Department of Computer Science and Engineering PES University, Bangalore, India

Lecture Notes
Python for Computational Problem Solving UE25CS151A

Lecture 71

Introduction to NumPy Arrays

Prepared By,
Prof. Kundhavai K R,
Assistant Professor
Dept. of CSE, PESU

Objective: To introduce the fundamental concepts of NumPy and equip beginners with the basic skills to create, manipulate, and perform operations on NumPy arrays.

1. Introduction to NumPy

What is NumPy?

NumPy, short for Numerical Python, is the most fundamental package for scientific computing in Python. At its core, it provides a powerful object: the N-dimensional array (ndarray). Think of it as a highly efficient, grid-like data structure for storing and manipulating numerical data.

Why use NumPy?

While Python's built-in lists are flexible, they are not ideal for numerical operations, especially on large datasets. NumPy arrays offer significant advantages:

- **Faster Operations**: NumPy operations are implemented in C, a low-level language, making them much faster than equivalent operations on Python lists.
- **Less Memory**: NumPy arrays are more compact and use less memory than Python lists.
- **Convenience**: NumPy provides a vast library of high-level mathematical functions that operate on entire arrays without the need for loops.

Installation and Import

First, you'll need to install it. Open your terminal or command prompt and type:

```
pip install numpy
```

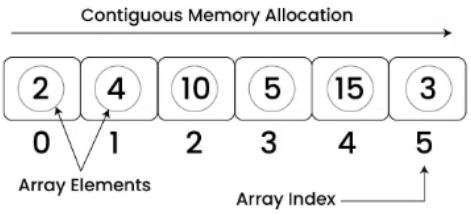
In any Python script where you want to use NumPy, you must import the library. The standard convention is to import it with the alias np.

```
import numpy as np
```

2. NumPy Arrays: The Basics

What is an array?

A NumPy array is a grid of values, all of the same data type. You can have 1-dimensional arrays (vectors), 2-dimensional arrays (matrices), and arrays with 3 or more dimensions (tensors).



A two-dimensional array would be like a table:

1	5	2	0
8	3	6	1
1	7	2	9

Creating a Basic Array

The most common way to create an array is from a Python list.

```
# Create a 1-dimensional array (vector)
a = np.array([1, 2, 3, 4, 5])
print(a)
# Output: [1 2 3 4 5]

# Create a 2-dimensional array (matrix)
b = np.array([[1, 2, 3], [4, 5, 6]])
print(b)
# Output:
# [[1 2 3]
#  [4 5 6]]
```

Array Attributes

Every array has important attributes that describe it:

- **ndarray.ndim**: The number of dimensions (or axes).
- **ndarray.shape**: A tuple of integers indicating the size of the array in each dimension.
- **ndarray.size**: The total number of elements in the array.
- **ndarray.dtype**: The data type of the elements in the array (e.g., int64, float64).

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

# Get the number of dimensions
print(arr.ndim)
# Output: 2

# Get the shape (2 rows, 3 columns)
print(arr.shape)
# Output: (2, 3)

# Get the total number of elements
print(arr.size)
# Output: 6

# Get the data type of the elements
print(arr.dtype)
# Output: int64
```

3. Array Creation and Manipulation

More Ways to Create Arrays

NumPy offers several functions to create arrays from scratch:

```
# Create a 3x3 array of all zeros
zeros_arr = np.zeros((3, 3))
print(zeros_arr)
# Output:
# [[0. 0. 0.]
#  [0. 0. 0.]
#  [0. 0. 0.]]

# Create a 2x4 array of all ones
ones_arr = np.ones((2, 4))
print(ones_arr)
# Output:
# [[1. 1. 1. 1.]
#  [1. 1. 1. 1.]]
```

```
# Create an array with a range of elements (from 0 up to, but not including, 10)
range_arr = np.arange(10)
print(range_arr)
# Output: [0 1 2 3 4 5 6 7 8 9]

# Create an array with 5 elements, evenly spaced between 0 and 1
linspace_arr = np.linspace(0, 1, 5)
print(linspace_arr)
# Output: [0.   0.25 0.5  0.75 1.  ]
```

Basic Array Operations

You can perform element-wise mathematical operations directly on arrays.

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

# Element-wise addition
print(a + b)
# Output: [5 7 9]
```

```
# Element-wise subtraction
print(b - a)
# Output: [3 3 3]

# Element-wise multiplication
print(a * 2)
# Output: [2 4 6]
```

Sorting and Concatenating

```
 #Sorting an array
unsorted_arr = np.array([3, 1, 4, 1, 5, 9, 2])
sorted_arr = np.sort(unsorted_arr)
print(sorted_arr)
# Output: [1 1 2 3 4 5 9]

# Concatenating arrays
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
c = np.concatenate((a, b))
print(c)
# Output: [1 2 3 4 5 6]
```

Reshaping an Array

You can change the shape of an array as long as the new shape has the same number of total elements.

```
# Create a 1D array with 6 elements
arr_1d = np.arange(6)
print("Original:", arr_1d)
# Output: Original: [0 1 2 3 4 5]

# Reshape it to a 2D array (2 rows, 3 columns)
arr_2d = arr_1d.reshape(2, 3)
print("Reshaped:\n", arr_2d)
# Output:
# Reshaped:
# [[0 1 2]
#  [3 4 5]]
```

4. Indexing and Slicing

Basic Indexing and Slicing

This works very similarly to Python lists.

```
# 1D array slicing
arr = np.arange(10)  # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# Get the element at index 3
print(arr[3])
# Output: 3

# Get a slice from index 2 to 5 (exclusive)
print(arr[2:5])
# Output: [2 3 4]

# 2D array indexing (matrix)
matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Get the element in the 2nd row, 3rd column (0-indexed)
# Format is [row, column]
print(matrix[1, 2])
# Output: 6
```

Advanced Indexing (Boolean Indexing)

This is a powerful feature where you can use an array of boolean values to select elements.

```
data = np.array([10, 20, 30, 40, 50])
# Find all elements greater than 25
condition = data > 25
print("Condition:", condition)
# Output: Condition: [False False  True  True  True]

# Use the boolean array to select the elements
print("Selected elements:", data[condition])
# Output: Selected elements: [30 40 50]

# You can also do it in one line
print("Selected in one line:", data[data > 25])
# Output: Selected in one line: [30 40 50]
```

NumPy Aggregation Functions Example

```
import numpy as np

# Create a sample array of student scores
scores = np.array([88, 92, 75, 95, 64, 85, 92])


# Find the highest and lowest scores
max_score = np.max(scores)
min_score = np.min(scores)

# Calculate the average (mean) score
mean_score = np.mean(scores)

# Calculate the standard deviation to see how spread out the scores are
std_dev = np.std(scores)

print(f"Student Scores: {scores}")
print(f"Highest Score: {max_score}") # Output: 95
print(f"Lowest Score: {min_score}")  # Output: 64
print(f"Average Score: {mean_score:.2f}") # Output: 84.43
print(f"Standard Deviation: {std_dev:.2f}") # Output: 10.55
```

- `np.max()` : Finds the largest value in an array.
- `np.min()` : Finds the smallest value in an array.
- `np.mean()` : Calculates the mathematical average of the array's values.
- `np.median()` : Finds the middle value after sorting the array.
- `np.std()` : Measures how spread out the values are from the average.
- `np.var()` : Calculates the variance, the square of the standard deviation.
- `np.prod()` : Returns the result of multiplying all elements in the array together.

5. ARRAYS VS. LISTS

Python lists are excellent, ***general*** purpose containers. However, they can be heterogeneous. Data intensive operations often benefit from homogeneity allowing for improved speed and reduced memory consumption. NumPy Arrays shine here.

LISTS	ARRAYS
Can store Heterogeneous types	Usually Homogeneous.
Dynamic in nature. Can grow or shrink in size.	Static and fixed.
Stores elements in separate object references	Stores elements in contiguous memory references.
Slower for large scale numeric operations because they lack vectorization.	Optimized for bulk, element-wise computations and numerical processing.
General Purpose	Math heavy scientific computations.