NumPy Reference Cheatsheet

1 Introduction

NumPy is the core numerical computing library in Python. It provides powerful tools for array operations, linear algebra, and random sampling.

2 Installation

To install NumPy, use pip:

```
pip install numpy
```

3 Importing NumPy

```
import numpy as np
```

4 Array Creation and Initialization

```
import numpy as np

# Float array
a = np.array([1, 2, 3])

# Integer array
ai = np.array([1, 2, 3], dtype=int)

# Zero matrix
z = np.zeros((2, 3))

# Ones matrix
z = np.ones((2, 3))

# Identity matrix
I = np.identity(3)
```

5 Array Attributes

```
a.shape      # Shape of array
a.ndim       # Number of dimensions
a.dtype      # Data type
a.size       # Total number of elements
```

6 Indexing, Slicing and Swapping

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

# Access by index
print(a[0])      # 10

# Slice a subarray
print(a[1:4])    # [20 30 40]

# 2D array slicing
b = np.array([[1, 2, 3], [4, 5, 6]])
print(b[0, 1])  # 2
print(b[:, 1])  # [2 5]
print(b[1, :])  # [4 5 6]

# Row swapping in 2D arrays
b[0], b[1] = b[1].copy(), b[0].copy()
print(b)
# Output:
# [[4 5 6]
#  [1 2 3]]
```

7 Looping with `range()` and `reversed(range())`

`range(stop)`

```
for i in range(5):
    print(i)
# Output: 0, 1, 2, 3, 4
```

`range(start, stop, step)`

```
for i in range(2, 10, 2):
    print(i)
# Output: 2, 4, 6, 8
```

`reversed(range(n))`

To loop backwards, use `reversed(range(n))`:

```
for i in reversed(range(3)):
    print(i)
# Output: 2, 1, 0
```

8 Linear Algebra Tools

```
# Element-wise operations
x = np.array([1, 2, 3])
y = np.array([4, 5, 6])

print(x + y)        # [5 7 9]
print(x * y)        # [4 10 18]

# Define a matrix for the next examples
A = np.array([[2, 1],
              [1, 3]])

# Matrix-vector multiplication using @
v = np.array([1, 0])
result = A @ v      # [2 1]

# Dot (inner) product of two vectors
u = np.array([1, 2])
w = np.array([3, 4])
dot = u @ w         # 11

# Transpose of a matrix
A_t = np.transpose(A)
```

Linear algebra submodule numpy.linalg

```
# Solving linear systems: solve Ax = b
A = np.array([[2, 1], [1, 3]])
b = np.array([1, 2])
x = np.linalg.solve(A, b)

# Euclidean norm (L2 length) of a vector
v = np.array([3, 4])
length = np.linalg.norm(v)  # 5.0

# Inverse of a matrix (if A is square)
B = np.linalg.inv(A)
```

9 Numerical Rounding and Comparison

```
x = np.array([1.499999, 2.500001])

# Round elements to nearest integer
xr = np.round(x)  # [1. 3.]

# Test approximate equality (within tolerance)
np.allclose(x, xr)  # False in this case
```

10 Generating Random Vectors

```
n = 3

# Generating random vectors using np.random.rand
v = np.random.rand(n)  # Random vector in [0, 1)^n

# Using np.random.uniform for more control over range
v_uniform = np.random.uniform(low=-10.0, high=5.0, size=n)  #
    Random vector in [-10, 5)

# Using np.random.randint for integer-valued vectors
v_int = np.random.randint(low=-10, high=5, size=n)  # Random
    integers in [-10, 5)

# Sampling using uniform distribution in a loop
samples_uniform = [np.random.uniform(-1.0, 1.0, size=n) for _ in
    range(num_samples)]

# Sampling integer vectors in a loop
samples_int = [np.random.randint(0, 10, size=n) for _ in range(
    num_samples)]
```

11 Other Useful NumPy Tools

- `np.copy(x)` — Make a safe copy
- `np.mean(x)` — Average of elements
- `np.sum(x)` — Total sum
- `np.max(x)` — Maximum element

12 Full documentation

This reference introduces the most commonly used NumPy features in computational linear algebra and random vector generation. For further learning, consult the official documentation: `https://numpy.org/doc/`