Question 1

<!ELEMENT Course_Catalog (Department*)>

Instructions: For each question, you are to write a DTD that validates against the corresponding XML data set. Our back-end will validate the sample data with your DTD and display the result. When you're satisfied with your solution for a given problem, click the "Submit" button to check your answer.

You may perform these exercises as many times as you like, so we strongly encourage you to keep working with them until you complete the exercises with full credit.

Q 1 (1 point possible)

In this question, you are to create a DTD for a small XML data set drawn from the Stanford course catalog. There are multiple departments, each with a department chair, some courses, and professors and/or lecturers who teach courses. The XML data is here.

Write a DTD for the XML data set.

Important: Do not include **<!DOCTYPE Course_Catalog [...]>** in your DTD. Your DTD should start with **<!ELEMENT Course_Catalog (Department*)>**.

```
1 Enter your DTD here
```

```
−<Course_Catalog>
  −<Department Code="CS">
      <Title>Computer Science</Title>
    −<Chair>
      −<Professor>
          <First_Name>Jennifer</First_Name>
          <Last_Name>Widom</Last_Name>
        </Professor>
      </Chair>
    −<Course Number="CS106A" Enrollment="1070">
        <Title>Programming Methodology</Title>
      −<Description>
          Introduction to the engineering of computer applications emphasizing modern software engineering principles.
        </Description>
      −<Instructors>
        −<Lecturer>
            <First_Name>Jerry</First_Name>
            <Middle_Initial>R.</Middle_Initial>
            <Last_Name>Cain</Last_Name>
          </Lecturer>
        −<Professor>
            <First_Name>Eric</First_Name>
            <Last_Name>Roberts</Last_Name>
          </Professor>
        −<Professor>
            <First_Name>Mehran</First_Name>
            <Last_Name>Sahami</Last_Name>
          </Professor>
```

XML Schema Overview

- XML Schema is a more powerful and flexible alternative to DTDs.
- The W3C XML Schema specification was published in 2001 and the latest version, 1.1, in 2012;
  - most software implements either 1.1 or 1.0 second edition from 2004.
- The actual XML Schema Documents that users create are called XSDs.
- A *schema* is any document that defines the structure of something
  - a DTD is a schema;
  - XML Schema is a particular schema language for XML

Benefits of W3C XML Schema

- Use basic XML element syntax
  - not DTD syntax.
- Supports XML Namespaces.
- Can constrain and validate the actual text
  - not just the elements and attributes.

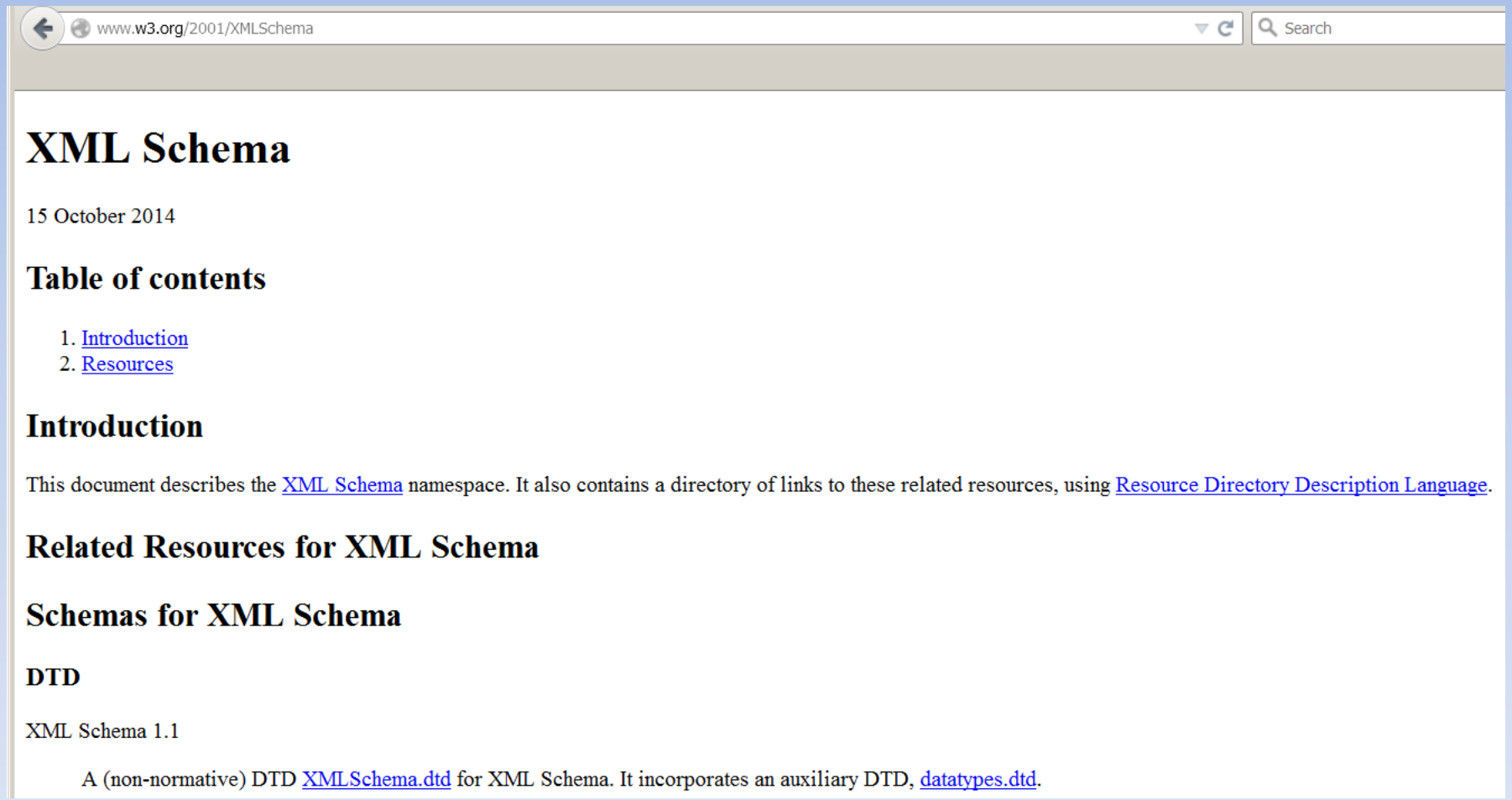
The Specifications

- You can find them at http://www.w3.org/XML/Schema or by going to http://www.w3.org/TR/tr-date-all and searching.

The xs:schema element

```
<? xml version = "1.0" encoding = "utf-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   . . .
</xs:schema>
```

http://www.w3.org/2001/XMLSchema



11.4.2 Elements

- Elements are an important part of XML
- Defining elements an important part of DTD's
- Defining elements an important part of XML Schema
  - NOTE: XML Schema is XML
  - XML Schema contains elements
  - Elements of the schema all begin with the tag "xs:" and are not part of the elements being defined by schema!

Element

<xs: element name = *element name* type = *element type* >

constraints and/or structure information

</xs:element>

Example 11.12: Here are title and year elements defined in XML Schema:

```
<xs:element name = "Title" type = "xs:string" />
<xs:element name = "Year" type = "xs:integer" />
```

- Tag is closed with />
- Needs no matching closing tag.

Complex Types

- Most common complex type is a sequence of elements.
- Repetition can be controlled by attributes:
  - minOccurs
  - maxOccurs
    - use maxOccurs= “unbounded”

Complex Types

```
<xs:complexType name = type name >
   <xs:sequence>
      list of element definitions
   </xs:sequence>
</xs:complexType>
```

Figure 11.11: Defining a complex type that is a sequence of elements

xs:element

- Declaring an element in XML Schema means associating the element name with a type, to say what the element must contain.

```
<xs:element name="name" type="type"
        ref="global element declaration"
        form="qualified|unqualified"
        minOccurs="min" maxOccurs="max",
        default="do not use", fixed="fixed value">
```

- The element name must be an XML name, starting with a Unicode letter or underscore, and not contain a colon.

```
1)  <? xml version = "1.0" encoding = "utf-8" ?>
2)  <xs:schema xmlns:xs = "http://www.w3.org/2001/XMLSchema">

3)     <xs:complexType name = "movieType">
4)        <xs:sequence>
5)           <xs:element name = "Title" type = "xs:string" />
6)           <xs:element name = "Year" type = "xs:integer" />
7)        </xs:sequence>
8)     </xs:complexType>

9)     <xs:element name = "Movies">
10)       <xs:complexType>
11)          <xs:sequence>
12)             <xs:element name = "Movie" type = "movieType"
                    minOccurs = "0" maxOccurs = "unbounded" />
13)          </xs:sequence>
14)       </xs:complexType>
15)    </xs:element>

16) </xs:schema>
```

Figure 11.12: A schema for movies in XML Schema

DTD

```
<!DOCTYPE Movies [
    <!ELEMENT Movies (Movie*)>
    <!ELEMENT Movie (Title, Year)>
    <!ELEMENT Title (#PCDATA)>
    <!ELEMENT Year (#PCDATA)>
]>
```



```
1)  <? xml version = "1.0" encodi
2)  <xs:schema xmlns:xs = "http:/

3)      <xs:complexType name = "movieType">
4)          <xs:sequence>
5)              <xs:element name = "Title" type = "xs:string" />
6)              <xs:element name = "Year" type = "xs:integer" />
7)          </xs:sequence>
8)      </xs:complexType>

9)      <xs:element name = "Movies">
10)         <xs:complexType>
11)             <xs:sequence>
12)                 <xs:element name = "Movie" type = "movieType"
                        minOccurs = "0" maxOccurs = "unbounded" />
13)             </xs:sequence>
14)         </xs:complexType>
15)     </xs:element>

16) </xs:schema>
```

Figure 11.12: A schema for movies in XML Schema

XML Schema

Built-in Schema Types 1

String Types

xs:string	Any XML character data
xs:normalizedString	string with spaces collapsed
xs:token	(specialized)
xs:byte	A number from -128 to 127 (8 bits signed)
xs:unsignedByte	A number from 0 to 255 (8 bits)
xs:base64Binary	An ASCII representation of binary data
xs:hexBinary	Base 16 representation of binary data

Built-in Schema Types 2

Integer Types

xs:integer	A whole number, arbitrary size
xs:positiveInteger	1, 2, 3, 4, etc.
xs:negativeInteger	-1, -2, -3, -4, etc
xs:nonNegativeInteger	0, 1, 2, …
xs:nonPositiveInteger	0, -1, -2,-3,…
xs:int, unsignedInt	(32-bit integer)
xs:long, unsignedLong	(64-bit integer)
xs:short, unsignedShort	(16-bit integer)

Built-in Schema Types 3

Floating-point Types

xs:decimal	arbitrary precision, e.g. 103.4242421
xs:float	IEEE 32-bit floating point

- In addition to numbers, you can use -0 (which is different from 0 in some systems) INF, -INF and NaN (Not a Number).
- Decimal and float can be positive or negative.
- There is no support for scientific notation (3.6E17).

Built-in Schema Types 4

Time and Date Types

xs:time	e.g. 15:45:17.000 (this is 3:45 pm)
xs:dateTime	e.g. 2016-07-17T15:45:17.000
xs:date	e.g. 2016-03-24 (year-month-day)

The date and time formats are defined by ISO 8601; although this is not freely available,
http://www.w3.org/TR/NOTE-datetime may be useful.

Built-in Schema Types 5

Durations

xs:duration a span of time: P12H is 12 hours

- Durations start with a P and then numbers followed by a unit, with Y, M, D (years months days), H, M, S (hours, minutes, seconds): P nY nM nD T nH nM nS (without spaces).
- The YMD and HMS can be omitted when not used, and the T can be omitted if there's no time part.
- E.g.: P14D (14 days); PT1H (one hour); P1D2H (26 hours)

Built-in Schema Types 6

Australian/Kangaroo Types

- These types all start with a “g”, like g’day mate, so the Working Group participants call them Australian.

xs:gYearMonth	e.g. 1998-07
xs:gYear	e.g. 2019
xs:gMonth	e.g. -07 (- followed by 2 digits!)
xs:gDay	e.g. --01 (-, -, 2 digits!)
xs:gMonthDay	e.g. -07-12 (July 12th)

Built-in Schema Types 7

Other Types

xs:boolean	true or false (1 and 0 also allowed)
xs:name	an XML name (can include a colon, :)
xs:QName	a namespace-Qualified name, e.g. a:b or b
xs:NCName	an unqualified (No Colon) name
xs:anyURI	Actually an IRI: international URL or URN
xs:language	A BCP47 language code, e.g. en-GB

see http://www.ietf.org/rfc/bcp/bcp47.txt

(*Not* RFC1766 mentioned in the book; it has changed)

Built-in Schema Types 8

XML Types

- You can also use the XML DTD types: ID, IDREF, IDREFS, ENTITY, ENTITIES, NOTATION, NOTATIONS and NMTOKENS; see the DTD module for more information.
- There are many types to remember! Most important are:
  - xs:integer
  - xs:string
  - xs:decimal
  - xs:dateTime

Declaring Attributes

- Use xs:attribute at the top level to define an attribute you can reference, or inside xs:complexType **as the last child** to add an attribute to an element:

```
<xs:complexType name="pageBreakType">
    <xs:attribute name="page" type="xs:integer" />
</xs:complexType>
<xs:element name="pageBreak" type="pageBreakType"
/>
```

- If you add pageBreak to the xs:choice in entryType, you can then include `<pageBreak page="23"/>` inside an entry element.

Optional Attributes

- The xs:attribute element can have a *use* attribute to say whether the attribute is required:
  - use="optional" - the attribute can be omitted; you can also add default="some value" and a schema processor *may* supply the default value on validation; this is not reliable.
  - use="required" - every instance of the element in the input must have this attribute, with a value which can be empty if the attribute's type allows it.
  - use="prohibited" - this is for advanced use, and disallows the attribute, e.g. when one type is based on another.

Declaring Attributes

```
<xs:attribute name = attribute name type = type name
    other information about the attribute />
```

Example 11.14: The notation

```
<xs:attribute name = "year" type = "xs:integer"
    default = "0" />
```

```
<xs:attribute name = "year" type = "xs:integer"
    use = "required" />
```

```
1)  <? xml version = "1.0" encoding = "utf-8" ?>
2)  <xs:schema xmlns:xs = "http://www.w3.org/2001/XMLSchema">

3)     <xs:complexType name = "movieType">
4)        <xs:attribute name = "title" type = "xs:string"
              use = "required" />
5)        <xs:attribute name = "year" type = "xs:integer"
              use = "required" />
6)     </xs:complexType>

7)     <xs:element name = "Movies">
8)        <xs:complexType>
9)           <xs:sequence>
10)             <xs:element name = "Movie" type = "movieType"
                    minOccurs = "0" maxOccurs = "unbounded" />
11)          </xs:sequence>
12)       </xs:complexType>
13)    </xs:element>

14) </xs:schema>
```

Figure 11.14: Using attributes in place of simple elements

```
1)  <? xml version = "1.0" encoding = "utf-8" ?>
2)  <xs:schema xmlns:xs = "http://www.w3.org/2001/XMLSchema">

3)     <xs:complexType name = "movieType">
4)        <xs:attribute name = "title" type = "xs:string"
              use = "required" />
5)        <xs:attribute name = "year" type = "xs:integer"
              use = "required" />
6)     </xs:complexType>

7)     <xs:element name = "Movies">
8)        <xs:complexType>
9)           <xs:sequence>
10)             <xs:element name = "Movie" type = "movieType"
                    minOccurs = "0" maxOccurs = "unbounded" />
11)          </xs:sequence>
12)       </xs:complexType>
13)    </xs:element>

14) </xs:schema>
```

Figure 11.14: Using attrib

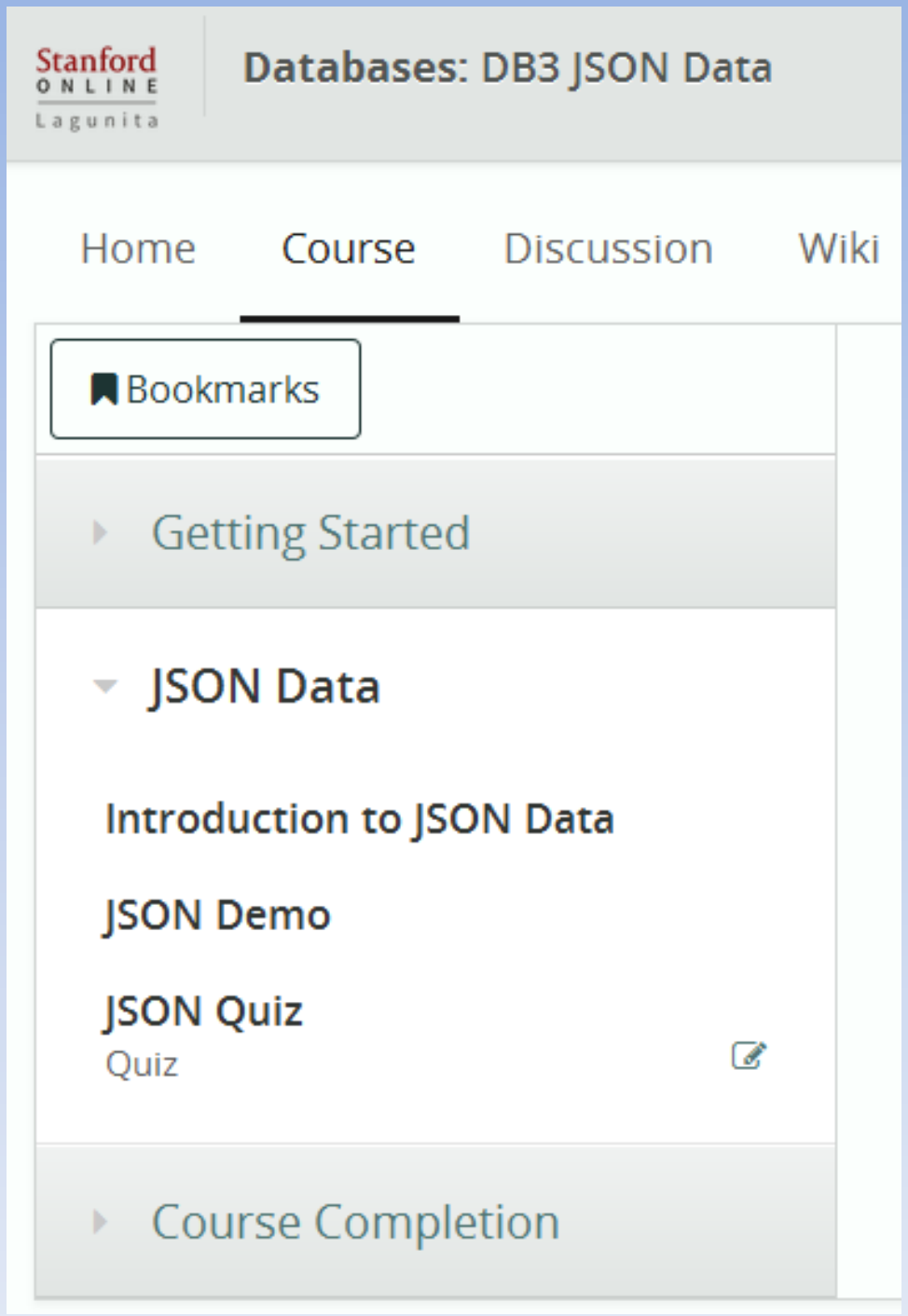

```
<!DOCTYPE Movies [
    <!ELEMENT Movies (Movie*)>
    <!ELEMENT Movie EMPTY>
        <!ATTLIST Movie
            title CDATA #REQUIRED
            year  CDATA #REQUIRED
        >
]>
```

Figure 11.15: DTD equivalent for Fig. 11.14

JavaScript Object Notation (JSON)

- Standard for serializing data objects (especially in files)
- Replacing XML for internet data/data interchange
- Self-Describing
- Great for semistructured data.

JSON


Stanford
ONLINE
Lagunita
Databases: DB3 JSON Data
Home
Course
Discussion
Wiki
Bookmarks
Getting Started
JSON Data
Introduction to JSON Data
JSON Demo
JSON Quiz
Quiz
Course Completion

JSON Example


```
{
	"_id" : 0,
	"name" : "aimee Zank",
	"scores" : [
		{
			"score" : 1.463179736705023,
			"type" : "exam"
		},
		{
			"score" : 11.78273309957772,
			"type" : "quiz"
		},
		{
			"score" : 35.8740349954354,
			"type" : "homework"
		}
	]
}
```

JSON Constructs

- Base values:
  - Numbers
  - Strings
  - Boolean
  - ….
- Objects
  - Set of Label:Value Pairs.
- Arrays
  - List of values

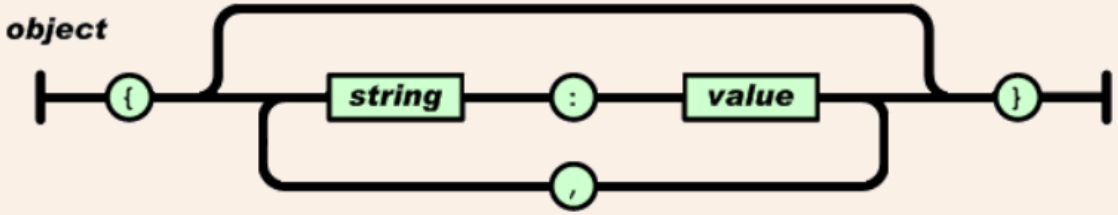
JSON Example


```
{
    "_id" : 0,
    "name" : "aimee Zank",
    "scores" : [
        {
            "score" : 1.463179736705023,
            "type" : "exam"
        },
        {
            "score" : 11.78273309957772,
            "type" : "quiz"
        },
        {
            "score" : 35.8740349954354,
            "type" : "homework"
        }
    ]
}
```

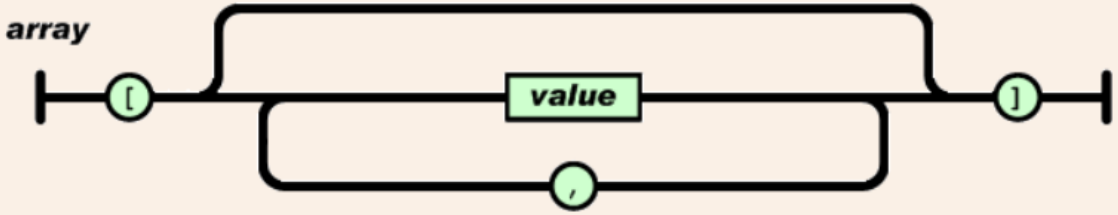
JSON w/ Python

- JSON files almost identical to Python Dictionaries
- Python Dictionaries are UNORDERED
  - May need to use list of ordered Pairs in some cases to preserve order in Python:
    - [(key1,value1), (key2,value2), … (keyn, valuen)]

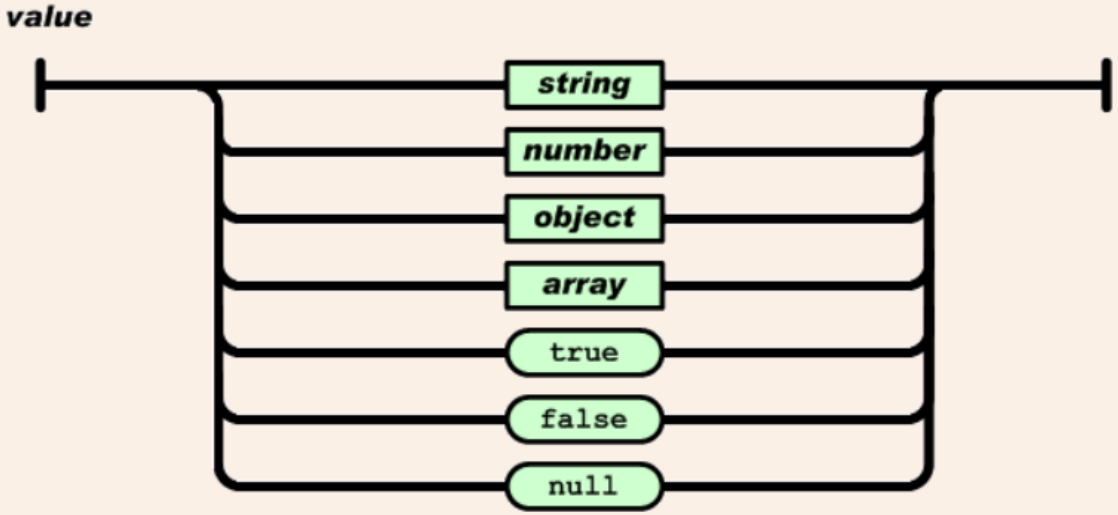
json.org



An *array* is an ordered collection of values. An array begins with `[` (left bracket) and ends with `]` (right bracket). Values are separated by `,` (comma).



A *value* can be a *string* in double quotes, or a *number*, or `true` or `false` or `null`, or an *object* or an *array*. These structures can be nested.



Legal JSON Object?

```
{ "name": "Smiley",
"age": 20,
"phone": { "888-123-4567", "888-765-4321" },
"email": "smiley@xyz.com",
"happy": true }
```

Legal JSON Object?

```
{ "name": "Smiley",
"age": 20,
"phone": null,
"email": "null",
"happy": true }
```

Legal JSON Object?

```
{ "name": "Smiley",
"age": 20,
"phone": {},
"email": "smiley@xyz.com",
"happy": true }
```

Legal JSON Object?

```
{ "name": "Smiley",
"age": 20,
"phone": null,
"email": null,
"happy": true }
```

Bookstore.json

```
{ "Books":
  [
    { "ISBN":"ISBN-0-13-713526-2",
      "Price":85,
      "Edition":3,
      "Title":"A First Course in Database Systems",
      "Authors":[ {"First_Name":"Jeffrey", "Last_Name":"Ullman"},
                  {"First_Name":"Jennifer", "Last_Name":"Widom"} ] }
    ,
    { "ISBN":"ISBN-0-13-815504-6",
      "Price":100,
      "Remark":"Buy this book bundled with 'A First Course' - a great deal!",
      "Title":"Database Systems:The Complete Book",
      "Authors":[ {"First_Name":"Hector", "Last_Name":"Garcia-Molina"},
                  {"First_Name":"Jeffrey", "Last_Name":"Ullman"},
                  {"First_Name":"Jennifer", "Last_Name":"Widom"} ] }
  ],
  "Magazines":
  [
    { "Title":"National Geographic",
      "Month":"January",
      "Year":2009 }
    ,
    { "Title":"Newsweek",
      "Month":"February",
      "Year":2009 }
  ]
}
```

JSON Schema

- Describes your existing data format
- clear, human- and machine-readable documentation
- complete structural validation, useful for
  - automated testing
  - validating client-submitted data

JSON Schema

json-schema.org

The home of JSON Schema

about | docs | examples | software

Basic example

Here is a basic example of a JSON Schema:

```
{
        "title": "Example Schema",
        "type": "object",
        "properties": {
                "firstName": {
                        "type": "string"
                },
                "lastName": {
                        "type": "string"
                },
                "age": {
                        "description": "Age in years",
                        "type": "integer",
                        "minimum": 0
                }
        },
        "required": ["firstName", "lastName"]
}
```

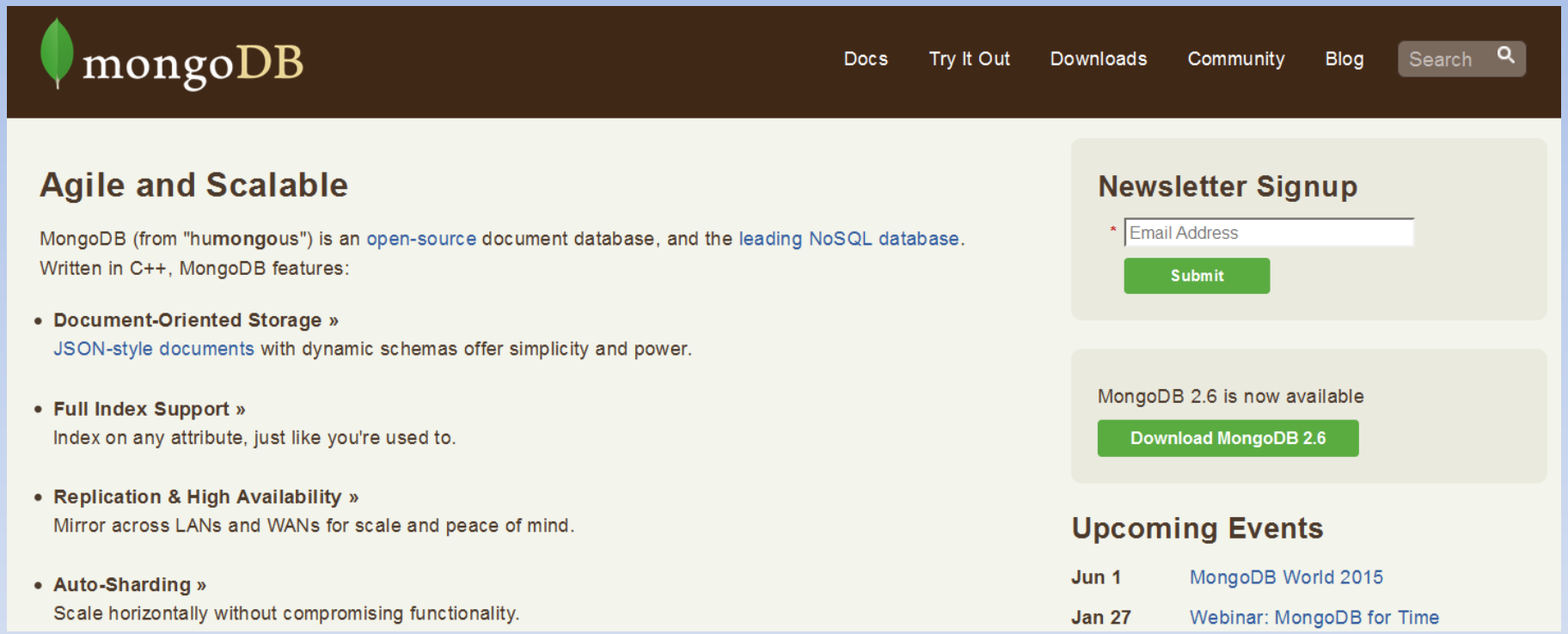
JSON Schema

```
{ "type":"object",
  "properties": {
     "Books": {
        "type":"array",
        "items": {
           "type":"object",
           "properties": {
              "ISBN": { "type":"string", "pattern":"ISBN*" },
              "Price": { "type":"integer",
                         "minimum":0, "maximum":200 },
              "Edition": { "type":"integer", "optional": true },
              "Remark": { "type":"string", "optional": true },
              "Title": { "type":"string" },
              "Authors": {
                 "type":"array",
                 "minItems":1,
                 "maxItems":10,
                 "items": {
                    "type":"object",
                    "properties": {
                       "First_Name": { "type":"string" },
                       "Last_Name": { "type":"string" }}}}}},
     "Magazines": {
        "type":"array",
        "items": {
           "type":"object",
           "properties": {
              "Title": { "type":"string" },
              "Month": { "type":"string",
                         "enum":["January","February"] },
              "Year": { "type":"integer" }}}}
}}
```

Bookstore.json

```
{ "Books":
  [
    { "ISBN":"ISBN-0-13-713526-2",
      "Price":85,
      "Edition":3,
      "Title":"A First Course in Database Systems",
      "Authors":[ {"First_Name":"Jeffrey", "Last_Name":"Ullman"},
                  {"First_Name":"Jennifer", "Last_Name":"Widom"} ] }
    ,
    { "ISBN":"ISBN-0-13-815504-6",
      "Price":100,
      "Remark":"Buy this book bundled with 'A First Course' - a great deal!",
      "Title":"Database Systems:The Complete Book",
      "Authors":[ {"First_Name":"Hector", "Last_Name":"Garcia-Molina"},
                  {"First_Name":"Jeffrey", "Last_Name":"Ullman"},
                  {"First_Name":"Jennifer", "Last_Name":"Widom"} ] }
  ],
  "Magazines":
  [
    { "Title":"National Geographic",
      "Month":"January",
      "Year":2009 }
    ,
    { "Title":"Newsweek",
      "Month":"February",
      "Year":2009 }
  ]
}
```

MongoDB


mongoDB
Docs
Try It Out
Downloads
Community
Blog
Search
Agile and Scalable
MongoDB (from "humongous") is an open-source document database, and the leading NoSQL database.
Written in C++, MongoDB features:
Document-Oriented Storage »
JSON-style documents with dynamic schemas offer simplicity and power.
Full Index Support »
Index on any attribute, just like you're used to.
Replication & High Availability »
Mirror across LANs and WANs for scale and peace of mind.
Auto-Sharding »
Scale horizontally without compromising functionality.
Newsletter Signup
* Email Address
Submit
MongoDB 2.6 is now available
Download MongoDB 2.6
Upcoming Events
Jun 1 MongoDB World 2015
Jan 27 Webinar: MongoDB for Time

What is MongoDB

- MongoDB is an open-source document database that provides high performance, high availability, and automatic scaling.
- Document Database
  - A record in MongoDB is a document, which is a data structure composed of field and value pairs.
  - MongoDB documents are similar to JSON objects. The values of fields may include other documents, arrays, and arrays of documents.

MongoDB

- The advantages of using documents are:
  - Documents (i.e. objects) correspond to native data types in many programming languages.
  - Embedded documents and arrays reduce need for expensive joins.
  - Dynamic schema supports fluent polymorphism.
    - Different Objects with same subset of elements can be treated the same.