Advanced NumPy Cheat Sheet

1. Array Creation & Initialization

Command	Example	Explanation
`np.array`	`np.array([,], dtype=float)`	Make array from iterable; specify datatype
`np.empty((3,4))`	`np.empty((3,4))`	Create uninitialized array (fast, values random)
`np.zeros_like(a)`	`np.zeros_like(existing_array)`	Create zero-array same shape/type as `a`
`np.arange(10, 0, -2)`	`np.arange(10, 0, -2)`	Range:
`np.random.default_rng()`	`rng = np.random.default_rng(42)`	Modern random number generator for reproducibility

2. Indexing & Advanced Slicing

Command	Example	Explanation
Boolean masking	`a[a > 5]`	Filter array with condition
Advanced integer indexing	`a[,]`	Select elements by tuple of indices
Fancy assignment	`a[,:] = 99`	Set rows 0 & 2 to 99
Multi-axis indexing (via `np.ix_`)	`a[np.ix_(,)]`	Outer product-like selection
np.take / np.put	`np.take(a, )`	Extract by flattened index
in-place update using mask	`a[a<0] = 0`	Set all negative values to zero

3. Broadcasting & Vectorization

Command	Example	Explanation
Broadcasting automatic	`a + b` where a.shape=(m,1), b.shape=(n,)	Auto "stretch" shapes for math
Vectorized operations	`np.exp(a)`, `np.sqrt(a)`, `np.log1p(a)`	Fast elementwise math, no loops

Memory-efficient view with `np.newaxis`	`a[:, np.newaxis] - b`	Broadcast differences row-wise

4. Performance & Memory

Command	Example	Explanation
Use `inplace` variations	`np.add(a, b, out=a)`	Update `a` without extra memory
Data types: minimize memory	`a = np.array(, dtype='int8')`	Tiny arrays, tune dtypes for speed
Convert/copy data type	`b = a.astype(np.float32, copy=False)`	Zero-copy type conversion if aligned
Array views (not copies)	`b = a[::2]`	Modifies original if `b` is edited

5. Linear Algebra (np.linalg)

Command	Example	Explanation
Matrix multiplication	`np.dot(A, B)` or `A @ B`	Matrix-product (2D arrays)
Solve Ax = b	`np.linalg.solve(A, b)`	Efficient linear system solution
Eigenvalues, vectors	`np.linalg.eig(A)`	Spectral analysis
SVD	`U,S,Vt = np.linalg.svd(A)`	Decompose matrix, PCA foundation
Norms	`np.linalg.norm(A)`, `np.linalg.norm(A, 'fro')`	L2 (default), Frobenius, L1, etc.
Inverse/Determinant	`np.linalg.inv(A)`, `np.linalg.det(A)`	For invertible matrices

6. Advanced Manipulation & Reshaping

Command	Example	Explanation
Axis-wise ops	`A.sum(axis=0)`, `A.mean(axis=1)`	Collapses rows (axis=0) or cols (axis=1)
Swap axes	`A.swapaxes(0,1)`	Interchange two axes

Roll axis	`A = np.rollaxis(A, 2)`	Move axes to new position
Reshape with -1	`A.reshape(-1, 3)`	Let NumPy infer size for one dimension
Tile/repeat	`np.tile(A, (3,2))`, `np.repeat(x, 4)`	Repeat array/matrix pattern
Block, hstack, vstack, dstack	`np.block([...])`, `np.vstack`	Advanced ways to join arrays

7. Random, Sampling & Probability Distributions

Command	Example	Explanation
Random normal	`rng.normal(0, 1, (1000,2))`	1000 samples, mean=0, std=1
Choice sampling	`rng.choice(, size=5, replace=False)`	Unique random draw
Multivariate normal	`rng.multivariate_normal(mean, cov, 500)`	Simulate correlated data
Random permutation (indices)	`rng.permutation(n)`	Shuffle indices 0...n-1

8. Broadcasting & Meshgrid for Advanced Vectorized Calculations

Command	Example	Explanation
Grid computation	`X, Y = np.meshgrid(x, y); Z = np.sin(X) * np.cos(Y)`	Evaluate f(x, y) on grid
Outer product	`np.outer(x, y)`	Matrix: all pairwise multiplications

9. Masked Arrays & Handling Missing Data

Command	Example	Explanation
np.isnan, np.isinf	`np.isnan(arr)`	Find NaNs/infs
np.nan_to_num	`np.nan_to_num(arr, nan=0.0)`	Replace NaNs/infs
np.ma.masked_where	`np.ma.masked_where(arr < 0, arr)`	Mask negative entries
np.ma.filled	`masked_arr.filled(0)`	Replace masked with fill value

10. Interfacing with Other Libraries

Command	Example	Explanation
Array to pandas DataFrame	`df = pd.DataFrame(a, columns=cols)`	Integrate with pandas for analysis
Read/Write file (CSV, NPY, NPZ)	`np.loadtxt('data.csv', delimiter=',')`, `np.savez('x.npz', x=a)`	Fast I/O
Interop with sklearn pre-processing	`from sklearn.preprocessing import StandardScaler`	Many sklearn tools take/return ndarrays

11. Universal Functions ("ufuncs") and Custom ufunc

- **Fast elementwise ops:** All math ops are vectorized.
- **Combine arrays:** `np.add`, `np.maximum`, `np.where(condition, a, b)`
- **Write your own:**

```
def f(x): return x**2 + 2*x + 1
vf = np.vectorize(f)
result = vf(np.arange(5))
```

12. Parallelization and Advanced Tricks

- **Multi-threaded BLAS**: NumPy dispatches to MKL/OpenBLAS for large matrix ops.
- **Batch processing:** Use `array.reshape` or `einsum` for efficient multi-array ops.
- **Einstein summation:**

```
np.einsum('ij,jk->ik', A, B)  # fast matrix multiplication
np.einsum('i,i->', a, a)      # dot product
```

- **Broadcast with np.newaxis and np.expand_dims for deep vectorization.**

13. Linear Algebra, Polynomials, FFT, and More

- **Polynomials:**

```
p = np.poly1d([1,2,3])
p(4)  # evaluate polynomial
p.deriv(), p.integ()  # derivative/integral
```

- **Fast Fourier Transform:**

```
freq = np.fft.rfft(signal)
amp = np.abs(freq)
```

- **Solving Ax = b with constraints:**
  Use `np.linalg.lstsq` (least squares), singular matrices, pseudo-inverse `np.linalg.pinv`.

Example: Batch Outer Products Using `einsum` for Deep Learning

```
# Compute outer product for each row in batch
batch = np.random.randn(100, 32)  # 100 samples, 32 features each
result = np.einsum('bi,bj->bij', batch, batch)
# result.shape == (100, 32, 32)
```

Pro Tips

- Prefer `np.dot`, `np.matmul`, or `@` for matrix math; `np.einsum` for complex batching.
- Use in-place ops and minimal dtype for memory efficiency.
- Chain slicing/broadcasting: no copies unless you modify.
- Profile performance with `%timeit` (in Jupyter/IPython) or `np.show_config()` to check BLAS backend.