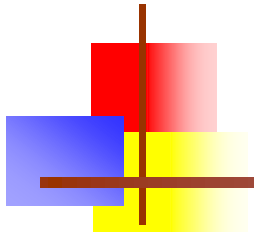
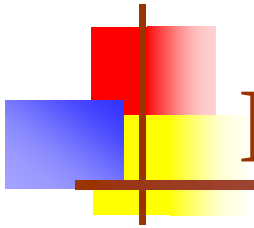


# XML



# HTML and XML

XML stands for eXtensible Markup Language

HTML is used to mark up text so it can be displayed to users

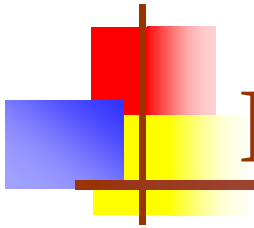
XML is used to mark up data so it can be processed by computers

HTML describes both structure (e.g. <p>, <h2>, ) and appearance (e.g. <br>, <font>, <i>)

XML describes only content, or "meaning"

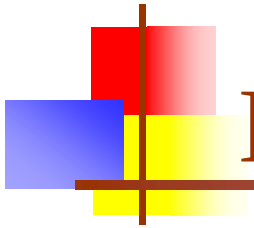
HTML uses a fixed, unchangeable set of tags

In XML, you make up your own tags



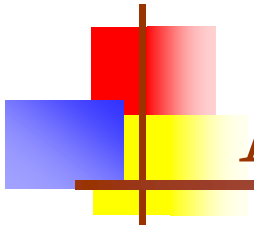
# HTML and XML

- HTML and XML look similar, because they are both SGML languages (SGML = Standard Generalized Markup Language)
  - Both HTML and XML use elements enclosed in tags (e.g. <body>This is an element</body>)
  - Both use tag attributes (e.g., <font face="Verdana" size="+1" color="red">)
  - Both use entities (<, >, &, ", ')
- More precisely,
  - HTML is defined in SGML
  - XML is a (very small) subset of SGML



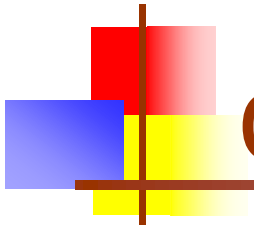
# Example XML document

```
<?xml version="1.0"?>
<weatherReport>
   <date>7/14/97</date>
   <city>North Place</city>, <state>NX</state>
   <country>USA</country>
   High Temp: <high scale="F">103</high>
   Low Temp: <low scale="F">70</low>
   Morning: <morning>Partly cloudy, Hazy</morning>
   Afternoon: <afternoon>Sunny &amp;amp; hot</afternoon>
   Evening: <evening>Clear and Cooler</evening>
</weatherReport>
```



# Another Example

```
<?xml version = "1.0"?>
<article>
<title>Simple XML</title>
 <date>September 19, 2001</date>
 <author>
          <firstName>Tem</firstName>
           <lastName>Nieto</lastName>
 </author>
 <summary>XML is pretty easy.</summary>
 <content>Once you have mastered XHTML, XML is easily
learned. You must remember that XML is not for
displaying information but for managing information.
 </content>
 </article>
```



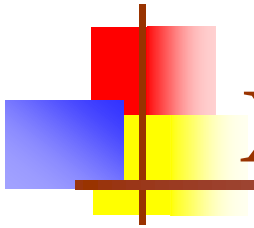
# Overall structure

- An XML document may start with one or more processing instructions (PIs) or directives:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="ss.css"?>
```

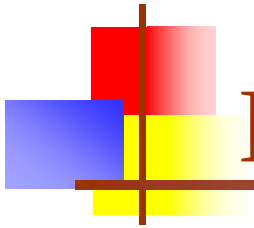
- Following the directives, there must be exactly *one* root element containing all the rest of the XML:

```
<weatherReport>
    ...
</weatherReport>
```



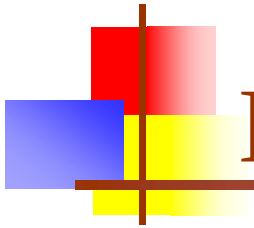
# XML building blocks

- Aside from the directives, an XML document is built from:
  - elements: high in <high scale="F">103</high>
  - tags, in pairs: <high scale="F">103</high>
  - attributes: <high scale="F">103</high>
  - entities: <afternoon>Sunny & hot</afternoon>
  - character data, which may be:
    - parsed (processed as XML)--this is the default
    - unparsed (all characters stand for themselves)



# Entities

- Five special characters must be written as entities:
  - `&` for `&` (almost always necessary)
  - `<` for `<` (almost always necessary)
  - `>` for `>` (not usually necessary)
  - `&quot;` for `"` (necessary inside double quotes)
  - `&apos;` for `'` (necessary inside single quotes)
- These entities can be used even in places where they are not absolutely required
- These are the *only* predefined entities in XML



# Elements and attributes

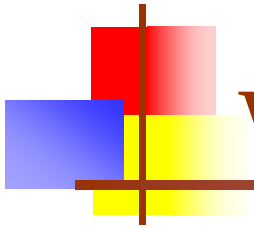
- Attributes and elements are somewhat interchangeable
- Example using just elements:

```
<name>
   <first>David</first>
   <last>Matuszek</last>
</name>
```

- Example using attributes:

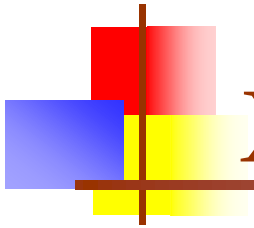
```
<name first="David" last="Matuszek"></name>
```

- You will find that elements are easier to use in your programs--this is a good reason to prefer them
- Attributes often contain metadata, such as unique IDs
- Generally speaking, browsers display only elements (values enclosed by tags), not tags and attributes



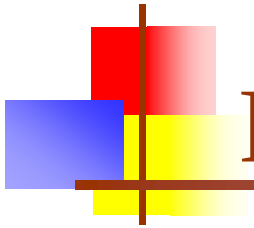
# Well-formed XML Document

- Every element must have *both* a start tag and an end tag, e.g. <name> ... </name>
  - But empty elements can be abbreviated: <break />.
  - XML tags are case sensitive
  - XML tags may not begin with the letters xml, in any combination of cases
- Elements must be properly nested, e.g. *not* bold and italic
- Every XML document must have one and only one root element
- The values of attributes must be enclosed in single or double quotes, e.g. <time unit="days">
- Character data cannot contain < or &



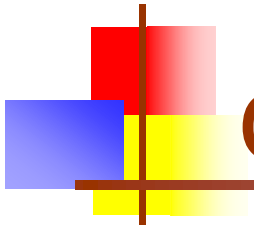
# XML declaration

- The XML declaration looks like this:
  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
  - The XML declaration is not required by browsers, but *is* required by most XML processors (so include it!)
  - If present, the XML declaration must be first--*not even whitespace* should precede it
  - Note that the brackets are <? and ?>
  - version="1.0" is required (this is the *only* version so far)
  - encoding can be "UTF-8" (ASCII) or "UTF-16" (Unicode), or something else, or it can be omitted
  - standalone tells whether there is a separate DTD



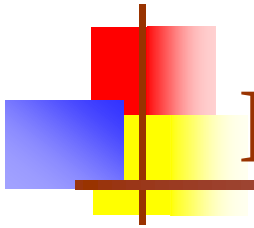
# Processing instructions

- PIs (Processing Instructions) may occur anywhere in the XML document (but usually first)
- A PI is a command to the program processing the XML document to handle it in a certain way
- XML documents are typically processed by more than one program
- Programs that do not recognize a given PI should just ignore it
- General format of a PI: <?*target instructions*?>
- Example: <?xml-stylesheet type="text/css" href="mySheet.css"?>



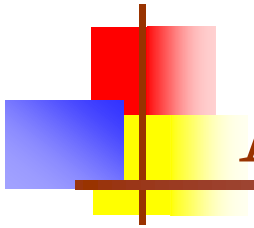
# Comments

- `<!-- This is a comment in both HTML and XML -->`
- Comments can be put anywhere in an XML document
- Comments are useful for:
  - Explaining the structure of an XML document
  - Commenting out parts of the XML during development and testing
- Comments are not elements and do not have an end tag
- The blanks after `<!--` and before `-->` are optional
- The character sequence `--` cannot occur in the comment
- The closing bracket *must* be `-->`
- Comments are not displayed by browsers, but can be seen by anyone who looks at the source code



# Names in XML

- Names (as used for tags and attributes) must begin with a letter or underscore, and can consist of:
  - Letters, both Roman (English) and foreign
  - Digits, both Roman and foreign
    - . (dot)
    - - (hyphen)
    - _ (underscore)
    - : (colon) should be used only for namespaces
  - Combining characters and extenders (not used in English)

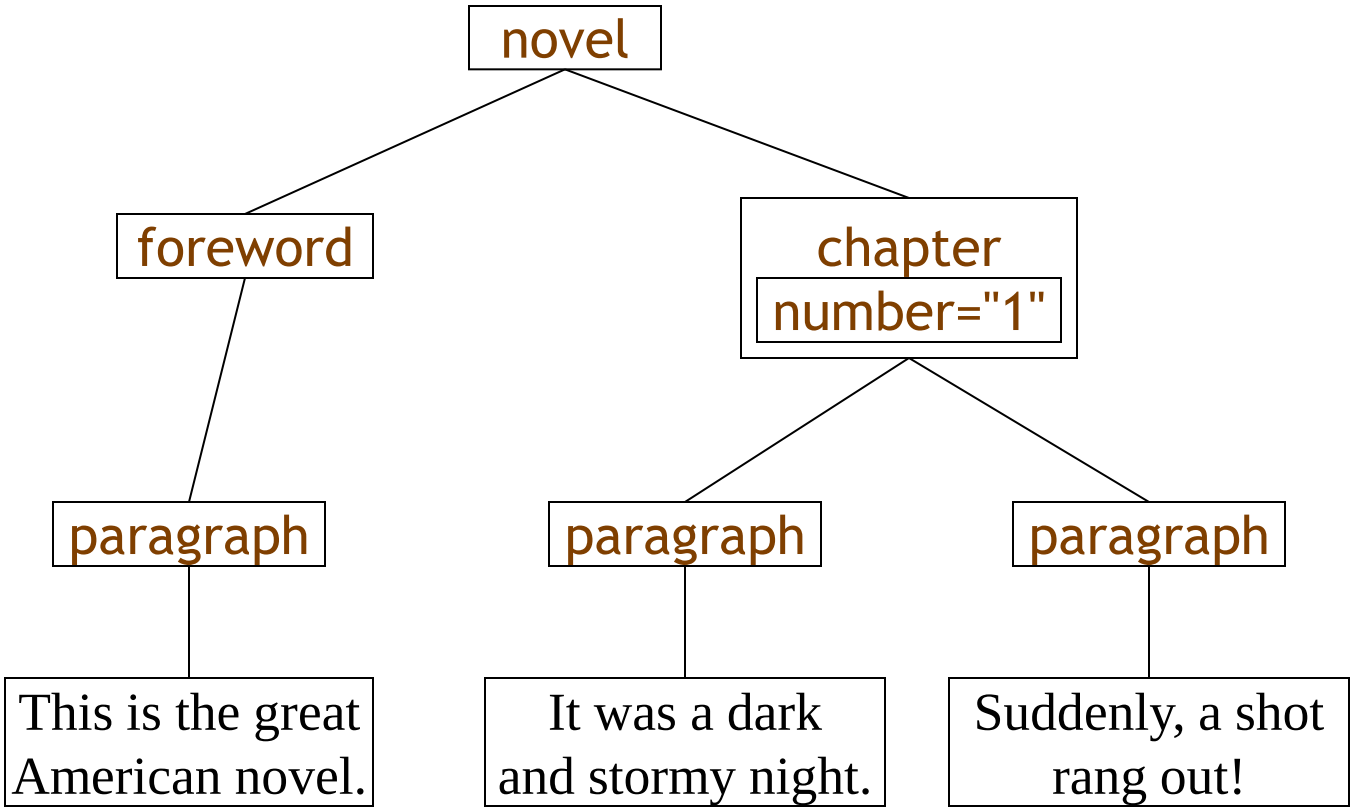


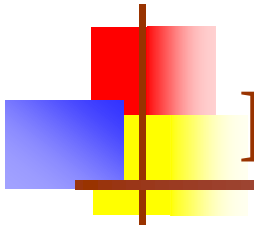
# Another well-structured example

```
<novel>
   <foreword>
      <paragraph> This is the great American novel.
      </paragraph>
</foreword>
   <chapter number="1">
      <paragraph>It was a dark and stormy night.
     </paragraph>
      <paragraph>Suddenly, a shot rang out!
      </paragraph>
   </chapter>
</novel>
```

# XML as a tree

- An XML document represents a hierarchy; a hierarchy is a tree





# Document Type Definition (DTD)

- The purpose of a DTD is to define the building blocks of an XML document.
- It defines the Document structure with a list of legal elements.
- A DTD can be declared in two ways
  - Inline DTD Declaration in the XML document itself
  - External DTD reference to XML Document.

# Internal DTD Declaration

- If the DTD is included in the XML source file, DTD definition should be wrapped in a DOCTYPE definition along with XML Definition.
- Syntax

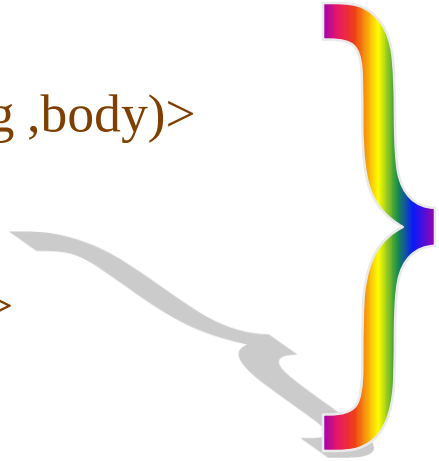
<?xml version="1.0">

<!DOCTYPE root-element [element-declarations]>

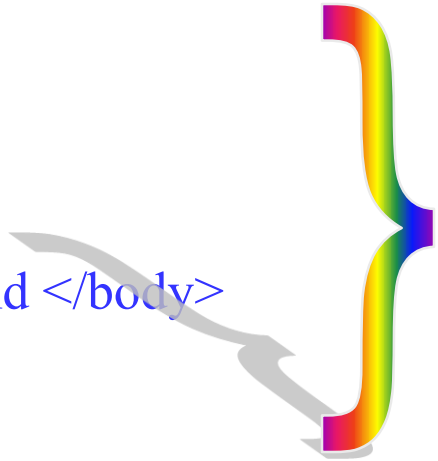
XML definition

# Simple Example

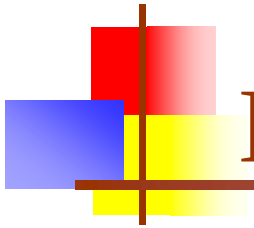
```
<?xml version="1.0">

<!DOCTYPE note [
    <!ELEMENT note (to ,from ,heading ,body)>
    <!ELEMENT to (#PCDATA)>
    <!ELEMENT from (#PCDATA)>
    <!ELEMENT heading (#PCDATA)>
    <!ELEMENT body (#PCDATA)>
 ]>

<note>
    <to> Tove </to>
    <from> Jani</from>
    <heading>Reminder </heading>
    <body> Don't forget me this weekend </body>
</note>
```



DTD



XML



# External DTD Declaration

- If the DTD is external to the XML source file, the external DTD file should be specified in a DOCTYPE definition without writing DTD definition again.
- Syntax

<?xml version="1.0">

<!DOCTYPE root-element SYSTEM "filename">

XML definition

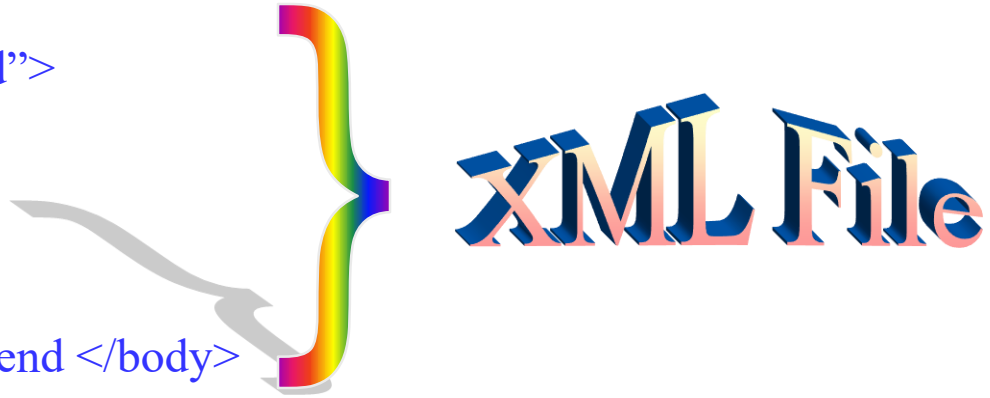
# Simple Example

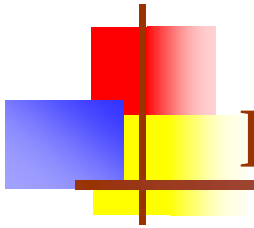
This is a copy of the file “note.dtd” containing the DTD

```
<!ELEMENT note (to ,from ,heading ,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```


This is the XML document with an external DTD

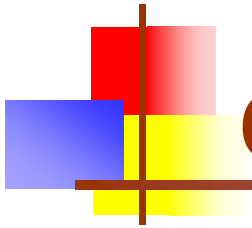
```
<?xml version=“1.0”>
<!DOCTYPE note SYSTEM “note.dtd”>
<note>
    <to> Tove </to>
    <from> Jani</from>
    <heading>Reminder </heading>
    <body> Don’t forget me this weekend </body>
</note>
```





# Basic Building Blocks of XML Document

- According to DTD point of view , all XML documents are made up by the building blocks like
  - Elements
  - Tags
  - Attributes
  - Entities
  - CDATA
  - PCDATA



# CDATA and PCDATA

- CDATA
  - CDATA means character data which is a text that will NOT be parsed by XML parsers.
  - Tags inside the text will NOT be treated as markup and entities will not be expanded.
- PCDATA
  - PCDATA means Parsable character data which is a text that will be parsed by XML Parser
  - Tags inside the text will NOT be treated as markup and entities will not be expanded.

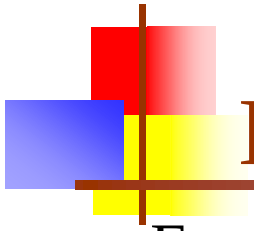
# Element Declaration in DTD

- In the DTD, XML elements are declared with element declaration.
- Syntax of an element declaration

<!ELEMENT element-name category>

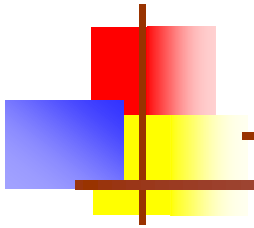
Or

<!ELEMENT element-name (element-content)>



# Different types of Element Declarations

- Empty Element Declaration
  - Empty elements are declared with the category keyword EMPTY
  - Syntax
    - <!ELEMENT element-name EMPTY>
  - DTD Example
    - <!ELEMENT br EMPTY>
  - XML Example
    - < br />
- Elements with only character data
  - Elements with only character data are declared with #PCDATA inside parenthesis
  - Syntax
    - <!ELEMENT element-name (#PCDATA)>
  - Example
    - <!ELEMENT from (#PCDATA)>



# - Contd…

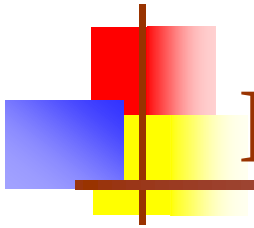
- Elements with a sequence of children
  - Elements with one or more children are defined with the name of the children elements inside parenthesis
  - Syntax
    - <!ELEMENT element-name (child element1, child element2, child element3,..) >
    - Example
      - <!ELEMENT note (to , from ,heading ,body)>
  - When children are declared in a sequence separated by comma, the children must be appear in the same sequence in the document.

# - Contd..

- Declaring only one occurrence of the element
  - Syntax
    - <!ELEMENT element –name (child-name)>
  - Example
    - <!ELEMENT note(to,from.heading,body)>
- Declaring minimum one occurrence of the same element
  - Syntax
    - <!ELEMENT element-name (child-name+)>
  - Example
    - <!ELEMENT students(student+)>
- Declaring zero or more occurrence of the same content
  - Syntax
    - <!ELEMENT element-name (child-name*)>
  - Example
    - <!ELEMENT students(students*)>

# - Contd ..

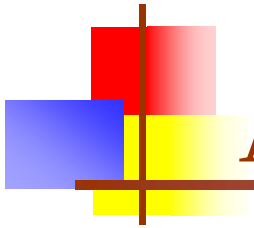
- Declaring zero or one occurrence of the same element
  - Syntax
    - <!ELEMENT element-name (child-name?)>
  - Example
    - <!ELEMENT note (message?)>
- Declaring either/or content
  - Syntax
    - <!ELEMENT element-name(child-name1|child-name2)
  - Example
    - <!ELEMENT note(to,from,header,(message|body))>
- Declaring Mixed Content
  - Syntax
    - <!ELEMENT element-name(child-name1 | child-name2,..)>
  - Example
    - <!ELEMENT note(#PCDATA | to | from | header | message)>



# Declaring Attributes

- In the DTD, XML element attributes are declared with an ATTLIST declaration. An attribute declaration has the syntax like:
  - <!ATTLIST element-name attribute-name attribute-type default-value>

- For the attributes it have the properties like
  - Name of the attribute
  - Type of the attributes
  - Default value if any.



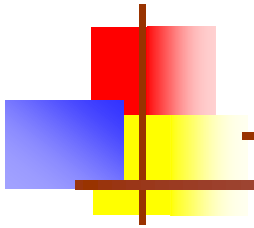
# Attribute Default Value

- The last entry in the attribute specification determines the attributes default value, if any, and tells whether or not the attribute is required.

| ***Specification*** | ***Specifies...*** |
|---|---|
| #REQUIRED | The attribute value must be specified in the document. |
| #IMPLIED | The value need not be specified in the document. If it isn't, the application will have a default value it uses. |
| "defaultValue" | The default value to use, if a value is not specified in the document. |
| #FIXED "fixedValue" | The value to use. If the document specifies any value at all, it must be the same. |

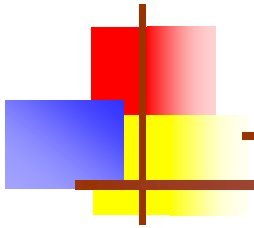
# Attribute declaration examples

- Default attribute value
  - Syntax:
    - <!ATTLIST element-name attribute-name CDATA "default-value">
  - DTD example:
    - <!ATTLIST payment type CDATA "cheque">
  - XML example:
    - <payment type=" cheque ">
- Specifying #IMPLIED
  - Syntax:
    - <!ATTLIST element-name attribute-name attribute-type #IMPLIED>
  - DTD example: **<!ATTLIST contact fax CDATA #IMPLIED>**
  - XML example: **<contact fax="555-667788">**



# - Contd..

- Specifying #REQUIRED
  - Syntax:
    - <!ATTLIST element-name attribute_name attribute-type #REQUIRED>
  - DTD example:
    - <!ATTLIST person number CDATA #REQUIRED>
  - XML example:
    - <person number="5677">
- Specifying #FIXED
  - Syntax:
    - <!ATTLIST element-name attribute-name attribute-type #FIXED "value">
  - DTD example:
    - <!ATTLIST sender company CDATA #FIXED "Microsoft">
  - XML example:
    - <sender company="Microsoft">



# - Contd..

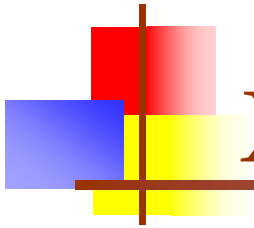
- Enumerated attribute values
  - Syntax:
    - <!ATTLIST element-name attribute-name (val1|val2|..) default-value>
  - DTD example:
    - <!ATTLIST payment type (cheque |cash) "cash">
  - XML example:
    - <payment type=" cheque "> or <payment type="cash">

# Declaring Entities

- Entities are the variables used to define shortcuts to common text.
- <!ENTITY> PI is used to declare user-defined entities.
- Syntax
  - <!ENTITY entity-name “entity-value”>
- DTD Example
  - <!ENTITY writer “Donald Duck.”>
  - <!ENTITY copyright “Copyright W3Schools>
  - <!ENTITY lt “<“ >

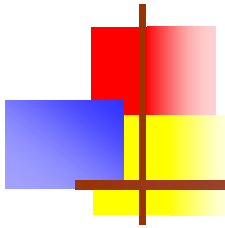
**XML Example**

  - <author> &writer; &copyright; </author>
  - 20 < 25 ( Means 20<25)



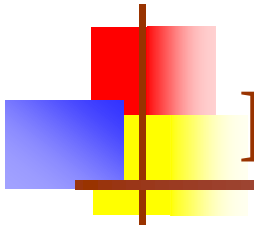
# XML Schema

- What is XML Schema?
  - XML Schema is vocabulary for expressing constraints for the validity of an XML document.
  - A piece of XML is valid if it satisfies the constraints expressed in another XML file, the schema file.
  - The idea is to check if the XML file is fit for a certain purpose.
  - When we say “XML Schemas,” we usually mean the W3C XML Schema Language
    - This is also known as “XML Schema Definition” language, or xs



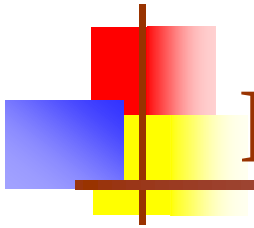
# Need of XML Schema

- DTDs provide a very weak specification language
  - We can’t put any restrictions on text content
  - We have very little control over mixed content (text plus elements)
  - We have little control over ordering of elements
- DTDs are written in a strange (non-XML) format
  - We need separate parsers for DTDs and XML
- The XML Schema Definition language solves these problems
  - xs gives us much more control over structure and content
  - xs is written in XML



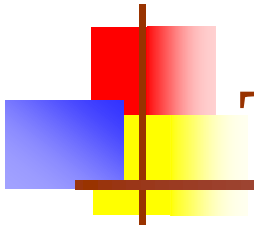
# Disadvantages of XML Schema

- DTDs have been around longer than xs
  - Therefore they are more widely used
  - Also, more tools support them
- xs is very verbose, even by XML standards
- More advanced XML Schema instructions can be non-intuitive and confusing
- Nevertheless, xs is not likely to go away quickly



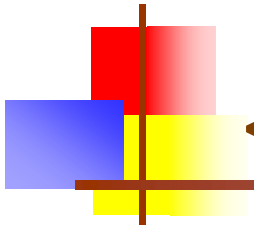
# Referring to a schema

- To refer to a DTD in an XML document, the reference goes *before* the root element:
  - <?xml version="1.0"?>
    <!DOCTYPE rootElement SYSTEM "*url*">
    <rootElement> ... </rootElement>
- To refer to an XML Schema in an XML document, the reference goes *in* the root element:
  - <?xml version="1.0"?>
    <rootElement
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    (The XML Schema Instance reference is required)
    xsi:schemaLocation="*url* xs">
    (This is where *your* XML Schema definition can be found)
    ...
    </rootElement>



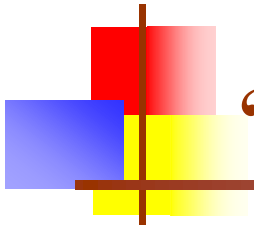
# The xs Document

- To prepare the XML document structure for the XML file, first we need to prepare the xs document.
- XML Schema is itself be XML file
- The file extension is .xs
- The root element is <schema>
- The xs starts like this:
  - <?xml version="1.0"?>
    <xs:schema
    xmlns:xs="http://www.w3.rg/2001/XMLSchema">



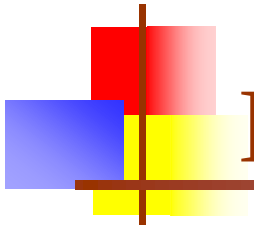
# <schema>

- The <schema> element may have attributes:
  - xmlns:xs="http://www.w3.org/2001/XMLSchema"
    - This is necessary to specify where all our xs tags are defined
  - elementFormDefault="qualified"
    - This means that all XML elements must be qualified with namespace.
  - targetNamespace=http://www.w3schools.com
    - indicates that all elements defined by this schema come from the http://www.w3schools.com namespace.
  - Xmlns=http://www.w3schools.com
    - Indicates that the default namespace is “http://www.w3schols.com”.



# “Simple” and “complex” elements

- A “simple” element is one that contains text and nothing else
  - A simple element cannot have attributes
  - A simple element cannot contain other elements
  - A simple element cannot be empty
  - However, the text can be of many different types, and may have various restrictions applied to it
- If an element isn’t simple, it’s “complex”
  - A complex element may have attributes
  - A complex element may be empty, or it may contain text, other elements, or both text and other elements



# Defining a simple element

- A simple element is defined as

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

  where:
  - *name* is the name of the element
  - the most common values for *type* are

    xs:boolean xs:integer
    xs:date xs:string
    xs:decimal xs:time
- Other attributes a simple element may have:
  - default="*default value*" *if no other value is specified*
  - fixed="*value*" *no other value may be specified*

# Simple Example

- xs Example

  <xs:element name="lastname" type="xs:string" />

  <xs:element name="age" type="xs:string" />

  <xs:element name="dateborn" type="xs:date" />
- XML Example

  <lastname> Vishnu </lastname>

  <age>34 </age>

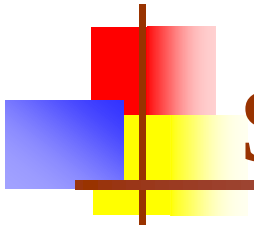
  <dateborn> 1968-03-27 </dateborn>
- Providing Default value Example
  - <xs:element name="color" type=xs:string default="red" />
- Providing Fixed Value Example
  - <xs:element name="color" type="xs:string" fixed="red" />

# Defining an attribute

- Attributes themselves are always declared as simple types
- An attribute is defined as

  <xs:attribute name="*name*" type="*type*" />

  where:
  - *name* and *type* are the same as for xs:element
- Other attributes a simple element may have:
  - default="*default value*" *if no other value is specified*
  - fixed="*value*" *no other value may be specified*
  - use="optional" *the attribute is not required (default)*
  - use="required" *the attribute must be present*



# Simple Example

- xs Example
  - <xs:attribute name="lang" type="xs:string">
- XML Example
  - <lastname lang="EN">Smith</lastname>
- Providing Default Value to the attribute
  - <xs:attribute name="lang" type="xs:string" default="EN" />
- Providing Fixed Value to the attribute
  - <xs:attribute name="lang" type="xs:string" fixed="EN" />

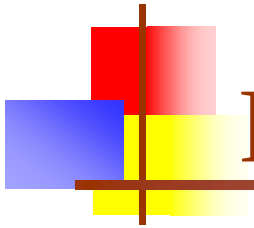
# Restrictions on Content

- With XML Schema , we can also add our own restrictions to our XML elements and attributes. These restrictions are called as **“Facets”.**
- The general form for putting a restriction on an element:
  - <xs:element name="*name*"> *(or* xs:attribute*)*

```
<xs:restriction base="type">
    ... the restrictions ...
</xs:restriction>
</xs:element>
```

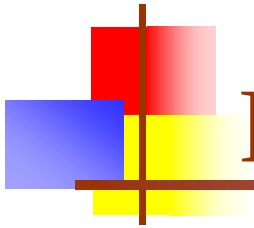
- For example:
  - <xs:element name="age">

```
<xs:restriction base="xs:integer">
    <xs:minInclusive value="0">
    <xs:maxInclusive value="60">
</xs:restriction>
</xs:element>
```



# Restrictions on numbers

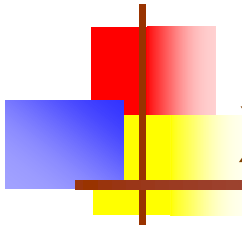
- minInclusive -- number must be ≥ the given *value*
- minExclusive -- number must be > the given *value*
- maxInclusive -- number must be ≤ the given *value*
- maxExclusive -- number must be < the given *value*
- totalDigits -- number must have exactly *value* digits
- fractionDigits -- number must have no more than *value* digits after the decimal point

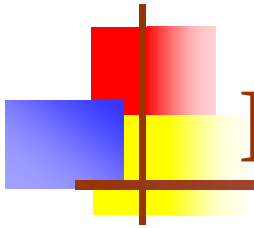


# Restrictions on strings

- length -- the string must contain exactly *value* characters
- minLength -- the string must contain at least *value* characters
- maxLength -- the string must contain no more than *value* characters
- pattern -- the *value* is a regular expression that the string must match
- whiteSpace -- not really a “restriction”--tells what to do with whitespace
  - value="preserve" Keep all whitespace
  - value="replace" Change all whitespace characters to spaces
  - value="collapse" Remove leading and trailing whitespace, and replace all sequences of whitespace with a single space

# Enumeration – Restrictions on set of Values

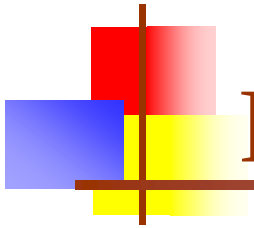
- An enumeration restricts the value to be one of a fixed set of values
- Example:
  - ```
    <xs:element name="season">
        <xs:simpleType>
            <xs:restriction  base="xs:string">
                <xs:enumeration value="Spring"/>
                <xs:enumeration value="Summer"/>
                <xs:enumeration value="Autumn"/>
                <xs:enumeration value="Fall"/>
                <xs:enumeration value="Winter"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:element>
    ```
    ```

# Complex elements

- A complex element is defined as

```
<xs:element   name="name">
    <xs:complexType>
        ... information about the complex type...
    </xs:complexType>
</xs:element>
```

- Example:

```
<xs:element   name="person">
    <xs:complexType>
        <xs:sequence>
            <xs:element  name="firstName"  type="xs:string" />
            <xs:element  name="lastName"  type="xs:string" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

- <xs:sequence> says that elements must occur in this order
- However attributes are always simple types

# Global and local definitions

- Elements declared at the “top level” of a <schema> are available for use throughout the schema
- Elements declared within a xs:complexType are local to that type
- Thus, in

```
<xs:element   name="person">
    <xs:complexType>
        <xs:sequence>
            <xs:element  name="firstName"  type="xs:string" />
            <xs:element  name="lastName"  type="xs:string" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

  the elements firstName and lastName are only locally declared
- The order of declarations at the “top level” of a <schema> *do not* specify the order in the XML data document



# xs:all

- xs:all allows elements to appear in any order
- ```
  <xs:element   name="person">
     <xs:complexType>
        <xs:all>
           <xs:element  name="firstName"  type="xs:string" />
           <xs:element  name="lastName"  type="xs:string" />
        </xs:all>
     </xs:complexType>
  </xs:element>
  ```
- Despite the name, the members of an xs:all group can occur once or not at all
- You can use minOccurs="*n*" and maxOccurs="*n*" to specify how many times an element may occur (default value is 1)
  - In this context, *n* may only be 0 or 1
  ```



# Mixed elements

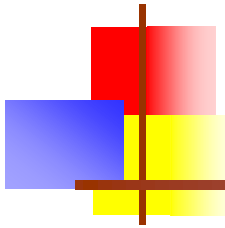
- Mixed elements may contain both text and elements
- We add mixed="true" to the xs:complexType element
- The text itself is not mentioned in the element, and may go anywhere (it is basically ignored)

```
<xs:complexType  name="paragraph"  mixed="true">
    <xs:sequence>
        <xs:element  name="someName"  type="xs:anyType"/>
   </xs:sequence>
</xs:complexType>
```



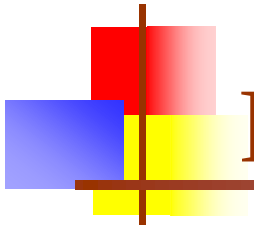
# Predefined string types

- A simple element is defined as:
  <xs:element name="*name*" type="*type*" />
- Here are a few of the possible string types:
  - xs:string -- a string
  - xs:normalizedString -- a string that doesn't contain tabs, newlines, or carriage returns
  - xs:token -- a string that doesn't contain any whitespace other than single spaces
- Allowable restrictions on strings:
  - enumeration, length, maxLength, minLength, pattern, whiteSpace



# Predefined date and time types

- xs:date -- A date in the format *YYYY-MM-DD*, for example, 2002-11-05
- xs:time -- A date in the format *hh:mm:ss* (hours, minutes, seconds)
- xs:dateTime -- Format is *YYYY-MM-DD*T*hh:mm:ss*
- Allowable restrictions on dates and times:
  - enumeration, minInclusive, maxExclusive, maxInclusive, maxExclusive, pattern, whiteSpace



# Predefined numeric types

- Here are some of the predefined numeric types:

xs:decimal
xs:byte
xs:short
xs:int
xs:long
xs:positiveInteger
xs:negativeInteger
xs:nonPositiveInteger
xs:nonNegativeInteger

- Allowable restrictions on numeric types:
  - enumeration, minInclusive, maxExclusive, maxInclusive, maxExclusive, fractionDigits, totalDigits, pattern, whiteSpace

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
                targetNamespace="http://www.books.org"
                xmlns="http://www.books.org"
                elementFormDefault="qualified">
  <xs:element name="BookStore">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Book" minOccurs="1" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Book">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Title" minOccurs="1" maxOccurs="1"/>
        <xs:element ref="Author" minOccurs="1" maxOccurs="1"/>
        <xs:element ref="Date" minOccurs="1" maxOccurs="1"/>
        <xs:element ref="ISBN" minOccurs="1" maxOccurs="1"/>
        <xs:element ref="Publisher" minOccurs="1" maxOccurs="1"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Title" type="xs:string"/>
  <xs:element name="Author" type="xs:string"/>
  <xs:element name="Date" type="xs:string"/>
  <xs:element name="ISBN" type="xs:string"/>
  <xs:element name="Publisher" type="xs:string"/>
</xs:schema>
```