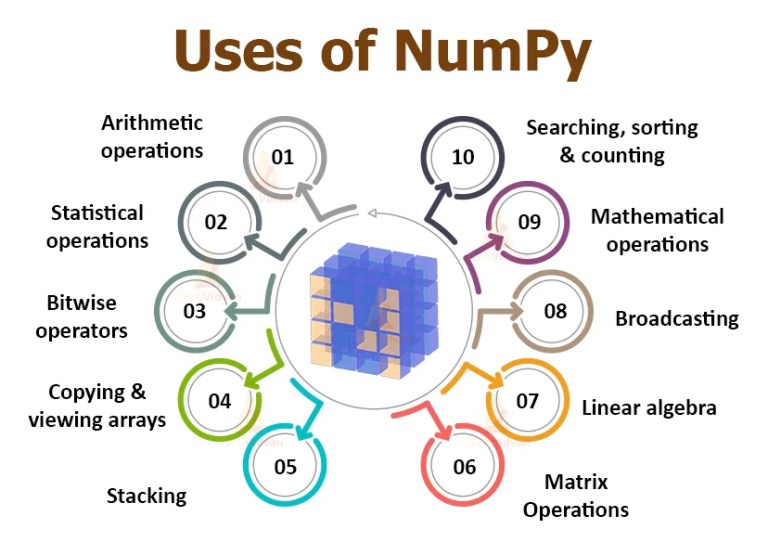
**2. NumPy (NUMERICAL PYTHON)**

***“The goal is to turn data into information, and information into insight.”***

***-Carly Fiorina***

**Introduction**

NumPy stands for **‘Numerical Python’**. It is a package for data analysis and scientific computing with Python. NumPy uses a multidimensional array object, and has functions and tools for working with these arrays. The powerful n-dimensional array in NumPy speeds-up data processing. NumPy can be easily interfaced with other Python packages and provides tools for integrating with other programming languages like C, C++ etc.

**Installing NumPy**

NumPy can be installed by typing following command: **pip install NumPy**



As we are using the jupyter notebook which was launched using the Anaconda, in which we have some of the python libraries already installed. Just we need to make sure that do we have the required library or not using the command **import numpy**,if we won’t get any error then it means we have that particular library installed and good to perform the function with respect to NumPy.



**To Find the NumPy version we are using**

**Array**

An array is a data type used to store multiple values using a single identifier (variable name). An array contains an ordered collection of data elements where each element is of the same type and can be referenced by its index (position).

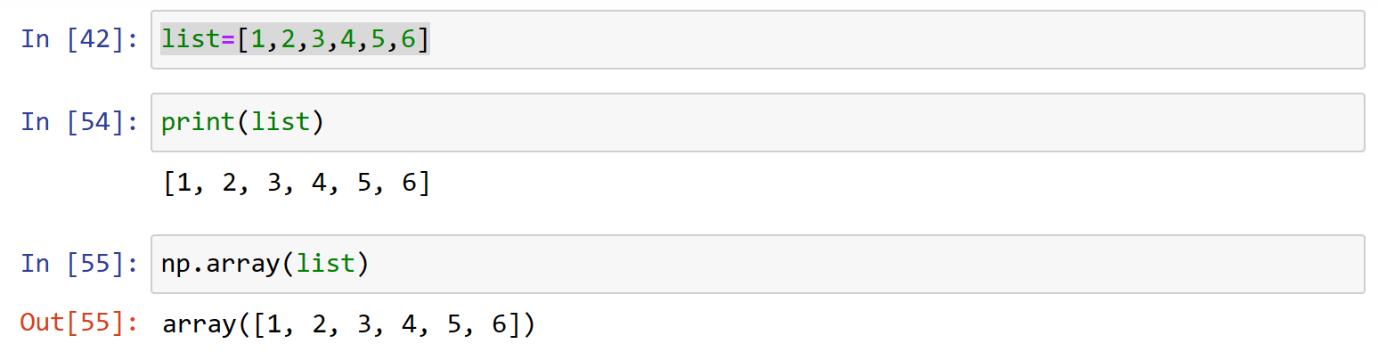
The important characteristics of an array are:

* Each element of ***the array is of same data type***, though the values stored in them may be different.
* The entire array is ***stored contiguously*** in memory. This makes operations on array fast.

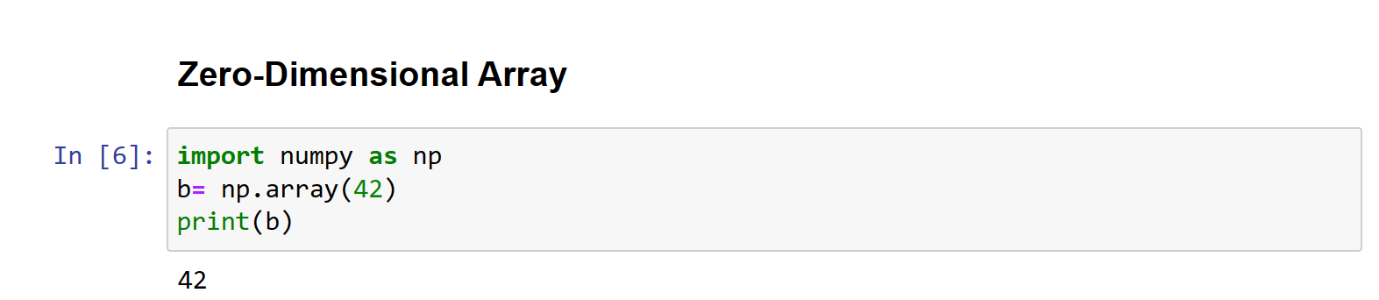
**NumPy Array**

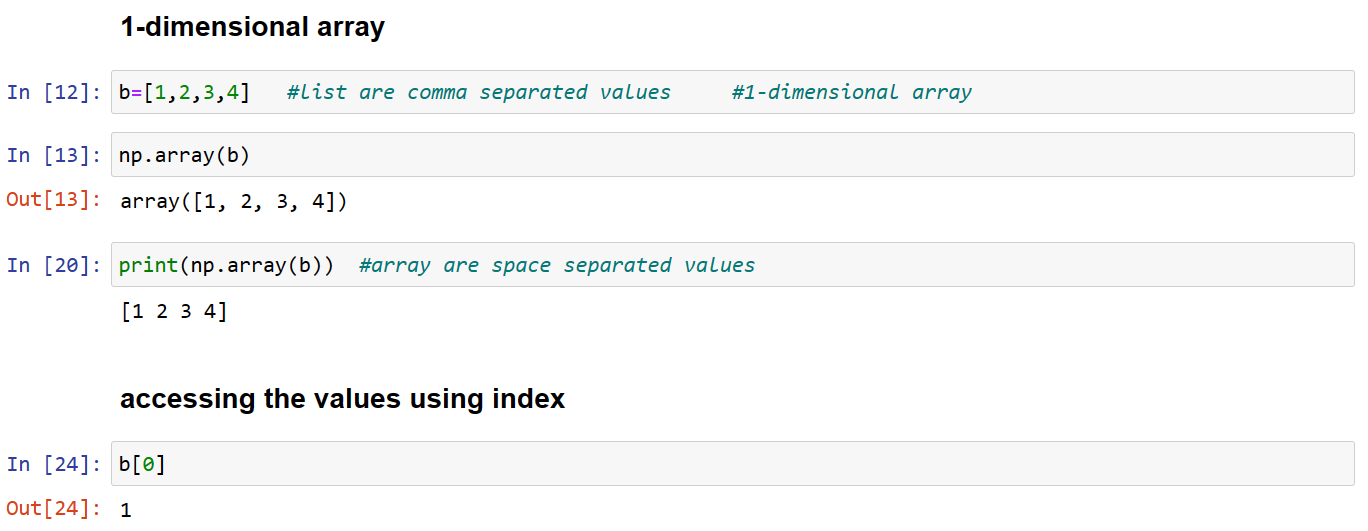
NumPy arrays are used to store lists of numerical data, vectors and matrices. The NumPy library has a large set of routines (built-in functions) for creating, manipulating, and transforming NumPy arrays. Python language also has an array. data structure, but it is not as versatile, efficient and useful as the NumPy array. The NumPy array is officially called ndarray but commonly known as array.

**Creation of NumPy Arrays from List**

The NumPy’s array() function converts a given list into an array.

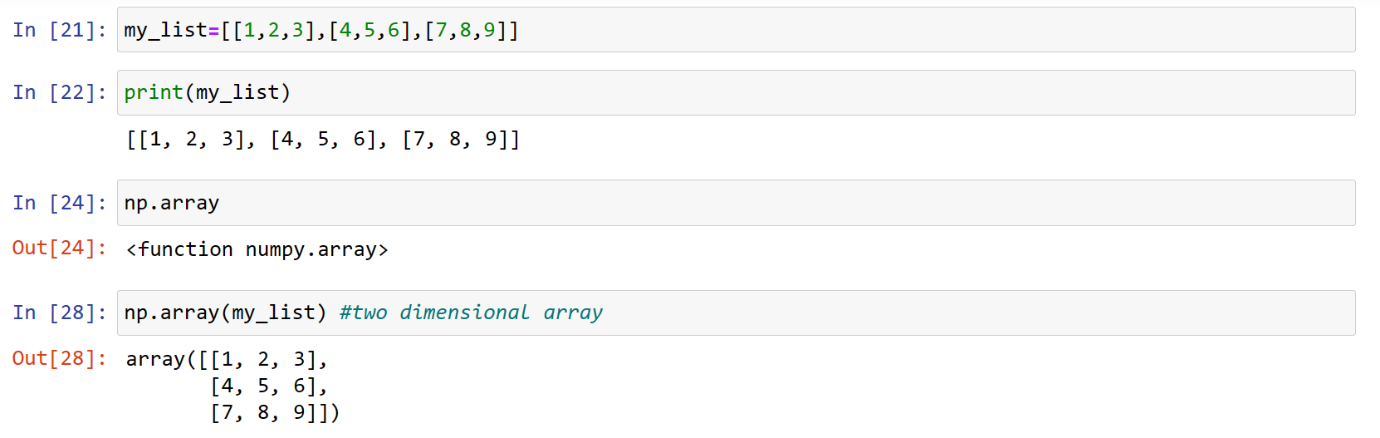
**Zero-Dimensional Array:** 0-D arrays, or Scalars, are the elements in an array. Each value in an array is a 0-D array.

**One Dimensional Array []:** An array with only single row of elements is called 1-D array

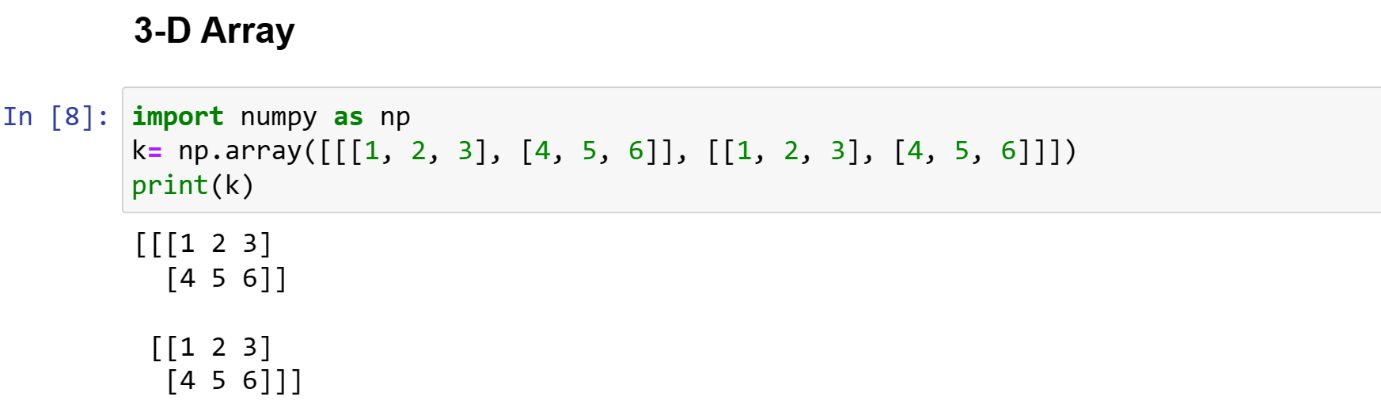


Observe that since there is a string value in the list, all integer and float values have been promoted to string, while converting the list to array.

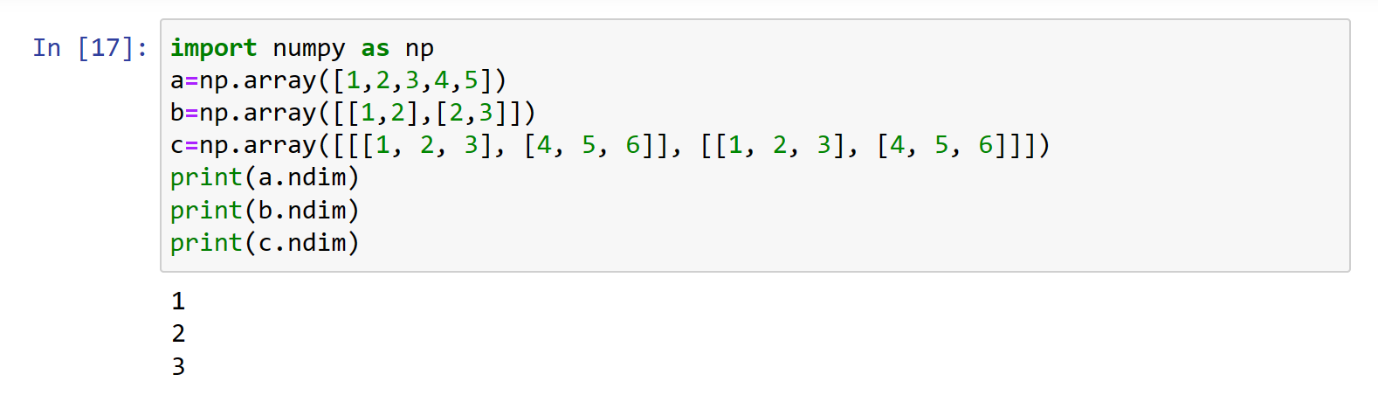
***Note:*** U32 means Unicode-32 data type and [] indicates 1-D array

**Two-Dimensional Array [[]]:** We can create a two-dimensional (2-D) arrays by passing nested lists to the array() function.

***Note:*** [[]] indicates the 2-D array

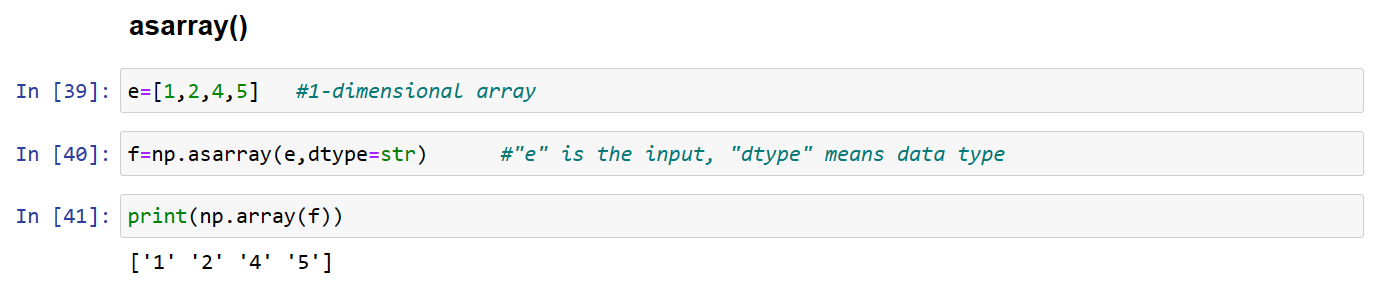
**Three-Dimensional Array [[[]]]:** An array that has 2-D arrays (matrices) as its elements is called 3-D array.

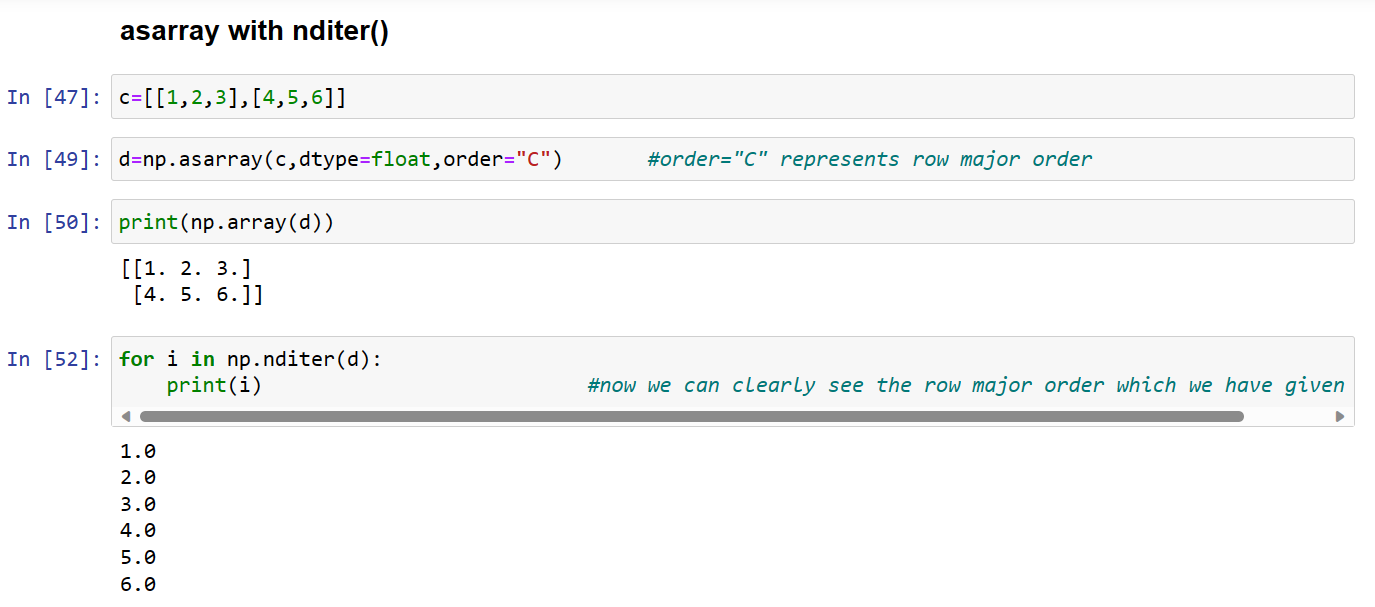
**To Check Number of Dimensions:**

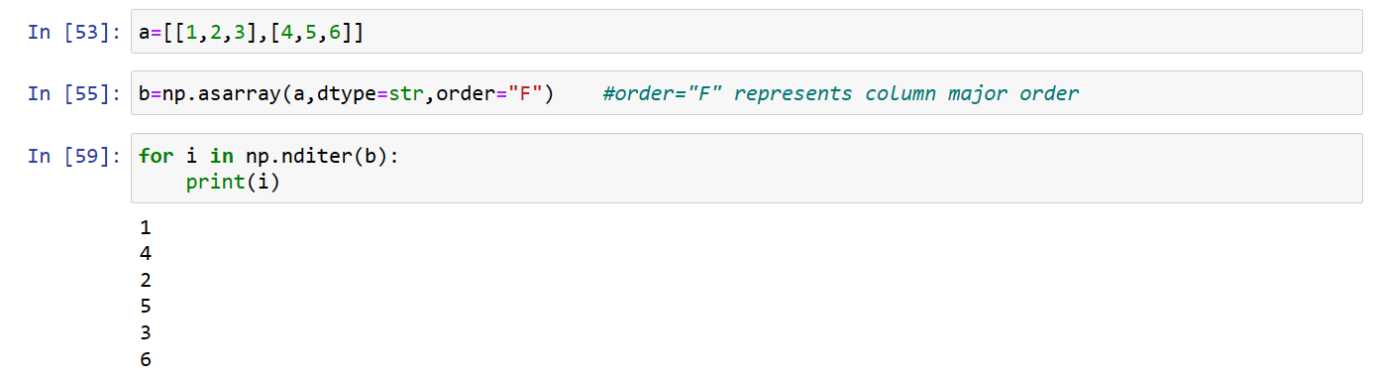
NumPy Arrays provides the ndim attribute that returns an integer that tells us how many dimensions the array has.

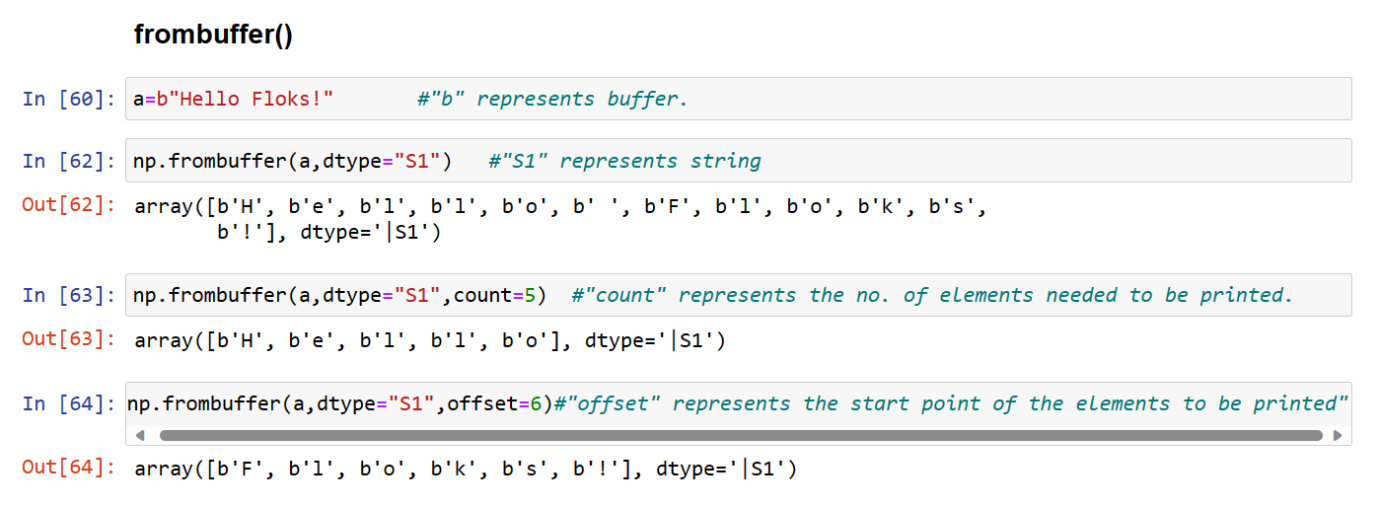
**Other types for creating an array:**

**asarray():** In NumPy, the numpy.asarray() function is used to convert a given input to an array. It is similar to the numpy.array() function, but it has some differences in behaviour.



**asarray() with nditer:** The numpy.ndarray object has an nditer attribute, which is an iterator object that can be used to iterate over the elements of an array. The numpy.asarray() function, when used with the nditer attribute, allows you to convert the input data to an array and simultaneously iterate over it. This can be useful for performing operations on each element of the array while converting the data type if needed.

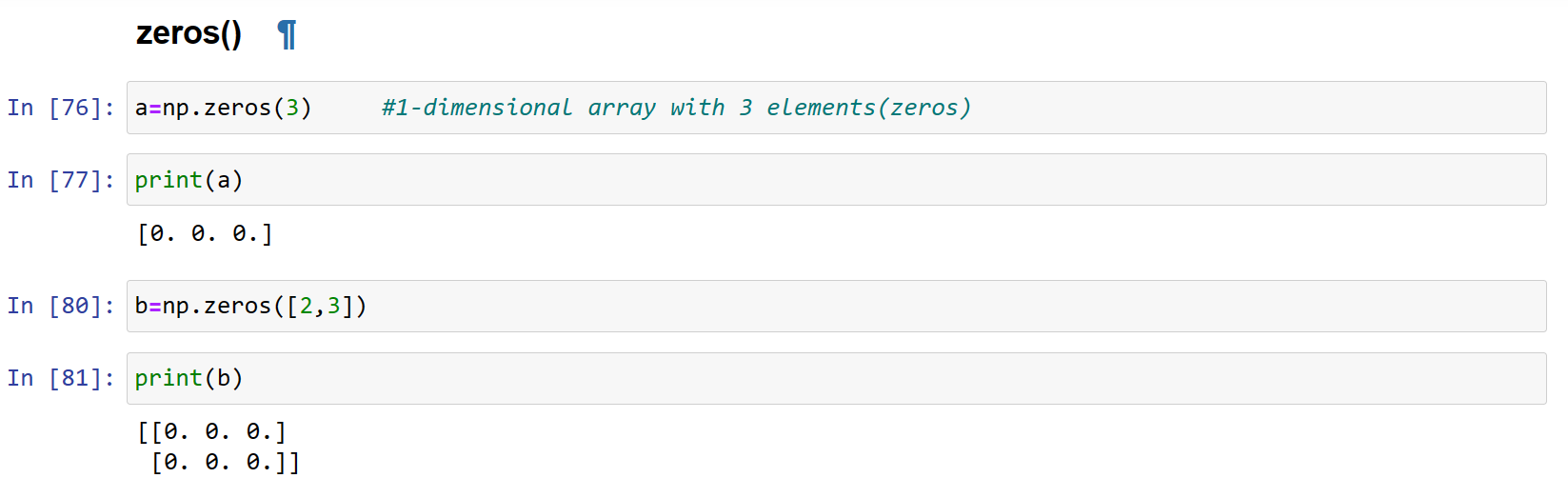


**frombuffer():** The numpy.frombuffer() function is used to interpret a buffer as a 1-dimensional array. It creates an array with data in the buffer without copying it. This function is particularly useful when working with binary data or when you have data stored in a buffer-like object, such as bytes.

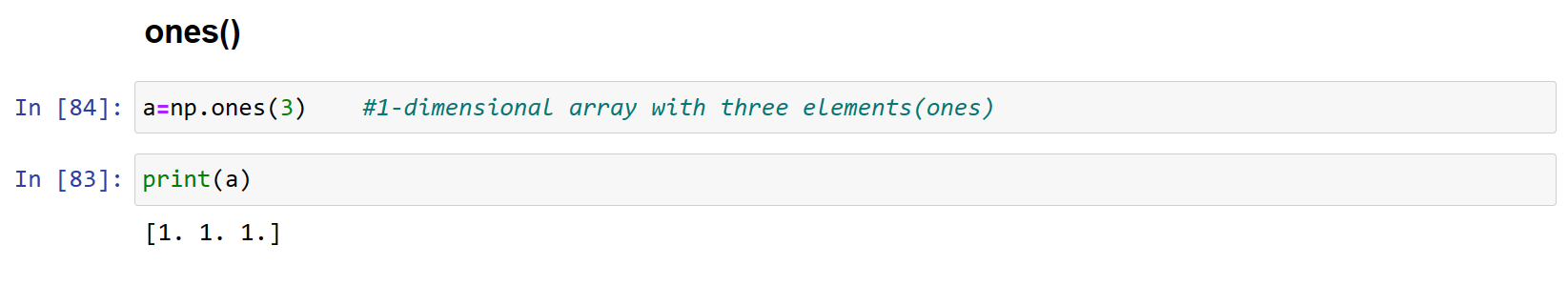
**fromiter():** The numpy.fromiter() function is used to create a 1-dimensional array from an iterable object, such as a Python iterator or generator. It allows you to efficiently create a NumPy array by reading elements from the iterable.

**Initializing Array/Assigning values to an array:**

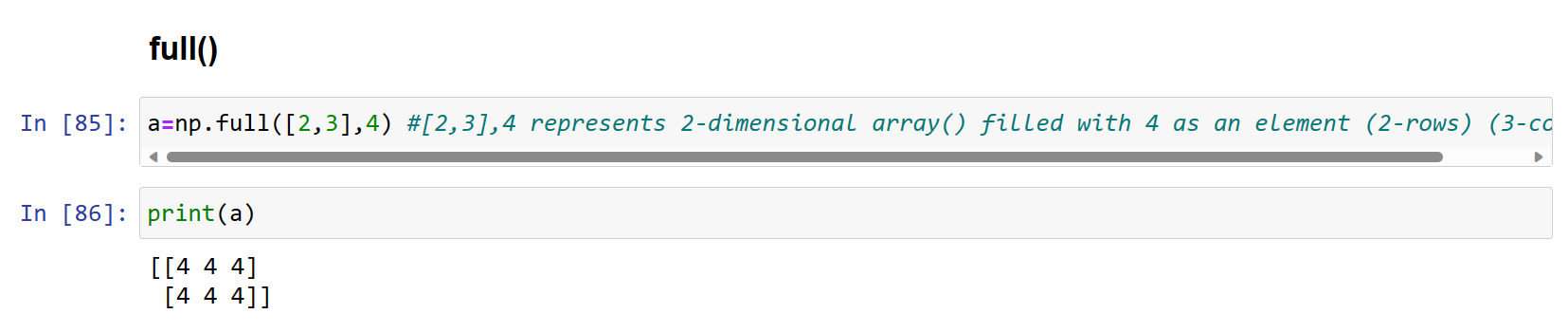
**zeros()**

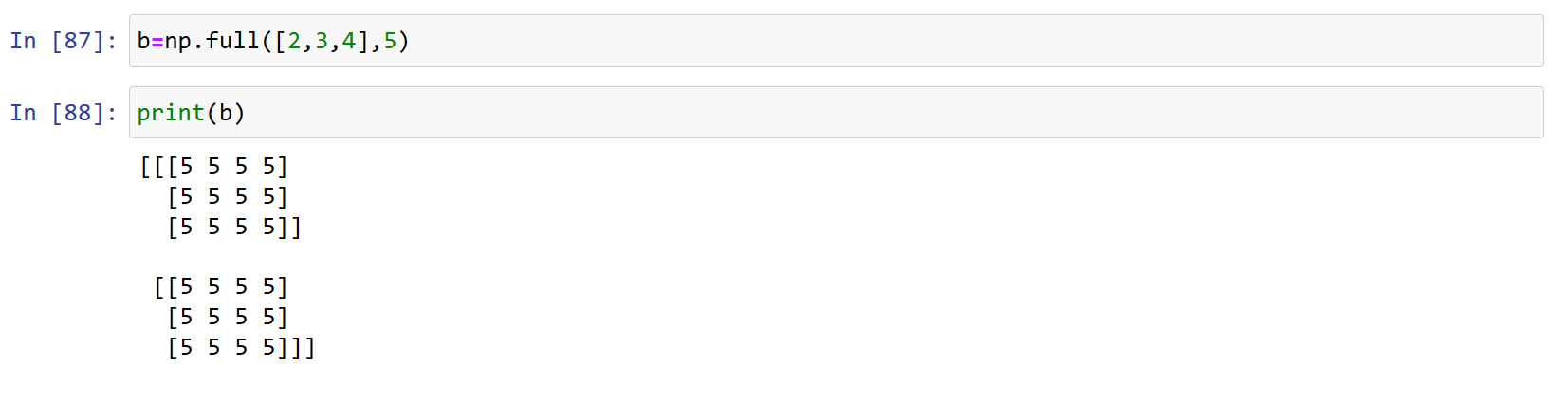


**ones()**

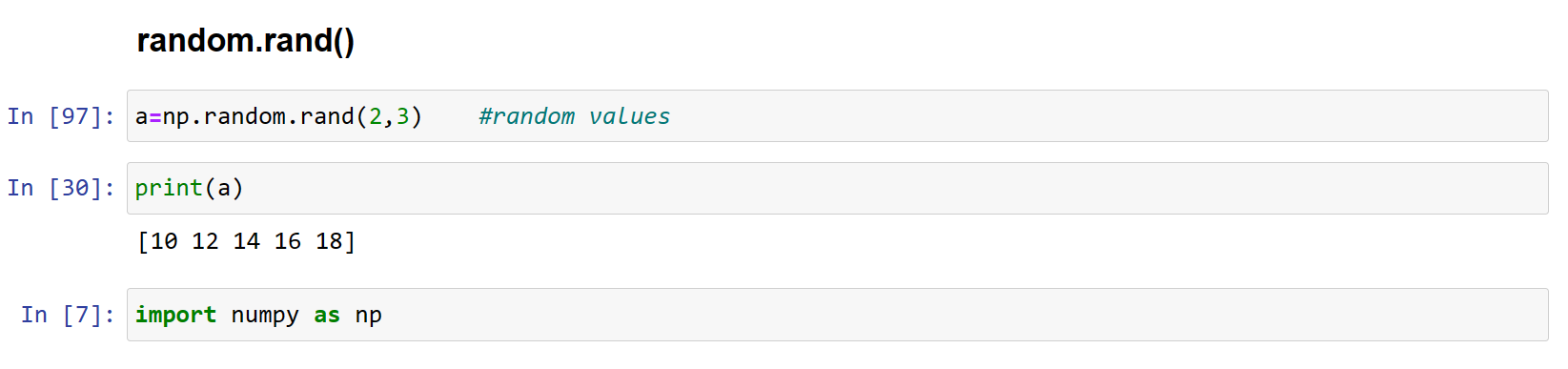


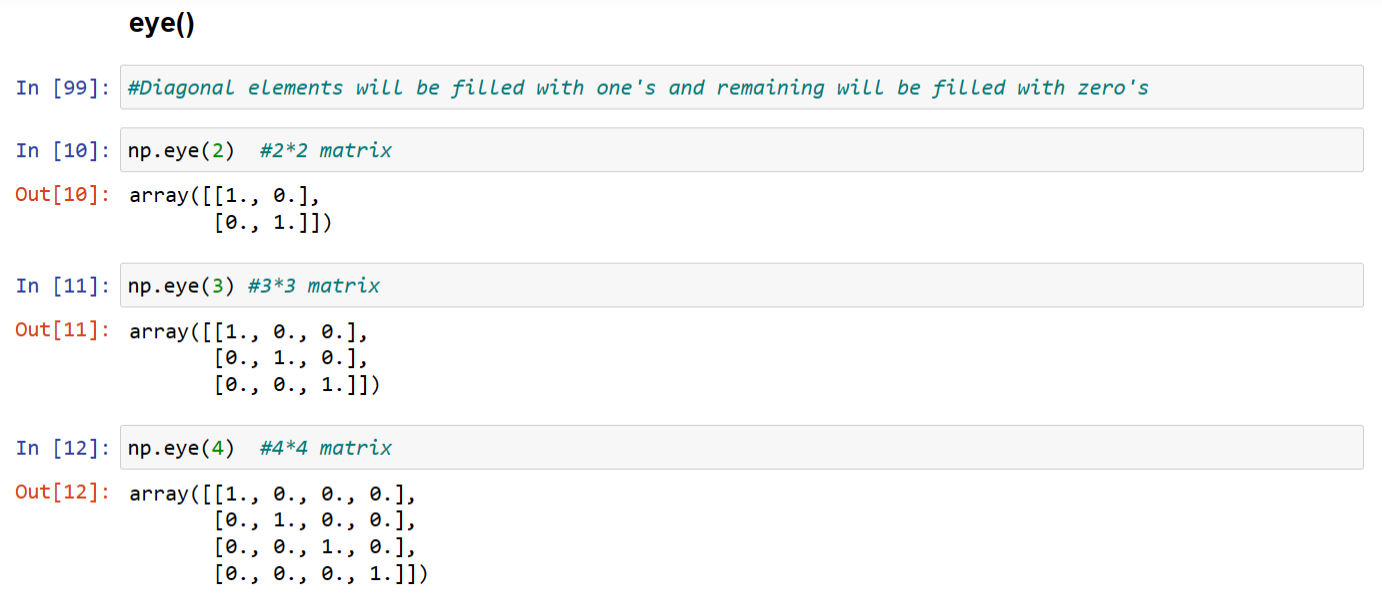
**full()**





**random.rand()**



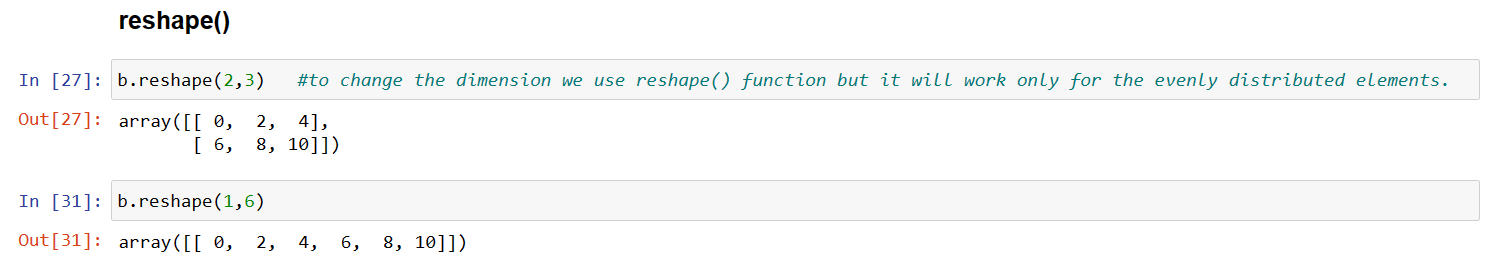
**eye()**

**Numerical Range:**

**arange()** We can create an array with numbers in a given range and sequence using the arange() function. This function is analogous to the range() function of Python.

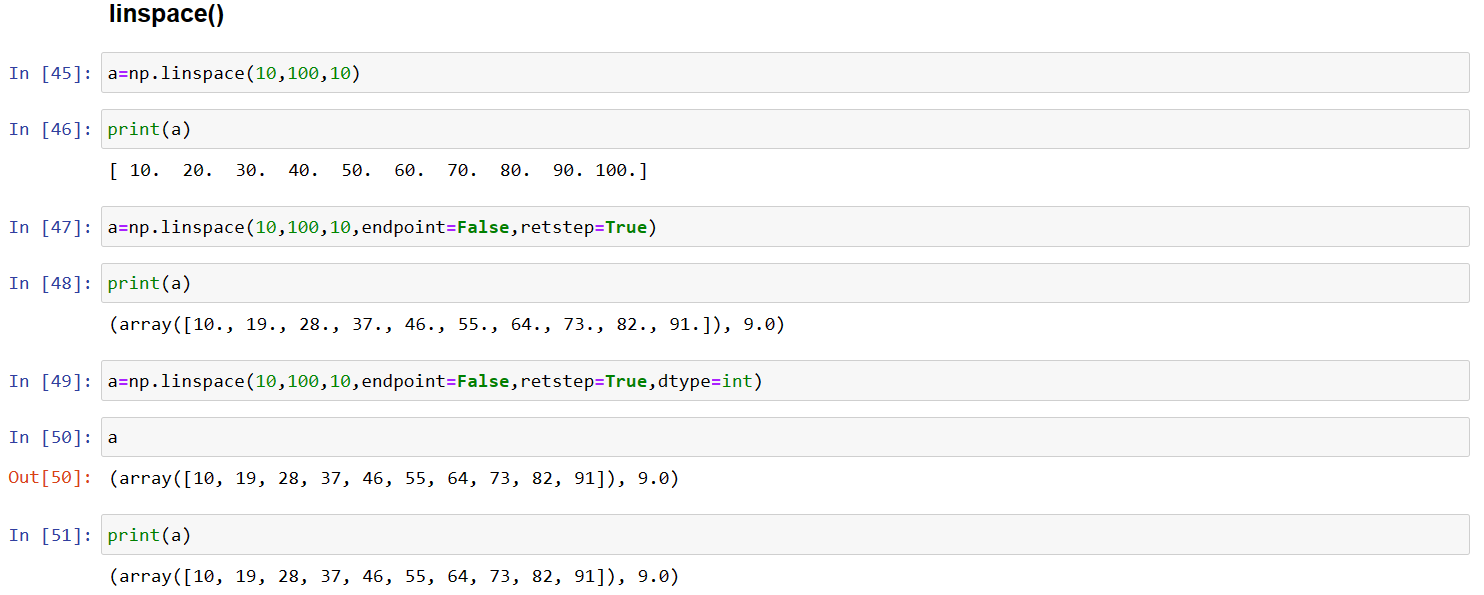
**reshape()**

We can modify the shape of an array using the reshape() function. Reshaping an array cannot be used to change the total number of elements in the array.



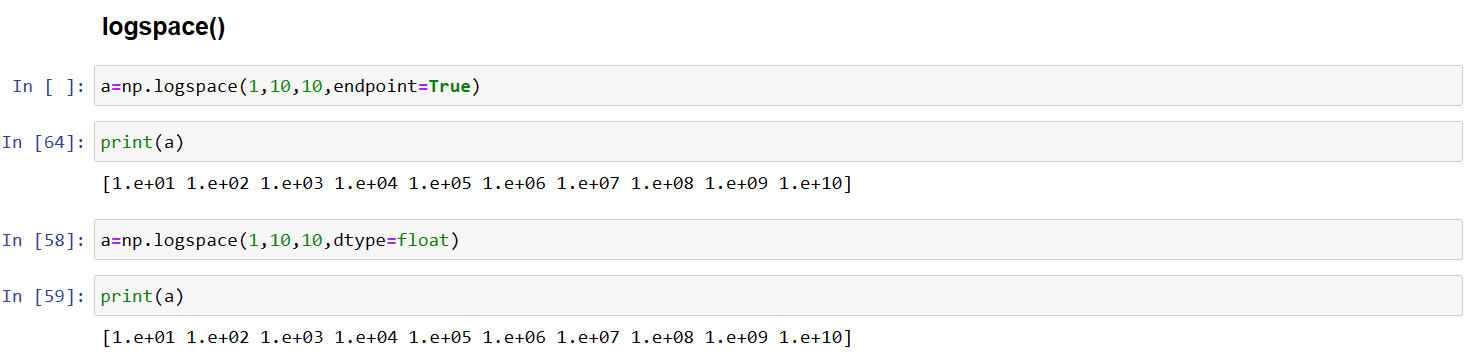
**linspace()**

In NumPy, the linspace() function is used to create an array of evenly spaced values over a specified range.



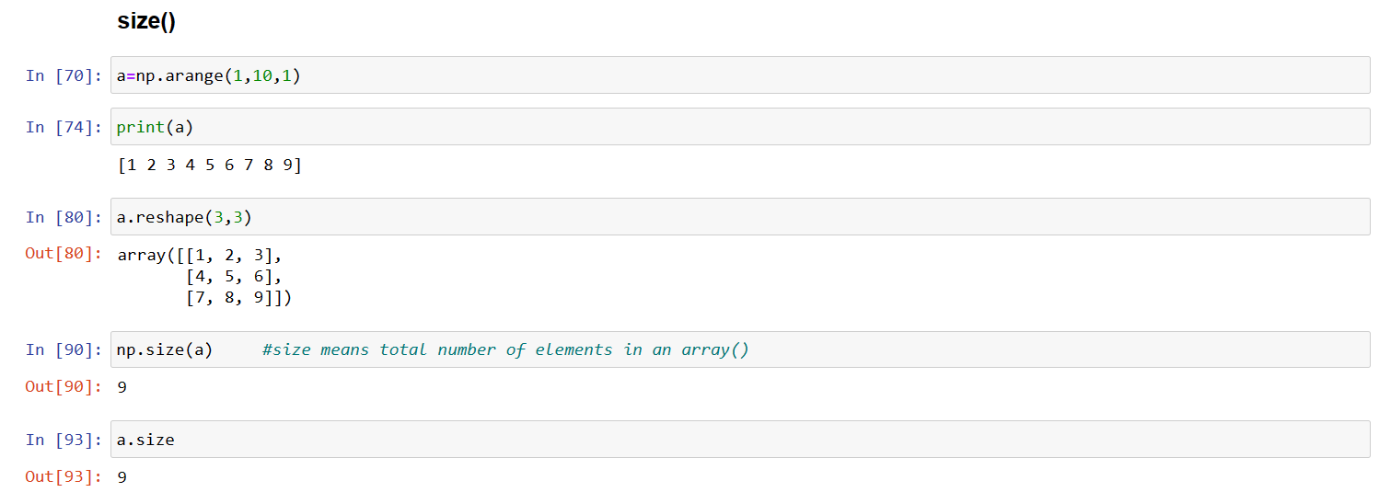
**logspace()**

In NumPy, the logspace() function is similar to linspace(), but it generates an array of numbers that are evenly spaced on a logarithmic scale.



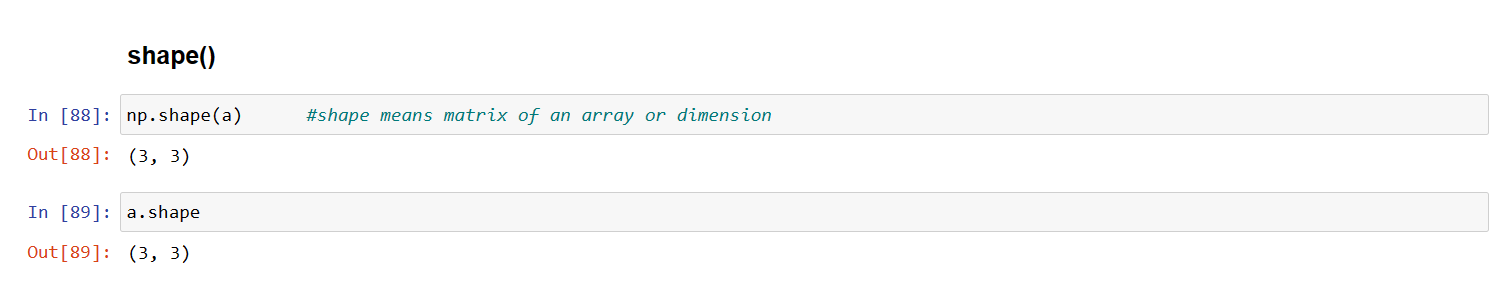
**Array Properties:**

**size()**

It gives the total number of elements of the array. This is equal to the product of the elements of shape.

**shape()**

ndarray.shape: It gives the sequence of integers indicating the size of the array for each dimension.



**dtype()**

The data type of the elements of the array. All the elements of an array are of same data type. Common data types are int32, int64, float32, float64, U32, etc.



**Array Operations:**

**Accessing and Slicing operations**

Sometimes we need to extract part of an array. This is done through slicing. We can define which part of the array to be sliced by specifying the start and end index values using [start : end] along with the array name.



**copy()**

In NumPy, the copy() function is not a standalone function like linspace() or logspace(). Instead, it is a method that can be applied to an existing array to create a copy of that array. This is important when you want to modify one array without affecting the original.



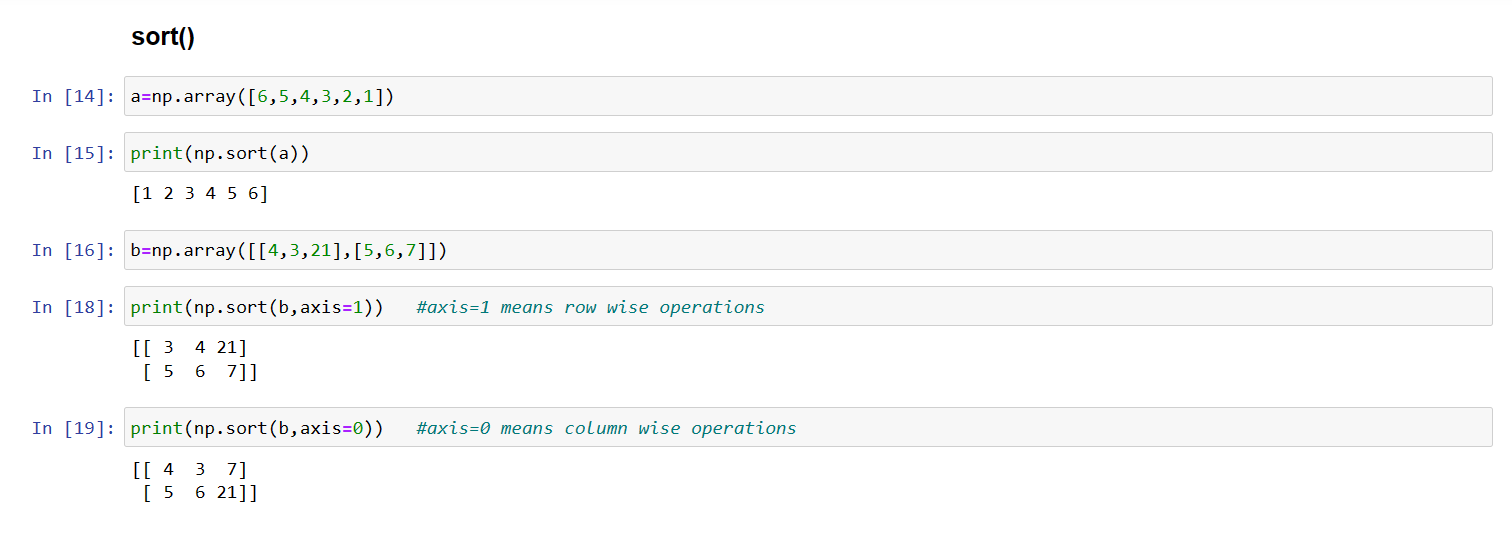
**view()**

In NumPy, the view() method is used to create a view of an array with the same data but potentially different shape or strides. It returns a new array object that looks at the same data as the original array but can have a different shape, size, or strides. Changes made to the view affect the original array, and vice versa.

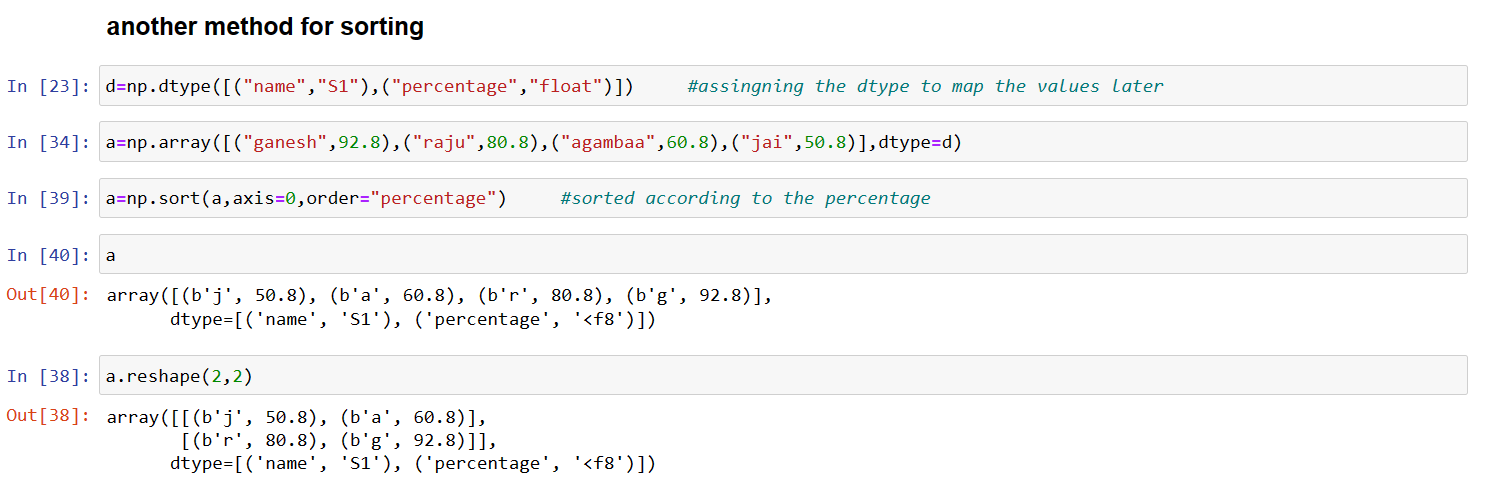


**sort()**

Sorting is to arrange the elements of an array in hierarchical order either ascending or descending. By default, numpy does sorting in ascending order.

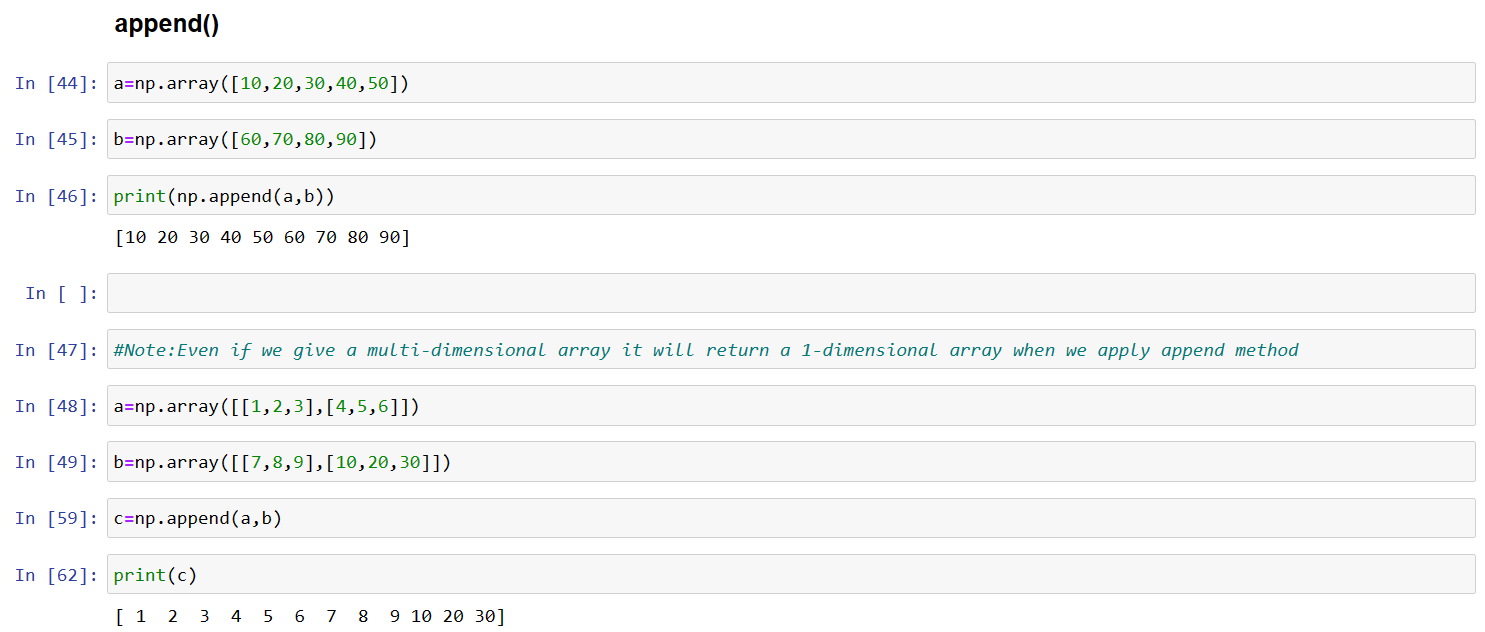


**Another method for sorting:**

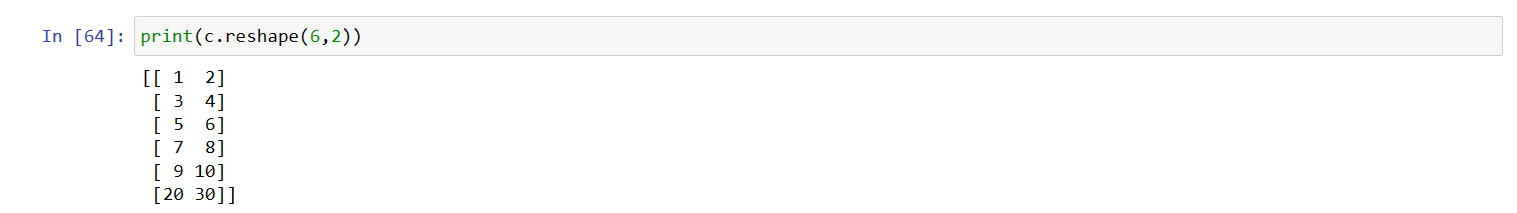


**append()**

In NumPy, the append() function is used to append values to the end of an array. It creates a new array with the combined values of the original array and the values to be appended. It's important to note that append() returns a new array and does not modify the original array in-place.

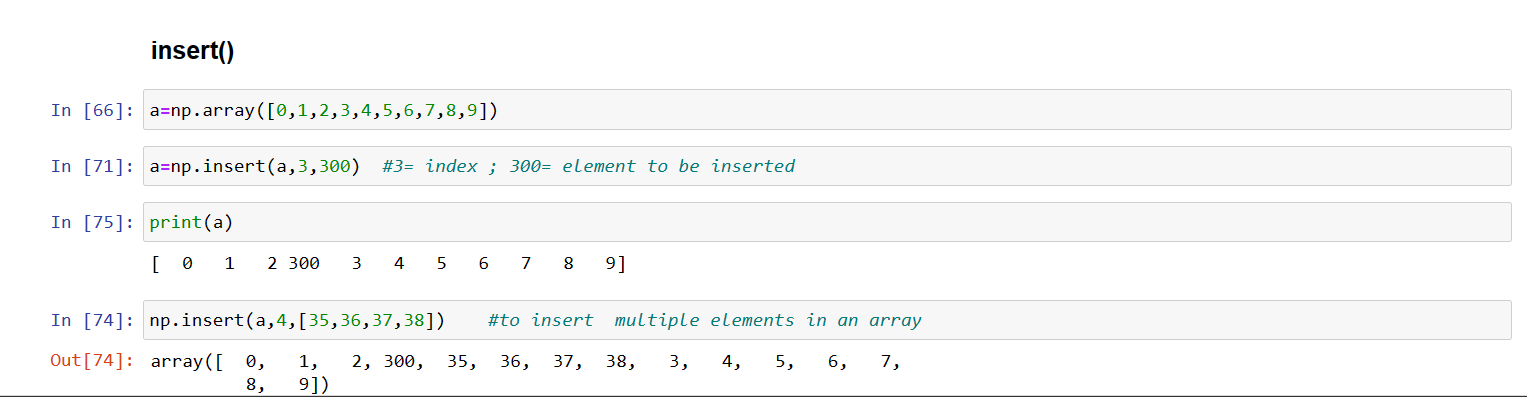


**we can change the dimensions using .reshape()**



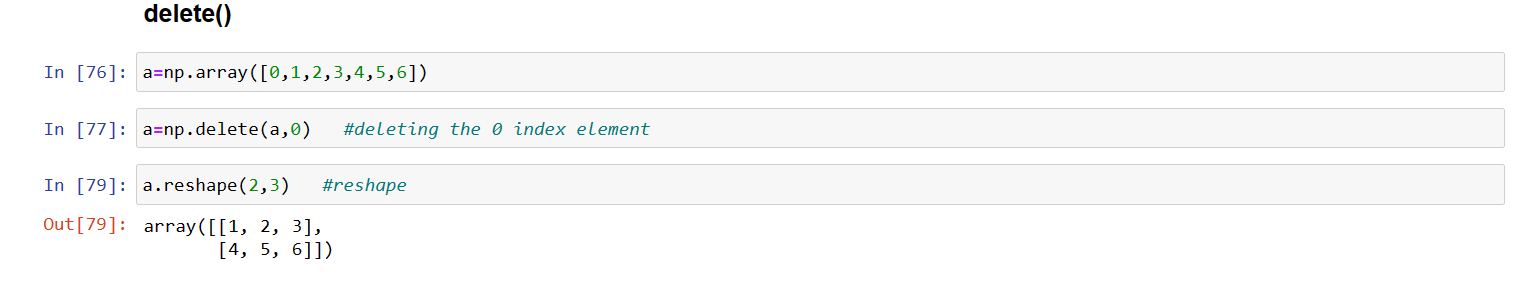
**insert()**

In NumPy, the insert() function is used to insert values along a specified axis before a given index. It creates a new array with the inserted values, and it does not modify the original array in-place.



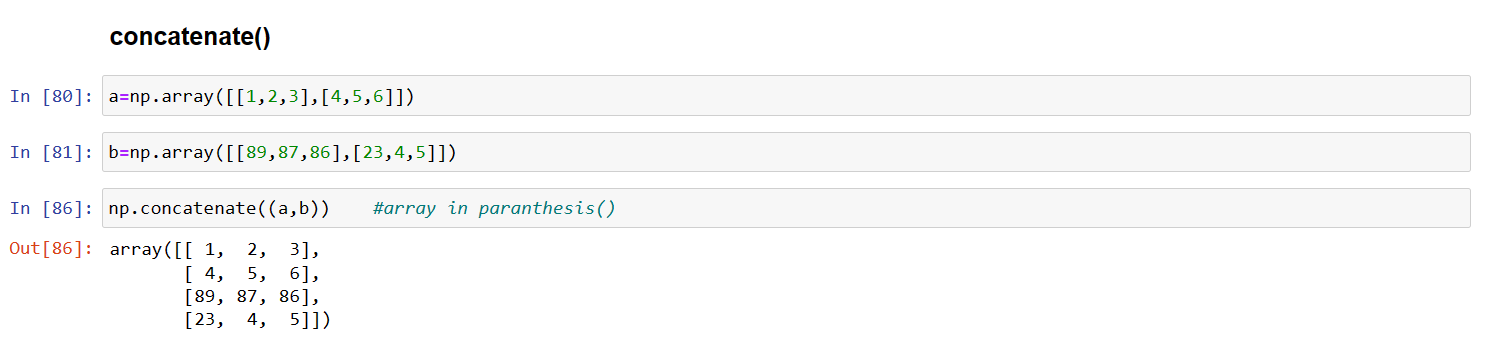
**delete()**

In NumPy, the delete() function is used to delete elements from an array along a specified axis. It returns a new array with the specified elements removed. Like other array manipulation functions in NumPy, delete() does not modify the original array in-place.



**concatenate()**

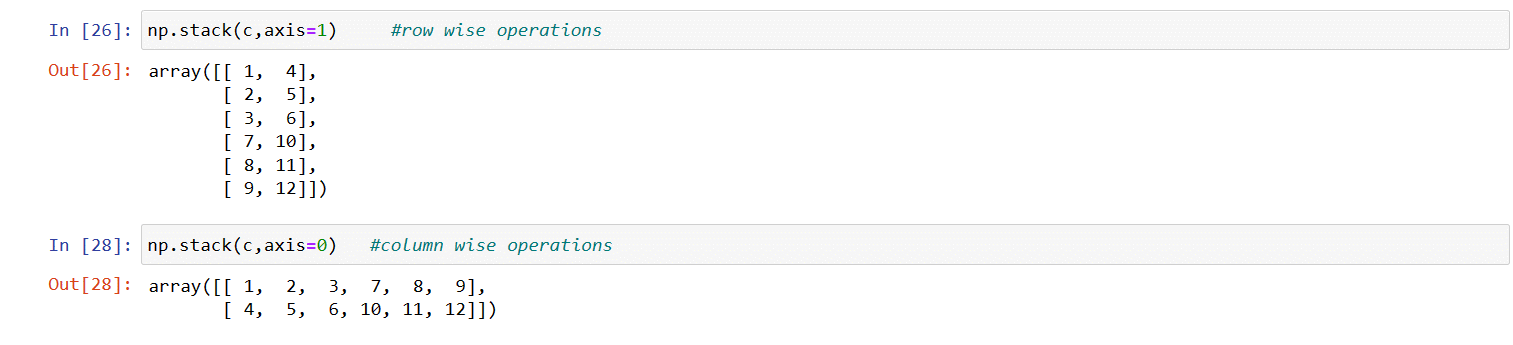
Concatenation means joining two or more arrays. Concatenating 1-D arrays means appending the sequences one after another.



**stack()**

In NumPy, the stack() function is used to join arrays along a new axis. It takes a sequence of arrays and stacks them along a new axis, which is specified by the axis parameter.





**vstack() vertical stack**

In NumPy, the vstack() function is a specific case of the more general numpy.concatenate() function. It is used to vertically stack arrays along the first axis (axis 0). This means it concatenates arrays vertically, increasing the number of rows.



**hstack() horizontal stack**

In NumPy, the hstack() function is used to horizontally stack arrays along the second axis (axis 1). This means it concatenates arrays horizontally, increasing the number of columns.

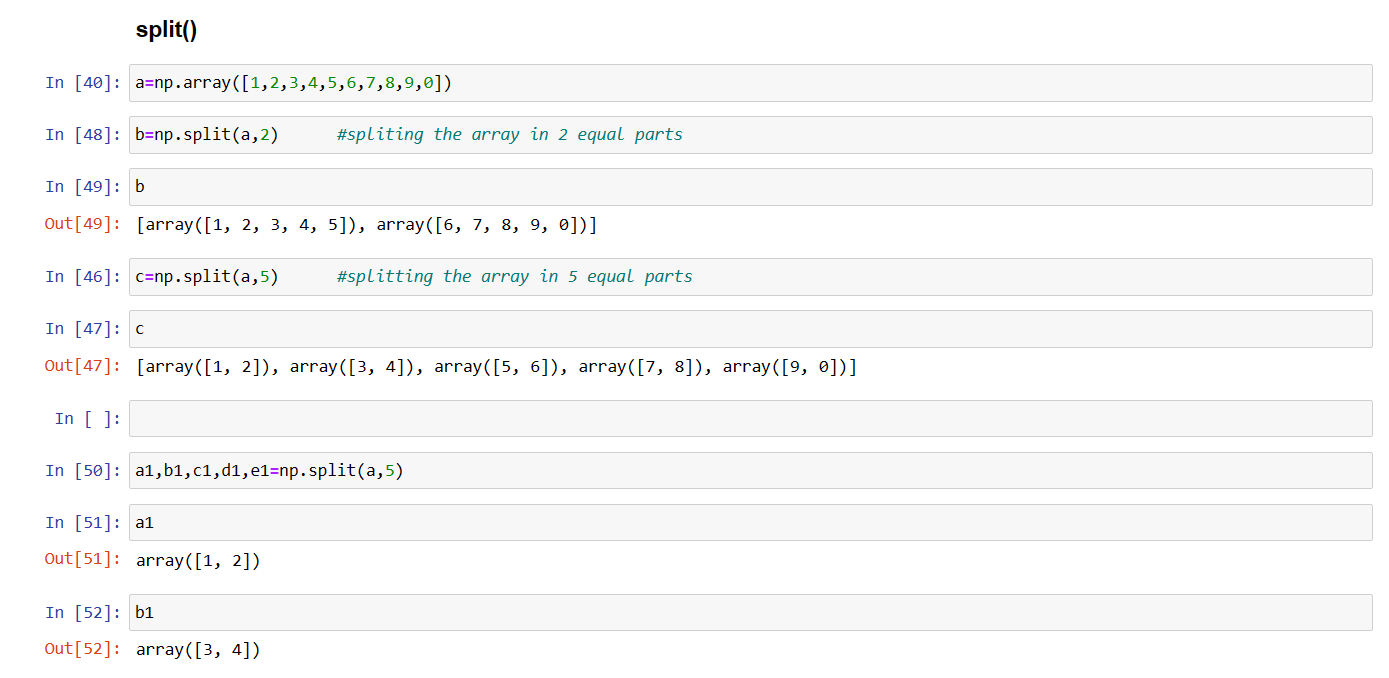


**dstack()**

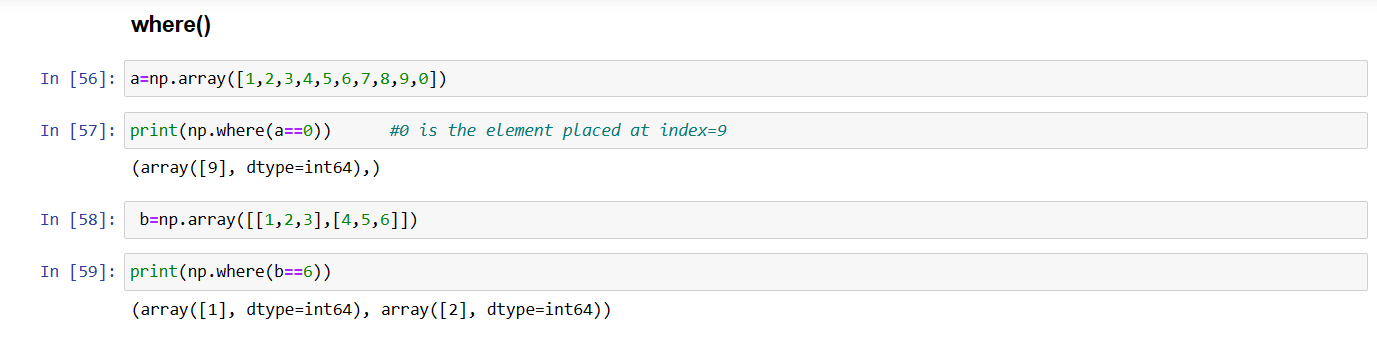


**split()**

numpy.split() slices apart an array into multiple sub-arrays along an axis.



**where()**



**searchsorted()**



**Arthematic Operations**

Arithmetic operations on NumPy arrays are fast and simple. When we perform a basic arithmetic operation like addition, subtraction, multiplication, division etc. on two arrays, the operation is done on each corresponding pair of elements. For instance, adding two arrays will result in the first element in the first array to be added to the first element in the second array, and so on. Consider the following element-wise operations on two arrays:

**Note:** It is important to note that for element-wise operations, size of both arrays must be same.

**add()**



**subtract()**



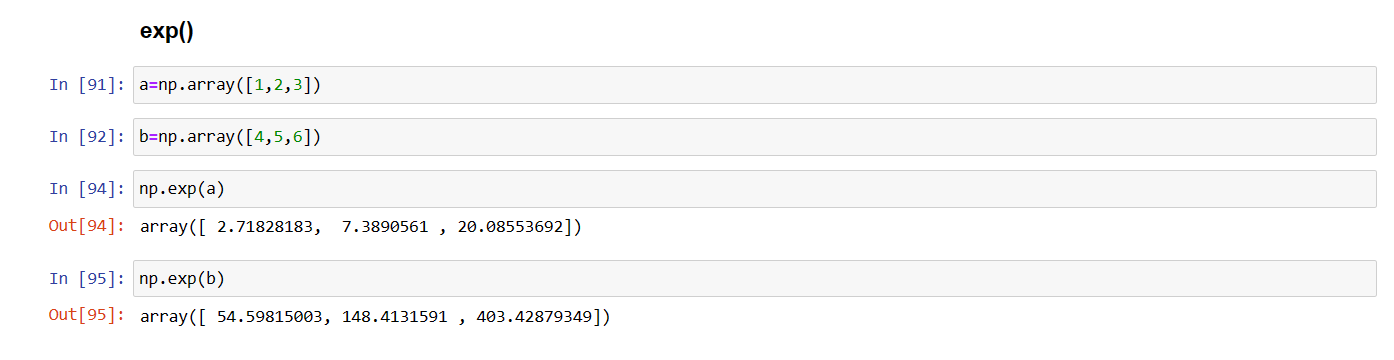
**multiply()**



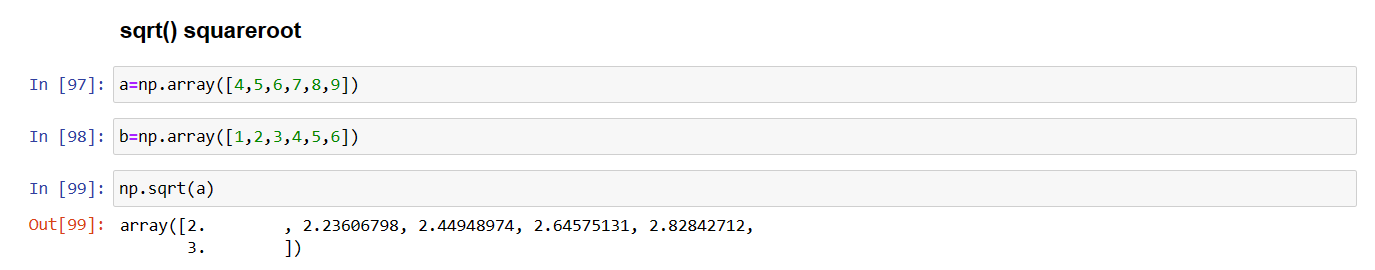
**divide()**



**exp()**



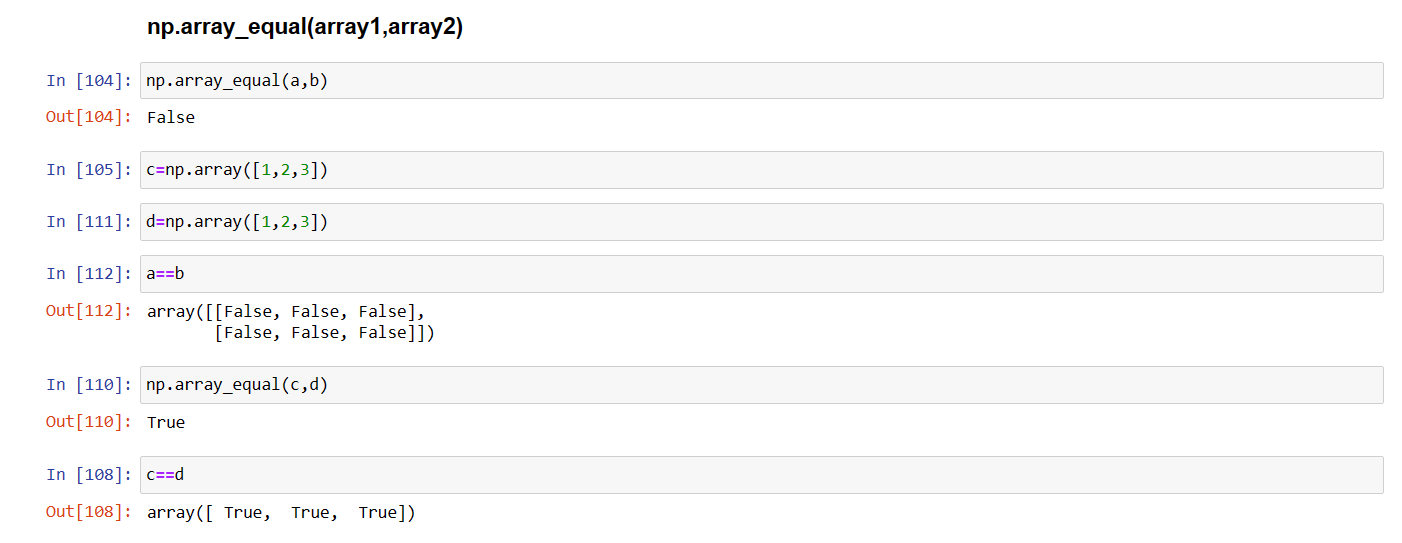
**sqrt() square root**



**Comparision**

**np.array\_equal(array1,array2)**





**Array Functions**

**Statistical Operations on Arrays**

**sum() , min() , max() , mean() , median() , var() , std()**

* The max() function finds the maximum element from an array.
* The min() function finds the minimum element from an array.
* The sum() function finds the sum of all elements of an array.
* The mean() function finds the average of elements of the array.
* The std() function is used to find standard deviation of an array of elements.



