NUMPY
Cheat Sheet

machinelearningplus.com

" Learn Applied Data Science "

NUMPY

NumPy (Numerical Python) is an open source python library used in almost every field of science and engineering.
It is sort of the standard for working with numerical data in python. NumPy API is used by many other popular packages such as Pandas, Matplotlib, SciPy, scikit learn etc.

It provides high performance multidimensional array objects and tools for processing these arrays and efficiently perform mathematical operations. NumPy's arrays are more compact than Python lists a list of lists as you'd describe.

IMPORT CONVENTION

```
>>> import numpy as np
```
Importing numpy as `np` is the commonly used convention.

ND ARRAY – ARR

An array object represents a multidimensional, homogeneous array of fixed size items.
The below value of arr, arr1 and arr2 will be used for the examples:

```
>>> arr  = np.array([(1.5, 2.5), (4, 5)],
           dtype=float)
>>> arr1 = np.array([5, 6, 3])
>>> arr2 = np.array([4, 5, 2])
```

IMPORT/EXPORT

Import files as numpy arrays and export the numpy arrays as text, csv files etc

IMPORT

`>>> np.loadtxt('file.txt')`	Load data when no value is missing
`>>> np.genfromtxt ('file.csv',delimiter=',')`	When file contains missing data
`>>> np.load('my_array.npy')`	Load arrays from .npy, .npz or pickled files.

EXPORT

`>>> np.savetxt('file.txt', arr,delimiter=' ')`	Write to a text file
`>>> np.savetxt ('file.csv',arr,delimi ter=',')`	Write to a CSV file
`>>> np.savez('array.npz', a, b)`	Save several arrays

DATA TYPE

`>>> np.int8`	8 bit integer type (8, 16, 32, 64 based on values)
`>>> np.float32`	32 bit float type (32, 64 based on values)
`>>> np.complex`	Complex number
`>>> np.bool`	Boolean type
`>>> np.object`	Python object type
`>>> np.str`	Fixed length string type
`>>> np.unicode`	Fixed length unicode type

CREATE ARRAYS

`>>> np.array([1,2,3])`	Create 1 D array
`>>> np.array([(1,2,3), (4.5,5,6)], dtype = float)`	Create 2 D array
`>>> np.array([[[(1,2,3), (4.5,5,6)], [[(2,3,4), (5,6,7)]], dtype = float)`	Create 3 D array

```
{
array([[[1. , 2. , 3. ],
        [4.5, 5. , 6. ]],
       [[2. , 3. , 4. ],
        [5. , 6. , 7. ]]])
}
```

INITIALIZING ARRAYS: Use below commands to create

`>>> np.zeros(5)`	1 D array with zero values
`>>> np.zeros((3,4))`	2 D array with zero values
`>>> np.ones((3,4))`	2 D array with value as 1

```
{
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
}
```

`>>> np.eye(5)`	5*5 identity array
`>>> np.full((3,3), 4)`	3*3 array with all values as 4
`>>> np.empty((3,2))`	Empty 3*2 array

RANDOM NUMBER ARRAYS

`>>> np.random.randint (1,10, (3,2))`	Return random integers from low (inclusive) to high (exclusive).

```
{
array([[9, 2],
       [4, 6],
       [5, 8]])
}
```

`>>> np.random.random ((3,3))`	Float Values between 0 and 1
`>>> np.random.seed (some_number)`	Same result every time with np.random

SEQUENCE ARRAYS

`>>> np.arange(5,25,6)`	Array of evenly spaced values starting from 5 to 25 with space as 6

```
{
array([ 5, 11, 17, 23])
}
```

`>>> np.linspace(5,25,6)`	Array of evenly spaced values starting from 5 to 25 with 6 equally spaced values

```
{
array([ 5., 9., 13., 17.,
        21., 25.])
}
```

INSPECT THE ARRAYS

`>>> arr.size`	Number of elements in the array
`>>> len(arr)`	Length of the array
`>>> arr.shape`	Dimensions of the array (rows, columns)
`>>> arr.dtype`	Data type of elements in the array arr
`>>> arr.astype(dtype)`	Change data type to data type dtype
`>>> arr.tolist()`	Convert the array to list
`>>> arr.nbytes`	Number of bytes occupied by array
`>>> np.info(np.zeros)`	Show documentation for np.zeros

MEMORY USAGE OF DATATYPES

```
>>> np.int8(10).nbytes:
1
>>> np.float32(10).nbytes:
4
>>> np.array('1', dtype='object').nbytes:
8
>>> np.array(1, dtype='bool').nbytes:
1
>>> np.array('1', dtype=str).nbytes:
4
```

ADD/ REMOVE ELEMENTS FROM ARRAY

`>>> np.append(arr, value)`	Append to the end of array
`>>> np.insert(arr,2,value)`	Insert to the array before index 2
`` `np.insert(arr,2,10) ` ``	

```
{
array([ 1., 2. , 10. ,
        4.5. , 5.)
}
```

`>>> np.delete(arr,2,axis=0)`	Delete row on index 2 of the array

ARRAY OPERATIONS – COPY AND SORT

`>>> arr.view()`	Create a new view of array
`>>> np.copy(arr)`	Create a copy
`>>> arr.copy()`	Create a deep copy
`>>> arr.sort()`	Sort an array

ARRAY MANIPULATION

`>>> arr.T`	Transpose the array

```
{
array([[1. , 4.5],
       [2. , 5. ],
}
```

`>>> arr.flatten()`	Flattens array arr to 1 D array

```
{
array([1. , 2. , 3. ,
       4.5])
}
```

`>>> arr.ravel()`	Flattened view. No Copying
`>>> arr.reshape(3,2)`	Reshapes array
`>>> arr.resize((3,2))`	Changes the shape in place

```
{
>>> arr = np.array
([(1.5, 2.5, 3.5),
(4, 5, 6)], dtype=float)

>>> arr.resize((3, 2))

    array([[1.5, 2.5],
           [3.5, 4. ],
           [5. , 6. ]])

}
```

`>>> np.hsplit(arr,2)`	Horizontally split at index 2

```
{

    [array([[1.5],
            [4. ]]),
     array([[2.5],
            [5. ]])]

}
```

`>>> np.vsplit(arr,2)`	Vertically split at index 2

```
{
    [array([[1.5, 2.5]]),
     array([[4., 5.]])]

}
```

`>>> np.concatenate ((arr1,arr2),axis=0)`	Concatenate arrays
`>>> np.vstack((arr1, arr2))`	Stack arrays in vertically

```
{
    array([[5, 6, 3],
           [4, 5, 2]])
}
```

`>>> np.hstack ((arr1,arr2))`	Stack arrays in horizontally
`>>> np.column_stack ((arr1,arr2))`	Stack 1-D arrays as columns into a 2-D array.

```
{
array([[5, 4],
       [6, 5],
       [3, 2]])
}
```

INDEX/ SUBSET/ SLICE ARRAYS

`>>> arr[5]`	Select item at the 5th index
`>>> arr[4,5]`	Select item at row 4 and column 5
`>>> arr[0:5]`	Select items between index 0 and 5

```
{
array([[[4.5, 5. , 6. ],
        [1. , 2. , 3. ]]])
}
```

`>>> arr[0:5,2]`	Select row 0 to 5 in column 2
`>>> arr[ : : 1]`	Reverse the array

```
{
array([[[2. , 3. , 4. ],
        [5. , 6. , 7. ]],
       [[4.5, 5. , 6. ],
        [1. , 2. , 3. ]]])
}
```

`>>> arr <5`	True in locations satisfying condition

```
{
array([[[ True,False,False],
        [ True,True,True]],
       [[ True,True,True],
        [False,False,
         False]]])
}
```

`>>> arr[arr <5]`	Select all elements less than 5

MATHEMATICAL OPERATION WITH ARRAY

ARITHMETIC OPERATIONS

`>>> arr1 - arr2`	Subtract arrays
`>>> np.subtract (arr1, arr2)`	Subtract arrays
`>>> arr1 + arr2`	Add arrays
`>>> np.add(arr1, arr2)`	Add arrays
`>>> arr1 / arr2`	Divide arrays

```
{
array([1.25, 1.2, 1.5])
}
```

`>>> np.divide(a,b)`	Divide arrays
`>>> arr1 * arr2`	Multiply arrays
`>>> np.multiply(a,b)`	Multiply arrays
`>>> arr1.dot(arr2)`	Dot product of arrays
`>>> np.power(arr1,arr2)`	Power value of array arr1 to arr2 element wise

```
{
array([625, 7776, 9])
}
```

STATISTICAL OPERATIONS

`>>> np.mean(arr)`	Mean
`>>> arr.sum()`	Sum
`>>> arr.cumsum()`	Cumulative sum

```
{
array([1. , 3. , 6. ,
       10.5, 15.5, 21.5])
}
```

`>>> arr.min()`	Minimum value
`>>> arr.max()`	Maximum value
`>>> np.std(arr)`	Standard deviation
`>>> np.var(arr)`	Variance
`>>> arr.corrcoef()`	Pearson product-moment correlation coefficients

```
{
array([[1. , 0.98],
       [0.98, 1. ]])
}
```

101 NumPy
Exercises for Data Analysis

https://www.machinelearningplus.com/python/101-numpy-exercises-python/