

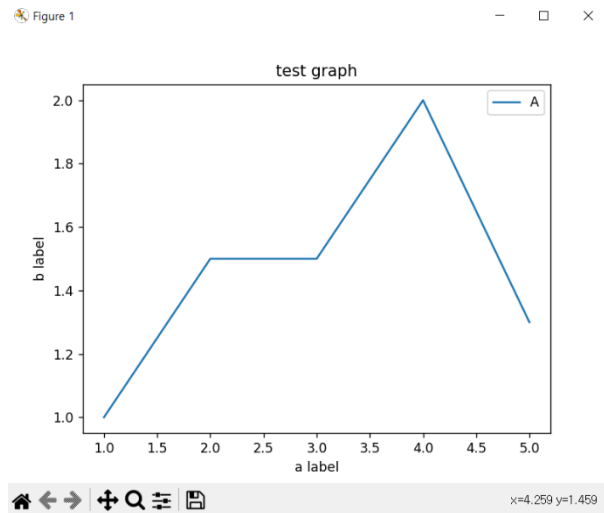
디지털 영상처리 연구실 연구보고서

김우현

##matplotlib

-> 파이썬 표준 시각화도구

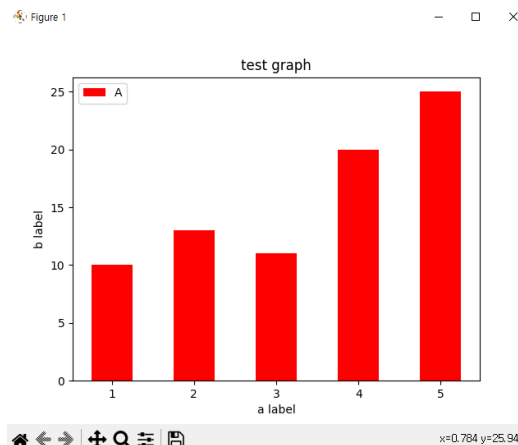
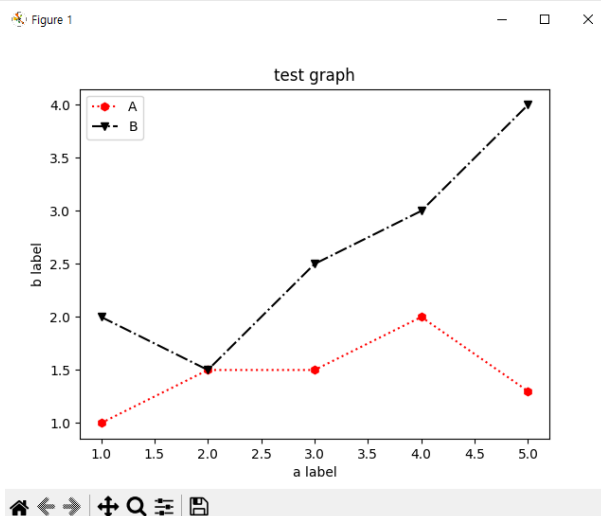
```
import matplotlib.pyplot as plt  
  
plt.plot([1,2,3,4,5],[1,1.5,1.5,2,1.3])  
  
plt.xlabel("a label")  
plt.ylabel("b label")  
  
plt.title("test graph")  
  
plt.legend(["A"])  
  
plt.show()
```



```
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 디스플레이



```
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])
```

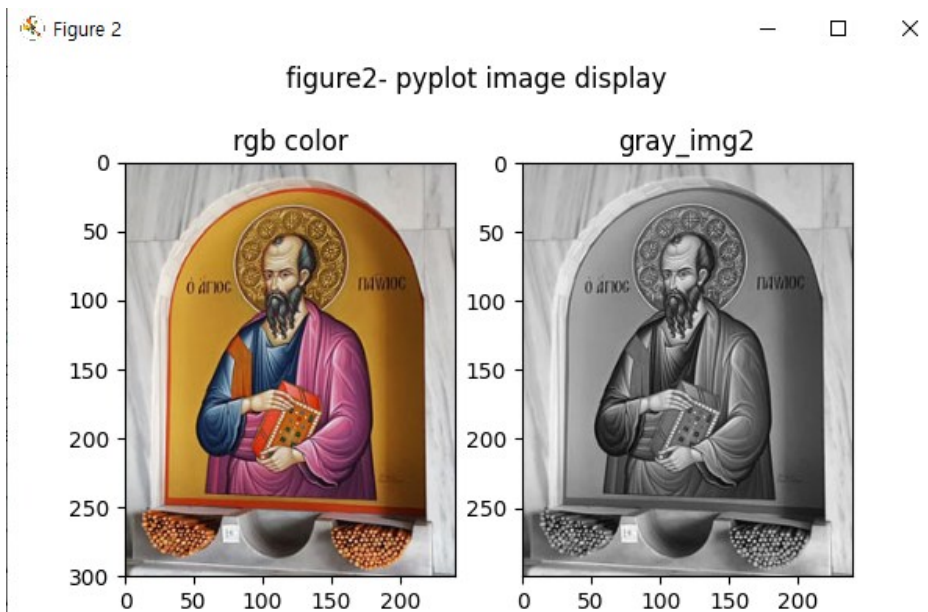
```
plt.title('rgb color')
```

```
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

-> $\text{dst} = \alpha * \text{src1}^T * \text{src2} + \beta * \text{src3}^T$

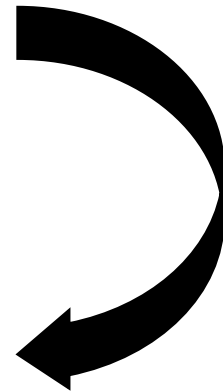
[src1] =	[src2] = ↵	[src3] = ↵
[[1. 2. 3.]	[[1. 2. 3.]↵	[[1. 2.]↵
[1. 2. 3.]]	[4. 5. 6.]]↵	[1. 2.]↵
	↵	[1. 2.]]↵

`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] =	[dst2] = ↵	[dst3] = ↵
[[5. 7. 9.]	[[14. 32.]↵	[[6. 12.]↵
[10. 14. 18.]	[14. 32.]]↵	[6. 12.]]↵
[15. 21. 27.]]		



#행렬 내적을 이용한 회전변환

$$\text{회전 변환 행렬} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix}$$

-> x,y를 세타 만큼 회전 시켜 x',y'좌표로 이동

$$\begin{pmatrix} x'_0 & y'_0 \\ x'_1 & y'_1 \\ x'_2 & y'_2 \\ x'_3 & y'_3 \end{pmatrix} = \begin{pmatrix} x_0 & y_0 \\ x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \end{pmatrix} \cdot \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}^T$$

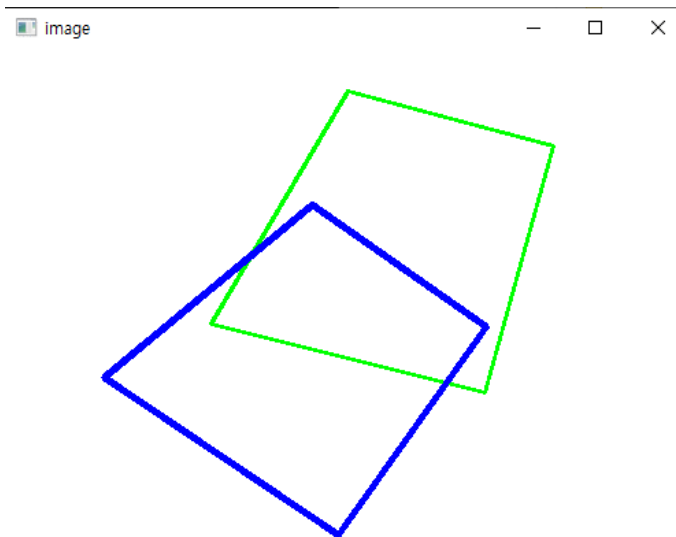
-> 여러 개의 좌표회전시 변환된 수식

-> 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?

$AA^{-1} = I$ 을 만족하면 A의 역행렬은 A^{-1} 입니다.

-선형 연립 방정식

$$\begin{cases} ax + by = p \\ cx + dy = q \end{cases} \Leftrightarrow \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} p \\ q \end{pmatrix}$$

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

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} \begin{pmatrix} p \\ q \end{pmatrix}$$

-아래의 연립방정식이 주어졌을경우

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

```
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
```

```
[dst1] = [2.5238097 2.0238097 4.7380953]
[dst2] = [2.5238097 2.0238097 4.7380953]
```

#cv2.DECOMP_LU 란?

-가우시안 소거법으로 역행렬 계산

$$\begin{cases} ax + by + cz = p \\ dx + ey + fz = q \\ hx + gy + iz = r \end{cases} \Leftrightarrow \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} p \\ q \\ r \end{pmatrix}$$

-가감법을 이용하여 단위행렬을 만드는 것이 가우시안 소거법!

$$\begin{cases} 1 \cdot x + 0 \cdot y + 0 \cdot z = s \\ 0 \cdot x + 1 \cdot y + 0 \cdot z = t \\ 0 \cdot x + 0 \cdot y + 1 \cdot z = u \end{cases} \Leftrightarrow \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} s \\ t \\ u \end{pmatrix}$$

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

$$\begin{cases} x + y + z = 5 \\ 2x + 3y + 5z = 8 \\ 4x + 5z = 2 \end{cases} \Leftrightarrow \left[\begin{array}{ccc|c} 1 & 1 & 1 & 5 \\ 2 & 3 & 5 & 8 \\ 4 & 0 & 5 & 2 \end{array} \right]$$



$$\begin{aligned} \left[\begin{array}{ccc|c} 1 & 1 & 1 & 5 \\ 2 & 3 & 5 & 8 \\ 4 & 0 & 5 & 2 \end{array} \right] &\xrightarrow{R_2 - 2R_1} \left[\begin{array}{ccc|c} 1 & 1 & 1 & 5 \\ 0 & 1 & 3 & -2 \\ 4 & 0 & 5 & 2 \end{array} \right] \\ &\xrightarrow{R_3 - 4R_1} \left[\begin{array}{ccc|c} 1 & 1 & 1 & 5 \\ 0 & 1 & 3 & -2 \\ 0 & -4 & 1 & -18 \end{array} \right] \\ &\xrightarrow{R_3 + 4R_2} \left[\begin{array}{ccc|c} 1 & 1 & 1 & 5 \\ 0 & 1 & 3 & -2 \\ 0 & 0 & 13 & -26 \end{array} \right] \\ &\xrightarrow{\frac{1}{13}R_3} \left[\begin{array}{ccc|c} 1 & 1 & 1 & 5 \\ 0 & 1 & 3 & -2 \\ 0 & 0 & 1 & -2 \end{array} \right] \\ &\xrightarrow{R_2 - 3R_3} \left[\begin{array}{ccc|c} 1 & 1 & 1 & 5 \\ 0 & 1 & 0 & 4 \\ 0 & 0 & 1 & -2 \end{array} \right] \\ &\xrightarrow{R_1 - R_3} \left[\begin{array}{ccc|c} 1 & 1 & 0 & 7 \\ 0 & 1 & 0 & 4 \\ 0 & 0 & 1 & -2 \end{array} \right] \\ &\xrightarrow{R_1 - R_2} \left[\begin{array}{ccc|c} 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & 4 \\ 0 & 0 & 1 & -2 \end{array} \right] \end{aligned}$$

##라즈베리 파이

#led동작

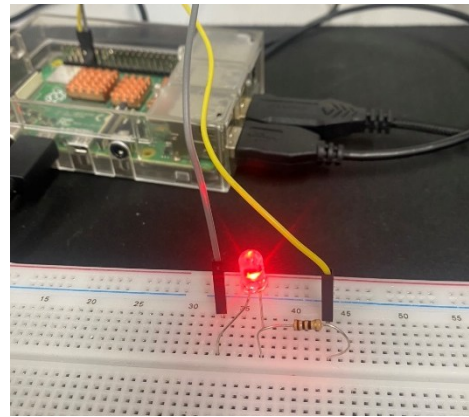
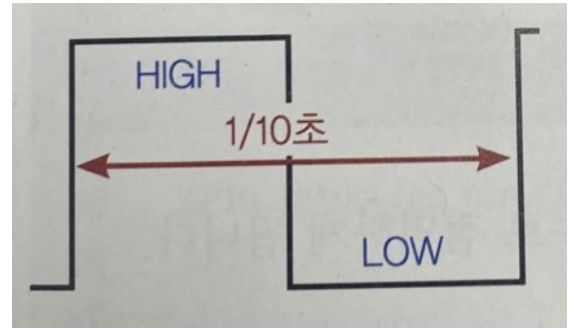
```
import RPi.GPIO as GPIO
import time

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

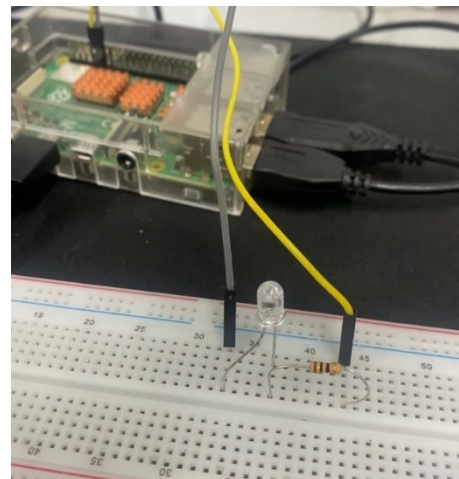
led_pin = 17

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

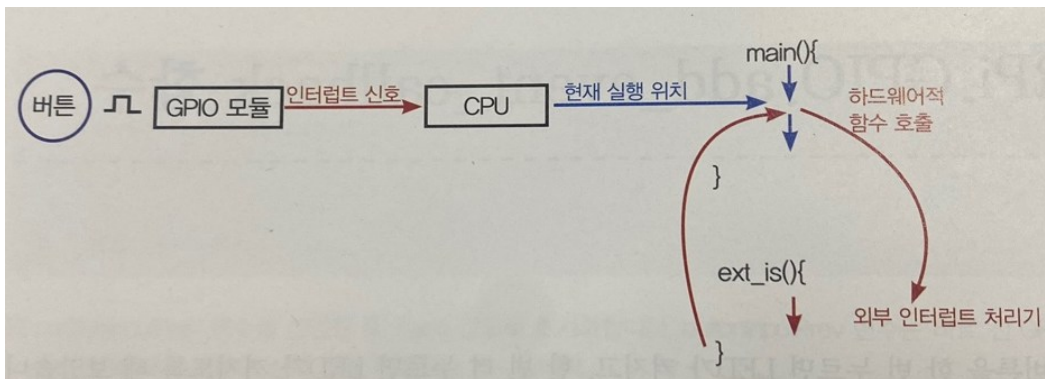
try:
    while True:
        pass
except KeyboardInterrupt:
    pass

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)
```

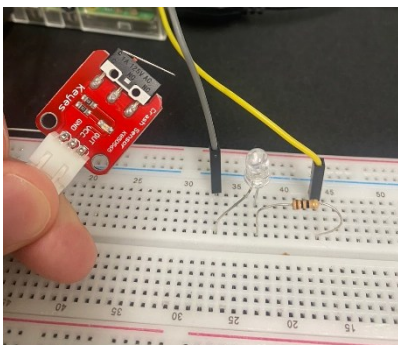
buttonPressed 함수호출

```
while True:
```

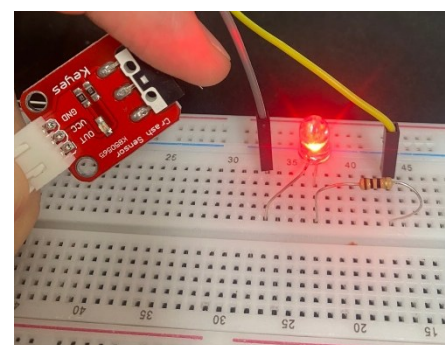
```
    if led_state_changed == True:
```

```
        led_state_changed = False
```

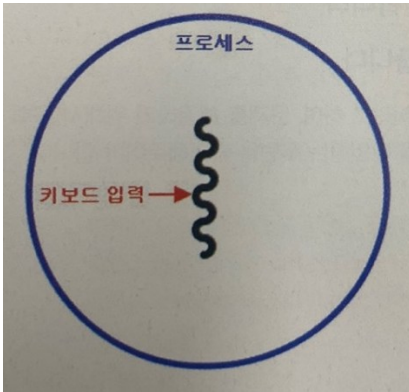
```
        GPIO.output(led_pin, led_state)
```



버튼 클릭 (0->1)

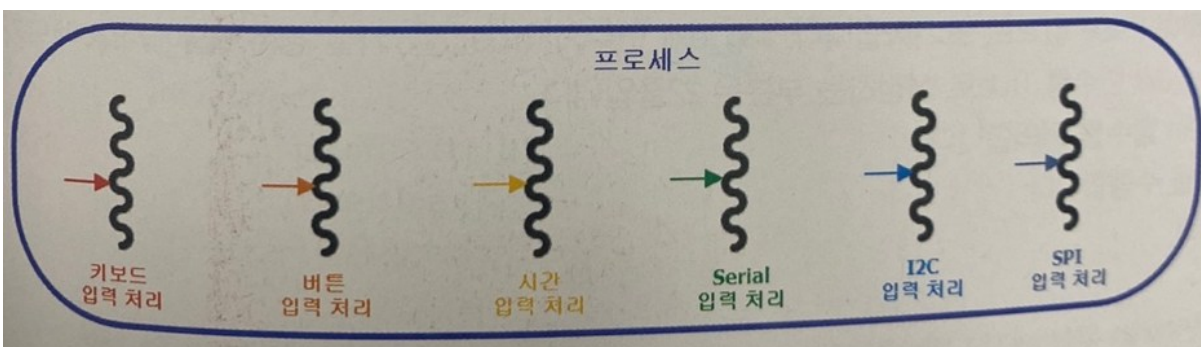


#THREAD



-> 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("Wtt1")
        time.sleep(0.5)
        if flag_exit: break

def t2_main():
    while True:
        print("WtWtt2")
        time.sleep(0.2)
        if flag_exit: break
```

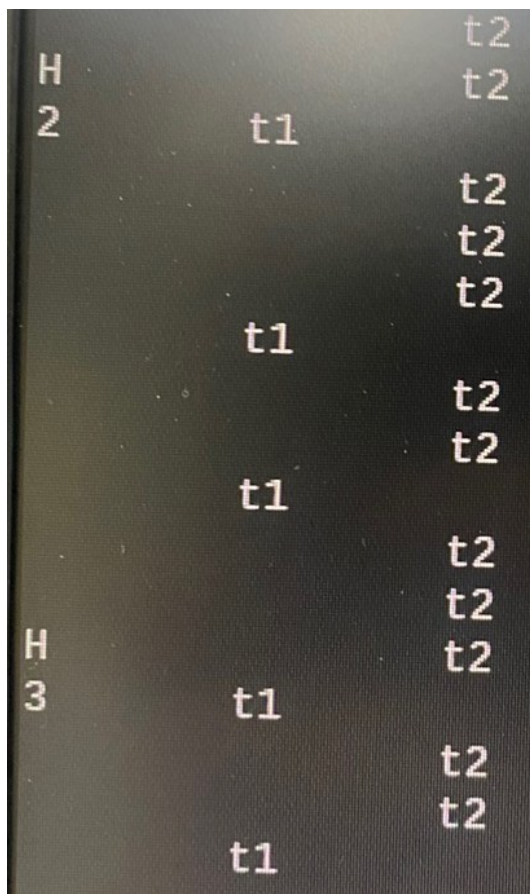
```

t1 = threading.Thread(target=t1_main)          #쓰레드 1 생성
t1.start()
t2 = threading.Thread(target=t2_main)          #쓰레드 2 생성
t2.start()

while True:
    userInput = input()
    print(userInput)

flag_exit = True
t1.join()
t2.join()

```



```

H      t1      t2
2      t1      t2
        t2
        t2
        t2
        t1
        t2
        t2
        t1
        t2
        t2
H      t1      t2
3      t1      t2
        t2
        t2
        t1

```

- >쓰레드 1 은 0.5 초 간격으로 print
- >쓰레드 2 는 0.2 초 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()
```