NUMPY

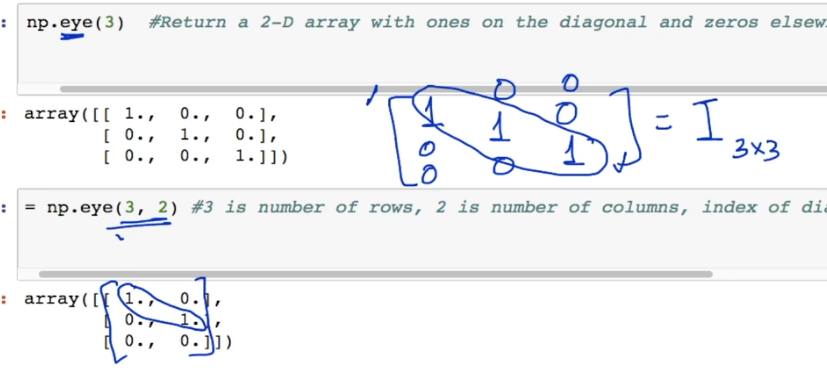
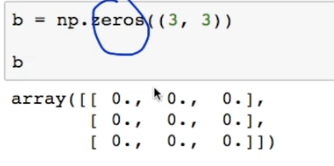
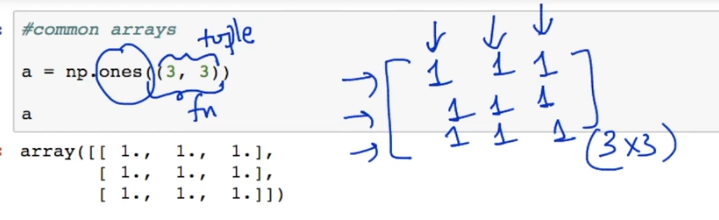
The numpy is a library in python that help us manage array. It is faster and more memory efficient than list.

1D array is called **vectors**

2D array is called **Matrix**

**Above this**  they are called **Tensors.**

Below are the list of functions that have been in the introduction lecture of numpy which can later be consulted from documentation of the **numpy. np.arange(), shape(), dim(), np.linespace(), np.ones(),**  I will snippet the snaps from the lecture below

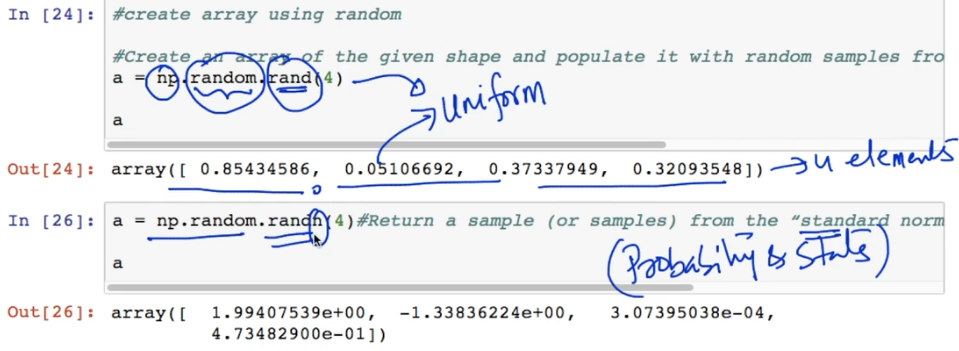


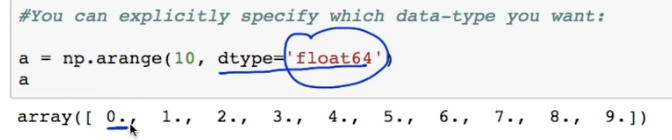
By default the output of np.zeros is in the float dtype.

if I want to see the diag element I just write **np.diag(a)**.

Sometimes in data science we need to have an array of random numbers these random number are of different types the one which we are going to see below are of *uniform distributed random number* and  *gaussian or standard normal random numbers* .

We will later learn about them in the probability section

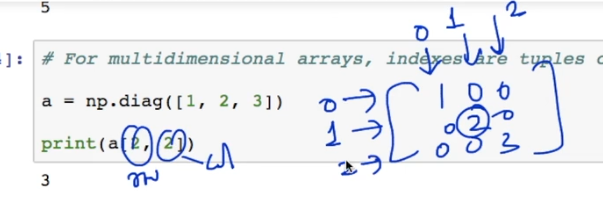
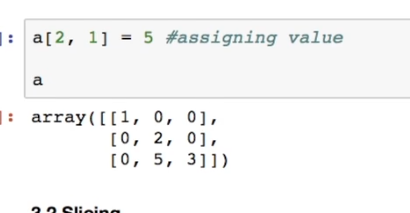


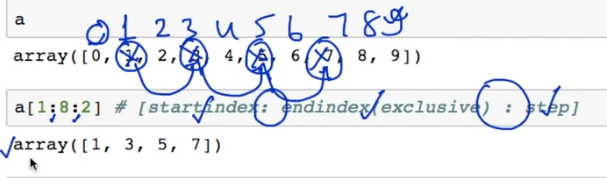
When we want to have the dtype of the array that we have created using np to be of type float then we do the following else bydefault its of type **int64.**

the float is represented by the decimal after the number eg **0.0, 1.0…etc**

We also have dtype as complex, bool , string as s6.

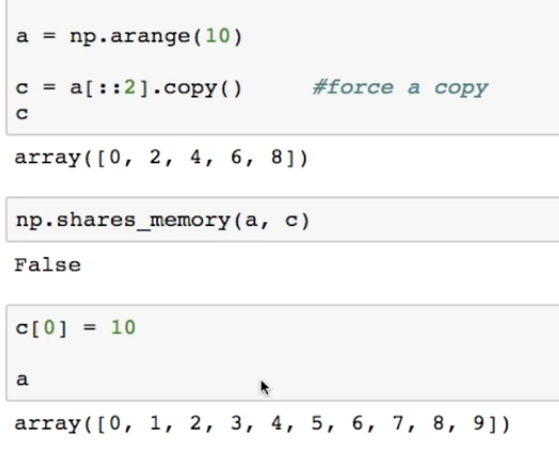
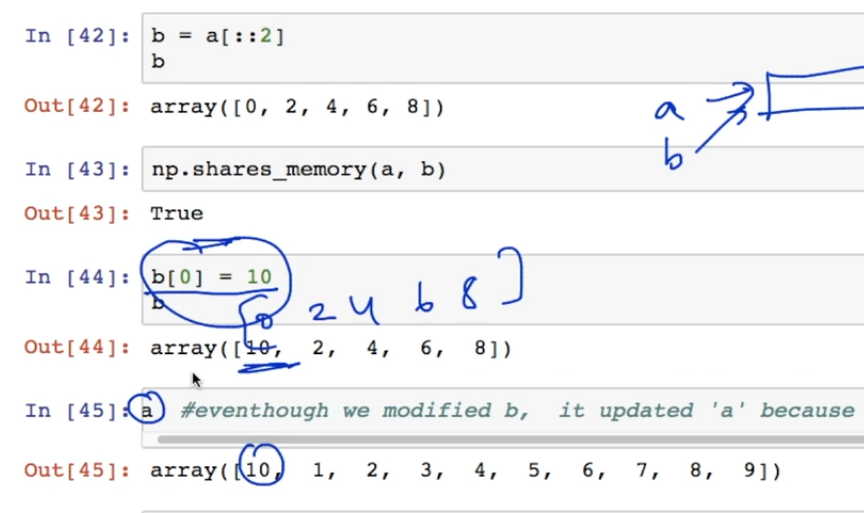
# Indexing and Slicing

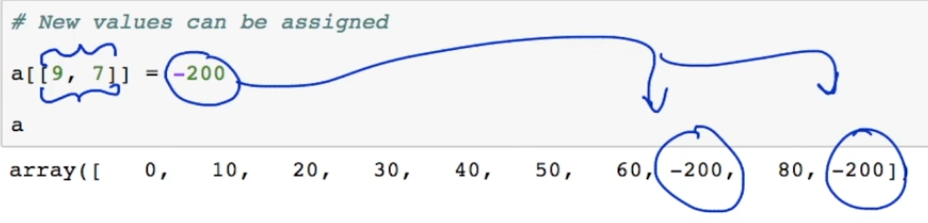


Here we see some slicing

In Numpy we see some memory efficient assignment. For instance we an array that is strored in some memory location in the computer then we have another variable in which we wish to store the sub array of the first array so in order to save space what the numpy does is it makes the second variable points to the memory location of the first array.

In the below image it can be seen how we have a var **a and b**. To be memory efficient how the memory space is shared and when we modify the value at one place the value at the other space also get modified.

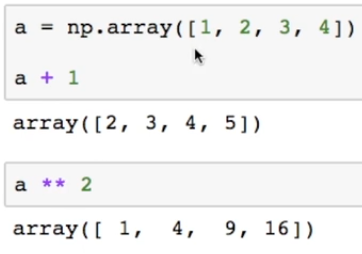
If we want to change this approach of the numpy and want to have a separate memory space for the second variable then we will do the following.

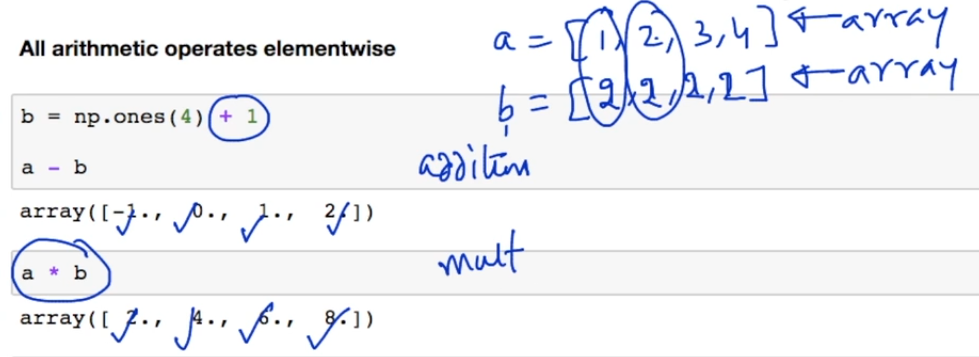


We can also modify or assign value to an already declared array using **[ ]**

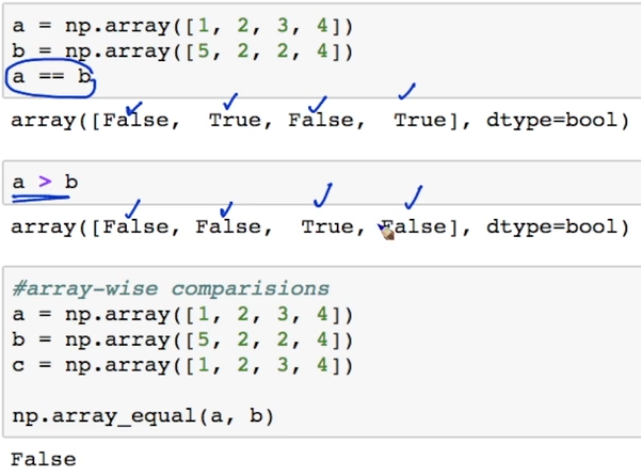
Numerical Operations on Numpy

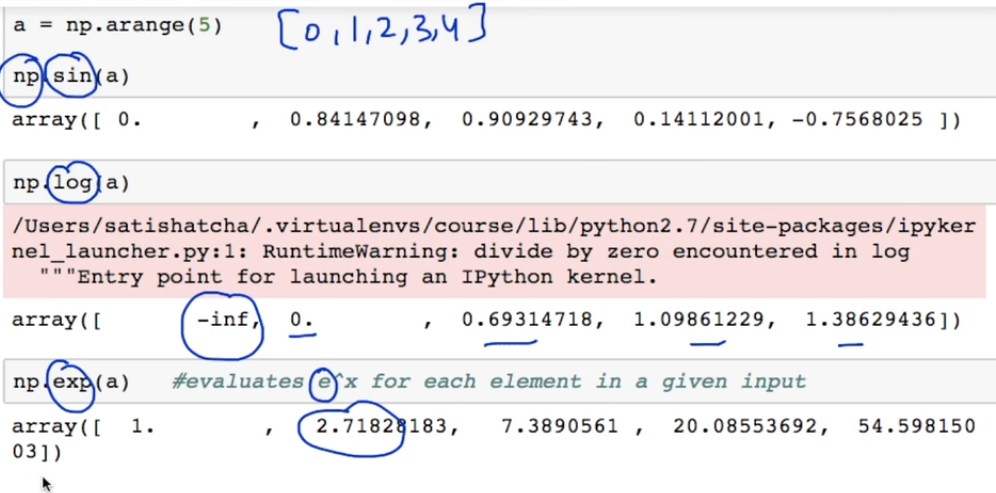
# Basic operations with Scalers





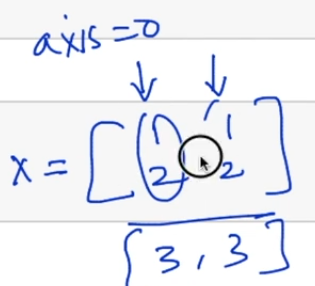
In this section I feel the screenshots are sufficient enough and whereever further explanation might be needed I will add a side note to it.

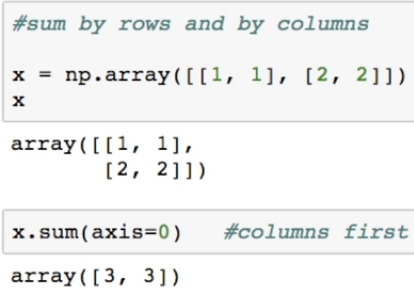


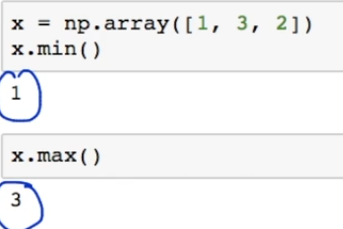
Below we have the mathematical functions

# Basic Reductions

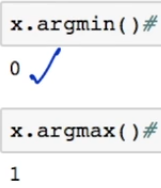
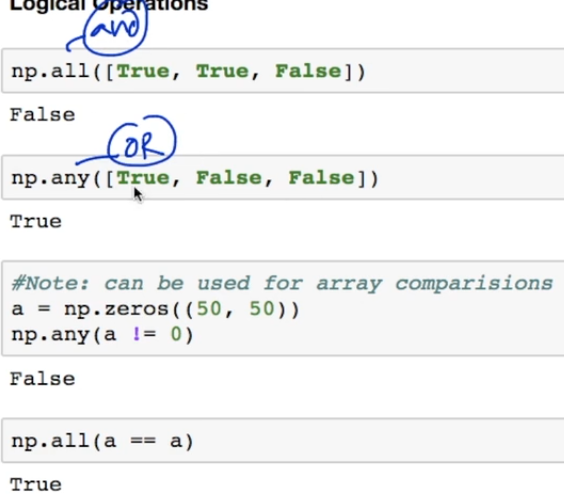
This is the sum of all the elements in the array







When we have an array the index at which the smallest element stands is called as the **argmin** where as the index at which we have the largest element is called **argmax**

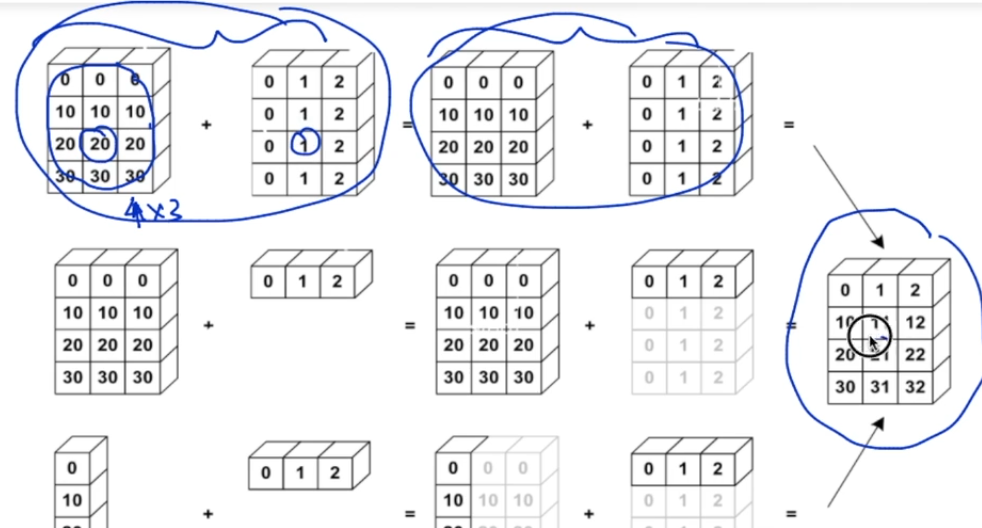


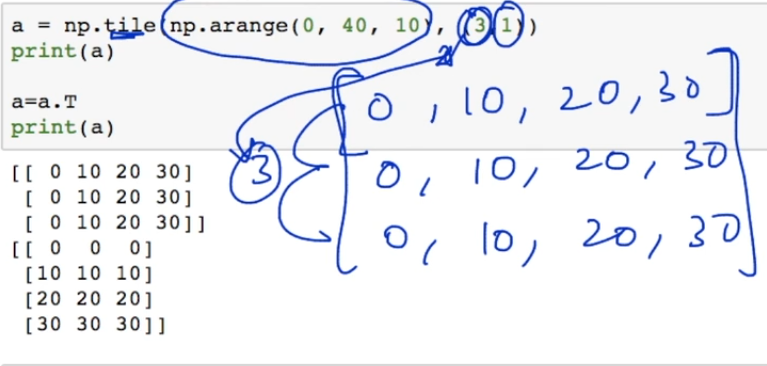
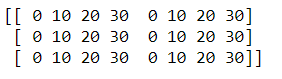
50x50 is a matrix of zeros and using the any() we check do we have any single element in this matrix which is not zero as the ans is no therefore we get false.

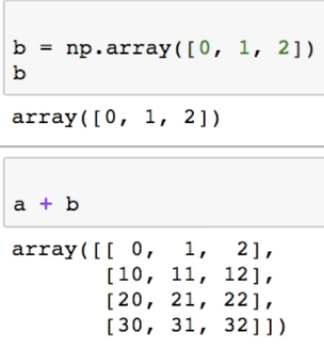
# Broadcasting

When the axis=0 it means coloums when the axis=1 it means rows

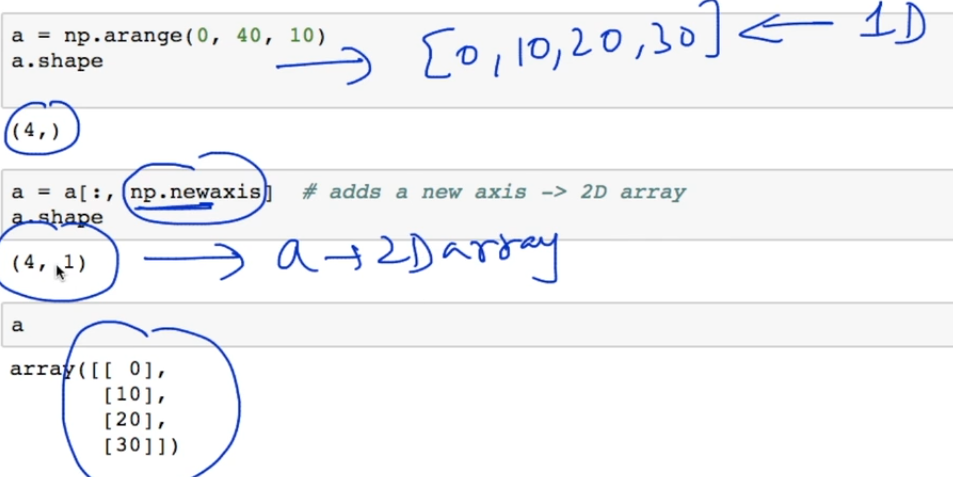
Broadcasting means that when the array adjust itself according to the size and shape of the array or matrix with which it has to perform some operation so that that operation can be performed which otherwise would have not been able to be performed due to the size and shape incompability.

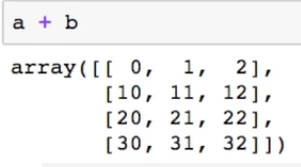
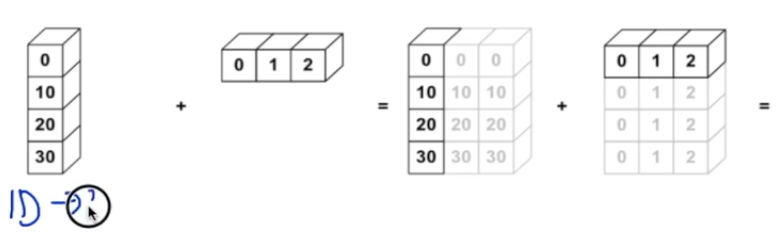
Here it is shown how the tree paths leads to the same result but in the second and the third path the concept of broadcasting has been used.

Here we are piling up the array using the tile function else the output would have been as follows Similarly the 1 in the (3,1) tells how many coloums we need if it would have been 2 we would have had the following result



The concept of broadcasting that we discussed above. Here is the programmatic implementations of second path of the idea. In this section of the code we can see how the second array adjust itself when we do **a+b** in order to become compatible with the size and shape of **a** so that the addition operation can be performed here.



Here what you have to focus on is the **np.newaxis.**  here we had a 1D array we converted it into a 2D array. The transpose option will not work here. Then we just simply add It to the b thus showing the third path of the **broadcasting concept**

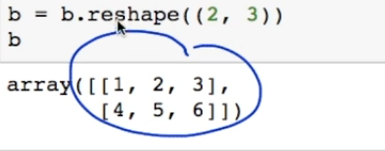
# Array Shape Manipulation

# **Flattening**

Here we convert a an array whether 2D or 3D in a 1D array. We use the function **ravel()** for this purpose

## Reshapping a Matrix

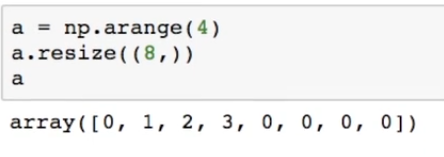


One thing that we should be aware of that numpy tries to be memory efficient so for example here **b** might not have been assigned a sperate memory space but might be using the same memory space of **a**

**Reshaping and Resizing**

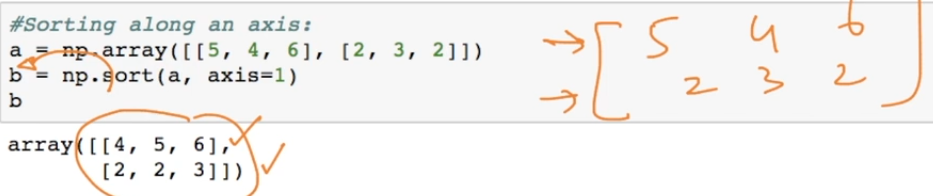
Now we will see some useful information related to reshaping and resizing. *One thing that you should keep In mind is that a bug can occur while reshaping matrices with regards to the memory location*. It will not be always that to be memory efficient the reshaped matrix to be pointed to the old memory location.

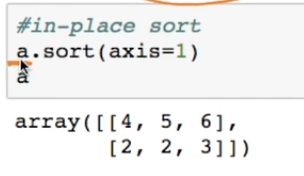
Below we see some more ways in which **reshape()**  is used in data science. For instance we have to construct a tensor 4x3x2 ( having 4 matrix of size 3x2) then to do it using **reshape()**

**Resize** is used to increase or decrease the size of the array. When we increase the size of the array then in such a case the space is filled with zeros.

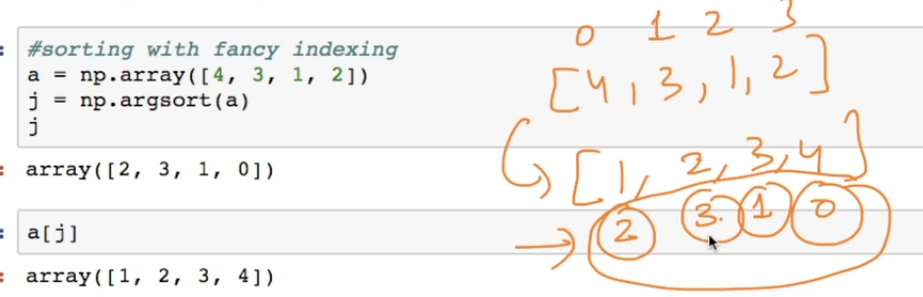
It could also be written as *a.resize(8*)

**Sorting**

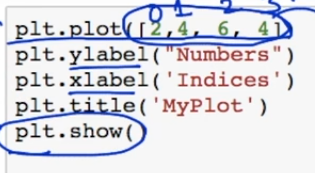
Here we are moving the sorted array into another varaibale



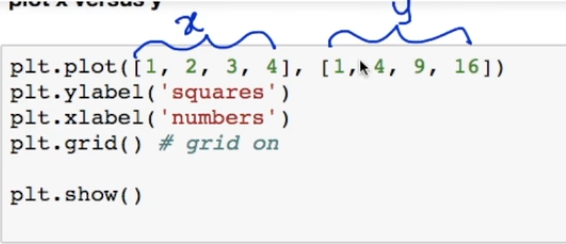
Here sort is happening in the main matrix

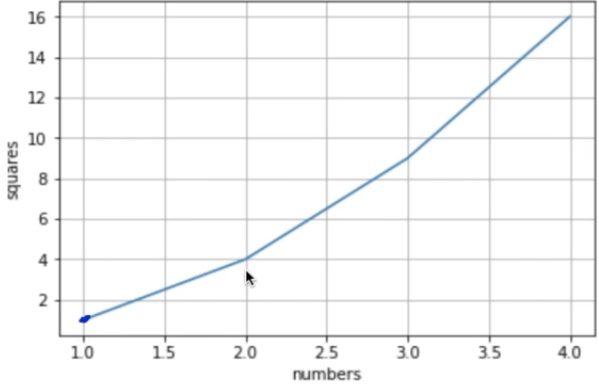


Here **np.argsort(a)** returns the index of the element of the array in the sorted form which is stored in j which is later used to access the element in the sorted form

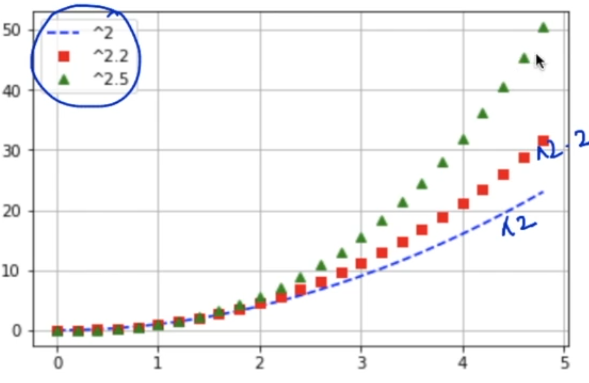
MATPLOTLIB

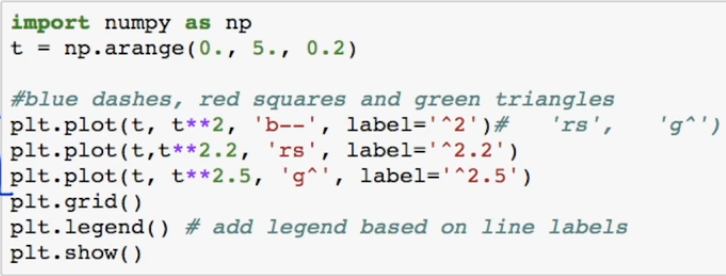
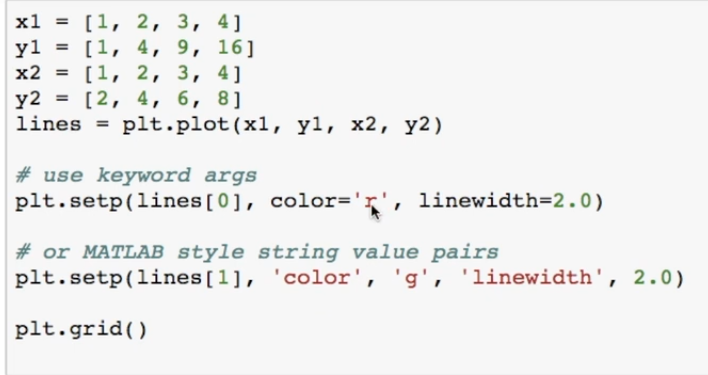
It works like matlab. When we don’t give the value of the either of the axis and we have given the values for one of the axis then the value of the other axis becomes the indexes of the given values

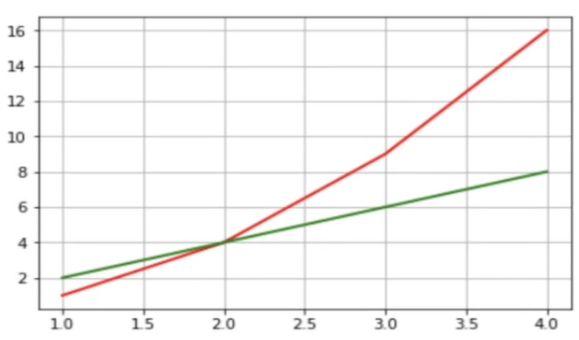


We can also have the grid in the plot that is displayed for that we have the following code.

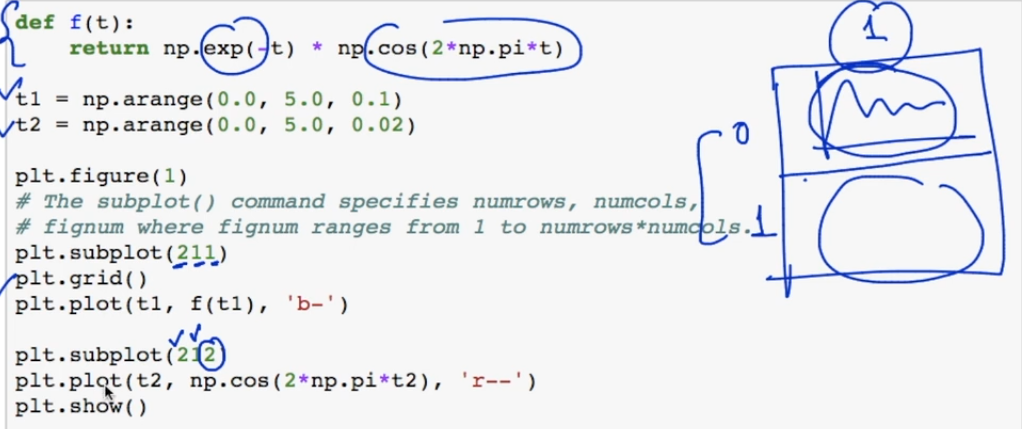
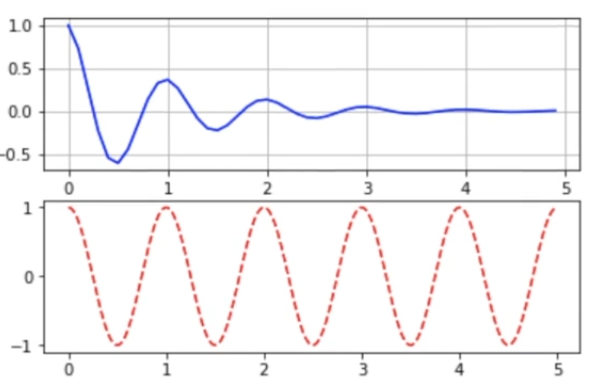
If we want to change the color of the line or replace the lien with points then we have the following line of code where **r** reperesnts **red**  and **o** represents **dot**

Now further in the lecture we explore more ways to modify the line into different forms such as dash , triangle or even square. We also add a legend by using the label mark. Below I have given the code and the output which I think should be sufficient for revision in future.



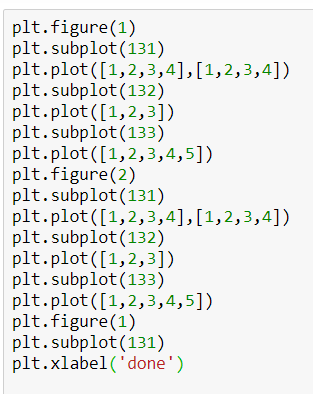
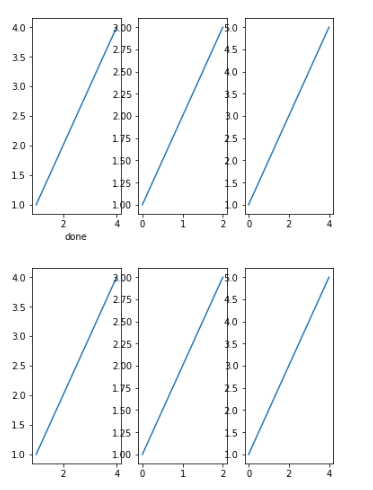
We now look at what happens when we have more than 2 sets of values here the line of x1 y1 and x2 y2 are stored in the variable lines. We can deal with both the lines individually using **setp()** and the result will be show in common graph where both the lines will be visible

Now we have something quite interesting on the next page. What if we want to have many graph ploted and shown. So for this we declare a **figure** that can have multiple plots. The figure can be divided into rows and coloums.



This refers to the section of the figure that we are dealing with

Here we will see on how to play around with figure() and subplot() I have practised in multiple times on the jupyter here I will only paste the code from the lecture



This is the code and the output from my Jupyter notebook here I have use the figure and subplot to arrange the plots

PANDAS

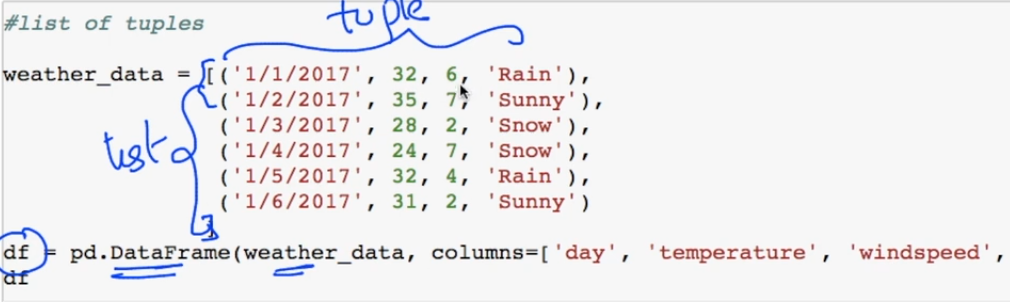
# **Getting Started with Pandas**

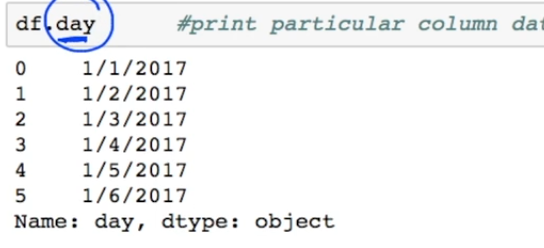
The first lecture seemed sufficient enough and did not require notes.

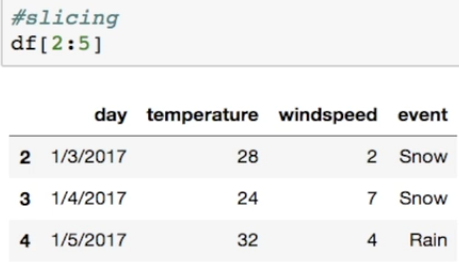
# **DataFrame basic**

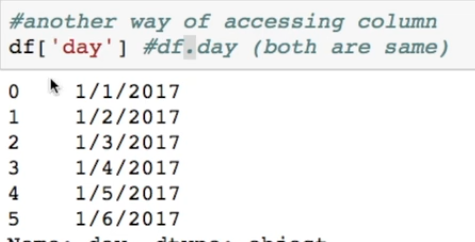
It is data structure similar to a table. I will include screen shots from the lecture that will be sufficient enough to understand the working of the data frame.

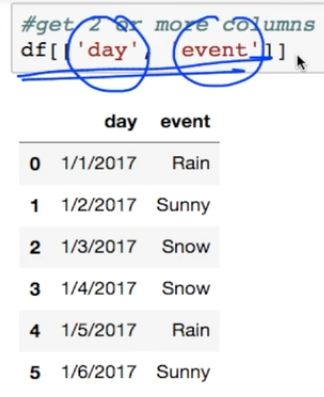
**CSV stands for comma separated values** as they are separated by comma.

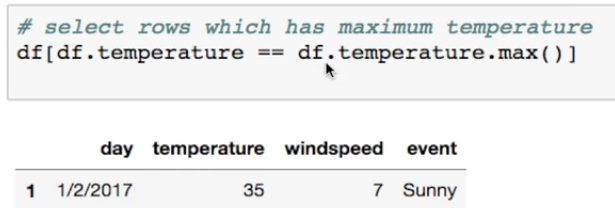
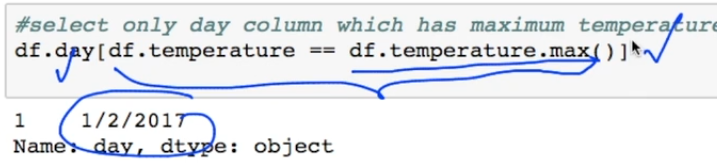
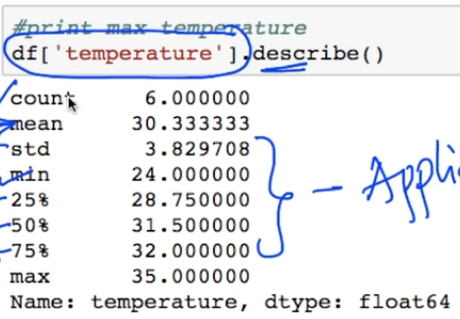
We can also construc a data frame as follows

Here I will mention some of the functions that are quite useful in pandas **pd.head(), pd.tail(). Pd.shape()**  to slice we do the following.



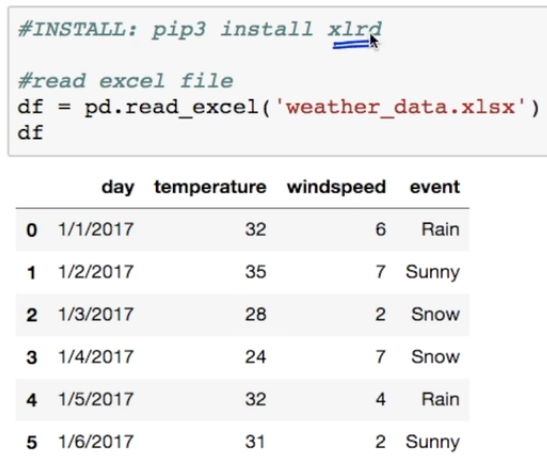




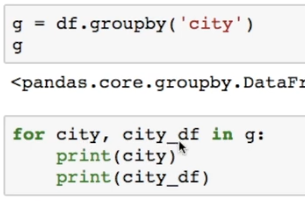
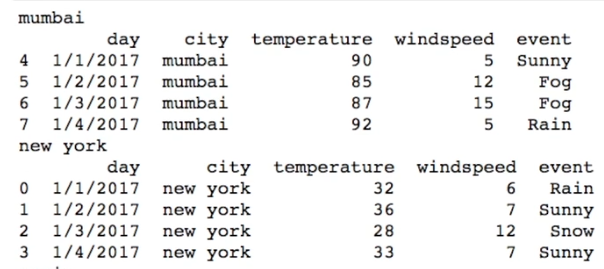


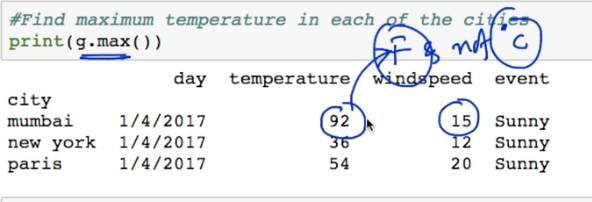
When we want to get all the info about the coloumn which include the average , standard diviation , total count we use the function describe()

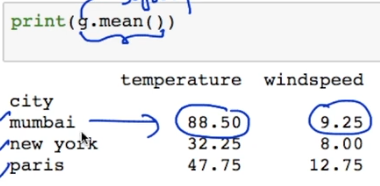
# **Key Operations on DataFrame**

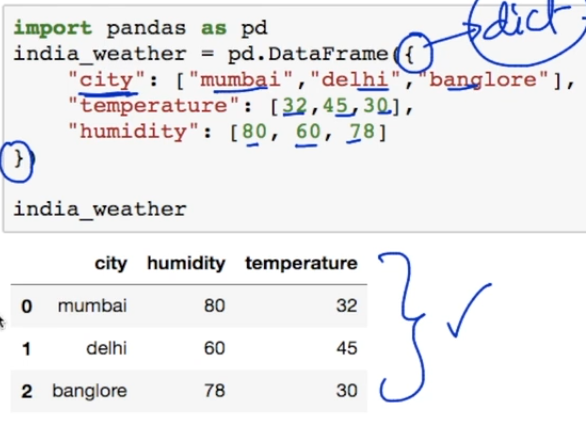
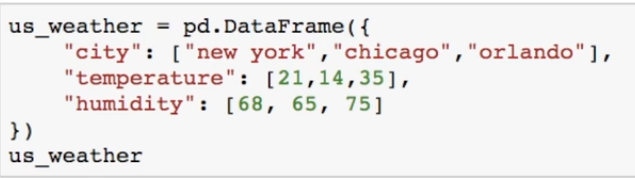


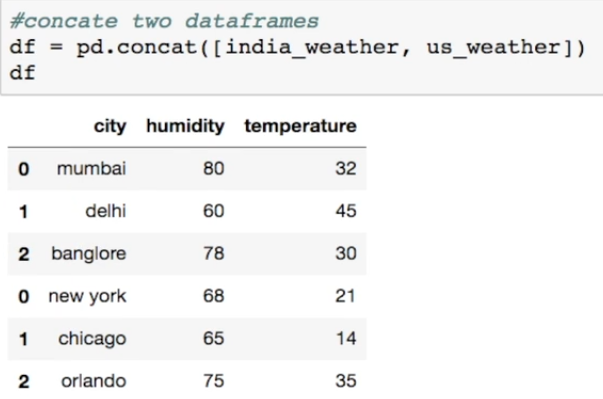
Above is how to store the data into csv and excel file

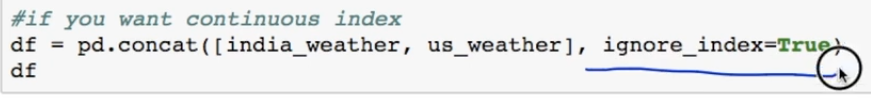
We have a dataset in which we have been given details about the weather of the different cities so we have used the groupby function inorder to help us organize the data through the cities

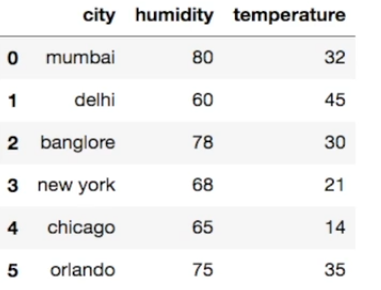
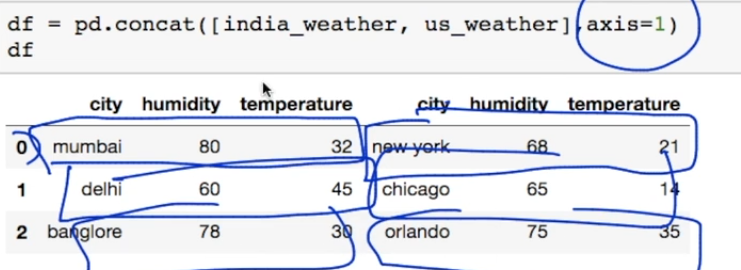




This is how we make the Dataframe using Dict

We concat both the above dataframe as follows

If we wish to have the proper indexing then we do the following

 If we want to have the tables concat coloumn wise unlike row wise that is done before then do the following

## **Merging of DataFrame**

