결과 보고서

임베디드응용및실습.
7주차 실습 과제

과 목 명	임베디드응용및실습
학 번	2019161068
이 름	백승진
제 출 일	2024년 10월 17일

1. 버튼 입력받기

① 코드를 추가하여 스위치가 눌렸을 때만 화면에 "click"이 표기되도록 변경

1) 코드

```
import RPi.GPIO as GPIO
import time

# 핀 번호 설정
SW1 = 5

# 초기 설정
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(SW1, GPIO.IN, pull_up_down = GPIO.PUD_DOWN)

try:
    while True:
        current_state = GPIO.input(SW1)
        if current_state == 1:
            print("click")
        else:
            print(current_state)
        time.sleep(0.1)
except KeyboardInterrupt:
    pass

GPIO.cleanup()
```

2) 해석

첫 번째는, 필요한 모듈과 함수를 import 하는 단계이다. 과제를 수행하는 데 필요한 모듈 및 함수는 RPi.GPIO 모듈, time 함수이다.

두 번째는, 사용할 모드에 맞는 핀 번호를 확인 후 명시적으로 정의한다. 과제를 수행하는 데 필요한 모드는 스위치 단 하나로, 번호는 5번이다.

세 번째는, 처음 실행할 때 환경 설정을 하는 단계이다. GPIO.setwarnings는 이전의 결과가 현재 실행에 영향을 줄 위험이 있는 경우에 경고 문구를 출력할지 말지를 결정하는 함수이다. 과제에서는 False로 설정한다. 다음으로 GPIO를 BCM 모드로 사용할 예정이므로, BCM 모드로 설정한다. 마지막으로 스위치는 입력 모드로 설정한다.

네 번째는, 예외 처리를 위해 try-except 문을 사용한다. 여기서 예외 처리는 실행 중 키보드 입력으로 'Ctrl + C'가 들어왔을 경우 try-except 문을 빠져나가는 것이다. 아닌 경우에는 try 문 내부의 블록을 실행한다.

다섯 번째는, try 문 내부의 블록이며, 키보드 입력으로 'Ctrl + C'가 들어오지 않는 한, 계속 동작하도록 True 조건의 while 문을 이용하였다.

여섯 번째는, while 문 내부의 블록으로, SW1이 눌리면 HIGH, 즉 1이 되는 것을 이용하여 해당 값을 저장 후 조건문으로 스위치가 눌렸는지 확인한다.

일곱 번째는, 조건문 내부의 블록으로, 만약 스위치가 눌리면 click을 출력하고, 눌리지 않으면 SW1의 상태값(0)을 출력한다.

마지막으로, 다음 실행에 영향을 주지 않도록, GPIO.cleanup() 함수를 이용하여 종료 및 초기화한다.

3) 결과

출력 결과
"HW1_1" 영상 참조

② 몇 번 스위치가 눌렸는지 확인할 수 있도록 "click x" 등으로 화면 출력

1) 코드

```
import RPi.GPIO as GPIO
import time

# 핀 번호 설정
SW1 = 5

# 스위치 리스트
switches = [SW1]

# 초기 설정
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(SW1, GPIO.IN, pull_up_down = GPIO.PUD_DOWN)

try:
    while True:
        for i in range(len(switches)):
            current_state = GPIO.input(switches[i])
            if current_state:
                print(f"You clicked switch{i + 1}")
            else:
                print(current_state)

        time.sleep(0.1)
except KeyboardInterrupt:
    pass

GPIO.cleanup()
```

2) 해석

첫 번째는, 필요한 모듈과 함수를 import 하는 단계이다. 과제를 수행하는 데 필요한 모듈 및 함수는 RPi.GPIO 모듈, time 함수이다.

두 번째는, 사용할 모드에 맞는 핀 번호를 확인 후 명시적으로 정의한다. 과제를 수행하는 데 필요한 모드는 스위치 단 하나로, 번호는 5번이다.

세 번째는, 스위치 핀 번호를 리스트로 구현한다. 이는 리스트의 인덱스를 이용해서 몇 번 스위치인지 확인하기 위해서이다.

네 번째는, 처음 실행할 때 환경 설정을 하는 단계이다. GPIO.setwarnings는 이전의 결과가 현재 실행에 영향을 줄 위험이 있는 경우에 경고 문구를 출력할지 말지를 결정하는 함수이다. 과제에서는 False로 설정한다. 다음으로 GPIO를 BCM 모드로 사용할 예정이므로, BCM 모드로 설정한다. 마지막으로 스위치는 입력 모드로 설정한다.

다섯 번째는, 예외 처리를 위해 try-except 문을 사용한다. 여기서 예외 처리는 실행 중 키보드 입력으로 'Ctrl + C'가 들어왔을 경우 try-except 문을 빠져나가는 것이다. 아닌 경우에는 try 문 내부의 블록을 실행한다.

여섯 번째는, try 문 내부의 블록이며, 키보드 입력으로 'Ctrl + C'가 들어오지 않는 한, 계속 동작하도록 True 조건의 while 문을 이용하였다.

일곱 번째는, while 문 내부의 블록으로, SW1이 눌리면 HIGH, 즉 1이 되는 것을 이용하여 해당 값을

저장 후 조건문으로 스위치가 눌렸는지 확인한다.
여덟 번째는, 조건문 내부의 블록으로, 만약 스위치가 눌리면 리스트의 인덱스를 이용하여 몇 번 스위치를 클릭했는지 출력하고, 눌리지 않으면 SW1의 상태값(0)을 출력한다.
마지막으로, 다음 실행에 영향을 주지 않도록, GPIO.cleanup() 함수를 이용하여 종료 및 초기화한다.

3) 결과

출력 결과
"HW1_2" 영상 참조

③ 스위치에서 손을 뗄 때는 동작하지 않고, 누를 때만 동작하도록 변경

1) 코드

```
import RPi.GPIO as GPIO
import time

# 핀 번호 설정
SW1 = 5

# 스위치 리스트
switches = [SW1]

# 초기 설정
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(SW1, GPIO.IN, pull_up_down = GPIO.PUD_DOWN)

# 이전 상태를 저장할 리스트 생성 및 초기화
previous_states = []
for sw in switches:
    previous_states.append(sw)

try:
    while True:
        for i in range(len(switches)):
            current_state = GPIO.input(switches[i])
            if current_state and not previous_states[i]:
                print(f"You clicked switch{i + 1}")
            previous_states[i] = current_state

        time.sleep(0.1)
except KeyboardInterrupt:
    pass

GPIO.cleanup()
```

2) 해석

첫 번째는, 필요한 모듈과 함수를 import 하는 단계이다. 과제를 수행하는 데 필요한 모듈 및 함수는 RPi.GPIO 모듈, time 함수이다.

두 번째는, 사용할 모드에 맞는 핀 번호를 확인 후 명시적으로 정의한다. 과제를 수행하는 데 필요한 모드는 스위치 단 하나로, 번호는 5번이다.

세 번째는, 스위치 핀 번호를 리스트로 구현한다. 이는 리스트의 인덱스를 이용해서 몇 번 스위치인지 확인하기 위해서이다.

네 번째는, 처음 실행할 때 환경 설정을 하는 단계이다. GPIO.setwarnings는 이전의 결과가 현재 실행에 영향을 줄 위험이 있는 경우에 경고 문구를 출력할지 말지를 결정하는 함수이다. 과제에서는 False로 설정한다. 다음으로 GPIO를 BCM 모드로 사용할 예정이므로, BCM 모드로 설정한다. 마지막으로 스위치는 입력 모드로 설정한다.

다섯 번째는, 각 스위치의 상태를 저장하는 리스트를 만들고, 초기화한다. 이는 스위치를 클릭했을 때, 전의 상태와 비교해서 처음 눌린 것인지 아니면 계속 누르고 있던 것인지 판단하기 위해서이다.

여섯 번째는, 예외 처리를 위해 try-except 문을 사용한다. 여기서 예외 처리는 실행 중 키보드 입력으로 'Ctrl + C'가 들어왔을 경우 try-except 문을 빠져나가는 것이다. 아닌 경우에는 try 문 내부의 블록을 실행한다.

일곱 번째는, try 문 내부의 블록이며, 키보드 입력으로 'Ctrl + C'가 들어오지 않는 한, 계속 동작하도록 True 조건의 while 문을 이용하였다.

여덟 번째는, while 문 내부의 블록으로, 각 스위치를 계속해서 모두 확인하기 위해 한 번 더 반복문 for를 사용했고, 여기서 반복 횟수는 스위치의 개수이다.

아홉 번째는, for 문 내부의 블록으로, 각 스위치의 상태를 저장하고, 이전 상태와 현재 상태를 비교하기 위해 조건문을 사용한다.

열 번째는, 조건문 내부의 블록으로, 만약 스위치가 눌리면 리스트의 인덱스를 이용하여 몇 번 스위치를 클릭했는지 출력하고, 눌리지 않으면 SW1의 상태값(0)을 출력한다.

마지막으로, 다음 실행에 영향을 주지 않도록, GPIO.cleanup() 함수를 이용하여 종료 및 초기화한다.

3) 결과

출력 결과
"HW1_3" 영상 참조

④ 4개의 스위치 입력받도록 구현.
※ 단, 리스트를 최대한 활용하여 GPIO 전/후 값을 저장한다.
< 참고 출력 화면 >

```
pi@admin:~/Downloads/AI_CAR $ python 3_2_5.py
('SW1 click', 1)
('SW1 click', 2)
('SW2 click', 1)
('SW4 click', 1)
('SW3 click', 1)
('SW1 click', 3)
```

1) 코드

```
import RPi.GPIO as GPIO
import time

# 핀 번호 설정
SW1 = 5
SW2 = 6
SW3 = 13
SW4 = 19

# 스위치 리스트
switches = [SW1, SW2, SW3, SW4]

# 초기 설정
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
for sw in switches:
    GPIO.setup(sw, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

# 카운트 리스트 (1번 버튼부터 4번 버튼 순서)
cntlist = [0, 0, 0, 0]

# 이전 상태를 저장할 리스트 생성 및 초기화
previous_states = []
for sw in switches:
    previous_states.append(sw)

try:
    print(f"click the switch!")
    while True:
        for i in range(len(switches)):
            current_state = GPIO.input(switches[i])
            if current_state and not previous_states[i]:
                cntlist[i] += 1
                print(f"You clicked switch{i + 1}, count = {cntlist}")
```

```
                previous_states[i] = current_state

            time.sleep(0.1)
except KeyboardInterrupt:
    pass

GPIO.cleanup()
```

2) 해석

첫 번째는, 필요한 모듈과 함수를 import 하는 단계이다. 과제를 수행하는 데 필요한 모듈 및 함수는 RPi.GPIO 모듈, time 함수이다.

두 번째는, 사용할 모드에 맞는 핀 번호를 확인 후 명시적으로 정의한다. 과제를 수행하는 데 필요한 모드는 스위치 4개뿐이며, 번호는 5, 6, 13, 19번이다.

세 번째는, 스위치 핀 번호를 리스트로 구현한다. 이는 리스트의 인덱스를 이용해서 몇 번 스위치인지 확인하기 위해서이다.

네 번째는, 처음 실행할 때 환경 설정을 하는 단계이다. GPIO.setwarnings는 이전의 결과가 현재 실행에 영향을 줄 위험이 있는 경우에 경고 문구를 출력할지 말지를 결정하는 함수이다. 과제에서는 False로 설정한다. 다음으로 GPIO를 BCM 모드로 사용할 예정이므로, BCM 모드로 설정한다. 마지막으로 각 스위치를 입력 모드로 설정한다.

다섯 번째는, 각 스위치의 입력 횟수를 카운트해야 하므로, 변수를 선언한다. 이때, 스위치별 입력 횟수를 한 번에 저장하기 위해 list로 선언하여 0으로 초기화한다.

여섯 번째는, 스위치를 누를 때만 한번 동작하도록 스위치별 이전 버튼의 상태 저장용 변수를 선언 후 False로 초기화한다.

일곱 번째는 예외 처리를 위해 try-except 문을 사용한다. 여기서 예외 처리는 실행 중 키보드 입력으로 'Ctrl + C'가 들어왔을 경우 try-except 문을 빠져나가는 것이다. 아닌 경우에는 try 문 내부의 블록을 실행한다.

여덟 번째는, try 문 내부의 블록으로, 먼저 처음 실행 시 어떤 동작을 하면 되는지 안내하기 위해, "click the switch!" 문장을 shell 창에 출력한다. 다음으로 키보드 입력으로 'Ctrl + C'가 들어오지 않는 한, 계속 동작하도록 True 조건의 while 문을 이용하였다.

아홉 번째는, while 문 내부의 블록으로, 이전의 클릭 상태가 false이면서 1(True)로 변경되었을 때 즉, 스위치가 눌렸을 때만 동작하도록 조건문으로 확인하였다. 동작 내용은 각 버튼의 카운트를 1 증가시키고, shell 창에 어떤 버튼이 눌렸는지 출력한다. 동작이 완료되면, 버튼 상태를 다시 현재 상태로 업데이트한다. 이때, 반복문에서 너무 빠르게 동작하면 충돌 문제가 일어날 수 있어 time.sleep() 함수를 이용하여 delay를 주었다.

마지막으로, 다음 실행에 영향을 주지 않도록, GPIO.cleanup() 함수를 이용하여 초기화한다.

3) 결과

출력 결과
"HW1_4" 영상 참조

2. 음계 출력하기

① "도레미파솔라시도" 음계를 출력

1) 코드

```
import RPi.GPIO as GPIO
import time

# 핀번호 설정
BUZZER = 12

# 초기 설정
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(BUZZER, GPIO.OUT)

# 음별 주파수
notes_dict = {
    'Do': 261.63,
    'Di' : 277.18,
    'Re': 293.66,
    'Ri' : 311.13,
    'Mi': 329.63,
    'Fa': 349.23,
    'Fi' : 369.99,
    'Sol': 392.00,
    'Si' : 415.30,
    'La': 440.00,
    'Li' : 466.16,
    'Ti': 493.88,
    '2Do': 523.25,
    '2Di' : 554.37,
    '2Re' : 587.33,
    '2Ri' : 622.25,
    '2Mi' : 659.25,
    '2Fa' : 698.46,
    '2Fi' : 739.99,
    '2Sol' : 783.99
}

# 과제에서 사용할 음
scale = ['Do', 'Re', 'Mi', 'Fa', 'Sol', 'La', 'Ti', '2Do']

try:
    p = GPIO.PWM(BUZZER, notes_dict[scale[0]])
    p.start(50)

    # 음계 리스트에서 하나씩 출력
    for note in scale:
        p.ChangeFrequency(notes_dict[note])
        time.sleep(0.2)
except KeyboardInterrupt:
```

```
    pass

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

2) 해석

첫 번째는, 필요한 모듈과 함수를 import 하는 단계이다. 과제를 수행하는 데 필요한 모듈 및 함수는 RPi.GPIO 모듈, time 함수이다.

두 번째는, 사용할 모드에 맞는 핀 번호를 확인 후 명시적으로 정의한다. 과제를 수행하는 데 필요한 모드는 Buzzer뿐이며, 번호는 12번이다.

세 번째는, 처음 실행할 때 환경 설정을 하는 단계이다. GPIO.setwarnings는 이전의 결과가 현재 실행에 영향을 줄 위험이 있는 경우에 경고 문구를 출력할지 말지를 결정하는 함수이다. 과제에서는 False로 설정한다. 다음으로 GPIO를 BCM 모드로 사용할 예정이므로, BCM 모드로 설정한다. 마지막으로 Buzzer를 출력 모드로 설정한다.

네 번째는, 필요한 음과 그에 맞는 주파수를 dictionary로 정의한다. '도레미파솔라시도'를 출력하기에는 배열 요소가 많지만, 추후 더 필요할 수도 있으므로 많이 정의하였다. 최저 음은 '4옥타브 도'이며, 최고 음은 '5옥타브 솔'이다. 이때, 편의상 옥타브는 낮춰서 표기하였다.

다섯 번째는, 과제에서 사용할 음계를 list로 정의하였다. 이번 음계는 '도레미파솔라시도'이다.

여섯 번째는, 예외 처리를 위해 try-except 문을 사용한다. 여기서 예외 처리는 실행 중 키보드 입력으로 'Ctrl + C'가 들어왔을 경우 try-except 문을 빠져나가는 것이다. 아닌 경우에는 try 문 내부의 블록을 실행한다.

일곱 번째는, try 문 내부의 블록으로, 먼저 Buzzer의 Frequency와 Duty Cycle을 설정한다. 과제에서 주파수는 첫 음, 펄스폭은 50%로 설정하였다. 다음으로 키보드 입력으로 'Ctrl + C'가 들어오지 않는 한, 사전에 정의한 음계를 하나씩 순차적으로 출력하도록 for 문을 이용하였다.

여덟 번째는, for 문 내부의 블록으로, 사전에 정의한 음계의 음들에 맞는 주파수로 하나씩 변경해가며 0.2초간 출력하도록 한다.

마지막으로, 다음 실행에 영향을 주지 않도록, p.stop() 함수와 GPIO.cleanup() 함수를 이용하여 종료 및 초기화한다.

3) 결과

출력음
"HW2_1" 영상 참조

② 나만의 경적 소리를 구현

1) 코드

```
import RPi.GPIO as GPIO
import time

# 핀 번호 설정
BUZZER = 12

# 초기 설정
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(BUZZER, GPIO.OUT)

# 음별 주파수
notes_dict = {
    'Do': 261.63,
    'Di' : 277.18,
    'Re': 293.66,
    'Ri' : 311.13,
    'Mi': 329.63,
    'Fa': 349.23,
    'Fi' : 369.99,
    'Sol': 392.00,
    'Si' : 415.30,
    'La': 440.00,
    'Li' : 466.16,
    'Ti': 493.88,
    '2Do': 523.25,
    '2Di' : 554.37,
    '2Re' : 587.33,
    '2Ri' : 622.25,
    '2Mi' : 659.25,
    '2Fa' : 698.46,
    '2Fi' : 739.99,
    '2Sol' : 783.99
}

# 호출한 음 출력
def play_ThatsHilarious():
    # 계이름
    Thats_scale = [
        ('Di', 0.2), ('Ri', 0.3), ('Fi', 0.3), ('Li', 0.3), ('Si', 0.3), ('Fi', 0.3), ('Li', 0.3), ('Si', 0.3), ('Fi', 0.3), ('Fi', 0.3), ('Fi', 0.3),
        ('Li', 0.3), ('Si', 0.3), ('Fi', 0.3), ('Si', 0.3), ('Di', 0.3), ('Di', 0.3), ('Fi', 0.3), ('Ti', 0.3), ('Li', 0.5), ('Si', 0.2), ('Fi', 1.1),
        ('Di', 0.3), ('Si', 0.3), ('Fi', 0.5), ('Fa', 0.2), ('Ri', 1.1),
        ('Fi', 0.3), ('Fi', 0.3), ('Li', 0.3), ('Si', 0.3), ('Fi', 0.3), ('Li', 0.3), ('Si', 0.3), ('Fi', 0.3), ('Fi', 0.3), ('Fi', 0.3),
        ('Li', 0.3), ('Si', 0.3), ('Fi', 0.3), ('Si', 0.3), ('Di', 0.3), ('Di', 0.3), ('Fi', 0.3), ('Ti', 0.3), ('Li', 0.5), ('Si', 0.2), ('Fi', 1.1),
        ('Di', 0.3), ('Si', 0.3), ('Fi', 0.5), ('Fa', 0.2), ('Ri', 1.1)
    ]

    # 첫 음 재생 전에 바로 주파수를 설정하고 잠깐 대기
```

```
    first_note, first_duration = Thats_scale[0]
    p.ChangeFrequency(notes_dict[first_note])
    p.ChangeDutyCycle(1)
    time.sleep(0.1)  # 첫 음이 건너뛰지 않도록 짧은 대기 시간 추가

    # 나머지 음 재생
    for note, duration in Thats_scale[1:]:
        p.ChangeFrequency(notes_dict[note])
        time.sleep(duration)

    # 연주 종료(재시작 대기)
    p.ChangeDutyCycle(0)

try:
    p = GPIO.PWM(BUZZER, 1)
    p.start(0)

    play_ThatsHilarious()
    time.sleep(0.2)
except KeyboardInterrupt:
    pass

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

2) 해석

첫 번째는, 필요한 모듈과 함수를 import 하는 단계이다. 과제를 수행하는 데 필요한 모듈 및 함수는 RPi.GPIO 모듈, time 함수이다.

두 번째는, 사용할 모드에 맞는 핀 번호를 확인 후 명시적으로 정의한다. 과제를 수행하는 데 필요한 모드는 Buzzer뿐이며, 번호는 12번이다.

세 번째는, 처음 실행할 때 환경 설정을 하는 단계이다. GPIO.setwarnings는 이전의 결과가 현재 실행에 영향을 줄 위험이 있는 경우에 경고 문구를 출력할지 말지를 결정하는 함수이다. 과제에서는 False로 설정한다. 다음으로 GPIO를 BCM 모드로 사용할 예정이므로, BCM 모드로 설정한다. 마지막으로 Buzzer를 출력 모드로 설정한다.

네 번째는, 필요한 음과 그에 맞는 주파수를 dictionary로 정의한다. '도레미파솔라시도'를 출력하기에는 배열 요소가 많지만, 추후 더 필요할 수도 있으므로 많이 정의하였다. 최저 음은 '4옥타브 도'이며, 최고 음은 '5옥타브 솔'이다. 이때, 편의상 옥타브는 낮춰서 표기하였다.

다섯 번째는, 과제에서 사용할 음계를 재생하는 함수 play_ThatsHilarious를 정의하였다. 노래는 Charlie Puth - That's Hilarious이다.

여섯 번째는, 함수 내부의 블록으로, 먼저 재생할 노래의 음을 리스트로 구현한다. 이때, 음과 함께 각 음의 지속 시간을 튜플로 묶는다. 이후, 첫 번째 음을 먼저 재생하는데, 이는 알고리즘에 의해 첫 음이 생략되지 않게 하기 위해서이다. 과제의 알고리즘에서는 음악을 재생하지 않을 때는 DutyCycle을 0으로 설정하여 어떤 잡음도 들리지 않게 구성하였는데, 이렇게 ChangeDutyCycle로 변화를 줄 때, 첫 음이 씹히는 현상이 발생하였다. 이를 위해, 첫 음만 따로 재생 후 time.sleep으로 대기 시간을 추가함으로써 해결하였다. 첫 음을 재생 후에는 나머지 음을 지속시간 만큼 재생하고, 음악이 종료되면 다시 DutyCycle을 0으로 설정한다.

일곱 번째는, 예외 처리를 위해 try-except 문을 사용한다. 여기서 예외 처리는 실행 중 키보드 입력으로 'Ctrl + C'가 들어왔을 경우 try-except 문을 빠져나가는 것이다. 아닌 경우에는 try 문 내부의 블록을 실행한다.

여덟 번째는, try 문 내부의 블록으로, 먼저 Buzzer의 Frequency와 Duty Cycle을 설정한다. 실행 시 어떤 동작도 하면 안 되기 때문에, 초깃값은 각각 1과 0이다. 다음으로, 실행 후 바로 음악이 출력되도록 play_ThatsHilarious 함수를 호출한다.

마지막으로, 다음 실행에 영향을 주지 않도록, p.stop() 함수와 GPIO.cleanup() 함수를 이용하여 종료 및 초기화한다.

3) 결과

출력음
"HW2_2" 영상 참조

③ 스위치를 한 번 누르면 경적 소리가 나도록 구현

1) 코드

```
import RPi.GPIO as GPIO
import time

# 핀 번호 설정
SW1 = 5
SW2 = 6
SW3 = 13
SW4 = 19
BUZZER = 12

# 스위치 리스트
switches = [SW1, SW2, SW3, SW4]

# 초기 설정
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(BUZZER, GPIO.OUT)
for sw in switches:
    GPIO.setup(sw, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

# 버튼 클릭 상태
prev_sw1 = False
prev_sw2 = False
prev_sw3 = False
prev_sw4 = False

# 음별 주파수
notes_dict = {
    'Do': 261.63,
    'Di' : 277.18,
    'Re': 293.66,
    'Ri' : 311.13,
    'Mi': 329.63,
    'Fa': 349.23,
    'Fi' : 369.99,
    'Sol': 392.00,
    'Si' : 415.30,
    'La': 440.00,
    'Li' : 466.16,
    'Ti': 493.88,
    '2Do': 523.25,
    '2Di' : 554.37,
    '2Re' : 587.33,
```

```
    '2Ri' : 622.25,
    '2Mi' : 659.25,
    '2Fa' : 698.46,
    '2Fi' : 739.99,
    '2Sol' : 783.99,
    '2Si' : 830.61,
    '2La' : 880.00,
    '2Li' : 932.33,
    '2Ti' : 987.77
}

# 1번 버튼 클릭 시 재생할 노래 (Yuuri - Leo)
def play_Leo():
    # 계이름
    leo_scale = [
        ('Do', 0.3), ('Fa', 0.3), ('Mi', 0.3), ('Fa', 0.3), ('Sol', 0.3), ('Sol', 0.6),
        ('Do', 0.3), ('Do', 0.3), ('Sol', 0.3), ('Fa', 0.3), ('Sol', 0.3), ('La', 0.3), ('La', 0.6),
        ('La', 0.3), ('Sol', 0.3), ('Fa', 0.3), ('Mi', 0.3), ('Fa', 0.3), ('La', 0.3), ('La', 0.3), ('Sol', 0.3),
        ('Re', 0.3), ('Mi', 0.3), ('Fa', 0.3), ('Mi', 0.3), ('Mi', 0.3), ('Fa', 0.3), ('Fa', 0.3)
        ]

    # 첫 음 재생 전에 바로 주파수를 설정하고 잠깐 대기
    first_note, first_dulation = leo_scale[0]
    p.ChangeFrequency(notes_dict[first_note])
    p.ChangeDutyCycle(1)
    time.sleep(0.2)  # 첫 음이 건너뛰지 않도록 짧은 대기 시간 추가

    # 연주 시작
    for note, dulation in leo_scale[1:]:
        p.ChangeFrequency(notes_dict[note])
        time.sleep(dulation)

    # 연주 종료(재시작 대기)
    p.ChangeDutyCycle(0)

# 2번 버튼 클릭 시 재생할 노래 (Imase - Night Dancer)
def play_NightDancer():
    # 계이름
    night_scale = [
        ('2Sol', 0.4), ('2Sol', 0.25), ('2Fi', 0.25), ('2Fi', 0.35), ('2Re', 0.25), ('2Do', 0.25), ('Li', 0.25),
        ('2Do', 0.25), ('Li', 0.25), ('2Do', 0.25), ('Li', 0.25), ('2Re', 0.5),
        ('2Do', 0.35), ('2Do', 0.25), ('Li', 0.25), ('2Do', 0.25), ('2Re', 0.25),
        ('2Fa', 0.25), ('2Re', 0.25), ('2Do', 0.25), ('Li', 0.25), ('2Do', 0.25), ('2Re', 0.25), ('2Re', 0.4)
        ]

    # 첫 음 재생 전에 바로 주파수를 설정하고 잠깐 대기
```

```
    first_note, first_dulation = night_scale[0]
    p.ChangeFrequency(notes_dict[first_note])
    p.ChangeDutyCycle(1)
    time.sleep(0.15)
    p.ChangeDutyCycle(0)
    time.sleep(0.3)
    p.ChangeDutyCycle(1)
    time.sleep(0.1)  # 첫 음이 건너뛰지 않도록 짧은 대기 시간 추가

    # 연주
    for note, dulation in night_scale[1:11]:
        p.ChangeFrequency(notes_dict[note])
        time.sleep(dulation)

    # 쉼표
    p.ChangeFrequency(notes_dict[night_scale[11][0]])
    time.sleep(0.3)
    p.ChangeDutyCycle(0)
    time.sleep(0.3)
    p.ChangeDutyCycle(1)
    time.sleep(0.2)  # 첫 음이 건너뛰지 않도록 짧은 대기 시간 추가

    # 연주
    for note, dulation in night_scale[12:]:
        p.ChangeFrequency(notes_dict[note])
        time.sleep(dulation)

    # 연주 종료(재시작 대기)
    p.ChangeDutyCycle(0)

# 3번 버튼 클릭 시 재생할 노래 (Yuuri - Betelgeuse)
def play_Betelgeuse():
    # 계이름
    bet_scale = [
        ('Do', 0.3), ('La', 0.2), ('Sol', 0.3), ('Fa', 0.2),
        ('Fa', 0.3), ('2Do', 0.2), ('La', 0.3), ('Sol', 0.4), ('Fa', 0.2), ('Fa', 0.3), ('2Do', 0.2), ('La', 0.3), ('Sol',
0.4), ('Fa', 0.2),
        ('Mi', 0.3), ('Fa', 0.3), ('Sol', 0.3), ('Fa', 0.7),
        ('Do', 0.3), ('Mi', 0.3), ('Fa', 0.3), ('Fa', 0.3), ('2Do', 0.2), ('La', 0.3), ('Sol', 0.4),
        ('Fa', 0.2), ('Fa', 0.3), ('2Do', 0.2), ('La', 0.3), ('Sol', 0.4),
        ('Fa', 0.2), ('Li', 0.3), ('La', 0.3), ('Sol', 0.3), ('Mi', 0.3), ('Fa', 0.3)
        ]

    # 첫 음 재생 전에 바로 주파수를 설정하고 잠깐 대기
    first_note, first_dulation = bet_scale[0]
    p.ChangeFrequency(notes_dict[first_note])
```

```
    p.ChangeDutyCycle(1)
    time.sleep(0.2)  # 첫 음이 건너뛰지 않도록 짧은 대기 시간 추가

    # 연주 시작
    for note, dulation in bet_scale[1:]:
        p.ChangeFrequency(notes_dict[note])
        time.sleep(dulation)

    # 연주 종료(재시작 대기)
    p.ChangeDutyCycle(0)

# 4번 버튼 클릭 시 재생할 노래 (Yonez Kenshi - Lemon)
def play_Lemon():
    # 계이름
    lemon_scale = [
        ('2Di', 0.2), ('2Ri', 0.15), ('2Di', 0.15), ('Ti', 0.15), ('Si', 0.2), ('Ti', 0.5), ('2Ri', 0.2), ('2Fi', 0.5), ('2Di',
0.2), ('Ti', 0.5),
        ('2Di', 0.2), ('2Ri', 0.15), ('2Di', 0.15), ('Ti', 0.15), ('Si', 0.2), ('Ti', 0.5), ('2Ri', 0.2), ('2Fi', 0.5), ('2Di',
0.2), ('Ti', 0.5),
        ('2Di', 0.2), ('2Ri', 0.15), ('2Di', 0.15), ('Ti', 0.15), ('Si', 0.2), ('Ti', 0.5), ('2Ri', 0.2), ('2Fi', 0.5),
        ('2Si', 0.2), ('2Fi', 0.5), ('2Fi', 0.2), ('2Ti', 0.5), ('2Li', 0.2), ('2Fi', 0.5), ('2Ri', 0.2), ('2Fi', 0.5), ('2Di',
1.0)
        ]

    # 첫 음 재생 전에 바로 주파수를 설정하고 잠깐 대기
    first_note, first_dulation = lemon_scale[0]
    p.ChangeFrequency(notes_dict[first_note])
    p.ChangeDutyCycle(1)
    time.sleep(0.2)  # 첫 음이 건너뛰지 않도록 짧은 대기 시간 추가

    # 연주 시작
    for note, dulation in lemon_scale[1:]:
        p.ChangeFrequency(notes_dict[note])
        time.sleep(dulation)

    # 연주 종료(재시작 대기)
    p.ChangeDutyCycle(0)

try:
    p = GPIO.PWM(BUZZER, 1)
    p.start(0)
    p.ChangeDutyCycle(0)

    while True:
        # None click 상태에서 클릭 시, 버튼별 노래 재생 함수 호출
        if GPIO.input(SW1) and not prev_sw1:
```

```
            play_Leo()
        elif GPIO.input(SW2) and not prev_sw2:
            play_NightDancer()
        elif GPIO.input(SW3) and not prev_sw3:
            play_Betelgeuse()
        elif GPIO.input(SW4) and not prev_sw4:
            play_Lemon()

        # 버튼 상태 업데이트
        prev_sw1 = GPIO.input(SW1)
        prev_sw2 = GPIO.input(SW2)
        prev_sw3 = GPIO.input(SW3)
        prev_sw4 = GPIO.input(SW4)

        time.sleep(0.1)
except KeyboardInterrupt:
    pass

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

2) 해석

첫 번째는, 필요한 모듈과 함수를 import 하는 단계이다. 과제를 수행하는 데 필요한 모듈 및 함수는 RPi.GPIO 모듈, time 함수이다.

두 번째는, 사용할 모드에 맞는 핀 번호를 확인 후 명시적으로 정의한다. 과제를 수행하는 데 필요한 모드는 스위치 4개와 Buzzer로, 번호는 각각 5, 6, 13, 19, 12번이다.

세 번째는, 스위치 핀 번호를 리스트로 구현한다. 이는 리스트의 인덱스를 이용해서 몇 번 스위치인지 확인하기 위해서이다.

네 번째는, 처음 실행할 때 환경 설정을 하는 단계이다. GPIO.setwarnings는 이전의 결과가 현재 실행에 영향을 줄 위험이 있는 경우에 경고 문구를 출력할지 말지를 결정하는 함수이다. 과제에서는 False로 설정한다. 다음으로 GPIO를 BCM 모드로 사용할 예정이므로, BCM 모드로 설정한다. 마지막으로 스위치는 입력 모드로, Buzzer는 출력 모드로 설정한다.

다섯 번째는, 각 스위치의 상태를 저장하는 리스트를 만들고, 초기화한다. 이는 스위치를 클릭했을 때, 전의 상태와 비교해서 처음 눌린 것인지 아니면 계속 누르고 있던 것인지 판단하기 위해서이다.

여섯 번째는, 필요한 음과 그에 맞는 주파수를 dictionary로 정의한다. 최저 음은 '4옥타브 도'이며, 최고 음은 '5옥타브 시'이다. 이때, 편의상 옥타브는 낮춰서 표기하였다.

일곱 번째는, 스위치별로 각자 다른 노래를 저장하고, 이를 출력하는 함수를 정의하였다. 총 4개이며, 노래는 순서대로 'Yuuri - Leo', 'Imase - Night Dancer', 'Yuuri - Betelgeuse', 'Yonez Kenshi - Lemon'이다.

여덟 번째는, 함수 내부의 블록으로, 먼저 재생할 노래의 음을 리스트로 구현한다. 이때, 음과 함께 각 음의 지속 시간을 튜플로 묶는다. 이후, 첫 번째 음을 먼저 재생하는데, 이는 알고리즘에 의해 첫 음이 생략되지 않게 하기 위해서이다. 과제의 알고리즘에서는 음악을 재생하지 않을 때는 DutyCycle을 0으로 설정하여 어떤 잡음도 들리지 않게 구성하였는데, 이렇게 ChangeDutyCycle로 변화를 줄 때, 첫 음이 씹히는 현상이 발생하였다. 이를 위해, 첫 음만 따로 재생 후 time.sleep으로 대기 시간을 추가함으로써 해결하였다. 첫 음을 재생 후에는 나머지 음을 지속시간 만큼 재생하고, 음악이 종료되면 다시 DutyCycle을 0으로 설정한다. 추가로, 특정 곡은 중간에 끊겨야 원곡과 유사한 느낌이 들기에, time.sleep() 함수를 이용하여 적절히 스타카토를 넣어주기도 하였다.

아홉 번째는, 예외 처리를 위해 try-except 문을 사용한다. 여기서 예외 처리는 실행 중 키보드 입력으로 'Ctrl + C'가 들어왔을 경우 try-except 문을 빠져나가는 것이다. 아닌 경우에는 try 문 내부의 블록을 실행한다.

열 번째는, try 문 내부의 블록으로, 먼저 Buzzer의 Frequency와 Duty Cycle을 설정한다. 여기서 처음에는 아무 소리도 나지 않도록 주파수는 1, 펄스폭은 0%로 설정하였다. 다음으로 키보드 입력으로 'Ctrl + C'가 들어오지 않는 한, 계속 동작하도록 True 조건의 while 문을 이용하였다.

열한 번째는, while 문 내부의 블록으로, 이전의 클릭 상태가 false이면서 1(True)로 변경되었을 때 즉, 각 스위치가 눌렸을 때만 동작하도록 조건문으로 확인하였다. 동작 내용은 각 스위치에 대응되는 노래를 재생하도록 함수를 호출하는 것이다. 한 곡이 끝나면, 버튼 상태를 다시 현재 상태로 업데이트한다. 이때, 반복문에서 너무 빠르게 동작하면 충돌 문제가 일어날 수 있어 time.sleep() 함수를 이용하여 delay를 주었다.

마지막으로, 다음 실행에 영향을 주지 않도록, p.stop() 함수와 GPIO.cleanup() 함수를 이용하여 종료 및 초기화한다.

3) 결과

출력음
"HW2_3" 영상 참조 '1번 스위치' : 'Yuuri - Leo' '2번 스위치' : 'Imase - Night Dancer' '3번 스위치' : 'Yuuri - Betelgeuse' '4번 스위치' : 'Yonez Kenshi - Lemon'

④ 스위치 4개를 사용하여 나만의 음악을 연주

1) 코드

```
import RPi.GPIO as GPIO
import time

# 핀 번호 설정
SW1 = 5
SW2 = 6
SW3 = 13
SW4 = 19
BUZZER = 12

# 스위치 리스트
switches = [SW1, SW2, SW3, SW4]

# 초기 설정
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(BUZZER, GPIO.OUT)
GPIO.setup(BUZZER, GPIO.OUT)
for sw in switches:
    GPIO.setup(sw, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

# 음별 주파수
notes_dict = {
    'Do': 261.63,
    'Di' : 277.18,
    'Re': 293.66,
    'Ri' : 311.13,
    'Mi': 329.63,
    'Fa': 349.23,
    'Fi' : 369.99,
    'Sol': 392.00,
    'Si' : 415.30,
    'La': 440.00,
    'Li' : 466.16,
    'Ti': 493.88,
    '2Do': 523.25,
    '2Di' : 554.37,
    '2Re' : 587.33,
    '2Ri' : 622.25,
    '2Mi' : 659.25,
    '2Fa' : 698.46,
    '2Fi' : 739.99,
    '2Sol' : 783.99
}

# 과제에서 사용할 음
scale = ['2Di', '2Li', 'Si', '2Fa']
```

```
# 현재 음
current_note = None

# 호출한 음 출력
def play_sound(note):
    # 전역 변수값 수정
    global current_note

    # 이전 음과 다른 음을 호출했다면
    if note != current_note:
        p.ChangeDutyCycle(1)
        # 현재 음 업데이트
        current_note = note
        # 현재 음 출력
        p.ChangeFrequency(notes_dict[current_note])

try:
    p = GPIO.PWM(BUZZER, 1)
    p.start(0)
    while True:
        # 버튼별 특정 음 재생을 위해 함수 호출
        if GPIO.input(SW1):
            play_sound('2Di')
        elif GPIO.input(SW2):
            play_sound('2Ri')
        elif GPIO.input(SW3):
            play_sound('Si')
        elif GPIO.input(SW4):
            play_sound('2Fa')
        else:
            # 만약 버튼을 누르지 않은 상태라면 음소거
            if current_note is not None:
                p.ChangeDutyCycle(0)
                current_note = None
except KeyboardInterrupt:
    pass

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

2) 해석

첫 번째는, 필요한 모듈과 함수를 import 하는 단계이다. 과제를 수행하는 데 필요한 모듈 및 함수는 RPi.GPIO 모듈, time 함수이다.

두 번째는, 사용할 모드에 맞는 핀 번호를 확인 후 명시적으로 정의한다. 과제를 수행하는 데 필요한 모드는 스위치 4개와 Buzzer로, 번호는 각각 5, 6, 13, 19, 12번이다.

세 번째는, 스위치 핀 번호를 리스트로 구현한다. 이는 리스트의 인덱스를 이용해서 몇 번 스위치인지 확인하기 위해서이다.

네 번째는, 처음 실행할 때 환경 설정을 하는 단계이다. GPIO.setwarnings는 이전의 결과가 현재 실행에 영향을 줄 위험이 있는 경우에 경고 문구를 출력할지 말지를 결정하는 함수이다. 과제에서는 False로 설정한다. 다음으로 GPIO를 BCM 모드로 사용할 예정이므로, BCM 모드로 설정한다. 마지막으로 스위치는

입력 모드로, Buzzer는 출력 모드로 설정한다.

다섯 번째는, 필요한 음과 그에 맞는 주파수를 dictionary로 정의한다. 최저 음은 '4옥타브 도'이며, 최고 음은 '5옥타브 솔'이다. 이때, 편의상 옥타브는 낮춰서 표기하였다.

여섯 번째는, 과제에서 사용할 음계를 list로 정의하였다. 이번 음계는 '5옥타브 도, 5옥타브 레, 시, 5옥타브 파'이다. 이 음들은 'Coldplay - Viva La Vida'의 코드 진행에 핵심인 음이다.

일곱 번째는, 여러 스위치를 누르더라도 한 번에 하나의 음만 재생되도록 현재 음을 저장하기 위한 변수를 선언하여 None으로 초기화한다.

여덟 번째는, 스위치에 맞는 음을 출력하는 함수를 정의한다. 앞서 현재 음을 저장하는 전역 변수를 사용하기 위해 선언하고, 만약 새로운 음을 호출했다면, 해당 음을 전역 변수에 저장하고, 재생한다.

아홉 번째는, 예외 처리를 위해 try-except 문을 사용한다. 여기서 예외 처리는 실행 중 키보드 입력으로 'Ctrl + C'가 들어왔을 경우 try-except 문을 빠져나가는 것이다. 아닌 경우에는 try 문 내부의 블록을 실행한다.

열 번째는, try 문 내부의 블록으로, 먼저 Buzzer의 Frequency와 Duty Cycle을 설정한다. 여기서 처음에는 아무 소리도 나지 않도록 주파수는 1, 펄스폭은 0%로 설정하였다. 다음으로 키보드 입력으로 'Ctrl + C'가 들어오지 않는 한, 계속 동작하도록 True 조건의 while 문을 이용하였다.

열한 번째는, while 문 내부의 블록으로, 각 스위치가 눌렸을 때, 해당 스위치에 맞는 음을 재생하도록 하는 함수를 호출한다. 이때, 어떤 버튼도 누르지 않고 있다면 펄스폭을 0%로 조정해 음을 소거한다. 보통 여기서 time.sleep()를 사용하였으나, 이번 과제에서는 스위치를 누르는 즉시, 계속해서 연주가 진행될 수 있도록 delay를 적용하지 않았다.

마지막으로, 다음 실행에 영향을 주지 않도록, p.stop() 함수와 GPIO.cleanup() 함수를 이용하여 종료 및 초기화한다.

3) 결과

출력음
"HW2_4" 영상 참조

3. 문제 2에서 압축 해제된 파일들을 IFC181_re.tar로 압축하시오.

① 오른쪽 모터 부분의 코드를 추가하여 동작과 정지를 일정하게 반복
※ 방향 : 정방향
※ 속도 : 50%

1) 코드

```
import RPi.GPIO as GPIO
import time

# 핀 번호 설정
PWMA = 18
PWMB = 23
AIN1 = 22
AIN2 = 27
BIN1 = 25
BIN2 = 24

# 초기 설정
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(PWMA, GPIO.OUT)
GPIO.setup(PWMB, GPIO.OUT)
GPIO.setup(AIN1, GPIO.OUT)
GPIO.setup(AIN2, GPIO.OUT)
GPIO.setup(BIN1, GPIO.OUT)
GPIO.setup(BIN2, GPIO.OUT)

# 출력 주파수 설정
L_Motor = GPIO.PWM(PWMA, 250)
R_Motor = GPIO.PWM(PWMB, 250)

# 초기 정지 상태
L_Motor.start(0)
R_Motor.start(0)

try:
    while True:
        # 1초 간 전진
        GPIO.output(AIN1, 0)
        GPIO.output(AIN2, 1)
        GPIO.output(BIN1, 0)
        GPIO.output(BIN2, 1)
        L_Motor.ChangeDutyCycle(50)
        R_Motor.ChangeDutyCycle(50)
        time.sleep(1.0)
```

```
        # 1초 간 정지
        GPIO.output(AIN1, 0)
        GPIO.output(AIN2, 1)
        GPIO.output(BIN1, 0)
        GPIO.output(BIN2, 1)
        L_Motor.ChangeDutyCycle(0)
        R_Motor.ChangeDutyCycle(0)
        time.sleep(1.0)
except KeyboardInterrupt:
    pass

GPIO.cleanup()
```

2) 해석

첫 번째는, 필요한 모듈과 함수를 import 하는 단계이다. 과제를 수행하는 데 필요한 모듈 및 함수는 RPi.GPIO 모듈, time 함수이다.

두 번째는, 사용할 모드에 맞는 핀 번호를 확인 후 명시적으로 정의한다. 과제를 수행하는 데 필요한 모드는 모터 속도 2개와 모터 방향 4개로, 번호는 각각 18, 23, 22, 27, 26, 24번이다.

세 번째는, 처음 실행할 때 환경 설정을 하는 단계이다. GPIO.setwarnings는 이전의 결과가 현재 실행에 영향을 줄 위험이 있는 경우에 경고 문구를 출력할지 말지를 결정하는 함수이다. 과제에서는 False로 설정한다. 다음으로 GPIO를 BCM 모드로 사용할 예정이므로, BCM 모드로 설정한다. 마지막으로 모터의 속도와 방향은 모두 출력 모드로 설정한다.

네 번째는, 모터의 주파수를 설정한다. 좌우 모두 250으로 설정하였다.

다섯 번째는, 초기 상태를 정지 상태로 설정한다.

여섯 번째는, 예외 처리를 위해 try-except 문을 사용한다. 여기서 예외 처리는 실행 중 키보드 입력으로 'Ctrl + C'가 들어왔을 경우 try-except 문을 빠져나가는 것이다. 아닌 경우에는 try 문 내부의 블록을 실행한다.

일곱 번째는, try 문 내부의 블록으로, 먼저 'Ctrl + C'가 들어오지 않는 한, 계속 동작하도록 True 조건의 while 문을 이용하였다.

여덟 번째는, while 문 내부의 블록으로, 각 모터의 방향을 앞으로 설정하여 속도 50만큼 1초 동안 전진한 후, 속도 0만큼 1초 동안 정지한다. 이는 반복문 내부에 있으므로, 전진 및 정지를 반복한다.

마지막으로, 다음 실행에 영향을 주지 않도록, p.stop() 함수와 GPIO.cleanup() 함수를 이용하여 종료 및 초기화한다.

3) 결과

동작 결과
"HW3_1" 영상 참조

② 스위치를 입력받아 자동차를 조종
※ SW1 : 전진
※ SW2 : 우회전
※ SW3 : 좌회전
※ SW4 : 후진

1) 코드

```
import RPi.GPIO as GPIO
import time

# 핀 번호 설정
SW1 = 5
SW2 = 6
SW3 = 13
SW4 = 19
PWMA = 18
PWMB = 23
AIN1 = 22
AIN2 = 27
BIN1 = 25
BIN2 = 24

# 초기 설정
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(SW1, GPIO.IN, pull_up_down = GPIO.PUD_DOWN)
GPIO.setup(SW2, GPIO.IN, pull_up_down = GPIO.PUD_DOWN)
GPIO.setup(SW3, GPIO.IN, pull_up_down = GPIO.PUD_DOWN)
GPIO.setup(SW4, GPIO.IN, pull_up_down = GPIO.PUD_DOWN)
GPIO.setup(PWMA, GPIO.OUT)
GPIO.setup(PWMB, GPIO.OUT)
GPIO.setup(AIN1, GPIO.OUT)
GPIO.setup(AIN2, GPIO.OUT)
GPIO.setup(BIN1, GPIO.OUT)
GPIO.setup(BIN2, GPIO.OUT)

# 출력 주파수 설정
L_Motor = GPIO.PWM(PWMA, 250)
R_Motor = GPIO.PWM(PWMB, 250)

# 초기 정지 상태
L_Motor.start(0)
R_Motor.start(0)

# 이전 버튼 상태
```

```
prev_btn = None

try:
    while True:
        if GPIO.input(SW1):
            if 'sw1' != prev_btn:
                print("You clicked switch1")
                # 현재 버튼 업데이트
                prev_btn = 'sw1'

                # 방향 및 속도 설정 후 출력
                GPIO.output(AIN1, 0)
                GPIO.output(AIN2, 1)
                GPIO.output(BIN1, 0)
                GPIO.output(BIN2, 1)
                L_Motor.ChangeDutyCycle(50)
                R_Motor.ChangeDutyCycle(50)
        elif GPIO.input(SW2):
            if 'sw2' != prev_btn:
                print("You clicked switch2")
                # 현재 버튼 업데이트
                prev_btn = 'sw2'

                # 방향 및 속도 설정 후 출력
                GPIO.output(AIN1, 0)
                GPIO.output(AIN2, 1)
                GPIO.output(BIN1, 0)
                GPIO.output(BIN2, 1)
                L_Motor.ChangeDutyCycle(50)
                R_Motor.ChangeDutyCycle(25)
        elif GPIO.input(SW3):
            if 'sw3' != prev_btn:
                print("You clicked switch3")
                # 현재 버튼 업데이트
                prev_btn = 'sw3'

                # 방향 및 속도 설정 후 출력
                GPIO.output(AIN1, 0)
                GPIO.output(AIN2, 1)
                GPIO.output(BIN1, 0)
                GPIO.output(BIN2, 1)
                L_Motor.ChangeDutyCycle(25)
                R_Motor.ChangeDutyCycle(50)
        elif GPIO.input(SW4):
            if 'sw2' != prev_btn:
                print("You clicked switch4")
```

```
                # 현재 버튼 업데이트
                prev_btn = 'sw2'

                # 방향 및 속도 설정 후 출력
                GPIO.output(AIN1, 1)
                GPIO.output(AIN2, 0)
                GPIO.output(BIN1, 1)
                GPIO.output(BIN2, 0)
                L_Motor.ChangeDutyCycle(50)
                R_Motor.ChangeDutyCycle(50)
        else:
            # 만약 버튼을 누르지 않은 상태라면 정지
            if prev_btn is not None:
                L_Motor.ChangeDutyCycle(0)
                R_Motor.ChangeDutyCycle(0)
                prev_btn = None
except KeyboardInterrupt:
    pass

GPIO.cleanup()
```

2) 해석

첫 번째는, 필요한 모듈과 함수를 import 하는 단계이다. 과제를 수행하는 데 필요한 모듈 및 함수는 RPi.GPIO 모듈, time 함수이다.

두 번째는, 사용할 모드에 맞는 핀 번호를 확인 후 명시적으로 정의한다. 과제를 수행하는 데 필요한 모드는 스위치 4개와 모터 속도 2개, 모터 방향 4개로, 번호는 각각 5, 6, 13, 19, 18, 23, 22, 27, 26, 24번이다.

세 번째는, 처음 실행할 때 환경 설정을 하는 단계이다. GPIO.setwarnings는 이전의 결과가 현재 실행에 영향을 줄 위험이 있는 경우에 경고 문구를 출력할지 말지를 결정하는 함수이다. 과제에서는 False로 설정한다. 다음으로 GPIO를 BCM 모드로 사용할 예정이므로, BCM 모드로 설정한다. 마지막으로 스위치는 입력 모드, 모터의 속도와 방향은 모두 출력 모드로 설정한다.

네 번째는, 모터의 주파수를 설정한다. 좌우 모두 250으로 설정하였다.

다섯 번째는, 초기 상태를 정지 상태로 설정한다.

여섯 번째는, 스위치를 누를 때만 한번 동작하도록 스위치별 이전 버튼의 상태 저장용 변수를 선언 후 None 또는 False로 초기화한다.

일곱 번째는, try 문 내부의 블록이며, 키보드 입력으로 'Ctrl + C'가 들어오지 않는 한, 계속 동작하도록 True 조건의 while 문을 이용하였다.

여덟 번째는, while 문 내부의 블록으로, 각 스위치가 눌렸는지 확인 후, 해당 스위치가 이전에 누르고 있던 상태인지 아닌지 확인한다. 만약 이전에 누르지 않고 처음 눌린 스위치라면, 어떤 스위치를 눌렀는지 출력 후, 현재 버튼 상태를 업데이트한다. 이후 각 스위치에 맞는 동작을 실행한다.

마지막으로, 다음 실행에 영향을 주지 않도록, p.stop() 함수와 GPIO.cleanup() 함수를 이용하여 종료 및 초기화한다.

3) 결과

동작 결과
"HW3_2" 영상 참조 1번 스위치 : 전진 2번 스위치 : 우회전 3번 스위치 : 좌회전 4번 스위치 : 후진