Fork me on GitHub

Scientific Python Cheatsheet

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 iteratable
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 = c[-1]                          # access last element
h = ['re', 'bl'] + ['gr']          # list concatenation
i = ['re'] * 5                     # repeat a list
['re', 'bl'].index('re')           # returns index of 're'
a.append('yellow')                 # add new element to end of list
a.extend(b)                        # add elements from list `b` to end of list `a`
a.insert(1, 'yellow')              # insert element in specified position
're' in ['re', 'bl']               # true if 're' in list
'fi' not in ['re', 'bl']           # true if 'fi' not in list
sorted([3, 2, 1])                  # returns sorted list
a.pop(2)                           # remove and return item at index (default last)
```

Dictionaries

```
a = {'red': 'rouge', 'blue': 'bleu'}         # dictionary
b = a['red']                                 # translate item
'red' in a                                   # true if dictionary a contains key 'red'
c = [value for key, value in a.items()]      # loop through contents
d = a.get('yellow', 'no translation found')  # return default
a.setdefault('extra', []).append('cyan')     # init key with default
a.update({'green': 'vert', 'brown': 'brun'}) # update dictionary by data from another one
a.keys()                                     # get list of keys
a.values()                                   # get list of values
a.items()                                    # get list of key-value pairs
del a['red']                                 # delete key and associated with it value
a.pop('blue')                                # remove specified key and return the corresponding value
```

Sets

```
a = {1, 2, 3}                                # initialize manually
b = set(range(5))                            # initialize from iteratable
a.add(13)                                    # add new element to set
a.discard(13)                                # discard element from set
a.update([21, 22, 23])                       # update set with elements from iterable
a.pop()                                      # remove and return an arbitrary set element
2 in {1, 2, 3}                               # true if 2 in set
5 not in {1, 2, 3}                           # true if 5 not in set
a.issubset(b)                                # test whether every element in a is in b
a <= b                                       # issubset in operator form
a.issuperset(b)                              # test whether every element in b is in a
a >= b                                       # issuperset in operator form
a.intersection(b)                            # return the intersection of two sets as a new set
a.difference(b)                              # return the difference of two or more sets as a new set
a - b                                        # difference in operator form
a.symmetric_difference(b)                    # return the symmetric difference of two sets as a new set
a.union(b)                                   # return the union of sets as a new set
c = frozenset()                              # the same as set but immutable
```

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 once
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_funct():
    print('func')

my_funct()
```

IPython

console

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

# run scripts / profile / debug
%run myscript.py

%timeit range(1000)         # measure runtime of statement
%run -t  myscript.py        # measure script execution time

%prun <statement>           # run statement with profiler
%prun -s <key> <statement>  # sort by key, e.g. "cumulative" or "calls"
%run -p  myfile.py          # profile script

%run -d myscript.py         # run script in debug mode
```

```
%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

# run code from clipboard
%paste
```

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[:-3] = 2                  # set all but last three elements to 2
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
a.tolist()             # convert (possibly multidimensional) array to list
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
np.where(cond)         # return indices where cond is True
np.where(cond, x, y)   # return elements from x or y depending on cond
```

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: a_mi b_in
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
```

linear algebra/ matrix math

```
evals, evecs = np.linalg.eig(a)      # Find eigenvalues and eigenvectors
evals, evecs = np.linalg.eigh(a)     # np.linalg.eig for hermitian matrix
```

reading/ writing files

```
np.loadtxt(fname/fobject, skiprows=2, delimiter=',')   # ascii data from file
np.savetxt(fname/fobject, array, fmt='%.5f')           # write ascii data
np.fromfile(fname/fobject, dtype=np.float32, count=5)  # binary data from file
np.tofile(fname/fobject)                               # write (C) binary data
np.save(fname/fobject, array)                          # save as numpy binary (.npy)
np.load(fname/fobject, mmap_mode='c')                  # load .npy file (memory mapped)
```

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 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 neares 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
fig.savefig('out.png')            # save png image
fig, axes = plt.subplots(5, 2, figsize=(5, 5)) # fig and 5 x 2 nparray of axes
ax = fig.add_subplot(3, 2, 2)     # add second subplot in a 3 x 2 grid
ax = plt.subplot2grid((2, 2), (0, 0), colspan=2)  # multi column/row axis
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

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

# simple 1d interpolator with axis argument:
from scipy.interpolate import interp1d
interpolator = interp1d(x, y, axis=2, fill_value=0., bounds_error=False)
y_new = interpolator(x_new)
```

Integration

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

linear algebra

```
from scipy import linalg
evals, evecs = linalg.eig(a)      # Find eigenvalues and eigenvectors
evals, evecs = linalg.eigh(a)     # linalg.eig for hermitian matrix
b = linalg.expm(a)                # Matrix exponential
c = linalg.logm(a)                # Matrix logarithm
```

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
raw = df.values                    # get raw data out of DataFrame object
cols = df.columns                  # get list of columns headers
df.dtypes                          # get data types of all columns
df.head(5)                         # get first 5 rows
df.describe()                      # get basic statisitics for all columns
df.index                           # get index column range

#column slicin
# (.loc[] and .ix[] are inclusive of the range of values selected)
df.col_name                         # select column values as a series by column name (not optimized)
df[['col_name']]                    # select column values as a dataframe by column name (not optimized)
df.loc[:, 'col_name']               # select column values as a series by column name
df.loc[:, ['col_name']]             # select column values as a dataframe by column name
df.iloc[:, 0]                       # select by column index
df.iloc[:, [0]]                     # select by column index, but as a dataframe
df.ix[:, 'col_name']                # hybrid approach with column name
df.ix[:, 0]                         # hybrid approach with column index

# row slicin
print(df[:2])                      # print first 2 rows of the dataframe
df.iloc[0:2, :]                    # select first 2 rows of the dataframe
df.loc[0:2,'col_name']             # select first 3 rows of the dataframe
df.loc[0:2, ['col_name1', 'col_name3', 'col_name6']]    # select first 3 rows of the 3 different columns
df.iloc[0:2,0:2]                   # select fisrt 3 rows and first 3 columns
# Again, .loc[] and .ix[] are inclusive

# Dicin
df[ df.col_name < 7 ]                            # select all rows where col_name < 7
df[ (df.col_name1 < 7) & (df.col_name2 == 0) ]       # combine multiple boolean indexing conditionals using bit-wise logical operators.
                                                     # Regular Python boolean operators (and, or) cannot be used here.
                                                     # Be sure to encapsulate each conditional in parenthesis to make this work.
df[df.recency < 7] = -100                        # writing to slice
```

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