# **Chapter-1 Basic Introduction to the Numpy**

## **Numpy Introduction**

* Numpy 🡺 Numerical Python Library
* Scipy 🡺 Scientific Python Library

Numpy is the entry point for Data Science

## **Need of Numpy**

Python performs basic mathematical operations.

But in Data Science, Machine Learning, Deep Learning and Artificial Intelligence require complex mathematical operations like

1. Creation of arrays

2. Perform several operations on those arrays

3. Integral calculus operations

4. Differential equations

5. Statistics related operations

These type of operations are not supported by normal Python. Numpy can perform all these operations.

## **Importance of Numpy**

* ndarray (n- dimensional array) is the basic data structure in numpy.
* Numpy is the backbone of remaining libraries like pandas, matplotlib, sklearn (scikit learn) etc.
* Numpy has Vectorization features

## **History Numpy**

* Numpy stands for Numerical Python
* It is the fundamental python library to perform complex numerical operations
* Numpy is developed on top of Numeric library in 2005.
* Numeric library developed by Jim Hugunin.
* Numpy is developed by Travis Oliphant and multiple contributors
* Numpy is freeware and open source library
* Numpy library is written in C and Python
* C language 🡺 performance is very high
* Most of Numpy is written in C, so performance-wise Numpy is the best
* Because of high speed, numpy is best choice for ML algorithms than traditions python's in-built data structures like List.

## **Features of Numpy**

1. Superfast because most of numpy is written in C language
2. Numpy Array (nd array) basic data structure in Numpy.
3. It is the backbone for remaining libraries like pandas, scikit-learn etc.
4. Numpy has vectorization feature which improves performance while iterating elements.

## **ndarray in Numpy**

* In numpy data is stored in the form of array.
* In numpy we can hold data by using Array Data Structure.
* The Arrays which are created by using numpy are called nd arrays
* nd array ==> N-Dimensional Array or Numpy Array
* This nd array is most commonly used in Data Science libraries like pandas, mscikit learn etc.

## **Application Areas of Numpy**

* To perform linear algebra functions
* To perform linear regression
* To perform logistic regression
* Deep Neural Networks
* K-means clustering
* Control Systems
* Operational Research

## **What is an Array**

* An indexed collection of homogenous data elements is nothing but array.
* It is the most commonly used concept in programming language like C/C++, java etc.
* By default, arrays concept is not available in python, instead we can use List. (But make sure list and array both are not same)

But in Python, we can create arrays in the following 2 ways:

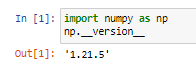
• By using array module

• By using numpy module

to install particular version of numpy 🡺 pip install numpy==1.20.0

**Note:** When we install Anaconda python. Then there is no need to install numpy explicitly. numpy module is implicitly installed along with Anaconda python.

# Check the version of Numpy installed



## **Python List Vs Numpy Array**

### **Similarities:**

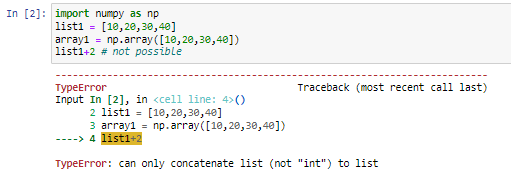
1. Both can be used to store data
2. The order will be preserved in both. Hence indexing and slicing concepts are applicable.
3. Both are mutable; i.e. we can change the content.

### **Differences:**

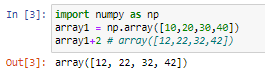
1. list is python's inbuilt type. we have to install and import numpy explicitly.
2. List can contain heterogeneous elements. But array contains only homogeneous elements.
3. On list, we cannot perform vector operations. But on ndarray we can perform vector operations.
4. Arrays consume less memory than list.
5. Arrays are superfast when compared with list.
6. Numpy arrays are more convenient to use while performing complex mathematical operations

# On list, we cannot perform vector operations.

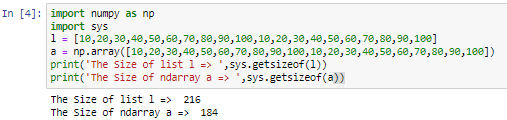
# But on ndarray we can perform vector operations.



# On ndarrays we can perform vector operations

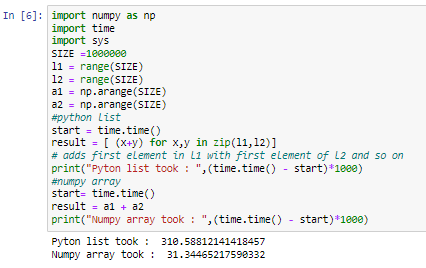


# Arrays consume less memory than list.



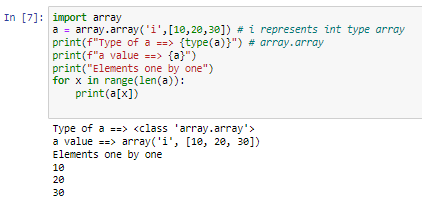
# Arrays are superfast when compared with list.





# **Chapter-2 Creation of Numpy Arrays**

## **Array (ndarray) creation using array module**



Note: array module is not recommended to use because much library support is not available

## **Array (ndarray) creation using Numpy**

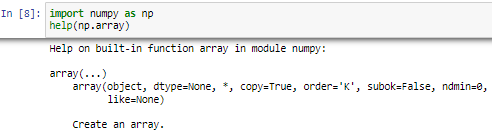
1. array()
2. arange()
3. linspace()
4. zeros()
5. ones()
6. full()
7. eye()
8. identity()
9. diag()
10. empty()
11. np.random module
    1. randint()
    2. rand()
    3. uniform()
    4. randn()
    5. normal()
    6. shuffle()

**Note:**

• np.random module is developed based on array.

• So performance of this module is more when compared with normal python random module

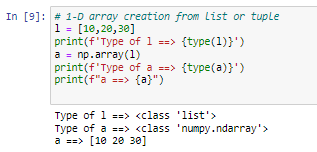
## **1. array()**



help(np.array) 🡺 gives documentation

## 

## **1-D array(Vector) creation using list or tuple**



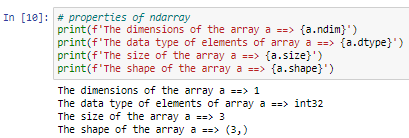
### **To know properties of ndarray**

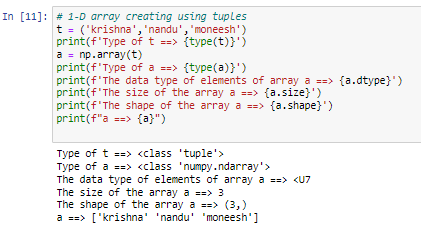
• np.ndim 🡺 to know the dimension of the ndarray

• np.dtype 🡺 to know the data type of the elements in the ndarray

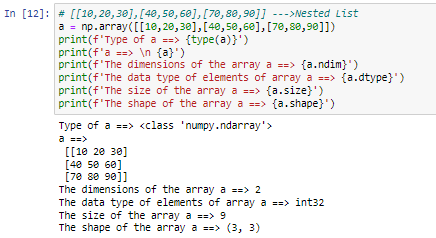
• np.size 🡺 to know the total number of elements in the array

• np.shape 🡺 returns the shape of an array in tuple form





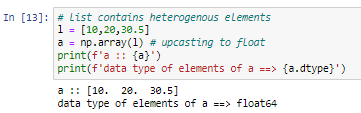
### **2-D array(Matrix) creation using Nested Lists**



**Note:**

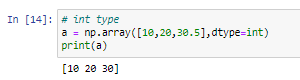
• Array contains only homogenous elements

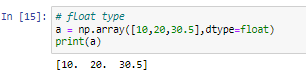
• If list contains heterogeneous elements, and while creating the array up casting will be performed.

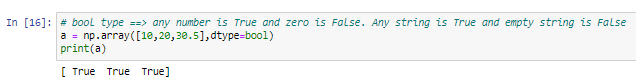


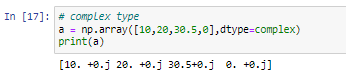
### **Creating arrays with particular datatype**

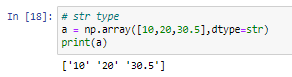
we have to use dtype argument while creating the array





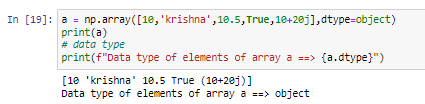






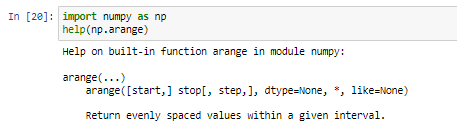
### **creating object type array**

* object is the parent of all int, float, bool, complex and str
* Here the elements are looks like heterogeneous. But the datatype of elements is 'object



## **2. arange()**

* we can create only 1-D arrays with arange() function



### **Python:**

1. range(n) ==> n values from 0 to n-1

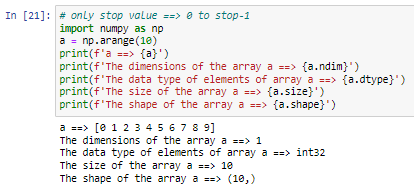
* E.g. range(4) ==> 0,1,2,3

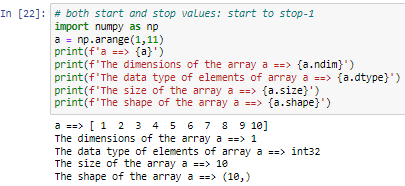
2. range(m,n)==> from m to n-1

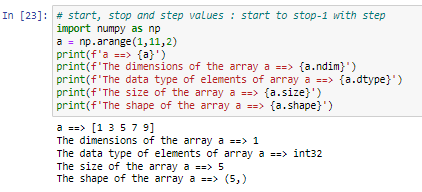
* range(2,7)==> 2,3,4,5,6

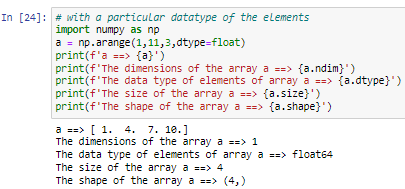
3. range(begin,end,step)

* range(1,11,1) ==> 1,2,3,4,5,6,7,8,9,10
* range(1,11,2) ==> 1,3,5,7,9
* range(1,11,3) ==> 1,4,7,10



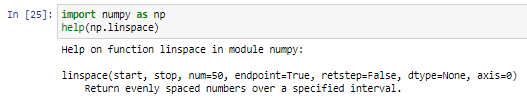






## **linspace()**

same as arange() only but in the specified interval, linearly spaced values



**linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None, axis=0)**

* Both start and stop are includded because endpoint=True
* If endpoint=False then stop is excluded
* retstep denotes the spacing between the points. If True then the value is returned
* calculation of spacing (stop-start)/(num-1) if endpoint=True
* calculation of spacing (stop-start)/(num) if endpoint=False

