NumPy Cheat Sheet

NumPy is a fast, powerful library for numerical computing with fixed-type multidimensional arrays. It enables high-performance operations and forms the core of tools like Pandas and Matplotlib.

Installation

```
pip install numpy
```

Import

```
import numpy as np
```

Array Creation

```
np.array([1, 2, 3])                          # 1D array
np.array([[1, 2], [3, 4]])                   # 2D array
np.zeros(64)                                 # 1D array of 64 zeros
np.zeros((2, 3))                             # 2x3 zero matrix
np.ones((2, 3))                              # 2x3 ones matrix
np.empty((2, 3))                             # 2x3 uninitialized array
np.arange(0, 10, 2)                          # Even numbers from 0 to 8
np.linspace(0, 1, 5)                         # 5 equally spaced values from 0 to 1
        (inclusive)
np.identity(3)                               # 3x3 identity matrix
np.random.random((3, 3))                     # 3x3 matrix with random floats in [0, 1)
np.pad(np.ones((2, 2)), 1, constant_values=0)  # Pad 2x2 ones matrix with zeros
```

Array Properties

```
a = np.array([1, 2, 3, 4], dtype='int16')
a.shape             # (4,) - 1D array with 4 elements
a.ndim              # 1 - Number of dimensions
a.size              # 4 - Total number of elements
a.dtype             # int16 - Data type of elements
a.itemsize          # 2 - Bytes per element
```

Reshape & Transpose

```
a.reshape(2, 2)     # Reshape 1D array to 2x2
a.T                 # Transpose of array (works for 2D+)
```

Indexing & Slicing

```
a[1, 2]             # Element at row 1, col 2 (2D array)
a[:, 1]             # All rows, column 1
a[1, :]             # Row 1, all columns
a[1:3, 0:2]         # Subarray (rows 1-2, cols 0-1)
a[::-1]             # Reverse array
np.where(a != 0)    # Indices of non-zero elements
q10.nonzero()       # Equivalent to np.where(q10 != 0)
```

Arithmetic Operations

```
a + b               # Element-wise addition
a - b               # Element-wise subtraction
a * b               # Element-wise multiplication
a / b               # Element-wise division
a // b              # Element-wise floor division
a % b               # Element-wise modulo
a ** 2              # Element-wise power
np.dot(a, b)        # Dot product
np.matmul(a, b)     # Matrix multiplication
```

Statistical Operations

```
a.min()             # Minimum value
a.max()             # Maximum value
a.sum()             # Sum of elements
a.mean()            # Mean of elements
a.std()             # Standard deviation
np.add.reduce(a)    # Sum via reduce
np.max(a, axis=1)   # Row-wise max
np.argmax(a)        # Index of max value
np.argmin(a)        # Index of min value
```

Logical & Comparison

```
a > 5               # Element-wise comparison
np.any(a > 5)       # Any element > 5?
np.all(a > 5)       # All elements > 5?
np.where(a > 0, 1, 0)         # Conditional selection
np.intersect1d(a, b)          # Common elements (1D output)
np.union1d(a, b)              # All unique elements (1D output)
np.allclose(a, b)             # Approximate equality (with tolerance)
np.array_equal(a, b)          # Exact equality
```

Bitwise & Special Operations

```
2 << a >> 2         # Bitwise shift operations
1j * a              # Multiply by imaginary unit (complex numbers)
a // 2              # Floor division
a % 2               # Modulo
np.sqrt(4)          # Square root
np.emath.sqrt(-1)   # Complex sqrt (returns 1j); np.sqrt(-1) returns nan
```

Tiling & Repeating

```
np.tile([1,2], 3)                   # Repeat array: [1 2 1 2 1 2]
np.repeat([1,2], 2)                 # Repeat elements: [1 1 2 2]
np.tile([[1,2],[3,4]], (2,3))       # Tile 2D array 2x vertically and 3x horizontally
```

Array Borders

```
Z = np.ones((5, 5))
Z = np.pad(Z, pad_width=1, mode='constant', constant_values=0)

Z[:, [0, -1]] = 0   # Set first/last columns to 0
Z[[0, -1], :] = 0   # Set first/last rows to 0
```

Normalization

```
(a - np.mean(a)) / np.std(a)    # Normalize array (zero mean, unit std)
```

Data Type Conversion

```
a.astype(np.uint8)              # Convert data type
```

Custom Data Type

```
color = np.dtype([('r',np.ubyte),('g',np.ubyte),('b',np.ubyte),('a',np.ubyte)])
```

Dates with NumPy

```
np.datetime64('today')                              # Today's date
np.arange('2020-01', '2020-02', dtype='datetime64[D]')  # All Jan 2020 dates
```

Mesh Grid

```
x, y = np.meshgrid(np.linspace(0,1,3), np.linspace(0,1,3))
```

Checkerboard Pattern

```
Z = np.zeros((8,8), dtype=int)
Z[1::2, ::2] = 1
Z[::2, 1::2] = 1
```

Matrix Operations

```
np.linalg.det(a)        # Determinant
np.linalg.inv(a)        # Inverse
np.linalg.eig(a)        # Eigenvalues and eigenvectors
```

Fancy Indexing

```
Z[:, [0, -1]] = 0       # Set first/last columns to 0
Z[[0, -1], :] = 0       # Set first/last rows to 0
```

Diagonal Matrix

```
np.diag(1+np.arange(4), k=-1)   # Below main diagonal
```

Unravel Index

```
np.unravel_index(99, (6,7,8))   # Convert flat index to multi-dimensional index
```

Linearly Spaced Vector

```
np.linspace(0,1,12)     # 12 values from 0 to 1
```

Sorting

```
np.sort(a)              # Return sorted array
a.sort()                # In-place sort
```

Stacking Arrays

```
np.vstack([a,b])        # Vertical stack
```

```
np.hstack([a,b])        # Horizontal stack
```

Loop Examples

```
for item in iterable:
    print(item)

# Indexes only
for i in range(len(iterable)):
    print(i, iterable[i])

# Index + item
for i, item in enumerate(iterable):
    print(i, item)

# 2D index + value (for arrays)
for (i, j), val in np.ndenumerate(array):
    print((i, j), val)
```