Numpy CheatSheet

Setup & basics

```
import numpy as np
```

Create arrays

```
a = np.array([1, 2, 3])           # 1-D array
b = np.array([[1, 2], [3, 4]])    # 2-D array (matrix)
c = np.zeros((3, 4))              # all zeros, shape 3×4
d = np.ones((2, 3), dtype=int)    # all ones, type int
e = np.arange(0, 10, 2)           # [0,2,4,6,8]
f = np.linspace(0, 1, 5)          # [0.,0.25,0.5,0.75,1.]
g = np.random.rand(3, 3)          # random floats in [0,1)
```

Inspect arrays

```
a.shape             # returns tuple of dimensions
a.dtype             # dtype of elements (e.g. int64, float64)
a.ndim              # number of dimensions
a.size              # total number of elements
a.itemsize          # size in bytes per element
a.reshape((rows, cols))  # create view with new shape
a.flatten()         # flatten to 1-D
a.T                 # transpose (for 2-D)
```

Vectorized ops & arithmetic

```
a = np.array([1,2,3])
b = np.array([10,20,30])

c = a + b           # [11,22,33]
d = a * b           # [10,40,90]
e = a ** 2          # [1,4,9]
f = a + 5           # [6,7,8] (broadcast)
```

```
g = np.sin(a)     # apply ufunc element-wise
```

Reduction functions

```
a.sum()          # sum of all elements
a.mean()
a.std()
a.min()
a.max()
a.argmin()       # index of min value
a.argmax()
```

Axis parameter

```
M = np.array([[1,2,3],[4,5,6]])
M.sum(axis=0)    # [5,7,9] sum columns
M.sum(axis=1)    # [6,15] sum rows
```

Indexing & slicing

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

a[2]            # 30
a[-1]           # 50
a[1:4]          # [20,30,40]
a[:3]           # [10,20,30]
a[3:]           # [40,50]
a[::2]          # [10,30,50]
```

Multi-D indexing

```
M = np.array([[1,2,3],[4,5,6],[7,8,9]])
M[1,2]          # 6
M[1]            # [4,5,6] (second row)
M[:,0]          # [1,4,7] (first column)
M[1:,1:]        # sub-matrix
```

Boolean indexing / masking

```
a = np.array([1,2,3,4,5])
mask = (a > 2)
a[mask]         # [3,4,5]
a[a % 2 == 0]   # [2,4]
```

Fancy indexing

```
a = np.array([5,10,15,20,25])
indices = [0,2,4]
a[indices]      # [5,15,25]
```

Broadcasting rules

- Two arrays are compatible for broadcasting if for each trailing dimension length either equal or one is 1.

Example:

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

Reshaping & concatenation

```
a = np.arange(6)          # [0,1,2,3,4,5]
a2 = a.reshape((2,3))     # [[0,1,2],[3,4,5]]
b = np.array([[6,7,8],[9,10,11]])
c = np.vstack((a2, b))    # vertical stack
d = np.hstack((a2, b))    # horizontal stack
e = np.concatenate((a2, b), axis=0)  # same as vstack
```

Linear algebra

```
A = np.array([[1,2],[3,4]])
B = np.array([[5,6],[7,8]])

C = np.dot(A, B)          # matrix multiply
D = A @ B                 # operator form in newer Python
eigvals, eigvecs = np.linalg.eig(A)
invA = np.linalg.inv(A)
detA = np.linalg.det(A)
```

Random numbers & distributions

```
np.random.seed(0)         # for reproducibility
r = np.random.rand(3,3)   # uniform [0,1)
r_int = np.random.randint(0, 10, size=(2,3))    # random ints
norm = np.random.normal(loc=0, scale=1, size=(1000,))  # Gaussian
```

Performance tips

- Use vectorized ops (`arr * 2`, `arr > 3`) whenever possible, avoid loops over arrays.
- Avoid `for x in arr:` if you can apply ufunc or vectorized operation.
- Use `arr.astype(...)` to cast dtypes if needed, but avoid unnecessary copies.
- For very large data, ensure arrays are contiguous (`arr = arr.copy(order='C')`).
- Memory access pattern matters. Row-major layout (C) is default. Use `.T` or `.reshape` carefully.

Interoperability with Python

```
lst = [1,2,3]
arr = np.array(lst)
```

```
lst2 = arr.tolist()       # convert back to Python list

for x in arr:             # still allowed, but slower than vector ops
    ...
```

When not to use NumPy (or less useful)

- Very small arrays (overhead of import and creation may dominate).
- Pure control-flow logic, heavy Python-object structures.
- When library is not allowed (e.g., some online coding platforms restrict imports).
- When you need mixed types (NumPy is homogeneous dtype).