XSL Tutorial 

In [ ]:

```
%%writefile examples/timetable.xml
<?xml version="1.0" ?>
<?xml-stylesheet href="timetable.xsl" type="text/xsl"?>
<timetable>
        <module>
                <name>Web App Dev</name>
                <faculty>
                        <lecturer>FS</lecturer>
                        <labAssist id="1">SC</labAssist>
                        <labAssist id="2">EMcL</labAssist>
                </faculty>
                <classes>
                        <lecture id="1">
                                <day>Mon</day>
                                <time>9am-11am</time>
                                <room type="computerLab">SCR2</room>
                        </lecture>
                        <lecture id="2">
                                <day>Thurs</day>
                                <time>2pm - 4pm</time>
                                <room type="computerLab">SCR1</room>
                        </lecture>
                        <lab>
                                <day>Fri</day>
                                <time>1pm - 3pm</time>
                                <room type="computerLab">SCR2</room>
                        </lab>
                </classes>
        </module>
        <module>
                <name>Intro. to DB</name>
                <faculty>
                        <lecturer>PH</lecturer>
                </faculty>
                <exam/>
                <classes>
```

```
                <lecture>
                        <day>Mon</day>
                        <time>11am-1pm</time>
                        <room type="computerLab">SCR1</room>
                </lecture>
                <lab>
                        <day>Thurs</day>
                        <time>4pm - 5pm</time>
                        <room type="computerLab">SCR1</room>
                </lab>
        </classes>
</module>
<module>
        <name>IT Proj Mgt</name>
        <faculty>
                <lecturer>EO'L</lecturer>
        </faculty>
        <exam/>
        <classes>
                <lecture>
                        <day>Mon</day>
                        <time>3pm - 5pm</time>
                        <room type="computerLab">SCR2</room>
                </lecture>
                <lab>
                        <day>Fri</day>
                        <time>11am - 12pm</time>
                        <room type="computerLab">SCR1</room>
                </lab>
        </classes>
</module>
<module>
        <name>Fund Bus Anal</name>
        <faculty>
                <lecturer>DB</lecturer>
        </faculty>
        <exam/>
        <classes>
```

```
                        <lecture>
                                <day>Tues</day>
                                <time>1pm - 3pm</time>
                                <room type="computerLab">SCR1</room>
                        </lecture>
                        <lab>
                                <day>Thurs</day>
                                <time>12pm - 1pm</time>
                                <room type="classroom">1.01</room>
                        </lab>
                </classes>
        </module>
        <module>
                <name>Software Apps for Bus</name>
                <faculty>
                        <lecturer>JM</lecturer>
                </faculty>
                <exam/>
                <classes>
                        <lecture>
                                <day>Tues</day>
                                <time>3pm - 5pm</time>
                                <room type="computerLab">SCR2</room>
                        </lecture>
                        <lab>
                                <day>Fri</day>
                                <time>9am-11am</time>
                                <room type="classroom">1.02</room>
                        </lab>
                </classes>
        </module>
</timetable>
```

In [ ]:

```
%%writefile examples/timetable.xsl
<?xml version="1.0"?>
<xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">
        <xsl:template match="*">
                <xsl:apply-templates/>
        </xsl:template>
        <xsl:template match="text()">
                <xsl:apply-templates/>
        </xsl:template>
        <xsl:template match="/">
                <html>
                        <head><title>Example</title>
                        </head>
                        <body>
                                <!--Question 1-->
                                <h2>Modules this semester:</h2>
                                <xsl:apply-templates select="/timetable/module/nam
e"/>
                                <!--Question 2-->
                                <h2>Modules with a Terminal Exam:</h2>
                                <xsl:apply-templates select="//exam"/>
                                <!--Question 3-->
                                <h2>Lab Schedule:</h2>
                                <xsl:apply-templates select="/timetable/module/cla
sses/lab"/>
                                <!--Question 4 -->
                                <h2>There is more than one lecture in:</h2>
                                <xsl:apply-templates select="/timetable/module/cla
sses/lecture[@id>'1']"/>
                                <!--Question 5-->
                                <h2>The day time and room of all classes in a comp
uter lab:</h2>
                                <xsl:apply-templates select="//room[@type='compute
rLab']"/>
                                <!--Question 6-->
                                <h2>The details of all Monday classes:</h2>
```

```
                        <xsl:apply-templates select="/timetable/module/classes/*/day[text()='Mon']"/>
                    </body>
                    </html>
        </xsl:template>
        <!--Question 1-->
        <xsl:template match="name">
                    <xsl:value-of select="."/><br/>
        </xsl:template>
        <!--Question 2-->
        <xsl:template match="exam">
                    <xsl:value-of select="../name"/><br/>
        </xsl:template>
        <!--Question 3-->
        <xsl:template match="lab">
                    <xsl:value-of select="../../name"/>
                    <ul>
                            <li><xsl:value-of select="day"/></li>
                            <li><xsl:value-of select="time"/></li>
                            <li><xsl:value-of select="room"/></li>
                    </ul>
        </xsl:template>
        <!--Question 4-->
        <xsl:template match="lecture">
            <xsl:for-each select="../../*">
                <xsl:value-of select="."/><br/>
            </xsl:for-each>
        </xsl:template>
        <!--Question 5-->
        <xsl:template match="room">
                    <xsl:value-of select="../../../name"/>,
                    <xsl:value-of select="../."/><br/>
        </xsl:template>
        <!--Question 6-->
        <xsl:template match="day">
                    <xsl:value-of select="../../../name"/><br/>
        </xsl:template>
</xsl:transform>
```

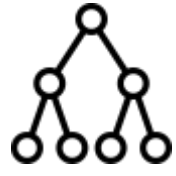
- Run the cell below with `Shift` + `Enter` to render the result

In [ ]:
```
from bs4 import BeautifulSoup
import lxml.etree as ET

dom = ET.parse('examples/timetable.xml')
xslt = ET.parse('examples/timetable.xsl')
transform = ET.XSLT(xslt)
newdom = transform(dom)
result = BeautifulSoup(str(newdom))
html = result.prettify()
print(html)
f=open("examples/timetable.html","w+")
f.write(html)
```

In [ ]:

```
from IPython.display import IFrame
IFrame(src='examples/timetable.html', width="100%", height="800px")
```

XML Schema 

- Press `Space` to navigate through the slides
- Use `Shift+Space` to go back

Introduction to XML Schema

- XML Schema is commonly known as **XML Schema Definition (XSD)**
    - It is used to **describe** and **validate** the **structure** and the **content** of XML data.
    - XML schema defines the **elements**, **attributes** and **data types**.
    - Schema element supports **Namespaces**.
    - It is similar to a database schema that describes the data in a database.

XML Schema

- The **purpose** of an XML Schema is to **define the legal building blocks** of an **XML** document
    - the **elements** and **attributes** that can appear in a document
    - the **number** of (and order of) **child elements**
    - **data types** for **elements** and **attributes**
    - **default** and **fixed values** for elements and attributes

Why Learn XML Schema?

- In the **XML** world, thousands of standardised XML formats are in daily use
- Many of these **XML** standards are defined by **XML Schemas**
- **XML Schema** is essentially a **recepie** for any of your **XML** files
- **XSD** files are written in **XML**, just like **XSL**

XSD Example

```xml
<?xml version="1.0"?>
<lecture id="1">Interactive Web Applications</lecture>
<!-- This is our original XML file -->
```

```xml
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="lecture" type="xs:string"/>
</xs:schema>
<!-- This is out XSD recepie for the XML above -->
```

XSD Example

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="lecture" type="xs:string"/>
</xs:schema>
```

- **XSD** is an **XML** document and starts with an **XML declaration**
- Elements that make up an **XML schema** must belong to the XML Schema **namespace** (http://www.w3.org/2001/XMLSchema (http://www.w3.org/2001/XMLSchema))
- Every schema consists of a single root `xs:schema` element that contains all **type definitions** and **element** and **attribute** declarations
- Elements are declared using the `xs:element` element

XSD Example

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="lecture" type="xs:string"/>
</xs:schema>
```

- The declaration in `xs:element` specifies the name and the type of the element using the name and type attributes.
- The type for the element **lecture** is a `string`
- `xs:string` which represents Unicode strings of characters
- The root of an instance XML document can be any element declared as child of the `xs:schema` element

Referencing XSD

- It is common to associate the instance document explicitly with the schema document
- Suppose we name `lecture.xsd` is the schema document reviewed above
- The instance document changes as follows:

```
<?xml version="1.0"?>
  <lecture xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="lecture.xsd">
  Interactive Web Applications
  </lecture>
```

XSD Components

- **ComplexTypes** and **SimpleTypes**
    - `ComplexType` is a container for other element definitions.
        - Allows you to specify which child elements an element can contain and to provide some structure within your XML documents.
    - `SimpleType` element is used only in the context of the text
        - `string` is an example of a SimpleType
- Elements have **complexTypes** or **simpleTypes**
- Attributes have **simpleTypes**
- **Element declaration** declares the name and the type of an element:
    - `<xs:element name="name" type="type"/>`
- **Attribute declaration** declares the name and the type of an attribute:
    - `<xs:attribute name="name" type="type"/>`

`simpleType`

- A simple type describes text without markup
- A simple type can be assigned either to an **element** or to an **attribute**.
    - `<xs:element name="street" type="xs:string"/>`
    - `<xs:attribute name="price" type="xs:decimal"/>`
- Some typical examples of primitive built-in simple types are:
    - string
    - integer
    - decimal
    - float
    - double
    - boolean
    - time
    - date

`simpleType`

- A **restriction** of an existing type, called based type, defines a new type by restricting the set of possible values of the base type.

Facet	Constraint
length	length of string or number of items of list
minLength	minimal length
maxLength	maximal length
pattern	regular expression pattern
enumeration	enumeration value
whiteSpace	white space normalization
maxInclusive	inclusive upper bound
maxExclusive	exclusive upper bound
minInclusive	inclusive lower bound
minExclusive	exclusive lower bound
totalDigits	maximum number of digits
fractionDigits	maximum number of fractional digits

`simpleType`

For instance, the following type definition uses the `simpleType` element to define the simple type `voteType` to be an integer from 18 to 30:

```
<xs:simpleType name="voteType">
  <xs:restriction base="xs:integer">
    <xs:length value="2" />
    <xs:minInclusive value="18"/>
    <xs:maxInclusive value="30"/>
  </xs:restriction>
</xs:simpleType>
```

`simpleType`

- The following example shows an **enumeration** simple type.
- The type `colorType` is the subset of strings containing `red`, `green` and `blue` values only.

```
<xs:simpleType name="colorType">
<xs:restriction base="xs:string">
  <xs:enumeration value="red"/>
  <xs:enumeration value="green"/>
  <xs:enumeration value="blue"/>
</xs:restriction>
</xs:simpleType>
```

`complexType`

- A complex type describes text that may contain markup in the form of elements and attributes.
- A complex type can be assigned to elements but not to attributes, whose type is always simple.
- An empty element with **no content** or **data type** can be created with `complexType`:

```
<xs:complexType name="empty_element"/>
```

- A slightly more complicated example is that of an element with empty content but having attributes.
- Here is an example:

```
<xs:complexType name="bookType">
    <xs:attribute name="price" type="xs:decimal"/>
  </xs:complexType>
```

- The type `bookType` describes an element with empty content and a single attribute `price` of type decimal.

`complexType` Sequence

- The sequence element defines a sequence of elements that must appear in the instance document in the order they are listed.
- An example follows:

```
<xs:element name="street" type="xs:string"/>
  <xs:element name="number" type="xs:integer"/>
  <xs:element name="city" type="xs:string"/>
  <xs:element name="state" type="xs:string"/>
  <xs:element name="address" type="addressType"/>

  <xs:complexType name="addressType">
    <xs:sequence>
      <xs:element ref="street"/>
      <xs:element ref="number"/>
      <xs:element ref="city"/>
      <xs:element ref="state"/>
    </xs:sequence>
  </xs:complexType>
```

- The element address must contain the subelements street, number, city and state in this order.

`complexType` Sequence

Attribute	Description
id	Optional. Specifies a unique ID for the element
maxOccurs	Optional. Specifies the maximum number of times the sequence element can occur in the parent element. The value can be any number >= 0, or if you want to set no limit on the maximum number, use the value "unbounded". Default value is 1
minOccurs	Optional. Specifies the minimum number of times the sequence element can occur in the parent element. The value can be any number >= 0. Default value is 1
any attributes	Optional. Specifies any other attributes with non-schema namespace

`complexType` Sequence

- This following example shows a declaration for an element `pets` that can have zero or more of the following elements, dog and cat, in the sequence element:

```
<xs:element name="pets">
<xs:complexType>
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element name="dog" type="xs:string"/>
    <xs:element name="cat" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
</xs:element>
```

`complexType` Choice

- The choice element allows to choose one element from a list.
- For instance, the contact element declared below can contain either an email element or a phone element, but not both:

```
<xs:element name="email" type="xs:string"/>
  <xs:element name="phone" type="xs:string"/>
  <xs:element name="contact" type="contactType"/>

  <xs:complexType name="contactType">
    <xs:choice>
      <xs:element ref="email"/>
      <xs:element ref="phone"/>
    </xs:choice>
  </xs:complexType>
```

`complexType` all

- The `all` element defines a set of elements that must appear in the instance document in any order.
- An example follows:

```
<xs:element name="email" type="xs:string"/>
  <xs:element name="phone" type="xs:string"/>
  <xs:element name="contact" type="contactType"/>

  <xs:complexType name="contactType">
    <xs:all>
      <xs:element ref="email"/>
      <xs:element ref="phone"/>
    </xs:all>
  </xs:complexType>
```

- In this case, the contact element must contain both email and phone subelements in any order.
- A few restrictions apply to the all construct: it can only contain element references or declarations.

`complexType` Mixed Content

- A mixed complex type element can contain attributes, elements and text.
- An example follows:

```
<letter>
    Dear Mr. <name>John Smith</name>.
    Your order <orderid>1032</orderid>
    will be shipped on <shipdate>2001-07-13</shipdate>.
  </letter>
```

```
<xs:element name="letter">
    <xs:complexType mixed="true">
      <xs:sequence>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="orderid" type="xs:positiveInteger"/>
        <xs:element name="shipdate" type="xs:date"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

Creating an XML Schema

- Let's have a look at this XML document called `shiporder.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<shiporder orderid="889923" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="shiporder.xsd">
  <orderperson>John Smith</orderperson>
  <shipto>
    <name>Ola Nordmann</name>
    <address>Langgt 23</address>
    <city>4000 Stavanger</city>
    <country>Norway</country>
  </shipto>
  <item>
    <title>Empire Burlesque</title>
    <note>Special Edition</note>
    <quantity>1</quantity>
    <price>10.90</price>
  </item>
  <item>
    <title>Hide your heart</title>
    <quantity>1</quantity>
    <price>9.90</price>
  </item>
</shiporder>
```

- The XML document presented consists of a root element `shiporder` that contains a required attribute called `orderid`.
- The `shiporder` element contains three different child elements:
    - `orderperson`, `shipto` and `item`
- The `item` element appears twice, and it contains a `title`, an optional `note` element, a `quantity`, and a `price` element.

- `xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"` tells the XML parser that this document should be validated against a schema.
- `xsi:noNamespaceSchemaLocation="shiporder.xsd` specifies WHERE the schema resides (here it is in the same folder as `shiporder.xml`).

- To create the schema we could simply follow the structure in the XML document and define each element as we find it.
- We will start with the standard XML declaration followed by the `xs:schema` element that defines a schema:

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

- Next, we have to define the `shiporder` element.
- This element has an attribute and it contains other elements, therefore we consider it as a **complex type**.
- The child elements of the `shiporder` element is surrounded by a xs:sequence element that defines an ordered sequence of sub elements:

```
<xs:element name="shiporder">
  <xs:complexType>
    <xs:sequence>
      ...
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

- Then we have to define the `orderperson` element as a simple type (because it does not contain any attributes or other elements).
- The type (xs:string) is prefixed with the namespace prefix associated with XML Schema that indicates a predefined schema data type:

```
<xs:element name="orderperson" type="xs:string"/>
```

- Next, we have to define two elements that are of the complex type: `shipto` and `item`.
- We start by defining the `shipto` element:

```
<xs:element name="shipto">
<xs:complexType>
  <xs:sequence>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="address" type="xs:string"/>
    <xs:element name="city" type="xs:string"/>
    <xs:element name="country" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
</xs:element>
```

- With schemas we can define the number of possible occurrences for an element with the `maxOccurs` and `minOccurs` attributes.
- `maxOccurs` specifies the maximum number of occurrences for an element and minOccurs specifies the minimum number of occurrences for an element.
- The default value for both `maxOccurs` and `minOccurs` is 1!

- Now we can define the `item` element.
- This element can appear multiple times inside a `shiporder` element.
- This is specified by setting the `maxOccurs` attribute of the `item` element to `unbounded` which means that there can be as many occurrences of the `item` element as the author wishes.
- Notice that the "note" element is optional.
- We have specified this by setting the `minOccurs` attribute to zero:

```
<xs:element name="item" maxOccurs="unbounded">
<xs:complexType>
  <xs:sequence>
    <xs:element name="title" type="xs:string"/>
    <xs:element name="note" type="xs:string" minOccurs="0"/>
    <xs:element name="quantity" type="xs:positiveInteger"/>
    <xs:element name="price" type="xs:decimal"/>
  </xs:sequence>
</xs:complexType>
</xs:element>
```

- We can now declare the attribute of the `shiporder` element.
- Since this is a required attribute we specify `use="required"`

```
<xs:attribute name="orderid" type="xs:string" use="required"/>
```

Here is the complete listing of the schema file called `shiporder.xsd`

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="shiporder">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="orderperson" type="xs:string"/>
      <xs:element name="shipto">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="address" type="xs:string"/>
            <xs:element name="city" type="xs:string"/>
            <xs:element name="country" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="item" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="title" type="xs:string"/>
            <xs:element name="note" type="xs:string" minOccurs="0"/>
            <xs:element name="quantity" type="xs:positiveInteger"/>
            <xs:element name="price" type="xs:decimal"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="orderid" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>
</xs:schema>
```

- The previous design method is very simple, but can be difficult to read and maintain when documents are complex.
- The next design method is based on defining all elements and attributes first, and then referring to them using the **ref** attribute.

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

<!-- definition of simple elements -->
<xs:element name="orderperson" type="xs:string"/>
<xs:element name="name" type="xs:string"/>
<xs:element name="address" type="xs:string"/>
<xs:element name="city" type="xs:string"/>
<xs:element name="country" type="xs:string"/>
<xs:element name="title" type="xs:string"/>
<xs:element name="note" type="xs:string"/>
<xs:element name="quantity" type="xs:positiveInteger"/>
<xs:element name="price" type="xs:decimal"/>

<!-- definition of attributes -->
<xs:attribute name="orderid" type="xs:string"/>

<!-- definition of complex elements -->
<xs:element name="shipto">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="name"/>
      <xs:element ref="address"/>
      <xs:element ref="city"/>
      <xs:element ref="country"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="item">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="title"/>
      <xs:element ref="note" minOccurs="0"/>
      <xs:element ref="quantity"/>
      <xs:element ref="price"/>
    </xs:sequence>
```

In [ ]:

```
%%writefile examples/shiporder.xml
<?xml version="1.0" encoding="UTF-8"?>
<shiporder orderid="889923" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="shiporder.xsd">
  <orderperson>John Smith</orderperson>
  <shipto>
    <name>Ola Nordmann</name>
    <address>Langgt 23</address>
    <city>4000 Stavanger</city>
    <country>Norway</country>
  </shipto>
  <item>
    <title>Empire Burlesque</title>
    <note>Special Edition</note>
    <quantity>1</quantity>
    <price>10.90</price>
  </item>
  <item>
    <title>Hide your heart</title>
    <quantity>1</quantity>
    <price>9.90</price>
  </item>
</shiporder>
```

In [314]:

```
%%writefile examples/shiporder.xsd
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<!-- definition of simple elements -->
<xs:element name="orderperson" type="xs:string"/>
<xs:element name="name" type="xs:string"/>
<xs:element name="address" type="xs:string"/>
<xs:element name="city" type="xs:string"/>
<xs:element name="country" type="xs:string"/>
<xs:element name="title" type="xs:string"/>
<xs:element name="note" type="xs:string"/>
<xs:element name="quantity" type="xs:positiveInteger"/>
<xs:element name="price" type="xs:decimal"/>

<!-- definition of attributes -->
<xs:attribute name="orderid" type="xs:string"/>

<!-- definition of complex elements -->
<xs:element name="shipto">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="name"/>
      <xs:element ref="address"/>
      <xs:element ref="city"/>
      <xs:element ref="country"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="item">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="title"/>
      <xs:element ref="note" minOccurs="0"/>
      <xs:element ref="quantity"/>
```

```
      <xs:element ref="price"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="shiporder">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="orderperson"/>
      <xs:element ref="shipto"/>
      <xs:element ref="item" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute ref="orderid" use="required"/>
  </xs:complexType>
</xs:element>

</xs:schema>
```

```
Overwriting examples/shiporder.xsd
```

Final Validation

In [315]:
```
import lxml.etree as ET

dom = ET.parse('examples/shiporder.xml')
xsd = ET.parse('examples/shiporder.xsd')
xmlschema = ET.XMLSchema(xsd)
print('XML document conforms to its XML Schema?')
print(xmlschema.validate(dom))
```

```
XML document conforms to its XML Schema?
True
```

Summary 

- **XML Schema is very powerful**
  - simple Types and complex Types
  - many pre-defined types
  - many ways to derive and create new types
  - full control and flexibility
  - fully inline with namespaces and other XML standards
- **XML Schema is too powerful?**
  - too complicated\, confusing?
  - difficult to implement
  - people use only a fraction anyway

XSL Tutorial 

- Create an XML file based on the XSD provided:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:simpleType name="numeric">
      <xsd:restriction base="xsd:string">
          <xsd:pattern value="[0--9]" />
      </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="character">
      <xsd:restriction base="xsd:string">
          <xsd:pattern value="[A--Z]" />
      </xsd:restriction>
  </xsd:simpleType>
  <xsd:element name="car" type="character" />
  <xsd:element name="num" type="numeric" />
  <xsd:element name="exercise" type="Texercise" />
  <xsd:complexType name="Texercise">
      <xsd:sequence>
          <xsd:element ref="car" minOccurs="6" maxOccurs="6" />
          <xsd:element ref="num" minOccurs="2" maxOccurs="2" />
          <xsd:element ref="car" />
          <xsd:element ref="num" minOccurs="2" maxOccurs="2" />
          <xsd:element ref="car" />
          <xsd:element ref="num" minOccurs="3" maxOccurs="3" />
          <xsd:element ref="car" />
      </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

- Validate that your XML file conforms to that XSD with the use of online tools:
    - XML-XSD Validator (http://www.freeformatter.com/xml-validator-xsd.html)