NumPy

NumPy is a popular Python library used for scientific computing. It provides a high-performance multi-dimensional array object, as well as tools for working with these arrays. NumPy is especially useful for numerical and scientific computations, such as linear algebra, statistical analysis, and machine learning.

NumPy is an essential component of the scientific Python ecosystem, and it is used extensively in many popular libraries, including SciPy, pandas, scikit-learn, and TensorFlow.

In this tutorial, we will cover the basics of using NumPy, including:

1. Installing NumPy
2. Creating NumPy arrays
3. Array indexing and slicing
4. Array manipulation
5. Mathematical operations
6. Broadcasting
7. Linear algebra operations

Installing NumPy

NumPy is available through the Python Package Index (PyPI), which can be installed using the following command in the terminal:

```
pip install numpy
```

Importing NumPy

The NumPy library is imported using the following command:

```python
import numpy
```

This imports the NumPy library and makes it available for use in our current Python session. We can then access NumPy functions using the `numpy` prefix, such as `numpy.array()`. However, it is more common to import NumPy using the alias `np`:

```python
import numpy as np
```

Using this alias, we can instead access NumPy functions using the `np` prefix, such as `np.array()`. This is the convention used in this tutorial.

Creating NumPy Arrays

The most fundamental object in NumPy is the ndarray (n-dimensional array) object. An ndarray is a collection of items of the same type, arranged in a rectangular shape. It can have any number of dimensions, although most commonly we will work with 1D, 2D, and 3D arrays. To create an NumPy array, we can use the `np.array()` function. For example:

```
import numpy as np

# Create a 1D array
a = np.array([1, 2, 3])
print(a)
```

```
# Create a 2D array
b = np.array([[1, 2, 3], [4, 5, 6]])
print(b)
```

```
# Create a 3D array
c = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
print(c)
```

Array Indexing

NumPy arrays can be indexed and sliced in the same way as regular Python lists. To access a single element of a NumPy array, use the syntax `array[index]`. To access a slice of a NumPy array, use the syntax `array[start:stop:step]` where `start` is the index of the first element in the slice, `stop` is the index of the last element in the slice, and `step` is the number of elements to skip between each element in the slice.

```
import numpy as np

a = np.array([1, 2, 3, 4, 5])
```

```
# Get the first element
print(a[0])
```

```
# Get the last element
print(a[-1])
```

```
# Get the first three elements
print(a[:3])
```

```
# Get every other element starting from the second element
print(a[1::2])
```

Indexing and slicing higher-dimensional arrays is done by passing multiple indices separated by commas. For 2D arrays, also referred to as matrices, the syntax is `array[row, column]`. For 3D arrays, the syntax is `array[matrix, row, column]`. For example:

```
import numpy as np

a = np.array([[1, 2, 3], [4, 5, 6]])
b = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
```

```
# Get the element from the second row and first column from the first array.
print(a[1, 0])
```

```
# Get the first row of the first array.
print(a[0])
```

```
# Get the last column of the first array.
print(a[:, -1])
```

```
# Get the element from the second matrix, first row and first column from the second array
print(b[1, 0, 0])
```

```
# Get the second matrix of the second array.
print(b[1])
```

```
# Get the first row from each matrix of the second array.
print(b[:, 0])
```

As you can see, there are multiple options to access the elements in your arrays and to slice them. It can be confusing at first, but with practice you will get used to it. For each higher dimension, we add another index separated by commas. This follows the formula `array[dim_n, dim_n-1, ..., dim_0]` where n is the number of dimensions.

Array Manipulation

NumPy arrays can be manipulated in a variety of ways. For example, we can reshape an array using the `array.reshape()` method. We can also transpose an array using the `array.transpose()` attribute. For example:

```python
import numpy as np

a = np.array([[1, 2, 3], [4, 5, 6]])
```

```python
# Reshape the array to have three rows and two columns
b = np.reshape(a, (3, 2))
print(b)
```

```python
# Transpose the array to have two rows and three columns
c = np.transpose(a)
print(c)
```

The transpose method can be also accessed in shorthand using the `array.T` attribute. For example:

```python
import numpy as np

a = np.array([[1, 2, 3], [4, 5, 6]])

# Transpose the array to have two rows and three columns
b = a.T
print(b)
```

Other useful methods include `array.flatten()`, which flattens the array into a 1D array, and `array.ravel()`, which also flattens the array into a 1D array, but returns a view of the original array instead of a copy. For example:

```python
import numpy as np

a = np.array([[1, 2, 3], [4, 5, 6]])
```

```python
# Flatten the array and return a copy
b = a.flatten()
print(b)
```

```
# Flatten the array and return a reference
c = a.ravel()
print(c)
```

```
# Alter the array and see the changes in b and c.
a[1, 1] = 10

print(b, c)
```

This tutorial only covers a small subset of the many useful operations that NumPy provides. For a more comprehensive list of NumPy operations, see the NumPy Reference.

Mathematical Operations

NumPy provides a wide range of mathematical functions and operations that can be applied to arrays. Some of the most commonly used functions include `np.sum()`, `np.mean()`, `np.std()`, `np.min()`, and `np.max()`. For example:

```
import numpy as np

a = np.array([1, 2, 3, 4, 5])
```

```
# Compute the sum of the array
print(np.sum(a))
```

```
# Compute the mean of the array
print(np.mean(a))
```

```
# Compute the standard deviation of the array
print(np.std(a))
```

```
# Find the minimum value in the array
print(np.min(a))
```

```
# Find the maximum value in the array
print(np.max(a))
```

Broadcasting

NumPy's broadcasting capability allows mathematical operations to be performed on arrays of different shapes and sizes. For example, you can add a scalar to an array, or add two arrays with different shapes. For example:

```
import numpy as np

a = np.array([[1, 2, 3], [4, 5, 6]])
b = np.array([10, 20, 30])
```

```
# Add the scalar 1 to every element of the array a
print(a + 1)
```

```
# Add the array b to each row of the array a
print(a + b)
```

```
# Multiply every element of the array a by 2
print(a * 2)
```

```
# Multiply each row of the array a by the array b
print(a * b)
```

Broadcasting operations will fail if the arrays do not have compatible shapes. For example, the following code will fail:

```
import numpy as np

a = np.array([[1, 2, 3], [4, 5, 6]])
b = np.array([[10, 20, 30, 40, 50, 60]])

# Add array a to array b
print(a + b)
```

Once again, this can be confusing at first, but with practice you will get used to it.

Linear Algebra Operations

NumPy provides a range of functions for performing linear algebra operations, such as matrix multiplication, dot products, and eigenvalue decomposition. For example:

```python
import numpy as np

a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6], [7, 8]])
```

```python
# Multiply two matrices
print(np.dot(a, b))
```

```python
# Compute the eigenvalues and eigenvectors of a matrix
eigvals, eigvecs = np.linalg.eig(a)
print(eigvals)
print(eigvecs)
```

Exercises

Some of these exercises may use methods that we have not covered in this tutorial. If you are unfamiliar with a method, you can use the NumPy Reference to look up the method and learn more about it.

1. Create a NumPy array of integers from 1 to 20 and print its shape, size, and data type.
2. Generate a 5x5 NumPy array of random integers between 1 and 100.
3. Create a 4x4 identity matrix using NumPy.
4. Create two 3x3 NumPy arrays, A and B, with random integers from 1 to 10. Calculate and print their sum, difference, and element-wise product.
5. Generate a 1D NumPy array of 20 evenly spaced numbers between 0 and 10 using the `np.linspace()` function.
6. Create a 6x6 NumPy array with random integers from 1 to 50. Calculate the mean, median, and standard deviation of its elements.
7. Reshape a 1D NumPy array of 16 elements into a 4x4 2D array.
8. Create a 5x5 NumPy array with random integers from 1 to 25. Extract a 3x3 subarray from the center of the original array.
9. Given a NumPy array, replace all elements that are greater than a specific value, say 10, with 0.
10. Create a 10x10 NumPy array with random integers from 1 to 100. Normalize the array by subtracting the mean and dividing by the standard deviation.

11. Create two 1D NumPy arrays of equal length, A and B, with random integers from 1 to 20. Calculate the Euclidean distance between A and B.
12. Given a 1D NumPy array, find the top 5 maximum values and their indices.
13. Create a 3x3 NumPy array with random integers from 1 to 9. Calculate its determinant and eigenvalues using NumPy's linear algebra functions.
14. Create a 5x5 NumPy array with random integers from 1 to 25. Sort each row in ascending order.
15. Create two 2x2 NumPy arrays, A and B, with random integers from 1 to 5. Calculate and print their matrix multiplication and element-wise product.

Project: Matrix Manipulation

Create a Python program that uses the NumPy package to perform various operations on matrices. The program should allow the user to enter matrices and perform operations such as addition, subtraction, multiplication, and transpose.

Project Requirements

1. The program should use the NumPy package to define matrices and perform operations on them.
2. The program should handle errors appropriately, for example, when the user enters matrices with different dimensions.

Example Output

```
Matrix Manipulation

Enter matrix A:
1 2 3
4 5 6
7 8 9

Enter matrix B:
9 8 7
6 5 4
3 2 1

Matrix A:
```

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

Matrix B:
[[9 8 7]
 [6 5 4]
 [3 2 1]]

Matrix A + B:
[[10 10 10]
 [10 10 10]
 [10 10 10]]

Matrix A - B:
[[-8 -6 -4]
 [-2  0  2]
 [ 4  6  8]]

Matrix A * B:
[[30 24 18]
 [84 69 54]
 [138 114 90]]

Matrix A transpose:
[[1 4 7]
 [2 5 8]
 [3 6 9]]
```

Further Reading

Check out the following resources to learn more about the NumPy package in python:

- Documentation
  - NumPy Documentation
  - NumPy Reference
- Tutorials
  - NumPy Tutorial
  - W3Schools
  - Tutorials Point
  - DataCamp

    - DataQuest

Now we're delving into into the big wide world of external Python packages. This is the bread and butter of Python, and it's where you'll find most of the functionality you'll need to do anything useful. These packages are written by other people, and they're usually pretty good at what they do. The only downside is that you have to install them yourself, and you have to learn how to use them independently.

Whilst these tutorials will give you a good overview of how to use the packages, you'll have to do a bit of research to find out how to do specific things. This is a good thing, because it means you'll be able to find the answers to your own questions, and you'll be able to use the packages in ways that the tutorials don't cover.

Remember, the internet is your friend. If you're stuck, search engines are your best friend. If you can't find the answer to your question, try reaching out to your colleagues or your tutor.