Arrays

To and from Python lists:

```
x = np.array([1,2,3]) # from Python list
y = x.tolist()        # to Python list
```

Working with files:

```
x = np.genfromtxt('file.csv', delimiter=',')
x = np.savetxt('file.csv', a, delimiter=',')
```

Filled arrays:

```
np.zeros(3)
np.ones(3,4)
np.full((2,3),8)       # filled array
np.eye(5)              # 5x5 identity matrix
np.linspace(0,100,6)  # linear spacing
np.arange(0,10,3)      # start, stop, step
```

Array properties:

```
x.size                # # of elements in x
x.ndim                # # of axes in x
x.shape               # shape of x
x.dtype               # data type of x
y = x.astype(int)     # convert data type
```

Views, deep copying:

```
y = x.view()          # view of array x
y = x.copy()          # copy of array x
```

Reshaping, resizing:

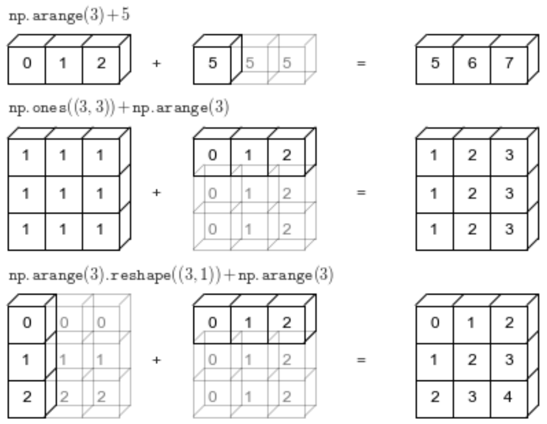
```
x.ravel()            # view of x as 1D
x.flatten()          # copy of x as 1D
x.T                  # view of x transposed
x.swapaxes(0,1)      # copy of x transposed
x.reshape((3,4))     # view with shape (3,4)
x.resize((3,4))      # resize array in place
np.resize(x,(3,4))   # resize array by copy
```

Concatenate, stack, split, squeeze:

```
np.concatenate((x,y), axis=0) # existing axis
np.stack((x,y), axis=1)      # new axis
np.split(x, 3, axis=0)       # equal size split
np.split(x, [2,3,5], axis=0) # custom intervals
x.squeeze() # view of x with unit dims removed
```

Broadcasting rules:

> When operating on two arrays, NumPy compares their shapes element-wise. It starts with the trailing dimensions and works its way foward.
>
> Two dimensions are compatible when
>
> (1) they are equal, or
> (2) one of them is 1


np.arange(3)+5
np.ones((3,3))+np.arange(3)
np.arange(3).reshape((3,1))+np.arange(3)

Array slicing:

```
x[::-1]        # reversed view of x
x[:5]          # first 5 elements of x
x[5:]          # after index 5 inclusive
x[4:7]         # 4 to 7 noninclusive
x[1::2]   # every other elem, idx 1 onwards
x[5::-2]  # every other elem, idx 5 down to 0
x[:,np.newaxis] # convert to column vector
```

Boolean indexing:

```
x > 5          # boolean mask of indices > 5
x[x > 5]       # array of elements > 5
```

Fancy indexing:

```
# 1D array of elems at idx 3,7,4
x[[3,7,4]]
# 1D array of elems at idx (0,2),(1,1),(2,3)
x[[0,1,2],[2,1,3]]
# 2D array of elems with broadcast indexing
x[row[:,np.newaxis],col]
```

Linear Algebra

```
np.dot(a,b)                  # dot product
np.inner(a,b)                # inner product
np.outer(a,b)                # outer product
np.matmul(a,b)               # matrix mult
np.kron(a,b)                 # kronecker product
np.linalg.matrix_power(a,n) # matrix power
np.linalg.svd(a)             # SVD
np.linalg.eig(a)             # eigendecomp
```

Random

```
np.random.rand(4,5)  # 4x5 floats in [0,1]
np.random.randint(5, size=(2,3)) # in [0,5)
```

Statistics

```
x.sum(axis=0)     # sum
x.mean(axis=0)    # mean
x.median(axis=0) # median
x.var(axis=0)     # variance
x.std(axis=0)     # standard deviation
x.min(axis=0)     # minimum
x.max(axis=0)     # maximum
x.argmin(axis=0) # index of minimum
x.argmax(axis=0) # index of maximum
x.any(axis=0)     # true iff any nonzero
x.all(axis=0)     # true iff all nonzero
```