NumPy

In numpy we can represent arrays in 1D, 2D, 3D, 4D, ...moreThe main difference between lists and numpy array 1.Numpy is fixed type (int32,int16,int8) 2.Faster to read less bytes of memory 3.No type checking 4.Numpy uses contiguous memory

Load NumPy

In [159…
```
import numpy as np
```

Basics operations

Initializing the array

In [160…
```
a = np.array([1,2,3,4]) #1D
print(a)
```

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

In [161…
```
b = np.array([[1,2,3,4,5],[6,7,8,9,10]])
print(b)
```

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

Get the dimensions

In [162…
```
print(a.ndim)
print(b.ndim)
```

```
1
2
```

Get shape --> Showing columns and rows

In [163…
```
b.shape # 2 Rows and 5 Columns
```

Out[163…
```
(2, 5)
```

Get the datatype of array

In [164…
```
a.dtype
```

Out[164…
```
dtype('int32')
```

We can give the datatype we want when we initialize the array

In [165…
```
c = np.array([1,2,3,46,6,6,6,6],dtype='int8') #int 32 bytes
print(c)
c.dtype
```

```
[ 1  2  3 46  6  6  6  6]
```

Out[165…
```
dtype('int8')
```

Get size of variable

In [166…
```
c.itemsize #It is because it is initialize with int32 So 32/8 = 4 bites
```

Out[166…
```
1
```

Get the total size

In [167…
```
c.nbytes #It is like len in lists
```

Out[167…
```
8
```

Accessing/Changing specific elements, rows, columns etc

In [168…
```
a2 = np.array([[1,2,3,4,5,6,7],[8,9,10,11,12,13,14]])
print(a2)
```

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

Get a specific element [row, column]

In [169…
```
a2[1,3]
```

Out[169…
```
11
```

Get a specific row

In [170…
```
a2[0,:]
```

Out[170…
```
array([1, 2, 3, 4, 5, 6, 7])
```

Get a specific column

In [171…
```
a2[:,2]
```

Out[171…
```
array([ 3, 10])
```

Using negative index [startindex:endindex:stepsize]

In [172…
```
a2[:,1:4-2]
```

Out[172…
```
array([[2],
       [9]])
```

Changing the specific element

In [173…
```
a2[1,4] = 100 #1st row and 4th column element
print(a2)
```

```
[[  1   2   3   4   5   6   7]
 [  8   9  10  11 100  13  14]]
```

Changing entire columns elements

In [174…
```
a2[:,2] = [209,203]
print(a2)
```

```
[[  1   2 209   4   5   6   7]
 [  8   9 203  11 100  13  14]]
```

3 d example

In [175…
```
b3 = np.array([[[1,2],[3,4]],[[5,6],[7,8]]])
print(b3)
```

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

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

Get specific element

In [176…
```
b3[0,1,0]
```

Out[176…
```
3
```

In [177…
```
b3[:,1,0]
```

Out[177…
```
array([3, 7])
```

In [178…
```
b3[0] #or b3[0,:,:]
```

Out[178…
```
array([[1, 2],
       [3, 4]])
```

Replacing the elements

In [179…
```
b3[1] = [[20,30],[40,68]]
print(b3)
```

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

 [[20 30]
  [40 68]]]
```

Initializing Different types of Arrays

All 0's matrix

In [180…
```
np.zeros(3) # 1 D
```

Out[180…
```
array([0., 0., 0.])
```

In [181…
```
np.zeros((2,3)) #3 D
```

Out[181…
```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

In [182…
```
np.zeros((1,2,3,4)) #4 D
```

Out[182…
```
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.]]]])
```

All 1's matrix

In [183…
```
np.ones(2)
```

Out[183…
```
array([1., 1.])
```

In [184…
```
np.ones((4,4),dtype='int32')
```

Out[184…
```
array([[1, 1, 1, 1],
       [1, 1, 1, 1],
       [1, 1, 1, 1],
       [1, 1, 1, 1]])
```

Any other number

In [185…
```
np.full((3,3),99)   #npfull((row,column),number)
```

Out[185…
```
array([[99, 99, 99],
       [99, 99, 99],
       [99, 99, 99]])
```

Any other number (full_like)

In [186…
```
np.full_like(a, 4 ) #full_like(defined array, number) #It will take the shape of al
```

Out[186…
```
array([4, 4, 4, 4])
```

Random Decimal numbers

In [187…
```
np.random.rand(4,2) #generate the random array of given shape
```

Out[187…
```
array([[0.77422961, 0.59482467],
       [0.23473232, 0.24127701],
       [0.13849835, 0.16297354],
       [0.95402683, 0.1056482 ]])
```

Random integer numbers

In [188…
```
np.random.randint(7,size=(3,4)) #np.random.randint(startfrom,end,size_of_array)
```

Out[188…
```
array([[1, 3, 4, 4],
       [4, 3, 3, 2],
       [4, 0, 2, 0]])
```

Identity matrix

In [189…
```
np.identity(4)
```

Out[189…
```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

Repeat array

In [190…
```
arr = np.array([[1,2,3]])
r1 = np.repeat(arr,3,axis=0)
print(r1)
```

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

Exercise 1 using above functions

In [191…
```
 #Example matrix in 5*5
array = np.array([[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]])
print(array)
print(array.shape)
```

```
[[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]]
(5, 5)
```

In [192…
```
#You can get the above array in other way
output = np.ones((5,5))
#print(output)

z = np.zeros((3,3))
z[1,1] = 9
#print(z)

output[1:4,1:4] = z
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.]]
```

Be careful when copying arrays

In [193…
```
a = np.array([1,2,3])
b = a #Copied elements of a to b using assignment operator
b[0] = 100 #Changed the value in b array
print(a) #Printed the a array but it is also changed
```

```
[100   2   3]
```

In [194…
```
#To avoid the above problem
d = np.array([3,4,5,6])
c = d.copy()
print(c)
```

```
[3 4 5 6]
```

Mathematics

In [195…
```
z = np.array([10,20,30,40,50])
print(z)
```

```
[10 20 30 40 50]
```

In [237…
```
#Adding two different sized matrix
x = np.array([[1],[2],[3]])
print(x)
y = np.array([4,5,6])
print(y)
print(x+y)
```

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

In [196…
```
z + 2 #add 2 to all elements in array
```

Out[196…
```
array([12, 22, 32, 42, 52])
```

In [197…
```
z - 2 #Subtract 2 from all elements
```

Out[197…
```
array([ 8, 18, 28, 38, 48])
```

In [198…
```
z * 2 #Mutiply 2 with all elements
```

Out[198…
```
array([ 20,  40,  60,  80, 100])
```

In [199…
```
z / 2 # Divide 2 to all elements
```

Out[199…
```
array([ 5., 10., 15., 20., 25.])
```

In [200…
```
z ** 2 #Gives the power of 2 of all elements
```

Out[200…
```
array([ 100,  400,  900, 1600, 2500])
```

In [201…
```
y = np.array([11,22,33,44,55])
z+y #Adding two arrays
```

Out[201…
```
array([ 21,  42,  63,  84, 105])
```

In [202…
```
#Sin, cos, tan values
np.sin(z)
```

Out[202…
```
array([-0.54402111,  0.91294525, -0.98803162,  0.74511316, -0.26237485])
```

In [203…
```
np.cos(z)
```

Out[203…
```
array([-0.83907153,  0.40808206,  0.15425145, -0.66693806,  0.96496603])
```

In [204…
```
np.tan(z)
```

Out[204…
```
array([ 0.64836083,  2.23716094, -6.4053312 , -1.11721493, -0.27190061])
```

Linear Algebra

Multipy the different sized matrix

In [205…
```
a = np.ones((2,3))
print(a)

b = np.full((3,2),10)
print(b)
```

```
[[1. 1. 1.]
 [1. 1. 1.]]
[[10 10]
 [10 10]
 [10 10]]
```

In [206…
```
c = np.matmul(a,b)
print(c)
```

```
[[30. 30.]
 [30. 30.]]
```

Calculate the determinant

In [207…
```
det = np.array([[3,4,5],[7,8,9],[10,23,12]])
print(det)
np.linalg.det(det)
```

```
[[ 3  4  5]
 [ 7  8  9]
 [10 23 12]]
```

Out[207…
```
95.99999999999999
```

Trace of the Matrix

In [208…
```
np.trace(det) # Sum of Diagonal elements
```

Out[208…
```
23
```

Rank of the Matrix

In [209…
```
np.linalg.matrix_rank(det)
```

Out[209…
```
3
```

Calculate Eigen Vectors

In [210…
```
eg = np.ones((3,2))
print(eg)
print(np.linalg.eig(det)) #This function gives eigen vectors and values for squared
np.linalg.eigvals(det) #Eigen values for standard matrix
```

```
[[1. 1.]
 [1. 1.]
 [1. 1.]]
EigResult(eigenvalues=array([27.76981839, -0.89132247, -3.87849593]), eigenvectors=a
rray([[-0.2470769 , -0.78837201, -0.36288044],
       [-0.47258859, -0.00204184, -0.4173918 ],
       [-0.84593914,  0.61519542,  0.83312776]]))
```

Out[210… `array([27.76981839, -0.89132247, -3.87849593])`

Matrix Norm

In [211…
```
print(eg)
np.linalg.norm(eg)
```

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

Out[211… `2.449489742783178`

Inverse of a Matrix

In [212…
```
np.linalg.inv(det)
```

Out[212…
```
array([[-1.15625   ,  0.69791667, -0.04166667],
       [ 0.0625    , -0.14583333,  0.08333333],
       [ 0.84375   , -0.30208333, -0.04166667]])
```

Statistics

In [213…
```
stats = np.array([[1,2,3,],[4,5,6]])
print(stats)
```

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

In [214…
```
np.min(stats)
```

Out[214… `1`

In [215…
```
np.max(stats)
```

Out[215… `6`

In [216…
```
np.max(stats,axis=1) #When we gave axis = 1 It will return both max values of that
```

Out[216… `array([3, 6])`

In [217…
```
np.sum(stats)
```

Out[217… `21`

In [218…
```
np.mean(stats)
```

Out[218…
```
3.5
```

In [219…
```
np.median(stats)
```

Out[219…
```
3.5
```

In [220…
```
np.var(stats)
```

Out[220…
```
2.9166666666666665
```

In [221…
```
np.std(stats)
```

Out[221…
```
1.707825127659933
```

Reorganizing Arrays

In [222…
```
before = np.array([[1,2,3,4],[5,6,7,8]])
print(before)
print(before.shape)
```

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

Reshaping the matrix

In [223…
```
after = before.reshape((4,2)) #The multiplication of shape should be equal to total
print(after)
```

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

Vertically stacking vectors

In [224…
```
#Vertical stacking is non other than appending the matrix
v1 = np.array([1,2,3,4])
v2 = np.array([5,6,7,8])

np.vstack([v1,v2,v2,v1])
```

Out[224…
```
array([[1, 2, 3, 4],
       [5, 6, 7, 8],
       [5, 6, 7, 8],
       [1, 2, 3, 4]])
```

In [225…
```
#Horizontal stacking is same as vertical but it will add horizontally

np.hstack((v1,v2))
```

Out[225…
```
array([1, 2, 3, 4, 5, 6, 7, 8])
```

Miscellaneous

Load data from file

In [226…
```
filedata = np.genfromtxt('numpydata.txt',delimiter = ',') #Default float type
filedata = filedata.astype('int32')
filedata
```

Out[226…
```
array([[ 1,  2,  3,  4,  1,  2],
       [ 4,  3,  5,  4,  3,  4],
       [ 5,  6,  7,  8,  6,  5],
       [ 1,  2,  3,  4,  5, 11]])
```

Boolean Masking and Advance indexing

In [227…
```
filedata[filedata >3]
```

Out[227…
```
array([ 4,  4,  5,  4,  4,  5,  6,  7,  8,  6,  5,  4,  5, 11])
```

In [228…
```
filedata < 4
```

Out[228…
```
array([[ True,  True,  True, False,  True,  True],
       [False,  True, False, False,  True, False],
       [False, False, False, False, False, False],
       [ True,  True,  True, False, False, False]])
```

In [229…
```
np.all(filedata > 4,axis = 0) # Go through each columns
```

Out[229…
```
array([False, False, False, False, False, False])
```

In [230…
```
((filedata < 4) & (filedata > 2))
```

Out[230…
```
array([[False, False,  True, False, False, False],
       [False,  True, False, False,  True, False],
       [False, False, False, False, False, False],
       [False, False,  True, False, False, False]])
```

In [231…
```
~((filedata < 4) & (filedata > 2))  # ~ indicate not
```

Out[231…
```
array([[ True,  True, False,  True,  True,  True],
       [ True, False,  True,  True, False,  True],
       [ True,  True,  True,  True,  True,  True],
       [ True,  True, False,  True,  True,  True]])
```

Example

In [232…
```
matrix = np.array([[1,2,3,4,5],[6,7,8,9,10],[11,12,13,14,15],[16,17,18,19,20],[21,2
print(matrix)
```

```
[[ 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]]
```

In [233…
```
matrix[2:4,0:2]
```

Out[233…
```
array([[11, 12],
       [16, 17]])
```

In [234…
```
matrix[[0,1,2,3],[1,2,3,4]]
```

Out[234…
```
array([ 2,  8, 14, 20])
```

In [235…
```
matrix[[0,4,5],3:]
```

Out[235…
```
array([[ 4,  5],
       [24, 25],
       [29, 30]])
```