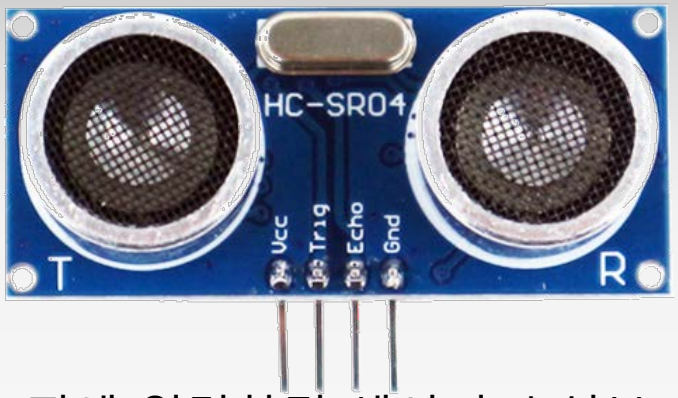
임베디드 보드 실습과 응용 프로그램

Chapter 5.

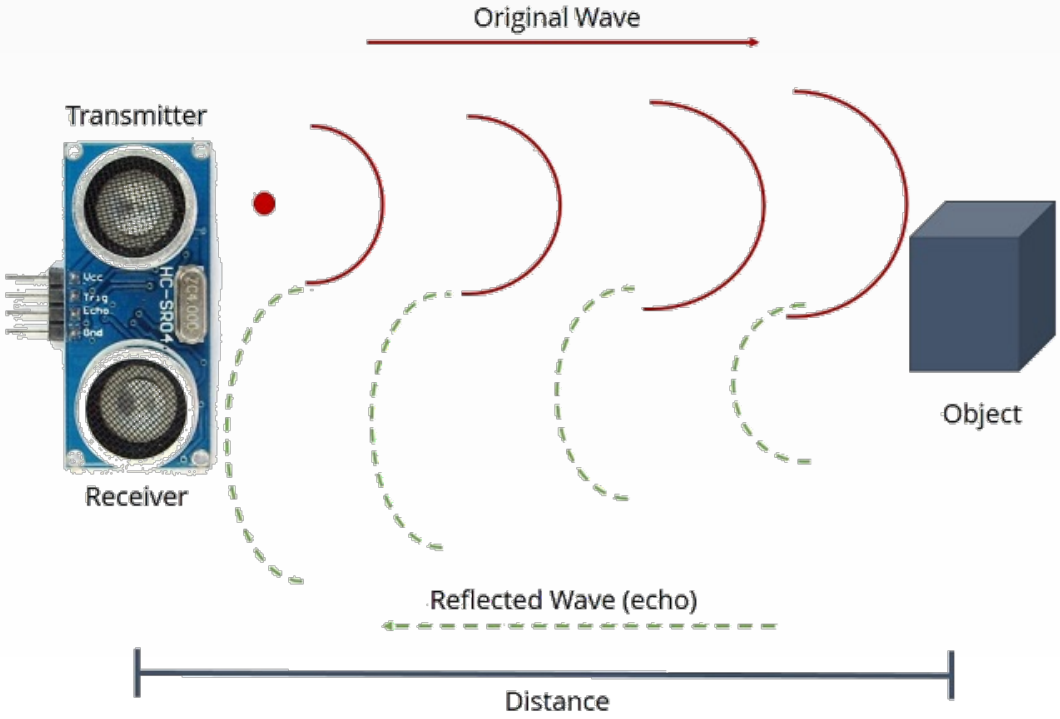
– 외부 디바이스 붙이기

최영근
010-5898-3202

초음파 센서

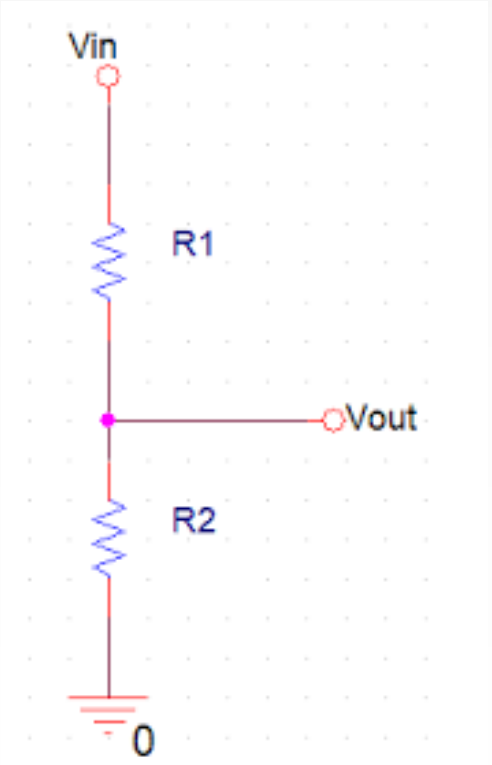


- **HC-SR04**
  - VCC는 5V, GND는 GND에 연결해서 작동
  - 10us의 펄스 신호를 발생시켜 Trig(Trigger) 핀에 입력하면 센서의 송신부(Transmitter)에서는 25us 음파 8개를 송신하고, 송신된 음파가 물체에 부딪혀서 돌아와서 수신부(Receiver)로 수신되는데,
  - Echo 핀에서는 음파가 송신된 시점부터 수신되는 시점까지 HIGH 신호를 출력



초음파 센서

- **전압 분배 회로**
  - HC-SR04는 5V의 HIGH 신호가 인가되며 라즈베리 파이는 3.3V 이하의 전압을 인가받을 수 있으므로 Echo 핀에는 전압 분배 회로를 구성해서 연결해야 한다.



  - 전압 분배 법칙 : $V_{out} = V_{in} \times \frac{R2}{R1 + R2}$

  - V_{in} : 5V, $R1$: 1kΩ, $R2$: 2kΩ 이면 V_{out}의 값은?

초음파 센서

- **초음파 센서를 이용한 물체의 거리 측정**

  - $s = vt$
  - 음파의 속도 (v) : 343m/s

  - Echo 핀으로 입력되는 HIGH 시간은 음파가 왕복하는 시간이므로
  - 센서와 물체 사이의 거리 s 를 구하기 위한 t 는 Echo 핀의 HIGH 시간 / 2

  - $s = 34300 \times \frac{(Echo\text{ 핀의 }HIGH\text{ 시간})}{2}$

    $= Echo\text{ 핀의 }HIGH\text{ 시간} \times 17150$

시간 측정

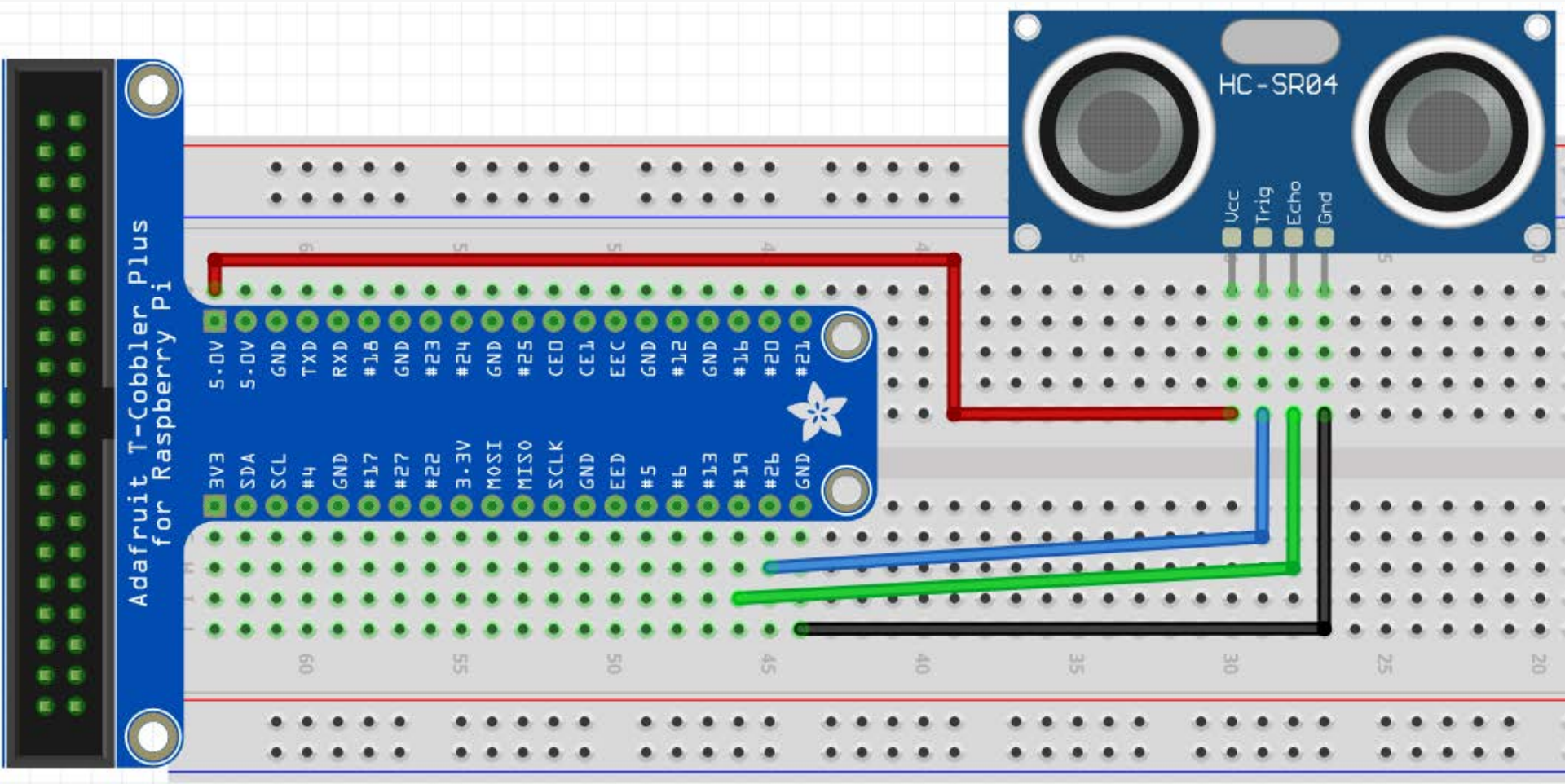
- **time.time()**
  - 1970년 1월 1일 0시 0분 0.00000초 (UTC) 로부터 몇 초의 시간이 지났는지 (Unix timestamp) 아래와 같은 실수 형식으로 반환하는 함수

  - Unix timestamp는
  - **시간 간격을 정수 또는 실수 값의 뺄셈으로 연산 가능**
  - 년, 월, 일, 시, 분, 초로 표현하는 것보다 작은 저장공간 차지
  - 시차 고려할 필요 없이 어느 국가에서나 동일한 값

```
>>> import time
>>> time.time()
1691509802.41505
```

초음파 센서

- 결선


HC-SR04
Vcc
Trig
Echo
Gnd
Adafruit T-Cobbler Plus
for Raspberry Pi
3V3
5.0V
SDA
5.0V
SCL
GND
#4
TXD
GND
RXD
#17
#18
#27
GND
#22
#23
3.3V
#24
MOSI
GND
MISO
#25
SCLK
CE0
GND
CE1
EEC
EED
GND
#5
#6
#12
#13
GND
#19
#16
#26
#20
GND
#21

초음파 센서

- 소스 코드

```
import RPi.GPIO as gpio
import time

gpio.setmode(gpio.BCM)

TRIG = 26
ECHO = 19

print(">>> Distance Measurement with HC-SR04")
gpio.setup(TRIG,gpio.OUT)
gpio.setup(ECHO,gpio.IN)

gpio.output(TRIG, False)
print("Waiting for HC-SR04 to settle")
time.sleep(2)

gpio.output(TRIG, True) # 10us 펄스 생성
time.sleep(0.00001)
gpio.output(TRIG, False)

while gpio.input(ECHO)==0:  # ECHO 핀으로 입력된 신호가 LOW->HIGH가 되었을 때
  pulse_start = time.time() # 시간 값을 저장

while gpio.input(ECHO)==1:  # ECHO 핀으로 입력된 신호가 HIGH->LOW가 되었을 때
  pulse_end = time.time()   # 시간 값을 저장

pulse_duration = pulse_end - pulse_start    # ECHO의 HIGH 신호가 유지된 시간
distance = pulse_duration * 17150           # 센서와 물체 사이의 거리
distance = round(distance, 2)               # 소수점 이하 2번째 자리까지만

print("Distance: %s cm" % distance)         # HC-SR04는 mm 단위 오차가 발생할 수 있으므로

gpio.cleanup()
```

RGB LED

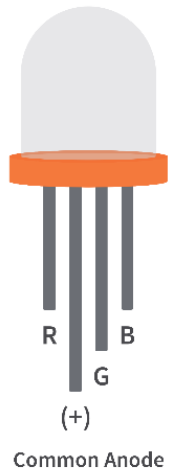
- **RGB color model**
  - 인간의 눈으로 인식할 수 있는 모든 색은 빛의 3원색인 빨강(Red), 초록(Green), 파랑(blue)를 적절한 비율로 혼합해서 얻을 수 있다.
  - 빛의 3원색은 각각의 색 성분을 하나의 축으로 하는 3차원 좌표 시스템에 대응시킬 수 있다.

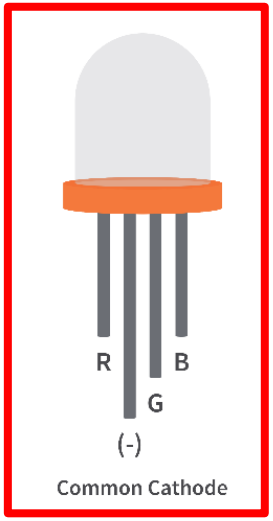
  - R, G, B 모든 성분이 0이면(0, 0, 0) 검은색,
  - R, G, B 모든 성분이 최댓값이라면(255, 255, 255) 흰색,
  - R과 G는 최댓값, B는 0이라면(255, 255, 0) 노란색

  - RGB 색상표
  - https://search.naver.com/search.naver?sm=tab_hty.top&where=nexearch&query=rgb+%EC%83%89%EC%83%81%ED%91%9C&oquery=rgb+%EC%83%89%EC%83%81%ED%91%9C&tqi=h4Cr%2Bwprvh8ssZZmZ7dssssssAR-043848

RGB LED

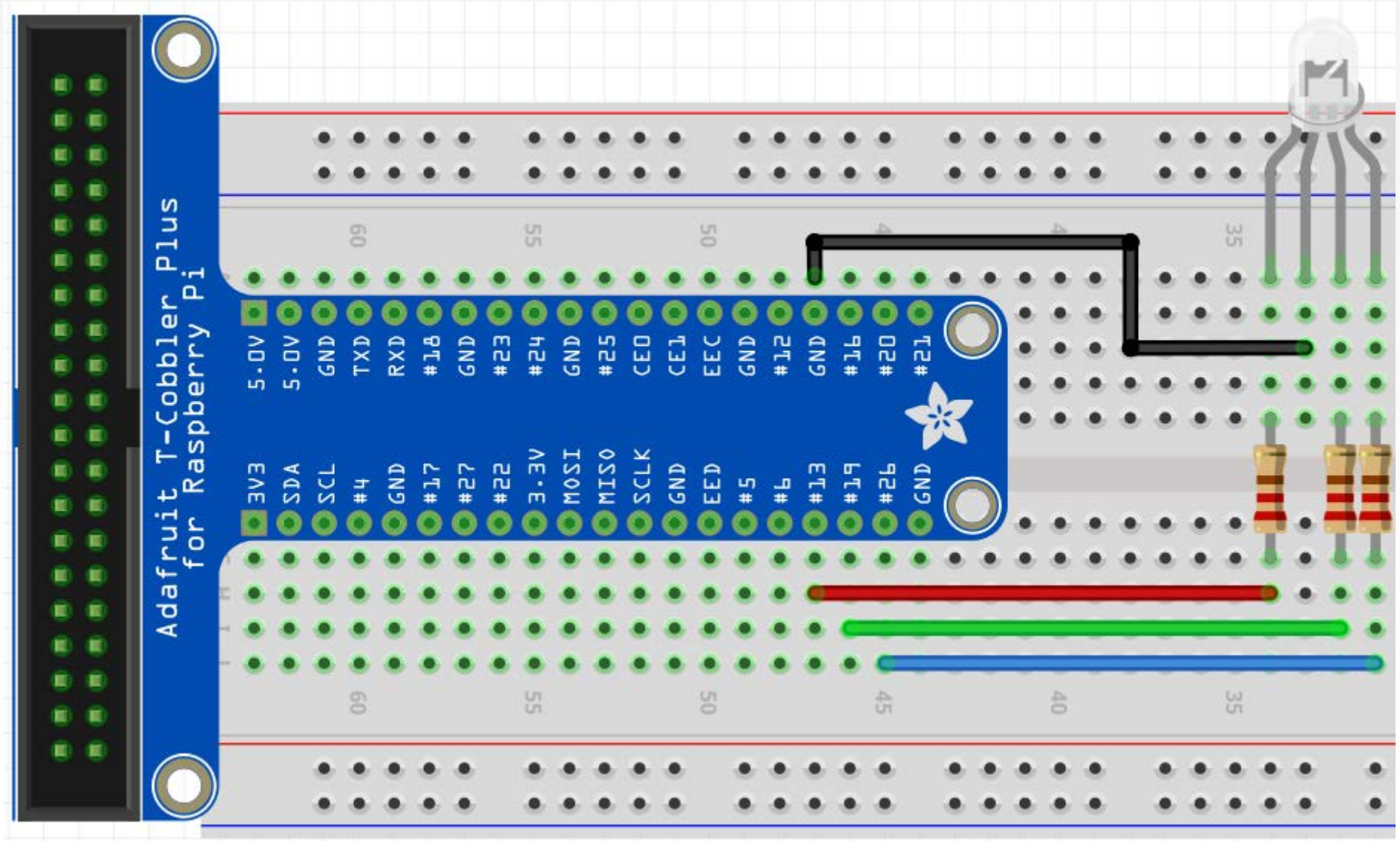
- **RGB LED**
  - RGB LED는 red, green, blue 3개의 LED를 RGB 컬러 모델에 의해 하나의 LED 색상으로 합친 LED라고 할 수 있다.
  - 아래 그림과 같이 4개의 다리가 있으며 3개의 다리를 통해 각각 R, G, B 성분의 값이 전송되어 색상을 결정한다.
  - 라즈베리 파이에 RGB LED를 연결할 때 RGB LED가 common Anode 타입일 경우에는 나머지 1개의 다리를 +5V에, common Cathode 타입일 경우에는 GND에 연결해주면 된다.
  - 220Ω 또는 330Ω의 저항을 각각의 다리와 연결하여 결선한다.





RGB LED

- 결선


Adafruit T-Cobbler Plus
for Raspberry Pi
3V3
5.0V
SDA
5.0V
SCL
GND
#4
TXD
GND
RXD
#17
#18
#27
GND
#22
#23
3.3V
#24
MOSI
GND
MISO
#25
SCLK
CE0
GND
CE1
EEDD
EEDC
#5
GND
#6
#12
#13
GND
#19
#16
#26
#20
GND
#21

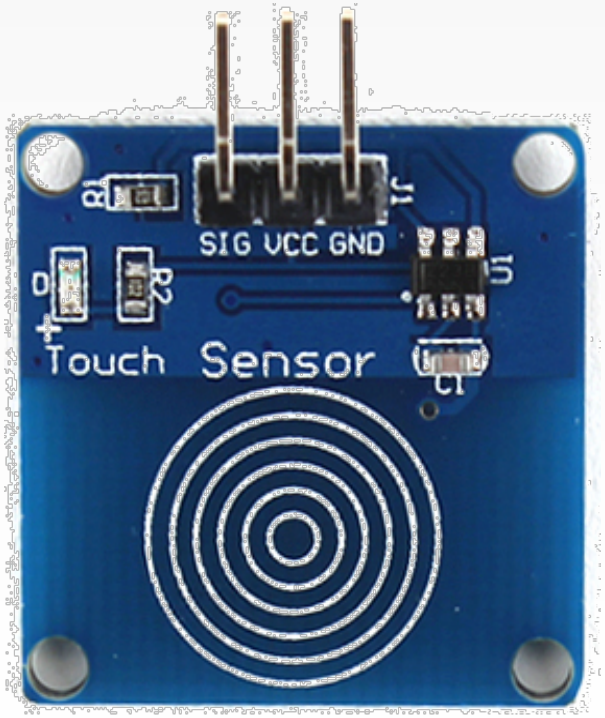
RGB LED

- 소스 코드

```
import RPi.GPIO as gpio

gpio.setmode(gpio.BCM)

red_pin     = 13
green_pin   = 19
blue_pin    = 26

gpio.setup(red_pin, gpio.OUT)
gpio.setup(green_pin, gpio.OUT)
gpio.setup(blue_pin, gpio.OUT)

red_val     = input("Red value (0~255) : ")     # R, G, B 값을 키보드로 입력
green_val   = input("Green value (0~255) : ")
blue_val    = input("Blue value (0~255) : ")

red = gpio.PWM(red_pin, 75)
green = gpio.PWM(green_pin, 75)
blue = gpio.PWM(blue_pin, 75)

try:
        while True:                             # R,G,B의 값은 0 ~ 255
                red.start((red_val/2.55))       # 0 ~ 100 사이의 듀티 비로 변환
                green.start((green_val/2.55))
                blue.start((blue_val/2.55))

except KeyboardInterrupt:
    pass

red.stop()
green.stop()
blue.stop()

gpio.cleanup()
```

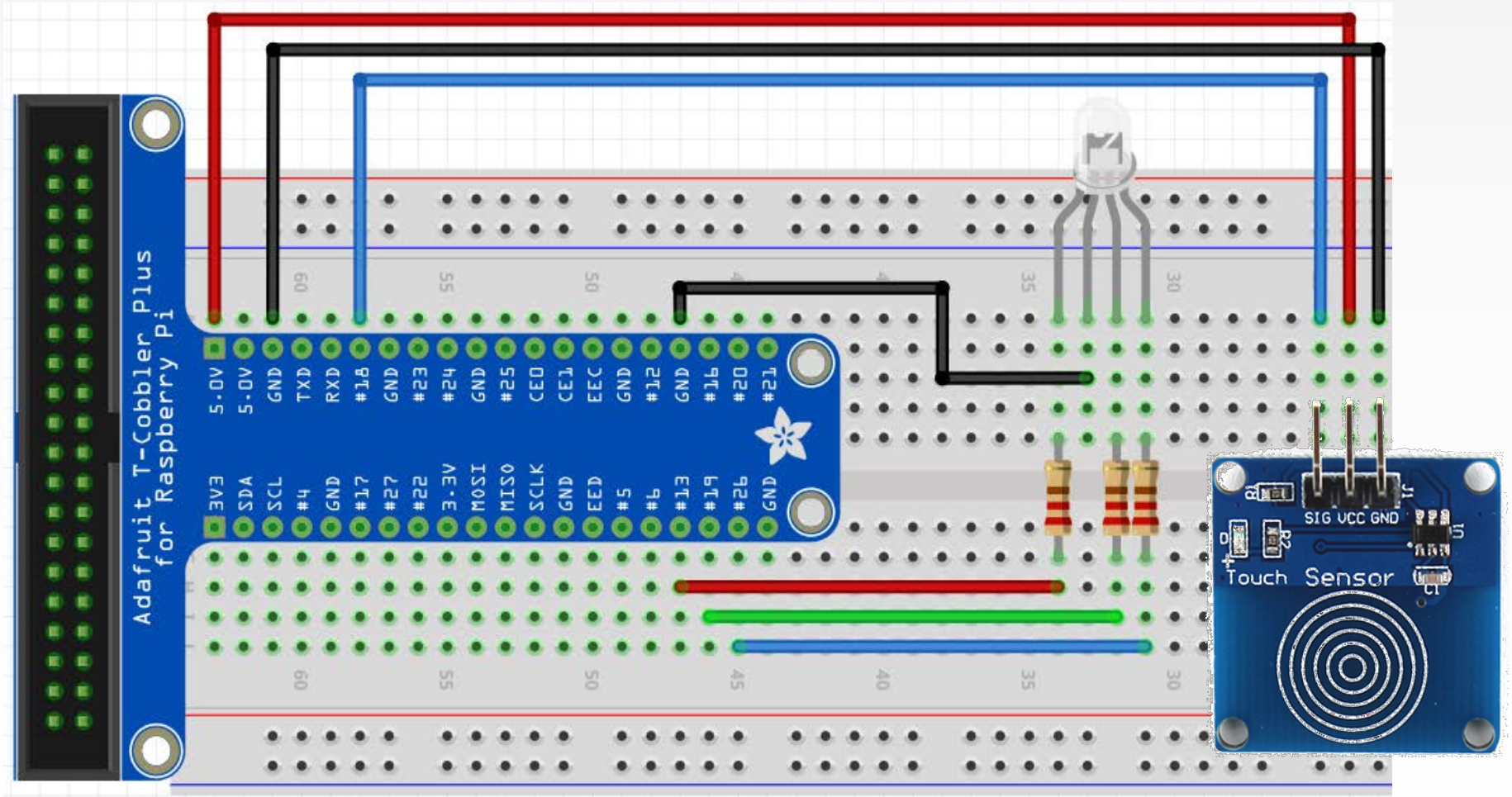
touch sensor

- **touch sensor**
  - 터치 센서는 버튼과 동일한 입력 방식을 가진 입력 기기.
  - VCC는 5V, GND는 GND, SIG는 소스 코드에서 지정된 입력 핀에 연결해서 작동



touch sensor / RGB LED

- 결선



touch sensor / RGB LED

- **소스 코드**
  - 구글 드라이브 Ch.05 폴더에 업로드된 RGB LED_touch.py 파일을 테스트해 보시오.
  - 터치 센서를 터치할 때마다 red, green, blue 중 랜덤한 색상이 출력되는데, 소스 코드에 포함된 magenta, white 등 다른 색상이 출력될 수 있도록 수정해 보시오.
  - GPIO.PWM()을 이용해 그 외의 다른 색상들도 랜덤하게 출력될 수 있도록 수정해 보시오.

리스트(LIST)

- **리스트 자료의 표현 방식**
  - C언어 등의 array(배열)과 유사한 구조이며 서로 다른 type의 자료들을 추가, 삭제, 추출 등 가능
  - 대괄호 안에 , 로 구분해서 자료들을 나열
  - append(), insert(), remove() 등으로 자료를 추가, 삭제할 수 있음


```
L1 = [1, 3.24, 4+5j, 'J', 'LAST']

print(L1[0])    # L1의 0번 자료를 출력
print(L1[1:3])  # L1의 1번 자료, 2번 자료를 출력하고 3번 자료에서 중지
print(L1[4][2]) # L1의 3번 자료의 2번 데이터인 S를 출력
```

```
>>>
==============
1
[3.24, (4+5j)]
S
>>>
```

리스트(LIST)

- 리스트의 자료를 for문으로 추출하기

```
ints = [5,9,17,25]

for i in ints:
    print(i)
```

```
5
9
17
25
```

- 자료를 오른쪽 정렬하기

```
strings = ['a','bc','def','ghijk']

for i in strings:
    print(i.rjust(10))
```

```
         a
        bc
       def
      ghijk
```

튜플(Tuple)

- **튜플 자료의 표현 방식**
  - 리스트와 유사한 구조이나 내용을 변경할 수 없음
  - 소괄호 안에 , 로 구분해서 자료들을 나열


```
L1 = (1, 3.24, 4+5j, 'J', 'LAST')

print(L1[0])    # L1의 0번 자료를 출력
print(L1[1:3])  # L1의 1번 자료, 2번 자료를 출력하고 3번 자료에서 중지
print(L1[4][2]) # L1의 3번 자료의 2번 데이터인 S를 출력
```

```
>>>
==============
1
(3.24, (4+5j))
S
>>>
```

딕셔너리(Dictionary)

- **딕셔너리 자료의 표현 방식**
  - '키':값 의 형태로 자료들을 나열
  - 중괄호 안에 , 로 구분해서 자료들을 나열
  - **딕셔너리 = { '키1':값, '키2':값, '키3':값}**
  - 키는 중복될 수 없으며, 값은 중복 가능

```
d1 = { 'str':'#1a',
       'dec':5,
       'flo':3.14,
       'tup':(3,78,99) }

print(d1)     # d1의 전체 자료를 출력
print(d1['dec'])  # d1의 dec 키의 자료를 출력

print(d1['tup'][2]) # d1의 tup 키의 자료(튜플)의 2번 데이터인 99를 출력
```

```
{'str': '#1a', 'dec': 5, 'flo': 3.14, 'tup': (3, 78, 99)}
5
99
```

현재 시각 출력

- **time.ctime()**
  - 현재 시각을 아래와 같은 문자열 형식으로 반환하는 함수

```
import time
time.ctime()
'Tue Aug  8 18:34:47 2023'
```

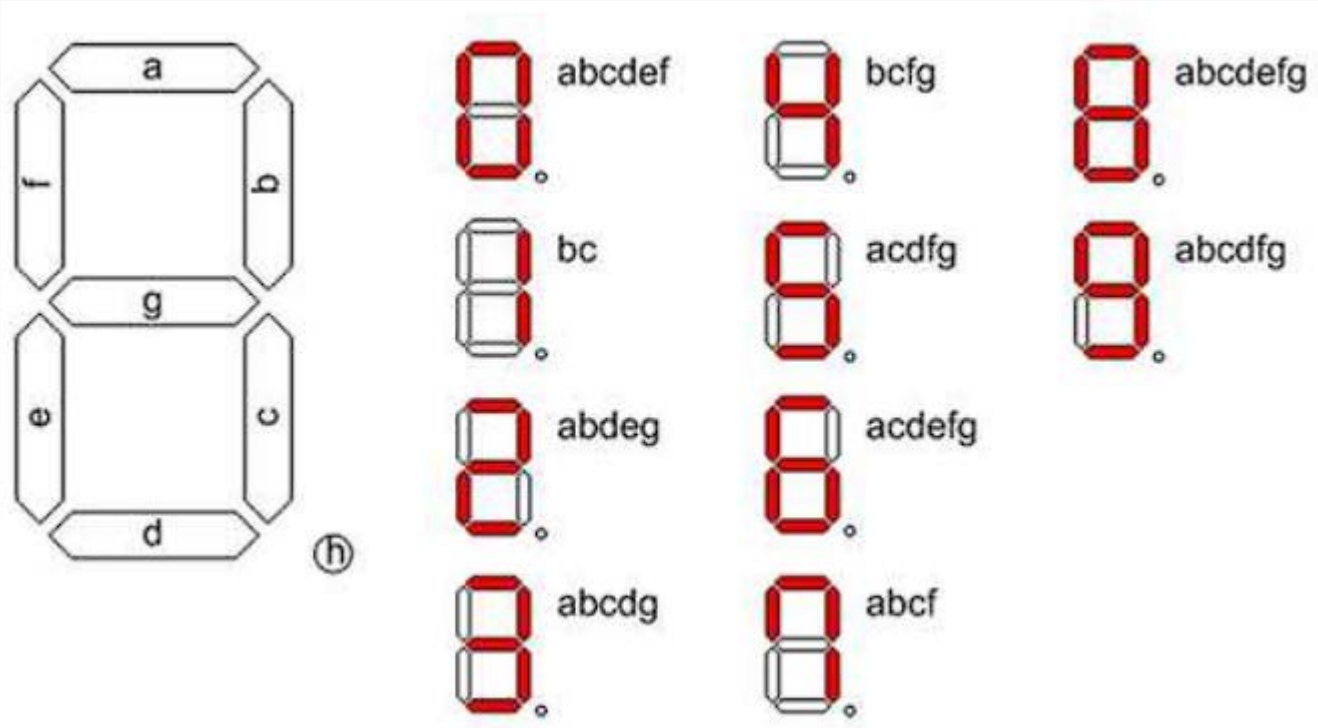
T	u	e		A	u	g			8		1	8	:	3	4	:	4	7		2	0	2	3
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

  - 현재 시, 분을 문자열 형식으로 추출

```
n = time.ctime()[11:13]+time.ctime()[14:16]
n
'1842'
```

FND(Flexible Numeric Display)

- **7-segment display**
  - 7개의 LED에서 발광되는 빛을 이용해서 특정한 숫자나 문자를 표현하는 디스플레이 장치
  - 디지털 시계, 전광판, 공정의 현황판 등에 활용



FND(Flexible Numeric Display)

- **7-segment display**
  - 이전 페이지와 같이 보통 LED의 ON/OFF에 의해 숫자를 표현하는 데 사용되며, 아래와 같이 여러 개의 7-segment를 결합해 여러 digit의 숫자나 문자열을 표현할 수도 있다.

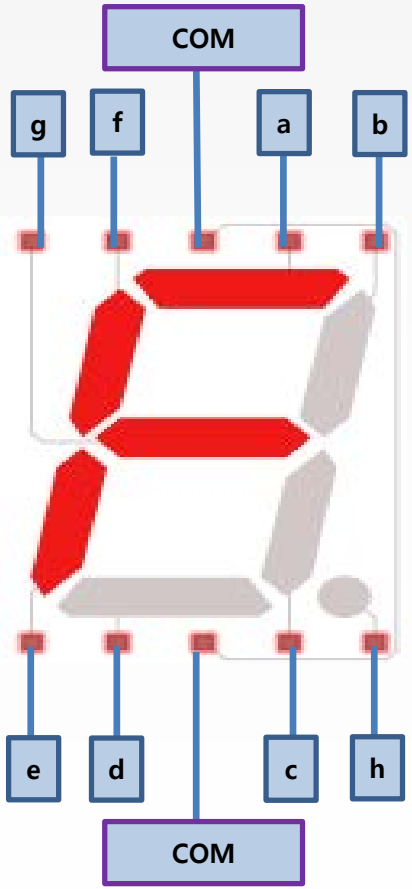
FND(Flexible Numeric Display)

- 7-segment display의 숫자 표시

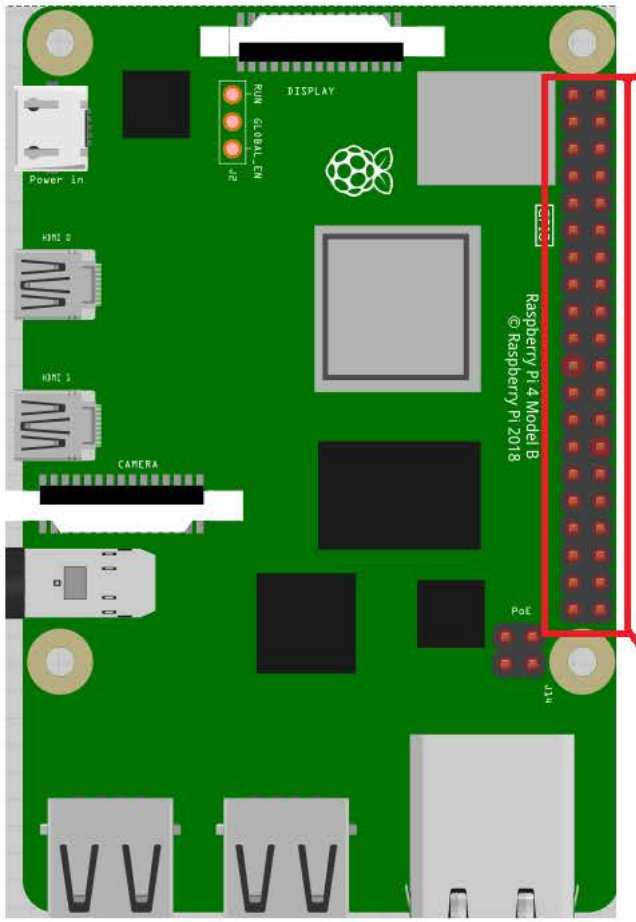
16진수	segment						
	a	b	c	d	e	f	g
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	1	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1
A	1	1	1	0	1	1	1
b	0	0	1	1	1	1	1
C	1	0	0	1	1	1	0
d	0	1	1	1	1	0	1
E	1	0	0	1	1	1	1
F	1	0	0	0	1	1	1

FND(Flexible Numeric Display)

- 7-segment display의 pin definition



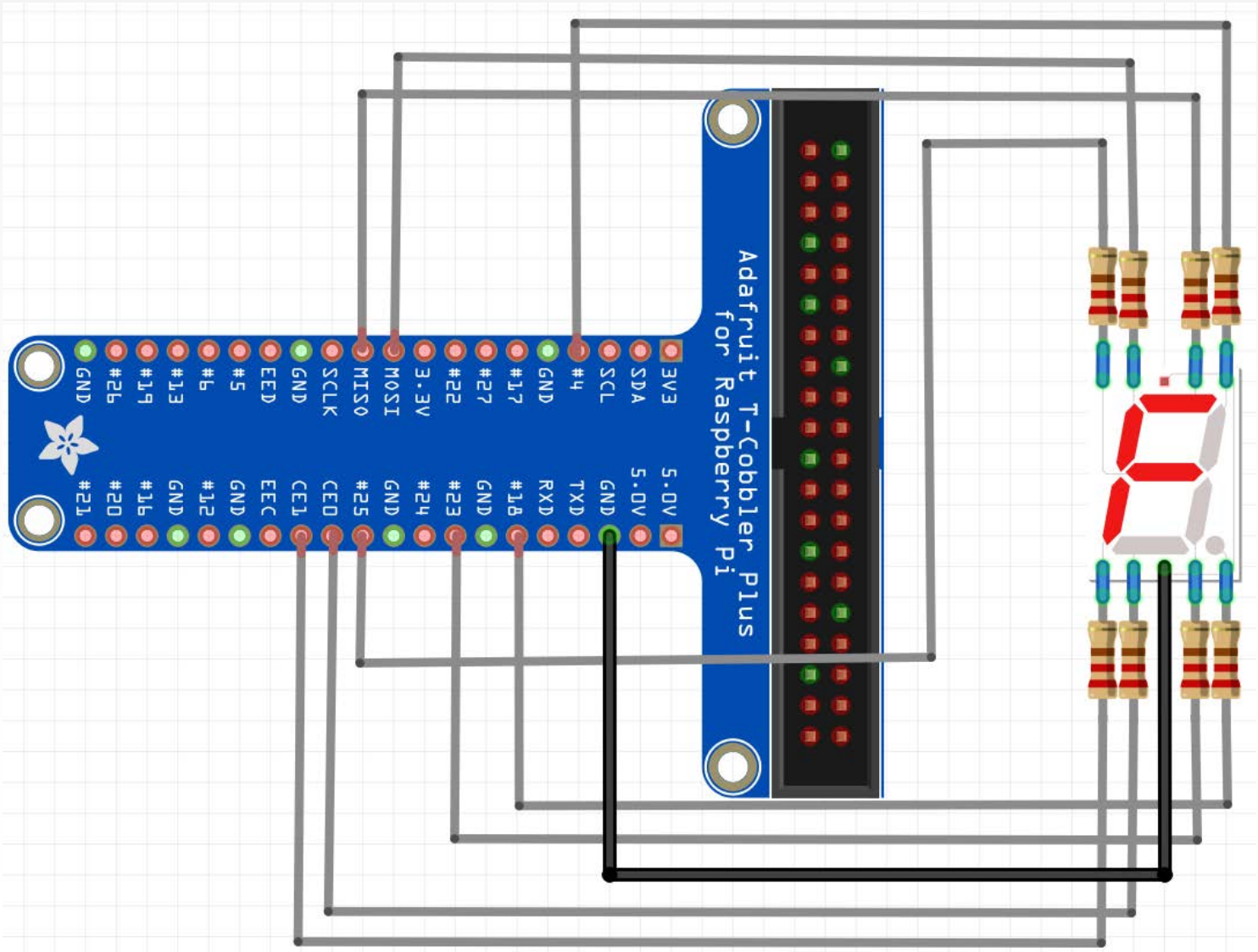
GPIO



wPi	BCM	Pin	No	No	Pin	BCM	wPi
		3.3V	1	2	5V		
GPIO 08	GPIO 02	SDA1	3	4	5V		
GPIO 09	GPIO 03	SCL1	5	6	GND		
GPIO 07	GPIO 04		7	8	TXD	GPIO 14	GPIO 15
		GND	9	10	RXD	GPIO 15	GPIO 16
GPIO 00	GPIO 17		11	12		GPIO 18	GPIO 01
GPIO 02	GPIO 27		13	14	GND		
GPIO 03	GPIO 22		15	16		GPIO 23	GPIO 04
		3.3V	17	18		GPIO 24	GPIO 05
GPIO 12	GPIO 10	MOSI	19	20	GND		
GPIO 13	GPIO 09	MISO	21	22		GPIO 25	GPIO 06
GPIO 14	GPIO 11	SCLK	23	24	CE0	GPIO 08	GPIO 10
		GND	25	26	CE1	GPIO 07	GPIO 11
GPIO 30	GPIO 00	SDA0	27	28	SCL0	GPIO 01	GPIO 31
GPIO 21	GPIO 05		29	30	GND		
GPIO 22	GPIO 06		31	32		GPIO 12	GPIO 26
GPIO 23	GPIO 13		33	34	GND		
GPIO 24	GPIO 19		35	36		GPIO 16	GPIO 27
GPIO 25	GPIO 26		37	38		GPIO 20	GPIO 28
		GND	39	40		GPIO 21	GPIO 29

FND(Flexible Numeric Display)

- 7-segment display 결선



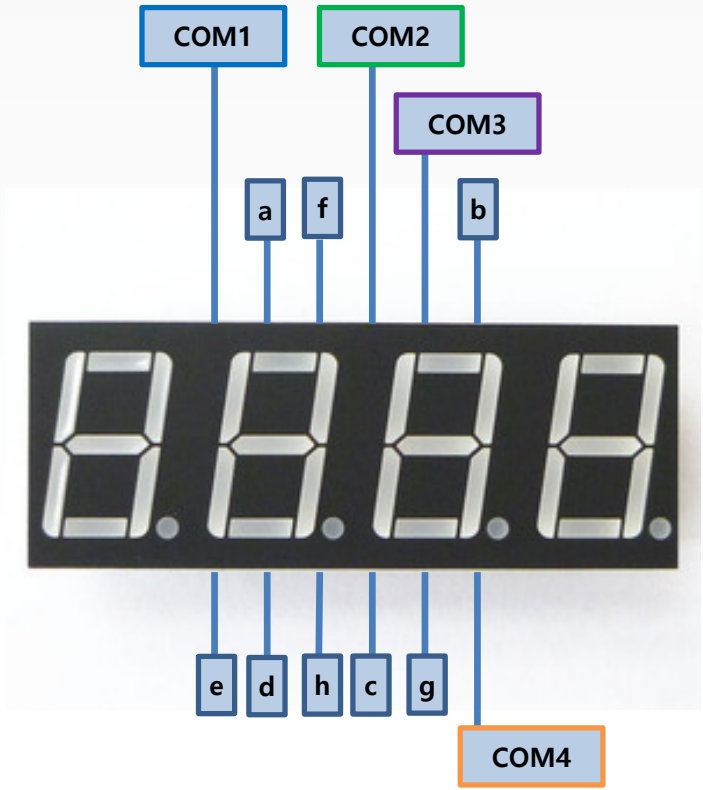
FND(Flexible Numeric Display)

- **7-segment display 소스 코드**
  - 구글 드라이브 Ch.05 폴더에 업로드된 7seg LED.py 파일을 테스트해 보시오.

  - 숫자가 정상적으로 출력된다면 Ch.05 12페이지를 참조하여 0 ~ 9가 출력된 후 'P' 'L' 'A' 'Y' 가 출력되도록 소스 코드를 수정해 보시오.

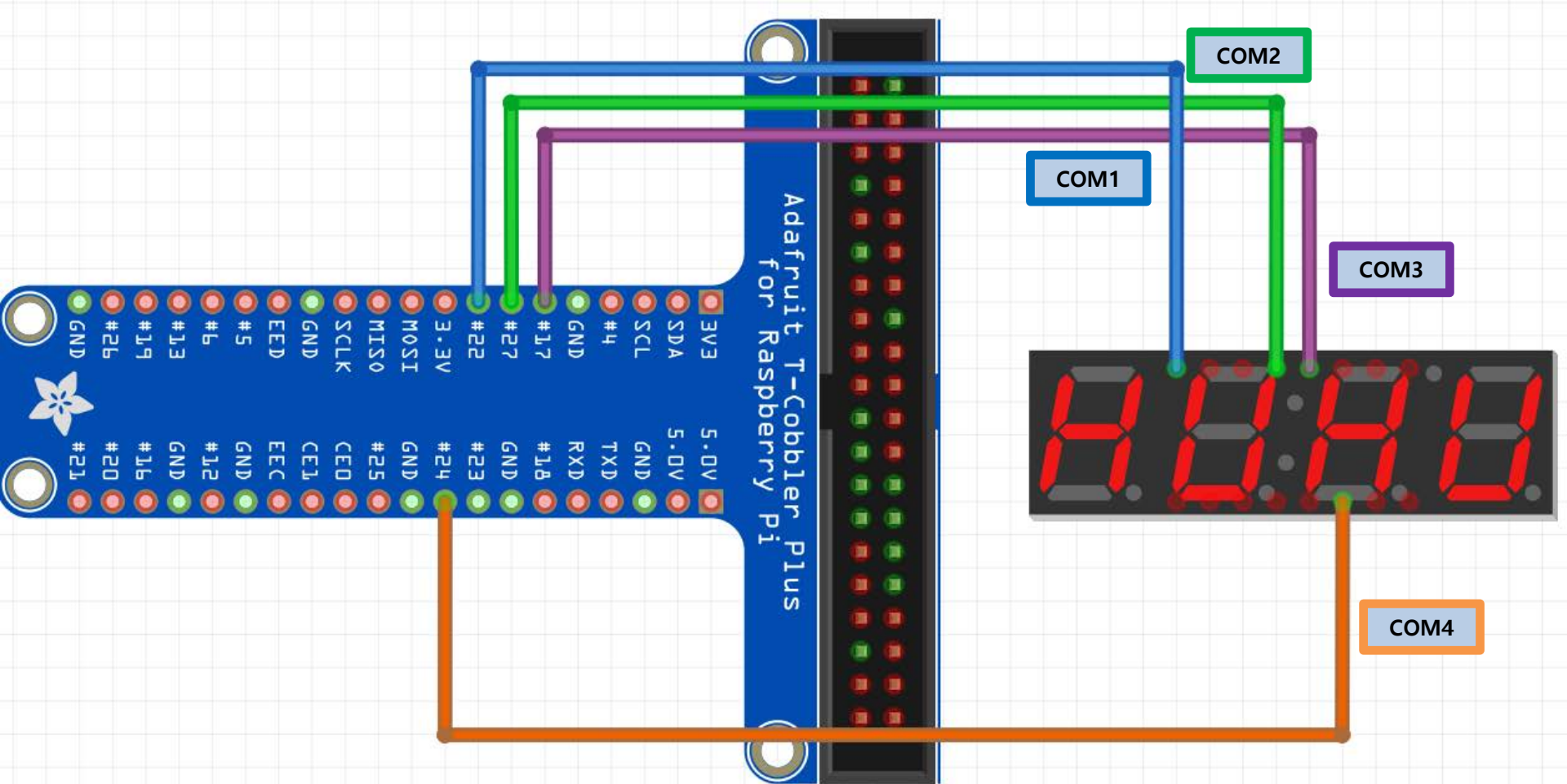
FND(Flexible Numeric Display)

- **static display / dynamic display**
  - 4 digit 7 segment LED에서 4개의 digit 모두를 한꺼번에 제어하는 방식(**static display**)는 사용하는 핀이 지나치게 많고 전류 소모량도 큼
  - 4개의 digit를 눈으로 확인할 수 없을 정도로 빠른 속도로 1개씩 on, off함으로써 사용할 핀의 개수를 줄이고 전류의 크기를 줄일 수 있음(**dynamic display**)



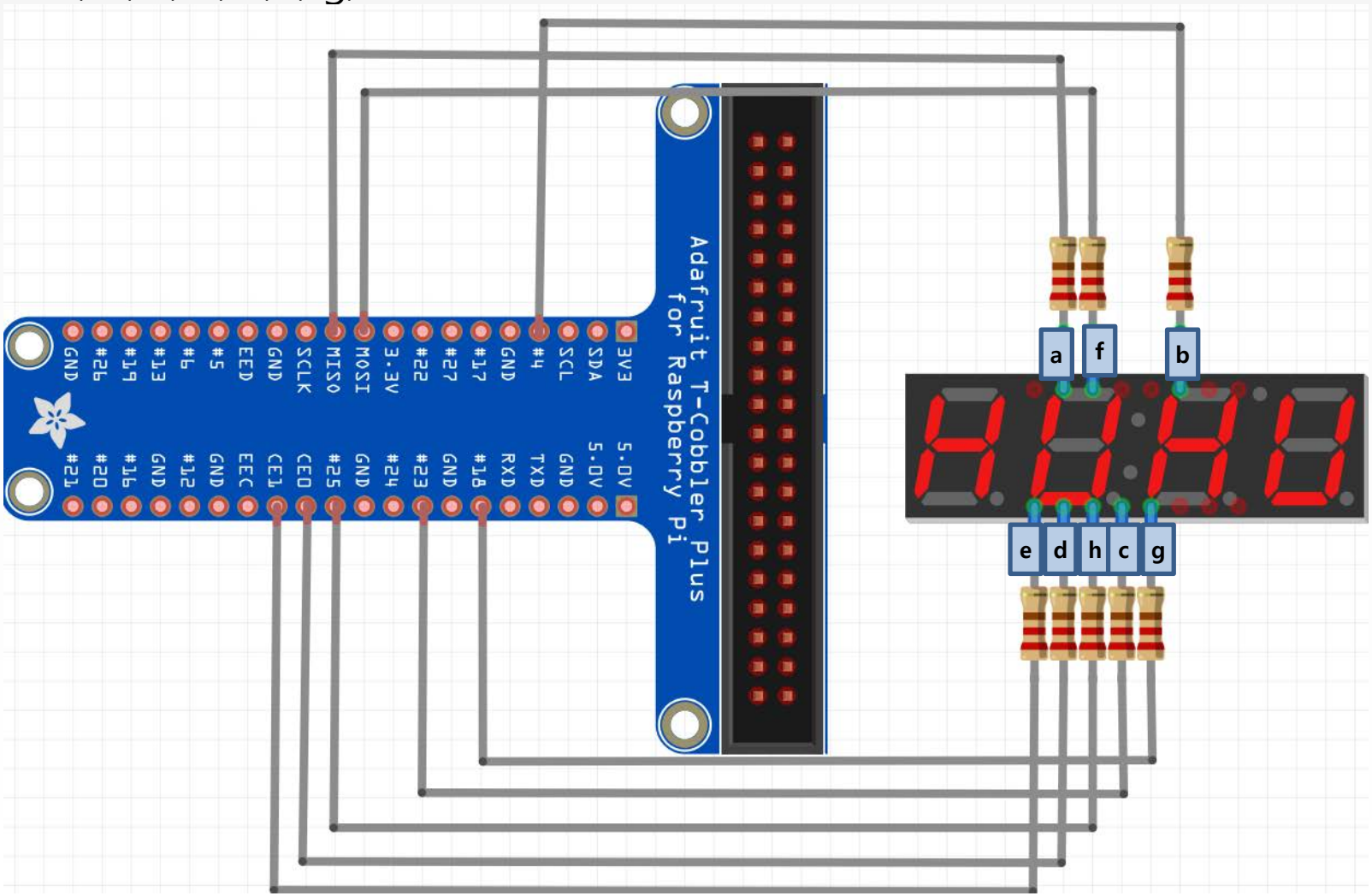
FND(Flexible Numeric Display)

- **4 digit 결선**
  - COM1, COM2, COM3, COM4를 결선



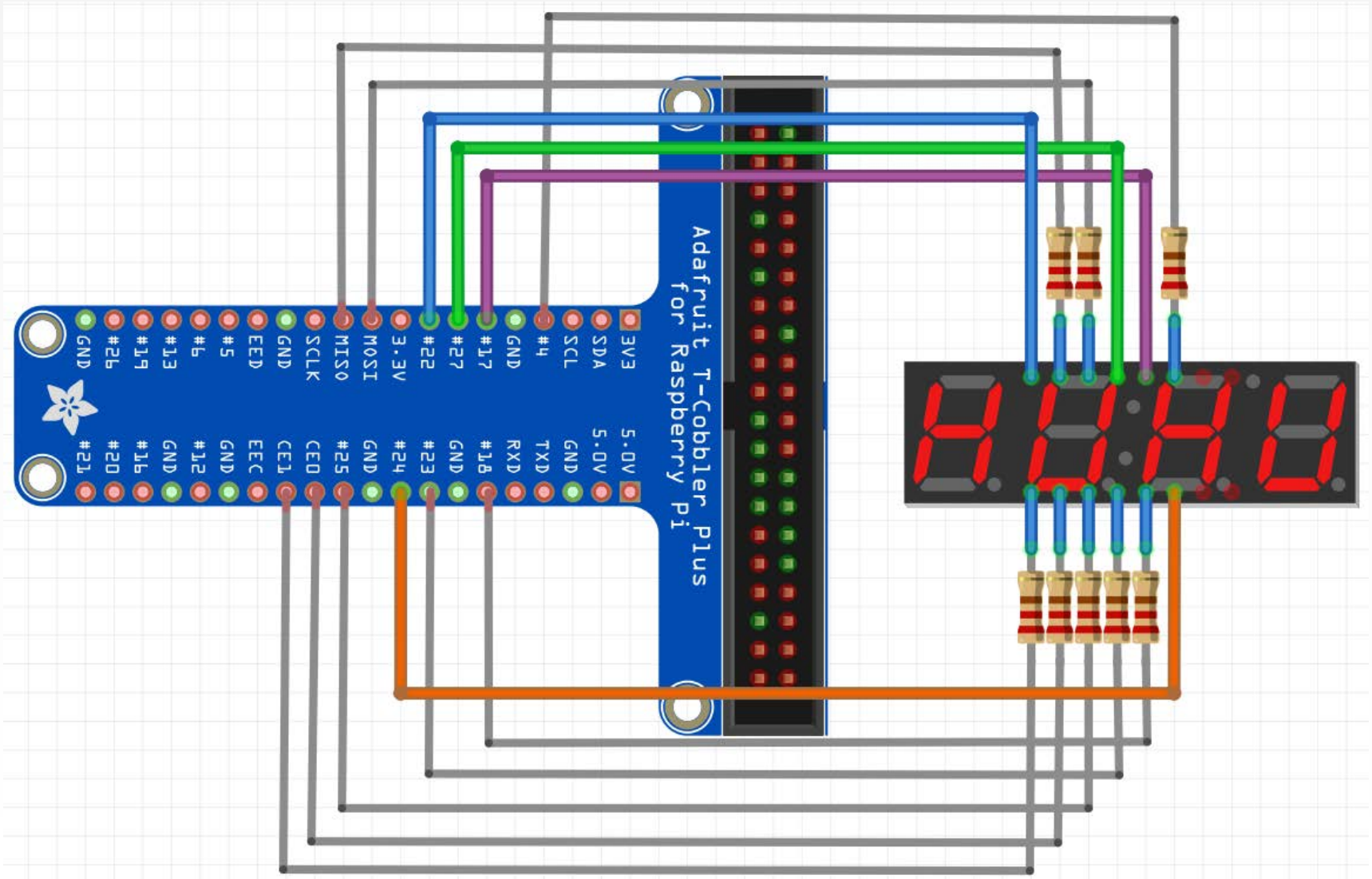
FND(Flexible Numeric Display)

- **7-segment display 결선**
  - a, b, c, d, e, f, g, h를 결선



FND(Flexible Numeric Display)

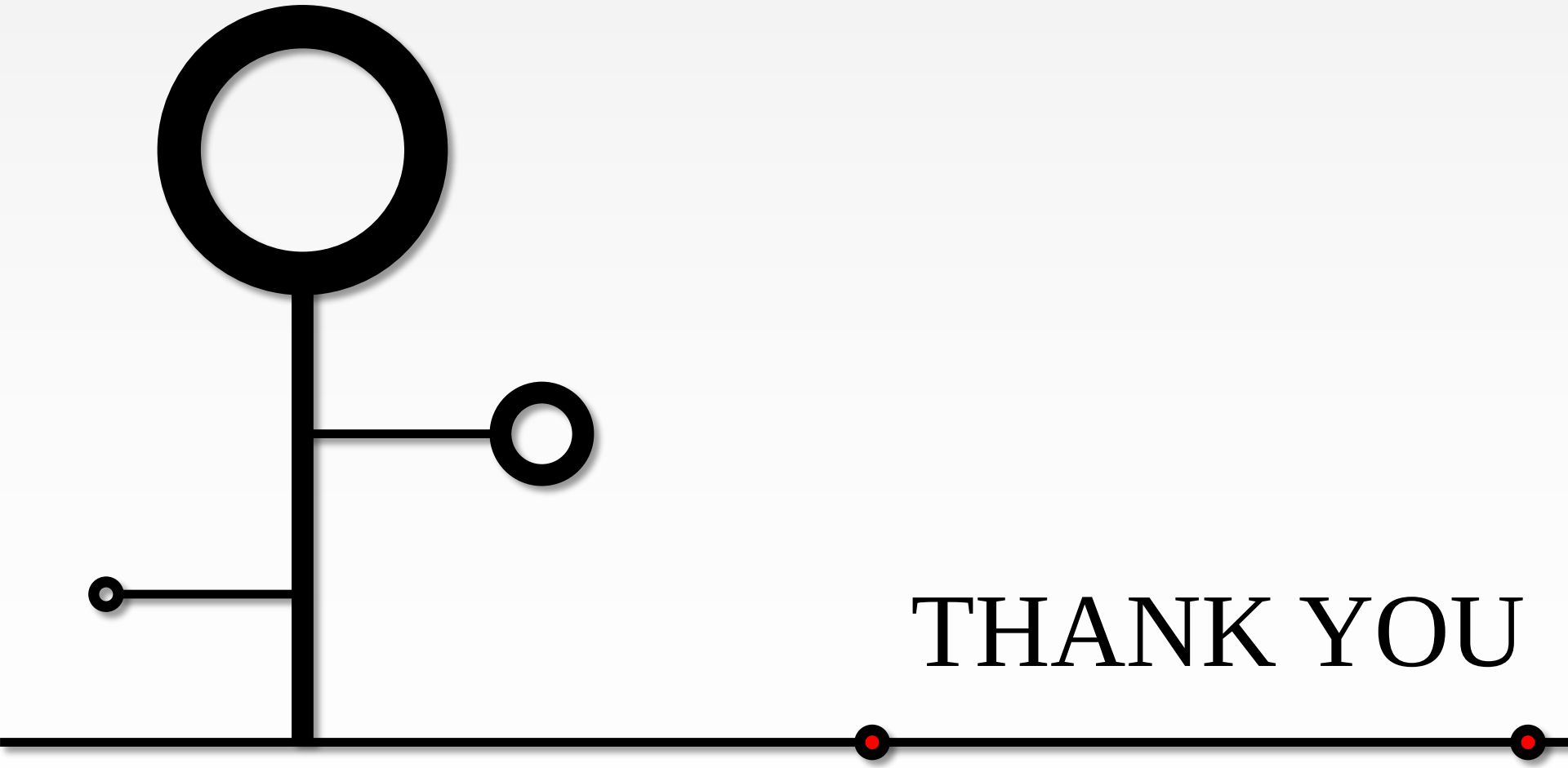
- 4 digit 7-segment display 결선 – 최종 형태



FND(Flexible Numeric Display)

- **4 digit 7-segment display 소스 코드**
  - 구글 드라이브 Ch.05 폴더에 업로드된 4 digit 7seg.py 파일을 테스트해 보시오.
  - 현재의 시각을 HHMM, 즉 시,분 형식으로 출력하며, 시와 분 사이의 digit point 1개가 1초 on, 1초 off됨

  - 4개의 digit point 모두가 1초 on, 1초 off되도록 수정해 보시오.



THANK YOU