NumPy Arrays

Date: October 21, 2025

Creating and Inspecting NumPy Arrays

Objective: To understand what a NumPy array is, how to create it, and how to inspect its basic properties.

Theory: NumPy (Numerical Python) is a foundational library for scientific computing in Python. The core object in NumPy is the ndarray (n-dimensional array), which is a powerful data structure that is significantly faster and more memory-efficient than standard Python lists for numerical operations.

Key Properties of ndarray:

- **ndim:** The number of dimensions of the array.
- **shape:** A tuple indicating the size of the array in each dimension.
- **size:** The total number of elements in the array.
- **dtype:** The data type of the elements in the array.

Program:

```
import numpy as np


# --- 1. Creating a 1D Array ---

# Create a 1-dimensional array from a Python list.

array_1d = np.array([1, 2, 3, 4, 5])

print("1D Array:")

print(array_1d)

print(f"Dimensions (ndim): {array_1d.ndim}")

print(f"Shape (shape): {array_1d.shape}")

print(f"Size (size): {array_1d.size}")

print(f"Data type (dtype): {array_1d.dtype}")

print("-" * 20)


# --- 2. Creating a 2D Array ---

# Create a 2-dimensional array (matrix) from a list of lists.

array_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
```

```
print("2D Array:")
print(array_2d)
print(f"Dimensions (ndim): {array_2d.ndim}")
print(f"Shape (shape): {array_2d.shape}")
print(f"Size (size): {array_2d.size}")
print(f"Data type (dtype): {array_2d.dtype}")
print("-" * 20)

# --- 3. Creating a 3D Array ---
# Create a 3-dimensional array from a list of 2D arrays.
array_3d = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
print("3D Array:")
print(array_3d)
print(f"Dimensions (ndim): {array_3d.ndim}")
print(f"Shape (shape): {array_3d.shape}")
print(f"Size (size): {array_3d.size}")
print(f"Data type (dtype): {array_3d.dtype}")
```

Output:

```
1D Array:
[1 2 3 4 5]
Dimensions (ndim): 1
Shape (shape): (5,)
Size (size): 5
Data type (dtype): int64
--------------------
2D Array:
[[1 2 3]
 [4 5 6]
```

```
 [7 8 9]]
Dimensions (ndim): 2
Shape (shape): (3, 3)
Size (size): 9
Data type (dtype): int64
--------------------
3D Array:
[[[1 2]
  [3 4]]

 [[5 6]
  [7 8]]]
Dimensions (ndim): 3
Shape (shape): (2, 2, 2)
Size (size): 8
Data type (dtype): int64
```

Array Initialization Functions

Objective: To learn about common NumPy functions for creating arrays with specific initial values.

Theory: NumPy provides convenient functions to create arrays of a given size initialized with placeholders or specific values. This is often more efficient than creating a Python list and converting it.

- **np.zeros():** Creates an array filled with zeros.
- **np.ones():** Creates an array filled with ones.
- **np.full():** Creates an array of a given size filled with a specified value.
- **np.arange():** Creates an array with a range of evenly spaced values (similar to Python's range()).
- **np.linspace():** Creates an array with a specified number of evenly spaced values over a given interval.

Program:

```
import numpy as np


# --- 1. Zeros and Ones ---
# Create a 3x3 array of zeros.
zeros_array = np.zeros((3, 3))
print("Array of zeros:")
print(zeros_array)


# Create a 2x4 array of ones.
ones_array = np.ones((2, 4))
print("\nArray of ones:")
print(ones_array)


# --- 2. Full Array ---
# Create a 2x2 array filled with the value 7.
full_array = np.full((2, 2), 7)
print("\nArray filled with 7s:")
print(full_array)


# --- 3. Range and Linspace ---
# Create an array with values from 0 to 9.
range_array = np.arange(10)
print("\nArray from arange(10):")
print(range_array)


# Create an array with 5 evenly spaced values from 0 to 1.
linspace_array = np.linspace(0, 1, 5)
print("\nArray from linspace(0, 1, 5):")
print(linspace_array)
```

Output:

```
Array of zeros:
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]

Array of ones:
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]]

Array filled with 7s:
[[7 7]
 [7 7]]

Array from arange(10):
[0 1 2 3 4 5 6 7 8 9]

Array from linspace(0, 1, 5):
[0.   0.25 0.5  0.75 1.  ]
```