

# NumPy

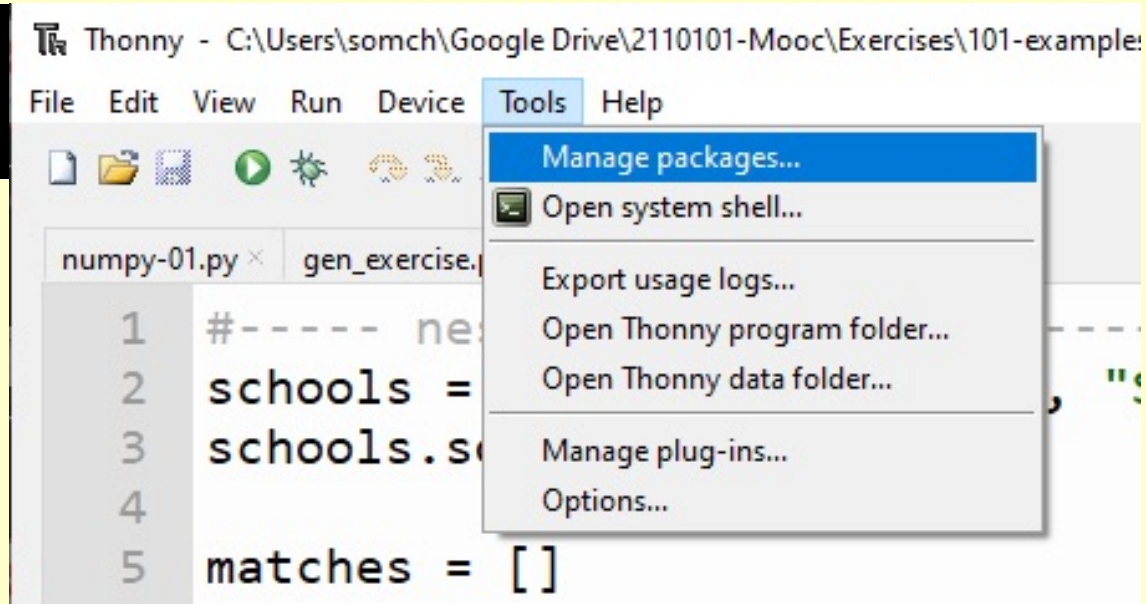
ภาควิชาวิศวกรรมคอมพิวเตอร์  
จุฬาลงกรณ์มหาวิทยาลัย

๒๕๖๒

# NumPy

- ชุดคำสั่งเพื่อการประมวลผลในงานทางวิทยาศาสตร์
- เขียนง่าย สั้น และทำงานเร็วมาก
- <https://www.numpy.org/>
- ไม่ได้มากับ Python ต้องติดตั้งเพิ่ม

```
C:\>pip install numpy
```



# การเก็บข้อมูล: NumPy Array

Array: คือการเก็บข้อมูลเป็นแถวลำดับเรียงกันไป คล้ายลิสต์ แต่มีข้อแตกต่าง เช่น

- ทุกช่องในอาเรย์เก็บข้อมูลเดียวกันหมด (มักเก็บจำนวน)
- เก็บข้อมูลได้หลายมิติ
  - 1 มิติ : vector  $[1, 2, 3, 4]$
  - 2 มิติ : matrix  $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
  - n มิติ : tensor  $\left[ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \right]$
- ใช้ tuple เป็น index เพื่อใช้ข้อมูลในอาเรย์ เช่น a เป็นอาเรย์ 2 มิติ เขียน a[ (1,2) ] หรือจะเขียน a[1,2] ก็ได้
- มี operators และ methods ให้ใช้งานมากมาย

# List vs. NumPy Array: ระยะทางทุกคู่จุด

```
def all_pair_distances(points):  
    # points เป็น nested list เช่น [[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 เป็น NumPy 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

การทดลอง n = 2000  
List: 5.44 s.  
NumPy: 0.337 s.

เข้าใจง่ายกว่า &  
ทำงานเร็วกว่ามาก

# NumPy Array ใน 2110101 (นิดเดียว)

การสร้างอาร์เรย์แบบต่าง ๆ

indexing

element-wise operations

broadcasting

ฟังก์ชันที่น่าสนใจ

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

# การสร้างอาร์เรย์

```
import numpy as np

a = np.array([1,2,3,4]) # สร้างจากลิสต์
b = np.array([[1,2],[3,4]],float) # สร้างจากลิสต์
c = np.ndarray( (2,3) ) # สร้างตามขนาด ค่าไม่รู้
d = np.ndarray( (2,3), int)
e = np.zeros( (2,3), int) # สร้างตามขนาด ค่า 0 หมด
f = np.ones( (2,3), int) # สร้างตามขนาด ค่า 1 หมด
g = np.zeros_like (f, float) # ขนาดเหมือน f ค่า 0 หมด
h = np.ones_like( e, float) # ขนาดเหมือน e ค่า 1 หมด
I = np.identity( 4, int) # identity matrix ขนาด 4x4
x = np.arange(0.0, 1.0, 0.1) # [0.0, 0.1, 0.2, ..., 0.9]
```

int ก็ได้, float ก็ได้ (range ได้แค่ int)

# array.shape

- `array.shape` คืน tuple บอกรายละเอียดของมิติ
  - `a = np.ones( (3, 4) )`  
จะได้ `a.shape` เป็น `(3, 4)`
    - `a.shape[0]` คือ 3 เป็นจำนวนแถว
    - `a.shape[1]` คือ 4 เป็นจำนวนคอลัมน์
- `len( array.shape )` เป็นขนาดของมิติ
  - `a = np.ones( (3, 4) )`  
จะได้ `len( a.shape )` เป็น 2

# array.reshape( newshape )

นำข้อมูลใน array มาจัดรูปแบบให้ตรงตาม shape

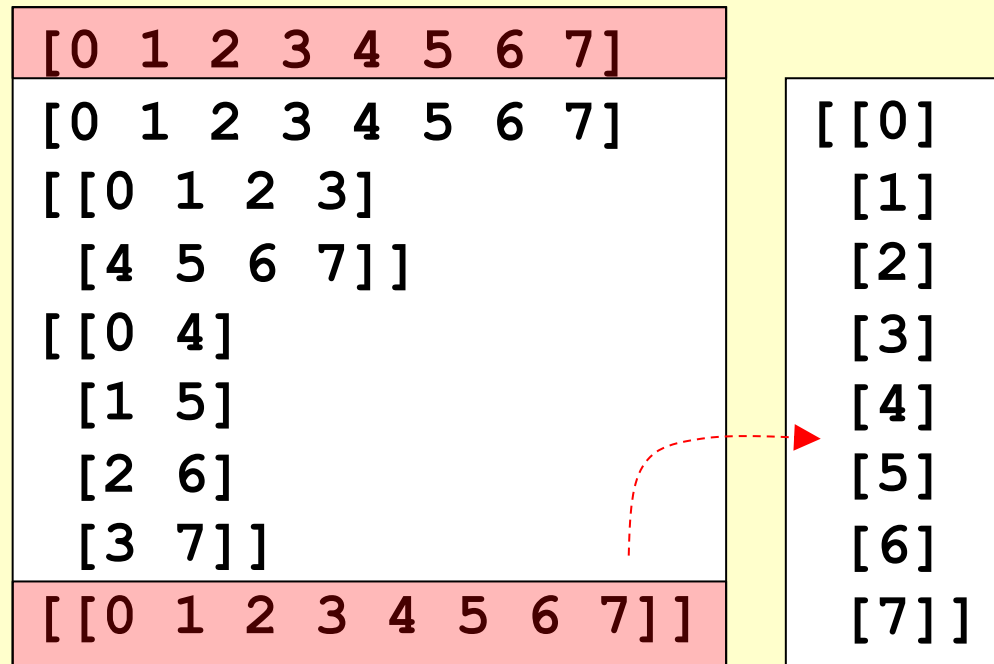
```
a = np.arange(8)          # [0 1 2 3 4 5 6 7]
b = a.reshape((2,4))      # [[0 1 2 3],
                           #  [4 5 6 7]]
c = b.reshape((4,2))      # [[0 1],
                           #  [2 3],
                           #  [4 5],
                           #  [6 7]]
d = c.reshape(8)      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



# array.T

- `a.T` คือ transpose ของอาร์เรย์ `a`
- `a` มี 1 มิติ, `a.T` เหมือน `a`
- `a` มี 2 มิติ, `a.T` คือ transpose ของเมทริกซ์ `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)
```



ข้อสังเกต: `a` ไม่เหมือน `c` (shape ไม่เหมือน)

# Indexing

ใช้ tuple ระบุตำแหน่งในอาเรย์

```
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
```

เขียน `A[i, j]` เหมือนกับ `A[(i, j)]`

เดี๋ยวละรู้ว่าการนับจำนวน 1 ใน `A` เขียน `np.sum(A==1)` ก็พอ

# Slicing: start : stop : step

- รูปแบบ: A[ เลือกแถว , เลือกคอลัมน์ ]
- ระวัง: A[ เลือกแถว ][ ตรงนี้ไม่ใช่เลือกคอลัมน์ ]

```
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]
                          #  [7 9]]
                          #  เลือกแถวคู่ คอลัมน์คู่
print( A[:,::-1, ::-1] )  # [[12 11 10]
                          #  [ 9  8  7]
                          #  [ 6  5  4]
                          #  [ 3  2  1]]
```

# 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]
```

The diagram illustrates the fancy indexing operation. It shows how the indices from the second array `B` are used to select elements from the first array `A`. Specifically, the first row of `B` contains indices `[1, 3, 2]` for the first dimension, and the second row contains indices `[2, 0, 1]` for the second dimension. Lines connect these indices to the corresponding elements in `A`: `A[1,2]` (value 2), `A[3,0]` (value 0), and `A[2,1]` (value 8). The resulting array `B` is `[2, 0, 8]`.

# แบบฝึกหัด

```
# A is a 2-d array
def get_column_from_bottom_to_top( A, c ):
    return _____ # บรรทัดเดียว

def get_odd_rows( A ):
    return _____ # บรรทัดเดียว

def get_even_rows_last_column( A ):
    return _____ # บรรทัดเดียว

def get_diagonal1( A ): # A is a square matrix
    _____
    return _____ # สองบรรทัด

def get_diagonal2( A ): # A is a square matrix
    _____
    return _____ # สองบรรทัด
```

# การนำค่าสเกลาร์ใส่ในอาเรย์

ใส่ค่าสเกลาร์ให้กับทุกช่องทางซ้ายของ =

```
A = np.zeros(8)
```

```
A[2:5] = 9
```

[0 0 9 9 9 0 0 0]

```
A = np.ndarray((4,4), int)
```

```
A[:, :] = 9
```

นำ 9 ใส่ทุกแถว, ทุกคอลัมน์

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

```
B = np.zeros((4,4), int)
```

```
B[:, 0::2] = 1
```

```
B[1::2, :] = 2
```

ใส่ 1 ในคอลัมน์คู่ของทุกแถว

ใส่ 2 ในแถวคี่ของทุกคอลัมน์

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

# การคำนวณแต่ละค่าในอาเรย์กับค่าสเกลาร์

คำนวณให้ตัวต่อตัว และคืนผลเป็นอาเรย์

```
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. ]
```

# หลายฟังก์ชันที่มีใน math มีใน numpy ด้วย

```
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)  
# [ 1.00000000e+00,  1.2246468e-16, -1.00000000e+00]
```



# การเปรียบเทียบค่าในอาเรย์กับสเกลาร์

เปรียบเทียบให้ตัวต่อตัว และคืนผลเป็นอาเรย์ True/False

```
a = np.array( [1, 2, 3, 4] )  
b = a > 3      # ได้ [False False False True]  
c = a%2 == 1   # ได้ [True False True False]
```

ต้องการนับจำนวนเลขคี่ในอาเรย์ a

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

ใน Python  
True มีค่า 1  
False มีค่า 0

# แบบฝึกหัด

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

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

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

# แบบฝึกหัด: Logistic Regression

สูตรทำนายโอกาส  $p(x)$  ที่นักเรียน  $x$  เรียนผ่านวิชาหนึ่ง  
จากจำนวนโจทย์ที่ทำ ( $x_0$ ) กับเกรดเฉลี่ยที่มี ( $x_1$ )

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

$$\text{logit}(x) = -3.98 + 0.1x_0 + 0.5x_1$$

จงเขียนโปรแกรมอ่านจำนวนโจทย์และเกรดเฉลี่ยของ  
นักเรียนกลุ่มหนึ่ง เพื่อคำนวณผลการทำนาย

# 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 ]

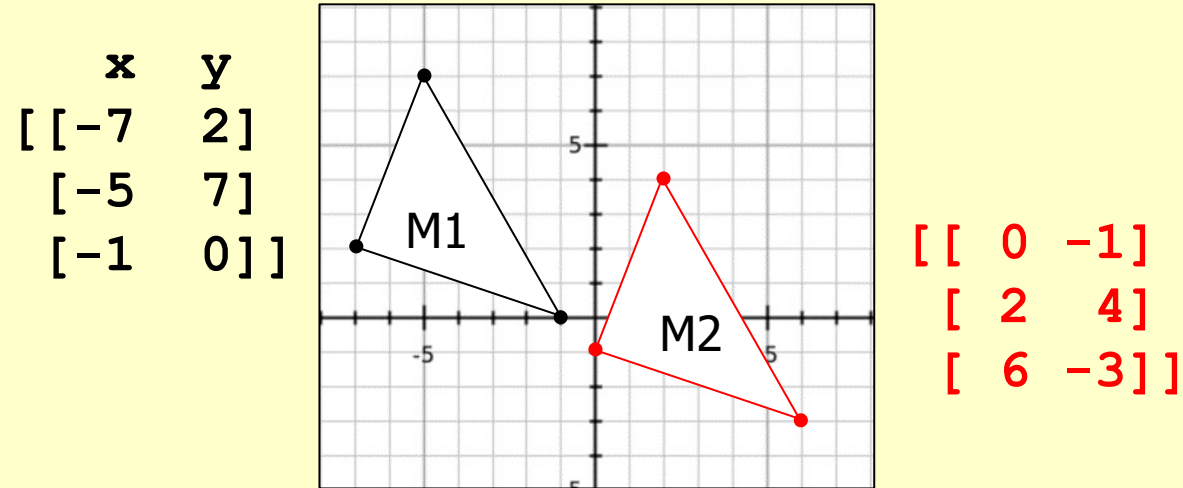
~ [ False True True False ]

ต้องใช้เครื่องหมาย &, |, ~ สำหรับการทำ and, or, not แบบตัวต่อตัว

# Element-Wise Logical Operators

```
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 ]          # ผิด
b = a[ 2 < a and a < 5 ]    # ผิด
b = a[ 2 < a & a < 5 ]      # ผิด
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]
```

# ตัวอย่าง : Matrix Translation



ต้องการย้ายทุกจุดไปทางขวา 7, ลงล่าง 3 คือบวกทุกจุดด้วย  $[7, -3]$

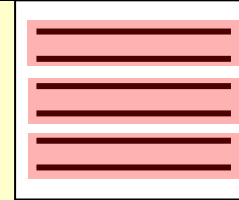
```
M1 = np.array([ [-7, 2], [-5, 7], [-1, 0] ])
T  = np.array([ [7, -3], [7, -3], [7, -3] ])
M2 = M1 + T
```

$$\begin{bmatrix} -7 & 2 \\ -5 & 7 \\ -1 & 0 \end{bmatrix} + \begin{bmatrix} 7 & -3 \\ 7 & -3 \\ 7 & -3 \end{bmatrix} \text{ ได้ } \begin{bmatrix} 0 & -1 \\ 2 & 4 \\ 6 & -3 \end{bmatrix}$$

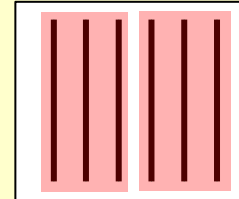
มีวิธีง่ายกว่านี้ โปรดติดตาม...

# แบบฝึกหัด

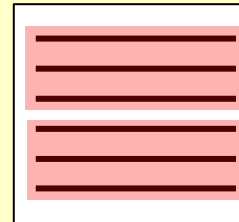
```
def sum_2_rows( M ):
```



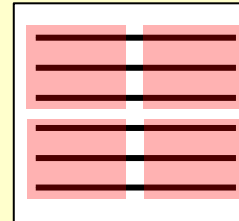
```
def sum_left_right( M ):
```



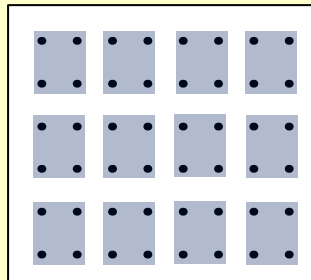
```
def sum_upper_lower( M ):
```



```
def sum_4_quadrants( M ):
```



```
def sum_4_cells( M ):
```





# Broadcasting

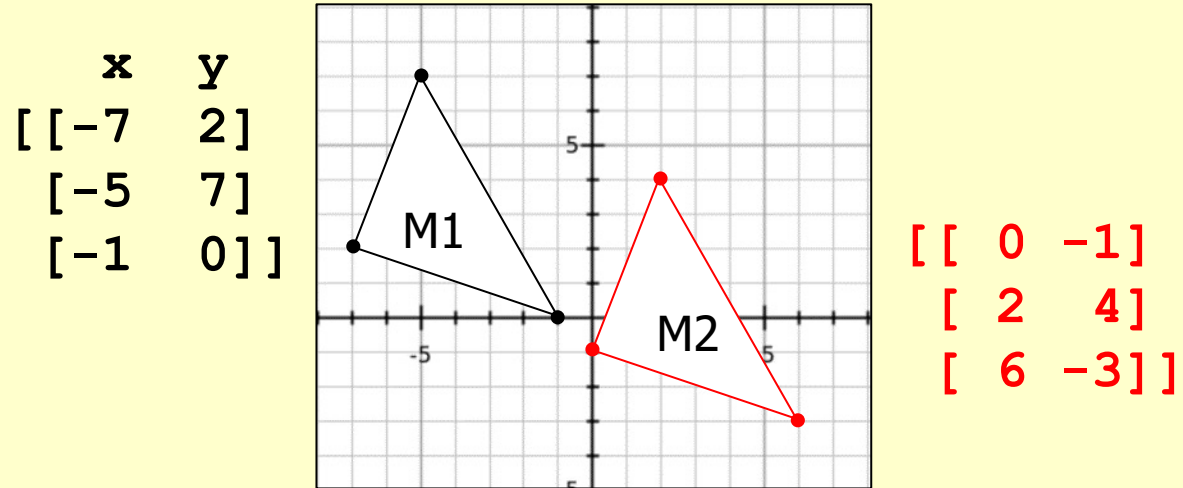
- เมื่อนำอาเรย์ 2 ตัวมาคำนวณแบบ element-wise แต่อาเรย์ทั้งสองมีขนาดไม่เท่ากัน
- ระบบจะ broadcast อาเรย์ตัวเล็ก (หรืออาจทำทั้งสองตัว) ให้มีขนาดเท่ากันก่อนทำงาน

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

- แต่บางครั้งก็ broadcast ไม่ได้

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

# ตัวอย่าง: Matrix Translation



ต้องการย้ายทุกจุดไปทางขวา 7, ลงล่าง 3 คือบวกทุกจุดด้วย  $[7, -3]$

M1 กับ T  
มีขนาดเท่ากัน

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

เขียนแค่นี้ก็พอ

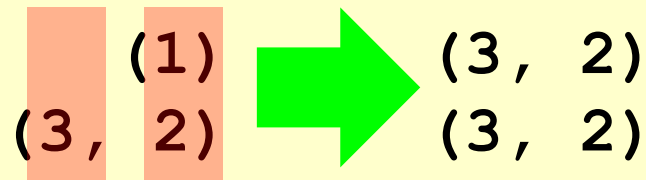
```
M2 = M1 + np.array( [7, -3] )
```

$$\begin{bmatrix} -7 & 2 \\ -5 & 7 \\ -1 & 0 \end{bmatrix} + \begin{bmatrix} 7 & -3 \end{bmatrix} \Rightarrow \begin{bmatrix} -7 & 2 \\ -5 & 7 \\ -1 & 0 \end{bmatrix} + \begin{bmatrix} 7 & -3 \\ 7 & -3 \\ 7 & -3 \end{bmatrix}$$

# ตัวอย่าง: broadcast ตัวเล็ก ให้เท่าตัวใหญ่

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

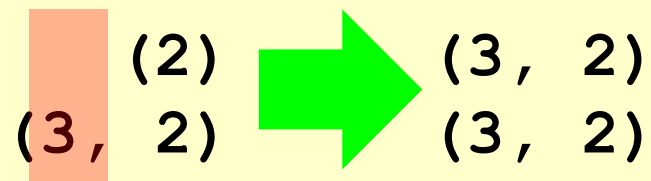
$$\begin{bmatrix} 2 + 1 & 2 + 2 \\ 2 + 3 & 2 + 4 \\ 2 + 5 & 2 + 6 \end{bmatrix}$$



# ตัวอย่าง: broadcast ตัวเล็ก ให้เท่าตัวใหญ่

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

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



$(3, 2) \rightarrow (3, 2)$

# ตัวอย่าง: broadcast ตัวเล็ก ให้เท่าตัวใหญ่

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

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

$$\begin{matrix} (3, 1) \\ (3, 2) \end{matrix} \xrightarrow{\quad} \begin{matrix} (3, 2) \\ (3, 2) \end{matrix}$$

## ตัวอย่าง: broadcast ทั้ง 2 ตัว

$$\begin{array}{ccc} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} + [4 \quad 5] & \xrightarrow{\quad} & \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 3 \end{bmatrix} + [4 \quad 5] & \xrightarrow{\quad} & \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 3 \end{bmatrix} + \begin{bmatrix} 4 & 5 \\ 4 & 5 \\ 4 & 5 \end{bmatrix} \\ (3, \text{1}) & & (\text{3}, 2) & & (3, 2) \\ (2) & & (2) & & (3, 2) \end{array}$$

# แบบฝึกหัด: Outer Product

จงเขียนโปรแกรมสร้างอาเรย์ที่เก็บสูตรคูณแม่ 1 ถึง 12 ข้างล่างนี้  
ด้วยคำสั่ง NumPy โดยไม่ต้องใช้วงวน

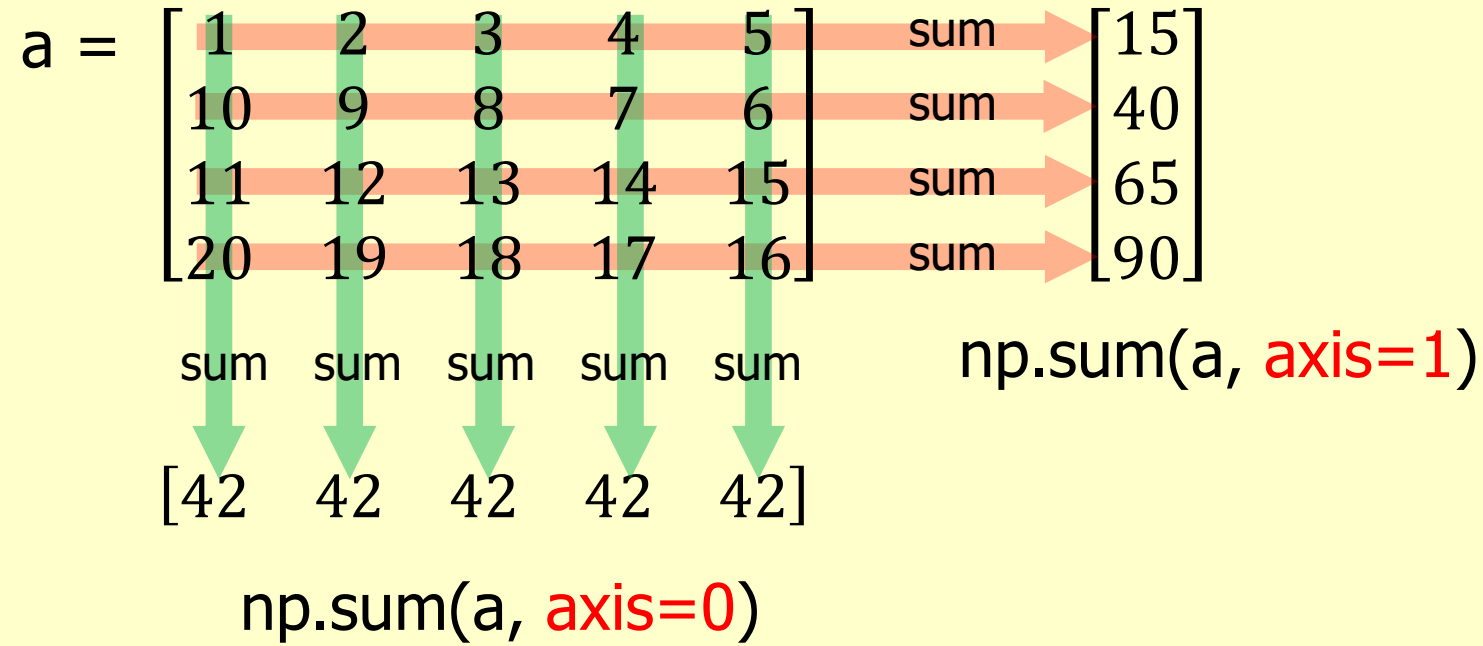
```
[[ 1  2  3  4  5  6  7  8  9 10 11 12]
 [ 2  4  6  8 10 12 14 16 18 20 22 24]
 [ 3  6  9 12 15 18 21 24 27 30 33 36]
 [ 4  8 12 16 20 24 28 32 36 40 44 48]
 [ 5 10 15 20 25 30 35 40 45 50 55 60]
 [ 6 12 18 24 30 36 42 48 54 60 66 72]
 [ 7 14 21 28 35 42 49 56 63 70 77 84]
 [ 8 16 24 32 40 48 56 64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

# ฟังก์ชันที่น่าสนใจของ NumPy

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

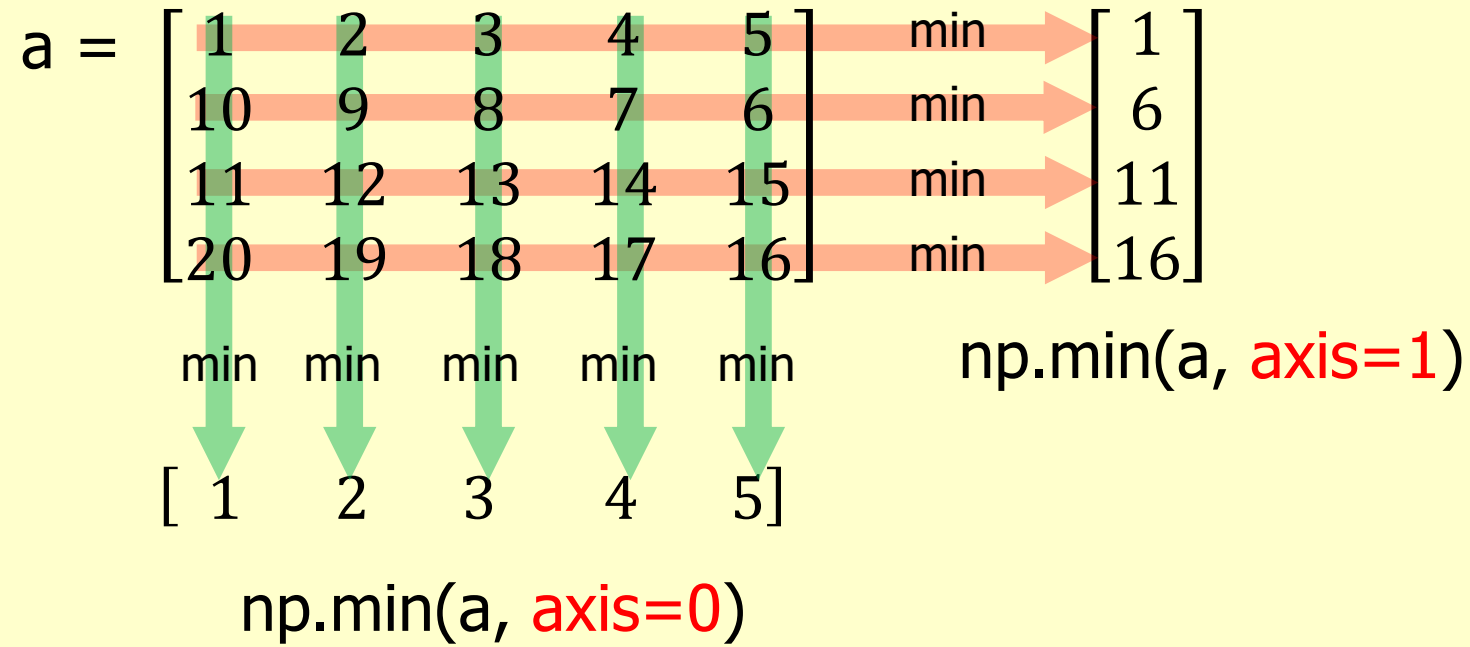


# np.sum



np.sum(a) ของทั้งหมดได้ 210

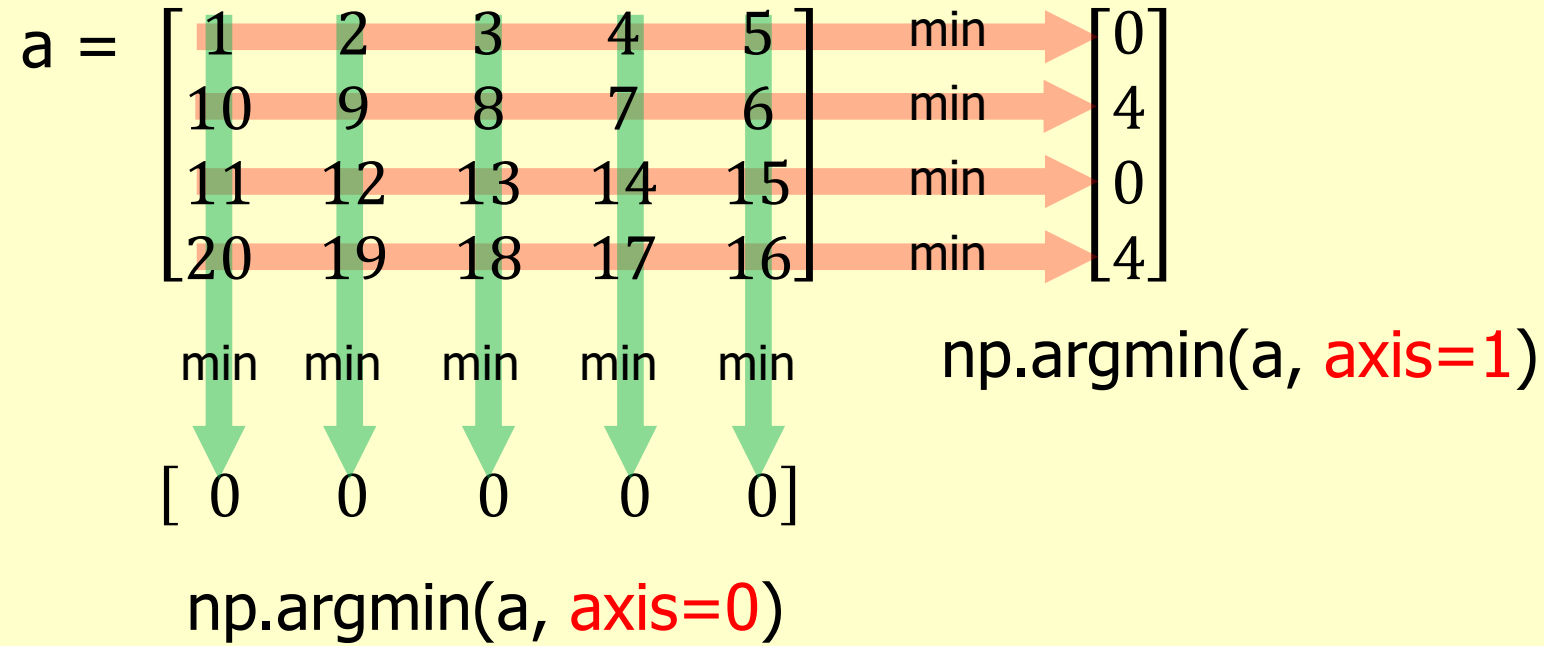
# np.min



`np.min(a)` ของทั้งหมดได้ 1

`np.max` ก็คล้าย `np.min` แต่ได้ค่ามากที่สุด

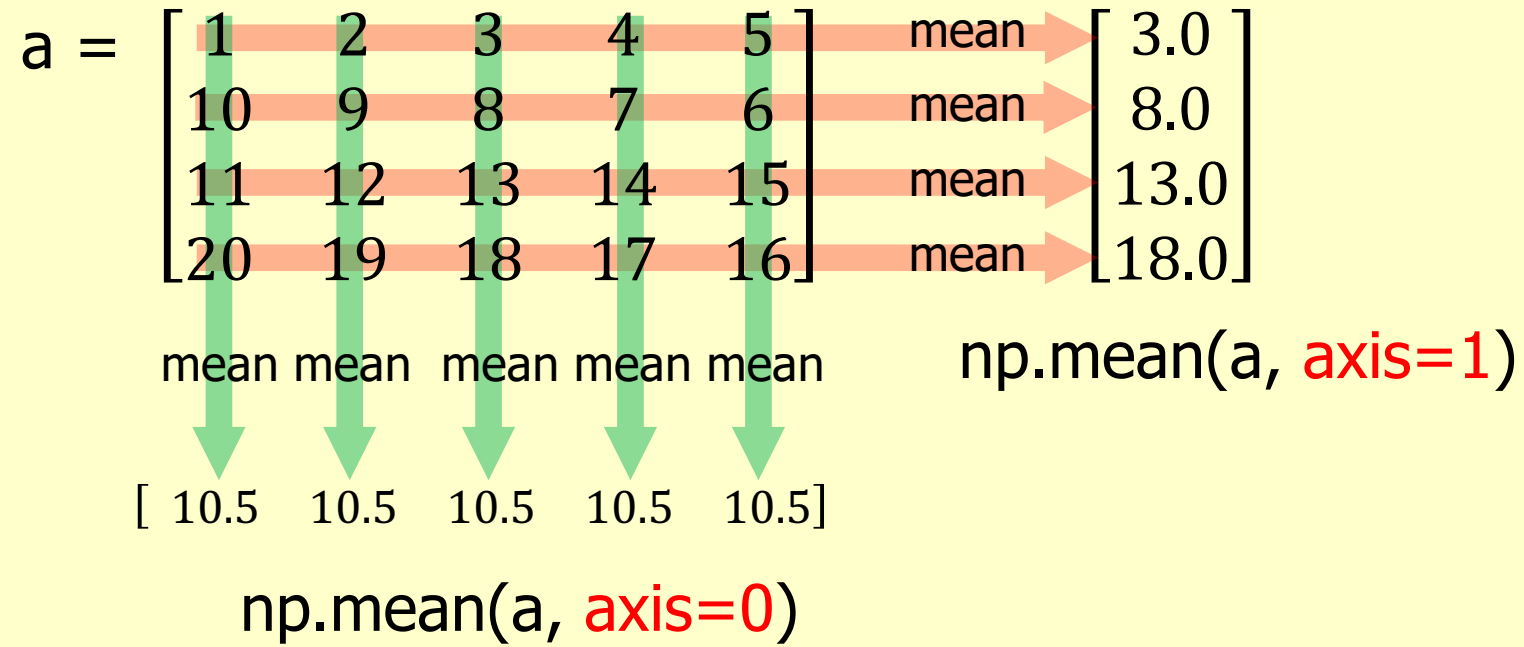
# np.argmin



`np.argmin(a)` ของทั้งหมดได้ 0

`np.argmax` ก็คล้าย `np.argmin` แต่ได้ตำแหน่งของค่ามากที่สุด

# np.mean



np.mean(a) ของทั้งหมดได้ 10.5

np.std ก็คล้าย np.mean แต่ได้เบี่ยงเบนมาตรฐาน

# np.dot

- np.dot(vector, vector)

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \cdot \begin{bmatrix} 4 & 5 & 6 \end{bmatrix} = 1 \cdot 4 + 2 \cdot 5 + 3 \cdot 6 = 32$$

- np.dot(vector, matrix)

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \cdot \begin{bmatrix} 4 & 7 \\ 5 & 8 \\ 6 & 9 \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \cdot \begin{bmatrix} 4 & 5 & 6 \end{bmatrix} & \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \cdot \begin{bmatrix} 7 & 8 & 9 \end{bmatrix} \end{bmatrix} \\ = \begin{bmatrix} 32 & 50 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \cdot \begin{bmatrix} 2 & 3 \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} 2 & 3 \end{bmatrix} & \begin{bmatrix} 3 & 4 \end{bmatrix} \cdot \begin{bmatrix} 2 & 3 \end{bmatrix} & \begin{bmatrix} 5 & 6 \end{bmatrix} \cdot \begin{bmatrix} 2 & 3 \end{bmatrix} \end{bmatrix} \\ = \begin{bmatrix} 8 & 18 & 28 \end{bmatrix}$$

- np.dot(matrix, matrix) ก็คือการคูณ matrix

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

# เขียน np.???(a,b) หรือ a.???(b) ก็ได้

```
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)
```

## ตัวอย่าง: รายได้รวมในสัปดาห์

ร้านขายอาหารตามสั่งมีราคาอาหารคือ  
ข้าวแกง 25 บาท  
ข้าวผัด 30 บาท  
สุกี้ทะเล 45 บาท  
ในสัปดาห์ที่ผ่านมาขายอาหารได้ดังนี้

	จันทร์	อังคาร	พุธ	พฤหัสบดี	ศุกร์
ข้าวแกง	75	120	70	90	80
ข้าวผัด	80	90	100	70	50
สุกี้ทะเล	50	45	70	65	50

⇒ [25 30 45]

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

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



weekly income

np.sum( ... ) → 35175

# รายงานรายได้ประจำสัปดาห์

ร้านขายอาหารตามสั่งมีราคาอาหารคือ  
ข้าวแกง 25 บาท  
ข้าวผัด 30 บาท  
สุกี้ทะเล 45 บาท  
ในสัปดาห์ที่ผ่านมาขายอาหารได้ดังนี้

	จันทร์	อังคาร	พุธ	พฤหัสบดี	ศุกร์
ข้าวแกง	75	120	70	90	80
ข้าวผัด	80	90	100	70	50
สุกี้ทะเล	50	45	70	65	50

[6525 7725 7900 7275 5750]

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

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  
-----  
Curry Rice --> 10875  
Fried Rice --> 11700  
Seafood Suki --> 12600  
Best menu = Seafood Suki

ขาดทุนเมื่อ  
< 7500



# รายงานรายได้ประจำสัปดาห์

```
def report(prices, dailysales, breakeven):
    days = ["MO", "TU", "WE", "TH", "FR"]
    menus = ["Curry Rice", "Fried Rice", "Seafood Suki"]

    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], '-->', menuincomes[i])

    print("Best menu =", menus[np.argmax(menuincomes)])
```

# แบบฝึกหัด: ใครคะแนนรวมต่ำกว่าคะแนนเฉลี่ย

```
          ID      Midterm Final Project
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])
```

คะแนนรวมของ 610011 =  $0.3 \times 80 + 0.5 \times 90 + 0.2 \times 70 = 83.0$

อยากรู้ว่าใครบ้างที่คะแนนรวมต่ำกว่าคะแนนเฉลี่ยของทั้งหมด