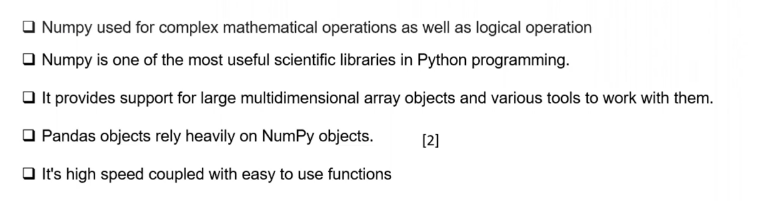
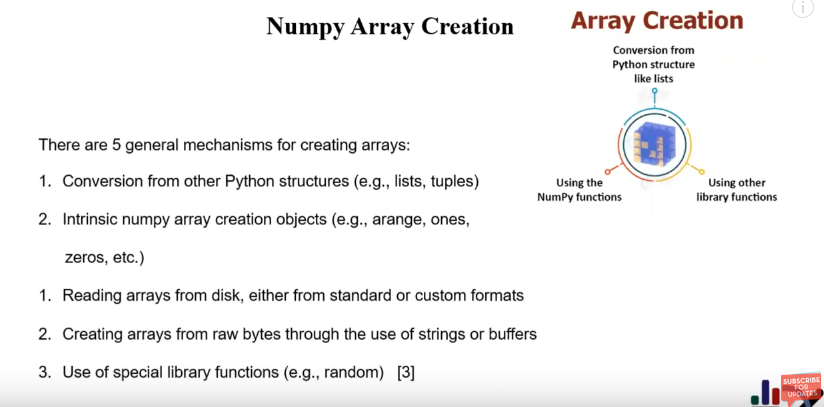
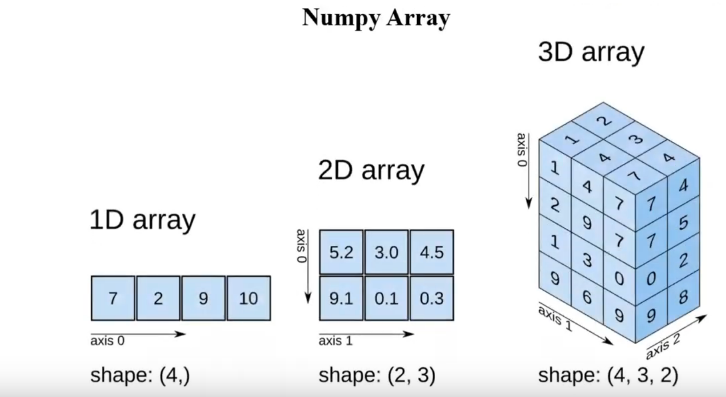
**Introduction to Numpy**

**Numpy:** *Numpy* is basically used to work with or manipulate large multidimensional array and matrix data. A large collection of mathematical functional operations like linear algebra routines, or fourier transforms can be done on these types of data using the *Numpy* library.

Other operations such as array creation or indexing or slicing these arrays/matrices or other numpy operations such as searching or sorting in an array can be done using the *numpy* library.





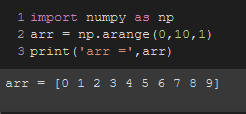


**Creating numpy arrays**

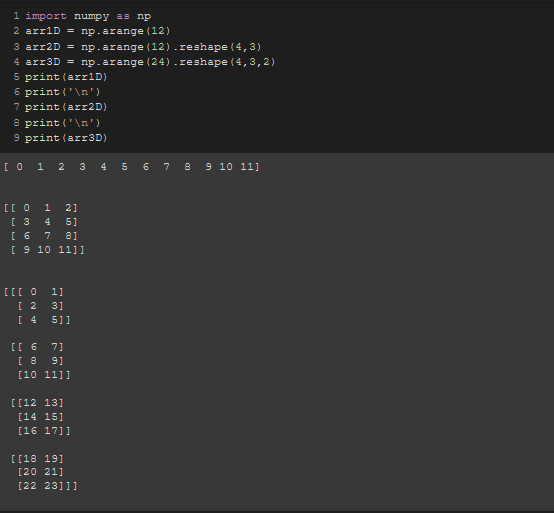
There are various ways to create a numpy function.

1. np.arange(starting\_point, Ending\_point, steps)

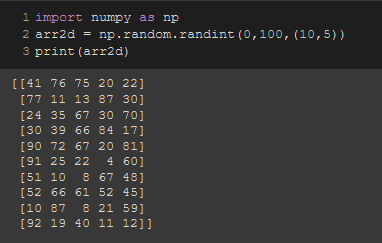
It's the same as the **range** function. Has a starting, ending and step value. Step valu is optional just like in the range function.



To create a multidimensional array,



TO create a multidimentional array randomly,

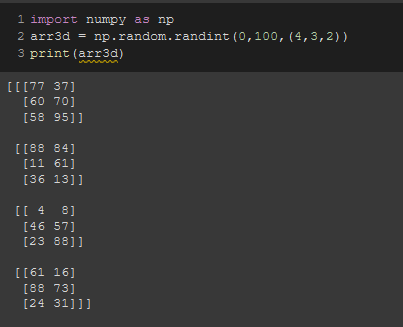


In the 2nd line,

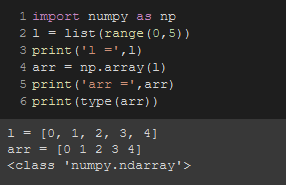
arr2d = np.random.randint(0,100,(10,5))

The last (10,5) part means that, there should be 10 rows and 5 columns

Similarly, to make a 3D array, we add another dimention.



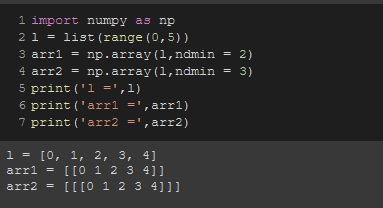
1. np.array()



Note that, Here lies the difference between a list and numpy array. List have commas between the elements, numpy arrays don’t.

This is the way to convert a list into a numpy array. It’s also showing the type of the array.

We can also turn it into multi dimensional. Have to mention the dimension using the parameter **“ndmin =”**



1. np.linspace()

This function is basically used to generate a linear sequence out of the range of numbers provided

The syntax is np.linspace(start value, end value, numbers of sample data, end point = True(if stop included), data type = None(int/float))



Between 0 to 10, it made 5 interval points which have 4 interval spaces (interval point-1)

1. np.logspace()

The numpy logspace() function is used to create an array of equally spaced values between two numbers on the logarithmic scale.

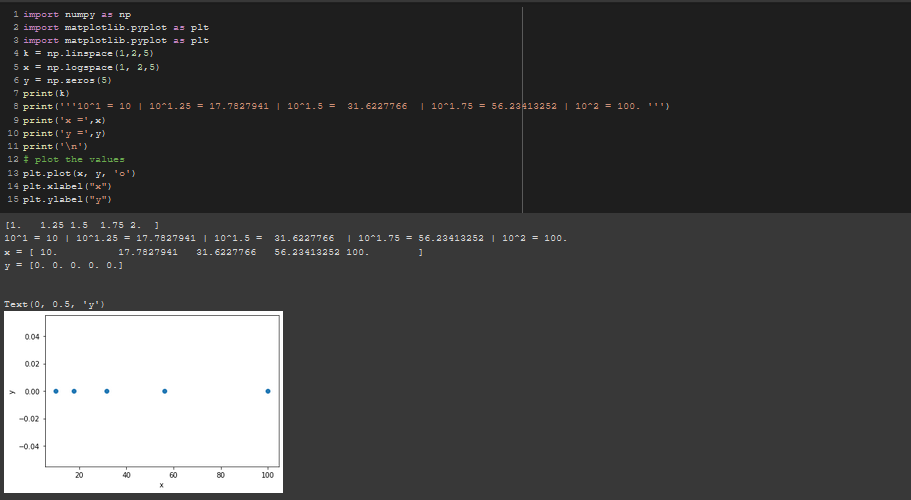
The following is the syntax:

np.logspace with all the default parameters

arr = np.logspace(start, stop, num=50, endpoint=True, base=10.0, dtype=None, axis=0)

mostly you'll be only using these parameters

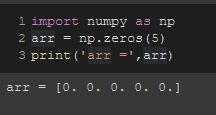
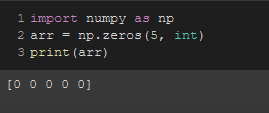
arr = np.linspace(start, stop, num, base)



1. np.zeros()

We can create a null matrix filled with zeros and according to our desired size too.

The syntax is = np.zeros(shape, data type= None, order = 'C')

These are 1D numpy arrays. ***One has float values and another has int.***

We can also create multiple dimensional arrays. Here are 2 examples of 2D & 3D.

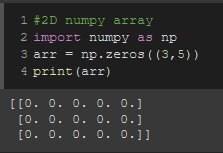


Fig: 2D numpy array

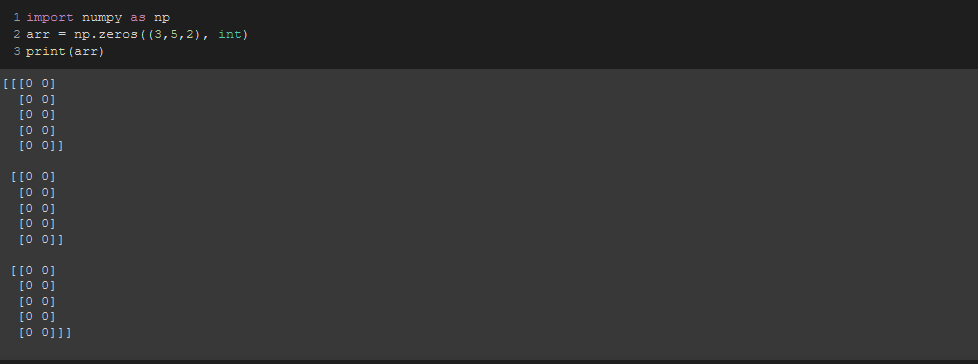
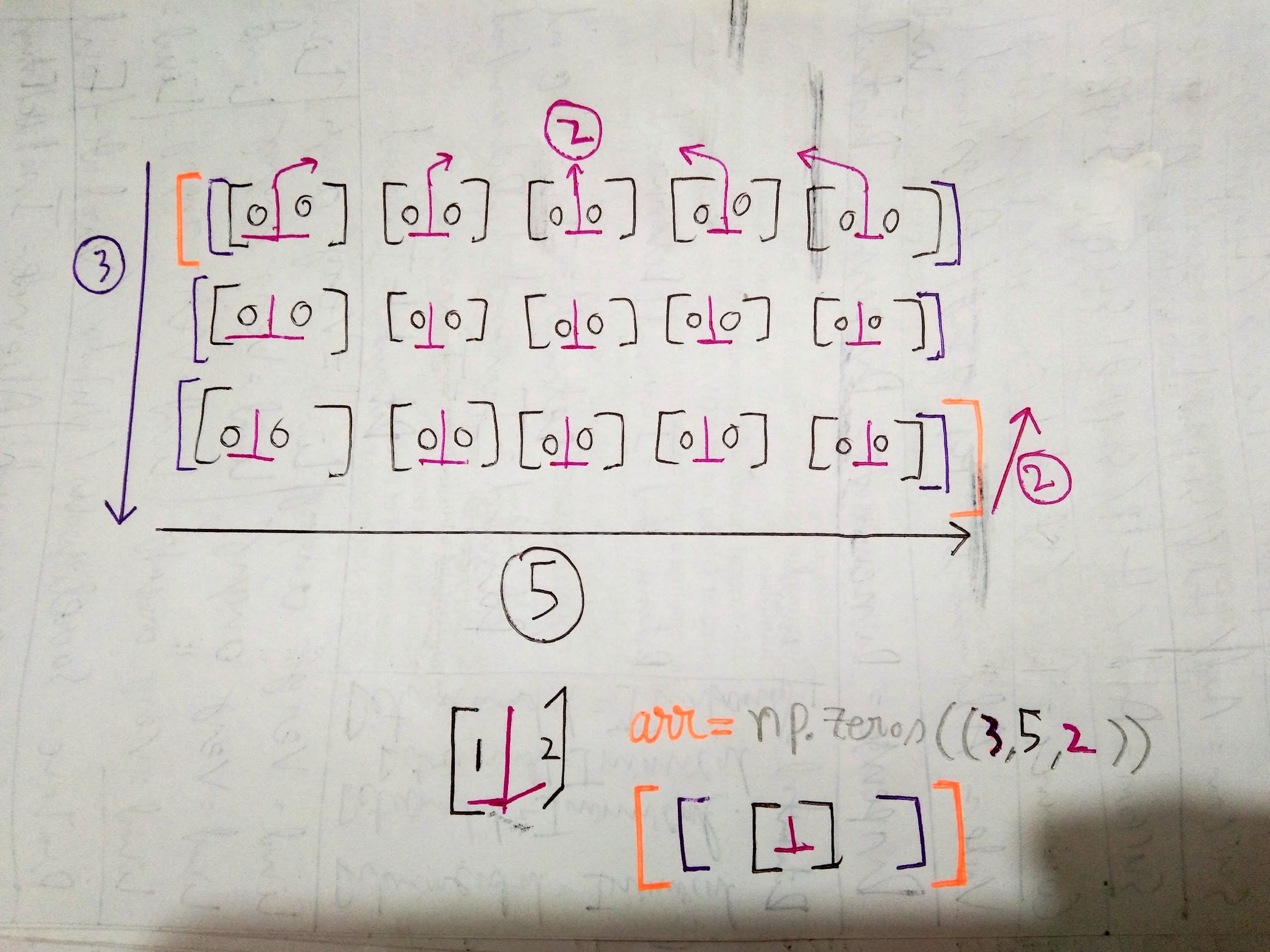
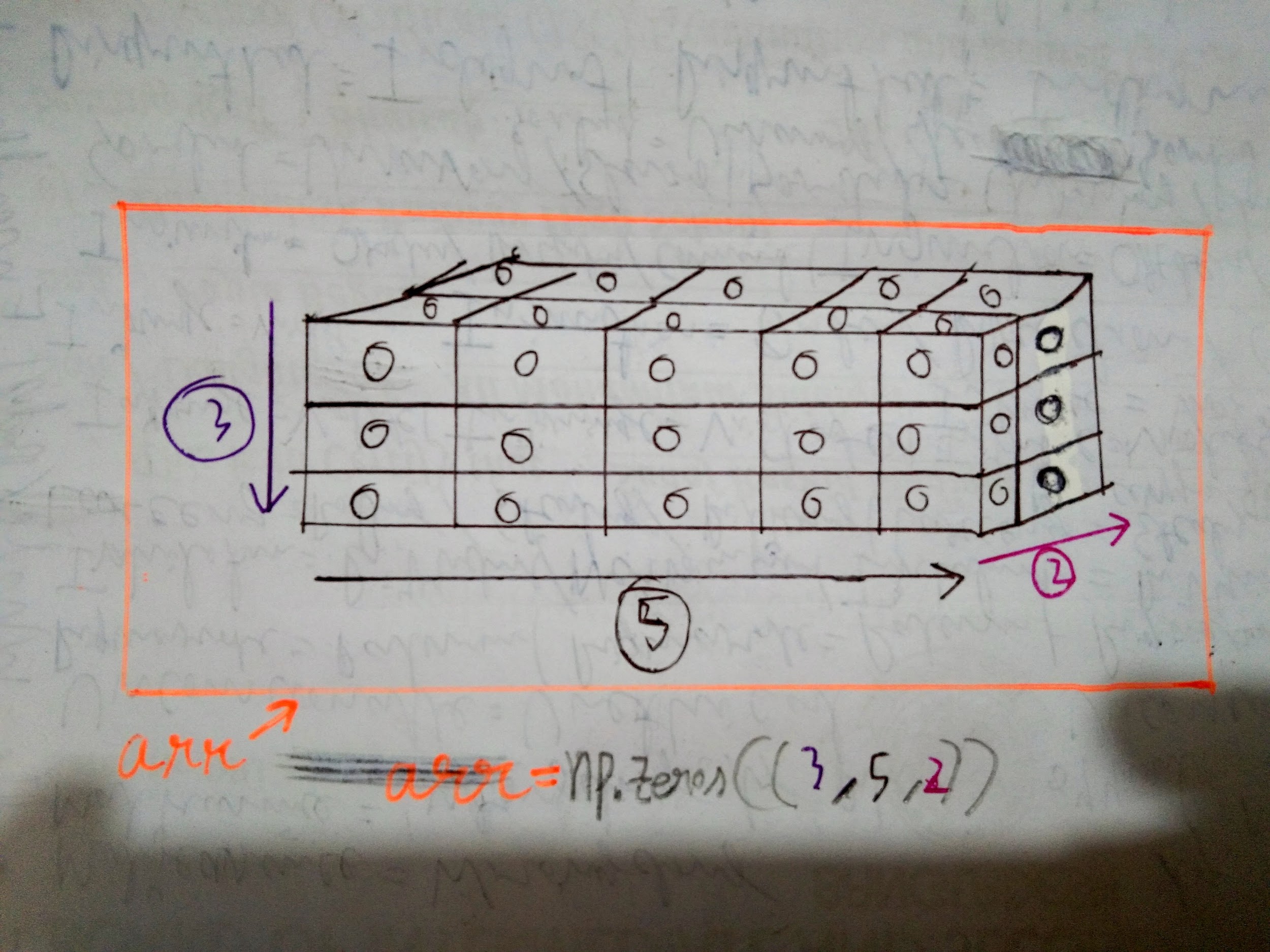


Fig: 3D numpy array

The dimensions are working like this



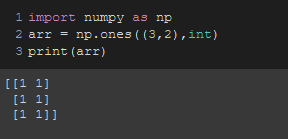
You can imagine there’s another flat block behind the block we’re seeing. Something like the picture below.



1. np.ones()

Similarly, We can create a matrix filled with only ones and according to our desired size too.

The syntax is = np.ones(shape, data type= None, order = 'C')

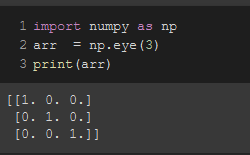


1. np.eye()

We can create identity matrix with np.eye()

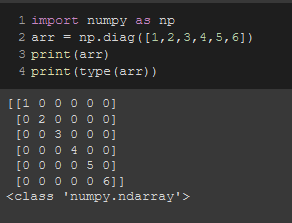
Syntax: np.eye(size)

Note: Identity matrix is always a square matrix



1. np.diag()

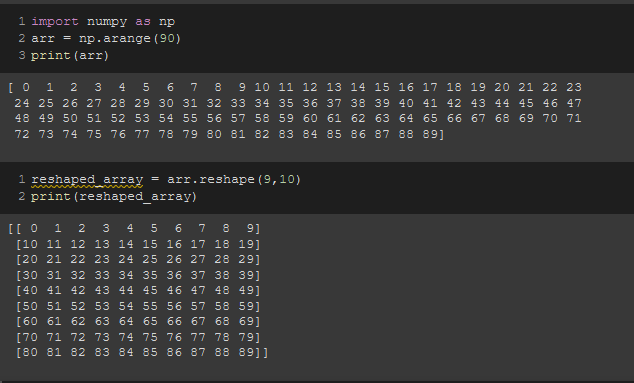
numpy. diag(v, k=0)[source] Extract a diagonal or construct a diagonal array.



1. .reshape()

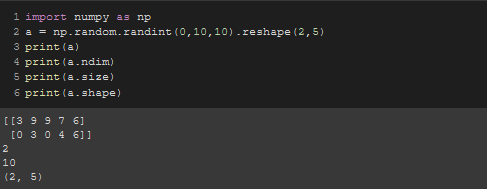
This function is used to reshape an array according to our desired shapes and sizes.

The syntax is ***array.reshape(row,column)***

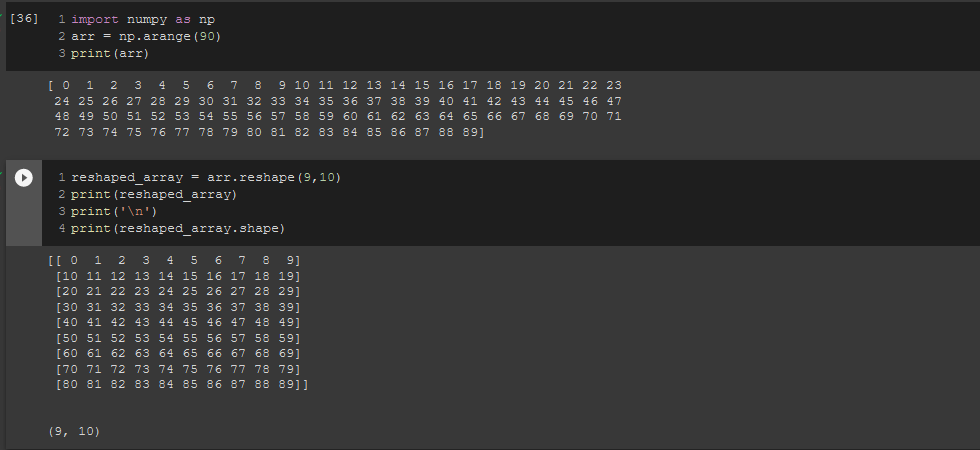


1. .ndim, .size, .shape

This is to see the dimension,size & shape of an array, We use the following syntax

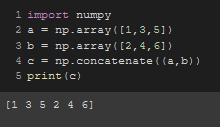


Same for multidimensional arrays.

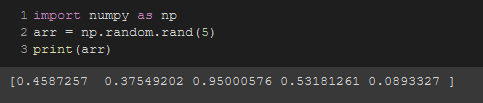


1. .concatenate()

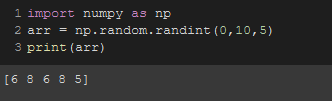
To concatenate 2 arrays, We use the concatenate() function



**Random generation in numpy**

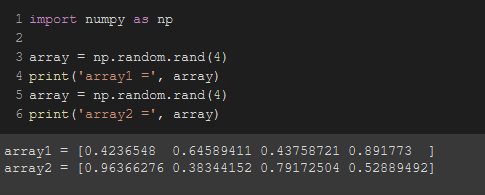


To specify a start and an end,   
**syntax:** arr = np.random.randint(start, end, array length)

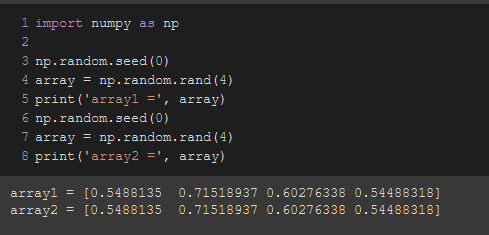


**Making random number numbers predictable:**

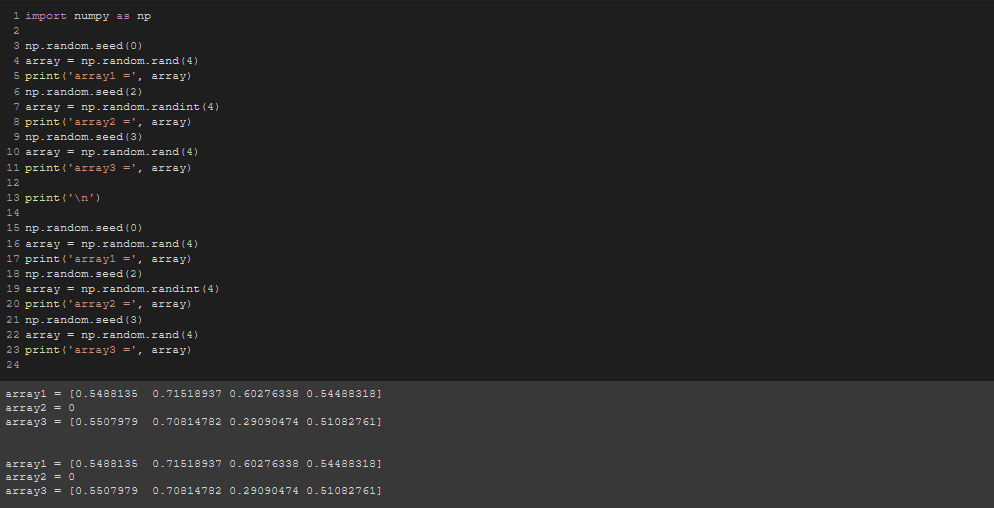
If we use the numpy random method, we will always get different sets of numbers



But with the ***.seed()*** method, We can get the same set of numbers every time



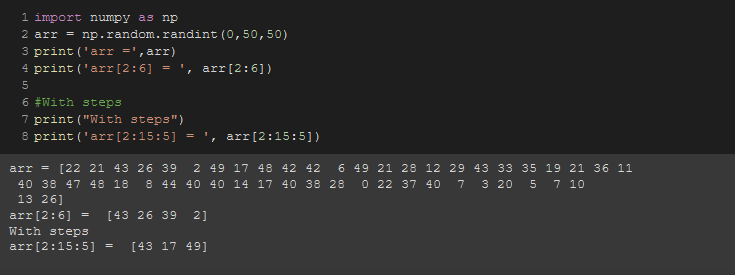
The input we give inside the ***.seed()*** method, the number/number it will repeat according to the previously saved sets of numbers indexed in the ***.seed()*** method.



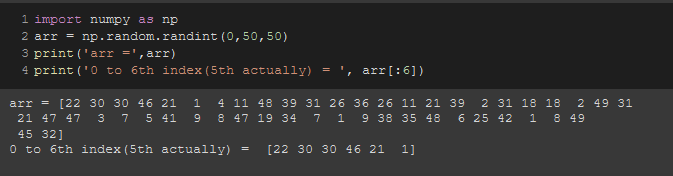
**Slicing an array in numpy**

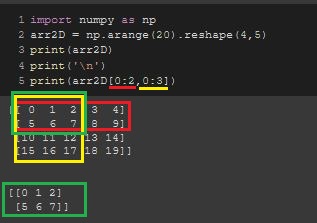
Slicing a numpy array is also the same as slicing a string. We pass the indexing in the following 2 ways.

1. [ start **:** end ]
2. [ start **:** end **:** step ]



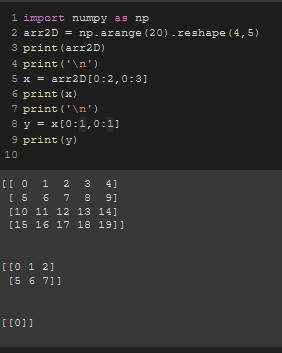
All the slicing operations are the same as slicing strings.



In a multidimensional array, array slicing is done by taking rows and columns common.  


This is for a 2D array.

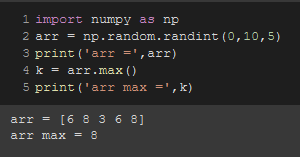
Similarly can be done for a 3D array.



**Searching in numpy array**

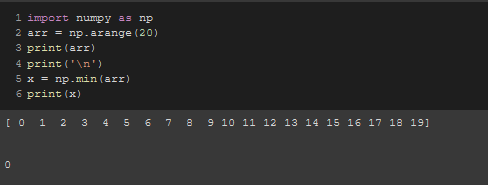
* **Finding maximum value in an array**

By using the .max() function, we can easily find the maximum value in an array.



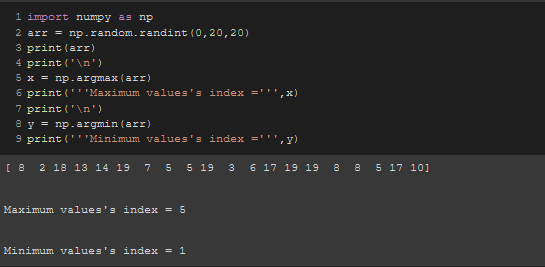
* **Finding minimum value in an array**

By using the .min() function, we can easily find the maximum value in an array.



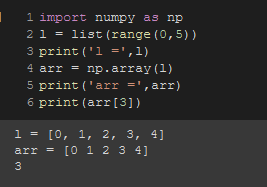
* **Finding the maximum & minimum value’s index numbers in an array**

By using the .argmax() & .argmin() function, we can easily find the maximum and minimum value’s index numbers within an array.

**  
Note:** It only gives the index of the first largest/smallest value it encounters & ignores the rest.

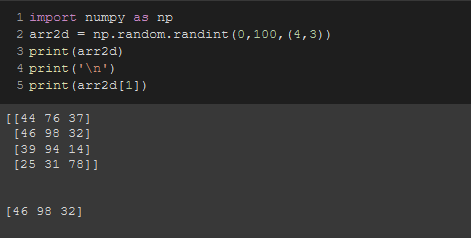
**Indexing in numpy**

Basically, The indexing is the same as a list. Starts from 0.

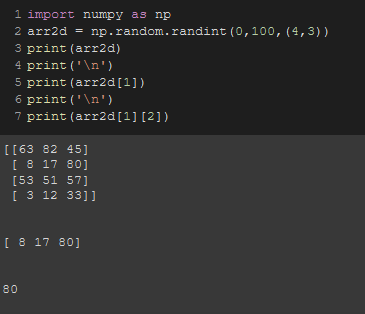


This is for a 1D array.

To look for an array element index wise in a 2D array, We use the following syntax,

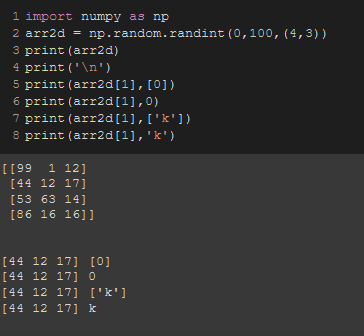


To see the elements from within that array element, we sue the following syntax.



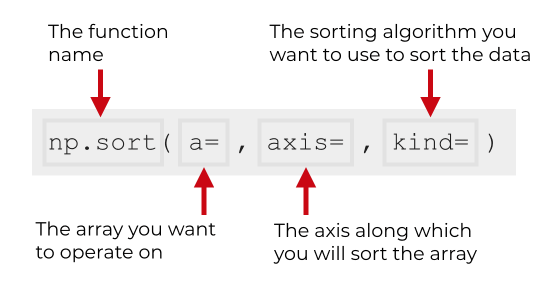
Basically, It’s like searching for an element within a nested list.

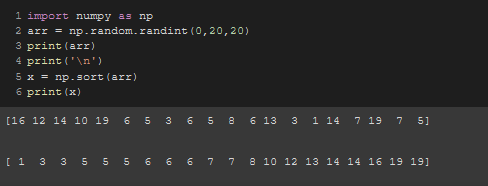
If we want to see something extra with that, we use the following syntax.



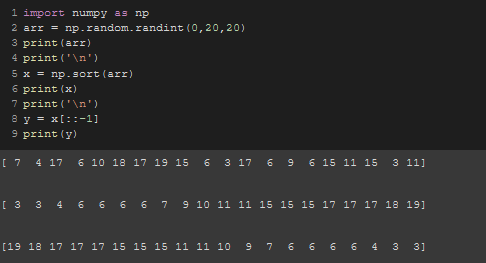
**Sorting in numpy**

To sort an array, The index is

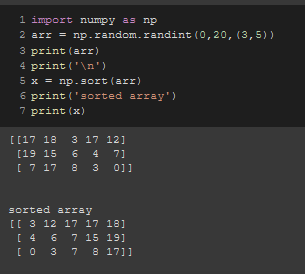




To do negative sorting, We use the following syntax.



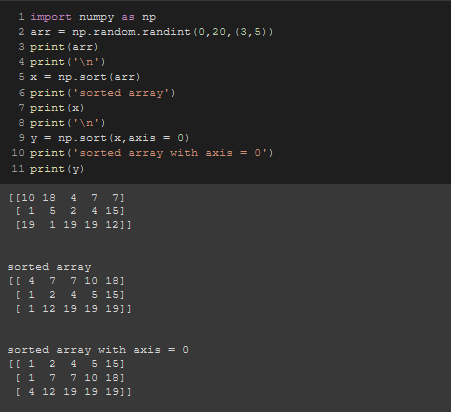
To sort multidimensional array,

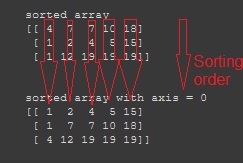


**Axis**

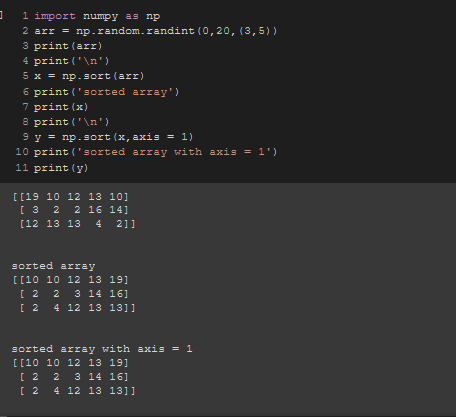
Axes are defined for arrays with more than one dimension. A 2-dimensional array has two corresponding axes: **the first running vertically downwards across rows (axis 0), and the second running horizontally across columns (axis 1).**

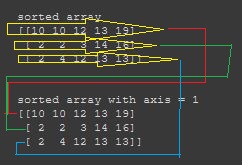
* When axis = 0



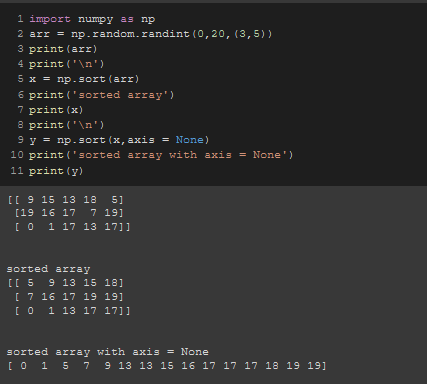


* When axis = 1





* When axis = None

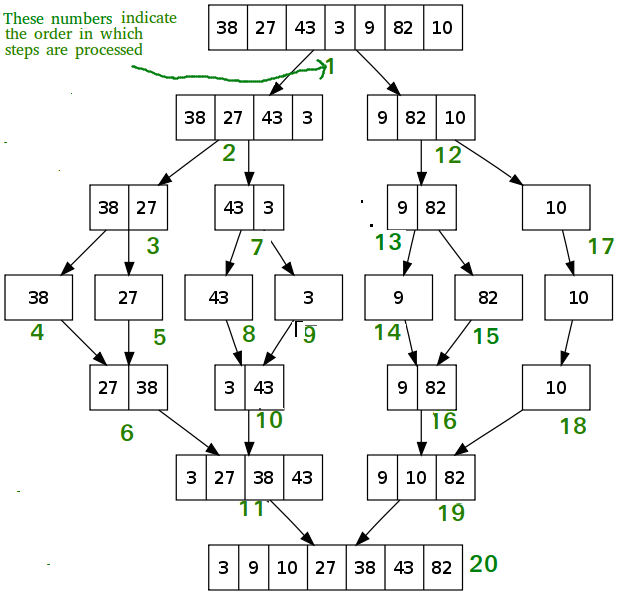


**Kinds**

Different kinds of sortings are,

* mergesort()

It divides the input array into two halves, calls itself for the two halves, and then merges the two sorted halves. The merge() function is used for merging two halves.



* quicksort()

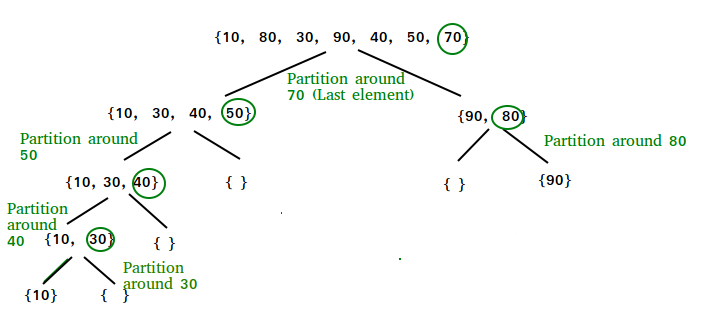
Like Merge Sort, QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.

Always pick first element as pivot.

Always pick last element as pivot (implemented below)

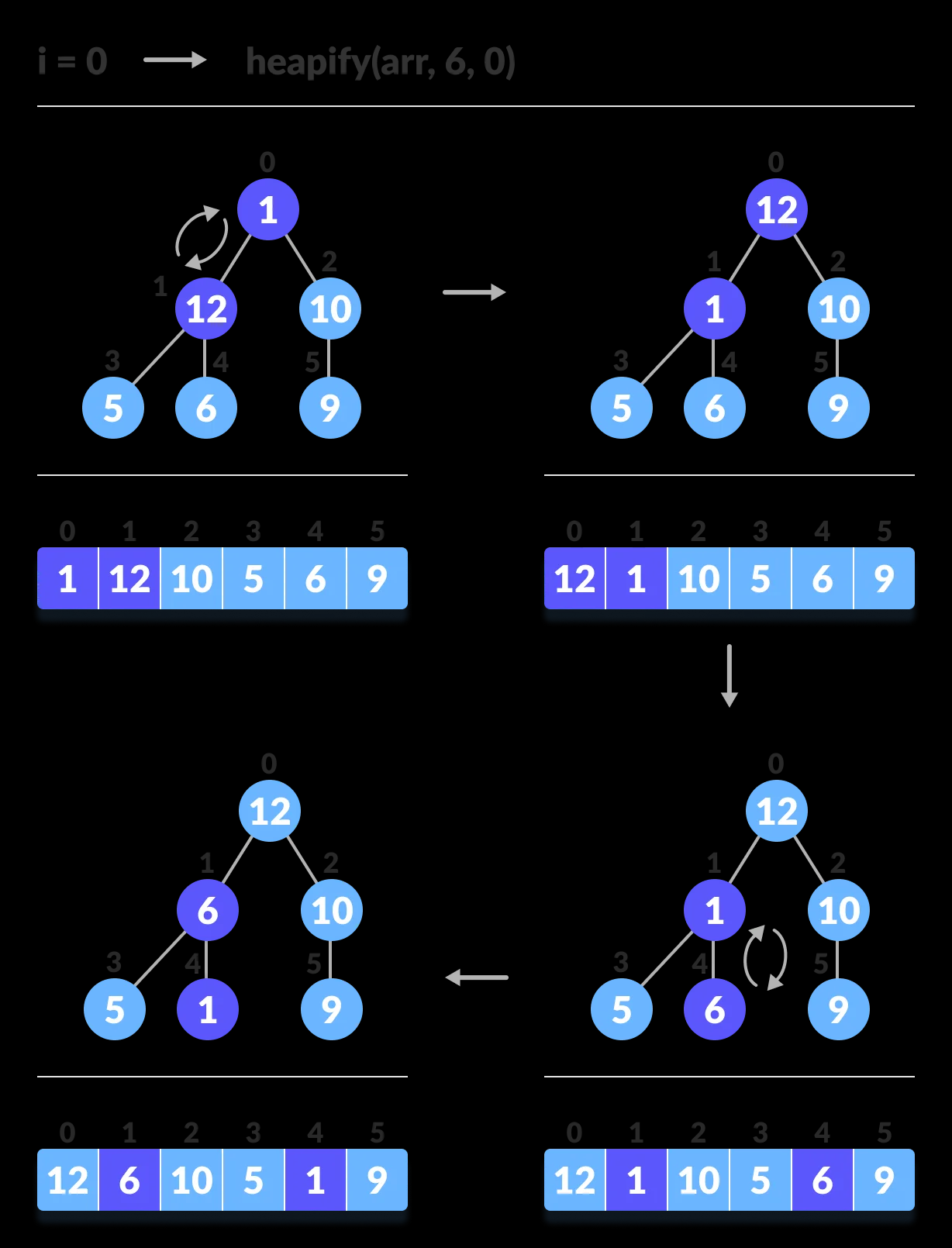
Pick a random element as pivot.

Pick median as pivot.



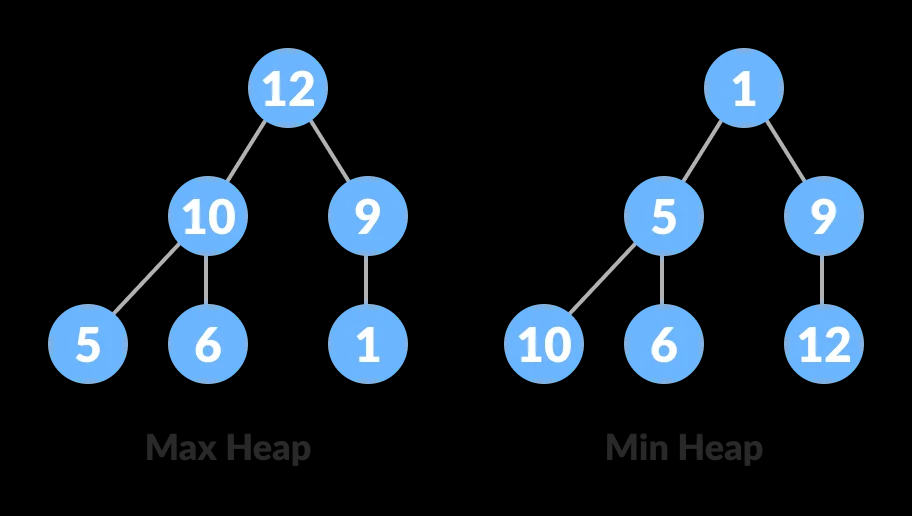
* heapsort()

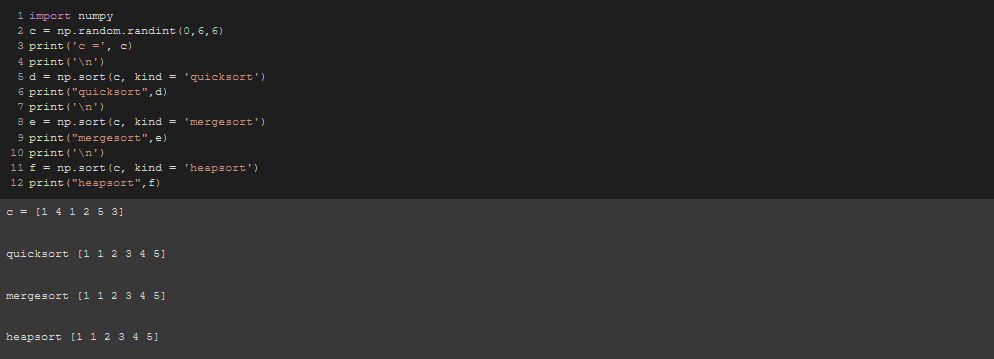
Heap sort is a comparison-based sorting technique based on Binary Heap data structure. It is similar to selection sort where we first find the minimum element and place the minimum element at the beginning. We repeat the same process for the remaining elements.



There are 2 types of heap. Those are,

* Max heap
* Min heap

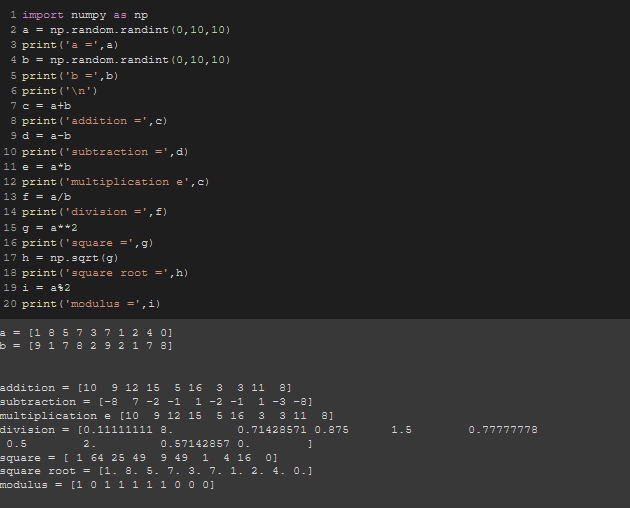




**Mathematical operations in numpy**

Multiple types of arithmetic operations can be done on numpy arrays.

**Note:** Array lengths has to be the same or array sizes has to be the same.



Same goes for 2D arrays

