

# NumPy for Data Analysis

WOMEN IN DATA  
ACADEMY

# Session Content



**What is NumPy?**



**Using the NumPy  
Library**



**Lists v Arrays**



**Slicing Arrays**



# What is NumPy?

NumPy is one of the most useful Maths and Scientific Python libraries. It is valuable in the world of data science and is key for Machine Learning.

- NumPy stands for Numerical Python (NumPy).
- NumPy is used for data analysis on arrays, it performs various tasks and handles large datasets effectively and efficiently at fast speeds.
- Arrays are a collection of data/values that can have more than one dimension. One dimension is known as a vector and a two-dimension array is a matrix!
- Not to be confused with Pandas – which is a tabular data tool



# How to import NumPy?



```
import numpy as np
```



You will need the above statement to import the library and by using np you are saying from now on it will be known as np (quicker to type np than NumPy)



List vs Array

# Array - ndarray

To overcome the problem of not being able to calculate effectively and efficiently we use **arrays** in NumPy.

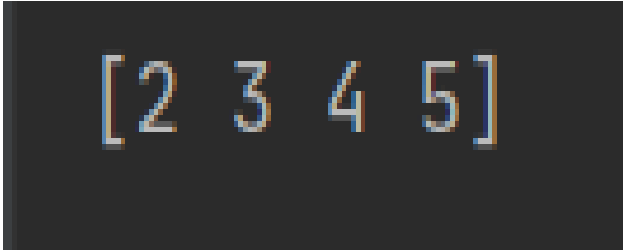
Arrays are a collection of entities/ values, that can have one or more dimensions. An array with one dimension is called a vector and a two dimensions is called a matrix. They store entities of the same data type and size. NumPy arrays are known for their high-performance but also offers efficient storage and data functions as the arrays grow.

- NumPy arrays are easy to create considering how they solve complex problems. Using an **nd array** is more efficient and faster than using a regular array.
- To create a basic np array, you use the `np.array()` function. You have to pass the values of the array as a list: **`array = np.array ([1, 2, 3, 4])`**

# ndarray Slicing

Slicing in python means extracting elements from an array.

```
import numpy as np  
  
array = np.array([1, 2, 3, 4, 5, 6, 7])  
  
print(array[1:5]) # slices data from index 1 to 6 - 0 indexed
```



```
[2 3 4 5]
```



# NumPy Zeros

NumPy has a function that allows you to create an array of all zeros for a given shape and type of array. To do this you use the **np.zeros()** function. Key point is to pass the shape of the array you want to create.

```
import numpy as np

zeros = np.zeros(10)

print (zeros)
```

```
[0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
```



# NumPy Ones

You can also create an array of all 1s using the **np.ones()** function.

```
import numpy as np  
  
ones = np.ones(8)  
  
print (ones)
```

```
[1.  1.  1.  1.  1.  1.  1.  1.]
```

# NumPy Random Array

Another method that is used a lot to create **ndarrays** is **np.random.rand()** function.

This function creates an array for a given shape with random values generated from 0 to 1.

```
import numpy as np

random = np.random.rand(3,4)

print (random)
```

```
[[0.49447036 0.84743384 0.20937511 0.65435976]
 [0.09825423 0.99120167 0.49042729 0.12040814]
 [0.99049738 0.20688358 0.03484015 0.26389752]]
```

# Identity Matrix (Imatrix )

The **np.eye function** returns an array with 1s as an identity along the diagonal shape of the matrix and 0s in the rest of the array.  
Meaning you can return a identity matrix with specified dimensions.

```
import numpy as np  
  
eye = np.eye (6)  
  
print (eye)
```

```
[[1.  0.  0.  0.  0.  0.]  
 [0.  1.  0.  0.  0.  0.]  
 [0.  0.  1.  0.  0.  0.]  
 [0.  0.  0.  1.  0.  0.]  
 [0.  0.  0.  0.  1.  0.]  
 [0.  0.  0.  0.  0.  1.]]
```

# Evenly Spaced Array

A quick way of getting an evenly spaced array is to use the `np.arange()` function.

```
import numpy as np  
  
even = np.arange(6)  
  
print(even)
```

```
[0 1 2 3 4 5]
```



# Evenly Spaced Array

You can also use the `np.arange` function to return an evenly spaced values within a given **interval**.

`np.arange (start, stop, step)`

So if we wrote `np.arange (0,12,2)` this means that we are creating an array from 0 to 12 by the increments of 2.

The start being 0, the stop being 12 and the 2 is the increment. Note that 10 is not going to be in the array.

```
import numpy as np  
  
array = np.arange(0, 15, 2)  
  
print(array)
```

```
[ 0  2  4  6  8 10 12 14]
```

# Shape and Reshape NumPy Arrays

To reshape your ndarray you can use the `np.shape()` function.  
This changes the shape of the array without changing any of actual data in the array.

```
import numpy as np

array = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

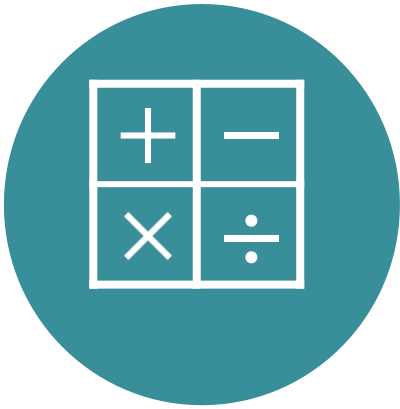
newArray = array.reshape(4, 3)

print(newArray)
```

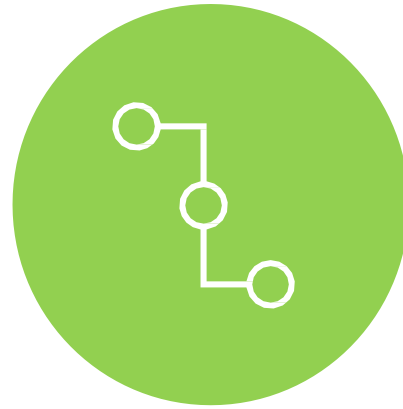
```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
```

Note it has printed 4 rows and 3 columns!.

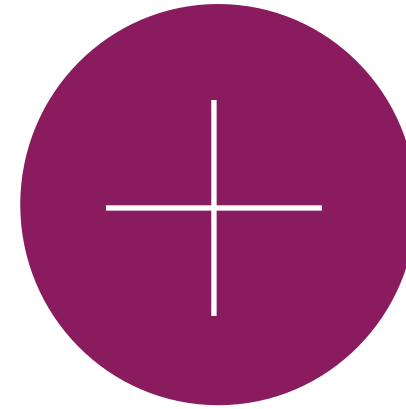
# Session Content



**MATHS  
OPERATORS**



**NP.DOT**



**NP.SUM**

# Maths in NumPy Arrays

You can do Maths using the basic arithmetic operators on NumPy arrays.

```
import numpy as np

array = np.arange(0,5)

print (array)

print("Subtract :", array - 10)
print("Multiply :", array * 10)
print("Divide :", array / 10)
print("Power :", array ** 2)
```

```
[0 1 2 3 4]
Subtract : [-10 -9 -8 -7 -6]
Multiply : [ 0 10 20 30 40]
Divide : [0.  0.1 0.2 0.3 0.4]
Power : [ 0  1  4  9 16]
```



# NP.DOT

This performs **matrix** multiplication so long as the two matrices are able to be multiplied, the same size. (meaning the column numbers need to match)

```
import numpy as np

a = np.arange (1,10).reshape (3,3)
b = np.arange (10,19).reshape (3,3)
c = np.dot(a, b)
print(c)
```

```
[[ 84  90  96]
 [201 216 231]
 [318 342 366]]
```

R1  $\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$   
 R2  $\begin{bmatrix} 4 & 5 & 6 \end{bmatrix}$   
 R3  $\begin{bmatrix} 7 & 8 & 9 \end{bmatrix}$

C1 C2 C3

$\begin{bmatrix} 10 & 11 & 12 \\ 13 & 14 & 15 \\ 16 & 17 & 18 \end{bmatrix}$

R1 X C1	R1 X C2	R1 X C3
$(1 \times 10) + (2 \times 13) + (3 \times 16)$	$(1 \times 11) + (2 \times 14) + (3 \times 17)$	$(1 \times 12) + (2 \times 15) + (3 \times 18)$
R2 X C1	R2 X C2	R2 X C3
R3 X C1	R3 X C2	R3 X C3



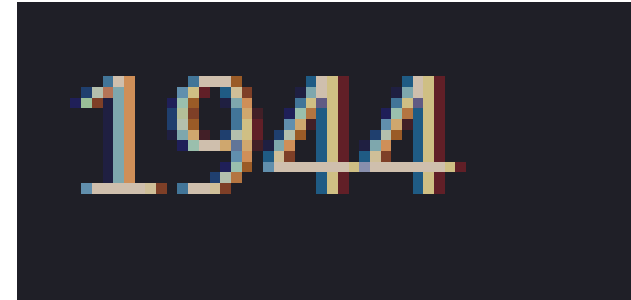
$\begin{bmatrix} 84 & 90 & 96 \\ 201 & 216 & 231 \\ 318 & 342 & 366 \end{bmatrix}$

# NP.SUM

This performs an overall sum of all the elements in the array – adds them together!

```
import numpy as np

a = np.arange (1,10).reshape (3,3)
b = np.arange (10,19).reshape (3,3)
c = np.dot(a, b)
sum = np.sum(c)
print(sum)
```



# Sum by Column

This performs an overall sum of all the elements in the array **column** – adds them together!

```
import numpy as np
# rows

a = np.arange (1,10).reshape (3,3)
b = np.arange (10,19).reshape (3,3)
c = np.dot(a, b)

e = np.sum(c,axis = 0)

print(e)
```

```
[ 270  648 1026]
```



# Sum by Row

This performs an overall sum of all the elements in the array **row** – adds them together!

```
import numpy as np
# rows

a = np.arange (1,10).reshape (3,3)
b = np.arange (10,19).reshape (3,3)
c = np.dot(a, b)

e = np.sum(c,axis = 1)

print(e)
```

```
[ 270  648 1026]
```

# Home Learning Project





1. Create a 1D array of numbers from 0 to 9
2. Create a 3×3 NumPy array of all Boolean value Trues
3. Extract all odd numbers from array of 1-10
4. Replace all odd numbers in an array of 1-10 with the value -1
5. Convert a 1D array to a 2D array with 2 rows
6. Create two arrays a and b, stack these two arrays vertically use the np.dot and np.sum to calculate totals

### Extension:

1. Create the following pattern without hardcoding. Use only NumPy functions.

```
#> array([1, 1, 1, 2, 2, 2, 3, 3, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3])
```

2. In two arrays a ( 1,2,3,4,5) and b ( 4,5,6,7,8,9) – remove all repeating items present in array b
3. Get all items between 5 and 10 from a and b and sum them together





# WOMEN IN DATA ACADEMY

TECH TALENT  
ACADEMY