XML Schema: complexType

- Specifies types of elements with structure:
  - Must use a **compositor** if $\geq 1$ subelements
  - Subelements with types
  - Min and max occurrences (default 1) of subelements
- Elements with text content are easy (like simple type)
- EMPTY elements: easy
  - Example?
  - Compare to nulls, later
XML Schema: Compositors

- **Sequence**: ordered list of subelements
  - Can occur within other compositors
  - Allows varying min and max occurrence
- **All**: unordered subelements
  - Must occur directly below root element
  - Max occurrence of each element is 1
- **Choice**: exclusive or: include one subelement
  - Can occur within other compositors
XML Schema: Main Namespaces

Part of the standard

- **xsd**: http://www.w3.org/2001/XMLSchema
  - Terms for defining schemas: schema, element, attribute, ...
  - The schema element has an attribute targetNamespace

- **xsi**: http://www.w3.org/2001/XMLSchema-instance
  - Terms for use in instances: schemaLocation, noNamespaceSchemaLocation, nil, type
  - targetNamespace: user-defined
XML Schema Instance Doc

<!— Comment —>

<Music xmlns="http://a.b.c/Muse"
       xmlns:xsi="the standard-xsi"
       xsi:schemaLocation="schema-URL schema-location-URL">
  <Music>
  ...
</Music>

Define null values as

<aElem xsi:nil="true"/>  ≠  Element with no text
XML Schema: Nullable

- An xsd:element declaration may state nullable='true'
  - An instance of the element might state xsi:nil="true"
- The instance would be valid even if no content is present, even if content is required by default
Creating XML Schema Docs: 1

Included into the same namespace as the including doc

```xml
<xsd:schema xmlns:xsd="the-standard-xsd"
    xmlns:xs="" the-target"">  
    <include xsd:schemaLocation="part-one.xsd"/>
4 <include xsd:schemaLocation="part-two.xsd"/>
    <!-- schemaLocation as in xsd, not xsi -->
</xsd:schema>
```
Creating XML Schema Docs: 2

- Use import instead of include
  - Imports may have different targets
  - Included schemas have the same target
  - Specify namespaces from which schemas are to be imported
  - Location of schemas not required and may be ignored if provided
Foreign Attributes in XML Schema

XML Schema elements allow attributes that are foreign, i.e., with a namespace other than the xsd namespace

- Must have an explicit namespace
- Can be used to insert any additional information, not interpreted by a processor
- Specific usage is with attributes from the xlink: namespace

```xml
<xsd:schema>
  <xsd:element name='course' type='cT'
                xlink:role='work' ncsu:offering='true'>
4  </xsd:schema>
```
XML Schema Style Guidelines: 1

- Flatten the structure of the schema
  - Don’t nest declarations as you would a desired instance document
  - Make sure that element names are not reused
- Unqualified attributes cannot be global
- If dealing with legacy documents with the same element names having different meanings, place them in different namespaces where possible
- Use named types where appropriate
XML Schema Style Guidelines: 2

- Don’t have elements with mixed content
- Don’t have attribute values that need parsing
- Add unique IDs for information that may repeat
- Group information that may repeat
- Emphasize commonalities and reuse
  - Derive types from related types
  - Create attribute groups

```xml
<p>ABC</p><b>DEF</b><i>GHI</i><br/>

<student> Kun-Ta  <name>Tsai</name></student>

<fname>  <lname>
```
XML Schema Documentation

xsd:annotation

- Should be the first subelement, except for the whole schema
- Container for two mixed-content subelements
  - xsd:documentation: for humans
  - xsd:appinfo: for machine-processible data
    - Such as application-specific metadata
    - Possibly using the Dublin Core vocabulary, which describes library content and other media
Module 3: XML Manipulation

Key XML query and manipulation languages include

- XPath
- XQuery
- XSLT

Later SQL
Metaphors for Handling XML: 1

How we conceptualize XML documents determines our approach for handling them

- **Text**: an XML document is text
  - Ignore any structure and perform simple pattern matches

- **Tags**: an XML document is text interspersed with "tags"
  - Treat each tag as an "event" during reading a document, as in SAX (Simple API for XML)  
    Specify callbacks
  - Construct regular expressions as in screen scraping
<Sgr ... ></Sgr></Sgr>
Metaphors for Handling XML: 2

- **Tree**: an XML document is a tree
  - Walk the tree using DOM (Document Object Model)

- **Template**: an XML document has regular structure
  - Let XPath, XSLT, XQuery do the work

- **Thought**: an XML document represents an information model
  - Access knowledge via RDF or OWL
XPath

Used as part of XPointer, SQL/XML, XQuery, and XSLT

- Models XML documents as trees with nodes
  - Elements
  - Attributes
  - Text (PCDATA)
  - Comments
- Root node: above root of document

Namespace declarations
Processing instructions
WAYS TO NAVIGATE A TREE

- Parent (...)
- Children
- Leaves
- Sibling < before, > after
- Descendants
- Ancestors
- Root
Achtung!

- Parent in XPath is like parent as traditionally in computer science
- Child in XPath is confusing:
  - An attribute is not a child of its parent
  - Makes a difference for recursion (e.g., in XSLT apply-templates)
- Our terminology follows computer science:
  - e-children, a-children, t-children
  - Sets via et-, ta-, and so on
XPath Location Paths: 1

- Relative or absolute
- Reminiscent of file system paths, but much more subtle
  - Name of an element to walk down
  - Leading /: root
  - /: indicates walking down a tree
  - .: currently matched (context) node
  - ../: parent node