day_two

May 27, 2016

1 Day 2

A vast amount of data exists on the web and is now publicly available. In this section, we give an overview of popular ways to retrieve data from the web, and walk through some important concerns and considerations.

1.1 Background

** 1) How does the web work? **
** - a) Examining a http request through your browser (Chrome/Firefox) **
** - b) Examining a http request through your console **
** 2) Web terminology: some important distinctions **
** - a) Web scraping vs APIs - what's the difference? **
** - b) Web scrapers vs crawlers & spiders - what's the difference? **
** 3) Building friendly bots: robots.txt and legality **

1.2 Tutorial

** 1) Creating a friendly bot on Wikipedia **
** 2) Twitter API **

In many cases, it's useful to control the order in which statements or function calls are executed or evaluated. A control flow statement determines which path or paths in a program should be followed. Control flow statements, for example, can:

- execute a set of statements if a condition or certain conditions are met
- execute a set of statements `n` times until a condition or certain conditions are met
- stop the execution of a program

How can we achieve this? The most well-known statement type is the `if` statement.

```
In [1]: x = 0
```

```
In [2]: if x == 0:
            print('x is zero')
```

```
x is zero
```

`if` statements make use of boolean expressions. If the expression (or set of expressions) evaluate to `True`, the indented statement gets executed. Otherwise, nothing happens.

```
In [3]: x = 1
```

```
In [4]: if x == 0:
            print('x is zero')
```

The code above is referred to as a clause. Clauses contain "headers" and "bodies." Clause headers begin with identifying keywords—in this case, `if`—include boolean expressions, and end with colons. The body is a group of indented statements controlled by the clause. This is also known as a "block."

Compound statements are made up of one or more clauses. For example, there might be two possibilities in which case we use the `else` keyword. We can combine the above as follows.

```
In [5]: if x == 0:
            print('x is zero')
        else:
            print('x is not zero')
```

```
x is not zero
```

Notice that clause headers are at the same indentation level.

When there are more than two possibilities, we can use what are called chained conditionals. For this, we use the `elif` keyword.

```
In [6]: if x == 0:
            print('x is zero')
        elif x < 0:
            print('x is negative')
        elif x > 0:
            print('x is positive')
```

```
x is positive
```

Of course, the code above only works if `x` is numeric. Assuming this is the case, all possible values of `x` are listed. Because of this, we can change the last clause (`elif x > 0`) to `else`.

There isn't a "right" way to do this. A good approach is to write it such that its easily readable for yourself and others.

What if `x` is *not* numeric? With the code as is, we'll get a `TypeError`. So, let's generalize what we have and wrap it in a function.

```
In [7]: def x_is(x):
            if type(x) is str:
                print('x is str')
            elif type(x) in [int, float]:
                if x == 0:
                    print('x is zero')
                elif x < 0:
                    print('x is negative')
                elif x > 0:
                    print('x is positive')
            else:
                print('invalid x value')
```

Before we call our function, let's explain what's going on. Our function, as defined, is an example of a "nested conditional." We first perform a type check and, if `x` is numeric, there are another set of conditions which are checked.

```
In [8]: x_is('ucb')
```

```
x is str
```

```
In [9]: x_is(1)
```

```
x is positive
```

```
In [10]: x_is(0)

x is zero

In [11]: x_is([1, 2, 3])

invalid x value

In [12]: x_is(None)

invalid x value
```

Sometimes, specifying `if/ifelse` decision trees can become very complicated, especially when dealing with foreign software that may present unexpected responses or datatypes. For example, data analysis software frequently encodes different types of null values in data, like:

- `0` for a value of 0
- `-` for missing, which should become `None` in Python
- `NaN` for something that can't be a number like infiniti, which might become `None` or `"NaN"`

Handling this with a bunch of `elif` statements hurts readibility in code, and can make it easier to commit errors while the code is being written. In Python, one way around this is to use `exception handling`. This means to let Python try to do something, and then tell it to do something else only if the original attempt fails. So, if we have data like:

```
In [13]: my_data = [1, 2, 0, None, "NaN"]
```

Trying to divide all of the values by a common denominator will raise a `TypeError` for both the `None` value and the string.

```
In [14]: new_data = []
         for item in my_data:
             new_data.append(item/10)


         ---------------------------------------------------------------------------

         TypeError                                 Traceback (most recent call last)

         <ipython-input-14-261a655468d4> in <module>()
           1 new_data = []
           2 for item in my_data:
         ----> 3     new_data.append(item/10)


         TypeError: unsupported operand type(s) for /: 'NoneType' and 'int'
```

We can get around this by wrapping this statement in a `try` clause

```
In [15]: new_data = []
         for item in my_data:
             try:
                 new_data.append(item/10)
             except:
                 pass
         print(new_data)
```

```
[0.1, 0.2, 0.0]
```

Notice that those two other values have disappeared – we have steamrolled any problem data, and now have less than we started with. We have also steamrolled **every possible error**, which means if Python raised a `SteppedOnCrackError: mothers back broken`, you would never know.

Until you got home.

We can rewrite this to substitute `None`, but only if there is a `TypeError`.

```
In [16]: new_data = []
         for item in my_data:
             try:
                 new_data.append(item/10)
             except TypeError:
                 new_data.append(None)
         print(new_data)
```

```
[0.1, 0.2, 0.0, None, None]
```

Let's try a challenge! Error handling - or having a computer program anticipate and respond to errors created by other functions - is a big part of programming. To give you a little more practice with this, we're going to have you team up with person sitting next to you and try challenge A in the challenges directory.

Control flow **DOCS**

2 Background:

2.1 1) How does the web work?

An extremely simplified model of the web is as follows. The World Wide Web is said to follow a client-server architecture, where clients (etc. the web browser on your computer) send requests to servers, and servers respond with resources. When you enter a URL (or Uniform Resource Locator) into your browser, your browser sends a http request with information about the resource you are looking for to a remote server, which the server returns, if available.

A server can be understood as a computer that has various files (resources) stored in its system, and that returns those files if it receives requests in a format it understands.

2.2 1a). Examining a request through your browser (Chrome/Firefox)

You can view the request sent by your browser by:

1) Opening a new tab in your browser

2) Enabling developer tools (**View -> Developer -> Developer Tools in Chrome** and **Tools -> Web Developer -> Toggle Tools in Firefox**)

3) Loading or reloading a web page (etc. www.google.com)

4) Navigating to the Network tab in the panel that appears at the bottom of the page.

2.2.1 Chrome Examine Request Example

2.2.2 Firefox Examine Request Example

These requests you send follow the HTTP protocol (Hypertext Transfer Protocol), part of which defines the information (along with the format) the server needs to receive to return the right resources. Your HTTP request contains **headers**, which contains information that the server needs to know in order to return the right information to you.

2.3 1b). Examining a http request through the console

Let's now try accessing the same server by using requests. Now, instead of sending the server a request through your browser, you are sending the server a request programmatically, through your console. The server returns some output to you, which the requests module parses as a python object.

```
In [17]: import requests

         r = requests.get("http://www.google.com")
```

This response object contains various information about the request you sent to the server, the resources returned, and information about the response the server returned to you, among other information. These are accessible through the **request** attribute, the **content** attribute and the **headers** attribute respectively, which we'll each examine below.

```
In [18]: type(r.request), type(r.content), type(r.headers)

Out[18]: (requests.models.PreparedRequest,
          bytes,
          requests.structures.CaseInsensitiveDict)
```

Here, we can see that **request** is an object with a custom type, **content** is a str value and **headers** is an object with "dict" in its name, suggesting we can interact with it like we would with a dictionary.

If we recall our simple model of the web, we sent a http request through our console to a remote server, which returned a response. Both the request and response contains information that first allows the server to determine the right resource to return, and then typically, our browser to interpret the returned object.

The content is the actual resource returned to us - let's take a look at the content first before examining the request and response objects more carefully. (We select the first 1000 characters b/c of the display limits of Jupyter/python notebook.)

```
In [19]: from pprint import pprint
         pprint(r.content[0:1000])
```

```
(b'<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang='
 b'"en"><head><meta content="Search the world\'s information, including webp'
 b'ages, images, videos and more. Google has many special features to help you '
 b'find exactly what you\'re looking for." name="description"><meta content='
 b'"noodp" name="robots"><meta content="text/html; charset=UTF-8" http-equiv="C'
 b'ontent-Type"><meta content="/images/branding/googleg/1x/googleg_standard_col'
 b'or_128dp.png" itemprop="image"><title>Google</title><script>(function(){wind'
 b"ow.google={kEI:'nsxIV7uFJMiCjwPEibToBQ',kEXPI:'1350654,3700062,3700258,37003"
 b'88,3700389,4026241,4029815,4031109,4032678,4036509,4036527,4038012,4039268,4'
 b'043492,4045841,4048347,4052304,4054551,4056038,4057739,4058337,4058543,40597'
 b'67,4061155,4061181,4061552,4061925,4062331,4062987,4063112,4063126,4063473,4'
 b'063879,4063929,4063960,4064449,4064501,4064904,4064931,4065154,4065406,40657'
 b'86,4065855,4066196,4066328,4066654,4066963,4067049,4067125,4067213,4067274,4'
 b'067384,4067518,40676')
```

2.3.1 HTML: language for computers

The content returned is written in **HTML (HyperText Markup Language)**, which is the default format in which web pages are returned. The content looks like gibberish at first, with little to no spacing. The reason for this is that this output is not designed for us to read, but for the browser to parse and present in a visual interface.

The HTML raw document contains both the text in the web page, such as "Google Research" or "I'm Feeling Lucky", as well as tags and information about how the text is to be formatted and presented,

including positioning, font size and the layout of the site. When we begin writing our web scraper for Wikipedia, we'll go into more detail how to navigate and parse the HTML structure to locate and extract the data you need.

If you save a web page as a ".html" file, and open the file in a text editor like Notepad++ or Sublime Text, this is the same format you'll see. Opening the file in a browser (i.e. by double-clicking it) gives you the Google home page you are familiar with.

Next, let's take a look at the request attribute. Notice that the request attribute is attached to our response object returned from requests.get, i.e. the http request has already been sent and the request attribute is provided for convenience to see what request headers you sent, after-the-fact.

Let's print out the headers associated with our request. The **url** and **method** attribute contains other key information associated with the request. We can see the **headers**, **url** and **method** attributes in the dir, you can also use the **getattr** function or just check to see if a word is in the headers list (if the headers list is too long).

```
In [20]: r.request.headers
```

```
Out[20]: {'Connection': 'keep-alive', 'Accept': '*/*', 'User-Agent': 'python-requests/2.9.1', 'Accept-E
```

2.3.2 Printing information associated with request

```
In [21]: pprint("url: " + r.request.url)
         pprint("method: " + r.request.method)
```

```
'url: http://www.google.com/'
'method: GET'
```

```
In [22]: pprint(r.request.headers.items())
```

```
ItemsView({'Connection': 'keep-alive', 'Accept': '*/*', 'User-Agent': 'python-requests/2.9.1', 'Accept-
```

The method associated with the request (GET here) is part of a number of other methods defined in the HTTP Protocol, including GET, POST, PUT, DELETE, etc.

Of these, the most common are GET and POST, with the GET method typically used for data retrieval and the POST method used to make changes in the server's database. We shall return to GET again in our Wikipedia web scraping tutorial, which is usually the only method used for web scraping.

We won't go too much into what some of these other header fields mean, which you should be able to find references for easily online (etc: https://en.wikipedia.org/wiki/List_of_HTTP_header_fields).

Nonetheless, when troubleshooting your code for extracting data from the web, you'll often find yourself examining the header fields for both the request and response messages.

To round out this section, let's briefly examine the headers associated with the response (rather than the request) with the techniques we've learned, which are directly available in the main response object we have been working with.

```
In [23]: pprint(response.headers.items())
```

```
---------------------------------------------------------------------------

NameError                                 Traceback (most recent call last)

<ipython-input-23-1002f5c92f98> in <module>()
----> 1 pprint(response.headers.items())


NameError: name 'response' is not defined
```

2.3.3 End Note: Browser vs. Console

From the server's perspective, the request it receives from your browser is not so different from the request received from your console (though some servers use a range of methods to determine if the request comes from a "valid" person using a browser, versus an automated program.)

The server relies on the header request fields to determine what to return, and includes a number of header fields in its response, in addition to its content.

The main difference is that in the browser, you interact with the server via a graphical user interface (GUI), so that much of the header specification, both in the request and response, remain invisible to you. In your console, you often have to specify or parse this content manually - while this involves more work, it also allows you a great deal more flexibility, and the ability to automate certain tasks.

2.4 2) Web terminology: some important distinctions

2.5 2b) Menagerie of tools: crawlers, spiders, scrapers - what's the difference?

Web crawlers or spiders are used by search engines to index the web. The metaphor is that of an automated bot with long, spindly legs, traversing from hyperlink to hyperlink. Search engines use these crawlers to continually traverse the web and index new or changed content, so that our search queries reflect the most recent and up-to-date content.

Web scraping is a little different. While many of the tools used may be identical or similar, web scraping "focuses more on the transformation of unstructured data on the web, typically in HTML format, into structured data that can be stored and analyzed in a central local database or spreadsheet." (https://en.wikipedia.org/wiki/Web_scraping) In other words, web scraping focuses on translating data into a form ready for storage and analysis (versus just indexing).

In many cases, to the server, these processes look somewhat identical. Resources are sent in response to requests. Rather, it is what is done to those resources after they are sent, and the overall goal, that differentiates web crawling and scraping.

Most websites want crawlers to find them so their pages appear on popular search engines, but see no clear-cut benefit when their content is parsed and converted into usable data. Beyond research, many companies also use web scraping (in a legal grey area or illegally) to repurpose content, etc, a real estate website scraping data from Craigslist to re-post as listings on their website.

2.6 4) Considerate robots and legality

Typically, in starting a new web scraping project, you'll want to follow these steps:
1) Find the websites' robots.txt and do not access those pages through your bot
2) Make sure your bot does not make too many requests in a specific period (etc. by using Python's sleep.wait function)
3) Look up the website's term of use or terms of service.

We'll discuss each of these briefly.

2.6.1 What data owners care about

Data owners are concerned with:
1) Keeping their website up
2) Protecting the commercial value of their data

Their policies and responses differ with respect to these two areas. You'll need to do some research to determine what is appropriate with regards to your research.

1) Keeping their website up Most commercial websites have strategies to throttle or block IPs that make too many requests within a fixed amount of time. Because a bot can make a large number of requests in a small amount of time (etc. entering 100 different terms into Google in one second), servers are able to determine if traffic is coming from a bot or a person (among many other methods). For companies that

rely on advertising, like Google or Twitter, these requests do not represent "human eyeballs" and need to be filtered out from their bill to advertisers.

In order to keep their site up and running, companies may block your IP temporarily or permanently if they detect too many requests coming from your IP, or other signs that requests are being made by a bot instead of a person. If you systematically down a site (such as sending millions of requests to an official government site), there is the small chance your actions may be interpreted maliciously (and regarded as hacking), with risk of prosecution.

2) Protecting the commercial value of their data Companies are also typically very protective of their data, especially data that ties directly into how they make money. A listings site (like Craigslist), for instance, would lose traffic if listings on its site were poached and transfered to a competitor, or if a rival company used scraping tools to derive lists of users to contact. For this reason, companies' term of use agreements are typically very restrictive of what you can do with their data.

Different companies may have a range of responses to your scraping, depending on what you do with the data. Typically, repurposing the data for a rival application or business will trigger a strong response from the company (i.e. legal attention). Publishing any analysis or results, either in a formal academic journal or on a blog or webpage, may be of less concern, though legal attention is still possible.

2.6.2 robots.txt: internet convention

The robots.txt file is typically located in the root folder of the site, with instructions to various services (User-agents) on what they are not allowed to scrape.

Typically, the robots.txt file is more geared towards search engines (and their crawlers) more than anything else.

However, companies and agencies typically will not want you to scrape any pages that they disallow search engines from accessing. Scraping these pages makes it more likely for your IP to be detected and blocked (along with other possible actions.)

Below is an example of reddit's robots.txt file: https://www.reddit.com/robots.txt 80legs User-agent: 008 Disallow: /

User-Agent: bender Disallow: /my$_s hiny_m etal_a ss$

User-Agent: Gort Disallow: /earth

User-Agent: * Disallow: /*.json Disallow: /*.json-compact Disallow: /*.json-html Disallow: /*.xml Disallow: /*.rss Disallow: /*.i Disallow: /*.embed Disallow: /*/comments/*?*sort= Disallow: /r/*/comments/*/*/c* Disallow: /comments/*/*/c* Disallow: /r/*/submit Disallow: /message/compose* Disallow: /api Disallow: /post Disallow: /submit Disallow: /goto Disallow: /*after= Disallow: /*before= Disallow: /domain/*t= Disallow: /login Disallow: /reddits/search Disallow: /search Disallow: /r/*/search Allow: / User blahblahblah provides a concise description of how to read the robots.txt file: https://www.reddit.com/r/learnprogramming/comments/3l1lcq/how_do_you_find_out_if_a_website_is_scrapable/ - The bot that calls itself 008 (apparently from 80legs) isn't allowed to access anything - bender is not allowed to visit my$_s hiny_m etal_a ss(it's a Futurama joke, the page doesn't actually exist)$ – $Gort isn't allowed to visit Earth(another joke, from The Day the Earth Stood Still)$ – $Other scrapers should avoid checking the API method sor "compose message" or 'search" or the "over 18?" page(because those aren'$ $wildcard category of what the site generally do not want bots to access. You should make sure your scraper does not access any of those$

3 Let's get started!

Now that we've gone through major concepts and tried out a few code snippets, let's hone our Python skills and build two basic bots, one on Wikipedia, and one using Twitter's API.

3.1 6) Tutorial 1: Creating a friendly bot on Wikipedia

Our first use case involves scraping some basic information about technology companies from Wikipedia. Say you are the chief innovation officer of a small city in the San Francisco Bay Area. A number of large-scale local investments in office space have taken place, with space opening up over the next few years. You wish

to be part of the trend of technology companies moving out of San Francisco and Silicon Valley. You have been networking and talking to companies at events and conferences, but would like a more systematic way of identifying companies to focus on.

You notice a list of 179 technology companies based in the San Francisco area on Wikipedia: https://en.wikipedia.org/wiki/Category:Technology_companies_based_in_the_San_Francisco_Bay_Area

Your goal is to scrape basic useful information about each company in a list, into which you can do some summary statistics to identify companies or even industries you are interested in focusing on.

** In particular, you want to know: **

1) what industry they are in
2) where the company is currently headquartered
3) the number of employees
4) website address of the company

This will allow you to know the current and budding tech hubs in the Bay area, get a better sense of your competition, and the number of jobs you can attract to your city. For convenience, you also collate the website addresses of the companies to pull into your list.

3.2 Examining the webpage structure

The first step is to figure out whether and how easily the data you want can be extracted, first by examining the webpage structure in your browser, then on your console.

You can inspect any element in your browser by right-clicking it and selecting inspect, which will bring up the Developer Tools pane.

Typically, you'll first want to identify the element that you want to pull data from. Next, you'll need to figure out a strategy to locate and "crawl" through relevant pages. In a forum, for instance, a bot may be set up to click the "Next page" button once all posts on a single page have been visited and saved. More advanced strategies would include visiting all hyperlinks on every page visited, so that the bot continually updates the list of links to crawl through.

Inspecting the Index page First, you will want to inspect the element associated with each link we want to visit on the index page.

Next, you will want to inspect the element with the data we would like to extract, corresponding to each link on the index page.

Inspecting each individual page In our case, it looks like the format of the data in both the index and individual pages are regular enough for us to be able to parse them programatically. We next confirm this by interacting with both the index and individual pages in our console.

3.3 Interacting with the webpage through the console

After examining the webpage structure through your browser, now it's time to interact with the underlying html code (what you see in the inspect element page) directly in your console. Both processes are useful to coming up with a strategy of how (and whether) data from the website can be scraped.

First, import requests and BeautifulSoup. Downloading a html copy of the site is as simple as:

```
In [24]: from bs4 import BeautifulSoup

         r = requests.get('http://en.wikipedia.org/wiki/Category:Technology_companies_based_in_the_San_
         print(r.content[0:300])
```

```
b'<!DOCTYPE html>\n<html lang="en" dir="ltr" class="client-nojs">\n<head>\n<meta charset="UTF-8"/>\n<tit
```

Once you've downloaded the html file, you'll now want to pass it into BeautifulSoup. BeautifulSoup converts the html file in an easily searchable and navigable structure, which you'll see in our examples below.

```
In [25]: soup = BeautifulSoup(r.content)
         type(soup)
```

```
/Users/dillon/anaconda/lib/python3.5/site-packages/bs4/__init__.py:166: UserWarning: No parser was explic

To get rid of this warning, change this:

 BeautifulSoup([your markup])

to this:

 BeautifulSoup([your markup], "lxml")

  markup_type=markup_type))
```

```
Out[25]: bs4.BeautifulSoup
```

You should have the browser page open side by step, identifying the elements you want to extract using the Inspect element tool, and then using BeautifulSoup's functions to see if you can retrieve them in the console. You can also scroll over each element in the Elements tab to see what they correspond to on the web page.

If you scroll over the div with id "mw-pages" on the Elements tab, you'll see that it corresponds to the entire "Technology companies based in the San Francisco Bay Area" pane.

Let's first try to select this, and confirm we've selected the right element by printing out the result. In the code, we are telling soup to find any elements with the "div" element tag, with id "mw-pages" that we saw in the inspect element pane.

```
In [26]: company_section = soup.findAll("div", {"id": "mw-pages"})
         print(type(company_section))
```

```
<class 'bs4.element.ResultSet'>
```

As we navigate the result returned, we see that it is a "ResultSet", which suggests that it can be retrieved by index. You can also just try it out.

```
In [27]: print(company_section[0])
```

```
<div id="mw-pages">
<h2><span id="Pages_in_category"></span>Pages in category "Technology companies based in the San Francis
<p>The following 199 pages are in this category, out of 199 total. This list may not reflect recent cha
</p><div class="mw-content-ltr" dir="ltr" lang="en"><div class="mw-category"><div class="mw-category-gr
<ul><li><a href="/wiki/HP_3PAR" title="HP 3PAR">HP 3PAR</a></li></ul></div><div class="mw-category-group
<ul><li><a href="/wiki/4D_Inc" title="4D Inc">4D Inc</a></li></ul></div><div class="mw-category-group"><
<ul><li><a href="/wiki/Achronix" title="Achronix">Achronix</a></li>
<li><a href="/wiki/Advanced_Micro_Devices" title="Advanced Micro Devices">Advanced Micro Devices</a></li
<li><a href="/wiki/Aerohive_Networks" title="Aerohive Networks">Aerohive Networks</a></li>
<li><a href="/wiki/Affymetrix" title="Affymetrix">Affymetrix</a></li>
<li><a href="/wiki/Agami_Systems" title="Agami Systems">Agami Systems</a></li>
<li><a href="/wiki/Agilent_Technologies" title="Agilent Technologies">Agilent Technologies</a></li>
<li><a href="/wiki/AirTouch" title="AirTouch">AirTouch</a></li>
<li><a href="/wiki/AKM_Semiconductor,_Inc." title="AKM Semiconductor, Inc.">AKM Semiconductor, Inc.</a><
<li><a href="/wiki/Alexa_Internet" title="Alexa Internet">Alexa Internet</a></li>
<li><a href="/wiki/All_Power_Labs" title="All Power Labs">All Power Labs</a></li>
<li><a href="/wiki/Alphabet_Energy" title="Alphabet Energy">Alphabet Energy</a></li>
<li><a href="/wiki/AlphaSense" title="AlphaSense">AlphaSense</a></li>
<li><a href="/wiki/Altamira_Software" title="Altamira Software">Altamira Software</a></li>
```

AltaVista
Altos Computer Systems</li
ALZA
<a href="/wiki/AMAX_Information_Technologies" title="AMAX Information Technologies">AMAX Information
American Logic Machines<
Amiga Corporation
Antec
Anthera Pharmaceuticals<
Apple Inc.
ASSIA (company)
Audience (company)
AuraOne Systems
Aureal Semiconductor
<a href="/wiki/BioMarin_Pharmaceutical" title="BioMarin Pharmaceutical">BioMarin Pharmaceutical<
BioPharm (US company)
Biosearch Technologies</li
BrightSource Energy
Brderbund
<a class="mw-redirect" href="/wiki/Byte_Shop" title="Byte Shop">B
Cask (company)
Cerego
Cetus Corporation
Cisco Systems
Clean Edge
Clickatell
Complete Genomics
Corsair Components
Covad
<a href="/wiki/Crossbar_(computer_hardware_manufacturer)" title="Crossbar (computer hardware manufac
Cupertino Electric
Cutter Laboratories
Cydrome
Cypress Semiconductor
</
Deeplearning4j
DiscoveRx
DNA2.0
DOER Marine
Dolby Laboratories
Double Robotics
Dust Networks <div class="mw-
<a href="/wiki/Electronics_for_Imaging" title="Electronics for Imaging">Electronics for Imaging<
Energy Recovery Inc.
Enphase Energy
Envivio
Etec Systems, Inc.
Everex
Exidy <h3>F</
<a href="/wiki/Fairchild_Semiconductor" title="Fairchild Semiconductor">Fairchild Semiconductor<
Fortify Software
Foundry Networks
Four-Phase Systems
Foveon <h3>
Genentech
General Magic

Genesys (company)
GlassPoint Solar
GlobalFoundries
Green Charge Networks
Gusto (software) <div c
Handspring (company)
<a href="/wiki/Hercules_Computer_Technology" title="Hercules Computer Technology">Hercules Computer
<a href="/wiki/Hewlett_Packard_Enterprise" title="Hewlett Packard Enterprise">Hewlett Packard Enterp
Hewlett-Packard
Hoopla Software
HP Inc.
<a href="/wiki/Human_Engineered_Software" title="Human Engineered Software">Human Engineered Softwar
Hurricane Electric
Ikanos Communications</l
Impax Laboratories
Intel
Intel Security
<a href="/wiki/Internet_Systems_Consortium" title="Internet Systems Consortium">Internet Systems Con
InvenSense
InVision Technologies
</
Jennerex
Juniper Networks <div c
<a href="/wiki/KleenSpeed_Technologies" title="KleenSpeed Technologies">KleenSpeed Technologies<
Kosan Biosciences <di
Lam Research
LeapFrog Enterprises
Lexar
<a href="/wiki/Lightwave_Electronics_Corporation" title="Lightwave Electronics Corporation">Lightwav
<a href="/wiki/Linear_Integrated_Systems" title="Linear Integrated Systems">Linear Integrated System
Linear Technology <di
Made In Space, Inc.
Makani Power
Maxim Integrated
<a class="mw-redirect" href="/wiki/McAfee,_Inc." title="McAfee, I
<a href="/wiki/McCune_Audio/Video/Lighting" title="McCune Audio/Video/Lighting">McCune Audio/Video/L
Media Vision
Medivation
Meka Robotics
<a href="/wiki/Mendel_Biotechnology,_Inc." title="Mendel Biotechnology, Inc.">Mendel Biotechnology,
Meru Networks
MeWe
<a href="/wiki/Meyer_Sound_Laboratories" title="Meyer Sound Laboratories">Meyer Sound Laboratories</
MIPS Technologies
MLab
Mozilla Corporation
</d
Nanosolar
National Semiconductor</li
Navigenics
<a href="/wiki/NeoPhotonics_Corporation" title="NeoPhotonics Corporation">NeoPhotonics Corporation</
Netscape
<a class="mw-redirect" href="/wiki/Netscape_Communications_Corpor
<a href="/wiki/Network_Computing_Devices" title="Network Computing Devices">Network Computing Device
NeXT
Next Thing Co.

NovaBay Pharmaceuticals<
NStigate Games
Numenta <h
Oak Technology
Oberheim Electronics
OLogic
OpenAI
Oracle Corporation
<a href="/wiki/Osborne_Computer_Corporation" title="Osborne Computer Corporation">Osborne Computer C
OSIsoft <h
P.A. Semi
<a class="mw-redirect" href="/wiki/PA_Semiconductor" title="PA Se
Palm, Inc.
Palo Alto Networks
<a href="/wiki/Peninsula_Engineering_Group,_Inc." title="Peninsula Engineering Group, Inc.">Peninsul
Peribit Networks
Pixo
Polycom
Primus Power
Processor Technology
Prosetta
Pyramid Technology
Qualcomm Atheros
Quantum Effect Devices</li
Quark Pharmaceuticals
Qume <h3>R</h3>
Rambus
Recommind
Redwood Robotics <div c
Sangfor Technologies
Seagate Technology
Sensory, Inc.
Shugart Associates
Sidecar (company)
Silego Technology Inc.</li
Silicon Graphics
Siluria Technologies
Skymind
Sling Media
SmugMug
SolarCity
Solectron
<a href="/wiki/Solido_Design_Automation" title="Solido Design Automation">Solido Design Automation</
SoloPower
Solyndra
Stem Cell Theranostics</li
SunPower
Sunrun
Supertek Computers
Sybase
Symyx Technologies
Synaptics <div class="mw-category-gr
Tabula (company)
Talari Networks

```
<li><a href="/wiki/Tandem_Computers" title="Tandem Computers">Tandem Computers</a></li>
<li><a href="/wiki/TeleVideo" title="TeleVideo">TeleVideo</a></li>
<li><a href="/wiki/Tengen_(company)" title="Tengen (company)">Tengen (company)</a></li>
<li><a href="/wiki/Theranos" title="Theranos">Theranos</a></li>
<li><a href="/wiki/Thoratec" title="Thoratec">Thoratec</a></li>
<li><a href="/wiki/Tout_(company)" title="Tout (company)">Tout (company)</a></li></ul></div><div class="
<ul><li><a href="/wiki/Ubiquitous_Energy" title="Ubiquitous Energy">Ubiquitous Energy</a></li>
<li><a href="/wiki/Umtech" title="Umtech">Umtech</a></li>
<li><a href="/wiki/UTStarcom" title="UTStarcom">UTStarcom</a></li></ul></div><div class="mw-category-gr
<ul><li><a href="/wiki/Velodyne_Inc." title="Velodyne Inc.">Velodyne Inc.</a></li>
<li><a href="/wiki/Vivante_Corporation" title="Vivante Corporation">Vivante Corporation</a></li>
<li><a href="/wiki/Volterra_Semiconductor" title="Volterra Semiconductor">Volterra Semiconductor</a></li
<li><a href="/wiki/VPL_Research" title="VPL Research">VPL Research</a></li>
<li><a href="/wiki/VSee" title="VSee">VSee</a></li>
<li><a href="/wiki/VW_Electronics_Research_Laboratory" title="VW Electronics Research Laboratory">VW Ele
<ul><li><a href="/wiki/@WalmartLabs" title="@WalmartLabs">@WalmartLabs</a></li>
<li><a href="/wiki/Willow_Garage" title="Willow Garage">Willow Garage</a></li>
<li><a href="/wiki/Wind_River_Systems" title="Wind River Systems">Wind River Systems</a></li></ul></div>
<ul><li><a href="/wiki/Xilinx" title="Xilinx">Xilinx</a></li></ul></div><div class="mw-category-group">
<ul><li><a href="/wiki/Yahoo!" title="Yahoo!">Yahoo!</a></li></ul></div><div class="mw-category-group">
<ul><li><a href="/wiki/Zenefits" title="Zenefits">Zenefits</a></li>
<li><a href="/wiki/Zscaler" title="Zscaler">Zscaler</a></li></ul></div></div></div>
</div>
```

You can see at the start of the element retrieved that it is indeed a division with id “mw-pages” - we can confirm by browsing the text that we’ve selected the correct element. Next, let’s retrieve each section (corresponding to each alphabet), now searching the company section with class type “mw-category-group”.

```
In [28]: each_alphabet = company_section[0].find_all("div", {"class":"mw-category-group"})
         print(len(each_alphabet))
         print(each_alphabet[0])
```

```
28
<div class="mw-category-group"><h3>3</h3>
<ul><li><a href="/wiki/HP_3PAR" title="HP 3PAR">HP 3PAR</a></li></ul></div>
```

Finally, within each section, we want to pull out the individual hyperlinks corresponding to each company. Let’s use the second element in the index (the letter “A” instead of the category group for “3”) as it has more than one company.

```
In [29]: alphabet_a = each_alphabet[1]
         print(alphabet_a)
```

```
<div class="mw-category-group"><h3>4</h3>
<ul><li><a href="/wiki/4D_Inc" title="4D Inc">4D Inc</a></li></ul></div>
```

We next want to select all elements with the “li” tag, and print them out to make sure they correspond to what we expect to see on the page.

```
In [30]: company_list = alphabet_a.find_all("li")
         for i in company_list:
             print("")
             print(i)
```

```
<li><a href="/wiki/4D_Inc" title="4D Inc">4D Inc</a></li>
```

If we select one company and print it out, we can see we’re pretty close.

```
In [31]: one_company = company_list[0]
         print(one_company)
```

```
<li><a href="/wiki/4D_Inc" title="4D Inc">4D Inc</a></li>
```

We can also select the next child element by doing the following:

```
In [32]: one_company.a
```

```
Out[32]: <a href="/wiki/4D_Inc" title="4D Inc">4D Inc</a>
```

And finally, get the attributes associated with the "a" hyperlink tag, which returns a Python dictionary.

```
In [33]: one_company.a.attrs
```

```
Out[33]: {'href': '/wiki/4D_Inc', 'title': '4D Inc'}
```

Now that we've received the element containing the element we want, we can also print out its parents to view the position within the html "tree."

```
In [34]: print(type(one_company.a.parents))
         for i in one_company.a.parents:
             print(i.name)
```

```
<class 'generator'>
li
ul
div
div
div
div
div
div
div
div
body
html
[document]
```

This lets you see the different nested elements you'll need to traverse to get to the element you need. In many cases, you'll use explicit selection of the element together with the find_all command to isolate the element you need.

Finally, let's write a loop to store all of our desired hyperlink dictionaries in a single Python list.

```
In [35]: link_list = []
         for each_section in company_section:
             company_list = each_section.find_all("li")
             for each_company in company_list:
                 new_dict = each_company.a
                 link_list.append(new_dict)
         print(len(link_list))
```

```
199
```

We now have a list of 175 hyperlinks to loop through for our next section.

Time for a challenge! To make sure that everyone is on the same page (and to give you a little more practice dealing with HTML), let's partner up with the person next to you and try challenge B, on using html, in the challenges directory.

Now using the list, let's load the first page and locate the text elements we want

```
In [36]: example_site = link_list[0]
         print(example_site)

         company_page = requests.get("http://wikipedia.org" + example_site['href'])
```

```
<a href="/wiki/HP_3PAR" title="HP 3PAR">HP 3PAR</a>
```

```
In [37]: print(company_page.content[0:200])
```

```
b'<!DOCTYPE html>\n<html lang="en" dir="ltr" class="client-nojs">\n<head>\n<meta charset="UTF-8"/>\n<tit
```

In your browser, you should be using inspect element to confirm the position of the desired element in the html tree. We can see the element is a table with class name "infobox vcard". Let's try to select this next. First, we need the html document into soup as we did before. (We convert to string just to allow us to print the first 500 charactes of the text here.)

```
In [38]: soup = BeautifulSoup(company_page.content)
         info_box = soup.find("table", {"class": "infobox vcard"})
         print(str(info_box)[0:500])
```

```
<table class="infobox vcard" style="width:22em">
<caption class="fn org">HP 3PAR</caption>
<tr>
<th scope="row" style="padding-right:0.5em;">
<div style="padding:0.1em 0;line-height:1.2em;"><a href="/wiki/Types_of_business_entity" title="Types of
</th>
<td class="category" style="line-height:1.35em;">Subsidiary</td>
</tr>
<tr>
<th scope="row" style="padding-right:0.5em;">Industry</th>
<td class="category" style="line-height:1.35em;"><a href="/wiki/Data_storage_dev

/Users/dillon/anaconda/lib/python3.5/site-packages/bs4/__init__.py:166: UserWarning: No parser was explic

To get rid of this warning, change this:

 BeautifulSoup([your markup])

to this:

 BeautifulSoup([your markup], "lxml")

  markup_type=markup_type))
```

Now, using the various tools we've had before, we can drill down to the specific element containing the data we need. As before, we select and print a single row to help guide the process.

```
In [39]: table_elements = info_box.find_all("tr")
         one_row = table_elements[0]
         print(one_row)
```

```
<tr>
<th scope="row" style="padding-right:0.5em;">
<div style="padding:0.1em 0;line-height:1.2em;"><a href="/wiki/Types_of_business_entity" title="Types of
</th>
<td class="category" style="line-height:1.35em;">Subsidiary</td>
</tr>
```

First, let's try to select the element containing the variable name "Type".

```
In [40]: print(one_row.th)
         print("")
         print(one_row.th.div)
         print("")
         print(one_row.th.div.text)
```

```
<th scope="row" style="padding-right:0.5em;">
<div style="padding:0.1em 0;line-height:1.2em;"><a href="/wiki/Types_of_business_entity" title="Types of
</th>

<div style="padding:0.1em 0;line-height:1.2em;"><a href="/wiki/Types_of_business_entity" title="Types of

Type
```

Next, let's select the element containing the variable value, in this case "Subsidiary".

```
In [41]: print(one_row.td)
         print("")
         print(one_row.td.text)
```

```
<td class="category" style="line-height:1.35em;">Subsidiary</td>

Subsidiary
```

Now, let's loop through all rows to get all data that's available on the company. Depending on how well-structured the data is, this can be something of a trial and error process.

```
In [42]: for one_row in table_elements:
             print(one_row.th.div.text + ": " + one_row.td.text)
```

```
Type: Subsidiary
```

```
        ---------------------------------------------------------------------------

        AttributeError                            Traceback (most recent call last)

        <ipython-input-42-5678f42391a1> in <module>()
          1 for one_row in table_elements:
    ----> 2     print(one_row.th.div.text + ": " + one_row.td.text)


        AttributeError: 'NoneType' object has no attribute 'text'
```

We get an AttributeError for the "NoneType" object due to some of the "th" elements being empty. If we do some simple Exception capturing, we can get the loop to run through.

```
In [43]: for one_row in table_elements:
             try:
                 print(one_row.th.div.text + ": " + one_row.td.text)
             except Exception:
                 continue
```

```
Type: Subsidiary
Area served: Worldwide
Key people: David C. Scott
(President), (CEO) & (Director)
Operating income: US$ -3.33 million (FY10)
Net income: US$ -3.18 million (FY10)
Number of employees: 657 (FY10)
```

Now that we have the data we need, let's store it in a Python dictionary.

```
In [44]: new_dict = {}
         for one_row in table_elements:
             try:
                 print(one_row.th.div.text + ": " + one_row.td.text)
                 new_dict[one_row.th.div.text] = one_row.td.text
             except Exception:
                 continue
```

```
Type: Subsidiary
Area served: Worldwide
Key people: David C. Scott
(President), (CEO) & (Director)
Operating income: US$ -3.33 million (FY10)
Net income: US$ -3.18 million (FY10)
Number of employees: 657 (FY10)
```

We can browse the dictionary to make sure it is capturing the data correctly.

```
In [45]: print(new_dict.keys())
         print(new_dict)
```

```
dict_keys(['Key people', 'Type', 'Number of employees', 'Area served', 'Operating income', 'Net income']
{'Key people': 'David C. Scott\n(President), (CEO) & (Director)', 'Type': 'Subsidiary', 'Number of emplo
```

Let's quickly rehash what we did. We first extracted a list of hyperlinks from our index page, storing in our link_list variable. Next, we visited one of the pages, pulled its html into soup, and extracted the data from the element with class name "infobox vcard" into our new_dict variable.

The next step is to write an overall loop so that we can collect the "infobox vcard" data for all elements in our list. info_box = soup.find("table", {"class": "infobox vcard"})

```
In [46]: list_of_dicts = []

         for each_link in link_list[0:3]:
             print("")
             print(each_link)
             print("")
             company_page = requests.get("http://wikipedia.org" + each_link['href'])
             soup = BeautifulSoup(company_page.content)
             info_box = soup.find("table", {"class": "infobox vcard"})
             table_elements = info_box.find_all("tr")
```

```
            new_dict = {}
            new_dict['Company_name'] = each_link['title']
            for one_row in table_elements:
                try:
                    print(one_row.th.div.text + ": " + one_row.td.text)
                    # we convert to string as a precaution to make sure more complex elements are stor
                    new_dict[one_row.th.div.text] = str(one_row.td.text)
                except Exception:
                    continue
            # add the dictionary after we've added all variable names and values to each dictionary
            list_of_dicts.append(new_dict)

        print("")
        print(len(list_of_dicts))
```

```
<a href="/wiki/HP_3PAR" title="HP 3PAR">HP 3PAR</a>

Type: Subsidiary
Area served: Worldwide
Key people: David C. Scott
(President), (CEO) & (Director)
Operating income: US$ -3.33 million (FY10)
Net income: US$ -3.18 million (FY10)
Number of employees: 657 (FY10)

<a href="/wiki/4D_Inc" title="4D Inc">4D Inc</a>

Formerly called: ACI US
Type: Private
Areas served: USA & English-speaking Canada

<a href="/wiki/Achronix" title="Achronix">Achronix</a>

Type: Private
Key people: Robert Blake (CEO), John Lofton Holt (Chairman), Rahul Nimaiyyar (VP HW Engineering), Kamal
Number of employees: <200

3

/Users/dillon/anaconda/lib/python3.5/site-packages/bs4/__init__.py:166: UserWarning: No parser was explic

To get rid of this warning, change this:

 BeautifulSoup([your markup])

to this:

 BeautifulSoup([your markup], "lxml")

  markup_type=markup_type))
```

Let's browse our list_of_dicts object to make sure it contains the data we need.

```
In [47]: print(list_of_dicts)
```

```
[{'Company_name': 'HP 3PAR', 'Key people': 'David C. Scott\n(President), (CEO) & (Director)', 'Type': 'S
```

Typically, a “friendly” bot would try to space out the number of requests (in addition to not scraping the pages robots.txt disallows and obeying the general terms of service) so that the server can manage its traffic. One simple way to do this is to add a time.sleep command into your loop.

As the entire class will be sharing the same IP, it’s recommended that you add a longer wait time and limit the number of companies from link_list you scrape while in class.

```
In [48]: import time

         list_of_dicts = []

         for each_link in link_list[0:3]:
             print("")
             wait_time = 1
             print('waiting ' + str(wait_time) + "s...")
             time.sleep(1)
             print("")
             print(each_link)
             print("")
             company_page = requests.get("http://wikipedia.org" + each_link['href'])
             soup = BeautifulSoup(company_page.content)
             info_box = soup.find("table", {"class": "infobox vcard"})
             table_elements = info_box.find_all("tr")
             new_dict = {}
             new_dict['Company_name'] = each_link['title']
             for one_row in table_elements:
                 try:
                     print(one_row.th.div.text + ": " + one_row.td.text)
                     # we convert to string as a precaution to make sure more complex elements are stor
                     new_dict[one_row.th.div.text] = str(one_row.td.text)
                 except Exception:
                     continue
             # add the dictionary after we've added all variable names and values to each dictionary
             list_of_dicts.append(new_dict)
```

```
waiting 1s...

<a href="/wiki/HP_3PAR" title="HP 3PAR">HP 3PAR</a>

Type: Subsidiary
Area served: Worldwide
Key people: David C. Scott
(President), (CEO) & (Director)
Operating income: US$ -3.33 million (FY10)
Net income: US$ -3.18 million (FY10)
Number of employees: 657 (FY10)

waiting 1s...

<a href="/wiki/4D_Inc" title="4D Inc">4D Inc</a>

Formerly called: ACI US
Type: Private
Areas served: USA & English-speaking Canada

waiting 1s...
```

```
<a href="/wiki/Achronix" title="Achronix">Achronix</a>

Type: Private
Key people: Robert Blake (CEO), John Lofton Holt (Chairman), Rahul Nimaiyyar (VP HW Engineering), Kamal
Number of employees: <200

/Users/dillon/anaconda/lib/python3.5/site-packages/bs4/__init__.py:166: UserWarning: No parser was explic

To get rid of this warning, change this:

 BeautifulSoup([your markup])

to this:

 BeautifulSoup([your markup], "lxml")

  markup_type=markup_type))
```

Now we have our list of dictionaries containing the data we want for each company. If you browse a few of the company pages, you'll notice that the number of variables each "infobox vcard" has is not always the same. Some dictionaries will have fields that others don't.

To store this data flexibly, we can iterate through all our dictionaries and collect all keys from them.

```
In [49]: key_list = []
         for each_dict in list_of_dicts:
             for key in each_dict.keys():
                 key_list.append(key)
         print(key_list)
```

```
['Company_name', 'Key people', 'Type', 'Number of employees', 'Net income', 'Operating income', 'Area se
```

Next, we convert the list to a set to remove all repeat keys. This then contains all unique keys across our dictionaries.

```
In [50]: key_set = set(key_list)
         print(key_set)
```

```
{'Company_name', 'Areas served', 'Formerly called', 'Operating income', 'Area served', 'Key people', 'Ty
```

We convert the set back to a list (and sort it) as csv.DictWriter takes a list for its fieldnames parameter.

```
In [51]: final_key_list = sorted(list(key_set))
         print(final_key_list)
```

```
['Area served', 'Areas served', 'Company_name', 'Formerly called', 'Key people', 'Net income', 'Number o
```

With our complete key list, we can now write our dictionary into a csv file.

```
In [52]: import csv

         outpath= "../data/02_companies.csv"
         outfile = open(outpath, 'w')

         writer = csv.DictWriter(outfile, fieldnames=final_key_list, dialect='excel')

         writer.writeheader()
```

```
        for row in list_of_dicts:
            writer.writerow(row)
            print(row)
        outfile.close()
        print("done")
```

```
{'Company_name': 'HP 3PAR', 'Key people': 'David C. Scott\n(President), (CEO) & (Director)', 'Type': 'Su
{'Company_name': '4D Inc', 'Formerly called': 'ACI US', 'Areas served': 'USA & English-speaking Canada',
{'Company_name': 'Achronix', 'Key people': 'Robert Blake (CEO), John Lofton Holt (Chairman), Rahul Nimai
done
```

You can open the file in Excel to view the data. Congrats, you just scraped valuable data for your project off the web!