XML & JSON

Introduction

- XML: Extensible Markup Language
- Defined by the WWW Consortium (W3C)
- Documents have tags giving extra information about sections of the document
  - E.g. <title> XML </title> <slide> Introduction …</slide>
- **Extensible**, unlike HTML
  - Users can add new tags, and *separately* specify how the tag should be handled for display

Example of Nested Elements

```
<?xml version = "1.0"?>
<bank-1>
    <customer>
        <customer_name> Hayes </customer_name>
        <customer_street> Main </customer_street>
        <customer_city>    Harrison </customer_city>
        <account>
            <account_number> A-102 </account_number>
            <branch_name>     Perryridge </branch_name>
            <balance>            400 </balance>
        </account>
        <account>
            ...
        </account>
    </customer>
    .
    .
</bank-1>
```

XML: Motivation

- Data interchange is critical in today's networked world
  - Examples:
    - Banking: funds transfer
    - Order processing (especially inter-company orders)
    - Scientific data
      - Chemistry, Genetics
  - Paper flow of information between organizations is being replaced by electronic flow of information
- Each application area has its own set of standards for representing information
- XML has become the basis for all new generation data interchange formats
  - For awhile, XML (extensible markup language) was the only choice for open data interchange. But over the years there has been a lot of transformation in the world of open data sharing. The more lightweight JSON (Javascript object notation) has become a popular alternative to XML for various reasons.

XML Motivation (Cont.)

- Earlier generation formats were based on plain text with line headers indicating the meaning of fields
  - Similar in concept to email headers
  - Does not allow for nested structures, no standard "type" language
  - Tied too closely to low level document structure (lines, spaces, etc)
- Each XML based standard defines what are valid elements, using
  - XML type specification languages to specify the syntax
    - DTD (Document Type Descriptors)
    - XML Schema
  - Plus textual descriptions of the semantics
- XML allows new tags to be defined as required
  - However, this may be constrained by DTDs
- A wide variety of tools is available for parsing, browsing and querying XML documents/data

Comparison with Structured (Relational) Data

- Inefficient: tags, which in effect represent schema information, are repeated
- Better than relational tuples as a data-exchange format
  - Unlike relational tuples, XML data is self-documenting due to presence of tags
  - Non-rigid format: tags can be added
  - Allows nested structures
  - Wide acceptance, not only in database systems, but also in browsers, tools, and applications

Structure of XML Data

- **Tag**: label for a section of data
- **Element**: section of data beginning with <*tagname*> and ending with matching </*tagname*>
- Elements must be properly nested
  - Proper nesting
    - <account> … <balance> …. </balance> </account>
  - Improper nesting
    - <account> … <balance> …. </account> </balance>
  - Formally: every start tag must have a unique matching end tag, that is in the context of the same parent element.
- Every document must have a single top-level element

Example of Nested Elements

```
<?xml version = "1.0"?>
<bank-1>
    <customer>
        <customer_name> Hayes </customer_name>
        <customer_street> Main </customer_street>
        <customer_city>    Harrison </customer_city>
        <account>
            <account_number> A-102 </account_number>
            <branch_name>     Perryridge </branch_name>
            <balance>            400 </balance>
        </account>
        <account>
            …
        </account>
    </customer>
    .
    .
</bank-1>
```

Structure of XML Data (Cont.)

- Mixture of text with sub-elements is legal in XML.
  - Example:

```
<account>
   This account is seldom used any more.
    <account_number> A-102</account_number>
    <branch_name> Perryridge</branch_name>
    <balance>400 </balance>
</account>
```

  - Useful for document markup, but discouraged for data representation

Attributes

- Elements can have **attributes**

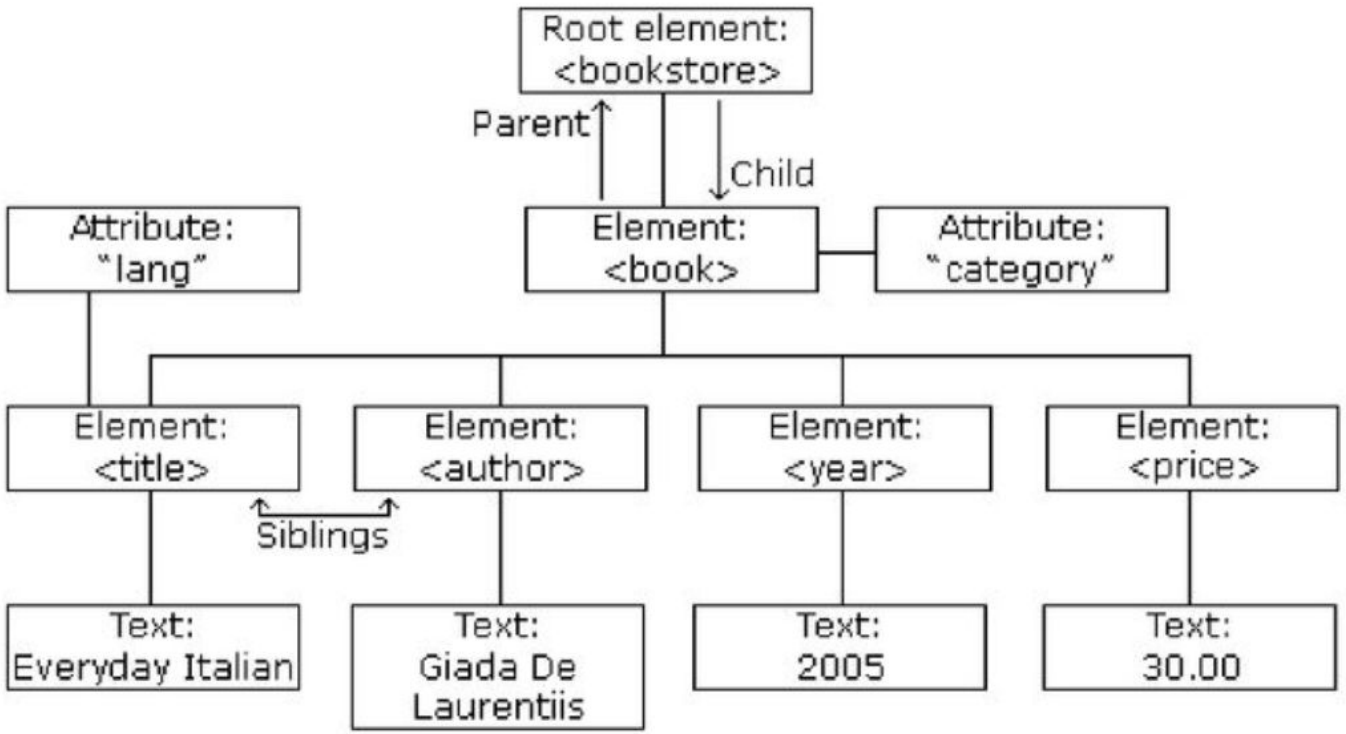
```
<account acct-type = "checking" >
    <account_number> A-102 </account_number>
    <branch_name> Perryridge </branch_name>
    <balance> 400 </balance>
</account>
```

- Attributes are specified by *name=value* pairs inside the starting tag of an element
- An element may have several attributes, but each attribute name can only occur once

```
<account  acct-type = "checking"  monthly-fee="5">
```

Class Activity 9

- Convert the following Tree structure to bookstore.xml



Attributes vs. Subelements

- Distinction between subelement and attribute
  - In the context of documents, attributes are part of markup, while subelement contents are part of the basic document contents
  - In the context of data representation, the difference is unclear and may be confusing
    - Same information can be represented in two ways

```
<account  account_number = "A-101">  …. </account>


<account>
   <account_number>A-101</account_number> …
 </account>
```

  - Suggestion: use attributes for identifiers of elements, and use subelements for contents

More on XML Syntax

- Elements without subelements or text content can be abbreviated by ending the start tag with a /> and deleting the end tag
  - <account number="A-101" branch="Perryridge" balance="200 />
- To store string data that may contain tags, without the tags being interpreted as subelements, use CDATA as below
  - <![CDATA[<account> … </account>]]>

  Here, <account> and </account> are treated as just strings

  CDATA stands for "character data", **text that will NOT be parsed by a parser**

XML Document Schema

- Database schemas constrain what information can be stored, and the data types of stored values
- XML documents are not required to have an associated schema
- However, schemas are very important for XML data exchange
  - Otherwise, a site cannot automatically interpret data received from another site
- Two mechanisms for specifying XML schema
  - **Document Type Definition (DTD)**
    - Widely used
  - **XML Schema**
    - Newer, increasing use

Why DTDs?

- XML documents are designed to be processed by computer programs
  - If you can put just *any* tags in an XML document, it's very hard to write a program that knows how to process the tags
  - A DTD specifies what tags may occur, when they may occur, and what attributes they may (or must) have
- A DTD allows the XML document to be verified (shown to be legal)
- A DTD that is shared across groups allows the groups to produce consistent XML documents

DTD example: XML

```
<?xml version="1.0"?>
<!DOCTYPE weatherReport SYSTEM
"http://www.mysite.com/mydoc.dtd">
<weatherReport>
    <date>05/29/2002</date>
    <location>
        <city>Philadelphia</city>,
        <state>PA</state>
        <country>USA</country>
    </location>
    <temperature-range>
        <high scale="F">84</high>
        <low scale="F">51</low>
    </temperature-range>
</weatherReport>
```

mydoc.dtd

```
<!ELEMENT weatherReport (date, location,
temperature-range)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT location (city, state, country)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT state (#PCDATA)>
<!ELEMENT country (#PCDATA)>
<!ELEMENT temperature-range
        ((low, high)|(high, low))>
<!ELEMENT low (#PCDATA)>
<!ELEMENT high (#PCDATA)>
<!ATTLIST low scale (C|F) #REQUIRED>
<!ATTLIST high scale (C|F) #REQUIRED>
```

XML Parsing

https://www.cs.odu.edu/~sampath/courses/f18/cs795/files/data/country_data.xml

```
import xml.etree.ElementTree as et
tree = et.parse('country_data.xml')
root = tree.getroot()
#root has a tag and a dictionary of attributes:
print(root.tag)
#print(root.attrib)
#Children are nested, and we can access specific child nodes by index:
print(root[0][1].text)
#It also has children nodes over which we can iterate:
for child in root:
    print(child.tag, child.attrib)
# For more information: https://docs.python.org/2/library/xml.etree.elementtree.html
```

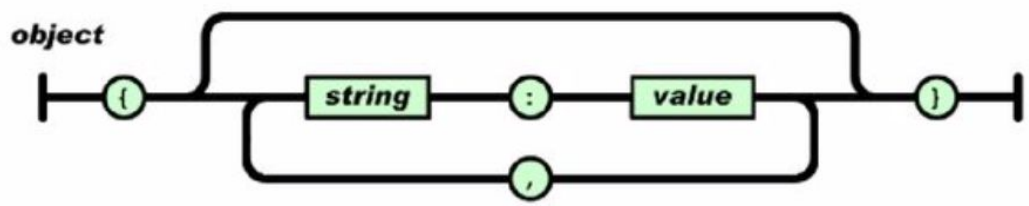
JSON as an XML Alternative

- JSON = JavaScript Object Notation
  - It's really language independent
  - most programming languages can easily read it and instantiate objects or some other data structure
- JSON is a light-weight alternative to XML for data-interchange
- Started gaining tracking ~2006 and now widely used
- http://json.org/ has more information

JSON Data – A name and a value

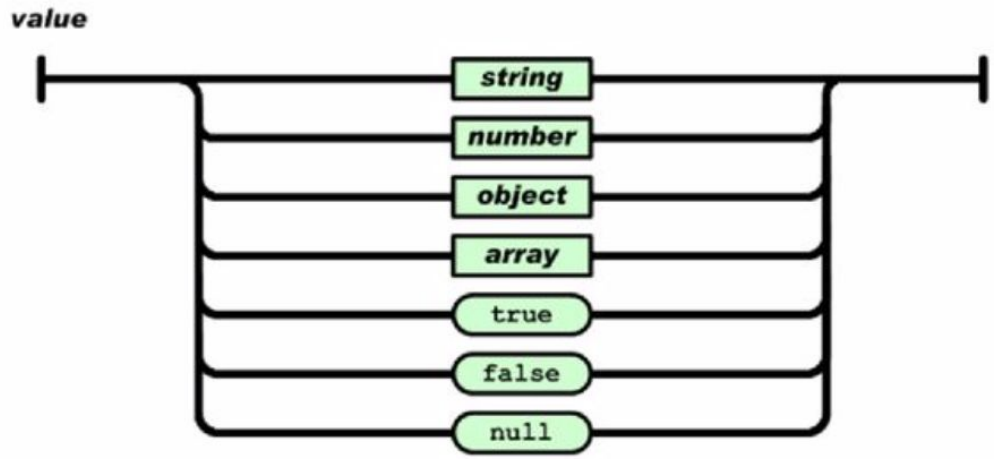
- A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value
- Unordered sets of name/value pairs
- Begins with { (left brace)
- Ends with } (right brace)
- Each name is followed by : (colon)
- Name/value pairs are separated by , (comma)

```
{
"employee_id": 1234567,
"name": "Jeff Fox",
"hire_date": "1/1/2013",
"location": "Norwalk, CT",
"consultant": false
}
```



JSON Data – A name and a value

- In JSON, *values* must be one of the following data types:
- a string
- a number
- an object (JSON object)
- an array
- a boolean
- null




```
{
"employee_id": 1234567,
"name": "Jeff Fox",
"hire_date": "1/1/2013",
"location": "Norwalk, CT",
"consultant": false
}
```

JSON Data – A name and a value

- Strings in JSON must be written in double quotes.

```
{ "name":"John" }
```

- Numbers in JSON must be an integer or a floating point.

```
{ "age":30 }
```

- Values in JSON can be objects.

```
{
"employee":{ "name":"John", "age":30, "city":"New York" }
}
```

- Values in JSON can be arrays.

```
{
"employees":[ "John", "Anna", "Peter" ]
}
```

JSON Parsing

```
import json

json_string = '{"first_name": "Guido", "last_name":"Rossum",
"phone":[9098693256, 9097846521]}'

parsed_json = json.loads(json_string)

data = DataFrame(parsed_json)

print(parsed_json['first_name'])

phone = list(parsed_json['phone'])

print(phone)

print(data)
```

Note: For external file read, use json.load and data pretty print to display the content of json file.

```
from pprint import pprint
data = json.load(open('data.json'))
pprint(data)
```

Class Activity 10

- Convert the following bookstore.xml to bookstore.json

```
<?xml version="1.0"?>
<bookstore>
    <book category="sci-fi">
            <title lang="en"> 2001</title>
            <author>Arthur C. Clarke</author>
            <price>$30.0</price>
            <year>1968</year>
    </book>
    <book>
            <title lang="rs">Story about a True Man</title>
            <author>Boris Polevoy</author>
            <price>$20.00</price>
            <year>1952</year>
     </book>
</bookstore>
```

XML vs JSON

- JSON is Like XML Because
  - Both JSON and XML are "self describing" (human readable)
  - Both JSON and XML are hierarchical (values within values)
  - Both JSON and XML can be parsed and used by lots of programming languages

- JSON is Unlike XML Because
  - JSON doesn't use end tag
  - JSON is shorter
  - JSON is quicker to read and write
  - JSON can use arrays
  - JSON has a better fit for OO systems than XML

- The biggest difference is:
  - XML has to be parsed with an XML parser. JSON can be parsed by a standard JavaScript function.

Why JSON?

Steps involved in exchanging data from web server to browser involves:

Using XML

1. Fetch an XML document from web server.
2. Use the XML DOM to loop through the document.
3. Extract values and store in variables.
4. It also involves type conversions.

Using JSON

1. Fetch a JSON string.
2. Parse the JSON using JavaScript functions.