CHAPTER 05

contents

- 5.1 기본 배열(Array) 처리 함수
- 5.2 채널 처리 함수
- 5.3 산술 연산 함수
- 5.4 원소의 절댓값 연산
- 5.5 통계 관련 함수

5.1 기본 배열(Array) 처리 함수

- 파이썬에서는 배열을 처리하기 위한 자료형
  - 열거형(sequence) 객체 - 리스트, 튜플, 사전(dictionary)
- 명칭 표현
  - 1차원 데이터 - 벡터
  - 2차원 데이터 – 행렬
  - 1차원과 2차원 데이터 통칭해서 배열

5.1 기본 배열(Array) 처리 함수

- 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

5.1 기본 배열(Array) 처리 함수 (실습)

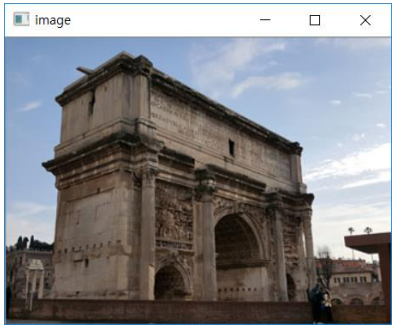
예제 5.1.1 행렬 처리 함수 – 01.mat_array.py

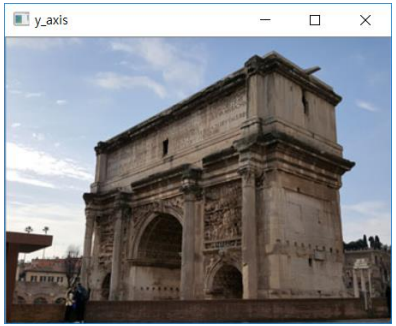
```
import cv2

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)
```

문자열로 변수 명령어 만듦
– 행렬 변수로 적용

5.1 기본 배열(Array) 처리 함수

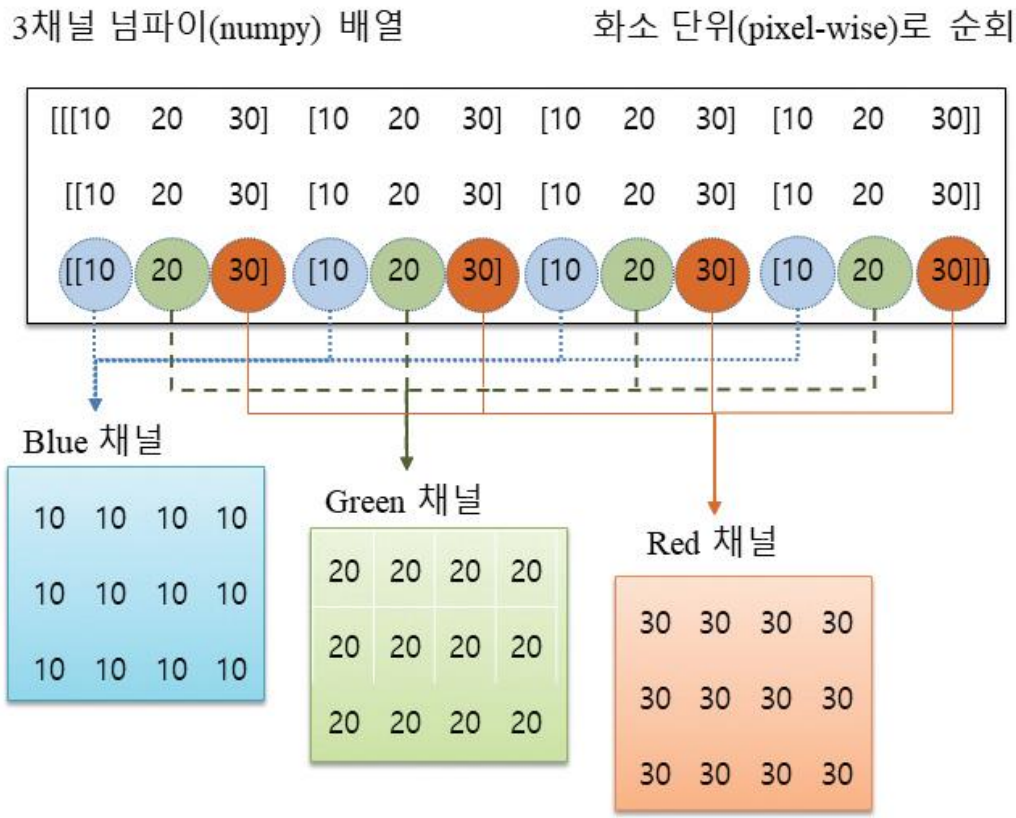
- 실행결과





5.2 채널 처리 함수

- 채널 개념



5.2 채널 처리 함수

- 채널 관련 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

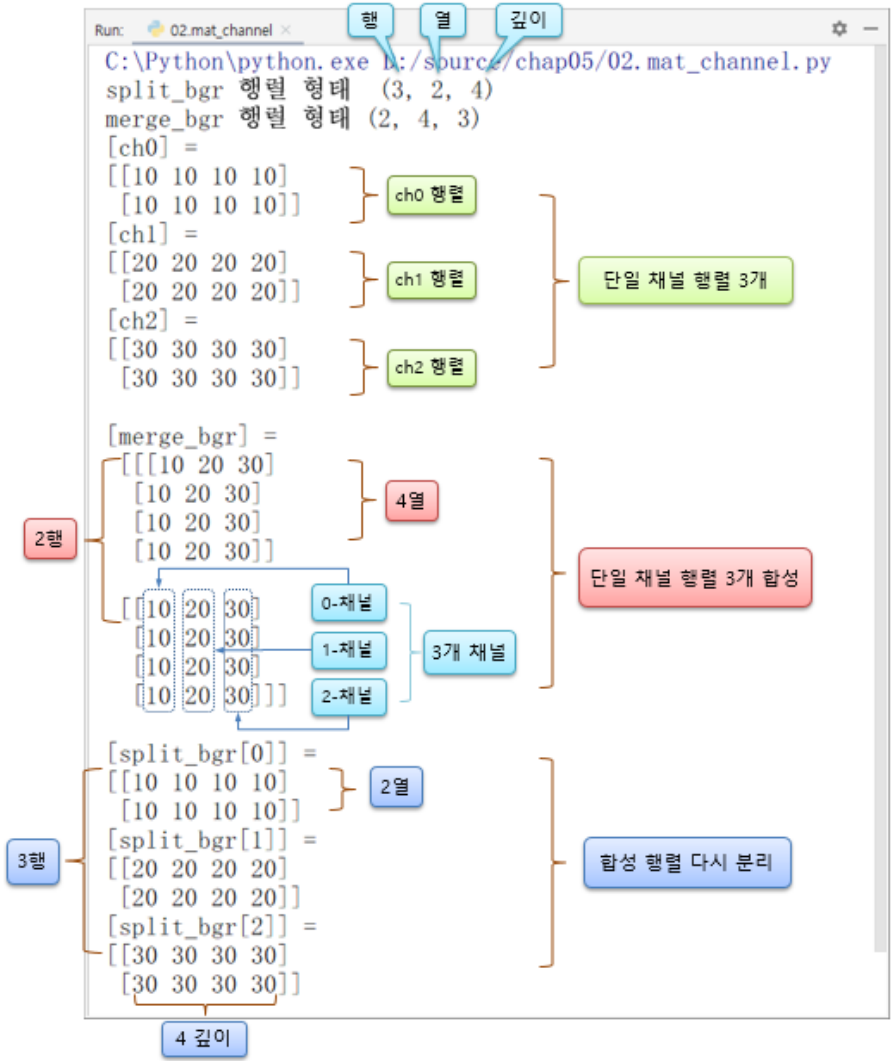
5.2 채널 처리 함수

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

```
import numpy as np
import cv2

## numpy.ndarray를 이용해 행렬 생성 및 초기화 방법
ch0 = np.zeros((2, 4), np.uint8) + 10          # 0 원소 행렬 선언 후 10 더하기
ch1 = np.ones((2, 4), np.uint8) * 20           # 1 원소 행렬 선언 후 20 곱하기
ch2 = np.full((2, 4), 30, np.uint8)            # 행렬을 생성하며 30으로 초기화

list_bgr = [ch0, ch1, ch2]                     # 단일채널 행렬들을 모아 리스트 구성
merge_bgr = cv2.merge(list_bgr)                # 채널 합성
split_bgr = cv2.split(merge_bgr)               # 채널 분리: 컬러 영상 → 3채널 분리

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

print("[split_bgr[0]] =\n%s " % split_bgr[0])  # 분리 채널 결과 출력
print("[split_bgr[1]] =\n%s " % split_bgr[1])
print("[split_bgr[2]] =\n%s " % split_bgr[2])
```

행렬의 형태 확인

5.2 채널 처리 함수


Run: 02.mat_channel
행
열
깊이
C:\Python\python.exe b:/source/chap05/02.mat_channel.py
split_bgr 행렬 형태 (3, 2, 4)
merge_bgr 행렬 형태 (2, 4, 3)
[ch0] =
[[10 10 10 10]
[10 10 10 10]]
ch0 행렬
[ch1] =
[[20 20 20 20]
[20 20 20 20]]
ch1 행렬
[ch2] =
[[30 30 30 30]
[30 30 30 30]]
ch2 행렬
단일 채널 행렬 3개
[merge_bgr] =
[[[10 20 30]
[10 20 30]
[10 20 30]
[10 20 30]]
4열
2행
[[10 20 30]
[10 20 30]
[10 20 30]
[10 20 30]]]
0-채널
1-채널
2-채널
3개 채널
단일 채널 행렬 3개 합성
[split_bgr[0]] =
[[10 10 10 10]
[10 10 10 10]]
2열
3행
[split_bgr[1]] =
[[20 20 20 20]
[20 20 20 20]]
[split_bgr[2]] =
[[30 30 30 30]
[30 30 30 30]]
합성 행렬 다시 분리
4 깊이

5.2 채널 처리 함수 (실습)

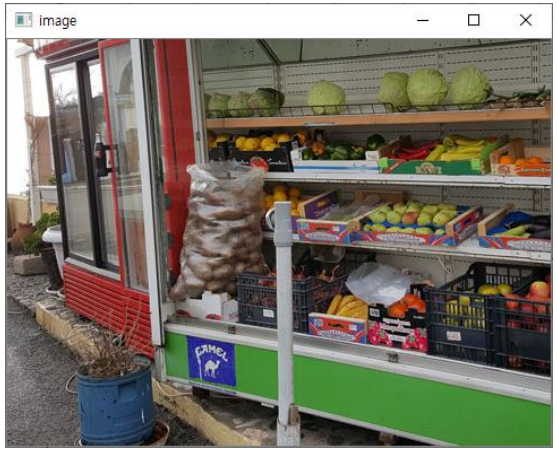
- 영상 채널 분리

예제 5.2.2 컬러 채널 분리 – 03.image_channels.py

```
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)
```

5.2 채널 처리 함수

- 실행결과



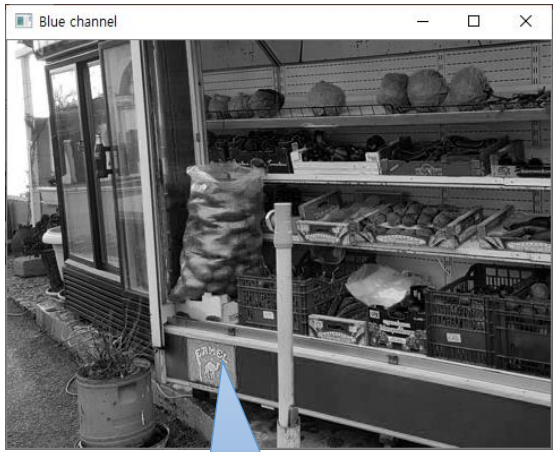
Run: 03.image_channels

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

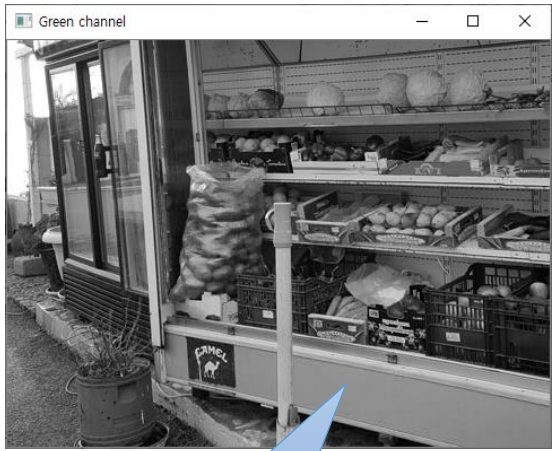
bgr 자료형

bgr 원소(단일채널) 자료형

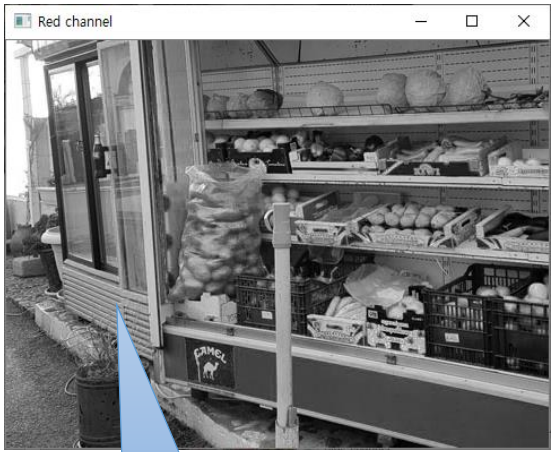
단일채널의 원소 자료형



파란색 - 밝은색



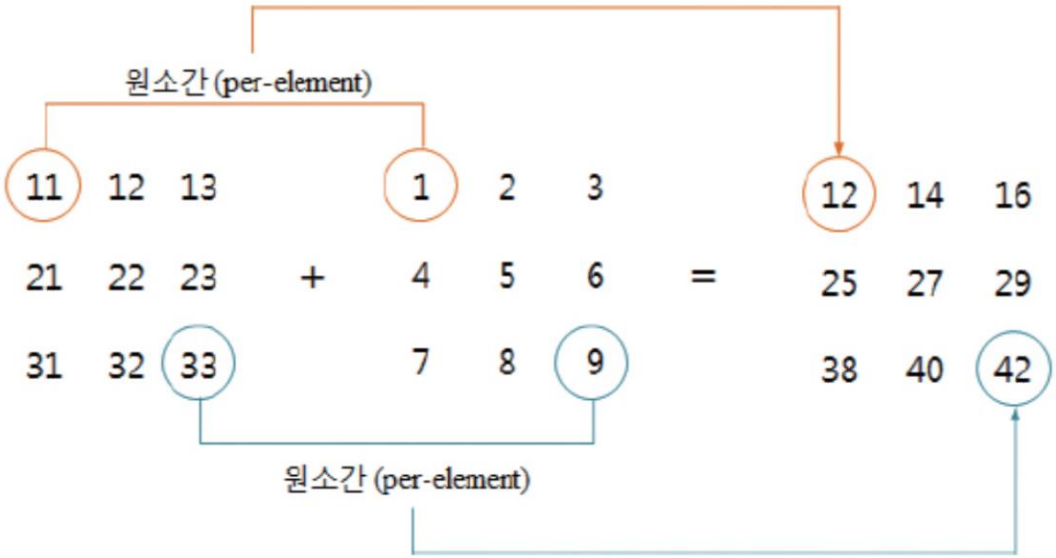
녹색 - 밝은색



붉은색 - 밝은색

5.3 산술 연산 함수

- 원소간(per-element, element-wise) 연산



〈그림 5.3.1〉 두 행렬의 원소 간 연산 과정

5.3.1 사칙 연산

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

5.3.1 사칙 연산 (실습)

예제 5.3.1 행렬 산술 연산 – 04.arithmethic_op.py

```
import numpy as np, cv2

m1 = np.full((3, 6), 10, np.uint8)                    # 단일채널 생성 및 초기화
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 후, 1을 할당

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

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

5.3.1 사칙 연산

- 실행 결과

```
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

5.3.2 지수, 로그, 제곱근 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 : $dst(i)=\begin{cases}\log\lvert src(i)\rvert & \text{if } src(i)\neq 0\\ c & otherwise\end{cases}$
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 : $dst(i)=\begin{cases}src(i)^{power} & \text{if } power \text{ is integer}\\ \lvert src(i)\rvert^{power} & otherwise\end{cases}$
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 : $x(i)=magnitude(i)\cdot\cos(angel(i))$

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

5.3.2 지수, 로그, 제곱근 관련 함수

예제 5.3.2 행렬 지수 및 로그 연산 – 05.exp_log.py

```python
import numpy as np, cv2

## ndarray 생성 예시
v1 = np.array([1, 2, 3], np.float32)                # 1차원 리스트로 행렬 생성
v2 = np.array([[1], [2], [3]], np.float32)          # 2차원 리스트(3행, 1열) – 열벡터
v3 = np.array([[1, 2, 3]], np.float32)              # 2차원 리스트(1행, 3열) – 행벡터

## OpenCV 산술 연산 함수 – 입력 인수로 ndarray 객체만 가능함
v_exp = cv2.exp(v1)                                 # 1차원 행렬에 대한 지수
m_exp = cv2.exp(v2)                                 # 행벡터(1×3)에 대한 지수 계산
m_exp = cv2.exp(v3)                                 # 열벡터(3×1)에 대한 지수 계산
v_log = cv2.log(v1)                                 # 로그 계산
m_sqrt= cv2.sqrt(v2)                                # 제곱근 계산
m_pow = cv2.pow(v3, 3)                              # 3의 거듭제곱 계산

## 행렬 정보 결과 출력
print("[v1] 형태: %s 원소: %s" % (v1.shape, v1))
print("[v2] 형태: %s 원소: %s" % (v2.shape, v2))
print("[v3] 형태: %s 원소: %s" % (v3.shape, v3))
print()
```

```python

## 행렬 정보 출력 – OpenCV 결과는 행렬로 반환됨
print("[v1_exp] 자료형: %s 형태: %s" % (type(v1_exp), v1_exp.shape))
print("[v2_exp] 자료형: %s 형태: %s" % (type(v2_exp), v2_exp.shape))
print("[v3_exp] 자료형: %s 형태: %s" % (type(v3_exp), v3_exp.shape))
print()

## 열벡터를 1행에 출력하는 예시
print("[log] =", log.T)                             # 전치하여 행벡터(1행, n열)로 변경
print("[sqrt] =", np.ravel(sqrt))                   # 전개하여 1차원 행렬로 변경
print("[pow] =", pow.flatten())                     # 전개하여 1차원 행렬로 변경
```

5.3.2 지수, 로그, 제곱근 관련 함수

- 실행결과

```
Run: 05.exp_log
C:\Python\python.exe D:/source/chap05/05.exp_log.py
[v1] 형태: (3,) 원소: [1. 2. 3.]
[v2] 형태: (3, 1) 원소:
[[1.]
 [2.]
 [3.]]
[v3] 형태: (1, 3) 원소: [[1. 2. 3.]]

[v1_exp] 자료형: <class 'numpy.ndarray'> 형태: (3, 1)
[v2_exp] 자료형: <class 'numpy.ndarray'> 형태: (3, 1)
[v3_exp] 자료형: <class 'numpy.ndarray'> 형태: (1, 3)

[log] = [[0.         0.6931472 1.0986123]]
[sqrt] = [1.         1.4142135 1.7320508]
[pow] = [ 1.  8. 27.]
```

3열 → 1차원 행렬

3행,1열 → 열벡터

1행, 3열 → 행벡터

OpenCV 함수에서 행렬로 반환하는 것은 대부분 ndarray 객체임

2차원 행렬로 반환

5.3.2 지수, 로그, 제곱근 관련 함수

예제 5.3.3 행렬 크기 및 위상 연산 – 06.magnitude.py

```
import numpy as np, cv2

x = np.array([1, 2, 3, 5, 10], np.float32)          # 리스트로 ndarray 객체 생성
y = np.array([2, 5, 7, 2, 9]).astype("float32")     # 행렬 생성 후 실수형 변환

mag = cv2.magnitude(x, y)                           # 크기 계산
ang = cv2.phase(x, y)                               # 각도(방향) 계산
p_mag, p_ang = cv2.cartToPolar(x, y)                # 극 좌표로 변환
x2, y2 = cv2.polarToCart(p_mag, p_ang)              # 직교좌표로 변환

print("[x] 형태: %s 원소: %s" % (x.shape, x) )      # 1차원 행렬
print("[mag] 형태: %s 원소: %s" % (mag.shape, mag) ) # 2차원 열벡터

print(">>>열벡터를 1행에 출력하는 방법")
print("[m_mag] = %s" % mag.T)                       # 행렬 전치
print("[p_mag] = %s" % np.ravel(p_mag))             # ravel() 함수로 전개
print("[p_ang] = %s" % np.ravel(p_ang))
print("[x_mat2] = %s" % x2.flatten())               # 2차원 행렬 전개
print("[y_mat2] = %s" % y2.flatten())
```

5.3.2 지수, 로그, 제곱근 관련 함수

- 실행 결과

```
Run: 06.magnitude
C:\Python\python.exe D:/source/chap05/06.magnitude.py
[x] 형태: (5,) 원소: [ 1.  2.  3.  5. 10.]
[mag] 형태: (5, 1) 원소: [[ 2.236068 ]
 [ 5.3851647]
 [ 7.615773 ]
 [ 5.3851647]
 [13.453624 ]]
>>>열벡터를 1행에 출력하는 방법
[m_mag] = [[ 2.236068   5.3851647  7.615773   5.3851647 13.453624 ]]
[p_mag] = [ 2.236068   5.3851647  7.615773   5.3851647 13.453624 ]
[p_ang] = [1.1071129 1.1902124 1.1658309 0.3805839 0.7329612]
[x_mat2] = [1.0000718 2.000388  3.0005162 4.999845  9.998685 ]
[y_mat2] = [1.9999641 4.9998446 6.999779  2.0003877 9.001461 ]
```

5.3.3 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

5.3.3 논리(비트) 연산 함수 (실습)

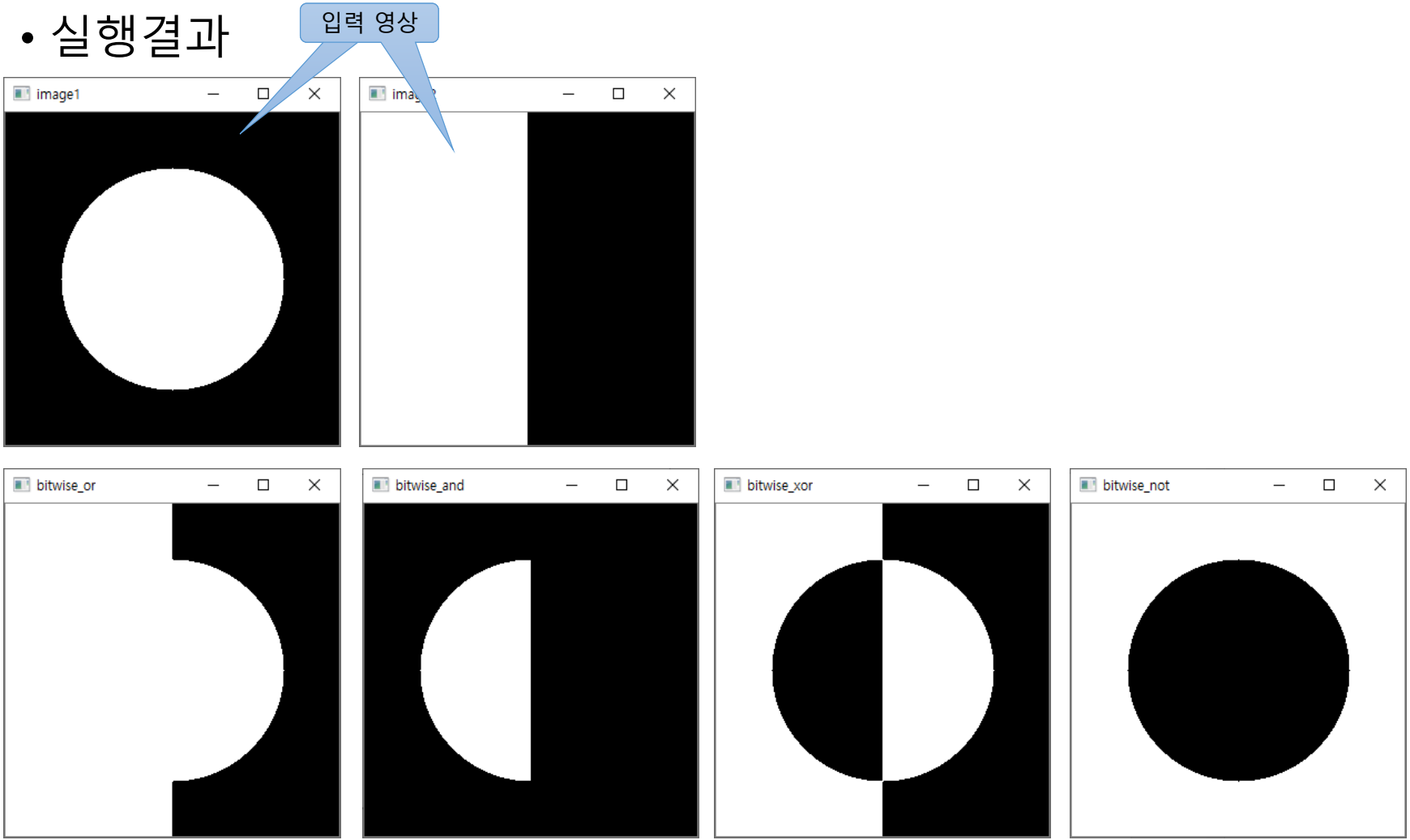
예제 5.3.4 행렬 비트 연산 – 07.bitwise_op.py

```
import numpy as np, cv2

image1 = np.zeros((300, 300), np.uint8)          # 300행, 300열 검은색(0) 영상 생성
image2 = image1.copy()                           # image1 복사

h, w = image1.shape[:2]
cx, cy = w//2, h//2                              # 중심 좌표
cv2.circle(image1, (cx, cy), 100, 255, -1)       # 중심에 원 그리기
cv2.rectangle(image2, (0, 0, cx, h), 255, -1)    # 영상의 가로 절반

image3 = cv2.bitwise_or(image1, image2)          # 원소 간 논리합
image4 = cv2.bitwise_and(image1, image2)         # 원소 간 논리곱
image5 = cv2.bitwise_xor(image1, image2)         # 원소 간 배타적 논리합
image6 = cv2.bitwise_not(image1)                 # 행렬 반전

cv2.imshow("image1", image1);        cv2.imshow("image2", image2)
cv2.imshow("bitwise_or", image3);    cv2.imshow("bitwise_and", image4)
cv2.imshow("bitwise_xor", image5);   cv2.imshow("bitwise_not", image6)
cv2.waitKey(0)
```

내부 채움

5.3.3 논리(비트) 연산 함수

- 실행결과



5.4.1 원소의 최솟값과 최댓값

함수 설명		
cv2.min(src1, src2[, dst]) → dst ■ 설명: 두 입력 배열의 원소 간 비교하여 작은 값을 출력 배열로 반환한다. ■ 수식 : $dst(i) = \min(src1(i), src2(i))$		
인수 설명	■ src1, src2	두 개의 입력 배열
	■ dst	계산 결과 출력 배열
cv2.max(src1, src2[, dst]) → dst ■ 설명: 두 입력 배열의 원소 간 비교하여 큰 값을 배열로 반환한다. ■ 수식: $dst(i) = \max(src1(i), src2(i))$		
cv2.minMaxLoc(src[, mask]) → minVal, maxVal, minLoc, maxLoc ■ 설명: 입력 배열에서 최솟값과 최댓값, 최솟값과 최댓값을 갖는 원소 위치을 반환한다.		
인수 설명	■ src	입력 배열
	■ minVal, maxVal	최솟값, 최댓값
	■ minLoc, maxLoc	최솟값, 최댓값을 갖는 원소 위치(정수형 튜플)

5.4.1 원소의 최솟값과 최댓값 (실습)

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

```
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) )
```

5.4.1 원소의 최솟값과 최댓값

- 실행결과

```
Run: 10.mat_min_max
C:\Python\python.exe D:/source/chap05/10.mat_min_max.py
[m1] =
[[ 10 200   5   7   9]
 [ 15  35  60  80 170]
 [100   2  55  37  70]]

[m_min] =
[[10 30  5  7  9]
 [15 30 30 30 30]
 [30  2 30 30 30]]

[m_max] =
[[ 50 200  50  50  50]
 [ 50  50  60  80 170]
 [100  50  55  50  70]]

m1 행렬 최솟값 좌표 (1, 2), 최솟값: 2
m1 행렬 최댓값 좌표 (1, 0), 최댓값: 200
```

vs. 30

vs.

50,	50,	50,	50,	50
50,	50,	50,	50,	50
50,	50,	50,	50,	50

[m2] 행렬

30보다 작은 값 유지

50보다 큰 값 유지

최소값 좌표 (1, 2)

최대값 좌표 (1,0)

5.5 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: $N=\sum_{i:src(i)\neq 0} 1$		

5.5 통계 관련 함수

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

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

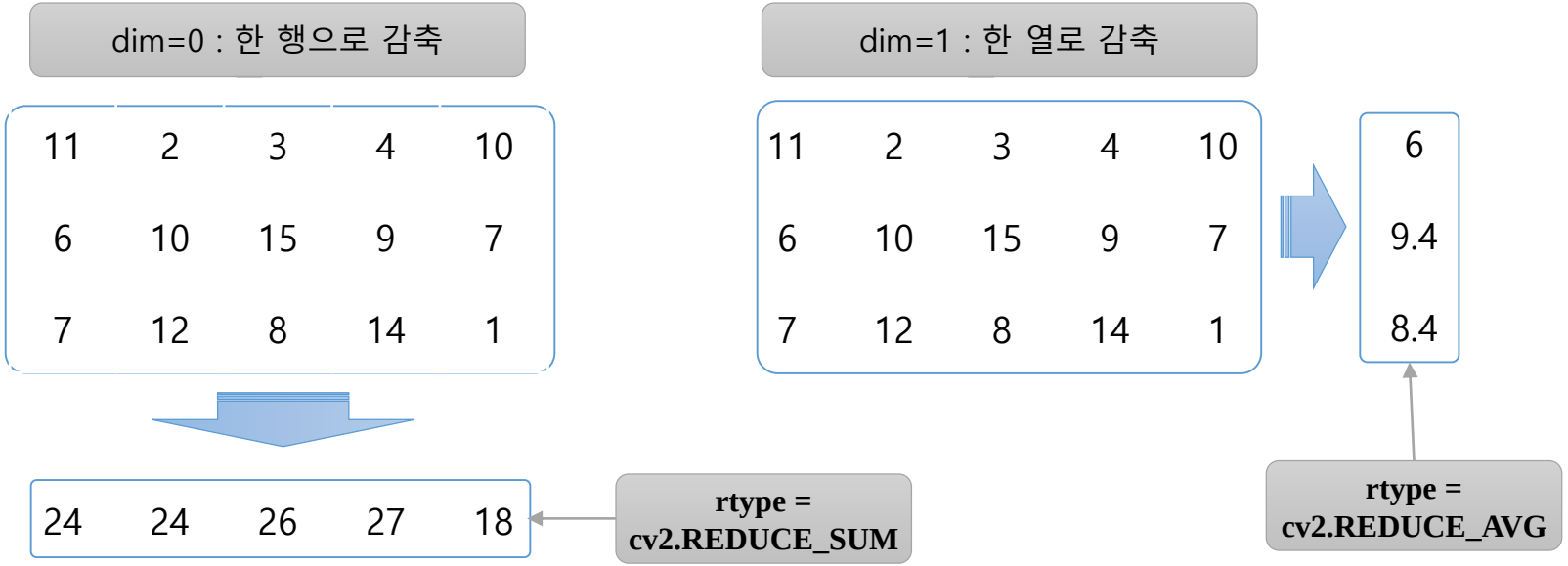
옵션 상수	값	설명
cv2.SORT_EVERY_ROW	0	각 행을 독립적으로 정렬
cv2.SORT_EVERY_COLUMN	1	각 열을 독립적으로 정렬
cv2.SORT_ASCENDING	0	오름차순으로 정렬
cv2.SORT_DESCENDING	16	내림차순으로 정렬

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.

5.5 통계 관련 함수

- reduce() 함수 감축 방향



5.5 통계 관련 함수

예제 5.5.1 행렬 합/평균 연산 – 12.sum_avg.py

```
01  import numpy as np, cv2
02
03  image = cv2.imread("images/sum_test.jpg", cv2.IMREAD_COLOR)
04  if image is None: raise Exception("영상파일 읽기 오류 발생")
05
06  mask = np.zeros(image.shape[:2], np.uint8)
07  mask[60:160, 20:120] = 255                          # 관심 영역에 값(255) 할당
08
09  sum_value    = cv2.sumElems(image)                  # 채널별 합 – 튜플로 반환
10  mean_value1 = cv2.mean(image)                       # 채널별 평균– 튜플로 반환
11  mean_value2 = cv2.mean(image, mask)
12
13  print("sum_value 자료형:", type(sum_value), type(sum_value[0]))    # 결과 행렬의 자료형
14  print("[sum_value] =", sum_value)
15  print("[mean_value1] =", mean_value1)
16  print("[mean_value2] =", mean_value2)
```

5.5 통계 관련 함수

```
17  print()
18
19  ## 평균과 표준편차 결과 저장
20  mean, stddev = cv2.meanStdDev(image)                    # 2 원소 튜플로 반환
21  mean2, stddev2 = cv2.meanStdDev(image, mask=mask)       # 마스크가 255인 영역만 계산
22  print("mean 자료형:", type(mean), type(mean[0][0]))      # 반환 행렬 자료형, 원소 자료형
23  print("[mean] =",  mean.flatten())                      # 벡터 변환 후 출력
24  print("[stddev] =", stddev.flatten())
25  print()
26
27  print("[mean2] =",  mean2.flatten())
28  print("[stddev2] =", stddev2.flatten())
29
30  cv2.imshow('image', image)
31  cv2.imshow('mask', mask)
32  cv2.waitKey(0)
```

5.5 통계 관련 함수

- 실행 결과

```
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)

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 ]
```

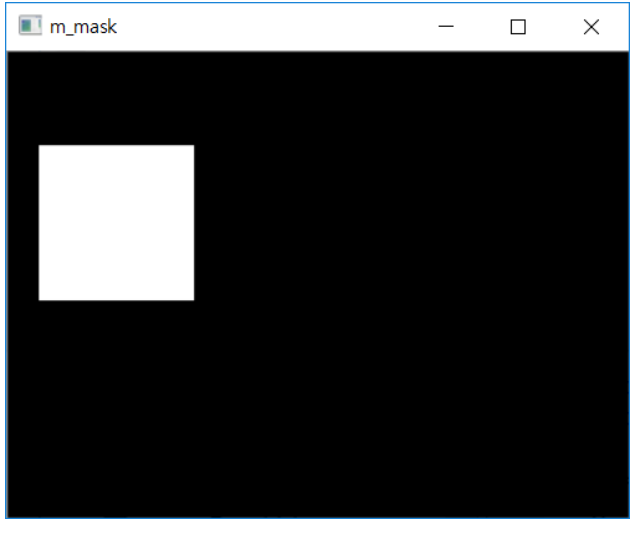
4 원소 튜플 반환 , 마지막 원소 0

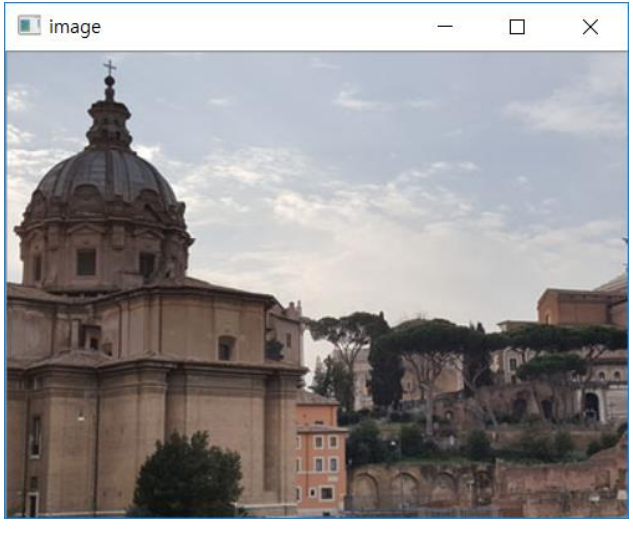
전체 영역 평균

관심 영역 평균

전체영역 평균, 표준편차

관심영역 평균, 표준편차

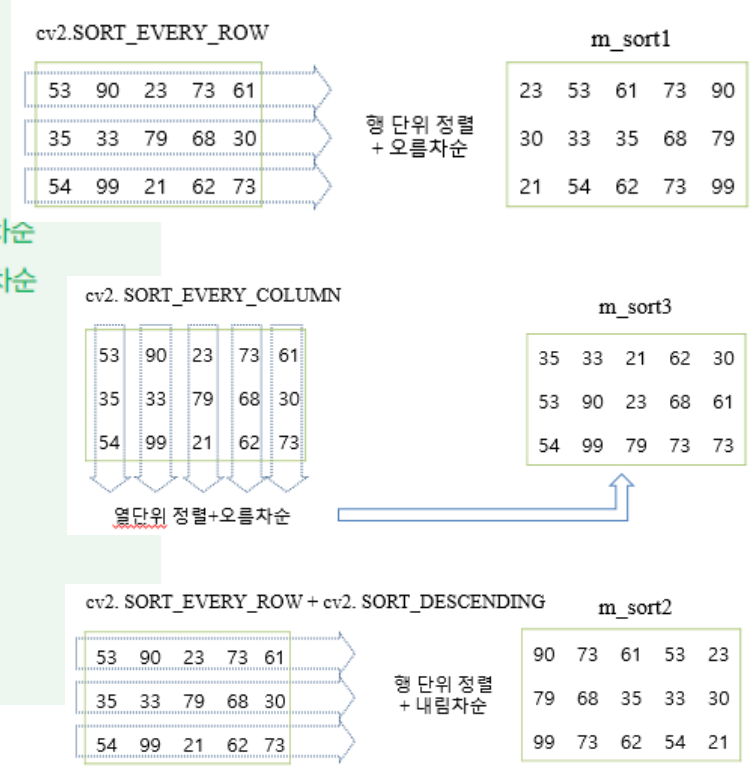




5.5 통계 관련 함수 (정렬)

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)))
```



5. 실습 과제

- (과제) p211. 7번.
  - 다음의 컬러 영상파일(logo.jpg)을 입력 받아서 RGB의 3개 채널을 분리하고, 각 채널을 컬러 영상으로 윈도우에 표시해 보자. 즉, Red 채널은 빨간색으로, Green 채널은 초록색으로, Blue 채널은 파란색으로 표현되도록 다음의 프로그램을 완성하시오.

```python
import numpy as np, cv2

logo  = cv2.imread("images/logo.jpg", cv2.IMREAD_COLOR)
if logo is None: raise Exception("영상 파일 읽기 오류 ")

blue, green, red = cv2.split(logo)
zero = np.zeros(logo.shape[:2], np.uint8)

cv2.imshow("logo", logo)
cv2.imshow("blue_img", blue_img)
cv2.imshow("green_img", green_img)
cv2.imshow("red_img", red_img)
cv2.waitKey()
```

- (보너스) p213. 12번.
  - 영상파일을 읽어서 메인 윈도우에 다음과 같이 출력하시오.
    - 관심영역 2개 선정 후, 한곳은 밝기를 50증가 / 다른곳은 영상의 화소 대비를 증가

5. 실습 규칙

- 실습 과제는 실습 시간내로 해결해야 합니다.
  - 해결 못한경우 실습 포인트를 얻지 못합니다.
  - -> 집에서 미리 예습하고 오길 권장합니다.

- 코드 공유/보여주기 금지. 의논 가능.

- 보너스문제까지 해결한 학생은 조기 퇴실 가능