In [1]:

```
conda install numpy
```

```
Collecting package metadata (current_repodata.json): ...working... done
Solving environment: ...working... done

# All requested packages already installed.


Note: you may need to restart the kernel to use updated packages.
```

In [1]:

```
import numpy as np
```

Numpy

- *NumPy stands for Numerical Python.*
- *NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.*
- *NumPy is the fundamental package for scientific computing in Python.*
- *NumPy is a Python library used for working with arrays.*
- *It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.*

Why Numpy?

- *In Python we have lists that serve the purpose of arrays, but they are slow to process.*
- *NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently.*
- *This is the main reason why NumPy is faster than lists. Also it is optimized to work with latest CPU architectures.*

Importing Numpy

Once NumPy is installed, import it in your applications by adding the `import` *keyword:*

In [2]:

```
import numpy
```

Now NumPy is imported and ready to use.

NumPy as np

`NumPy` is usually imported under the `np` alias.

Create an alias with the `as` keyword while importing.

Now the NumPy package can be referred to as `np` instead of `numpy`.

In [4]:

```
import numpy as np
```

Checking NumPy Version

The version string is stored under `__version__` attribute.

In [3]:

```
print(np.__version__)
```

```
1.21.5
```

NumPy Creating Arrays

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.

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:

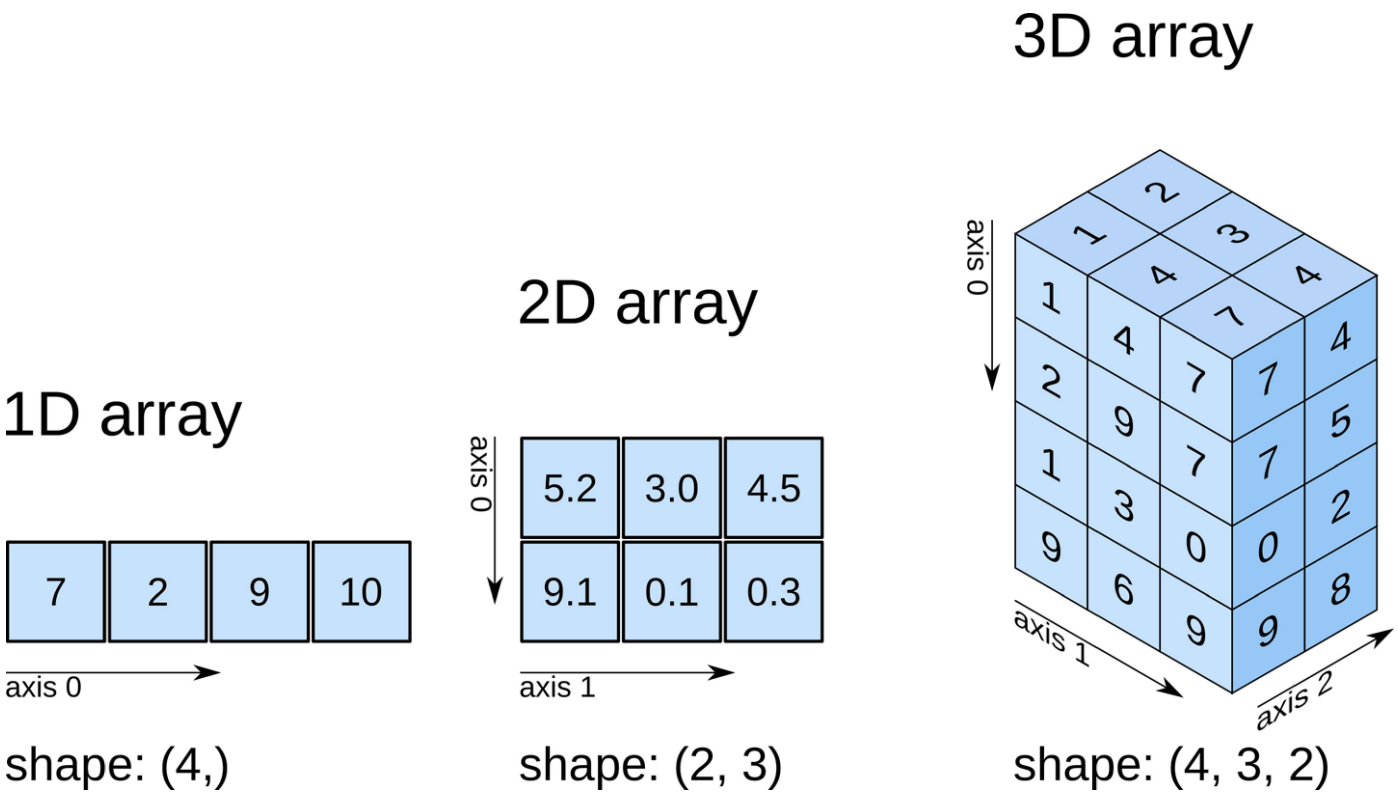
In [7]:

```
arr = np.array([1, 2, 3, 4, 5])
print(arr)
print(type(arr))
```

```
[1 2 3 4 5]
<class 'numpy.ndarray'>
```

Dimensions in Arrays

A dimension in arrays is one level of array depth (nested arrays).



0-D Arrays

0-D arrays, or Scalars, are the elements in an array. Each value in an array is a 0-D array

In [3]:

```
arr = np.array(42)
print(arr)
```

```
42
```

1-D Arrays

An array that has 0-D arrays as its elements is called uni-dimensional or 1-D array.

These are the most common and basic arrays.

In [4]:

```
arr = np.array([1, 2, 3, 4, 5])
print(arr)
```

```
[1 2 3 4 5]
```

2-D Arrays

An array that has 1-D arrays as its elements is called a 2-D array.

These are often used to represent matrix or 2nd order tensors.

In [6]:

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

```
[[1 2 3]
 [4 5 6]]
```

3-D arrays

An array that has 2-D arrays (matrices) as its elements is called 3-D array.

In [7]:

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

```
[[[1 2 3]
  [4 5 6]]

 [[1 2 3]
  [4 5 6]]]
```

Higher Dimensional Arrays

An array can have any number of dimensions.

When the array is created, you can define the number of dimensions by using the `ndmin` argument.

In [12]:

```
arr = np.array([1, 2, 3, 4], ndmin=5)
print(arr)
print('number of dimensions :', arr.ndim)
```

```
[[[[[1 2 3 4]]]]]
number of dimensions : 5
```

NumPy - Data Types

*NumPy supports a much greater variety of numerical types than Python does. *

Each built-in data type has a character code that uniquely identifies it.

- 'b' − boolean
- 'i' − (signed) integer
- 'u' − unsigned integer
- 'f' − floating-point
- 'c' − complex-floating point
- 'm' − timedelta
- 'M' − datetime
- 'O' − (Python) objects
- 'S', 'a' − (byte-)string
- 'U' − Unicode
- 'V' − raw data (void)

Checking the Data Type of an Array

The NumPy array object has a property called `dtype` that returns the data type of the array:

In [13]:

```
arr = np.array([1, 2, 3, 4])
print(arr.dtype)
```

```
int32
```

In [14]:

```
arr = np.array(['apple', 'banana', 'cherry'])
print(arr.dtype)
```

```
<U6
```

NumPy - Array Attributes

1. ndarray.shape

NumPy arrays have an attribute called `shape` that returns a tuple with each index having the number of corresponding elements. It can also be used to resize the array.

In [10]:

```
a = np.array([[1,2,3],[4,5,6]])
print(a)
print(a.shape)
```

```
[[1 2 3]
 [4 5 6]]
(2, 3)
```

a) Reshaping arrays

Reshaping means changing the shape of an array.

The shape of an array is the number of elements in each dimension.

By reshaping we can add or remove dimensions or change number of elements in each dimension.

In [16]:

```
a = np.array([[1,2,3],[4,5,6]])
a.shape = (3,2)
print(a)
```

```
[[1 2]
 [3 4]
 [5 6]]
```

ndarray.reshape

NumPy also provides a `reshape` function to resize an array.

In [17]:

```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
newarr = arr.reshape(4, 3)
print(newarr)
```

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
```

Converting 1D array with 8 elements to a 2D array with 3 elements in each dimension (will raise an error)

In [18]:

```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
newarr = arr.reshape(3, 3)
print(newarr)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [18], in <cell line: 2>()
      1 arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
----> 2 newarr = arr.reshape(3, 3)
      3 print(newarr)

ValueError: cannot reshape array of size 8 into shape (3,3)
```

2. ndarray.ndim

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

In [11]:

```
a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```

```
0
1
2
3
```

3. numpy.itemsize

This array attribute returns the length of each element of array in bytes.

In [20]:

```
a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(a.itemsize)
print(b.itemsize)
print(c.itemsize)
print(d.itemsize)
```

```
4
4
4
4
```

3. ndarray.size()

In Python, `numpy.size()` function count the number of elements along a given axis.

In [22]:

```
a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(a.size)
print(b.size)
print(c.size)
print(d.size)
```

```
1
5
6
12
```

4. ndarray.nbytes

`ndarray.nbytes()` function return total bytes consumed by the elements of the array.

In [23]:

```
a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(a.nbytes)
print(b.nbytes)
print(c.nbytes)
print(d.nbytes)
```

```
4
20
24
48
```

NumPy - Array Creation Routines

A new ndarray object can be constructed by any of the following array creation routines or using a low-level ndarray constructor.

numpy.empty

It creates an uninitialized array of specified shape and dtype. It uses the following constructor –

```
numpy.empty(shape, dtype = float, order = 'C')
```

The Constructor takes the following parameters:

S.No.	Parameter	Description
1	Shape	Shape of an empty array in int or tuple of int

S.No.	Parameter	Description
2	Dtype	Desired output data type.Optional
3	Order	'C' for C-style row-major array, 'F' for FORTAN style column-major array

In [3]:

```
x = np.empty([3,2], dtype = int)
print (x)
```

```
[[3997779 3801155]
 [5242972 7274610]
 [7471207 7143521]]
```

Note **–** *The elements in an array show random values as they are not initialized.*

numpy.zeros

Returns a new array of specified size, filled with zeros.

```
numpy.zeros(shape, dtype = float, order = 'C')
```

The Constructor takes the following parameters:

S.No.	Parameter	Description
1	Shape	Shape of an empty array in int or tuple of int
2	Dtype	Desired output data type.Optional
3	Order	'C' for C-style row-major array, 'F' for FORTAN style column-major array

In [28]:

```
x = np.zeros((5,), dtype = int)
print (x)
```

```
[0 0 0 0 0]
```

In [30]:

```
x = np.zeros((2,2))
print (x)
```

```
[[0. 0.]
 [0. 0.]]
```

numpy.ones

Returns a new array of specified size and type, filled with ones.

```
numpy.ones(shape, dtype = None, order = 'C')
```

The Constructor takes the following parameters:

S.No.	Parameter	Description
1	Shape	Shape of an empty array in int or tuple of int
2	Dtype	Desired output data type.Optional
3	Order	'C' for C-style row-major array, 'F' for FORTAN style column-major array

In [29]:

```
x = np.ones(5)
print (x)
```

```
[1. 1. 1. 1. 1.]
```

In [32]:

```
import numpy as np
x = np.ones([2,2], dtype = int)
print (x)
```

```
[[1 1]
 [1 1]]
```

numpy.identity

Return the identity array. The identity array is a square array with ones on the main diagonal.

*numpy.identity(n, dtype=None, *, like=None)*

In [4]:

```
np.identity(3)
```

Out[4]:

```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

numpy.full

Return a new array of given shape and type, filled with fill_value.

*numpy.full(shape, fill_value, dtype=None, order='C', *, like=None)*

In [5]:

```
np.full((2, 2), 10)
```

Out[5]:

```
array([[10, 10],
       [10, 10]])
```

numpy.copy

Return an array copy of the given object

```
numpy.copy(a, order='K', subok=False)
```

In [7]:

```
a = np.array([1, 'm', [2, 3, 4]], dtype=object)
b = np.copy(a)
b[2][0] = 10
print(a)
```

```
[1 'm' list([10, 3, 4])]
```

Random sampling (numpy.random)

Numpy's random number routines produce pseudo random numbers using combinations of a BitGenerator to create sequences and a Generator to use those sequences to sample from different statistical distributions.

Generate a random integer from 0 to 100:

NumPy offers the `random` module to work with random numbers.

In [16]:

```
from numpy import random
x = random.randint(100)
print(x)
```

```
89
```

Generate a random float from 0 to 1:

The random module's `rand()` method returns a random float between 0 and 1:

In [18]:

```
from numpy import random
x = random.rand()
print(x)
```

```
0.7024881396842702
```

Generate a 1-D array containing 5 random integers from 0 to 100:

The `randint()` method takes a `size` parameter where you can specify the shape of an array.

In [20]:

```
from numpy import random
x=random.randint(100, size=(5))
print(x)
```

```
[ 8 62 27  9 63]
```

Generate a 1-D array containing 5 random floats:

The `rand()` method also allows you to specify the shape of the array.

In [21]:

```
from numpy import random
x = random.rand(5)
print(x)
```

```
[0.00690724 0.74890675 0.11569792 0.81008688 0.14056211]
```

Generate Random Number From Array

The `choice()` method allows you to generate a random value based on an array of values. The `choice()` method takes an array as a parameter and randomly returns one of the values.

In [22]:

```
# Return one of the values in an array:
from numpy import random
x = random.choice([3, 5, 7, 9])
print(x)
```

```
9
```

Generate a 2-D array that consists of the values in the array parameter (3, 5, 7, and 9):

The `choice()` method also allows you to return an array of values. Add a `size` parameter to specify the shape of the array.

In [23]:

```
from numpy import random
x = random.choice([3, 5, 7, 9], size=(3, 5))
print(x)
```

```
[[9 3 5 7 7]
 [7 5 5 9 5]
 [9 9 5 7 3]]
```

Numpy Arrange

numpy.arange

This function returns an ndarray object containing evenly spaced values within a given range. The format of the function is as follows –

```
numpy.arange(start, stop, step, dtype)
```

The Constructor takes the following parameters:

S.No.	Parameter	Description
1	start	The start of an interval.If omitted,defaults to 0
2	stop	The end of an interval (not including this number)
3	step	Spacing between values,default is 1
4	dtype	Data type of resulting ndarray.If not given,data type of input is used

In [33]:

```
x = np.arange(5)
print (x)
```

```
[0 1 2 3 4]
```

In [34]:

```
x = np.arange(5, dtype = float)
print (x)
```

```
[0. 1. 2. 3. 4.]
```

In [35]:

```
x = np.arange(10,20,2)
print (x)
```

```
[10 12 14 16 18]
```

In [56]:

```
x = np.arange(0,20.6,2.3)
print (x)
```

```
[ 0.   2.3  4.6  6.9  9.2 11.5 13.8 16.1 18.4]
```

NumPy - Indexing & Slicing

Indexing

This mechanism helps in selecting any arbitrary item in an array based on its Ndimensional index. Each integer array represents the number of indexes into that dimension. When the index consists of as many integer arrays as the dimensions of the target ndarray, it becomes straightforward.

In [50]:

```
x = np.array([[1, 2], [3, 4], [5, 6]])
print(x[1,1])
```

```
4
```

In [49]:

```
x = np.array([[1, 2], [3, 4], [5, 6]])
y = x[[0,1,2], [0,1,0]]
print (y)
```

```
[1 4 5]
```

The selection includes elements at (0,0), (1,1) and (2,0) from the first array.

Slicing

A Python slice object is constructed by giving start, stop, and step parameters to the built-in slice function. This slice object is passed to the array to extract a part of array.

In [37]:

```
a = np.arange(10)
s = slice(2,7,2)
print (a[s])
```

```
[2 4 6]
```

The same result can also be obtained by giving the slicing parameters separated by a colon : (start:stop:step) directly to the ndarray object.

In [48]:

```
# 1-D array
a = np.arange(10)
print(a[2:7:2])
print(a[2:])
print(a[2:5])
```

```
[2 4 6]
[2 3 4 5 6 7 8 9]
[2 3 4]
```

In [47]:

```
#2-D array
a = np.array([[1,2,3],[3,4,5],[4,5,6]])
print(a)
print(a[1:])
print(a[1:,2])
```

```
[[1 2 3]
 [3 4 5]
 [4 5 6]]
[[3 4 5]
 [4 5 6]]
[5 6]
```

In [14]:

```
#3-D array
x = np.arange(45).reshape(3,3,5)
print(x)
print(x[1:, 0])
print(x[1:, 0:2, 1:4])
```

```
[[[ 0  1  2  3  4]
  [ 5  6  7  8  9]
  [10 11 12 13 14]]

 [[15 16 17 18 19]
  [20 21 22 23 24]
  [25 26 27 28 29]]

 [[30 31 32 33 34]
  [35 36 37 38 39]
  [40 41 42 43 44]]]
[[15 16 17 18 19]
 [30 31 32 33 34]]
[[[16 17 18]
  [21 22 23]]

 [[31 32 33]
  [36 37 38]]]
```

NumPy - Arithmetic Operations

Input arrays for performing arithmetic operations such as `add()`, `subtract()`, `multiply()`, *and* `divide()` *must be either of the same shape or should conform to array broadcasting rules.*

In [53]:

```
a = np.array([1,2,3])

print('First array:')
print(a)
print('Second array:')
b = np.array([4,5,6])
print(b)
print('Add the two arrays:')
print (np.add(a,b))
print('Subtract the two arrays:')
print(np.subtract(a,b))
print('Multiply the two arrays:')
print(np.multiply(a,b))
print('Divide the two arrays:')
print (np.divide(a,b))
print('Power function:')
print(np.power(a,b))
```

```
First array:
[1 2 3]
Second array:
[4 5 6]
Add the two arrays:
[5 7 9]
Subtract the two arrays:
[-3 -3 -3]
Multiply the two arrays:
[ 4 10 18]
Divide the two arrays:
[0.25 0.4  0.5 ]
Power function:
[  1  32 729]
```

In [55]:

```
a = np.arange(9).reshape(3,3)

print('First array:')
print(a)
print('Second array:')
b = np.array([10,10,10])
print(b)
print('Add the two arrays:')
print (np.add(a,b))
print('Subtract the two arrays:')
print(np.subtract(a,b))
print('Multiply the two arrays:')
print(np.multiply(a,b))
print('Divide the two arrays:')
print (np.divide(a,b))
print('Power function:')
print(np.power(a,b))
```

```
First array:
[[0 1 2]
 [3 4 5]
 [6 7 8]]
Second array:
[10 10 10]
Add the two arrays:
[[10 11 12]
 [13 14 15]
 [16 17 18]]
Subtract the two arrays:
[[-10  -9  -8]
 [ -7  -6  -5]
 [ -4  -3  -2]]
Multiply the two arrays:
[[ 0 10 20]
 [30 40 50]
 [60 70 80]]
Divide the two arrays:
[[0.  0.1 0.2]
 [0.3 0.4 0.5]
 [0.6 0.7 0.8]]
Power function:
[[         0          1       1024]
 [     59049    1048576    9765625]
 [  60466176  282475249 1073741824]]
```

NumPy - Statistical Functions

NumPy has quite a few useful statistical functions for finding minimum, maximum, percentile standard deviation and variance, etc. from the given elements in the array.

Order Statistics:

Function	Description
ptp(a[, axis, out,keepdims])	Range of values (maximum - minimum) along an axis

Function	Description
percentile(a, q[, axis, out, ...])	Compute the q-th percentile of the data along the specified axis
nanpercentile(a, q[, axis, out, ...])	Compute the qth percentile of the data along the specified axis, while ignoring nan values
quantile(a, q[, axis, out, overwrite_input, ...])	Compute the q-th quantile of the data along the specified axis
nanquantile(a, q[, axis, out, ...])	Compute the qth quantile of the data along the specified axis, while ignoring nan values

Averages and variances:

Function	Description
median(a[, axis, out, overwrite_input, keepdims])	Compute the median along the specified axis
average(a[, axis, weights, returned, keepdims])	Compute the weighted average along the specified axis
mean(a[, axis, dtype, out, keepdims, where])	Compute the arithmetic mean along the specified axis
std(a[, axis, dtype, out, ddof, keepdims, where])	Compute the standard deviation along the specified axis
var(a[, axis, dtype, out, ddof, keepdims, where])	Compute the variance along the specified axis
nanmedian(a[, axis, out, overwrite_input, ...])	Compute the median along the specified axis, while ignoring NaNs
nanmean(a[, axis, dtype, out, keepdims, where])	Compute the arithmetic mean along the specified axis, ignoring NaNs
nanstd(a[, axis, dtype, out, ddof, ...])	Compute the standard deviation along the specified axis, while ignoring NaNs
nanvar(a[, axis, dtype, out, ddof, ...])	Compute the variance along the specified axis, while ignoring NaNs

Correlating:

Function	Description
corrcoef(x[, y, rowvar, bias, ddof, dtype])	Return Pearson product-moment correlation coefficients
correlate(a, v[, mode])	Cross-correlation of two 1-dimensional sequences
cov(m[, y,rowvar,bias,ddof,fweights,...])	Estimate a covariance matrix, given data and weights

Matrix Transpose

With the help of Numpy `matrix.transpose()` method, we can find the transpose of the matrix by using the `matrix.transpose()` method.

In [58]:

```
c = np.array([[1, 2, 3],
              [4, 5, 6],
              [7,8,9]])
print(c)
print("Transpose matrix is:")
print(c.transpose())
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
Transpose matrix is:
[[1 4 7]
 [2 5 8]
 [3 6 9]]
```

We can also find the transpose of the matrix by using the `matrix.T` method.

In [60]:

```
c = np.array([[1, 2, 3],
              [4, 5, 6],
              [7,8,9]])
print(c)
print("Transpose matrix is:")
print(c.T)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
Transpose matrix is:
[[1 4 7]
 [2 5 8]
 [3 6 9]]
```

Matrix Inverse

`numpy.matrix.I` returns the (multiplicative) inverse of invertible self.

NOTE: First we have to convert the ndarray into matrix using `np.matrix(ndarray)` . `matrix.getI()` returns the inverse of the given matrix.

In [15]:

```
c = np.array([[1, 2],
              [3,4]])
print(c)
print("Inverse matrix is:")
m=np.matrix(c)
print(m.getI())
```

```
[[1 2]
 [3 4]]
Transpose matrix is:
[[-2.   1. ]
 [ 1.5 -0.5]]
```

Lab Programs:-

9. write a program to calculate the sum of every column in a numpy array.

In [1]:

```
import numpy as np
l1 = []
arr = np.array([[1,2,3],[4,5,6],[7,8,9]])
for i in range(0,3):
    l1.append(sum(arr[0:,i]))
print(l1)
```

```
[12, 15, 18]
```

10. write a python program to claculate the sum of every row in a numpy array.

In [2]:

```
import numpy as np
l1 = []
arr = np.array([[1,2,3],[4,5,6],[7,8,9]])
for i in range(0,3):
    l1.append(sum(arr[i,0:]))
print(l1)
```

```
[6, 15, 24]
```

11. write a numpy program to compute the 80 percentile for all elements in a given array along the second axis.

In [4]:

```
import numpy as np
x = np.arange(12).reshape((2, 6))
print("\nOriginal array:")
print(x)
r1 = np.percentile(x, 80, 1)
print("\n80th percentile for all elements of the said array along the second axis:")
print(r1)
```

```
Original array:
[[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]]

80th percentile for all elements of the said array along the second axis:
[ 4. 10.]
```

12. write a numpy program to compute the median of flattned given array.

In [5]:

```
import numpy as np
s = np.array([[0,1,2,3,4,5],[6,7,8,9,10,11]])
x = s.flatten()
print(x)
np.median(x)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11]
```

Out[5]:

```
5.5
```

13. write a python program to compute the weighted of the given array.

In [6]:

```
import numpy as np
x = np.arange(0,5)
weight = np.arange(1,6)
wa = np.average(x, weights = weight)
print(wa)
```

```
2.6666666666666665
```

14. Write a Numpy program to compute the covarience matrix of two given array.

In [7]:

```
import numpy as np
x = list(int(num) for num in input("Enter values of x:").split())
y = list(int(num) for num in input("Enter values of y:").split())
print("x:",x)
print("y:",y)
print("Covariance matrix of two given arrays:",np.cov(x,y))
```

```
Enter values of x:1 2 3 4 5
Enter values of y:6 7 8 9 10
x: [1, 2, 3, 4, 5]
y: [6, 7, 8, 9, 10]
Covariance matrix of two given arrays: [[2.5 2.5]
 [2.5 2.5]]
```

15. write a NumPy program to compute cross-corelation of two given arrays.

In [8]:

```
import numpy as np
x = list(int(num) for num in input("Enter values of x:").split())
y = list(int(num) for num in input("Enter values of y:").split())
print("x:",x)
print("y:",y)
print("Cross-correlation matrix of two given arrays:",np.correlate(x,y))
```

```
Enter values of x:1 2 3 4 5
Enter values of y:6 7 8 9 10
x: [1, 2, 3, 4, 5]
y: [6, 7, 8, 9, 10]
Cross-correlation matrix of two given arrays: [130]
```

16. write a python program to compute the weighted average along the specified axis of the given flattened array

In [9]:

```
import numpy as np
y = np.arange(0,9).reshape(3,3)
weight = np.arange(1,4)
wa = np.average(y,axis = 1,weights = weight)
print(wa)
```

```
[1.33333333 4.33333333 7.33333333]
```

In [ ]:

```

```