NumPy Quick Reference

Array creation, math, linear algebra, and more

Array Creation

From Lists

```
import numpy as np
a = np.array([1, 2, 3])        # 1D
b = np.array([[1, 2], [3, 4]])  # 2D
```

Built-in Constructors

```
np.zeros((2, 3))     # 2x3 of zeros
np.ones((3, 3))      # 3x3 of ones
np.eye(4)            # 4x4 identity matrix
np.arange(0, 10, 2)  # [0, 2, 4, 6, 8]
np.linspace(0, 1, 5) # 5 evenly spaced
```

Array Properties

`a.shape`	Dimensions as tuple: `(3, 4)`
`a.ndim`	Number of dimensions
`a.size`	Total number of elements
`a.dtype`	Data type: `float64`, `int32`, etc.

Indexing & Slicing

Basic Indexing

```
a = np.array([[1, 2, 3], [4, 5, 6]])
a[0, 1]    # 2 (row 0, col 1)
a[1]       # [4, 5, 6] (row 1)
a[:, 0]    # [1, 4] (all rows, col 0)
```

Slicing

```
a[0, 1:]   # [2, 3] (row 0, col 1 onward)
a[:, :2]   # first 2 columns
a[::2]     # every other row
```

Boolean Indexing

```
a = np.array([10, 20, 30, 40])
a[a > 15]   # [20, 30, 40]
a[a % 20 == 0]  # [20, 40]
```

Array Operations

Element-wise Operations

```
a = np.array([1, 2, 3])
a + 10    # [11, 12, 13]
a * 2     # [2, 4, 6]
a ** 2    # [1, 4, 9]
a + a     # [2, 4, 6]
```

Comparison

```
a = np.array([1, 2, 3, 4])
a > 2          # [False, False, True, True]
np.where(a > 2, a, 0)  # [0, 0, 3, 4]
```

Aggregation

`a.sum()`	Sum of all elements
`a.mean()`	Arithmetic mean
`a.std()`	Standard deviation
`a.min() / a.max()`	Min / max value
`a.argmin() / a.argmax()`	Index of min / max
`a.cumsum()`	Cumulative sum

Add `axis=0` (columns) or `axis=1` (rows) for per-axis results

Math Functions

Common Functions

`np.sqrt(a)`	Square root of each element
`np.abs(a)`	Absolute value
`np.exp(a)`	e^x for each element
`np.log(a)`	Natural log (ln)
`np.log10(a)`	Base-10 log
`np.sin(a) / np.cos(a)`	Trig functions (radians)
`np.round(a, 2)`	Round to 2 decimals
`np.clip(a, lo, hi)`	Clamp values to [lo, hi]

Linear Algebra

Matrix Operations

```
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
A @ B              # matrix multiply
np.dot(A, B)       # same as A @ B
A.T                # transpose
```

Decomposition & Solving

```
np.linalg.inv(A)       # inverse
np.linalg.det(A)       # determinant
np.linalg.eig(A)       # eigenvalues/vectors
np.linalg.solve(A, b)  # solve Ax = b
```

Random

Random Number Generation

```
rng = np.random.default_rng(42)  # seeded
rng.random((2, 3))     # uniform [0, 1)
rng.integers(1, 10, 5) # 5 ints in [1, 10)
rng.normal(0, 1, 100)  # 100 from N(0,1)
rng.choice([1, 2, 3], size=2)  # sample
```

Legacy API

```
np.random.seed(42)
np.random.rand(3, 3)     # uniform 3x3
np.random.randn(3, 3)    # standard normal
np.random.shuffle(arr)   # in-place shuffle
```

Reshaping

Shape Manipulation

```
a = np.arange(12)
a.reshape(3, 4)    # 3x4 matrix
a.reshape(3, -1)   # infer columns
a.flatten()        # back to 1D (copy)
a.ravel()          # back to 1D (view)
```

Stacking & Splitting

```
np.vstack([a, b])  # stack vertically
np.hstack([a, b])  # stack horizontally
np.concatenate([a, b], axis=0)
np.split(a, 3)     # split into 3 parts
```

Broadcasting

How Broadcasting Works

```
a = np.array([[1, 2, 3],
              [4, 5, 6]])  # shape (2,3)
b = np.array([10, 20, 30]) # shape (3,)
a + b  # b broadcasts to (2,3)
```

Rules

`Rule 1`	Prepend 1s to shorter shape until ranks match
`Rule 2`	Dimensions match if equal or one is 1
`Rule 3`	Size-1 dimensions stretch to match the other

File I/O

NumPy Binary

```
np.save("data.npy", arr)     # single array
arr = np.load("data.npy")
np.savez("data.npz", a=x, b=y) # multiple
d = np.load("data.npz"); d["a"]
```

Text Files

```
np.savetxt("data.csv", arr, delimiter=",")
arr = np.loadtxt("data.csv", delimiter=",")
arr = np.genfromtxt("data.csv", delimiter=",",
                    skip_header=1)
```

Common Patterns

Normalize to [0, 1]

```
normalized = (a - a.min()) / (a.max() - a.min())
```

Euclidean Distance

```
dist = np.sqrt(np.sum((a - b) ** 2))
# or: np.linalg.norm(a - b)
```

Unique Values & Counts

```
vals, counts = np.unique(a, return_counts=True)
dict(zip(vals, counts))
```

Sorting

```
np.sort(a)            # sorted copy
idx = np.argsort(a)   # indices that sort
a[idx]                # apply sort order
```