Comprehensive NumPy Cheat Sheet

This cheat sheet covers the essential and advanced features of NumPy, a powerful library for numerical computing in Python.

1. Introduction

1.1 What is NumPy?

NumPy is a fundamental package for scientific computing in Python. It provides support for large multidimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.

1.2 Installing NumPy

You can install NumPy using pip:

```
pip install numpy
```

Or, if you are using Anaconda:

```
conda install numpy
```

1.3 Importing NumPy

To use NumPy, you need to import it in your script:

In [1]:
```
import numpy as np
# Now you can use NumPy functions with the np prefix
array = np.array([1, 2, 3, 4])
print(array)  # Output: [1 2 3 4]
```

```
[1 2 3 4]
```

2. NumPy Arrays

2.1 Creating Arrays

NumPy arrays can be created in several ways.

In [2]:
```
# Creating arrays
import numpy as np

# From a Python list
array_from_list = np.array([1, 2, 3, 4])
print(array_from_list)  # Output: [1 2 3 4]

# Using NumPy functions
array_zeros = np.zeros((3, 3))
print(array_zeros)
# Output:
# [[0. 0. 0.]
#  [0. 0. 0.]
#  [0. 0. 0.]]

array_ones = np.ones((2, 2))
print(array_ones)
# Output:
# [[1. 1.]
#  [1. 1.]]

array_arange = np.arange(10)
print(array_arange)  # Output: [0 1 2 3 4 5 6 7 8 9]
```

```
[1 2 3 4]
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
[[1. 1.]
 [1. 1.]]
[0 1 2 3 4 5 6 7 8 9]
```

2.2 Accessing Array Elements

You can access elements in a NumPy array using indices.

In [3]:
```
# Accessing elements
array = np.array([1, 2, 3, 4, 5])
print(array[0])  # Output: 1
print(array[4])  # Output: 5

# For multidimensional arrays
matrix = np.array([[1, 2], [3, 4], [5, 6]])
print(matrix[0, 1])  # Output: 2
print(matrix[2, 0])  # Output: 5
```

```
1
5
2
5
```

2.3 Array Slicing

Slicing in NumPy arrays works similarly to slicing in Python lists.

In [4]:
```
# Array slicing
array = np.array([1, 2, 3, 4, 5, 6])
print(array[1:4])  # Output: [2 3 4]
print(array[:3])   # Output: [1 2 3]
print(array[3:])   # Output: [4 5 6]
print(array[::2])  # Output: [1 3 5]
```

```
[2 3 4]
[1 2 3]
[4 5 6]
[1 3 5]
```

2.4 Array Shape and Reshape

You can check and modify the shape of an array using the `shape` and `reshape` methods.

In [5]:
```
# Array shape and reshape
array = np.array([[1, 2, 3], [4, 5, 6]])
print(array.shape)  # Output: (2, 3)

# Reshaping the array
reshaped_array = array.reshape((3, 2))
print(reshaped_array)
# Output:
# [[1 2]
#  [3 4]
#  [5 6]]
```

```
(2, 3)
[[1 2]
 [3 4]
 [5 6]]
```

3. Array Operations

3.1 Arithmetic Operations

NumPy allows you to perform element-wise arithmetic operations on arrays.

In [6]:
```
# Arithmetic operations
array1 = np.array([1, 2, 3, 4])
array2 = np.array([5, 6, 7, 8])

print(array1 + array2)  # Output: [ 6  8 10 12]
print(array1 - array2)  # Output: [-4 -4 -4 -4]
print(array1 * array2)  # Output: [ 5 12 21 32]
print(array1 / array2)  # Output: [0.2 0.33333333 0.42857143 0.5]
```

```
[ 6  8 10 12]
[-4 -4 -4 -4]
[ 5 12 21 32]
[0.2        0.33333333 0.42857143 0.5       ]
```

3.2 Broadcasting

Broadcasting allows you to perform arithmetic operations on arrays of different shapes.

In [7]:
```
# Broadcasting example
array1 = np.array([1, 2, 3, 4])
array2 = np.array([2])

print(array1 * array2)  # Output: [2 4 6 8]
```

```
[2 4 6 8]
```

3.3 Universal Functions (ufuncs)

NumPy provides universal functions (ufuncs) which are functions that operate element-wise on an array.

In [8]:
```
# Universal functions
array = np.array([1, 2, 3, 4])
print(np.sqrt(array))  # Output: [1.  1.41421356 1.73205081 2. ]
print(np.exp(array))   # Output: [ 2.71828183  7.3890561  20.085536
print(np.sin(array))   # Output: [ 0.84147098  0.90929743  0.141120
```

```
[1.         1.41421356 1.73205081 2.        ]
[ 2.71828183  7.3890561  20.08553692 54.59815003]
[ 0.84147098  0.90929743  0.14112001 -0.7568025 ]
```

4. Array Manipulation

4.1 Joining Arrays

You can join multiple arrays into one using functions like `np.concatenate`, `np.vstack`, and `np.hstack`.

In [9]:
```
# Joining arrays
array1 = np.array([1, 2, 3])
array2 = np.array([4, 5, 6])

# Using concatenate
joined_array = np.concatenate((array1, array2))
print(joined_array)  # Output: [1 2 3 4 5 6]

# Using vstack
stacked_array_v = np.vstack((array1, array2))
print(stacked_array_v)
# Output:
# [[1 2 3]
#  [4 5 6]]

# Using hstack
stacked_array_h = np.hstack((array1, array2))
print(stacked_array_h)  # Output: [1 2 3 4 5 6]
```

```
[1 2 3 4 5 6]
[[1 2 3]
 [4 5 6]]
[1 2 3 4 5 6]
```

4.2 Splitting Arrays

You can split an array into multiple arrays using functions like `np.split`, `np.vsplit`, and `np.hsplit`.

In [10]:
```
# Splitting arrays
array = np.array([1, 2, 3, 4, 5, 6])

# Using split
split_array = np.split(array, 3)
print(split_array)
# Output:
# [array([1, 2]), array([3, 4]), array([5, 6])]
```

```
[array([1, 2]), array([3, 4]), array([5, 6])]
```

4.3 Sorting Arrays

You can sort the elements of an array in ascending or descending order using the `np.sort` function.

In [11]:
```
# Sorting arrays
array = np.array([3, 1, 2, 5, 4])
sorted_array = np.sort(array)
print(sorted_array)  # Output: [1 2 3 4 5]
```

```
[1 2 3 4 5]
```

4.4 Copying Arrays

You can create copies of arrays using the `copy` method to avoid modifying the original array.

In [12]:
```
# Copying arrays
array = np.array([1, 2, 3, 4])
copied_array = array.copy()
copied_array[0] = 99
print(array)        # Output: [1 2 3 4]
print(copied_array)  # Output: [99  2  3  4]
```

```
[1 2 3 4]
[99  2  3  4]
```

5. Statistical Operations

5.1 Basic Statistical Functions

NumPy provides a variety of statistical functions for performing computations on arrays.

In [13]:
```
# Basic statistical functions
array = np.array([1, 2, 3, 4, 5, 6])
print(np.mean(array))  # Output: 3.5
print(np.median(array))  # Output: 3.5
print(np.std(array))  # Output: 1.707825127659933
```

```
3.5
3.5
1.707825127659933
```

5.2 Aggregation Functions

Aggregation functions allow you to summarize data.

In [14]:
```
# Aggregation functions
print(np.sum(array))  # Output: 21
print(np.prod(array))  # Output: 720
print(np.cumsum(array))  # Output: [ 1  3  6 10 15 21]
print(np.cumprod(array))  # Output: [  1   2   6  24 120 720]
```

```
21
720
[ 1  3  6 10 15 21]
[  1   2   6  24 120 720]
```

5.3 Statistical Methods (mean, median, std)

These methods are used to compute the mean, median, and standard deviation of an array.

In [16]:
```
# Statistical methods
print(array.mean())  # Output: 3.5
print(array.std())  # Output: 1.707825127659933
```

```
3.5
1.707825127659933
```

6. Linear Algebra

6.1 Matrix Multiplication

NumPy provides functions for matrix multiplication.

In [17]:
```
# Matrix multiplication
matrix1 = np.array([[1, 2], [3, 4]])
matrix2 = np.array([[5, 6], [7, 8]])
product = np.dot(matrix1, matrix2)
print(product)
# Output:
# [[19 22]
#  [43 50]]
```

```
[[19 22]
 [43 50]]
```

6.2 Determinant and Inverse

You can compute the determinant and inverse of a matrix using NumPy.

In [18]:
```
# Determinant and inverse
matrix = np.array([[1, 2], [3, 4]])
det = np.linalg.det(matrix)
inv = np.linalg.inv(matrix)
print(det)  # Output: -2.0000000000000004
print(inv)
# Output:
# [[-2.   1. ]
#  [ 1.5 -0.5]]
```

```
-2.0000000000000004
[[-2.   1. ]
 [ 1.5 -0.5]]
```

6.3 Eigenvalues and Eigenvectors

NumPy can also compute eigenvalues and eigenvectors.

In [19]:
```
# Eigenvalues and eigenvectors
eigenvalues, eigenvectors = np.linalg.eig(matrix)
print(eigenvalues)  # Output: [-0.37228132  5.37228132]
print(eigenvectors)
# Output:
# [[-0.82456484 -0.41597356]
#  [ 0.56576746 -0.90937671]]
```

```
[-0.37228132  5.37228132]
[[-0.82456484 -0.41597356]
 [ 0.56576746 -0.90937671]]
```

7. NumPy Random Module

7.1 Generating Random Numbers

NumPy's random module can be used to generate random numbers.

In [20]:
```
# Generating random numbers
random_array = np.random.rand(3, 3)
print(random_array)
# Output:
# [[0.5488135  0.71518937 0.60276338]
#  [0.54488318 0.4236548  0.64589411]
#  [0.43758721 0.891773   0.96366276]]
```

```
[[0.21177229 0.7636598  0.2815125 ]
 [0.87611048 0.4066717  0.34164311]
 [0.50710694 0.47424093 0.56981746]]
```

7.2 Random Distributions

NumPy's random module can also generate random samples from various distributions.

In [21]:
```
# Random distributions
normal_dist = np.random.randn(3, 3)
print(normal_dist)
# Output:
# [[ 1.76405235  0.40015721  0.97873798]
#  [ 2.2408932   1.86755799 -0.97727788]
#  [ 0.95008842 -0.15135721 -0.10321885]]

poisson_dist = np.random.poisson(5, 10)
print(poisson_dist)  # Output: [ 6  5  3  3  6  9  4  5  5  7]
```

```
[[ 2.01148613 -0.14233857 -0.39404202]
 [ 1.02521821  0.13756554  0.49950549]
 [ 0.25065534 -0.43861953 -1.07299838]]
[7 2 5 6 6 1 4 5 4 3]
```

8. Advanced Topics

8.1 Fancy Indexing

Fancy indexing allows NumPy arrays to be indexed with arrays or sequences.

In [22]:
```
# Fancy indexing
array = np.array([1, 2, 3, 4, 5, 6])
indices = [0, 2, 4]
fancy_indexed_array = array[indices]
print(fancy_indexed_array)  # Output: [1 3 5]
```

```
[1 3 5]
```

8.2 Vectorization

Vectorization is the process of performing operations on entire arrays rather than individual elements.

In [23]:
```
# Vectorization example
array = np.array([1, 2, 3, 4, 5, 6])
vectorized_array = array * 2
print(vectorized_array)  # Output: [ 2  4  6  8 10 12]
```

```
[ 2  4  6  8 10 12]
```

8.3 Memory Layout

NumPy arrays can have different memory layouts (C-contiguous or Fortran-contiguous). You can check the memory layout using the `flags` attribute.

In [24]:
```
# Memory layout
array = np.array([[1, 2, 3], [4, 5, 6]], order='C')
print(array.flags)

array_fortran = np.array([[1, 2, 3], [4, 5, 6]], order='F')
print(array_fortran.flags)
```

```
  C_CONTIGUOUS : True
  F_CONTIGUOUS : False
  OWNDATA : True
  WRITEABLE : True
  ALIGNED : True
  WRITEBACKIFCOPY : False
  UPDATEIFCOPY : False

  C_CONTIGUOUS : False
  F_CONTIGUOUS : True
  OWNDATA : True
  WRITEABLE : True
  ALIGNED : True
  WRITEBACKIFCOPY : False
  UPDATEIFCOPY : False
```

9. Practical Applications

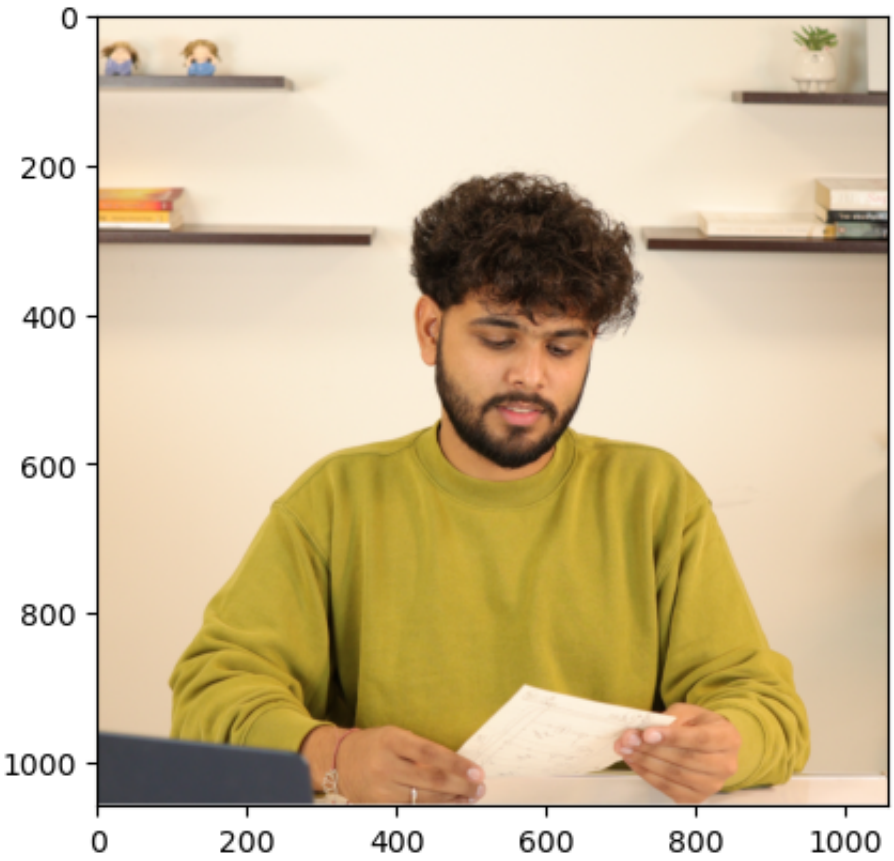
9.1 Image Processing

NumPy can be used for basic image processing tasks such as loading, manipulating, and saving images.

In [33]:
```
# Image processing example
import imageio
import matplotlib.pyplot as plt

# Load an image
image = imageio.imread('img.jpg')
plt.imshow(image, cmap='gray')
plt.show()
```

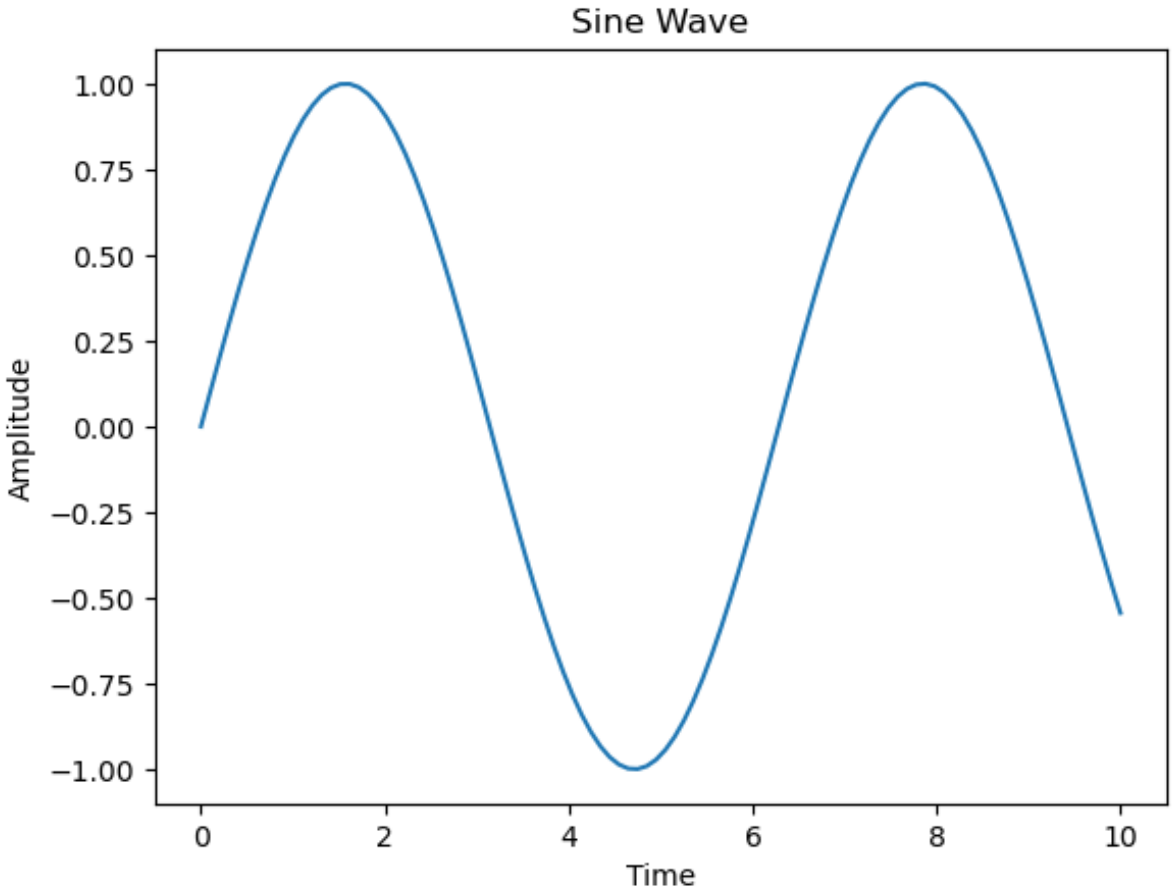
```
/var/folders/rq/lwddjvtn5n9gjw05hwrg3hp80000gp/T/ipykernel_9300/19
34184168.py:6: DeprecationWarning: Starting with ImageIO v3 the be
havior of this function will switch to that of iio.v3.imread. To k
eep the current behavior (and make this warning dissapear) use `im
port imageio.v2 as imageio` or call `imageio.v2.imread` directly.
  image = imageio.imread('img.jpg')
```



9.2 Numerical Simulations

NumPy is often used in scientific computing and simulations.

In [34]:
```
# Numerical simulation example
time = np.linspace(0, 10, 100)
amplitude = np.sin(time)
plt.plot(time, amplitude)
plt.title('Sine Wave')
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.show()
```



10. Conclusion

10.1 Summary

NumPy is a powerful library for numerical computing, providing support for large multidimensional arrays and a wide range of mathematical functions.

10.2 Further Reading and Resources

For more information, check out the official NumPy documentation (https://numpy.org/doc/). Additional resources:

- NumPy User Guide (https://numpy.org/doc/stable/user/index.html)
- NumPy Reference (https://numpy.org/doc/stable/reference/index.html)