🧠 NumPy Cheat Sheet with One-Line Comments

📦 Import

```
import numpy as np  # Standard way to import NumPy
```

📐 Creating Arrays

```
np.array([1, 2, 3])          # Create an array from a list
np.zeros((2, 3))             # Create an array filled with zeros
np.ones((2, 2))              # Create an array filled with ones
np.full((2, 2), 7)           # Create an array filled with the number 7
np.eye(3)                    # Create a 3x3 identity matrix
np.arange(0, 10, 2)          # Create array from 0 to 10 with step of 2
np.linspace(0, 1, 5)         # Create 5 evenly spaced values from 0 to 1
np.random.rand(2, 3)         # Create 2x3 array of random floats (0-1)
np.random.randn(2, 3)        # Create 2x3 array from standard normal distribution
np.random.randint(1, 10, (2, 3)) # Random integers from 1 to 9 in 2x3 shape
```

📏 Array Attributes

```
a.shape      # Shape of the array (rows, cols)
a.ndim       # Number of dimensions
a.size       # Total number of elements
a.dtype      # Data type of elements
a.itemsize   # Byte size of one element
a.nbytes     # Total memory consumed (bytes)
a.T          # Transpose of array
```

🔁 Reshape & Resize

```
a.reshape(3, 2)            # Reshape without changing data
a.ravel()                  # Flatten array (returns view)
a.flatten()                # Flatten array (returns copy)
a.resize((3, 3))           # Resize array in-place
np.expand_dims(a, axis=0)  # Add a dimension at axis
np.squeeze(a)              # Remove single-dimensional entries
```

🧮 Math Operations

```
a + b, a - b, a * b        # Element-wise operations
a / b, a ** 2              # Element-wise divide, power
np.add(a, b)               # Element-wise addition
np.exp(a), np.log(a)       # Exponential and logarithmic
np.sqrt(a)                 # Square root
np.abs(a), np.round(a)     # Absolute and round
```

📊 Statistics & Aggregation

```
np.sum(a)                  # Sum of all elements
np.mean(a)                 # Mean value
np.std(a), np.var(a)       # Standard deviation, variance
np.min(a), np.max(a)       # Minimum and maximum
np.argmin(a), np.argmax(a) # Index of min and max
np.median(a)               # Median value
np.percentile(a, 75)       # 75th percentile
```

🔍 Indexing & Slicing

```
a[0], a[-1]              # Access first/last element

a[1:3], a[:, 0]          # Slice rows/columns

a[::2]                   # Every second element

a[a > 5]                 # Boolean masking

np.where(a > 5)          # Return indices where condition is True

np.argwhere(a == 3)      # Indices where elements are equal to 3

np.extract(a % 2 == 0, a)   # Extract even numbers
```

🔧 Modify Arrays

```
np.append(a, [10, 11])      # Append values

np.insert(a, 2, [5, 6])     # Insert at position

np.delete(a, [1, 2])        # Delete by index

np.put(a, [0, 3], [99, 100])  # Replace by index

np.clip(a, 0, 10)           # Limit values between 0 and 10
```

🧩 Concatenate & Split

```
np.concatenate([a, b], axis=0)  # Join along rows

np.stack([a, b], axis=0)        # Stack with new axis

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

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

np.split(a, 2)                  # Split into 2 equal parts

np.array_split(a, 3)            # Split into 3 parts (not necessarily equal)
```

🌀 Sorting & Searching

```
np.sort(a)                # Sort elements
np.argsort(a)             # Indices of sorted elements
np.searchsorted(a, 5)     # Index to insert to maintain order
np.where(a == 10)         # Indices where element is 10
np.isin(a, [2, 4, 6])     # Check if elements exist
```

🧠 Set Operations

```
np.unique(a)              # Unique sorted elements
np.intersect1d(a, b)      # Common elements
np.union1d(a, b)          # Union of both arrays
np.setdiff1d(a, b)        # Elements in a not in b
np.setxor1d(a, b)         # Elements in either, not both
np.in1d(a, b)             # Check if elements of a in b
```

🚀 Broadcasting

```
a + 5                     # Scalar automatically broadcasts
a + b                     # Arrays of different shapes can still work
```

✅ Use when array shapes differ but still compatible by rules.

🎲 Random Functions

```
np.random.seed(0)             # Fix random seed for reproducibility
np.random.rand(2, 3)          # Uniform [0, 1)
np.random.randn(2, 3)         # Standard normal distribution
np.random.randint(1, 10, (2, 3)) # Random ints in shape
np.random.choice([1, 2, 3], 5)   # Random samples from list
np.random.permutation(10)        # Random permutation of range
```

🧮 Linear Algebra (np.linalg)

```
np.dot(a, b)              # Matrix multiplication
np.matmul(a, b)             # Matrix multiplication (preferred)
np.linalg.inv(a)            # Matrix inverse
np.linalg.det(a)            # Determinant
np.linalg.eig(a)            # Eigenvalues and eigenvectors
np.linalg.norm(a)             # Vector/matrix norm
np.linalg.solve(A, b)         # Solve system of linear equations
```

💾 File I/O

```
np.save("arr.npy", a)         # Save array in binary format
np.load("arr.npy")            # Load .npy file
np.savetxt("arr.csv", a, delimiter=",") # Save to CSV
np.loadtxt("arr.csv", delimiter=",")    # Load from CSV
```

⚙️ Utility Functions

```
np.isnan(a), np.isinf(a)        # Check for NaN or Inf
np.allclose(a, b)             # Compare with tolerance
np.array_equal(a, b)            # Compare arrays for equality
np.meshgrid(x, y)             # Create coordinate matrices
np.tile(a, (2, 3))            # Repeat array in grid
np.repeat(a, 3)               # Repeat elements
np.flip(a), np.roll(a, 2)       # Reverse or rotate array
```

🚀 Advanced NumPy Methods Cheat Sheet (With One-Line Descriptions)

🔢 Array Creation (Advanced)

```
np.fromfunction(func, shape)      # Create array from function over indices
np.fromiter(iterable, dtype)      # Create array from iterable
np.fromstring(string, dtype)      # Convert string to array
np.empty((3, 3))                  # Uninitialized array (faster)
np.empty_like(a)                  # Empty array with same shape as `a`
np.meshgrid(x, y)                 # Create coordinate matrix from vectors
np.ogrid[:3, :4]                  # Open grid (memory efficient)
np.mgrid[:3, :4]                  # Dense grid
```

🔁 Iteration & Custom Application

```
np.nditer(a)                      # Efficient multidimensional iterator
np.ndenumerate(a)                 # Iterator with index and value
np.vectorize(func)                # Vectorize scalar function over arrays
np.apply_along_axis(func, axis, arr)# Apply func to 1D slices of 2D array
np.apply_over_axes(func, arr, axes) # Apply func across specified axes
```

⚡ Performance & Memory

```
np.copy(a, order='C')             # Copy array with memory layout
np.ascontiguousarray(a)           # Ensure C-style contiguous layout
np.asfortranarray(a)              # Ensure Fortran-style layout
np.array_equal(a, b)              # Check if arrays are exactly equal
np.allclose(a, b)                 # Check if arrays are equal within tolerance
a.flags                           # Show memory layout (C/F-contiguous)
```

🧪 Advanced Mathematical Functions

```
np.isfinite(a), np.isinf(a), np.isnan(a)   # Detect special values
np.nan_to_num(a)                    # Replace NaN, inf with numbers
np.interp(x, xp, fp)                # 1D linear interpolation
np.gradient(a)                      # Numerical gradient
np.diff(a)                          # Discrete difference along axis
np.cov(m)                           # Covariance matrix
np.corrcoef(m)                      # Correlation coefficients
np.histogram(a, bins=10)            # Histogram of array
```

🔍 Set & Logic (Advanced)

```
np.in1d(a, b)                       # Boolean mask where elements of a are in b
np.setdiff1d(a, b)                  # Elements in a not in b
np.setxor1d(a, b)                   # Symmetric difference
np.intersect1d(a, b, assume_unique=True)  # Common elements
np.union1d(a, b)                    # Union of unique elements
np.unique(a, return_counts=True)        # Unique values with counts
```

🧱 Matrix & Linear Algebra

```
np.dot(a, b)                        # Matrix multiplication
np.matmul(a, b)                     # Preferred matrix multiply (broadcast-aware)
np.einsum('ij,jk->ik', a, b)            # Einstein summation (optimized math)
np.linalg.inv(a)                    # Matrix inverse
np.linalg.pinv(a)                   # Moore-Penrose pseudo-inverse
np.linalg.det(a)                    # Determinant
np.linalg.matrix_rank(a)                # Matrix rank
np.linalg.eig(a)                    # Eigenvalues/vectors
np.linalg.qr(a)                  # QR decomposition
np.linalg.svd(a)                  # Singular value decomposition
np.linalg.solve(A, b)               # Solve system Ax = b
np.linalg.norm(a, ord=2)              # Matrix/vector norm
```

🧼 Handling Missing Data (NaN-safe methods)

```
np.isnan(a)                     # Detect NaNs
np.nanmean(a), np.nanstd(a)          # Ignore NaNs in mean/std
np.nanmin(a), np.nanmax(a)           # Ignore NaNs in min/max
np.nanargmin(a), np.nanargmax(a)       # Ignore NaNs for index of min/max
```

🔀 Repetition, Tiling & Stacking

```
np.repeat(a, repeats)               # Repeat each element
np.tile(a, reps)                  # Repeat array like a grid
np.column_stack([a, b])              # Stack 1D as columns
np.row_stack([a, b])                # Stack arrays as rows
np.hstack([a, b]), np.vstack([a, b])     # Horizontal / vertical stack
np.dstack([a, b])                  # Stack along depth (3D)
```

🔧 Array Structure Manipulation

```
np.roll(a, shift)                 # Roll elements cyclically
np.flip(a)                      # Reverse array along axis
np.rot90(a, k=1)                  # Rotate 90 degrees (2D)
np.swapaxes(a, 0, 1)                # Swap two axes
np.moveaxis(a, source, destination)      # Move one axis to new position
```

🧬 Index Tricks

```
np.r_[1:5, 10:15]               # Concatenate ranges

np.c_[a, b]                     # Combine as columns

np.diag(a)                      # Extract or create diagonal

np.tril(a), np.triu(a)          # Lower/Upper triangle

np.fill_diagonal(a, value)      # Set diagonal values
```

💾 Advanced File I/O

```
np.genfromtxt('data.csv', delimiter=',')  # Load data with missing values

np.loadtxt('data.csv', skiprows=1)        # Load text skipping headers

np.memmap('data.dat', dtype='float32', mode='r', shape=(1000,1000))
```

✅ np.memmap: Handle huge files that don’t fit in memory.
