



## AI4D LAB TRAINING

---

[Zephania Reuben \(https://nsoma.me\)](https://nsoma.me)

July 17, 2023

---

## DATA SCIENCE | NUMPY

---

### OUTLINES:

- Introduction to NumPy
- Getting started with NumPy
- Creating NumPy array
- NumPy arrays operations
- Inspecting size and shape
- Extracting items from an array
- Reverse rows and columns
- Computing mean,min,max on the ndarray
- Reshaping and Flattening multidimensional array
- Creating sequences, repetitions, and random numbers
- Getting unique items and count

# Introduction to NumPy

## What is NumPy?

- The NumPy library (NumPy, Numerical Python) is a package of high level mathematical functions and tools that are designed to work with the numpy array objects.
- A numpy array is a high performance multidimensional data structure more efficient than a [Python list](#)
- At the core, numpy provides the excellent ndarray objects, short for n-dimensional arrays.
- In a 'ndarray' object, it can store multiple items of the same data type.

## Getting started with NumPy

### Installing Matplotlib

- **Installing Matplotlib on Linux**

- In the following table, we will present some of the common Linux distributions package names for Matplotlib and the tools we can use to install the package:

Distribution	Package Name
Debian or Ubuntu (And other Debian derivatives)	<code>sudo apt-get install python-numpy</code>
Fedora	<code>sudo dnf install numpy</code>

- **Installing Matplotlib on Windows**

- Open command line interface (CMD) use the next command to install **Matplotlib**
- `python -m pip install numpy`

### Testing our installation

- To ensure we have correctly installed Matplotlib and its dependencies, a very simple test can be carried out in the following manner:

```
In [1]: import numpy
print("Numpy version : ", numpy.__version__)
```

```
Numpy version : 1.24.3
```

## Creating NumPy arrays

- There are multiple ways to create a numpy array, most of which will be covered as you read this. However one of the most common ways is to create one from a list or a list like an object by passing it to the `np.array` function.

```
In [2]: import numpy as np
```

```
In [3]: f = np.array([1,3,5,6,9])
```

```
In [4]: type(f)
```

```
Out[4]: numpy.ndarray
```

```
In [5]: # Create an 1d array from a list  
  
list1 = [0,1,2,3,4]
```

```
In [6]: type(list1)
```

```
Out[6]: list
```

```
In [8]: arr1d = np.array(list1)  
# Print the array and its type  
type(arr1d)
```

```
Out[8]: numpy.ndarray
```

```
In [ ]: arr1d
```

```
In [ ]: # [5] - 0D - Scalar, shape : (1)  
# [4,5,6] - 1D - Vector, shape: (1, 3)  
# [[6,6,7], [3,2,1]] - 2D - Matrix, shape : (2, 3)  
# [[[6,6,7], [3,2,1]], [[6,6,7], [3,2,1]]] - 3D - Tensor, shape: (2, 2, 3)
```

```
In [ ]: [  
        [1,4],  
        [9,8]  
    ]  
    (2,2)
```

```
In [16]: a = np.array([[[1,4],[9,8]], [[6,8],[9,5]]])
```

```
In [17]: a.shape
```

```
Out[17]: (2, 2, 2)
```

```
In [18]: # Create a 2d array from a list of lists
list2 = [[0,1,2], [3,4,5], [6,7,8]]

arr2d = np.array(list2)

arr2d
```

```
Out[18]: array([[0, 1, 2],
               [3, 4, 5],
               [6, 7, 8]])
```

```
In [19]: # Create a float 2d array
arr2d_f = np.array(list2, dtype='float')

arr2d_f
```

```
Out[19]: array([[0., 1., 2.],
               [3., 4., 5.],
               [6., 7., 8.]])
```

```
In [20]: # Convert to 'int' datatype
arr2d_f = arr2d_f.astype('int')
```

```
In [21]: arr2d_f.dtype
```

```
Out[21]: dtype('int64')
```

```
In [22]: # Convert an array back to a list
arr2d_f.tolist()
```

```
Out[22]: [[0, 1, 2], [3, 4, 5], [6, 7, 8]]
```

## NumPy arrays operations

Different operations can be applied to numpy array like arithmetic and logical operations:-

```
In [23]: import numpy as np

x = np.array([[1,2,3,4],[5,6,7,8]]) # 2x4

y = np.array([[9,10,11,12],[13,14,15,16]]) # 2x4
```

```
In [24]: np.add(x,y) # x+y
```

```
Out[24]: array([[10, 12, 14, 16],  
               [18, 20, 22, 24]])
```

```
In [25]: x + y
```

```
Out[25]: array([[10, 12, 14, 16],  
               [18, 20, 22, 24]])
```

```
In [26]: #Getting reminder  
np.remainder(y,x)
```

```
Out[26]: array([[0, 0, 2, 0],  
               [3, 2, 1, 0]])
```

```
In [27]: #powering an ndarray  
np.power(x,2)
```

```
Out[27]: array([[ 1,  4,  9, 16],  
               [25, 36, 49, 64]])
```

```
In [28]: #subtracting a ndarray  
y-x #np.subtract(y,x)
```

```
Out[28]: array([[8, 8, 8, 8],  
               [8, 8, 8, 8]])
```

```
In [29]: #logical operator  
x<5
```

```
Out[29]: array([[ True,  True,  True,  True],  
               [False, False, False, False]])
```

```
In [30]: np.less(x,5)
```

```
Out[30]: array([[ True,  True,  True,  True],  
               [False, False, False, False]])
```

```
In [31]: np.greater_equal(y,10)
```

```
Out[31]: array([[False,  True,  True,  True],  
               [ True,  True,  True,  True]])
```

## Inspecting shape and size

```
In [32]: # Create a 2d array with 3 rows and 4 columns
list2 = [[1, 2, 3, 4],[3, 4, 5, 6], [5, 6, 7, 8]] # 3x4
arr2 = np.array(list2, dtype='float')
arr2
```

```
Out[32]: array([[1., 2., 3., 4.],
               [3., 4., 5., 6.],
               [5., 6., 7., 8.]])
```

```
In [33]: b = np.array([3,9,9,67,43,12])
```

```
In [46]: b[5]
```

```
Out[46]: 12
```

```
In [48]: b[-2]
```

```
Out[48]: 43
```

```
In [45]: b[::-1]
```

```
Out[45]: array([12, 43, 67,  9,  9,  3])
```

```
In [36]: b[1:4] # first to n-1
```

```
Out[36]: array([ 9, 67])
```

```
In [37]: arr2[1:3,1:3]
```

```
Out[37]: array([[4., 5.],
               [6., 7.]])
```

```
In [38]: arr2[:2, :2] #[[1,2],[3,4]]
```

```
Out[38]: array([[1., 2.],
               [3., 4.]])
```

```
In [39]: np.transpose(arr2)
```

```
Out[39]: array([[1., 3., 5.],
               [2., 4., 6.],
               [3., 5., 7.],
               [4., 6., 8.]])
```

```
In [40]: np.trace(arr2)
```

```
Out[40]: 12.0
```

```
In [41]: np.diagonal(arr2)
```

```
Out[41]: array([1., 4., 7.])
```

```
In [42]: # shape
print('Shape: ', arr2.shape)
```

```
Shape: (3, 4)
```

```
In [43]: # dtype
print('Datatype: ', arr2.dtype)
```

```
Datatype: float64
```

```
In [44]: # size
print('Size: ', arr2.size)
```

```
Size: 12
```

```
In [ ]:
```

## Extracting items from an array

- Elements are extracted from specific portions on an array using indexing starting with 0, something similar to python lists.
- But unlike lists, numpy arrays can optionally accept as many parameters in the square brackets as there is number of dimensions.

```
In [ ]: # Extract the first 2 rows and columns
arr2[:2, :2]
```

```
In [ ]: #list2[:2, :2] # error
```

## Reverse rows and columns

- Reversing an array works like how you would do with lists, but you need to do for all the axes (dimensions) if you want a complete reversal.

```
In [ ]: # Reverse only the row positions
arr2[::-1, ]
```

```
In [ ]: # Reverse the row and column positions
arr2[::-1, ::-1]
```

## Computing mean,min,max on the ndarray

```
In [49]: # mean
print("Mean value is: ", arr2.mean())
```

Mean value is: 4.5

```
In [50]: #max
print("Max value is: ", arr2.max())
```

Max value is: 8.0

```
In [51]: #min
print("Min value is: ", arr2.min())
```

Min value is: 1.0

```
In [52]: np.median(arr2)
```

Out[52]: 4.5

```
In [53]: np.std(arr2)
```

Out[53]: 1.9790570145063195

## Reshaping and Flattening multidimensional array

- Reshaping is changing the arrangement of items so that shape of the array changes while maintaining the same number of dimensions.
- Flattening, however, will convert a multi-dimensional array to a flat 1d array. And not any other shape.

```
In [54]: arr2
```

```
Out[54]: array([[1., 2., 3., 4.],
                [3., 4., 5., 6.],
                [5., 6., 7., 8.]])
```



```
In [55]: arr2.reshape(2,6)
```

```
Out[55]: array([[1., 2., 3., 4., 3., 4.],
               [5., 6., 5., 6., 7., 8.]])
```

```
In [56]: # Reshape a 3x4 array to 4x3 array
arr2.reshape(2, 3)
```

```
-----
-----
ValueError                                Traceback (most recent c
all last)
Cell In[56], line 2
      1 # Reshape a 3x4 array to 4x3 array
----> 2 arr2.reshape(2, 3)

ValueError: cannot reshape array of size 12 into shape (2,3)
```

```
In [57]: a = arr2.reshape(-1,1) # (12,1)
```

```
In [58]: a.shape
```

```
Out[58]: (12, 1)
```

```
In [ ]: a
```

```
In [59]: b = arr2.reshape(1,-1) # (1,12)
```

```
In [60]: b.shape
```

```
Out[60]: (1, 12)
```

```
In [61]: b
```

```
Out[61]: array([[1., 2., 3., 4., 3., 4., 5., 6., 5., 6., 7., 8.]])
```

```
In [ ]: A = (mxn)
        B = (pxq)

        AxB = (n=p)
```

```
In [ ]:
```

```
In [62]: # Flatten it to a 1d array
arr2.flatten()
```

```
Out[62]: array([1., 2., 3., 4., 3., 4., 5., 6., 5., 6., 7., 8.])
```

## Creating sequences, repetitions, and random numbers

The `np.arange` function comes handy to create customised number sequences as ndarray. You can set the starting and end positions using `np.arange`.

```
In [63]: # Lower limit is 0 be default  
print(np.arange(5)) #[first_index,last_index-1,step]  
[0 1 2 3 4]
```

```
In [64]: # 0 to 9  
print(np.arange(0, 10))  
[0 1 2 3 4 5 6 7 8 9]
```

```
In [65]: # 0 to 9 with step of 2  
print(np.arange(0, 10, 2))  
[0 2 4 6 8]
```

```
In [67]: # 10 to 1, decreasing order  
print(np.arange(10, 0, -2))  
[10  8  6  4  2]
```

```
In [69]: a = [1,2,3]
```

```
In [70]: # Repeat whole of 'a' two times  
print('Tile: ', np.tile(a, 3))  
Tile:      [1 2 3 1 2 3 1 2 3]
```

```
In [71]: # Repeat each element of 'a' two times  
print('Repeat: ', np.repeat(a, 4))  
Repeat:    [1 1 1 1 2 2 2 2 3 3 3 3]
```

```
In [79]: np.random.seed(10)
```

```
In [86]: # Random numbers between [0,1) of shape 2,2  
np.random.seed(10)  
print(np.random.rand(2,2))  
[[0.77132064 0.02075195]  
 [0.63364823 0.74880388]]
```

```
In [84]: # Normal distribution with mean=0 and variance=1 of shape 2,2
print(np.random.randn(2,2))

[[-1.54540029 -0.00838385]
 [ 0.62133597 -0.72008556]]
```

```
In [76]: # Random integers between [0, 10) of shape 2,2
print(np.random.randint(0, 10, size=[2,2]))

[[9 6]
 [1 6]]
```

```
In [77]: # One random number between [0,1)
print(np.random.random())

0.1590875025180919
```

```
In [78]: # Random numbers between [0,1) of shape 2,2
print(np.random.random(size=[2,2]))

[[0.75004353 0.63986856]
 [0.68368209 0.84047986]]
```

```
In [ ]: # Pick 10 items from a given list, with equal probability
print(np.random.choice(['a', 'e', 'i', 'o', 'u'], size=10))
```

```
In [ ]: # Pick 10 items from a given list with a predefined probability 'p'
print(np.random.choice(['a', 'e', 'i', 'o', 'u'], size=10, p=[0.0,
.4, 0.0, 0.0, 0.6])) # picks more o's
```

## Getting unique items and count

```
In [89]: # Create random integers of size 10 between [0,10)
np.random.seed(30)
arr_rand = np.random.randint(0, 10, size=10)
```

```
In [90]: print(arr_rand)

[5 5 4 7 2 5 1 3 9 7]
```

```
In [91]: # Get the unique items and their counts
unqs, counts = np.unique(arr_rand, return_counts=True)
```

```
In [92]: print("Unique items : ", unqs)

Unique items : [1 2 3 4 5 7 9]
```

```
In [93]: print("Counts      : ", counts)
```

```
Counts      :  [1 1 1 1 3 2 1]
```

```
In [ ]:
```