NumPy Cheat Sheet (Theory Only)

---

1. Basics of NumPy

NumPy (Numerical Python) is a library for **numerical computing** in Python.

Provides **multi-dimensional arrays** (ndarray) and **mathematical functions** for efficient operations.

Used in **scientific computing, data analysis, machine learning, and engineering applications**.

Importing NumPy:

```
import numpy as np
```

Creating Arrays:

```
arr = np.array([1, 2, 3, 4])
print(arr)  # Output: [1 2 3 4]
```

Properties of an Array:

```
print(arr.shape)  # Shape of array
print(arr.ndim)   # Number of dimensions
print(arr.size)   # Total number of elements
print(arr.dtype)  # Data type of elements
```

---

2. Computation on NumPy (Element-wise Operations)

Arithmetic Operations on Arrays:

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

print(a + b)  # Output: [5 7 9]
print(a * b)  # Output: [ 4 10 18]
print(a ** 2) # Output: [1 4 9]
```

Universal Functions (ufuncs):

```
print(np.sin(a))  # Sine function on each element
print(np.log(a))  # Natural logarithm
print(np.exp(a))  # Exponential function
```

---

3. Aggregations (Summarizing Data)

Common Aggregations:

```
arr = np.array([1, 2, 3, 4, 5])

print(arr.sum())      # Output: 15
print(arr.mean())     # Output: 3.0
print(arr.min())      # Output: 1
print(arr.max())      # Output: 5
print(arr.std())      # Standard deviation
```

Aggregation along Axis (Rows/Columns):

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

print(matrix.sum(axis=0))  # Column-wise sum
print(matrix.sum(axis=1))  # Row-wise sum
```

---

4. Computation on Arrays

Broadcasting (Operations on Different Sized Arrays):

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

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

print(A + B)
# Output:
# [[2 4 6]
#  [5 7 9]]
```

Reshaping Arrays:

```
arr = np.arange(6).reshape(2, 3)
print(arr)
# Output:
# [[0 1 2]
#  [3 4 5]]
```

---

5. Comparisons, Masks, and Boolean Arrays

Comparing Arrays:

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

print(arr > 20)  # Output: [False False  True  True]
```

Using Boolean Masks:

```
mask = arr > 20
print(arr[mask])  # Output: [30 40]
```

Replacing Values with Conditions:

```
arr[arr > 20] = 100
print(arr)  # Output: [10 20 100 100]
```

---

6. Fancy Indexing (Advanced Indexing)

Indexing with Lists or Arrays:

```
arr = np.array([10, 20, 30, 40, 50])
indices = [0, 2, 4]

print(arr[indices])  # Output: [10 30 50]
```

Indexing a 2D Array:

```
matrix = np.array([[1, 2, 3], [4, 5, 6]])
rows = np.array([0, 1])
cols = np.array([2, 1])

print(matrix[rows, cols])  # Output: [3 5]
```

---

7. Sorting Arrays

Sorting a 1D Array:

```
arr = np.array([3, 1, 5, 2, 4])
sorted_arr = np.sort(arr)
print(sorted_arr)  # Output: [1 2 3 4 5]
```

Sorting a 2D Array:

```
matrix = np.array([[5, 2, 9], [3, 8, 1]])
print(np.sort(matrix, axis=1))  # Row-wise sorting
print(np.sort(matrix, axis=0))  # Column-wise sorting
```

Argsort (Indices of Sorted Elements):

```
indices = np.argsort(arr)
print(indices)  # Output: [1 3 0 4 2]
```

---

8. Structured Data: NumPy’s Structured Array

Structured Arrays store mixed data types.

Useful for **handling tabular data (similar to pandas DataFrame).**

Example:

```
data = np.array([(1, "Alice", 25.5), (2, "Bob", 30.0)],
                 dtype=[("ID", "i4"), ("Name", "U10"), ("Age", "f4")])

print(data["Name"])  # Output: ['Alice' 'Bob']
print(data["Age"])   # Output: [25.5 30.0]
```

Key Takeaways

NumPy Arrays → Efficiently store and manipulate data.
Element-wise Computation → Mathematical operations apply to all elements.
Aggregation → Functions like sum(), mean(), max() summarize data.
Boolean Masking → Filtering values based on conditions.
Fancy Indexing → Advanced indexing using lists and arrays.
Sorting Arrays → Sorting and retrieving sorted indices using np.sort().
Structured Arrays → Handle mixed data types efficiently.

This **NumPy Cheat Sheet** covers **arrays, computations, aggregations, comparisons, masking, fancy indexing, sorting, and structured data**. Let me know if you need further explanations!