EXTENSIBLE MARKUP LANGUAGE 

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

XML TECHNOLOGIES 

- **Presentation**
  - **Cascading Style Sheets (CSS)**
  - **eXtensible Stylesheet Language (XSL)**
  - **XPath**, XQuery
- **Structure**
  - **XML Schema**, RelaxNG, RDF Schema
  - **Document Type Definition (DTD)**
- **Syntax**
  - **XML Namespaces**
  - **XML 1.0**

XML CONSTRUCTS 

- Markup and Content
- XML declaration
- Tags
- Elements
- Attributes
- Entities
- Character data
- Processing instructions
- Comments
- Parser
- DTD

MARKUP AND CONTENT

- The characters making up an XML document are divided into markup and content, which may be distinguished by the application of simple syntactic rules
- Generally, strings that constitute markup either begin with the character < and end with a >, or they begin with the character `&` and end with a `;`

XML DECLARATION 

- XML documents may begin with an XML declaration that describes some information about themselves
- An example is: `<?xml version="1.0" encoding="UTF-8" standalone="yes"?>`

- **encoding**
    - UTF-8 includes encodings for most of the worlds common alphabets
- **standalone**
    - Determines if the document contains external entities such as Document Type Definition (DTD)
    - `standalone="yes"` means that the XML processor must use the DTD for validation only and will not be used for:
        - default values for attributes
        - entity declarations, etc.

TAG 

- A tag is a markup construct that begins with < and ends with >. Tags come in three flavors:
    - start-tag, such as `<section>`
    - end-tag, such as `</section>`
    - empty-element tag, such as `<line-break />`

ELEMENT 

- An element is a logical document component that either begins with a start-tag and ends with a matching end-tag or consists only of an empty-element tag.
- The characters between the start-tag and end-tag, if any, are the element's content, and may contain markup, including other elements, which are called child elements.
- An example is:

```
<greeting>Hello, world!</greeting>
<line-break />
```

- Four different content types:
  - Data content: `<module>Interactive Web Applications</module>`
  - Element content: `<lecturer id='20191234' />`
  - Mixed content: `<text> this is <bold> bold </bold> text </text>`
  - Empty: `<paragraph/>`

ATTRIBUTE 

- An attribute is a markup construct consisting of a name–value pair that exists within a start-tag or empty-element tag.
- An example is `<img src="u2.jpg" alt="U2" />`, where the names of the attributes are "src" and "alt", and their values are "u2.jpg" and "U2" respectively.
- Another example is `<step number="3">Connect A to B.</step>`, where the name of the attribute is "number" and its value is "3".
- An XML attribute can only have a single value and each attribute can appear at most once on each element.
- The order of attributes is insignificant: `<doc type="book" asin="B0093SZ14U">`

ELEMENT VS. ATTRIBUTE 

Element	Attribute
Constituent data	Inherent data
Used for content	Used for meta-data
White space can be ignored or preserved	No further nesting possible (atomic data)
Nesting allowed (child elements)	Default values
Convenient for large values, or binary entities	Minimal datatypes

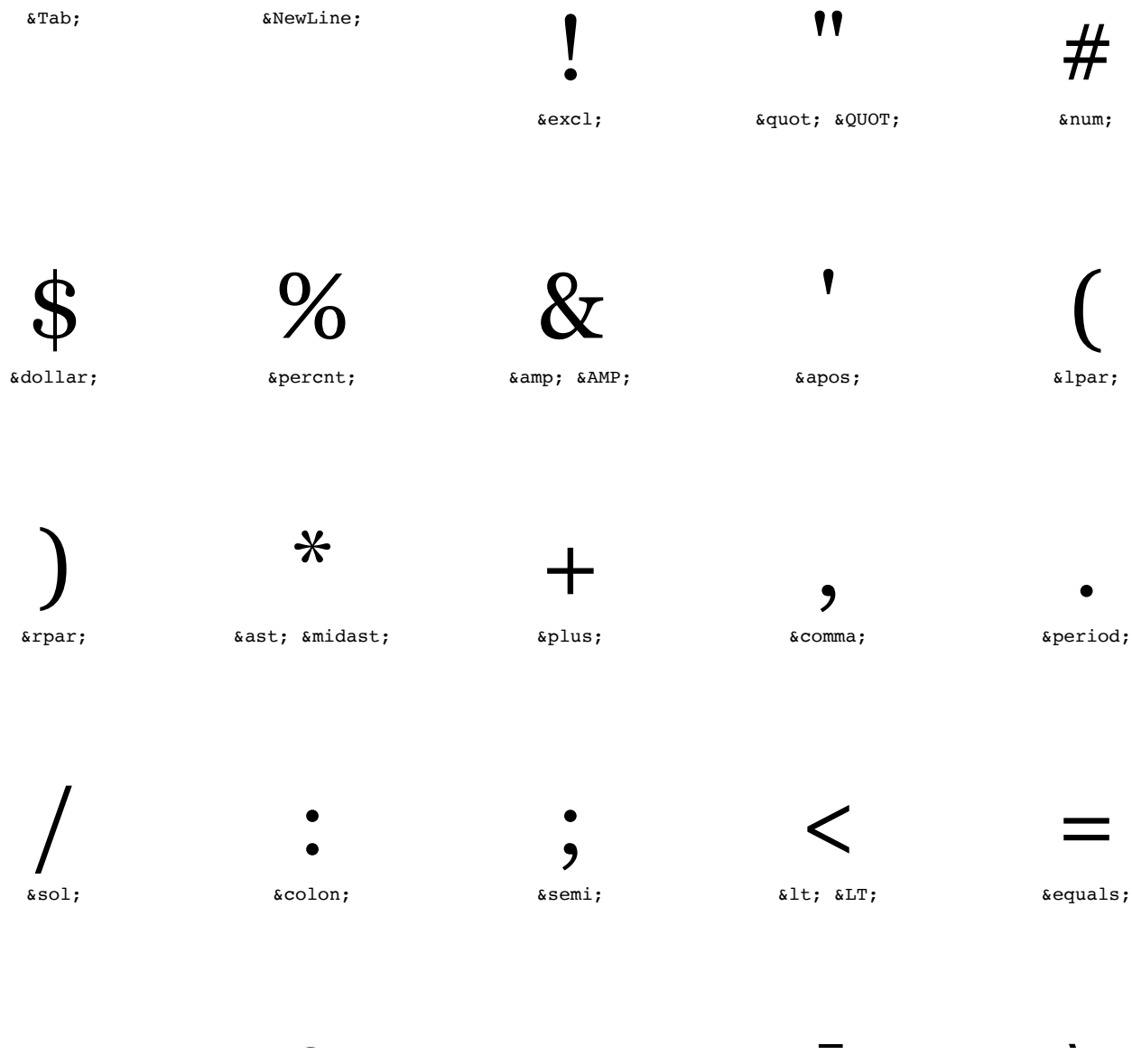
ENTITIES 

- Storage units for repeated text (must be defined in DTD)
- Character entities are used to insert characters that cannot be typed directly
- XML contains a number of 'built-in' entities
- Remember your faviourite **escape characters** used in HTML and JavaScript?

In [72]:

```
from IPython.display import IFrame
IFrame(src='https://dev.w3.org/html5/html-author/charref', width="100%", height="600px")
```

Out[72]:



CDATA 

- Character data is classified as markup and indicates that a certain portion of the document is general character data and is classified as content.
    - Starts with `<![CDATA[` and ends with `]]>`:
      `<![CDATA[<sender>John Smith</sender>]]>`
      is identical to
      `&lt;sender&gt;John Smith&lt;/sender&gt;`
    - In case you don't want to **encapsulate** through the use of **entities**

PROCESSING INSTRUCTIONS 

- Pass additional information to application (e.g. parser)
- Application-specific instructions
- Consists of a PI Target and PI Value
- Processed by applications that recognise the PI Target

```
<?xml-stylesheet type='text/css' href='style.css'?>
<?xml-stylesheet type='text/xsl' href='style.xsl'?>
<?myapp filename='test.txt'?>
```

COMMENTS 

- Used to comment XML documents
- Not considered to be part of an XML document

- An XML parser is not required to pass comments to higher-level applications

```
<!—- one-line comment -->

<!--
 This is a
 multi-line comment
 -->
```

XML PARSER 

- A parser is a piece of program that takes a physical representation of some data and converts it into an in-memory form for the program as a whole to use
- Parsers are used everywhere in software
- An XML Parser is a parser that is designed to read XML and create a way for programs to use XML

DOCUMENT TYPE DECLARATION 

- A DTD defines the valid building blocks of an XML document. It defines the document structure with a list of validated elements and attributes
- A DTD can be declared inline inside an XML document or as an external reference:
    - Internal/Embedded DTD
    - External DTD
- XML Schema superseeded DTD

DTD EXAMPLE 

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE note SYSTEM "Note.dtd">
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

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

WELL-FORMED XML 

- XML Declaration required
- At least one element
    - Exactly one root element
- Empty elements are written in one of two ways:
    - Closing tag `<br></br>`
    - Special start tag `<br />`
- For non-empty elements, closing tags are required
- Start tag must match closing tag (name & case)
- Correct nesting of elements
- Attribute values must always be quoted

WHAT ARE XML NAMESPACES?

- W3C recommendation (January 1999)
- Each XML vocabulary is considered to own a namespace in which all elements (and attributes) are unique
- A single document can use elements and attributes from multiple namespaces
    - A prefix is declared for each namespace used within a document.
    - The namespace is identified using a URI (Uniform Resource Identifier)
- An element or attribute can be associated with a namespace by placing the namespace prefix before its name (i.e. 'prefix:name')
    - Elements (and attributes) belonging to the default namespace do not require a prefix

WHY NAMESPACES?

- Important for creating XML documents containing different types of data
- An XML document can be assembled using elements (and attributes) from different XML vocabularies
- Must be able to:
    - avoid conflicts between names
    - identify the vocabulary an element belongs to

- We implement namespaces by attaching prefixes to elements and attributes:
    - `<product:description></product:description>`
        - `product` is the prefix
        - `description` is the local part
        - `prefix + local part = qualified name`

NAMESPACES EXAMPLE 

```
<root>
    <htm:table xmlns:htm="http://www.crazyfurniture.com/co/html">
        <htm:tr>
            <htm:td>Crazy Furniture Ltd.</htm:td>
            <htm:td>Insanely Expensive</htm:td>
        </htm:tr>
    </htm:table>
    <furn:table xmlns:furn="http://www.crazyfurniture.com/co/furniture">
        <furn:type>Coffee</furn:type>
        <furn:material>Plastic</furn:material>
        <furn:price ccy="EUR">17865.99</furn:price>
    </furn:table>
</root>
```

NAMESPACES EXAMPLE 

Namespaces can reside can be declared in the root element instead:

```
<root xmlns:htm="http://www.crazyfurniture.com/co/html" xmlns:furn="http://www.cr
azyfurniture.com/co/furniture">
    <htm:table xmlns:htm="http://www.crazyfurniture.com/co/html">
        <htm:tr>
            <htm:td>Crazy Furniture Ltd.</htm:td>
            <htm:td>Insanely Expensive</htm:td>
        </htm:tr>
    </htm:table>
    <furn:table xmlns:furn="http://www.crazyfurniture.com/co/furniture">
        <furn:type>Coffee</furn:type>
        <furn:material>Plastic</furn:material>
        <furn:price ccy="EUR">17865.99</furn:price>
    </furn:table>
</root>
```

TUTOTIAL

- Create your own XML file that can store the following information:

```
Jones, Fred
    home: (512) 555-3301
    work: (512) 555-2212
Reynolds, Biff
    home: (512) 555-2222
    Birthday: July 31st
Smith, Bill
    home: (512) 555-2323
    cell: (512) 555-2231
    Contractor
```

- Please assemble your file in the next cell
- The following line **must be** your first line: `%%writefile tut1.xml` in order to create the file
- Put in your XML 1.0 declaration
- Add `<?xml-stylesheet href="tut1.css"?>` as well, as you will be creating styling for your file too
- Add all necessary elements and content
- When finished, press `Shift` + `Enter` to run your code and save `tut1.xml` file
- Go to the next cell (slide) and press `Shift` + `Enter` to see your final file

In [65]:
```
%%writefile tut1.xml

Jones, Fred
    home: (512) 555-3301
    work: (512) 555-2212
Reynolds, Biff
    home: (512) 555-2222
    Birthday: July 31st
Smith, Bill
    home: (512) 555-2323
    cell: (512) 555-2231
    Contractor
```

```
Overwriting tut1.xml
```

YOUR RESULTING XML FILE

In [67]:
```
from IPython.display import IFrame
IFrame(src='tut1.xml', width="100%", height="200px")
```

Out[67]:

This page contains the following errors:

```
error on line 2 at column 1: Document is empty
```

Below is a rendering of the page up to the first error.

- Now let's create CSS for your file to highlight certain elements within your `tut1.xml` file
- Please assemble your file in the next cell (slide)
- The following line **must be** your first line: `%%writefile tut1.css` in order to create the file
- Write your styling based on the elements you created in `tut1.xml` file
- Use `display:block` to make elements appear on separate lines
- When finished, press `Shift` + `Enter` to run your code and save `tut1.xml` file
- Go to the next cell and press `Shift` + `Enter` to see your final file

In [70]:
```
%%writefile tut1.css
// Your CSS code goes in here
```

```
Overwriting tut1.css
```

In [71]:
```
from IPython.display import IFrame
IFrame(src='tut1.css', width="100%", height="200px")
```

Out[71]:
```
// Your CSS code goes in here
```

If you did everything correctly, then run the next slide (cell) with `Shift + Enter` to see the beautiful work you created!

In [61]:
```
from IPython.display import IFrame
IFrame(src='tut1.xml', width="100%", height="600px")
```

Out[61]:

This page contains the following errors:

```
error on line 2 at column 1: Document is empty
```

Below is a rendering of the page up to the first error.