

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

<code>a.reshape(2, 3)</code>	<code># Reshape array</code>
<code>a.T</code>	<code># Transpose</code>

Indexing & Slicing

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

Arithmetic Operations

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

Statistical Operations

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

Logical & Comparison

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

Bitwise & Special

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

Tiling & Repeating

<code>np.tile([1,2], 3)</code>	<code># Repeat array: [1 2</code>
<code>1 2 1 2]</code>	
<code>np.repeat([1,2], 2)</code>	
<code># Repeat elements: [1 1 2 2]</code>	
<code># 2D Tiling</code>	
<code>np.tile([[1,2],[3,4]], (2,3))</code>	
<code># Output:</code>	
<code># [[1 2 1 2 1 2]</code>	
<code># [3 4 3 4 3 4]</code>	
<code># [1 2 1 2 1 2]</code>	
<code># [3 4 3 4 3 4]]</code>	

Normalization

<code>(a - np.mean(a)) / np.std(a)</code>	<code># Normalize</code>
---	--------------------------

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[:,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
```

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