Engineering Programming Cheat Sheet

Variables and strings

Variables are used to store values. A string is a series of characters, surrounded by single or double quotes.

Hello world

```
print(“Hello world!”)
```

Concatenation (combining strings)

```
first_name = 'albert'
last_name = 'einstein'
full_name = first_name + ' ' + last_name
```

Lists

A list stores a series of items in a particular order. You access items using an index, or within a loop.

Make a list

```
bikes = ['trek', 'redline', 'giant']
```

Get the first item in a list

```
first_bike = bikes[0]
```

Get the last item in a list

```
last_bike = bikes[-1]
```

Looping through a list

```
for bike in bikes:
    print(bike)
```

Adding items to a list

```
bikes = []
bikes.append('trek')
list3 = list1 + list2
```

Making numerical lists

```
squares = []
for x in range(1, 11):
    squares.append(x**2)
```

List comprehensions

```
squares = [x**2 for x in range(1, 11)]
```

Slicing a list

```
finishers = ['sam', 'bob', 'ada', 'bea']
first_two = finishers[:2]
```

Copying a list

```
copy_of_bikes = bikes[:]
```

Zip

zip “pairs” up the elements of a number of lists, tuples, or other sequences to create a list of tuples:

```
seq1 = ['foo', 'bar', 'baz']
seq2 = ['one', 'two']
zipped = zip(seq1, seq2)
list(zipped)
Out: [('foo', 'one'), ('bar', 'two')]
```

If Statements

If statements are used to test for particular conditions and respond appropriately.

Conditional tests

equals	`x == 42`
not equal	`x != 42`

greater than
or equal to
	`x > 42`
	`x >= 42`

less than
or equal to
	`x < 42`
	`x <= 42`

Conditional test with lists

```
first_bike = bikes[0]
'trek' in bikes
'surly' not in bikes
```

Dictionaries

Dictionaries store connections between pieces of information. Each item in a dictionary is a key-value pair.

A simple dictionary

```
alien = {'color': 'green', 'points': 5}
```

Accessing a value

```
print("The alien's color is " + alien['color'])
```

Adding elements

```
alien['x_position'] = 0
alien.update({'size': 42, 'eyes': 3})
```

Looping through all key-value pairs

```
fav_numbers = {'eric': 17, 'ever': 4}
for name, number in fav_numbers.items():
print(name + ' loves ' + str(number))
```

Looping through all keys

```
for name in fav_numbers.keys():
    print(name + ' loves a number')
```

Looping through all the values

```
for number in fav_numbers.values():
    print(str(number) + ' is a favorite')
```

Creating a dict from a pair of lists

```
mapping = dict(zip(key_list, value_list))
```

User Input

Your programs can prompt the user for input. All input is stored as a string.

Prompting for a value

```
name = input("What's your name? ")
print("Hello, " + name + "!")
```

Prompting for numerical input

```
age = input("How old are you? ")
age = int(age)
pi = input("What's the value of pi? ")
pi = float(pi)
```

Functions

Functions are named blocks of code, designed to do one specific job. Information passed to a function is called an argument, and information received by a function is called a parameter.

A simple function

```
def greet_user():
   print("Hello!")
greet_user()
```

Passing an argument

```
def greet_user(username):
"""Display a personalized greeting."""
    print("Hello, " + username + "!")
greet_user('jesse')
```

Default values for parameters

```
def make_pizza(topping='bacon'):
"""Make a single-topping pizza."""
print("Have a " + topping + " pizza!")
make_pizza()
make_pizza('pepperoni')
```

Returning a value

```
def add_numbers(x, y):
"""Add two numbers and return the sum."""
return x + y
sum = add_numbers(3, 5)
print(sum)
```

Working with Files

Your programs can read from files and write to files.

Path

Absolute (complete location) - *D:\documents\mydocument.doc*
Relative (to the current directory) - *mydocument.doc*

Opening a file

```
path = 'mydocument.doc'
mode = 'r'
with open(path, mode) as f:
   pass
```

Modes

'r'	read-only
'w'	write-only, erasing existing file
'a'	append to existing file
'r+'	read and write

Reading and writing

```
     with open('demo.txt', 'w') as f:
2       f.writelines("one text line %d\n" %i for i in range(4))
     with open('demo.txt', 'r') as f:

2       lines = f.readlines()

3 Out: ['line 0\n', 'line 1\n', 'line 2\n', 'line 3\n']
```

KISS – Keep It Short and Simple

Simple is better than complex

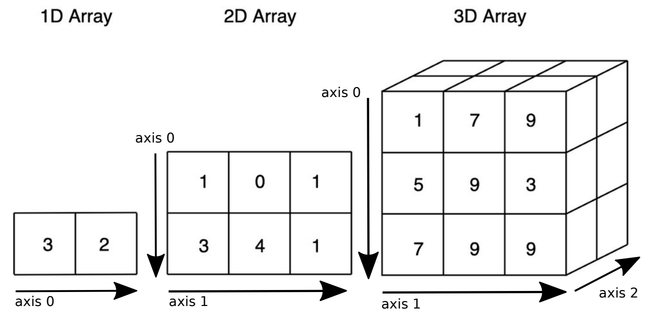
If you have a choice between a simple and a complex solution, and both work, use the simple solution. Your code will be easier to maintain, and it will be easier for you and others to build on that code later on.

NumPy

Import Convention

```
> import numpy as np
```

NumPy Arrays



Data Types

> np.int64	Signed 64-bit integer types
> np.float32	Standard double-precision floating point
> np.complex	Complex numbers represented by 128 floats
> np.bool	Boolean type storing TRUE and FALSE values
> np.object	Python object type
> np.string_	Fixed-length string type
> np.unicode_	Fixed-length unicode type

Creating Arrays

> arr = [1,2,3]	
> a = np.array(arr)	
> b = np.array([(1.5,2,3) (4,5,6)], dtype = float)	
> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]], dtype = float)	
> np.zeros(shape = (3,4))	Create an array of zeros
> np.ones(shape, dtype)	Create an array of ones
> d = np.arange(start, stop, step)	Create an array of evenly spaced values (step value)
> np.linspace(start, stop, number)	Create an array of evenly spaced values (number of samples)
> np.full(shape, fill_value)	Create an array filled with *fill_value*
> np.eye(N)	Create a NxN identity matrix
> np.random.randint(low, high, size)	Create an array with random int values
> np.empty(shape)	Create an empty array
> e = np.copy(a)	Create a copy of the array

Inspecting your Array

> a.shape	Array dimensions
> len(a)	Length of array
> b.ndim	Number of array dimensions
> e.size	Number of array elements
> b.dtype	Data type of array elements
> b.dtype.name	Name of data type
> b.astype(int)	Convert an array to a different type

Array Mathematics

Arithmetic Operations

> a – b	Substraction
> np.substract(a, b)	
> b + a	Addition
> np.add(a, b)	
> a / b	Division
> np.divide(a, b)	
> a * b	Multiplication
> np.multiply(a, b)	
> np.exp(b)	Exponentiation
> np.sqrt(b)	Square root
> np.sin(a)	Sine
> np.cos(b)	Cosine
> np.log(a)	Natural logarithm
> a.dot(b)	Dot product
> np.cross(a, b)	Cross product

Comparison

> a == b	
array([[False, True, True],	Element-wise comparison
[False, False, False]])	
> a <= 2	Element-wise comparison
array([True, False, False])	
> np.array_equal(a, b)	Array-wise comparison
> np.all(a, axis)	Test whether all array elements along
> np.all(a==b)	a given axis evaluate to True

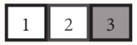
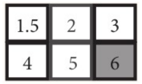
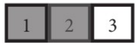
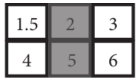
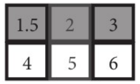
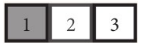
Aggregate Functions

> a.sum()	Array-wise sum
> a.min()	Array-wise minimum value
> b.max(axis=0)	Maximum value of an array row
> b.cumsum(axis=1)	Cumulative sum of the elements
> a.mean()	Mean
> b.median()	Median
> a.corrcoef()	Correlation coefficient
> np.std(b)	Standard deviation

Sorting Arrays

> a.sort()	Sort an array
> c.sort(axis=0)	Sort the elements of an array's axis

Subsetting, Slicing, Indexing

Subsetting		
> a[2]		Select the element at the 2nd index
> b[1,2]		Select the element at row 1 column 2 (equivalent to b[1][2])
Slicing		
> a[0:2]		Select items at index 0 and 1
> b[0:2,1]		Select items at rows 0 and 1 in column 1
> b[:1]		Select all items at row 0 (equivalent to b[0:1, :])
> c[1,…] array([[[3., 2., 1.], [4., 5., 6.]]])		Same as [1,:,:]
> a[::-1] array([3, 2, 1])		Reversed array a, -1 is step
Boolean Indexing		
> a[a<2]		Select elements from a less than 2
Fancy Indexing		
> b[[1, 0, 1, 0],[0, 1, 2, 0]] array([4. , 2. , 6. , 1.5])		Select elements (1,0) , (0,1) , (1,2) and (0,0)
> b[[1, 0, 1, 0]][:,[0,1,2,0]] array([[4., 5., 6., 4.], [1.5, 2., 3., 1.5], [4., 5., 6., 4.], [1.5, 2., 3., 1.5]])		Select a subset of the matrix's rows and columns

Array Manipulation

Transposing Array	
> i = np.transpose(b)	Permute array dimensions
> i.T	
Change Array Shape	
> b.reshape(3,-2)	Reshape, but don't change data
Adding/Removing Elements	
> h.resize(shape=(2,6))	Return a new array with shape
> np.append(h,g)	Append items to an array
> np.insert(arr,index,vals,axis)	Insert items in an array
> np.delete(arr,index,axis)	Delete items from an array
Combining Arrays	
> np.concatenate((a,d),axis)	Concatenate arrays
> np.vstack((a,b))	Stack vertically (row-wise)
> np.hstack((a,b))	Stack horizontally (column-wise)
> np.column_stack((a,b))	Create stacked column-wise arrays

Python For Data Science *Cheat Sheet*

Pandas Basics

Learn Python for Data Science **Interactively** at www.DataCamp.com

Pandas

The **Pandas** library is built on NumPy and provides easy-to-use **data structures** and **data analysis** tools for the Python programming language.

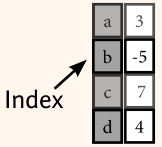
Use the following import convention:

```
>>> import pandas as pd
```

Pandas Data Structures

Series

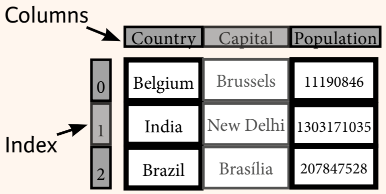
A **one-dimensional** labeled array capable of holding any data type



a	3
b	-5
c	7
d	4

```
>>> s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
```

DataFrame



	Country	Capital	Population
0	Belgium	Brussels	11190846
1	India	New Delhi	1303171035
2	Brazil	Brasília	207847528

A **two-dimensional** labeled data structure with columns of potentially different types

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
            'Capital': ['Brussels', 'New Delhi', 'Brasília'],
            'Population': [11190846, 1303171035, 207847528]}

>>> df = pd.DataFrame(data,
                      columns=['Country', 'Capital', 'Population'])
```

Asking For Help

```
>>> help(pd.Series.loc)
```

Selection

Also see NumPy Arrays

Getting

`>>> s['b']`
`-5`
	Get one element

`>>> df[1:]`
`   Country     Capital   Population`
`1  India    New Delhi   1303171035`
`2  Brazil   Brasília    207847528`
	Get subset of a DataFrame

Selecting, Boolean Indexing & Setting

By Position

`>>> df.iloc([0],[0])`
`'Belgium'`
`>>> df.iat([0],[0])`
`'Belgium'`
	Select single value by row & column

By Label

`>>> df.loc([0], ['Country'])`
`'Belgium'`
`>>> df.at([0], ['Country'])`
`'Belgium'`
	Select single value by row & column labels

By Label/Position

`>>> df.ix[2]`
`Country      Brazil`
`Capital    Brasília`
`Population  207847528`
	Select single row of subset of rows

`>>> df.ix[:,'Capital']`
`0     Brussels`
`1    New Delhi`
`2     Brasília`
	Select a single column of subset of columns

`>>> df.ix[1,'Capital']`
`'New Delhi'`
	Select rows and columns

Boolean Indexing

`>>> s[~(s > 1)]`	Series s where value is not >1
`>>> s[(s < -1) | (s > 2)]`	s where value is <-1 or >2
`>>> df[df['Population']>1200000000]`	Use filter to adjust DataFrame

Setting

`>>> s['a'] = 6`	Set index a of Series s to 6

I/O

Read and Write to CSV

```
>>> pd.read_csv('file.csv', header=None, nrows=5)
>>> df.to_csv('myDataFrame.csv')
```

Read and Write to Excel

```
>>> pd.read_excel('file.xlsx')
>>> pd.to_excel('dir/myDataFrame.xlsx', sheet_name='Sheet1')
```

Read multiple sheets from the same file

```
>>> xlsx = pd.ExcelFile('file.xls')
>>> df = pd.read_excel(xlsx, 'Sheet1')
```

Read and Write to SQL Query or Database Table

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite:///:memory:')
>>> pd.read_sql("SELECT * FROM my_table;", engine)
>>> pd.read_sql_table('my_table', engine)
>>> pd.read_sql_query("SELECT * FROM my_table;", engine)
```

`read_sql()` is a convenience wrapper around `read_sql_table()` and `read_sql_query()`

```
>>> pd.to_sql('myDf', engine)
```

Dropping

`>>> s.drop(['a', 'c'])`	Drop values from rows (axis=0)
`>>> df.drop('Country', axis=1)`	Drop values from columns(axis=1)

Sort & Rank

`>>> df.sort_index()`	Sort by labels along an axis
`>>> df.sort_values(by='Country')`	Sort by the values along an axis
`>>> df.rank()`	Assign ranks to entries

Retrieving Series/DataFrame Information

Basic Information

`>>> df.shape`	(rows,columns)
`>>> df.index`	Describe index
`>>> df.columns`	Describe DataFrame columns
`>>> df.info()`	Info on DataFrame
`>>> df.count()`	Number of non-NA values

Summary

`>>> df.sum()`	Sum of values
`>>> df.cumsum()`	Cummulative sum of values
`>>> df.min()/df.max()`	Minimum/maximum values
`>>> df.idxmin()/df.idxmax()`	Minimum/Maximum index value
`>>> df.describe()`	Summary statistics
`>>> df.mean()`	Mean of values
`>>> df.median()`	Median of values

Applying Functions

`>>> f = lambda x: x*2`	
`>>> df.apply(f)`	Apply function
`>>> df.applymap(f)`	Apply function element-wise

Data Alignment

Internal Data Alignment

NA values are introduced in the indices that don’t overlap:

```
>>> s3 = pd.Series([7, -2, 3], index=['a', 'c', 'd'])
>>> s + s3
  a     10.0
  b     NaN
  c     5.0
  d     7.0
```

Arithmetic Operations with Fill Methods

You can also do the internal data alignment yourself with the help of the fill methods:

```
>>> s.add(s3, fill_value=0)
  a    10.0
  b    -5.0
  c    5.0
  d    7.0
>>> s.sub(s3, fill_value=2)
>>> s.div(s3, fill_value=4)
>>> s.mul(s3, fill_value=3)
```

DataCamp
Learn Python for Data Science **Interactively**

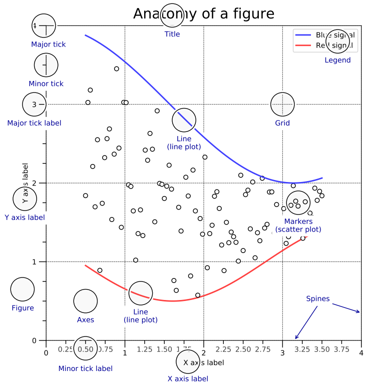
matplotlib

Cheat sheet Version 3.2

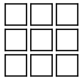
Quick start

```
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='C1')

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

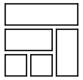
Anatomy of a figure



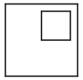
Subplots layout



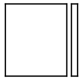
`subplot[s](cols,rows,…)`
`fig, axs = plt.subplots(3,3)`



`G = gridspec(cols,rows,…)`
`ax = G[0,:]`



`ax.inset_axes(extent)`

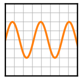


`d=make_axes_locatable(ax)`
`ax=d.new_horizontal('10%')`

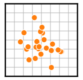
Getting help

matplotlib.org
github.com/matplotlib/matplotlib/issues
discourse.matplotlib.org
stackoverflow.com/matplotlib
gitter.im/matplotlib
twitter.com/matplotlib
Matplotlib users mailing list

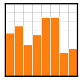
Basic plots



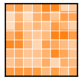
`plot([X],Y,[fmt],…)`
X, **Y**, fmt, color, marker, linestyle



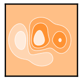
`scatter(X,Y,…)`
X, **Y**, [s]izes, [c]olors, markers, cmap



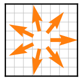
`bar[h](x,height,…)`
x, **height**, width, bottom, align, color



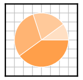
`imshow(Z,[cmap],…)`
Z, cmap, interpolation, extent, origin



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

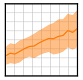


`quiver([X],[Y],U,V,…)`
X, Y, **U**, **V**, C, units, angles



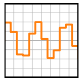
`pie(X,[explode],…)`
Z, explode, labels, colors, radius

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

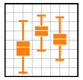


`fill[_between][x]( … )`
X, **Y1**, Y2, color, where

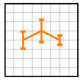
Advanced plots



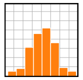
`step(X,Y,[fmt],…)`
X, **Y**, fmt, color, marker, where



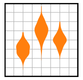
`boxplot(X,…)`
X, notch, sym, bootstrap, widths



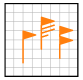
`errorbar(X,Y,xerr,yerr,…)`
X, **Y**, xerr, yerr, fmt



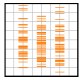
`hist(X, bins, …)`
X, bins, range, density, weights



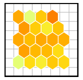
`violinplot(D,…)`
D, positions, widths, vert



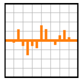
`barbs([X],[Y], U, V, …)`
X, Y, **U**, **V**, C, length, pivot, sizes



`eventplot(positions,…)`
positions, orientation, lineoffsets



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



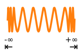
`xcorr(X,Y,…)`
X, **Y**, normed, detrend

Scales

`ax.set_[xy]scale(scale,…)`

`linear`
any values

`log`
values > 0

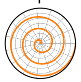


`symlog`
any values

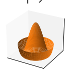
`logit`
0 < values < 1

Projections

`subplot(…,projection=p)`



p='polar'



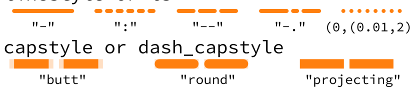
p='3d'

`p=Orthographic()`
from cartopy.crs import Cartographic

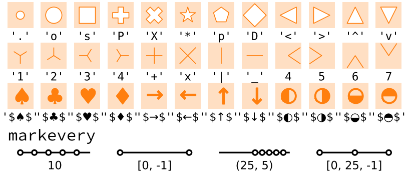
Lines

`linestyle or ls`

`capstyle or dash_capstyle`



Markers



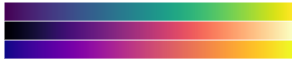
Colors

C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	'Cn'
b	g	r	c	m	y	k	w			'x'
DarkRed	Firebrick	Crimson	IndianRed	Salmon						'name'
(1,0,0)	(1,0,0,0.75)	(1,0,0,0.5)	(1,0,0,0.25)							(R,G,B[,A])
#FF0000	#FF0000BB	#FF000088	#FF000044							'#RRGGBB[AA]'
0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0 'x.y'

Colormaps

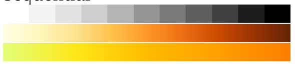
`plt.get_cmap(name)`

Uniform



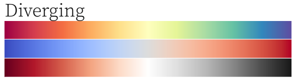
viridis
magma
plasma

Sequential



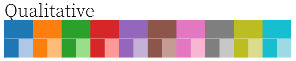
Greys
YlOrBr
Wistia

Diverging



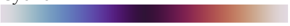
Spectral
coolwarm
RdGy

Qualitative



tab10
tab20

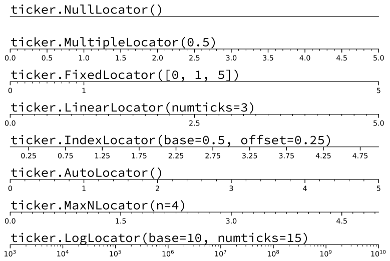
Cyclic



twilight

Tick locators

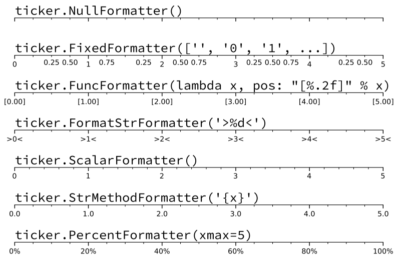
```
from matplotlib import ticker
ax.[xy]axis.set_[minor|major]_locator(locator)
```



```
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

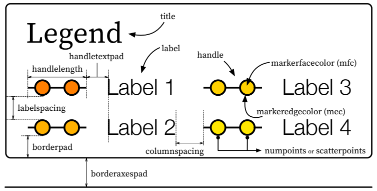
```
from matplotlib import ticker
ax.[xy]axis.set_[minor|major]_formatter(formatter)
```



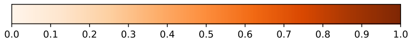
```
ticker.NullFormatter()
ticker.FixedFormatter(['', '0', '1', ...])
ticker.FuncFormatter(lambda x, pos: "[%.2f]" % x)
ticker.FormatStrFormatter('>%d<')
ticker.ScalarFormatter()
ticker.StrMethodFormatter('{x}')
ticker.PercentFormatter(xmax=5)
```

Ornaments

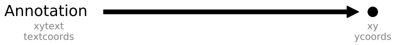
`ax.legend(…)`
handles, labels, loc, title, frameon



`ax.colorbar(…)`
mappable, ax, cax, orientation



`ax.annotate(…)`
text, **xy**, **xytext**, xycoords, textcoords, arrowprops



Event handling

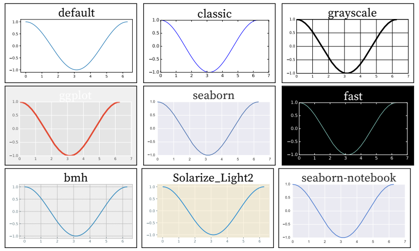
```
fig, ax = plt.subplots()
def on_click(event):
    print(event)
fig.canvas.mpl_connect(
    'button_press_event', on_click)
```

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)`



Quick reminder

```
ax.grid()
ax.patch.set_alpha(0)
ax.set_[xy]lim(vmin, vmax)
ax.set_[xy]label(label)
ax.set_[xy]ticks(list)
ax.set_[xy]ticklabels(list)
ax.set_[sup]title(title)
ax.tick_params(width=10, …)
ax.set_axis_[on|off]()

ax.tight_layout()
plt.gcf(), plt.gca()
mpl.rc('axes', linewidth=1, …)
fig.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

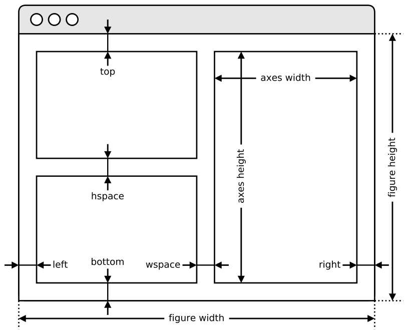
Ten Simple Rules

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

Axes adjustements

API

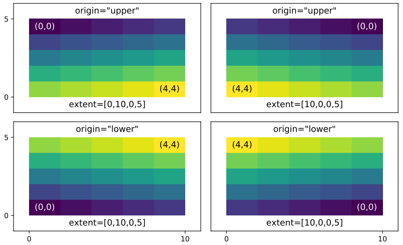
plt.**subplot_adjust**(…)



Extent & origin

API

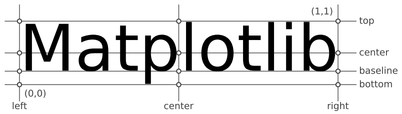
ax.**imshow**(extent=…, origin=…)



Text alignments

API

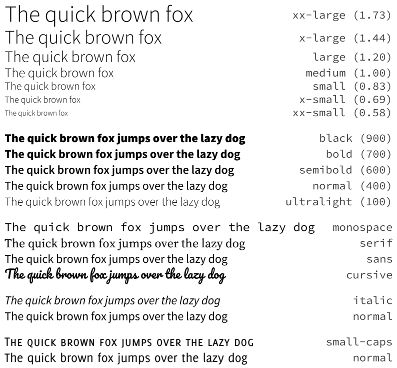
ax.**text**(…, ha=… , va=…, …)



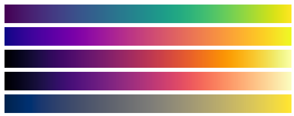
Text parameters

API

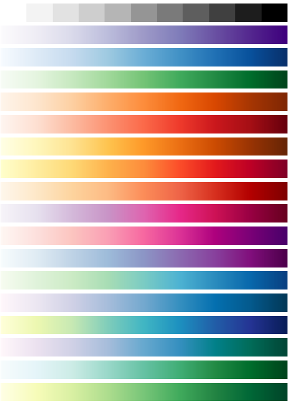
ax.**text**(…, family=… , size=…, weight = …)
ax.**text**(…, fontproperties = …)


The quick brown fox
xx-large (1.73)
The quick brown fox
x-large (1.44)
The quick brown fox
large (1.20)
The quick brown fox
medium (1.00)
The quick brown fox
small (0.83)
The quick brown fox
x-small (0.69)
The quick brown fox
xx-small (0.58)
The quick brown fox jumps over the lazy dog
black (900)
The quick brown fox jumps over the lazy dog
bold (700)
The quick brown fox jumps over the lazy dog
semibold (600)
The quick brown fox jumps over the lazy dog
normal (400)
The quick brown fox jumps over the lazy dog
ultralight (100)
The quick brown fox jumps over the lazy dog
monospace
The quick brown fox jumps over the lazy dog
serif
The quick brown fox jumps over the lazy dog
sans
The quick brown fox jumps over the lazy dog
cursive
The quick brown fox jumps over the lazy dog
italic
The quick brown fox jumps over the lazy dog
normal
THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG
small-caps
The quick brown fox jumps over the lazy dog
normal

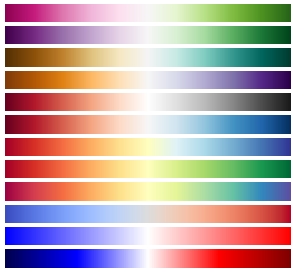
Uniform colormaps



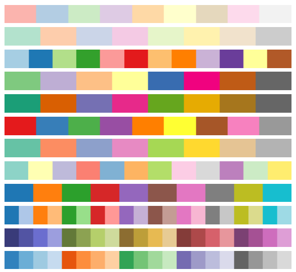
Sequential colormaps



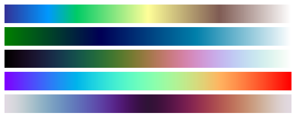
Diverging colormaps



Qualitative colormaps



Miscellaneous colormaps



Color names

API

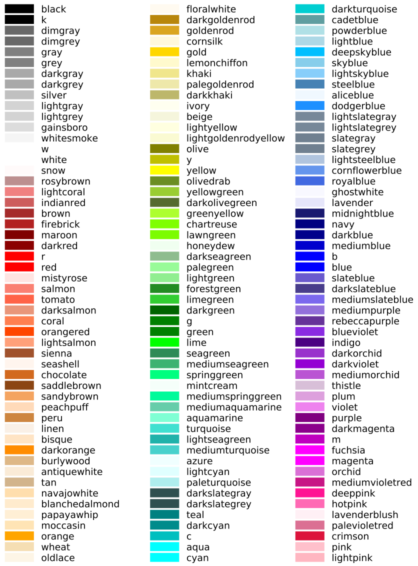
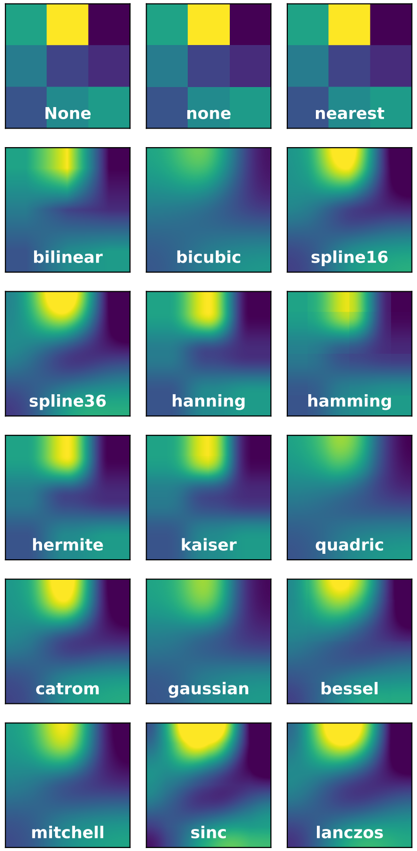
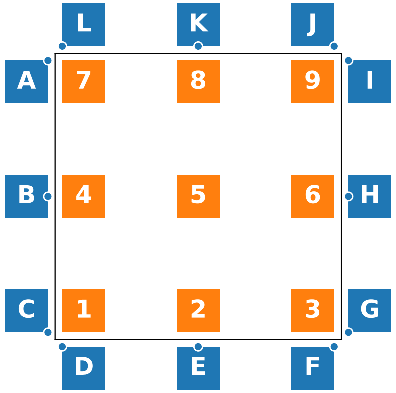


Image interpolation

API



Legend placement



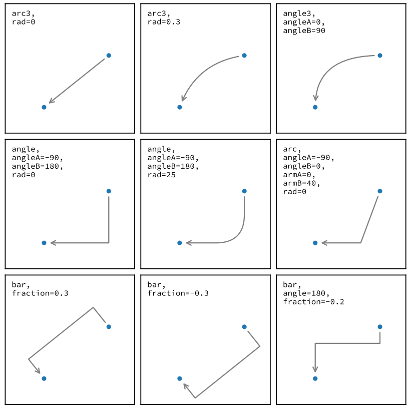
ax.**legend**(loc="string", bbox_to_anchor=(x,y))

1: lower left 2: lower center 3: lower right
4: left 5: center 6: right
7: upper left 8: upper center 9: upper right

A: upper right / (-.1,.9) B: right / (-.1,.5)
C: lower right / (-.1,.1) D: upper left / (-.1,-.1)
E: upper center / (.5,-.1) F: upper right / (.9,-.1)
G: lower left / (1.1,.1) H: left / (1.1,.5)
I: upper left / (1.1,.9) J: lower right / (.9,1.1)
K: lower center / (.5,1.1) L: lower left / (.1,1.1)

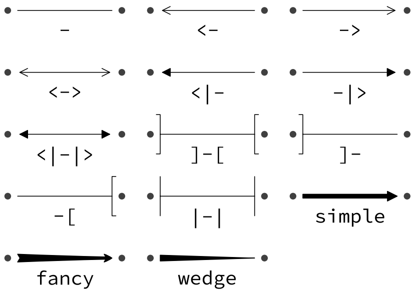
Annotation connection styles

API



Annotation arrow styles

API



How do I …

… resize a figure?
→ fig.set_size_inches(w,h)
… save a figure?
→ fig.savefig("figure.pdf")
… save a transparent figure?
→ fig.savefig("figure.pdf", transparent=True)
… clear a figure?
→ ax.clear()
… close all figures?
→ plt.close("all")
… remove ticks?
→ ax.set_xticks([])
… remove tick labels ?
→ ax.set_[xy]ticklabels([])
… rotate tick labels ?
→ ax.set_[xy]ticks(rotation=90)
… hide top spine?
→ ax.spines['top'].set_visible(False)
… hide legend border?
→ ax.legend(frameon=False)
… show error as shaded region?
→ ax.fill_between(X, Y+error, Y-error)
… draw a rectangle?
→ ax.add_patch(plt.Rectangle((0, 0),1,1)
… draw a vertical line?
→ ax.axvline(x=0.5)
… draw outside frame?
→ ax.plot(…, clip_on=False)
… use transparency?
→ ax.plot(…, alpha=0.25)
… convert an RGB image into a gray image?
→ gray = 0.2989*R+0.5870*G+0.1140*B
… set figure background color?
→ fig.patch.set_facecolor("grey")
… get a reversed colormap?
→ plt.get_cmap("viridis_r")
… get a discrete colormap?
→ plt.get_cmap("viridis", 10)
… show a figure for one second?
→ fig.show(block=False), time.sleep(1)

Performance tips

```
scatter(X, Y)                                   slow
plot(X, Y, marker="o", ls="")                   fast
for i in range(n): plot(X[i])                   slow
plot(sum([x+[None] for x in X],[]))             fast
cla(), imshow(…), canvas.draw()                 slow
im.set_data(…), canvas.draw()                   fast
```

Beyond Matplotlib

Seaborn: Statistical Data Visualization
Cartopy: Geospatial Data Processing
yt: Volumetric data Visualization
mpld3: Bringing Matplotlib to the browser
Datashader: Large data processing pipeline
plotnine: A Grammar of Graphics for Python

NUMFOCUS
OPEN CODE = BETTER SCIENCE