

9 Projektionen und Vektordaten

9.1 Das Modul pyproj

Proj.4 ist eine Bibliothek zur Konversion von verschiedenen (kartografischen) Projektions-Systemen resp. Referenzsystemen. Die Bibliothek wurde ursprünglich von Gerald Everding (USGS) entwickelt, ist heute aber ein OSGeo Projekt, welches von Frank Warmerdam unterhalten wird. Proj.4 wird in der Programmiersprache C entwickelt, daraus wird ein Python-Modul abgeleitet.

(siehe auch: <https://www.swisstopo.admin.ch/de/karten-daten-online/calculation-services.html>)

Installation (Anaconda-Prompt)

```
conda install pyproj
```

Proj.4: <https://trac.osgeo.org/proj/>
pyproj (Python Modul): <https://code.google.com/p/pyproj/>

Wichtige Referenzsysteme:

Geografisches WGS84 (3D): EPSG: 4326

Web-Mapping Mercator (z.B. Google Maps): EPSG: 3857
(EPSG: 900913 nicht verwenden!)

CH1903/LV03: EPSG: 21781

CH1903r/LV95: EPSG: 2056

EPSG = European Petroleum Survey Group

Weitere Referenzsysteme respektive deren Definitionen können unter folgendem Link betrachtet werden: <http://epsg.io>.

9.1.1 Koordinatentransformation

Um eine Koordinatentransformation durchzuführen wird ein Transformer-Objekt erstellt. Dabei wird die Beschreibung des Referenzsystems am einfachsten als EPSG Code angegeben.

```
from pyproj import Transformer

transformer = Transformer.from_crs('EPSG:2056', 'EPSG:4326')

resultat = transformer.transform(2600000, 1200000)

print(resultat)
```

Die Beschreibung des Koordinatensystems kann auch ein WKT-String (OGC) oder über proj4 Parameter definiert werden. Wir verwenden im Moment der Einfachheit halber zunächst nur einen EPSG Code.

9.1.2 Koordinatentransformation eines CSV Datensatzes

Das amtliche Ortschaftenverzeichnis mit Postleitzahl und Perimeter kann unter folgendem Link als csv heruntergeladen werden:

Direkter Link LV95: http://data.geo.admin.ch/ch.swisstopo-vd.ortschaftenverzeichnis_plz/PLZO_CSV_LV95.zip

```
import csv
from pyproj import Transformer

transformer = Transformer.from_crs('EPSG:2056', 'EPSG:4326')

file = open("PLZO_CSV_LV95.csv", encoding="cp1252")

reader = csv.DictReader(file, delimiter=";")
for row in reader:
    x = float(row["E"])
    y = float(row["N"])
    lat, lng = transformer.transform(x, y)

    Ortschaftsname = row["Ortschaftsname"]
    PLZ = row["PLZ"]
    Zusatzziffer = row["Zusatzziffer"]
    Gemeindename = row["Gemeindename"]
    Kanton = row["Kantonskürzel"]

    print(Ortschaftsname, PLZ, Zusatzziffer, Gemeindename, Kanton, lng, lat)

file.close()
```

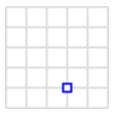
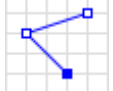


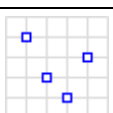
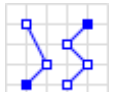
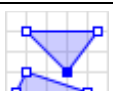
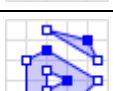
9.1.3 Shapely

Shapely ist eine Bibliothek für die Analyse und Bearbeitung geometrischer Objekte im kartesischen Raum. Shapely vereinfacht auch den Daten-Zugriff aus ogr. Die Dokumentation ist verfügbar unter: <http://toblerity.org/shapely/>

Die Installation über die Anaconda Prompt erfolgt mit:

```
conda install shapely
```

Mit WKT lassen sich geometrische Objekte relativ einfach darzustellen:

Typ	Beispiel	
Point		POINT (30 10)
LineString		LINESTRING (30 10, 10 30, 40 40)
Polygon		POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))
		POLYGON ((35 10, 45 45, 15 40, 10 20, 35 10), (20 30, 35 35, 30 20, 20 30))
MultiPoint		MULTIPOINT ((10 40), (40 30), (20 20), (30 10)) MULTIPOINT (10 40, 40 30, 20 20, 30 10)
MultiLineString		MULTILINESTRING ((10 10, 20 20, 10 40), (40 40, 30 30, 40 20, 30 10))
MultiPolygon		MULTIPOLYGON (((30 20, 45 40, 10 40, 30 20)), ((15 5, 40 10, 10 20, 5 10, 15 5)))
		MULTIPOLYGON (((40 40, 20 45, 45 30, 40 40)), ((20 35, 10 30, 10 10, 30 5, 45 20, 20 35), (30 20, 20 15, 20 25, 30 20)))

Quelle: http://en.wikipedia.org/wiki/Well-known_text

in ogr kann eine Geometrie auch direkt aus einem WKT erstellt werden, dies funktioniert folgendermassen:

```
import ogr
```

Sehen wir uns zunächst folgendes Beispiel an:

Mit Shapely generieren wir ein Punkt.

```
from shapely.geometry import Point

FHNW = Point([47.534792, 7.641529])
FHNW.wkt
```

```
from shapely.geometry import Point

BSL = Point([47.598829, 7.529104])
BSL.wkt
```

Alternativ können wir anstelle von Point auch LineString, Polygon, MultiPolygon usw. verwenden.

Laden wir nun ein Multipolygon (Schweiz) als wkt-String:

Dazu verwenden wir das submodul «wkt»:

```
from shapely import wkt
```

Und öffnen das Multipolygon der Schweiz:

```
file = open("schweiz.wkt")
ch = file.read()
file.close()

schweiz = wkt.loads(ch)
```

Mit Matplotlib können wir die aussenhülle des Polygons darstellen. Bei Multipolygons müssen wir die einzelnen Polygone mit einer for-Schleife iterieren.

```
import matplotlib.pyplot as plt

for geometry in schweiz.geoms:
    x,y = geometry.exterior.xy
    plt.plot(x,y)

plt.show()
```

9.1.4 Binäre Operationen

Eine weitere wichtige geometrische Operation ist die Beziehung zwischen zwei Objekten. Shapely stellt unter anderem folgende binäre Operationen zur Verfügung:

contains	Returns True if the interior of the object intersects the interior of the other but does not contain it, and the dimension of the intersection is less than the dimension of the one or the other.
intersects	Returns True if the boundary and interior of the object intersect in any way with those of the other.
within	Returns True if the object's boundary and interior intersect only with the interior of the other (not its boundary or exterior).
touches	Returns True if the objects have at least one point in common and their interiors do not intersect with any part of the other.
crosses	Returns True if the interior of the object intersects the interior of the other but does not contain it, and the dimension of the intersection is less than the dimension of the one or the other.
equals	Returns True if the set-theoretic boundary, interior, and exterior of the object coincide with those of the other.

(mehr unter: <http://toblerity.org/shapely/manual.html#binary-predicates>)

Beispiel: Wir testen, ob die Punkte innerhalb der Schweiz sind:

```
BSL.within(schweiz)
```

```
FHNW.within(schweiz)
```

9.1.5 Mehrere Polygone und Operationen

Mit Shapely können wir auch Mengenoperationen mit geometrischen Objekten ausführen.

Wir erstellen zwei Polygone:

```
wkt1 = "POLYGON (( -5 -5, 5 -5, 5 5, -5 5, -5 -5))"
wkt2 = "POLYGON ((1 -1, 4 -1, 4 1, 1 1, 1 4, -1 4, -1 1, -4 1, -4 -1, -1 -1, -1 -4, 1 -4, 1 -1))"
quadrat = shapely.wkt.loads(wkt1)
kreuz = shapely.wkt.loads(wkt2)
```

Hinweis: Interiors und exteriors sind immer im entgegengesetzten Uhrzeigersinn!

```
import matplotlib.pyplot as plt

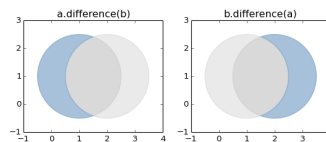
x,y = quadrat.exterior.xy
plt.plot(x,y, 'b-')

x,y = kreuz.exterior.xy
plt.plot(x,y, 'r-')

plt.show()
```

Differenz

```
diff = shape1.difference(shape2)
```

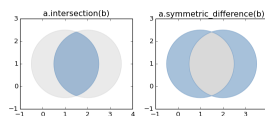


Schnittmenge (Intersection)

```
inter = shape1.intersection(shape2)
```

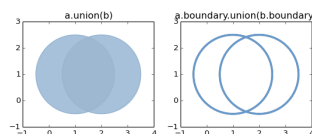
Symmetrische Differenz

```
sdiff = shape1.symmetric_difference(shape2)
```



Vereinigung (Union)

```
uni = shape1.union(shape2)
```



9.1.6 Delaunay Triangulierung

```
from shapely.ops import triangulate
import shapely.wkt
import numpy as np
wkt = "POLYGON ((1 -1, 4 -1, 4 1, 1 1, 1 4, -1 4, -1 1, -4 1, -4 -1, -1 -1, -1 -4, 1 -4, 1 -1))"
data = shapely.wkt.loads(wkt)
triangles = triangulate(data)
```

```
for tri in triangles:
    x, y = tri.exterior.coords.xy
    plt.plot(x,y)
plt.show()
```