
<?xml version="1.0" ?><!--
================================================================ --><!--
books.xml                                                        --><!--
Author: Bob Dondero                                              --><!--
================================================================
--><books>
   <book>
      <author>Kernighan</author>
      <title>The Practice of Programming</title>
      <price currency="dollars">40.74</price>
   </book>
   <book>
      <author>Kernighan</author>
      <title>The C Programming Language</title>
      <price currency="dollars">24.99</price>
   </book>
   <book>
      <author>Sedgewick</author>
      <title>Algorithms in C</title>
      <price currency="dollars">61.59</price>
   </book>
</books>
```

DOM Pgmming: Examples

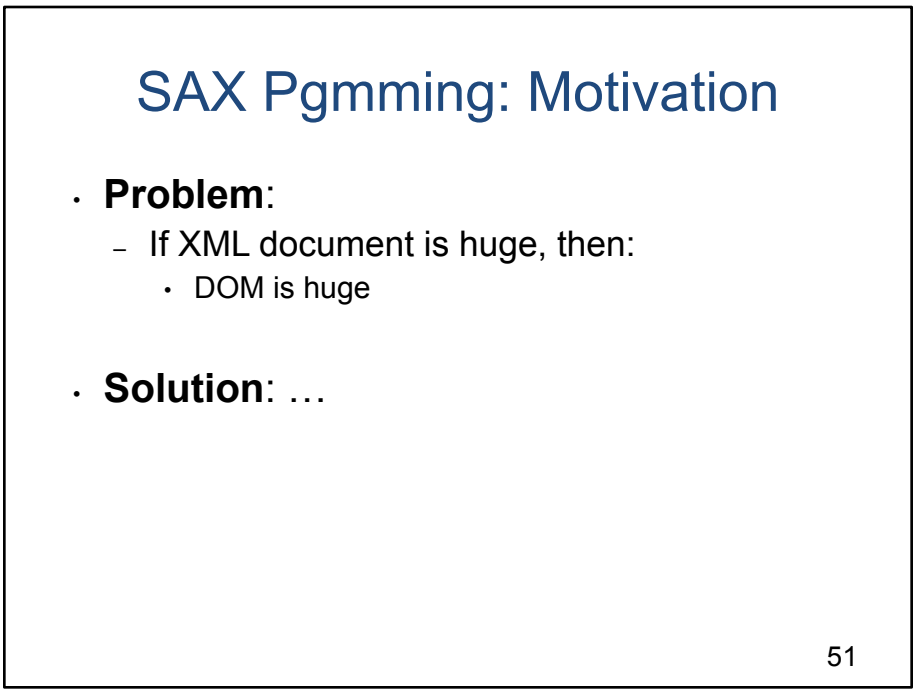
- See **roundtripxml.js**
  - Note:
    - In JavaScript, converting DOM tree to XML doc is easy
      - `xmlStr = xmlserializer.serializeToString(xmlDoc)`

DOM Pgmming: Examples

```
Doesn't
handle
processing
instructions!
```

Agenda

- XML
- XML programming: DOM
- **XML programming: SAX**



SAX Pgmming: Motivation

Problem:

If XML document is huge, then:

DOM is huge

DOM programming creates entire DOM and then use only a small part of it

Space inefficient

DOM might not fit in memory

Solution: …

SAX Pgmming

- A SAX parser:
  - Traverses given XML document
  - Calls methods (which you define) as it encounters each start tag, end tag, text, etc.

SAX Pgmming: Example

- **writeauthorssax** programs
  - The job:
    - Write all authors in books.xml

SAX Pgmming: Example

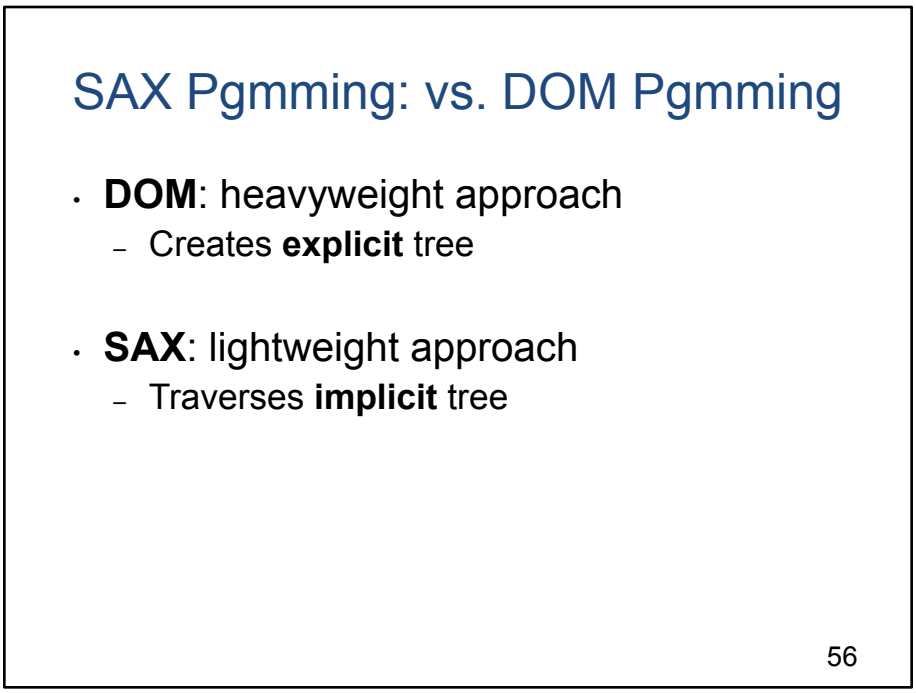
- See **writeauthorssax.py**
  - Traverses implicit DOM
  - Much more space efficient

```
$ python writeauthorssax.py
Kernighan
Kernighan
Sedgewick
$
```

SAX Pgmming: Example

- See **writeauthorssax.js**
  - Traverses implicit DOM
  - Much more space efficient

```
$ node writeauthorssax.js
Kernighan
Kernighan
Sedgewick
$
```



SAX Pgmming vs. DOM Pgmming

DOM: heavyweight approach

Creates **explicit** tree

Use when you care about the entire document structure

SAX: lightweight approach

Traverses **implicit** tree

Use when:

You don't care about the entire document structure

The XML document is huge (and DOM tree doesn't fit into memory)

Summary

- We have covered:
  - XML
  - XML programming: DOM
  - XML programming: SAX
- See also:
  - **Appendix**: XML Checking

Appendix:
XML Checking

XML Checking

- To run the example Python programs in this appendix:
  - `python –m pip install lxml`

XML Checking: Well-Formedness

- Computer science jargon…
  - An XML doc is ***well-formed*** iff it conforms to the XML specification

XML Checking: Well-Formedness

- See books.xml (revisited)
- See **booksmalformed.xml**
- See **checkxml.py**

XML Checking: Well-Formedness

```
$ python checkxml.py books.xml
The document is well-formed.
$ python checkxml.py booksmalformed.xml
mismatched tag: line 22, column 39
```

XML Checking: Validity

- ***DTD (Document Type Definition)***
  - An ISO standard
  - A DTD defines whether an XML doc is **valid**
- ***XSD (XML Schema Definition)***
  - A W3C standard
  - A XSD defines whether an XML doc is **valid**
- Others: ***RELAX NG***, ***Schematron***, ***DSDL***, …
  - See Wikipedia “XML” page
- Can use to define a comm protocol

XML Checking: Validity

- Computer science jargon…
  - A DTD or XSD defines a ***grammar***
  - The grammar defines a ***language***
    - A language is a set of documents
  - An XML document is ***valid*** with respect to a DTD or XSD iff it is an element of the language defined by that DTD or XSD

XML Checking: Validity

- See **books.dtd**
- See **booksusingdtd.xml**
- See **booksusingdtdinvalid.xml**
- See **checkxmlusingdtd.py**

XML Checking: Validity

```
$ python checkxmlusingdtd.py booksusingdtd.xml
The document is well-formed and valid.
$ python checkxmlusingdtd.py booksusingdtdinvalid.xml
Element book content does not follow the DTD,
expecting (author , title , price), got (author price
title ), line 25, column 11 (<string>, line 25)
$
```

XML Checking: Validity

- See **books.xsd**
- See **booksusingxsd.xml**
- See **booksusingxsdinvalid.xml**
- See **checkxmlusingxsd.py**

XML Checking: Validity

```
$ python checkxmlusingxsd.py booksusingxsd.xml
books.xsd
The document is well-formed and valid.
$ python checkxmlusingxsd.py booksusingxsdinvalid.xml
books.xsd
Element 'price': This element is not expected.
Expected is ( title ). (<string>, line 0)
$
```

XML Checking: Validity

- Advantages of validation via **DTDs**:
  - DTDs can be defined inline
  - DTDs are compact and readable
  - DTDs are widely supported

XML Checking: Validity

- Advantages of validation via **XSDs**:
  - XSDs support strong typing
    - Can specify that an element contains an integral number, real number, date, ...
  - XSDs provide natural way to map XML doc to typed objects
  - XSDs are written in XML; so can create/parse using XML tools
    - Can define a XSD, which defines a XSD, which ...