

## แนะนำ NumPy เพื่องาน OpenCV

โปรแกรมไพธอน (Python) เป็นโปรแกรมเอนกประสงค์ตัวหนึ่งที่ได้รับคามนิยมอย่างแพร่หลาย สร้างโดย Guido Van Rossum โดยเขาต้องการให้งานเขียนโปรแกรมได้ง่ายๆ การทำงานของไพธอนทำงานผ่าน ตัวแปลภาษาที่เรียกว่าอินเตอร์พรีเตอร์ (Interpreter) ซึ่งหมายความว่าต้องทำการแปลทุกครั้งเพื่อให้โปรแกรมทำงาน ซึ่งแน่นอนว่าโปรแกรมย่อมทำงานได้ช้ากว่า โปรแกรมภาษาที่ใช้การแปลผ่าน คอมไพเลอร์ (Compiler)

NumPy เป็นชุดคำสั่งสำหรับการคำนวณทางตัวเลข ที่สร้างขึ้นจากภาษาไพธอน เมื่อติดตั้ง Anaconda แล้วจะมี NumPy มาด้วย มีฟังก์ชันที่สำคัญคืออาร์เรย์ที่เก็บ ชุดตัวเลข หรือเมตริก ที่นำไปแทน ชุดของเมตสี หรือ พิกเซล (pixel)

สำหรับ Mat เป็นคลาสหนึ่งที่ใช้คำนวณ ภาพ หรือชุดของสี ที่ประกอบด้วยหลายเมตสี โดยคลาสนี้ มีสองประกอบสำคัญคือ Header ที่ระบุถึงขนาดของเมตริก และสื่อที่เก็บภาพ และอีกส่วนประกอบหนึ่งคือ Data ที่ระบุถึงข้อมูลของสี และตำแหน่งข้อมูลสี

### การสร้างอาร์เรย์ (Array)

อาร์เรย์ของ NumPy นำมาใช้เพื่อการคำนวณจะมีประสิทธิภาพมากกว่า List ของไพธอน โดยเฉพาะข้อมูลที่มีขนาดใหญ่ ข้อมูลที่มีหลายมิติ เช่นในข้อมูลในรูปเมตริก ดังนั้นทุกโครงสร้างของ OpenCV จะแปลงเป็นอาร์เรย์ของ NumPy เช่นการอ่านภาพ ก็จะเก็บในรูปอาร์เรย์

ตัวอย่างต่อไปนี้แสดงถึงการสร้าง Tuple และการสร้างอาร์เรย์แบบ NumPy

#### Code 1.

```
import numpy as np

#code 1.1
t = tuple( )
t = (1, 2, 3, 4, 5)
print(t)
# (1, 2, 3, 4, 5)

#code 1.2
a = np.array([1,2,3], int)
print(a)
# [1,2,3]

#code 1.3
b = np.arange(6)
print(b)
# [0 1 2 3 4 5]

#code 1.4
c = np.arange(6, dtype=np.int64)
print (c)
# [0 1 2 3 4 5]

#code 1.5
d = np.arange(3, 6)
print(d)
# [3, 4, 5]
```

จากตัวอย่างนี้จะพบว่าเราสร้างอาร์เรย์ แบบไม่กำหนดชนิดข้อมูล และแบบกำหนดชนิดข้อมูลได้ สำหรับคำสั่ง `arange` หมายถึงการกำหนดระยะตั้งแต่ 0 ถึง ก่อนระยะที่กำหนด ในตัวอย่างนี้ใช้ 6 ซึ่งหมายถึงข้อมูลตั้งแต่ 0 ถึง 5 การกำหนดระยะในไพธอน ใช้งานในลักษณะนี้เหมือนกันหมด

### รูปแบบการสร้าง

```
numpy.arange([start, ]stop, [step, ]dtype=None)
```

จากตัวอย่างการสร้างที่ผ่านมา ใช้ตามรูปแบบข้างต้น

- `start` แทนการค่าเริ่มต้น ซึ่งไม่ใส่จะหมายถึงเริ่มจากศูนย์
- `stop` แทนค่าที่จะหยุด
- `step` แทน ค่าที่เลื่อนทีละเท่าใด ซึ่งไม่ใส่หมายถึง เพิ่มทีละหนึ่ง
- และ `dtype` คือชนิดข้อมูล ถ้าไม่ใส่จะหมายถึง จะใช้ชนิดตามข้อมูลที่ใน `start` และ `stop`

### การอ่านค่าข้อมูลภายในอาร์เรย์

เมื่อสร้างข้อมูลอาร์เรย์ได้แล้ว ทีนี้มาถึงคราที่ต้อง อ่านอาร์เรย์ สำหรับ Python มีลักษณะการอ่านอาร์เรย์ที่ไม่เหมือนภาษาอื่น ๆ มากนัก เช่น ถ้ากำหนดให้ `a` เป็นอาร์เรย์มีข้อมูลตั้งแต่ 0 -8 จากการประกาศ `np.arange(9)`

- อ่านเป็นช่วง ด้วย `[2:5]` จะได้ 2 - 4
- อ่านเป็นช่วงและกระโดดที่สอง ด้วย `[:7:2]` จะได้ช่วง 0 ถึง 7 แต่ข้ามไปที่ละ 1 ค่า จึงได้ 0, 2, 4, 6
- อ่านเป็นช่วง แต่ยกเว้นค่าหลัง เช่น `[:-2]` จะได้ทั้งช่วงของอาร์เรย์ แต่ยกเว้น 2 ค่าสุดท้าย จึงได้ 0 - 6
- อ่านเป็นช่วง แต่ถอยหลัง เช่น `[::-1]` จะได้ทั้งช่วงอาร์เรย์ แต่เรียงจากหลังไปหน้า

### Code 2.

```
a = np.arange(9)

print(a)
# [0 1 2 3 4 5 6 7 8]

print(a[1])
# 1

print(a[2:5])
# [2 3 4]

print(a[:7:2])
# [0 2 4 6]

print(a[:-2])
# [0 1 2 3 4 5 6]

print(a[::-1])
# [8 7 6 5 4 3 2 1 0]
```

## อาร์เรย์หลายมิติ

ตัวอย่างต่อไปนี้จะแสดงการสร้างอาร์เรย์หลายมิติ และการเข้าถึงอาร์เรย์มิติต่างๆ

### Code 3.

```
c = np.array([[1,2,3], [4, 5, 6]], int)
print(c)
print("Shape:",c.shape)
print("[0,0]:",c[0,0])
print("[0,1]:",c[0,1])

...

[[1 2 3]
 [4 5 6]]
Shape: (2, 3)
[0,0]: 1
[0,1]: 2

...
```

การสร้างอาร์เรย์หลายมิติ ซึ่งจำเป็นต้องใช้กับรูปภาพ เช่น การสร้างอาร์เรย์ที่ประกอบไปด้วย ชุดตัวเลขศูนย์

### Code 4.

```
import numpy as np
a = np.zeros((2,2), dtype=np.uint8)
print(a)
...

[[0 0]
 [0 0]]
...

print(a.shape)
#print (2,2)
```

จากตัวอย่างนี้ เป็นการสร้างอาร์เรย์ศูนย์ มีขนาด 2x2 ซึ่งในกรณีใช้งานกับ OpenCV แทนหนึ่งเม็ดสีเทา จากชนิดข้อมูล 8 บิต ซึ่งมีจำนวนตัวเลข ตั้งแต่ 0 – 255 และมีค่าเป็นศูนย์หมดหมายถึงสีดำ

## ตัดแต่งขนาดอาร์เรย์

อาร์เรย์ ที่สร้างขึ้นแล้วสามารถที่จะแต่งขนาดขึ้นมาใหม่ได้ เช่น ถ้าประกาศให้อาร์เรย์มีขนาด 1 มิติ มีเลขตั้งแต่ 1 – 12 แล้ว กำหนดให้มีขนาดหรือมิติใหม่ เป็น (2, 2, 3) ซึ่งหมายความว่า ให้มีขนาด 2 มิติหลัก ภายในมิติหลักก็มีขนาด 2x3 หรือ 2 แถว 3 คอลัมน์

### Code 5.

```
import numpy as np
a = np.arange(12)
#[ 0  1  2  3  4  5  6  7  8  9 10 11]
print(a.shape)
# (12,)
b = a.reshape(2,2,3)
...

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

       [[ 6,  7,  8],
```

```
... [ 9, 10, 11]])
```

```
print(b.shape)  
# (2, 2, 3)
```

การอ่านข้อมูลในมิติต่าง มีเทคนิคที่อ่านได้หลายแบบ จากที่พบว่าเราอ่านแบบระบุแต่ละมิติไปแล้ว ในตัวอย่างต่อไปนี้จะแสดงการอ่านมิติ ด้วยการระบุ สามจุด (...) และ คอลอนน์ (:) ซึ่งหมายถึงเลือกทั้งหมด

Code 6.

---

```
b[0, 0, 0] # fist element of frist dimension : 0  
b[0, :, :] # all fist dimension  
# array([[0, 1, 2], [3, 4, 5]])  
  
# fist dimension, first row of first dimension  
b[0,0]  
# array([0, 1, 2])  
  
# all first dimension  
# b[0, ...]  
# array([[0, 1, 2], [3, 4, 5]])  
  
# all dimension, then second columns  
#b[..., 1]  
#array([[ 1, 4], [ 7, 10]])  
  
# all dimension, then second rows  
#b[:,1]  
# array([[ 3, 4, 5], [ 9, 10, 11]])  
  
# first dimension, all rows, then all column except the first  
#b[0, :, -1]  
#array([2, 5])  
  
#fist dimension, all rows but reversed, then all columns except the first  
#b[0,::-1, -1]  
#array([5, 2])
```

### Numpy ใน OpenCV

ถ้าหากว่าเราแปลงสีโทเนเทา เป็นสี BGR (Blue Green Red) ก็ควรเป็นอาร์เรย์ขนาด 3 มิติ ตัวอย่างต่อไปนี้จะแสดงการแปลงเป็นสี BGR และทดสอบหาขนาด (shape)

Code 7.

---

```
import cv2  
  
img = cv2.imread('d:/chess.jpg')  
  
print(img.shape)  
#print (395, 550, 3)
```

หมายเหตุ การใช้ cv2 ไม่ได้หมายความว่าป็นรุ่น OpenCV 2 ณ ปัจจุบันใช้ OpenCV 3 แล้ว แต่ที่เรียก ว่า cv2 เพราะใช้กับ OpenCV ที่เขียนด้วย C++ แต่ถ้าใช้ cv จะมาจาก OpenCV ที่เขียนด้วย C ดั้งนั้นแล้ว ตัวเลขหลัง cv จึงหมายถึงการคอมไพล์มาจากภาษา C หรือ C++

ดังเห็นในตัวอย่างแล้วว่า ภาพมีขนาด 3 มิติ ซึ่งในที่นี้คือ (395, 550, 3) ซึ่งหมายความว่า ภาพมีขนาด ยาว x กว้าง = 395 x 550 และในมิติที่สาม มี 3 คอลัมน์ ซึ่งข้อมูลสุดท้าย คือค่า B-G-R

### การอ่านภาพ

OpenCV สนับสนุนการอ่านและเขียนภาพ ทั้งที่มาจากภาพ ในรูปแบบต่างๆ เช่น BMP, PNG, JPEG, และ TIFF คำสั่งที่ใช้ อ่านคือ `imread()` และเขียนคือ `imwrite()` และทั้งหมดนี้ต้องมาจากการนำเข้า OpenCv ก่อน

#### Code 8.

---

```
import cv2
import numpy as np

img = cv2.imread('d:/chess.jpg')

cv2.imshow('img', img)
cv2.waitKey()
cv2.destroyAllWindows()
```

การอ่านของ OpenCV โดยค่าปริยายจะอ่านในรูปแบบ BGR แม้ว่าภาพที่อ่านจะเป็นภาพโทนเทาก็ตาม พื้นที่เก็บสี BGR เหมือนกับ RGB (red-green-blue) ต่างเพียงลำดับสีกลับกัน

กรณีที่เราสามารถกำหนดชนิดการอ่าน ได้นอกจากการอ่านแบบค่าปริยาย โดยใส่ค่า เป็นตัวเลขแทน หรือค่า อื่นๆ เหมือนกับตัวอย่างที่ผ่านมา

```
IMREAD_ANYCOLOR = 4
IMREAD_ANYDEPTH = 2
IMREAD_COLOR = 1
IMREAD_GRAYSCALE = 0
IMREAD_LOAD_GDAL = 8
IMREAD_UNCHANGED = -1
```

### คลาส Mat

จากตัวอย่างข้างต้น การอ่านภาพ แล้วเก็บในตัวแปร `img` ซึ่งมาจากคลาส `Mat` คลาสนี้จึงเป็นที่เก็บของภาพ และภาพนี้อยู่ในรูปเมตริก ตัวอย่างต่อไปจะทดสอบอ่านภาพ ในรูปแบบสีโทนเทา และแสดงเมตริก

#### Code 9.

---

```
import cv2

img = cv2.imread('d:/chess.jpg', cv2.IMREAD_GRAYSCALE)
print(img.shape)
#print (359, 550)
```

ตัวอย่างภาพ `chess.jpg` นี้เป็นภาพขนาดพิกเซล สูง 395 (vertical) และกว้าง 550 (horizontal) และค่าข้อมูลข้างในแทนตัวเลข ตั้งแต่ 0-255 ซึ่งแทนสีโทนสีเทา

## ใช้ Numpy ดำเนินการกับภาพ

ภาพใน OpenCV มีอาร์เรย์ 2 มิติ สำหรับภาพโทนเทา เช่น ภาพสีขาว เมื่ออ่าน `img[0,0]` เป็นการระบุตำแหน่งแรก แกน Y หรือแนวตั้งตรง และตำแหน่งที่สองแทนตำแหน่ง X หรือแนวนอน ณ จุดนี้จะเก็บค่า 255

ตัวอย่างต่อไปนี้เป็น การอ่านภาพขนาดเล็กมาก ขนาด 10x10 อ่านในลักษณะโทนเทา (0 – 255) ให้สังเกตว่ามี 10 แถว และ 10 คอลัมน์

### Code 10.

---

```
import numpy as np
img = cv2.imread('d:/icon.png',0)
print(img.shape)
print(img)

...
(10, 10)
[[255 255 255 255 247 247 255 255 255 255]
 [255 255 181 20 0 0 20 182 255 255]
 [255 180 0 0 5 5 0 0 182 255]
 [255 18 0 0 106 106 0 0 21 255]
 [247 0 0 0 39 39 0 0 0 247]
 [247 0 0 0 126 126 0 0 0 247]
 [255 19 0 0 128 128 0 0 20 255]
 [255 181 0 0 2 2 0 0 181 255]
 [255 255 181 19 0 0 20 182 255 255]
 [255 255 255 255 247 247 255 255 255 255]]
...
```

จากการอ่านรูปนี้ พอจินตนาการรูปได้ว่า ขอบรูปที่ขาว เพราะมีค่าเป็นสี 255 และภายในเป็นสีขาว เพราะมีสีใกล้เคียง 0 หากว่าเราต้องการจะพิมพ์ที่ละตัวที่ละบรรทัด เราต้องเข้าใจว่า เราต้องพิมพ์ แถวก่อน ต่อมาพิมพ์แถวมีคอลัมน์ ซึ่งแต่พิมพ์แต่ละตัวหมายถึงพิมพ์แต่ละคอลัมน์ ดูได้จากตัวอย่างต่อไปนี้

### Code 11.

---

```
import cv2
import numpy as np
img = cv2.imread('d:/icon.png',0)

for row in img:
    for col in row:
        print(col)
```

เมื่อเปรียบเทียบกับ การพิมพ์ก่อนหน้ากับการทดลองพิมพ์นี้ จะพบว่า ตัวเลข เป็น 255 ตัวที่ห้า เป็น 247 และตัวที่ 13 เป็น 181 และตัวสุดท้ายเป็น 255

หากว่าเราจะกลับสีขาวเป็นดำ และดำเป็นขาว โดยกำหนดว่า ถ้า สีมาทางสี 255 ให้เปลี่ยนเป็นสี 0 และถ้าสีมาทาง 0 ให้เปลี่ยนเป็น สี 255

การเข้าถึงอาร์เรย์ สองมิติต้องใช้ สองวงเล็บ เช่น รอบแรกเป็นรอบที่ *i* และรอบที่สองเป็น *j* ดังนั้นตำแหน่งปัจจุบันจึงเป็น ตำแหน่งอาร์เรย์ที่ [*i*, *j*] ในการทำงานนี้จะต้องใช้ `enumerate` เพื่อดึงค่า *i* และ *j* ออกมา ดูตัวอย่างต่อไปนี้ ประกอบ

Code 12.

```
import cv2
import numpy as np
img = cv2.imread('d:/icon.png',0)
newImg = np.array(img) #copy image

for i, row in enumerate(newImg):
    for j, col in enumerate(row):
        if (col>240):
            newImg[i,j]=0
        if (col<10):
            newImg[i,j]=255

print(newImg)
'''
[[ 0  0  0  0  0  0  0  0  0]
 [ 0  0 181 20 255 255 20 182  0]
 [ 0 180 255 255 255 255 255 182  0]
 [ 0  18 255 255 106 106 255 255 21]
 [ 0 255 255 255 39 39 255 255 255]
 [ 0 255 255 255 126 126 255 255 255]
 [ 0  19 255 255 128 128 255 255 20]
 [ 0 181 255 255 255 255 255 181  0]
 [ 0  0 181 19 255 255 20 182  0]
 [ 0  0  0  0  0  0  0  0  0]]
'''
```

### การตัดลอกภาพบางส่วน

ดังเห็นแล้วว่า เราตัดแปลงค่าของสีได้ ถ้าอย่างนี้เราก็ตัดแปลงรูปภาพในลักษณะอื่นๆ ได้อีก เช่น การตัดลอกบางส่วนของภาพ การตัดลอกบางส่วนของภาพจะต้องทราบขนาดของภาพก่อน ซึ่งหาด้วยคำสั่ง shape ต่อมาเราก็เลือกที่บางส่วนของภาพที่ต้องการจะตัดลอก ใช้ การตัดลอกอาร์เรย์ บางส่วน

ตัวอย่างต่อไปนี้ ทำตามขั้นตอนข้างต้น เมื่อตัดลอกไปเก็บที่ตัวแปรใหม่แล้ว ให้นำไปแสดงผลต่อไป ด้วย imshow และรอคำสั่งกดปุ่มใดๆ บนคีย์บอร์ด จึงจะปิดหน้าต่างรูปภาพ

Code 13.

```
import cv2
import numpy as np
img = cv2.imread('d:/sea.jpg',0)

print(img.shape) #300,400

new_img = img[:200,:450] #h:200, w:450

cv2.imshow('new_img', new_img)
cv2.waitKey()
cv2.destroyAllWindows()
```

### การภาพกลับด้าน

การทำให้ภาพหมุนซ้าย-ขวา หรือกลับด้าน ทำเพียงเรียงค่าสีใหม่ เช่น เรียงจากหลังไปหน้า ทำให้ภาพกลับด้านได้ หากยังจำกันได้ว่า การเรียงค่าถอยหลัง ใช้[::-1] ซึ่งแปล เลือกทั้งหมด(มิติแรก แล้วเรียงถอยหลัง ที่นี้มาทดลองการทํากับภาพสองมิติ (โทนเทา) ให้ภาพกลับด้าน ได้ดังตัวอย่างต่อไปนี้

Code 14.

```
import cv2
import numpy as np
img = cv2.imread('d:/sea.jpg', 0)
new_img = img[:,::-1] #right to left

cv2.imshow('new_img', new_img)
cv2.waitKey()
cv2.destroyAllWindows()
```



รูป 1 รูปดั้งเดิม (ซ้าย) รูปกลับด้าน (ขวา)<sup>1</sup>

#### การภาพกลับสัดส่วน

การทำให้สัดส่วนไม่เหมือนเดิมเราเคยใช้ได้จากคำสั่ง reshape เช่น จัดสัดส่วนเดิม เป็น (300, 450) จะกลับให้เป็น (450, 300) ก็ทำได้ด้วยคำสั่ง reshape นี้

Code 15.

```
import cv2
import numpy as np
img = cv2.imread('d:/sea.jpg', 0)
print(img.shape) #(300, 450)

img_reshape = img.reshape(450,300)
cv2.imshow('reshape', img_reshape)

cv2.waitKey()
cv2.destroyAllWindows()
```

ถ้าอย่างให้มีส่วนอื่นๆ ก็ทำได้อีก แต่จำนวน ข้อมูลต้อง ต้องเป็นสัดส่วนกัน และจำนวนที่เท่ากัน ตรวจสอบได้จาก การคูณกัน แล้ว มีค่าเท่าเดิม เช่น ของเดิม คือ (450, 300) แปลงไปเป็น (200, 675) ผลคูณของใหม่เท่ากับของเดิม

---

<sup>1</sup> นำมาจาก <https://www.shine.cn/archive/metro/entertainment-and-culture/Flights-securing-chances-for-full-moon-view-among-popular-travel-products-as-locals-plan-for-upcoming-holiday/shdaily.shtml>





รูป 2 สัดส่วนใหม่ 200 x 675

ในตัวอย่างก่อนหน้านี การแปลงสี โทนขาว ให้เป็นโทนดำ ด้วยการวนซ้ำให้มิติของอาร์เรย์ แล้วทำให้ชีวิตยุ่งยาก ยังมีอีกวิธีหนึ่ง คือการแปลงข้อมูลมิติอาร์เรย์ ให้อยู่ในมิติเดียว หรือ แถวเดียว แล้วทำการแปลงค่าสี เมื่อแปลงเสร็จแล้ว ค่อยแปลงสัดส่วน เป็นสัดส่วนเดิมอีกครั้ง

### การทำภาพกลับสี (อีกครั้ง)

ตัวอย่างต่อไปนี้เป็นการแปลงโทนสี โดยใช้ np.array(img) โดย img เป็นอาร์เรย์ของข้อมูลภาพ ที่ทำให้มิติของภาพ เป็นมิติเดียว แล้วแต่งสี ก่อนที่จะตัดแปลงเป็นมิติภาพเดิม ฟังก์ชันที่ทำให้เรียงเป็นมิติเดียวคือ flatten( )



รูป 3 รูปดั้งเดิม (ซ้าย) รูปกลับสี (ขวา)

Code 16.

```
import cv2
import numpy as np
img = cv2.imread('d:/sea.jpg', 0)
print(img.shape)

img_flat = img.flatten()
print(img_flat[0]) #print 11

index = 0
for i in img_flat:
    if i>230: img_flat[index] = 0
    if i<30 : img_flat[index] = 255
    index += 1

print(img_flat[0]) #print 255
#img_flat.shape =(300,450)
img_flat.resize(300,450)
```

```
cv2.imshow('new_img', img_flat)
cv2.waitKey()
cv2.destroyAllWindows()
```

จากตัวอย่างนี้จะเห็นว่า เราใช้การแปลงสี กลับจากโทนเทาเป็นโทนขาว ในการแปลงขนาดเมตริกซ์ แทนที่จะใช้ **reshape** เราใช้ **shape** หรือ **resize()** แทน

ตาราง 1 สรุปการใช้คำสั่งต่างๆ ของ NumPy บบนี้

คำสั่ง	ผลลัพธ์	ความหมายเพิ่มเติม
<pre>a = np.array([1,2,3]) print(a) print(a.size) print(a.dtype)</pre>	<pre>[1 2 3] 3 int32</pre>	
<pre>b = np.arange(4) print(b)</pre>	<pre>[0, 1, 2, 3]</pre>	
<pre>c = np.arange(2, 4) print(c)</pre>	<pre>[2, 3]</pre>	
<pre>d = np.arange(10) print(d[0]) print(d[:7]) print(d[:-2]) print(d[::-1])</pre>	<pre>0 [0 1 2 3 4 5 6] [0 1 2 3 4 5 6 7] [9 8 7 6 5 4 3 2 1]</pre>	
<pre>e = np.arange(5, dtype=np.uint64) print(e) print(e.dtype)</pre>	<pre>[0, 1 2 3 4] unit64</pre>	un-sign integer 64 bit
<pre>f = np.arange(12).reshape(2,2,3) print(f.shape) print(f)  print(f[0,0,0]) print(f[0,0]) print(f[0])  print(f[0,:,:]) print(f[:,0])  print(f[:,0,:]) print(f[...,:])</pre>	<pre>(2, 2, 3) [[[ 0  1  2]  [ 3  4  5]]  [[ 6  7  8]  [ 9 10 11]]]  0 [0 1 2] [[ 0  1  2]  [ 3  4  5]]  = e[0] = e[0,...] = e[0,:] [[0 1 2]  [6 7 8]]  = e[:,0] [[0 3]  [6 9]]</pre>	<p>first dimension : all second dimension: index 0</p> <p>dimension : all only first index</p>