OpenCV 기본 배열 연산

담당교수: 김민기

목 차

1. 기본 배열(array) 처리 함수
2. 채널 처리 함수
3. 산술 연산 함수
4. 절대값, 최대값, 최소값 관련 함수
5. 통계 관련 함수
6. 행렬 연산 함수

1. 기본 배열 처리 함수

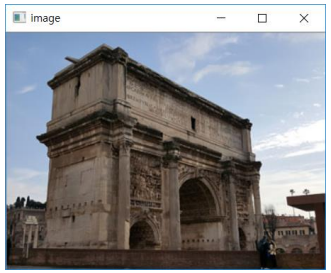
- ❖Python에서 배열 처리를 위한 자료형
  - 리스트(list), 튜플(tuple), 사전(dictionary)
  - 1차원 배열(벡터), 2차원 배열(행렬)
- ❖명도 영상
  - 2차원 배열로 표현
- ❖컬러 영상
  - 빨강(R), 녹색(G), 파랑(B) 각기 독립적인 2차원 배열
  - 2차원 배열 3개로 컬러 영상을 표현
    → 3차원 배열 사용

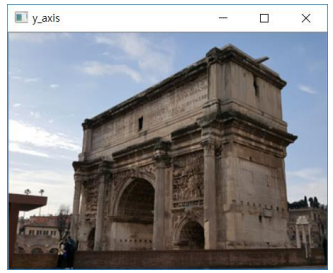
기본 배열 처리 함수

함수 설명	
cv2.flip(src, flipCode[, dst]) → dst ■설명: 입력된 2차원 배열을 수직, 수평, 양축으로 뒤집는다.	
인수 설명	■src , dst 입력 배열, 출력 배열 ■flipCode 배열을 뒤집는 축 - 0 : x축을 기준으로 위아래로 뒤집는다. - 1 : y축을 기준으로 좌우로 뒤집는다. - -1 : 양축(x축, y축 모두)을 기준으로 뒤집는다.
cv2.repeat(src, ny, nx[, dst]) → dst ■설명: 입력 배열의 반복된 복사본으로 출력 배열을 채운다.	
인수 설명	■src, dst 입력 배열, 출력 배열 ■ny, nx 수직 방향 , 수평방향 반복 횟수
cv2.transpose(src[, dst]) → dst ■설명: 입력 행렬의 전치 행렬을 출력으로 반환한다.	
인수 설명	■src, dst 입력 배열, 출력 배열

예제 5.1.1 행렬 처리 함수 – 01.mat_array.py

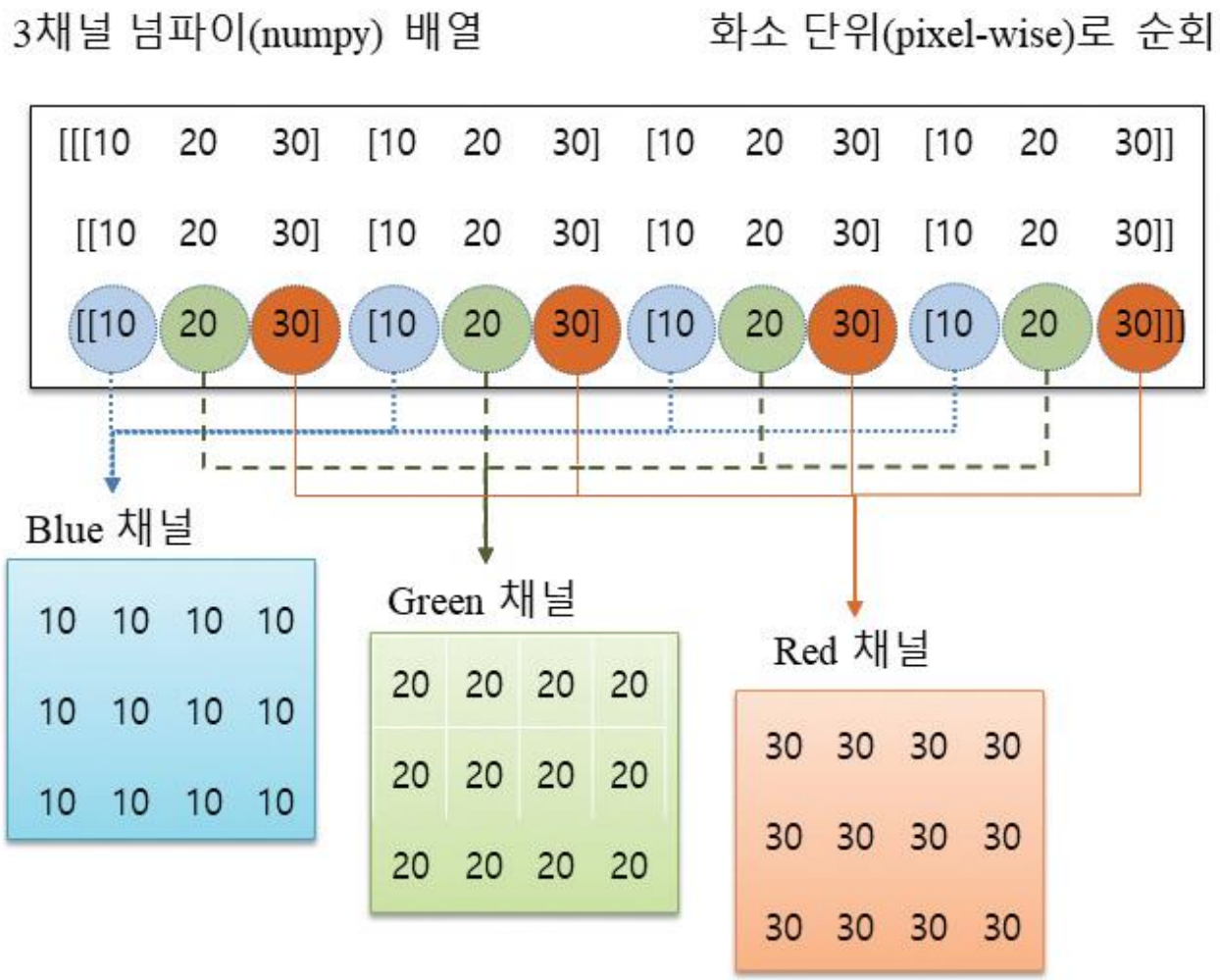
```
import cv2

image = cv2.imread("images/flip_test.jpg", cv2.IMREAD_COLOR)
if image is None: raise Exception("영상파일 읽기 오류 발생")        # 예외 처리

x_axis = cv2.flip(image, 0)                                        # x축 기준 상하 뒤집기
y_axis = cv2.flip(image, 1)                                        # y축 기준 좌우 뒤집기
xy_axis = cv2.flip(image, -1)
rep_image = cv2.repeat(image, 1, 2)                                # 반복 복사
trans_image = cv2.transpose(image)                                 # 행렬 전치

## 각 행렬을 영상으로 표시
titles = ['image', 'x_axis', 'y_axis', 'xy_axis', 'rep_image', 'trans_image']
for title in titles:
    cv2.imshow(title, eval(title))
cv2.waitKey(0)
```





2. 채널 처리 함수

❖채널 개념

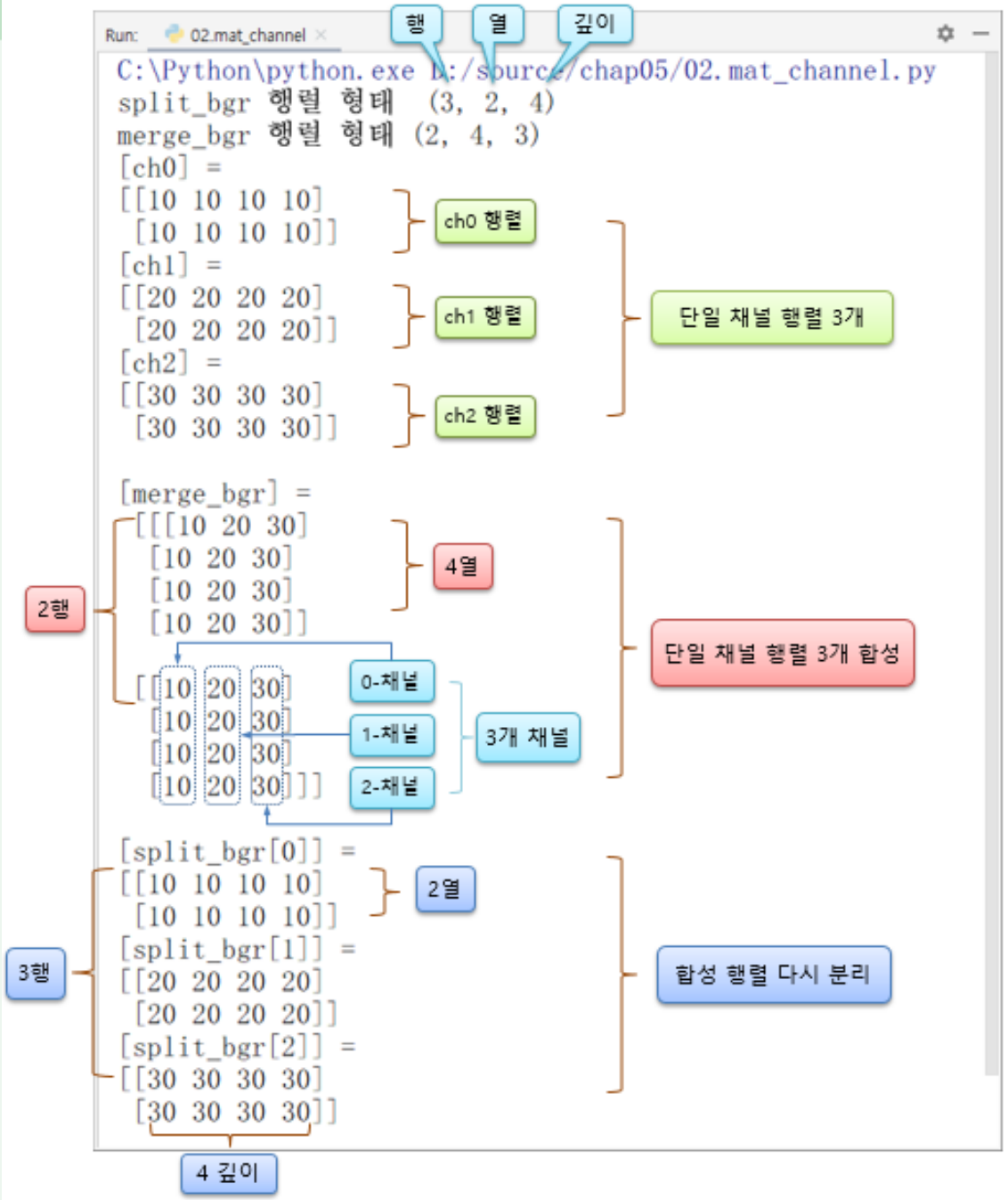


채널 처리 함수

함수 설명		
cv2.merge(mv[, dst]) → dst ■ 설명: 여러 개의 단일채널 배열을 다채널 배열로 합성한다.		
인수 설명	■ mv	합성될 입력 배열 혹은 벡터, 합성될 단일채널 배열들의 크기와 깊이(depth)가 동일해야 함
	■ dst	입력 배열과 같은 크기와 같은 깊이의 출력 배열
cv2.split(m[, mv]) → mv ■ 설명: 다채널 배열을 여러 개의 단일채널 배열로 분리한다.		
인수 설명	■ m	입력되는 다채널 배열
	■ mv	분리되어 반환되는 단일채널 배열들의 벡터

예제 5.2.1 채널 분리 및 합성 – 02.mat_channel.py

```python
import numpy as np
import cv2

## numpy.ndarray를 이용해 행렬 생성 및 초기화 방법
ch0 = np.zeros((2, 4), np.uint8) + 10
ch1 = np.ones((2, 4), np.uint8) * 20
ch2 = np.full((2, 4), 30, np.uint8)

list_bgr = [ch0, ch1, ch2]
merge_bgr = cv2.merge(list_bgr)
split_bgr = cv2.split(merge_bgr)

print("split_bgr 행렬 형태", np.array(split_bgr).shape)
print("merge_bgr 행렬 형태", merge_bgr.shape)
print("[ch0] = \n%s" % ch0)
print("[ch1] = \n%s" % ch1)
print("[ch2] = \n%s\n" % ch2)
print("[merge_bgr] = \n %s\n" % merge_bgr)

print("[split_bgr[0]] =\n%s " % split_bgr[0])
print("[split_bgr[1]] =\n%s " % split_bgr[1])
print("[split_bgr[2]] =\n%s " % split_bgr[2])
```



예제 5.2.2 컬러 채널 분리 – 03.image_channels.py

```python
import cv2

image = cv2.imread("images/color.jpg", cv2.IMREAD_COLOR)          # 영상 읽기
if image is None: raise Exception("영상파일 읽기 오류")            # 예외 처리
if image.ndim != 3: raise Exception("컬러 영상 아님")    # 예외 처리-컬러 영상 확인

bgr = cv2.split(image)                                  # 채널 분리: 컬러 영상 → 3채널 분리
# blue, green, red = cv2.split(image)                   # 3개 변수로 반환받기 가능
print("bgr 자료형:", type(bgr), type(bgr[0]), type(bgr[0][0][0]) )
print("bgr 원소개수:" len(bgr))

## 각 채널을 윈도우에 띄우기
cv2.imshow("image", image)
cv2.imshow("Blue channel" , bgr[0])                     # Blue 채널
cv2.imshow("Green channel", bgr[1])                     # Green 채널
cv2.imshow("Red channel" , bgr[2])                      # Red 채널
# cv2.imshow("Blue channel" , image[:,:,0])             # 넘파이 객체 인덱싱 방식
# cv2.imshow("Green channel", image[:,:,1])
# cv2.imshow("Red channel" , image[:,:,2])
cv2.waitKey(0)
```

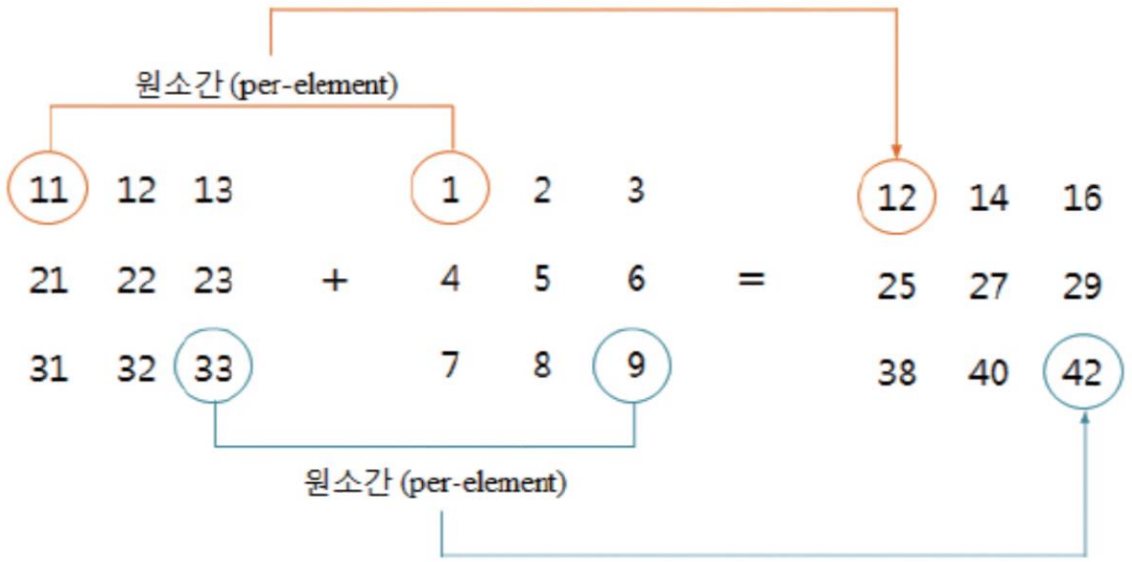
Run: 03.image_channels

```
C:\Python\python.exe D:/source/chap05/03.image_channels.py
bgr 자료형: <class 'list'> <class 'numpy.ndarray'> <class 'numpy.uint8'>
bgr 원소개수: 3
```

3. 산술 연산 함수

- ❖ 사칙 연산 함수
- ❖ 지수, 로그, 제곱근 관련 함수
- ❖ 논리(비트) 연산 함수

원소 간(per-element)
연산 수행 방식



〈그림 5.3.1〉 두 행렬의 원소 간 연산 과정

함수 설명		
cv2.add(src1, src2[, dst[, mask[, dtype]]]) → dst ■ 설명: 두 개의 배열 혹은 배열과 스칼라의 각 원소 간 합을 계산한다. 입력 인수 src1, src2 중 하나는 스칼라값 일 수 있다. ■ 수식 : $dst(i)=saturate(src1(i)+src2(i))$ if $mask(i)\neq 0$ $dst(i)=saturate(src1 \quad +src2(i))$ if $mask(i)\neq 0$ $dst(i)=saturate(src1(i)+src2 \quad)$ if $mask(i)\neq 0$		
인수 설명	■ src1	첫 번째 입력 배열 혹은 스칼라
	■ src2	두 번째 입력 배열 혹은 스칼라
	■ dst	계산된 결과의 출력 배열
	■ mask	연산 마스크: 0이 아닌 마스크 원소의 위치만 연산 수행(8비트 단일채널)
	■ dtype	출력 배열의 깊이
cv2.subtract(src1, src2[, dst[, mask[, dtype]]]) → dst ■ 설명: 두 개의 배열 혹은 배열과 스칼라의 각 원소 간 차분을 계산한다. add() 함수의 인수와 동일하다. ■ 수식 : $dst(i)=saturate(src1(i)-src2(i))$ if $mask(i)\neq 0$ $dst(i)=saturate(src1 \quad -src2(i))$ if $mask(i)\neq 0$ $dst(i)=saturate(src1(i)-src2 \quad)$ if $mask(i)\neq 0$		
cv2.multiply(src1, src2[, dst[, scale[, dtype]]]) → dst ■ 설명: 두 배열의 각 원소 간 곱을 계산한다. ■ 수식: $dst(i)=saturate(scale \cdot src1(i)) \cdot src2(i)$		
인수 설명	■ scale	두 배열의 원소 간 곱할 때 추가로 곱해주는 배율

cv2.divide(src1, src2[, dst[, scale[, dtype]]]) → dst

- 설명: 두 배열의 각 원소 간 나눗셈을 수행한다.
- 수식: $dst(i) = saturate(scale \cdot src1(i)/src2(i))$

cv2.divide(scale, src2[, dst[, dtype]]) → dst

- 설명: 스칼라값과 행렬원소간 나눗셈을 수행한다.
- 수식: $dst(i) = scale/src2(i)$

cv2.addWeighted(src1, alpha, src2, beta, gamma[, dst[, dtype]]) → dst

- 설명: 두 배열의 각 원소에 가중치를 곱한 후에 각 원소 간 합 즉, 가중된(weighted) 합을 계산한다.
- 수식: $dst(i) = saturate(src1(i) \cdot alpha + src2(i) \cdot beta + gamma$

인수 설명		
	■ alpha	첫 번째 배열의 모든 원소에 대한 가중치
	■ beta	두 번째 배열의 모든 원소에 대한 가중치
	■ gamma	두 배열의 원소 간 합에 추가로 더해주는 스칼라

예제 5.3.1 행렬 산술 연산 – 04.arithmethic_op.py

```python
import numpy as np, cv2

m1 = np.full((3, 6), 10, np.uint8)                    # 단일채널 생성 및 초
m2 = np.full((3, 6), 50, np.uint8)
m_mask = np.zeros(m1.shape, np.uint8)                 # 마스크 생성
m_mask[ :, 3: ] = 1                                   # 관심 영역을 지정한

m_add1 = cv2.add(m1, m2)                              # 행렬 덧셈
m_add2 = cv2.add(m1, m2, mask=m_mask)                 # 관심 영역만 덧셈 수

## 행렬 나눗셈 수행
m_div1 = cv2.divide(m1, m2)
m1 = m1.astype(np.float32)                            # 소수 부분 보존위해
m2 = np.float32(m2)                                   # 형변환 방법2
m_div2 = cv2.divide(m1, m2)

titles = ['m1', 'm2', 'm_mask', 'm_add1', 'm_add2', 'm_div1', 'm_div2']
for title in titles:
    print("[%s] = \n%s \n" % (title, eval(title)))
```

```
Run: 04.arithmethic_op
C:\Python\python.exe D:/source/chap05/04.arithmethic_op.py
[m1] =
[[10. 10. 10. 10. 10. 10.]
 [10. 10. 10. 10. 10. 10.]
 [10. 10. 10. 10. 10. 10.]]

[m2] =
[[50. 50. 50. 50. 50. 50.]
 [50. 50. 50. 50. 50. 50.]
 [50. 50. 50. 50. 50. 50.]]

[m_mask] =
[[0 0 0 1 1 1]
 [0 0 0 1 1 1]
 [0 0 0 1 1 1]]

[m_add1] =
[[60 60 60 60 60 60]
 [60 60 60 60 60 60]
 [60 60 60 60 60 60]]

[m_add2] =
[[ 0  0  0 60 60 60]
 [ 0  0  0 60 60 60]
 [ 0  0  0 60 60 60]]

[m_div1] =
[[0 0 0 0 0 0]
 [0 0 0 0 0 0]
 [0 0 0 0 0 0]]

[m_div2] =
[[0.2 0.2 0.2 0.2 0.2 0.2]
 [0.2 0.2 0.2 0.2 0.2 0.2]
 [0.2 0.2 0.2 0.2 0.2 0.2]]
```

관심 영역 [:, 3:]

mask 원소가 1인 위치만 연산 수행

나눗셈으로 인한 소수점 이하 값이 소실

형변환으로 소수점이하 값 유지

지수, 로그, 루트 관련 함수

함수 설명

cv2.exp(src[, dst]) → dst

- 설명: 모든 배열 원소의 지수(exponent)를 계산한다.
- 수식 : $dst(i)=e^{src(i)}$

인수 설명		
	■src, dst	입력 배열, 입력 배열과 같은 크기와 타입의 출력 배열

cv2.log(src[, dst]) → dst

- 설명: 모든 배열 원소의 절대값에 대한 자연 로그를 계산한다.
- 수식 : $dst(i)=\begin{cases}\log|src(i)| & \text{if } src(i)\neq 0\\ c & otherwise\end{cases}$

cv2.sqrt(src[, dst]) → dst

- 설명: 모든 배열 원소에 대해 제곱근을 계산한다.
- 수식 : $dst(i)=\sqrt{src(i)}$

cv2.pow(src, power[, dst]) → dst

- 설명: 모든 배열 원소에 대해서 제곱 승수를 계산한다.
- 수식 : $dst(i)=\begin{cases}src(i)^{power} & \text{if } power \text{ is integer}\\ |src(i)|^{power} & otherwise\end{cases}$

인수 설명		
	■power	제곱 승수

cv2.magnitude(x, y[, magnitude]) → magnitude

- 설명: 2차원 배열들의 크기(magnitude)를 계산한다.
- 수식 : $magnitude(i)=\sqrt{x(i)^2+y(i)^2}$

인수 설명		
	■x, y	x, y 좌표들의 입력 배열
	■magnitude	입력 배열과 같은 크기의 출력 배열

cv2.phase(x, y[, angle[, angleInDegrees]]) → angle

- 설명: 2차원 배열의 회전 각도를 계산한다.

수식 : $angle(i)=\mathrm{atan2}(y(i), x(i))\cdot[180/\pi]$

인수 설명		
	■angle	각도들의 출력 배열
	■angleInDegrees	True: 각을 도(degree)로 측정, False: 각을 라디안(radian)으로 측정

cv2.cartToPolar(x, y[, magnitude[, angle[, angleInDegrees]]]) → magnitude, angle

- 설명: 2차원 배열들의 크기(magnitude)와 각도를 계산한다.
- 수식 : $magnitude(i)=\sqrt{x(i)^2+y(i)^2}$

  $angle(i)=\tan^{-1}(y(i), x(i))\cdot[180/\pi]$

cv2.polarToCart(magnitude, angle[, x[, y[, angleInDegrees]]]) → x, y

- 설명: 각도와 크기(magnitude)로부터 2차원 배열들의 좌표를 계산한다.
- 수식 : $x(i)=magnitude(i)\cdot\cos(angel(i))$

  $y(i)=magnitude(i)\cdot\sin(angel(i))$

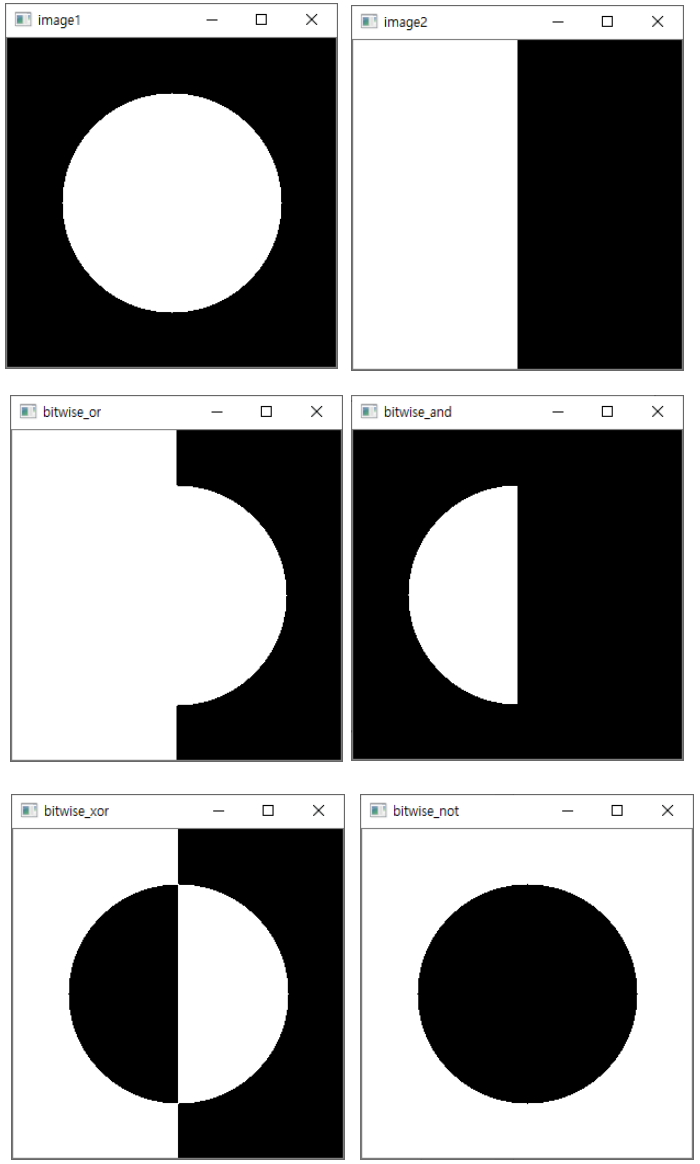
논리(비트) 연산 함수

함수 설명
cv2.bitwise_and(src1, src2[, dst[, mask]]) → dst ■설명: 두 배열의 원소 간 혹은 배열 원소와 스칼라 간의 비트별(bit-wise) 논리곱(AND) 연산을 수행한다. 입력 인수 src1, src2 중 하나는 스칼라값일 수 있다. ■수식: $dst(i)=src1(i)\wedge src2(i)$ if $mask(i)\neq 0$ $dst(i)=src1(i)\wedge src2$ if $mask(i)\neq 0$ $dst(i)=src1\wedge src2(i)$ if $mask(i)\neq 0$
cv2.bitwise_or(src1, src2[, dst[, mask]]) → dst ■설명: 두 개의 배열 원소 간 혹은 배열 원소와 스칼라 간의 비트별 논리합(OR) 연산을 수행한다. ■수식: $dst(i)=src1(i)\vee src2(i)$ if $mask(i)\neq 0$ $dst(i)=src1(i)\vee src2$ if $mask(i)\neq 0$ $dst(i)=src1\vee src2(i)$ if $mask(i)\neq 0$
cv2.bitwise_xor(src1, src2[, dst[, mask]]) → dst ■설명: 두 개의 배열 원소 간 혹은 배열 원소와 스칼라 간의 비트별 배타적 논리합(OR) 연산을 수행한다.
cv2.bitwise_not(src[, dst[, mask]]) → dst ■설명: 입력 배열의 모든 원소마다 비트 보수 연산을 한다. 쉽게 말하자면 반전시킨다. ■수식: $dst(i)=\sim src(i)$

인수 설명	■src1	첫 번째 입력 배열 혹은 스칼라값
	■src2	두 번째 입력 배열 혹은 스칼라값
	■dst	입력 배열과 같은 크기의 출력 배열
	■mask	마스크 연산 수행(8비트 단일채널 배열) – 마스크 배열의 원소가 0이 아닌 좌표만 계산을 수행

예제 5.3.4 행렬 비트 연산 – 07.bitwise_op.py

```
01  import numpy as np, cv2
02
03  image1 = np.zeros((300, 300), np.uint8)          # 300행, 300열 검은색(0) 영상 생성
04  image2 = image1.copy()                           # image1 복사
05
06  h, w = image1.shape[:2]
07  cx, cy = w//2, h//2                              # 중심 좌표
08  cv2.circle(image1, (cx, cy), 100, 255, -1)       # 중심에 원 그리기
09  cv2.rectangle(image2, (0, 0, cx, h), 255, -1)    # 영상의 가로 절반
10
11  image3 = cv2.bitwise_or(image1, image2)          # 원소 간 논리합
12  image4 = cv2.bitwise_and(image1, image2)         # 원소 간 논리곱
13  image5 = cv2.bitwise_xor(image1, image2)         # 원소 간 배타적 논리합
14  image6 = cv2.bitwise_not(image1)                 # 행렬 반전
15
16  cv2.imshow("image1", image1);        cv2.imshow("image2", image2)
17  cv2.imshow("bitwise_or", image3);    cv2.imshow("bitwise_and", image4)
18  cv2.imshow("bitwise_xor", image5);   cv2.imshow("bitwise_not", image6)
19  cv2.waitKey(0)
```



OpenCV

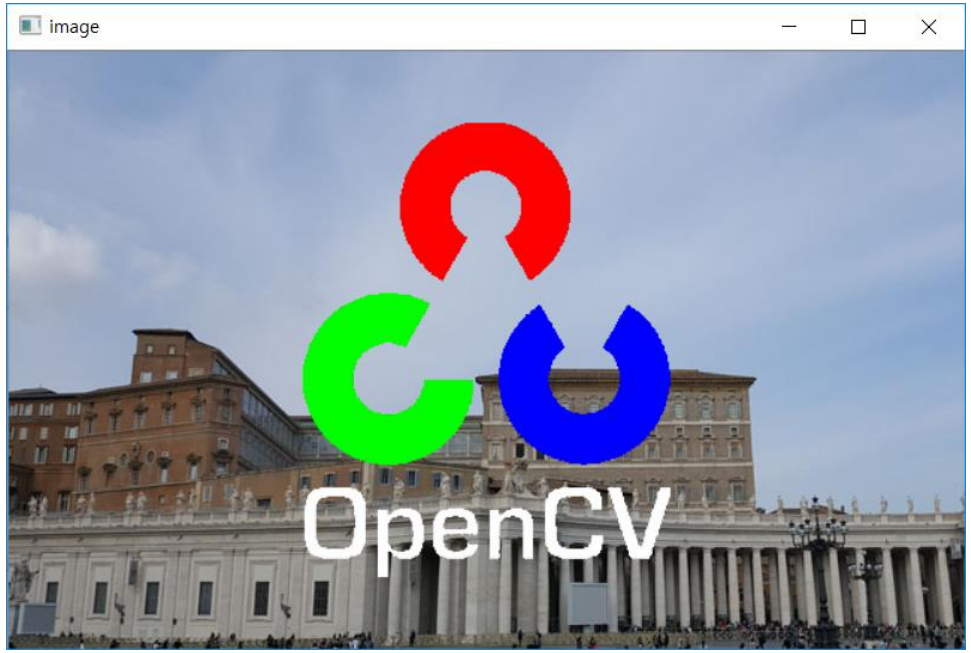
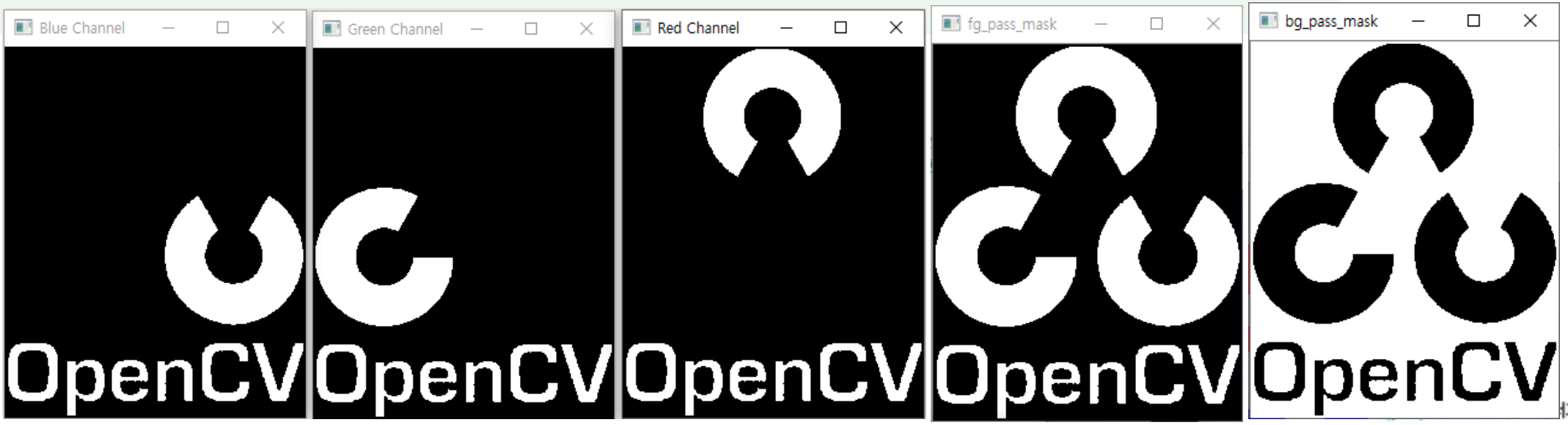


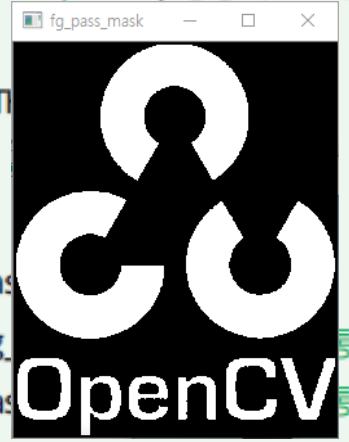
image
OpenCV

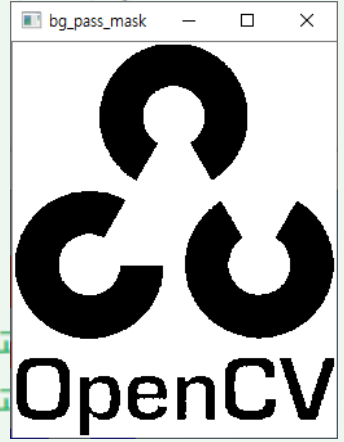
심화예제 5.3.4 행렬 비트 연산2 – 08.bitwise_overlap.py

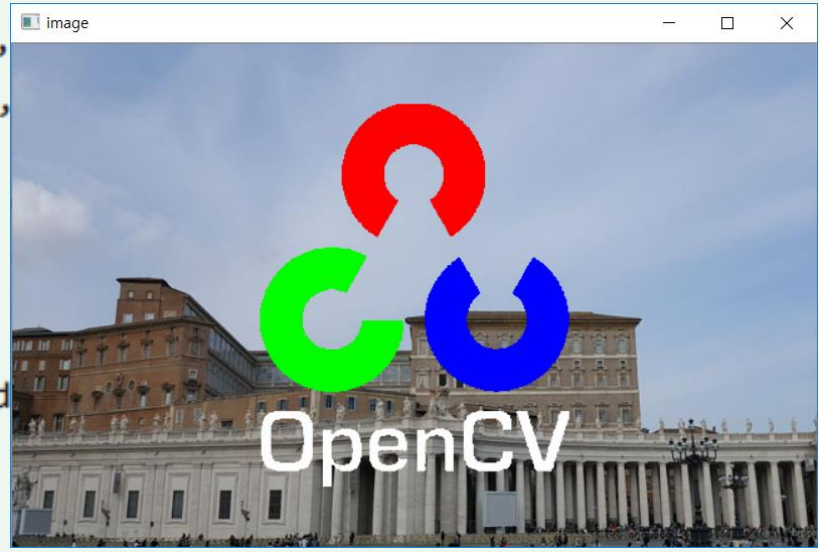
```
import numpy as np, cv2

image = cv2.imread("images/bit_test.jpg", cv2.IMREAD_COLOR)      # 원
logo  = cv2.imread("images/logo.jpg", cv2.IMREAD_COLOR)          # 로
if image is None or logo is None: raise Exception("영상파일 읽기 오

masks = cv2.threshold(logo, 220, 255, cv2.THRESH_BINARY)[1]      # 로
masks = cv2.split(masks)

fg_pass_mask = cv2.bitwise_or(masks[0], masks[1])
fg_pass_mask = cv2.bitwise_or(masks[2], fg_pass_mask)  # 전경 통과 마스크
bg_pass_mask = cv2.bitwise_not(fg_pass_mask)           # 배경 통과 마스크
```

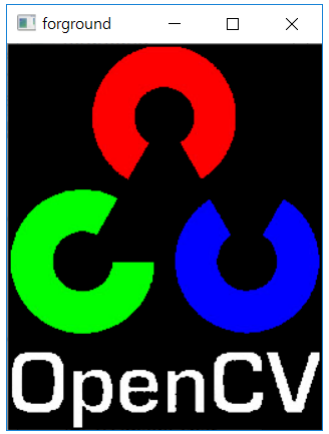


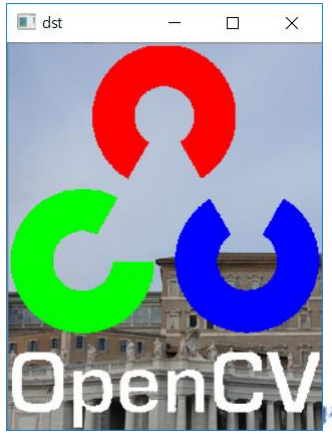
```
masks = cv2.threshold(logo, 220, 255, cv2.T
masks = cv2.split(masks)

fg_pass_mask = cv2.bitwise_or(masks[0], mas
fg_pass_mask = cv2.bitwise_or(masks[2], fg_
bg_pass_mask = cv2.bitwise_not(fg_pass_mas

(H, W), (h, w) = image.shape[:2], logo.shape[:2]        # 전체 영상, 로고 영상 크기
x, y = (W-w)//2, (H-h)//2                                # 시작 좌표 계산
roi = image[y:y+h, x:x+w]                                # 관심 영역(roi) 지정

## 행렬 논리곱과 마스킹을 이용한 관심 영역 복사
foreground = cv2.bitwise_and(logo, logo,
background = cv2.bitwise_and(roi , roi ,

dst = cv2.add(background, foreground)
image[y:y+h, x:x+w] = dst

cv2.imshow('background', background
cv2.imshow('dst', dst);
cv2.waitKey()
```











4. 절대값, 최대값, 최소값 관련 함수

❖원소의 절대값 관련 함수

❖원소의 최대값, 최소값 관련 함수

예제 5.4.1 행렬 절댓값 및 차분 연산 – 10.mat_abs.py

```python
import numpy as np, cv2

image1 = cv2.imread("images/abs_test1.jpg", cv2.IMREAD_GRAYSCALE) # 명암도 영상 읽기
image2 = cv2.imread("images/abs_test2.jpg", cv2.IMREAD_GRAYSCALE)
if image1 is None or image2 is None: raise Exception("영상파일 읽기 오류")

dif_img1 = cv2.subtract(image1, image2)                    # 차분 연산
dif_img2 = cv2.subtract(np.int16(image1), np.int16(image2))  # 음수 결과 보존
abs_dif1 = np.absolute(dif_img2).astype('uint8')
abs_dif2 = cv2.absdiff(image1, image2)                     # 차분 절댓값 계산

x, y, w, h = 100, 150, 7, 3                                # 관심 영역
print("[dif_img1(roi) uint8] = \n%s\n" % dif_img1[y:y+h, x:x+w]) # 관심 영역 원소값 출력
print("[dif_img2(roi) int16]  = \n%s\n"  % dif_img2[y:y+h, x:x+w])
print("[abs_dif1(roi)] = \n%s\n" % abs_dif1[y:y+h, x:x+w])
print("[abs_dif2(roi)] = \n%s\n" % abs_dif2[y:y+h, x:x+w])

titles = ['image1', 'image2', 'dif_img1', 'abs_dif1', 'abs_dif2']  # 윈도우 제목 리스트
for title in titles:
    cv2.imshow(title, eval(title))                         #
cv2.waitKey(0)
```

Run: 09.mat_abs

```
C:\Python\python.exe D:/source/chap05/09.mat_abs.py
[dif_img1(roi) uint8] =
[[ 0  0  0  0  9 12  7]
 [ 0  0  0  0  4  9  3]
 [ 0  0  0 15  0  4  0]]

[dif_img2(roi) int16]  =
[[-100 -106  -80   -6    9   12    7]
 [-105 -109  -72   -4    4    9    3]
 [-106 -109  -58   15   -1    4    0]]

[abs_dif1(roi)] =
[[100 106  80   6   9  12   7]
 [105 109  72   4   4   9   3]
 [106 109  58  15   1   4   0]]

[abs_dif2(roi)] =
[[100 106  80   6   9  12   7]
 [105 109  72   4   4   9   3]
 [106 109  58  15   1   4   0]]
```

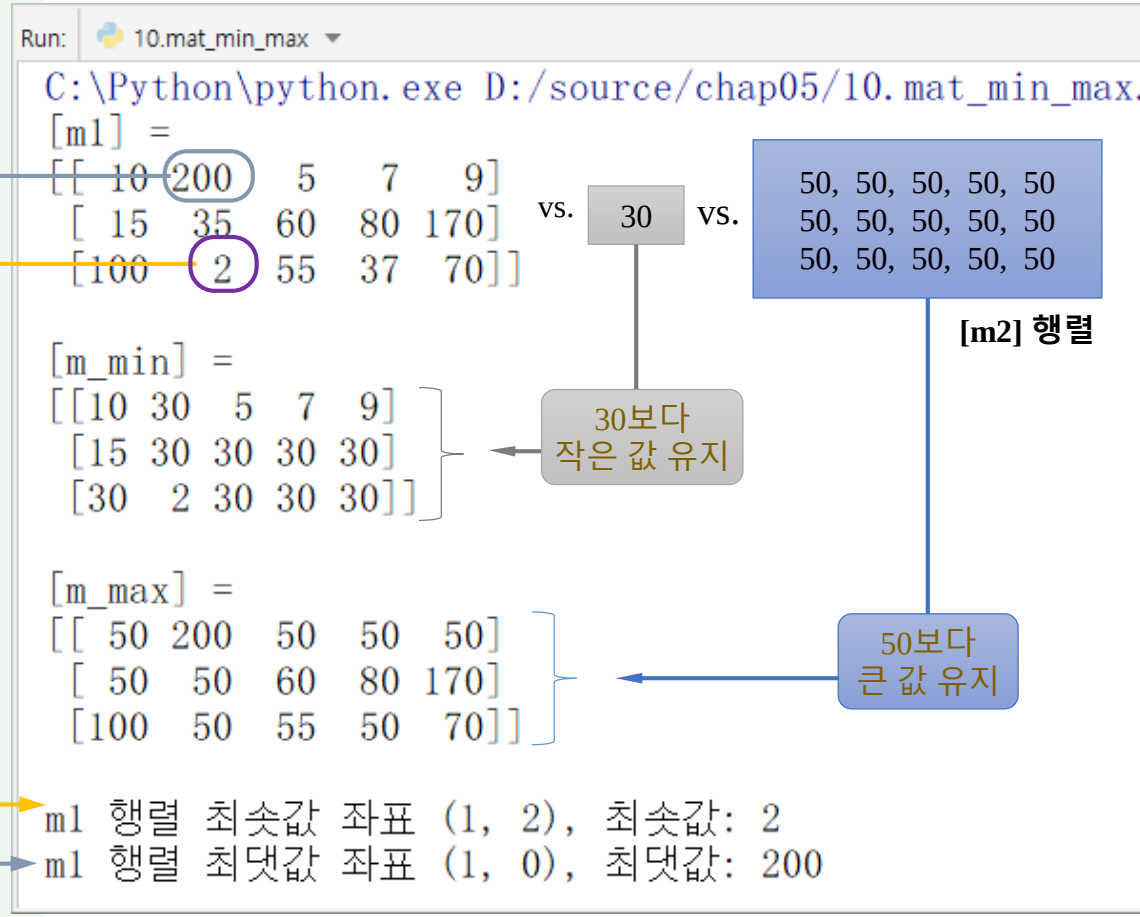
차분 영상의 관심영역 일부

int16형으로 변환 후 계산

차분 영상의 절대값 계산 결과

예제 5.4.2 행렬 최소값 및 최대값 연산 – 10.mat_min_max.py

```python
import numpy as np, cv2

data = [  10, 200, 5, 7, 9,                    # 1차원 리스트 생성
          15, 35, 60, 80, 170,
          100, 2, 55, 37, 70 ]
m1 = np.reshape(data, (3, 5))                  # 리스트 행태 변환하여 2차원 행렬 생성
m2 = np.full((3, 5), 50)                       # 원소값 50인 2차원 행렬 생성

m_min = cv2.min(m1, 30)                        # 행렬 원소와 스칼라 간 최솟값을 행렬로 저장
m_max = cv2.max(m1, m2)                        # 두 행렬 원소간 최댓값 계산

## 행렬의 최솟값/최댓값과 그 좌표들을 반환
min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(m1)

print("[m1] = \n%s\n" % m1)
print("[m_min] = \n%s\n" % m_min)
print("[m_max] = \n%s\n" % m_max)

## min_loc와max_loc 좌표는(y, x)이므로 행렬의 좌표 위치와 반대임
print("m1 행렬 최솟값 좌표%s, 최솟값: %d" %(min_loc, min_val) )
print("m1 행렬 최댓값 좌표%s, 최댓값: %d" %(max_loc, max_val) )
```

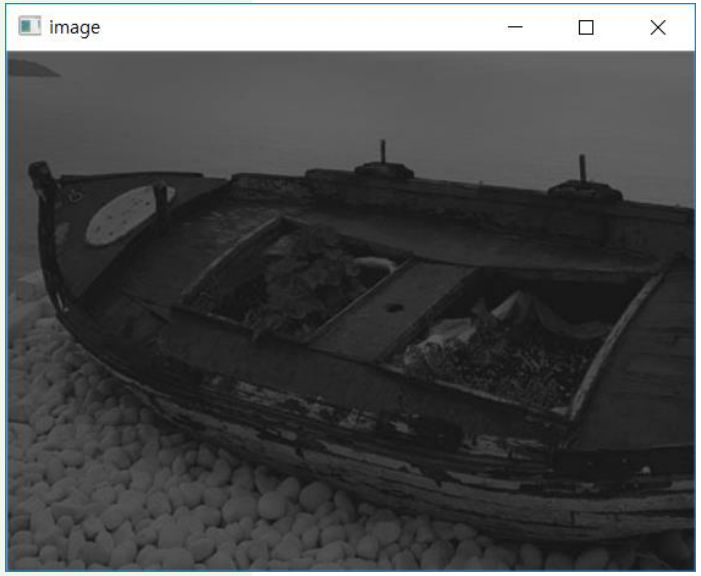


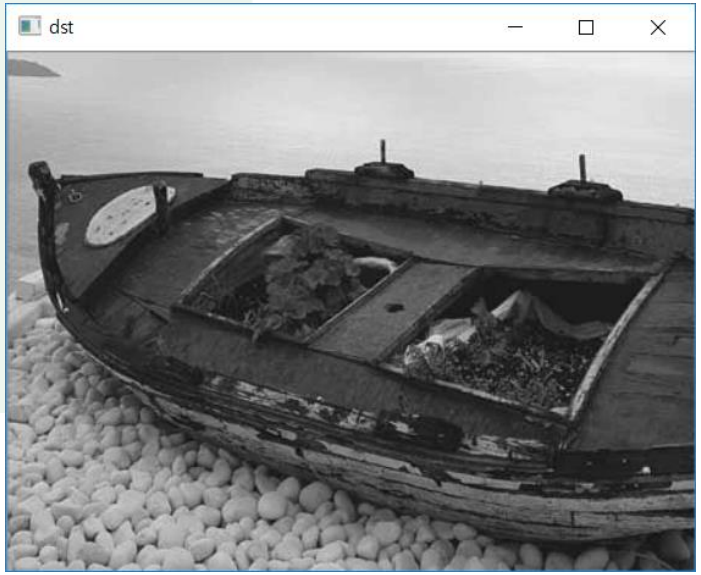
심화예제 5.4.3 영상 최소값 최대값 연산 – 11.image_min_max.py

```python
import numpy as np, cv2

image = cv2.imread("images/minMax.jpg", cv2.IMREAD_GRAYSCALE)
if image is None: raise Exception("영상파일 읽기 오류 발생")

min_val, max_val, _, _ = cv2.minMaxLoc(image)  # 최솟값과 최댓값 가져오기

ratio = 255 / (max_val - min_val)
dst = np.round((image - min_val) * ratio).astype('uint8')
min_dst, max_dst, _, _ = cv2.minMaxLoc(dst)

print("원본 영상 최솟값= %d, 최댓값= %d" % (min_val, max_val))
print("수정 영상 최솟값= %d, 최댓값= %d" % (min_dst, max_dst))
cv2.imshow('image', image)
cv2.imshow('dst', dst)
cv2.waitKey(0)
```

Run: 11.image_min_max

```
C:\Python\python.exe D:/source/chap05/11.image_min_max.py
원본 영상 최솟값= 13 , 최댓값= 107
수정 영상 최솟값= 0 , 최댓값= 255
```





5. 통계 관련 함수

함수 설명	
cv2.sumElems(src) → retval ■ 설명: 배열의 각 채널별로 원소들의 합 N을 계산하여 스칼라값으로 반환한다. ■ 수식: $S=\sum_{i} src(i)$	
인수 설명	■ src 1개에서 4개 채널을 갖는 입력 배열
cv2.mean(src[, mask]) → retval ■ 설명: 배열의 각 채널별로 원소들의 평균을 계산하여 스칼라값으로 반환한다. ■ 수식: $N=\sum_{i:mask(i)\neq 0} 1$ $M_c=\left(\sum_{i:mask(i)\neq 0} src(i)\right)/N$	
인수 설명	■ src 1개에서 4개 채널을 갖는 입력 배열 ■ mask 연산 마스크 – 마스크가 0이 아닌 좌표만 연산 수행
cv2.meanStdDev(src[, mean[, stddev[, mask]]]) → mean, stddev ■ 설명: 배열 원소들의 평균과 표준편차를 계산한다.	

인수 설명	■ src	1개에서 4개 채널을 갖는 입력 배열
	■ mean	계산된 평균이 반환되는 출력 인수, np.float64형으로 반환
	■ stddev	계산된 표준편차가 반환되는 출력 인수, np.float64형으로 반환
	■ mask	연산 마스크 – 마스크가 0이 아닌 좌표만 연산 수행

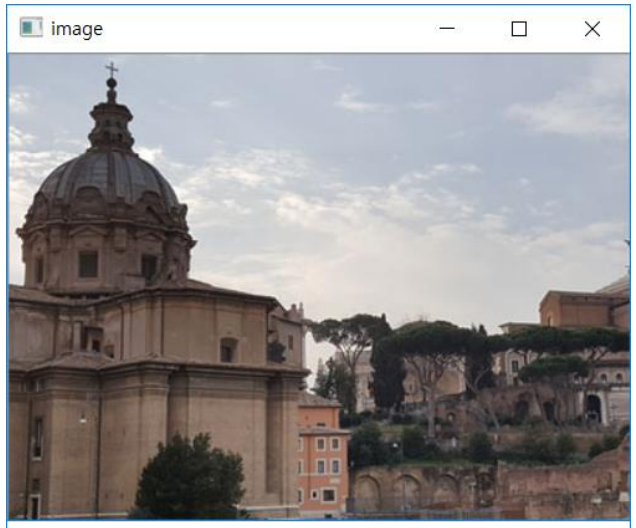
cv2.countNonZero(src) → retval

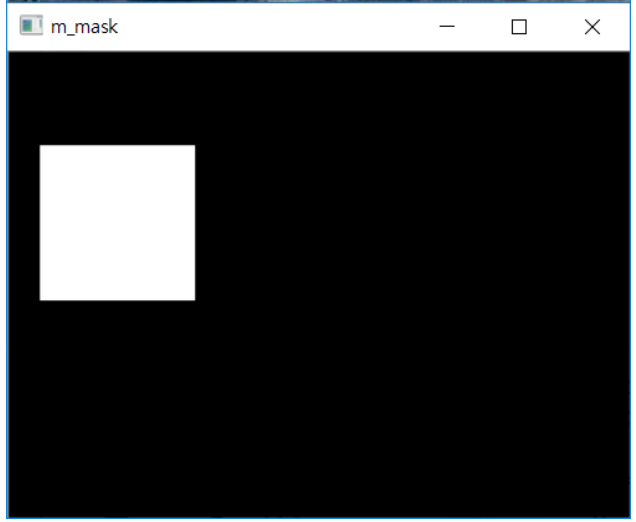
■ 설명: 0이 아닌 배열 원소를 개수N을 반환한다.

■ 수식: $N = \sum_{I:src(I) \neq 0} 1$

예제 5.5.1 행렬 합/평균 연산 – 12.sum_avg.py

```python
import numpy as np, cv2

image = cv2.imread("images/sum_test.jpg", cv2.IMREAD_COLOR)
if image is None: raise Exception("영상파일 읽기 오류 발생")

mask = np.zeros(image.shape[:2], np.uint8)
mask[60:160, 20:120] = 255                          # 관심 영역에 값(255) 할당

sum_value    = cv2.sumElems(image)                  # 채널별 합 – 튜플로 반환
mean_value1 = cv2.mean(image)                       # 채널별 평균– 튜플로 반환
mean_value2 = cv2.mean(image, mask)

print("sum_value 자료형:", type(sum_value), type(sum_value[0]))    # 결과 행렬의 자료형
print("[sum_value] =", sum_value)
print("[mean_value1] =", mean_value1)
print("[mean_value2] =", mean_value2)
```






```
Run: 12.sum_avg
C:\Python\python.exe D:/source/chap05/12.sum_avg.p
sum_value 자료형: <class 'tuple'> <class 'float'>
[sum_value] =  (15865577.0, 15880547.0, 16470875.0, 0.0)
[mean_value1] =  (132.21314166666667, 132.33789166666668, 137.25729166666667, 0.0)
[mean_value2] =  (80.26520000000001, 81.59740000000001, 90.3211, 0.0)
```

4 원소 튜블 반환 , 마지막 원소 0

전체 영역 평균

관심 영역 평균

```python
print()

## 평균과 표준편차 결과 저장
mean, stddev = cv2.meanStdDev(image)                    # 2 원소 튜플로 반환
mean2, stddev2 = cv2.meanStdDev(image, mask=mask)       # 마스크가 255인 영역만 계산
print("mean 자료형:", type(mean), type(mean[0][0]))      # 반환 행렬 자료형, 원소 자료형
print("[mean] =",  mean.flatten())                      # 벡터 변환 후 출력
print("[stddev] =", stddev.flatten())
print()

print("[mean2] =",  mean2.flatten())
print("[stddev2] =", stddev2.flatten())

cv2.imshow('image', image)
cv2.imshow('mask', mask)
cv2.waitKey(0)
```

```
mean 자료형: <class 'numpy.ndarray'> <class 'numpy.float64'>
[mean] =  [132.21314167 132.33789167 137.25729167]
[stddev] =  [73.35044328 68.76754506 63.96477788]

[mean2] =  [80.2652 81.5974 90.3211]
[stddev2] =  [58.91488326 57.57273064 54.0648388 ]
```

전체영역 평균, 표준편차

관심영역 평균, 표준편차

cv2.reduce(src, dim, rtype[, dst[, dtype]]) → dst

■ 설명: 행렬을 열방향/행방향으로 옵션 상수(rtype)에 따라 축소한다.

인수 설명

■ src 2차원 입력 배열 (np.float32, np.float64형만 수행 가능)

■ dst 출력 벡터, 감소방향과 타입은 dim, dtype 인수에 따라 정해짐

■ dim 행렬이 축소될 때 차원 감소 첨자

– 0 : 열 방향으로 연산하여 1행으로 축소

– 1 : 행 방향으로 연산하여 1열로 감소

■ rtype 축소 연산 종류

옵션 상수	값	설명
cv2.REDUCE_SUM	0	행렬의 모든 행(열)들을 합한다.
cv2.REDUCE_AVG	1	행렬의 모든 행(열)들을 평균한다.
cv2.REDUCE_MAX	3	행렬의 모든 행(열)들의 최댓값을 구한다.
cv2.REDUCE_MIN	4	행렬의 모든 행(열)들의 최솟값을 구한다.

■ dtype 감소된 벡터의 자료형

dim=0 : 한 행으로 감축

11	2	3	4	10
6	10	15	9	7
7	12	8	14	1

24	24	26	27	18

rtype = cv2.REDUCE_SUM

dim=1 : 한 열로 감축

11	2	3	4	10
6	10	15	9	7
7	12	8	14	1

6
9.4
8.4

rtype = cv2.REDUCE_AVG

예제 5.5.5 행렬 축소 연산 – 17.mat_reduce.py

```
01  import numpy as np, cv2
02
03  m = np.random.rand(3, 5) * 1000//10                    # 0~99 사이 실수(float)생성
04
05  reduce_sum     = cv2.reduce(m, dim=0, rtype=cv2.REDUCE_SUM)     # 0 – 열방향 축소
06  reduce_avg     = cv2.reduce(m, dim=1, rtype=cv2.REDUCE_AVG)     # 1 – 행방향 축소
07  reduce_max     = cv2.reduce(m, dim=0, rtype=cv2.REDUCE_MAX)
08  reduce_min     = cv2.reduce(m, dim=1, rtype=cv2.REDUCE_MIN)
09
10  print("[m1] = \n%s\n" % m1)
11  print("[m_reduce_sum] =", reduce_sum.flatten())
12  print("[m_reduce_avg] =", reduce_avg.flatten())
13  print("[m_reduce_max] =", reduce_max.flatten())
14  print("[m_reduce_min] =", reduce_min.flatten())
```

Run: 16.mat_reduce

```
C:\Python\python.exe D:/source/chap05/16.mat_reduce.py
[m1] =
[[41.  16.  59.  33.  38.]
 [34.  59.  23.  32.  64.]
 [14.  79.  86.  76.  84.]]

[m_reduce_sum] = [ 89.  154.  168.  141.  186.]
[m_reduce_avg] = [37.4 42.4 67.8]
[m_reduce_max] = [41.  79.  86.  76.  84.]
[m_reduce_min] = [16.  23.  14.]
```

cv2.sort(src, flags[, dst]) → dst

■ 설명: 행렬의 각 행 혹은 각 열의 방향으로 정렬한다.

인수 설명

■ src 단일채널 입력 배열

■ dst 정렬된 출력 배열

■ flags 연산 플래그 – 다음의 상수를 조합해서 정렬 방식 구성

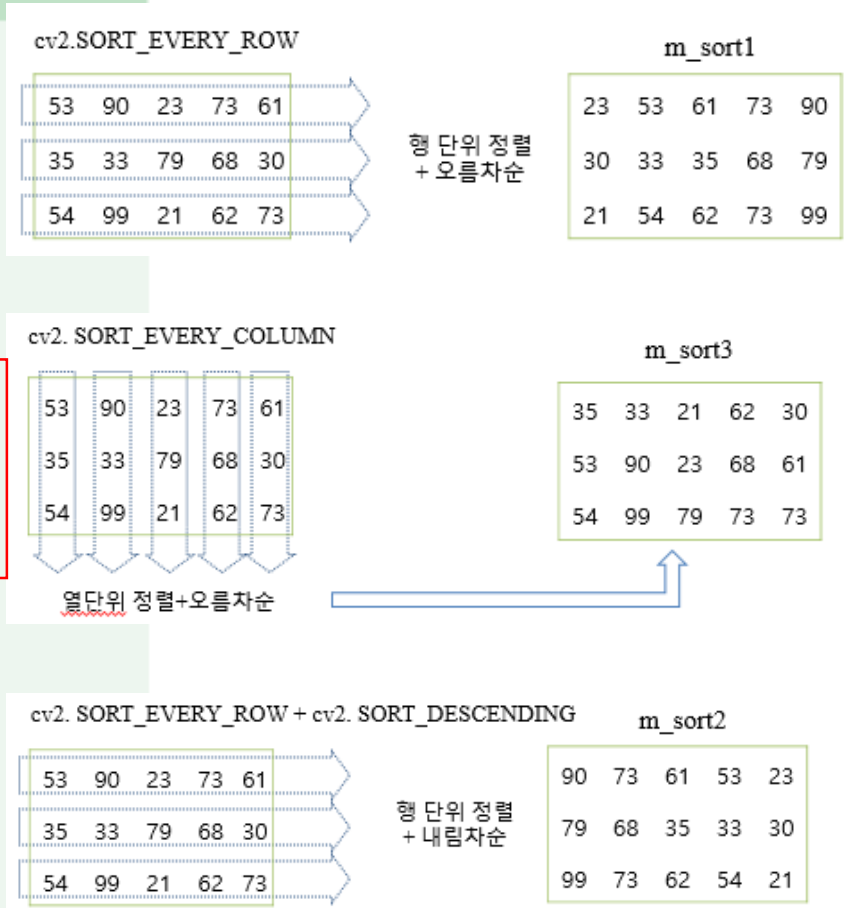
옵션 상수	값	설명
cv2.SORT_EVERY_ROW	0	각 행을 독립적으로 정렬
cv2.SORT_EVERY_COLUMN	1	각 열을 독립적으로 정렬
cv2.SORT_ASCENDING	0	오름차순으로 정렬
cv2.SORT_DESCENDING	16	내림차순으로 정렬

cv2.sortIdx(src, flags[, dst]) → dst

■ 설명: 행렬의 각 행 혹은 각 열로 정렬한다. 출력 배열(dst)에 정렬된 원소의 첨자들을 저장한다. 인수는 cv2.sort()와 동일하다.

예제 5.5.2 행렬 원소 정렬 – 13.sort.py

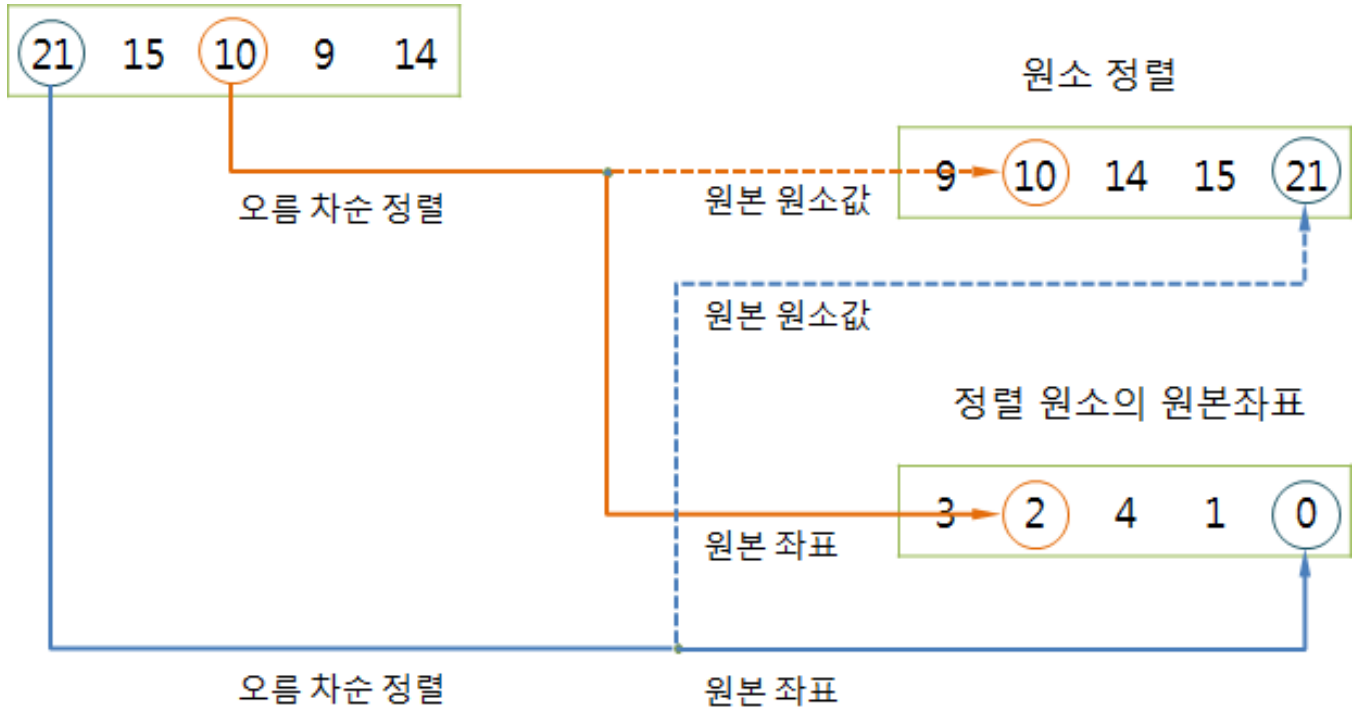
```
import numpy as np, cv2

m = np.random.randint(0, 100, 15).reshape(3,5)          # 임의 난수 생성
## 행렬 원소 정렬
sort1 = cv2.sort(m, cv2.SORT_EVERY_ROW)                 # 행단위(가로 방향) 오름차순
sort2 = cv2.sort(m, cv2.SORT_EVERY_COLUMN)              # 열단위(세로 방향) 오름차순
sort3 = cv2.sort(m, cv2.SORT_EVERY_ROW+cv2.SORT_DESCENDING) # 행단위 내림차순
sort4 = np.sort(m, axis=1)                              # x축 (가로 방향) 정렬
sort5 = np.sort(m, axis=0)                              # y축 (세로 방향) 정렬
sort6 = np.sort(m, axis=1)[:, ::-1]                     # 열 방향 내림차순 정렬

titles= ['m', 'sort1', 'sort2', 'sort3', 'sort4', 'sort5', 'sort6']
for title in titles:
    print("[%s] = \n%s\n" %(title, eval(title)))
```



통계 관련 함수

cv2.sortIdx() 함수 사용 예

- 행렬을 정렬한 후 원래 데이터의 위치를 반환



예제 5.5.3 정렬 인덱스 반환 – 14.sortidx.py

```
01  import numpy as np, cv2
02
03  m = np.random.randint(0, 100, 15).reshape(3,5)          # 임의 난수 생성
04
05  m_sort1 = cv2.sortIdx(m, cv2.SORT_EVERY_ROW)            # 원본 좌표(정렬 인덱스)
06  m_sort2 = cv2.sortIdx(m, cv2.SORT_EVERY_COLUMN)         # 열 방향 정렬
07  m_sort3 = np.argsort(m, axis=0)                         # y축 방향 정렬
08
09  print("[m1] = \n%s\n" % m)
10  print("[m_sort1] = \n%s\n" % m_sort1)
11  print("[m_sort2] = \n%s\n" % m_sort2)
12  print("[m_sort3] = \n%s\n" % m_sort3)
```

```
Run: 14.sortidx
C:\Python\python.exe I
[m1] =
[[20 14 32 91 55]
 [ 2 13 31 55 95]
 [81  6 50 11 13]]

[m_sort1] =
[[1 0 2 4 3]
 [0 1 2 3 4]
 [1 3 4 2 0]]

[m_sort2] =
[[1 2 1 2 2]
 [0 1 0 1 0]
 [2 0 2 0 1]]

[m_sort3] =
[[1 2 1 2 2]
 [0 1 0 1 0]
 [2 0 2 0 1]]
```

6. 행렬 연산 함수

General Matrix Multiplication

함수 설명
cv2.gemm(src1, src2, alpha, src3, beta[, dst[, flags]]) → dst ■ 설명: 일반화된 행렬 곱셈을 수행한다. ■ 수식: $dst = alpha \cdot src1^T \cdot src2 + beta \cdot src3^T$

인수 설명

- ■ src1, src2 — 행렬 곱을 위한 두 입력 행렬(np.float32/np.float64형 2채널까지 가능)
- ■ alpha — 행렬 곱($src1^T \cdot src2$)에 대한 가중치
- ■ src3 — 행렬 곱($src1^T \cdot src2$)에 더해지는 델타 행렬
- ■ beta — src3 행렬에 곱해지는 가중치
- ■ dst — 출력 행렬
- ■ flags — 연산 플래그 – 옵션을 조합하여 입력 행렬들을 전치 (default = 0)

옵션	값	설명
cv2.GEMM_1_T	1	src1을 전치
cv2.GEMM_2_T	2	src2을 전치
cv2.GEMM_3_T	4	src3을 전치

예제 5.6.1 행렬곱(내적) 연산 – 18.gemm.py

```
import numpy as np, cv2

src1 = np.array([1, 2, 3, 1, 2, 3], np.float32).reshape(2, 3)      # 2×3 행렬 선언
src2 = np.array([1, 2, 3, 4, 5, 6], np.float32).reshape(2, 3)
src3 = np.array([1, 2, 1, 2, 1, 2], np.float32).reshape(3, 2)      # 3×2 행렬 선언
alpha, beta = 1.0, 1.0

dst1 = cv2.gemm(src1, src2, alpha, None, beta, flags=cv2.GEMM_1_T)
dst2 = cv2.gemm(src1, src2, alpha, None, beta, flags=cv2.GEMM_2_T)
dst3 = cv2.gemm(src1, src3, alpha, None, beta)

titles = ['src1', 'src1', 'src1', 'dst1', 'dst2', 'dst3']
for title in titles:
    print("[%s] = \n%s\n" % (title, eval(title)))
```

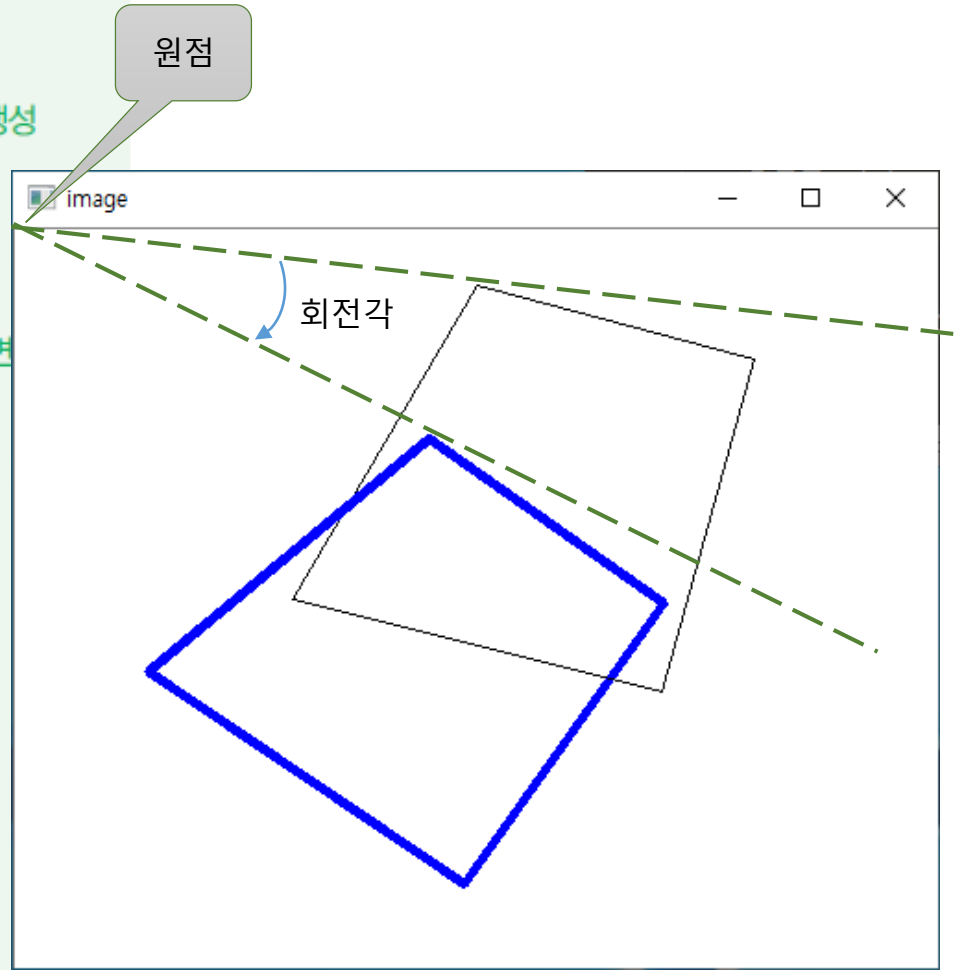
$$src1^T \cdot src2 = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix}^T \cdot \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 3 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} = \begin{bmatrix} 5 & 7 & 9 \\ 10 & 14 & 18 \\ 15 & 21 & 27 \end{bmatrix}$$

$$src1 \cdot src2^T = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}^T = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix} \cdot \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix} = \begin{bmatrix} 14 & 32 \\ 14 & 32 \end{bmatrix}$$

$$src1 \cdot src3 = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 \\ 1 & 2 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} 6 & 12 \\ 6 & 12 \end{bmatrix}$$

심화예제 5.6.2 cv2.gemm()을 이용한 회전 변환 – 19.point_transform.py

```
import numpy as np, cv2

theta = 20 * np.pi / 180                                  # 회전할 라디안 각도 계산
rot_mat = np.array([ [np.cos(theta), -np.sin(theta)],
                     [np.sin(theta),  np.cos(theta)]], np.float32) # 회전 변환 행렬 생성

pts1 = np.array([ (250, 30), (400, 70),
                  (350, 250), (150, 200)], np.float32)            # 입력 좌표 지정
pts2 = cv2.gemm(pts1, rot_mat, 1, None, 1, flags=cv2.GEMM_2_T )# 행렬곱으로 회전변

for i, (pt1, pt2) in enumerate(zip(pts1, pts2)):
    print("pts1[%d] = %s, pst2[%d]= %s" %(i, pt1, i, pt2))

## 영상 생성 및 4개 좌표 그리기
image = np.full((400, 500, 3), 255, np.uint8)
cv2.polylines(image, [np.int32(pts1)], True, (0, 255, 0), 2)
cv2.polylines(image, [np.int32(pts2)], True, (255, 0, 0), 3)
cv2.imshow('image', image)
cv2.waitKey(0)
```



6. 행렬 연산 함수

함수 설명		
cv2.invert(src[, dst[, flags]]) → retval, dst ■ 설명: 행렬의 역행렬을 계산한다.(입력 행렬이 정방 행렬이 아니면 의사 역행렬 계산)		
인수 설명	■ src	$M \times N$크기의 부동소수점 입력 행렬
	■ dst	src와 크기와 타입이 같은 출력 행렬
	■ flags	역행렬의 계산 방법에 대한 플래그
cv2.solve(src1, src2[, dst[, flags]]) → retval, dst ■ 설명: 연립 방정식이나 최소자승 문제를 해결한다. ■ 수식: $dst = \text{argmin}\|src1 \cdot X - src1\|$		
인수 설명	■ src1	연립방정식의 계수 행렬
	■ src2	연립방정식의 상수 행렬
	■ dst	출력 행렬
	■ flags	풀이(역행렬 계산 플래그) 방법

〈표 5.6.1〉 flags 옵션의 역행렬의 계산 방법

옵션	값	설명
cv2.DECOMP_LU	0	가우시안 소거법으로 역행렬 계산 - 입력 행렬은 역행렬이 존재하는 정방행렬
cv2.DECOMP_SVD	1	특이치 분해 방법으로 역행렬 계산 - 입력 행렬이 정방행렬이 아닌 경우 의사 역행렬 계산
cv2.DECOMP_CHOLESKY	3	촐레스키(cholesky) 분해로 역행렬 계산 - 입력 행렬이 역행렬이 존재하는 정방행렬, 대칭행렬이며 양의 정부호 행렬

예제 5.6.3 cv2.invert()와 cv2.solve()로 연립방정식 풀이 – 20.equation.py

```
01  import numpy as np, cv2
02
03  data = [ 3, 0, 6, -3, 4, 2, -5,-1, 9]                # 연립방정식의 계수들
04  m1 = np.array(data, np.float32).reshape(3,3)          # 계수들을 행렬로 생성
05  m2 = np.array([36, 10, 28], np.float32)               # 상수 벡터
06
07  ret, inv = cv2.invert(m1, cv2.DECOMP_LU)              # 역행렬 계산
08  if ret:
09      dst1 = inv.dot(m2)                                # numpy 제공 행렬곱 함수
10      dst2 = cv2.gemm(inv, m2, 1, None, 1)              # OpenCV 제공 행렬곱 함수
11      _ , dst3 = cv2.solve(m1, m2, cv2.DECOMP_LU)       # 연립방정식 풀이
12
13      print("[inv] = \n%s\n" % inv)
14      print("[dst1] =", dst1.flatten())                 # 행렬을 벡터로 변환
15      print("[dst2] =", dst2.flatten())
16      print("[dst3] =", dst3.flatten())
17  else:
18      print("역행렬이 존재하지 않습니다.")
```

연립방정식:

$$\begin{aligned} 3x_1 + + 6x_3 &= 36 \\ -3x_1 + 4x_2 + 2x_3 &= 10 \\ -5x_1 - 1x_2 + 9x_3 &= 28 \end{aligned}$$

$$m_1 \times x = m_2$$

$$m_1^{-1} \times m_1 \times x = m_1^{-1} \times m_2$$

$$x = m_1^{-1} \times m_2$$

Run: 19.equation

```
C:\Python\python.exe D:/source/chap05/19.equation.py
[inv] =
[[ 0.15079366 -0.02380952 -0.0952381 ]
 [ 0.06746032  0.22619048 -0.0952381 ]
 [ 0.09126984  0.01190476  0.04761905]]

[dst1] = [2.5238097 2.0238097 4.7380953]
[dst2] = [2.5238097 2.0238097 4.7380953]
[dst3] = [2.5238094 2.0238094 4.7380953]
```

Summary

- ❖기본 배열 처리 함수
  - Flip(), repeat(), transpose()
- ❖ 채널 합성 및 분리 함수
  - merge(), split()
  - 컬러 영상(3개 채널), 명암도 영상(단일 채널)
- ❖두 배열(행렬 혹은 벡터) 간의 사칙연산 함수
  - add(), subtract(), multiply(), divide()
  - 이 함수들은 두 행렬의 원소 간(per-element)에 연산을 수행
  - mask 행렬을 이용해서 특정 영역만 연산이 가능
- ❖행렬에 대한 비트 연산 함수를 제공
  - bitwise_and(), bitwise_or(), bitwise_not(), bitwise_xor()

Summary

- ❖행렬에서 최소값과 최대값을 구하는 함수
  - 두 행렬의 원소 간에 작은 값들과 큰 값: min(), max()
  - 하나의 행렬에서 최소값, 최대값과 원소의 위치를 가져옴: minMaxIdx(), minMaxLoc()
- ❖하나의 행렬에서 통계적인 요소를 계산해주는 함수공
  - 합계: sum(), 평균: mean(), 평균, 표준편차: meanStdDev()
- ❖행렬의 원소를 정렬하는 함수를 제공
  - 원소의 정렬: sort(),
  - 정렬된 원소의 원 위치를 반환: sortIdx()
- ❖ 행렬 연산
  - 행렬의 곱: gemm(), 역행렬: invert(), 연립방정식의 해: solve()