

# CH. 5. GPIO

# Section 01 GPIO의 특징

---

## □ 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 GPIO의 특징

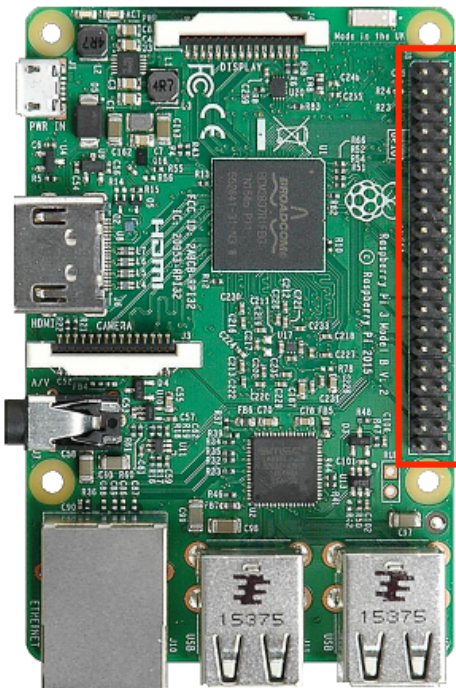
---

## □ 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 GPIO의 특징

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



라즈베리파이3 Model B

Raspberry Pi 3 GPIO Header

Pin#	NAME		NAME	Pin#
01	3.3v DC Power	⬛ ⬛	DC Power 5v	02
03	GPIO02 (SDA1 , I <sup>2</sup> C)	⬛ ⬛	DC Power 5v	04
05	GPIO03 (SCL1 , I <sup>2</sup> 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 <sup>2</sup> C ID EEPROM)	⬛ ⬛	(I <sup>2</sup> 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

# Section 01 GPIO의 특징

## □ 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

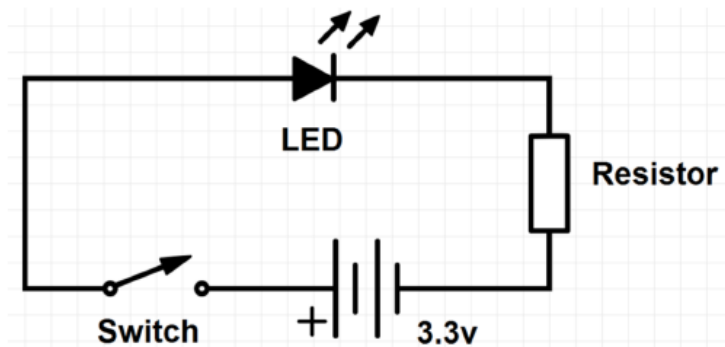
### WiringPi GPIO ←

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

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

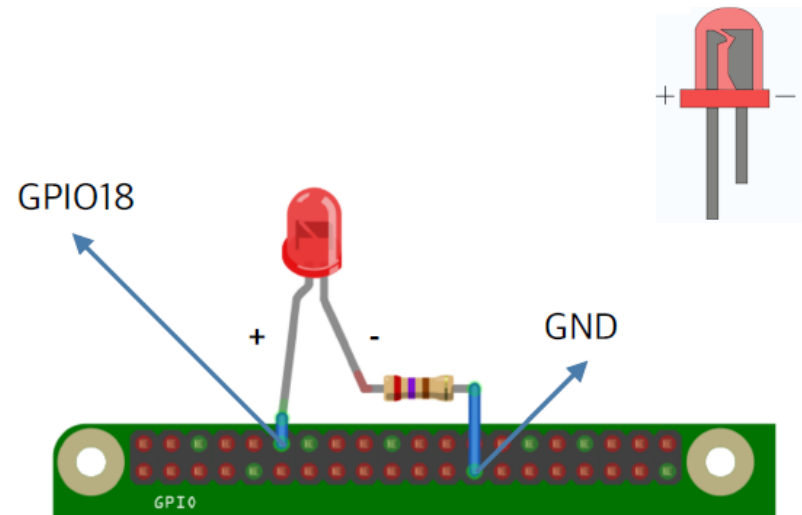
# Section 01 GPIO의 특징

## □ GPIO 작동 원리



LED를 켜고 끌 수 있는 가장 단순한 전기회로

=



라즈베리파이를 사용하여 만든 동일한 회로

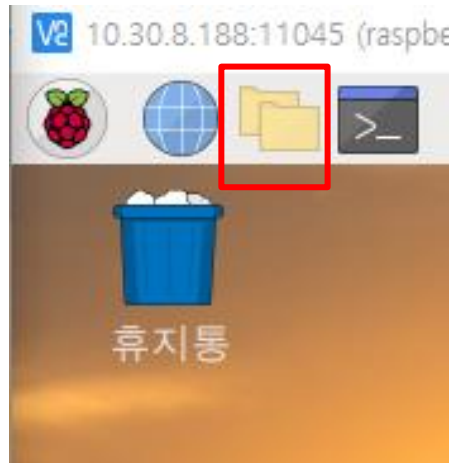
- GPIO가 스위치와 배터리의 기능을 모두 담당한다.
- 각각의 핀은 켜고 끌 수 있다. (김 : HIGH, 끄 : LOW)
- HIGH일때 출력은 3.3V, LOW일때 출력은 0V

## Section 02 LED 제어

---

### □ 환경 설정

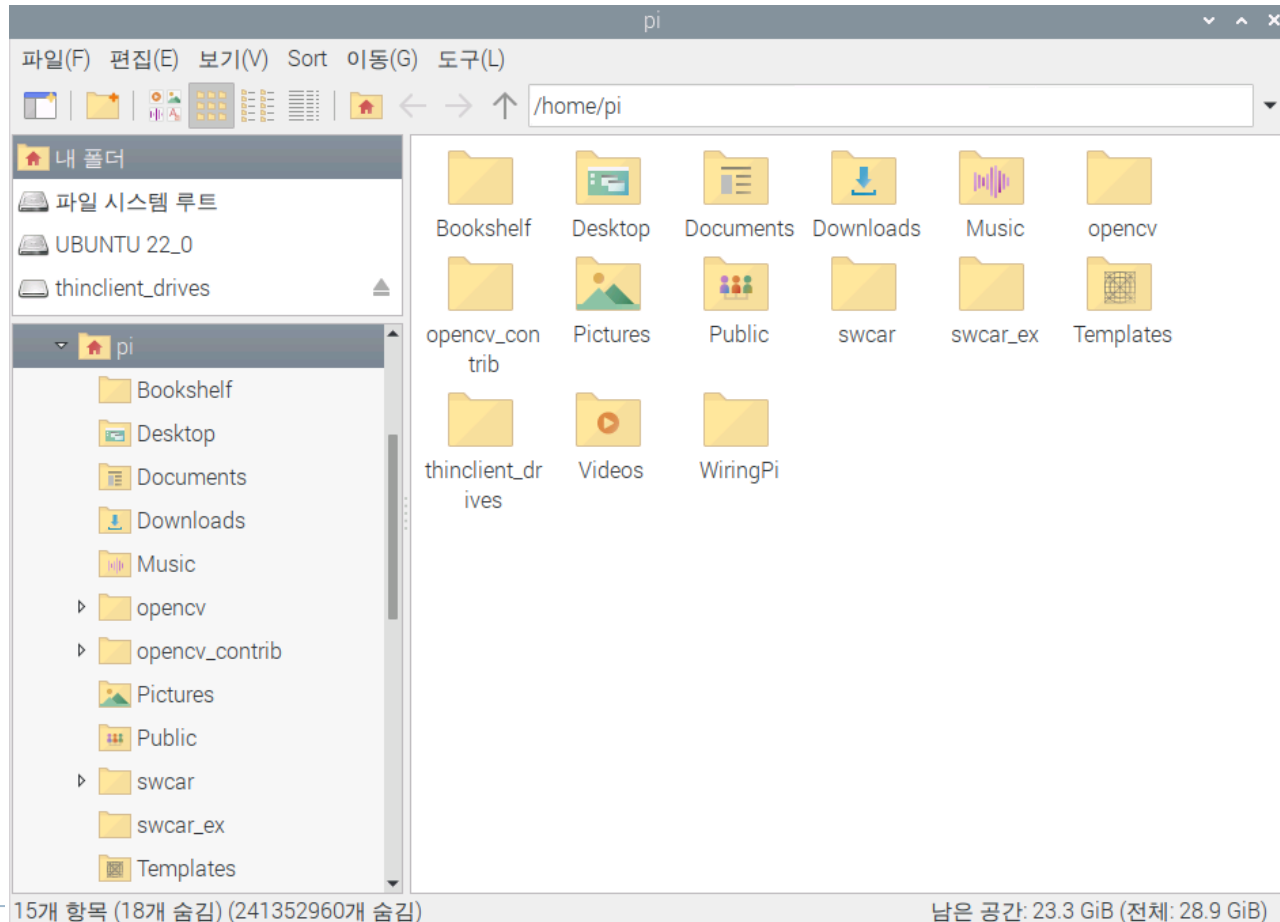
- 다음의 [파일 매니저] 버튼을 클릭



# Section 02 LED 제어

## □ 환경 설정

- 다음과 같이 [파일 매니저]가 실행





# Section 02 LED 제어

---

## □ 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 02 LED 제어

## □ 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 02 LED 제어

---

## □ 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 02 LED 제어

---

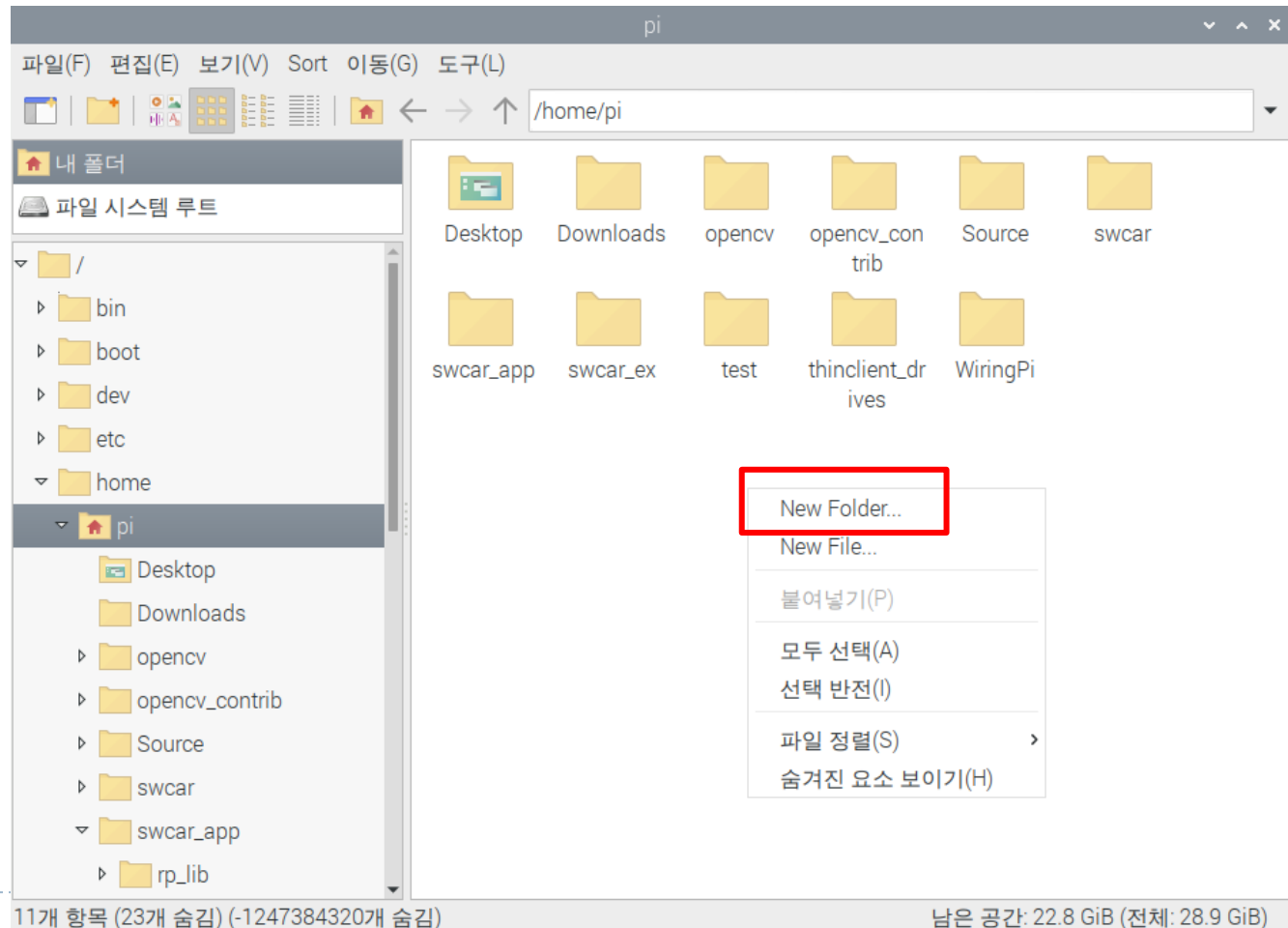
### □ 환경 설정

- 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 02 LED 제어

## □ 환경 설정

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

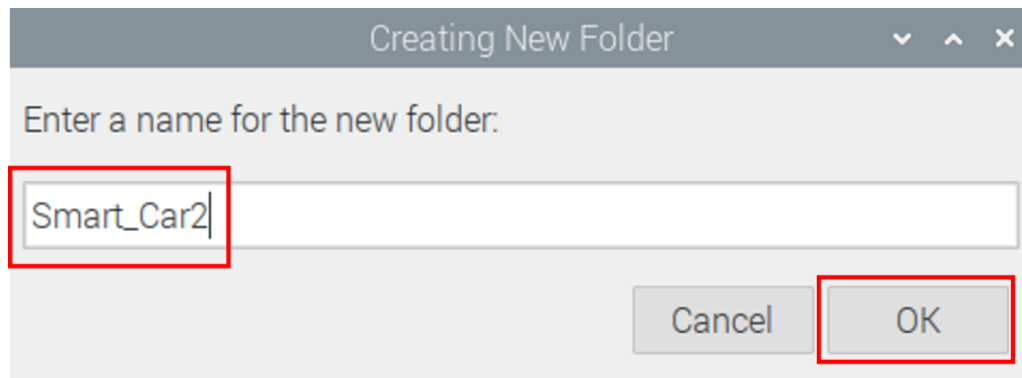


## Section 02 LED 제어

---

### □ 환경 설정

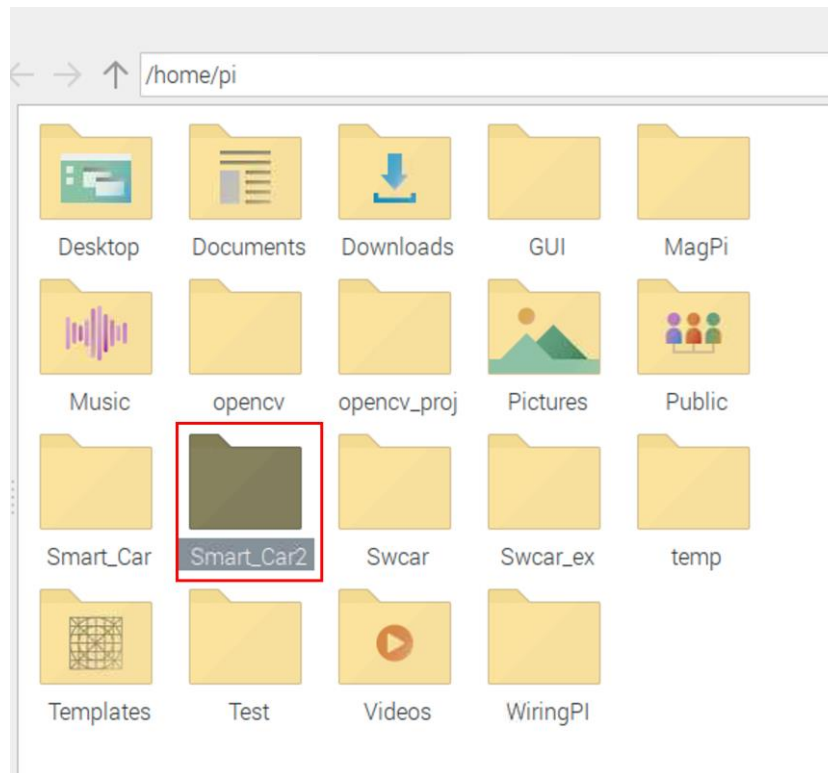
- 폴더이름을 “Smart\_Car2”로 정한 후 [OK] 버튼을 누른다.



## Section 02 LED 제어

### □ 환경 설정

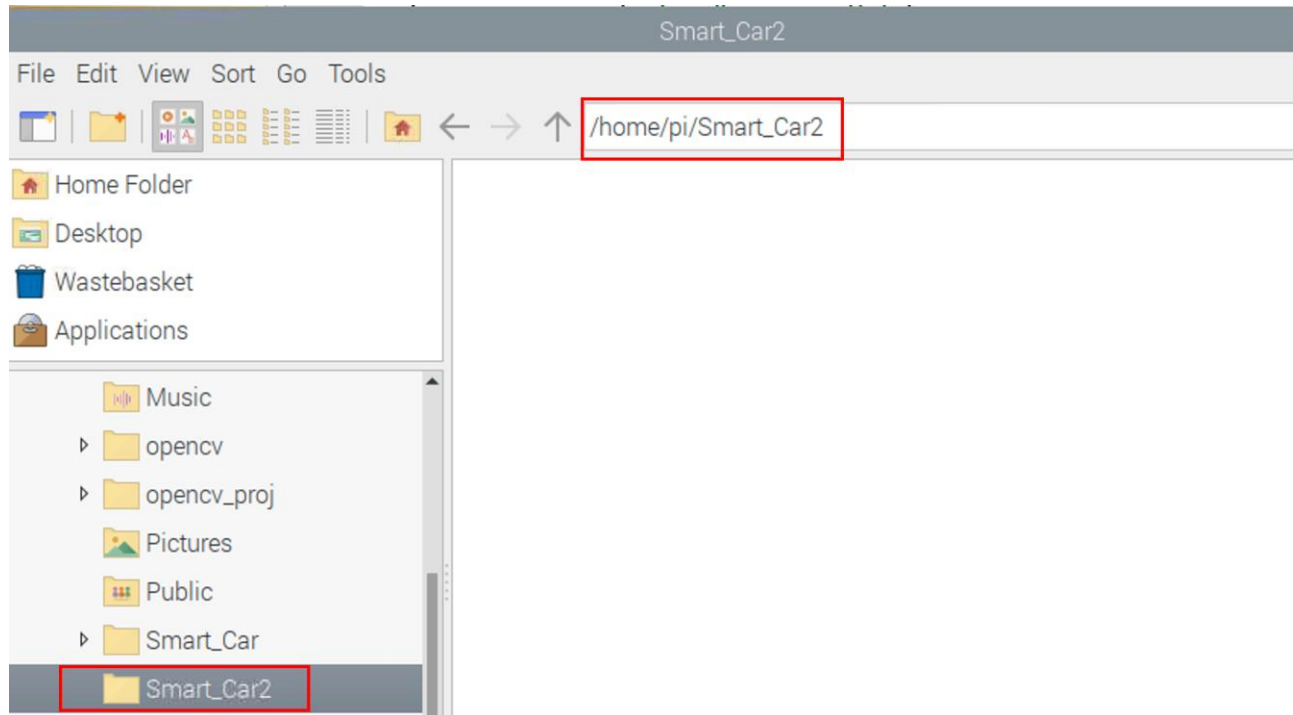
- “Smart\_Car2”이름으로 폴더가 생성되었다.
- 윈도우에서와 같이 폴더를 더블 클릭하여 폴더로 들어간다.



## Section 02 LED 제어

### □ 환경 설정

- /home/pi/Smart\_Car2 폴더로 이동하였다.

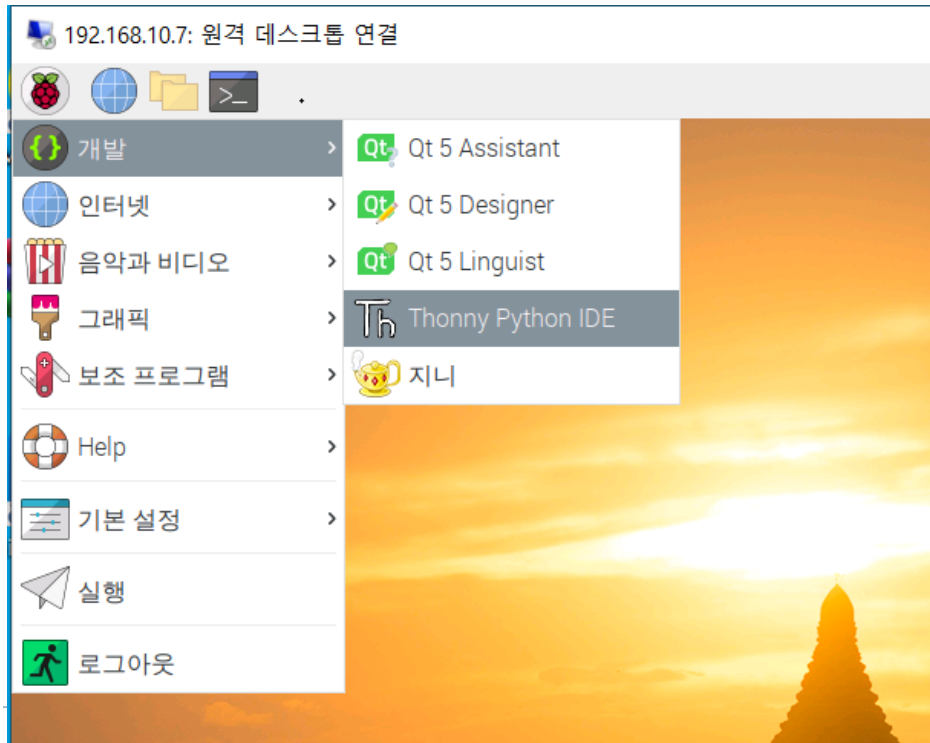




# Section 02 LED 제어

## □ 환경 설정

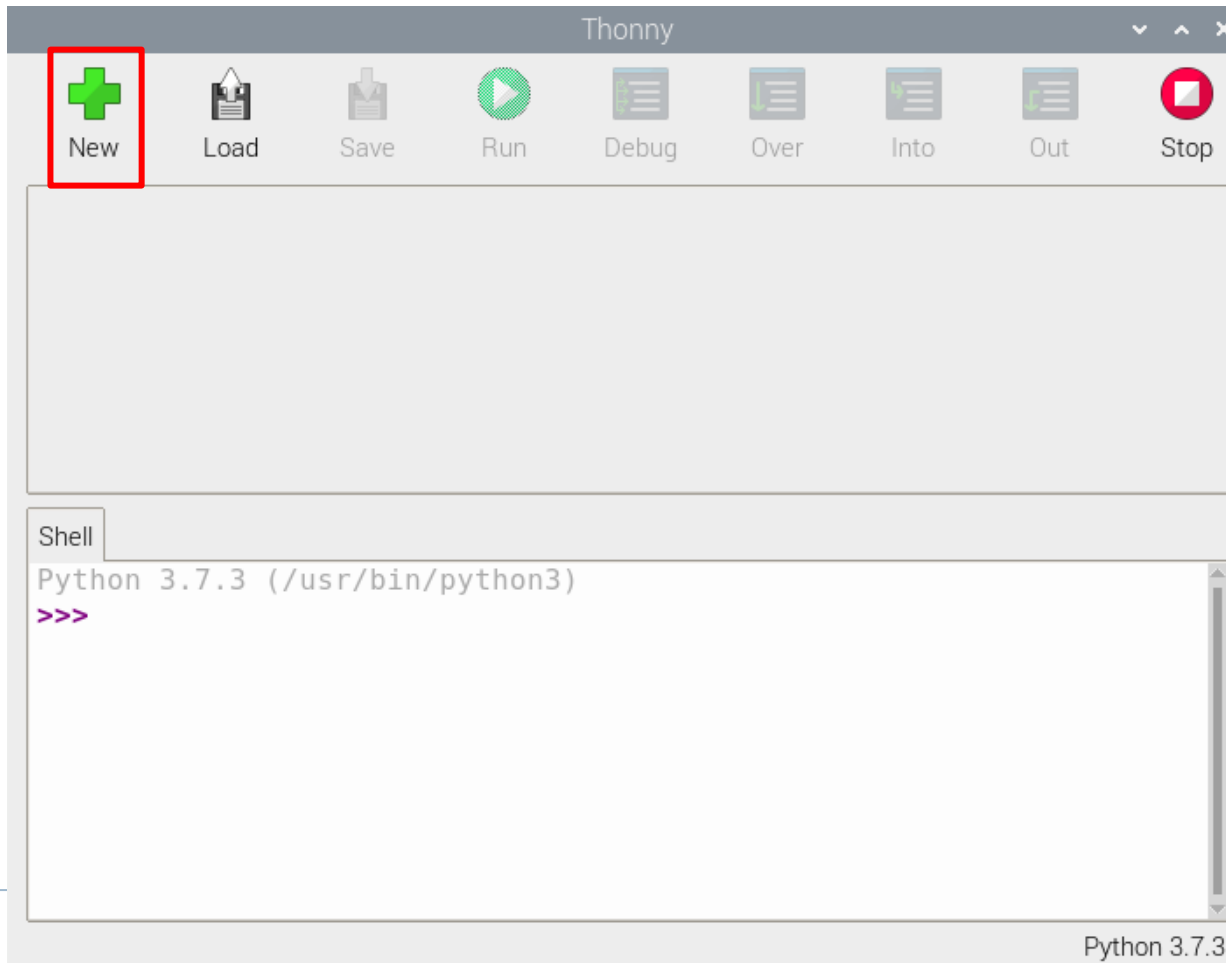
- 라즈베리파이에서 파이썬을 이용해서 프로그램하기 위해 Thonny Python IDE를 사용한다.
- Thonny Python IDE를 실행하기 위해 라즈베리파이 아이콘 -> Programming -> Thonny python IDE를 클릭한다.



# Section 02 LED 제어

## □ 환경 설정

- 다음과 같은 화면이 출력되면 [New] 아이콘을 클릭한 후 코드를 작성한다.



# Section 02 LED 제어

## □ 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 02 LED 제어

---

### □ 파이썬 코딩

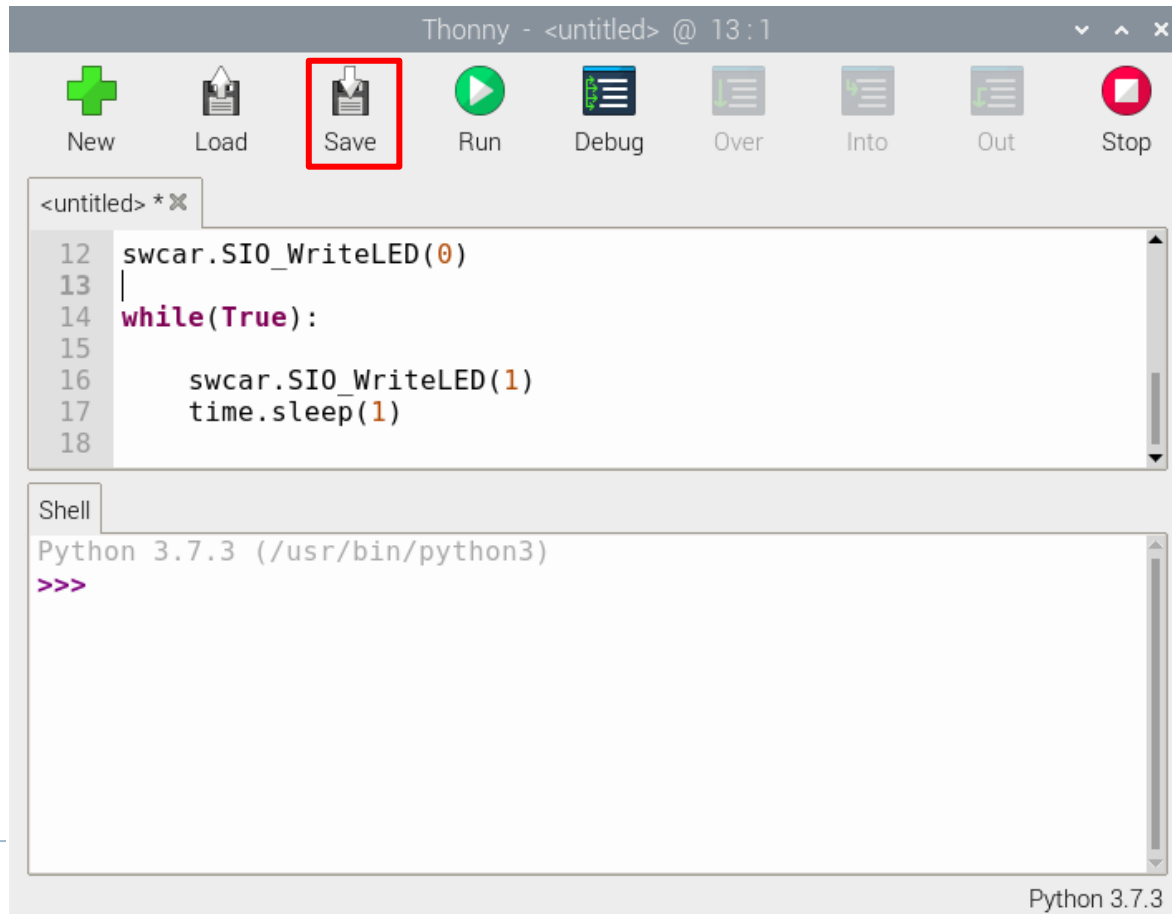
- 다음과 같은 코드를 Thonny IDE의 코드영역에 작성한 다음 [Run] 아이콘을 클릭하여 코드를 실행한다.

```
13 while(True):  
14     swcar.SIO_WriteLED(15)  
15     time.sleep(1)
```

## Section 02 LED 제어

### □ 환경 설정

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



## Section 02 LED 제어

---

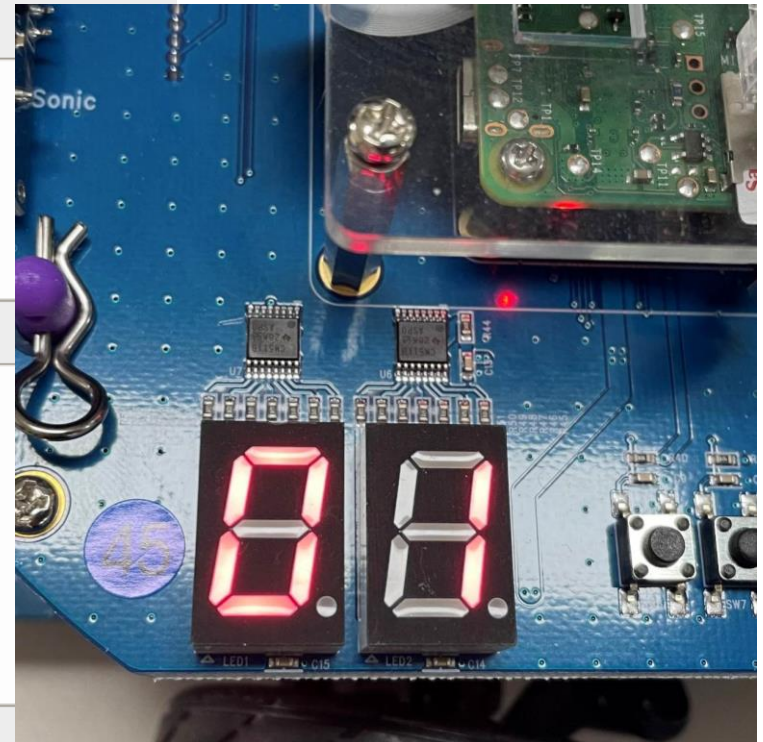
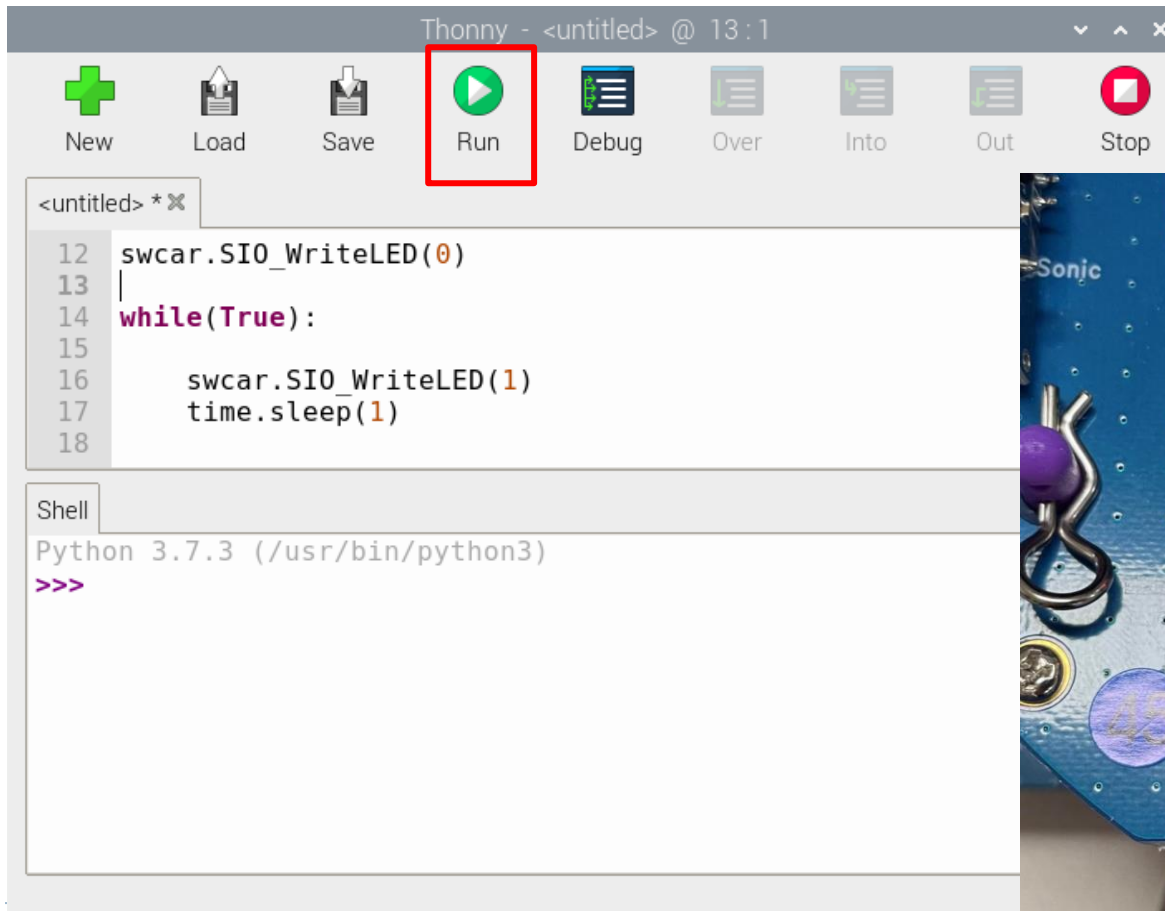
### □ 환경 설정

- 파일의 저장위치는 방금 생성한 /home/pi/Smart\_Car2 폴더로 [Smart\_Car2] 폴더를 더블 클릭하여 폴더에 접속한다.
- 이름을 ex5\_1.py로 입력한 후 [OK] 버튼을 눌러 저장한다.

# Section 02 LED 제어

## □ 환경 설정

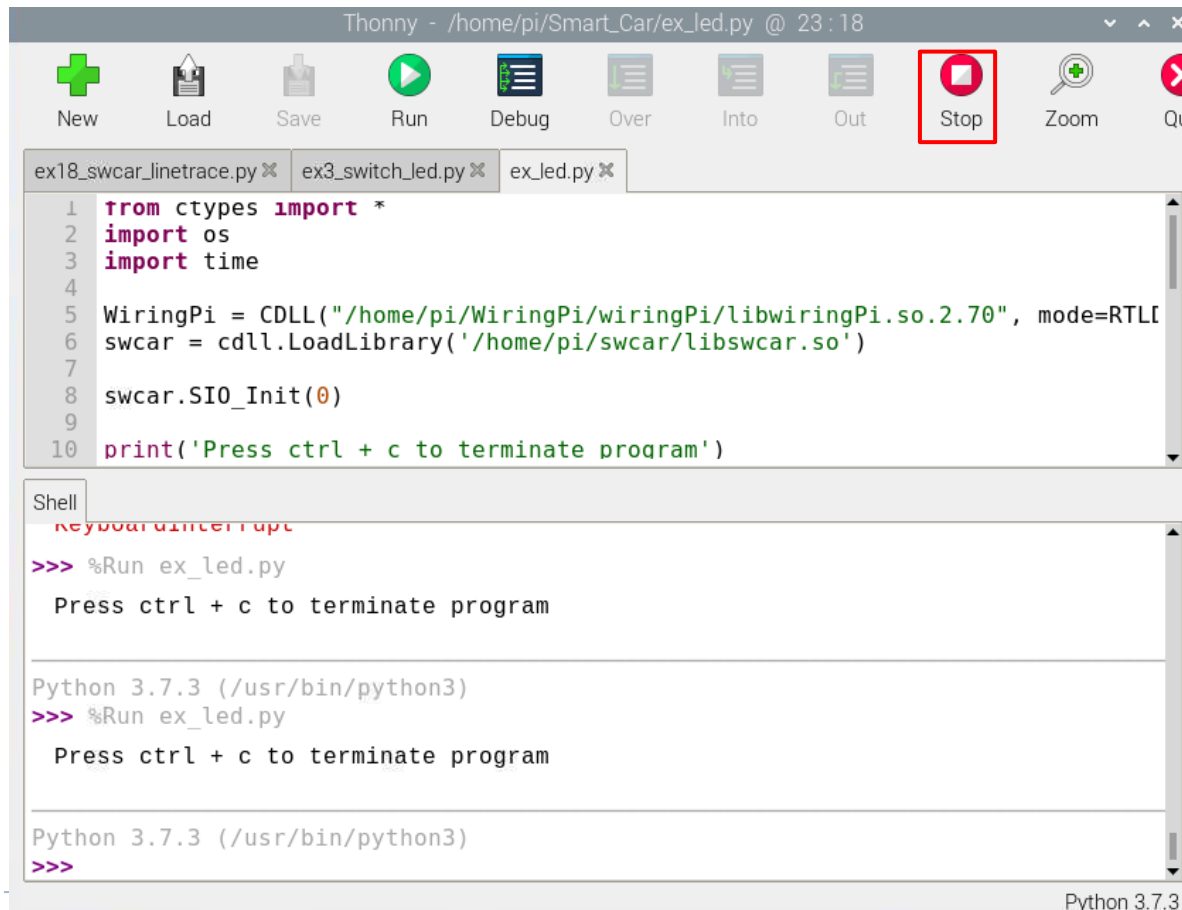
- [Run]을 클릭하여 실행한다.



# Section 02 LED 제어

## □ 파이썬 코딩

- 코드를 종료하고 싶다면 Stop 버튼을 눌러 정지한다.



Thonny - /home/pi/Smart\_Car/ex\_led.py @ 23:18

New Load Save Run Debug Over Into Out Stop Zoom Qu

ex18\_swcar\_linetrace.py ✕ ex3\_switch\_led.py ✕ ex\_led.py ✕

```
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')
```

Shell

```
KeyboardInterrupt
>>> %Run ex_led.py
Press ctrl + c to terminate program

Python 3.7.3 (/usr/bin/python3)
>>> %Run ex_led.py
Press ctrl + c to terminate program

Python 3.7.3 (/usr/bin/python3)
>>>
```

Python 3.7.3



# Section 02 LED 제어

## □ 파이썬 코딩

- Ctrl + C를 클릭하면 키보드 인터럽트가 발생했다는 메시지가 출력된다.

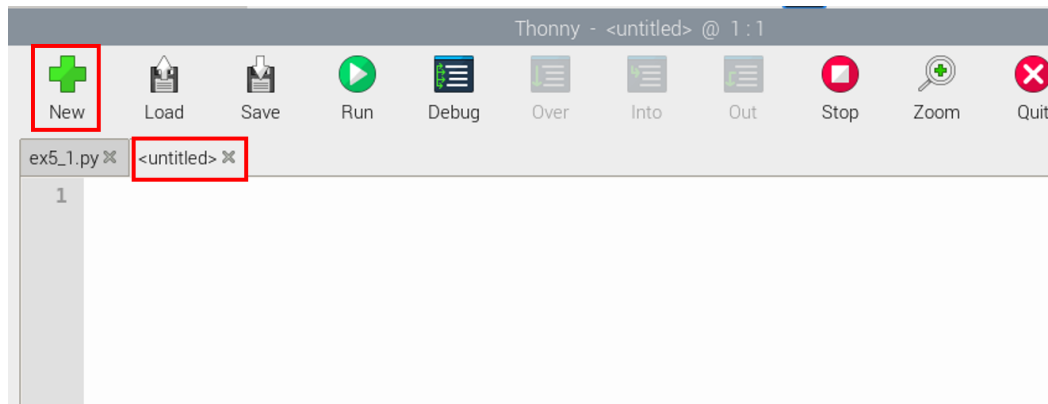
```
Shell
Press ctrl + c to terminate program
ir = 0b0
ir = 0b0
Traceback (most recent call last):
  File "/home/pi/Smart_Car/ex_ir.py", line 15, in <module>
    time.sleep(1)
KeyboardInterrupt
```

- 이 메시지를 없애보자.
- 이를 위해 예외처리를 사용할 것이다.

## Section 02 LED 제어

### □ 파이썬 코딩

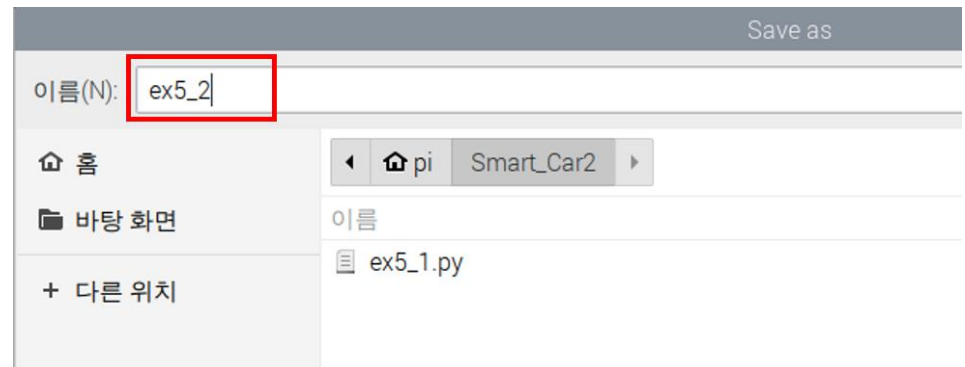
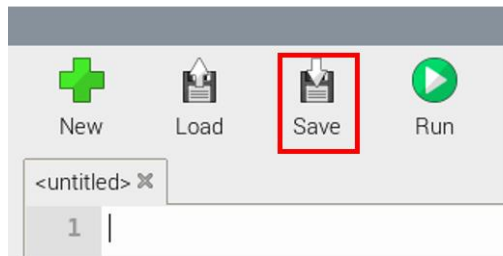
- New 버튼을 클릭하여 새로운 탭을 생성하였다.
- 새로운 탭이 생성되면 <untitled>로 되어 있다.



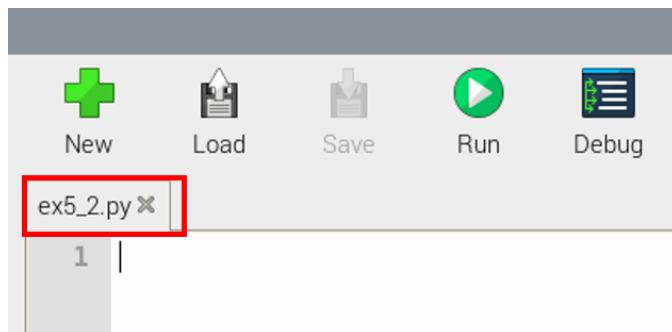
## Section 02 LED 제어

### □ 파이썬 코딩

- Save 버튼을 클릭하여 /home/pi/Smart\_Car2 폴더안에 [ex5\_2.py] 의 이름으로 저장한다.



- [ex5\_2 . py]로 저장되었다.



## Section 02 LED 제어

### □ 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_GLO
  BAL)
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
12 swcar.SIO_WriteLED(1)
13 i = 0
```

## Section 02 LED 제어

### □ 파이썬 코딩

- 다음과 같은 코드를 Thonny IDE의 코드영역에 작성한 다음 실행한다.

```
12 try:
13     while(true):
14         i += 1
15         if i > 99 :
16             i = 0
17         swcar.SIO_WriteLED(i)
21         time.sleep(1)
22
23 except KeyboardInterrupt:
24     pass
```

## Section 03 버튼 제어

---

### □ 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 03 버튼 제어

---

### □ 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 03 버튼 제어

---

### □ 라즈베리파이 GPIO 입력

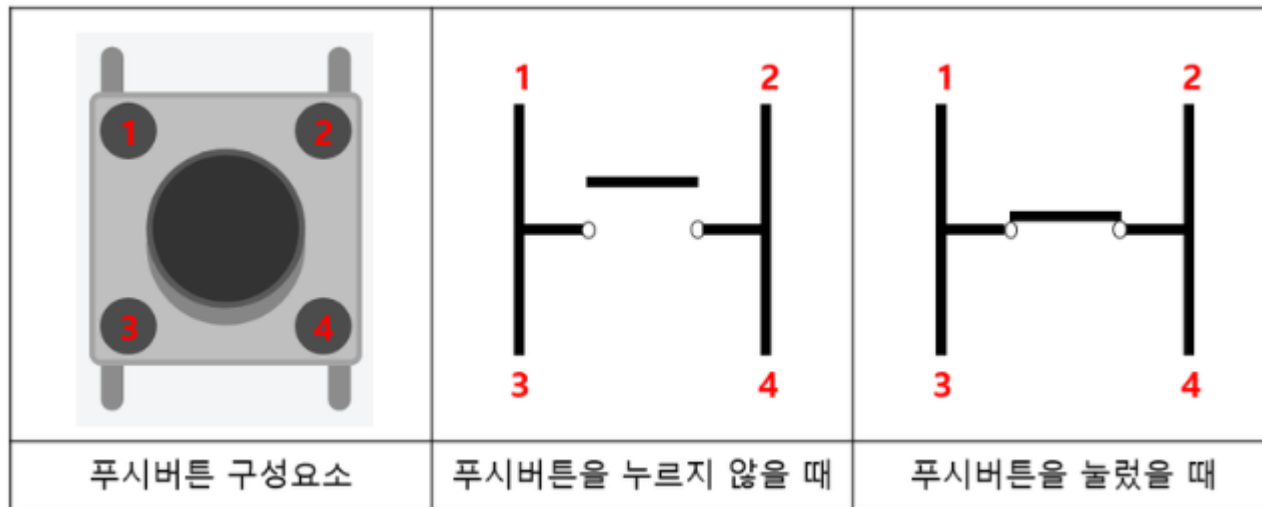
- 결국 전자적 관점에서 볼 때 모든 컴퓨터(범용 프로세서, 전용 프로세서 또는 컨트롤러)는 데이터를 입력하고 입력을 처리하고 데이터 또는 전자 신호를 출력할 수 있는 디지털 전자 회로이다.
- "입력-처리-출력"이 주어지면 컴퓨터는 지정된 작업이나 작업을 수행할 수 있다.
- 라즈베리파이는 단일 보드 컴퓨터로서 출력과 입력도 가능하다.
- GPIO 핀은 3.3V TTL 내전압성이므로 그에 따라 논리 신호를 출력 및 입력할 수 있다.
- RPi의 GPIO에 대한 입력 신호는 순간 유형 스위치를 통한 데이터 입력에 사용할 수 있다.



## Section 03 버튼 제어

### □ 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 03 버튼 제어

---

### □ 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 03 버튼 제어

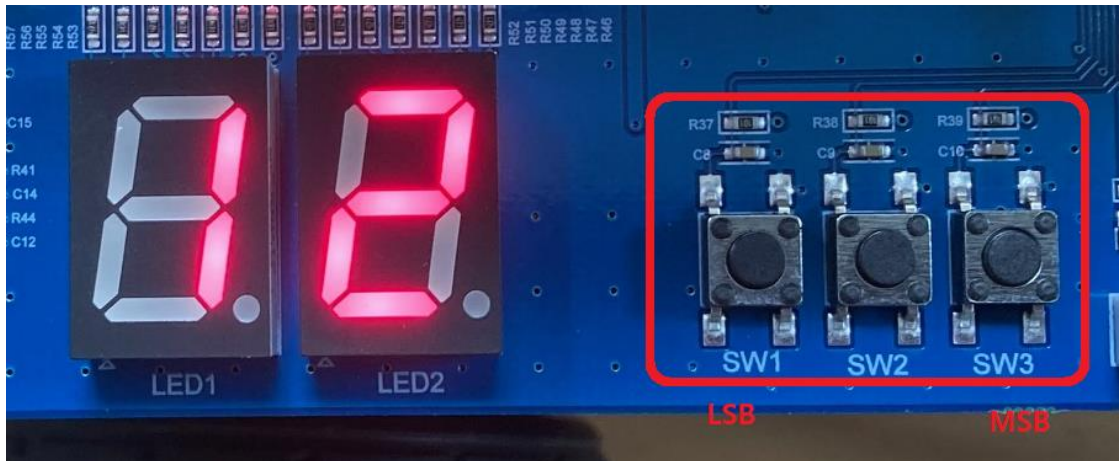
### □ 디지털 입력 감지

- 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 03 버튼 제어

### □ 자동차 구조

- 자동차에는 3개의 스위치가 있다.
- 스위치는 3비트 값으로 표현되며, 눌렀을 때 0, 누르지 않으면 1의 상태가 된다.



## Section 03 버튼 제어

### □ 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 swcar.SIO_WriteLED(0)
13 high = 0
14 new = 0
15
```

## Section 03 버튼 제어

### □ 버튼 제어 코드

- 다음과 같은 코드를 작성하여 스위치의 값을 입력받아 보자.

```
15 while True:
16     switch = swcar.SIO_ReadSwitch()
17     if (switch // 1 % 2) == 0:
18         new = 1
19     elif (switch // 2 % 2) == 0:
20         new = 2
21     elif (switch // 4 % 2) == 0:
22         new = 3
23
24     if (new > 0)
25         swcar.SIO_WriteLED(high * 10 + new)
26         high = new
27         new = 0
28
29     time.sleep(0.1)
```

## Section 03 버튼 제어

---

### □ 버튼 제어 코드

- Run을 클릭하여 코드를 실행한다.
- 스위치를 누르게 되면, 스위치의 위치에 따라 1,2,3 의 값이 7 Segment LED를 통해 표시되며, 신규값이 우측에 표시되고, 과거값이 왼쪽으로 Shift 되도록 만들어 졌다.
- 스위치를 좌측에서부터 순차적으로 누르게 되면, 7 Segment LED 의 표시 값이 00 > 01 > 12 > 23 로 변화 하는 것을 볼 수 있을 것이다.

---

# Q&A

