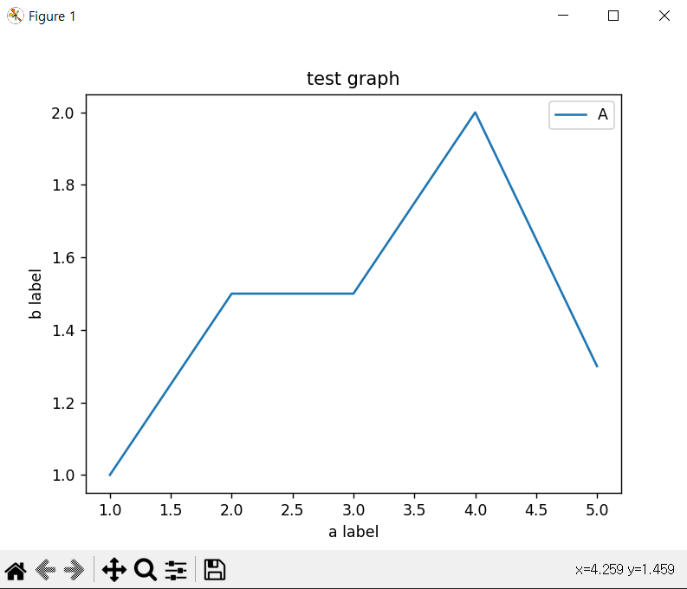
디지털 영상처리 연구실 연구보고서

김우헌

##matplotlib

-> 파이썬 표준 시각화도구

import matplotlib.pyplot as plt

plt.plot([1,2,3,4,5],[1,1.5,1.5,2,1.3])

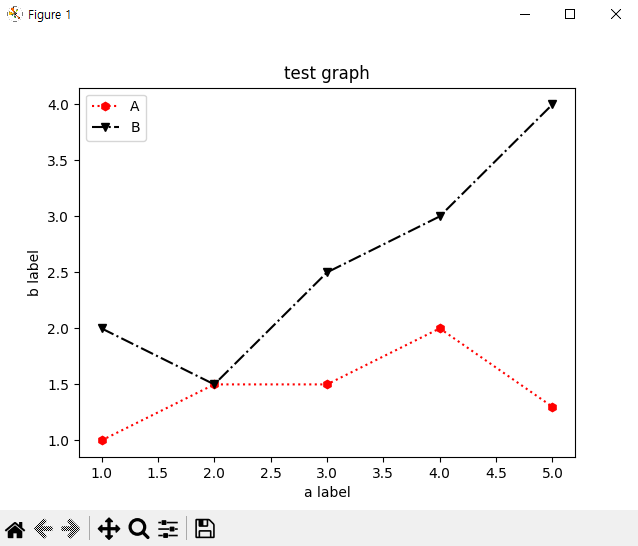
plt.xlabel("a label")

plt.ylabel("b label")

plt.title("test graph")

plt.legend(["A"])

plt.show()

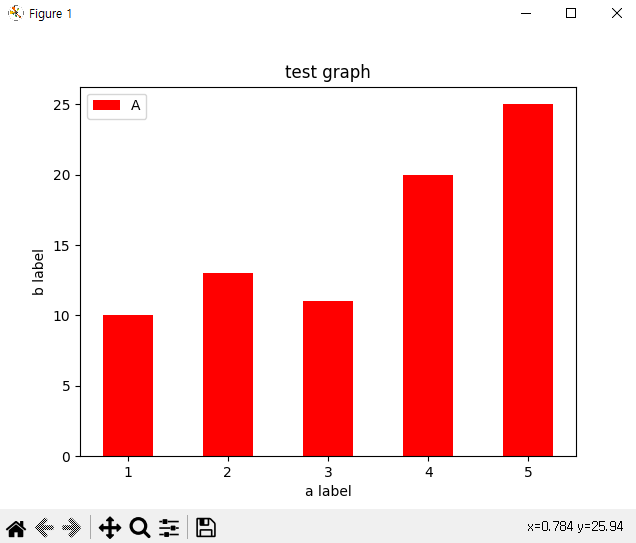


plt.plot([1,2,3,4,5],[1,1.5,1.5,2,1.3],"h:r")

plt.plot([1,2,3,4,5],[2,1.5,2.5,3,4],"k-.v")

->fmt인자 [color][line][marker]

값을 변경하여 여러가지 스타일로 설정가능



plt.bar([1, 2, 3, 4, 5], [10, 13, 11, 20, 25],

color="red", width=0.5, antialiased=True, fill=True)

-> plt.bar()를 통한 막대그래프 시각화

fig1 = plt.figure(num=1, figsize=(3,4))

plt.imshow(image)

plt.title('figure1- original(bgr)')

plt.axis('off')

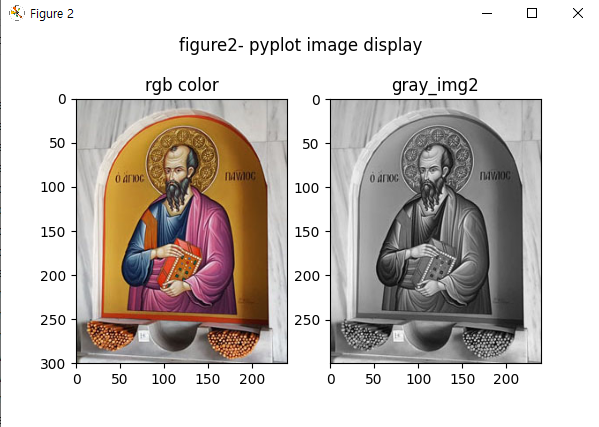
plt.tight\_layout()

->opencv에서 image load시 bgr 순서로 저장

->원본이미지의 보색을 가진 이미지 디스플레잉

fig2 = plt.figure(num=2, figsize=(6,4))

plt.suptitle('figure2- pyplot image display')

plt.subplot(1,2,1)

plt.imshow(rgb\_img)

plt.axis([0,cols, rows,0])

plt.title('rgb color')

plt.subplot(1,2,2)

plt.imshow(gray\_img)

plt.title('gray\_img2')

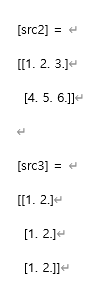
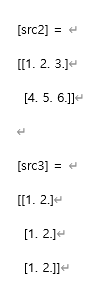
plt.show()

-> plt.subplot(행,열,순번) 으로 서브플롯 추가하는 함수

##행렬 연산함수

cv2.gemm(src1, src2, alpha, src3, beta[, dst[, flags]]) -> dst

-> dst = alpha \* src1^T \* src2 + beta\*src3^T



[src1] =

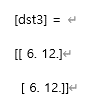
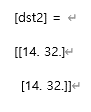
[[1. 2. 3.]

[1. 2. 3.]]

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)

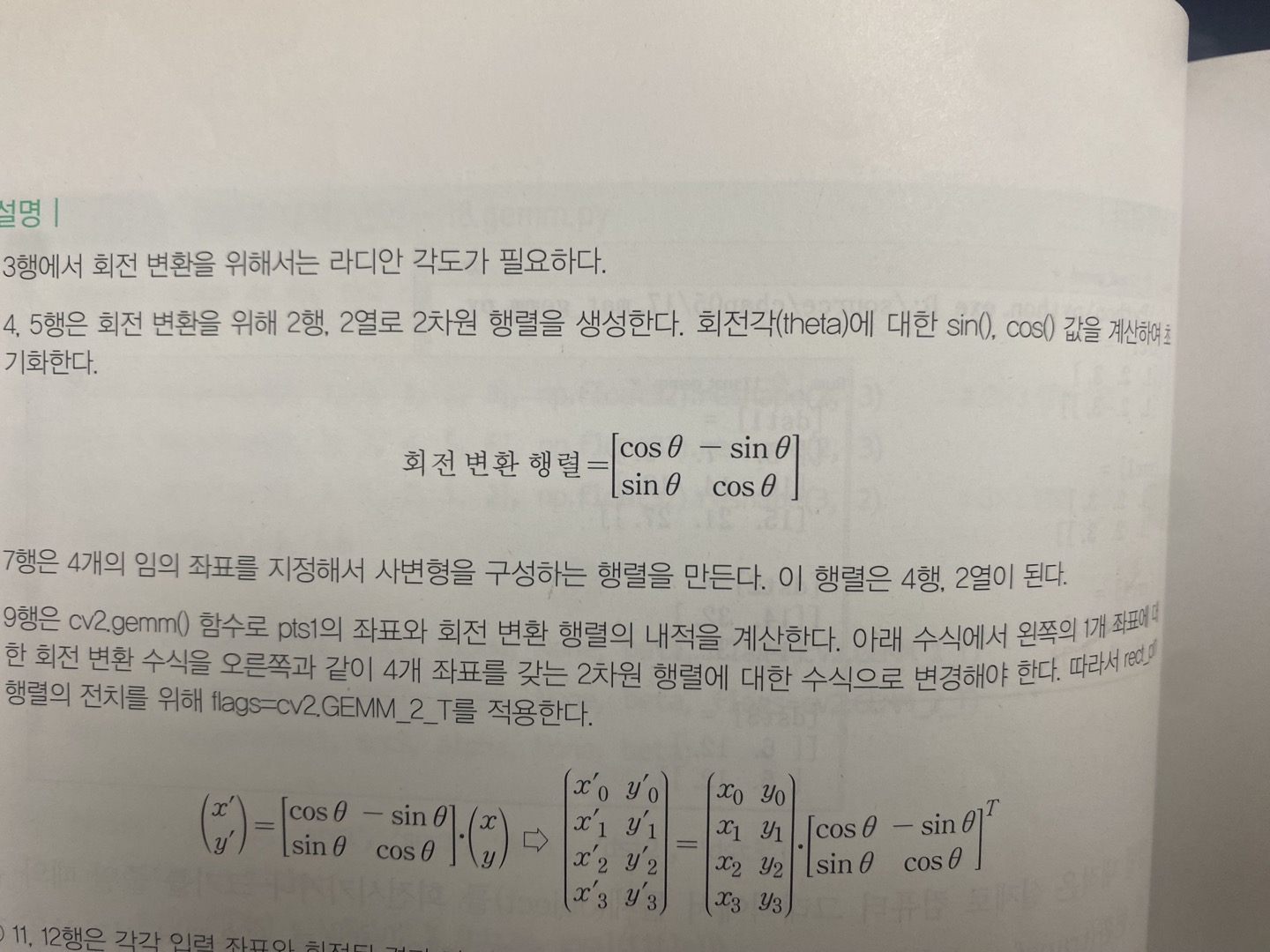
[dst1] =

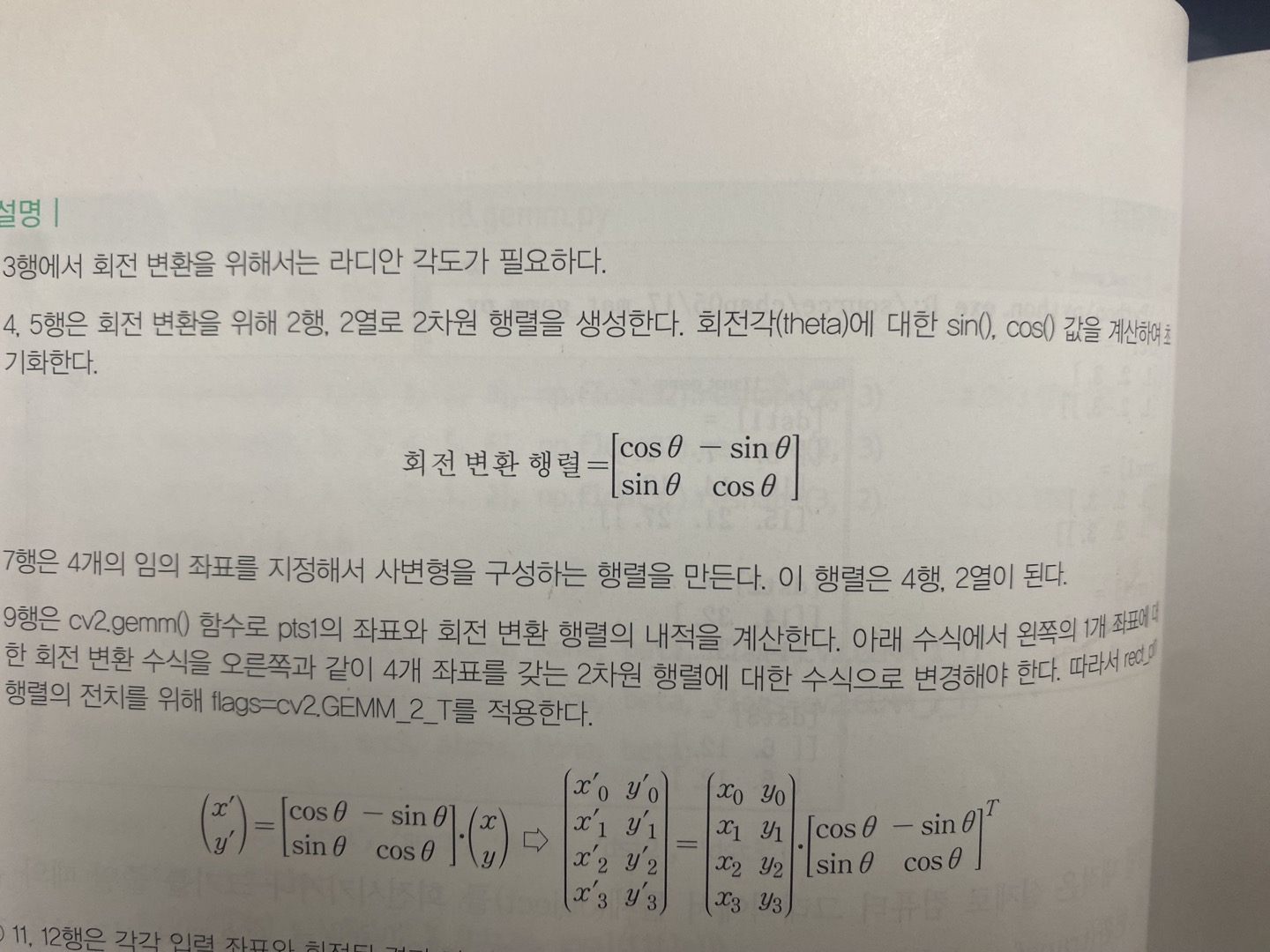
[[ 5. 7. 9.]

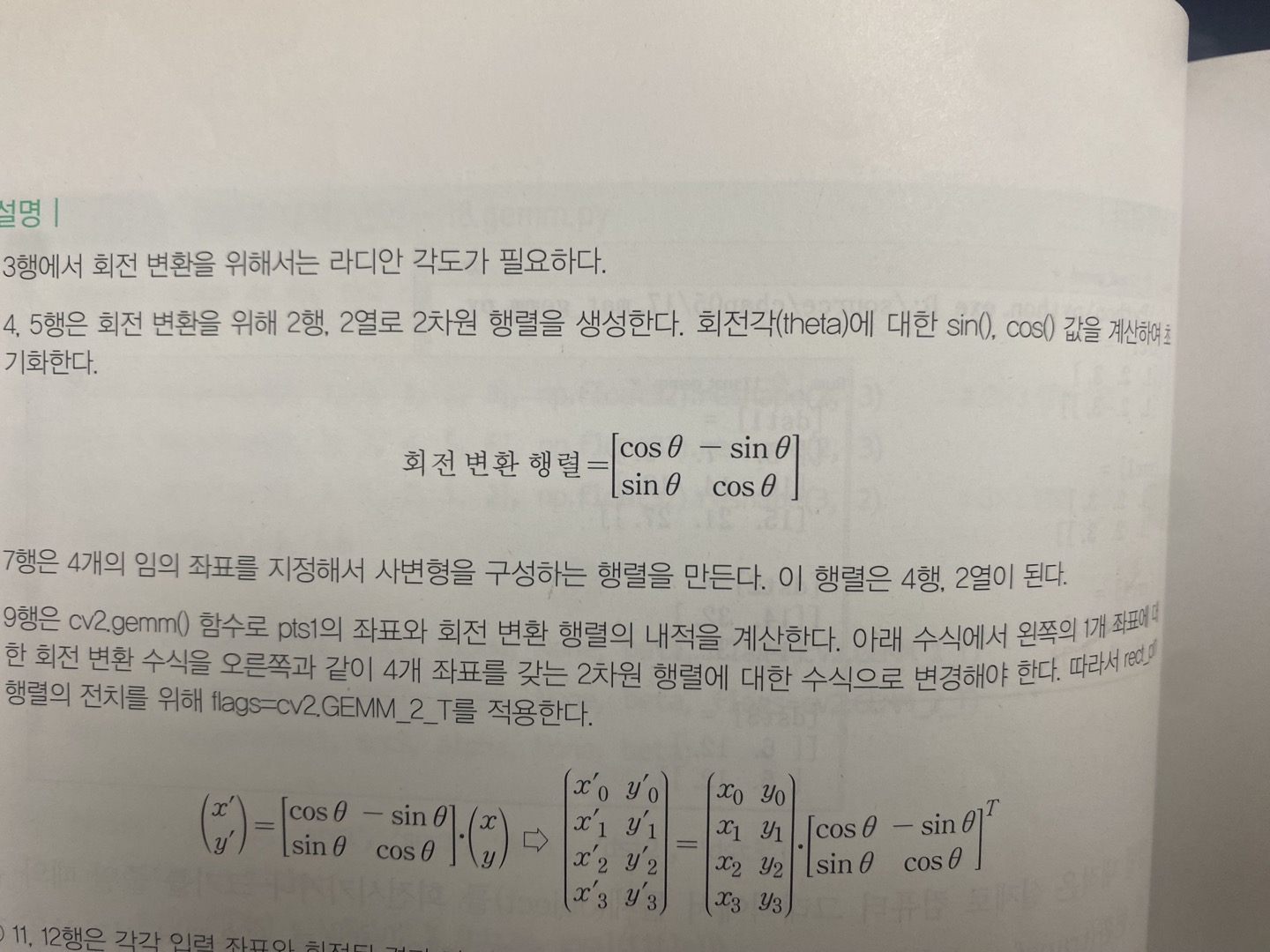
[10. 14. 18.]

[15. 21. 27.]]

#행렬 내적을 이용한 회전변환



 -> **x,y를 세타 만큼 회전 시켜 x`,y`좌표로 이동**

****

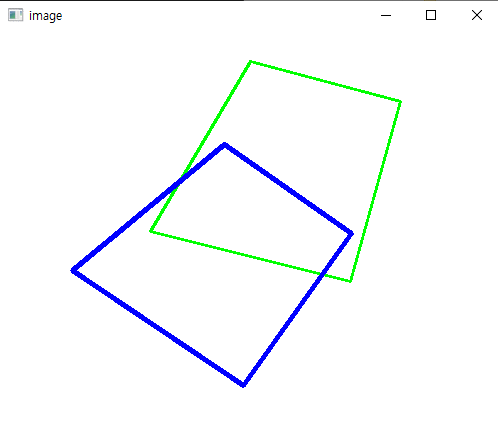
-> 여러 개의 좌표회전시 변환된 수식

->gemm함수 사용하여 회전변환!

pts1 = np.array([(250, 30), (400, 70), (350, 250), (150, 200)], np.float32)

rot\_mat = np.array([[np.cos(theta), -np.sin(theta)], [np.sin(theta), np.cos(theta)]] , np.float32)

pts2 = cv2.gemm(pts1,rot\_mat,1, None, 1,flags=cv2.GEMM\_2\_T)



-->세타= 20으로 설정한 후 회전 변환한 모습

#역행렬을 이용한 선형 연립 방정식

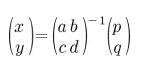
- 역행렬이란?

을 만족하면 A의 역행렬은 입니다.

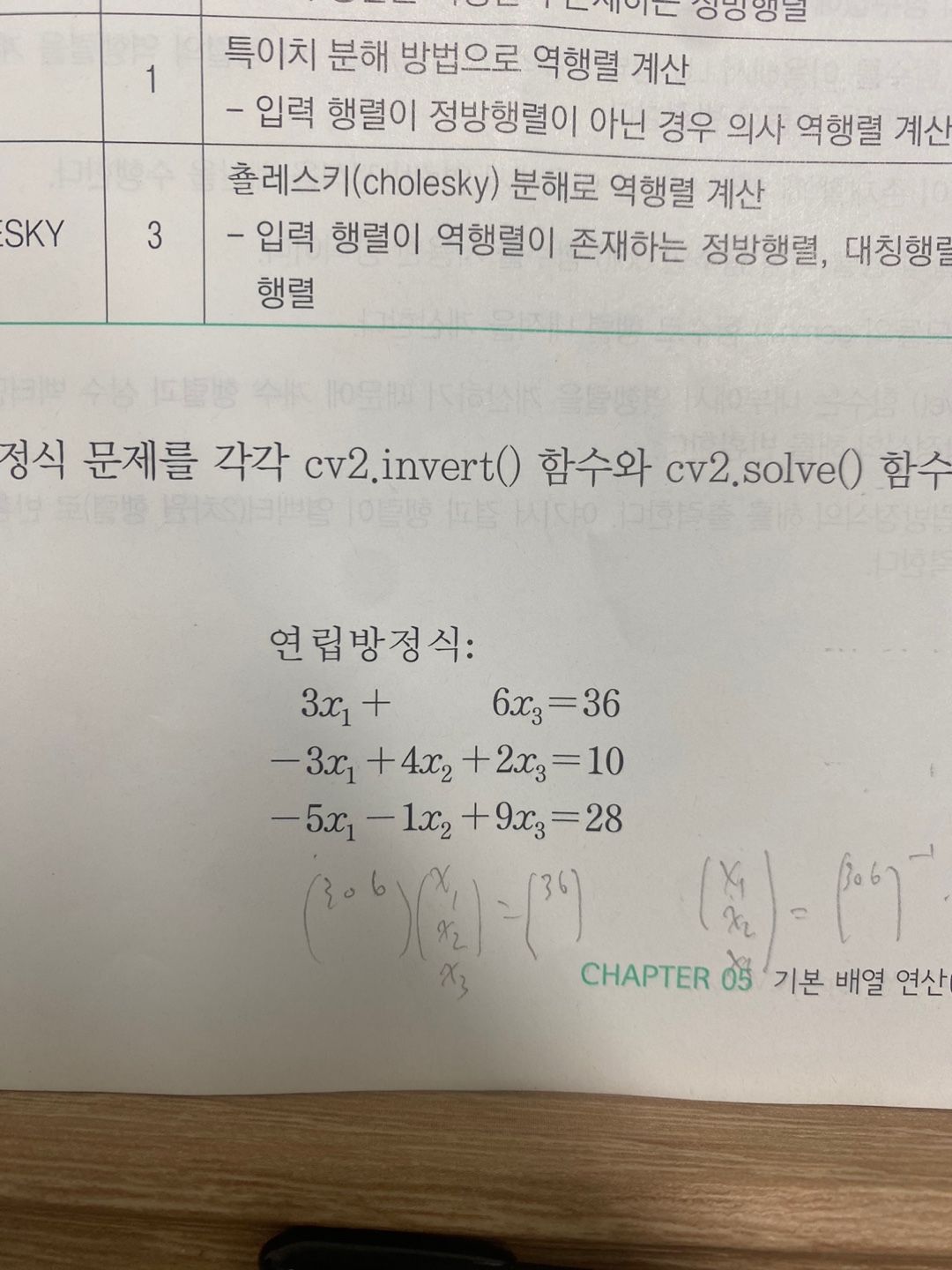
-선형 연립 방정식



- 선형 연립 방정식을 역행렬을 이용하여 간단하게 X,Y값 구할 수 있습니다.



-아래의 연립방정식이 주어졌을경우



data = [ 3, 0, 6, -3, 4, 2, -5,-1, 9]

m1 = np.array(data, np.float32).reshape(3,3)

m2 = np.array([36, 10, 28], np.float32)

ret, inv = cv2.invert(m1, cv2.DECOMP\_LU) ->역행렬 계산

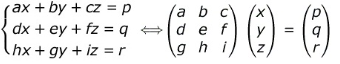
dst1 = cv2.gemm(inv, m2, 1, None, 1) ->역행렬을 통한 풀이

ret, dst2 = cv2.solve(m1, m2, cv2.DECOMP\_LU) ->solve함수를 통한 풀이

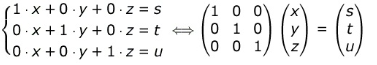


#cv2.DECOMP\_LU 란?

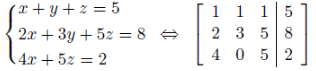
-가우시안 소거법으로 역행렬 계산

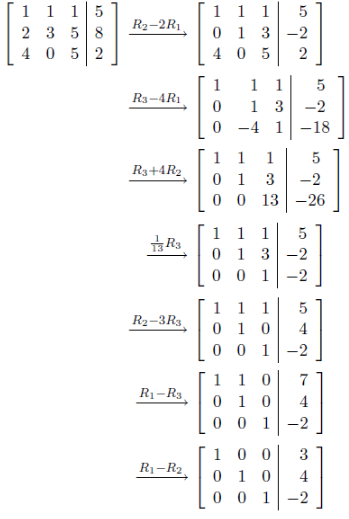


-가감법을 이용하여 단위행렬을 만드는 것이 가우시안 소거법!

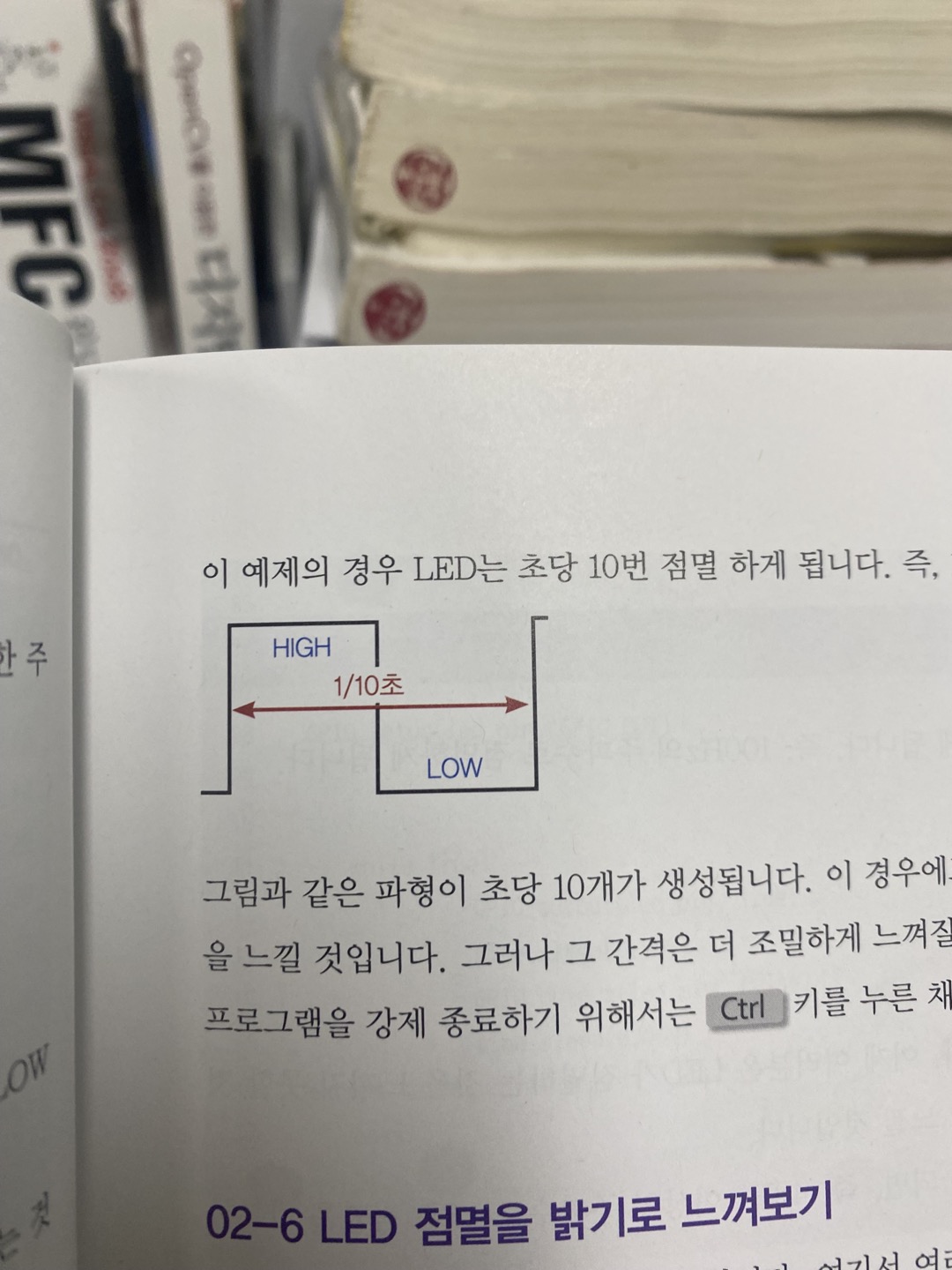


-양변에 같은 수를 더하거나 빼어도 등식은 성립하는 성질을 이용하여 단위행렬로 전환





##라즈베리 파이

#led동작

import RPi.GPIO as GPIO

import time

led\_pin =17

GPIO.setmode(GPIO.BCM) #핀번호 설정

GPIO.setup(led\_pin, GPIO.OUT)

try:

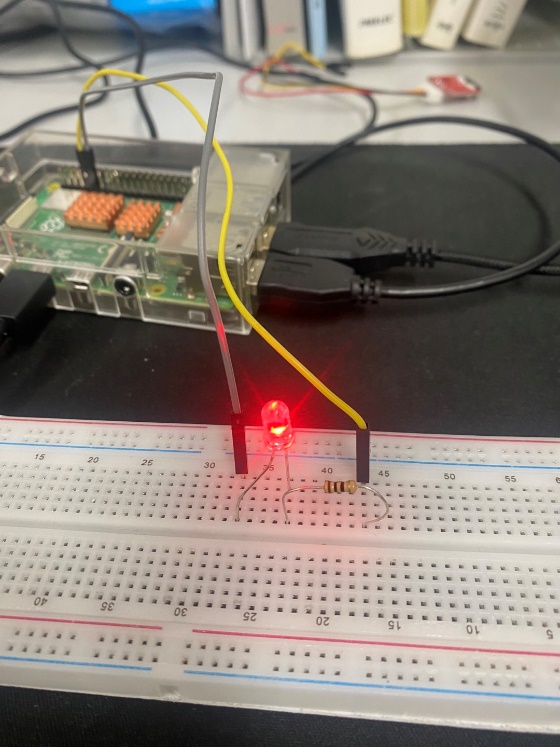
while True:

GPIO.output(led\_pin, True)

time.sleep(0.05)

GPIO.output(led\_pin, False)

time.sleep(0.05)

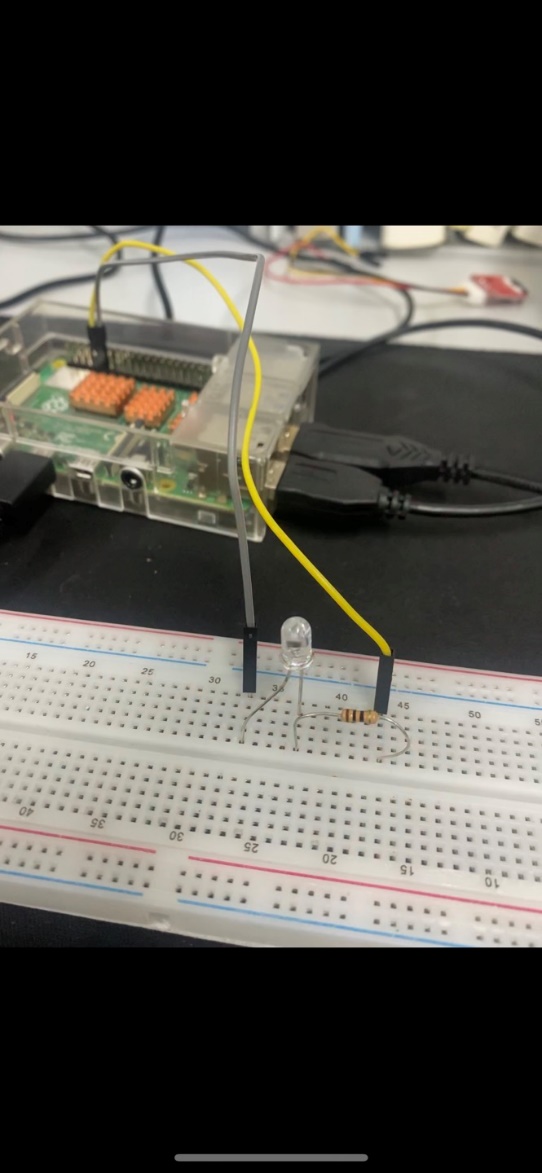
except KeyboardInterrupt:

pass

GPIO.cleanup()

\*\*PWM 함수 사용

import RPi.GPIO as GPIO



led\_pin =17

GPIO.setmode(GPIO.BCM)

GPIO.setup(led\_pin, GPIO.OUT)

pwm = GPIO.PWM(led\_pin, 10.0) #10 Hz

pwm.start(50.0) # 0.0~100.0

try:

while True:

pass

except KeyboardInterrupt:

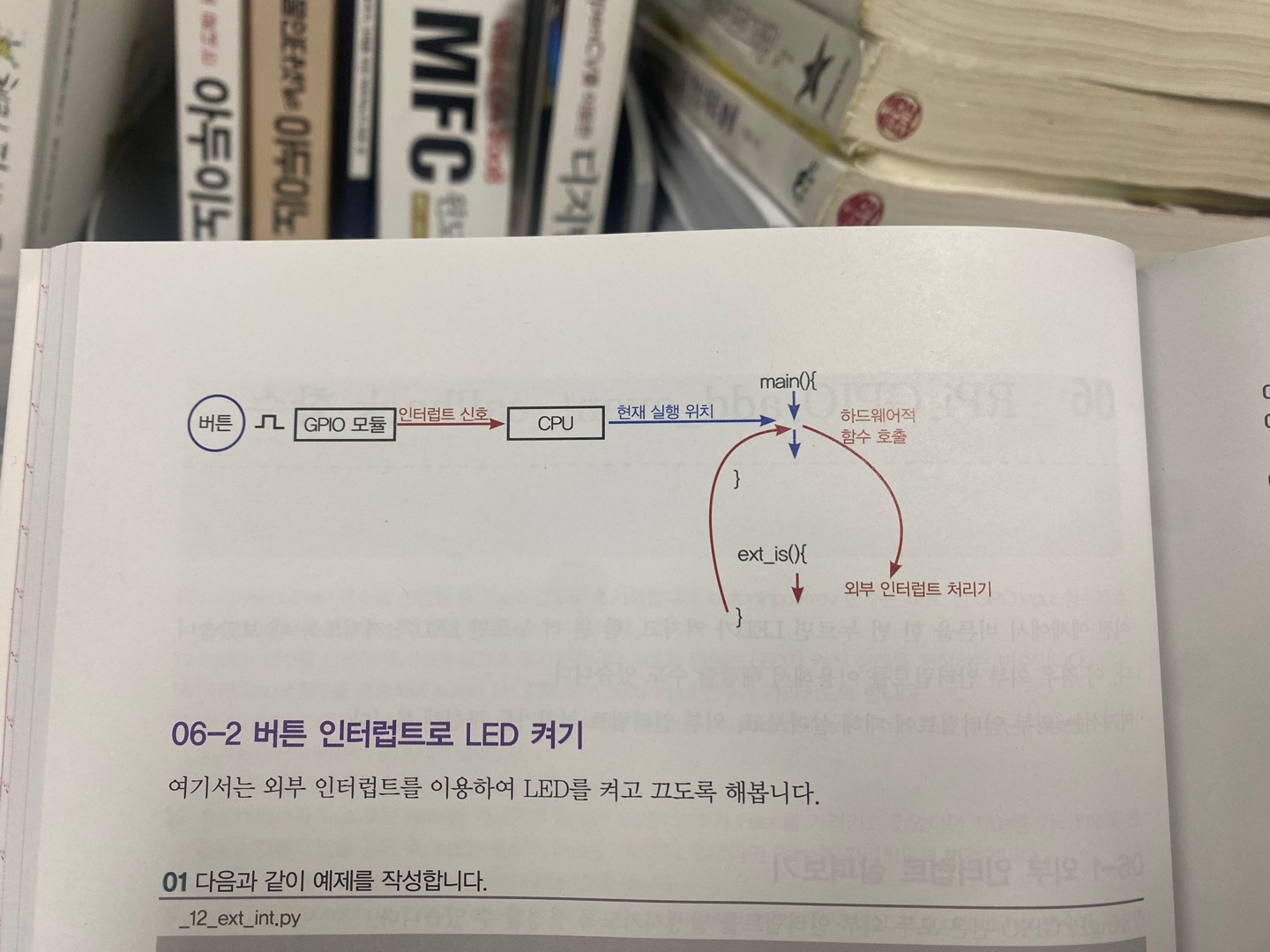
pass

pwm.stop()

GPIO.cleanup()

#인터럽트

->버튼을 통하여 외부 인터럽트 발생



import RPi.GPIO as GPIO

led\_state = False

led\_state\_changed = False #초기상태 설정

def buttonPressed(channel):

global led\_state

global led\_state\_changed

led\_state = True if not led\_state else False

led\_state\_changed = True

button\_pin =27

led\_pin =17

GPIO.setmode(GPIO.BCM)

GPIO.setup(led\_pin, GPIO.OUT)

GPIO.setup(button\_pin, GPIO.IN)

GPIO.add\_event\_detect(button\_pin, GPIO.RISING) #RISING(LOW->HI)

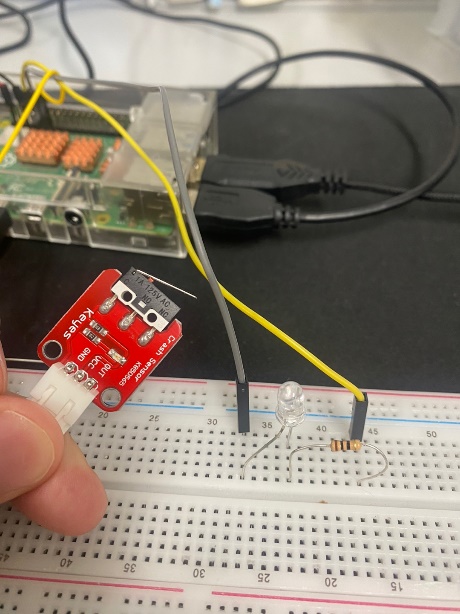
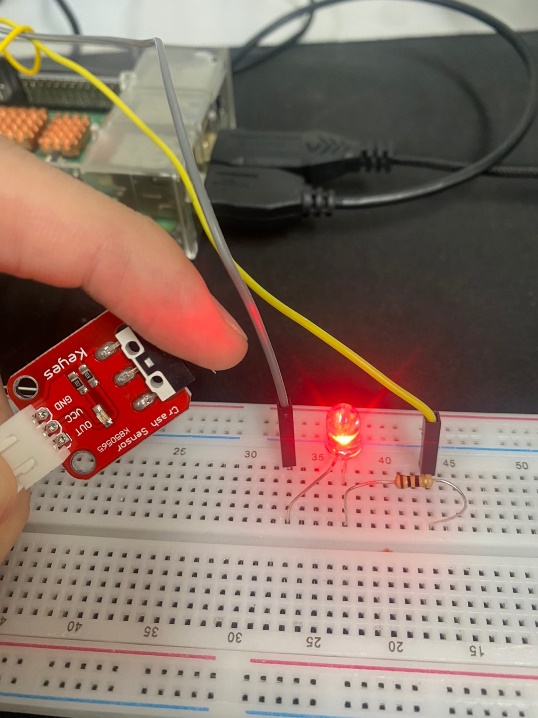
GPIO.add\_event\_callback(button\_pin, buttonPressed) # buttonPressed 함수호출

while True:

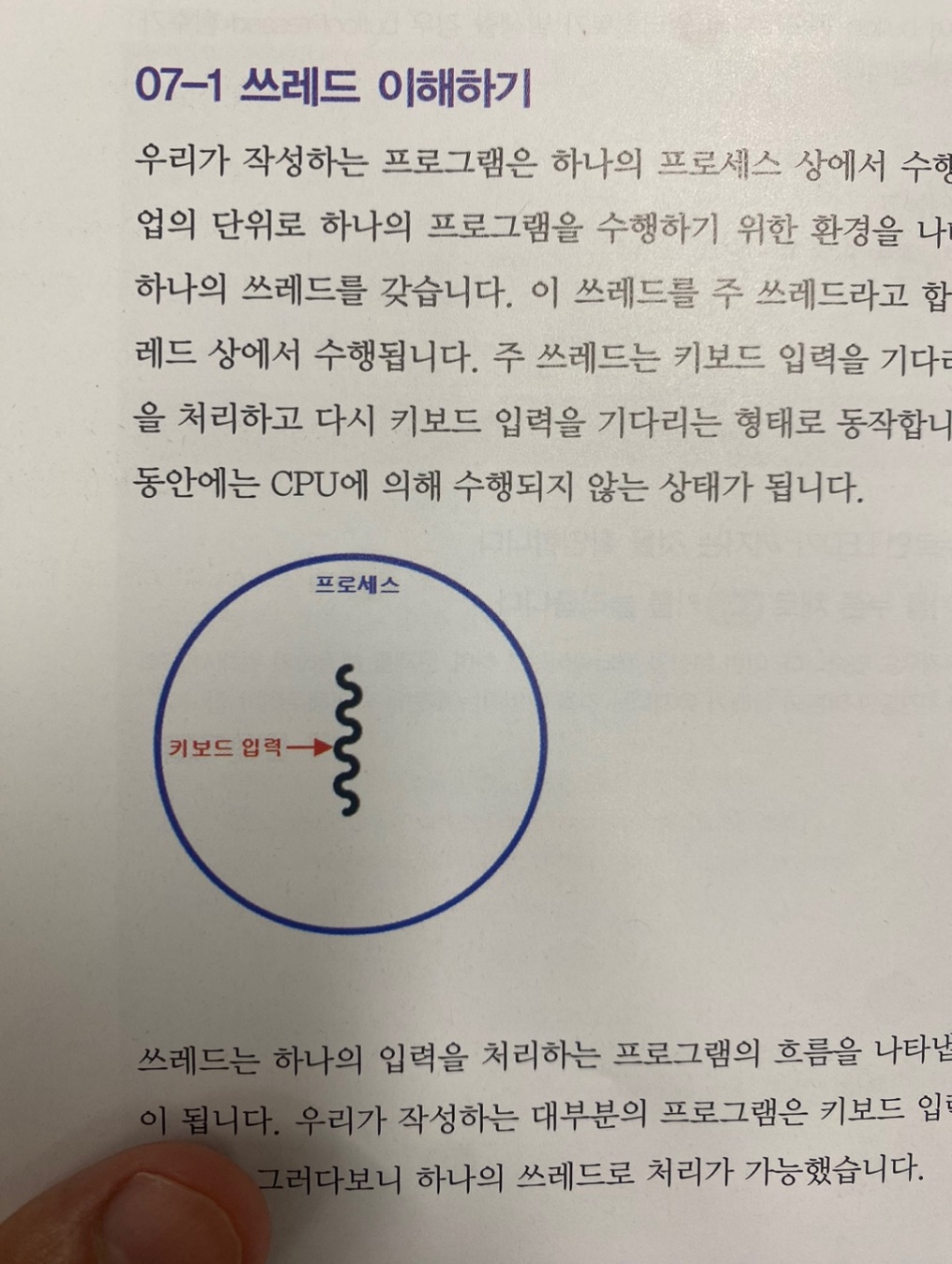
if led\_state\_changed == True:

led\_state\_changed = False

GPIO.output(led\_pin, led\_state)

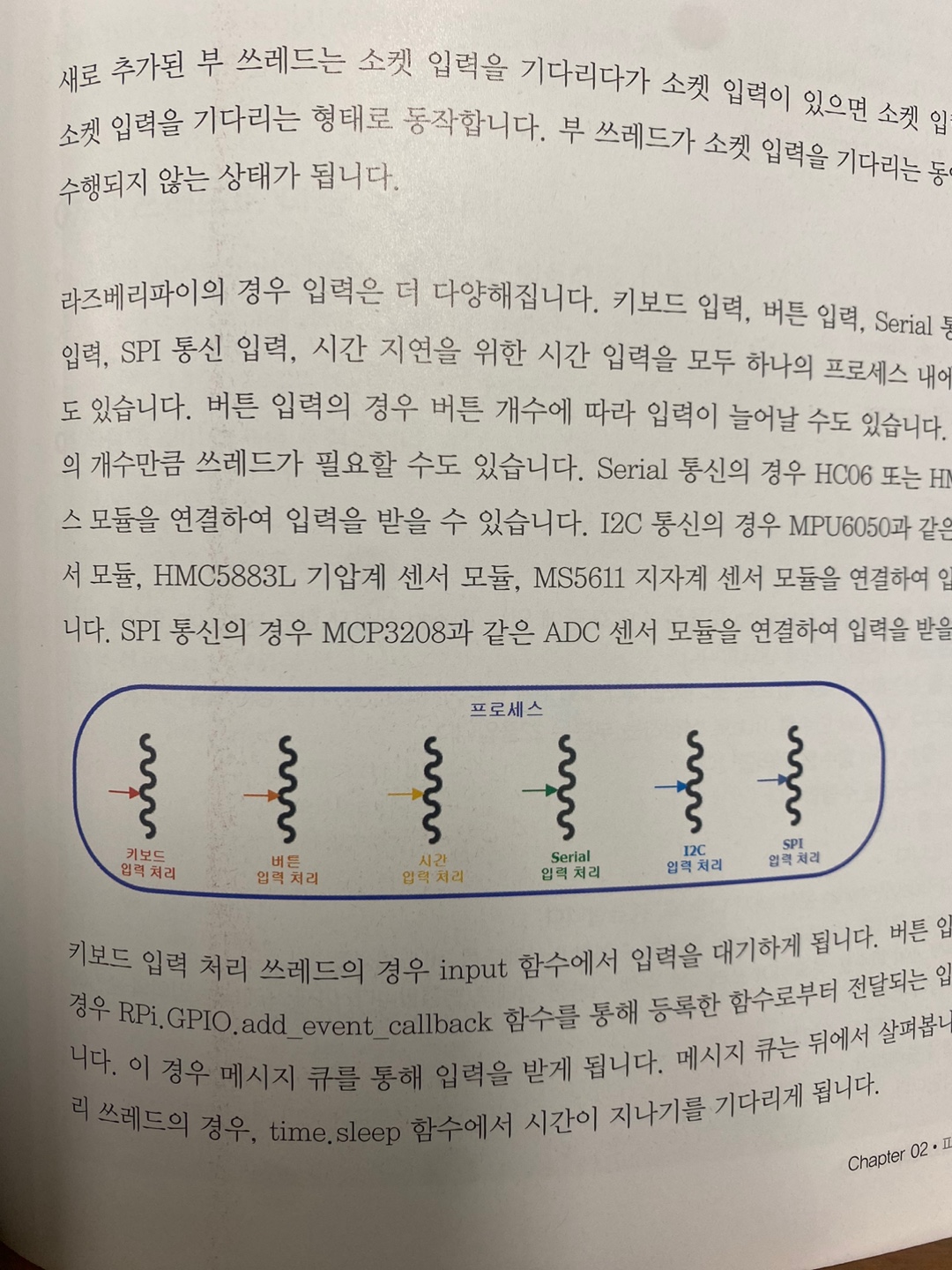
 버튼 클릭 (0->1) 

#THREAD



-> 일반적인 프로그램 같은경우 하나의 프로세스 상에서 수행

->하나의 프로세스=하나의 쓰레드(주쓰레드)



-> 리눅스 운영체제는 쓰레드 프로그램 지원

->하나의 프로세스 안에 다중 쓰레드 지원!

import threading

import time

flag\_exit = False

def t1\_main():

while True:

print("\tt1")

time.sleep(0.5)

if flag\_exit: break

def t2\_main():

while True:

print("\t\tt2")

time.sleep(0.2)

if flag\_exit: break

t1 = threading.Thread(target=t1\_main) #쓰레드1 생성

t1.start()

t2 = threading.Thread(target=t2\_main) #쓰레드2 생성

t2.start()

while True:

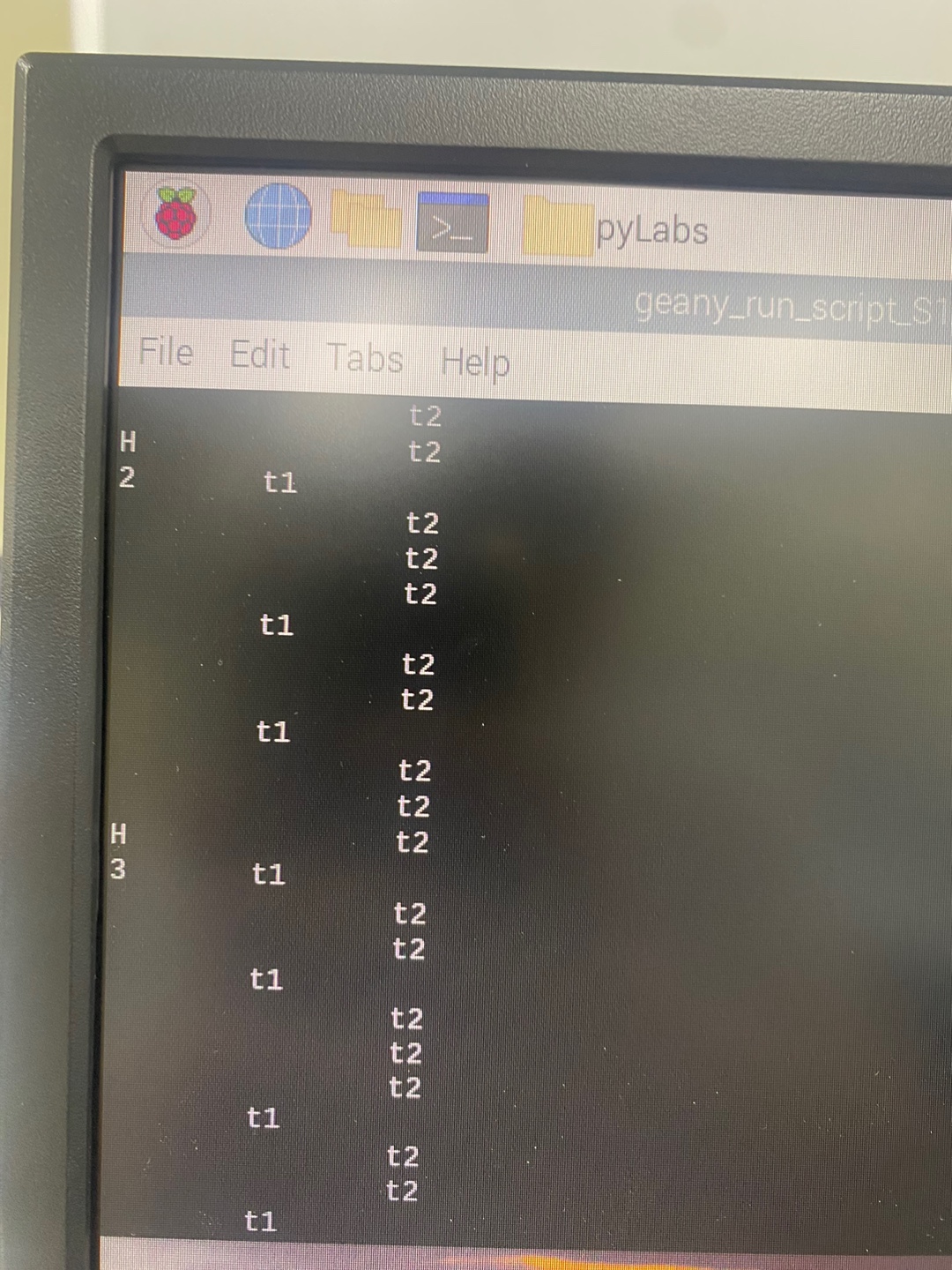
userInput = input()

print(userInput)

flag\_exit = True

t1.join()

t2.join()



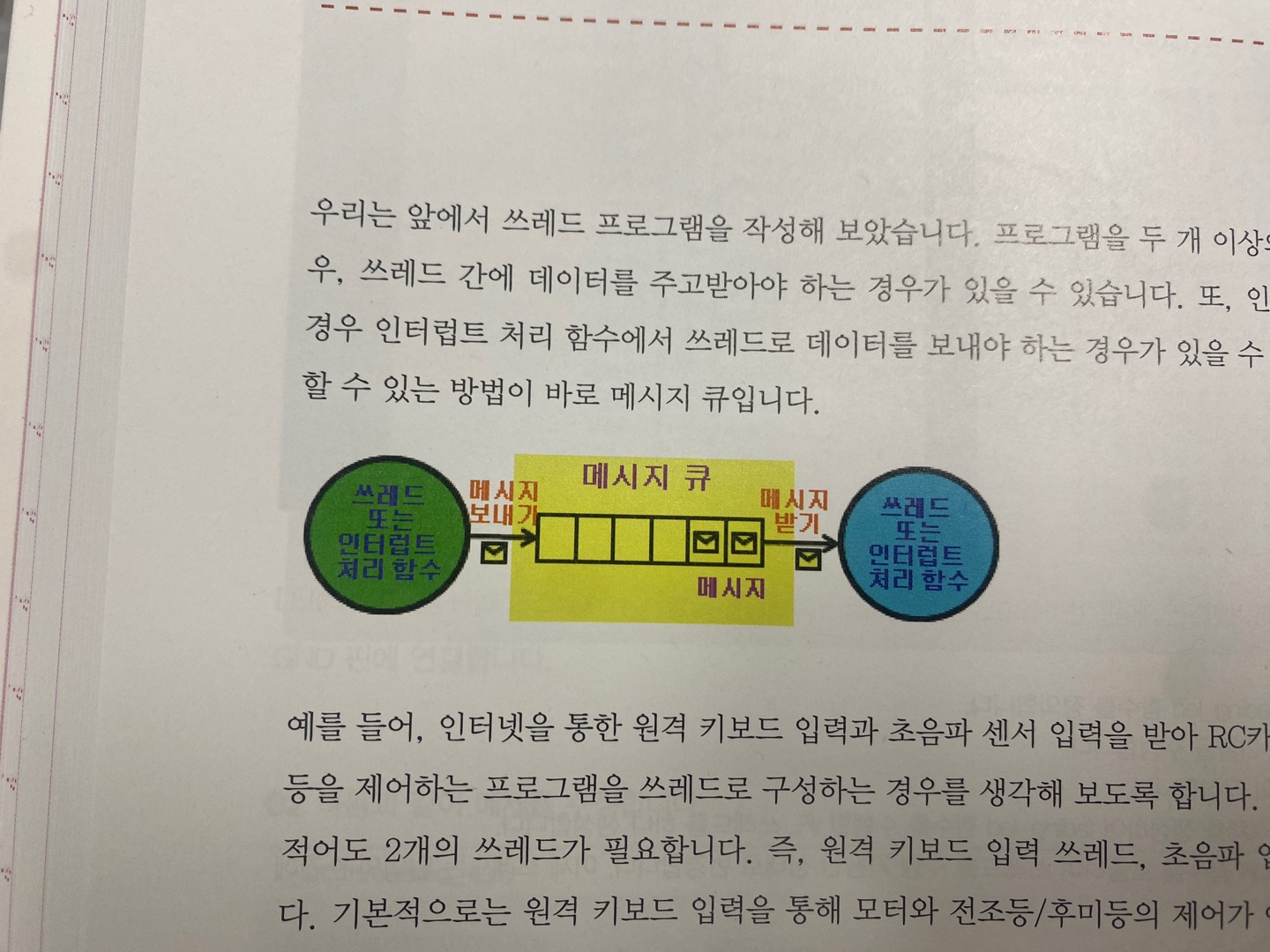
->쓰레드1 은 0.5초 간격으로 print

->쓰레드 2는 0.2초 간격으로 print

->주쓰레드 에서는 input()으로 키보드 입력 받아서 print

#메시지 큐 통신

->두개이상의 쓰레드로 구성할 경우 쓰레드간 데이터 통신을 하기위한 방법



import queue

import RPi.GPIO as GPIO

import time

HOW\_MANY\_MESSAGES =10

mq =queue.Queue(HOW\_MANY\_MESSAGES)

led\_state = False

def buttonPressed(channel):

global led\_state

led\_state = True if not led\_state else False

mq.put(led\_state)

button\_pin =27

led\_pin =22

GPIO.setmode(GPIO.BCM)

GPIO.setup(led\_pin, GPIO.OUT)

GPIO.setup(button\_pin, GPIO.IN)

GPIO.add\_event\_detect(button\_pin, GPIO.RISING)

GPIO.add\_event\_callback(button\_pin, buttonPressed)

while True:

value = mq.get()

GPIO.output(led\_pin, value)

GPIO.cleanup()