

Numpy Cheat Sheet

PYTHON PACKAGE

CREATED BY: ARIANNE COLTON AND SEAN CHEN

NUMPY (NUMERICAL PYTHON)

What is NumPy?

Foundation package for scientific computing in Python

Why NumPy?

- Numpy '**ndarray**' is a much more efficient way of storing and manipulating "**numerical data**" than the built-in Python data structures.
- Libraries written in lower-level languages, such as C, can operate on data stored in Numpy '**ndarray**' without copying any data.

N-DIMENSIONAL ARRAY (NDARRAY)

What is NdArray?

Fast and space-efficient multidimensional array (container for homogeneous data) providing vectorized arithmetic operations

Create NdArray	<code>np.array(seq1)</code> # seq1 - is any sequence like object, i.e. [1, 2, 3]
Create Special NdArray	<code>1, np.zeros(10)</code> # one dimensional ndarray with 10 elements of value 0 <code>2, np.ones(2, 3)</code> # two dimensional ndarray with 6 elements of value 1 <code>3, np.empty(3, 4, 5) *</code> # three dimensional ndarray of uninitialized values <code>4, np.eye(N) or np.identity(N)</code> # creates N by N identity matrix
NdArray version of Python's range	<code>np.arange(1, 10)</code>
Get # of Dimension	<code>ndarray1.ndim</code>
Get Dimension Size	<code>dim1size, dim2size, .. = ndarray1.shape</code>
Get Data Type **	<code>ndarray1.dtype</code>
Explicit Casting	<code>ndarray2 = ndarray1.astype(np.int32) ***</code>

* Cannot assume empty() will return all zeros. It could be garbage values.

** Default data type is '**np.float64**'. This is equivalent to Python's float type which is 8 bytes (64 bits); thus the name 'float64'.
*** If casting were to fail for some reason, '**TypeError**' will be raised.

SLICING (INDEXING/SUBSETTING)

- Slicing (i.e. `ndarray1[2:6]`) is a '**view**' on the original array. **Data is NOT copied**. Any modifications (i.e. `ndarray1[2:6] = 8`) to the 'view' will be reflected in the original array.

- Instead of a 'view', explicit copy of slicing via :

```
ndarray1[2:6].copy()
```

- Multidimensional array indexing notation :

```
ndarray1[0][2] or ndarray1[0, 2]
```

* Boolean indexing :

```
ndarray1[(names == 'Bob') | (names == 'Will'), 2:]
```

'2:' means select from 3rd column on

* Selecting data by boolean indexing **ALWAYS** creates a copy of the data.

* The 'and' and 'or' keywords do NOT work with boolean arrays. Use & and |.

* Fancy indexing (aka 'indexing using integer arrays')

Select a subset of rows in a particular order :

```
ndarray1[ [3, 8, 4] ]  
ndarray1[ [-1, 6] ]
```

negative indices select rows from the end

* Fancy indexing **ALWAYS** creates a copy of the data.

NUMPY (NUMERICAL PYTHON)

Setting data with assignment :

```
ndarray1[ndarray1 < 0] = 0 *
```

* If ndarray1 is two-dimensions, `ndarray1 < 0` creates a two-dimensional boolean array.

COMMON OPERATIONS

1. Transposing

- A special form of reshaping which returns a '**view**' on the underlying data without copying anything.

```
ndarray1.transpose() or  
ndarray1.T or  
ndarray1.swapaxes(0, 1)
```

2. Vectorized wrappers (for functions that take scalar values)

- `math.sqrt()` works on only a scalar

```
np.sqrt(seq1) # any sequence (list,  
ndarray, etc) to return a ndarray
```

3. Vectorized expressions

- `np.where(cond, x, y)` is a vectorized version of the expression 'x if condition else y'

```
np.where([True, False], [1, 2],  
[2, 3]) => ndarray (1, 3)
```

- Common Usages :

```
np.where(matrixArray > 0, 1, -1)  
=> a new array (same shape) of 1 or -1 values
```

```
np.where(cond, 1, 0).argmax() *
```

=> Find the first True element

* `argmax()` can be used to find the index of the maximum element. Example usage is find the first element that has a "price > number" in an array of price data.

4. Aggregations/Reductions Methods (i.e. mean, sum, std)

Compute mean	<code>ndarray1.mean()</code> or <code>np.mean(ndarray1)</code>
Compute statistics over axis *	<code>ndarray1.mean(axis = 1)</code> <code>ndarray1.sum(axis = 0)</code>

* axis = 0 means column axis, 1 is row axis.

5. Boolean arrays methods

Count # of 'Trues' in boolean array	<code>(ndarray1 > 0).sum()</code>
If at least one value is 'True'	<code>ndarray1.any()</code>
If all values are 'True'	<code>ndarray1.all()</code>

Note: These methods also work with non-boolean arrays, where non-zero elements evaluate to True.

6. Sorting

Inplace sorting	<code>ndarray1.sort()</code>
Return a sorted copy instead of inplace	<code>sorted1 = np.sort(ndarray1)</code>

7. Set methods

Return sorted unique values	<code>np.unique(ndarray1)</code>
Test membership of ndarray1 values in [2, 3, 6]	<code>resultBooleanArray = np.in1d(ndarray1, [2, 3, 6])</code>

- Other set methods : `intersect1d()`, `union1d()`, `setdiff1d()`, `setxor1d()`

8. Random number generation (np.random)

- Supplements the built-in Python random * with functions for efficiently generating whole arrays of sample values from many kinds of probability distributions.

```
samples = np.random.normal(size=(3, 3))
```

* Python built-in random ONLY samples one value at a time.

Created by Arianne Colton and Sean Chen

www.datasciencefree.com

Based on content from

'Python for Data Analysis' by Wes McKinney

Updated: August 18, 2016