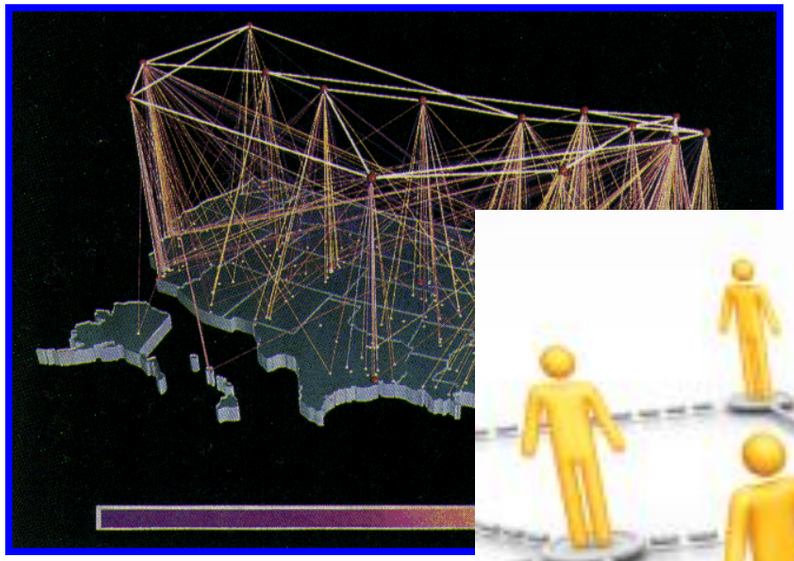
https://bmtgoncalves.github.io/EABDA17/

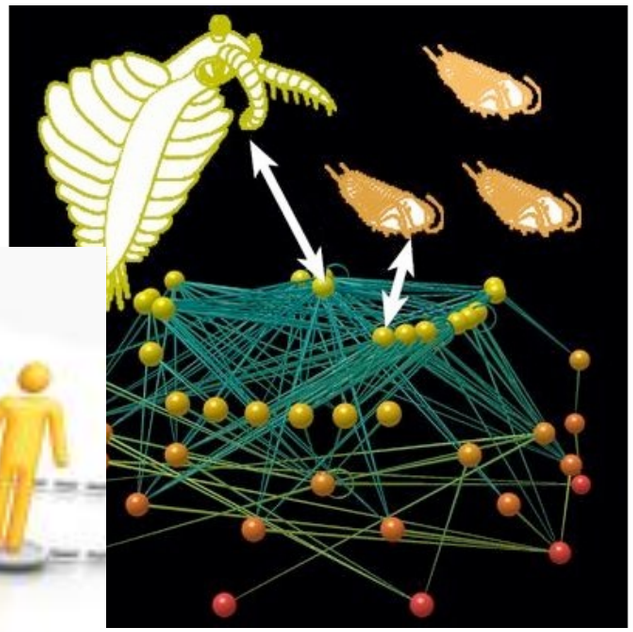
Data Mining and Spatial Analysis

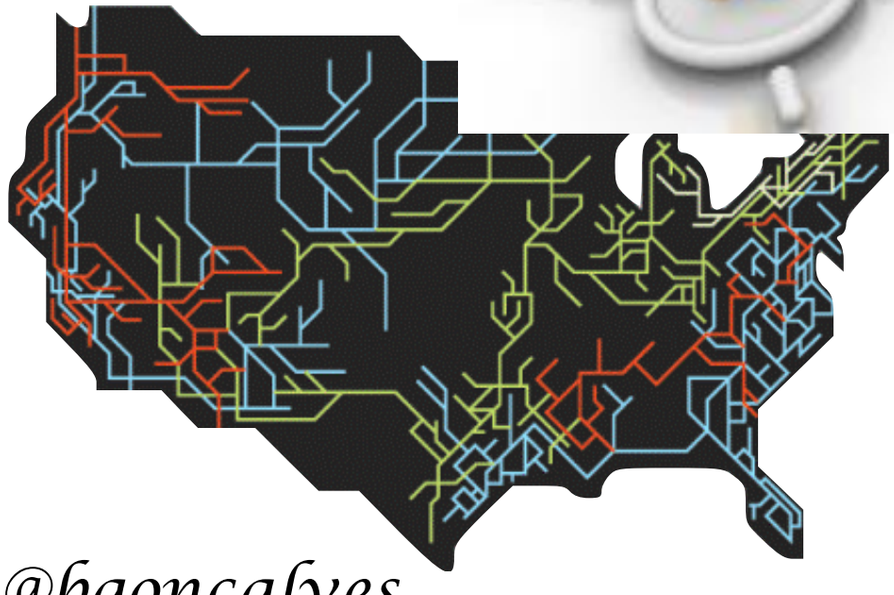
Bruno Gonçalves
www.bgoncalves.com

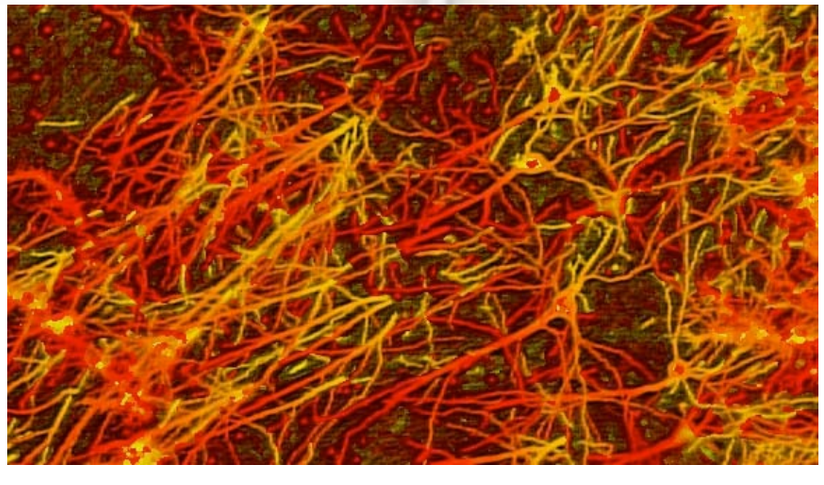
Requirements

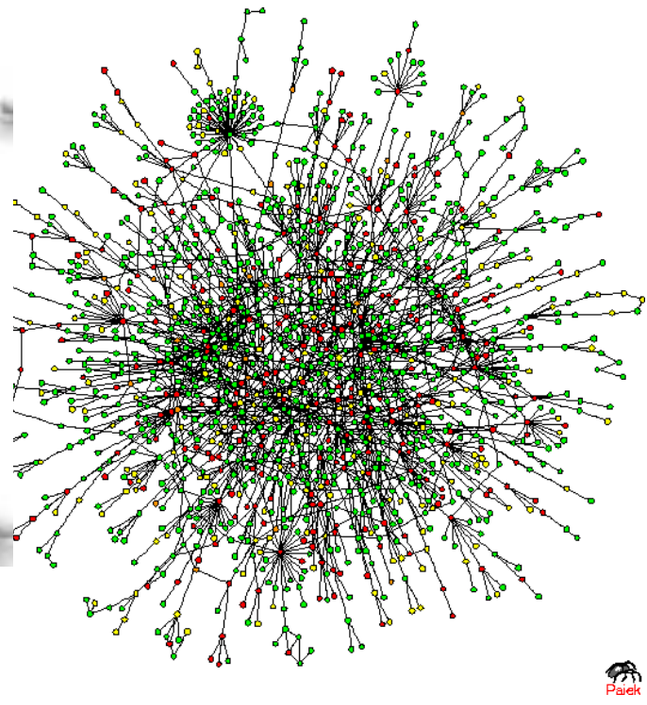
https://bmtgoncalves.github.io/EABDA17/



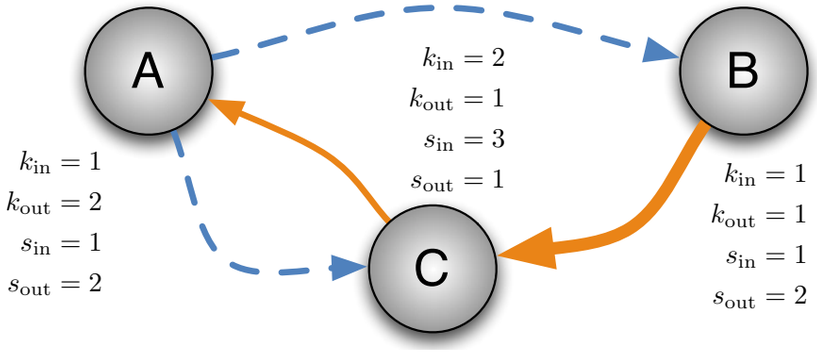


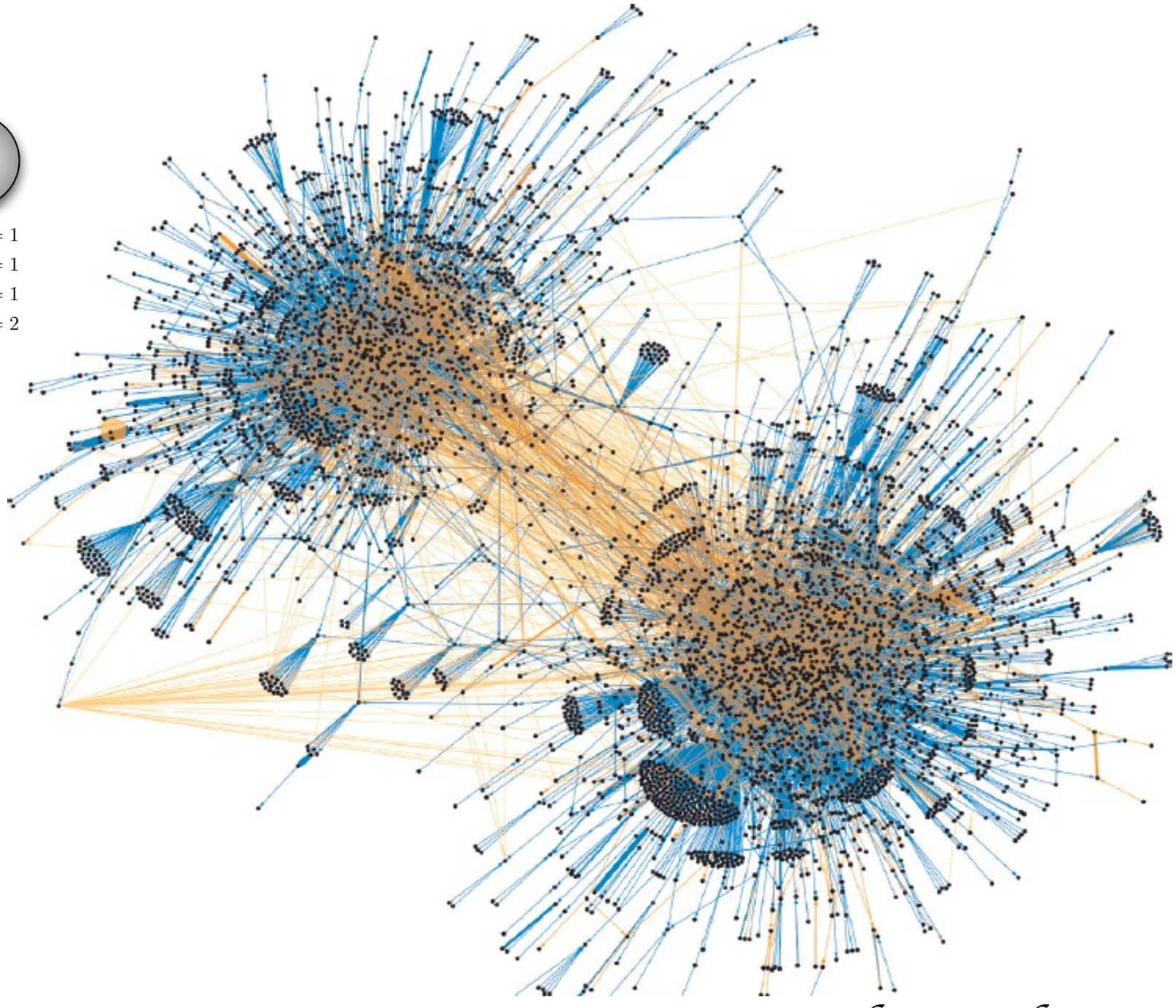





Palek

Information Flow





NetworkX

- High productivity software for complex networks
- Simple Python interface
- Four types of graphs supported:
    - Graph - UnDirected
    - DiGraph - Undirected
    - MultiGraph - Multi-edged Graph
    - MultiDiGraph - Directed Multigraph
- Similar interface for all types of graphs
- Nodes can be any type of Python object - Practical way to manage relationships

Growing Graphs

- .add_node(node_id) Add a single node with ID node_id
- .add_nodes_from() Add a list of node ids
- .add_edge(node_i, node_j) Adds an edge between node_i and node_j
- .add_edges_from() Adds a list of edges. Individual edges are represented by tuples
- .remove_node(node_id)/.remove_nodes_from() Removing a node removes all associated edges
- .remove_edge(node_i, node_j)/.remove_edges_from()

Graph Properties

- .nodes() Returns the list of nodes

- .edges() Returns the list of edges

- .degree() Returns a dict with each nodes degree .in_degree()/.out_degree() returns dicts with in/out degree for DiGraphs

- .is_connected() Returns true if the node is connected

- .is_weakly_connected()/.is_strongly_connected() for DiGraph

- .connected_components() A list of nodes for each connected component

NetworkX - Example

```
import networkx as NX
import numpy as np
from collections import Counter
import matplotlib.pyplot as plt

def BarabasiAlbert(N=1000000):
    G = NX.Graph()

    nodes = range(N)
    G.add_nodes_from(nodes)

    edges = [0,1,1,2,2,0]

    for node_i in range(3, N):
        pos = np.random.randint(len(edges))
        node_j = edges[pos]

        edges.append(node_i)
        edges.append(node_j)

    edges = zip(nodes, edges[1::2])

    G.add_edges_from(edges)

    return G
```

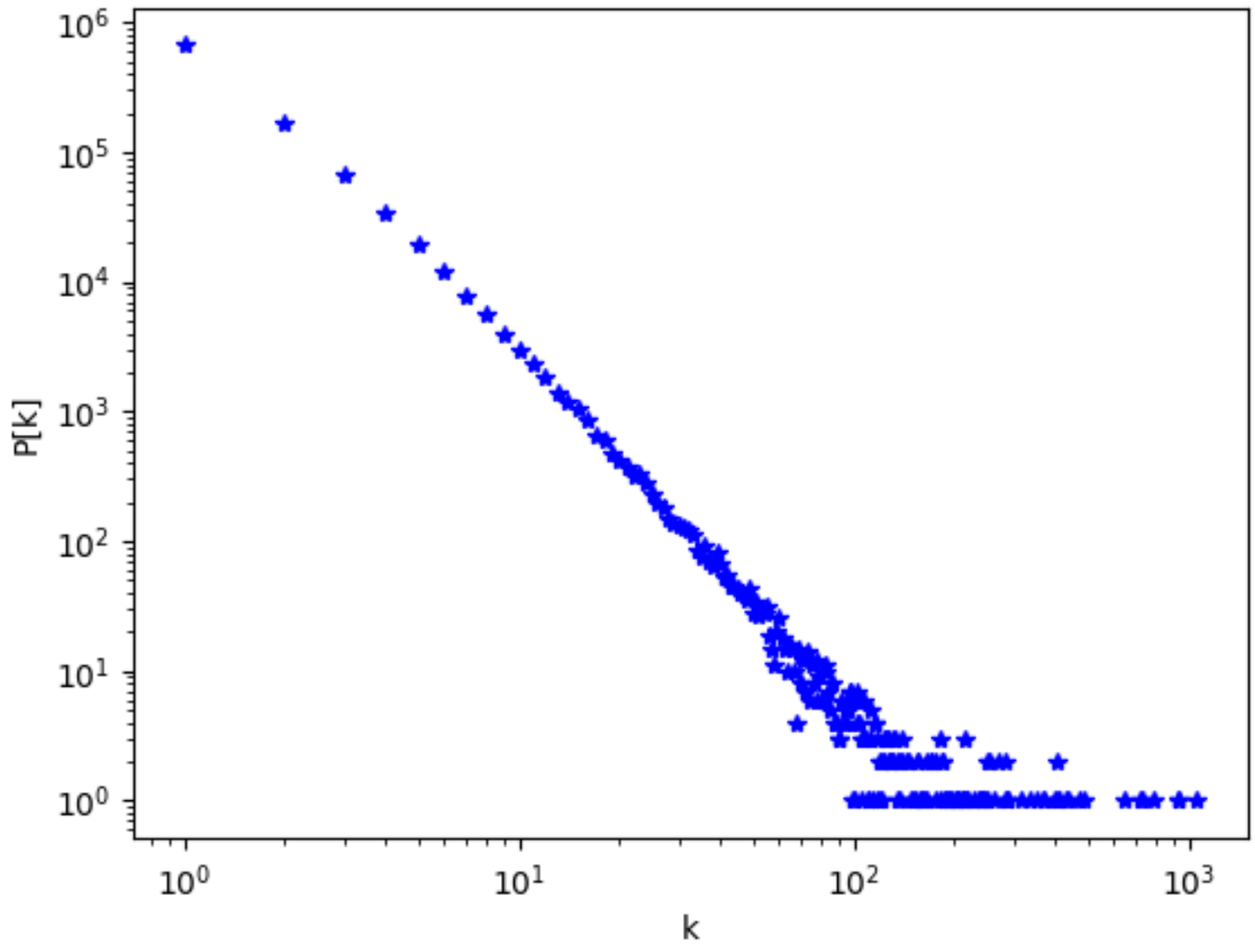

networkx_demo.py

NetworkX - Example

```
import networkx as NX
import numpy as np
from collections import Counter
import matplotlib.pyplot as plt

(…)

net = BarabasiAlbert()

degrees = net.degree()
Pk = np.array(list(Counter(degrees.values()).items()))

plt.loglog(Pk.T[0], Pk.T[1], 'b*')
plt.xlabel('k')
plt.ylabel('P[k]')
plt.savefig('Pk.png')
plt.close()

print("Number of nodes:", net.number_of_nodes())
print("Number of edges:", net.number_of_edges())
```


networkx_demo.py

NetworkX - Example



Snowball Sampling

- Commonly used in Social Science and Computer Science

    1. Start with a single node (or small number of nodes)

    2. Get "friends" list

    3. For each friend get the "friend" list

    4. Repeat for a fixed number of layers or until enough users have been connected

- Generates a connected component from each seed

- Quickly generates a *lot* of data/API calls

Snowball Sampling

```python
import networkx as NX

def snowball(net, seed, max_depth = 3, maxnodes=1000):
    seen = set()
    queue = set()

    queue.add(seed)
    queue2 = set()

    for _ in range(max_depth+1):
        while queue:
            user_id = queue.pop()
            seen.add(user_id)

            NN = net.neighbors(user_id)

            for node in NN:
                if node not in seen:
                    queue2.add(node)

        queue.update(queue2)
        queue2 = set()

    return seen

net = NX.connected_watts_strogatz_graph(10000, 4, 0.01)
neve = snowball(net, 0)

print(neve)
```

snowball.py

Authentication

Authentication Methodologies

- Much of the content available online is only accessible to specific individuals for privacy, copyright protection, etc…

- Three main ways of authenticating users:

    - BasicAuth - The first and most basic one. Plain text user name and password sent to the server

    - OAuth 1 - Developed by a consortium of Industry leaders to provide transparent and secure authentication.

    - OAuth 2 - An improvement on OAuth 1 designed to allow users to more easily share they're content on social media, etc…

    - OpenID - A predecessor to OAuth that has gone out of favor.

BasicAuth

http://requests.readthedocs.org/en/latest/

- "The mother of all authentication protocols"
- Insecure but easy to use with standard implementations in all networking tools
- In particular, in requests:
  - requests.get(url, auth=("user", "pass")) open the given url and authenticate with username="user" and password="pass"

```
import requests
import sys

url = "http://httpbin.org/basic-auth/user/passwd"

request = requests.get(url, auth=("user", "passwd"))

if request.status_code != 200:
    print("Error found", request.get_code(),
file=sys.stderr)

content_type = request.headers["content-type"]

response = request.json()

if response["authenticated"]:
    print("Authentication Successful")
```

basic_auth.py

OAuth 1

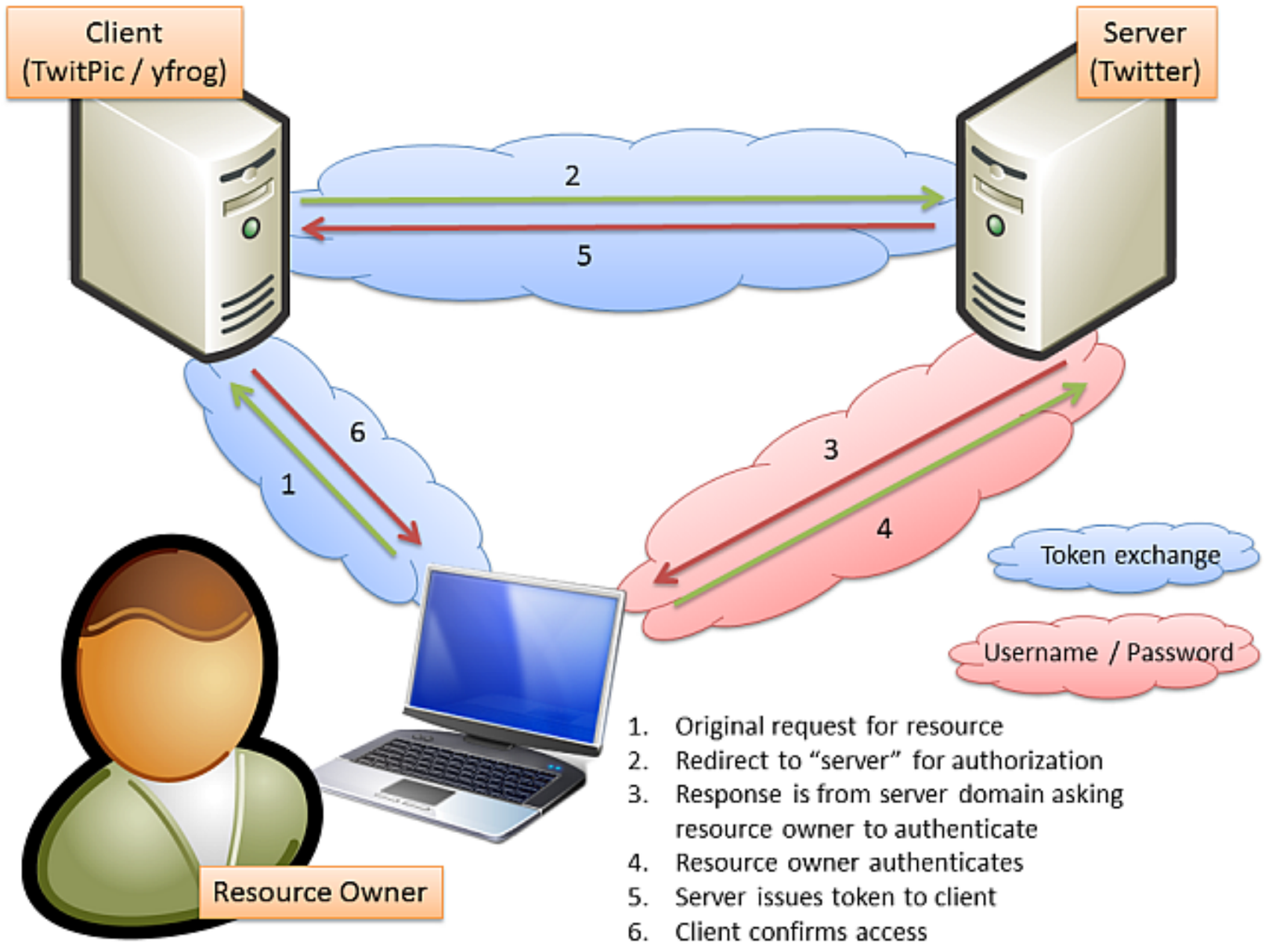
http://hueniverse.com/oauth/
https://tools.ietf.org/html/rfc5849

- "An open protocol to allow secure authorization in a simple and standard method from web, mobile and desktop applications."

- The idea is to allow for a safe way to share privileges without divulging private credentials
  - Give XPTO Application permission to post to your Twitter account without having to trust the developers of XPTO with your username/password and while being able to unilaterally revoke privileges.

OAuth 1

http://hueniverse.com/oauth/
https://tools.ietf.org/html/rfc5849



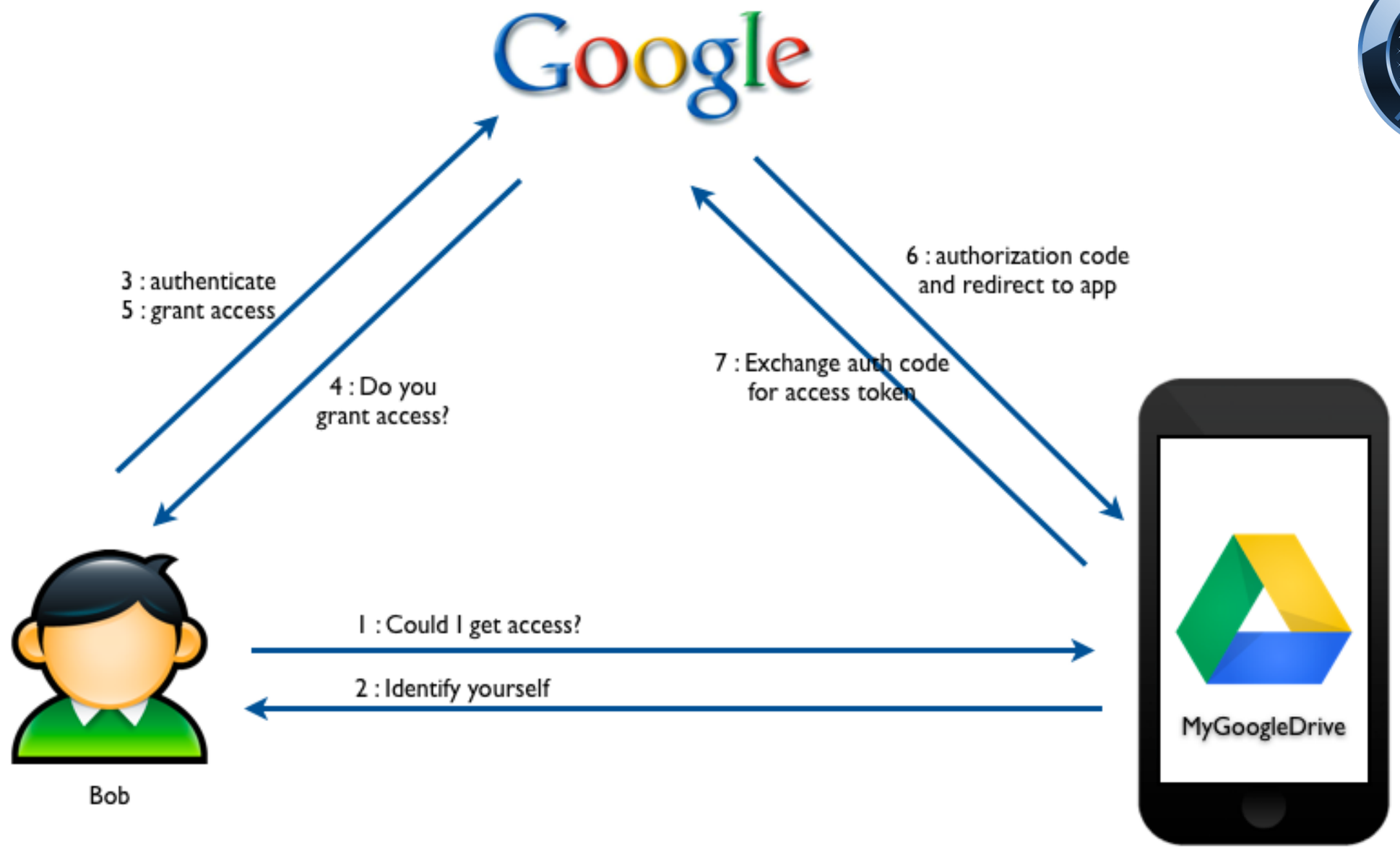
OAuth 1

http://hueniverse.com/oauth/
https://tools.ietf.org/html/rfc5849

- After the "OAuth dance" is concluded, client application has two sets of keys:
  - one that uses to identify itself as a valid application (api_key, api_secret)
  - one that uses to identify the user it wants to access (token, token_secret)
- You can revoke access at any time by letting the token provider that a given app is no long authorized (invalidating token and token_secret).

OAuth 2

https://tools.ietf.org/html/rfc6749
https://tools.ietf.org/html/rfc6750



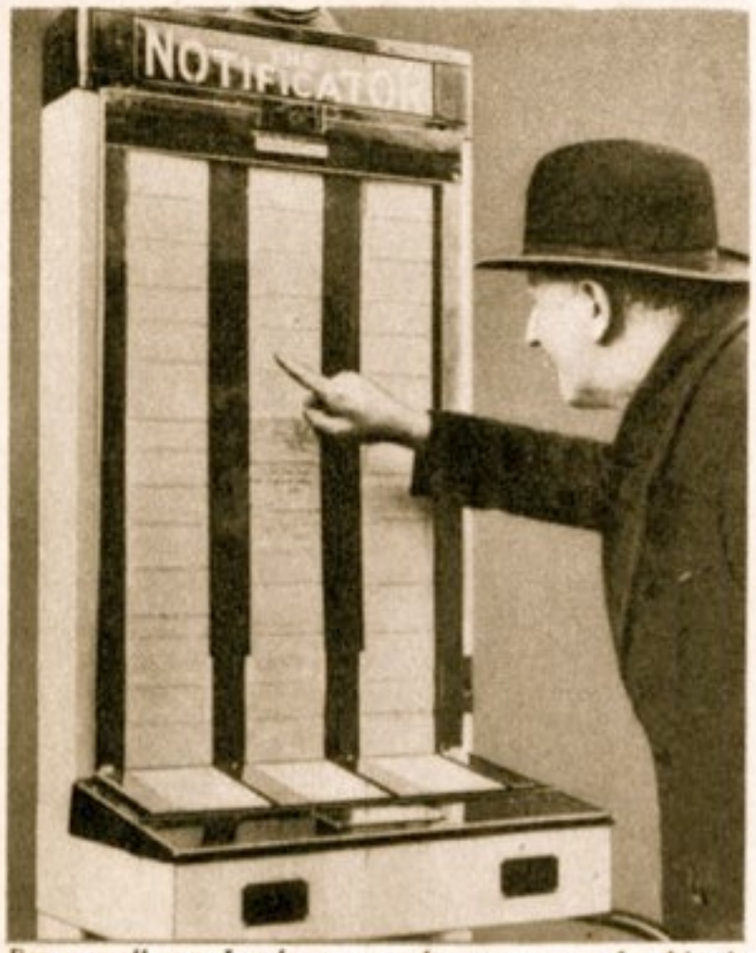
OAuth 2

https://tools.ietf.org/html/rfc6749
https://tools.ietf.org/html/rfc6750

- Latest version of OAuth protocol
    - Similar “dance” required
    - Allows for “bearer tokens” - access is given to anyone able to provide a valid token without any further restrictions or authentication
        - access tokens are provided along with the request for the resource through a secure connection
        - tokens can expire automatically
- We will use both OAuth and OAuth2 over the next few days

Twitter

Robot Messenger Displays Person-to-Person Notes In Public



For a small sum Londoners may leave messages for friends in public places. When written on "notificator," message moves up behind window, remaining in view for two hours.

TO AID persons who wish to make or cancel appointments or inform friends of their whereabouts, a robot message carrier has been introduced in London, England.

Known as the "notificator," the new machine is installed in streets, stores, railroad stations or other public places where individuals may leave messages for friends.

The user walks up on a small platform in front of the machine, writes a brief message on a continuous strip of paper and drops a coin in the slot. The inscription moves up behind a glass panel where it remains in public view for at least two hours so that the person for whom it is intended may have sufficient time to observe the note at the appointed place. The machine is similar in appearance to a candy-vending device.

Source: Modern Mechanix (Aug, 1935)

Anatomy of a Tweet

Anatomy of a Tweet


```
[u'contributors',
 u'truncated',
 u'text',
 u'in_reply_to_status_id',
 u'id',
 u'favorite_count',
 u'source',
 u'retweeted',
 u'coordinates',
 u'entities',
 u'in_reply_to_screen_name',
 u'in_reply_to_user_id',
 u'retweet_count',
 u'id_str',
 u'favorited',
 u'user',
 u'geo',
 u'in_reply_to_user_id_str',
 u'possibly_sensitive',
 u'lang',
 u'created_at',
 u'in_reply_to_status_id_str',
 u'place',
 u'metadata']
```

Anatomy of a Tweet

```
[u'contributors',
 u'truncated',
 u'text',
 u'in_reply_to_status_id',
 u'id',
 u'favorite_count',
 u'source',
 u'retweeted',
 u'coordinates',
 u'entities',
 u'in_reply_to_screen_name',
 u'in_reply_to_user_id',
 u'retweet_count',
 u'id_str',
 u'favorited',
 u'user',
 u'geo',
 u'in_reply_to_user_id_str',
 u'possibly_sensitive',
 u'lang',
 u'created_at',
 u'in_reply_to_status_id_str',
 u'place',
 u'metadata']
```

```
[u'follow_request_sent',
 u'profile_use_background_image',
 u'default_profile_image',
 u'id',
 u'profile_background_image_url_https',
 u'verified',
 u'profile_text_color',
 u'profile_image_url_https',
 u'profile_sidebar_fill_color',
 u'entities',
 u'followers_count',
 u'profile_sidebar_border_color',
 u'id_str',
 u'profile_background_color',
 u'listed_count',
 u'is_translation_enabled',
 u'utc_offset',
 u'statuses_count',
 u'description',
 u'friends_count',
 u'location',
 u'profile_link_color',
 u'profile_image_url',
 u'following',
 u'geo_enabled',
 u'profile_banner_url',
 u'profile_background_image_url',
 u'screen_name',
 u'lang',
```

```
 u'profile_background_tile',
 u'favourites_count',
 u'name',
 u'notifications',
 u'url',
 u'created_at',
 u'contributors_enabled',
 u'time_zone',
 u'protected',
 u'default_profile',
 u'is_translator']
```

Anatomy of a Tweet

```
[u'contributors',
 u'truncated',
 u'text',
 u'in_reply_to_status_id',
 u'id',
 u'favorite_count',
 u'source',
 u'retweeted',
 u'coordinates',
 u'entities',
 u'in_reply_to_screen_name',
 u'in_reply_to_user_id',
 u'retweet_count',
 u'id_str',
 u'favorited',
 u'user',
 u'geo',
 u'in_reply_to_user_id_str',
 u'possibly_sensitive',
 u'lang',
 u'created_at',
 u'in_reply_to_status_id_str',
 u'place',
 u'metadata']
```

```
u"I'm at Terminal Rodovi\xe1rio de Feira de Santana
 (Feira de Santana, BA) http://t.co/WirvdHwYMq"
```

```
u'<a href="http://foursquare.com" rel="nofollow">
  foursquare</a>'
```

```
[u'symbols',
 u'user_mentions',
 u'hashtags',
 u'urls']
```

```
[u'type',
 u'coordinates']
```

Anatomy of a Tweet

```
[u'contributors',
 u'truncated',
 u'text',
 u'in_reply_to_status_id',
 u'id',
 u'favorite_count',
 u'source',
 u'retweeted',
 u'coordinates',
 u'entities',
 u'in_reply_to_screen_name',
 u'in_reply_to_user_id',
 u'retweet_count',
 u'id_str',
 u'favorited',
 u'user',
 u'geo',
 u'in_reply_to_user_id_str',
 u'possibly_sensitive',
 u'lang',
 u'created_at',
 u'in_reply_to_status_id_str',
 u'place',
 u'metadata']
```

```
u"I'm at Terminal Rodovi\xe1rio de Feira de Santana
 (Feira de Santana, BA) http://t.co/WirvdHwYMq"
```

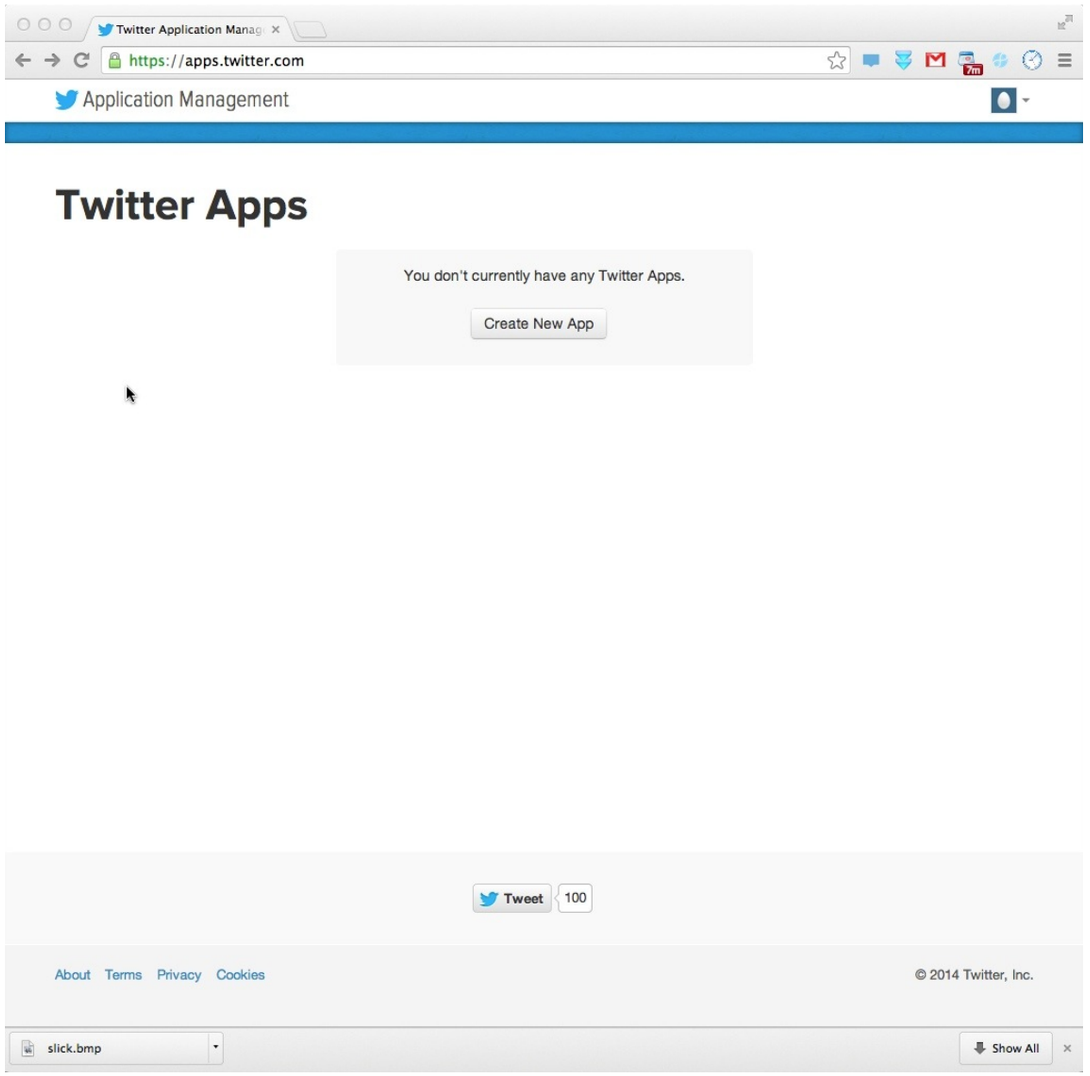
```
u'<a href="http://foursquare.com" rel="nofollow">
  foursquare</a>'
```

```
[u'symbols',
 u'user_mentions',
 u'hashtags',
 u'urls']
```

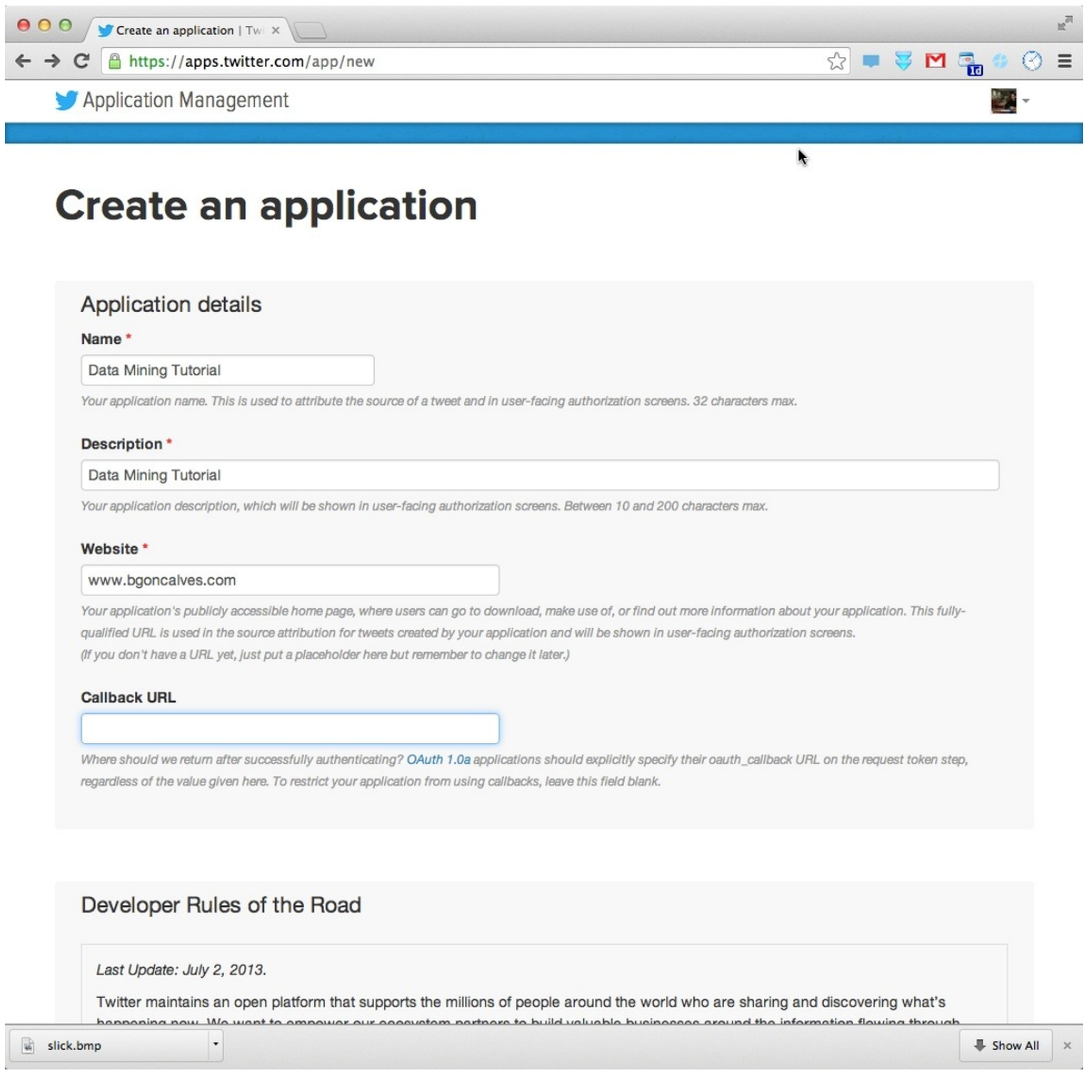
```
{u'display_url': u'4sq.com/1k5MeYF',
 u'expanded_url': u'http://4sq.com/1k5MeYF',
 u'indices': [70, 92],
 u'url': u'http://t.co/WirvdHwYMq'}
```

```
[u'type',
 u'coordinates']
```

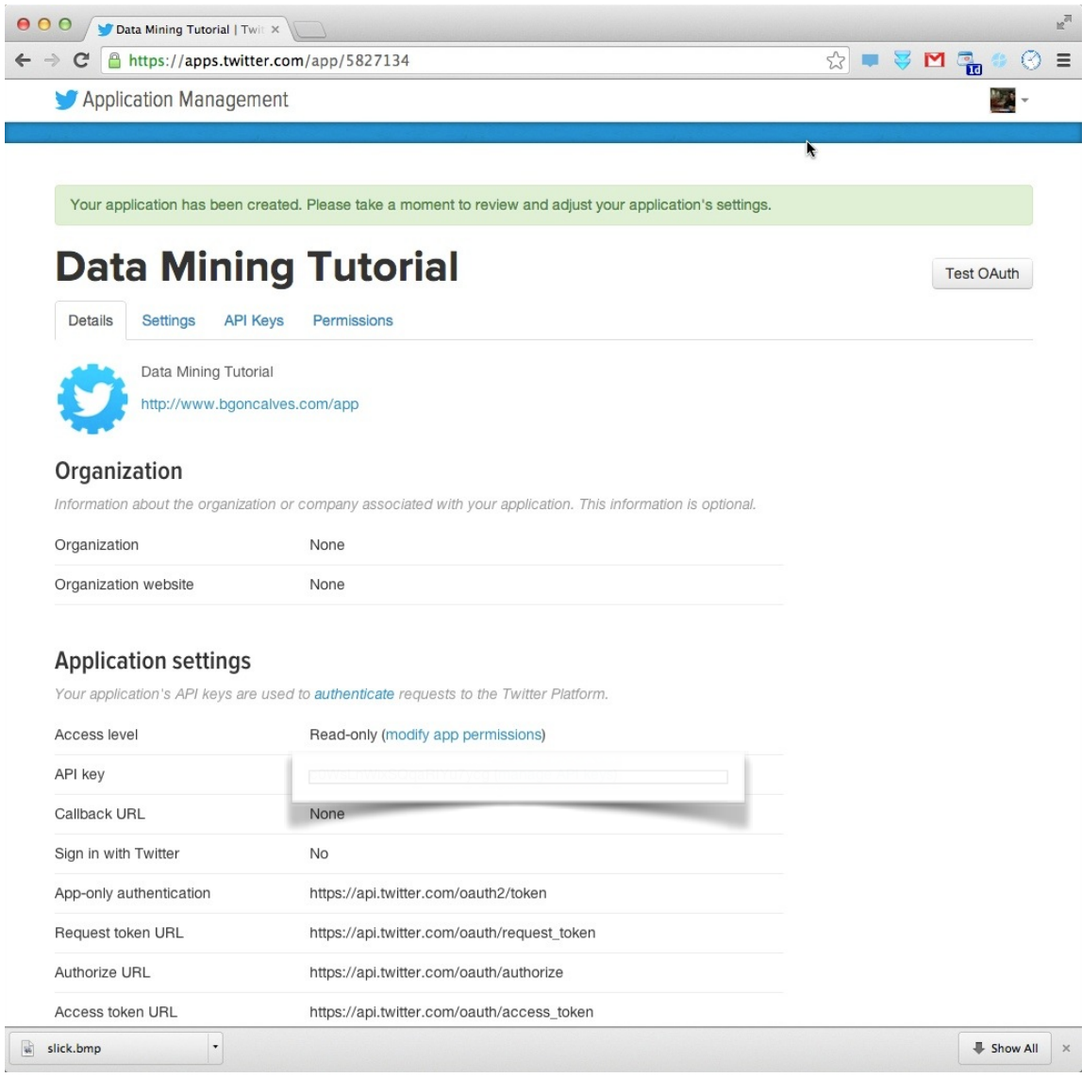
Registering an Application



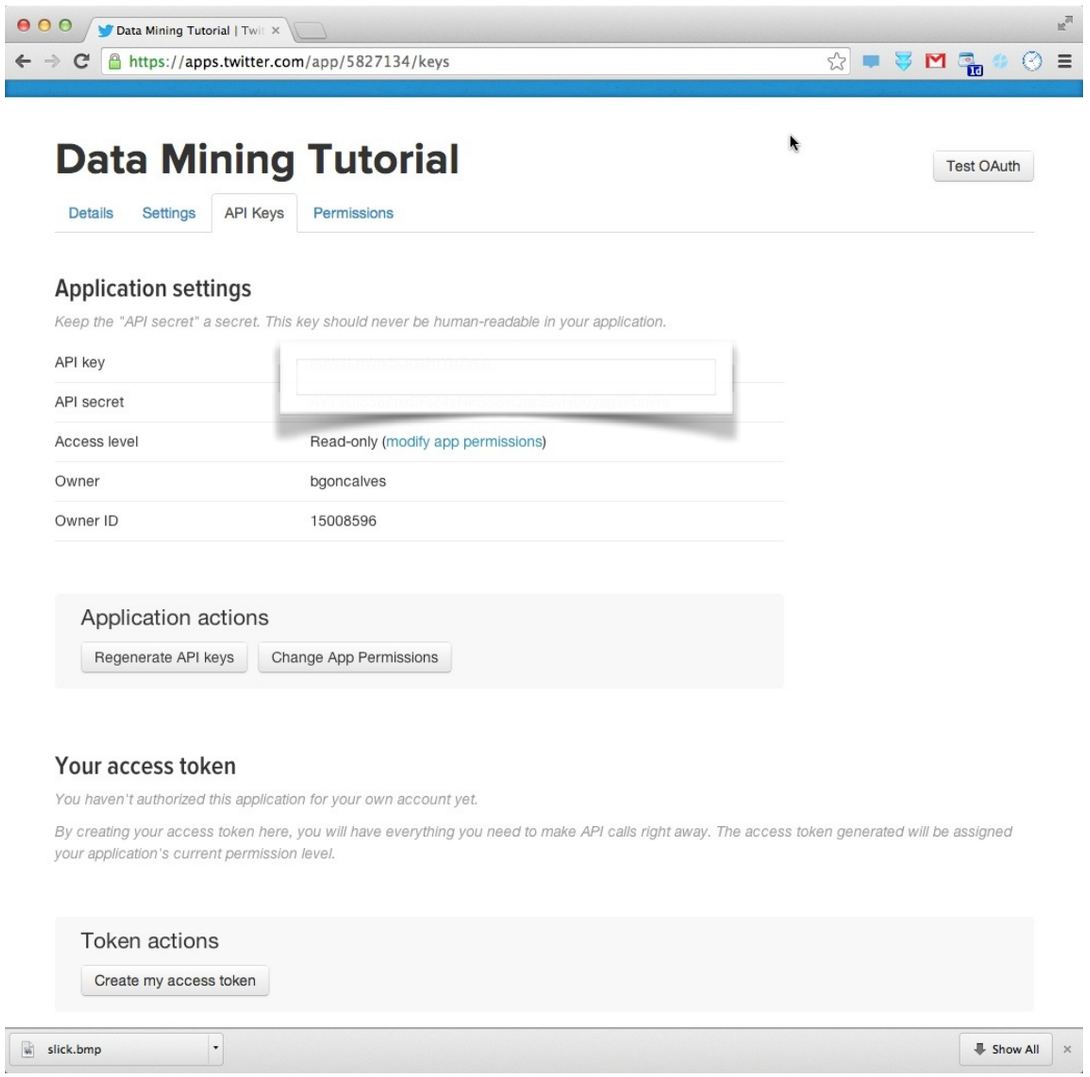
Registering an Application



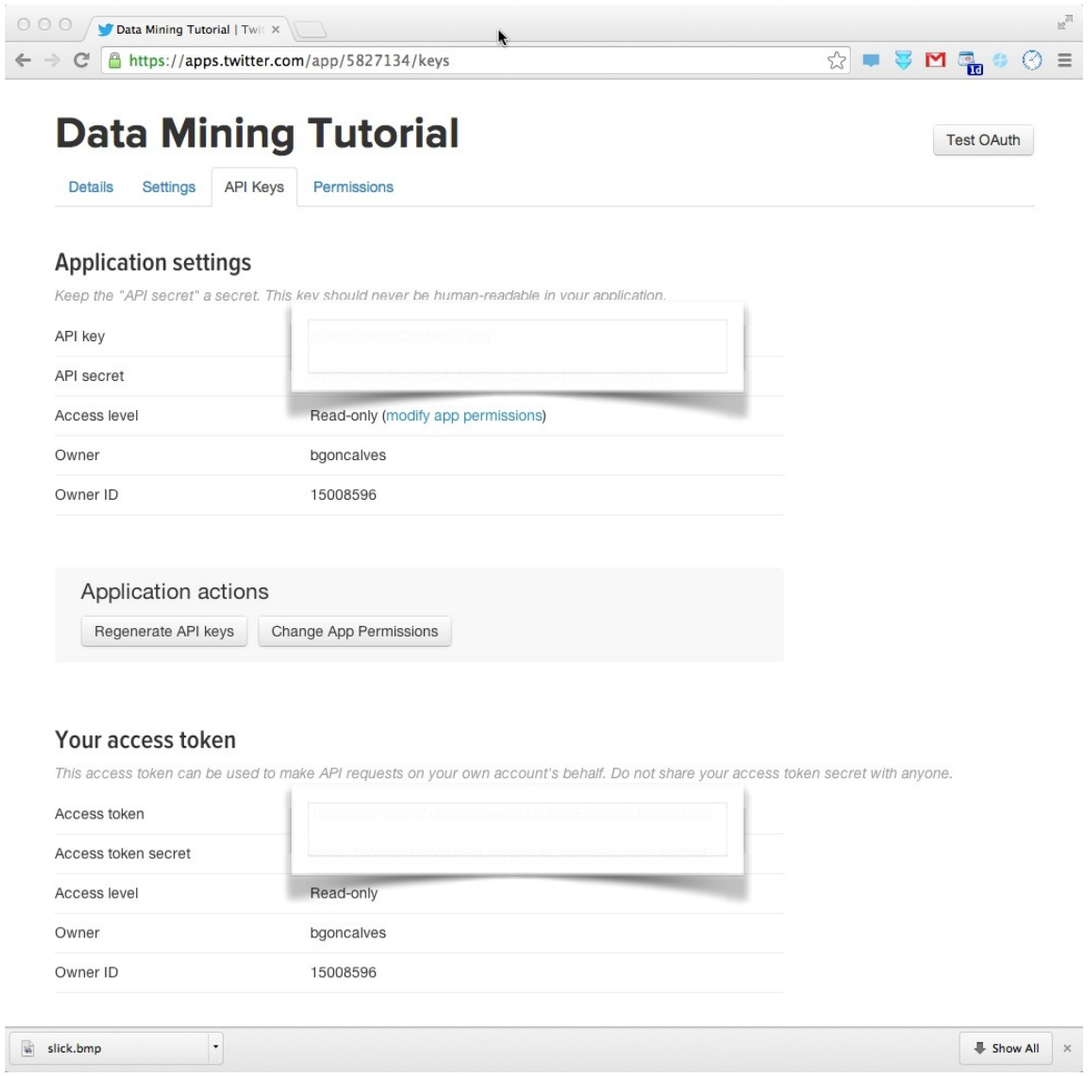
Registering an Application



Registering an Application



Registering an Application



API Basics

https://dev.twitter.com/docs

- The twitter module provides the oauth interface. We just need to provide the right credentials.

- Best to keep the credentials in a dict and parametrize our calls with the dict key. This way we can switch between different accounts easily.

- .Twitter(auth) takes an OAuth instance as argument and returns a Twitter object that we can use to interact with the API

- Twitter methods mimic API structure

- 4 basic types of objects:

    - Tweets

    - Users

    - Entities

Authenticating with the API

```
import tweepy
from twitter_accounts import accounts

app = accounts["social"]

auth = twitter.oauth.OAuth(app["token"],
                           app["token_secret"],
                           app["api_key"],
                           app["api_secret"])

twitter_api = twitter.Twitter(auth=auth)
```

- In the remainder of this course, the accounts dict will live inside the twitter_accounts.py file
- 4 basic types of objects:
  - Tweets
  - Users
  - Entities
  - Places

twitter_authentication.py

Searching for Tweets

https://dev.twitter.com/docs/api/1.1/get/search/tweets

- .search.tweets(query, count)
  - query is the content to search for
  - count is the maximum number of results to return
- returns dict with a list of "statuses" and "search_metadata"


```
{u'completed_in': 0.027,
u'count': 15,
u'max_id': 438088492577345536,
u'max_id_str': u'438088492577345536',
u'next_results': u'?max_id=438088485145034752&q=soccer&include_entities=1',
u'query': u'soccer',
u'refresh_url': u'?since_id=438088492577345536&q=soccer&include_entities=1',
u'since_id': 0,
u'since_id_str': u'0'}
```


- search_results["search_metadata"]["next_results"] can be used to get the next page of results

Searching for Tweets

https://dev.twitter.com/docs/api/1.1/get/search/tweets

```
query = "instagram"
count = 200

search_results = twitter_api.search.tweets(q=query, count=count)

statuses = search_results["statuses"]
tweet_count = 0

while True:
    try:
        next_results = search_results["search_metadata"]["next_results"]

        args = dict(parse.parse_qsl(next_results[1:]))

        search_results = twitter_api.search.tweets(**args)
        statuses = search_results["statuses"]

        print(search_results["search_metadata"]["max_id"])

        for tweet in statuses:
            tweet_count += 1

            if tweet_count % 10000 == 0:
                print(tweet_count,file=sys.stderr)

            print (tweet["text"])
    except:
        break
```


twitter_search.py

Streaming data

https://dev.twitter.com/docs/api/1.1/post/statuses/filter

- The Streaming api provides realtime data, subject to filters
- Use TwitterStream instead of Twitter object (.TwitterStream(auth=twitter_api.auth))
- .status.filter(track=q) will return tweets that match the query q in real time
  - Returns generator that you can iterate over

Streaming data

https://dev.twitter.com/docs/api/1.1/post/statuses/filter

```
import twitter
from twitter_accounts import accounts

app = accounts["social"]

auth = twitter.oauth.OAuth(app["token"],
                           app["token_secret"],
                           app["api_key"],
                           app["api_secret"])

stream_api = twitter.TwitterStream(auth=auth)

query = "bieber"

stream_results = stream_api.statuses.filter(track=query)

for tweet in stream_results:
    print(tweet["text"])
```


twitter_stream.py

User profiles

https://dev.twitter.com/docs/api/1.1/get/users/lookup

- .users.lookup() returns user profile information for a list of user_ids or screen_names

- list should be comma separated and provided as a string

```
import twitter
from twitter_accounts import accounts

app = accounts["social"]

auth = twitter.oauth.OAuth(app["token"],
                           app["token_secret"],
                           app["api_key"],
                           app["api_secret"])

twitter_api = twitter.Twitter(auth=auth)

screen_names = ",".join(["diunito", "giaruffo"])

search_results = twitter_api.users.lookup(screen_name=screen_names)

for user in search_results:
  print(user["screen_name"], "has", user["followers_count"], "followers")
```


twitter_users.py

Social Connections

https://dev.twitter.com/docs/api/1.1/get/friends/ids
https://dev.twitter.com/docs/api/1.1/get/followers/ids

- .friends.ids() and .followers.ids() returns a list of up to 5000 of a users friends or followers for a given screen_name or user_id

- result is a dict containing multiple fields:

```
[u'next_cursor_str',
 u'previous_cursor',
 u'ids',
 u'next_cursor',
 u'previous_cursor_str']
```

- ids are contained in results[“ids”].

- results[“next_cursor”] allows us to obtain the next page of results.

- .friends.ids(screen_name=screen_name, cursor=results[“next_cursor”]) will return the next page of results

- cursor=0 means no more results

Social Connections

https://dev.twitter.com/docs/api/1.1/get/friends/ids
https://dev.twitter.com/docs/api/1.1/get/followers/ids

```
import twitter
from twitter_accounts import accounts

app = accounts["social"]

auth = twitter.oauth.OAuth(app["token"],
                           app["token_secret"],
                           app["api_key"],
                           app["api_secret"])

twitter_api = twitter.Twitter(auth=auth)

screen_name = "stephen_wolfram"

cursor = -1
followers = []

while cursor != 0:
    result = twitter_api.followers.ids(screen_name=screen_name,
cursor=cursor)

    followers += result["ids"]
    cursor = result["next_cursor"]

print("Found", len(followers), "Followers")
```


twitter_followers.py

User Timeline

https://dev.twitter.com/docs/api/1.1/get/statuses/user_timeline

- .statuses.user_timeline() returns a set of tweets posted by a single user
- Important options:
  - include_rts='true' to Include retweets by this user
  - count=200 number of tweets to return in each call
  - trim_user='true' to not include the user information (save bandwidth and processing time)
  - max_id=1234 to include only tweets with an id lower than 1234
- Returns at most 200 tweets in each call. Can get all of a users tweets (up to 3200) with multiple calls using max_id

User Timeline

https://dev.twitter.com/docs/api/1.1/get/statuses/user_timeline

```
import twitter
from twitter_accounts import accounts

app = accounts["social"]

auth = twitter.oauth.OAuth(app["token"],
                           app["token_secret"],
                           app["api_key"],
                           app["api_secret"])

twitter_api = twitter.Twitter(auth=auth)
screen_name = "BarackObama"

args = { "count" : 200,
         "trim_user": "true",
         "include_rts": "true"
        }

tweets = twitter_api.statuses.user_timeline(screen_name = screen_name, **args)
tweets_new = tweets

while len(tweets_new) > 0:
    max_id = tweets[-1]["id"] - 1
    tweets_new = twitter_api.statuses.user_timeline(screen_name = screen_name, max_id=max_id, **args)
    tweets += tweets_new

print("Found", len(tweets), "tweets")
```


twitter_timeline.py

Social Interactions

```
import twitter
from twitter_accounts import accounts

app = accounts["social"]

auth = twitter.oauth.OAuth(app["token"],
                           app["token_secret"],
                           app["api_key"],
                           app["api_secret"])

twitter_api = twitter.Twitter(auth=auth)
screen_name = "bgoncalves"
args = { "count" : 200, "trim_user": "true", "include_rts": "true"}

tweets = twitter_api.statuses.user_timeline(screen_name=screen_name, **args)
tweets_new = tweets

while len(tweets_new) > 0:
    max_id = tweets[-1]["id"] - 1
    tweets_new = twitter_api.statuses.user_timeline(screen_name=screen_name, max_id=max_id, **args)
    tweets += tweets_new

user = tweets[0]["user"]["id"]

for tweet in tweets:
    if "retweeted_status" in tweet:
        print(user, "->", tweet["retweeted_status"]["user"]["id"])
    elif tweet["in_reply_to_user_id"]:
        print(tweet["in_reply_to_user_id"], "->", user)
```


twitter_interactions.py

Streaming Geocoded data

https://dev.twitter.com/streaming/overview/request-parameters#locations

- The Streaming api provides realtime data, subject to filters

- Use TwitterStream instead of Twitter object (.TwitterStream(auth=twitter_api.auth))

- .status.filter(track=q) will return tweets that match the query q in real time

- Returns generator that you can iterate over

- .status.filter(locations=bb) will return tweets that occur within the bounding box bb in real time

- bb is a comma separated pair of lon/lat coordinates.
  - -180,-90,180,90 - World
  - -74,40,-73,41 - NYC

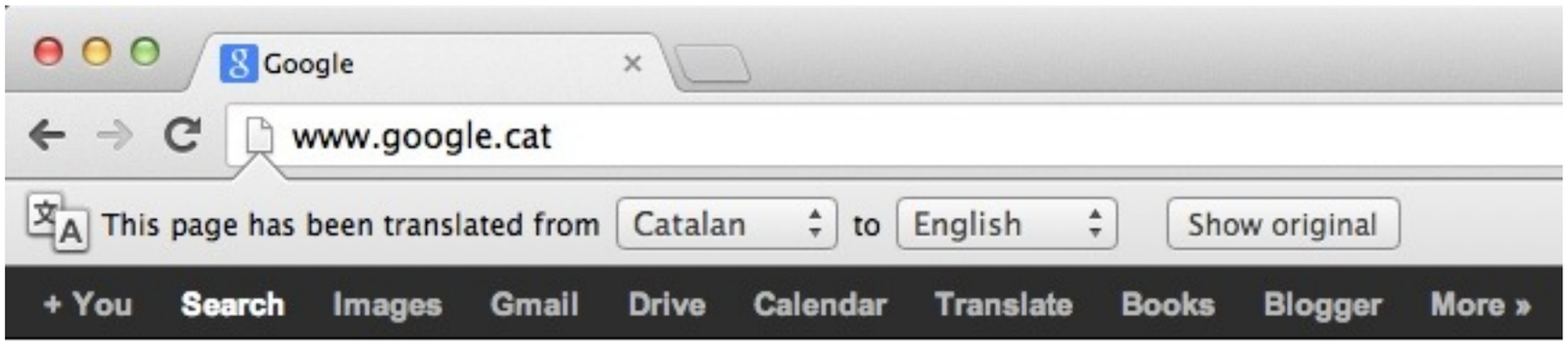
Streaming Geocoded data

https://dev.twitter.com/streaming/overview/request-parameters#locations

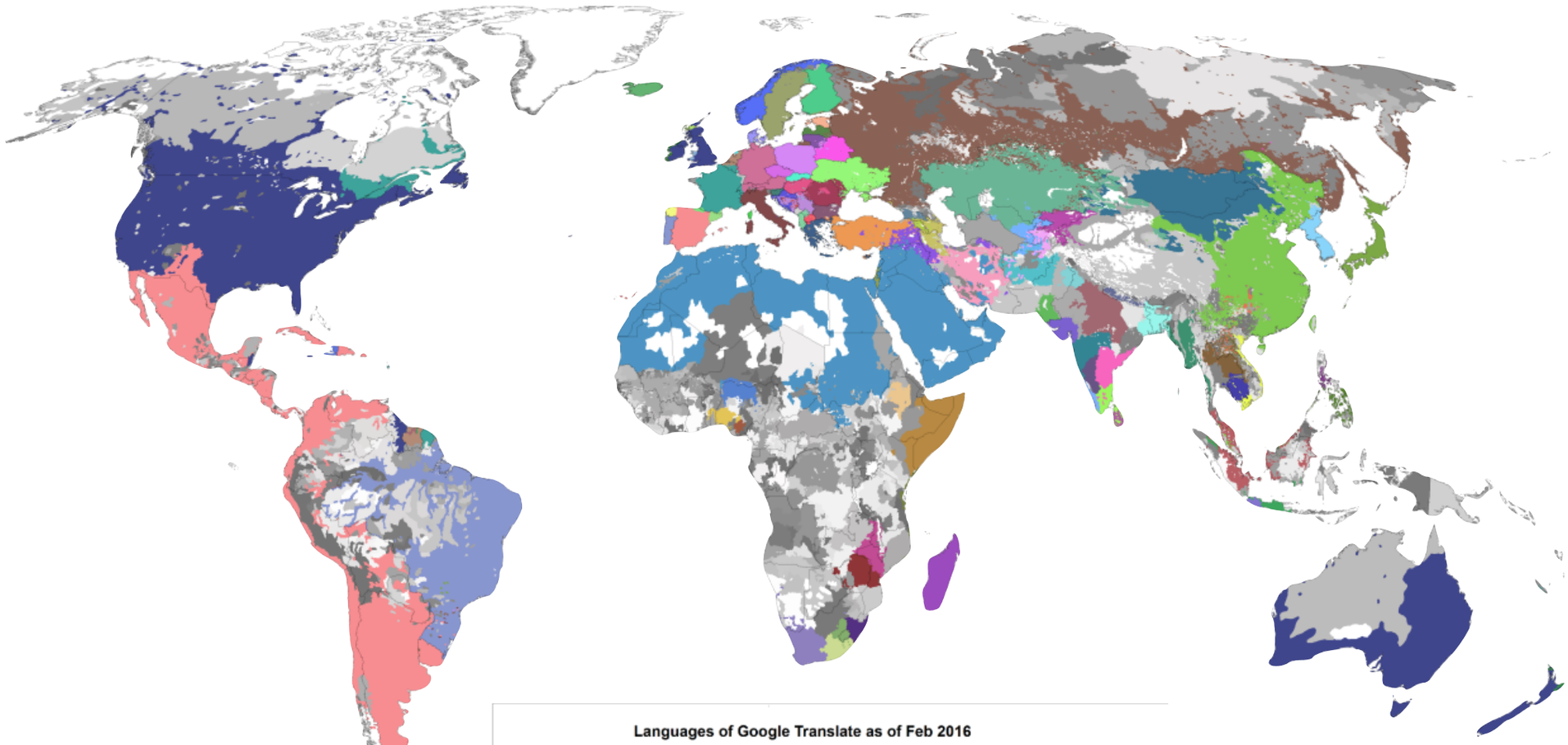
```
import twitter
from twitter_accounts import accounts
import sys
import gzip

app = accounts["social"]

auth = twitter.oauth.OAuth(app["token"],
                           app["token_secret"],
                           app["api_key"],
                           app["api_secret"])

stream_api = twitter.TwitterStream(auth=auth)

query = "-74,40,-73,41"  # NYC
stream_results = stream_api.statuses.filter(locations=query)
tweet_count = 0

fp = gzip.open("NYC.json.gz", "a")

for tweet in stream_results:
    try:
        tweet_count += 1
        print (tweet_count, tweet["id"])
        print(tweet, file=fp)
    except:
        pass

    if tweet_count % 10000 == 0:
        print(tweet_count, file=sys.stderr)
        break
```

twitter_location.py

Chromium Compact Language Detector



Chromium Compact Language Detector



cld

https://github.com/mikemccand/chromium-compact-language-detector

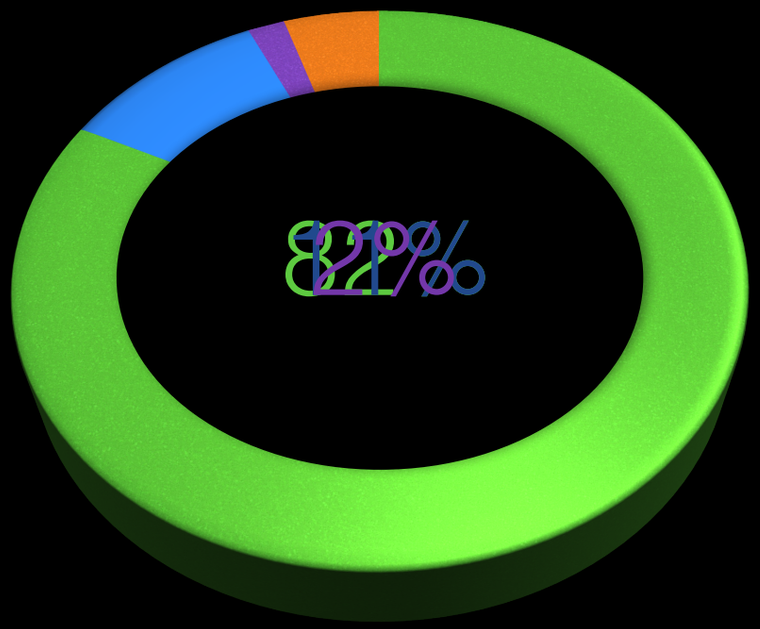
- C package with C++ and Python bindings
- Particularly messy to install, but very fast and easy to use
- cld.detect(text) returns a list of possible languages and their likelihood
- Very conservative. If it thinks that it can't give a reliable answer, it will return "Unknown"
- Version 2 is better suited to longer pieces of text and it allows for the possibility of different sections being in different languages.

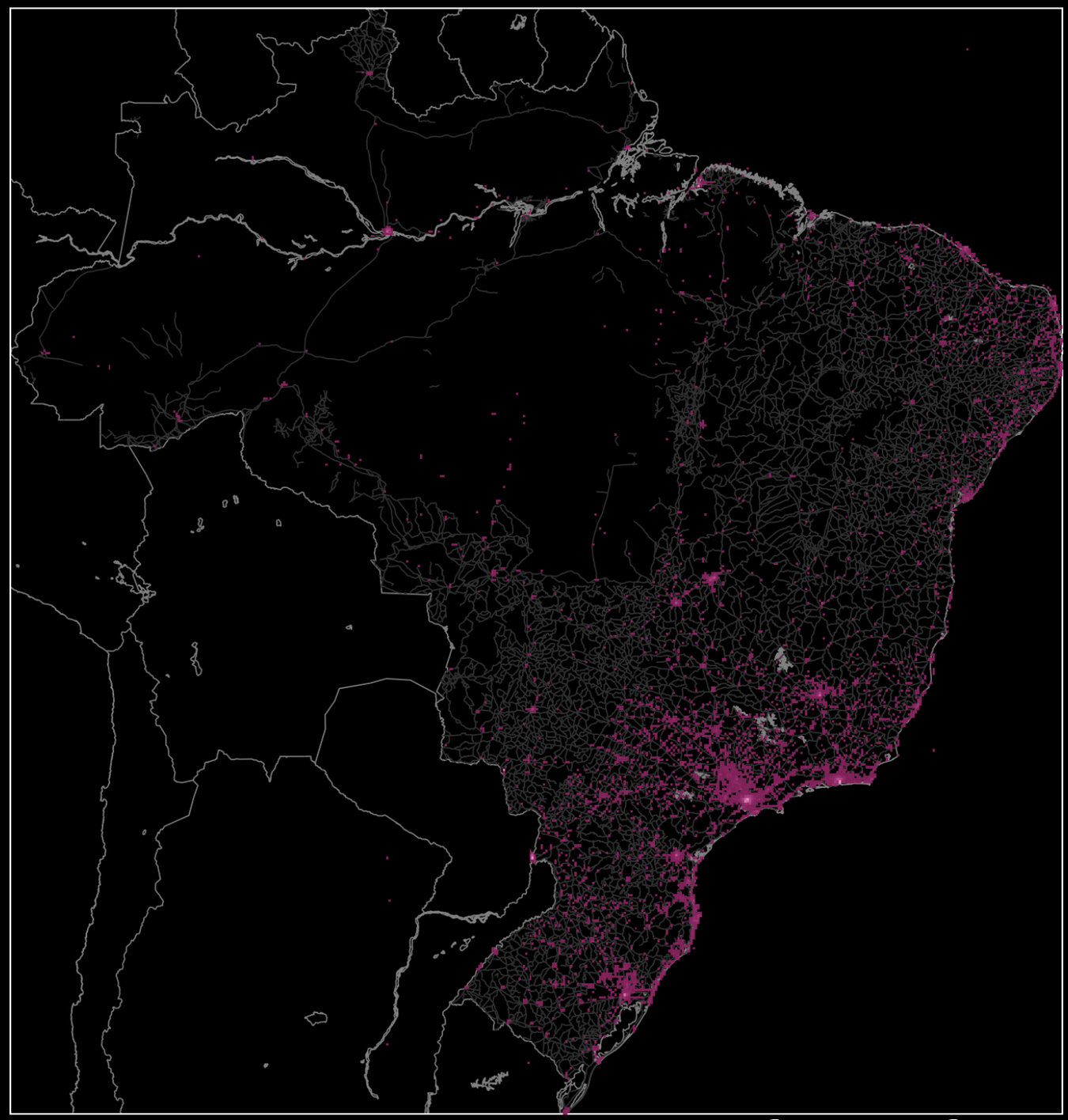
```
import cld

text_it = "Wales lancia la Wikipedia delle news. Contro il fake in campo anche Google"
text_en = "Cassini Spacecraft Re-Establishes Contact After 'Dive' Between Saturn And Its Rings"

lang_it = cld.detect(text_it)
lang_en = cld.detect(text_en)

print(text_it, "is in", lang_it)
print(text_en, "is in", lang_en)
```


language_cld.py

Signal By Language

- Portuguese
- English
- Spanish
- Other





langid

https://github.com/saffsd/langid.py

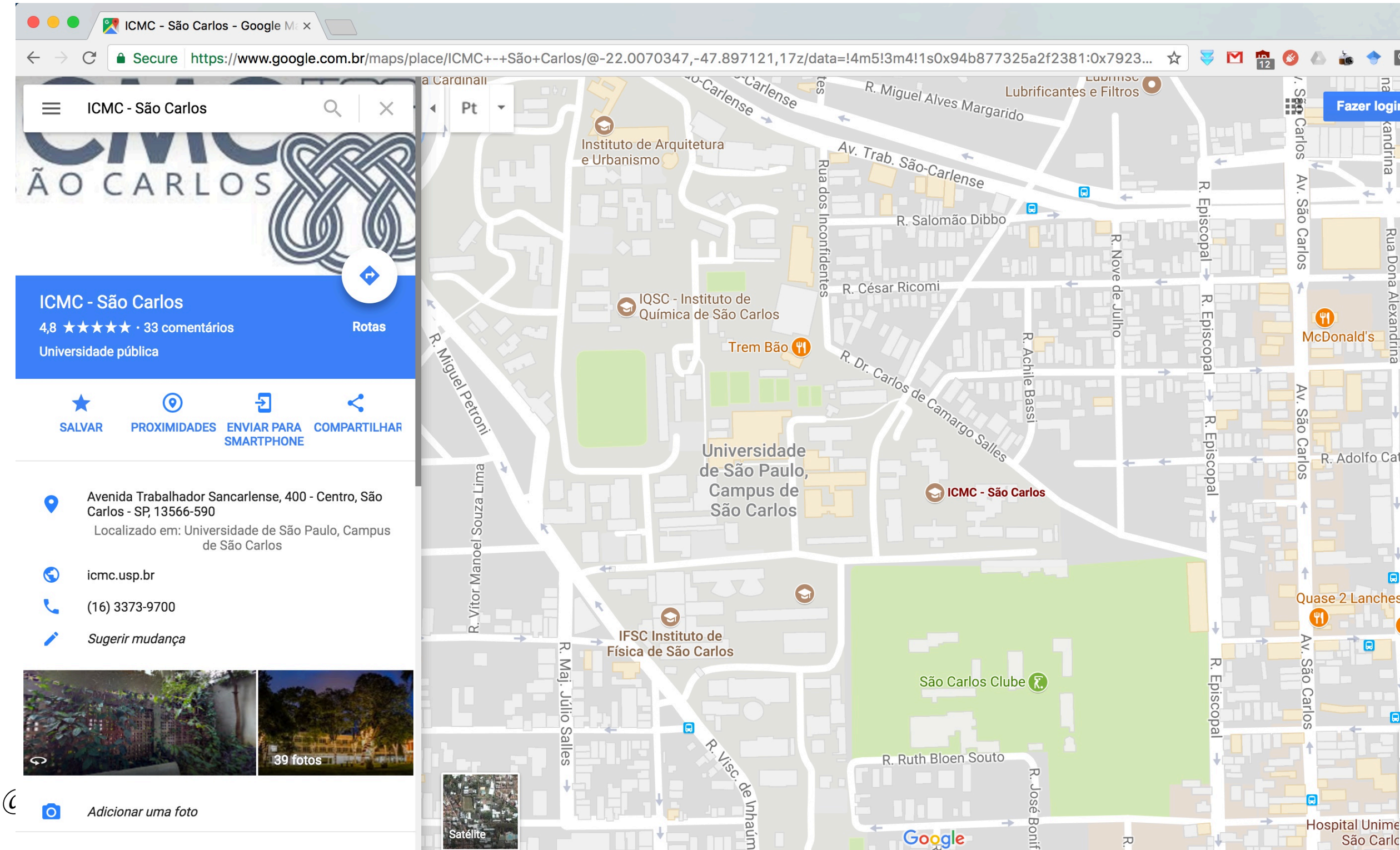
- Entirely written in Python
- Supports 97 languages
- Has the option of retrieving the likelihood for every known language.
- .classify(text) - Returns the most likely language and it's likelihood
- .rank(text) - Returns the results for all known languages.

```
import langid

text_it = "Wales lancia la Wikipedia delle news. Contro il fake in campo anche Google"
text_en = "Cassini Spacecraft Re-Establishes Contact After 'Dive' Between Saturn And Its Rings"

lang_it = langid.classify(text_it)
lang_en = langid.classify(text_en)

print(text_it, "is in", lang_it)
print(text_en, "is in", lang_en)
```

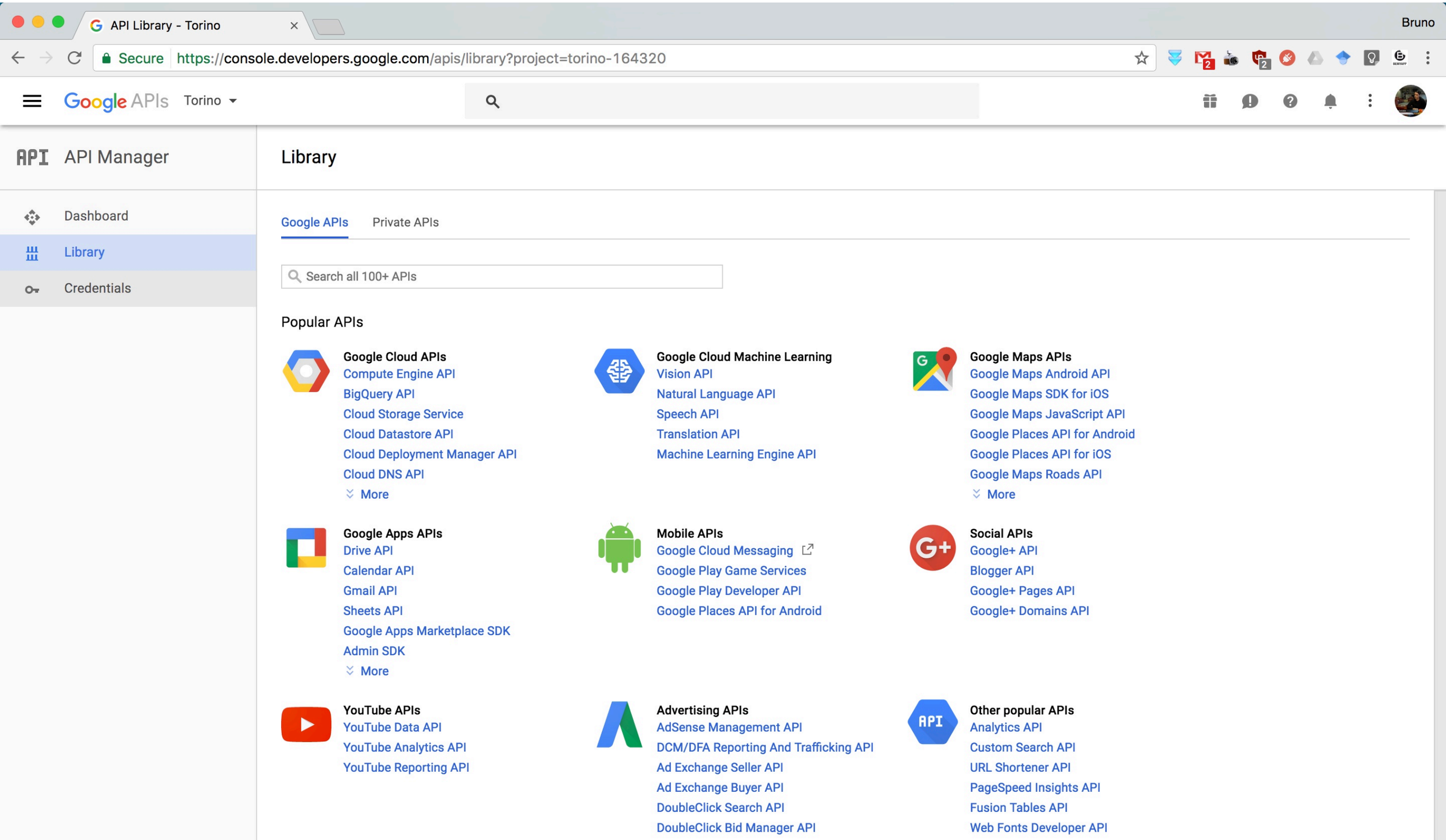

language_langid.py

Google Maps APIs

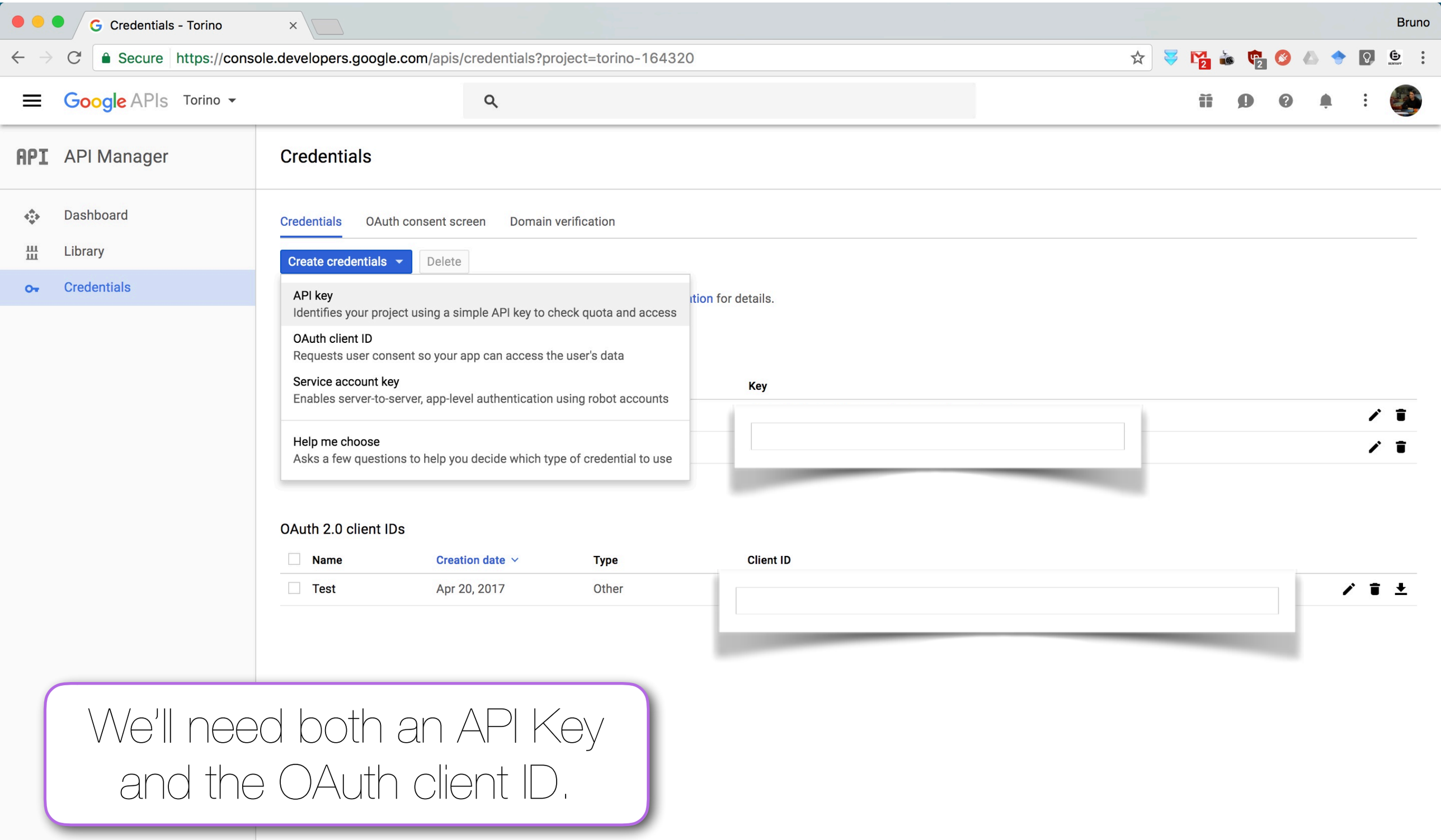
Google Maps


ICMC - São Carlos - Google Ma
Secure
https://www.google.com.br/maps/place/ICMC+-+São+Carlos/@-22.0070347,-47.897121,17z/data=!4m5!3m4!1s0x94b877325a2f2381:0x7923...
ICMC - São Carlos
Pt
ICMC - São Carlos
4,8 ★★★★★ · 33 comentários
Universidade pública
Rotas
SALVAR
PROXIMIDADES
ENVIAR PARA SMARTPHONE
COMPARTILHAR
Avenida Trabalhador Sancarlense, 400 - Centro, São Carlos - SP, 13566-590
Localizado em: Universidade de São Paulo, Campus de São Carlos
icmc.usp.br
(16) 3373-9700
Sugerir mudança
39 fotos
Adicionar uma foto
Satélite
Instituto de Arquitetura e Urbanismo
IQSC - Instituto de Química de São Carlos
Trem Bão
Universidade de São Paulo, Campus de São Carlos
IFSC Instituto de Física de São Carlos
São Carlos Clube
R. Miguel Alves Margarido
Lubrificantes e Filtros
Av. Trab. São-Carlense
R. Salomão Dibbo
R. César Ricomi
R. Dr. Carlos de Camargo Salles
R. Achile Bassi
R. Episcopal
Av. São Carlos
McDonald's
R. Adolfo Cat
Quase 2 Lanches
R. Miguel Petroni
R. Vitor Manoel Souza Lima
R. Maj. Júlio Salles
R. Visc. de Inhaúm
R. Ruth Bloen Souto
R. José Bonif
Hospital Unime São Carlo
Google

Google APIs

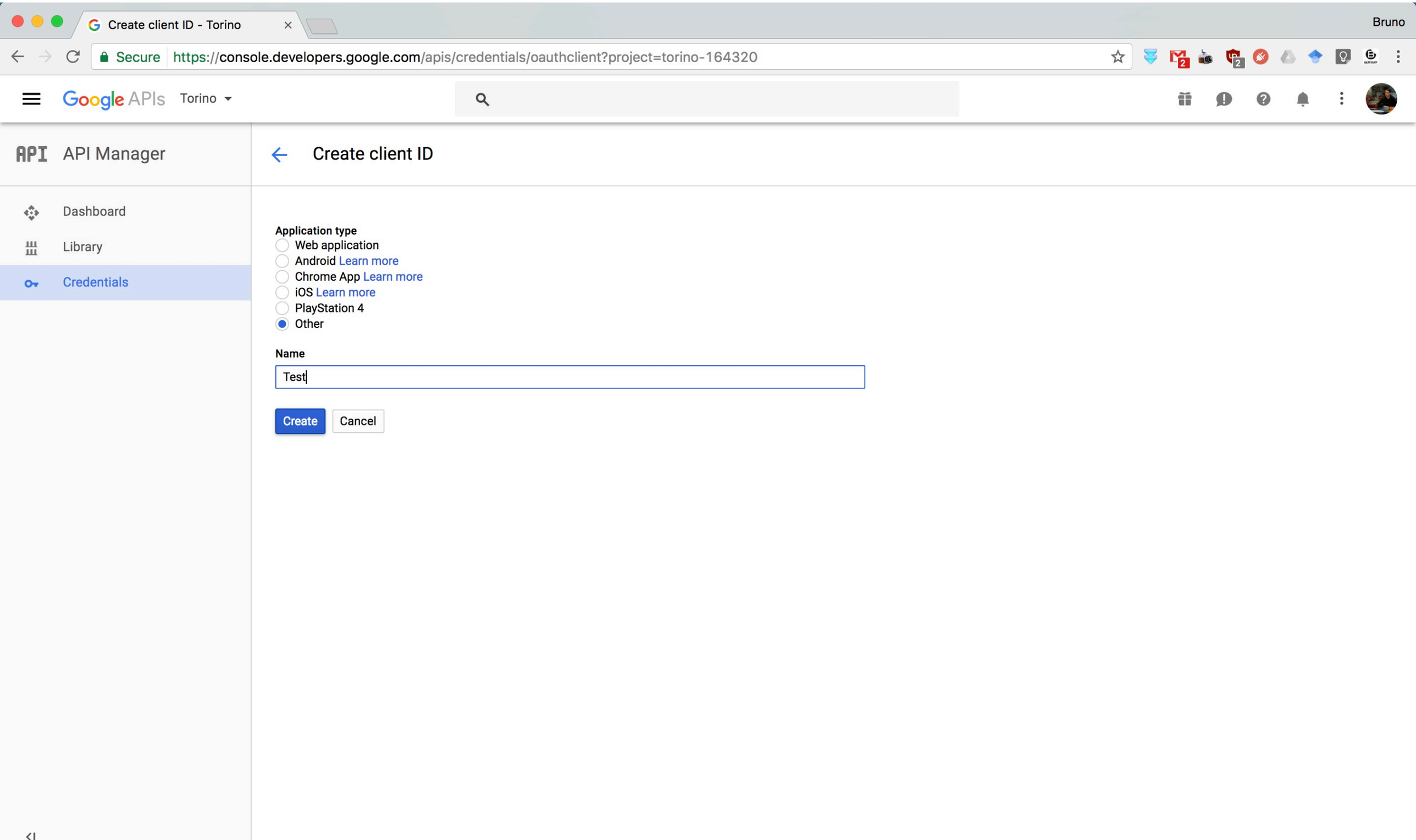

API Library - Torino
Bruno
Secure https://console.developers.google.com/apis/library?project=torino-164320
Google APIs Torino
API Manager
Library
Dashboard
Library
Credentials
Google APIs
Private APIs
Search all 100+ APIs
Popular APIs
Google Cloud APIs
Compute Engine API
BigQuery API
Cloud Storage Service
Cloud Datastore API
Cloud Deployment Manager API
Cloud DNS API
More
Google Cloud Machine Learning
Vision API
Natural Language API
Speech API
Translation API
Machine Learning Engine API
Google Maps APIs
Google Maps Android API
Google Maps SDK for iOS
Google Maps JavaScript API
Google Places API for Android
Google Places API for iOS
Google Maps Roads API
More
Google Apps APIs
Drive API
Calendar API
Gmail API
Sheets API
Google Apps Marketplace SDK
Admin SDK
More
Mobile APIs
Google Cloud Messaging
Google Play Game Services
Google Play Developer API
Google Places API for Android
Social APIs
Google+ API
Blogger API
Google+ Pages API
Google+ Domains API
YouTube APIs
YouTube Data API
YouTube Analytics API
YouTube Reporting API
Advertising APIs
AdSense Management API
DCM/DFA Reporting And Trafficking API
Ad Exchange Seller API
Ad Exchange Buyer API
DoubleClick Search API
DoubleClick Bid Manager API
Other popular APIs
Analytics API
Custom Search API
URL Shortener API
PageSpeed Insights API
Fusion Tables API
Web Fonts Developer API

Google Credentials

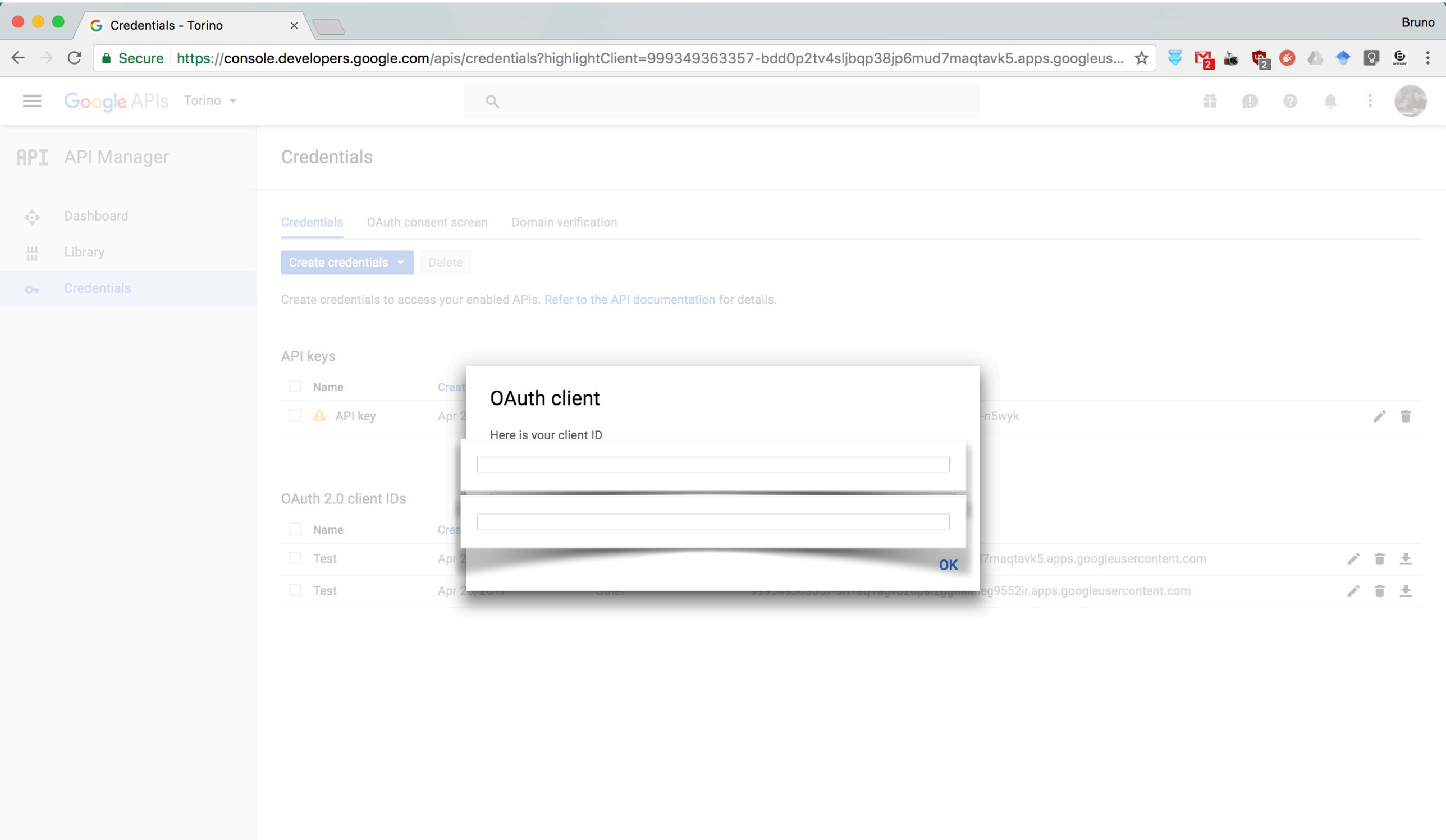


We'll need both an API Key and the OAuth client ID.

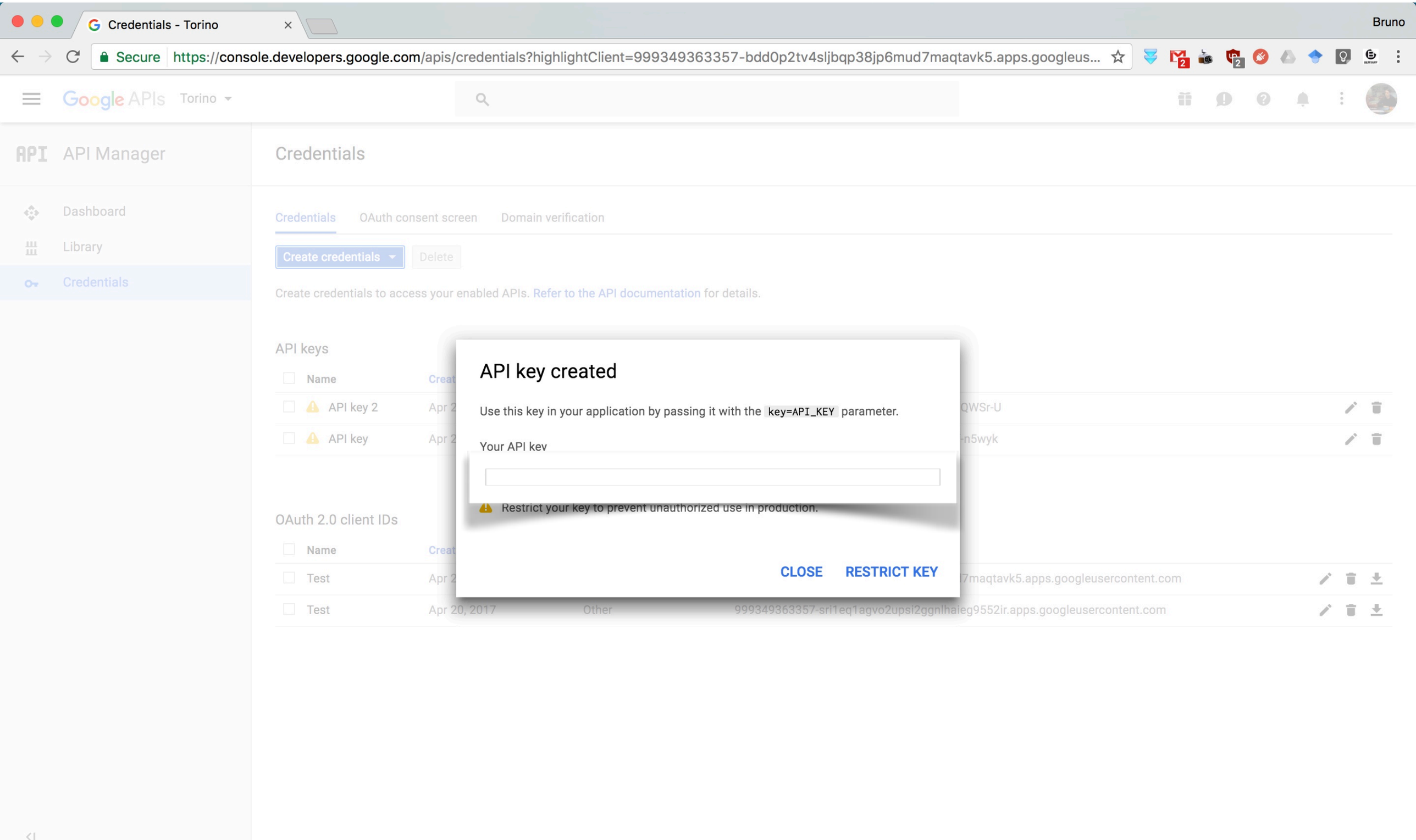
Google Credentials



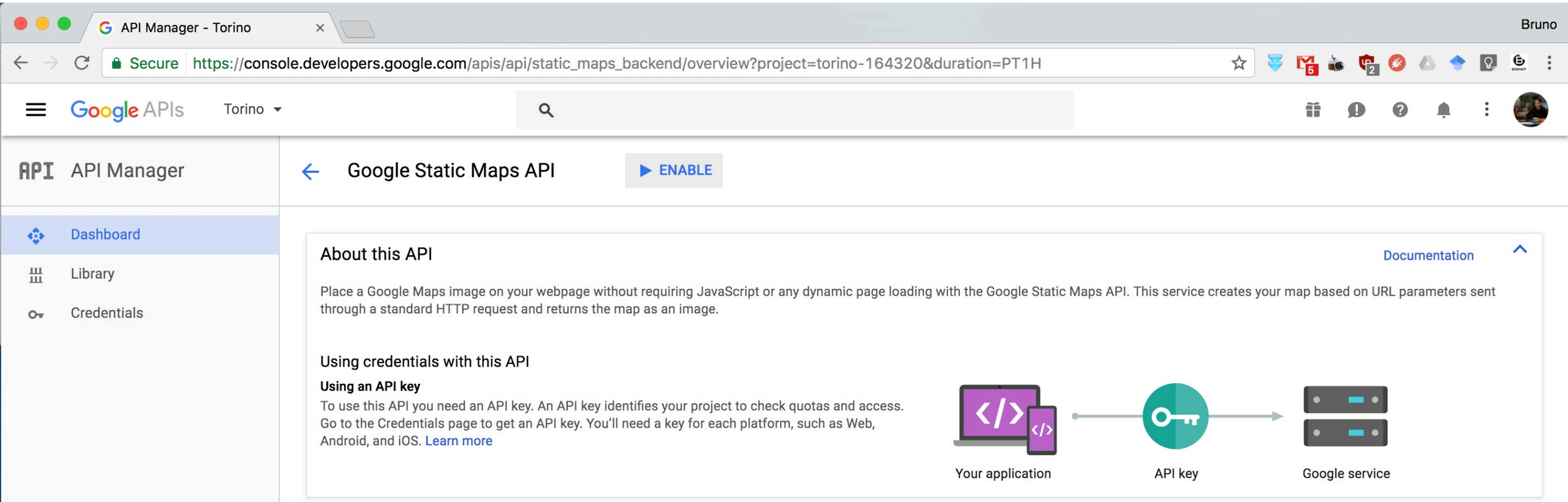
Google Credentials



Google Credentials

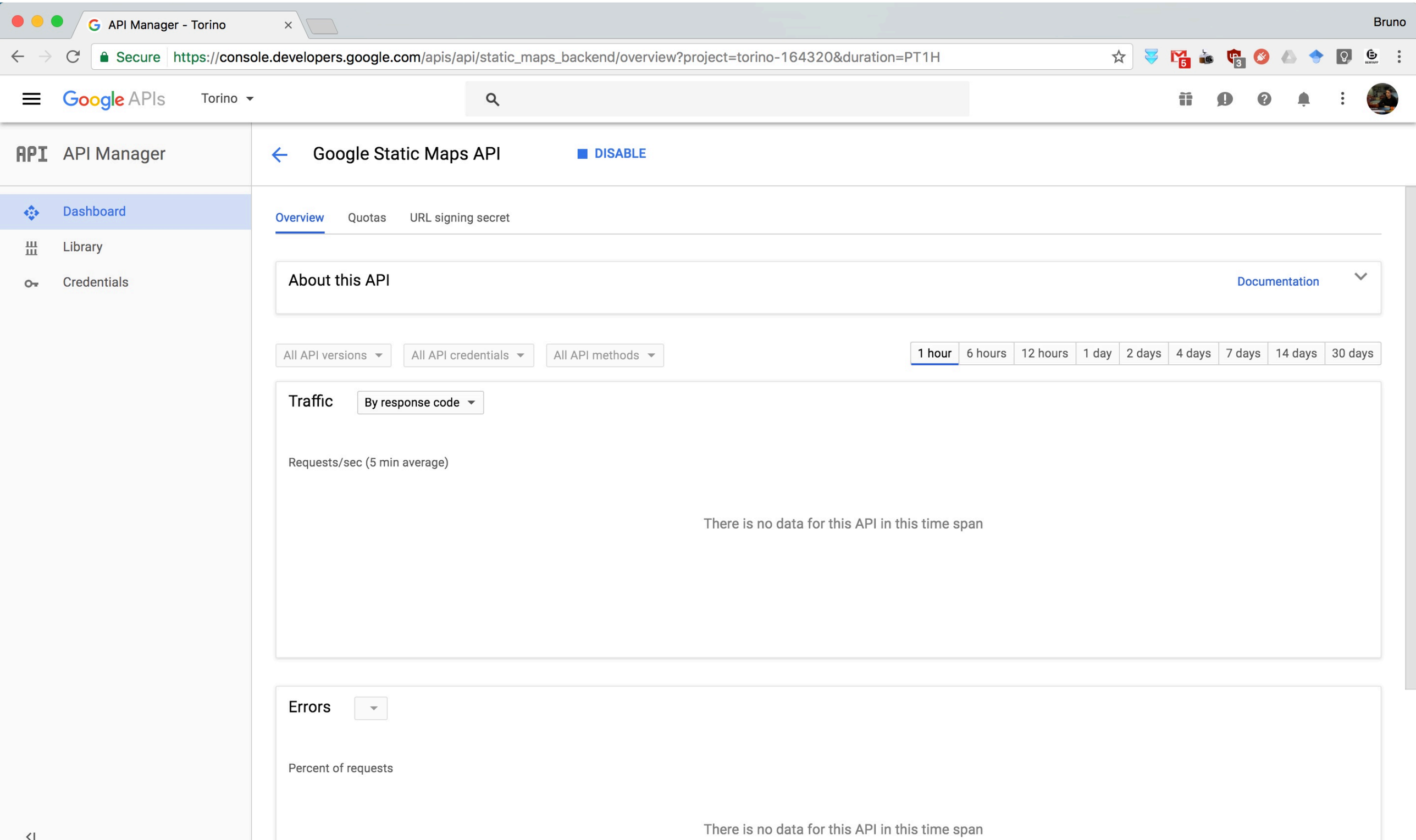


API Authorization



Each API has to be "Enabled" for this specific set of credentials

API Authorization



Google Maps APIs

https://github.com/googlemaps/google-maps-services-python

- As before, we store our credentials in a dictionary kept in an external file

```
accounts = {
    "torino": {
        "api_key": "API_KEY",
        "client_id": "CLIENT_ID",
        "client_secret": "CLIENT_SECRET"
    }
}
```

- Google provides dozens of different APIs for various purposes.
- The googlemaps Python library provides easy access to some of the GIS related ones:
  - Directions API
  - Distance Matrix API
  - Elevation API
  - Geocoding API
  - Geolocation API
  - Time Zone API
  - Roads API
  - Places API

google_accounts_STUB.py

Google Maps APIs

https://github.com/googlemaps/google-maps-services-python

- As before, we store our credentials in a dictionary kept in an external file

```
accounts = {
    "torino": {
        "api_key": "API_KEY",
        "client_id": "CLIENT_ID",
        "client_secret": "CLIENT_SECRET"
    }
}
```

- Google provides dozens of different APIs for various purposes.

- The googlemaps Python library provides easy access to some of the GIS related ones:
  - Directions API -
  - Distance Matrix API
  - Elevation API
  - Geocoding API
  - Geolocation API
  - Time Zone API
  - Roads API
  - Places API

google_accounts_STUB.py

Google Geocoding API

https://developers.google.com/maps/documentation/geocoding/start

- Allows us to obtain the latitude and longitude of a specific location.

```
import googlemaps
from google_accounts import accounts

app = accounts["torino"]
gmaps = googlemaps.Client(key=app["api_key"])

geocode_result = gmaps.geocode('JFK Airport')

print(geocode_result[0]["geography"]["location"])
```

- As with previous APIs, it returns a JSON object with a list of results.
- Each result provides detailed information such as the address, type of location and a unique id.
- Within the "geometry" field there is:
  - a "location" field with the latitude and longitude.
  - a "viewport" field with the northeast and southwest coordinates of the respective bbox.
- The "type_id" can be used with the Places API to obtain further information, reviews, etc…

google_geocode.py

Google Geocoding API

https://developers.google.com/maps/documentation/geocoding/start

```
[{'address_components': [{'long_name': 'John F. Kennedy International Airport',
                          'short_name': 'John F. Kennedy International Airport',
                          'types': ['airport',
                                    'establishment',
                                    'point_of_interest']},
                         {'long_name': 'Queens',
                          'short_name': 'Queens',
                          'types': ['political',
                                    'sublocality',
                                    'sublocality_level_1']},
                         {'long_name': 'Queens County',
                          'short_name': 'Queens County',
                          'types': ['administrative_area_level_2',
                                    'political']},
                         {'long_name': 'New York',
                          'short_name': 'NY',
                          'types': ['administrative_area_level_1',
                                    'political']},
                         {'long_name': 'United States',
                          'short_name': 'US',
                          'types': ['country', 'political']},
                         {'long_name': '11430',
                          'short_name': '11430',
                          'types': ['postal_code']}],
'formatted_address': 'John F. Kennedy International Airport, Queens, NY '
                     '11430, USA',
'geometry': {'location': {'lat': 40.6413111, 'lng': -73.77813909999999},
             'location_type': 'APPROXIMATE',
             'viewport': {'northeast': {'lat': 40.64266008029149,
                                        'lng': -73.7767901197085},
                          'southwest': {'lat': 40.63996211970849,
                                        'lng': -73.7794880802915}}},
'place_id': 'ChIJR0lA1VBmwokR8BGfSBOyT-w',
'types': ['airport', 'establishment', 'point_of_interest']}]
```

Google Geocoding API

https://developers.google.com/maps/documentation/geocoding/start

```
[{'address_components': [{'long_name': 'John F. Kennedy International Airport',
                          'short_name': 'John F. Kennedy International Airport',
                          'types': ['airport',
                                    'establishment',
                                    'point_of_interest']},
                         {'long_name': 'Queens',
                          'short_name': 'Queens',
                          'types': ['political',
                                    'sublocality',
                                    'sublocality_level_1']},
                         {'long_name': 'Queens County',
                          'short_name': 'Queens County',
                          'types': ['administrative area level 2',
                                    'political']},
                         {'long_name': 'New York',
                          'short_name': 'NY',
                          'types': ['administrative_area_level_1',
                                    'political']},
                         {'long_name': 'United States',
                          'short_name': 'US',
                          'types': ['country', 'political']},
                         {'long_name': '11430',
                          'short_name': '11430',
                          'types': ['postal_code']}],
'formatted_address': 'John F. Kennedy International Airport, Queens, NY '
                     '11430, USA',
'geometry': {'location': {'lat': 40.6413111, 'lng': -73.77813909999999},
             'location_type': 'APPROXIMATE',
             'viewport': {'northeast': {'lat': 40.64266008029149,
                                        'lng': -73.7767901197085},
                          'southwest': {'lat': 40.63996211970849,
                                        'lng': -73.7794880802915}}},
'place_id': 'ChIJR0lA1VBmwokR8BGfSBOyT-w',
'types': ['airport', 'establishment', 'point_of_interest']}]
```

@bgon

Google Reverse Geocoding

https://developers.google.com/maps/documentation/geocoding/start

- Allows us to discover what a given set of coordinates represents
- .reverse_geocode((lat, lon)) - Just requires a lat/lon tuple.
- Google has access to many layers of GIS information and the results reflect this. The output is a list of results at various degrees of precision.
    - Building,
    - Nearest Road
    - County
    - Zip code
    - State
    - Country
    - etc…

Google Reverse Geocoding

https://developers.google.com/maps/documentation/geocoding/start


```
{'address_components': [{'long_name': 'New York',
                         'short_name': 'New York',
                         'types': ['locality', 'political']},
                        {'long_name': 'New York',
                         'short_name': 'NY',
                         'types': ['administrative_area_level_1', 'political']},
                        {'long_name': 'United States',
                         'short_name': 'US',
                         'types': ['country', 'political']}],
'formatted_address': 'New York, NY, USA',
'geometry': {'bounds': {'northeast': {'lat': 40.9175771,
                                      'lng': -73.70027209999999},
                        'southwest': {'lat': 40.4773991,
                                      'lng': -74.25908989999999}},
             'location': {'lat': 40.7127837, 'lng': -74.0059413},
             'location_type': 'APPROXIMATE',
             'viewport': {'northeast': {'lat': 40.9152555,
                                        'lng': -73.70027209999999},
                          'southwest': {'lat': 40.4960439,
                                        'lng': -74.2557349}}},
'place_id': 'ChIJOwg_06VPwokRYv534QaPC8g',
'types': ['locality', 'political']}
```

Google Reverse Geocoding

https://developers.google.com/maps/documentation/geocoding/start


```
{'address_components': [{'long_name': 'New York',
                         'short_name': 'New York',
                         'types': ['locality', 'political']},
                        {'long_name': 'New York',
                         'short_name': 'NY',
                         'types': ['administrative_area_level_1', 'political']},
                        {'long_name': 'United States',
                         'short_name': 'US',
                         'types': ['country', 'political']}],
'formatted_address': 'New York, NY, USA',
'geometry': {'bounds': {'northeast': {'lat': 40.9175771,
                                      'lng': -73.70027209999999},
                        'southwest': {'lat': 40.4773991,
                                      'lng': -74.25908989999999}},
             'location': {'lat': 40.7127837, 'lng': -74.0059413},
             'location_type': 'APPROXIMATE',
             'viewport': {'northeast': {'lat': 40.9152555,
                                        'lng': -73.70027209999999},
                          'southwest': {'lat': 40.4960439,
                                        'lng': -74.2557349}}},
'place_id': 'ChIJOwg_06VPwokRYv534QaPC8g',
'types': ['locality', 'political']}
```

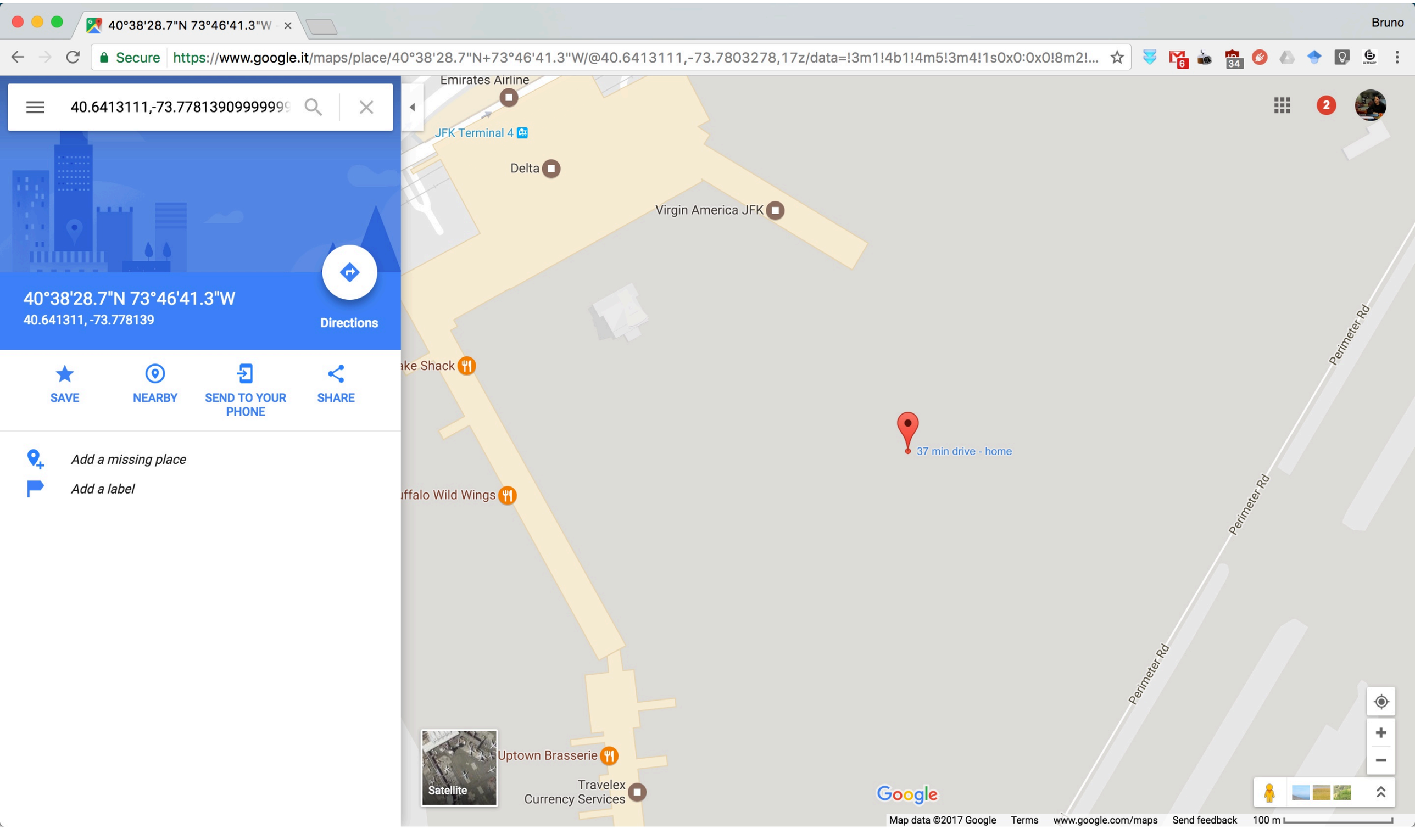
Google Reverse Geocoding

https://developers.google.com/maps/documentation/geocoding/start

```python
import googlemaps
from google_accounts import accounts

app = accounts["torino"]
gmaps = googlemaps.Client(key=app["api_key"])

reverse_result = gmaps.reverse_geocode((40.6413111,-73.77813909999999))

for result in reverse_result:
    print(result["types"], result["formatted_address"])
```

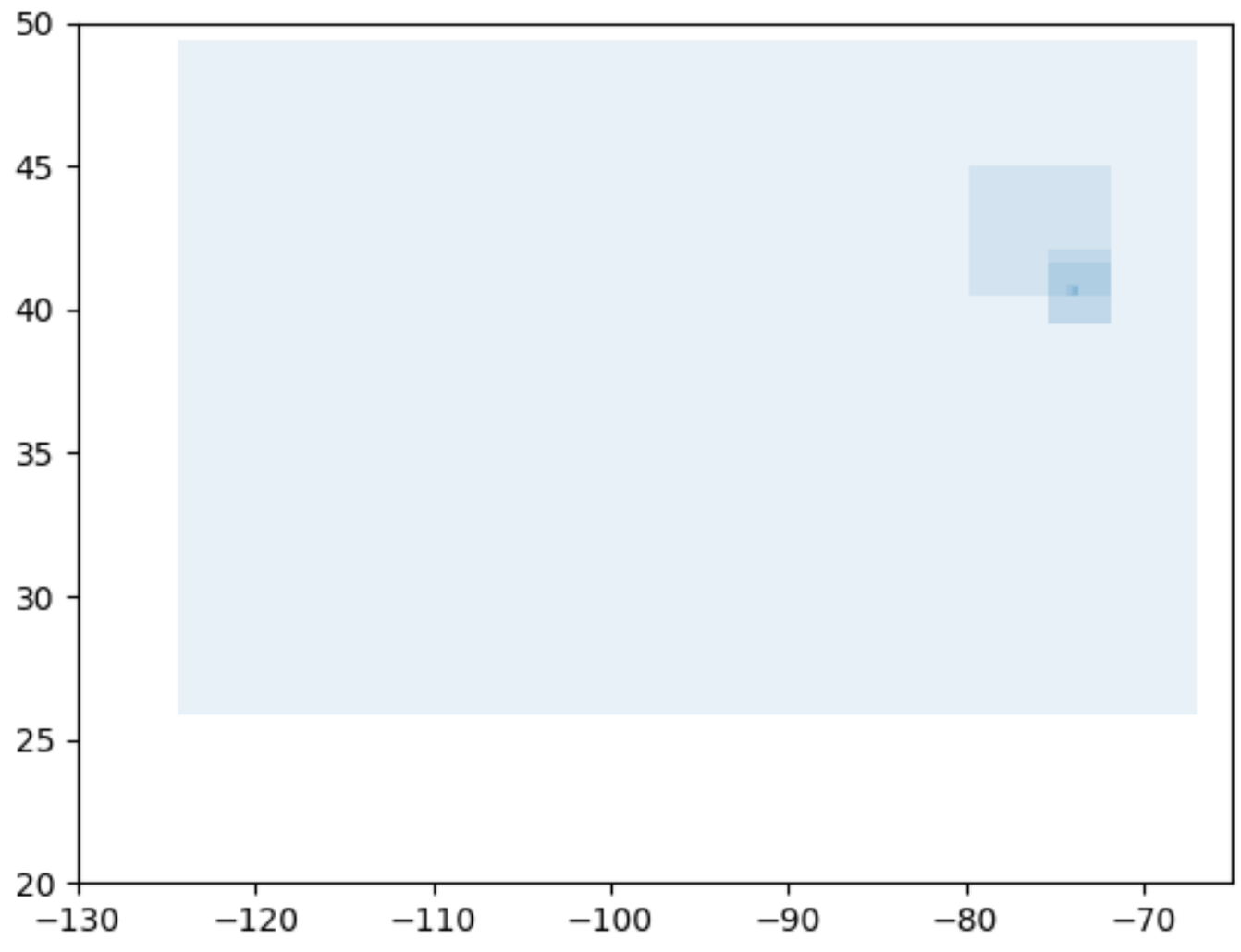
google_reverse_geocode.py

Google Reverse Geocoding



Google Reverse Geocoding

```
impo
impo
impo

reve

plt.
plt.
plt.
curr

for

plt.
```



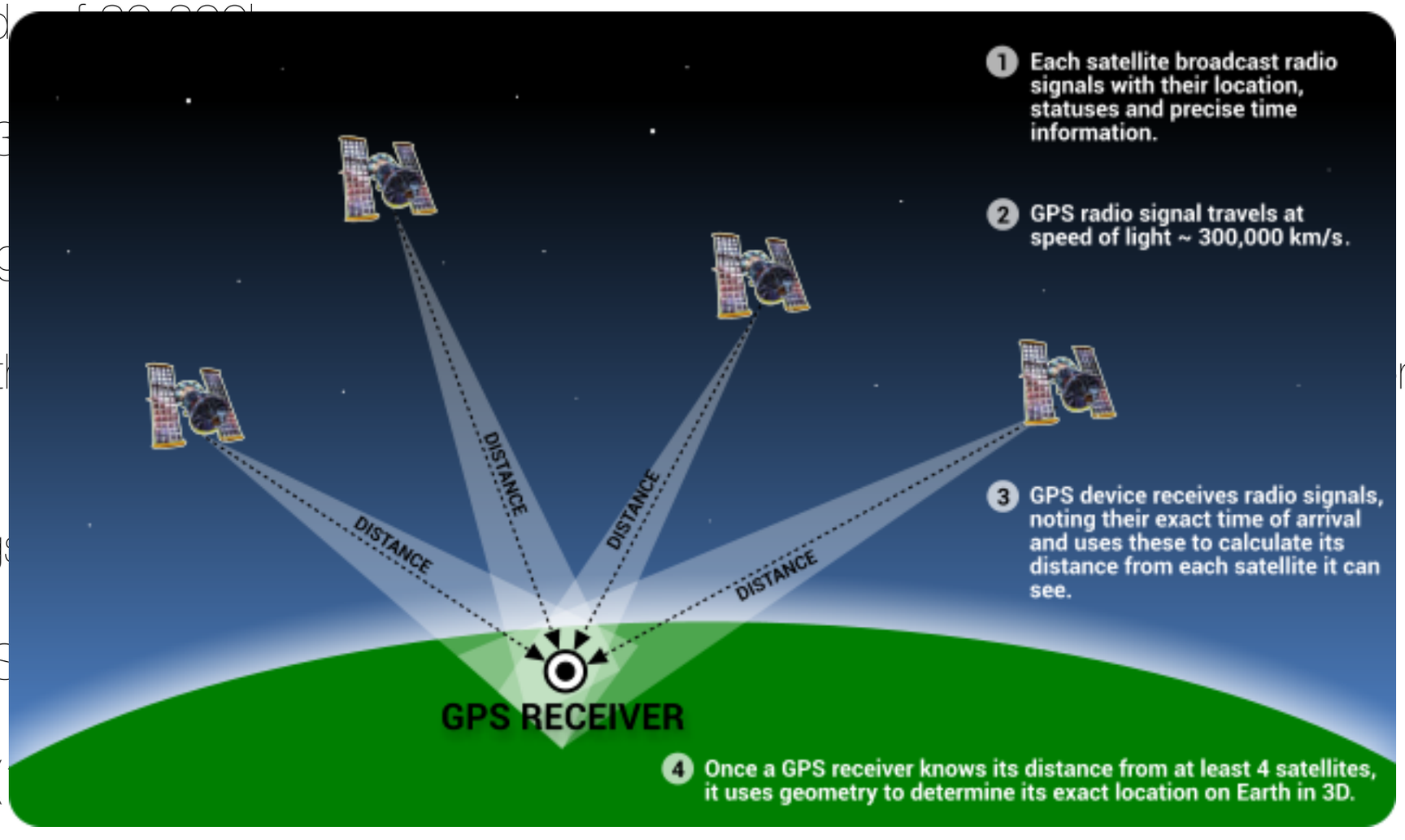

```
))
```

plot_rectangles.py

Geographical Information Systems

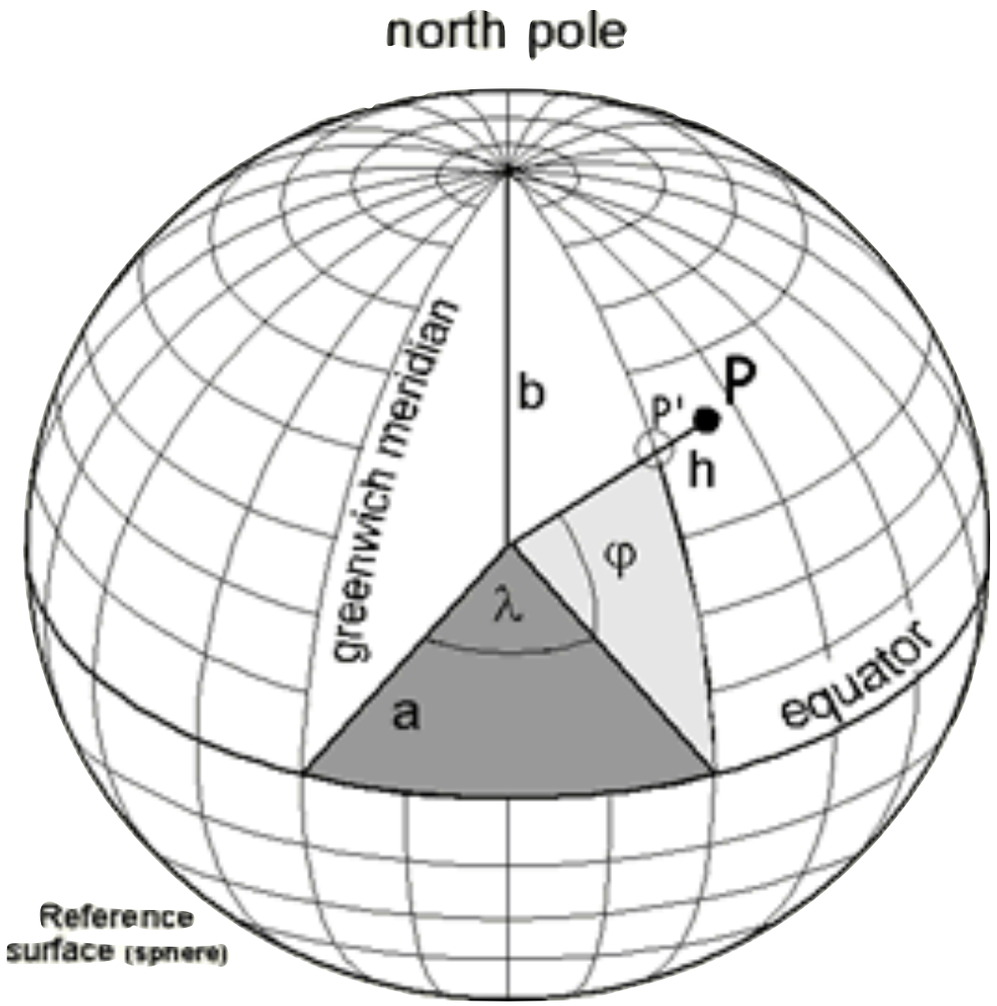
Global Positioning System (GPS)

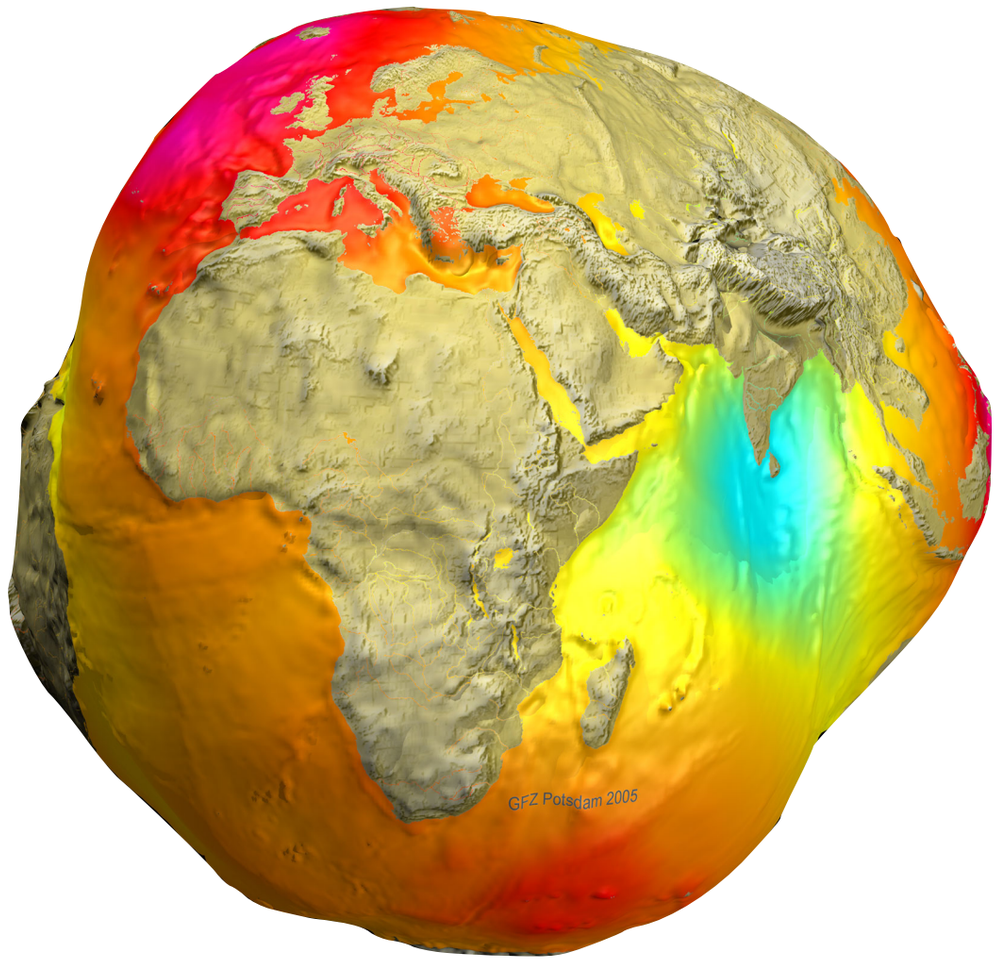
- 32 satellites
- Orbital altitud
- Traveling at 3
- Broadcasting
- Position on t m 4 or more satellites.
- GPS belongs
  - GLONAS
  - BeiDou (
  - Galileo - Europe



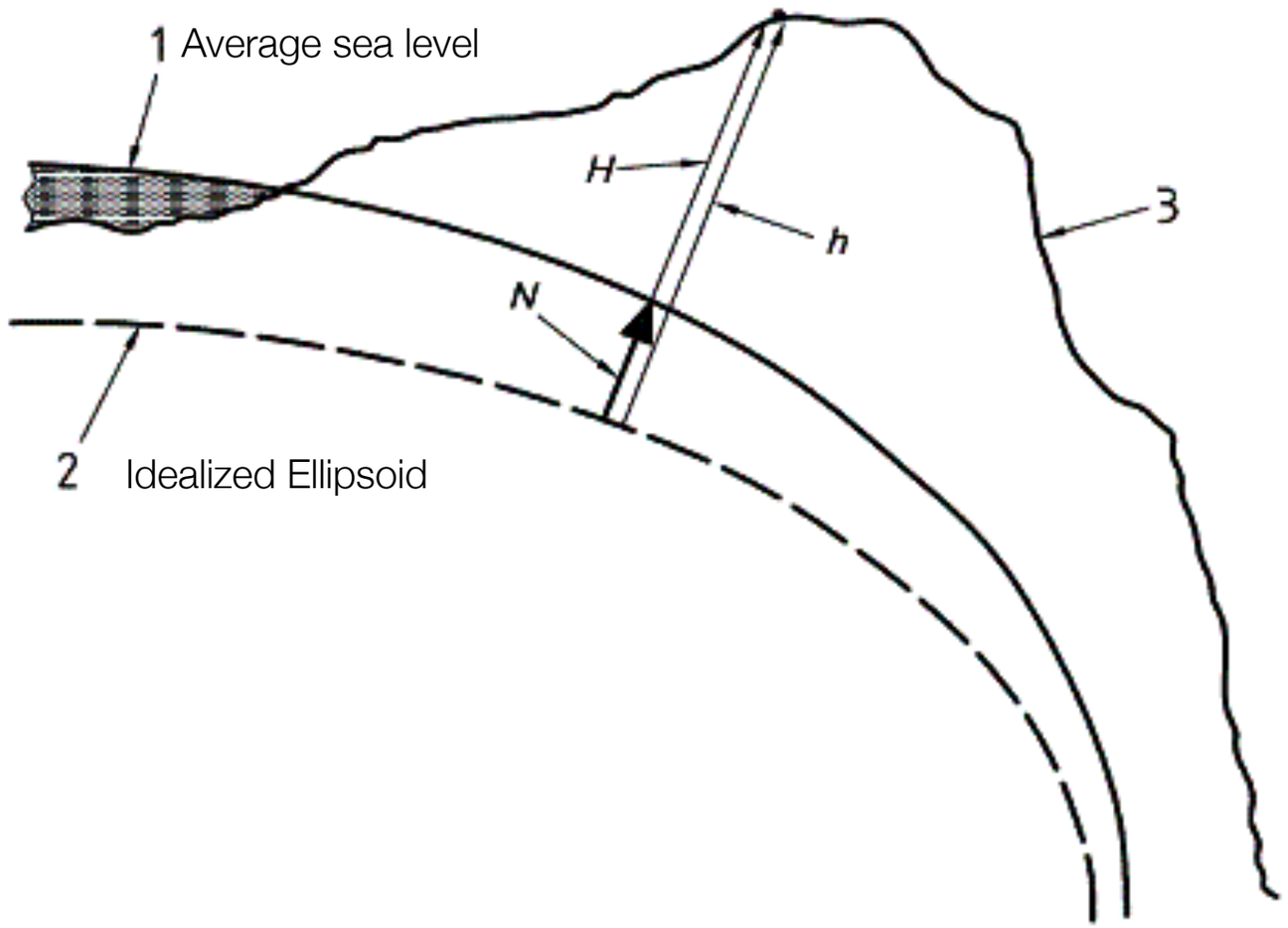
Global Positioning System (GPS)

Global Positioning System (GPS)





Global Positioning System (GPS)



Key

1 geoid
2 ellipsoid
3 surface of the Earth

h = ellipsoidal height, measured from ellipsoid along perpendicular passing through point; $h = H + N$
H = gravity-related height, measured along direction of gravity from vertical datum plane at geoide
N = geoid height, height of geoid above ellipsoid

WGS84

http://www.unoosa.org/pdf/icg/2012/template/WGS_84.pdf

- "WGS 84 is an Earth-centered, Earth-fixed terrestrial reference system and geodetic datum"

Coordinate System: Cartesian Coordinates (X, Y, Z). WGS 84 (G1674) follows the criteria outlined in the International Earth Rotation Service (IERS) Technical Note 21. The WGS 84 Coordinate System origin also serves as the geometric center of the WGS 84 Ellipsoid and the Z-axis serves as the rotational axis of this ellipsoid of revolution. WGS 84 geodetic coordinates are generated by using its reference ellipsoid.

Defining Parameters: WGS 84 identifies four defining parameters. These are the semi-major axis of the WGS 84 ellipsoid, the flattening factor of the Earth, the nominal mean angular velocity of the Earth, and the geocentric gravitational constant as specified below.

Parameter	Notation	Value
Semi-major Axis	a	6378137.0 meters
Flattening Factor of the Earth	1/f	298.257223563
Nominal Mean Angular Velocity of the Earth	ω	7292115×10^{-11} radians/second
Geocentric Gravitational Constant (Mass of Earth's Atmosphere Included)	GM**	$3.986004418 \times 10^{14}$ meter^3/second^2

**The value of GM for GPS users is 3.9860050×10^{14} m^3/sec^2 as specified in the references below.

GIS Data Systems

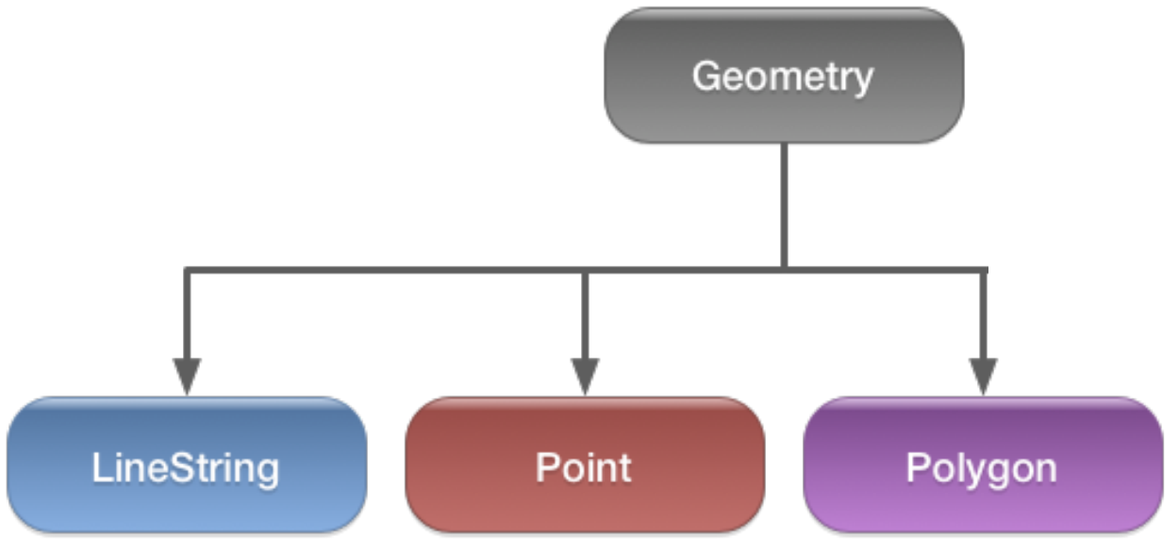
Vector

- Data types:
  - Points
  - Lines
  - Polygons
- Discrete shapes and boundaries
- Spatial, Database and Network analysis
- Shapefile, GeoJSON, GML, etc…

Raster

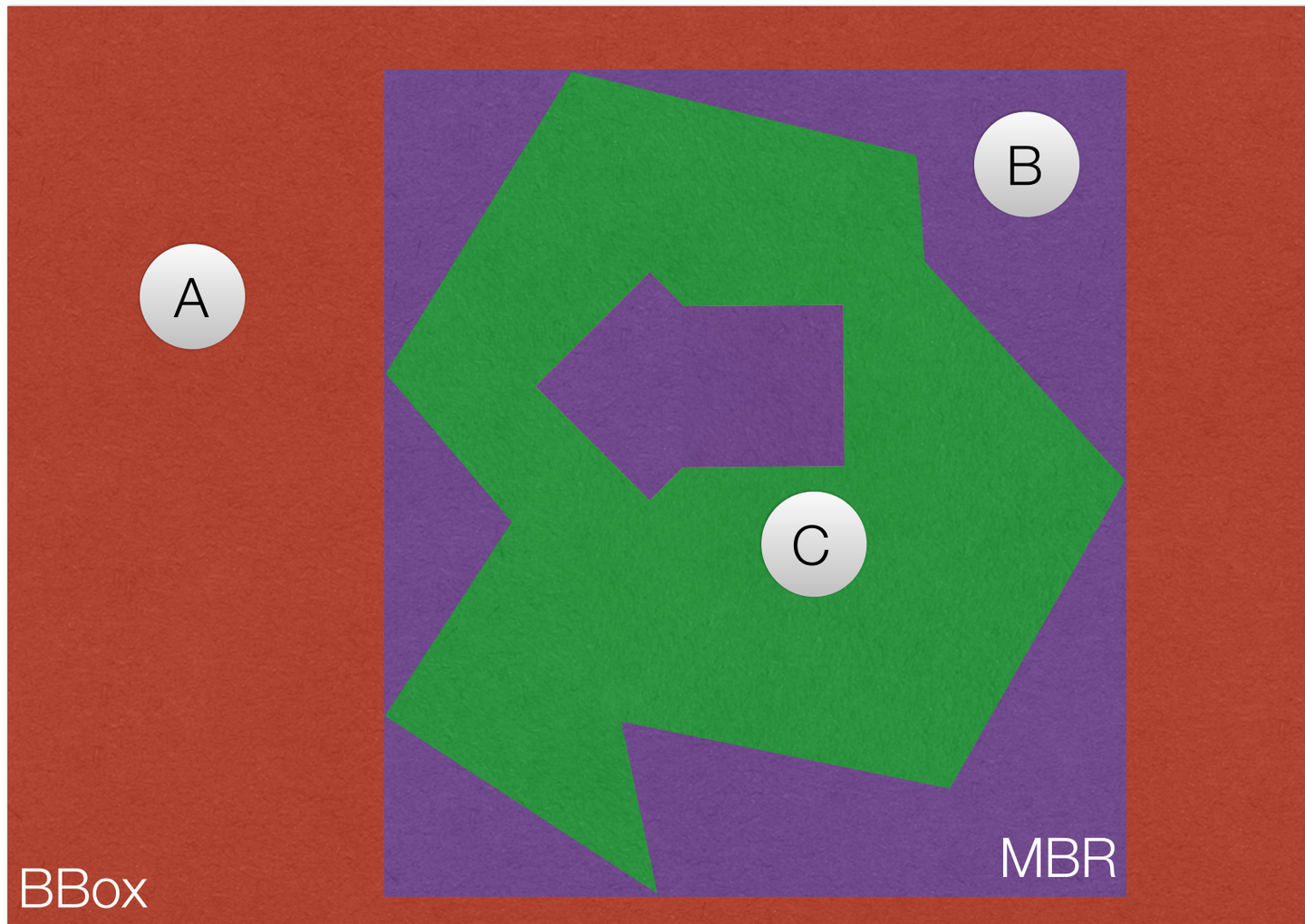
- Data types:
  - Cells
  - Pixels
  - Elements
- Dense data, Continuous surfaces
- Spatial Analysis and modeling
- GeoTIFF, ASC, JPEG2000, etc…

GIS Vector Data types



Fundamental Concepts

- **Bounding box** - A rectangular box containing the shape of interest
- **Minimum Bounding Rectangle** - The smallest possible Bounding Box that still contains the shape
- **Spatial Reference Identifier** - Maps "flat" lat/lon to a curved surface
  - Specially important to measure distances!
- Rectangles make it easy to quickly check relative positions, but lack precision
  - A within BBox but outside MBR
  - B within MBR but outside shape
  - C within shape
- If BBoxes don't overlap, neither do the shapes they contain

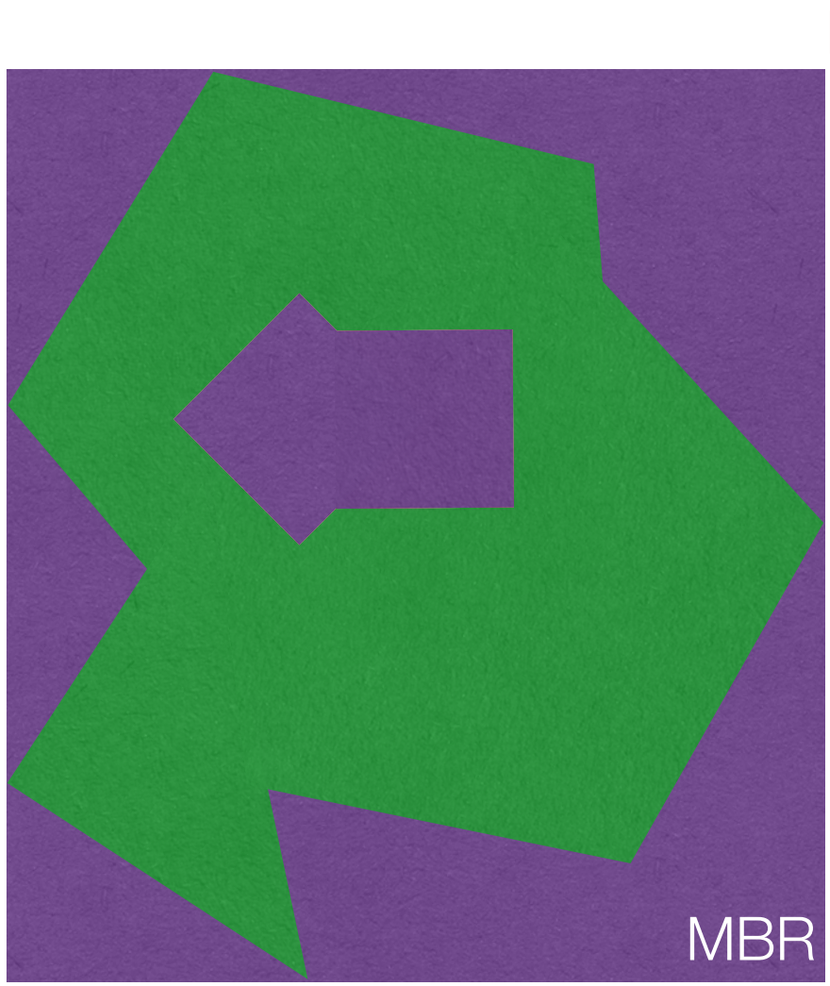


Get MBR/BBox

- You'll often see the term **BBox** to mean the **MBR**

- The MBR can be uniquely defined by the coordinates of two corners.

- Coordinates are simply the extreme values in the four "cardinal" directions, North, South, East and West of all the points defining the shape.



GeoJSON

https://tools.ietf.org/html/rfc7946

- A GeoJSON object is a JSON object specially tailored to spatial data

- It defines

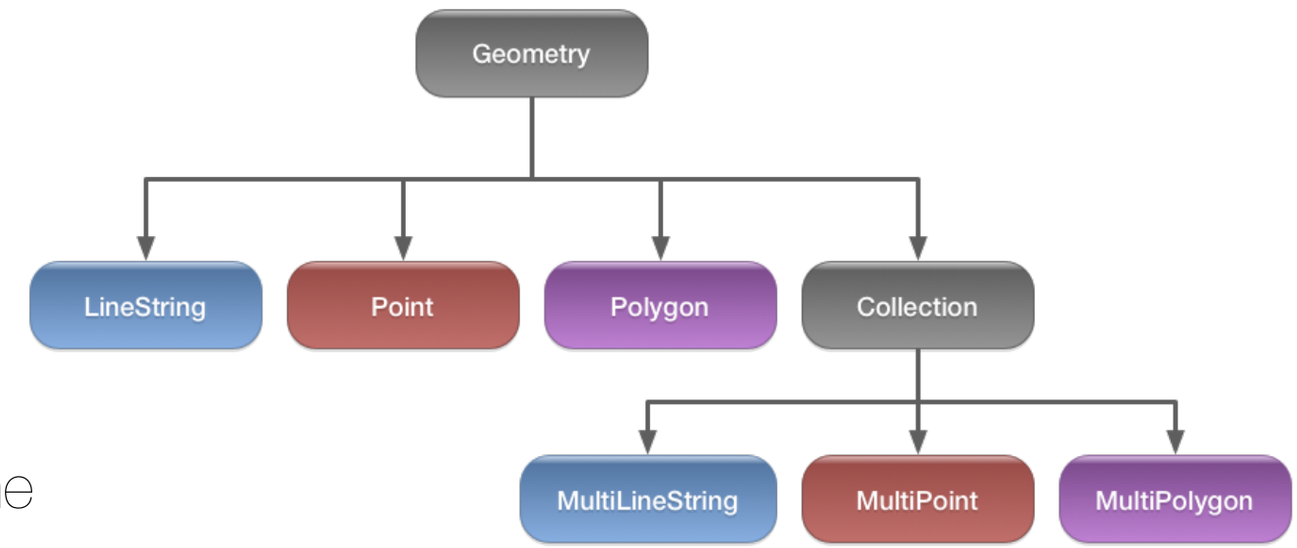
    - Geometry - a region of space

    - Feature - a spatially bounded entity that contains a contains a Geometry object

    - FeatureCollection - A list of Features

- Supports all types of GIS data types:

- Widely supported as an open standard online



GeoJSON

```
['type',
 'crs',
 'features']
```

GeoJSON

```
['type',          'FeatureCollection'
 'crs',           {"properties": {"name": "urn:ogc:def:crs:EPSG::4258"},
 'features']       "type": "name"}
```

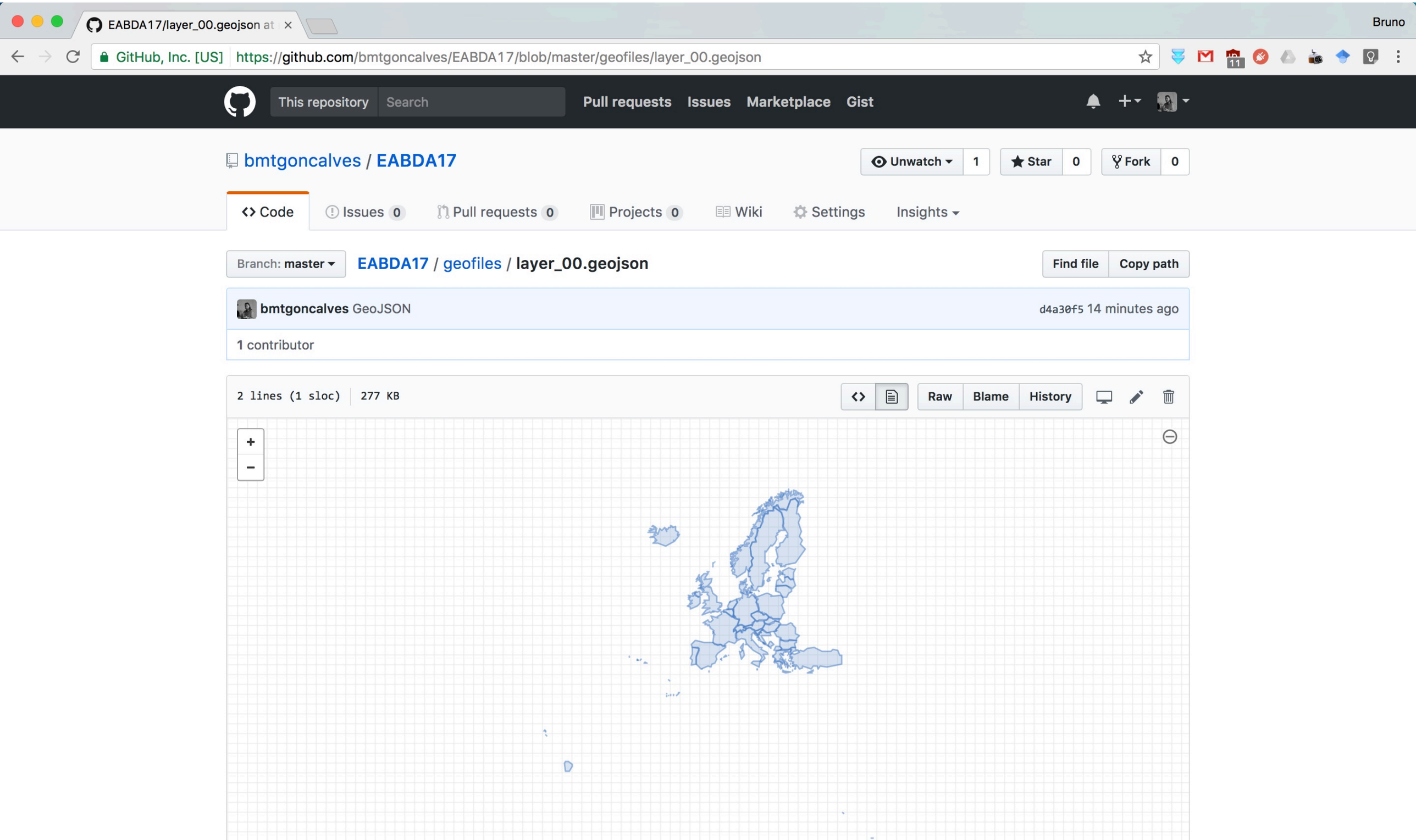
GeoJSON

```
['type',
 'crs',
 'features']   {'geometry': {'coordinates': [[[[19.224069, 43.527541],
                                               [19.227058, 43.47903949800002],
                                               (…)
                                               [19.049223, 43.501789001000002],
                                               [19.224069, 43.527541]]]],
                             'type': 'MultiPolygon'},
                'properties': {'NUTS_ID': 'ME',
                               'SHAPE_AREA': 1.50797533788,
                               'SHAPE_LEN': 7.40877787465,
                               'STAT_LEVL_': 0},
               'type': 'Feature'}
```

GeoJSON

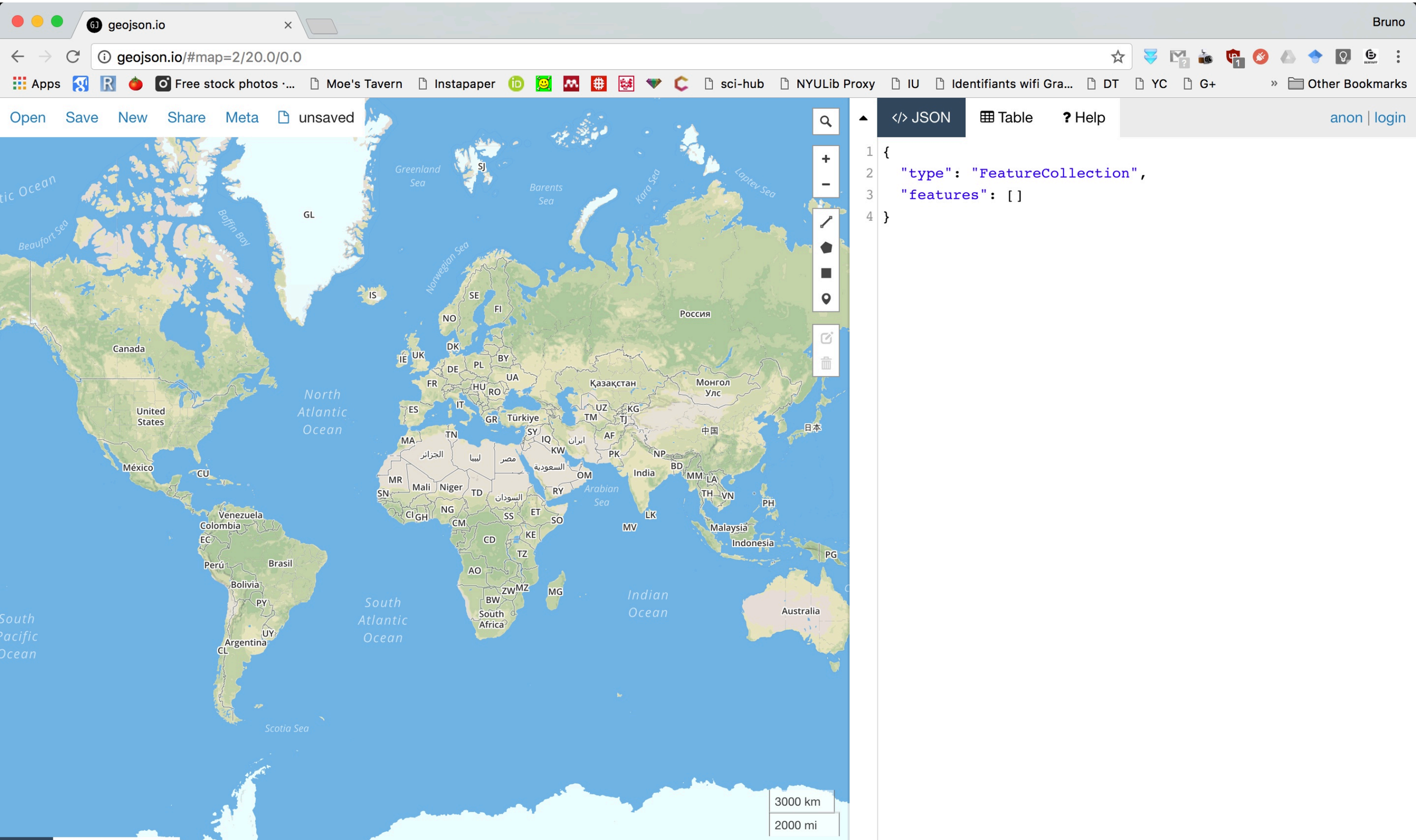
```
['type',
 'crs',
 'features']   {'geometry': {'coordinates': [[[[19.224069, 43.527541],
                                               [19.227058, 43.47903949800002],
                                               (…)
                                               [19.049223, 43.501789001000002],
                                               [19.224069, 43.527541]]]],
                             'type': 'MultiPolygon'},
                'properties': {'NUTS_ID': 'ME',
                               'SHAPE_AREA': 1.50797533788,
                               'SHAPE_LEN': 7.40877787465,
                               'STAT_LEVL_': 0},
               'type': 'Feature'}
```

GitHub support



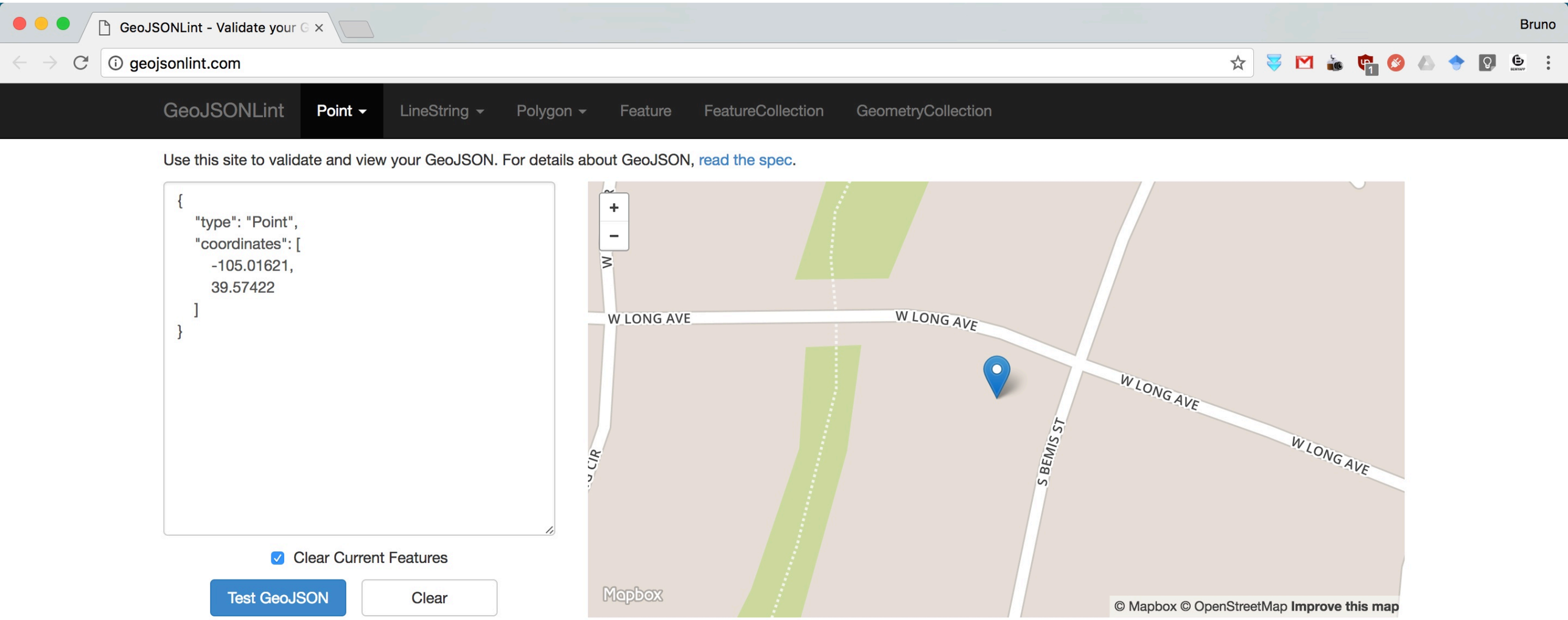
GeoJSON

geojson.io



GeoJSON

http://geojsonlint.com/



GeoJSON

```python
def get_bbox(country):
    maxLat = None
    maxLon = None
    minLat = None
    minLon = None

    for polygon in country["geometry"]["coordinates"]:
        coords = np.array(polygon)[0]

        curMaxLat = np.max(coords.T[1])
        curMinLat = np.min(coords.T[1])

        curMaxLon = np.max(coords.T[0])
        curMinLon = np.min(coords.T[0])

        if maxLat is None or curMaxLat > maxLat:
            maxLat = curMaxLat

        if maxLon is None or curMaxLon > maxLon:
            maxLon = curMaxLon

        if minLat is None or curMinLat < minLat:
            minLat = curMinLat

        if  minLon is None or curMinLon < minLon:
            minLon = curMinLon

    return maxLat, maxLon, minLat, minLon
```

get_bbox.py

GeoJSON

```
def plot_country(country):
    for polygon in country["geometry"]["coordinates"]:
        coords = np.array(polygon)

        plt.plot(coords.T[0], coords.T[1])

    maxLat, maxLon, minLat, minLon = get_bbox(country)

    plt.xlim(minLon, maxLon)
    plt.ylim(minLat, maxLat)
```

```
data = json.load(open('geofiles/NUTS_RG_20M_2013.geojson'))

countries = {}

countries["crs"] = data["crs"]
countries["type"] = data["type"]
countries = {}

for feat in data["features"]:
    if feat["properties"]["STAT_LEVL_"] == 0:
        countries[feat["properties"]["NUTS_ID"]] = feat

country = countries["EL"]

plot_country(country)
plt.savefig('Greece.png')
```


get_bbox.py

Shapefiles

http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf

- Open specification developed by ESRI, still the current leader in commercial GIS software
- shapefiles aren't actua (individual) files…
- but actually a set of files sharing the same name but with different extensions:

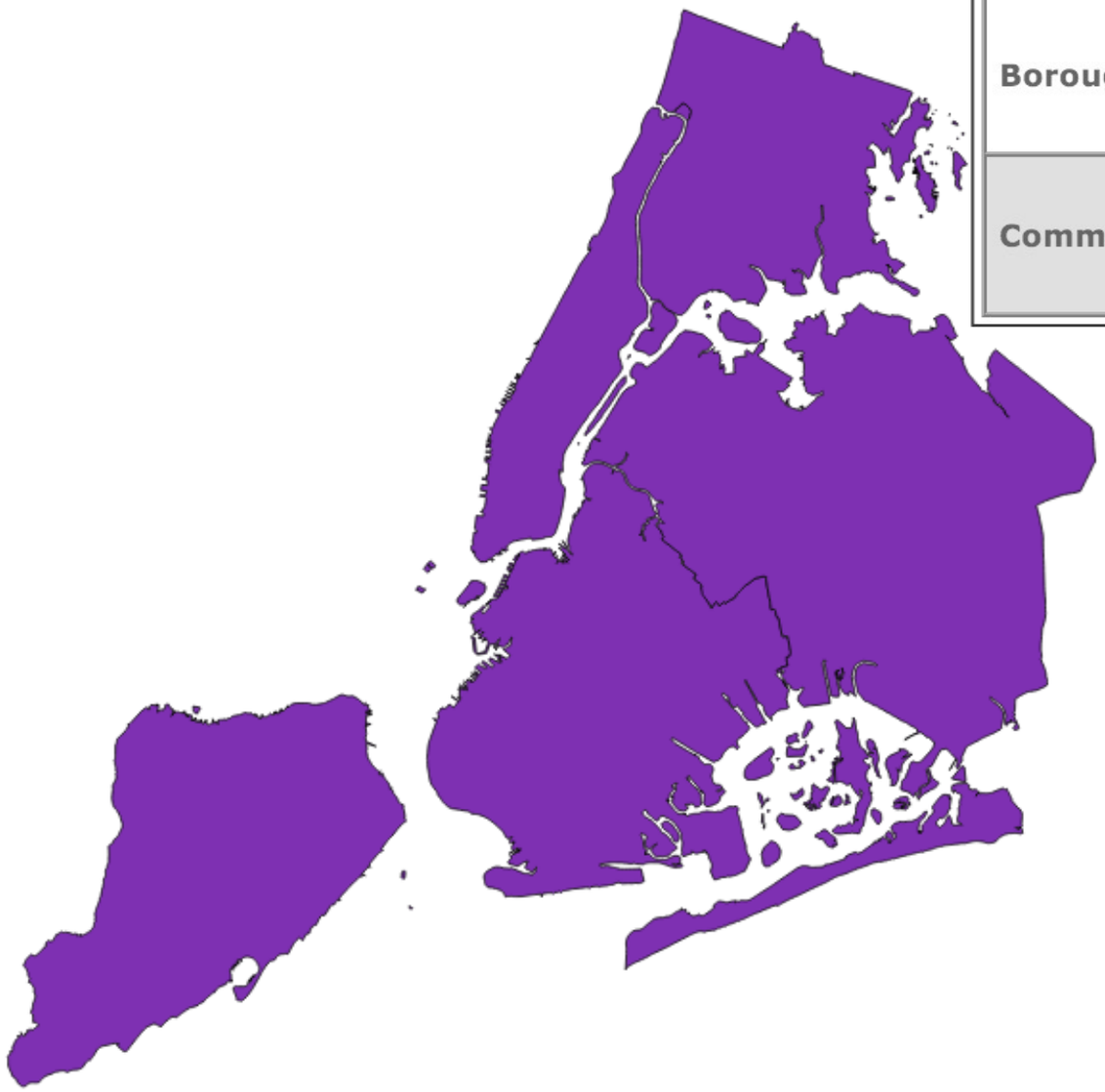
```
(py35) (master) bgoncalves@underdark:$ls -l
total 4856
-rw-r--r--  1 bgoncalves  staff      536 Apr 17 12:40 nybb_wgs84.dbf
-rw-r--r--  1 bgoncalves  staff      143 Apr 17 12:40 nybb_wgs84.prj
-rw-r--r--  1 bgoncalves  staff      257 Apr 17 12:40 nybb_wgs84.qpj
-rw-r--r--  1 bgoncalves  staff  1217376 Apr 17 12:40 nybb_wgs84.shp
-rw-r--r--  1 bgoncalves  staff      140 Apr 17 12:40 nybb_wgs84.shx
(py35) (master) bgoncalves@underdark:$
```

- the actual set of files changes depending on the contents, but three files are usually present:
  - .shp - also commonly referred to as "the" shapefile. Contains the geometric information
  - .dbf - a simple database containing the feature attribute table.
  - .shx - a spatial index, not strictly required

Shapefiles

http://www.nyc.gov/html/dcp/html/bytes/districts_download_metadata.shtml#bcd

Borough Boundaries & Community Districts	Download	Metadata
Borough Boundaries (Clipped to Shoreline)	(645k)	
Borough Boundaries (Water Areas Included)	(31k)	
Community Districts (Clipped to Shoreline)	(772k)	



QGIS Demo

pyshp

https://github.com/GeospatialPython/pyshp

- pyshp defines utility functions to load and manipulate Shapefiles programmatically.
- The shapefile module handles the most common operations:
  - .Reader(filename) - Returns a Reader object
- Reader.records()/Reader.iterRecords() returns/iterates over the different records present in the shapefile
- Reader.shapes()/Reader.iterShapes() - returns/Iterates over the different shapes present in the shapefile
- Reader.shapeRecords()/Reader.iterShapeRecords() returns/Iterates over both shapes and records present in the shapefile
- Reader.record(index)/Reader.shape(index)/Reader.shapeRecord(index) - return the record/shape/shapeRecord at index position index
- Reader.numRecords - returns the number of records in the shapefile

pyshp

https://github.com/GeospatialPython/pyshp

```
import sys
import shapefile

shp = shapefile.Reader('geofiles/nybb_15c/nybb_wgs84.shp')

print("Found", shp.numRecords, "records:")

recordDict = dict(zip([record[1] for record in shp.iterRecords()], range(shp.numRecords)))

for record, id in recordDict.items():
    print(id, record)
```


shapefile_load.py

pyshp

https://github.com/GeospatialPython/pyshp
http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf

- shape objects contain several fields:
  - bbox - lower left and upper right x,y coordinates (long/lat) - optional
  - parts - list of indexes for the first point of each of the parts making up the shape.
  - points - x,y coordinates for each point in the shape.
  - shapeType - integer representing the shape type - all shapes in a shapefile are required to be of the same shapeType or null.

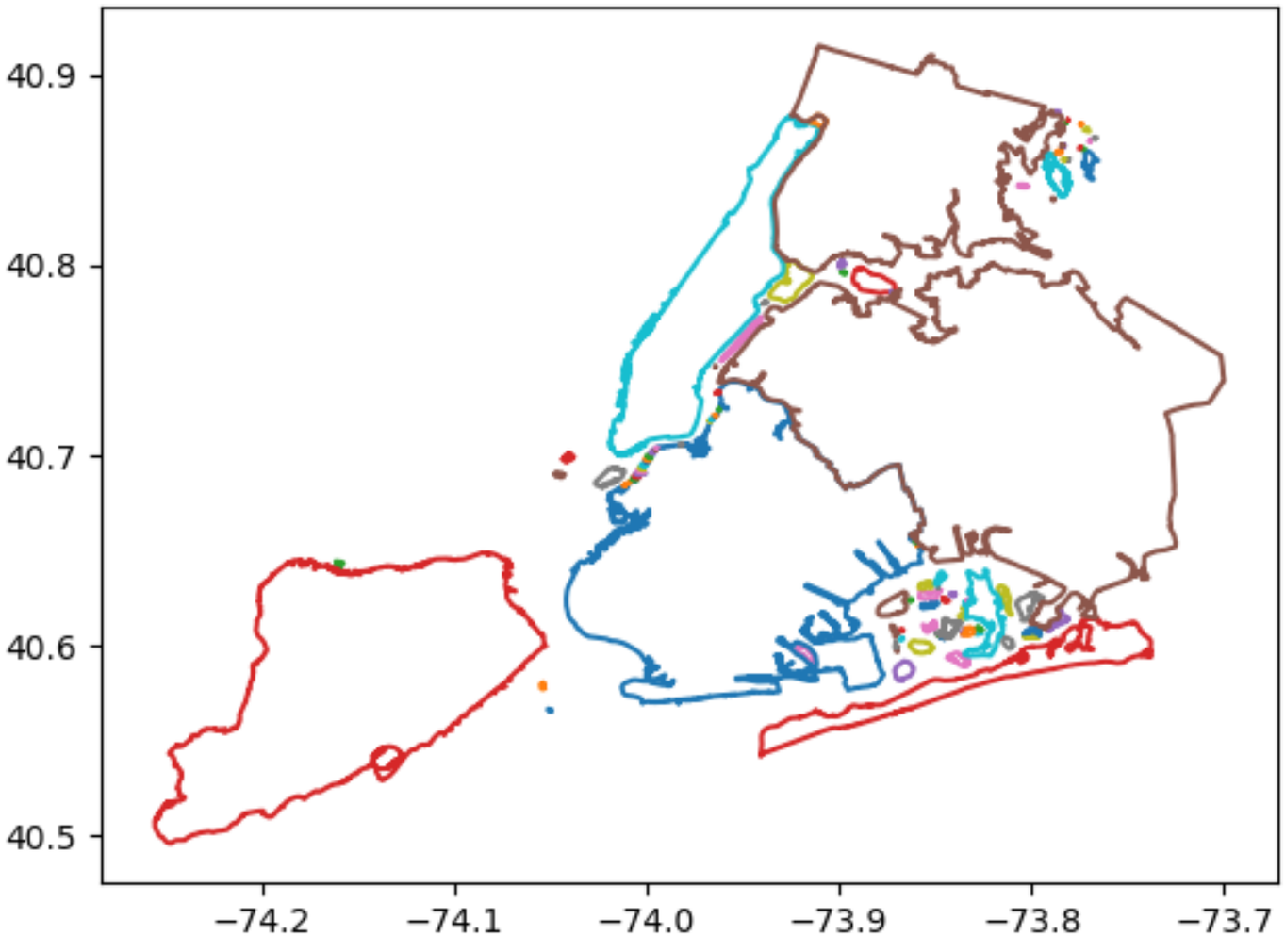
Value	Shape Type
0	Null Shape
1	Point
3	PolyLine
5	Polygon
8	MultiPoint
11	PointZ
13	PolyLineZ
15	PolygonZ
18	MultiPointZ
21	PointM
23	PolyLineM
25	PolygonM
28	MultiPointM
31	MultiPatch

pyshp

```
import shapefile
import matplotlib.pyplot as plt
import numpy as np

shp = shapefile.Reader('geofiles/nybb_15c/nybb_wgs84.shp')

pos = None
count = 0
for shape in shp.iterShapes():
    points = np.array(shape.points)
    parts = shape.parts
    parts.append(len(shape.points))

    for i in range(len(parts)-1):
        plt.plot(points.T[0][parts[i]:parts[i+1]], points.T[1][parts[i]:parts[i+1]])

plt.savefig('NYC.png')
```


plot_shapefile.py

pyshp



@bgoncalves

shapely

http://toblerity.org/shapely/manual.html

- Shapely defines geometric objects under shapely.geometry:
  - Point
  - Polygon
  - MultiPolygon
  - shape() Convenience function that creates the appropriate geometric object
- and common operations
  - .crosses(shape) - if it partially overlaps shape
  - .contains(shape) - wether it contains or not the object shape
  - .within(shape)- wether it is contained by object shape
  - .touches(shape) - if the boudaries of this object touch shape

shapely

http://toblerity.org/shapely/manual.html

- shape objects provide useful fields to query a shapes properties:
  - .centroid - The centroid ("center of mass") of the object
  - .area - returns the area of the object
  - .bounds - the MBR of the shape in (minx, miny, maxx, maxy) format
  - .length - the length of the shape
  - .geom_type - the Geometry Type of the object
- shapely.shape is also able to easily load pyshp's shape objects to allow for further manipulations.

shapely

http://toblerity.org/shapely/manual.html

```
import sys
import shapefile
from shapely.geometry import shape

shp = shapefile.Reader('geofiles/nybb_15c/nybb_wgs84.shp')

recordDict = dict(zip([record[1] for record in shp.iterRecords()], range(shp.numRecords)))

manhattan = shape(shp.shape(recordDict["Manhattan"]))

print("Centroid:", manhattan.centroid)
print("Bounding box:", manhattan.bounds)
print("Geometry type:", manhattan.geom_type)
print("Length:", manhattan.length)
```


shapefile_shape_properties.py

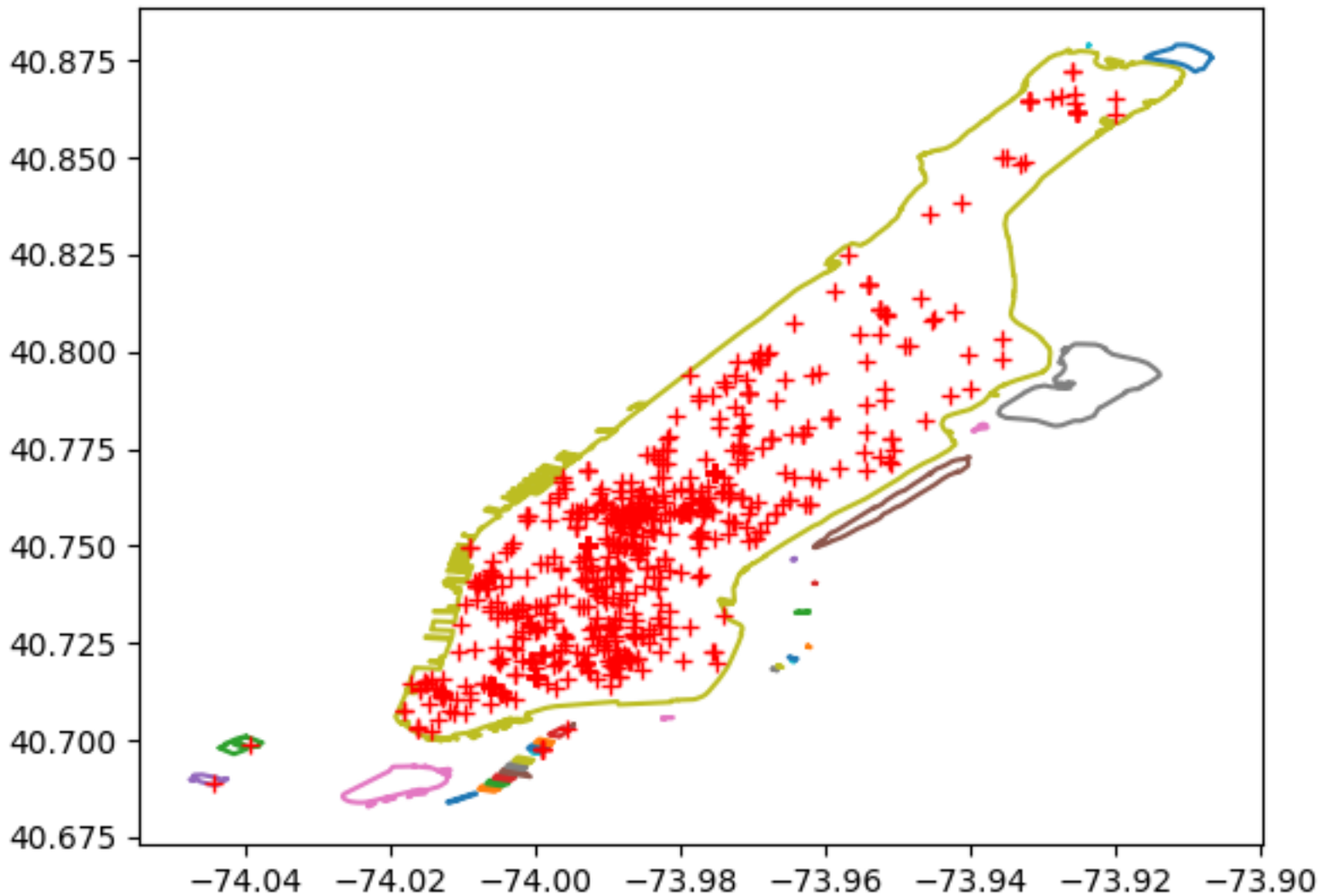
Filter points within a Shapefile

```
import sys
import shapefile
from shapely.geometry import shape, Point
import gzip

shp = shapefile.Reader('geofiles/nybb_15c/nybb_wgs84.shp')

recordDict = dict(zip([record[1] for record in shp.iterRecords()], range(shp.numRecords)))

manhattan = shape(shp.shape(recordDict["Manhattan"]))
fp = gzip.open("Manhattan.json.gz", "w")

for line in gzip.open("NYC.json.gz"):
    try:
        tweet = eval(line.strip())

        if "coordinates" in tweet and tweet["coordinates"] is not None:
            point = Point(tweet["coordinates"]["coordinates"])

            if manhattan.contains(point):
                fp.write(line)
    except:
        pass

fp.close()
```

@bgoncalves

shapefile_filter.py

Filter points within a Shapefile



@bgoncalves

Twitter places

- As we saw last week, Twitter defines a "coordinates" field in tweets
- There is also a "place" field that we glossed over.
- The **place** object contains also geographical information, but at a courser resolution than the **coordinates** field.
- Each place has a unique place_id, a bounding_box and some geographical information, such as **country** and **full_name**:

```
{'attributes': {},
 'bounding_box': {'coordinates': [[[-74.041878, 40.570842],
    [-74.041878, 40.739434],
    [-73.855673, 40.739434],
    [-73.855673, 40.570842]]],
  'type': 'Polygon'},
 'country': 'United States',
 'country_code': 'US',
 'full_name': 'Brooklyn, NY',
 'id': '011add077f4d2da3',
 'name': 'Brooklyn',
 'place_type': 'city',
 'url': 'https://api.twitter.com/1.1/geo/id/011add077f4d2da3.json'}
```

- places can be of several different types: 'admin', 'city', 'neighborhood', 'poi'

Twitter places

- As we saw earlier, Twitter defines a "coordinates" field in tweets
- There is also a "place" field that we glossed over.
- The **place** object contains also geographical information, but at a courser resolution than the **coordinates** field.
- Each place has a unique place_id, a bounding_box and some geographical information, such as **country** and **full_name**:

```
{'attributes': {},
 'bounding_box': {'coordinates': [[[-74.041878, 40.570842],
    [-74.041878, 40.739434],
    [-73.855673, 40.739434],
    [-73.855673, 40.570842]]],
  'type': 'Polygon'},
 'country': 'United States',
 'country_code': 'US',
 'full_name': 'Brooklyn, NY',
 'id': '011add077f4d2da3',
 'name': 'Brooklyn',
 'place_type': 'city',
 'url': 'https://api.twitter.com/1.1/geo/id/011add077f4d2da3.json'}
```

The bounding_box field is GeoJSON formatted and compatible with pyshp.shape

- places can be of several different types: 'admin', 'city', 'neighborhood', 'poi'

Twitter places

https://dev.twitter.com/overview/api/places

Place Attributes

Place Attributes are metadata about places. An attribute is a key-value pair of arbitrary strings, but with some conventions.

Below are a number of well-known place attributes which may, or may not exist in the returned data. These attributes are provided when the place was created in the Twitter places database.

Key	Description
street_address	
locality	the city the place is in
region	the administrative region the place is in
iso3	the country code
postal_code	in the preferred local format for the place
phone	in the preferred local format for the place, include long distance code
twitter	twitter screen-name, without @
url	official/canonical URL for place
app:id	An ID or comma separated list of IDs representing the place in the applications place database.

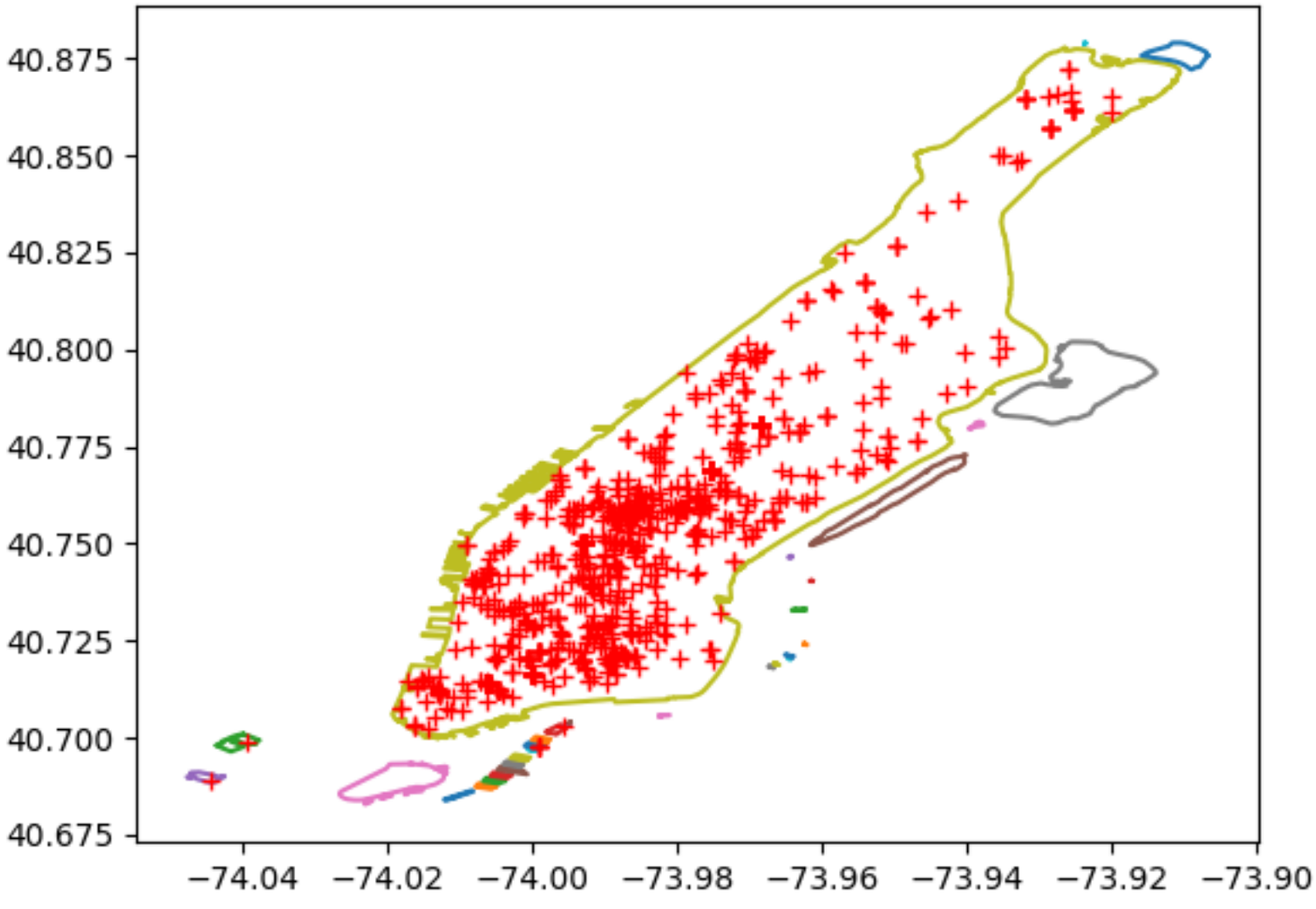
Keys can be no longer than 140 characters in length. Values are unicode strings and are restricted to 2000 characters.

Filter points and places

```
import sys
import shapefile
from shapely.geometry import shape, Point
import gzip

shp = shapefile.Reader('geofiles/nybb_15c/nybb_wgs84.shp')

recordDict = dict(zip([record[1] for record in shp.iterRecords()], range(shp.numRecords)))

manhattan = shape(shp.shape(recordDict["Manhattan"]))
fp = gzip.open("Manhattan_places.json.gz", "w")

for line in gzip.open("NYC.json.gz"):
    try:
        tweet = eval(line.strip())
        point = None

        if "coordinates" in tweet and tweet["coordinates"] is not None:
            point = Point(tweet["coordinates"]["coordinates"])
        else:
            if "place" in tweet and tweet["place"]["bounding_box"] is not None:
                bbox = shape(tweet["place"]["bounding_box"])
                point = bbox.centroid

        if point is not None and manhattan.contains(point):
            fp.write(line)
    except:
        pass

fp.close()
```

shapefile_filter_places.py

Filter points and places



Filter points and places

```
import sys
import gzip
import numpy as np
import shapefile
from shapely.geometry import shape, Point
import matplotlib.pyplot as plt

shp = shapefile.Reader('geofiles/nybb_15c/nybb_wgs84.shp')
recordDict = dict(zip([record[1] for record in shp.iterRecords()],
range(shp.numRecords)))

manhattan = shp.shape(recordDict["Manhattan"])

points = np.array(manhattan.points)
parts = manhattan.parts
parts.append(len(manhattan.points))

for i in range(len(parts)-1):
      plt.plot(points.T[0][parts[i]:parts[i+1]], points.T[1]
[parts[i]:parts[i+1]])

points_X = []
points_Y = []

for line in gzip.open(sys.argv[1]):
    try:
        tweet = eval(line.strip())
        point = None

        if "coordinates" in tweet and tweet["coordinates"] is not None:
            point = Point(tweet["coordinates"]["coordinates"])
        else:
            if "place" in tweet and tweet["place"]["bounding_box"] is not
None:
                  bbox = shape(tweet["place"]["bounding_box"])
                  point = bbox.centroid

        if point is not None:
            points_X.append(point.x)
            points_Y.append(point.y)
    except:
        pass

plt.plot(points_X, points_Y, 'r+')

plt.savefig(sys.argv[1] + '.png')
```

@bgoncalves

plot_shapefile_points.py

Calculating distances

https://en.wikipedia.org/wiki/Vincenty%27s_formulae
https://en.wikipedia.org/wiki/Great-circle_distance
https://en.wikipedia.org/wiki/Haversine_formula

- Earlier we saw how to obtain the distance between two points using the Google Maps API.

- But what is the shortest distance between two **arbitrary** points on the surface of the Earth?

- This depends strongly on our model of the Earth:

  - Great Circle - Assumes that the Earth is a perfect sphere of a given radius

    - Usually uses the Haversine formula $\Delta\sigma = 2\arcsin\sqrt{\sin^2\left(\frac{\Delta\phi}{2}\right) + \cos\phi_1 \cdot \cos\phi_2 \cdot \sin^2\left(\frac{\Delta\lambda}{2}\right)}$

  - Vincenty - Uses a (more) accurate ellipsoid model of the Earth

geopy

https://geopy.readthedocs.io/en/1.10.0/

- geopy provides two different types of functionality
    - geopy.geocoders - a unified interface to several geocoding services (Google Maps, Nominatim, Yahoo, Bing, etc…)
    - geopy.distance - state of the art distance calculations
- We will focus just on the distance module:
    - distance.vincenty(p1, p2) - Calculate the vincenty distance between p1 and p2
    - distance.great_circle(p1, p2) - Calculate the great circle distance between p1 and p2
    - distance.distance(p1, p2) - an alias to distance.vincenty to be used as a default.

geopy

https://geopy.readthedocs.io/en/1.10.0/

- all distance functions return a Distance object.
- the Distance object provides properties that represent the result in different units:
  - .km/.kilometers
  - .m/.meters
  - .mi/.miles
  - .ft/.feet
  - .nm/.nautical
- it also allows us to recalculate the result using different ellipsoids:
  - .set_ellipsoid('elliipsoid')
  - by default WGS-84 is used.

```
                 model             major (km)   minor (km)     flattening
ELLIPSOIDS = {'WGS-84':        (6378.137,    6356.7523142,  1 /
              'GRS-80':        (6378.137,    6356.7523141,  1 /
              'Airy (1830)':   (6377.563396, 6356.256909,   1 /
              'Intl 1924':     (6378.388,    6356.911946,   1 / 297.0),
              'Clarke (1880)': (6378.249145, 6356.51486955, 1 / 293.465),
              'GRS-67':        (6378.1600,   6356.774719,   1 / 298.25),
              }
```

geopy

https://geopy.readthedocs.io/en/1.10.0/

```
from geopy import distance

p1 = (41.49008, -71.312796)
p2 = (41.499498, -81.695391)

dist_vincenty = distance.vincenty(p1,
p2).meters
dist_great = distance.great_circle(p1,
p2).meters

print("Vincenty:", dist_vincenty)
print("Great Circles:", dist_great)
```

geopy_distance.py

GIS Data Systems

Vector

- Data types:
  - Points
  - Lines
  - Polygons
- Discrete shapes and boundaries
- Spatial, Database and Network analysis
- Shapefile, GeoJSON, GML, etc…

Raster

- Data types:
  - Cells
  - Pixels
  - Elements
- Dense data, Continuous surfaces
- Spatial Analysis and modeling
- GeoTIFF, ASC, JPEG2000, etc…

ASCII Grid

- Perhaps the simples raster file
- ASCII text based
- A small header

```
ncols         246
nrows         119
xllcorner     -126.500000000000
yllcorner     22.750000000000
cellsize      0.250000000000
NODATA_value  -9999
```

- Followed by rows of numbers
- Very convenient to Read and Write

geofiles/US_pop.asc

ASCII Gri

```python
import numpy as np
import matplotlib.pyplot as plt

def map_points(xllcorner, yllcorner, cellsize, nrows, x, y):
    x = int((x-xllcorner)/cellsize)
    y = (nrows-1)-int((y-yllcorner)/cellsize)

    return x, y

fp = open("geofiles/US_pop.asc")
ncols, count = fp.readline().split()
ncols = int(count)
nrows, count = fp.readline().split()
nrows = int(count)
xllcorner, value = fp.readline().split()
xllcorner = float(value)
yllcorner, value = fp.readline().split()
yllcorner = float(value)
cellsize, value = fp.readline().split()
cellsize = float(value)

NODATA_value, value = fp.readline().split()
NODATA_value = float(value)

data = []
for line in fp:
    fields = line.strip().split()
    data.append([float(field) for field in fields])

data = np.array(data)
data[data==NODATA_value] = 0

x = -74.243251
y = 40.730503

coord_x, coord_y = map_points(xllcorner, yllcorner, cellsize, nrows, x, y)
print(data[coord_y, coord_x])
```

asc_load.py

ASCII Gri

```python
import numpy as np
import matplotlib.pyplot as plt

def map_points(xllcorner, yllcorner, cellsize, nrows, x, y):
    x = int((x-xllcorner)/cellsize)
    y = (nrows-1)-int((y-yllcorner)/cellsize)

    return x, y

fp = open("geofiles/US_pop.asc")
ncols, count = fp.readline().split()
ncols = int(count)
nrows, count = fp.readline().split()
nrows = int(count)
xllcorner, value = fp.readline().split()
xllcorner = float(value)
yllcorner, value = fp.readline().split()
yllcorner = float(value)
cellsize, value = fp.readline().split()
cellsize = float(value)

NODATA_value, value = fp.readline().split()
NODATA_value = float(value)

data = []
for line in fp:
    fields = line.strip().split()
    data.append([float(field) for field in fields])

data = np.array(data)
data[data==NODATA_value] = 0

x = -74.243251
y = 40.730503

coord_x, coord_y = map_points(xllcorner, yllcorner, cellsize, nrows, x, y)
print(data[coord_y, coord_x])
```

asc_load.py

ASCII Grid

asc_load.py

ASCII Grid

- This type of grid is a very convenient way to aggregate spatial data.
- Simply map **lat, lon** pairs to matrix entries and then increment the values
- All we need is to define the **bbox** we are interested in, and the size of each cell and create a matrix with that shape.

```
import numpy as np
from shapely.geometry import shape, Point
import shapefile

shp = shapefile.Reader('geofiles/nybb_15c/nybb_wgs84.shp')
recordDict = dict(zip([record[1] for record in shp.iterRecords()],
range(shp.numRecords)))

manhattan = shp.shape(recordDict["Manhattan"])

xllcorner, yllcorner, xurcorner, yurcorner = manhattan.bbox

cellsize = 0.01

ncols = int((xurcorner-xllcorner)/cellsize)
nrows = int((yurcorner-yllcorner)/cellsize)

data = np.zeros((nrows, ncols), dtype='int')

print(data.shape)
```

asc_generate_grid.py

Aggregate

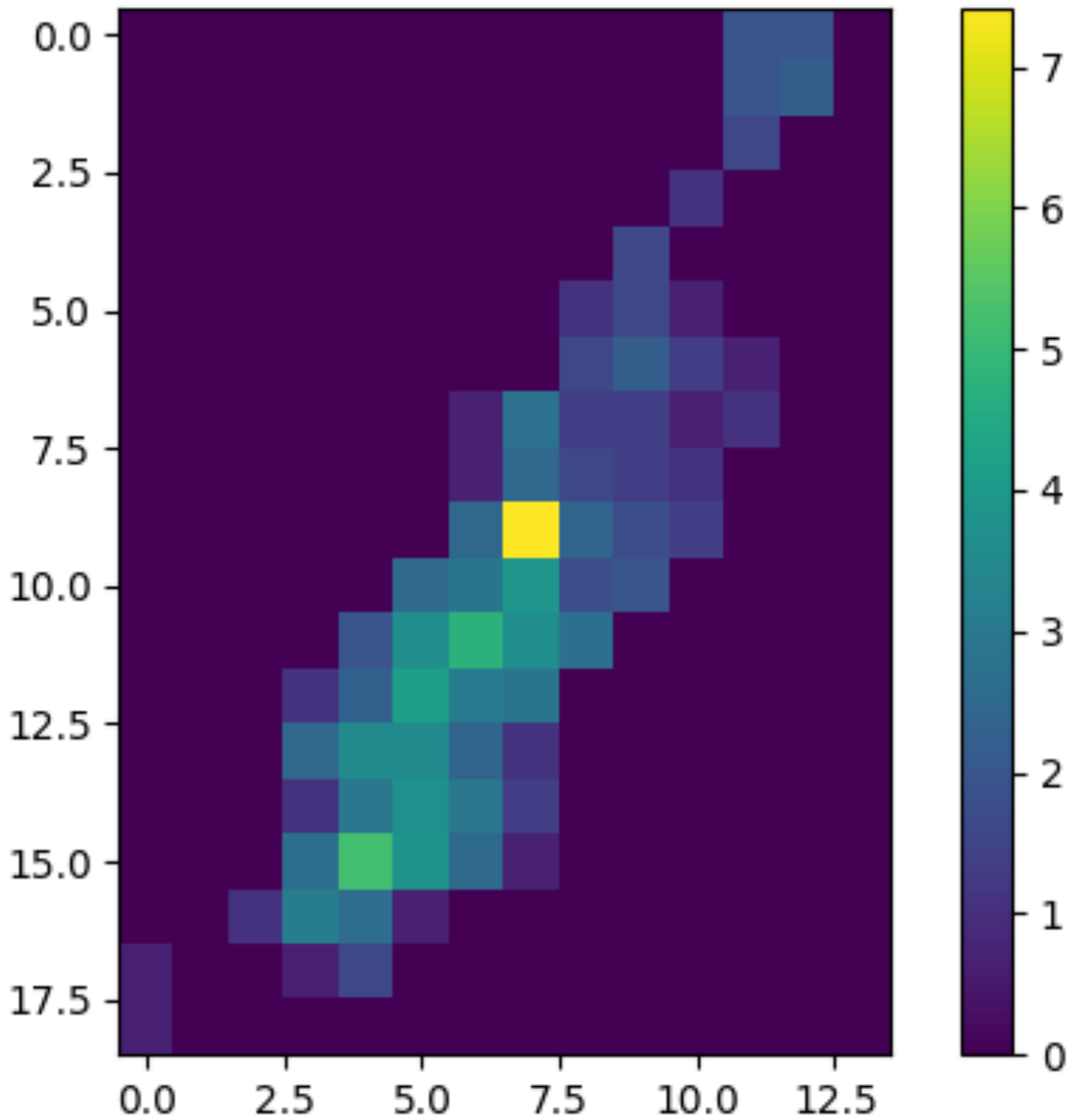
```
import sys
import numpy as np
import shapefile
from shapely.geometry import shape, Point
import matplotlib.pyplot as plt
import gzip

def map_points(xllcorner, yllcorner, cellsize, nrows, x, y):
    x = int((x-xllcorner)/cellsize)
    y = (nrows-1)-int((y-yllcorner)/cellsize)

    return x, y

def save_asc(data, xllcorner, yllcorner, cellsize, filename):
    fp = open(filename, "w")

    nrows, ncols = data.shape

    print("ncols", ncols, file=fp)
    print("nrows", nrows, file=fp)
    print("xllcorner", xllcorner, file=fp)
    print("yllcorner", yllcorner, file=fp)
    print("cellsize", cellsize, file=fp)
    print("NODATA_value", "-9999", file=fp)

    for i in range(nrows):
        for j in range(ncols):
            print(("%u " % data[i, j]), end="", file=fp)

        print("\n", end="", file=fp)

    fp.close()
```

shapefile_filter_aggregate.py

```
shp = shapefile.Reader('geofiles/nybb_15c/nybb_wgs84.shp')
recordDict = dict(zip([record[1] for record in shp.iterRecords()], range(shp.numRecords)))
manhattan = shape(shp.shape(recordDict["Manhattan"]))

xllcorner, yllcorner, xurcorner, yurcorner = manhattan.bounds
cellsize = 0.01

ncols = int((xurcorner-xllcorner)/cellsize)
nrows = int((yurcorner-yllcorner)/cellsize)

data = np.zeros((nrows, ncols), dtype='int')

for line in gzip.open("NYC.json.gz"):
    try:
        tweet = eval(line.strip())
        point = None

        if "coordinates" in tweet and tweet["coordinates"] is not None:
            point = Point(tweet["coordinates"]["coordinates"])
        else:
            if "place" in tweet and tweet["place"]["bounding_box"] is not None:
                bbox = shape(tweet["place"]["bounding_box"])
                point = bbox.centroid

        if point is not None and manhattan.contains(point):
            coord_x, coord_y = map_points(xllcorner, yllcorner, cellsize, nrows, point.x, point.y)
            data[coord_y, coord_x] += 1

    except:
        pass


save_asc(data, xllcorner, yllcorner, cellsize, "Manhattan.asc")

plt.imshow(np.log(data+1))
plt.colorbar()
plt.savefig('Manhattan_cells.png')
```

shapefile_filter_aggregate.py

Aggregate

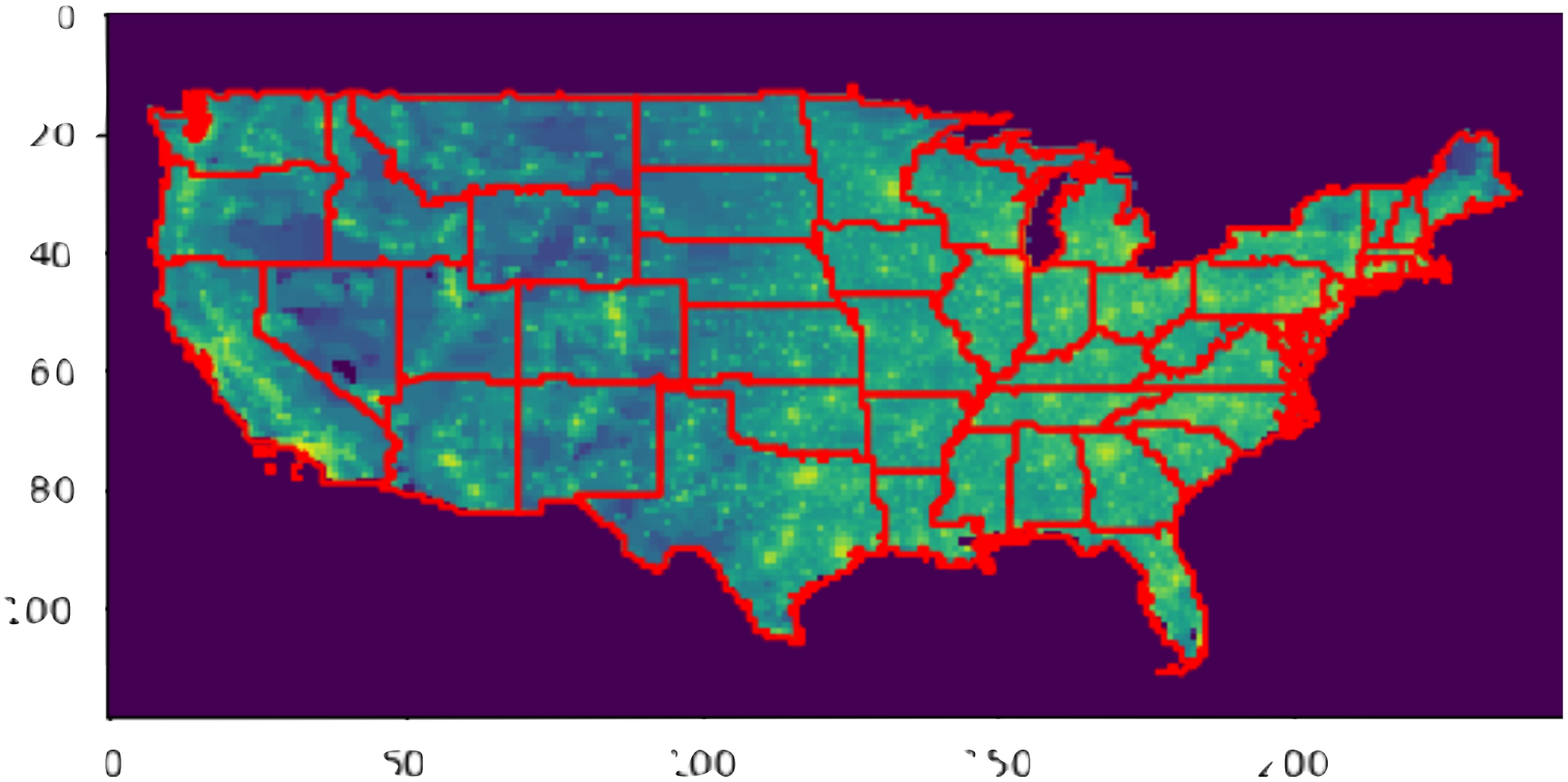


Overlap a raster and a shapefile

```
import numpy as np
import matplotlib.pyplot as plt
import shapefile

(…)

fig, ax = plt.subplots(1,1)
data, xllcorner, yllcorner, cellsize  = load_asc('geofiles/US_pop.asc')
ax.imshow(np.log(data+1))

shp = shapefile.Reader('geofiles/48States/48States.shp')

pos = None
count = 0
for shape in shp.iterShapes():
        points = np.array(shape.points)
        parts = shape.parts
        parts.append(len(shape.points))

        for i in range(len(parts)-1):
            positions = []

            for j in range(parts[i+1]-parts[i]):
                x_orig = points.T[0][parts[i]+j]
                y_orig = points.T[1][parts[i]+j]
                x, y = map_points(xllcorner, yllcorner, cellsize, data.shape[0], x_orig, y_orig)
                positions.append([x, y])

            positions = np.array(positions)

            ax.plot(positions.T[0], positions.T[1], 'r-')


fig.savefig('US overlap.png')
```

matplotlib_overlap.py

Overlap a raster and a shapefile



@bgoncalves

Matplotlib Basemap

Basemap

- The Basemap module is the workhorse and returns a Basemap object when instantiated.
- The Basemap object has many useful methods to assist in drawing a map:
  - .drawcoastlines() - To draw the coastlines of continents
  - .drawmapboundary() - To draw the boundary of the map
  - .fillcontinents() - add color to the continents
  - etc…
- The constructor for Basemap can take many different arguments to be able to handle different projections, but it defaults to the Plate Carrée projection centered at (0, 0)
- The minimal map is simply:

```
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt

map = Basemap()
map.drawcoastlines()
plt.savefig('basemap_demo.png')
```

- Please note that with the .drawcoastlines() call nothing is plotted as our map has no content.

Basemap

- We can also visualize just specific regions by setting the bbox and center coordinates by setting

llcrnrlon, llcrnrlat, urcrnrlon, urcrnrlat

- And

lat_0, lon_0

- Respectively.
- We can convert arbitrary lat, lon values to map coordinates by calling the map() object directly.
- After we obtain the map coordinates we can add them to the map by calling the .plot(x, y) method of the map object.

Basemap Example

```
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt

map = Basemap(projection='ortho', lat_0=0, lon_0=0)

map.drawmapboundary(fill_color='aqua')
map.fillcontinents(color='coral',lake_color='aqua')
map.drawcoastlines()

x, y = map(0, 0)

map.plot(x, y, marker='D',color='m')

plt.savefig('globe.png')
```

basemap_example.py

Basemap Example