Markup languages:
Adding structure to text

http://www.zeroplayer.com/images/funny-pictures-cat-has-obvious-hat.jpg
What is markup language?

hed: Why the Star needs its own editors
deb: No one else has experience, knowledge and investment in Star's excellence to maintain the "brand," say journalists.

As I mentioned at townhall meetings earlier this fall, the Star's strategic plan calls for a fundamental transformation from a newspaper company into a multi-platform news and content organization. This will involve structuring the organization around the core capabilities that drive the business, and leveraging these core capabilities across new and emerging platforms. Beyond these core areas, we must find the best way to operate our business at the lowest possible cost, including contracting out non-core functions where there is a sound business case to do so. This will involve what is likely to be the biggest restructuring of the Star's workforce in its history. It won't be easy. Changes will affect every job in every corner of the organization.

In light of the magnitude of the changes ahead, we are today launching a Voluntary Separation Program, to provide staff with additional choices. Details of the VSP are provided in the attached letter from Human Resources. Over the last month, we have engaged the CEP in discussions on the provisions of the VSP, and we are very pleased to advise that the final program reflects a signed agreement between the Star and the CEP.

What is markup language?

“A markup language is a system for annotating a text in a way which is syntactically distinguishable from that text.”

http://en.wikipedia.org/wiki/Markup_language
China’s Tough Measures on Flu Appear to Be Effective
By EDWARD WONG
Published: November 11, 2009

Few farmers in this southern Chinese village gave much thought to the swine flu epidemic that had begun spreading rapidly in the United States early this summer until police sealed its 100 residents off from the outside world for about a week. It turned out that a visitor from California had shown symptoms of the swine flu virus, or H1N1, when he arrived for a funeral.
How might we use a markup language?

China’s Tough Measures on Flu Appear to Be Effective
By EDWARD WONG
Published: November 11, 2009

Few farmers in this southern Chinese village gave much thought to the swine flu epidemic that had begun spreading rapidly in the United States early this summer until police sealed its 100 residents off from the outside world for about a week. It turned out that a visitor from California had shown symptoms of the swine flu virus, or H1N1, when he arrived for a funeral.

China’s Tough Measures on Flu Appear to Be Effective
By EDWARD WONG
Published: November 11, 2009

Few farmers in this southern Chinese village gave much thought to the swine flu epidemic that had begun spreading rapidly in the United States early this summer until police sealed its 100 residents off from the outside world for about a week. It turned out that a visitor from California had shown symptoms of the swine flu virus, or H1N1, when he arrived for a funeral.
How might we use a markup language?

China’s Tough Measures on Flu Appear to Be Effective
By EDWARD WONG
Published: November 11, 2009

Few farmers in this southern Chinese village gave much thought to the swine flu epidemic that had begun spreading rapidly in the United States early this summer until police sealed its 100 residents off from the outside world for about a week. It turned out that a visitor from California had shown symptoms of the swine flu virus, or H1N1, when he arrived for a funeral.

China’s Tough Measures on Flu Appear to Be Effective
By EDWARD WONG
Published: November 11, 2009

Few farmers in this southern Chinese village gave much thought to the swine flu epidemic that had begun spreading rapidly in the United States early this summer until police sealed its 100 residents off from the outside world for about a week. It turned out that a visitor from California had shown symptoms of the swine flu virus, or H1N1, when he arrived for a funeral.
China’s Tough Measures on Flu Appear to Be Effective
By EDWARD WONG
Published: November 11, 2009

Few farmers in this southern Chinese village gave much thought to the swine flu epidemic that had begun spreading rapidly in the United States early this summer until police sealed its 100 residents off from the outside world for about a week. It turned out that a visitor from California had shown symptoms of the swine flu virus, or H1N1, when he arrived for a funeral.
How might we use a markup language?

China’s Tough Measures on Flu Appear to Be Effective
By EDWARD WONG
Published: November 11, 2009

Few farmers in this southern Chinese village gave much thought to the swine flu epidemic that had begun spreading rapidly in the United States early this summer until police sealed its 100 residents off from the outside world for about a week. It turned out that a visitor from California had shown symptoms of the swine flu virus, or H1N1, when he arrived for a funeral.
One important family of markup languages are the SGML derivatives:

HTML  XML  DocBook
Few farmers in this southern Chinese village gave much thought to the swine flu epidemic that had begun spreading rapidly in the United States early this summer until police sealed its 100 residents off from the outside world for about a week. It turned out that a visitor from California had shown symptoms of the swine flu virus, or H1N1, when he arrived for a funeral.
HTML is used to control web browsers:

```
<html>
  <head>
    <title>Hello, World!</title>
  </head>
  <body>
    <h1>Hello, World!</h1>
    HTML is nifty.
  </body>
</html>
```
## Important HTML tags:

<table>
<thead>
<tr>
<th>tag</th>
<th>purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>html</td>
<td>Marks beginning and end of document</td>
</tr>
<tr>
<td>head</td>
<td>Contains document metadata</td>
</tr>
<tr>
<td>body</td>
<td>Contains document content</td>
</tr>
<tr>
<td>h1, h2, h3</td>
<td>Headings</td>
</tr>
</tbody>
</table>
Every HTML page needs to have certain elements:

```html
<html>
  <head>
    <title>Hello, World!</title>
  </head>
  <body>
    <h1>Hello, World!</h1>
    HTML is nifty.
  </body>
</html>
```
## Important HTML tags:

<table>
<thead>
<tr>
<th>tag</th>
<th>purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>html</td>
<td>Marks beginning and end of document</td>
</tr>
<tr>
<td>head</td>
<td>Contains document metadata</td>
</tr>
<tr>
<td>body</td>
<td>Contains document content</td>
</tr>
<tr>
<td>h1, h2, h3</td>
<td>Headings</td>
</tr>
<tr>
<td>a</td>
<td>Hyperlinks</td>
</tr>
<tr>
<td>img</td>
<td>Images</td>
</tr>
<tr>
<td>ul/ol</td>
<td>Lists (bulleted &amp; numbered)</td>
</tr>
<tr>
<td>li</td>
<td>List entries</td>
</tr>
</tbody>
</table>
The a tag creates a hyperlink...

```html
<html>
  ...
  <body>
    <h1>Hello, World!</h1>
    <a href="http://www.google.com">HTML is nifty</a>
  </body>
</html>
```
Tags can have “attributes”:

```html
<html>
  ...
  <body>
    <h1>Hello, World!</h1>

    <a href="http://www.google.com">HTML is nifty</a>
  </body>
</html>
```

An attribute has a *name* and a *value*.

name: href
value: http://www.google.com
HTML is used to control web browsers:

```html
<html>
  ...
  <body>
    <h1 align="right">Hello, World!</h1>
    <a href="http://www.google.com">HTML is nifty</a>
  </body>
</html>
```
## Important HTML tags:

<table>
<thead>
<tr>
<th>tag</th>
<th>purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>html</td>
<td>Marks beginning and end of document</td>
</tr>
<tr>
<td>head</td>
<td>Contains document metadata</td>
</tr>
<tr>
<td>body</td>
<td>Contains document content</td>
</tr>
<tr>
<td>h1, h2, h3</td>
<td>Headings</td>
</tr>
<tr>
<td>a</td>
<td>Hyperlinks</td>
</tr>
<tr>
<td>img</td>
<td>Images</td>
</tr>
<tr>
<td>ul/ol</td>
<td>Lists (bulleted &amp; numbered)</td>
</tr>
<tr>
<td>li</td>
<td>List entries</td>
</tr>
</tbody>
</table>
The `img` tag can insert images...

```html
<img src="monkey-typing.jpg"/>
```

Hello, World!

HTML is nifty.
The `<ul>`, `<ol>`, and `<li>` tags do lists:

```html
<ul>
  <li>Monkeys are neat.</li>
  <li>I like monkeys.</li>
</ul>

<ol>
  <li>Find monkeys.</li>
  <li>...</li>
  <li>Profit!</li>
</ol>
```
We could fill an entire class with nothing but HTML, and still miss stuff...

... but that’s enough of the basics to get started.
HTML is about describing the presentation and behavior of documents...

... what about XML?
XML is about representing the *structure* and *semantics* of a document or data set.

This data has no semantics...

Name?

Email?

WTF?

<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
<th>Email</th>
<th>Abilites</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>1.5</td>
<td><a href="mailto:alice@ohsu.edu">alice@ohsu.edu</a></td>
<td>AB</td>
</tr>
<tr>
<td>Bob</td>
<td>1.8</td>
<td><a href="mailto:bob@ohsu.edu">bob@ohsu.edu</a></td>
<td>O</td>
</tr>
<tr>
<td>Carol</td>
<td>1.4</td>
<td><a href="mailto:carol@ohsu.edu">carol@ohsu.edu</a></td>
<td>AB</td>
</tr>
</tbody>
</table>
XML is about representing the structure and semantics of a document or data set.

```xml
<?xml version="1.0" encoding="utf-8"?>
<donors>
  <person>
    <name>Alice</name>
    <height>1.5</height>
    <email>alice@ohsu.edu</email>
    <bloodType>AB</bloodType>
  </person>
  <person>
    <name>Bob</name>
    ...
  </person>
  ...
</donors>
```

Much better!
Besides us not knowing a data set’s semantics, what else could go wrong with this file?

<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
<th>Email</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>1.5</td>
<td><a href="mailto:alice@ohsu.edu">alice@ohsu.edu</a></td>
<td>AB</td>
</tr>
<tr>
<td>Bob</td>
<td>1.8</td>
<td><a href="mailto:_bob@ohsu.edu">_bob@ohsu.edu</a></td>
<td>O</td>
</tr>
<tr>
<td>Carol</td>
<td>1.4</td>
<td><a href="mailto:carol@ohsu.edu">carol@ohsu.edu</a></td>
<td>AB</td>
</tr>
</tbody>
</table>
Even if we know how the data is *supposed* to be laid out in the file...

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>1.5</td>
</tr>
<tr>
<td>Bob</td>
<td>1.8</td>
</tr>
<tr>
<td>Carol</td>
<td>1.4</td>
</tr>
</tbody>
</table>

...there's no guarantee that it *is* laid out that way...

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>1.5</td>
</tr>
<tr>
<td>Bob</td>
<td>0</td>
</tr>
<tr>
<td>Carol</td>
<td>1.4</td>
</tr>
</tbody>
</table>
Even if we know how the data is *supposed* to be laid out in the file...

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Email</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>1.5</td>
<td><a href="mailto:alice@ohsu.edu">alice@ohsu.edu</a></td>
<td>AB</td>
</tr>
<tr>
<td>Bob</td>
<td>1.8</td>
<td><a href="mailto:bob@ohsu.edu">bob@ohsu.edu</a></td>
<td>O</td>
</tr>
<tr>
<td>Carol</td>
<td>1.4</td>
<td><a href="mailto:carol@ohsu.edu">carol@ohsu.edu</a></td>
<td>AB</td>
</tr>
</tbody>
</table>

Alice, 1.5, *alice@ohsu.edu*, AB
Bob, 1.8, *bob@ohsu.edu*, O
Carol, 1.4, *carol@ohsu.edu*, AB

Alice, 1.5, *alice@ohsu.edu*, AB
Bob, *monkeys*, *bob@ohsu.edu*, O
Carol, 1.4, *carol@ohsu.edu*, Z

... or whether it's contents are valid.
Even if we somehow are sure that the data is laid out correctly...

<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
<th>Email</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>1.5</td>
<td><a href="mailto:alice@ohsu.edu">alice@ohsu.edu</a></td>
<td>AB</td>
</tr>
<tr>
<td>Bob</td>
<td>1.8</td>
<td><a href="mailto:bob@ohsu.edu">bob@ohsu.edu</a></td>
<td>O</td>
</tr>
<tr>
<td>Carol</td>
<td>1.4</td>
<td><a href="mailto:carol@ohsu.edu">carol@ohsu.edu</a></td>
<td>AB</td>
</tr>
</tbody>
</table>

... all sorts of character encoding problems could pop up!
Finally, even if we get the encoding right...

Alice, 1.5, alice@ohsu.edu, AB
Bob, 1.8, bob@ohsu.edu, O
Carol, 1.4, carol@ohsu.edu, AB

... some of the data could be missing, and we’d have no way of knowing.
Finally, even if we get the encoding right...

Alice, 1.5, alice@ohsu.edu, AB
Bob, 1.8, bob@ohsu.edu, O
Carol, 1.4, carol@ohsu.edu, AB

... some of the data could be missing, and we’d have no way of knowing.
XML addresses each of these problems.

1. Incomplete data file
2. Malformed data
3. Encoding issues
An XML document has two different levels of validity:

1. “Well-formed”
2. “Valid” according to some specification
An XML document that is not “well-formed” cannot be parsed.

Rule #1: All tags must close, or be self-closing.
An XML document that is not well-formed cannot be parsed.

Rule #2: Tags must close in a last-in, first-out fashion.

This is also called “proper nesting”.
An XML document that is not well-formed cannot be parsed.

<foo>Monkeys are <bar>neat</bar>, said Steve.</foo>

**Rule #3: The document must contain a single root node.**

<foo>Monkeys are <bar>neat</bar>, said Steve.</foo><foo>I certainly think so, said Kyle.</foo>
An XML document that is not well-formed cannot be parsed.

**Rule #3: The document must contain a single root node.**

```
<foo>Monkeys are <bar>neat</bar>, said Steve.</foo>
```

```
<foo>Monkeys are <bar>neat</bar>, said Steve.</foo>
```

```
foo
  ├── "monkeys are"
  │    └── "neat"
  └── bar
      └── "neat"
  , said Steve.
```
An XML document that is not well-formed cannot be parsed.

**Rule #3: The document must contain a single root node.**
An XML document that is not well-formed cannot be parsed.

Rule #3: The document must contain a single root node.

```xml
<comment>
  <foo>
    Monkeys are <bar>neat</bar>, said Steve.
  </foo>
  <foo>
    I certainly think so, said Kyle.
  </foo>
</comment>
```
An XML document that is not well-formed cannot be parsed.

**Rule #3: The document must contain a single root node.**

```
<foo>Monkeys are <bar>neat</bar>, said Steve.</foo>
```
An XML document that is not well-formed cannot be parsed.

<foo>Monkeys are <bar>neat</bar>, said Steve</foo>

**Rule #4: Tags are case-sensitive, and so start and close tags must match case.**

<Foo>monkeys are <bar>neat</bar>, don’t you think?</foo>
An XML document that is not well-formed cannot be parsed.

Rule #5: No reserved characters may be present in text, element names, or element attributes: <, >, &, ‘, “

<foo publisher="O’Reilly">Did you & Kyle say that there were < 5 monkeys in the room?</foo>

<foo publisher="O&apos;Reilly">Did you &amp; Kyle say that there were &lt; 5 monkeys in the room</foo>
An XML document that is not well-formed cannot be parsed.

**Rule #5: No reserved characters may be present:** `<`, `>`, `&`, `'`, `“`

<table>
<thead>
<tr>
<th>Character</th>
<th>Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&amp;</code></td>
<td><code>&amp;amp;</code></td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td><code>&amp;lt;</code></td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td><code>&amp;gt;</code></td>
</tr>
<tr>
<td><code>'</code></td>
<td><code>&amp;apos;</code></td>
</tr>
<tr>
<td><code>“</code></td>
<td><code>&amp;quot;</code></td>
</tr>
</tbody>
</table>

Predefined “entities”
XML addresses each of these problems.

1. Incomplete data file
2. Malformed data
3. Encoding issues

An incomplete file will be malformed, and therefore can’t be parsed!
In addition to being well-formed, an XML document may be “valid”.
What does “valid” mean?

A “valid” document might contain only certain elements...

... or certain combinations of elements...

... or even elements containing only certain kinds of data (number, string, etc.)
Exactly how to do this is beyond the scope of this lecture, but two words you should know are:

**DTD (Document Type Definition)**

Used to describe which tags may exist in an XML document, and how they may be nested within each other.
Exactly how to do this is beyond the scope of this lecture, but two words you should know are:

*Schema* (a.k.a. *XML Schema, W3C Schema, RELAX-NG*, etc.)

Much more powerful than DTD- a schema describes not only what elements may appear, but also what may go inside them.
DTD Example:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE people_list SYSTEM "example.dtd">
<people_list>
  <person>
    <name>Fred Bloggs</name>
    <birthdate>27/11/2008</birthdate>
    <gender>Male</gender>
  </person>
</people_list>
```

http://en.wikipedia.org/wiki/Document_Type_Definition
Schema Example:

```xml
<?xml version="1.0" encoding="utf-8"?>
<Address xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="SimpleAddress.xsd">
  <Recipient>Mr. Walter C. Brown</Recipient>
  <House>49</House>
  <Street>Featherstone Street</Street>
  <Town>LONDON</Town>
  <PostCode>EC1Y 8SY</PostCode>
  <Country>UK</Country>
</Address>
```

Using a DTD or a schema, you can automatically ensure that an XML document contains what you expect it to contain.

More code that you don’t have to write!
XML addresses each of these problems.

✓ 1. Incomplete data file
✓ 2. Malformed data
✓ 3. Encoding issues

An invalid file won’t be parsed!
Encoding issues:

“Thou shalt use Unicode.”

<?xml version="1.0" encoding="utf-8"?>

The character 中 is hard to type.

The character \&#x4e2d; is hard to type.
XML addresses each of these problems.

✓ 1. Incomplete data file
✓ 2. Malformed data
✓ 3. Encoding issues

Unicode characters can be represented easily in XML, and it’s possible to embed a file’s encoding within the file!
XML does have shortcomings...

1. Verbosity
2. Harder to parse

```python
for r in open('myfile.txt'):
    fields = r.split(',
    name = fields[0]
    print name
```
Parsing XML is usually a matter of accessing the document’s tree:

```
<comment>
  <foo>
    Monkeys are neat, said Steve.
  </foo>
  <foo>
    I certainly think so, said Kyle.
  </foo>
</comment>
```
Parsing XML is usually a matter of accessing the document’s tree:

```
<comment>
  <foo>
    Monkeys are <bar>neat</bar>, said Steve.
  </foo>
  <foo>
    I certainly think so, said Kyle.
  </foo>
</comment>
```
Parsing XML is usually a matter of accessing the document’s tree:

```
<comment>
  <foo>
    Monkeys are <bar>neat</bar>, said Steve.
  </foo>
  <foo>
    I certainly think so, said Kyle.
  </foo>
</comment>
```
Parsing XML is usually a matter of accessing a document’s tree:

```xml
<comment>
  <foo>
    Monkeys are neat, said Steve.
  </foo>
  <foo>
    I certainly think so, said Kyle.
  </foo>
</comment>
```
There are many ways to parse XML...
Python provides a wide range of tools and libraries that implement these approaches.
<?xml version="1.0" encoding="utf-8"?>
<donors>
  <person type="student">
    <name>Alice</name>
    <height>1.5</height>
    <email>alice@ohsu.edu</email>
    <bloodType>AB</bloodType>
  </person>
  <person type="instructor">
    <name>Bob</name>
    ...
  </person>
  ...
</donors>
The simplest is with minidom- basic DOM parsing:

```python
from xml.dom.minidom import parse

x = parse('blood_bank.xml')

people = x.getElementsByTagName('person')

print "there are %d people." % len(people)

for p in people:
    name = p.getElementsByTagName('name')[0].firstChild.nodeValue
    blood_type = p.getElementsByTagName('bloodType')[0].firstChild.nodeValue
    person_type = p.getAttribute('type')
    print "%s has blood type %s and is of type: %s" % (name, blood_type, person_type)
```

there are 3 people.
Alice has blood type AB and is of type: student
Bob has blood type O and is of type: instructor
Carol has blood type AB and is of type: student
That's an awful lot of work... there are easier ways.

My favorite is XPath.
An XPath expression describes a pattern, and finds all matching nodes.

/donors/person/name

<?xml version="1.0" encoding="utf-8"?>
<donors>
<person type="student">
  <name>Alice</name>
  <height>1.5</height>
  <email>alice@ohsu.edu</email>
  <bloodType>AB</bloodType>
</person>
<person type="instructor">
  <name>Bob</name>
  ...
</person>
...</donors>
An XPath expression describes a pattern, and finds all matching nodes.

/donors/person[@type='student']/name

<?xml version="1.0" encoding="utf-8"?>
<donors>
  <person type="student">
    <name>Alice</name>
    <height>1.5</height>
    <email>alice@ohsu.edu</email>
    <bloodType>AB</bloodType>
  </person>
  <person type="instructor">
    <name>Bob</name>
    ...
  </person>
  ...
</donors>
Python has several ways to do XPath, but ElementTree is easy and included:

```python
import xml.etree.ElementTree as ET

d = ET.parse('blood_bank.xml')

names = d.findall("person/name")
for n in names:
    print n

people = d.findall('person')

for p in people:
    blood_type = p.find('bloodType').text
    name = p.find("name").text
    person_type = p.attrib['type']
    print "%s has blood type %s and is of type: %s" % (name, blood_type, person_type)
```
What about *generating* XML?

Here’s what *NOT* to do:

```python
out = open('myfile.xml','w')
names = ['Kyle','Aaron','Steven']
out.write("<instructors>
"
for n in names:
    out.write("<name>" + n + "</name>\n")
out.write("</instructors>\n")
```

```xml
<instructors>
    <name>Kyle</name>
    <name>Aaron</name>
    <name>Steven</name>
</instructors>
```
What about generating XML?

Why not?

```python
out = open('myfile.xml','w')

names = ['Ben & Jerry\'s','O\'Reilly','\<3 Monkeys']

out.write('<instructors>
')
for n in names:
    out.write("<name>" + n + "</name>\n")

out.write('</instructors>
')
```

```
<instructors>
  <name>Ben & Jerry\'s</name>
  <name>O\'Reilly</name>
  <name><3 Monkeys</name>
</instructors>
```

Not well-formed XML!
Instead, use one of Python’s many XML generation libraries:

```python
import xml.etree.ElementTree as ET
root = ET.Element("instructors")

for n in names:
    a = ET.SubElement(root, "name")
    a.text = n

tree = ET.ElementTree(root)
tree.write("myfile.xml")
```

```
<instructors>
    <name>Ben &amp; Jerry's</name>
    <name>O'Reilly</name>
    <name>&lt;3 Monkeys</name>
</instructors>
```
You can generate HTML this way, too!

```python
import xml.etree.ElementTree as ET

root = ET.Element("html")

head = ET.SubElement(root, "head")
title = ET.SubElement(head, "title")
title.text = "Some nifty page"

body = ET.SubElement(root, "body")
body.text = "body goes here"

p = ET.SubElement(body, "p")

link = ET.SubElement(body, "a")
link.attrib["href"] = "http://www.google.com"
link.text = "Google"

tree = ET.ElementTree(root)
tree.write("foo.html")
```
You can generate HTML this way, too!

```html
<html>
  <head>
    <title>Some nifty page</title>
  </head>
  <body>
    body goes here
    <p />
    <a href="http://www.google.com">Google</a>
  </body>
</html>
```
XMLBuilder is less work to use, but does not come with Python.

```python
from xmlbuilder import XMLBuilder
x = XMLBuilder(format=True)

with x.html:
    with x.head:
        with x.title:
            x << "Some nifty page..."
    with x.body:
        x << "body goes here"
        x << ('p',{})
        x << ('a', 'Google', {'href':'http://www.google.com'})

print unicode(x)
```
More interesting example (ElementTree):

```python
import xml.etree.ElementTree as ET

root = ET.Element("html")

head = ET.SubElement(root, "head")
title = ET.SubElement(head, "title")
title.text = "Some nifty page"

body = ET.SubElement(root, "body")

list = ET.SubElement(body, "ul")

names = ["Steven","Kyle","Aaron"]

for n in names:
    i = ET.SubElement(list,"li")
    i.text = n

tree = ET.ElementTree(root)

tree.write("foo.html")
```

```html
<html>
    <head>
        <title>
            Some nifty page
        </title>
    </head>
    <body>
        <ul>
            <li>Steven</li>
            <li>Kyle</li>
            <li>Aaron</li>
        </ul>
    </body>
</html>
```
More interesting example (XMLBuilder):

```python
from xmlbuilder import XMLBuilder
x = XMLBuilder(format=True)
names = ['Steven', 'Kyle', 'Aaron']

with x.html:
    with x.head:
        with x.title:
            x << 'Some nifty page...
    with x.body:
        with x.ul:
            for n in names:
                with x.li:
                    x << n

print unicode(x)
```

```html
<html>
  <head>
    <title>
      Some nifty page
    </title>
  </head>
  <body>
    <ul>
      <li>Steven</li>
      <li>Kyle</li>
      <li>Aaron</li>
    </ul>
  </body>
</html>
```
Installing XMLBuilder:

1. Install Python’s easy_install tool
2. easy_install xmlbuilder

On a Mac or Linux machine:
http://pypi.python.org/pypi/setuptools#cygwin-mac-os-x-linux-other

Windows:

http://stackoverflow.com/questions/309412/how-to-setup-setuptools-for-python-2-6-on-windows
http://showmedo.com/videotutorials/video?name=2070000&fromSeriesID=207