Web Scraping 101 with Python (/2013/03/03/web-scraping-101-with-python/)

March 03, 2013 | Tags: scraping (/tag/scraping.html) python (/tag/python.html) data (/tag/data.html) tutorial (/tag/tutorial.html)

---

*After you're done reading, check out my follow-up to this post **here (/2013/04/29/more-web-scraping-with-python/)**.*

Yea, yea, I know I said I was going to **write more (http://www.gregreda.com/2013/01/23/translating sql-to-pandas-part1/)** on **pandas (http://pandas.pydata.org)**, but recently I've had a couple friends ask me if I could teach them how to scrape data. While they said they were able to find a ton of resources online, all assumed some level of knowledge already. Here's my attempt at assuming a very minimal knowledge of programming.

Getting Setup

We're going to be using Python 2.7, **BeautifulSoup (http://www.crummy.com/software/BeautifulSoup** and **lxml (http://lxml.de/)**. If you don't already have Python 2.7, you'll want to download the proper version for your OS **here (http://python.org/download/releases/2.7.3/)**.

To check if you have Python 2.7 on OSX, open up **Terminal (http://en.wikipedia.org/wiki/Terminal_(OS_X))** and type *python --version*. You should see something like this:

```
mario:~ gjreda$ python --version
Python 2.7.2
```

Next, you'll need to install **BeautifulSoup (http://www.crummy.com/software/BeautifulSoup** If you're on OSX, you'll already have **setuptools (http://pypi.python.org/pypi/setuptools)** installed. Let's use it to install **pip (http://www.pip-installer.org/en/latest/)** and use that for package management instead.

In Terminal, run *sudo easy_install pip*. You'll be prompted for your password - type it in and let it run. Once that's done, again in Terminal, *sudo pip install BeautifulSoup4*. Finally, you'll need to **install lxml (http://lxml.de/installation.html)**.

A few scraping rules

Now that we have the packages we need, we can start scraping. But first, a couple of rules.

1. You should check a site's terms and conditions before you scrape them. It's their data and they likely have some rules to govern it.
2. Be nice - A computer will send web requests much quicker than a user can. Make sure you space out your requests a bit so that you don't hammer the site's server.
3. Scrapers break - Sites change their layout all the time. If that happens, be prepared to rewrite your code.
4. Web pages are inconsistent - There's sometimes some manual clean up that has to happen even after you've gotten your data.

Finding your data

For this example, we're going to use the **Chicago Reader's Best of 2011 (http://www.chicagoreader.com/chicago/best-of-chicago-2011/BestOf?oid=4100483)** list. Why? Because I think it's a great example of terrible data presentation on the web. Go ahead and browse it for a bit.

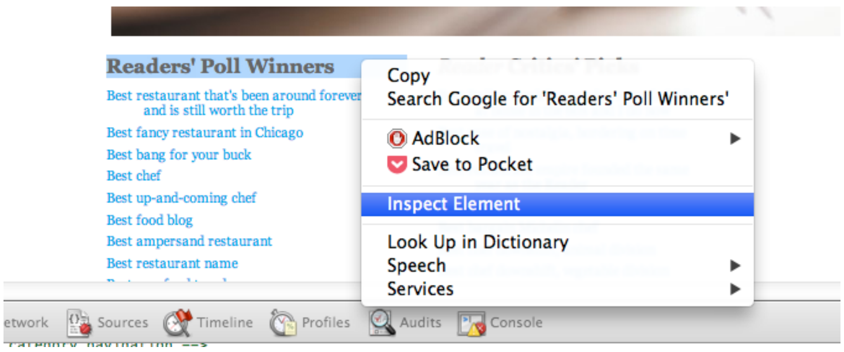
All you want to see is a list of the category, winner, and maybe the runners-up, right? But you have to continuously click link upon link, slowly navigating your way through the list.

Hopefully in your clicking you noticed the important thing though - all the pages are structured the same.

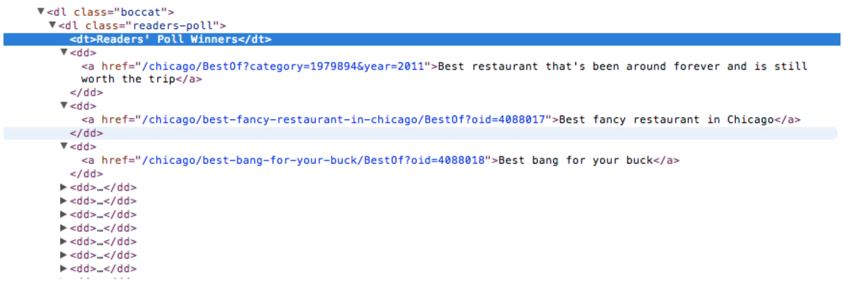
Planning your code

In looking at the **Food and Drink (http://www.chicagoreader.com/chicago/best-of-chicago-2011-food-drink/BestOf?oid=4106228)** section of the Best of 2011 list, we see that all the categories are a link. Each of those links has the winner, maybe some information about the winner (like an address), and the runners-up. It's probably a good idea to break these things into separate functions in our code.

To start, we need to take a look at the HTML that displays these categories. If you're in Chrome or Firefox, highlight "Readers' Poll Winners", right-click, and select Inspect Element.



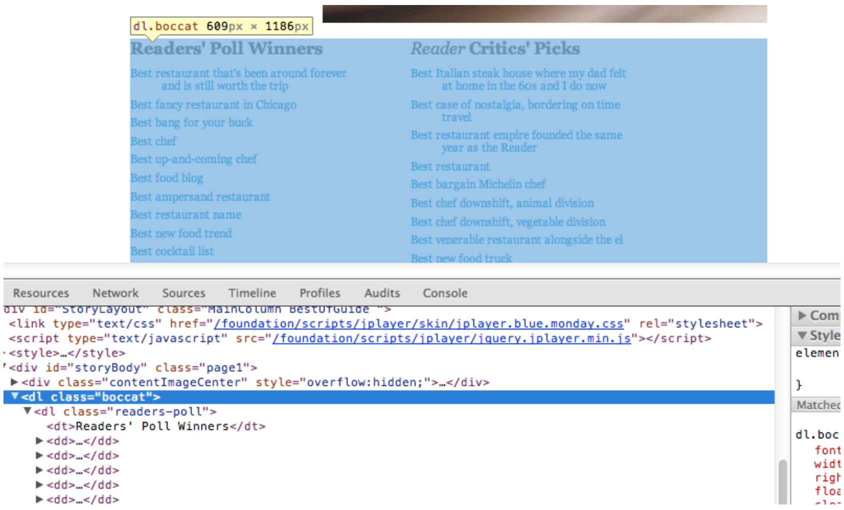
This opens up the browser's Developer Tools (in Firefox, you might now have to click the HTML button on the right side of the developer pane to fully show it). Now we'll be able to see the page layout. The browser has brought us directly to the piece of HTML that's used to display the "Readers' Poll Winners" `<dt>` element.



```
▼<dl class="boccat">
  ▼<dl class="readers-poll">
     <dt>Readers' Poll Winners</dt>
    ▼<dd>
       <a href="/chicago/BestOf?category=1979894&year=2011">Best restaurant that's been around forever and is still
       worth the trip</a>
     </dd>
    ▼<dd>
       <a href="/chicago/best-fancy-restaurant-in-chicago/BestOf?oid=4088017">Best fancy restaurant in Chicago</a>
     </dd>
    ▼<dd>
       <a href="/chicago/best-bang-for-your-buck/BestOf?oid=4088018">Best bang for your buck</a>
     </dd>
    ▶<dd>…</dd>
    ▶<dd>…</dd>
    ▶<dd>…</dd>
    ▶<dd>…</dd>
    ▶<dd>…</dd>
    ▶<dd>…</dd>
    ▶<dd>…</dd>
```

This seems to be the area of code where there's going to be some consistency in how the category links are displayed. See that `<dl class="boccat">`

just above our "Readers' Poll Winners" line? If you mouse over that line in your browser's dev tools, you'll notice that it highlights the **entire section** of category links we want. And every category link is within a *<dd>* element. Perfect! Let's get all of them.


dl.boccat 609px × 1186px
Readers' Poll Winners
Best restaurant that's been around forever and is still worth the trip
Best fancy restaurant in Chicago
Best bang for your buck
Best chef
Best up-and-coming chef
Best food blog
Best ampersand restaurant
Best restaurant name
Best new food trend
Best cocktail list
Reader Critics' Picks
Best Italian steak house where my dad felt at home in the 60s and I do now
Best case of nostalgia, bordering on time travel
Best restaurant empire founded the same year as the Reader
Best restaurant
Best bargain Michelin chef
Best chef downshift, animal division
Best chef downshift, vegetable division
Best venerable restaurant alongside the el
Best new food truck
Resources
Network
Sources
Timeline
Profiles
Audits
Console
Com
Style
Matched

```
div id="storyLayout" class="mainColumn bestOfGuide ">
<link type="text/css" href="/foundation/scripts/jplayer/skin/jplayer.blue.monday.css" rel="stylesheet">
<script type="text/javascript" src="/foundation/scripts/jplayer/jquery.jplayer.min.js"></script>
<style>…</style>
<div id="storyBody" class="page1">
 ▶<div class="contentImageCenter" style="overflow:hidden;">…</div>
 ▼<dl class="boccat">
   ▼<dl class="readers-poll">
      <dt>Readers' Poll Winners</dt>
     ▶<dd>…</dd>
     ▶<dd>…</dd>
     ▶<dd>…</dd>
     ▶<dd>…</dd>
     ▶<dd>…</dd>
```

```
element
}
dl.boc
  font
  widt
  righ
  floa
```

Our first function – getting the category links

Now that we know we know the *<dl class="boccat">* section holds all the links we want, let's write some code to find that section, and then grab all of the links within the *<dd>* elements of that section.

```
from bs4 import BeautifulSoup
from urllib2 import urlopen

BASE_URL = "http://www.chicagoreader.com"

def get_category_links(section_url):
    html = urlopen(section_url).read()
    soup = BeautifulSoup(html, "lxml")
    boccat = soup.find("dl", "boccat")
    category_links = [BASE_URL + dd.a["href"] for dd
 in boccat.findAll("dd")]
    return category_links
```

Hopefully this code is relatively easy to follow, but if not, here's what we're doing:

1. Loading the urlopen function from the urllib2 library into our local **namespace (http://en.wikipedia.org/wiki/Namespace_(compute**
2. Loading the BeautifulSoup class from the bs4 (BeautifulSoup4) library into our local namespace.
3. Setting a variable named *BASE_URL* to "http://www.chicagoreader.com". We do this because the links used through the site are relative - meaning they do not include the base domain. In order to store

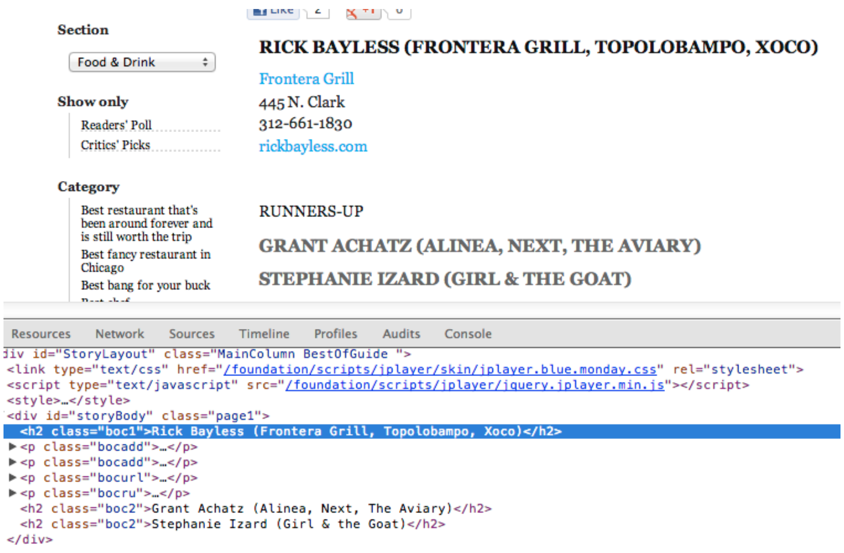
our links properly, we need to concatenate the base domain with each relative link.

4. Define a function named *get_category_links*.
    1. The function requires a parameter of *section_url*. In this example, we're going to use the **Food and Drink (http://www.chicagoreader.com/chicago/best-of-chicago-2011-food-drink/BestOf?oid=4106228)** section of the BOC list, however we could use a different section URL - for instance, the **City Life (http://www.chicagoreader.com/chicago/best-of-chicago-2011-city-life/BestOf?oid=4106233)** section's URL. We're able to create just one generic function because each section page is structured the same.
    2. Open the section_url and read it in the *html* object.
    3. Create an object called *soup* based on the BeautifulSoup class. The *soup* object is an **instance (http://en.wikipedia.org/wiki/Instance_(compute** of the BeautifulSoup class. It is initialized with the html object and parsed with **lxml (http://lxml.de/)**.
    4. In our BeautifulSoup instance (which we called *soup*), find the `<dl>` element with a class of "boccat" and store that section in a variable called *boccat*.
    5. This is a **list comprehension (http://docs.python.org/2/tutorial/datastructures comprehensions)**. For every `<dd>` element found within our *boccat* variable, we're getting the href of its `<a>` element (our category links) and concatenating on our *BASE_URL* to make it a complete link. All of these links are being stored in a list called *category_links*. You could also write this line with a **for loop (http://docs.python.org/2/tutorial/controlflow.ht statements)**, but I prefer a list comprehension here because of its simplicity.
    6. Finally, our function returns the *category_links* list that we created on the previous line.

Our second function - getting the category, winner, and runners-up

Now that we have our list of category links, we'd better start going through them to get our winners and runners-up. Let's figure out which elements contain the parts we care about.

If we look at the **Best Chef (http://www.chicagoreader.com/chicago/best-chef/BestOf?oid=4088191)** category, we can see that our category is in *<h1 class="headline">*. Shortly after that, we find our winner and runners-up stored in *<h2 class="boc1">* and *<h2 class="boc2">*, respectively.


Section
Food & Drink
Show only
Readers' Poll
Critics' Picks
Category
Best restaurant that's been around forever and is still worth the trip
Best fancy restaurant in Chicago
Best bang for your buck

RICK BAYLESS (FRONTERA GRILL, TOPOLOBAMPO, XOCO)

Frontera Grill
445 N. Clark
312-661-1830
rickbayless.com

RUNNERS-UP

GRANT ACHATZ (ALINEA, NEXT, THE AVIARY)

STEPHANIE IZARD (GIRL & THE GOAT)

```
div id="StoryLayout" class="MainColumn BestOfGuide ">
<link type="text/css" href="/foundation/scripts/jplayer/skin/jplayer.blue.monday.css" rel="stylesheet">
<script type="text/javascript" src="/foundation/scripts/jplayer/jquery.jplayer.min.js"></script>
<style>…</style>
<div id="storyBody" class="page1">
  <h2 class="boc1">Rick Bayless (Frontera Grill, Topolobampo, Xoco)</h2>
▶ <p class="bocadd">…</p>
▶ <p class="bocadd">…</p>
▶ <p class="bocurl">…</p>
▶ <p class="bocru">…</p>
  <h2 class="boc2">Grant Achatz (Alinea, Next, The Aviary)</h2>
  <h2 class="boc2">Stephanie Izard (Girl & the Goat)</h2>
</div>
```

Let's write some code to get all of them.

```
def get_category_winner(category_url):
    html = urlopen(category_url).read()
    soup = BeautifulSoup(html, "lxml")
    category = soup.find("h1", "headline").string
    winner = [h2.string for h2 in soup.findAll("h2",
 "boc1")]
    runners_up = [h2.string for h2 in soup.findAll("h2"
 "boc2")]
    return {"category": category,
            "category_url": category_url,
            "winner": winner,
            "runners_up": runners_up}
```

It's very similar to our last function, but let's walk through it anyway.

1. Define a function called *get_category_winner*. It requires a *category_url*.
2. Lines two and three are actually exactly the same as before - we'll come back to this in the next section.
3. Find the string within the *<h1 class="headline">* element and store it in a variable named category.
4. Another list comprehension - store the string within every *<h2 class="boc1">* element in a list called *winner*. But shouldn't there be only one winner? You'd think that, but some have multiple (e.g. **Best Bang for your Buck (http://www.chicagoreader.com/chicago/best-bang-for-your-buck/BestOf?oid=4088018)**).
5. Same as the previous line, but this time we're getting the runners-up.
6. Finally, return a **dictionary (http://docs.python.org/2/tutorial/datastructures.htr**

with our data.

DRY - Don't Repeat Yourself

As mentioned in the previous section, lines two and three of our second function mirror lines in our first function.

Imagine a scenario where we want to change the parser we're passing into our BeautifulSoup instance (in this case, lxml). With the way we've currently written our code, we'd have to make that change in two places. Now imagine you've written many more functions to scrape this data - maybe one to get addresses and another to get **paragraphs of text about the winner (http://www.chicagoreader.com/chicago/best-new-food-truckfood/BestOf?oid=4101387)** - you've likely repeated those same two lines of code in these functions and you now have to remember to make changes in four different places. That's not ideal.

A good principle in writing code is **DRY - Don't Repeat Yourself (http://en.wikipedia.org/wiki/Don't_repeat_yoursel** When you notice that you've written the same lines of code a couple times throughout your script, it's probably a good idea to step back and think if there's a better way to structure that piece.

In our case, we're going to write another function to simply process a URL and return a BeautifulSoup instance. We can then call this function in our other functions instead of duplicating our logic.

```
def make_soup(url):
    html = urlopen(url).read()
    return BeautifulSoup(html, "lxml")
```

We'll have to change our other functions a bit now, but it's pretty minor - we just need to replace our duplicated lines with the following:

```
soup = make_soup(url) # where url is the url we're p
assing into the original function
```

Putting it all together

Now that we have our main functions written, we can write a script to output the data however we'd like. Want to write to a CSV file? Check out Python's **DictWriter (http://docs.python.org/2/library/csv.html#csv.Dic** class. Storing the data in a database? Check out the **sqlite3 (http://docs.python.org/2/library/sqlite3.html)** or **other various database libraries (http://wiki.python.org/moin/DatabaseInterfaces)**. While both tasks are somewhat outside of my intentions for this post, if there's interest, let me know in the comments and I'd be happy to write more.

Hopefully you found this post useful. I've put a final example script in **this gist (http://bit.ly/13yd9ng)**.

Tweet Share 127

← Back to Home (/)