



NumPy Cheat Sheet

Arrays, indexing, broadcasting, vectorized math, statistics, random numbers, linear algebra, file I/O, examples, and troubleshooting.

ARRAY_EXAMPLE.PY

```
import numpy as np

sales = np.array([
    [120, 150, 180],
    [ 90, 110, 140]
])

totals = sales.sum(axis=1)
print(totals) # [450 340]
```

NUMPY 2.5 | FREE PRINTABLE REFERENCE

Created by CheatSheetSilo Editorial Team
cheatsheetsilo.com

CONTENTS

NumPy Cheat Sheet

A compact, print-friendly companion to the complete online NumPy guide.

01 Quick Reference	02 Install & Import
03 Array Creation	04 Array Attributes
05 Data Types	06 Indexing & Slicing
07 Boolean Masking	08 Reshape & Flatten
09 Transpose & Axes	10 Broadcasting
11 Vectorized Math	12 Universal Functions
13 Aggregations & Statistics	14 Sorting & Searching
15 Joining & Splitting	16 Copies & Views
17 Missing Values	18 Random Numbers
19 Linear Algebra	20 File I/O
21 Practical Examples	22 Performance & Best Practices
23 Errors & Troubleshooting	24 FAQ

How to use this PDF

Start with the Quick Reference, then jump to the topic that matches your task. Code examples use the conventional import alias: `import numpy as np`.

SECTION 01

Quick Reference

The most frequently used NumPy syntax for creating, inspecting, selecting, reshaping, summarizing, and combining arrays.

TASK	SYNTAX	RESULT
Import NumPy	<code>import numpy as np</code>	Use the standard np alias
Create array	<code>np.array([1, 2, 3])</code>	Create a 1D ndarray
Create zeros	<code>np.zeros((2, 3))</code>	Create a zero-filled array
Sequence	<code>np.arange(0, 10, 2)</code>	Values from 0 through 8
Inspect shape	<code>arr.shape</code>	Length of every axis
Filter	<code>arr[arr > 0]</code>	Keep positive values
Reshape	<code>arr.reshape(2, 3)</code>	Reorganize six elements
Column mean	<code>arr.mean(axis=0)</code>	One mean per column
Join	<code>np.concatenate([a, b])</code>	Combine compatible arrays

Copy-ready example

PYTHON

```
import numpy as np

sales = np.array([[120, 150, 180], [90, 110, 140]])
high_sales = sales[sales >= 140]
column_totals = sales.sum(axis=0)
row_averages = sales.mean(axis=1)

print(high_sales)
print(column_totals)
print(row_averages)
```

Axis reminder

axis=0 calculates down rows and returns one result per column. axis=1 calculates across columns and returns one result per row.

SECTION 02

Install & Import

Install NumPy into an isolated environment, import it with the conventional alias, and verify the active version.

TASK	SYNTAX	RESULT
Install with pip	<code>python -m pip install numpy</code>	Install from PyPI
Upgrade	<code>python -m pip install --upgrade numpy</code>	Upgrade active environment
Install with conda	<code>conda install numpy</code>	Install from conda channels
Import	<code>import numpy as np</code>	Use the standard alias

TASK	SYNTAX	RESULT
Version	<code>np.__version__</code>	Show imported version
Build details	<code>np.show_config()</code>	Show linked libraries

Copy-ready example

TERMINAL

```
python -m venv numpy-env

# macOS / Linux
source numpy-env/bin/activate

# Windows PowerShell
.\numpy-env\Scripts\Activate.ps1

python -m pip install --upgrade pip
python -m pip install numpy
```

Environment check

If import fails, confirm that the terminal, editor, and notebook use the same Python environment where NumPy was installed.

SECTION 03

Array Creation

Create arrays from Python sequences or generate initialized arrays, numeric ranges, and common matrix patterns.

TASK	SYNTAX	RESULT
From list	<code>np.array([10, 20, 30])</code>	One-dimensional array
2D array	<code>np.array([[1, 2], [3, 4]])</code>	Two rows and columns
Zeros	<code>np.zeros((2, 3), dtype=int)</code>	Zero-filled integer array
Ones	<code>np.ones((3, 2))</code>	One-filled float array
Fixed value	<code>np.full((2, 4), 7)</code>	Fill every position with 7
Range	<code>np.arange(0, 12, 2)</code>	Defined step size
Even spacing	<code>np.linspace(0, 1, 5)</code>	Defined number of values
Identity	<code>np.eye(3)</code>	Ones on the diagonal

Copy-ready example

PYTHON

```
prices = np.array([19.99, 24.50, 18.75], dtype=np.float64)
sales = np.array([[120, 150, 180], [90, 110, 140]])
zeros = np.zeros_like(sales)
weights = np.full_like(sales, 2)
percentages = np.linspace(0, 1, 5)
```

Choose the sequence function
Use `arange()` when the step size matters and `linspace()` when the number of evenly spaced values matters.

SECTION 04

Array Attributes

Inspect dimensions, shape, data type, element count, and memory use before transforming or calculating with an array.

TASK	SYNTAX	RESULT
Shape	<code>arr.shape</code>	Tuple of axis lengths
Dimensions	<code>arr.ndim</code>	Number of axes
Elements	<code>arr.size</code>	Total element count
Data type	<code>arr.dtype</code>	Element interpretation
Bytes per item	<code>arr.itemsize</code>	Storage per element
Total bytes	<code>arr.nbytes</code>	Element storage total
Memory steps	<code>arr.strides</code>	Bytes advanced per axis
Layout flags	<code>arr.flags</code>	Ownership and contiguity

Copy-ready example

PYTHON

```
sales = np.array([[120, 150, 180], [90, 110, 140]], dtype=np.int64)

print(sales.shape)    # (2, 3)
print(sales.ndim)     # 2
print(sales.size)     # 6
print(sales.dtype)    # int64
print(sales.itemsize) # 8
print(sales.nbytes)   # 48
```

Validate early
Checking shape and dtype early prevents broadcasting errors, incorrect axes, and unexpected type conversion.

SECTION 05

Data Types

Select a dtype that safely represents the required range and precision while avoiding unnecessary memory use.

TASK	SYNTAX	RESULT
Small signed integer	<code>np.int8</code>	-128 through 127
Large signed integer	<code>np.int64</code>	Large whole numbers
Unsigned byte	<code>np.uint8</code>	0 through 255

TASK	SYNTAX	RESULT
Single precision	<code>np.float32</code>	4-byte floating point
Double precision	<code>np.float64</code>	8-byte floating point
Boolean	<code>np.bool_</code>	True or False
Date	<code>datetime64[D]</code>	Daily datetime precision
Duration	<code>timedelta64[D]</code>	Duration in days

Copy-ready example

PYTHON

```
text = np.array(["10", "20", "30"])
integers = text.astype(np.int64)

measurements = np.array([12.8, 17.4, 21.9])
whole = measurements.astype(np.int64) # [12 17 21]

print(np.iinfo(np.int16).min, np.iinfo(np.int16).max)
print(np.finfo(np.float32).eps)
```

Conversion warning

Converting floats to integers discards the fractional part. Round first when a specific rounding rule is needed.

SECTION 06

Indexing & Slicing

Select individual values, ranges, rows, columns, and multidimensional subarrays with zero-based indexing.

TASK	SYNTAX	RESULT
First	<code>arr[0]</code>	Element at index 0
Last	<code>arr[-1]</code>	Final element
Range	<code>arr[1:4]</code>	Indexes 1 through 3
Every second	<code>arr[::2]</code>	Step size of two
Reverse	<code>arr[::-1]</code>	Reverse order
Cell	<code>matrix[1, 2]</code>	Row 1, column 2
Row	<code>matrix[1, :]</code>	All columns from row 1
Column	<code>matrix[:, 1]</code>	Column 1 from every row

Copy-ready example

PYTHON

```
sales = np.array([[120, 150, 180],
                  [ 90, 110, 140],
                  [200, 175, 225]])

second_row = sales[1, :]
first_column = sales[:, 0]
center = sales[0:2, 1:3]
independent = sales[:, 1].copy()
```

Slices share memory

Basic slices usually return views. Call `copy()` when modifications must not affect the source array.

SECTION 07

Boolean Masking

Build boolean conditions to filter values, combine rules, replace matches, and locate matching positions.

TASK	SYNTAX	RESULT
Greater than	<code>arr[arr > 10]</code>	Keep values above 10
Range	<code>arr[(arr >= 10) & (arr <= 50)]</code>	Keep 10 through 50
Either condition	<code>arr[(arr < 0) (arr > 100)]</code>	Keep outside range
Invert mask	<code>arr[~mask]</code>	Keep false positions
Membership	<code>np.isin(arr, allowed)</code>	Boolean membership mask
Replace	<code>np.where(mask, a, b)</code>	Choose element-wise
Indexes	<code>np.flatnonzero(mask)</code>	Flat matching indexes
Count	<code>np.count_nonzero(mask)</code>	Number of matches

Copy-ready example

PYTHON

```
sales = np.array([120, 85, 210, 95, 175, 60])

high = sales[sales >= 150]
in_range = sales[(sales >= 90) & (sales <= 180)]
labels = np.where(sales >= 100, "High", "Low")
indexes = np.flatnonzero(sales >= 150)
```

Boolean operators

Use `&`, `|`, and `~` with arrays. Put each comparison in parentheses; do not use Python `and`, `or`, or `not`.

SECTION 08

Reshape & Flatten

Reorganize dimensions, flatten multidimensional data, and add or remove size-one axes.

TASK	SYNTAX	RESULT
Reshape	<code>arr.reshape(2, 3)</code>	Two rows, three columns
Infer dimension	<code>arr.reshape(2, -1)</code>	Calculate one axis
Flatten if possible	<code>arr.ravel()</code>	View when possible
Independent flat copy	<code>arr.flatten()</code>	Always a copy
Flatten by reshape	<code>arr.reshape(-1)</code>	One dimension
Remove size-one axes	<code>np.squeeze(arr)</code>	Drop length-one axes
Add axis	<code>np.expand_dims(arr, 0)</code>	Insert a dimension
Column shape	<code>arr[:, np.newaxis]</code>	Shape (n, 1)

Copy-ready example

PYTHON

```
values = np.arange(1, 13)
matrix = values.reshape(3, 4)
two_rows = values.reshape(2, -1)
flat_view = matrix.ravel()
flat_copy = matrix.flatten()
column = values[:, np.newaxis]
```

Preserve element count

A reshape must preserve the total number of elements. Only one requested dimension may use -1.

SECTION 09

Transpose & Axes

Swap rows and columns, reorder dimensions, and use axis arguments to control multidimensional calculations.

TASK	SYNTAX	RESULT
Transpose matrix	<code>matrix.T</code>	Swap rows and columns
Transpose function	<code>np.transpose(arr)</code>	Reverse axes by default
Reorder axes	<code>np.transpose(arr, (1, 2, 0))</code>	Custom axis order
Swap axes	<code>np.swapaxes(arr, 0, 2)</code>	Exchange two axes
Move axis	<code>np.moveaxis(arr, 0, -1)</code>	Move first axis last
Column totals	<code>matrix.sum(axis=0)</code>	One per column
Row totals	<code>matrix.sum(axis=1)</code>	One per row
Keep dimension	<code>sum(..., keepdims=True)</code>	Retain length-one axis

Copy-ready example

PYTHON

```
matrix = np.array([[10, 20, 30], [40, 50, 60]])
transposed = matrix.T

column_totals = matrix.sum(axis=0)
row_totals = matrix.sum(axis=1)
row_totals_2d = matrix.sum(axis=1, keepdims=True)

print(matrix.shape, transposed.shape) # (2, 3) (3, 2)
```

Read shapes left to right

For shape (2, 3, 4), axis 0 has length 2, axis 1 has length 3, and axis 2 has length 4.

SECTION 10

Broadcasting

Combine differently shaped arrays without manually repeating values. Compatible dimensions are equal or have length one.

TASK	SYNTAX	RESULT
Scalar	(3, 4) + ()	Result (3, 4)
Matrix + row	(3, 4) + (4,)	Result (3, 4)
Matrix + column	(3, 4) + (3, 1)	Result (3, 4)
Column + row	(3, 1) + (1, 4)	Result (3, 4)
Stack + matrix	(2, 3, 4) + (3, 4)	Result (2, 3, 4)
Incompatible	(3, 4) + (3,)	Broadcasting error

Copy-ready example

PYTHON

```
sales = np.array([[120, 150, 180], [90, 110, 140]])

column_adjustments = np.array([10, 20, 30])
adjusted_columns = sales + column_adjustments

row_rates = np.array([1.10, 0.90])[ :, np.newaxis]
adjusted_rows = sales * row_rates
```

Compare from the right

When shapes fail, compare dimensions from right to left and add a size-one axis where the intended alignment is clear.

SECTION 11

Vectorized Math

Apply arithmetic to complete arrays rather than processing one value at a time in Python loops.

TASK	SYNTAX	RESULT
Addition	<code>a + b</code>	Element-wise addition
Subtraction	<code>a - b</code>	Element-wise subtraction
Multiplication	<code>a * b</code>	Element-wise product
Division	<code>a / b</code>	True division
Floor division	<code>a // b</code>	Round toward -infinity
Power	<code>a ** 2</code>	Square every element
Remainder	<code>a % b</code>	Division remainder
Matrix product	<code>a @ b</code>	Rows multiplied by columns

Copy-ready example

PYTHON

```
units = np.array([12, 8, 15, 10])
prices = np.array([25.0, 40.0, 18.0, 32.0])
costs = np.array([14.0, 22.0, 10.0, 20.0])

revenue = units * prices
profit = revenue - units * costs
margin = np.divide(profit, revenue, where=revenue != 0)
print(profit.sum())
```

Prefer array expressions

Use operators, broadcasting, aggregations, and ufuncs before reaching for explicit Python iteration.

SECTION 12

Universal Functions

Apply optimized element-wise functions with support for broadcasting, conditional execution, output arrays, and reductions.

TASK	SYNTAX	RESULT
Absolute	<code>np.abs(arr)</code>	Magnitude
Square root	<code>np.sqrt(arr)</code>	Non-negative square root
Exponential	<code>np.exp(arr)</code>	e raised to values
Natural log	<code>np.log(arr)</code>	Log of positive values
Sine	<code>np.sin(arr)</code>	Angles in radians
Round down	<code>np.floor(arr)</code>	Toward -infinity
Element minimum	<code>np.minimum(a, b)</code>	Smaller corresponding value

TASK	SYNTAX	RESULT
Finite mask	<code>np.isfinite(arr)</code>	Exclude NaN and infinity

Copy-ready example

PYTHON

```
values = np.array([1.0, 4.0, 9.0, 16.0])
result = np.empty_like(values)
np.sqrt(values, out=result)

running_total = np.add.accumulate(values)
product = np.multiply.reduce(values)
table = np.multiply.outer([1, 2, 3], [10, 20, 30])
```

Validate domains

`sqrt()`, `log()`, and similar functions can produce NaN, infinity, or warnings when inputs fall outside their valid domain.

SECTION 13

Aggregations & Statistics

Summarize complete arrays or selected axes with totals, averages, variation, percentiles, extremes, and cumulative values.

TASK	SYNTAX	RESULT
Total	<code>np.sum(arr)</code>	Sum of values
Mean	<code>np.mean(arr)</code>	Arithmetic average
Weighted mean	<code>np.average(arr, weights=w)</code>	Weighted average
Median	<code>np.median(arr)</code>	Middle sorted value
Standard deviation	<code>np.std(arr, ddof=1)</code>	Sample deviation
Variance	<code>np.var(arr, ddof=1)</code>	Sample variance
Quartiles	<code>np.percentile(arr, [25, 50, 75])</code>	Q1, median, Q3
Maximum index	<code>np.argmax(arr)</code>	First maximum position
Running total	<code>np.cumsum(arr)</code>	Cumulative sum

Copy-ready example

PYTHON

```
sales = np.array([[120, 150, 180],
                  [ 90, 110, 140],
                  [200, 175, 225]])

column_totals = sales.sum(axis=0)
row_means = sales.mean(axis=1)
quartiles = np.percentile(sales, [25, 50, 75])
sample_std = sales.std(ddof=1)
```

Choose ddof deliberately

NumPy uses ddof=0 by default for population variance and standard deviation. A common sample estimate uses ddof=1.

SECTION 14

Sorting & Searching

Order values, return ordering indexes, retrieve top subsets, find insertion positions, and count unique values.

TASK	SYNTAX	RESULT
Sort	<code>np.sort(arr)</code>	Ascending sorted copy
Descending	<code>np.sort(arr)[::-1]</code>	Reverse ascending order
Ordering indexes	<code>np.argsort(arr)</code>	Indexes that sort
Small subset	<code>np.partition(arr, 3)[:3]</code>	Three small values
Large indexes	<code>np.argpartition(arr, -3)[-3:]</code>	Three large positions
Insertion	<code>np.searchsorted(arr, 25)</code>	Position in sorted data
Matching indexes	<code>np.flatnonzero(mask)</code>	Flat true positions
Unique counts	<code>np.unique(arr, return_counts=True)</code>	Values and frequency

Copy-ready example

PYTHON

```
products = np.array(["Keyboard", "Mouse", "Monitor", "Webcam"])
sales = np.array([120, 180, 95, 150])

order = np.argsort(sales)[::-1]
print(products[order])
print(sales[order])

top = np.argpartition(sales, -2)[-2:]
```

Partition is partial

partition() is efficient for top or bottom subsets, but the selected subset must be sorted separately if final order matters.

SECTION 15

Joining & Splitting

Combine arrays along existing or new axes and divide arrays into equal, uneven, row, or column sections.

TASK	SYNTAX	RESULT
Join existing axis	<code>np.concatenate([a, b], axis=0)</code>	Combine compatible arrays
Create new axis	<code>np.stack([a, b], axis=0)</code>	Inputs require equal shape
Stack rows	<code>np.vstack([a, b])</code>	Place vertically

TASK	SYNTAX	RESULT
Stack columns	<code>np.column_stack([a, b])</code>	1D arrays become columns
Equal sections	<code>np.split(arr, 3)</code>	Requires equal division
Uneven sections	<code>np.array_split(arr, 3)</code>	Allows uneven groups
Split columns	<code>np.hsplit(matrix, 2)</code>	Horizontal division
Split rows	<code>np.vsplit(matrix, 2)</code>	Vertical division

Copy-ready example

PYTHON

```
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

flat = np.concatenate([a, b]) # shape (6,)
rows = np.stack([a, b], axis=0) # shape (2, 3)
cols = np.stack([a, b], axis=1) # shape (3, 2)

groups = np.array_split(np.arange(10), 3)
```

Check shapes

For `concatenate()`, dimensions other than the joining axis must match. For `stack()`, every input shape must match.

SECTION 16

Copies & Views

Understand shared memory so that efficient views do not cause unintended modification of source data.

TASK	SYNTAX	RESULT
Explicit view	<code>view = arr.view()</code>	Shared underlying data
Independent copy	<code>copy = arr.copy()</code>	Independent storage
Shared check	<code>np.shares_memory(a, b)</code>	Definite memory sharing
Possible sharing	<code>np.may_share_memory(a, b)</code>	Conservative check
Owner reference	<code>arr.base</code>	Object backing the data
Own data flag	<code>arr.flags['OWNDATA']</code>	Memory ownership
Independent slice	<code>arr[1:4].copy()</code>	Safe to modify

Copy-ready example

PYTHON

```
original = np.array([10, 20, 30, 40, 50])

view = original[1:4]
copy = original[1:4].copy()

view[0] = 999 # changes original
copy[0] = 555 # does not change original

print(np.shares_memory(original, view)) # True
print(np.shares_memory(original, copy)) # False
```

Copy only when needed

Views save memory and allocation time. Copy at the point where independent modification or long-term isolation is required.

SECTION 17

Missing Values

Detect NaN, infinity, and missing datetimes; calculate NaN-aware summaries; and replace or mask unavailable data.

TASK	SYNTAX	RESULT
Detect NaN	<code>np.isnan(arr)</code>	NaN mask
Finite values	<code>np.isfinite(arr)</code>	Exclude NaN and infinity
Infinity	<code>np.isinf(arr)</code>	Positive or negative infinity
Remove NaN	<code>arr[~np.isnan(arr)]</code>	Keep available values
NaN-aware mean	<code>np.nanmean(arr)</code>	Ignore NaN
NaN-aware sum	<code>np.nansum(arr)</code>	Treat NaN as zero
Replace invalid	<code>np.nan_to_num(arr, nan=0.0)</code>	Defined replacements
Missing dates	<code>np.isnat(dates)</code>	NaT mask

Copy-ready example

PYTHON

```
values = np.array([12.5, np.nan, 18.0, np.inf, 21.5])

finite = values[np.isfinite(values)]
missing_count = np.count_nonzero(np.isnan(values))
average = np.nanmean(values[np.isfinite(values)])
cleaned = np.nan_to_num(values, nan=0.0, posinf=999.0)

dates = np.array(["2026-01-15", "NaT"], dtype="datetime64[D]")
```

Do not hide missingness

Keep the original missing-value mask before filtering or filling so the data-cleaning decision remains traceable.

SECTION 18

Random Numbers

Use the modern Generator API for simulations, sampling, shuffling, and draws from probability distributions.

TASK	SYNTAX	RESULT
Generator	<code>rng = np.random.default_rng()</code>	Automatic seed
Repeatable	<code>rng = np.random.default_rng(42)</code>	Fixed seed
Integers	<code>rng.integers(1, 7, size=10)</code>	Ten dice values
Decimals	<code>rng.random((2, 3))</code>	Values in [0, 1)
Uniform	<code>rng.uniform(10, 20, size=5)</code>	Uniform interval
Normal	<code>rng.normal(100, 15, size=1000)</code>	Gaussian samples
Sample	<code>rng.choice(arr, 3, replace=False)</code>	Unique sample
Permutation	<code>rng.permutation(arr)</code>	Shuffled copy
Shuffle	<code>rng.shuffle(arr)</code>	Modify in place

Copy-ready example

PYTHON

```
rng = np.random.default_rng(seed=42)

dice = rng.integers(1, 7, size=12)
heights = rng.normal(loc=170.0, scale=8.0, size=1000)
sample = rng.choice(np.arange(100), size=10, replace=False)

values = np.arange(10)
shuffled_copy = rng.permutation(values)
```

Not for security

NumPy random generators are for statistics and simulations, not passwords, tokens, encryption keys, or authentication.

SECTION 19

Linear Algebra

Multiply matrices, solve linear systems, calculate norms, inspect matrix properties, and perform standard decompositions.

TASK	SYNTAX	RESULT
Matrix product	<code>a @ b</code>	Rows times columns
Dot product	<code>np.dot(a, b)</code>	Vector or array product
Solve system	<code>np.linalg.solve(a, b)</code>	Square full-rank system
Least squares	<code>np.linalg.lstsq(a, b, rcond=None)</code>	Minimize residuals
Inverse	<code>np.linalg.inv(a)</code>	Invertible square matrix
Determinant	<code>np.linalg.det(a)</code>	Square matrix property

TASK	SYNTAX	RESULT
Rank	<code>np.linalg.matrix_rank(a)</code>	Independent dimensions
Norm	<code>np.linalg.norm(vector)</code>	Vector length
Eigen	<code>np.linalg.eigh(a)</code>	Symmetric matrix eigenpairs
SVD	<code>np.linalg.svd(a)</code>	Singular decomposition

Copy-ready example

PYTHON

```
# 3x + y = 9; x + 2y = 8
a = np.array([[3.0, 1.0], [1.0, 2.0]])
b = np.array([9.0, 8.0])

x = np.linalg.solve(a, b)      # [2. 3.]
det = np.linalg.det(a)
rank = np.linalg.matrix_rank(a)
condition = np.linalg.cond(a)
```

Prefer `solve()`

To solve $A @ x = b$, use `solve(A, b)` rather than explicitly calculating `inv(A) @ b`.

SECTION 20

File I/O

Store arrays in NPY or NPZ formats and exchange simple numeric data through text or CSV files.

TASK	SYNTAX	RESULT
Save one	<code>np.save('data.npy', arr)</code>	Preserve dtype and shape
Load one	<code>np.load('data.npy')</code>	Read NPY array
Save many	<code>np.savez('data.npz', a=a, b=b)</code>	Named NPZ archive
Compressed	<code>np.savez_compressed('data.npz', a=a)</code>	Compressed archive
Export CSV	<code>np.savetxt('data.csv', arr, delimiter=',')</code>	Numeric text
Load CSV	<code>np.loadtxt('data.csv', delimiter=',')</code>	Regular numeric data
Missing text	<code>np.genfromtxt('data.csv', delimiter=',')</code>	Missing-value support
Memory map	<code>np.load('data.npy', mmap_mode='r')</code>	Disk-backed access

Copy-ready example

PYTHON

```
sales = np.array([[120, 150, 180], [90, 110, 140]])
np.save("sales.npy", sales)
loaded = np.load("sales.npy", allow_pickle=False)

np.savez_compressed("financial.npz", sales=sales)

np.savetxt("sales.csv", sales, delimiter=",", fmt="%d")
csv_data = np.loadtxt("sales.csv", delimiter=",")
```

Security

Keep `allow_pickle=False` when loading untrusted data. Pickled object data can execute malicious code.

SECTION 21

Practical Examples

Combine NumPy techniques into reusable workflows for sales analysis, growth, normalization, outlier detection, and ranking.

TASK	SYNTAX	RESULT
Sales summary	<code>sales.sum(axis=1)</code>	One total per region
Best performer	<code>np.argmax(totals)</code>	Index of maximum
Growth	<code>(current - previous) / previous</code>	Period change
Moving average	<code>sliding_window_view(data, 3).mean(1)</code>	Rolling summary
Standardize	<code>(x - x.mean(0)) / x.std(0)</code>	Column z-scores
Outliers	<code>(x < lower) (x > upper)</code>	IQR mask
Weighted score	<code>(scores * weights).sum(1)</code>	Multi-criteria result

Copy-ready example

PYTHON

```
regions = np.array(["North", "South", "West"])
sales = np.array([[120, 150, 180],
                  [ 90, 110, 140],
                  [200, 175, 225]])

totals = sales.sum(axis=1)
best = regions[np.argmax(totals)]

q1, q3 = np.percentile(totals, [25, 75])
iqr = q3 - q1
outliers = totals[(totals < q1 - 1.5*iqr) | (totals > q3 + 1.5*iqr)]
```

Validate inputs

Check shape, dtype, and finite values before analysis. Arrays can contain correct-looking values in an incompatible structure.

SECTION 22

Performance & Best Practices

Vectorize meaningful operations, control allocation, choose safe dtypes, and measure representative workloads before optimizing.

TASK	SYNTAX	RESULT
Element calculations	Array operators / ufuncs	Avoid per-element Python loops
Repeated output	Preallocate final shape	Avoid repeated append
Shape expansion	Broadcasting	Avoid unnecessary tile
Memory	Smallest safe dtype	Avoid excess precision
Temporary output	ufunc out=	Reuse storage when safe
Subarrays	Views	Copy only for independence
Comparison	np.allclose()	Avoid exact float equality
Optimization	timeit / profiler	Measure real bottlenecks

Copy-ready example

PYTHON

```
values = np.arange(1_000_000, dtype=np.float32)

# Vectorized calculation
result = values * 1.25 + 10

# Reuse output storage
roots = np.empty_like(values)
np.sqrt(values, out=roots)

# Validate floating-point results
assert np.allclose(result[:3], [10.0, 11.25, 12.5])
```

Benchmark before and after

Keep the clearest correct implementation unless an alternative produces a meaningful improvement for the actual workload.

SECTION 23

Errors & Troubleshooting

Most NumPy errors become clearer after inspecting shape, dimensions, dtype, element count, and finite values.

TASK	SYNTAX	RESULT
Broadcast error	print(a.shape, b.shape)	Compare from the right
Reshape error	print(arr.size)	Check requested product
Index error	print(arr.shape)	Validate axis length
Too many indexes	print(arr.ndim)	Check dimensions

TASK	SYNTAX	RESULT
AxisError	<code>axis < arr.ndim</code>	Use existing axis
Casting error	<code>arr.astype(np.float64)</code>	Use compatible dtype
Invalid numeric	<code>np.isfinite(arr)</code>	Find NaN or infinity
Singular matrix	<code>np.linalg.matrix_rank(a)</code>	No unique solution

Copy-ready example

PYTHON

```
def validate_numeric_2d(arr, columns):
    if arr.ndim != 2:
        raise ValueError(f"Expected 2D, received {arr.ndim}D")
    if arr.shape[1] != columns:
        raise ValueError(f"Expected {columns} columns")
    if not np.issubdtype(arr.dtype, np.number):
        raise TypeError("Expected numeric dtype")
    if not np.all(np.isfinite(arr)):
        raise ValueError("Data contains NaN or infinity")
```

Debug metadata first

Start with shape, ndim, dtype, size, and isfinite() instead of printing an entire large array.

SECTION 24

NumPy FAQ

Concise answers to common questions about arrays, performance, axes, broadcasting, missing data, random numbers, and matrix operations.

What is NumPy used for? Fast numerical computing with multidimensional arrays, including statistics, simulation, scientific computing, and linear algebra.	How is an ndarray different from a Python list? An ndarray normally has one dtype, compact storage, multidimensional indexing, broadcasting, and vectorized numerical operations.
When should I use NumPy instead of pandas? Use NumPy for dense numerical arrays and matrix calculations. Use pandas when labeled tabular data, joins, mixed dtypes, and grouped operations matter.	What do axis=0 and axis=1 mean? For a 2D array, axis=0 reduces rows and returns one value per column; axis=1 reduces columns and returns one value per row.
What is broadcasting? It is NumPy's rule for calculating with differently shaped but compatible arrays without manually repeating data.	What is the difference between a view and a copy? A view shares underlying data with another array. A copy owns independent data storage.
How do I change dtype? Use <code>astype()</code> , while checking for lost precision, truncated fractions, and values outside the target range.	How do I ignore NaN? Use NaN-aware functions such as <code>nanmean()</code> , <code>nansum()</code> , <code>nanmedian()</code> , and <code>nanmax()</code> when excluding missing observations is appropriate.
How do I create repeatable random values? Create one Generator with <code>np.random.default_rng(seed)</code> and reuse it throughout the workflow.	What is the difference between * and @? The <code>*</code> operator multiplies corresponding elements. The <code>@</code> operator performs a matrix product.
Why does reshape fail? The product of the requested dimensions must equal <code>arr.size</code> . One dimension may be -1 so NumPy can infer it.	Why are float results sometimes inexact? Many decimals have no exact binary floating-point representation. Compare results with <code>isclose()</code> or <code>allclose()</code> .