แนะนำ NumPy เพื่องาน OpenCV

โปรแกรมไพธอน (Python) เป็นโปรแกรมเอนกประสงค์ตัวหนึ่งที่ได้รับความนิยมอย่างแปรเหลาย สร้างโดย Guido Van Rossum โดยเขาต้องการให้งานเขียนโปรแกรมได้ง่ายๆ การทำงานของไพธอนทำงานผ่าน ตัวแปลภาษาที่เรียกว่า อินเตอร์พลีเตอร์ (Interpreter) ซึ่งหมายความว่าจะต้องทำการแปลทุกครั้งที่ให้โปรแกรมทำงาน ซึ่งแน่นอนว่าโปรแกรมย่อมทำงานได้ช้ากว่า โปรแกรมภาษาที่ใช้การแปลผ่าน คอมไพล์เลอร์ (Complier)

NumPy เป็นชุดคำสั่งสำหรับการคำนวณทางตัวเลข ที่สร้างขึ้นจากภาษาไพธอน เมื่อติดตั้ง Anaconda แล้วจะมี NumPy มาด้วย มีฟังก์ชันทีสำคัญคืออาร์เรย์ที่เก็บ ชุดตัวเลข หรือเมตริก ที่นำไปแทน ชุดของเม็ดสี หรือ พิเซล (pixel)

ติดตั้ง Numpy และ OpenCV ผ่าน CLI (Command Line Interface)

หากว่าติดตั้ง Python แล้ว ให้ติดตั้ง pip ซึ่งเป็นตัวจัดการโมดูลสำหรับ Python วิธีการ ให้เข้าสู่ CLI แล้วมีลำดับดังนี้

```
>python -m pip install -U pip
>pip install numpy
>pip install opencv-contrib-python
```

เมื่อติดตั้งเสร็จ ให้ทดสอบ OpenCV ดังนี้ (ยังอยู่ใน CLI)

```
>python
>>>import cv2
```

หากว่าไม่มีอะไรแจ้งความผิดพลาดแสดงว่า การใช้งาน OpenCV ก็ถือว่าได้ติดตั้งเสร็จสิ้นแล้ว

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

การสร้างอาร์เรย์ (Array)

อาร์เรย์ของ NumPy นำมาใช้เพื่อการคำนวณจะมีประสิทธิภาพมากกว่า List ของไพธอน โดยเฉพาะข้อมูลที่มีขนาดใหญ่ ข้อมูลที่มีหลายมิติ เช่นในข้อมูลในรูปเมตริก ดังนั้นทุกโครงสร้างของ OpenCV จะแปลงเป็นอาร์เรย์ของ NumPy เช่น การอ่านภาพ 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 ใช้ตามนรูปแบบข้างต้น

- start แทนการค่าเริ่มต้น ซึ่งไม่ใส่จะหมายถึงเริ่มจากศูนย์
- stop แทนค่าที่จะหยุด
- step แทน ค่าที่เลื่อนที่ละเท่าใด ซึ่งไม่ใส่หมายถึง เพิ่มที่ละหนึ่ง
- และ dtype คือชนิดข้อมูล ถ้าไม่ใส่จะหมายถึง จะใช้ชนิดตามข้อมูลที่ใน start และ stop

ตาราง 1 ชนิดข้อมูลในอาร์เรย์

ชนิดข้อมูล	หมายถึง
bool_	ค่าจริง/เท็จ เก็บในรูป ไบท์ (Byte)
int8, int16, int32, int64	ค่าตัวเลขจำนวนเต็ม เช่น int64, int32
uint8, uint16, uint32, uint64	ค่าตัวเลข ไม่มีเครื่องหมายลบ เช่น unit8 เก็บค่า ตั้งแต่ 0 - 255
float_	ค่าทศนิยม หรือ float64 แต่สามารถระบุขนาดอื่นๆ ได้เช่น float16, float32

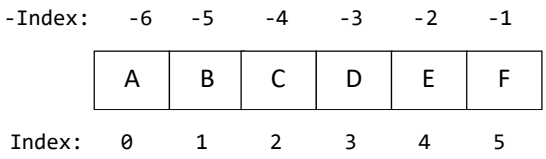
การอ่านค่าข้อมูลภายในอาร์เรย์

เมื่อสร้างข้อมูลอาร์เรย์ได้แล้ว ทีนี้มาถึงคราที่ต้อง อ่านอาร์เรย์ สำหรับ Python มีลักษณะการอ่านอาร์เรย์ที่ไม่เหมือนภาษาอื่นๆ 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

เพื่อทำความเข้าใจ ให้ดีขึ้น ให้พิจารณา จาก**รูปที่ 1** ถ้าให้อาร์เรย์ มีค่า A - G ค่าดัชนีจะเริ่มจาก 0 ถึง 5 เพราะมี 6 ค่า ซึ่งไพธอน อ่าน ย้อนหลังได้ โดยเริ่ม จาก -1 เช่น ถ้าต้องการอ่านค่า F ใช้เลข -1 ได้

ให้ดูจาก ตัวอย่าง **Code 2** แสดงการอธิบายที่ผ่านมานี้



รูป 1 อาร์เรย์ 6 ค่า

Code 2.

```
import numpy as np
a = np.array(['a', 'b', 'c', 'd', 'e', 'f'])
print([a[-1]) # f
```

เพื่อประกอบความเข้าใจการอ่านอาร์เรย์ให้ดีขึ้น ให้ตัวอย่างโปรแกรมต่อไป ทดสอบความเข้าใจ การอ่านอาร์เรย์ในลักษณะต่างๆ

Code 3.

```
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

ตัวอย่างต่อไปนี้แสดงการสร้างอาร์เรย์หลายมิติ และการเข้าถึงอาร์เรย์มิติต่างๆ

Code 4.

```
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 5.

```
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 บิท ซึ่งมีจำนวนตัวเลข ตั้งแต่ 0 – 255 และมีค่าเป็นศูนย์หมดหมายถึงสีดำ

ตัดแต่งขนาดอาร์เรย์

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

Code 6.

```
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 จากที่พบว่าว่าเราอ่านแบบระบุแต่ละมิติไปแล้ว ในตัวอย่างต่อไปนี้ จะแสดงการอ่านมิติ ด้วยการระบุ สามจุด(. . .) และ คอลอนน์ (:) ซึ่งหมายถึงเลือกทั้งหมด

Code 7.

```
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

ถ้าหากว่าเราแปลงสีโทนเทา เป็นสี BGR (Blue Green Red) ก็ควรเป็นอาร์เรย์ขนาด 3 มิติ ตัวอย่างต่อไปนี้แสดงการแปลงเป็นสี BGR และทดสอบหาขนาด (shape)

Code 8.

```
import cv2

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

print(img.shape)
#print (395, 550, 3)
```

หมายเหตุ *การใช้ cv2 ไม่ได้หมายความว่าเป็นรุ่น OpenCV 2 ณ ปัจจุบันใช้ OpenCV 3 แล้ว แต่ที่เรียก ว่า cv2 เพราะใช้กับ OpenCV ที่เขียนด้วย C++ แต่ถ้าใช้ cv จะมาจาก OpenCV ที่เขียนด้วย C ดังนั้นแล้ว ตัวเลขหลัง cv จึงหมายถึงการคอมไฟล์มาจากภาษา 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 คำสั่งที่ใช้อ่านคือ imread( ) และเขียนคือ imwrite( ) และทั้งหมดนี้ต้องมาจากการนำเข้า OpenCv ก่อน

Code 9.

```
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

กรณีที่อ่านเราสามารถกำหนดชนิดการอ่าน ได้นอกจากการอ่านแบบค่าปริยาย โดยใส่ค่า เป็นตัวเลขแทน หรือค่าอีนัม 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 10.

```
import cv2

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

การอ่านภาพโทนเทา :

ให้ IMEAD_GRAYSCALE หรือ 0 แทนได้ เช่น

```
img = cv2.imread('d:/chess.jpg', 0)
```

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

ใช้ Numpy ดำเนินการกับภาพ

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

ตัวอย่างต่อไปนี้เป็นการอ่านภาพขนาดเล็กมาก ขนาด 10x10 อ่านในลักษณะโทนเทา (0 – 255) ให้สังเกตว่า มี 10 แถว และ 10 คอลัมน์

Code 11.

```
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 หากว่าเราต้องการจะพิมพ์ที่ละตัวทีละบรรทัด เราต้องเข้าใจว่า เราต้องพิมพ์ แถวก่อน ต่อมาพิมพ์แถวมีคอลัมน์ ซึ่งแต่พิมพ์แต่ละตัวหมายถึงพิมพ์แต่ละคอลัมน์ ดูได้จากตัวอย่างต่อไปนี้

Code 12.

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

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

เมื่อเปรียบเทียบกับ การพิมพ์ก่อนหน้านี้กับการการทดลองพิมพ์นี้ 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 สองวนซ้ำ เช่น รอบแรกเป็นรอบที่ i และรอบที่สองเป็น j ดังนั้นตำแหน่งปัจจุบันจึงเป็นตำแหน่งอาร์เรย์ที่ [i, j] ในการทำงานนี้จะต้องใช้ enumerate เพื่อดึงค่า ดัชนี ให้ได้ค่า i และ j ออกมา ดูตัวอย่างต่อไปนี้ประกอบ

Code 13.

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

การคัดลอกภาพบางส่วน

ดังเห็นแล้วว่า เราดัดแปลงค่าของสีได้ ถ้าอย่างนี้เราก็ดัดแปลงรูปภาพในลักษณะอื่นๆ ได้อีก เช่น การคัดลอกบางส่วนของภาพ การคัดลอกบางส่วนของภาพจะต้องทราบขนาดของภาพก่อน ซึ่งหาด้วยคำสั่ง shape ต่อมาเราก็เลือกที่บางส่วนของภาพที่ต้องการจะคัดลอก ใช้ การคัดลอกอาร์เรย์ บางส่วน

ตัวอย่างต่อไปนี้ ทำตามขั้นตอนข้างต้น เมื่อคัดลอกไปเก็บที่ตัวแปรใหม่แล้ว ให้นำไปแสดงผลต่อไป ด้วย imshow และรอคำสั่งกดปุ่มใดๆ บนคีย์บอร์ด จึงจะปิดหน้าต่างรูปภาพ

Code 14.

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

จากตัวอย่างนี้ ใช้การคัดลอก รูป sea.img ที่แปลงเป็นโทนเทาแล้ว ที่มีชื่อตัวแปร img แล้วทำการคัดลอก จากตำแหน่ง 0:200 ในมิติแรก (แถว) และ 0:450 ในมิติที่สอง (คอลัมน์) มาเก็บในตัวแปรใหม่ ชื่อ new_img ต่อด้วยการแสดงผลรูปใหม่ มี ชื่อหน้าต่าง ว่า new imgage (Window's title) จากรูปใหม่ที่ new_img

จากแนวคิดลอกรูปนี้ ถ้าต้องการคัดลอก บางส่วน ของรูป แต่ ยังคงขนาดรูปเดิมอยู่ จะต้องสร้างรูปใหม่ให้มีขนาดเท่ากับรูปเดิมซึ่งตรวจสอบให้ได้ด้วยคำสั่ง shape ให้เป็นรูปที่มีพื้นสีทั้งหมดเป็นสีดำ ด้วยคำสั่ง np.zeros((300, 400), uint8)

การปิดหน้าต่าง Windows :

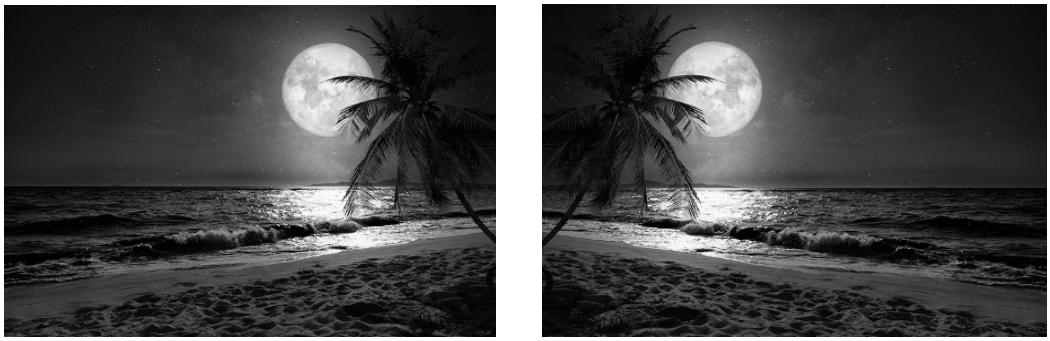
ให้ กดคีย์บอร์ด ใดๆ จากคำสั่ง c2.waitKey( ) ขณะที่กำลังทำงานกับหน้าต่างภาพ (Active windows) ห้ามใช้เม้าท์กด ปุ่มกากบาท ที่มุมบนสุดขาวสุดของหน้าต่าง เพราะจะทำให้โปรแกรม ไพธอน ค้างได้

การภาพกลับด้าน

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

Code 15.

```
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 2 รูปดั้งเดิม (ซ้าย) รูปกลับด้าน (ขวา)[1]

การภาพกลับสัดส่วน

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

Code 16.

```
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 ตรวจสอบได้จากการคูณกัน แล้ว มีค่าเท่าเดิม เช่น ของเดิม คือ (450, 300) แปลงไปเป็น (200, 675) ผลคูณของใหม่เท่ากับของเดิม

[1] 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 3 สัดส่วนใหม่ 200 x 675

ในตัวอย่างก่อนหน้านี้ การแปลงสี โทนขาว ให้เป็นโทนดำ ด้วยการวนซ้ำให้มิติของอาร์เรย์ แล้วทำให้ชีวิตยุ่งยาก ยังมีอีกวิธีหนึ่ง คือการแปลงข้อมูลมิติอาร์เรย์ ให้อยู่ในมิติเดียว หรือ แถวเดียว แล้วทำการแปลงค่าสี เมื่อแปลงเสร็จแล้ว ค่อนแปลงสัดส่วน เป็นสัดส่วนเดิมอีกครั้ง

การทำภาพกลับสี (อีกครั้ง)

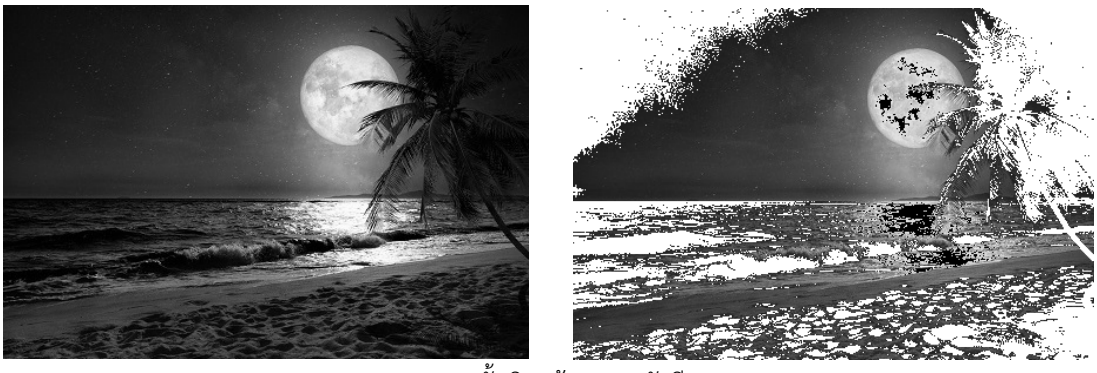
ตัวอย่างต่อไปนี้เป็นการแปลงโทนสี โดยใช้ np.array(img) โดย img เป็นอาร์เรย์ของข้อมูลภาพ ที่ทำให้มิติของภาพ เป็นมิติเดียว แล้วแต่งสี ก่อนที่จะดัดแปลงเป็นมิติภาพเดิม ฟังก์ชันที่ทำให้เรียงเป็นมิติเดียวคือ flatten( )

Code 17.

```
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 ในการแปลงขนาดเมติกซ์ แทนที่จะใช้ **reshape** เราใช้ shape หรือ **resize( )** แทน

ยังมีฟังก์ชันที่คล้ายๆ กับ flatten( ) คือ ravel( ) ใช้งานได้เหมือนกัน ต่างกันตรงที่ ravel( ) ไม่ทำสำเนาข้อมูลใหม่คืนมาให้ แสดงว่าต้นฉบับจะถูกจัดเรียงข้อมูลใหม่ในแนวราบ ซึ่งใช้ ravel( ) ก็ดีที่ไม่ต้องสร้างหน่วยความจำเพิ่ม แต่ต้องระมัดระวังกับข้อมูลต้นฉบับที่จะถูกดัดแปลง



รูป 4 รูปดั้งเดิม (ซ้าย) รูปกลับสี (ขวา)

ตาราง 2 สรุปการใช้คำสั่งต่างๆ ของ NumPy บทนี้

คำสั่ง	ผลลัพธ์	ความหมายเพิ่มเติม
a = np.array([1,2,3]) print(a) print(a.size) print(a.dtype)	 [1 2 3] 3 int32	
b = np.arange(4) print(b)	 [0, 1, 2, 3]	
c = np.arange(2, 4) print(c)	 [2, 3]	
d = np.arang(10) pirnt(d[0]) print(d[:7] print(d[:-2]) print(d[::-1])	 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]	
e = np.arange(5, dtype=np.uint64) print(e) print(e.dtype)	 [0, 1 2 3 4] unit64	 un-sign integer 64 bit

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,:])
	
	(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]
		
		first dimension : all
		second dimension: index 0

`print(f[...,0])`	`[[0 3]` ` [6 9]]`	`dimension : all` `only first index`
`img.flatten()` `img.ravel()`		`return a new flatten img` `return the flatten img`

แบบฝึกหัด

1. ให้เลือกรูป ใดรูปหนึ่ง เช่น รูป คน ให้ เลือกเฉพาะหัว หรือใบ หน้า ที่ยังคงรูปเดิม หรือคัดลอกเฉพาะส่วนที่ต้องการ แต่ส่วนอื่น เป็นพื้นที่ขาว หรือ ดำทั้งหมด
2. ให้เลือกรูป หนึ่ง แล้ว กลับ ซ้ายเป็นขวา และกลับบนลงล่าง
3. ให้เลือกรูป หนึง แล้ว แปลงขนด จากกว้างเป็น ยาว และยาวเป็นกว้าง (หมุน 90 องศา) โดยังคง รูปเหมือนเดิม