Chapter 02
파이썬 패키지 사용하기

이번 장에서는 라즈베리파이에서 제공하는 파이썬 패키지의 사용법을 익혀봅니다.

01 print 함수

여기서는 print 함수에 대한 사용법을 익혀봅니다. print 함수는 문자열과 숫자를 출력해 주는 함수로 프로그램 내부의 중요한 정보를 사용자에게 알려줍니다. 파이썬에서 문자열을 출력하고자 할 경우엔 print 함수를 사용하면 됩니다.

01 print

print 함수는 문자열과 숫자를 출력해 주는 함수로 프로그램 내부의 중요한 정보를 사용자에게 알려줍니다.

1. 다음과 같이 예제를 작성합니다.

_01_print.py
```
1 : print("Hello. I'm Raspberry Pi~")
```
1 : print 함수를 호출하여 “Hello. I'm Raspberry Pi~" 문자열을 출력합니다.

2. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _01_print.py
```

프로그램을 실행시키면 “Hello. I'm Raspberry Pi~" 문자열이 출력됩니다.

```
Hello. I'm Raspberry Pi~
```

02 while

어떤 일을 반복하고자 할 경우엔 while 문을 사용합니다.

1. 다음과 같이 예제를 수정합니다.

_02_while_ture.py
```
1 : while True:
2 :     print("Hello. I'm Raspberry Pi~")
```
1 : while True 문을 수행하여 1,2 줄을 무한 반복합니다. 온점(:)은 while 문의 시작을 나타냅니다. while 문 안에서 동작하는 문장은 while 문 보다 탭 문자 하나만큼 들여 써야 합니다. 또는 스페이스 문자 4 개만큼 들여 쓰기도 합니다. 일반적으로 스페이스 문자 4만큼 들여 씁니다. 여기서는 print 함수가 while 문의 영향을 받습니다.
2 : print 함수를 호출하여 “Hello. I'm Raspberry Pi~" 문자열을 출력합니다.

2. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _02_while_ture.py
```

프로그램을 실행시키면 “Hello. I'm Raspberry Pi~" 문자열이 빠른 속도로 무한 출력됩니다.

```
Hello. I'm Raspberry Pi~
Hello. I'm Raspberry Pi~
Hello. I'm Raspberry Pi~
```

프로그램을 강제 종료하기 위해서는 CTRL 키를 누른 채로 c 키를 눌러줍니다. 그러면 다음과 같이 키보드 인터럽트가 발생했다는 메시지가 뜹니다.

```
^CHello. I'm Raspberry Pi~
Traceback (most recent call last):
  File "_02_while_ture.py", line 2, in <module>
    print("Hello. I'm Raspberry Pi~")
KeyboardInterrupt
```

인터럽트 처리 메시지를 보이지 않게 하기 위해서는 키보드 인터럽트를 직접 처리해주면 됩니다.

03 try~except

여기서는 키보드 인터럽트를 처리하기 위해 try~except문을 살펴봅니다.

1. 다음과 같이 예제를 작성합니다.

_03_try_except.py

```
1 : try:
2 :     while True:
3 :         print("Hello. I'm Raspberry Pi~")
4 : except KeyboardInterrupt:
5 :     pass
```

1,4 : 키보드 인터럽트를 처리하기 위해 try~except 문을 사용합니다. try~except 문은 예외처리를 하고자 할 경우 사용하며 여기서는 키보드 인터럽트 처리를 위해 사용하고 있습니다. try~except 문은 정상적인 처리와 예외 처리를 따로 구분하여 코드에 대한 가독성을 높여 주는 역할을 합니다. try 문 아래에서 정상적인 처리를 하며 정상적인 처리를 하다가 예외가 발생할 경우 except 문으로 이동하여 처리합니다.
2 : while 문을 try 문보다 탭 문자 하나만 큼 들여 써서 try문 안에서 동작하게 합니다.
5 : pass문을 이용하여 아무것도 수행하지 않습니다. pass 문은 특별히 수행하고자 할 코드가 없을 경우 사용합니다. 온점(:)으로 시작하는 구문, 예를 들어 여기서는 try, except, while 문은 적어도 하나의 문장을 수행해야 하며, 굳이 수행하고자 하는 문장이 없을 경우 pass 문을 사용합니다.

2. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _03_try_except.py
```

프로그램을 실행시키면 “Hello. I'm Raspberry Pi~" 문자열이 빠른 속도로 반복해서 출력됩니다.

```
Hello. I'm Raspberry Pi~
Hello. I'm Raspberry Pi~
Hello. I'm Raspberry Pi~
```

프로그램을 강제 종료하기 위해서 CTRL 키를 누른 채로 c키를 눌러줍니다. 키보드 인터럽트에 대해 pass 문을 이용하여 처리하는 것을 볼 수 있습니다.

```
Hello. I'm Raspberry Pi~
Hello. I'm Raspberry Pi~
^CHello. I'm Raspberry Pi~
```

*** 처리해야할 오류가 둘일 경우엔 try~except~except 문을 사용하여 처리할 수 있습니다. 즉, 오류의 종류에 따라 except 문을 추가할 수 있습니다.

04 time.sleep

시간에 대한 지연을 주고자 할 경우엔 time 라이브러리의 sleep 함수를 사용합니다.

1. 다음과 같이 예제를 작성합니다.

_04_time.py

```
1 : import time
2 :
3 : try:
4 :     while True:
5 :         print("Hello. I'm Raspberry Pi~")
6 :         time.sleep(0.5)
7 : except KeyboardInterrupt:
8 :     pass
```

1 : time 모듈을 불러옵니다. 6줄에서 time 모듈이 제공하는 sleep 함수를 사용하기 위해 필요합니다.
6 : time 모듈이 제공하는 sleep 함수를 호출하여 0.5초간 지연을 줍니다.

2. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _04_time.py
```

프로그램을 실행시키면 “Hello. I'm Raspberry Pi~" 문자열이 0.5초마다 반복해서 출력됩니다.

```
Hello. I'm Raspberry Pi~
Hello. I'm Raspberry Pi~
Hello. I'm Raspberry Pi~
```

프로그램을 강제 종료하기 위해서는 CTRL 키를 누른 채로 c키를 눌러줍니다.

05 문자열, 숫자 출력하기

여기서는 print 함수를 이용하여 문자열, 숫자를 출력해봅니다.

1. 다음과 같이 예제를 작성합니다.

_05_print.py

```
1 : print("Hello. I'm Raspberry Pi~")
2 : print(78)
3 : print(1.23456)
```

1 : 문자열을 출력합니다.
2 : 10진수 정수 78을 출력합니다.
3 : 실수 1.23456을 10진 실수 문자열로 변환하여 출력합니다.

2. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _05_print.py
```

실행 결과는 다음과 같습니다.

```
Hello. I'm Raspberry Pi~
78
1.23456
```

06 형식 문자열 사용하기

여기서는 문자열 형식을 이용하여 문자열, 숫자를 출력해봅니다. 문자열 형식은 출력하고자 하는 문자열 내에 % 문자를 이용하여 문자열과 숫자를 표시하는 방법입니다.

1. 다음과 같이 예제를 작성합니다.

_05_print_2.py

```
1 : print("%s" %"Hello. I'm Raspberry Pi~")
2 : print("%d" %78)
3 : print("%f" %1.23456)
```

1 : %s는 문자열 형식(string format)을 나타내며 %s 자리에 들어갈 문자열은 %"Hello. I'm

Raspberry Pi~" 부분이 됩니다. 주의할 점은 첫 번째 문자열과 두 번째 문자열 사이에 쉼표(,)가 들어가지 않습니다.
2 : %d는 십진수 형식(decimal format)을 나타내며 %d 자리에 들어갈 문자열은 %78 부분이 됩니다.
3 : %f는 실수 형식(floating point format)을 나타내며 %f 자리에 들어갈 문자열은 %1.23456 부분이 됩니다.

2. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _05_print_2.py
```

실행 결과는 다음과 같습니다.

```
Hello. I'm Raspberry Pi~
78
1.234560
```

07 정수, 실수 출력하기

여기서는 문자열 형식을 이용하여 10진수와 16진수 정수를 출력해 봅니다. 또, 10진 실수의 소수점이하 출력을 조절해 봅니다.

1. 다음과 같이 예제를 작성합니다.

_05_print_3.py

```
1 : print("%d" %78)
2 : print("%d %x" %(78, 78))
3 : print("%.0f" %1.23456)
4 : print("%.2f" %1.23456)
5 : print("%.4f" %1.23456)
```

1 : %d 형식은 정수를 10진수 문자열로 변환하는 형식입니다. 여기서는 정수 78을 10진수 문자열로 변환하여 출력합니다. %d에 맞추어 출력할 숫자는 % 뒤에 붙여줍니다. %78과 같이 % 뒤에 10진수 78을 붙였습니다.
2 : %x 형식은 정수를 16진수 문자열로 변환하는 형식입니다. 여기서는 정수 78을 10진수와 16진수 문자열로 변환하여 출력합니다. 포맷이 하나 이상일 경우엔 %뒤에 ( ) 안에 넣어줍니다. % 뒤에 (78, 78)를 붙였습니다.
3 : 실수 1.23456을 소수점 이하 0개까지 10진 실수 문자열로 변환하여 출력합니다.
4 : 실수 1.23456을 소수점 이하 2개까지 10진 실수 문자열로 변환하여 출력합니다.
5 : 실수 1.23456을 소수점 이하 4개까지 10진 실수 문자열로 변환하여 출력합니다.

2. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _05_print_3.py
```

실행 결과는 다음과 같습니다.

```
78
78 4e
1
1.23
1.2346
```

08 str.format 함수 사용해 보기

이전 예제에서 살펴본 %를 이용한 문자열 출력은 C에서 사용하던 방식입니다. 여기서는 파이썬3 이후부터 지원하는 str.format 함수를 이용한 방법을 소개합니다.

1. 다음과 같이 이전 예제를 수정합니다.

_05_print_4.py

```
1 : print("{}".format(78))
2 : print("{} {:x}".format(78, 78))
3 : print("{:.0f}".format(1.23456))
4 : print("{:.2f}".format(1.23456))
5 : print("{:.4f}".format(1.23456))
```

1 : 정수 78을 출력합니다. str.format 함수는 출력하고자 하는 문자열에 대해 format 함수를 붙여서 사용합니다. format 함수의 인자에 대응하는 문자열은 중괄호 {}로 표현합니다.
2 : 정수 78을 십진수와 십육진수로 표현합니다. 십육진수로 표현하고자 할 경우엔 중괄호 {} 안에 형식 문자를 넣어줍니다. 십육진수의 형식 문자는 :x입니다. 이전 예제에서 %대신 :을 사용합니다. 문자열 내의 첫 번째 중괄호는 format 함수의 첫 번째 인자, 두 번째 중괄호는 두 번째 인자에 대응됩니다.
3 : 실수 1.23456 값을 소수점 이하 0개까지 10진 실수 문자열로 변환하여 출력합니다. 실수의 기본 형식은 :f입니다. 이전 예제에서 %대신 :을 사용합니다.
4 : 실수 1.23456 값을 소수점 이하 2개까지 10진 실수 문자열로 변환하여 출력합니다.
8 : 실수 1.23456 값을 소수점 이하 4개까지 10진 실수 문자열로 변환하여 출력합니다.

2. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _05_print_4.py
```

실행 결과는 다음과 같습니다.

```
78
78 4e
1
1.23
1.2346
```

09 파이썬 수행 속도 측정하기

라즈베리파이 상에서 파이썬은 얼마나 빨리 동작할까요? 여기서는 라즈베리파이 상에서 파이썬이 얼마나 빨리 동작하는지 테스트해보도록 합니다.

1. 다음과 같이 예제를 작성합니다.

_05_print_5.py

```
1  : import time
2  : start = time.time()
3  :
4  : cnt =0
5  : while True:
6  :     cnt = cnt +1
7  :     if cnt>10000000:
8  :             break
9  :
10 : end = time.time()
11 : print(cnt)
12 : print(end - start)
```

1 : time 모듈을 불러옵니다. time 모듈은 현재 시간을 측정할 수 있는 time 함수를 가지고 있습니다.
2 : time.time 함수를 호출하여 현재 시간을 얻어와 start 변수에 저장합니다.
4 : cnt 변수를 선언한 후, 0으로 초기화합니다.
5 : 계속해서 5~8줄을 수행합니다.
6 : cnt 변수를 하나 증가시킵니다.
7 : cnt 변수 값이 10000000(천만)보다 크면
8 : break 문을 이용하여 while 문을 빠져나와 10줄로 이동합니다.
10 : time.time 함수를 호출하여 현재 시간을 얻어와 end 변수에 저장합니다.
11 : print 함수를 호출하여 cnt 변수 값을 출력합니다.
12 : print 함수를 호출하여 (end-start) 값을 출력합니다. 5~8줄을 1000만 번 수행하는데 걸리는 시간을 출력합니다.

2. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _05_print_5.py
```

실행 결과는 다음과 같습니다. 다음은 Raspberry Pi 4에서 파이썬의 실행 속도입니다.

```
10000001
2.763632059097290
```

cnt 변수를 천만번 세는데 2.76초 정도가 걸립니다.

*** 참고로 다음은 Raspberry Pi 3에서 파이썬의 실행 속도입니다.

```
10000001
8.56763482093811
```

cnt 변수를 천만번 세는데 8.57초 정도가 걸립니다.

*** 참고로 다음은 i7-8750H CPU 기반의 PC에서 파이썬의 실행 속도입니다.

```
========== RESTART: C:/Users/edu/Desktop/perf3.py ==========
10000001
0.8551285266876221
>>>
```

cnt 변수를 천만번 세는데 0.855초 정도가 걸립니다. Raspberry Pi 4보다 3.2배 정도 빠른 속도입니다.

C 언어 수행 속도 측정하기

*** 이 책의 주제와는 직접적으로 상관이 없는 예제이므로 그냥 넘어가셔도 됩니다.

그러면 라즈베리파이 상에서 C 언어는 얼마나 빨리 동작할까요? 여기서는 라즈베리파이 상에서 C 언어가 얼마나 빨리 동작하는지 테스트해보도록 합니다.

1. 다음과 같이 예제를 작성합니다.

_05_print_6.c

```
1  : #include <stdio.h>
2  : #include <sys/time.h>
3  :
4  : long micros()
5  : {
6  :     struct timeval currentTime;
7  :     gettimeofday(&currentTime, NULL);
8  :     return currentTime.tv_sec *1000000 + currentTime.tv_usec;
9  : }
10 :
11 : int main()
12 : {
13 :     int cnt;
14 :     long start, end;
15 :
16 :     start = micros();
17 :     while (1==1) {
18 :             cnt = cnt +1;
19 :             if (cnt>10000000)
20 :                     break;
21 :     }
22 :     end = micros();
23 :     printf("%d\n", cnt);
24 :     printf("%f\n", (end-start)/1000000.0);
25 : }
```

1 : stdio.h 파일을 포함합니다. 23,24줄에 있는 printf 함수를 사용하기 위해 필요합니다.

2 : sys/time.h 파일을 포함합니다. 6,7줄에 있는 timeval 구조체와 gettimeofday 함수를 사용하기 위해 필요합니다.
4~9 : micros 함수를 정의합니다.
6 : timeval 구조체 변수인 currentTime 변수를 선언합니다.
7 : gettimeofday 함수를 호출하여 현재 시간을 currentTime 변수로 가져옵니다.
8 : 현재 시간의 초(tv_sec)에 1000000(=백만)을 곱한 후, 현재 시간의 마이크로 초(tv_usec)를 더해서 함수 결과 값으로 줍니다. 이렇게 하면 현재 시간을 마이크로 초 단위의 결과 값으로 주게 됩니다. 1초는 1000000(=백만)마이크로초입니다.
11~25 : main 함수를 정의합니다.
13 : cnt 정수 변수를 선언한 후, 0으로 초기화합니다.
14 : start, end 정수 변수를 선언합니다. start, end 변수는 long 형의 정수 변수입니다.
16 : micros 함수를 호출하여 현재 시간을 얻어와 start 변수에 저장합니다.
17 : 계속해서 17~20줄을 수행합니다. (1==1)은 항상 맞는 상태를 나타냅니다.
18 : cnt 변수를 하나 증가시킵니다.
19 : cnt 변수 값이 10000000(천만)보다 크면
20 : break 문을 이용하여 17줄에 있는 while 문을 빠져 나와 22줄로 이동합니다.
22 : micros 함수를 호출하여 현재 시간을 얻어와 end 변수에 저장합니다.
23 : printf 함수를 호출하여 cnt 변수 값을 출력합니다.
24 : printf 함수를 호출하여 (end-start)/1000000.0 값을 출력합니다. 17~20줄을 1000만 번 수행하는데 걸리는 시간을 초 단위로 출력합니다.

2. 다음과 같이 예제를 실행합니다.

```
$ gcc _05_print_6.c -o _05_print_6
$ ./_05_print_6
```

실행 결과는 다음과 같습니다. 다음은 Raspberry Pi 4에서 C 언어의 실행 속도입니다.

```
10000001
0.108881
```

cnt 변수를 천만번 세는데 0.11초 정도가 걸립니다. 파이썬 언어보다 약 25배 빠릅니다.

참고로 다음은 Raspberry Pi 3에서 C 언어의 실행 속도입니다.

```
10000001
0.112728
```

C 언어의 경우 Raspberry Pi 3 와 4의 속도와 거의 같습니다.

02 Rpi.GPIO.output 함수

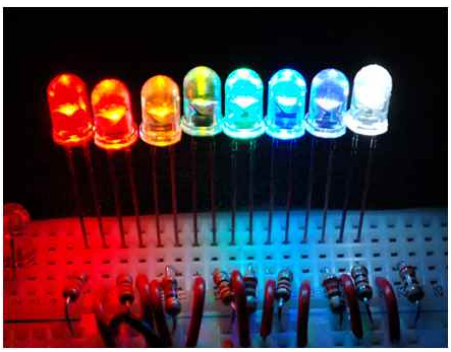
RPi.GPIO 모듈이 제공하는 output 함수는 할당된 핀에 True 또는 False을 써서 할당된 핀을 VCC 또는 GND로 연결하는 역할을 합니다. 여기서는 RPi.GPIO 모듈이 제공하는 output 함수를 이용하여 LED를 켜보고 꺼보는 예제를 수행해 봅니다.

01 부품 살펴보기

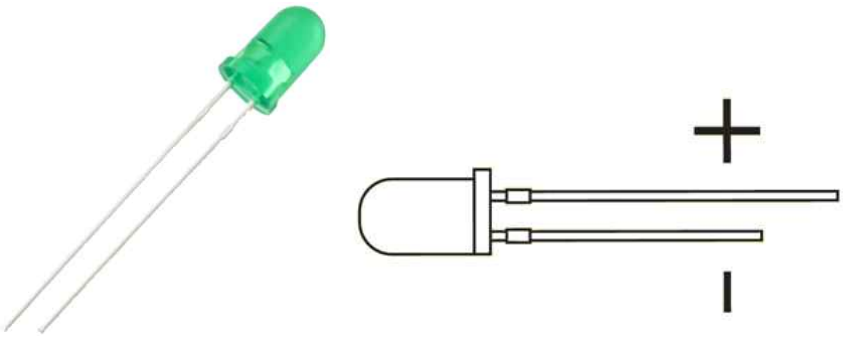
먼저 GPIO와 관련된 부품을 살펴보도록 합니다.

LED

LED는 크기나 색깔, 동작 전압에 따라 여러 가지 형태가 존재합니다.

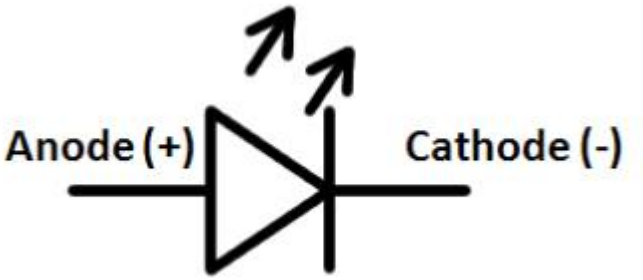


LED의 모양은 다음과 같으며, 긴 핀과 짧은 핀을 갖습니다.

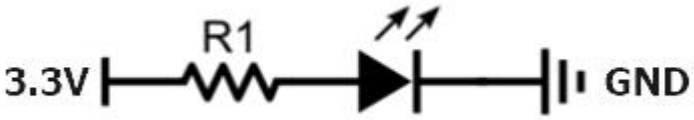


LED는 방향성이 있습니다. 즉, 회로에 연결할 때 방향을 고려해야 합니다. 긴 핀을 전원의 양극(VCC, 3.3V), 짧은 핀을 음극(GND, 0V)으로 연결합니다. 반대로 연결할 경우 전류가 흐르지 못해 LED가 켜지지 않습니다.

LED를 나타내는 기호는 다음과 같습니다. Anode(+)에서 Cathode(-)로 전류가 흐릅니다.

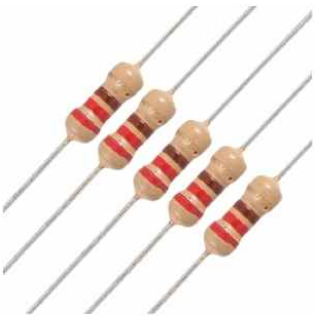


LED는 저항과 직렬로 연결해야 하며, 라즈베리파이 보드에서는 3.3V와 0V 사이에 연결해 줍니다. LED를 위한 저항은 보통 220 Ohm 또는 330 Ohm을 사용합니다.



저항

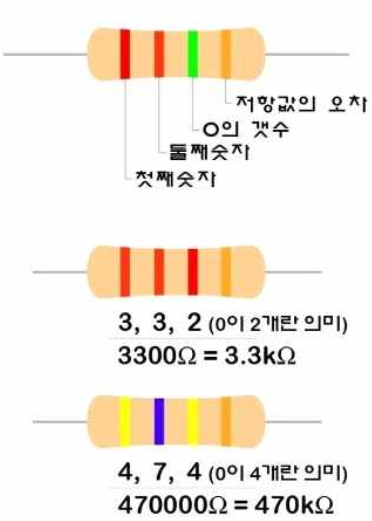
다음 저항은 220 Ohm 저항입니다. 저항은 전류의 양을 조절하는 역할을 합니다. 저항은 방향성이 없기 때문에 VCC와 GND에 어떤 방향으로도 연결할 수 있습니다.



다음은 저항 기호를 나타냅니다.

❶ 저항 읽는 법

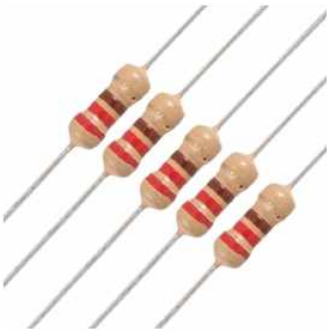
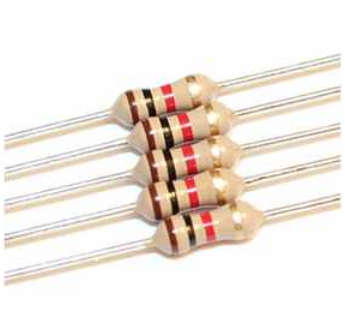
다음 그림을 이용하면 저항 값을 읽을 수 있습니다.



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

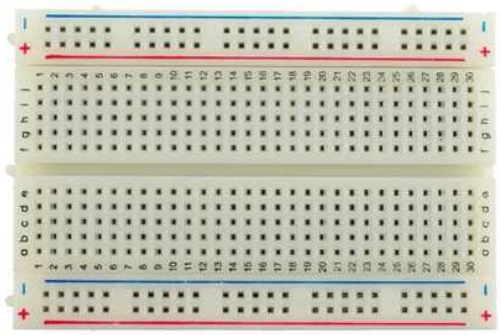
❷ 자주 사용하는 저항

다음은 이 책에서 주로 사용하는 저항의 종류입니다.

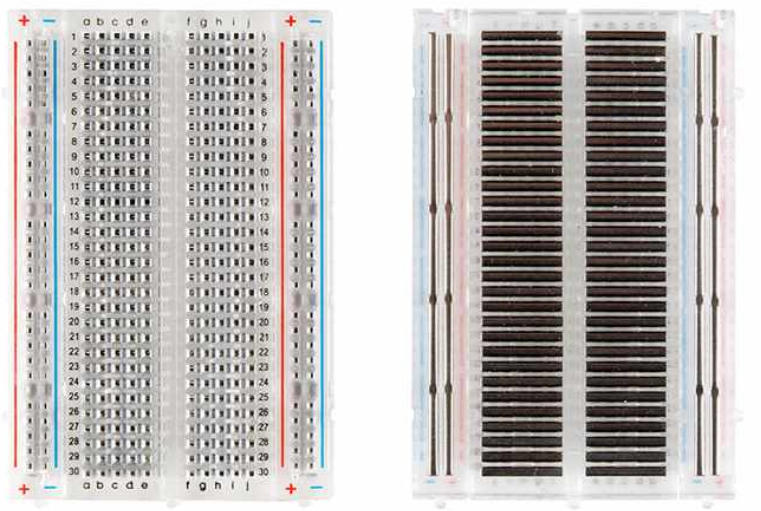
<table>
<tr>
<td> 
220 Ohm</td>
<td> 
330 Ohm</td>
</tr>
<tr>
<td> 
1K Ohm</td>
<td> 
10K Ohm</td>
</tr>
</table>

브레드 보드

다음은 브레드 보드입니다. 브레드 보드를 사용하면 납땜을 하지 않고, 시험용 회로를 구성할 수 있습니다. 일반적으로 빨간 선을 VCC, 파란 선을 GND에 연결합니다.



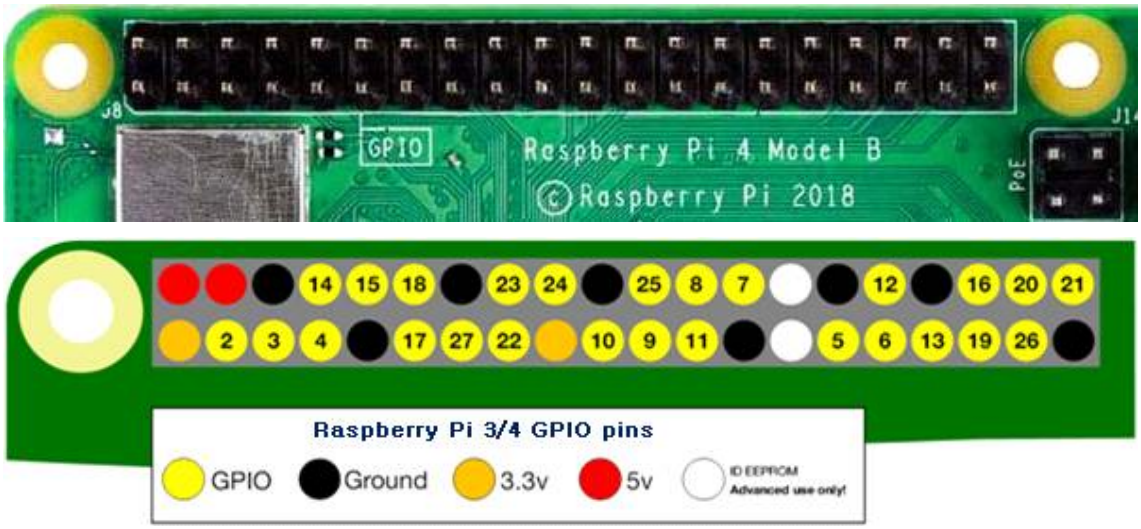
브레드 보드의 내부 구조는 다음과 같으며, 전선의 집합입니다.



전선

다음은 이 책에서 사용하는 전선입니다.

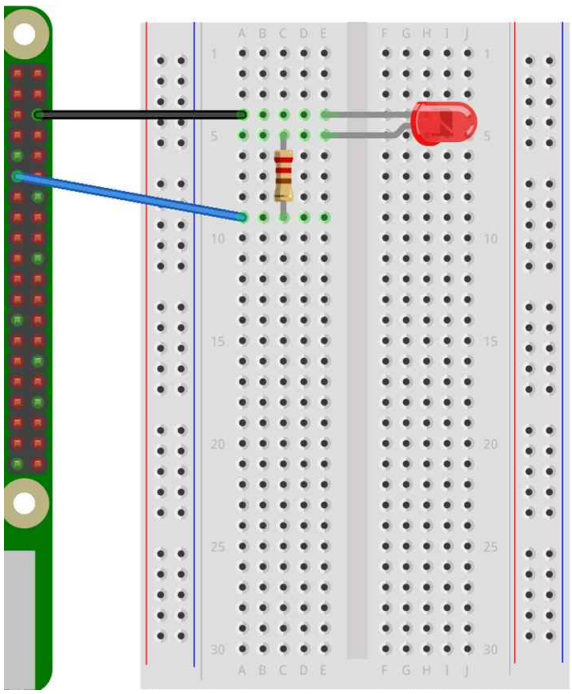
라즈베리파이 보드



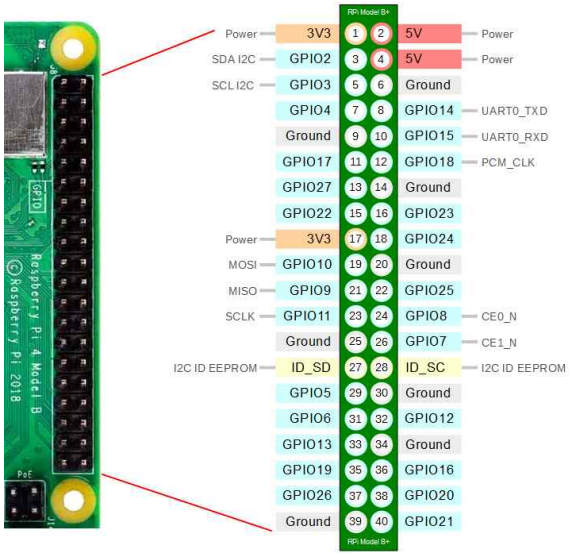
Rpi.GPIO.output 함수를 통해 제어할 수 있는 핀은 노란색으로 표시된 GPIO 핀들입니다. GPIO(General Purpose Input Output)는 범용 목적 입력 출력을 의미하며, 라즈베리파이 SOC(System on Chip) 내부에 있는 CPU, UART, I2C, Timer 등의 하드웨어 모듈에 할당하여 사용하는 입출력 핀입니다.

02 LED 회로 구성하기

다음과 같이 회로를 구성합니다.



다음 핀 맵을 참조합니다.



LED의 긴 핀(+)을 220 Ohm 저항을 통해 라즈베리파이 보드의 GPIO 17번 핀에 연결합니다. LED의 짧은 핀(-)은 GND 핀에 연결합니다.

03 LED 켜고 끄기

여기서는 Rpi.GPIO.output 함수를 이용하여 LED를 켜고 꺼봅니다.

LED 켜기

먼저 Rpi.GPIO.output 함수를 이용하여 LED를 켜 봅니다.

1. 다음과 같이 예제를 작성합니다.

_06_gpio_output.py

```
1  : import RPi.GPIO as GPIO
2  :
3  : led_pin =17
4  :
5  : GPIO.setmode(GPIO.BCM)
6  :
7  : GPIO.setup(led_pin, GPIO.OUT)
8  :
9  : GPIO.output(led_pin, True)
10 :
11 : try:
12 :     while True:
13 :             pass
14 : except KeyboardInterrupt:
15 :     pass
16 :
17 : GPIO.cleanup()
```

1 : RPi.GPIO 모듈을 GPIO라는 이름으로 불러옵니다. RPi.GPIO 모듈은 5,7,9,17줄에 있는 setmode, setup, output, cleanup 함수들을 가지고 있으며 이 함수들을 사용하기 위해 필요합니다.

3 : led_pin 변수를 선언한 후, 17로 초기화합니다. 여기서 17은 BCM GPIO 핀 번호를 나타냅니다.

5 : GPIO.setmode 함수를 호출하여 BCM GPIO 핀 번호를 사용하도록 설정합니다.

7 : GPIO.setup 함수를 호출하여 led_pin을 GPIO 출력으로 설정합니다. 이렇게 하면 led_pin으로 True 또는 False를 써 led_pin에 연결된 LED를 켜거나 끌 수 있습니다.

9 : GPIO.output 함수를 호출하여 led_pin을 True로 설정합니다. 이렇게 하면 led_pin에 연결된 LED가 켜집니다.

12,13 : 빈 while 문을 수행하여 LED가 켜진 상태를 유지하도록 합니다. while 문을 끝내기 위해서는 CTRL+c 키를 누릅니다. CTRL+c 키를 누르면 14줄로 이동하여 키보드 인터럽트를 처리합니다.

17 : GPIO.cleanup 함수를 호출하여 GPIO 핀의 상태를 초기화해 줍니다.

2. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _06_gpio_output.py
```

LED가 켜지는 것을 확인합니다. CTRL+c 키를 누르면 LED가 꺼지면서 프로그램의 수행이 멈추게 됩니다.

파이썬 쉘 도움말 보기

독자 여러분은 앞으로 다양한 함수를 사용해 라즈베리파이를 활용하게 됩니다. 그래서 여기서는 파이썬 쉘을 통해 함수에 대한 설명을 볼 수 있는 방법을 소개합니다.

1. 다음과 같이 차례대로 명령을 실행합니다.

```
pi@raspberrypi:~/pyLabs $ python3 ❶
Python 3.7.3 (default, Dec 20 2019, 18:57:59)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import RPi.GPIO as GPIO ❷
>>> help(GPIO.setmode) ❸
```

❶ python3 명령을 주어 파이썬 쉘을 실행시킵니다. ❷ RPi.GPIO 모듈을 GPIO라는 이름으로 불러옵니다. 모듈이 가진 함수나 변수 등의 설명글을 보기 위해서 먼저 import 해야 합니다. ❸ help(GPIO.setmode) 명령을 이용해 GPIO.setmode 함수를 살펴봅니다. 살펴보고자 하는 함수나 변수 등을 help 명령의 인자로 주면 됩니다.

2. 그러면 다음과 같이 설명글이 나옵니다.

```
Help on built-in function setmode in module RPi._GPIO: ❶

setmode(...) ❷
    Set up numbering mode to use for channels. ❸
    BOARD - Use Raspberry Pi board numbers
    BCM   - Use Broadcom GPIO 00..nn numbers ❹
(END)
```

❶ RPi._GPIO 모듈 안에 있는 내장 setmode 함수에 대한 설명글입니다.
❷ setmode 함수를 나타냅니다.
❸ 핀 번호 형식을 설정합니다. 함수의 기능을 설명하는 부분입니다.
❹ 핀 번호 형식을 설명합니다. BOARD는 라즈베리파이 보드의 번호를 사용하는 형식이고, BCM은 Broadcom 사에서 제공하는 GPIO 번호를 사용하는 형식입니다.

종료는 q 문자를 입력합니다.

3. GPIO.setup, GPIO.output, GPIO.cleanup 함수도 같은 방법으로 살펴봅니다.

```
help(GPIO.setup)
```

```
help(GPIO.output)
help(GPIO.cleanup)
```

4. 파이썬 쉘을 빠져 나올 때는 다음과 같이 quit() 함수를 수행해 줍니다.

```
>>> quit()
pi@raspberrypi:~ $
```

LED 끄기

다음은 Rpi.GPIO.output 함수를 이용하여 LED를 꺼 봅니다.

1. 다음과 같이 예제를 수정합니다.

_06_gpio_output_2.py

```
1  : import RPi.GPIO as GPIO
2  : import time
3  :
4  : led_pin =17
5  :
6  : GPIO.setmode(GPIO.BCM)
7  :
8  : GPIO.setup(led_pin, GPIO.OUT)
9  :
10 : GPIO.output(led_pin, True)
11 : time.sleep(2.0)
12 : GPIO.output(led_pin, False)
13 :
14 : GPIO.cleanup()
```

2 : sleep 함수를 사용하기 위하여 time 모듈을 불러옵니다.

10 : GPIO.output 함수를 호출하여 led_pin을 True로 설정합니다. 이렇게 하면 led_pin에 연결된 LED가 켜집니다.

11 : time.sleep 함수를 호출하여 2.0초간 기다립니다.

12 : GPIO.output 함수를 호출하여 led_pin을 False로 설정합니다. 이렇게 하면 led_pin에 연결된 LED가 꺼집니다.

2. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _06_gpio_output_2.py
```

LED가 켜졌다 2.0초 후에 꺼지는 것을 확인합니다.

04 LED 점멸 반복해보기

여기서는 Rpi.GPIO.output 함수를 이용하여 LED를 켜고 끄고를 반복해 봅니다.

1. 다음과 같이 예제를 수정합니다.

_06_gpio_output_3.py

```
1  : import RPi.GPIO as GPIO
2  : import time
3  :
4  : led_pin =17
5  :
6  : GPIO.setmode(GPIO.BCM)
7  :
8  : GPIO.setup(led_pin, GPIO.OUT)
9  :
10 : try:
11 :     while True:
12 :         GPIO.output(led_pin, True)
13 :         time.sleep(0.5)
14 :         GPIO.output(led_pin, False)
15 :         time.sleep(0.5)
16 : except KeyboardInterrupt:
17 :     pass
18 :
19 : GPIO.cleanup()
```

11 : 계속해서 11~15줄을 수행합니다.

12 : GPIO.output 함수를 호출하여 led_pin을 True로 설정합니다. 이렇게 하면 led_pin에 연결된 LED가 켜집니다.

13 : time.sleep 함수를 호출하여 0.5 초간 지연을 줍니다.

14 : GPIO.output 함수를 호출하여 led_pin을 False로 설정합니다. 이렇게 하면 led_pin에 연결된 LED가 꺼집니다.

15 : time.sleep 함수를 호출하여 0.5 초간 지연을 줍니다.

2. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _06_gpio_output_3.py
```

1초 주기로 LED가 켜졌다 꺼졌다 하는 것을 확인합니다. 즉, 1Hz의 주파수로 LED가 점멸하는 것을 확인합니다.

LED의 점등은 led_pin을 통해 나오는 True 값에 의해 발생합니다. LED의 소등은 led_pin을 통해 나오는 False 값에 의해 발생합니다. 즉, led_pin으로는 위 그림과 같이 True, False 값에 의해 HIGH, LOW 신호가 1초 주기로 나오게 되며, 이 값들에 의해 LED는 점멸을 반복

하게 됩니다. 그리고 이 경우 여러분은 LED가 점멸 하는 것을 느낄 수 있습니다.

프로그램을 강제 종료하기 위해서는 CTRL 키를 누른 채로 c키를 눌러줍니다.

05 LED 점멸 간격 줄여보기

여기서는 Rpi.GPIO.output 함수와 time.sleep 함수를 이용하여 아래와 같은 사각 파형에 대한 주파수와 상하비의 개념을 이해해 보도록 합니다.

주파수란 1초간 반복되는 사각 파형의 개수를 의미하며, 상하비란 사각 파형의 HIGH 구간과 LOW 구간의 비를 의미합니다.

이제 LED의 점멸 간격을 줄여보도록 합니다. 그러면 여러분은 좀 더 조밀하게 LED가 점멸하는 것을 느낄 것입니다.

1. 다음과 같이 예제를 수정합니다.

_06_gpio_output_4.py

```
1  : import RPi.GPIO as GPIO
2  : import time
3  :
4  : led_pin =17
5  :
6  : GPIO.setmode(GPIO.BCM)
7  :
8  : GPIO.setup(led_pin, GPIO.OUT)
9  :
10 : try:
11 :   while True:
12 :           GPIO.output(led_pin, True)
13 :           time.sleep(0.05)
14 :           GPIO.output(led_pin, False)
15 :           time.sleep(0.05)
16 : except KeyboardInterrupt:
17 :   pass
18 :
19 : GPIO.cleanup()
```

13,15 : 0.5를 0.05로 변경합니다. 즉, 0.05 초간 지연을 줍니다.

2. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _06_gpio_output_4.py
```

이 예제의 경우 LED는 초당 10번 점멸 하게 됩니다. 즉, 10Hz의 주파수로 점멸하게 됩니다.

그림과 같은 파형이 초당 10개가 생성됩니다. 이 경우에도 여러분은 반복적으로 LED가 점멸하는 것을 느낄 것입니다. 그러나 그 간격은 더 조밀하게 느껴질 것입니다.

프로그램을 강제 종료하기 위해서는 CTRL 키를 누른 채로 c키를 눌러줍니다.

06 LED 점멸을 밝기로 느껴보기

LED의 점멸 간격을 더 줄여보도록 합니다. 여기서 여러분은 LED의 점멸을 느끼지 못하게 될 것입니다. 오히려 LED가 일정한 밝기로 켜져 있다고 느낄 것입니다.

1. 다음과 같이 예제를 수정합니다.

_06_gpio_output_5.py

```
1  : import RPi.GPIO as GPIO
2  : import time
3  :
4  : led_pin =17
5  :
6  : GPIO.setmode(GPIO.BCM)
7  :
8  : GPIO.setup(led_pin, GPIO.OUT)
9  :
10 : try:
11 :   while True:
12 :             GPIO.output(led_pin, True)
13 :             time.sleep(0.005)
14 :             GPIO.output(led_pin, False)
15 :             time.sleep(0.005)
16 : except KeyboardInterrupt:
17 :   pass
18 :
19 : GPIO.cleanup()
```

13,15 : 0.05를 0.005으로 변경합니다. 즉, 0.005 초간 지연을 줍니다.

2. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _06_gpio_output_5.py
```

이 예제의 경우 LED는 초당 100번 점멸 하게 됩니다. 즉, 100Hz의 주파수로 점멸하게 됩니

다.

그림과 같은 파형이 초당 100개가 생성됩니다. 이제 여러분은 LED가 점멸하는 것을 느끼지 못할 것입니다. 오히려 LED가 일정하게 켜져 있다고 느낄 것입니다.

일반적으로 이러한 파형이 초당 43개 이상이 되면, 즉, 43Hz 이상의 주파수로 LED 점멸을 반복하면 우리는 그것을 느끼기 어렵습니다.

프로그램을 강제 종료하기 위해서는 CTRL 키를 누른 채로 c키를 눌러줍니다.

*** 파이썬의 경우 통역 방식의 언어이기 때문에 실제 실행 속도는 C 언어와 같은 번역 방식의 언어보다 많이 늦습니다. 그래서 이 예제의 경우 실제로는 100Hz의 속도를 내기 어려울 수 있습니다. 그래서 LED가 깜빡이는 현상이 발생할 수 있습니다.

07 LED 밝기 변경해보기

이제 Rpi.GPIO.output 함수와 time.sleep 함수를 이용하여 LED의 밝기를 변경해 보도록 합니다. 이전 예제의 경우 LED는 100Hz의 속도로 50%는 점등을, 50%는 소등을 반복하였습니다. 그리고 이 경우 우리는 LED의 밝기를 평균값인 50%의 밝기로 느꼈습니다. 만약 LED에 대해 10%는 점등을, 90%는 소등을 반복한다면 우리는 LED의 밝기를 어떻게 느낄까요? 평균 10%의 밝기로 느끼게 되지 않을까요? 예제를 통해 확인해 보도록 합니다.

LED 어둡게 해 보기

먼저 사각파형의 HIGH 구간을 10%로 해 LED를 어둡게 해 봅니다.

1. 다음과 같이 예제를 수정합니다.

_06_gpio_output_6.py

```
1  : import RPi.GPIO as GPIO
2  : import time
3  :
4  : led_pin =17
```

```
5  :
6  : GPIO.setmode(GPIO.BCM)
7  :
8  : GPIO.setup(led_pin, GPIO.OUT)
9  :
10 : try:
11 :   while True:
12 :             GPIO.output(led_pin, True)
13 :             time.sleep(0.001)
14 :             GPIO.output(led_pin, False)
15 :             time.sleep(0.009)
16 : except KeyboardInterrupt:
17 :   pass
18 :
19 : GPIO.cleanup()
```

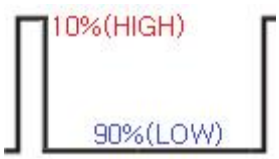
13 : 0.005를 0.001로 변경합니다.

15 : 0.005를 0.009로 변경합니다.

2. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _06_gpio_output_6.py
```

이 예제의 경우도 LED는 초당 100번 점멸 하게 됩니다. 즉, 100Hz의 주파수로 점멸하게 됩니다. 그러나 10%는 점등 상태로, 90%는 소등 상태로 있게 됩니다. 그래서 우리는 LED가 이전 예제에 비해 어둡다고 느끼게 됩니다.



그림에서 LED는 실제로 10%만 점등 상태이지만 100Hz의 주파수로 점멸하기 때문에 우리는 10%의 평균 밝기로 느끼게 됩니다. 10%는 True 값에 의해 켜져 있고 90%는 False 값에 의해 꺼져있으며, 이 경우 (HIGH:LOW)=(1:9)이 되게 됩니다. 즉, 상하비가 1:9가 됩니다.

프로그램을 강제 종료하기 위해서는 CTRL 키를 누른 채로 c키를 눌러줍니다.

LED 밝게 해 보기

다음은 사각파형의 HIGH 구간을 90%로 해 LED를 밝게 해 봅니다.

1. 다음과 같이 예제를 수정합니다.

_06_gpio_output_7.py

```
1  : import RPi.GPIO as GPIO
```

```
2  :  import time
3  :
4  :  led_pin =17
5  :
6  :  GPIO.setmode(GPIO.BCM)
7  :
8  :  GPIO.setup(led_pin, GPIO.OUT)
9  :
10 :  try:
11 :    while True:
12 :              GPIO.output(led_pin, True)
13 :              time.sleep(0.009)
14 :              GPIO.output(led_pin, False)
15 :              time.sleep(0.001)
16 :  except KeyboardInterrupt:
17 :    pass
18 :
19 :  GPIO.cleanup()
```

13 : 0.001을 0.009로 변경합니다.

15 : 0.009를 0.001로 변경합니다.

2. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _06_gpio_output_7.py
```

이 예제의 경우도 LED는 초당 100번 점멸 하게 됩니다. 즉, 100Hz의 주파수로 점멸하게 됩니다. 그러나 90%는 점등 상태로, 10%는 소등 상태로 있게 됩니다. 그래서 우리는 LED가 이전 예제에 비해 아주 밝다고 느끼게 됩니다.

그림에서 LED는 실제로 90%만 점등 상태이지만 100Hz의 주파수로 점멸하기 때문에 우리는 90%의 평균 밝기로 느끼게 됩니다. 90%는 HIGH 구간에 의해 켜져 있고 10%는 LOW 구간에 의해 꺼져있으며, 이 경우 (HIGH:LOW)=(9:1)이 되게 됩니다. 즉, 상하비가 9:1이 됩니다.

프로그램을 강제 종료하기 위해서는 CTRL 키를 누른 채로 c키를 눌러줍니다.

08 LED 밝기 조절해보기

여기서는 10밀리 초 간격으로 시작해서 1초 간격으로 다음의 상하비로 LED의 밝기를 조절해

보도록 합니다.

```
0:10, 1:9, 2:8, 3:7 ... 10:0
```

즉, HIGH 구간의 개수는 0부터 10까지 차례로 늘어나며, 반대로 LOW 구간의 개수는 10부터 0까지 차례로 줄게 됩니다.

1. 다음과 같이 예제를 수정합니다.

_06_gpio_output_8.py

```
1  : import RPi.GPIO as GPIO
2  : import time
3  :
4  : led_pin =17
5  :
6  : GPIO.setmode(GPIO.BCM)
7  :
8  : GPIO.setup(led_pin, GPIO.OUT)
9  :
10 : try:
11 :   while True:
12 :           for t_high in range(0,11):
13 :                   GPIO.output(led_pin, True)
14 :                   time.sleep(t_high*0.001)
15 :                   GPIO.output(led_pin, False)
16 :                   time.sleep((10-t_high)*0.001)
17 : except KeyboardInterrupt:
18 :   pass
19 :
20 : GPIO.cleanup()
```

12 : t_high 변수를 0이상 11 미만의 정수에 대해, 13~16줄을 수행합니다.

13,14 : LED를 켜고 0.001*t_high 초만큼 기다립니다.

15,16 : LED를 끄고 0.001*(10-t_high) 초만큼 기다립니다.

14,16 : 0.001*(t_high + (10 - t_high)) = 0.001*10 = 0.01초가 되어 for문을 한 번 도는 데는 10밀리 초 정도가 되며 for문 전체를 도는 데는 10밀리 초*11회=110밀리 초 정도가 됩니다.

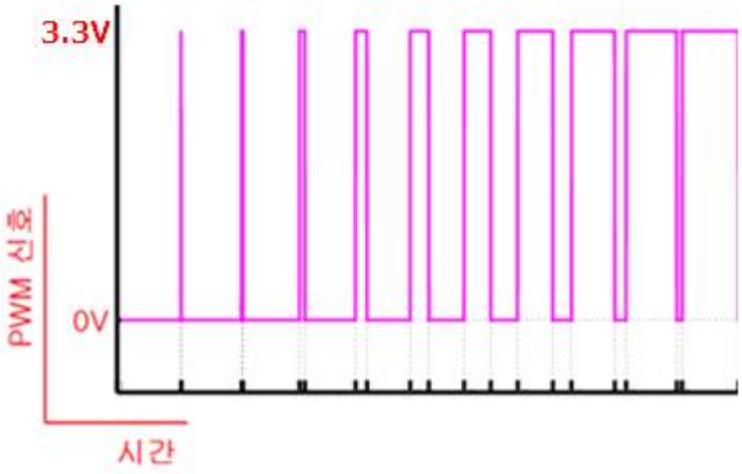
2. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _06_gpio_output_8.py
```

10밀리 초 간격으로 다음의 비율로 LED가 밝아집니다.

```
0%, 10% 20%, 30%, ... 100%
```

아래와 같은 형태의 파형으로 LED의 밝기가 변합니다.



이 예제의 경우 밝기의 변화가 너무 빨라 밝기가 변하는 것을 느끼기는 힘듭니다. 깜빡임으로 느낄 수도 있습니다. 밝기 변화 주기가 110밀리 초이며, 이는 초당 9번 정도의 횟수가 되기 때문에 느끼기 어려울 수 있습니다.

프로그램을 강제 종료하기 위해서는 CTRL 키를 누른 채로 c키를 눌러줍니다.

3. 다음과 같이 예제를 수정합니다.

_06_gpio_output_9.py

```
1  : import RPi.GPIO as GPIO
2  : import time
3  :
4  : led_pin =17
5  :
6  : GPIO.setmode(GPIO.BCM)
7  :
8  : GPIO.setup(led_pin, GPIO.OUT)
9  :
10 : try:
11 :   while True:
12 :           for t_high in range(0,11):
13 :                   cnt =0
14 :                   while True:
15 :                           GPIO.output(led_pin, True)
16 :                           time.sleep(t_high*0.001)
17 :                           GPIO.output(led_pin, False)
18 :                           time.sleep((10-t_high)*0.001)
19 :
20 :                           cnt +=1
21 :                           if cnt==10: break
22 : except KeyboardInterrupt:
23 :   pass
24 :
```

```
25  :  GPIO.cleanup()
```

13 : cnt 변수를 선언한 후, 0으로 초기화합니다.

14 : 계속해서 14~21줄을 수행합니다.

20 : cnt 값을 하나씩 증가시킵니다.

21 : cnt 값이 10이 되면 내부 while 문을 나옵니다.

이렇게 하면 14~21줄을 cnt값이 0에서 9까지 10회 반복하게 됩니다. 그러면 t_high 값을 유지하는 시간을 10밀리 초(0.01초)에서 100밀리 초(0.1초)로 늘릴 수 있습니다. for 문을 수행하는 시간도 110밀리 초(0.11초)에서 1100밀리 초(1.1초)로 늘릴 수 있으며, 우리는 LED 밝기의 변화를 느낄 수 있습니다.

4. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _06_gpio_output_9.py
```

1.1 초 주기로 LED의 밝기가 변하는 것을 느낄 수 있습니다.

프로그램을 강제 종료하기 위해서는 CTRL 키를 누른 채로 c키를 눌러줍니다.

5. 다음과 같이 예제를 수정합니다.

_06_gpio_output_10.py

```
1   :  import RPi.GPIO as GPIO
2   :  import time
3   :
4   :  led_pin =17
5   :
6   :  GPIO.setmode(GPIO.BCM)
7   :
8   :  GPIO.setup(led_pin, GPIO.OUT)
9   :
10  :  try:
11  :    while True:
12  :            for t_high in range(0,11):
13  :                    cnt =0
14  :                    while True:
15  :                            GPIO.output(led_pin, True)
16  :                            time.sleep(t_high*0.001)
17  :                            GPIO.output(led_pin, False)
18  :                            time.sleep((10-t_high)*0.001)
19  :
20  :                            cnt +=1
21  :                            if cnt==10: break
22  :            for t_high in range(10,-1,-1):
23  :                    cnt =0
24  :                    while True:
25  :                            GPIO.output(led_pin, True)
26  :                            time.sleep(t_high*0.001)
```

```
27 :                                GPIO.output(led_pin, False)
28 :                                time.sleep((10-t_high)*0.001)
29 :
30 :                                cnt +=1
31 :                                if cnt==10: break
32 : except KeyboardInterrupt:
33 :   pass
34 :
35 : GPIO.cleanup()
```

22 : t_high 변수를 10부터 -1초과까지 1씩 감소시켜가면서, 23~31줄을 수행합니다.

23~31 : 13~21줄의 내용과 같습니다.

이렇게 하면 첫 번째 for문에 의해서 LED가 1.1초간 밝아지며, 두 번째 for문에 의해서 LED가 1.1초간 어두워집니다.

6. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _06_gpio_output_10.py
```

LED가 1.1초간 밝아지고, 1.1초간 어두워지는 동작을 반복하는 것을 확인합니다.

프로그램을 강제 종료하기 위해서는 CTRL 키를 누른 채로 c키를 눌러줍니다.

03 Rpi.GPIO.PWM 모듈

이전 예제에서 우리는 100Hz의 속도로 0~10 개의 True 값으로 LED의 밝기를 조절해 보았습니다. Rpi.GPIO.PWM 모듈을 사용할 경우 빠른 주파수와 더 조밀한 상하비로 LED의 밝기를 조절할 수 있습니다. GPIO.PWM 모듈을 이용하여 상하비를 0.0~100.0% 단계로 조절할 수 있습니다.

Rpi.GPIO.PWM 모듈은 GPIO 핀에 소프트웨어적으로 아래와 같은 형태의 사각 파형을 내보낼 수 있습니다.

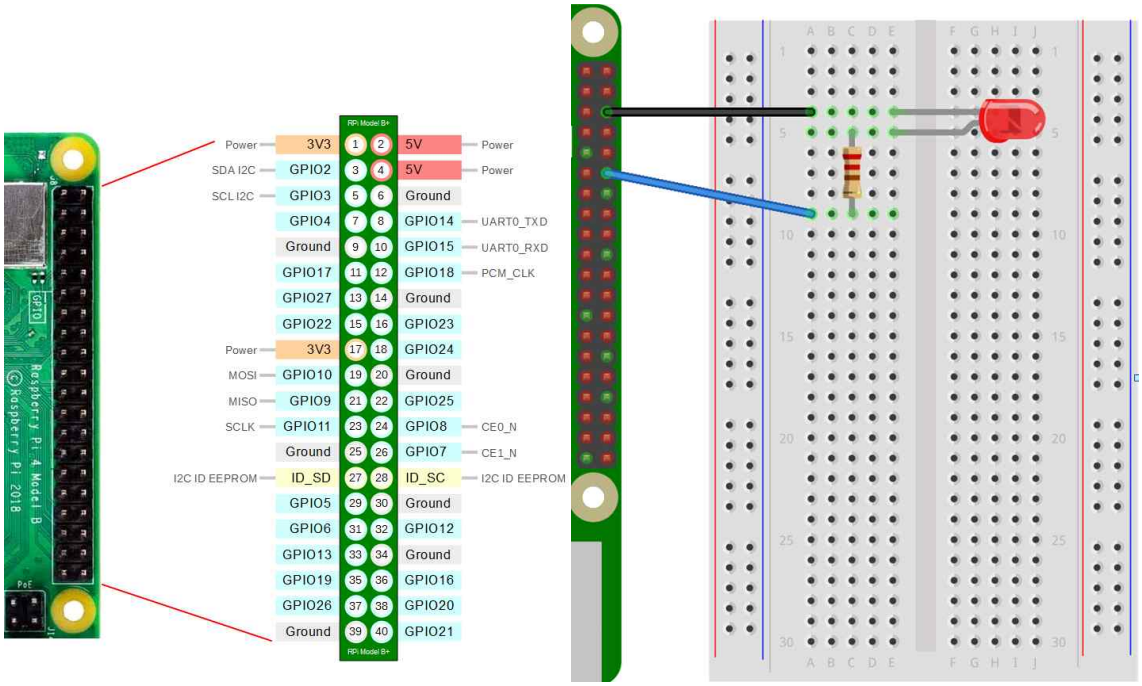
다음에 노랗게 표시된 GPIO 핀을 제어할 수 있습니다.

소프트웨어적이란 말은 CPU가 소프트웨어를 읽고 수행한다는 의미로 CPU가 직접 핀 제어를 통해 신호를 내 보낸다는 의미입니다. PWM 하드웨어가 직접 수행하는 것에 비해 주파수와 듀티비의 정밀도에 제약이 있습니다. 우리는 뒤에서 PCA9685라는 PWM 하드웨어 모듈을 이용하여 하드웨어적으로 사각파형을 생성해 봅니다.

여기서는 RPi.GPIO 모듈이 제공하는 PWM 클래스를 이용하여 PWM 객체를 생성한 후, LED의 밝기를 조절해봅니다. 또 부저를 통해 음악을 연주해 봅니다. 마지막으로 움직임을 만들어 내기 위해서 서보모터를 제어해 봅니다.

01 LED 회로 구성하기

핀 맵을 참조하여 다음과 같이 회로를 구성합니다.



LED의 긴 핀(+)을 220 Ohm 저항을 통해 라즈베리파이 보드의 GPIO 18번 핀에 연결합니다. LED의 짧은 핀(-)은 GND 핀에 연결합니다.

02 LED 점멸 반복해보기

먼저 Rpi.GPIO.PWM 클래스를 이용하여 LED 점멸을 반복해 봅니다.

1. 다음과 같이 예제를 작성합니다.

_07_pwm_output.py

```
1  : import RPi.GPIO as GPIO
2  :
3  : led_pin =18
4  :
5  : GPIO.setmode(GPIO.BCM)
6  :
7  : GPIO.setup(led_pin, GPIO.OUT)
8  :
9  : pwm = GPIO.PWM(led_pin, 1.0) # 1.0Hz
10 : pwm.start(50.0) # 0.0~100.0
11 :
12 : try:
13 :   while True:
14 :             pass
15 : except KeyboardInterrupt:
16 :   pass
17 :
18 : pwm.stop()
19 : GPIO.cleanup()
```

1 : RPi.GPIO 모듈을 GPIO로 불러옵니다. RPi.GPIO 모듈은 5,7,19줄에 있는 setmode, setup,cleanup 함수들과 9줄에 있는 PWM 클래스를 가지고 있습니다.
3 : led_pin 변수를 선언한 후, 18로 초기화합니다. 여기서 18은 BCM GPIO 핀 번호를 나타냅니다.
5 : GPIO.setmode 함수를 호출하여 BCM GPIO 핀 번호를 사용하도록 설정합니다.
7 : GPIO.setup 함수를 호출하여 led_pin을 GPIO 출력으로 설정합니다.
9 : GPIO.PWM 객체를 하나 생성한 후, pwm 변수가 가리키도록 합니다. GPIO.PWM 객체 생성 시, 첫 번째 인자는 핀 번호가 되며, 두 번째 인자는 주파수 값이 됩니다. 주파수 값은 0.0보다 큰 실수 값입니다. 예제에서는 1.0을 주고 있으며, 이 경우 1.0Hz의 주파수가 led_pin에 생성됩니다.
10 : pwm 객체의 start 함수를 호출하여 PWM 파형을 내보내기 시작합니다. start 함수의 인자는 0.0~100.0 사이의 실수 값으로 사각파형의 HIGH 구간의 비율을 나타냅니다. 여기서는 PWM 파형의 HIGH 구간을 50.0%로 설정하고 있습니다.
13,14 : 빈 while 문을 수행하여 LED 핀으로 나가는 PWM 파형이 유지되도록 합니다. while 문을 끝내기 위해서는 CTRL+c 키를 누릅니다.
18 : pwm 객체에 대해 stop 함수를 호출하여 PWM 파형 출력을 멈춥니다.
19 : GPIO.cleanup 함수를 호출하여 GPIO 핀의 상태를 초기화해 줍니다.

2. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _07_pwm_output.py
```

1초 주기로 LED가 점멸 하는 것을 확인합니다. 즉, 1Hz의 주파수로 LED의 점멸을 확인합니다.

GPIO.PWM 도움말 보기

GPIO.PWM 모듈에 대해 파이썬 쉘을 이용하여 살펴봅니다.

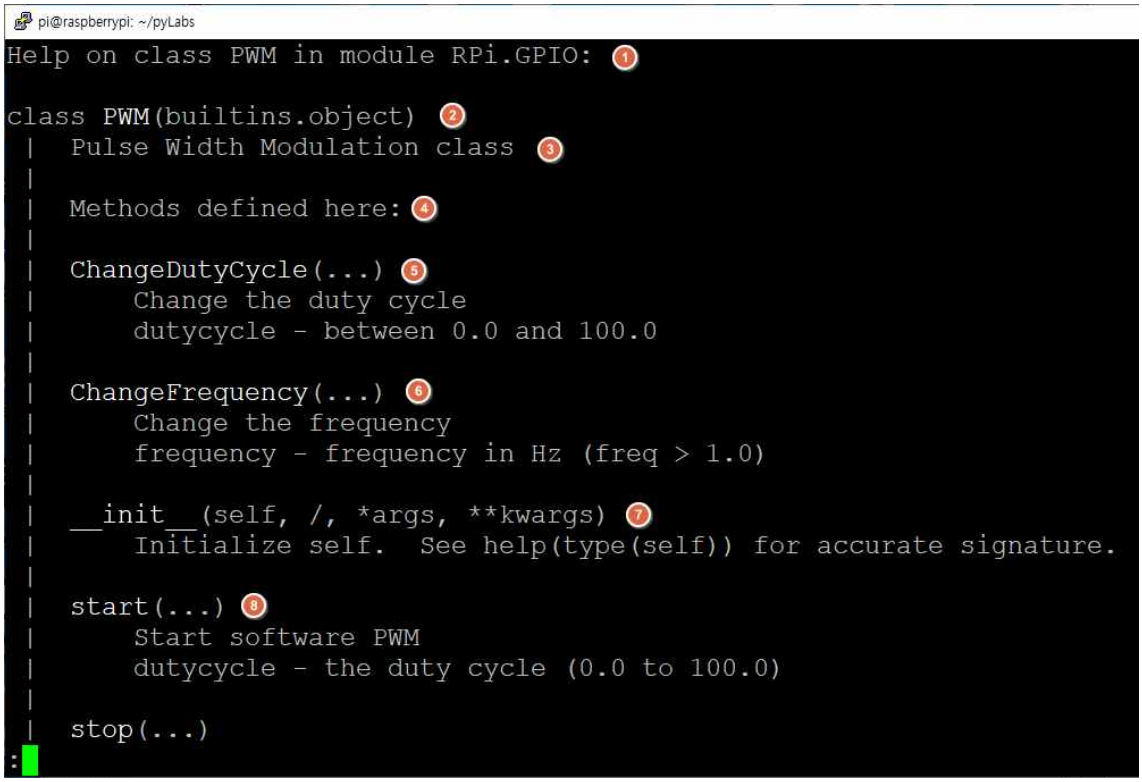
1. 다음과 같이 차례대로 명령을 실행합니다.

```
pi@raspberrypi:~/pyLabs $ python3 ❶
Python 3.7.3 (default, Dec 20 2019, 18:57:59)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import RPi.GPIO as GPIO ❷
>>> help(GPIO.PWM) ❸
```

❶ python3 명령을 주어 파이썬 쉘을 실행시킵니다. ❷ RPi.GPIO 모듈을 GPIO라는 이름으로

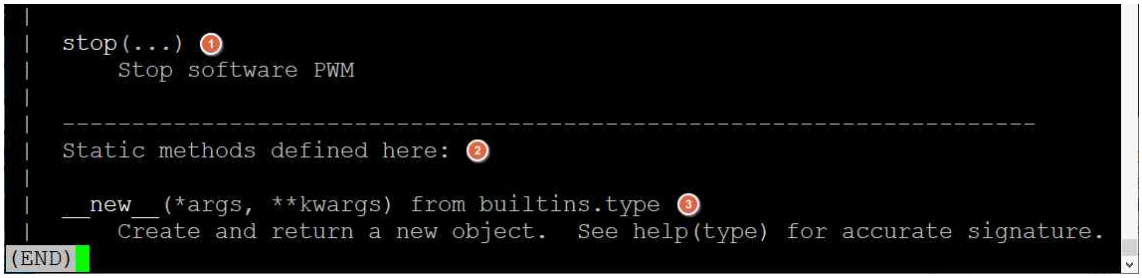
불러옵니다. 모듈이 가진 함수나 변수 등의 설명글을 보기 위해서 먼저 import 해야 합니다.
❸ help(GPIO.PWM) 명령을 이용해 GPIO.PWM 모듈을 살펴봅니다. 살펴보고자 하는 함수, 변수, 모듈 등을 help 명령의 인자로 주면 됩니다.

2. 다음과 같이 PWM에 대한 설명글이 나옵니다.



❶ RPi._GPIO 모듈 안에 있는 PWM 클래스에 대한 설명글입니다.
❷ PWM 클래스를 나타냅니다. 최상위 내장 클래스인 builtins.object 클래스를 상속합니다.
❸ 파형 폭 조절 클래스입니다. 클래스의 기능을 설명하는 부분입니다.
❹ 클래스 함수인 메쏘드 정의 시작 부분입니다. 메쏘드는 방법이라는 뜻으로 클래스 함수를 나타냅니다.
❺ ChangeDutyCycle 함수에 대한 설명 부분입니다. 듀티 사이클을 변경합니다. 듀티 사이클은 사각 파형의 HIGH 구간을 나타냅니다. dutycycle 값은 0.0~100.0 사이값을 줄 수 있습니다.
❻ ChangeFrequency 함수에 대한 설명 부분입니다. 주파수를 변경합니다. 주파수는 Hz 단위이며 1.0보다 커야 합니다.
❼ PWM 객체 생성시 객체 초기화함수입니다. 생성자 함수라고도 합니다.
❽ start 함수에 대한 설명 부분입니다. 소프트웨어 PWM을 시작합니다. dutycycle 값은 0.0~100.0 사이값을 줄 수 있습니다.

스페이스 키를 쳐봅니다. 다음 화면으로 넘어갑니다.



❶ stop 함수에 대한 설명 부분입니다. 소프트웨어 PWM을 멈춥니다.
❷ 클래스 정적 함수인 메쏘드 정의 시작 부분입니다. 메쏘드는 방법이라는 뜻으로 클래스 함수를 나타냅니다.
❸ 새로운 객체를 생성하고 내어주는 객체 생성함수입니다.

종료는 q 문자를 입력합니다. 내용이 많을 때는 pgup, pgdn 키나 방향키 등을 이용하여 이동하면서 봅니다.

3. 파이썬 쉘을 빠져 나올 때는 다음과 같이 quit() 함수를 수행해 줍니다.

03 LED 점멸 간격 줄여보기

이제 LED의 점멸 간격을 줄여보도록 합니다. 그러면 여러분은 좀 더 조밀하게 LED가 점멸하는 것을 느낄 것입니다.

1. 예제를 다음과 같이 수정합니다.

_07_pwm_output_2.py

```
1  : import RPi.GPIO as GPIO
2  :
3  : led_pin =18
4  :
5  : GPIO.setmode(GPIO.BCM)
6  :
7  : GPIO.setup(led_pin, GPIO.OUT)
8  :
9  : pwm = GPIO.PWM(led_pin, 10.0)
10 : pwm.start(50.0) # 0.0~100.0
11 :
12 : try:
13 :   while True:
14 :             pass
15 : except KeyboardInterrupt:
16 :   pass
17 :
18 : pwm.stop()
```

```
19  :  GPIO.cleanup()
```

9 : GPIO.PWM 객체 생성 부분에서 두 번째 인자를 10.0으로 변경합니다. 이 경우 10.0Hz의 주파수가 led_pin에 생성됩니다.

2. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _07_pwm_output_2.py
```

이 예제의 경우 LED는 초당 10번 점멸 하게 됩니다. 즉, 10Hz의 주파수로 점멸하게 됩니다.

04 LED 점멸을 밝기로 느껴보기

LED의 점멸 간격을 더 줄여보도록 합니다. 여기서 여러분은 LED의 점멸을 느끼지 못하게 될 것입니다. 오히려 LED가 일정하게 켜져 있다고 느낄 것입니다.

1. 예제를 다음과 같이 수정합니다.

_07_pwm_output_3.py

```
1   :  import RPi.GPIO as GPIO
2   :
3   :  led_pin =18
4   :
5   :  GPIO.setmode(GPIO.BCM)
6   :
7   :  GPIO.setup(led_pin, GPIO.OUT)
8   :
9   :  pwm = GPIO.PWM(led_pin, 100.0)
10  :  pwm.start(50.0) # 0.0~100.0
11  :
12  :  try:
13  :    while True:
14  :              pass
15  :  except KeyboardInterrupt:
16  :    pass
17  :
18  :  pwm.stop()
19  :  GPIO.cleanup()
```

9 : GPIO.PWM 객체 생성 부분에서 두 번째 인자를 100.0으로 변경합니다. 이 경우 100.0Hz의 주파수가 led_pin에 생성됩니다.

2. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _07_pwm_output_3.py
```

이 예제의 경우 LED는 초당 100번 점멸하게 됩니다. 즉, 100Hz의 주파수로 점멸하게 됩니다.

그림과 같은 파형이 초당 100개가 생성됩니다. 이제 여러분은 LED가 점멸하는 것을 느끼지 못할 것입니다. 오히려 LED가 일정한 밝기로 켜져 있다고 느낄 것입니다.

05 LED 밝기 1000단계 조절해 보기

주파수를 늘리면 LED의 점멸이 더 부드러워집니다. 여기서는 주파수를 늘여 LED 점멸을 좀 더 부드럽게 만들어 봅니다.

1. 예제를 다음과 같이 수정합니다.

_07_pwm_output_4.py

```
1  : import RPi.GPIO as GPIO
2  :
3  : led_pin =18
4  :
5  : GPIO.setmode(GPIO.BCM)
6  :
7  : GPIO.setup(led_pin, GPIO.OUT)
8  :
9  : pwm = GPIO.PWM(led_pin, 1000.0)
10 : pwm.start(50.0) # 0.0~100.0
11 :
12 : try:
13 :   while True:
14 :             pass
15 : except KeyboardInterrupt:
16 :   pass
17 :
18 : pwm.stop()
19 : GPIO.cleanup()
```

9 : GPIO.PWM 클래스 객체 생성 부분에서 두 번째 인자를 1000.0으로 변경합니다. 이 경우 1000.0Hz의 주파수가 led_pin에 생성됩니다.

2. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _07_pwm_output_4.py
```

이 예제의 경우 LED는 초당 1000번 점멸하게 됩니다. 즉, 1000Hz의 주파수로 점멸하게 됩니다. 이전 예제와 마찬가지로 LED가 일정한 밝기로 켜져 있다고 느낄 것입니다.

3. 예제를 다음과 같이 수정합니다.

_07_pwm_output_5.py

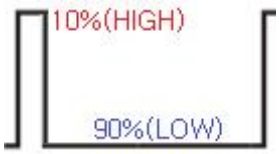
```
1  : import RPi.GPIO as GPIO
2  :
3  : led_pin =18
4  :
5  : GPIO.setmode(GPIO.BCM)
6  :
7  : GPIO.setup(led_pin, GPIO.OUT)
8  :
9  : pwm = GPIO.PWM(led_pin, 1000.0)
10 : pwm.start(10.0) # 0.0~100.0
11 :
12 : try:
13 :   while True:
14 :           pass
15 : except KeyboardInterrupt:
16 :   pass
17 :
18 : pwm.stop()
19 : GPIO.cleanup()
```

10 : pwm 객체의 start 함수의 인자를 10.0으로 변경해 줍니다. 이렇게 하면 PWM 파형의 HIGH 구간이 10.0%로 설정됩니다.

4. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _07_pwm_output_5.py
```

이 예제의 경우도 LED는 초당 1000번 점멸 하게 됩니다. 즉, 1000Hz의 주파수로 점멸하게 됩니다. 그러나 10%는 점등 상태로, 90%는 소등 상태로 있게 됩니다. 그래서 우리는 LED가 이전 예제에 비해 어둡다고 느끼게 됩니다.



그림에서 LED는 실제로 10%만 점등 상태이지만 1000Hz의 주파수로 점멸하기 때문에 우리는 10%의 평균 밝기로 느끼게 됩니다.

프로그램을 강제 종료하기 위해서는 CTRL 키를 누른 채로 c키를 눌러줍니다.

5. 예제를 다음과 같이 수정합니다.

_07_pwm_output_6.py

```
1  : import RPi.GPIO as GPIO
2  :
3  : led_pin =18
4  :
5  : GPIO.setmode(GPIO.BCM)
6  :
7  : GPIO.setup(led_pin, GPIO.OUT)
8  :
9  : pwm = GPIO.PWM(led_pin, 1000.0)
10 : pwm.start(90.0) # 0.0~100.0
11 :
12 : try:
13 :   while True:
14 :           pass
15 : except KeyboardInterrupt:
16 :   pass
17 :
18 : pwm.stop()
19 : GPIO.cleanup()
```

10 : pwm 객체의 start 함수의 인자를 90.0으로 변경해 줍니다. 이렇게 하면 PWM 파형의 HIGH 구간이 90.0%로 설정됩니다.

6. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _07_pwm_output_6.py
```

이 예제의 경우도 LED는 초당 1000번 점멸 하게 됩니다. 즉, 1000Hz의 주파수로 점멸하게 됩니다. 그러나 90%는 점등 상태로, 10%는 소등 상태로 있게 됩니다. 그래서 우리는 LED가 이전 예제에 비해 아주 밝다고 느끼게 됩니다.

그림에서 LED는 실제로 90%만 점등 상태이지만 100Hz의 주파수로 점멸하기 때문에 우리는 90%의 평균 밝기로 느끼게 됩니다. 90%는 HIGH 구간에 의해 켜져 있고 10%는 LOW 구간에 의해 꺼져있으며, 이 경우 (HIGH:LOW)=(9:1)이 되게 됩니다. 즉, 상하비가 9:1이 됩니다.

프로그램을 강제 종료하기 위해서는 CTRL 키를 누른 채로 c키를 눌러줍니다.

7. 예제를 다음과 같이 수정합니다.

_07_pwm_output_7.py

```
1  : import RPi.GPIO as GPIO
2  : import time
3  :
4  : led_pin =18
5  :
6  : GPIO.setmode(GPIO.BCM)
7  :
8  : GPIO.setup(led_pin, GPIO.OUT)
9  :
10 : pwm = GPIO.PWM(led_pin, 1000.0) # 1.0Hz
11 : pwm.start(0.0) # 0.0~100.0
12 :
13 : try:
14 :   while True:
15 :           for t_high in range(0,101):
16 :                   pwm.ChangeDutyCycle(t_high)
17 :                   time.sleep(0.01)
18 : except KeyboardInterrupt:
19 :   pass
20 :
21 : pwm.stop()
22 : GPIO.cleanup()
```

15 : t_high 변수를 0부터 101미만의 정수에 대해, 16,17줄을 수행합니다.

16 : pwm 객체의 ChangeDutyCycle 함수를 호출하여 PWM 파형의 상하비를 변경해 줍니다. ChangeDutyCycle 함수는 PWM 파형의 상하비를 변경하는 함수로 인자로 0.0~100.0 사이의 실수값을 줄 수 있습니다. 11 번째 줄에서 start 함수를 호출하여 PWM 파형의 초기 상하비를 설정한 이후에, 상하비를 변경하고자 할 경우엔 ChangeDutyCycle 함수를 사용합니다.

17 : time.sleep 함수를 호출하여 0.01초만큼 기다립니다.

8. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _07_pwm_output_7.py
```

0.01초(=10밀리 초) 간격으로 다음의 비율로 LED가 밝아집니다.

```
0%, 1% 2%, 3%, ..., 97%, 98%, 99%, 100%
```

프로그램을 강제 종료하기 위해서는 CTRL 키를 누른 채로 c키를 눌러줍니다.

9. 예제를 다음과 같이 수정합니다.

_07_pwm_output_8.py

```
1  : import RPi.GPIO as GPIO
2  : import time
3  :
4  : led_pin =18
```

```
5  :
6  : GPIO.setmode(GPIO.BCM)
7  :
8  : GPIO.setup(led_pin, GPIO.OUT)
9  :
10 : pwm = GPIO.PWM(led_pin, 1000.0) # 1.0Hz
11 : pwm.start(0.0) # 0.0~100.0
12 :
13 : try:
14 :   while True:
15 :           for t_high in range(0,101):
16 :                   pwm.ChangeDutyCycle(t_high)
17 :                   time.sleep(0.01)
18 :           for t_high in range(100,-1,-1):
19 :                   pwm.ChangeDutyCycle(t_high)
20 :                   time.sleep(0.01)
21 : except KeyboardInterrupt:
22 :   pass
23 :
24 : pwm.stop()
25 : GPIO.cleanup()
```

18 : t_high 변수를 100부터 -1초과까지 1씩 감소시켜가면서, 19,20줄의 동작을 수행합니다.

10. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _07_pwm_output_8.py
```

약 1초간 0~100 단계로 LED의 밝기가 증가하고 약 1초간 100~0 단계로 LED의 밝기가 감소하는 동작을 반복하는 것을 볼 수 있습니다.

프로그램을 강제 종료하기 위해서는 CTRL 키를 누른 채로 c키를 눌러줍니다.

06 부저 살펴보기

본 책에서 사용할 부저는 다음과 같은 모양입니다.

피에조 부저는 극성을 가집니다.

소리와 주파수 이해하기

다음은 소리에 따른 주파수 표를 나타냅니다.

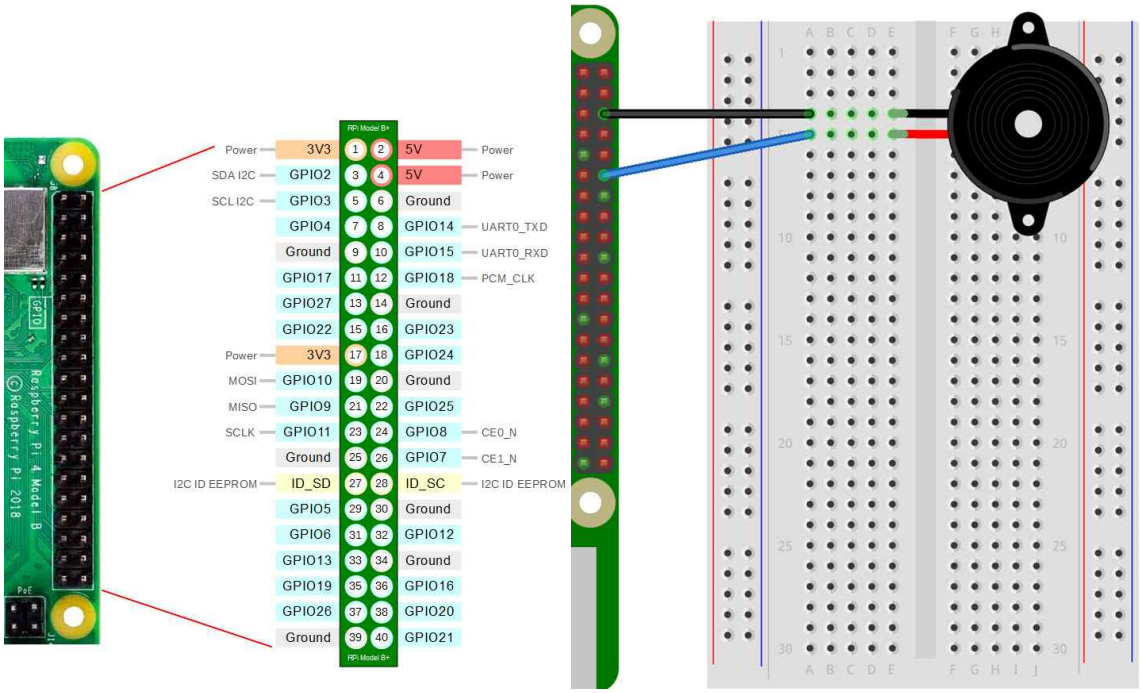
옥타브 및 음계별 표준 주파수 (단위 : Hz)

옥타브 / 음계	1	2	3	4	5	6	7	8
C(도)	32.7032	65.4064	130.8128	261.6256	523.2511	1046.502	2093.005	4186.009
C#	34.6478	69.2957	138.5913	277.1826	554.3653	1108.731	2217.461	4434.922
D(레)	36.7081	73.4162	146.8324	293.6648	587.3295	1174.659	2349.318	4698.636
D#	38.8909	77.7817	155.5635	311.1270	622.2540	1244.508	2489.016	4978.032
E(미)	41.2034	82.4069	164.8138	329.6276	659.2551	1318.510	2637.020	5274.041
F(파)	43.6535	87.3071	174.6141	349.2282	698.4565	1396.913	2793.826	5587.652
F#	46.2493	92.4986	184.9972	369.9944	739.9888	1479.978	2959.955	5919.911
G(솔)	48.9994	97.9989	195.9977	391.9954	783.9909	1567.982	3135.963	6271.927
G#	51.9130	103.8262	207.6523	415.3047	830.6094	1661.219	3322.438	6644.875
A(라)	55.0000	110.0000	220.0000	440.0000	880.0000	1760.000	3520.000	7040.000
A#	58.2705	116.5409	233.0819	466.1638	932.3275	1864.655	3729.310	7458.620
B(시)	61.7354	123.4708	246.9417	493.8833	987.7666	1975.533	3951.066	7902.133

예를 들어 4 옥타브에서 도 음에 대한 주파수는 262 Hz가 됩니다. 즉, 1초에 262 개의 사각 파형을 만들어 내면 도 음이 나게 됩니다. 레는 294 Hz, 미는 330 Hz, 파는 349 Hz, 솔은 392 Hz, 라는 440 Hz, 시는 494 Hz, 5 옥타브의 도는 523 Hz가 됩니다.

07 부저 회로 구성하기

핀 맵을 참조하여 다음과 같이 라즈베리파이와 연결합니다.



부저의 +핀을 라즈베리파이 보드의 GPIO18번 핀에 연결합니다. 부저의 다른 핀은 GND 핀에 연결합니다.

08 부저 소리내보기

여기서는 부저를 이용하여 도음과 레음을 내보겠습니다.

1. 다음과 같이 예제를 작성합니다.

_08_pwm_buzzer.py

```
1  : import RPi.GPIO as GPIO
2  : import time
3  :
4  : buzzer_pin =18
5  :
6  : GPIO.setmode(GPIO.BCM)
7  :
8  : GPIO.setup(buzzer_pin, GPIO.OUT)
9  :
10 : pwm = GPIO.PWM(buzzer_pin, 262)
11 : pwm.start(50.0)
12 :
13 : time.sleep(2.0)
14 : pwm.ChangeDutyCycle(0.0)
15 :
16 : pwm.stop()
17 : GPIO.cleanup()
```

4 : buzzer_pin 변수를 선언하고, 18번 핀으로 초기화합니다.

10 : GPIO.PWM 객체를 생성하면서 buzzer_pin으로 나갈 PWM 파형의 주파수를 262Hz로 설정합니다. 이렇게 하면 262Hz의 주파수가 생성되며 도 음을 낼 수 있습니다.

11 : pwm 객체에 대해 start 함수를 호출하여 PWM 구동을 시작합니다. 인자로 50.0을 줍니다. 이렇게 하면 사각 파형이 50%의 HIGH 구간을 갖습니다.

13 : 2.0초 동안 기다립니다.

14 : pwm 객체에 대해 ChangeDutyCycle 함수를 호출하여 사각 파형의 HIGH 구간을 0%로 변경합니다. 이렇게 하면 소리가 꺼지게 됩니다.

2. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _08_pwm_buzzer.py
```

부저에서 2초간 나는 도 음을 확인합니다.

3. 다음과 같이 예제를 수정합니다.

_08_pwm_buzzer_2.py

```
1  : import RPi.GPIO as GPIO
2  : import time
3  :
4  : buzzer_pin =18
5  :
6  : GPIO.setmode(GPIO.BCM)
7  :
8  : GPIO.setup(buzzer_pin, GPIO.OUT)
9  :
10 : pwm = GPIO.PWM(buzzer_pin, 1.0)
11 : pwm.start(50.0)
12 :
13 : for cnt in range(0,3):
14 :   pwm.ChangeFrequency(262)
15 :   time.sleep(0.5)
16 :   pwm.ChangeFrequency(294)
17 :   time.sleep(0.5)
18 :
19 : pwm.ChangeDutyCycle(0.0)
20 :
21 : pwm.stop()
22 : GPIO.cleanup()
```

10 : GPIO.PWM 객체를 생성하면서 buzzer_pin으로 나갈 PWM 파형의 주파수를 1.0Hz로 설정합니다. 주파수 값은 0.0 보다 커야 합니다.

11 : pwm 객체에 대해 start 함수를 호출 PWM 구동을 시작합니다. 인자로 50.0을 줍니다.

13 : cnt 변수 값을 0부터 3 미만의 정수에 대해 14~17줄의 동작을 3회 반복합니다.

14 : pwm 객체에 대해 ChangeFrequency 함수를 호출하여 PWM 파형의 주파수를 262로 변경해 줍니다. 262는 4옥타브 도 음의 주파수입니다. ChangeFrequency 함수는 PWM 파형

의 주파수를 변경하는 함수입니다. 10 번째 줄에서 PWM 객체를 생성하여 PWM 파형의 초기 주파수를 설정한 이후에, 주파수를 변경하고자 할 경우엔 ChangeFrequency 함수를 사용합니다.
15 : time.sleep 함수를 호출하여 0.5초간 기다립니다.
16,17 : 레 음을 1초간 냅니다. 294는 4옥타브 레 음의 주파수입니다.

4. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _08_pwm_buzzer_2.py
```

도 음과 레 음이 0.5초 간격으로 3회 반복되는 것을 확인합니다.

09 부저 멜로디 연주하기

여기서는 부저를 이용하여 멜로디를 생성해 보도록 하겠습니다.

1. 다음과 같이 예제를 수정합니다.

_08_pwm_buzzer_3.py

```
1  : import RPi.GPIO as GPIO
2  : import time
3  :
4  : buzzer_pin =18
5  :
6  : GPIO.setmode(GPIO.BCM)
7  :
8  : GPIO.setup(buzzer_pin, GPIO.OUT)
9  :
10 : pwm = GPIO.PWM(buzzer_pin, 1.0)
11 : pwm.start(50.0)
12 :
13 : melody = [262,294,330,349,392,440,494,523]
14 :
15 : for note in range(0,8):
16 :     pwm.ChangeFrequency(melody[note])
17 :     time.sleep(0.5)
18 :
19 : pwm.ChangeDutyCycle(0.0)
20 :
21 : pwm.stop()
22 : GPIO.cleanup()
```

13 : 4 옥타브의 도, 레, 미, 파, 솔, 라, 시와 5 옥타브의 도에 해당하는 주파수를 값으로 갖는 목록 객체를 만든 후, melody 변수를 생성하여 가리키도록 합니다.
15 : note 변수 값을 0부터 8 미만의 정수에 대해 16,17줄을 수행합니다.
16 : pwm 객체에 대해 ChangeFrequency 함수를 호출하여 melody[note]주파수로 설정하고 있습니다.
17 : 0.5초간 기다립니다.

19 : pwm 객체에 대해 ChangeDutyCycle 함수를 호출하여 사각 파형의 HIGH 구간을 0.0%로 설정합니다. 이렇게 하면 부저 음이 나지 않게 됩니다.

2. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _08_pwm_buzzer_3.py
```

0.5초 간격으로 도, 레, 미, 파, 솔, 라, 시, 도 음이 연주되는 것을 확인합니다.

3. 다음과 같이 예제를 수정합니다.

_08_pwm_buzzer_4.py

```
1  : import RPi.GPIO as GPIO
2  : import time
3  :
4  : buzzer_pin =18
5  :
6  : GPIO.setmode(GPIO.BCM)
7  :
8  : GPIO.setup(buzzer_pin, GPIO.OUT)
9  :
10 : pwm = GPIO.PWM(buzzer_pin, 1.0)
11 : pwm.start(50.0)
12 :
13 : melody = [262,294,330,349,392,440,494,523]
14 :
15 : for note in range(0,8):
16 :     pwm.ChangeFrequency(melody[note])
17 :     time.sleep(0.5)
18 :
19 : for note in range(7,-1,-1):
20 :     pwm.ChangeFrequency(melody[note])
21 :     time.sleep(0.5)
22 :
23 : pwm.ChangeDutyCycle(0.0)
24 :
25 : pwm.stop()
26 : GPIO.cleanup()
```

21 : note 변수 값을 7부터 -1초과까지 1씩 감소시켜가면서 20,21줄을 수행합니다.

2. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _08_pwm_buzzer_4.py
```

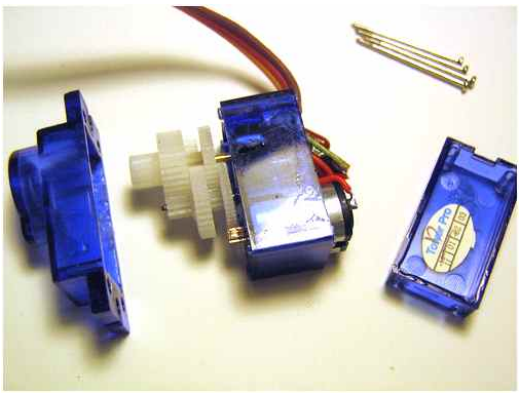
0.5초 간격으로 도, 레, 미, 파, 솔, 라, 시, 도, 도, 시, 라, 솔, 파, 미, 레, 도 음이 연주되는 것을 확인합니다.

10 서보 모터 살펴보기

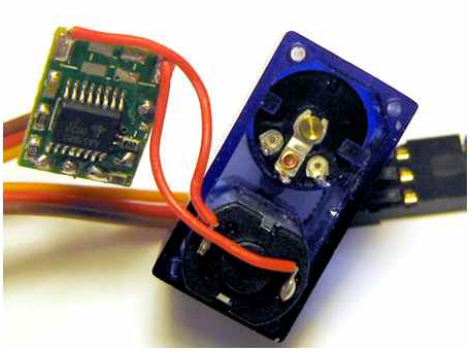
본 책에서 사용할 서보 모터는 다음과 같습니다.

일반적으로 서보 모터는 0~180도 범위에서 움직입니다.

다음은 서보 모터를 분해한 모습입니다.



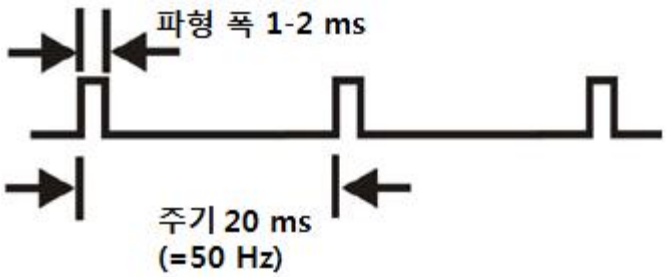
서보모터는 크게 DC 모터, 기어 시스템, 가변 저항, 제어 기판으로 구성됩니다.
다음은 아래쪽에서 살펴본 부분입니다.



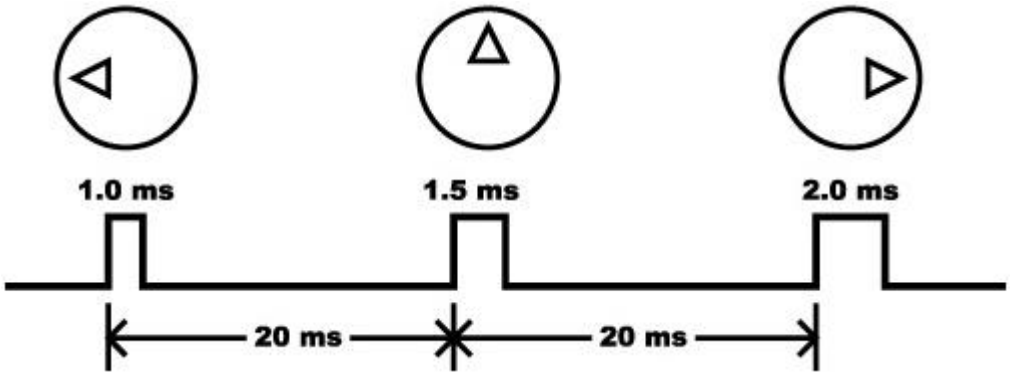
제어기판, DC 모터, 가변 저항을 볼 수 있습니다.

서보 모터 파형 이해하기

서보 모터 파형의 주기는 일반적으로 20 ms이며, 주파수는 50Hz입니다. 입력 파형의 HIGH 구간은 1~2ms 사이가 되어야 합니다.



서보 모터는 입력 파형의 HIGH 구간의 폭에 따라 움직이는 각도가 달라집니다.

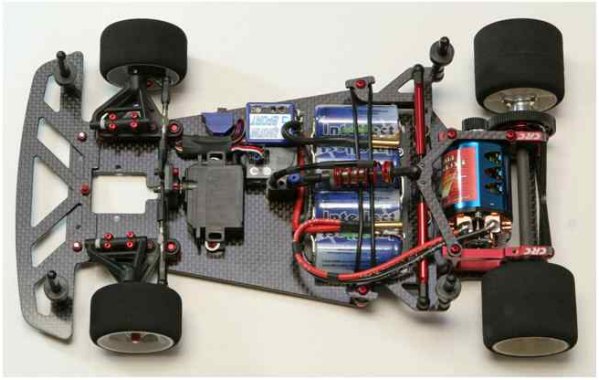


입력 파형의 HIGH 구간이 1.0 밀리 초일 경우엔 0 도, 2.0 밀리 초일 경우엔 180 도가 되며, 나머지 각도는 1.0 밀리 초와 2.0 밀리 초 사이에서 비례적으로 결정됩니다.

*** 이 책에서 사용하는 SG90 서보모터의 경우 0.6 밀리 초와 2.5 밀리 초의 True 값을 주어야 0도에서 180도 범위를 움직입니다.

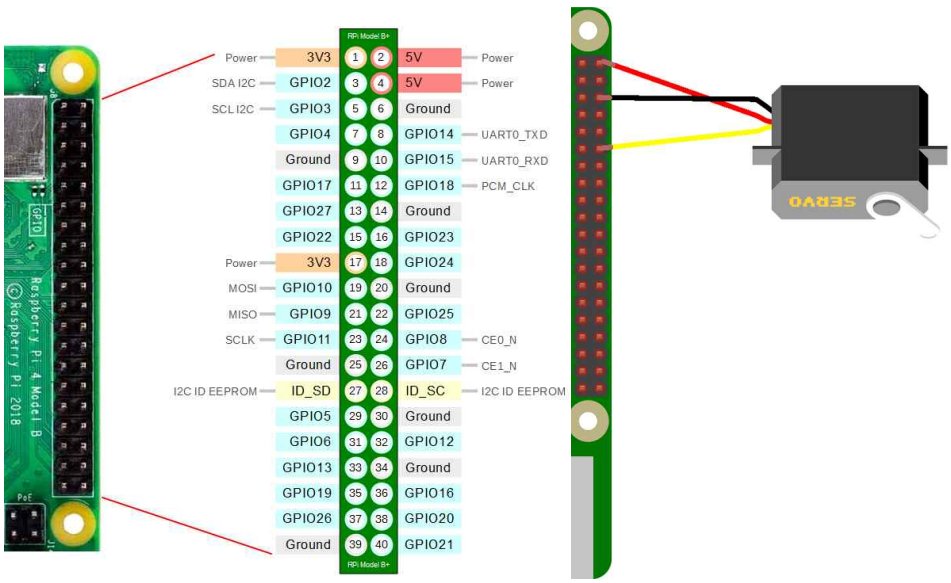
서보 모터는 다음과 같이 로봇의 관절을 구현하기 위해 사용할 수 있습니다.

또 RC 카의 방향을 조절하는 데에도 이용합니다.



11 서보 모터 회로 구성하기

다음 핀 맵을 참조하여 라즈베리파이 보드와는 다음과 같이 연결합니다.



서보 모터의 노란색 전선을 라즈베리파이 보드의 GPIO18번 핀에 연결합니다. 서보 모터의 검은색 또는 갈색 전선을 라즈베리파이 보드의 GND 핀에 연결합니다. 서보 모터의 빨간색 전선을 라즈베리파이 보드의 5V 핀에 연결합니다. 3.3V에 연결할 경우 서보 모터의 동력원이 약하며, 모터 회전 시 라즈베리파이가 리셋 되어 재부팅될 수도 있습니다. 서보 모터를 이용하여 프로젝트를 수행할 때에는 서보 모터의 동력원을 외부 배터리로 해야 합니다. 서보 모터는 내부적으로 회로가 구성되어 있으므로 외부에 추가 회로를 붙이지 않아도 됩니다.

12 서보 모터 각도 조절해보기

여기서는 서보모터의 각도를 0도, 90도로 조절해봅니다.

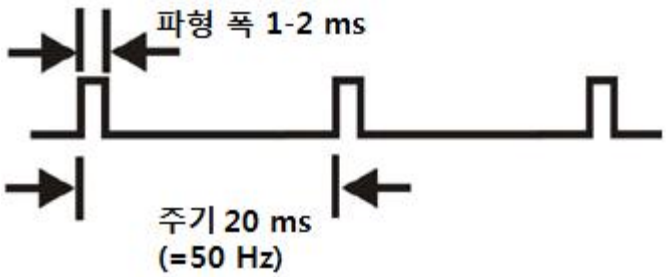
1. 다음과 같이 예제를 작성합니다.

_09_pwm_servo.py

```
1   :  import RPi.GPIO as GPIO
2   :  import time
3   :
4   :  servo_pin =18
5   :
6   :  GPIO.setmode(GPIO.BCM)
7   :
8   :  GPIO.setup(servo_pin, GPIO.OUT)
9   :
10  :  pwm = GPIO.PWM(servo_pin, 50) # 50Hz
11  :  pwm.start(3.0) #0.6ms
12  :
13  :  time.sleep(2.0)
14  :  pwm.ChangeDutyCycle(0.0)
15  :
16  :  pwm.stop()
17  :  GPIO.cleanup()
```

4 : 정수 객체 18을 생성한 후, sevro_pin 변수를 선언하여 가리키게 합니다.

10 : GPIO.PWM 객체를 생성하면서 servo_pin으로 나갈 PWM 파형의 주파수를 50Hz로 설정합니다. 50Hz는 서보 제어를 위해 필요한 주파수입니다.



11 : pwm 객체에 대해 start 함수를 호출하여 PWM 구동을 시작합니다. 인자로 3.0을 줍니다. 이렇게 하면 servo_pin으로 0.6 밀리초 동안 HIGH 값이 나가며, 서보 모터는 0도로 회전합니다. 인자가 3일 경우 3%를 의미하며 20 밀리초*(3/100) = 0.6 밀리초가 됩니다.

13 : 2초 동안 기다립니다.

14 : pwm 객체에 대해 ChangeDutyCycle 함수를 호출하여 사각 파형의 HIGH 구간을 0.0%로 설정합니다. 이렇게 하면 서보의 동작이 멈춥니다.

2. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _09_pwm_servo.py
```

서보가 0도 각도로 회전하는 것을 확인합니다.

3. 다음과 같이 예제를 수정합니다.

_09_pwm_servo_2.py

```
1  : import RPi.GPIO as GPIO
2  : import time
3  :
4  : servo_pin =18
5  :
6  : GPIO.setmode(GPIO.BCM)
7  :
8  : GPIO.setup(servo_pin, GPIO.OUT)
9  :
10 : pwm = GPIO.PWM(servo_pin, 50)
11 : pwm.start(3.0)
12 :
13 : for cnt in range(0,3):
14 :   pwm.ChangeDutyCycle(3.0)
15 :   time.sleep(1.0)
16 :   pwm.ChangeDutyCycle(12.5)
17 :   time.sleep(1.0)
18 :
19 : pwm.ChangeDutyCycle(0.0)
20 :
21 : pwm.stop()
22 : GPIO.cleanup()
```

13 : cnt 변수 값을 0부터 3 미만까지 1씩 증가시켜가면서 14~17줄의 동작을 3회 반복합니다.

14,15 : 0도로 이동합니다. 서보가 회전하기 위해서는 일정한 시간이 필요하며, 여기서는 1초간 이동할 시간을 줍니다.

16,17 : 90도로 이동합니다.

2. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _09_pwm_servo_2.py
```

서보가 0도과 90도를 2초 주기로 3회 회전하는 것을 확인합니다.

13 서보 모터 0~180도 조절해보기

여기서는 0도에서 180도까지 조절해 보도록 합니다.

1. 다음과 같이 예제를 수정합니다.

_09_pwm_servo_3.py

```
1  : import RPi.GPIO as GPIO
2  : import time
3  :
4  : servo_pin =18
5  :
6  : GPIO.setmode(GPIO.BCM)
7  :
8  : GPIO.setup(servo_pin, GPIO.OUT)
```

```
9   :
10  :  pwm = GPIO.PWM(servo_pin, 50)
11  :  pwm.start(3.0)
12  :
13  :  for t_high in range(30,125):
14  :    pwm.ChangeDutyCycle(t_high/10.0)
15  :    time.sleep(0.02)
16  :
17  :  pwm.ChangeDutyCycle(3.0)
18  :  time.sleep(1.0)
19  :  pwm.ChangeDutyCycle(0.0)
20  :
21  :  pwm.stop()
22  :  GPIO.cleanup()
```

13 : for 문을 사용하여 t_high 변수 값을 30부터 125 미만까지 1 간격으로 변경하면서, 14,15줄을 수행합니다.

14 : pwm 객체에 대해 ChangeDutyCycle 함수를 호출하여 t_high 변수 값을 10.0으로 나눈 값을 인자로 줍니다. 이렇게 하면 3.0, 3.1, 3.2, ..., 12.4에 해당하는 비율 값이 차례로 인자로 넘어갑니다.

15 : 0.02초(=2밀리초) 동안 기다립니다.

17 : pwm 객체에 대해 ChangeDutyCycle 함수를 호출하여 servo_pin으로 나가는 PWM의 HIGH 구간의 비율을 3.0으로 설정하여 서보모터의 각도가 0도가 되게 합니다.

18 : 1.0초 동안 기다립니다.

19 : pwm 객체에 대해 ChangeDutyCycle 함수를 호출하여 servo_pin으로 나가는 사각 파형의 HIGH 구간을 0.0%로 설정합니다. 이렇게 하면 서보의 동작이 멈춥니다.

2. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _09_pwm_servo_3.py
```

서보모터가 약 1.9초 동안 0도에서 180도까지 회전한 후, 0도로 다시 돌아오는 것을 확인합니다.

3. 다음과 같이 예제를 수정합니다.

_09_pwm_servo_4.py

```
1   :  import RPi.GPIO as GPIO
2   :  import time
3   :
4   :  servo_pin =18
5   :
6   :  GPIO.setmode(GPIO.BCM)
7   :
8   :  GPIO.setup(servo_pin, GPIO.OUT)
9   :
10  :  pwm = GPIO.PWM(servo_pin, 50)
11  :  pwm.start(3.0)
12  :
13  :  for t_high in range(30,125):
```

```
14 :     pwm.ChangeDutyCycle(t_high/10.0)
15 :     time.sleep(0.02)
16 :
17 :  for t_high in range(124,30,-1):
18 :     pwm.ChangeDutyCycle(t_high/10.0)
19 :     time.sleep(0.02)
20 :
21 :  pwm.ChangeDutyCycle(0.0)
22 :
23 :  pwm.stop()
24 :  GPIO.cleanup()
```

17 : t_high 변수를 124부터 30 초과까지 1씩 감소시켜가면서, 18,19줄의 동작을 수행합니다.

4. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _09_pwm_servo_4.py
```

서보모터가 약 1.9초 동안 0도에서 180도까지 회전한 후, 다시 약 1.9초 동안 180도에서 0도까지 회전하는 것을 확인합니다.

04 input 함수

input 함수는 사용자 입력을 받는 함수입니다. 사용자로부터 명령을 받고자 할 경우 input 함수를 사용할 수 있습니다.

01 사용자 입력 받기

1. 다음과 같이 예제를 작성합니다.

_10_input.py

```
1 : try:
2 :     while True:
3 :         userInput = input() # for string
4 :         print(userInput)
5 :
6 : except KeyboardInterrupt:
7 :     pass
```

3 : input 함수를 호출하여 키보드로 입력받은 문자열을 userInput 변수로 받습니다.
4 : print 함수를 호출하여 사용자로부터 전달된 문자열을 출력합니다.

2. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _10_input.py
```

키보드를 통해 문자열, 숫자를 입력해봅니다.

프로그램을 강제 종료하기 위해서는 CTRL 키를 누른 채로 c키를 눌러줍니다.

*** 파이썬 2.7의 경우는 raw_input 함수를 사용합니다.

02 파이썬 프롬프트 흉내내기

여기서는 파이썬 프롬프트처럼 프롬프트를 표시해 봅니다. 프롬프트는 사용자 입력 위치를 표시하는 문자열입니다.

1. 다음과 같이 예제를 작성합니다.

_10_input_2.py

```
1 : try:
2 :     while True:
3 :         userInput = input(">>> ")
4 :         print("You entered", userInput)
```

```
5  :                if userInput =="quit()":
6  :                                    print("bye...")
7  :                                    break
8  :
9  : except KeyboardInterrupt:
10 :     pass
```

3 : input 함수 호출 시 인자로 “>>> ” 문자열을 줍니다. input 함수에 주는 인자는 사용자 입력 위치를 표시하는 문자열입니다.

4 : print 함수를 호출하여 사용자로부터 전달된 문자열을 출력합니다.

5 : 사용자 입력값이 “quit()” 문자열이면

6 : print 함수를 호출하여 “bye” 문자열을 출력하고

7 : break 문으로 while 문을 빠져 나옵니다.

2. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _10_input_2.py
```

키보드를 통해 문자열, 숫자를 입력해봅니다. 종료하기 위해서 quit()를 입력합니다.

프로그램을 강제 종료하기 위해서는 CTRL 키를 누른 채로 c키를 눌러줍니다.

05 RPi.GPIO.input 함수

RPi.GPIO.input 함수는 라즈베리파이에서 논리적으로 0, 1을 전기적으로는 0V, 3.3V를 읽는 함수입니다.

다음에 노랗게 표시된 GPIO 핀의 상태를 읽을 수 있습니다.

01 0, 1 읽어보기

여기서는 RPi.GPIO.input 함수를 이용하여 0과 1을 읽어봅니다.

1. 다음과 같이 예제를 작성합니다.

_11_gpio_input.py

```
1  : import RPi.GPIO as GPIO
2  :
3  : button_pin =27
4  :
5  : GPIO.setmode(GPIO.BCM)
6  :
7  : GPIO.setup(button_pin, GPIO.IN)
8  :
9  : try:
10 :   while True:
11 :           buttonInput = GPIO.input(button_pin)
12 :           print(buttonInput)
13 :
14 : except KeyboardInterrupt:
15 :   pass
16 :
17 : GPIO.cleanup()
```

1 : RPi.GPIO 모듈을 GPIO로 불러옵니다. RPi.GPIO 모듈은 5,7,11,17줄에 있는 setmode, setup, input, cleanup 함수를 가지고 있으며, 버튼 입력을 받기 위해 필요합니다.

3 : 정수 객체 27을 생성한 후, button_pin 변수로 가리키게 합니다. 여기서 27은 BCM GPIO 핀 번호를 나타냅니다.

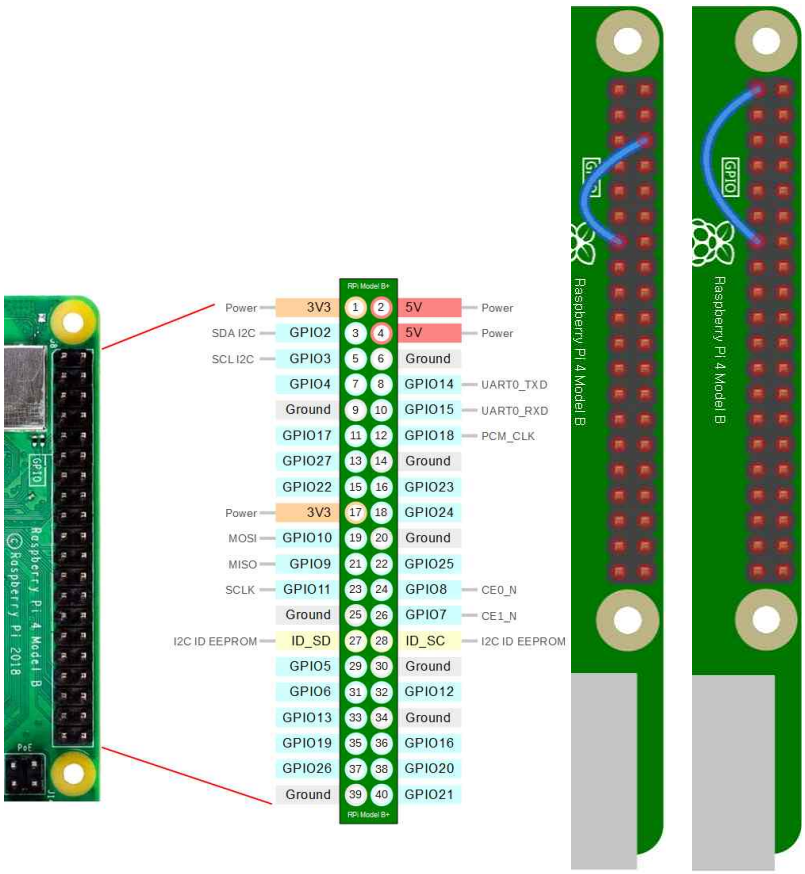
5 : GPIO.setmode 함수를 호출하여 BCM GPIO 핀 번호를 사용하도록 설정합니다.

7 : GPIO.setup 함수를 호출하여 button_pin을 GPIO 입력으로 설정합니다.
10 : 계속 반복해서 10~12줄을 수행합니다. while 문을 끝내기 위해서는 CTRL+c 키를 누릅니다.
11 : GPIO.input 함수를 호출하여 button_pin 값을 읽어 buttonInput 변수가 가리키도록 합니다.
12 : buttonInput 변수 값을 출력합니다.
17 : GPIO.cleanup 함수를 호출하여 GPIO 핀의 상태를 초기화해 줍니다.

2. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _11_gpio_input.py
```

다음과 같이 핀 맵을 참조하여 GPIO27 번 핀을 0V, 3.3V 연결해 테스트해 봅니다.



0V에 연결하여 0값이 출력되는 것을 확인합니다. 3.3V에 연결하여 1값이 출력되는 것을 확인합니다.

프로그램을 강제 종료하기 위해서는 CTRL 키를 누른 채로 c키를 눌러줍니다.

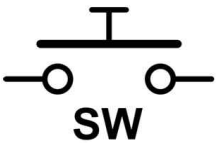
*** 선이 연결되어 있지 않은 상태에서도 0 또는 1이 입력되기도 합니다. 이 경우 핀이 떠 있는 상태라고 하며 값이 정의되지 않은 상태입니다. 따라서 입력되는 값에 대해 논리적인 의미를 두지 않습니다.

02 푸시버튼 살펴보기

본책에서 사용할 푸시 버튼의 모양은 다음과 같습니다.

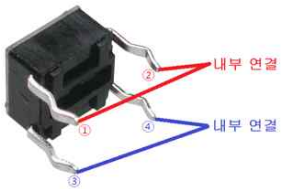
가운데 버튼을 누르면 양 쪽의 핀이 연결되는 구조입니다.

푸시 버튼을 나타내는 기호는 다음과 같고, 극성은 없습니다.

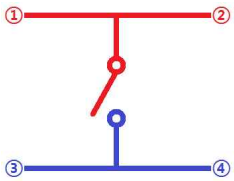


일반적인 푸시 버튼의 모양은 다음과 같습니다.

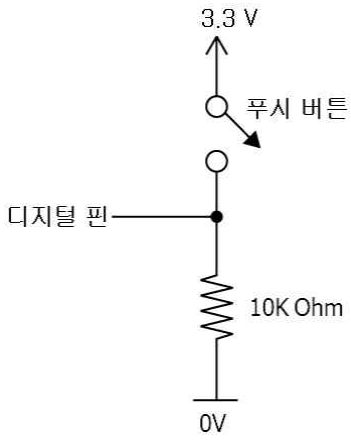
다음과 같이 두 쌍의 핀이 있으며, 각 쌍은 내부적으로 연결되어 있습니다.



내부적인 연결은 다음과 같습니다.

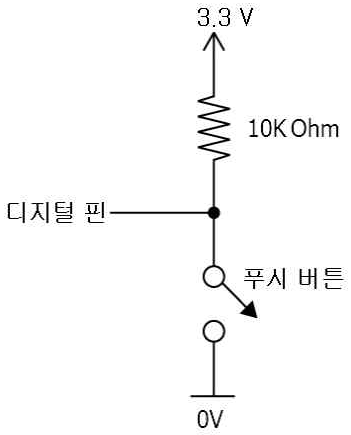


버튼 입력 회로는 일반적으로 다음과 같습니다.



이 경우 디지털 핀은 버튼이 눌리지 않았을 때는 10K Ohm 저항을 통해 0V로 연결되며, 논리적으로 0 값이 입력됩니다(10K Ohm 저항 대신에 220 Ohm, 330 Ohm, 1K Ohm 저항을 사용하는 경우도 있습니다. 그러나 저항값이 너무 낮으면 흐르는 전류량이 많아져 전력 소모가 심해집니다). 버튼을 눌렀을 경우에 디지털 핀은 3.3V로 연결되며, 논리적으로 1 값이 입력됩니다. 저항이 없는 상태에서 버튼을 누를 경우 3.3V와 0V가 직접 연결되는 단락 회로(short-circuit)가 만들어지며, 이 경우 저항이 0 Ω에 가까운 회로가 만들어집니다. 이럴 경우 옴의 법칙(I = V/R)에 의해 아주 큰 전류가 흐르게 되고, 보호 회로가 없을 경우에 칩이 망가질 수 있습니다. 저항은 단락 회로를 방지하는 역할을 하게 됩니다.

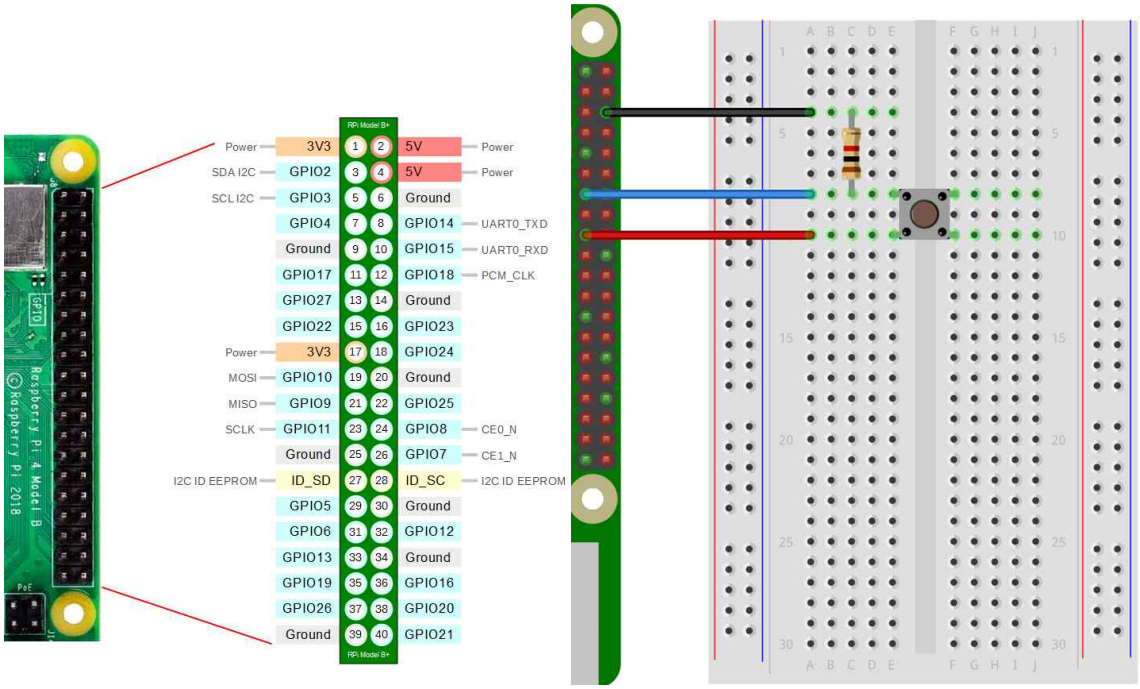
버튼 입력 회로는 다음과 같이 구성할 수도 있습니다.



이 경우 디지털 핀은 버튼이 눌리지 않았을 때는 220 Ohm 저항을 통해 3.3V로 연결되며, 논리적으로 1 값이 입력됩니다. 220 Ohm 저항 대신에 330 Ohm, 1K Ohm, 10K Ohm 저항을 사용할 수도 있습니다. 버튼을 눌렀을 경우에 디지털 핀은 0V로 연결되며, 논리적으로 0 값이 입력됩니다.

03 버튼 회로 구성하기

핀 맵을 참조하여 다음과 같이 회로를 구성합니다.



버튼의 한 쪽 핀을 3.3V로 연결합니다. 그림에서는 빨간색 전선 부분입니다. 버튼의 다른 쪽 핀을 1K Ohm 저항을 통해 GND로 연결해 줍니다. 그림에서는 검은색 전선 부분입니다. 저항의 다른 쪽 핀을 2 번 핀에 연결합니다.

다음과 같이 08_gpio_input.py 프로그램을 수행하여 테스트를 수행합니다.

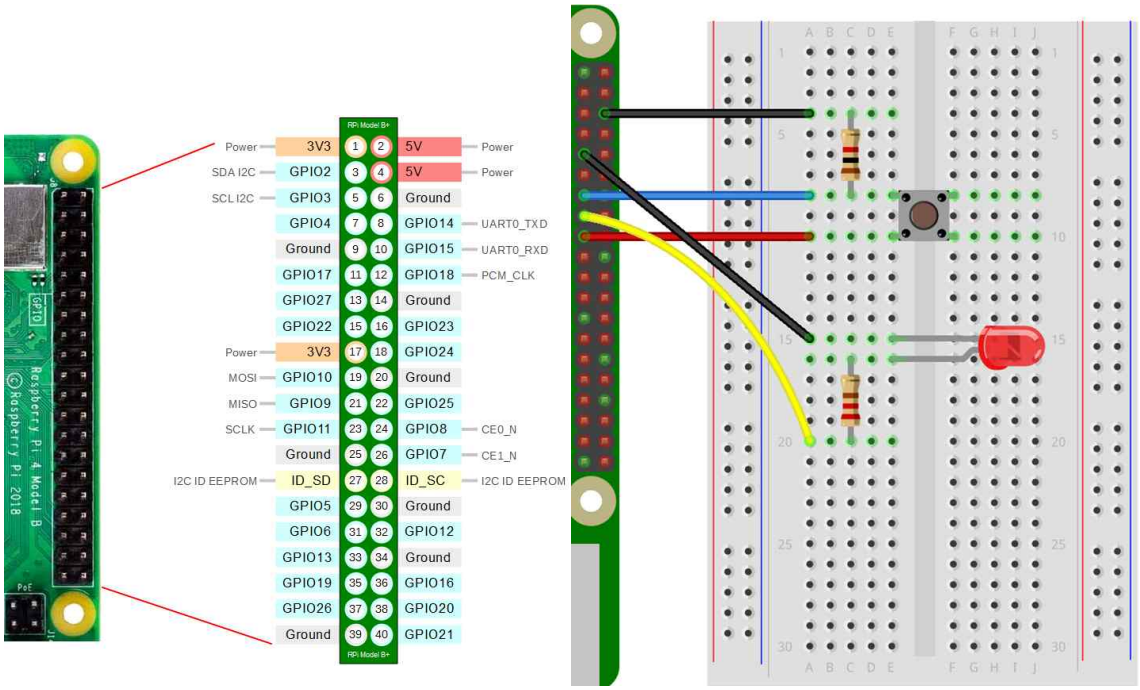
```
$ sudo python3 _11_gpio_input.py
```

버튼을 누른 채 값을 읽어 봅니다. 버튼을 떼고 값을 읽어봅니다.

04 버튼 값에 따라 LED 켜기

여기서는 버튼을 누르면 LED가 켜지고 버튼을 떼면 LED가 꺼지도록 프로그램을 작성해 보도록 합니다.

1. 핀 맵을 참조하여 다음과 같이 회로를 구성합니다.



버튼의 한 쪽 핀을 3.3V로 연결합니다. 그림에서는 빨간색 전선 부분입니다. 버튼의 다른 쪽 핀을 10K Ohm 저항을 통해 GND로 연결해 줍니다. 그림에서는 검은색 전선 부분입니다. 저항의 다른 쪽 핀을 2 번 핀에 연결합니다. LED의 긴 핀(+)을 220 Ohm 저항을 통해 라즈베리파이 보드의 3번 핀에 연결합니다. LED의 짧은 핀(-)은 GND 핀에 연결합니다.

2. 다음과 같이 예제를 작성합니다.

_11_gpio_input_2.py

```
1  : import RPi.GPIO as GPIO
2  :
3  : button_pin =27
4  : led_pin =22
5  :
6  : GPIO.setmode(GPIO.BCM)
7  :
8  : GPIO.setup(button_pin, GPIO.IN)
9  : GPIO.setup(led_pin, GPIO.OUT)
10 :
11 : try:
12 :   while True:
13 :           buttonInput = GPIO.input(button_pin)
14 :           GPIO.output(led_pin, buttonInput)
15 :
16 : except KeyboardInterrupt:
17 :   pass
18 :
19 : GPIO.cleanup()
```

4 : 정수 객체 22를 생성한 후, led_pin 변수로 가리키게 합니다.

9 : led_pin을 출력으로 설정합니다.

13 : GPIO.input 함수를 호출하여 button_pin 값을 읽어 buttonInput 변수가 가리키도록 합

니다.
14 : GPIO.output 함수를 호출하여 buttonInput 값을 led_pin으로 씁니다.

3. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _11_gpio_input_2.py
```

버튼을 누르면 LED가 켜지고 버튼을 떼면 LED가 꺼지는 것을 확인합니다.

프로그램을 강제 종료하기 위해서는 CTRL 키를 누른 채로 c키를 눌러줍니다.

05 버튼 토글하기

이전 예제에서는 버튼을 누르고 있어야만 LED가 켜졌습니다. 버튼을 떼게 되면 LED가 꺼지게 되어 불편합니다. 여기서는 버튼을 한 번 누르면 LED가 켜지고, 한 번 더 누르면 LED가 꺼지도록 해 봅니다.

1. 다음과 같이 예제를 수정합니다.

_11_gpio_input_3.py

```
1  : import RPi.GPIO as GPIO
2  :
3  : button_pin =27
4  : led_pin =22
5  :
6  : GPIO.setmode(GPIO.BCM)
7  :
8  : GPIO.setup(button_pin, GPIO.IN)
9  : GPIO.setup(led_pin, GPIO.OUT)
10 :
11 : buttonInputPrev = False
12 : ledOn = False
13 :
14 : try:
15 :     while True:
16 :             buttonInput = GPIO.input(button_pin)
17 :
18 :             if buttonInput and not buttonInputPrev:
19 :                     print("rising edge")
20 :                     ledOn = True if not ledOn else False
21 :                     GPIO.output(led_pin, ledOn)
22 :             elif not buttonInput and buttonInputPrev:
23 :                     print("falling edge")
24 :             else: pass
25 :
26 :             buttonInputPrev = buttonInput
27 :
28 : except KeyboardInterrupt:
29 :     pass
```

```
30 :
31 : GPIO.cleanup()
```

11 : buttonInputPrev 변수를 선언한 후, False 값으로 초기화합니다. buttonInputPrev 변수는 바로 전 GPIO.input 함수가 호출되었을 때의 버튼의 상태 값을 저장하는 변수입니다.
12 : ledOn 변수를 선언한 후, False 값으로 초기화합니다. ledOn 변수는 LED가 켜진 상태를 저장하는 변수입니다.
16 : GPIO.input 함수를 호출하여 button_pin 값을 읽어 buttonInput 변수가 가리키도록 합니다.
18 : buttonInput 변수가 True를 가리키고, 즉, 현재 버튼이 눌려졌고, buttonInputPrev 변수가 True가 아닌 False를 가리키고, 즉, 이전에 버튼이 눌려지지 않았으면
19 : print 함수를 호출하여 “rising edge” 문자열을 출력하고
20 : ledOn 변수가 True 또는 False를 가리키게 합니다. ledOn 변수가 False를 가리키고 있었다면 True를 가리키도록 변경하고 그렇지 않을 경우, 즉, ledOn 변수가 True를 가리키고 있었다면 False를 가리키도록 변경합니다.
21 : GPIO.output 함수를 호출하여 led_pin에 ledOn 값을 씁니다.
22 : 그렇지 않고 buttonInput 변수가 False를 가리키고, 즉, 현재 버튼이 눌려져있지 않고, buttonInputPrev 변수가 True를 가리키고 있으면, 즉, 이전에 버튼이 눌려져 있으면
23 : print 함수를 호출하여 “falling edge” 문자열을 출력하고
24 : 그렇지 않으면, 즉, buttonInput 값과 buttonInputPrev 값이 동시에 True이거나 동시에 False이면 아무것도 수행하지 않습니다.
26 : buttonInput이 가리키는 값을 buttonInputPrev 변수가 가리키도록 합니다.

2. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _11_gpio_input_3.py
```

버튼을 누르면 LED가 켜지고 버튼을 떼면 LED가 꺼지는 것을 확인합니다.

프로그램을 강제 종료하기 위해서는 CTRL 키를 누른 채로 c키를 눌러줍니다.

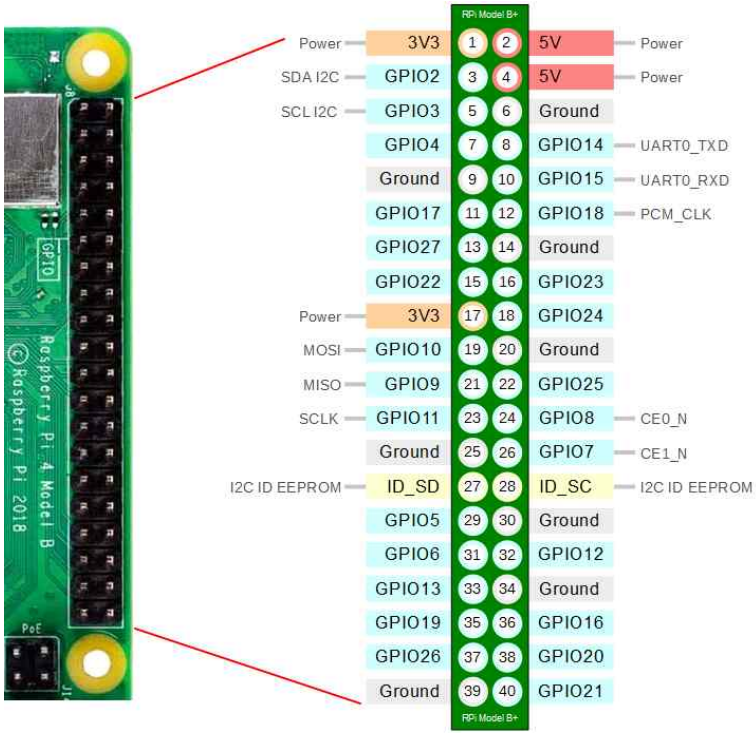
06 RPi.GPIO.add_event_callback 함수

이전 예제에서 버튼을 한 번 누르면 LED가 켜지고, 한 번 더 누르면 LED가 꺼지도록 해 보았습니다. 이 경우 외부 인터럽트를 이용해서 해결할 수도 있습니다.

여기서는 외부 인터럽트에 대해 살펴보고, 외부 인터럽트 처리기를 구현해 봅니다.

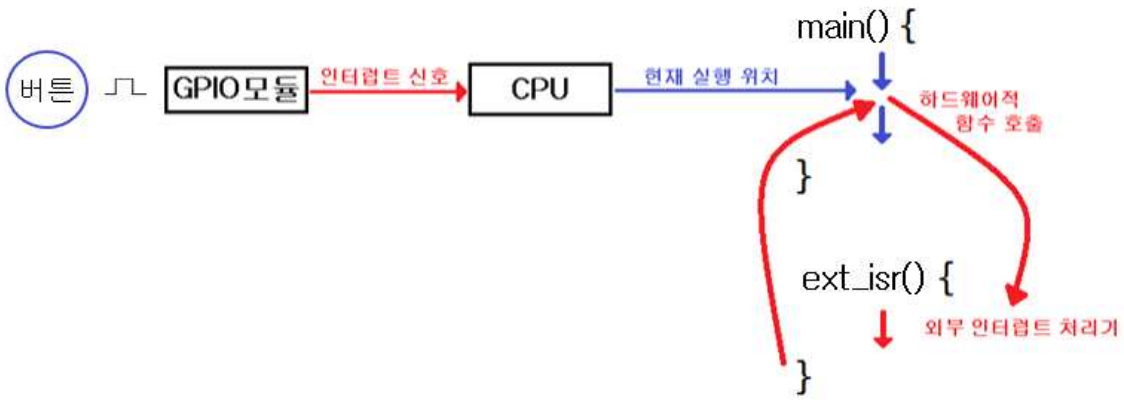
01 외부 인터럽트 살펴보기

WiringPi GPIO 핀은 모두 외부 인터럽트를 발생시키도록 설정할 수 있습니다.



외부 인터럽트 처리하기

GPIO 핀으로 입력되는 값이 0에서 1로 또는 1에서 0으로 신호가 바뀌면 GPIO 모듈을 통해서 BCM2835 내부에 있는 CPU로 인터럽트 신호를 보낼 수 있습니다. CPU는 인터럽트 신호를 받으면, 하드웨어적으로 함수를 호출하게 되는데, 이 때 수행되는 함수가 외부 인터럽트 처리 함수가 됩니다. CPU는 인터럽트 처리 함수를 수행하고 나서는 원래 수행되던 코드로 돌아갑니다.



02 버튼 인터럽트로 LED 켜기

여기서는 외부 인터럽트를 이용하여 LED를 켜고 끄도록 해봅니다.

1. 다음과 같이 예제를 작성합니다.

_12_ext_int.py

```
1  : import RPi.GPIO as GPIO
2  :
3  : led_state = False
4  : led_state_changed = False
5  : def buttonPressed(channel):
6  :   global led_state
7  :   global led_state_changed
8  :   led_state = True if not led_state else False
9  :   led_state_changed = True
10 :
11 : button_pin =27
12 : led_pin =22
13 :
14 : GPIO.setmode(GPIO.BCM)
15 :
16 : GPIO.setup(led_pin, GPIO.OUT)
17 :
18 : GPIO.setup(button_pin, GPIO.IN)
19 : GPIO.add_event_detect(button_pin, GPIO.RISING)
20 : GPIO.add_event_callback(button_pin, buttonPressed)
21 :
22 : try:
23 :   while True:
24 :           if led_state_changed == True:
25 :                   led_state_changed = False
26 :                   GPIO.output(led_pin, led_state)
27 :
28 : except KeyboardInterrupt:
29 :   pass
30 :
```

```
31  :  GPIO.cleanup()
```

3 : led_state 변수를 선언하여 False를 가리키게 합니다. led_state 변수는 LED의 상태 값을 가리키는 변수입니다.

4 : led_state_changed 변수를 선언하여 False를 가리키게 합니다. led_state_changed 변수는 LED의 상태가 바뀌었다는 것을 알리는 변수입니다.

5~9: buttonPressed 함수를 정의합니다. 버튼이 눌렸을 경우 수행되는 함수입니다.

6 : led_state 변수를 전역으로 선언합니다. 8 번째 줄에서 led_state 변수 값을 변경하는데, 전역 변수를 함수 내에서 변경하고자 할 경우엔 global 키워드를 붙여주어야 합니다. 그렇지 않을 경우 같은 이름을 가진 buttonPressed 함수의 지역 변수를 생성하려고 시도합니다.

7: led_state_changed 변수를 전역으로 선언합니다. 9 번째 줄에서 led_state_changed 변수 값을 변경하는데, 전역 변수를 함수 내에서 변경하고자 할 경우엔 global 키워드를 붙여주어야 합니다. 그렇지 않을 경우 같은 이름을 가진 buttonPressed 함수의 지역 변수가 생성됩니다.

8 : led_state 변수 값이 False이면 True를 led_state 변수에 대입합니다. led_state 변수 값이 True이면 False 값을 led_state 변수에 대입합니다.

9 : led_state_changed 변수 값을 True로 설정하여 led_state 변수 값이 바뀌었다는 것을 표시합니다.

18 : GPIO.setup 함수를 호출하여 button_pin을 GPIO 입력으로 설정합니다. 인터럽트 핀으로 사용할 경우에도 GPIO 입력으로 설정해 주어야 합니다.

19 : GPIO.add_event_detect 함수를 호출하여 button 핀의 값이 LOW에서 HIGH로 상승하는 순간 인터럽트가 발생하도록 설정합니다.

20 : GPIO.add_event_callback 함수를 호출하여 button 핀으로부터 인터럽트가 발생할 경우 buttonPressed 함수가 호출될 수 있도록 buttonPressed 함수를 등록합니다.

23 : 계속해서 23~26줄을 수행합니다.

24 : led_state_changed 변수 값이 True이면

25 : led_state_changed 변수 값을 False로 돌려놓고

26 : GPIO.output 함수를 호출하여 led_pin에 led_state 값을 씁니다.

2. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _12_ext_int.py
```

버튼을 누르면 LED가 켜지고, 다시 버튼을 누르면 LED가 꺼지는 것을 확인합니다.

프로그램을 강제 종료하기 위해서는 CTRL 키를 누른 채로 c키를 눌러줍니다.

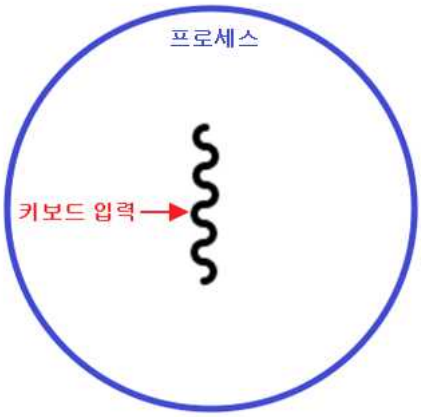
*** 버튼을 누르면 LED가 깜빡이며 이전상태를 유지하는 경우도 있습니다. 이런 현상을 채터링이라고 하며, 문제를 해결하기 위해서는 회로 상에는 축전지를 장착하고, 소프트웨어적으로는 일정 시간동안 버튼의 상태가 유지되는 것을 확인하는 루틴을 추가해주어야 합니다.

07 threading.Thread 클래스

라즈베리파이는 리눅스 운영체제를 기반으로 동작합니다. 리눅스 운영체제는 쓰레드 프로그램을 지원합니다. 쓰레드 프로그램은 하나의 프로그램에서 여러 가지 입력을 받아 처리해야하는 경우에 필요합니다. 예를 들어, 하나의 프로그램에서 키보드 입력, 버튼 입력, 시간 지연을 위한 시간 입력을 동시에 처리해야 할 경우에 필요합니다. 여기서는 쓰레드 생성을 위해 threading.Thread 클래스를 소개합니다.

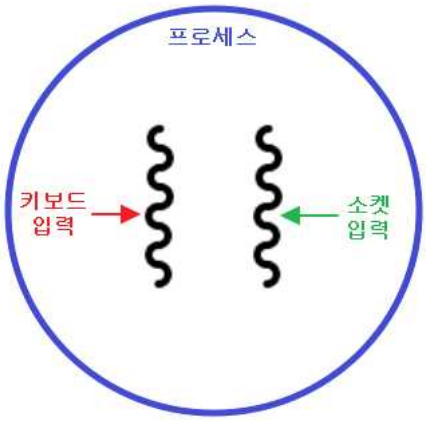
01 쓰레드 이해하기

우리가 작성하는 프로그램은 하나의 프로세스 상에서 수행됩니다. 프로세스란 CPU가 수행하는 작업의 단위로 하나의 프로그램을 수행하기 위한 환경을 나타냅니다. 하나의 프로세스는 기본적으로 하나의 쓰레드를 갖습니다. 이 쓰레드를 주 쓰레드라고 합니다. 우리가 작성하는 프로그램은 주 쓰레드 상에서 수행됩니다. 주 쓰레드는 키보드 입력을 기다리다가 키보드 입력이 있으면 키보드 입력을 처리하고 다시 키보드 입력을 기다리는 형태로 동작합니다. 주 쓰레드가 키보드 입력을 기다리는 동안에는 CPU에 의해 수행되지 않는 상태가 됩니다.



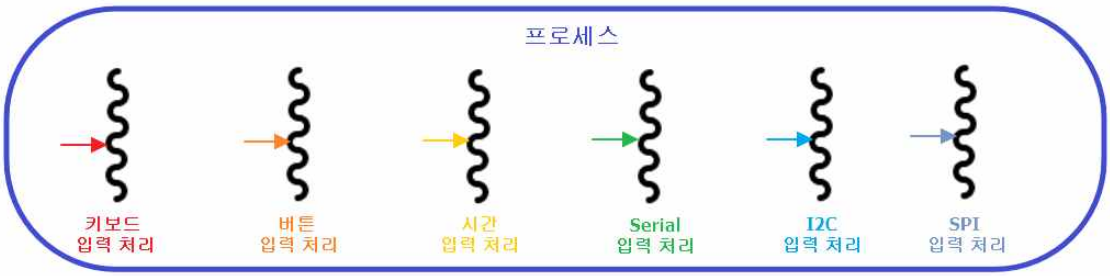
쓰레드는 하나의 입력을 처리하는 프로그램의 흐름을 나타냅니다. 주 쓰레드의 입력은 키보드 입력이 됩니다. 우리가 작성하는 대부분의 프로그램은 키보드 입력을 기다리다 처리하는 구조로 되어 있습니다. 그러다보니 하나의 쓰레드로 처리가 가능했습니다.

둘 이상의 쓰레드가 필요한 환경은 통신 프로그램입니다. 예를 들어, 온라인 게임과 같은 프로그램입니다. 통신 프로그램의 경우엔 지역 사용자의 키보드 입력도 처리해야 하지만 소켓을 통해 입력되는 원격 사용자의 키보드 입력도 처리해야 합니다. 그래서 원격 사용자의 입력을 처리하기 위한 쓰레드가 하나 더 필요합니다. 이 때 추가되는 쓰레드를 부 쓰레드라고 합니다.



새로 추가된 부 쓰레드는 소켓 입력을 기다리다가 소켓 입력이 있으면 소켓 입력을 처리하고 다시 소켓 입력을 기다리는 형태로 동작합니다. 부 쓰레드가 소켓 입력을 기다리는 동안에는 CPU에 의해 수행되지 않는 상태가 됩니다.

라즈베리파이의 경우 입력은 더 다양해집니다. 키보드 입력, 버튼 입력, Serial 통신 입력, I2C 통신 입력, SPI 통신 입력, 시간 지연을 위한 시간 입력을 모두 하나의 프로세스 내에서 처리해야 할 수도 있습니다. 버튼 입력의 경우 버튼 개수에 따라 입력이 늘어날 수도 있습니다. 이 경우 버튼 입력의 개수만큼 쓰레드가 필요할 수도 있습니다. Serial 통신의 경우 HC06 또는 HM10과 같은 블루투스 모듈을 연결하여 입력을 받을 수 있습니다. I2C 통신의 경우 MPU6050과 같은 가속도 자이로 센서 모듈, HMC5883L 기압계 센서 모듈, MS5611 지자계 센서 모듈을 연결하여 입력을 받을 수 있습니다. SPI 통신의 경우 MCP3208과 같은 ADC 센서 모듈을 연결하여 입력을 받을 수 있습니다.



키보드 입력 처리 쓰레드의 경우 input 함수에서 입력을 대기하게 됩니다. 버튼 입력 처리 쓰레드의 경우 RPi.GPIO.add_event_callback 함수를 통해 등록한 함수로부터 전달되는 입력을 기다리게 됩니다. 이 경우 메시지 큐를 통해 입력을 받게 됩니다. 메시지 큐는 뒤에서 살펴봅니다. 시간 입력 처리 쓰레드의 경우, time.sleep 함수에서 시간이 지나기를 기다리게 됩니다.

02 쓰레드 생성하기

여기서는 threading.Thread 클래스를 이용하여 쓰레드를 하나 생성한 후, 파이썬 프로그램을 읽고 수행하는 파이썬 쉘과 동시에 작업을 수행해 보도록 합니다.

1. 다음과 같이 예제를 작성합니다.

_13_threading.py

```
1  : import threading
2  : import time
3  :
4  : flag_exit = False
5  : def t1_main():
6  :     while True:
7  :         print("\tt1")
8  :         time.sleep(0.5)
9  :         if flag_exit: break
10 :
11 : t1 = threading.Thread(target=t1_main)
12 : t1.start()
13 :
14 : try:
15 :     while True:
16 :         print("main")
17 :         time.sleep(1.0);
18 :
19 : except KeyboardInterrupt:
20 :     pass
21 :
22 : flag_exit = True
23 : t1.join()
```

1 : threading 모듈을 불러옵니다. threading 모듈은 11,12,23 줄에 있는 Thread 생성자 함수, start, join 함수를 가지고 있으며 쓰레드를 사용하기 위해 필요합니다.

4 : flag_exit 변수를 선언하여 False 값으로 초기화합니다. flag_exit 변수가 True값을 가질 경우 9줄에서 쓰레드가 종료되도록 합니다. flag_exit 변수를 True로 설정하는 부분은 22줄입니다.

5~9 : 쓰레드가 수행할 t1_main 함수를 정의합니다.

6 : 계속 반복해서 6~9줄을 수행합니다.

7 : 탭,t1 문자열을 출력하고,

8 : 0.5초간 기다립니다.

9 : flag_exit 값이 True이면 while 문을 빠져 나온 후, 종료합니다.

11 : threading.Thread 객체를 생성하여 t1_main 함수를 수행할 t1 쓰레드를 하나 생성합니다.

12 : t1 객체에 대해 start 함수를 호출하여 쓰레드를 수행 가능한 상태로 변경합니다. 이제 쓰레드는 임의의 순간에 수행될 수 있습니다.

15 : 계속 반복해서 15~17줄을 수행합니다.

16 : main 문자열을 출력하고,

18 : 1.0초간 기다립니다.

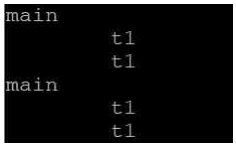
22 : 키보드 인터럽트가 발생하면 flag_exit를 True로 설정하여 쓰레드가 종료되도록 합니다.

23 : t1.join 함수를 호출하여 쓰레드가 종료되기를 기다립니다. 쓰레드가 종료되면 주 루틴도 종료됩니다.

2. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _13_threading.py
```

주 루틴과 t1 함수가 동시에 수행되는 것을 확인합니다. 주 루틴은 파이썬 쉘이 직접 수행하며 t1 함수는 threading.Thread 함수에 의해 생성된 쓰레드에서 수행됩니다.



프로그램을 강제 종료하기 위해서는 CTRL 키를 누른 채로 c키를 눌러줍니다.

03 쓰레드로 다중 작업하기

여기서는 threading.Thread 클래스를 이용하여 쓰레드를 하나 더 생성한 후, 파이썬 쉘과 함께 3 개의 쓰레드가 동시에 수행되도록 해 봅니다.

1. 다음과 같이 예제를 수정합니다.

_13_threading_2.py

```
1  : import threading
2  : import time
3  :
4  : flag_exit = False
5  : def t1_main():
6  :     while True:
7  :         print("\tt1")
8  :         time.sleep(0.5)
9  :         if flag_exit: break
10 :
11 : def t2_main():
12 :     while True:
13 :         print("\t\tt2")
14 :         time.sleep(0.2)
15 :         if flag_exit: break
16 :
17 : t1 = threading.Thread(target=t1_main)
18 : t1.start()
19 : t2 = threading.Thread(target=t2_main)
20 : t2.start()
21 :
22 : try:
23 :     while True:
24 :         userInput = input()
25 :         print(userInput)
26 :
27 : except KeyboardInterrupt:
28 :     pass
29 :
30 : flag_exit = True
31 : t1.join()
```

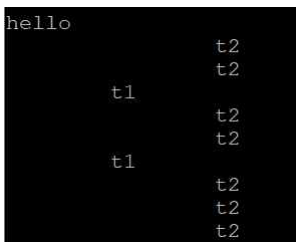
```
32 : t2.join()
```

11~15 : 쓰레드가 수행할 t2_main 함수를 정의합니다.
12 : 계속 반복해서 13~15줄을 수행합니다.
13 : 탭,탭,t2 문자열을 출력하고,
14 : 0.2초간 기다립니다.
15 : flag_exit 값이 True이면 while 문을 빠져 나온 후, 종료합니다.
19 : threading.Thread 객체를 생성하여 t2_main 함수를 수행할 t2 쓰레드를 하나 더 생성합니다.
20 : t2 객체에 대해 start 함수를 호출하여 쓰레드를 수행 가능한 상태로 변경합니다. 이제 쓰레드는 임의의 순간에 수행될 수 있습니다.
24 : 주 쓰레드는 input 함수를 호출하여 사용자 입력을 기다립니다.
25 : 사용자 입력을 출력합니다.
32 : t2.join 함수를 호출하여 쓰레드가 종료되기를 기다립니다. 쓰레드가 종료되면 주 루틴도 종료됩니다.

2. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _13_threading_2.py
```

주 루틴과 t1_main, t2_main 함수가 동시에 수행되는 것을 확인합니다. 주 루틴은 파이썬 쉘이 수행하며 t1_main, t2_main 함수는 threading.Thread 클래스에 의해 생성된 2개의 쓰레드에서 수행됩니다. 키보드에 hello 문자열을 입력한 후, 엔터키를 쳐 봅니다. t1_main, t2_main 함수를 수행하는 쓰레드는 주기적으로 화면으로 출력을 하고, 파이썬 쉘은 사용자 입력을 기다리다가 사용자 입력이 있으면 입력받은 문자열을 출력합니다.



프로그램을 강제 종료하기 위해서는 CTRL 키를 누른 채로 c키를 눌러줍니다.

04 쓰레드로 LED 점멸 반복해보기

여기서는 쓰레드를 생성하여 LED의 점멸을 반복해보도록 합니다.

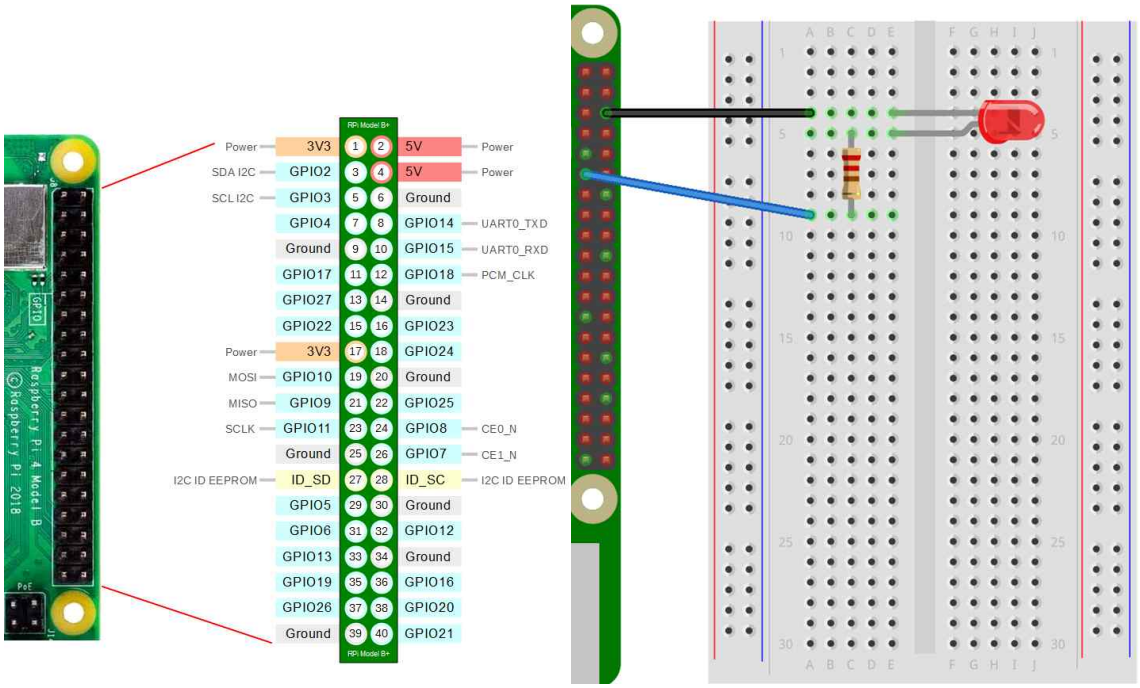
우리는 앞에서 다음과 같은 예제를 수행해 보았습니다.

```
_06_gpio_output_3.py
1  : import RPi.GPIO as GPIO
```

```
2  : import time
3  :
4  : led_pin =17
5  :
6  : GPIO.setmode(GPIO.BCM)
7  :
8  : GPIO.setup(led_pin, GPIO.OUT)
9  :
10 : try:
11 :     while True:
12 :             GPIO.output(led_pin, True)
13 :             time.sleep(0.5)
14 :             GPIO.output(led_pin, False)
15 :             time.sleep(0.5)
16 : except KeyboardInterrupt:
17 :     pass
18 :
19 : GPIO.cleanup()
```

이 예제는 단일 작업에 대한 테스트를 수행하는 데는 문제가 없지만 여러 가지 작업을 동시에 수행하고자 할 경우엔 문제가 생깁니다. threading.Thread 클래스를 이용하면 간단하게 LED의 점멸을 반복할 수 있습니다.

1. 핀 맵을 참조하여 다음과 같이 회로를 구성합니다.



LED의 긴 핀(+)을 220 Ohm 저항을 통해 라즈베리파이 보드의 0번 핀에 연결합니다. LED의 짧은 핀(-)은 GND 핀에 연결합니다.

2. 다음과 같이 예제를 수정합니다.

_13_threading_3.py

```
1  : import threading
2  : import time
3  : import RPi.GPIO as GPIO
4  :
5  : led_pin =17
6  :
7  : flag_exit = False
8  : def blink_led():
9  :     while True:
10 :             GPIO.output(led_pin, True)
11 :             time.sleep(0.5)
12 :             GPIO.output(led_pin, False)
13 :             time.sleep(0.5)
14 :
15 :             if flag_exit: break
16 :
17 : GPIO.setmode(GPIO.BCM)
18 : GPIO.setup(led_pin, GPIO.OUT)
19 :
20 : tBL = threading.Thread(target=blink_led)
21 : tBL.start()
22 :
23 : try:
24 :     while True:
25 :             print("main")
26 :             time.sleep(1.0);
27 :
28 : except KeyboardInterrupt:
29 :     pass
30 :
31 : flag_exit = True
32 : tBL.join()
```

8~15 : 쓰레드가 수행할 blink_led 함수를 정의합니다.

9 : 계속 반복해서 9~15줄을 수행합니다.

10~13 : 앞의 예제와 똑같이 작성합니다.

20 : threading.Thread 객체를 생성하여 blink_led 함수를 수행할 tBL 쓰레드를 하나 생성합니다.

21 : tBL 객체에 대해 start 함수를 호출하여 쓰레드를 수행 가능한 상태로 변경합니다. 이제 쓰레드는 임의의 순간에 수행될 수 있습니다.

24 : 계속 반복해서 24~26줄을 수행합니다.

25 : main 문자열을 출력하고,

26 : 1.0초간 기다립니다.

2. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _13_threading_3.py
```

주 루틴에서는 1초에 한 번씩 main 문자열이 출력되고, blink_led 함수에서는 1초 주기로 LED 점멸을 반복합니다. 주 루틴은 파이썬 쉘에 의해서 수행되며 blink_led 함수는

threading.Thread 클래스에 의해서 생성된 tBL 쓰레드에 의해서 수행됩니다.

프로그램을 강제 종료하기 위해서는 CTRL 키를 누른 채로 c키를 눌러줍니다.

05 쓰레드로 LED 밝기 조절해보기

여기서는 쓰레드를 이용하여 LED의 밝기를 조절해봅니다.

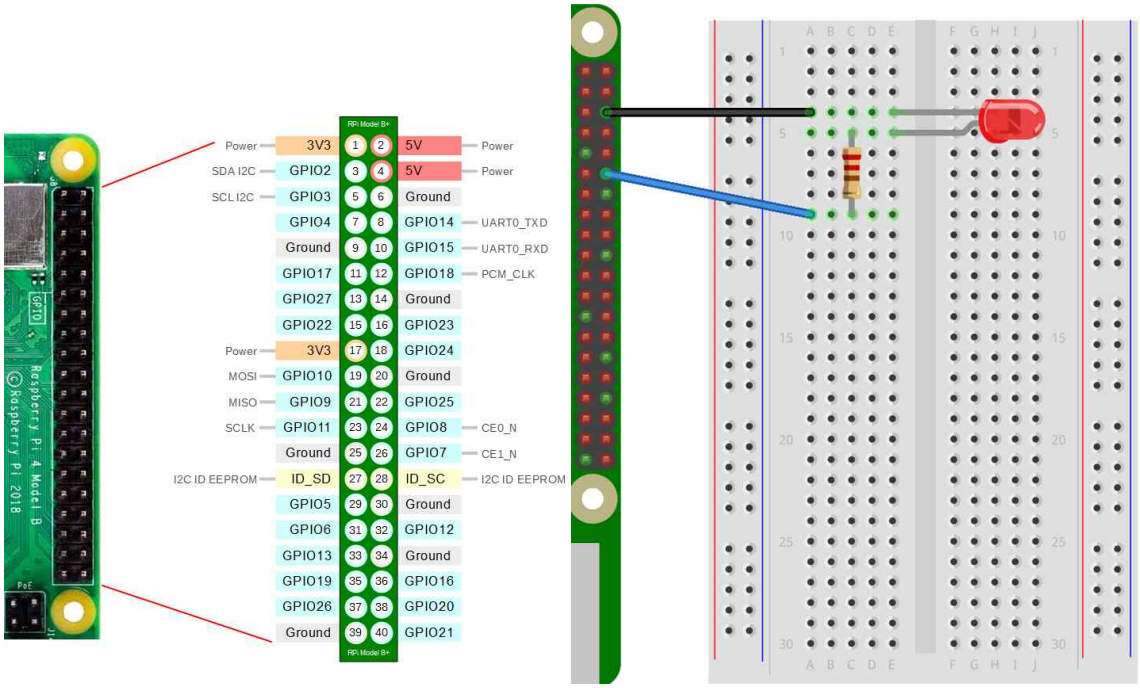
우리는 앞에서 다음과 같은 예제를 수행해 보았습니다.

_07_pwm_output_8.py

```
1  : import RPi.GPIO as GPIO
2  : import time
3  :
4  : led_pin =18
5  :
6  : GPIO.setmode(GPIO.BCM)
7  :
8  : GPIO.setup(led_pin, GPIO.OUT)
9  :
10 : pwm = GPIO.PWM(led_pin, 1000.0) # 1.0Hz
11 : pwm.start(0.0) # 0.0~100.0
12 :
13 : try:
14 :   while True:
15 :           for t_high in range(0,101):
16 :                   pwm.ChangeDutyCycle(t_high)
17 :                   time.sleep(0.01)
18 :           for t_high in range(100,-1,-1):
19 :                   pwm.ChangeDutyCycle(t_high)
20 :                   time.sleep(0.01)
21 : except KeyboardInterrupt:
22 :   pass
23 :
24 : pwm.stop()
25 : GPIO.cleanup()
```

이 예제의 경우도 threading.Thread 클래스를 이용하면 쉽게 해결할 수 있습니다.

1. 핀 맵을 참조하여 다음과 같이 회로를 구성합니다.



LED의 긴 핀(+)을 220 Ohm 저항을 통해 라즈베리파이 보드의 1번 핀에 연결합니다. LED의 짧은 핀(-)은 GND 핀에 연결합니다.

2. 다음과 같이 예제를 수정합니다.

_13_threading_4.py

```
1  : import threading
2  : import time
3  : import RPi.GPIO as GPIO
4  :
5  : led_pin =18
6  : GPIO.setmode(GPIO.BCM)
7  : GPIO.setup(led_pin, GPIO.OUT)
8  : pwm = GPIO.PWM(led_pin, 1000.0)
9  : pwm.start(0)
10 :
11 : flag_exit = False
12 : def fading_led():
13 :     while True:
14 :             for t_high in range(0,101):
15 :                     pwm.ChangeDutyCycle(t_high)
16 :                     time.sleep(0.01)
17 :             for t_high in range(100,-1,-1):
18 :                     pwm.ChangeDutyCycle(t_high)
19 :                     time.sleep(0.01)
20 :
21 :             if flag_exit: break
22 :
23 : tFL = threading.Thread(target=fading_led)
24 : tFL.start()
25 :
26 : try:
```

```
27 :     while True:
28 :             print("main")
29 :             time.sleep(1.0);
30 :
31 : except KeyboardInterrupt:
32 :     pass
33 :
34 : flag_exit = True
35 : tFL.join()
36 :
37 : pwm.stop()
38 : GPIO.cleanup()
```

12~21 : 쓰레드가 수행할 fading_led 함수를 정의합니다.

13 : 계속 반복해서 13~21줄을 수행합니다.

14~19 : 앞의 예제와 똑같이 작성합니다.

23 : threading.Thread 객체를 생성하여 fading_led 함수를 수행할 tFL 쓰레드를 하나 생성합니다.

24 : tFL 객체에 대해 start 함수를 호출하여 쓰레드를 수행 가능한 상태로 변경합니다. 이제 쓰레드는 임의의 순간에 수행될 수 있습니다.

27 : 계속 반복해서 27~29줄을 수행합니다.

28 : main 문자열을 출력하고,

29 : 1.0초간 기다립니다.

3. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _13_threading_4.py
```

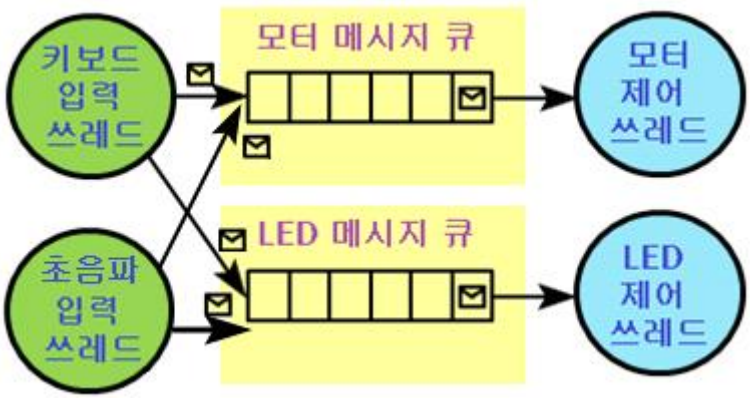
파이썬 쉘이 수행하는 주 루틴에서는 1초에 한 번씩 main 문자열이 출력되고, fading_led 함수에서는 약 2초 주기로 LED가 밝아지고 어두워지기를 반복합니다.

프로그램을 강제 종료하기 위해서는 CTRL 키를 누른 채로 c키를 눌러줍니다.

08 메시지 큐 통신

우리는 앞에서 쓰레드 프로그램을 작성해 보았습니다. 프로그램을 두 개 이상의 쓰레드로 구성할 경우, 쓰레드 간에 데이터를 주고받아야 하는 경우가 있을 수 있습니다. 또, 인터럽트를 사용해야 할 경우 인터럽트 처리 함수에서 쓰레드로 데이터를 보내야 하는 경우가 있을 수 있습니다. 이 때, 사용할 수 있는 방법이 바로 메시지 큐입니다.

예를 들어, 인터넷을 통한 원격 키보드 입력과 초음파 센서 입력을 받아 RC카의 모터와 전조등/후미등을 제어하는 프로그램을 쓰레드로 구성하는 경우를 생각해 보도록 합니다. 일단 입력이 2개이므로 적어도 2개의 쓰레드가 필요합니다. 즉, 원격 키보드 입력 쓰레드, 초음파 입력 쓰레드가 필요합니다. 기본적으로는 원격 키보드 입력을 통해 모터와 전조등/후미등의 제어가 이루어지게 됩니다. 또, 초음파 센서를 통해 물체가 감지될 경우 모터의 동작을 멈추고 비상 상황을 나타내도록 전조등/후미등을 제어해야 할 수도 있습니다. 하나의 출력에 대해 2 이상의 쓰레드가 접근할 경우에 출력을 위한 쓰레드가 있으면 편리합니다. 따라서 모터 출력과 전조등/후미등 출력을 담당할 쓰레드 2개가 더 필요합니다. 즉, 모터 제어 쓰레드, 전조등/후미등 제어 쓰레드가 필요합니다. 이상에서 프로그램의 구성을 원격 키보드 입력 쓰레드, 물체를 감지하기 위한 초음파 센서 입력 쓰레드, 바퀴에 장착된 모터를 제어하는 쓰레드, 전조등과 후미 등 역할을 하는 LED를 제어하는 쓰레드로 구성할 수 있습니다. 사용자가 전진 명령을 보낼 경우 키보드 입력 쓰레드는 명령을 해석한 후, 전진 메시지를 만들어 모터 제어 쓰레드로 보냅니다. 사용자가 전조등 후미 등 점등 명령을 보내면 키보드 입력 쓰레드는 명령을 해석한 후, 전조등 후미 등 점등 메시지를 만들어 LED 제어 쓰레드로 보냅니다. 초음파 입력 쓰레드는 초음파 센서를 통해 전방에 물체를 감지할 경우, 정지 메시지를 만들어 모터 제어 쓰레드로 보냅니다. 또 비상 상태를 나타내도록 후미등 깜빡이 메시지를 만들어 LED 제어 쓰레드로 보냅니다.



프로그램을 쓰레드로 구성할 경우 이와 같이 쓰레드 간에 메시지를 주고받는 경우가 필요해지며, 이 때, 메시지 큐를 통해 메시지를 주고받게 됩니다.

여기서는 쓰레드와 쓰레드 간, 인터럽트 처리 함수와 쓰레드 간에 데이터를 주고받기 위한 메시지 큐의 사용법을 살펴봅니다.

01 주 루틴과 쓰레드 간 메시지 큐 통신하기

여기서는 queue.Queue 클래스를 이용하여 메시지 큐를 생성한 후, 메시지 큐를 이용하여 파이썬 쉘과 쓰레드 간에 메시지를 주고받아 봅니다.

1. 다음과 같이 예제를 작성합니다.

_14_mqueue.py

```
1  : import queue
2  : import threading
3  : import time
4  :
5  : HOW_MANY_MESSAGES =10
6  : mq = queue.Queue(HOW_MANY_MESSAGES)
7  :
8  : flag_exit = False
9  : def t1():
10 :     value =0
11 :
12 :     while True:
13 :             value = value +1
14 :             mq.put(value)
15 :             time.sleep(0.1)
16 :
17 :             if flag_exit: break
18 :
19 : tMQ = threading.Thread(target=t1)
20 : tMQ.start()
21 :
22 : try:
23 :     while True:
24 :             value = mq.get()
25 :             print("Read Data %d" %value)
26 :
27 : except KeyboardInterrupt:
28 :     pass
29 :
30 : flag_exit = True
31 : tMQ.join()
```

1 : queue 모듈을 불러옵니다. queue 모듈은 6,14,24 줄에 있는 Queue 생성자 함수, put, get 함수를 가지고 있으며 메시지 큐를 사용하기 위해 필요합니다.

5 : HOW_MANY_MESSAGES 변수를 선언한 후, 10으로 설정합니다.

HOW_MANY_MESSAGES는 메시지 큐에 저장할 수 있는 최대 메시지의 개수를 나타냅니다.
6 : queue.Queue 객체를 생성하여 메시지 큐를 생성합니다. 객체 생성 시 최대 메시지의 개수를 인자로 줍니다.
8 : flag_exit 변수를 선언하여 False 값으로 초기화합니다. flag_exit 변수가 True값을 가질 경우 17줄에서 쓰레드가 종료되도록 합니다. flag_exit 변수를 True로 설정하는 부분은 30줄입니다.
9~17 : 쓰레드가 수행할 t1 함수를 정의합니다.
10 : 보내고자 하는 메시지를 저장할 value 변수를 하나 선언합니다.
12 : 계속 반복해서 12~17줄을 수행합니다.
13 : value 변수의 값을 하나 증가시킵니다.
14 : mq.put 함수를 호출하여 메시지 큐에 value 값을 씁니다.
15 : 0.1 초 동안 기다립니다.
17 : flag_exit 값이 True이면 while 문을 빠져 나온 후, 종료합니다.
18 : threading.Thread 객체를 생성하여 t1 함수를 수행할 tMQ 쓰레드를 하나 생성합니다.
19 : tMQ 객체에 대해 start 함수를 호출하여 쓰레드를 수행 가능한 상태로 변경합니다. 이제 쓰레드는 임의의 순간에 수행될 수 있습니다.
23 : 계속 반복해서 23~25줄을 수행합니다.
24 : mq.get 함수를 호출하여 메시지 큐에 있는 메시지를 value 변수로 읽어냅니다. 읽을 메시지가 없을 경우 쓰레드는 메시지를 기다리게 됩니다. 24줄에 있는 value 변수는 주 루틴의 변수이며, 10줄에 있는 value 변수는 t1 함수의 변수로 서로 다른 변수입니다.
25 : value 값을 출력합니다.
30 : 키보드 인터럽트가 발생하면 flag_exit를 True로 설정하여 쓰레드가 종료되도록 합니다.
31 : tMQ.join 함수를 호출하여 쓰레드가 종료되기를 기다립니다. 쓰레드가 종료되면 주 루틴도 종료됩니다.

2. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _14_mqueue.py
```

다음과 같이 메시지가 전달되어 출력되는 것을 확인합니다.

```
Read Data 20
Read Data 21
Read Data 22
Read Data 23
Read Data 24
Read Data 25
Read Data 26
Read Data 27
Read Data 28
Read Data 29
```

프로그램을 강제 종료하기 위해서는 CTRL 키를 누른 채로 c키를 눌러줍니다.

02 인터럽트 처리 함수와 쓰레드 간 메시지 큐 통신하기

여기서는 인터럽트 처리 함수와 파이썬 쉘 간에 메시지를 주고받아 봅니다.

우리는 앞에서 다음과 같은 예제를 수행해 보았습니다.

_12_ext_int.py

```
1  : import RPi.GPIO as GPIO
2  :
3  : led_state = False
4  : led_state_changed = False
5  : def buttonPressed(channel):
6  :   global led_state
7  :   global led_state_changed
8  :   led_state = True if not led_state else False
9  :   led_state_changed = True
10 :
11 : button_pin =27
12 : led_pin =22
13 :
14 : GPIO.setmode(GPIO.BCM)
15 :
16 : GPIO.setup(led_pin, GPIO.OUT)
17 :
18 : GPIO.setup(button_pin, GPIO.IN)
19 : GPIO.add_event_detect(button_pin, GPIO.RISING)
20 : GPIO.add_event_callback(button_pin, buttonPressed)
21 :
22 : try:
23 :   while True:
24 :           if led_state_changed == True:
25 :                   led_state_changed = False
26 :                   GPIO.output(led_pin, led_state)
27 :
28 : except KeyboardInterrupt:
29 :   pass
30 :
31 : GPIO.cleanup()
```

이 예제의 경우엔 파이썬 프로그램의 주 루틴에 있는 while 문(23줄)에서 바쁜 대기를 수행하게 됩니다. 리눅스에는 많은 프로세스와 쓰레드들이 작업을 수행하는 환경이며, 바쁜 대기를 수행할 경우엔 쓰레드가 특별히 하는 일 없이 while 루프를 돌게 됩니다. 이럴 경우 시스템 전체적으로 성능을 떨어뜨릴 수가 있습니다. 여기서는 메시지 큐를 이용하여 데이터가 새로 들어올 때만 처리하도록 루틴을 수정해 보도록 합니다.

1. 다음과 같이 예제를 작성합니다.

_14_mqueue_2.py

```
1  : import queue
2  : import RPi.GPIO as GPIO
3  : import time
4  :
5  : HOW_MANY_MESSAGES =10
6  : mq =queue.Queue(HOW_MANY_MESSAGES)
7  :
8  : led_state = False
9  : def buttonPressed(channel):
10 :     global led_state
11 :     led_state = True if not led_state else False
12 :     mq.put(led_state)
13 :
14 : button_pin =27
15 : led_pin =22
16 :
17 : GPIO.setmode(GPIO.BCM)
18 :
19 : GPIO.setup(led_pin, GPIO.OUT)
20 :
21 : GPIO.setup(button_pin, GPIO.IN)
22 : GPIO.add_event_detect(button_pin, GPIO.RISING)
23 : GPIO.add_event_callback(button_pin, buttonPressed)
24 :
25 : try:
26 :     while True:
27 :             value = mq.get()
28 :             GPIO.output(led_pin, value)
29 :
30 : except KeyboardInterrupt:
31 :     pass
32 :
33 : GPIO.cleanup()
```

23 : GPIO.add_event_callback 함수를 호출하여 버튼을 눌러 신호가 올라갈 때 수행될 buttonPressed 함수를 등록합니다.

27 : mq.get 함수를 호출하여 메시지 큐에 있는 메시지를 value 변수로 읽어냅니다. 읽을 메시지가 없을 경우 쓰레드는 메시지를 기다리게 됩니다.

28 : GPIO.output 함수를 호출하여 led_pin에 value 변수 값을 출력합니다.

8 : 보내고자 하는 메시지를 저장할 정수 변수 led_state를 선언합니다.

11 : led_state 값이 False이면 True로 그렇지 않으면 False로 led_state 값을 변경합니다.

12 : mq.put 함수를 호출하여 메시지 큐에 led_state 값을 씁니다.

2. 다음과 같이 예제를 실행합니다.

```
$ sudo python3 _14_mqueue_2.py
```

버튼을 누르면 LED가 켜지고, 다시 버튼을 누르면 LED가 꺼지는 것을 확인합니다.

프로그램을 강제 종료하기 위해서는 CTRL 키를 누른 채로 c키를 눌러줍니다.

Chapter 03
입출력 함수 조합하기

우리는 지금까지 화면 출력, LED 출력, 부저 출력, 서보 출력, 키보드 입력, 버튼 입력 프로그램 작성법을 배웠습니다. 필자가 강의를 하면서 입력과 출력을 연결하는 프로그램을 작성하지 못하는 수강생이 의외로 많다는 것을 알게 되었습니다. 그러한 수강생들은 입력과 출력을 연결하는 프로그램을 작성한 후 아주 신기해하고 기뻐했습니다. 여기서는 입력과 출력을 연결하는 프로그램을 독자 여러분이 직접 작성해 보며, 스스로의 하드웨어 입출력 프로그래밍 실력을 가늠해봅니다. 또 그러한 과정에서 프로그래밍의 기쁨을 독자 여러분께 드리고자 합니다.

01 단위 입력 단위 출력 연결하기

여기서는 키보드 입력, 버튼 입력을 받아 LED를 켜고 끄거나, LED의 밝기를 조절하거나, 부저의 음을 조절하거나, 서보의 각도를 조절하는 프로그램을 독자 여러분이 스스로 작성해 봅니다.

01 도전과제 1

여기서는 키보드 입력을 받아 LED를 켜고 끄거나, LED의 밝기를 조절하거나, 부저의 음을 조절하거나, 서보의 각도를 조절하는 프로그램을 작성해 봅니다.

문제 1

라즈베리파이 보드의 GPIO 17번 핀으로 제어하는 LED 회로를 구성하시오.

키보드 입력을 받아 GPIO17 번 핀에 연결된 LED를 켜고 끄는 프로그램을 작성하시오.
n 키를 누르면 LED가 켜지고,
f 키를 누르면 LED가 꺼지도록 합니다.

파일 이름은 challange111.py로 합니다.

문제 2

라즈베리파이 보드의 GPIO18번 핀으로 제어하는 LED 회로를 구성하시오.

키보드 입력을 받아 GPIO18번 핀에 연결된 LED 밝기를 0%, 50%, 100%로 조절하는 프로그램을 작성하시오.
숫자 0 키를 누르면 0%,
숫자 5 키를 누르면 50%,
t 키를 누르면 100%로 조절하도록 합니다.

파일 이름은 challange112.py로 합니다.

문제 3

라즈베리파이 보드의 GPIO19번 핀으로 제어하는 부저 회로를 구성하시오.

키보드 입력을 받아 GPIO19번 핀에 연결된 부저로 도, 레, 미, 파, 솔, 라, 시, 도 음을 내는

피아노 프로그램을 작성하시오.
a 키를 누르면 4 옥타브 도를 0.5초 동안,
s 키를 누르면 4 옥타브 레를 0.5초 동안,
d 키를 누르면 4 옥타브 미를 0.5초 동안,
f 키를 누르면 4 옥타브 파를 0.5초 동안,
g 키를 누르면 4 옥타브 솔을 0.5초 동안,
h 키를 누르면 4 옥타브 라를 0.5초 동안,
j 키를 누르면 4 옥타브 시를 0.5초 동안,
k 키를 누르면 5 옥타브 도를 0.5초 동안 소리가 나도록 합니다.

파일 이름은 challange113.py로 합니다.

문제 4

라즈베리파이 보드의 GPIO19번 핀으로 제어하는 서보 회로를 구성하시오.

키보드 입력을 받아 GPIO19번 핀에 연결된 서보의 각도를 0도, 90도, 180도로 조절하는 프로그램을 작성하시오.
q 키를 누르면 0도,
w 키를 누르면 90도,
e 키를 누르면 180도로 조절하도록 합니다

파일 이름은 challange114.py로 합니다.

02 도전과제 2

여기서는 버튼 입력을 받아 LED를 켜고 끄거나, LED의 밝기를 조절하거나, 부저의 음을 조절하거나, 서보의 각도를 조절하는 프로그램을 작성해 봅니다.

문제 1

라즈베리파이 보드의 GPIO22번 핀으로 버튼 입력을 받고,
GPIO17번 핀으로 제어하는 LED 회로를 구성하시오.

버튼 입력을 GPIO22번 핀으로 받아 GPIO17번 핀에 연결된 LED를 켜고 끄는 프로그램을 작성하시오. 버튼 입력은 인터럽트로 처리합니다.
첫 번째 버튼 누름에 LED가 켜지고,
두 번째 버튼 누름에 LED가 꺼지도록 합니다.

파일 이름은 challange121.py로 합니다.

문제 2

라즈베리파이 보드의 GPIO22번 핀으로 버튼 입력을 받고,
GPIO18번 핀으로 제어하는 LED 회로를 구성하시오.

버튼 입력을 GPIO22번 핀으로 받아 GPIO18번 핀에 연결된 LED 밝기를 0%, 50%, 100%로 조절하는 프로그램을 작성하시오. 버튼 입력은 인터럽트로 처리합니다.
첫 번째 버튼 누름에 0%,
두 번째 버튼 누름에 50%,
세 번째 버튼 누름에 100%로 조절하도록 합니다.

파일 이름은 challange122.py로 합니다.

문제 3

라즈베리파이 보드의 GPIO22번 핀으로 버튼 입력을 받고,
GPIO19번 핀으로 제어하는 부저 회로를 구성하시오.

버튼 입력을 GPIO22번 핀으로 받아 GPIO19번 핀에 연결된 부저로 도, 레, 미, 파, 솔, 라, 시, 도 음을 내는 프로그램을 작성하시오. 버튼 입력은 인터럽트로 처리합니다.
첫 번째 버튼 누름에 4 옥타브 도를 0.5초 동안,
두 번째 버튼 누름에 4 옥타브 레를 0.5초 동안,
세 번째 버튼 누름에 4 옥타브 미를 0.5초 동안,
네 번째 버튼 누름에 4 옥타브 파를 0.5초 동안,
다섯 번째 버튼 누름에 4 옥타브 솔을 0.5초 동안,
여섯 번째 버튼 누름에 4 옥타브 라를 0.5초 동안,
일곱 번째 버튼 누름에 4 옥타브 시를 0.5초 동안,
여덟 번째 버튼 누름에 5 옥타브 도를 0.5초 동안 소리가 나도록 합니다.

파일 이름은 challange123.py로 합니다.

문제 4

라즈베리파이 보드의 GPIO22번 핀으로 버튼 입력을 받고,
GPIO19번 핀으로 제어하는 서보 회로를 구성하시오.

버튼 입력을 GPIO22번 핀으로 받아 GPIO19번 핀에 연결된 서보의 각도를 0도, 90도, 180도로 조절하는 프로그램을 작성하시오. 버튼 입력은 인터럽트로 처리합니다.
첫 번째 버튼 누름에 0도,

두 번째 버튼 누름에 90도,
세 번째 버튼 누름에 180도로 조절하도록 합니다.

파일 이름은 challange124.py로 합니다.

02 사용자 입력 다중 출력 연결하기

여기서는 사용자 입력을 받아 LED를 켜고 끄거나, LED의 밝기를 조절하거나, 서보의 각도를 조절하는 프로그램을 독자 여러분이 스스로 작성해 봅니다.

도전 과제

라즈베리파이 보드의 GPIO17번 핀으로 제어하는 빨간색 LED 회로,
라즈베리파이 보드의 GPIO18번 핀으로 제어하는 파란색 LED 회로,
라즈베리파이 보드의 GPIO19번 핀으로 제어하는 서보 회로를 구성하시오.

다음 예제를 완성하시오. 파일 이름은 challange200.py로 합니다.

```
def showMenu():
    print("==<<MENU>>==");
    print("n. 빨간색 LED 켜기");
    print("f. 빨간색 LED 끄기");
    print("0. 파란색 LED 밝기 0%");
    print("5. 파란색 LED 밝기 50%");
    print("t. 파란색 LED 밝기 100%");
    print("q. Servo 180도");
    print("w. Servo  90도");
    print("e. Servo   0도");

showMenu()
while True:
    userInput = input(">>>");
    # 여기를 채워 예제를 완성합니다.
```

03 다중 입력 다중 출력 연결하기

여기서는 키보드 입력을 받아 서보의 각도를 조절하거나, 버튼 입력을 받아 3개의 LED를 차

레대로 켜고 끄는 프로그램을 독자 여러분이 스스로 작성해 봅니다.

도전과제

회로 구성

라즈베리파이 보드의 GPIO22번 핀으로 버튼 입력을 받는 회로, 라즈베리파이 보드의 GPIO17번 핀으로 제어하는 빨간색 LED 회로, 라즈베리파이 보드의 GPIO27번 핀으로 제어하는 초록색 LED 회로, 라즈베리파이 보드의 GPIO23번 핀으로 제어하는 파란색 LED 회로, 라즈베리파이 보드의 GPIO19번 핀으로 제어하는 서보 회로를 구성하시오.

프로그래밍

키보드 입력을 받아 GPIO19번 핀에 연결된 서보의 각도를 0도, 90도, 180도로 조절하고, 버튼 입력을 GPIO22번 핀으로 받아 GPIO17, GPIO27, GPIO23 번 핀에 연결된 빨간색, 초록색, 파란색 LED를 차례로 켜는 프로그램을 작성하시오. 버튼 입력은 인터럽트로 처리합니다. 키보드 q 키를 누르면 서보 모터 0도, 키보드 w 키를 누르면 서보 모터 90도, 키보드 e 키를 누르면 서보 모터 180도로 조절하도록 합니다. 첫 번째 버튼 누름에 빨간색 LED가 켜지고, 두 번째 버튼 누름에 초록색 LED가 켜지고, 세 번째 버튼 누름에 파란색 LED가 켜지고, 네 번째 버튼 누름에 모든 LED가 꺼지도록 합니다. 파일 이름은 challange300.py로 합니다.

04 쓰레드로 다중 주기 작업 처리하기

여기서는 쓰레드를 이용하여 다중 주기 작업을 처리해봅니다.

01 도전과제 1

회로 구성

라즈베리파이 보드의 GPIO17번 핀으로 제어하는 빨간색 LED 회로, 라즈베리파이 보드의 GPIO27번 핀으로 제어하는 초록색 LED 회로,

라즈베리파이 보드의 GPIO23번 핀으로 제어하는 파란색 LED 회로를 구성하시오.

프로그래밍

빨간색 LED는 0.7초 주기로 깜빡이고,
초록색 LED는 1.3초 주기로 깜빡이고,
파란색 LED는 1.7초 주기로 깜빡이는 프로그램을 작성하시오.
2 개의 쓰레드를 생성하여
파이썬 쉘이 직접 수행하는 주 루틴은 빨간색 LED를 0.7초 주기로 깜빡이고,
첫 번째 하위 쓰레드는 초록색 LED를 1.3초 주기로 깜빡이고,
두 번째 하위 쓰레드는 파란색 LED를 1.7초 주기로 깜빡이도록 합니다.

파일 이름은 challange410.py로 합니다.

02 도전과제 2

회로 구성

라즈베리파이 보드의 GPIO18번 핀으로 제어하는 빨간색 LED 회로,
라즈베리파이 보드의 GPIO19번 핀으로 제어하는 초록색 LED 회로를 구성하시오.

프로그래밍

빨간색 LED는 0.7초 주기로 밝아지고 어두워지고를 반복하고,
초록색 LED는 1.3초 주기로 밝아지고 어두워지고를 반복하는 프로그램을 작성하시오.
1 개의 쓰레드를 생성하여
파이선 쉘이 직접 수행하는 주 루틴은 빨간색 LED를 0.7초 주기로 밝아지고 어두워지고를 반복하게 하고,
하위 쓰레드는 초록색 LED를 1.3초 주기로 밝아지고 어두워지고를 반복하게 합니다.

파일 이름은 challange420.py로 합니다.

03 도전과제 3

회로 구성

라즈베리파이 보드의 GPIO17번 핀으로 제어하는 빨간색 LED 회로,
라즈베리파이 보드의 GPIO27번 핀으로 제어하는 초록색 LED 회로,
라즈베리파이 보드의 GPIO18번 핀으로 제어하는 파란색 LED 회로,
라즈베리파이 보드의 GPIO19번 핀으로 제어하는 서보 회로를 구성하시오.

프로그래밍

빨간색 LED는 0.6초 주기로 깜빡이고,
초록색 LED는 1.2초 주기로 깜빡이고,
파란색 LED는 2.048초 주기로 1024 단계로 밝아지고 어두워지고를 반복하고,
서보 모터는 3.6초 주기로 0~180도를 반 회전하는 프로그램을 작성하시오.
3 개의 쓰레드를 생성하여
파이선 쉘이 직접 수행하는 주 루틴은 빨간색 LED를 0.6초 주기로 깜빡이고,
첫 번째 하위 쓰레드는 초록색 LED를 1.2초 주기로 깜빡이고,
두 번째 하위 쓰레드는 파란색 LED를 2.048초 주기로 1024 단계로 밝아지고 어두워지고를 반복하게 하고,
세 번째 하위 쓰레드는 서보 모터를 3.6초 주기로 0~180도를 반 회전하게 합니다.

파일 이름은 challange430.py로 합니다.

05 쓰레드로 다중 입력 다중 출력 처리하기

01 도전과제

회로 구성

라즈베리파이 보드의 GPIO22번 핀으로 버튼 입력을 받는 회로,
라즈베리파이 보드의 GPIO17번 핀으로 제어하는 빨간색 LED 회로,
라즈베리파이 보드의 GPIO27번 핀으로 제어하는 초록색 LED 회로,
라즈베리파이 보드의 GPIO23번 핀으로 제어하는 파란색 LED 회로,
라즈베리파이 보드의 GPIO24번 핀으로 제어하는 투명색 LED 회로,
라즈베리파이 보드의 GPIO18번 핀으로 제어하는 노란색 LED 회로,
라즈베리파이 보드의 GPIO19번 핀으로 제어하는 서보 회로를 구성하시오.

프로그래밍

첫 번째 버튼 누름에 빨간색 LED가 켜지고,
두 번째 버튼 누름에 초록색 LED가 켜지고,
세 번째 버튼 누름에 파란색 LED가 켜지고,
네 번째 버튼 누름에 모든 LED가 꺼지도록 하고,
투명색 LED는 1.2초 주기로 깜빡이고,
노란색 LED는 2.048초 주기로 1024 단계로 밝아지고 어두워지고를 반복하고,
키보드 입력을 받아 GPIO19번 핀에 연결된 서보의 각도를 0도, 90도, 180도로 조절하는 프로그램을 작성하시오. 버튼 입력은 인터럽트로 처리합니다.

1 개의 버튼 인터럽트 처리 함수를 등록하고, 3 개의 쓰레드를 생성하여
버튼 인터럽트 처리 함수에서는
 첫 번째 버튼 누름에 빨간색 LED를 켜라는 메시지를 주 쓰레드로 보내고,
 두 번째 버튼 누름에 초록색 LED를 켜라는 메시지를 주 쓰레드로 보내고,
 세 번째 버튼 누름에 파란색 LED를 켜라는 메시지를 주 쓰레드로 보내고,
 네 번째 버튼 누름에 모든 LED를 끄라는 메시지를 주 쓰레드로 보냅니다.
파이썬 쉘이 수행하는 주 루틴은
 버튼 인터럽트 처리함수로부터 메시지를 기다리다가
 빨간색 LED를 켜라는 메시지를 받으면 빨간색 LED를 켜고,
 초록색 LED를 켜라는 메시지를 받으면 초록색 LED를 켜고,
 파란색 LED를 켜라는 메시지를 받으면 파란색 LED를 켜고,
 모든 LED를 끄라는 메시지를 받으면 모든 LED를 끄게 합니다.
첫 번째 하위 쓰레드는 투명색 LED를 1.2초 주기로 깜빡이게 합니다.
두 번째 하위 쓰레드는 노란색 LED를 2.048초 주기로 1024 단계로 밝아지고 어두워지고를 반복하게 합니다.
세 번째 하위 쓰레드는
 키보드 q키를 누르면 서보 모터 0도,
 키보드 w키를 누르면 서보 모터 90도,
 키보드 e키를 누르면 서보 모터 180도로 조절하게 합니다.

파일 이름은 challange500.py로 합니다.