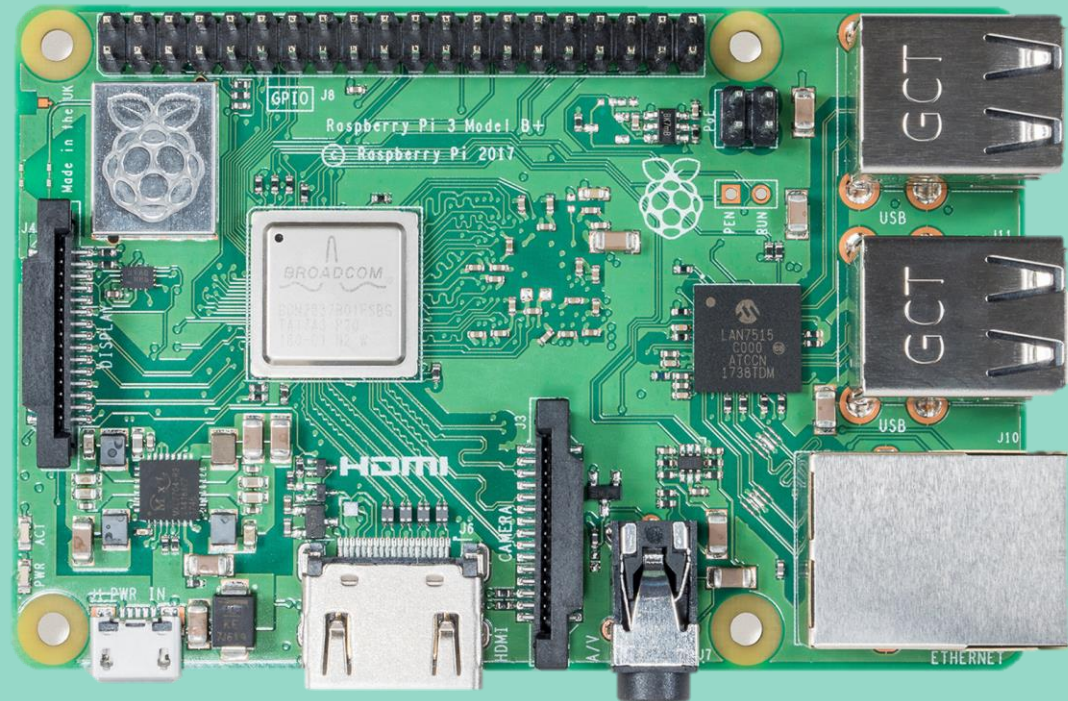


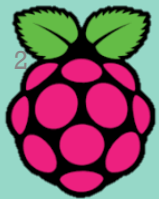
손쉽게 만드는 스마트기기

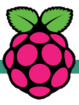
with 라즈베리파이 1



04

하드웨어 제어

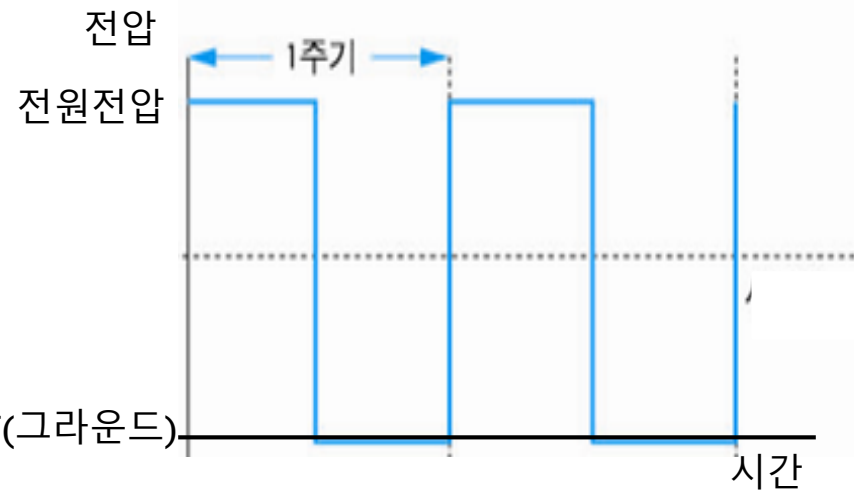
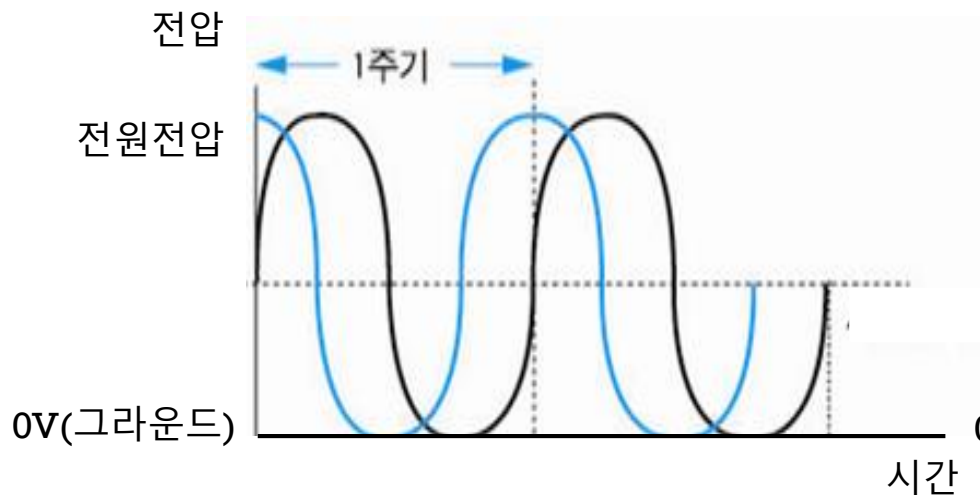


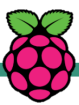


1. GPIO 제어

1. 아날로그 신호와 디지털 신호

- 신호 : 전기를 부호로 사용해서 내용을 전달하는 것
- 아날로그 신호 : 물질이나 시스템 등의 상태를 연속된 값으로 나타낸 값
- 디지털 신호 : 자료나 정보 따위를 이진수 같은 유한 자릿수의 수열로 나타낸 것



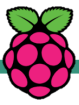


1. GPIO 제어

➤ 디지털 신호

- 표준 5 V 논리레벨 0.7 V 이하 ➔ 0 또는 LOW
3.3 V 이상 ➔ 1 또는 HIGH 로 인식
- GPIO 핀은 0.7 V 이하 ➔ 0 또는 LOW
2.7 V ~ 3.3 V ➔ 1 또는 HIGH 로 인식

주의. 라즈베리 파이의 GPIO 핀은 3.3V 이상의 입력 전압이 들어오면 망가진다



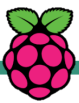
1. GPIO 제어

2. GPIO (General Purpose Input/Output) 제어



Alternate Function					Alternate Function
	3.3V PWR	1		2	5V PWR
I2C1 SDA	GPIO 2	3		4	5V PWR
I2C1 SCL	GPIO 3	5		6	GND
	GPIO 4	7		8	UART0 TX
	GND	9		10	UART0 RX
	GPIO 17	11		12	GPIO 18
	GPIO 27	13		14	GND
	GPIO 22	15		16	GPIO 23
	3.3V PWR	17		18	GPIO 24
SPI0 MOSI	GPIO 10	19		20	GND
SPI0 MISO	GPIO 9	21		22	GPIO 25
SPI0 SCLK	GPIO 11	23		24	GPIO 8
	GND	25		26	GPIO 7
	Reserved	27		28	Reserved
	GPIO 5	29		30	GND
	GPIO 6	31		32	GPIO 12
	GPIO 13	33		34	GND
SPI1 MISO	GPIO 19	35		36	GPIO 16
	GPIO 26	37		38	GPIO 20
	GND	39		40	GPIO 21
					SPI0 CS0
					SPI0 CS1
					SPI1 CS0
					SPI1 MOSI
					SPI1 SCLK

40핀 확장 핀 맵



1. GPIO 제어

2. GPIO 제어

- 하드웨어 제어를 위한 프로그램 설치

- 파이썬 – Rpi.GPIO 모듈

쉽다

인터프리터 언어 → 실행속도 느림

- C언어 – WiringPi 라이브러리

실시간제어나 time-critical한 분야 제어

컴파일 언어 → 실행속도 빠름



1. GPIO 제어

2. GPIO 제어

➤ 출력

- SW에서 핀의 방향을 출력으로 설정
- GPIO 핀의 전기적 상태를
H(3.3V) 혹은 L(0V)로 변경

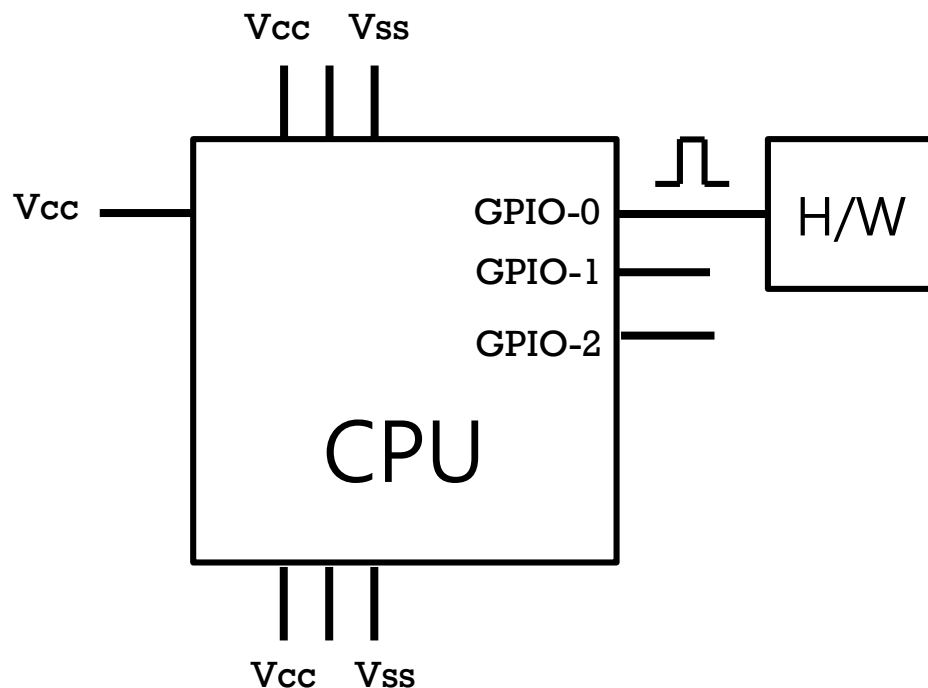
➤ 입력

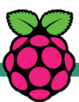
- SW에서 핀의 방향을 입력으로 설정
- GPIO 핀의 전기적 상태가
H(3.3V) 혹은 L(0V)인지 확인

➤ RPi.GPIO (→ `import RPi.GPIO as GPIO` 추가)

➔ 라즈베리파이의 확장 커넥터 핀을 범용 입출력(GPIO)

핀으로 제어하기위한 파이썬 라이브러리





1. GPIO 제어

2. GPIO 제어

2.1 Rpi.GPIO를 이용한 제어


- GPIO 설정함수

`GPIO.setmode(GPIO.BOARD)`

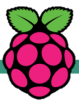
➔ 핀 번호 인덱싱: 보드 기준 설정

setmode에서
GPIO.BOARD로 정의하면
01 ~ 40번까지의 숫자로
핀 번호를 사용

예를 들어 13번핀은
GPIO27인데, 프로그램에서는
13번으로 사용



Pin#	NAME		NAME	Pin#
01	3.3v DC Power		DC Power 5v	02
03	GPIO02 (SDA1 , I ² C)		DC Power 5v	04
05	GPIO03 (SCL1 , I ² C)		Ground	06
07	GPIO04 (GPIO_GCLK)		(TXD0) GPIO14	08
09	Ground		(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)		(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)		Ground	14
15	GPIO22 (GPIO_GEN3)		(GPIO_GEN4) GPIO23	16
17	3.3v DC Power		(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)		Ground	20
21	GPIO09 (SPI_MISO)		(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)		(SPI_CE0_N) GPIO08	24
25	Ground		(SPI_CE1_N) GPIO07	26
27	ID_SD (I ² C ID EEPROM)		(I ² C ID EEPROM) ID_SC	28
29	GPIO05		Ground	30
31	GPIO06		GPIO12	32
33	GPIO13		Ground	34
35	GPIO19		GPIO16	36
37	GPIO26		GPIO20	38
39	Ground		GPIO21	40



1. GPIO 제어

2. GPIO제어

2.1 Rpi.GPIO를 이용한 제어


- GPIO 설정함수





















`GPIO.setmode(GPIO.BCM)`

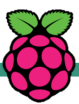
➔ 핀 번호 인덱싱: CPU 기준 설정

`setmode`에서
`GPIO.BCM`으로 정의하면
GPIO 뒤에 있는 숫자로
핀 번호를 사용한다

예를 들어 13번핀은
`GPIO27`인데, 프로그램에서는
27번으로 사용한다



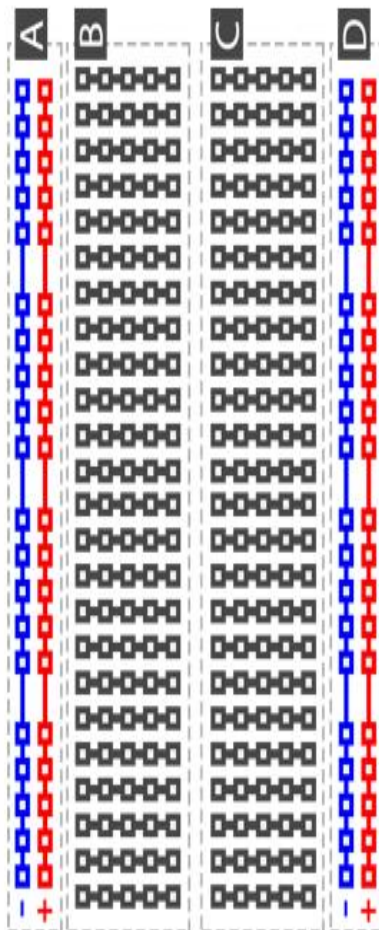
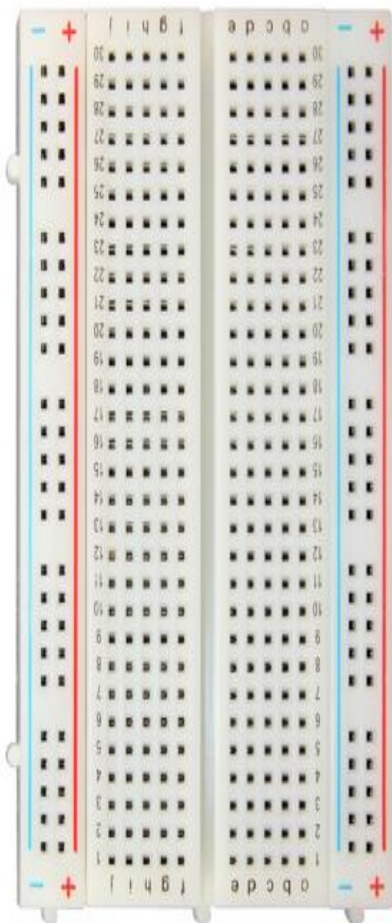
Pin#	NAME		NAME	Pin#
01	3.3v DC Power		DC Power 5v	02
03	GPIO02 (SDA1 , I ² C)		DC Power 5v	04
05	GPIO03 (SCL1 , I ² C)		Ground	06
07	GPIO04 (GPIO_GCLK)		(TXD0) GPIO14	08
09	Ground		(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)		(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)		Ground	14
15	GPIO22 (GPIO_GEN3)		(GPIO_GEN4) GPIO23	16
17	3.3v DC Power		(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)		Ground	20
21	GPIO09 (SPI_MISO)		(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)		(SPI_CE0_N) GPIO08	24
25	Ground		(SPI_CE1_N) GPIO07	26
27	ID_SD (I ² C ID EEPROM)		(I ² C ID EEPROM) ID_SC	28
29	GPIO05		Ground	30
31	GPIO06		GPIO12	32
33	GPIO13		Ground	34
35	GPIO19		GPIO16	36
37	GPIO26		GPIO20	38
39	Ground		GPIO21	40



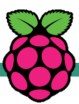
1. GPIO 제어

2. GPIO제어

2.2 브레드보드



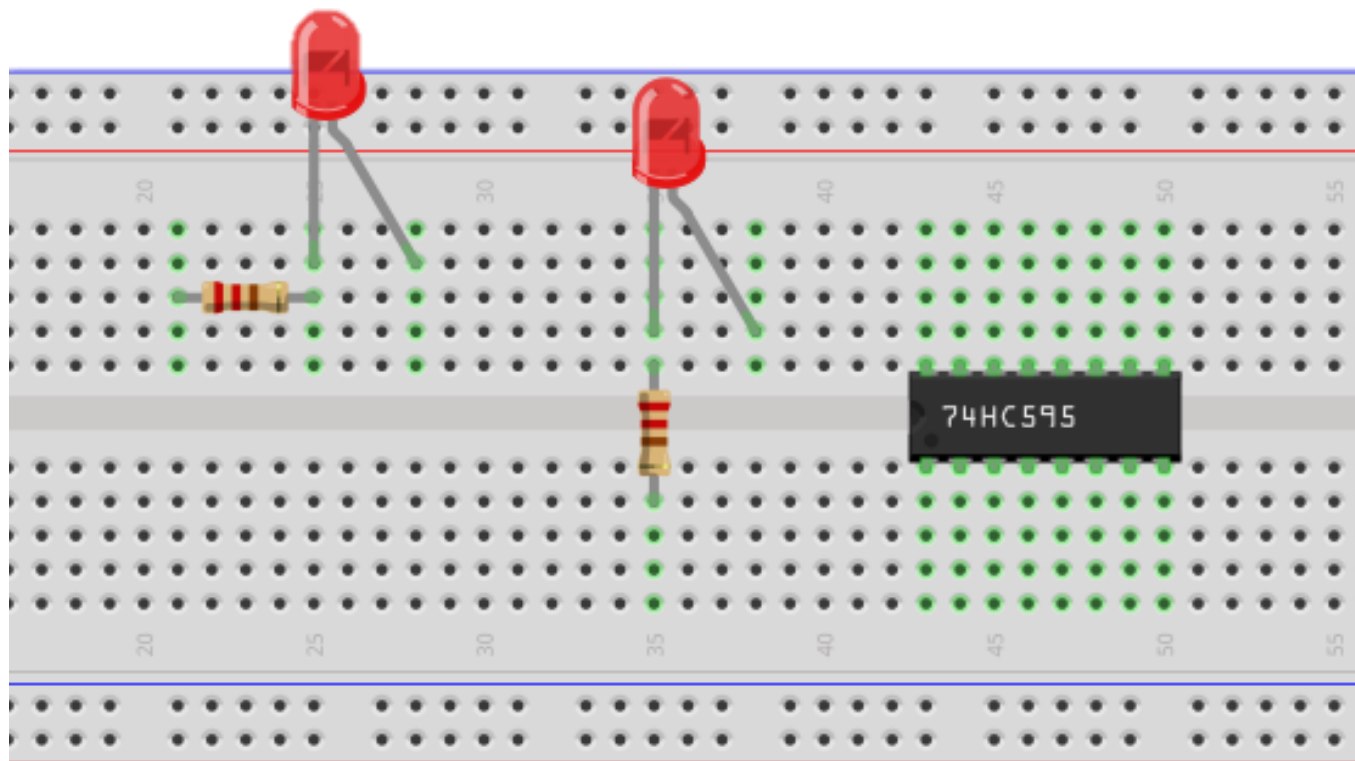
- 세로줄은 버스영역
빨간선은 +
파란선은 - 전원선 연결
- 가로줄은 IC/영역
부품영역.
- 부품영역 위 아래 서로 5칸
씩 격리, 위 아래 5칸은
서로 전기가 통하지 않음

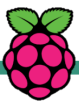


1. GPIO 제어

2. GPIO제어

2.2 브레드보드





1. GPIO 제어

2. GPIO제어

2.3 파이썬 라이브러리 설치

- Rpi.GPIO 새 버전 설치

```
$ sudo apt-get update
```

```
$ sudo apt-get upgrade
```

- 파이썬 모듈을 구축하기 위한 python-dev 패키지 설치

```
$ sudo apt-get install python-dev
```

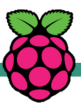
- 파이썬 패키지관리 소프트웨어 pip 설치

```
$ sudo apt-get install python-pip
```

```
$ pip --version
```

```
$ sudo pip install --upgrade pip
```

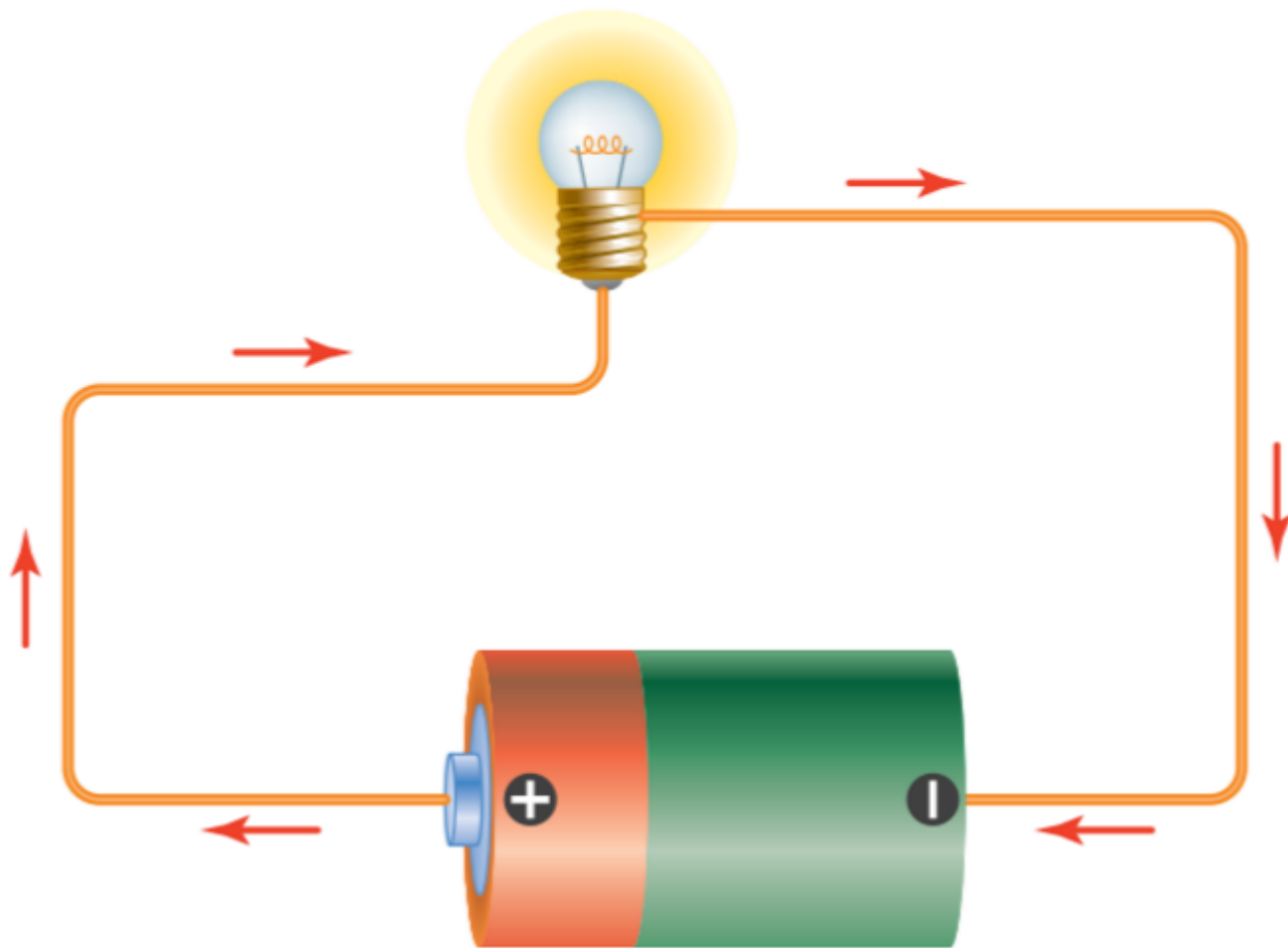
```
$ pip --version
```

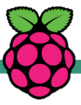


2. LED와 스위치

1. LED 켜기

- 전기회로 꾸미기



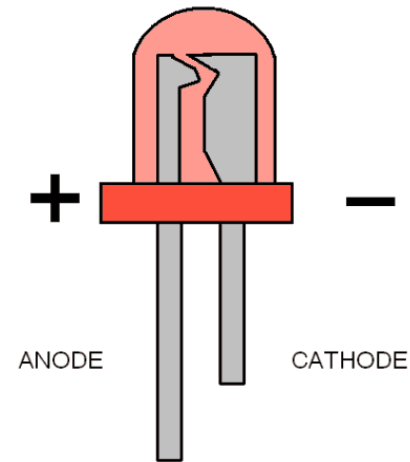


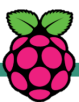
2. LED와 스위치

1. LED 켜기

1.1 LED 란 ?

- Light-Emitting Diode, 빛을 내는 다이오드
- +(양극, ANODE) 와 - (음극, CATHODE)극성이 있다.
- 5V 에 긴 선인 + 연결 , 짧은 선인 - 극을 GND에 연결





2. LED와 스위치

1. LED 켜기

1.2 저항이란 ?

- 전류의 흐름을 조절하는 전자부품 (단위 : Ω (옴))
- 극성은 없다.
- 전기를 열로 바꿔주는 전자부품.
- 저항 값이 높을 수록 더 많은 전기를 열로 바꿔준다.
- 과전류로부터 전자부품을 보호하기 위해 사용

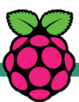


220 Ω 저항
(빨 빨 갈)



1K Ω 저항
(갈 검 빨)

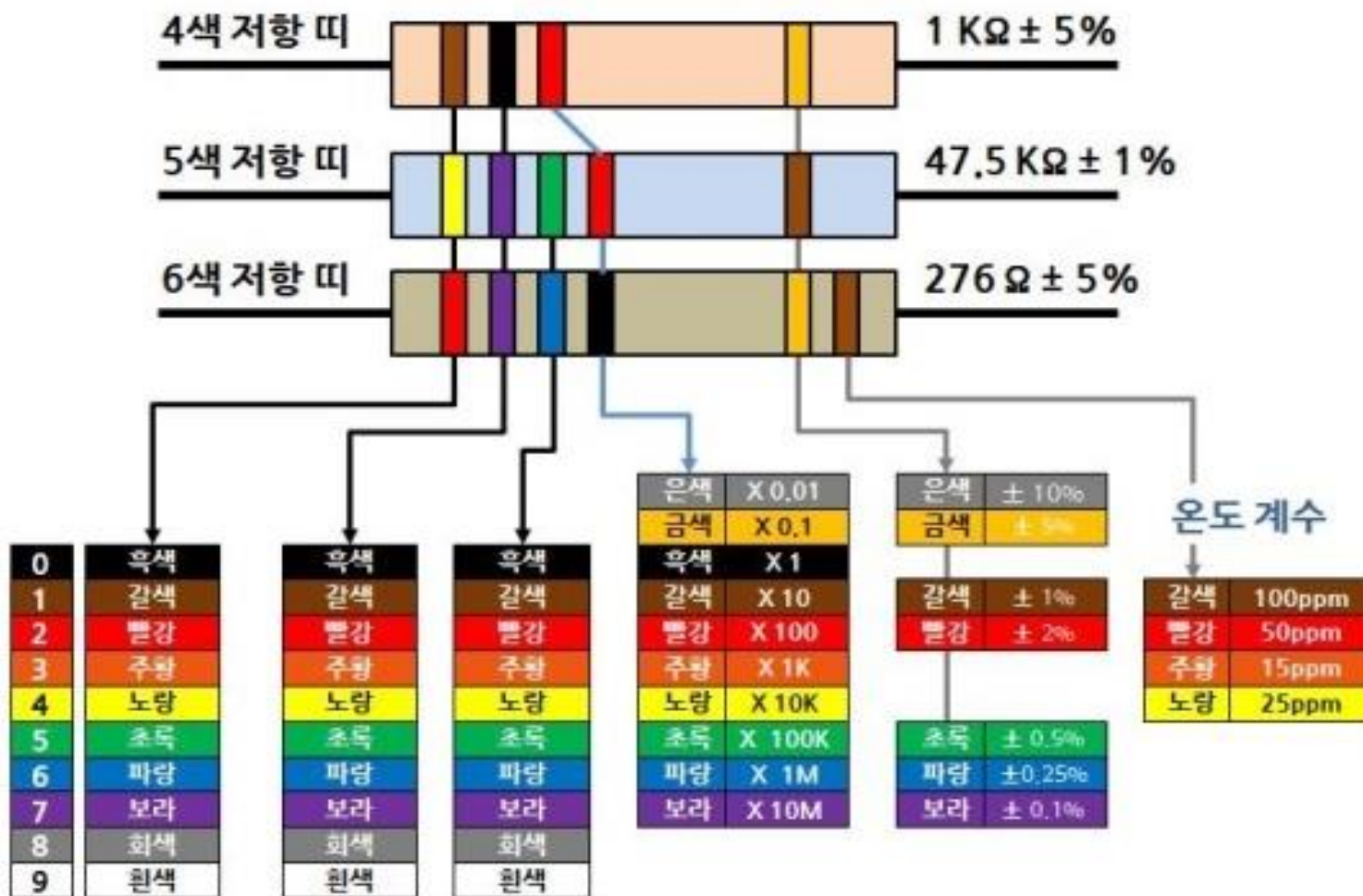


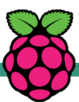


2. LED와 스위치

1. LED 켜기

1.3 저항 읽는 법

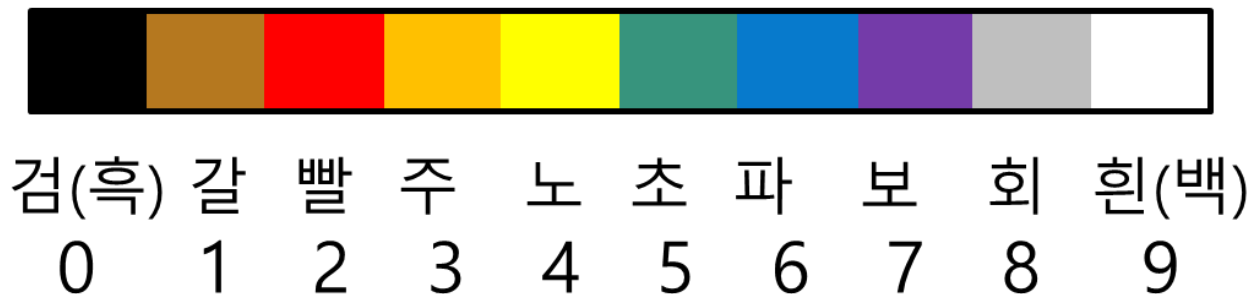
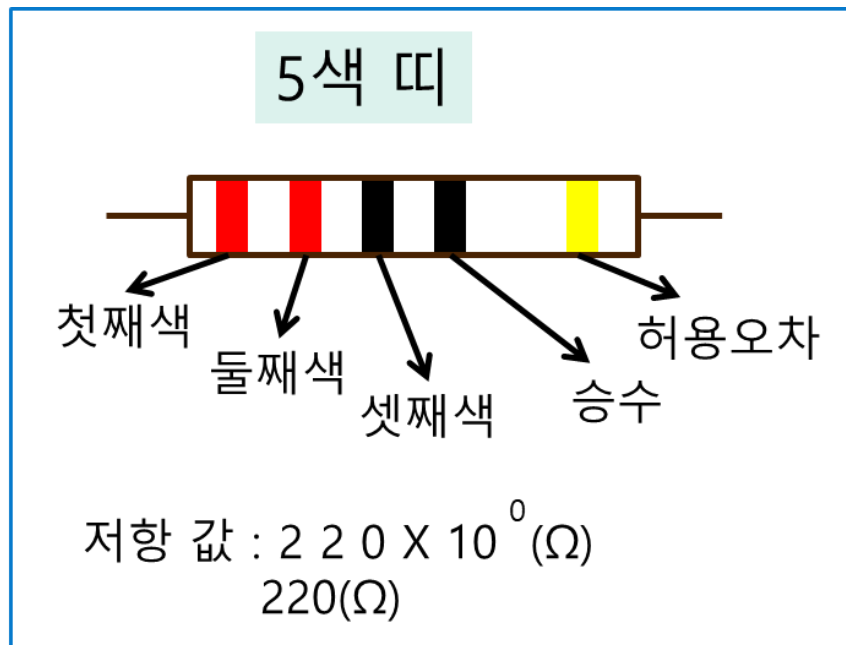
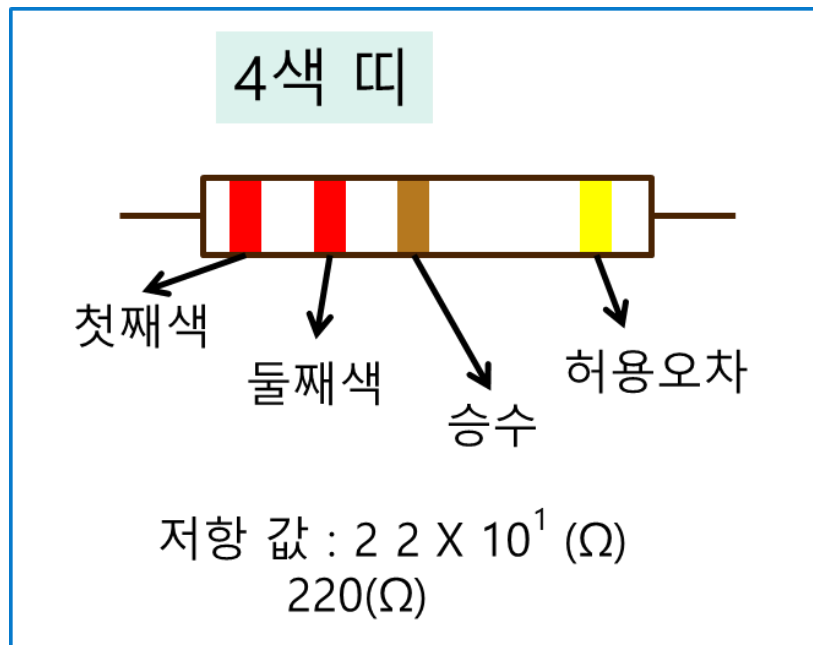




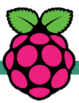
2. LED와 스위치

1. LED 켜기

1.3 저항 읽는 법



허용오차
금 : $\pm 5\%$
은 : $\pm 10\%$



2. LED와 스위치

1. LED 켜기

1.4 Rpi.GPIO를 이용한 제어

- GPIO 설정 함수

- GPIO.setup(n, GPIO.OUT)

- GPIO.setup(11, GPIO.OUT) → 11번 핀을 출력으로 지정

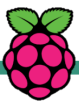
- GPIO.setup(11, GPIO.OUT, initial=GPIO.LOW) → 초기값 지정

- GPIO.setup(n, GPIO.IN)

- GPIO.setup(13, GPIO.IN) → 13번 핀을 입력으로 지정

- GPIO.cleanup() → 모든 GPIO 핀 개방, 종료시 반드시 실행

- GPIO.cleanup(11) → 11번 GPIO 핀 개방

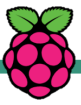


2. LED와 스위치

1. LED 켜기

1.4 Rpi.GPIO를 이용한 제어

- GPIO 입력 함수
 - GPIO.input(n) → n번 핀의 전기적 상태를 1/0으로 반환
GPIO.input(5)
- GPIO 출력 함수
 - GPIO.output(n, 1) → 핀 번호 n의 출력을 HIGH로 설정
 - GPIO.output(n, 0) → 핀 번호 n의 출력을 LOW로 설정

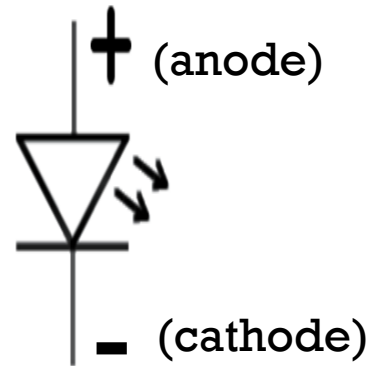
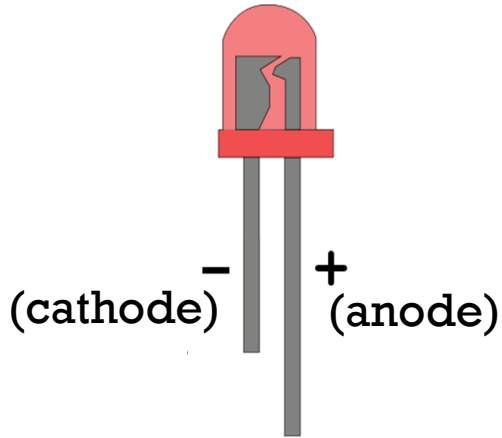


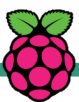
2. LED와 스위치

1. LED 켜기

1.4 Rpi.GPIO를 이용한 제어

- GPIO 출력 제어 (LED 켜기)

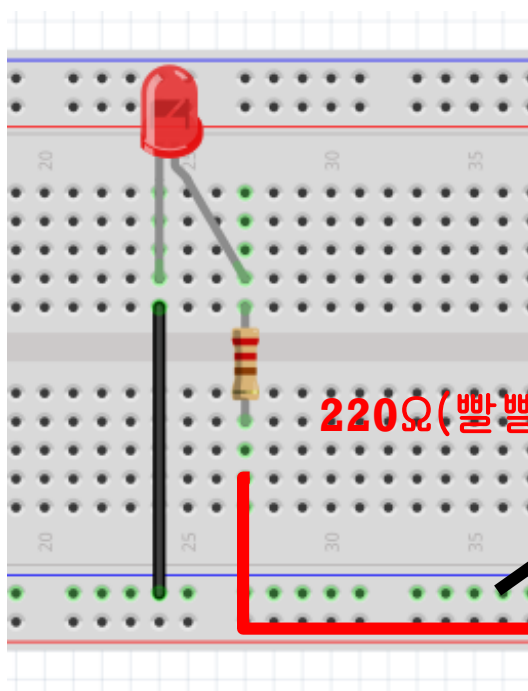




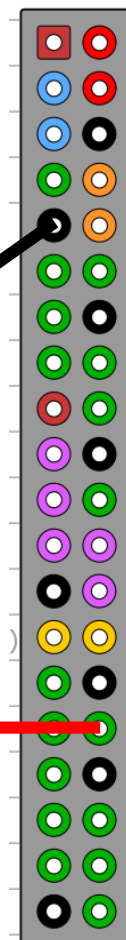
2. LED와 스위치

1. LED 켜기

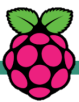
1.5 LED 3초 동안 켜기



GND



NAME	Pin#
DC Power 5v	02
DC Power 5v	04
Ground	06
(TXD0) GPIO14	08
(RXD0) GPIO15	10
(GPIO_GEN1) GPIO18	12
Ground	14
(GPIO_GEN4) GPIO23	16
(GPIO_GEN5) GPIO24	18
Ground	20
(GPIO_GEN6) GPIO25	22
(SPI_CE0_N) GPIO08	24
(SPI_CE1_N) GPIO07	26
(I ² C ID EEPROM) ID_SC	28
Ground	30
GPIO12	32
Ground	34
GPIO16	36
GPIO20	38
GPIO21	40



2. LED와 스위치

1. LED 켜기

1.5 LED 3초 동안 켜기

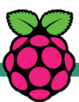
File name: LED_01.py

```
import RPi.GPIO as GPIO
from time import sleep
```

```
GPIO.setmode(GPIO.BCM)
GPIO.setup(12, GPIO.OUT, initial=GPIO.LOW)
```

```
GPIO.output(12, GPIO.HIGH)
sleep(3)           → 3초 지연
```

```
GPIO.cleanup() → 실행하지 않으면 다음 실행시 경고 메시지
```

2. LED와 스위치

1. LED 켜기

1.6 LED 켜다 끄기

File name: LED_02.py

```
import RPi.GPIO as GPIO  
from time import sleep
```

```
GPIO.setmode(GPIO.BCM)  
GPIO.setup(12, GPIO.OUT, initial=GPIO.LOW)
```

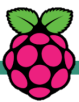
```
GPIO.output(12, GPIO.HIGH)  
sleep(0.5)
```

```
GPIO.output(12, GPIO.LOW)  
sleep(0.5)
```

```
GPIO.output(12, GPIO.HIGH)  
sleep(0.5)
```

```
GPIO.output(12, GPIO.LOW)  
sleep(0.5)
```

```
GPIO.cleanup()
```



2. LED와 스위치

File name: LED_03.py

```
import RPi.GPIO as GPIO
from time import sleep
```

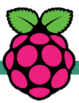
```
led = 12
GPIO.setmode(GPIO.BCM)
GPIO.setup(led, GPIO.OUT)    → 12번 핀 출력 설정
```

```
try:
    while True:              → 무한 루프
        GPIO.output(led, GPIO.HIGH)    # GPIO.output(led, 1)
        sleep(0.5)

        GPIO.output(led, GPIO.LOW)     # GPIO.output(led, 0)
        sleep(0.5)
```

```
except KeyboardInterrupt:    → 키보드에서 Ctrl+C 누르면 여기로 진입
    pass
```

```
GPIO.cleanup()              → GPIO 핀 개방
```



2. LED와 스위치

1. LED 켜기

1.6 LED 켜다 끄기 – GUI 버튼을 누를 때 마다 켜지거나 꺼지는 LED

File name: LED_04.py

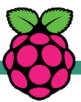
```
import RPi.GPIO as GPIO
import tkinter
```

```
led = 12
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(led, GPIO.OUT, initial = GPIO.LOW)
```

#LED를 켜고 끄는 함수 정의

```
def func_led():
    #11번 핀에서 나오는 입력 값을 반전해서 출력
    GPIO.output(led, not GPIO.input(led))
```

```
root = tkinter.Tk()
root.geometry('300x300+50+50')
```



2. LED와 스위치

File name: LED_04.py

```
label = tkinter.Label(root, text = 'press button')
```

```
label.pack()
```

```
# root에 표시할 버튼 정의
```

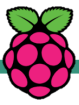
```
button = tkinter.Button(root, text = 'LED', command=func_led)
```

```
button.pack()
```

```
root.mainloop()
```

```
# GPIO 개방
```

```
GPIO.cleanup()
```



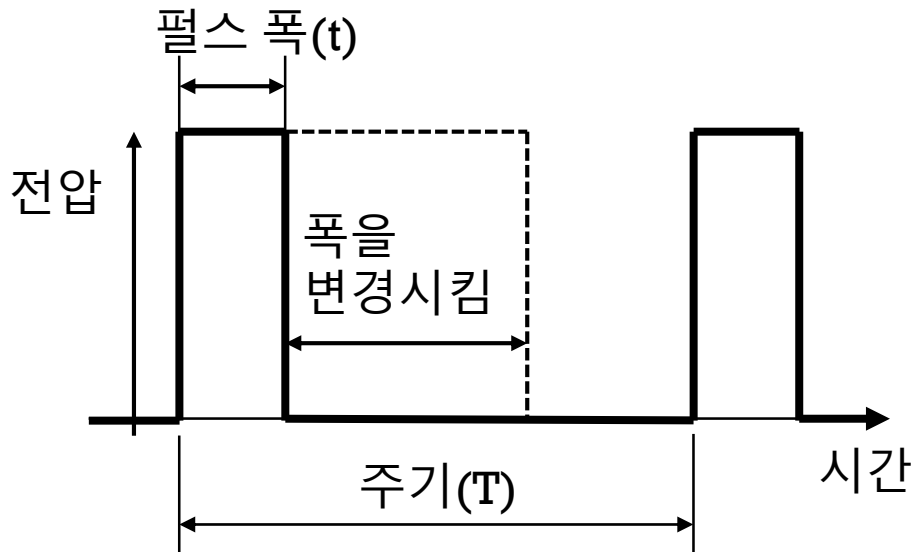
2. LED와 스위치

2. LED 밝기 조절

2.1 PWM(Pulse Width Modulation, 펄스 폭 변조)

- 일정 간격으로 신호의 하이 레벨과 로우 레벨의 폭을 전환해서 중간 값을 유사하게 표현하는 신호방식

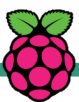
PWM 신호



- 듀티비(duty ratio)

PWM 신호 주기에 대한 펄스 폭 비율

$$\text{듀티비 } D = \frac{t}{T}$$



2. LED와 스위치

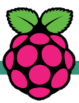
2. LED 밝기 조절

2.2 PWM(Pulse Width Modulation, 펄스 폭 변조) 신호 제어

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

PWM 객체의 메서드

메서드	설명	인수	인수 설명
start(dc)	PWM신호 출력	dc:듀티비	듀티비를 부동소수로 지정(0.0~100.0 %)
ChangeFrequency(freq)	PWM 신호의 주파수 변경	freq:주파수	주파수(Hz) 지정
ChangeDutyCycle(dc)	PWM신호의 듀티비 변경	dc:듀티비	듀티비를 부동소수로 지정(0.0~100.0 %)
stop()	PWM신호 정지	없음	



2. LED와 스위치

2. LED 밝기 조절

2.3 LED 밝기 조절

File name: LED_05.py

```
import RPi.GPIO as GPIO
from time import sleep
```

```
led = 12
GPIO.setmode(GPIO.BCM)
GPIO.setup(led, GPIO.OUT, initial = GPIO.LOW)
```

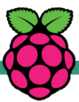
```
# 듀티 비 목록 작성
```

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

```
# PWM 객체 인스턴스 작성
```

```
# 출력 핀 : 12 번, 주파수 100 Hz
```

```
p = GPIO.PWM(led, 100)
```

2. LED와 스위치

File name: LED_05.py

```
# PWM 신호 출력
```

```
p.start(0)
```

```
try:
```

```
    while True:
```

→ 무한 루프

```
        for val in dc:
```

```
            p.ChangeDutyCycle(val)
```

```
            sleep(0.5)
```

```
        dc.reverse()
```

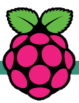
```
        sleep(1)
```

```
except KeyboardInterrupt:
```

```
    pass
```

```
p.stop()
```

```
GPIO.cleanup()
```



2. LED와 스위치

2. LED 밝기 조절

2.4 슬라이더로 LED 밝기 조절

File name: LED_06.py

```
import RPi.GPIO as GPIO
from tkinter import *
```

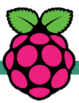
```
led = 12
GPIO.setmode(GPIO.BCM)
```

```
GPIO.setup(led, GPIO.OUT, initial = GPIO.LOW)
```

```
# PWM 객체 인스턴스 작성
p = GPIO.PWM(led, 100)
```

```
# Tk 객체 인스턴스 작성
root = Tk()
```

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



2. LED와 스위치

File name: LED_06.py

```
p.start(0)
```

```
# 듀티 비를 변경하는 함수 정의
def change_duty(dc):
    p.ChangeDutyCycle(led_val.get())
```

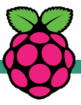
```
# root 에 표시할 슬라이더 정의
# 레이블 LED, 수평으로 표시, 숫자 범위는 0 ~ 100
s = Scale(root, label = 'LED', orient = 'h', from_ = 0, to = 100, w
          variable = led_val, command = change_duty)
```

```
# 슬라이더 배치
s.pack()
```

```
root.mainloop()
```

```
p.stop()
```

```
GPIO.cleanup()
```

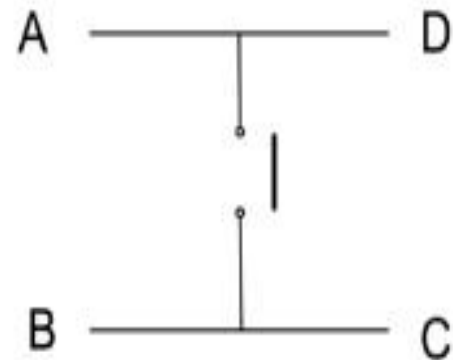


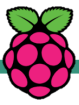
2. LED와 스위치

3. 스위치

- 디지털 신호 입력 장치
- 푸시 버튼 스위치

스마트폰의 버튼, 키보드, 세탁기버튼, 전자레인지버튼...

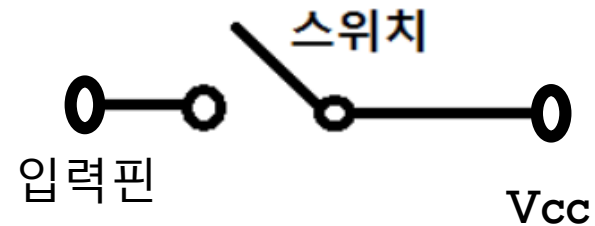
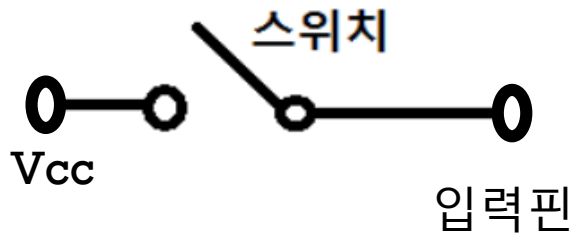


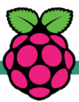


2. LED와 스위치

3. 스위치

3.1 플로팅(floating) - 스위치가 열렸을 때 0과 1사이에서 방황하는 아무런 상태도 아닌 값 출력하는 상태





2. LED와 스위치

3. 스위치

3.2 스위치 입력 값 체크

File name: SW_01.py

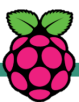
```
import RPi.GPIO as GPIO
from time import sleep
```

```
switch = 20
GPIO.setmode(GPIO.BCM)
GPIO.setup(switch, GPIO.IN)    → 20번 핀을 입력으로 설정
```

```
try:
    while True:
        switch_in = GPIO.input(switch)
        print(switch_in)        → 입력 값을 출력
        sleep(0.1)
```

```
except KeyboardInterrupt:
    pass
```

```
GPIO.cleanup()
```

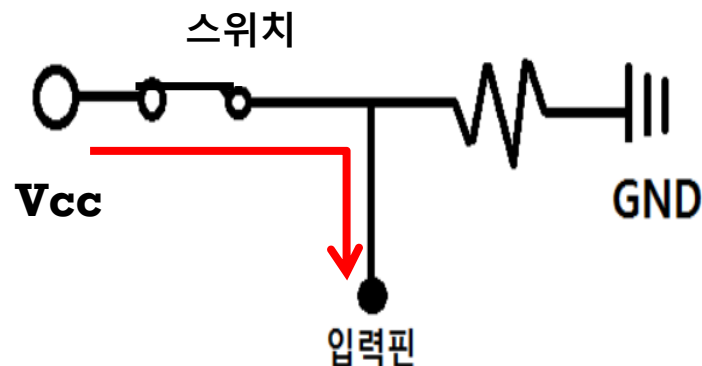
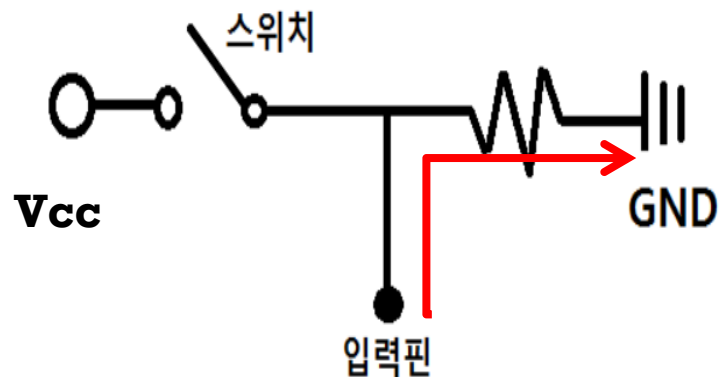


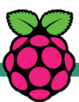
2. LED와 스위치

3. 스위치

3.3 풀다운(pulldown) 스위치

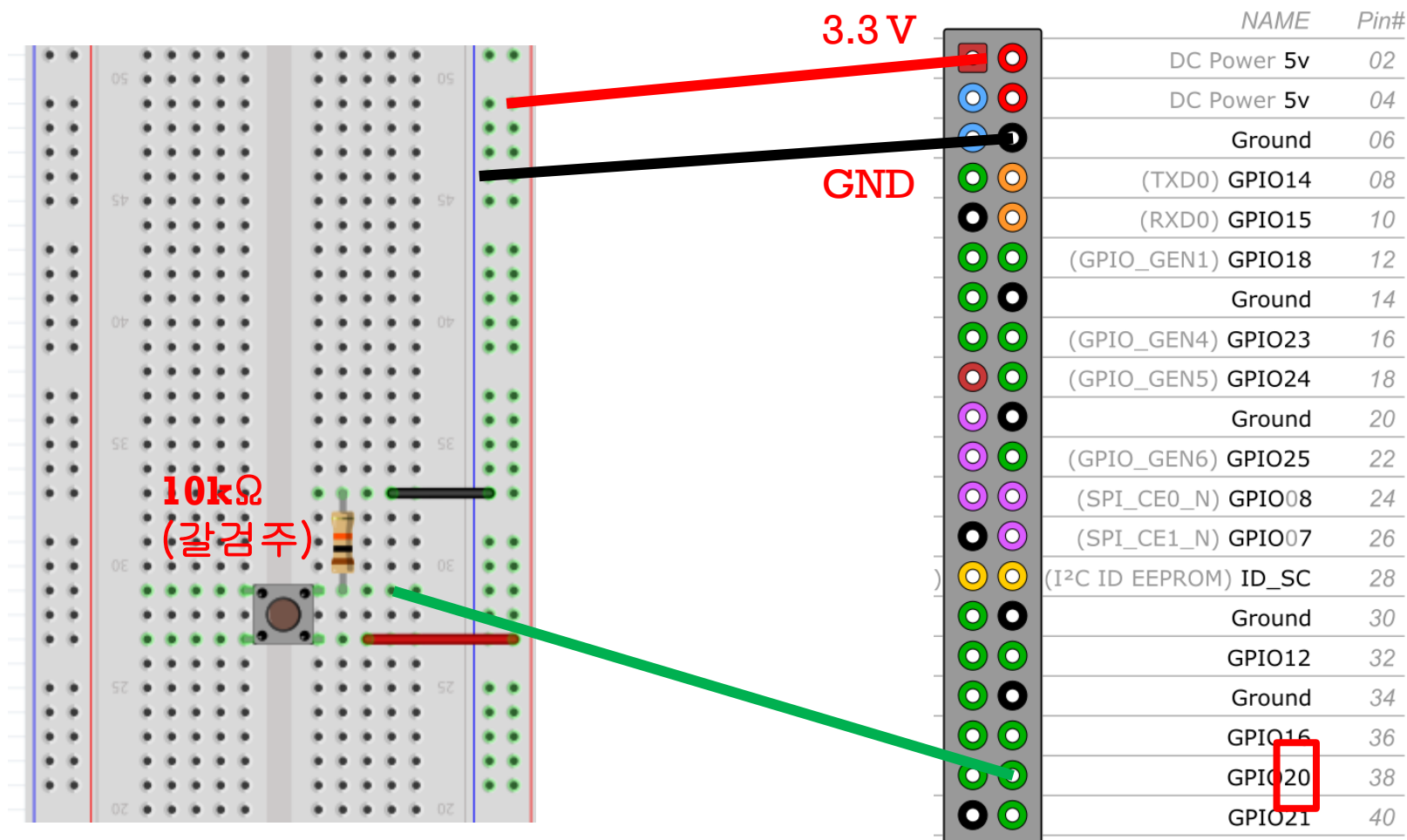
- 플로팅 상태의 값을 다운시켜버린다는 의미를 가진다.
- 입력 핀과 GND사이를 저항을 두고 연결한다.
- 버튼이 눌러지지 않으면 OFF 상태로 입력 핀의 전압은 LOW(0V)가 된다.
- 버튼을 누르면 HIGH(3.3V)

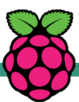




2. LED와 스위치

➤ 풀다운(pulldown) 스위치



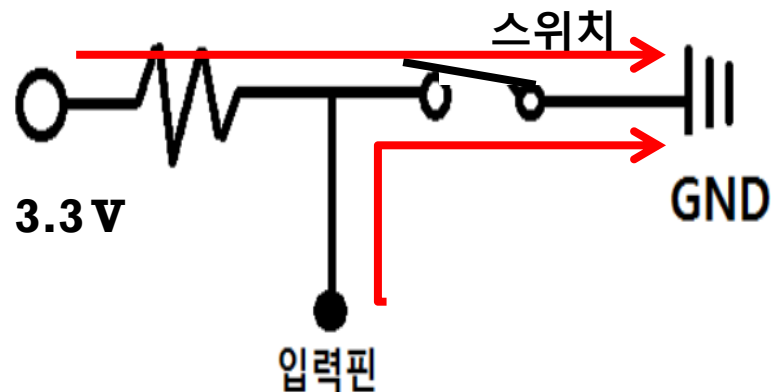
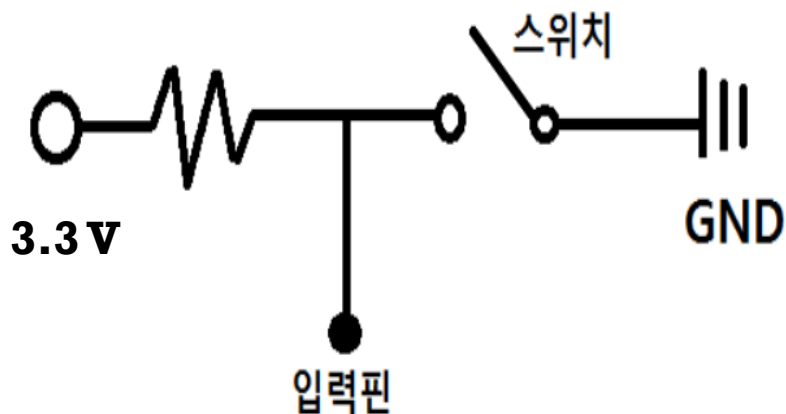


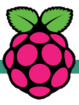
2. LED와 스위치

3. 스위치

3.4 풀업(pullup) 스위치

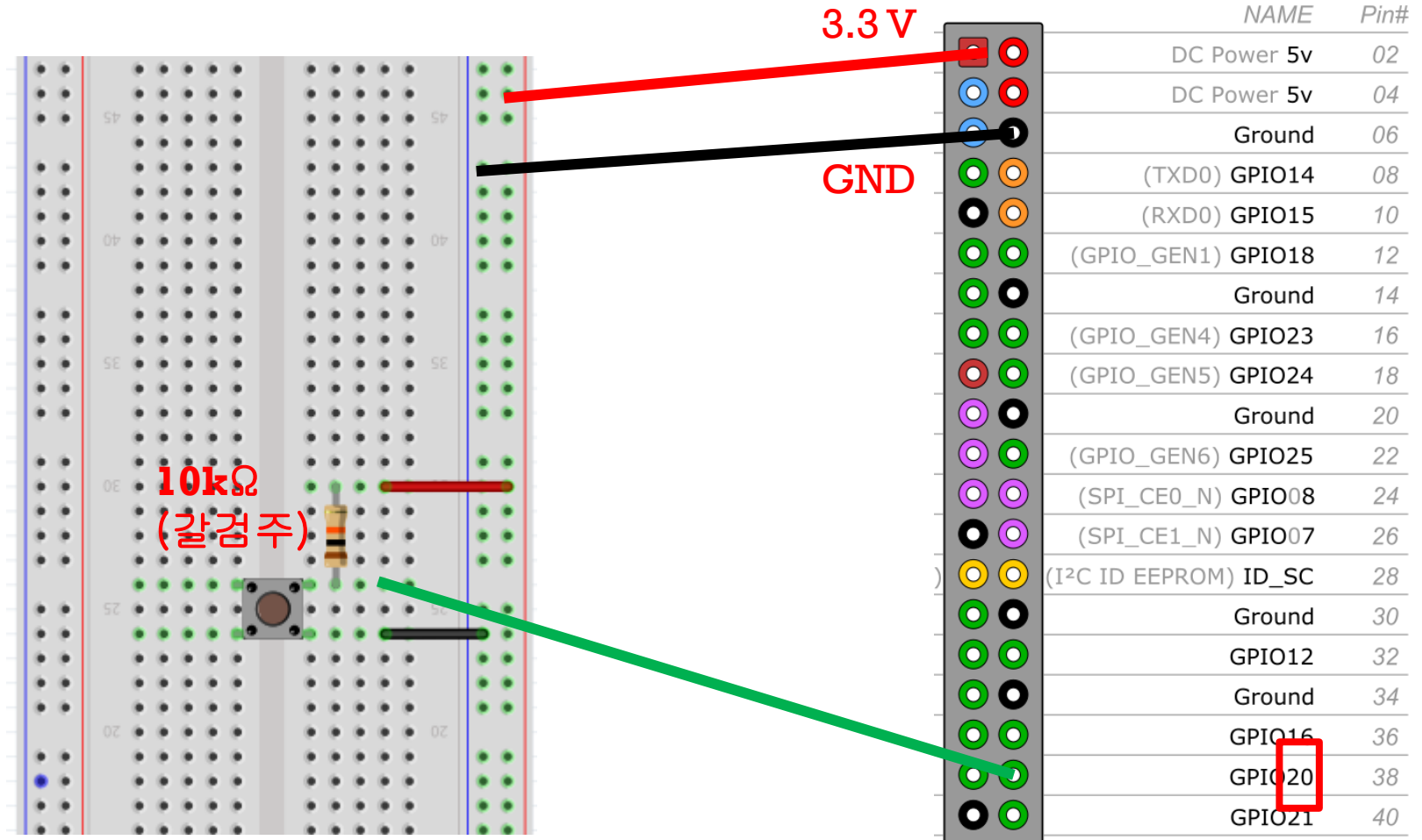
- 플로팅 상태일때의 값을 끌어올린다는 의미를 가진다.
- 입력 핀과 전원(5V)사이를 저항을 두고 연결한다.
- 버튼이 눌러지지 않으면 입력 핀의 전압은 HIGH(3.3V)가 된다.
- 버튼을 누르면 LOW(0V)

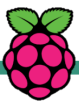




2. LED와 스위치

➤ 풀업(pullup) 스위치





2. LED와 스위치

3. 스위치

3.5 스위치로 LED 제어 – 버튼을 누르면 LED 켜기

File name: SW_02.py

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

```
led = 12
```

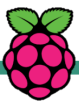
```
switch= 20
```

```
# GPIO 입출력 선언
```

```
GPIO.setmode(GPIO.BCM)
```

```
GPIO.setup(switch, GPIO.IN)
```

```
GPIO.setup(led, GPIO.OUT, initial=GPIO.LOW)
```



2. LED와 스위치

File name: SW_02.py

```
try:
    while True:
        # 스위치 상태를 읽어 변수에 할당
        switch_in = GPIO.input(switch)

        if switch_in == 0:
            GPIO.output(led, GPIO.HIGH)
        else:
            GPIO.output(led, GPIO.LOW)

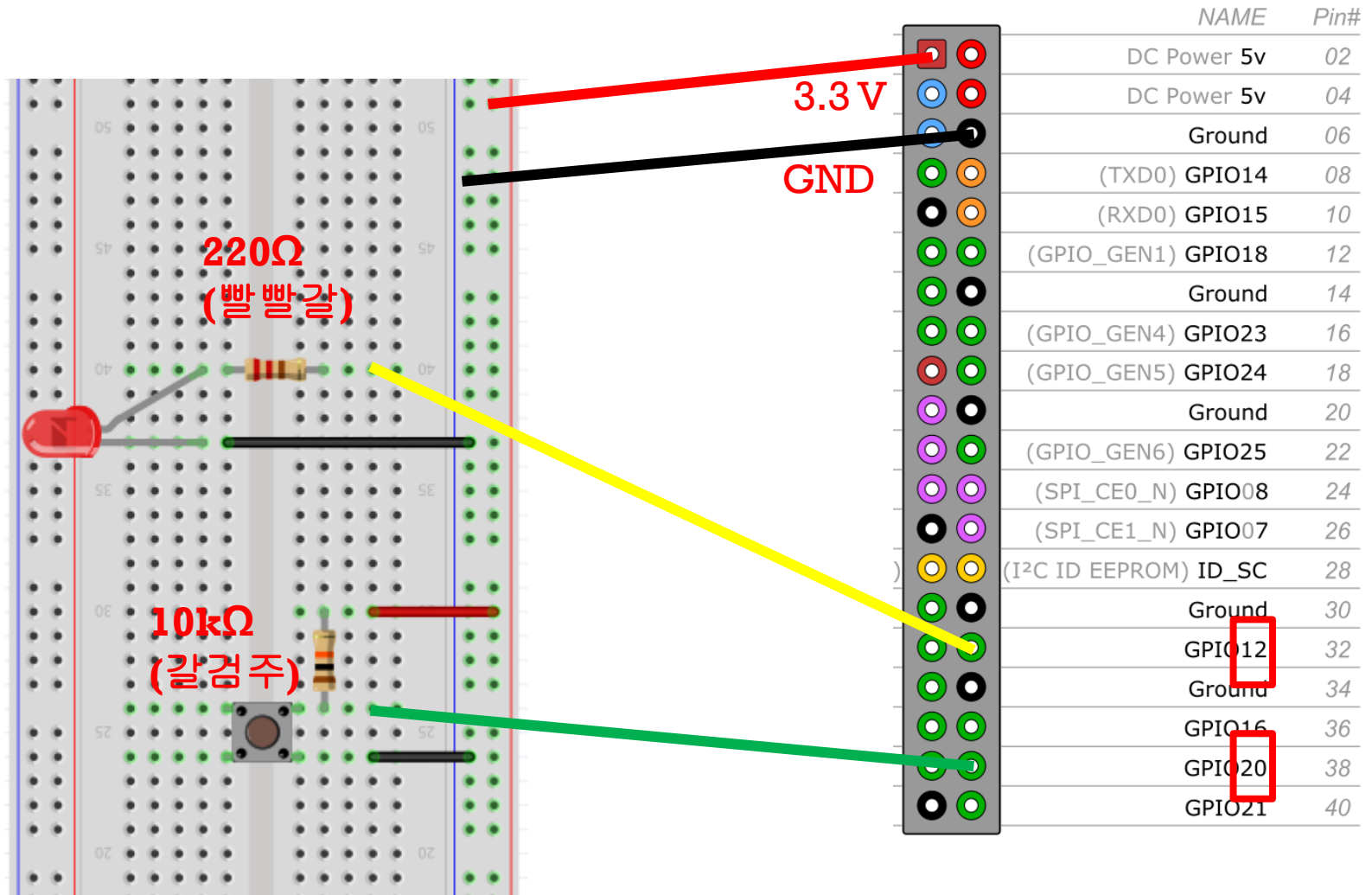
except KeyboardInterrupt:
    pass

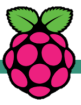
GPIO.cleanup()
```



2. LED와 스위치

- 스위치로 LED 제어 - 버튼을 누르면 LED 켜기



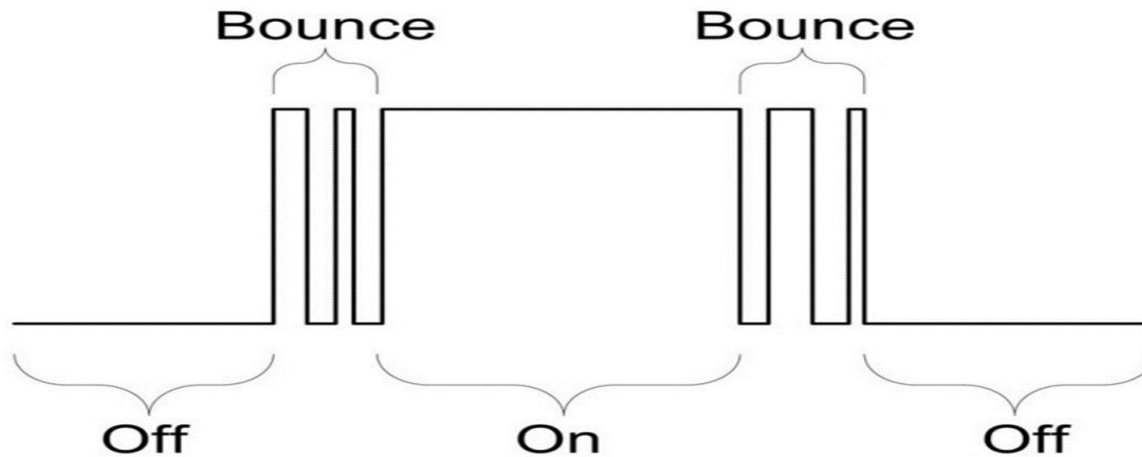


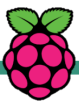
2. LED와 스위치

3. 스위치

3.6 채터링(Chattering)

- 키가 눌러지거나 떼는 순간 아주 짧은 시간 동안 발생하는 전기적 잡음 신호





2. LED와 스위치

3. 스위치

3.6 채터링(Chattering)

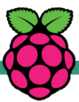
- 스위치로 LED 제어 – 버튼을 누르면 LED 끄거나 켜기 (토글 스위치)

File name: SW_03.py

```
import RPi.GPIO as GPIO
from time import sleep
```

```
led = 12
switch = 20
state = 0
```

```
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(sw, GPIO.IN)
GPIO.setup(led, GPIO.OUT, initial=GPIO.LOW)
```

2. LED와 스위치

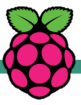
File name: SW_03.py

```
try:
    while True:
        if GPIO.input(switch) == 0:

            while True:
                if GPIO.input(switch) == 0:
                    state = not state
                    GPIO.output(led, state)
                    sleep(0.2)
                    break

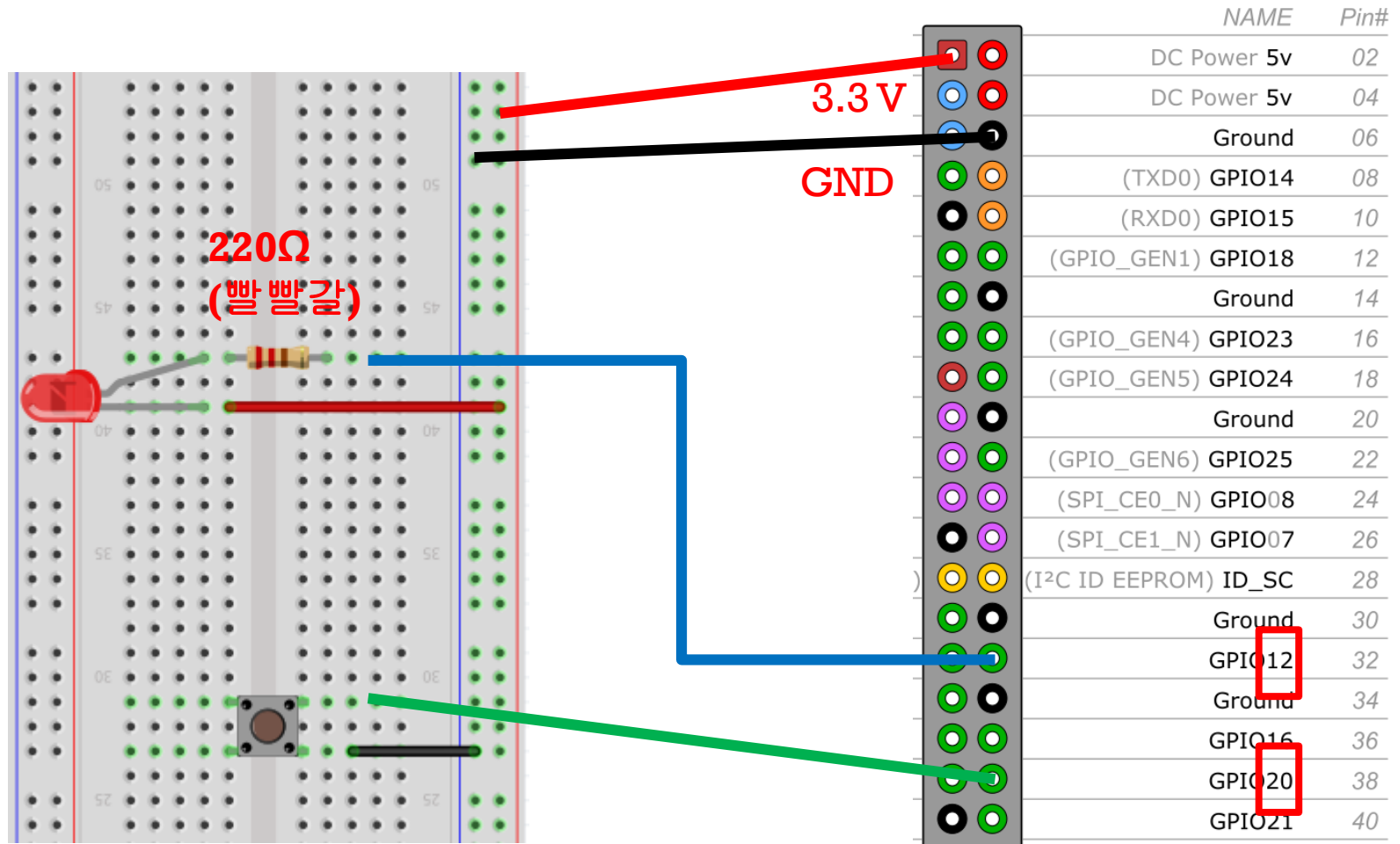
except KeyboardInterrupt:
    pass

GPIO.cleanup()
```



2. LED와 스위치

- 스위치로 LED 제어 - 버튼을 누르면 LED 끄거나 켜기 (내부 풀업 저항 사용)





2. LED와 스위치

3. 스위치

3.6 채터링(Chattering)

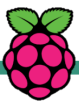
- 스위치로 LED 제어 – 버튼을 누르면 LED 끄거나 켜기 (내부 풀업 저항 사용)

File name: SW_04.py

```
import RPi.GPIO as GPIO
from time import sleep
```

```
led = 12
switch = 20
state = 0
```

```
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(switch, GPIO.IN, pull_up_down = GPIO.PUD_UP)
GPIO.setup(led, GPIO.OUT, initial=GPIO.LOW)
```



2. LED와 스위치

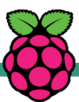
File name: SW_04.py

```
try:
    while True:
        if GPIO.input(switch) == 0:

            while True:
                if GPIO.input(switch) == 0:
                    state = not state
                    GPIO.output(led, state)
                    sleep(0.2)
                    break

except KeyboardInterrupt:
    pass

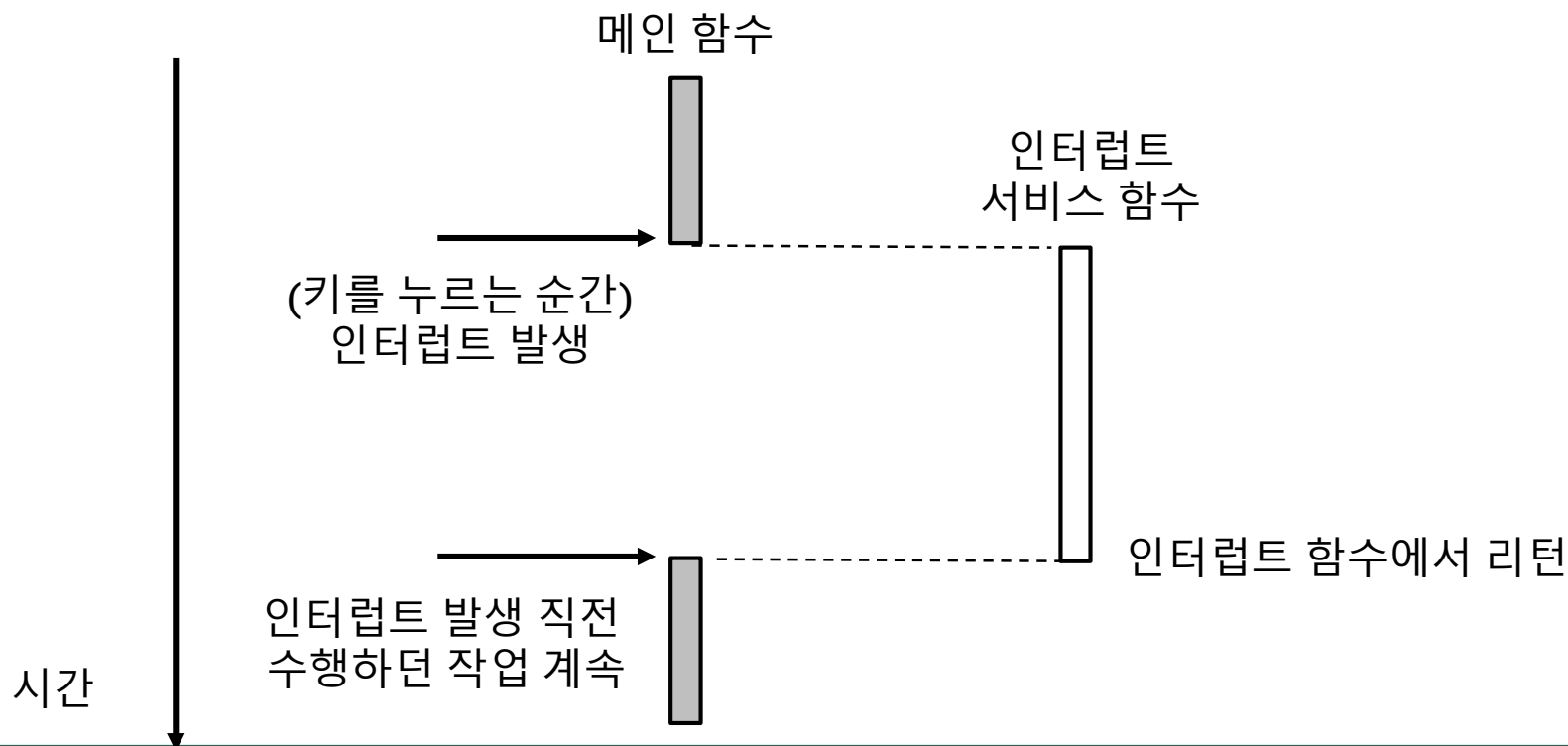
GPIO.cleanup()
```

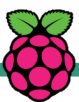


3. 인터럽트 제어

1. 인터럽트제어

- 인터럽트 – 기존에 CPU에서 처리하던 프로그램을 중단하고 인터럽트를 요청한 프로그램으로 실행 제어권을 넘기는 것
- ISR(interrupt Service Routine) – 인터럽트 발생으로 수행되는 함수





3. 인터럽트 제어

1. 인터럽트제어

- 다수의 프로그램들이 동시에 수행되는 것처럼 느껴지는 멀티 태스킹 효과를 얻을 수 있다.

폴링에지(Falling Edge) : 전기적 신호가 HIGH에서 LOW로 떨어지는 순간

라이징에지(Rising Edge) : 전기적 신호가 LOW에서 HIGH로 올라가는 순간

콜백(Callback) 함수 : 직접 명시적으로 호출하는 것이 아니라 운영체제에 의해 호출되는 함수 - ISR 함수는 콜백함수

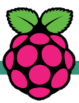
```
GPIO.add_event_detect(channel, edge, callback, bounce_time)
```

channel 핀번호

edge GPIO.FALLING/GPIO.RISING/GPIO.BOTH

callback ISR로 사용할 함수

bounce_time 전기적 잡음을 무시할 시간(msec)



3. 인터럽트 제어

1. 인터럽트제어

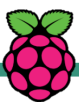
- 폴링 방식

```
while True:
    if GPIO.input(PIN_NUM) == 1 :
        GPIO.output(BUZ, GPIO.HIGH)
        sleep(0.1)
    else :
        GPIO.output(BUZ, GPIO.LOW)
```

- 인터럽트 방식

```
def callback_func1(pin):
    print("pin %d was pressed" %pin)
```

```
GPIO.add_event_detect(pin_num, GPIO.RISING, callback=callback_func1,
bouncetime = 200)
```



3. 인터럽트 제어

1. 인터럽트제어

1.1 스위치를 누르면 메시지 출력

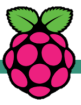
File name: interrupt_01.py

```
import RPi.GPIO as GPIO
from time import sleep
```

```
switch = 20
GPIO.setmode(GPIO.BCM)
GPIO.setup(switch, GPIO.IN, pull_up_down = GPIO.PUD_UP)
```

```
def KeyHandler(pin):    # 인터럽트 서비스 함수
    print("Key is pressed [%d]" % pin)
```

```
# 인터럽트 서비스 함수 등록
GPIO.add_event_detect(switch, GPIO.FALLING, callback = KeyHandler)
```

3. 인터럽트 제어

File name: interrupt_01.py

```
num = 0
```

```
try:
```

```
    while True:
```

```
        print("sec : %d" %num)
```

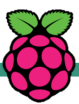
```
        num = num + 1
```

```
        sleep(1)
```

```
except KeyboardInterrupt:
```

```
    pass
```

```
GPIO.cleanup()
```



3. 인터럽트 제어

1. 인터럽트제어

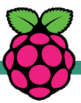
1.2 토글스위치

File name: toggle_01.py

```
import RPi.GPIO as GPIO
from time import sleep
```

```
led = 12
switch = 20
led_on = 0
```

```
GPIO.setmode(GPIO.BCM)
GPIO.setup(switch, GPIO.IN, pull_up_down = GPIO.PUD_UP)
GPIO.setup(led, GPIO.OUT, initial=GPIO.LOW)
```



3. 인터럽트 제어

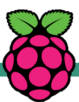
File name: `toggle_01.py`

```
try:
    while True:
        switch_in = GPIO.input(switch)

        if switch_in == 0:
            led_on = led_on ^ 1
            GPIO.output(led, led_on)
            sleep(0.2)

except KeyboardInterrupt:
    pass

GPIO.cleanup()
```



3. 인터럽트 제어

1. 인터럽트제어

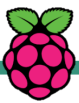
1.2 토글스위치 - 인터럽트제어

File name: interrupt_02.py

```
import RPi.GPIO as GPIO
from time import sleep
```

```
led_on = 0
led = 12
switch = 20
```

```
GPIO.setmode(GPIO.BCM)
GPIO.setup(switch, GPIO.IN, pull_up_down = GPIO.PUD_UP)
GPIO.setup(led, GPIO.OUT, initial=GPIO.LOW)
```



3. 인터럽트 제어

File name: `interrupt_02.py`

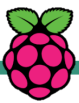
```
def keyHandler(n):
    global led_on
    led_on = led_on ^ 1

GPIO.add_event_detect(switch, GPIO.FALLING, callback=keyHandler, \
                      bouncetime = 100)

try:
    while True:
        GPIO.output(led, led_on)

except KeyboardInterrupt:
    pass

GPIO.cleanup()
```



3. 인터럽트 제어

1. 인터럽트제어

1.3 버튼을 누르면 LED 켜기(On, Off 표시하기)

File name: interrupt_03.py

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

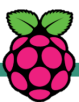
```
led = 12
switch= 20
```

```
# GPIO 입출력 선언
GPIO.setmode(GPIO.BCM)
GPIO.setup(switch, GPIO.IN, pull_up_down = GPIO.PUD_UP)
GPIO.setup(led, GPIO.OUT, initial=GPIO.LOW)
```

```
root = tkinter.Tk()
```

```
# 원을 그리기 위해 캔버스 객체 인스턴스 작성
c = tkinter.Canvas(root, width = 200, height = 200)
```

```
c.pack()
```



3. 인터럽트 제어

File name: interrupt_03.py

원 작성

```
cc = c.create_oval(50, 50, 150, 150, fill = '')
```

스위치가 눌러지면 실행할 함수

```
def check_switch(channel):
```

```
    switch_in = GPIO.input(channel)
```

```
    if switch_in == 0 :
```

```
        GPIO.output(led, GPIO.HIGH)
```

```
        c.itemconfig(cc, fill = 'red')
```

```
    else:
```

```
        GPIO.output(led, GPIO.LOW)
```

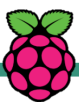
```
        c.itemconfig(cc, fill = '')
```

스위치 상태가 변할 때 check_switch 함수를 호출

```
GPIO.add_event_detect(switch, GPIO.BOTH, callback = check_switch, \n                      bouncetime = 100)
```

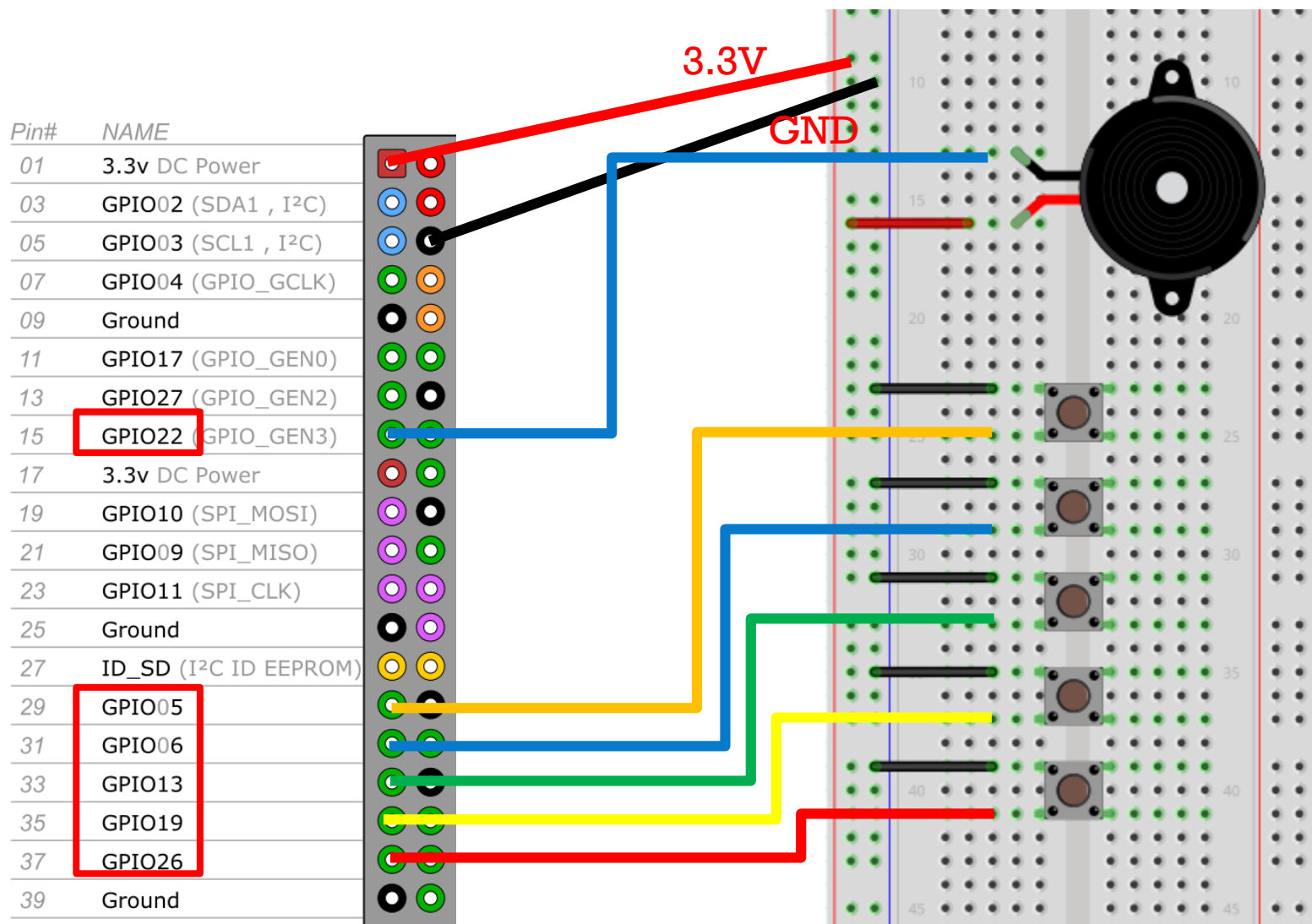
```
root.mainloop()
```

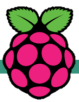
```
GPIO.cleanup()
```



3. 인터럽트 제어

3. 피아노연주





3. 인터럽트 제어

3. 부저 소리내기

File name: `buzzer_1.py`

```
import RPi.GPIO as GPIO
from time import sleep
```

```
buzzer = 22
```

```
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(buzzer, GPIO.OUT)
```

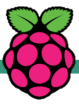
```
P = GPIO.PWM(buzzer, 1)
```

```
p.start(50)
```

```
p.ChangeFrequency(262)
sleep(3)
```

```
p.stop()
```

```
GPIO.cleanup()
```



3. 인터럽트 제어

File name: `buzzer_2.py`

```
import RPi.GPIO as GPIO
from time import sleep
```

```
buzzer = 22
```

```
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(buzzer, GPIO.OUT)
```

```
P = GPIO.PWM(buzzer, 1)
```

```
p.start(50)
```

```
p.ChangeFrequency(262)
sleep(1)
```

```
p.ChangeFrequency(294)
sleep(1)
```

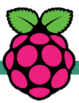
```
p.ChangeFrequency(330)
sleep(1)
```

```
p.ChangeFrequency(349)
sleep(1)
```

```
p.ChangeFrequency(392)
sleep(1)
```

```
p.stop()
```

```
GPIO.cleanup()
```



3. 인터럽트 제어

File name: `buzzer_3.py`

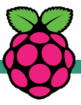
```
import RPi.GPIO as GPIO
from time import sleep
```

```
buzzer = 22
switch = 20
```

```
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(buzzer, GPIO.OUT)
GPIO.setup(switch, GPIO.IN, pull_up_down = GPIO.PUD_UP)
```

```
P = GPIO.PWM(buzzer, 1)
```

```
p.start(50)
```



3. 인터럽트 제어

File name: buzzer_3.py

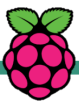
```
try:
    while True:
        if GPIO.input(switch) == GPIO.LOW:
            p.ChangeDutyCycle(50)
            p.ChangeFrequency(262)

        else:
            p.ChangeDutyCycle(0)

except KeyboardInterrupt:
    pass

finally:
    p.stop()

GPIO.cleanup()
```



3. 인터럽트 제어

File name: `buzzer_4.py`

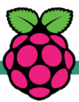
```
import RPi.GPIO as GPIO
from time import sleep
```

```
buzzer = 22
switch = 20
```

```
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(buzzer, GPIO.OUT)
GPIO.setup(switch, GPIO.IN, pull_up_down = GPIO.PUD_UP)
```

```
P = GPIO.PWM(buzzer, 1)
```

```
p.start(0)
```



3. 인터럽트 제어

File name: `buzzer_4.py`

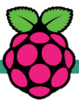
```
def sound_buz(pin):
    if GPIO.input(pin) == GPIO.LOW:
        p.ChangeDutyCycle(50)
        p.ChangeFrequency(262)
    else:
        p.ChangeDutyCycle(0)

GPIO.add_event_detect(switch, GPIO.BOTH, callback = sound_buz, \
    bouncetime = 100)

try:
    while True:
        pass

finally:
    p.stop()

GPIO.cleanup()
```



3. 인터럽트 제어

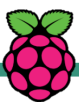
4. 피아노 연주

File name: `piano_1.py`

```
import RPi.GPIO as GPIO
from time import sleep
```

```
buzzer = 22
do = 5
re = 6
mi = 13
fa = 19
sol = 26
freq = [262, 294, 330, 349, 392]
# scale = [5, 6, 13, 19, 26]
```

```
GPIO.setmode(GPIO.BCM)
GPIO.setup(buzzer, GPIO.OUT)
GPIO.setup(do, GPIO.IN, pull_up_down = GPIO.PUD_UP)
GPIO.setup(re, GPIO.IN, pull_up_down = GPIO.PUD_UP)
GPIO.setup(mi, GPIO.IN, pull_up_down = GPIO.PUD_UP)
GPIO.setup(fa, GPIO.IN, pull_up_down = GPIO.PUD_UP)
GPIO.setup(sol, GPIO.IN, pull_up_down = GPIO.PUD_UP)
```



3. 인터럽트 제어

File name: piano_1.py

```
p = GPIO.PWM(buzzer, 1)
p.start(0)
```

```
try:
```

```
    while True:
```

```
        if GPIO.input(do) == GPIO.LOW :
            p.ChangeDutyCycle(50)
            p.ChangeFrequency(freq[0])
```

```
        elif GPIO.input(re) == GPIO.LOW :
            p.ChangeDutyCycle(50)
            p.ChangeFrequency(freq[1])
```

```
        elif GPIO.input(mi) == GPIO.LOW :
            p.ChangeDutyCycle(50)
            p.ChangeFrequency(freq[2])
```

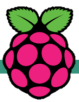
```
        elif GPIO.input(fa) == GPIO.LOW :
            p.ChangeDutyCycle(50)
            p.ChangeFrequency(freq[3])
```

```
        elif GPIO.input(sol) == GPIO.LOW :
            p.ChangeDutyCycle(50)
            p.ChangeFrequency(freq[4])
```

```
        else:
            p.ChangeDutyCycle(0)
```

```
    finally:
        p.stop()
```

```
GPIO.cleanup()
```

3. 인터럽트 제어

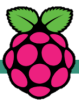
4. 피아노 연주

File name: `piano_2.py`

```
import RPi.GPIO as GPIO
from time import sleep
```

```
buzzer = 22
do = 5
re = 6
mi = 13
fa = 19
sol = 26
```

```
GPIO.setmode(GPIO.BCM)
GPIO.setup(buzzer, GPIO.OUT)
GPIO.setup(do, GPIO.IN, pull_up_down = GPIO.PUD_UP)
GPIO.setup(re, GPIO.IN, pull_up_down = GPIO.PUD_UP)
GPIO.setup(mi, GPIO.IN, pull_up_down = GPIO.PUD_UP)
GPIO.setup(fa, GPIO.IN, pull_up_down = GPIO.PUD_UP)
GPIO.setup(sol, GPIO.IN, pull_up_down = GPIO.PUD_UP)
```



3. 인터럽트 제어

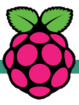
File name: piano_2.py

```
freq = [262, 294, 330, 349, 392]
dic_pin = {5:0, 6:1, 13:2, 19:3, 26:4}
```

```
p = GPIO.PWM(buzzer, 1)
```

```
p.start(0)
```

```
def sound_buz(pin):
    if GPIO.input(pin) == GPIO.LOW:
        p.ChangeDutyCycle(50)
        p.ChangeFrequency(freq[dic_pin[pin]])
    else :
        p.ChangeDutyCycle(0)
```



3. 인터럽트 제어

File name: piano_2.py

```
GPIO.add_event_detect(do, GPIO.BOTH, callback = sound_buz, bouncetime = 100)
GPIO.add_event_detect(re, GPIO.BOTH, callback = sound_buz, bouncetime = 100)
GPIO.add_event_detect(mi, GPIO.BOTH, callback = sound_buz, bouncetime = 100)
GPIO.add_event_detect(fa, GPIO.BOTH, callback = sound_buz, bouncetime = 100)
GPIO.add_event_detect(sol, GPIO.BOTH, callback = sound_buz, bouncetime = 100)

try:
    while True:
        pass

finally:
    p.stop()
    GPIO.cleanup()
```