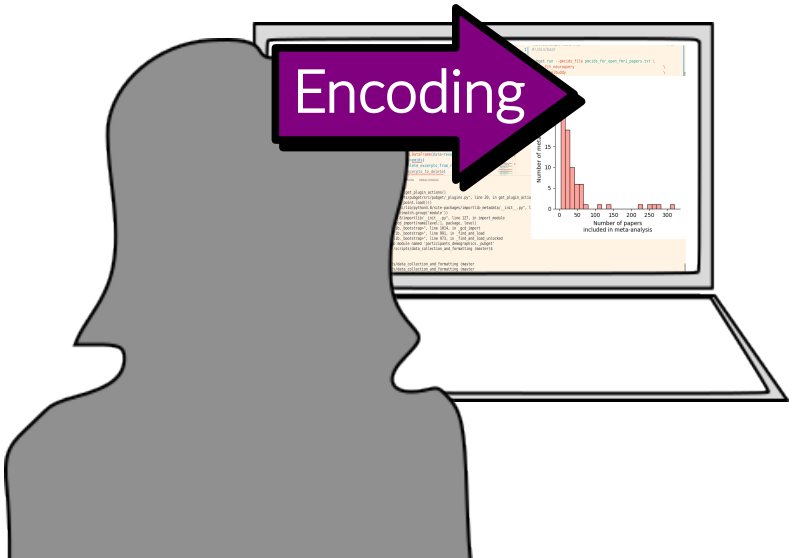
McGill ORIGAMI Lab Fonds de recherche Santé Québec NSERC CRSNG

Introduction to Data Visualization

Part 2: Encoding

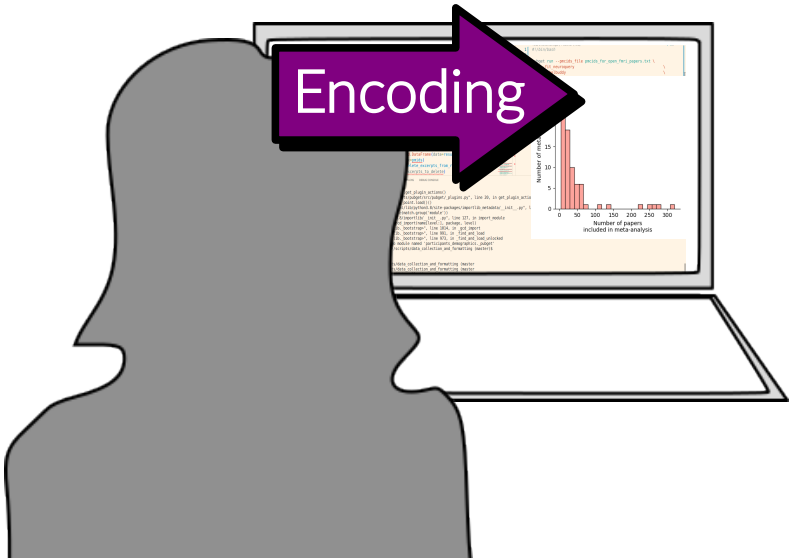


Kendra Oudyk

McGill ORIGAMI Lab Fonds de recherche Santé Québec NSERC CRSNG

Introduction to Data Visualization

Part 2: Encoding



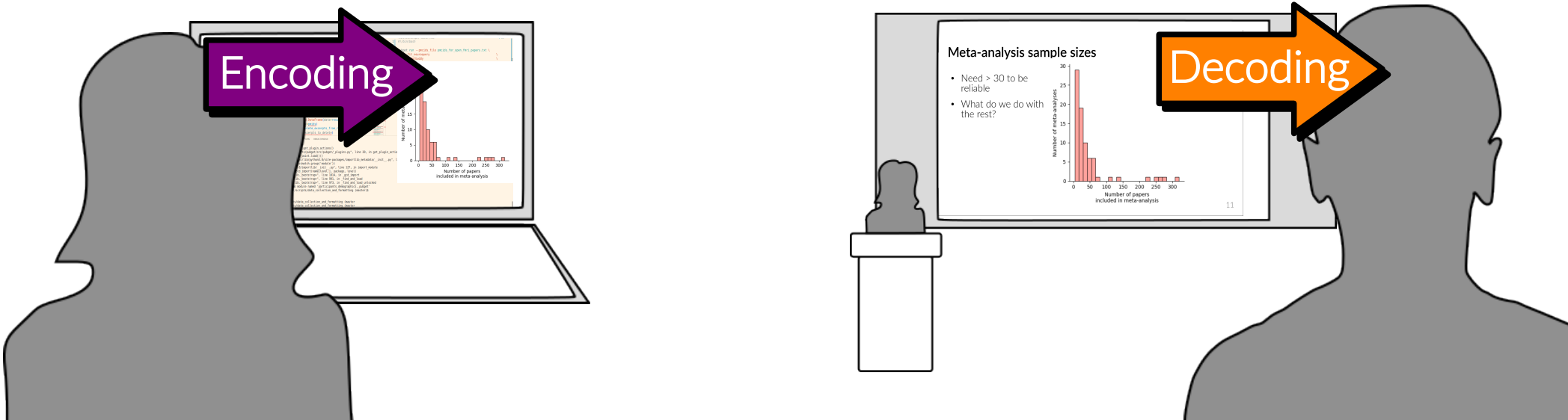
Kendra Oudyk

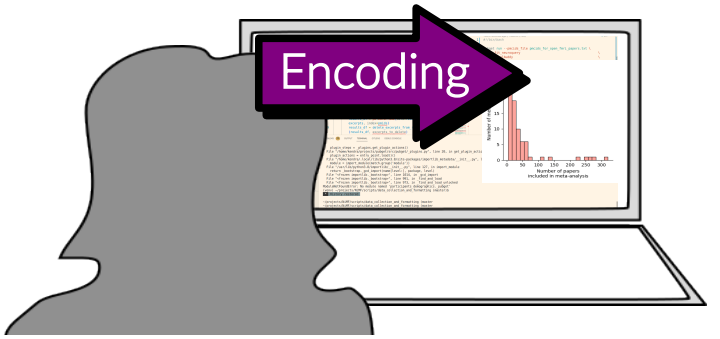
Goal

Use principles of visual encoding and decoding

to efficiently create visualizations

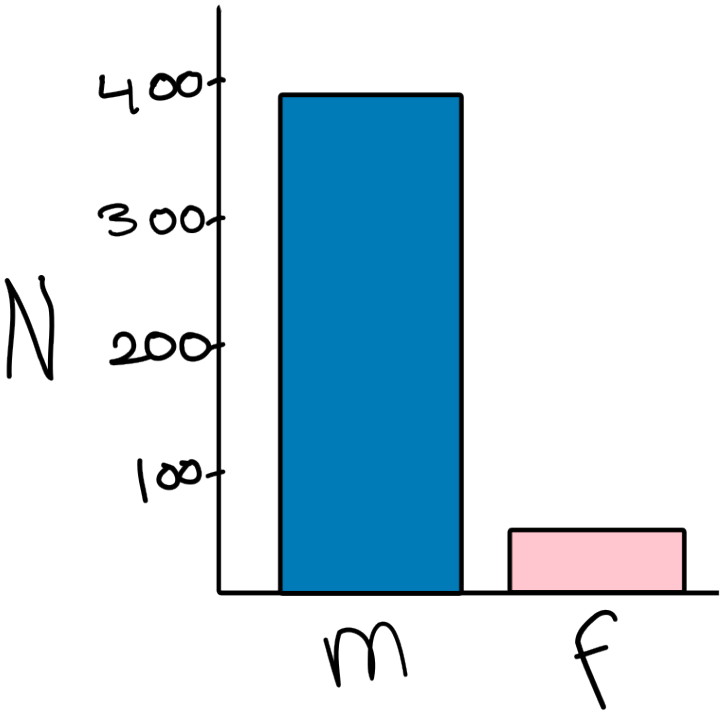
that are effective and reproducible

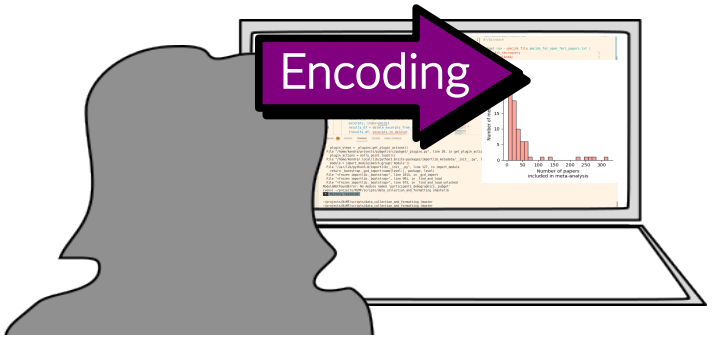




To program a figure efficiently and reproducibly, we should understand

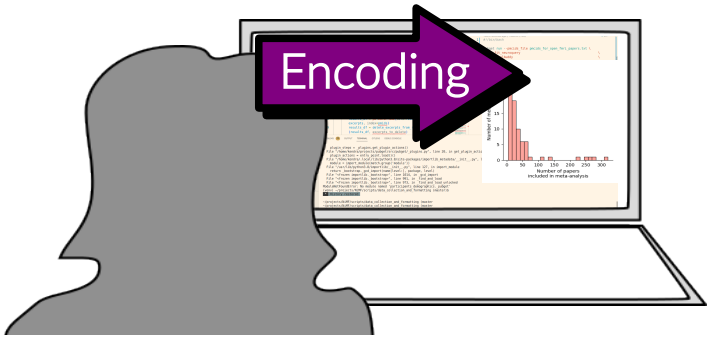
- Some computer graphics
  - How the computer makes figure
- Matplotlib basics
  - What you and matplotlib need to do
- Higher-level libraries
  - More complex visualizations





To program a figure efficiently and reproducibly, we should understand

- Some computer graphics
  - How the computer makes figure
- Matplotlib basics
  - What you and matplotlib need to do
- Higher-level libraries
  - More complex visualizations



To program a figure efficiently and reproducibly, we should understand

- Some computer graphics
  - How the computer makes figure
- Matplotlib basics
  - What you and matplotlib need to do
- Higher-level libraries
  - More complex visualizations

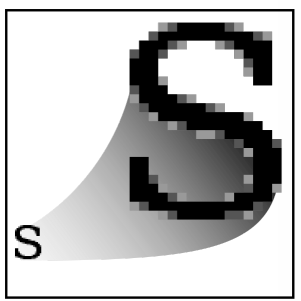
File formats

https://www.pagecloud.com/blog/web-images-png-vs-jpg-vs-gif-vs-svg

File formats

Mathematically-defined shapes

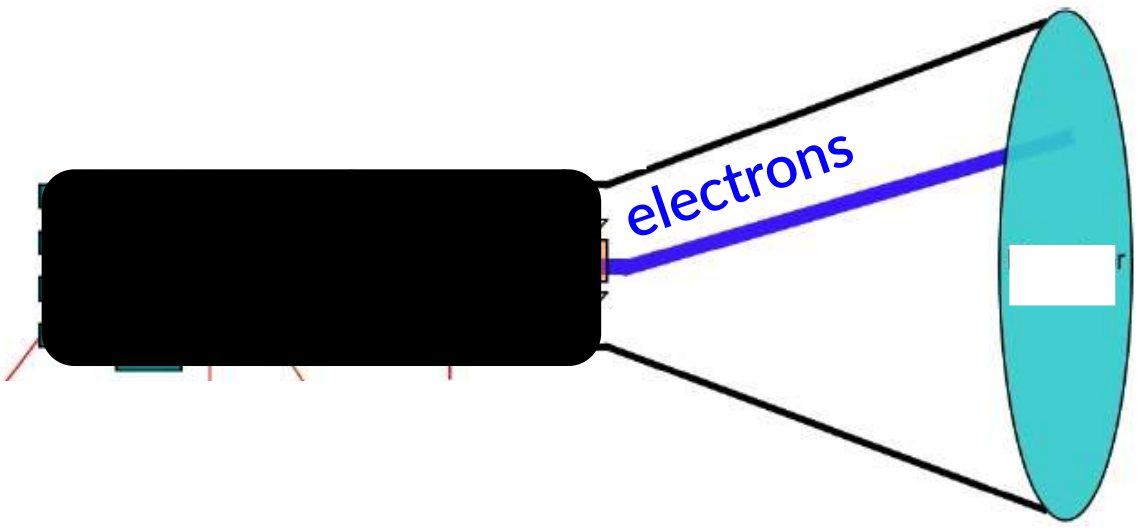
Pixels



	JPG	GIF	PNG	SVG
← VECTOR				✓
← RASTER	✓	✓	✓	

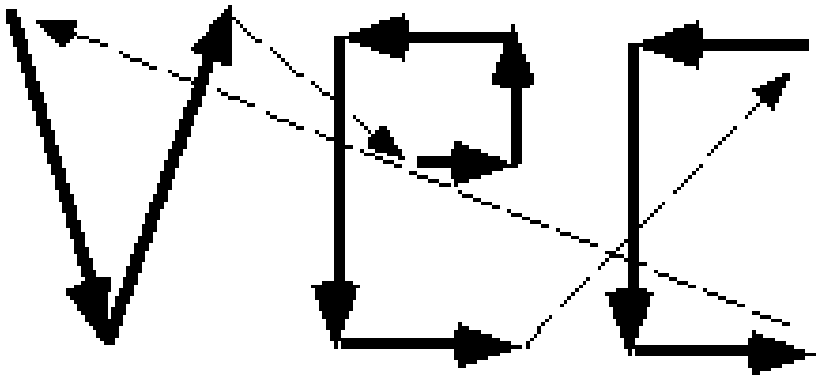
https://www.pagecloud.com/blog/web-images-png-vs-jpg-vs-gif-vs-svg

Cathode ray tube

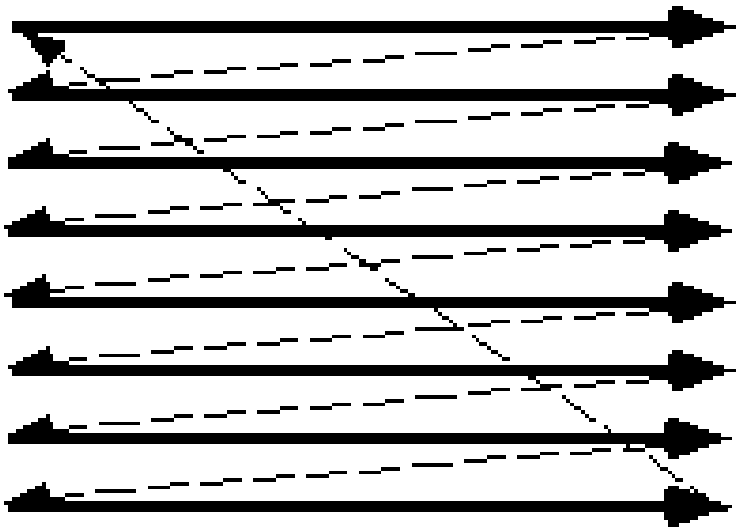


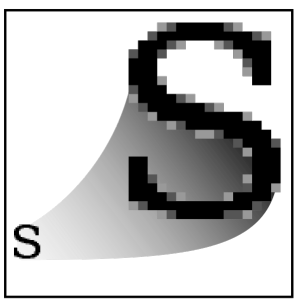
https://www.tutorialspoint.com/computer_graphics/computer_graphics_quick_guide.htm

Vector scan



Raster scan





https://www.cs.uic.edu/~jbell/CourseNotes/ComputerGraphics/VectorVsRaster.html

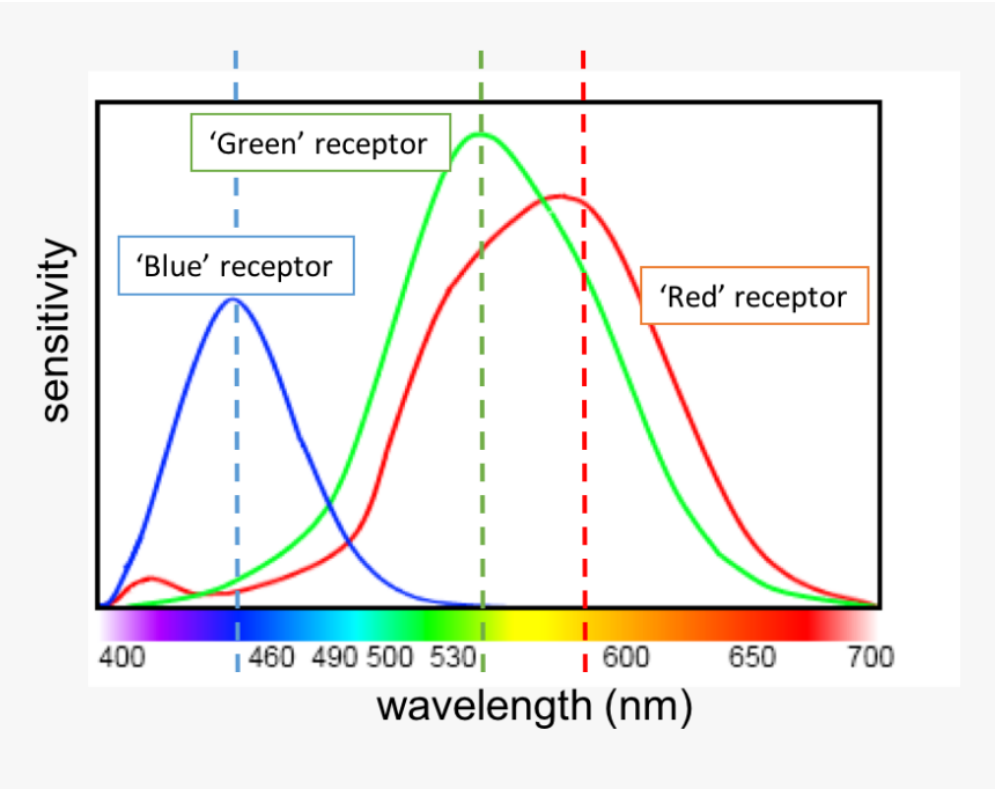
File formats

	JPG	GIF	PNG	SVG
VECTOR				✓
RASTER	✓	✓	✓	
TRANSPARENCY			✓	✓
ANIMATION		✓	✓	✓
LOSSY	✓			

https://www.pagecloud.com/blog/web-images-png-vs-jpg-vs-gif-vs-svg

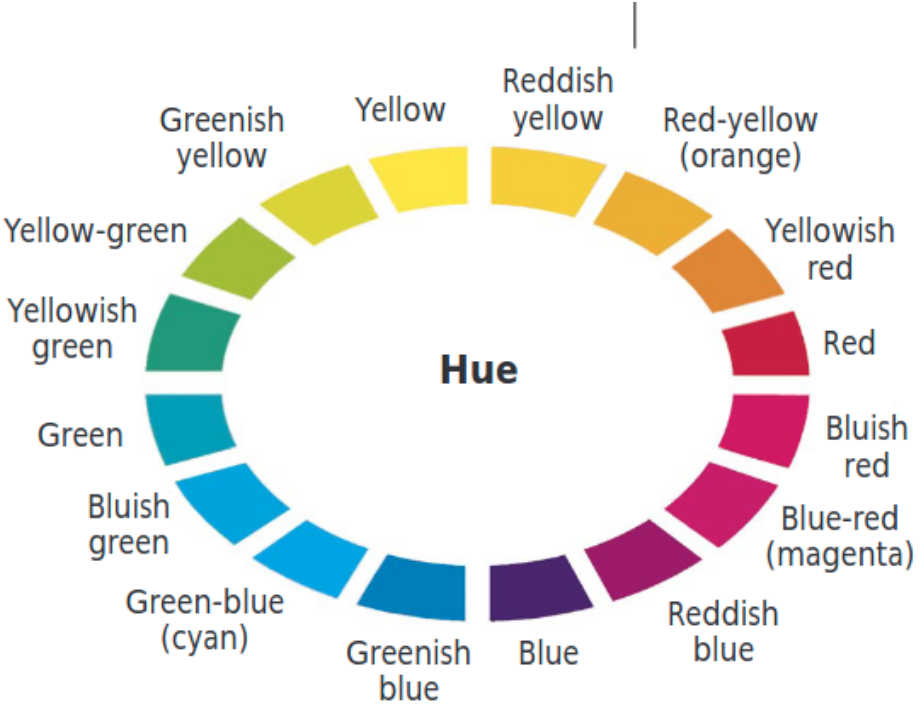
Color: salient but complicated

RGB: Types of photoreceptors in the retina



https://www.simplypsychology.org/what-is-the-trichromatic-theory-of-color-vision.html

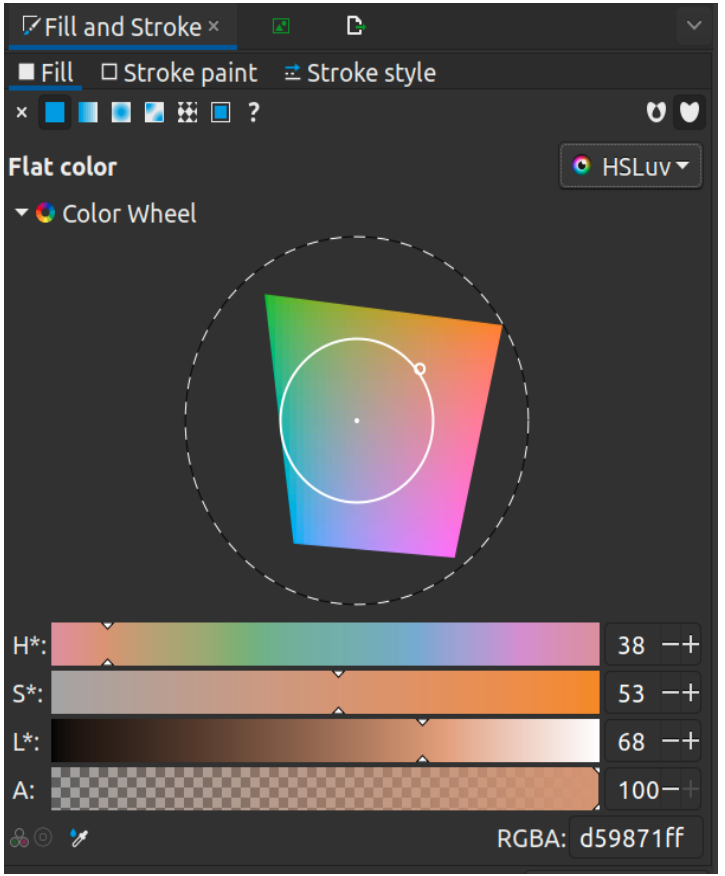
Hue: how we casually talk about color



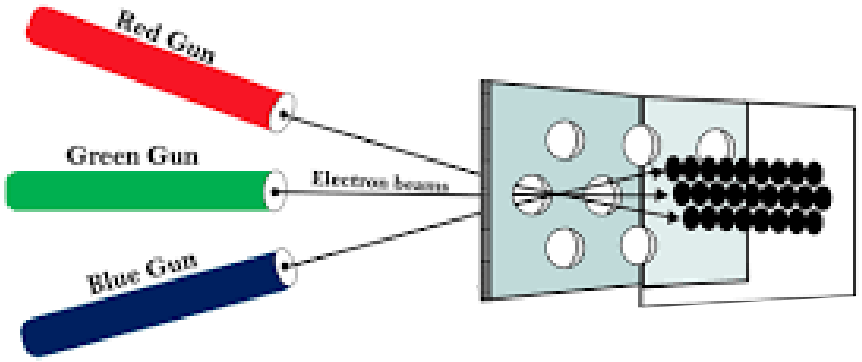
https://www.engr.colostate.edu/ECE666/Handouts/WritingPapers/UsingColorEffectively.pdf

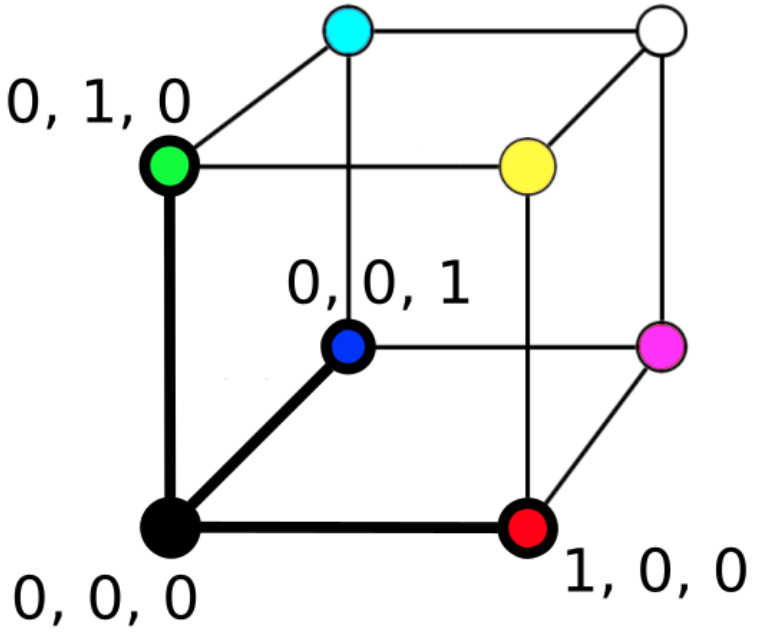
HSLuv: how we talk about color perception

Hue
Saturation
Lightness / **L**uminance



RGB: How a computer produces color





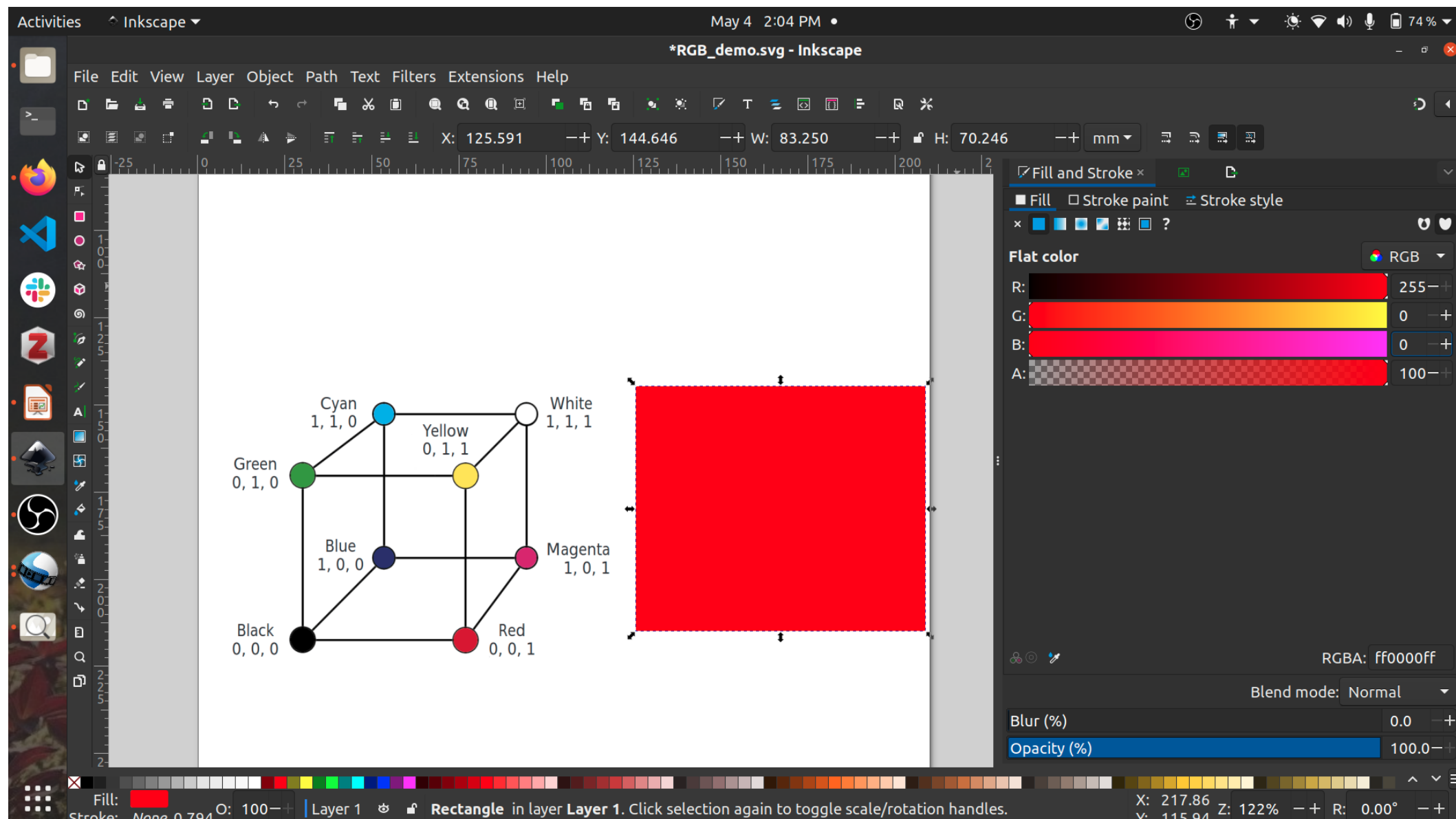
https://www.javatpoint.com/computer-graphics-color-crt-monitors

Specifying colors

Color formats

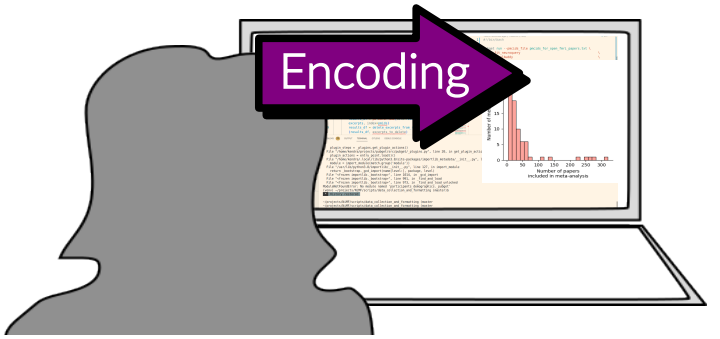
Matplotlib recognizes the following formats to specify a color.

Format	Example
RGB or RGBA (red, green, blue, alpha) tuple of float values in a closed interval [0, 1].	`(0.1, 0.2, 0.5)` `(0.1, 0.2, 0.5, 0.3)`
Case-insensitive hex RGB or RGBA string.	`'#0f0f0f'` `'#0f0f0f80'`
Case-insensitive RGB or RGBA string equivalent hex shorthand of duplicated	`'#abc'` as `'#aabbcc'` `'#fb1'` as `'#ffbb11'`


Activities
Inkscape
May 4 2:04 PM
74 %
*RGB_demo.svg - Inkscape
File Edit View Layer Object Path Text Filters Extensions Help
X: 125.591
Y: 144.646
W: 83.250
H: 70.246
mm
Cyan
1, 1, 0
White
1, 1, 1
Yellow
0, 1, 1
Green
0, 1, 0
Blue
1, 0, 0
Magenta
1, 0, 1
Black
0, 0, 0
Red
0, 0, 1
Fill and Stroke
Fill
Stroke paint
Stroke style
Flat color
RGB
R: 255
G: 0
B: 0
A: 100
RGBA: ff0000ff
Blend mode: Normal
Blur (%) 0.0
Opacity (%) 100.0
Fill:
Stroke: None 0.794
O: 100
Layer 1
Rectangle in layer Layer 1. Click selection again to toggle scale/rotation handles.
X: 217.86
Z: 122%
R: 0.00°

TL;DR

- Images have different properties/features because of **how they’re made**
- The way a computer produces color (**RGB) is not intuitive**
  - You’ll probably pick colors using named colors or an HSL tool



To program a figure efficiently and reproducibly, we should understand

- Some computer graphics
  - How the computer makes figure
- Matplotlib basics
  - What you and matplotlib need to do
- Higher-level libraries
  - More complex visualizations

Home
PUBLIC
Questions
Tags
Users
Companies
COLLECTIVES
Explore Collectives
TEAMS

Adding a matplotlib legend

Asked 9 years, 7 months ago Modified 29 days ago Viewed 1.2m times

Ask Question

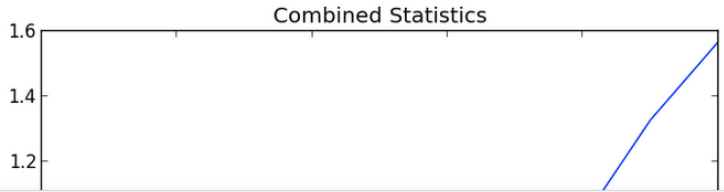
469

How can one create a legend for a line graph in `Matplotlib`'s `PyPlot` without creating any extra variables?

Please consider the graphing script below:

```
if __name__ == '__main__':
    PyPlot.plot(total_lengths, sort_times_bubble, 'b-',
                total_lengths, sort_times_ins, 'r-',
                total_lengths, sort_times_merge_r, 'g+',
                total_lengths, sort_times_merge_i, 'p-', )
    PyPlot.title("Combined Statistics")
    PyPlot.xlabel("Length of list (number)")
    PyPlot.ylabel("Time taken (seconds)")
    PyPlot.show()
```

As you can see, this is a very basic use of `matplotlib`'s `PyPlot`. This ideally generates a graph like the one below:



The Overflow Blog
Don't panic! A playbook for managing any production incident
How to land a job in climate tech
Featured on Meta
New blog post from our CEO Prashanth: Community is the future of AI
We are updating our Code of Conduct and we would like your feedback
Temporary policy: ChatGPT is banned
Content Discovery initiative April 13 update: Related questions using a...
The [connect] tag is being burninated
Removal of the Census badge
Linked
0 Add legends in pyplot with Pandas Dataframe data

Home
PUBLIC
Questions
Tags
Users
Companies
COLLECTIVES
Explore Collectives
TEAMS

Here's an example to help you out ...

27

```
fig = plt.figure(figsize=(10,5))
ax = fig.add_subplot(111)
ax.set_title('ADR vs Rating (CS:GO)')
ax.scatter(x=data[:,0],y=data[:,1],label='Data')
plt.plot(data[:,0], m*data[:,0] + b,color='red',label='Our Fitting
Line')
ax.set_xlabel('ADR')
ax.set_ylabel('Rating')
ax.legend(loc='best')
plt.show()
```

Share Improve this answer Follow

answered Dec 6, 2017 at 7:00
Akash Kandpal

Home
PUBLIC
Questions
Tags
Users
Companies
COLLECTIVES
Explore Collectives
TEAMS

Add a comment

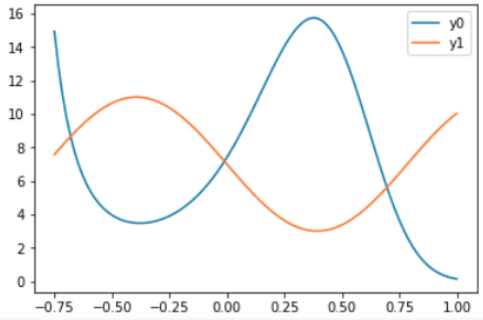
62

You can access the Axes instance (`ax`) with `plt.gca()`. In this case, you can use

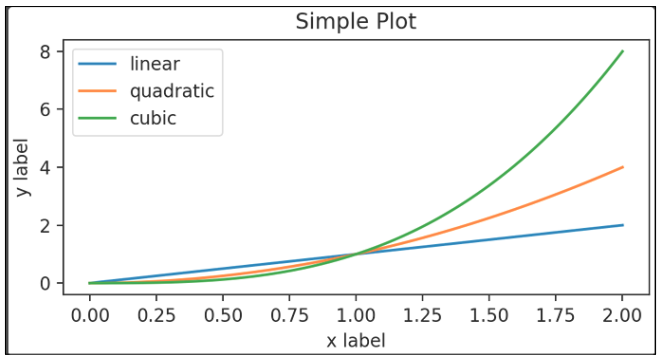
```
plt.gca().legend()
```

You can do this either by using the `label=` keyword in each of your `plt.plot()` calls or by assigning your labels as a tuple or list within `legend`, as in this working example:

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(-0.75,1,100)
y0 = np.exp(2 + 3*x - 7*x**3)
y1 = 7-4*np.sin(4*x)
plt.plot(x,y0,x,y1)
plt.gca().legend(('y0','y1'))
plt.show()
```



Coding styles



Explicit (object-oriented)

```
x = np.linspace(0, 2, 100)  # Sample data.

# Note that even in the OO-style, we use `.pyplot.figure` to create the Figure.
fig, ax = plt.subplots(figsize=(5, 2.7), layout='constrained')
ax.plot(x, x, label='linear')  # Plot some data on the axes.
ax.plot(x, x**2, label='quadratic')  # Plot more data on the axes...
ax.plot(x, x**3, label='cubic')  # ... and some more.
ax.set_xlabel('x label')  # Add an x-label to the axes.
ax.set_ylabel('y label')  # Add a y-label to the axes.
ax.set_title("Simple Plot")  # Add a title to the axes.
ax.legend()  # Add a legend.
```

Implicit

```
x = np.linspace(0, 2, 100)  # Sample data.

plt.figure(figsize=(5, 2.7), layout='constrained')
plt.plot(x, x, label='linear')  # Plot some data on the (implicit) axes.
plt.plot(x, x**2, label='quadratic')  # etc.
plt.plot(x, x**3, label='cubic')
plt.xlabel('x label')
plt.ylabel('y label')
plt.title("Simple Plot")
plt.legend()
```


https://matplotlib.org/stable/tutorials/introductory/quick_start.html

Some objects

Explicit (**object**-oriented)

```
x = np.linspace(0, 2, 100)  # Sample data.

                 at even in the OO-style, we use `.pyplot.figure` to create the Figure.
fig  ax          plt.subplots(figsize=(5, 2.7), layout='constrained')
                 , x, label='linear')  # Plot some data on the axes.
ax.plot(x, x**2, label='quadratic')  # Plot more data on the axes...
x.plot(x, x**3, label='cubic')  # ... and some more.
ax.set_xlabel('x label')  # Add an x-label to the axes.
ax.set_ylabel('y label')  # Add a y-label to the axes.
ax.set_title("Simple Plot")  # Add a title to the axes.
ax.legend()  # Add a legend.
```

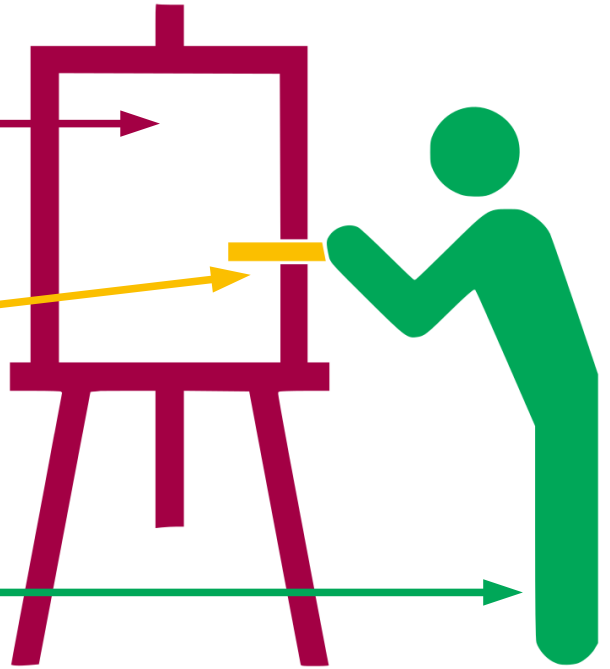
Figure
- The top-level container for all elements of the plot

Axes
- An **Artist** subclass attached to a **Figure**
- Incl. All elements of an individual (sub-)plot

Artist
- Abstract base class
- Everything visible is a subclass of **Artist**

3 layers of matplotlib

- "matplotlib.backend_bases.FigureCanvas
  is the area onto which the figure is drawn
  - matplotlib.backend_bases.Renderer
    is the object which knows how to draw on the FigureCanvas
    - matplotlib.artist.Artist
      is the object that knows how to use a renderer to paint onto the canvas."



What you'll be working with 95% of the time

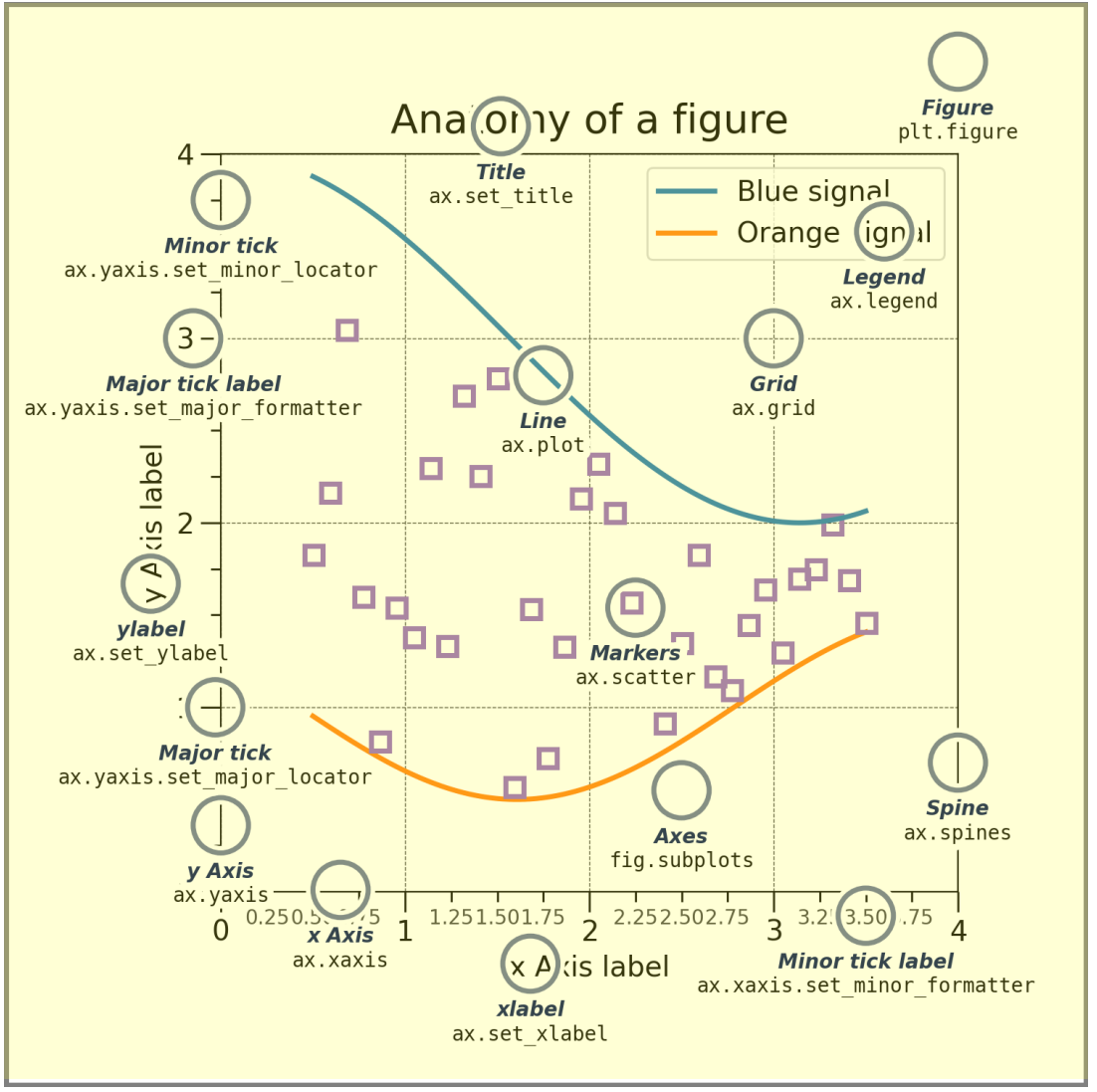
https://matplotlib.org/stable/tutorials/intermediate/artists.html

https://www.svgrepo.com/svg/307645/art-artist-craft-canvas

2 types of artists

1) Primatives: things to draw
   - Line2D
   - Rectangle
   - Text
   - AxesImage

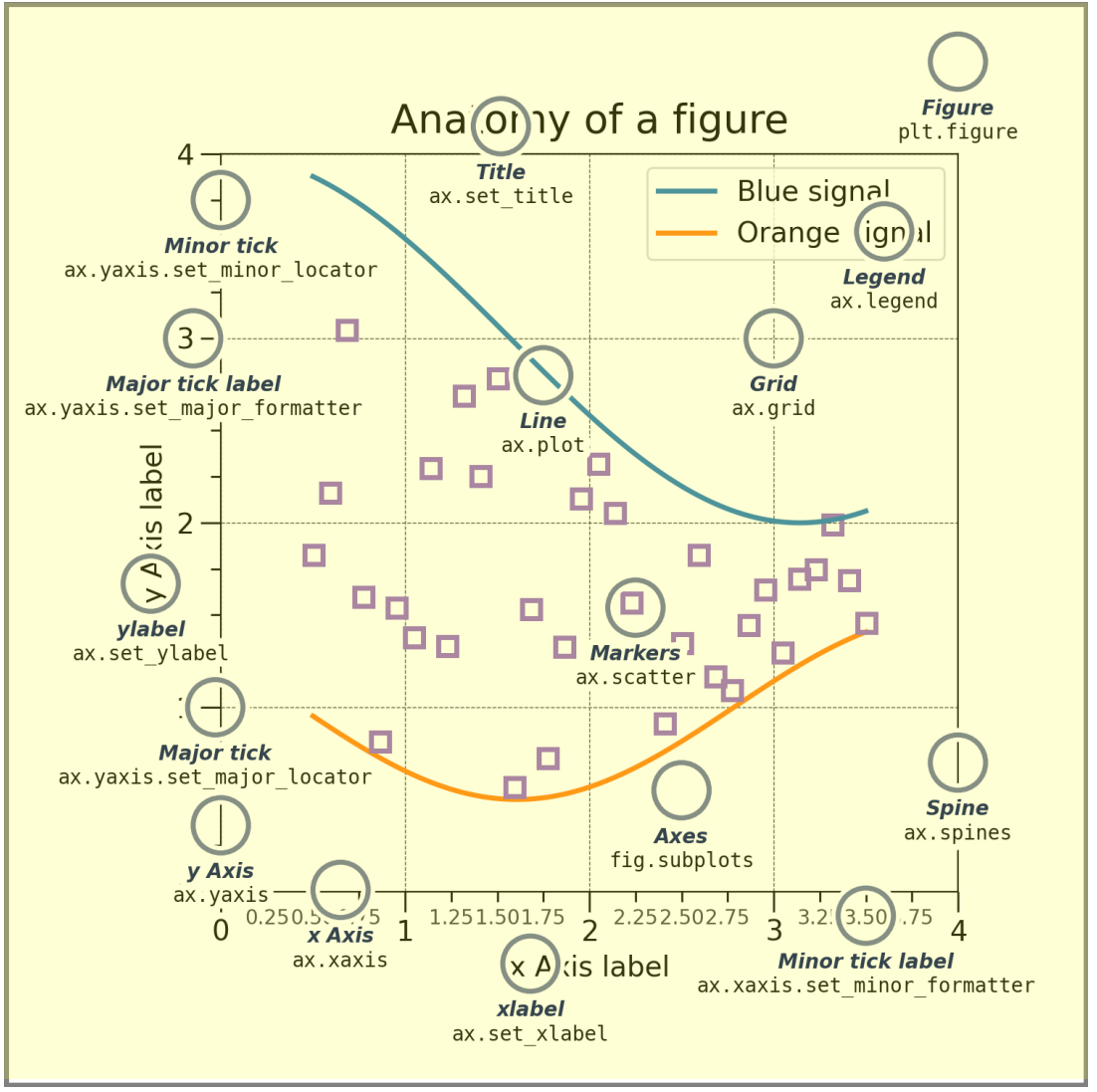
2) Containers: where to draw them
   - Figure
   - Axes
   - Axis



2 types of artists

1) Primatives: things to draw
   - Line2D
   - Rectangle
   - Text
   - AxesImage

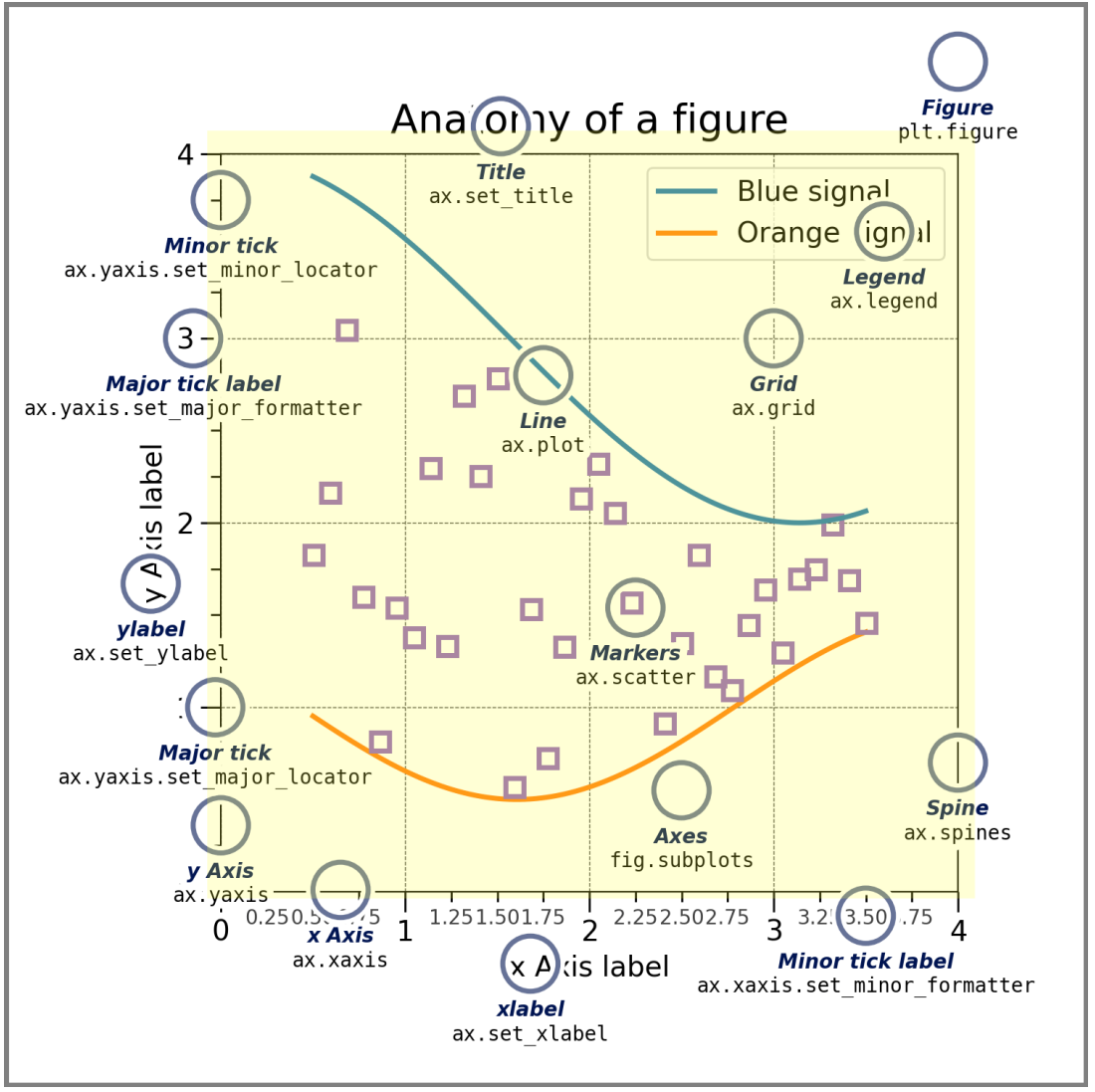
2) Containers: where to draw them
   - Figure
   - Axes
   - Axis



2 types of artists

1) Primatives: things to draw
   - Line2D
   - Rectangle
   - Text
   - AxesImage

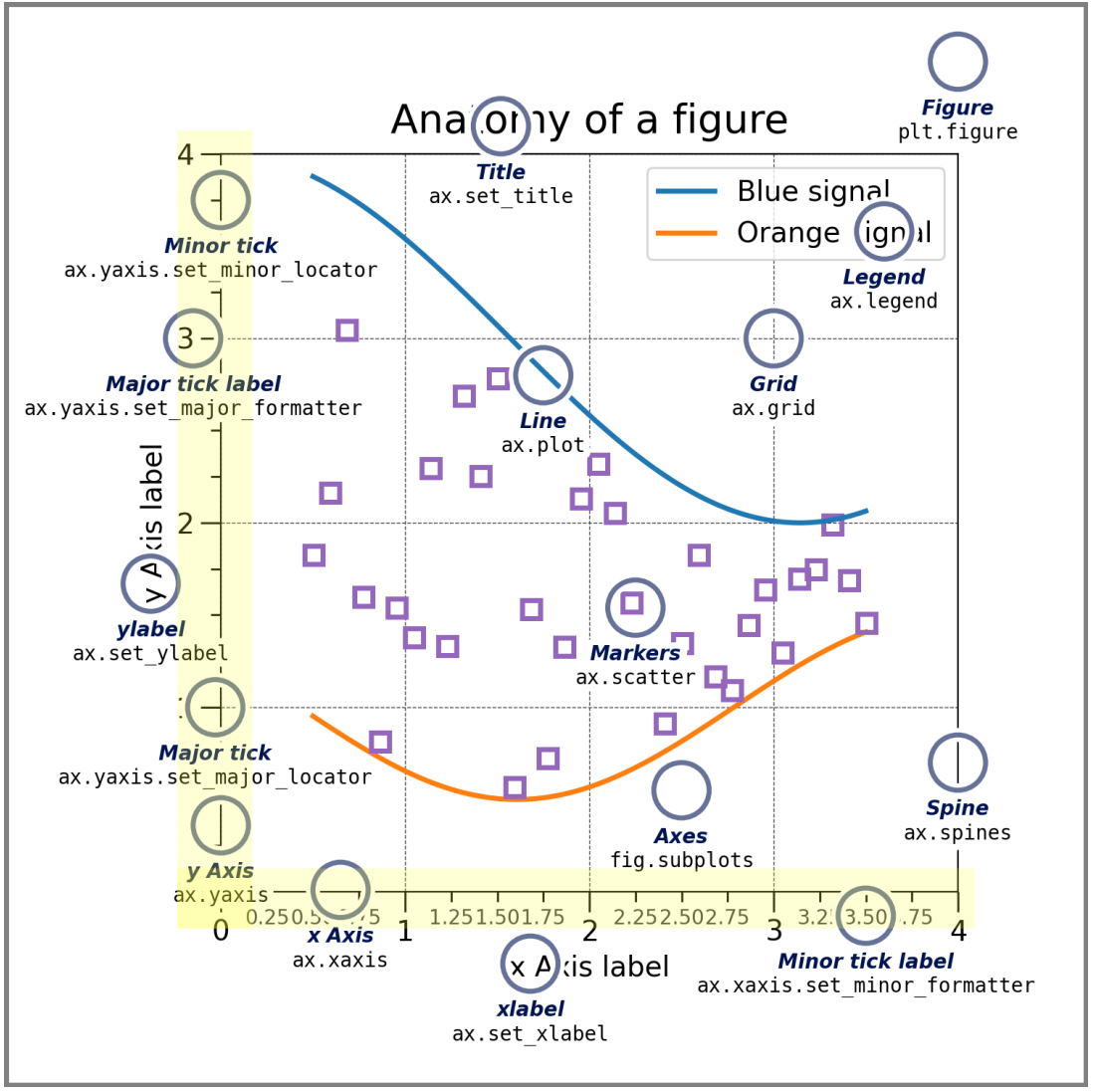
2) Containers: where to draw them
   - Figure
   - Axes
   - Axis



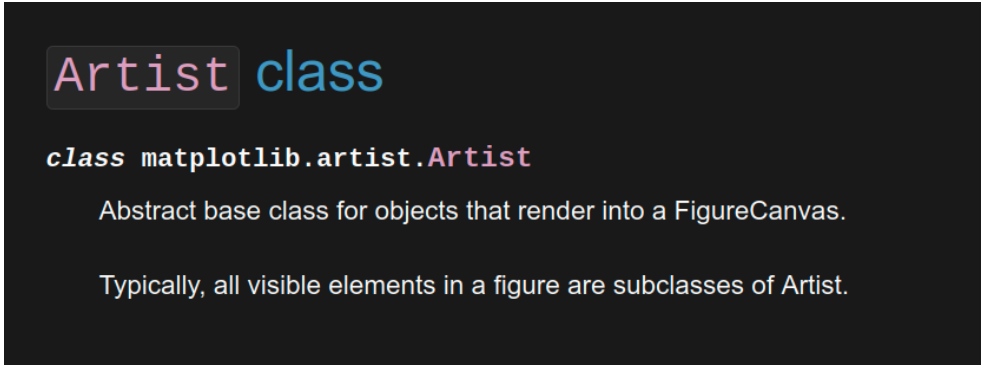
2 types of artists

1) Primatives: things to draw
   - Line2D
   - Rectangle
   - Text
   - AxesImage

2) Containers: where to draw them
   - Figure
   - Axes
   - Axis



What actually is an Artist??



What actually is an Artist??

`Artist` class

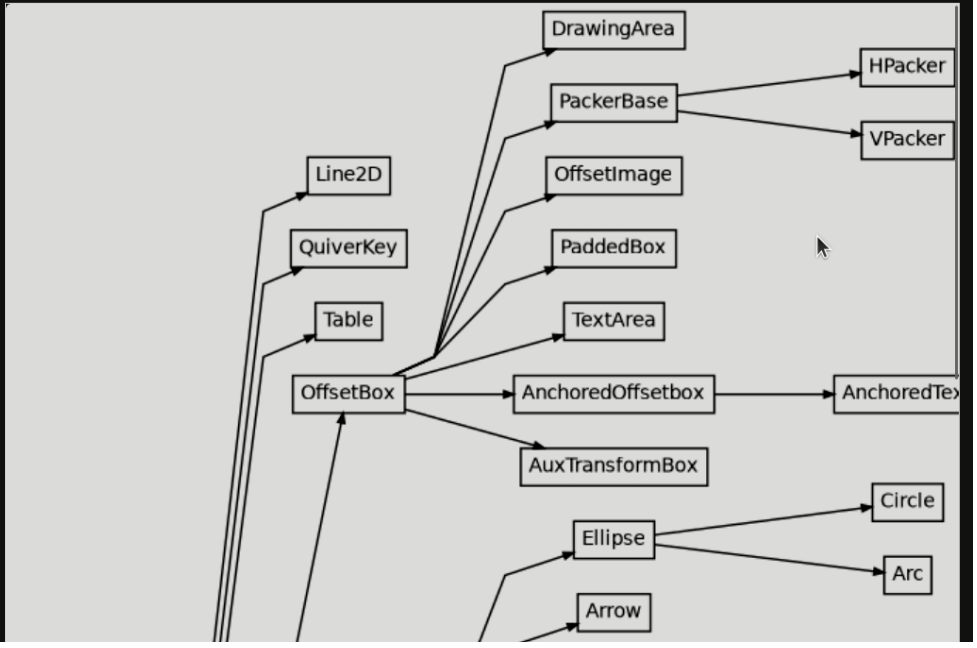
class **matplotlib.artist.Artist**

Abstract base class for objects that render into a FigureCanvas.

Typically, all visible elements in a figure are subclasses of Artist.

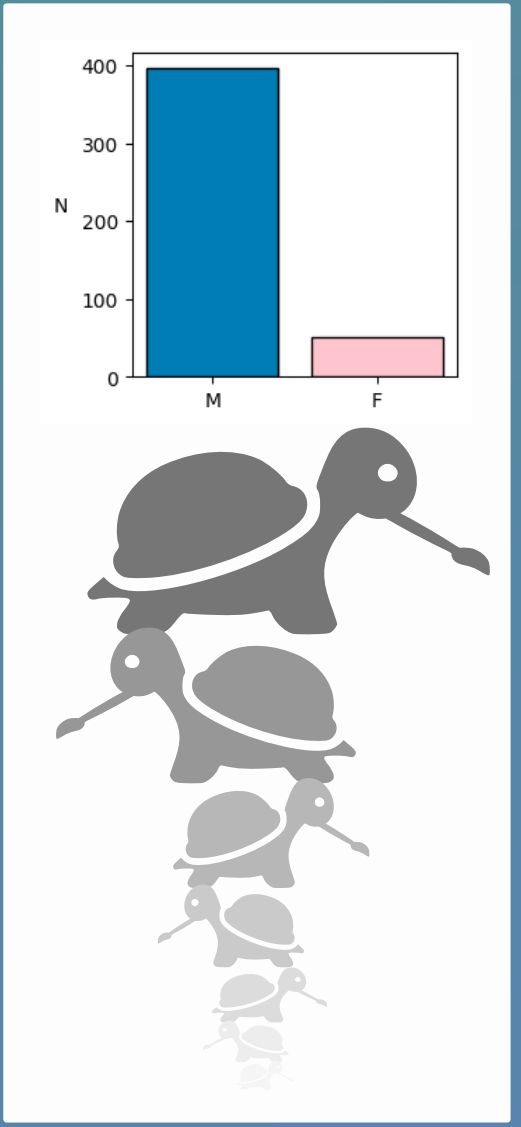
`matplotlib.artist`

Inheritance Diagrams



TL;DR for our purposes,

it’s Artists all the way down

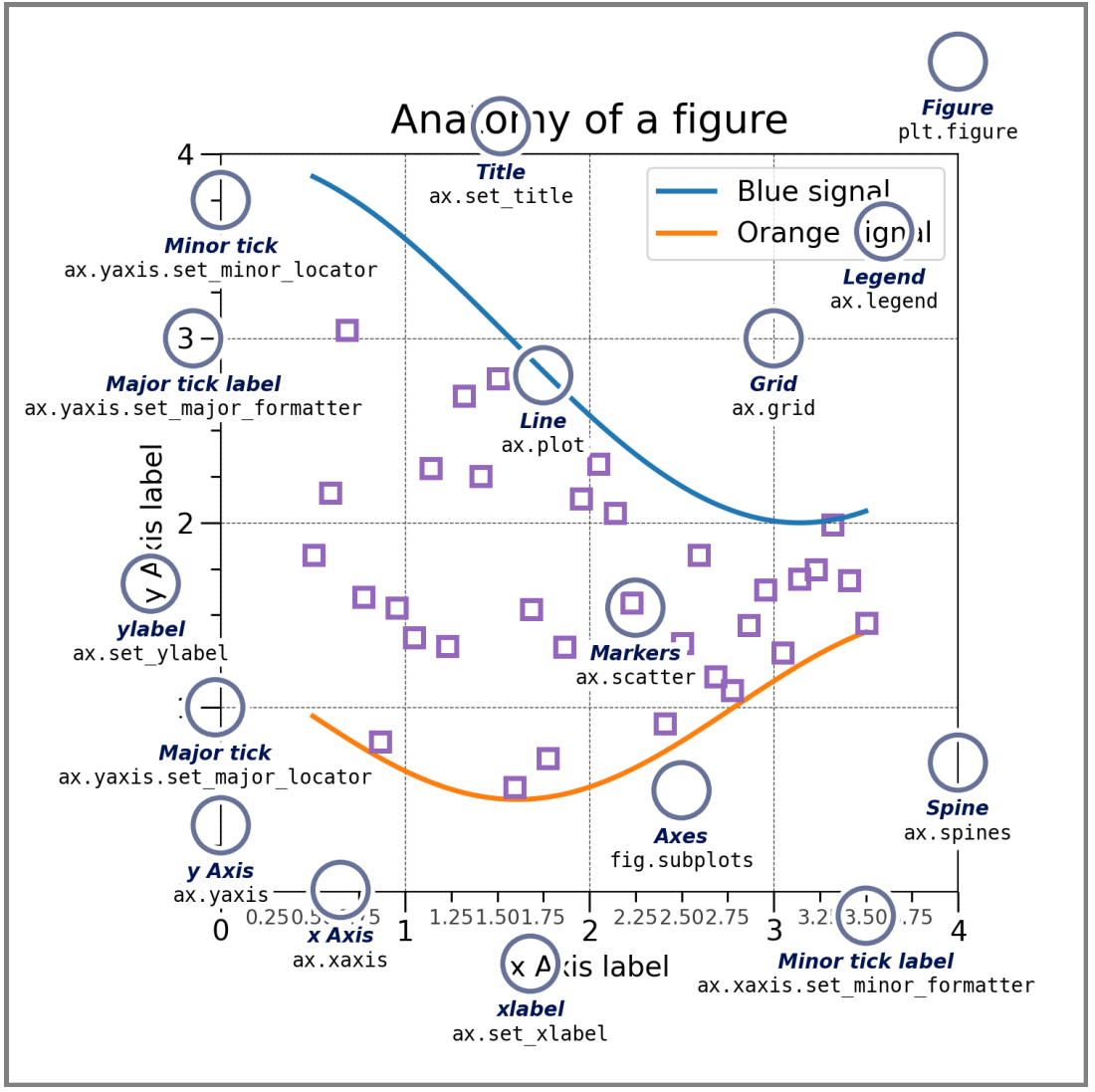


Axes helper methods for plot types

- ax.plot() # line graph
- ax.bar() # bar graph
- ax.imshow() # image

So you don’t have to construct figures from shapes

Helper methods for customization



https://matplotlib.org/stable/tutorials/introductory/quick_start.html#sphx-glr-tutorials-introductory-quick-start-py

matplotlib Cheat sheet

Version 3.5.0

Quick start

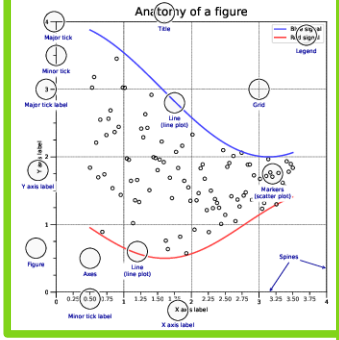
```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt


X = np.linspace(0, 2*np.pi, 100)
Y = np.cos(X)

fig, ax = plt.subplots()
ax.plot(X, Y, color='green')

fig.savefig("figure.pdf")
fig.show()
```

Anatomy of a figure



Subplots layout

```
subplot[s](rows,cols,…)
fig, axs = plt.subplots(3, 3)

G = gridspec(rows,cols,…)
ax = G[0,:]

ax.inset_axes(extent)

d=make_axes_locatable(ax)
ax = d.new_horizontal('10%')
```

Getting help

- matplotlib.org
- github.com/matplotlib/matplotlib/issues
- discourse.matplotlib.org
- stackoverflow.com/questions/tagged/matplotlib
- https://gitter.im/matplotlib/matplotlib
- twitter.com/matplotlib
- Matplotlib users mailing list

Basic plots

```
plot([X],Y,[fmt],…)
X, Y, fmt, color, marker, linestyle

scatter(X,Y,…)
X, Y, [s]izes, [c]olors, marker, cmap

bar[h](x,height,…)
x, height, width, bottom, align, color

imshow(Z,…)
Z, cmap, interpolation, extent, origin

contour[f]([X],[Y],Z,…)
X, Y, Z, levels, colors, extent, origin

pcolormesh([X],[Y],Z,…)
X, Y, Z, vmin, vmax, cmap

quiver([X],[Y],U,V,…)
X, Y, U, V, C, units, angles

pie(X,…)
Z, explode, labels, colors, radius

text(x,y,text,…)
x, y, text, va, ha, size, weight, transform

fill[_between][x](…)
X, Y1, Y2, color, where
```

Advanced plots

```
step(X,Y,[fmt],…)
X, Y, fmt, color, marker, where

boxplot(X,…)
X, notch, sym, bootstrap, widths

errorbar(X,Y,xerr,yerr,…)
X, Y, xerr, yerr, fmt

hist(X, bins, …)
X, bins, range, density, weights

violinplot(D,…)
D, positions, widths, vert

barbs([X],[Y], U, V, …)
X, Y, U, V, C, length, pivot, sizes

eventplot(positions,…)
positions, orientation, lineoffsets

hexbin(X,Y,C,…)
X, Y, C, gridsize, bins
```

Scales

```
ax.set_[xy]scale(scale,…)
linear  any values
log     values > 0
symlog  any values
logit   0 < values < 1
```

Projections

```
subplot(…,projection=p)
p='polar'   p='3d'
p=ccrs.Orthographic()
import cartopy.crs as ccrs
```

Lines

```
linestyle or ls
"-"  "--"  "-."  ":"  (0,(0.01,2))
capstyle or dash_capstyle
"butt"  "round"  "projecting"
```

Markers

```
'.' 'o' 's' 'P' 'X' '*' 'p' 'D' '<' '>' '^' 'v'
'1' '2' '3' '4' '+' 'x' '|' '_' 4 5 6 7
markevery
10   [0, -1]   (25, 5)   [0, 25, -1]
```

Colors

```
'Cn'  'x'  'name'  (R,G,B[,A])  '#RRGGBB[AA]'  'x.y'
```

Colormaps

```
plt.get_cmap(name)
```

Uniform: viridis, magma, plasma

Sequential: Greys, YlOrBr, Wistia

Diverging: Spectral, coolwarm, RdGy

Qualitative: tab10, tab20

Cyclic: twilight

Tick locators

```
from matplotlib import ticker
ax.[xy]axis.set_[minor|major]_locator(locator)
ticker.NullLocator()
ticker.MultipleLocator(0.5)
ticker.FixedLocator([0, 1, 5])
ticker.LinearLocator(numticks=3)
ticker.IndexLocator(base=0.5, offset=0.25)
ticker.AutoLocator()
ticker.MaxNLocator(n=4)
ticker.LogLocator(base=10, numticks=15)
```

Tick formatters

```
from matplotlib import ticker
ax.[xy]axis.set_[minor|major]_formatter(formatter)
ticker.NullFormatter()
ticker.FixedFormatter(['zero', 'one', 'two', …])
ticker.FuncFormatter(lambda x, pos: "[%.2f]" % x)
ticker.FormatStrFormatter('>%d<')
ticker.ScalarFormatter()
ticker.StrMethodFormatter('{x}')
ticker.PercentFormatter(xmax=5)
```

Ornaments

```
ax.legend(…)
handles, labels, loc, title, frameon
ax.colorbar(…)
mappable, ax, cax, orientation
ax.annotate(…)
text, xy, xytext, xycoords, textcoords, arrowprops
```

Event handling

```
fig, ax = plt.subplots()
def on_click(event):
  print(event)
fig.canvas.mpl_connect(
  'button_press_event', on_click)
```

Animation

```
import matplotlib.animation as mpla

T = np.linspace(0, 2*np.pi, 100)
S = np.sin(T)
line, = plt.plot(T, S)
def animate(i):
    line.set_ydata(np.sin(T+i/50))
anim = mpla.FuncAnimation(
    plt.gcf(), animate, interval=5)
plt.show()
```

Styles

```
plt.style.use(style)
```

default, classic, grayscale, ggplot, seaborn, fast, bmh, Solarize_Light2, seaborn-notebook

Quick reminder

```
ax.grid()
ax.set_[xy]lim(vmin, vmax)
ax.set_[xy]label(label)
ax.set_[xy]ticks(ticks, [labels])
ax.set_[xy]ticklabels(labels)
ax.set_title(title)
ax.tick_params(width=10, …)
ax.set_axis_[on|off]()
fig.suptitle(title)
fig.tight_layout()
plt.gcf(), plt.gca()
mpl.rc('axes', linewidth=1, …)
[fig|ax].patch.set_alpha(0)
text=r'$\frac{-e^{i\pi}}{2^n}$'
```

Keyboard shortcuts

ctrl + s	Save	ctrl + w	Close plot
r	Reset view	f	Fullscreen 0/1
f	View forward	b	View back
p	Pan view	o	Zoom to rect
x	X pan/zoom	y	Y pan/zoom
g	Minor grid 0/1	G	Major grid 0/1
l	X axis log/linear	L	Y axis log/linear

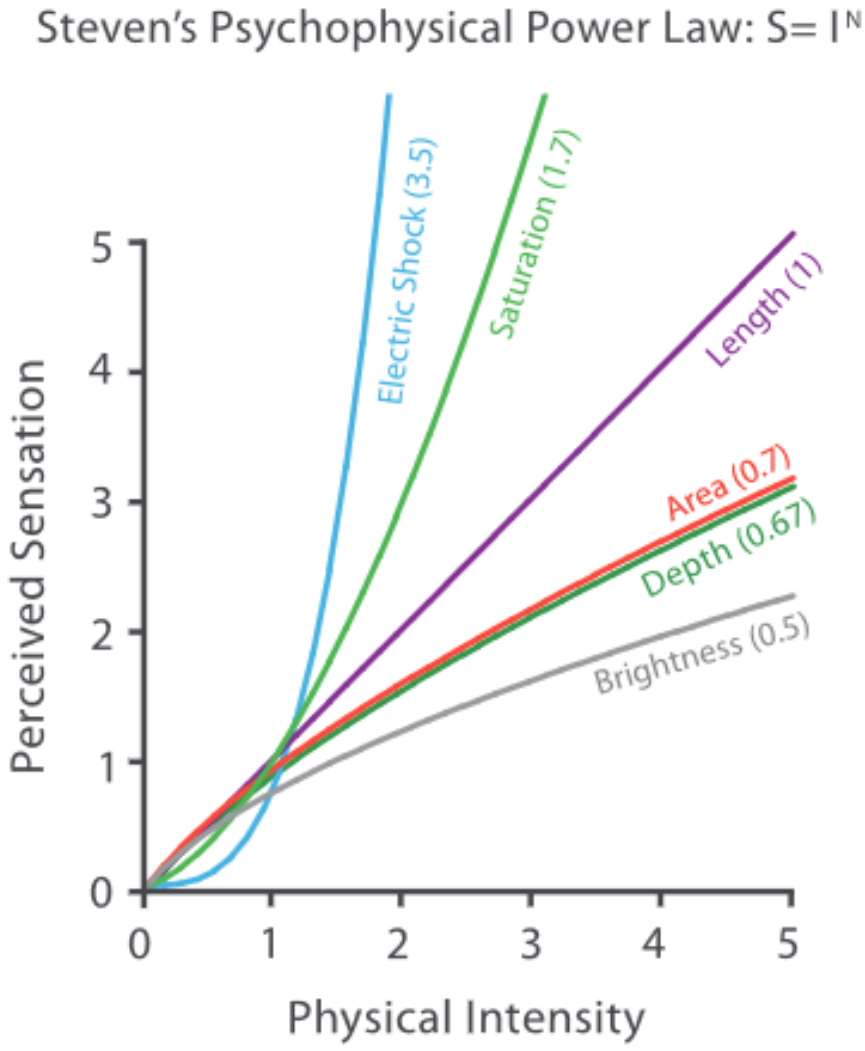
Ten simple rules

1. Know your audience
2. Identify your message
3. Adapt the figure
4. Captions are not optional
5. Do not trust the defaults
6. Use color effectively
7. Do not mislead the reader
8. Avoid "chartjunk"
9. Message trumps beauty
10. Get the right tool

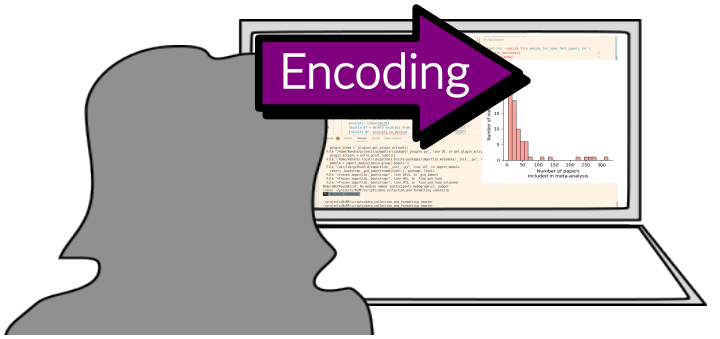
https://matplotlib.org/cheatsheets/_images/cheatsheets-1.png

Live coding

- Replicating this figure



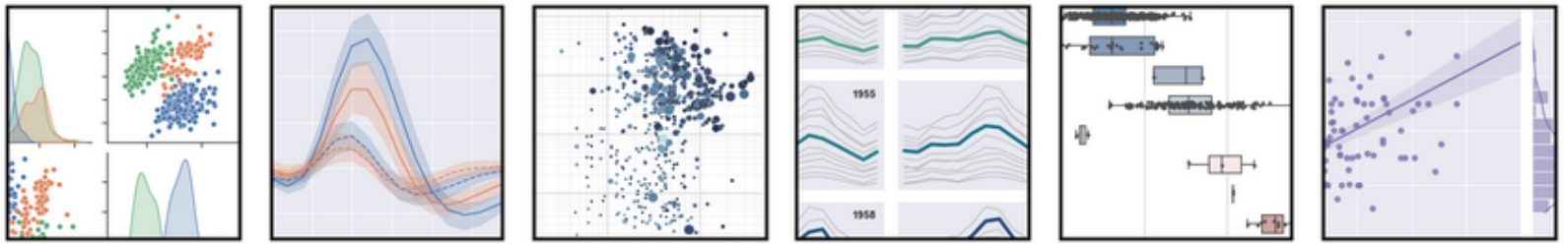
Munzner, T. (2014). Visualization analysis and design. CRC press.



To program a figure efficiently and reproducibly, we should understand

- Some computer graphics
  - How the computer makes figure
- Matplotlib basics
  - What you and matplotlib need to do
- Higher-level libraries
  - More complex visualizations

seaborn: statistical data visualization



Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

Features of Seaborn

- Works well with **pandas.DataFrame()** objects
- Default colors are from **perceptually uniform** pallettes
- Can **calculate stats** as well as visualize
- More easily visualize **complex data** (e.g, multivariate)

Nilearn

https://nilearn.github.io/stable/plotting/index.html

Nilearn

Search

Quickstart

Examples

User guide

1. Introduction
2. What is `nilearn`?
3. Using `nilearn` for the first time
4. Machine learning applications to Neuroimaging
5. Decoding and MVPA: predicting from brain images

7. Plotting brain images

In this section, we detail the general tools to visualize neuroimaging volumes and surfaces with nilearn.

Nilearn comes with plotting function to display brain maps coming from Nifti-like images, in the `nilearn.plotting` module.

Code examples

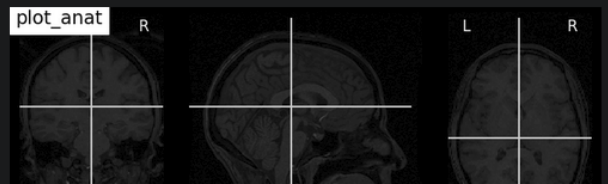
Nilearn has a whole section of the example gallery on plotting.

A small tour of the plotting functions can be found in the example Plotting tools in nilearn.

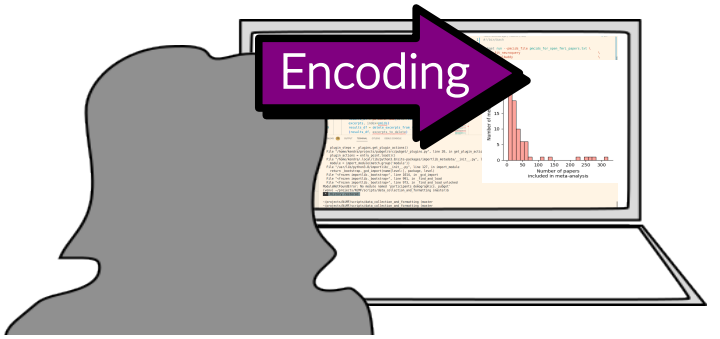
Finally, note that, as always in the nilearn documentation, clicking on a figure will take you to the code that generates it.

7.1. Different plotting functions

Nilearn has a set of plotting functions to plot brain volumes that are fined tuned to specific applications. Amongst other things, they use different heuristics to find cutting coordinates.



`plot_anat`
Plotting an anatomical image



To program a figure efficiently and reproducibly, we should understand

- Some computer graphics
  - How the computer makes figure
- Matplotlib basics
  - What you and matplotlib need to do
- Higher-level libraries
  - More complex visualizations