NumPy Tutorial

[***How to create a basic array*** 2](#_Toc94134668)

[***Adding, removing, and sorting elements*** 2](#_Toc94134669)

[***How do you know the shape and size of an array?*** 3](#_Toc94134670)

[***Can you reshape an array?*** 4](#_Toc94134671)

[***How to convert a 1D array into a 2D array (how to add a new axis to an array)*** 4](#_Toc94134672)

[***Indexing and slicing*** 5](#_Toc94134673)

[***How to create an array from existing data*** 7](#_Toc94134674)

[***Basic array operations*** 8](#_Toc94134675)

[***Broadcasting*** 9](#_Toc94134676)

[***More useful array operations*** 10](#_Toc94134677)

[***Creating matrices*** 10](#_Toc94134678)

[***Generating random numbers*** 12](#_Toc94134679)

[***How to get unique items and counts*** 12](#_Toc94134680)

[***Transposing and reshaping a matrix*** 13](#_Toc94134681)

[***How to reverse an array*** 13](#_Toc94134682)

[***Reshaping and flattening multidimensional arrays*** 14](#_Toc94134683)

[***How to access the docstring for more information*** 15](#_Toc94134684)

[***Working with mathematical formulas*** 16](#_Toc94134685)

[***How to save and load NumPy objects*** 16](#_Toc94134686)

[***Importing and exporting a CSV*** 17](#_Toc94134687)

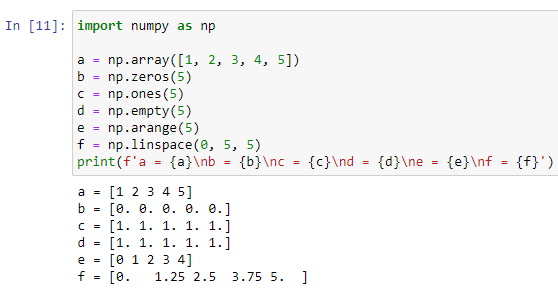
[***Plotting arrays with Matplotlib*** 18](#_Toc94134688)

# How to create a basic array

To create a NumPy array, you can use the next function:

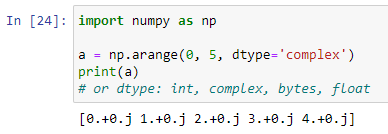
*np.array, np.zeros, np.ones, np.empty, np.arange, np.linspace*

For examples:



You can explicitly specify which data type you want using the ‘*dtype’* keyword.

For examples:



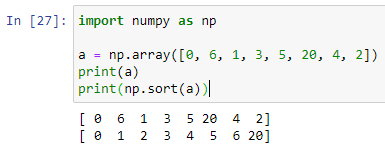
# ***Adding, removing, and sorting elements***

Useful functions:

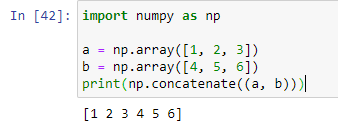
*np.sort, np.concatenate, np.argsort, np.lexsort, np.searchsorted, np.partition.*

With *np.sort* you can specify the axis, kind, and order when you call the functions.

Quickly sort the numbers in ascending order:

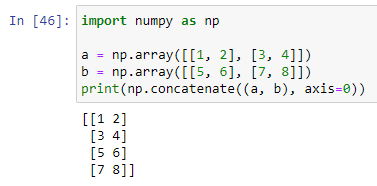
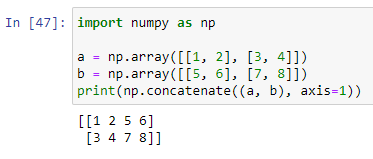


For concatenate you can use *np.concatenate*, note that passing tuple:

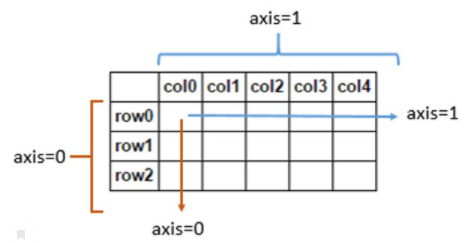


If you want concatenate n-dimensional array, you should be specify axis.

Below represented example:

Now, you can see how set axis in concatenate changing result. For better understanding, how worked parameter axis in bidimensional array, look at the picture below:



# ***How do you know the shape and size of an array?***

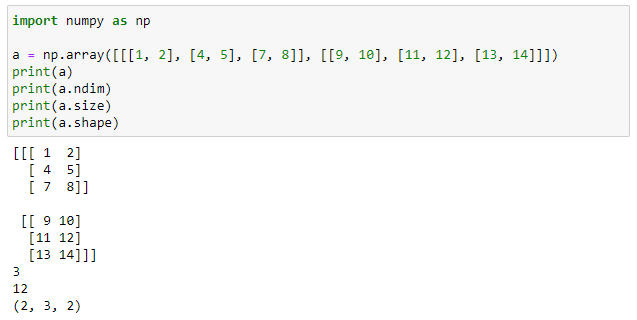
Obtain information about array: *ndarray.ndim, ndarray.size, ndarray.shape*

*ndarray.ndim –* will tell you number of axes, or dimensions of the array.

*ndarray.size –* will tell you total number of elements of the array.

*ndarray.shape –* will tell you tuple numbers that indicate rows and length it of the array.

For example:

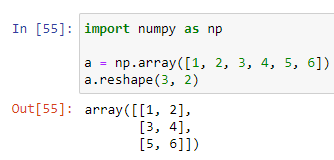


# ***Can you reshape an array?***

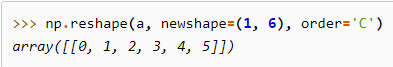
Of course, you can use this function: *arr.reshape()*

Note that, reshape`s array should have the same number of elements as the original array.

For example, we can use *reshape* for obtain array with three rows and two columns, but initial array not changing:



You can specify optional parameters: *newshape, order.*



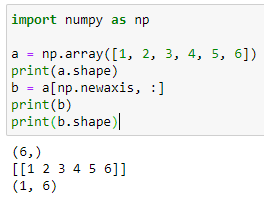
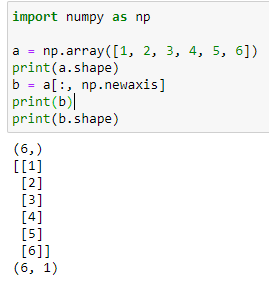
*newshape –* new shape you want. You can specify integer or a tuple of integers. If you specify and integer, the result will be a simply array of that length. The shape should be compatible with the original shape.

*order –* C means to read/write the elements C-like index order; F means to read/write the elements using Fortran-like index order and etc.

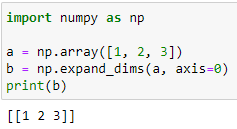
# ***How to convert a 1D array into a 2D array (how to add a new axis to an array)***

This section covers *np.newaxis, np.expand\_dims.*

The *np.newaxis* increase your existing array by one dimensional (1D->2D)(2D-3D…end etc.).

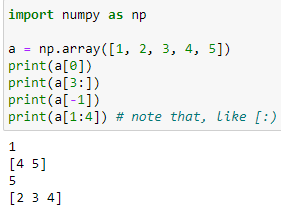
The *np.expand\_dims* like *np.newaxis*. You can specify number axis. For example:



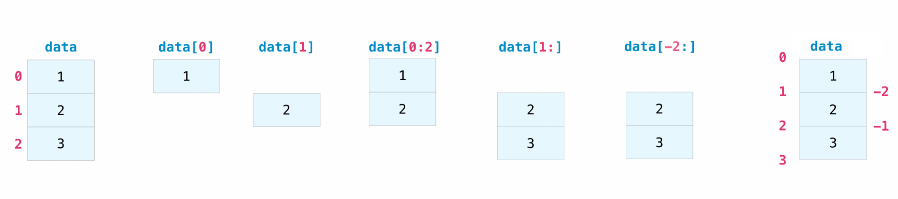
# ***Indexing and slicing***

You can index and slice NumPy arrays in the same ways can slice Python lists. If you want to take a section of your array, or slice for further operations, this part for you.

Look at some examples:



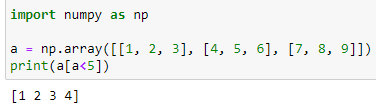
How do work it:



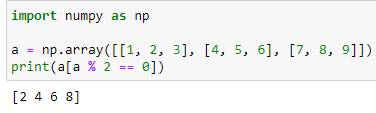
If you want print all element from your array that fulfill certain conditions. It`s very easy, you can use mathematics symbols and compose an expression. For example:

Now we will start with this bidimensional array: .

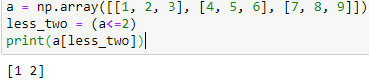
All elements that less five:



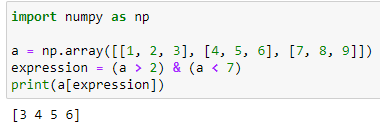
All elements that division by 2:



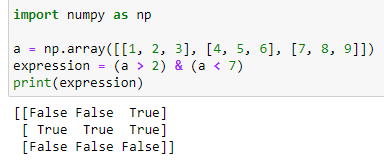
You can assign the expression: 



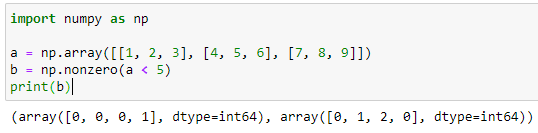
Stack your expressions assigning logical symbols: &, |



You can use this expression for return boolean grid vales specified array. This can be useful with array that contain names or other categorical values. Example:

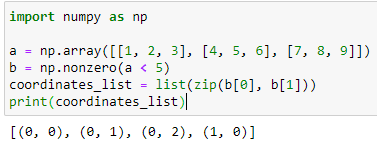


Example, if you want returned list indices elements array, that fulfill certain condition. You can use function *np.nonzero().* You can use function np.nonzero(). Function returns tuple with two array in the first array is rows, second is columns. It`s like a coordinate grid. Below pictured:

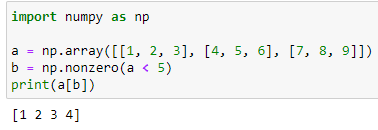


“coordinate grid”.. Take first element within first array and first element within second array and we obtain number = 1. That awesome, how easy worked with array in NumPy.

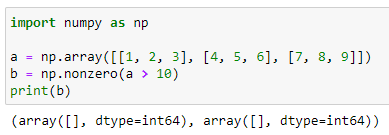
Hence, we can obtain list of coordinates. Use Python function zip [fuck.. it`s cool]:



You can also use result *np.nonzero()* for obtain array`s element.



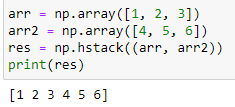
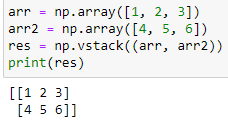
If element that you are looking for doesn`t exist in array, function returned tuple with empty array:



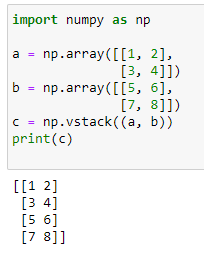
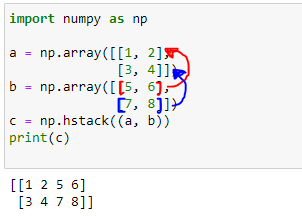
# ***How to create an array from existing data***

In this part: *np.vstack(), np.hstack(), np.hsplit(), np.vsplit(), .view(), copy().*

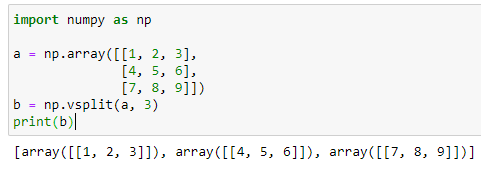
You can stack two existing arrays, both vertically and horizontally. Example on one-dimensional:

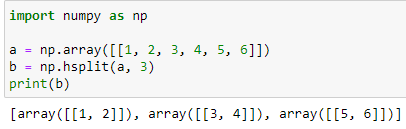
Example on bidimensional arrays:

If you wanted to split array, example, into rows, you can use *np.vsplit():*

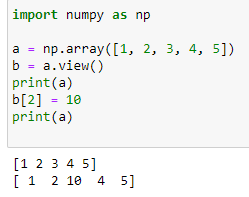


Also if you wanted split an array into several smaller arrays, you can using *np.hsplit().* You should specify number of equally shaped arrays or the colums after which the division should occur.

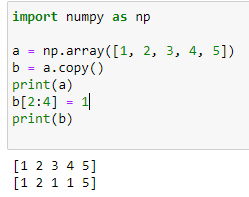


You can use view method to create a new array object which contaning same data (shallow copy). Numpy Functions, as well as operations like indexing and slicing, will return views whenever possible. This saves memory and is faster. **Note that important to be aware if this – modifying data in a view also modifies the original array!**

For example, we created view an array. Let`s try modifying element in view array:

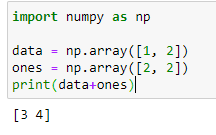


Using *copy* method will make a complete copy of the array and its data – deep copy. For example:

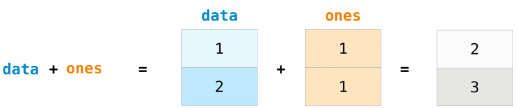


# ***Basic array operations***

You can addition arrays with the help of symbol ‘+’.

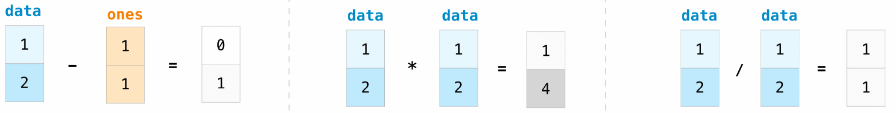


How it worked:

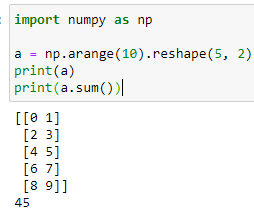


Also, you can subtraction, multiplication, division...

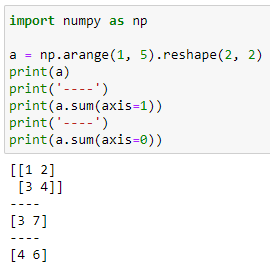
Look at picture below:



If you want to find sum all array`s element, you can use *sum* function. It works for 1D, 2D, 3D and etc.

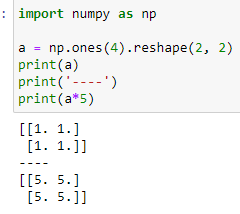
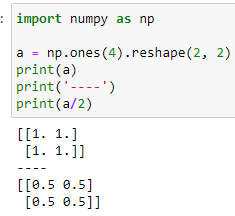
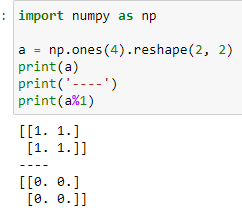


To add the rows or the columns in a 2D array, you would specify the axis:

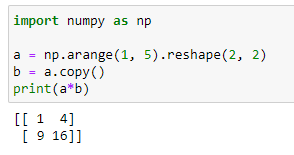
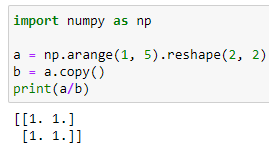


# ***Broadcasting***

You can use basic mathematic operations on array. NumPy understands that the operations should happen with each cell. That concept is called **broadcasting**. For example:

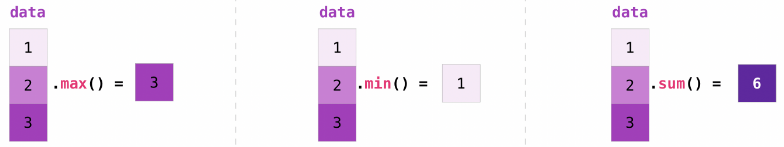
If you wanted, example, multiply an array by an array, then the dimensions of arrays must be compatible.

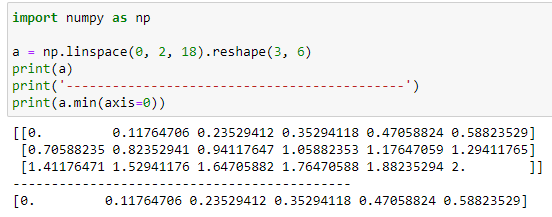
# ***More useful array operations***

NumPy also performs aggregation functions. Sush as: *min, max, sum, mean, product, std*…

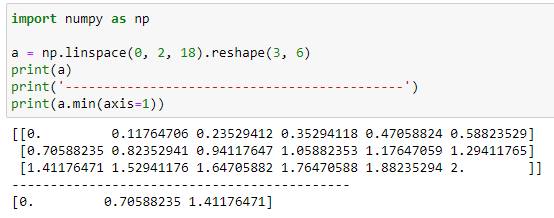
Example:



You can specify on which axis you want the aggregation function to be computed. For example, you can find the minimum value within each column, by specifying *axis=0:*



or each row:

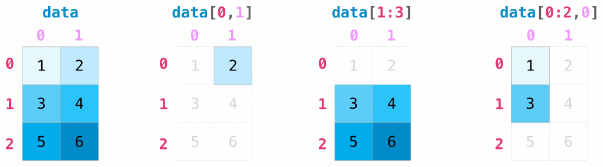


# ***Creating matrices***

If you want create matrix, you can pass python list of list in NumPy. Pictured below:

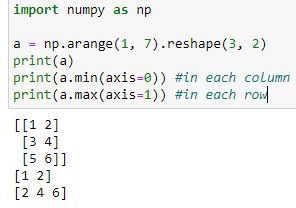


Indexing and slicing operations:

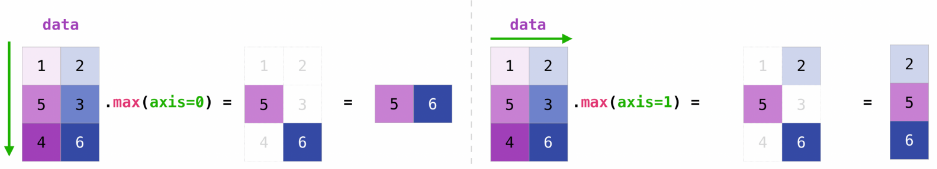


You can aggregate matrices with help: *max(), min(), sum().*

Also can aggregate across column and rows, specifying *axis* parameter:

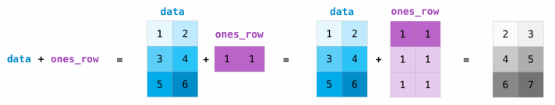


Ultra-super-maximum example for beginners:

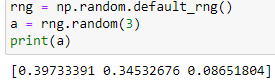


Add matrix:

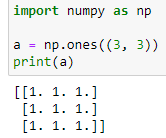


You can use arithmetic operations on matrices of different sizes , but only if matrix have one column or one row. In this case NumPy will use broadcast rules. 

If you need initialize random values of an array, you can use *np.random.default\_rng()*

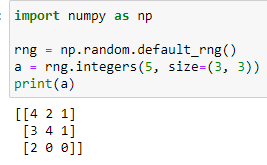


If you need to create an array special shape. You can specify tuple describing the dimensions:



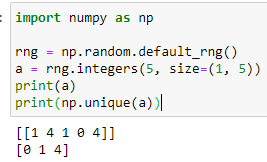
# ***Generating random numbers***

Note that, when you generate integers low-number inclusive, high-number not inclusive:

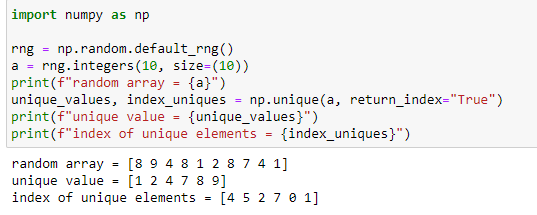


# ***How to get unique items and counts***

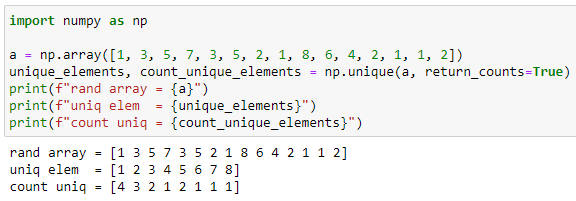
You can find unique elements in an array with *np.unique:*



If you want have index all unique elements, just pass *return\_index=True*  argument in *np.unique():*

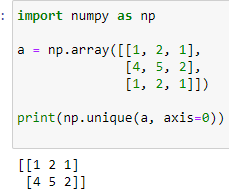
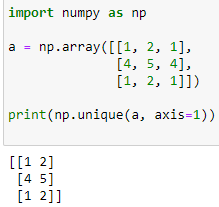


You can pass *return\_counts* argument in *np.unique()* for count of unique elements in a NumPy array:



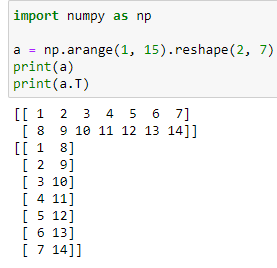
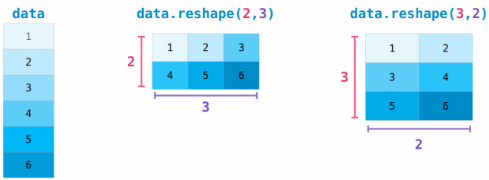
This also works with 2D arrays! O\_o . But, your, for example, 2D array will be flattened on 1D array.

If you want find unique rows or unique columns, you can pass *axis* argument. Example:

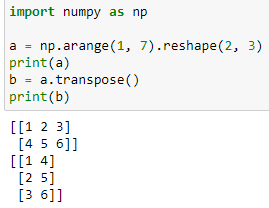
 

# ***Transposing and reshaping a matrix***

If you want transposing matrix, you can user *array.T.* Switch your array shape can be *reshape():*

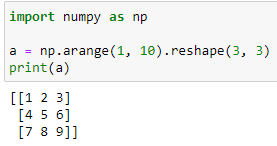
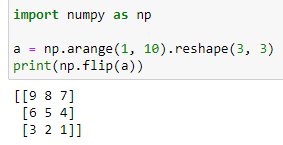
 

You can also use *.transpose()* to reverse or change the axes of an array:

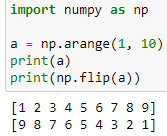


# ***How to reverse an array***

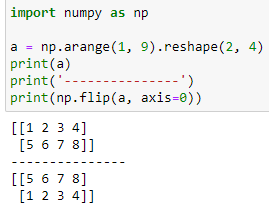
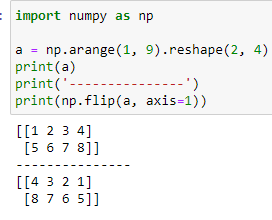
NumPy`s *np.flip()* function allows you to flip, reverse content of an array along an axis. You should specify array and axis. If you don’t specify axis, NumPy will be reverse the contents along all of the axes of your input array. Let see you how work it unspecified axis:

Now we will be reversing 1D array:



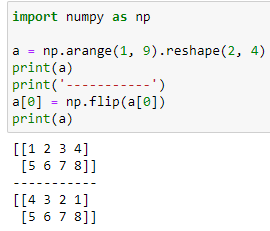
You can reverse content in all rows or columns, specify *axis* argument:

*Note that:*

* *axis=0 it`s reverse of rows*
* *axis=1 it`s reverse of columns*

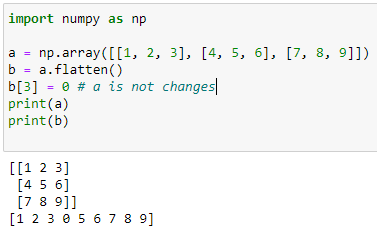
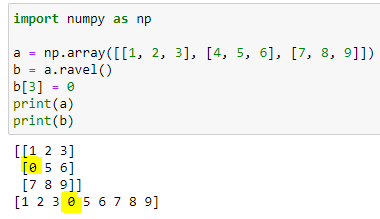
You can also reverse the contents of only one column or row. For example, flip content if the zero row:



# ***Reshaping and flattening multidimensional arrays***

This section covers: *.flatten(), ravel()*

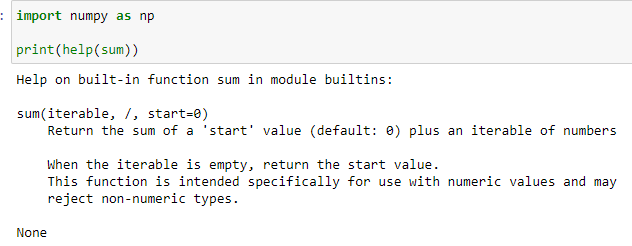
Difference between the two functions is that *ravel()* created new array. It is actually view, so change in new array will affect the parent array, be careful.

# ***How to access the docstring for more information***

This section covers: *help(), ?, ??*

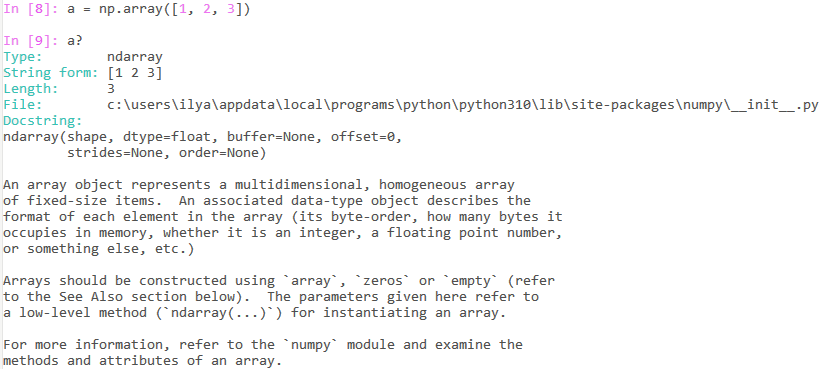
Each object in NumPy contains the reference to a string, which is know as the *docstring*. This *docstring* contains a quick and concise summary of the object and how to use it. You can use *help()* functions if you need more information that you need:



When you create your documentation, example, for the your function, you should be use string literal:

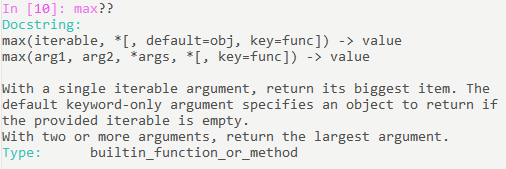
*“ “ “ text “ “ “* or *‘ ‘ ‘ text ‘ ‘ ‘*

If you use IPython, you can obtain information with help *?:*



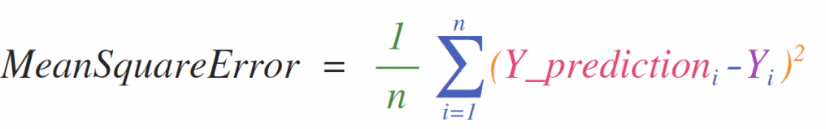
**MORE! MORE! INFORMATION TO GOD OF INFORMATION!**

You can read course code of interesting element:



# ***Working with mathematical formulas***

For example you want computing mean square error, below pictured formula:

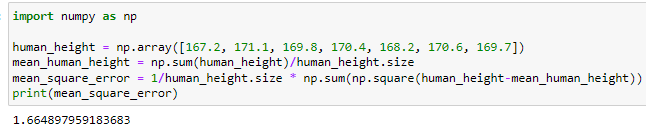


With NumPy it`s simply:



Predictions array and labels array can contain one or a thousand values, but they have same size!

For example:



# ***How to save and load NumPy objects***

This section covers: *np.save, np.savez, np.savetxt, np.load, np.loadtxt*

The ndarray object can be saved and loaded from disk files with NumPy functions.

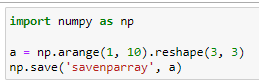
*np.loadtxt, np.savetxt –* normal text files

*np.load, np.save –* functions that handle binary files with **.npy** file extension

*np.savez –* function that handles file with **.npz** file extension

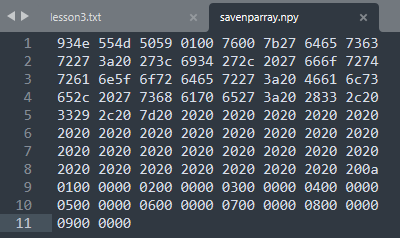
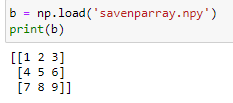
*np.savez\_compressed –* save several arrays into a single file in compressed npz format

Let`s save an array:



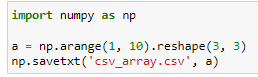
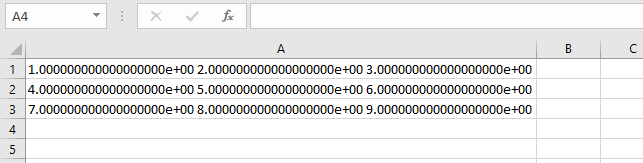
and checking the content in *savenparray:*



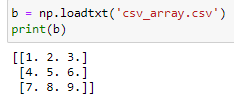
 

Now we will loading data, see up:

You can also save ndarray as a *.csv* with help *np.savetxt*:

Load:



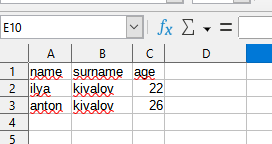
Functions *savetxt(), loadtxt()* additional optional parameters such as header, footer, and delimetr. If you need more sophisticated handling of your text file, you will want to use the *genfromtxt* function.

With *savetxt()* you can specify headers, footers, comments, and more.

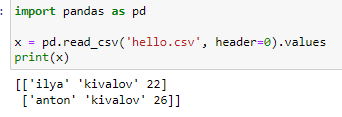
# ***Importing and exporting a CSV***

For simple to read in a CSV file, you can use the best and easiest way to do this is to use Pandas! (O..O)

If you have for example that file:

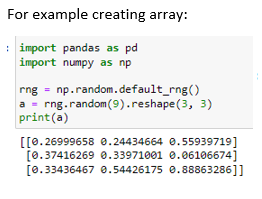
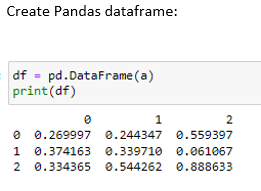
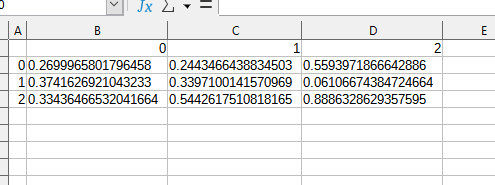


You can use *pd.read\_csv():*



You can also select columns that you need:



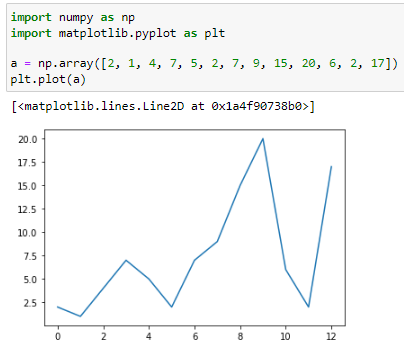
If you want create a Pandas dataframe from the values in your array and then write the data frame to a CSV file:  

# ***Plotting arrays with Matplotlib***

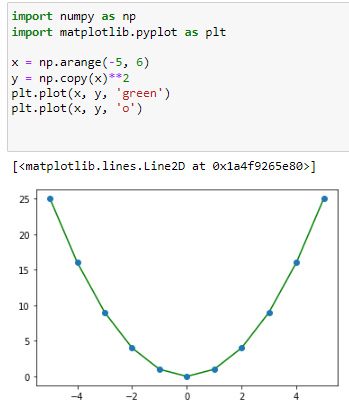
If you need to generate a plot for your values, it`s very simple with Matplotlib.

For example we use this array: 

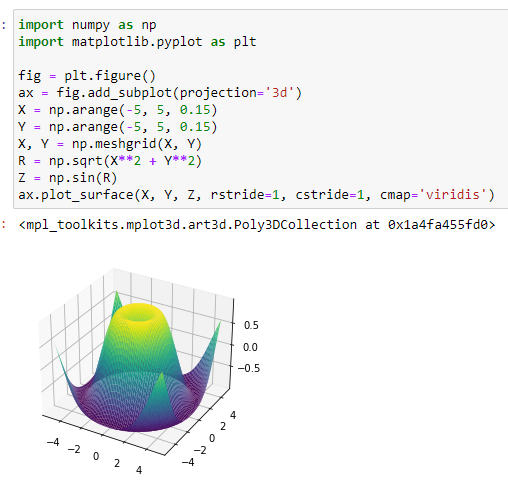
All that you need is one function - *.plot()*. See below:



For example let`s build graph *y=x^2:*



With Matplotlib, you have acces to an enormous number of visualisation options. Really beautiful:



It`s all. Was very interesting. Next step is NumPy [fundamentals](https://numpy.org/devdocs/user/basics.html). Also learn library Matplotlib. Goodbye!