NumPy (`import numpy as np`)

array initialization

```
np.array([2, 3, 4])             # direct initialization
np.empty(20, dtype=np.float32)  # single precision array of size 20
np.zeros(200)                   # initialize 200 zeros
np.ones((3,3), dtype=np.int32)  # 3 x 3 integer matrix with ones
np.eye(200)                     # ones on the diagonal
np.zeros_like(a)                # array with zeros and the shape of a
np.linspace(0., 10., 100)       # 100 points from 0 to 10
np.arange(0, 100, 2)            # points from 0 to <100 with step 2
np.logspace(-5, 2, 100)         # 100 log-spaced from 1e-5 -> 1e2
np.copy(a)                      # copy array to new memory
```

indexing

```
a = np.arange(100)          # initialization with 0 - 99
a[:3] = 0                   # set the first three indices to zero
a[2:5] = 1                  # set indices 2-4 to 1
a[:-3] = 2                  # set all but last three elements to 2
a[start:stop:step]          # general form of indexing/slicing
a[None, :]                  # transform to column vector
a[[1, 1, 3, 8]]             # return array with values of the indices
a = a.reshape(10, 10)       # transform to 10 x 10 matrix
a.T                         # return transposed view
b = np.transpose(a, (1, 0)) # transpose array to new axis order
a[a < 2]                    # values with elementwise condition
```

array properties and operations

```
a.shape                # a tuple with the lengths of each axis
len(a)                 # length of axis 0
a.ndim                 # number of dimensions (axes)
a.sort(axis=1)         # sort array along axis
a.flatten()            # collapse array to one dimension
a.conj()               # return complex conjugate
a.astype(np.int16)     # cast to integer
a.tolist()             # convert (possibly multidimensional) array to list
np.argmax(a, axis=1)   # return index of maximum along a given axis
np.cumsum(a)           # return cumulative sum
np.any(a)              # True if any element is True
np.all(a)              # True if all elements are True
np.argsort(a, axis=1)  # return sorted index array along axis
np.where(cond)         # return indices where cond is True
np.where(cond, x, y)   # return elements from x or y depending on cond
```

boolean arrays

```
a < 2                       # returns array with boolean values
(a < 2) & (b > 10)          # elementwise logical and
(a < 2) | (b > 10)          # elementwise logical or
~a                          # invert boolean array
```

elementwise operations and math functions

```
a * 5             # multiplication with scalar
a + 5             # addition with scalar
a + b             # addition with array b
a / b             # division with b (np.NaN for division by zero)
np.exp(a)         # exponential (complex and real)
np.power(a, b)    # a to the power b
np.sin(a)         # sine
np.cos(a)         # cosine
np.arctan2(a, b)  # arctan(a/b)
np.arcsin(a)      # arcsin
np.radians(a)     # degrees to radians
np.degrees(a)     # radians to degrees
np.var(a)         # variance of array
np.std(a, axis=1) # standard deviation
```

inner/ outer products

```
np.dot(a, b)                # inner product: a_mi b_in
np.einsum('ij,kj->ik', a, b)  # einstein summation convention
np.sum(a, axis=1)           # sum over axis 1
np.abs(a)                   # return absolute values
a[None, :] + b[:, None]     # outer sum
a[None, :] * b[:, None]     # outer product
np.outer(a, b)              # outer product
np.sum(a * a.T)             # matrix norm
```

linear algebra/ matrix math

```
evals, evecs = np.linalg.eig(a)     # Find eigenvalues and eigenvectors
evals, evecs = np.linalg.eigh(a)    # np.linalg.eig for hermitian matrix
```

reading/ writing files

```
np.loadtxt(fname/fobject, skiprows=2, delimiter=',')   # ascii data from file
np.savetxt(fname/fobject, array, fmt='%.5f')           # write ascii data
np.fromfile(fname/fobject, dtype=np.float32, count=5)  # binary data from file
np.tofile(fname/fobject)                               # write (C) binary data
np.save(fname/fobject, array)                          # save as numpy binary (.npy)
np.load(fname/fobject, mmap_mode='c')                  # load .npy file (memory mapped)
```

interpolation, integration, optimization

```
np.trapz(a, x=x, axis=1)  # integrate along axis 1
np.interp(x, xp, yp)      # interpolate function xp, yp at points x
np.linalg.lstsq(a, b)     # solve a x = b in least square sense
```

fft

```
np.fft.fft(a)               # complex fourier transform of a
f = np.fft.fftfreq(len(a))  # fft frequencies
np.fft.fftshift(f)          # shifts zero frequency to the middle
np.fft.rfft(a)              # real fourier transform of a
np.fft.rfftfreq(len(a))     # real fft frequencies
```

rounding

```
np.ceil(a)   # rounds to nearest upper int
np.floor(a)  # rounds to nearest lower int
np.round(a)  # rounds to neares int
```

random variables

```
from np.random import normal, seed, rand, uniform, randint
normal(loc=0, scale=2, size=100)  # 100 normal distributed
seed(23032)                       # resets the seed value
rand(200)                         # 200 random numbers in [0, 1)
uniform(1, 30, 200)               # 200 random numbers in [1, 30)
randint(1, 16, 300)               # 300 random integers in [1, 16)
```