

Scientific Python Cheatsheet

Fork me on GitHub

- [Scientific Python Cheatsheet](#)
 - [Pure Python](#)
 - [Types](#)
 - [Lists](#)
 - [Dictionaries](#)
 - [Strings](#)
 - [Operators](#)
 - [Control Flow](#)
 - [Functions, Classes, Generators, Decorators](#)
 - [IPython](#)
 - [console](#)
 - [debugger](#)
 - [command line](#)
 - [NumPy](#)
 - [array initialization](#)
 - [indexing](#)
 - [array properties and operations](#)
 - [boolean arrays](#)
 - [elementwise operations and math functions](#)
 - [inner / outer products](#)
 - [reading/ writing files](#)
 - [interpolation, integration, optimization](#)
 - [fft](#)
 - [rounding](#)
 - [random variables](#)
 - [Matplotlib](#)
 - [figures and axes](#)
 - [figures and axes properties](#)
 - [plotting routines](#)
 - [Scipy](#)
 - [interpolation](#)
 - [integration](#)
 - [Pandas](#)
 - [data structures](#)
 - [DataFrame](#)

Pure Python

Types

```
a = 2           # integer
b = 5.0         # float
c = 8.3e5       # exponential
d = 1.5 + 0.5j  # complex
e = 4 > 5       # boolean
f = 'word'      # string
```

Lists

```
a = ['red', 'blue', 'green']      # manually initialization
b = list(range(5))                # initialize from iterable
c = [nu**2 for nu in b]            # list comprehension
```

```

d = [nu**2 for nu in b if nu < 3] # conditioned list comprehension
e = c[0]                          # access element
f = c[1:2]                        # access a slice of the list
g = ['re', 'bl'] + ['gr']        # list concatenation
h = ['re'] * 5                    # repeat a list
['re', 'bl'].index('re')         # returns index of 're'
're' in ['re', 'bl']             # true if 're' in list
sorted([3, 2, 1])                # returns sorted list

```

Dictionaries

```

a = {'red': 'rouge', 'blue': 'bleu'} # dictionary
b = a['red']                          # translate item
c = [value for key, value in a.items()] # loop through contents
d = a.get('yellow', 'no translation found') # return default
e = a.setdefault('extra', []).append('cyan') # init key with default

```

Strings

```

a = 'red'                            # assignment
char = a[2]                          # access individual characters
'red ' + 'blue'                      # string concatenation
'1, 2, three'.split(',')              # split string into list
'.'.join(['1', '2', 'three'])         # concatenate list into string

```

Operators

```

a = 2                                # assignment
a += 1 (*=, /=)                     # change and assign
3 + 2                                # addition
3 / 2                                # integer (python2) or float (python3) division
3 // 2                               # integer division
3 * 2                                # multiplication
3 ** 2                               # exponent
3 % 2                                # remainder
abs(a)                               # absolute value
1 == 1                               # equal
2 > 1                                # larger
2 < 1                                # smaller
1 != 2                               # not equal
1 != 2 and 2 < 3                     # logical AND
1 != 2 or 2 < 3                      # logical OR
not 1 == 2                           # logical NOT
'a' in b                             # test if a is in b
a is b                               # test if objects point to the same memory (id)

```

Control Flow

```

# if/elif/else
a, b = 1, 2
if a + b == 3:
    print('True')
elif a + b == 1:
    print('False')
else:
    print('?')

# for
a = ['red', 'blue', 'green']
for color in a:
    print(color)

# while
number = 1
while number < 10:
    print(number)
    number += 1

```

```
# break
number = 1
while True:
    print(number)
    number += 1
    if number > 10:
        break

# continue
for i in range(20):
    if i % 2 == 0:
        continue
    print(i)
```

Functions, Classes, Generators, Decorators

```
# Function groups code statements and possibly
# returns a derived value
def myfunc(a1, a2):
    return a1 + a2
```

```
x = myfunc(a1, a2)
```

```
# Class groups attributes (data)
# and associated methods (functions)
class Point(object):
    def __init__(self, x):
        self.x = x
    def __call__(self):
        print(self.x)
```

```
x = Point(3)
```

```
# Generator iterates without
# creating all values at ones
def firstn(n):
    num = 0
    while num < n:
        yield num
        num += 1
```

```
x = [i for i in firstn(10)]
```

```
# Decorator can be used to modify
# the behaviour of a function
```

```
class myDecorator(object):
    def __init__(self, f):
        self.f = f
    def __call__(self):
        print("call")
        self.f()
```

```
@myDecorator
def my_func():
    print('func')
```

```
my_func()
```

IPython

console

```
<object>? # Information about the object
<object>.<TAB> # tab completion
```

```
# measure runtime of a function:
```

```
%timeit range(1000)
100000 loops, best of 3: 7.76 us per loop

# run scripts and debug
%run
%run -d # run in debug mode
%run -t # measures execution time
%run -p # runs a profiler
%debug # jumps to the debugger after an exception

%pdb # run debugger automatically on exception

# examine history
%history
%history ~1/1-5 # lines 1-5 of last session

# run shell commands
!make # prefix command with "!"

# clean namespace
%reset
```

debugger

```
n # execute next line
b 42 # set breakpoint in the main file at line 42
b myfile.py:42 # set breakpoint in 'myfile.py' at line 42
c # continue execution
l # show current position in the code
p data # print the 'data' variable
pp data # pretty print the 'data' variable
s # step into subroutine
a # print arguments that a function received
pp locals() # show all variables in local scope
pp globals() # show all variables in global scope
```

command line

```
ipython --pdb -- myscript.py argument1 --option1 # debug after exception
ipython -i -- myscript.py argument1 --option1 # console after finish
```

NumPy (import numpy as np)

array initialization

```
np.array([2, 3, 4]) # direct initialization
np.empty(20, dtype=np.float32) # single precision array of size 20
np.zeros(200) # initialize 200 zeros
np.ones((3,3), dtype=np.int32) # 3 x 3 integer matrix with ones
np.eye(200) # ones on the diagonal
np.zeros_like(a) # array with zeros and the shape of a
np.linspace(0., 10., 100) # 100 points from 0 to 10
np.arange(0, 100, 2) # points from 0 to <100 with step 2
np.logspace(-5, 2, 100) # 100 log-spaced from 1e-5 -> 1e2
np.copy(a) # copy array to new memory
```

indexing

```
a = np.arange(100) # initialization with 0 - 99
a[:3] = 0 # set the first three indices to zero
a[2:5] = 1 # set indices 2-4 to 1
a[start:stop:step] # general form of indexing/slicing
a[None, :] # transform to column vector
a[[1, 1, 3, 8]] # return array with values of the indices
a = a.reshape(10, 10) # transform to 10 x 10 matrix
a.T # return transposed view
```

```
b = np.transpose(a, (1, 0)) # transpose array to new axis order
a[a < 2]                    # values with elementwise condition
```

array properties and operations

```
a.shape                # a tuple with the lengths of each axis
len(a)                 # length of axis 0
a.ndim                 # number of dimensions (axes)
a.sort(axis=1)         # sort array along axis
a.flatten()            # collapse array to one dimension
a.conj()               # return complex conjugate
a.astype(np.int16)     # cast to integer
np.argmax(a, axis=1)   # return index of maximum along a given axis
np.cumsum(a)           # return cumulative sum
np.any(a)              # True if any element is True
np.all(a)              # True if all elements are True
np.argsort(a, axis=1)  # return sorted index array along axis
```

boolean arrays

```
a < 2                  # returns array with boolean values
(a < 2) & (b > 10)     # elementwise logical and
(a < 2) | (b > 10)     # elementwise logical or
~a                     # invert boolean array
```

elementwise operations and math functions

```
a * 5                  # multiplication with scalar
a + 5                  # addition with scalar
a + b                  # addition with array b
a / b                  # division with b (np.NaN for division by zero)
np.exp(a)              # exponential (complex and real)
np.power(a, b)         # a to the power b
np.sin(a)              # sine
np.cos(a)              # cosine
np.arctan2(a, b)       # arctan(a/b)
np.arcsin(a)           # arcsin
np.radians(a)          # degrees to radians
np.degrees(a)          # radians to degrees
np.var(a)              # variance of array
np.std(a, axis=1)      # standard deviation
```

inner / outer products

```
np.dot(a, b)           # inner product: ami bin
np.einsum('ij,kj->ik', a, b) # einstein summation convention
np.sum(a, axis=1)      # sum over axis 1
np.abs(a)              # return absolute values
a[None, :] + b[:, None] # outer sum
a[None, :] * b[:, None] # outer product
np.outer(a, b)         # outer product
np.sum(a * a.T)        # matrix norm
```

reading/ writing files

```
np.fromfile(fname/fobject, dtype=np.float32, count=5) # binary data from file
np.loadtxt(fname/fobject, skiprows=2, delimiter=',') # ascii data from file
np.savetxt(fname/fobject, array, fmt='%.5f')          # write ascii data
np.tofile(fname/fobject)                              # write (C) binary data
```

interpolation, integration, optimization

```
np.trapz(a, x=x, axis=1) # integrate along axis 1
np.interp(x, xp, yp)     # interpolate function xp, yp at points x
```

```
np.linalg.lstsq(a, b)      # solve  $a \cdot x = b$  in least square sense
```

fft

```
np.fft.fft(a)              # complex fourier transform of a
f = np.fft.fftfreq(len(a)) # fft frequencies
np.fft.fftshift(f)         # shifts zero frequency to the middle
np.fft.rfft(a)             # real fourier transform of a
np.fft.rfftfreq(len(a))    # real fft frequencies
```

rounding

```
np.ceil(a)    # rounds to nearest upper int
np.floor(a)   # rounds to nearest lower int
np.round(a)   # rounds to nearest int
```

random variables

```
from np.random import normal, seed, rand, uniform, randint
normal(loc=0, scale=2, size=100) # 100 normal distributed
seed(23032)                      # resets the seed value
rand(200)                       # 200 random numbers in [0, 1)
uniform(1, 30, 200)             # 200 random numbers in [1, 30)
randint(1, 16, 300)            # 300 random integers in [1, 16)
```

Matplotlib (import matplotlib.pyplot as plt)

figures and axes

```
fig = plt.figure(figsize=(5, 2)) # initialize figure
ax = fig.add_subplot(3, 2, 2)    # add second subplot in a 3 x 2 grid
fig, axes = plt.subplots(5, 2, figsize=(5, 5)) # fig and 5 x 2 ndarray of axes
ax = fig.add_axes([left, bottom, width, height]) # add custom axis
```

figures and axes properties

```
fig.suptitle('title')          # big figure title
fig.subplots_adjust(bottom=0.1, right=0.8, top=0.9, wspace=0.2,
                    hspace=0.5) # adjust subplot positions
fig.tight_layout(pad=0.1, h_pad=0.5, w_pad=0.5,
                rect=None)      # adjust subplots to fit into fig
ax.set_xlabel('xbla')          # set xlabel
ax.set_ylabel('ybla')          # set ylabel
ax.set_xlim(1, 2)              # sets x limits
ax.set_ylim(3, 4)              # sets y limits
ax.set_title('blabla')         # sets the axis title
ax.set(xlabel='bla')           # set multiple parameters at once
ax.legend(loc='upper center')  # activate legend
ax.grid(True, which='both')    # activate grid
bbox = ax.get_position()       # returns the axes bounding box
bbox.x0 + bbox.width           # bounding box parameters
```

plotting routines

```
ax.plot(x,y, '-o', c='red', lw=2, label='bla') # plots a line
ax.scatter(x,y, s=20, c=color)                 # scatter plot
ax.pcolormesh(xx, yy, zz, shading='gouraud')    # fast colormesh
ax.colormesh(xx, yy, zz, norm=norm)            # slower colormesh
ax.contour(xx, yy, zz, cmap='jet')             # contour lines
ax.contourf(xx, yy, zz, vmin=2, vmax=4)        # filled contours
n, bins, patch = ax.hist(x, 50)               # histogram
ax.imshow(matrix, origin='lower',
            extent=(x1, x2, y1, y2))          # show image
ax.specgram(y, FS=0.1, noverlap=128,
```

```
        scale='linear') #plot a spectrogram
ax.text(x, y, string, fontsize=12, color='m') # write text
```

Scipy (import scipy as sci)

interpolation

```
from scipy.ndimage import map_coordinates # interpolates data
pts_new = map_coordinates(data, float_indices, # at index positions
                          order=3)
```

Integration

```
from scipy.integrate import quad # definite integral of python
value = quad(func, low_lim, up_lim) # function/method
```

Pandas (import pandas as pd)

Data structures

```
s = pd.Series(np.random.rand(1000), index=range(1000)) # series
index = pd.date_range("13/06/2016", periods=1000) # time index
df = pd.DataFrame(np.zeros((1000, 3)), index=index,
                  columns=["A", "B", "C"]) # DataFrame
```

DataFrame

```
df = pd.read_csv("filename.csv") # read and load CSV file in a DataFrame
print(df[:2]) # print first 2 lines of the DataFrame
raw = df.values # get raw data out of DataFrame object
cols = df.columns # get list of columns headers
```

[Scientific python cheat sheet](#) is maintained by [IPGP](#). This page was generated by [GitHub Pages](#) using the [Cayman theme](#) by [Jason Long](#).