STATS 700-002
Data analysis using Python

Lecture 2: Structured Data from the Web
Reminder

Homework 1 is available and due Wednesday, November 15th by 11:59pm

Start yesterday! If you run into trouble, please email me or your GSI and **come to office hours**.

If you are having issues with installation, compilation, etc (i.e., problems not directly related to the homework assignment), post about it on Canvas!
Lots of interesting data resides on websites

**HTML**: HyperText Markup Language
- Specifies basically everything you see on the Internet

**XML**: EXtensible Markup Language
- Designed to be an easier way for storing data, similar framework to HTML

**JSON**: JavaScript Object Notation
- Designed to be a saner version of XML

**SQL**: Structured Query Language
- IBM-designed language for interacting with databases

**APIs**: Application Programming Interface
- Allow interaction with website functionality (e.g., Google maps)
Three Aspects of Data on the Web

**Location:** URL (Uniform Resource Locator), IP address
- Specifies location of a computer on a network

**Protocol:** HTTP, HTTPS, FTP, SMTP
- Specifies how computers on a network should communicate with one another

**Content:** HTML (for example)
- Contains actual information, e.g., tells browser what to display and how

We’ll mostly be concerned with website content. Wikipedia has good entries on network protocols. Classic textbook is *Computer Networks* by A. S. Tanenbaum.
Client-server model

HTTP is

**Connectionless:** after a request is made, the client disconnects and waits

**Media agnostic:** any kind of data can be sent over HTTP

**Stateless:** server and client “forget about each other” after a request
Anatomy of a URL

**Protocol**
- Specifies how the client (i.e., your browser) will communicate with server.

**Hostname**
- Gives a human-readable name to location of the server on the network.

**Filename**
- Names a specific file on the server that the client wishes to access.

**Note:** Often the extension of the file will indicate what type it is (e.g., html, txt, pdf, etc), but not always. Often, must determine the type of the file based on its contents. This can almost always be done automatically.
Accessing websites in Python: urllib2

Python library for opening URLs and interacting with websites
  ●  https://docs.python.org/2/library/urllib2.html#

Software development community is moving towards requests
  ●  https://requests.readthedocs.io/en/master/
  ●  a bit over-powered for what we want to do, but feel free to use it in HWs

Note: using urllib2 in Python 3 may incur a couple of hiccups because the module was split into submodules. See documentation for details. Let me know if you run into trouble and I’ll do my best to help!
Using urllib2

urllib2.urlopen() : opens the given url, returns a file-like object

```
1 import urllib2
2 response = urllib2.urlopen('https://umich.edu/~klevin')
```

Three basic methods
- `getcode()` : return the HTTP status code of the response
- `geturl()` : return URL of the resource retrieved (e.g., see if redirected)
- `info()` : return meta-information from the page, such as headers
getcode() 

HTTP includes success/error status codes  
**Ex:** 200 OK, 301 Moved Permanently, 404 Not Found, 503 Service Unavailable  

```
import urllib2
response = urllib2.urlopen('https://umich.edu/~klevin
response.getcode()
```

200

```
import urllib2
response = urllib2.urlopen('https://umich.edu/~klevin/nonsense.html
response.getcode()
```

HTTPError: HTTP Error 404: Not Found

**Note:** I cropped a bunch of error information, which will normally be useful!
geturl()

```python
import urllib2
response = urllib2.urlopen('https://umich.edu/~klevin/
response.geturl()

'http://www-personal.umich.edu/~klevin/'
```

Different URLs, owing to automatic redirect.
info()

Returns a dictionary-like object with information about the page you retrieved.

```
import urllib2
response = urllib2.urlopen('https://umich.edu/~klevin')
print response.info()
```

Set-Cookie: PERSONALCOOKIE=R3382578549; path=/
Date: Thu, 28 Sep 2017 18:21:14 GMT
Server: Apache
Accept-Ranges: bytes
Content-Length: 3952
Connection: close
Content-Type: text/html; charset=utf-8

Very useful, for example, when you aren’t sure of content type or character set, though nowadays most of those things are handled automatically by parsers
HTML Crash Course

HTML is a markup language.

<tag_name attr1="value" attr2="differentValue">String contents</tag_name>

Basic unit: **tag**
   (usually) a start and end tag, like <p>contents</p>

Contents of a tag may contain more tags:
   <head><title>The Title</title></head>
   <p>This tag links to <a href="google.com">Google</a></p>
Tags have attributes, which are specified after the tag name, in (key,value) pairs of the form key="val"

Example: hyperlink tags

```
<a href="umich.edu/~klevin">My personal webpage</a>
```

Corresponds to a link to **My personal webpage**.

The href attribute specifies where the hyperlink should point.
HTML Crash Course: Recap

Of special interest in your homework: HTML tables

https://www.w3schools.com/html/html_tables.asp
https://www.w3.org/TR/html401/struct/tables.html
Okay, back to `urllib2`

`urllib2` reads a webpage (full of HTML) and returns a “response” object

The response object can be treated like a file:

```python
import urllib2
def read_webpage(url):
    response = urllib2.urlopen(url)
    return response.read()  
```
Okay, back to `urllib2`

`urllib2` reads a webpage (full of HTML) and returns a “response” object.

The response object can be treated like a file:

```python
import urllib2
response = urllib2.urlopen('https://wikipedia.org')
response.read()
```

What a mess! How am I supposed to do anything with this?!
Parsing HTML/XML in Python: beautifulsoup

Python library for working with HTML/XML data
- Builds nice tree representation of markup data...
- ...and provides tools for working with that tree

Documentation: https://www.crummy.com/software/BeautifulSoup/bs4/doc/

Good tutorial: http://www.pythonforbeginners.com/python-on-the-web/beautifulsoup-4-python/

Installation: “pip install beautifulsoup” or follow instructions for conda or...
Parsing HTML/XML in Python: beautifulsoup

BeautifulSoup turns HTML mess into a (sometimes complex) tree

Four basic kinds of objects:

- **Tag**: corresponds to HTML tags
  - `<[name] [attr]="xyz">[string]<[/[name]> )
  - Two important attributes: tag.name, tag.string
  - Also has dictionary-like structure for accessing attributes
- **NavigableString**: special kind of string for use in bs4
- **BeautifulSoup**: represents the HTML document itself
- **Comment**: special kind of NavigableString for HTML comments
Example (from the BeautifulSoup docs)

```python
html_doc = ""
<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>""

from bs4 import BeautifulSoup
parsed = BeautifulSoup(html_doc, 'html.parser')
```

Follow along at home: [https://www.crummy.com/software/BeautifulSoup/bs4/doc/#quick-start](https://www.crummy.com/software/BeautifulSoup/bs4/doc/#quick-start)
print(parsed.prettify())

<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 class="sister" href="http://example.com/elsie" id="link1">
Elsie
</a>,
<a class="sister" href="http://example.com/lacie" id="link2">
Lacie
</a>
and
<a class="sister" href="http://example.com/tillie" id="link3">
Tillie
</a>;

and they lived at the bottom of a well.
</p>
</body>
</html>
BeautifulSoup allows navigation of the HTML tags

Finds all the tags that have the name `a`, which is the HTML tag for a link.

The `href` attribute in a tag with name `a` contains the actual url for use in the link.
Once upon a time there were three little sisters; and their names were Elsie, Lacie, and Tillie; and they all lived at the bottom of a well.
The Dormouse’s story

Once upon a time there were three little sisters; and their names were Elsie, Lacie and Tillie; and they all lived at the bottom of a well.

Question: what are the attributes of this node in the tree? That is, what are the attributes of this tag?
Navigating the HTML tree

Tag name gets the first tag of that type in the tree.

If a tag’s child is a string, access it with `tag.string`.

Can go down the tree by asking for tags of tags of...
Navigating the HTML tree

Access a list of children of a tag with `.contents`

Or get the same information in a Python iterator with `.children`

Recurse down the whole tree with `.descendants`
Navigating the HTML tree

The tree structure means that every tag has a parent (except the “root” tag, which has parent “None”).

Access a tag’s parent tag with `.parent`

Get the whole chain of parents back to the root with `.parents`

Move “left and right” in the tree with `.previous_sibling` and `.next_sibling`
Searching the tree: `find_all` and related methods

1. `parsed = BeautifulSoup(html_doc, 'html.parser')`

2. `parsed.find_all('p')`  
   Finds all tags with name `p`

3. `parsed.find_all(['a', 'b'])`  
   Finds all tags with names matching either `a` or `b`

4. `import re`

5. `parsed.find_all(re.compile(r'^b'))`  
   Finds all tags whose names match the given regex.
More about `find_all`

```python
8 def has_class_but_no_id(tag):
9     return tag.has_attr('class') and not tag.has_attr('id')
10    parsed.find_all(has_class_but_no_id)
```

Note: by default, `find_all` recurses down the whole tree, but you can have it only search the immediate children of the tag by passing the flag `recursive=False`.

See https://www.crummy.com/software/BeautifulSoup/bs4/doc/#find-all for more.
This `<p>` tag contains a full sentence, but some parts of that sentence are links, so `p.string` fails. What do I do if I want to get the full string without the links?

**Note:** common cause of bugs/errors in `BeautifulSoup` is trying to access `tag.string` when it doesn’t exist!
A note on attributes

HTML attributes and Python attributes are **different things!**

But in BeautifulSoup they collide in a weird way

BeautifulSoup tags have their HTML attributes accessible like a dictionary:

```
import bs4

shortdoc=""

<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>;</p

and they lived at the bottom of a well.</p>


pshort = BeautifulSoup(shortdoc, 'html.parser')

print pshort.p['class']
```

[u'story']

BeautifulSoup tags have their *children* accessible as Python attributes:

```
print pshort.p.a
```

```<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>```
XML - eXtensible Markup Language, .xml


**Core idea:** separate data from its presentation

Note that HTML *doesn’t* do this-- the HTML for the webpage is the data

But XML is tag-based, very similar to HTML

BeautifulSoup will parse XML

https://www.crummy.com/software/BeautifulSoup/bs4/doc/#installing-a-parser

We won’t talk much about XML, because it’s falling out of favor, replaced by...
JSON - JavaScript Object Notation

Commonly used by website APIs

Basic building blocks:
  ● attribute–value pairs
  ● array data

Example (right) from wikipedia:
Possible JSON representation of a person

```json
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    },
    {
      "type": "mobile",
      "number": "123 456-7890"
    }
  ],
  "children": [],
  "spouse": null
}
```
Python json module

```python
import json

json_string = ' {
  "first_name": "Claude", 
  "last_name": "Shannon",
  "alma_mater": "University of Michigan"
}

parsed_json = json.loads(json_string)

parsed_json

{u'alma_mater': u'University of Michigan',
 u'first_name': u'Claude',
 u'last_name': u'Shannon'}

json.dumps(parsed_json)

' {
  "alma_mater": "University of Michigan", 
  "first_name": "Claude", 
  "last_name": "Shannon"
}'}

JSON string encoding information about information theorist Claude Shannon

**json.loads** parses a string and returns a JSON object.

**json.dumps** turns a JSON object back into a string.
Python `json` module

```python
parsed_json = {
    'alma_mater': 'University of Michigan',
    'first_name': 'Claude',
    'last_name': 'Shannon'}
```

```python
parsed_json['alma_mater']
'u'University of Michigan'
```

```python
parsed_json['first_name']
'u'Claude'
```

```python
parsed_json['middle_name']
```

```

JSON object returned by `json.loads` acts just like a Python dictionary.

```

```

Traceback (most recent call last)
<ipython-input-440-a100eb80552a> in <module>()
----> 1 parsed_json['middle_name']

KeyError: 'middle_name'
```
JSON objects can have very complicated structure
JSON objects can have very complicated structure

```json
complex_json_string="{"id": "0001", "type": "donut", "name": "Cake", "ppu": 0.55, "batters": {
    "batter": [
        { "id": "1001", "type": "Regular" },
        { "id": "1002", "type": "Chocolate" },
        { "id": "1003", "type": "Blueberry" },
        { "id": "1004", "type": "Devil's Food" }
    ],
    "topping": [
        { "id": "5001", "type": "None" },
        { "id": "5002", "type": "Glazed" },
        { "id": "5005", "type": "Sugar" },
        { "id": "5007", "type": "Powdered Sugar" },
        { "id": "5006", "type": "Chocolate with Sprinkles" },
        { "id": "5003", "type": "Chocolate" },
        { "id": "5004", "type": "Maple" }
    ]
}"
```

This can get out of hand quickly, if you’re trying to work with large collections of data. For an application like that, you are better off using a database, about which we’ll learn in our next lecture.
Readings (for this lecture)

Required:
   Severance Chapter 12 (HTTP, HTML), Chapter 13 (XML, JSON)
   BeautifulSoup documentation (just Quick Start)
      https://www.crummy.com/software/BeautifulSoup/bs4/doc/

Recommended:
   BeautifulSoup documentation (everything up to sections about CSS)
      https://www.crummy.com/software/BeautifulSoup/bs4/doc/
Readings (for next lecture)

Required:
- Oracle relational databases overview (only the overview!)
  https://docs.oracle.com/javase/tutorial/jdbc/overview/database.html
- First section of Python sqlite3 documentation
  https://docs.python.org/2/library/sqlite3.html
  https://docs.python.org/3/library/sqlite3.html

Recommended:
- w3schools SQL tutorial: https://www.w3schools.com/sql/