



4장

전자 회로 공작

4장 전자 회로 공작

1. 전자 회로 공작의 기본 지식
2. 준비물
3. LED와 스위치
4. 모터 돌리기
5. 카메라 사용하기
6. IC 연결하기 ① (SPI)
7. IC 연결하기 ② (I2C)

1. 전자 회로 공작의 기본 지식

3. 전자 부품의 절대 최대 정격

» 전자 부품의 절대 최대 정격

- 전자 부품의 데이터 시트에는 절대 최대 정격이라는 항목이 있음
- 절대 최대 정격은 ‘순간이라도 넘기게 되면 부품이 고장나는 값’임
- 부품이 고장나지 않도록 이 값을 반드시 지켜야 함
- 예를 들어 LED는 순방향 전류의 절대 최대 정격이 정해져 있음
- LED 전류의 절대 최대 정격이 30mA라면 이 LED에 흐르는 전류는 30mA를 넘지 않아야 함
- 30mA 이상의 전류가 흐르지 않게 하려면 여기서 옴의 법칙이 등장함
- 회로의 전류를 조절하려면 저항 값을 바꾸면 됨
- 어떤 저항 값을 회로에 붙여야 하는지는 앞에서 본 식으로 계산해 알 수 있음
- 전원 전압이 5V라고 하면 다음과 같이 계산할 수 있음

$$R = \frac{V}{I} = 5V \div 0.03A = 166.67\Omega$$

3. 전자 부품의 절대 최대 정격

» 전자 부품의 절대 최대 정격

- 계산 결과에 따라 166.67Ω 보다 큰 저항을 회로에 붙이면 절대 최대 정격을 넘는 전류가 흐르는 것을 막을 수 있음
- 실제로 LED에 흐르는 전류 값을 계산하려면 다른 요소들도 고려해야 함
- 안전하면서도 고장나지 않게 전자 부품을 사용하려면 부품 데이터 시트에 정해진 전류 값과 전압 값 범위 안에서 사용해야 함
- 이때 활약하는 것이 옴의 법칙임











4. 라즈베리 파이로 전자 회로 제어하기

» 라즈베리 파이로 전자 회로 제어하기

- 일반적으로 마이크로프로세서 등은 디지털 회로로 구성
- 다루는 신호도 대부분 디지털 신호임
- 라즈베리 파이로 전자 회로를 제어할 때도 디지털 신호를 출력해서 회로에 명령을 보냄
- 회로에서 디지털 신호를 입력받아 처리함
- 라즈베리 파이에서 디지털 신호를 입출력하려면 확장 커넥터를 사용해야 함
- 확장 커넥터에 다양한 전자 부품을 연결한 후 파이썬 같은 프로그램을 통해 제어할 수 있음

4. 라즈베리 파이로 전자 회로 제어하기

▼ 그림 4-3 라즈베리 파이 확장 커넥터

비고	기능	핀 이름	핀 번호	핀 번호	핀 이름	기능	비고
50mA까지(1번 핀과 17번 핀의 합계)		3.3V	1		2	5V	
1.8kΩ 풀업 저항이 달려 있음	I2C(SDA)	GPIO2	3		4	5V	
1.8kΩ 풀업 저항이 달려 있음	I2C(SCL)	GPIO3	5		6	GND	
	GPCLK0	GPIO4	7		8	GPIO14	UART0(TXD) 부팅할 때 시리얼 콘솔로 사용
		GND	9		10	GPIO15	UART0(RXD) 부팅할 때 시리얼 콘솔로 사용
		GPIO17	11		12	GPIO18	PCM_CLK, PWM0
	PCM_DOUT	GPIO27	13		14	GND	
		GPIO22	15		16	GPIO23	
50mA까지(1번 핀과 17번 핀의 합계)		3.3V	17		18	GPIO24	
	SPI0(MOSI)	GPIO10	19		20	GND	

4. 라즈베리 파이로 전자 회로 제어하기

비고	기능	핀 이름	핀 번호		핀 번호	핀 이름	기능	비고
	SPI0(MISO)	GPIO9	21	⊙ ⊙	22	GPIO25		
	SPI0(SCLK)	GPIO11	23	⊙ ⊙	24	GPIO8	SPI0(CS0)	
		GND	25	⊙ ⊙	26	GPIO7	SPI0(CS1)	
확장 기판(hat)에 탑재된 EEPROM용 신호		ID_SD	27	⊙ ⊙	28	ID_SC		확장 기판(hat)에 탑재된 EEPROM용 신호
		GPIO5	29	⊙ ⊙	30	GND		
		GPIO6	31	⊙ ⊙	32	GPIO12	PWM0	
	PWM1	GPIO13	33	⊙ ⊙	34	GND		
그 외의 기능: PCM_FS	SPI1(MISO), PWM1	GPIO19	35	⊙ ⊙	36	GPIO16	SPI1(CS2)	
		GPIO26	37	⊙ ⊙	38	GPIO20	SPI1(MOSI)	그 외의 기능: PCM_DIN
		GND	39	⊙ ⊙	40	GPIO21	SPI1(SCLK)	그 외의 기능: PCM_DOUT

4. 라즈베리 파이로 전자 회로 제어하기

» 라즈베리 파이로 전자 회로 제어하기

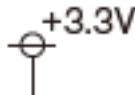



- GND(그라운드) 핀은 전지로 치면 음극임
- 3.3V 핀은 GND 핀에 3.3V 전압을 출력하는 핀으로 전지로 치면 양극이라 전원 핀이 됨
- 마찬가지로 5V 핀은 GND 핀에 5V 전압을 출력하는 전원 핀임
- GPIO 핀은 프로그램에서 제어할 수 있는 핀임
- GPIO(General Purpose Input/Output)는 범용 입출력을 뜻함
- 핀은 프로그램에서 신호를 입력하는 핀(입력 핀) 또는 신호를 출력하는 핀(출력 핀)으로 설정할 수 있음
- GPIO 핀은 디지털 신호만 사용함
- 라즈베리 파이의 확장 커넥터에는 아날로그 신호를 직접 입력하거나 출력할 수 있는 핀이 없음
- 아날로그 신호를 다루려면 전용 IC를 확장 커넥터에 연결해야 함

5. 회로도 기호




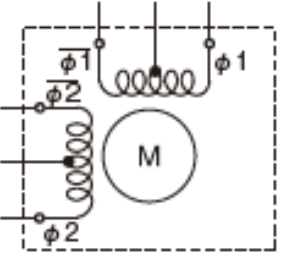
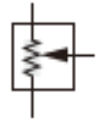
» 회로도 기호

- 전자 회로 공작에서 피할 수 없는 것이 회로도임
- 회로도는 전자 부품 배치와 접속을 표시한 그림으로 전자 회로를 만들 때 매우 중요함

▼ 표 4-1 회로도 기호와 의미

회로도 기호	의미
	전원. 회로 하나에 전원이 여러 개 있다면 옆에 전압을 적어서 구별함
	그라운드. 회로의 기본 전위(0V)를 나타냄(전위차가 전압)
	전지. 선이 긴 쪽이 양극
	저항. 기호 옆에 저항 값을 적음

5. 회로도 기호

회로도 기호	의미
 <p>애노드 캐소드</p>	LED. 삼각형의 밑변이 양극(anode, 애노드), 삼각형의 꼭짓점과 선 쪽이 음극(cathode, 캐소드)
	버튼식 스위치
	DC 모터
	스텝핑 모터
	가변저항 기호 옆에 저항 값을 적음

2. 준비물



1. 이것부터 준비하기

» 이것부터 준비하기

- 라즈베리 파이로 전자 회로 공작을 하려면 우선 브레드보드와 점퍼 와이어를 준비해야 함
- 브레드보드는 전기 부품이나 전선을 연결할 수 있는 구멍이 많이 있는 기판임
- 구멍 안에 전극이 있어서 전자 부품을 꽂는 것만으로도 간단히 전자 회로를 구성할 수 있음

▼ 그림 4-4 브레드보드



1. 이것부터 준비하기

» 이것부터 준비하기

- 점퍼 와이어는 브레드보드에 연결(배선)하는 전용 전선임

▼ 그림 4-5 점퍼 와이어

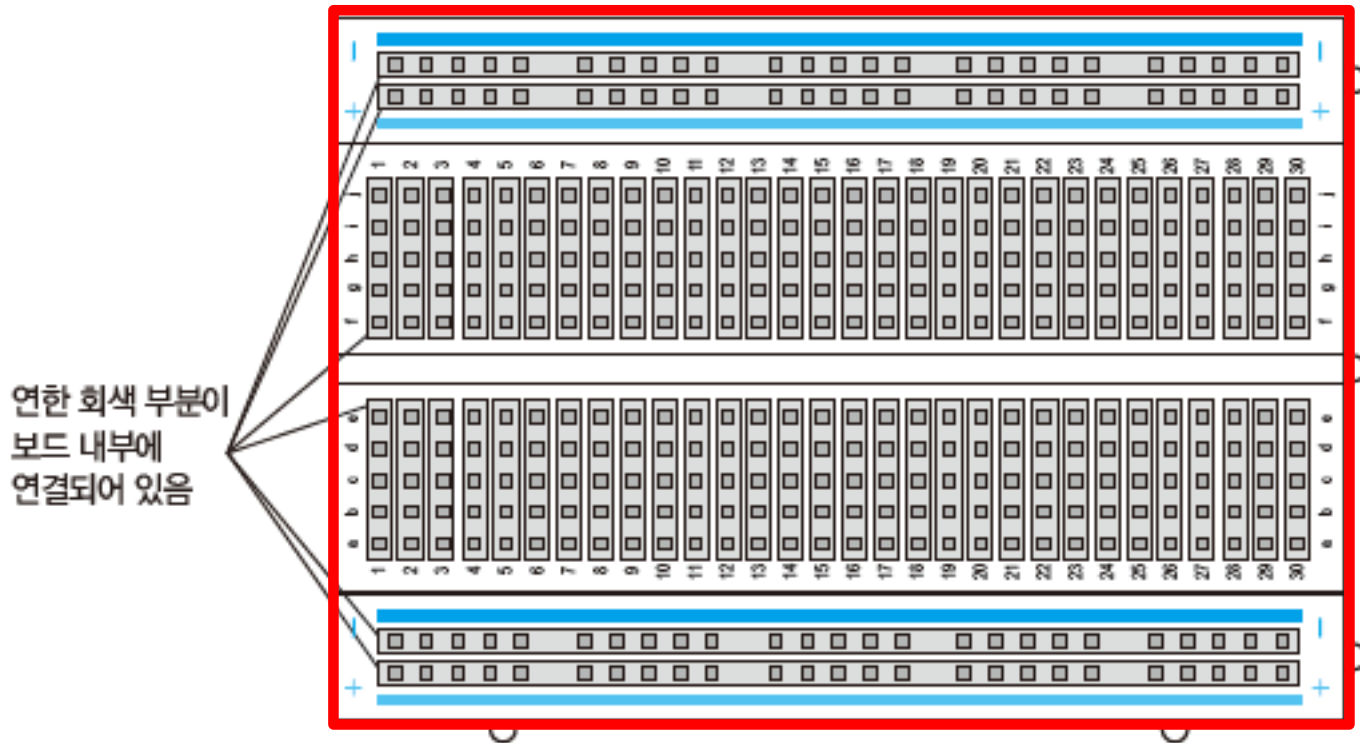


2. 브레드보드의 종류와 구성

» 브레드보드의 종류와 구성

- 브레드보드의 내부 구조를 살펴보면 각 구멍이 다음 그림처럼 연결되어 있음

▼ 그림 4-8 브레드보드의 내부 구조

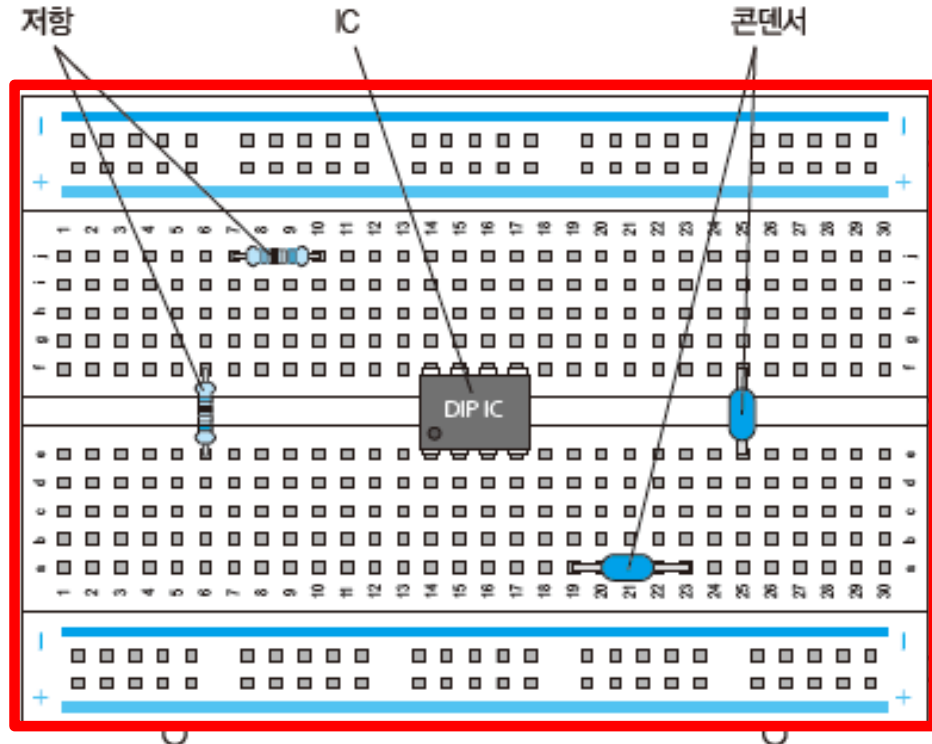


2. 브레드보드의 종류와 구성

» 브레드보드의 종류와 구성

- 저항이나 LED 같은 전자 부품은 다음 그림과 같이 연결함
- IC 등 양쪽에 핀이 연결된 부품은 중간 홈에 걸쳐서 꽂음

▼ 그림 4-9 전자 부품 연결

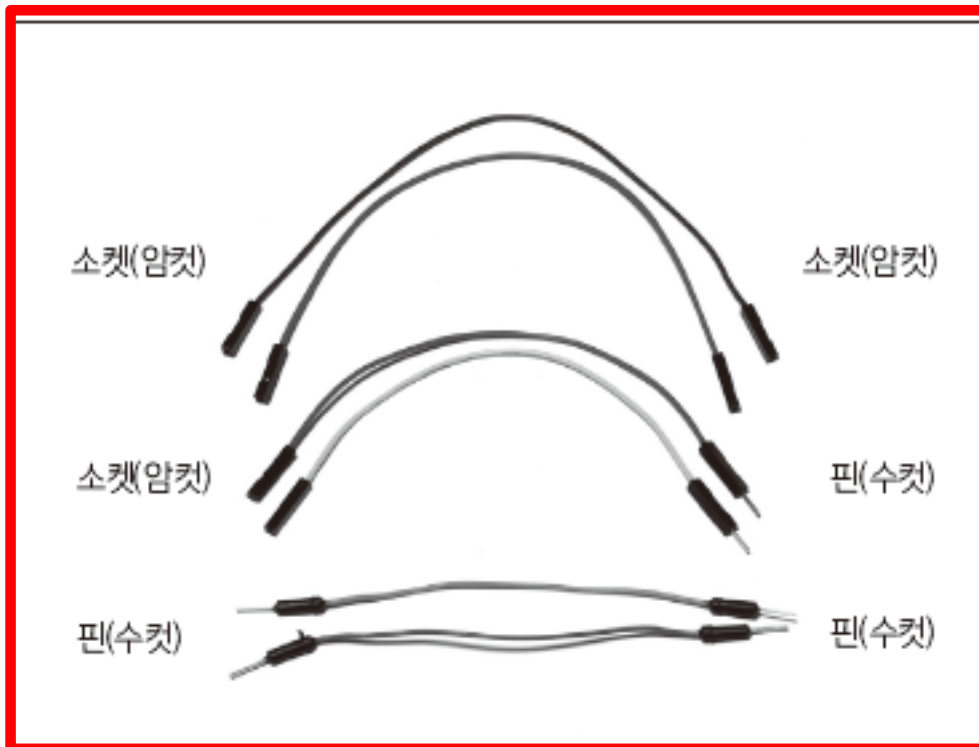


3. 점퍼 와이어의 종류

» 점퍼 와이어의 종류

- 연선 타입은 끝부분이 핀(수컷)인 것과 소켓(암컷)인 것이 있음

▼ 그림 4-11 연선 타입 점퍼 와이어





3. 점퍼 와이어의 종류

» 점퍼 와이어의 종류

- 3절 이후의 회로에서는 브레드보드에 부품끼리 연결할 때는 ‘단선 타입’이나 ‘핀(수컷)-핀(수컷) 형 연선 타입’ 점퍼 와이어를 사용함
- 라즈베리 파이의 확장 커넥터와 브레드보드를 연결할 때는 ‘핀(수컷)-소켓(암컷) 형 연선 타입’ 점퍼 와이어를 사용함

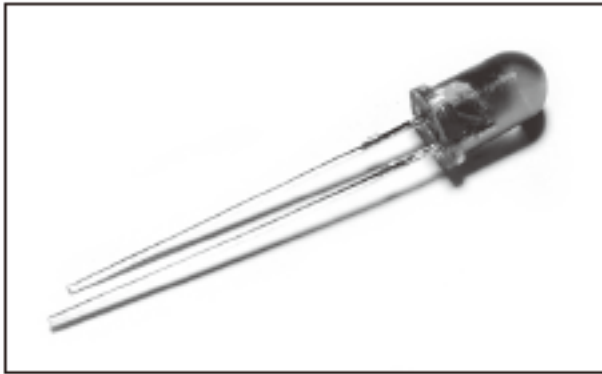
3. LED와 스위치

1. LED란?

» LED란?

- LED(Light Emitting Diode, 발광 다이오드)는 전류를 흐르게 하면 발광하는 반도체 소자임
- 반도체란 전류가 흐르는 방향이 정해져 있는 전자 부품임

▼ 그림 4-17 LED



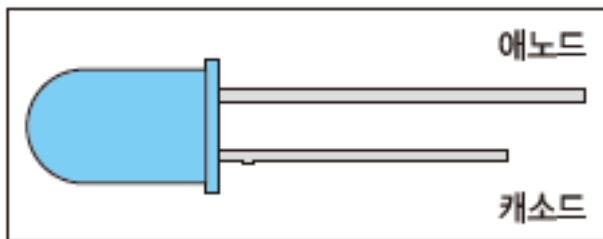


1. LED란?

» LED란?

- LED뿐만 아니라 일반적으로 다이오드라고 부르는 전자 부품에는 애노드(anode, 양극, +극)와 캐소드(cathode, 음극, -극) 단자가 있음
- LED의 애노드에 전원의 +극을 연결하고 캐소드에 -극을 연결하면 전류가 흘러서 LED가 켜짐

▼ 그림 4-18 LED의 애노드 단자와 캐소드 단자



1. LED란?

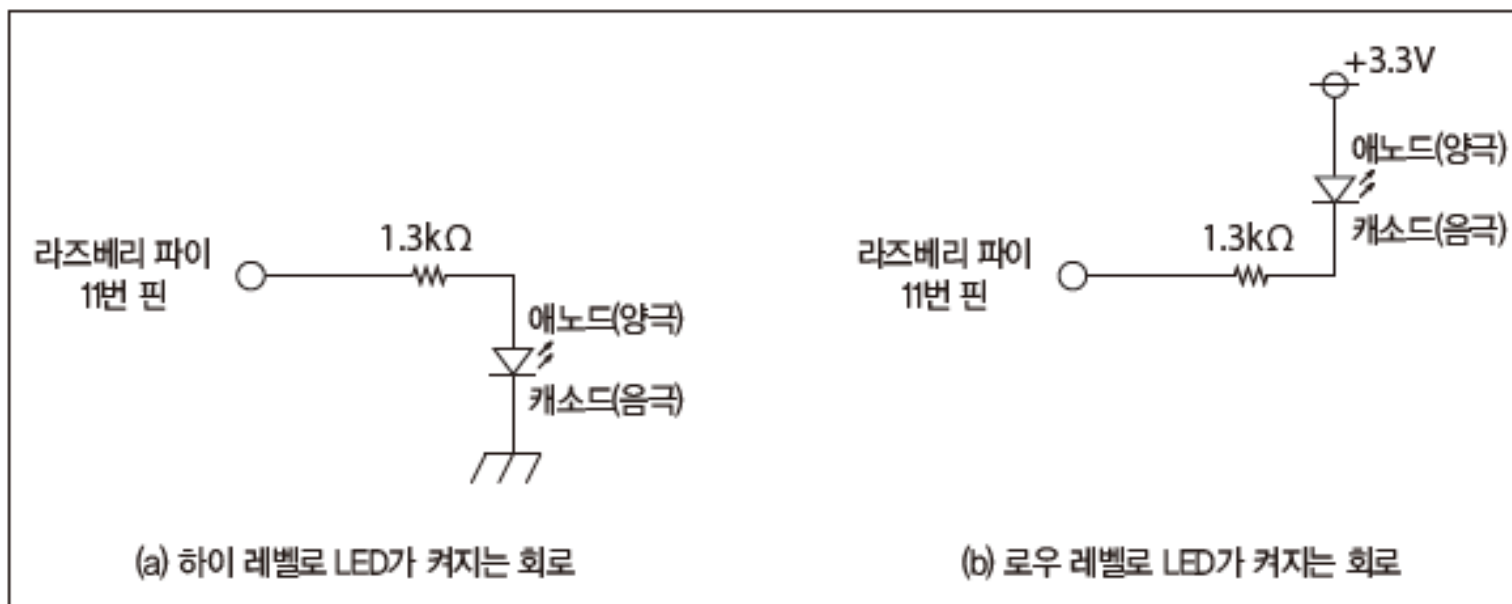


» LED란?

- 데이터 시트의 절대 최대 정격 값(Value)은 단 한순간도 넘겨서는 안 되는 값임
- 순전류 IF는 30mA로 정해져 있어서 흐르는 전류가 30mA를 넘으면 LED가 고장나 버림
- 그림 4-19에서 아래쪽에 있는 '전기적 및 광학적 특성'이 LED의 성능에 관한 부분임
- 이 값의 범위 내에서만 사용하면 고장나는 일 없이 LED의 성능을 최대한 발휘시킬 수 있음
- LED를 사용할 때 중요한 점은 순전압 VF와 광도 IV임
- 순전압(순방향 전압)은 LED 같은 다이오드의 특징으로 애노드(양극)에서 캐소드(음극) 방향으로 전류가 흐를 때 다이오드 양끝에 발생하는 전압을 말함
- 광도란 특정 전류가 흐를 때의 LED 밝기임
- 이 데이터 시트에는 전류 20mA가 흐를 때의 밝기가 적혀 있음
- LED는 이 전류 값보다 작으면 어두워지고 크면 밝아짐
- 전류 값은 절대 최대 정격을 넘을 수 없으니 주의해야 함
- LED는 전류에 따라 켜지므로 라즈베리 파이 프로세서(SoC)가 출력하는 신호에 따라 전류의 유무, 즉 전압의 유무를 제어하면 켜다 켜다 할 수 있음

2. LED 연결하기

▼ 그림 4-20 LED를 연결한 회로





2. LED 연결하기

» LED 연결하기

- 라즈베리 파이가 제어하는 LED 점등 회로에는 두 종류가 있는데, 이 두 회로는 LED가 켜지는 데 필요한 라즈베리 파이의 전압 레벨이 다름
- a 회로는 라즈베리 파이의 11번 핀이 하이 레벨일 때 LED가 켜지고, b 회로는 라즈베리 파이의 11번 핀이 로우 레벨일 때 LED가 켜짐
- a 회로를 사용해 보면 LED에 직접 연결된 저항은 LED에 흐르는 전류 값을 조절함
- 이 저항 값은 LED에 흐르게 할 전류 값을 기준으로 계산함
 - **저항 값 = (전원 전압 - 순전압) ÷ 흐르게 할 전류**

2. LED 연결하기



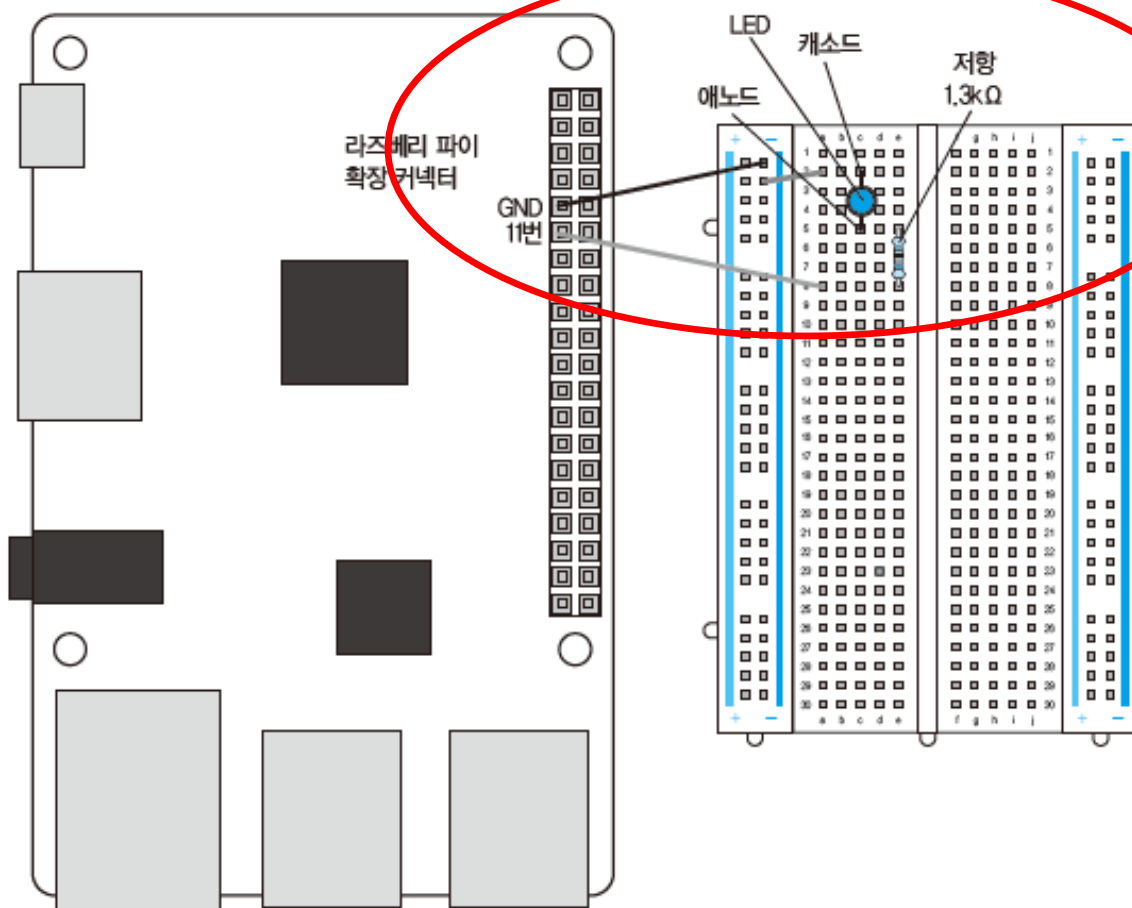
» LED 연결하기

- 라즈베리 파이의 전원 전압은 3.3V임
- 순전압은 LED의 데이터 시트에 적힌 값(이 예에서는 2.0V)이고, 흐르게 할 전류는 1mA
- 라즈베리 파이 프로세서 포트에 흐르게 할 수 있는 전류는 표준으로 8mA까지이므로 이 값을 넘지 않도록 함
- 방금 본 저항 값 식에 대입하면 다음과 같음
$$(3.3 - 2.0) \div 0.001 = 1300 [\Omega]$$
- 필요한 저항은 1.3k Ω 이 됨

2. LED 연결하기

» LED 연결하기

- 이 회로를 브레드보드를 통해 만들어 보면 브레드보드의 실제 배선도는 다음 그림과 같음





3. RPi.GPIO 라이브러리

» RPi.GPIO 라이브러리

- RPi.GPIO 라이브러리는 라즈베리 파이의 확장 커넥터 핀을 범용 입출력(GPIO) 핀으로 제어하기 위한 파이썬 라이브러리임
- 이 라이브러리를 사용하려면 파이썬 프로그램 앞부분에 다음 코드를 추가해야 함

```
import RPi.GPIO as GPIO
```

▼ 표 4-3 RPi.GPIO 라이브러리 속성

속성	설명
RPI_INFO	라즈베리 파이 보드 리비전 등의 정보(이전에 사용하던 RPI_REVISION은 폐기 예정)
VERSION	RPi.GPIO 버전

3. RPi.GPIO 라이브러리



▼ 표 4-4 RPi.GPIO 라이브러리 메서드

메서드	설명	인수	인수 설명
setmode(numbering)	확장 커넥터 핀의 번호 할당 방법을 지정	numbering: GPIO, BOARD 또는 GPIO, BCM	GPIO,BOARD는 커넥터 핀 번호, GPIO,BCM은 CPU 핀 번호(핀 번호는 그림 4-3 참조)
setwarnings(mode)	GPIO 핀이 기본 상태(입력 핀) 외로 설정된 핀을 프로그램에서 변경하려고 할 때 경고를 출력할지 지정	mode: True 또는 False	False로 설정하면 경고를 출력하지 않음
setup(channel, input/output, [pull_up_down 또는 initial])	사용할 GPIO 핀을 설정. GPIO 핀을 사용할 때는 필수	channel: 핀 번호	핀 번호는 setmode 메서드를 통해 설정한 할당 방법으로 지정
		input/output: GPIO, IN 또는 GPIO, OUT	입력이면 GPIO,IN, 출력이면 GPIO,OUT
		pull_up_down: GPIO,PUD_UP 또는 GPIO,PUD_DOWN	입력 핀이면 풀업으로 할지 풀다운으로 할지 지정. 풀업은 GPIO,PUD_UP, 풀다운은 GPIO,PUD_DOWN
		initial: GPIO,HIGH 또는 GPIO,LOW	출력 핀일 때 초기 상태를 지정. GPIO,HIGH는 하이 레벨, GPIO,LOW는 로우 레벨

3. RPi.GPIO 라이브러리



메서드	설명	인수	인수 설명
output(channel, state)	channel로 지정한 핀에 state로 지정한 값을 출력	channel: 핀 번호	핀 번호는 setmode 메서드를 통해 설정한 할당 방법으로 지정
		state: 0/GPIO,LOW/ False 또는 1/GPIO, HIGH/True	출력할 값을 지정. 0/GPIO, LOW/False는 로우 레벨, 1/ GPIO,HIGH/True는 하이레벨
cleanup(channel)	channel에서 사용한 GPIO를 개방. 인수가 없으면 모든 GPIO를 개방. 프로그램을 종료할 때 반드시 실행	channel: 핀 번호	핀 번호는 setmode 메서드를 통해 설정한 할당 방법으로 지정

3. RPi.GPIO 라이브러리

메서드	설명	인수	인수 설명
wait_for_ edge(channel, edge, timeout)	channel로 지정한 핀에 edge로 지정한 이벤트가 발생할 때까지 대기	channel: 핀 번호	핀 번호는 setmode 메서드를 통해 설정한 할당 방법으로 지정
		edge: GPIO.FALLING, GPIO.RISING, GPIO.BOTH	신호가 떨어지거나(GPIO. FALLING), 오르거나(GPIO. RISING), 양쪽(GPIO.BOTH) 중 하나를 지정
		timeout: 타임아웃	이벤트 대기 중 타임아웃할 시간을 지정. 단위는 밀리초 (msec)
add_event_detect (channel, edge, [call back, bouncetime])	channel로 지정한 핀에 edge로 지정한 이벤트가 발생했을 때 callback으 로 지정한 함수를 실행	channel: 핀 번호	핀 번호는 setmode 메서드를 통해 설정한 할당 방법으로 지정
		edge: GPIO.FALLING, GPIO.RISING, GPIO.BOTH	신호가 떨어지거나(GPIO. FALLING), 오르거나(GPIO. RISING), 양쪽(GPIO.BOTH) 중 하나를 지정
		callback: 함수	이벤트 발생 시 호출할 함수를 지정
		bouncetime: 대기 시간	채터링*을 제거하기 위해 대기 시간을 밀리초(msec)로 지정

3. RPi.GPIO 라이브러리

메서드	설명	인수	인수 설명
event_detected (channel)	미리 add_event_detect 메서드로 지정한 channel 에 지정한 이벤트가 발생하 면 True를 반환	channel: 핀 번호	핀 번호는 setmode 메서드를 통해 설정한 할당 방법으로 지정
add_event_callback (channel, callback[, bouncetime])	미리 add_event_detect 로 지정한 channel에 지 정한 이벤트가 발생하면 함 수 callback 실행을 추가함	channel: 핀 번호	핀 번호는 setmode 메서드를 통해 설정한 할당 방법으로 지정
		callback: 함수	이벤트 발생 시 호출할 함수를 지정
		bouncetime: 대기 시간	채터링을 제거하기 위해 대기 시간을 밀리초(msec)로 지정
remove_event_ detect(channel)	channel에 지정한 핀의 이벤트 검출을 정지시킴	channel: 핀 번호	핀 번호는 setmode 메서드를 통해 설정한 할당 방법으로 지정



4. 파이썬 프로그램(LED 깜빡이기)

▼ 코드 4-1 LED_01.py

```
# coding: utf-8

# GPIO 라이브러리 импорт
import RPi.GPIO as GPIO

# time 라이브러리 импорт
import time

# 핀 번호 할당 방법은 커넥터 핀 번호로 설정
GPIO.setmode(GPIO.BOARD)

# 사용할 핀 번호 할당
LED = 11

# 11번 핀을 출력 핀으로 설정, 초기 출력은 로우 레벨
GPIO.setup(LED, GPIO.OUT, initial=GPIO.LOW)
```


4. 파이썬 프로그램(LED 깜빡이기)

```
# 예외 처리
try:

    # 무한 반복
    while 1:

        # 하이 레벨 출력
        GPIO.output(LED, GPIO.HIGH)

        # 0.5초 대기
        time.sleep(0.5)

        # 로우 레벨 출력
        GPIO.output(LED, GPIO.LOW)
```



4. 파이썬 프로그램(LED 깜빡이기)

```
# 0.5초 대기
time.sleep(0.5)

# 키보드 예외를 검출
except KeyboardInterrupt:

    # 아무것도 안 함
    pass

# GPIO 개방
GPIO.cleanup()
```

4. 파이썬 프로그램(LED 깜빡이기)

» 파이썬 프로그램(LED 깜빡이기)

- 이 프로그램은 LED를 연결한 11번 GPIO 핀의 출력 신호를 무한 반복에서 하이 레벨과 로우 레벨 반복으로 변경
- 무한 반복을 하므로 프로그램을 정지하는 처리가 필요함
- try ~ except 문으로 예외 처리를 하는데 except에 KeyboardInterrupt를 지정해 키보드에서 끼어들기 입력이 들어오는지 확인
- 마지막에 GPIO.cleanup()을 실행해서 GPIO를 개방시키는 걸 주목
- 프로그램을 실행하면 0.5초 간격으로 LED가 켜졌다 꺼졌다 함
- 프로그램을 종료하려면 셸을 실행한 상태에서 Ctrl + C 키를 누르거나 Thonny의 Interrupt/Reset 버튼을 누름



4. 파이썬 프로그램(LED 깜빡이기)

» 파이썬 프로그램(LED 깜빡이기)

- 지금부터는 화면의 버튼을 누를 때마다 LED가 켜지거나 꺼지는 GUI 프로그램을 만들어 보자

▼ 코드 4-2 tk_LED_01.py

```
# coding: utf-8

# GPIO 라이브러리 импорт
import RPi.GPIO as GPIO

# Tkinter 라이브러리 импорт
import tkinter as tk # Python3
# import Tkinter as tk # Python2
```



4. 파이썬 프로그램(LED 깜빡이기)

```
# 핀 번호 할당 방법은 커넥터 핀 번호로 설정
GPIO.setmode(GPIO.BOARD)

# 사용할 핀 번호 할당
LED = 11

# 11번 핀을 출력 핀으로 설정, 초기 출력은 로우 레벨
GPIO.setup(LED, GPIO.OUT, initial=GPIO.LOW)

# LED를 켜고 끄는 함수를 정의
def func():

    # 11번 핀에서 나오는 입력 값을 반전해서 출력
    GPIO.output(LED, not GPIO.input(LED))
```



4. 파이썬 프로그램(LED 깜빡이기)

```
# Tk 객체 인스턴스 root 작성
root = tk.Tk()

# root에 표시할 레이블 정의
label = tk.Label(root, text='press button')

# 레이블 배치
label.pack()

# root에 표시할 버튼 정의
button = tk.Button(root, text='LED', command=func)

# 버튼 배치
button.pack()

# root 표시
root.mainloop()

# GPIO 개방
GPIO.cleanup()
```

4. 파이썬 프로그램(LED 깜빡이기)

» 파이썬 프로그램(LED 깜빡이기)

- root 창에 레이블과 버튼을 붙인 후 버튼이 눌리면 LED를 켜거나 끄는 func() 함수를 호출함
- LED 상태를 전환하려면 이 프로그램처럼 우선 현재 GPIO의 상태를 읽고, 그 값을 논리 연산자 not으로 반전시켜서 GPIO에 출력하면 됨
- 디지털 신호라면 받는 값이 0(로우 레벨)과 1(하이 레벨)뿐이므로 not 연산자를 통해 지금 상태의 반전 값을 만듦
- 프로그램을 실행하면 다음과 같은 창이 표시되고 LED 버튼을 누르면 LED가 켜지고, LED가 켜진 상태에서 다시 버튼을 누르면 꺼짐

▼ 그림 4-22 tk_LED_01.py 실행 결과

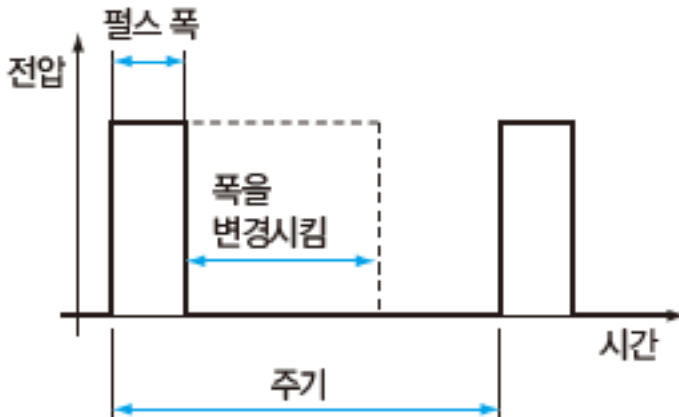


5. LED 밝기 조절하기

» LED 밝기 조절하기

- LED의 밝기를 조절하려면 PWM(Pulse Width Modulation, 펄스 폭 변조) 신호를 사용함
- PWM이란 펄스 폭 변조라고 함
- 일정 간격으로 신호의 하이 레벨과 로우 레벨의 폭을 전환해서 원래 디지털 신호에서라면 표현할 수 없는 하이 레벨과 로우 레벨의 중간값을 유사하게 표현하는 신호 방식임

▼ 그림 4-23 PWM 신호



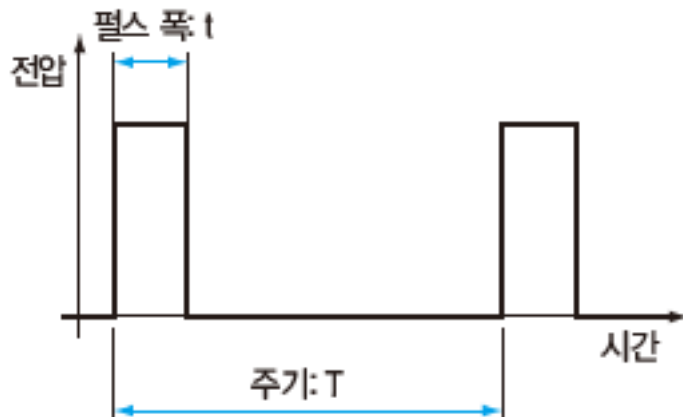


5. LED 밝기 조절하기

» LED 밝기 조절하기

- LED를 하이 레벨로 켜는 회로라면 하이 레벨의 시간(펄스 폭)이 짧을수록 LED가 어둡게 켜지고 길수록 밝게 켜짐
- 이때 PWM 신호 주기에 대한 펄스 폭 비율을 듀티 비(duty ratio) 또는 듀티라 함

▼ 그림 4-24 듀티 비



$$\text{듀티 비: } D = \frac{t}{T}$$



5. LED 밝기 조절하기

» LED 밝기 조절하기

- 파이썬에서 PWM 신호를 출력하기 위해 RPi.GPIO 라이브러리를 임포트함
- 다음과 같이 작성해서 PWM 객체 인스턴스를 생성함

```
import RPi.GPIO as GPIO  
  
p = GPIO.PWM(channel, frequency)
```

5. LED 밝기 조절하기

» LED 밝기 조절하기

- 인수 channel에는 PWM 신호를 출력할 핀 번호를 지정함
- frequency에는 PWM 신호의 주파수를 지정함

▼ 표 4-5 PWM 객체의 메서드

메서드	설명	인수	인수 설명
start(dc)	PWM 신호 출력	dc: 듀티 비	듀티 비를 부동소수로 지정(0.0~100.0%)
ChangeFrequency(freq)	PWM 신호의 주파수 변경	freq: 주파수	주파수(Hz) 지정
ChangeDutyCycle(dc)	PWM 신호의 듀티 비 변경	dc: 듀티 비	듀티 비를 부동소수로 지정(0.0~100.0%)
stop()	PWM 신호 정지	없음	



6. 파이썬 프로그램(PWM 신호)

» 파이썬 프로그램(PWM 신호)

- LED가 꺼진 상태에서 완전히 켜진 상태가 될 때까지 점점 밝아졌다가 다시 점점 어두워지는 프로그램을 만들어 보자

▼ 코드 4-3 LED_02.py

```
# coding: utf-8

# GPIO 라이브러리 импорт
import RPi.GPIO as GPIO

# time 라이브러리 импорт
import time

# 핀 번호 할당 방법은 커넥터 핀 번호로 설정
GPIO.setmode(GPIO.BOARD)

# 사용할 핀 번호 할당
LED = 11
```



6. 파이썬 프로그램(PWM 신호)

```
# 듀티 비 목록 작성
dc = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 15, 20, 30, 50, 70, 100]

# 11번 핀을 출력 핀으로 설정, 초기 출력은 로우 레벨
GPIO.setup(LED, GPIO.OUT, initial=GPIO.LOW)

# PWM 객체 인스턴스 작성
# 출력 핀: 11번, 주파수: 100Hz
p = GPIO.PWM(LED, 100)

# PWM 신호 출력
p.start(0)

# 예외 처리
try:
```



6. 파이썬 프로그램(PWM 신호)

```
# 무한 반복
while 1:
    # 듀티 비를 0부터 100까지 목록에 따라 변경
    for val in dc:
        # 듀티 비 설정
        p.ChangeDutyCycle(val)
        # 0.1초 대기
        time.sleep(0.1)
    # 목록 나열 순서를 역순으로 변경
    dc.reverse()
    # 0.1초 대기
    time.sleep(0.1)

# 키보드 예외 검출
except KeyboardInterrupt:
    # 아무것도 안 함
    pass
```

6. 파이썬 프로그램(PWM 신호)



```
# PWM 정지  
p.stop()  
  
# GPIO 개방  
GPIO.cleanup()
```



6. 파이썬 프로그램(PWM 신호)

» 파이썬 프로그램(PWM 신호)

- 리스트형 변수 dc에 듀티 비 목록을 할당해서 for 반복으로 듀티 비를 순서대로 추출함
- 그에 따라 PWM 신호를 출력해서 LED를 켜
- 이때 LED는 점점 밝아짐
- for 반복을 빠져나오면 reverse() 메서드를 써서 목록 순서를 반대로 만듦
- 다시 for 반복으로 들어가면 이번에는 LED가 점점 어두워짐
- 프로그램을 종료하려면 Thonny를 활성화한 상태에서 Ctrl + C 키를 누르거나 Interrupt/Reset 버튼을 누름

6. 파이썬 프로그램(PWM 신호)

» 파이썬 프로그램(PWM 신호)

- 다음은 슬라이더로 LED 밝기를 조절하는 GUI 프로그램을 만들어 보자

▼ 코드 4-4 tk_LED_02.py

```
# coding: utf-8

# GPIO 라이브러리 импорт
import RPi.GPIO as GPIO

# Tkinter 라이브러리 импорт
import tkinter as tk # Python3
# import Tkinter as tk # Python2

# 핀 번호 할당 방법은 커넥터 핀 번호로 설정
GPIO.setmode(GPIO.BOARD)

# 사용할 핀 번호 할당
LED = 11

# 11번 핀을 출력 핀으로 설정, 초기 출력은 로우 레벨
GPIO.setup(LED, GPIO.OUT, initial=GPIO.LOW)
```



6. 파이썬 프로그램(PWM 신호)

```
# PWM 객체 인스턴스 생성
# 출력 핀: 11번, 주파수: 100Hz
p = GPIO.PWM(LED, 100)

# Tk 객체 인스턴스 root 작성
root = tk.Tk()

# 슬라이더 값으로 사용할 Variable 객체 인스턴스를 부동소수로 작성
led_val = tk.DoubleVar()
# 0 지정
led_val.set(0)

# PWM 신호 출력
p.start(0)

# 듀티 비를 변경하는 함수 정의
def change_duty(dc):
```



6. 파이썬 프로그램(PWM 신호)

```
# 듀티 비 변경
p.ChangeDutyCycle(led_val.get())

# root에 표시할 슬라이더 정의
# 레이블 LED, 수평으로 표시, 숫자 범위는 0~100
s = tk.Scale(root, label = 'LED', orient = 'h', \
             from_ = 0, to = 100, variable = led_val,
             command = change_duty)

# 슬라이더 배치
s.pack()

# root 표시
root.mainloop()

# PWM 정지
p.stop()

# GPIO 개방
GPIO.cleanup()
```



6. 파이썬 프로그램(PWM 신호)

» 파이썬 프로그램(PWM 신호)

- 이 프로그램에서는 슬라이더를 표시하기 위해 Scale 객체를 사용함
- 슬라이더를 움직이면 슬라이드 탭의 위치에 따라 값이 변하므로 get() 메서드로 그 값을 얻어서 부동소수
- 형 Variable의 객체 led_val에 할당함
- 그 값을 듀티 비를 변경하는 change_duty 메서드에 넘겨서 LED 밝기를 바꿈
- 프로그램을 실행하면 다음과 같은 창이 표시됨
- 슬라이더의 슬라이드 탭을 움직여 LED 밝기가 변하는지 확인해 보기 바람

▼ 그림 4-25 tk_LED_02.py 실행 결과



7. 스위치란?

» 스위치란?

- 다음은 입력 회로의 기본인 스위치임
- 스위치는 켜거나 끄는 조작을 라즈베리 파이의 전자 회로에 전달하는 부품임
- 스위치 종류는 형태나 목적에 따라 다양함

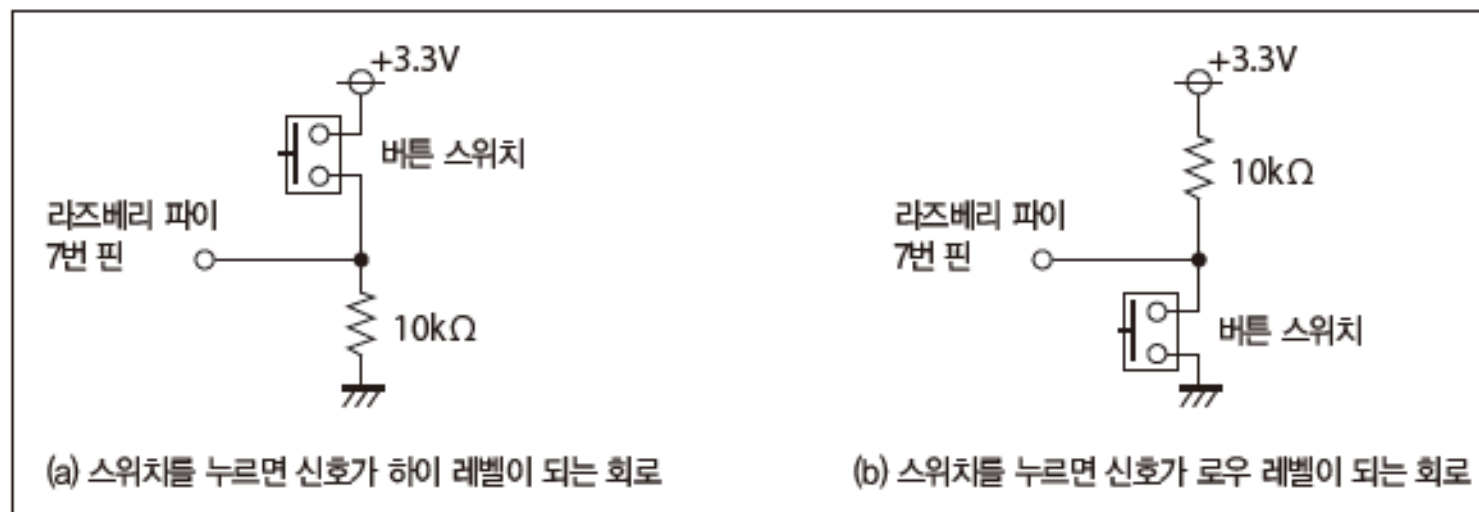
▼ 표 4-6 스위치 종류

버튼 스위치	토글 스위치	슬라이드 스위치
		
버튼을 누르면 접점이 연결되는 스위치. 누르는 동안 연결됨	레버를 쓰러트리면 접속이 전환되는 스위치. 스위치에서 손을 떼도 상태는 유지됨	스위치를 슬라이드해서 접촉을 바꾸는 스위치. 스위치에서 손을 떼도 상태는 유지됨

- 여기서는 버튼을 누를 때만 전기가 통하는 버튼 스위치를 사용

8. 스위치 연결하기

▼ 그림 4-26 스위치 연결 회로



8. 스위치 연결하기

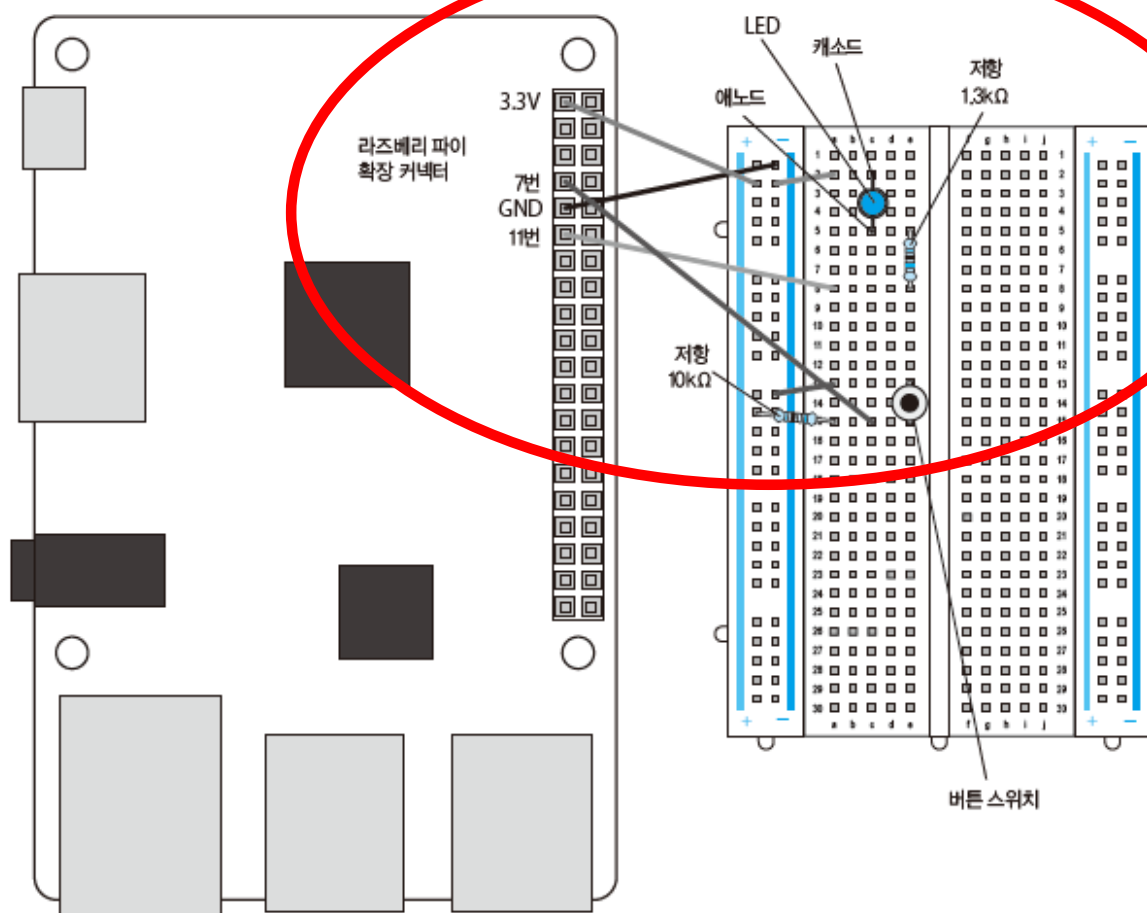


» 스위치 연결하기

- 스위치와 저항 배치를 바꾸면 스위치를 누를 때 전압이 로우 레벨에서 하이레벨이 될지, 하이 레벨에서 로우 레벨이 될지가 변함
- 핀이 입력 상태라면 이 핀이 받은 신호는 하이 레벨이거나 로우 레벨이어야 함
- 하이 레벨과 로우 레벨의 중간값을 받으면 입력 상태가 불안정해져서 프로그램과 전자 회로의 오동작 원인이 됨
- 보통은 저항을 사용해서 항상 한쪽 레벨이 되도록 회로를 구성함
- 이때 하이 레벨이 되게 만드는 저항을 풀업 저항이라 함
- 로우 레벨이 되게 만드는 저항을 풀다운 저항이라 함
- 그림 4-26처럼 두 회로에 있는 $10k\Omega$ 저항이 a 회로에서는 풀다운 저항으로 동작하고, b 회로에서는 풀업 저항으로 동작함

8. 스위치 연결하기

▼ 그림 4-27 실제 배선도(스위치)



8. 스위치 연결하기

▼ 표 4-7 스위치 회로의 부품 목록

부품명	부품 번호 · 규격	제조사	개수
LED	OSDR5113A	OptoSupply	1
버튼 스위치	SKRGAAD010	알프스	1
저항	1.3k Ω	지정된 것 없음	1
저항	10k Ω	지정된 것 없음	1

9. 파이썬 프로그램(스위치 입력)

» 파이썬 프로그램(스위치 입력)

- 이번에는 파이썬으로 스위치가 눌린 것을 감지해서 LED에 신호를 출력하는 프로그램을 만들어 보자

▼ 코드 4-5 Switch_01.py

```
# coding: utf-8

# GPIO 라이브러리 임포트
import RPi.GPIO as GPIO

# 핀 번호 할당 방법을 커넥터 핀 번호로 설정
GPIO.setmode(GPIO.BOARD)
```

9. 파이썬 프로그램(스위치 입력)

```
# 사용하는 핀 번호를 할당
SW = 7
LED = 11

# 11번 핀을 출력 핀으로 설정, 초기 출력은 로우 레벨
GPIO.setup(LED, GPIO.OUT, initial=GPIO.LOW)

# 7번 핀을 입력 핀으로 설정
GPIO.setup(SW, GPIO.IN)

# 예외 처리
try:
```



9. 파이썬 프로그램(스위치 입력)

```
# 무한 반복
while 1:
    # 스위치 상태를 변수 key_in에 할당
    key_in = GPIO.input(SW)
    # 변수 key_in 상태 판별
    if key_in==0:
        # 하이 레벨 출력
        GPIO.output(LED, GPIO.HIGH)
    else:
        # 로우 레벨 출력
        GPIO.output(LED, GPIO.LOW)

# 키보드 예외 검출
except KeyboardInterrupt:
    # 아무것도 하지 않음
    pass

# GPIO 개방
GPIO.cleanup()
```

9. 파이썬 프로그램(스위치 입력)

» 파이썬 프로그램(스위치 입력)

- 7번 핀을 스위치 상태를 받는 입력 핀으로 설정해서 그 상태를 변수 key_in에 할당하고 할당한 값을 판별함
- 이 예제에서는 스위치를 누르면 로우 레벨이 되므로 key_in이 0이면 스위치가 눌렸다고 봄
- 라즈베리 파이의 GPIO에는 풀업·풀다운 구성이 있음
- 7번 핀을 입력 핀으로 설정하는 줄을 다음과 같이 변경하면 스위치에 연결하는 저항을 생략할 수 있음

```
# 7번 핀을 풀업인 입력 핀으로 설정
```

```
GPIO.setup(SW, GPIO.IN, pull_up_down=GPIO.PUD_UP)
```



9. 파이썬 프로그램(스위치 입력)

» 파이썬 프로그램(스위치 입력)

- 스위치를 누를 때마다 LED가 켜지고 화면에 표시된 원의 색상이 변하는 GUI 프로그램을 만들어 보자
- 사용하는 회로는 이전과 같음

▼ 코드 4-6 tk_Switch_01.py

```
# coding: utf-8

# GPIO 라이브러리 импорт
import RPi.GPIO as GPIO

# Tkinter 라이브러리 импорт
import tkinter as tk # Python3
# import Tkinter as tk # Python2

# 핀 번호 할당 방법은 커넥터 핀 번호로 설정
GPIO.setmode(GPIO.BOARD)
```



9. 파이썬 프로그램(스위치 입력)

```
# 사용할 핀 번호를 할당
SW = 7
LED = 11

# 11번 핀을 출력 핀으로 설정, 초기 출력은 로우 레벨
GPIO.setup(LED, GPIO.OUT, initial=GPIO.LOW)

# 7번 핀을 풀업 입력 핀으로 설정
GPIO.setup(SW, GPIO.IN, pull_up_down=GPIO.PUD_UP)

# Tk 객체 인스턴스 작성
root = tk.Tk()

# 원을 그리기 위해 Canvas 객체 인스턴스 작성
# 너비: 200, 높이: 200
c = tk.Canvas(root, width = 200, height = 200)

# Canvas 배치
c.pack()
```

9. 파이썬 프로그램(스위치 입력)

```
# 원 작성
# 좌표 (50,50)에서 (150,150), 색: 투명
cc = c.create_oval(50, 50, 150, 150, fill = '')

# 스위치가 눌리면 실행할 함수를 정의
def check_SW(channel):
    # 스위치 상태를 변수 key_in에 할당
    key_in = GPIO.input(channel)
    # 변수 key_in 상태를 판별
    if key_in==0:
        # 하이 레벨 출력
        GPIO.output(LED, GPIO.HIGH)
        # 원을 빨강으로 채움
        c.itemconfig(cc, fill='red')
    else:
        # 로우 레벨 출력
        GPIO.output(LED, GPIO.LOW)
        # 원을 투명하게 바꿈
        c.itemconfig(cc, fill='')
```


9. 파이썬 프로그램(스위치 입력)

```
# 스위치 상태가 변할 때 check_SW 함수를 호출
GPIO.add_event_detect(SW, GPIO.BOTH, callback=check_SW)

# root 표시
root.mainloop()

# GPIO 개방
GPIO.cleanup()
```



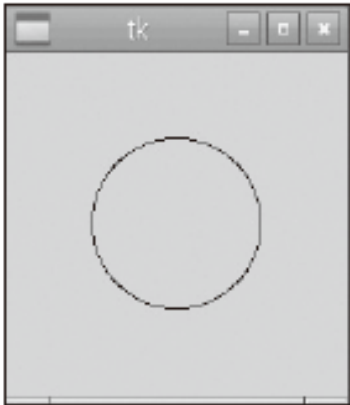
9. 파이썬 프로그램(스위치 입력)

» 파이썬 프로그램(스위치 입력)

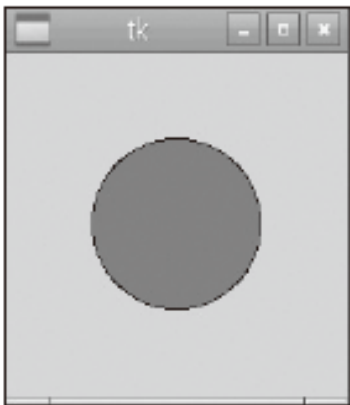
- create_oval 메서드로 원을 그림
- 스위치를 누를 때마다 스위치 색상을 itemconfig 메서드로 변경함
- 스위치가 눌려졌는지는 add_event_detect 메서드로 검출함
- 검출할 타이밍을 GPIO.BOTH로 지정했으므로 내리거나(하이 레벨에서 로우 레벨로 변화) 오르거나(로우 레벨에서 하이 레벨로 변화) 둘 다 check_SW 함수가 호출됨
- check_SW 함수에서는 스위치 상태를 판별한 후 판별한 결과에 따라 원의 색상을 정함

9. 파이썬 프로그램(스위치 입력)

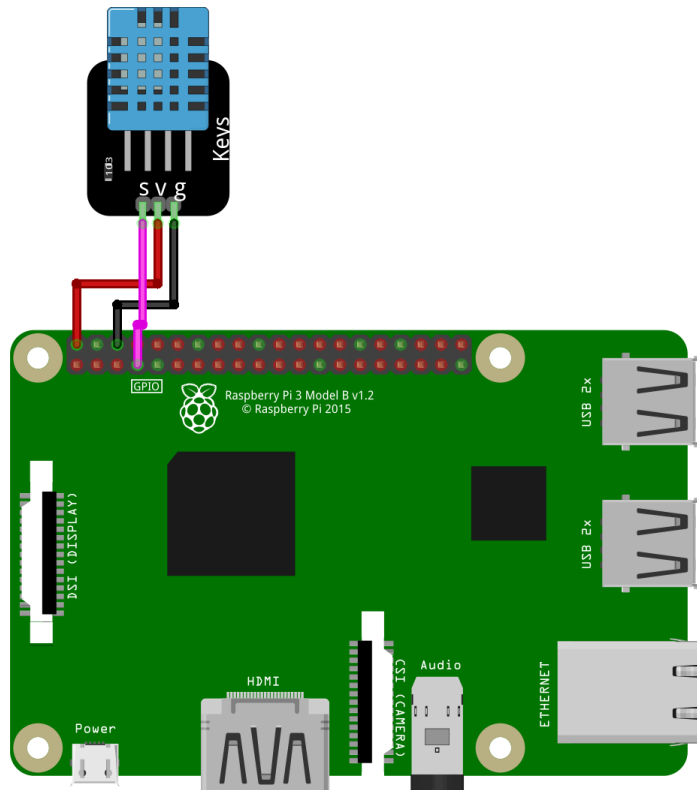
▼ 그림 4-28 tk_Switch_01.py 실행 결과 (스위치를 누르지 않았을 때)



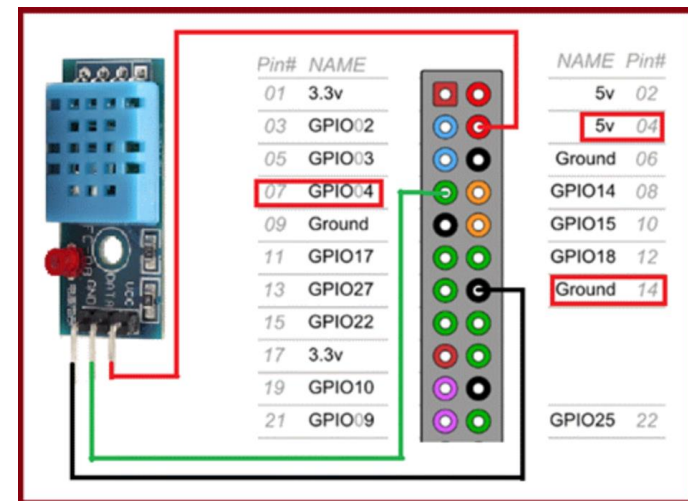
▼ 그림 4-28 tk_Switch_01.py 실행 결과 (스위치를 누르지 않았을 때)



10. DH11 온습도 센서 연결하기



fritzing



11. 파이썬 프로그램 (DH11 온습도 센서)

» Adafruit_DHT 라이브러리를 사용

```
$git clone https://github.com/adafruit/Adafruit_Python_DHT.git
$cd Adafruit_Python_DHT/
$sudo apt update
$sudo apt install build-essential python-dev python-openssl
$sudo python setup.py install
```

» 기본 온습도 출력

```
$ cd Adafruit_Python_DHT/examples
$ python AdafruitDHT.py 11 4
```

```
pi@raspberrypi:~/Adafruit_Python_DHT/examples $ ls
AdafruitDHT.py  google_spreadsheet.py  simpletest.py
pi@raspberrypi:~/Adafruit_Python_DHT/examples $ python AdafruitDHT.py 11 4
Temp=28.0* Humidity=43.0%
pi@raspberrypi:~/Adafruit_Python_DHT/examples $
```

11. 파이썬 프로그램 (DH11 온습도 센서)

» 파이썬 프로그램 (DH11 온습도 센서 입력)

▶ 코드 dh11.py

```
1 import time
2 import Adafruit_DHT
3 sensor = Adafruit_DHT.DHT11
4 pin = 4
5 try:
6     while True :
7         h, t = Adafruit_DHT.read_retry(sensor, pin)
8         if h is not None and t is not None :
9             print("Temperature = {0:0.1f}*C Humidity = {1:0.1f}%".format(t, h))
10        else :
11            print('Read error')
12        time.sleep(1)
13 except KeyboardInterrupt:
14     print("Terminated by Keyboard")
15
16 finally:
17     print("End of Program")
```



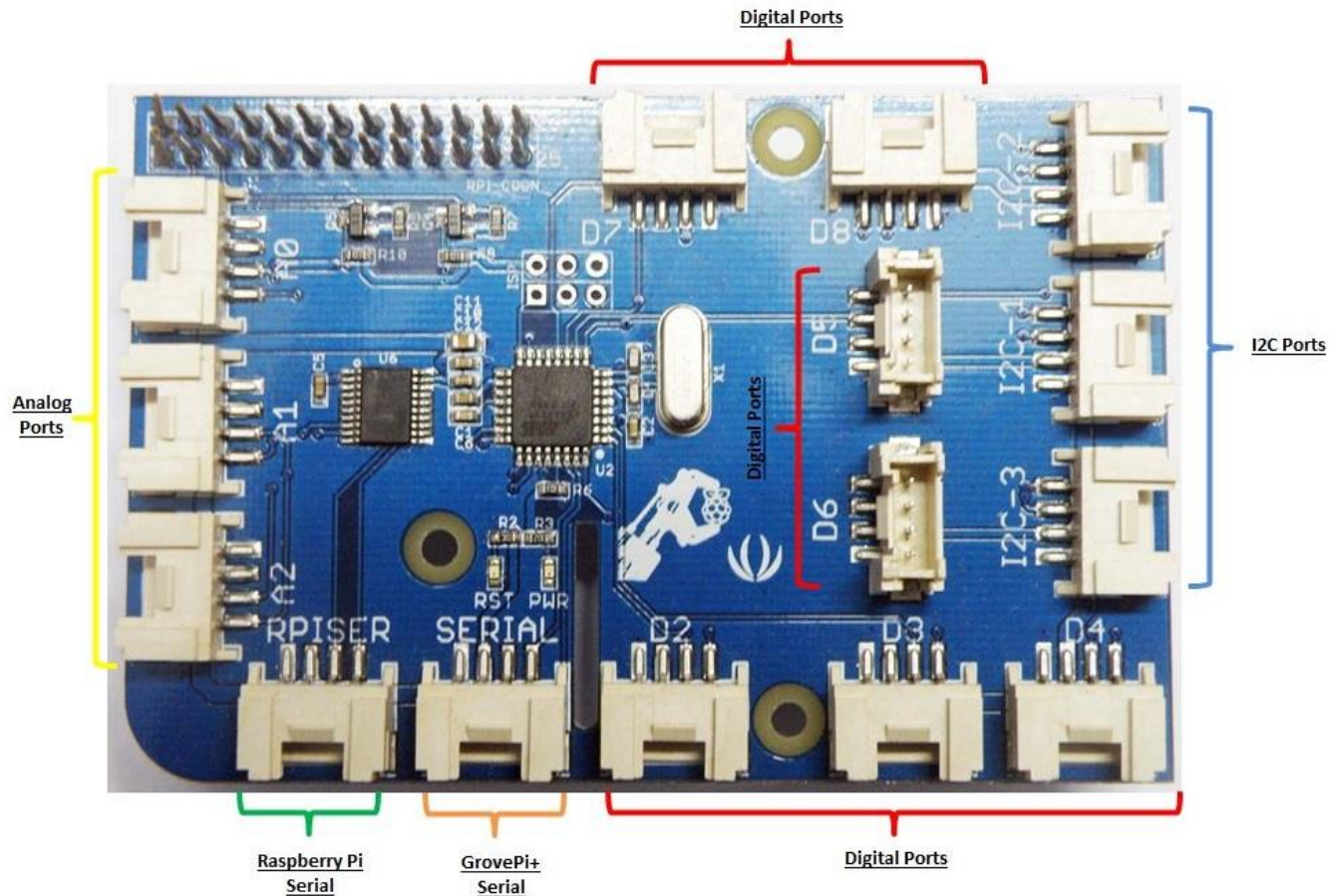
11. 파이썬 프로그램 (DH11 온습도 센서)

» 프로그램 실행

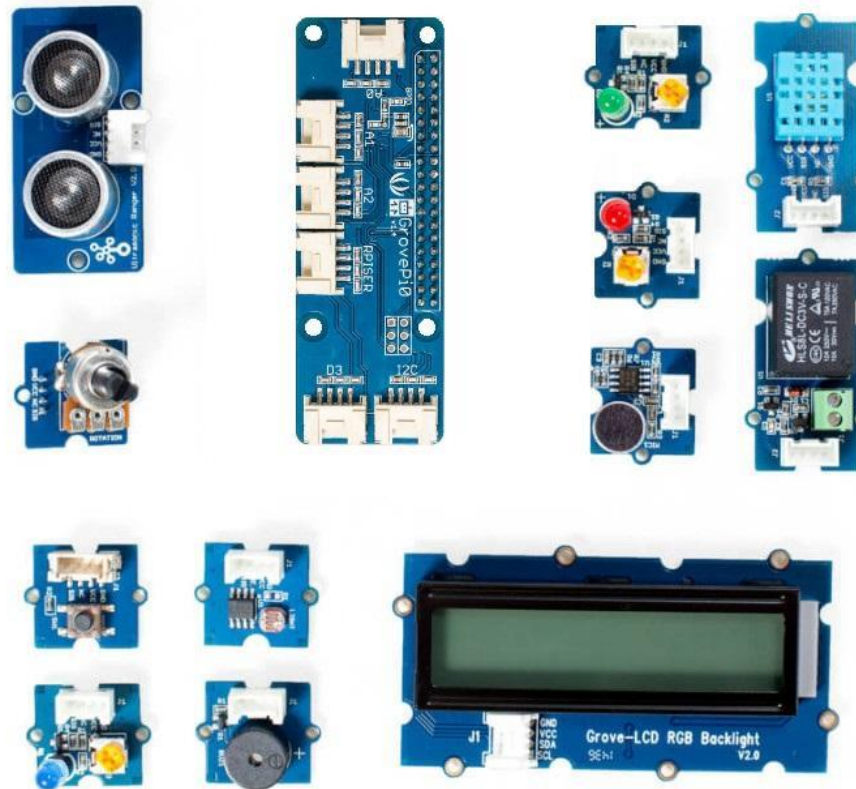
```
pi@raspberrypi:~ $ python3 dht11.py
Temperature = 27.0°C Humidity = 46.0%
Temperature = 27.0°C Humidity = 47.0%
Temperature = 27.0°C Humidity = 47.0%
Temperature = 27.0°C Humidity = 46.0%
Temperature = 27.0°C Humidity = 62.0%
```

12. GrovePi+

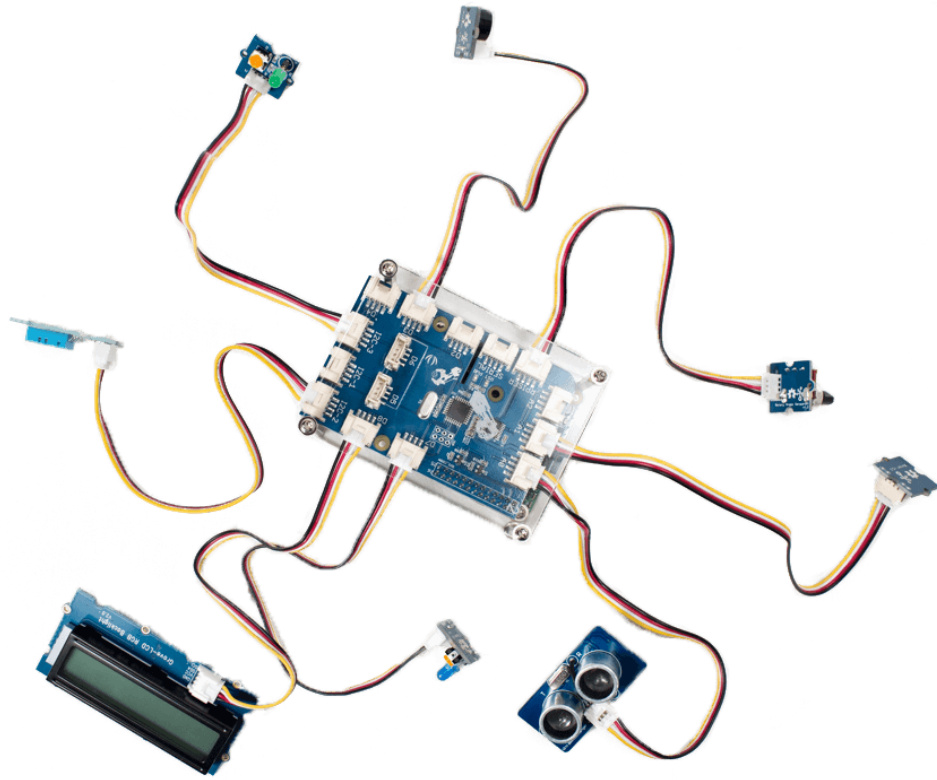
GROVEPI+ PORT DESCRIPTION



13. Grove 센서



14. 라즈베리파이 + GrovePi + 센서 연결하기



15. 파이썬 프로그램 (Grove DHT11 센서 연결)

```
import grovepi
import math

# Connect the Grove Temperature & Humidity Sensor Pro to digital port D4
sensor = 4 # The Sensor goes on digital port 4.
blue = 0 # The Blue colored sensor.
white = 1 # The White colored sensor.

while True:
    try:
        # This example uses the blue colored sensor.
        # The first parameter is the port, the second parameter is the type of sensor.
        [temp, humidity] = grovepi.dht(sensor, blue)
        if math.isnan(temp) == False and math.isnan(humidity) == False:
            print("temp = %.02f C humidity = %.02f%%"%(temp, humidity))
    except IOError: print ("Error")
```

