Beginner's Guide to NumPy

Estimated Time : 10 Minutes

Objective:

In this reading, you'll learn:

- Basics of NumPy
- How to create NumPy arrays
- Array attributes and indexing
- Basic operations like addition and multiplication

What is NumPy?

NumPy, short for **Num**erical **Py**thon, is a fundamental library for numerical and scientific computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of high-level mathematical functions to operate on these arrays. NumPy serves as the foundation for many data science and machine learning libraries, making it an essential tool for data analysis and scientific research in Python.

Key aspects of NumPy in Python:

- **Efficient data structures**: NumPy introduces efficient array structures, which are faster and more memory-efficient than Python lists. This is crucial for handling large data sets.
- **Multi-dimensional arrays**: NumPy allows you to work with multi-dimensional arrays, enabling the representation of matrices and tensors. This is particularly useful in scientific computing.
- **Element-wise operations**: NumPy simplifies element-wise mathematical operations on arrays, making it easy to perform calculations on entire data sets in one go.
- **Random number generation**: It provides a wide range of functions for generating random numbers and random data, which is useful for simulations and statistical analysis.
- **Integration with other libraries**: NumPy seamlessly integrates with other data science libraries like SciPy, Pandas, and Matplotlib, enhancing its utility in various domains.
- **Performance optimization**: NumPy functions are implemented in low-level languages like C and Fortran, which significantly boosts their performance. It's a go-to choice when speed is essential.

Installation

If you haven't already installed NumPy, you can do so using `pip` :

```
pip install numpy
```

Creating NumPy arrays

You can create NumPy arrays from Python lists. These arrays can be one-dimensional or multi-dimensional.

Creating 1D array

```
import numpy as np
```

import numpy as np: In this line, the NumPy library is imported and assigned an alias `np` to make it easier to reference in the code.

```
# Creating a 1D array
arr_1d = np.array([1, 2, 3, 4, 5]) # **np.array()** is used to create NumPy arrays.
```

arr_1d = np.array([1, 2, 3, 4, 5]): in this line, a one-dimensional NumPy array named `arr_1d` is created. It uses the `np.array()` function to convert a Python list [1, 2, 3, 4, 5] into a NumPy array. This array contains five elements, which are 1, 2, 3, 4, and 5. `arr_1d` is a 1D array because it has a single row of elements.

Creating 2D array

```
import numpy as np
```

import numpy as np: In this line, the NumPy library is imported and assigned an alias `np` to make it easier to reference in the code.

```
# Creating a 2D array
arr_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
```

arr_2d = np.array([1, 2, 3], [4, 5, 6], [7, 8, 9]): In this line, a two-dimensional NumPy array named `arr_2d` is created. It uses the `np.array()` function to convert a list of lists into a 2D NumPy array.
The outer list contains three inner lists, each of which represents a row of elements. So, `arr_2d` is a 2D array with three rows and three columns. The elements in this array form a matrix with values from 1 to 9, organized in a 3x3 grid.

Array attributes

NumPy arrays have several useful attributes:

```
# Array attributes
print(arr_2d.ndim)  # ndim : Represents the number of dimensions or "rank" of the array.
# output : 2
print(arr_2d.shape)  # shape : Returns a tuple indicating the number of rows and columns in the array.
# Output : (3, 3)
print(arr_2d.size) # size: Provides the total number of elements in the array.
# Output : 9
```

Indexing and slicing

You can access elements of a NumPy array using indexing and slicing:

In this line, the third element (index 2) of the 1D array `arr_1d` is accessed.

```
# Indexing and slicing
print(arr_1d[2])          # Accessing an element (3rd element)
```

In this line, the element in the 2nd row (index 1) and 3rd column (index 2) of the 2D array `arr_2d` is accessed.

```
print(arr_2d[1, 2])       # Accessing an element (2nd row, 3rd column)
```

In this line, the 2nd row (index 1) of the 2D array `arr_2d` is accessed.

```
print(arr_2d[1])          # Accessing a row (2nd row)
```

In this line, the 2nd column (index 1) of the 2D array `arr_2d` is accessed.

```
print(arr_2d[:, 1])       # Accessing a column (2nd column)
```

Basic operations

NumPy simplifies basic operations on arrays:

Element-wise arithmetic operations:

Addition, subtraction, multiplication, and division of arrays with scalars or other arrays.

Array addition

```
# Array addition
array1 = np.array([1, 2, 3])
array2 = np.array([4, 5, 6])
result = array1 + array2
print(result)  # [5 7 9]
```

Scalar multiplication

```
# Scalar multiplication
array = np.array([1, 2, 3])
result = array * 2 # each element of an array is multiplied by 2
print(result)  # [2 4 6]
```

Element-wise multiplication (Hadamard Product)

```
# Element-wise multiplication (Hadamard product)
array1 = np.array([1, 2, 3])
array2 = np.array([4, 5, 6])
result = array1 * array2
print(result)  # [4 10 18]
```

Matrix multiplication

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

NumPy simplifies these operations, making it easier and more efficient than traditional Python lists.

Operation with NumPy

Here's the list of operation which can be performed using Numpy

Operation	Description	Example
Array Creation	Creating a NumPy array.	`arr = np.array([1, 2, 3, 4, 5])`
Element-Wise Arithmetic	Element-wise addition, subtraction, and so on.	`result = arr1 + arr2`
Scalar Arithmetic	Scalar addition, subtraction, and so on.	`result = arr * 2`
Element-Wise Functions	Applying functions to each element.	`result = np.sqrt(arr)`
Sum and Mean	Calculating the sum and mean of an array.Calculating the sum and mean of an array.	`total = np.sum(arr) average = np.mean(arr)`
Maximum and Minimum Values	Finding the maximum and minimum values.	`max_val = np.max(arr) min_val = np.min(arr)`
Reshaping	Changing the shape of an array.	`reshaped_arr = arr.reshape(2, 3)`
Transposition	Transposing a multi-dimensional array.	`transposed_arr = arr.T`
Matrix Multiplication	Performing matrix multiplication.	`result = np.dot(matrix1, matrix2)`

Conclusion

NumPy is a fundamental library for data science and numerical computations. This guide covers the basics of NumPy, and there's much more to explore. Visit numpy.org for more information and examples.

Author

Akansha Yadav