**What is NumPy?**

* NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.
* NumPy is a Python library.
* NumPy is a Python library used for working with arrays.
* NumPy is short for "Numerical Python".
* NumPy can be used to perform a wide variety of mathematical operations on arrays.
* It adds powerful data structures to Python that guarantee efficient calculations with arrays and matrices.
* it supplies an enormous library of high-level mathematical functions that operate on these arrays and matrices.
* NumPy is a general-purpose array-processing package.
* It is the fundamental package for scientific computing with Python.
* It is open-source software.
* It also has functions for working in domain of linear algebra, fourier transform, and matrices.

## Why Use NumPy?

* In Python we have lists that serve the purpose of arrays, but they are slow to process.
* NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.
* The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.
* Arrays are very frequently used in data science, where speed and resources are very important.

**Arrays in NumPy**

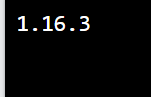
* NumPy is used to work with arrays. The array object in NumPy is called ndarray.
* We can create a NumPy ndarray object by using the array() function.
* **type():** This built-in Python function tells us the type of the object passed to it.
* To create an ndarray, we can pass a list, tuple or any array-like object into the array() method, and it will be converted into an ndarray

## Checking NumPy Version

The version string is stored under \_\_version\_\_ attribute.

import numpy as np  
  
print(np.\_\_version\_\_)

output:



**1st program:**

**import** numpy as np

# Creating array object

arr **=** np.array( [[ 1, 2, 3],

                 [ 4, 2, 5]] )

# Printing type of arr object

print("Array is of type: ", type(arr))

# Printing array dimensions (axes)

**print**("No. of dimensions: ", arr.ndim)

# Printing shape of array

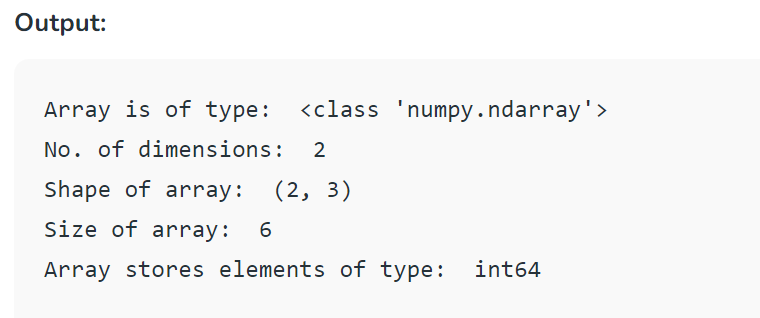
**print**("Shape of array: ", arr.shape)

# Printing size (total number of elements) of array

print("Size of array: ", arr.size)

# Printing type of elements in array

print("Array stores elements of type: ", arr.dtype)



* **Creating a Numpy Array**  
  Arrays in Numpy can be created by multiple ways, with various number of Ranks, defining the size of the Array.
* Arrays can also be created with the use of various data types such as lists, tuples, etc.
* The type of the resultant array is deduced from the type of the elements in the sequences.  
  **Note:** Type of array can be explicitly defined while creating the array.

# Python program for

# Creation of Arrays

**import** numpy as np

# Creating a rank 1 Array

arr **=** np.array([1, 2, 3])

print("Array with Rank 1: \n",arr)

# Creating a rank 2 Array

arr **=** np.array([[1, 2, 3],

                [4, 5, 6]])

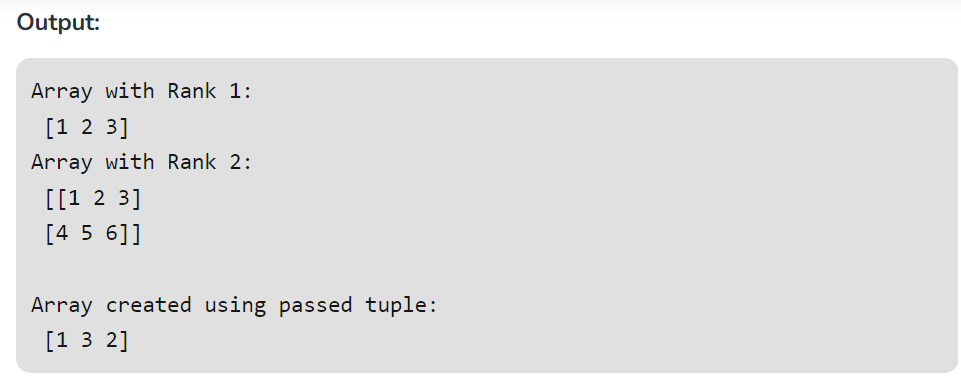
print("Array with Rank 2: \n", arr)

# Creating an array from tuple

arr **=** np.array((1, 3, 2))

print("\nArray created using "

      "passed tuple:\n", arr)



**2nd Programe:**

**Accessing the array Index**  
In a numpy array, indexing or accessing the array index can be done in multiple ways.

To print a range of an array, slicing is done. Slicing of an array is defining a range in a new array which is used to print a range of elements from the original array. Since, sliced array holds a range of elements of the original array, modifying content with the help of sliced array modifies the original array content.

# Python program to demonstrate

# indexing in numpy array

**import** numpy as np

# Initial Array

arr **=** np.array([[**-**1, 2, 0, 4],

                [4, **-**0.5, 6, 0],

                [2.6, 0, 7, 8],

                [3, **-**7, 4, 2.0]])

print("Initial Array: ")

print(arr)

# Printing a range of Array

# with the use of slicing method

sliced\_arr **=** arr[:2, ::2]

print ("Array with first 2 rows and"

    " alternate columns(0 and 2):\n", sliced\_arr)

# Printing elements at

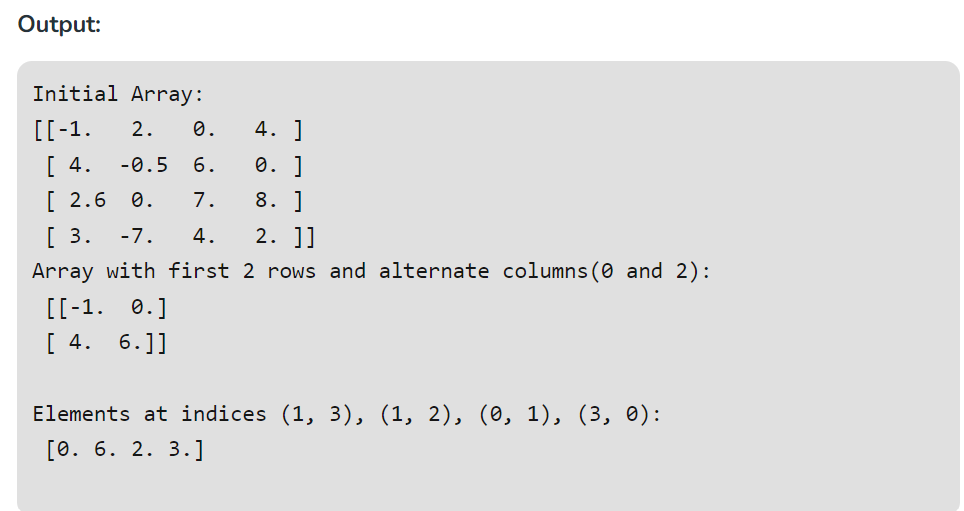
# specific Indices

Index\_arr **=** arr[[1, 1, 0, 3],

                [3, 2, 1, 0]]

print ("\nElements at indices (1, 3), "

    "(1, 2), (0, 1), (3, 0):\n", Index\_arr)



**3rd program**

**Basic Array Operations**  
In numpy, arrays allow a wide range of operations which can be performed on a particular array or a combination of Arrays. These operation include some basic Mathematical operation as well as Unary and Binary operations.

# Python program to demonstrate

# basic operations on single array

**import** numpy as np

# Defining Array 1

a **=** np.array([[1, 2],

              [3, 4]])

# Defining Array 2

b **=** np.array([[4, 3],

              [2, 1]])

# Adding 1 to every element

print ("Adding 1 to every element:", a **+** 1)

# Subtracting 2 from each element

print ("\nSubtracting 2 from each element:", b **-** 2)

# sum of array elements

# Performing Unary operations

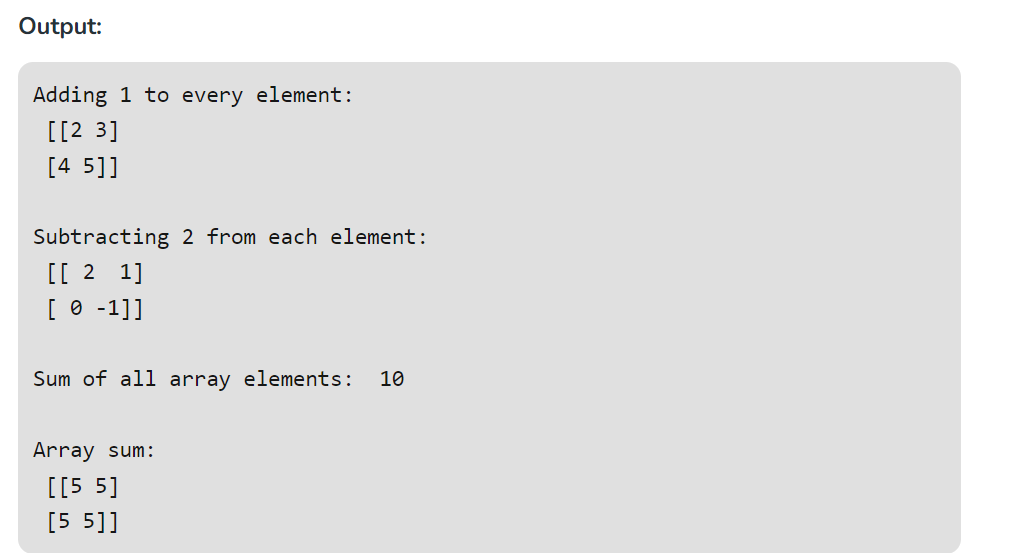
print ("\nSum of all array "

       "elements: ", a.sum())

# Adding two arrays

# Performing Binary operations

print ("\nArray sum:\n", a **+** b)

****

**4th program**

**Constructing a Datatype Object**  
In Numpy, datatypes of Arrays need not to be defined unless a specific datatype is required. Numpy tries to guess the datatype for Arrays which are not predefined in the constructor function.

# Python Program to create

# a data type object

**import** numpy as np

# Integer datatype

# guessed by Numpy

x **=** np.array([1, 2])

print("Integer Datatype: ")

print(x.dtype)

# Float datatype

# guessed by Numpy

x **=** np.array([1.0, 2.0])

print("\nFloat Datatype: ")

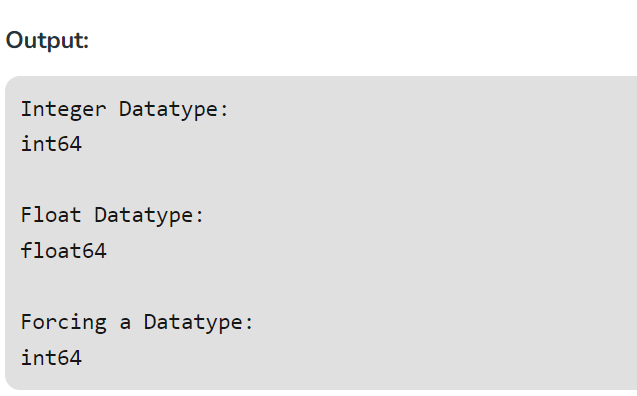
print(x.dtype)

# Forced Datatype

x **=** np.array([1, 2], dtype **=** np.int64)

print("\nForcing a Datatype: ")

print(x.dtype)

****

**5th program**

**Math Operations on DataType array**  
In Numpy arrays, basic mathematical operations are performed element-wise on the array. These operations are applied both as operator overloads and as functions. Many useful functions are provided in Numpy for performing computations on Arrays such as **sum**: for addition of Array elements, **T**: for Transpose of elements, etc.

# Python Program to create

# a data type object

**import** numpy as np

# First Array

arr1 **=** np.array([[4, 7], [2, 6]],

                 dtype **=** np.float64)

# Second Array

arr2 **=** np.array([[3, 6], [2, 8]],

                 dtype **=** np.float64)

# Addition of two Arrays

Sum **=** np.add(arr1, arr2)

print("Addition of Two Arrays: ")

print(Sum)

# Addition of all Array elements

# using predefined sum method

Sum1 **=** np.sum(arr1)

print("\nAddition of Array elements: ")

print(Sum1)

# Square root of Array

Sqrt **=** np.sqrt(arr1)

print("\nSquare root of Array1 elements: ")

print(Sqrt)

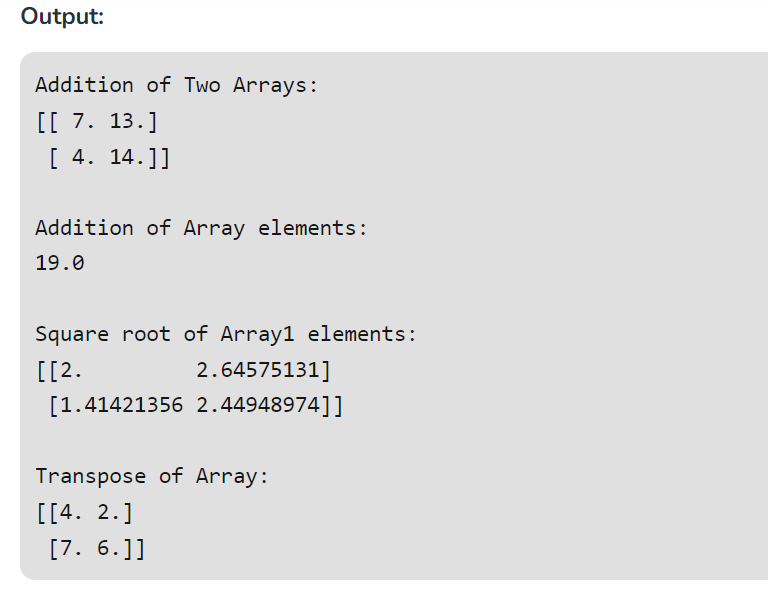
# Transpose of Array

# using In-built function 'T'

Trans\_arr **=** arr1.T

print("\nTranspose of Array: ")

print(Trans\_arr)

****

**6th program**

**Reshaping array:** We can use **reshape** method to reshape an array. Consider an array with shape (a1, a2, a3, …, aN). We can reshape and convert it into another array with shape (b1, b2, b3, …, bM). The only required condition is a1 x a2 x a3 … x aN = b1 x b2 x b3 … x bM. (i.e. the original size of the array remains unchanged.)

# Reshaping 3X4 array to 2X2X3 array

arr **=** np.array([[1, 2, 3, 4],

                [5, 2, 4, 2],

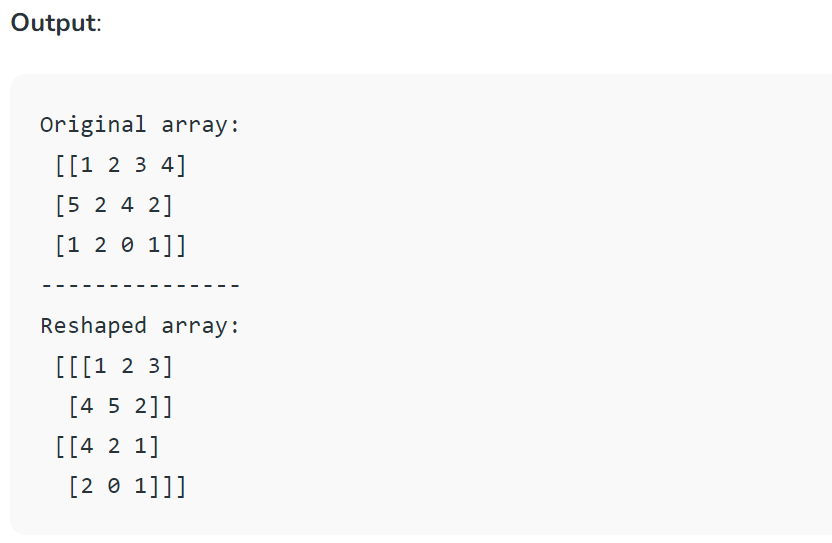
                [1, 2, 0, 1]])

newarr **=** arr.reshape(2, 2, 3)

**print** ("Original array:\n", arr)

**print**("---------------")

**print** ("Reshaped array:\n", newarr)

****

**7th program:**

**Flatten array:** We can use **flatten** method to get a copy of the array collapsed into **one dimension**. It accepts order argument. The default value is (for row-major order). Use for column-major order.

# Flatten array

arr **=** np.array([[1, 2, 3], [4, 5, 6]])

flat\_arr **=** arr.flatten()

print ("Original array:\n", arr)

print ("Fattened array:\n", flat\_arr)



# Matrix manipulation in Python

In python matrix can be implemented as 2D [list](https://www.geeksforgeeks.org/python-set-3-strings-lists-tuples-iterations/) or 2D Array. Forming matrix from latter, gives the additional functionalities for performing various operations in matrix.

**Operation on Matrix :**

* **1. add() :-** This function is used to perform**element wise matrix addition**.
* **2. subtract() :-** This function is used to perform **element wise matrix subtraction**.
* **3. divide() :-** This function is used to perform**element wise matrix division**.

# Python code to demonstrate matrix operations

# add(), subtract() and divide()

# importing numpy for matrix operations

**import** numpy

# initializing matrices

x **=** numpy.array([[1, 2], [4, 5]])

y **=** numpy.array([[7, 8], [9, 10]])

# using add() to add matrices

print ("The element wise addition of matrix **is** : ")

print (numpy.add(x,y))

# using subtract() to subtract matrices

print ("The element wise subtraction of matrix **is** : ")

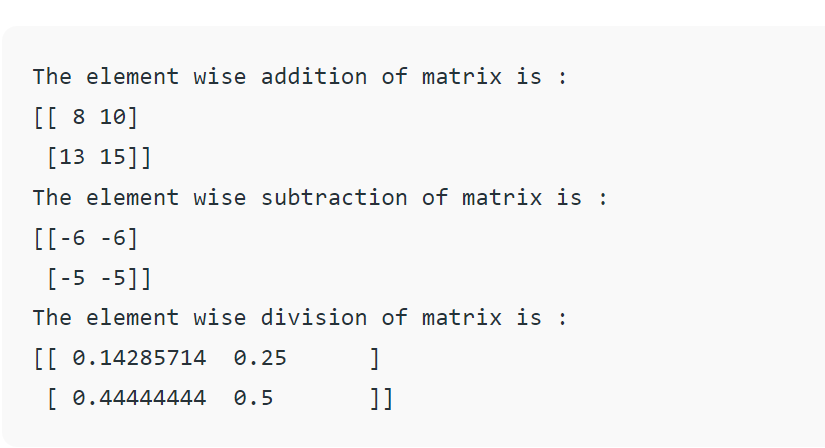
print (numpy.subtract(x,y))

# using divide() to divide matrices

print ("The element wise division of matrix **is** : ")

print (numpy.divide(x,y))

output:

****

* **4. multiply() :-** This function is used to perform **element wise matrix multiplication**.
* **5. dot() :-** This function is used to compute the**matrix multiplication, rather than element wise multiplication**.

# Python code to demonstrate matrix operations

# multiply() and dot()

# importing numpy for matrix operations

**import** numpy

# initializing matrices

x **=** numpy.array([[1, 2], [4, 5]])

y **=** numpy.array([[7, 8], [9, 10]])

# using multiply() to multiply matrices element wise

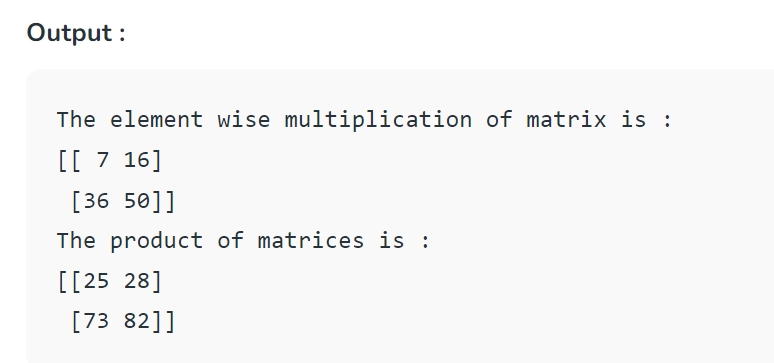
**print** ("The element wise multiplication of matrix **is** : ")

print (numpy.multiply(x,y))

# using dot() to multiply matrices

print ("The product of matrices **is** : ")

print (numpy.dot(x,y))

****

* **6. sqrt() :-** This function is used to compute the **square root of each element** of matrix.
* **7. sum(x,axis) :-**This function is used to **add all the elements in matrix**. Optional “axis” argument computes the **column sum if axis is 0** and **row sum if axis is 1**.
* **8. “T” :-** This argument is used to **transpose** the specified matrix.

# Python code to demonstrate matrix operations

# sqrt(), sum() and "T"

# importing numpy for matrix operations

**import** numpy

# initializing matrices

x **=** numpy.array([[1, 2], [4, 5]])

y **=** numpy.array([[7, 8], [9, 10]])

# using sqrt() to print the square root of matrix

print ("The element wise square root **is** : ")

**print** (numpy.sqrt(x))

# using sum() to print summation of all elements of matrix

print ("The summation of all matrix element **is** : ")

print (numpy.sum(y))

# using sum(axis=0) to print summation of all columns of matrix

**print** ("The column wise summation of all matrix  **is** : ")

**print** (numpy.sum(y,axis**=**0))

# using sum(axis=1) to print summation of all rows of matrix

**print** ("The row wise summation of all matrix  **is** : ")

**print** (numpy.sum(y,axis**=**1))

# using "T" to transpose the matrix

print ("The transpose of given matrix **is** : ")

print (x.T)

**Output :**

The element wise square root is :

[[ 1. 1.41421356]

[ 2. 2.23606798]]

The summation of all matrix element is :

34

The column wise summation of all matrix is :

[16 18]

The row wise summation of all matrix is :

[15 19]

The transpose of given matrix is :

[[1 4]

[2 5]]

### Using nested loops:

* Define matrices A and B.
* Get the number of rows and columns of the matrices using the len() function.
* Initialize matrices C, D, and E with zeros using nested loops or list comprehension.
* Use nested loops or list comprehension to perform the element-wise addition, subtraction, and division of matrices.
* Print the resulting matrices C, D, and E.

|  |
| --- |
| A **=** [[1,2],[4,5]]  B **=** [[7,8],[9,10]]  rows **=** len(A)  cols **=** len(A[0])    # Element wise addition  C **=** [[0 **for** i **in** range(cols)] **for** j **in** range(rows)]  **for** i **in** range(rows):  **for** j **in** range(cols):          C[i][j] **=** A[i][j] **+** B[i][j]  **print**("Addition of matrices: \n", C)    # Element wise subtraction  D **=** [[0 **for** i **in** range(cols)] **for** j **in** range(rows)]  **for** i **in** range(rows):  **for** j **in** range(cols):          D[i][j] **=** A[i][j] **-** B[i][j]  print("Subtraction of matrices: \n", D)    # Element wise division  E **=** [[0 **for** i **in** range(cols)] **for** j **in** range(rows)]  **for** i **in** range(rows):  **for** j **in** range(cols):          E[i][j] **=** A[i][j] **/** B[i][j]  print("Division of matrices: \n", E) |

**Output**

Addition of matrices:

[[8, 10], [13, 15]]

Subtraction of matrices:

[[-6, -6], [-5, -5]]

Division of matrices:

[[0.14285714285714285, 0.25], [0.4444444444444444, 0.5]]