Numpy Cheat Sheet

Numerical Python

NumPy provides a high-performance multi-dimensional array object (ndarray), and tools for working with these arrays.

Import convention

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

Creating Numpy Arrays

Creating From Lists

```
>>> np.array([3,4], dtype=float)
```
float array from list

Creating Constant Numpy Arrays

`>>> np.zeros((3,4))`	array of zeros
`>>> np.ones((2,3,4))`	array of ones
`>>> e=np.full((3,4),7)`	constant array

Creating Sequential Arrays

Arrays of evenly spaced values	
`np.arange(10,25,5)`	step value
`l=np.linspace(0,3,4)`	# samples
`np.empty((3,2))`	empty array
`f=np.random.random((2,2))`	random array

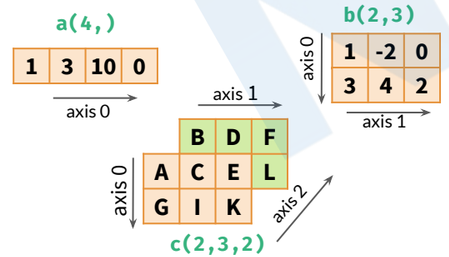
1D, 2D, 3D Numpy Arrays

```
>>> a = np.array([1, 3, 10, 0])

>>> b = np.array([[1, -2, 0],
                  [3, 4, 2]])

>>> c = np.array([[['A', 'B'],
                   ['C', 'D'],
                   ['E', 'F']],

                  [['G', 'H'],
                   ['I', 'J'],
                   ['K', 'L']]])
```



Other Array Construction Methods

`>>> np.eye(2)`	2 x 2 identity matrix
`>>> np.diag((1,3,5))`	diagonal matrix
`>>> d = a.copy()`	Copy of array

nD Array Properties

`>>> a.size`	Number of array elements
`>>> a.ndim`	Number of array dimensions
`>>> a.shape`	Array dimensions
`>>> a.dtype`	Data type of array elements

Indexing and Slicing

1D Arrays

Indexing

1 3 10 0	`a[2]`
1 3 10 0	`a[-3]`
1 3 10 0	`a[0:2]`

Slicing

1 3 10 0	`a[0:3:2]`
1 5 5 5	`d[1:4]=5`

2D Arrays

1 -2 0 3 4 2	`b[1,2]`
1 -2 0 3 4 2	`b[0]`
1 -2 0 3 4 2	`b[:,0]`

`>>> b[0:2]`	Slicing rows.
`>>> b[0:2,1:3]`	Slicing rows and columns.
`>>> b[:,0:2]=4`	Modifying slices.

Advanced Indexing & Slicing

Fancy Indexing

```
>>> i = np.array([0,2])
>>> a[i]
    [1,10]

>>> b[[0,2], [1,2]]
```

Boolean Masking

```
>>>m=np.array([True,False,False,True])
>>> a[m]
    [1,0]
```

Array Operations and Ufuncs

Operations on Numpy Arrays

Operator Overloading

`>>> np.add(a,l)` is same as `a + l`

Vectorized Operations with NumPy

Arithmetic Operations

`>>> np.add(a,l)` `array([1,4,12,3])`	`a + l`
`>>> np.subtract(a,l)`	`a - l`
`>>> np.multiply(a,l)`	`a * l`
`>>> np.divide(a,l)`	`a / l`
`>>> np.floor_divide(a,l)`	`a // l`
`>>> np.mod(a,l)`	`a % l`
`>>> np.power(a,l)`	`a ** l`

Relational Operations

`>>> np.greater(a,l)`	`a > l`
`>>> np.greater_equal(a,l)`	`a ⩾ l`
`>>> np.less(a,l)`	`a < l`
`>>> np.less_equal(a,l)`	`a ⩽ l`
`>>> np.equal(a,l)`	`a == l`
`>>> np.not_equal(a,l)`	`a ≠ l`

Bitwise Operations

`>>> np.bitwise_and(a,l)`	`a & l`
`>>> np.bitwise_or(a,l)`	`a | l`
`>>> np.bitwise_xor(a,l)`	`a ^ l`
`>>> np.invert(a)`	`~a`
`>>> np.left_shift(a,l)`	`a << l`
`>>> np.right_shift(a,l)`	`a >> l`

Linear Algebra

`>>> np.matmul(b,e)`	Matrix multiplication
`>>> np.transpose(b)`	Transpose
`>>> np.linalg.det(f)`	Determinant
`>>> np.linalg.inv(f)`	Inverse

Transpose

b

1	-2	0
3	4	2

→ b.T

1	3
-2	4
0	2

Broadcasting & Masking

Two arrays are said to be compatible in a dimension if

1. they have the same size in that dimension, or
2. one of the arrays has size 1 in that dimension

```
>>> g = np.array([[1, -2, 0, 1],
                  [3, 4, 2, 0]])
>>> a + g
```

1	3	10	0
1	3	10	0

+

1	-2	0	1
3	4	2	0

=

2	1	10	1
4	7	12	0

Masking

```
>>> mask = a > 3
>>> a[mask]
```

1	3	10	0

> 3

False	False	True	False

→ | 1 | 3 | 10 | 0 | → 10

Useful Methods in Numpy

Sum of all elements

```
>>> np.sum(a)
>>> a.sum()
>>> np.add.reduce()
```

Maximum element

```
>>> b.max()
>>>np.max(b,axis=0)
```

`>>> np.argmax(a)`	Index of max element
`>>> np.sort(a)`	Sorted copy of array
`>>> a.sort()`	Sorts an array in-place
`>>> np.argsort(a)`	Indices that sort array.

More Numpy Methods

`>>> np.unique(a)`	Unique sorted elements
`>>> np.where(a>5, a*2, a*a)`	If true a*2, else a*a
`>>> np.any(a)`	Any of the elements evaluates to True ?
`>>> np.all(a)`	All the elements evaluate to True ?
`>>> np.nan`	NaN constant
`>>> np.inf`	Positive Infinity

Array Manipulations

Changing shape of an array

```
>>> np.reshape(c,(2,2,3))
```

B	D	F
A	C	E	L
G	I	K

→

C	F
B	E	L
A	D	K
G	J

```
>>> np.ravel(b)
```
Contiguous flattened array

```
>>> b.flatten()
```
Flatten array to one dimension

```
>>> a[np.newaxis,:]
```
Increase dimension by one

```
>>> np.squeeze(a)
```
Removes single- dimensional entries

Changing the axes of an array

```
>>> np.moveaxis(c, [0,1], [1, 2])
```
Moves axes from old to new positions

```
>>> np.swapaxes(c, 1, 2)
```
Interchanges axis-1 and axis-2

B	D	F
A	C	E	L
G	I	K

axis 1, axis 2 →

E	F
C	D	L
A	B	J
G	H

axis 1, axis 2

Splitting an array

`>>> np.split(b, 2, axis=0)`	Splits b into 2 equal arrays
`>>> np.hsplit(b,[2])`	Splits at 2nd column
`>>> np.vsplit(b,[1])`	Splits at 1st row

Joining arrays

`>>> np.concatenate( (a,l), axis=0)`	Joins sequence of arrays along axis
`>>>np.hstack((a,l))`	Stacks column-wise
`>>>np.vstack((a,l))`	Stacks row-wise
`>>> np.stack([a, l], axis = 0)`	Joins along new axis.

hstack

1.	3.	10.	0.	0.	1.	2.	3.

vstack

1.	3.	10.	0.
0.	1.	2.	3.

Tiling Arrays

`>>> np.repeat(a,2)`	Repeats elements

Adding and Removing elements

`>>> np.delete(b,1,0)`	Deletes elements
`>>> np.insert(b,1, [4,4],axis=1)`	Inserts elements
`>>> np.append(a,l, axis=0)`	Appends elements

Padding

```
>>> np.pad(b, (1,2), 'constant', constant_values=(-1,-2))
```
Pads with a constant value.

```
>>> np.pad(b, (1, 1), 'edge')
```
Pads with the edge values of the array.

Edge padding

1	1	-2	0	0
1	1	-2	0	0
3	3	4	2	2
3	3	4	2	2