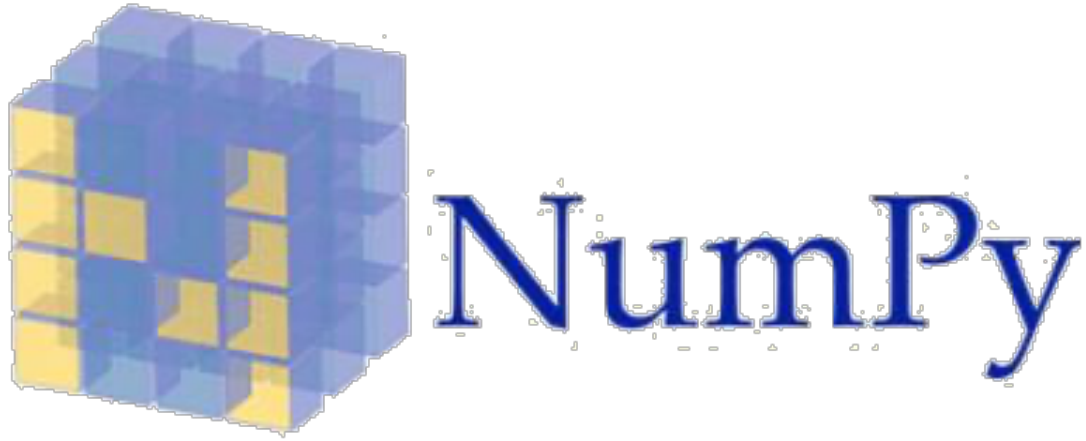
Topic 2

Data Manipulation

Using The Numpy Package



Contents

- Intro to Numpy
- Creating Numpy arrays
- Numpy data types
- Inspecting Numpy arrays
- Manipulating Array Shapes
  - Changing array shape
  - Transpose array
  - Add/ remove elements
  - Combine arrays
  - Split arrays
  - Converting arrays
- Copying Arrays
- Sorting Arrays
- Subsetting, Slicing and Indexing Numpy arrays
- Array Mathematics
  - Arithmetic operations
  - Logical operations
  - Mathematical and Statistical Methods
- File I/O on Numpy arrays

Intro to Numpy

What is Numpy?

- **NumPy** (short for numerical Python) is an open source Python library for scientific computing.
- It lets you work with **arrays** and matrices in a natural way
- Contains a long list of useful mathematical functions, including some functions for linear algebra, Fourier transformation, and random number generation routines.

Why use NumPy?

- NumPy code is much **cleaner** than regular Python code to accomplish the same tasks. Fewer loops required as operations work directly on arrays and matrices
- Has many convenience and mathematical functions that make **coding much easier**
- Underlying algorithms designed with high performance in mind. Large portions of NumPy are written in C. This makes **NumPy faster than pure Python** code
- NumPy's arrays are stored more **efficiently** than an equivalent data structure in base Python, such as a list of lists. The bigger the array, the more it pays off to use NumPy

A simple application

Imagine that we want to add two vectors called a and b

a = [0,1,4,9,16...] # The vector a holds the squares of integers 0 to n

b = [0,1,8,27,64...] # The vector b holds the cubes of integers 0 to n

```
def pythonsum(n):
    a = list(range(n))
    b = list(range(n))
    c = []

    for i in range(len(a)):
        a[i] = i ** 2
        b[i] = i ** 3
        c.append(a[i] + b[i])

    return c
```

Using base Python

```
import numpy as np

def numpysum(n):
    a = np.arange(n) ** 2
    b = np.arange(n) ** 3
    c = a + b
    return c
```

Using NumPy

Numpy Documentation

- Official documentation
  - https://numpy.org/

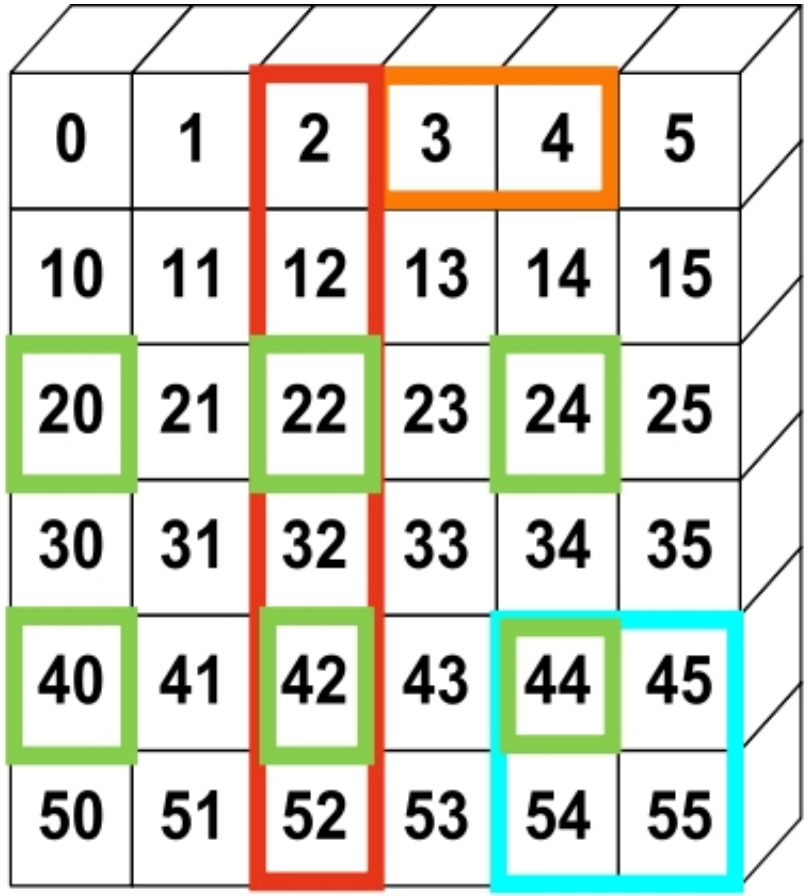
- NumPy Tutorial from W3Schools
  - https://www.w3schools.com/python/numpy_intro.asp

Creating Numpy Arrays

Numpy Array

- NumPy provides an N-dimensional array type, the **ndarray**, which describes a collection of "items" of the same type
- The "type" can be any arbitrary structure of bytes and specified using the data type

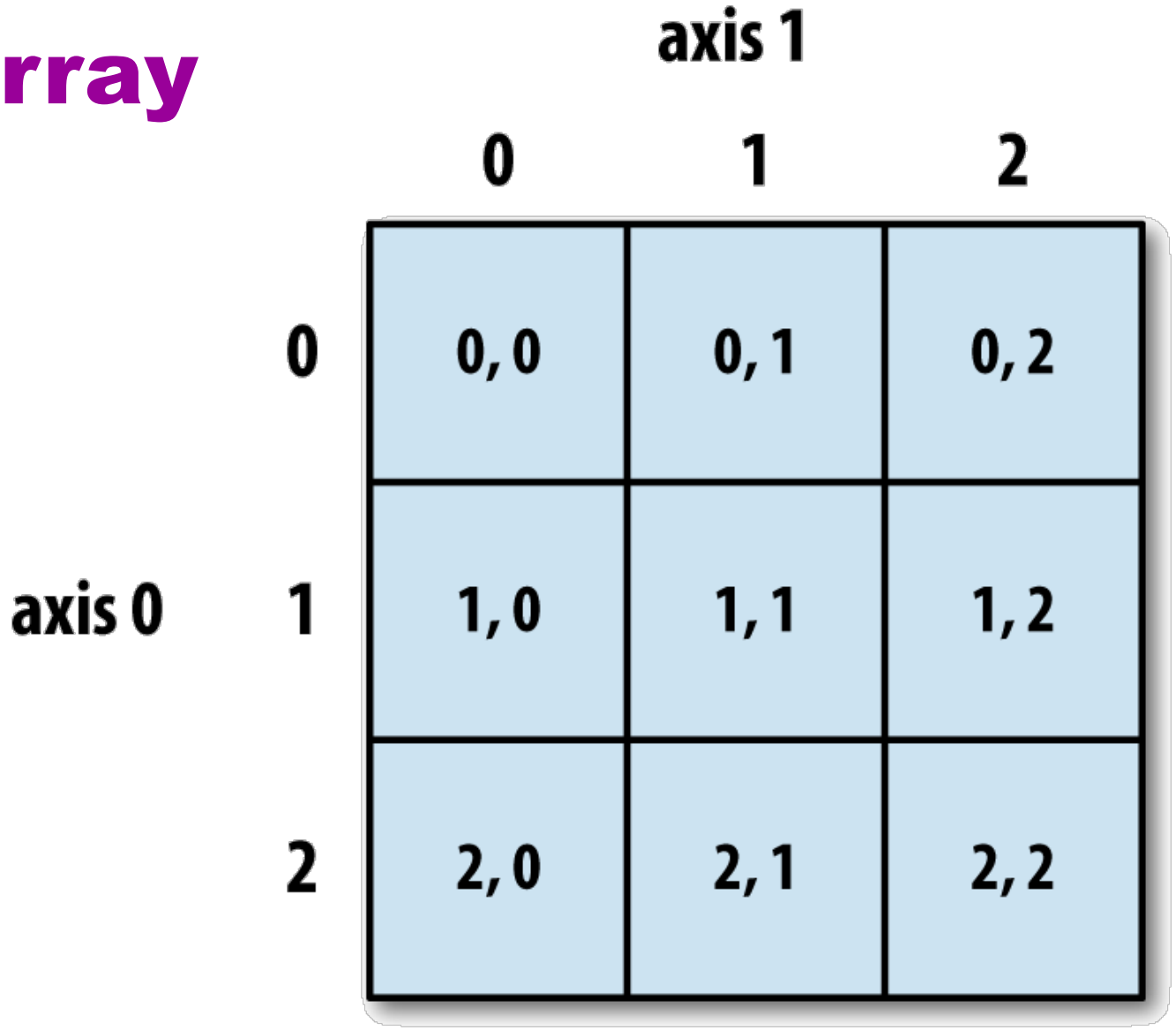
```
import numpy as np
# Create a 2-d array (6x6)
a = np.array([(0,1,2,3,4,5),
              (10,11,12,13,14,15),
               ...
              (50,51,52,53,54,55)])

print(a[:,2])
```



[2,12,22,32,42,52]

Structure of ndarray

- This is an example of a 3x3 ndarray
- The size of the array is 9 (i.e. 9 elements inside) and the dimension is 2 (i.e. 2D array)



Using .array() method

```
import numpy as np

a = np.array([1,2,3]) # Create a 1-d array of integers

b = np.array([(1,2,3),(4,5,6)]) # Create a 2-d array (2x3)

# Create a 1-d array of strings
c = np.array(['Apple', 'Orange', 'Pear', 'Durian'])


# Create a 2-d array (2x4) of type float
d = np.array([(1.5,2.5,3.5,4.5),(5.5,6.5,7.5,8.5)],dtype=float)
```

Using zeros() and ones()

```
import numpy as np

# Create a 3x4 2-d array, with initial value zero
a = np.zeros((3,4))
print(a)


# Create a 2x3x4 3-d array with initial value one
b = np.ones((2,3,4))
print(b)
```

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

Using arange()

```
import numpy as np

# numpy.arange(start,stop,step)

# Create an array that starts from zero and ends at 8
a = np.arange(0,9)
print(a)

# Create array that starts from 10 and ends at 19 with interval 2
c = np.arange(10,20,2)
print(c)
```

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

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

Using arange()

```
import numpy as np

# Create an array that contains the dates in month of Aug 2017
d = np.arange('2017-08-01','2017-09-01',dtype='datetime64')
print(d)
```

```
['2017-08-01' '2017-08-02' '2017-08-03' '2017-08-04' '2017-08-05'
 '2017-08-06' '2017-08-07' '2017-08-08' '2017-08-09' '2017-08-10'
 '2017-08-11' '2017-08-12' '2017-08-13' '2017-08-14' '2017-08-15'
 '2017-08-16' '2017-08-17' '2017-08-18' '2017-08-19' '2017-08-20'
 '2017-08-21' '2017-08-22' '2017-08-23' '2017-08-24' '2017-08-25'
 '2017-08-26' '2017-08-27' '2017-08-28' '2017-08-29' '2017-08-30'
 '2017-08-31']
```

Using linspace()

```
import numpy as np

# numpy.linspace(start,stop,num)

# Create array that starts with 0 and ends at 2
# with 9 samples in between
d = np.linspace(0,2,9)
print(d)
```

```
[ 0.    0.25  0.5   0.75  1.    1.25  1.5   1.75  2.  ]
```

Using full() and eye()

```
import numpy as np

# Create a constant array with a specified value
# numpy.full(shape, fill_value)
h = 7.0
e = np.full((2,2),h)
print(e)

# Create a 2x2 identity matrix
f = np.eye(2)
print(f)
```

```
[[ 7.  7.]
 [ 7.  7.]]
```

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

Using random() and randint()

```
import numpy as np

# Create a 2x2 array with random floats in the interval 0.0 to 1.0
g = np.random.random((2,2))
print(g)
```

```
 [[ 0.2535411   0.46228199]
 [ 0.54930425  0.65506307]]
```

```
# Create a 3x2x4 array with random numbers
# between 10 and 50 (not including 50)
h = np.random.randint(10,50,(3,2,4))
print(h)
```

```
[[[21 15 21 37]
  [47 14 16 45]]

 [[27 20 48 18]
  [45 36 43 17]]

 [[33 43 17 24]
  [35 10 11 18]]]
```

Using empty()

```
import numpy as np


# Create a 3x2  empty array
h = np.empty((3,2))
print(h)
```

```
[[  6.23042070e-307   1.89146896e-307]
 [  1.37961302e-306   1.05699242e-307]
 [  3.11523242e-307   1.44025809e+214]]
```

Using reshape()

```
import numpy as np

# Create 20 numbers from 1 to 20 as a 1-d array
# then reshape this to a 2d array of shape 4x5
a = np.arange(1,21).reshape(4,5)
print(a)
```

```
[[ 1  2  3  4  5]
 [ 6  7  8  9 10]
 [11 12 13 14 15]
 [16 17 18 19 20]]
```

Put different ndarrays into 1 single ndarray

```
import numpy as np

c1 = np.array([1,2,3,4,5])
c2 = np.arange(2,21,2)
c3 = np.random.randint(1,100,10)

c = np.array([c1,c2,c3])
```

```
[array([1, 2, 3, 4, 5]) array([ 2,  4,  6,  8, 10, 12, 14, 16, 18, 20])
 array([45, 75, 82, 80, 68,  6, 66, 71, 88, 89])]
```

Practical 2 Section 3

Numpy Data Types

Numpy Data Types

np.int32	32-bit integer
np.int64	64-bit integer
np.float64	64-bit decimal number
np.bool	Boolean type storing TRUE and FALSE values
np.object	Python object type
np.unicode	Fixed-length Unicode string type

Numpy Data Types

```
import numpy as np

a = np.arange(10)
print(f'type(a) is {type(a)}')
print(f'a.dtype is {a.dtype}')
```

```
type(a) is <class 'numpy.ndarray'>
a.dtype is int32
```

Numpy Data Types

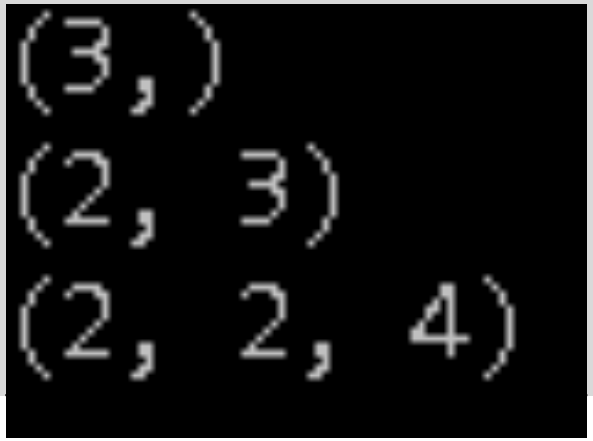
```
import numpy as np
import math

a = np.array([math.pi],dtype=np.int32)

b = np.array([math.pi],dtype=np.float64)

c = np.array([math.pi],dtype=np.bool)

d = np.array([math.pi],dtype=np.object)

e = np.array([math.pi],dtype=np.unicode)
```

```
math.pi 3.141592653589793
[3] int32
[ 3.14159265] float64
[ True] bool
[3.141592653589793] object
['3.141592653589793'] <U17
```

Inspecting your array

shape

```
import numpy as np

# Create 3 arrays with different shapes
a = np.array([1,2,3])
b = np.array([(1,2,3),(4,5,6)])
c = np.array([[(1,2,3,4),(5,6,7,8)], [(9,10,11,12),(13,14,15,16)]])

#shape returns the number of elements in each dimension
print(a.shape)

print(b.shape)

print(c.shape)
```



len, ndim, size

```
import numpy as np
# Create 3 arrays with different shapes and sizes
a = np.array([1,2,3,4,5,6])
b = np.array([(1,2,3),(4,5,6)])
c = np.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)]])

# Size of the first dimension
print('====len====')
print(len(a));print(len(b));print(len(c))

# Number of dimensions
print('====ndim====')
print(a.ndim);print(b.ndim);print(c.ndim)

# Total number of elements across dimensions
print('====size====')
print(a.size);print(b.size); print(c.size)
```

```
====len====
6
2
3
====ndim====
1
2
3
====size====
6
6
24
```

Practical 2 Section 4

Functions to manipulate array shapes

CHANGE ARRAY SHAPE
flatten()
reshape()
shape()
resize()

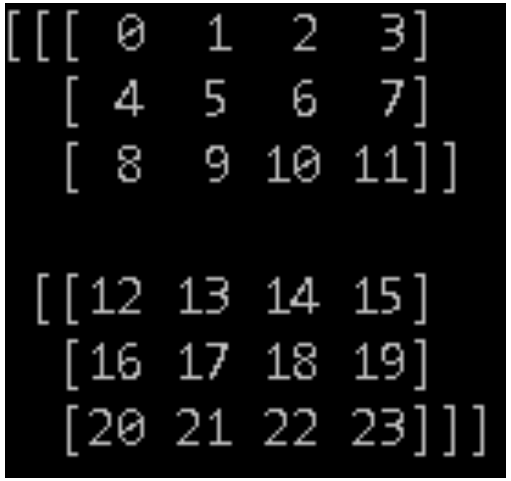
COMBINE ARRAYS
concatenate

ADD/REMOVE ELEMENTS
append()
insert()
delete()

TRANSPOSE ARRAY
transpose

Change array shape - flatten()

- Use this function to convert your M-D array to a 1-D array
- Return value is a copy of the original array

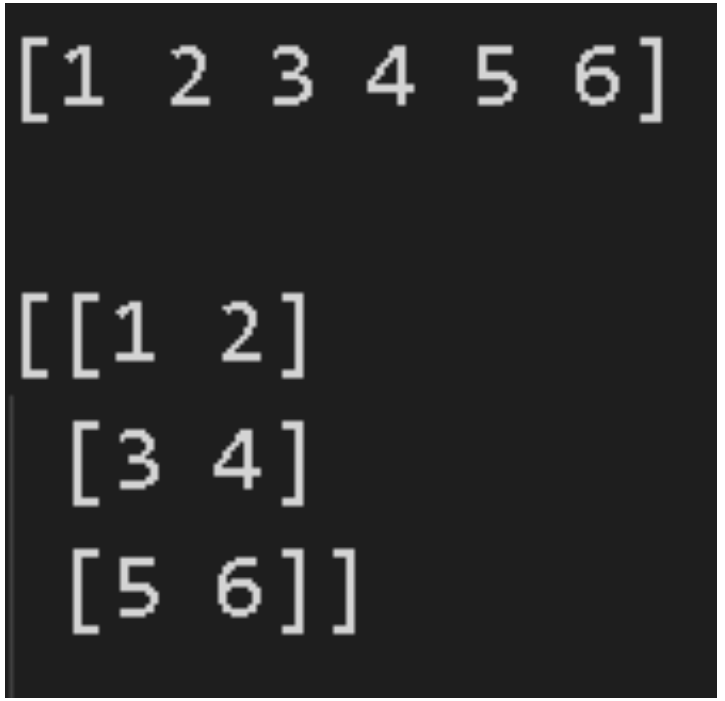



```
import numpy as np

b = np.arange(24).reshape(2,3,4) # b= [[[0,1,2,3],[4,5,6,7],[8..],[20,21,22,23]]]

c = b.flatten()

print(b)
print(c) #[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23]

c[0] = 100;  # change the first value in the copy
print(b)
print(c) #[ 100  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23]
```

Change array shape - reshape()

- reshape gives a new shape to an array without changing its data. It creates a new array and does not modify the original array itself

```
import numpy

my_array =
numpy.array([1,2,3,4,5,6])
print(my_array)
print()
print(numpy.reshape(my_array,
(3,2)))
```



shape

- The **shape** attribute an be used to get array dimensions or to change array dimensions

```
import numpy as np

a= np.array([1, 2, 3, 4, 5])
print(a.shape)     #(5,) -> 5 rows and 0
columns

b = np.array([[1, 2],[3, 4],[6,5]])
print(b.shape) #(3, 2) -> 3 rows and 2
columns
```

transpose()

- The transpose of an array can be obtained by using transpose() method
- transpose() is both a library level function and an instance method.
- It can be called as **numpy.transpose(ndarray)** or **numpy.ndarray.transpose()**.
- ndarray has an attribute named 'T', which returns the transpose of the array.

```
import numpy as np

x = np.array(([10,20,30,40], [50,60,70,80], [90,
85, 75, 45]))
print(x)
print(x.transpose())
print(np.transpose(x))
print(x.T)
```

```
[[10 20 30 40]
 [50 60 70 80]
 [90 85 75 45]]

[[10 50 90]
 [20 60 85]
 [30 70 75]
 [40 80 45]]
```

resize()

- The **resize()** method works just like the reshape() function, but modifies the array it operates on:

```
import numpy as np

a=np.array([[0,1],[2,3]]) #2x2

a = np.resize(a,(4,1))

a = np.resize(a,(2,3)) # not allow in reshape
```

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

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

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

concatenate() - 1-d arrays

- Two or more arrays can be concatenated using concatenate() function along an axis
- The arrays must have the same shape, except in the dimension corresponding to axis (the first, by default).

numpy.concatenate((a1, a2, ...), axis=0)

```
import numpy as np

# 1-D array
x = np.arange(5)
y = np.arange(6,10)
z = np.arange(11,15)

print(x);print(y); print(z)
print(np.concatenate((x,y,z)))
```

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

concatenate()
2-d arrays on axis=0

```
import numpy as np

# Reshaped to 2-D
x = np.arange(1,5).reshape(2,2)
y = np.arange(6,12).reshape(3,2)
z = np.arange(8,16).reshape(4,2)

print(x)
print(y)
print(z)
print(np.concatenate((x,y,z)))
```

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

```
[[ 6  7]
 [ 8  9]
 [10 11]]
```

```
[[ 8  9]
 [10 11]
 [12 13]
 [14 15]]
```

```
[[ 1  2]
 [ 3  4]
 [ 6  7]
 [ 8  9]
 [10 11]
 [ 8  9]
 [10 11]
 [12 13]
 [14 15]]
```

concatenate() - 2-d arrays on axis=1

```
import numpy as np

x = np.arange(1,5).reshape(2,2)
y = np.arange(6,12).reshape(2,3)
z = np.arange(8,16).reshape(2,4)

print(x)
print(y)
print(z)
print(np.concatenate((x,y,z),axis=1))
```

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

[[ 6  7  8]
 [ 9 10 11]]

[[ 8  9 10 11]
 [12 13 14 15]]

[[ 1  2  6  7  8  8  9 10 11]
 [ 3  4  9 10 11 12 13 14 15]]
```

append()

- Append values to the end of an array
- values must be of the correct shape (the same shape as arr, excluding axis)
- If axis is not specified, values can be any shape and will be flattened before use.

numpy.append(arr, values, axis=None)

```
import numpy as np

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

x1 = np.append(x, np.array([(7,8,9)]),axis = 0)
x2 = np.append(x, np.array([(7,8), (9,10)]),axis =

print(x1);print()
print(x2);print()
```

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

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

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

Functions to split arrays

split()	This function splits one-dimensional arrays as columns to create a two-dimensional array
hsplit()	This function splits arrays horizontally
vsplit()	This function splits arrays vertically

split

numpy.split(array, indices_or_sections, axis=0)

- Split an array into multiple sub-arrays.

```
import numpy as np

x = np.arange(9.0)
y = np.split(x, 3)

print(x)
print(y)
```

```
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.]
```

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

hsplit

- Split an array into multiple sub-arrays horizontally (column-wise)
- Equivalent to split with axis=1, the array is always split along the second axis regardless of the array dimension

```
[[  0.   1.   2.   3.]
 [  4.   5.   6.   7.]
 [  8.   9.  10.  11.]
 [ 12.  13.  14.  15.]]
```

```
import numpy as np

x = np.arange(16.0).reshape(4, 4)
y = np.hsplit(x, 2)

print(x)
print(y)
```

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

vsplit

- Split an array into multiple sub-arrays vertically (row-wise)
- equivalent to split with axis=0 (default), the array is always split along the first axis regardless of the array dimension

```
[[  0.   1.   2.   3.]
 [  4.   5.   6.   7.]
 [  8.   9.  10.  11.]
 [ 12.  13.  14.  15.]]
```

```
import numpy as np

x = np.arange(16.0).reshape(4, 4)
y = np.vsplit(x, 2)

print(x)
print(y)
```

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

Copying arrays

numpy.copy

```
import numpy as np

# Create 2 arrays with different data types, shapes and sizes
a = np.array([10,2,8,4,6,1,5,9,3,7])
b = np.array([("Red","Blue","Yellow"),("Green","Cyan","Magenta")])

npc = np.copy(a)

npc.sort()

print(a)
print(npc)
```

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

Sorting arrays

Sorting arrays

```
import numpy as np

# Create 2 arrays with different data types, shapes and sizes
a = np.array([10,2,8,4,6,1,5,9,3,7])
b = np.array([("Red","Blue","Yellow"),("Green","Cyan","Magenta")])

a.sort()
b.sort(axis=1)#column

print(a)
print(b)
```

```
[ 1  2  3  4  5  6  7  8  9 10]
[['Blue' 'Red' 'Yellow']
 ['Cyan' 'Green' 'Magenta']]
```

Subsetting, Slicing and Indexing

One-dimensional slicing and indexing

```
import numpy as np

a = np.arange(9) # a = [0,1,2,3,4,5,6,7,8]

# Select the element at index 2
print(a[2]) # 2

# Select elements from index 0 to 7 with step 2
print(a[:7:2]) # 0 2 4 6

# Select and reverse elements from index 0 to the end
print(a[::-1])    # 8 7 6 5 4 3 2 1
```

Multi-dimensional Slicing and indexing 1

```
import numpy as np

# Create an array with 0 to 23 and reshape it into a 2x3x4 array
# Think of it as an Excel workbook with 2 worksheets
# Each worksheet has 3 rows,4 columns
b = np.arange(24).reshape(2,3,4)
print(b)
```

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

 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]
```

Multi-dimensional Slicing and indexing 1

```
# Select worksheet 2, Row 2, Col 4
print(b[1,1,3])  # 19

# Select both worksheets, Row 1, Col 1
print(b[:,0,0])  # 0 12

# Select first worksheet, all rows and columns
# Method 1
print(b[0])
# Method 2
print(b[0, :, :])
# Method 3, Ellipsis replaces the multiple colons
print(b[0, ...])
```

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

 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]
```

Multi-dimensional Slicing and indexing 2

```
# Select first worksheet, second row, all columns
print(b[0,1])     # 4 5 6 7

# Select first worksheet, all rows, second column
print(b[0,:,1])    # 1 5 9

# Select every 2 elements in first worksheet, 2nd row, all cols
print(b[0,1,::2])  # 4 6

# Select all worksheets, all rows, 2nd col
print(b[...,1])  # 1 5 9 13 17 21

# Select 2nd row elements, regardless of worksheet and col
print(b[:,1,:])    # 4 5 6 7 16 17 18 19

# Select 1st worksheet, all rows, last column
print(b[0,:,-1])  #3 7 11
```

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

 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]
```

Multi-dimensional Slicing and indexing 3

```
# Select first worksheet, all rows, last col, but in reverse
print(b[0,::-1, -1])  # 11 7 3

# Select 1st worksheet, last col, every 2 rows
print(b[0,::2,-1])  # 3 11

# Select all worksheets in reverse, all rows, all col
print(b[::-1])
```

```
[[[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]

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

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

 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]
```

Boolean Indexing

- Boolean indexing lets you indicate if an element should be included
- In the example below, we want to create an array with only the even numbers

```
import numpy as np

a = np.arange(6).reshape(2,3)
b = a % 2 == 0
a = a[b]

print(a)
```

Arithmetic operators

- The standard arithmetic operators such as: +, -, *, /, **, % are applied on individual elements, so, the arrays have to be of the same size

```
import numpy as np

x = np.array(([10,20,30], [40,50,60]))
y = np.array(([1,2,3], [4,5,6]))

print(x)
print(y)

print(x+y)
print(x-y)
print(x*y)
print(x/y)
print(x%y)
```

```
x: [[10 20 30]
 [40 50 60]]
y: [[1 2 3]
 [4 5 6]]
```

Logical operators

- Similarly, logical operators >, < , == are applied on individual elements, so arrays have to be of same size

```
import numpy as np

x = np.array(([10,20,30], [40,50,60]))
y = np.array(([1,2,3], [4,5,6]))

print(x)
print(y)

print(x>y)
print(x<y)
print(x==y)
```

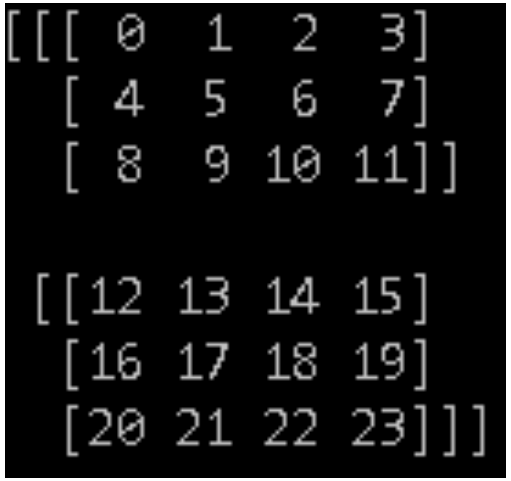
```
x: [[10 20 30]
 [40 50 60]]
y: [[1 2 3]
 [4 5 6]]
```

Mathematical and Statistical Methods

sum()	Sum of all the elements in the array or along an axis. Zero-length arrays have sum 0.
mean()	Arithmetic mean. Zero-length arrays have NaN mean.
median()	
cumsum()	Return the cumulative sum of the elements along a given axis.
cumprod()	This function stacks one-dimensional arrays as columns to create a two-dimensional array
std, var	Returns the standard deviation, a measure of the spread of a distribution, of the array elements Returns the variance of the array elements, a measure of the spread of a distribution
min,max	Return the minimum, maximum along a given axis
argmin, argmax	Return indices of the minimum, maximum values along the given axis

sum()

- Sum of all the elements in the array or along an axis
- Zero-length arrays have sum 0

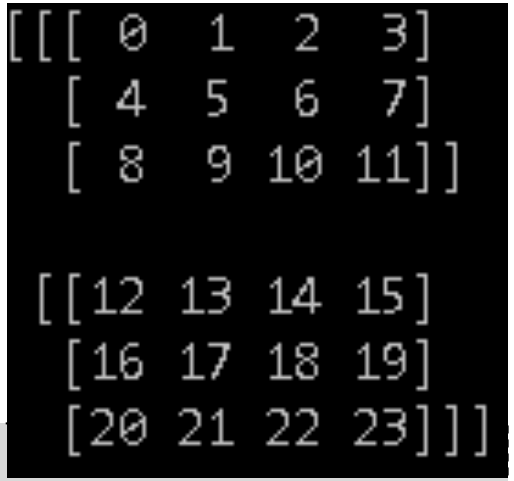


numpy.sum(*a*, *axis=None*, *dtype=None*, *out=None*, *keepdims=<class numpy._globals._NoValue>*)

```
import numpy as np

b = np.arange(24).reshape(6,4) # b= [[[0,1,2,3],[4,5,6,7],[8..],[20,21,22,23]]]

print(b.sum()) # 276
```

mean()

- Sum of all the elements in the array or along an axis
- Zero-length arrays have mean 0




```
import numpy as np

b = np.arange(24) # b= [0,1,2,3,4,... 23]

print(b.mean()) # 11.5
```

median()

numpy.median(a, axis=None, out=None, overwrite_input=False, keepdims=False)

- Compute the median along the specified axis.
- Returns the median of the array elements

```
import numpy as np

a = np.array([[10, 7, 4], [3,
2, 1]])

median = np.median(a)

print(median)
```

```
Median: 3.5
```

min(), max()

- ndarray.min(axis=None, out=None, keepdims=False)
- ndarray.max(axis=None, out=None, keepdims=False)

```
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]
 [12 13 14]
 [15 16 17]
 [18 19 20]
 [21 22 23]]
```

```
import numpy as np

b = np.arange(24).reshape(8,3)

print(b.min())
print(b.min(axis=0))
print(b.min(axis=1))

print(b.max())
print(b.max(axis=0))
print(b.max(axis=1))
```

```
[0 1 2]
[ 0  3  6  9 12 15 18 21]
```

```
[21 22 23]
[ 2  5  8 11 14 17 20 23]
```

argmin(), argmax()

- numpy.argmin(a, axis=None, out=None)
- numpy.argmax(a,axis=None, out=None)

```
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]
 [12 13 14]
 [15 16 17]
 [18 19 20]
 [21 22 23]]
```

```
import numpy as np

b = np.arange(24).reshape(8,3)
print(b)

print(b.argmin())
print(b.argmin(axis=0))
print(b.argmin(axis=1))

print(b.argmax())
print(b.argmax(axis=0))
print(b.argmax(axis=1))
```

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

```
23
[7 7 7]
[2 2 2 2 2 2 2 2]
```

cumsum()

- Return the cumulative sum of the elements along a given axis
- numpy.cumsum(a, axis=None, dtype=None, out=None)

```
import numpy as np

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

b = np.cumsum(a)
print(b)   # [1 3 6 10 15 21]

c = np.arange(1,10).reshape(3,3)
print(c)

d = np.cumsum(c,axis=0)
print(d)
```

```
[[1 2 3]
 [4 5 6]]
[ 1  3  6 10 15 21]
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[[ 1  2  3]
 [ 5  7  9]
 [12 15 18]]
```

cumprod()

- Return the cumulative product of the elements along a given axis
- numpy.cumprod(a, axis=None, dtype=None, out=None)

```
import numpy as np

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

b = np.cumprod(a)
print(b)  #[1 2 6 24 120 720]

c = np.arange(1,10).reshape(3,3)
print(c)

d = np.cumprod(c,axis=1)
print(d)
```

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

[  1   2   6  24 120 720]
```

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

[[  1   2   6]
 [  4  20 120]
 [  7  56 504]]
```

std()

- Compute the standard deviation along the specified axis
- numpy.cumprod(a, axis=None, dtype=None, out=None)

```
import numpy as np

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

b = np.std(a)
print(b)
```

var()

- Compute the variance along the specified axis
- numpy.var(a, axis=None, dtype=None, out=None, ddof=0)

```
import numpy as np

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

print(np.var(a, axis=0))
print(np.var(a, axis=1))

print(np.var(a, axis=1,dtype=np.float32))
print(np.var(a, axis=1,dtype=np.float64))
```

File Input and Output with Arrays

- NumPy is able to save and load data in both text and binary format

Function	Description
load()	Load a Numpy array from disk
save()	Save a Numpy array to a binary file
savez()	Save several arrays into a single file in uncompressed .npz format.

Function	Description
loadtxt(filename)	Load data from a text file. Each row in the text file must have the same number of values.
genfromtxt(filename)	Load data from a text file, with missing values handled as specified.
savetxt(filename)	Save an array to a text file.

Using genfromtxt()

```
import numpy as np


data = np.genfromtxt("data/coe-results.csv",
            delimiter=',',
          names=True, dtype=('U7', int, 'U10', int, int, int, int))


print(data)
print(data.shape)
```

The End