**Inspecting Python For Data Science** *Cheat Sheet*

**Your Array** NumPy Basics Learn Python for Data Science **Interactively** at www.DataCamp.com

**NumPy**

**Asking For Help** >>> np.info(np.ndarray.dtype) **Sorting Arrays** >>> a.sort() Sort an array >>> c.sort(axis=0) Sort the elements of an array's axis

**DataCamp Learn Python for Data Science Interactively**

**Also see Lists**

**Subsetting** >>> a[2] Select the element at the 2nd index 3 >>> b[1,2] Select the element at row 1 column 2

6.0 (equivalent to b[1][2]) **Slicing** >>> a[0:2] Select items at index 0 and 1 array([1, 2]) >>> b[0:2,1] Select items at rows 0 and 1 in column 1

array([ 2., 5.])

>>> b[:1] Select all items at row 0

array([[1.5, 2., 3.]]) (equivalent to b[0:1, :]) >>> c[1,...] Same as [1,:,:]

array([[[ 3., 2., 1.], [ 4., 5., 6.]]]) >>> a[ : :-1] Reversed array a array([3, 2, 1])

**Boolean Indexing** >>> a[a<2] Select elements from a less than 2

array([1]) **Fancy Indexing** >>> b[[1, 0, 1, 0],[0, 1, 2, 0]] Select elements (1,0),(0,1),(1,2) and (0,0)

array([ 4. , 2. , 6. , 1.5]) >>> b[[1, 0, 1, 0]][:,[0,1,2,0]] array([[ [ [ [ 1.5, 4. 1.5, 4. , ,5. 2. 5. 2. , , , , 6. 3. 6. 3. , , , , 4. 1.5], 4. 1.5]])

], ], Select and columns a subset of the matrix’s rows

>>> 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**

**Aggregate Functions**

>>> np.loadtxt("myfile.txt") >>> np.genfromtxt("my\_file.csv", delimiter=',') >>> np.savetxt("myarray.txt", a, delimiter=" ")

>>> 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

1 2 3

1.5 2 3

4 5 6

**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

1 2 3

The **NumPy** library is the core library for scientific computing in

1.5 2 3 Python. It provides a high-performance multidimensional array

4 5 6

object, and tools for working with these arrays.

1.5 2 3

4 5 6

Use the following import convention:

>>> import numpy as np 1 2 3

**Creating Arrays**

**Arithmetic Operations**

**NumPy Arrays**

**1D array 2D array 3D array**

1 2 3

axis 1

axis 0

1.5 2 3 4 5 6

axis 2 axis 1axis 0

**Array Manipulation**

**Transposing Array** >>> i = np.transpose(b) Permute array dimensions >>> i.T Permute array dimensions

>>> np.zeros((3,4)) Create an array of zeros >>> np.ones((2,3,4),dtype=np.int16) Create an array of ones

**Changing Array Shape** >>> b.ravel() Flatten the array >>> d = np.arange(10,25,5) Create an array of evenly >>> g.reshape(3,-2) Reshape, but don’t change data

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

**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 >>> np.empty((3,2)) Create an empty array

**I/O**

**Combining Arrays** >>> np.concatenate((a,d),axis=0) Concatenate arrays

array([ 1, 2, 3, 10, 15, 20])

**Saving & Loading On Disk** >>> np.vstack((a,b)) Stack arrays vertically (row-wise) array([[ 1. , 2. , 3. ], [ 1.5, 2. , 3. ], >>> np.save('my\_array', a) >>> np.savez('array.npz', a, b) >>> np.load('my\_array.npy')

[ 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.], **Saving & Loading Text Files**

[ 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

**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

**Splitting Arrays** >>> np.hsplit(a,3) Split the array horizontally at the 3rd

[array([1]),array([2]),array([3])] index >>> 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.]]])]

>>> np.string\_ Fixed-length string type >>> np.unicode\_ Fixed-length unicode type

2

**Subsetting, Slicing, Indexing**

**Array Mathematics**

>>> 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

>>> 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