TechTalent Academy
Safeguarding Policy

*"Protecting an adult's right to live in **safety, free from abuse and neglect**. It is about people and organisations working together to **prevent and stop both the risks and experience of abuse or neglect**, while at the same time making sure that the **adult's wellbeing is promoted** including, where appropriate, having regard to their views, wishes, feelings and beliefs in deciding on any action. This must recognise that adults sometimes have complex interpersonal relationships and may be ambivalent, unclear or unrealistic about their personal circumstances."*

If you have a safeguarding concern, please raise this with your tutor or via the safeguarding link on our website:

https://www.techtalent.co.uk/safeguarding-statement

TechTalent's safeguarding lead is: **Max Ruddock**

Starter Activity.

NumPy is a Python library used for working with arrays.

Why would that be useful for data analysis?

techtalent
academy

TechTalent Academy

Data Science Course

NumPy

Lesson Objectives.

- What is NumPy?
- Lists Vs arrays
- Array functions
- Data Distribution

techtalent academy

What is NumPy?

NumPy is one of the most useful Maths and Scientific Python library. 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 (also offers efficient storage and data functions as the arrays grow
- Not to be confused with Pandas – which is a tabular data tool

NumPy

How to Import NumPy.

techtalent academy

Pip install numpy

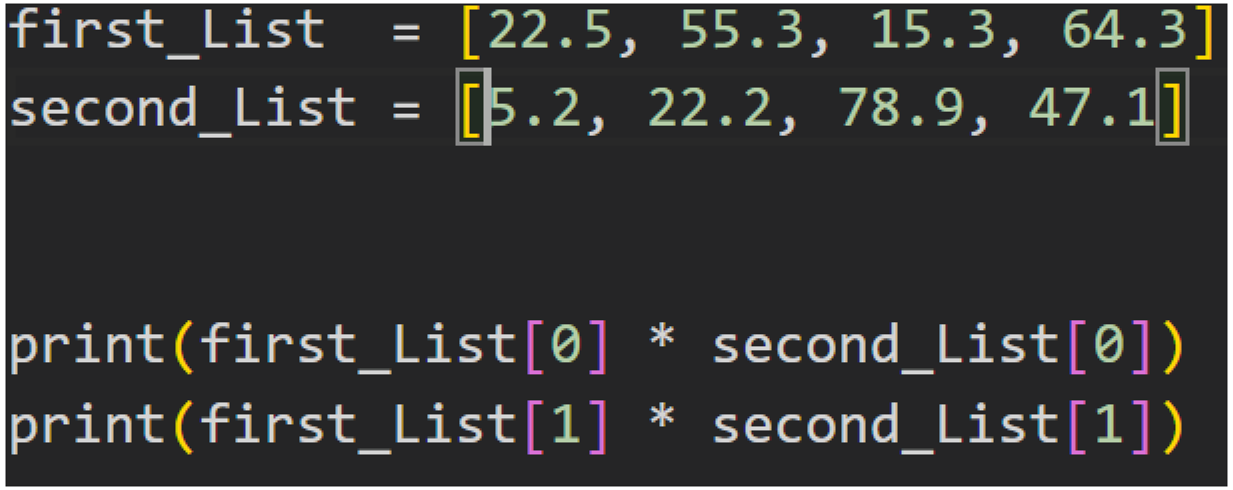
import numpy as np

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)

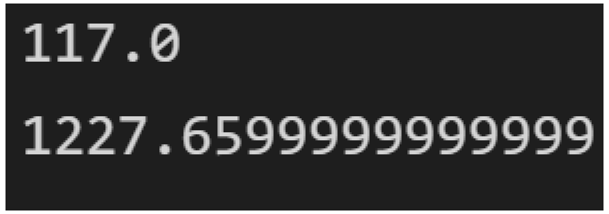
Official documentation:
https://numpy.org/doc/stable/user/index.html#user

List V Array.

To print the calculation of 22.5 * 5.2, you have to do each element individually which becomes a lengthy process. This is where NumPy is better to do calculations over the entire array.



```
first_List  = [22.5, 55.3, 15.3, 64.3]
second_List = [5.2, 22.2, 78.9, 47.1]


print(first_List[0] * second_List[0])
print(first_List[1] * second_List[1])
```



```
117.0
1227.6599999999999
```

Array.

- 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!
- Numpy arrays store entities of the **same data type** and have fixed size.
- 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:

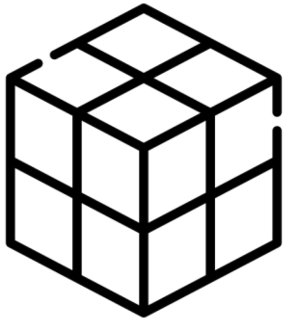
  array = np.array ([1, 2, 3, 4])

1	4	78

Vector/1d array

23	7	12
5	5	33
6	32	2

Matrix/2d array



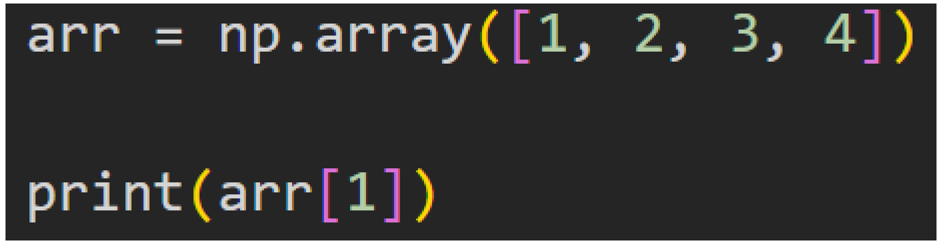
nd-array

Task: Array.

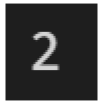
- Install the NumPy library using pip or the terminal.
- Create a 1D array.
- Create a 2D array.

Indexing.

We can index arrays, just like we can index lists.

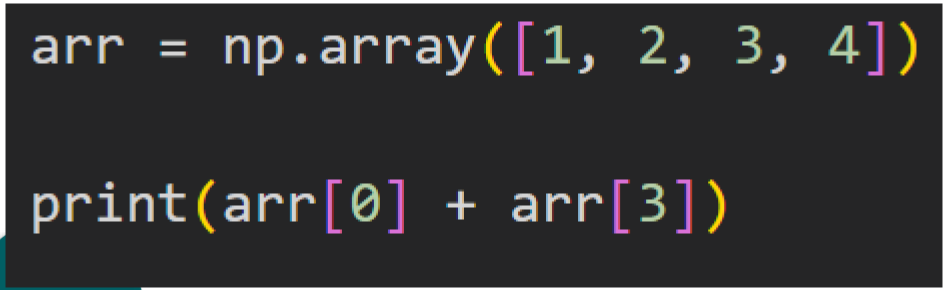


```python
arr = np.array([1, 2, 3, 4])

print(arr[1])
```

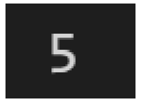


```
2
```

We can also do basic math calculations.



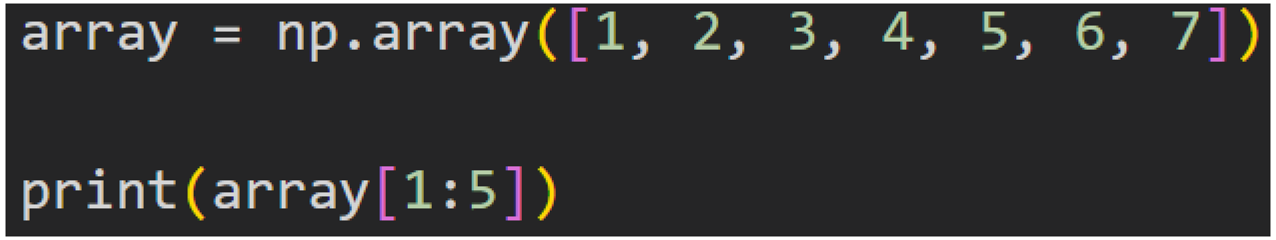
```python
arr = np.array([1, 2, 3, 4])

print(arr[0] + arr[3])
```



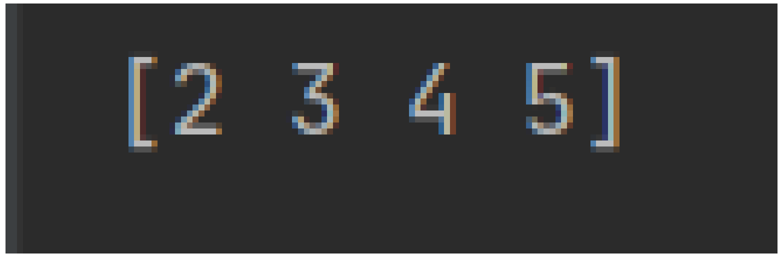
```
5
```

ndarray Slicing.

Slicing in python means extracting elements from an array.



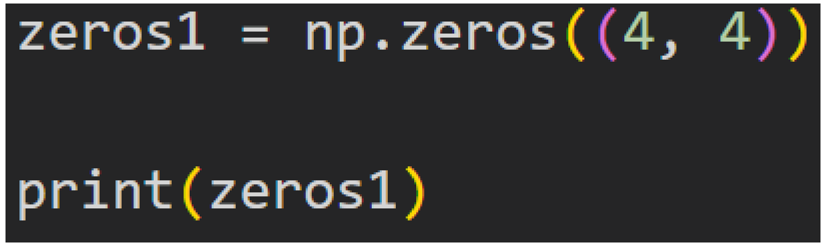
```
array = np.array([1, 2, 3, 4, 5, 6, 7])

print(array[1:5])
```



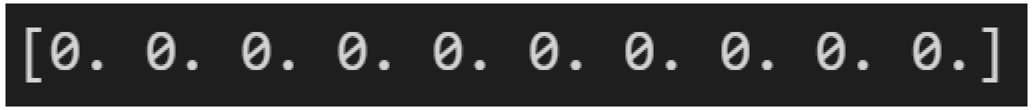
```
[2 3 4 5]
```

NumPy Zeros and Ones.

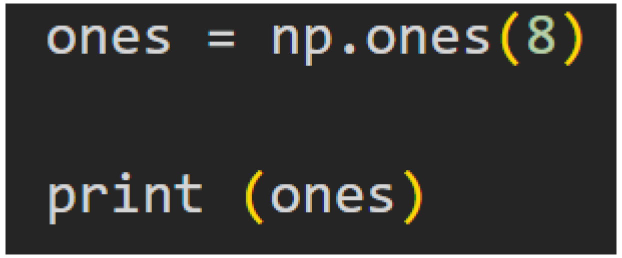
NumPy has a function that allows you to create an array of all zeros or ones for a given shape and type of array. To do this you use the **np.zeros()** or **np.ones()** functions . Key point is to pass the shape of the array you want to create.



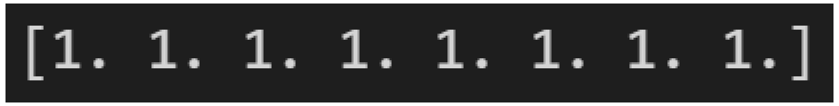
```
zeros1 = np.zeros((4, 4))

print(zeros1)
```



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



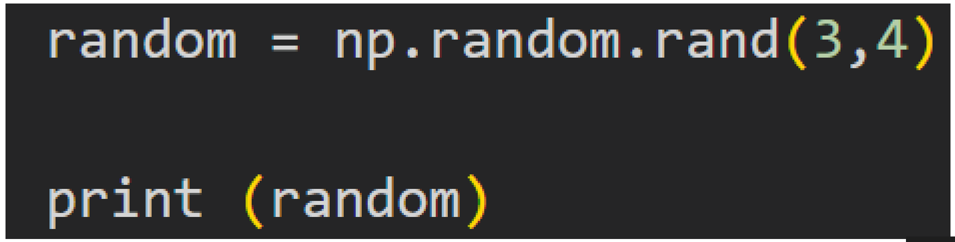
```
ones = np.ones(8)

print (ones)
```



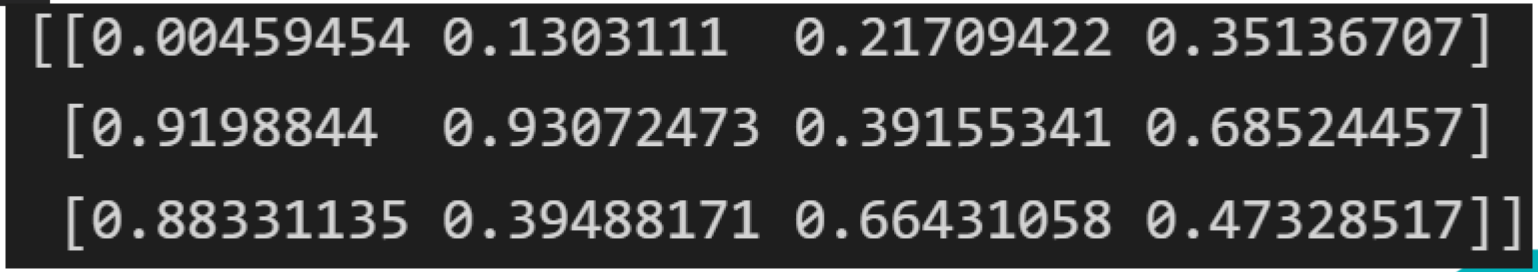
```
[1. 1. 1. 1. 1. 1. 1. 1.]
```

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.



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

print (random)
```



```
[[0.00459454 0.1303111  0.21709422 0.35136707]
 [0.9198844  0.93072473 0.39155341 0.68524457]
 [0.88331135 0.39488171 0.66431058 0.47328517]]
```

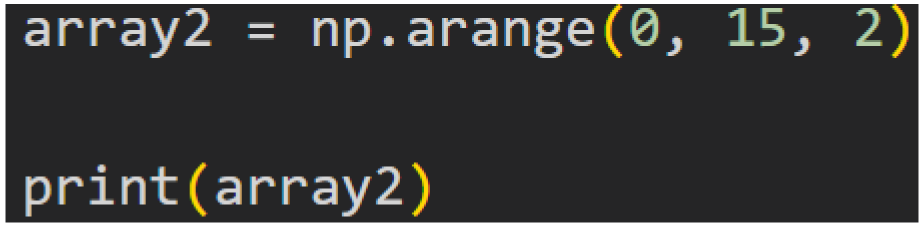
NP.arange().

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

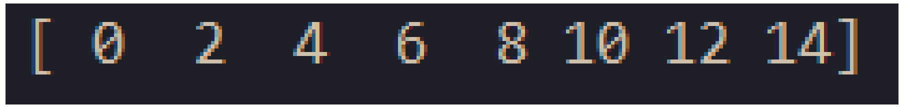
You can use also use the np.arange function to return an evenly spaced values within a given **interval**.
np.arange (start, stop, step)

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.



```
array2 = np.arange(0, 15, 2)

print(array2)
```



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

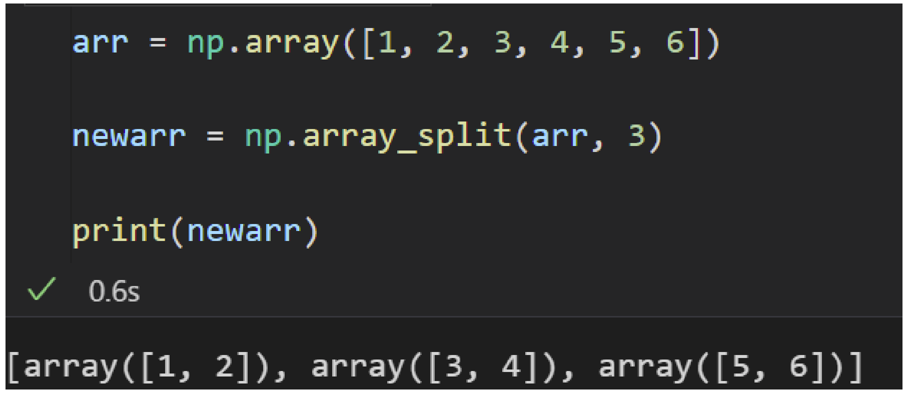
Shape and Reshape NumPy Arrays.

NumPy arrays have an attribute called .shape() that returns a tuple with each index having the number of corresponding elements.

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

Split and Join arrays.

We use **array_split()** for splitting arrays, we pass it the array we want to split and the number of splits

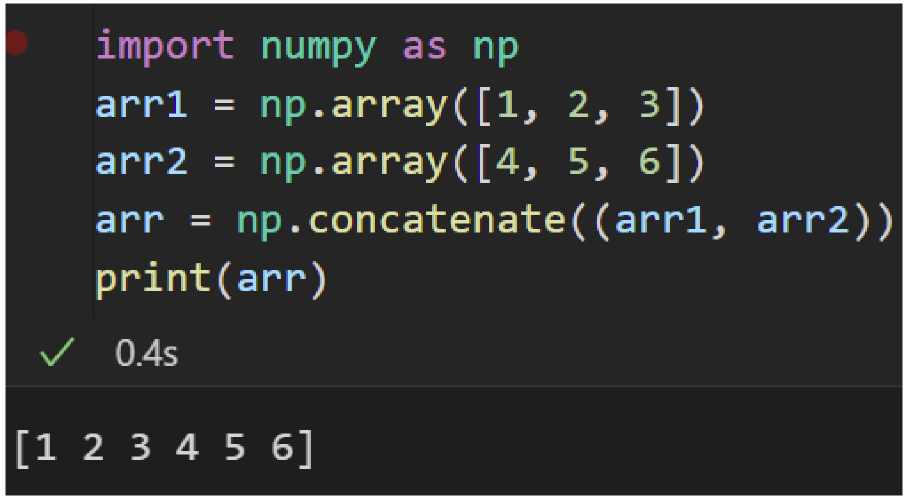


```
arr = np.array([1, 2, 3, 4, 5, 6])

newarr = np.array_split(arr, 3)

print(newarr)
```

✓ 0.6s

```
[array([1, 2]), array([3, 4]), array([5, 6])]
```

We pass a sequence of arrays that we want to join to the **concatenate()** function, along with the axis.



```
import numpy as np
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
arr = np.concatenate((arr1, arr2))
print(arr)
```

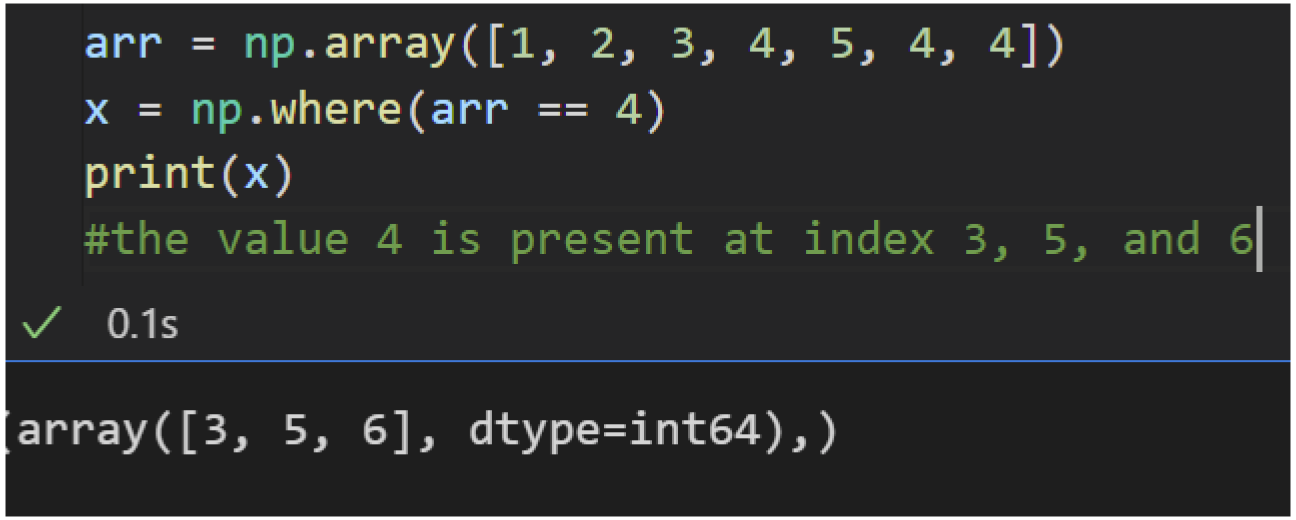
✓ 0.4s

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

Search Arrays.

You can search an array for a certain value, and return the indexes that get a match.

To search an array, use the **where()** method.



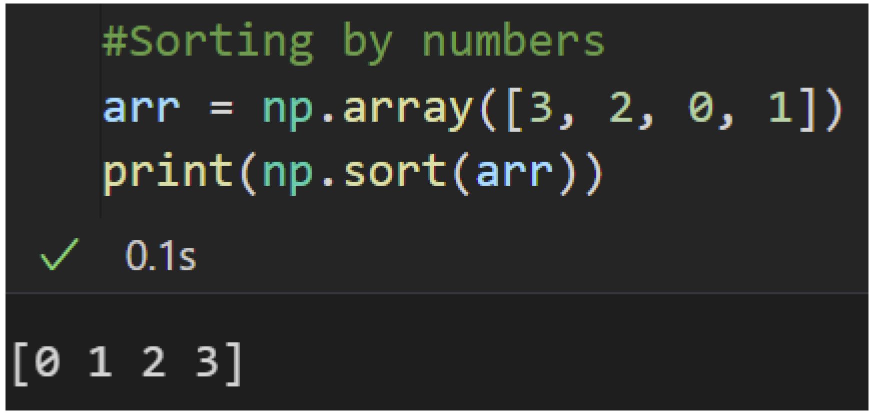
```
arr = np.array([1, 2, 3, 4, 5, 4, 4])
x = np.where(arr == 4)
print(x)
#the value 4 is present at index 3, 5, and 6
```

✓ 0.1s

```
(array([3, 5, 6], dtype=int64),)
```

Sorting Arrays.

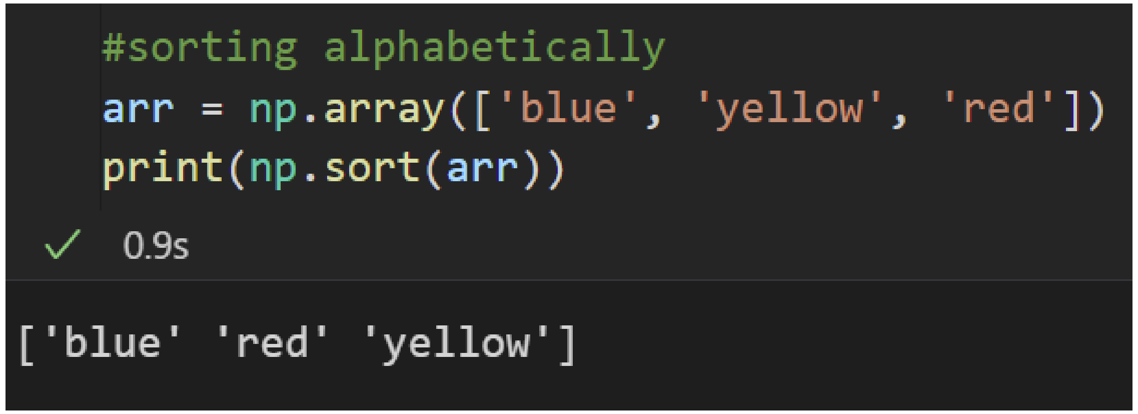
Sorting means putting elements in an ordered sequence like numeric or alphabetical, ascending or descending. NumPy has a function called **sort()**, that will sort a specified array.



```
#Sorting by numbers
arr = np.array([3, 2, 0, 1])
print(np.sort(arr))
```

✓ 0.1s

```
[0 1 2 3]
```



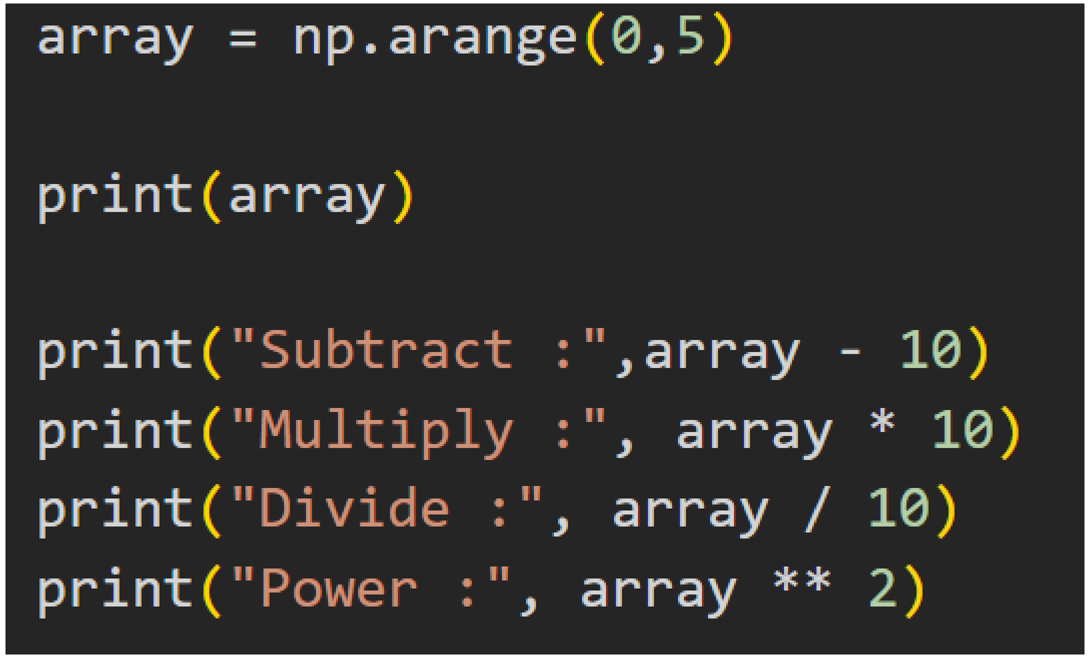
```
#sorting alphabetically
arr = np.array(['blue', 'yellow', 'red'])
print(np.sort(arr))
```

✓ 0.9s

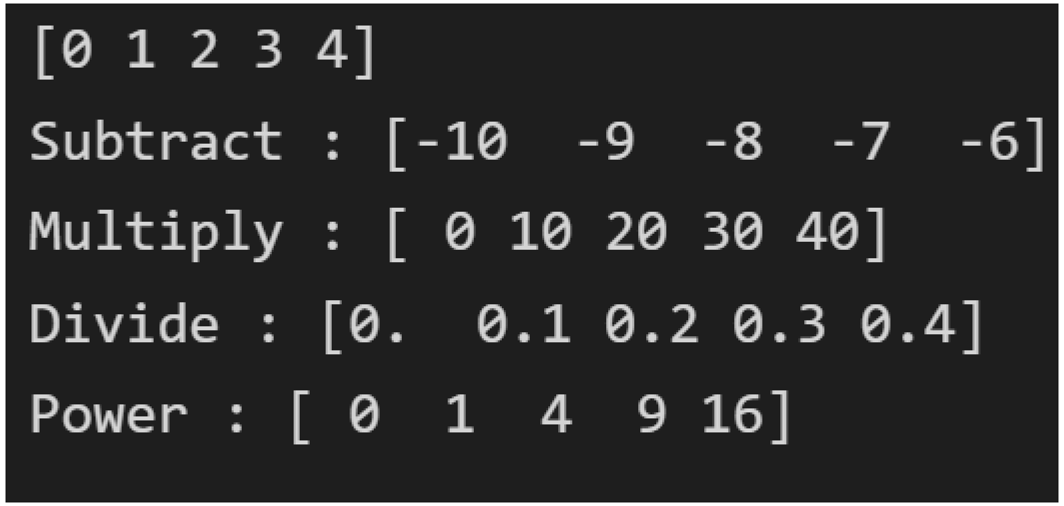
```
['blue' 'red' 'yellow']
```

Maths in NumPy.

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



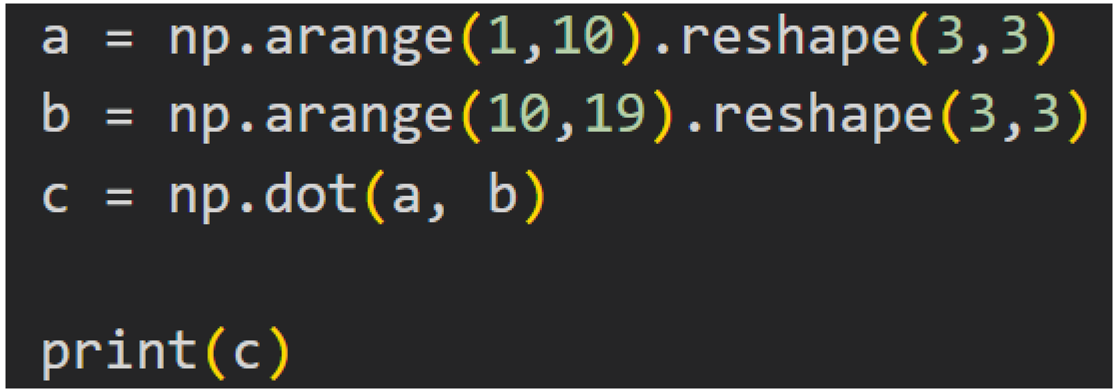
```python
array = np.arange(0,5)

print(array)

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



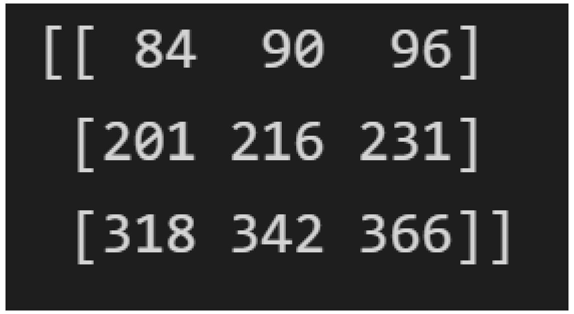
```
[0 1 2 3 4]
Subtract : [-10  -9  -8  -7  -6]
Multiply : [ 0 10 20 30 40]
Divide : [0.  0.1 0.2 0.3 0.4]
Power : [ 0  1  4  9 16]
```

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).



```
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]]
```

Live demonstration
http://matrixmultiplication.xyz/

NP Dot.

techtalent academy

$$\begin{array}{c|c} \text{R1} & [[1\ \ 2\ \ 3] \\ \text{R2} & \ [4\ \ 5\ \ 6] \\ \text{R3} & \ [7\ \ 8\ \ 9]] \end{array}$$

$$\begin{array}{ccc} \text{C1} & \text{C2} & \text{C3} \\ \hline [[10 & 11 & 12] \\ \ [13 & 14 & 15] \\ \ [16 & 17 & 18]] \end{array}$$

R1 X C1 (1x10)+(2x13)+(3x16)	**R1 X C2** (1x11)+(2x14)+(3x17)	**R1 X C3** (1x12)+(2x15)+(3x18)
R2 X C1	**R2 X C2**	**R2 X C3**
R3 X C1	**R3 X C2**	**R3 X C3**

$\downarrow$

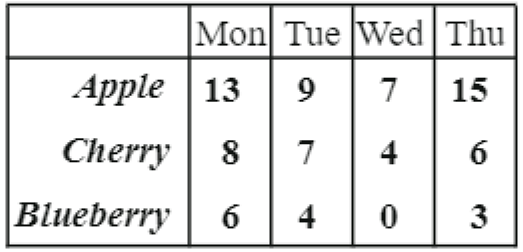
```
[[ 84  90  96]
 [201 216 231]
 [318 342 366]]
```

Why Do Matrix Multiplication?

techtalent academy

- Apple pies cost **£3** each
- Cherry pies cost **£4** each
- Blueberry pies cost **£2** each

This is what they sold in 4 days.



	Mon	Tue	Wed	Thu
Apple	13	9	7	15
Cherry	8	7	4	6
Blueberry	6	4	0	3

Now think about this ... the **value of sales** for Monday is calculated this way:
£3×13 + £4×8 + £2×6 = £83

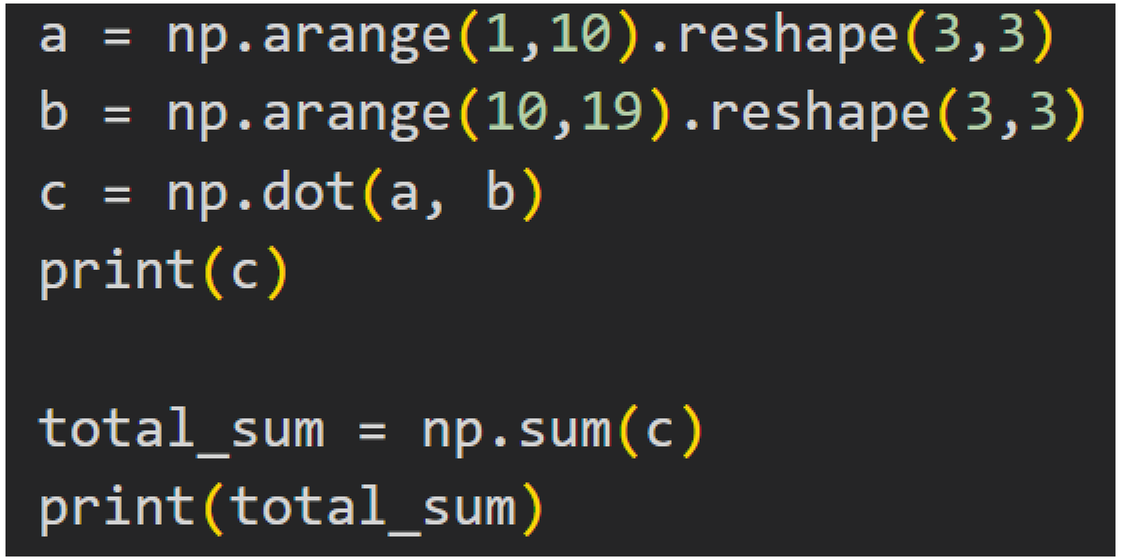
- The sales for Monday were:
Apple pies: **£3×13=£39**, Cherry pies: **£4×8=£32**, and Blueberry pies: **£2×6=£12**. Together that is £39 + £32 + £12 = **£83**
- Tuesday: **£3×9 + £4×7 + £2×4 = £63**
- Wednesday: **£3×7 + £4×4 + £2×0 = £37**
- Thursday: **£3×15 + £4×6 + £2×3 = £75**

We can use matrix multiplication to do the calculation for us.

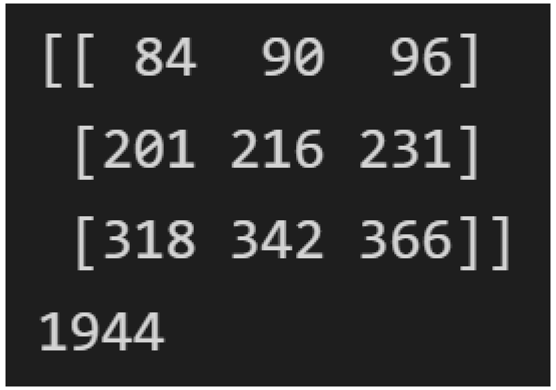
$$\begin{pmatrix} 3 & 4 & 2 \end{pmatrix} \times \begin{pmatrix} 13 & 9 & 7 & 15 \\ 8 & 7 & 4 & 6 \\ 6 & 4 & 0 & 3 \end{pmatrix} = \begin{pmatrix} 83 & 63 & 37 & 75 \end{pmatrix}$$

NP Sum.

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



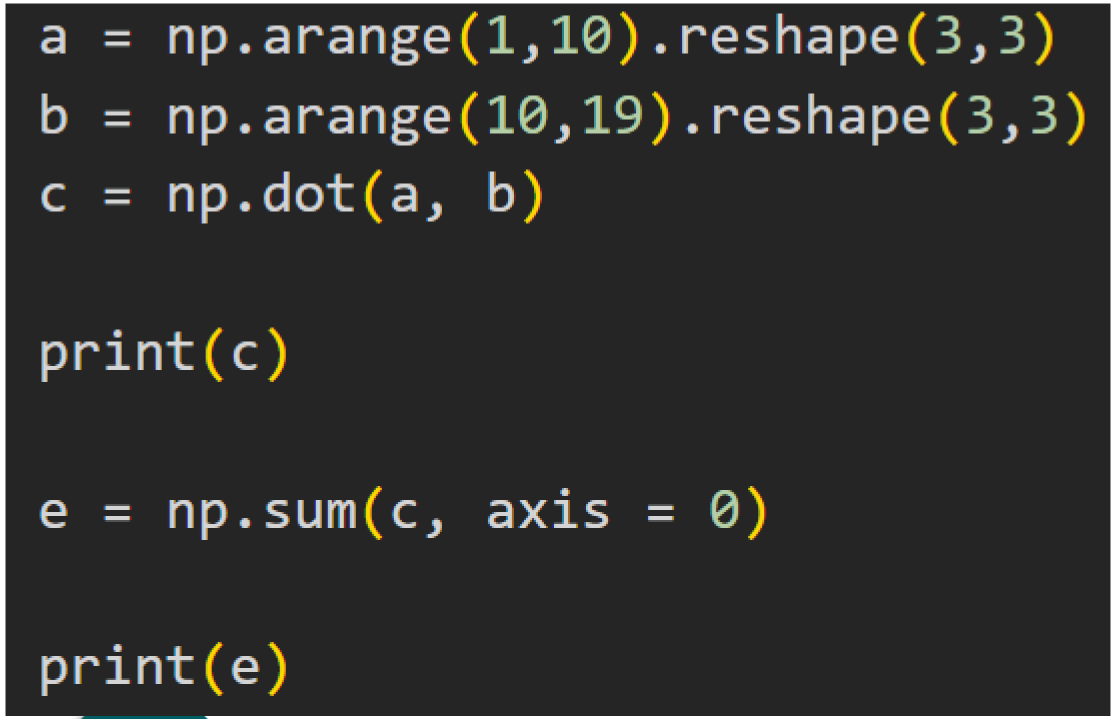
```
a = np.arange(1,10).reshape(3,3)
b = np.arange(10,19).reshape(3,3)
c = np.dot(a, b)
print(c)

total_sum = np.sum(c)
print(total_sum)
```



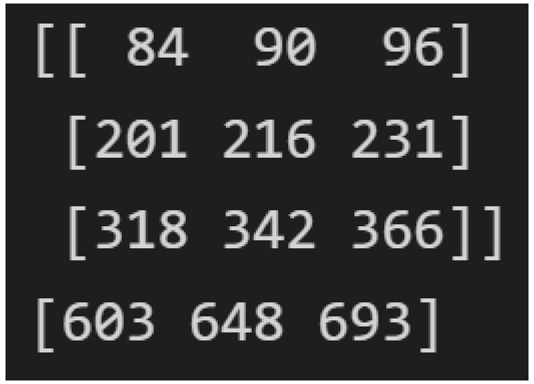
```
[[ 84  90  96]
 [201 216 231]
 [318 342 366]]
1944
```

Sum by Column.

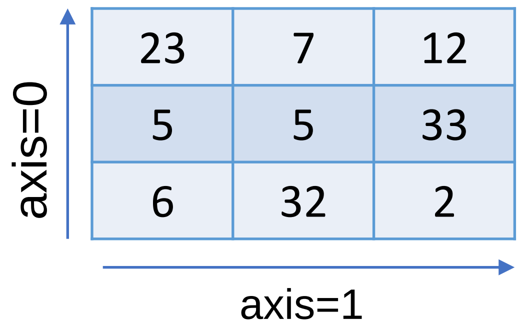
This preforms an overall sum of all the elements in the array column – adds them together!



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

print(c)

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

print(e)
```



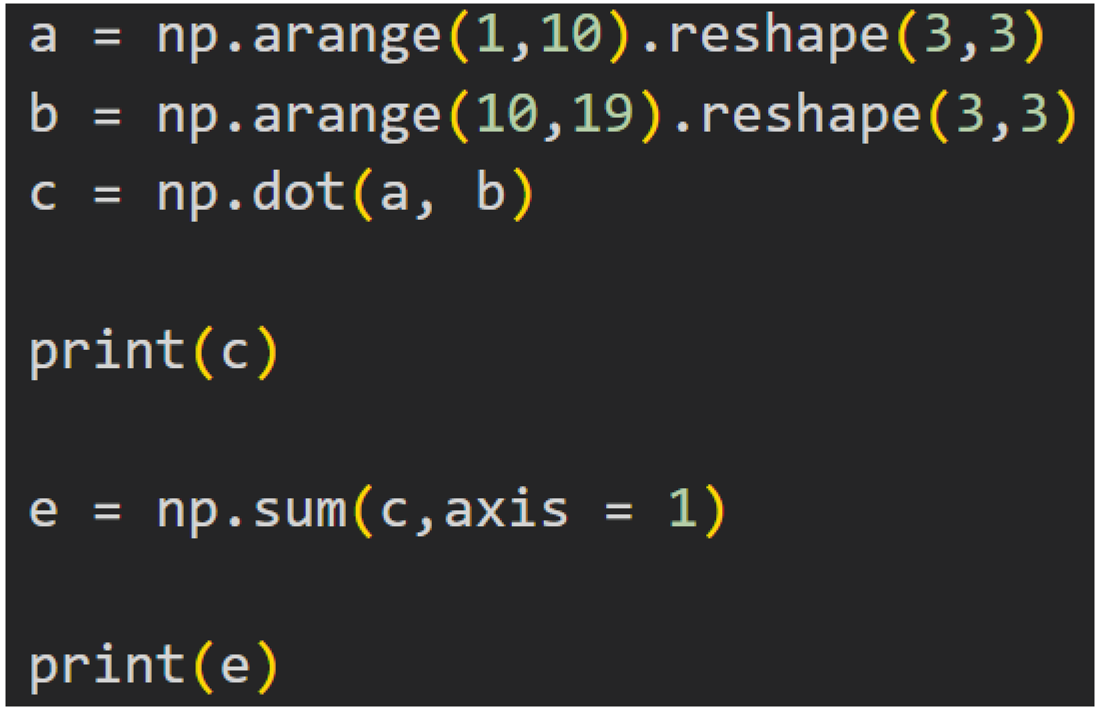
```
[[ 84  90  96]
 [201 216 231]
 [318 342 366]]
[603 648 693]
```



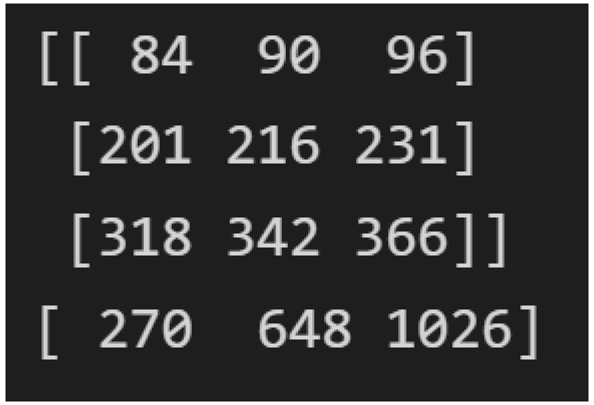
23	7	12
5	5	33
6	32	2

Sum by Row.

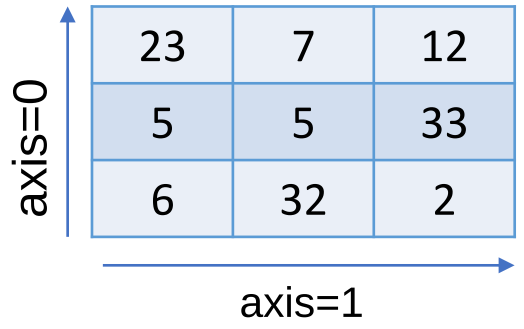
This preforms an overall sum of all the elements in the array row – adds them together!



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

print(c)


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


print(e)
```



```
[[ 84  90  96]
 [201 216 231]
 [318 342 366]]
[ 270  648 1026]
```



23	7	12
5	5	33
6	32	2

Task: NumPy arithmetic operations.

Take a look at the NumPy documentation

- Practice using the different arithmetic operations

https://numpy.org/doc/stable/reference/routines.math.html#arithmetic-operations

Or

- Practice matrix multiplication

http://matrixmultiplication.xyz/

Mean, Median & Mode.

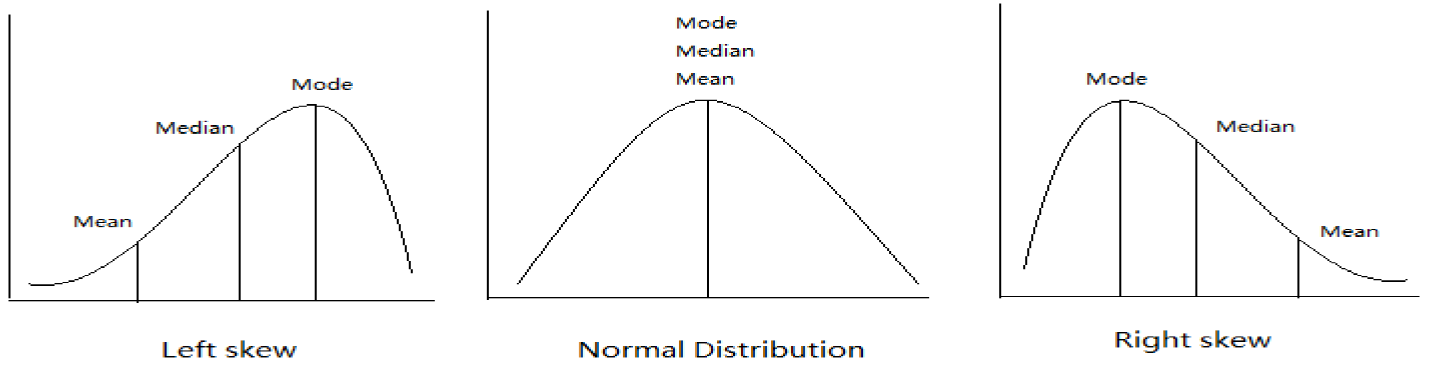
Mean, median and mode are a type of average value, which describes where centre of the data is located.

Mean: It is usually referred to as '**the average**'. The mean is the sum of all the values in the data divided by the total number of values in the data.

Median: is the **middle** value in a data set ordered from low to high.

Mode: The mode is the value(s) that are the most **common** in the data.

Why do we calculate these values?
The centre of the data points is usually a good summery of the type of data we can expect from the group as a whole. But! Be careful, it can also be misleading….

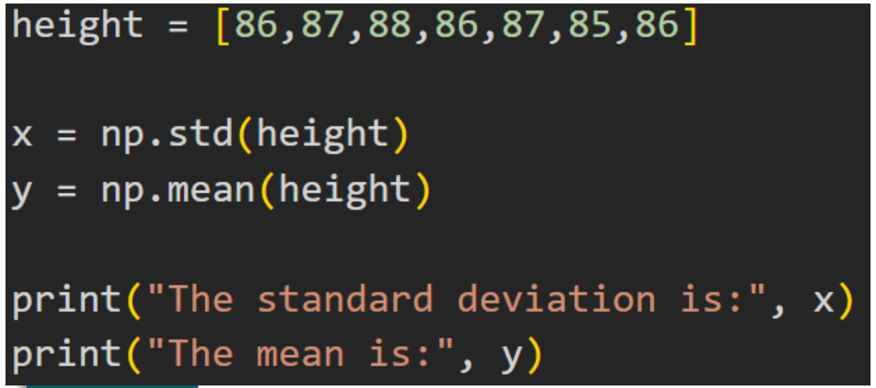


Standard Deviation.

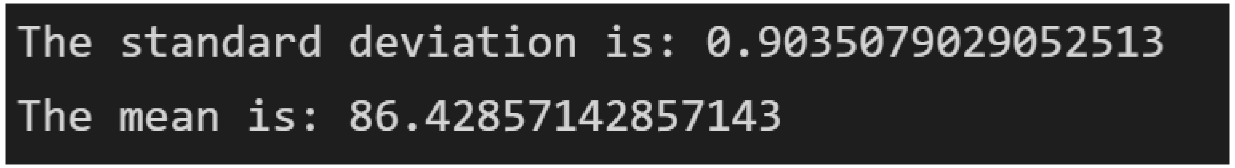
- Standard deviation is the most commonly used measure of variation, which describes how spread out the data is or the average amount we expect a point to differ (or deviate) from the mean.

- Low standard deviation means data are clustered around the mean, and high standard deviation indicates data are more spread out.

- By knowing where the centre is located and how spread out the values are, we can gain a good understanding of the distribution of values in any dataset.



```python
height = [86,87,88,86,87,85,86]

x = np.std(height)
y = np.mean(height)

print("The standard deviation is:", x)
print("The mean is:", y)
```



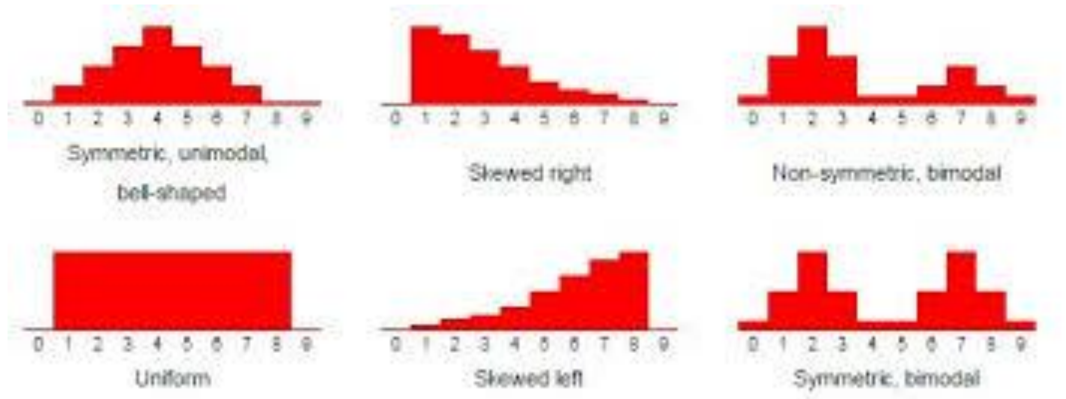
```
The standard deviation is: 0.9035079029052513
The mean is: 86.42857142857143
```

Data Distribution.

- Data distribution is all the possible values of the data. It also tells you how often each value occurs.

- Often, the data in a distribution will be ordered from smallest to largest, and graphs and charts allow you to easily see both the values and the frequency with which they appear.

- From a distribution you can calculate the probability of any one particular observation in the sample space, or the likelihood that an observation will have a value which is less than (or greater than) a point of interest.



Types of Distribution.

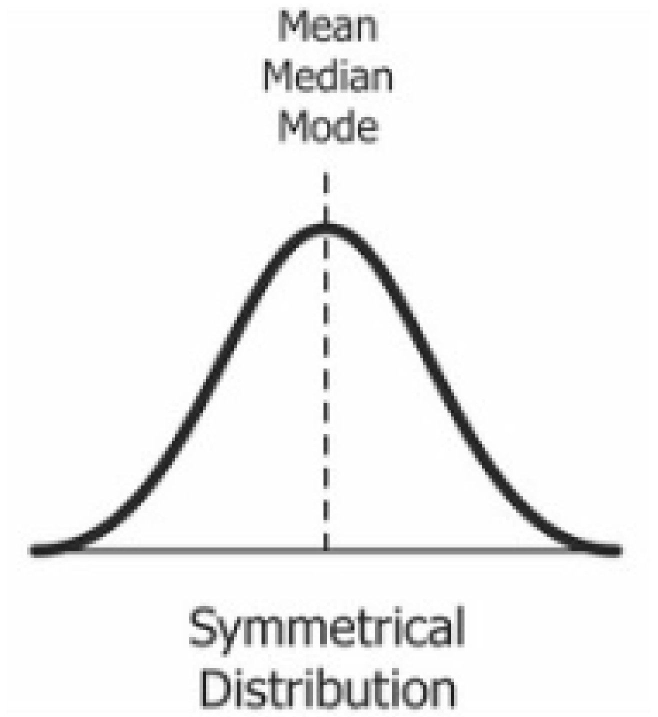
When collecting data to make observations about the world it usually just isn't possible to collect all the data. So instead we take a sample of the population, and then use the shape of our samples to make inferences about the true underlying distribution our data.

- Normal
- Binomial
- Uniform
- Logistic
- Poisson
- Multinomial
- Exponential
- Chi square
- Rayleigh
- Pareto
- Zipf

https://thecrashcourse.com/courses/the-shape-of-data-distributions-crash-course-statistics-7/

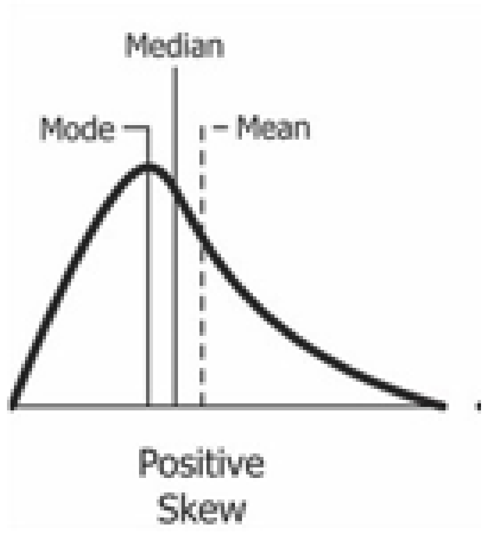
Normal Distribution.

- Many real world examples of data are normally distributed such as IQ, height and blood pressure
- It is often referred to as a 'bell curve' because of it's shape
- Most of the values are around the centre
- The median and mean are equal
- It has only one mode
- It is symmetric, meaning it decreases the same amount on the left and the right of the centre

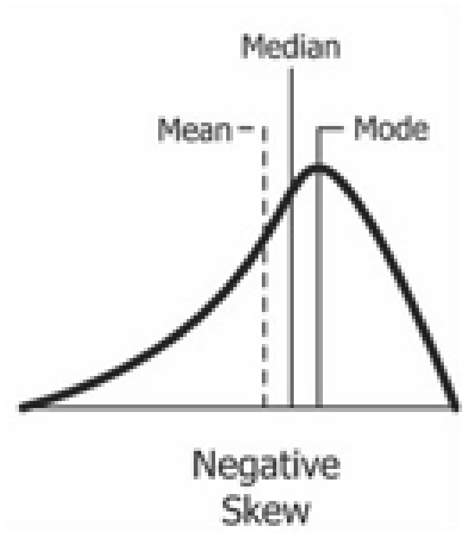


Positive and Negative Skews.

In a positively skewed distribution, the mean is greater than the median as the data is more towards the lower side and the mean average of all the values. In contrast, the median is the middle value of the data.



In a negatively skewed distribution, the mean is lower than the median as the data is more towards the higher side and the mean average of all the values.

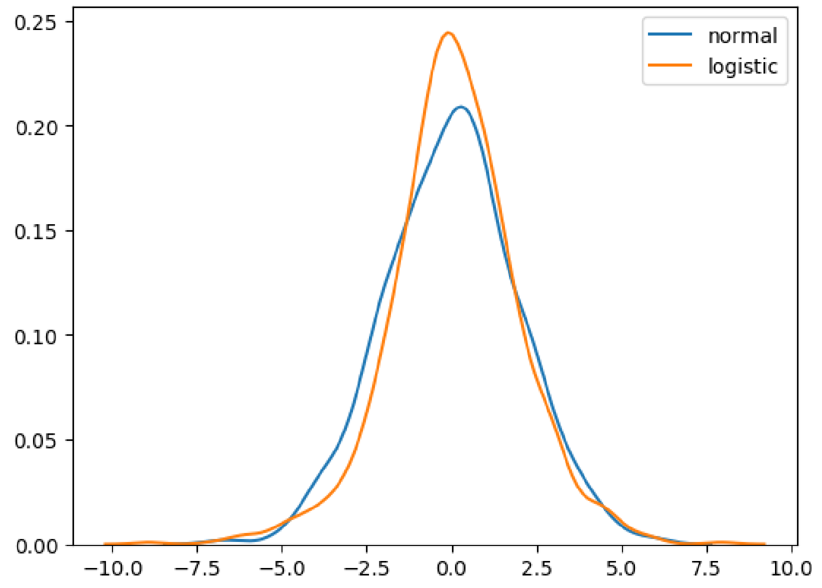


Logistic Distribution.

Logistic Distribution is used to describe growth.
Used extensively in machine learning in logistic regression, neural networks (which we cover later in the course).

It can be used to describe population growth over time or chemical reactions. This distribution is symmetrical and takes 2 parameters: the mean or the average value and the scale, controlling the variance.

Normal and logistic distribution are near identical, but logistic distribution has more area under the tails, meaning it represents more possibility of occurrence of an event further away from mean



Plenary.

techtalent academy

Insert the correct syntax for printing the number 50 from the array.

```
arr = np.array([10, 20, 30, 40, 50])

print(arr     )
```

Use a correct NumPy method to change the shape of an array from 2-D to 1-D.

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

newarr = arr.
```

Use the correct NumPy method to find all items with the value 4.

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

x = np.       (arr == 4)
```