

# Numpy Cheat Sheet

```
In [1]: import numpy as np
```

```
In [2]: %precision 2
ipython_plain = get_ipython().display_formatter.formatters['text/plain']
ipython_plain.for_type(np.float64, ipython_plain.lookup_by_type(float));
```

## Create

```
In [3]: v3 = np.array([1, 2, 3])
v3
```

```
Out[3]: array([1, 2, 3])
```

```
In [4]: m33 = np.array([[1, 2, 3],
                        [4, 5, 6],
                        [7, 8, 9]])
m33
```

```
Out[4]: array([[1, 2, 3],
               [4, 5, 6],
               [7, 8, 9]])
```

```
In [5]: from scipy.sparse import csr_matrix
sparse = csr_matrix(np.array([[0, 0, 1]]))
(sparse + sparse)[0, 2]
```

```
Out[5]: 2
```

```
In [6]: v2 = np.array([1, 2])
v2a = v2
v2a[0] = 3
v2
# NOTE: Direct assignment results in a reference!
```

```
Out[6]: array([3, 2])
```

```
In [7]: v2b = v2 + 1
v2b[0] = 5
v2b, v2
```

```
Out[7]: (array([5, 3]), array([3, 2]))
```

```
In [8]: v2c = v2.copy()
v2c[0] = 0
v2
```

```
Out[8]: array([3, 2])
```

## Stacking

```
In [9]: np.column_stack((v3, v3))
```

```
Out[9]: array([[1, 1],
               [2, 2],
               [3, 3]])
```

```
In [10]: np.hstack((m33, m33))
```

```
Out[10]: array([[1, 2, 3, 1, 2, 3],
                [4, 5, 6, 4, 5, 6],
                [7, 8, 9, 7, 8, 9]])
```

```
In [11]: np.vstack((v3, v3))
```

```
Out[11]: array([[1, 2, 3],
                [1, 2, 3]])
```

## Describe

```
In [12]: m33.shape
```

```
Out[12]: (3, 3)
```

```
In [13]: m33.ndim
```

```
Out[13]: 2
```

```
In [14]: m33.size
```

```
Out[14]: 9
```

## Select

```
In [15]: v3[2]
```

```
Out[15]: 3
```

```
In [16]: m33[2, 2]
```

```
Out[16]: 9
```

```
In [17]: print( m33[:, 2] )
print( m33[2, :] )
# NOTE: Returns 1-D *Vectors*

[3 6 9]
[7 8 9]
```

```
In [18]: print( m33[:, 2:3] )
print( m33[2:3, :] )
# NOTE: Returns 2-D *Matrixes*
```

```
[[3]
 [6]
 [9]]
[[7 8 9]]
```

```
In [19]: m33[1]
# NOTE: Returns 1-D *Vector*
```

```
Out[19]: array([4, 5, 6])
```

```
In [20]: m33[[1]]
# NOTE: Returns 2-D *Matrix*
```

```
Out[20]: array([[4, 5, 6]])
```

## np.vectorize()

```
In [21]: vect_fn = np.vectorize(lambda x: 1 if x < 3 else 0)
vect_fn(m33)
```

```
Out[21]: array([[1, 1, 0],
                [0, 0, 0],
                [0, 0, 0]])
```

## Aggregate

```
In [22]: m33
```

```
Out[22]: array([[1, 2, 3],
                [4, 5, 6],
                [7, 8, 9]])
```

```
In [23]: print( m33.min() )
print( m33.max() )
```

```
1
9
```

```
In [24]: print( m33.max(axis=0) )
print( m33.max(axis=1) )
# NOTE: Returns 1-D *Vector*
```

```
[7 8 9]
[3 6 9]
```

```
In [25]: m33.mean(), m33.std(), m33.var()
```

```
Out[25]: (5.00, 2.58, 6.67)
```

## Reshape

```
In [26]: m23 = np.array([[1, 2, 3],
                        [4, 5, 6]])
m23
```

```
Out[26]: array([[1, 2, 3],
               [4, 5, 6]])
```

```
In [27]: m23.reshape(3, 2)
```

```
Out[27]: array([[1, 2],
               [3, 4],
               [5, 6]])
```

```
In [28]: m23.reshape(1, -1)
```

```
Out[28]: array([[1, 2, 3, 4, 5, 6]])
```

```
In [29]: m23.ravel()
# m23.reshape(-1)
# m23.flatten()
# NOTE: Returns 1-D *Vector*
```

```
Out[29]: array([1, 2, 3, 4, 5, 6])
```

```
In [30]: m23.T
```

```
Out[30]: array([[1, 4],
               [2, 5],
               [3, 6]])
```

```
In [31]: v3, v3.T
# NOTE: Vectors cannot be Transposed!
```

```
Out[31]: (array([1, 2, 3]), array([1, 2, 3]))
```

```
In [32]: v3.reshape(-1, 1)
# convert horz "vector" to vert "vector"
```

```
Out[32]: array([[1],
               [2],
               [3]])
```

```
In [33]: v3[None]
# wrap/encapsulate inside an additional outer dimension
```

```
Out[33]: array([[1, 2, 3]])
```

```
In [34]: np.array([[1], [2]]).squeeze()
# unwrap innermost dimension
```

```
Out[34]: array([1, 2])
```

## Linear Algebra

```
In [35]: np.triu(m33), np.tril(m33)
```

```
Out[35]: (array([[1, 2, 3],
                [0, 5, 6],
                [0, 0, 9]]),
          array([[1, 0, 0],
                [4, 5, 0],
                [7, 8, 9]]))
```

```
In [36]: v3
```

```
Out[36]: array([1, 2, 3])
```

```
In [37]: v3 @ v3
# vector dot product
```

```
Out[37]: 14
```

```
In [38]: m33 @ m33
```

```
Out[38]: array([[ 30,  36,  42],
                [ 66,  81,  96],
                [102, 126, 150]])
```

```
In [39]: m33 * m33
```

```
Out[39]: array([[ 1,  4,  9],
                [16, 25, 36],
                [49, 64, 81]])
```

```
In [40]: np.linalg.inv(np.array([[1,0], [0,2]]))
```

```
Out[40]: array([[1. , 0. ],
                [0. , 0.5]])
```

## Random

```
In [41]: np.random.seed(0)
```

```
In [42]: np.random.random(3)
# between 0.0 and 1.0
```

```
Out[42]: array([0.55, 0.72, 0.6 ])
```

```
In [43]: [np.random.randint(3) for _ in range(10)]
# (<max-excl>)
```

```
Out[43]: [1, 1, 2, 0, 2, 0, 0, 0, 2, 1]
```

```
In [44]: [np.random.randint(1, 4) for _ in range(10)]
# (<min-incl>, <max-excl>)
```

```
Out[44]: [3, 3, 1, 2, 2, 2, 2, 1, 2, 1]
```

```
In [45]: np.random.uniform(1, 5, 15)
# (<min>, <max>, <size>)
```

```
Out[45]: array([1.08, 4.33, 4.11, 4.48, 4.91, 4.2 , 2.85, 4.12, 1.47, 3.56, 1.57,
                4.78, 3.09, 2.66, 2.06])
```

```
In [46]: np.random.normal(5, 1, 15)
# (<mean>, <stddev>, <size>)

Out[46]: array([4.76, 6.51, 4.67, 5.05, 6.46, 6.54, 5.57, 5.15, 3.92, 6.4 , 6.79,
4.43, 5.18, 4.54, 3.91])

In [47]: np.random.choice(v3, size=5, replace=True)
# 'replace': with/without replacement

Out[47]: array([1, 3, 2, 1, 2])
```

## Filter

```
In [48]: m23

Out[48]: array([[1, 2, 3],
[4, 5, 6]])

In [49]: m23 < 4

Out[49]: array([[ True,  True,  True],
[False, False, False]])

In [50]: (m23 < 2) | (m23 > 4)

Out[50]: array([[ True, False, False],
[False,  True,  True]])

In [51]: m23[(m23 < 2) | (m23 > 4)]

Out[51]: array([1, 5, 6])

In [52]: np.where(m23 < 4, 1, -1)

Out[52]: array([[ 1,  1,  1],
[-1, -1, -1]])
```

## Misc

```
In [53]: v10 = np.linspace(0, 10, 10)
v10

Out[53]: array([ 0. ,  1.11,  2.22,  3.33,  4.44,  5.56,  6.67,  7.78,  8.89,
10.  ])
```

```
In [54]: np.percentile(v10, [25, 50, 100])
# Returns value(s) corresponding to given percentiles

Out[54]: array([ 2.5,  5. , 10. ])
```

```
In [55]: np.where((v10 < 3) | (v10 > 6))

Out[55]: (array([0, 1, 2, 6, 7, 8, 9]),)
```

```
In [56]: v10 < 3
```

```
Out[56]: array([ True,  True,  True, False, False, False, False, False, False,
        False])
```

```
In [57]: mu = np.array([[1, 2, 1], [1, 3, 1], [1, 2, 1]])
print(mu, np.unique(mu), np.unique(mu, axis=0), np.unique(mu, axis=1), sep='\n\')
```

```
[[1 2 1]
 [1 3 1]
 [1 2 1]]
```

```
[1 2 3]
```

```
[[1 2 1]
 [1 3 1]]
```

```
[[1 2]
 [1 3]
 [1 2]]
```

```
In [ ]:
```