NumPy Cheatsheet & Quick Reference

A concise reference guide to essential NumPy operations for scientific computing in Python. Covers array creation, manipulation, indexing, and common mathematical and statistical functions.

Getting Started & Array Creation

Introduction & Import

NumPy is the fundamental package for scientific computing with Python.

Importing NumPy:

```
import numpy as np
```

Importing/Exporting Data

`np.loadtxt('file.txt')`	From a text file.
`np.genfromtxt('file.csv', delimiter=',')`	From a CSV file, handles missing values.
`np.savetxt('file.txt', arr, delimiter=' ')`	Writes array `arr` to a text file.
`np.savetxt('file.csv', arr, delimiter=',')`	Writes array `arr` to a CSV file.

Creating Arrays

`np.array([1,2,3])`	One-dimensional array. **Example:** `arr = np.array([1, 2, 3])`
`np.array([(1,2,3),(4,5,6)])`	Two-dimensional array. **Example:** `arr2d = np.array([(1,2,3),(4,5,6)])`
`np.zeros(3)`	1D array of length 3 with all values 0. **Example:** `np.zeros(3) # Output: [0. 0. 0.]`
`np.ones((3,4))`	3x4 array with all values 1. **Example:** `np.ones((3,4))`
`np.eye(5)`	5x5 identity matrix (1 on diagonal, 0 elsewhere). **Example:** `np.eye(3)`

`np.linspace(0, 100, 6)`	Array of 6 evenly spaced values from 0 to 100.
	Example:
	`np.linspace(0, 100, 6)`
	`# Output: [ 0. 20. 40. 60. 80. 100.]`

`np.arange(0, 10, 3)`	Array of values from 0 up to (but not including) 10, with step 3. **Example:** `np.arange(0, 10, 3) # Output: [0 3 6 9]`
`np.full((2,3), 8)`	2x3 array with all values 8. **Example:** `np.full((2,3), 8)`
`np.random.rand(4,5)`	4x5 array of random floats between 0 and 1.
`np.random.rand(6,7) * 100`	6x7 array of random floats between 0 and 100.
`np.random.randint(5, size=(2,3))`	2x3 array with random integers between 0 (inclusive) and 5 (exclusive). **Example:** `np.random.randint(5, size=(2,3))`

Properties & Manipulation

Inspecting Properties

`arr.size`	Returns the total number of elements in `arr`.

`arr.shape`	Returns the dimensions of `arr` as a tuple (rows, columns, ...).
	Example:
	`arr2d = np.array([(1,2,3),(4,5,6)])`
	`print(arr2d.shape) # Output: (2, 3)`

`arr.dtype`	Returns the data type of the elements in `arr`.
`arr.astype(dtype)`	Converts the elements of `arr` to the specified `dtype`.

`arr.tolist()`	Converts `arr` to a Python list.
	Example:
	`arr = np.array([1, 2, 3])`
	`print(arr.tolist()) # Output: [1, 2, 3]`

`np.info(np.eye)`	View documentation for a specific NumPy function.

Copying, Sorting & Reshaping

`np.copy(arr)`	Creates a deep copy of `arr` in new memory.
`arr.view(dtype)`	Creates a new view of `arr`'s data with a different data type.

`arr.sort()`	Sorts `arr` in-place.
	Example:
	`arr = np.array([3, 1, 2])`
	`arr.sort()`
	`print(arr) # Output: [1 2 3]`

`arr.sort(axis=0)`	Sorts `arr` along the specified axis.
`two_d_arr.flatten()`	Flattens a 2D array `two_d_arr` to a 1D array.

`arr.T`	Transposes `arr` (rows become columns and vice versa).
	Example:
	`arr = np.array([[1,2],[3,4]])`
	`print(arr.T)`
	`# Output:`
	`# [[1 3]`
	`#  [2 4]]`

`arr.reshape(3,4)`	Reshapes `arr` to 3 rows and 4 columns without changing data (returns a new view).
`arr.resize((5,6))`	Changes the shape of `arr` in-place to 5x6, filling new values with 0.

Adding/Removing Elements

`np.append(arr, values)`	Appends `values` to the end of `arr`.

`np.insert(arr, 2, values)`	Inserts `values` into `arr` before index 2.
	Example:
	`arr = np.array([1, 2, 4, 5])`
	`np.insert(arr, 2, 3) # Output: [1 2 3 4 5]`

`np.delete(arr, 3, axis=0)`	Deletes the row at index 3 from `arr`.
`np.delete(arr, 4, axis=1)`	Deletes the column at index 4 from `arr`.

Combining/Splitting

`np.concatenate((arr1, arr2), axis=0)`	Adds `arr2` as rows to the end of `arr1` (vertical stack).
	Example:
	`a = np.array([[1, 2], [3, 4]])`
	`b = np.array([[5, 6]])`
	`np.concatenate((a, b), axis=0)`

`np.concatenate((arr1, arr2), axis=1)`	Adds `arr2` as columns to the end of `arr1` (horizontal stack).
	Example:
	`a = np.array([[1, 2], [3, 4]])`
	`b = np.array([[5], [6]])`
	`np.concatenate((a, b), axis=1)`

`np.split(arr, 3)`	Splits `arr` into 3 sub-arrays along the first axis.
`np.hsplit(arr, 5)`	Splits `arr` horizontally into 5 sub-arrays (along axis 1).
`np.vstack((arr1, arr2))`	Stack arrays vertically (row wise) - Equivalent to `np.concatenate((arr1, arr2), axis=0)`.
`np.hstack((arr1, arr2))`	Stack arrays horizontally (column wise) - Equivalent to `np.concatenate((arr1, arr2), axis=1)`.

Accessing & Operations

Indexing, Slicing & Subsetting

`arr[5]`	Returns the element at index 5 (for 1D array).
`arr[2,5]`	Returns the element at row index 2 and column index 5 (for 2D array).
`arr[1] = 4`	Assigns the value 4 to the element at index 1.
`arr[1,3] = 10`	Assigns the value 10 to the element at row index 1 and column index 3.
`arr[0:3]`	Returns a slice containing elements from index 0 up to (but not including) index 3. (For 2D: returns rows 0, 1, 2).
`arr[0:3, 4]`	Returns a slice containing elements from rows 0, 1, 2 at column 4.
`arr[:2]`	Returns a slice containing elements from the beginning up to (but not including) index 2. (For 2D: returns rows 0, 1).

`arr[:, 1]`	Returns a slice containing elements at index 1 from all rows (i.e., the second column).
	Example:
	`arr2d = np.array([[1,2,3],[4,5,6]])`
	`print(arr2d[:, 1]) # Output: [2 5]`

`arr < 5`	Returns a boolean array of the same shape as `arr`, where `True` indicates the element is less than 5.
`(arr1 < 3) & (arr2 > 5)`	Returns a boolean array based on combined conditions using logical operators (`&` for AND, `|` for OR, `~` for NOT).
`~arr`	Inverts a boolean array.
`arr[arr < 5]`	Returns a 1D array containing only the elements from `arr` where the corresponding boolean array (generated by `arr < 5`) is `True`.

Element-wise Math

`np.add(arr1, arr2)` or `arr1 + arr2`	Element-wise addition of `arr1` and `arr2`.
`np.subtract(arr1, arr2)` or `arr1 - arr2`	Element-wise subtraction of `arr2` from `arr1`.
`np.multiply(arr1, arr2)` or `arr1 * arr2`	Element-wise multiplication of `arr1` by `arr2`.
`np.divide(arr1, arr2)` or `arr1 / arr2`	Element-wise division of `arr1` by `arr2`. Returns `np.nan` for division by zero.
`np.power(arr1, arr2)` or `arr1 ** arr2`	Element-wise raise `arr1` to the power of `arr2`.
`np.sqrt(arr)`	Calculates the square root of each element in `arr`.
`np.sin(arr)`, `np.cos(arr)`, `np.tan(arr)`	Trigonometric functions applied element-wise.
`np.log(arr)`, `np.log10(arr)`	Natural logarithm and base-10 logarithm element-wise.
`np.abs(arr)`	Calculates the absolute value of each element in `arr`.
`np.ceil(arr)`	Rounds up to the nearest integer element-wise.
`np.floor(arr)`	Rounds down to the nearest integer element-wise.
`np.round(arr)`	Rounds to the nearest integer element-wise.

Scalar Operations

`np.add(arr, 1)` or `arr + 1`	Adds 1 to each element in `arr`.
`np.subtract(arr, 2)` or `arr - 2`	Subtracts 2 from each element in `arr`.
`np.multiply(arr, 3)` or `arr * 3`	Multiplies each element in `arr` by 3.
`np.divide(arr, 4)` or `arr / 4`	Divides each element in `arr` by 4.
`np.power(arr, 5)` or `arr ** 5`	Raises each element in `arr` to the power of 5.

Statistics

`np.mean(arr)` or `arr.mean()`	Returns the mean (average) of all elements in `arr`.

`np.mean(arr, axis=0)`	Returns the mean along a specific axis (e.g., column-wise for axis=0).
	Example (2D array):
	`arr = np.array([[1,2],[3,4]])`
	`np.mean(arr, axis=0) # Output: [2. 3.] (mean of each column)`
	`np.mean(arr, axis=1) # Output: [1.5 3.5] (mean of each row)`

`arr.sum()` or `np.sum(arr)`	Returns the sum of all elements in `arr`.
`arr.sum(axis=0)`	Returns the sum along a specific axis.
`arr.min()` or `np.min(arr)`	Returns the minimum value in `arr`.
`arr.min(axis=0)`	Returns the minimum value along a specific axis.
`arr.max()` or `np.max(arr)`	Returns the maximum value in `arr`.
`arr.max(axis=0)`	Returns the maximum value along a specific axis.
`np.var(arr)`	Returns the variance of `arr`.
`np.std(arr)`	Returns the standard deviation of `arr`.
`np.std(arr, axis=1)`	Returns the standard deviation along a specific axis.
`arr.corrcoef()`	Returns the correlation coefficient matrix of `arr`.
`np.median(arr)`	Calculates the median of `arr`.

Comparison

`np.array_equal(arr1, arr2)`	Returns `True` if `arr1` and `arr2` have the same elements and shape, `False` otherwise.
`arr1 > arr2`, `arr1 < arr2`, `arr1 >= arr2`, `arr1 <= arr2`, `arr1 == arr2`, `arr1 != arr2`	Element-wise comparison operators. Return boolean arrays.