Things I Wish I Knew About Python Four Years Ago, Or

How to avoid doing any actual coding of your own

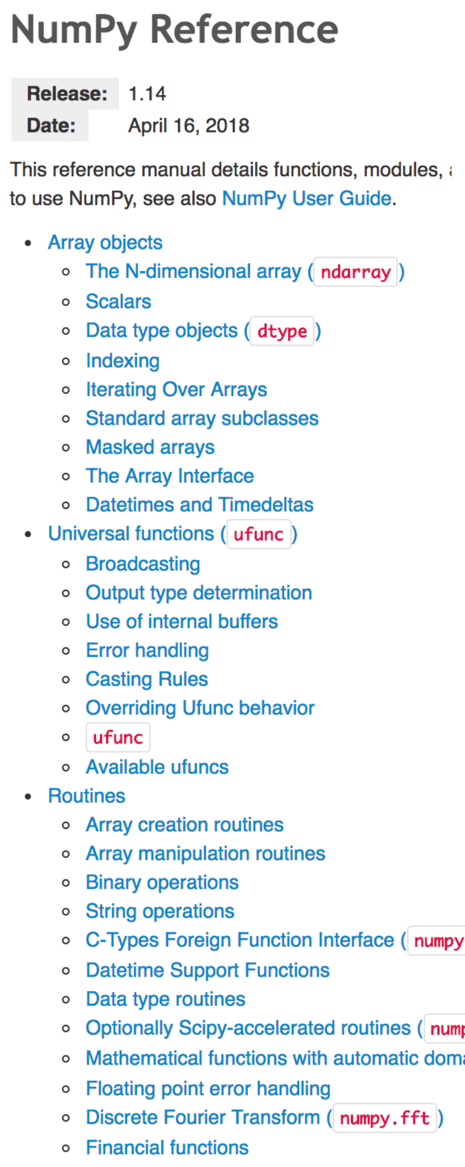
Python ~~2~~ vs 3

~~Python 2~~	Python 3
xrange(0, 5)	range(0, 5)
3 / 2 = 1	3 / 2 = 1.5
print “hi”	print(“hi”)

from __future__ import division, print

Never Doing Any Work of Your Own

NumPy



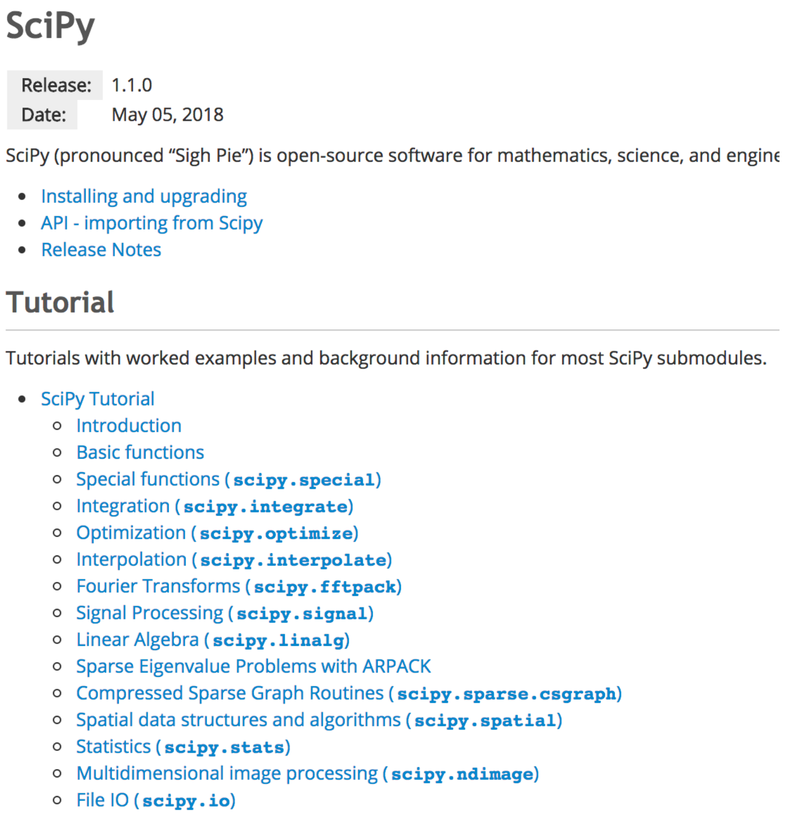
NumPy Reference

Release: 1.14
Date: April 16, 2018

This reference manual details functions, modules, a
to use NumPy, see also NumPy User Guide.

- Array objects
  - The N-dimensional array (`ndarray`)
  - Scalars
  - Data type objects (`dtype`)
  - Indexing
  - Iterating Over Arrays
  - Standard array subclasses
  - Masked arrays
  - The Array Interface
  - Datetimes and Timedeltas
- Universal functions (`ufunc`)
  - Broadcasting
  - Output type determination
  - Use of internal buffers
  - Error handling
  - Casting Rules
  - Overriding Ufunc behavior
  - `ufunc`
  - Available ufuncs
- Routines
  - Array creation routines
  - Array manipulation routines
  - Binary operations
  - String operations
  - C-Types Foreign Function Interface (`numpy.ctypeslib`)
  - Datetime Support Functions
  - Data type routines
  - Optionally Scipy-accelerated routines (`numpy.dual`)
  - Mathematical functions with automatic domain (`numpy.emath`)
  - Floating point error handling
  - Discrete Fourier Transform (`numpy.fft`)
  - Financial functions

Scipy



SciPy

Release: 1.1.0
Date: May 05, 2018

SciPy (pronounced "Sigh Pie") is open-source software for mathematics, science, and engine

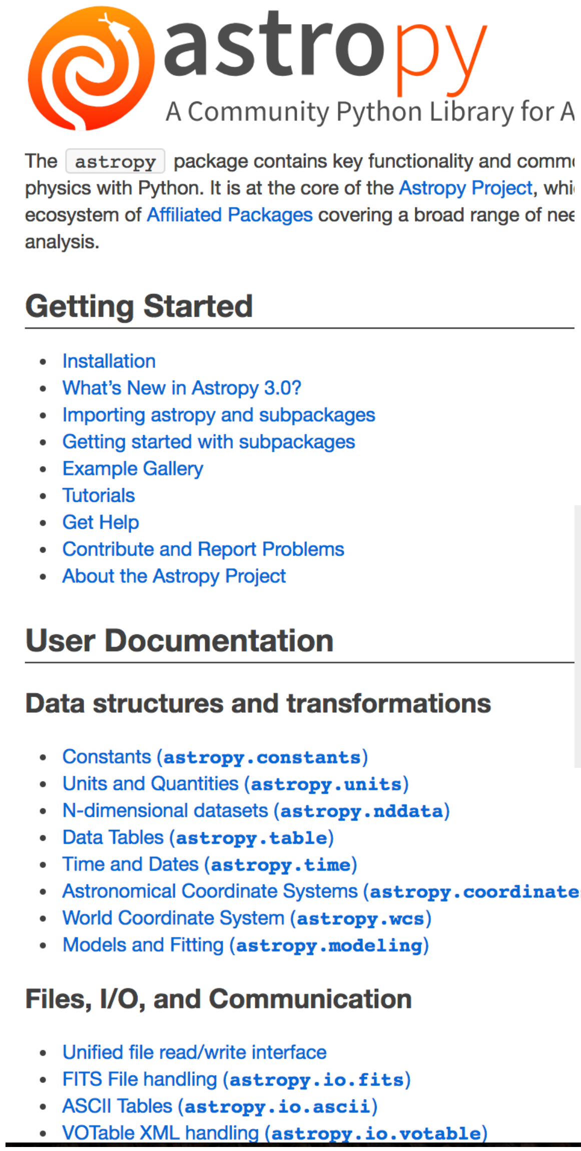
- Installing and upgrading
- API - importing from Scipy
- Release Notes

Tutorial

Tutorials with worked examples and background information for most SciPy submodules.

- SciPy Tutorial
  - Introduction
  - Basic functions
  - Special functions (`scipy.special`)
  - Integration (`scipy.integrate`)
  - Optimization (`scipy.optimize`)
  - Interpolation (`scipy.interpolate`)
  - Fourier Transforms (`scipy.fftpack`)
  - Signal Processing (`scipy.signal`)
  - Linear Algebra (`scipy.linalg`)
  - Sparse Eigenvalue Problems with ARPACK
  - Compressed Sparse Graph Routines (`scipy.sparse.csgraph`)
  - Spatial data structures and algorithms (`scipy.spatial`)
  - Statistics (`scipy.stats`)
  - Multidimensional image processing (`scipy.ndimage`)
  - File IO (`scipy.io`)

astropy



astropy
A Community Python Library for A

The `astropy` package contains key functionality and comm
physics with Python. It is at the core of the Astropy Project, whi
ecosystem of Affiliated Packages covering a broad range of nee
analysis.

Getting Started

- Installation
- What's New in Astropy 3.0?
- Importing astropy and subpackages
- Getting started with subpackages
- Example Gallery
- Tutorials
- Get Help
- Contribute and Report Problems
- About the Astropy Project

User Documentation

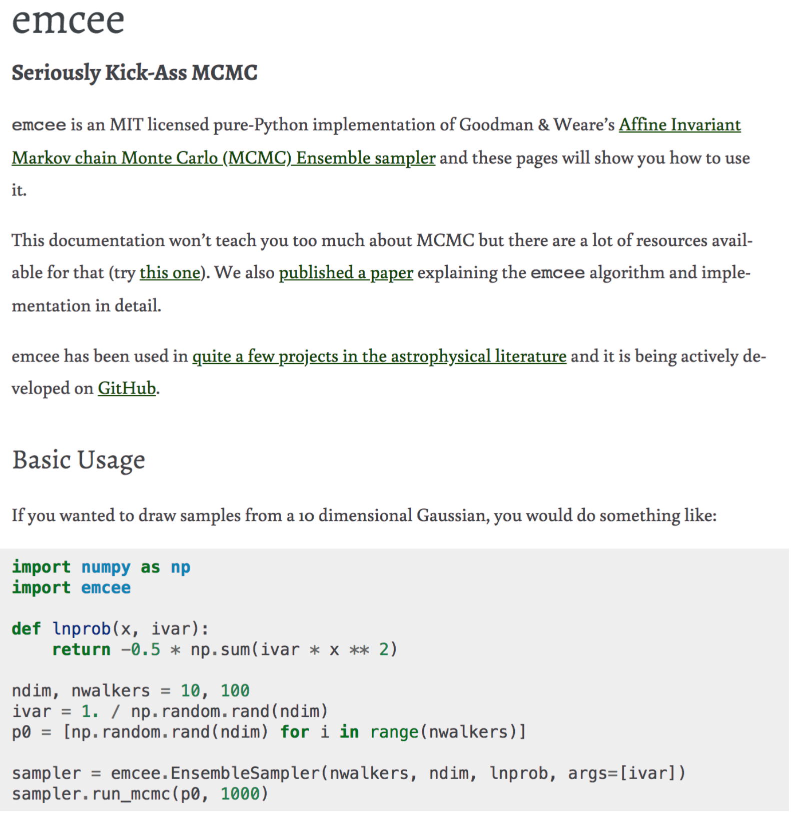
Data structures and transformations

- Constants (`astropy.constants`)
- Units and Quantities (`astropy.units`)
- N-dimensional datasets (`astropy.nddata`)
- Data Tables (`astropy.table`)
- Time and Dates (`astropy.time`)
- Astronomical Coordinate Systems (`astropy.coordinates`)
- World Coordinate System (`astropy.wcs`)
- Models and Fitting (`astropy.modeling`)

Files, I/O, and Communication

- Unified file read/write interface
- FITS File handling (`astropy.io.fits`)
- ASCII Tables (`astropy.io.ascii`)
- VOTable XML handling (`astropy.io.votable`)

emcee



emcee

Seriously Kick-Ass MCMC

emcee is an MIT licensed pure-Python implementation of Goodman & Weare's Affine Invariant Markov chain Monte Carlo (MCMC) Ensemble sampler and these pages will show you how to use it.

This documentation won't teach you too much about MCMC but there are a lot of resources available for that (try this one). We also published a paper explaining the emcee algorithm and implementation in detail.

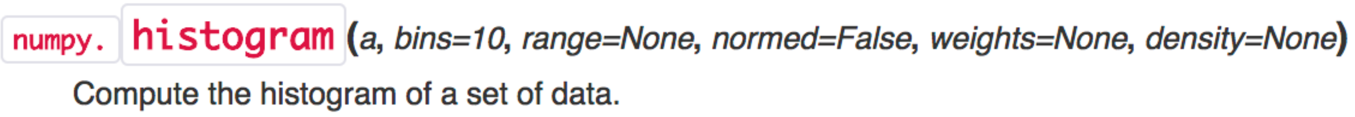
emcee has been used in quite a few projects in the astrophysical literature and it is being actively developed on GitHub.

Basic Usage

If you wanted to draw samples from a 10 dimensional Gaussian, you would do something like:

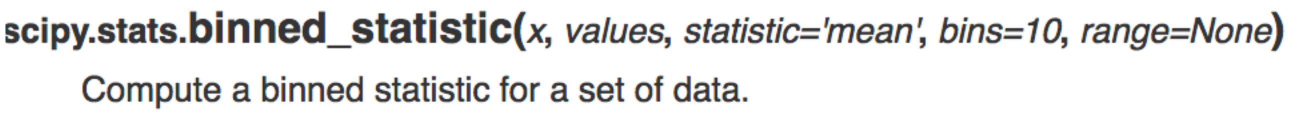
```
import numpy as np
import emcee

def lnprob(x, ivar):
    return -0.5 * np.sum(ivar * x ** 2)

ndim, nwalkers = 10, 100
ivar = 1. / np.random.rand(ndim)
p0 = [np.random.rand(ndim) for i in range(nwalkers)]

sampler = emcee.EnsembleSampler(nwalkers, ndim, lnprob, args=[ivar])
sampler.run_mcmc(p0, 1000)
```

Maths Done For You



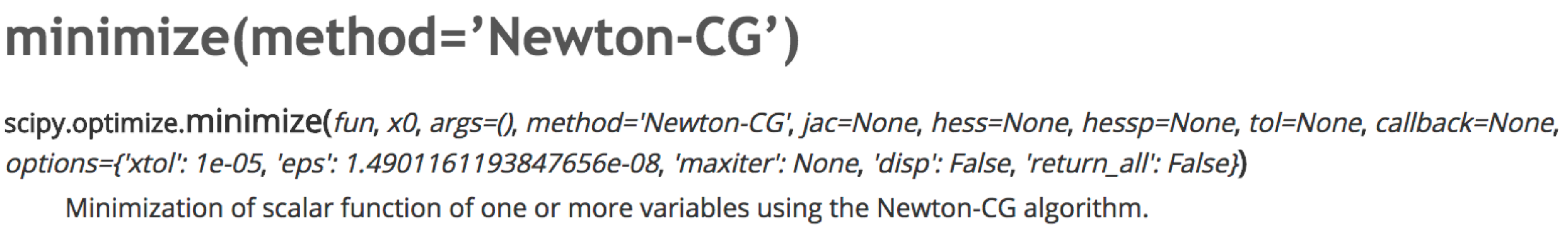
numpy.histogram(*a*, *bins=10*, *range=None*, *normed=False*, *weights=None*, *density=None*)

Compute the histogram of a set of data.



scipy.stats.binned_statistic(*x*, *values*, *statistic='mean'*, *bins=10*, *range=None*)

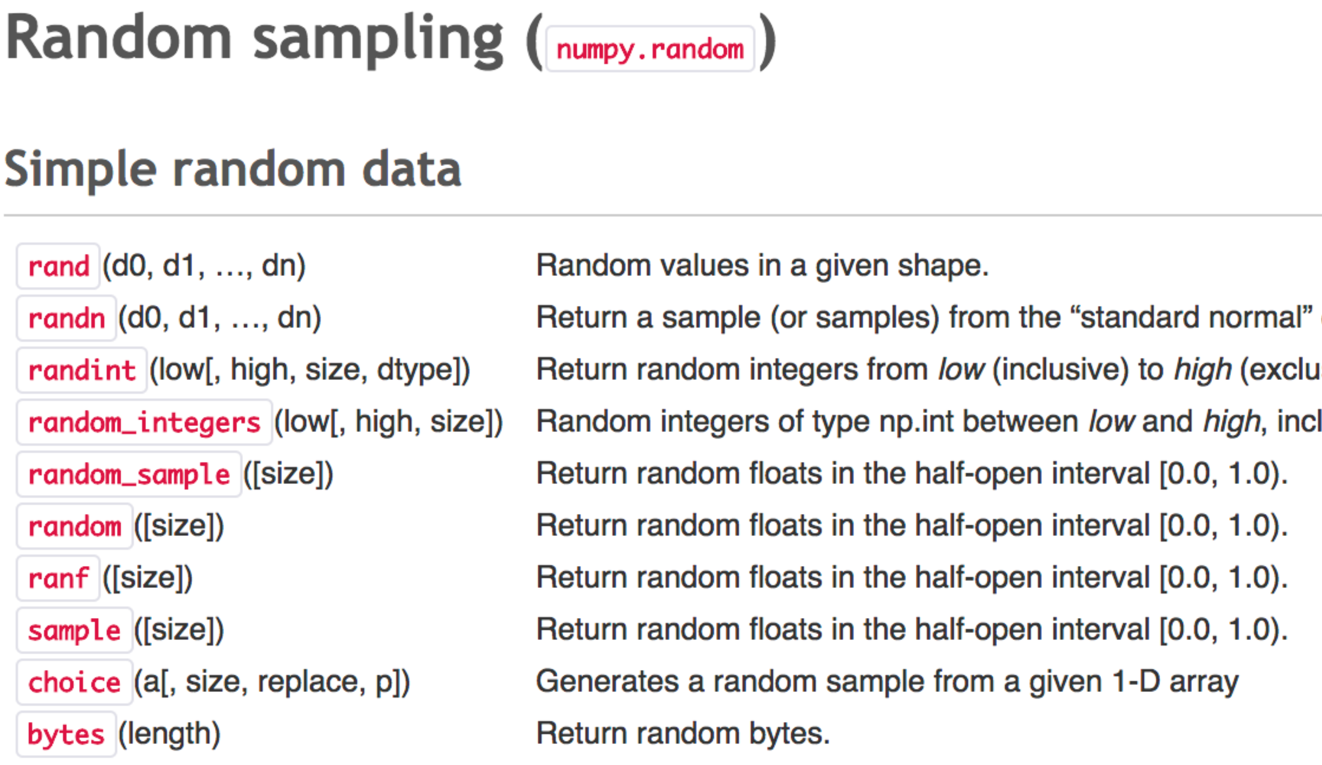
Compute a binned statistic for a set of data.



minimize(method=’Newton-CG’)

scipy.optimize.minimize(*fun*, *x0*, *args=()*, *method='Newton-CG'*, *jac=None*, *hess=None*, *hessp=None*, *tol=None*, *callback=None*, *options={'xtol': 1e-05, 'eps': 1.4901161193847656e-08, 'maxiter': None, 'disp': False, 'return_all': False}*)

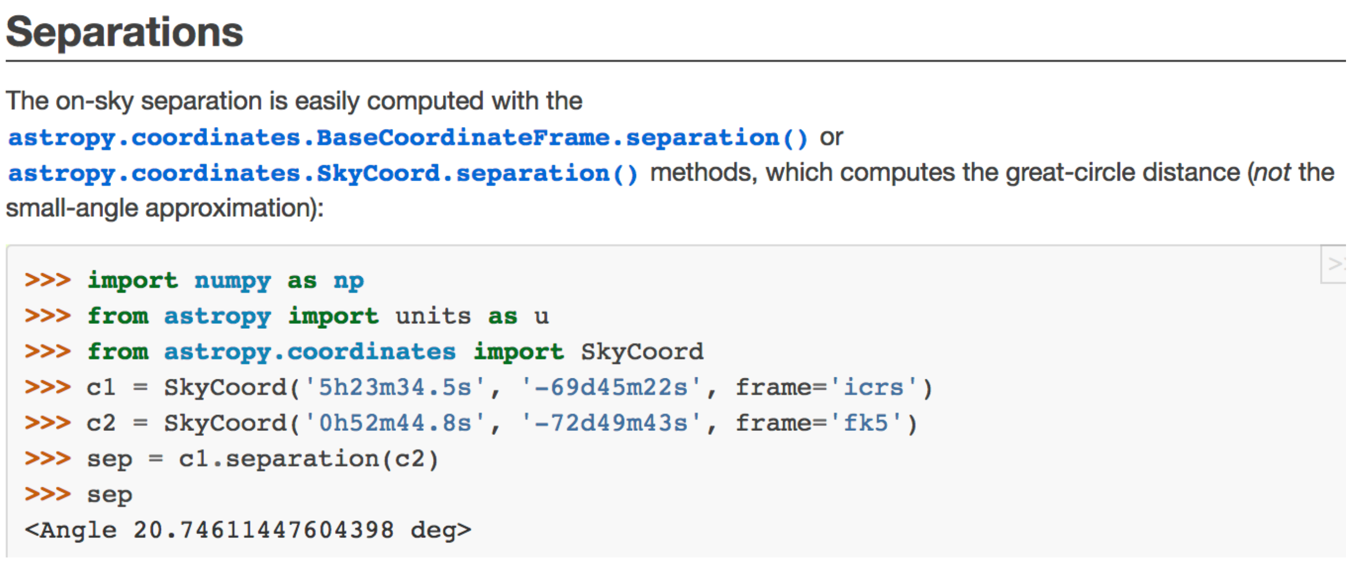
Minimization of scalar function of one or more variables using the Newton-CG algorithm.



Random sampling (numpy.random)

Simple random data

rand (d0, d1, …, dn)	Random values in a given shape.
randn (d0, d1, …, dn)	Return a sample (or samples) from the “standard normal” distrit
randint (low[, high, size, dtype])	Return random integers from *low* (inclusive) to *high* (exclusive).
random_integers (low[, high, size])	Random integers of type np.int between *low* and *high*, inclusive
random_sample ([size])	Return random floats in the half-open interval [0.0, 1.0).
random ([size])	Return random floats in the half-open interval [0.0, 1.0).
ranf ([size])	Return random floats in the half-open interval [0.0, 1.0).
sample ([size])	Return random floats in the half-open interval [0.0, 1.0).
choice (a[, size, replace, p])	Generates a random sample from a given 1-D array
bytes (length)	Return random bytes.

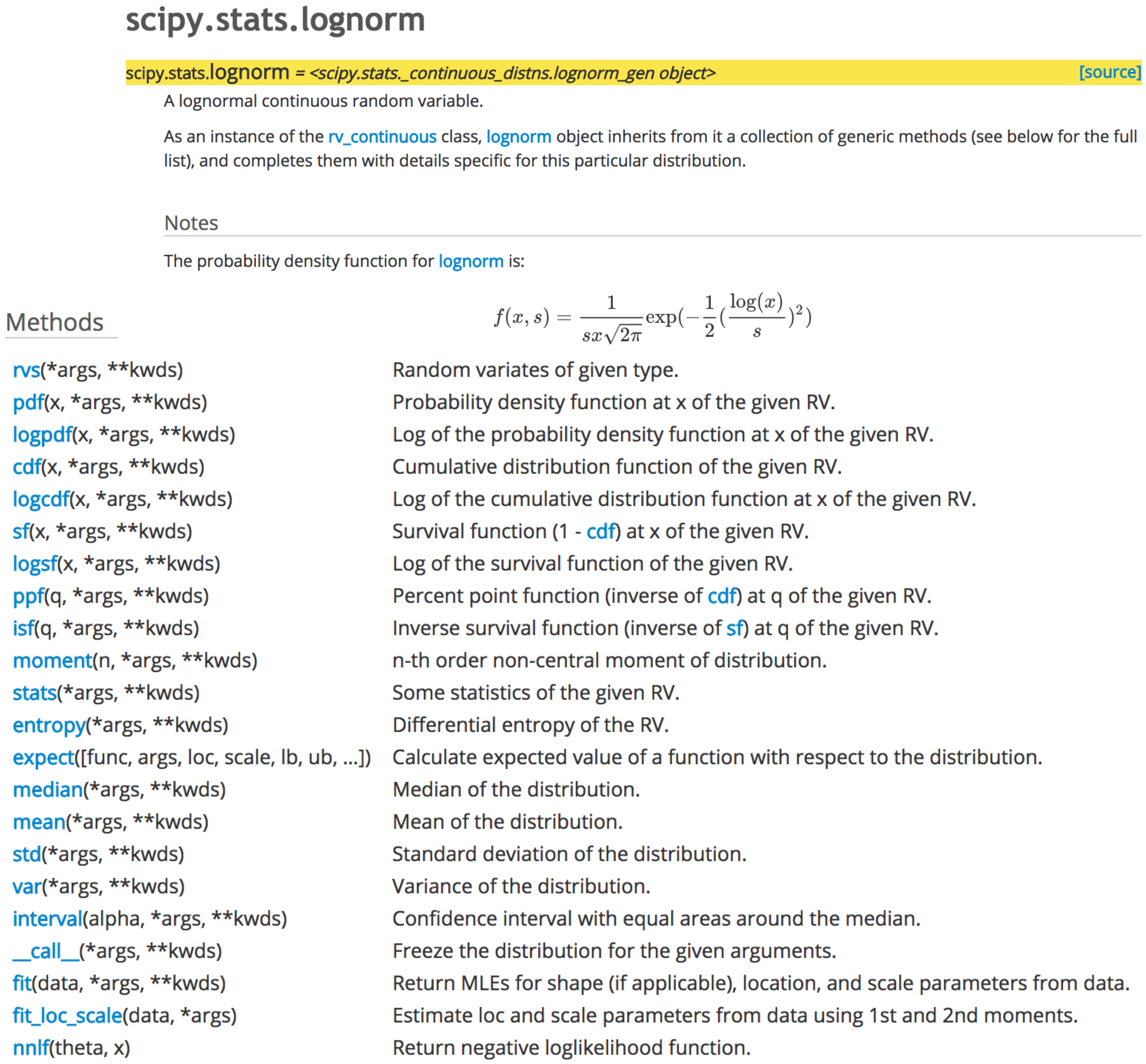


Separations

The on-sky separation is easily computed with the `astropy.coordinates.BaseCoordinateFrame.separation()` or `astropy.coordinates.SkyCoord.separation()` methods, which computes the great-circle distance (*not* the small-angle approximation):

```
>>> import numpy as np
>>> from astropy import units as u
>>> from astropy.coordinates import SkyCoord
>>> c1 = SkyCoord('5h23m34.5s', '-69d45m22s', frame='icrs')
>>> c2 = SkyCoord('0h52m44.8s', '-72d49m43s', frame='fk5')
>>> sep = c1.separation(c2)
>>> sep
<Angle 20.74611447604398 deg>
```

Maths Done For You



scipy.stats.lognorm

scipy.stats.**lognorm** = *<scipy.stats._continuous_distns.lognorm_gen object>* [source]

A lognormal continuous random variable.

As an instance of the rv_continuous class, lognorm object inherits from it a collection of generic methods (see below for the full list), and completes them with details specific for this particular distribution.

Notes

The probability density function for lognorm is:

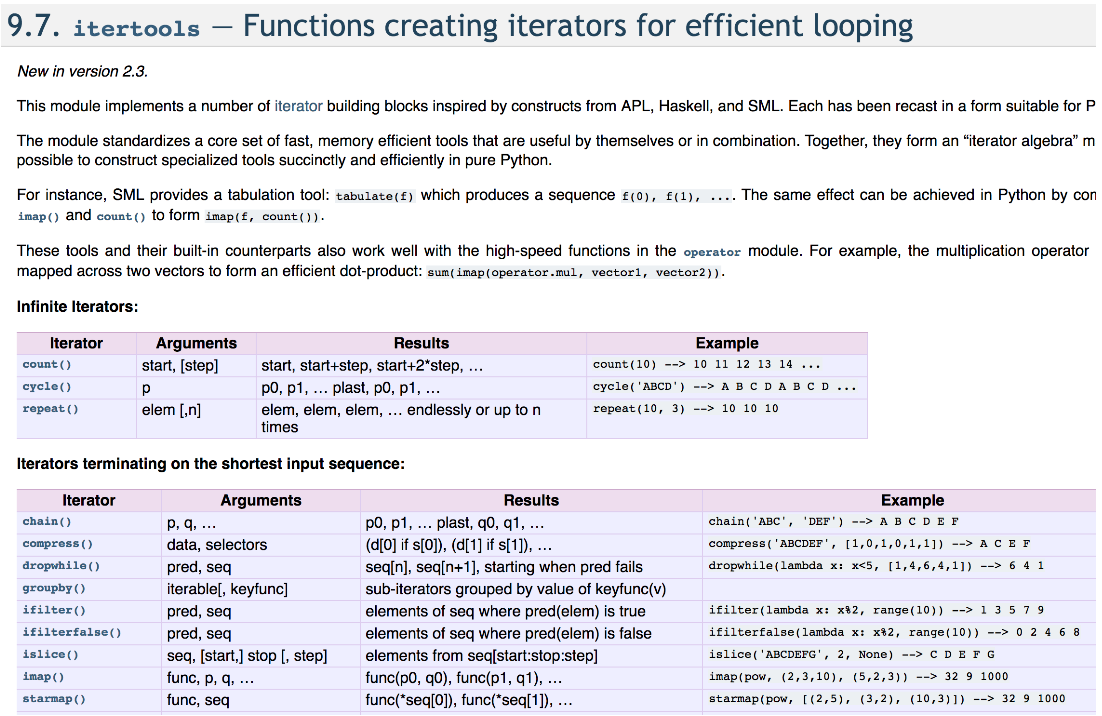
$$f(x, s) = \frac{1}{sx\sqrt{2\pi}} \exp(-\frac{1}{2}(\frac{\log(x)}{s})^2)$$

Methods

rvs(*args, **kwds)	Random variates of given type.
pdf(x, *args, **kwds)	Probability density function at x of the given RV.
logpdf(x, *args, **kwds)	Log of the probability density function at x of the given RV.
cdf(x, *args, **kwds)	Cumulative distribution function of the given RV.
logcdf(x, *args, **kwds)	Log of the cumulative distribution function at x of the given RV.
sf(x, *args, **kwds)	Survival function (1 - cdf) at x of the given RV.
logsf(x, *args, **kwds)	Log of the survival function of the given RV.
ppf(q, *args, **kwds)	Percent point function (inverse of cdf) at q of the given RV.
isf(q, *args, **kwds)	Inverse survival function (inverse of sf) at q of the given RV.
moment(n, *args, **kwds)	n-th order non-central moment of distribution.
stats(*args, **kwds)	Some statistics of the given RV.
entropy(*args, **kwds)	Differential entropy of the RV.
expect([func, args, loc, scale, lb, ub, ...])	Calculate expected value of a function with respect to the distribution.
median(*args, **kwds)	Median of the distribution.
mean(*args, **kwds)	Mean of the distribution.
std(*args, **kwds)	Standard deviation of the distribution.
var(*args, **kwds)	Variance of the distribution.
interval(alpha, *args, **kwds)	Confidence interval with equal areas around the median.
__call__(*args, **kwds)	Freeze the distribution for the given arguments.
fit(data, *args, **kwds)	Return MLEs for shape (if applicable), location, and scale parameters from data.
fit_loc_scale(data, *args)	Estimate loc and scale parameters from data using 1st and 2nd moments.
nnlf(theta, x)	Return negative loglikelihood function.

Never Having To Think About What’s Happening

Powerful for loops



9.7. itertools — Functions creating iterators for efficient looping

New in version 2.3.

This module implements a number of iterator building blocks inspired by constructs from APL, Haskell, and SML. Each has been recast in a form suitable for P

The module standardizes a core set of fast, memory efficient tools that are useful by themselves or in combination. Together, they form an “iterator algebra” m possible to construct specialized tools succinctly and efficiently in pure Python.

For instance, SML provides a tabulation tool: `tabulate(f)` which produces a sequence `f(0), f(1), ...`. The same effect can be achieved in Python by con `imap()` and `count()` to form `imap(f, count())`.

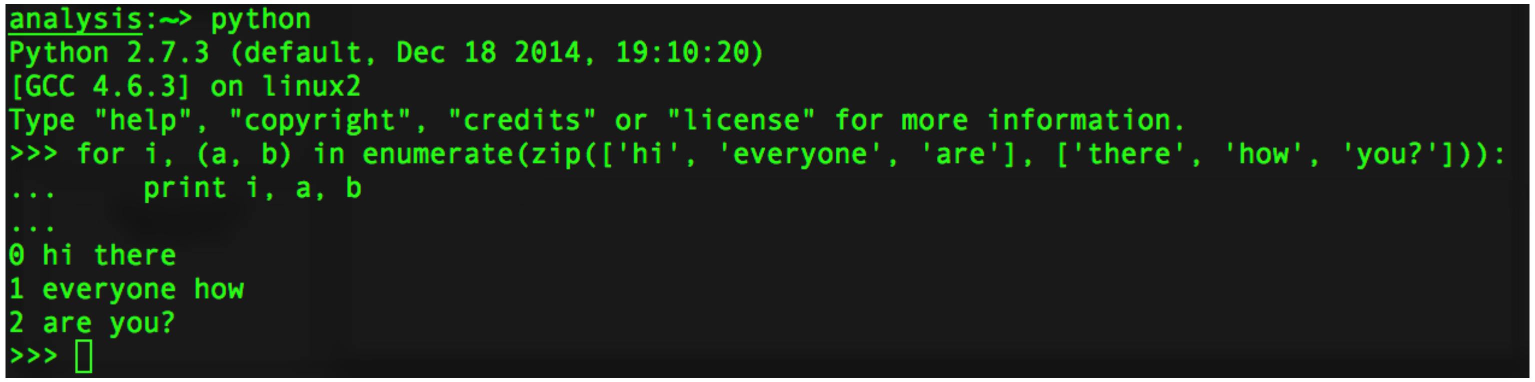
These tools and their built-in counterparts also work well with the high-speed functions in the `operator` module. For example, the multiplication operator mapped across two vectors to form an efficient dot-product: `sum(imap(operator.mul, vector1, vector2))`.

Infinite Iterators:

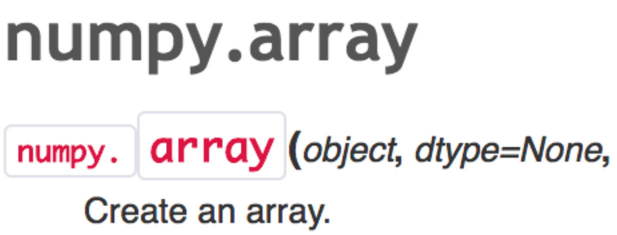
Iterator	Arguments	Results	Example
`count()`	start, [step]	start, start+step, start+2*step, …	`count(10) --> 10 11 12 13 14 ...`
`cycle()`	p	p0, p1, … plast, p0, p1, …	`cycle('ABCD') --> A B C D A B C D ...`
`repeat()`	elem [,n]	elem, elem, elem, … endlessly or up to n times	`repeat(10, 3) --> 10 10 10`

Iterators terminating on the shortest input sequence:

Iterator	Arguments	Results	Example
`chain()`	p, q, …	p0, p1, … plast, q0, q1, …	`chain('ABC', 'DEF') --> A B C D E F`
`compress()`	data, selectors	(d[0] if s[0]), (d[1] if s[1]), …	`compress('ABCDEF', [1,0,1,0,1,1]) --> A C E F`
`dropwhile()`	pred, seq	seq[n], seq[n+1], starting when pred fails	`dropwhile(lambda x: x<5, [1,4,6,4,1]) --> 6 4 1`
`groupby()`	iterable[, keyfunc]	sub-iterators grouped by value of keyfunc(v)	
`ifilter()`	pred, seq	elements of seq where pred(elem) is true	`ifilter(lambda x: x%2, range(10)) --> 1 3 5 7 9`
`ifilterfalse()`	pred, seq	elements of seq where pred(elem) is false	`ifilterfalse(lambda x: x%2, range(10)) --> 0 2 4 6 8`
`islice()`	seq, [start,] stop [, step]	elements from seq[start:stop:step]	`islice('ABCDEFG', 2, None) --> C D E F G`
`imap()`	func, p, q, …	func(p0, q0), func(p1, q1), …	`imap(pow, (2,3,10), (5,2,3)) --> 32 9 1000`
`starmap()`	func, seq	func(*seq[0]), func(*seq[1]), …	`starmap(pow, [(2,5), (3,2), (10,3)]) --> 32 9 1000`



```
analysis:~> python
Python 2.7.3 (default, Dec 18 2014, 19:10:20)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> for i, (a, b) in enumerate(zip(['hi', 'everyone', 'are'], ['there', 'how', 'you?'])):
...     print i, a, b
...
0 hi there
1 everyone how
2 are you?
>>> 
```

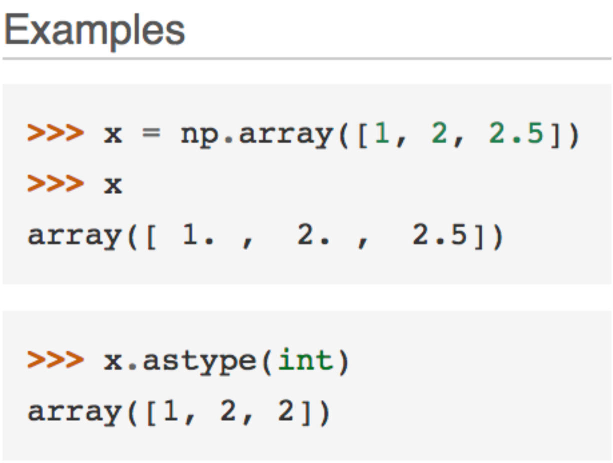


numpy.array

`numpy.array`(*object*, *dtype=None*,

Create an array.

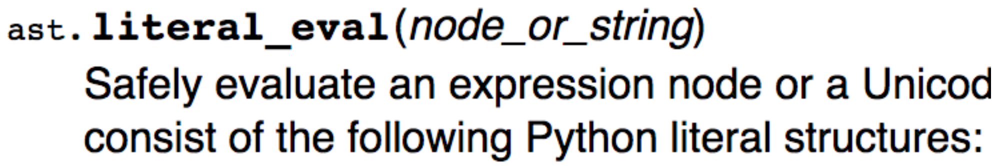
Data type control



Examples

```
>>> x = np.array([1, 2, 2.5])
>>> x
array([ 1. ,  2. ,  2.5])
```

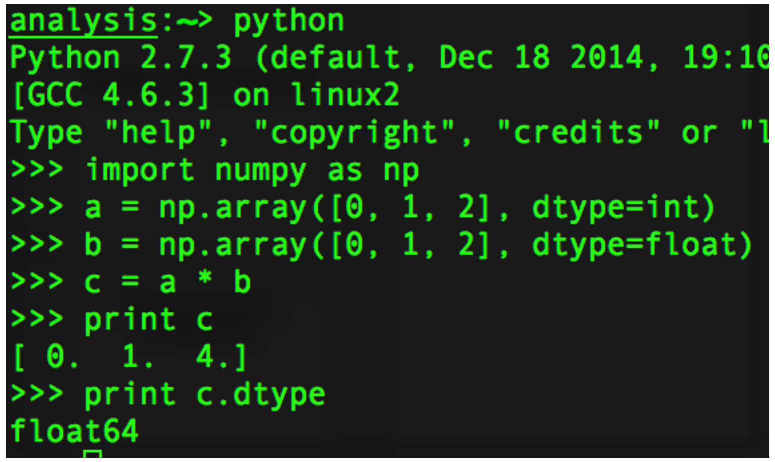
```
>>> x.astype(int)
array([1, 2, 2])
```



`ast.literal_eval`(*node_or_string*)

Safely evaluate an expression node or a Unicod consist of the following Python literal structures:

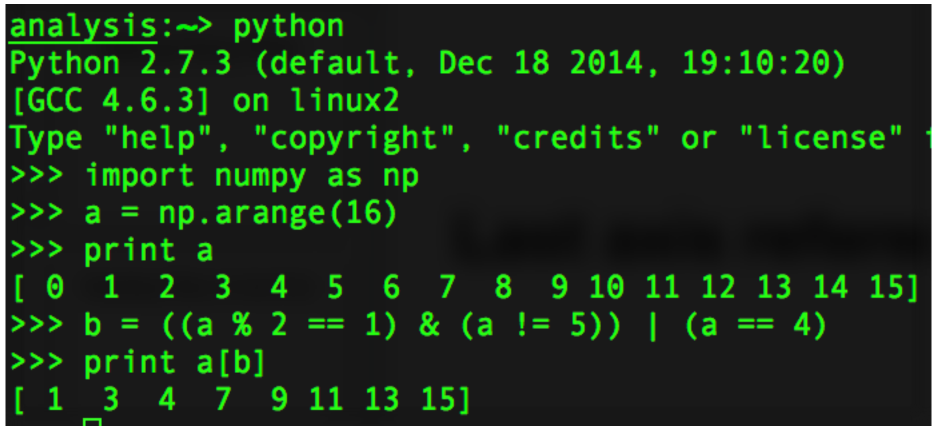
Beware array casting!



```
analysis:~> python
Python 2.7.3 (default, Dec 18 2014, 19:10
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "l
>>> import numpy as np
>>> a = np.array([0, 1, 2], dtype=int)
>>> b = np.array([0, 1, 2], dtype=float)
>>> c = a * b
>>> print c
[ 0.  1.  4.]
>>> print c.dtype
float64
```

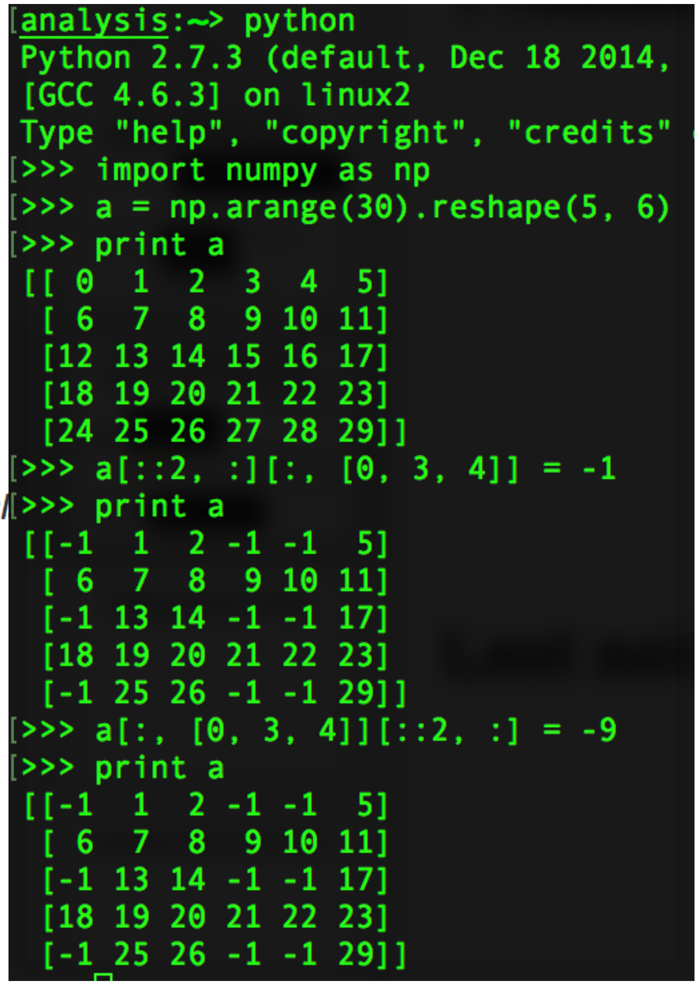
Never Having To Think About What’s Happening

Logical Slices

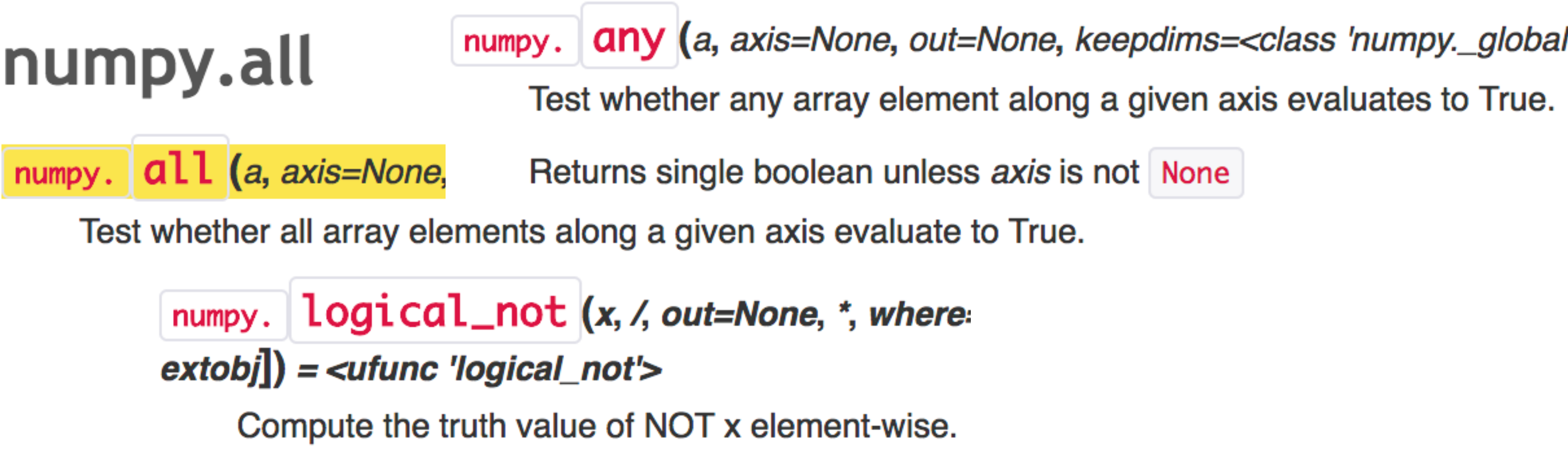


```
analysis:~> python
Python 2.7.3 (default, Dec 18 2014, 19:10:20)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" f
>>> import numpy as np
>>> a = np.arange(16)
>>> print a
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15]
>>> b = ((a % 2 == 1) & (a != 5)) | (a == 4)
>>> print a[b]
[ 1  3  4  7  9 11 13 15]
```

Beware view vs copy!



```
[analysis:~> python
 Python 2.7.3 (default, Dec 18 2014,
 [GCC 4.6.3] on linux2
 Type "help", "copyright", "credits"
[>>> import numpy as np
[>>> a = np.arange(30).reshape(5, 6)
[>>> print a
 [[ 0  1  2  3  4  5]
  [ 6  7  8  9 10 11]
  [12 13 14 15 16 17]
  [18 19 20 21 22 23]
  [24 25 26 27 28 29]]
[>>> a[::2, :][:, [0, 3, 4]] = -1
[>>> print a
 [[-1  1  2 -1 -1  5]
  [ 6  7  8  9 10 11]
  [-1 13 14 -1 -1 17]
  [18 19 20 21 22 23]
  [-1 25 26 -1 -1 29]]
[>>> a[:, [0, 3, 4]][::2, :] = -9
[>>> print a
 [[-1  1  2 -1 -1  5]
  [ 6  7  8  9 10 11]
  [-1 13 14 -1 -1 17]
  [18 19 20 21 22 23]
  [-1 25 26 -1 -1 29]]
```



numpy.all

`numpy.all`(*a, axis=None,*

Test whether all array elements along a given axis evaluate to True.

`numpy.any`(*a, axis=None, out=None, keepdims=<class 'numpy._global*

Test whether any array element along a given axis evaluates to True.

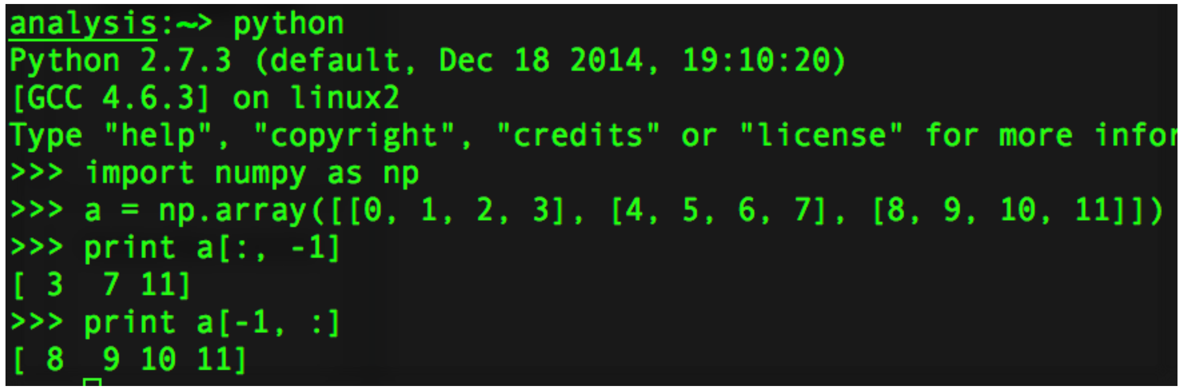
Returns single boolean unless *axis* is not `None`

`numpy.logical_not`(***x, /, out=None, *, where***

***extobj*]) = <ufunc 'logical_not'>**

Compute the truth value of NOT x element-wise.

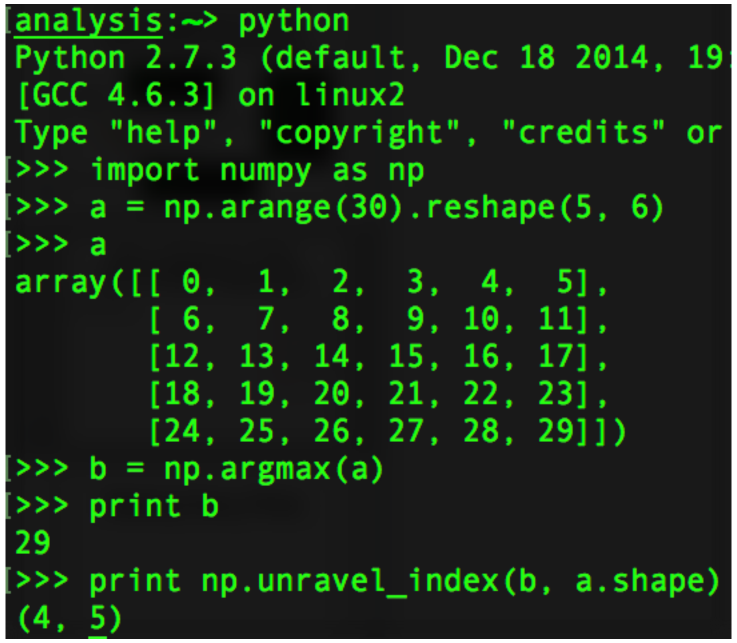
Last axis referencing



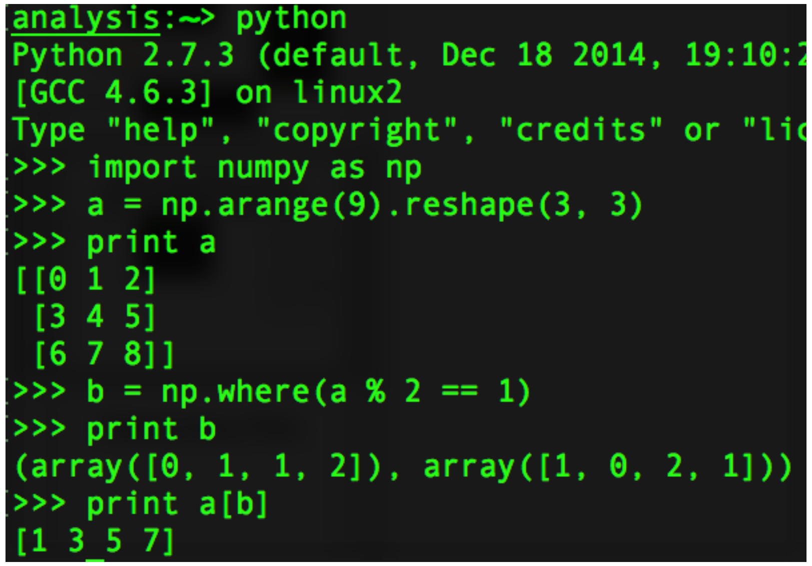
```
analysis:~> python
Python 2.7.3 (default, Dec 18 2014, 19:10:20)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more infor
>>> import numpy as np
>>> a = np.array([[0, 1, 2, 3], [4, 5, 6, 7], [8, 9, 10, 11]])
>>> print a[:, -1]
[ 3  7 11]
>>> print a[-1, :]
[ 8  9 10 11]
```

Never Having To Think About What's Happening

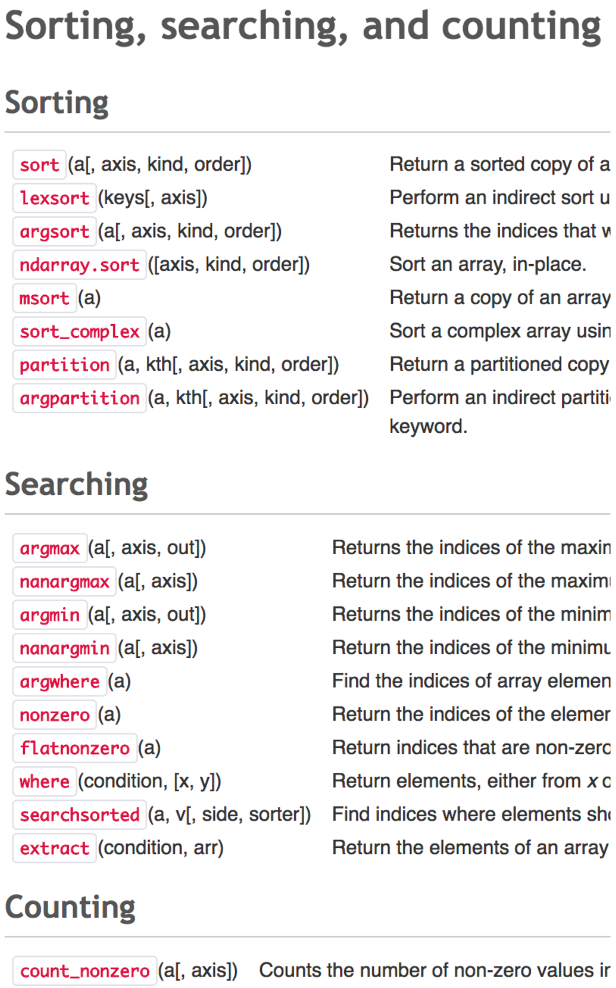
Index Manipulation



```
analysis:~> python
Python 2.7.3 (default, Dec 18 2014, 19
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or
>>> import numpy as np
>>> a = np.arange(30).reshape(5, 6)
>>> a
array([[ 0,  1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11],
       [12, 13, 14, 15, 16, 17],
       [18, 19, 20, 21, 22, 23],
       [24, 25, 26, 27, 28, 29]])
>>> b = np.argmax(a)
>>> print b
29
>>> print np.unravel_index(b, a.shape)
(4, 5)
```



```
analysis:~> python
Python 2.7.3 (default, Dec 18 2014, 19:10:2
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "lic
>>> import numpy as np
>>> a = np.arange(9).reshape(3, 3)
>>> print a
[[0 1 2]
 [3 4 5]
 [6 7 8]]
>>> b = np.where(a % 2 == 1)
>>> print b
(array([0, 1, 1, 2]), array([1, 0, 2, 1]))
>>> print a[b]
[1 3 5 7]
```



Sorting, searching, and counting

Sorting

`sort` (a[, axis, kind, order])	Return a sorted copy of a
`lexsort` (keys[, axis])	Perform an indirect sort u
`argsort` (a[, axis, kind, order])	Returns the indices that w
`ndarray.sort` ([axis, kind, order])	Sort an array, in-place.
`msort` (a)	Return a copy of an array
`sort_complex` (a)	Sort a complex array usin
`partition` (a, kth[, axis, kind, order])	Return a partitioned copy
`argpartition` (a, kth[, axis, kind, order])	Perform an indirect partiti keyword.

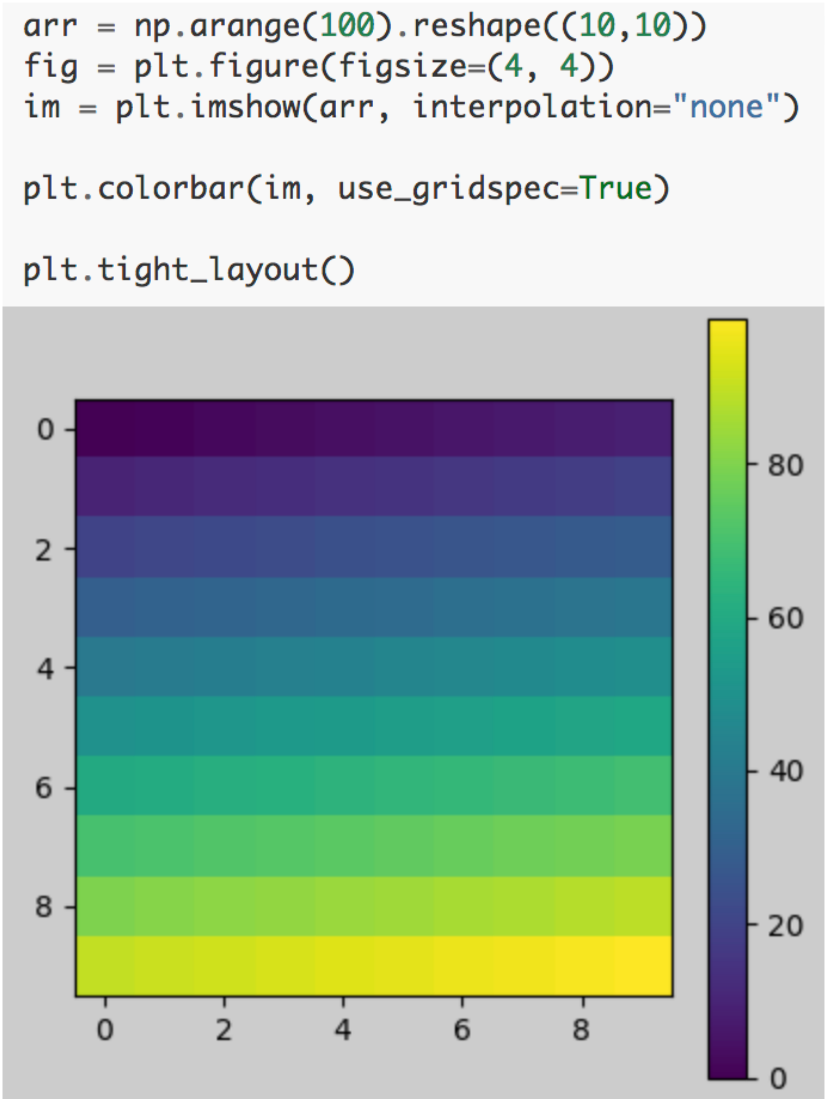
Searching

`argmax` (a[, axis, out])	Returns the indices of the maxin
`nanargmax` (a[, axis])	Return the indices of the maximu
`argmin` (a[, axis, out])	Returns the indices of the minim
`nanargmin` (a[, axis])	Return the indices of the minimu
`argwhere` (a)	Find the indices of array elemen
`nonzero` (a)	Return the indices of the elemer
`flatnonzero` (a)	Return indices that are non-zerc
`where` (condition, [x, y])	Return elements, either from *x* c
`searchsorted` (a, v[, side, sorter])	Find indices where elements sho
`extract` (condition, arr)	Return the elements of an array

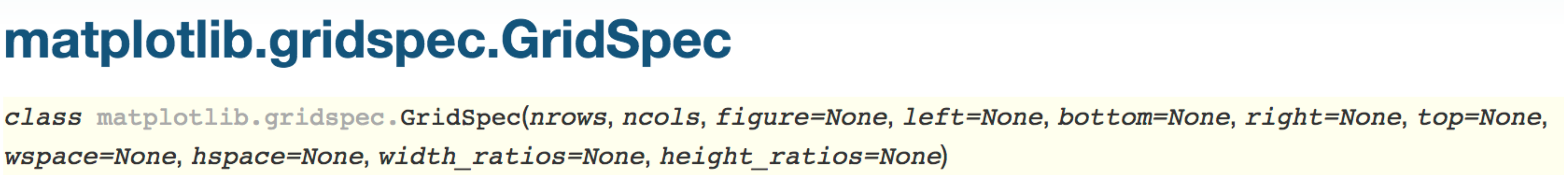
Counting

`count_nonzero` (a[, axis])	Counts the number of non-zero values ir

Making Fancy Plots

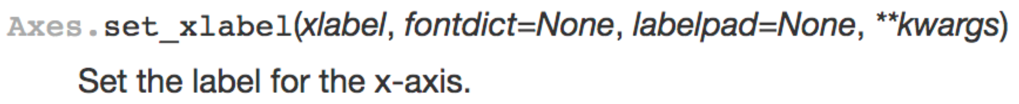


```
arr = np.arange(100).reshape((10,10))
fig = plt.figure(figsize=(4, 4))
im = plt.imshow(arr, interpolation="none")

plt.colorbar(im, use_gridspec=True)

plt.tight_layout()
```



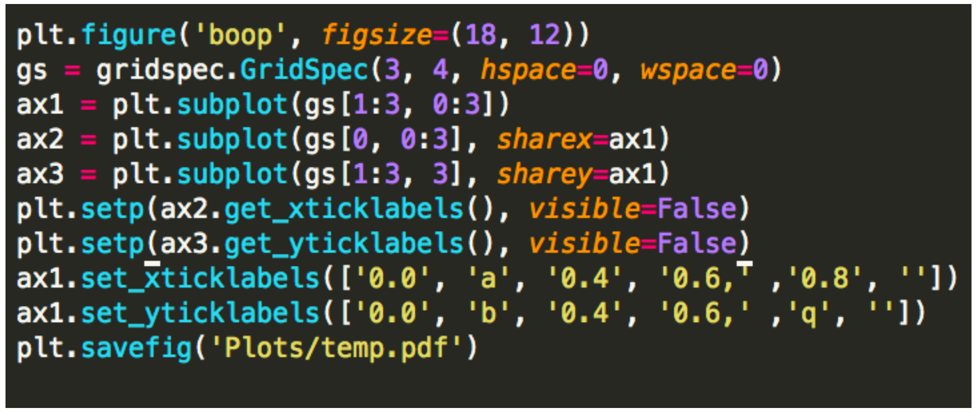
matplotlib.gridspec.GridSpec

class matplotlib.gridspec.GridSpec(*nrows*, *ncols*, *figure=None*, *left=None*, *bottom=None*, *right=None*, *top=None*, *wspace=None*, *hspace=None*, *width_ratios=None*, *height_ratios=None*)

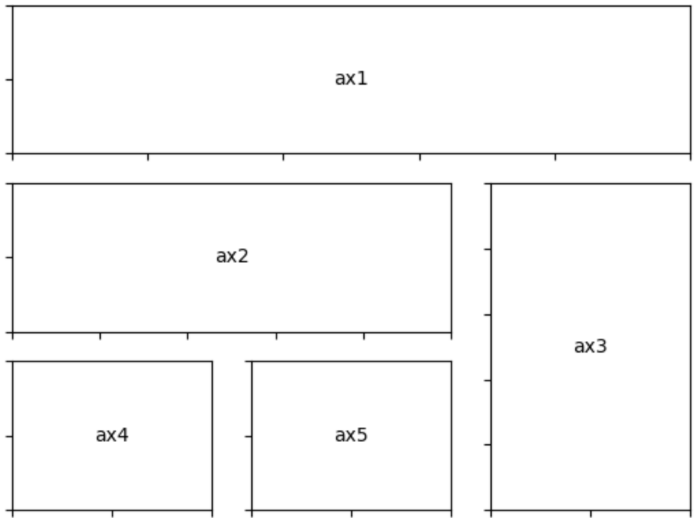


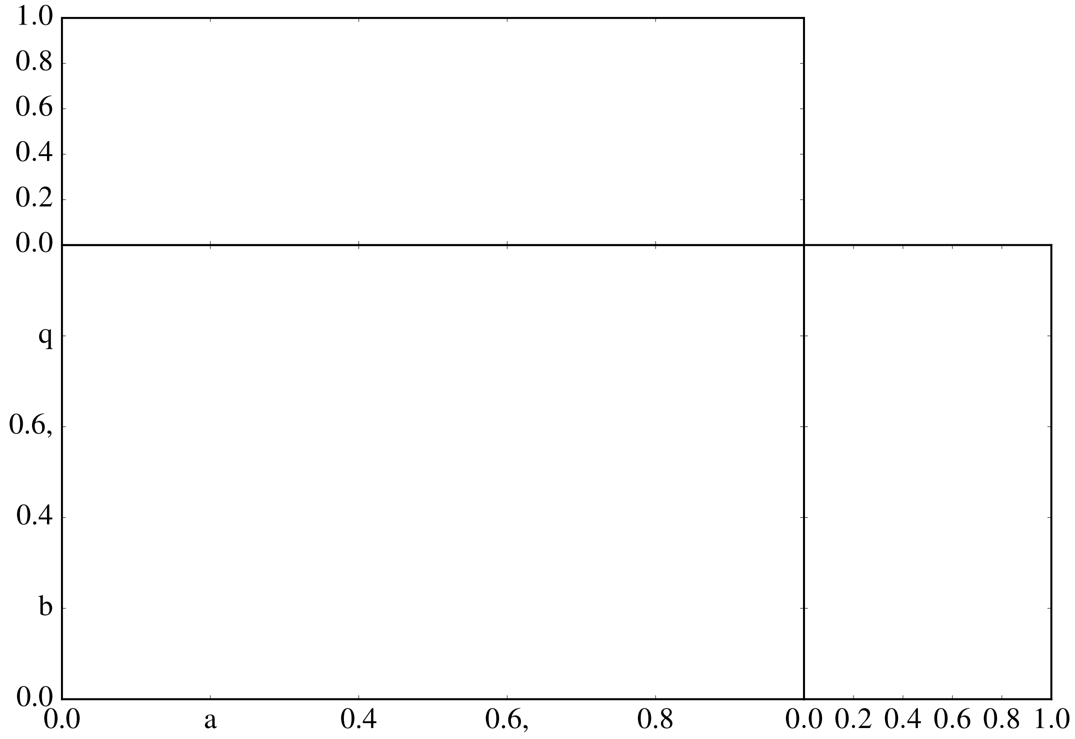
Axes.set_xlabel(*xlabel*, *fontdict=None*, *labelpad=None*, ***kwargs*)

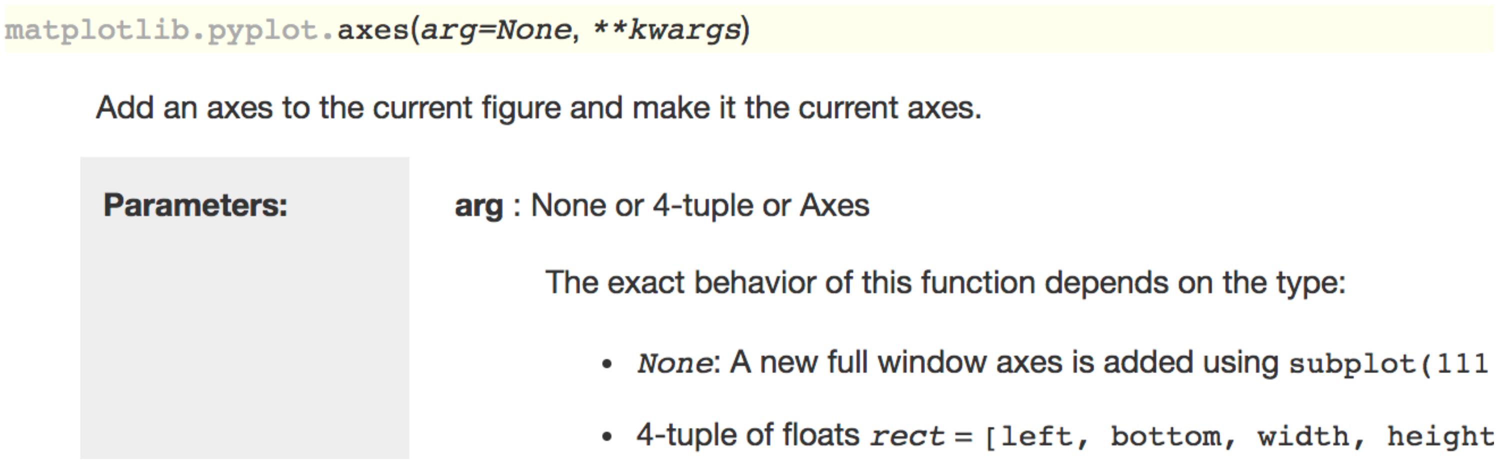
Set the label for the x-axis.



```
plt.figure('boop', figsize=(18, 12))
gs = gridspec.GridSpec(3, 4, hspace=0, wspace=0)
ax1 = plt.subplot(gs[1:3, 0:3])
ax2 = plt.subplot(gs[0, 0:3], sharex=ax1)
ax3 = plt.subplot(gs[1:3, 3], sharey=ax1)
plt.setp(ax2.get_xticklabels(), visible=False)
plt.setp(ax3.get_yticklabels(), visible=False)
ax1.set_xticklabels(['0.0', 'a', '0.4', '0.6,' ,'0.8', ''])
ax1.set_yticklabels(['0.0', 'b', '0.4', '0.6,' ,'q', ''])
plt.savefig('Plots/temp.pdf')
```







matplotlib.pyplot.axes(*arg=None*, ***kwargs*)

Add an axes to the current figure and make it the current axes.

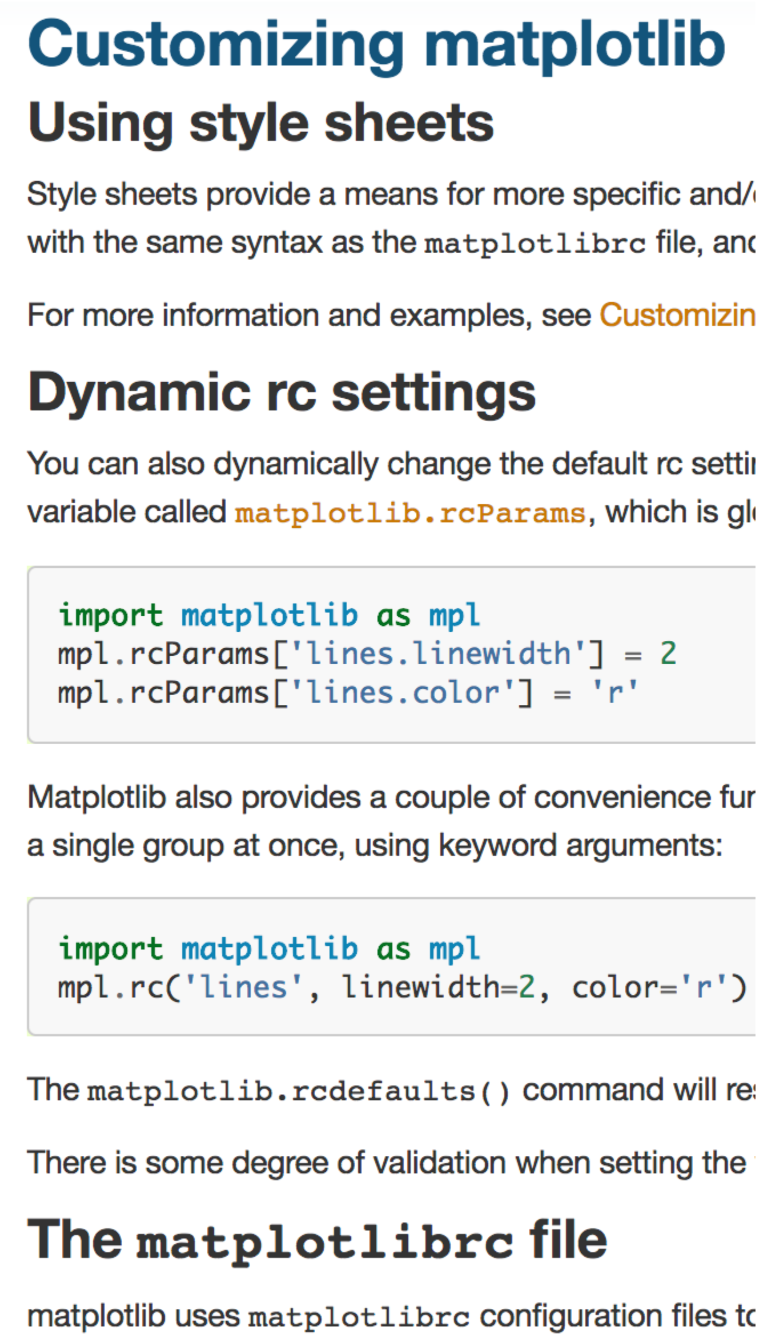
Parameters:

arg : None or 4-tuple or Axes

The exact behavior of this function depends on the type:

- *None*: A new full window axes is added using `subplot(111`
- 4-tuple of floats *rect* = `[left, bottom, width, height`

Making Fancy Plots



Customizing matplotlib

Using style sheets

Style sheets provide a means for more specific and/
with the same syntax as the `matplotlibrc` file, and

For more information and examples, see Customizin

Dynamic rc settings

You can also dynamically change the default rc setti
variable called `matplotlib.rcParams`, which is gl

```
import matplotlib as mpl
mpl.rcParams['lines.linewidth'] = 2
mpl.rcParams['lines.color'] = 'r'
```

Matplotlib also provides a couple of convenience fur
a single group at once, using keyword arguments:

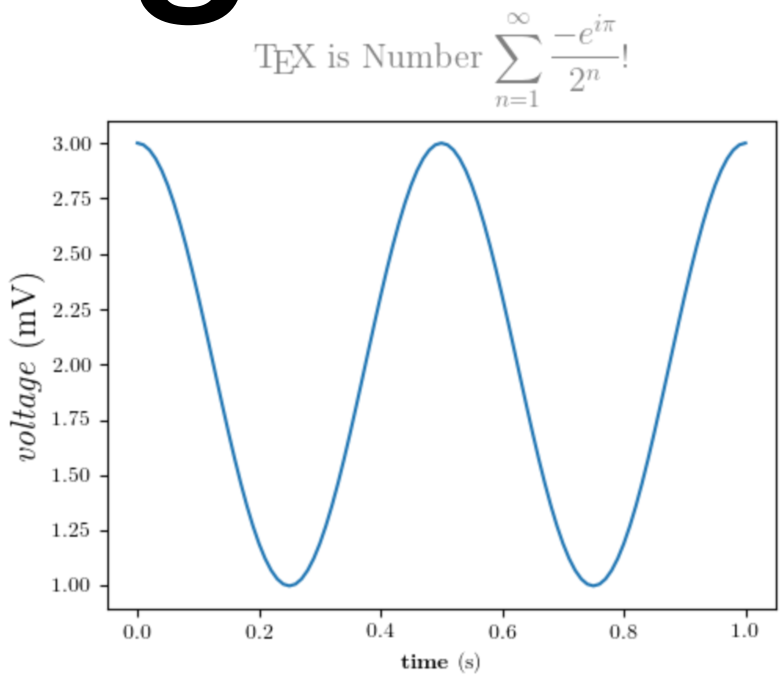
```
import matplotlib as mpl
mpl.rc('lines', linewidth=2, color='r')
```

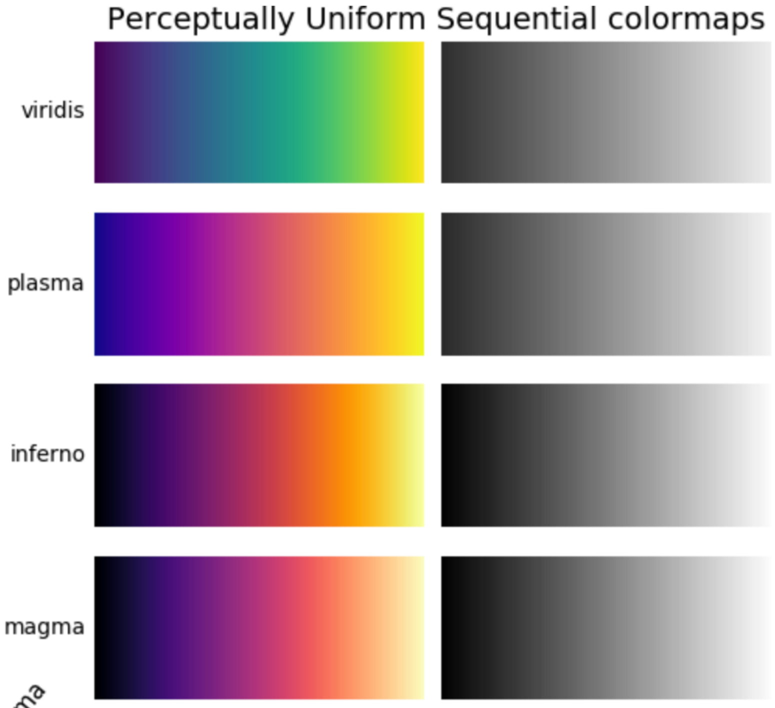
The `matplotlib.rcdefaults()` command will re

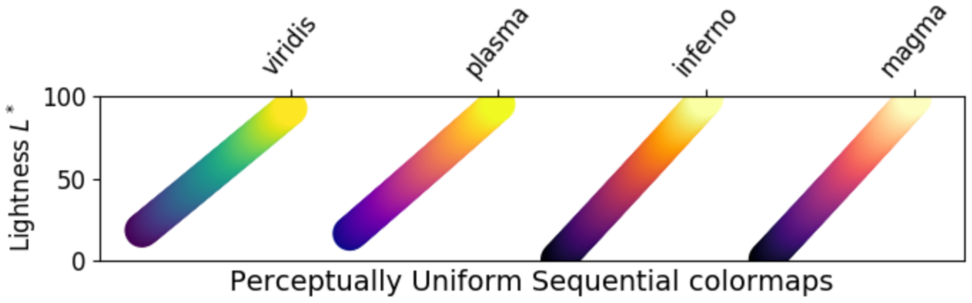
There is some degree of validation when setting the

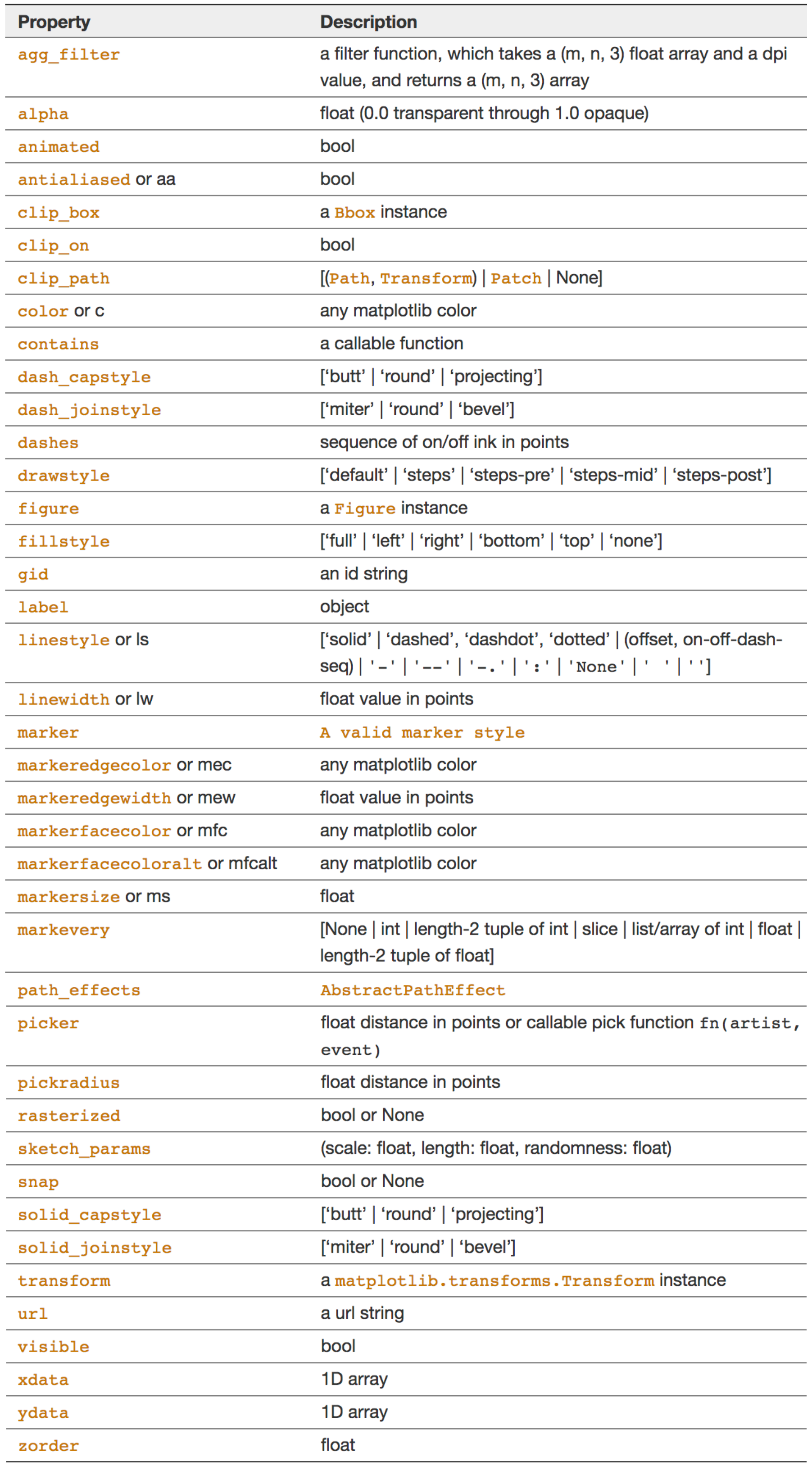
The `matplotlibrc` file

matplotlib uses `matplotlibrc` configuration files to



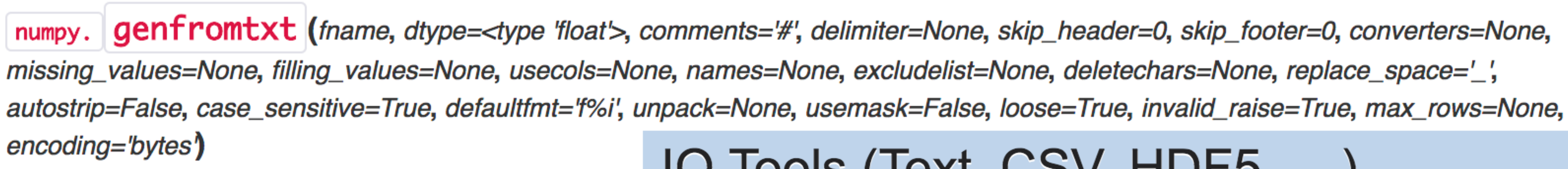




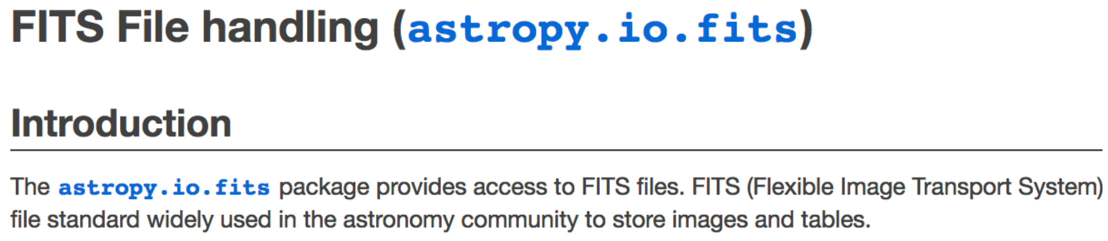


Property	Description
`agg_filter`	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array
`alpha`	float (0.0 transparent through 1.0 opaque)
`animated`	bool
`antialiased` or aa	bool
`clip_box`	a `Bbox` instance
`clip_on`	bool
`clip_path`	[(`Path`, `Transform`) | `Patch` | None]
`color` or c	any matplotlib color
`contains`	a callable function
`dash_capstyle`	['butt' | 'round' | 'projecting']
`dash_joinstyle`	['miter' | 'round' | 'bevel']
`dashes`	sequence of on/off ink in points
`drawstyle`	['default' | 'steps' | 'steps-pre' | 'steps-mid' | 'steps-post']
`figure`	a `Figure` instance
`fillstyle`	['full' | 'left' | 'right' | 'bottom' | 'top' | 'none']
`gid`	an id string
`label`	object
`linestyle` or ls	['solid' | 'dashed', 'dashdot', 'dotted' | (offset, on-off-dash-seq) | `'-'` | `'--'` | `'-.'` | `':'` | `'None'` | `' '` | `''`]
`linewidth` or lw	float value in points
`marker`	`A valid marker style`
`markeredgecolor` or mec	any matplotlib color
`markeredgewidth` or mew	float value in points
`markerfacecolor` or mfc	any matplotlib color
`markerfacecoloralt` or mfcalt	any matplotlib color
`markersize` or ms	float
`markevery`	[None | int | length-2 tuple of int | slice | list/array of int | float | length-2 tuple of float]
`path_effects`	`AbstractPathEffect`
`picker`	float distance in points or callable pick function `fn(artist, event)`
`pickradius`	float distance in points
`rasterized`	bool or None
`sketch_params`	(scale: float, length: float, randomness: float)
`snap`	bool or None
`solid_capstyle`	['butt' | 'round' | 'projecting']
`solid_joinstyle`	['miter' | 'round' | 'bevel']
`transform`	a `matplotlib.transforms.Transform` instance
`url`	a url string
`visible`	bool
`xdata`	1D array
`ydata`	1D array
`zorder`	float

Reading Files In And Out



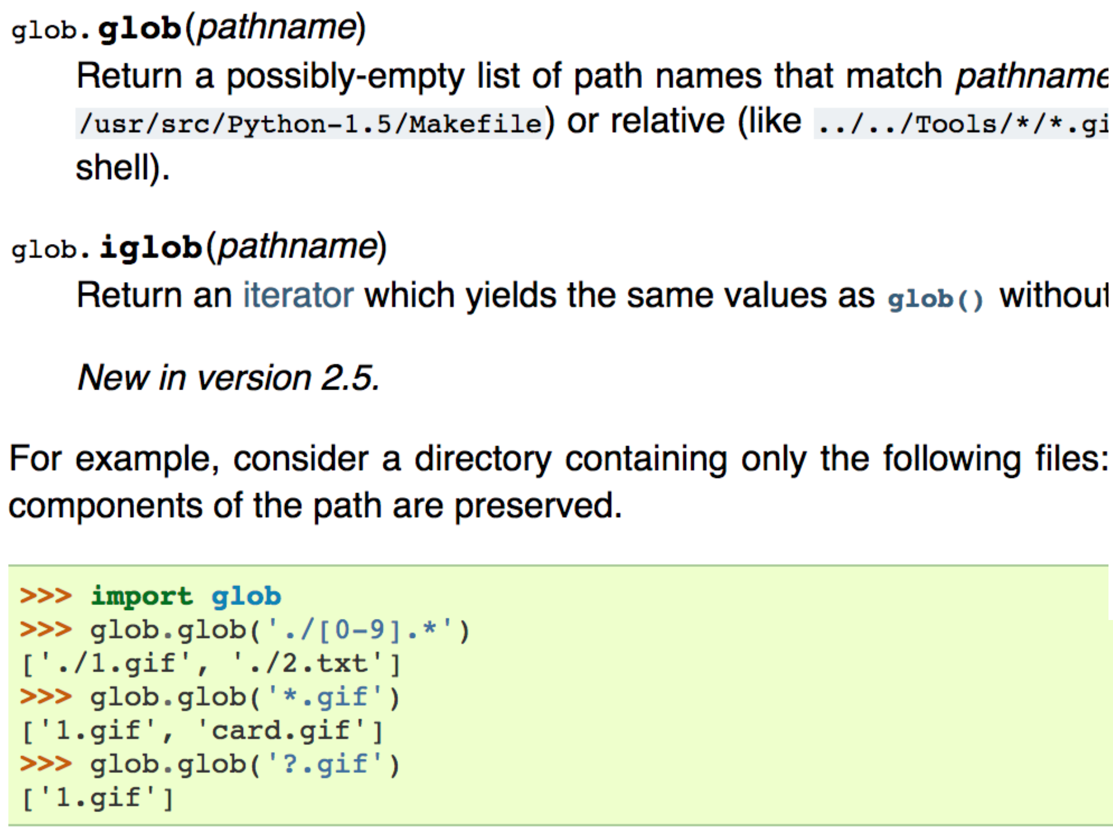
```
numpy.genfromtxt(fname, dtype=<type 'float'>, comments='#', delimiter=None, skip_header=0, skip_footer=0, converters=None,
missing_values=None, filling_values=None, usecols=None, names=None, excludelist=None, deletechars=None, replace_space='_',
autostrip=False, case_sensitive=True, defaultfmt='f%i', unpack=None, usemask=False, loose=True, invalid_raise=True, max_rows=None,
encoding='bytes')
```



FITS File handling (`astropy.io.fits`)

Introduction

The `astropy.io.fits` package provides access to FITS files. FITS (Flexible Image Transport System)
file standard widely used in the astronomy community to store images and tables.



`glob.glob`(*pathname*)

Return a possibly-empty list of path names that match *pathname*
`/usr/src/Python-1.5/Makefile`) or relative (like `../../Tools/*/*.gi`
shell).

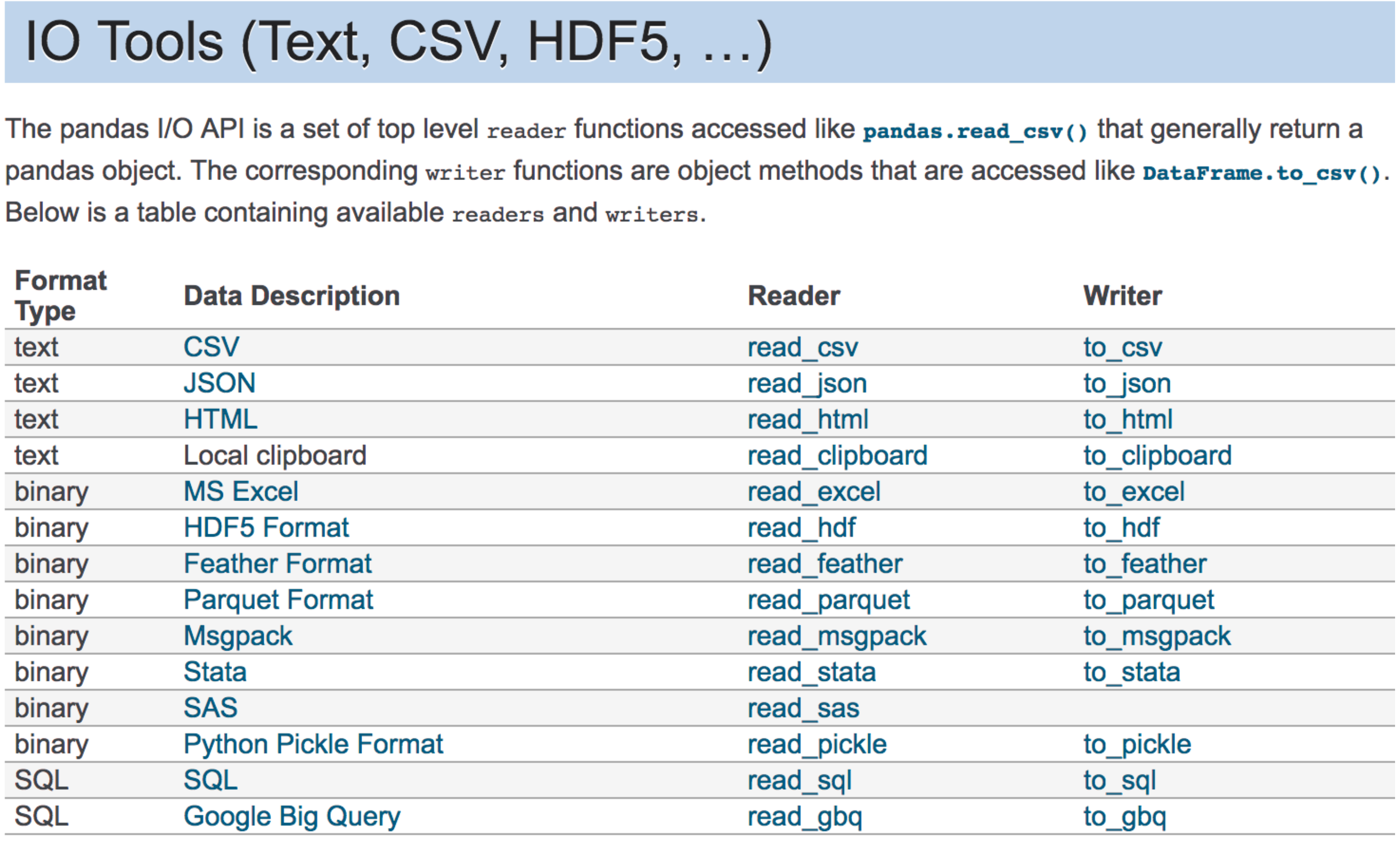
`glob.iglob`(*pathname*)

Return an iterator which yields the same values as `glob()` withoui

New in version 2.5.

For example, consider a directory containing only the following files:
components of the path are preserved.

```
>>> import glob
>>> glob.glob('./[0-9].*')
['./1.gif', './2.txt']
>>> glob.glob('*.gif')
['1.gif', 'card.gif']
>>> glob.glob('?.gif')
['1.gif']
```



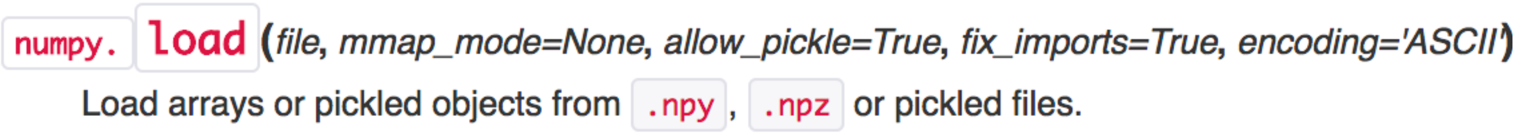
IO Tools (Text, CSV, HDF5, …)

The pandas I/O API is a set of top level `reader` functions accessed like `pandas.read_csv()` that generally return a pandas object. The corresponding `writer` functions are object methods that are accessed like `DataFrame.to_csv()`. Below is a table containing available `readers` and `writers`.

Format Type	Data Description	Reader	Writer
text	CSV	read_csv	to_csv
text	JSON	read_json	to_json
text	HTML	read_html	to_html
text	Local clipboard	read_clipboard	to_clipboard
binary	MS Excel	read_excel	to_excel
binary	HDF5 Format	read_hdf	to_hdf
binary	Feather Format	read_feather	to_feather
binary	Parquet Format	read_parquet	to_parquet
binary	Msgpack	read_msgpack	to_msgpack
binary	Stata	read_stata	to_stata
binary	SAS	read_sas	
binary	Python Pickle Format	read_pickle	to_pickle
SQL	SQL	read_sql	to_sql
SQL	Google Big Query	read_gbq	to_gbq

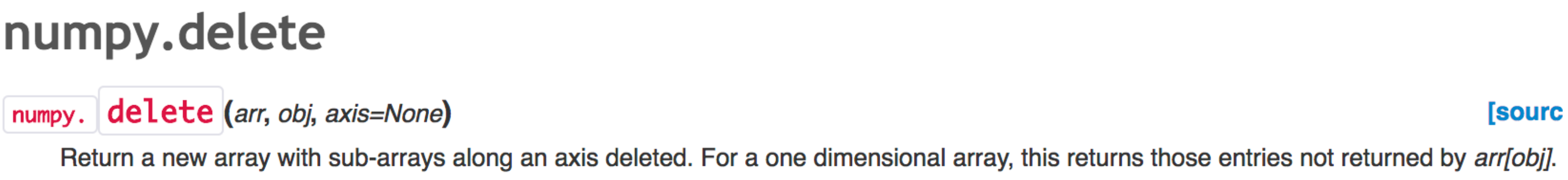
import os
os.remove(file)
os.system('terminal command')
os.path.exists(file)
os.makedirs(/path/to/directory)

Array Creation



`numpy.` `load` (*file, mmap_mode=None, allow_pickle=True, fix_imports=True, encoding='ASCII'*)

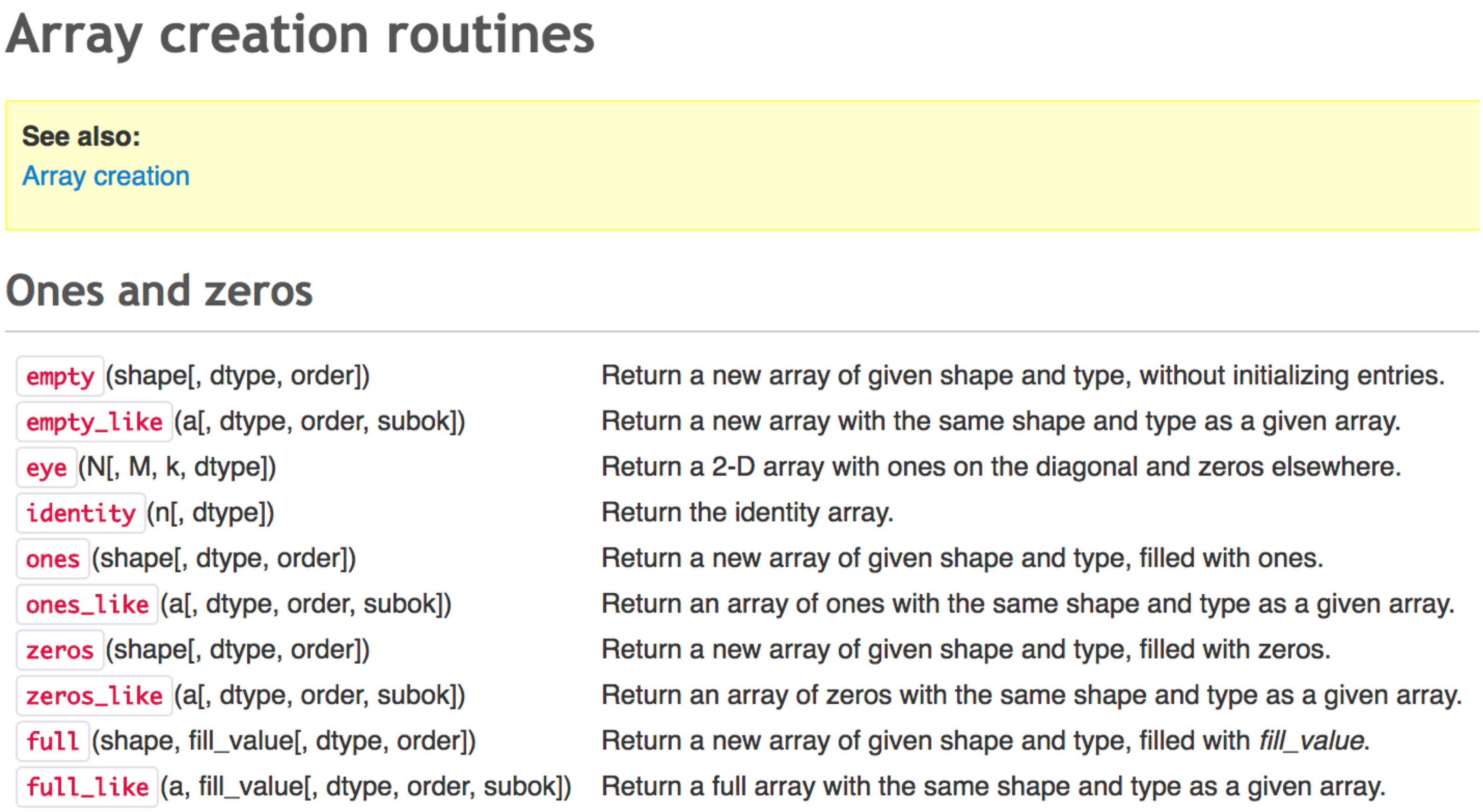
Load arrays or pickled objects from `.npy`, `.npz` or pickled files.



numpy.delete

`numpy.` `delete` (*arr, obj, axis=None*) [sourc

Return a new array with sub-arrays along an axis deleted. For a one dimensional array, this returns those entries not returned by *arr[obj]*.

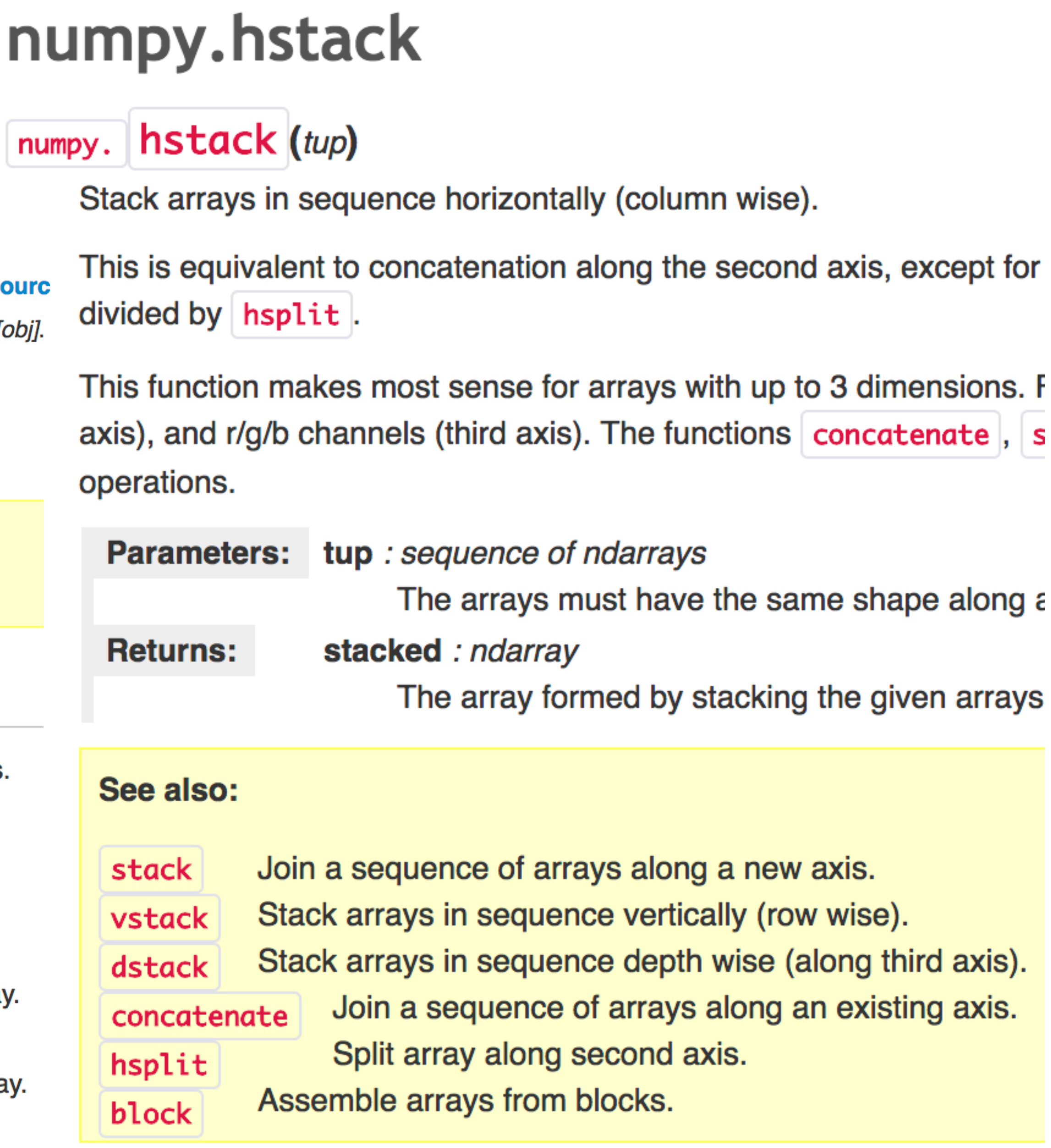


Array creation routines

See also:
Array creation

Ones and zeros

`empty` (shape[, dtype, order])	Return a new array of given shape and type, without initializing entries.
`empty_like` (a[, dtype, order, subok])	Return a new array with the same shape and type as a given array.
`eye` (N[, M, k, dtype])	Return a 2-D array with ones on the diagonal and zeros elsewhere.
`identity` (n[, dtype])	Return the identity array.
`ones` (shape[, dtype, order])	Return a new array of given shape and type, filled with ones.
`ones_like` (a[, dtype, order, subok])	Return an array of ones with the same shape and type as a given array.
`zeros` (shape[, dtype, order])	Return a new array of given shape and type, filled with zeros.
`zeros_like` (a[, dtype, order, subok])	Return an array of zeros with the same shape and type as a given array.
`full` (shape, fill_value[, dtype, order])	Return a new array of given shape and type, filled with *fill_value*.
`full_like` (a, fill_value[, dtype, order, subok])	Return a full array with the same shape and type as a given array.



numpy.hstack

`numpy.` `hstack` (*tup*)

Stack arrays in sequence horizontally (column wise).

This is equivalent to concatenation along the second axis, except for divided by `hsplit`.

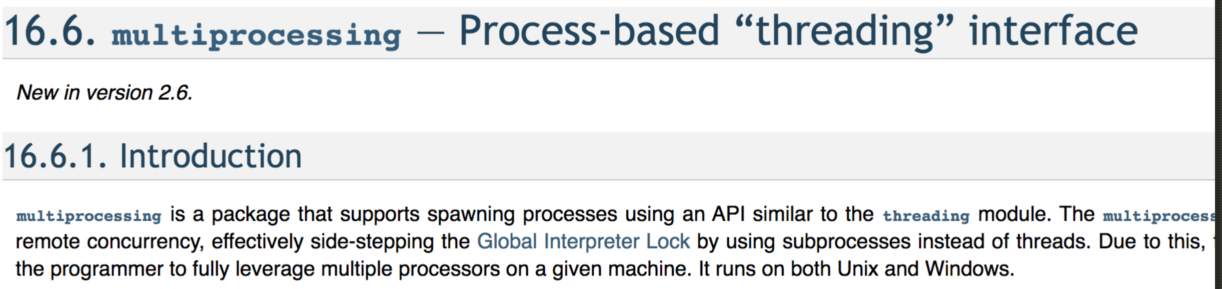
This function makes most sense for arrays with up to 3 dimensions. F axis), and r/g/b channels (third axis). The functions `concatenate`, `s` operations.

Parameters:	**tup** : *sequence of ndarrays* The arrays must have the same shape along a
Returns:	**stacked** : *ndarray* The array formed by stacking the given arrays

See also:

`stack`	Join a sequence of arrays along a new axis.
`vstack`	Stack arrays in sequence vertically (row wise).
`dstack`	Stack arrays in sequence depth wise (along third axis).
`concatenate`	Join a sequence of arrays along an existing axis.
`hsplit`	Split array along second axis.
`block`	Assemble arrays from blocks.

Maximising CPU Usage

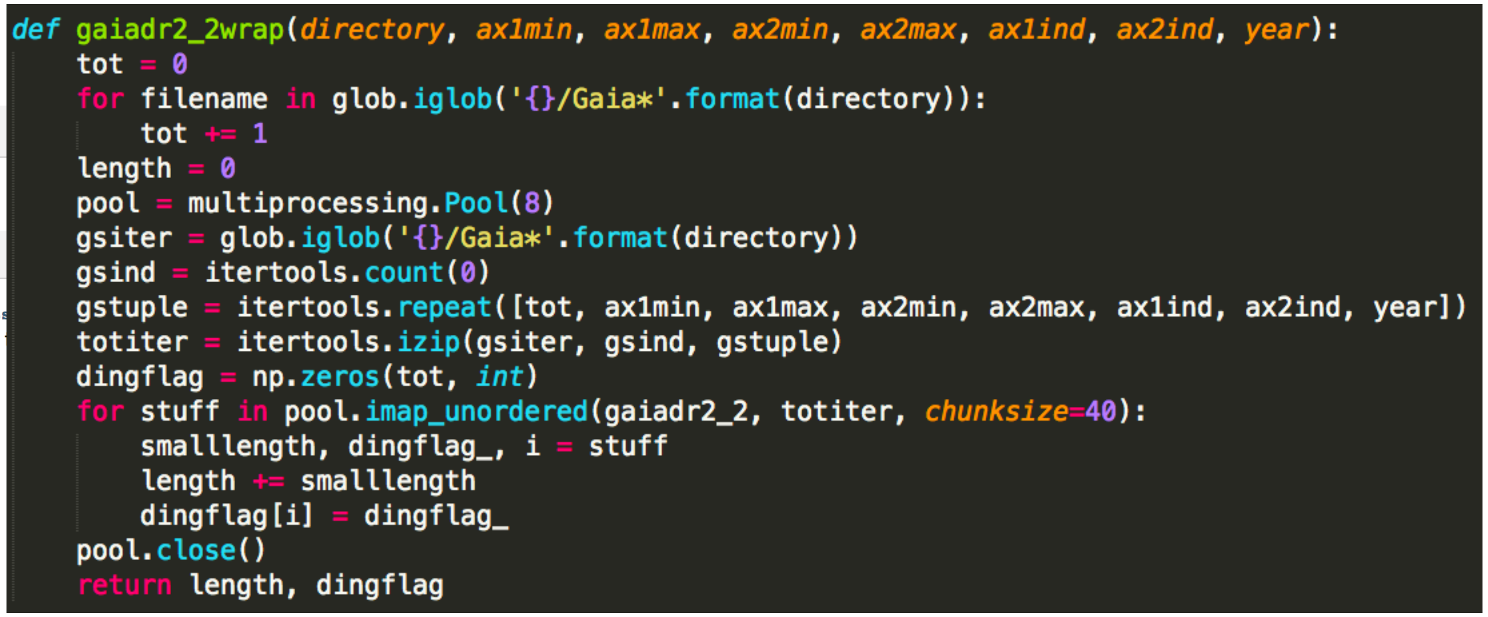


16.6. `multiprocessing` — Process-based "threading" interface

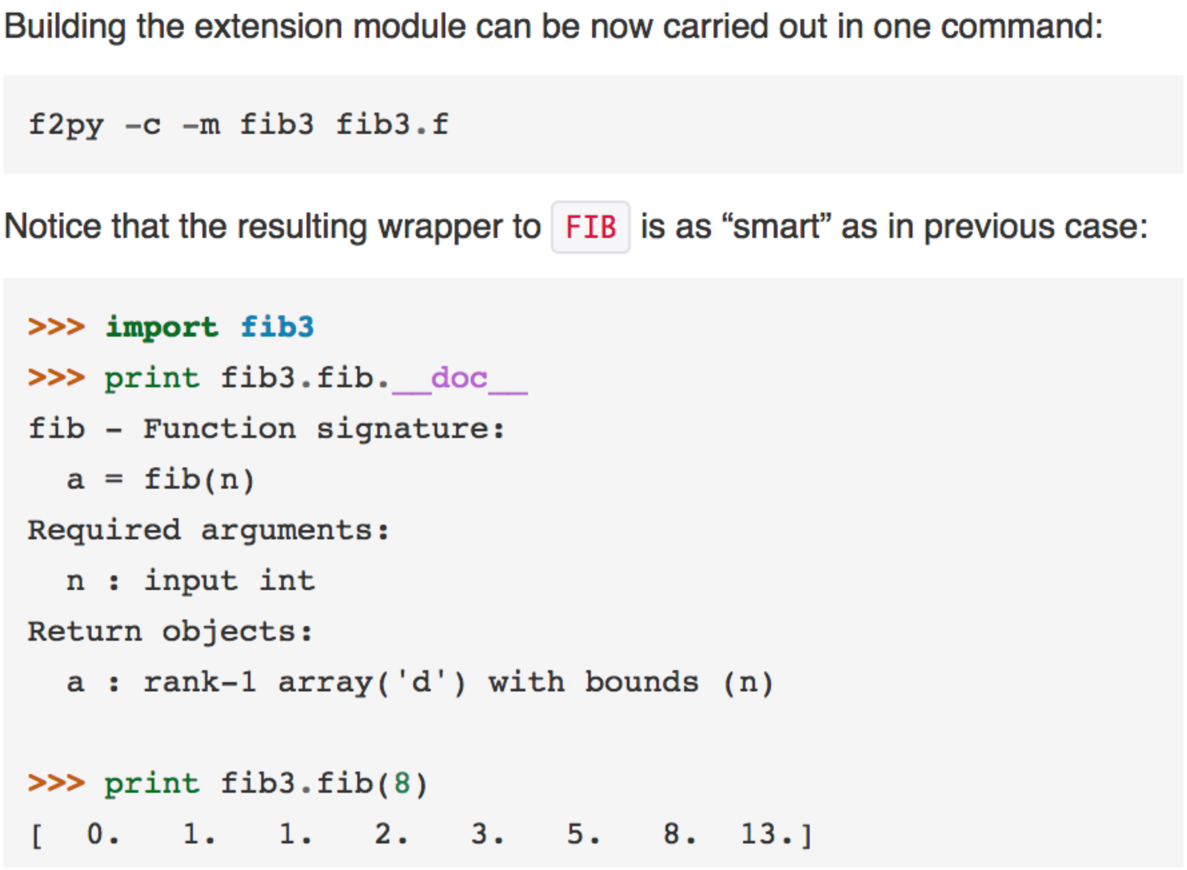
New in version 2.6.

16.6.1. Introduction

`multiprocessing` is a package that supports spawning processes using an API similar to the `threading` module. The `multiprocess`
remote concurrency, effectively side-stepping the Global Interpreter Lock by using subprocesses instead of threads. Due to this,
the programmer to fully leverage multiple processors on a given machine. It runs on both Unix and Windows.



```python
def gaiadr2_2wrap(directory, ax1min, ax1max, ax2min, ax2max, ax1ind, ax2ind, year):
    tot = 0
    for filename in glob.iglob('{}/Gaia*'.format(directory)):
        tot += 1
    length = 0
    pool = multiprocessing.Pool(8)
    gsiter = glob.iglob('{}/Gaia*'.format(directory))
    gsind = itertools.count(0)
    gstuple = itertools.repeat([tot, ax1min, ax1max, ax2min, ax2max, ax1ind, ax2ind, year])
    totiter = itertools.izip(gsiter, gsind, gstuple)
    dingflag = np.zeros(tot, int)
    for stuff in pool.imap_unordered(gaiadr2_2, totiter, chunksize=40):
        smalllength, dingflag_, i = stuff
        length += smalllength
        dingflag[i] = dingflag_
    pool.close()
    return length, dingflag
```

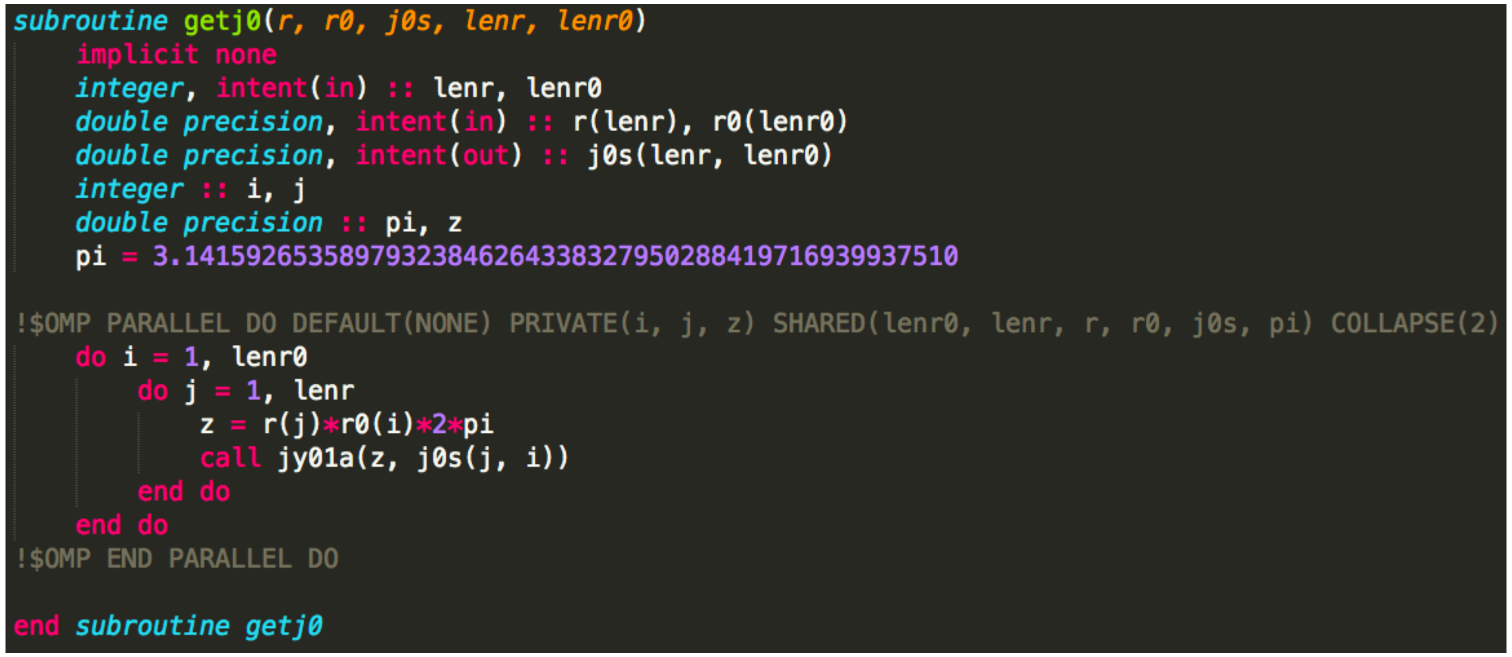


Building the extension module can be now carried out in one command:

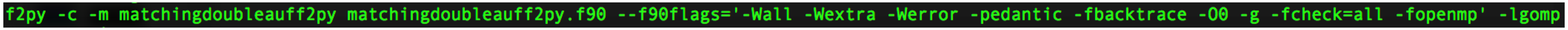
```
f2py -c -m fib3 fib3.f
```

Notice that the resulting wrapper to `FIB` is as "smart" as in previous case:

```
>>> import fib3
>>> print fib3.fib.__doc__
fib - Function signature:
  a = fib(n)
Required arguments:
  n : input int
Return objects:
  a : rank-1 array('d') with bounds (n)

>>> print fib3.fib(8)
[  0.   1.   1.   2.   3.   5.   8.  13.]
```

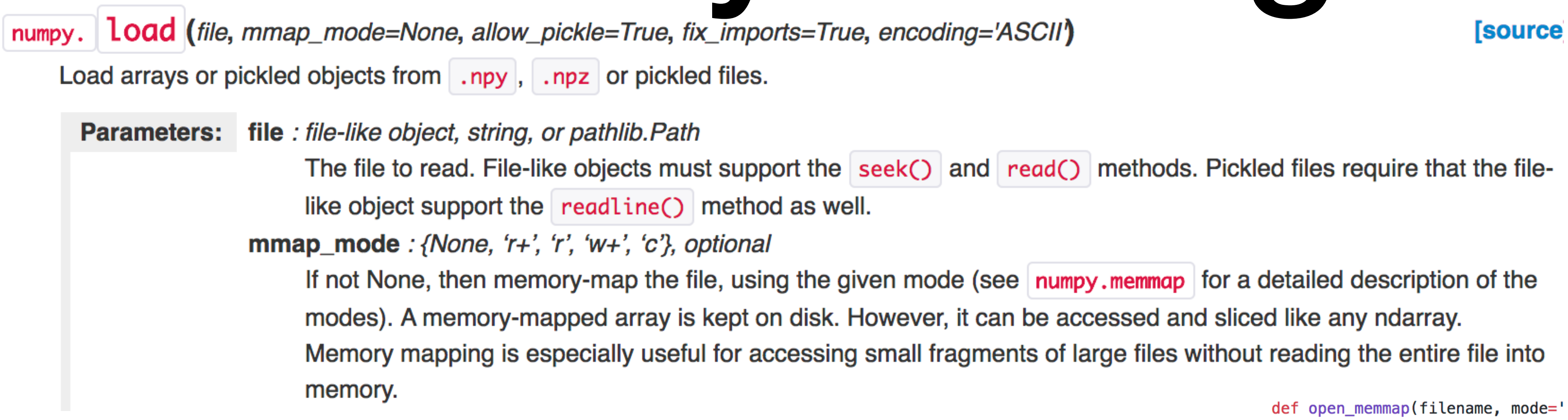


```fortran
subroutine getj0(r, r0, j0s, lenr, lenr0)
    implicit none
    integer, intent(in) :: lenr, lenr0
    double precision, intent(in) :: r(lenr), r0(lenr0)
    double precision, intent(out) :: j0s(lenr, lenr0)
    integer :: i, j
    double precision :: pi, z
    pi = 3.14159265358979323846264338327950288419716939937510

!$OMP PARALLEL DO DEFAULT(NONE) PRIVATE(i, j, z) SHARED(lenr0, lenr, r, r0, j0s, pi) COLLAPSE(2)
    do i = 1, lenr0
        do j = 1, lenr
            z = r(j)*r0(i)*2*pi
            call jy01a(z, j0s(j, i))
        end do
    end do
!$OMP END PARALLEL DO

end subroutine getj0
```



```
f2py -c -m matchingdoubleauff2py matchingdoubleauff2py.f90 --f90flags='-Wall -Wextra -Werror -pedantic -fbacktrace -O0 -g -fcheck=all -fopenmp' -lgomp
```

Memory Management



`numpy.` **`load`**(*file, mmap_mode=None, allow_pickle=True, fix_imports=True, encoding='ASCII'*) [source

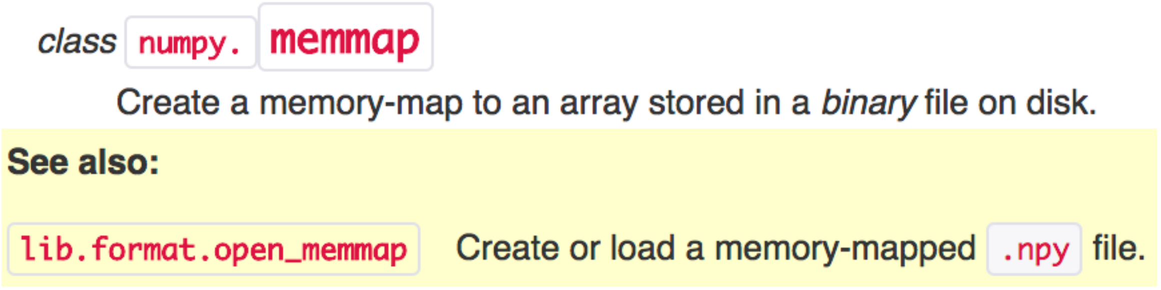
Load arrays or pickled objects from `.npy`, `.npz` or pickled files.

Parameters: **file** : *file-like object, string, or pathlib.Path*

The file to read. File-like objects must support the `seek()` and `read()` methods. Pickled files require that the file-like object support the `readline()` method as well.

mmap_mode : *{None, 'r+', 'r', 'w+', 'c'}, optional*

If not None, then memory-map the file, using the given mode (see `numpy.memmap` for a detailed description of the modes). A memory-mapped array is kept on disk. However, it can be accessed and sliced like any ndarray. Memory mapping is especially useful for accessing small fragments of large files without reading the entire file into memory.

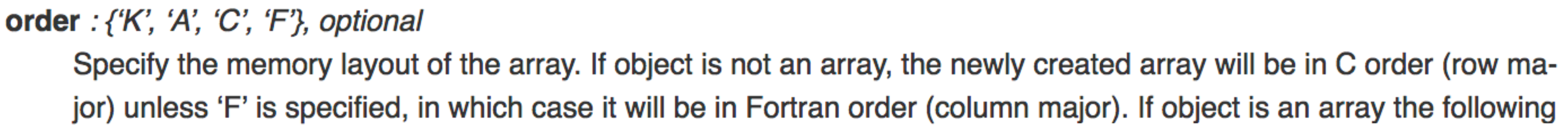


class `numpy.` **`memmap`**

Create a memory-map to an array stored in a *binary* file on disk.

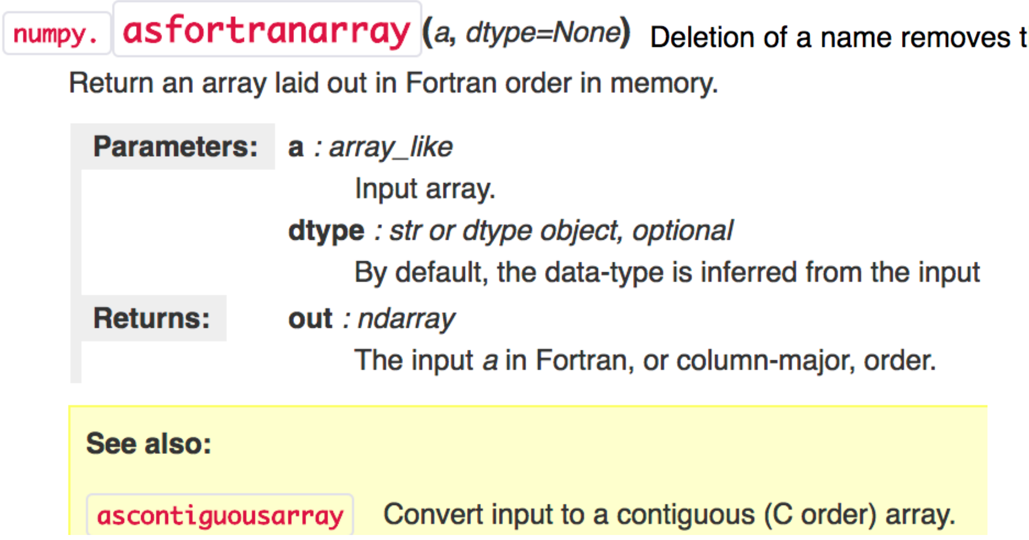
See also:

`lib.format.open_memmap` Create or load a memory-mapped `.npy` file.



order : *{'K', 'A', 'C', 'F'}, optional*

Specify the memory layout of the array. If object is not an array, the newly created array will be in C order (row major) unless 'F' is specified, in which case it will be in Fortran order (column major). If object is an array the following holds.



`numpy.` **`asfortranarray`**(*a, dtype=None*)

Return an array laid out in Fortran order in memory.

Parameters: **a** : *array_like*

Input array.

dtype : *str or dtype object, optional*

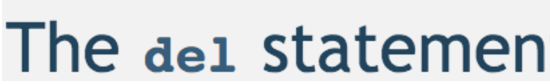
By default, the data-type is inferred from the input

Returns: **out** : *ndarray*

The input *a* in Fortran, or column-major, order.

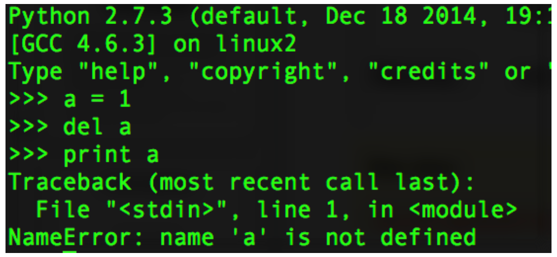
See also:

`ascontiguousarray` Convert input to a contiguous (C order) array.

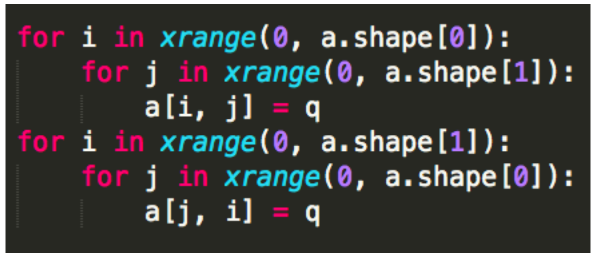


The `del` statement

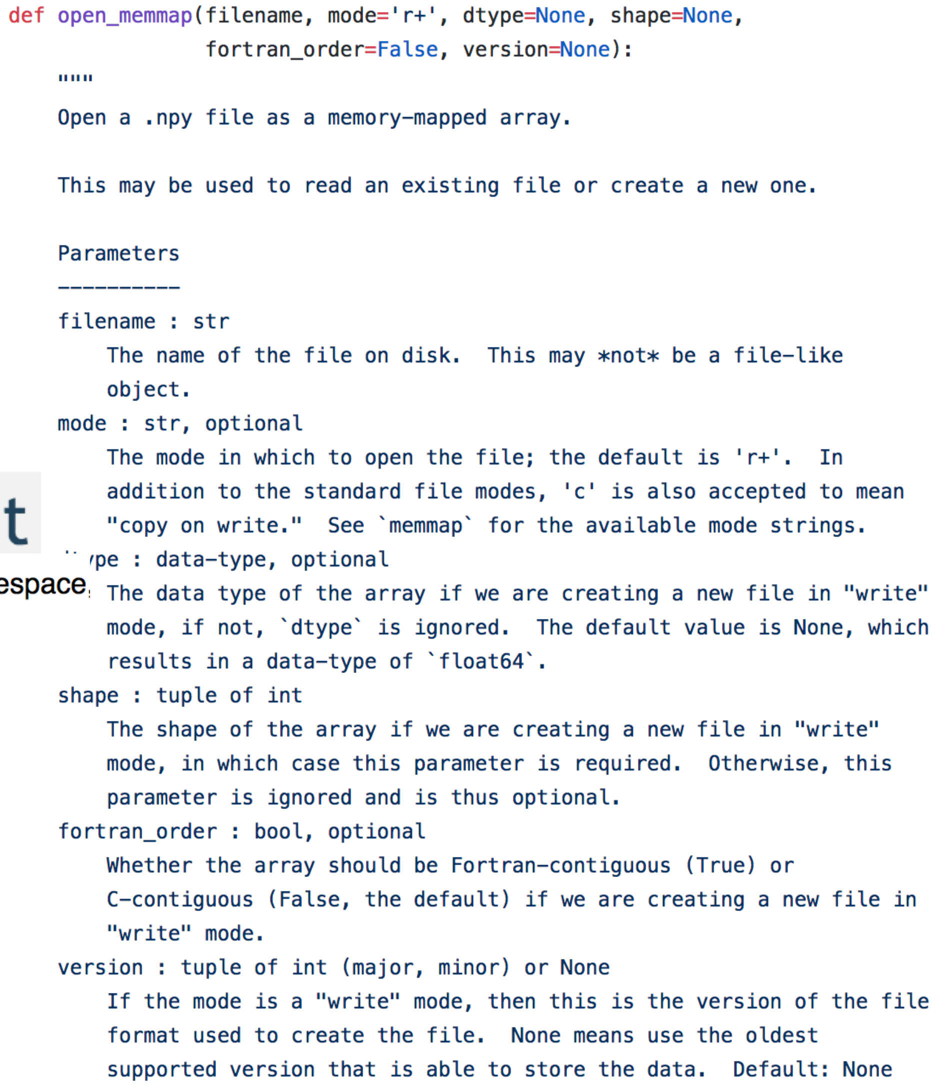
Deletion of a name removes the binding of that name from the local or global namespace.



```
Python 2.7.3 (default, Dec 18 2014, 19:1
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "
>>> a = 1
>>> del a
>>> print a
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'a' is not defined
```



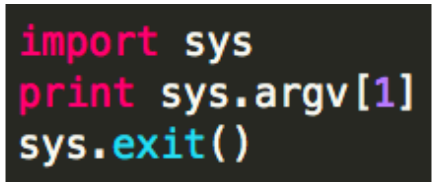
```
for i in xrange(0, a.shape[0]):
    for j in xrange(0, a.shape[1]):
        a[i, j] = q
for i in xrange(0, a.shape[1]):
    for j in xrange(0, a.shape[0]):
        a[j, i] = q
```



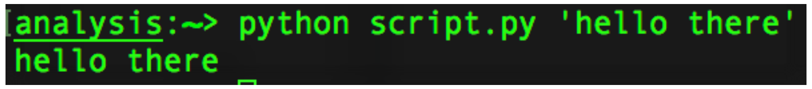
```
def open_memmap(filename, mode='r+', dtype=None, shape=None,
                fortran_order=False, version=None):
    """
    Open a .npy file as a memory-mapped array.

    This may be used to read an existing file or create a new one.

    Parameters
    ----------
    filename : str
        The name of the file on disk.  This may *not* be a file-like
        object.
    mode : str, optional
        The mode in which to open the file; the default is 'r+'.  In
        addition to the standard file modes, 'c' is also accepted to mean
        "copy on write."  See `memmap` for the available mode strings.
    dtype : data-type, optional
        The data type of the array if we are creating a new file in "write"
        mode, if not, `dtype` is ignored.  The default value is None, which
        results in a data-type of `float64`.
    shape : tuple of int
        The shape of the array if we are creating a new file in "write"
        mode, in which case this parameter is required.  Otherwise, this
        parameter is ignored and is thus optional.
    fortran_order : bool, optional
        Whether the array should be Fortran-contiguous (True) or
        C-contiguous (False, the default) if we are creating a new file in
        "write" mode.
    version : tuple of int (major, minor) or None
        If the mode is a "write" mode, then this is the version of the file
        format used to create the file.  None means use the oldest
        supported version that is able to store the data.  Default: None
```

Making Life A Bit Easier

import sys
sys.exit()
sys.stdout.flush()
sys.argv



```
import sys
print sys.argv[1]
sys.exit()
```



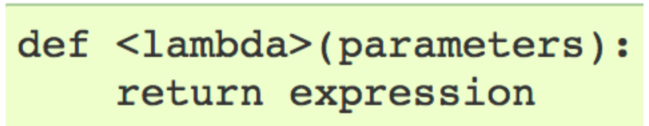
```
analysis:~> python script.py 'hello there'
hello there
```

nohup python script.py >& log.log &

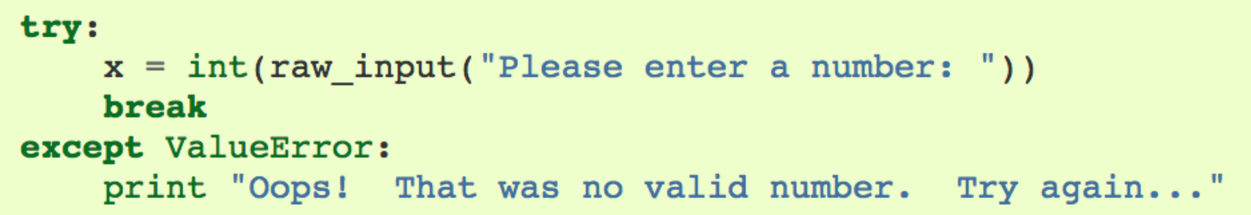
if __name__ == ‘__main__’:
useful for differentiating between a python script imported into another, and a script run from the terminal

a = [i2 for i in b]**
a = 1 if b > 0 else 0

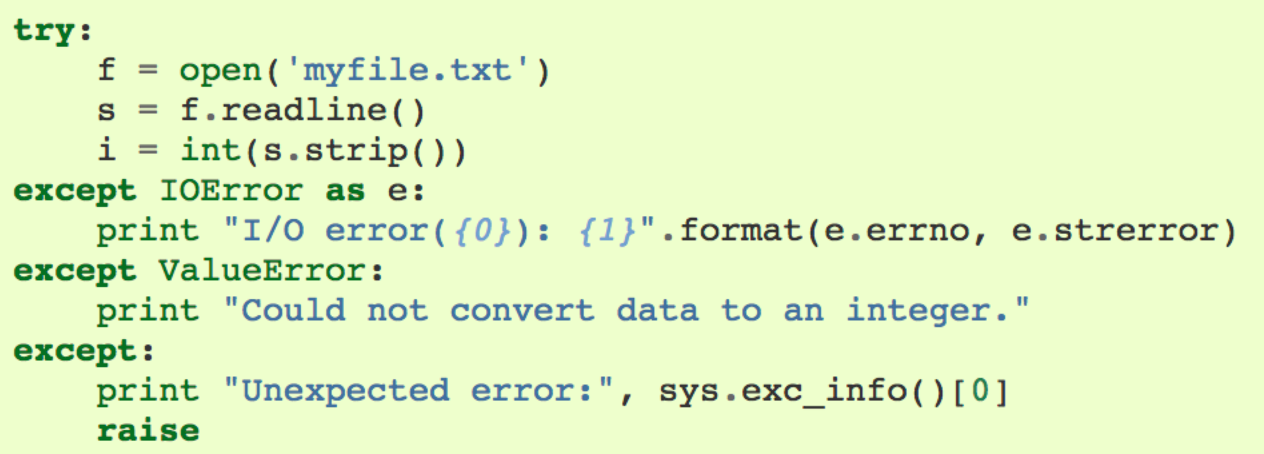
lambda a, b: a+b



```
def <lambda>(parameters):
    return expression
```



```
try:
    x = int(raw_input("Please enter a number: "))
    break
except ValueError:
    print "Oops!  That was no valid number.  Try again..."
```



```
try:
    f = open('myfile.txt')
    s = f.readline()
    i = int(s.strip())
except IOError as e:
    print "I/O error({0}): {1}".format(e.errno, e.strerror)
except ValueError:
    print "Could not convert data to an integer."
except:
    print "Unexpected error:", sys.exc_info()[0]
    raise
```

Conclusions

- Avoid writing pure-python functions of your own to do basic (or even intermediate) mathematical/graphical things unless absolutely necessary
- Numpy arrays can be thrown around quite a lot, slice liberally (while ensuring views rather than copies where possible) to manipulate arrays making code easier to understand.
- Matplotlib can do a lot of very fancy things if needed - or just save a set of paper-worthy settings and forget.
- Python is fantastic for file IO and N-D array stacking.
- Multiple options to minimise memory usage and maximise parallelisation.
- Easier to ask forgiveness than permission.

Conclusions

Read the Documentation.

Conclusions

No, really, read the documentation.

Conclusions

Seriously, it probably already exists, you should read the documentation.

Conclusions

Read the documentation.

Conclusions

Read the documentation.

Conclusions

- Avoid writing pure-python functions of your own to do basic (or even intermediate) mathematical/graphical things unless absolutely necessary
- Numpy arrays can be thrown around quite a lot, slice liberally (while ensuring views rather than copies where possible) to manipulate arrays making code easier to understand.
- Matplotlib can do a lot of very fancy things if needed - or just save a set of paper-worthy settings and forget.
- Python is fantastic for file IO and N-D array stacking.
- Multiple options to minimise memory usage and maximise parallelisation.
- Easier to ask forgiveness than permission.