
th
100th Anniversary of
Chula Engineering 2013

11 Numpy


python
powered

Numpy

- Core library for scientific computing in Python
- Simple, concise, and very fast
- https://numpy.org/
- Not a built-in library. We need to install it ourselves.

```
In [1]: !pip install numpy
```

Numpy Array

Array: keeps data in rows, similar to list but with the following differences:

- All data elements have the same data type (normally numbers)
- Can keep data with various dimensions

  1D : vector [1, 2, 3, 4]

  2D : matrix $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

  3D : tensor $\left[\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}\right]$
- Accessible using tuple as index, e.g., if `a` is 2D array, then we can access `a` using `a[(1,2)]` or `a[1,2]`
- Numpy has many useful operators and methods

List vs. Numpy Array: all pair distances


```
def all_pair_distances(points):
    # points are nested list, e.g., [[0,0],[0,3],[4,0]]
    n = len(points)
    D = [[0.0] * n for i in range(n)]
    for i in range(n):
        for j in range(i+1, n):
            dx = points[i][0] - points[j][0]
            dy = points[i][1] - points[j][1]
            D[i][j] = D[j][i] = (dx**2+dy**2)**0.5
    return D
```

List

```
def all_pair_distances(points):
    # points are array
    n = len(points)
    X = points[:, 0]
    Y = points[:, 1]
    dx = X - X.reshape(n, 1)
    dy = Y - Y.reshape(n, 1)
    D = (dx**2+dy**2)**0.5
    return D
```

Numpy Array

More understandable
Much faster

python powered

Numpy Array in 2190101 (briefly)

- Array constructions
- Indexing
- Element-wise operations
- Broadcasting
- Some widely used functions

(sum, min, max, argmin, argmax, mean, std, dot)

Array constructions

100th Anniversary of Chula Engineering 2013

```
import numpy as np

a = np.array([1,2,3,4])      # constructed from list
b = np.array([[1,2],[3,4]])   # constructed from list
c = np.ndarray((2,3))        # constructed by size
d = np.ndarray((2,3), int)
e = np.zeros((2,3), int)     # constructed by size with all 0
f = np.ones((2,3), int)      # constructed by size with all 1
g = np.zeros_like(f, float) # constructed as f with all 0 float
h = np.ones_like(e, float)  # constructed as e with all 1 float
I = np.identity(4, int)      # constructed an identity matrix
                             # with size 4 x 4
x = np.arange(0.0,1.0,0.1) # [0.0,0.1,…, 1.0]
```


python powered

array.shape

- `array.shape` returns tuple about size of each dimension
  - `a = np.ones((3,4))` will has shape as `(3,4)`
  - `a.shape[0]` is `3` => number of rows
  - `a.shape[1]` is `4` => number of columns
- `len(array.shape)` returns number of dimensions
  - `a = np.ones((3,4))`
  - `len(a.shape)` is `2`

array.reshape (newshape)


```
a = np.arange(8)          # [0 1 2 3 4 5 6 7]
b = a.reshape((2,4))      # [[0 1 2 3],
                          #  [4 5 6 7]]
c = b.reshape((4,2))      # [[0 1],
                          #  [2 3],
                          #  [4 5],
                          #  [6 7]]
d = c.reshape(8)          # [0 1 2 3 4 5 6 7]
d = c.reshape((2,3))      # ERROR
```

array.T

- `a.T` is transpose of array `a`
- If `a` is 1D, `a.T` will be the same as `a`
- If `a` is 2D, `a.T` will be the transpose of matrix `a`

```
a = np.arange(8)
print(a)
print(a.T)
b = a.reshape((2,4))
print(b)
print(b.T)
c = a.reshape((1,8))
print(c)
print(c.T)
```

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

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

Indexing

Use tuple to specify a specific position in an array

```
import numpy as np

def count_ones(A):
    c = 0
    for i in range(A.shape[0]):
        for j in range(A.shape[1]):
            if A[i,j] == 1:
                c += 1
    return c
```

We can use `A[i,j]` or `A[(i,j)]`

To count 1 in array A, we can just call `np.sum(A==1)`

Slicing: start: stop: step

100th Anniversary of Chula Engineering 2013

- Format: A[select row, select column]
- Beware!!: A[select row][here is NOT select column]

```
a = [[1,2,3],[4,5,6],[7,8,9],[10,11,12]]
print(a[::2])          # [[1,2,3],[7,8,9]]
print(a[::2][::2])     # [[1,2,3]]
A = np.array(a)
print(A[::2])          # [[1 2 3]
                       #  [7 8 9]]
print(A[::2][::2])     # [[1 2 3]]
print(A[::2,::2])      # [[1 3]
select even row select even column
                       #  [7 9]]
print(A[::-1,::-1])    # [[12 11 10]
                       #  [ 9  8  7]
                       #  [ 6  5  4]
                       #  [ 3  2  1]]
```


python powered

Fancy Indexing


```
# a = [0,10,20,30,40,50,60,70,80,90]
a = np.arange(0, 100, 10)
b = a[0::2]                          # b = [0 20 40 60 80]
c = a[[8, 1, 9, 0]]                  # c = [80 10 90 0]
d = c[[True,False,False,True]]  # d = [80 0]

A = np.array([[1,2,3],[4,5,6],[7,8,9],[0,1,0]])
B = A[[1,3,2], [2,0,1]]

#
#    A[1,2], A[3,0], A[2,1]    B = [6 0 8]
```

Practice#1


```
# A is a 2D array
def get_column_from_bottom_to_top(A, c):
    return_____________________________  # one line
def get_odd_rows(A):
    return_____________________________  # one line
def get_even_rows_last_column(A):
    return_____________________________  # one line
def get_diagonal1(A): # A is a square matrix
    ___________________________________
    return_____________________________  # two lines
def get_diagonal2(A): # A is a square matrix
    ___________________________________
    return_____________________________ # two lines
```

Assign scalar value into array


```
A = np.zeros(8)
A[2:5] = 9
```

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

```
A = np.ndarray((4,4), int)
A[:,:] = 9
```

$$\begin{bmatrix} 9 & 9 & 9 & 9 \\ 9 & 9 & 9 & 9 \\ 9 & 9 & 9 & 9 \\ 9 & 9 & 9 & 9 \end{bmatrix}$$

Assign 9 to all rows and columns

```
B = np.zeros((4,4), int)
B[:,0::2] = 1
B[1::2,:] = 2
```

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 2 & 2 & 2 & 2 \\ 1 & 0 & 1 & 0 \\ 2 & 2 & 2 & 2 \end{bmatrix}$$

Assign 1 into the even columns of all rows
Assign 2 into all columns of the odd rows

Calculation using Array data with scalar

Calculated as element-wise and return the result as array

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

b = a + 1              # [ 2   3   4   5   6]
c = a**2 + 1           # [ 2   5   10  17  26]
d = a/2                # [0.5  1.0  1.5  2.0  2.5]
```

```
def toCM(inches):
    return inches * 2.54

d = np.array([0, 10, 12, 100])
print(toCM(d))
```

```
[0.   25.4   30.48   254.]
```

Several math functions in numpy

100th Anniversary of Chula Engineering 2013

```
a = np.array([10,100,1000,10000])
b = np.log10(a)     # [1. 2. 3. 4.]

c = np.array([np.pi, 2*np.pi, 3*np.pi])
d = np.sin(c/2)

# [1.0000000e+00 1.2246468e-16 -1.0000000e+00]
```


python powered

Array and scalar comparison

Compared as element-wise and return the result as array of True/False

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

b = a > 3              # [False False False True]
c = a%2==1             # [True False True False]
```

Count odd numbers in an array

```
def count_odds(a):
    return sum(a%2==1)
def get_odds(a):
    return a[a%2==1] # select only True elements
def get_odd_positions(a):
    pos = np.arange(a.shape[0])
    return pos[a%2==1]
```

In Python
True is 1
False is 0

Practice#2


```
def toCelsius(f):
    # f = [temperature in Fahrenheit, …]


def BMI(wh):
    # [[w1,h1],[w2,h2], …]


def distanceTo(P, p):
    # distance from p to al points in P
```

Practice#3: Logistic Regression

The probability p(x) that a student x will pass the subject, based on the number of exercises he/she practiced and his/her GPAX is calculated as follows:

$$p(x) = \frac{1}{1 + e^{-logit(x)}}$$

$$logit(x) = -3.98 + 0.1x_o + 0.5\,x_1$$

You task: write a python program to read the number of done exercises and GPAX of a set of students and print out the p(x) for each student.

Element-wise Operations


```
x = [1,2,3]
y = [4,5,6]
z = x + y  # concatenation [1,2,3,4,5,6]
```

```
u = np.array([1,2,3])
v = np.array([4,5,6])
w = u + v  # element-wise addition
           # [1+4,2+5,3+6] = [5,7,9]

A = np.array([[1,2,3],[4,5,6],[7,8,9]])
I = np.identity(A.shape[0], int)
B = I*A    # element-wise multiplication
```

$$\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 9\end{bmatrix} * \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix} = \begin{bmatrix}1 & 0 & 0\\ 0 & 5 & 0\\ 0 & 0 & 9\end{bmatrix}$$

Element-wise Logical Operators

~~[True,True,False,False] and [False,True,True,False]~~
~~[True,True,False,False] or [False,True,True,False]~~
~~not [False,True,True,False]~~

```
[True,True,False,False] & [False,True,True,False]
[True,True,False,False] | [False,True,True,False]
                ~ [False,True,True,False]
```

We must use &,|,~ for the element-wise logical operation

python powered

Element-wise Logical Operators

100th Anniversary of Chula Engineering 2013

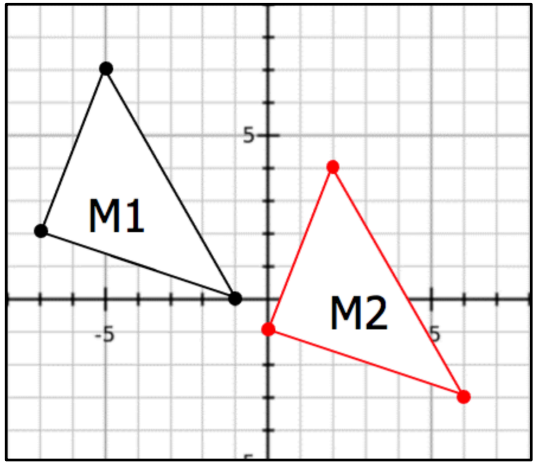
```
a = np.array([9,3,0,2,6])
b = a[a<5]
b = a[[False,True,True,True,False]]
    #[          3,   0,   2       ]


b = a[2 < a < 5]            # ERROR
b = a[2 < a and a < 5]      # ERROR
b = a[2 < a & a < 5]        # ERROR
b = a[(2 < a) & (a < 5)]    # OK
   #a[[T,T,F,F,T] & [F,T,T,T,F]]
   #a[[F,T,F,F,F]]
   # [3]
```


python powered

Example: Matrix Translation


```
     x   y
[[-7   2]
 [-5   7]
 [-1   0]]
```




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

We would like to move all points to the right 7 steps and to downward 3 steps. Hence, we need to add all points with [7, -3]

```
M1 = np.array([[-7,2],[-5,7],[-1,0]])
T  = np.array([[7,-3],[7,-3],[7,-3]])
M2 = M1+T
```

```
[[-7   2]      [[7  -3]      [[0  -1]
 [-5   7]   +   [7  -3]   =>  [2   4]
 [-1   0]]      [7  -3]]      [6  -3]]
```


python powered

Practice#4

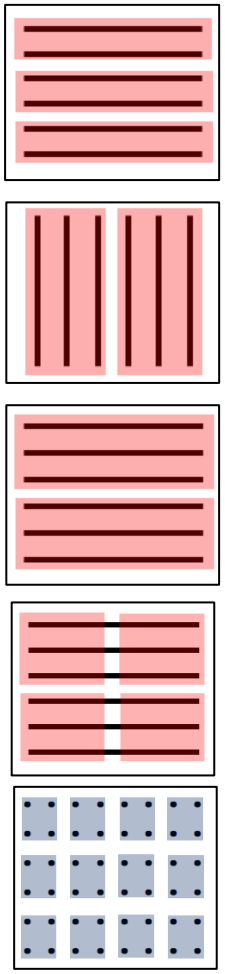
```
def sum_2_rows(M):

def sum_left_right(M):

def sum_upper_low(M):

def sum_4_quadrants(M):

def sum_4_cells(M):
```



Broadcasting

- When two arrays with different dimension sizes are operated together as element-wise
- The system will broadcast the dimension with smaller size to be equal to the dimension with larger size before performing an operation

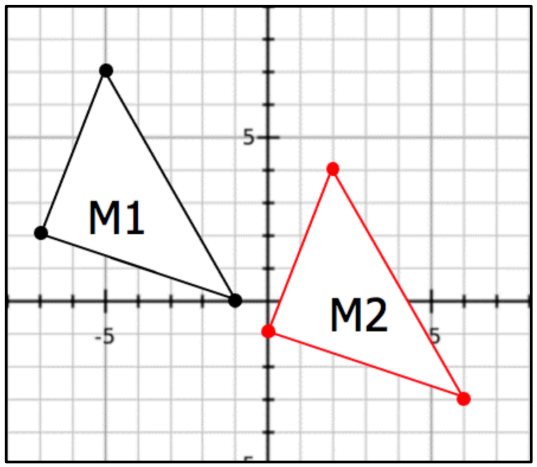
$$[2 \quad 3] + \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \Rightarrow \begin{bmatrix} 2 & 3 \\ 2 & 3 \\ 2 & 3 \end{bmatrix} + \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

- Sometimes, the broadcasting won't work

$$[2 \quad 3 \quad 4] + \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \quad \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix} + \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

Example: Matrix Translation


```
     x   y
[[-7   2]
 [-5   7]
 [-1   0]]
```




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

We would like to move all points to the right 7 steps and to downward 3 steps. Hence, we need to add all points with [7, -3]

```
M1 = np.array([[-7,2],[-5,7,[-1,0])
T  = np.array([7,-3],[7,-3],[7,-3])
M2 = M1+T
```

M2 = M1 + np.array([7,-3])

```
[[-7   2]                          [[-7   2]       [[7  -3]
 [-5   7]   +   [7  -3]     =>      [-5   7]   +    [7  -3]
 [-1   0]]                          [-1   0]]       [7  -3]]
```


python powered

Example: broadcast smaller dimension size to a bigger one (1)

100th Anniversary of Chula Engineering 2013

```
x = np.array([[1,2],[3,4],[5,6]])
u = np.array([2]) + x
```

$$\begin{bmatrix} 2+1 & 2+2 \\ 2+3 & 2+4 \\ 2+5 & 2+6 \end{bmatrix}$$

python powered

Example: broadcast smaller dimension size to a bigger one (2)


```
x = np.array([[1,2],[3,4],[5,6]])
u = np.array([10,20]) + x
```

$$\begin{bmatrix} 10+1 & 20+2 \\ 10+3 & 20+4 \\ 10+5 & 20+6 \end{bmatrix}$$

```
  (2)          (3, 2)
(3, 2)   →     (3, 2)
```

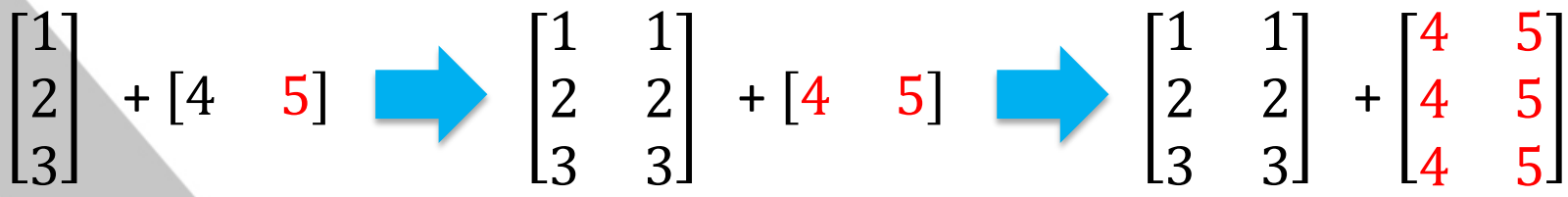
Example: broadcast smaller dimension size to a bigger one (3)


```
x = np.array([[1,2],[3,4],[5,6]])
u = np.array([[10],[20],[30]]) + x
```

$$\begin{bmatrix} 10+1 & 10+2 \\ 20+3 & 20+4 \\ 30+5 & 30+6 \end{bmatrix}$$

```
(3, 1)        (3, 2)
(3, 2)        (3, 2)
```

Example: broadcast both arrays




```
(3, 1)
(2)
```

```
(3, 2)
 (2)
```

```
(3, 2)
(3, 2)
```

Practice#5

100th Anniversary of Chula Engineering 2013

Write a program to construct a multiplication table using NumPy array and not using any loops.

```
[[  1   2   3   4   5   6   7   8   9  10  11  12]
 [  2   4   6   8  10  12  14  16  18  20  22  24]
 [  3   6   9  12  15  18  21  24  27  30  33  36]
 [  4   8  12  16  20  24  28  32  36  40  44  48]
 [  5  10  15  20  25  30  35  40  45  50  55  60]
 [  6  12  18  24  30  36  42  48  54  60  66  72]
 [  7  14  21  28  35  42  49  56  63  70  77  84]
 [  8  16  24  32  40  48  56  64  72  80  88  96]
 [  9  18  27  34  45  54  63  72  81  90  99 108]
 [ 10  20  30  40  50  60  70  80  90 100 110 120]
 [ 11  22  33  44  55  66  77  88  99 110 121 132]
 [ 12  24  36  48  60  72  84  96 108 120 132 144]]
```

python powered

Practice#5

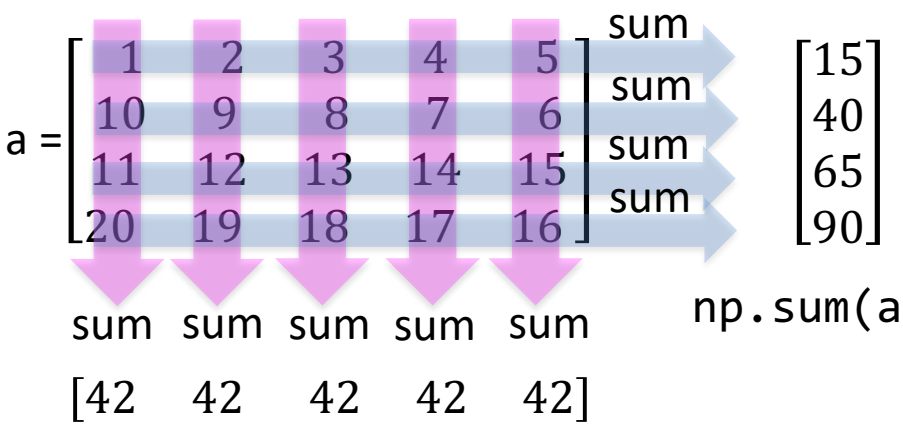
Write a program to construct a multiplication table using NumPy array and not using any loops.

```
[[  1   2   3   4   5   6   7   8   9  10  11  12]
 [  2   4   6   8  10  12  14  16  18  20  22  24]
 [  3   6   9  12  15  18  21  24  27  30  33  36]
 [  4   8  12  16  20  24  28  32  36  40  44  48]
 [  5  10  15  20  25  30  35  40  45  50  55  60]
 [  6  12  18  24  30  36  42  48  54  60  66  72]
 [  7  14  21  28  35  42  49  56  63  70  77  84]
 [  8  16  24  32  40  48  56  64  72  80  88  96]
 [  9  18  27  34  45  54  63  72  81  90  99 108]
 [ 10  20  30  40  50  60  70  80  90 100 110 120]
 [ 11  22  33  44  55  66  77  88  99 110 121 132]
 [ 12  24  36  48  60  72  84  96 108 120 132 144]]
```

Interesting functions in NumPy

- np.sum
- np.max, np.argmax
- np.min, np.argmin
- np.mean, np.std
- np.dot

np.sum

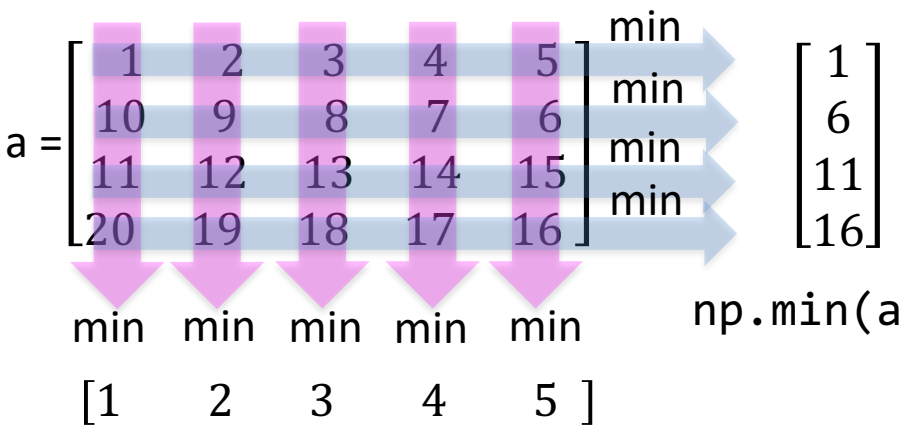



```
np.sum(a,axis=1)
```

```
np.sum(a,axis=0)
```

```
np.sum(a) = sum of all values = 210
```

np.min



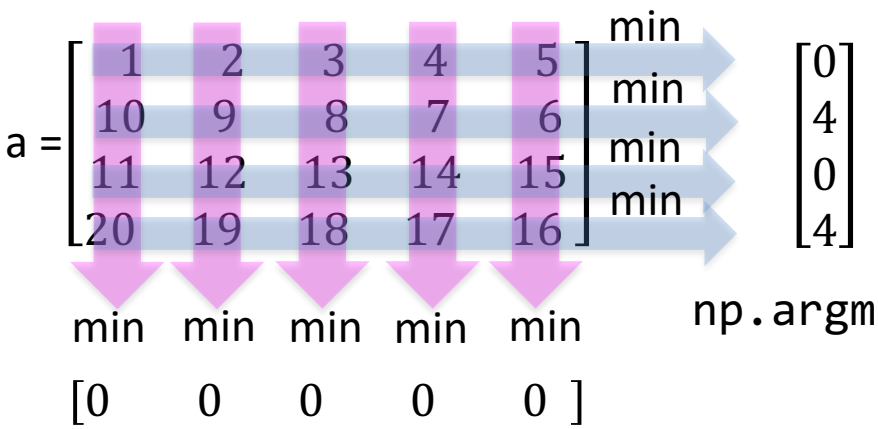

```
np.min(a,axis=1)
```

```
np.min(a,axis=0)
```

```
np.min(a) = min of all values = 1
```

```
np.max is similar to np.min but with max value
```

np.agrmin



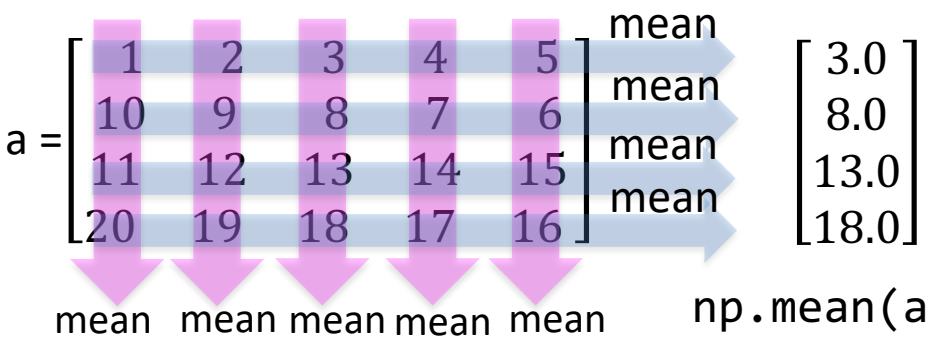

```
np.argmin(a,axis=1)
```

```
np.argmin(a,axis=0)
```

```
np.argmin(a) = index of the minimum
of all values = 0
```

np.argmax is similar to np.argmin but with the index of maximum value.

np.mean




```
np.mean(a,axis=1)
```

[10.5 10.5 10.5 10.5 10.5]

```
np.mean(a,axis=0)
```

```
np.mean(a) = mean of all values = 10.5
```

np.std is similar to np.mean but with the calculated standard deviation.

np.dot

`np.dot(vector, vector)`

$$[1 \;\; 2 \;\; 3] \cdot [4 \;\; 5 \;\; 6] = 1 \cdot 4 + 2 \cdot 5 + 3 \cdot 6 = 32$$

`np.dot(vector, matrix)`

$$[1 \;\; 2 \;\; 3] \cdot \begin{bmatrix} 4 & 7 \\ 5 & 8 \\ 6 & 9 \end{bmatrix} = [[1 \;\; 2 \;\; 3] \cdot [4 \;\; 5 \;\; 6] \;\; [1 \;\; 2 \;\; 3] \cdot [7 \;\; 8 \;\; 9]]$$

$$= [32 \;\; 50]$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \cdot [2 \;\; 3] = [[1 \;\; 2] \cdot [2 \;\; 3] \;\; [3 \;\; 4] \cdot [2 \;\; 3] \;\; [5 \;\; 6] \cdot [2 \;\; 3]]$$

$$= [8 \;\; 18 \;\; 28]$$

`np.dot(matrix, matrix) : matrix multiplication`

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 & 64 \\ 139 & 154 \end{bmatrix}$$

Either np.???(a,b) or a.???(b) is fine


```
import numpy as np

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

a = np.dot(x,y)
a = x.dot(y)

b = np.sum(x,axis=0)
b = x.sum(axis=0)

c = np.mean(x,axis=1)
c = x.mean(axis=1)
```

Example: Total weekly income

A food shop provides 3 menu as follows:
M1: Chicken curry with rice 25 Baht
M2: Fired rice 30 Baht
M3: Sukiyaki 45 Baht
Last week, the number of orders for each menu were noted below.

	Mon	Tue	Wed	Thurs	Fri
M1	75	120	70	90	80
M2	80	90	100	70	50
M3	50	45	70	65	50

prices

$$[25 \quad 30 \quad 45]$$

dailysales

$$\begin{bmatrix} 75 & 120 & 70 & 90 & 80 \\ 80 & 90 & 100 & 70 & 50 \\ 50 & 45 & 70 & 65 & 50 \end{bmatrix}$$

$$[25 \quad 30 \quad 45] \cdot \begin{bmatrix} 75 & 120 & 70 & 90 & 80 \\ 80 & 90 & 100 & 70 & 50 \\ 50 & 45 & 70 & 65 & 50 \end{bmatrix} = [6525 \quad 7725 \quad 7900 \quad 7275 \quad 5750]$$

Weekly income

`np.sum(…)`

python powered

Example: Weekly income report

A food shop provides 3 menu as follows:

M1: Curry rice 25 Baht
M2: Fired rice 30 Baht
M3: Seafood Sukiyaki 45 Baht

Last week, the number of orders for each menu were noted below.

	Mon	Tue	Wed	Thurs	Fri
M1	75	120	70	90	80
M2	80	90	100	70	50
M3	50	45	70	65	50

```
MO  --> 6525
TU  --> 7725
WE  --> 7900
TH  --> 7275
FR  --> 5750
Weekly income = 35175
Daily average = 7035.0
Best sales day = WE
Sales loss on: MO, TH, FR
----------------------
M1  -->  10875
M2  -->  11700
M3  -->  12600
Best menu = Seafood Sukiyaki
```

[6525 7725 7900 7275 5750]

```
dailyincomes   = np.dot(prices, dailysales)
weeklyincome   = np.sum(dailyincomes)
dailyaverage   = np.mean(dailyincomes)
best_day_index = np.argmax(dailyincomes)
```

loss when income < 7500

python powered

Example: Weekly income report


```
def report(prices,dailysales,breakeven):
    days = [“MO”, “TU”, “WE”, “TH”, “FR”]
    menus = [“Curry rice”, “Fried rice”, “Seafood Sukiyaki”]

    dailyincomes = np.dot(prices,dailysales)
    for i in range(len(days)):
        print(days[i], “-->”, dailyincomes[i])
    print(“weekly income = ”, np.sum(dailyincomes)
    print(“daily average = ”, np.mean(dailyincomes)
    print(“best sales day = ”, days[np.argmax(dailyincomes)]

    loss = np.array(days)[dailyincomes < breakeven]
    print(“Sales loss on:”, “, ”.join(loss))
    print(“---------------------”)

    menuincomes = np.sum(dailysales,axis=1) * prices
    for i in range(len(menus)):
        print(menus[i], “-->”, menuincome[i])
    print(“Best menu =”, menus[np.argmax(menuincomes)])
```

Practice#6

100th Anniversary of Chula Engineering 2013

```
data = np.array([ [610011, 80, 90, 70],
                  [610022, 50, 80, 68],
                  [610033, 70, 85, 80],
                  [610044, 60, 50, 90],
                  [610055, 90, 74, 70]])

weight = np.array([0.3, 0.5, 0.2])
```

Total scores of 610011 = 0.3x80 + 0.5x90 + 0.2x70 = 83.0

Question: Which students got the total scores less than the average total score of all students?

python powered