# gpxpy -- GPX file parser

## Contenu

# gpxpy -- GPX file parser

http://www.trackprofiler.com/gpxpy/index.html

This is a simple python library for parsing and manipulating GPX files. GPX is an XML based format for GPS tracks.

You can see it in action on Trackprofiler.

## Usage

```
import gpxpy.parser as parser

gpx_file = open( 'test_files/cerknicko-jezero.gpx', 'r' )

gpx_parser = parser.GPXParser( gpx_file )
gpx_parser.parse()

gpx_file.close()

gpx = gpx_parser.get_gpx()

for track in gpx.tracks:
    for segment in track.segments:
        for point in segment.points:
            print 'Point at ({0},{1}) -> {2}'.format( point.latitude,
point.longitude, point.elevation )

for waypoint in gpx.waypoints:
    print 'waypoint {0} -> ({1},{2})'.format( waypoint.name,
waypoint.latitude, waypoint.longitude )

for route in gpx.routes:
    print 'Route:'
    for point in route:
        print 'Point at ({0},{1}) -> {2}'.format( point.latitude,
point.longitude, point.elevation )

# There are more utility methods and functions...

# You can manipulate/add/remove tracks, segments, points, waypoints and
routes and
# get the GPX XML file from the resulting object:

print 'GPX:', gpx.to_xml()
```

The code can be downloaded on github.

# Exploring GPX files

https://ocefpaf.github.io/python4oceanographers/blog/2014/08/18/gpx/

This post is an exercise on how to explore GPX files. Most of what I did in this post learned from this tutorial.

Deep down the GPX file format is just a XML document text. They can be parsed with any XML parser out there, but the gpxpy module makes that task much easier. Here is a quick example on how to load and explore the data inside a GPX file.

## In [2]:

```
import gpxpy
gpx = gpxpy.parse(open('./data/2014_08_05_farol.gpx'))

print("{} track(s)".format(len(gpx.tracks)))
track = gpx.tracks[0]

print("{} segment(s)".format(len(track.segments)))
segment = track.segments[0]

print("{} point(s)".format(len(segment.points)))
1 track(s)
1 segment(s)
1027 point(s)
```

Now let's extract the data for all those points. Here I have 1 track and 1 segment, but a GPX file might contain multiple tracks and segments. The best practice here is to always loop through all tracks and segments.

## In [3]:

```
data = []
segment_length = segment.length_3d()
for point_idx, point in enumerate(segment.points):
    data.append([point.longitude, point.latitude,
                 point.elevation, point.time,
segment.get_speed(point_idx)])

from pandas import DataFrame

columns = ['Longitude', 'Latitude', 'Altitude', 'Time', 'Speed']
df = DataFrame(data, columns=columns)
df.head()
```

**Out[3]:**

| | Longitude | Latitude | Altitude | Time | Speed |
|---|---|---|---|---|---|
| **0** | -38.502595 | -13.005390 | 10.9 | 2014-08-05 17:52:49.330 | NaN |
| **1** | -38.502605 | -13.005415 | 11.8 | 2014-08-05 17:52:49.770 | 2.672951 |
| **2** | -38.502575 | -13.005507 | 11.7 | 2014-08-05 17:52:54.730 | 3.059732 |
| **3** | -38.502545 | -13.005595 | 11.6 | 2014-08-05 17:52:57.750 | 4.220779 |
| **4** | -38.502515 | -13.005680 | 11.4 | 2014-08-05 17:53:00.720 | 3.939967 |

I want to plot the direction of the movement with a quiver plot. For that I will need the `u` and `v` velocity components. And to compute `u` and `v` I need the angle associated to each speed data. Instead of re-inventing the wheel I will use the `seawater` library `sw.dist` function to calculate the angles.

I also smoothed the data a little bit to improve the plot. (GPX data from smart-phones can be very noisy.)

# In [4]:

```
import numpy as np
import seawater as sw
from oceans.ff_tools import smoo1

_, angles = sw.dist(df['Latitude'], df['Longitude'])
angles = np.r_[0, np.deg2rad(angles)]

# Normalize the speed to use as the length of the arrows
r = df['Speed'] / df['Speed'].max()
kw = dict(window_len=31, window='hanning')
df['u'] = smoo1(r * np.cos(angles), **kw)
df['v'] = smoo1(r * np.sin(angles), **kw)
```

Now let's use mplleaflet to plot the track and the direction.

# In [5]:

```
import mplleaflet
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
df = df.dropna()
ax.plot(df['Longitude'], df['Latitude'],
        color='darkorange', linewidth=5, alpha=0.5)
sub = 10
ax.quiver(df['Longitude'][::sub], df['Latitude'][::sub], df['u'][::sub],
df['v'][::sub], color='deepskyblue', alpha=0.8, scale=10)
mplleaflet.display(fig=fig, tiles='esri_aerial')
```

**Out[5]:**


If you have tons of GPX files with your run data, it might come in handy to define a function to read them all at once.

# In [6]:

```
import os
from glob import glob

def load_run_data(gpx_path, filter=""):
    gpx_files = glob(os.path.join(gpx_path, filter + "*.gpx"))
    run_data = []
    for file_idx, gpx_file in enumerate(gpx_files):
        gpx = gpxpy.parse(open(gpx_file, 'r'))
        # Loop through tracks
        for track_idx, track in enumerate(gpx.tracks):
            track_name = track.name
            track_time = track.get_time_bounds().start_time
            track_length = track.length_3d()
            track_duration = track.get_duration()
            track_speed = track.get_moving_data().max_speed

            for seg_idx, segment in enumerate(track.segments):
                segment_length = segment.length_3d()
                for point_idx, point in enumerate(segment.points):
                    run_data.append([file_idx, os.path.basename(gpx_file),
track_idx, track_name,
                                     track_time, track_length,
track_duration, track_speed,
                                     seg_idx, segment_length, point.time,
point.latitude,
                                     point.longitude, point.elevation,
segment.get_speed(point_idx)])
    return run_data
```

# In [7]:

```
data = load_run_data(gpx_path='./data/GPX/', filter="")
df = DataFrame(data, columns=['File_Index', 'File_Name', 'Index', 'Name',
```

```
                                'Time', 'Length', 'Duration', 'Max_Speed',
                                'Segment_Index', 'Segment_Length',
'Point_Time', 'Point_Latitude',
                                'Point_Longitude', 'Point_Elevation',
'Point_Speed'])

HTML(df.head().to_html(max_cols=4))
```

**Out[7]:**

| | File_Index | File_Name | ... | Point_Elevation | Point_Speed |
|---|---|---|---|---|---|
| **0** | 0 | 2013_08_04_USP.gpx | ... | 764.0 | NaN |
| **1** | 0 | 2013_08_04_USP.gpx | ... | 767.0 | 1.726115 |
| **2** | 0 | 2013_08_04_USP.gpx | ... | 769.5 | 3.601075 |
| **3** | 0 | 2013_08_04_USP.gpx | ... | 772.0 | 3.540769 |
| **4** | 0 | 2013_08_04_USP.gpx | ... | 774.5 | 3.025701 |

Here I will clean up the DataFrame and convert the distances to km.

# In [8]:

```
cols = ['File_Index', 'Time', 'Length', 'Duration', 'Max_Speed']
tracks = df[cols].copy()
tracks['Length'] /= 1e3
tracks.drop_duplicates(inplace=True)
tracks.head()
```

**Out[8]:**

| | File_Index | Time | Length | Duration | Max_Speed |
|---|---|---|---|---|---|
| **0** | 0 | 2013-08-04 16:10:00 | 7.562737 | 6481 | 4.473565 |
| **712** | 1 | 2013-04-15 18:05:00 | 6.504129 | 2455 | 6.344318 |
| **877** | 2 | 2013-08-08 17:18:00 | 7.746483 | 2620 | 5.609248 |
| **1477** | 3 | 2013-04-22 17:42:00 | 7.281408 | 2445 | 5.618331 |
| **2192** | 4 | 2013-09-05 16:36:00 | 7.628724 | 4540 | 3.321567 |

And finally let's add a Track Year and Month columns based on track time. That way we can explore the run data with some stats and bar plots.

# In [9]:

```
tracks['Year'] = tracks['Time'].apply(lambda x: x.year)
tracks['Month'] = tracks['Time'].apply(lambda x: x.month)
tracks_grouped = tracks.groupby(['Year','Month'])
tracks_grouped.describe().head()
```

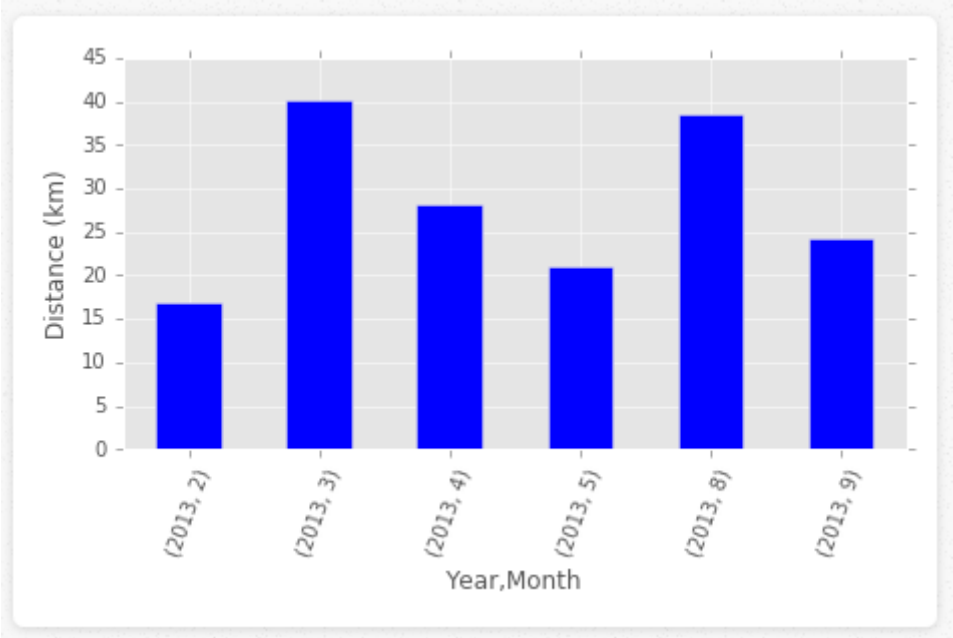
**Out[9]:**

| | | | Duration | File_Index | Length | Max_Speed |
|---|---|---|---|---|---|---|
| **Year** | **Month** | | | | | |

| | | | Duration | File_Index | Length | Max_Speed |
|---|---|---|---|---|---|---|
| **Year** | **Month** | | | | | |
| **2013** | **2** | **count** | 2.000000 | 2.000000 | 2.000000 | 2.000000 |
| | | **mean** | 2924.000000 | 12.000000 | 8.419711 | 4.691473 |
| | | **std** | 124.450793 | 8.485281 | 0.024573 | 0.740730 |
| | | **min** | 2836.000000 | 6.000000 | 8.402335 | 4.167698 |
| | | **25%** | 2880.000000 | 9.000000 | 8.411023 | 4.429585 |

# In [10]:

```
figsize=(7, 3.5)

tracks_grouped = tracks.groupby(['Year', 'Month'])
ax = tracks_grouped['Length'].sum().plot(kind='bar', figsize=figsize)
xlabels = [text.get_text() for text in  ax.get_xticklabels()]
ax.set_xticklabels(xlabels, rotation=70)
_ = ax.set_ylabel('Distance (km)')
```




Bad news! My goal was to run 50 km per month... I am clear way too far from accomplishing it! (Not to mentioned the fact that there is no data from 2014!)

To close this post I want to produce a plot similar to this using my run data.

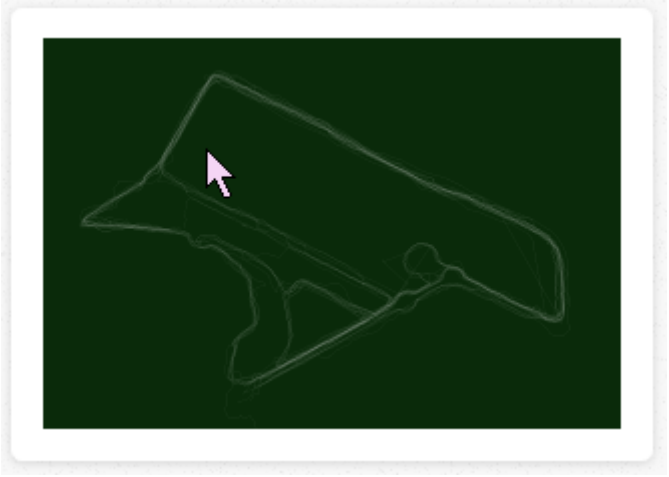
# In [11]:

```
def load_run_data(gpx_path, filter=""):
    gpx_files = glob(os.path.join(gpx_path, filter+"*.gpx"))
    run_data = []
    for file_idx, gpx_file in enumerate(gpx_files):
        try:
            gpx = gpxpy.parse(open(gpx_file, 'r'))
        except:
```

```
            os.remove(gpx_file)
            continue
        run_data_tmp = [[file_idx, gpx_file,
point.latitude,point.longitude, point.elevation]
                        for track in gpx.tracks
                            for segment in track.segments
                                for point in segment.points]
        run_data += run_data_tmp
    return run_data

def clear_frame(ax):
    ax.xaxis.set_visible(False)
    ax.yaxis.set_visible(False)
    for spine in ax.spines.values():
        spine.set_visible(False)

def plot_run_data(coords, **kwargs):
    columns = ['Index', 'File_Name', 'Latitude', 'Longitude', 'Altitude']
    coords_df = DataFrame(coords, columns=columns)
    grouped = coords_df.groupby('Index')
    fig, ax = plt.subplots(figsize=kwargs.get('figsize', (13 ,8)))

    bgcolor = kwargs.get('bgcolor', '#001933')
    color = kwargs.get('color', '#FFFFFF')
    linewidth = kwargs.get('linewidth', .035)
    alpha = kwargs.get('alpha', 0.5)

    kw = dict(color=color, linewidth=linewidth, alpha=alpha)
    for k, group in grouped:
        ax.plot(group['Longitude'], group['Latitude'], **kw)
    ax.grid(False)
    ax.patch.set_facecolor(bgcolor)
    ax.set_aspect('auto','box','C')
    clear_frame(ax)
    fig.subplots_adjust(left=0, right=1, top=1, bottom=.1)
    return ax
```

# In [12]:

```
df = load_run_data(gpx_path='./data/GPX/')
ax = plot_run_data(df, figsize=(4, 3), alpha=0.85,
                   bgcolor='#0A2A0A')
_ = ax.axis([-46.74, -46.71, -23.57, -23.55])
```



I tried to find public run data for Salvador to discover the best places to run here using that kind of plot. First I tried RunKeeper, the app does make their public data available online, but it is not a popular app in Brazil and I could not find any tracks for Salvador in the database. Sportstraker, on the other hand, is very popular here. But Sportstraker do not publish the public data online.

If you read this and have some GPX files data from your training and want to see a map of Salvador most popular places to run, get in touch!

# In [13]:

```
HTML(html)
```

### Out[13]:

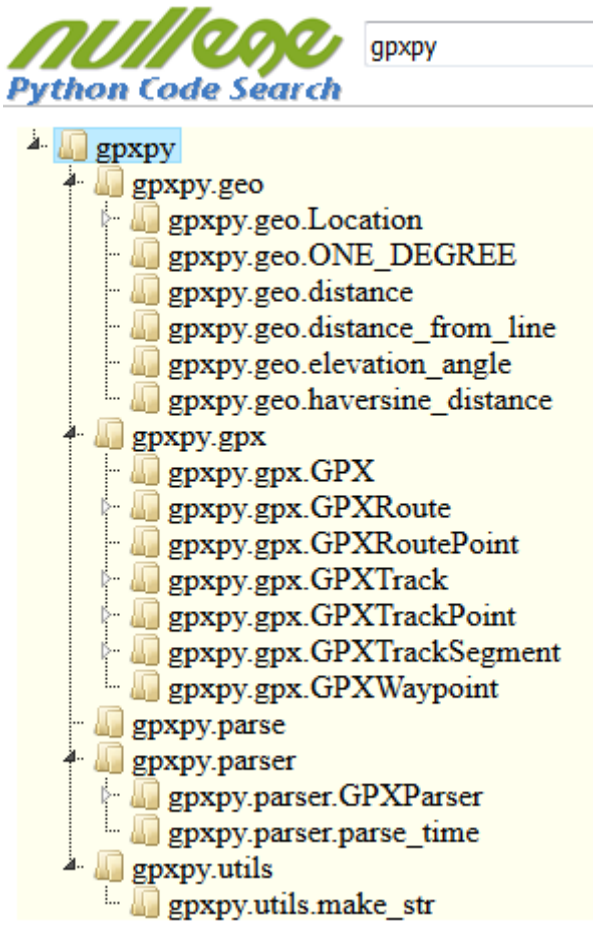
This post was written as an IPython notebook. It is available for download or as a static html.



Posted by Filipe Fernandes Aug 18, 2014

# Nullege

http://nullege.com/codes/search/gpxpy



# gpxpy -- GPX file parser

This is a simple Python library for parsing and manipulating GPX files. GPX is an XML based format for GPS tracks.

You can see it in action on my online GPS track editor and organizer.

There is also a Golang port of gpxpy: gpxgo.

See also srtm.py if your track lacks elevation data.

https://github.com/tkrajina/gpxpy

## Usage

```
import gpxpy
import gpxpy.gpx

# Parsing an existing file:
# -------------------------

gpx_file = open('test_files/cerknicko-jezero.gpx', 'r')

gpx = gpxpy.parse(gpx_file)

for track in gpx.tracks:
    for segment in track.segments:
        for point in segment.points:
            print 'Point at ({0},{1}) -> {2}'.format(point.latitude,
point.longitude, point.elevation)

for waypoint in gpx.waypoints:
    print 'waypoint {0} -> ({1},{2})'.format(waypoint.name,
waypoint.latitude, waypoint.longitude)

for route in gpx.routes:
    print 'Route:'
    for point in route.points:
        print 'Point at ({0},{1}) -> {2}'.format(point.latitude,
point.longitude, point.elevation)

# There are many more utility methods and functions:
# You can manipulate/add/remove tracks, segments, points, waypoints and
routes and
# get the GPX XML file from the resulting object:

print 'GPX:', gpx.to_xml()

# Creating a new file:
# --------------------
```

```
gpx = gpxpy.gpx.GPX()

# Create first track in our GPX:
gpx_track = gpxpy.gpx.GPXTrack()
gpx.tracks.append(gpx_track)

# Create first segment in our GPX track:
gpx_segment = gpxpy.gpx.GPXTrackSegment()
gpx_track.segments.append(gpx_segment)

# Create points:
gpx_segment.points.append(gpxpy.gpx.GPXTrackPoint(2.1234, 5.1234,
elevation=1234))
gpx_segment.points.append(gpxpy.gpx.GPXTrackPoint(2.1235, 5.1235,
elevation=1235))
gpx_segment.points.append(gpxpy.gpx.GPXTrackPoint(2.1236, 5.1236,
elevation=1236))

# You can add routes and waypoints, too...

print 'Created GPX:', gpx.to_xml()
```

# GPX Version:

gpx.py can parse and generate GPX 1.0 and 1.1 files. Note that the generated file will always be a valid XML document, but it may not be (strictly speaking) a valid GPX document. For example, if you set gpx.email to "my.email AT mail.com" the generated GPX tag won't confirm to the regex pattern. And the file won't be valid. Most applications will ignore such errors, but... Be aware of this!

WARNING: The only part of the GPX standard which is not completely implemented are GPX extensions. The API for GPX extensions will change in future versions!!!

Be aware that the gpxpy object model *is not 100% equivalent* with the underlying GPX XML file schema. That's because the library object model works with both GPX 1.0 and 1.1.

For example, the GPX 1.0 specified a `speed` attribute for every track point, but that was removed in GPX 1.1. If you parse GPX 1.0 and serialize back with `gpx.to_xml()` everything will work fine. But if you have a GPX 1.1 object, changes in the `speed` attribute will be lost after `gpx.to_xml()`. If you want to force using 1.0, you can `gpx.to_xml(version="1.0")`. Another possibility is to use `extensions` to save the speed in GPX 1.1.

# XML parsing

If lxml is available, then it will be used for XML parsing. Otherwise minidom is used. Note that lxml is 2-3 times faster so, if you can choose -- use it :)

The GPX version is automatically determined when parsing by reading the version attribute in the gpx node. If this attribute is not present then the version is assumed to be 1.0. A specific version can be forced by setting the `version` parameter in the parse function. Possible values for the 'version' parameter are `1.0`, `1.1` and `None`.

# Pull requests

OK, so you found a bug and fixed it. Before sending a pull request -- check that all tests are OK with Python 2.6+ and Python 3+.

Run all tests with:

```
$ python -m unittest test
$ python3 -m unittest test
```

Run only minidom parser tests with:

```
$ python -m unittest test.MinidomTests
$ python3 -m unittest test.MinidomTests
```

Run only lxml parser tests with:

```
$ python -m unittest test.LxmlTests
$ python3 -m unittest test.LxmlTests
```

Run a single test with:

```
$ python -m unittest test.LxmlTests.test_method
$ python3 -m unittest test.LxmlTests.test_method
```

# GPXInfo

The repository contain a little command line utility to extract basic statistics from a file. Example usage:

```
$ gpxinfo voznjica.gpx
File: voznjica.gpx
  Length 2D: 63.6441229018
  Length 3D: 63.8391428454
  Moving time: 02:56:03
  Stopped time: 00:21:38
  Max speed: 14.187909492m/s = 51.0764741713km/h
  Total uphill: 1103.1626183m
  Total downhill: 1087.7812703m
  Started: 2013-06-01 06:46:53
  Ended: 2013-06-01 10:23:45
```

# License

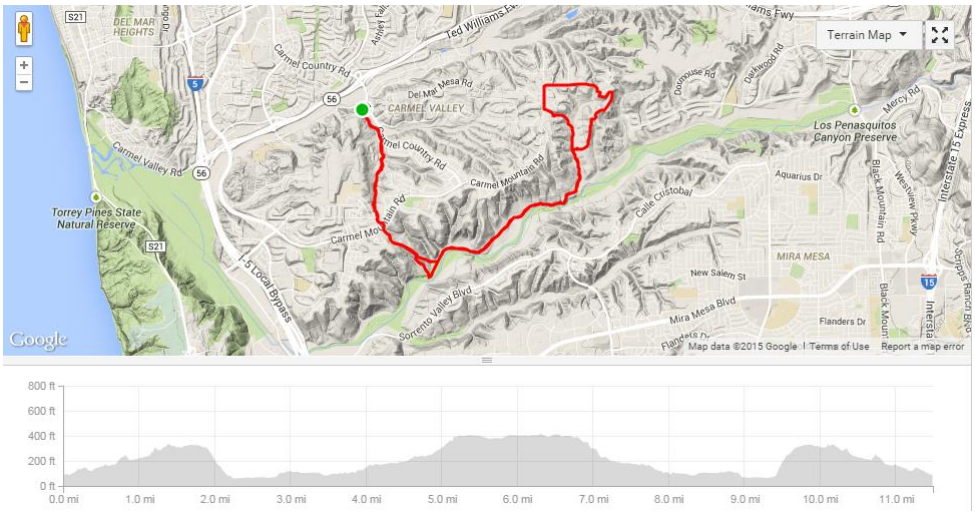
GPX.py is licensed under the Apache License, Version 2.0

# Visualizing Strava Tracks with Python

http://andykee.com/visualizing-strava-tracks-with-python.html

I love using Strava to track of my biking and snowboarding. While I rarely compete with the hoards of people who are much faster than I am (or are using DigitalEPO), I do like to compare my times against my own PR's.

Strava generally does a nice job of allowing you to explore individual activities, but no method exists for simultaneously visualizing multiple activities. I commonly ride the same trails but slightly vary my route. Take the ride below for example: this represents one of perhaps a dozen different rides I've done on this same trail system. It would be nice to visualize where I ride the most and where I ride the least.




Thankfully, Strava makes it really easy to export raw position data from completed activities as GPX files. GPX (GPS Exchange Format) is an XML data format for GPS data. The GPX format is able to describe waypoints, tracks, and routes. Strava uses tracks to store activity position data as follows:

```
<gpx>
  <trk>
    <trkseg>
      <trkpt lat="DD.ddd" lon="DD.ddd">
        <ele>MMM.mm</ele>
        <time>YYY-MM-DD HH:MM:SS</time>
      </trkpt>
      ...
    </trkseg>
  </trk>
</gpx>
```

We'll make use of two Python libraries here: `gpxpy` to read GPX files into Python and `matplotlib` to for plotting figures. First, let's load an activity into Python:

```
import gpxpy
import matplotlib.pyplot as plt

gpx_file = open('20140918-LPQ-Ride.gpx', 'r')
gpx = gpxpy.parse(gpx_file)
```

We can then traverse the GPX data structure to extract position data. In this case, since we're only visualizing the data in 2D, there's no need to pull the elevation data along with each point.

```
lat = []
lon = []

for track in gpx.tracks:
    for segment in track.segments:
        for point in segment.points:
            lat.append(point.latitude)
            lon.append(point.longitude)
```

Finally, `matplotlib` allows us to plot the activity:

```
fig = plt.figure(facecolor = '0.05')
ax = plt.Axes(fig, [0., 0., 1., 1.], )
ax.set_aspect('equal')
ax.set_axis_off()
fig.add_axes(ax)
plt.plot(lon, lat, color = 'deepskyblue', lw = 0.2, alpha = 0.8)
```

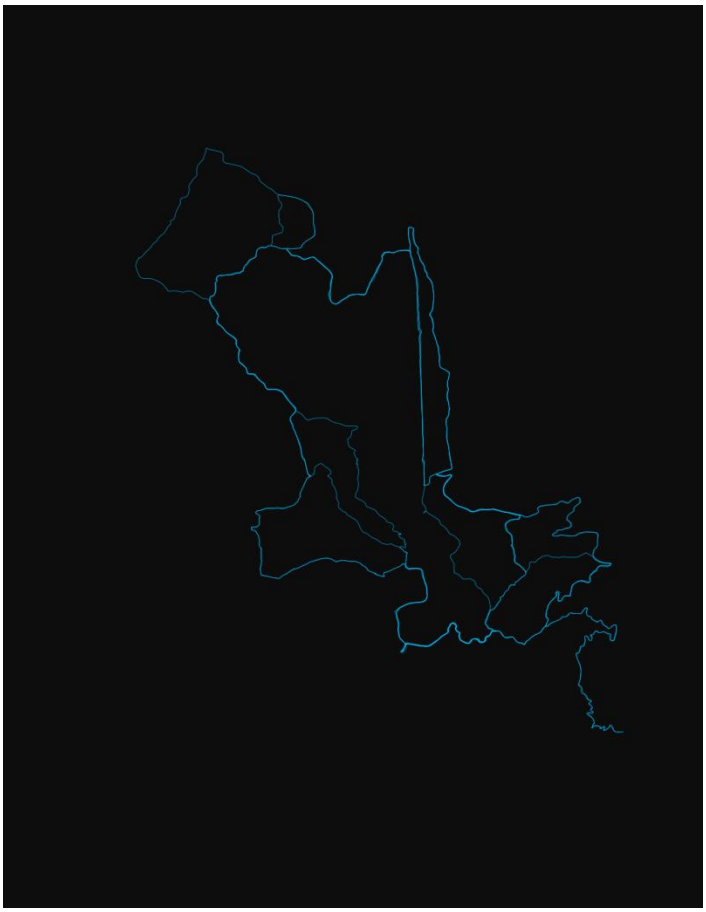
The resulting plot, unsurprisingly, looks like the track we saw on Strava's web app (above). This little bit of code provides the foundation for exploring and visualizing trail usage trends over time.

With a little more Python code, we can read in a bunch of Strava activities from a directory, plot them together, and save the result to a file:

```
from os import listdir
from os.path import isfile, join
import matplotlib.pyplot as plt
```

```
import gpxpy

data_path = 'lpq'
data = [f for f in listdir(data_path) if isfile(join(data_path,f))]

lat = []
lon = []

fig = plt.figure(facecolor = '0.05')
ax = plt.Axes(fig, [0., 0., 1., 1.], )
ax.set_aspect('equal')
ax.set_axis_off()
fig.add_axes(ax)

for activity in data:
    gpx_filename = join(data_path,activity)
    gpx_file = open(gpx_filename, 'r')
    gpx = gpxpy.parse(gpx_file)

    for track in gpx.tracks:
        for segment in track.segments:
            for point in segment.points:
                lat.append(point.latitude)
                lon.append(point.longitude)
    plt.plot(lon, lat, color = 'deepskyblue', lw = 0.2, alpha = 0.8)
    lat = []
    lon = []

filename = data_path + '.png'
plt.savefig(filename, facecolor = fig.get_facecolor(), bbox_inches='tight',
pad_inches=0, dpi=300)
```

I made a few more for two other trail systems I commonly ride. Next up: visualizing my Moves data.



# GPX PY Overview

http://nbviewer.jupyter.org/github/titsworth/hsvpy-runtalk/blob/master/2%20-%20GPXPy%20Overview.ipynb

The gpxpy package provides a bunch of great tools for parsing, correcting, and manipulating GPX files. It also has some nice analysis capabilities built into it. You can get a few valuable metrics about your run/bike right out of the box. These include

- Length of a run
- Min and max elevation of a run
- Duration of run
- Max speed

Let's start by importing package of course! We'll also go ahead and import other packages that will be used in this notebook.

In [1]:
```
import gpxpy
import gpxpy.gpx

import glob
import os
import datetime
import pandas as pd
import matplotlib.pyplot as plt

%matplotlib inline
```

## A Quick Look at the GPX format

Before we really get started, we shoul probably see what the GPX format is.

Let's load up the first file in our data set and look at it's contents.

In [2]:
```
gpx_files = glob.glob(os.path.join("rundata","*.gpx"))
```

In [3]:
```
%cat ./${os.path.join("rundata",gpx_files[0])}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<gpx
  version="1.1"
  creator="RunKeeper - http://www.runkeeper.com"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://www.topografix.com/GPX/1/1"
  xsi:schemaLocation="http://www.topografix.com/GPX/1/1 http://www.topografix.com/GPX/1/1/gpx.xsd"
  xmlns:gpxtpx="http://www.garmin.com/xmlschemas/TrackPointExtension/v1">
<trk>
  <name><![CDATA[Running 6/21/11 5:49 am]]></name>
  <time>2011-06-21T05:49:43Z</time>
<trkseg>
<trkpt lat="34.789056733" lon="-86.791616417"><ele>216.7</ele><time>2011-06-21T05:49:43Z</time></trkpt>
<trkpt lat="34.789059133" lon="-86.791517717"><ele>216.9</ele><time>2011-06-21T05:49:45Z</time></trkpt>
<trkpt lat="34.789063417" lon="-86.791391000"><ele>217.0</ele><time>2011-06-21T05:49:50Z</time></trkpt>
<trkpt lat="34.789069200" lon="-86.791261650"><ele>217.1</ele><time>2011-06-21T05:49:54Z</time></trkpt>
<trkpt lat="34.789081917" lon="-86.791144083"><ele>217.2</ele><time>2011-06-21T05:49:57Z</time></trkpt>
<trkpt lat="34.789059500" lon="-86.791042233"><ele>217.3</ele><time>2011-06-21T05:50:01Z</time></trkpt>
<trkpt lat="34.789030983" lon="-86.790939500"><ele>217.3</ele><time>2011-06-21T05:50:06Z</time></trkpt>
```

.../...

```
<trkpt lat="34.791683667" lon="-86.787290250"><ele>212.0</ele><time>2011-06-21T06:18:10Z</time></trkpt>
<trkpt lat="34.791592517" lon="-86.787318300"><ele>212.0</ele><time>2011-06-21T06:18:14Z</time></trkpt>
<trkpt lat="34.791494767" lon="-86.787355100"><ele>212.0</ele><time>2011-06-21T06:18:18Z</time></trkpt>
</trkseg>
</trk>
</gpx>
```

# gpxpy's Representation

Now let's see what gpxpy does with the data

```
In [4]: gpx = gpxpy.parse(open(gpx_files[97]))
        gpx
```

```
Out[4]: GPX(tracks=[GPXTrack(name=u'Running 8/11/12 6:02 am', number=0, segments=[GPX
```

You can see what functions are available using IPython's tab completion.

```
In [5]: gpx.tracks
```

```
Out[5]: [GPXTrack(name=u'Running 8/11/12 6:02 am', number=0, segments=[GPXTrackSegment(points=[...
```

Let's look at the track data

```
In [6]: gpx_track = gpx.tracks[0]
        gpx_track
```

```
Out[6]: GPXTrack(name=u'Running 8/11/12 6:02 am', number=0, segments=[GPXTrac
```

### Track Summary

Some information on this track

```
In [7]: print("Name: " + gpx_track.name)
        print("Description: " + str(gpx_track.description))
        print("Start: " + str(gpx_track.get_time_bounds().start_time.isoformat()))
        print("End: " + str(gpx_track.get_time_bounds().end_time))

        bounds = gpx_track.get_bounds()
        print("Latitude Bounds: (%f, %f)" % (bounds.min_latitude, bounds.max_latitude))
        print("Longitude Bounds: (%f, %f)" % (bounds.min_longitude, bounds.max_longitude))
```

```
Name: Running 8/11/12 6:02 am
Description: None
Start: 2012-08-11T06:02:09
End: 2012-08-11 07:31:42
Latitude Bounds: (34.713744, 34.735524)
Longitude Bounds: (-86.703750, -86.676670)
```

### Run Duration

Duration returned in seconds.

```
In [8]: gpx_track.get_duration()*1./60
Out[8]: 89.55
```

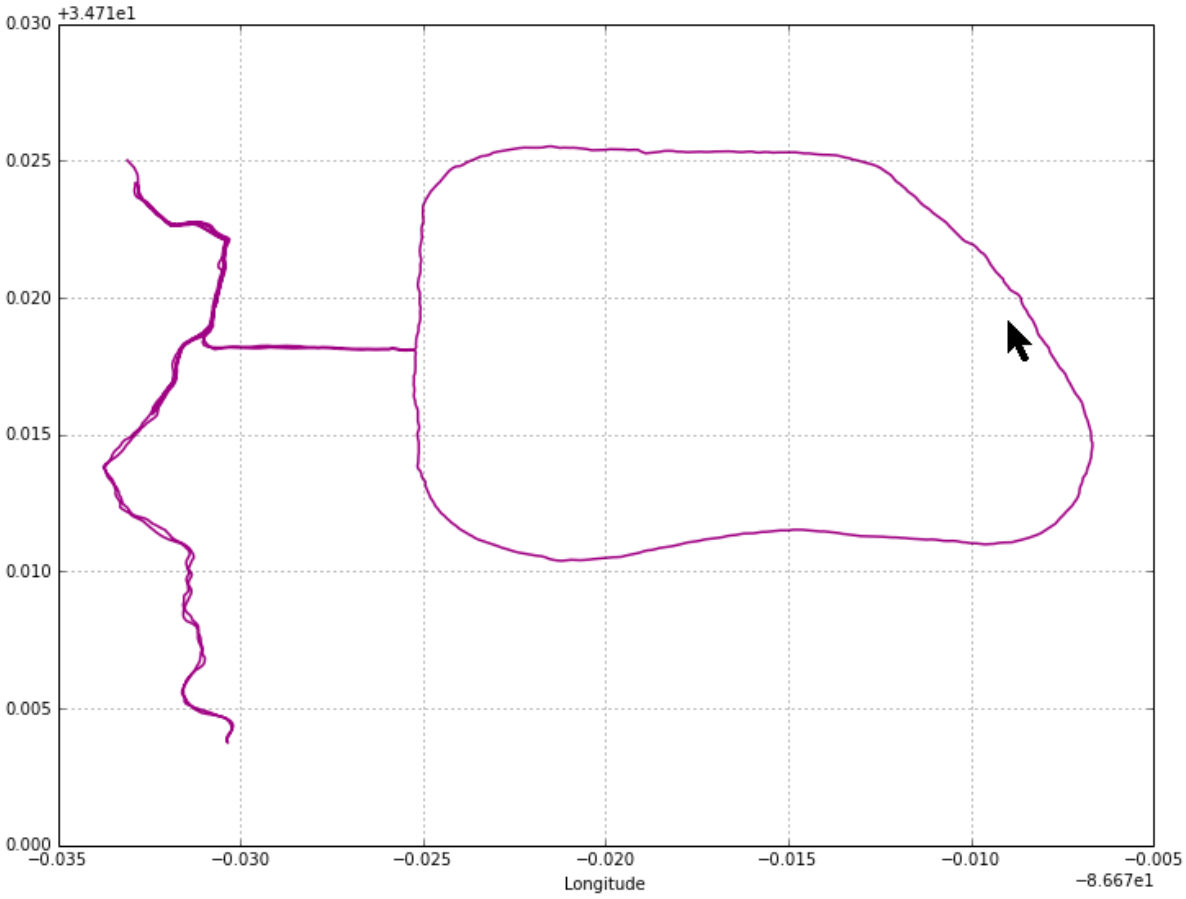
### Run Distance

What wat the length of the run? Length returned in meters. 2d and 3d distance is available.

```
In [9]: gpx_track.length_2d()
Out[9]: 14487.726193240629

In [10]: gpx_track.length_3d()
Out[10]: 14495.595071043379
```

### Quick Visualization

```
In [11]: track_coords = [[point.latitude,point.longitude, point.elevation]
                                  for track in gpx.tracks
                                      for segment in track.segments
                                          for point in segment.points]

         coords_df = pd.DataFrame(track_coords, columns=['Latitude','Longitude','Altitude'])

         fig = plt.figure(figsize=(12,9))
         coords_df.plot('Longitude','Latitude', color='#A00084', linewidth=1.5)
Out[11]: <matplotlib.axes.AxesSubplot at 0x106e71e10>
```


+3.471e1
0.030
0.025
0.020
0.015
0.010
0.005
0.000
−0.035
−0.030
−0.025
−0.020
−0.015
−0.010
−0.005
Longitude
−8.667e1