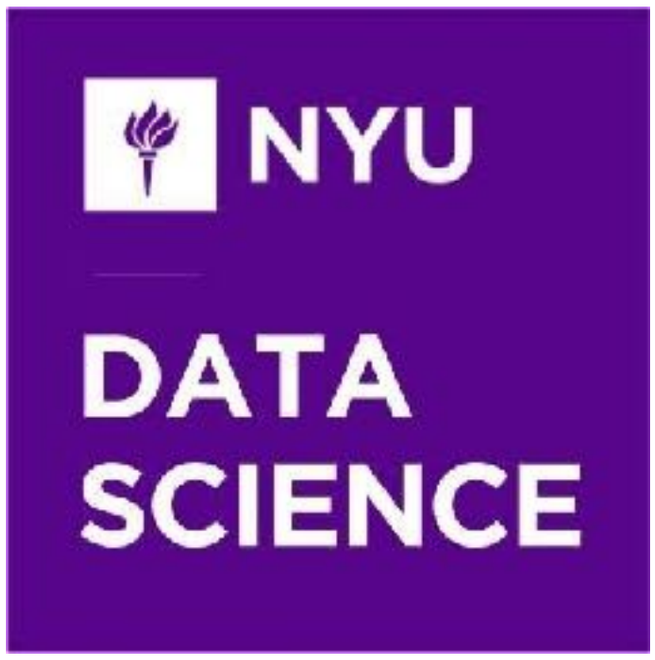
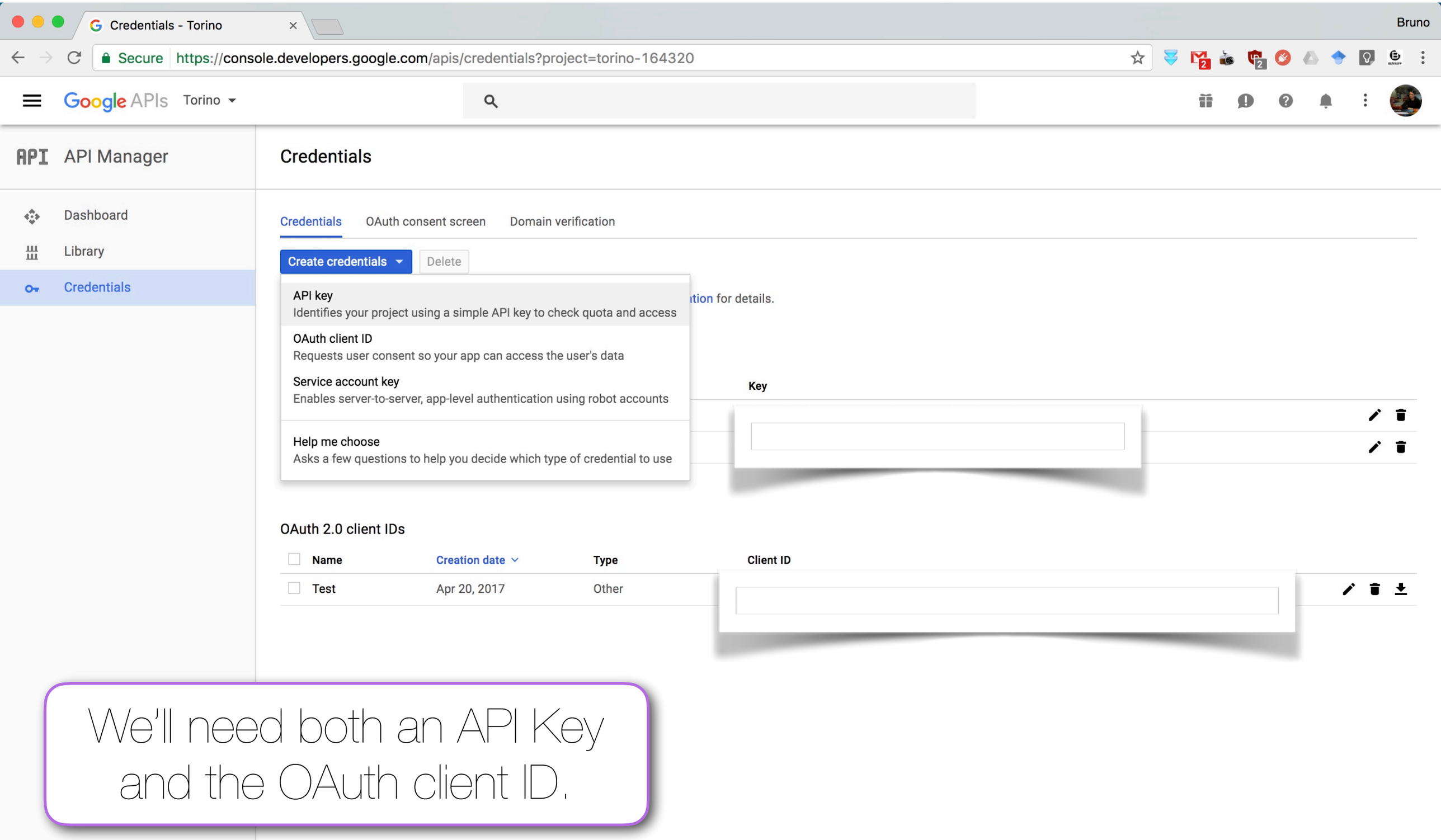
https://bmtgoncalves.github.io/TorinoCourse/

# Lecture IV - Geocoding and Spatial Analysis

Bruno Gonçalves
*www.bgoncalves.com*

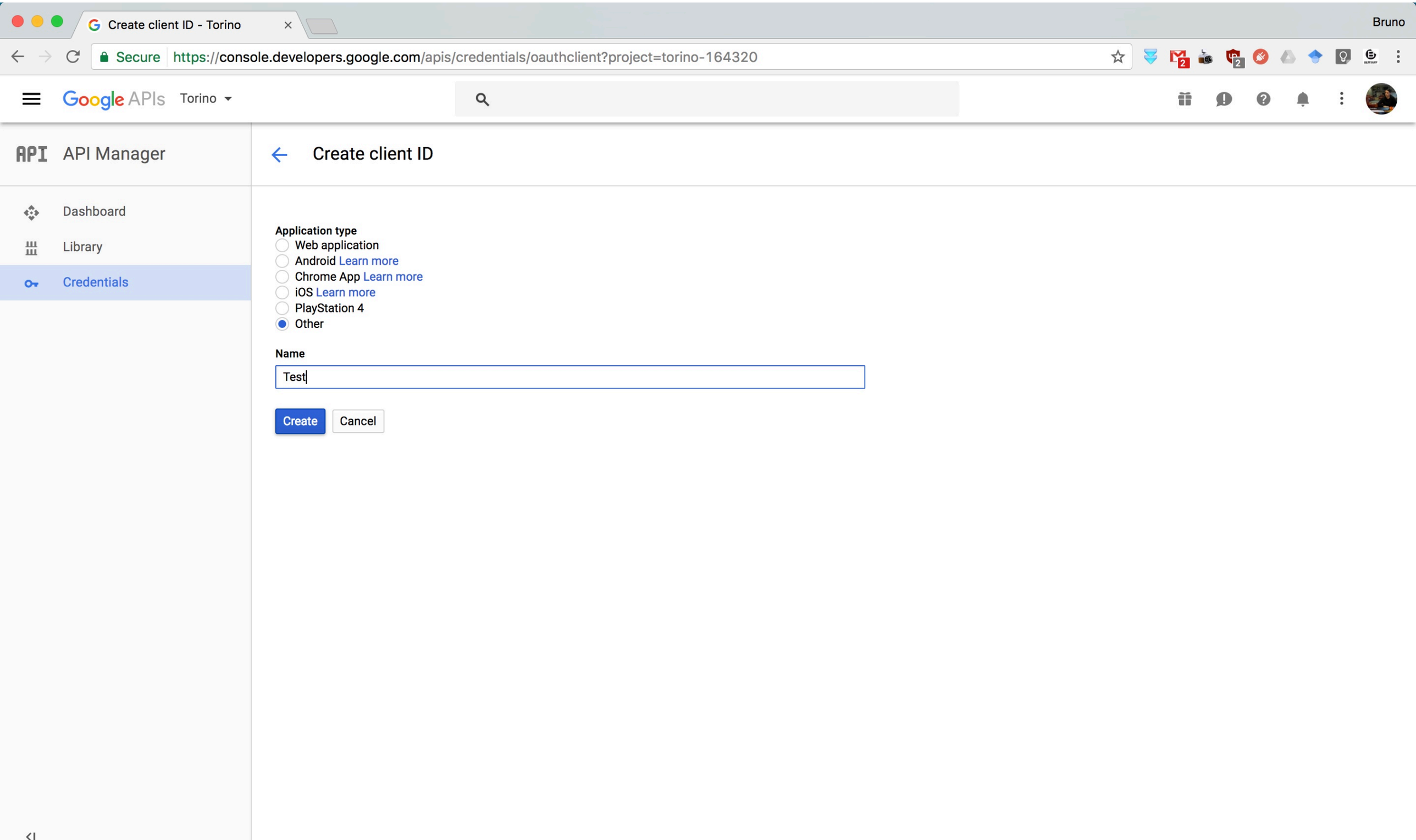


# Google Credentials

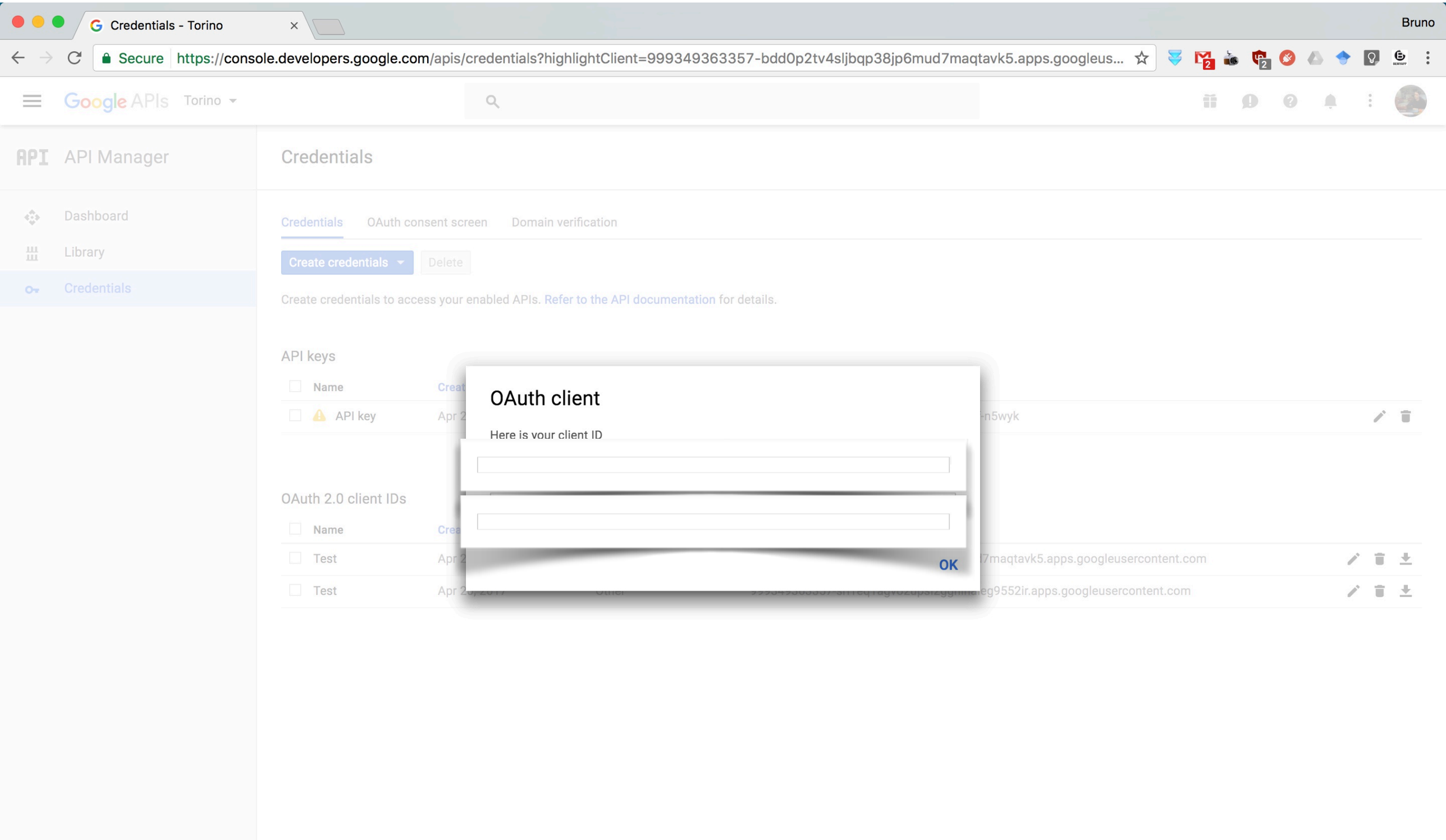



We'll need both an API Key and the OAuth client ID.

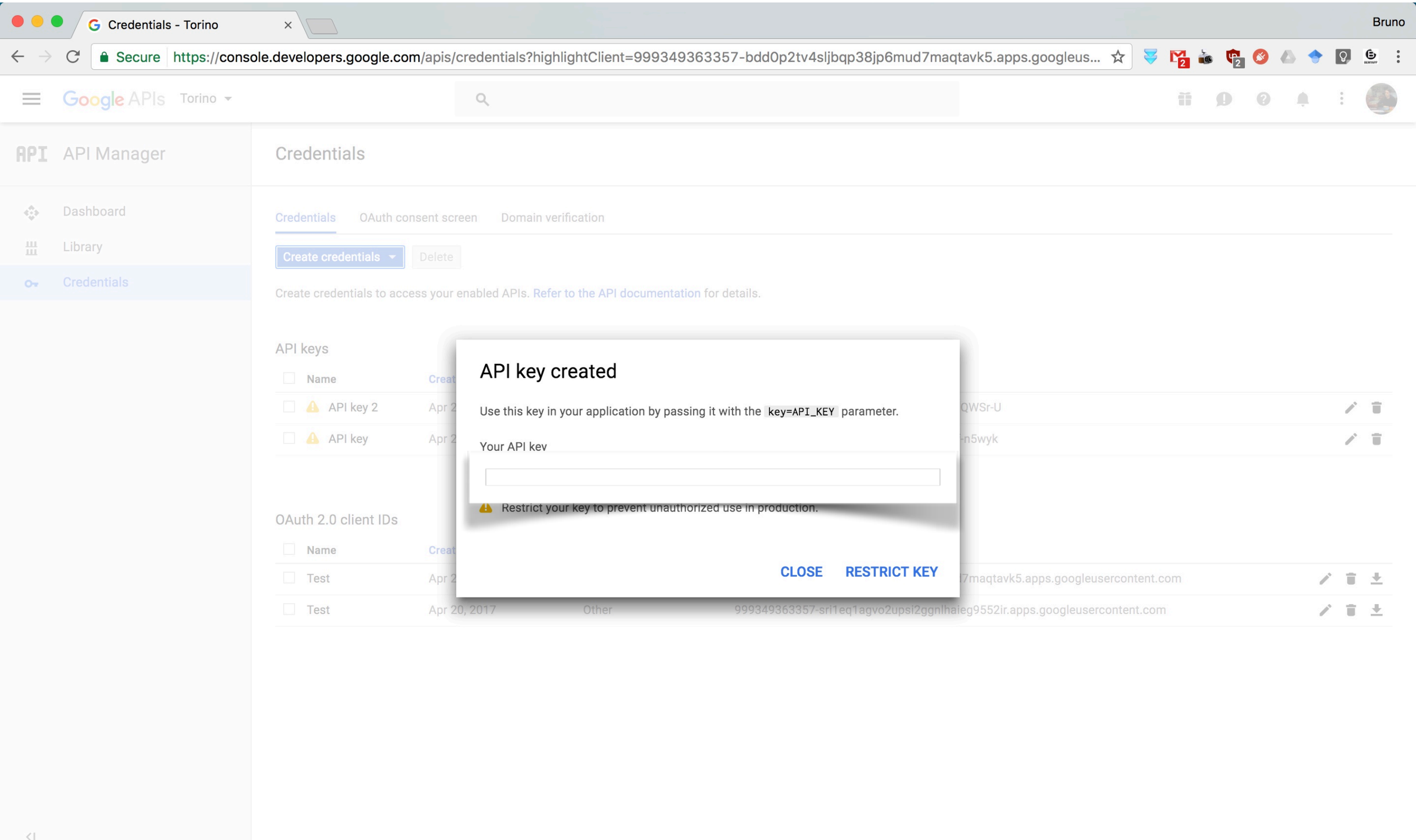
# Google Credentials



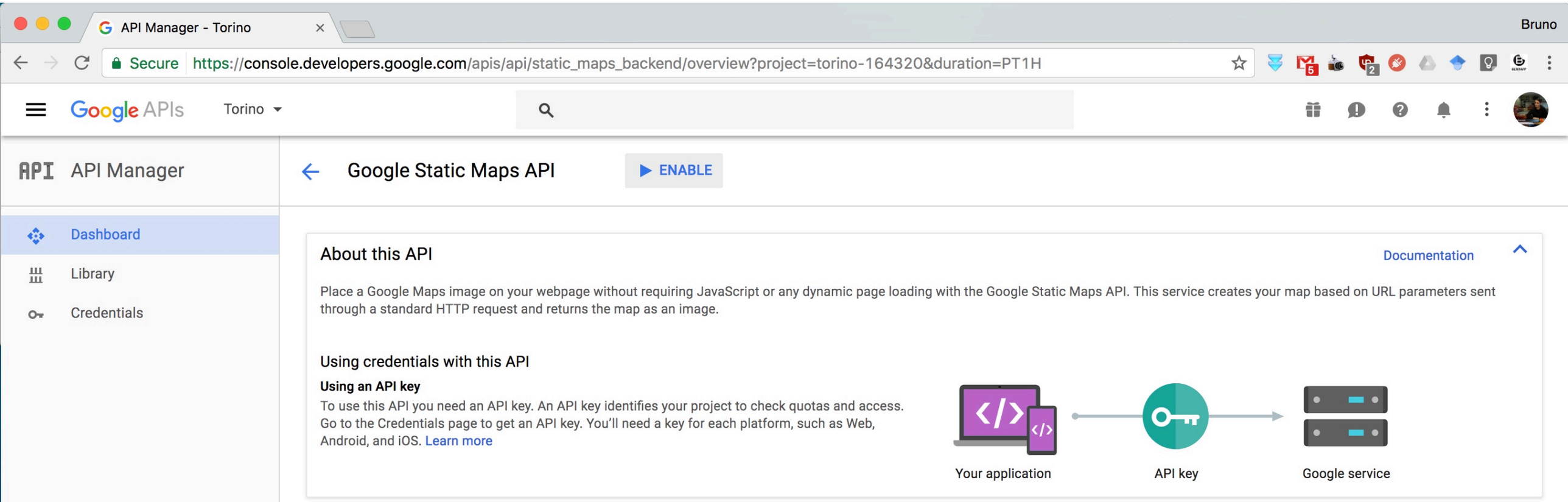
# Google Credentials



# Google Credentials

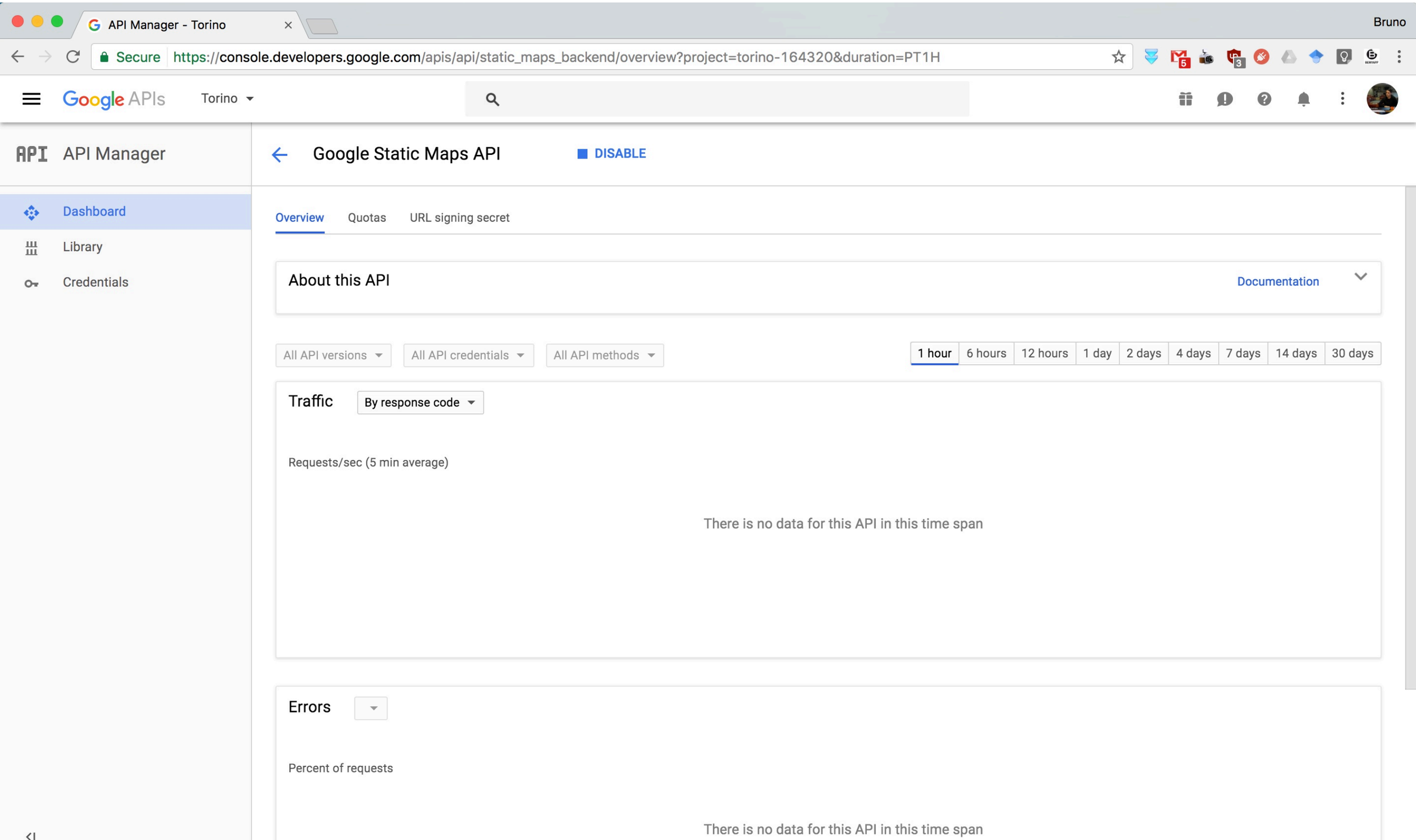


# API Authorization




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']}]
```

@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']}
```

# Challenge - Google Reverse Geocoding

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

- Reverse Geocode the coordinates:

40.6413111,-73.77813909999999

- and print the “types” and “formated_address” of each of the results

# Challenge - Google Reverse Geocoding

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

- Reverse Geocode the coordinates:

40.6413111,-73.77813909999999

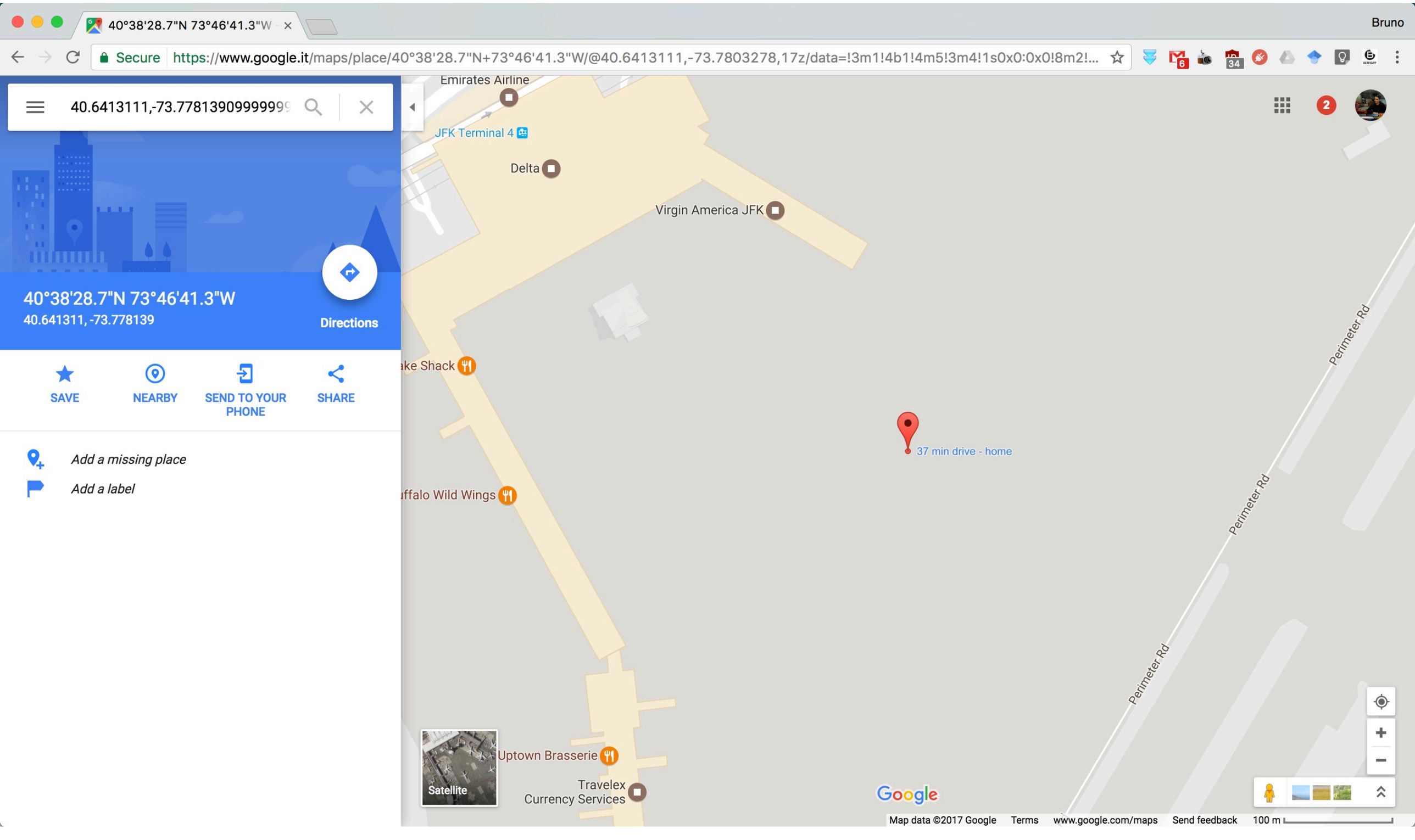
- and print the “types” and “formated_address” of each of the results

```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



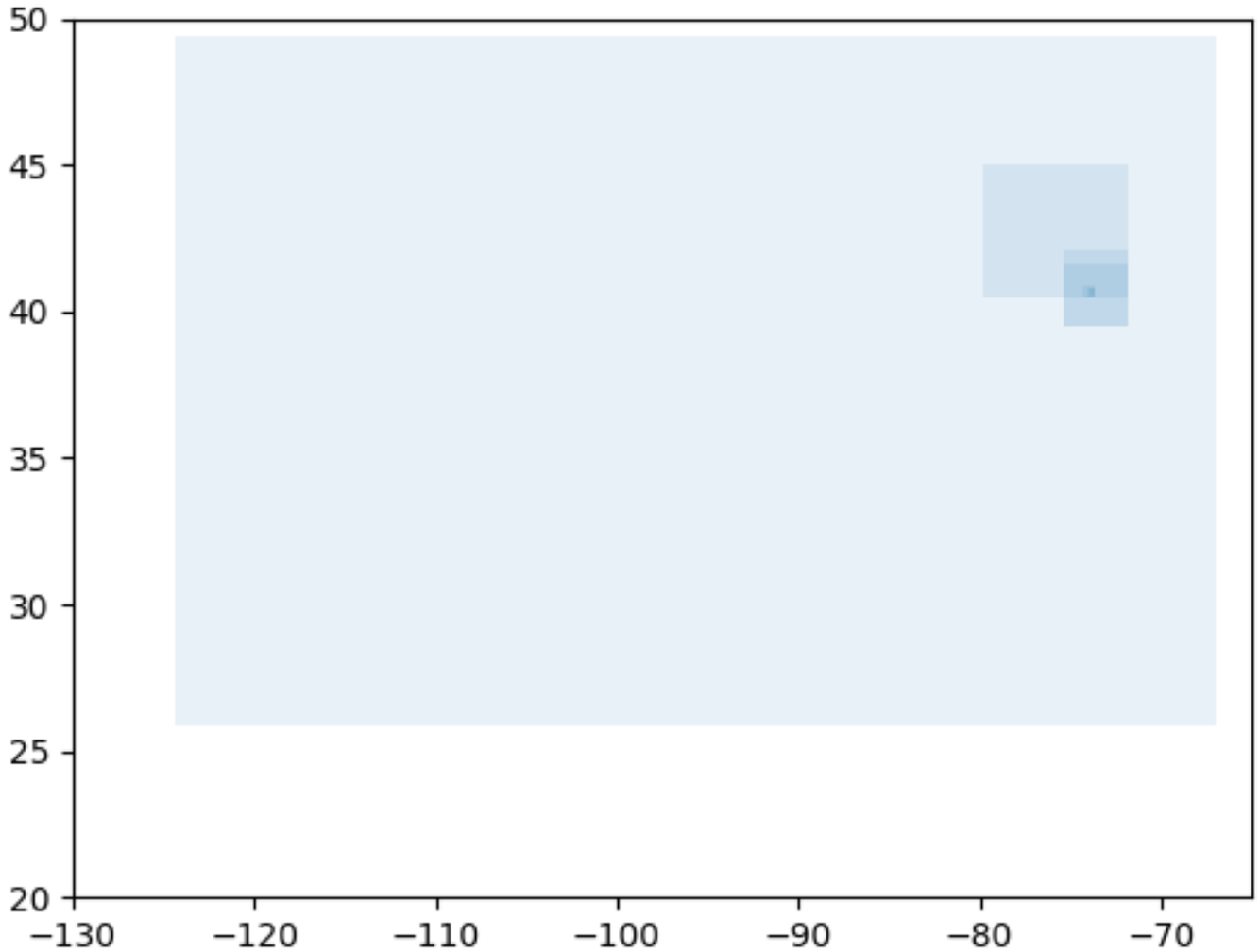
# Google Reverse Geocoding

```
import json
import matplotlib.pyplot as plt
import matplotlib.patches as patches

reverse_result = json.load(open('reverse.json'))

plt.figure()
plt.xlim(-130,-65)
plt.ylim(20,50)
currentAxis = plt.gca()


for result in reverse_result:
    viewport = result["geometry"]["viewport"]
    xy = (viewport['southwest']['lng'], viewport['southwest']['lat'])
    width = viewport['northeast']['lng']-viewport['southwest']['lng']
    height = viewport['northeast']['lat']-viewport['southwest']['lat']

    currentAxis.add_patch(patches.Rectangle(xy, width, height, alpha=.1))

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



plot_rectangles.py

# Google Reverse Geocoding






plot_rectangles.py

# Google Places

https://developers.google.com/places/

- Provides information about specific venues and locations
- .place(place_id) - Information about a specific place_id
- .places(query, location=None, radius=None, page_token=None) - query google maps for places within a certain radius of a given location
    - location - lat/long coordinates
    - radius - distance in meters
    - page_token - the token to paginate through the results
- Other options are also available to define price range, result language, type of result, etc…

```
['html_attributions',
 'results',
 'status',
 'next_page_token']
```

# Google Places

https://developers.google.com/places/

- Provides information about specific venues and locations
- .place(place_id) - Information about a specific place_id
- .places(query, location=None, radius=None, page_token=None) - query google maps for places within a certain radius of a given location
    - location - lat/long coordinates
    - radius - distance in meters
    - page_token - the token to paginate through the results
- Other options are also available to define price range, result language, type of result, etc…

```
['html_attributions',
 'results',
 'status',
 'next_page_token']
```

# Google Places

https://developers.google.com/places/

```
['types',
 'id',
 'place_id',
 'address_components',
 'international_phone_number',
 'name',
 'reviews',
 'adr_address',
'vicinity',
'url',
'utc_offset',
'website',
'geometry',
'reference',
'rating',
'scope',
'formatted_address',
'photos',
'icon',
'formatted_phone_number']
```

# Google Places

https://developers.google.com/places/

```
['types',                          ['airport', 'point_of_interest', 'establishment']
 'id',
 'place_id',
 'address_components',
 'international_phone_number',
 'name',                           'John F. Kennedy International Airport'
 'reviews',
 'adr_address',
 'vicinity',
 'url',
 'utc_offset',
 'website',
 'geometry',                      {'location': {'lat': 40.64131109999999, 'lng': -73.77813909999999},
 'reference',                      'viewport': {'northeast': {'lat': 40.67117114999999, 'lng': -73.76339675},
 'rating',                                      'southwest': {'lat': 40.63135774999999,
 'scope',                                                     'lng': -73.82236615000001}}}
 'formatted_address',
 'photos',
 'icon',
 'formatted_phone_number']
```

# Challenge - Google Places

https://developers.google.com/places/

- Search for “pizza” within a 10km radius of

45.090219, 7.659144

- and print the formatted address of the second page of results.

# Challenge - Google Places

https://developers.google.com/places/

- Search for “pizza” within a 10km radius of

45.090219, 7.659144

- and print the formatted address of the second page of results.

```python
import googlemaps
from google_accounts import accounts
import time

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

geocode_result = gmaps.places("pizza", (45.090219, 7.659144), 10000)
result = geocode_result["results"][0]
print(result["formatted_address"])

time.sleep(1)
page_token = geocode_result["next_page_token"]
geocode_result = gmaps.places(None, page_token=page_token)
result = geocode_result["results"][0]

print(result["formatted_address"])
```

google_place_search.py

# Challenge - Google Places

https://developers.google.com/places/

```
{'formatted_address': 'Corso Peschiera, 312/A, 10139 Torino, Italy',
 'geometry': {'location': {'lat': 45.07176469999999, 'lng': 7.6317175},
              'viewport': {'northeast': {'lat': 45.07308433029149,
                                         'lng': 7.633044980291503},
                           'southwest': {'lat': 45.0703863697085,
                                         'lng': 7.630347019708498}}},
 'icon': 'https://maps.gstatic.com/mapfiles/place_api/icons/generic_business-71.png',
 'id': '644f3c75d97ae5ec66ffca7a1d2a34a5a98abad4',
 'name': 'Pizza Loca',
 'opening_hours': {'open_now': True, 'weekday_text': []},
 'place_id': 'ChIJnZx7tJJsiEcR4bCZfBz3VSs',
 'rating': 5,
 'reference': 'CmRRAAAAFdoEAMs_mvHc7wP6-2pJ_Qzs9Yw4u7aMi51cyk05rfITav2I-
r6gK_pDUAsfjxSJAf0YvIL0xaPZ7R_1XiFLJNyNiQ2TArhZMumL7J3B55CroVEQdNnbcvHwsJmgT53DEhDkbpkUbiFn4Ntnj3G
3K_tgGhQNSiQrCbG8gAM1ZQii-zshdQgmbw',
 'types': ['meal_delivery',
           'meal_takeaway',
           'restaurant',
           'food',
           'point_of_interest',
           'establishment']}
```

*@bgoncalves*

# Google Directions

https://developers.google.com/maps/documentation/directions/

- Provides step by step directions from point A to point B.
- Directions are always for the shortest path
- .directions((start_lat, start_lon), (end_lat, end_lon)) - Directions between two sets of GPS coordinates
- .directions("start_address", "end_address") - Geolocate each address and provide directions from start_address to end_address.
- Each step of the journey comes with start_location and end_location GPS coordinates as well as distance, duration and html_instructions.

```
{'distance': {'text': '0.5 km', 'value': 464},
 'duration': {'text': '2 mins', 'value': 98},
 'end_location': {'lat': 45.5014209, 'lng': 9.1882349},
 'html_instructions': 'Continue onto <b>Via Valassina</b>',
 'polyline': {'points':
'{eutGagaw@YGSG_AWq@Sc@MkFaBeDeAgDiAEA[M'},
 'start_location': {'lat': 45.4974178, 'lng': 9.1865724},
 'travel mode': 'DRIVING'}
```

# Challenge - Google Directions

https://developers.google.com/maps/documentation/directions/

- Obtain the driving directions from coordinates:

45.067450, 7.685880

- to coordinates:

45.485895, 9.204283

- and print out all the html_instructions.

# Challenge - Google Directions

https://developers.google.com/maps/documentation/directions/

- Obtain the driving directions from coordinates:

45.067450, 7.685880

- to coordina
- and print ou

```python
import googlemaps
from google_accounts import accounts

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

directions = gmaps.directions((45.067450, 7.685880), (45.485895, 9.204283))

steps = directions[0]["legs"][0]["steps"]

for step in steps:
    print(step["html_instructions"])
```

google_directions.py

# Google Directions

https://developers.google.com/maps/documentation/directions/

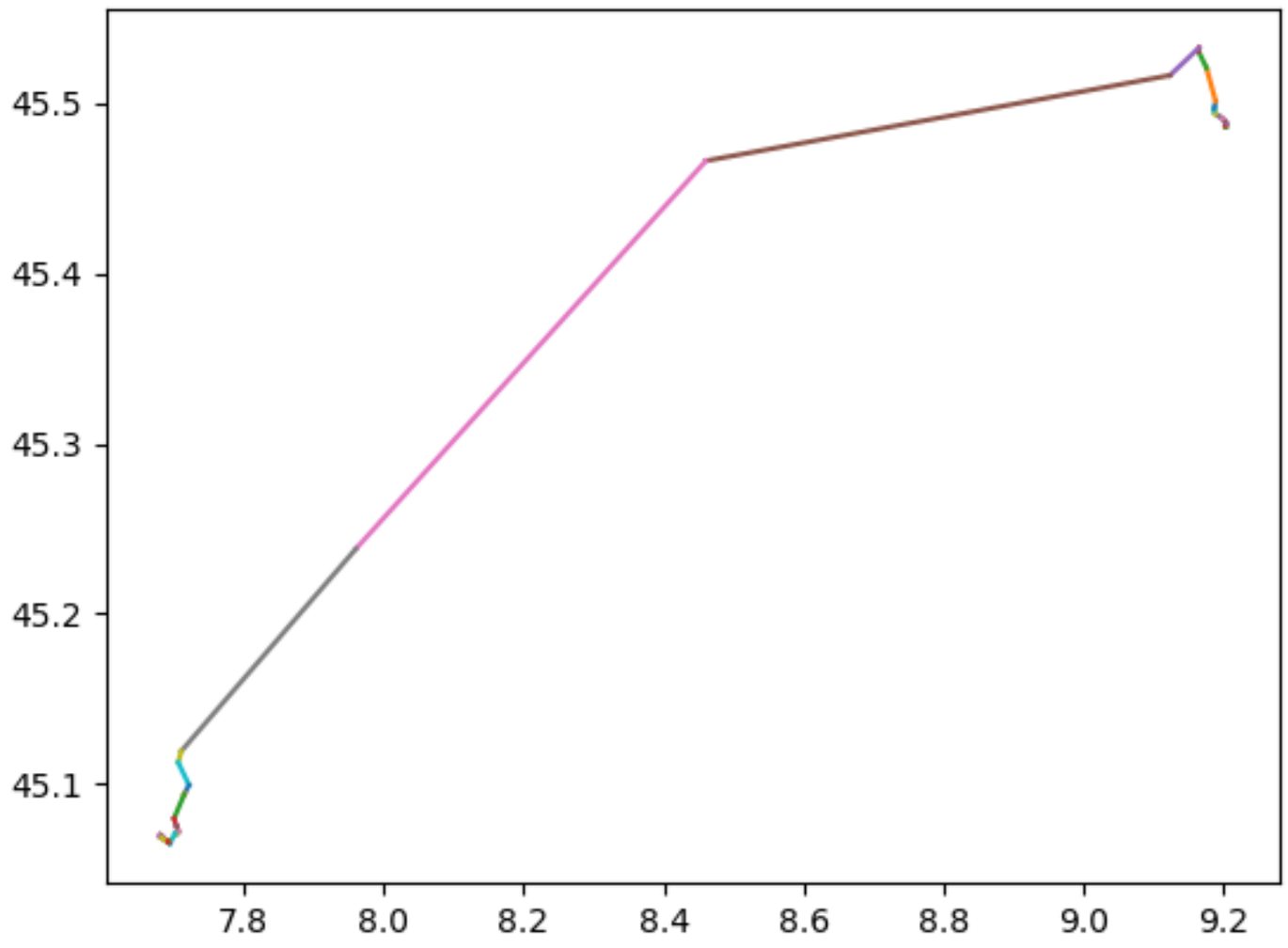
```python
import googlemaps
from google_accounts import accounts
import matplotlib.pyplot as plt

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

directions = gmaps.directions((45.485895, 9.204283),(45.067450, 7.685880))

steps = directions[0]["legs"][0]["steps"]

for step in steps:
    start = (step["start_location"]["lng"], step["start_location"]["lat"])
    stop = (step["end_location"]["lng"], step["end_location"]["lat"])

    plt.plot([start[0], stop[0]], [start[1], stop[1]])

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



plot_google_directions.py

# Google Directions

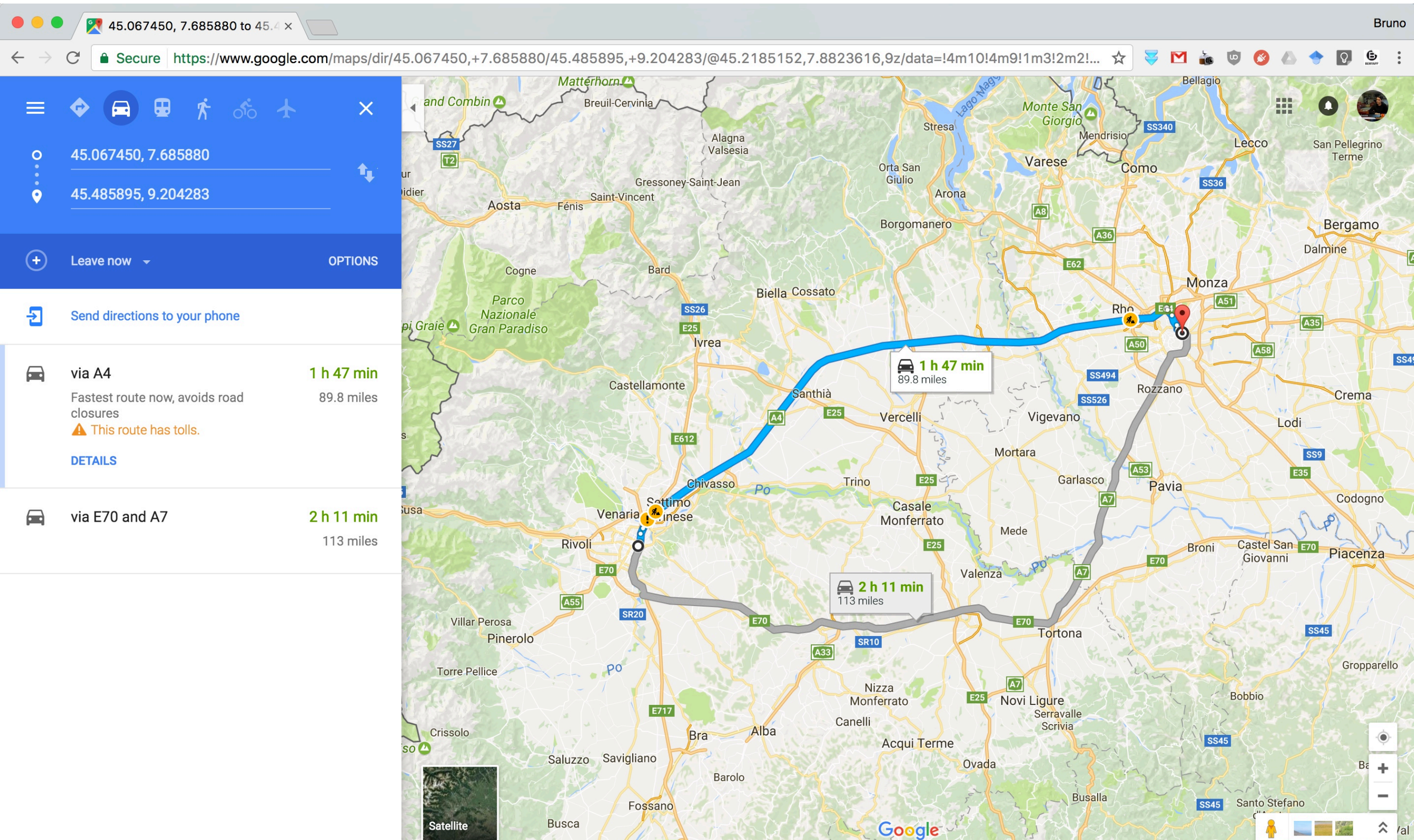
https://developers.google.com/maps/documentation/directions/




*@bgoncalves*

# Google Directions

https://developers.google.com/maps/documentation/directions/



# Google Distance Matrix

https://developers.google.com/maps/documentation/distance-matrix/

- Returns the travel time and distance for all pairs of origins and destinations
- Results are returned in rows, each row containing one origin paired with each destination.
- .distance_matrix(origins, destinations) - Both origins and destinations can be listed as either lat/long pairs or addresses
- Options for **mode** of transportation, **departure** or **arrival** time, etc…

# Challenge - Google Distance Matrix

https://developers.google.com/maps/documentation/distance-matrix/

- Obtain the distance and time matrix from these origins:

(45.485895, 9.204283), (41.912199, 12.499857)

- to these destinations:

(45.067450, 7.685880), (43.775475, 11.257102)

- And print out the resulting object

# Challenge - Google Distance Matrix

https://developers.google.com/maps/documentation/distance-matrix/

```python
import googlemaps
from google_accounts import accounts
from pprint import pprint

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

origins = [(45.485895, 9.204283), (41.912199, 12.499857)]
destinations = [(45.067450, 7.685880), (43.775475, 11.257102)]

matrix = gmaps.distance_matrix(origins, destinations)

pprint(matrix)
```



google_distance.py

# Google Distance Matrix

https://developers.google.com/maps/documentation/distance-matrix/


```
{'destination_addresses': ['Via Maria Vittoria, 12, 10123 Torino, Italy',
                           'Via Ricasoli, 26-30, 50122 Firenze, Italy'],
 'origin_addresses': ['Piazza Quattro Novembre, 581, 20124 Milano, Italy',
                      'Via Brescia, 23, 00198 Roma, Italy'],
 'rows': [{'elements': [{'distance': {'text': '147 km', 'value': 147321},
                         'duration': {'text': '1 hour 54 mins', 'value': 6868},
                         'status': 'OK'},
                        {'distance': {'text': '307 km', 'value': 306952},
                         'duration': {'text': '3 hours 23 mins','value': 12167},
                         'status': 'OK'}]},
          {'elements': [{'distance': {'text': '690 km', 'value': 689739},
                         'duration': {'text': '6 hours 49 mins','value': 24521},
                         'status': 'OK'},
                        {'distance': {'text': '276 km', 'value': 275684},
                         'duration': {'text': '3 hours 6 mins', 'value': 11163},
                         'status': 'OK'}]}],
 'status': 'OK'}
```


*@bgoncalves*

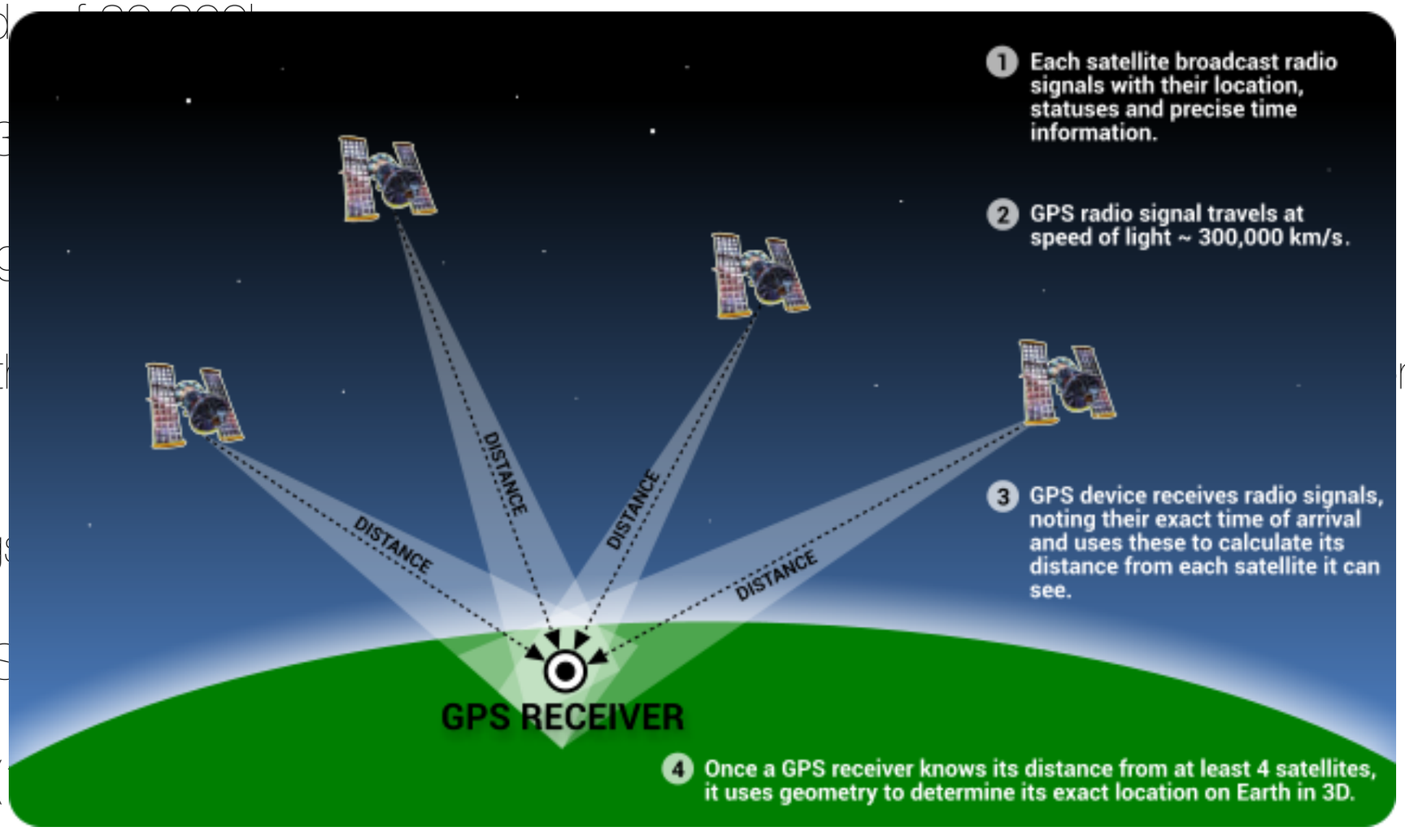
# Geographical Information Systems

# Global Positioning System (GPS)

- 32 satellites
- Orbital altitude of 20,200km
- Traveling at 3.9km/s
- Broadcasting their timestamped position
- Position on the ground is determined by GPS devices by triangulating information from 4 or more satellites.
- GPS belongs to the US government by tother similar systems exist
    - GLONASS - Russia
    - BeiDou (1/2) - China
    - Galileo - Europe

# 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)







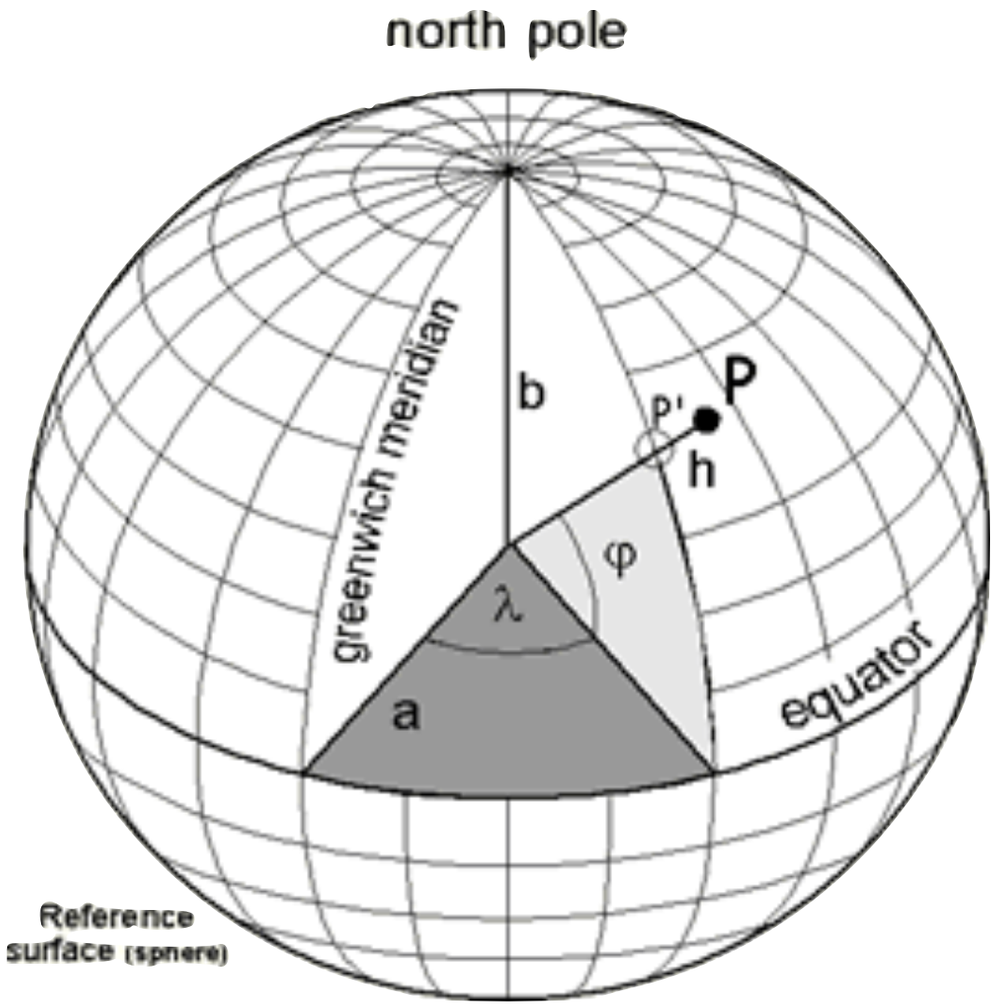


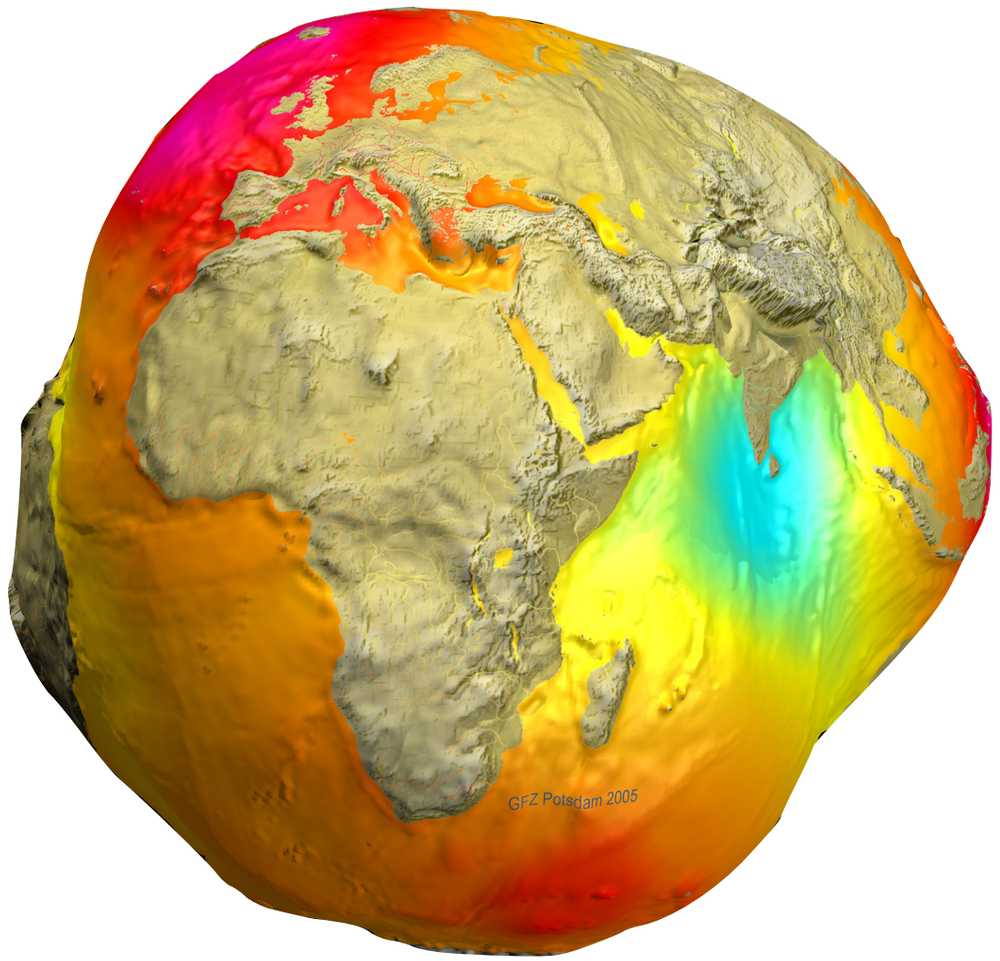




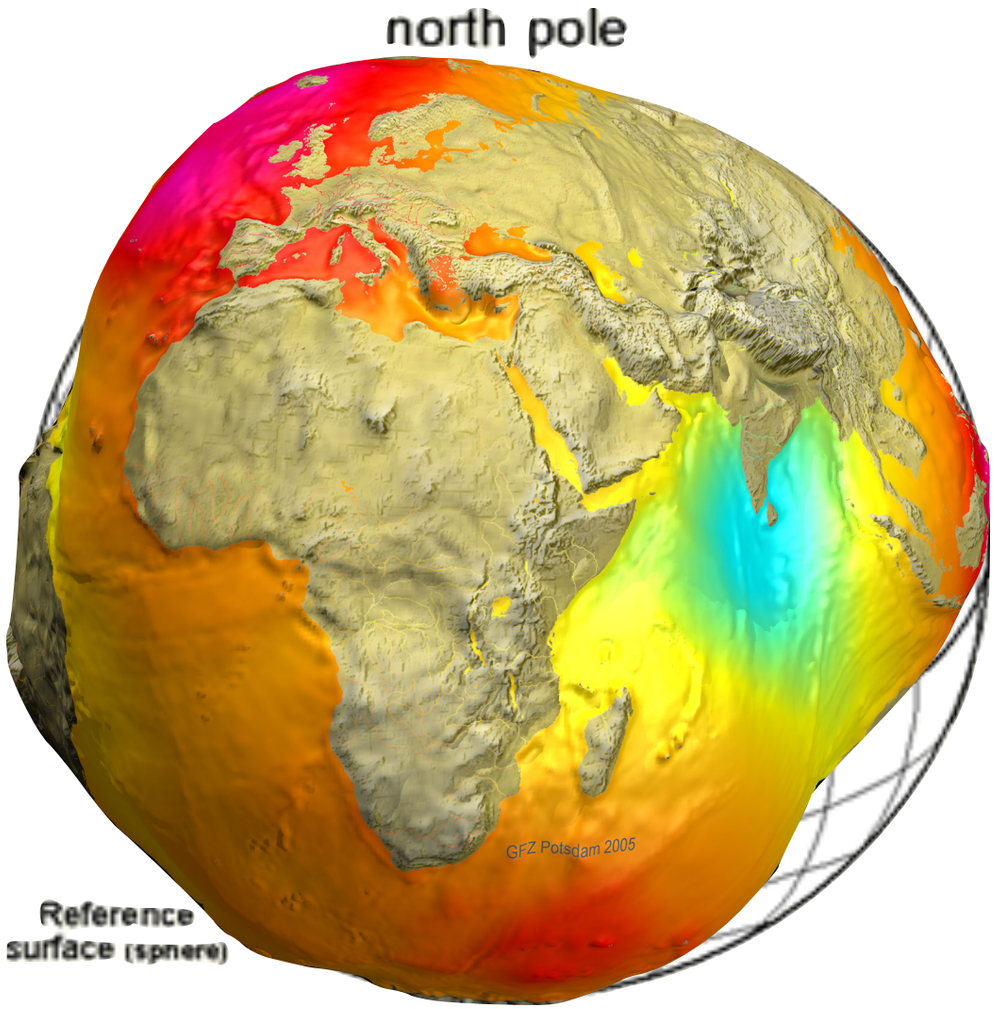
# Global Positioning System (GPS)

# Global Positioning System (GPS)

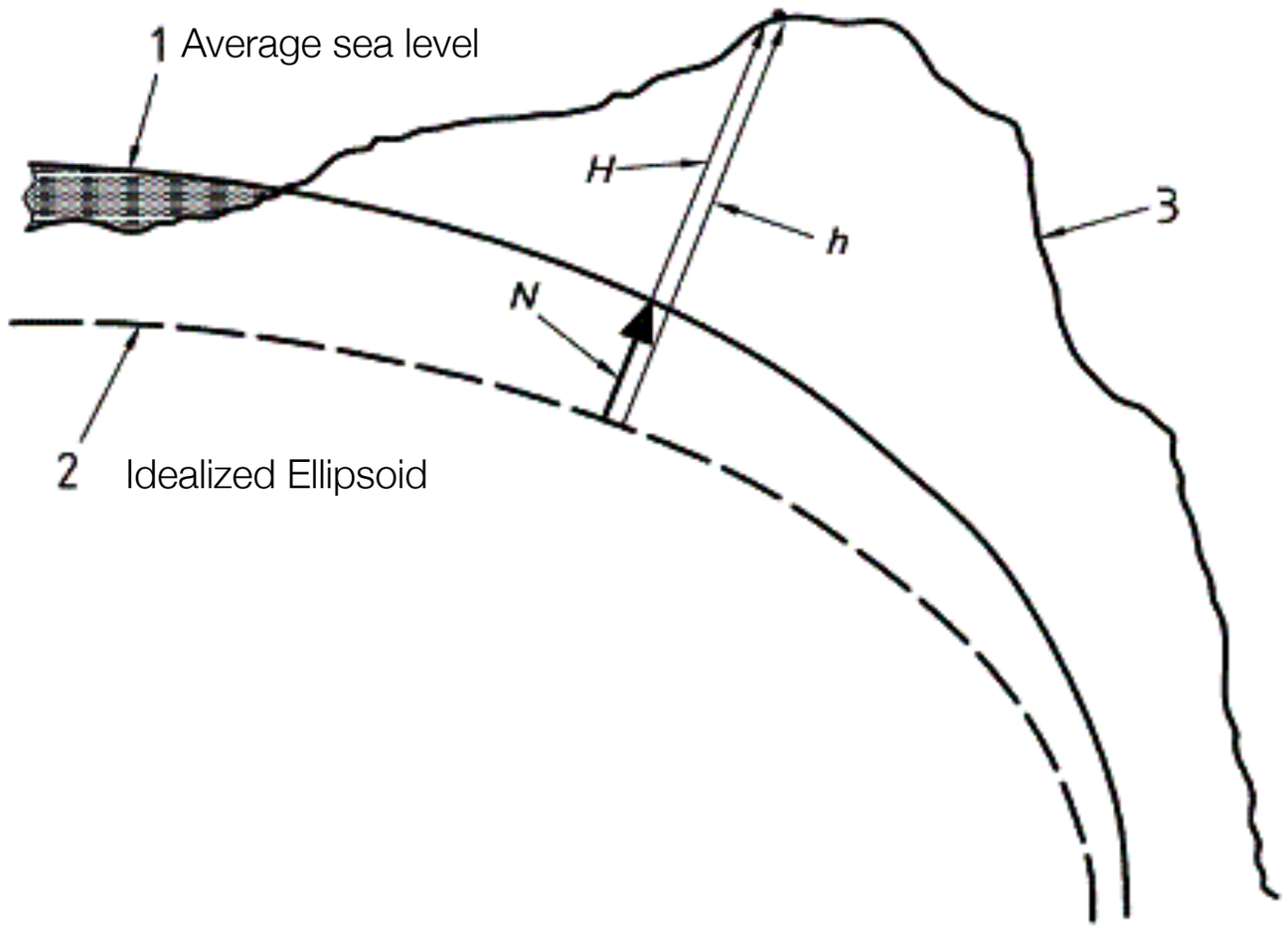


# Global Positioning System (GPS)



# 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 | $\omega$ | $7292115 \times 10^{-11}$ 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 $3.9860050 \times 10^{14}$ m$^3$/sec$^2$ as specified in the references below.

# GIS Data Systems

## Vector

- Data types:
    - Points
    - Lines
    - Polygons

## Raster

- Data types:
    - Cells
    - Pixels
    - Elements

# 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 Data Systems

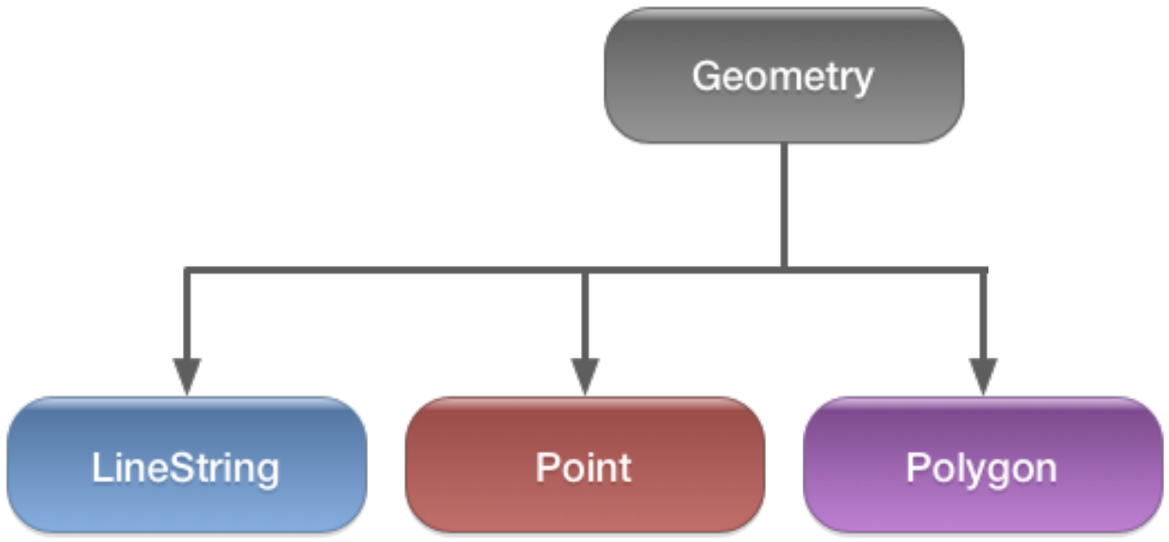
## Vector

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

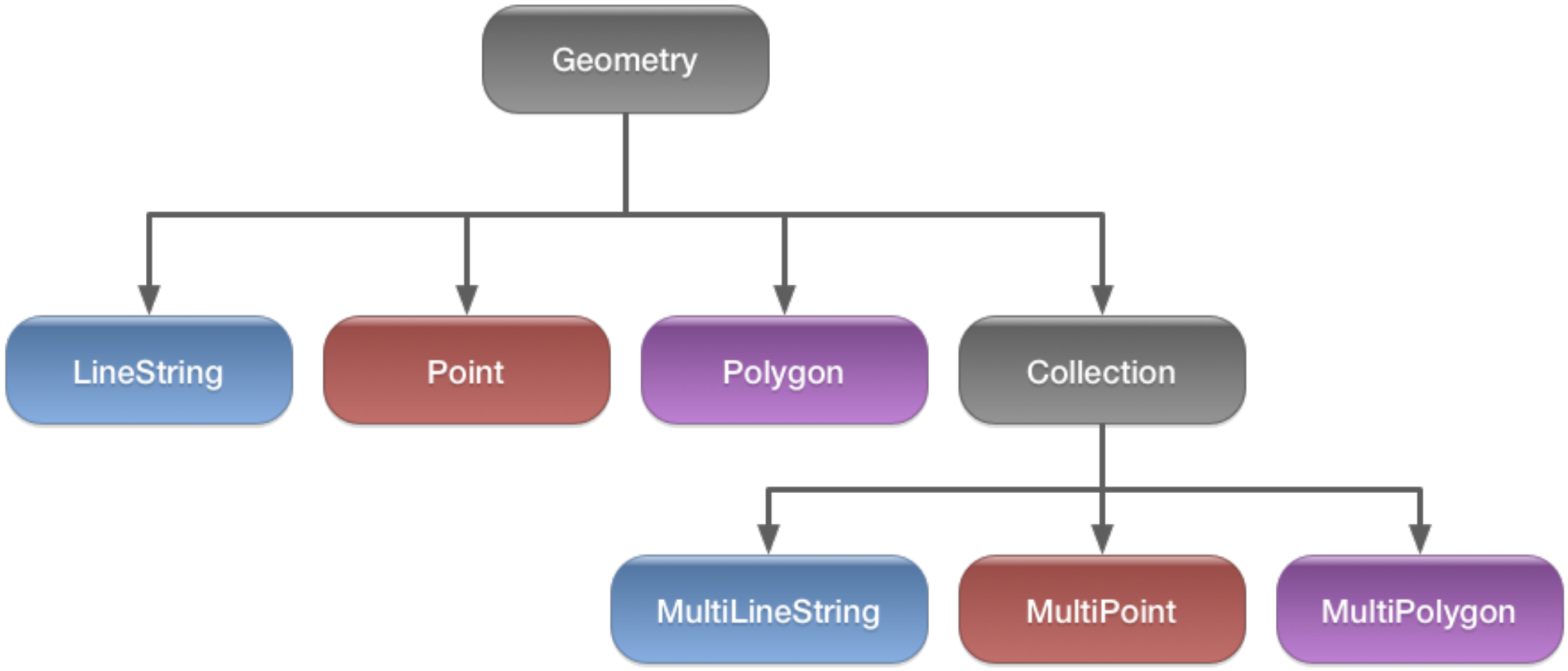
## Raster

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

# GIS Vector Data types

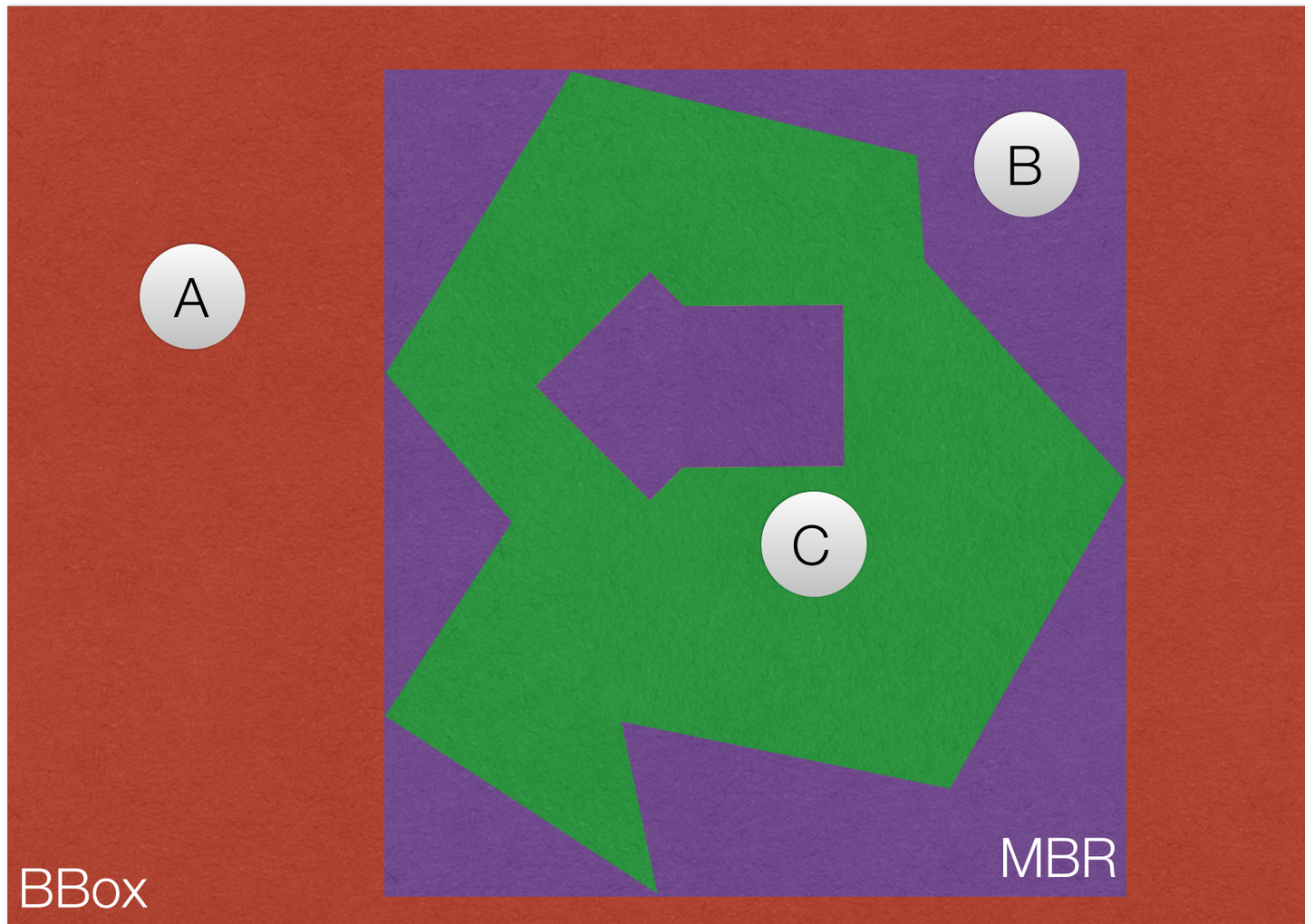


# GIS Vector Data types



# 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

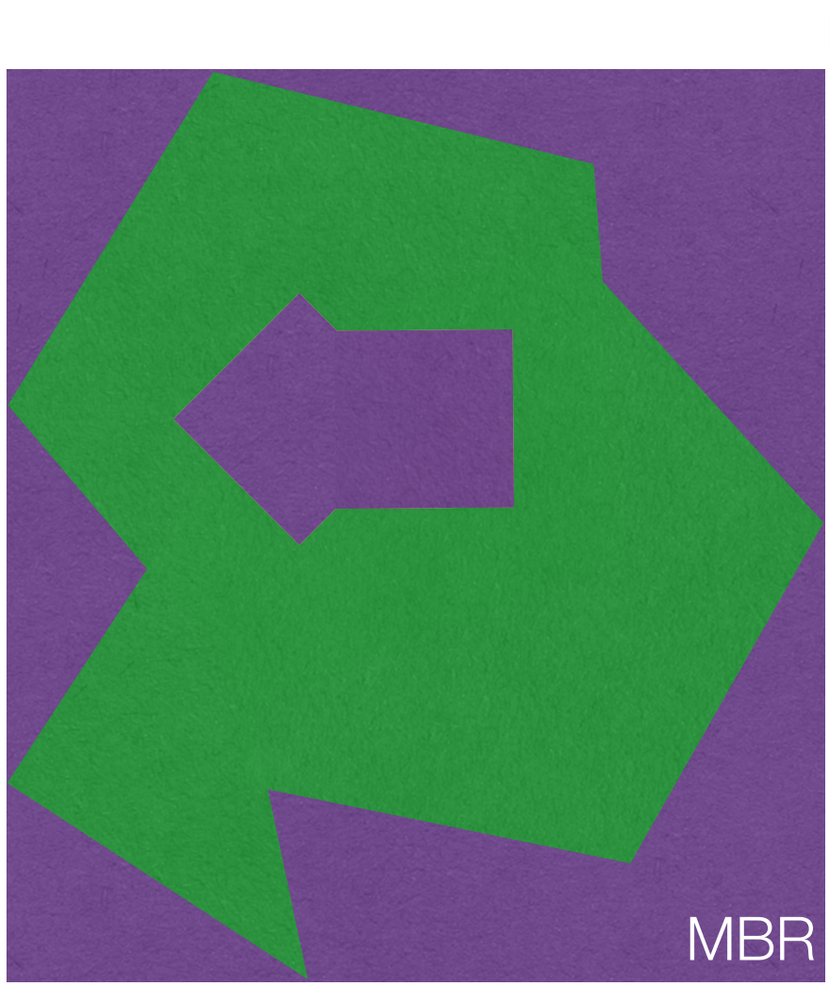


# Get MBR/BBox

- 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

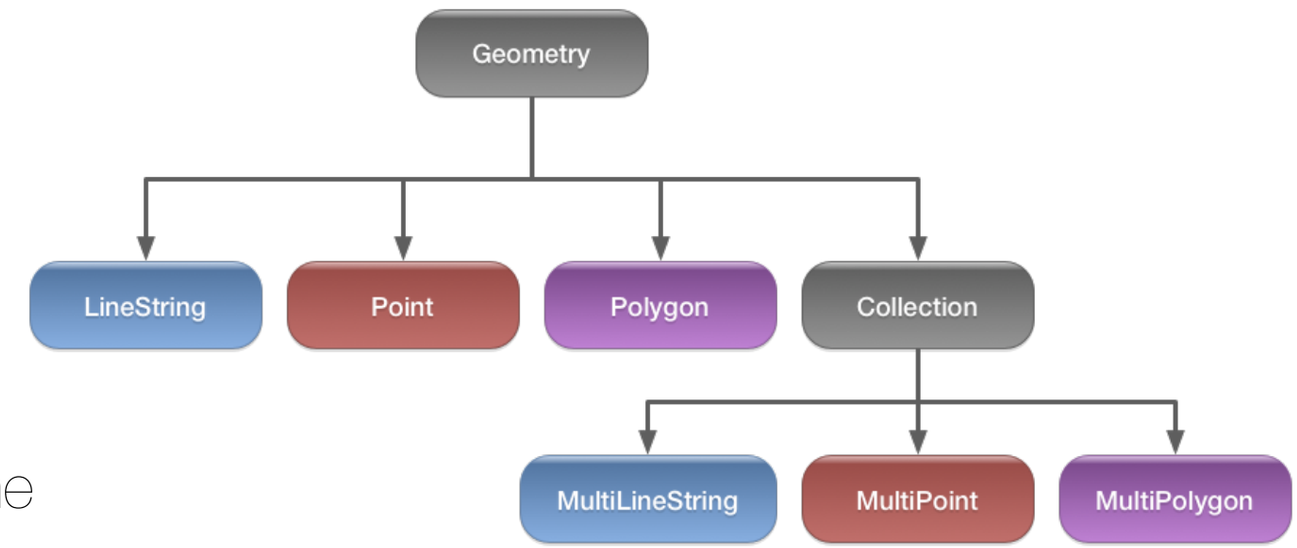
    - Geometry - a region of space

    - 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



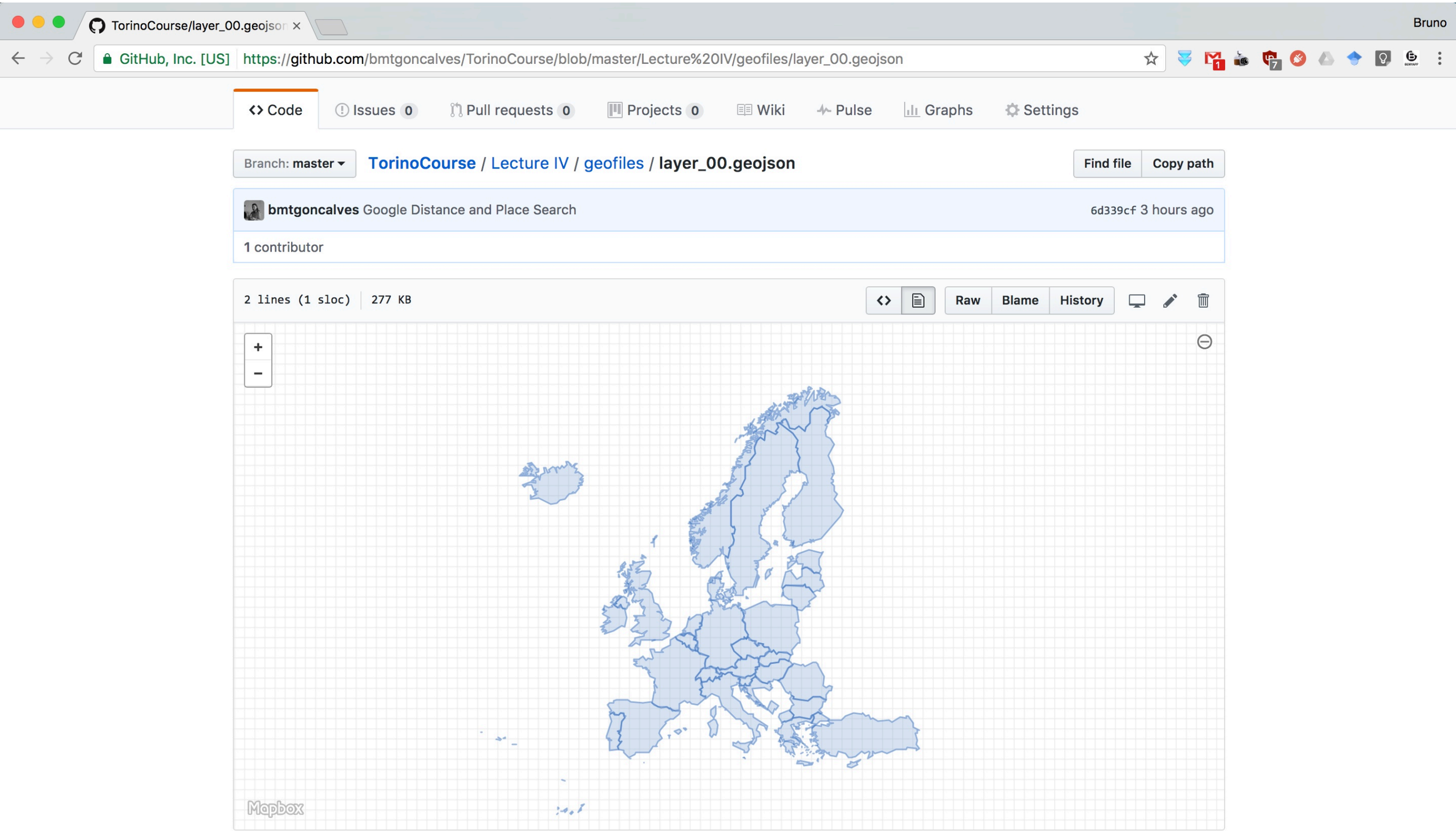
# GeoJSON

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

# GeoJSON

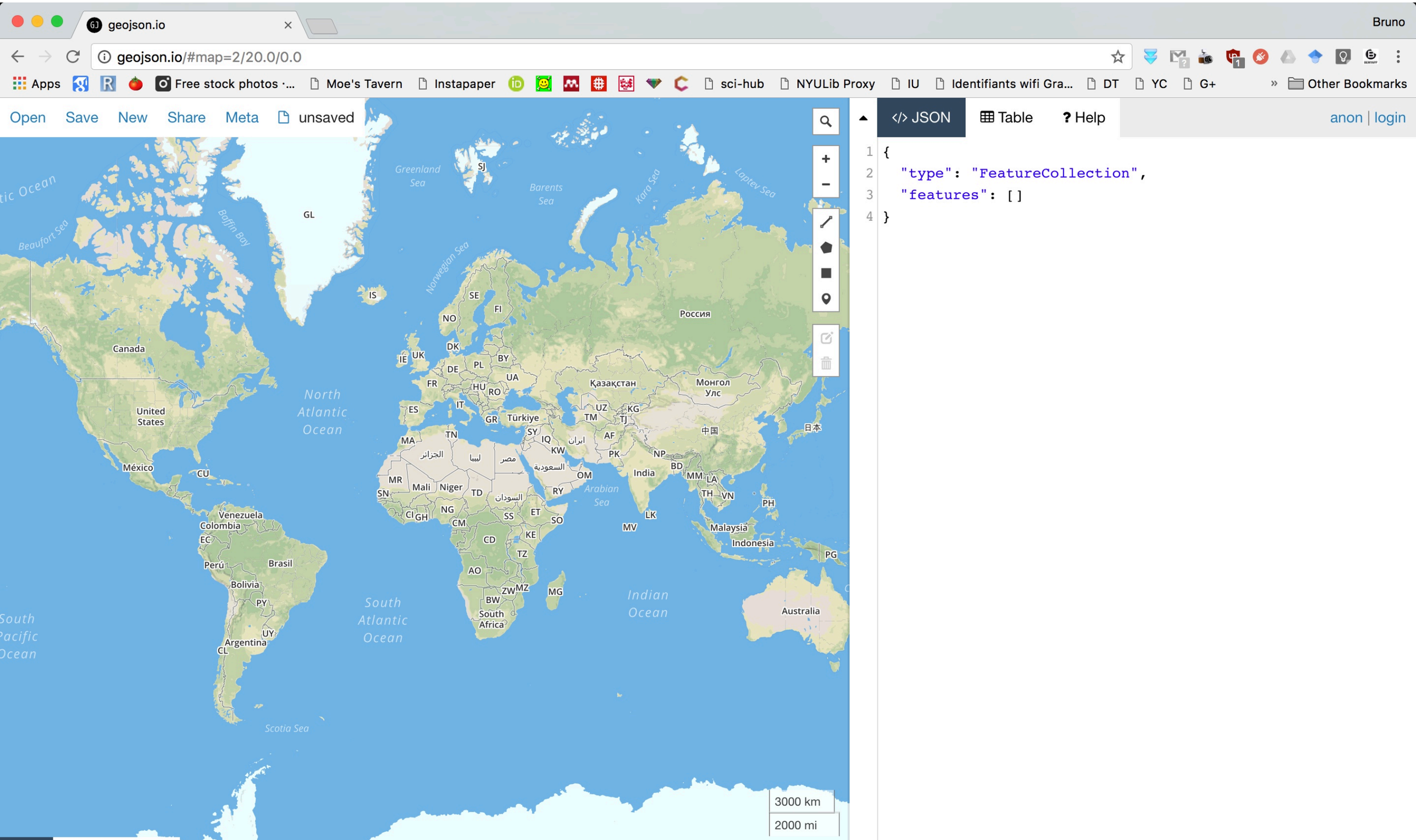
```
['type',
 'crs',
 'features']   {'geometry': {'coordinates': [[[[19.224069, 43.527541],
                                               [19.227058, 43.47903949800002],
                                               (…)
                                               [19.049223, 43.501789001000002],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

# GitHub support



# GeoJSON

geojson.io



# Challenge - GeoJSON

- Write a short function to calculate the Bounding Box (MBR) from a GeoJSON file like the one in the **geofiles/** folder

# Challenge - 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

# Challenge - 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:

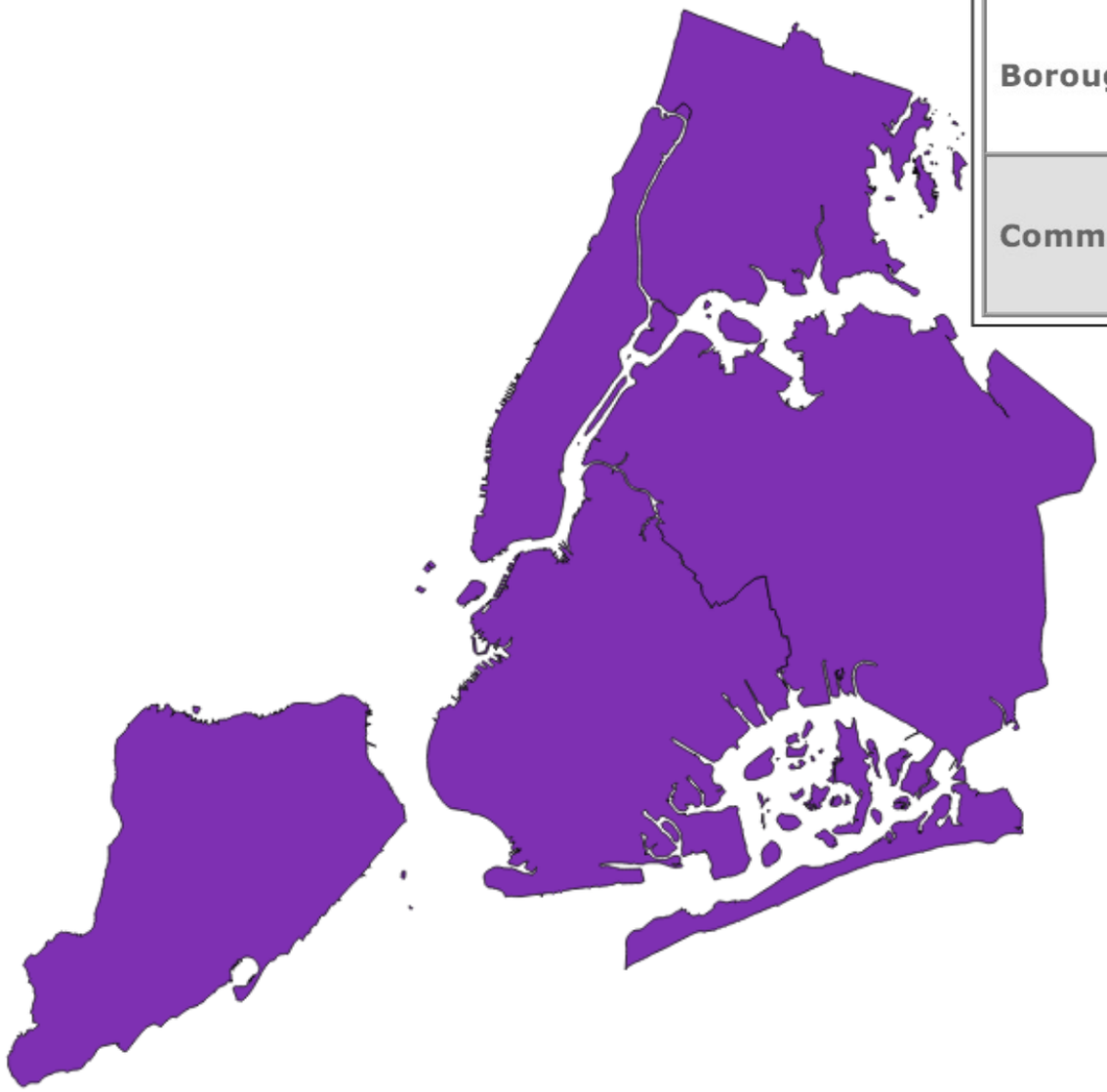
```
(py35) (master) bgoncalves@underdark:$ls -l
total 4856
-rw-r--r--@ 1 bgoncalves  staff      537 Apr 17 12:40 nybb.dbf
-rw-r--r--@ 1 bgoncalves  staff      562 Apr 17 12:40 nybb.prj
-rw-r--r--@ 1 bgoncalves  staff  1217376 Apr 17 12:40 nybb.shp
-rw-r--r--@ 1 bgoncalves  staff    12905 Apr 17 12:40 nybb.shp.xml
-rw-r--r--@ 1 bgoncalves  staff      140 Apr 17 12:40 nybb.shx
-rw-r--r--  1 bgoncalves  staff      536 Apr 17 12:40 nybb_wgs84.dbf
-rw-r--r--  1 bgoncalves  staff      143 Apr 17 12:40 nybb_wgs84.prj
-rw-r--r--  1 bgoncalves  staff      257 Apr 17 12:40 nybb_wgs84.qpj
-rw-r--r--  1 bgoncalves  staff  1217376 Apr 17 12:40 nybb_wgs84.shp
-rw-r--r--  1 bgoncalves  staff      140 Apr 17 12:40 nybb_wgs84.shx
(py35) (master) bgoncalves@underdark:$
```

- 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

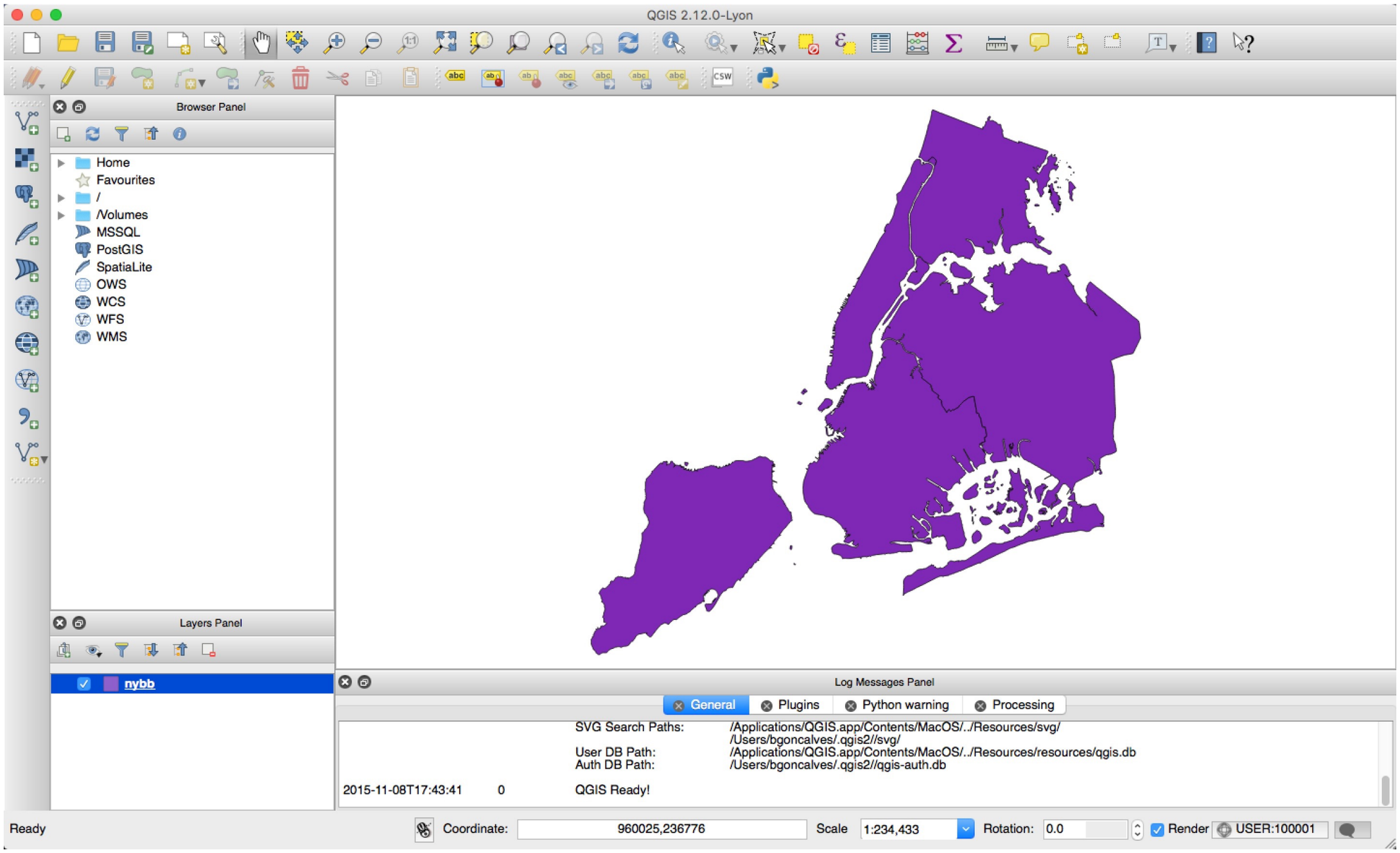
http://www.nyc.gov/html/dcp/html/bytes/districts_download_metadata.shtml#bcd

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

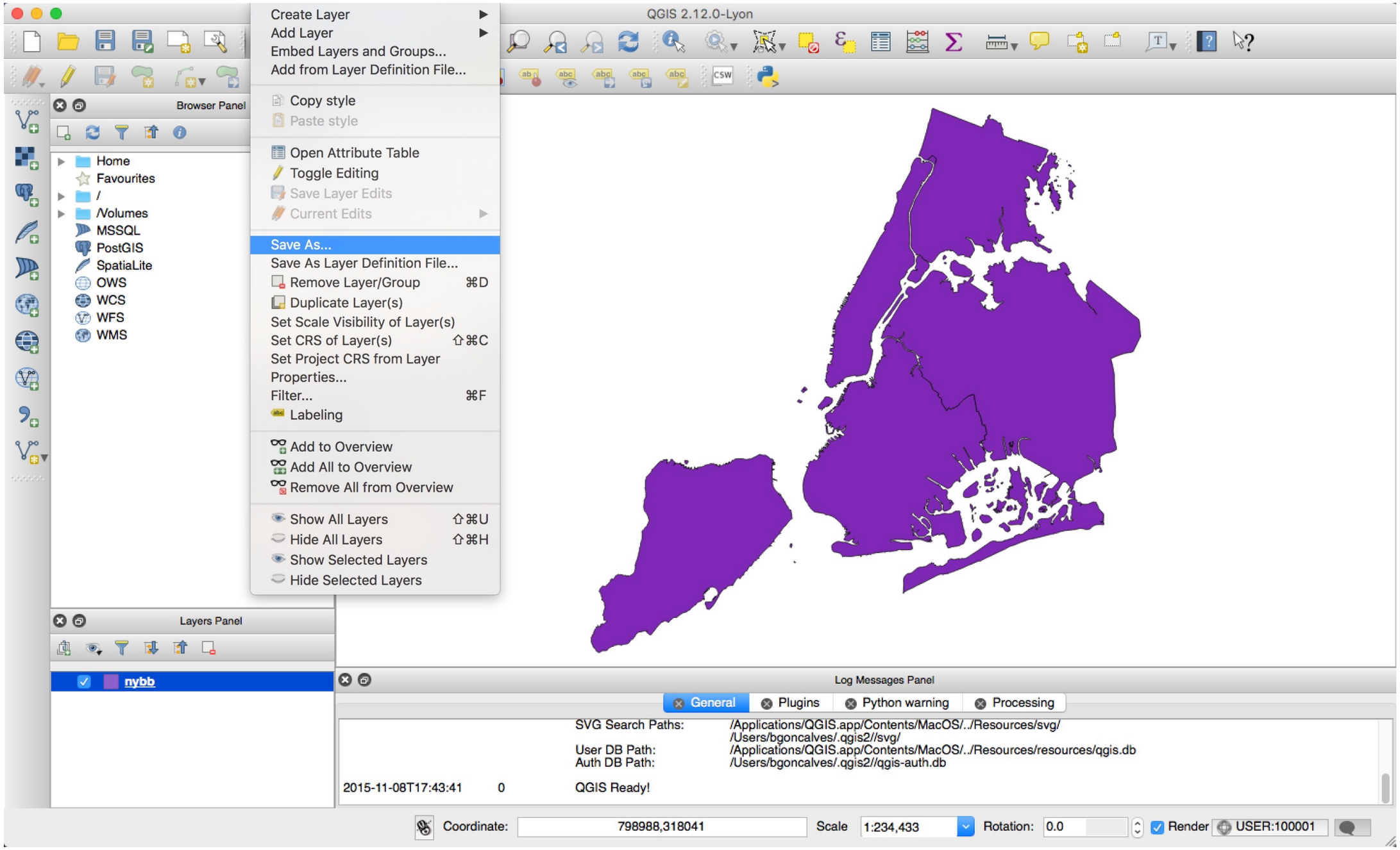


- Unfortunately it doesn't use the right reference system (WGS84), so we must convert it.

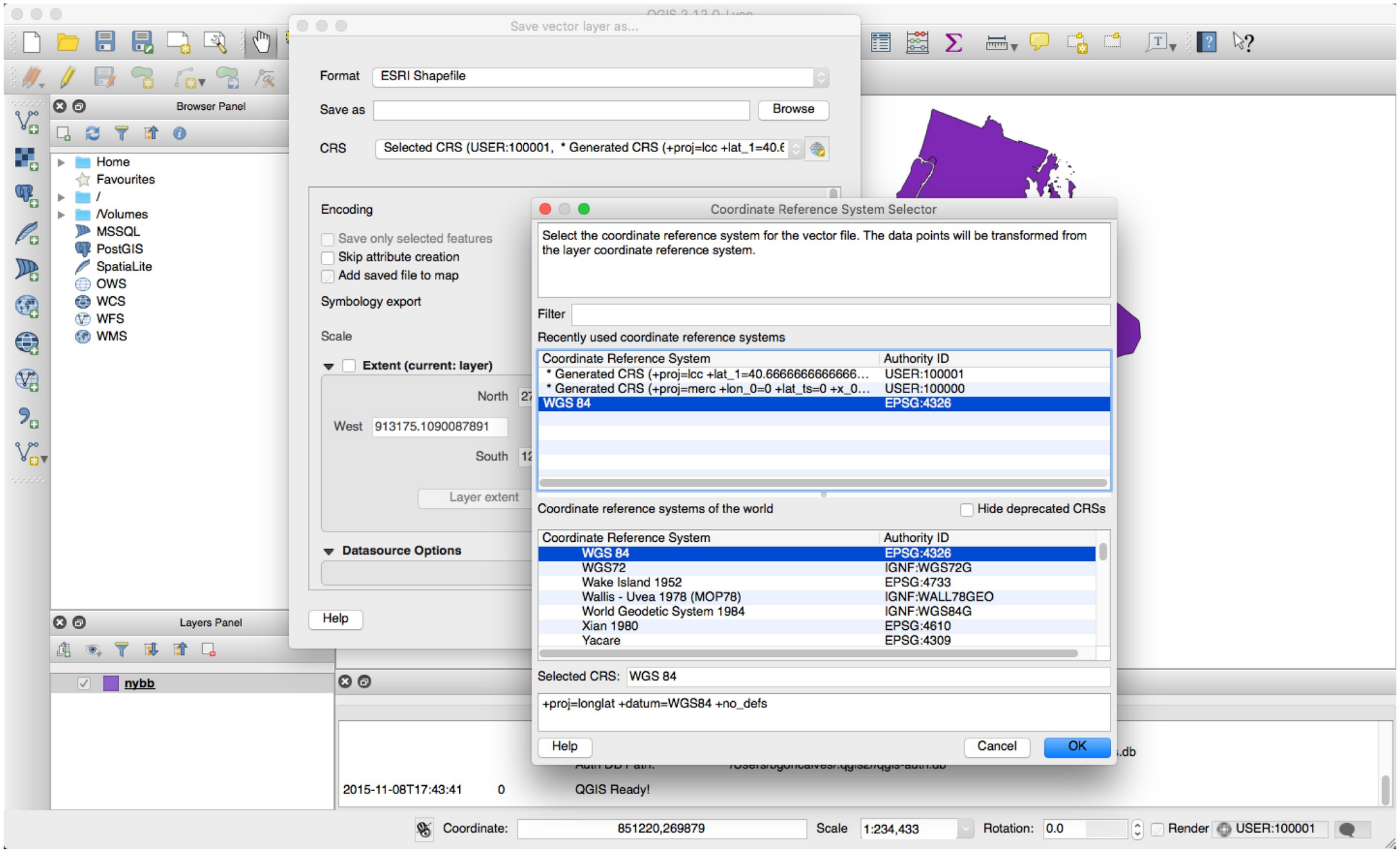
# Shapefiles


QGIS 2.12.0-Lyon
Browser Panel
Home
Favourites
/
/Volumes
MSSQL
PostGIS
SpatiaLite
OWS
WCS
WFS
WMS
Layers Panel
nybb
Log Messages Panel
General
Plugins
Python warning
Processing
SVG Search Paths: /Applications/QGIS.app/Contents/MacOS/../Resources/svg/
/Users/bgoncalves/.qgis2//svg/
User DB Path: /Applications/QGIS.app/Contents/MacOS/../Resources/resources/qgis.db
Auth DB Path: /Users/bgoncalves/.qgis2//qgis-auth.db
2015-11-08T17:43:41 0 QGIS Ready!
Ready
Coordinate: 960025,236776
Scale 1:234,433
Rotation: 0.0
Render
USER:100001

# Shapefiles


QGIS 2.12.0-Lyon
Browser Panel
Home
Favourites
/
/Volumes
MSSQL
PostGIS
SpatiaLite
OWS
WCS
WFS
WMS
Create Layer
Add Layer
Embed Layers and Groups...
Add from Layer Definition File...
Copy style
Paste style
Open Attribute Table
Toggle Editing
Save Layer Edits
Current Edits
Save As...
Save As Layer Definition File...
Remove Layer/Group ⌘D
Duplicate Layer(s)
Set Scale Visibility of Layer(s)
Set CRS of Layer(s) ⇧⌘C
Set Project CRS from Layer
Properties...
Filter... ⌘F
Labeling
Add to Overview
Add All to Overview
Remove All from Overview
Show All Layers ⇧⌘U
Hide All Layers ⇧⌘H
Show Selected Layers
Hide Selected Layers
Layers Panel
nybb
Log Messages Panel
General
Plugins
Python warning
Processing
SVG Search Paths: /Applications/QGIS.app/Contents/MacOS/../Resources/svg/
/Users/bgoncalves/.qgis2//svg/
User DB Path: /Applications/QGIS.app/Contents/MacOS/../Resources/resources/qgis.db
Auth DB Path: /Users/bgoncalves/.qgis2//qgis-auth.db
2015-11-08T17:43:41 0 QGIS Ready!
Coordinate: 798988,318041
Scale 1:234,433
Rotation: 0.0
Render
USER:100001

# Shapefiles


Save vector layer as...
Format
ESRI Shapefile
Save as
Browse
CRS
Selected CRS (USER:100001, * Generated CRS (+proj=lcc +lat_1=40.6
Encoding
Save only selected features
Skip attribute creation
Add saved file to map
Symbology export
Scale
Extent (current: layer)
North
West
913175.1090087891
South
Layer extent
Datasource Options
Help
Coordinate Reference System Selector
Select the coordinate reference system for the vector file. The data points will be transformed from the layer coordinate reference system.
Filter
Recently used coordinate reference systems
Coordinate Reference System
Authority ID
* Generated CRS (+proj=lcc +lat_1=40.6666666666666…
USER:100001
* Generated CRS (+proj=merc +lon_0=0 +lat_ts=0 +x_0…
USER:100000
WGS 84
EPSG:4326
Coordinate reference systems of the world
Hide deprecated CRSs
Coordinate Reference System
Authority ID
WGS 84
EPSG:4326
WGS72
IGNF:WGS72G
Wake Island 1952
EPSG:4733
Wallis - Uvea 1978 (MOP78)
IGNF:WALL78GEO
World Geodetic System 1984
IGNF:WGS84G
Xian 1980
EPSG:4610
Yacare
EPSG:4309
Selected CRS:
WGS 84
+proj=longlat +datum=WGS84 +no_defs
Help
Cancel
OK
Browser Panel
Home
Favourites
/
/Volumes
MSSQL
PostGIS
SpatiaLite
OWS
WCS
WFS
WMS
Layers Panel
nybb
2015-11-08T17:43:41
0
QGIS Ready!
Coordinate:
851220,269879
Scale
1:234,433
Rotation:
0.0
Render
USER:100001

# 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 |

# Challenge - pyshp

- Write a simple script to plot out all the shapes in:

geofiles/nybb_15c/nybb_wgs84.shp

# Challenge - pyshp

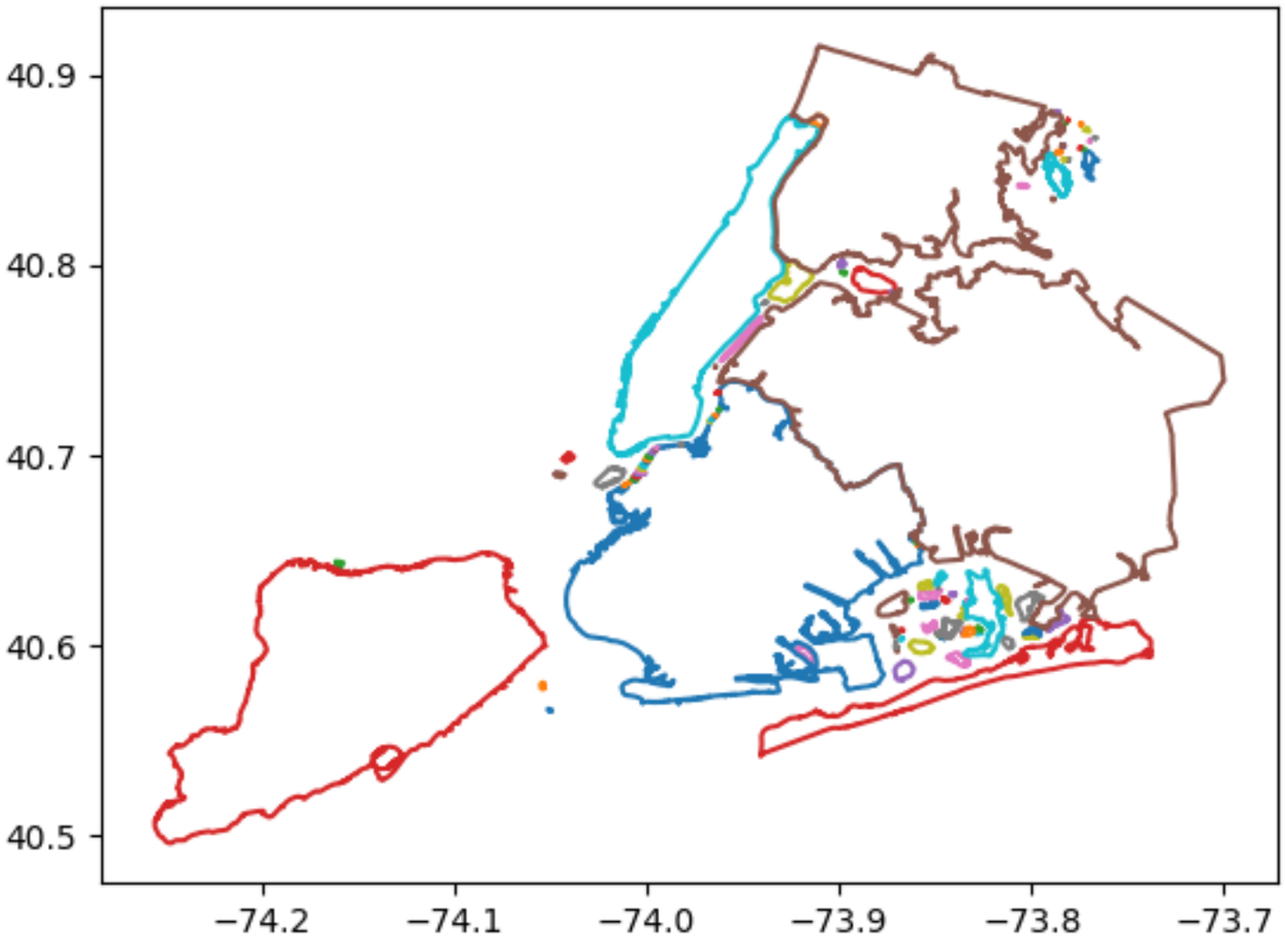
- Write a simple script to plot out all the shapes in:

geofiles/nybb_15c/nybb_wgs84.shp

```
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

# Challenge - 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

# 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

# Challenge - Filter points within a Shapefile

- Load each tweet in **NYC.json.gz** file by using:

```
tweet = eval(line.strip())
```

- on each line and write to **Manhattan.json.gz** all the tweets within Manhattan, as defined by

```
geofiles/nybb_15c/nybb_wgs84.shp
```

# Challenge - Filter points within a Shapefile

- Load each tweet in **NYC.json.gz** file by using:

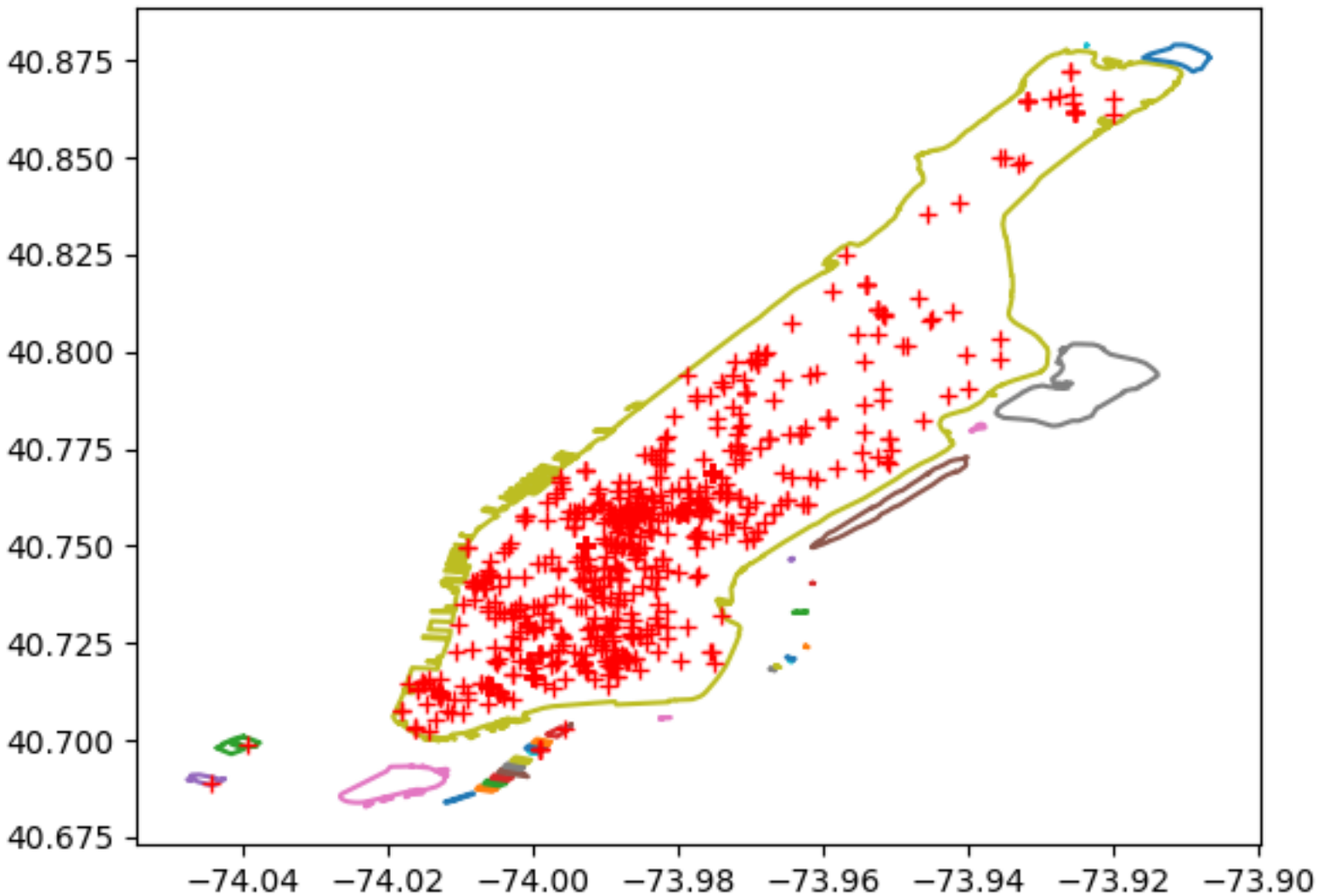
```
tweet = eval(line.strip())
```

- 

```
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()
```

shapefile_filter.py

# Challenge - Filter points within a Shapefile




*@bgoncalves*