

NumPy Cheat Sheet

[NumPy](#) is the fundamental package for scientific computing with Python.

This cheat sheet acts as a intro to Python for data science. ***Contact me [here](#) for typos or suggestions, and - of course - fork and tune it to your taste!***

Index

1. [Basics](#)
 - [Placeholders](#)
 - [Examples](#)
2. [Arrays](#)
 - [Properties](#)
 - [Copying/Sorting](#)
 - [Examples](#)
 - [Array Manipulation](#)
 - [Adding/Removing Elements](#)
 - [Combining Arrays](#)
 - [Splitting Arrays](#)
3. [Mathematics](#)
 - [Arithmetic Operations](#)
 - [Examples](#)
 - [Comparison](#)
 - [Examples](#)
 - [Basic Statistics](#)
 - [More](#)
4. [Slicing and Subsetting](#)
 - [Examples](#)

- 5. [Tricks](#)
- 6. [Credits](#)

Basics

One of the most commonly used functions of NumPy are *NumPy arrays*: The essential difference between *lists* and *NumPy arrays* is functionality and speed. *lists* give you basic operation, but *NumPy* adds FFTs, convolutions, fast searching, basic statistics, linear algebra, histograms, etc.

The most important difference for data science is the ability to do **element-wise calculations** with *NumPy arrays*.

axis 0 always refers to row

axis 1 always refers to column

Operator	Description	Documentation
<code>np.array([1,2,3])</code>	1d array	link
<code>np.array([(1,2,3), (4,5,6)])</code>	2d array	see above
<code>np.arange(start, stop, step)</code>	range array	link

Placeholders

Operators	Description	Documentation
<code>np.linspace(0,2,9)</code>	Add evenly spaced values btw intervall to array of length	link

<code>np.zeros((1,2))</code>	Create and array filled with zeros	link
<code>np.ones((1,2))</code>	Creates an array filled with ones	link
<code>np.random.random((5,5))</code>	Creates random array	link
<code>np.empty((2,2))</code>	Creates an empty array	link

Examples

```
# 1 dimensional
x = np.array([1,2,3])
# 2 dimensional
y = np.array([(1,2,3),(4,5,6)])

x = np.arange(3)
>>> array([0, 1, 2])

y = np.arange(3.0)
>>> array([ 0.,  1.,  2.])

x = np.arange(3,7)
>>> array([3, 4, 5, 6])

y = np.arange(3,7,2)
>>> array([3, 5])
```

Array

Array Properties

Syntax	Description	Documentation
<code>array.shape</code>	Dimensions (Rows,Columns)	link
<code>len(array)</code>	Length of Array	link
<code>array.ndim</code>	Number of Array Dimensions	link
<code>array.size</code>	Number of Array Elements	link
<code>array.dtype</code>	Data Type	link
<code>array.astype(type)</code>	Converts to Data Type	link
<code>type(array)</code>	Type of Array	link

Copying/Sorting

Operators	Descriptions	Documentation
<code>np.copy(array)</code>	Creates copy of array	link
<code>other = array.copy()</code>	Creates deep copy of array	see above
<code>array.sort()</code>	Sorts an array	link
<code>array.sort(axis=0)</code>	Sorts axis of array	see above

Examples

```
# Sort sorts in ascending order
y = np.array([10, 9, 8, 7, 6, 5, 4, 3, 2, 1])
y.sort()
```

```
print(y)
>>> [ 1  2  3  4  5  6  7  8  9 10]
```

Array Manipulation Routines

Adding or Removing Elements

Operator	Description	Documentation
<code>np.append(a,b)</code>	Append items to array	link
<code>np.insert(array, 1, 2, axis)</code>	Insert items into array at axis 0 or 1	link
<code>array.resize((2,4))</code>	Resize array to shape(2,4)	link
<code>np.delete(array,1,axis)</code>	Deletes items from array	link

Combining Arrays

Operator	Description	Documentation
<code>np.concatenate((a,b),axis=0)</code>	Concatenates 2 arrays, adds to end	link
<code>np.vstack((a,b))</code>	Stack array row-wise	link
<code>np.hstack((a,b))</code>	Stack array column wise	link

Splitting Arrays

Operator	Description	Documentation
<code>numpy.split()</code>		link
<code>np.array_split(array, 3)</code>	Split an array in sub-arrays of (nearly) identical size	link
<code>numpy.hsplit(array, 3)</code>	Split the array horizontally at 3rd index	link

More

Operator	Description	Documentation
<code>other = ndarray.flatten()</code>	Flattens a 2d array to 1d	link
<code>array = np.transpose(other) array.T</code>	Transpose array	link

Mathematics

Operations

Operator	Description	Documentation
<code>np.add(x, y)</code>	Addition	link
<code>np.subtract(x, y)</code>	Subtraction	link

<code>np.divide(x,y)</code>	Division	link
<code>np.multiply(x,y)</code>	Multiplication	link
<code>np.sqrt(x)</code>	Square Root	link
<code>np.sin(x)</code>	Element-wise sine	link
<code>np.cos(x)</code>	Element-wise cosine	link
<code>np.log(x)</code>	Element-wise natural log	link
<code>np.dot(x,y)</code>	Dot product	link

Remember: NumPy array operations work element-wise.

Example

```
# If a 1d array is added to a 2d array (or the other way), N
# chooses the array with smaller dimension and adds it to the
# with bigger dimension
a = np.array([1, 2, 3])
b = np.array([(1, 2, 3), (4, 5, 6)])
print(np.add(a, b))
>>> [[2 4 6]
      [5 7 9]]
```

Comparison

Operator	Description	Documentation
<code>==</code>	Equal	link
<code>!=</code>	Not equal	link
<code><</code>	Smaller than	link
<code>></code>	Greater than	link

<code><=</code>	Smaller than or equal	link
<code>>=</code>	Greater than or equal	link
<code>np.array_equal(x,y)</code>	Array-wise comparison	link

Example

```
# Using comparison operators will create boolean NumPy array.
z = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
c = z < 6
print(c)
>>> [ True  True  True  True  True False False False False False]
```

Basic Statistics

Operator	Description	Documentation
<code>array.mean()</code> <code>np.mean(array)</code>	Mean	link
<code>np.median(array)</code>	Median	link
<code>array.corrcoef()</code>	Correlation Coefficient	link
<code>array.std(array)</code>	Standard Deviation	link

More

Operator	Description	Documentation
<code>array.sum()</code>	Array-wise sum	link

<code>array.min()</code>	Array-wise minimum value	link
<code>array.max(axis=0)</code>	Maximum value of specified axis	
<code>array.cumsum(axis=0)</code>	Cumulative sum of specified axis	link

Slicing and Subsetting

Operator	Description	Documentation
<code>array[i]</code>	1d array at index i	link
<code>array[i,j]</code>	2d array at index[i][j]	see above
<code>array[i<2]</code>	Boolean Indexing	see above
<code>array[0:2]</code>	Select items of index 0 and 1	see above
<code>array[0:2,1]</code>	Select items of rows 0 and 1 at column 1	see above
<code>array[:1]</code>	Select items of row 0 (equals <code>array[0:1, :]</code>)	see above
<code>array[1,...]</code>	equals <code>array[1,:::]</code>	see above
<code>array[: :-1]</code>	Reverses <code>array</code>	see above

Examples

```
b = np.array([(1, 2, 3), (4, 5, 6)])
```

*# The index *before* the comma refers to *rows*,*

```
# the index after the comma refers to columns
print(b[0:1, 2])
>>> [3]

print(b[:len(b), 2])
>>> [3 6]

print(b[0, :])
>>> [1 2 3]

print(b[0, 2:])
>>> [3]

print(b[:, 0])
>>> [1 4]
```

Tricks

This is a growing list of examples. Know a good trick? Let me know [here](#) or fork it and create a pull request.

boolean indexing

```
# Index trick when working with two np-arrays
a = np.array([1,2,3,6,1,4,1])
b = np.array([5,6,7,8,3,1,2])

# Only saves a at index where b == 1
other_a = a[b == '1']
#Saves every spot in a except at index where b != 1
other_other_a = a[b != '1']
```

```
x = np.array([4,6,8,1,2,6,9])
y = x > 5
print(x[y])
>>> [6 8 6 9]

# Even shorter
x = np.array([1, 2, 3, 4, 4, 35, 212, 5, 5, 6])
print(x[x < 5])
>>> [1 2 3 4 4]
```

Credits

[Datacamp](#), [Quandl](#) & [Official docs](#)