Introduction to Shapely

TL;DR

Geometry in GeoPandas is handled by Shapely. The Shapely User Manual is here.

If you just want to break through the abstractions and grab the data, here is a quick example.

We load the GeoDataFrame containing the data for the Hawaiian archipelegos, as shown elsewhere.

The dataframe has a 'geometry' which can be indexed by an int using `.iloc`. There is a single member which is a MultiPolygon (a collection) with 9 members.

The MultiPolygon's `.geoms` attribute gives a GeoSeries of Polygon objects. This can also be indexed by an int. There are 9 elements, corresponding to the 9 islands.

Each of the island Polygons has an exterior (and empty interior) and the coordinates are then obtained by `coords.xy`:

```
>>> mp = gdf['geometry'].iloc[0]
>>> big_island = mp.geoms[0]
>>> X,Y = big_island.exterior.coords.xy
>>> len(X)
230
>>>
```

The polygons or islands don't appear to be associated with any names.

Shapes

Shapely objects include things like

- Point
- LineString
- LinearRing
- Polygon
- MultiPolygon

Space is usually restricted strictly to two dimensions, there is no idea of elevation, just `xy` coords (but see e.g. `help(Point)`.

There are no curves in Shapely-land, just polygons with more and more points.

Points, LineStrings and LinearRings

```
>>> import shapely as shp
>>>
>>> p = shp.Point([0,1])  # Point(0,1) also works
>>> print(p)
POINT (0 1)
>>>
```

A Point has a surprising number of methods.

A LineString is an array of points. We construct one from a list of `(x,y)` tuples.

```
>>> ls = shp.LineString([[0,0],[1,0],[1,1],[0,1]])
>>> ls
<LINESTRING (0 0, 1 0, 1 1, 0 1)>
>>>
```

The constructor will also take an array of point objects.

```
>>> q = shp.Point([1,0])
>>> shp.LineString([p,q]).length
1.4142135623730951
>>> ls.area
0.0
>>>
```

A LinearRing is closed.

First, add a copy of the first point to the end of our LineString:

```
>>> ls = shp.LineString([[0,0],[1,0],[1,1],[0,1],[0,0]])
>>> lr = shp.LinearRing(ls)
>>> print(lr)
LINEARRING (0 0, 1 0, 1 1, 0 1, 0 0)
>>> lr.is_valid
True
>>>
```

> A LinearRing may not cross itself, and may not touch itself at a single point.

Polygons

```
>>> poly = shp.Polygon(lr)
>>> poly
<POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0))>
>>> poly.is_valid
True
>>> poly.area
1.0
>>> poly.contains(shp.Point([0.5,0.5]))
True
>>> poly.geom_type
'Polygon'
>>>
```

Somewhat surprisingly a Polygon object may contain more than one geometric shape. The second object (or more) is a "hole" in the outer shape.

```
>>> outer = shp.LinearRing([[0,0],[4,0],[4,4],[0,4],[0,0]])
>>> inner = shp.LinearRing([[1,1],[3,1],[3,3],[1,3],[1,1]])
>>> p = shp.Polygon(outer,holes=[inner])
>>> p
<POLYGON ((0 0, 4 0, 4 4, 0 4, 0 0), (1 1, 3 1, 3 3, 1 3, 1 1))>
>>> p.is_valid
True
>>> p.area
12.0
>>>
```

Area subtracts the holes.

The inner shape may touch the outer shape, but only at precisely one point.

Collections

> Heterogeneous collections of geometric objects may result from some Shapely operations. For example, two LineStrings may intersect along a line [or] at a point.

The collection we will run into a lot is the MultiPolygon.

```
>>> p1 = shp.Polygon([[0,0],[1,0],[1,1],[0,1],[0,0]])
>>> p2 = shp.Polygon([[2,0],[3,0],[3,1],[2,1],[2,0]])
>>> mp = shp.MultiPolygon([p1,p2])
>>> mp.is_valid
True
>>> mp.geoms
<shapely.geometry.base.GeometrySequence object at 0x113187ee0>
>>> mp.geoms[0]
<POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0))>
>>>
```

Alternatively, you can do a set operation:

```
>>> p1.union(p2)
<MULTIPOLYGON (((0 0, 1 0, 1 1, 0 1, 0 0)), ((2 0, 3 0, 3 1, 2 1, 2 0)))>
>>>
```

Deconstructing Hawaii

We make this stuff concrete by drilling down into a MultiPolygon to the underlying data.

Run these commands as a script. First our imports:

```
import sys,os,subprocess
import geopandas as gpd
import matplotlib.pyplot as plt

func_name = __file__.split('.')[0]
```

Load the data for the US

```
fn = 'gz_2010_us_040_00_5m'
HOME = os.path.expanduser('~')
path = HOME + '/data/' + fn
gdf = gpd.read_file(path)
```

Filter for just FIPS == '15', i.e. Hawaii.

```
sel = gdf['STATE'] == '15'
gdf = gdf[sel]
print(gdf.shape)  # (1,6)  # gdf has one row
```

`gdf['geometry']` is its "active" geometry. (Meaning: a dataframe may contain other "geometries", Polygons, etc., but only one is active at a time)

```
gs = gdf['geometry']
print(gs.shape)   # (1,)
print(type(gs))
```

prints

```
<class 'geopandas.geoseries.GeoSeries'>
```

It can be subscripted.

```
mp =gdf['geometry'].iloc[0]
```

`mp` is a MultiPolygon. It has component `geoms`.

```
geos = mp.geoms
print(len(geos))   # 9
```

Hawaii has 9 component polygons, 9 islands. `geos` is a shapely.GeometrySequence. In turn, it can also be subscripted.

```
poly = geos[0]
```

`poly` is a `shapely.geometry.polygon.Polygon` The polygon consists of an exterior and one or more interiors (which are not present in this case).

So finally, we get the points of the exterior LinearRing of a Polygon by:

```
ext = poly.exterior
X,Y = ext.coords.xy
X = X.tolist()
print(len(X))      # 230
```

230 points.

We can iterate through the 9 polygons:

```
for p in geos:
    print('%3.5f' % p.area)
    print(p.area)
```

which prints

```
'''
0.89870
0.00001
0.01629
0.12611
0.05853
0.00995
0.16353
0.03155
0.13710
>
'''
```

What is the order here? The biggest is first, but they are not listed by size. In alphabetical order the islands are

- Ford Island
- Hawaii
- Kaho'olawe
- Kauai
- Lanai
- Maui
- Molokai
- Ni'ihau'
- O'ahu

That doesn't seem to be it either.

Using locations to identify the polygons

It took a few minutes, but I used Google Maps to click on a point in the interior of each island and then copy the longitude and latitude of the point to a dictionary.

```
D = {"Hawaii":[-155.519783,19.625055],
     "Kaho'olawe":[-156.607857,20.550829],
     "Kauai":[-159.567160,22.017814],
     "Lanai":[-156.930387,20.834303],
     "Maui":[-156.279557,20.758340],
     "Molokai":[-156.986996,21.134644],
     "Ni'ihau":[-160.148047,21.904692],
     "O'ahu":[-157.968125,21.488976],
     "Ford Island":[-157.959627,21.363596]}
```

```
def h(row):
    i = row.name[1]
    poly = row['geometry']
    for k in D:
        if poly.contains(Point(D[k])):
            print(i,k)

exp.apply(h,axis=1)
```

How does this work? `df.apply(f,axis=1)` calls `f` for each row of the **exploded** dataframe, each of which holds a Polygon. Crucially, each of these rows has the properties of the parent dataframe (each row *is* a dataframe).

[I am using `h` in the code because I already have `f` and `g` earlier in the script].

So `row['geometry']` gives us each polygon itself. Then we just hunt through the keys to see if any of the localized points lies within the polygon.

We grab the row index as before, just to print it.

This prints

```
0 Hawaii
2 Ni'ihau
3 Kauai
4 Molokai
5 Kaho'olawe
6 Maui
7 Lanai
8 O'ahu
8 Ford Island
```

Index 1 is actually Ford Island (judging by the extremely small area) and where the mark goes if you plot it. But we are not picking up the point I chose, even though O'ahu does. Let's postpone solving that problem for another day.