

Numpy Cheat Sheet:

Numpy

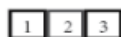
The NumPy library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays

Use the following import convention:

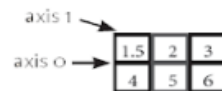
```
>>> import numpy as np
```

NumPy Arrays

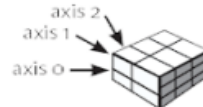
1D array



2D array



3D array



> Creating Arrays

```
>>> 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

```
>>> np.zeros((3,4)) #Create an array of zeros
>>> np.ones((2,3,4),dtype=np.int16) #Create an array of ones
>>> d = np.arange(10,25,5) #Create an array of evenly 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
>>> np.empty((3,2)) #Create an empty array
```

> I/O

Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.savez('array.npz', a, b)
>>> np.load('my_array.npy')
```

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
```

Aggregate Functions

```
>>> 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
>>> np.median(b) #Median
>>> np.corrcoef(a) #Correlation coefficient
>>> np.std(b) #Standard deviation
```

> Sorting Arrays

```
>>> a.sort() #Sort an array
>>> c.sort(axis=0) #Sort the elements of an array's axis
```

> Subsetting, Slicing, Indexing

Subsetting

```
>>> a[2] #Select the element at the 2nd index
3
>>> b[1,2] #Select the element at row 1 column 2 (equivalent to b[1][2])
6.0
```

1	2	3
1.5	2	3
4	5	6

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 (equivalent to b[0:1, :])
array([[1.5, 2., 3.]])
>>> c[1,...] #Same as [1,:,:]
array([[[ 3., 2., 1.],
        [ 4., 5., 6.]])])
>>> a[ : :-1] #Reversed array a array([3, 2, 1])
```

1	2	3
1.5	2	3
4	5	6
1.5	2	3
4	5	6

Boolean Indexing

```
>>> a[a<2] #Select elements from a less than 2
array([1])
```

1	2	3
---	---	---

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]] #Select a subset of the matrix's rows and columns
array([[ 4., 5., 6., 4. ],
       [ 1.5, 2., 3., 1.5],
       [ 4., 5., 6., 4. ],
       [ 1.5, 2., 3., 1.5]])
```