

Prepared by Asif Bhat

## Numpy Tutorial

```
In [187]: # Import Numpy Library  
import numpy as np  
import warnings  
warnings.filterwarnings("ignore")  
from IPython.display import Image
```



### Numpy Array Creation

```
In [188]: list1 = [10,20,30,40,50,60]  
list1
```

Out[188]: [10, 20, 30, 40, 50, 60]

```
In [189]: # Display the type of an object  
type(list1)
```

Out[189]: list

```
In [190]: #Convert List to Numpy Array  
arr1 = np.array(list1)  
arr1
```

Out[190]: array([10, 20, 30, 40, 50, 60])

```
In [191]: #Memory address of an array object  
arr1.data
```

Out[191]: <memory at 0x000001C2B747E348>

```
In [192]: # Display type of an object  
type(arr1)
```

Out[192]: numpy.ndarray

```
In [193]: #Datatype of array  
arr1.dtype
```

Out[193]: dtype('int32')



```
In [194]: # Convert Integer Array to FLOAT  
arr1.astype(float)
```

```
Out[194]: array([10., 20., 30., 40., 50., 60.])
```

```
In [195]: # Generate evenly spaced numbers (space =1) between 0 to 10  
np.arange(0,10)
```

```
Out[195]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In [196]: # Generate numbers between 0 to 100 with a space of 10  
np.arange(0,100,10)
```

```
Out[196]: array([ 0, 10, 20, 30, 40, 50, 60, 70, 80, 90])
```

```
In [197]: # Generate numbers between 10 to 100 with a space of 10 in descending order  
np.arange(100, 10, -10)
```

```
Out[197]: array([100, 90, 80, 70, 60, 50, 40, 30, 20])
```

```
In [198]: #Shape of Array  
arr3 = np.arange(0,10)  
arr3.shape
```

```
Out[198]: (10,)
```

```
In [199]: arr3
```

```
Out[199]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In [200]: # Size of array  
arr3.size
```

```
Out[200]: 10
```

```
In [201]: # Dimension  
arr3.ndim
```

```
Out[201]: 1
```

```
In [202]: # Datatype of object  
arr3.dtype
```

```
Out[202]: dtype('int32')
```

```
In [203]: # Bytes consumed by one element of an array object  
arr3.itemsize
```

```
Out[203]: 4
```

```
In [204]: # Bytes consumed by an array object  
arr3.nbytes
```

```
Out[204]: 40
```

```
In [205]: # Length of array  
len(arr3)
```

```
Out[205]: 10
```

```
In [206]: # Generate an array of zeros  
np.zeros(10)
```

```
Out[206]: array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

```
In [207]: # Generate an array of ones with given shape  
np.ones(10)
```

```
Out[207]: array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

```
In [208]: # Repeat 10 five times in an array  
np.repeat(10,5)
```

```
Out[208]: array([10, 10, 10, 10, 10])
```

```
In [209]: # Repeat each element in array 'a' thrice  
a= np.array([10,20,30])  
np.repeat(a,3)
```

```
Out[209]: array([10, 10, 10, 20, 20, 20, 30, 30, 30])
```

```
In [210]: # Array of 10's  
np.full(5,10)
```

```
Out[210]: array([10, 10, 10, 10, 10])
```

```
In [211]: # Generate array of Odd numbers  
ar1 = np.arange(1,20)  
ar1[ar1%2 ==1]
```

```
Out[211]: array([ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19])
```

```
In [212]: # Generate array of even numbers  
ar1 = np.arange(1,20)  
ar1[ar1%2 == 0]
```

```
Out[212]: array([ 2,  4,  6,  8, 10, 12, 14, 16, 18])
```

```
In [213]: # Generate evenly spaced 4 numbers between 10 to 20.  
np.linspace(10,20,4)
```

```
Out[213]: array([10.          , 13.33333333, 16.66666667, 20.          ])
```

```
In [214]: # Generate evenly spaced 11 numbers between 10 to 20.  
np.linspace(10,20,11)
```

```
Out[214]: array([10., 11., 12., 13., 14., 15., 16., 17., 18., 19., 20.])
```

```
In [215]: # Create an array of random values  
np.random.random(4)
```

```
Out[215]: array([0.61387161, 0.7734601 , 0.48868515, 0.05535259])
```

```
In [216]: # Generate an array of Random Integer numbers  
np.random.randint(0,500,5)
```

```
Out[216]: array([359,    3, 200, 437, 400])
```

```
In [217]: # Generate an array of Random Integer numbers  
np.random.randint(0,500,10)
```

```
Out[217]: array([402, 196, 481, 426, 245,   19, 292, 233, 399, 175])
```

```
In [218]: # Using random.seed we can generate same number of Random numbers  
np.random.seed(123)  
np.random.randint(0,100,10)
```

```
Out[218]: array([66, 92, 98, 17, 83, 57, 86, 97, 96, 47])
```

```
In [219]: # Using random.seed we can generate same number of Random numbers  
np.random.seed(123)  
np.random.randint(0,100,10)
```

```
Out[219]: array([66, 92, 98, 17, 83, 57, 86, 97, 96, 47])
```

```
In [220]: # Using random.seed we can generate same number of Random numbers  
np.random.seed(101)  
np.random.randint(0,100,10)
```

```
Out[220]: array([95, 11, 81, 70, 63, 87, 75,   9, 77, 40])
```

```
In [221]: # Using random.seed we can generate same number of Random numbers  
np.random.seed(101)  
np.random.randint(0,100,10)
```

```
Out[221]: array([95, 11, 81, 70, 63, 87, 75,   9, 77, 40])
```

```
In [222]: # Generate array of Random float numbers  
f1 = np.random.uniform(5,10, size=(10))  
f1
```

```
Out[222]: array([6.5348311 , 9.4680654 , 8.60771931, 5.94969477, 7.77113796,  
                6.76065977, 5.90946201, 8.92800881, 9.82741611, 6.16176831])
```

```
In [223]: # Extract Integer part  
np.floor(f1)
```

```
Out[223]: array([6., 9., 8., 5., 7., 6., 5., 8., 9., 6.])
```

```
In [224]: # Truncate decimal part  
np.trunc(f1)
```

```
Out[224]: array([6., 9., 8., 5., 7., 6., 5., 8., 9., 6.])
```

```
In [225]: # Convert Float Array to Integer array  
f1.astype(int)
```

```
Out[225]: array([6, 9, 8, 5, 7, 6, 5, 8, 9, 6])
```

```
In [226]: # Normal distribution (mean=0 and variance=1)  
b2 = np.random.randn(10)  
b2
```

```
Out[226]: array([ 0.18869531, -0.75887206, -0.93323722,  0.95505651,  0.19079432,  
                1.97875732,  2.60596728,  0.68350889,  0.30266545,  1.69372293])
```

```
In [227]: arr1
```

```
Out[227]: array([10, 20, 30, 40, 50, 60])
```

```
In [228]: # Enumerate for Numpy Arrays  
for index, value in np.ndenumerate(arr1):  
    print(index, value)
```

```
(0,) 10  
(1,) 20  
(2,) 30  
(3,) 40  
(4,) 50  
(5,) 60
```



## Operations on an Array

```
In [229]: arr2 = np.arange(1,20)
          arr2
```

```
Out[229]: array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
                18, 19])
```

```
In [230]: # Sum of all elements in an array
          arr2.sum()
```

```
Out[230]: 190
```

```
In [231]: # Cumulative Sum
          np.cumsum(arr2)
```

```
Out[231]: array([ 1,  3,  6, 10, 15, 21, 28, 36, 45, 55, 66, 78, 91,
                105, 120, 136, 153, 171, 190], dtype=int32)
```

```
In [232]: # Find Minimum number in an array
          arr2.min()
```

```
Out[232]: 1
```

```
In [233]: # Find MAX number in an array
          arr2.max()
```

```
Out[233]: 19
```

```
In [234]: # Find INDEX of Minimum number in an array
          arr2.argmin()
```

```
Out[234]: 0
```

```
In [235]: # Find INDEX of MAX number in an array
          arr2.argmax()
```

```
Out[235]: 18
```

```
In [236]: # Find mean of all numbers in an array
          arr2.mean()
```

```
Out[236]: 10.0
```

```
In [237]: # Find median of all numbers present in arr2
          np.median(arr2)
```

```
Out[237]: 10.0
```

```
In [238]: # Variance
          np.var(arr2)
```

```
Out[238]: 30.0
```

```
In [239]: # Standard deviation  
np.std(arr2)
```

```
Out[239]: 5.477225575051661
```

```
In [240]: # Calculating percentiles  
np.percentile(arr2,70)
```

```
Out[240]: 13.6
```

```
In [241]: # 10th & 70th percentile  
np.percentile(arr2,[10,70])
```

```
Out[241]: array([ 2.8, 13.6])
```

## ▼ Operations on a 2D Array

```
In [242]: A = np.array([[1,2,3,0] , [5,6,7,22] , [10 , 11 , 1 ,13] , [14,15,16,3]])  
A
```

```
Out[242]: array([[ 1,  2,  3,  0],  
                [ 5,  6,  7, 22],  
                [10, 11,  1, 13],  
                [14, 15, 16,  3]])
```

```
In [243]: # SUM of all numbers in a 2D array  
A.sum()
```

```
Out[243]: 129
```

```
In [244]: # MAX number in a 2D array  
A.max()
```

```
Out[244]: 22
```

```
In [245]: # Minimum  
A.min()
```

```
Out[245]: 0
```

```
In [246]: # Column wise mimimum value  
np.amin(A, axis=0)
```

```
Out[246]: array([1, 2, 1, 0])
```

```
In [247]: # Row wise mimimum value  
np.amin(A, axis=1)
```

```
Out[247]: array([0, 5, 1, 3])
```

```
In [248]: # Mean of all numbers in a 2D array  
A.mean()
```

Out[248]: 8.0625

```
In [249]: # Mean  
np.mean(A)
```

Out[249]: 8.0625

```
In [250]: # Median  
np.median(A)
```

Out[250]: 6.5

```
In [251]: # 50 percentile = Median  
np.percentile(A,50)
```

Out[251]: 6.5

```
In [252]: np.var(A)
```

Out[252]: 40.30859375

```
In [253]: np.std(A)
```

Out[253]: 6.348904925260734

```
In [254]: np.percentile(arr2,70)
```

Out[254]: 13.6

```
In [255]: # Enumerate for Numpy 2D Arrays  
for index, value in np.ndenumerate(A):  
    print(index, value)
```

```
(0, 0) 1  
(0, 1) 2  
(0, 2) 3  
(0, 3) 0  
(1, 0) 5  
(1, 1) 6  
(1, 2) 7  
(1, 3) 22  
(2, 0) 10  
(2, 1) 11  
(2, 2) 1  
(2, 3) 13  
(3, 0) 14  
(3, 1) 15  
(3, 2) 16  
(3, 3) 3
```



## Reading elements of an array

```
In [256]: a = np.array([7,5,3,9,0,2])
```

```
In [257]: # Access first element of the array  
a[0]
```

```
Out[257]: 7
```

```
In [258]: # Access all elements of Array except first one.  
a[1:]
```

```
Out[258]: array([5, 3, 9, 0, 2])
```

```
In [259]: # Fetch 2nd , 3rd & 4th value from the Array  
a[1:4]
```

```
Out[259]: array([5, 3, 9])
```

```
In [260]: # Get last element of the array  
a[-1]
```

```
Out[260]: 2
```

```
In [261]: a[-3]
```

```
Out[261]: 9
```

```
In [262]: a[-6]
```

```
Out[262]: 7
```

```
In [263]: a[-3:-1]
```

```
Out[263]: array([9, 0])
```

## Replace elements in array

```
In [264]: ar = np.arange(1,20)  
ar
```

```
Out[264]: array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,  
                18, 19])
```

```
In [265]: # Replace EVEN numbers with ZERO
rep1 = np.where(ar % 2 == 0, 0, ar)
print(rep1)

[ 1  0  3  0  5  0  7  0  9  0 11  0 13  0 15  0 17  0 19]
```

```
In [266]: ar2 = np.array([10, 20, 30, 10, 10, 20, 20])
ar2
```

```
Out[266]: array([10, 20, 30, 10, 10, 20, 20])
```

```
In [267]: # Replace 10 with value 99
rep2 = np.where(ar2 == 10, 99, ar2)
print(rep2)

[99 20 30 99 99 20 20]
```

```
In [268]: p2 = np.arange(0,100,10)
p2
```

```
Out[268]: array([ 0, 10, 20, 30, 40, 50, 60, 70, 80, 90])
```

```
In [269]: # Replace values at INDEX Loc 0,3,5 with 33,55,99
np.put(p2, [0, 3, 5], [33, 55, 99])
p2
```

```
Out[269]: array([33, 10, 20, 55, 40, 99, 60, 70, 80, 90])
```

## ▼ Missing Values in an array

```
In [270]: a = np.array([10, np.nan, 20, 30, 60, np.nan, 90, np.inf])
a
```

```
Out[270]: array([10., nan, 20., 30., 60., nan, 90., inf])
```

```
In [271]: # Search for missing values and return as a boolean array
np.isnan(a)
```

```
Out[271]: array([False,  True, False, False, False,  True, False, False])
```

```
In [272]: # Index of missing values in an array
np.where(np.isnan(a))
```

```
Out[272]: (array([1, 5], dtype=int64),)
```

```
In [273]: # Replace all missing values with 99
a[np.isnan(a)] = 99
a
```

```
Out[273]: array([10., 99., 20., 30., 60., 99., 90., inf])
```

```
In [274]: # Check if array has any NULL value
np.isnan(a).any()
```

Out[274]: False

```
In [275]: A = np.array([[1,2,np.nan,4] , [np.nan,6,7,8] , [10 , np.nan , 12 ,13] , [14,15,
A
```

Out[275]: array([[ 1., 2., nan, 4.],  
[nan, 6., 7., 8.],  
[10., nan, 12., 13.],  
[14., 15., 16., 17.]])

```
In [276]: # Search for missing values and return as a boolean array
np.isnan(A)
```

Out[276]: array([[False, False, True, False],  
[ True, False, False, False],  
[False, True, False, False],  
[False, False, False, False]])

```
In [277]: # Index of missing values in an array
np.where(np.isnan(A))
```

Out[277]: (array([0, 1, 2], dtype=int64), array([2, 0, 1], dtype=int64))

## ▼ Stack Arrays Vertically

```
In [278]: a = np.zeros(20).reshape(2,-1)
b = np.repeat(1, 20).reshape(2,-1)
a
```

Out[278]: array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],  
[0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]])

```
In [279]: b
```

Out[279]: array([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1],  
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])

```
In [280]: np.vstack([a,b])
```

Out[280]: array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],  
[0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],  
[1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],  
[1., 1., 1., 1., 1., 1., 1., 1., 1., 1.]])

```
In [281]: a1 = np.array([[1], [2], [3]])
b1 = np.array([[4], [5], [6]])
```

In [282]: `a1`

Out[282]: `array([[1],  
[2],  
[3]])`

In [283]: `b1`

Out[283]: `array([[4],  
[5],  
[6]])`

In [287]: `np.vstack([a1,b1])`

Out[287]: `array([[1],  
[2],  
[3],  
[4],  
[5],  
[6]])`



## Stack Arrays Horizontally

In [288]: `np.hstack([a,b])`

Out[288]: `array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1.,  
1., 1., 1., 1.],  
[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1.,  
1., 1., 1., 1.]])`

In [289]: `np.hstack([a1,b1])`

Out[289]: `array([[1, 4],  
[2, 5],  
[3, 6]])`



## Common items between two Arrays

In [290]: `c1 = np.array([10,20,30,40,50,60])  
c2 = np.array([12,20,33,40,55,60])`

In [291]: `np.intersect1d(c1,c2)`

Out[291]: `array([20, 40, 60])`



## Remove Common Elements

```
In [292]: # Remove common elements of C1 & C2 array from C1  
np.setdiff1d(c1,c2)
```

```
Out[292]: array([10, 30, 50])
```



## Process Elements on Conditions

```
In [293]: a = np.array([1,2,3,6,8])  
b = np.array([10,2,30,60,8])  
  
np.where(a == b) # returns the indices of elements in an input array where the g
```

```
Out[293]: (array([1, 4], dtype=int64),)
```

```
In [294]: # Return an array where condition is satisfied  
a[np.where(a == b)]
```

```
Out[294]: array([2, 8])
```

```
In [295]: # Return all numbers between 20 & 35  
a1 = np.arange(0,60)  
a1[np.where ((a1>20) & (a1<35))]
```

```
Out[295]: array([21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34])
```

```
In [296]: # Return all numbers between 20 & 35 OR numbers divisible by 10  
a1 = np.arange(0,60)  
a1[np.where (((a1>20) & (a1<35)) | (a1 % 10 ==0)) ]
```

```
Out[296]: array([ 0, 10, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,  
                40, 50])
```

```
In [297]: # Return all numbers between 20 & 35 using np.logical_and  
a1[np.where(np.logical_and(a1>20, a1<35))]
```

```
Out[297]: array([21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34])
```



## Check for elements in an Array using isin()

```
In [300]: a = np.array([10,20,30,40,50,60,70])  
a
```

```
Out[300]: array([10, 20, 30, 40, 50, 60, 70])
```

```
In [301]: # Check whether number 11 & 20 are present in an array  
np.isin(a, [11,20])
```

```
Out[301]: array([False,  True, False, False, False, False, False])
```

```
In [521]: #Display the matching numbers  
a[np.isin(a,20)]
```

```
Out[521]: array([20])
```

```
In [522]: # Check whether number 33 is present in an array  
np.isin(a, 33)
```

```
Out[522]: array([False, False, False, False, False, False, False])
```

```
In [523]: a[np.isin(a, 33)]
```

```
Out[523]: array([], dtype=int32)
```

```
In [525]: b = np.array([10,20,30,40,10,10,70,80,70,90])  
b
```

```
Out[525]: array([10, 20, 30, 40, 10, 10, 70, 80, 70, 90])
```

```
In [526]: # Check whether number 10 & 70 are present in an array  
np.isin(b, [10,70])
```

```
Out[526]: array([ True, False, False, False,  True,  True,  True, False,  True,  
                False])
```

```
In [517]: # Display the indices where match occurred  
np.where(np.isin(b, [10,70]))
```

```
Out[517]: (array([0, 4, 5, 6, 8], dtype=int64),)
```

```
In [518]: # Display the matching values  
b[np.where(np.isin(b, [10,70]))]
```

```
Out[518]: array([10, 10, 10, 70, 70])
```

```
In [527]: # Display the matching values  
b[np.isin(b, [10,70])]
```

```
Out[527]: array([10, 10, 10, 70, 70])
```



## Reverse Array

```
In [598]: a4 = np.arange(10,30)
```

In [599]: a4

Out[599]: array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29])

In [600]: *# Reverse the array*  
a4[::-1]

Out[600]: array([29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10])

In [601]: *# Reverse the array*  
np.flip(a4)

Out[601]: array([29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10])

In [604]: a3 = np.array([[3,2,8,1] , [70,50,10,67] , [45,25,75,15] , [12,9,77,4]])  
a3

Out[604]: array([[ 3, 2, 8, 1],  
[70, 50, 10, 67],  
[45, 25, 75, 15],  
[12, 9, 77, 4]])

In [605]: *# Reverse ROW positions*  
a3[::-1,]

Out[605]: array([[12, 9, 77, 4],  
[45, 25, 75, 15],  
[70, 50, 10, 67],  
[ 3, 2, 8, 1]])

In [610]: *# Reverse COLUMN positions*  
a3[:,::-1]

Out[610]: array([[ 1, 8, 2, 3],  
[67, 10, 50, 70],  
[15, 75, 25, 45],  
[ 4, 77, 9, 12]])

In [607]: *# Reverse both ROW & COLUMN positions*  
a3[::-1,::-1]

Out[607]: array([[ 4, 77, 9, 12],  
[15, 75, 25, 45],  
[67, 10, 50, 70],  
[ 1, 8, 2, 3]])



## Sorting Array

```
In [579]: a = np.array([10,5,2,22,12,92,17,33])
```

```
In [580]: # Sort array in ascending order
np.sort(a)
```

```
Out[580]: array([ 2,  5, 10, 12, 17, 22, 33, 92])
```

```
In [581]: a3 = np.array([[3,2,8,1] , [70,50,10,67] , [45,25,75,15]])
a3
```

```
Out[581]: array([[ 3,  2,  8,  1],
                 [70, 50, 10, 67],
                 [45, 25, 75, 15]])
```

```
In [582]: # Sort along rows
np.sort(a3)
```

```
Out[582]: array([[ 1,  2,  3,  8],
                 [10, 50, 67, 70],
                 [15, 25, 45, 75]])
```

```
In [583]: # Sort along rows
np.sort(a3,axis =1)
```

```
Out[583]: array([[ 1,  2,  3,  8],
                 [10, 50, 67, 70],
                 [15, 25, 45, 75]])
```

```
In [584]: # Sort along columns
np.sort(a3,axis =0)
```

```
Out[584]: array([[ 3,  2,  8,  1],
                 [45, 25, 10, 15],
                 [70, 50, 75, 67]])
```

```
In [585]: # Sort in descending order
b = np.sort(a)
b = b[::-1]
b
```

```
Out[585]: array([92, 33, 22, 17, 12, 10,  5,  2])
```

```
In [590]: # Sort in descending order
c = np.sort(a)
np.flip(c)
```

```
Out[590]: array([92, 33, 22, 17, 12, 10,  5,  2])
```



```
In [567]: # Sort in descending order  
a[::-1].sort()  
a
```

```
Out[567]: array([92, 33, 22, 17, 12, 10,  5,  2])
```



## "N" Largest & Smallest Numbers in an Array

```
In [766]: p = np.arange(0,50)  
p
```

```
Out[766]: array([ 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, 45, 46, 47, 48, 49])
```

```
In [767]: np.random.shuffle(p)  
p
```

```
Out[767]: array([33, 48, 14, 20, 44, 29,  4, 46, 18, 45, 21,  2,  7, 30, 17, 40, 37,  
                42, 34, 25, 35, 38, 43,  8, 24, 32, 10, 36,  0, 26, 12,  9,  3, 39,  
                6, 49, 23, 13,  1,  5, 19, 27, 47, 15, 22, 11, 41, 31, 16, 28])
```

```
In [768]: # Return "n" Largest numbers in an Array  
n = 4  
p[np.argsort(p)[-n:]]
```

```
Out[768]: array([46, 47, 48, 49])
```

```
In [769]: # Return "n" Largest numbers in an Array  
p[np.argpartition(-p,n)[:n]]
```

```
Out[769]: array([48, 47, 49, 46])
```

```
In [770]: # Return "n" smallest numbers in an Array  
p[np.argsort(-p)[-n:]]
```

```
Out[770]: array([3, 2, 1, 0])
```

```
In [771]: # Return "n" smallest numbers in an Array  
p[np.argpartition(p,n)[:n]]
```

```
Out[771]: array([1, 0, 2, 3])
```



## Repeating Sequences

```
In [656]: a5 = [10,20,30]
          a5
```

```
Out[656]: [10, 20, 30]
```

```
In [657]: # Repeat whole array twice
          np.tile(a5, 2)
```

```
Out[657]: array([10, 20, 30, 10, 20, 30])
```

```
In [658]: # Repeat each element in an array thrice
          np.repeat(a5, 3)
```

```
Out[658]: array([10, 10, 10, 20, 20, 20, 30, 30, 30])
```



## Compare Arrays

```
In [697]: d1 = np.arange(0,10)
          d1
```

```
Out[697]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In [698]: d2 = np.arange(0,10)
          d2
```

```
Out[698]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In [703]: d3 = np.arange(10,20)
          d3
```

```
Out[703]: array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19])
```

```
In [707]: d4 = d1[::-1]
          d4
```

```
Out[707]: array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
```

```
In [704]: # Compare arrays using "allclose" function. If this function returns True then A
          res1 = np.allclose(d1,d2)
          res1
```

```
Out[704]: True
```

```
In [705]: # Compare arrays using "allclose" function. If this function returns False then
          res2 = np.allclose(d1,d3)
          res2
```

```
Out[705]: False
```

```
In [709]: # Compare arrays using "allclose" function.
res3 = np.allclose(d1,d4)
res3
```

Out[709]: False



## Frequent Values in an Array

```
In [782]: # unique numbers in an array
b = np.array([10,10,10,20,30,20,30,30,20,10,10,30,10])
np.unique(b)
```

Out[782]: array([10, 20, 30])

```
In [783]: # unique numbers in an array along with the count E.g value 10 occurred maximum
val , count = np.unique(b,return_counts=True)
val,count
```

Out[783]: (array([10, 20, 30]), array([6, 3, 4], dtype=int64))

```
In [784]: # 10 is the most frequent value
np.bincount(b).argmax()
```

Out[784]: 10



## Read-Only Array

```
In [710]: d5 = np.arange(10,100,10)
d5
```

Out[710]: array([10, 20, 30, 40, 50, 60, 70, 80, 90])

```
In [711]: # Make arrays immutable
d5.flags.writeable = False
```

```
In [712]: d5[0] = 99
```

```
-----
ValueError                                Traceback (most recent call last)
<ipython-input-712-cff52f234eeb> in <module>
----> 1 d5[0] = 99

ValueError: assignment destination is read-only
```

```
In [713]: d5[2] = 11
```

```
-----  
ValueError                                Traceback (most recent call last)  
<ipython-input-713-a3b6c959d563> in <module>  
----> 1 d5[2] = 11  
  
ValueError: assignment destination is read-only
```

## ▼ Load & Save

```
In [167]: # Load data from a text file using loadtxt  
p4 = np.loadtxt('sample.txt',  
                dtype = np.integer # Decides the datatype of resulting array  
                )  
p4
```

```
Out[167]: array([[24, 29, 88],  
                 [ 1,  0,  8],  
                 [33,  7, 99],  
                 [39, 11, 98],  
                 [22, 76, 87]])
```

```
In [168]: # Load data from a text file using genfromtxt  
p5 = np.genfromtxt('sample0.txt', dtype='str')  
p5
```

```
Out[168]: array([[ 'Asif', 'India', 'Cricket'],  
                 [ 'John', 'USA', 'Hockey'],  
                 [ 'Ramiro', 'Canada', 'Football']], dtype='<U8')
```

```
In [169]: # Accessing specific rows  
p5[0]
```

```
Out[169]: array([ 'Asif', 'India', 'Cricket'], dtype='<U8')
```

```
In [170]: # Accessing specific columns  
p5[:,0]
```

```
Out[170]: array([ 'Asif', 'John', 'Ramiro'], dtype='<U8')
```

```
In [171]: p6 = np.genfromtxt('sample2.txt',
                        delimiter=' ',
                        dtype=None,
                        names=('Name', 'ID', 'Age'))

p6
```

```
Out[171]: array([(b'Name', b'ID', b'Age'), (b'Asif', b'22', b'29'),
                (b'John', b'45', b'33'), (b'Ramiro', b'55', b'67'),
                (b'Michael', b'67', b'55'), (b'Klaus', b'44', b'32'),
                (b'Sajad', b'23', b'53')],
              dtype=[('Name', 'S7'), ('ID', 'S2'), ('Age', 'S3')])
```

```
In [172]: # Skip header using "skiprows" parameter
p6 = np.loadtxt('sample2.txt',
                delimiter=' ',
                dtype=[('Name', str, 50), ('ID', np.integer), ('Age', np.integer)],
                skiprows=1)

p6
```

```
Out[172]: array([(b'Asif', 22, 29), (b'John', 45, 33), (b'Ramiro', 55, 67),
                (b'Michael', 67, 55), (b'Klaus', 44, 32), (b'Sajad', 23, 53)],
              dtype=[('Name', '<U50'), ('ID', '<i4'), ('Age', '<i4')])
```

```
In [173]: # Return only first & third column using "usecols" parameter
np.loadtxt('sample.txt', delimiter=' ', usecols=(0, 2))
```

```
Out[173]: array([[24., 88.],
                [ 1.,  8.],
                [33., 99.],
                [39., 98.],
                [22., 87.]])
```

```
In [174]: # Return only three rows using "max_rows" parameter
p6 = np.loadtxt('sample2.txt',
                delimiter=' ',
                dtype=[('Name', str, 50), ('ID', np.integer), ('Age', np.integer)],
                skiprows=1,
                max_rows=3)

p6
```

```
Out[174]: array([(b'Asif', 22, 29), (b'John', 45, 33), (b'Ramiro', 55, 67)],
              dtype=[('Name', '<U50'), ('ID', '<i4'), ('Age', '<i4')])
```

```
In [175]: # Skip header using "skip_header" parameter
p6 = np.genfromtxt('sample2.txt',
                  delimiter=' ',
                  dtype=[('Name', str, 50), ('ID', np.integer), ('Age', np.float64)],
                  names=('Name', 'ID', 'Age'),
                  skip_header=1)

p6
```

```
Out[175]: array([('Asif', 22, 29.), ('John', 45, 33.), ('Ramiro', 55, 67.),
                ('Michael', 67, 55.), ('Klaus', 44, 32.), ('Sajad', 23, 53.)],
          dtype=[('Name', '<U50'), ('ID', '<i4'), ('Age', '<f8')])
```

```
In [176]: p7 = np.arange(10,200,11)
p7
```

```
Out[176]: array([ 10, 21, 32, 43, 54, 65, 76, 87, 98, 109, 120, 131, 142,
                153, 164, 175, 186, 197])
```

```
In [177]: np.savetxt('test3.csv', p7, delimiter=',')
```

```
In [178]: p8 = np.arange(0,121).reshape(11,11)
p8
```

```
Out[178]: array([[ 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, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54],
                 [55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65],
                 [66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76],
                 [77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87],
                 [88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98],
                 [99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109],
                 [110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120]])
```

```
In [179]: np.save('test4.npy', p8)
```

```
In [180]: p9 = np.load('test4.npy')
p9
```

```
Out[180]: array([[ 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, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54],
                 [55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65],
                 [66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76],
                 [77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87],
                 [88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98],
                 [99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109],
                 [110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120]])
```

```
In [181]: np.save('numpyfile', p8)
```

```
In [182]: p10 = np.load('numpyfile.npy')
p10
```

```
Out[182]: array([[ 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, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54],
 [55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65],
 [66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76],
 [77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87],
 [88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98],
 [99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109],
 [110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120]])
```

```
In [183]: p11 = np.arange(0,1000000).reshape(1000,1000)
p11
```



```
Out[183]: array([[ 0,  1,  2, ..., 997, 998, 999],
 [1000, 1001, 1002, ..., 1997, 1998, 1999],
 [2000, 2001, 2002, ..., 2997, 2998, 2999],
 ...,
 [997000, 997001, 997002, ..., 997997, 997998, 997999],
 [998000, 998001, 998002, ..., 998997, 998998, 998999],
 [999000, 999001, 999002, ..., 999997, 999998, 999999]])
```

```
In [184]: # Save Numpy array to a compressed file
np.savez_compressed('test6.npz', p11)
```

```
In [185]: # Save Numpy array to a npy file
np.save('test7.npy', p11)
```

```
In [186]: # Compressed file size is much lesser than normal npy file
Image(filename='load_save.PNG')
```

```
Out[186]:
```

 test6.npz	22-07-2020 16:02	NPZ File	1,351 KB
 test7.npy	22-07-2020 16:02	NPY File	3,907 KB



## Printing Options

```
In [388]: # Display values upto 4 decimal place
np.set_printoptions(precision=4)
a = np.array([12.654398765 , 90.7864098354674])
a
```

```
Out[388]: array([12.6544, 90.7864])
```

```
In [389]: # Display values upto 2 decimal place  
np.set_printoptions(precision=2)  
a = np.array([12.654398765 , 90.7864098354674])  
a
```

```
Out[389]: array([12.65, 90.79])
```

```
In [400]: # Array Summarization  
np.set_printoptions(threshold=3)  
np.arange(200)
```

```
Out[400]: array([ 0,  1,  2, ..., 197, 198, 199])
```

```
In [404]: # Reset Formatter  
np.set_printoptions(precision=8, suppress=False, threshold=1000, formatter=None)  
a = np.array([12.654398765 , 90.7864098354674])  
a
```

```
Out[404]: array([12.65439876, 90.78640984])
```

```
In [728]: np.arange(1,1100)
```

```
Out[728]: array([  1,   2,   3, ..., 1097, 1098, 1099])
```



```
In [733]: # Display all values
np.set_printoptions(threshold=np.inf)
np.arange(1,1100)
```

```
Out[733]: array([ 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,
 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55,
 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66,
 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77,
 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88,
 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99,
100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110,
111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121,
122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132,
133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143,
144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154,
155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165,
166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176,
177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187,
188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198,
199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209,
210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220,
221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231,
232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242,
243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253,
254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264,
265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275,
276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286,
287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297,
298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308,
309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319,
320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330,
331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341,
342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352,
353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363,
364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374,
375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385,
386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396,
397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407,
408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418,
419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429,
430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440,
441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451,
452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462,
463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473,
474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 484,
485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495,
496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506,
507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517,
518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528,
529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539,
540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550,
551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561,
562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572,
```

```
573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583,
584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594,
595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605,
606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616,
617, 618, 619, 620, 621, 622, 623, 624, 625, 626, 627,
628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638,
639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649,
650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660,
661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671,
672, 673, 674, 675, 676, 677, 678, 679, 680, 681, 682,
683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 693,
694, 695, 696, 697, 698, 699, 700, 701, 702, 703, 704,
705, 706, 707, 708, 709, 710, 711, 712, 713, 714, 715,
716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726,
727, 728, 729, 730, 731, 732, 733, 734, 735, 736, 737,
738, 739, 740, 741, 742, 743, 744, 745, 746, 747, 748,
749, 750, 751, 752, 753, 754, 755, 756, 757, 758, 759,
760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770,
771, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781,
782, 783, 784, 785, 786, 787, 788, 789, 790, 791, 792,
793, 794, 795, 796, 797, 798, 799, 800, 801, 802, 803,
804, 805, 806, 807, 808, 809, 810, 811, 812, 813, 814,
815, 816, 817, 818, 819, 820, 821, 822, 823, 824, 825,
826, 827, 828, 829, 830, 831, 832, 833, 834, 835, 836,
837, 838, 839, 840, 841, 842, 843, 844, 845, 846, 847,
848, 849, 850, 851, 852, 853, 854, 855, 856, 857, 858,
859, 860, 861, 862, 863, 864, 865, 866, 867, 868, 869,
870, 871, 872, 873, 874, 875, 876, 877, 878, 879, 880,
881, 882, 883, 884, 885, 886, 887, 888, 889, 890, 891,
892, 893, 894, 895, 896, 897, 898, 899, 900, 901, 902,
903, 904, 905, 906, 907, 908, 909, 910, 911, 912, 913,
914, 915, 916, 917, 918, 919, 920, 921, 922, 923, 924,
925, 926, 927, 928, 929, 930, 931, 932, 933, 934, 935,
936, 937, 938, 939, 940, 941, 942, 943, 944, 945, 946,
947, 948, 949, 950, 951, 952, 953, 954, 955, 956, 957,
958, 959, 960, 961, 962, 963, 964, 965, 966, 967, 968,
969, 970, 971, 972, 973, 974, 975, 976, 977, 978, 979,
980, 981, 982, 983, 984, 985, 986, 987, 988, 989, 990,
991, 992, 993, 994, 995, 996, 997, 998, 999, 1000, 1001,
1002, 1003, 1004, 1005, 1006, 1007, 1008, 1009, 1010, 1011, 1012,
1013, 1014, 1015, 1016, 1017, 1018, 1019, 1020, 1021, 1022, 1023,
1024, 1025, 1026, 1027, 1028, 1029, 1030, 1031, 1032, 1033, 1034,
1035, 1036, 1037, 1038, 1039, 1040, 1041, 1042, 1043, 1044, 1045,
1046, 1047, 1048, 1049, 1050, 1051, 1052, 1053, 1054, 1055, 1056,
1057, 1058, 1059, 1060, 1061, 1062, 1063, 1064, 1065, 1066, 1067,
1068, 1069, 1070, 1071, 1072, 1073, 1074, 1075, 1076, 1077, 1078,
1079, 1080, 1081, 1082, 1083, 1084, 1085, 1086, 1087, 1088, 1089,
1090, 1091, 1092, 1093, 1094, 1095, 1096, 1097, 1098, 1099])
```



## Vector Addition

```
In [187]: v1 = np.array([1,2])  
v2 = np.array([3,4])  
v3 = v1+v2  
v3 = np.add(v1,v2)  
print('V3 = ',v3)
```

V3 = [4 6]



## Multiplication of vectors

```
In [188]: a1 = [5 , 6 ,8]  
a2 = [4, 7 , 9]  
print(np.multiply(a1,a2))
```

[20 42 72]



## Dot Product

[https://www.youtube.com/watch?v=WNulhXo39\\_k](https://www.youtube.com/watch?v=WNulhXo39_k) ([https://www.youtube.com/watch?v=WNulhXo39\\_k](https://www.youtube.com/watch?v=WNulhXo39_k))

<https://www.youtube.com/watch?v=LyGKycYT2v0> (<https://www.youtube.com/watch?v=LyGKycYT2v0>)

```
In [189]: a1 = np.array([1,2,3])
a2 = np.array([4,5,6])

dotp = a1@a2
print(" Dot product - ",dotp)

dotp = np.dot(a1,a2)
print(" Dot product usign np.dot",dotp)

dotp = np.inner(a1,a2)
print(" Dot product usign np.inner", dotp)

dotp = sum(np.multiply(a1,a2))
print(" Dot product usign np.multiply & sum",dotp)

dotp = np.matmul(a1,a2)
print(" Dot product usign np.matmul",dotp)

dotp = 0
for i in range(len(a1)):
    dotp = dotp + a1[i]*a2[i]
print(" Dot product usign for loop" , dotp)
```

```
Dot product - 32
Dot product usign np.dot 32
Dot product usign np.inner 32
Dot product usign np.multiply & sum 32
Dot product usign np.matmul 32
Dot product usign for loop 32
```

## ▼ Length of Vector

```
In [190]: v3 = np.array([1,2,3,4,5,6])
length = np.sqrt(np.dot(v3,v3))
length
```

Out[190]: 9.539392014169456

```
In [191]: v3 = np.array([1,2,3,4,5,6])
length = np.sqrt(sum(np.multiply(v3,v3)))
length
```

Out[191]: 9.539392014169456

```
In [193]: v3 = np.array([1,2,3,4,5,6])
length = np.sqrt(np.matmul(v3,v3))
length
```

Out[193]: 9.539392014169456

## ▼ Normalized Vector

How to normalize a vector : <https://www.youtube.com/watch?v=7fn03DIW3Ak>  
<https://www.youtube.com/watch?v=7fn03DIW3Ak>

```
In [194]: #First Method
v1 = [2,3]
length_v1 = np.sqrt(np.dot(v1,v1))
norm_v1 = v1/length_v1
length_v1 , norm_v1
```

Out[194]: (3.605551275463989, array([0.5547002 , 0.83205029]))

```
In [199]: #Second Method
v1 = [2,3]
norm_v1 = v1/np.linalg.norm(v1)
norm_v1
```

Out[199]: array([0.5547002 , 0.83205029])

## ▼ Angle between vectors

```
In [200]: #First Method
v1 = np.array([8,4])
v2 = np.array([-4,8])
ang = np.rad2deg(np.arccos( np.dot(v1,v2) / (np.linalg.norm(v1)*np.linalg.norm(v2)) ))
ang
```

Out[200]: 90.0

```
In [201]: #Second Method
v1 = np.array([4,3])
v2 = np.array([-3,4])
lengthV1 = np.sqrt(np.dot(v1,v1))
lengthV2 = np.sqrt(np.dot(v2,v2))
ang = np.rad2deg(np.arccos( np.dot(v1,v2) / (lengthV1 * lengthV2)))
print('Angle between Vectors - %s' %ang)
```

Angle between Vectors - 90.0

## ▼ Inner & outer products

Inner and Outer Product :

<https://www.youtube.com/watch?v=FCmH4MqbFGs&t=2s> (<https://www.youtube.com/watch?v=FCmH4MqbFGs&t=2s>)

<https://www.youtube.com/watch?v=FCmH4MqbFGs> (<https://www.youtube.com/watch?v=FCmH4MqbFGs>)

```
In [203]: v1 = np.array([1,2,3])
v2 = np.array([4,5,6])
np.inner(v1,v2)

print("\n Inner Product ==> \n", np.inner(v1,v2))
print("\n Outer Product ==> \n", np.outer(v1,v2))
```

```
Inner Product ==>
32
```

```
Outer Product ==>
[[ 4  5  6]
 [ 8 10 12]
 [12 15 18]]
```

## ▼ Vector Cross Product

```
In [204]: v1 = np.array([1,2,3])
v2 = np.array([4,5,6])
print("\nVector Cross Product ==> \n", np.cross(v1,v2))
```

```
Vector Cross Product ==>
[-3  6 -3]
```

## ▼ Matrix Creation

```
In [644]: # Create a 4x4 matrix
A = np.array([[1,2,3,4] , [5,6,7,8] , [10 , 11 , 12 ,13] , [14,15,16,17]])
A
```

```
Out[644]: array([[ 1,  2,  3,  4],
 [ 5,  6,  7,  8],
 [10, 11, 12, 13],
 [14, 15, 16, 17]])
```

```
In [125]: # Datatype of Matrix
A.dtype
```

```
Out[125]: dtype('int32')
```

```
In [126]: B = np.array([[1.5,2.07,3,4] , [5,6,7,8] , [10 , 11 , 12 ,13] , [14,15,16,17]])
B
```

```
Out[126]: array([[ 1.5 ,  2.07,  3.  ,  4.  ],
 [ 5.  ,  6.  ,  7.  ,  8.  ],
 [10.  , 11.  , 12.  , 13.  ],
 [14.  , 15.  , 16.  , 17.  ]])
```

```
In [127]: # Datatype of Matrix  
B.dtype
```

```
Out[127]: dtype('float64')
```

```
In [121]: # Shape of Matrix  
A.shape
```

```
Out[121]: (4, 4)
```

```
In [133]: # Generate a 4x4 zero matrix  
np.zeros((4,4))
```

```
Out[133]: array([[0., 0., 0., 0.],  
                [0., 0., 0., 0.],  
                [0., 0., 0., 0.],  
                [0., 0., 0., 0.]])
```

```
In [134]: #Shape of Matrix  
z1 = np.zeros((4,4))  
z1.shape
```

```
Out[134]: (4, 4)
```

```
In [11]: # Generate a 5x5 matrix filled with ones  
np.ones((5,5))
```

```
Out[11]: array([[1., 1., 1., 1., 1.],  
                [1., 1., 1., 1., 1.],  
                [1., 1., 1., 1., 1.],  
                [1., 1., 1., 1., 1.],  
                [1., 1., 1., 1., 1.]])
```

```
In [55]: # Return 10x10 matrix of random integer numbers between 0 to 500  
np.random.randint(0,500, (10,10))
```

```
Out[55]: array([[229, 366, 71, 357, 452, 244, 407, 163, 207, 226],  
                [451, 338, 441, 461, 46, 131, 46, 485, 285, 470],  
                [149, 378, 21, 465, 23, 235, 254, 383, 94, 356],  
                [199, 276, 27, 459, 5, 305, 470, 217, 191, 82],  
                [77, 358, 131, 184, 383, 142, 383, 49, 343, 52],  
                [253, 397, 431, 433, 280, 404, 448, 180, 316, 303],  
                [370, 285, 316, 309, 395, 40, 219, 301, 97, 408],  
                [292, 166, 137, 125, 52, 67, 299, 129, 79, 68],  
                [196, 484, 61, 146, 307, 270, 412, 401, 87, 46],  
                [52, 144, 454, 455, 84, 10, 190, 362, 96, 122]])
```

```
In [137]: arr2
```

```
Out[137]: array([644, 575, 936, 757, 316, 732, 704, 110, 5, 908, 477, 40, 49,  
                851, 623, 506, 136, 371, 925, 883])
```

```
In [90]: arr2.reshape(5,4)
```

```
Out[90]: array([[644, 575, 936, 757],
                [316, 732, 704, 110],
                [  5, 908, 477,  40],
                [ 49, 851, 623, 506],
                [136, 371, 925, 883]])
```

```
In [91]: mat1 = np.random.randint(0,1000,100).reshape(10,10)
         mat1
```

```
Out[91]: array([[ 92, 907, 507, 394, 625, 478, 419, 540,   3, 851],
                [340, 303, 526, 250, 709, 505, 956, 197, 632, 947],
                [262, 984, 103, 229, 366,  71, 357, 964, 244, 919],
                [675, 207, 226, 451, 850, 953, 461,  46, 643, 558],
                [508, 997, 797, 470, 149, 378,  21, 465, 535, 235],
                [254, 383,  94, 356, 711, 788, 539, 971,   5, 305],
                [982, 217, 703,  82, 589, 358, 643, 696, 895, 654],
                [383, 561, 855,  52, 253, 397, 943, 945, 280, 404],
                [960, 692, 828, 815, 370, 285, 828, 309, 395,  40],
                [219, 813, 609, 920, 804, 678, 649, 125, 564,  67]])
```

```
In [69]: mat1[0,0]
```

```
Out[69]: 644
```

```
In [70]: mat1[mat1 > 500]
```

```
Out[70]: array([644, 575, 936, 757, 732, 704, 908, 851, 623, 506, 925, 883, 556,
                840, 638, 906, 735, 619, 896, 503, 574, 676, 979, 831, 519, 906,
                615, 750, 503, 615, 911, 512, 628, 760, 865, 989, 664, 676, 892,
                703, 542, 956, 615, 923, 776, 854, 794, 855, 686, 950, 741, 685,
                570])
```

```
In [206]: # Identity Matrix : https://en.wikipedia.org/wiki/Identity\_matrix
```

```
I = np.eye(9)
I
```

```
Out[206]: array([[1., 0., 0., 0., 0., 0., 0., 0., 0.],
                [0., 1., 0., 0., 0., 0., 0., 0., 0.],
                [0., 0., 1., 0., 0., 0., 0., 0., 0.],
                [0., 0., 0., 1., 0., 0., 0., 0., 0.],
                [0., 0., 0., 0., 1., 0., 0., 0., 0.],
                [0., 0., 0., 0., 0., 1., 0., 0., 0.],
                [0., 0., 0., 0., 0., 0., 1., 0., 0.],
                [0., 0., 0., 0., 0., 0., 0., 1., 0.],
                [0., 0., 0., 0., 0., 0., 0., 0., 1.]])
```



In [207]: `# Diagonal Matrix : https://en.wikipedia.org/wiki/Diagonal\_matrix`

```
D = np.diag([1,2,3,4,5,6,7,8])
D
```

Out[207]: `array([[1, 0, 0, 0, 0, 0, 0, 0],
 [0, 2, 0, 0, 0, 0, 0, 0],
 [0, 0, 3, 0, 0, 0, 0, 0],
 [0, 0, 0, 4, 0, 0, 0, 0],
 [0, 0, 0, 0, 5, 0, 0, 0],
 [0, 0, 0, 0, 0, 6, 0, 0],
 [0, 0, 0, 0, 0, 0, 7, 0],
 [0, 0, 0, 0, 0, 0, 0, 8]])`

In [208]: `# Traingular Matrices (lower & Upper triangular matrix) : https://en.wikipedia.org/wiki/Triangular\_matrix`

```
M = np.random.randn(5,5)
U = np.triu(M)
L = np.tril(M)
print("lower triangular matrix - \n" , M)
print("\n")

print("lower triangular matrix - \n" , L)
print("\n")

print("Upper triangular matrix - \n" , U)
```

lower triangular matrix -

```
[[ 0.65111795 -0.31931804 -0.84807698  0.60596535 -2.01816824]
 [ 0.74012206  0.52881349 -0.58900053  0.18869531 -0.75887206]
 [-0.93323722  0.95505651  0.19079432  1.97875732  2.60596728]
 [ 0.68350889  0.30266545  1.69372293 -1.70608593 -1.15911942]
 [-0.13484072  0.39052784  0.16690464  0.18450186  0.80770591]]
```

lower triangular matrix -

```
[[ 0.65111795  0.         0.         0.         0.         ]
 [ 0.74012206  0.52881349  0.         0.         0.         ]
 [-0.93323722  0.95505651  0.19079432  0.         0.         ]
 [ 0.68350889  0.30266545  1.69372293 -1.70608593  0.         ]
 [-0.13484072  0.39052784  0.16690464  0.18450186  0.80770591]]
```

Upper triangular matrix -

```
[[ 0.65111795 -0.31931804 -0.84807698  0.60596535 -2.01816824]
 [ 0.         0.52881349 -0.58900053  0.18869531 -0.75887206]
 [ 0.         0.         0.19079432  1.97875732  2.60596728]
 [ 0.         0.         0.         -1.70608593 -1.15911942]
 [ 0.         0.         0.         0.         0.80770591]]
```

```
In [210]: # Generate a 5X5 matrix with a given fill value of 8  
np.full((5,5) , 8)
```

```
Out[210]: array([[8, 8, 8, 8, 8],  
                [8, 8, 8, 8, 8],  
                [8, 8, 8, 8, 8],  
                [8, 8, 8, 8, 8],  
                [8, 8, 8, 8, 8]])
```

```
In [371]: # Generate 5X5 matrix of Random float numbers between 10 to 20  
np.random.uniform(10,20, size=(5,5))
```

```
Out[371]: array([[13.51434265, 17.33567613, 19.13889527, 17.00987494, 13.88531272],  
                [19.42259289, 17.36491331, 12.38464388, 18.23773728, 17.60613445],  
                [13.94709074, 12.00187917, 17.12596473, 18.45308897, 13.68646541],  
                [14.36980119, 13.56597664, 12.39737407, 16.53378141, 13.90439201],  
                [16.57783018, 13.62273355, 13.56502014, 11.952516 , 19.87312751]])
```

```
In [211]: A
```

```
Out[211]: array([[ 1,  2,  3,  4],  
                [ 5,  6,  7,  8],  
                [10, 11, 12, 13],  
                [14, 15, 16, 17]])
```

```
In [645]: # Collapse Matrix into one dimension array  
A.flatten()
```

```
Out[645]: array([ 1,  2,  3,  4,  5,  6,  7,  8, 10, 11, 12, 13, 14, 15, 16, 17])
```

```
In [646]: # Collapse Matrix into one dimension array  
A.ravel()
```

```
Out[646]: array([ 1,  2,  3,  4,  5,  6,  7,  8, 10, 11, 12, 13, 14, 15, 16, 17])
```



## Reading elements of a Matrix

```
In [153]: A
```

```
Out[153]: array([[ 1,  2,  3,  4],  
                [ 5,  6,  7,  8],  
                [10, 11, 12, 13],  
                [14, 15, 16, 17]])
```

```
In [154]: # Fetch first row of matrix  
A[0,]
```

```
Out[154]: array([1, 2, 3, 4])
```

```
In [155]: # Fetch first column of matrix  
A[:,0]
```

```
Out[155]: array([ 1,  5, 10, 14])
```

```
In [156]: # Fetch first element of the matrix  
A[0,0]
```

```
Out[156]: 1
```

```
In [157]: A[1:3 , 1:3]
```

```
Out[157]: array([[ 6,  7],  
                [11, 12]])
```



## Reverse Rows / Columns of a Matrix

```
In [380]: arr = np.arange(16).reshape(4,4)  
arr
```

```
Out[380]: array([[ 0,  1,  2,  3],  
                [ 4,  5,  6,  7],  
                [ 8,  9, 10, 11],  
                [12, 13, 14, 15]])
```

```
In [381]: # Reverse rows  
arr[::-1]
```

```
Out[381]: array([[12, 13, 14, 15],  
                [ 8,  9, 10, 11],  
                [ 4,  5,  6,  7],  
                [ 0,  1,  2,  3]])
```

```
In [382]: #Reverse Columns  
arr[:, ::-1]
```

```
Out[382]: array([[ 3,  2,  1,  0],  
                [ 7,  6,  5,  4],  
                [11, 10,  9,  8],  
                [15, 14, 13, 12]])
```



## SWAP Rows & Columns

```
In [890]: m1 = np.arange(0,16).reshape(4,4)
          m1
```

```
Out[890]: array([[ 0,  1,  2,  3],
                 [ 4,  5,  6,  7],
                 [ 8,  9, 10, 11],
                 [12, 13, 14, 15]])
```

```
In [892]: # SWAP rows 0 & 1
          m1[[0,1]] = m1[[1,0]]
          m1
```

```
Out[892]: array([[ 0,  1,  2,  3],
                 [ 4,  5,  6,  7],
                 [ 8,  9, 10, 11],
                 [12, 13, 14, 15]])
```

```
In [893]: # SWAP rows 2 & 3
          m1[[3,2]] = m1[[2,3]]
          m1
```

```
Out[893]: array([[ 0,  1,  2,  3],
                 [ 4,  5,  6,  7],
                 [12, 13, 14, 15],
                 [ 8,  9, 10, 11]])
```

```
In [895]: m2 = np.arange(0,36).reshape(6,6)
          m2
```

```
Out[895]: array([[ 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]])
```

```
In [897]: # Swap columns 0 & 1
          m2[:,[0, 1]] = m2[:,[1, 0]]
          m2
```

```
Out[897]: array([[ 6,  0,  2,  3,  4,  5],
                 [ 7,  6,  8,  9, 10, 11],
                 [13, 12, 14, 15, 16, 17],
                 [19, 18, 20, 21, 22, 23],
                 [25, 24, 26, 27, 28, 29],
                 [31, 30, 32, 33, 34, 35]])
```

```
In [898]: # Swap columns 2 & 3
m2[:,[2, 3]] = m2[:,[3, 2]]
m2
```

```
Out[898]: array([[ 6,  0,  3,  2,  4,  5],
 [ 7,  6,  9,  8, 10, 11],
 [13, 12, 15, 14, 16, 17],
 [19, 18, 21, 20, 22, 23],
 [25, 24, 27, 26, 28, 29],
 [31, 30, 33, 32, 34, 35]])
```

## ▼ Concatenate Matrices

Matrix Concatenation :

<https://docs.scipy.org/doc/numpy/reference/generated/numpy.concatenate.html>

(<https://docs.scipy.org/doc/numpy/reference/generated/numpy.concatenate.html>)

```
In [216]: A = np.array([[1,2] , [3,4] ,[5,6]])
B = np.array([[1,1] , [1,1]])
C = np.concatenate((A,B))
C
```

```
Out[216]: array([[1, 2],
 [3, 4],
 [5, 6],
 [1, 1],
 [1, 1]])
```

## ▼ Matrix Addition

Matrix Addition : [https://www.youtube.com/watch?v=ZCmVpGv6\\_1g](https://www.youtube.com/watch?v=ZCmVpGv6_1g)

([https://www.youtube.com/watch?v=ZCmVpGv6\\_1g](https://www.youtube.com/watch?v=ZCmVpGv6_1g))

```
In [217]: #####
M = np.array([[1,2,3],[4,-3,6],[7,8,0]])
N = np.array([[1,1,1],[2,2,2],[3,3,3]])

print("\n First Matrix (M) ==> \n", M)
print("\n Second Matrix (N) ==> \n", N)

C = M+N
print("\n Matrix Addition (M+N) ==> \n", C)

# OR

C = np.add(M,N,dtype = np.float64)
print("\n Matrix Addition using np.add ==> \n", C)

#####
```

First Matrix (M) ==>

```
[[ 1  2  3]
 [ 4 -3  6]
 [ 7  8  0]]
```

Second Matrix (N) ==>

```
[[1 1 1]
 [2 2 2]
 [3 3 3]]
```

Matrix Addition (M+N) ==>

```
[[ 2  3  4]
 [ 6 -1  8]
 [10 11  3]]
```

Matrix Addition using np.add ==>

```
[[ 2.  3.  4.]
 [ 6. -1.  8.]
 [10. 11.  3.]]
```



## Matrix subtraction

Matrix subtraction : [https://www.youtube.com/watch?](https://www.youtube.com/watch?v=7jb_AO_hRc8&list=PLmdFyQYShrjcoVkhCClwxNj9N4rW1-T5l&index=8)

[v=7jb\\_AO\\_hRc8&list=PLmdFyQYShrjcoVkhCClwxNj9N4rW1-T5l&index=8](https://www.youtube.com/watch?v=7jb_AO_hRc8&list=PLmdFyQYShrjcoVkhCClwxNj9N4rW1-T5l&index=8)

[\(https://www.youtube.com/watch?v=7jb\\_AO\\_hRc8&list=PLmdFyQYShrjcoVkhCClwxNj9N4rW1-T5l&index=8\)](https://www.youtube.com/watch?v=7jb_AO_hRc8&list=PLmdFyQYShrjcoVkhCClwxNj9N4rW1-T5l&index=8)

In [218]:

```
#####
M = np.array([[1,2,3],[4,-3,6],[7,8,0]])
N = np.array([[1,1,1],[2,2,2],[3,3,3]])

print("\n First Matrix (M) ==> \n", M)
print("\n Second Matrix (N) ==> \n", N)

C = M-N
print("\n Matrix Subtraction (M-N) ==> \n", C)

# OR

C = np.subtract(M,N,dtype = np.float64)
print("\n Matrix Subtraction using np.subtract ==> \n", C)

#####
```

First Matrix (M) ==>

```
[[ 1  2  3]
 [ 4 -3  6]
 [ 7  8  0]]
```

Second Matrix (N) ==>

```
[[1 1 1]
 [2 2 2]
 [3 3 3]]
```

Matrix Subtraction (M-N) ==>

```
[[ 0  1  2]
 [ 2 -5  4]
 [ 4  5 -3]]
```

Matrix Subtraction using np.subtract ==>

```
[[ 0.  1.  2.]
 [ 2. -5.  4.]
 [ 4.  5. -3.]]
```



## Matrices Scalar Multiplication

Matrices Scalar Multiplication : <https://www.youtube.com/watch?v=4IHytQH1iS8&list=PLmdFyQYShrjcoVkhCClwxNj9N4rW1-T5l&index=9>  
<https://www.youtube.com/watch?v=4IHytQH1iS8&list=PLmdFyQYShrjcoVkhCClwxNj9N4rW1-T5l&index=9>

```
In [219]: M = np.array([[1,2,3],[4,-3,6],[7,8,0]])  
  
C = 10  
  
print("\n Matrix (M) ==> \n", M)  
  
print("\nMatrices Scalar Multiplication ==> \n", C*M)  
  
# OR  
  
print("\nMatrices Scalar Multiplication ==> \n", np.multiply(C,M))
```

```
Matrix (M) ==>  
[[ 1  2  3]  
 [ 4 -3  6]  
 [ 7  8  0]]
```

```
Matrices Scalar Multiplication ==>  
[[ 10  20  30]  
 [ 40 -30  60]  
 [ 70  80   0]]
```

```
Matrices Scalar Multiplication ==>  
[[ 10  20  30]  
 [ 40 -30  60]  
 [ 70  80   0]]
```



## Transpose of a matrix

Transpose of a matrix : [https://www.youtube.com/watch?v=g\\_Rz94DXvNo&list=PLmdFyQYShrjcoVkhCClwxNj9N4rW1-T5I&index=13](https://www.youtube.com/watch?v=g_Rz94DXvNo&list=PLmdFyQYShrjcoVkhCClwxNj9N4rW1-T5I&index=13)  
([https://www.youtube.com/watch?v=g\\_Rz94DXvNo&list=PLmdFyQYShrjcoVkhCClwxNj9N4rW1-T5I&index=13](https://www.youtube.com/watch?v=g_Rz94DXvNo&list=PLmdFyQYShrjcoVkhCClwxNj9N4rW1-T5I&index=13))



```
In [220]: M = np.array([[1,2,3],[4,-3,6],[7,8,0]])

print("\n Matrix (M) ==> \n", M)

print("\nTranspose of M ==> \n", np.transpose(M))

# OR

print("\nTranspose of M ==> \n", M.T)
```

```
Matrix (M) ==>
[[ 1  2  3]
 [ 4 -3  6]
 [ 7  8  0]]
```

```
Transpose of M ==>
[[ 1  4  7]
 [ 2 -3  8]
 [ 3  6  0]]
```

```
Transpose of M ==>
[[ 1  4  7]
 [ 2 -3  8]
 [ 3  6  0]]
```

## ▼ Determinant of a matrix

Determinant of a matrix :

<https://www.youtube.com/watch?v=21LWuY8i6Hw&t=88s> (<https://www.youtube.com/watch?v=21LWuY8i6Hw&t=88s>)

[https://www.youtube.com/watch?v=lp3X9LOh2dk&list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE\\_ab&index=6](https://www.youtube.com/watch?v=lp3X9LOh2dk&list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab&index=6)  
([https://www.youtube.com/watch?v=lp3X9LOh2dk&list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE\\_ab&index=6](https://www.youtube.com/watch?v=lp3X9LOh2dk&list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab&index=6))

```
In [222]: M = np.array([[1,2,3],[4,-3,6],[7,8,0]])

print("\n Matrix (M) ==> \n", M)

print("\nDeterminant of M ==> ", np.linalg.det(M))
```

```
Matrix (M) ==>
[[ 1  2  3]
 [ 4 -3  6]
 [ 7  8  0]]
```

```
Determinant of M ==> 195.0
```

## Rank of a matrix

```
In [224]: M = np.array([[1,2,3],[4,-3,6],[7,8,0]])

print("\n Matrix (M) ==> \n", M)

print("\nRank of M ==> ", np.linalg.matrix_rank(M))
```

```
Matrix (M) ==>
[[ 1  2  3]
 [ 4 -3  6]
 [ 7  8  0]]
```

Rank of M ==> 3

## Trace of matrix

```
In [225]: M = np.array([[1,2,3],[4,-3,6],[7,8,0]])

print("\n Matrix (M) ==> \n", M)

print("\nTrace of M ==> ", np.trace(M))
```

```
Matrix (M) ==>
[[ 1  2  3]
 [ 4 -3  6]
 [ 7  8  0]]
```

Trace of M ==> -2

## Inverse of matrix A

Inverse of matrix : <https://www.youtube.com/watch?v=pKZyszzmyeQ>  
(<https://www.youtube.com/watch?v=pKZyszzmyeQ>)

```
In [226]: M = np.array([[1,2,3],[4,-3,6],[7,8,0]])

print("\n Matrix (M) ==> \n", M)

print("\nInverse of M ==> \n", np.linalg.inv(M))
```

Matrix (M) ==>

```
[[ 1  2  3]
 [ 4 -3  6]
 [ 7  8  0]]
```

Inverse of M ==>

```
[[-0.24615385  0.12307692  0.10769231]
 [ 0.21538462 -0.10769231  0.03076923]
 [ 0.27179487  0.03076923 -0.05641026]]
```



## Matrix Multiplication (pointwise multiplication)

```
In [227]: M = np.array([[1,2,3],[4,-3,6],[7,8,0]])
N = np.array([[1,1,1],[2,2,2],[3,3,3]])

print("\n First Matrix (M) ==> \n", M)
print("\n Second Matrix (N) ==> \n", N)

print("\n Point-Wise Multiplication of M & N ==> \n", M*N)

# OR

print("\n Point-Wise Multiplication of M & N ==> \n", np.multiply(M,N))
```

First Matrix (M) ==>

```
[[ 1  2  3]
 [ 4 -3  6]
 [ 7  8  0]]
```

Second Matrix (N) ==>

```
[[1 1 1]
 [2 2 2]
 [3 3 3]]
```

Point-Wise Multiplication of M & N ==>

```
[[ 1  2  3]
 [ 8 -6 12]
 [21 24  0]]
```

Point-Wise Multiplication of M & N ==>

```
[[ 1  2  3]
 [ 8 -6 12]
 [21 24  0]]
```

# Matrix dot product

Matrix Multiplication :

<https://www.youtube.com/watch?v=vzt9c7iWPxs&t=207s> (<https://www.youtube.com/watch?v=vzt9c7iWPxs&t=207s>)

[https://www.youtube.com/watch?v=XkY2DOUCWMU&list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE\\_ab&index=4](https://www.youtube.com/watch?v=XkY2DOUCWMU&list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab&index=4)  
([https://www.youtube.com/watch?v=XkY2DOUCWMU&list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE\\_ab&index=4](https://www.youtube.com/watch?v=XkY2DOUCWMU&list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab&index=4))

```
In [228]: M = np.array([[1,2,3],[4,-3,6],[7,8,0]])
          N = np.array([[1,1,1],[2,2,2],[3,3,3]])

          print("\n First Matrix (M) ==> \n", M)
          print("\n Second Matrix (N) ==> \n", N)

          print("\n Matrix Dot Product ==> \n", M@N)

          # OR

          print("\n Matrix Dot Product using np.matmul ==> \n", np.matmul(M,N))

          # OR

          print("\n Matrix Dot Product using np.dot ==> \n", np.dot(M,N))
```

```
First Matrix (M) ==>
[[ 1  2  3]
 [ 4 -3  6]
 [ 7  8  0]]
```

```
Second Matrix (N) ==>
[[1 1 1]
 [2 2 2]
 [3 3 3]]
```

```
Matrix Dot Product ==>
[[14 14 14]
 [16 16 16]
 [23 23 23]]
```

```
Matrix Dot Product using np.matmul ==>
[[14 14 14]
 [16 16 16]
 [23 23 23]]
```

```
Matrix Dot Product using np.dot ==>
[[14 14 14]
 [16 16 16]
 [23 23 23]]
```

## Matrix Division

```
In [229]: M = np.array([[1,2,3],[4,-3,6],[7,8,0]])
N = np.array([[1,1,1],[2,2,2],[3,3,3]])

print("\n First Matrix (M) ==> \n", M)
print("\n Second Matrix (N) ==> \n", N)

print("\n Matrix Division (M/N) ==> \n", M/N)

# OR

print("\n Matrix Division (M/N) ==> \n", np.divide(M,N))
```

```
First Matrix (M) ==>
[[ 1  2  3]
 [ 4 -3  6]
 [ 7  8  0]]
```

```
Second Matrix (N) ==>
[[1 1 1]
 [2 2 2]
 [3 3 3]]
```

```
Matrix Division (M/N) ==>
[[ 1.         2.         3.         ]
 [ 2.        -1.5        3.         ]
 [ 2.33333333  2.66666667  0.         ]]
```

```
Matrix Division (M/N) ==>
[[ 1.         2.         3.         ]
 [ 2.        -1.5        3.         ]
 [ 2.33333333  2.66666667  0.         ]]
```

## Sum of all elements in a matrix

```
In [230]: N = np.array([[1,1,1],[2,2,2],[3,3,3]])

print("\n Matrix (N) ==> \n", N)

print ("Sum of all elements in a Matrix ==>")
print (np.sum(N))
```

```
Matrix (N) ==>
[[1 1 1]
 [2 2 2]
 [3 3 3]]
Sum of all elements in a Matrix ==>
18
```

## ▼ Column-Wise Addition

```
In [232]: N = np.array([[1,1,1],[2,2,2],[3,3,3]])

print("\n Matrix (N) ==> \n", N)

print ("Column-Wise summation ==> ")
print (np.sum(N,axis=0))
```

```
Matrix (N) ==>
[[1 1 1]
 [2 2 2]
 [3 3 3]]
Column-Wise summation ==>
[6 6 6]
```

## ▼ Row-Wise Addition

```
In [233]: N = np.array([[1,1,1],[2,2,2],[3,3,3]])

print("\n Matrix (N) ==> \n", N)

print ("Row-Wise summation ==>")
print (np.sum(N,axis=1))
```

```
Matrix (N) ==>
[[1 1 1]
 [2 2 2]
 [3 3 3]]
Row-Wise summation ==>
[3 6 9]
```

## ▼ Kronecker Product of matrices

Kronecker Product of matrices : <https://www.youtube.com/watch?v=e1UJXvu8VZk>  
[\(https://www.youtube.com/watch?v=e1UJXvu8VZk\)](https://www.youtube.com/watch?v=e1UJXvu8VZk)

```
In [235]: M1 = np.array([[1,2,3] , [4,5,6]])
M1
```

```
Out[235]: array([[1, 2, 3],
                [4, 5, 6]])
```

```
In [236]: M2 = np.array([[10,10,10],[10,10,10]])  
M2
```

```
Out[236]: array([[10, 10, 10],  
                [10, 10, 10]])
```

```
In [237]: np.kron(M1,M2)
```

```
Out[237]: array([[10, 10, 10, 20, 20, 20, 30, 30, 30],  
                [10, 10, 10, 20, 20, 20, 30, 30, 30],  
                [40, 40, 40, 50, 50, 50, 60, 60, 60],  
                [40, 40, 40, 50, 50, 50, 60, 60, 60]])
```

## ▼ Matrix Powers

```
In [238]: M1 = np.array([[1,2],[4,5]])  
M1
```

```
Out[238]: array([[1, 2],  
                [4, 5]])
```

```
In [239]: #Matrix to the power 3  
  
M1@M1@M1
```

```
Out[239]: array([[ 57,  78],  
                [156, 213]])
```

```
In [240]: #Matrix to the power 3  
  
np.linalg.matrix_power(M1,3)
```

```
Out[240]: array([[ 57,  78],  
                [156, 213]])
```

## ▼ Tensor

What is Tensor :

- <https://www.youtube.com/watch?v=f5liqUk0ZTw> (<https://www.youtube.com/watch?v=f5liqUk0ZTw>)
- <https://www.youtube.com/watch?v=bpG3ggDM80w&t=634s> (<https://www.youtube.com/watch?v=bpG3ggDM80w&t=634s>)
- <https://www.youtube.com/watch?v=uaQeXi4E7gA> (<https://www.youtube.com/watch?v=uaQeXi4E7gA>)

```
In [242]: # Create Tensor

T1 = np.array([
    [[1,2,3], [4,5,6], [7,8,9]],
    [[10,20,30], [40,50,60], [70,80,90]],
    [[100,200,300], [400,500,600], [700,800,900]],
])

T1
```

```
Out[242]: array([[ [ 1,  2,  3],
                  [ 4,  5,  6],
                  [ 7,  8,  9]],

                [[ 10,  20,  30],
                  [ 40,  50,  60],
                  [ 70,  80,  90]],

                [[100, 200, 300],
                  [400, 500, 600],
                  [700, 800, 900]]])
```

```
In [243]: T2 = np.array([
    [[0,0,0], [0,0,0], [0,0,0]],
    [[1,1,1], [1,1,1], [1,1,1]],
    [[2,2,2], [2,2,2], [2,2,2]]

])

T2
```

```
Out[243]: array([[ [0, 0, 0],
                  [0, 0, 0],
                  [0, 0, 0]],

                [[1, 1, 1],
                  [1, 1, 1],
                  [1, 1, 1]],

                [[2, 2, 2],
                  [2, 2, 2],
                  [2, 2, 2]]])
```



## Tensor Addition



```
In [244]: A = T1+T2
          A
```

```
Out[244]: array([[ 1,  2,  3],
                 [ 4,  5,  6],
                 [ 7,  8,  9]],

               [[ 11, 21, 31],
                [ 41, 51, 61],
                [ 71, 81, 91]],

               [[102, 202, 302],
                [402, 502, 602],
                [702, 802, 902]])
```

```
In [245]: np.add(T1,T2)
```

```
Out[245]: array([[ 1,  2,  3],
                 [ 4,  5,  6],
                 [ 7,  8,  9]],

               [[ 11, 21, 31],
                [ 41, 51, 61],
                [ 71, 81, 91]],

               [[102, 202, 302],
                [402, 502, 602],
                [702, 802, 902]])
```



## Tensor Subtraction

```
In [246]: S = T1-T2
          S
```

```
Out[246]: array([[ 1,  2,  3],
                 [ 4,  5,  6],
                 [ 7,  8,  9]],

               [[ 9, 19, 29],
                [ 39, 49, 59],
                [ 69, 79, 89]],

               [[ 98, 198, 298],
                [398, 498, 598],
                [698, 798, 898]])
```

```
In [247]: np.subtract(T1,T2)
```

```
Out[247]: array([[ 1,  2,  3],
                 [ 4,  5,  6],
                 [ 7,  8,  9]],

              [[ 9, 19, 29],
               [39, 49, 59],
               [69, 79, 89]],

              [[ 98, 198, 298],
               [398, 498, 598],
               [698, 798, 898]])
```



## Tensor Element-Wise Product

```
In [248]: P = T1*T2
          P
```

```
Out[248]: array([[ 0,  0,  0],
                 [ 0,  0,  0],
                 [ 0,  0,  0]],

              [[ 10,  20,  30],
               [ 40,  50,  60],
               [ 70,  80,  90]],

              [[ 200,  400,  600],
               [ 800, 1000, 1200],
               [1400, 1600, 1800]])
```

```
In [249]: np.multiply(T1,T2)
```

```
Out[249]: array([[ 0,  0,  0],
                 [ 0,  0,  0],
                 [ 0,  0,  0]],

              [[ 10,  20,  30],
               [ 40,  50,  60],
               [ 70,  80,  90]],

              [[ 200,  400,  600],
               [ 800, 1000, 1200],
               [1400, 1600, 1800]])
```



## Tensor Element-Wise Division

In [250]: `D = T1/T2`  
`D`

C:\Anaconda\lib\site-packages\ipykernel\_launcher.py:1: RuntimeWarning: divide by zero encountered in true\_divide  
 """Entry point for launching an IPython kernel.

Out[250]: `array([[ [ inf, inf, inf],  
[ inf, inf, inf],  
[ inf, inf, inf]],  
  
[[ 10., 20., 30.],  
[ 40., 50., 60.],  
[ 70., 80., 90.]],  
  
[[ 50., 100., 150.],  
[200., 250., 300.],  
[350., 400., 450.]])`

In [251]: `np.divide(T1,T2)`

C:\Anaconda\lib\site-packages\ipykernel\_launcher.py:1: RuntimeWarning: divide by zero encountered in true\_divide  
 """Entry point for launching an IPython kernel.

Out[251]: `array([[ [ inf, inf, inf],  
[ inf, inf, inf],  
[ inf, inf, inf]],  
  
[[ 10., 20., 30.],  
[ 40., 50., 60.],  
[ 70., 80., 90.]],  
  
[[ 50., 100., 150.],  
[200., 250., 300.],  
[350., 400., 450.]])`



## Tensor Dot Product

In [252]: `T1`

Out[252]: `array([[ [ 1, 2, 3],  
[ 4, 5, 6],  
[ 7, 8, 9]],  
  
[[ 10, 20, 30],  
[ 40, 50, 60],  
[ 70, 80, 90]],  
  
[[100, 200, 300],  
[400, 500, 600],  
[700, 800, 900]])`

In [253]: T2

```
Out[253]: array([[0, 0, 0],
                [0, 0, 0],
                [0, 0, 0]],

               [[1, 1, 1],
                [1, 1, 1],
                [1, 1, 1]],

               [[2, 2, 2],
                [2, 2, 2],
                [2, 2, 2]])
```

In [254]: np.tensordot(T1,T2)

```
Out[254]: array([[ 63,   63,   63],
                 [ 630,  630,  630],
                 [6300, 6300, 6300]])
```



## Solving Equations

$$AX = B$$

Solving Equations :

- <https://www.youtube.com/watch?v=NNmiOoWt86M> (<https://www.youtube.com/watch?v=NNmiOoWt86M>)
- <https://www.youtube.com/watch?v=a2z7sZ4MSqo> (<https://www.youtube.com/watch?v=a2z7sZ4MSqo>)

In [256]: A = np.array([[1,2,3] , [4,5,6] , [7,8,9]])  
A

```
Out[256]: array([[1, 2, 3],
                 [4, 5, 6],
                 [7, 8, 9]])
```

In [257]: B = np.random.random((3,1))  
B

```
Out[257]: array([[0.09714648],
                 [0.10284749],
                 [0.7015073 ]])
```

```
In [258]: # 1st Method  
X = np.dot(np.linalg.inv(A) , B)  
X
```

```
Out[258]: array([[ 1.86931429e+15],  
                [-3.73862857e+15],  
                [ 1.86931429e+15]])
```

```
In [259]: # 2nd Method  
X = np.matmul(np.linalg.inv(A) , B)  
X
```

```
Out[259]: array([[ 1.86931429e+15],  
                [-3.73862857e+15],  
                [ 1.86931429e+15]])
```

```
In [260]: # 3rd Method  
X = np.linalg.inv(A)@B  
X
```

```
Out[260]: array([[ 1.86931429e+15],  
                [-3.73862857e+15],  
                [ 1.86931429e+15]])
```

```
In [261]: # 4th Method  
X = np.linalg.solve(A,B)  
X
```

```
Out[261]: array([[ 1.86931429e+15],  
                [-3.73862857e+15],  
                [ 1.86931429e+15]])
```



# END