

# Python For Data Science Cheat Sheet

## NumPy Basics

Learn Python for Data Science Interactively at [www.DataCamp.com](https://www.datacamp.com)



### 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:



NumPy

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

### NumPy Arrays

#### 1D array

```
[1 2 3]
```

#### 2D array

axis 1  
axis 0

```
[[1.5 2. 3.]  
 [4. 5. 6.]]
```

#### 3D array

axis 2  
axis 1  
axis 0

```
[[[1.5 2. 3.]  
 [4. 5. 6.]]  
 [[1.5 2. 3.]  
 [4. 5. 6.]]  
 [[1.5 2. 3.]  
 [4. 5. 6.]]]
```

### 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))  
>>> np.ones((2,3,4),dtype=np.int16)  
>>> d = np.arange(10,25,5)  
  
>>> np.linspace(0,2,9)  
  
>>> e = np.full((2,2),7)  
>>> f = np.eye(2)  
>>> np.random.random((2,2))  
>>> np.empty((3,2))
```

Create an array of zeros  
Create an array of ones  
Create an array of evenly spaced values (step value)  
Create an array of evenly spaced values (number of samples)  
Create a constant array  
Create a 2X2 identity matrix  
Create an array with random values  
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')
```

#### Saving & Loading Text Files

```
>>> np.loadtxt("myfile.txt")  
>>> np.genfromtxt("my_file.csv", delimiter=',')  
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

### Data Types

```
>>> np.int64  
>>> np.float32  
>>> np.complex  
>>> np.bool  
>>> np.object  
>>> np.string_  
>>> np.unicode_
```

Signed 64-bit integer types  
Standard double-precision floating point  
Complex numbers represented by 128 floats  
Boolean type storing TRUE and FALSE values  
Python object type  
Fixed-length string type  
Fixed-length unicode type

### Inspecting Your Array

```
>>> a.shape  
>>> len(a)  
>>> b.ndim  
>>> e.size  
>>> b.dtype  
>>> b.dtype.name  
>>> b.astype(int)
```

Array dimensions  
Length of array  
Number of array dimensions  
Number of array elements  
Data type of array elements  
Name of data type  
Convert an array to a different type

### Asking For Help

```
>>> np.info(np.ndarray.dtype)
```

### Array Mathematics

#### Arithmetic Operations

```
>>> g = a - b  
array([[ -0.5,  0. ,  0. ],  
       [ -3. , -3. , -3. ]])  
>>> np.subtract(a,b)  
>>> b + a  
array([[ 2.5,  4. ,  6. ],  
       [ 5. ,  7. ,  9. ]])  
>>> np.add(b,a)  
>>> a / b  
array([[ 0.66666667,  1. ,  1. ],  
       [ 0.25 ,  0.4 ,  0.5 ]])  
>>> np.divide(a,b)  
>>> a * b  
array([[ 1.5,  4. ,  9. ],  
       [ 4. , 10. , 18. ]])  
>>> np.multiply(a,b)  
>>> np.exp(b)  
>>> np.sqrt(b)  
>>> np.sin(a)  
>>> np.cos(b)  
>>> np.log(a)  
>>> e.dot(f)  
array([[ 7. ,  7. ],  
       [ 7. ,  7.]])
```

Subtraction  
Subtraction  
Addition  
Addition  
Division  
Division  
Multiplication  
Multiplication  
Exponentiation  
Square root  
Print sines of an array  
Element-wise cosine  
Element-wise natural logarithm  
Dot product

#### Comparison

```
>>> a == b  
array([[False,  True,  True],  
       [False, False, False]], dtype=bool)  
>>> a < 2  
array([[True, False, False], dtype=bool)  
>>> np.array_equal(a, b)
```

Element-wise comparison  
Element-wise comparison  
Array-wise comparison

#### Aggregate Functions

```
>>> a.sum()  
>>> a.min()  
>>> b.max(axis=0)  
>>> b.cumsum(axis=1)  
>>> a.mean()  
>>> b.median()  
>>> a.corrcoef()  
>>> np.std(b)
```

Array-wise sum  
Array-wise minimum value  
Maximum value of an array row  
Cumulative sum of the elements  
Mean  
Median  
Correlation coefficient  
Standard deviation

### Copying Arrays

```
>>> h = a.view()  
>>> np.copy(a)  
>>> h = a.copy()
```

Create a view of the array with the same data  
Create a copy of the array  
Create a deep copy of the array

### Sorting Arrays

```
>>> a.sort()  
>>> c.sort(axis=0)
```

Sort an array  
Sort the elements of an array's axis

### Subsetting, Slicing, Indexing

Also see Lists

#### Subsetting

```
>>> a[2]  
3
```

```
[1 2 3]
```

Select the element at the 2nd index

```
>>> b[1,2]  
6.0
```

```
[[1.5 2. 3.]  
 [4. 5. 6.]]
```

Select the element at row 1 column 2 (equivalent to `b[1][2]`)

#### Slicing

```
>>> a[0:2]  
array([1, 2])
```

```
[1 2 3]
```

Select items at index 0 and 1

```
>>> b[0:2,1]  
array([ 2.,  5.])
```

```
[[1.5 2. 3.]  
 [4. 5. 6.]]
```

Select items at rows 0 and 1 in column 1

```
>>> b[:1]  
array([[1.5, 2., 3.]])
```

```
[[1.5 2. 3.]  
 [4. 5. 6.]]
```

Select all items at row 0 (equivalent to `b[0:1, :]`)

```
>>> c[1,...]  
array([[ 3.,  2.,  1.],  
       [ 4.,  5.,  6.]])
```

Same as `[1, :, :]`

```
>>> a[: :-1]  
array([3, 2, 1])
```

Reversed array `a`

#### Boolean Indexing

```
>>> a[a<2]  
array([1])
```

```
[1 2 3]
```

Select elements from `a` less than 2

#### Fancy Indexing

```
>>> b[[1, 0, 1, 0], [0, 1, 2, 0]]  
array([ 4. ,  2. ,  6. ,  1.5])
```

Select elements (1,0), (0,1), (1,2) and (0,0)

```
>>> b[[1, 0, 1, 0]][:, [0,1,2,0]]  
array([[ 4.,  5.,  6.,  4.],  
       [ 1.5,  2.,  3.,  1.5],  
       [ 4.,  5.,  6.,  4.],  
       [ 1.5,  2.,  3.,  1.5]])
```

Select a subset of the matrix's rows and columns

### Array Manipulation

#### Transposing Array

```
>>> i = np.transpose(b)  
>>> i.T
```

Permute array dimensions  
Permute array dimensions

#### Changing Array Shape

```
>>> b.ravel()
```

Flatten the array

```
>>> g.reshape(3,-2)
```

Reshape, but don't change data

#### Adding/Removing Elements

```
>>> h.resize((2,6))  
>>> np.append(h,g)  
>>> np.insert(a, 1, 5)  
>>> np.delete(a, [1])
```

Return a new array with shape (2,6)  
Append items to an array  
Insert items in an array  
Delete items from an array

#### Combining Arrays

```
>>> np.concatenate((a,d),axis=0)  
array([ 1,  2,  3, 10, 15, 20])  
>>> np.vstack((a,b))  
array([[ 1. ,  2. ,  3. ],  
       [ 1.5,  2. ,  3. ],  
       [ 4. ,  5. ,  6. ]])  
>>> np.r_[e,f]  
>>> np.hstack((e,f))  
array([[ 7.,  7.,  1.,  0.],  
       [ 7.,  7.,  0.,  1.]])  
>>> np.column_stack((a,d))  
array([[ 1, 10],  
       [ 2, 15],  
       [ 3, 20]])  
>>> np.c_[a,d]
```

Concatenate arrays

Stack arrays vertically (row-wise)

Stack arrays vertically (row-wise)

Stack arrays horizontally (column-wise)

Create stacked column-wise arrays

Create stacked column-wise arrays

#### Splitting Arrays

```
>>> np.hsplit(a,3)  
[array([1]),array([2]),array([3])]  
>>> np.vsplit(c,2)  
[array([[ 1.5,  2. ,  1. ],  
       [ 4. ,  5. ,  6. ]]),  
 array([[ 3.,  2.,  3.],  
       [ 4. ,  5. ,  6.]])]
```

Split the array horizontally at the 3rd index

Split the array vertically at the 2nd index

DataCamp

Learn Python for Data Science Interactively

