NumPy Python array computing

CONCEPTS

An ndarray holds one dtype in a fixed-shape block. Whole-array expressions run in C, so replace Python loops with them.

Create array, arange, zeros
↓ shape
Reshape reshape, T, stack
↓ compute
Compute ufuncs, broadcasting
↓ reduce
Reduce sum, mean, argmax
↓ store
Save save, savez, savetxt

Views vs copies
Basic slices are views: writing to a[1:3] changes a. Fancy (list or mask) indexing returns a copy. Use a.copy() to detach.

CREATE

`np.array([[1, 2], [3, 4]])`
Array from nested lists

`np.zeros((<r>, <c>))`
Array of zeros

`np.arange(<start>, <stop>, <step>)`
Evenly spaced by step

`np.linspace(<a>, <b>, <n>)`
n points, both ends included

`np.array(<seq>, dtype=np.float32)`
Choose the dtype

INSPECT

`a.shape`
Tuple of axis lengths

`a.ndim / a.size`
Number of axes / of elements

`a.dtype`
Element type

INDEX AND SLICE

`a[<i>, <j>]`
One element

`a[<a>:<b>:<step>]`
Slice (end excluded)

INDEX AND SLICE (cont.)

`a[:, <j>]`
One column

`a[<i>, :]`
One row

`a[a > <x>]`
Elements matching a mask

`a[[<i>, <j>, <k>]]`
Pick rows by index list

`a[:, np.newaxis]`
Add an axis of length 1

`np.where(<cond>, <x>, <y>)`
x where true, else y

SHAPE AND JOIN

`a.reshape(<r>, -1)`
Reshape, -1 infers one length

`a.T`
Transpose

`np.concatenate([<a>, <b>], axis=0)`
Join along an existing axis

MATH AND BROADCASTING

`a + b, a * 2, a ** 2`
Elementwise arithmetic

`a > <x>, a == b`
Elementwise comparison gives a bool array

`np.isnan(a)`
Mask of NaN values

Broadcasting rules

Shapes	Result
(3, 4) + (4,)	(3, 4)
(3, 1) + (1, 4)	(3, 4)
(3, 4) + (3,)	error

Broadcasting
Shapes line up from the right. Each pair of lengths must be equal or one of them 1, which is stretched. Use a[:, None] to fix a mismatch.

REDUCTIONS

`a.sum() / a.mean()`
Total / average of everything

REDUCTIONS (cont.)

`a.sum(axis=0)`
Collapse rows: one value per column

`a.min() / a.max()`
Smallest / largest

`a.argmax() / a.argmin()`
Index of largest / smallest (flat)

`np.any(<mask>) / np.all(<mask>)`
Any / all True

`np.nansum(a) / np.nanmean(a)`
Reductions that skip NaN

SORT AND SEARCH

`np.sort(a)`
Sorted copy

`np.argsort(a)`
Indices that would sort a

`np.unique(a, return_counts=True)`
Distinct values and their counts

LINEAR ALGEBRA

`a @ b`
Matrix product

`np.linalg.solve(<a>, <b>)`
Solve a x = b

RANDOM

`rng = np.random.default_rng(<seed>)`
Create a seeded generator

`rng.random((<r>, <c>))`
Uniform floats in [0, 1)

`rng.integers(<lo>, <hi>, size=<n>)`
Random ints, hi excluded

`rng.normal(<mu>, <sigma>, size=<n>)`
Normal samples

SAVE AND LOAD

`np.save("<file>.npy", <a>)`
Write one array (binary)

`np.load("<file>.npy")`
Read it back