Session-8 Numpy

19 July 2024 02:08 PM

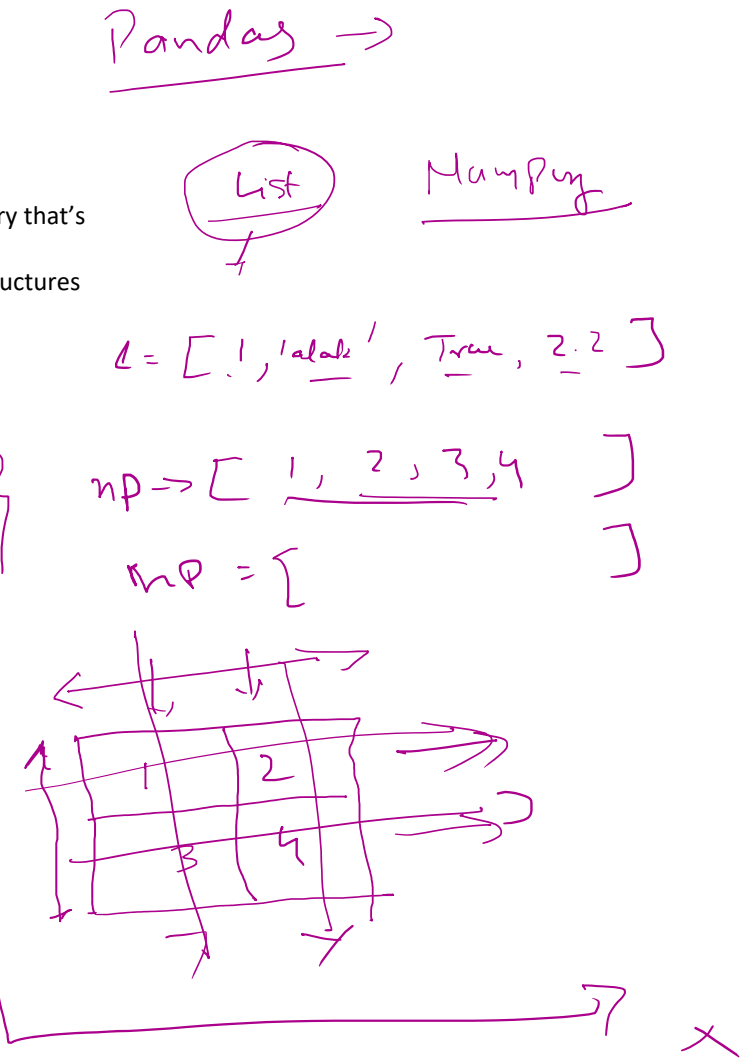
By Alok Ranjan
Linkedin- www.linkedin.com/in/alok-ranjandigu-coder
GitHub- https://github.com/alokyadav2020

Numpy

- NumPy (**Num**erical **Py**thon) is an open source Python library that's widely used in science and engineering.
  The NumPy library contains multidimensional array data structures

1D array

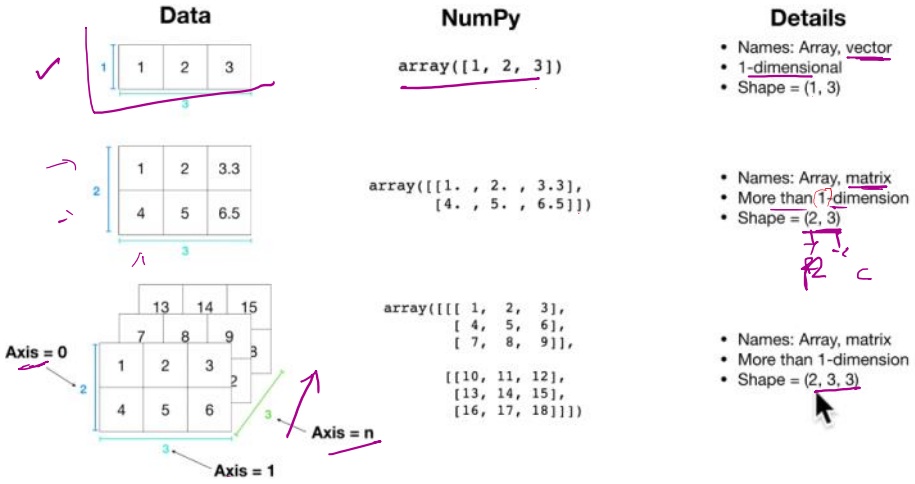
- Create One Dimension array
- With List, random.rand, random.randint,Zeros,Ones,random.random, np.arange
- Ndim
- Shape
- Size
- Type
- Dtype
- Max
- Min
- Sum
- Mean
- Array Arithmetic(scaler,vector)



2D & 3D array

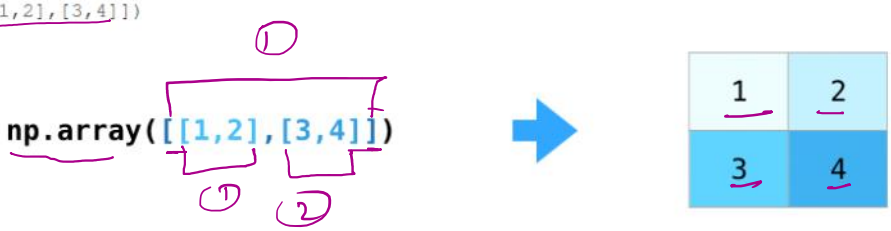
- Create 2D & 3D array
- With List, random.rand, random.randint,Zeros,Ones,random.random, np.arange
- Ndim
- Shape
- Size
- Type
- Dtype
- Max
- Min
- Sum
- Mean
- Array Arithmetic(scaler,vector)

Anatomy of a NumPy array

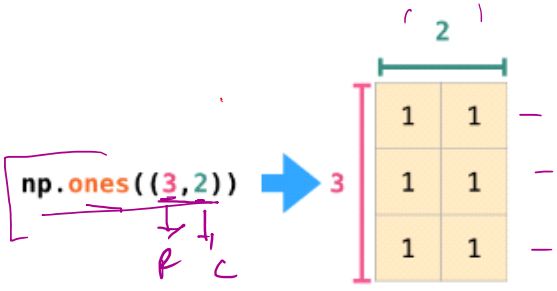


2D Array

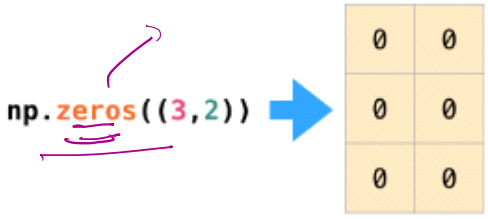
```
np.array([[1,2],[3,4]])
```



Ones 2D array

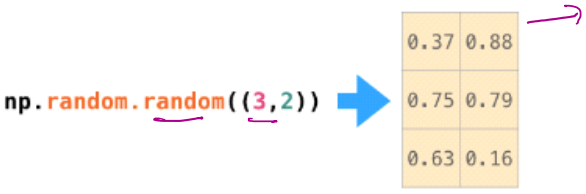


Zeros 2D array



Screen clipping taken: 20-07-2024 04:17 PM

Random 2D array

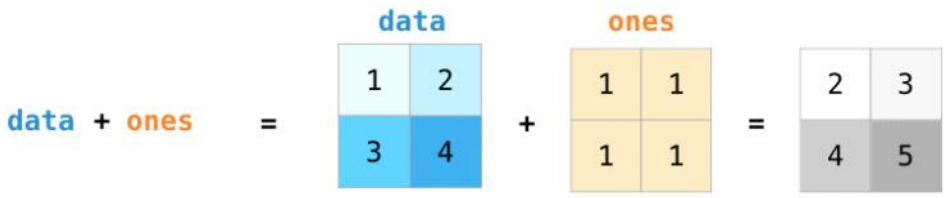


Matrix Arithmetic

We can add and multiply matrices using arithmetic operators (+-*/) if the two matrices are the same size. NumPy handles those as position-wise operations:

1.

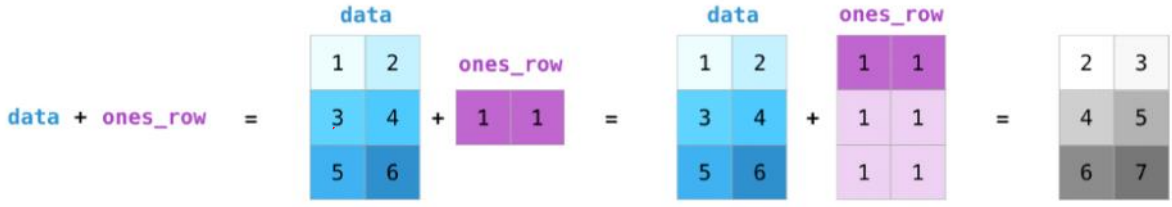
```
>>> data = np.array([[1, 2], [3, 4]])
>>> ones = np.array([[1, 1], [1, 1]])
>>> data + ones
array([[2, 3],
       [4, 5]])
```



2.

You can do these arithmetic operations on matrices of different sizes, but only if one matrix has only one column or one row. In this case, NumPy will use its broadcast rules for the operation.

```
>>> data = np.array([[1, 2], [3, 4], [5, 6]])
>>> ones_row = np.array([[1, 1]])
>>> data + ones_row
array([[2, 3],
       [4, 5],
       [6, 7]])
```



Broadcasting

NumPy understands that the multiplication should happen with each cell. That concept is called **broadcasting**. Broadcasting is a mechanism that allows NumPy to perform operations on arrays of different shapes.

There are times when you might want to carry out an operation between an array and a single number (also called *an operation between a vector and a scalar*) or between arrays of two different sizes.

For example, your array (we'll call it "data") might contain information about distance in miles but you want to convert the information to kilometers. You can perform this operation with:

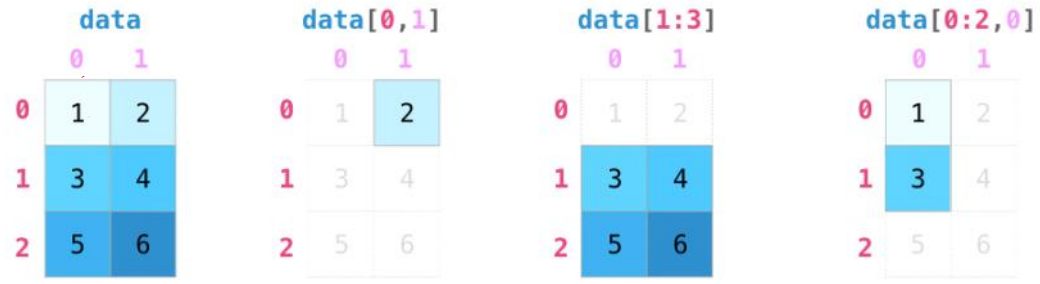
```
>>> data = np.array([1.0, 2.0])
>>> data * 1.6
array([1.6, 3.2])
```

Matrix Indexing

Indexing and slicing operations are useful when you're manipulating matrices:

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

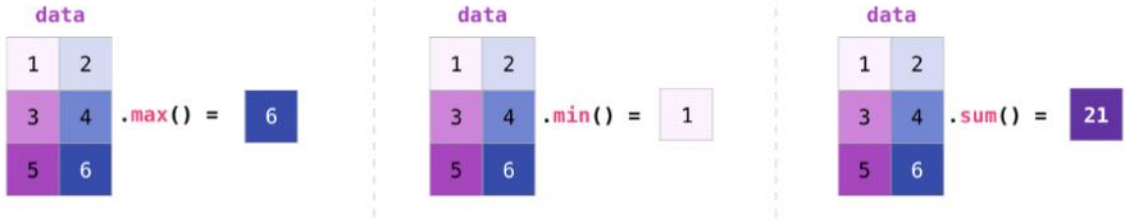
```
>>> data[0, 1]
2
>>> data[1:3]
array([[3, 4],
       [5, 6]])
>>> data[0:2, 0]
array([1, 3])
```



Matrix Aggregation

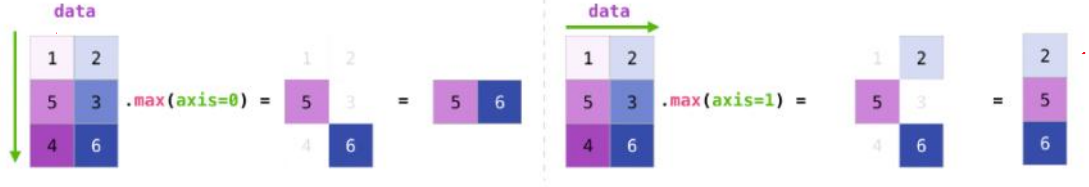
You can aggregate matrices the same way you aggregated vectors:

```
>>> data.max()
6
>>> data.min()
1
>>> data.sum()
21
```



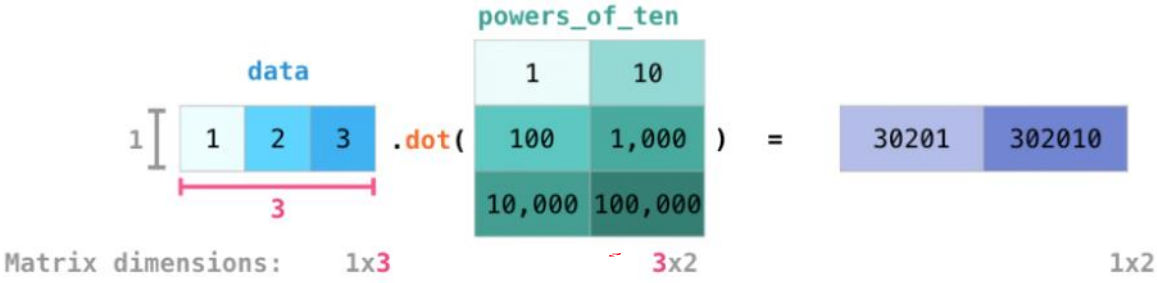
Not only can we aggregate all the values in a matrix, but we can also aggregate across the rows or columns by using the axis parameter:

```
>>> data = np.array([[1, 2], [5, 3], [4, 6]])
>>> data
array([[1, 2],
       [5, 3],
       [4, 6]])
>>> data.max(axis=0)
array([5, 6])
>>> data.max(axis=1)
array([2, 5, 6])
```

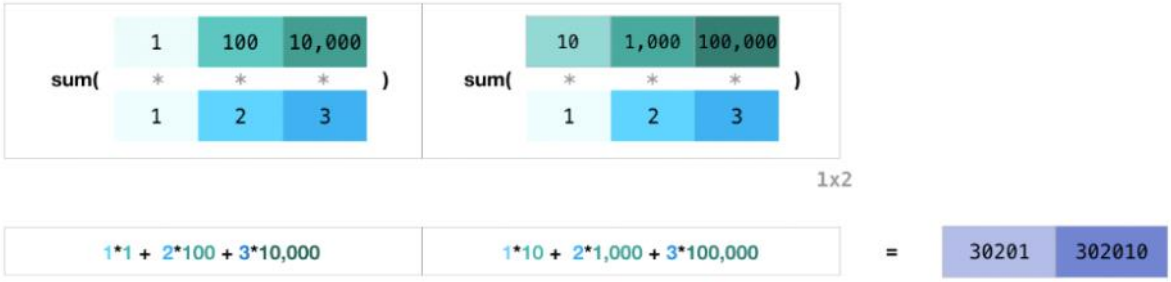


Dot Product

A key distinction to make with arithmetic is the case of matrix multiplication using the dot product. NumPy gives every matrix a dot() method we can use to carry-out dot product operations with other matrices:



I've added matrix dimensions at the bottom of this figure to stress that the two matrices have to have the same dimension on the side they face each other with. You can visualize this operation as looking like this:



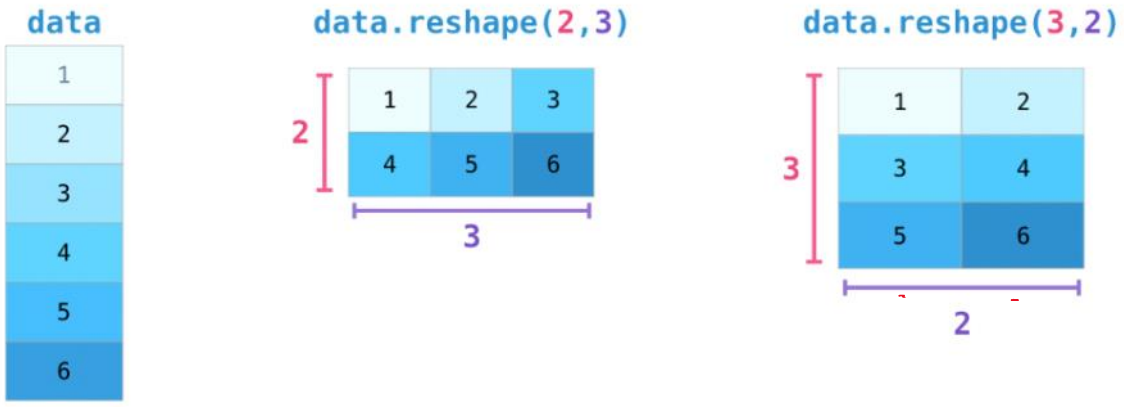
Transposing

A common need when dealing with matrices is the need to rotate them. This is often the case when we need to take the dot product of two matrices and need to align the dimension they share. NumPy arrays have a convenient property called T to get the transpose of a matrix:

Reshaping

In more advanced use case, you may find yourself needing to switch the dimensions of a certain matrix. This is often the case in machine learning applications where a certain model expects a certain shape for the inputs that is different from your dataset. NumPy's reshape() method is useful in these cases. You just pass it the new dimensions you want for the matrix. You can pass -1 for a dimension and NumPy can infer the correct dimension based on your matrix:

```
>>> data.reshape(2, 3)
array([[1, 2, 3],
       [4, 5, 6]])
>>> data.reshape(3, 2)
array([[1, 2],
       [3, 4],
       [5, 6]])
```



Yet More Dimensions

NumPy can do everything we've mentioned in any number of dimensions. Its central data structure is called ndarray (N-Dimensional Array) for a reason.

In a lot of ways, dealing with a new dimension is just adding a comma to the parameters of a NumPy function: