**API Access**

This is API Access, aka getting data from websites that actually intend you to do so.

A small sampling of APIs:

1. Twitter
2. Google Maps
3. Facebook
4. Watson
5. Linkedin

Etc.…….

<http://www.programmableweb.com/news/most-popular-apis-least-one-will-surprise-you/2014/01/23>

APIs have a couple of huge advantages over web scraping. The first is that the query you give will be structured – you will ask the API for specific data. The second is that the response will also will be structured – the API will give you back information that you always parse in the same way.

The disadvantages are mainly two. First, APIs generally rate-limit their users, so you have a cap (usually per hour or per day) on how much data you can scrape. Second, although better than the inconsistency of web scraping, there is no one standard form of API – each website constructs its own from thin air, with its own quirks, and there is a learning curve to each.

**Google Places API – Geocoding**

In this exercise, we start with a 10mb csv containing a bunch of (very) unclean location data. Our goal is to take this messy data, and output as many cleaned locations as we can.

To do so, we make use of the Google Places API. Essentially, we’re going to feed the API the messy location names, and it’s going to return clean locations (where it can). This is roughly equivalent to typing each piece of data into Google Maps, on your browser, and letting Google try to figure out what the hell you meant.

**Setup – Getting the Data**

1. First, let’s grab the csv: download “Chai\_affil.csv” from <https://github.com/bellmast/WebScrapingClass>
2. If you take a look at the file, it has ~150,000 rows, and 5 columns. The first column is a unique ID, and the next four columns are the super messy location data.

**Setup – Credentials for Google API**

1. Next, we need to get set up with an API key. This is how Google tracks who’s using their service, and it’s also how they rate-limit.
2. Go to <https://console.developers.google.com>
3. Click on “More” under Google Maps APIs, and go to Google Places API Web Service
4. It’ll ask you to Create Project, click on that.
5. Project name doesn’t matter: click ‘No’ on the first checkbox, ‘Yes’ on the second (agreeing to terms), hit create.
6. Now hit Enable.
7. Now do “Go to Credentials”
8. Select Web Server from the dropdown
9. Leave IP address field blank
10. Hit Create API Key
11. Copy the new key to your clipboard, save it somewhere (even a Python window)

**Testing – Accessing the API**

1. Search for any given place using the following URL structure (I’m searching for “Bernal Heights, San Francisco”, here) (make sure to replace the key at the end with your own): maps.googleapis.com/maps/api/place/textsearch/json?query=Bernal Heights, San Francisco&key=###YOUR#KEY#GOES#HERE###
2. If it succeeded, you’ll see a structured json webpage, like this: 
3. We want to access this same data through the Python shell. Start by importing a couple of libraries:

>>> import json

>>> import requests

* 1. The first library, **json**, allows us to read the data structure that we see. We’ll parse it in an identical way to dictionaries
  2. The second library, **requests** (which you may have to ‘pip install requests’ through the command line / terminal), is similar to **urllib2** that we used last week. This library is a little harder to use in the shell, but works better for API calls.

1. Next, let’s construct our URL

>>> baseUrl = "https://maps.googleapis.com/maps/api/place/textsearch/json?query="

>>> query = "Bernal Heights Sang Frannn"

>>> endUrl = "&key="

>>> ourKey = "AIzaSyAewX85GT\_T2IilEoBBlZDXCX77XHro0mY" #####Change this to your key!!!!

>>> finalUrl = baseUrl+query+endUrl+ourKey

>>> with requests.Session() as session:

request = session.get(finalUrl)

outputText = json.loads(request.text)

* 1. In the first handful of lines, we’re just constructing the same URL that we had in our web browser
  2. Then, with requests, we’re visiting that webpage with Python and loading the data with the json library.
  3. Just as a check, you can call “outputText” to see the same exact thing we had in our webpage

1. We navigate this outputText just like a dictionary:

>>> outputText["results"][0]["name"]

u'CHC Wellness'

>>> outputText["results"][1]["name"]

u'City Harvest Church'

>>> outputText["status"]

u'OK'

**Compiling – Turning “mess” into “yes!”**

1. Finally, grab the script “affil geocoder cleaned classExample.py” from <https://github.com/bellmast/WebScrapingClass>
2. I’ve commented out this code pretty thoroughly, but we’re going to go through it together in class (writing it fresh together would take too long for the time we have!)
3. One last note – your initial key is rate limited to 1,000 requests per day, and each “search” request we’re running takes up 10 requests. So if you run this code, you’ll initially only be able to geocode 100 locations per day!
   1. To get around this, you can first authenticate your key by providing a credit card here: <https://developers.google.com/places/web-service/usage>. **This is just to verify your identity, they absolutely do not charge you**. I can vouch for this personally, with 4 different credit cards. This increases your limit to 150,000 requests per day, so that you can do 15,000 search requests per day!
   2. If you want even more, you’ll have to use a separate Google account, and a separate credit card, to verify another key. You can add these additional keys at the top of the code, so that you can do 30,000, 45,000, 60,000 requests per day (etc.).
4. Finally, just to make sure your coding is running correctly, you can find “affil\_geocoding\_example.csv” in the github, which is a partially completed file (I believe I ran it for about 400 lines).