Algorithms and Data Structures for Data Science
lab_matplotlib

CS 277

Brad Solomon

February 2, 2024

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science

Learning Objectives

Introduce Python visualization with matplotlib

Practice common plot shapes (dot plot, line plot, bar chart)

Review list manipulations and paired list indexing

Explore fundamentals of math evaluation and numpy

Python Importing

We've seen that Python packages can be big (Pandas ~25 MB)

For many big packages, its better to just import the parts you want

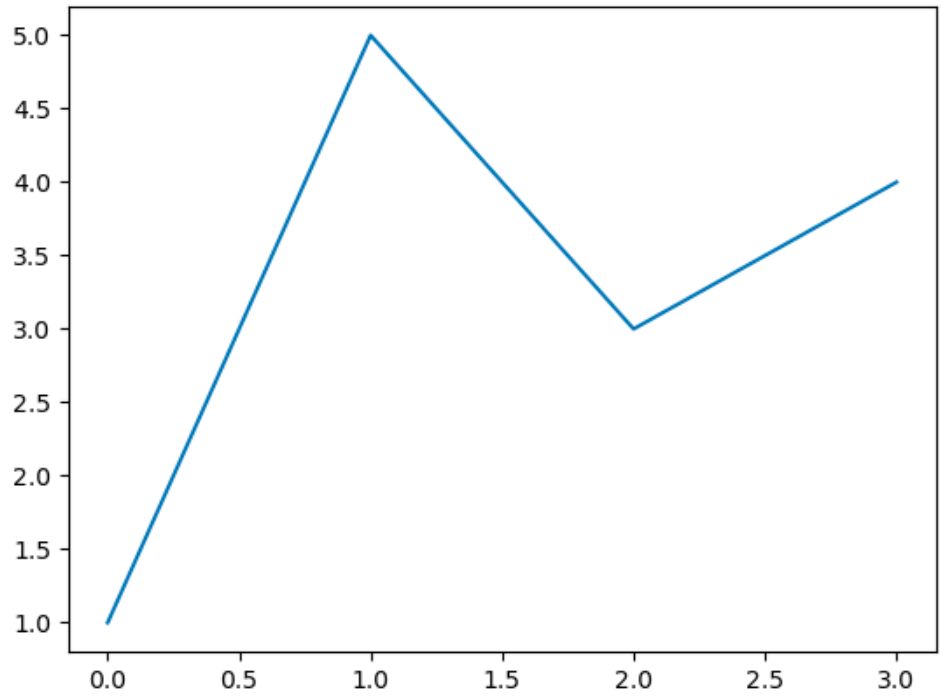
```
import matplotlib.pyplot as plt
```

```
import matplotlib.patches as patches
```

matplotlib.pyplot

```
plt.plot(<single list>)
```

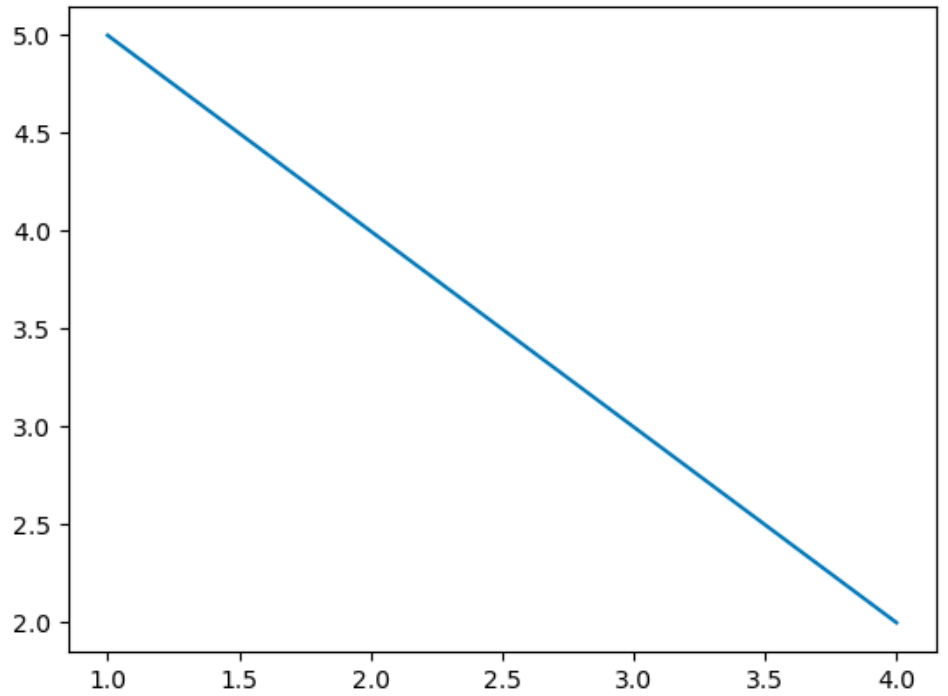
```
plt.plot([1, 5, 3, 4])
```



matplotlib.pyplot

```
plt.plot(<x list>, <y list>)
```

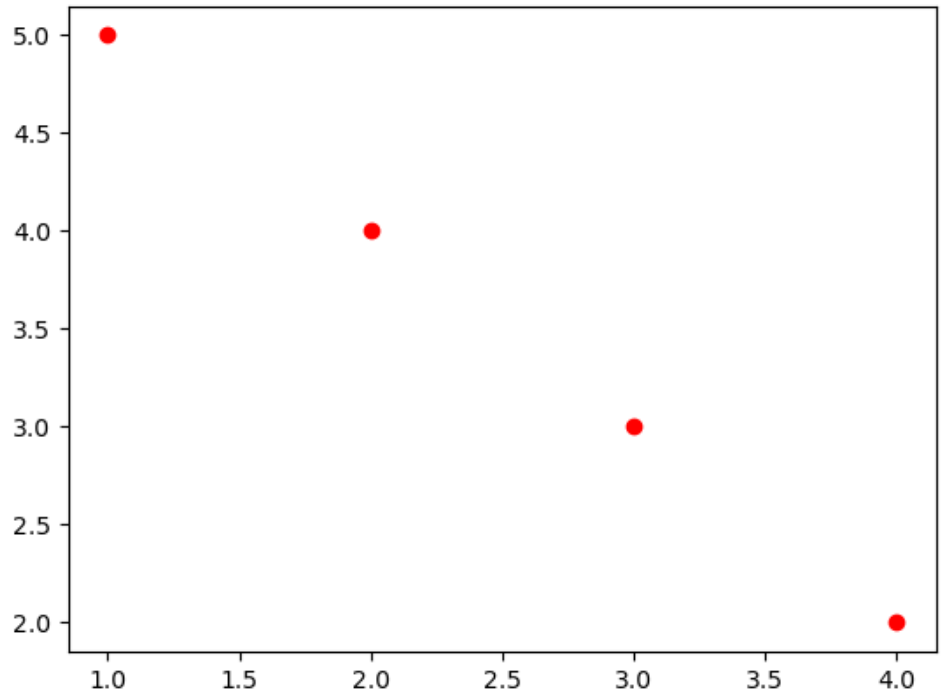
```
plt.plot([1, 2, 3, 4], [5, 4, 3, 2])
```



matplotlib.pyplot

```
plt.plot(<x list>, <y list>, <format>)
```

```
plt.plot([1, 2, 3, 4], [5, 4, 3, 2], 'ro')
```



matplotlib.pyplot

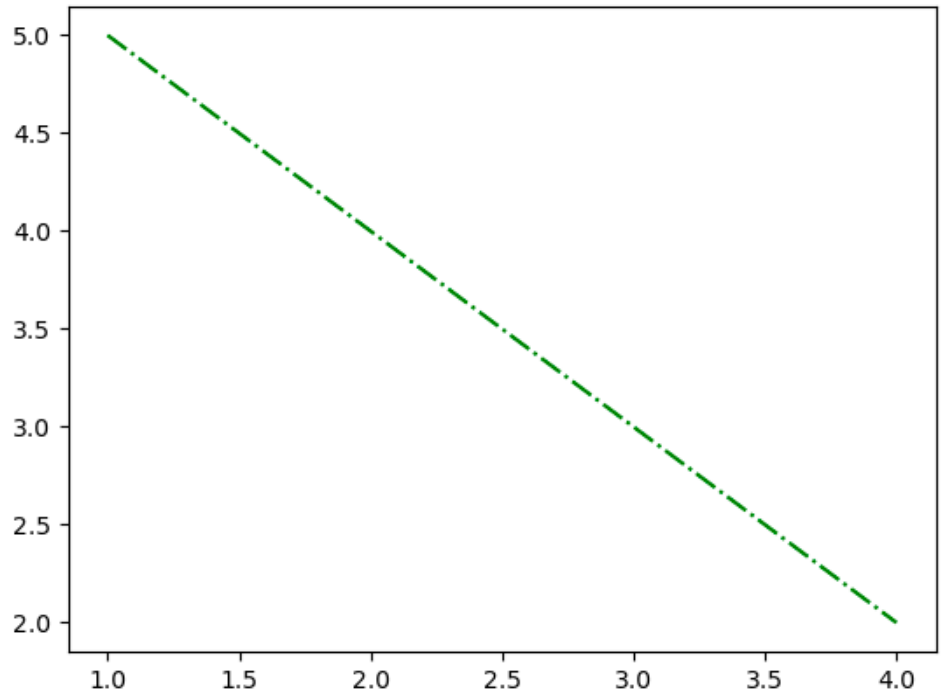
```
plt.plot(<x list>, <y list>, <format>)
```

```
plt.plot([1, 2, 3, 4], [5, 4, 3, 2], '-.g')
```

Char	Marker
.	Point
,	pixel
o	circle
*	star
x	X

Char	Line
-	solid
--	dashed
-.	dash-dot
:	Dotted

Char	Color
b	Blue
g	Green
k	Black
r	Red
c	Cyan



matplotlib.pyplot

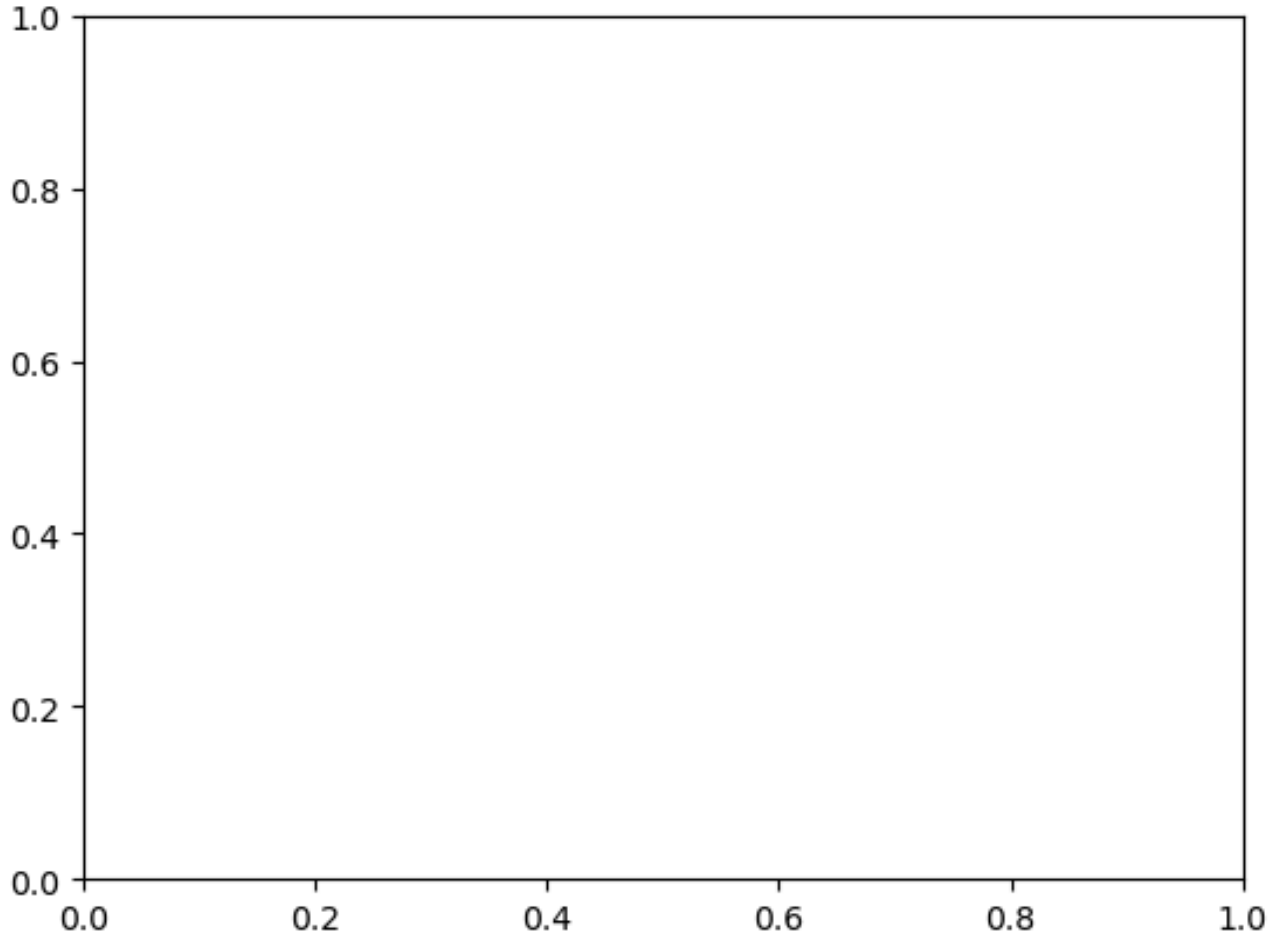
```
plt.plot(*args, scalex=True, scaley=True, data=None, **kwargs)
```

Property	Description
agg_filter	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array and two offsets from the bottom left corner of the image
alpha	scalar or None
animated	bool
antialiased or aa	bool
clip_box	Bbox
clip_on	bool
clip_path	Patch or (Path, Transform) or None
color or c	color
dash_capstyle	CapStyle or {'butt', 'projecting', 'round'}
dash_joinstyle	JoinStyle or {'miter', 'round', 'bevel'}
dashes	sequence of floats (on/off ink in points) or (None, None)
data	(2, N) array or two 1D arrays
drawstyle or ds	{'default', 'steps', 'steps-pre', 'steps-mid', 'steps-post'}, default: 'default'
figure	Figure
fillstyle	{'full', 'left', 'right', 'bottom', 'top', 'none'}
gid	str
in_layout	bool
label	object
linestyle or ls	{'-', '--', '-.', ':', '', (offset, on-off-seq), ...}

And many more!

matplotlib.pyplot.subplots

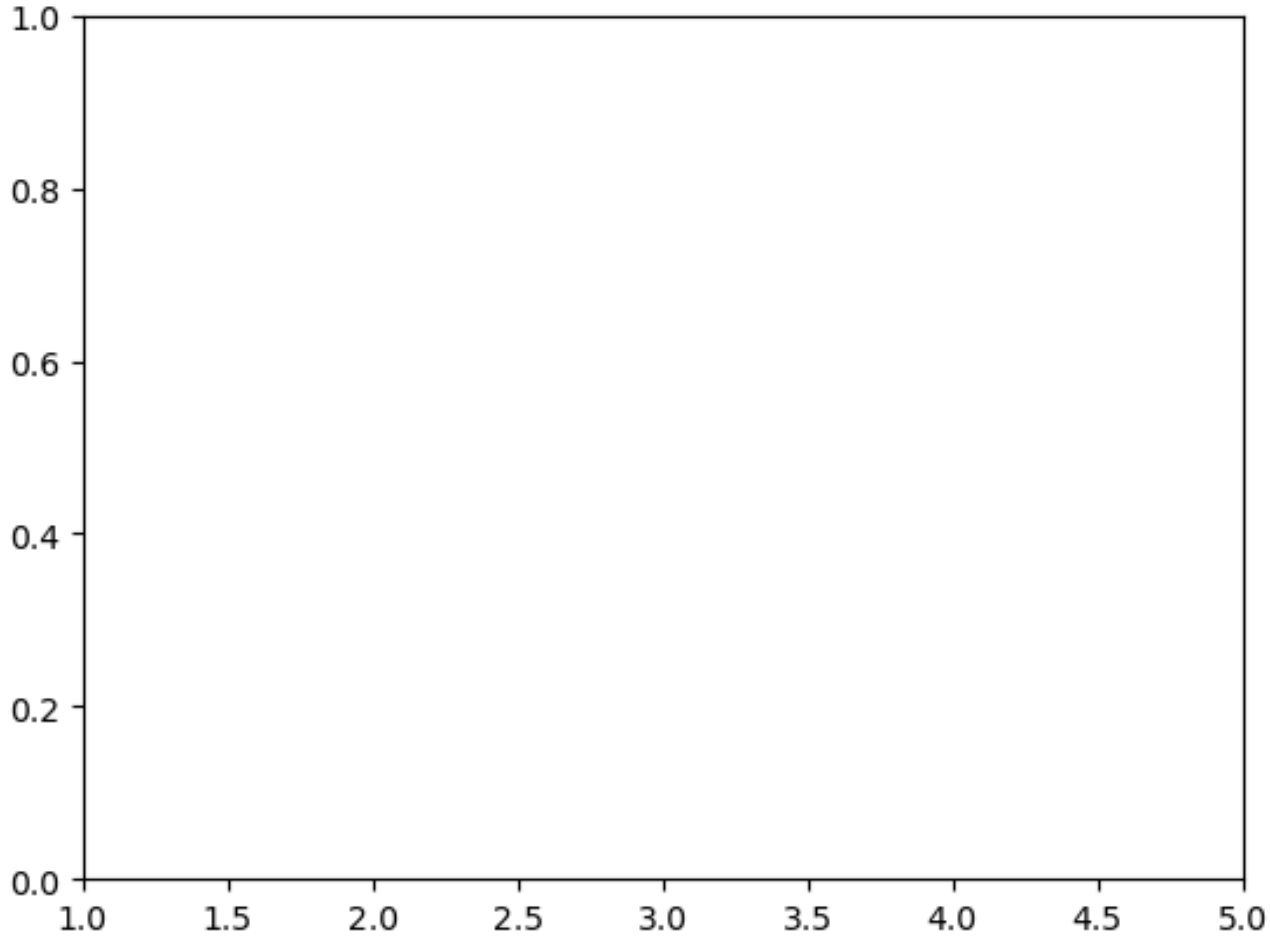
```
fig, ax = plt.subplots()
```



matplotlib.pyplot.subplots

```
fig, ax = plt.subplots()

ax.set_xlim(1, 5)
```



matplotlib.pyplot.subplots

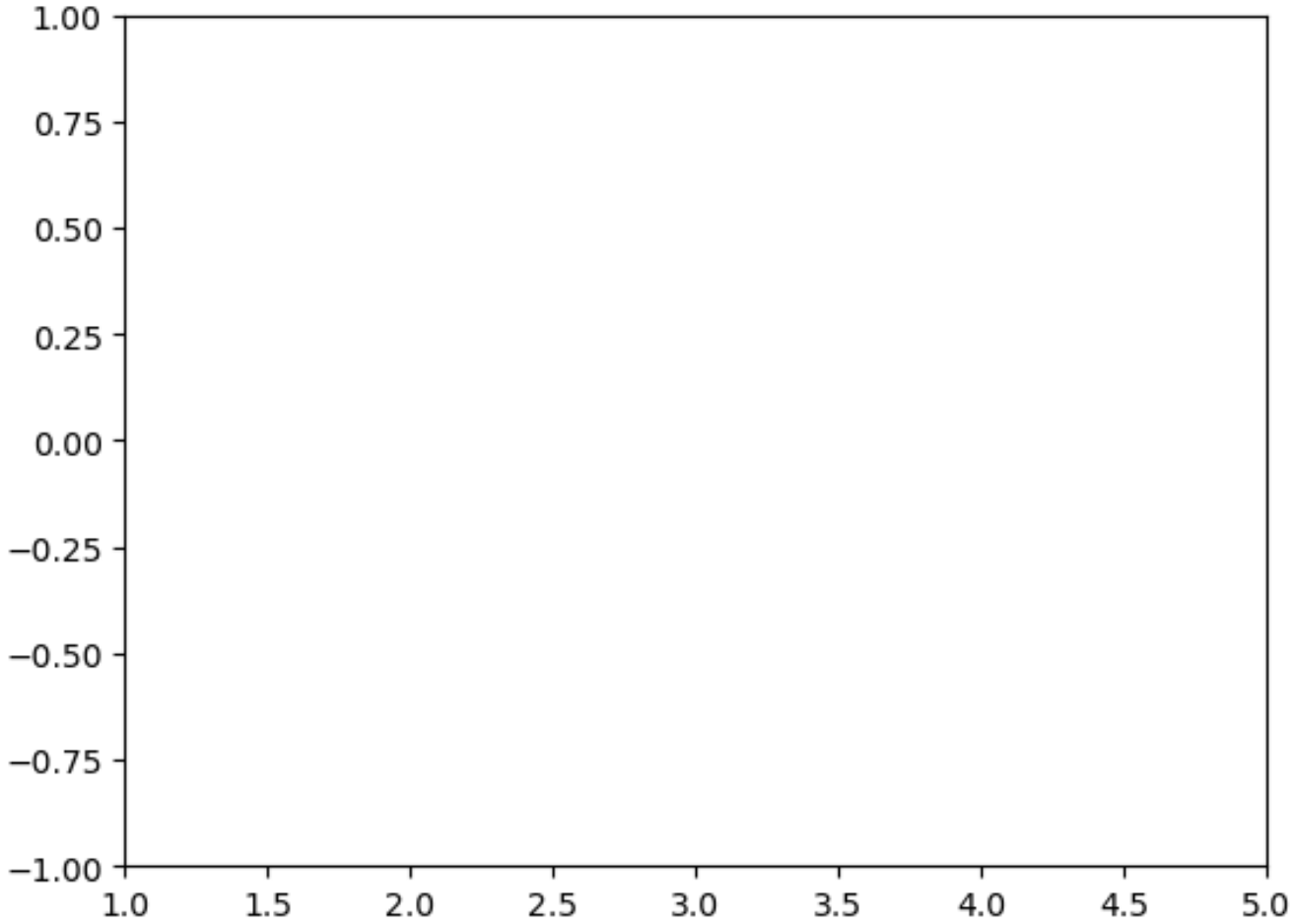
```
fig, ax = plt.subplots()

ax.set_xlim(1, 5)

ax.set_ylim(-1, 1)



ax.set(xlim=(1,5),\
ylim=(-1,1) )
```



NEWSLIDE: matplotlib.savefig()

I forgot to mention during the starting lab how to save a file! It’s just this:

```
plt.savefig(<fname>)
```

matplotlib.patches

What if I want to plot a dataset without drawing lines between them?

matplotlib.patches

```
patches.Circle( (x, y), radius, color )
```

```
patches.Rectangle( (x, y), width, height, color)
```

```
patches.RegularPolygon((x, y), numVertices, radius, color)
```

matplotlib.patches

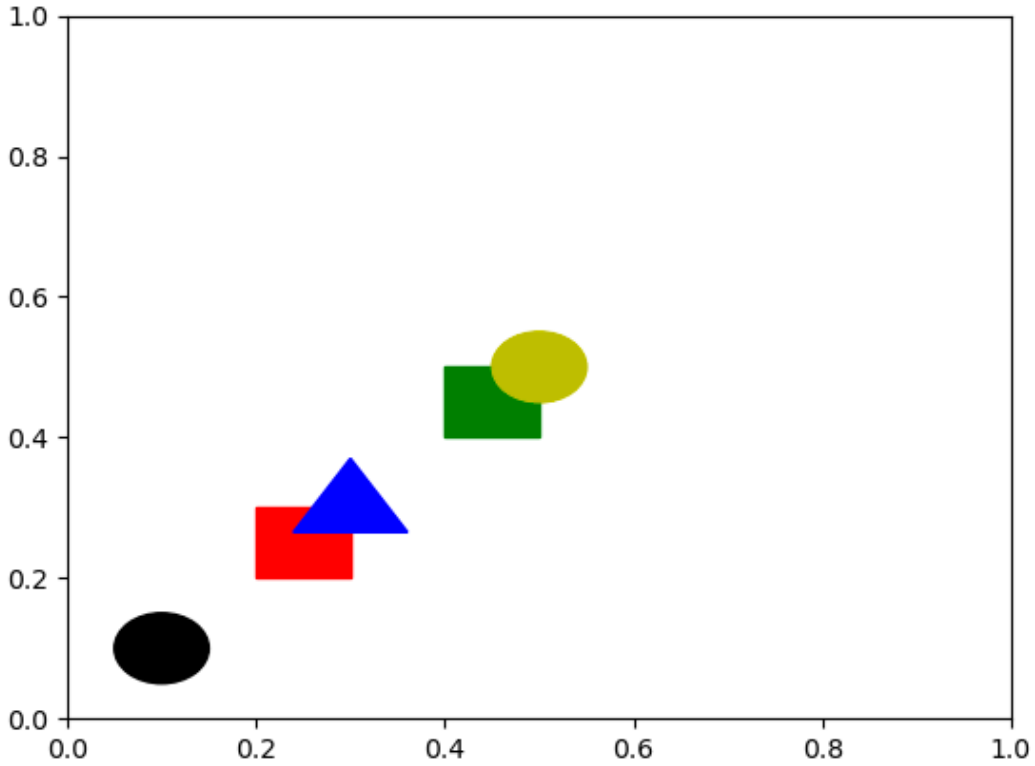
```
fig, ax = plt.subplots()

p=patches.Circle( ... )

ax.add_patch(p)

p=patches.Rectangle( ... )

ax.add_patch(p)
```



numpy

Numpy is meant for scientific computing on large multi-dimensional arrays

Numpy arrays work very differently from Python arrays!

```
 1 import numpy as np
 2
 3 print([1, 2, 3] + [3, 4, 5])
 4
 5
 6 x = np.array([1, 2, 3])
 7 y = np.array([3, 4, 5])
 8
 9
10 print(x+y)
11
12
13 z = np.append(x, y)
14 print(z)
15
16
```

numpy in the context of matplotlib

This assignment will not require you to manipulate numpy lists

That said, many test cases built using `linspace`

```
numpy.linspace(start, stop, num=50)
```

```
numpy.linspace(0, 10, num=11)
```

numpy: practical list builder for matplotlib

```
xList = list(np.linspace(-np.pi, np.pi, 5))
```

[-3.141592653589793, -1.5707963267948966, 0.0, 1.5707963267948966, 3.141592653589793]

```
yList = list(np.sin(xList))
```

[-1.2246467991473532e-16, -1.0, 0.0, 1.0, 1.2246467991473532e-16]

Python eval( ): numpy and list

Given an equation with variables, evaluate expression and return value

```
val = eval(equation)
```

```
1  x = np.linspace(0, 9, 10)
2
3  equation = "2*x+1"
4
5  y = eval(equation)
6
7  print(y)
8
9
```

```
1  x = 1
2
3  equation = "2*x+1"
4
5  y = eval(equation)
6
7  print(y)
8
9
```

matplotlib.animation

A single `plot` produces a **frame.** A gif is nothing more than many frames

```
1  fig, ax = plt.subplots()
2  line, = plt.plot([], [])
3
4  xList = np.linspace(0, 1, 11)
5  yList = np.linspace(0, 1, 11)
6
7  x = []
8  y = []
9
10 def animate(i):
11     x.append(xList[i])
12     y.append(yList[i])
13
14     line.set_data(x, y)
15
16     return line,
17
18 ani = animation.FuncAnimation(fig, animate, frames=len(xList), interval=10, blit=True)
19 ani.save(fname, fps=2)
20
```

matplotlib.animation

A single `plot` produces a **frame.** A gif is nothing more than many frames

```
 1  fig, ax = plt.subplots()
 2  line, = plt.plot([], [])
 3
 4  xList = np.linspace(0, 1, 11)
 5  yList = np.linspace(0, 1, 11)
 6
 7  x = []
 8  y = []
 9
10  def animate(i):
11      x.append(xList[i])
12      y.append(yList[i])
13
14      line.set_data(x, y)
15
16      return line,
17
18  ani = animation.FuncAnimation(fig, animate, frames=len(xList), interval=10, blit=True)
19  ani.save(fname, fps=2)
20
```

matplotlib.animation

A single `plot` produces a **frame.** A gif is nothing more than many frames

```
 1  fig, ax = plt.subplots()
 2  line, = plt.plot([], [])
 3
 4  xList = np.linspace(0, 1, 11)
 5  yList = np.linspace(0, 1, 11)
 6
 7  x = []
 8  y = []
 9
10  def animate(i):
11      x.append(xList[i])
12      y.append(yList[i])
13
14      line.set_data(x, y)
15
16      return line,
17
18  ani = animation.FuncAnimation(fig, animate, frames=len(xList), interval=10, blit=True)
19  ani.save(fname, fps=2)
20
```

Gif Comparison (PDF won't show this)