Sample solutions

Since Notebook 6 is an optional assignment, we are releasing these sample solutions concurrently.

part0 (Score: 0.0 / 0.0)

1. Test cell (Score: 0.0 / 0.0)
2. Test cell (Score: 0.0 / 0.0)
3. Test cell (Score: 0.0 / 0.0)
4. Test cell (Score: 0.0 / 0.0)

Important note! Before you turn in this lab notebook, make sure everything runs as expected:

- First, **restart the kernel** -- in the menubar, select Kernel→Restart.
- Then **run all cells** -- in the menubar, select Cell→Run All.

Make sure you fill in any place that says `YOUR CODE HERE` or "YOUR ANSWER HERE."

Part 0: Mining the web

Perhaps the richest source of openly available data today is the Web (http://www.computerhistory.org/revolution/networking/19/314)! In this lab, you'll explore some of the basic programming tools you need to scrape web data.

Warnings.

1. If you are using one of the cloud-based Jupyter installations to run this notebook, such as Microsoft Azure Notebooks (https://notebooks.azure.com) or Vocareum (https://vocareum.org), it's likely you will encounter problems due to restrictions on access to remote servers.
2. Even if you are using a home or local installation of Jupyter, you may encounter problems if you attempt to access a site too many times or too rapidly. That can happen if your internet service provider (ISP) or the target website detect your accesses as "unusual" and reject them. It's easy to imagine accidentally writing an infinite loop that tries to access a page and being seen from the other side as a malicious program. :)

The Requests module

Python's Requests module (http://requests.readthedocs.io/en/latest/user/quickstart/) to download a web page.

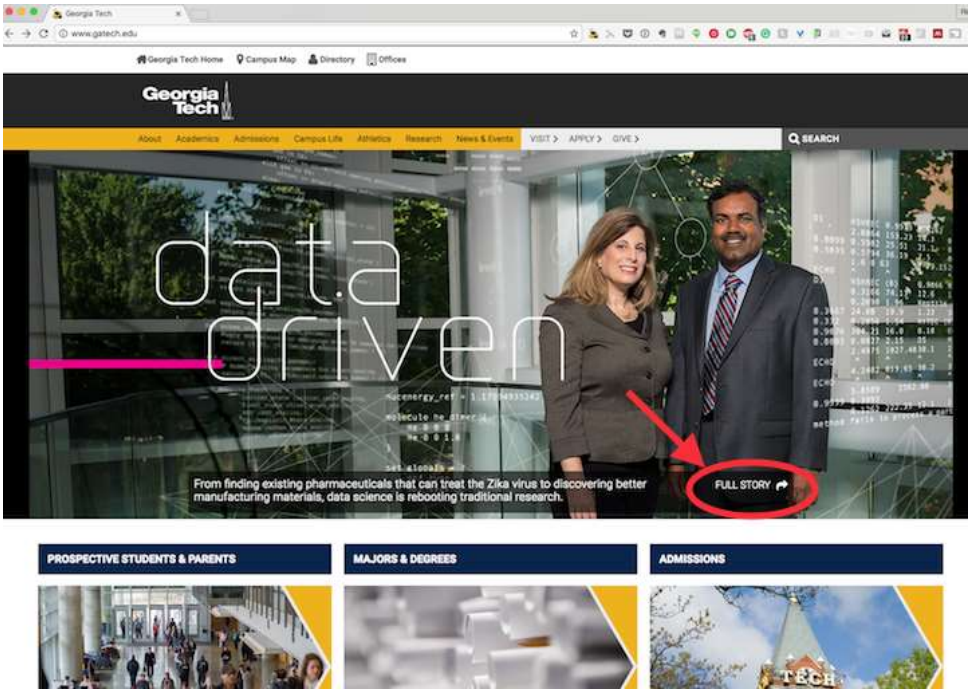
For instance, here is a code fragment to download the Georgia Tech (http://www.gatech.edu) home page and print the first 250 characters. You might also want to view the source (http://www.computerhope.com/issues/ch000746.htm) of Georgia Tech's home page to get a nicely formatted view, and compare its output to what you see above.

In [1]:
```
import requests

response = requests.get('http://www.gatech.edu/')
webpage = response.text  # or response.content for raw bytes

print(webpage[0:250]) # Prints the first hundred characters only
```

```
<!DOCTYPE html>
<html lang="en" dir="ltr"
  xmlns:content="http://purl.org/rss/1.0/modules/content/"
  xmlns:dc="http://purl.org/dc/terms/"
  xmlns:foaf="http://xmlns.com/foaf/0.1/"
  xmlns:og="http://ogp.me/ns#"
  xmlns:rdfs="http://www.w3.org/2000
```

Exercise 1. Given the contents of the GT home page as above, write a function that returns a list of links (URLs) of the "top stories" on the page.

For instance, on Friday, September 9, 2016, here was the front page:



The top stories cycle through in the large image placeholder shown above. We want your function to return the list of URLs behind each of the "Full Story" links, highlighted in red. If no URLs can be found, the function should return an empty list.

In [2]: Student's answer (Top)

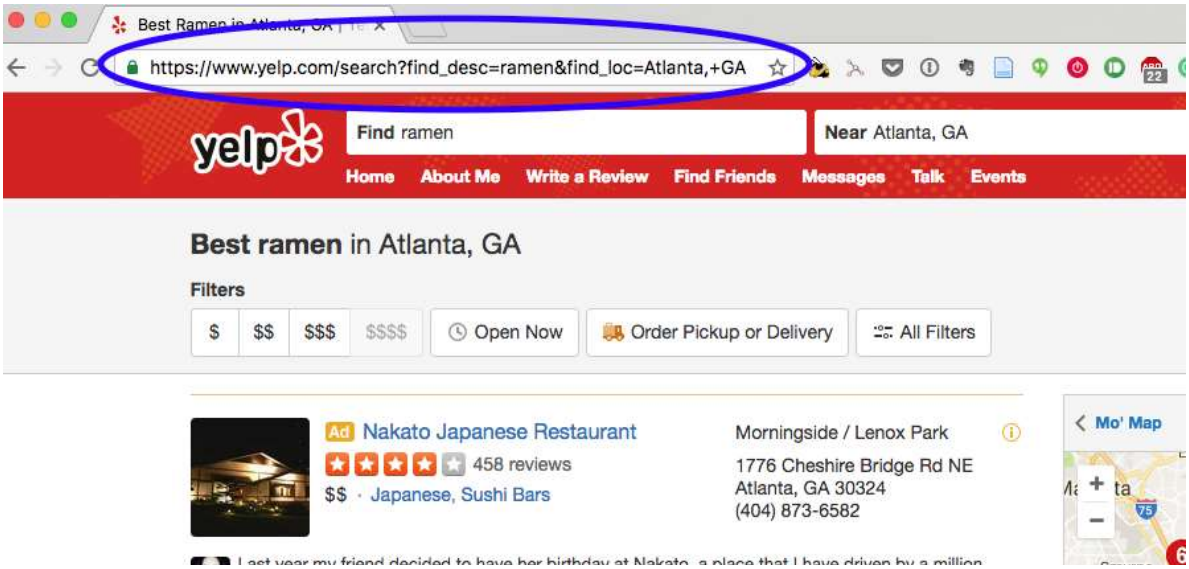
```
import re # Maybe you want to use a regular expression?

def get_gt_top_stories(webpage_text):
    """Given the HTML text for the GT front page, returns a list
    of the URLs of the top stories or an empty list if none are
    found.
    """
    pattern = '''<a class="slide-link" href="(?P<url>[^"]+)"'''
    return re.findall(pattern, webpage_text)
```

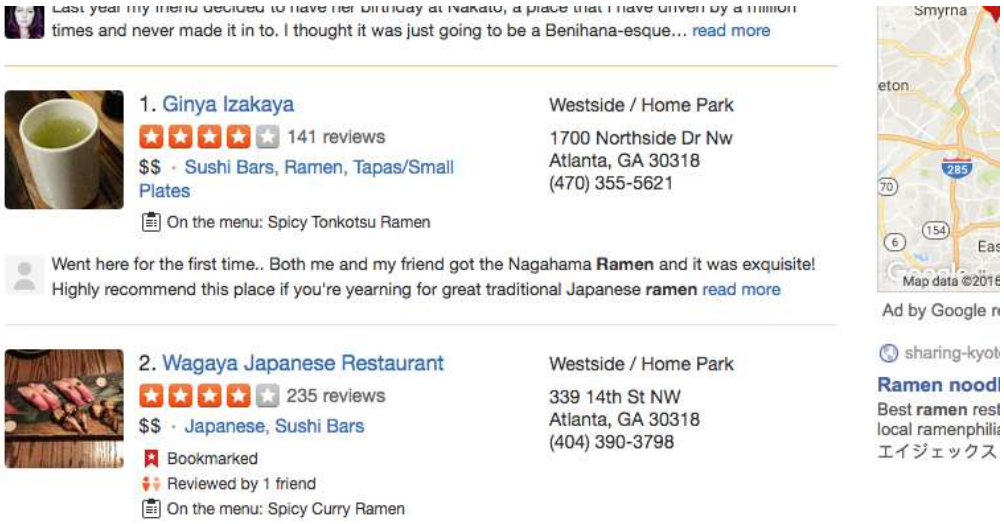
In [3]:
```
top_stories = get_gt_top_stories(webpage)
print("Links to GT's top stories:", top_stories)
```

```
Links to GT's top stories: ['http://development.gatech.edu/care-and-feeding-student-startups',
'http://www.rh.gatech.edu/news/595812/engineering-research-center-will-help-expand-use-therapies
-based-living-cells']
```

A more complex example

Go to Yelp! (http://www.yelp.com) and look up `ramen` in `Atlanta, GA`. Take note of the URL:





This URL encodes what is known as an *HTTP "get"* method (or request). It basically means a URL with two parts: a *command* followed by one or more *arguments*. In this case, the command is everything up to and including the word `search`; the arguments are the rest, where individual arguments are separated by the & or #.

> "HTTP" stands for "HyperText Transport Protocol," which is a standardized set of communication protocols that allow *web clients*, like your web browser or your Python program, to communicate with *web servers*.

In this next example, let's see how to build a "get request" with the `requests` module. It's pretty easy!

In [4]:
```
url_command = 'http://yelp.com/search'
url_args = {'find_desc': "ramen",
            'find_loc': "atlanta, ga"}
response = requests.get (url_command, params=url_args)

print ("==> Downloading from: '%s'" % response.url) # confirm URL
print ("\n==> Excerpt from this URL:\n\n%s\n" % response.text[0:100])
```

```
==> Downloading from: 'https://www.yelp.com/search?find_desc=ramen&find_loc=atlanta%2C+ga'

==> Excerpt from this URL:

<!DOCTYPE HTML>

<!--[if lt IE 7 ]> <html xmlns:fb="http://www.facebook.com/2008/fbml" class="ie6 ie
```

Exercise 2. Given a search topic, location, and a rank k, return the name of the k-th item of a Yelp! search. If there is no k-th item, return `None`.

> The demo query above only gives you a website with the top 10 items, meaning you could only use it for $k \leq 10$. Figure out how to modify it to solve the problem when $k > 10$.

In [5]:
Student's answer (Top)

```
def find_yelp_item (topic, location, k):
    """Returns the k-th suggested item from Yelp! in Atlanta for the given topic."""
    import re
    if k < 1: return None

    # Download page
    url_command = 'http://yelp.com/search'
    url_args = {'find_desc': topic,
                'find_loc': location,
                'start': k-1
               }

    response = requests.get (url_command, params=url_args)
    if not response: return None

    # Split page into lines
    lines = response.text.split ('\n')

    # Look for the line containing the name of the k-th item
    item_pattern = re.compile ('<span class="indexed-biz-name">{}\..*<span >(?P<item_name>.*)
</span></a>'.format (k))
    for l in lines:
        item_match = item_pattern.search (l)
```

```
        if item_match:
            return item_match.group ('item_name')

    # No matches, evidently
    return None
```

In [6]:

Grade cell: `yelp_atl__test1` Score: 0.0 / 0.0 (Top)

```
assert find_yelp_item('fried chicken', 'Atlanta, GA', -1) is None # Tests an invalid value for
 'k'
```

Search queries on Yelp! don't always return the same answers, since the site is always changing! Also, your results might not match a query you do via your web browser (*why not?*). As such, you should manually check your answers.

In [7]:

Grade cell: `yelp_atl__test2` Score: 0.0 / 0.0 (Top)

```
item = find_yelp_item ('fried chicken', 'Atlanta, GA', 1)
print (item)

# The most likely answer on September 19, 2017:
#assert item in ['Gus’s World Famous <span class="highlighted">Fried</span> <span class="highl
ighted">Chicken</span>',
#                'Gus’s World Famous Fried Chicken']
```

```
Gus’s World Famous <span class="highlighted">Fried</span> <span class="highlighted">Chicken</spa
n>
```

In [8]:

Grade cell: `yelp_atl__test3` Score: 0.0 / 0.0 (Top)

```
item = find_yelp_item ('fried chicken', 'Atlanta, GA', 5)
print (item)

# The most likely answer on September 19, 2017:
#assert item == 'Richards’ Southern Fried'
```

```
Colonnade Restaurant
```

In [9]:

Grade cell: `yelp_atl__test4` Score: 0.0 / 0.0 (Top)

```
item = find_yelp_item('fried chicken', 'Atlanta, GA', 17)
print(item)

# Most likely correct answer as of September 19, 2017:
#assert item == 'Sway'
```

```
Buttermilk Kitchen
```

part1 (Score: 0.0 / 0.0)

1. Written response (Score: 0.0 / 0.0)
2. Test cell (Score: 0.0 / 0.0)
3. Test cell (Score: 0.0 / 0.0)

Important note! Before you turn in this lab notebook, make sure everything runs as expected:

- First, **restart the kernel** -- in the menubar, select Kernel→Restart.
- Then **run all cells** -- in the menubar, select Cell→Run All.

Make sure you fill in any place that says `YOUR CODE HERE` or "YOUR ANSWER HERE."

Part 1: Tools to process HTML

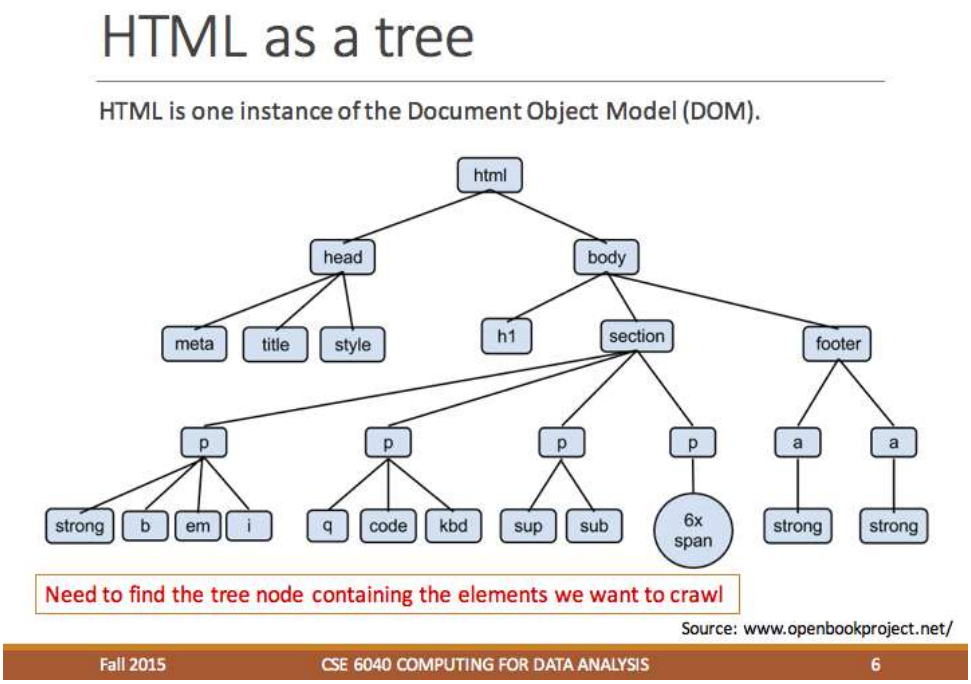
In Part 0, you downloaded real web pages and manipulated them using "conventional" string processing tools, like `str()` functions or regular expressions ().

However, web pages are stored in HTML (hypertext markup language ()), which is a highly structured format. As such, it makes sense to use specialized tools to understand and process its structure. That's the subject of this notebook.

Parsing HTML: The Beautiful Soup module

One such package to help process HTML is Beautiful Soup (https://www.crummy.com/software/BeautifulSoup/). The following is a quick tutorial on how to use it.

Any HTML document may be modeled as an object in computer science known as a tree (https://en.wikipedia.org/wiki/Tree_(data_structure)):



There are different ways to define trees, but for our purposes, the following will be sufficient.

Consider a tree is a collection of *nodes*, which are the labeled boxes in the figure, and *edges*, which are the line segments connecting nodes, with the following special structure.

- The node at the top is called the *root*. Here, the root is labeled `html` and abstractly represents the entire HTML document.
- Regard each edge as always "pointing" from the node at its top end to the node at its bottom end. For any edge, the node at its top end is the *parent* and the node at the bottom end is a *child*. Like real families, a parent can be a child. For example, the node labeled `head` is the child of `html` and the parent of `meta`, `title`, and `style`.
- The *descendant* of a node x is any node y for which there is a path from x going down to y. For example, the node labeled `6x span` is a descendant of the node `body`. All nodes are descendants of the root.
- Any node with *no* descendants is a *leaf*.
- Any node that is neither a root nor a leaf is an *internal node*.
- There are no *cycles*. A cycle would be a loop. For instance, if you were to add an edge between the two lower rightmost nodes labeled, `strong` and `strong`, that would create a loop and the object would no longer be a tree.

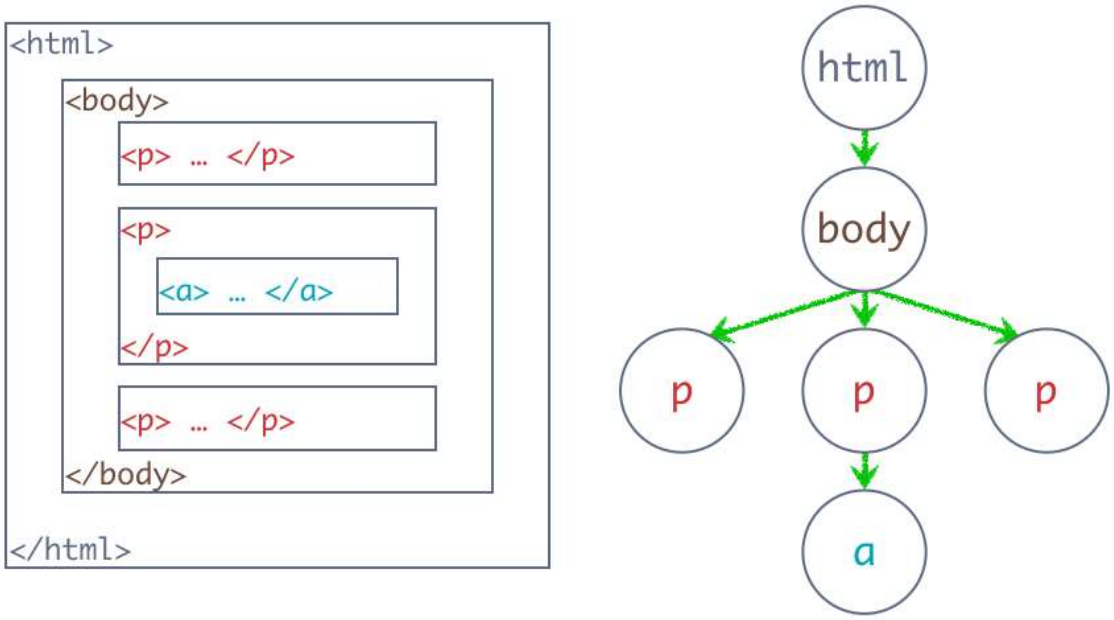
> For whatever reason, computer scientists usually view trees upside down (https://www.quora.com/Why-are-trees-in-computer-science-generally-drawn-upside-down-from-how-trees-are-in-real-life), with the "root" at the top and the "leaves" at the bottom.

The Beautiful Soup package gives you a data structure for traversing this tree. For instance, consider an HTML file with the contents below, shown both as code and pictorially.

In [1]:
```
some_page = """
<html>
  <body>
    <p>First paragraph.</p>
    <p>Second paragraph, which links to the <a href="http://www.gatech.edu">Georgia Tech websit
e</a>.</p>
    <p>Third paragraph.</p>
  </body>
</html>
"""
```

```
print(some_page)
```

```
<html>
  <body>
    <p>First paragraph.</p>
    <p>Second paragraph, which links to the <a href="http://www.gatech.edu">Georgia Tech website
</a>.</p>
    <p>Third paragraph.</p>
  </body>
</html>
```



Exercise 0. Besides HTML files, what else have we seen in this class that could be represented by a tree? Briefly and roughly explain what and how.

Student's answer Score: 0.0 / 0.0 (Top)

Answer. One thing that has a natural tree representation is a Python program! For example, can you draw the following program as a tree?

```
import re

def scan_lines(text, pattern):
    matches = []
    for line in text.split('\n'):
        if re.search(pattern, text) is not None:
            matches.append(True)
        else:
            matches.append(False)
    return matches
```

Using Beautiful Soup

Here is how you might use Beautiful Soup to inspect the structure of `some_page`.

Let's start by taking the contents of the page above (`some_page`) and asking Beautiful Soup to process it. Let's store the result in object named `soup`, and then explore its contents:

In [2]:
```
from bs4 import BeautifulSoup

soup = BeautifulSoup(some_page, "lxml")

print('1. soup ==', soup) # Print the HTML contents
print('\n2. soup.html ==', soup.html) # Root of the tree
print('\n3. soup.html.body ==', soup.html.body) # A child tag
print('\n4. soup.html.body.p ==', soup.html.body.p) # Another child tag
print('\n5. soup.html.body.contents ==', type(soup.html.body.contents), '::', soup.html.body.co
ntents)
```

```
1. soup == <html>
<body>
<p>First paragraph.</p>
<p>Second paragraph, which links to the <a href="http://www.gatech.edu">Georgia Tech website</a
>.</p>
<p>Third paragraph.</p>
</body>
</html>


2. soup.html == <html>
<body>
<p>First paragraph.</p>
<p>Second paragraph, which links to the <a href="http://www.gatech.edu">Georgia Tech website</a
>.</p>
<p>Third paragraph.</p>
</body>
</html>

3. soup.html.body == <body>
<p>First paragraph.</p>
<p>Second paragraph, which links to the <a href="http://www.gatech.edu">Georgia Tech website</a
>.</p>
<p>Third paragraph.</p>
</body>

4. soup.html.body.p == <p>First paragraph.</p>

5. soup.html.body.contents == <class 'list'> :: ['\n', <p>First paragraph.</p>, '\n', <p>Second
paragraph, which links to the <a href="http://www.gatech.edu">Georgia Tech website</a>.</p>,
'\n', <p>Third paragraph.</p>, '\n']
```

Observe that the `.` notation allows us to reference HTML tags---that is, the stuff enclosed in angle brackets in the original HTML, e.g., `<html> ... </html>`, `<body> ... </body>`---as they are nested. But in the case of the `<body> ... </body>` tag, there are multiple subtags. Evidently, `soup.html.body.contents` contains these, as a list, which we know how to manipulate.

In [3]:
```
# Enumerate all tags within the <body> ... </body> tag:
for i, elem in enumerate (soup.html.body.contents):
    print ("[{:4d}]".format (i), type (elem), '\n\t==>', "'{}'".format (elem))

# Reference one of these, element 3:
elem3 = soup.html.body.contents[3]
print(elem3.contents)
```

```
[   0] <class 'bs4.element.NavigableString'>
        ==> '
'
[   1] <class 'bs4.element.Tag'>
        ==> '<p>First paragraph.</p>'
[   2] <class 'bs4.element.NavigableString'>
        ==> '
'
[   3] <class 'bs4.element.Tag'>
        ==> '<p>Second paragraph, which links to the <a href="http://www.gatech.edu">Georgia Tec
h website</a>.</p>'
[   4] <class 'bs4.element.NavigableString'>
        ==> '
'
[   5] <class 'bs4.element.Tag'>
        ==> '<p>Third paragraph.</p>'
[   6] <class 'bs4.element.NavigableString'>
        ==> '
'
['Second paragraph, which links to the ', <a href="http://www.gatech.edu">Georgia Tech website</
a>, '.']
```

Exercise 1. Write a statement that navigates to the tag representing the GT website link. Store this resulting tag object in a variable called `link`.

In [4]:
Student's answer (Top)

```
link = soup.html.body.contents[3].contents[1]

print(link)
```

```
<a href="http://www.gatech.edu">Georgia Tech website</a>
```

In [5]:
Grade cell: `ex5_test` Score: 0.0 / 0.0 (Top)

```
# Checks your link. Can you understand what it is doing?
import bs4
```

```
assert type(link) is bs4.element.Tag
assert link.name == 'a'
assert link['href'] == 'http://www.gatech.edu'
assert link.contents == ['Georgia Tech website']
```

Other navigation tools

This lab includes a static copy of the Yelp! results for a search of "universities" in ATL. Let's start by downloading this file.

In [6]:
```
# Run me: Code to download sample HTML file

import requests
import os
import hashlib

yelp_htm = 'yelp_atl_unies.html'
yelp_htm_checksum = 'a940e7cd0c8c408a5dd2098a87303afe'

if os.path.exists('.voc'):
    data_url = 'https://cse6040.gatech.edu/datasets/yelp-example-uni/{}'.format(yelp_htm)
else:
    data_url = 'https://github.com/cse6040/labs-fa17/raw/master/datasets/{}'.format(yelp_htm)

if not os.path.exists(yelp_htm):
    print("Downloading: {} ...".format(data_url))
    r = requests.get(data_url)
    with open(yelp_htm, 'w', encoding=r.encoding) as f:
        f.write(r.text)

with open(yelp_htm, 'r') as f:
    yelp_html = f.read().encode(encoding='utf-8')
    checksum = hashlib.md5(yelp_html).hexdigest()
    assert checksum == yelp_htm_checksum, "Downloaded file has incorrect checksum!"

print("'{}' is ready!".format(yelp_htm))
```

```
'yelp_atl_unies.html' is ready!
```

Next, inspect and run this code, which prints the top (number one) result.

In [7]:
```
uni_html_text = open (yelp_htm, 'r').read()
uni_soup = BeautifulSoup(uni_html_text, "lxml")

print("The number 1 ATL university according to Yelp!:")

uni_1 = uni_soup.html.body \
    .contents[7] \
    .contents[9] \
    .contents[3] \
    .contents[1] \
    .contents[3] \
    .contents[1] \
    .contents[1] \
    .contents[7] \
    .contents[3] \
    .contents[5] \
    .contents[1] \
    .contents[1] \
    .contents[1] \
    .contents[1] \
    .contents[3] \
    .contents[1] \
    .contents[1] \
    .contents[1] \
    .contents[0] \
    .contents[0]

print(uni_1)
```

```
The number 1 ATL university according to Yelp!:
Georgia Institute of Technology
```

We hope it is self-evident that the above method to navigate to a particular tag or element is not terribly productive or robust, particularly if there are small modifications to the HTML.

Here is an alternative. Inspect the raw HTML and observe that every non-ad search result appears in a tag of the form,

```
<span class="indexed-biz-name">1.         <a class="biz-name js-analytics-click" data-analytics-label="biz-
name" href="/biz/georgia-institute-of-technology-atlanta-2" data-hovercard-id="gBX8UvhOwtdD5tGJeU-hxg" ><sp
an >Georgia Institute of Technology</span></a>
</span>
```

Beautiful Soup gives us a way to search for specific tags.

In [8]:
```
indexed_unies = uni_soup.find_all(attrs={'class': 'indexed-biz-name'})
print("*** First 5 of {} results ***\n\n{}".format(len(indexed_unies), indexed_unies[:5]))
```

```
*** First 5 of 30 results ***

[<span class="indexed-biz-name">1.         <a class="biz-name js-analytics-click" data-analytics
-label="biz-name" data-hovercard-id="gBX8UvhOwtdD5tGJeU-hxg" href="/biz/georgia-institute-of-tec
hnology-atlanta-2"><span>Georgia Institute of Technology</span></a>
</span>, <span class="indexed-biz-name">2.         <a class="biz-name js-analytics-click" data-a
nalytics-label="biz-name" data-hovercard-id="13oCD5wffSr2ypav9MpCsQ" href="/biz/emory-university
-atlanta-2"><span>Emory <span class="highlighted">University</span></span></a>
</span>, <span class="indexed-biz-name">3.         <a class="biz-name js-analytics-click" data-a
nalytics-label="biz-name" data-hovercard-id="jAebE83Ox0lPCNsJoQII4A" href="/biz/spelman-college-
atlanta"><span>Spelman College</span></a>
</span>, <span class="indexed-biz-name">4.         <a class="biz-name js-analytics-click" data-a
nalytics-label="biz-name" data-hovercard-id="8YFAsc5dK1K6FYbn9sVQ7g" href="/biz/oglethorpe-unive
rsity-atlanta"><span>Oglethorpe <span class="highlighted">University</span></span></a>
</span>, <span class="indexed-biz-name">5.         <a class="biz-name js-analytics-click" data-a
nalytics-label="biz-name" data-hovercard-id="o5tSSq2nJA-vseLTEDW9UA" href="/biz/georgia-state-un
iversity-atlanta-3"><span>Georgia State <span class="highlighted">University</span></span></a>
</span>]
```

Exercise 2. Based on the above, write a function that, given a Yelp! search results page such as `uni_soup` above, returns the name of the number 1 indexed search result.

In [9]:

Student's answer (Top)

```
def get_top_yelp_result(soup):
    """Given a Yelp! search result as a Beautiful Soup page,
    returns the name of the number 1 indexed result.
    """
    bizzes = soup.find_all (attrs={'class': 'indexed-biz-name'})
    top_biz = bizzes[0]
    rank = top_biz.contents[0].strip ()
    assert rank == '1.'
    return top_biz.contents[1].contents[0].contents[0]
```

In [10]:

Grade cell: `ex6_test` Score: 0.0 / 0.0 (Top)

```
print(get_top_yelp_result(uni_soup))
assert get_top_yelp_result(uni_soup) == 'Georgia Institute of Technology'
```

```
Georgia Institute of Technology
```

This mini-tutorial only scratches the surface of what is possible with Beautiful Soup. As always, refer to the package's documentation (https://www.crummy.com/software/BeautifulSoup/) for all the awesome deets!

---

part2 (Score: 0.0 / 0.0)

1. Test cell (Score: 0.0 / 0.0)
2. Test cell (Score: 0.0 / 0.0)

Important note! Before you turn in this lab notebook, make sure everything runs as expected:

- First, **restart the kernel** -- in the menubar, select Kernel→Restart.
- Then **run all cells** -- in the menubar, select Cell→Run All.

Make sure you fill in any place that says `YOUR CODE HERE` or "YOUR ANSWER HERE."

Part 2: Mining the web: Web APIs

We hope the preceding exercise was painful: even with tools to process HTML, it is rough downloading raw HTML and trying to extract information from it!

> Can you think of any other reasons why scraping websites for data in this way is not a good idea?

Luckily, many websites provide an application programming interface (API) for querying their data or otherwise accessing their services from your programs. For instance, Twitter provides a web API for gathering tweets, Flickr provides one for gathering image data, and Github for accessing information about repository histories.

These kinds of web APIs are much easier to use than, for instance, the preceding technique which scrapes raw web pages and then has to parse the resulting HTML. Moreover, there are more scalable in the sense that the web servers can transmit structured data in a less verbose form than raw HTML.

As a starting example, here is some code to look at the activity on Github related to the public version of our course's materials.

In [1]:
```
import requests

response = requests.get ('https://api.github.com/repos/cse6040/labs-fa17/events')

print ("==> .headers:", response.headers, "\n")
```

```
==> .headers: {'Server': 'GitHub.com', 'X-XSS-Protection': '1; mode=block', 'Access-Control-Expo
se-Headers': 'ETag, Link, X-GitHub-OTP, X-RateLimit-Limit, X-RateLimit-Remaining, X-RateLimit-Re
set, X-OAuth-Scopes, X-Accepted-OAuth-Scopes, X-Poll-Interval', 'Transfer-Encoding': 'chunked',
'Content-Encoding': 'gzip', 'Access-Control-Allow-Origin': '*', 'Strict-Transport-Security': 'ma
x-age=31536000; includeSubdomains; preload', 'X-GitHub-Media-Type': 'github.v3; format=json', 'X
-Runtime-rack': '0.028060', 'X-Content-Type-Options': 'nosniff', 'Status': '200 OK', 'X-Poll-Int
erval': '60', 'Content-Security-Policy': "default-src 'none'", 'Content-Type': 'application/jso
n; charset=utf-8', 'Date': 'Wed, 20 Sep 2017 00:37:31 GMT', 'X-RateLimit-Limit': '60', 'X-GitHub
-Request-Id': 'CDDA:3B92:5B2248B:C97424F:59C1B84B', 'Cache-Control': 'public, max-age=60, s-maxa
ge=60', 'Last-Modified': 'Tue, 19 Sep 2017 22:48:55 GMT', 'ETag': 'W/"911dffe660c66d48fac1229aff
f5fd9b"', 'X-RateLimit-Remaining': '56', 'X-Frame-Options': 'deny', 'Vary': 'Accept', 'X-RateLim
it-Reset': '1505871386'}
```

Note the `Content-Type` of the response:

In [2]:
```
print (response.headers['Content-Type'])
```

```
application/json; charset=utf-8
```

The response is in JSON format, which is an open format for exchanging semi-structured data. (JSON stands for **J**ava**S**cript **O**bject **N**otation.) JSON is designed to be human-readable and machine-readable, and maps especially well in Python to nested dictionaries. Let's take a look.

> See also this tutorial (http://www.w3schools.com/json/) for a JSON primer. JSON is among *the* universal formats for sharing data on the web; see, for instance, https://www.sitepoint.com/10-example-json-files/ (https://www.sitepoint.com/10-example-json-files/).

In [3]:
```
import json
print(type(response.json ()))
print(json.dumps(response.json()[:3], sort_keys=True, indent=2))
```

```
<class 'list'>
[
  {
    "actor": {
      "avatar_url": "https://avatars.githubusercontent.com/u/5316640?",
      "display_login": "rvuduc",
      "gravatar_id": "",
      "id": 5316640,
      "login": "rvuduc",
      "url": "https://api.github.com/users/rvuduc"
    },
    "created_at": "2017-09-19T22:48:55Z",
    "id": "6611692513",
    "org": {
      "avatar_url": "https://avatars.githubusercontent.com/u/31073927?",
      "gravatar_id": "",
      "id": 31073927,
      "login": "cse6040",
      "url": "https://api.github.com/orgs/cse6040"
    },
    "payload": {
      "before": "572167f7d65edd7a7aa12a5dd17d703656678d14",
      "commits": [
        {
          "author": {
            "email": "richie@cc.gatech.edu",
            "name": "Richard (Rich) Vuduc"
          },
          "distinct": true,
          "message": "Added Yelp! ATL universities dataset",
          "sha": "fa1e95d4c350506b772e56bd41a01479ee3951f2",
```

```
          "url": "https://api.github.com/repos/cse6040/labs-fa17/commits/fa1e95d4c350506b772e56b
d41a01479ee3951f2"
        },
        {
          "author": {
            "email": "richie@cc.gatech.edu",
            "name": "Richard (Rich) Vuduc"
          },
          "distinct": true,
          "message": "Merge branch 'master' of github.com:cse6040/labs-fa17",
          "sha": "c138fd4aa94c56debe3e640dfd4a1ba53afcdb88",
          "url": "https://api.github.com/repos/cse6040/labs-fa17/commits/c138fd4aa94c56debe3e640
dfd4a1ba53afcdb88"
        }
      ],
      "distinct_size": 2,
      "head": "c138fd4aa94c56debe3e640dfd4a1ba53afcdb88",
      "push_id": 1996190687,
      "ref": "refs/heads/master",
      "size": 2
    },
    "public": true,
    "repo": {
      "id": 100506580,
      "name": "cse6040/labs-fa17",
      "url": "https://api.github.com/repos/cse6040/labs-fa17"
    },
    "type": "PushEvent"
  },
  {
    "actor": {
      "avatar_url": "https://avatars.githubusercontent.com/u/5316640?",
      "display_login": "rvuduc",
      "gravatar_id": "",
      "id": 5316640,
      "login": "rvuduc",
      "url": "https://api.github.com/users/rvuduc"
    },
    "created_at": "2017-09-19T19:38:11Z",
    "id": "6610937299",
    "org": {
      "avatar_url": "https://avatars.githubusercontent.com/u/31073927?",
      "gravatar_id": "",
      "id": 31073927,
      "login": "cse6040",
      "url": "https://api.github.com/orgs/cse6040"
    },
    "payload": {
      "before": "917fffcca44b55042095d5be7de61b65414bad76",
      "commits": [
        {
          "author": {
            "email": "richie@cc.gatech.edu",
            "name": "Richard (Rich) Vuduc"
          },
          "distinct": true,
          "message": "Added Yelp\\! ATL universities raw search results",
          "sha": "572167f7d65edd7a7aa12a5dd17d703656678d14",
          "url": "https://api.github.com/repos/cse6040/labs-fa17/commits/572167f7d65edd7a7aa12a5
dd17d703656678d14"
        }
      ],
      "distinct_size": 1,
      "head": "572167f7d65edd7a7aa12a5dd17d703656678d14",
      "push_id": 1995800949,
      "ref": "refs/heads/master",
      "size": 1
    },
    "public": true,
    "repo": {
      "id": 100506580,
      "name": "cse6040/labs-fa17",
      "url": "https://api.github.com/repos/cse6040/labs-fa17"
    },
    "type": "PushEvent"
  },
  {
    "actor": {
      "avatar_url": "https://avatars.githubusercontent.com/u/5316640?",
      "display_login": "rvuduc",
      "gravatar_id": "",
      "id": 5316640,
      "login": "rvuduc",
      "url": "https://api.github.com/users/rvuduc"
    },
    "created_at": "2017-09-14T23:28:12Z",
    "id": "6593207015",
    "org": {
      "avatar_url": "https://avatars.githubusercontent.com/u/31073927?",
```

```
      "gravatar_id": "",
      "id": 31073927,
      "login": "cse6040",
      "url": "https://api.github.com/orgs/cse6040"
    },
    "payload": {
      "before": "4365a884df66895ad18e5f41a280cc81797e7923",
      "commits": [
        {
          "author": {
            "email": "richie@cc.gatech.edu",
            "name": "Richard (Rich) Vuduc"
          },
          "distinct": true,
          "message": "Added Lab 5 (regular expressions)",
          "sha": "917fffcca44b55042095d5be7de61b65414bad76",
          "url": "https://api.github.com/repos/cse6040/labs-fa17/commits/917fffcca44b55042095d5b
e7de61b65414bad76"
        }
      ],
      "distinct_size": 1,
      "head": "917fffcca44b55042095d5be7de61b65414bad76",
      "push_id": 1986327053,
      "ref": "refs/heads/master",
      "size": 1
    },
    "public": true,
    "repo": {
      "id": 100506580,
      "name": "cse6040/labs-fa17",
      "url": "https://api.github.com/repos/cse6040/labs-fa17"
    },
    "type": "PushEvent"
  }
]
```


Exercise 0. It should be self-evident that the JSON response above consists of a sequence of records, which we will refer to as *events*. Each event is associated with an *actor*. Write some code to extract a dictionary of all actors, where the key is the actor's login and the value is the actor's URL.

In [4]:

Student's answer (Top)

```
def extract_actors (json_github_events):
    """Given JSON records for events in a GitHub repo,
    returns a dictionary of the actors and their URLs.
    """
    urls = {}
    for event in json_github_events:
        actor = event['actor']['display_login']
        url = event['actor']['url']
        urls[actor] = url
    return urls
```

In [5]:

Grade cell: `extract_actors_test` Score: 0.0 / 0.0 (Top)

```
actor_urls = extract_actors(response.json ())

for actor, url in actor_urls.items ():
    print ('{}: {}'.format(actor, url))
    assert url == "https://api.github.com/users/{}".format(actor)
```

```
rvuduc: https://api.github.com/users/rvuduc
Augus-Kong: https://api.github.com/users/Augus-Kong
```

Exercise 1. Write some code that goes to each actor's URL and determines their name. If an actor URL is invalid, that actor should not appear in the output.

In [6]:

Student's answer (Top)

```
def lookup_names (actor_urls):
    """Given a dictionary of (actor, url) pairs, looks up the JSON at
    the URL and extracts the user's name (if any). Returns a new
    dictionary of (actor, name) pairs.
    """
    import re

    names = {}
    for actor, url in actor_urls.items ():
        response = requests.get (url)

        # Possible error conditions
```

```
        # Possible error conditions
        if response is None: continue
        if re.search ('application/json', response.headers['Content-Type']) is None: continue
        if 'name' not in response.json (): continue

        names[actor] = response.json ()['name']
    return names
```

In [7]:

Grade cell: get_names_test Score: 0.0 / 0.0 (Top)

```
actor_names = lookup_names (actor_urls)

for actor, name in actor_names.items ():
    print ("{}: {}".format (actor, name))

assert actor_names['rvuduc'] == 'Rich Vuduc (personal account)'
```

```
rvuduc: Rich Vuduc (personal account)
Augus-Kong: XIANGYU KONG
```

That's the end of this notebook. Processing JSON is fairly straightforward, because it maps very naturally to nested dictionaries in Python. You might search the web for other sources of JSON data, including this one (https://www.yelp.com/dataset/challenge), and do your own processing!