SI 601

Retrieving and manipulating structured data: HTML, XML, JSON, and Web APIs

Chris Teplovs (cteplovs@umich.edu)

Lead Developer, Digital Innovation Greenhouse

Adjunct Lecturer, School of Information

Some material courtesy of: Kevyn Collins-Thompson,

Lab & Homework Updates

- Homework 1 and Lab 1 grades to be released shortly: see SungJin (sjnam@umich.edu) for questions
- 1-page proposal due today
- Homework 3, Lab 3 released

SI 601 Data Manipulation: Class Schedule

(Some details may change)

Date	Topic	Assignments Due (before start of class)
Sep 9	Course introduction Basics of Programming with Python	Install software as described in welcome email
Sep 16	Text Processing and Pattern Extraction with Regular Expressions	Homework 1, Lab 1
Sep 23	**Fetching and Parsing Web content: HTML, JSON, XML**	**Homework 2, Lab 2 1-page Project Proposal Due**
Sep 30	Fetching data from Large Online Services Querying data in a SQL Database	Homework 3, Lab 3
Oct 7	Large-scale data manipulation with MapReduce and Hadoop	Homework 4, Lab 4
Oct 14	Advanced topics: learning analytics, synthetic data	Homework 5, Lab 5
Oct 21	Course Review, Final project presentations	Project report due

Today's Class Roadmap

1. Fetching Web content: `urllib2`
2. Parsing Web content: `beautifulsoup`
   (a) HTML parsing and manipulation
   (b) XML parsing and manipulation
3. JSON parsing and manipulation: `json`
4. Web services
5. Graph visualization: `pydot` and GraphViz

Lab 3: HTML and JSON

Homework 3: Putting it all together

You know the data's out there.. but how do you get it?

- Important to understand the basics of Web data transfer

- Original Web = **HTML** (what) + **URL** (where) + **HTTP** (how)

- URL: Uniform Resource Locator
  - Identifies a resource http://www.umich.edu/index.html
  - First part of a URL is the **protocol** to use
  - A protocol is a way of communicating that's agreed on in advance

- HTTP: Hyper Text Transfer Protocol
  - The network protocol of the Web
  - Transfers *resources* not just files: a resource is identified by a URL
    - Files, dynamically-generated server script output, media stream…
  - HTTP specifies the format of a request and a response
  - HTTP is a state-less protocol: does not maintain connection information between transactions.

- Client-server model
  1. a) Client (your browser) opens a connection and
     b) Sends a request message to an HTTP server (www.microsoft.com)
  2. Server returns a response message, usually containing requested resource.

The HTTP request model

http://ai.umich.edu/

Client

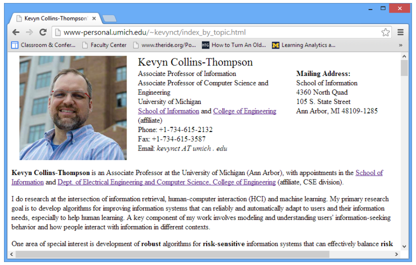
Request Line

```
GET / HTTP/1.1
Accept:*/*
Accept-Language: en-gb
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0)
Host: www.umich.com
Connection: Keep-Alive
```

Server

1. HTTP Request

2. HTTP Response
(with HTML page)



Kevyn Collins-Thompson

www-personal.umich.edu/~kevynct/index_by_topic.html

Kevyn Collins-Thompson
Associate Professor of Information
Associate Professor of Computer Science and Engineering
University of Michigan
School of Information and College of Engineering (affiliate)
Phone: +1-734-615-2132
Fax: +1-734-615-3587
Email: *kevynct AT umich . edu*

Mailing Address:
School of Information
4360 North Quad
105 S. State Street
Ann Arbor, MI 48109-1285

Kevyn Collins-Thompson is an Associate Professor at the University of Michigan (Ann Arbor), with appointments in the School of Information and Dept. of Electrical Engineering and Computer Science, College of Engineering (affiliate, CSE division).

I do research at the intersection of information retrieval, human-computer interaction (HCI) and machine learning. My primary research goal is to develop algorithms for improving information systems that can reliably and automatically adapt to users and their information needs, especially to help human learning. A key component of my work involves modeling and understanding users' information-seeking behavior and how people interact with information in different contexts.

One area of special interest is development of **robust** algorithms for **risk-sensitive** information systems that can effectively balance **risk**

Status Line

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.1
D  e: Sun, 22 Sep 2013 12:04:43 GMT
X    ered-By: ASP.NET
X     t-Version: 1.1.4322
          trol: no-cache
           cache Expires  -1
              text/html; charset=utf-8
            ngth: 8307

<
     ead>
...
```

Content Type
(and Encoding)

text/html is a MIME type

Examples of other types:
text/plain, application/json, application/pdf
image/gif, video/quicktime

Content

Python module for fetching Web resources: urllib2

```
import urllib2
response = urllib2.urlopen('http://ai.umich.edu/')
html_doc = response.read()
```

Also works for other transfer protocols, e.g. file:// and ftp://

Status codes

```
response.code  == 200  Success
response.code  == 401  Authentication required
response.code  == 403  Request forbidden
response.code  == 404  Page not found
```

`response.code in 300-range` are redirects, but default handlers process those and you won't see them unless you do special things.

Other urllib2 methods

```
response.geturl()
```

The actual URL fetched (may redirect from what you requested)

```
>>> response =
urllib2.urlopen('http://digitaleducation.umich.edu/')
>>> response.geturl()
'http://rsonal.umich.edu/~kevynct/'
```

```
response.info()
```

An instance of `httplib.HTTPMessage` describing the page fetched. Contains a dict:

```
>>> response.info().dict
{'content-length': '32890', 'accept-ranges': 'bytes',
'server': 'Apache', 'connection': 'close', 'date': 'Sat,
21 Sep 2013 06:04:50 GMT', 'content-type': 'text/html;
charset=utf-8'}
```

Parsing structured content

Beautiful Soup
is a powerful, widely-used parsing module

- A Python module that wraps existing HTML, XML parsers
- Installation:

```
pip install beautifulsoup4
easy_install beautifulsoup4
```

- It does Unicode conversion:
  - Incoming docs (with HTML entities) → Unicode
  - Output docs → UTF8
- After parsing a page, you can do things like this:
  - Find all the links on the page
  - Find all the links of class externalLink
  - Find all the links whose urls match "foo.com"
  - Find the table heading that's got bold text, then get that text

Reference: http://www.crummy.com/software/BeautifulSoup/

A word about HTML parsing

- HTML looks simple enough: it must be really easy to parse Web pages, right?
- Wrong! It's surprisingly difficult.
- Reason #1: Many Web pages are malformed. Part of the hardest part of parsing is to find and fix these..

```
BeautifulSoup("<a><b /></a>")
# <html><head></head><body><a><b></b></a></body></html>
```

- Reason #2: Some Web pages are technically valid, but auto-generated HTML that breaks the parser's worst-case assumptions

```
<a href="./foo/../foo/../[681 times]/baz">bar</a>
```

Beautiful Soup Example

To parse a document, pass it into the BeautifulSoup constructor.
You can pass in a string or an open filehandle:

```
from bs4 import BeautifulSoup

soup = BeautifulSoup(open("index.html"))
soup = BeautifulSoup("<html>data</html>")
```

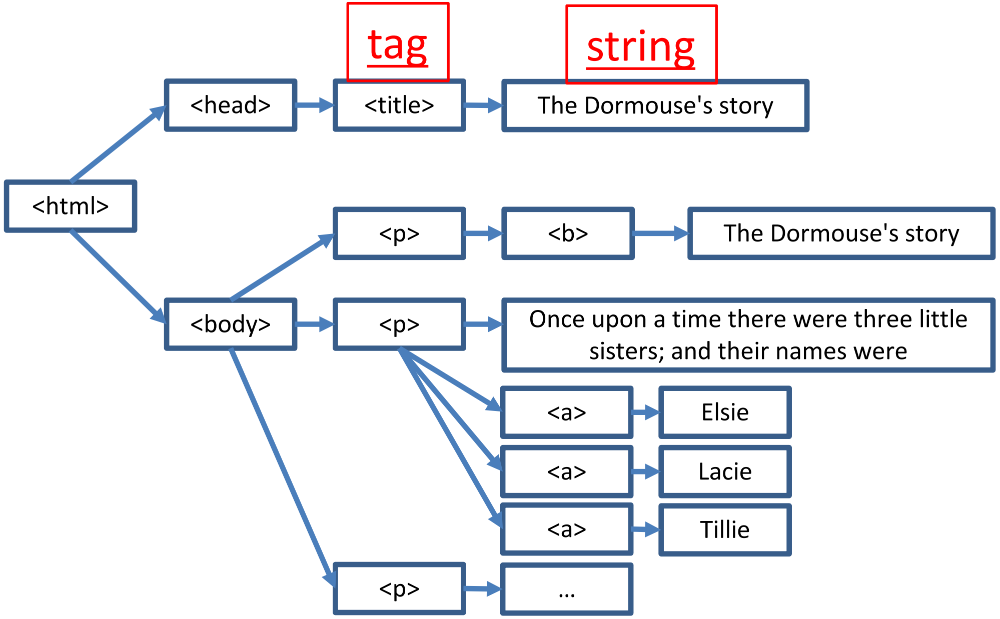
1. First, the document is converted to Unicode: HTML entities are converted to Unicode characters

```
    BeautifulSoup("Sacr&eacute; bleu!")
    <html><head></head><body>Sacré bleu!</body></html>
```

2. Beautiful Soup then parses the document using the best available parser.
   It will use an HTML parser unless you specifically tell it to use an XML parser. (See Parsing XML.)
   It turns a complex HTML document into a complex tree of Python objects
3. You can manipulate, e.g. change tag names
4. Then optionally save a new file

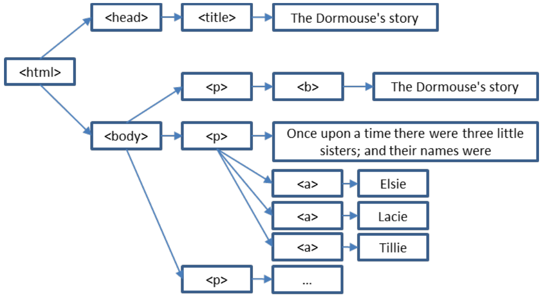
BeautifulSoup tag tree: A graphical example of simple HTML

```
<html><head><title>The Dormouse's story</title></head>
<body>
<p class="title"><b>The Dormouse's story</b></p>

<p class="story">Once upon a time there were three little sisters; and their names were
<a href="http://example.com/elsie" class="sister" id="link1">Elsie</a>,
<a href="http://example.com/lacie" class="sister" id="link2">Lacie</a> and
<a href="http://example.com/tillie" class="sister" id="link3">Tillie</a>;
and they lived at the bottom of a well.</p>

<p class="story">...</p>
```



Source: http://www.crummy.com/software/BeautifulSoup/bs4/doc/

Navigating the parse tree


<html>
<head>
<title>
The Dormouse's story
<body>
<p>
<b>
The Dormouse's story
<p>
Once upon a time there were three little sisters; and their names were
<a>
Elsie
<a>
Lacie
<a>
Tillie
<p>
...

- Use tag name to get the *first* tag by that name

```
soup.head
soup.title
soup.a
```

- These return tag objects.
- If a tag's child is a string, use **.string**

```
soup.title.string  'The Dormouse's story'
```

- You can zoom in like this:

```
soup.body.b
```

- Getting a tag's direct children: .contents and .children

```
head_tag = soup.head
head_tag.contents
[<title>The Dormouse's story</title>]
title_tag = head_tag.contents[0]
title_tag
# <title>The Dormouse's story</title>
title_tag.contents
# [u'The Dormouse's story']
for child in title_tag.children
   print(child)
for child in head_tag.descendants # recursively iterate
   print(child)
```

Searching and filtering the parse tree

Searching and filtering the tree: `find_all` method

```
find_all(tag_name, attributes, recursive,
         text, limit, **kwargs)
```

Returns a ResultSet object: a list of tags and strings

tag_name argument:

- `soup.find_all('a')`
- `movie_table = soup.find_all('table')[0]`
  `for row in movie_table.find_all('tr'):`
- `soup.find_all(["a", "b"])` finds all 'a' AND all 'b' tags

Custom name functions

```
def has_class_but_no_id(tag):
    return tag.has_attr('class')
        and not tag.has_attr('id')
```

Pass this function into find_all() and you'll pick up all the <p> tags:

```
soup.find_all(has_class_but_no_id)
# [<p class="title"><b>The Dormouse's story</b></p>,
#  <p class="story">Once upon a time there were...</p>,
#  <p class="story">...</p>]
```

In general, find_all looks through all tag descendants and returns the ones that match your filter conditions.

```
<html><head><title>The Dormouse's
story</title></head>
<body>
<p class="title"><b>The Dormouse's
story</b></p>

<p class="story">Once upon a time there
were three little sisters; and their
names
<a href="http://example.com/elsie"
class="sister" id="link1">Elsie</a>,
<a href="http://example.com/lacie"
class="sister" id="link2">Lacie</a> and
<a href="http://example.com/tillie"
class="sister" id="link3">Tillie</a>;
and they lived at the bottom of a
well.</p>

<p class="story">...</p>
```

Searching and filtering the tree: `find_all` method

```
find_all(tag_name, attributes, recursive,
text, limit,**kwargs)
```

`attributes` argument:

Any unrecognized argument will be turned into a filter on that tag attribute:

```
soup.find_all(href="elsie")
# [<a class="sister" href="http://example.com/elsie"
id="link1">Elsie</a>]
```

`text` argument:

```
soup.find_all("a", text="Elsie")
# [<a href="http://example.com/elsie" class="sister"
id="link1">Elsie</a>]
```

`limit` argument:

```
soup.find_all("a", limit=2)
# [<a class="sister" href="http://example.com/elsie"
id="link1">Elsie</a>,
#  <a class="sister" href="http://example.com/lacie"
id="link2">Lacie</a>]
```

Good for large documents where you only need a few results

`recursive` argument:

Search direct children only: `recursive=False`

```
<html><head><title>The Dormouse's
story</title></head>
<body>
<p class="title"><b>The Dormouse's
story</b></p>

<p class="story">Once upon a time there
were three little sisters; and their
names
<a href="http://example.com/elsie"
class="sister" id="link1">Elsie</a>,
<a href="http://example.com/lacie"
class="sister" id="link2">Lacie</a> and
<a href="http://example.com/tillie"
class="sister" id="link3">Tillie</a>;
and they lived at the bottom of a
well.</p>

<p class="story">...</p>
```

You can filter the tree with regular expressions (remember those?)

```
import re
for tag in
soup.find_all(re.compile("^b")):
    print(tag.name)
# body
# b
for tag in
soup.find_all(re.compile("t")):
    print(tag.name)
# html
# title
```

```
<html><head><title>The Dormouse's
story</title></head>
<body>
<p class="title"><b>The Dormouse's
story</b></p>

<p class="story">Once upon a time there
were three little sisters; and their
names
<a href="http://example.com/elsie"
class="sister" id="link1">Elsie</a>,
<a href="http://example.com/lacie"
class="sister" id="link2">Lacie</a> and
<a href="http://example.com/tillie"
class="sister" id="link3">Tillie</a>;
and they lived at the bottom of a
well.</p>

<p class="story">...</p>
```

Source: http://www.crummy.com/software/BeautifulSoup/bs4/doc/

find_all exercises for the reader

"Find all the links on the page"

"Find all the links of class externalLink"

"Find all the links whose urls match "foo.com"

"Find the table heading that's got bold text, then get that text"

Other beautifulsoup methods

`prettify()`: Turn a parse tree into nicely formatted Unicode with each HTML tag on its own line

```
sibling_soup = 
BeautifulSoup("<a><b>text1</b><c>text2</c></b></a>")
print(sibling_soup.prettify())
# <html>
#  <body>
#   <a>
#    <b>
#     text1
#    </b>
#    <c>
#     text2
#    </c>
#   </a>
#  </body>
# </html>
```

Source: http://www.crummy.com/software/BeautifulSoup/bs4/doc/

Manipulating HTML: tags

```
soup = BeautifulSoup('<b class="boldest">Extremely bold</b>')
tag = soup.b

tag.name = "blockquote"
tag['class'] = 'verybold'
tag['id'] = 1
tag
# <blockquote class="verybold" id="1">Extremely bold</blockquote>

del tag['class']
del tag['id']
tag
# <blockquote>Extremely bold</blockquote>
```

Source: http://www.crummy.com/software/BeautifulSoup/bs4/doc/

Manipulating HTML: strings

```
markup = '<a href="http://example.com/">I linked to
<i>example.com</i></a>'
soup = BeautifulSoup(markup)

tag = soup.a
tag.string = "New link text."
tag
# <a href="http://example.com/">New link text.</a>
```

Source: http://www.crummy.com/software/BeautifulSoup/bs4/doc/

Fetching and parsing together

```
response =
urllib2.urlopen('http://www.imdb.com/search/
title?at=0&sort=num_votes&count=100')
html_doc = response.read()
soup = BeautifulSoup(html_doc)
movie_table = soup.find_all('table')[0]
for row in movie_table.find_all('tr'):
   row.a.get('href')
```

XML

Beyond Web pages: XML
RSS feeds use XML

- What are RSS feeds?
- Rich Site Summary
- Really Simple Syndication
- Channel- and item-based model
- Used for news feeds, weather, …
- Developed starting in the late 1990s by Netscape, Dave Winer, and others

```
<?xml version="1.0" encoding="UTF-8" ?>
<rss version="2.0">
<channel>
 <title>RSS Title</title>
 <description>This is an example of an RSS feed</description>
 <link>http://www.someexamplerssdomain.com/main.html</link>
 <lastBuildDate>Mon, 06 Sep 2010 00:01:00 +0000 </lastBuildDate>
 <pubDate>Mon, 06 Sep 2009 16:20:00 +0000 </pubDate>
 <ttl>1800</ttl>
 <item>
  <title>Example entry</title>
  <description>Here is some text containing an interesting
description.</description>
  <link>http://www.wikipedia.org/</link>
  <guid>unique string per item</guid>
  <pubDate>Mon, 06 Sep 2009 16:20:00 +0000 </pubDate>
 </item>
 <item>
   …
</item>
</channel>
</rss>
```

Examples:

http://rss.weather.com/weather/rss/local/48109

http://news.yahoo.com/rss/entertainment

Source: http://en.wikipedia.org/wiki/RSS

XML Facts

- eXtensible Markup Language
- Separation of data and its presentation
  - in contrast to HTML
- Simple tag-based file format
- Developed for the Web 1996-1998 out of older SGML tag spec
- File type is .xml, MIME type is application/xml
- Applications
  - Heavy-duty web services
  - Document archiving
  - Information exchange between applications
  - XML databases
  - Web feeds: RSS feeds, Atom feeds, etc.

http://en.wikipedia.org/wiki/XML

Example XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <note>
      <to>Tove</to>
      <from>Jani</from>
      <heading>Reminder</heading>
      <body>Don't forget me this weekend!</body>
  </note>
```

http://www.w3schools.com/xml/xml_tree.asp

Root Node

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <note>
     <to>Tove</to>
     <from>Jani</from>
     <heading>Reminder</heading>
     <body>Don't forget me this weekend!</body>
  </note>
```

Child nodes

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <note>
      <to>Tove</to>
      <from>Jani</from>
      <heading>Reminder</heading>
      <body>Don't forget me this weekend!</body>
  </note>
```

Attributes

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <note date="10/01/2008">
      <to>Tove</to>
      <from>Jani</from>
      <heading>Reminder</heading>
      <body>Don't forget me this weekend!</body>
  </note>
```

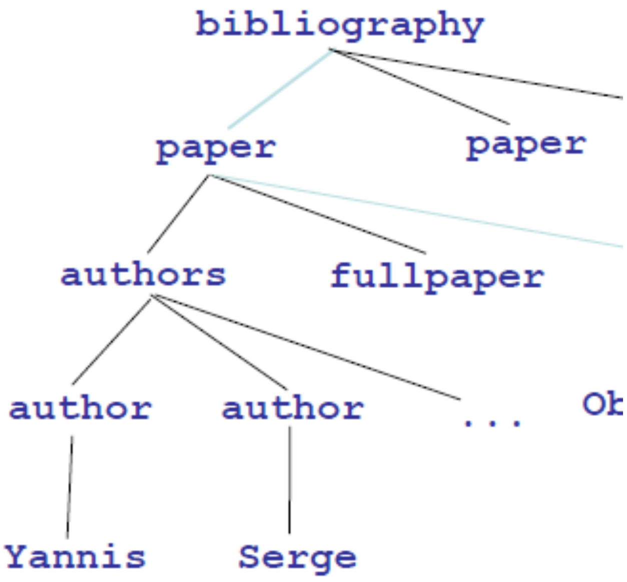
date = attribute name

"10/01/2008"= attribute value

Example XML with Unicode encoding

```
<?xml version="1.0" encoding="UTF-8" ?>
<俄语>данные</俄语>
```

Tree Structure



```
<bibliography>
   <paper >
       <authors>
              <author>Yannis</author>
              <author>Serge</author>
   ...
       </authors>
       <fullpaper>Object Fusion</fullpaper>
   ...
   </paper>
</bibliography>
```

XML parsing in Python

- BeautifulSoup(markup,"xml")
  - Must install the lxml parser first (`pip install lxml`)
- Various other modules can be used:
  - xml.dom.* modules
  - xml.sax.* modules – an event-based API meant to parse huge documents "on the fly" without loading them wholly into memory
  - xml.parser.expat – a direct, low level API to the C-based expat parser
  - **xml.etree.ElementTree** - provides a lightweight Pythonic API that is easy to work with

XML tag clouds!

Alternate Python XML parsing

Default XML support in python

```
import xml.etree.ElementTree as elementtree
```

Install ElementTree toolkit:

- http://effbot.org/downloads/#elementtree

This will install the elementtree package

Useful documentation of elements, etc.:

Element Type: http://effbot.org/zone/element.htm

Element has: tag, attributes, text, child elements

```
from elementtree.ElementTree import parse
dom = parse('senate-lobbying-2013_1_1_1.xml')
root = dom.getroot()
for node in root:
    print node.tag
```

XML government lobbying data sample

See `senate-lobbying-2013_1_1_1.xml` in Resources/Week 3

```
<?xml version='1.0' encoding='UTF-16'?>
<PublicFilings>
<Filing ID="306B3144-3E4F-48CF-98F1-C6BF455B6A6B“  Year="2012" Received="2013-01-
01T00:58:03.067" Amount="15000" Period="4th Quarter (Oct 1 - Dec 31)">
   <Registrant RegistrantID="6848" RegistrantName="Marshall Brachman" />
   <Client ClientName="ADAMS COUNTY COLORADO" ClientID="12"
   <Lobbyists>
      <Lobbyist LobbyistName="BRACHMAN, MARSHALL A" />
   </Lobbyists>
   <GovernmentEntities>
          <GovernmentEntity GovEntityName="SENATE" />
          <GovernmentEntity GovEntityName="Federal Aviation Administration (FAA)" />
          <GovernmentEntity GovEntityName="HOUSE OF REPRESENTATIVES" />
   </GovernmentEntities>
   <Issues>
      <Issue Code="BUDGET/APPROPRIATIONS" SpecificIssue="DERA funding regarding Rocky
Mountain Arsenal&#xA;DoD Appropriations for the above&#xA;TTHUD funding for railroad
grade separation&#xA;funding for contract tower program and commercial flight program"
/>
      <Issue Code="DEFENSE" SpecificIssue="DERA funding regarding Rocky Mountain
Arsenal&#xA;DoD Authorization" />
      <Issue Code="NATURAL RESOURCES" SpecificIssue="land trade issues regarding the
Rocky Mountain Arsenal NWP" />
   </Issues>
</Filing>
```

Tag Clouds with `pytagcloud`

```
from pytagcloud import create_tag_image, make_tags
from pytagcloud.lang.counter import get_tag_counts
YOUR_TEXT = "A tag cloud is a visual representation for
text data, typically\ used to depict keyword metadata
on websites, or to visualize free form text."
tags = make_tags(get_tag_counts(YOUR_TEXT),
      maxsize=120)
create_tag_image(tags, 'cloud_large.png',
       size=(900, 600), fontname='Lobster')
```

May need to limit input length to a representative sample of text if module is too slow.

https://pypi.python.org/pypi/pytagcloud/

Example: Creating tag cloud from lobbying data

```
<Issue Code="DEFENSE"
    SpecificIssue="DERA funding regarding Rocky Mountain Arsenal&#xA;
                   DoD Authorization" />
```

```
#import easy to use xml parser called minidom:
from pytagcloud import create_tag_image, make_tags
from pytagcloud.lang.counter import get_tag_counts
from elementtree.ElementTree import parse

allText = ""

dom = parse('senate-lobbying-2013_1_1_1.xml')
#retrieve the first xml tag (<tag>data</tag>) that the
parser finds with name tagName:
filinglist = dom.getroot()
for filing in filinglist:
    issues = filing.getiterator('Issues')
    if len(issues) > 0:
        issuelist = issues[0].getiterator('Issue')
        for i in issuelist:
            allText =
allText.join(i.attrib.get('SpecificIssue'))
            allText = allText.join(" ")

tags = make_tags(get_tag_counts(allText), maxsize=80)
create_tag_image(tags, 'lobbying_cloud_large.png',
size=(900, 600), fontname='Lobster')
import webbrowser
webbrowser.open('lobbying_cloud_large.png')
```

funding
act
federal
bill
issues
hr
reform
programs
program
appropriations
legislation
seeking
tax
support
grant
project
energy
health
education
transportation
related
trade
development
implementation
regulations
community
fy13
policy
food
appropriation
regulatory
protection
security
highway
defense
revitalization
environmental
national
operations
2011
access
care
science
general
monitor
research
export
water
business
credit
service
services
estate
sugar
labor
united
state
112
res
visa
wto
relating
affecting
farm
section
foreign
including
immigration
reauthorization
2012
map
advocate
safety
legislative
agriculture
treatment
regulation
procurement
veterans
investment
physician
reimbursement
medicare
administration
communities
housing
promotion
duty
frank
department
human
supporting
2013
economic
infrastructure
information
government
monitoring
conservation
technology
financial
agencies
travel
small
clean
srpt
public
military
provisions
hospitals
sustainable
continuing
insurance
coverage
authorization
cybersecurity
partnership
agjobs
school
workforce
tourism
consumer
regarding
industry
matters
agricultural
standards
construction
initiative
owned
oppose
drugs
assistance
manufacturing
processing
heritage
affordable
international
homeland
freedom
efficient
agency
lunch
activities
student
students
epa
fy
hrpt
dodd
domestic
pertaining
pipeline
production
cultural
redevelopment
fairness
hemisphere
products
extend
wastewater
within
equipment
preference
imaging
plans
urban
policies
changes
certain
healthcare
child
sexual
requirements
promote
temporarily
transparency
documentation
ergonomics
obesity
hospital
bills
market
coalition
training
global
benefits
medicaid
throughout
korea
4348
factors
comprehensive
electric
enhancing
jobs
japan
justice
temporary
direct
corporation
technologies
schools
keystone
patient
grants
private
intelligence
software
fund
biodiesel
biotechnology
american
climate
collection
reliability
conference
performance
committee
elementary
sharing
language
simplification
higher
relief
wall
rates
3855
2046
airport
tariff
innovation
center
centers
proposed
potential
extension
sequestration
surface
funds
provide
fuel
data
initiatives
corporate
commodities
sales
waiver
management
amendment
improve
passage
mental
america
1653
959
1746
117
hr3039
5741
2010
rural
internet
hydro
renewable
facilities
budget
commerce
congress
computers
ethanol
brownfield
appraisal
bonds

JSON

JSON:
JavaScript Object Notation

```
{"employees": [
        { "firstName":"John" , "lastName":"Doe" },
        { "firstName":"Anna" , "lastName":"Smith" },
        { "firstName":"Peter" , "lastName":"Jones" }
   ]
}
```

- JSON syntax is a subset of JavaScript object notation
  - Data is in **name/value pairs**
  - Data is separated by commas
  - **Curly braces** hold objects
  - **Square brackets** hold arrays
- Filename extension `".json"`
- MIME type (e.g. HTTP header) `application/json`

JSON:
JavaScript Object Notation

- Light-weight interchange format for storing and exchanging text information
  - Designed for saving/loading data to/from Javascript apps
  - Because of this similarity, instead of using a parser, a JavaScript program can use the built-in eval() function
- Similar to XML but smaller, easier and faster to parse
- Self-describing, language-independent
- Popular and widely available:
  - JSON parsers and generators exist for many programming languages and platforms

http://www.w3schools.com/json/json_intro.asp

XML vs JSON

- JSON is a lot like XML
  - JSON is plain text
  - JSON is "self-describing" (human readable)
  - JSON is hierarchical (values within values)
  - JSON can be parsed by JavaScript
  - JSON data can be transported using AJAX (client, async web apps)
- JSON is very different from XML
  - No end tags
  - Shorter
  - Quicker to read and write
  - Can be parsed using built-in JavaScript eval()
  - Uses arrays
  - No reserved words
- (Almost) a Subset of YAML 1.2 http://yaml.org/

AJAX applications

- Using XML
  - Fetch an XML document
  - Use the XML DOM to loop through the document
  - Extract values and store in variables
- Using JSON
  - Fetch a JSON string
  - eval() (or better, parse) the JSON string
  - Result: object!

JSON and Python

Default support:

```
import json
```

- Encoding Python objects & pretty printing

  ```
  json.dumps(…)
  ```

- Decoding JSON

  ```
  json.loads(…)
  ```

http://docs.python.org/2/library/json.html

Using the `json` module

```
import json

data = [ { 'a':'A', 'b':(2, 4), 'c':3.0 } ]
print 'DATA:', repr(data)   # repr: string version of an object

data_string = json.dumps(data) # dumps: create JSON string
print 'JSON:', data_string
```

Values are encoded in a manner very similar to Python’s repr() output.

```
$ python json_simple_types.py

DATA: [{'a': 'A', 'c': 3.0, 'b': (2, 4)}]
JSON: [{"a": "A", "c": 3.0, "b": [2, 4]}]
```

Source: http://www.doughellmann.com/PyMOTW/json/

Using the `json` module

Encoding, then re-decoding may not give exactly the same type of object.

```
import json

data = [ { 'a':'A', 'b':(2, 4), 'c':3.0 } ]
data_string = json.dumps(data)
print 'ENCODED:', data_string

decoded = json.loads(data_string)
print 'DECODED:', decoded

print 'ORIGINAL:', type(data[0]['b'])
print 'DECODED :', type(decoded[0]['b'])
```

To create JSON, strings are converted to **unicode** and **tuples** become **lists**.

```
$ python json_simple_types_decode.py

ENCODED: [{"a": "A", "c": 3.0, "b": [2, 4]}]
DECODED: [{u'a': u'A', u'c': 3.0, u'b': [2, 4]}]
ORIGINAL: <type 'tuple'>
DECODED : <type 'list'>
```

Source: http://www.doughellmann.com/PyMOTW/json/

Many useful options to `dumps` and `loads`: `sort_keys, indent`

```
data = [ { 'a':'A', 'b':(2, 4), 'c':3.0 } ]
print 'JSON:', json.dumps(data)
```

sort_keys

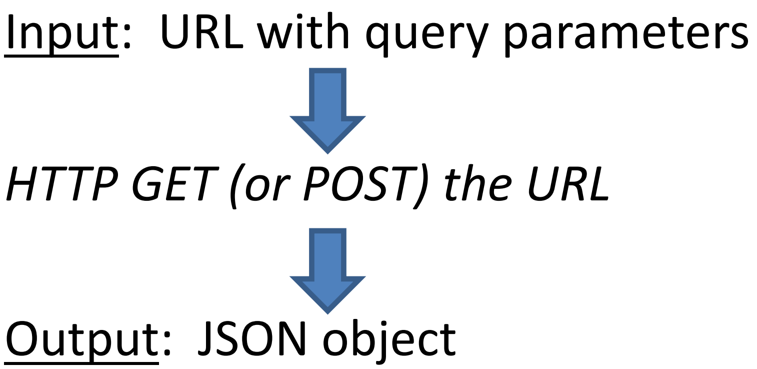
```
print 'SORT:', json.dumps(data, sort_keys=True)
JSON: [{"a": "A", "c": 3.0, "b": [2, 4]}]   # random
SORT: [{"a": "A", "b": [2, 4], "c": 3.0}]   # sorted
```

indent

```
print 'INDENT:', json.dumps(data, sort_keys=True, indent=2)
INDENT: [
  {
    "a": "A",
    "b": [
      2,
      4
    ],
    "c": 3.0
  }
]
```

Web APIs

Web services often use APIs
(API: Application Programming Interface)

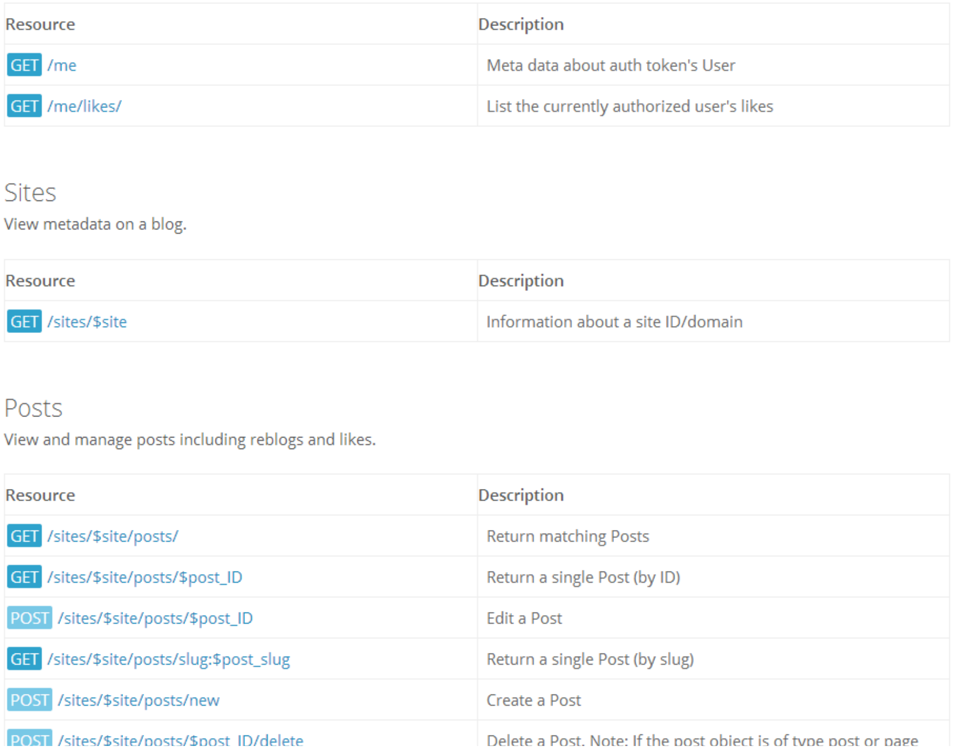


Input: URL with query parameters

HTTP GET (or POST) the URL

Output: JSON object

WordPress Blog API



Resource	Description
GET /me	Meta data about auth token's User
GET /me/likes/	List the currently authorized user's likes

Sites

View metadata on a blog.

Resource	Description
GET /sites/$site	Information about a site ID/domain

Posts

View and manage posts including reblogs and likes.

Resource	Description
GET /sites/$site/posts/	Return matching Posts
GET /sites/$site/posts/$post_ID	Return a single Post (by ID)
POST /sites/$site/posts/$post_ID	Edit a Post
GET /sites/$site/posts/slug:$post_slug	Return a single Post (by slug)
POST /sites/$site/posts/new	Create a Post
POST /sites/$site/posts/$post_ID/delete	Delete a Post. Note: If the post object is of type post or page

Source: http://developer.wordpress.com/docs/api/

An acronym you should know: Representational State Transfer (REST) APIs

- Uniform client–server interface
  - Uniform interface separates clients from servers.
  - Clients are not ncerned with data st age (portabil
  - Servers are not co ed with the u terface (ity, scalability)
- Stateless
  - No client context being s
  - Each client request has all
- Cacheable
  - Clients can cache r
  - Responses define them g stale data
- Layered system
  - A client can't t the connec
- Code on demand (optional)
  - Servers can temporarily ext funct of a y the tran r of executable code.
  - e.g. Javascript

Most HTTP-based Web services (and certainly the simple ones) are REST APIs

http://en.wikipedia.org/wiki/Representational_state_transfer

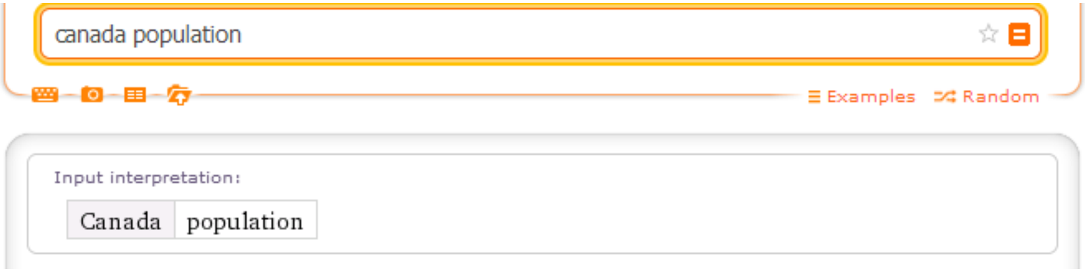
WordPress API: Find all posts for a given blog

Request: https://public-api.wordpress.com/rest/v1/sites/**en.blog.wordpress.com**/posts/?pretty=1

```
{
  "found": 894,
  "posts": [
    {
      "ID": 20243,
      "author": {
        "ID": 47411601,
        "email": false,
        "name": "Ben Huberman",
        "nice_name": "benhuberman",
        "URL": "",
        "avatar_URL": "http:\/\/0.gravatar.com\/avatar\/663dcd498e8c5f255bfb230a7ba07678?s=96&d=retro",
        "profile_URL": "http:\/\/en.gravatar.com\/benhuberman"
      },
      "date": "2013-09-20T16:00:41+00:00",
      "modified": "2013-09-19T18:00:57+00:00",
      "title": "Unbound Creativity: Art Blogs on WordPress.com",
      "URL": "http:\/\/en.blog.wordpress.com\/2013\/09\/20\/unbound-art-blogs\/",
      "short_URL": "http:\/\/wp.me\/pf2B5-5gv",
      "content": "<p>From painting and photography... \n",
      "excerpt": "<p>From painting and photography to performance art, the art scene on\n",
      "slug": "unbound-art-blogs",
      "status": "publish",
```

Content Types used by Web Services

- Mainly two types:
  - JSON-based
    - Facebook, Twitter, Yelp, most Google web services
  - XML-based
    - Some Bing, Google web services, eBay
- Most Web Services are JSON-based now
  - Some provide both JSON and XML format APIs
- Industry has been moving away from XML to JSON:
  - XML is overkill
  - XML is not easy to work with

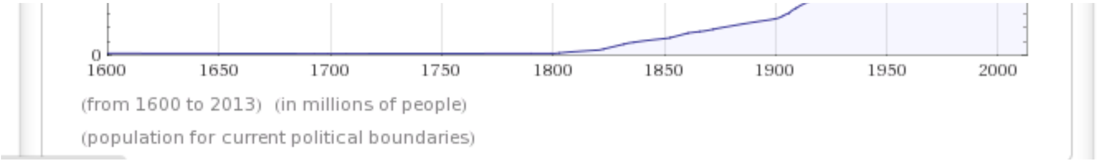


Input interpretation:

Canada	population

This XML file does not appear to have any style information associated with it. The document tree is shown below.

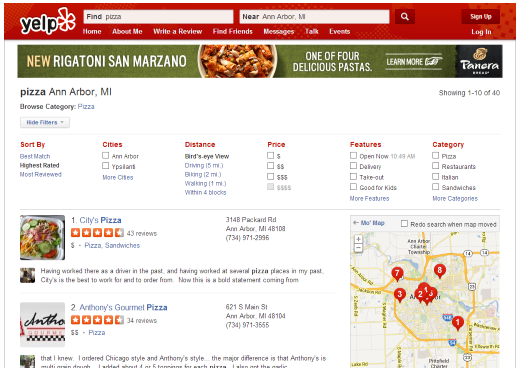
```
▼<queryresult success="true" error="false" numpods="7" datatypes="AdministrativeDivision,City,Country" timedout="" timedoutpods="" timing="2.981" parsetiming="0.229" parsetimedout="false"
 recalculate="" id="MSPa62461cd41b4a3d3c6b5e000023i9iegia06ifc8g" host="http://www4b.wolframalpha.com" server="33" related="http://www4b.wolframalpha.com/api/v2/relatedQueries.jsp?
 id=MSPa62471cd41b4a3d3c6b5e00004c40a47g4fe213ib&s=33" version="2.6">
  ▼<pod title="Input interpretation" scanner="Identity" id="Input" position="100" error="false" numsubpods="1">
    ▼<subpod title="">
      <plaintext>Canada | population</plaintext>
      <img src="http://www4b.wolframalpha.com/Calculate/MSP/MSP62481cd41b4a3d3c6b5e000029i10aec59cha5d1?MSPStoreType=image/gif&s=33" alt="Canada | population" title="Canada | population"
      width="152" height="23"/>
    </subpod>
  </pod>
  ▼<pod title="Result" scanner="Data" id="Result" position="200" error="false" numsubpods="1" primary="true">
    ▼<subpod title="">
      ▼<plaintext>
        35 million people (world rank: 37th) (2013 estimate)
      </plaintext>
      <img src="http://www4b.wolframalpha.com/Calculate/MSP/MSP62491cd41b4a3d3c6b5e00002592d7ac22b2eag0?MSPStoreType=image/gif&s=33" alt="35 million people (world rank: 37th) (2013 estimate)"
      title="35 million people (world rank: 37th) (2013 estimate)" width="303" height="18"/>
    </subpod>
  </pod>
  ▼<pod title="Recent population history" scanner="Data" id="RecentHistory:Population:CountryData" position="300" error="false" numsubpods="1">
    ▼<subpod title="">
      <plaintext/>
      <img src="http://www4b.wolframalpha.com/Calculate/MSP/MSP62501cd41b4a3d3c6b5e0000672fc77131101650?MSPStoreType=image/gif&s=33" alt="" title="" width="377" height="160"/>
    </subpod>
    ▼<states count="2">
      <state name="Show projections" input="RecentHistory:Population:CountryData__Show projections"/>
      <state name="Log scale" input="RecentHistory:Population:CountryData__Log scale"/>
    </states>
  </pod>
  ▼<pod title="Long-term population history" scanner="Data" id="LongTermHistory:Population:CountryData" position="400" error="false" numsubpods="1">
    ▼<subpod title="">
      <plaintext/>
      <img src="http://www4b.wolframalpha.com/Calculate/MSP/MSP62511cd41b4a3d3c6b5e00005b3bf2061f4hg1fi?MSPStoreType=image/gif&s=33" alt="" title="" width="496" height="207"/>
    </subpod>
    ▼<states count="2">
      <state name="Show projections" input="LongTermHistory:Population:CountryData__Show projections"/>
      <state name="Log scale" input="LongTermHistory:Population:CountryData__Log scale"/>
    </states>
  </pod>
  ▼<pod title="Demographics" scanner="Data" id="DemographicProperties:CountryData" position="500" error="false" numsubpods="1">
    ▼<subpod title="">
      ▼<plaintext>
```



(from 1600 to 2013) (in millions of people)

(population for current political boundaries)

Yelp API



pizza Ann Arbor, MI

Showing 1-10 of 40

Browse Category: Pizza

1. City's **Pizza** — 43 reviews — $ · Pizza, Sandwiches
   3148 Packard Rd
   Ann Arbor, MI 48108
   (734) 971-2996

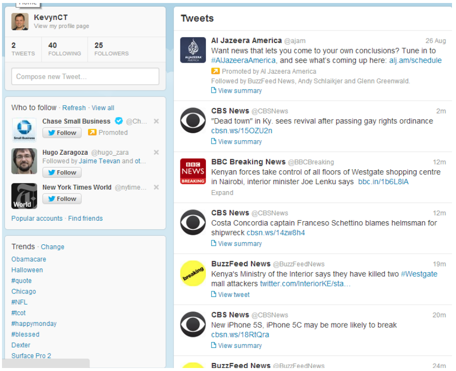
   Having worked there as a driver in the past, and having worked at several **pizza** places in my past, City's is the best to work for and to order from. Now this is a bold statement coming from

2. Anthony's Gourmet **Pizza** — 34 reviews — $$ · Pizza
   621 S Main St
   Ann Arbor, MI 48104
   (734) 971-3555

   that I knew. I ordered Chicago style and Anthony's style... the major difference is that Anthony's is

- JSON based
- Documentation: http://www.yelp.com/developers/documentation/v2/overview
- Needs authentication
- Example code in Files/Lecture/Week 3:…
  – yelp_api_example.py

Twitter API



Tweets

Al Jazeera America @ajam 26 Aug
Want news that lets you come to your own conclusions? Tune in to #AlJazeeraAmerica, and see what's coming up here: alj.am/schedule
Promoted by Al Jazeera America
Followed by BuzzFeed News, Andy Schlaikjer and Glenn Greenwald.
View summary

CBS News @CBSNews 2m
"Dead town" in Ky. sees revival after passing gay rights ordinance cbsn.ws/15OZU2n
View summary

BBC Breaking News @BBCBreaking 12m
Kenyan forces take control of all floors of Westgate shopping centre in Nairobi, interior minister Joe Lenku says bbc.in/1b6L8IA
Expand

CBS News @CBSNews 12m
Costa Concordia captain Franceso Schettino blames helmsman for shipwreck cbsn.ws/14zw8h4
View summary

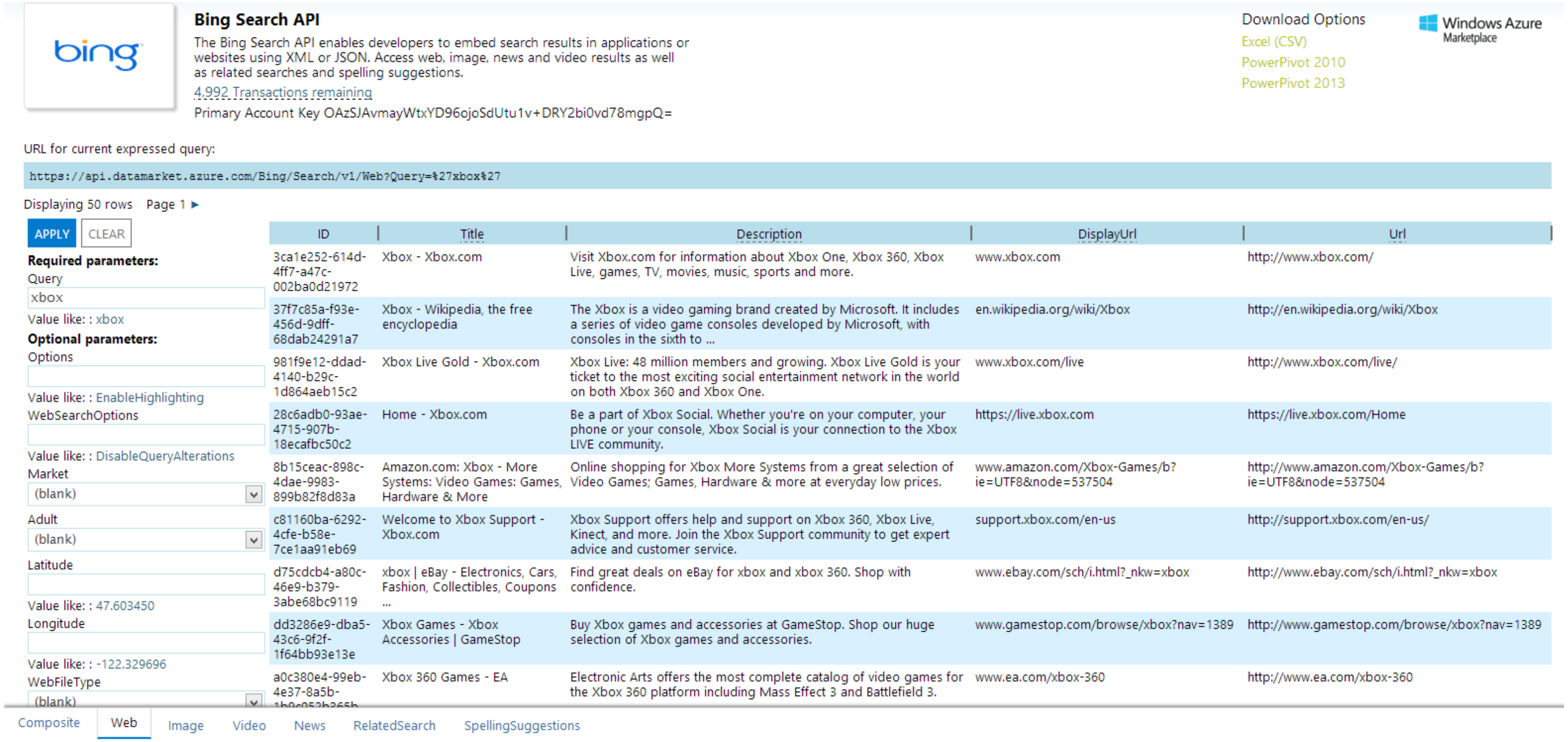
BuzzFeed News @BuzzFeedNews 19m
Kenya's Ministry of the Interior says they have killed two #Westgate mall attackers twitter.com/InteriorKE/sta...
View tweet

CBS News @CBSNews 20m
New iPhone 5S, iPhone 5C may be more likely to break cbsn.ws/18RtQra
View summary

BuzzFeed News @BuzzFeedNews 24m

- JSON based
- Documentation: https://dev.twitter.com/docs/api/1.1
- Twitter API also needs authentication
- Example code in Files/Lecture/Week 3:...
  - – twitter_api_example.py

Bing Search API



Bing Search API

The Bing Search API enables developers to embed search results in applications or websites using XML or JSON. Access web, image, news and video results as well as related searches and spelling suggestions.

4,992 Transactions remaining

Primary Account Key OAzSJAvmayWtxYD96ojoSdUtu1v+DRY2bi0vd78mgpQ=

Download Options

Excel (CSV)

PowerPivot 2010

PowerPivot 2013

URL for current expressed query:

```
https://api.datamarket.azure.com/Bing/Search/v1/Web?Query=%27xbox%27
```

Displaying 50 rows Page 1 ►

APPLY CLEAR

Required parameters:

Query: xbox

Value like: : xbox

Optional parameters:

Options:

Value like: : EnableHighlighting

WebSearchOptions:

Value like: : DisableQueryAlterations

Market: (blank)

Adult: (blank)

Latitude:

Value like: : 47.603450

Longitude:

Value like: : -122.329696

WebFileType: (blank)

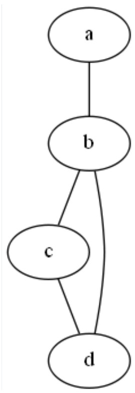
ID	Title	Description	DisplayUrl	Url
3ca1e252-614d-4ff7-a47c-002ba0d21972	Xbox - Xbox.com	Visit Xbox.com for information about Xbox One, Xbox 360, Xbox Live, games, TV, movies, music, sports and more.	www.xbox.com	http://www.xbox.com/
37f7c85a-f93e-456d-9dff-68dab24291a7	Xbox - Wikipedia, the free encyclopedia	The Xbox is a video gaming brand created by Microsoft. It includes a series of video game consoles developed by Microsoft, with consoles in the sixth to ...	en.wikipedia.org/wiki/Xbox	http://en.wikipedia.org/wiki/Xbox
981f9e12-ddad-4140-b29c-1d864aeb15c2	Xbox Live Gold - Xbox.com	Xbox Live: 48 million members and growing. Xbox Live Gold is your ticket to the most exciting social entertainment network in the world on both Xbox 360 and Xbox One.	www.xbox.com/live	http://www.xbox.com/live/
28c6adb0-93ae-4715-907b-18ecafbc50c2	Home - Xbox.com	Be a part of Xbox Social. Whether you're on your computer, your phone or your console, Xbox Social is your connection to the Xbox LIVE community.	https://live.xbox.com	https://live.xbox.com/Home
8b15ceac-898c-4dae-9983-899b82f8d83a	Amazon.com: Xbox - More Systems: Video Games: Games, Hardware & More	Online shopping for Xbox More Systems from a great selection of Video Games; Games, Hardware & more at everyday low prices.	www.amazon.com/Xbox-Games/b?ie=UTF8&node=537504	http://www.amazon.com/Xbox-Games/b?ie=UTF8&node=537504
c81160ba-6292-4cfe-b58e-7ce1aa91eb69	Welcome to Xbox Support - Xbox.com	Xbox Support offers help and support on Xbox 360, Xbox Live, Kinect, and more. Join the Xbox Support community to get expert advice and customer service.	support.xbox.com/en-us	http://support.xbox.com/en-us/
d75cdcb4-a80c-46e9-b379-3abe68bc9119	xbox | eBay - Electronics, Cars, Fashion, Collectibles, Coupons ...	Find great deals on eBay for xbox and xbox 360. Shop with confidence.	www.ebay.com/sch/i.html?_nkw=xbox	http://www.ebay.com/sch/i.html?_nkw=xbox
dd3286e9-dba5-43c6-9f2f-1f64bb93e13e	Xbox Games - Xbox Accessories | GameStop	Buy Xbox games and accessories at GameStop. Shop our huge selection of Xbox games and accessories.	www.gamestop.com/browse/xbox?nav=1389	http://www.gamestop.com/browse/xbox?nav=1389
a0c380e4-99eb-4e37-8a5b-1b0c052b365b	Xbox 360 Games - EA	Electronic Arts offers the most complete catalog of video games for the Xbox 360 platform including Mass Effect 3 and Battlefield 3.	www.ea.com/xbox-360	http://www.ea.com/xbox-360

- JSON or XML
- Requires API key registration, authentication

Handy packages:
pydot, GraphViz, and itertools

Graph (network) creation with `pydot` Graph visualization with Graphviz

```
graph = pydot.Dot(graph_type='graph', charset="utf8")
edge = pydot.Edge('a', 'b')
graph.add_edge(edge)
edge = pydot.Edge('b', 'c')
graph.add_edge(edge)
edge = pydot.Edge('b', 'd')
graph.add_edge(edge)
edge = pydot.Edge('c', 'd')
graph.add_edge(edge)
graph.write(graph_output_name)
```



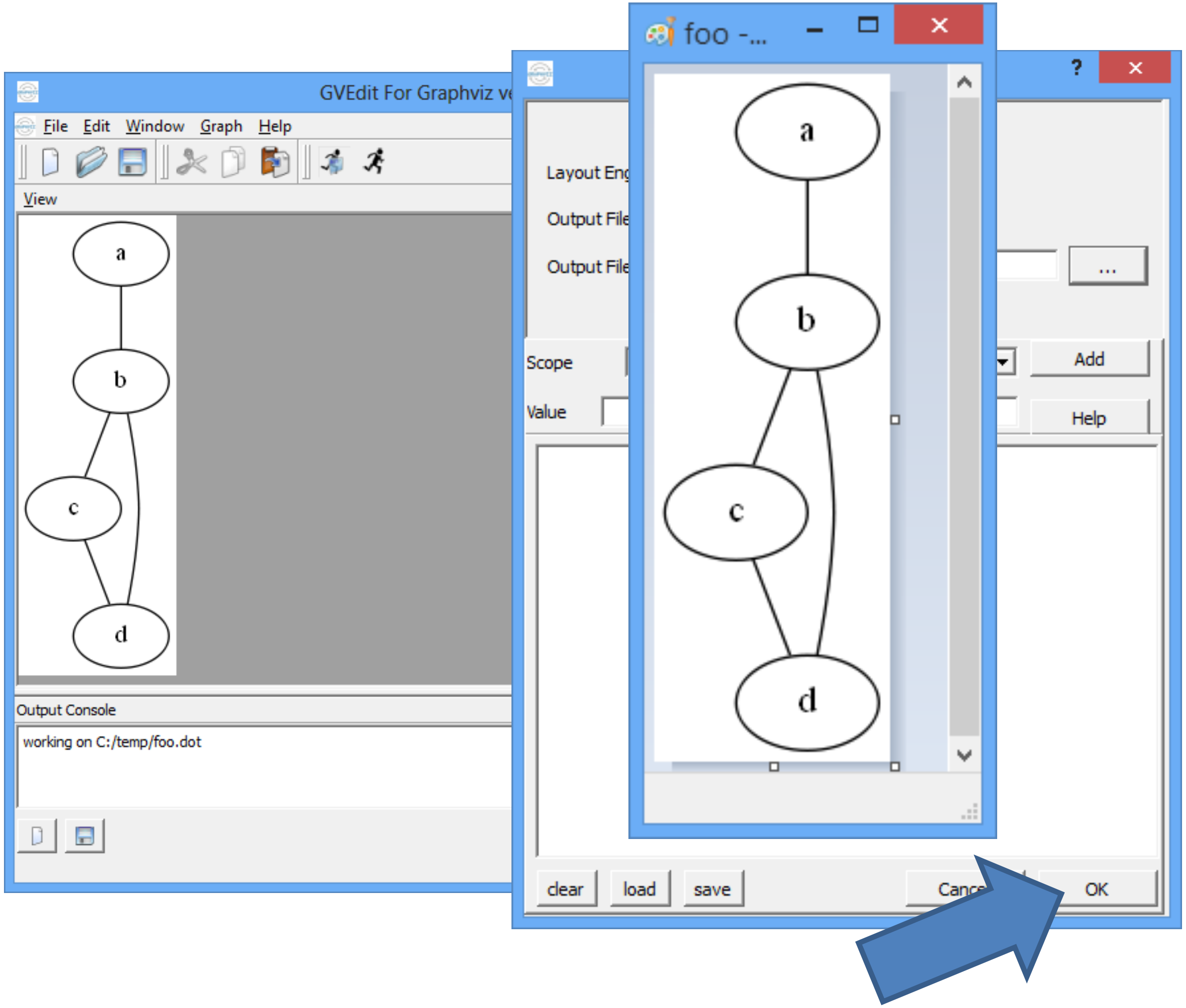
Result

.dot file

```
graph G {
charset="utf8"
a -- b;
b -- c;
b -- d;
c -- d;
}
```

Reference: https://code.google.com/p/pydot/

Use gvedit from GraphViz to load the .dot file and generate an image



working on C:/temp/foo.dot

(c) 2016 Chris Teplovs

The `itertools` module

```
>>> print(list(itertools.permutations('ABCDE', r=3)))
[('A', 'B', 'C'), ('A', 'B', 'D'), ('A', 'B', 'E'), ('A', 'C', 'B'), ('A', 'C', 'D'),
 ('A', 'C', 'E'), ('A', 'D', 'B'), ('A', 'D', 'C'), ('A', 'D', 'E'), ('A', 'E', 'B'),
 ('A', 'E', 'C'), ('A', 'E', 'D'), ('B', 'A', 'C'), ('B', 'A', 'D'), ('B', 'A', 'E'),
 ('B', 'C', 'A'), ('B', 'C', 'D'), ('B', 'C', 'E'), ('B', 'D', 'A'), ('B', 'D', 'C'),
 ('B', 'D', 'E'), ('B', 'E', 'A'), ('B', 'E', 'C'), ('B', 'E', 'D'), ('C', 'A', 'B'),
 ('C', 'A', 'D'), ('C', 'A', 'E'), ('C', 'B', 'A'), ('C', 'B', 'D'), ('C', 'B', 'E'),
 ('C', 'D', 'A'), ('C', 'D', 'B'), ('C', 'D', 'E'), ('C', 'E', 'A'), ('C', 'E', 'B'),
 ('C', 'E', 'D'), ('D', 'A', 'B'), ('D', 'A', 'C'), ('D', 'A', 'E'), ('D', 'B', 'A'),
 ('D', 'B', 'C'), ('D', 'B', 'E'), ('D', 'C', 'A'), ('D', 'C', 'B'), ('D', 'C', 'E'),
 ('D', 'E', 'A'), ('D', 'E', 'B'), ('D', 'E', 'C'), ('E', 'A', 'B'), ('E', 'A', 'C'),
 ('E', 'A', 'D'), ('E', 'B', 'A'), ('E', 'B', 'C'), ('E', 'B', 'D'), ('E', 'C', 'A'),
 ('E', 'C', 'B'), ('E', 'C', 'D'), ('E', 'D', 'A'), ('E', 'D', 'B'), ('E', 'D', 'C')]
```

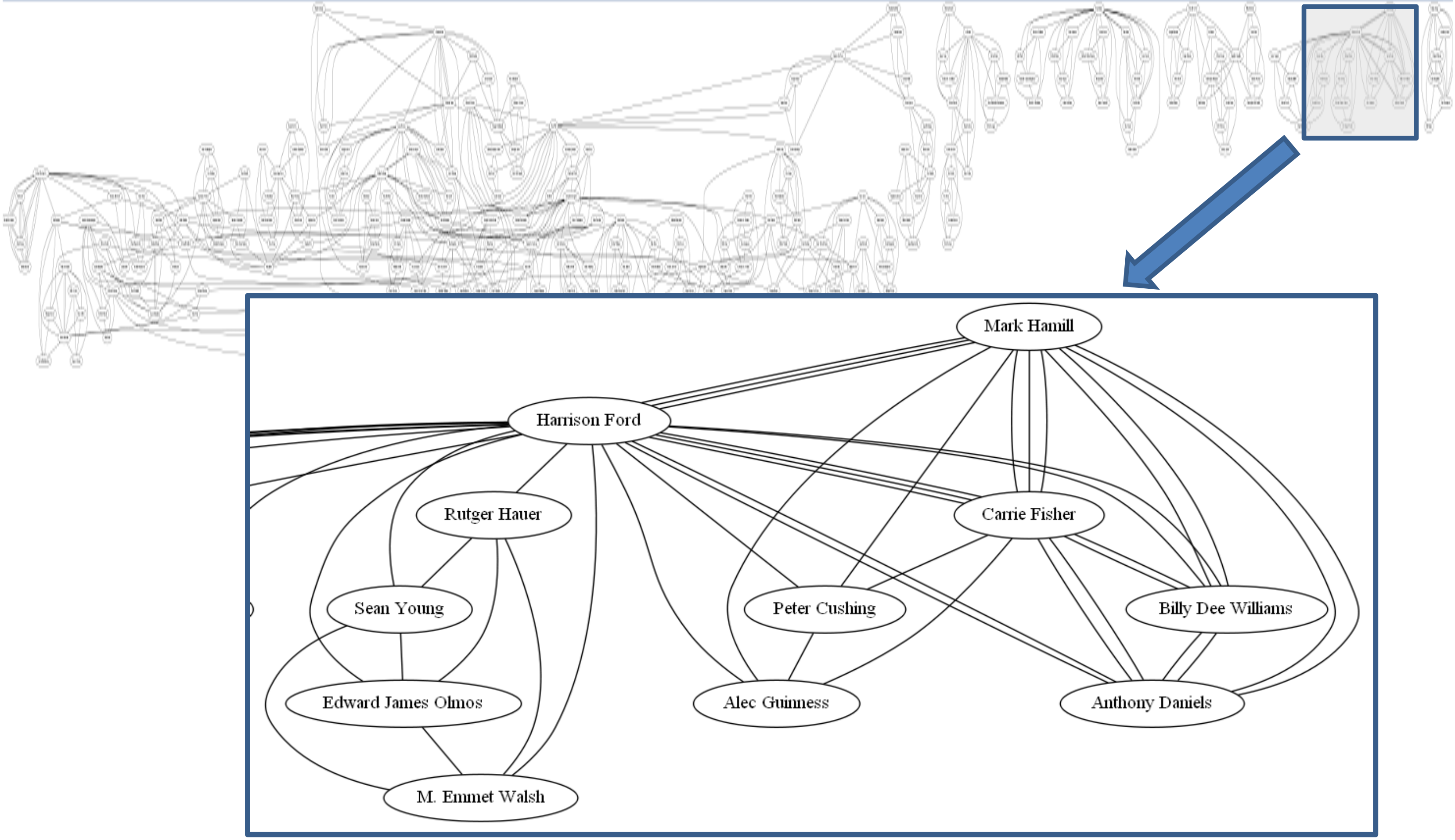
```
>>> import itertools
>>> actors = ['Humphrey Bogart', 'Ingrid Bergman', 'Claude Rains']
```

How do you get a list of all pairs of actors who appeared together?

```
>>> print(list(itertools.combinations(actors, 2)))
[('Humphrey Bogart', 'Ingrid Bergman'), ('Humphrey Bogart', 'Claude Rains'), ('Ingrid
Bergman', 'Claude Rains')]
```

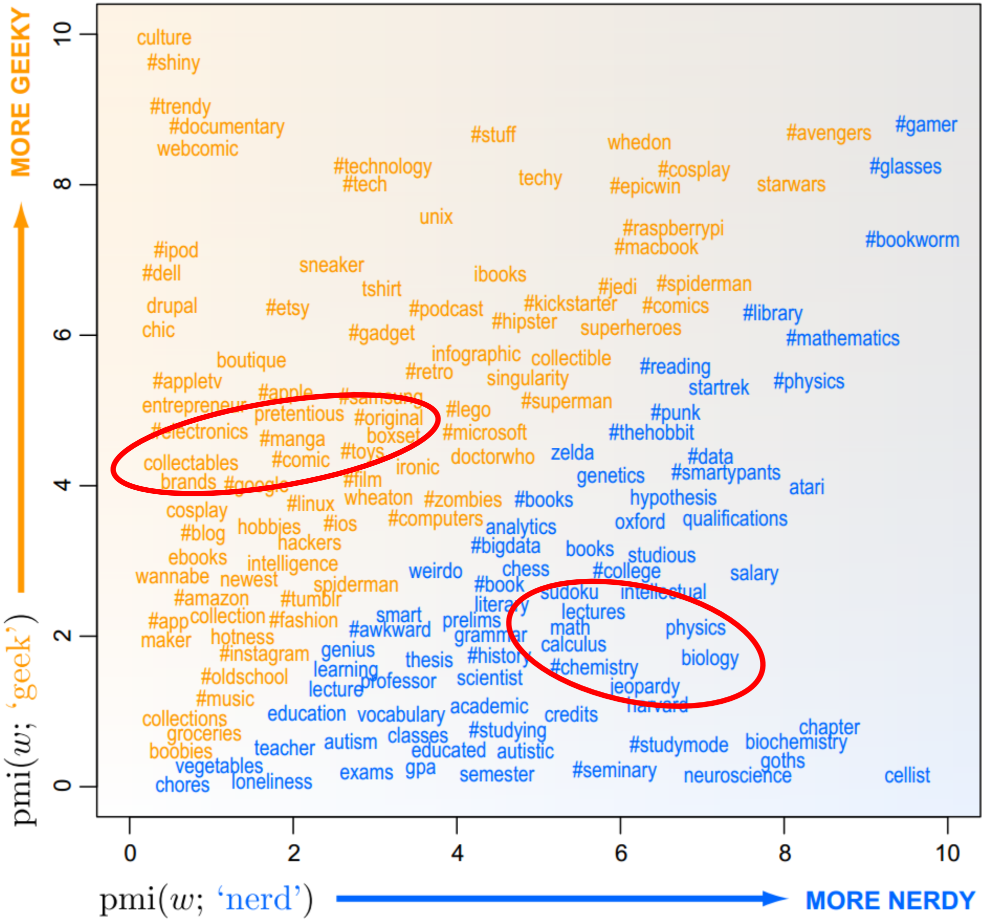
Reference: http://docs.python.org/2/library/itertools

Homework 3:
Internet Movie Database + GraphViz



Future sneak peek: Geeks vs Nerds!

Estimating associations in large-scale data



Source: http://slackprop.wordpress.com/2013/06/03/on-geek-versus-nerd/

What you should know

- The basics of how HTTP works and how to fetch Web content using urllib2
- How to take an HTML or XML response from urllib2 and parse it using BeautifulSoup
- JSON and how to read/write it
- Become familiar with what's available via Web services and REST APIs
- The basics of graph visualization with pydot

Resources

- Chapter 12, 13, Severance
- HTTP tutorial
  - http://www.jmarshall.com/easy/http/
- Urllib2
  - http://docs.python.org/2/howto/urllib2.html
  - Review of HTML elements
    - http://www.w3schools.com/html/html_elements.asp
- BeautifulSoup
  - http://www.crummy.com/software/BeautifulSoup/bs4/doc/
- json module tutorial:
  - http://www.doughellmann.com/PyMOTW/json/
- Graphviz is open source graph visualization software. http://www.graphviz.org/
- pydot is a Python interface to Graphviz's Dot language. http://code.google.com/p/pydot/

Lab 3: BeautifulSoup and json

Extra resources

ET Tutorial

- http://docs.python.org/2/library/xml.etree.elementtree.html

APIs vs Web scraping

- Should you 'scrape' HTML pages to extract information, or go through an API?
- This article has a good discussion:

  http://lethain.com/an-introduction-to-compassionate-screenscraping/