Python For Data Science *Cheat Sheet*

NumPy Basics

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

NumPy

The **NumPy** library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

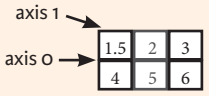
Use the following import convention:

```
>>> import numpy as np
```

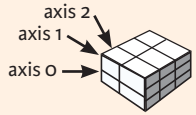
NumPy Arrays

1D array

2D array



3D array



Creating Arrays

```
>>> a = np.array([1,2,3])
>>> 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)
```

Initial Placeholders

```
>>> np.zeros((3,4))                    Create an array of zeros
>>> np.ones((2,3,4),dtype=np.int16)    Create an array of ones
>>> d = np.arange(10,25,5)             Create an array of evenly
                                       spaced values (step value)
>>> np.linspace(0,2,9)                 Create an array of evenly
                                       spaced values (number of samples)
>>> e = np.full((2,2),7)               Create a constant array
>>> f = np.eye(2)                      Create a 2X2 identity matrix
>>> np.random.random((2,2))            Create an array with random values
>>> np.empty((3,2))                    Create an empty array
```

I/O

Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.savez('array.npz', a, b)
>>> np.load('my_array.npy')
```

Saving & Loading Text Files

```
>>> np.loadtxt("myfile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=',')
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

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
```

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
```

Asking For Help

```
>>> np.info(np.ndarray.dtype)
```

Array Mathematics

Arithmetic Operations

```
>>> g = a - b                          Subtraction
  array([[-0.5,  0. ,  0. ],
         [-3. , -3. , -3. ]])
>>> np.subtract(a,b)                   Subtraction
>>> b + a                              Addition
  array([[ 2.5,  4. ,  6. ],
         [ 5. ,  7. ,  9. ]])
>>> np.add(b,a)                        Addition
>>> a / b                              Division
  array([[ 0.66666667,  1.        ,  1.        ],
         [ 0.25      ,  0.4       ,  0.5       ]])
>>> np.divide(a,b)                     Division
>>> a * b                              Multiplication
  array([[  1.5,   4. ,   9. ],
         [  4. ,  10. ,  18. ]])
>>> np.multiply(a,b)                   Multiplication
>>> np.exp(b)                          Exponentiation
>>> np.sqrt(b)                         Square root
>>> np.sin(a)                          Print sines of an array
>>> np.cos(b)                          Element-wise cosine
>>> np.log(a)                          Element-wise natural logarithm
>>> e.dot(f)                           Dot product
  array([[ 7.,  7.],
         [ 7.,  7.]])
```

Comparison

```
>>> a == b                             Element-wise comparison
  array([[False,  True,  True],
         [False, False, False]], dtype=bool)
>>> a < 2                              Element-wise comparison
  array([True, False, False], dtype=bool)
>>> np.array_equal(a, b)               Array-wise comparison
```

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
```

Copying Arrays

```
>>> h = a.view()          Create a view of the array with the same data
>>> np.copy(a)            Create a copy of the array
>>> h = a.copy()          Create a deep copy of the array
```

Sorting Arrays

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

Subsetting, Slicing, Indexing

Also see Lists

Subsetting


```
>>> a[2]
  3
```

Select the element at the 2nd index


```
>>> b[1,2]
  6.0
```

Select the element at row 1 column 2 (equivalent to `b[1][2]`)

Slicing


```
>>> a[0:2]
  array([1, 2])
```

Select items at index 0 and 1


```
>>> b[0:2,1]
  array([ 2.,  5.])
```

Select items at rows 0 and 1 in column 1


```
>>> b[:1]
  array([[1.5, 2., 3.]])
```

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`

Boolean Indexing


```
>>> a[a<2]
  array([1])
```

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                        Permute array dimensions
```

Changing Array Shape

```
>>> b.ravel()                  Flatten the array
>>> g.reshape(3,-2)            Reshape, but don't change data
```

Adding/Removing Elements

```
>>> h.resize((2,6))            Return a new array with shape (2,6)
>>> np.append(h,g)             Append items to an array
>>> np.insert(a, 1, 5)         Insert items in an array
>>> np.delete(a,[1])           Delete items from an array
```

Combining Arrays

```
>>> np.concatenate((a,d),axis=0)        Concatenate arrays
  array([ 1,  2,  3, 10, 15, 20])
>>> np.vstack((a,b))                    Stack arrays vertically (row-wise)
  array([[ 1. ,  2. ,  3. ],
         [ 1.5,  2. ,  3. ],
         [ 4. ,  5. ,  6. ]])
>>> np.r_[e,f]                          Stack arrays vertically (row-wise)
>>> np.hstack((e,f))                    Stack arrays horizontally (column-wise)
  array([[ 7.,  7.,  1.,  0.],
         [ 7.,  7.,  0.,  1.]])
>>> np.column_stack((a,d))              Create stacked column-wise arrays
  array([[ 1, 10],
         [ 2, 15],
         [ 3, 20]])
>>> np.c_[a,d]                          Create stacked column-wise arrays
```

Splitting Arrays

```
>>> np.hsplit(a,3)                      Split the array horizontally at the 3rd index
  [array([1]),array([2]),array([3])]
>>> np.vsplit(c,2)                      Split the array vertically at the 2nd index
[array([[[ 1.5,  2. ,  1. ],
         [ 4. ,  5. ,  6. ]]]),
 array([[[ 3.,  2.,  3.],
         [ 4.,  5.,  6.]]])]
```

Python & Pylab Cheat Sheet

Running

`python3`	standard python shell.
`ipython3`	improved interactive shell.
`ipython3 --pylab`	ipython including pylab
`python3` *file.py*	run *file.py*
`python3 -i` *file.py*	run *file.py*, stay in interactive mode

To quit use `exit()` or [ctrl]+[d]

Getting Help

`help()`	interactive Help
`help(`*object*`)`	help for *object*
object`?`	ipython: help for *object*
object`??`	ipython: extended help for *object*
`%magic`	ipython: help on magic commands

Import Syntax, e.g. for π

`import numpy`	use: `numpy.pi`
`import numpy as np`	use: `np.pi`
`from numpy import pi`	use: `pi`
`from numpy import *`	use: `pi` (use sparingly)

Types

`i = 1`	Integer		
`f = 1.`	Float		
`c = 1+2j`	Complex	with this:	
`True/False`	Boolean	`c.real`	1.0
`'abc'`	String	`c.imag`	2.0
`"abc"`	String	`c.conjugate()`	1-2j

Operators

mathematics		comparison	
`+`	addition	`=`	assign
`-`	subtraction	`==`	equal
`*`	multiplication	`!=`	unequal
`i/i`	int division	`<`	less
`i/f`	float division	`<=`	less-equal
`**`	power	`>=`	greater-equal
`%`	modulo	`>`	greater

Basic Syntax

`raw_input('`*foo*`')`	read string from command-line
`class` *Foo*`(Object):` ...	class definition
`def` *bar*`(args):` ...	function/method definition
`if` *c*`:` ... `elif` *c*`:` ... `else:`	branching
`try:` ... `except` *Error*`:` ...	exception handling
`while` *cond*`:` ...	while loop
`for` *item* `in` *list*`:` ...	for loop
[*item* `for` *item* `in` *list*`]`	for loop, list notation

Useful tools

`pylint` *file.py*	static code checker
`pydoc` *file*	parse docstring to man-page
`python3 -m doctest` *file.py*	run examples in docstring
`python3 -m pdb` *file.py*	run in debugger

NumPy & Friends

The following import statement is assumed:
`from pylab import *`

General Math

`f`: float, `c`: complex:

`abs(c)`	absolute value of f or c
`sign(c)`	get sign of f or c
`fix(f)`	round towards 0
`floor(f)`	round towards $-\inf$
`ceil(f)`	round towards $+\inf$
`f.round(p)`	round f to p places
`angle(c)`	angle of complex number
`sin(c)`	sinus of argument
`arcsin(c)`	arcsin of argument
`cos, tan,...`	analogous

Defining Lists, Arrays, Matrices

`l`: list, `a`: array:

`[[1,2],[3,4,5]]`	basic list
`array([[1,2],[3,4]])`	array from "rectangular" list
`matrix([[1,2],[3,4]])`	matrix from 2d-list
`range(min, max, step)`	integers in [min, max)
`list(range(...))`	list from range()
`arange(min, max, step)`	integer array in [min, max)
`frange(min, max, step)`	float array in [min, max]
`linspace(min, max, num)`	num samples in [min, max]
`meshgrid(x,y)`	create coord-matrices
`zeros`, `ones`, `eye`	generate special arrays

Element Access

`l[row][col]`	list: basic access
`l[min:max]`	list: range access [min,max)
`a[row,col]` or `a[row][col]`	array: basic access
`a[min:max,min:max]`	array: range access [min,max)
`a[`*list*`]`	array: select indices in *list*
`a[np.where(`*cond*`)]`	array: select where *cond* true

List/Array Properties

`len(l)`	size of first dim
`a.size`	total number of entries
`a.ndim`	number of dimensions
`a.shape`	size along dimensions
`ravel(l)` or `a.ravel()`	convert to 1-dim
`a.flat`	iterate all entries

Matrix Operations

`a`: array, `M`: matrix:

`a*a`	element-wise product
`dot(a,a)` or `M*M`	dot product
`cross(a,a)`	cross product
`inv(a)` or `M.I`	inverted matrix
`transpose(a)` or `M.T`	transposed matrix
`det(a)`	calculate determinate

Statistics

`sum(l,d)` or `a.sum(d)`	sum elements along d
`mean(l,d)` or `a.mean(d)`	mean along d
`std(l,d)` or `a.std(d)`	standard deviation along d
`min(l,d)` or `a.min(d)`	minima along d
`max(l,d)` or `a.max(d)`	maxima along d

Misc functions

`loadtxt(`*file*`)`	read values from *file*
`polyval(coeff,xvals)`	evaluate polynomial at xvals
`roots(coeff)`	find roots of polynomial
`map(`*func*`,`*list*`)`	apply func on each element of list

Plotting

Plot Types

`plot(xvals, yvals, 'g+')`	mark 3 points with green +
`errorbar()`	like `plot` with error bars
`semilogx()`, `semilogx()`	like `plot`, semi-log axis
`loglog()`	double logarithmic plot
`polar(phi_vals, rvals)`	plot in polar coordinates
`hist(vals, n_bins)`	create histogram from values
`bar(low_edge, vals, width)`	create bar-plot
`contour(xvals,yvals,zvals)`	create contour-plot

Pylab Plotting Equivalences

`figure()`	`fig = figure()`
	`ax = axes()`
`subplot(2,1,1)`	`ax = fig.add_subplot(2,1,1)`
`plot()`	`ax.plot()`
`errorbar()`	`ax.errorbar()`
`semilogx, ...`	analogous
`polar()`	`axes(polar=True)` and `ax.plot()`
`axis()`	`ax.set_xlim()`, `ax.set_ylim()`
`grid()`	`ax.grid()`
`title()`	`ax.set_title()`
`xlabel()`	`ax.set_xlabel()`
`legend()`	`ax.legend()`
`colorbar()`	`fig.colorbar(plot)`

Plotting 3D

```
from mpl_toolkits.mplot3d import Axes3D
```

`ax = fig.add_subplot(...,projection='3d')`	
or `ax = Axes3D(fig)`	create 3d-axes object
`ax.plot(xvals, yvals, zvals)`	normal plot in 3d
`ax.plot_wireframe`	wire mesh
`ax.plot_surface`	colored surface