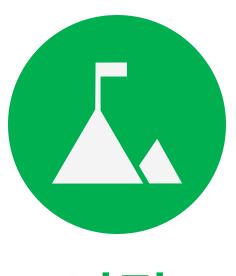
미니 프로젝트 Ver 1.0 스핀오프

디지털 영상 처리 w. Python

[Intel] 엣지 AI SW 아카데미 - 절차지향 프로그래밍

- 프로젝트의 목표
- 개발 환경
- 화면 구성 및 기능
- 부가 기능
- 마치며

프로젝트의 목표



비전

C언어 이외의 언어를 사용해 각 언어의 유사점과 차이점을 직접 체감해 보는 것

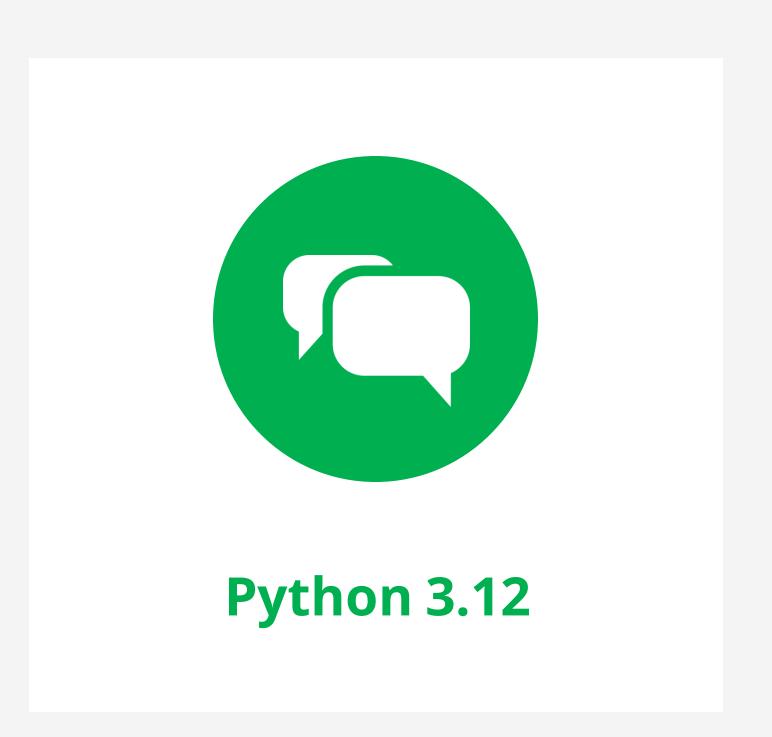


미션

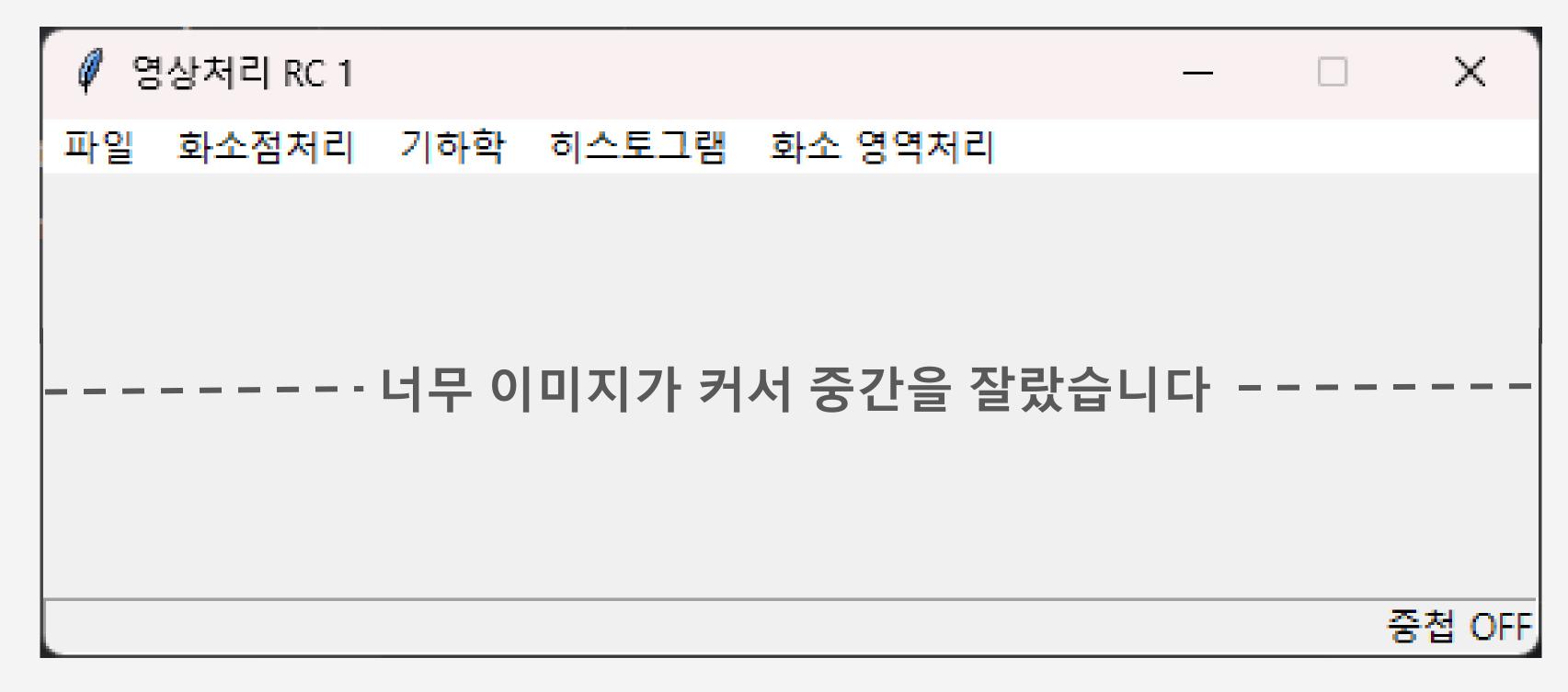
C언어로 구현했던 영상 처리 프로그램을 Python으로 구현하는 것

개발 환경





화면 구성 및 기능 초기화면

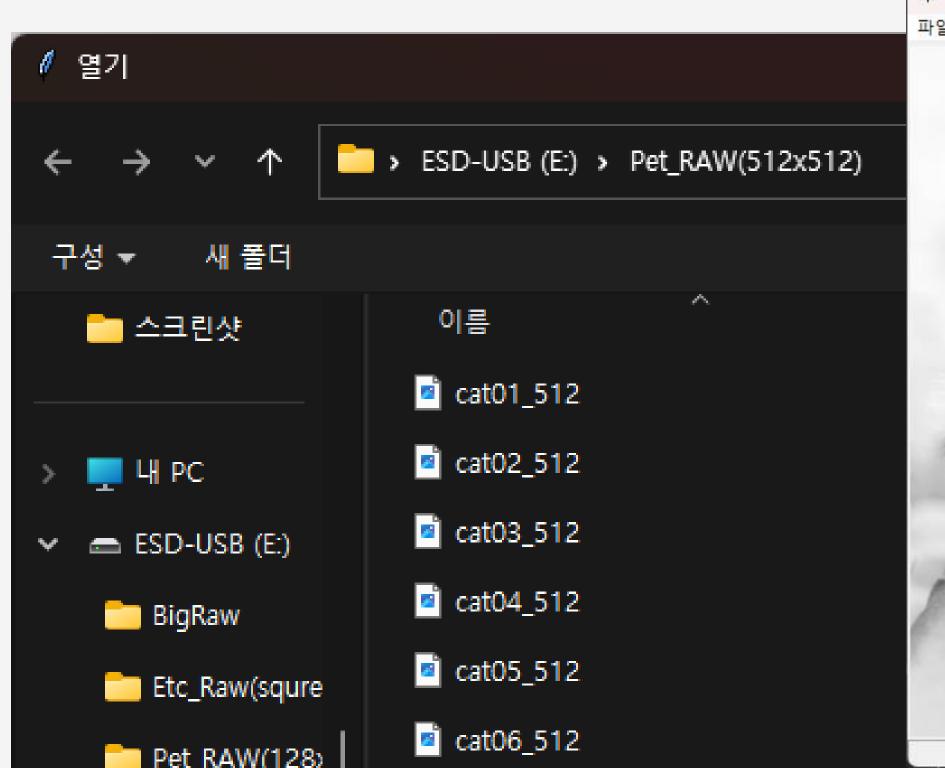


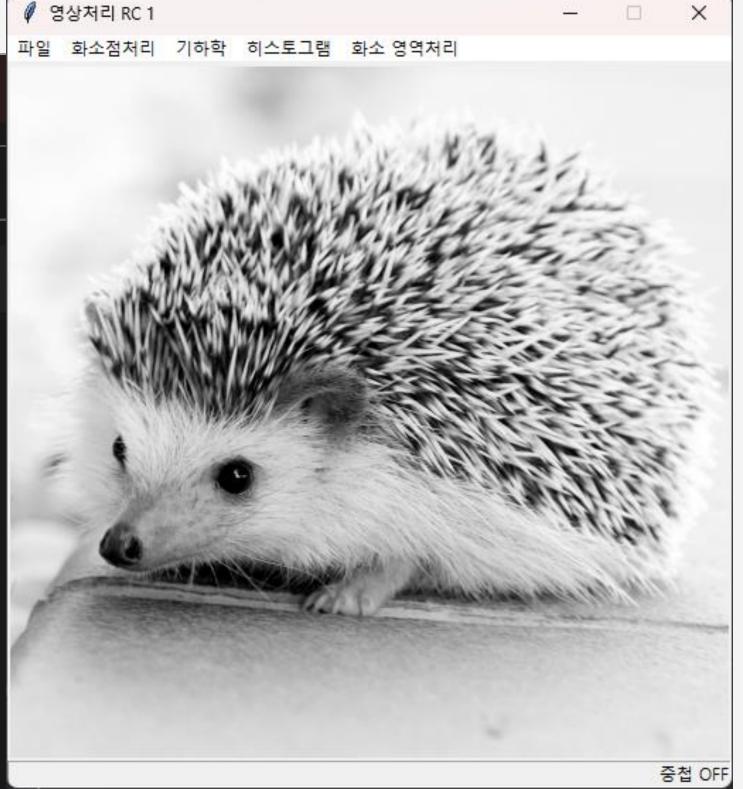
화면 구성 및 기능 파일 메뉴

			_			×
기하학	히스토그램	화소 영역처리				
	기하학	기하학 히스토그램	기하학 히스토그램 화소 영역처리			

열기 저장 중첩 종료

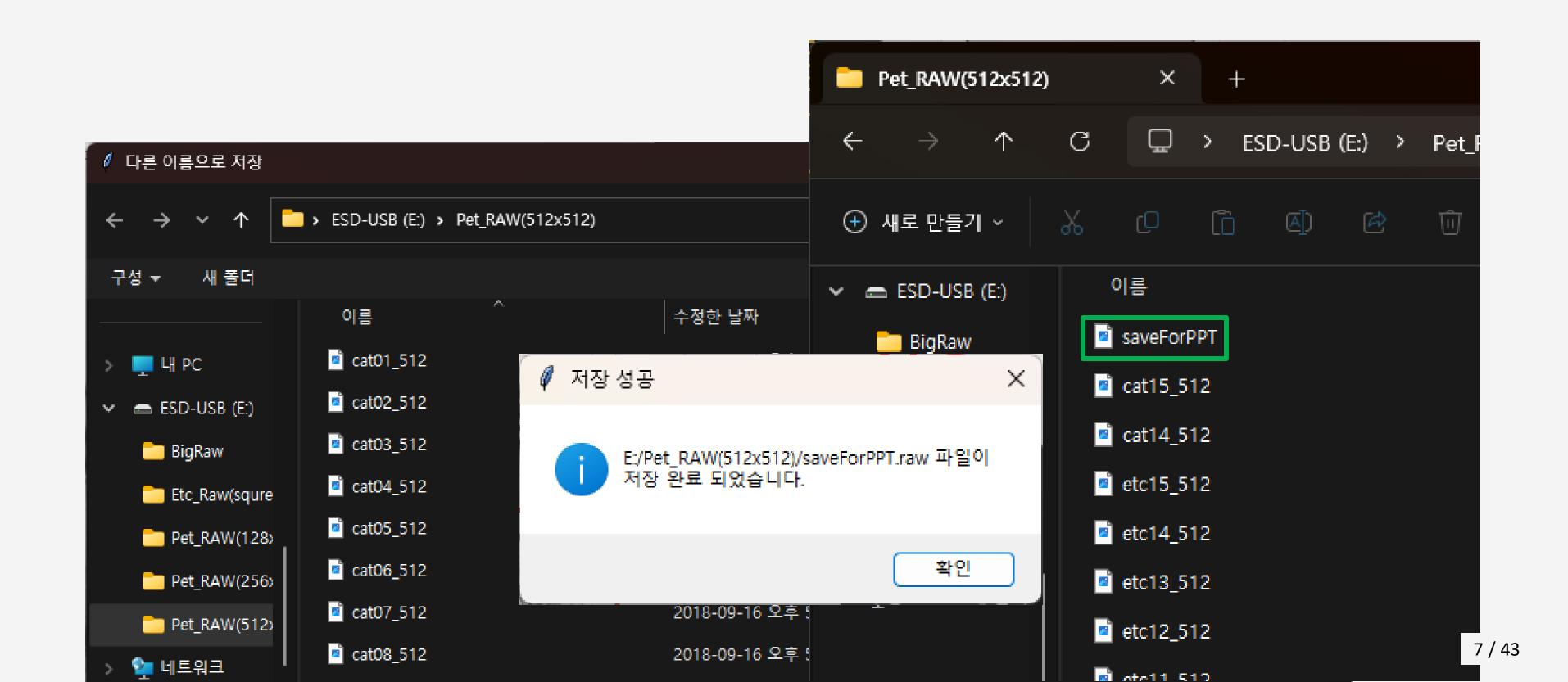
화면 구성 및 기능 열기 창으로 파일 가져오기





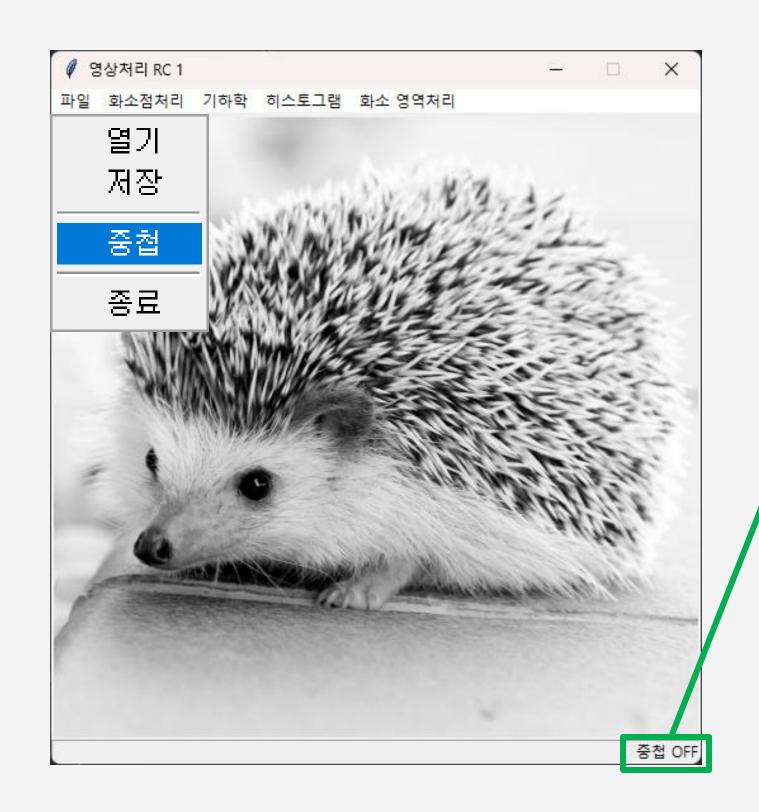
열기 저장 중첩 종료

화면 구성 및 기능 저장 창으로 파일 저장하기

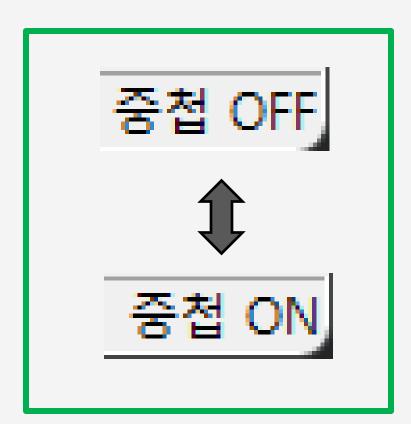


종료

화면 구성 및 기능 효과 중첩 ON / OFF 및 확인



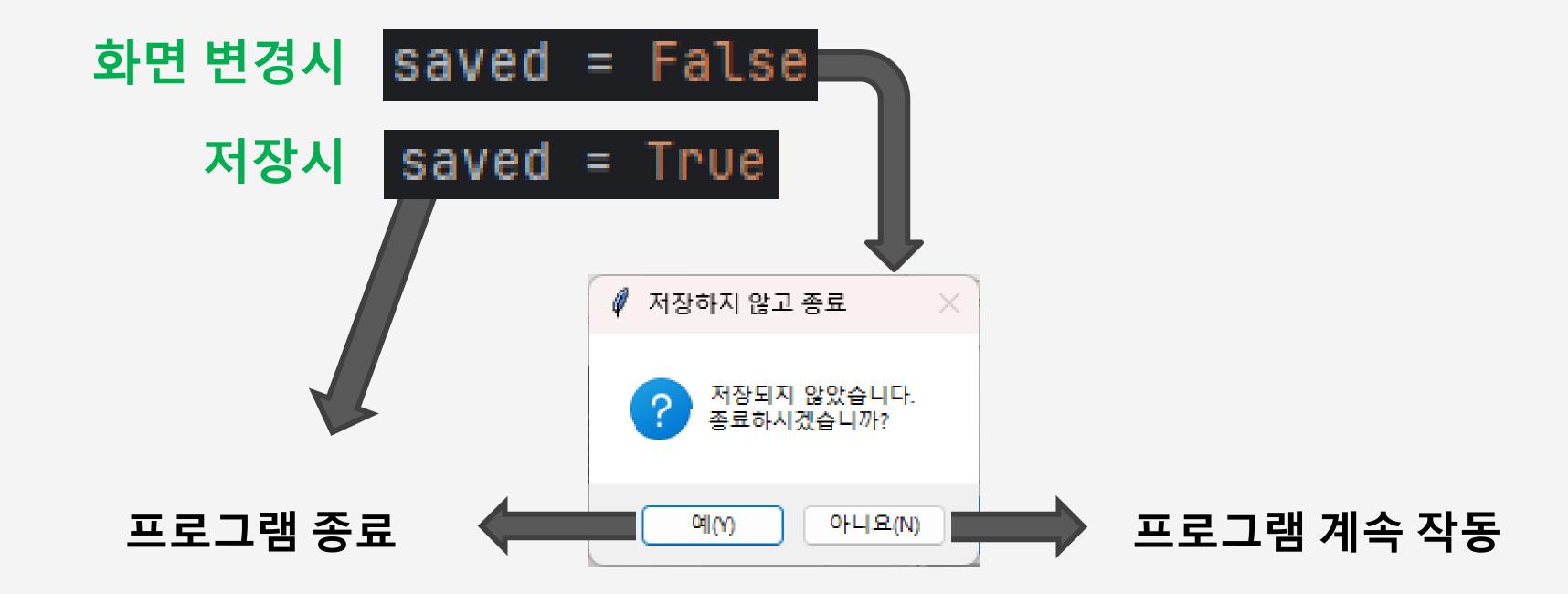
클릭 시 변경



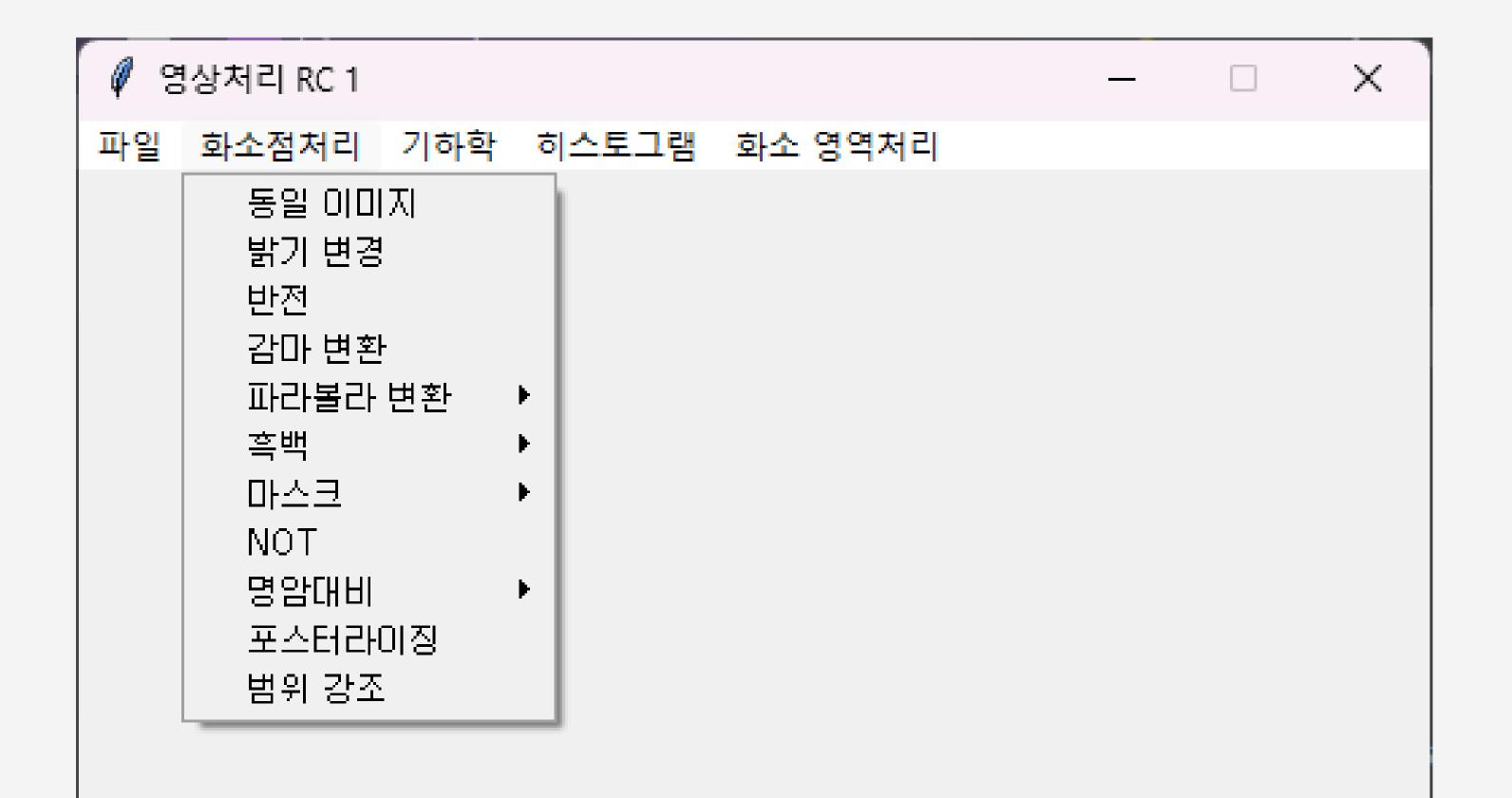
global statusbar, accumulation
 statusbar.config(text=accumul)

종료

화면 구성 및 기능 저장 여부를 확인하고 종료하기



화면 구성 및 기능 화소점처리 메뉴



화면 구성 및 기능 화소점처리 > 동일영상 알고리즘





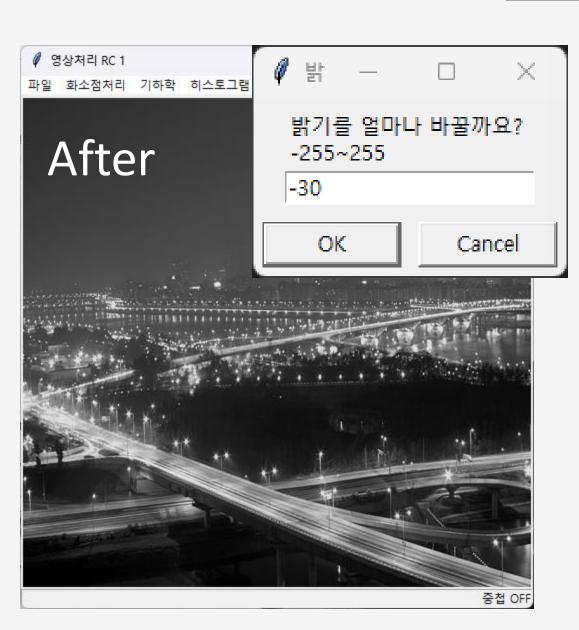
outImage[i][k] = inImage[i][k]

동일 이미지 밝기 변경 반전 감마 변환 파라볼라 변환 흑백 마스크 NOT 명암대비 포스터라이징 범위 강조

화면 구성 및 기능 화소점처리 > 밝기 변경







```
px = inImage[i][k] + scale
if (px > 255):
    px = 255
```

```
if (px < 0):
    px = 0
outImage[i][k] = px</pre>
```

동일 이미지

감마 변환

명암대비 포스터라이징 범위 강조

화면구성 및 기능 화소점처리 > 반전





outImage[i][k] = 255 - inImage[i][k]

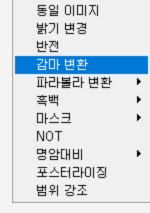
동일 이미지 밝기 변경

감마 변환 파라볼라 변환

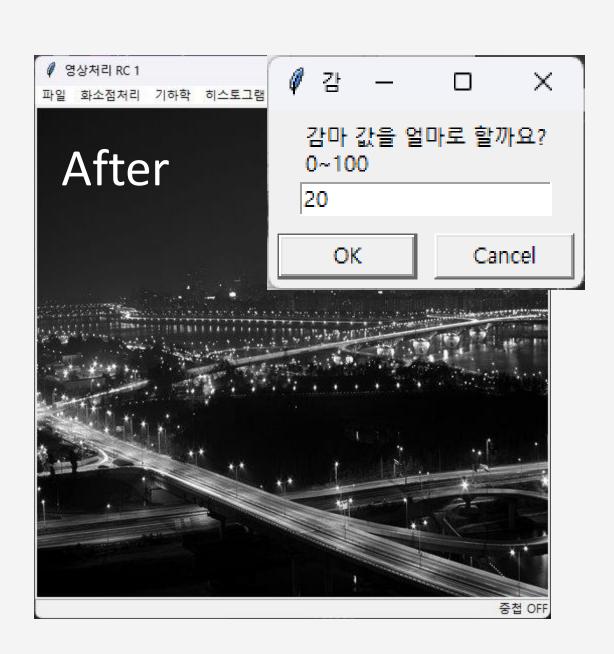
마스크

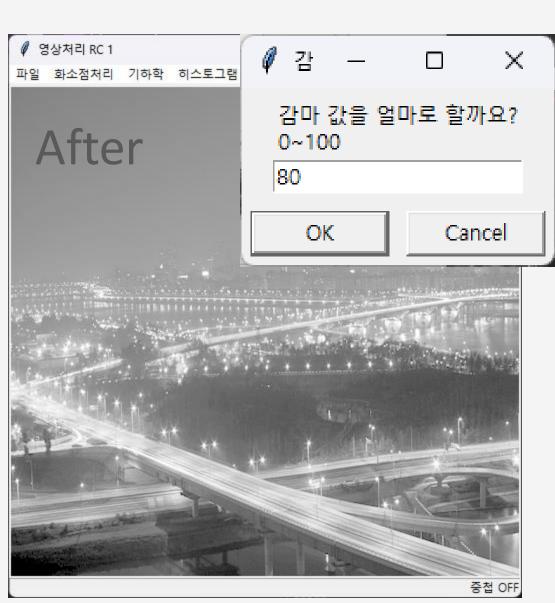
명암대비 포스터라이징 범위 강조

화면 구성 및 기능 화소점처리 > 감마값 변환









OutImage =
$$255 * \left(\begin{array}{c} 30\\ -----\\ 255 \end{array}\right)$$

입력 받은 수치를 유의미하게 관찰할 수 있는 정도

outImage[i][k] = int(255 * (inImage[i][k] / 255) ** (30 / scale))

반전 감마 변환 <mark>파라볼라</mark> 흑백 마스크 NOT 명암대비

동일 미미지

포스터라이징 범위 강조

화면 구성 및 기능 화소점처리 > 감마값 변환





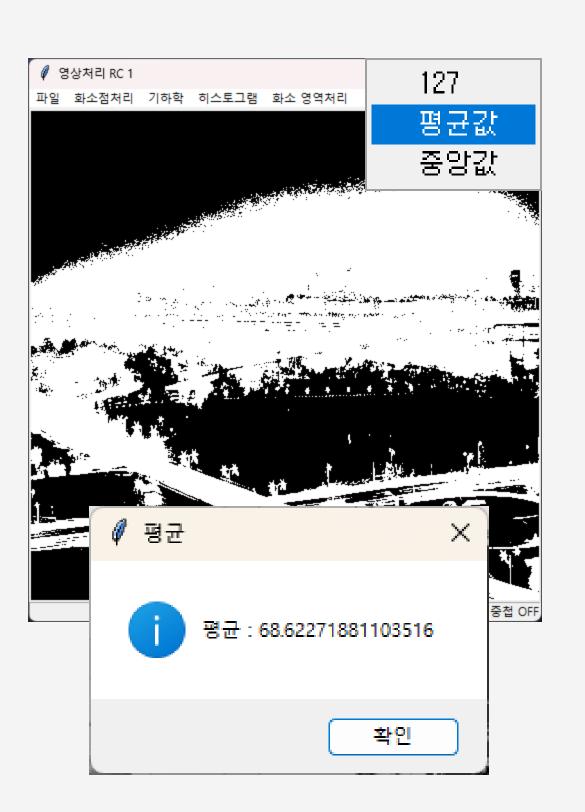


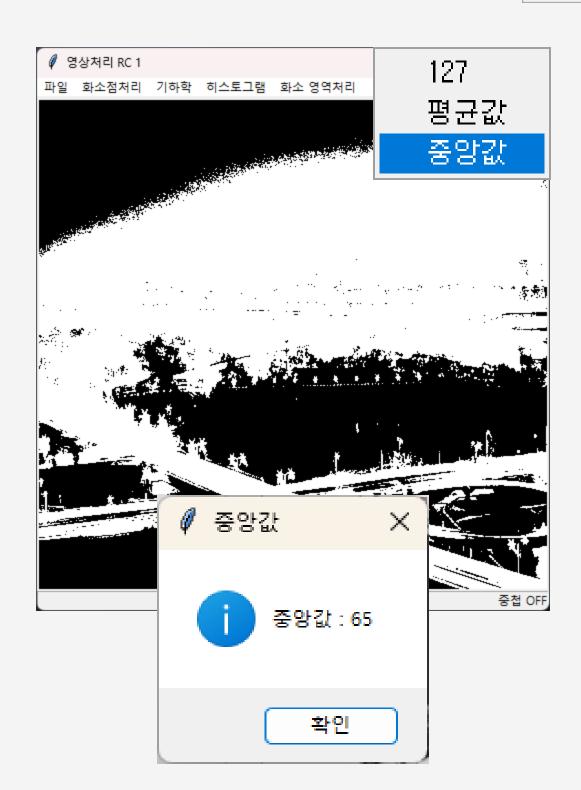
int(255. * ((inImage[i][k] / 127.) - 1) ** 2)

int(255 - 255 * ((inImage[i][k] / 127) - 1) ** 2)

화면 구성 및 기능 화소점처리 > 감마값 변환





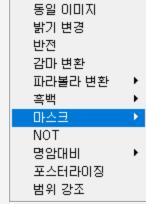


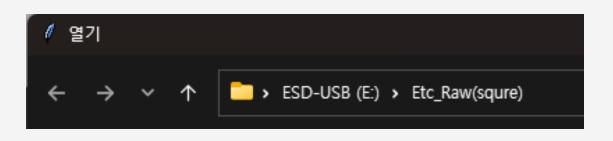
동일 이미지 밝기 변경

감마 변환 파라볼라 변환

마스크 NOT 명암대비 포스터라이징 범위 강조

화면 구성 및 기능 화소점처리 > 감마값 변환





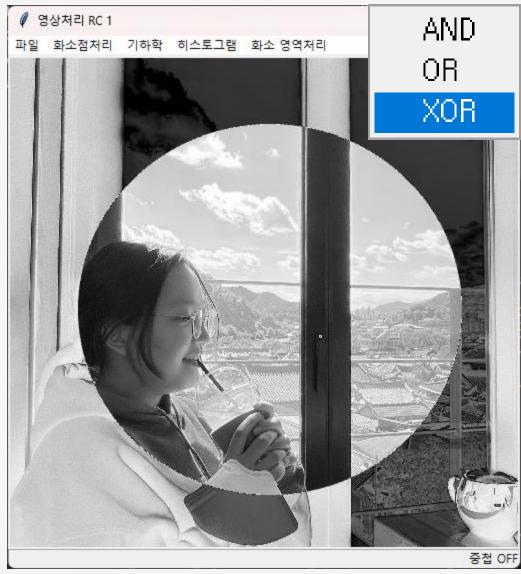
열기로 마스크 이미지 결정



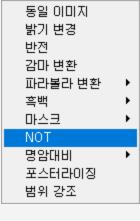








화면 구성 및 기능 화소점처리 > NOT 반전





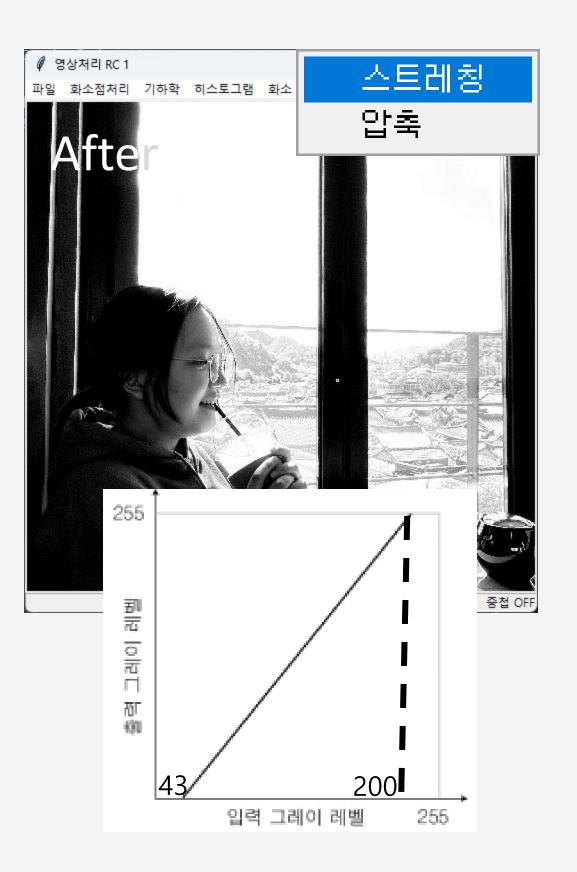


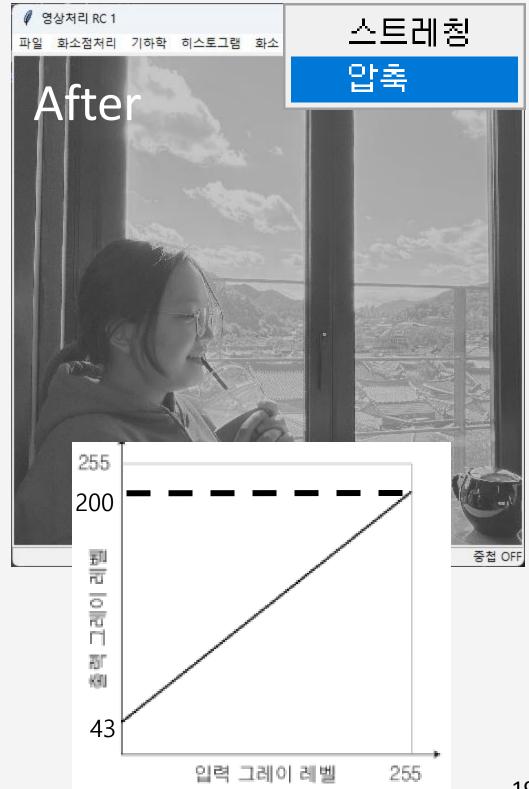
파이썬의 데이터 타입은 int로 음수로 넘어가기도 하고 255를 넘기도 하여 둘 모두를 처리하도록 함

```
val = ~inImage[i][k]
if (val < 0):
    val = abs(val % 255)
outImage[i][k] = val</pre>
```

화면 구성 및 기능 화소점처리 > 명함대비







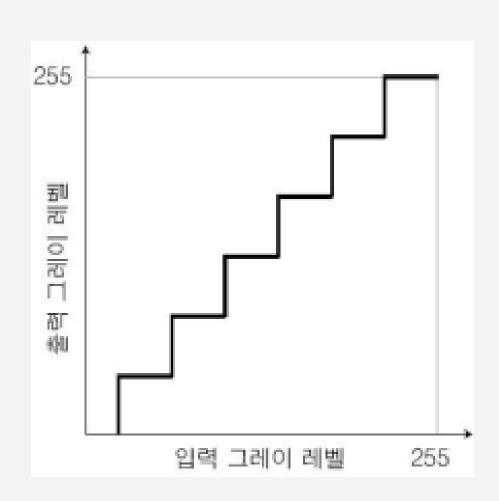
동일 이미지 밝기 변경 반전 감마 변환 파라볼라 변환 흑백 마스크 NOT 명암대비 포스터라이징 범위 강조

19 / 43

화면 구성 및 기능 화소점처리 > 포스터라이징







동일 이미지 밝기 변경

감마 변환 파라볼라 변환

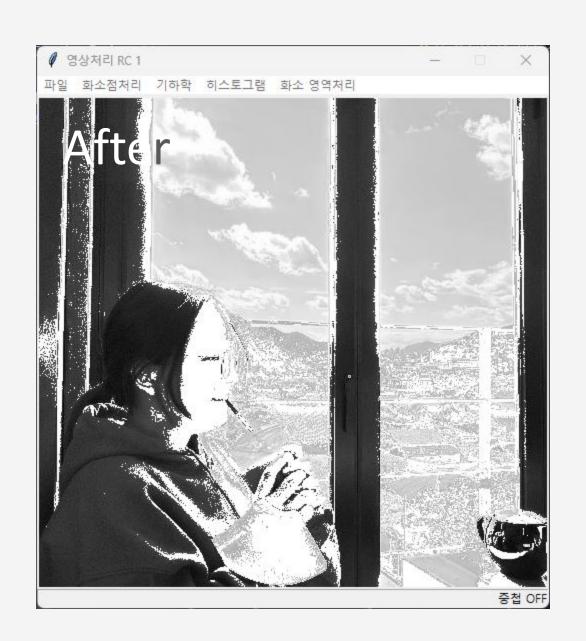
흑백

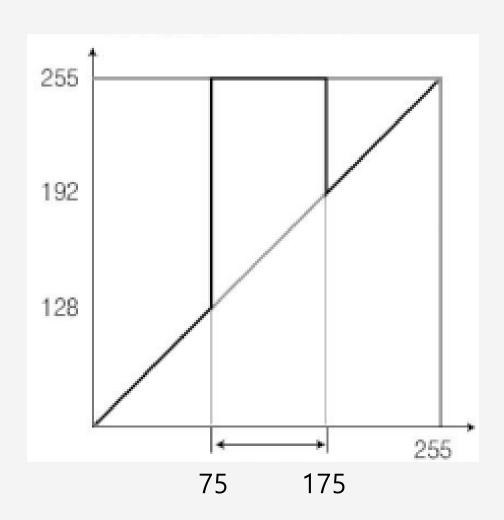
NOT 명암대비

범위 강조

화면 구성 및 기능 화소점처리 > 포스터라이징







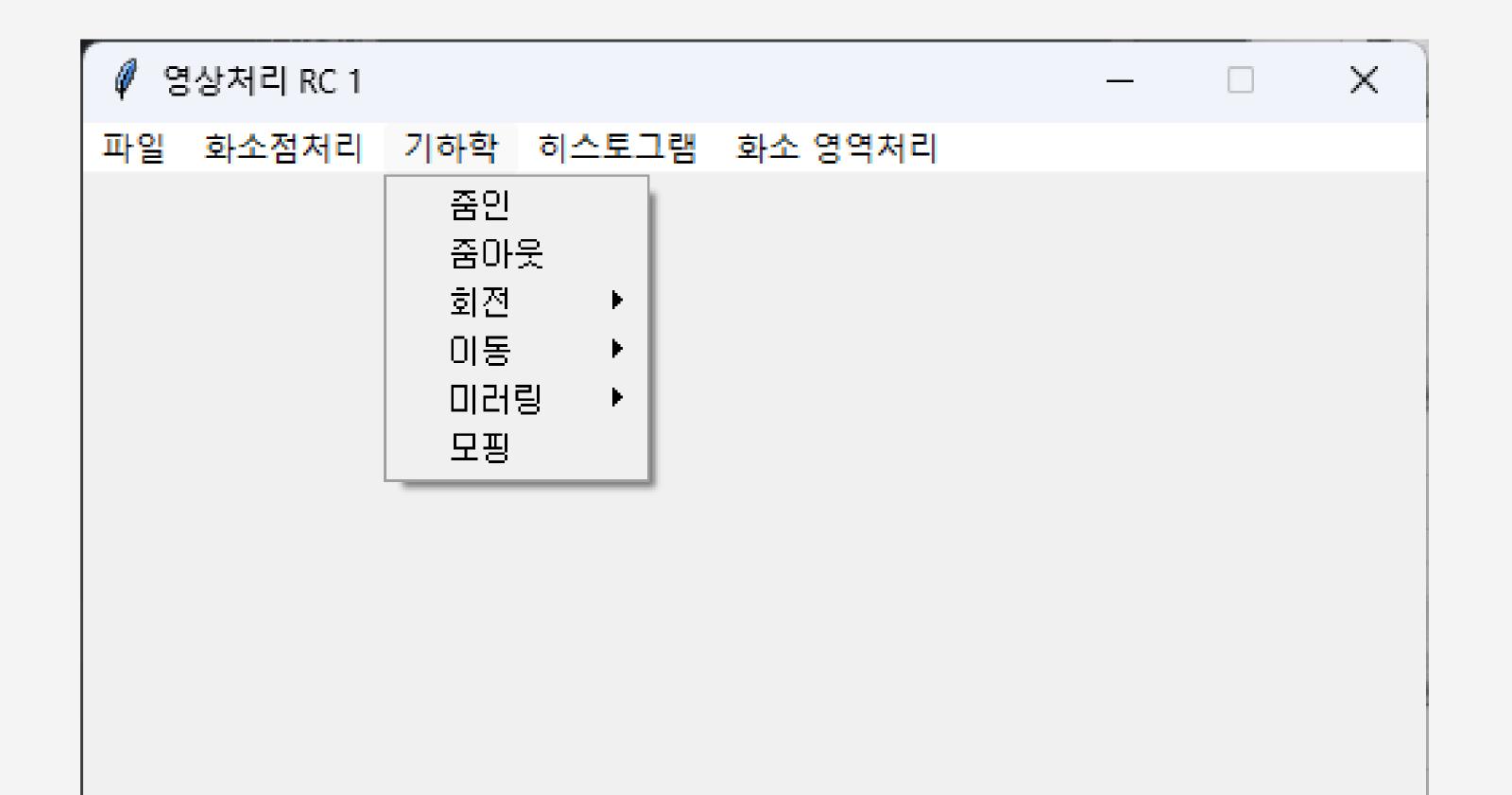
반전 감마 변환 파라볼라 변환 흑백 마스크 NOT 명암대비

동일 이미지 밝기 변경

범위 강조

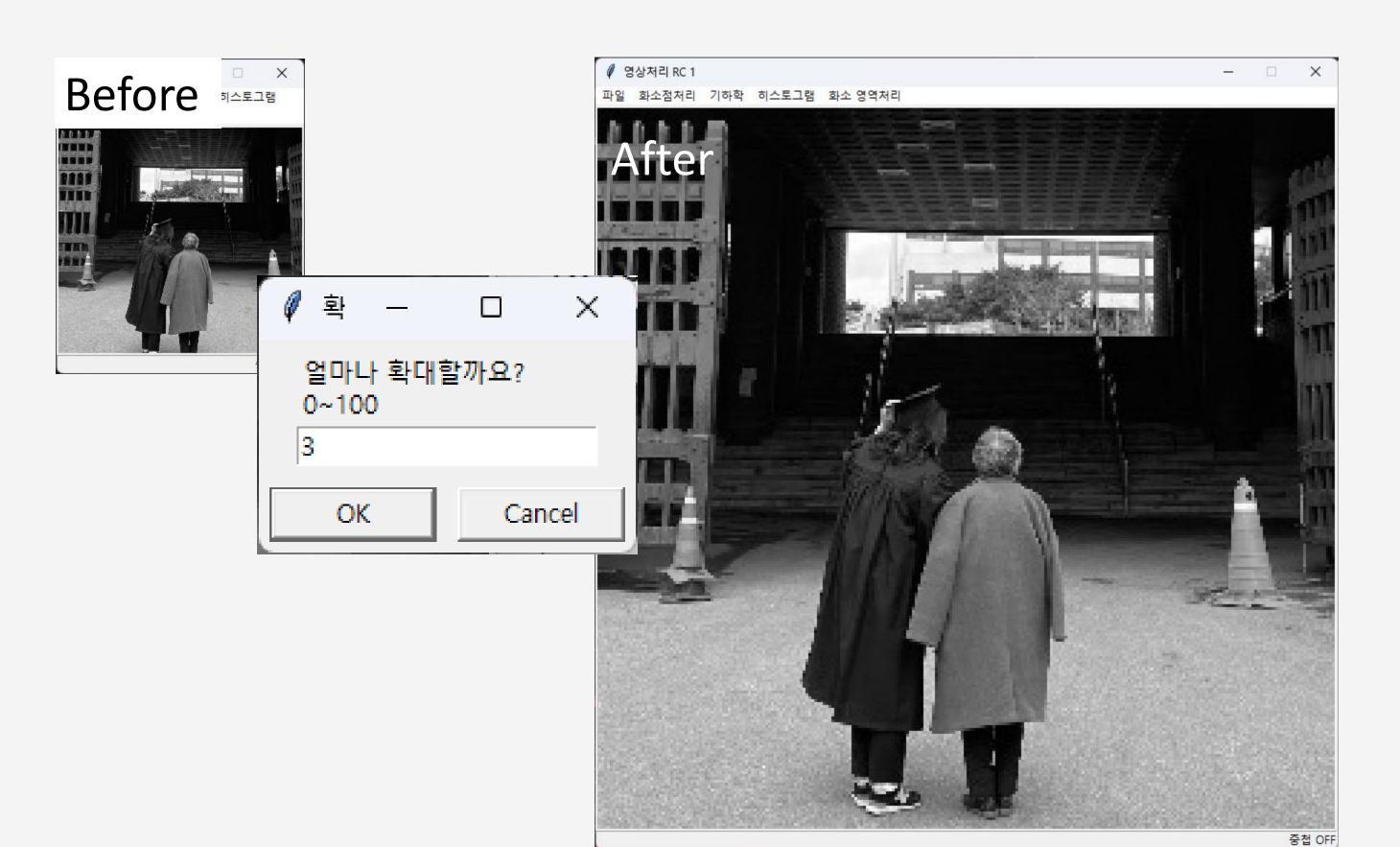
포스터라이징

화면 구성 및 기능 기하학 처리 메뉴



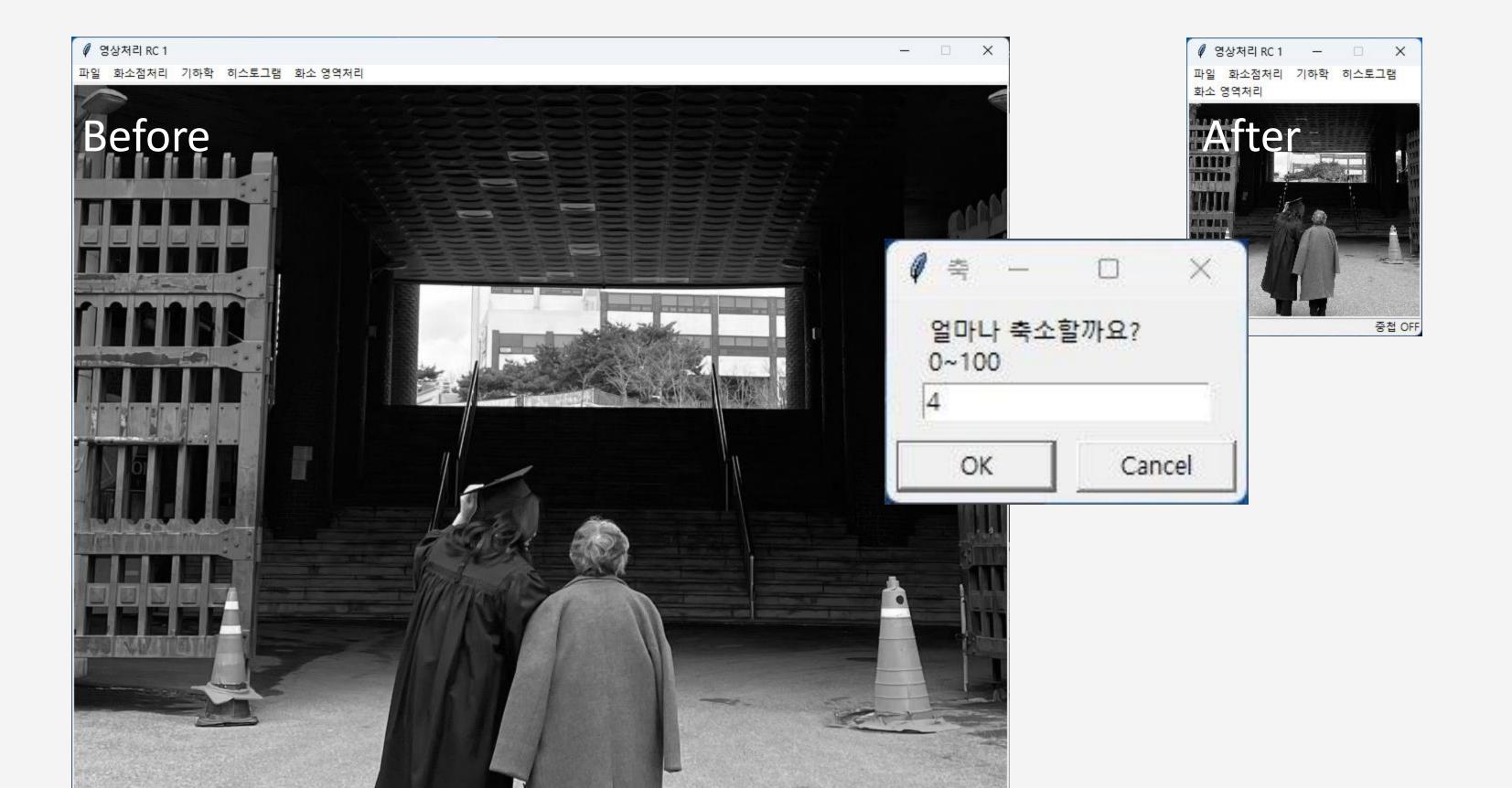
줌인 줌아웃 회전 ▶ 이동 ▶ 미러링 ▶

화면구성및기능 기하학처리 > 줌인



줌인 줌마웃 회전 • 이동 • 미러링 • 모핑

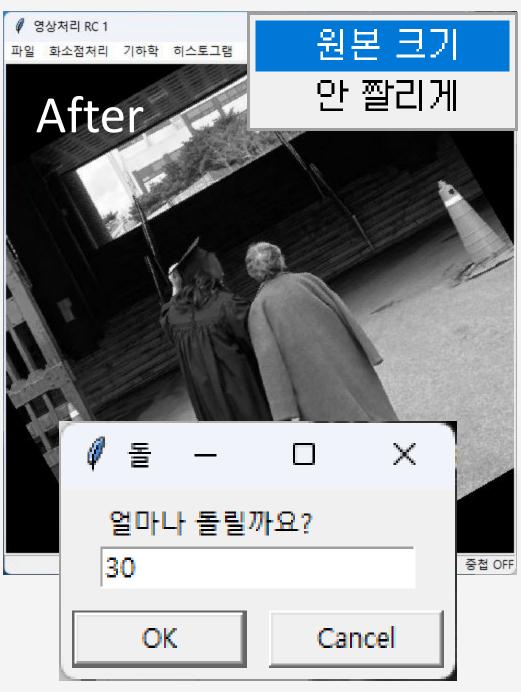
화면 구성 및 기능 기하학처리 > 줌아웃

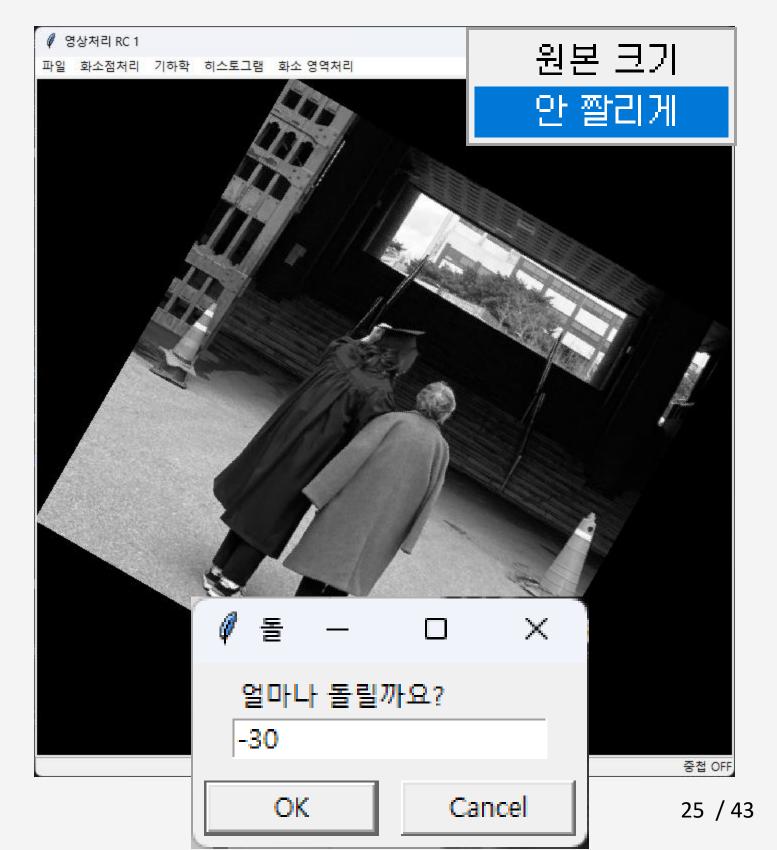


줌인 줌아웃 회전 이동 미러링 모핑

화면구성 및 기능 기하학처리 > 회전





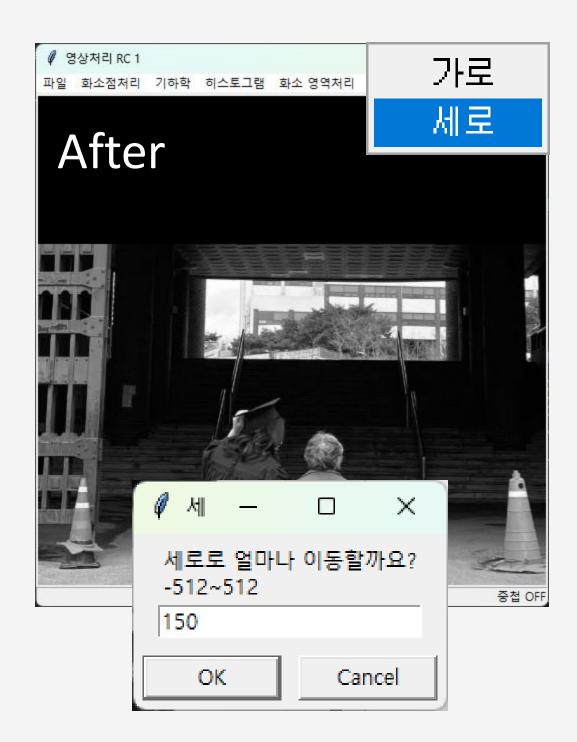


줌인 줌마웃 회전 ♪ 이동 ♪ 미러링 ♪

화면구성 및 기능 기하학처리 > 이동







화면구성 및 기능 기하학처리 > 이동



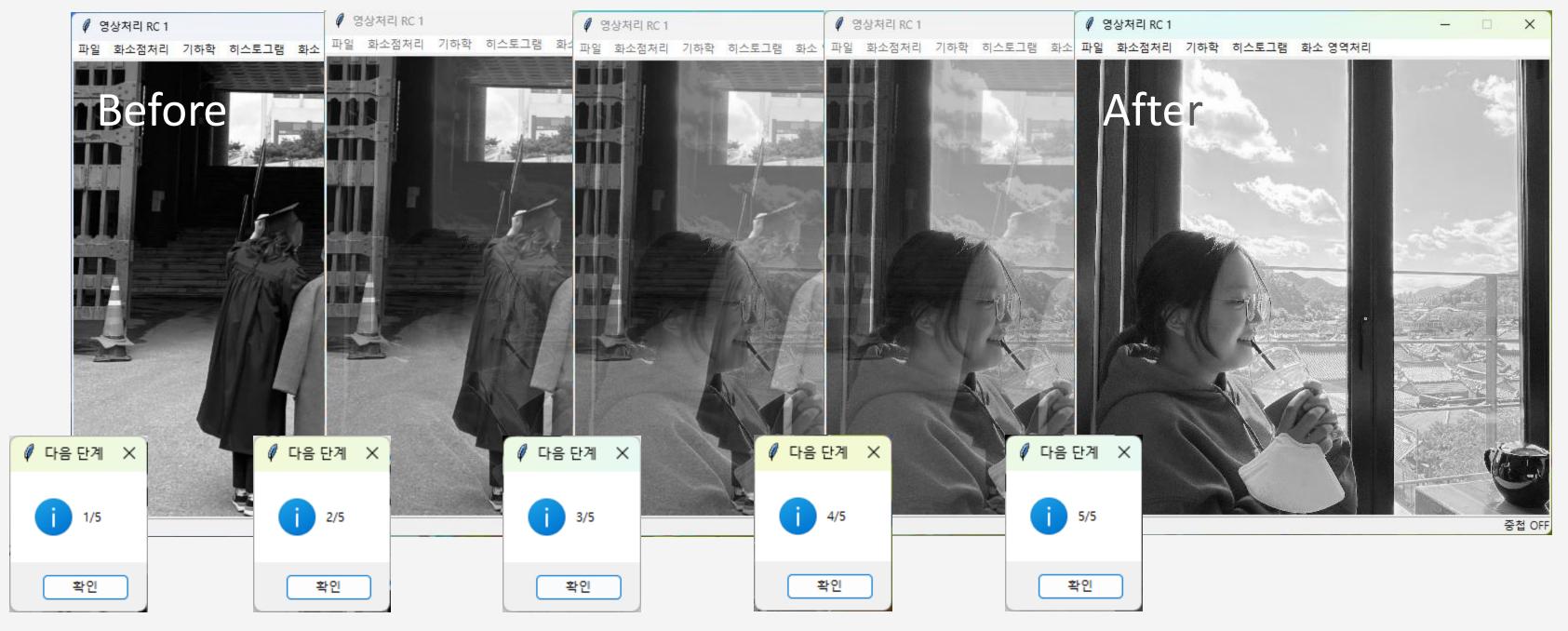




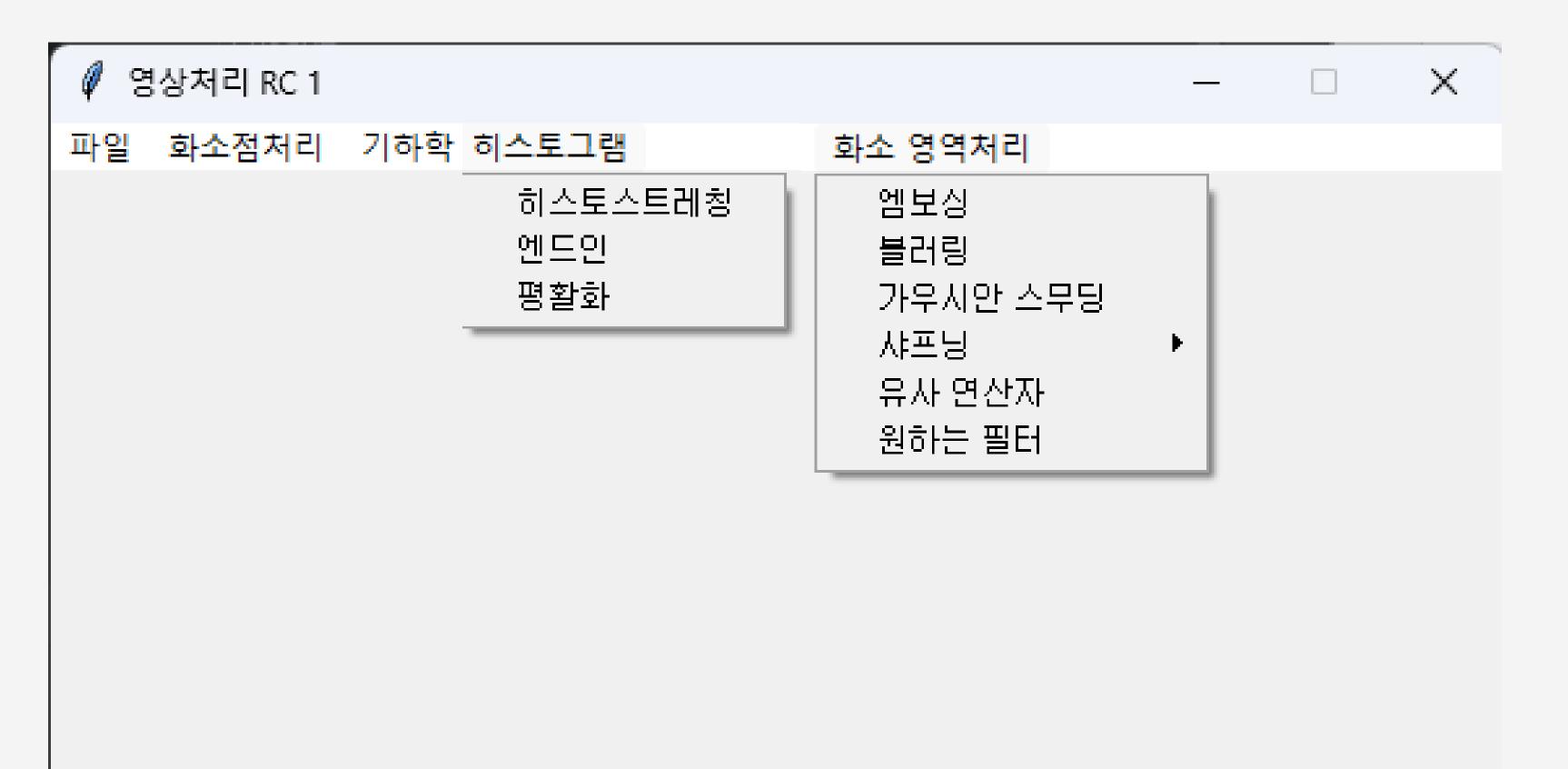
줌인 줌아웃 회전 ▶ 이동 ▶ 미러링 ▶

화면 구성 및 기능 기하학처리 > 모핑

∅ 모 −		
몇 단계가 필 0~100	요한가요?	
5		
OK	Cancel	



화면 구성 및 기능 히스토그램 & 화소 영역 처리 메뉴



화면 구성 및 기능 히스토그램 > 히스토스트레칭





해당 사진의 픽셀 중 최대 밝기와 최저 밝기를 찾음

```
if (inImage[i][k] < low):
    low = inImage[i][k]
if (inImage[i][k] > high):
    high = inImage[i][k]
```

최대 밝기와 최저 밝기를 기반으로 기존 픽셀 값을 갱신

```
(high - low) * 255.0)
if (new > 255):
   new = 255
if (new < 0):
   new = 0
outImage[i][k] = new</pre>
```

화면 구성 및 기능 히스토그램 > 엔드인





해당 사진의 픽셀 중 최대 밝기와 최저 밝기를 찾음

> 최대 밝기와 최저 밝기를 상황에 맞게 변경

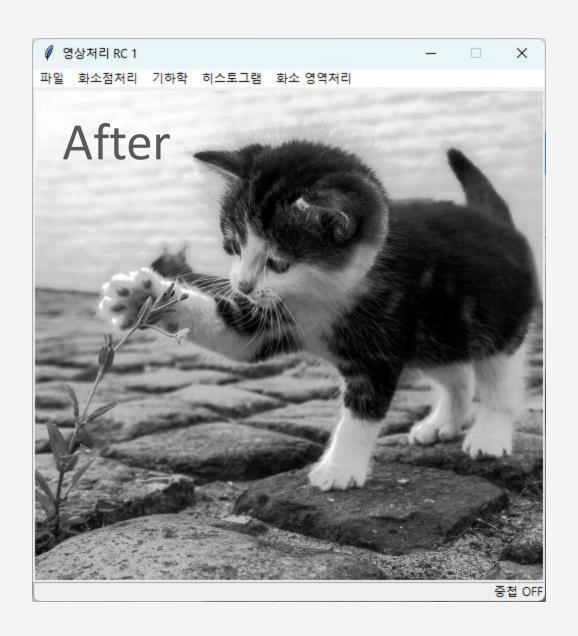
> > high -= 50 low += 50

구조적으로 히스토스트레칭과 거의 동일

최대 밝기와 최저 밝기를 기반으로 기존 픽셀 값을 갱신

화면 구성 및 기능 히스토그램 > 평활화





각 밝기의 빈도를 일정하게 만드는 처리

> 밝은 사진은 전체적으로 어두워짐

어두운 사진은 전체적으로 밝아짐

히스토그램 생성

각 픽셀값의 빈도를 셈



누적 히스토그램 생성 히스토그램 누적합 배열 생성

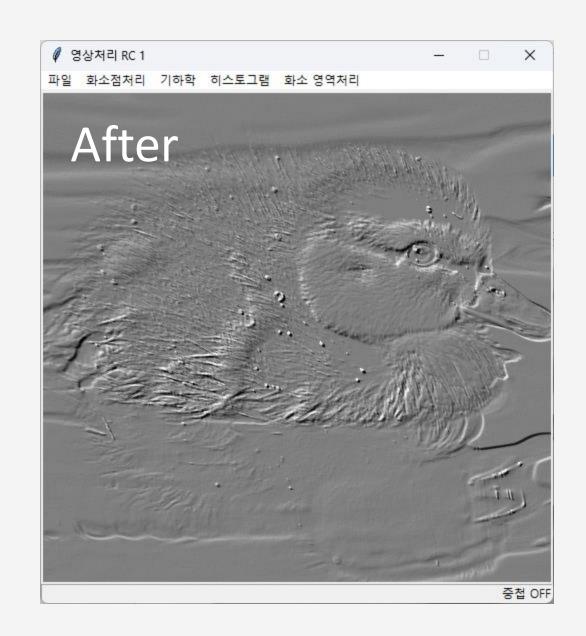


히스토그램 정규화

누적합 배열을 평균냄

화면 구성 및 기능 화소영역처리 > 엠보싱





사진에 일부만 튀어나오게 하는 마스크를 씌움

임시배열의 (1,1)부터 출력배열에 복사



필터의 합이 0이면 임시배열에서 127을 더함



임시 저장 배열 생성

사진 크기보다 높이와 너비가 (필터크기-1)만큼 크게 생성



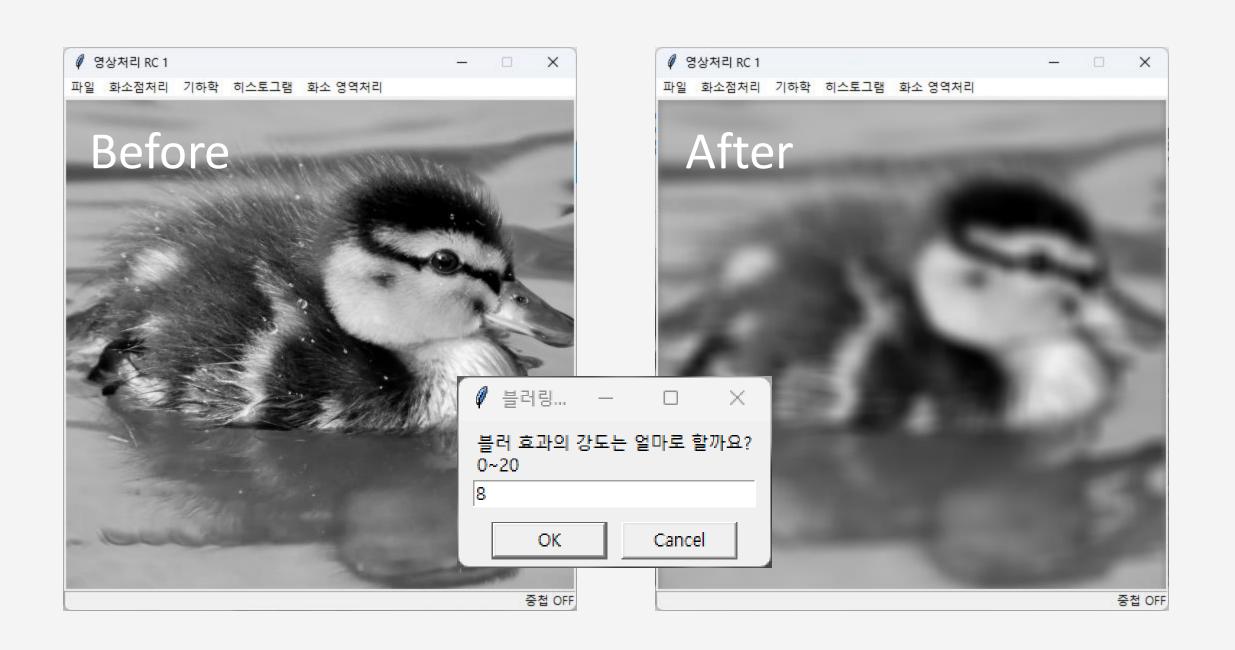
127으로 초기화 입력 사진 복사 (1, 1)부터 시작



임시배열의 (1,1)부터 마스크를 씌움

마스크와 겹쳐진 곳끼리는 곱하고 곱해진 값의 평균으로 임시배열을 변경

화면 구성 및 기능 화소영역처리 > 블러링

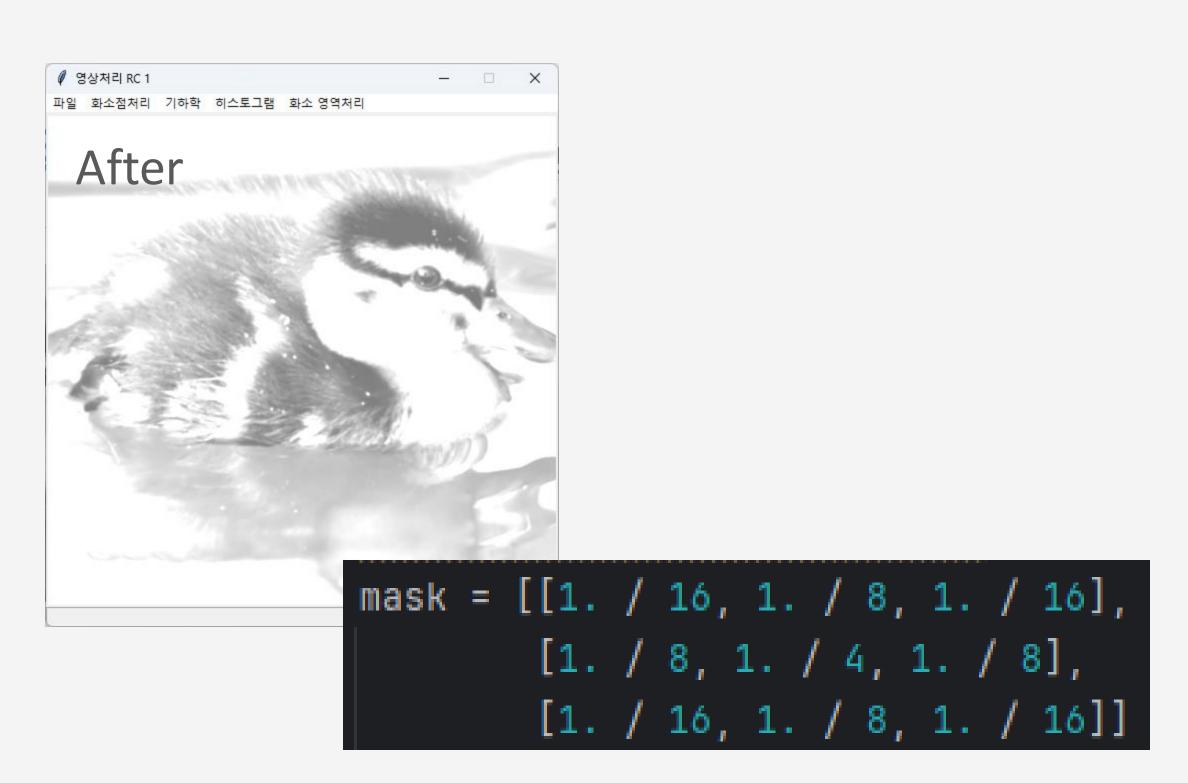


블러 강도를 입력받지만 내부적인 처리로는 필터의 크기는 (입력값) * 2 + 1

또한 필터의 내부 값은 1 / (입력값) ** 2으로 채움

화면 구성 및 기능 화소영역처리 > 가우시안 스무딩



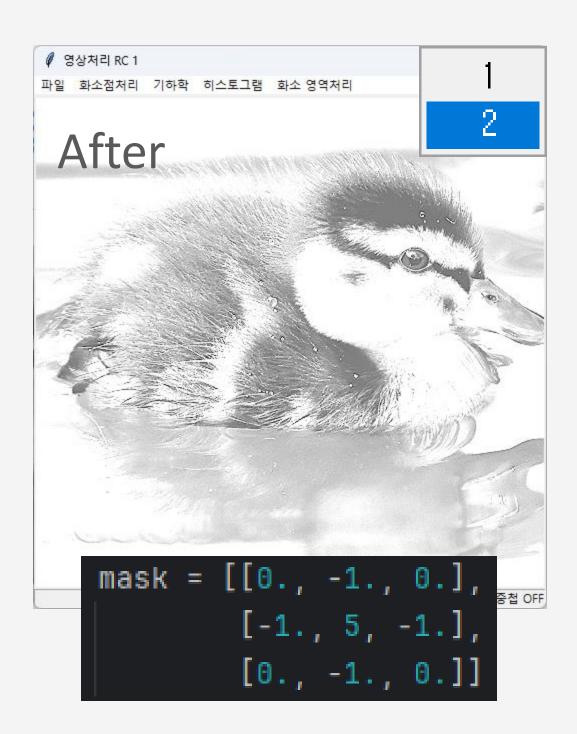


엠보싱 블러링 가우시안 스무딩 <mark>샤프닝 ▶</mark> 유사 연산자 원하는 필터

화면 구성 및 기능 화소영역처리 > 샤프닝

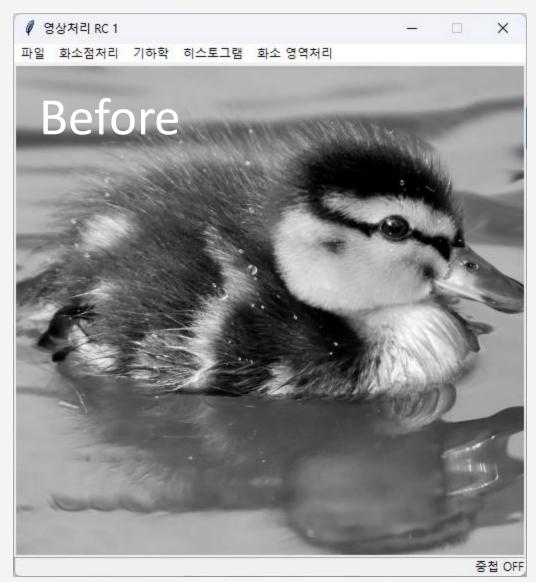






엠보싱 블러링 가우시안 스무딩 샤프닝 ► <mark>유사 연산자</mark> 원하는 필터

화면 구성 및 기능 화소영역처리 > 유사 연산자





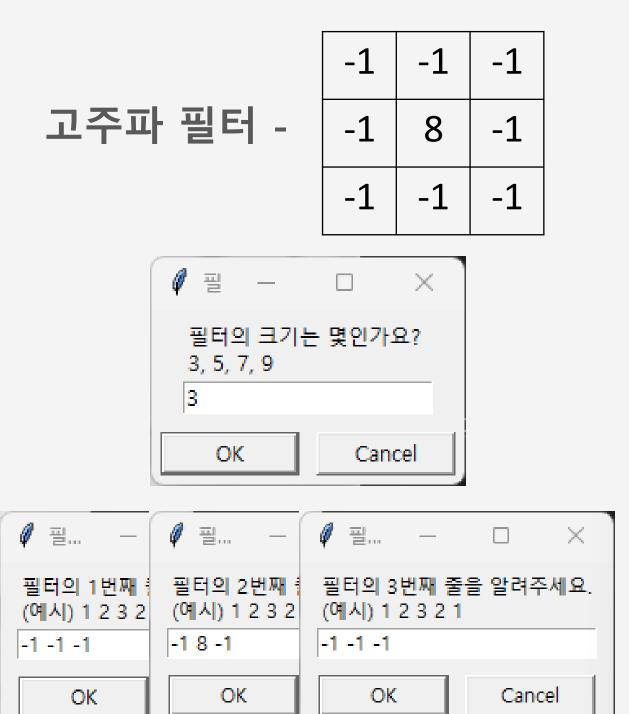
3x3으로 픽셀 주변의 값과 자신의 차이 중 가장 큰 값의 절댓값을 출력 이미지의 값으로 하는 연산

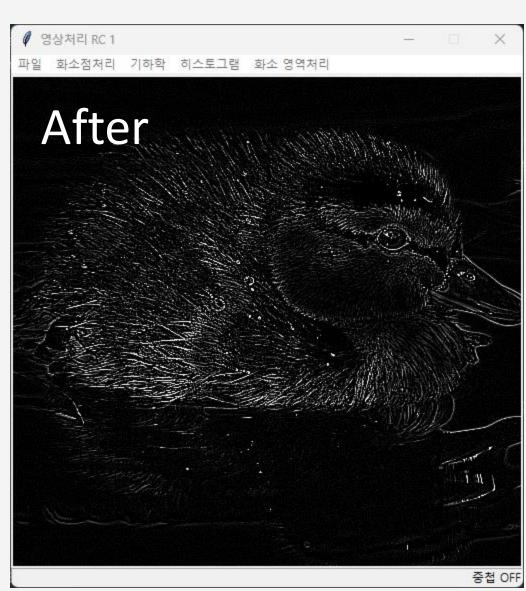
```
S = abs(tmpInImage[i + 1][k + 1] - tmpInImage[i][k])
if ((m != 1) & (n != 1)):
    S = max(abs(tmpInImage[i + 1][k + 1] - tmpInImage[i + m][k + n]), S)
    tmpOutImage[i][k] = S
```

엠보싱 블러링 가우시안 스무딩 샤프닝 ▶ 유사 연산자 원하는 필터

화면 구성 및 기능 화소영역처리 > 원하는 필터

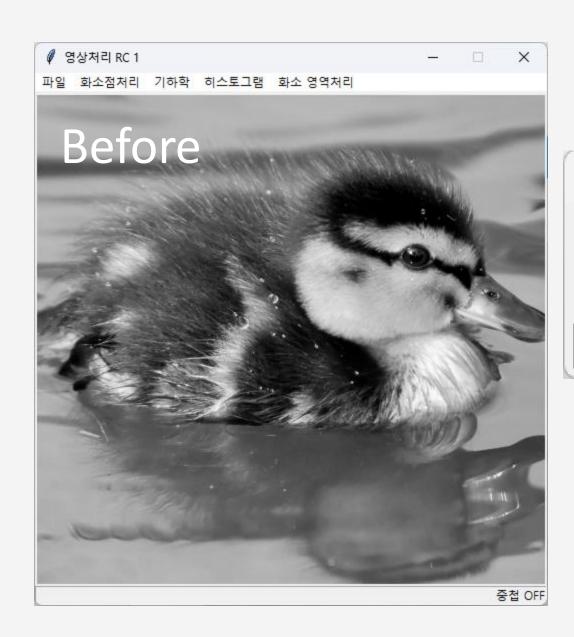






엠보싱 블러링 가우시안 스무딩 샤프닝 유사 연산자 원하는 필터

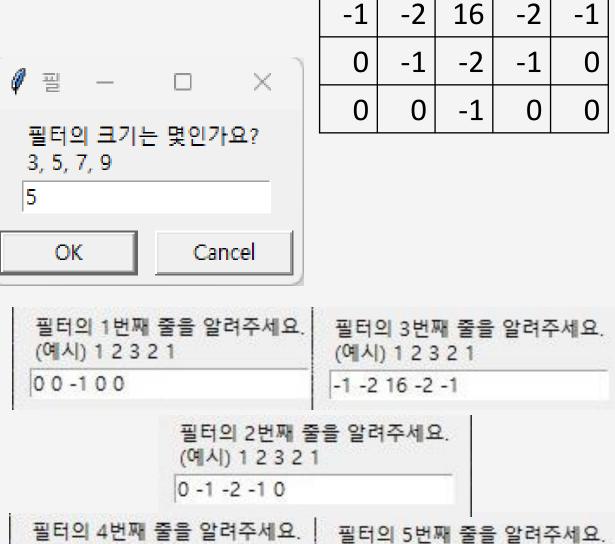
화면 구성 및 기능 화소영역처리 > 원하는 필터



LoG 필터 -

(예시) 12321

0 -1 -2 -1 0



(예시) 12321

00-100



엠보싱 블러링 가우시안 스무딩 샤프닝 ► 유사 연산자 원하는 필터

화면 구성 및 기능 화소영역처리 > 원하는 필터

1. 필터의 크기를 입력 받아 저장

scale = askinteger(title: "필터 크기"

2. 필터의 한 줄을 입력 받음

OK

Cancel

3. 입력 받은 한 줄의 값을 띄어쓰기를 기준으로 나누어 문자가 아닐 경우에만 배열에 저장

```
tmpList = tmp.split(" ")
for i in range(scale):
   if (i < len(tmpList) and not (tmpList[i].isalpha())):
      tmpMask.append(int(tmpList[i]))
   else:
      tmpMask.append(0)</pre>
```

0	0	-1	0	0

화면 구성 및 기능 화소영역처리 > 원하는 필터

4. 2번과 3번을 필터 크기만큼 반복

0	0	-1	0	0
0	-1	-2	-1	0
-1	-2	16	-2	-1
0	-1	-2	-1	0
0	0	-1	0	0

5. 필터의 합을 확인하고0이 아닐 경우,1이 되도록 연산

6. 입력 이미지에 필터를 씌움

```
for i in range(scale):
    for k in range(scale):
        sum += mask[i][k]

if (sum != 0):
    for i in range(scale):
        for k in range(scale):
        mask[i][k] /= sum
```

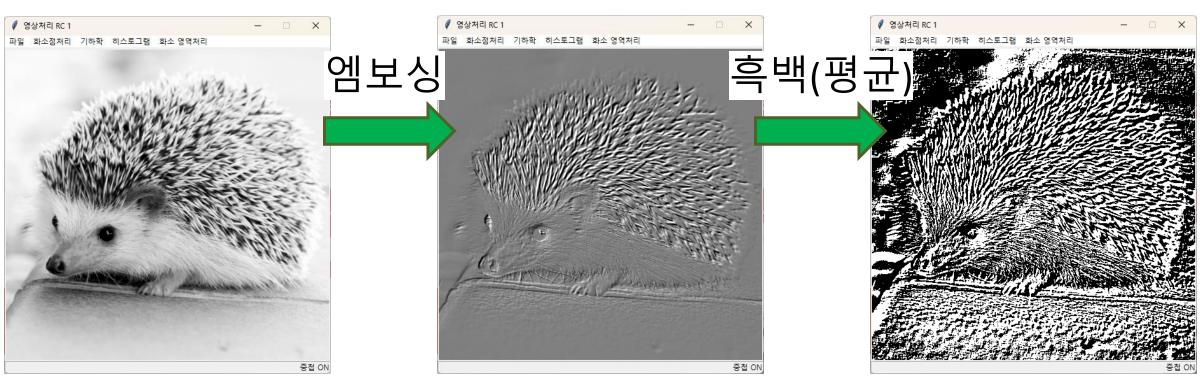
7. 필터의 합이 0이었다면 각 픽셀에 127

```
if (sum != 0):
    for i in range(outH):
        for k in range(outW):
        tmpOutImage[i][k] += 127.0
```

부가 기능



효과 중첩 전역 변수를 통해 효과 중첩을 관리합니다.



마치며

좋았던 점

프로젝트에서 파이썬을 사용해본 점

아쉬운 점

기간이 비교적 짧아, None 입력에 예외 처리를 못한 점

추후 계획

되돌리기 기능을 어떻게 추가할지 생각 해보자

장혜원



M hw11435@naver.com