디지털 영상처리

OpenCV 기본 연산

학습목표

- 기본 배열(Array) 처리 함수
- 채널 처리 함수
- 산술 연산 함수
- 원소의 절댓값 연산
- 통계 관련 함수
- 행렬 연산 함수

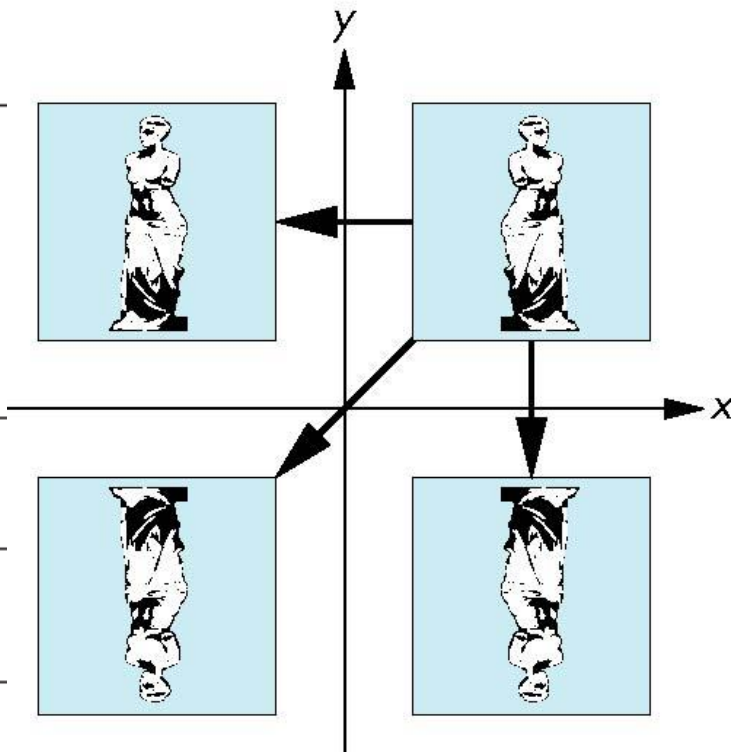
기본 배열(Array) 처리 함수

- 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

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



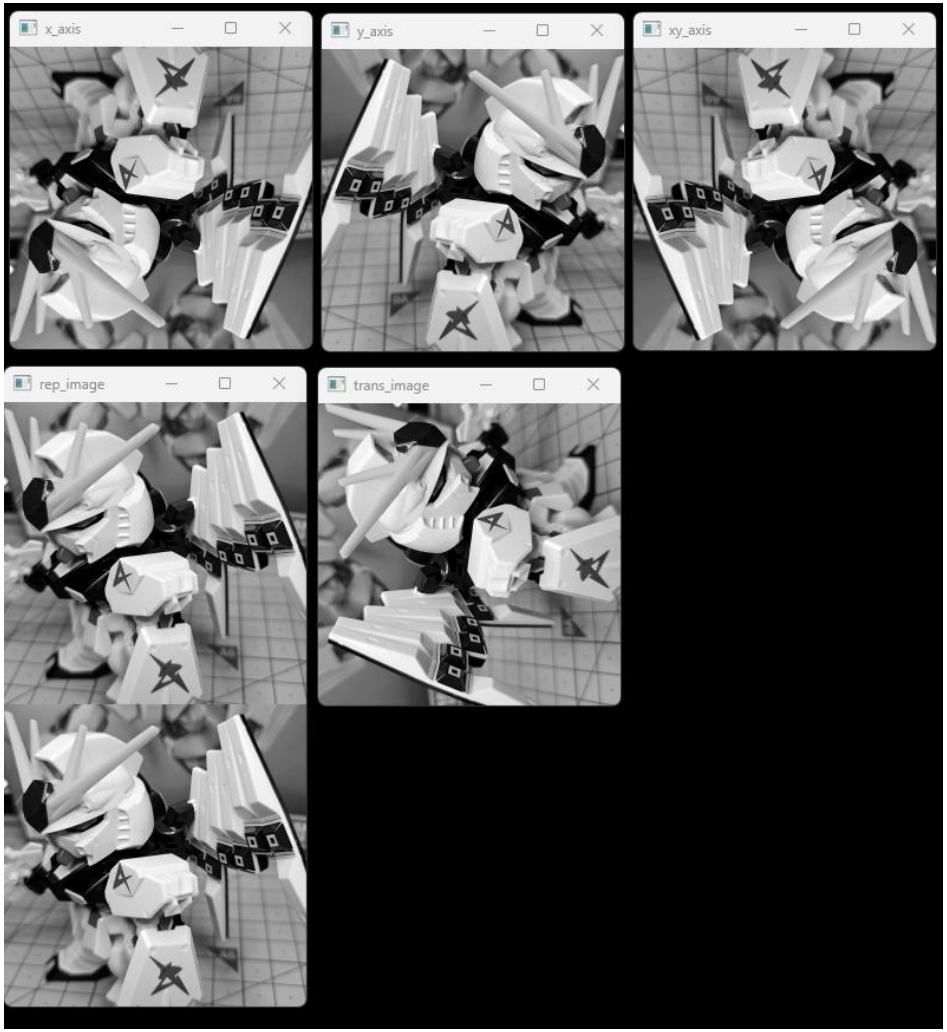
기본 배열(Array) 처리 함수

예제: 01.mat_array

```
import cv2

# 이미지 읽기
img_path = "images/gundam.jpg"
image = cv2.imread(img_path, cv2.IMREAD_COLOR)
if image is None:
    raise Exception("Error: Unable to read the image file.")  # 예외 처리

# 이미지 변환
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)               # 상하좌우 뒤집기
rep_image = cv2.repeat(image, 2, 1)         # Y축으로 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)
cv2.destroyAllWindows()
```

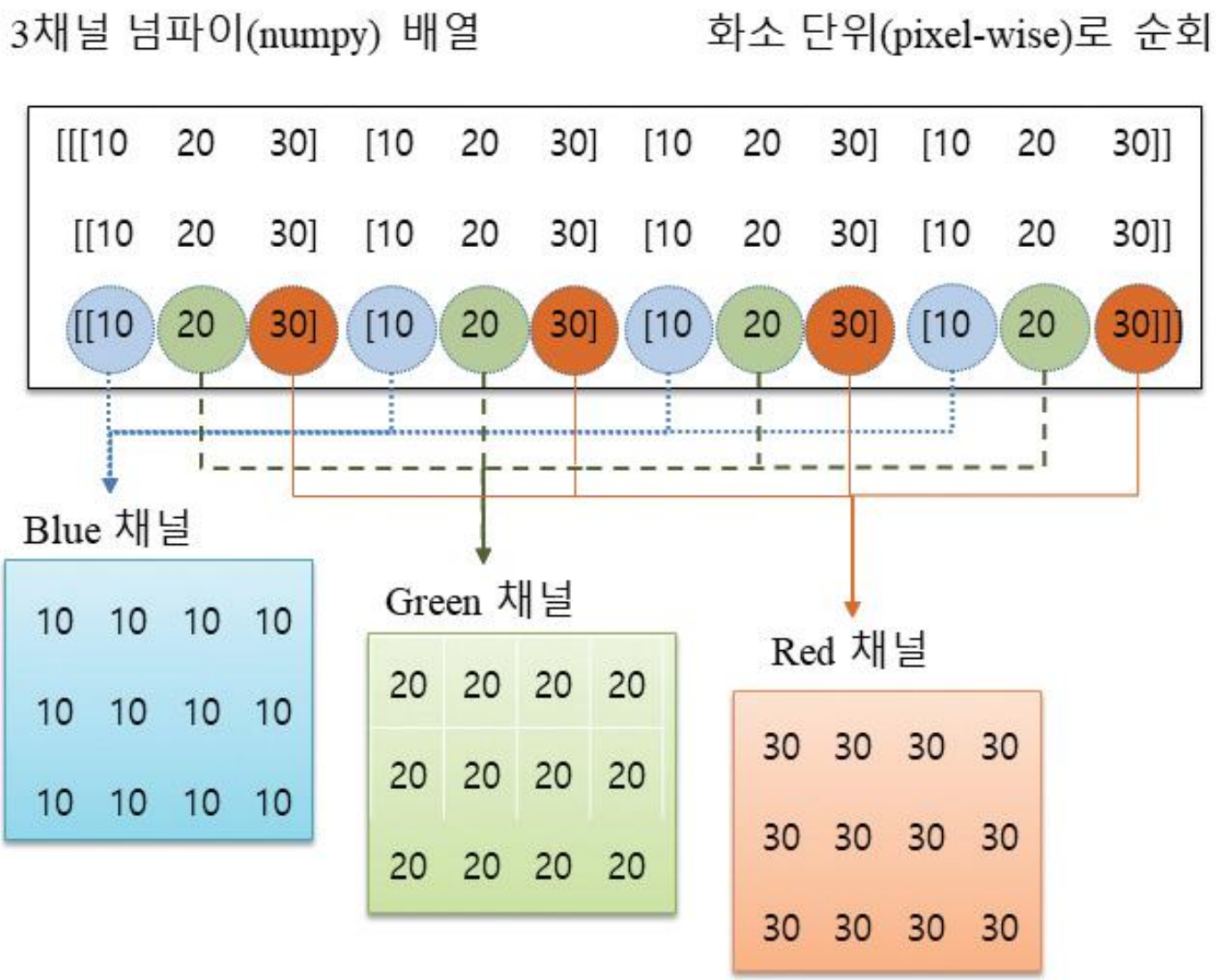
기본 배열(Array) 처리 함수

- 실행결과



채널 처리 함수

- 채널 개념


3채널 넘파이(numpy) 배열
화소 단위(pixel-wise)로 순회
[[[10 20 30] [10 20 30] [10 20 30] [10 20 30]]
[[10 20 30] [10 20 30] [10 20 30] [10 20 30]]
[[10 20 30] [10 20 30] [10 20 30] [10 20 30]]]
Blue 채널
10 10 10 10
10 10 10 10
10 10 10 10
Green 채널
20 20 20 20
20 20 20 20
20 20 20 20
Red 채널
30 30 30 30
30 30 30 30
30 30 30 30

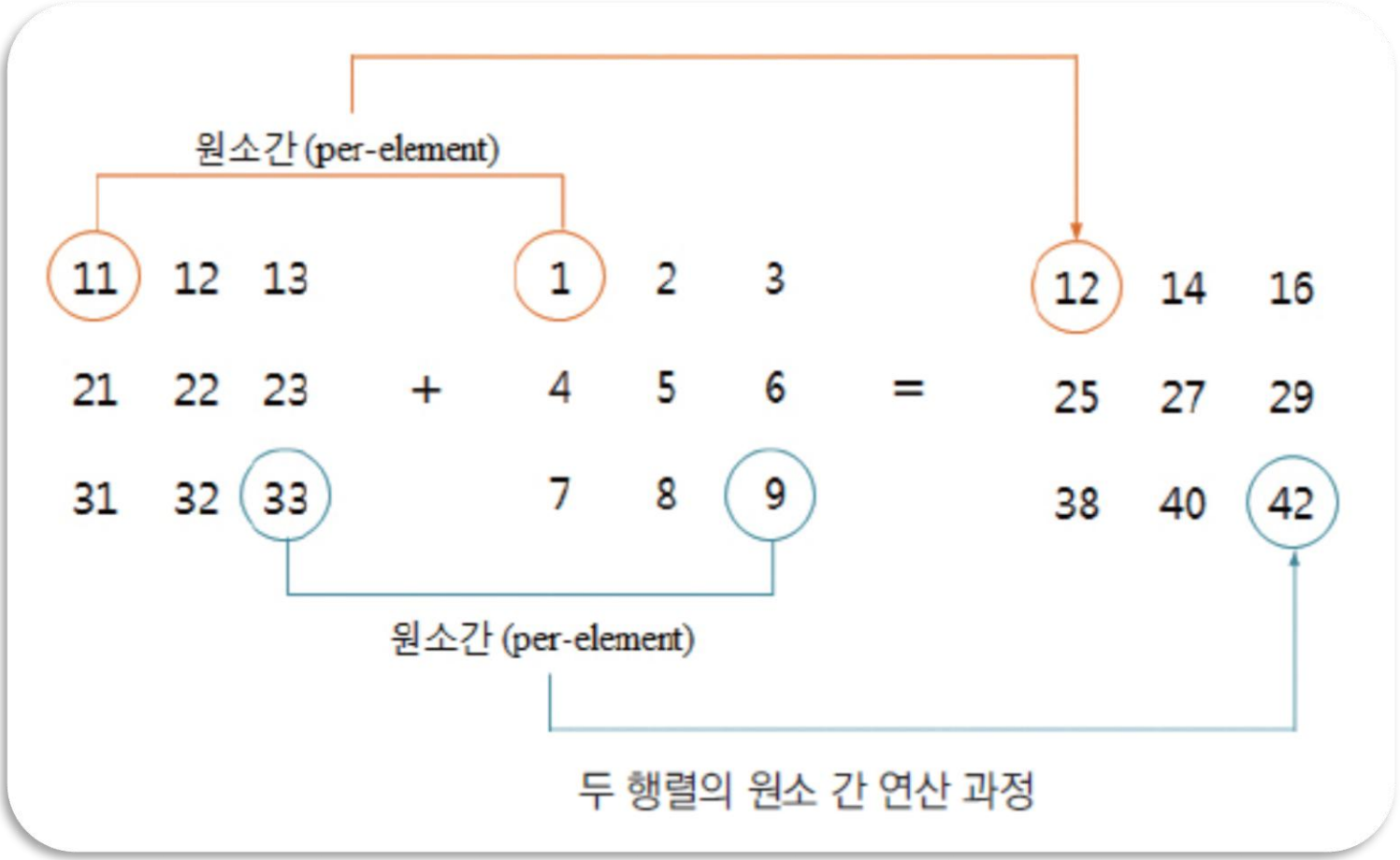
채널 처리 함수

- 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

산술 연산 함수

- 원소 간(per-element, element-wise) 연산



두 행렬의 원소 간 연산 과정

산술 연산

예제: 03.image_channels

```
import cv2

# 이미지 읽기
image_path = "images/lenna.jpg"
image = cv2.imread(image_path, cv2.IMREAD_COLOR)
if image is None:
    raise Exception("Error: Unable to read the image file.")  # 예외 처리
if image.ndim != 3:
    raise Exception("Error: The image is not a color image.")  # 컬러 영상인지 확인

# 채널 분리
blue_channel = image[:, :, 0]
green_channel = image[:, :, 1]
red_channel = image[:, :, 2]

# 자료형 체크 및 정보 출력
print("blue_channel 자료형:", type(blue_channel), blue_channel.dtype)
print("green_channel 자료형:", type(green_channel), green_channel.dtype)
print("red_channel 자료형:", type(red_channel), red_channel.dtype)

# 각 채널을 윈도우에 띄우기
cv2.imshow("Original Image", image)
cv2.imshow("Blue Channel", blue_channel)  # Blue 채널
cv2.imshow("Green Channel", green_channel)  # Green 채널
cv2.imshow("Red Channel", red_channel)  # Red 채널

cv2.waitKey(0)
cv2.destroyAllWindows()
```

사칙 연산


```
rad/python.exe c:/Users/nicesk/Desktop/OpenCV/OpenCV_
blue_channel 자료형: <class 'numpy.ndarray'> uint8
green_channel 자료형: <class 'numpy.ndarray'> uint8
red_channel 자료형: <class 'numpy.ndarray'> uint8
```

사칙 연산

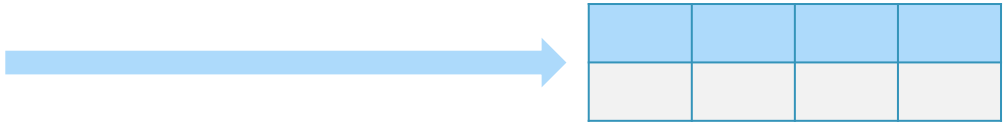
예제: 04.arithmethic_op

```
import numpy as np
import cv2

# 단일 채널 3개 생성
ch0 = np.full((2, 4), 10, np.uint8)  # 10으로 채운 행렬
ch1 = np.full((2, 4), 20, np.uint8)  # 20으로 채운 행렬
ch2 = np.full((2, 4), 30, np.uint8)  # 30으로 채운 행렬

# 단일 채널 리스트 구성
list_bgr = [ch0, ch1, ch2]

# 채널 합성
merge_bgr = cv2.merge(list_bgr)

# 채널 분리
split_bgr = cv2.split(merge_bgr)

# 결과 출력
print('split_bgr 행렬 형태:', np.array(split_bgr).shape)
print('merge_bgr 행렬 형태:', merge_bgr.shape)

print("[ch0] = \n", ch0)
print("[ch1] = \n", ch1)
print("[ch2] = \n", ch2)
print("[merge_bgr] = \n", merge_bgr)

for i, ch in enumerate(split_bgr):
    print(f"[split_bgr[{i}]] = \n{ch}")
```


그려보자

사칙 연산

```
import numpy as np
import cv2

# 단일 채널 3개 생성
ch0 = np.full((2, 4), 10, np.uint8)  # 10으로 채운 행렬
ch1 = np.full((2, 4), 20, np.uint8)  # 20으로 채운 행렬
ch2 = np.full((2, 4), 30, np.uint8)  # 30으로 채운 행렬

# 단일 채널 리스트 구성
list_bgr = [ch0, ch1, ch2]

# 채널 합성
merge_bgr = cv2.merge(list_bgr)

# 채널 분리
split_bgr = cv2.split(merge_bgr)

# 결과 출력
print('split_bgr 행렬 형태:', np.array(split_bgr).shape)
print('merge_bgr 행렬 형태:', merge_bgr.shape)

print("[ch0] = \n", ch0)
print("[ch1] = \n", ch1)
print("[ch2] = \n", ch2)
print("[merge_bgr] = \n", merge_bgr)

for i, ch in enumerate(split_bgr):
    print(f"[split_bgr[{i}]] = \n{ch}")
```



사칙 연산

```
import numpy as np
import cv2

# 단일 채널 3개 생성
ch0 = np.full((2, 4), 10, np.uint8)  # 10으로 채운 행렬
ch1 = np.full((2, 4), 20, np.uint8)  # 20으로 채운 행렬
ch2 = np.full((2, 4), 30, np.uint8)  # 30으로 채운 행렬

# 단일 채널 리스트 구성
list_bgr = [ch0, ch1, ch2]

# 채널 합성
merge_bgr = cv2.mer

# 채널 분리
split_bgr = cv2.spl

# 결과 출력
print('split_bgr 행
print('merge_bgr 행

print("[ch0] = \n",
print("[ch1] = \n",
print("[ch2] = \n",
print("[merge_bgr]

for i, ch in enumer
    print(f"[split_
```

```
PROBLEMS 1  OUTPUT  DEBUG CONSOLE  TERMINAL

 [10 10 10 10]]
[split_bgr[1]] =
[[20 20 20 20]
 [20 20 20 20]]
[split_bgr[2]] =
[[30 30 30 30]
 [30 30 30 30]]
PS C:\Users\nicesk\Desktop\OpenCV\OpenCV_Code\S
```

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

Saturation연산: 한계 값을 정하고, 그 값을 벗어나는 경우는 모두 특정 값으로 계산하는 방식. 이미지에서는 0이하는 모두 0, 255이상은 모두 255로 표현

Modulo연산: ‘a와 b는 n으로 나눈 나머지 값이 같다’ 라는 의미 이미지에서는 연산의 결과가 256보다 큰 경우는 256으로 나눈 나머지 값으로 결정

사칙 연산

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 : $dst(i) = saturate(src1(i) - src2(i)) \quad \text{if } mask(i) \neq 0$

  $dst(i) = saturate(src1 \quad - src2(i)) \quad \text{if } mask(i) \neq 0$

  $dst(i) = saturate(src1(i) - src2 \quad) \quad \text{if } mask(i) \neq 0$

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: $dst(i) = scale/src2(i)$

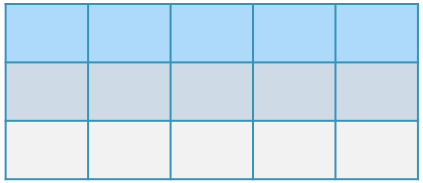
cv2.addWeighted(src1, alpha, src2, beta, gamma[, dst[, dtype]]) → dst

- 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

사칙 연산

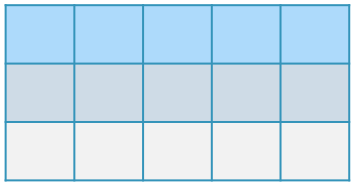
```
import numpy as np
import cv2

# 단일 채널 생성 및 초기화
matrix1 = np.full((3, 5), 20, np.uint8)
matrix2 = np.full((3, 5), 40, np.uint8)

# 마스크 생성
mask = np.zeros(matrix1.shape, np.uint8)
mask[:, 1:4] = 1  # 관심 영역을 지정한 후, 1을 할당
```

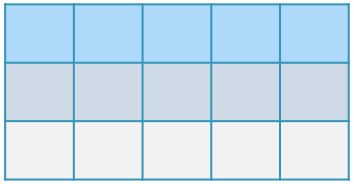


직접 코드를 생성해 보자!!!!

사칙 연산

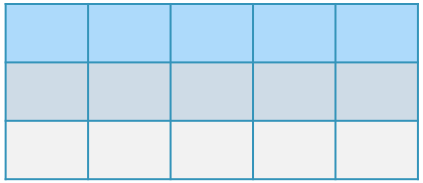
```
import numpy as np
import cv2

# 단일 채널 생성 및 초기화
matrix1 = np.full((3, 5), 20, np.uint8)
matrix2 = np.full((3, 5), 40, np.uint8)

# 마스크 생성
mask = np.zeros(matrix1.shape, np.uint8)
mask[:, 1:4] = 1  # 관심 영역을 지정한 후, 1을 할당
```





사칙 연산

```
import numpy as np
import cv2

# 단일 채널 생성 및 초기화
matrix1 = np.full((3, 5), 20, np.uint8)
matrix2 = np.full((3, 5), 40, np.uint8)

# 마스크 생성
mask = np.zeros(matrix1.shape, np.uint8)
mask[:, 1:4] = 1  # 관심 영역을 지정한 후, 1을 할당
```



0	1	1	1	0
0	1	1	1	0
0	1	1	1	0

사칙 연산

```
import numpy as np
import cv2

# 단일 채널 생성 및 초기화
matrix1 = np.full((3, 5), 20, np.uint8)
matrix2 = np.full((3, 5), 40, np.uint8)

# 마스크 생성
mask = np.zeros(matrix1.shape, np.uint8)
mask[:, 1:4] = 1  # 관심 영역을 지정한 후, 1을 할당

# 행렬 덧셈
add_result1 = cv2.add(matrix1, matrix2)
add_result2 = cv2.add(matrix1, matrix2, mask=mask)  # 관심 영역만 덧셈 수행

# 행렬 나눗셈
div_result = cv2.divide(matrix1, matrix2)

# 결과 출력
titles = ['matrix1', 'matrix2', 'mask', 'add_result1', 'add_result2', 'div_result']
for title in titles:
    print(f"[{title}] = \n{eval(title)} \n")
```

60	60	60	60	60
60	60	60	60	60
60	60	60	60	60

0	1	1	1	0
0	1	1	1	0
0	1	1	1	0

사칙 연산

- 실행 결과

```
[mask] =
[[0 1 1 1 0]
 [0 1 1 1 0]
 [0 1 1 1 0]]

[add_result1] =
[[60 60 60 60 60]
 [60 60 60 60 60]
 [60 60 60 60 60]]

[add_result2] =
[[ 0 60 60 60  0]
 [ 0 60 60 60  0]
 [ 0 60 60 60  0]]

[div_result] =
[[0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]]

PS C:\Users\nicesk\Desktop\O
```

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 : $magnitude(i)=\sqrt{x(i)^2+y(i)^2}$		

지수, 로그, 제곱근 관련 함수

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

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

■설명: 2차원 배열의 회전 각도를 계산한다.

수식 : $angle(i) = \mathrm{a\,tan\,2}(y(i), x(i)) \cdot [180/\pi]$

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

- x — 벡터의 x 좌표를 나타내는 실수 행렬 또는 벡터
- y — 벡터의 y 좌표를 나타내는 실수 행렬 또는 벡터. x와 크기와 타입이 같아야 합니다.
- angle — 벡터의 방향을 나타내는 실수 행렬 또는 벡터. x와 같은 크기, 같은 타입을 갖습니다.
- angleInDegrees — 이 값이 true이면 각도(degree) 단위를 사용하고, false이면 라디안(radian) 단위를 사용합니다.

$$\mathrm{angle}(I) = \mathrm{atan2}\left(\frac{\mathrm{y}(I)}{\mathrm{x}(I)}\right)$$

지수, 로그, 제곱근 관련 함수

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

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

■설명: 2차원 배열의 회전 각도를 계산한다.

수식 : $angle(i) = \mathrm{a\,tan\,2}(y(i), x(i)) \cdot [180/\pi]$

인수	■angle	각도들의 출력 배열
설명	■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

논리(비트) 연산 함수

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)
```

내부 채움

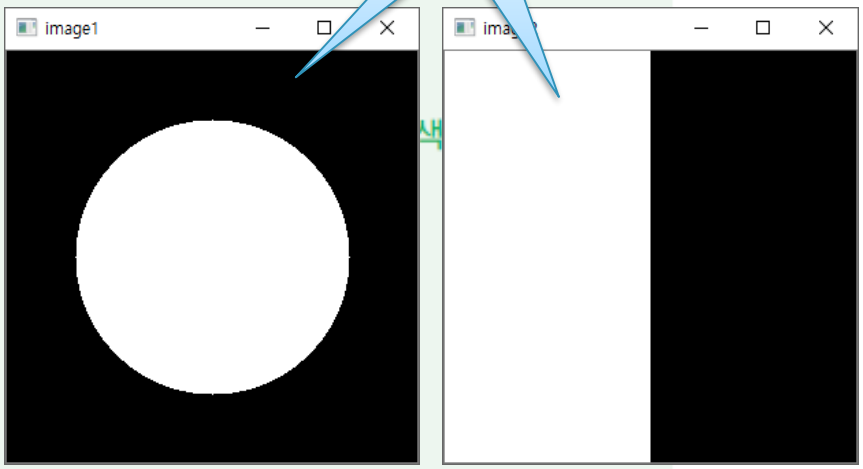
논리(비트) 연산 함수

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)
04  image2 = image1.copy()
05
06  h, w = image1.shape[:2]
07  cx, cy = w//2, h//2
08  cv2.circle(image1, (cx, cy), 100, 255, -1)
09  cv2.rectangle(image2, (0, 0, cx, h), 255, -1)
10
11  image3 = cv2.bitwise_or(image1, image2)
12  image4 = cv2.bitwise_and(image1, image2)
13  image5 = cv2.bitwise_xor(image1, image2)
14  image6 = cv2.bitwise_not(image1)
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)
```

내부 채움

입력 영상



영상의 가로 절반

원소 간 논리합

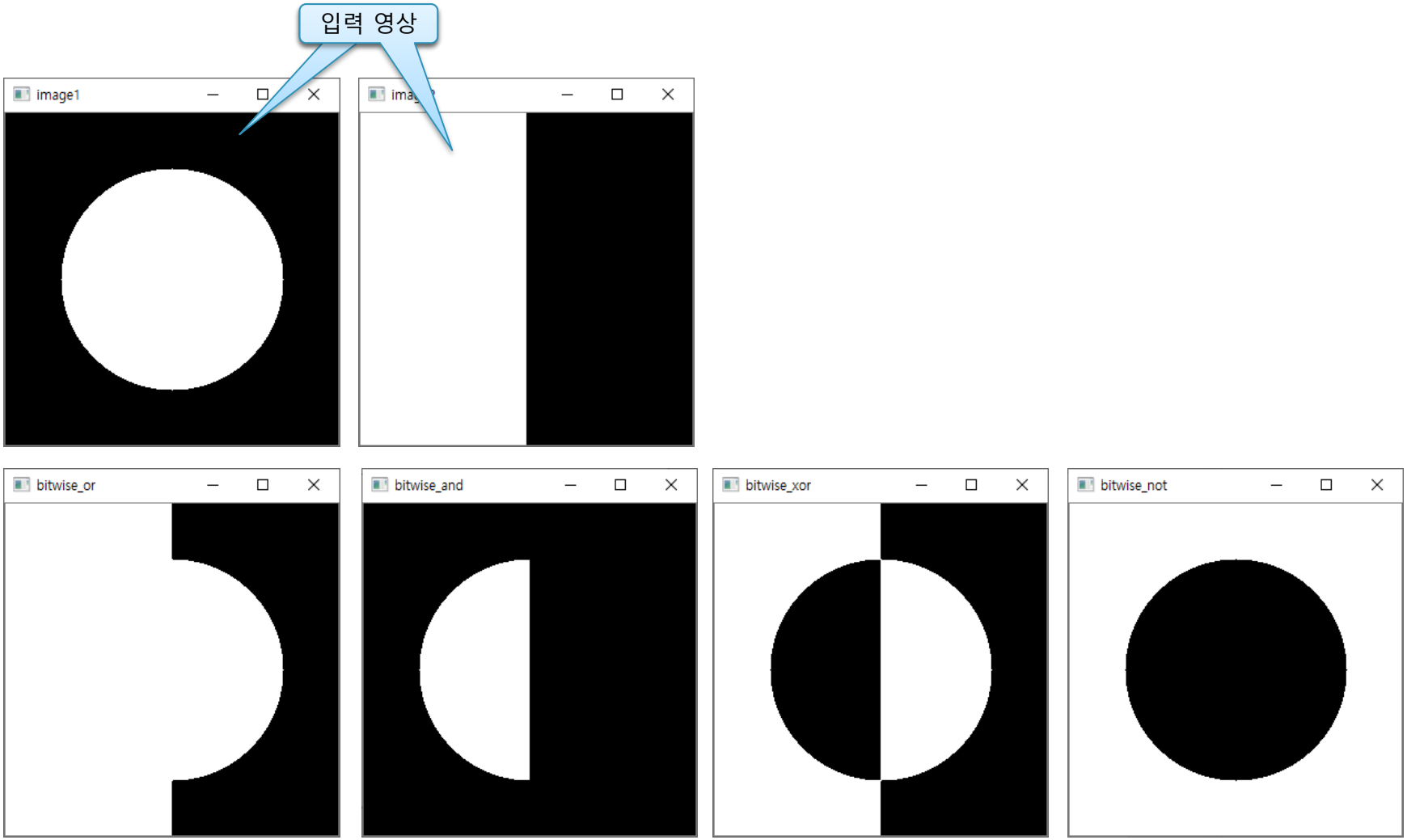
원소 간 논리곱

원소 간 배타적 논리합

행렬 반전

논리(비트) 연산 함수

- 실행결과



원소의 절댓값 연산

함수 설명		
cv2.absdiff(src1, src2[, dst]) → dst ■ 설명: 두 배열간 각 원소간(per-element) 차분 절댓값을 계산한다. src1, src2 중 하나는 스칼라값이 될 수 있다. ■ 수식 : $dst(i)=saturate\|src1(i)-src2(i)\|$ $dst(i)=saturate\|src1(i)-src2\|$ $dst(i)=saturate\|src1-src2(i)\|$		
인수 설명	■ src1, src2	첫 번째 입력 배열, 두 번째 입력 배열
	■ dst	계산된 결과 출력 배열
cv2.convertScaleAbs(src[, dst[, alpha[, beta]]]) → dst ■ 설명: 입력 배열의 각 원소에 alpha만큼 배율을 곱하고 beta 만큼 더한 후에 절댓값을 계산한 결과를 8비트 자료형으로 변환한다. ■ 수식 : $dst(i)=saturate\ cast<uchar>(\|src(i)*\alpha+\beta\|)$		
인수 설명	■ alpha	입력 배열의 각 원소에 곱해지는 스케일 팩터(scale factor)
	■ beta	스케일된 값에 더해지는 델타 옵션

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 ■ dst	두 개의 입력 배열 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 ■ minVal, maxVal ■ minLoc, maxLoc	입력 배열 최솟값, 최댓값 최솟값, 최댓값을 갖는 원소 위치(정수형 튜플)

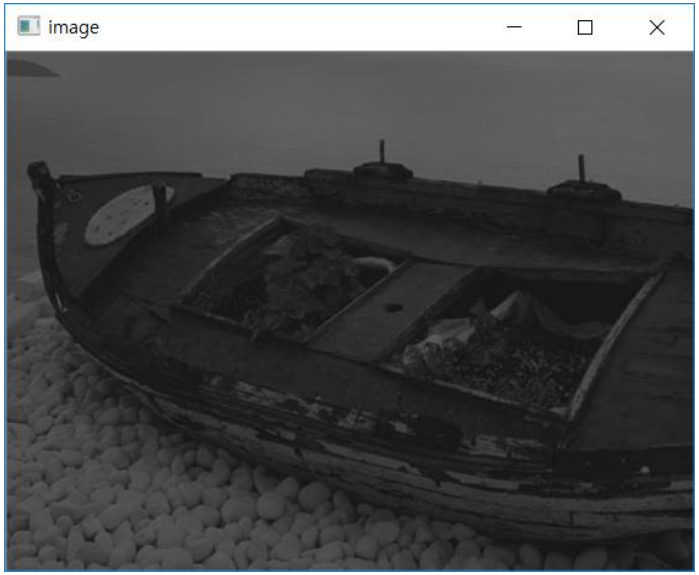
원소의 최솟값과 최댓값

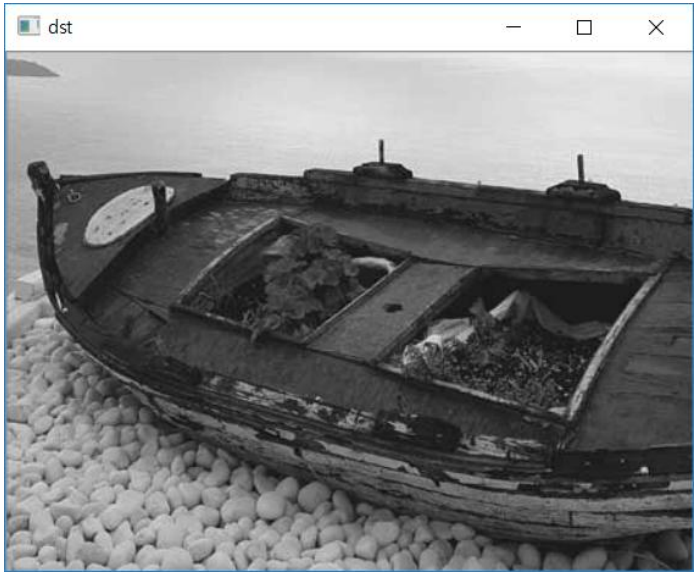
심화예제 5.4.3 영상 최소값 최대값 연산 – 11.image_min_max.py

```
01  import numpy as np, cv2
02
03  image = cv2.imread("images/minMax.jpg", cv2.IMREAD_GRAYSCALE)
04  if image is None: raise Exception("영상파일 읽기 오류 발생")
05
06  min_val, max_val, _, _ = cv2.minMaxLoc(image)  # 최솟값과 최댓값 가져오기
07
08  ratio = 255 / (max_val - min_val)
09  dst = np.round((image - min_val) * ratio).astype('uint8')
10  min_dst, max_dst, _, _ = cv2.minMaxLoc(dst)
11
12  print("원본 영상 최솟값= %d, 최댓값= %d" % (min_val, max_val))
13  print("수정 영상 최솟값= %d, 최댓값= %d" % (min_dst, max_dst))
14  cv2.imshow('image', image)
15  cv2.imshow('dst', dst)
16  cv2.waitKey(0)
```

원소의 최솟값과 최댓값

```
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
```





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 ■ mask	1개에서 4개 채널을 갖는 입력 배열 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 ■ mean ■ stddev ■ mask	1개에서 4개 채널을 갖는 입력 배열 계산된 평균이 반환되는 출력 인수, np.float64형으로 반환 계산된 표준편차가 반환되는 출력 인수, np.float64형으로 반환 연산 마스크 – 마스크가 0이 아닌 좌표만 연산 수행
cv2.countNonZero(src) → retval ■ 설명: 0이 아닌 배열 원소를 개수N을 반환한다. ■ 수식: $N=\sum_{i:src(i)\neq 0} 1$		

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

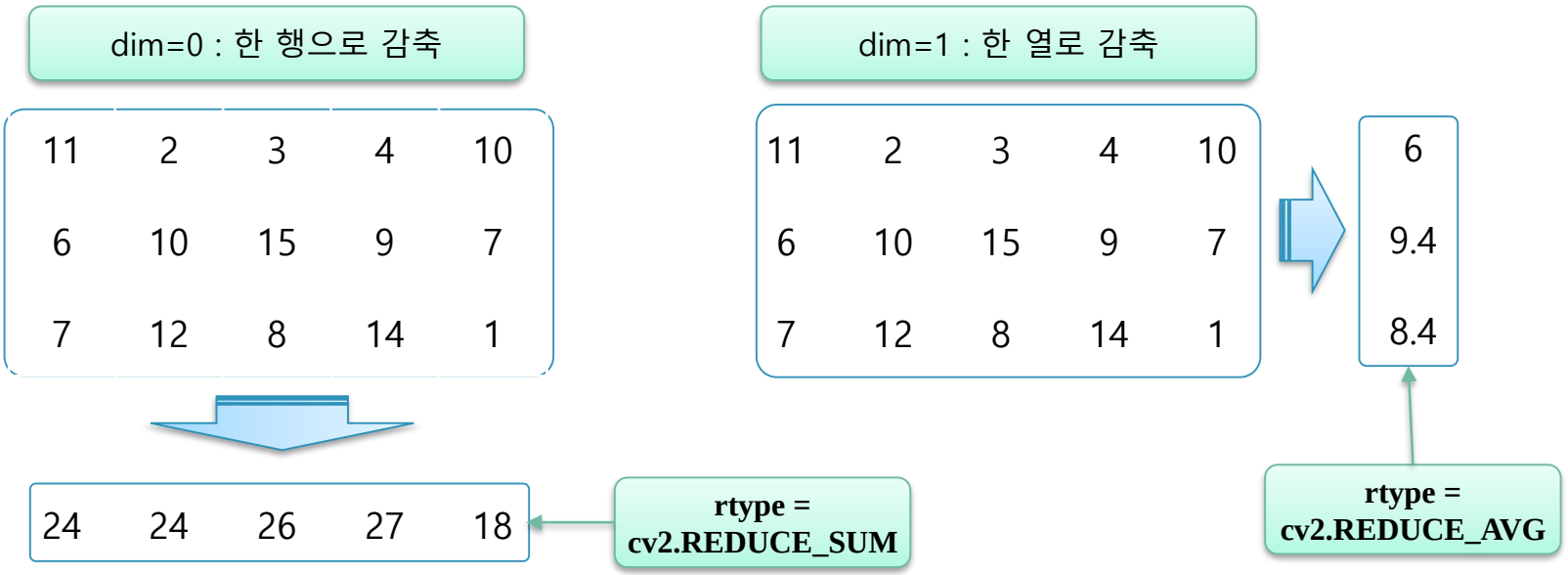
- 0 : 열 방향으로 연산하여 1행으로 축소
- 1 : 행 방향으로 연산하여 1열로 감소

■ 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

- reduce() 함수 감축 방향



통계 관련 함수

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.

- cv2.sortIdx() 함수 설명