

## CH. 2. 라즈베리파이 GPIO

# Section 01 GPIO를 이용한 LED 제어하기

---

## □ GPIO (General-Purpose Input/Output)

- 주변장치와 통신하기 위해 범용으로 사용되는 입출력 핀(포트)
- GPIO 핀은 디지털 입력 혹은 출력에 사용될 수 있고, 입력과 출력 모두 3.3V로 동작한다.
- 입력과 출력을 마음대로 선택할 수 있고, 0과 1의 출력 신호를 임의로 만들어 줄 수 있다.
- 아두이노와 달리 라즈베리 파이는 아날로그 입력을 지원하지 않는다.
- 아날로그 입력을 사용하기 위해 ADC (Analog to Digital converter)를 사용해야 한다.

# Section 01 GPIO를 이용한 LED 제어하기

---

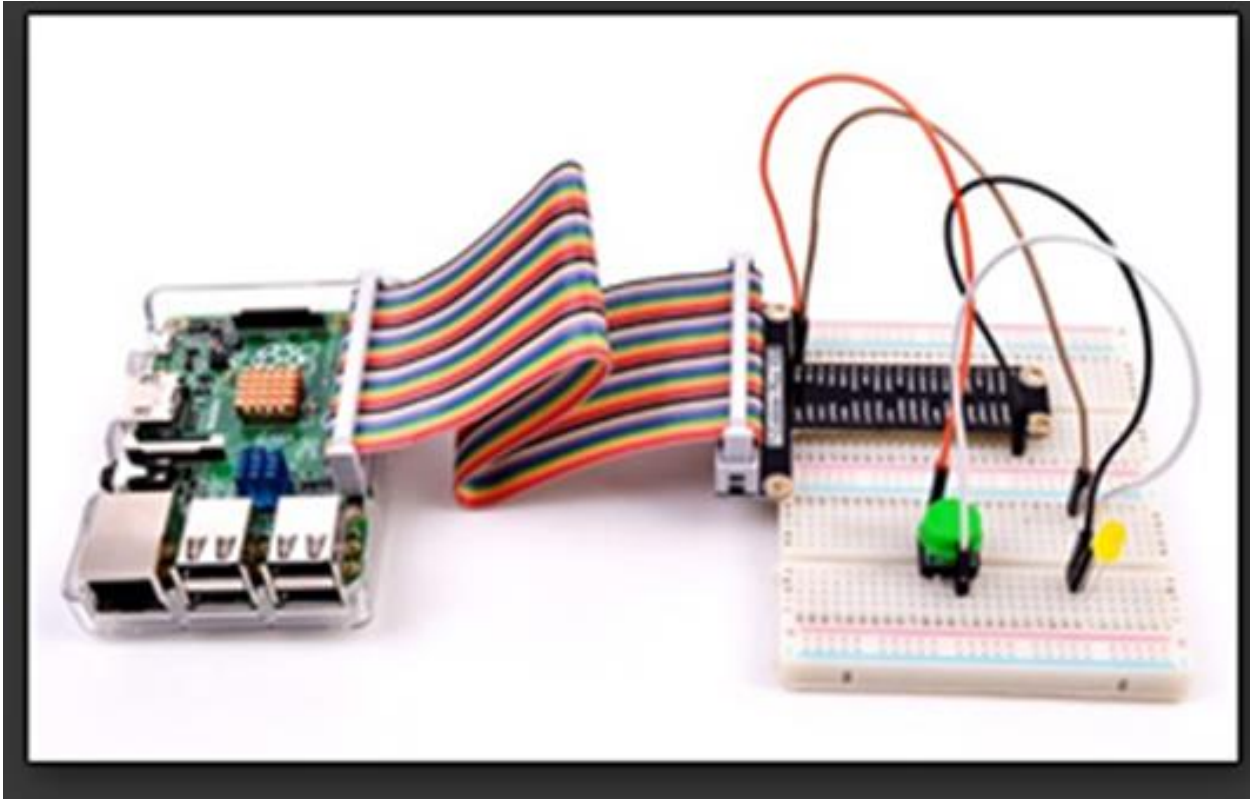
## □ GPIO의 용도, 할 수 있는 것

- 센서라든지 다른 컴퓨터 및 장치로부터 오는 신호를 받아 상호작용하는 프로그램을 만들 수 있다.
- 출력 또한 LED를 켜는 것에서부터 신호나 데이터를 다른 장치에 보내는 것까지 무엇이든 될 수 있다.
- 만약 라즈베리 파이가 네트워크에 연결되어 있다면, 접속된 장치들은 어디에서든 제어할 수 있고, 해당 장치가 데이터를 보내도록 할 수 있다.
- 인터넷 상에서 물리적 장치에 대한 연결과 제어를 할 수 있다는 것이 강력한 특징이다.

# Section 01 GPIO를 이용한 LED 제어하기

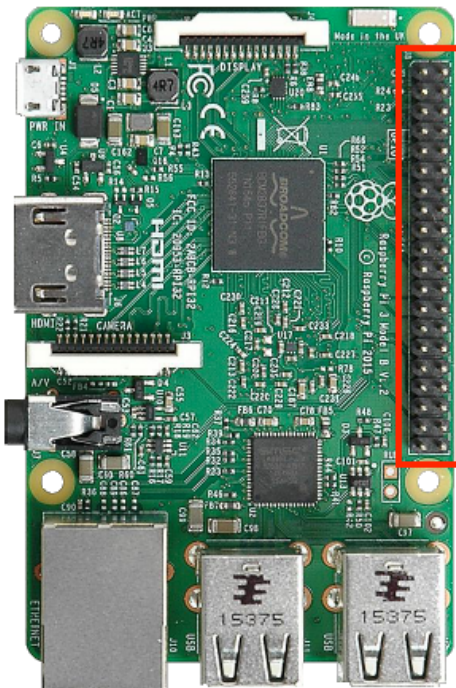
---

## □ GPIO 기본 연결 방법



# Section 01 GPIO를 이용한 LED 제어하기

## □ GPIO (General-Purpose Input/Output)



라즈베리파이3 Model B

Raspberry Pi 3 GPIO Header

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

## Section 01 GPIO를 이용한 LED 제어하기

## ❑ GPIO (General-Purpose Input/Output)



## → 공식 GPIO

WiringPi 를 사용하지 않고  
직접 GPIO 에서 연결하는  
센서 혹은 하드웨어 연결 시  
해당 GPIO 번호를 이용하여  
사용함

## WiringPi GPIO ←

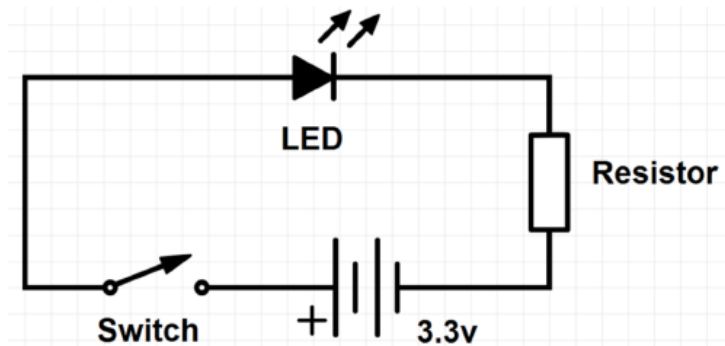
Wiring Pi 를 이용하는 경우,  
실제 사용하는 Pin 번호와  
Wiring Pi 가 사용하는 Pin  
번호가 차이가 있으므로 사  
용시 주의가 필요함

Raspberry Pi J8 Header (Model B+)					
GPIO#	NAME		NAME	GPIO#	
	3.3 VDC Power	1		2	5.0 VDC Power
8	GPIO 8 SDA1 (I2C)	3		4	5.0 VDC Power
9	GPIO 9 SCL1 (I2C)	5		6	Ground
7	GPIO 7 GPCLK0	7		8	GPIO 15 Tx0 (RS232) 15
	Ground	9		10	GPIO 16 Rx0 (RS232) 16
0	GPIO 0	11		12	GPIO 1 PCM_CLK/PWM0 1
2	GPIO 2	13		14	Ground
3	GPIO 3	15		16	GPIO 4 4
	3.3 VDC Power	17		18	GPIO 5 5
12	GPIO 12 MOSI (SPI)	19		20	Ground
13	GPIO 13 MISO (SPI)	21		22	GPIO 6 6
14	GPIO 14 SCLK (SPI)	23		24	GPIO 10 CE0 (SPI) 10
	Ground	25		26	GPIO 11 CE1 (SPI) 11
	SDA0 (I2C ID EEPROM)	27		28	SCL0 (I2C ID EEPROM)
21	GPIO 21 GPCLK1	29		30	Ground
22	GPIO 22 GPCLK2	31		32	GPIO 26 PWM0 26
23	GPIO 23 PWM1	33		34	Ground
24	GPIO 24 PCM_FS/PWM1	35		36	GPIO 27 27
25	GPIO 25	37		38	GPIO 28 PCM_DIN 28
	Ground	39		40	GPIO 29 PCM_DOUT 29

<http://www.pi4j.com>

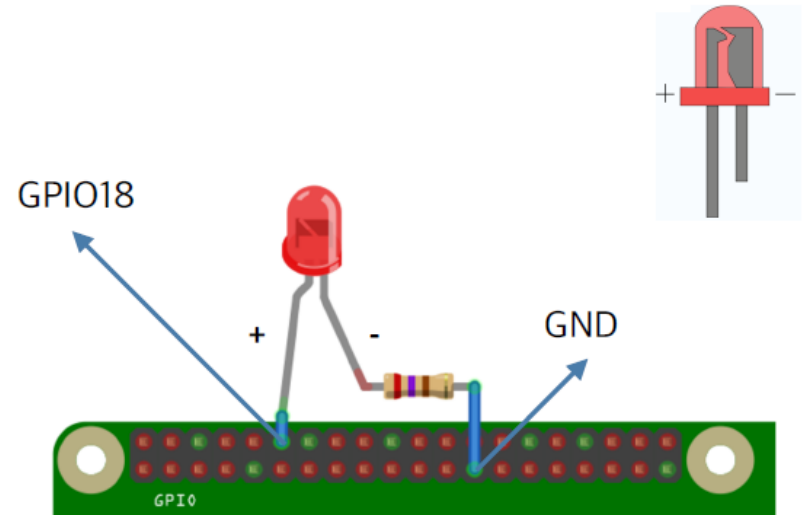
# Section 01 GPIO를 이용한 LED 제어하기

## □ GPIO 작동 원리



LED를 켜고 끌 수 있는 가장 단순한 전기회로

=



라즈베리파이를 사용하여 만든 동일한 회로

- GPIO가 스위치와 배터리의 기능을 모두 담당한다.
- 각각의 핀은 켜고 끌 수 있다. (켜 : HIGH, 끄 : LOW)
- HIGH일때 출력은 3.3V, LOW일때 출력은 0V

# Section 01 GPIO를 이용한 LED 제어하기

## □wiringPi 설치

- `sudo apt-get update`와 `sudo apt-get upgrade` 명령을 통해 라즈베리 파이의 업데이트 및 업그레이드를 수행.
- 중간에 추가 설치를 위해 용량이 필요하다는 문구가 나오면 Y키를 입력.

```
sudo apt-get update  
sudo apt-get upgrade
```

- `git clone https://github.com/WiringPi/WiringPi` 의 명령어로 라이브러리를 다운로드.

```
pi@raspberrypi:~ $ git clone https://github.com/WiringPi/WiringPi  
Cloning into 'WiringPi'...  
remote: Enumerating objects: 1385, done.  
remote: Total 1385 (delta 0), reused 0 (delta 0), pack-reused 1385  
Receiving objects: 100% (1385/1385), 713.54 KiB | 885.00 KiB/s, done.  
Resolving deltas: 100% (861/861), done.
```



# Section 01 GPIO를 이용한 LED 제어하기

## □wiringPi 설치

- cd WiringPi로 디렉토리를 변경.

```
pi@raspberrypi:~ $ cd WiringPi/  
pi@raspberrypi:~/WiringPi $
```

- ./build로 WiringPi 라이브러리를 빌드.

```
pi@raspberrypi:~/WiringPi $ ./build  
wiringPi Build script
```

- gpio -v를 통해 라이브러리의 성공 여부를 확인.

```
pi@raspberrypi:~/WiringPi $ gpio -v  
gpio version: 2.60  
Copyright (c) 2012-2018 Gordon Henderson  
This is free software with ABSOLUTELY NO WARRANTY.  
For details type: gpio -warranty  
  
Raspberry Pi Details:  
Type: Pi 3, Revision: 02, Memory: 1024MB, Maker: Sony  
* Device tree is enabled.  
*--> Raspberry Pi 3 Model B Rev 1.2  
* This Raspberry Pi supports user-level GPIO access.
```

# Section 01 GPIO를 이용한 LED 제어하기

## □ LED 제어 실습을 위한 준비물

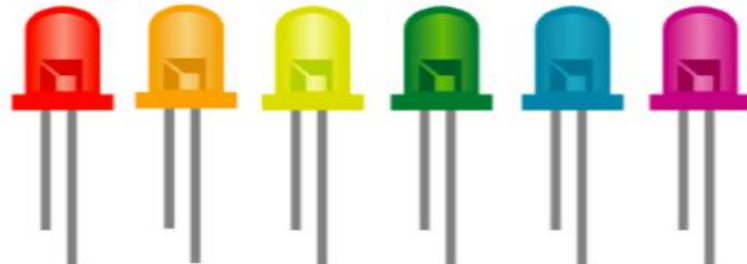
- 점퍼선 or 점퍼와이어



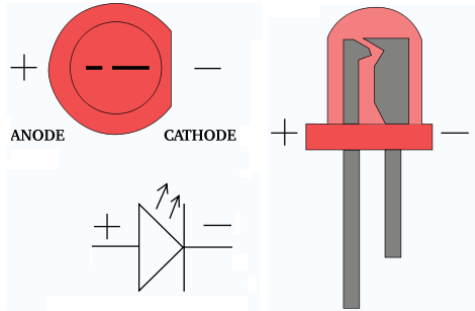
- 저항



- LED



# Section 01 GPIO를 이용한 LED 제어하기



단자가 긴 쪽이 (+)

LED소자의 꺾인 쪽이 (-)

## LED(Light Emitting Diode)

발광 다이오드는 순방향으로 전압을 가했을 때 발광하는 반도체 소자이다. 발광 원리는 전계 발광 효과를 이용하고 있다. 또한 수명도 백열등보다 매우 길다.

발광색은 사용되는 재료에 따라서 다르며 자외선 영역에서 가시광선, 적외선 영역까지 발광하는 것을 제조할 수 있다. 일리노이 대학의 닉 호로니악이 1962년에 최초로 개발하였다.

# Section 01 GPIO를 이용한 LED 제어하기

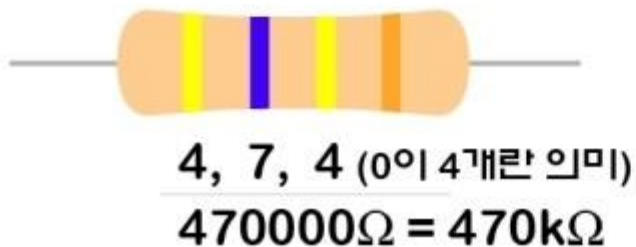
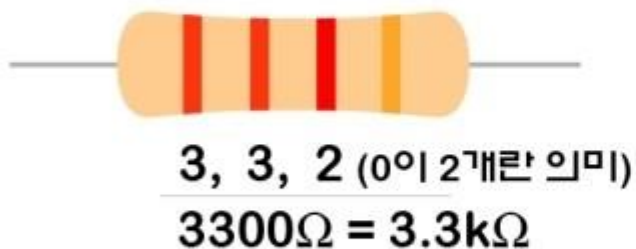
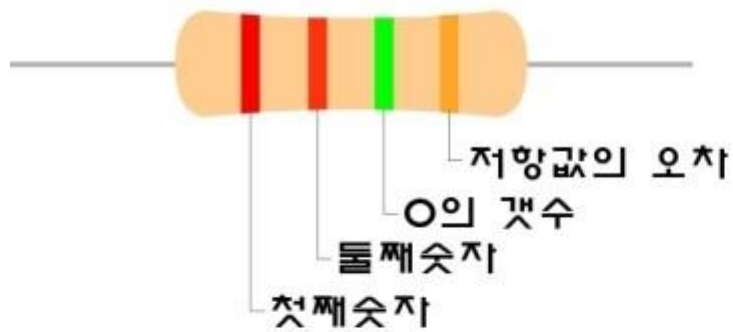
---

## □ 저항 (register)

- 성능이 낮은 전도체의 양쪽 끝 또는 측면에 도선이나 전도체 2개가 붙어있는 형태로 되어 있음 단위는 옴(ohm)이며, 기호로는 그리스어의 오메가를 사용
- 용도
  - 캐패시터 충전을 제한
  - 양극성 트랜지스터와 같은 반도체 부품의 전압 제어
  - LED 또는 기타 반도체 부품의 과다 전류 방지
  - 다른 부품과 결합하여 사용하는 오디오 회로에서 주파수 응답의 조정 또는 제한
  - 디지털 논리 회로에 입력되는 전압의 풀업 저항 또는 풀다운 저항용
  - 회로 내 한 지점에서의 전압 제어용

# Section 01 GPIO를 이용한 LED 제어하기

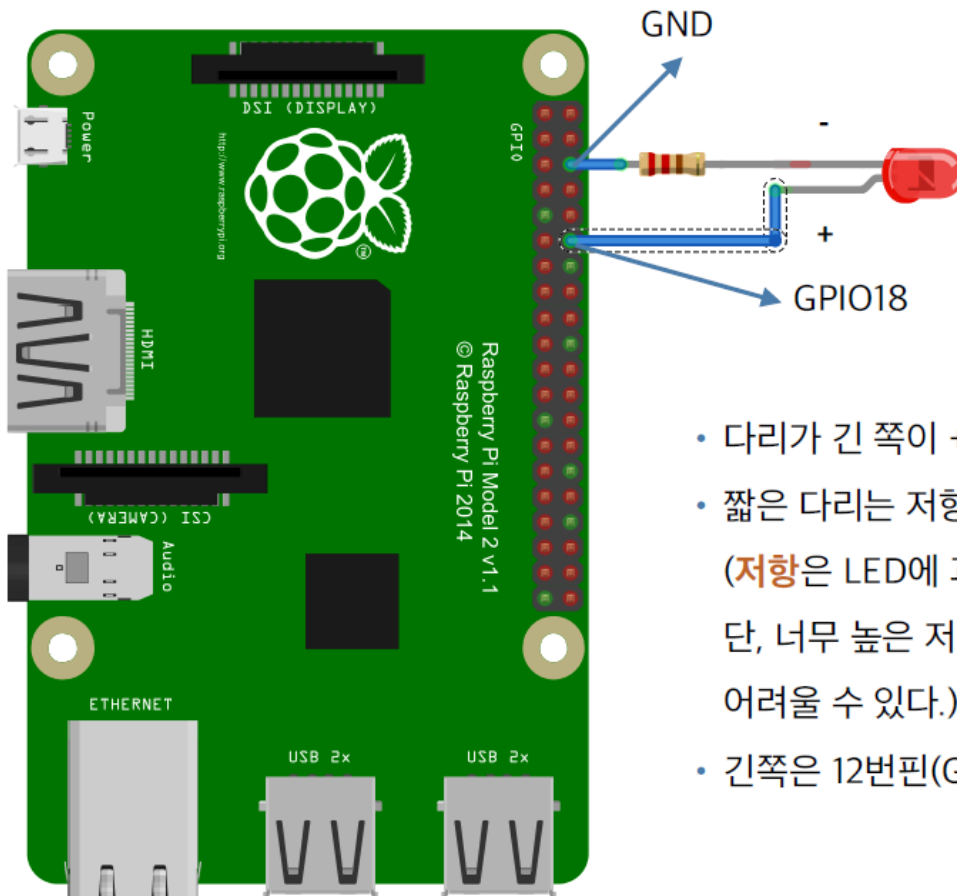
## □ 저항 (register)



색	값
	검정색
	갈 색
	빨강색
	주황색
	노란색
	초록색
	파란색
	보라색
	회 색
	하얀색
	은 색
	금 색
	0
	1
	2
	3
	4
	5
	6
	7
	8
	9
	$\pm 10\%$
	$\pm 5\%$

# Section 01 GPIO를 이용한 LED 제어하기

## □ 라즈베리 파이 회로 구성



- 다리가 긴 쪽이 +극, 다리가 짧은 쪽이 -극
- 짧은 다리는 저항과 직렬 연결한 후 6번핀(GND)에 연결  
(저항은 LED에 과전류가 흐르는 것을 막아주는 역할을 한다.  
단, 너무 높은 저항을 사용하면 LED의 불빛을 눈으로 확인하기  
어려울 수 있다.)
- 긴쪽은 12번핀(GPIO18)과 연결한다.

# Section 01 GPIO를 이용한 LED 제어하기

---

## □ 파이썬 코딩

- GPIO 모듈 불러오기

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

# Section 01 GPIO를 이용한 LED 제어하기

## □ 파이썬 코딩


### • 핀번호 할당 방법 정의

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

```
GPIO.setmode(GPIO.BOARD)
```

위와 같이 `setmode`에서  
`GPIO.BOARD`로 정의하면  
오른쪽 그림에서  
01 ~ 40번까지의 숫자로  
핀 번호를 사용하겠다는  
의미임

예를 들어 11번핀은  
`GPIO17`인데, 프로그램에서  
11번으로 사용함



Pin#	NAME		NAME	Pin#
01	3.3v DC Power	Red	DC Power 5v	02
03	GPIO02 (SDA1, I2C)	Blue	DC Power 5v	04
05	GPIO03 (SCL1, I2C)	Blue	Ground	06
07	GPIO04 (GPIO_GCLK)	Green	(TXD0) GPIO14	08
09	Ground	Black	(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)	Green	(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)	Green	Ground	14
15	GPIO22 (GPIO_GEN3)	Green	(GPIO_GEN4) GPIO23	16
17	3.3v DC Power	Red	(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)	Purple	Ground	20
21	GPIO09 (SPI_MISO)	Purple	(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)	Purple	(SPI_CE0_N) GPIO08	24
25	Ground	Black	(SPI_CE1_N) GPIO07	26
27	ID_SD (I2C ID EEPROM)	Yellow	(I2C ID EEPROM) ID_SC	28
29	GPIO05	Green	Ground	30
31	GPIO06	Green	GPIO12	32
33	GPIO13	Green	Ground	34
35	GPIO19	Green	GPIO16	36
37	GPIO26	Green	GPIO20	38
39	Ground	Black	GPIO21	40



# Section 01 GPIO를 이용한 LED 제어하기

## □ 파이썬 코딩

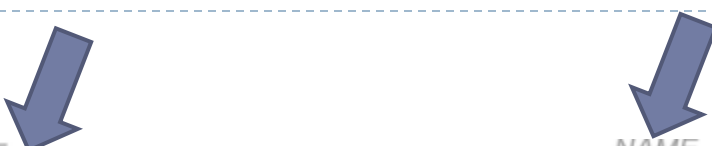
### • 핀번호 할당 방법 정의

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

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

위와 같이 `setmode`에서  
`GPIO.BCM`으로 정의하면  
오른쪽 그림에서  
GPIO이름 뒤에 있는  
번호로 핀 번호를 사용하겠다는  
의미임

예를 들어 11번핀은  
GPIO17인데, 프로그램에서  
17번으로 사용함



Pin#	NAME	NAME	Pin#
01	3.3v DC Power	DC Power 5v	02
03	GPIO02 (SDA1 , I2C)	DC Power 5v	04
05	GPIO03 (SCL1 , I2C)	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 (I2C ID EEPROM)	(I2C 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

# Section 01 GPIO를 이용한 LED 제어하기

## □ 파이썬 코딩

### • 핀번호 용도 및 초기값 설정

```
1 import RPi.GPIO as GPIO
2
3 GPIO.setmode(GPIO. BOARD)
4
5 GPIO.setup(18, GPIO.OUT, initial=GPIO.LOW)
```

위의 값은 18번 핀을 OUT (출력) 목적으로 사용하며  
초기값은 LOW 상태로 시작하도록 설정함

# Section 01 GPIO를 이용한 LED 제어하기

## □ GPIO 주요 함수들

- gpio readall
  - 이 명령은 보드에 대한 핀 정보를 볼 수 있는 그림을 보여준다.
  - 라즈베리파이 핀을 이용할 때 번호를 BCM 또는 wPi를 이용할 수 있다.
  - wiringPi 라이브러리를 이용하면 wPi 번호를 보면 된다.
  - Mode는 핀의 입출력 정보를 V는 핀의 값을 의미한다.

Pi 4B											
BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM	
		3.3v			1	2		5v			
2	8	SDA.1	IN	1	3	4		5v			
3	9	SCL.1	IN	1	5	6		0v			
4	7	GPIO. 7	IN	1	7	8	1	IN	15	14	
		0v			9	10	1	IN	16	15	
17	0	GPIO. 0	IN	0	11	12	0	IN	GPIO. 1	1	18
27	2	GPIO. 2	IN	0	13	14		0v			
22	3	GPIO. 3	IN	0	15	16	0	IN	GPIO. 4	4	23
		3.3v			17	18	0	IN	GPIO. 5	5	24
10	12	MOSI	IN	0	19	20		0v			
9	13	MISO	IN	0	21	22	0	IN	GPIO. 6	6	25
11	14	SCLK	IN	0	23	24	1	IN	CE0	10	8
		0v			25	26	1	IN	CE1	11	7
0	30	SDA.0	IN	1	27	28	1	IN	SCL.0	31	1
5	21	GPIO.21	IN	1	29	30		0v			
6	22	GPIO.22	IN	1	31	32	0	IN	GPIO.26	26	12
13	23	GPIO.23	IN	0	33	34		0v			
19	24	GPIO.24	IN	0	35	36	0	IN	GPIO.27	27	16
26	25	GPIO.25	IN	0	37	38	0	IN	GPIO.28	28	20
		0v			39	40	0	IN	GPIO.29	29	21
BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM	
Pi 4B											



# Section 01 GPIO를 이용한 LED 제어하기

## □ GPIO 주요 함수들

- gpio read [pin]
  - 현재 포트의 값을 읽을 수 있다.
- gpio read 1을 통해 wPi 1번의 값을 읽어올 수 있다.
- gpio readall을 해서 보면 V값이 1인 것을 확인할 수 있다.

```
pi@raspberrypi:~ $ gpio read 1
0
pi@raspberrypi:~ $ gpio readall
```

BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM
		3.3v			1	2		5v		
2	8	SDA.1	IN	1	3	4		5v		
3	9	SCL.1	IN	1	5	6		0v		
4	7	GPIO. 7	IN	1	7	8	1	IN	TxD	15
		0v			9	10	1	IN	RxD	16
17	0	GPIO. 0	IN	0	11	12	0	OUT	GPIO. 1	1
27	2	GPIO. 2	IN	0	13	14		0v		
22	3	GPIO. 3	IN	0	15	16	0	IN	GPIO. 4	4
		3.3v			17	18	0	IN	GPIO. 5	5
10	12	MOSI	IN	0	19	20		0v		
9	13	MISO	IN	0	21	22	0	IN	GPIO. 6	6
11	14	SCLK	IN	0	23	24	1	IN	CE0	10
		0v			25	26	1	IN	CE1	11
0	30	SDA.0	IN	1	27	28	1	IN	SCL.0	31
5	21	GPIO.21	IN	1	29	30		0v		
6	22	GPIO.22	IN	1	31	32	0	IN	GPIO.26	26
13	23	GPIO.23	IN	0	33	34		0v		
19	24	GPIO.24	IN	0	35	36	0	IN	GPIO.27	27
26	25	GPIO.25	IN	0	37	38	0	IN	GPIO.28	28
		0v			39	40	0	IN	GPIO.29	29

```
pi@raspberrypi:~ $
```

# Section 01 GPIO를 이용한 LED 제어하기

## □ GPIO 주요 함수들

- gpio mode [pin] [in/out/clock]
  - 현재 포트의 모드를 설정할 수 있다.
- Gpio mode 1 out을 통해 wPi 1번 핀의 mode가 in에서 out으로 변경되었다.

```
pi@raspberrypi:~$ gpio mode 1 out
pi@raspberrypi:~$ gpio readall
```

Pi 4B											
BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM	
		3.3v			1	2		5v			
2	8	SDA.1	IN	1	3	4		5v			
3	9	SCL.1	IN	1	5	6		0v			
4	7	GPIO. 7	IN	1	7	8	1	IN	15	14	
		0v			9	10	1	IN	16	15	
17	0	GPIO. 0	IN	0	11	12	0	OUT	1	18	
27	2	GPIO. 2	IN	0	13	14		0v			
22	3	GPIO. 3	IN	0	15	16	0	IN	4	23	
		3.3v			17	18	0	IN	5	24	
10	12	MOSI	IN	0	19	20		0v			
9	13	MISO	IN	0	21	22	0	IN	6	25	
11	14	SCLK	IN	0	23	24	1	IN	10	8	
		0v			25	26	1	IN	11	7	
0	30	SDA.0	IN	1	27	28	1	IN	31	1	
5	21	GPIO.21	IN	1	29	30		0v			
6	22	GPIO.22	IN	1	31	32	0	IN	26	12	
13	23	GPIO.23	IN	0	33	34		0v			
19	24	GPIO.24	IN	0	35	36	0	IN	27	16	
26	25	GPIO.25	IN	0	37	38	0	IN	28	20	
		0v			39	40	0	IN	29	21	

```
pi@raspberrypi:~$
```

# Section 01 GPIO를 이용한 LED 제어하기

## □ GPIO 주요 함수들

- `gpio write [pin] [1/0]`
  - 지정한 포트에 값을 쓸 수 있다.
- `gpio write 1 1`을 통해 wpi 1번 포트에 1값을 쓸 수 있다.
- `gpio readall`을 통해 1이 쓰여진 것을 확인할 수 있다.

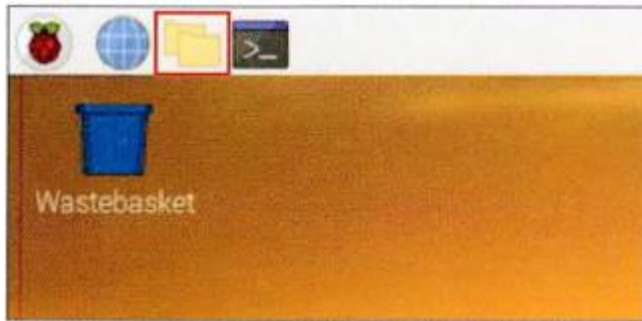
```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
pi@raspberrypi:~ $ gpio write 1 1
pi@raspberrypi:~ $ gpio readall
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| BCM | wPi |   Name   | Mode | V | Physical | V | Mode |   Name   | wPi | BCM |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 2   | 8   | SDA.1    | IN    | 1 | 3   | 4   |      | 5v       |      |      |
| 3   | 9   | SCL.1    | IN    | 1 | 5   | 6   |      | 5v       |      |      |
| 4   | 7   | GPIO. 7  | IN    | 1 | 7   | 8   | 1 IN | TxD      | 15  | 14  |
|     |     | 0v       |       |   | 9   | 10  | 1 IN | RxD      | 16  | 15  |
| 17  | 0   | GPIO. 0  | IN    | 0 | 11  | 12  | 1 OUT| GPIO. 1  | 1   | 18  |
| 27  | 2   | GPIO. 2  | IN    | 0 | 13  | 14  |      | 0v       |      |      |
| 22  | 3   | GPIO. 3  | IN    | 0 | 15  | 16  | 0 IN | GPIO. 4  | 4   | 23  |
|     |     | 3.3v     |       |   | 17  | 18  | 0 IN | GPIO. 5  | 5   | 24  |
| 10  | 12  | MOSI     | IN    | 0 | 19  | 20  |      | 0v       |      |      |
| 9   | 13  | MISO     | IN    | 0 | 21  | 22  | 0 IN | GPIO. 6  | 6   | 25  |
| 11  | 14  | SCLK     | IN    | 0 | 23  | 24  | 1 IN | CE0      | 10  | 8   |
|     |     | 0v       |       |   | 25  | 26  | 1 IN | CE1      | 11  | 7   |
| 0   | 30  | SDA.0    | IN    | 1 | 27  | 28  | 1 IN | SCL.0    | 31  | 1   |
| 5   | 21  | GPIO.21  | IN    | 1 | 29  | 30  |      | 0v       |      |      |
| 6   | 22  | GPIO.22  | IN    | 1 | 31  | 32  | 0 IN | GPIO.26  | 26  | 12  |
| 13  | 23  | GPIO.23  | IN    | 0 | 33  | 34  |      | 0v       |      |      |
| 19  | 24  | GPIO.24  | IN    | 0 | 35  | 36  | 0 IN | GPIO.27  | 27  | 16  |
| 26  | 25  | GPIO.25  | IN    | 0 | 37  | 38  | 0 IN | GPIO.28  | 28  | 20  |
|     |     | 0v       |       |   | 39  | 40  | 0 IN | GPIO.29  | 29  | 21  |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| BCM | wPi |   Name   | Mode | V | Physical | V | Mode |   Name   | wPi | BCM |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
pi@raspberrypi:~ $
```

# Section 01 GPIO를 이용한 LED 제어하기

---

## □ 환경 설정

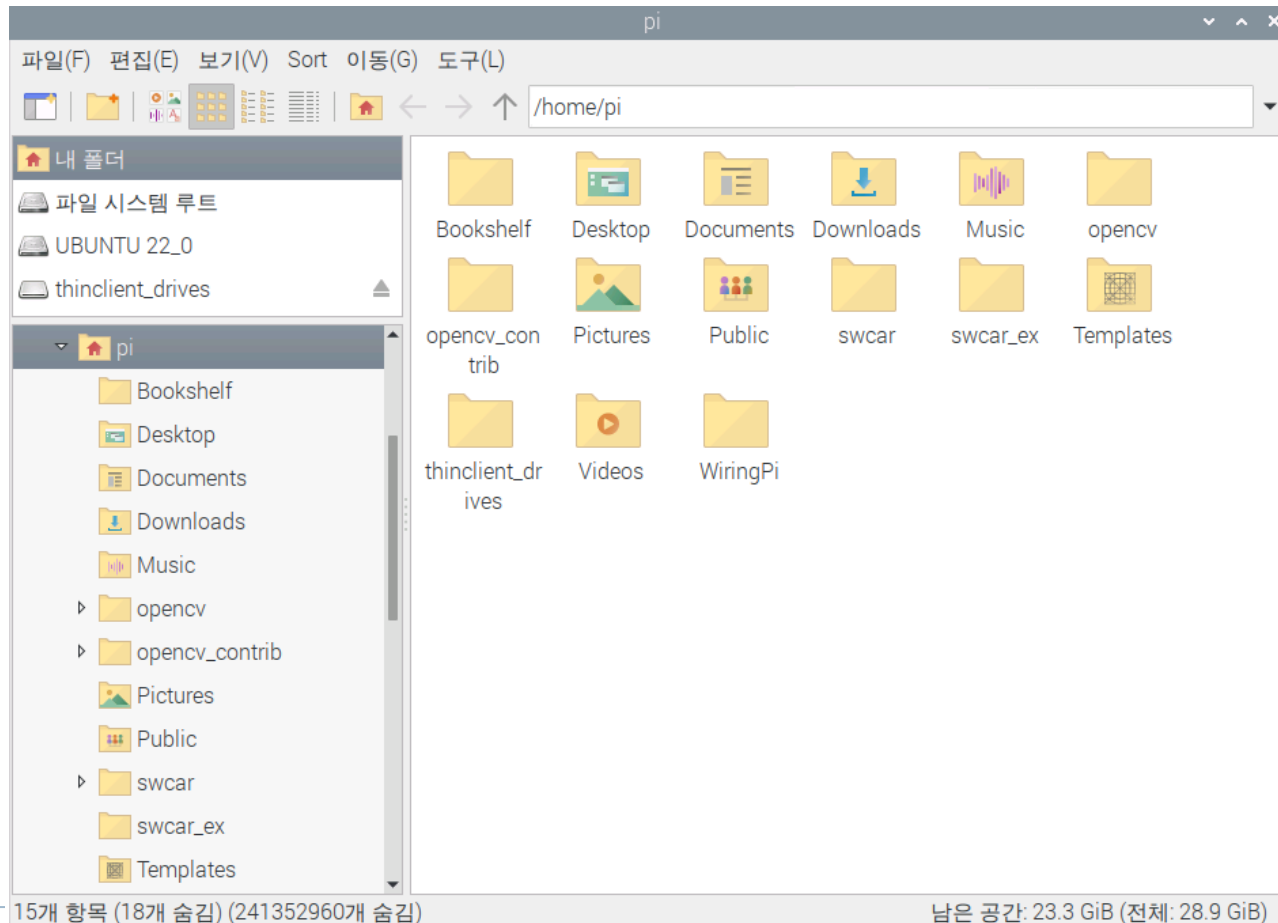
- 다음의 [파일 매니저] 버튼을 클릭



# Section 01 GPIO를 이용한 LED 제어하기

## □ 환경 설정

- 다음과 같이 [파일 매니저]가 실행





# Section 01 GPIO를 이용한 LED 제어하기

---

## □ 환경 설정

- 홈 폴더에서 스마트카를 위한 폴더는 아래와 같다.
  - Opencv : OpenCV 라이브러리를 위한 폴더
  - Swcar\_ex : swcar 파이썬 예제 폴더
  - WiringPI : WiringPI 라이브러리를 위한 폴더
  - Swcar : swcar Library 폴더

# Section 01 GPIO를 이용한 LED 제어하기

## □ 환경 설정

- Swcar 라이브러리의 함수들은 다음과 같다.

### ■ 함수 목록

- ▶ SWBOX\* SIO\_Init();
- ▶ char\* SIO\_WriteLED(int bTest);
- ▶ char\* SIO\_ReadSwitch();
- ▶ char\* SIO\_WriteMotor(int iSpeed);
- ▶ char\* SIO\_WriteServo(int iRange, int iAngle);
- ▶ char\* SIO\_ReadIR();
- ▶ char\* SIO\_ReadDistUS(int bFront);
- ▶ char\* SIO\_ReadDistLS();
- ▶ char\* SIO\_ReadZyro(int bTest);
- ▶ char\* SIO\_ReadGPS();
- ▶ int SIO\_ReadCOM();
- ▶ int SIO\_WriteCOM();

# Section 01 GPIO를 이용한 LED 제어하기

---

## □ 환경 설정

- SWBOX\*      `SIO_Init(int bUsePWM);`
  - Software I/O(SIO) 모듈을 초기화하며, 프로그램 시작시에 호출해야 한다.
  - Input parameter
    - bUsePWM: PWM 기능을 사용하려면 TRUE, 아니면 FALSE를 입력한다.
  - Return Value
    - SWBOX : SWBOX 구조체의 포인터를 리턴한다.
    - 이 포인터를 이용하여 내부 변수들에 직접 접근할 수 있다.

# Section 01 GPIO를 이용한 LED 제어하기

---

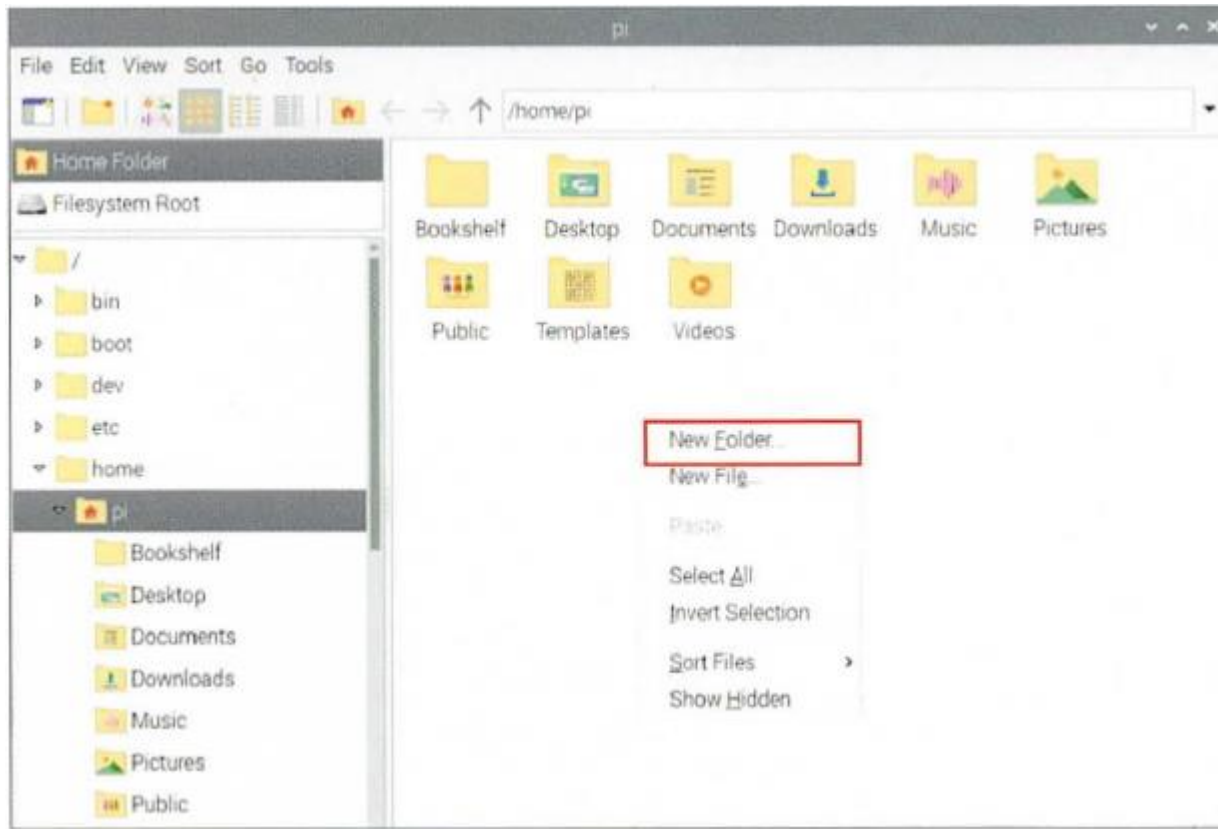
## □ 환경 설정

- int SIO\_WriteLED(int iLED);
  - **메인보드 LED를 켜거나 끈다.**
  - Input parameter
    - iLED : 4개의 LED로 각각 1,2,4,8의 값을 더한 값을 사용한다.
- Return Value
  - Int : iLED[0]부터 iLED[3]의 hexa 값

# Section 01 GPIO를 이용한 LED 제어하기

## □ 환경 설정

- 빈 공간에 마우스 오른쪽쪽을 클릭한 후 [New Folder ...]을 클릭한다.

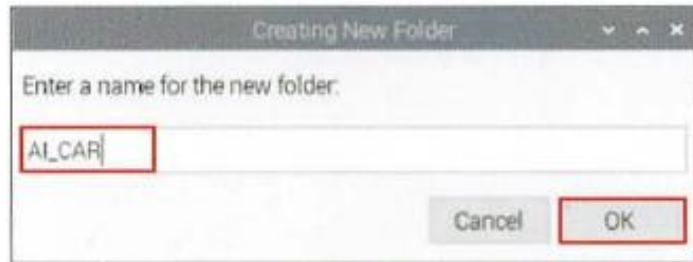


# Section 01 GPIO를 이용한 LED 제어하기

---

## □ 환경 설정

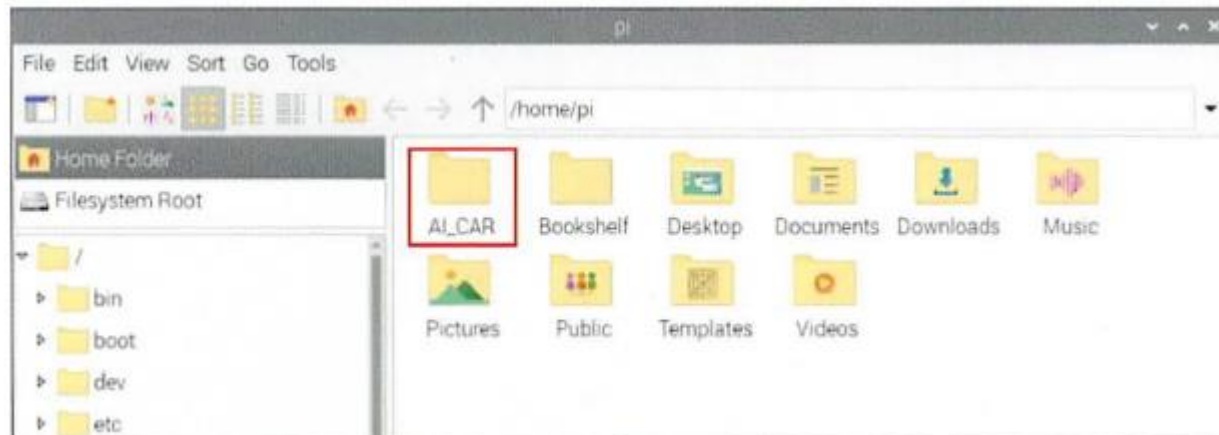
- 폴더이름을 “Smart\_Car”로 정한 후 [OK] 버튼을 누른다.



# Section 01 GPIO를 이용한 LED 제어하기

## □ 환경 설정

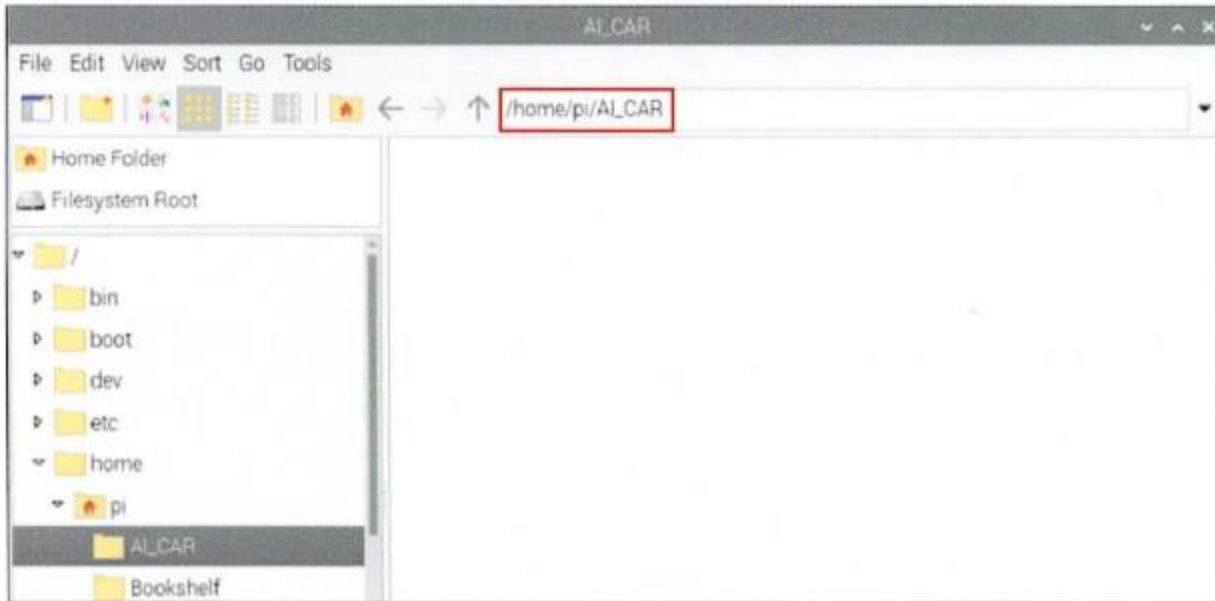
- “Smart\_Car”이름으로 폴더가 생성되었다.
- 윈도우에서와 같이 폴더를 더블 클릭하여 폴더로 들어간다.



# Section 01 GPIO를 이용한 LED 제어하기

## □ 환경 설정

- /home/pi/Smart\_Car 폴더로 이동하였다.

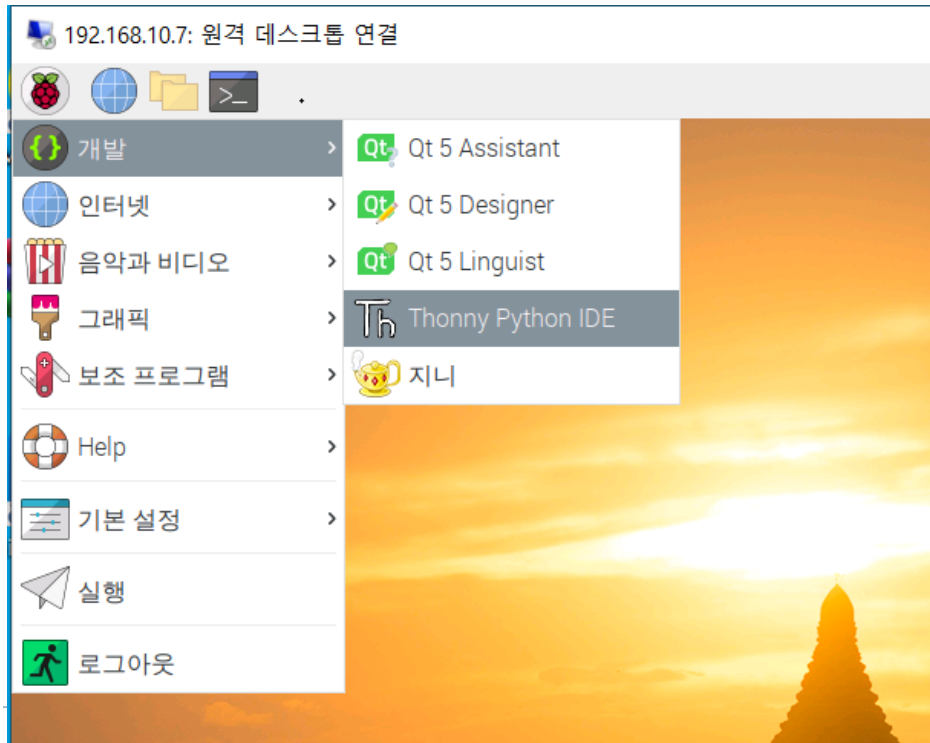




# Section 01 GPIO를 이용한 LED 제어하기

## □ 환경 설정

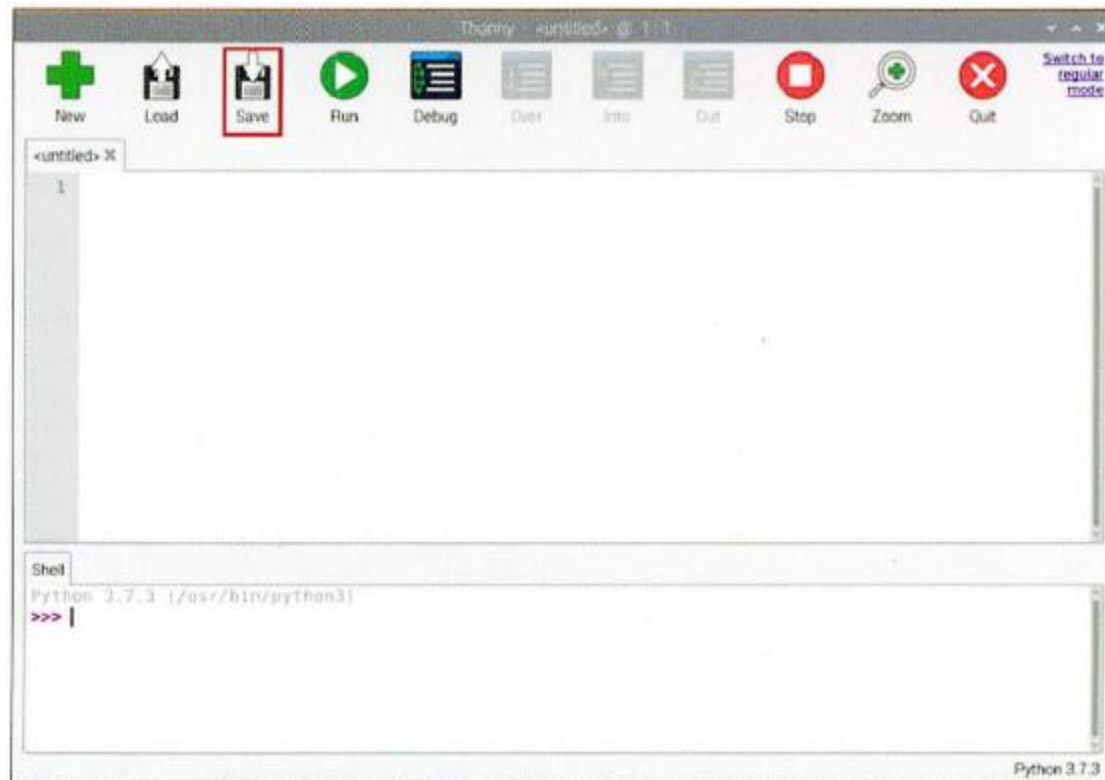
- 라즈베리파이에서 파이썬을 이용해서 프로그램하기 위해 Thonny Python IDE를 사용한다.
- Thonny Python IDE를 실행하기 위해 라즈베리파이 아이콘 -> Programming -> Thonny python IDE를 클릭한다.



# Section 01 GPIO를 이용한 LED 제어하기

## □ 환경 설정

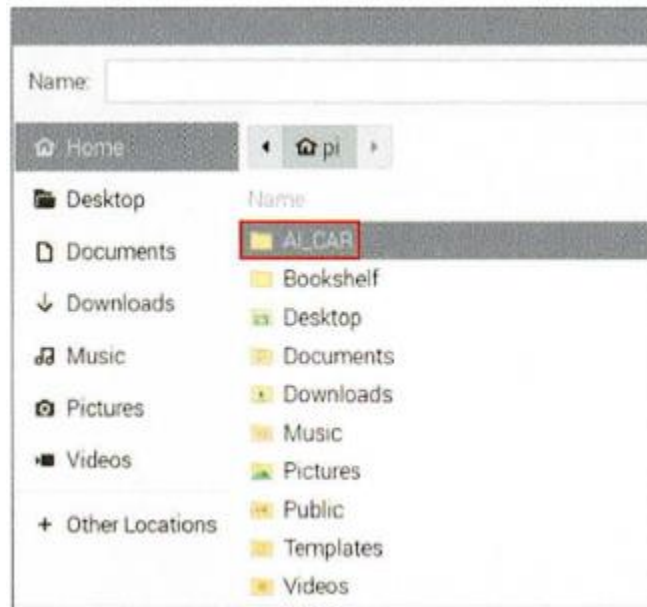
- 새로 생성되는 파이썬 파일을 저장해야 실행할 수 있다.
- [SAVE] 버튼을 클릭하여 파일을 저장한다.



# Section 01 GPIO를 이용한 LED 제어하기

## □ 환경 설정

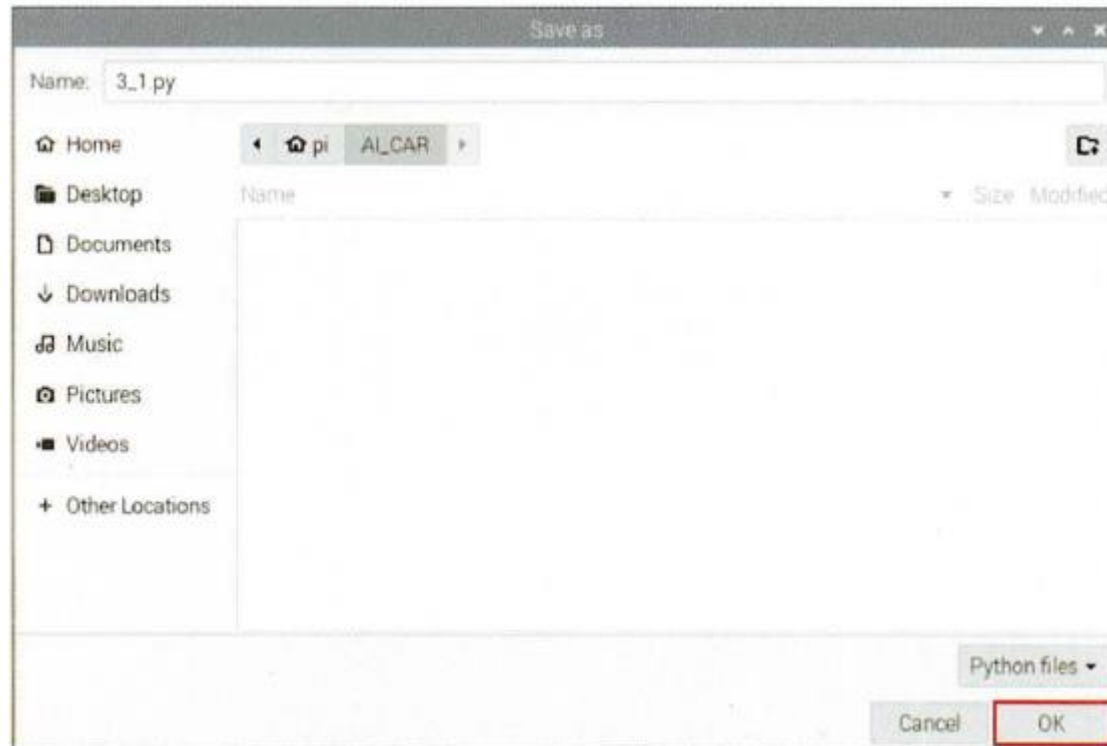
- 파일의 저장위치는 방금 생성한 /home/pi/Smart\_Car 폴더로 [Smart\_Car] 폴더를 더블 클릭하여 폴더에 접속한다.



# Section 01 GPIO를 이용한 LED 제어하기

## □ 환경 설정

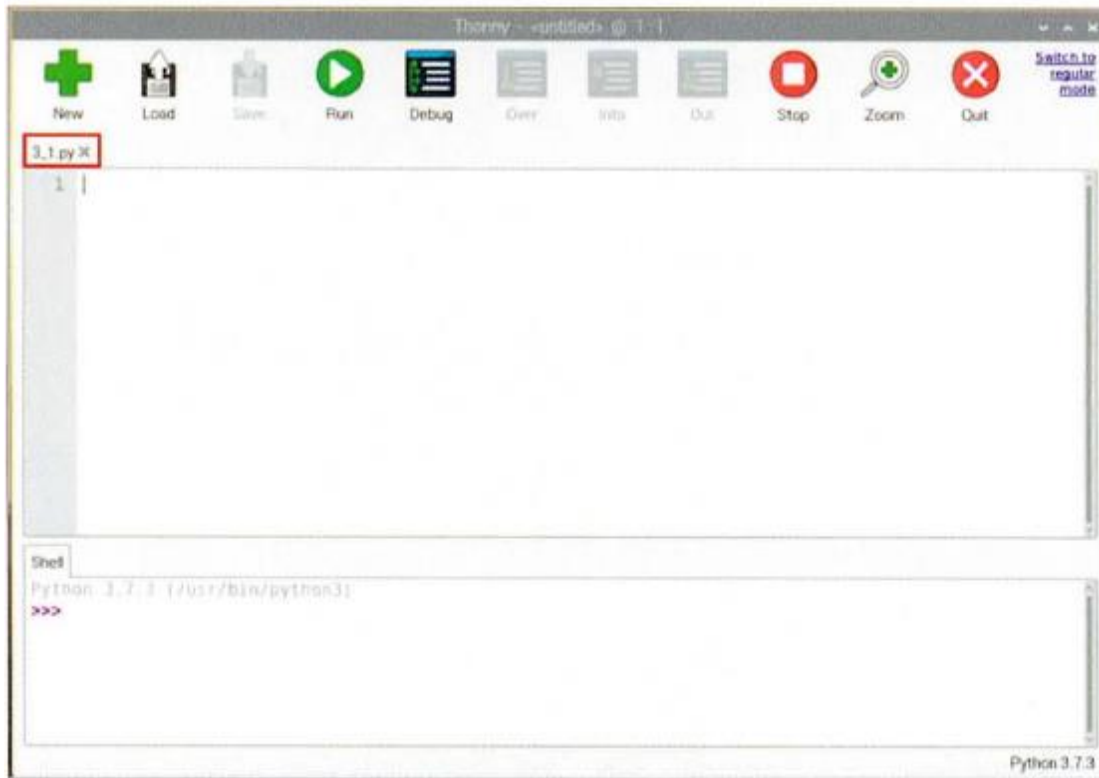
- 이름을 3\_1.py로 입력한 후 [OK] 버튼을 눌러 저장한다.



# Section 01 GPIO를 이용한 LED 제어하기

## □ 환경 설정

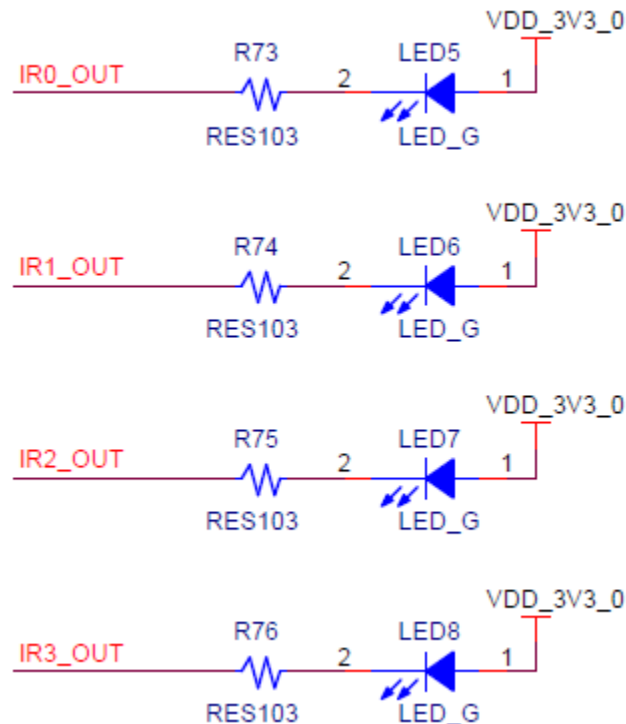
- 3\_1.py의 이름으로 저장되었다.



# Section 01 GPIO를 이용한 LED 제어하기

## □ 파이썬 코딩

- 파이썬 프로그램을 이용하여 자동차에 부착된 LED를 제어하는 코드를 만들어보자.
- 자동차에는 총 4개의 LED가 있다.
- 4 개의 LED는 LED5(GPIO16), LED6(GPIO17), LED7(GPIO18), LED8(GPIO19)으로 되어 있다.



# Section 01 GPIO를 이용한 LED 제어하기

## □ 파이썬 코딩

- 다음과 같은 코드를 Thonny IDE의 코드영역에 작성한 다음 [Run] 아이콘을 클릭하여 코드를 실행한다.

```
1  from ctypes import *
2  import os
3  import time
4
5  WiringPi = CDLL("/home/pi/WiringPi/wiringPi/libwiringPi.so.2.70",
6  mode=RTLD_GLOBAL)
7
8  swcar = cdll.LoadLibrary('home/pi/swcar/libswcar.so')
9
10 swcar.SIO_Init(0)
11
12 print('press ctrl + c to terminate program')
```

# Section 01 GPIO를 이용한 LED 제어하기

## □ 파이썬 코딩

- 다음과 같은 코드를 Thonny IDE의 코드영역에 작성한 다음 [Run] 아이콘을 클릭하여 코드를 실행한다.

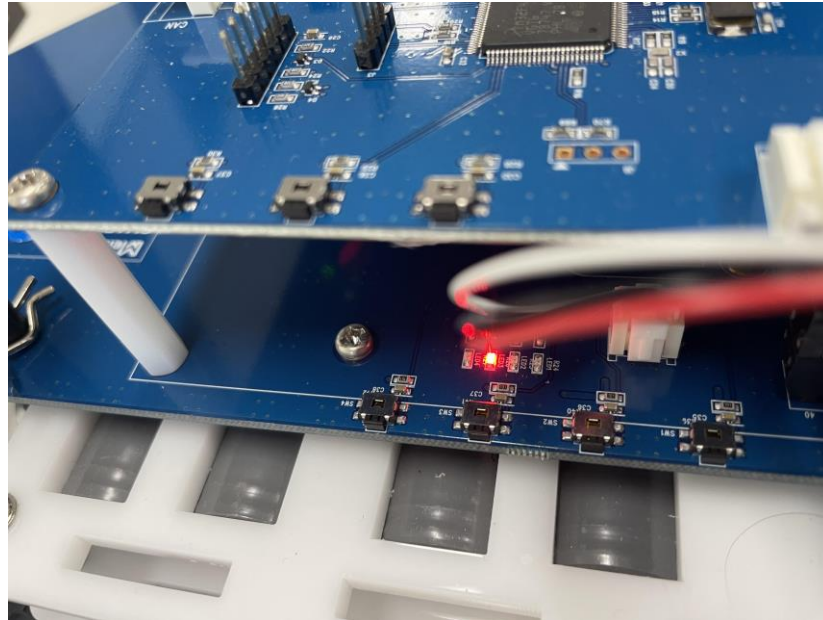
```
13 while(true):
14     swcar.SIO_WriteLED(1)
15     time.sleep(1)
16     swcar.SIO_WriteLED(2)
17     time.sleep(1)
18     swcar.SIO_WriteLED(4)
19     time.sleep(1)
20     swcar.SIO_WriteLED(8)
21     time.sleep(1)
```



# Section 01 GPIO를 이용한 LED 제어하기

## □ 파이썬 코딩

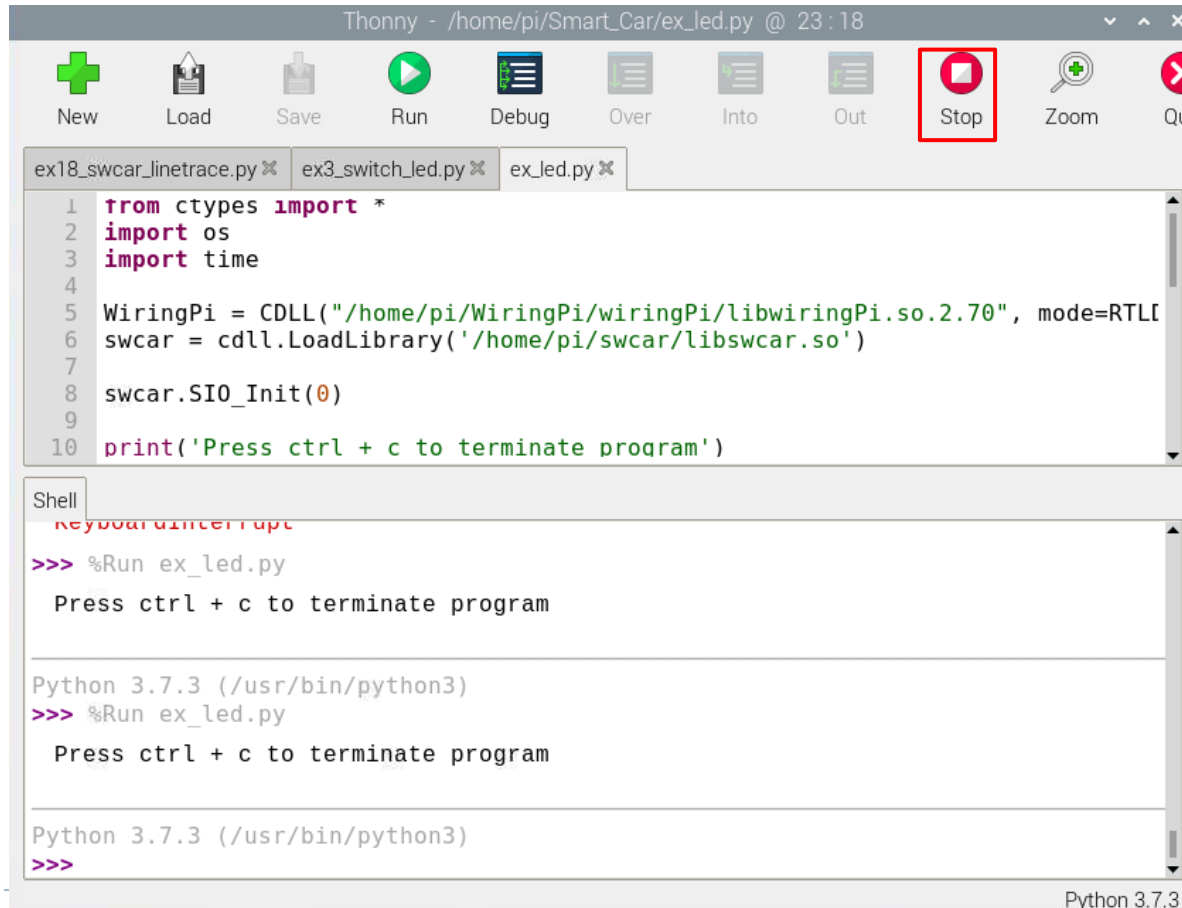
- 코드를 실행하면 자동차 키트 오른쪽 측면에 LED가 1초간격으로 순차적으로 on/off 되는 것을 확인할 수 있다.



# Section 01 GPIO를 이용한 LED 제어하기

## □ 파이썬 코딩

- 코드를 종료하고 싶다면 Stop 버튼을 눌러 정지한다.



The screenshot shows the Thonny IDE window titled "Thonny - /home/pi/Smart\_Car/ex\_led.py @ 23:18". The top toolbar contains icons for New, Load, Save, Run, Debug, Over, Into, Out, Stop, Zoom, and Quit. The "Stop" button, represented by a red square with a white circle and a diagonal line, is highlighted with a red rectangle. Below the toolbar, three tabs are open: "ex18\_swcar\_linetrace.py", "ex3\_switch\_led.py", and "ex\_led.py". The "ex\_led.py" tab is active, displaying the following Python code:

```
1 from ctypes import *
2 import os
3 import time
4
5 WiringPi = CDLL("/home/pi/WiringPi/wiringPi/libwiringPi.so.2.70", mode=RTLD
6 swcar = cdll.LoadLibrary('/home/pi/swcar/libswcar.so')
7
8 swcar.SIO_Init(0)
9
10 print('Press ctrl + c to terminate program')
```

Below the code editor is a "Shell" window. It shows the output of running the script twice. The first run shows a "KeyboardInterrupt" message, followed by the prompt ">>> %Run ex\_led.py" and the output "Press ctrl + c to terminate program". The second run shows the prompt "Python 3.7.3 (/usr/bin/python3)", followed by ">>> %Run ex\_led.py" and the output "Press ctrl + c to terminate program". The third run shows the prompt "Python 3.7.3 (/usr/bin/python3)" and ">>>". The bottom right corner of the window displays "Python 3.7.3".

# Section 01 GPIO를 이용한 LED 제어하기

## □ 파이썬 코딩

- Ctrl + C를 클릭하면 키보드 인터럽트가 발생했다는 메시지가 출력된다.

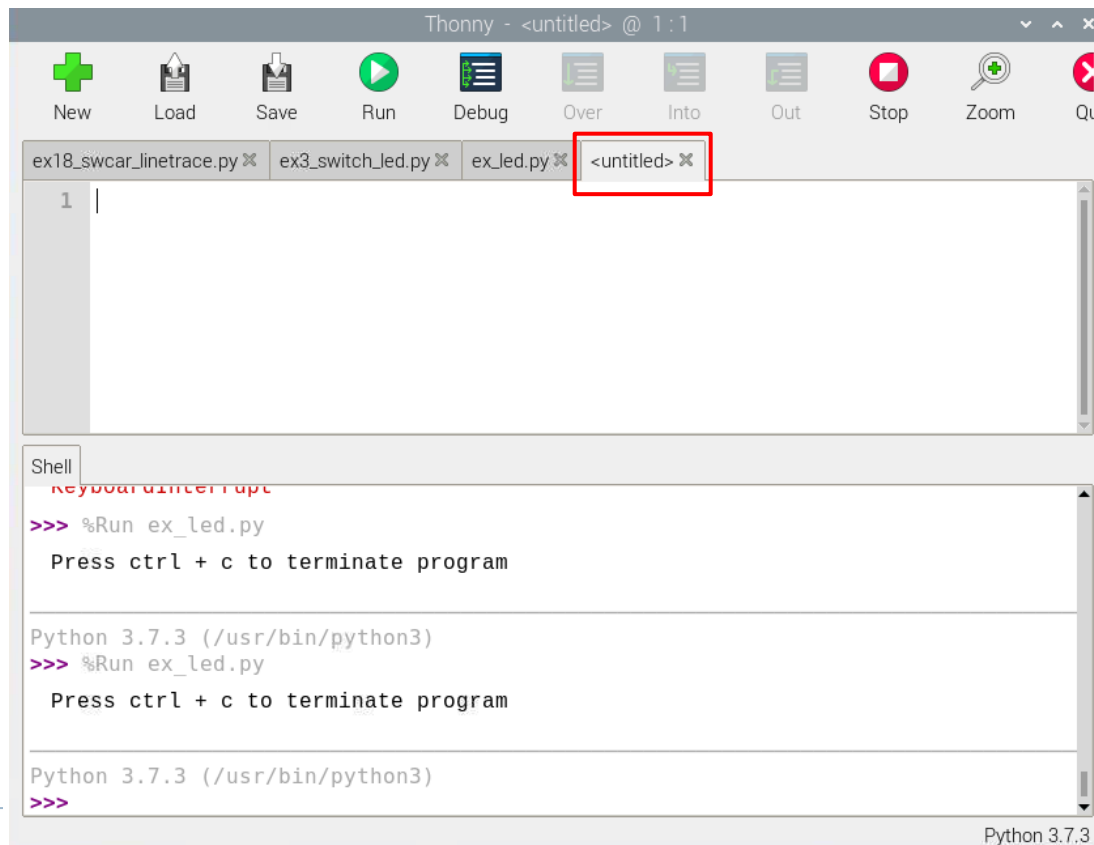
```
Shell
Press Ctrl + C to terminate program
ir = 0b0
ir = 0b0
Traceback (most recent call last):
  File "/home/pi/Smart_Car/ex_ir.py", line 15, in <module>
    time.sleep(1)
KeyboardInterrupt
```

- 이 메시지를 없애보자.
- 이를 위해 예외처리를 사용할 것이다.

# Section 01 GPIO를 이용한 LED 제어하기

## □ 파이썬 코딩

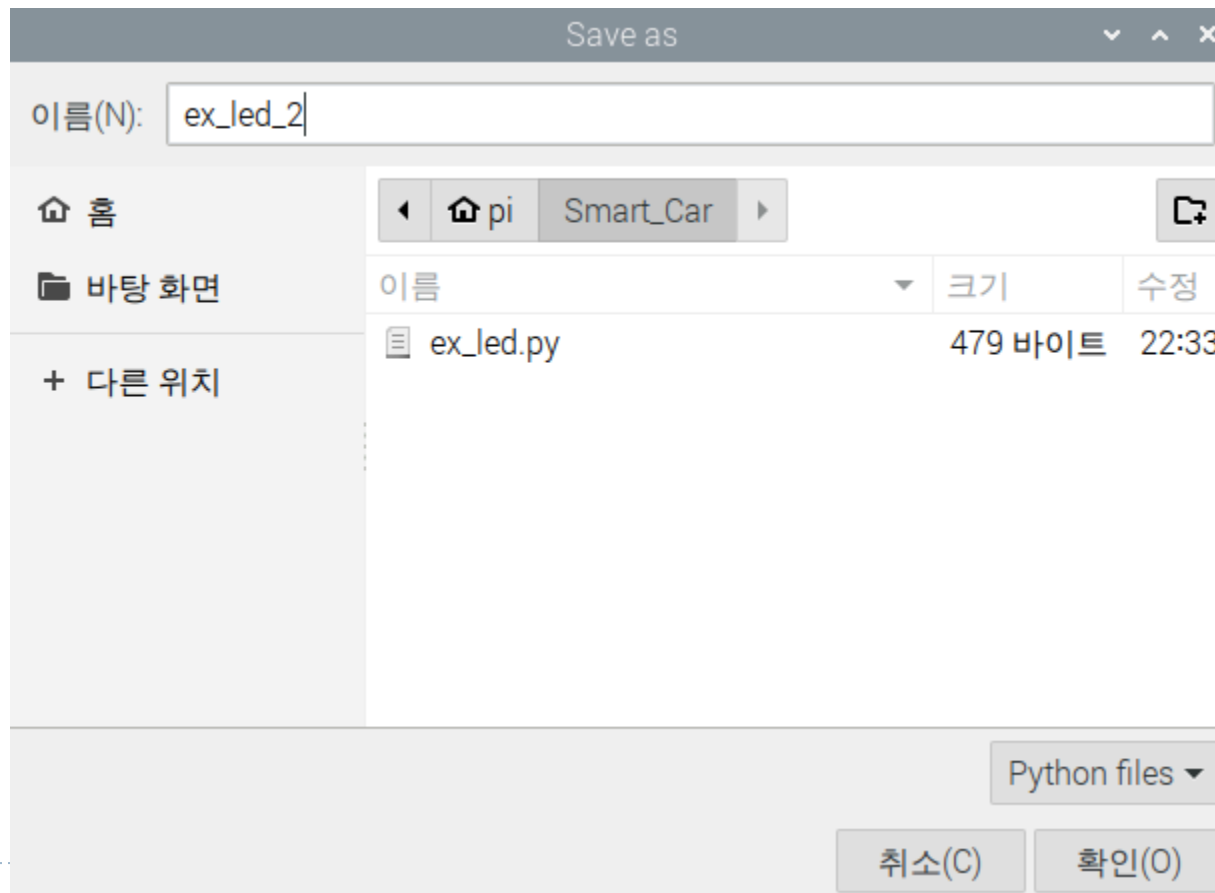
- New 버튼을 클릭하여 새로운 탭을 생성하였다.
- 새로운 탭이 생성되면 <untitled>로 되어 있다.



# Section 01 GPIO를 이용한 LED 제어하기

## □ 파이썬 코딩

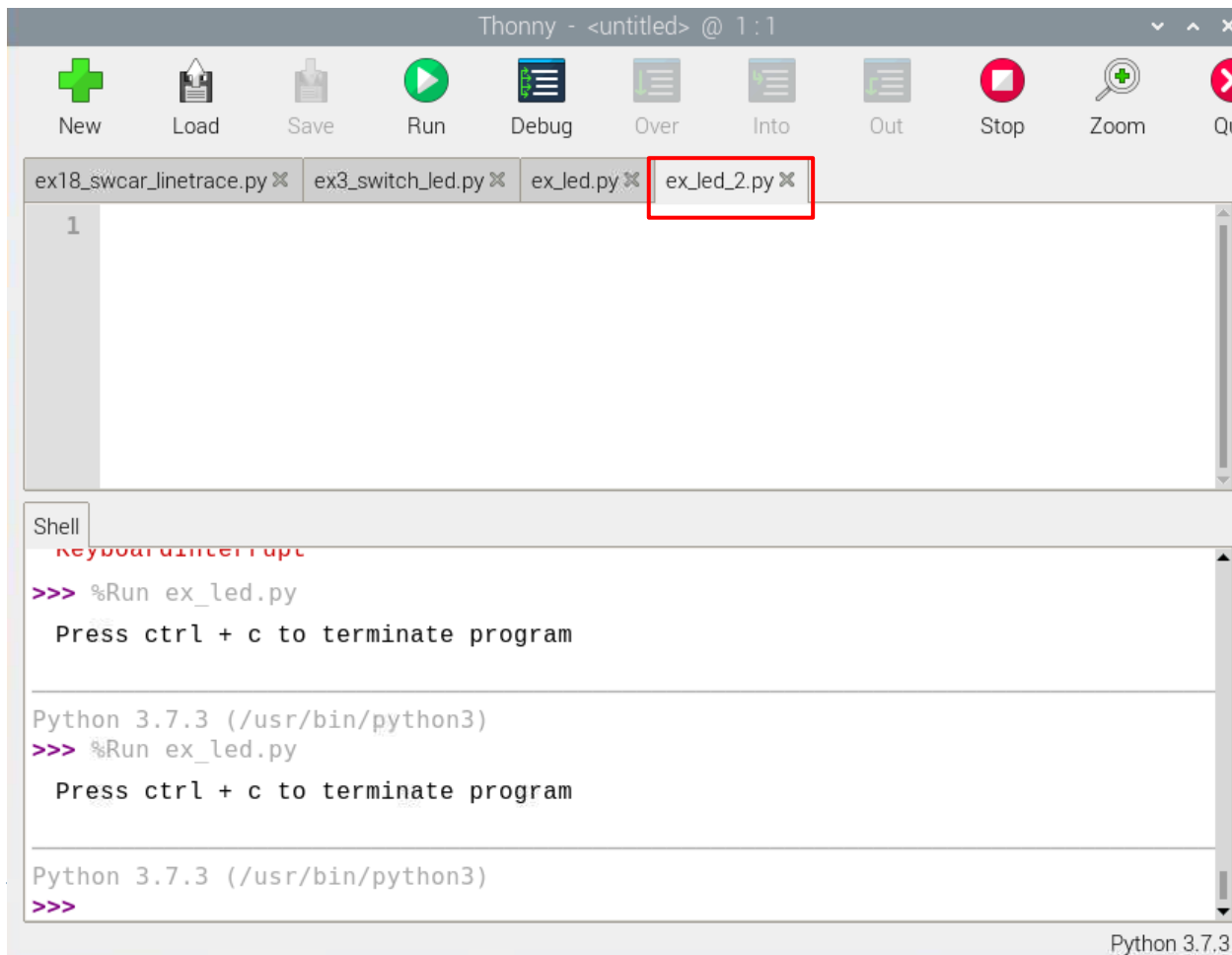
- Save 버튼을 클릭하여 /home/pi/Smart\_Car 폴더안에 [ex\_led\_2.py] 의 이름으로 저장한다.



# Section 01 GPIO를 이용한 LED 제어하기

## □ 파이썬 코딩

- [ex\_led\_2.py]로 저장되었다.



# Section 01 GPIO를 이용한 LED 제어하기

## □ 파이썬 코딩

- 다음과 같은 코드를 Thonny IDE의 코드영역에 작성한 다음 실행한다.

```
1  from ctypes import *
2  import os
3  import time
4
5  WiringPi = CDLL("/home/pi/WiringPi/wiringPi/libwiringPi.so.2.70", mode=RTLD_GL
  OBAL)
6  swcar = cdll.LoadLibrary('home/pi/swcar/libswcar.so')
7
8  swcar.SIO_Init(0)
9
10 print('press ctrl + c to terminate program')
11
```

# Section 01 GPIO를 이용한 LED 제어하기

## □ 파이썬 코딩

- 다음과 같은 코드를 Thonny IDE의 코드영역에 작성한 다음 실행한다.

```
12 try:
13     while(true):
14         swcar.SIO_WriteLED(1)
15         time.sleep(1)
16         swcar.SIO_WriteLED(2)
17         time.sleep(1)
18         swcar.SIO_WriteLED(4)
19         time.sleep(1)
20         swcar.SIO_WriteLED(8)
21         time.sleep(1)
22
23 except KeyboardInterrupt:
24     pass
```



# Section 02 GPIO를 이용한 버튼 제어하기

---

## □ 라즈베리파이 GPIO 입력

- 임베디드 애플리케이션 인터페이스에 사용되는 모든 컨트롤러 또는 프로세서는 5가지 방식으로 임베디드 전자 장치와 상호 작용한다.
  - 1. 디지털 출력
  - 2. 디지털 입력
  - 3. 아날로그 입력
  - 4. 아날로그 출력
  - 5. 직렬 통신
- 우리는 이미 앞 절에서 Raspberry Pi의 디지털 출력을 사용하는 방법을 다루었다.
- RPi의 범용 입/출력(GPIO)은 3.3V와 호환된다.
- 이 절에서는 Raspberry Pi 에서 디지털 입력을 사용하는 방법에 대해 설명한다.

## Section 02 GPIO를 이용한 버튼 제어하기

---

### □ 라즈베리파이 GPIO 입력

- 그렇게 하기 위해 푸시 버튼을 RPi의 GPIO에 인터페이스하고 여기에서 디지털 입력을 감지한다.
- 또한 Python의 IDLE(통합 개발 및 학습 환경) 콘솔에서 이 디지털 입력을 모니터링한다.
- 다음을 위해 컨트롤러/프로세서에 디지털 또는 논리 입력(High or Low)이 필요하다.
  - 주변기기에서 데이터 입력
  - 인터페이스된 외부 장치와 상호 작용
  - 인간-컴퓨터 인터페이스 설계

## Section 02 GPIO를 이용한 버튼 제어하기

---

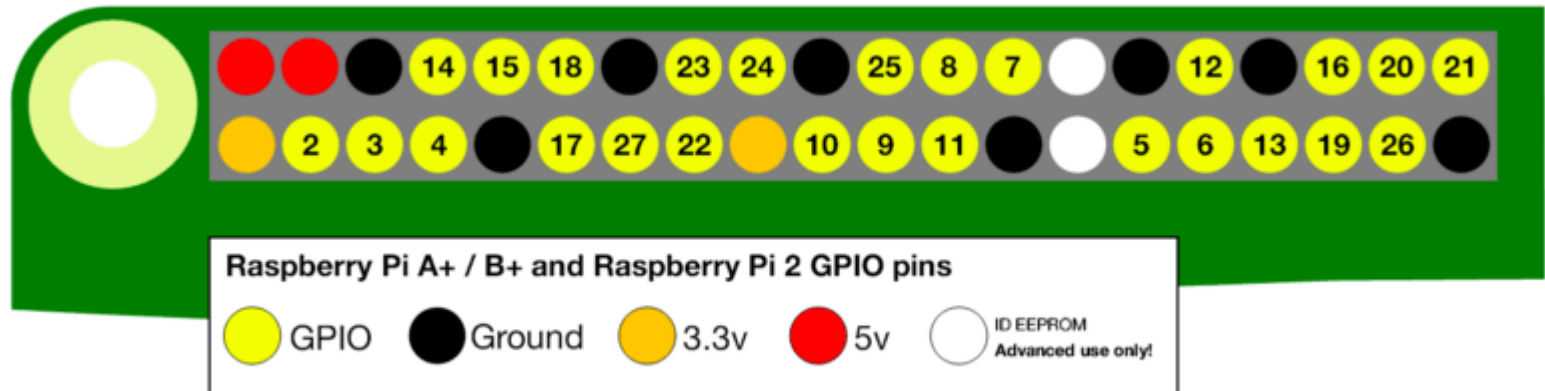
### □ 라즈베리파이 GPIO 입력

- 결국 전자적 관점에서 볼 때 모든 컴퓨터(범용 프로세서, 전용 프로세서 또는 컨트롤러)는 데이터를 입력하고 입력을 처리하고 데이터 또는 전자 신호를 출력할 수 있는 디지털 전자 회로이다.
- "입력-처리-출력"이 주어지면 컴퓨터는 지정된 작업이나 작업을 수행할 수 있다.
- 라즈베리파이는 단일 보드 컴퓨터로서 출력과 입력도 가능하다.
- GPIO 핀은 3.3V TTL 내전압성이므로 그에 따라 논리 신호를 출력 및 입력할 수 있다.
- RPi의 GPIO에 대한 입력 신호는 순간 유형 스위치를 통한 데이터 입력에 사용할 수 있다.

## Section 02 GPIO를 이용한 버튼 제어하기

### □ 라즈베리파이 GPIO 입력

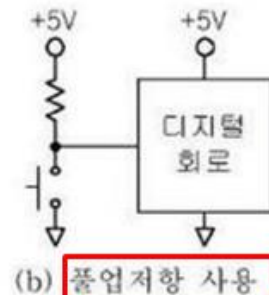
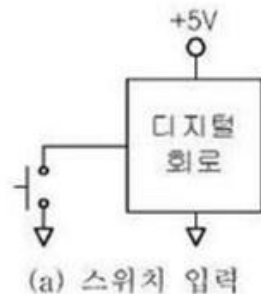
- 라즈베리파이 40핀 헤더에는 디지털 신호를 출력하고 디지털 입력을 받을 수 있는 26개의 핀이 있다.
- 이 디지털 입/출력 핀(GPIO)은 3.3V TTL 논리 신호와 호환된다.
- 고정 풀업 저항이 있는 GPIO2(보드 핀 번호 3) 및 GPIO3(보드 핀 번호 5)을 제외하고 내장된 풀업 또는 풀다운 저항을 사용하도록 핀을 구성할 수 있다.
- GPIO 핀은 논리적 HIGH의 경우 3.3V를 출력하고 논리적 LOW의 경우 0V를 출력한다.
- 그들은 3.3V를 논리적 HIGH로, 0V를 논리적 LOW로 읽는다.



## Section 02 GPIO를 이용한 버튼 제어하기

### □ Pull-up/Pull-down 저항

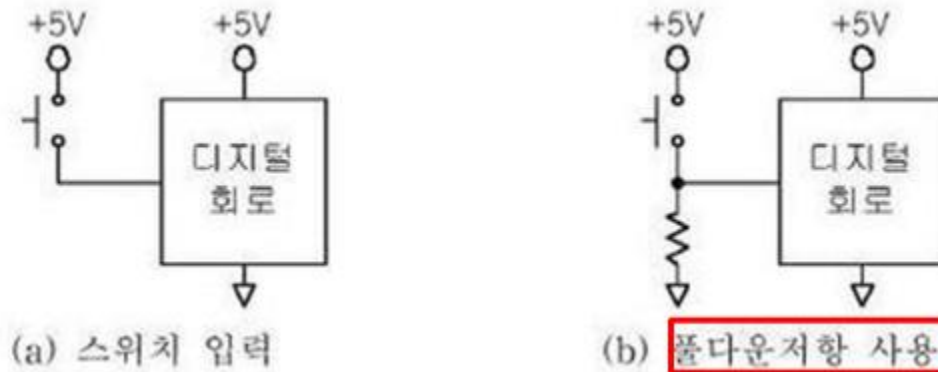
- GPIO 핀이 내장된 풀업 또는 풀다운 저항(사용자 프로그램에서)을 사용하도록 구성되지 않은 경우 외부에서 수행해야 하며 그렇지 않으면 부동 상태로 유지된다.
- GPIO는 매우 민감하며 약간의 변경에도 영향을 받을 수 있다.
- 예를 들어 사용자의 손가락, 공기 또는 브레드보드 연결에서 표유 정전 용량을 선택할 수 있다.
- 따라서 VCC(또는 RPi의 경우 3.3V) 또는 접지에 영구적으로 연결되어야 한다.



스위치	ON	OFF
(a)그림	0V(Low)	Floating
(b)그림	0V(Low)	+5V(High)

## Section 02 GPIO를 이용한 버튼 제어하기

### □ Pull-up/Pull-down 저항



스위치	ON	OFF
(a)그림	+5V(High)	Floating
(b)그림	+5V(High)	0V(Low)

- 위 회로가 의미하는 것은 디지털 입력 핀을 1과 0,으로 고정되게 만들어 준다는 의미이다.
- 1일 때는 확실히 1로 만들어 주기 위해 Pull-up 저항을, 0일 때는 확실히 0을 만들어 주기 위해 Pull-down 저항을 사용하는 이유다.

## Section 02 GPIO를 이용한 버튼 제어하기

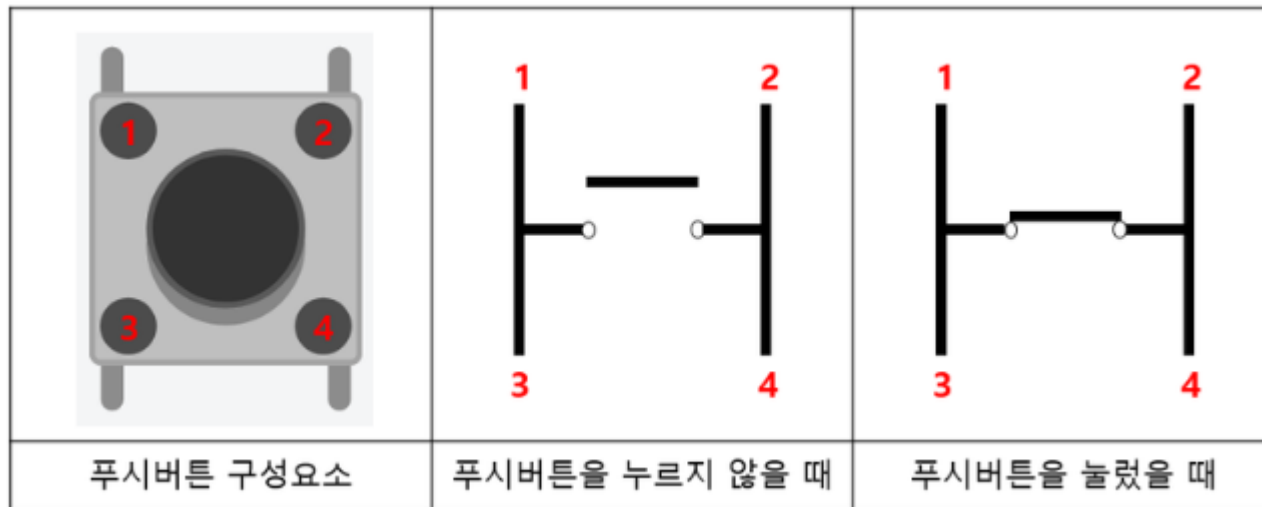
### □ Pull-up/Pull-down 저항

- LOW 신호를 디지털/논리 입력으로 감지하도록 핀을 구성해야 하는 경우 HIGH로 당겨야 한다.
- 그렇게 하려면 스위치의 한쪽 끝을 접지에 연결해야 하고 다른 쪽 끝은 풀업 저항을 통해 VCC(3.3V)에 유선으로 연결해야 한다.
- 이 구성에서 핀(기본값)은 HIGH 논리 신호를 수신한다.
- 버튼이나 스위치를 누르면 논리적 LOW가 수신된다.
- HIGH 신호를 디지털/논리 입력으로 감지하도록 핀을 구성해야 하는 경우 LOW로 당겨야 한다.
- 이렇게 하려면 스위치의 한쪽 끝을 VCC(3.3V)에 연결하고 다른 쪽 끝을 풀다운 저항을 통해 접지에 고정 배선해야 한다.
- 이 구성에서 핀(기본값)은 LOW 논리 신호를 수신한다.
- 버튼이나 스위치를 누르면 논리적 HIGH가 수신된다.
- 풀업 또는 풀다운 저항은 잠재적으로 핀을 손상시킬 수 있는 전압 서지 또는 표류 커패시턴스로부터 채널(GPIO 핀)을 보호한다.

## Section 02 GPIO를 이용한 버튼 제어하기

### □ 푸시 버튼

- 푸시 버튼은 가장 단순한 순간 유형의 스위치이다.
- 이들은 2단자 또는 4단자 스위치일 수 있다.
- 4단자 스위치는 양쪽 단자가 짧다.
- 이것은 두 개의 회로 분기가 동시에 전환될 수 있음을 의미한다.
- 그러나 2단자 푸시 버튼은 회로의 단일 분기만 전환할 수 있다.





## Section 02 GPIO를 이용한 버튼 제어하기

---

### □ 푸시 버튼 인터페이스

- 푸시 버튼 은 Raspberry Pi 의 GPIO 핀과 쉽게 인터페이스할 수 있다.
- GPIO2(보드 핀 번호 3) 및 GPIO3(보드 핀 번호 5)을 제외한 모든 GPIO 핀은 내부 풀업 또는 풀다운을 사용하도록 구성할 수 있다.
- 사용자 프로그램이 논리적 HIGH를 읽도록 설계된 경우 내부 또는 외부 풀다운이 발생해야 한다.
- 사용자 프로그램이 논리적 LOW를 읽도록 설계된 경우 외부 풀업이 수행되어야 한다.
- " 읽기 논리 레벨 " 은 사용자 프로그램에 따라 데이터 값 또는 제어 신호로 해석될 수 있다.

## Section 02 GPIO를 이용한 버튼 제어하기

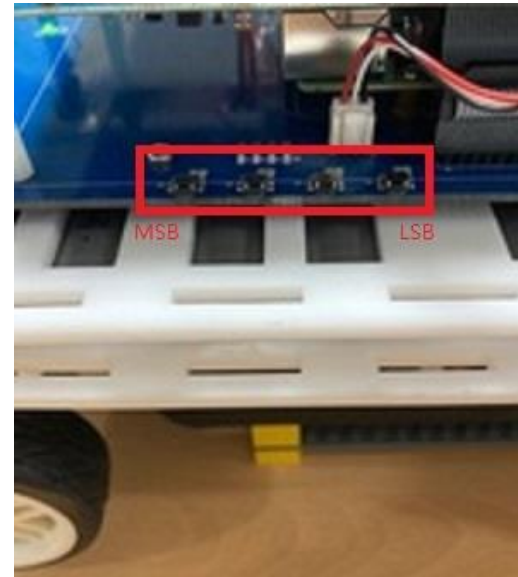
### □ 디지털 입력 감지

- 컨트롤러/프로세서가 디지털 입력을 감지하는 방법에는 여러 가지가 있다.
- 이러한 방법은 사용자 프로그램에서 구현된다.
- 논리적 입력을 감지하는 가장 기본적인 방법은 특정 시점의 입력값을 확인하는 것이다.
- 이것을 "투표"라고 한다.
- 그러나 이 방법에서 컨트롤러/프로세서는 사용자 프로그램이 잘못된 시간에 값을 읽는 경우 입력 읽기를 놓칠 수 있다.
- 폴링의 경우 스위치의 상태는 "조건부인 경우"에 의해 확인되고 루프에서 확인된다.
- 그러나 이것은 프로세서 집약적인 작업이다.
- 그렇게 하는 또 다른 방법은 단순히 인터럽트 또는 에지 감지를 사용하여 입력을 감지하는 것이다.
- 이 방법에서 사용자 프로그램은 GPIO 핀에서 HIGH에서 LOW로의 전환(하강 에지) 또는 LOW에서 HIGH로의 전환(상승 에지)을 기다린다.

## Section 02 GPIO를 이용한 버튼 제어하기

### □ 자동차 구조

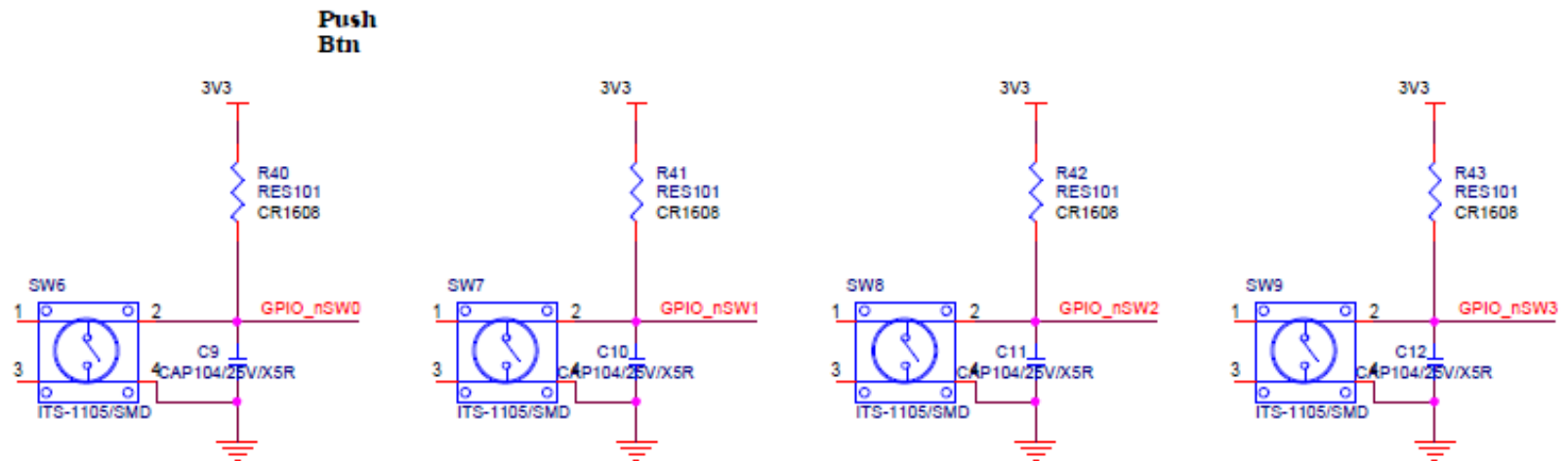
- 자동차에는 4개의 스위치가 있다.
  - SW6 = GPIO24
  - SW7 = GPIO25
  - SW8 = GPIO26
  - SW9 = GPIO27



## Section 02 GPIO를 이용한 버튼 제어하기

### □ 자동차 구조

- 스위치의 회로는 다음과 같다.



# Section 02 GPIO를 이용한 버튼 제어하기

## □ 버튼 제어 코드

- 다음과 같은 코드를 작성하여 스위치의 값을 입력받아 보자.

```
1  from ctypes import *
2  import os
3  import time
4
5  WiringPi = CDLL("/home/pi/WiringPi/wiringPi/libwiringPi.so.2.70", mode=RT
  LD_GLOBAL)
6  swcar = cdll.LoadLibrary('home/pi/swcar/libswcar.so')
7
8  swcar.SIO_Init(0)
9
10 print('press ctrl + c to terminate program')
11 print("push switch if you want.")
12
13 switch = 0
14 switch_pre = 0
15
```

## Section 02 GPIO를 이용한 버튼 제어하기

---

### □ 버튼 제어 코드

- 다음과 같은 코드를 작성하여 스위치의 값을 입력받아 보자.

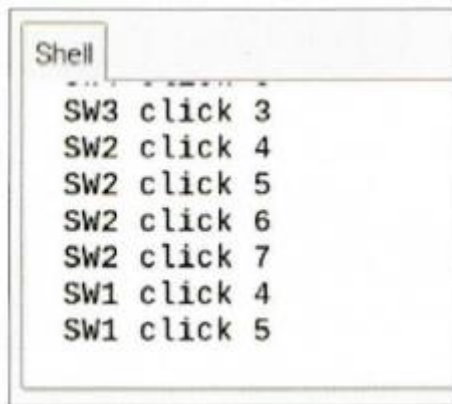
```
15
16 while True:
17     switch = swcar.SIO_ReadSwitch()
18     if (switch != switch_pre):
19         print('switch = ', bin(switch))
20         switch_pre = switch
```

## Section 02 GPIO를 이용한 버튼 제어하기

---

### □ 버튼 제어 코드

- Run을 클릭하여 코드를 실행한다.
- 스위치 4개가 모두 각각의 값을 가지며 동작한다.



```
Shell
...
SW3 click 3
SW2 click 4
SW2 click 5
SW2 click 6
SW2 click 7
SW1 click 4
SW1 click 5
```

---

# Q&A

