EMERITUS
INSTITUTE OF MANAGEMENT

WEEK 7

DATA EXTRACTION - GETTING DATA FROM THE INTERNET- PART 1

MITSloan Executive Education

Columbia Business School
AT THE VERY CENTER OF BUSINESS™
EXECUTIVE EDUCATION

Tuck
EXECUTIVE EDUCATION
at Dartmouth

Getting Data: Part 1

The problem!

Data exists in different sources and different formats and we have to work with whatever format we get

or

We go to data analysis with data in the format we have, not the format we want!

Sources of data

local data

- csv files
- pdf files
- xls files

web data

- json
- xml
- html

database servers

- mysql
- postgres
- mongoDB

Getting Data: Part 1

RESTful Web Services

REST: Representational State Transfer

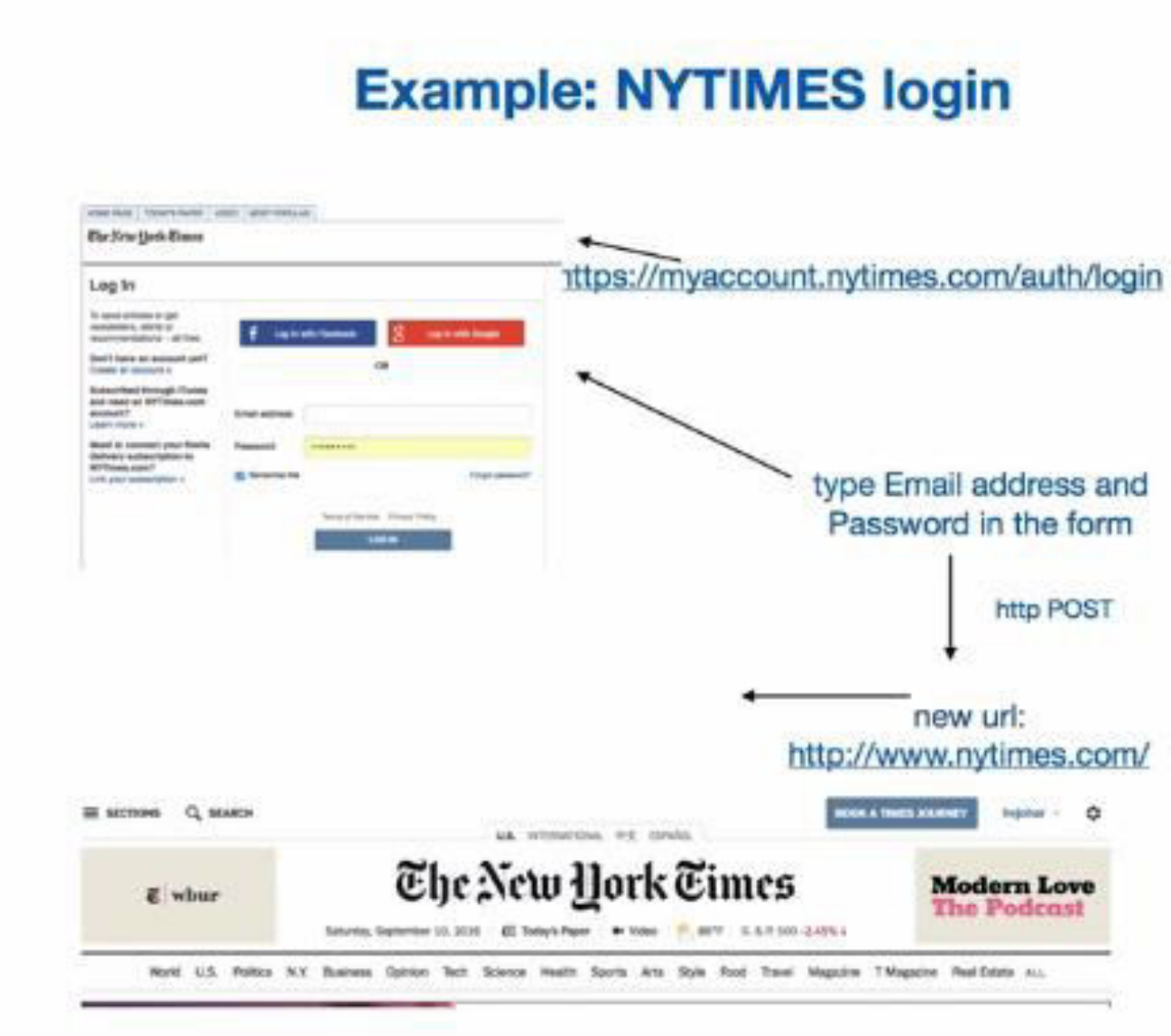
"A network of web pages connected through links and HTTP commands (GET, POST, etc.)"

RESTful: A web service that conforms to the REST standards

URLs: RESTful Web Services deliver resources to the client. Each resources (html, json, image, etc.) is associated with a URL and an HTTP method

RESTful: A web service that conforms to the REST standards

Getting Data: Part 1



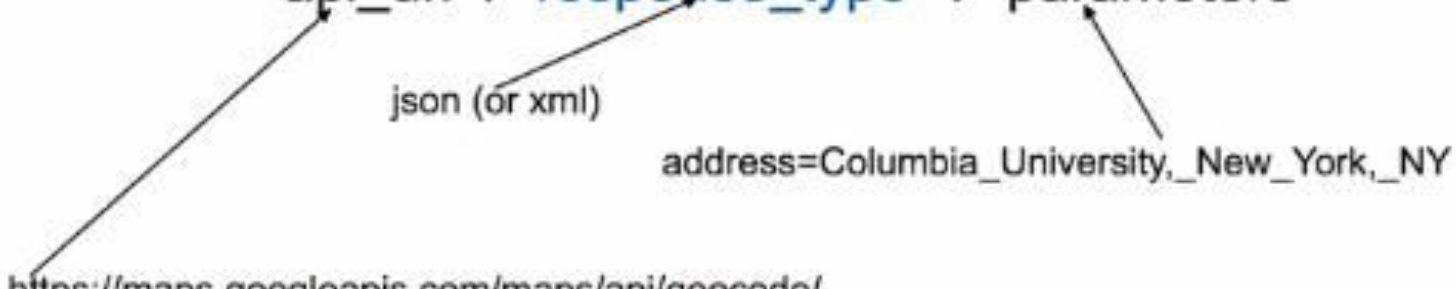
Getting Data: Part 1

Example: Google GEOCODING API

HTTP GET request
with a JSON
response

https://maps.googleapis.com/maps/api/geocode/json?address=Columbia_University,_New_York,_NY

All google API requests take the form:
<api_url>/<response_type>?<parameters>



Getting Data: Part 2 (Pending)

To get data programmatically, you require:

The ability to

* create and send HTTP requests
* receive and process HTTP responses
* convert data residing in JSON/XML/ HTML format into python objects

Getting Data: Part 2

Python libraries for getting web data

* Send an http request and get an http response
  * requests
  * urllib.requests (urllib2 on python2)
* parse the response and extract data
  * json
  * lxml
  * BeautifulSoup, Selenium (for html data)

http requests

requests: Python library for handling http requests and responses

http://docs.python-requests.org/en/master/

using requests

* Import the library
  ```
  import requests
  ```
* Construct the url
  ```
  url = “http://www.epicurious.com/search/Tofu+Chili”
  ```
* Send the request and get a response
  ```
  response = requests.get(url)
  ```
* Check if the request was successful
  ```
  if response.status_code == 200:
  “SUCCESS”!!!!
  else:
  “FAILURE”!!!
  ```

Getting Data: Part 2

Using *request* example

Step 1: Import the requests library

```
In [ ]: import requests
```

Step 2: Send an HTTP request, get the response, and save in a variable

```
In [ ]: response = requests.get("http://www.epicurious.com/search/Tofu+Chili")
```

Step 3: Check the response status code to see if everything went as planned

- status code 200: the request response cycle was successful
- any other status code: it didn't work (e.g., 404 = page not found)

```
In [ ]: print(response.status_code)
```

Step 4: Get the content of the response

- Convert to utf-8 if necessary

```
In [ ]: response.content.decode('utf-8')
```

Problem: Get the contents of Wikipedia's main page and look for the string "Did you know" in it

```
In [10]: url = "https://en.wikipedia.org/wiki/main_page"
         #The rest of your code should go below this line
```

Getting Data: Part 2

Using *request* example

Step 4: Get the content of the response

- Convert to utf-8 if necessary

```
In [10]: response.content.decode('utf-8')
```

```
Out[10]: '<!doctype html>\n<html>\n  <head><meta charset="utf-8">\n<meta name="apple-itunes-app" content="app-id=312101965" />
\n<title>Search | Epicurious.com</title>\n<link rel="dns-prefetch" href="//www.epicurious.com">\n<link rel="dns-prefe
tch" href="//assets.adobedtm.com">\n<link rel="dns-prefetch" href="//www.google-analytics.com">\n<link rel="dns-prefe
tch" href="//tpc.googlesyndication.com">\n<link rel="dns-prefetch" href="//static.parsely.com">\n<link rel="dns-prefe
tch" href="//cdn.optimizely.com">\n<link rel="dns-prefetch" href="//condenast.demdex.net">\n<link rel="dns-prefetch"
href="//capture.condenastdigital.com">\n<link rel="dns-prefetch" href="//pixel.condenastdigital.com">\n<link rel="dns
-prefetch" href="//use.typekit.net">\n<link rel="dns-prefetch" href="//fonts.typekit.net">\n<link rel="dns-prefetch"
href="//p.typekit.net">\n<link rel="dns-prefetch" href="//assets.epicurious.com">\n<link rel="dns-prefetch" href="//a
d.doubleclick.net">\n<link rel="dns-prefetch" href="//pagead2.googlesyndication.com">\n<link rel="dns-prefetch" href=
"//z.moatads.com">\n\n<meta content="en_US" property="og:locale" itemprop="inLanguage" />\n<meta http-equiv="x-ua-com
patible" content="IE=edge" />\n\n<meta http-equiv="cache-control" content="no-cache" />\n<meta http-equiv="pragma" co
ntent="no-cache" />\n\n<meta itemprop="name" content="Search | Epicurious.com" />\n<meta itemprop="logo" content="htt
p://www.epicurious.com/static/img/misc/epicurious-social-logo.png" />\n\n<meta name="description" content="Easily sea
rch and browse more than 37,000 recipes, articles, galleries, menus, and videos from Epicurious.com, Bon Appétit, and
```


```
In [10]: response.content.decode('utf-8')
```


```
Out[10]: '<!doctype html>\n<html>\n  <head><meta charset="utf-8">\n<meta name="apple-itunes-app" content="app-id=312101965" />
\n<title>Search | Epicurious.com</title>\n<link rel="dns-prefetch" href="//www.epicurious.com">\n<link rel="dns-prefe
tch" href="//assets.adobedtm.com">\n<link rel="dns-prefetch" href="//www.google-analytics.com">\n<link rel="dns-prefe
tch" href="//tpc.googlesyndication.com">\n<link rel="dns-prefetch" href="//static.parsely.com">\n<link rel="dns-prefe
tch" href="//cdn.optimizely.com">\n<link rel="dns-prefetch" href="//condenast.demdex.net">\n<link rel="dns-prefetch"
href="//capture.condenastdigital.com">\n<link rel="dns-prefetch" href="//pixel.condenastdigital.com">\n<link rel="dns
-prefetch" href="//use.typekit.net">\n<link rel="dns-prefetch" href="//fonts.typekit.net">\n<link rel="dns-prefetch"
href="//p.typekit.net">\n<link rel="dns-prefetch" href="//assets.epicurious.com">\n<link rel="dns-prefetch" href="//a
d.doubleclick.net">\n<link rel="dns-prefetch" href="//pagead2.googlesyndication.com">\n<link rel="dns-prefetch" href=
"//z.moatads.com">\n\n<meta content="en_US" property="og:locale" itemprop="inLanguage" />\n<meta http-equiv="x-ua-com
patible" content="IE=edge" />\n\n<meta http-equiv="cache-control" content="no-cache" />\n<meta http-equiv="pragma" co
ntent="no-cache" />\n\n<meta itemprop="name" content="Search | Epicurious.com" />\n<meta itemprop="logo" content="htt
p://www.epicurious.com/static/img/misc/epicurious-social-logo.png" />\n\n<meta name="description" content="Easily sea
rch and browse more than 37,000 recipes, articles, galleries, menus, and videos from Epicurious.com, Bon Appétit, and
other partners." />\n    <meta itemprop="author" content="Epicurious" />\n\n\n<link rel="canonical" href="http://www.
epicurious.com/search/Tofu%2BChili" />\n\n<meta name="copyright" content="Copyright (c) 2017 Conde Nast" />\n<meta na
me="p:domain_verify" content="9c2002da922784afad64b638161c75f7" />\n\n\n<meta property="og:title" content="Search | E
picurious.com" />\n<meta property="og:type" content="website" />\n<meta property="og:url" content="http://www.epicuri
ous.com/search/Tofu%2BChili" />\n<meta property="og:description" content="Easily search and browse more than 37,000 r
```

Web Data Format

web data formats

* HTML
  the common format when scraping web pages for data

* JSON or XML
  usually when accessing data through an API or when the server is explicitly sharing data with you

Web Data Format

JSON

JavaScript Object Notation

- Standard for "serializing" data objects for storage or transmission
- Human-readable, useful for data interchange
- Also useful for representing and storing semistructured data
- Stored as plain (byte strings or utf-8 strings) text

JSON constructs and Python equivalents

JSON	Python
number	int,float
string	str
Null	None
true/false	True/False
Object	dict
Array	list

python json library

json.loads(<str>): converts a JSON string to python objects

json.dumps(<python_object>): converts a python object into a JSON formatted string

Web Data Format

Python Data List

Json converts entire data string to equivalent python object

JSON

- The python library - json - deals with converting text to and from JSON

```
In [16]: import json
         data_string = '[{"b": [2, 4], "c": 3.0, "a": "A"}]'
         python_data = json.loads(data_string)
         print(python_data)
```

```
[{'b': [2, 4], 'c': 3.0, 'a': 'A'}]
```

```
In [18]: type(python_data)
```

```
Out[18]: list
```

json.loads recursively decodes a string in JSON format into equivalent python objects

- data_string's outermost element is converted into a python list
- the first element of that list is converted into a dictionary
- the key of that dictionary is converted into a string
- the value of that dictionary is converted into a list of two integer elements

```
In [21]: print(type(data_string),type(python_data))
         print(type(python_data[0]),python_data[0])
         print(type(python_data[0]['b']),python_data[0]['b'])
         print((python_data[0]['a']))
```

```
<class 'str'> <class 'list'>
<class 'dict'> {'b': [2, 4], 'c': 3.0, 'a': 'A'}
<class 'list'> [2, 4]
A
```

JSON, Google API: Part 1

requests and json

The response object handles json

the requests library handles spaces in a url for you

```
address="Columbia University, New York, NY"
url="https://maps.googleapis.com/maps/api/geocode/json?address=%s" % (address)
response = requests.get(url).json()
```

response now points to a python data object

requests.json returns an exception if the JSON object is ill-formed

note that some http errors are returned as JSON

always check for exceptions!

JSON, Google API: Part 1

Python's Try and except mechanism to check errors

requests and json

```
address="Columbia University, New York, NY"
url="https://maps.googleapis.com/maps/api/geocode/json?address=%s" % (address)
try:
    response = requests.get(url)
    if not response.status_code == 200:
        print("HTTP error",response.status_code)
    else:
        try:
            response_data = response.json()
        except:
            print("Response not in valid JSON format")
except:
    print("Something went wrong with requests.get")
print(type(response_data))
```

XML: Part 1

Extensible Markup Language

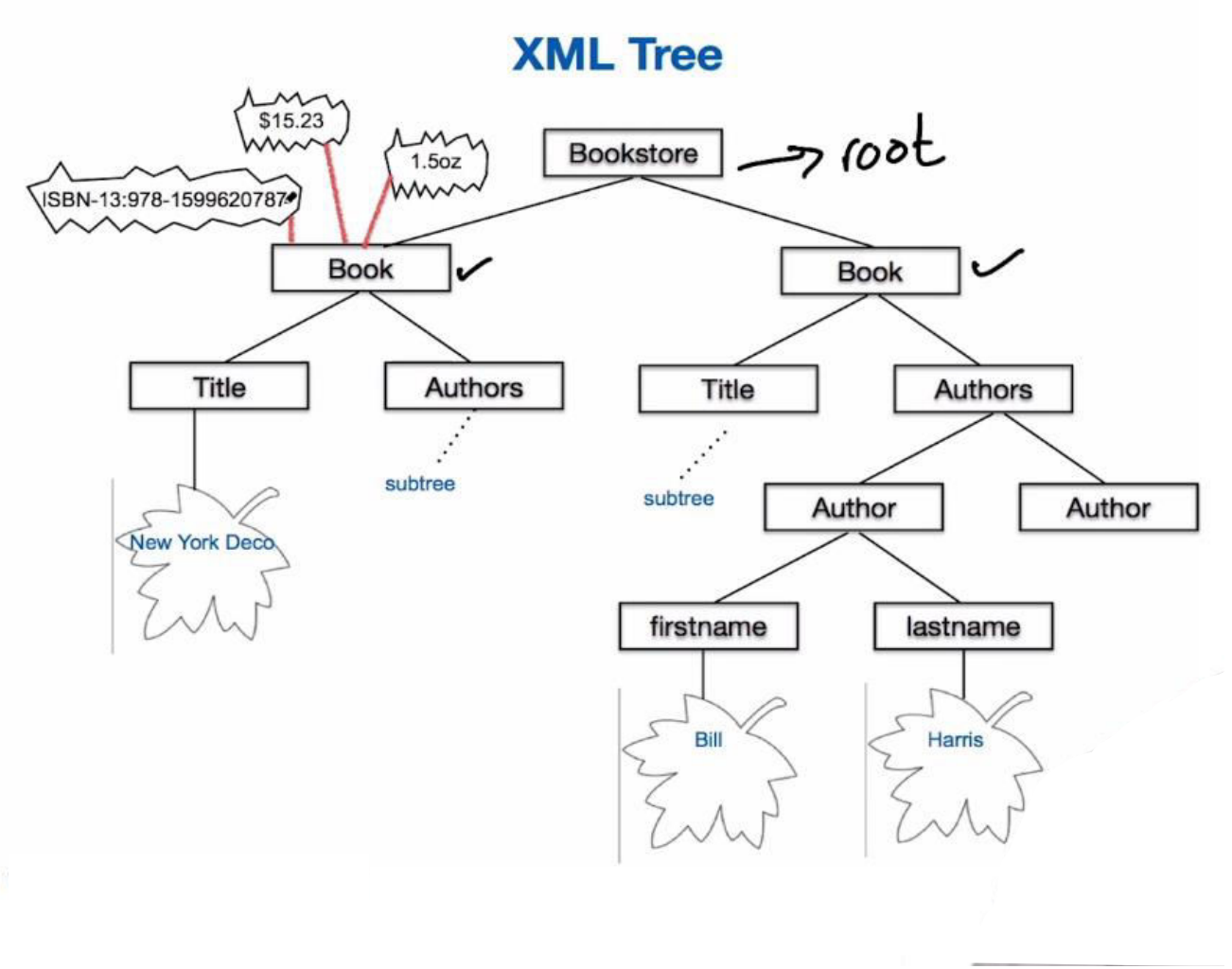
- Tree structure
- Tagged elements (nested)
- Attributes
- Text (leaves of the tree)

XML: Part 1

xml: Example

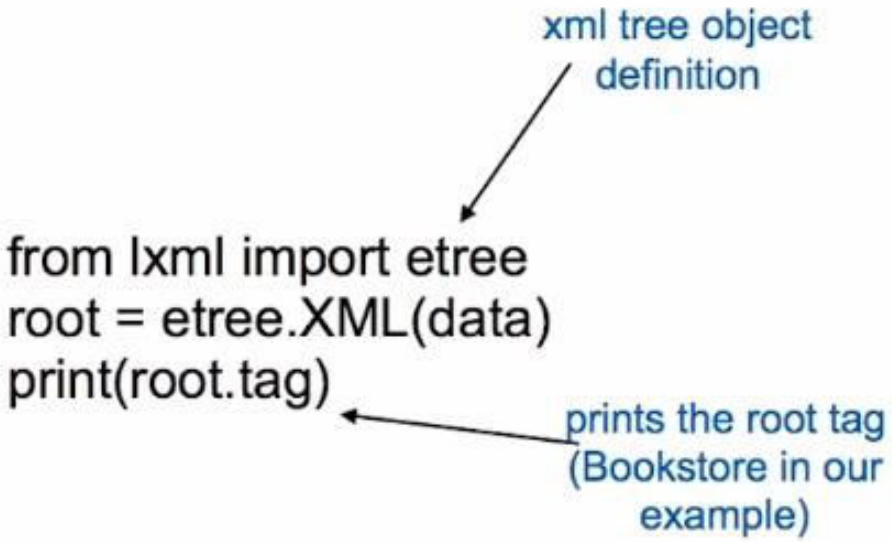
```
<Bookstore>
  <Book ISBN="ISBN-13:978-1599620787" Price="15.23" Weight="1.5">
    <Title>New York Deco</Title>
    <Authors>
      <Author Residence="New York City">
        <First_Name>Richard</First_Name>
        <Last_Name>Berenholtz</Last_Name>
      </Author>
    </Authors>
  </Book>
  <Book ISBN="ISBN-13:978-1579128562" Price="15.80">
    <Remark>
    Five Hundred Buildings of New York and over one million other books are available for Amazon Kindle.
    </Remark>
    <Title>Five Hundred Buildings of New York</Title>
    <Authors>
      <Author Residence="Beijing">
        <First_Name>Bill</First_Name>
        <Last_Name>Harris</Last_Name>
      </Author>
      <Author Residence="New York City">
        <First_Name>Jorg</First_Name>
        <Last_Name>Brockmann</Last_Name>
      </Author>
    </Authors>
  </Book>
</Bookstore>
```

XML: Part 1


XML Tree
$15.23
1.5oz
ISBN-13:978-1599620787
Bookstore
root
Book
Book
Title
Authors
Title
Authors
subtree
subtree
New York Deco
Author
Author
firstname
lastname
Bill
Harris

XML: Part 1

lxml: Python xml library



http://lxml.de/1.3/tutorial.html

XML: Part 2

Get the root of the tree

XML

- The python library - lxml - deals with converting an xml string to python objects and vice versa

In [ ]:
```
data_string = """
<Bookstore>
   <Book ISBN="ISBN-13:978-1599620787" Price="15.23" Weight="1.5">
      <Title>New York Deco</Title>
      <Authors>
         <Author Residence="New York City">
            <First_Name>Richard</First_Name>
            <Last_Name>Berenholtz</Last_Name>
         </Author>
      </Authors>
   </Book>
   <Book ISBN="ISBN-13:978-1579128562" Price="15.80">
      <Remark>
      Five Hundred Buildings of New York and over one million other books are available for Amazon Kindle.
      </Remark>
      <Title>Five Hundred Buildings of New York</Title>
      <Authors>
         <Author Residence="Beijing">
            <First_Name>Bill</First_Name>
            <Last_Name>Harris</Last_Name>
         </Author>
         <Author Residence="New York City">
            <First_Name>Jorg</First_Name>
            <Last_Name>Brockmann</Last_Name>
         </Author>
      </Authors>
   </Book>
</Bookstore>
"""
```

In [ ]:
```
from lxml import etree
root = etree.XML(data_string)
print(root.tag,type(root.tag))
```

In [ ]:
```
print(etree.tostring(root, pretty_print=True).decode("utf-8"))
```

XML: Part 2

Using Iterator and XPath

lxml: Iterating over elements

```
for element in root.iter():
    print(element.tag)
```

iter is an 'iterator'. it generates a sequence of elements in the order they appear in the xml code

lxml: using XPath

XPath: expression for navigating through an xml tree

```
for element in root.findall('Book/Title'):
    print(element.text)
```

```
In [47]: for element in root.iter("Author"):
             print(element.find('First_Name').text,element.find('Last_Name').text)

Richard Berenholtz
Bill Harris
Jorg Brockmann
```

```
In [53]: for element in root.findall("Book/Title"):
             print(element.text)

New York Deco
Five Hundred Buildings of New York
```

XML: Part 2

lxml: Finding by attribute value

Using values of attributes as filters

- Example: Find the first name of the author of a book that weighs 1.5 oz

```
In [55]: root.find('Book[@Weight="1.5"]/Authors/Author/First_Name').text
Out[55]: 'Richard'
```

root.find('Book[@Weight="1.5"]/Authors/Author/First_Name').text

www.emeritus.org