Different types of dimensions in NumPy

Fatemeh Shetabivash

0-Dimensional (Scalar) Array:

A 0-dimensional array is the simplest form, containing a single value or scalar.

It represents a single element, such as a number, string, or boolean value.

Example: arr = np.array(5)

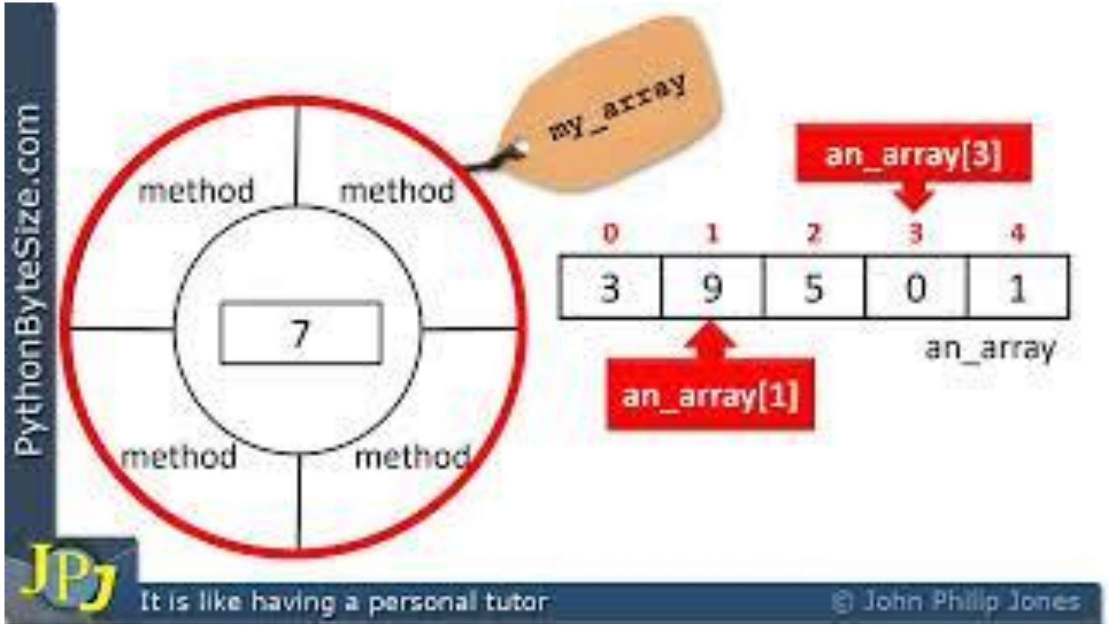
While using a simple variable for single values is possible, using a 0-dimensional array aligns with the principles and advantages of working with arrays in NumPy, offering greater flexibility, consistency, and compatibility with existing code and libraries.

Using a 0-dimensional array instead of a simple variable offers several advantages:

- Compatibility with NumPy: NumPy is a powerful library for numerical computing in Python. It operates efficiently on arrays rather than individual variables. By representing a single value as a 0-dimensional array, you can leverage the capabilities of NumPy and perform array-based operations, calculations, and manipulations.
- Consistent Data Structure: By using arrays consistently, even for single values, you maintain a uniform data structure throughout your code. This can simplify your code logic and make it more readable and maintainable. It also allows you to seamlessly switch between different array dimensions as your requirements evolve.

Using a 0-dimensional array instead of a simple variable offers several advantages:

- Easier Integration with Existing Code: If you're working with code that primarily uses arrays, using a 0-dimensional array ensures compatibility and seamless integration. You can treat the 0-dimensional array as any other array in your codebase, enabling smooth interoperability and reducing the need for special cases.
- Enhanced Functionality: NumPy provides a wide range of functions and operations optimized for arrays. By using a 0-dimensional array, you can take advantage of these functions directly, without needing to convert between variable types or write custom logic for scalar values.
- Broadcasting: Broadcasting is a powerful feature in NumPy that allows arrays with different shapes to be used in operations together. By representing a single value as a 0-dimensional array, you can easily combine it with arrays of any dimensionality using broadcasting rules, expanding the possibilities for efficient and concise computations.

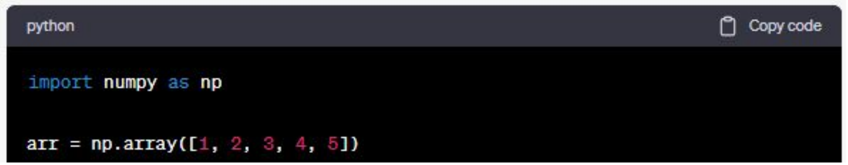


https://youtu.be/kGteVFZHjtQ

1-Dimensional Array:

A 1-dimensional array is like a list or a sequence of elements arranged in a linear fashion.

It has a single axis and can be visualized as a row of values.



```python
import numpy as np

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

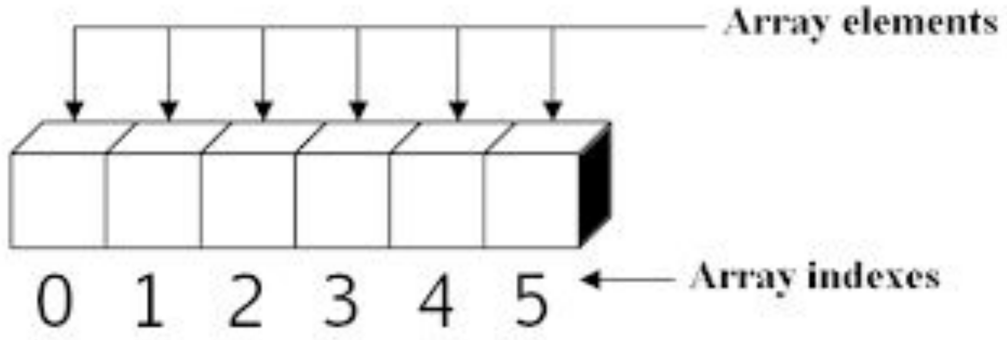
Overall, using a 1-dimensional array in NumPy offers enhanced performance, memory efficiency, and a wide range of specialized functions and methods tailored for efficient numerical computing. It allows for concise, vectorized operations and seamless integration with other scientific libraries, making it a preferred choice over simple lists for many data-related tasks.

Using a 1-dimensional array instead of a simple list offers several advantages:

- Efficient Numerical Operations: Arrays in NumPy are designed for efficient numerical operations. They provide optimized implementations of mathematical functions and operations that can be performed on the entire array at once, resulting in faster computations compared to traditional loops over a list.
- Memory Efficiency: NumPy arrays are more memory-efficient than lists. They store data in a contiguous block of memory, whereas lists in Python are implemented as a collection of individual objects with additional memory overhead. This makes arrays more suitable for handling large datasets and performing memory-intensive operations.
- Vectorized Operations: Arrays support vectorized operations, which means you can apply mathematical operations to the entire array or a subset of it without writing explicit loops. This leads to concise and efficient code, especially when dealing with mathematical computations, data processing, and scientific computing tasks.

Using a 1-dimensional array instead of a simple list offers several advantages:

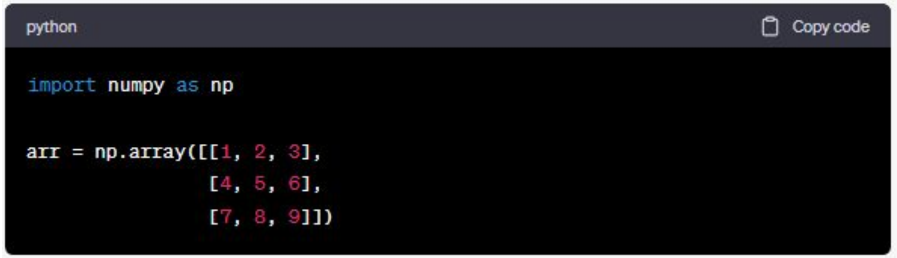
- Improved Functionality: NumPy provides a vast range of functions and methods specifically designed for arrays. These include statistical operations, linear algebra functions, array manipulation methods, and much more. By using arrays, you gain access to this extensive functionality, which simplifies your code and allows for more concise and expressive solutions.
- Integration with Other Libraries: Many scientific and data-related libraries in Python, such as SciPy, Pandas, and Matplotlib, rely heavily on NumPy arrays as their fundamental data structure. By using arrays instead of lists, you ensure seamless integration and compatibility with these libraries, enabling you to take full advantage of their capabilities.
- Performance Optimization: NumPy arrays are implemented in low-level languages like C, which leads to faster execution compared to Python's built-in data structures. This performance optimization is particularly beneficial when working with large datasets or computationally intensive tasks.



One-dimensional array with six elements

2-Dimensional Array:

A 2-dimensional array represents a matrix or a table with rows and columns.
It has two axes, commonly referred to as rows and columns.
It can be visualized as a grid or a spreadsheet.



```python
import numpy as np

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

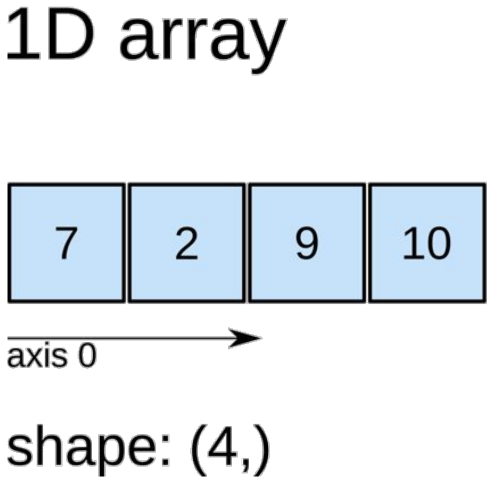
Using a 2-dimensional array in NumPy offers improved memory efficiency, efficient element access, simplified indexing and slicing, vectorized operations, and integration with other scientific libraries. It provides a structured and optimized data structure for working with multi-dimensional data, making it a preferred choice over a list of lists in many data-related applications

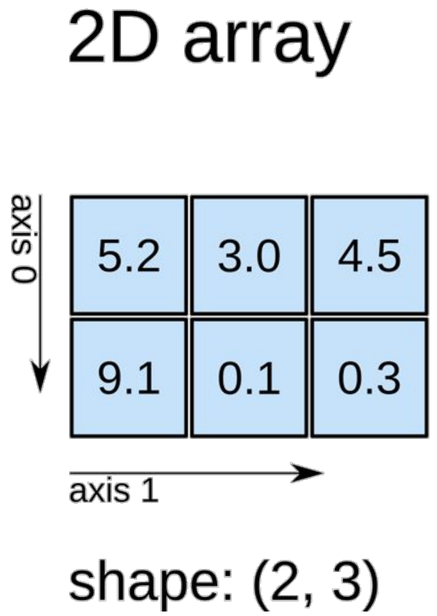
Using a 2-dimensional array instead of a list of lists offers several advantages:

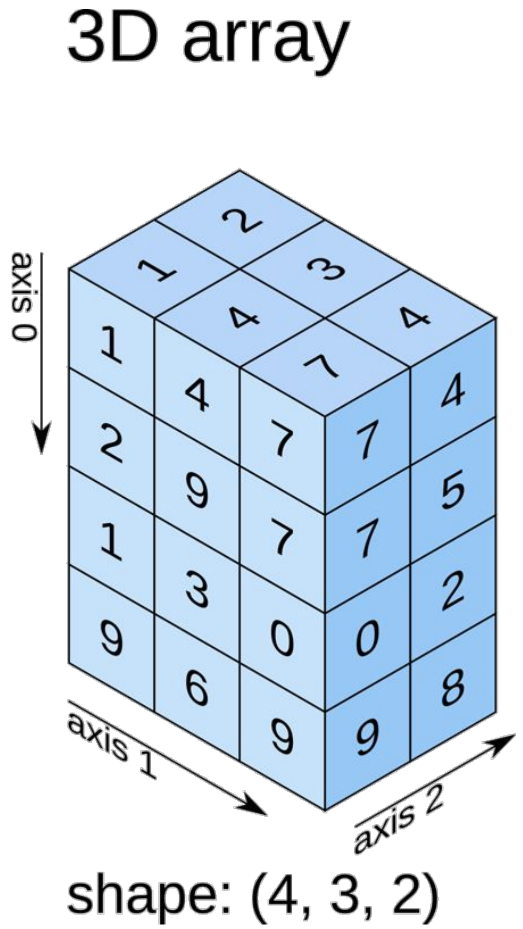
- Efficient Memory Storage: In a 2-dimensional array, the data is stored in a contiguous block of memory, which results in better memory utilization compared to a list of lists. This leads to improved performance and memory efficiency, especially when dealing with large datasets or performing operations that require accessing elements in a structured manner.
- Efficient Element Access: Accessing elements in a 2-dimensional array is more efficient than accessing elements in a list of lists. In a 2-dimensional array, you can directly access an element using its row and column indices, which requires a simple calculation to determine the memory location. In contrast, accessing elements in a list of lists involves multiple levels of indirection, resulting in slower access times.
- Simplified Indexing and Slicing: NumPy arrays provide powerful indexing and slicing capabilities that simplify working with multi-dimensional data. You can easily extract rows, columns, or specific subsets of data using concise syntax. This allows for more readable and efficient code compared to manually manipulating nested lists.

Using a 2-dimensional (and more dimension) array instead of a list of lists offers several advantages:

- Efficient Vectorized Operations: NumPy arrays support vectorized operations, allowing you to perform computations on entire rows or columns of a 2-dimensional array without the need for explicit loops. This leads to concise and efficient code, especially when dealing with mathematical and scientific computations.
- Broad Range of Functions and Methods: NumPy provides a wide range of functions and methods optimized for multi-dimensional arrays. These include statistical functions, linear algebra operations, matrix manipulations, and more. By using a 2-dimensional array, you gain access to these specialized functions, which simplify complex operations and enable efficient data manipulation.
- Integration with Other Libraries: Many scientific and data-related libraries in Python, such as Pandas, Matplotlib, and Scikit-learn, rely heavily on NumPy arrays as their underlying data structure. By using a 2-dimensional array, you ensure seamless integration with these libraries and can take full advantage of their capabilities.

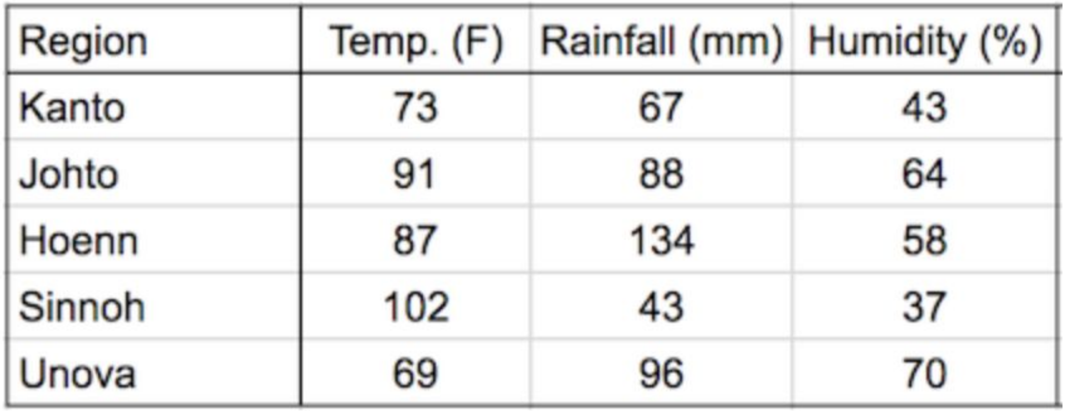

1D array
7
2
9
10
axis 0
shape: (4,)


2D array
axis 0
5.2
3.0
4.5
9.1
0.1
0.3
axis 1
shape: (2, 3)


3D array
axis 0
axis 1
axis 2
shape: (4, 3, 2)

Example of 2D data

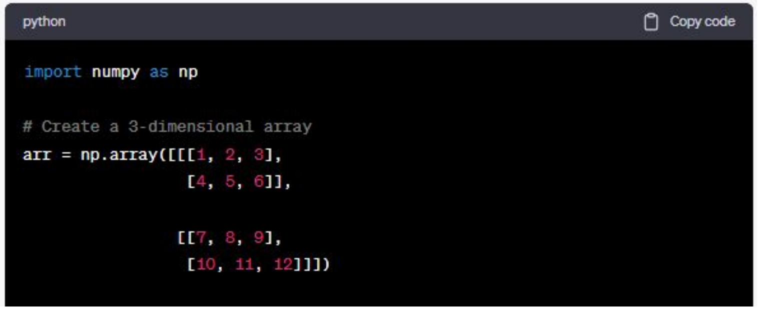
Given some climate data for a region:



Region	Temp. (F)	Rainfall (mm)	Humidity (%)
Kanto	73	67	43
Johto	91	88	64
Hoenn	87	134	58
Sinnoh	102	43	37
Unova	69	96	70

3-Dimensional Array:

- A 3-dimensional array represents a collection of 2D arrays or multiple stacked matrices.
- It has three axes, often referred to as depth, rows, and columns.
- It can be visualized as a stack of 2D arrays or as a cube-like structure.

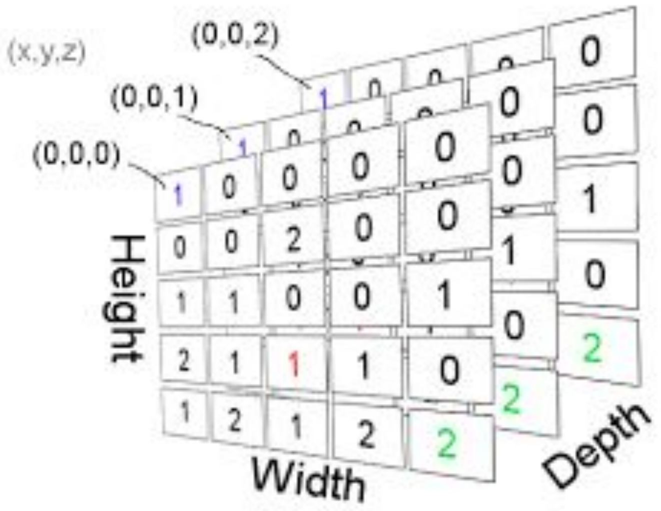


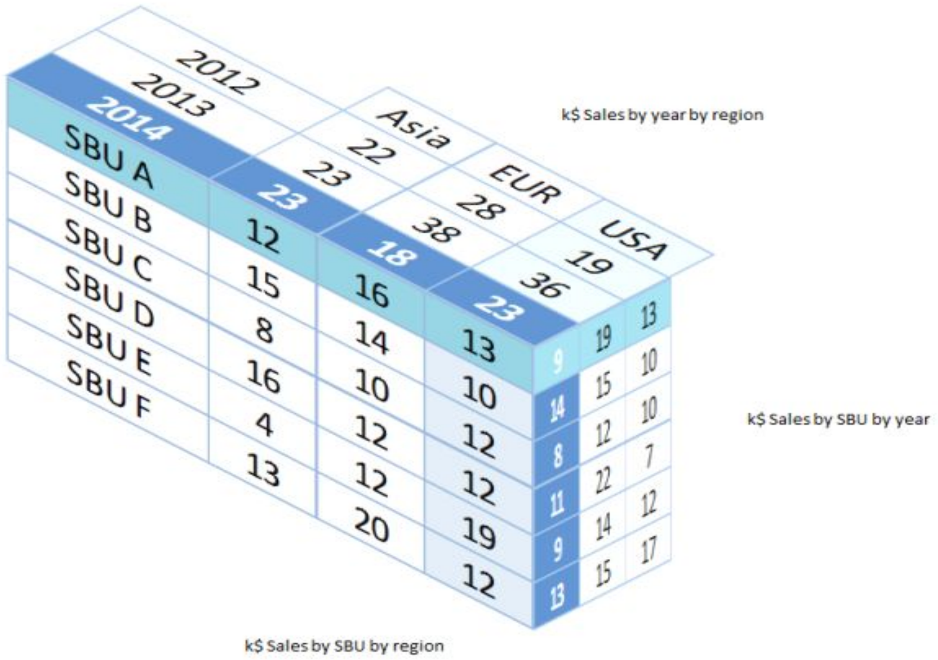
```python
import numpy as np

# Create a 3-dimensional array
arr = np.array([[[1, 2, 3],
                 [4, 5, 6]],

                [[7, 8, 9],
                 [10, 11, 12]]])
```

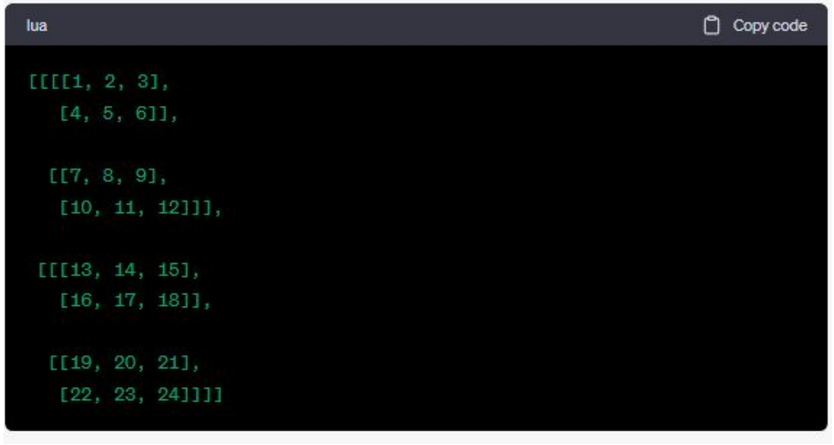
Example of 3D data





4-Dimensional Array:

- A 4-dimensional array represents a collection of 3D arrays or volumes.
- It has four axes, commonly referred to as time, depth, rows, and columns.
- It can be visualized as a stack of 3D arrays or as a multi-dimensional cube.



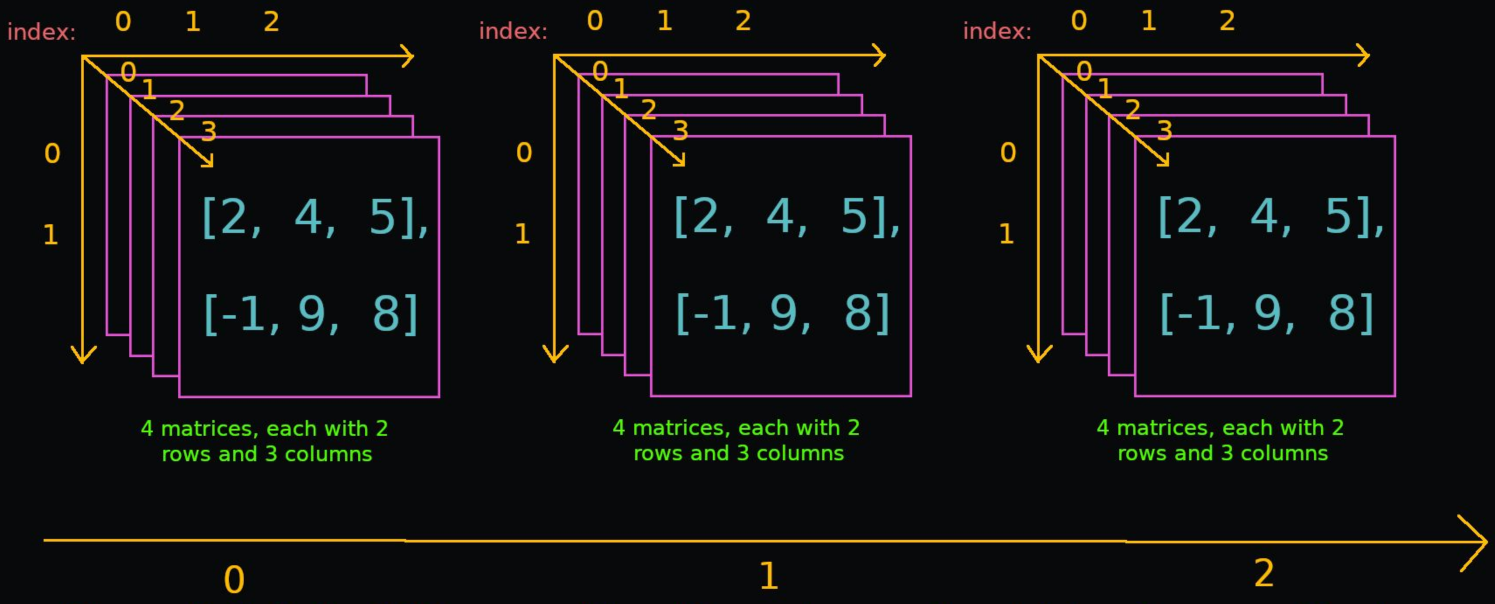
```
[[[[1, 2, 3],
   [4, 5, 6]],


  [[7, 8, 9],
   [10, 11, 12]]],


 [[[13, 14, 15],
   [16, 17, 18]],


  [[19, 20, 21],
   [22, 23, 24]]]]
```

4D Arrays



index: 0 1 2
0
1
0 1 2 3
[2, 4, 5],
[-1, 9, 8]
4 matrices, each with 2 rows and 3 columns
index: 0 1 2
0
1
0 1 2 3
[2, 4, 5],
[-1, 9, 8]
4 matrices, each with 2 rows and 3 columns
index: 0 1 2
0
1
0 1 2 3
[2, 4, 5],
[-1, 9, 8]
4 matrices, each with 2 rows and 3 columns
0
1
2

Shape: (3, 4, 2, 3) - 3 3D arrays of 4 matrices, where each matrix has 2 rows and 3 columns

Thank You!