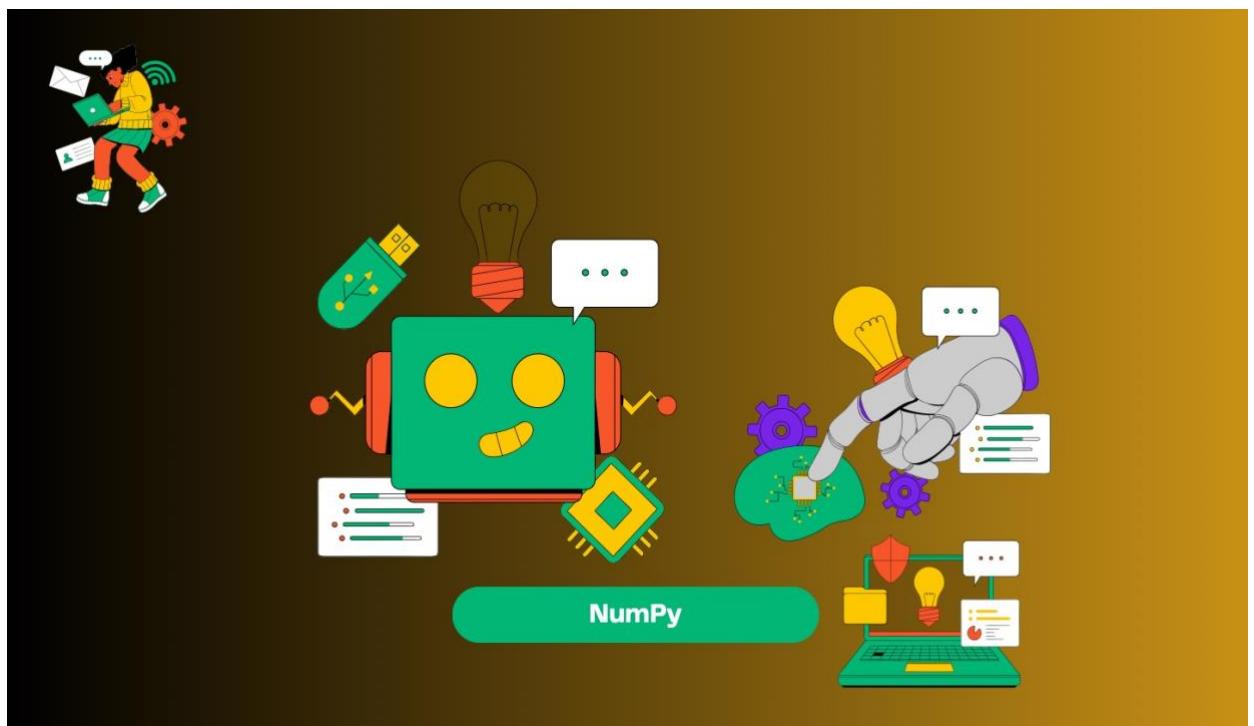


NUMPY

For Machine learning



Sadia Khatun

TABLE OF CONTENTS

Chapter 1: NumPy For Beginners	4
1.1 Introduction.....	4
1.2 The Basics (creating arrays, shape, size, data type).....	6
1.3 Accessing/changing Specific Elements, Row, Columns, etc(slicing)	8
1.4 Initializing Different Arrays (1s, 0s, full, random, etc)	13
1.5 Problem #1 (how do you initialize the array?).....	18
1.6 Be careful when copying variables!	19
1.7 Basic Mathematics (arithmetic, trigonometry, etc)	20
1.8 Linear Algebra	21
1.9 Statistics.....	22
1.10 Reorganizing Array.....	23
1.11 Load data in from a file	25
1.12 Advanced Indexing and Boolean Masking	25
1.13 Problem #2(How do you index these values?).....	27
1.14 Solution:	28
Chapter 2: More basic	29
2.1 Range	29
2.2 Adding, removing, and sorting elements	30
2.3 Transposing and reshaping a matrix	31
2.4 How to reverse an array	33
2.5 Reshaping and flattening multidimensional array	34
2.6 Working with mathematical formulas	35
2.7 How to save and load NumPy Objects	36
2.8 Importing and exporting a CSV	37
2.9 Plotting arrays with Matplotlib.....	39
Chapter 3: Visualize Different Type Of Operation Of NumPy	41

Best Wishes !!



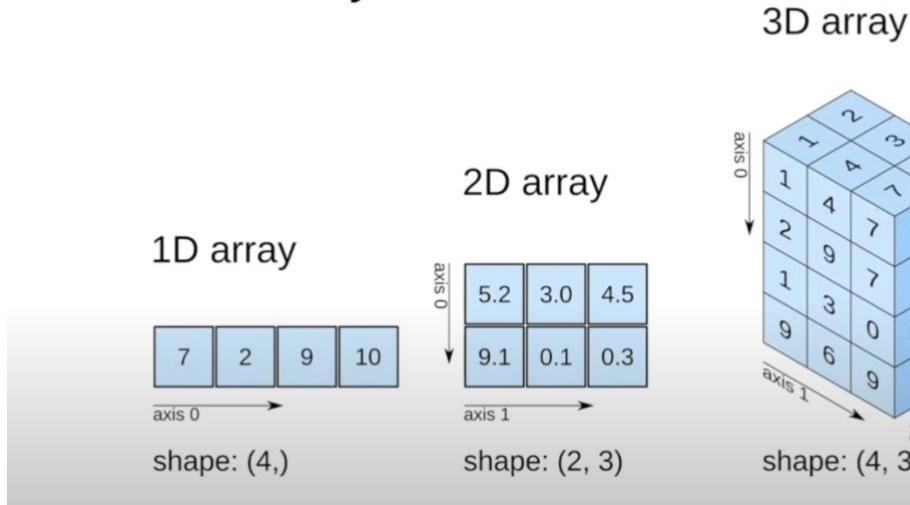
CHAPTER 1: NUMPY FOR BEGINNERS

1.1 Introduction

What is NumPy?

NumPy is a multi-dimensional array library.

What is NumPy?



Why use NumPy over list?

How are Lists different from NumPy?

Lists	NumPy
 Slow Insertion, deletion, appending, concatenation, etc. <code>a = [1,3,5]</code> <code>b = [1,2,3]</code> <code>a*b = ERROR</code>	 Fast Insertion, deletion, appending, concatenation, etc. Lots More ...! <code>a = np.array([1,3,5])</code> <code>b = np.array([1,2,3])</code> <code>a*b = np.array([1,6,15])</code>

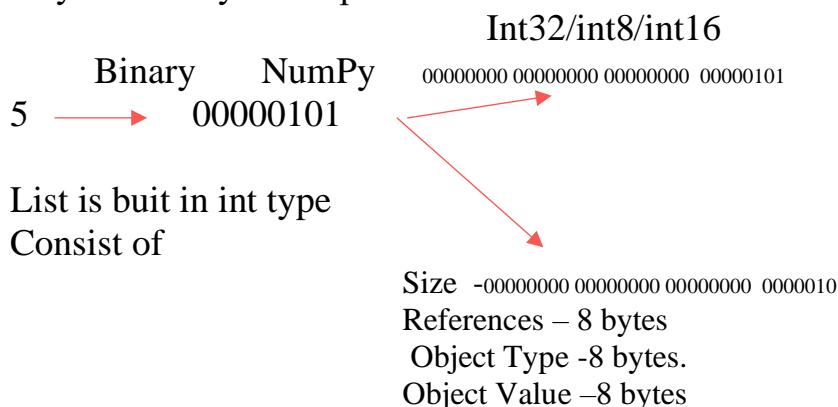
Why Is NumPy Faster? –Fixed Type

1. Faster to read less bytes of memory.
2. No type checking when iterating through objects.

3	1	4
5	7	9
3	4	6

3x3 matrix

Let's see how this integer values differ between Numpy and Lists . Let's Just zoom in on that's 5 there in that matrix. Our computer don't see 5 they see binary that represents that's 5.



So lists require lots more space than Numpy.

Why Is NumPy Faster? –Contiguous Memory



- 1.SIMD Vector Processing
- 2.Effective Cache Utilization

Applications of NumPy?

1. Mathematics (MATLAB Replacement)
2. Plotting (Matplotlib)
3. Backend (Pandas Connect 4, Digital Photography)
4. Machine Learning

1.2 The Basics (creating arrays, shape, size, data type)

```
import numpy as np
```

creating array:

```
[9] 1 a=np.array([1,2,3],dtype='int32')
     2 print(a)
[1 2 3]
```

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

Basic Operation:

Get dimension

X.NDIM

Get shape

x.shape

Get data type

x.dtype

Get each item size in bytes

x.itemsize

Get total size in bytes

x nbytes

Get number of elements in array

x.size

```
1 c=np.array([[1,2],[3,4],[5,6]],dtype='int8')
2 print(c)
3 print(c.shape)
4 print(c.dtype)
5 print(c.ndim)

[[1 2]
 [3 4]
 [5 6]]
(3, 2)
int8
2
```

```
[20] 1 b=np.array([[1,2,3],[1,3,4],[3,4,5]])
2 print(b)

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

[18] 1 # size of each elements in byte as we define 64 bit so its give 8 bytes
2 b.itemsize

8

[19] 1 b.dtype

dtype('int64')

[17] 1 # total size
2 b.nbytes # 8bytes * 9 elements

72

▶ 1 b.size #number of element

9
```

1.3 Accessing/changing Specific Elements, Row, Columns, etc(slicing)

3. Accessing/changing Specific Elements, Row, Columns, etc(slicing)

index starting from (0) so if we say 1st row means (0) index value

Indented block

```
[24] 1 a=np.array([[1,2,3,4,5,6,7],[8,9,10,11,12,13,14]])
2 print(a)
3 print(a.shape)
```

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

```
[25] 1 #Get a spacific element x[r,c], if we want 5 from this matrix
2 a[0,4]
3
```

```
5
```

```
[26] 1 # #Get a spacific element x[r,c], if we want 5 from this matrix
2 #but using negetive index
3
4 a[0,-3]
```

```
5
```

▶ 1 #Get a spacific row , we want 2nd row ,a[2nd row, every element of col (:)]
2
3 a[1, :]

```
array([ 8,  9, 10, 11, 12, 13, 14])
```

```
[28] 1 #Get a spacific column ,we want 3rd column
2
3 a[:, 2]
```

```
array([ 3, 10])
```

```
[29] 1 #Getting a little more fancy [startindex:endindex: stepsize]
2 # we want 2 , 4 , 6 element from 1st row
3 #endindex not will be include
4
5 a[0,1:6:2]
```

```
array([2, 4, 6])
```

```
[31] 1 #Getting a little more fancy [startindex:endindex: stepsize]
2 # we want 2 , 4 , 6 element from 1st row
3 #endindex not will be include
4 #use negative index
5 a[0, 1:-1:2]
```

```
array([2, 4, 6])
```

```
[32] 1 a=np.array([[1,2,3,4,5,6,7],[8,9,10,11,12,13,14]])
2 print(a)
3 print(a.shape)
```

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

```
[33] 1 #change element in matrix
2 # we want to replace 7 value with 20
3 # its 1st row and 7th col
4
5 a[0,6]=20
6 print(a)
```

```
[[ 1  2  3  4  5  6 20]
 [ 8  9 10 11 12 13 14]]
```

```
[34] 1 #change element in matrix
2 # we want replace 4th column by 7
3
4 a[:,3]=7
5 print(a)
```

```
[[ 1  2  3  7  5  6 20]
 [ 8  9 10  7 12 13 14]]
```

```
[47] 1 # create 3x3x3 matrix
2
3 c=np.array([[1,2,3],[3,2,3],[9,2,0]],[[1,3,3],[1,8,3],[1,2,3]],[[1,5,3],[1,5,3],[1,0,3]])|
4 print(c)

[[[1 2 3]
 [3 2 3]
 [9 2 0]]

 [[1 3 3]
 [1 8 3]
 [1 2 3]]

 [[1 5 3]
 [1 5 3]
 [1 0 3]]]

[48] 1 # Get [9,2,0]
2 c[0,2,:]
3

array([9, 2, 0])

[43] 1 #get just 9
2
3 c[0,2,0]

9

[51] 1 #get last value from each matrix
2
3 c[:,2,2]

array([0, 3, 3])

[61] 1 #replace
2 c[:,1,:]

array([[3, 2, 3],
 [1, 8, 3],
 [1, 5, 3]])
```

```
[35] 1 #change element in matrix  
2 # we want replace 4th column by 7,3  
3 #dimension should be same  
4 a[:,3]=[7,3]  
5 print(a)
```

```
[[ 1  2  3  7  5  6 20]  
 [ 8  9 10  3 12 13 14]]
```

3-d Example

```
[36] 1 b=np.array([[1,2],[3,4],[5,6],[7,8]])  
2 print(b)
```

```
[[[1 2]  
 [3 4]]  
  
 [[5 6]  
 [7 8]]]
```

```
[37] 1 # Get spacific element (work outside in )  
2 # we want 4  
3 # 1st matrix 2nd row 2nd column  
4 #[matrix, row, column]  
5  
6 b[0,1,1]
```

```
4
```

```
[38] 1 # want 3,4 from 1st matrix and 7,8 from 2nd matrix  
2 b[:,1,:]
```

```
array([[3, 4],  
       [7, 8]])
```

```
[61] 1 #replace  
2 c[:,1,:]  
  
array([[3, 2, 3],  
       [1, 8, 3],  
       [1, 5, 3]])  
  
▶ 1 c[:,1,:]=[2,3,4],[5,6,7],[9,0,8]  
2 c  
  
array([[[1, 2, 3],  
        [2, 3, 4],  
        [9, 2, 0]],  
  
       [[[1, 3, 3],  
         [5, 6, 7],  
         [1, 2, 3]],  
  
       [[[1, 5, 3],  
         [9, 0, 8],  
         [1, 0, 3]]])
```

1.4 Initializing Different Arrays (1s, 0s, full, random, etc)

operation	example
np.zeros()	<code>np.zeros((2, 4))</code>
np.ones	<code>np.ones((4, 2, 2), dtype='int32')</code>
np.full()	<code>np.full((3, 4), 999, dtype='float32')</code>
np.full_like	<code>b=np.full_like(a, 4)</code>
np.random.rand	<code>np.random.rand(3, 3)</code>
np.random.rand_sample	<code>np.random.random_sample(a.shape)</code>
np.random.randint()	<code>np.random.randint(100, size=(3, 3))</code>
np.identity	<code>np.identity(5)</code>
np.repeat	<code>r1=np.repeat(arr, 3)</code>
a.copy	<code>b=a.copy()</code>

initializing Different Types of arrays

```
[34] 1 #All 0s mtraix
2 # 2-D 2x4 matrix ((r,c))
3 np.zeros((2,4))

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

```
[35] 1 #All 0s mtraix
2 # 3-D 2x4x3 matrix
3 #2 matrix 4 row and 3 column in each
4 np.zeros((2,4,3))

array([[[0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.]],

       [[0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.]]])
```

```
[72] 1 #All 0s mtraix
2 # 4-D 3x4x3x3 matrix
3 #(3x2=6, 3x3 matrix)=(3,4,3,3)
4
5 np.zeros((3,2,3,3))

array([[[[0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.]],

       [[0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.]]],

      [[[0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.]],

       [[0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.]]],

      [[[0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.]],

       [[0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.]]]])
```

```
[40] 1 #all ones matrix
2 np.ones((4,2,2),dtype='int32')

array([[[1, 1],
       [1, 1]],

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

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

       [[1, 1],
       [1, 1]]], dtype=int32)

[41] 1 #any other number
2 # we want 3x4 matrix all value will be 999
3
4 np.full((3,4),999, dtype='float32')

array([[999., 999., 999., 999.],
       [999., 999., 999., 999.],
       [999., 999., 999., 999.]], dtype=float32)

[43] 1 #like a shape of a pre define matrix
2 np.full(a.shape,4)

array([[4, 4, 4, 4, 4, 4, 4],
       [4, 4, 4, 4, 4, 4, 4]])

[44] 1 # Any other number (full_like)
2 # like a shape of a pre define matrix
3 #if we use full_like no need to use a.shape
4 #just use a ( a is previously de fine matrix which shape was 2x7)
5 b=np.full_like(a,4)
6 print(b)
7 print(b.shape)

[[4 4 4 4 4 4 4]
 [4 4 4 4 4 4 4]]
(2, 7)
```

```
[46] 1 #Random decimal number
      2
      3 np.random.rand(3,3)

array([[0.4094865 , 0.84005832, 0.5917549 ],
       [0.87885947, 0.5329299 , 0.00576045],
       [0.25541651, 0.77344551, 0.97978664]])

[47] 1 #if we want to pass shape we must use rand_sample
      2 np.random.random_sample(a.shape)

array([[0.07303068, 0.05024743, 0.86696108, 0.44838652, 0.52121979,
       0.80348145, 0.48693642],
       [0.74299627, 0.927289 , 0.01115618, 0.44459581, 0.92134962,
       0.19499349, 0.13181815]])
```

```
[48] 1 # we want Random Integer values
      2 np.random.randint(7)

      3
```

```
[50] 1 # we want Random Integer values
      2 #and 3x3 matrix
      3 #random number between 100
      4 np.random.randint(100,size=(3,3))

array([[ 9, 81, 26],
       [51, 63,  4],
       [ 6, 54, 41]])
```

▶ 1 # we want Random Integer values
 2 #and 3x3 matrix
 3 #random number 10 between 100 exclude 100
 4 np.random.randint(10,100,size=(5,5))

```
array([[29, 47, 30, 48, 83],
       [76, 79, 32, 32, 48],
       [95, 19, 14, 32, 42],
       [68, 67, 93, 34, 56],
       [86, 79, 46, 38, 78]])
```

```
[54] 1 # we want Random Integer values  
2 #and 3x3 matrix  
3 #random number -10 between 100 exlcude 100  
4 np.random.randint(-10,100,size=(5,5))
```

```
array([[74, 33, 22, 54, 89],  
       [81, 69, 23, 3, 4],  
       [77, -6, 47, 56, 58],  
       [76, -7, 84, 86, 23],  
       [61, 25, 81, 10, -9]])
```

```
[55] 1 #the idendity matrix  
2 # by default square matrix  
3 # need to pass just 1 parameter  
4  
5 np.identity(5)
```

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

```
[57] 1 # repeat a array  
2 #repeat arr each value 3 times  
3 arr=np.array([1,2,3,4])  
4 r1=np.repeat(arr,3)  
5 print(r1)
```

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

```
[59] 1 # repeat a array  
2 #repeat arr matrix each value 3 times  
3 arr=np.array([[1,2,3,4]])  
4 r1=np.repeat(arr,3, axis=0)  
5 print(r1)
```

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

1.5 Problem #1 (how do you initialize the array?)

Problem #1

want to create this matrix

[1 1 1 1 1]

[1 0 0 0 1]

[1 0 9 0 1]

[1 0 0 0 1]

[1 1 1 1 1]

```
[66]  1 #method 1
      2 b=np.ones((5,5),dtype='int8')
      3 b[1:4:,1:4:]=np.zeros((3,3))
      4 b[2,2]=9
      5 print(b)
```

```
[[1 1 1 1 1]
 [1 0 0 0 1]
 [1 0 9 0 1]
 [1 0 0 0 1]
 [1 1 1 1 1]]
```

```
[68]  1 #method 2
      2 output = np.ones((5,5))
      3 z=np.zeros((3,3))
      4 z[1,1]=9
      5 output[1:4,1:4]=z
      6 print(output)
```

```
[[1. 1. 1. 1. 1.]
 [1. 0. 0. 0. 1.]
 [1. 0. 9. 0. 1.]
 [1. 0. 0. 0. 1.]
 [1. 1. 1. 1. 1.]]
```

```
[69]  1 #method 3
      2 #use negative index
      3 output = np.ones((5,5))
      4 z=np.zeros((3,3))
      5 z[1,1]=9
      6 output[1:-1,1:-1]=z
      7 print(output)
```

```
[[1. 1. 1. 1. 1.]
 [1. 0. 0. 0. 1.]
 [1. 0. 9. 0. 1.]
 [1. 0. 0. 0. 1.]
 [1. 1. 1. 1. 1.]]
```

1.6 Be careful when copying variables!

Be careful when coping arrays!!

```
[70] 1 a=np.array([1,2,3])
2 # b point to same a
3 #so if we change b then a also will be change
4 b=a
5 b[0]=100
6 print(b)
7 print(a)
```

```
[100    2    3]  
[100    2    3]
```

```
1 #solution of this we have to tell numpy to create a copy of a
2 a=np.array([1,2,3])
3 b=a.copy()
4 b[0]=100
5 print(b)
6 print(a)
```

```
[100    2    3]  
[1 2 3]
```

1.7 Basic Mathematics (arithmetic, trigonometry, etc)

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

[74] 1 # add 2 with every element
  2 a+2
  array([3, 4, 5, 6])

[75] 1 # sub 2 with every element
  2 a-2
  array([-1,  0,  1,  2])

[76] 1 a*2
  array([2, 4, 6, 8])

[77] 1 a /2
  array([0.5, 1. , 1.5, 2. ])

▶ 1 a**2
  array([ 1,  4,  9, 16])

[79] 1 b=np.array([1,0,1,0])
  2 a+b
  array([2, 2, 4, 4])

[80] 1 np.cos(b)
  array([0.54030231, 1.           , 0.54030231, 1.           ])

[82] 1 np.sin(a)
  array([ 0.84147098,  0.90929743,  0.14112001, -0.7568025 ])
```

We can do lots more cool stuff with mathematics link is below:

<https://docs.scipy.org/doc/numpy/reference/routines.math.html>

1.8 Linear Algebra

```
[87] 1 a=np.ones((2,3))
2 print(a)
3
4 b=np.full((3,2),2)
5 print(b)
6
7 # not gonna work a*b
8 # nxm = mxp
9 np.matmul(a,b)

[[1. 1. 1.]
 [1. 1. 1.]]
[[2 2]
 [2 2]
 [2 2]]
array([[6., 6.],
 [6., 6.]])
```

```
[90] 1 # Find the determinant
2 # determinant of identity matrix is one
3 c=np.identity(3)
4
5 np.linalg.det(c)
6
```

```
1.0
```

For more cool stuff visit this link below:

<https://docs.scipy.org/doc/numpy/reference/routines.linalg.html>

```
#Determinant
# Trace
# Singular Vector Decomposition
# Eigenvalues
# Matrix Norm
# Inverse
# Etc
```

1.9 Statistics

statistics

```
[92] 1 stats = np.array([[1,2,3],[4,5,6]])
2 stats
```

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

```
[93] 1 # minimum value of matrix
2 np.min(stats)
```

```
1
```

▶ 1 # minimum value of matrix
2 #for each row use axis=1
3 np.min(stats , axis=1)

```
array([1, 4])
```

```
[95] 1 # minimum value of matrix
2 #for each column use axis=0
3 np.min(stats , axis=0)
```

```
array([1, 2, 3])
```

▶ 1 np.max(stats)

```
6
```

```
[97] 1 np.sum(stats)
```

```
21
```

```
[98] 1 #sum all value of each row
2 np.sum(stats, axis=1)
```

```
array([ 6, 15])
```

1.10 Reorganizing Array

Reorganizing Array

```
[103] 1 before=np.array([[1,2,3,4],[5,6,7,8]])
2
3 print(before)
4 #if total number of values in a matrix
5 # does not same as total number of values of
6 # in reshape matrix then gives errors
7 after=before.reshape((4,2))
8 print(after)
```

```
[[1 2 3 4]
 [5 6 7 8]]
[[1 2]
 [3 4]
 [5 6]
 [7 8]]
```

```
[104] 1 after=before.reshape((2,2,2))
2 print(after)
```

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

 [[5 6]
 [7 8]]]
```



```
1 #vertically stacking vectors
2 #dim is important
3 # simple concate to matrix one top of the other
4
5 v1=np.array([1,2,3,4])
6 v2=np.array([5,6,7,8])
7 np.vstack([v1,v2])
8
```

```
array([[1, 2, 3, 4],
 [5, 6, 7, 8]])
```

```
▶ 1 #vertically stacking vectores
  2 #dim is important
  3 # simple concate to matrix one top of the other
  4
  5 v1=np.array([1,2,3,4])
  6 v2=np.array([5,6,7,8])
  7 np.vstack([v1,v1,v2,v2])
  8

array([[1, 2, 3, 4],
       [1, 2, 3, 4],
       [5, 6, 7, 8],
       [5, 6, 7, 8]])

▶ 1 #horizontal stack
  2
  3 h1=np.array([1,2,3,4])
  4 h2=np.array([5,6,7,8])
  5 np.hstack((h1,h2))
  6
  7

array([1, 2, 3, 4, 5, 6, 7, 8])

[109] 1 #horizontal stack
  2
  3 h1=np.ones((2,4))
  4 h2=np.zeros((2,2))
  5
  6 np.hstack((h1,h2))
  7

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

1.11 Load data in from a file

Miscellaneous

Load Data from File

```
▶ 1 # we have to use delimiter properly
  2 # accordingly which one use in separate data in file
  3 filedata = np.genfromtxt('data.txt',delimiter=',')
  4 #by default its float
  5 filedata

array([[ 1.,  13.,  21.,  11., 196.,  75.,   4.,   3.,  34.,   6.,
         7.,
        8.,   0.,   1.,   2.,   3.,   4.,   5.],
       [ 3.,  42.,  12.,  33., 766.,  75.,   4.,   55.,   6.,   4.,
         3.,
        4.,   5.,   6.,   7.,   0.,  11.,  12.],
       [ 1.,  22.,  33.,  11., 999.,  11.,   2.,   1.,  78.,   0.,
         1.,
        2.,   9.,   8.,   7.,   1.,  76.,  88.]])]

[116] 1 # if we want to int then need to type cast
      2
      3 filedata.astype('int32')
      4
      5 filedata
      6 # actually copy change to other file
      7 filedata =filedata.astype('int32')
      8 filedata

array([[ 1,  13,  21,  11, 196,  75,   4,   3,  34,   6,   7,
         8,   0,
        1,   2,   3,   4,   5],
       [ 3,  42,  12,  33, 766,  75,   4,   55,   6,   4,
         3,
        6,   7,   0,  11,  12],
       [ 1,  22,  33,  11, 999,  11,   2,   1,  78,   0,
         1,
        2,   9,
        8,   7,   1,  76,  88]], dtype=int32)
```

Create this data.txt file:

```
1,13,21,11,196,75,4,3,34,6,7,8,0,1,2,3,4,5
3,42,12,33,766,75,4,55,6,4,3,4,5,6,7,0,11,12
1,22,33,11,999,11,2,1,78,0,1,2,9,8,7,1,76,88
```

1.12 Advanced Indexing and Boolean Masking

```

[117] 1 #if value is greater than 50 give true
      2 filedatal>50

      array([[False, False, False, False, True, True, False, False,
              False, False, False, False, False, False, False, False],
             [False, False, False, False, True, True, False, True, False,
              False, False, False, False, False, False, False, False],
             [False, False, False, False, True, True, False, True, False,
              False, False, False, False, False, False, False, False],
             [False, False, False, False, True, False, False, False, True,
              False, False, False, False, False, False, False, True]])

[118] 1 #give those values which value is greater than 50
      2 filedatal[filedata>50]

      array([196, 75, 766, 75, 55, 999, 78, 76, 88], dtype=int32)

[120] 1 #any value greater than 50
      2 np.any(filedata>50, axis=0)

      array([False, False, False, False, True, True, False, True, True,
              False, False, False, False, False, False, True, True])

[121] 1 #any value greater than 50
      2 np.all(filedata>50, axis=0)

      array([False, False, False, False, True, False, False, False,
              False, False, False, False, False, False, False, False])

[122] 1 ((filedata>50) & (filedata<100))

      array([[False, False, False, False, False, True, False, False, False,
              False, False, False, False, False, False, False, False],
             [False, False, False, False, False, True, False, True, False,
              True, False, False, False, False, False, False, False],
             [False, False, False, False, False, True, False, True, False,
              False, False, False, False, False, False, False, False],
             [False, False, False, False, False, True, False, True, False,
              True, False, False, False, False, False, False, True],
             [False, False, False, False, False, True, False, True, False,
              True, False, False, False, False, False, False, True]])

[123] 1 (~((filedata>50) & (filedata<100)))

      array([[ True,  True,  True,  True,  True, False,  True,  True,  True,
              True,  True,  True,  True,  True,  True,  True],
             [ True,  True,  True,  True,  True, True,  False,  True,  False,
              True,  True,  True,  True,  True,  True,  True,  True],
             [ True,  True,  True,  True,  True, True,  True,  True,  True,
              True,  True,  True,  True,  True,  True,  True,  True],
             [ True,  True,  True,  True,  True, True,  True,  True,  True,
              True,  True,  True,  True,  True,  True,  False,  True],
             [ True,  True,  True,  True,  True, True,  True,  True,  True,
              True,  True,  True,  True,  True,  True,  False,  True]])

```

1.13 Problem #2(How do you index these values?)

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

1.14 Solution:

```
[126] 1 a=np.array([[1,2,3,4,5],[6,7,8,9,10],[11,12,13,14,15],  
2 | [16,17,18,19,20],[21,22,23,24,24],[26,27,28,29,30]])  
3 print(a)  
  
[[ 1  2  3  4  5]  
 [ 6  7  8  9 10]  
 [11 12 13 14 15]  
 [16 17 18 19 20]  
 [21 22 23 24 24]  
 [26 27 28 29 30]]  
  
[127] 1 # get the value of 11,12,16,17  
2 # blue area  
3 a[2:4,0:2]  
  
array([[11, 12],  
       [16, 17]])  
  
▶ 1 # get the 2,8,14,20  
2 #green mark values  
3  
4 #use list as index [row index], [column index]  
5  
6 a[[0,1,2,3],[1,2,3,4]]  
  
array([ 2,  8, 14, 20])  
  
[129] 1 #get 4,5,24,25,29,30  
2 #red mark area values  
3 # use list and slicing  
4  
5 a[[0,3,4],3:]  
6  
  
array([[ 4,  5],  
       [19, 20],  
       [24, 24]])
```

This Notebook co-lab link:

https://colab.research.google.com/drive/1SjaKmTaHdG_ZKFM0Mjo0DcrGf4GQlToO?usp=sharing

CHAPTER 2: MORE BASIC

2.1 Range

This section covers

`np.arange()`,
`np.linspace()`

You can create an array with a range of elements:

```
>>> np.arange(4)
array([0, 1, 2, 3])
```

And even an array that contains a range of evenly spaced intervals. To do this, you will specify the **first number**, **last number**, and the **step size**.

```
>>> np.arange(2, 9, 2)
array([2, 4, 6, 8])
```

You can also use `np.linspace()` to create an array with values that are spaced linearly in a specified interval:

```
>>> np.linspace(0, 10, num=5)
array([ 0. ,  2.5,  5. ,  7.5, 10. ])
```

2.2 Adding, removing, and sorting elements

This section covers np.sort(), np.concatenate()

You can quickly sort the numbers in ascending order with:

```
>>> np.sort(arr)
array([1, 2, 3, 4, 5, 6, 7, 8])
```

If you start with these arrays:

```
>>> a = np.array([1, 2, 3, 4])
>>> b = np.array([5, 6, 7, 8])
```

You can concatenate them with `np.concatenate()`.

```
>>> np.concatenate((a, b))
array([1, 2, 3, 4, 5, 6, 7, 8])
```

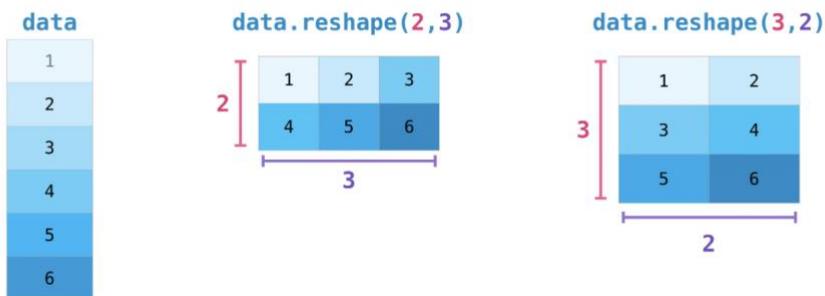
2.3 Transposing and reshaping a matrix

This section covers

`arr.reshape()`, `arr.transpose()`, `arr.T`

You may also need to switch the dimensions of a matrix. This can happen when, for example, you have a model that expects a certain input shape that is different from your dataset. This is where the `reshape` method can be useful. You simply need to pass in the new dimensions that you want for the matrix.

```
>>> data.reshape(2, 3)
array([[1, 2, 3],
       [4, 5, 6]])
>>> data.reshape(3, 2)
array([[1, 2],
       [3, 4],
       [5, 6]])
```



You can also use `.transpose()` to reverse or change the axes of an array according to the values you specify.

If you start with this array:

```
>>> arr = np.arange(6).reshape((2, 3))
>>> arr
array([[0, 1, 2],
       [3, 4, 5]])
```

You can transpose your array with `arr.transpose()`.

```
>>> arr.transpose()
array([[0, 3],
       [1, 4],
       [2, 5]])
```

You can also use `arr.T`:

```
>>> arr.T
array([[0, 3],
       [1, 4],
       [2, 5]])
```

2.4 How to reverse an array

This section covers np.flip()

Reversing a 1D array

If you begin with a 1D array like this one:

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

You can reverse it with:

```
>>> reversed_arr = np.flip(arr)
```

If you want to print your reversed array, you can run:

```
>>> print('Reversed Array: ', reversed_arr)
Reversed Array: [8 7 6 5 4 3 2 1]
```

2.5 Reshaping and flattening multidimensional array

This section covers

.flatten(), ravel()

If you start with this array:

```
>>> x = np.array([[1 , 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
```

You can use `flatten` to flatten your array into a 1D array.

```
>>> x.flatten()
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
```

When you use `flatten`, changes to your new array won't change the parent array.

For example:

```
>>> a1 = x.flatten()
>>> a1[0] = 99
>>> print(x) # Original array
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
>>> print(a1) # New array
[99  2  3  4  5  6  7  8  9 10 11 12]
```

But when you use `ravel`, the changes you make to the new array will affect the parent array.

For example:

```
>>> a2 = x.ravel()
>>> a2[0] = 98
>>> print(x) # Original array
[[98  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
>>> print(a2) # New array
[98  2  3  4  5  6  7  8  9 10 11 12]
```

2.6 Working with mathematical formulas

$$MeanSquareError = \frac{1}{n} \sum_{i=1}^n (Y_{prediction_i} - Y_i)^2$$

Implementing this formula is simple and straightforward in NumPy:

```
error = (1/n) * np.sum(np.square(predictions - labels))
```

What makes this work so well is that `predictions` and `labels` can contain one or a thousand values. They only need to be the same size.

You can visualize it this way:

```
predictions  labels
      |        |
      1        1
      |        |
      1        2
      |        |
      1        3
```

error = (1/3) * np.sum(np.square(1 - 2))

2.7 How to save and load NumPy Objects

This section covers

`np.save`, `np.savez`, `np.savetxt`, `np.load`, `np.loadtxt`

.npy file extension

It's easy to save and load an array with `np.save()`. Just make sure to specify the array you want to save and a file name. For example, if you create this array:

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

You can save it as "filename.npy" with:

```
>>> np.save('filename', a)
```

You can use `np.load()` to reconstruct your array.

```
>>> b = np.load('filename.npy')
```

If you want to check your array, you can run:

```
>>> print(b)
[1 2 3 4 5 6]
```

You can save a NumPy array as a plain text file like a `.csv` or `.txt` file with `np.savetxt`.

For example, if you create this array:

```
>>> csv_arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
```

You can easily save it as a `.csv` file with the name "new_file.csv" like this:

```
>>> np.savetxt('new_file.csv', csv_arr)
```

You can quickly and easily load your saved text file using `loadtxt()`:

```
>>> np.loadtxt('new_file.csv')
array([1., 2., 3., 4., 5., 6., 7., 8.])
```

2.8 Importing and exporting a CSV

It's simple to read in a CSV that contains existing information. The best and easiest way to do this is to use [Pandas](#).

```
>>> import pandas as pd

>>> # If all of your columns are the same type:
>>> x = pd.read_csv('music.csv', header=0).values
>>> print(x)
[['Billie Holiday' 'Jazz' 1300000 27000000]
 ['Jimmie Hendrix' 'Rock' 2700000 70000000]
 ['Miles Davis' 'Jazz' 1500000 48000000]
 ['SIA' 'Pop' 2000000 74000000]]

>>> # You can also simply select the columns you need:
>>> x = pd.read_csv('music.csv', usecols=['Artist', 'Plays']).values
>>> print(x)
[['Billie Holiday' 27000000]
 ['Jimmie Hendrix' 70000000]
 ['Miles Davis' 48000000]
 ['SIA' 74000000]]
```

The diagram illustrates the process of reading a CSV file into a Pandas DataFrame. On the left, a screenshot of Microsoft Excel shows a table named 'music.csv' with four columns: Artist, Genre, Listeners, and Plays. The data consists of five rows for artists Billie Holiday, Jimi Hendrix, Miles Davis, and SIA. An arrow points from the Excel table to the right, where the corresponding Pandas DataFrame is shown.

	Artist	Genre	Listeners	Plays
0	Billie Holiday	Jazz	1,300,000	27,000,000
1	Jimi Hendrix	Rock	2,700,000	70,000,000
2	Miles Davis	Jazz	1,500,000	48,000,000
3	SIA	Pop	2,000,000	74,000,000

If you created this array "a"

```
>>> a = np.array([[-2.58289208,  0.43014843, -1.24082018, 1.59572603],  
...                 [ 0.99027828,  1.17150989,  0.94125714, -0.14692469],  
...                 [ 0.76989341,  0.81299683, -0.95068423,  0.11769564],  
...                 [ 0.20484034,  0.34784527,  1.96979195,  0.51992837]])
```

You could create a Pandas dataframe

```
>>> df = pd.DataFrame(a)  
>>> print(df)  
          0         1         2         3  
0 -2.582892  0.430148 -1.240820  1.595726  
1  0.990278  1.171510  0.941257 -0.146925  
2  0.769893  0.812997 -0.950684  0.117696  
3  0.204840  0.347845  1.969792  0.519928
```

You can easily save your dataframe with:

```
>>> df.to_csv('pd.csv')
```

And read your CSV with:

```
>>> data = pd.read_csv('pd.csv')
```

	Unnamed: 0	0	1	2	3
0	0	-2.582892	0.430148	-1.240820	1.595726
1	1	0.990278	1.171510	0.941257	-0.146925
2	2	0.769893	0.812997	-0.950684	0.117696
3	3	0.204840	0.347845	1.969792	0.519928

2.9 Plotting arrays with Matplotlib

If you need to generate a plot for your values, it's very simple with **Matplotlib**.

For example, you may have an array like this one:

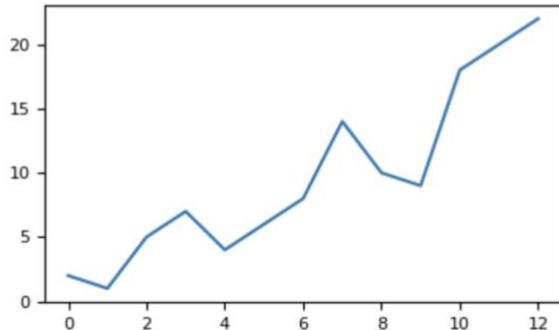
```
>>> a = np.array([2, 1, 5, 7, 4, 6, 8, 14, 10, 9, 18, 20, 22])
```

If you already have Matplotlib installed, you can import it with:

```
>>> import matplotlib.pyplot as plt  
  
# If you're using Jupyter Notebook, you may also want to run the following  
# line of code to display your code in the notebook:  
  
%matplotlib inline
```

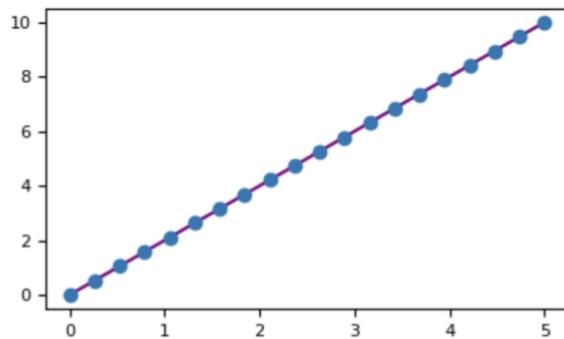
All you need to do to plot your values is run:

```
>>> plt.plot(a)  
  
# If you are running from a command line, you may need to do this:  
# >>> plt.show()
```



For example, you can plot a 1D array like this:

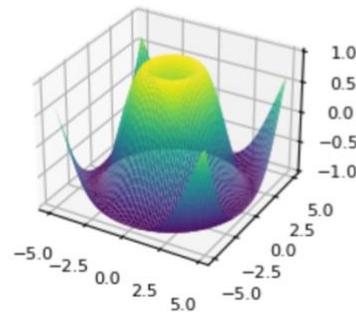
```
>>> x = np.linspace(0, 5, 20)
>>> y = np.linspace(0, 10, 20)
>>> plt.plot(x, y, 'purple') # line
>>> plt.plot(x, y, 'o')      # dots
```



With Matplotlib, you have access to an enormous number of visualization options.

```
>>> fig = plt.figure()
>>> ax = fig.add_subplot(projection='3d')
>>> X = np.arange(-5, 5, 0.15)
>>> Y = np.arange(-5, 5, 0.15)
>>> X, Y = np.meshgrid(X, Y)
>>> R = np.sqrt(X**2 + Y**2)
>>> Z = np.sin(R)

>>> ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap='viridis')
```



CHAPTER 3: VISUALIZE DIFFERENT TYPE OF OPERATION OF NUMPY

Command

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



NumPy Array

1
2
3

#

data	data[0]	data[1]	data[0:2]	data[1:]	data[-2:]	data
0 1	1		1 2	2		0 1
1 2		2		2	2	1 2
2 3				3	3	2 3

#

data	ones
<code>data = np.array([1,2])</code>	<code>ones = np.ones(2)</code>

#

$$\text{data} + \text{ones} = \begin{array}{|c|}\hline 1 \\ \hline 2 \\ \hline \end{array} + \begin{array}{|c|}\hline 1 \\ \hline 1 \\ \hline \end{array} = \begin{array}{|c|}\hline 2 \\ \hline 3 \\ \hline \end{array}$$

#

$$\text{data} - \text{ones} = \begin{array}{|c|}\hline 1 \\ \hline 2 \\ \hline \end{array} - \begin{array}{|c|}\hline 1 \\ \hline 1 \\ \hline \end{array} = \begin{array}{|c|}\hline 0 \\ \hline 1 \\ \hline \end{array}$$
$$\text{data} * \text{data} = \begin{array}{|c|}\hline 1 \\ \hline 2 \\ \hline \end{array} * \begin{array}{|c|}\hline 1 \\ \hline 2 \\ \hline \end{array} = \begin{array}{|c|}\hline 1 \\ \hline 4 \\ \hline \end{array}$$
$$\text{data} / \text{data} = \begin{array}{|c|}\hline 1 \\ \hline 2 \\ \hline \end{array} / \begin{array}{|c|}\hline 1 \\ \hline 2 \\ \hline \end{array} = \begin{array}{|c|}\hline 1 \\ \hline 1 \\ \hline \end{array}$$

#

$$\begin{array}{|c|} \hline 1 \\ \hline 2 \\ \hline \end{array} * 1.6 = \begin{array}{|c|} \hline 1 \\ \hline 2 \\ \hline \end{array} * \begin{array}{|c|} \hline 1.6 \\ \hline 1.6 \\ \hline \end{array} = \begin{array}{|c|} \hline 1.6 \\ \hline 3.2 \\ \hline \end{array}$$

data

1
2
3

.max() = 3

data

1
2
3

.min() = 1

data

1
2
3

.sum() = 6

#

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

1	2
3	4
5	6

#

0	1
1	2
2	3

data

0	1	2
1	3	4
2	5	6

data[0,1]

0	1	2
1	3	4
2	5	6

data[1:3]

0	1	2
1	3	4
2	5	6

data[0:2,0]

0	1	2
1	3	4
2	5	6

#

data

1	2
3	4
5	6

.max() = 6

data

1	2
3	4
5	6

.min() = 1

data

1	2
3	4
5	6

.sum() = 21

#

$$\begin{array}{c}
 \text{data} \\
 \downarrow \\
 \begin{array}{|c|c|} \hline 1 & 2 \\ \hline 5 & 3 \\ \hline 4 & 6 \\ \hline \end{array} .\max(\text{axis}=0) = \begin{array}{|c|c|} \hline 1 & 2 \\ \hline 5 & 3 \\ \hline 4 & 6 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 5 & 6 \\ \hline \end{array}
 \end{array}
 \quad
 \begin{array}{c}
 \text{data} \\
 \rightarrow \\
 \begin{array}{|c|c|} \hline 1 & 2 \\ \hline 5 & 3 \\ \hline 4 & 6 \\ \hline \end{array} .\max(\text{axis}=1) = \begin{array}{|c|c|} \hline 1 & 2 \\ \hline 5 & 3 \\ \hline 4 & 6 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 2 \\ \hline 5 \\ \hline 6 \\ \hline \end{array}
 \end{array}$$

#

$$\begin{array}{ccc}
 \text{data} & & \text{ones} \\
 \begin{array}{|c|c|} \hline 1 & 2 \\ \hline 3 & 4 \\ \hline \end{array} + \begin{array}{|c|c|} \hline 1 & 1 \\ \hline 1 & 1 \\ \hline \end{array} & = & \begin{array}{|c|c|} \hline 2 & 3 \\ \hline 4 & 5 \\ \hline \end{array}
 \end{array}$$

#

$$\begin{array}{ccccc}
 \text{data} & & \text{data} & & \text{ones_row} \\
 \begin{array}{|c|c|} \hline 1 & 2 \\ \hline 3 & 4 \\ \hline 5 & 6 \\ \hline \end{array} + \begin{array}{|c|c|} \hline 1 & 1 \\ \hline \end{array} & = & \begin{array}{|c|c|} \hline 1 & 2 \\ \hline 3 & 4 \\ \hline 5 & 6 \\ \hline \end{array} + \begin{array}{|c|c|} \hline 1 & 1 \\ \hline 1 & 1 \\ \hline \end{array} & = & \begin{array}{|c|c|} \hline 2 & 3 \\ \hline 4 & 5 \\ \hline 6 & 7 \\ \hline \end{array}
 \end{array}$$

#

$$\begin{array}{ccc}
 \text{np.ones}(3) \rightarrow \begin{array}{|c|} \hline 1 \\ \hline 1 \\ \hline 1 \\ \hline \end{array} & \text{np.zeros}(3) \rightarrow \begin{array}{|c|} \hline 0 \\ \hline 0 \\ \hline 0 \\ \hline \end{array} & \text{np.random.random}(3) \rightarrow \begin{array}{|c|} \hline 0.5967 \\ \hline 0.0606 \\ \hline 0.2223 \\ \hline \end{array}
 \end{array}$$

#

$$\begin{array}{ccc}
 \text{np.ones}((3,2)) \rightarrow 3 \begin{array}{|c|c|} \hline 1 & 1 \\ \hline 1 & 1 \\ \hline 1 & 1 \\ \hline \end{array} & \text{np.zeros}((3,2)) \rightarrow \begin{array}{|c|c|} \hline 0 & 0 \\ \hline 0 & 0 \\ \hline 0 & 0 \\ \hline \end{array} & \text{np.random.random}((3,2)) \rightarrow \begin{array}{|c|c|} \hline 0.37 & 0.88 \\ \hline 0.75 & 0.79 \\ \hline 0.63 & 0.16 \\ \hline \end{array}
 \end{array}$$

#

data	data.T
1 2	1 3 5
3 4	2 4 6
5 6	

data	data.reshape(2,3)	data.reshape(3,2)
1 2 3 4 5 6		

#

```
error = (1/3) * np.sum(np.square(
```

0
-1
-2

```
error = (1/3) * np.sum(
```

0
1
4

```
error = (1/3) * 5
```

)

Explore Advance Operation By Yourself!!!!