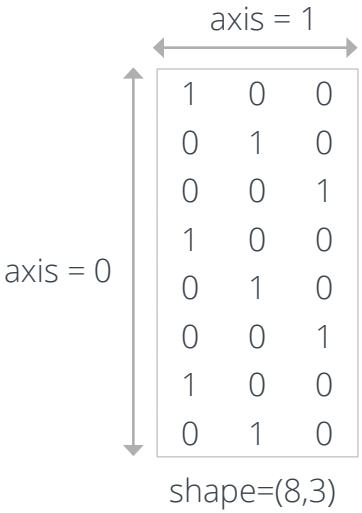
TOOL

NumPy Array Tip Sheet

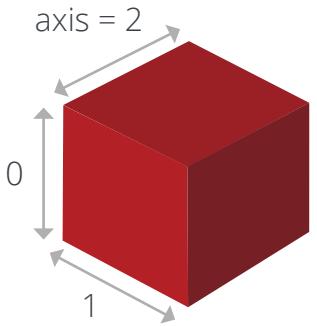
Arrays are the central data type introduced in the NumPy package. Technically, array objects are of type **`numpy.ndarray`**, which stands for "n-dimensional array." Arrays are accessible by importing the NumPy module, although we will use the conventional shorthand here: **`import numpy as np`**. Arrays are similar in some respects to Python lists but are multidimensional, homogeneous in type, and support compact and efficient array-level manipulations. Documentation can be found online at www.numpy.org/doc.

Anatomy of an array



The **axes** of an array describe the order of indexing into the array; e.g., axis=0 refers to the first index coordinate, axis=1 the second, etc.

The **shape** of an array is a tuple indicating the number of elements along each axis. An existing array **a** has an attribute **a.shape** which contains this tuple.



- All elements must be the same dtype (data type).
- The default dtype is float.
- Arrays constructed from a list of mixed dtype will be upcast to the "greatest" common type.

Constructing arrays

- `np.array(alist):` Construct an n-dimensional array from a Python list (all elements of list must be of same length).

```
a = np.array([[1,2,3],[4,5,6]])
b = np.array([i*i for i in range(100) if
i%2==1])
c = b.tolist()                          # convert array back to Python list
```

- `np.zeros(shape, dtype=float):` Construct an n-dimensional array of the specified shape, filled with zeros of the specified dtype.

```
a = np.zeros(100)                       # a 100-element array of float zeros
b = np.zeros((2,8), int)                # a 2x8 array of int zeros
c = np.zeros((N,M,L), bool)             # a NxMxL array of bool zeros
```

- `np.ones(shape, dtype=float):` Construct an n-dimensional array of the specified shape, filled with ones of the specified dtype.

```
a = np.ones(10, int)                    # a 10-element array of int ones
b = np.pi * np.ones((5,5))              # a useful way to fill up an array with a specified value
```

- `np.transpose(a)`

```
b = np.transpose(a)                     # return new array with a's dimensions reversed
b = a.T                                 # equivalent to np.transpose(a)
```

- np.arange and np.linspace

```
a = np.arange(start, stop, increment)        # like Python range, but with (potentially) real-valued arrays
b = np.linspace(start, stop, num_elements)   # create array of equally spaced points based on specifed number of points
```

- Random array constructors in np.random

```
a = np.random.random((100,100))              # 100x100 array of floats uniform on [0.,1.)
b = np.random.randint(0,10, (100,))          # 100 random ints uniform on [0, 10); i.e., not including the upper bound 10
c = np.random.standard_normal((5,5,5))       # zero-mean, unit-variance Gaussian random numbers in a 5x5x5 array
```

Indexing arrays

- Multidimensional indexing

```
elem = a[i,j,k]                              # extract element at position (i,j,k) in array
```

- "Negative" indexing (wrap around the end of the array)

```
last = a[-1]                                 # the last element of the array (returns array if a.ndim > 1)
last_elem = a[-1,-1]                         # the lower-right element of an array (returns array if a.ndim > 2)
```

- Arrays as indices

```
i = np.array([0,1,2,1])                          # array of indices for the first axis
j = np.array([1,2,3,4])                          # array of indices for the second axis
a[i,j]                                           # return array([a[0,1], a[1,2], a[2,3], a[1,4]]) based on
#                                                  i and j above
b = np.array([True, False, True, False])
a[b]                                             # return array([a[0], a[2]]) since only b[0] and b[2] are
                                                   True
```

Slicing arrays (extracting subsections)

- Slice a defined subblock

```
section = a[10:20, 30:40]                        # 10x10 subblock starting at [10,30]
```

- Grab everything up to the beginning/end of the array

```
asection = a[10:, 30:]                           # missing stop index implies until end of array
bsection = b[:10, :30]                           # missing start index implies until start of array
```

- Grab an entire column

```
x = a[:, 0]                                      # get everything in the 0th column (missing start and
                                                   stop)
y = a[:, 1]                                      # get everything in the 1st column
```

- Slice off the tail end of an array

```
tail = a[-10:]                                   # grab the last 10 elements of the array
slab = b[:, -10:]                                # grab a slab of width 10 off the “side” of the array
interior = c[1:-1, 1:-1, 1:-1]                   # slice out everything but the outer shell
```

Element-wise functions on arrays

- Arithmetic operations

```
c = a + b                                  # add a and b element-wise (must be same shape)
d = c + 2                                  # add 2 to every element of c
h = e * f                                  # multiply e and f element-wise (NOT matrix multiplication)
g = -h                                     # negate every element of h
m = c ** 2                                 # compute the square of every element of c
z = w > 0.0                                # return Boolean array indicating which elements are > 0.0
logspace = 10**np.linspace(-6, -1, 50)     # array of 50 equally spaced-in log points between 1.0e-06 and 1.0e-01
```

- Trigonometric operations

```
y = np.sin(x)                              # sin of every element of x
w = np.cos(2*np.pi*x)                      # cos of 2*pi*x
```

Summation of arrays

- Simple sums

```
s  = np.sum(a)                             # sum all elements in a, returning a scalar (single number)
s0 = np.sum(a, axis=0)                     # sum elements along specified axis (=0), returning an array of remaining shape
s1 = np.sum(a, axis=1)                     # sum elements along axis=1; i.e., over rows if a is 2-dimensional
```

- Averaging, etc.

```
m = np.mean(a, axis)                    # compute mean along the specified axis (over entire
                                          array if axis=None)
s = np.std(a, axis)                     # compute standard deviation along the specified axis
                                          (over entire array if axis=None)
```

- Cumulative sums

```
s0 = np.cumsum(a, axis=0)               # cumulatively sum over 0 axis, returning array with
                                          same shape as a
s0 = np.cumsum(a)                       # cumulatively sum over 0 axis, returning 1D array of
                                          length shape[0]*shape[1]*...*shape[dim-1]
```

Some other useful functions and methods

Many of these work both as separate functions (e.g., `np.sum(a)`) as well as array methods (e.g., `a.sum()`).

- `np.any(a):` Return True if any element of a is True.
- `np.all(a):` Return True if all elements of a are True.
- `np.concatenate((a1, a2, ...), axis):` Concatenate tuple of arrays along specified axis.
- `np.min(a, axis=None), np.max(a, axis=None):` Get min/max values of a along specified axis (global min/max if axis=None).
- `np.argmin(a, axis=None), np.argmax(a, axis=None):` Get indices of min/max of a along specified axis (global min/max if axis=None).
- `np.reshape(a, newshape):` Reshape a to new shape (must conserve total number of elements).
- `np.histogram, np.histogram2d, np.histogramdd:` 1-dimensional, 2-dimensional, and d-dimensional histograms, respectively.
- `np.round(a, decimals=0):` Round elements of array a to specified number of decimals.
- `np.sign(a):` Return array of same shape as a, with -1 where a < 0, 0 where a = 0, and +1 where a > 0.
- `np.abs(a):` Return array of same shape as a, with the absolute value of each corresponding element.
- `np.unique(a):` Return sorted unique elements of array a.
- `np.where(condition, x, y):` Return array with same shape as condition, where values from x are inserted in positions where condition is True, and values from y where condition is False.