NumPy Cheat Sheet

NumPy is a fast, powerful library for numerical computing with fixed-type multidimensional arrays, enabling high-performance operations and forming the core of tools like Pandas and Matplotlib.

Installation

```
pip install numpy
```

Import

```
import numpy as np
```

Array Creation

```
np.array([1, 2, 3])                          # 1D array
np.array([[1, 2], [3, 4]])                   # 2D array
np.zeros((2, 3))                             # 2x3 zero matrix
np.ones((2, 3))                              # 2x3 ones matrix
np.empty((2, 3))                             # 2x3 uninitialized
np.arange(0, 10, 2)                          # 0 to 8 step 2
np.linspace(0, 1, 5)                         # 5 values 0 to 1
np.identity(3)                               # 3x3 identity matrix
np.random.random((3, 3))                     # 3x3 random matrix
np.pad(np.ones((2,2)), 1, constant_values=0)# Padding with zeros
```

Array Properties

```
a.shape                                      # (rows, cols)
a.ndim                                       # Number of
        dimensions
a.size                                       # Total elements
a.dtype                                      # Data type
a.itemsize                                   # Bytes per element
```

Reshape & Transpose

```
a.reshape(2, 3)                    # Reshape array
a.T                                # Transpose
```

Indexing & Slicing

```
a[1, 2]                            # Element at row 1,
        col 2
a[:, 1]                            # All rows, col 1
a[1, :]                            # Row 1, all cols
a[1:3, 0:2]                        # Subarray
a[::-1]                            # Reverse array
np.where(a != 0)                   # Non-zero indices
```

Arithmetic Operations

```
a + b                              # Element-wise
        addition
a - b                              # Subtraction
a * b                              # Multiplication
a / b                              # Division
a ** 2                             # Power
np.dot(a, b)                       # Dot product
np.matmul(a, b)                    # Matrix
        multiplication
```

Statistical Operations

```
a.min()                            # Minimum value
a.max()                            # Maximum value
a.sum()                            # Sum of elements
a.mean()                           # Mean
a.std()                            # Standard deviation
np.add.reduce(a)                  # Sum via reduce
np.max(a, axis=1)                  # Row-wise max
np.argmax(a)                       # Index of max
np.argmin(a)                       # Index of min
```

Logical & Comparison

```
a > 5                                   # Element-wise
        comparison
np.any(a > 5)                           # Any > 5?
np.all(a > 5)                           # All > 5?
np.where(a > 0, 1, 0)                   # Conditional select
np.intersect1d(a, b)                    # Common elements
np.union1d(a, b)                        # All unique elements
np.allclose(a, b)                       # Approximately equal
np.array_equal(a, b)                    # Exactly equal
```

Bitwise & Special

```
2 << a >> 2                             # Bit shifting
1j * a                                  # Complex
        multiplication
a // 2                                  # Floor division
a % 2                                   # Modulo
np.sqrt(4)                              # Square root
np.emath.sqrt(-1)                       # Complex sqrt
```

Tiling & Repeating

```
np.tile([1,2], 3)                       # Repeat array: [1 2
        1 2 1 2]
np.repeat([1,2], 2)
        # Repeat elements: [1 1 2 2]

# 2D Tiling
np.tile([[1,2],[3,4]], (2,3))
# Output:
# [[1 2 1 2 1 2]
#  [3 4 3 4 3 4]
#  [1 2 1 2 1 2]
#  [3 4 3 4 3 4]]
```

Normalization

```
(a - np.mean(a)) / np.std(a)            # Normalize
```

Data Type Conversion

```
a.astype(np.uint8)                                  # Cast type
```

Custom Data Type

```
color = np.dtype([('r',np.ubyte),('g',np.ubyte),('b',np.ubyte),
        ('a',np.ubyte)])
```

Dates with NumPy

```
np.datetime64('today')                              # Today's date
np.arange('2020-01', '2020-02', dtype='datetime64[D]')
        # All Jan dates
```

Mesh Grid

```
x, y = np.meshgrid(np.linspace(0,1,3), np.linspace(0,1,3))
```

Checkerboard Pattern

```
Z = np.zeros((8,8), dtype=int); Z[1::2,::2]=1; Z[::2,1::2]=1
```

Matrix Operations

```
np.linalg.det(a)                                    # Determinant
np.linalg.inv(a)                                    # Inverse
np.linalg.eig(a)                                    # Eigenvalues/vectors
```

Fancy Indexing

```
Z[:, [0, -1]] = 0                                   # Set first/last col
        to 0
Z[[0, -1], :] = 0                                   # Set first/last row
        to 0
```

Diagonal Matrix

```
np.diag(1+np.arange(4), k=-1)                # Below diagonal
```

Unravel Index

```
np.unravel_index(99, (6,7,8))                # Convert flat index
        to coords
```

Linearly Spaced Vector

```
np.linspace(0,1,12)
        # 12 values from 0 to 1
```

Sorting

```
np.sort(a)                                   # Return sorted array
a.sort()                                     # Sort in-place
```

Stacking Arrays

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

Sum Differences

```
sum(range(5), -1)                            # Python: 9
np.sum(range(5), -1)                         # NumPy: 10
```

String Formatting

Old Style

```
"I have %d apples" % 5                       # Integer
"Price: %.2f" % 3.1415                       # Float 2 decimals
"Hex: %x" % 255                              # Hexadecimal
```

f-Strings (Modern)

```
f"I have {5} apples"                    # Integer
f"Price: {3.1415:.2f}"                  # Float 2 decimals
f"Hex: {255:x}"                         # Hexadecimal
```

Help & Info

```
np.info('add')                          # Info on function
```

Python For Loops – Quick Cheatsheet

Loop Through Items

```
for item in iterable:
    print(item)
```

Loop Through Indexes

```
for i in range(len(iterable)):
    print(i, iterable[i])
```

Loop Index + Item

```
for i, item in enumerate(iterable):
    print(i, item)
```

2D NumPy Array with Index

```
import numpy as np
for (i, j), val in np.ndenumerate(arr):
    print((i, j), val)
```

Summary

Task	Best Method
Items only	`for item in iterable`
Indexes only	`for i in range(len(...))`
Index + item	`enumerate()`
2D array index	`np.ndenumerate()`

Summary: `np.unravel_index(np.argmin(D), A.shape)`

Purpose:

Find the **multi-dimensional index** of the **minimum value** in array `D` (with shape of `A`).

Breakdown:

- `np.argmin(D)` → Flat index of min value in `D`
- `np.unravel_index(..., A.shape)` → Converts flat index to `(row, col)` in shape of `A`

Example:

```
import numpy as np
A = np.array([[10, 3, 5],
              [7, 2, 8]])
D = A.copy()
```

```
idx = np.unravel_index(np.argmin(D), A.shape)
print("Min Value:", A[idx])     # 2
print("Index:", idx)            # (1, 1)
```

Result:

`A[1, 1] = 2` → Smallest value with index `(1, 1)` # Summary: `np.put(Z, np.random.choice(range(n*n), p, replace=False), 1)`

Purpose:

Randomly set **p unique elements** in array `Z` to 1.

Breakdown:

- `range(n*n)` → Flat indices of `Z` (if Z is `n x n`)
- `np.random.choice(..., p, replace=False)` → Pick p unique random indices
- `np.put(Z, indices, 1)` → Set `Z[indices] = 1`

Example:

```
import numpy as np
n, p = 3, 4
Z = np.zeros((n, n), dtype=int)
np.put(Z, np.random.choice(range(n*n), p, replace=False), 1)
print(Z)
```

Result:

4 random positions in `Z` are set to 1 (no repeats).

Summary: `np.isnan(Z).all(axis=0)`

Purpose:

Check which **columns** in array `Z` are **entirely NaN**.

Breakdown:

- `np.isnan(Z)` → Boolean array: `True` where `Z` is NaN
- `.all(axis=0)` → For each column, `True` if **all values are NaN**

Example:

```
import numpy as np
Z = np.array([[np.nan, 1.0],
              [np.nan, np.nan],
              [np.nan, 3.0]])

result = np.isnan(Z).all(axis=0)
print(result)  # [ True False ]
```

Result:

- Column 0 → All NaN → `True`
- Column 1 → Not all NaN → `False`

Python Loops – NumPy Style

Task	Best Practice
Items only	`for item in iterable`
Indexes only	`for i in range(len(...))`
Index + item	`enumerate(iterable)`
2D index + val	`np.ndenumerate(array)`

2D Loop Example

```
for (i, j), val in np.ndenumerate(arr):
    print((i, j), val)
```

One-Liner Examples

Function	One-Liner Example	Description
`np.argmin()`	`idx = np.argmin(a)`	Get index of the min value in flattened array a
`np.unravel_index()`	`multi_idx = np.unravel_index(idx, a.shape)`	Convert flat idx to (row, col)
`np.put()`	`np.put(a, [1, 3], 9)`	Set positions 1 and 3 in flattened a to 9
Set min to 0	`np.put(a, [np.argmin(a)], 0)`	Replace min value in a with 0
Random set to 1	`np.put(a, np.random.choice(a.size, 4, replace=False), 1)`	Set 4 random positions in a to 1

---

np.isnan(Z).all(axis=0)

Purpose: Check which columns in array `Z` are entirely NaN.

```python result = np.isnan(Z).all(axis=0) print(result) # Example: [ True False ]

**Note**: Use # comments to understand code snippets. Many operations are **broadcastable** and **vectorized** for performance.
```