

Scientific Python Cheatsheet

Pure Python

Types

```
a = 2 # integer
b = 5.0 # float
c = 8.3e5 # exponential
d = 1.5 + 0.5j # complex
e = 3 > 4 # boolean
f = "word" # string
```

Lists

```
a = ["red", "blue", "green"] # manually initialization
b = range(5) # initialization through a function
c = [nu**2 for nu in b] # initialize through list comprehension
d = [nu**2 for nu in b if b < 3] # list comprehension with condition
e = c[0] # access element
f = e[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
z = ["red"] + ["green", "blue"] # list concatenation
```

Dictionaries

```
a = {"red": "rouge", "blue": "bleu", "green": "vert"} # dictionary
b = a["red"] # translate item
c = [value for key, value in b.items()] # loop through contents
d = a.get("yellow", "no translation found") # return 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 division (python2) or float division (python3)
3 // 2 # integer division
3 * 2 # multiplication
3 ** 2 # exponent
3 % 2 # remainder
abs() # 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

```
def myfunc(a1, a2):
    return x
```

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

Class

```
class Point(object):
    def __init__(self, x):
        self.x = x
    def __call__(self):
        print self.x
```

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

Generators

```
def firstn(n):
    num = 0
    while num < n:
        yield num
        num += 1

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

Decorators

```
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()
```

NumPy

array initialization

```
np.array([2, 3, 4]) # direct initialization
np.empty(20, dtype=np.float32) # single precision array with 20 entries
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) # returns array with zeros and the same shape as a
np.linspace(0., 10., 100) # 100 points from 0 to 10
np.arange(0, 100, 2) # points from 0 to <100 with step width 2
np.logspace(-5, 2, 100) # 100 logarithmically spaced points between 1e-5 and 1e2
np.copy(a) # copy array to new memory
```

reading/ writing files

```
np.fromfile(fname/object, dtype=np.float32, count=5) # read binary data from file
np.loadtxt(fname/object, skiprows=2, delimiter=",") # read ascii data from file
```

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=2) # 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
```

indexing

```
a = np.arange(100) # initialization with 0 - 99
a[: 3] = 0 # set the first three indices to zero
a[1: 5] = 1 # set indices 1-4 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
np.transpose(a, (2, 1, 0)) # transpose array to new axis order
a[a < 2] # returns array that fulfills elementwise condition
```

boolean arrays

```
a < 2 # returns array with boolean values
np.logical_and(a < 2, b > 10) # elementwise logical and
np.logical_or(a < 2, b > 10) # elementwise logical or
~a # invert boolean array
np.invert(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.sin(a) # sine
```

```

np.cos(a) # cosine
np.arctan2(y,x) # arctan(y/x)
np.arcsin(x) # 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 matrix product: a_mi b_in
np.einsum("ijkl,klmn->ijmn", a, b) # einstein summation convention
np.sum(a, axis=1) # sum over axis 1
np.abs(a) # return array with 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

```

interpolation, integration

```

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

```

fft

```

np.fft.fft(y) # complex fourier transform of y
np.fft.fftfreqs(len(y)) # fft frequencies for a given length
np.fft.fftshift(freqs) # shifts zero frequency to the middle
np.fft.rfft(y) # real fourier transform of y
np.fft.rfftfreqs(len(y)) # real fft frequencies for a given length

```

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

```

np.random.normal(loc=0, scale=2, size=100) # 100 normal distributed random numbers
np.random.seed(23032) # resets the seed value

```

```

np.random.rand(200) # 200 random numbers in [0, 1)
np.random.uniform(1, 30, 200) # 200 random numbers in [1, 30)
np.random.random_integers(1, 15, 300) # 300 random integers between [1, 10]

```

Matplotlib

figures and axes

```

fig = plt.figure(figsize=(5, 2), facecolor="black") # 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)) # return fig and array of axes in a 5 x 2 grid
ax = fig.add_axes([left, bottom, width, height]) # manually add axes at a certain position

```

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 perfectly into fig
ax.set_xlabel() # set xlabel
ax.set_ylabel() # 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 function
ax.colormesh(xx,yy,zz, norm=norm) # slower colormesh function
ax.contour(xx,yy,zz, cmap="jet") # contour line plot
ax.contourf(xx,yy,zz, vmin=2, vmax=4) # filled contours plot
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

```