NumPy tutorial

Outlines

- What is NumPy?
- NumPy installation
- Import NumPy library
- Advantage of NumPy array over a python list
- Basic array operation
- Mathematical operation
- Indexing & Slicing
- Quiz
- Iterating through arrays
- Stacking through two arrays
- Split arrays
- Indexing with boolean arrays
- Iterate numpy array using nditer

What is NumPy?

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

Click here for more information (https://numpy.org/doc/stable/user/whatisnumpy.html)

NumPy's most useful feature is n dimentional array object.

NumPy installation

After installing python, from start type cmd. Then in your cmd type **pip install numpy**

Import NumPy library

In [95]:

```
import numpy as np
```

In [97]:

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

Out[97]:

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

In [98]:

```
a[0]
```

Out[98]:

```
1
```

Advantage of NumPy array over a python list

1. It requires less memory
2. It is fast
3. It is convinient

In [105]:

```
import sys
# comparing size of one element in list & array

l = range(1000)
print(sys.getsizeof(1)*len(l))
array = np.arange(1000)
print(array.size*array.itemsize)
```

```
28000
4000
```

In [108]:

```
import time
SIZE = 1000
l1 = range(SIZE)
l2 = range(SIZE)
start = time.time()
result = [(x+y) for x,y in zip(l1,l2)]
print("python list took: ", (time.time()-start)*10000)
```

```
python list took:  170.0139045715332
```

In [109]:

```
a1 = np.arange(SIZE)
a2 = np.arange(SIZE)
start=time.time()
result = a1 + a2
print("numpy took: ", (time.time()-start)*10000)
```

```
numpy took:  10.001659393310547
```

Basic array operation

In [3]:

```
import numpy as np
# create 2D array
a = np.array ([[1,2],[3,4]])
a
```

Out[3]:

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

In [4]:

```
# print the dimention
a.ndim
```

Out[4]:

```
2
```

In [5]:

```
# print the byte size of each element
a.itemsize
```

Out[5]:

```
4
```

In [6]:

```
a.dtype
```

Out[6]:

```
dtype('int32')
```

In [8]:

```
b = np.array([[1.0,2.0],[3.0,4.0]])
b.itemsize
```

Out[8]:

```
8
```

In [9]:

```
b.dtype
```

Out[9]:

```
dtype('float64')
```

integer numbers = int32 occupied 4 bytes

float numbers = float64 occupied 8 bytes

In [10]:

```
# print the number of elements
a.size
```

Out[10]:

```
4
```

In [12]:

```
# number of rows and columns
a.shape
```

Out[12]:

```
(2, 2)
```

In [13]:

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

Out[13]:

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

```
In [14]: a = np.array ([[1,2],[3,4],[5,6]],dtype=complex)
         a
```

```
Out[14]: array([[1.+0.j, 2.+0.j],
                [3.+0.j, 4.+0.j],
                [5.+0.j, 6.+0.j]])
```

```
In [16]: a = np.zeros((5,6))
         a
```

```
Out[16]: 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.]])
```

```
In [20]: a = np.ones((2,2))
         a
```

```
Out[20]: array([[1., 1.],
                [1., 1.]])
```

```
In [28]: # create l=[0,1,2,3,4]
         l=range(5)
         l
```

```
Out[28]: range(0, 5)
```

```
In [25]: l[0]
```

```
Out[25]: 0
```

```
In [26]: # create array =[0,1,2,3,4]
         a=np.arange(0,5)
         a
```

```
Out[26]: array([0, 1, 2, 3, 4])
```

```
In [30]: # initialize array from 1 to 9 with steps of 2 numbers
         a = np.arange(1,10,2)
         a
```

```
Out[30]: array([1, 3, 5, 7, 9])
```

```
In [31]: # linspace (start,stop,num=...)
         # print 10 numbers between 1 and 5 which are linearly spaced
         a = np.linspace(1,5,10)
         a
```

```
Out[31]: array([1.        , 1.44444444, 1.88888889, 2.33333333, 2.77777778,
                3.22222222, 3.66666667, 4.11111111, 4.55555556, 5.        ])
```

In [34]:

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

Out[34]:

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

In [35]:

```
a.shape
```

Out[35]:

```
(3, 2)
```

In [36]:

```
a.reshape(2,3)
```

Out[36]:

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

In [37]:

```
# make an array to 1 dimention
a.ravel()
```

Out[37]:

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

Mathematical operation

In [40]:

```
a.min()
```

Out[40]:

```
1
```

In [41]:

```
a.max()
```

Out[41]:

```
6
```

In [43]:

```
a
```

Out[43]:

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

In [42]:

```
# sum all of the elements 1+2+3+4+5+6
a.sum()
```

Out[42]:

```
21
```

In [44]:

```
# sum the elements in columns [(1+3+5),(2+4+6)]
a.sum(axis=0)
```

Out[44]:

```
array([ 9, 12])
```

```
In [45]: # sum the elements in rows [(1+2),(3+4),(5+6)]
         a.sum(axis=1)

Out[45]: array([ 3,  7, 11])
```

```
In [48]: b = np.array([[1,4],[9,16]])
         b

Out[48]: array([[ 1,  4],
                [ 9, 16]])
```

```
In [49]: # calculate the square root of b
         np.sqrt(b)

Out[49]: array([[1., 2.],
                [3., 4.]])
```

```
In [50]: # calculate standard deviation of b
         np.std(b)

Out[50]: 5.678908345800274
```

$$mean=\bar{x}=\frac{1+4+9+16}{4}=7.5$$

$$std=\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i-\bar{x})^2}=\sqrt{\frac{(1-7.5)^2+(4-7.5)^2+(9-7.5)^2+(16-7.5)^2}{4}}=5.67$$

```
In [53]: a = np.array([[1,2],[3,4]])
         b = np.array([[5,6],[7,8]])
```

```
In [54]: a+b

Out[54]: array([[ 6,  8],
                [10, 12]])
```

```
In [55]: a*b

Out[55]: array([[ 5, 12],
                [21, 32]])
```

```
In [56]: a/b

Out[56]: array([[0.2       , 0.33333333],
                [0.42857143, 0.5       ]])
```

```
In [59]: # matrix production
         a.dot(b)

Out[59]: array([[19, 22],
                [43, 50]])
```

Indexing & Slicing

```
In [60]: l = [4,5,6,8,10]
         l[0:3]
```

```
Out[60]: [4, 5, 6]
```

```
In [61]: a = np.array([4,5,6,8,10])
         a[0:3]
```

```
Out[61]: array([4, 5, 6])
```

```
In [63]: a = np.array([[6,7,8],[1,2,3],[9,3,2]])
         a
```

```
Out[63]: array([[6, 7, 8],
                [1, 2, 3],
                [9, 3, 2]])
```

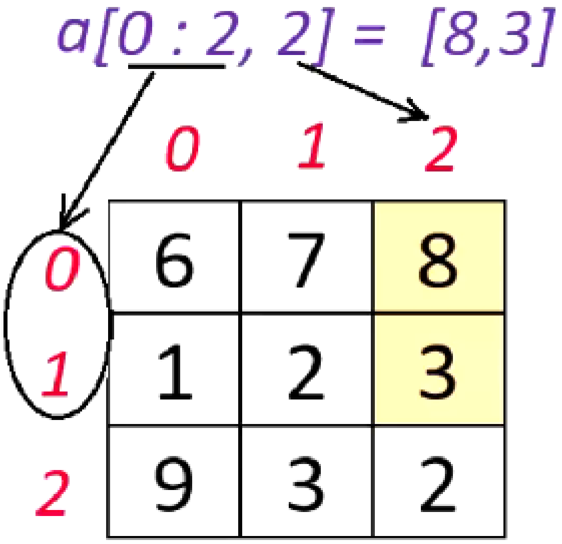
```
In [64]: a[1,2]
```

```
Out[64]: 3
```

a[1,2] row1 column 2 i.e. 3

```
In [65]: a[0:2,2]
```

```
Out[65]: array([8, 3])
```



a[0 : 2, 2] = [8,3]

	0	1	2
0	6	7	8
1	1	2	3
2	9	3	2

```
In [67]: # a [-1] is the last element
         a[-1]
```

```
Out[67]: array([9, 3, 2])
```

Quiz

Print column 1 & 2 ?

click here for solution

```
In [68]: a
```

```
Out[68]: array([[6, 7, 8],
                [1, 2, 3],
                [9, 3, 2]])
```

Iterating through arrays

```
In [69]: for row in a:
             print(row)
```

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

```
In [70]: # print every element in new line
         for cell in a.flat:
             print(cell)
```

```
6
7
8
1
2
3
9
3
2
```

Stacking through two arrays

```
In [71]: a = np.arange(6).reshape(3,2)
         b = np.arange(6,12).reshape(3,2)
```

```
In [72]: a
```

```
Out[72]: array([[0, 1],
                [2, 3],
                [4, 5]])
```

```
In [73]: b
```

```
Out[73]: array([[ 6,  7],
                [ 8,  9],
                [10, 11]])
```

```
In [75]: # vertical stack
         np.vstack((a,b))
```

```
Out[75]: array([[ 0,  1],
                [ 2,  3],
                [ 4,  5],
                [ 6,  7],
                [ 8,  9],
                [10, 11]])
```

```
In [76]: # horizetal stack
         np.hstack((a,b))
```

```
Out[76]: array([[ 0,  1,  6,  7],
                [ 2,  3,  8,  9],
                [ 4,  5, 10, 11]])
```

Split arrays

```
In [77]: a = np.arange(30).reshape(2,15)
         a
```

```
Out[77]: 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]])
```

```
In [80]: # split a into three different arrays (horizental)
         result = np.hsplit(a,3)
```

```
In [81]: result[0]
```

```
Out[81]: array([[ 0,  1,  2,  3,  4],
                [15, 16, 17, 18, 19]])
```

```
In [82]: result[1]
```

```
Out[82]: array([[ 5,  6,  7,  8,  9],
                [20, 21, 22, 23, 24]])
```

```
In [83]: result[2]
```

```
Out[83]: array([[10, 11, 12, 13, 14],
                [25, 26, 27, 28, 29]])
```

```
In [84]: # vertical split
         b = np.vsplit(a,2)
```

In [85]:

```
b[0]
```

Out[85]:

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

In [86]:

```
b[1]
```

Out[86]:

```
array([[15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29]])
```

Indexing with boolean arrays

In [87]:

```
a = np.arange(12).reshape(3,4)
a
```

Out[87]:

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [88]:

```
b = a>4
b
```

Out[88]:

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

In [89]:

```
a[b]
```

Out[89]:

```
array([ 5,  6,  7,  8,  9, 10, 11])
```

a[b] looks into b and wherever it True it return those elements from a

This is a cool way of extracting all the elements which are greater than 4 from your original array.

This is also useful if you want to replace those elements with a cerain numbers.

In [91]:

```
# Replce any element greater than 4 by -1
a[b]=-1
a
```

Out[91]:

```
array([[ 0,  1,  2,  3],
       [ 4, -1, -1, -1],
       [-1, -1, -1, -1]])
```

Iterate numpy array using nditer

In [92]:

```
import numpy as np
```

```
In [93]: a = np.arange(12).reshape(3,4)
         a
```

```
Out[93]: array([[ 0,  1,  2,  3],
                [ 4,  5,  6,  7],
                [ 8,  9, 10, 11]])
```

```
In [94]: for row in a:
             print(row)
```

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

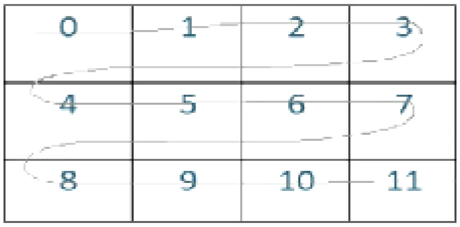
```
In [96]: for cell in a.flatten():
             print(cell)
```

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

```
In [97]: # nditer allows you to do iteration in veriety ways
         for x in np.nditer (a,order='c'):
             print(x)
```

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

C order

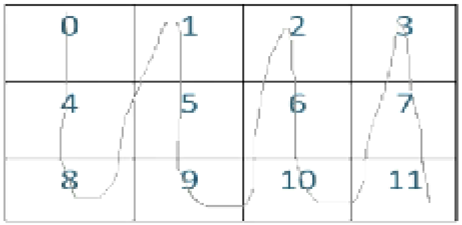


In [98]:

```
for x in np.nditer(a,order='f'):
    print(x)
```

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

Fortran order



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

In [106]:

```
a = np.arange(12).reshape(3,4)
```

In [107]:

```
# print column by column with flags option
for x in np.nditer(a,order='f',flags=['external_loop']):
    print(x)
```

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

In [108]:

```
# modify elements in 2D array and print square of each element
for x in np.nditer(a,op_flags=['readwrite']):
               x[...]=x*x
a
```

Out[108]:

```
array([[  0,   1,   4,   9],
       [ 16,  25,  36,  49],
       [ 64,  81, 100, 121]])
```

In [109]:

```
# How to iterate into two numpy array simultaneously
b = np.arange(3,15,4).reshape(3,1)
b
```

Out[109]:

```
array([[ 3],
       [ 7],
       [11]])
```

In [110]:

```
a
```

Out[110]:

```
array([[  0,   1,   4,   9],
       [ 16,  25,  36,  49],
       [ 64,  81, 100, 121]])
```

In [111]:

```
for x,y in np.nditer([a,b]):
    print(x,y)
```

```
0 3
1 3
4 3
9 3
16 7
25 7
36 7
49 7
64 11
81 11
100 11
121 11
```

When it goes to the first element in a the second value is 3 until the second row in a

To iterate into two numpy array simultaneously, their shape must be compatible:

1. Their dimention are equal, or
2. one of them is 1.

Date	Author
2021-07-22	**Ehsan Zia**