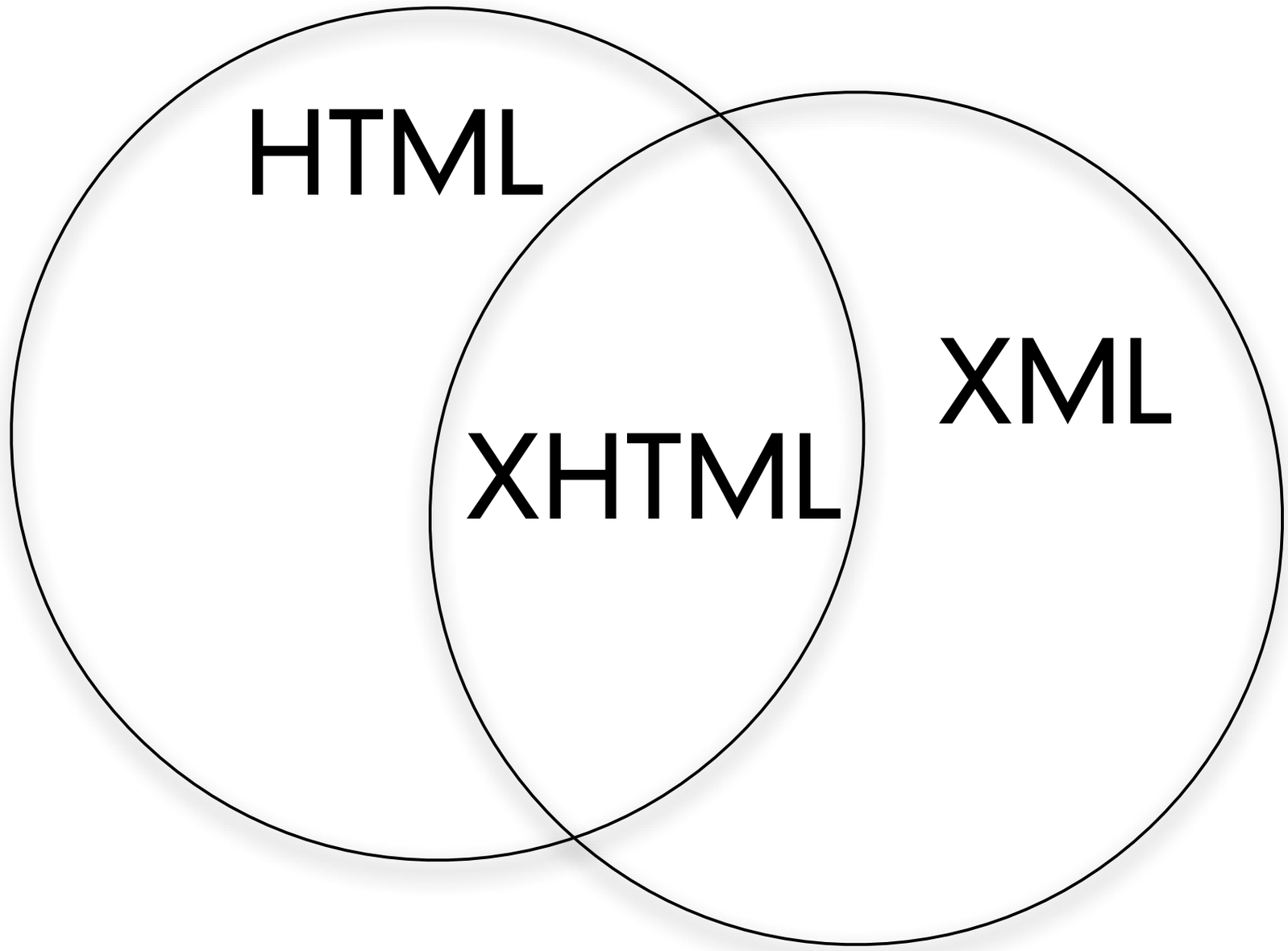
User Interaction: XML and JSON

Assoc. Professor Donald J. Patterson
INF 133 Fall 2013

HTML and XML



- HTML, XML and JSON
  - Structured Data Formats that evolved with the web
  - Text with a syntax applied
  - They can represent a huge variety of information
  - They enable data transport
    - Different systems and technologies and programming languages depend on the syntax being standardized

```
<?xml version="1.0"?>
<note>
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
</note>
```

- What is XML?
  - XML stands for “eXtensible Markup Language”
  - XML was designed in the context of separating
    - data from display
  - XML tags are not predefined
    - You define your own tags
  - XML is designed to be self-descriptive

- The Difference Between XML and HTML
- XML
  - designed to transport and store data
  - It looks like HTML
  - The focus is on what the data is
- HTML
  - originally focussed on how data looks
  - it typically is “broken-XML”
  - XHTML is
    - HTML that conforms to XML standard

- XML Does not DO Anything
  - It is a data format
  - A program must be written to manipulate the data
    - To search the data
    - To display the data
    - To change the data
- Even though the data seems to be associated with a task it is still just data.

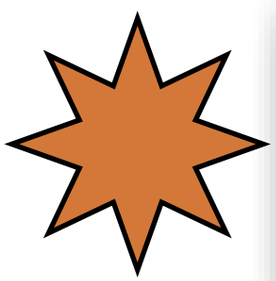
Schema

Tags

Characters

Schema

Tags



Characters

- XML is Just Plain Text
  - There is nothing fancy about the storage
  - A program that can read and write text can read and write XML
- an XML-aware application
  - Expects a valid tag structure
  - Interprets the tags in a particular way

```
<?xml version="1.1" encoding="utf-8" standalone="yes"?>
```

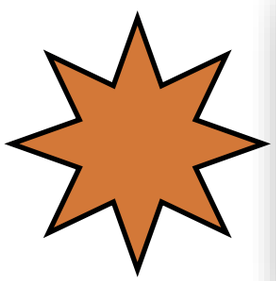
- XML declaration
  - version
    - 1.0
      - declaration is optional, defaults assumed
    - 1.1
      - declaration is mandatory
      - some encoding ambiguities resolved between Unicode versions
  - encoding
    - how are UNICODE characters represented
  - standalone
    - can this document be DTD validated without retrieving external documents?

Schema

Tags

Characters

Schema



Tags

Characters

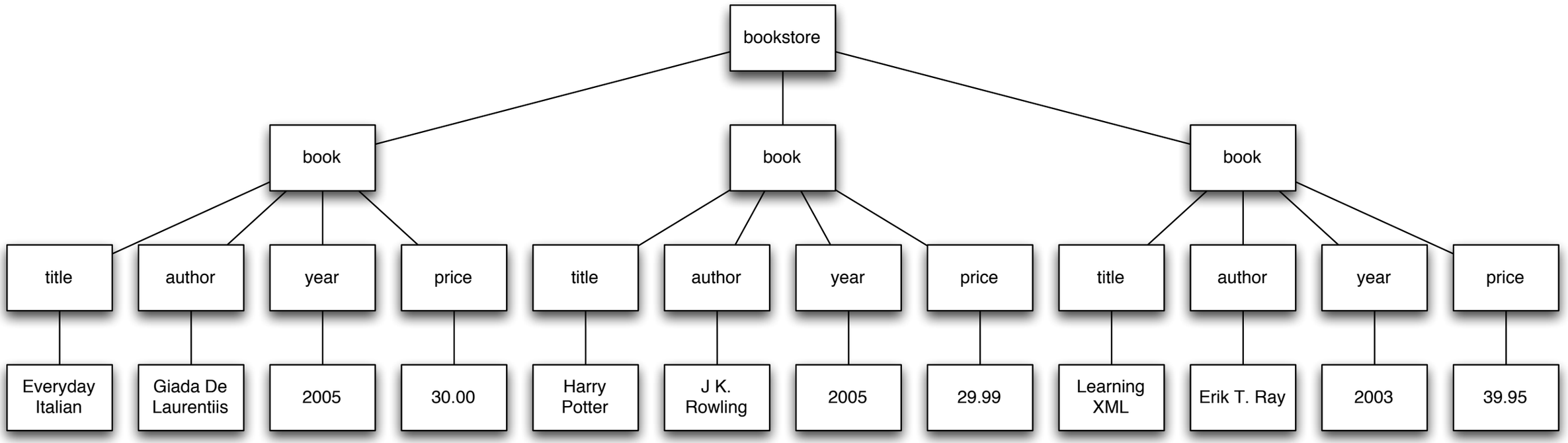
- With XML You Invent Your Own Tags
  - <from> and <to>
    - are not defined anywhere official
    - they are invented by the author
  - There are no predefined tags
- In contrast, HTML has predefined tags
  - <p> <href> etc.,
- In XML the author defines the tags and the structure
  - within the bounds of a “valid XML document”

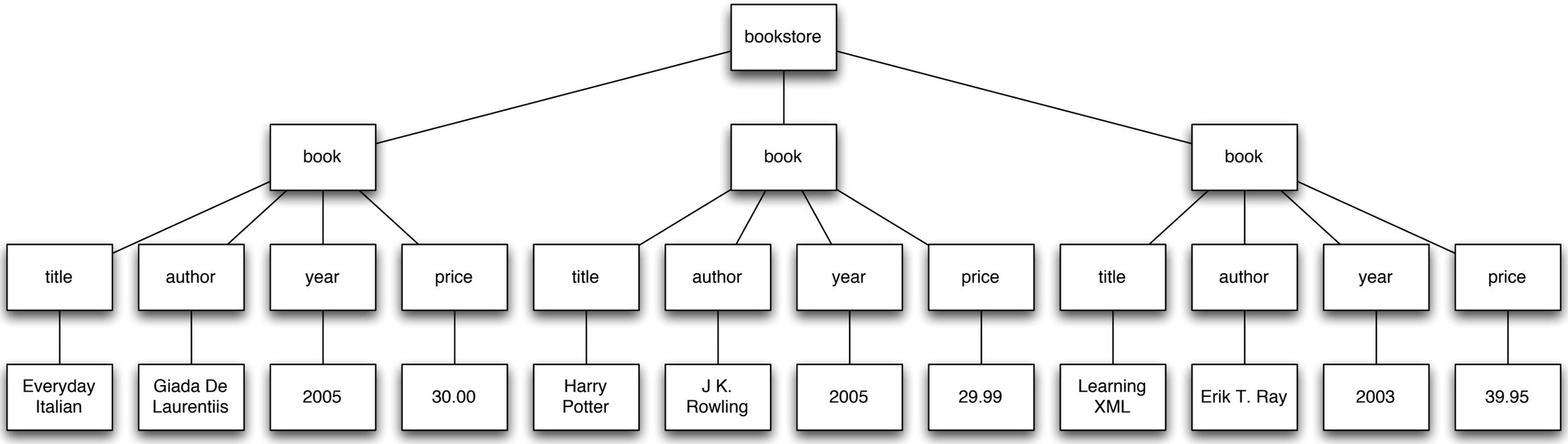
- XML is Not a Replacement for HTML
  - XML complements HTML
  - XHTML is an XML syntax compliant version of HTML
    - It has tags defined by a standards body

- XML Separates Data from HTML presentation
- XML Simplifies Data Sharing
- XML Simplifies Data Transport
- XML Simplifies Platform Changes
- XML Makes Your Data More Available

- XML is Used to Create New Internet Languages
  - XHTML the latest version of HTML
  - WSDL for describing available web services
  - WAP and WML as markup languages for handheld devices
  - RSS languages for news feeds
  - RDF and OWL for describing resources and ontology
  - SMIL for describing multimedia for the web

- XML uses a tree structure
  - with a root element
  - and child elements
- tags indicate the start and end of an element
- opening tag looks like this:
  - <tag>
- a closing tag looks like this:
  - </tag>
- A valid XML document has exactly one closing tag for every opening tag


bookstore
book
book
book
title
author
year
price
title
author
year
price
title
author
year
price
Everyday Italian
Giada De Laurentiis
2005
30.00
Harry Potter
J K. Rowling
2005
29.99
Learning XML
Erik T. Ray
2003
39.95

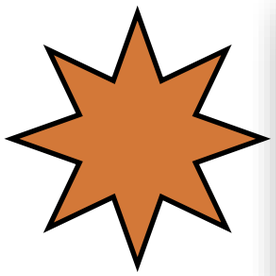



```xml
<bookstore>
    <book category="COOKING">
        <title lang="en">Everyday Italian</title>
        <author>Giada De Laurentiis</author>
        <year>2005</year>
        <price>30.00</price>
    </book>
    <book category="CHILDREN">
        <title lang="en">Harry Potter</title>
        <author>J K. Rowling</author>
        <year>2005</year>
        <price>29.99</price>
    </book>
    <book category="WEB">
        <title lang="en">Learning XML</title>
        <author>Erik T. Ray</author>
        <year>2003</year>
        <price>39.95</price>
    </book>
</bookstore>
```

Schema

Tags

Characters



Schema

Tags

Characters

```
<!DOCTYPE bookstore [

<!ELEMENT bookstore (book+)>
<!ELEMENT book (title,author,year,(price)+)>
<!ELEMENT title (CDATA)>
<!ELEMENT author (CDATA)>
<!ELEMENT year (CDATA)>
<!ELEMENT price (CDATA)>

<!ATTLIST book category CDATA #REQUIRED>
<!ATTLIST title lang CDATA #IMPLIED>

]>
```

Details

- Details
  - All XML Elements Must Have a Closing Tag
  - HTML
    - This is a paragraph
    - This is another paragraph
- XML
  - This is a paragraph
  - This is another paragraph

- Details

- Details
  - Empty XML Elements may use a short cut closing tag

- Details
  - Empty XML Elements may use a short cut closing tag
    - <nothing></nothing>

- Details
  - Empty XML Elements may use a short cut closing tag
    - <nothing></nothing>
    - <nothing/>

- Details
  - Empty XML Elements may use a short cut closing tag
    - <nothing></nothing>
    - <nothing/>

- Details
  - Empty XML Elements may use a short cut closing tag
    - <nothing></nothing>
    - <nothing/>

- <img src=”picture.jpg”>

- Details
  - Empty XML Elements may use a short cut closing tag
    - <nothing></nothing>
    - <nothing/>

- <img src=”picture.jpg”>
- <img src=”picture.jpg”></img>

- Details
  - Empty XML Elements may use a short cut closing tag
    - <nothing></nothing>
    - <nothing/>

- <img src=”picture.jpg”>
- <img src=”picture.jpg”></img>
-

- Details
  - Empty XML Elements may use a short cut closing tag
  - ✓ <nothing></nothing>
  - ✓ <nothing/>

- ✗ <img src="picture.jpg">
- ✓ <img src="picture.jpg"></img>
- ✓

- Details
  - XML Tags are Case Sensitive
    - <Message>This is incorrect</message>
    - <message>This is correct</message>
    - <Message>This is correct</Message>

- Details
  - XML Elements Must be Properly Nested
    - HTML might have this
      - This text is bold and italic
- Valid XML requires this:
  - This text is bold and italic

- Details
  - XML Documents Must Have a Root Element
    - This is the top-level tag
      - <root>
      - <child>
      - <subchild>.....</subchild>
      - </child>
      - </root>

- Details
  - XML Nodes may have attributes
  - Which describe the tag
  - XML Attribute Values Must be Quoted
    - Invalid:

```
<note date=12/11/2007>
  <to>Tove</to>
  <from>Jani</from>
</note>
```

- Valid:

```
<note date="12/11/2007">
  <to>Tove</to>
  <from>Jani</from>
</note>
```

- Details
  - Special characters:
    - If you put a "<" in your data it will mess up XML parsing
      - <message>if salary < 1000 then</message>
    - 5 characters are like this
      - & < > " '

- `&` → & (ampersand, U+0026)
- `<` → < (less-than sign, U+003C)
- `>` → > (greater-than sign, U+003E)
- `&quot;` → " (quotation mark, U+0022)
- `&apos;` → ' (apostrophe, U+0027)

- <message>if salary < 1000 then</message>

http://en.wikipedia.org/wiki/Character_encodings_in_HTML

- Details
  - Comments in XML
    - <!-- This is a comment -->
  - White-space is preserved
    - <message>There is a lot of space</message>

- Attributes and Elements are pretty interchangeable

```
<person sex="female">
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>
```

```
<person>
  <sex>female</sex>
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>
```

```
<note date="10/01/2008">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>

<note>
  <date>10/01/2008</date>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>

<note>
  <date>
    <day>10</day>
    <month>01</month>
    <year>2008</year>
  </date>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

- On beyond XML
  - XML validation
  - Schemas like XML - DTD
  - Namespaces
  - XSLT
    - transforms XML to HTML for viewing

- Demo:
  - Look at Chrome debugging tools to see the "Document Object Model"


JSON

JSON

- JSON
  - also structured text
  - also with a syntax applied
  - it can also represent a huge variety of information
  - It also enables data transport
    - Across systems, languages, and networks
- So what does JSON look like?

JSON


```
{
   "place":[
      {
         "suggestion":"at home",
         "meta":{
            "id":"null",
            "index":0
         },
         "size":"20.0"
      }
   ],
   "activity":[
      {
         "suggestion":"working",
         "meta":{
            "id":"null",
            "index":2
         },
         "size":"10.558333333333334"
      },
      {
         "suggestion":"sleeping",
         "meta":{
            "id":"null",
            "index":3
         },
         "size":"10.0"
      }
   ],
   "other":[
      {
         "suggestion":"(do not disturb)",
         "meta":{
            "id":"null",
            "index":1
         },
         "size":"10.0"
      }
   ],
   "error":[
      "false"
   ]
}
```

JSON

- What is JSON?
  - JSON stands for “JavaScript Object Notation”
  - JSON was designed to pass data around between browsers and servers
  - JSON has no tags, only data
  - JSON has no meta-data

JSON

- JSON also does not DO Anything
  - It is a data format
  - A program must be written to manipulate the data
    - To search the data
    - To display the data
    - To change the data

JSON

- JSON was developed by people who thought that the meta-data in XML was
  - unnecessary
  - too big
  - too hard to maintain
  - not that valuable
- It also happens to be the native data storage format in Javascript / browsers

JSON

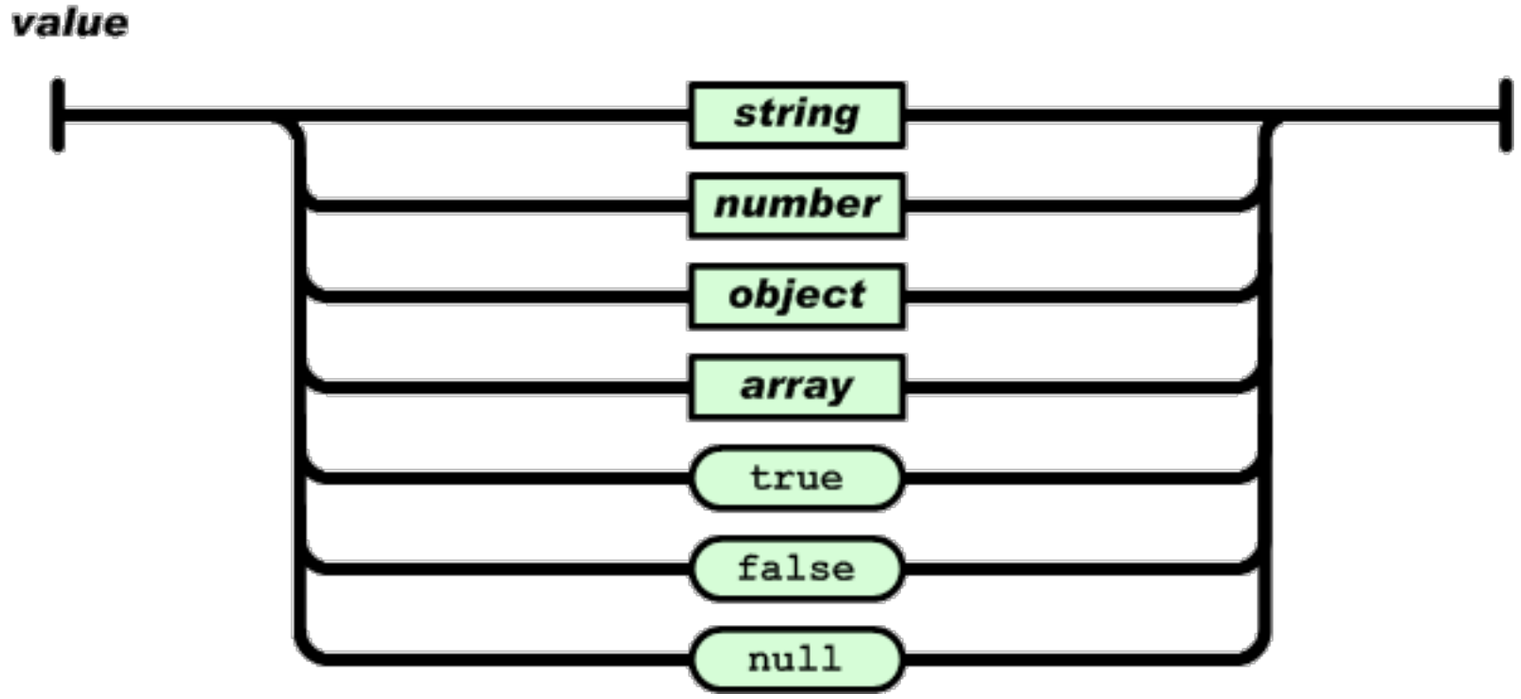
http://json.org/

- Details
  - Two basic structures
    - object:
      - name/value pairs
      - think Map
    - array
      - list of values
      - think List

JSON

http://json.org/

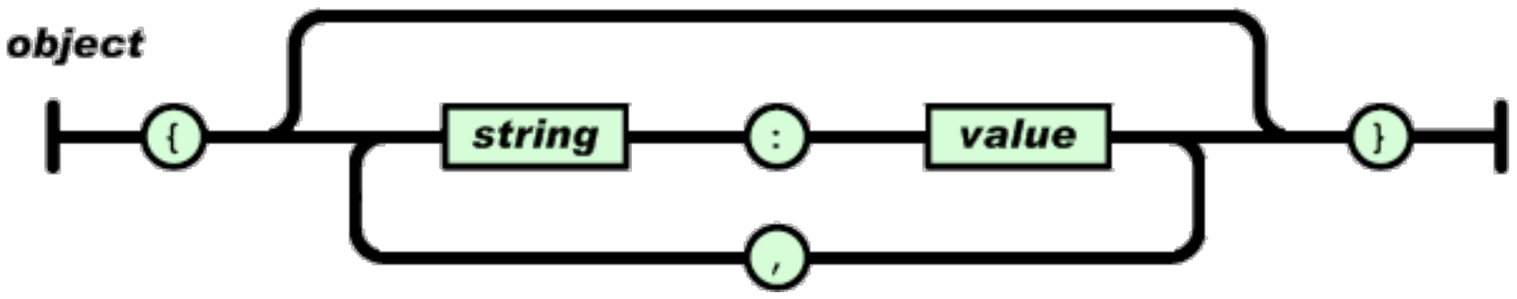
- Details
  - The basic type is a value which can be
    - a string
    - a number
    - an object
    - an array
    - “true”
    - “false”
    - “null”



JSON

http://json.org/

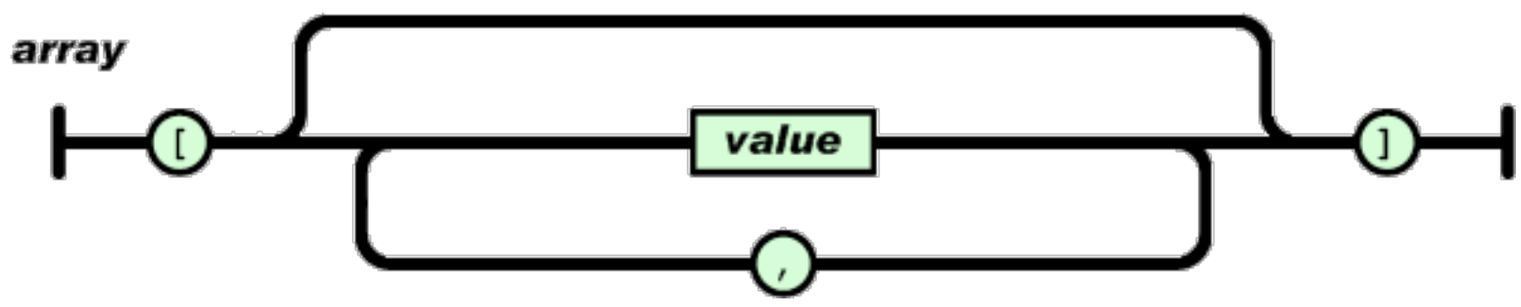
- Details
  - Object
    - Delimited by curly braces
    - name/values are separated by colons
    - elements are separated by commas
      - names are always strings
      - values are always values



JSON

http://json.org/

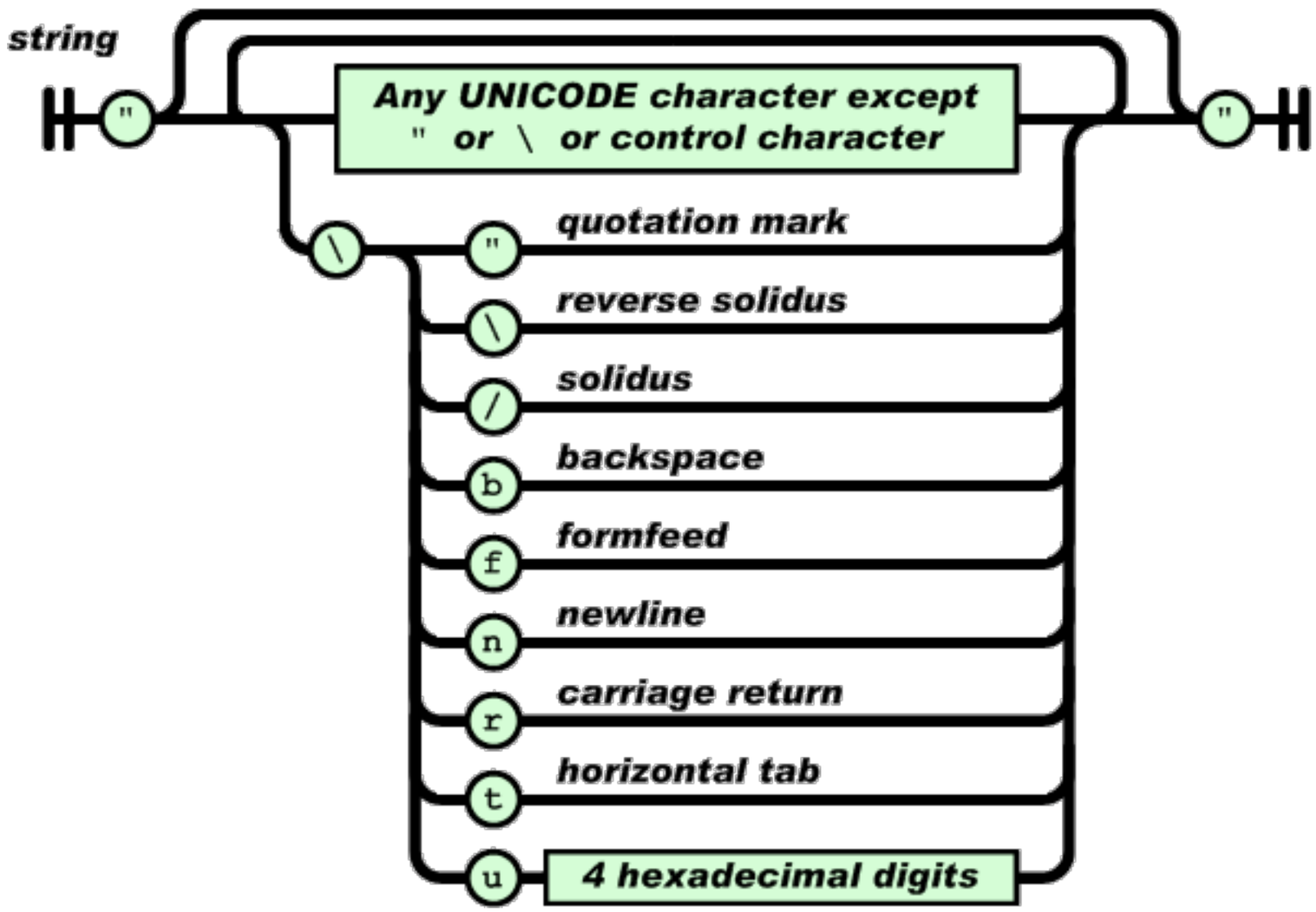
- Details
  - Array
    - Delimited by square braces
    - elements are separated by commas
      - elements are always values



JSON

http://json.org/

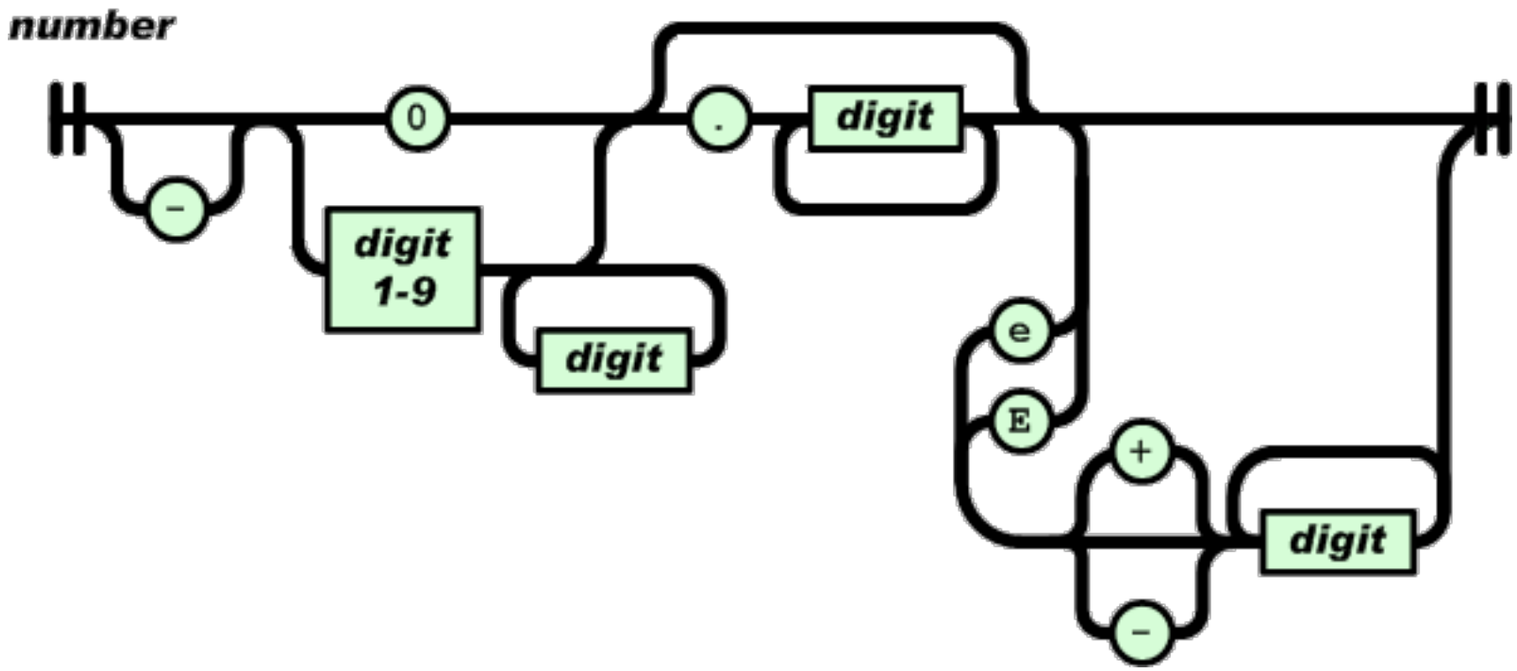
- Details
  - String
    - is UNICODE, recommended is “utf-8”
    - is always in double quotes
    - uses \ escape sequences



JSON

http://json.org/

- Details
  - Number



JSON

http://json.org/

- Details
  - White space outside of quotes is ignored

JSON


```
{
   "place":[
      {
         "suggestion":"at home",
         "meta":{
            "id":"null",
            "index":0
         },
         "size":"20.0"
      }
   ],
   "activity":[
      {
         "suggestion":"working",
         "meta":{
            "id":"null",
            "index":2
         },
         "size":"10.558333333333334"
      },
      {
         "suggestion":"sleeping",
         "meta":{
            "id":"null",
            "index":3
         },
         "size":"10.0"
      }
   ],
   "other":[
      {
         "suggestion":"(do not disturb)",
         "meta":{
            "id":"null",
            "index":1
         },
         "size":"10.0"
      }
   ],
   "error":[
      "false"
   ]
}
```

JSON

http://json.org/

- Supported languages
  - ASP, ActionScript, C, C++,C#,ColdFusion, D, Delphi, E, Eiffel, Erlang, Fan, Flex, Haskell, haXe, Java, JavaScript, Lasso,Lisp,LotusScript, Lua, Objective C, Objective CAML, OpenLaszlo, Perl, PHP, Pike, PL/SQL,PowerShell, Prolog, Pythong, R, Realbasic, Rebol, Ruby, Squeak, Tcl, Visual Basic, Visual FoxPro

JSON

- On beyond JSON
  - JSON validation tools are easy to find
    - For example, jsonlint.com
  - No defined schema language
  - No built-in namespaces (no meta-data!)
  - No built-in transformation languages

XML vs JSON

- XML is like a Ferrari
  - A Ferrari will get you to Las Vegas faster
- JSON is like a good bicycle
  - A bicycle can go off-road

- XML is beautiful and powerful
- XML is well-engineered and well-researched
- JSON is much lighter weight
- JSON is easier to just get going fast

XML vs JSON

- XML is like a Ferrari
  - A Ferrari will get you to Las Vegas faster
- JSON is like a good bicycle
  - A bicycle can go off-road

- XML is beautiful and powerful
- XML is well-engineered and well-researched
- JSON is much lighter weight
- JSON is easier to just get going fast

XML vs JSON

- XML is like a Ferrari
  - A Ferrari will get you to Las Vegas faster
- JSON is like a good bicycle
  - A bicycle can go off-road

- XML is beautiful and powerful
- XML is well-engineered and well-researched
- JSON is much lighter weight
- JSON is easier to just get going fast

JSON vs XML

http://www.secretgeek.net/json_3mins.asp

- JSON is like XML
  - They are both human-readable text
  - They are both hierarchical/ tree-structured
  - Both can be parsed and used in many languages
  - Both can be passed in AJAX requests
    - (despite the X in AJAX)

JSON vs XML

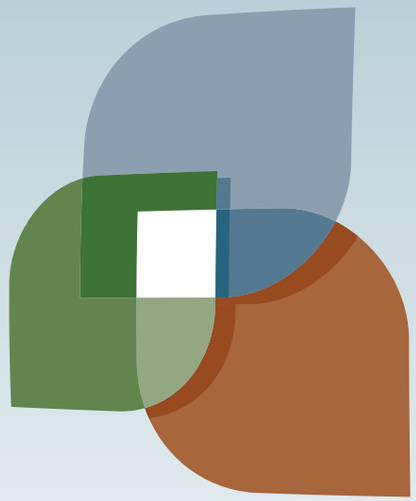
http://www.secretgeek.net/json_3mins.asp

- JSON is different than XML
  - JSON does not have tags
  - JSON is less verbose
    - quicker to write
    - quicker to read
    - quicker to transport
- JSON can be parsed trivially using the eval() procedure in Javascript
- JSON has arrays, XML does not
- XML is extensible JSON usually isn't

JSON vs XML

http://www.secretgeek.net/json_3mins.asp

- Using either looks like:
  - get the JSON/XML string
  - convert it to a data structure
    - JSON -> eval(<string>)
    - XML -> some parse function (lib dependent)
  - Use the data structure
- Do not process either type of data by “hand”.
  - input: Use a library to parse the data
  - output:
    - Create the data in native data structures
    - Use a program or method to output the data structure in JSON/XML



LUCI

Thursday, October 10, 13