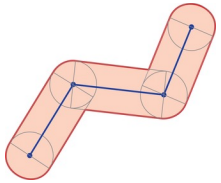
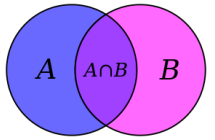
Geospatial Data Science
Content Block II: *Techniques*
Lab 5
Geospatial Queries

Austin J. Brockmeier, Ph.D.

Wednesday, March 9th, 2023

Outline

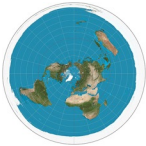




Relevant Scriptable/Programmable Computational Tools:

`Python`: programming language used broadly in data science practice and perfect for GDS

- Packages of relevant code
  - `Numpy` Numerical computing for Python
    - Mathematical functions on arrays (vectors, matrices, tensors) of numbers
  - `Geopandas` Combines GIS with data processing of `Pandas`
    - Coordinate awareness (can pre-project data)
    - https://geopandas.org/en/stable/community/ecosystem.html
  - `Shapely`
    - Compute on planar geometry (does not use geographic distances or elevations only 2D Euclidean space)
  - `Pandas` Python Data Analysis Library specifically for "panel data"
    - spreadsheet-like framework for computing and storing attributes of data in series



Action items

Jupyter notebooks

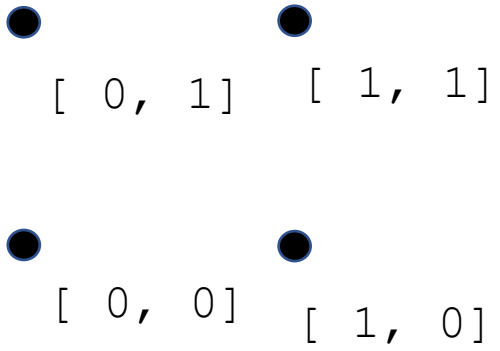
Similar to interactive shell blocks for Mathematica, MATLAB , RStudio etc.

Google's Colaboratory runs Jupyter notebooks on the web

https://colab.research.google.com/

Topics

- NumPy arrays and transpose
- Calculating distance in Numpy
- Shapely geometric objects by coordinates and points
- Calculating distance in Shapely
- Euclidean buffers
- Lists of shapes



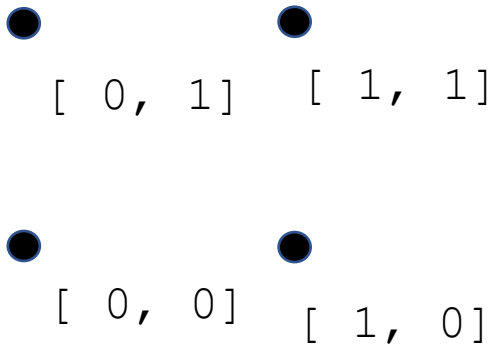
How to compute the distance to the origin for each?

$$\begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \boldsymbol{x}_3 \end{bmatrix} = [\boldsymbol{x}_i]_{i=0}^{3} \in \mathbb{R}^{4\times 2},$$ is a **matrix**

consisting of a series (a linear string) of four 2D vectors

Note: A set of 4 points can define a path/curve, a ring, a polygon, many combinations of the above, or just 4 points!

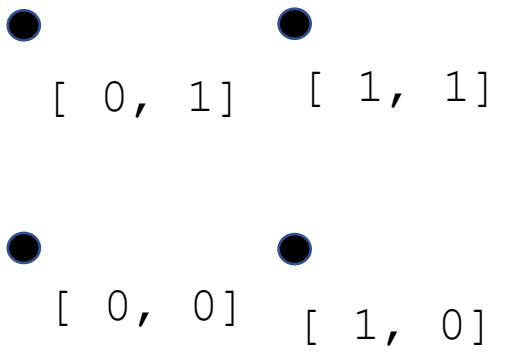
NumPy

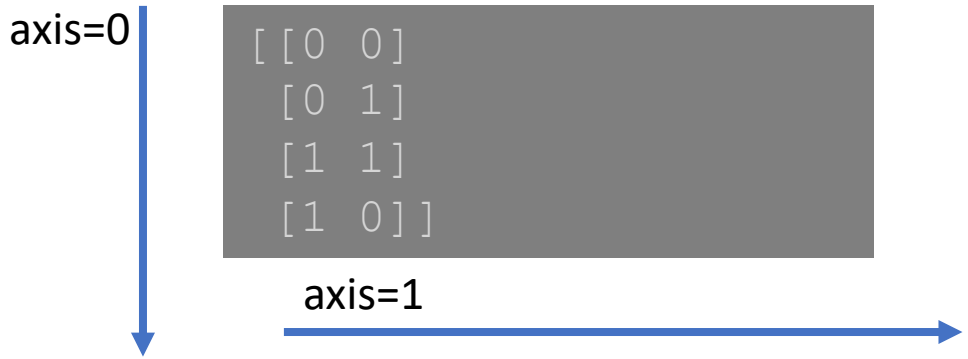



```
[[0 0]
 [0 1]
 [1 1]
 [1 0]]
```

```python
import numpy as np

# define 1 point

x = np.array([0,0])    # [0, 0]
x_coord,y_coord = -0.553, 0.1
x = np.array([x_coord,y_coord]) # [-0.553, 0.1]

# define 4 points
xs = np.array([ [ 0, 0], [0, 1], [1, 1], [1, 0] ])
# how we arrange data (rows or columns) is important
```

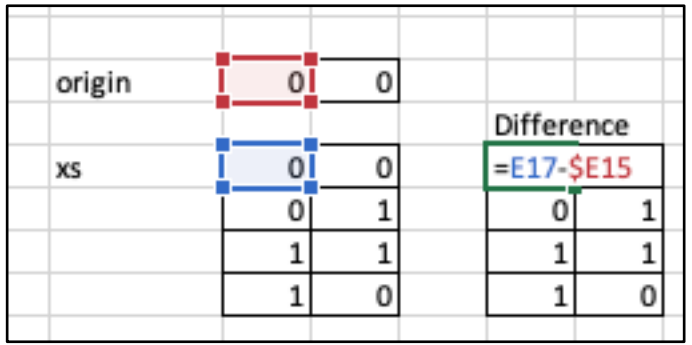
Array functions along axes or an axis

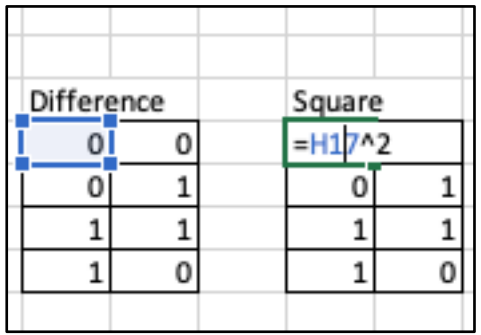


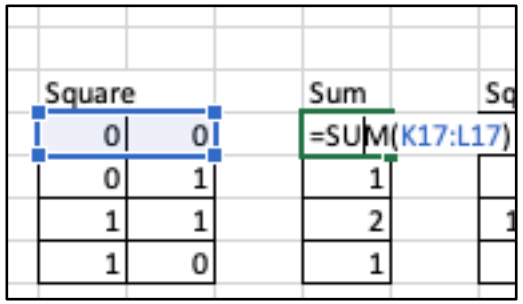


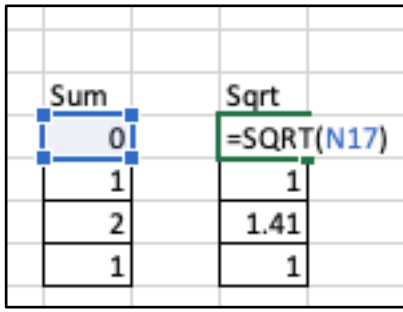

```
xs = np.array([ [ 0, 0], [0, 1], [1, 1], [1, 0] ])
origin = np.array([0,0])   # [0, 0]
# take difference of each coordinate for each point 'broadcast'
dists = np.sqrt( np.sum( (origin - xs)**2, axis=1) )
print(dists)
```

```
[0. 1. 1.41421356 1. ]
```



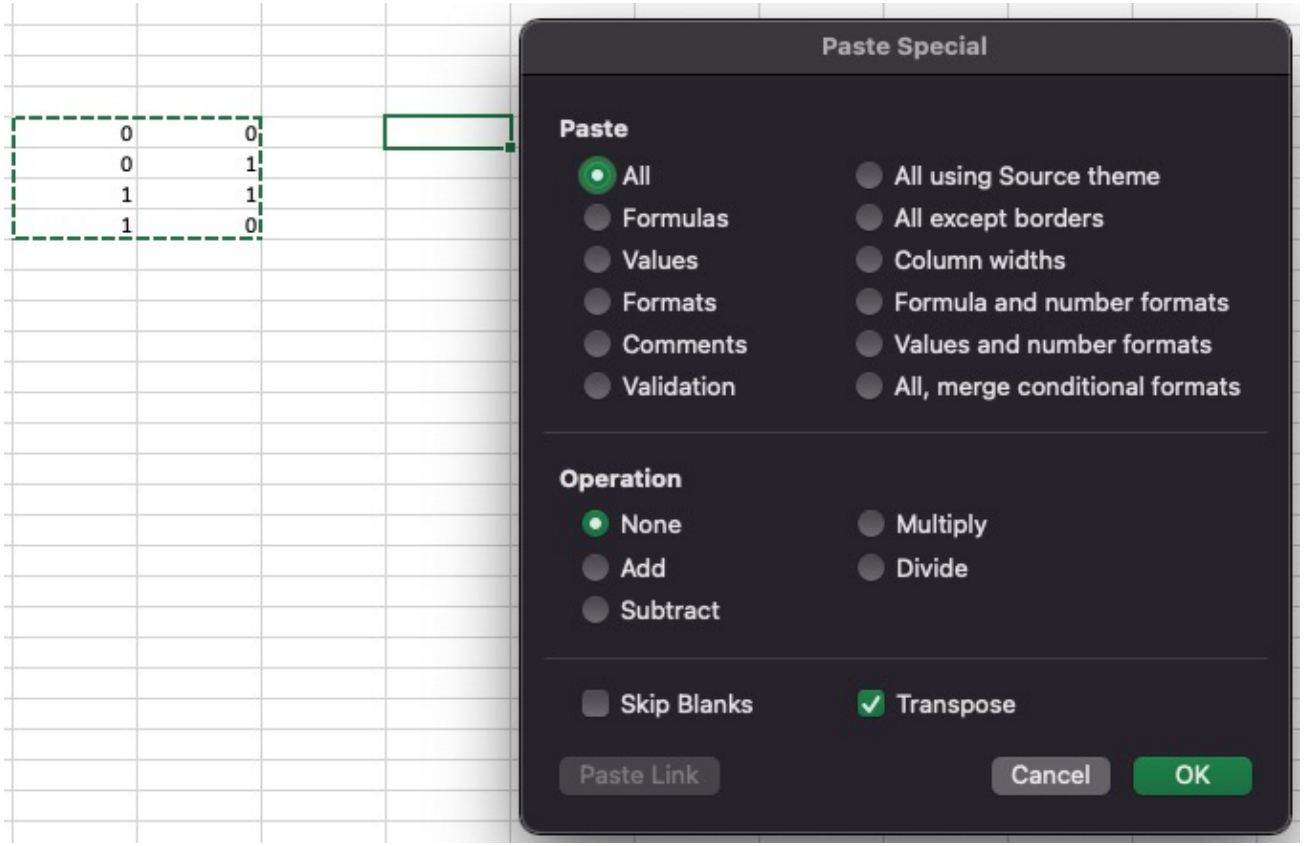






Transpose $\mathbf{T}$

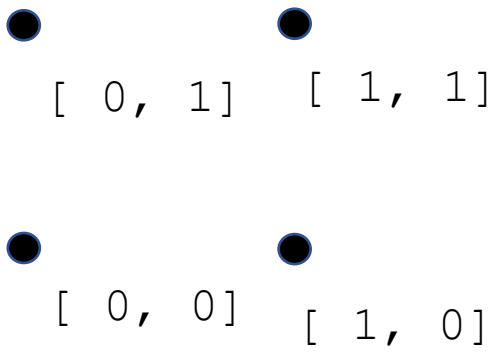
- [0, 1]
- [1, 1]
- [0, 0]
- [1, 0]

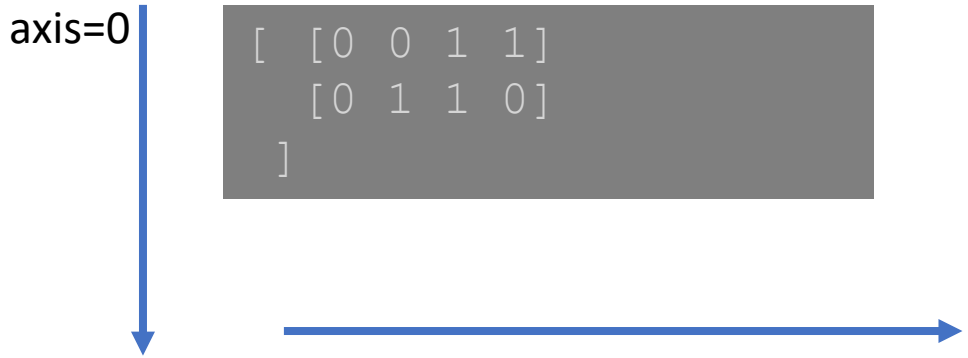


0	0
0	1
1	1
1	0

0	0	1	1
0	1	1	0

Transpose `T`



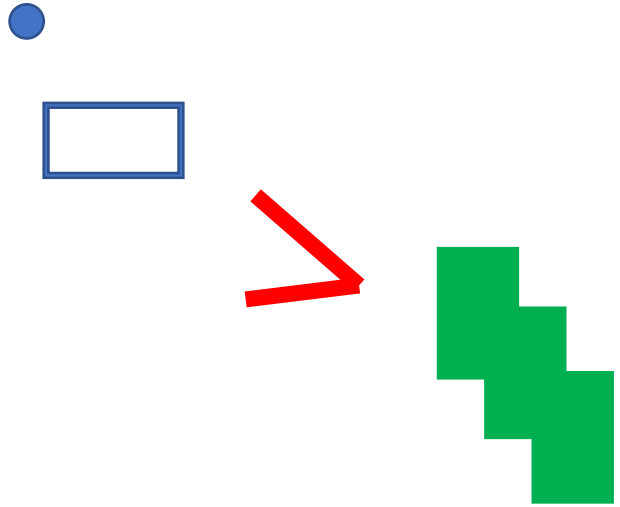



```
xs = np.array([ [ 0, 0], [0, 1], [1, 1], [1, 0] ]).T
# Note  matrix transpose at the end of the statement^
print(xs)
origin = np.array([0,0]).T # cannot transpose, HUGE GOTCHA
origin = origin[:, np.newaxis] # add new dimension
print(origin)
dists2 = np.sqrt(np.sum( (origin - xs)**2, axis=0))
```

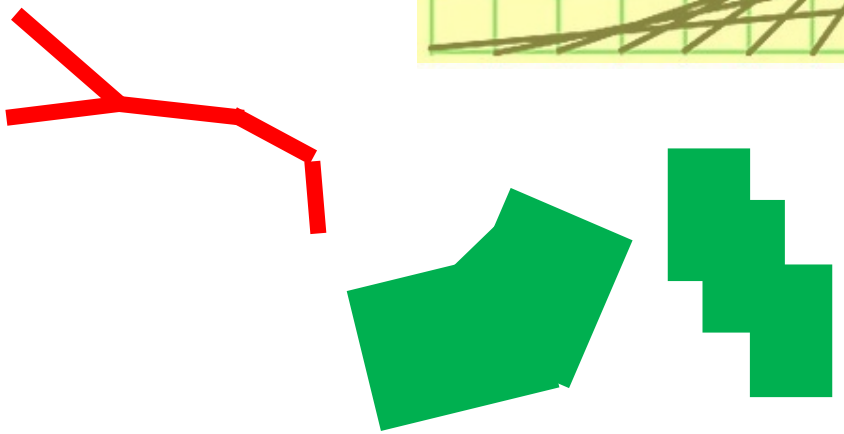
```
[[0 0 1 1] [0 1 1 0]]
[[0]
 [0]]
[0. 1. 1.41421356 1. ]
```

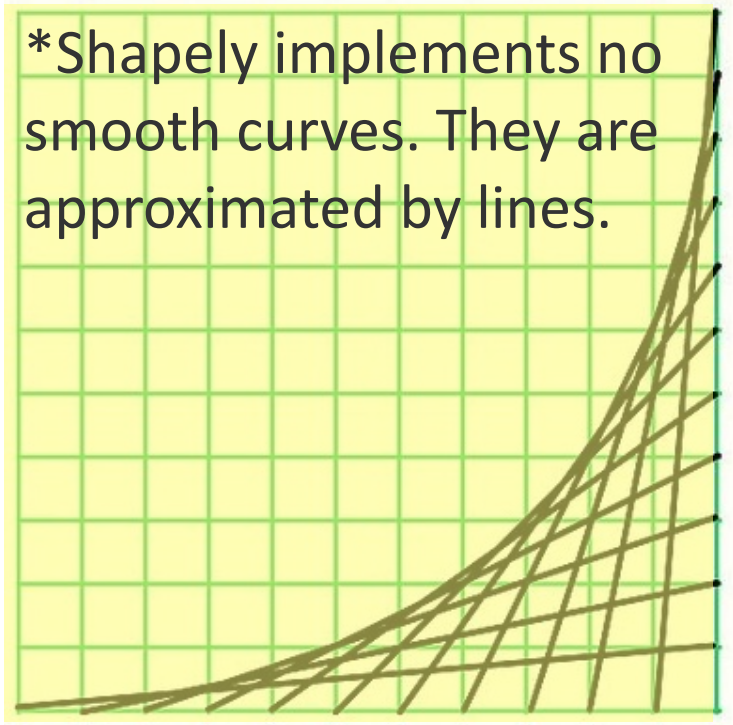
Geospatial sets in a planar/Euclidean space
`Shapely` classes

- Point `Point`
- Ring `LinearRing`
- Curve* `LineString`
- Surface `Polygon`



- Set of points `MultiPoint`
- Set of curves `MultiLineString`
- Set of surfaces `MultiPolygon`





Defining geometric objects in Shapely

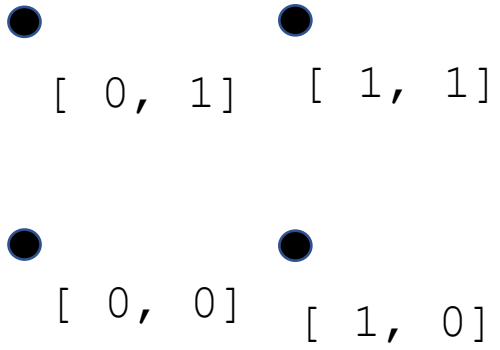
```
import numpy as np
from shapely import Point, LineString, LinearRing, Polygon

xs = np.array([ [ 0, 0], [0, 1], [1, 1], [1, 0] ])
points = [Point(x) for x in xs] # create a list of points
print(points)
```

```
[<POINT (0 0)>, <POINT (0 1)>, <POINT (1 1)>, <POINT (1 0)>]
```

```
path = LineString(xs)
display(path)
path2 = LineString(points)
display(path2)
ring = LinearRing(points)
display(ring)
```

Defining geometric objects in Shapely

```
square = Polygon(points)
display(square)
```



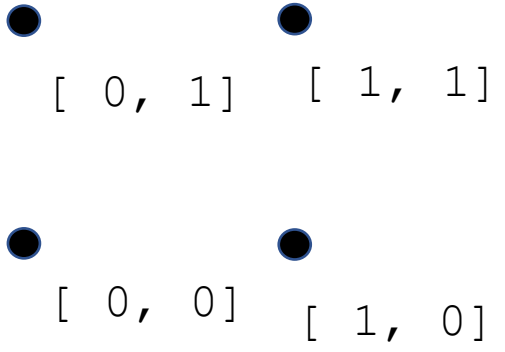
Calculating distances in Shapely

```
import numpy as np
from shapely import Point, LineString, LinearRing, Polygon

xs = np.array([ [ 0, 0], [0, 1], [1, 1], [1, 0] ])
points = [Point(x) for x in xs] # create a list of points
print(points)
```

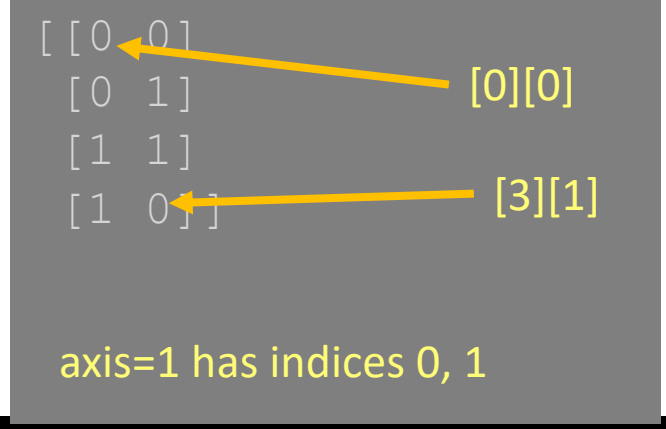
```
[<POINT (0 0)>, <POINT (0 1)>, <POINT (1 1)>, <POINT (1 0)>]
```

```
origin_pt = Point(0,0)
print([origin_pt.distance(x) for x in points])
```

```
[0.0, 1.0, 1.4142135623730951, 1.0]
```



Indexing and getting coordinates from an object in Python




```
point = Point(2,1)
print(list(point.coords))
print(point.coords[0][0])
print(point.coords[0][1])

print(list(path.coords))
print(path.coords[3][:])
print(path.coords[:][0])
print(path.coords[0][:])
print(path.coords[-1][:])
```

```
[(2.0, 1.0)]
2.0
1.0

[(0.0, 0.0), (0.0, 1.0), (1.0, 1.0), (1.0, 0.0)]
(1.0, 0.0)
(0.0, 0.0)
(0.0, 0.0)
(1.0, 0.0)
```

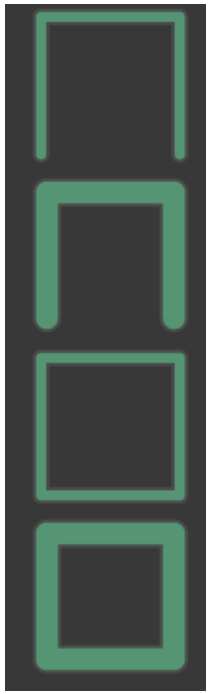
Colon : is short for all the entries

-1 is the last entry on an axis

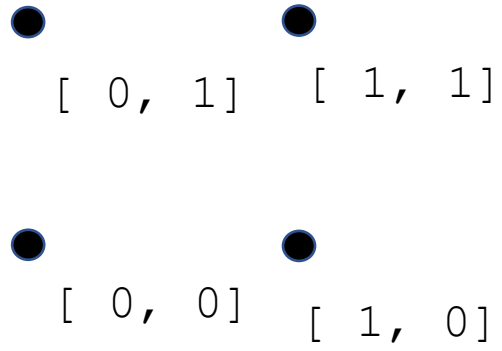
Shapely doesn't know the Earch:
No concept of geodesicis

Defining Euclidean buffers in Shapely

```
display(path.buffer(0.05))
display(path.buffer(0.1))
display(ring.buffer(0.05))
display(ring.buffer(0.1))
```



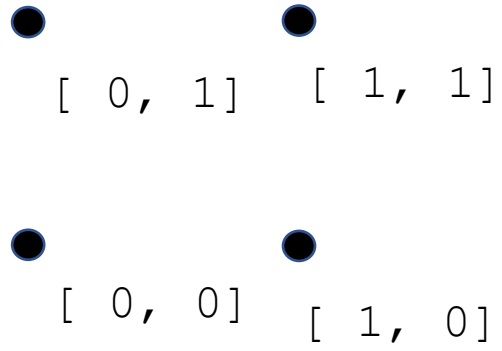
display previews are not to scale!



Defining geometric buffers in Shapely

```
display(square.buffer(0.1))
display(square.buffer(1.1))
```

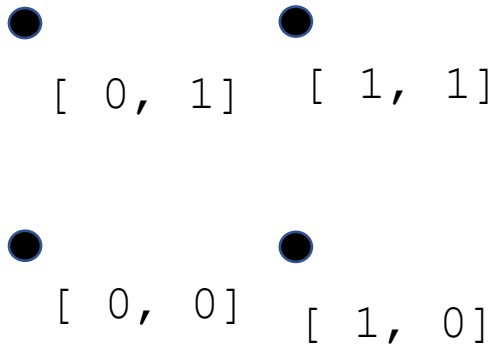
`display` previews are not to scale!
Need to use proper plotting tools!



Computing areas w/ and w/o Euclidean buffers in Shapely

```
print(square.area)
print(square.buffer(0.1).area)
print(ring.area)
print(ring.buffer(0.1).area)
```

```
1.0
1.4313654849054596
0.0
0.7913654849054594
```



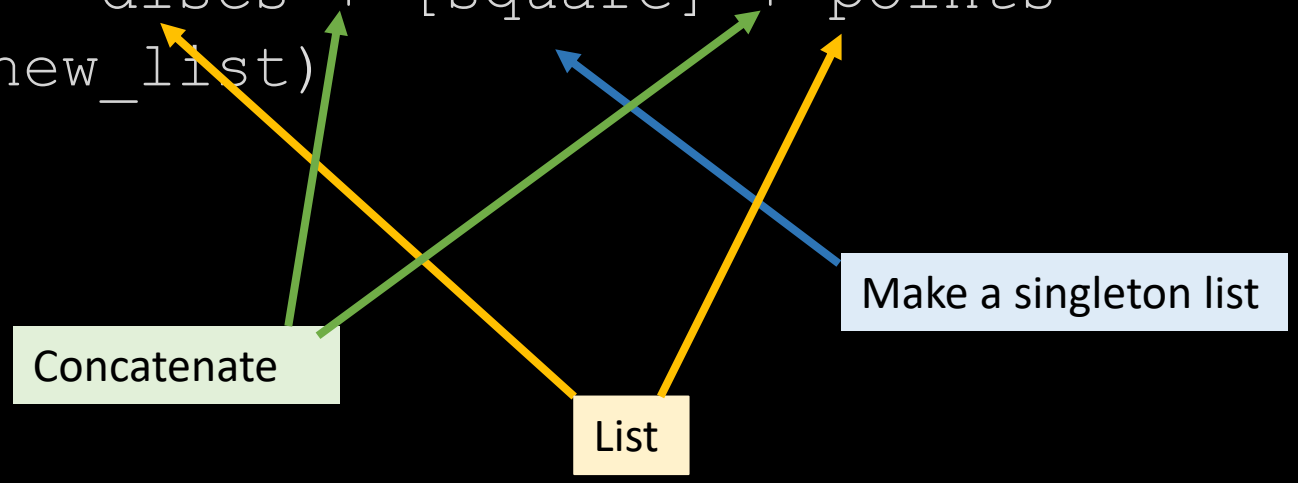
Checking length

```
print(list(path.coords))
print(path.length)

print(list(ring.coords))
print(ring.length)
```

```
[(0.0, 0.0), (0.0, 1.0), (1.0, 1.0), (1.0, 0.0)]
3.0

[(0.0, 0.0), (0.0, 1.0), (1.0, 1.0), (1.0, 0.0), (0.0, 0.0)]
4.0
```

Defining a list of shapes

```
xs = np.array([ [ 0, 0], [0, 1], [1, 1], [1, 0] ])
points = [Point(x) for x in xs] # create a list of points
# create a list of circles (technically discs!) by buffering
discs = [a.buffer(0.25) for a in points ]
new_list = discs + [square] + points
display(new_list)
```




```
[<POLYGON ((0.25 0, 0.249 -0.025, 0.245 -0.049, 0.239 -0.073, 0.231 -0.096, 0...>,
<POLYGON ((0.25 1, 0.249 0.975, 0.245 0.951, 0.239 0.927, 0.231 0.904, 0.22 ...>,
<POLYGON ((1.25 1, 1.249 0.975, 1.245 0.951, 1.239 0.927, 1.231 0.904, 1.22 ...>,
<POLYGON ((1.25 0, 1.249 -0.025, 1.245 -0.049, 1.239 -0.073, 1.231 -0.096, 1...>,
<POLYGON ((0 0, 0 1, 1 1, 1 0, 0 0))>, <POINT (0 0)>, <POINT (0 1)>, <POINT (1 1)>,
<POINT (1 0)>]
```

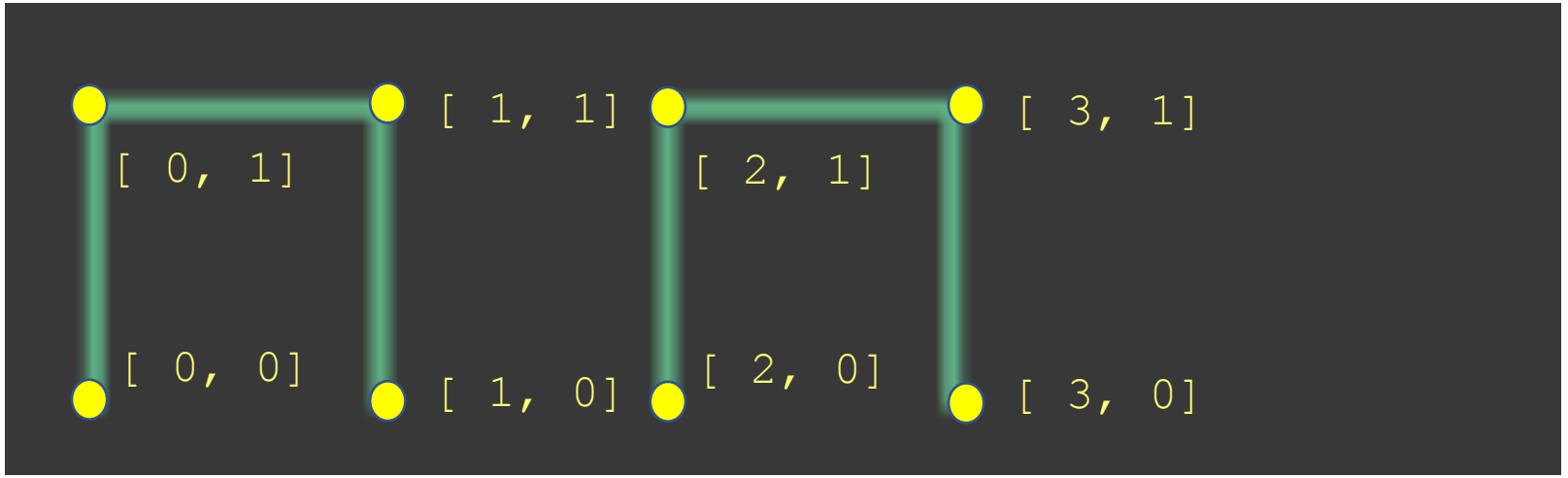
Topics

- Interactions
- Plotting
- Regular grids

Manipulating coordinates (and `union`)

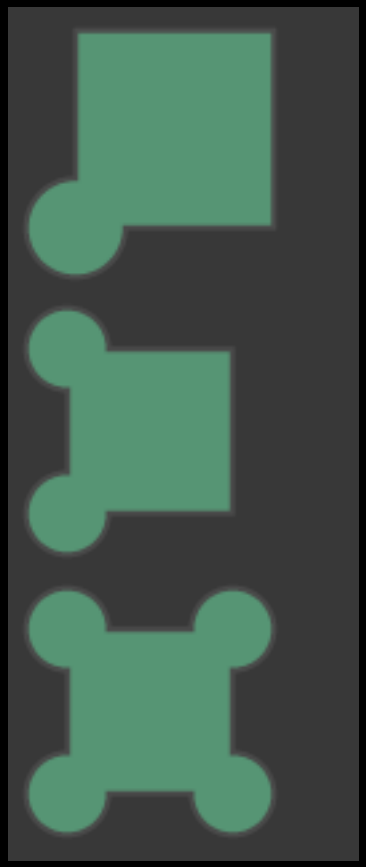
```
path_off = LineString(np.array(path.xy).T + [2, 0 ])
scene1 = path.union(path_off)
print(scene1)
display(scene1)
```

```
MULTILINESTRING ((0 0, 0 1, 1 1, 1 0), (2 0, 2 1, 3 1, 3 0))
```



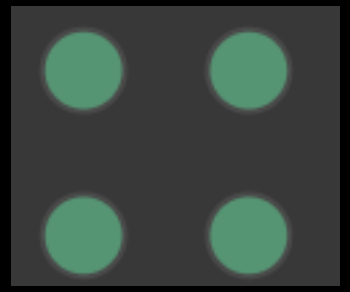
Shapely operations: `union` on polygons

```
xs = np.array([ [ 0, 0], [0, 1], [1, 1], [1, 0] ])
points = [Point(x) for x in xs] # create a list of points

circles = [a.buffer(0.25) for a in points ]

poly1 = square.union(circles[0])
display(poly1)

poly2 = square.union(circles[0]).union(circles[1])
display(poly2)

poly = square
for circ in circles:
  poly = poly.union(circ)

display(poly)
```



Shapely operations: union_all on polygons

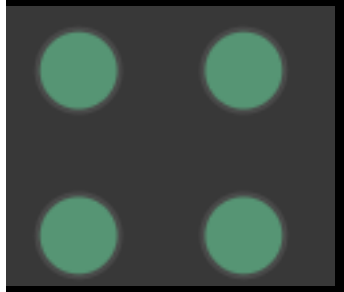
```
from shapely import union_all

circles = [a.buffer(0.25) for a in points ]
poly_new = union_all(circles ) # takes in a list of shapes
display(poly_new)
```



```
poly_new2 = union_all(circles + [square] )
display(poly_new2)
```

Shapely operations: difference on polygons

```
display(poly_new)
```



```
display(square.difference(poly_new))
```

```
display(square.difference(shield))
```

Manipulating coordinates (Polygon's with holes)

Polygon(*shell*[, *holes=None*])

```
small_ring = np.array(ring.xy).T /4
print(small_ring)


frame = Polygon(ring.coords,[small_ring+[0.1, 0.65],
                             small_ring+[0.65, 0.1] ])
```

```
[[0.  0.  ]
 [0.  0.25]
 [0.25 0.25]
 [0.25 0.  ]
 [0.  0.  ]]
```

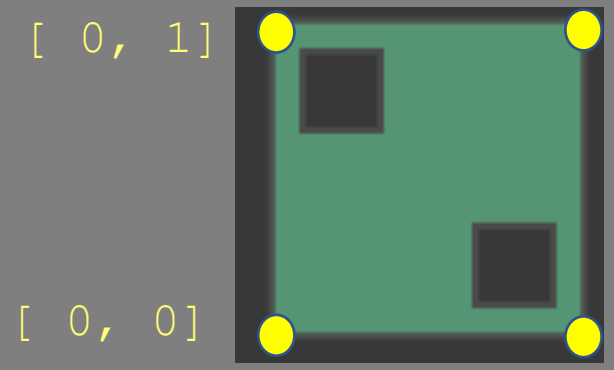
What will this look like?

Manipulating coordinates (Polygon's with holes)

Polygon(*shell*[, *holes=None*])

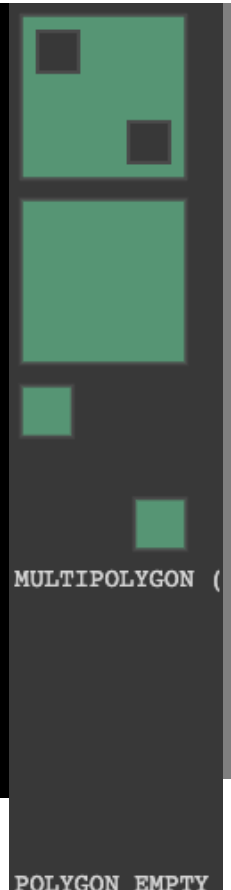
```
small_ring = np.array(ring.xy).T /4
print(small_ring)


frame = Polygon(ring.coords,[small_ring+[0.1, 0.65],
                             small_ring+[0.65, 0.1] ])
```

```
[[0.   0.  ]
 [0.   0.25]
 [0.25 0.25]
 [0.25 0.  ]
 [0.   0.  ]]
```



Shapely operations symmetric `difference` on shapes

```
display(frame)
display(square)

poly = square.difference(frame)
display(poly)
print(poly)

poly2 = frame.difference(square)
display(poly2)
print(poly2)
```



MULTIPOLYGON (((0.1 0.9, 0.35 0.9, 0.35 0.65, 0.1 0.65, 0.1 0.9)), ((0.65 0.35, 0.9 0.35, 0.9 0.1, 0.65 0.1, 0.65 0.35)))

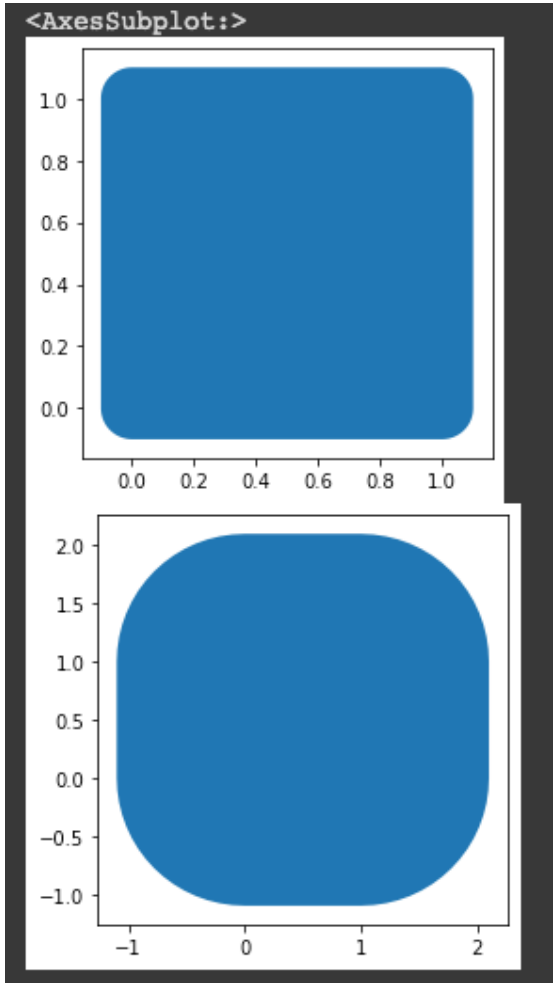
POLYGON EMPTY

Plotting

```
display(square.buffer(0.1))
display(square.buffer(1.1))
```

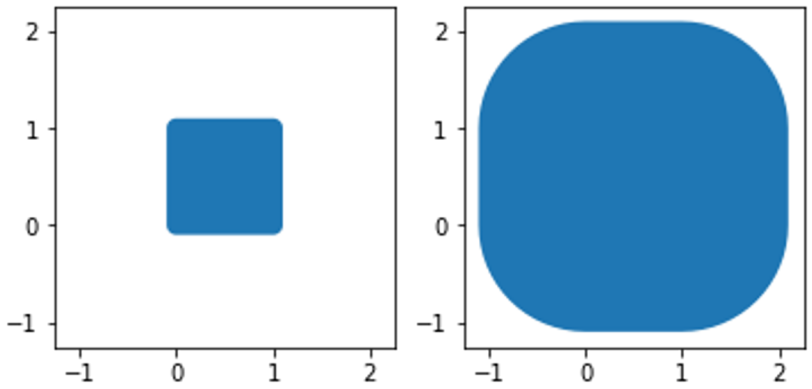
```
!pip install geopandas
```

```
from matplotlib import pyplot as plt
import geopandas as gpd


s1 = gpd.GeoSeries(square.buffer(0.1))
s2 = gpd.GeoSeries(square.buffer(1.1))

s1.plot()
s2.plot()
plt.show()
```



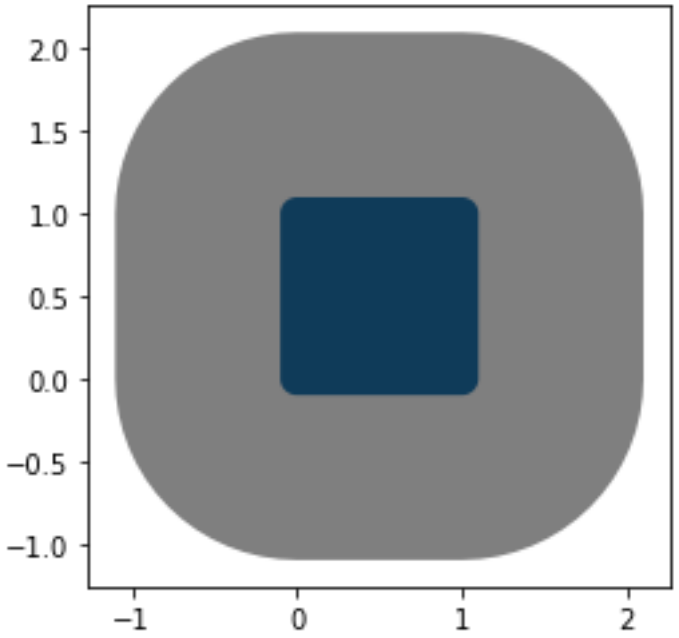
Plotting creating subplots and sharing axes

```
fig = plt.figure()

# Make a figure with two subplots in (horizontal is second axis))
ax1 = fig.add_subplot(1,2,1) # 1 vertical, 2 horizontal, 1st subplot
ax2 = fig.add_subplot(122, sharex=ax1, sharey=ax1)
# 1 vertical, 2 horizontal, 2nd subplot
# Link axes

s1 = gpd.GeoSeries([square.buffer(0.1)])
s2 = gpd.GeoSeries(square.buffer(1.1))

s1.plot(ax=ax1)
s2.plot(ax=ax2)
```



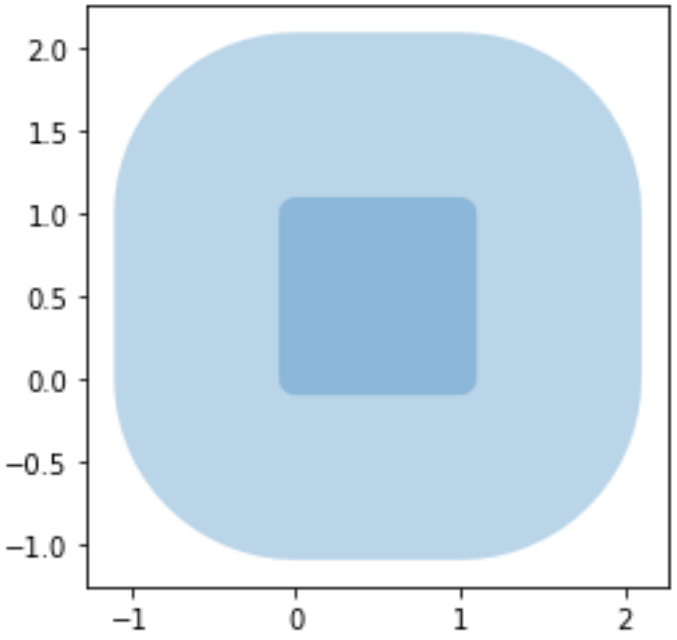
Adding to existing axis and changing color

```
s1 = gpd.GeoSeries(square.buffer(0.1))
s2 = gpd.GeoSeries(square.buffer(1.1))

ax1 = s1.plot()
s2.plot(facecolor='k',alpha=0.5,ax=ax1)
plt.show()
```



Creating one GeoSeries

```
squares = [square.buffer(0.1),square.buffer(1.1)]

s1 = gpd.GeoSeries(squares)
ax1 = s1.plot(alpha=0.3)

plt.show()
```

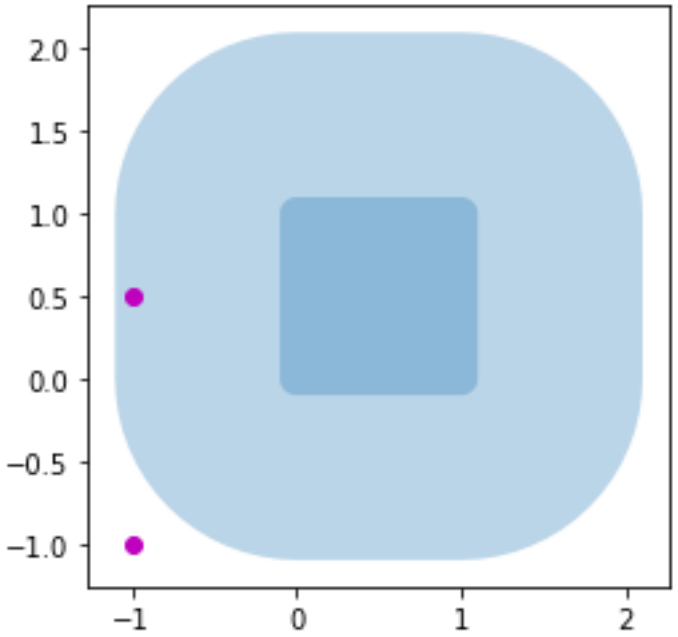


https://geopandas.org/en/stable/docs/reference/api/geopandas.GeoSeries.plot.html

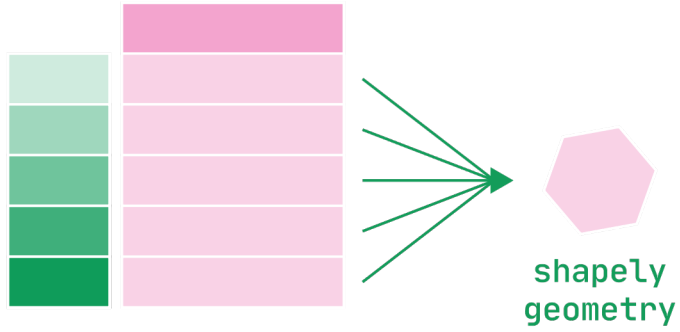
color *str, np.array, pd.Series, List (default None)*If specified, all objects will be colored uniformly.

Calculating distance between `GeoSeries` & an object

```
point1 = Point(-1, 0.5)
point2 = Point(-1, -1)
s2 = gpd.GeoSeries([point1,point2])

ax1 = s1.plot(alpha=0.3)
s2.plot(facecolor='m',ax=ax1)
plt.show()
print(s1.distance(s2[0]))
print(s1.distance(s2[1]))
```




```
0 0.9
1 0.0
dtype: float64

0 1.314214
1 0.314214
dtype: float64
```

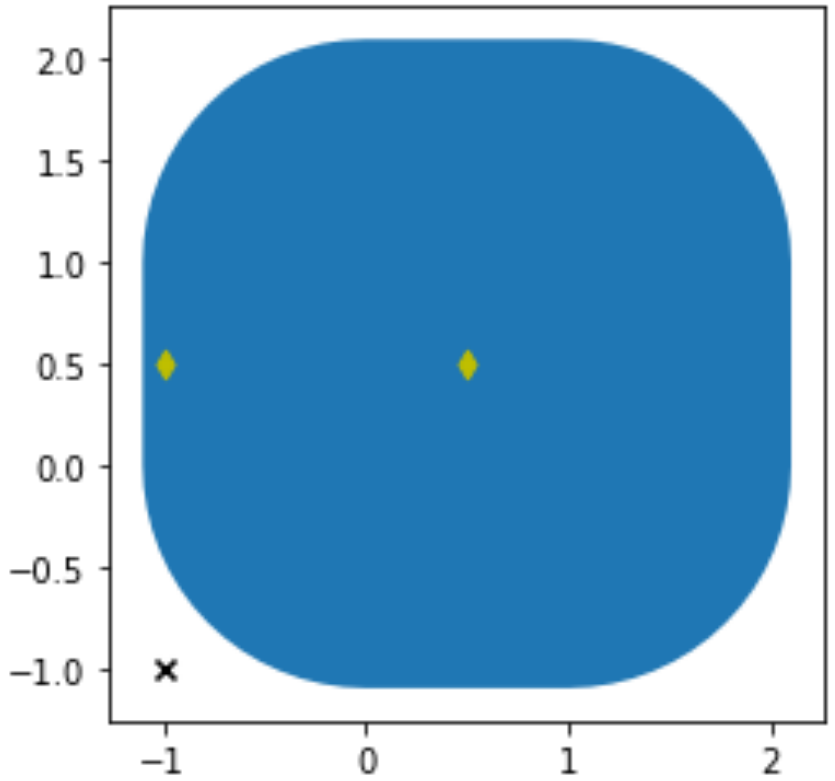


https://geopandas.org/en/stable/docs/reference/api/geopandas.GeoSeries.distance.html

Testing `intersects` for a `GeoSeries` & an object

```
point0 = Point(0.5, 0.5)
point1 = Point(-1, 0.5)
point2 = Point(-1, -1)

s2 = gpd.GeoSeries([point0, point1, point2])


obj = square.buffer(1.1)
tv = s2.intersects(obj)
print(tv)

ax1 = gpd.GeoSeries(obj).plot()
s2[tv].plot(facecolor='y', marker='d', ax=ax1)
s2[tv==False].plot(facecolor='k', marker='x', ax=ax1)
plt.show()
```

```
0 True
1 True
2 False
dtype: bool
```



Exercise 1: Create a plot of overlapping shapes

Create a plot with one square with sides of length 2 centered at the origin (0,0), intersects a circle of radius 1 such that the center of the circle is at a midpoint of one of the square's sides.

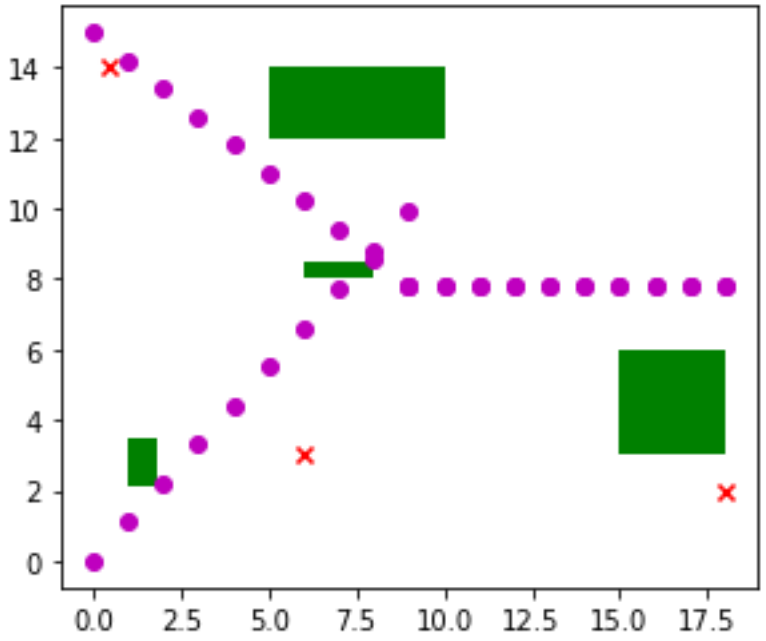
Make sure both the original shapes and intersection are shaded by their transparency.

Calculate the area of the intersection.
Write code that takes an arbitrary input point and calculates the distance to the intersection

Test this code for a set of points both inside of the original shapes, but not in the intersection, and outside the original. Colorcode and change the markers on the the test points based on whether they are in the intersection

Exercise 2: Geospatial queries

Given the lists of spatial objects:
Purple = household
Green = park
Red x = grocery store
Find and plot the subset of households that within a radius of 7 units from any grocery store and 2 miles from any part of a park.

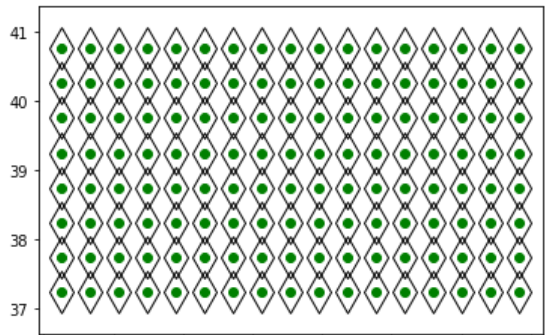


Topics

- Regular grids
- GeoPandas

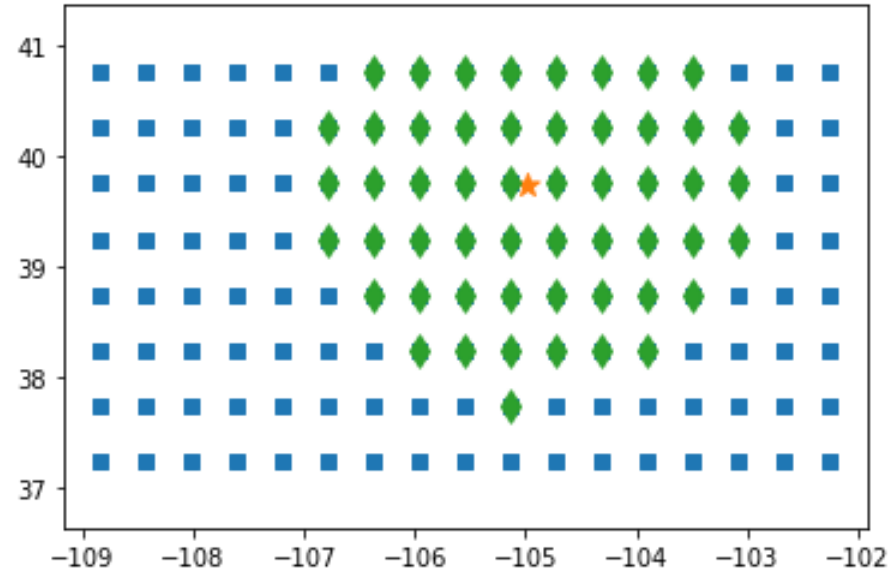
Making a grid using NumPy

```
a = 1/3
lat_steps = 9
lon_steps = 18
lat = np.linspace(37,41,lat_steps)
lat_step = lat[1]-lat[0]
#lat \in [37°N, 41°N] and lon \in [102°02'48"W, 109°02'48"W]}.
lon = np.linspace(102+2/60+48/60/60, 109+2/60+48/60/60, lon_steps)
lon_step = lon[1]-lon[0]
xs, ys, = np.meshgrid(lon[:-1]+lon_step/2,lat[:-1]+lat_step/2)
Xs = [xs,ys]
plt.plot(xs, ys, marker='o', color='g', linestyle='none')
plt.axis('equal')
# show that we have same set of points
plt.plot(Xs[0], Xs[1], marker='d', color='k', markerfacecolor='none', markersize=20,
linestyle='none')
plt.show()
```



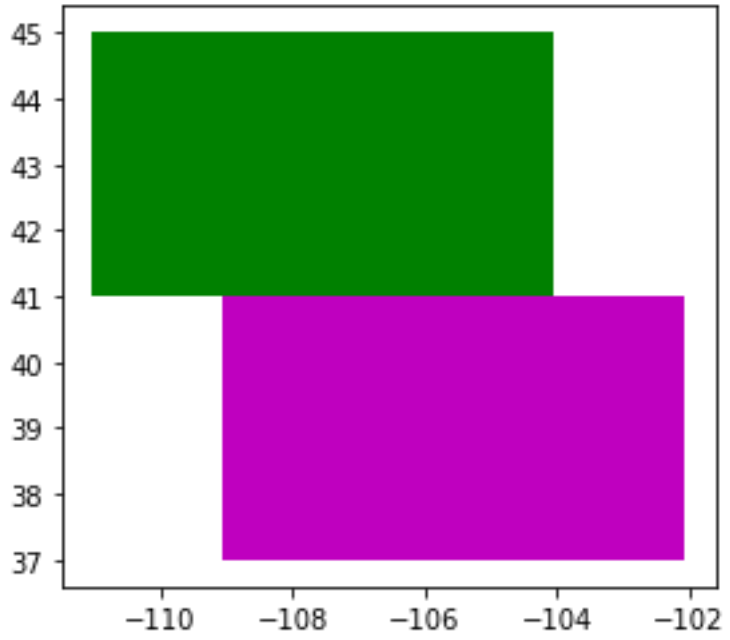
Flatting the coordinates of the lists and making a point `GeoSeries`

```
grid_points = [Point( [-x,y]) for x,y in zip(xs.flatten() , ys.flatten())]

s1 = gpd.GeoSeries(grid_points)
ax1 = s1.plot(marker='s')
capital = [39.7392, -104.9903] # Denver, CO lat and long
capital_loc = Point(capital[::-1])
gpd.GeoSeries(capital_loc).plot(marker='*',markersize=100, ax=ax1)
s1[s1.intersects(capital_loc.buffer(2.0))].plot(marker='d', markersize=90, ax=ax1)

plt.axis('equal')
plt.show()
```



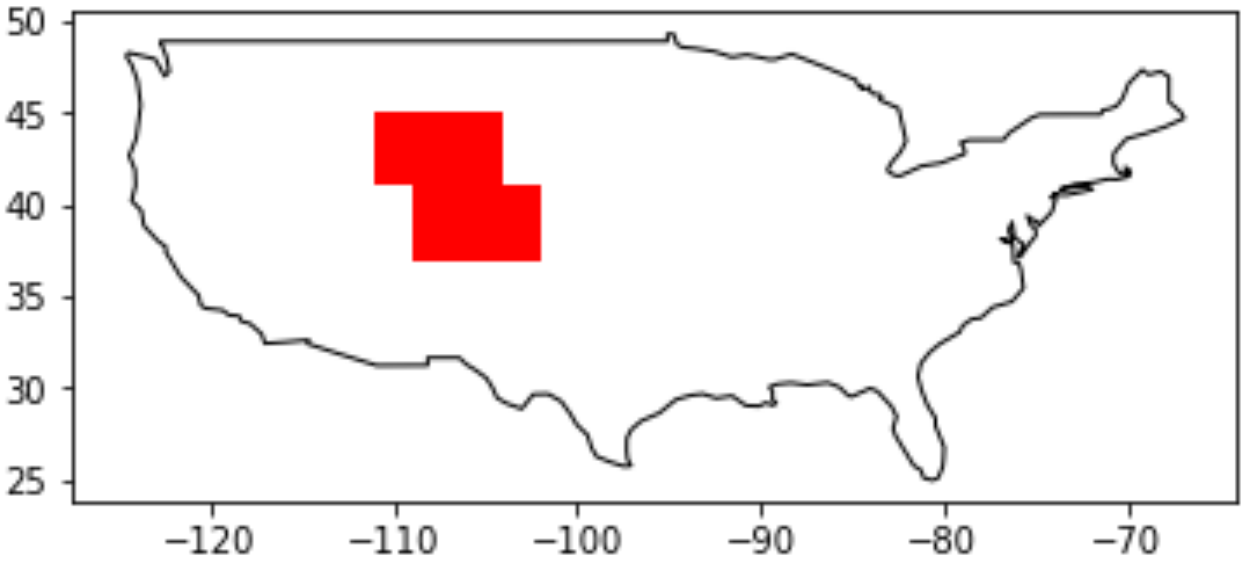
Making a box

```
from shapely.geometry import box
co_lat = [37,41]
co_lon = [102+2/60+48/60/60, 109+2/60+48/60/60]

#Latitude 41°N to 45°N
#Longitude 104°3'W to 111°3'W
wy_lat = [41,45]
wy_lon = [104+3/60, 111+3/60]
#(minx, miny, maxx, maxy, ccw=True)

co = box(-co_lon[0],co_lat[0],-
co_lon[1],co_lat[1])
wy = box(-wy_lon[0],wy_lat[0],-
wy_lon[1],wy_lat[1])

s_co = gpd.GeoSeries(co)
ax1 = s_co.plot(color='m')

s_wy = gpd.GeoSeries(wy)
s_wy.plot(color='g', ax=ax1)
plt.show()
```



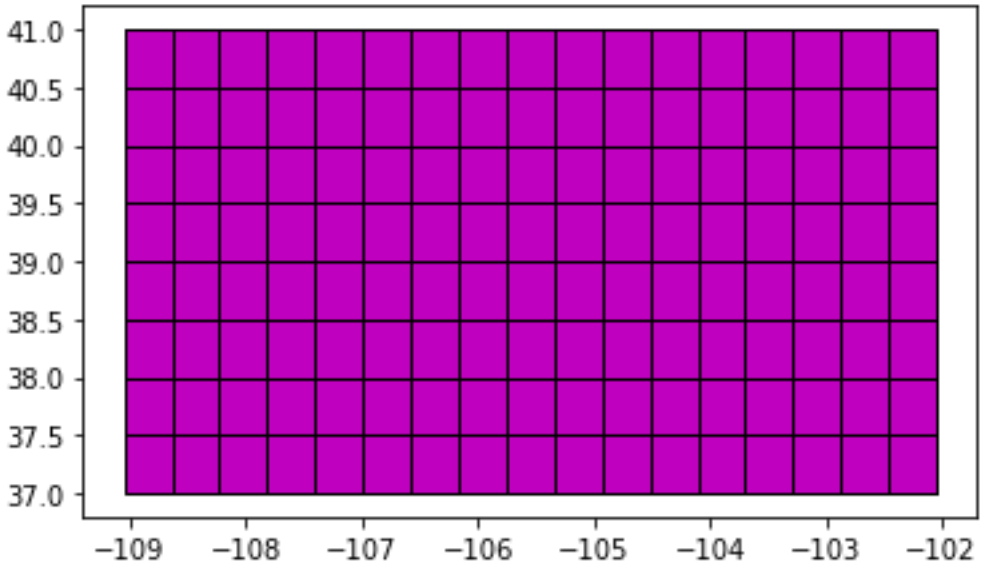
Use low-resolution natural Earth and get lower 48 states for reference

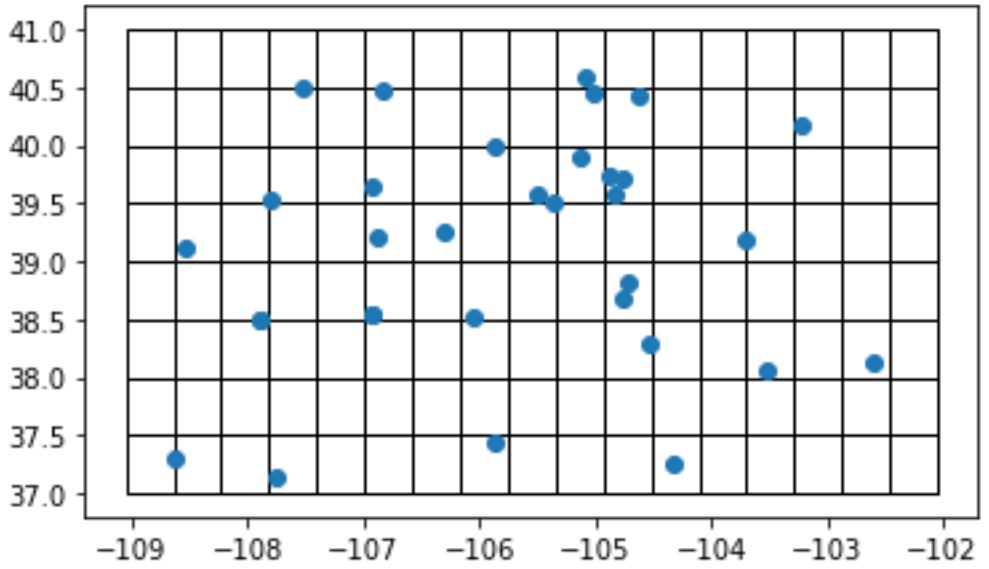
```
world = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))
usa_df = world[world.iso_a3 == 'USA']
l48_geo = list(usa_df['geometry'].iloc[0].geoms)[0] # break out the lower48

ax1 = gpd.GeoSeries(l48_geo).plot(
color='white', edgecolor='black')

co_wy_df = gpd.GeoDataFrame({ 'name': ['CO','WY'], 'geometry': [co,wy] })
co_wy_df.plot(ax=ax1, color='red')
plt.show()
```



Create a series of boxes based on regular grid of Colorado

```
co_mesh_boxes = gpd.GeoSeries([ box(pt.x-lon_step/2, pt.y-lat_step/2,
pt.x+lon_step/2, pt.y+lat_step/2) for pt in grid_points])
ax1 = gpd.GeoSeries(co).plot(color='m')
co_mesh_boxes.plot(color='None', ax=ax1)
plt.show()
# Colorado cities !!!
co_city_array = np.array(co_cities).reshape((-1,2))
s_cities = gpd.GeoSeries( [Point(-x[1],x[0]) for x in co_city_array])
ax1 = co_mesh_boxes.plot(color='None')
s_cities.plot(ax=ax1)

plt.show()
```

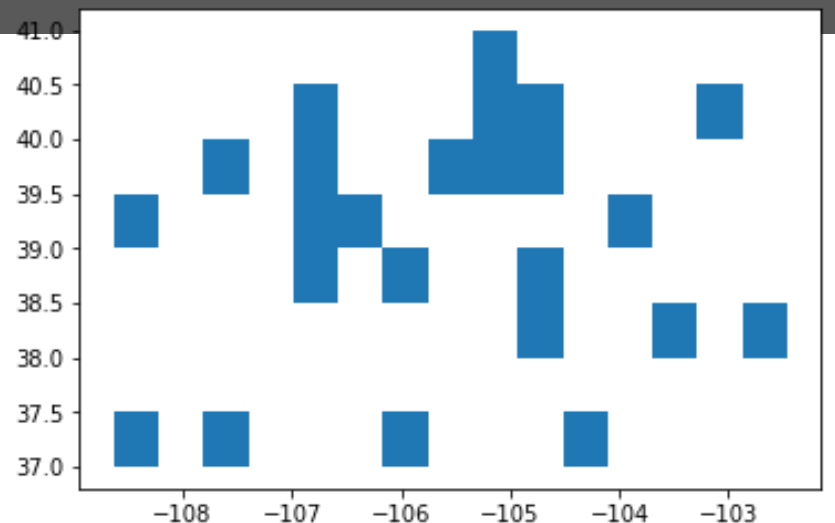




GeoPandas GeoDataFrame

```
city_df = gpd.GeoDataFrame({'geometry': s_cities})
grid_df = gpd.GeoDataFrame({'geometry': co_mesh_boxes })

print(grid_df.head())

sjoin_df = city_df.sjoin(grid_df, how="right", predicate="within")
ax0 = sjoin_df[sjoin_df['index_left'].notnull()].plot()
```

```
0 POLYGON ((-102.04667 37.00000, -102.04667 37.5...
1 POLYGON ((-102.45843 37.00000, -102.45843 37.5...
2 POLYGON ((-102.87020 37.00000, -102.87020 37.5...
3 POLYGON ((-103.28196 37.00000, -103.28196 37.5...
4 POLYGON ((-103.69373 37.00000, -103.69373 37.5...
```



Extra Exercise 3: Countably infinite set

Consider a countably infinite set defined by a point at (0,0), then moving right 1 unit and adding a point at (0,1), then moving up 1 unit and adding a point at (1,1), moving left 2 units and adding a point (-1,1), moving down 3 units and adding a point at (-1,-2), … adding a point along a line 90 degrees from the *k*-th movement and at a distance of (*k*-1)+*k* units.

Write a function to generate the first N point locations.
Call the function for N = 20 and return the results in an array.
Write code to plot an array of 2D points in a 2D scatter plot.

Compute the distance for each point k =1,...,N to the origin

Plot the distances using a stem (or bar plot) where the vertical axis is the distance and the horizontal is the index.

Extra Exercise 4: Count major cities by continent and make a choropleth

https://geopandas.org/en/stable/docs/user_guide/mapping.html
Based on the example for Colorado, move to a larger scale
Count cities per continent and make a choropleth for the counts