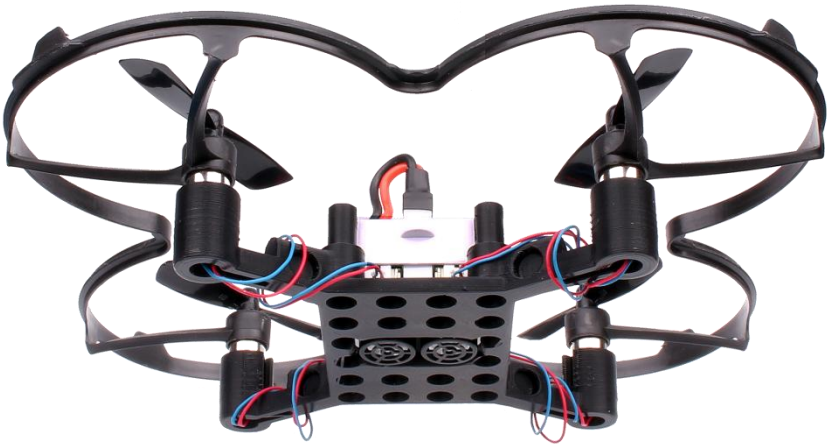
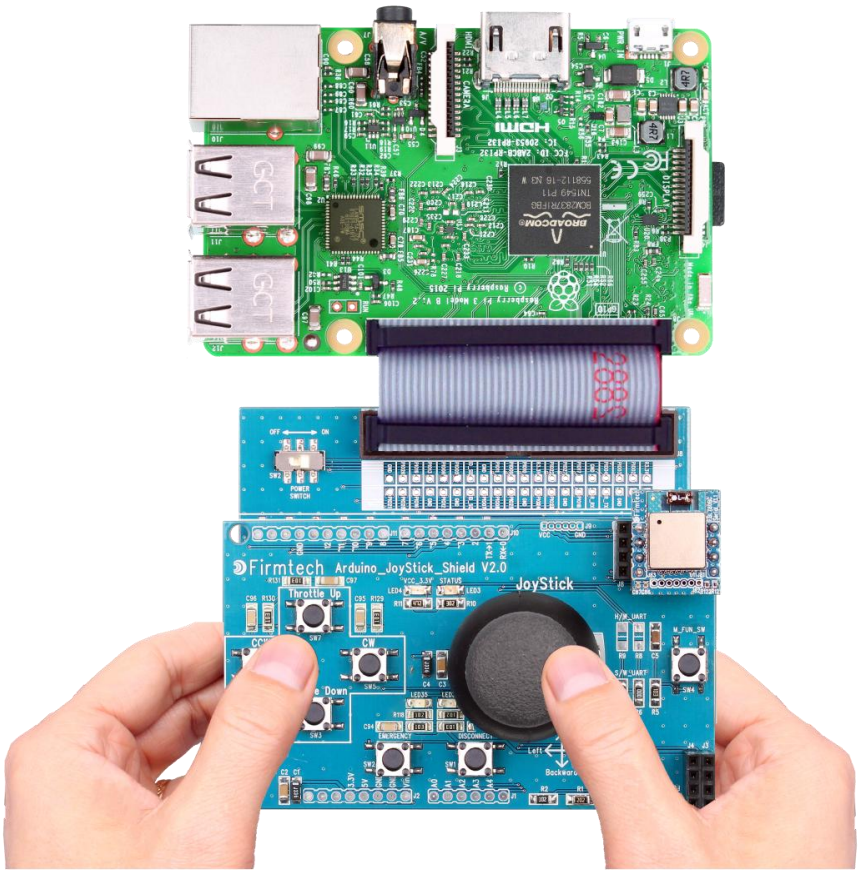
Firmtech

Raspberry Pi 3+Drone Kit

기능별 소스 분류에 대한 상세 설명

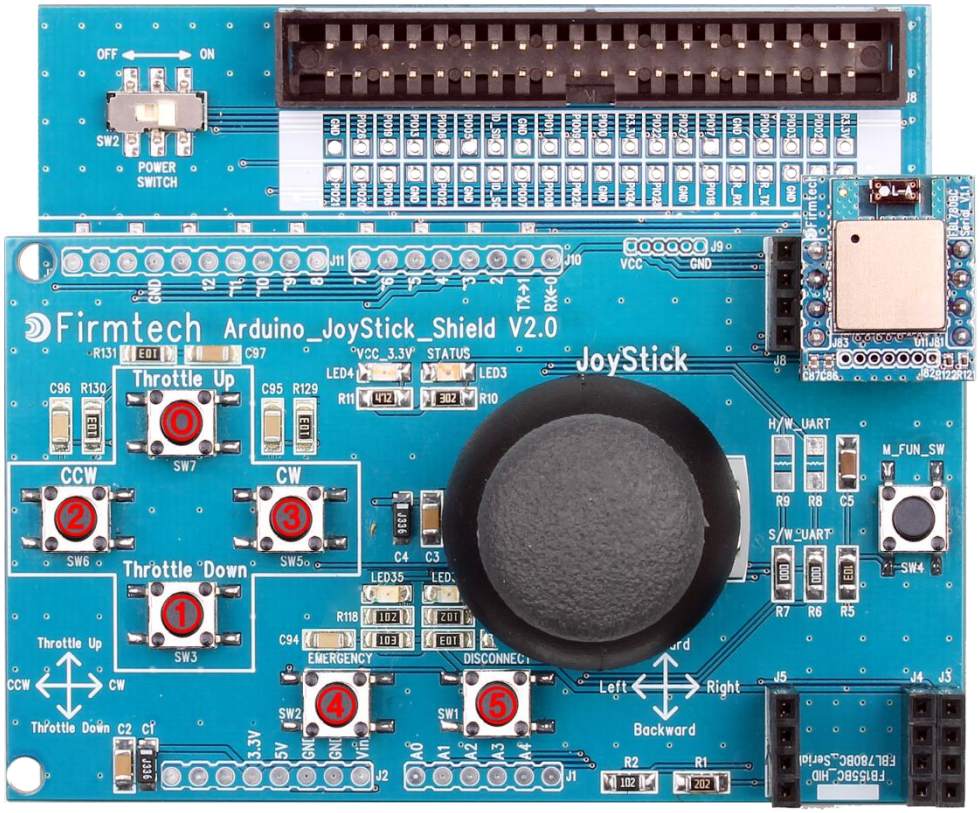




기능별 소스 분류에 대한 상세 설명(목차)

No	소스 화일명 (확장자 ino)	소스 세부 설명
01	Joystick_01_Debug_01	시리얼 포트를 통한 메시지 출력 방법 익히기-(1)
02	Joystick_01_Debug_02	시리얼 포트를 통한 메시지 출력 방법 익히기-(2)
03	Joystick_02_Variable	변수 사용법 익히기
04	Joystick_03_String_01	스트링(문자열) 사용법 익히기-(1)
05	Joystick_03_String_02	스트링(문자열) 사용법 익히기-(2)
06	Joystick_04_Pio_01	디지털 입,출력핀 사용법 익히기-(1)
07	Joystick_04_Pio_02	디지털 입,출력핀 사용법 익히기-(2)
08	Joystick_05_SPI_Adc_01	아날로그 입력 단자 사용법 익히기-(1)
09	Joystick_05_SPI_Adc_02	아날로그 입력 단자 사용법 익히기-(2)
10	Joystick_05_SPI_Adc_03	아날로그 입력 단자 사용법 익히기-(3)
11	Joystick_06_Function_01	함수 사용법 익히기-(1)
12	Joystick_06_Function_02	함수 사용법 익히기-(2)
13	Joystick_07_Checksum_01	데이터 통신 오류 검출에 활용되는 Checksum 사용법 익히기-(1)
14	Joystick_07_Checksum_02	데이터 통신 오류 검출에 활용되는 Checksum 사용법 익히기-(2)
15	Joystick_08_if-elif	if-elif문 사용법 익히기
16	Joystick_09_find_01	파이썬 내부 함수 사용법 익히기 - 특정 문자열 검색-(1)
17	Joystick_09_find_02	파이썬 내부 함수 사용법 익히기 - 특정 문자열 검색-(2)
18	Joystick_10_Serial_01	파이썬 내부 함수 사용법 익히기 - 하드웨어 UART (TX,RX,GND) 설정-(1)
19	Joystick_10_Serial_02	파이썬 내부 함수 사용법 익히기 - 하드웨어 UART (TX,RX,GND) 설정-(2)
20	Joystick_11_AT-Command	파이썬으로 블루투스 모듈(BLE 모듈) 사용법 익히기-(1)
21	Joystick_12_BT-Connect	파이썬으로 블루투스 모듈(BLE 모듈) 사용법 익히기-(2) 드론과의 연결 진행
22	Joystick_13_Check-Message_01	파이썬으로 블루투스 모듈(BLE 모듈) 사용법 익히기-(3)
23	Joystick_13_Check-Message_02	파이썬으로 블루투스 모듈(BLE 모듈) 사용법 익히기-(4)
24	Joystick_14_DataPacket_01	드론과의 통신에 사용되는 데이터 송,수신 패킷 구조 익히기-(1)
25	Joystick_14_DataPacket_02	드론과의 통신에 사용되는 데이터 송,수신 패킷 구조 익히기-(2)
26	Joystick_14_DataPacket_03	드론과의 통신에 사용되는 데이터 송,수신 패킷 구조 익히기-(3)
27	Joystick_14_DataPacket_04	드론을 제어해 보자 - 드론 전진, 후진
28	Joystick_14_DataPacket_05	드론을 제어해 보자 - 드론 좌측 이동, 우측 이동
29	Joystick_14_DataPacket_06	드론을 제어해 보자 - 드론 좌회전, 우회전
30	Joystick_14_DataPacket_07	드론을 제어해 보자 - 드론 비행고도(높이) 조정
31	Joystick_20_Drone_Shield	라즈베리파이 3+ 드론 킷 전체 소스

Joystick Shield Description



번호	포트 번호	기능 구분	기 능
⓪	PIO22	Throttle UP	드론을 상승 시킵니다.
①	PIO27	Throttle Down	드론을 하강 시킵니다.
②	PIO17	CCW	드론을 좌회전(제자리에서) 시킵니다.
③	PIO23	CW	드론을 우회전(제자리에서) 시킵니다.
④	PIO24	Emergency	드론과의 연결 요청시 사용됩니다. 드론과 연결 후에는 응급 버튼으로 사용
⑤	PIO25	Disconnect	드론과 연결 후 연결 해지 버튼으로 사용

〈 라즈베리 드론 **"조이스틱 쉴드"** 소스 뽀개기 〉

사전 작업
라즈베리파이 40핀 케이블을 이용하여 라즈베리파이 확장 보드에 연결해야 합니다. (확장보드 전원은 OFF된 상태입니다.)

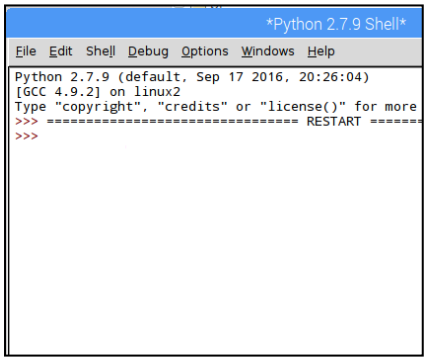
1. Joystick_01_Debug_01

파이썬은 "Python Shell"(콘솔)을 이용한 디버깅 메시지 확인이 쉽습니다.

```
print("RaspberryPi Debug Test 01\n")

while True:
    continue
```

"print()"만 사용하면 Python Shell(콘솔) 사용 준비 완료됩니다.
출력하고자 하는 디버깅 메시지를 "print()"에 입력합니다.
여기서는 "RaspberryPi Debug Test 01"을 입력하였습니다.

프로그램의 지속적인 반복 수행을 위해 "while True: continue"를 사용합니다.



① Python 2(IDLE) 프로그램으로 Joystick_01_Debug_01.py를 오픈합니다.
② Python 2 프로그램을 실행합니다.(Run Module 실행 또는 F5를 누릅니다.)
③ Python Shell이 실행됩니다.
④ 출력되는 디버깅 데이터를 확인합니다.
⑤ Python Shell에서 "Ctrl+C" 입력하여 프로그램을 종료합니다.

2. Joystick_01_Debug_02

디버깅 메시지는 문자 메시지 뿐만 아니라 숫자도 확인이 가능합니다.

```
import time

#-------------------------------------
i = 0
#-------------------------------------

print("RaspberryPi Debug Test 02\n")

while True:
    print(i)
    i += 1

    if i > 10:
        i = 0
        print("Console Test\n")

    time.sleep(0.3)
```

"time"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.

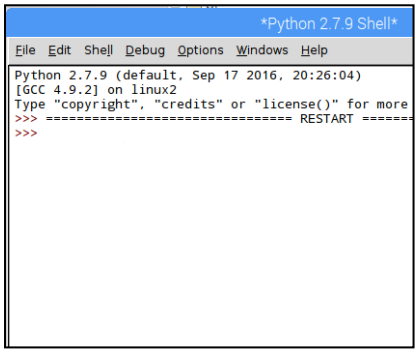
숫자를 저장할 공간으로 'i'를 사용합니다.

출력하고자 하는 디버깅 메시지를 "print()"에 입력합니다.
여기서는 "RaspberryPi Debug Test 02"를 입력하였습니다.

프로그램의 지속적인 반복 수행을 위해 "while True:"를 사용합니다.
저장된 숫자를 출력하기 위해 "print(i)"를 사용합니다.
i는 1씩 증가하도록 합니다.

i가 10보다 커지면 i를 0으로 저장하고 "Console Test"라는 메시지를 출력하도록 합니다.

이 동작은 300ms간격으로 반복됩니다.



① Python 2(IDLE) 프로그램으로 Joystick_01_Debug_02.py를 오픈합니다.
② Python 2 프로그램을 실행합니다.(Run Module 실행 또는 F5를 누릅니다.)
③ Python Shell이 실행됩니다.
④ 출력되는 디버깅 데이터를 확인합니다.
⑤ Python Shell에서 "Ctrl+C" 입력하여 프로그램을 종료합니다.

3. Joystick_02_Variable

변수는 변화되는 수를 저장하는 공간입니다.

하나의 공간에 다른 값의 수를 저장할 수 있습니다.

변수 i에 0이라는 수를 저장하는 경우: i = 0, 이 순간 변수 i는 숫자 0과 같습니다.
변수 i에 5라는 수를 저장하는 경우: i = 5, 이 순간 변수 i는 숫자 5와 같습니다.

```
import time

#-------------------------------------
i = 0
#-------------------------------------

print("RaspberryPi Variable Test\n")

for i in range(100):
    print(i)
    time.sleep(0.1)

while True:
    continue
```

"time"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.

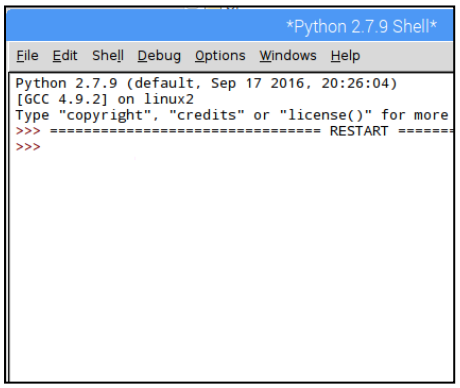
숫자를 저장할 공간으로 'i'를 사용합니다.

출력하고자 하는 디버깅 메시지를 "print()"에 입력합니다.
여기서는 "RaspberryPi Variable Test"를 입력하였습니다.

숫자는 0에서 99까지 1씩 증가하도록 합니다.
변화된 숫자는 i에 저장됩니다.
저장된 숫자를 출력하기 위해서 "print(i)"를 사용합니다.

프로그램의 지속적인 반복 수행을 위해 "while True: continue"를 사용합니다.

"while True: continue"에 의해 Joystick_02_Variable 프로그램은 아무런 작업을 하지 않으면서 수행됩니다.



① Python 2(IDLE) 프로그램으로 Joystick_02_Variable.py를 오픈합니다.
② Python 2 프로그램을 실행합니다.(Run Module 실행 또는 F5를 누릅니다.)
③ Python Shell이 실행됩니다.
④ 출력되는 디버깅 데이터를 확인합니다.
⑤ Python Shell에서 "Ctrl+C" 입력하여 프로그램을 종료합니다.

4. Joystick_03_String_01

스트링은 문자열을 저장하는 공간입니다. 즉, 문자열을 저장하는 변수입니다.

하나의 공간에 다른 문자열을 저장할 수 있습니다.

스트링 변수 st에 "abc"를 저장하는 경우: st = "abc", 이 순간 스트링 변수 st는 문자열 "abc"와 같습니다.
스트링 변수 st에 "12b"를 저장하는 경우: st = "12b", 이 순간 스트링 변수 st는 문자열 "12b"와 같습니다.

```
testString = ""
#-------------------------------------

print("RaspberryPi String Test 01\n")

testString = "1234567890"
print(testString)

while True:
    continue
```

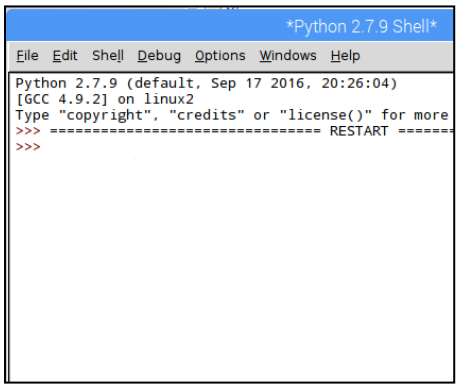
문자열을 저장할 공간으로 'testString'을 사용합니다.

출력하고자 하는 디버깅 메시지를 "print()"에 입력합니다.
여기서는 "RaspberryPi String Test 01"을 입력하였습니다.

스트링 변수에 "1234567890"을 저장합니다.
스트링 변수를 출력하기 위해 "print(testString)"을 사용합니다.

프로그램의 지속적인 반복 수행을 위해 "while True: continue"를 사용합니다.

"while True: continue"에 의해 Joystick_03_String_01 프로그램은 아무런 작업을 하지 않으면서 수행됩니다.



① Python 2(IDLE) 프로그램으로 Joystick_03_String_01.py를 오픈합니다.
② Python 2 프로그램을 실행합니다.(Run Module 실행 또는 F5를 누릅니다.)
③ Python Shell이 실행됩니다.
④ 출력되는 디버깅 데이터를 확인합니다.
⑤ Python Shell에서 "Ctrl+C" 입력하여 프로그램을 종료합니다.

5. Joystick_03_String_02

```
testString = ""
#-------------------------------------

print("RaspberryPi String Test 02\n")

testString = "1234567890"
print(testString)

testString = "abc"
print(testString)

while True:
    continue
```

문자열을 저장할 공간으로 'testString'을 사용합니다.

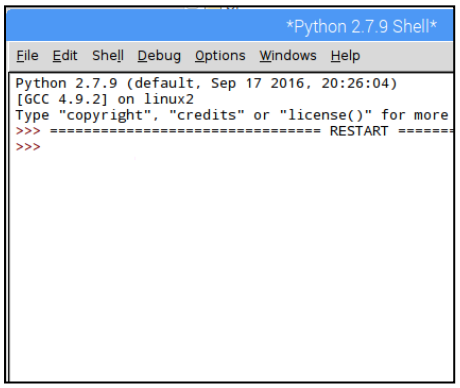
출력하고자 하는 디버깅 메시지를 "print()"에 입력합니다.
여기서는 "RaspberryPi String Test 02"를 입력하였습니다.

스트링 변수에 "1234567890"을 저장합니다.
스트링 변수를 출력하기 위해 "print(testString)을 사용합니다.

스트링 변수에 "abc"를 저장합니다.
스트링 변수를 출력하기 위해 "print(testString)을 사용합니다.

프로그램의 지속적인 반복 수행을 위해 "while True: continue"를 사용합니다.

"while True: continue"에 의해 Joystick_03_String_01 프로그램은 아무런 작업을 하지 않으면서 수행됩니다.



① Python 2(IDLE) 프로그램으로 Joystick_03_String_02.py를 오픈합니다.
② Python 2 프로그램을 실행합니다.(Run Module 실행 또는 F5를 누릅니다.)
③ Python Shell이 실행됩니다.
④ 출력되는 디버깅 데이터를 확인합니다.
⑤ Python Shell에서 "Ctrl+C" 입력하여 프로그램을 종료합니다.

6. Joystick_04_Pio_01

디지털 포트는 전기적 신호를 Low/High로 내보내거나 Low/High로 입력된 상태를 체크합니다.

펌테크의 **"조이스틱 쉴드"**는 입력 상태 체크만 가능합니다.

펌테크의 **"조이스틱 쉴드"**는 High상태에서 Low상태로 변화된 경우를 체크합니다.

6개의 디지털 포트를 사용합니다.

```
import time
import RPi.GPIO as GPIO

#--------------------------------------
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

#--------------------------------------
pio_list = [22, 27, 17, 23, 24, 25]

for i in range(6):
    GPIO.setup(pio_list[i], GPIO.IN)

#--------------------------------------

print("RaspberryPi Pio Test 01\n")

while True:
    if GPIO.input(pio_list[4]) == 0:
        print("Press 4")
        time.sleep(0.1)
```

"time"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.
"PIO"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.

"PIO" 포트의 번호 사용 방식을 "BCM"으로 설정합니다.
"PIO" 포트의 사용 중에 프로그램을 종료하는 경우 발생되는 warning을 출력되지 않도록 설정합니다.

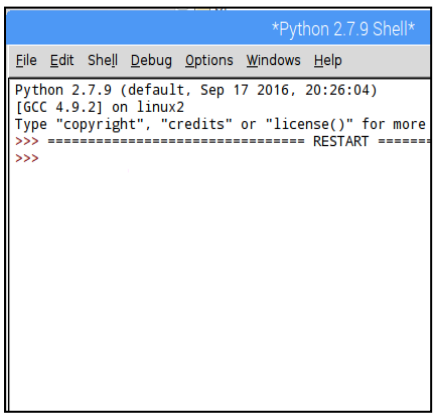
"PIO" 포트로 사용할 포트를 'pio_list"라는 리스트에 설정합니다.
GPIO.setup()을 사용하여 6개의 디지털 포트를 입력으로 설정합니다.

출력하고자 하는 디버깅 메시지를 "print()"에 입력합니다.
여기서는 "RaspberryPi Pio Test 01"을 입력하였습니다.

프로그램의 지속적인 반복 수행을 위해 "while True:"를 사용합니다.

"if GPIO.input(pio_list[4]) == 0"을 사용하여 디지털 4번째 포트가 High 상태에서 Low 상태로 변경되었는지 체크를 합니다. (포트의 시작은 0번째 포트부터 시작합니다.)

4번째 포트가 Low 상태로 변경된 경우, "print("Press 4")"를 이용하여 메시지를 출력합니다.



① 라즈베리파이 확장보드에 조이스틱 쉴드를 장착합니다.
② 라즈베리파이 확장보드의 전원을 ON 합니다.
③ Python 2(IDLE) 프로그램으로 Joystick_04_Pio_01.py를 오픈합니다.
④ Python 2 프로그램을 실행합니다.(Run Module 실행 또는 F5를 누릅니다.)
⑤ Python Shell이 실행됩니다.
⑥ 출력되는 디버깅 데이터를 확인합니다.
⑦ 조이스틱 쉴드의 4번 스위치를 누릅니다.
⑧ 출력되는 디버깅 데이터를 확인합니다.
⑨ Python Shell에서 "Ctrl+C" 입력하여 프로그램을 종료합니다.
⑩ 라즈베리파이 확장보드의 전원을 OFF 합니다.

7. Joystick_04_Pio_02

```
import time
import RPi.GPIO as GPIO

#-------------------------------------
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

#-------------------------------------
pio_list = [22, 27, 17, 23, 24, 25]

for i in range(6):
    GPIO.setup(pio_list[i], GPIO.IN)

#-------------------------------------

print("RaspberryPi Pio Test 02\n")

while True:
    if GPIO.input(pio_list[0]) == 0:
        print("Press 0")
        time.sleep(0.1)
    if GPIO.input(pio_list[1]) == 0:
        print("Press 1")
        time.sleep(0.1)
    if GPIO.input(pio_list[2]) == 0:
        print("Press 2")
        time.sleep(0.1)
    if GPIO.input(pio_list[3]) == 0:
        print("Press 3")
        time.sleep(0.1)
    if GPIO.input(pio_list[4]) == 0:
        print("Press 4")
        time.sleep(0.1)
    if GPIO.input(pio_list[5]) == 0:
        print("Press 5")
        time.sleep(0.1)
```

"time"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.
"PIO"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.

"PIO" 포트의 번호 사용 방식을 "BCM"으로 설정합니다.
"PIO" 포트의 사용 중에 프로그램을 종료하는 경우 발생되는 warning을 출력되지 않도록 설정합니다.

"PIO" 포트로 사용할 포트를 'pio_list"라는 리스트에 설정합니다.
GPIO.setup()을 사용하여 6개의 디지털 포트를 입력으로 설정합니다.

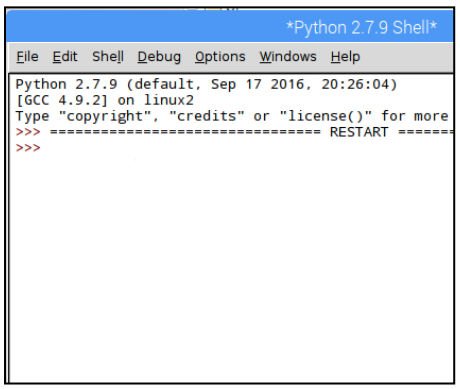
출력하고자 하는 디버깅 메시지를 "print()"에 입력합니다.
여기서는 "RaspberryPi Pio Test 02"를 입력하였습니다.

프로그램의 지속적인 반복 수행을 위해 "while True:"를 사용합니다.

"if GPIO.input(pio_list[0]) == 0"을 사용하여 디지털 0번째 포트가 High 상태에서 Low 상태로 변경되었는지 체크를 합니다.
(포트의 시작은 0번째 포트부터 시작합니다.)

0번째 포트가 Low 상태로 변경된 경우, "print("Press 0")"을 이용하여 메시지를 출력합니다.

0 ~ 5번째 포트가 Low 상태로 변경됐는지 체크합니다.



① 라즈베리파이 확장보드에 조이스틱 쉴드를 장착합니다.
② 라즈베리파이 확장보드의 전원을 ON 합니다.
③ Python 2(IDLE) 프로그램으로 Joystick_04_Pio_02.py를 오픈합니다.
④ Python 2 프로그램을 실행합니다.(Run Module 실행 또는 F5를 누릅니다.)
⑤ Python Shell이 실행됩니다.
⑥ 출력되는 디버깅 데이터를 확인합니다.
⑦ 조이스틱 쉴드의 0~5번 스위치를 누릅니다.
⑧ 출력되는 디버깅 데이터를 확인합니다.
⑨ Python Shell에서 "Ctrl+C" 입력하여 프로그램을 종료합니다.
⑩ 라즈베리파이 확장보드의 전원을 OFF 합니다.

8. Joystick_05_SPI_Adc_01

라즈베리파이는 기본적으로 SPI 통신을 사용할 수 없습니다.
사전 준비 작업을 진행하여 라즈베리파이가 SPI Module을 부팅 시 로딩되도록 해야 합니다.

라즈베리파이는 아날로그 포트가 없습니다.
라즈베리파이에서 아날로그 데이터를 입력 받기 위해서는 ADC 변환 칩을 사용해야 합니다.
ADC 변환 칩 MCP3004는 4채널 아날로그 데이터 입력이 가능하고(0~3채널), 변환된 ADC 데이터를 SPI 통신으로 출력합니다.

MCP3004를 사용하여 0 ~ 5V 사이의 연속적인 아날로그 값을 입력 받습니다.
MCP3004를 사용하여 연속적인 아날로그 값의 순간 값을 읽습니다.
MCP3004는 연속적인 아날로그 값을 읽어 0 ~ 1024의 디지털 값(순간 값)으로 변경합니다.

아날로그 값으로 0V가 입력된 경우, MCP3004는 디지털 값 0을 출력합니다.
아날로그 값으로 2.5V가 입력된 경우, MCP3004는 디지털 값 512를 출력합니다.
아날로그 값으로 5V가 입력된 경우, MCP3004는 디지털 값 1024를 출력합니다.

"조이스틱 쉴드" 조정장치의 좌우(roll) 이동은 MCP3004의 2번 채널에 연결되어 있습니다.

```
#--------------------------------------
import time
import spidev

#--------------------------------------
spi = spidev.SpiDev()
spi.open(0,0)
spi.max_speed_hz = 1000000

#--------------------------------------
#--------------------------------------

print("\nRaspberryPi SPI-Adc Test 01\n")

while True:
    data = spi.xfer2([1, (0x08 + 2) << 4, 0])
    adc_out = ((data[1] & 0x03) << 8) + data[2]
    time.sleep(0.01)

    print(adc_out)
    time.sleep(0.01)
```

"time"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.
"SPI"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.
"SPI" 라이브러리를 사용하기 위해 "인자"를 생성합니다.
생성된 "인자"를 사용하여 라이브러리 초기화를 진행합니다.
최대 클럭 스피드를 1MHz로 설정합니다.

출력하고자 하는 디버깅 메시지를 "print()"에 입력합니다.
여기서는 "RaspberryPi SPI-Adc Test 01"을 입력하였습니다.

프로그램의 지속적인 반복 수행을 위해 "while True:"를 사용합니다.

"SPI" 라이브러리에서 제공하는 함수를 이용하여 2번 채널의 ADC 값을 읽어와 data에 저장합니다.

data에 저장된 데이터 중 10비트만 adc_out에 저장합니다.

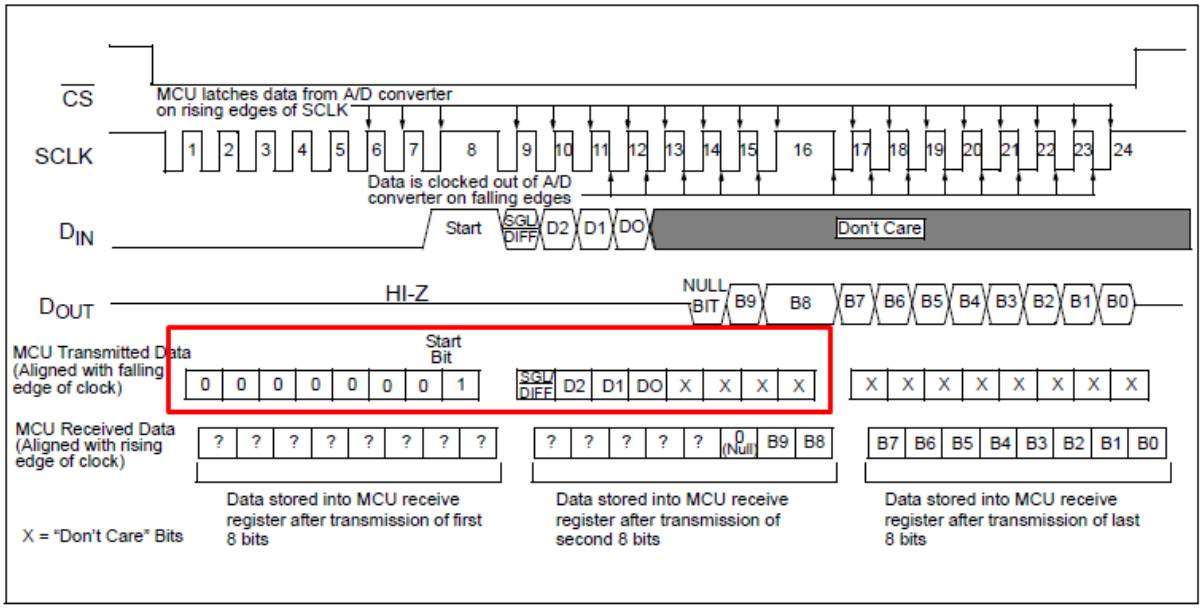
약 10ms 대기합니다.

"print(adc_out)"을 이용하여 ADC값을 출력합니다.

이 동작은 10ms 간격으로 반복됩니다.

SPI 통신을 하고자 하는 경우 파이썬에서 제공하는 외부 함수를 사용하면 쉽습니다.
"open(bus, device)" => bus, device로 지정한 SPI 디바이스에 접속합니다. (사용 가능하도록 합니다.)
"xfer2([values])" => 리스트 values로 지정한 데이터를 SPI 디바이스에 송신함과 동시에 수신한 데이터를 리스트로 돌려줍니다. 전송 중에 CS신호는 액티브인 상태를 유지합니다.

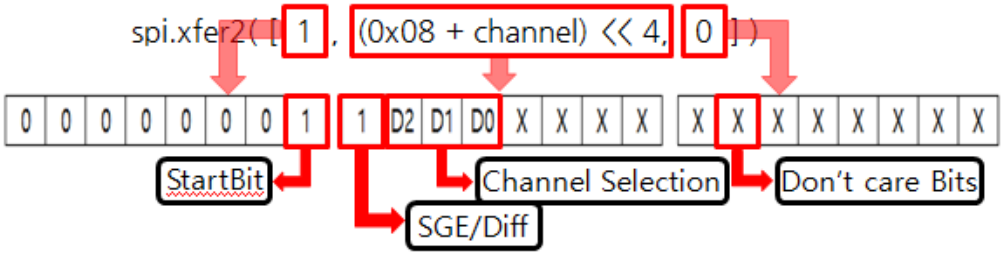
"data = xfer2([1, (0x08 + 2) <<4, 0])" 만드는 자료



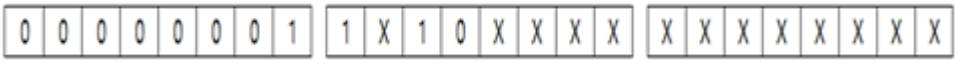
Control Bit Selections				Input Configuration	Channel Selection
Single/ Diff	D2*	D1	D0		
1	X	0	0	single-ended	CH0
1	X	0	1	single-ended	CH1
1	X	1	0	single-ended	CH2
1	X	1	1	single-ended	CH3

Xfer2 : spi.xfer2() 메서드를 사용하여 MCP3004에 3byte의 Transmitted Data를 보내면 해당 채널의 ADC값 3byte가 Received Data로 발생됩니다.

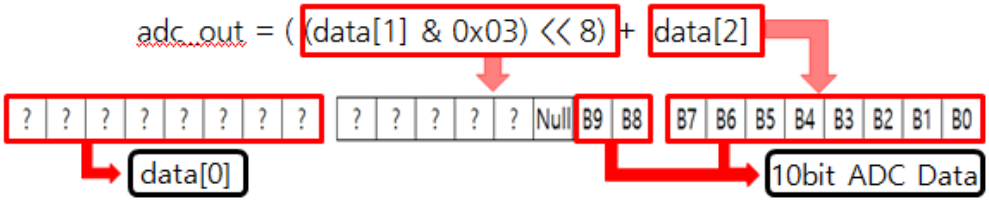
▶ Transmitted Data 3 byte는 아래와 같습니다.



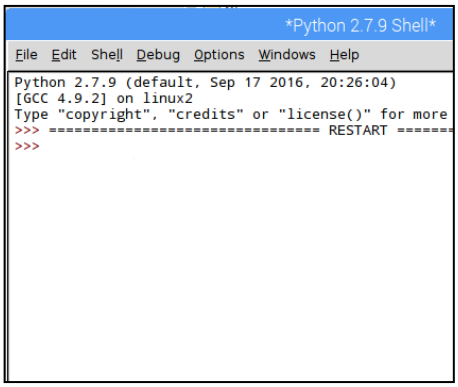
예) CH2의 ADC값을 확인하기 위해서는 아래와 같이 Transmitted Data를 보내야 합니다.



▶ Received Data 3 byte는 아래와 같습니다.



3 byte 중 data[0]은 필요 없는 값이고, data[1]은 하위 2 bit만 필요하여 0x03 과 비트 논리곱 한 후 2 byte 데이터를 10 bit 데이터로 합하기 위하여 상위의 자리수로 변경하기 위해 8 좌시프트 하고 0data[2]를 더합니다.



① 라즈베리파이 확장보드에 조이스틱 쉴드를 장착합니다.
② 라즈베리파이 확장보드의 전원을 ON 합니다.
③ Python 2(IDLE) 프로그램으로 Joystick_05_SPI-Adc_01.py를 오픈합니다.
④ Python 2 프로그램을 실행합니다.(Run Module 실행 또는 F5를 누릅니다.)
⑤ Python Shell이 실행됩니다.
⑥ 출력되는 디버깅 데이터를 확인합니다.
⑦ 조이스틱 쉴드의 조이스틱을 좌우로 조정합니다.
⑧ 출력되는 디버깅 데이터를 확인합니다.
⑨ Python Shell에서 "Ctrl+C" 입력하여 프로그램을 종료합니다.
⑩ 라즈베리파이 확장보드의 전원을 OFF 합니다.

9. Joystick_05_SPI_Adc_02

"조이스틱 쉴드" 조정장치의 좌우(roll) 이동은 MCP3004칩의 2번 채널에 연결되어 있습니다.
"조이스틱 쉴드" 조정장치의 전후(pitch) 이동은 MCP3004칩의 3번 채널에 연결되어 있습니다.

```
#--------------------------
import time
import spidev

#--------------------------
spi = spidev.SpiDev()
spi.open(0,0)
spi.max_speed_hz = 1000000

#--------------------------
#--------------------------

print("\nRaspberryPi SPI-Adc Test 02\n")

while True:
    data = spi.xfer2([1, (0x08 + 2) << 4, 0])
    adc_out2 = ((data[1] & 0x03) << 8) + data[2]
    time.sleep(0.01)

    data = spi.xfer2([1, (0x08 + 3) << 4, 0])
    adc_out3 = ((data[1] & 0x03) << 8) + data[2]
    time.sleep(0.01)

    print("adc2 = %d   adc3 = %d" %(adc_out2, adc_out3))
    time.sleep(0.01)
```

"time"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.
"SPI"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.
"SPI" 라이브러리를 사용하기 위해 "인자"를 생성합니다.
생성된 "인자"를 사용하여 라이브러리 초기화를 진행합니다.
최대 클럭 스피드를 1MHz로 설정합니다.

출력하고자 하는 디버깅 메시지를 "print()"에 입력합니다.
여기서는 "RaspberryPi SPI-Adc Test 02"를 입력하였습니다.

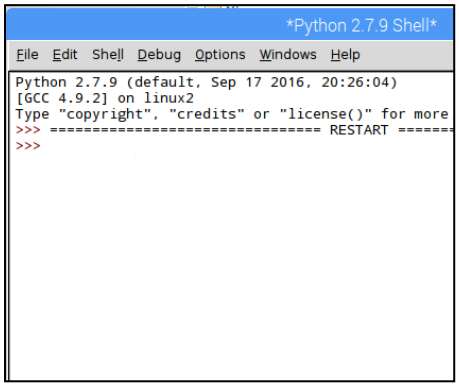
프로그램의 지속적인 반복 수행을 위해 "while True:"를 사용합니다.

"SPI" 라이브러리에서 제공하는 함수를 이용하여 2번 채널의 ADC 값을 읽어와 data에 저장합니다.
data에 저장된 데이터 중 10비트만 adc_out2에 저장합니다.
약 10ms 대기합니다.

"SPI" 라이브러리에서 제공하는 함수를 이용하여 3번 채널의 ADC 값을 읽어와 data에 저장합니다.
data에 저장된 데이터 중 10비트만 adc_out3에 저장합니다.
약 10ms 대기합니다.

"print()"을 이용하여 adc_out2와 adc_out3에 저장된 내용을 출력합니다.

이 동작은 10ms 간격으로 반복됩니다.



① 라즈베리파이 확장보드에 조이스틱 쉴드를 장착합니다.
② 라즈베리파이 확장보드의 전원을 ON 합니다.
③ Python 2(IDLE) 프로그램으로 Joystick_05_SPI-Adc_02.py를 오픈합니다.
④ Python 2 프로그램을 실행합니다.(Run Module 실행 또는 F5를 누릅니다.)
⑤ Python Shell이 실행됩니다.
⑥ 출력되는 디버깅 데이터를 확인합니다.
⑦ 조이스틱 쉴드의 조이스틱을 좌/우, 전/후 조정합니다.
⑧ 출력되는 디버깅 데이터를 확인합니다.
⑨ Python Shell에서 "Ctrl+C" 입력하여 프로그램을 종료합니다.
⑩ 라즈베리파이 확장보드의 전원을 OFF 합니다.

10. Joystick_05_SPI_Adc_03

ADC 값을 읽어오는 부분을 함수로 만듭니다. (함수에 대한 내용은 다음 챕터를 참고하시기 바랍니다.)

```
#--------------------------
import time
import spidev

#--------------------------
spi = spidev.SpiDev()
spi.open(0,0)
spi.max_speed_hz = 1000000

#--------------------------
def analog_read(channel):
    data = spi.xfer2([1, (0x08 + channel) << 4, 0])
    adc_out = ((data[1] & 0x03) << 8) + data[2]
    time.sleep(0.01)

    return adc_out

#--------------------------

print("\nRaspberryPi SPI-Adc Test 03\n")

while True:
    adc2 = analog_read(2)
    adc3 = analog_read(3)

    print("adc2 = %d   adc3 = %d" %(adc2, adc3))
    time.sleep(0.01)
```

"time"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.
"SPI"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.
"SPI" 라이브러리를 사용하기 위해 "인자"를 생성합니다.
생성된 "인자"를 사용하여 라이브러리 초기화를 진행합니다.
최대 클럭 스피드를 1MHz로 설정합니다.

"analog_read()"란 이름으로 함수를 만듭니다.
이 함수는 MCP3004 칩에 연결된 ADC값을 전달된 채널을 이용하여 읽고, 전달합니다.

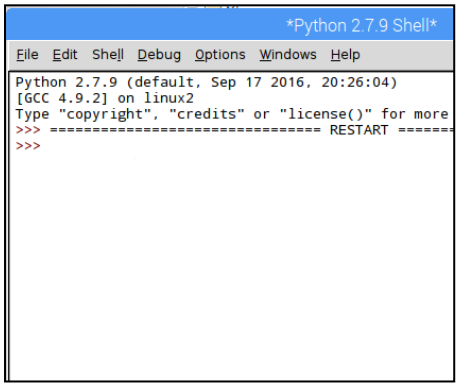
출력하고자 하는 디버깅 메시지를 "print()"에 입력합니다.
여기서는 "RaspberryPi SPI-Adc Test 03"을 입력하였습니다.

프로그램의 지속적인 반복 수행을 위해 "while True:"를 사용합니다.

"analog_read(2)" 함수를 호출하여 2번 채널 ADC 값을 읽어 adc2에 저장합니다.
"analog_read(3)" 함수를 호출하여 3번 채널 ADC 값을 읽어 adc3에 저장합니다.

"print()"을 이용하여 adc2와 adc3에 저장된 내용을 출력합니다.

이 동작은 10ms 간격으로 반복됩니다.



① 라즈베리파이 확장보드에 조이스틱 쉴드를 장착합니다.
② 라즈베리파이 확장보드의 전원을 ON 합니다.
③ Python 2(IDLE) 프로그램으로 Joystick_05_SPI-Adc_03.py를 오픈합니다.
④ Python 2 프로그램을 실행합니다.(Run Module 실행 또는 F5를 누릅니다.)
⑤ Python Shell이 실행됩니다.
⑥ 출력되는 디버깅 데이터를 확인합니다.
⑦ 조이스틱 쉴드의 조이스틱을 좌/우, 전/후 조정합니다.
⑧ 출력되는 디버깅 데이터를 확인합니다.
⑨ Python Shell에서 "Ctrl+C" 입력하여 프로그램을 종료합니다.
⑩ 라즈베리파이 확장보드의 전원을 OFF 합니다.

11. Joystick_06_Function_01

"함수"는 프로그램 작성의 효율을 높이고 소스 분석의 이해도를 높이기 위해 사용합니다.

특별한 기능을 수행하는 부분을 따로 모아 놓으면, 효율과 이해도가 높아집니다.

특별한 기능이 필요한 경우, 기능을 모아 놓은 함수를 호출한다면 프로그램 작성의 효율이 높아집니다.

특별한 기능에 알맞은 이름으로 함수를 정의한 경우, 함수의 이름 만으로도 그 특별한 기능이 어떤 기능을 수행하는지 파악이 가능하여 소스 분석의 이해도가 높아집니다.

```
def disp11111():
    print("11111")

#--------------------------------------

print("RaspberryPi Function Test 01\n")

disp11111()

while True:
    continue
```

"disp11111()"이란 이름으로 함수를 만듭니다.

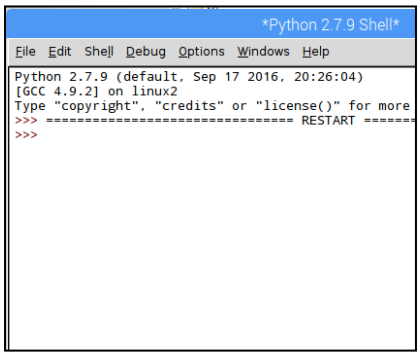
이 함수는 Python Shell(콘솔)로 "11111"을 출력하는 기능을 합니다.

출력하고자 하는 디버깅 메시지를 "print()"에 입력합니다.
여기서는 "RaspberryPi Function Test 01"을 입력하였습니다.

작성한 함수를 실행하기 위해 함수의 이름 "disp11111()"을 이용하여 함수를 호출합니다.

프로그램의 지속적인 반복 수행을 위해 "while True: continue"를 사용합니다.

"while True: continue"에 의해 Joystick_06_Function_01 프로그램은 아무런 작업을 하지 않으면서 수행됩니다.



① Python 2(IDLE) 프로그램으로 Joystick_06_Function_01.py를 오픈합니다.
② Python 2 프로그램을 실행합니다.(Run Module 실행 또는 F5를 누릅니다.)
③ Python Shell이 실행됩니다.
④ 출력되는 디버깅 데이터를 확인합니다.
⑤ Python Shell에서 "Ctrl+C" 입력하여 프로그램을 종료합니다.

12. Joystick_06_Function_02

```
def disp11111():
    print("11111")

def calculation():
    a = 1
    b = 2
    c = 0

    c = a+b
    print("a+b = %d" %c)

#----------------------------------------

print("RaspberryPi Function Test 02\n")

disp11111()

calculation()

while True:
    continue
```

"disp11111()"이란 이름으로 함수를 만듭니다.
이 함수는 Python Shell(콘솔)로 "11111"을 출력하는 기능을 합니다.

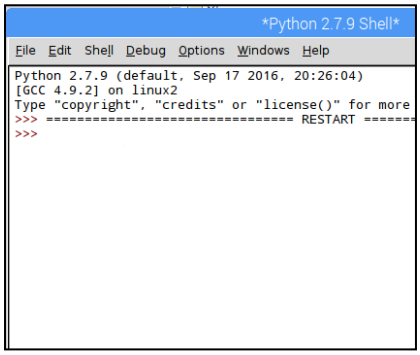
"calculation()"이란 이름으로 함수를 만듭니다.
이 함수는 변수 a에 1을 저장하고, 변수 b에 2를 저장, a와 b를 더해서 변수 c에 저장을 하고 Python Shell(콘솔)로 결과값을 출력하는 기능을 합니다.

출력하고자 하는 디버깅 메시지를 "print()"에 입력합니다.
여기서는 "RaspberryPi Function Test 02"를 입력하였습니다.

작성한 함수를 실행하기 위해 함수의 이름 disp11111()"와 "calculation()"을 이용하여 함수를 호출합니다.

프로그램의 지속적인 반복 수행을 위해 "while True: continue"를 사용합니다.

"while True: continue"에 의해 Joystick_06_Function_02 프로그램은 아무런 작업을 하지 않으면서 수행됩니다.



① Python 2(IDLE) 프로그램으로 Joystick_06_Function_02.py를 오픈합니다.
② Python 2 프로그램을 실행합니다.(Run Module 실행 또는 F5를 누릅니다.)
③ Python Shell이 실행됩니다.
④ 출력되는 디버깅 데이터를 확인합니다.
⑤ Python Shell에서 "Ctrl+C" 입력하여 프로그램을 종료합니다.

13. Joystick_07_Checksum_01

데이터를 전송할 때 오류를 검출하기 위해 checkSum을 이용합니다.

checkSum은 데이터의 정확성을 검사하기 위한 용도로 입력 데이터나 전송 데이터의 맨 마지막에 앞서 보낸 모든 데이터를 다 합한 합계를 따로 보내는 것입니다.

데이터를 받아들이는 측에서는 하나씩 받아들여 합산한 다음 이를 최종적으로 들어온 검사 합계와 비교하여 착오가 있는지를 점검합니다.

```
import time

#------------------------------------------------
#------------------------------------------------
data1 = 0x00
data2 = 0x00
data3 = 0x00
data4 = 0x00
data5 = 0x00
data6 = 0x00
checkSum = 0

#------------------------------------------------

print("RaspberryPi Checksum Test 01\n")

for data1 in range(20):
    checkSum = data1 + data2 + data3 + data4 + data5 + data6
    checkSum = checkSum & 0x00ff
    print(hex(checkSum))
    time.sleep(0.1)

while True:
    continue
```

"time"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.

송신 데이터로 data1 ~ data6까지 변수를 지정합니다.
결과 저장 변수로 checkSum을 지정합니다.

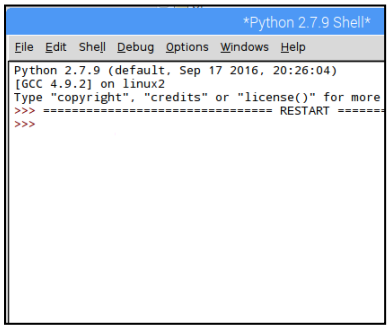
출력하고자 하는 디버깅 메시지를 "print()"에 입력합니다.
여기서는 "RaspberryPi Checksum Test 01"을 입력하였습니다.

체크섬 값의 변화를 알아보기 위해 모든 데이터를 0으로 하고 data1만 0부터 19까지 변화시킵니다.

계산된 체크섬 값을 Python Shell(콘솔)로 출력시킵니다.

프로그램의 지속적인 반복 수행을 위해 "while True: continue"를 사용합니다.

"while True: continue"에 의해 Joystick_07_Checksum_01 프로그램은 아무런 작업을 하지 않으면서 수행됩니다.



① Python 2(IDLE) 프로그램으로 Joystick_07_Checksum_01.py를 오픈합니다.
② Python 2 프로그램을 실행합니다.(Run Module 실행 또는 F5를 누릅니다.)
③ Python Shell이 실행됩니다.
④ 출력되는 디버깅 데이터를 확인합니다.
⑤ Python Shell에서 "Ctrl+C" 입력하여 프로그램을 종료합니다.

14. Joystick_07_Checksum_02

```
import time

#------------------------------------------------
#------------------------------------------------
data1 = 0xa1
data2 = 0x64
data3 = 0x64
data4 = 0x64
data5 = 0x78
data6 = 0x01
checkSum = 0

#------------------------------------------------

print("RaspberryPi Checksum Test 02\n")

while True:
    checkSum = data1 + data2 + data3 + data4 + data5 + data6
    checkSum = checkSum & 0x00ff
    print(hex(checkSum))
    time.sleep(0.5)
```

"time"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.

각각의 data에 값을 넣고 Checksum을 확인합니다.

출력하고자 하는 디버깅 메시지를 "print()"에 입력합니다.
여기서는 "RaspberryPi Checksum Test 02"를 입력하였습니다.

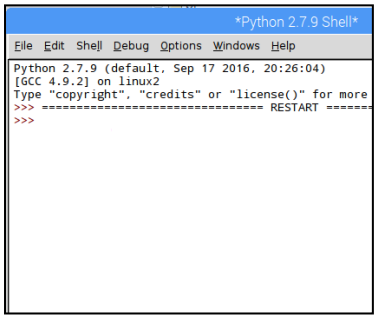
프로그램의 지속적인 반복 수행을 위해 "while True:"를 사용합니다.

정해진 data를 이용하여 계산된 체크섬 값을 Python Shell(콘솔)로 출력시킵니다.

이 동작은 500ms간격으로 반복됩니다.

▶ Checksum은 data1, data2, data3, data4, data5, data6의 Hex 값을 더하고, 더한 값과 0x00ff Hex 값을 비트 논리곱 하는 방식으로 구성이 되어있습니다.

예) data 1 = 0xa1, data2 = 0x64, data3 = 0x64, data4 = 0x64, data5 = 0x50, data6 = 0x05 일 경우
checkSum = 0xa1 + 0x64 + 0x64 + 0x64+ 0x50 + 0x05
= 0x0222
checkSum = 0x0222 & 0x00ff
= 0x0022 의 값을 가지게 됩니다.

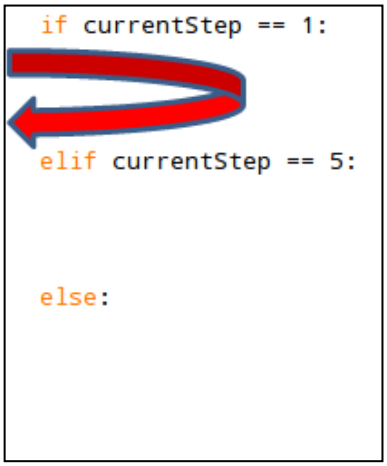


① Python 2(IDLE) 프로그램으로 Joystick_07_Checksum_02.py를 오픈합니다.
② Python 2 프로그램을 실행합니다.(Run Module 실행 또는 F5를 누릅니다.)
③ Python Shell이 실행됩니다.
④ 출력되는 디버깅 데이터를 확인합니다.
⑤ Python Shell에서 "Ctrl+C" 입력하여 프로그램을 종료합니다.

15. Joystick_08_if-elif

주어진 조건에 따라 실행되는 부분을 다르게 하고자 하는 경우 **"if ~ elif 문"**을 사용합니다.

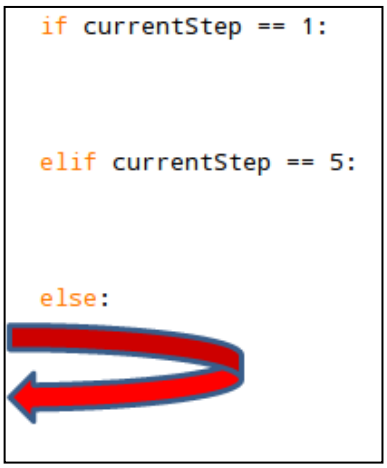
조건은 if나 elif 다음에 넣고, 실행되는 사항은 if나 elif 아래에 구성합니다.



조건인 "currentStep"의 값에 따라 실행되는 부분이 다릅니다.

"currentStep = 1"이 되면, "if currentStep == 1:"이 실행됩니다.

"currentStep = 5"가 되면, "elif currentStep == 5:"가 실행됩니다.



"currentStep = 3"이 되면, "else:"가 실행됩니다.

```
import time

#-----------------------------------------
#-----------------------------------------
currentStep = 1
#-----------------------------------------

print("RaspberryPi if-elif Test\n")

while True:
    if currentStep == 1:
        print("if-elif 1-step Operation")
        currentStep = 5
        time.sleep(0.5)
    elif currentStep == 5:
        print("if-elif 5-step Operation")
        currentStep += 1
        time.sleep(0.5)
    else:
        print("if-elif default Operation")
        time.sleep(0.5)
```

"time"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.

조건 저장 변수로 "currentStep"을 사용합니다.
currentStep의 초기값을 1로 설정합니다.

출력하고자 하는 디버깅 메시지를 "print()"에 입력합니다.
여기서는 "RaspberryPi if-elif Test"를 입력하였습니다.

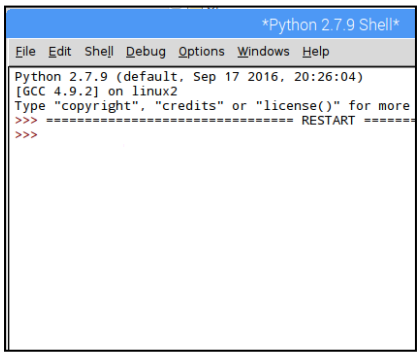
프로그램의 지속적인 반복 수행을 위해 "while True:"를 사용합니다.

"currentStep = 1"에 의해 "if currentStep == 1:"부분이 실행됩니다.
Python Shell(콘솔)로 "if-elif 1-step Operation"을 출력합니다.
"currentStep = 5"로 설정을 진행합니다.
약 500ms 대기합니다.
"if-elif"문을 종료합니다.

"currentStep = 5"에 의해 "elif currentStep == 5:"부분이 실행됩니다.
Python Shell(콘솔)로 "if-elif 5-step Operation"을 출력합니다.
"currentStep += 1"에 의해 "currentStep = 6"으로 설정을 진행합니다.
약 500ms 대기합니다.
"if-elif"문을 종료합니다.

"currentStep = 6"에 의해 "else:"부분이 실행됩니다.
Python Shell(콘솔)로 "if-elif default Operation"을 출력합니다.
약 500ms 대기합니다.
"if-elif"문을 종료합니다.

"currentStep = 6"에 의해 "else:"부분이 계속 실행됩니다.



① Python 2(IDLE) 프로그램으로 Joystick_08_if-elif.py를 오픈합니다.
② Python 2 프로그램을 실행합니다.(Run Module 실행 또는 F5를 누릅니다.)
③ Python Shell이 실행됩니다.
④ 출력되는 디버깅 데이터를 확인합니다.
⑤ Python Shell에서 "Ctrl+C" 입력하여 프로그램을 종료합니다.

16. Joystick_09_find_01

시리얼 데이터에서 특별한 문자열을 찾고자 하는 경우 파이썬에서 제공하는 내부 함수를 사용하면 쉽게 찾을 수 있습니다.

"len()" => 문자열의 길이를 반환합니다.
: len()은 비교하는 시리얼 데이터의 길이를 반환합니다.
"find(찾고 싶은 문자열, 시작 인덱스, 종료 인덱스)" => 찾고 싶은 문자열이 있는 경우, 인덱스를 반환합니다.
=> 문자열이 없는 경우, -1을 반환합니다.
=> "시작 인덱스" / "종료 인덱스"는 생략 가능합니다.
:find("12", 0, 10)은 주어진 문자열의 0에서 9번째 사이에 "12"가 있는지를 검사합니다.
:주어진 문자열이 "0012000000"이면 반환되는 인덱스는 2 입니다.
:주어진 문자열이 "0000000012"이면 반환되는 인덱스는 8 입니다.
:주어진 문자열이 "0000000000"이면 반환되는 인덱스는 -1 입니다.

```
testString = "1234567890"
uartLength = 0
#----------------------------------------

print("RaspberryPi Find Test 01\n")

print(testString.find("ab",0,10))

print(testString.find("12",0,3))

uartLength = len(testString)
print("Length: %d" %(uartLength-2))

print(testString.find("90",uartLength-2))

while True:
    continue
```

문자열 저장 변수로 testString을 사용합니다.
testString에 "1234567890"을 저장합니다.
문자열 길이 저장 변수로 uartLength를 사용합니다.
초기값을 0으로 저장합니다.

출력하고자 하는 디버깅 메시지를 "print()"에 입력합니다.
여기서는 "RaspberryPi Find Test 01"을 입력하였습니다.

testString 변수에 "ab"가 있는지 알아보기 위해
testString.find("ab",0,10)을 사용합니다.
반환된 인덱스를 확인하기 위해 "print()"를 사용합니다.

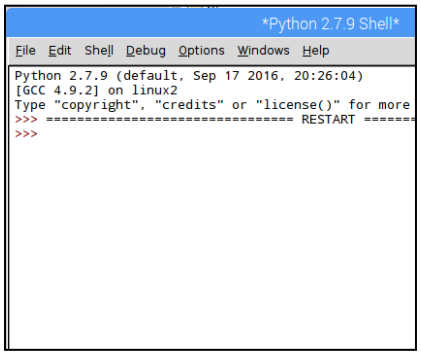
testString 변수에 "12"가 있는지 알아보기 위해
testString.find("12",0,3)을 사용합니다.
반환된 인덱스를 확인하기 위해 "print()"를 사용합니다.

testString 변수에 저장된 문자열의 길이를 반환하기 위해 "len(testString)"을 사용하고 반환된 값을 uartLength에 저장합니다. 반환된 길이를 확인하기 위해 "print()"를 사용합니다.

testString 변수에 "90"이 있는지 알아보기 위해 testString,find("90",uartLength-2)를 사용합니다. 반환된 인덱스를 확인하기 위해 "print()"를 사용합니다.

프로그램의 지속적인 반복 수행을 위해 "while True: continue"를 사용합니다.

"while True: continue"에 의해 Joystick_09_Find_01 프로그램은 아무런 작업을 하지 않으면서 수행됩니다.



① Python 2(IDLE) 프로그램으로 Joystick_09_Find_01.py를 오픈합니다.
② Python 2 프로그램을 실행합니다.(Run Module 실행 또는 F5를 누릅니다.)
③ Python Shell이 실행됩니다.
④ 출력되는 디버깅 데이터를 확인합니다.
⑤ Python Shell에서 "Ctrl+C" 입력하여 프로그램을 종료합니다.

17. Joystick_09_find_02

```
testString = "1234567890"
uartLength = 0
#-------------------------------------------------

print("RaspberryPi Find Test 02\n")

uartLength = len(testString)
print("Length: %d" %uartLength)

if testString.find("12",0,2) == 0:
    print("Start find OK")

if testString.find("90",uartLength-2) == uartLength-2:
    print("End find OK")

while True:
    continue
```

문자열 저장 변수로 testString을 사용합니다.
testString에 "1234567890"을 저장합니다.
문자열 길이 저장 변수로 uartLength를 사용합니다.
초기값을 0으로 저장합니다.

출력하고자 하는 디버깅 메시지를 "print()"에 입력합니다.
여기서는 "RaspberryPi Find Test 02"를 입력하였습니다.

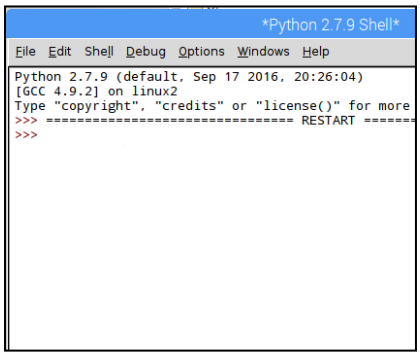
testString 변수에 저장된 문자열의 길이를 반환하기 위해 "len(testString)"을 사용하고 반환된 값을 uartLength에 저장합니다. 반환된 길이를 확인하기 위해 "print()"를 사용합니다.

testString 변수에 "12"가 있는지 알아보기 위해 testString.find("12",0,2)을 사용합니다. 반환된 인덱스가 0인지 비교하고, 0인 경우 print("Start find OK")를 출력합니다. 반환된 인덱스가 0이라는 것은 문자열의 제일 앞에 찾고자 하는 문자가 있다는 것입니다.

testString 변수에 "90"이 있는지 알아보기 위해 testString.find("90",uartLength-2)을 사용합니다. 반환된 인덱스가 "uartLength-2"인지 비교하고, 같은 경우 print("End find OK")를 출력합니다. 반환된 인덱스가 "uartLength-2"라는 것은 문자열의 마지막에서 2번째에 찾고자 하는 문자가 있다는 것입니다.

프로그램의 지속적인 반복 수행을 위해 "while True: continue"를 사용합니다.

"while True: continue"에 의해 Joystick_09_Find_02 프로그램은 아무런 작업을 하지 않으면서 수행됩니다.



① Python 2(IDLE) 프로그램으로 Joystick_09_Find_02.py를 오픈합니다.
② Python 2 프로그램을 실행합니다.(Run Module 실행 또는 F5를 누릅니다.)
③ Python Shell이 실행됩니다.
④ 출력되는 디버깅 데이터를 확인합니다.
⑤ Python Shell에서 "Ctrl+C" 입력하여 프로그램을 종료합니다.

18. Joystick_10_Serial_01

라즈베리파이는 하드웨어 시리얼 포트가 1개 있습니다.

하드웨어 시리얼 포트를 사용하고자 하는 경우 파이썬에서 제공하는 외부 함수를 사용해야 합니다.

"inWaiting()" => 하드웨어 시리얼 포트로 입력된 데이터가 있는 경우 1(참)을 반환합니다.
"read()" => 하드웨어 시리얼 포트로 입력된 데이터를 읽어옵니다.
"write()" => 하드웨어 시리얼 포트로 시리얼 데이터를 출력합니다..

```
import time
import serial

#-----------------------------------------------
ser = serial.Serial('/dev/ttyS0',9600,timeout=0.001)

#-----------------------------------------------
testString = ""

#-----------------------------------------------

print("RaspberryPi Serial Test 01\n")

while True:
    if ser.inWaiting():
        time.sleep(0.5)
        while ser.inWaiting():
            testString += ser.read()

        print(testString)
        testString = ""
```

"time"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.
"serial"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.

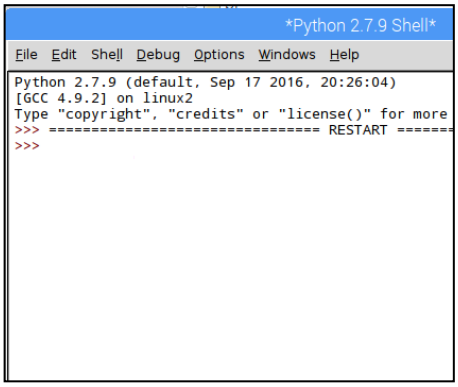
"serial" 라이브러리를 사용하기 위해 "인자"를 생성합니다.
생성된 하드웨어 시리얼 포트는 9600bps로 동작됩니다.

문자열 저장 변수로 testString을 사용합니다.

출력하고자 하는 디버깅 메시지를 "print()"에 입력합니다.
여기서는 "RaspberryPi Serial Test 01"을 입력하였습니다.

프로그램의 지속적인 반복 수행을 위해 "while True:"를 사용합니다.

하드웨어 시리얼 포트로 입력된 데이터가 있으면 "ser.inWaiting()"이 1(참)을 반환합니다.
입력된 시리얼 데이터가 있으면 약 500ms를 대기합니다.
입력된 시리얼 데이터가 있는 동안(while ser.inWaiting():) "ser.read()"에 의해 읽어온 시리얼 데이터를 "testString"에 저장합니다.
"testString"에 저장된 데이터를 확인하기 위해 "print()"를 사용합니다.
"testString"을 초기화합니다.



① 라즈베리파이 확장보드에 조이스틱 쉴드를 장착합니다.
② 조이스틱 쉴드에 BLE 장치를 장착합니다.
③ Python 2(IDLE) 프로그램으로 Joystick_10_Serial_01.py를 오픈합니다.
④ Python 2 프로그램을 실행합니다.(Run Module 실행 또는 F5를 누릅니다.)
⑤ Python Shell이 실행됩니다.
⑥ 출력되는 디버깅 데이터를 확인합니다.
⑦ 라즈베리파이 확장보드의 전원을 ON 합니다.
⑧ 출력되는 디버깅 데이터를 확인합니다.
⑨ Python Shell에서 "Ctrl+C" 입력하여 프로그램을 종료합니다.
⑩ 라즈베리파이 확장보드의 전원을 OFF 합니다.

※ BLE 장치에서 출력되는 시리얼 데이터를 확인하기 위해 라즈베리파이 확장보드의 전원을 나중에 ON 합니다.

19. Joystick_10_Serial_02

라즈베리파이는 하드웨어 시리얼 포트가 1개 있습니다.
보통 하드웨어 시리얼 포트는 다른 장치와 시리얼 통신용으로 사용됩니다.

파이썬은 Python Shell(콘솔)을 이용하여 디버깅 메시지를 출력하거나, 키보드 데이터의 입력이 가능합니다.
Python Shell(콘솔)을 이용하여 입력된 키보드 데이터를 하드웨어 시리얼 포트로 출력하고, 하드웨어 시리얼 포트로 입력된 데이터를 Python Shell(콘솔)에 출력할 수 있습니다.

이 기능은 라즈베리파이를 가운데 두고 Python Shell(콘솔)을 이용하여 명령어를 입력하면 라즈베리파이가 입력된 명령어를 하드웨어 시리얼 포트로 출력하고, 하드웨어 시리얼 포트로 입력된 응답 값을 라즈베리파이가 Python Shell(콘솔)로 출력하는 것입니다.

Python Shell(콘솔)에 AT Command를 입력하면, 라즈베리파이는 입력 된 AT Command를 하드웨어 시리얼 포트를 통해 BLE 장치(FBL780_Serial)에 전달합니다.
하드웨어 시리얼 포트를 통해 BLE 장치(FBL780_Serial)가 응답 값을 전달하면, 라즈베리파이는 입력된 응답 값을 Python Shell(콘솔)에 출력합니다.

```
import time
import serial

#---------------------------------------------------
ser = serial.Serial('/dev/ttyS0',9600,timeout=0.001)

#---------------------------------------------------
testString = ""

#---------------------------------------------------

print("RaspberryPi Serial Test 02\n")

while True:
    if ser.inWaiting():
        testString = ""
        time.sleep(0.5)
        while ser.inWaiting():
            testString += ser.read()

        print(testString)
        testString = ""

    testString = raw_input("Enter AT Command: ")
    ser.write(testString)
    ser.write("\r")
    print("Wait Response Command for 3s...")
    time.sleep(3)
```

"time"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.
"serial"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.

"serial" 라이브러리를 사용하기 위해 "인자"를 생성합니다.
생성된 하드웨어 시리얼 포트는 9600bps로 동작됩니다.

문자열 저장 변수로 testString을 사용합니다.

출력하고자 하는 디버깅 메시지를 "print()"에 입력합니다.
여기서는 "RaspberryPi Serial Test 02"를 입력하였습니다.

프로그램의 지속적인 반복 수행을 위해 "while True:"를 사용합니다.

하드웨어 시리얼 포트로 입력된 데이터가 있으면 "ser.inWaiting()"은 1(참)을 반환합니다.

입력된 시리얼 데이터가 있으면 약 500ms를 대기합니다.

입력된 시리얼 데이터가 있는 동안(while ser.inWaiting():) "ser.read()"에 의해 읽어온 시리얼 데이터를 "testString"에 저장합니다.
"testString"에 저장된 데이터를 확인하기 위해 "print()"를 사용합니다.
"testString"을 초기화합니다.

"raw_input()"을 사용하여 Python Shell(콘솔)로부터 키보드 데이터를 입력 받습니다. 키보드 데이터 입력이 완료되면 "엔터키"를 입력합니다.
키보드로부터 입력된 데이터는 "testString"에 저장됩니다.
"ser.write(testString)"을 사용하여 "testString"에 저장되어 있는 키보드 데이터를 하드웨어 시리얼 포트로 출력합니다.
"ser.write("₩r")"을 사용하여 0x0d를 하드웨어 시리얼 포트로 출력합니다.
약 3초의 시간을 대기합니다. (응답 값이 발생되는 시간을 기다리기 위해 약 3초를 대기합니다.)

"조이스틱 쉴드"에 사용하는 BLE 장치(FBL780_Serial)는 Central Role으로 동작해야 합니다.

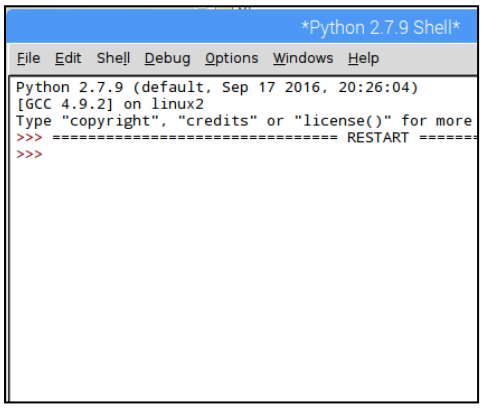
▶ BLE 장치가 Central Role으로 동작하는지 확인하는 방법은 아래와 같습니다.
(1) Python Shell(콘솔)에 "AT+GETROLE" 명령어를 입력하시면 됩니다.
(2) 응답값으로 "C" 가 송신되면 BLE 장치가 Central Role으로 동작하는 것입니다.

▶ BLE 장치가 Central Role으로 동작하지 않는 경우 설정하는 방법은 아래와 같습니다.
(1) Python Shell(콘솔)에 "AT+SETROLEC" 명령어를 입력하시면 됩니다.
(2) 응답값으로 "OK" 가 송신되면 Python Shell(콘솔)에 "ATZ" 명령어를 입력하시면 됩니다.

"조이스틱 쉴드"에 사용하는 BLE 장치(FBL780_Serial)는 SMODE가 0으로 동작해야 합니다.

▶ BLE 장치가 SMODE 0으로 동작하는지 확인하는 방법은 아래와 같습니다.
(1) Python Shell(콘솔)에 "AT+GETSMODE" 명령어를 입력하시면 됩니다.
(2) 응답값으로 "0" 이 송신되면 BLE 장치가 SMODE 0으로 동작하는 것입니다.

▶ BLE 장치가 SMODE 0으로 동작하지 않을 경우 설정하는 방법은 아래와 같습니다.
(1) Python Shell(콘솔)에 "AT+SETSMODE0" 명령어를 입력하시면 됩니다.
(2) 응답값으로 "OK" 가 송신되면 Python Shell(콘솔)에 "ATZ" 명령어를 입력하시면 됩니다.



① 라즈베리파이 확장보드에 조이스틱 쉴드를 장착합니다.
② 조이스틱 쉴드에 BLE 장치를 장착합니다.
③ 라즈베리파이 확장보드의 전원을 ON 합니다.
④ Python 2(IDLE) 프로그램으로 Joystick_10_Serial_02.py를 오픈합니다.
⑤ Python 2 프로그램을 실행합니다.(Run Module 실행 또는 F5를 누릅니다.)
⑥ Python Shell이 실행됩니다.
⑦ 출력되는 디버깅 데이터를 확인합니다.
⑧ Python Shell에 AT Command를 입력합니다.
⑨ 출력되는 디버깅 데이터를 확인합니다.
⑩ Python Shell에서 "Ctrl+C" 입력하여 프로그램을 종료합니다.
⑪ 라즈베리파이 확장보드의 전원을 OFF 합니다.

20. Joystick_11_AT-Command

프로그램 상에서 하드웨어 시리얼 포트를 통해 직접 BLE 장치(FBL780_Serial)에 명령어를 전달합니다.

```
import time
import serial

#---------------------------------------------------
ser = serial.Serial('/dev/ttyS0',9600,timeout=0.001)

#---------------------------------------------------
testString = ""

#---------------------------------------------------

print("RaspberryPi AT Command Test\n")

time.sleep(0.5)
ser.write("at")
ser.write("\r")
time.sleep(0.5)

while True:
    if ser.inWaiting():
        testString = ""
        time.sleep(0.5)
        while ser.inWaiting():
            testString += ser.read()

        print(testString)
        testString = ""

    testString = raw_input("Enter AT Command: ")
    ser.write(testString)
    ser.write("\r")
    print("Wait Response Command for 3s...")
    time.sleep(3)
```

"time"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.
"serial"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.

"serial" 라이브러리를 사용하기 위해 "인자"를 생성합니다.
생성된 하드웨어 시리얼 포트는 9600bps로 동작됩니다.

문자열 저장 변수로 testString을 사용합니다.

출력하고자 하는 디버깅 메시지를 "print()"에 입력합니다..
여기서는 "RaspberryPi AT Command Test"를 입력하였습니다.

BLE 장치가 시작되기를 기다리기 위해 약 500ms 대기합니다.

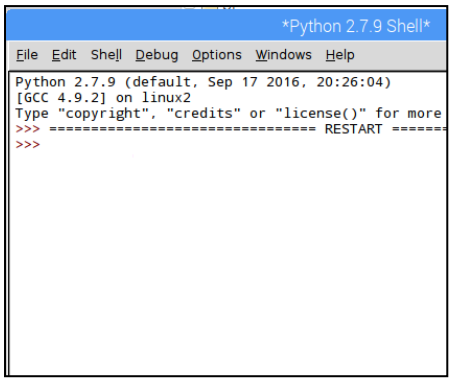
하드웨어 시리얼 포트로 AT Command를 전달합니다.
전달하고자 하는 AT Command "at"를 "ser.write()"에 입력합니다.

AT Command 전달이 완료된 것을 알리기 위해 "₩r"을 "ser.write()"에 입력합니다.

프로그램의 지속적인 반복 수행을 위해 "while True:"를 사용합니다.

하드웨어 시리얼 포트로 입력된 데이터가 있으면 "ser.inWaiting()"이 1(참)을 반환합니다.
입력된 시리얼 데이터가 있으면 약 500ms를 대기합니다.
입력된 시리얼 데이터가 있는 동안(while ser.inWaiting():) "ser.read()"에 의해 읽어온 시리얼 데이터를 "testString"에 저장합니다.
"testString"에 저장된 데이터를 확인하기 위해 "print()"를 사용합니다.
"testString"을 초기화합니다.

"raw_input()"을 사용하여 Python Shell(콘솔)로부터 키보드 데이터를 입력 받습니다. 키보드 데이터 입력이 완료되면 "엔터키"를 입력합니다.
키보드로부터 입력된 데이터는 "testString"에 저장됩니다.
"ser.write(testString)"을 사용하여 "testString"에 저장되어 있는 키보드 데이터를 하드웨어 시리얼 포트로 출력합니다.
"ser.write("₩r")"을 사용하여 0x0d를 하드웨어 시리얼 포트로 출력합니다.
약 3초의 시간을 대기합니다. (응답 값이 발생되는 시간을 기다리기 위해 약 3초를 대기합니다.)



① 라즈베리파이 확장보드에 조이스틱 쉴드를 장착합니다.
② 조이스틱 쉴드에 BLE 장치를 장착합니다.
③ 라즈베리파이 확장보드의 전원을 ON 합니다.
④ Python 2(IDLE) 프로그램으로 Joystick_11_AT-Command.py를 오픈합니다.
⑤ Python 2 프로그램을 실행합니다.(Run Module 실행 또는 F5를 누릅니다.)
⑥ Python Shell이 실행됩니다.
⑦ 출력되는 디버깅 데이터를 확인합니다.
⑧ Python Shell에 AT Command를 입력합니다.
⑨ 출력되는 디버깅 데이터를 확인합니다.
⑩ Python Shell에서 "Ctrl+C" 입력하여 프로그램을 종료합니다.
⑪ 라즈베리파이 확장보드의 전원을 OFF 합니다.

21. Joystick_12_BT-Connect

"if ~ elif"문과 "PIO Input"기능을 이용하여 BLE 장치(FBL780_Serial)에 AT Command를 전달하고, 펌테크 드론과 "Bluetooth 연결 / 종료"를 진행합니다.

```
import serial
import RPi.GPIO as GPIO

#-------------------------------------------------
ser = serial.Serial('/dev/ttyS0',9600,timeout=0.001)

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

#-------------------------------------------------
pio_list = [22, 27, 17, 23, 24, 25]

for i in range(6):
    GPIO.setup(pio_list[i], GPIO.IN)

currentStep = 0

#-------------------------------------------------

while True:
    if currentStep == 0:
        print("RaspberryPi BT Connect Test\n")
        currentStep += 1
    elif currentStep == 1:
        if GPIO.input(pio_list[4]) == 0:
            print("Pressed Connect Button")
            ser.write("atd")
            ser.write("083a5c1f015b")
            ser.write("\r")
            currentStep += 1
    elif currentStep == 2:
        if GPIO.input(pio_list[5]) == 0:
            print("Pressed Disconnect Button")
            ser.write("ath")
            ser.write("\r")
            currentStep = 1
```

"serial"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.
"PIO"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.
"serial" 라이브러리를 사용하기 위해 "인자"를 생성합니다.
생성된 하드웨어 시리얼 포트는 9600bps로 동작됩니다.
"PIO" 포트의 번호 사용 방식을 "BCM"으로 설정합니다.
"PIO" 포트의 사용 중에 프로그램을 종료하는 경우 발생되는 warning을 출력되지 않도록 설정합니다.
"PIO" 포트로 사용할 포트를 'pio_list"라는 리스트에 설정합니다.
GPIO.setup()을 사용하여 6개의 디지털 포트를 입력으로 설정합니다.

조건문에 사용할 변수를 "currentStep"으로 정의하고 초기값을 0으로 설정합니다.

프로그램의 지속적인 반복 수행을 위해 "while True:"를 사용합니다.

"currentStep = 0"에 의해 "if currentStep == 0:"부분이 실행됩니다.
Python Shell(콘솔)로 "RaspberryPi BT Connect Test" 출력합니다.

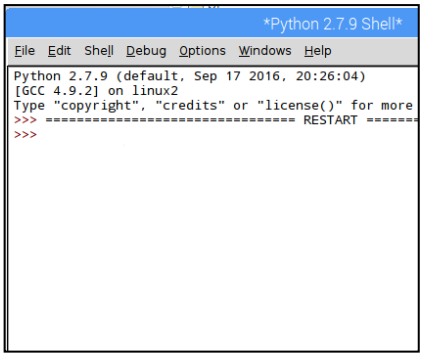
"currentStep = 1"로 설정을 진행합니다.
"if-elif"문을 종료합니다.

"currentStep = 1"에 의해 "elif currentStep == 1:"부분이 실행됩니다.
"if GPIO.input(pio_list[4]) == 0"을 사용하여 디지털 4번째 포트가 High 상태에서 Low 상태로 변경되었는지 체크를 합니다. (포트의 시작은 0번째 포트부터 시작합니다.)
4번째 포트가 Low 상태로 변경된 경우, Python Shell(콘솔)로 "Pressed Connect Button"을 출력합니다.
하드웨어 시리얼 포트로 "atd083a5c1f015b₩r"을 출력합니다.
"currentStep = 2"로 설정을 진행합니다.
"if-elif"문을 종료합니다.

"currentStep = 2"에 의해 "elif currentStep == 2:"부분이 실행됩니다.
"if GPIO.input(pio_list[5]) == 0"을 사용하여 디지털 5번째 포트가 High 상태에서 Low 상태로 변경되었는지 체크를 합니다. (포트의 시작은 0번째 포트부터 시작합니다.)
5번째 포트가 Low 상태로 변경된 경우, Python Shell(콘솔)로 "Pressed Disconnect Button"을 출력합니다.
하드웨어 시리얼 포트로 "ath₩r"을 출력합니다.
"currentStep = 1"로 설정을 진행합니다.
"if-elif"문을 종료합니다.

"currentStep = 1"에 의해 "if currentStep == 1:"부분부터 다시 실행됩니다.

※ "atd"명령어 뒤의 숫자(083a5c1f015b)는 연결할 펌테크 드론에 따라 다르게 입력되어야 합니다.



① 라즈베리파이 확장보드에 조이스틱 쉴드를 장착합니다.
② 조이스틱 쉴드에 BLE 장치를 장착합니다.
③ 라즈베리파이 확장보드의 전원을 ON 합니다.
④ Python 2(IDLE) 프로그램으로 Joystick_12_BT-Connect.py를 오픈합니다.
⑤ Python 2 프로그램을 실행합니다.(Run Module 실행 또는 F5를 누릅니다.)
⑥ Python Shell이 실행됩니다.
⑦ 펌테크 드론의 전원을 ON 합니다.
⑧ 4번 스위치 눌러서 블루투스 연결을 진행합니다. (연결되면 **"조이스틱 쉴드"**의 Status LED가 ON 됩니다.)
⑨ 출력되는 디버깅 데이터를 확인합니다.
⑩ 5번 스위치 눌러서 블루투스 연결 종료를 진행합니다.
⑪ 출력되는 디버깅 데이터를 확인합니다.
⑫ Python Shell에서 "Ctrl+C" 입력하여 프로그램을 종료합니다.
⑬ 라즈베리파이 확장보드의 전원을 OFF 합니다.
⑭ 펌테크 드론의 전원을 OFF 합니다.

22. Joystick_13_Check-Message_01

BLE 장치(FBL780_Serial)에 AT Command 전달 후, 원하는 응답 값 체크를 진행합니다.

"if ~ elif"문을 이용하여 순차적으로 진행하면서 AT Command를 전달하고 응답 값을 체크합니다.

```
import serial
import RPi.GPIO as GPIO
import time

#-------------------------------------------------
ser = serial.Serial('/dev/ttyS0',9600,timeout=0.001)

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

currentStep = 0
uartString = ""
uartLength = 0

#-------------------------------------------------
pio_list = [22, 27, 17, 23, 24, 25]

for i in range(6):
    GPIO.setup(pio_list[i], GPIO.IN)

while True:
    if currentStep == 0:
        print("RaspberryPi Check Message Test 01\n")
        currentStep += 1

    elif currentStep == 1:
        if GPIO.input(pio_list[4]) == 0:
            print("Pressed Connect Button")
            ser.write("atd")
            ser.write("083a5c1f015b")
            ser.write("\r")
            time.sleep(0.3)
            uartString = ""
            currentStep += 1

    elif currentStep == 2:
        if ser.inWaiting():
            uartString += ser.read()
            uartLength = len(uartString)
            if uartLength > 4 and uartString.find("\r\n",0,2) == 0 \
            and uartString.find("\r\n",uartLength-2) == uartLength - 2:
                currentStep += 1
```

"serial"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.
"PIO"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.
"time"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.
"serial" 라이브러리를 사용하기 위해 "인자"를 생성합니다.
생성된 하드웨어 시리얼 포트는 9600bps로 동작됩니다.
"PIO" 포트의 번호 사용 방식을 "BCM"으로 설정합니다.
"PIO" 포트의 사용 중에 프로그램을 종료하는 경우 발생되는 warning을 출력되지 않도록 설정합니다.
조건문에 사용할 변수를 "currentStep"으로 정의하고 초기값을 0으로 설정합니다.
문자열을 저장할 변수를 "uartString"으로 선언하고 초기화합니다.

문자열 길이를 저장할 변수를 "uartLength"로 정의하고 초기값을 0으로 설정합니다.

"PIO" 포트로 사용할 포트를 'pio_list"라는 리스트에 설정합니다.

GPIO.setup()을 사용하여 6개의 디지털 포트를 입력으로 설정합니다.

프로그램의 지속적인 반복 수행을 위해 "while True:"를 사용합니다.

"currentStep = 0"에 의해 "if currentStep == 0:"부분이 실행됩니다.
Python Shell(콘솔)로 "RaspberryPi Check Message Test 01"을 출력합니다.
"currentStep = 1"로 설정을 진행합니다.
"if-elif"문을 종료합니다.

"currentStep = 1"에 의해 "elif currentStep == 1:"부분이 실행됩니다.
"if GPIO.input(pio_list[4]) == 0"을 사용하여 디지털 4번째 포트가 High 상태에서 Low 상태로 변경되었는지 체크를 합니다. (포트의 시작은 0번째 포트부터 시작합니다.)
4번째 포트가 Low 상태로 변경된 경우, Python Shell(콘솔)로 "Pressed Connect Button"을 출력합니다.
하드웨어 시리얼 포트로 "atd083a5c1f015b₩r"을 출력합니다.
약 300ms 시간 대기합니다.
"uartString"을 초기화합니다.
"currentStep = 2"로 설정을 진행합니다.
"if-elif"문을 종료합니다.

"currentStep = 2"에 의해 "elif currentStep == 2:"부분이 실행됩니다.
하드웨어 시리얼 포트로 입력된 데이터가 있으면 "ser.inWaiting()"이 1(참)을 반환합니다.
입력된 시리얼 데이터가 있으면 "ser.read()"로 읽어서 "uartString"에 저장합니다.
"uartString"에 저장된 데이터의 길이를 구해 "uartLength"에 저장합니다.
"uartString"에 저장된 문자열의 길이가 4보다 크고, "₩r₩n"으로 시작되며, "₩r₩n"으로 끝나는지 체크합니다.
(펌테크 제품은 AT Command를 입력한 경우, 모든 응답 값이 4보다 크고 "₩r₩n"으로 시작해서 "₩r₩n"으로 끝납니다.)
응답 값이 4보다 크고 "₩r₩n"으로 시작해서 "₩r₩n"으로 종료되는 경우, currentStep = 3으로 설정을 진행합니다.
"if-elif"문을 종료합니다.

```
elif currentStep == 3:
    if uartString.find("\r\nOK\r\n",0,6) == 0:
        print("Received OK")
        time.sleep(0.3)
        uartString = ""
        currentStep += 1

elif currentStep == 4:
    if ser.inWaiting():
        uartString += ser.read()
        uartLength = len(uartString)
        if uartLength > 4 and uartString.find("\r\n",0,2) == 0 \
        and uartString.find("\r\n",uartLength-2) == uartLength - 2:
            currentStep += 1

elif currentStep == 5:
    if uartString.find("\r\nCONNECT ",0,10) == 0:
        print("Received CONNECT")
        time.sleep(0.3)
        uartString = ""
        currentStep += 1

elif currentStep == 6:
    if ser.inWaiting():
        ser.read()

    if GPIO.input(pio_list[5]) == 0:
        print("Pressed Disconnect Button")
        ser.write("ath")
        ser.write("\r")
        time.sleep(0.3)
        uartString = ""
        currentStep += 1

elif currentStep == 7:
    if ser.inWaiting():
        uartString += ser.read()
        uartLength = len(uartString)
        if uartLength > 4 and uartString.find("\r\n",0,2) == 0 \
        and uartString.find("\r\n",uartLength-2) == uartLength - 2:
            currentStep += 1
```

"currentStep = 3"에 의해 "elif currentStep == 3:"부분이 실행됩니다.
"find()"를 사용하여 "uartString"변수에 저장된 문자열이 "₩r₩nOK₩r₩n"인지 체크합니다.
비교 값이 맞는 경우, Python Shell(콘솔)로 "Received OK"를 출력합니다.
약 300ms 시간 대기합니다.
"uartString"을 초기화합니다..
"currentStep = 4"로 설정을 진행합니다.
"if-elif"문을 종료합니다.

"currentStep = 4"에 의해 "elif currentStep == 4:"부분이 실행됩니다.
하드웨어 시리얼 포트로 입력된 데이터가 있으면 "ser.inWaiting()"이 1(참)을 반환합니다.
입력된 시리얼 데이터가 있으면 "ser.read()"로 읽어서 "uartString"에 저장합니다.
"uartString"에 저장된 데이터의 길이를 구해 "uartLength"에 저장합니다.
"uartString"에 저장된 문자열의 길이가 4보다 크고, "₩r₩n"으로 시작되며, "₩r₩n"으로 끝나는지 체크합니다.
(펌테크 제품은 AT Command를 입력한 경우, 모든 응답 값이 4보다 크고 "₩r₩n"으로 시작해서 "₩r₩n"으로 끝납니다.)
응답 값이 4보다 크고 "₩r₩n"으로 시작해서 "₩r₩n"으로 종료되는 경우, currentStep = 5로 설정을 진행합니다.
"if-elif"문을 종료합니다.

"currentStep = 5"에 의해 "elif currentStep == 5:"부분이 실행됩니다.
"find()"를 사용하여 "uartString"변수에 저장된 문자열이 "₩r₩nCONNECT "인지 체크합니다.
비교 값이 맞는 경우, Python Shell(콘솔)로 "Received CONNECT"를 출력합니다.
약 300ms 시간 대기합니다.
"uartString"을 초기화합니다.
"currentStep = 6"으로 설정을 진행합니다.
"if-elif"문을 종료합니다.

"currentStep = 6"에 의해 "elif currentStep == 6:"부분이 실행됩니다.
하드웨어 시리얼 포트로 입력된 데이터가 있으면 "ser.inWaiting()"이 1(참)을 반환합니다.
입력된 시리얼 데이터가 있으면 "ser.read()"로 읽기만 하고 아무런 처리를 하지 않습니다.
"if GPIO.input(pio_list[5]) == 0"을 사용하여 디지털 5번째 포트가 High 상태에서 Low 상태로 변경되었는지 체크를 합니다. (포트의 시작은 0번째 포트부터 시작합니다.)
5번째 포트가 Low 상태로 변경된 경우, Python Shell(콘솔)로 "Pressed Disconnect Button"을 출력합니다.

하드웨어 시리얼 포트로 "ath₩r"을 출력합니다.
약 300ms 시간 대기합니다.
"uartString"을 초기화합니다.
"currentStep = 7"로 설정을 진행합니다.
"if-elif"문을 종료합니다.

"currentStep = 7"에 의해 "elif currentStep == 7:"부분이 실행됩니다.
하드웨어 시리얼 포트로 입력된 데이터가 있으면 "ser.inWaiting()"이 1(참)을 반환합니다.
입력된 시리얼 데이터가 있으면 "ser.read()"로 읽어서 "uartString"에 저장합니다.
"uartString"에 저장된 데이터의 길이를 구해 "uartLength"에 저장합니다.
"uartString"에 저장된 문자열의 길이가 4보다 크고, "₩r₩n"으로 시작되며, "₩r₩n"으로 끝나는지 체크합니다.
(펌테크 제품은 AT Command를 입력한 경우, 모든 응답 값이 4보다 크고 "₩r₩n"으로 시작해서 "₩r₩n"으로 끝납니다.)
응답 값이 4보다 크고 "₩r₩n"으로 시작해서 "₩r₩n"으로 종료되는 경우, currentStep = 8로 설정을 진행합니다.
"if-elif"문을 종료합니다.

```
elif currentStep == 8:
    if uartString.find("\r\nOK\r\n",0,6) == 0:
        print("Received OK")
        time.sleep(0.3)
        uartString = ""
        currentStep += 1

elif currentStep == 9:
    if ser.inWaiting():
        uartString += ser.read()
        uartLength = len(uartString)
        if uartLength > 4 and uartString.find("\r\n",0,2) == 0 \
        and uartString.find("\r\n",uartLength-2) == uartLength - 2:
            currentStep += 1

elif currentStep == 10:
    if uartString.find("\r\nDISCONNECT",0,12) == 0:
        print("Received DISCONNECT")
        time.sleep(0.3)
        uartString = ""
        currentStep += 1

elif currentStep == 11:
    if ser.inWaiting():
        uartString += ser.read()
        uartLength = len(uartString)
        if uartLength > 4 and uartString.find("\r\n",0,2) == 0 \
        and uartString.find("\r\n",uartLength-2) == uartLength - 2:
            currentStep += 1

elif currentStep == 12:
    if uartString.find("\r\nREADY",0,7) == 0:
        print("Received READY")
        time.sleep(0.3)
        uartString = ""
        currentStep = 1
```

"currentStep = 8"에 의해 "elif currentStep == 8:"부분이 실행됩니다.
"find()"를 사용하여 "uartString"변수에 저장된 문자열이 "₩r₩nOK₩r₩n"인지 체크합니다.
비교 값이 맞는 경우, Python Shell(콘솔)로 "Received OK"를 출력합니다.
약 300ms 시간 대기합니다.
"uartString"을 초기화합니다.
"currentStep = 9"로 설정을 진행합니다.
"if-elif"문을 종료합니다.

"currentStep = 9"에 의해 "elif currentStep == 9:"부분이 실행됩니다.
하드웨어 시리얼 포트로 입력된 데이터가 있으면 "ser.inWaiting()"이 1(참)을 반환합니다.
입력된 시리얼 데이터가 있으면 "ser.read()"로 읽어서 "uartString"에 저장합니다.
"uartString"에 저장된 데이터의 길이를 구해 "uartLength"에 저장합니다.

"uartString"에 저장된 문자열의 길이가 4보다 크고, "₩r₩n"으로 시작되며, "₩r₩n"으로 끝나는지 체크합니다.
(펌테크 제품은 AT Command를 입력한 경우, 모든 응답 값이 4보다 크고 "₩r₩n"으로 시작해서 "₩r₩n"으로 끝납니다.)
응답 값이 4보다 크고 "₩r₩n"으로 시작해서 "₩r₩n"으로 종료되는 경우, currentStep = 10으로 설정을 진행합니다.
"if-elif"문을 종료합니다.

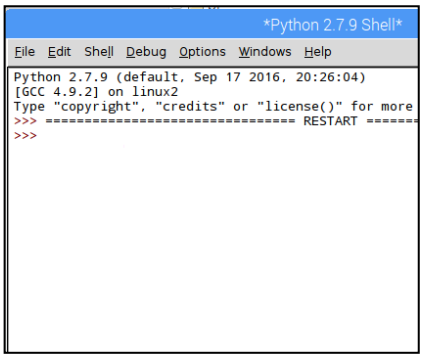
"currentStep = 10"에 의해 "elif currentStep == 10:"부분이 실행됩니다.
"find()"를 사용하여 "uartString"변수에 저장된 문자열이 "₩r₩nDISCONNECT"인지 체크합니다.
비교 값이 맞는 경우, Python Shell(콘솔)로 "Received DISCONNECT"를 출력합니다.
약 300ms 시간 대기합니다.
"uartString"을 초기화합니다.
"currentStep = 11"로 설정을 진행합니다.
"if-elif"문을 종료합니다.

"currentStep = 11"에 의해 "elif currentStep == 11:"부분이 실행됩니다.
하드웨어 시리얼 포트로 입력된 데이터가 있으면 "ser.inWaiting()"이 1(참)을 반환합니다.
입력된 시리얼 데이터가 있으면 "ser.read()"로 읽어서 "uartString"에 저장합니다.
"uartString"에 저장된 데이터의 길이를 구해 "uartLength"에 저장합니다.
"uartString"에 저장된 문자열의 길이가 4보다 크고, "₩r₩n"으로 시작되며, "₩r₩n"으로 끝나는지 체크합니다.
(펌테크 제품은 AT Command를 입력한 경우, 모든 응답 값이 4보다 크고 "₩r₩n"으로 시작해서 "₩r₩n"으로 끝납니다.)
응답 값이 4보다 크고 "₩r₩n"으로 시작해서 "₩r₩n"으로 종료되는 경우, currentStep = 12로 설정을 진행합니다.
"if-elif"문을 종료합니다.

"currentStep = 12"에 의해 "elif currentStep == 12:"부분이 실행됩니다.
"find()"를 사용하여 "uartString"변수에 저장된 문자열이 "₩r₩nREADY₩r₩n"인지 체크합니다.
비교 값이 맞는 경우, Python Shell(콘솔)로 "Received READY"로 출력합니다.
약 300ms 시간 대기합니다.
"uartString"을 초기화합니다.
"currentStep = 1"로 설정을 진행합니다.
"if-elif"문을 종료합니다.

"currentStep = 1"에 의해 "if currentStep == 1:"부분부터 다시 실행됩니다.

※ "atd"명령어 뒤의 숫자(083a5c1f015b)는 연결할 펌테크 드론에 따라 다르게 입력되어야 합니다.



① 라즈베리파이 확장보드에 조이스틱 쉴드를 장착합니다.
② 조이스틱 쉴드에 BLE 장치를 장착합니다.
③ 라즈베리파이 확장보드의 전원을 ON 합니다.
④ Python 2(IDLE) 프로그램으로 Joystick_13_Check-Message_01.py를 오픈합니다.
⑤ Python 2 프로그램을 실행합니다.(Run Module 실행 또는 F5를 누릅니다.)
⑥ Python Shell이 실행됩니다.
⑦ 펌테크 드론의 전원을 ON 합니다.
⑧ 4번 스위치 눌러서 블루투스 연결을 진행합니다. (연결되면 **"조이스틱 쉴드"**의 Status LED가 ON 됩니다.)
⑨ 출력되는 디버깅 데이터를 확인합니다.
⑩ 5번 스위치 눌러서 블루투스 연결 종료를 진행합니다.
⑪ 출력되는 디버깅 데이터를 확인합니다.
⑫ Python Shell에서 "Ctrl+C" 입력하여 프로그램을 종료합니다.
⑬ 라즈베리파이 확장보드의 전원을 OFF 합니다.
⑭ 펌테크 드론의 전원을 OFF 합니다.

23. Joystick_13_Check-Message_02

Joystic_13_Check-Message_01의 경우, 순차적으로 진행을 하기 때문에 프로그램이 비 효율적으로 운영됩니다.
예를 들면, "하드웨어 시리얼 포트로 데이터가 입력됐는지 체크"하는 부분이 너무 많이 작성되었습니다.
이 부분을 한 개만 작성하여 프로그램이 조금 더 효율적으로 운영되도록 구성하는 단계입니다.

```
import serial
import RPi.GPIO as GPIO
import time

#-------------------------------------------------
ser = serial.Serial('/dev/ttyS0',9600,timeout=0.001)

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

currentStep = 1
oldStep = 0
uartString = ""
uartLength = 0

#-------------------------------------------------
def checkNextStep():
    global currentStep
    global oldStep
    global uartString

    time.sleep(0.3)
    uartString = ""
    oldStep = currentStep
    currentStep = 0

#-------------------------------------------------
pio_list = [22, 27, 17, 23, 24, 25]

for i in range(6):
    GPIO.setup(pio_list[i], GPIO.IN)

while True:
    if currentStep == 0:
        if ser.inWaiting():
            uartString += ser.read()
            uartLength = len(uartString)
            if uartLength > 4 and uartString.find("\r\n",0,2) == 0 \
            and uartString.find("\r\n",uartLength-2) == uartLength - 2:
                currentStep = oldStep
                currentStep += 1
```

"serial"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.
"PIO"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.
"time"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.
"serial" 라이브러리를 사용하기 위해 "인자"를 생성합니다.
생성된 하드웨어 시리얼 포트는 9600bps로 동작됩니다.
"PIO" 포트의 번호 사용 방식을 "BCM"으로 설정합니다.
"PIO" 포트의 사용 중에 프로그램을 종료하는 경우 발생되는 warning을 출력되지 않도록 설정합니다.
조건문에 사용할 변수를 "currentStep"으로 정의하고 초기값을 1로 설정합니다.
이전 진행 스텝 저장 변수로 "oldStep"을 정의하고 초기값을 0으로 설정합니다.

문자열 저장할 변수를 "uartString"으로 선언하고 초기화합니다.
문자열 길이를 저장할 변수를 "uartLength"로 정의하고 초기값을 0으로 설정합니다.

"checkNextStep()"이름으로 함수를 정의합니다.
이 함수는 300ms 대기 / 문자열 저장변수 초기화 / oldStep = currentStep 실행 / currentStep = 0으로 실행합니다.
"checkNextStep()"함수는 현재 진행 중이던 스텝을 저장하고, 다음 스텝을 0으로 설정하여 하드웨어 시리얼 포트로 입력된 데이터를 체크해야 하는 경우에 사용하는 함수입니다.

"PIO" 포트로 사용할 포트를 'pio_list"라는 리스트에 설정합니다.
GPIO.setup()을 사용하여 6개의 디지털 포트를 입력으로 설정합니다.

프로그램의 지속적인 반복 수행을 위해 "while True:"를 사용합니다.

"if currentStep == 0:"부분은 하드웨어 시리얼 포트로 입력된 데이터 중 필요한 데이터를 찾는 기능을 수행합니다.
하드웨어 시리얼 포트로 입력된 데이터가 있으면 "ser.inWaiting()"이 1(참)을 반환합니다.
입력된 시리얼 데이터가 있으면 "ser.read()"로 읽어서 "uartString"에 저장합니다.
"uartString"에 저장된 데이터의 길이를 구해 "uartLength"에 저장합니다.
"uartString"에 저장된 문자열의 길이가 4보다 크고, "₩r₩n"으로 시작되며, "₩r₩n"으로 끝나는지 체크합니다.
(펌테크 제품은 AT Command를 입력한 경우, 모든 응답 값이 4보다 크고 "₩r₩n"으로 시작해서 "₩r₩n"으로 끝납니다.)
응답 값이 4보다 크고 "₩r₩n"으로 시작해서 "₩r₩n"으로 종료되는 경우, currentStep = oldStep과 currentStep +=1을 진행해서 이전에 진행했던 스텝의 다음 스텝을 진행하도록 합니다.
"if-elif"문을 종료합니다.

"currentStep = 1"에 의해 "elif currentStep == 1:"부분이 실행됩니다.

```
elif currentStep == 1:
    print("\nRaspberryPi Check Message Test 02\n")
    currentStep += 1

elif currentStep == 2:
    if GPIO.input(pio_list[4]) == 0:
        print("Pressed Connect Button")
        ser.write("atd")
        ser.write("083a5c1f015b")
        ser.write("\r")
        checkNextStep()

elif currentStep == 3:
    if uartString.find("\r\nOK\r\n",0,6) == 0:
        print("Received OK")
        checkNextStep()

elif currentStep == 4:
    if uartString.find("\r\nCONNECT ",0,10) == 0:
        print("Received CONNECT")
        time.sleep(0.3)
        uartString = ""
        currentStep += 1

elif currentStep == 5:
    if ser.inWaiting():
        ser.read()

    if GPIO.input(pio_list[5]) == 0:
        print("Pressed Disconnect Button")
        ser.write("ath")
        ser.write("\r")
        checkNextStep()

elif currentStep == 6:
    if uartString.find("\r\nOK\r\n",0,6) == 0:
        print("Received OK")
        checkNextStep()

elif currentStep == 7:
    if uartString.find("\r\nDISCONNECT",0,12) == 0:
        print("Received DISCONNECT")
        checkNextStep()

elif currentStep == 8:
    if uartString.find("\r\nREADY",0,7) == 0:
        print("Received READY")
        time.sleep(0.3)
        uartString = ""
        currentStep = 2
```

Python Shell(콘솔)로 "RaspberryPi Check Message Test 02₩n"를 출력합니다.
"currentStep = 2"로 설정을 진행합니다.
"if-elif"문을 종료합니다.

"currentStep = 2"에 의해 "elif currentStep == 2:"부분이 실행됩니다.
"if GPIO.input(pio_list[4]) == 0"을 사용하여 디지털 4번째 포트가 High 상태에서 Low 상태로 변경되었는지 체크를 합니다.
(포트의 시작은 0번째 포트부터 시작합니다.)
4번째 포트가 Low 상태로 변경된 경우, Python Shell(콘솔)로 "Pressed Connect Button"을 출력합니다.
하드웨어 시리얼 포트로 "atd083a5c1f015b₩r"을 출력합니다.
"checkNextStep()"함수를 호출합니다.
"if-elif"문을 종료합니다.

"checkNextStep()"함수는 현재 진행 중이던 스텝을 저장하고(oldStep = currentStep) 다음 스텝을 0으로 설정하여(currentStep = 0) 하드웨어 시리얼 포트로 입력된 데이터를 체크해야 하는 경우에 사용하는 함수입니다.
"checkNextStep()"에 의해 "스텝 0"이 실행되는 경우, 하드웨어 시리얼 포트로 입력된 데이터의 비교가 완료되면 "currentStep = oldStep"과 "currentStep =+ 1"에 의해 "currentStep = 3"으로 설정됩니다.

"currentStep = 3"에 의해 "elif currentStep == 3:"부분이 실행됩니다.
"find()"를 사용하여 "uartString"변수에 저장된 문자열이 "₩r₩nOK₩r₩n"인지 체크합니다.
비교 값이 맞는 경우, Python Shell(콘솔)로 "Received OK"를 출력합니다.

"checkNextStep()"함수를 호출합니다.
"if-elif"문을 종료합니다.

"checkNextStep()"함수는 현재 진행 중이던 스텝을 저장하고(oldStep = currentStep) 다음 스텝을 0으로 설정하여 (currentStep = 0) 하드웨어 시리얼 포트로 입력된 데이터를 체크해야 하는 경우에 사용하는 함수입니다.
"checkNextStep()"에 의해 "스텝 0"이 실행되는 경우, 하드웨어 시리얼 포트로 입력된 데이터의 비교가 완료되면 "currentStep = oldStep"과 "currentStep =+ 1"에 의해 "currentStep = 4"로 설정됩니다.

"currentStep = 4"에 의해 "elif currentStep == 4:"부분이 실행됩니다.
"find()"를 사용하여 "uartString"변수에 저장된 문자열이 "₩r₩nCONNECT "인지 체크합니다.
비교 값이 맞는 경우, Python Shell(콘솔)로 "Received CONNECT"를 출력합니다.
약 300ms 시간 대기합니다.
"uartString"을 초기화합니다.
"currentStep = 5"로 설정을 진행합니다.
"if-elif"문을 종료합니다.

"currentStep = 5"에 의해 "elif currentStep == 5:"부분이 실행됩니다.
하드웨어 시리얼 포트로 입력된 데이터가 있으면 "ser.inWaiting()"이 1(참)을 반환합니다.
입력된 시리얼 데이터가 있으면 "ser.read()"로 읽기만 하고 아무런 처리를 하지 않습니다.

"if GPIO.input(pio_list[5]) == 0"을 사용하여 디지털 5번째 포트가 High 상태에서 Low 상태로 변경되었는지 체크를 합니다. (포트의 시작은 0번째 포트부터 시작합니다.)
5번째 포트가 Low 상태로 변경된 경우, Python Shell(콘솔)로 "Pressed Disconnect Button"을 출력합니다.
하드웨어 시리얼 포트로 "ath₩r"를 출력합니다.
"checkNextStep()"함수를 호출합니다.
"if-elif"문을 종료합니다.

"checkNextStep()"함수는 현재 진행 중이던 스텝을 저장하고(oldStep = currentStep) 다음 스텝을 0으로 설정하여 (currentStep = 0) 하드웨어 시리얼 포트로 입력된 데이터를 체크해야 하는 경우에 사용하는 함수입니다.
"checkNextStep()"에 의해 "스텝 0"이 실행되는 경우, 하드웨어 시리얼 포트로 입력된 데이터의 비교가 완료되면 "currentStep = oldStep"과 "currentStep =+ 1"에 의해 "currentStep = 6"으로 설정됩니다.

"currentStep = 6"에 의해 "elif currentStep == 6:"부분이 실행됩니다.
"find()"를 사용하여 "uartString"변수에 저장된 문자열이 "₩r₩nOK₩r₩n"인지 체크합니다.
비교 값이 맞는 경우, Python Shell(콘솔)로 "Received OK"를 출력합니다.
"checkNextStep()"함수를 호출합니다.
"if-elif"문을 종료합니다.

"checkNextStep()"함수는 현재 진행 중이던 스텝을 저장하고(oldStep = currentStep) 다음 스텝을 0으로 설정하여 (currentStep = 0) 하드웨어 시리얼 포트로 입력된 데이터를 체크해야 하는 경우에 사용하는 함수입니다.
"checkNextStep()"에 의해 "스텝 0"이 실행되는 경우, 하드웨어 시리얼 포트로 입력된 데이터의 비교가 완료되면 "currentStep = oldStep"과 "currentStep =+ 1"에 의해 "currentStep = 7"로 설정됩니다.

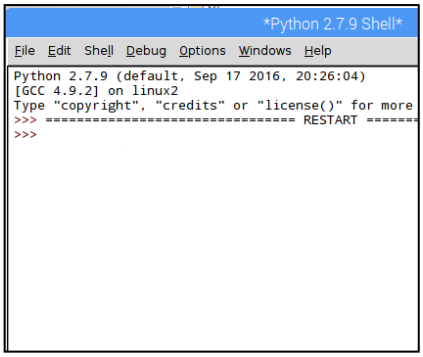
"currentStep = 7"에 의해 "elif currentStep == 7:"부분이 실행됩니다.
"find()"를 사용하여 "uartString"변수에 저장된 문자열이 "₩r₩nDISCONNECT"인지 체크합니다.
비교 값이 맞는 경우, Python Shell(콘솔)로 "Received DISCONNECT"를 출력합니다.
"checkNextStep()"함수를 호출합니다.
"if-elif"문을 종료합니다.

"checkNextStep()"함수는 현재 진행 중이던 스텝을 저장하고(oldStep = currentStep) 다음 스텝을 0으로 설정하여 (currentStep = 0) 하드웨어 시리얼 포트로 입력된 데이터를 체크해야 하는 경우에 사용하는 함수입니다.
"checkNextStep()"에 의해 "스텝 0"이 실행되는 경우, 하드웨어 시리얼 포트로 입력된 데이터의 비교가 완료되면 "currentStep = oldStep"과 "currentStep =+ 1"에 의해 "currentStep = 8"로 설정됩니다.

"currentStep = 8"에 의해 "elif currentStep == 8:"부분이 실행됩니다.
"find()"를 사용하여 "uartString"변수에 저장된 문자열이 "₩r₩nREADY"인지 체크합니다.
비교 값이 맞는 경우, Python Shell(콘솔)로 "Received READY" 출력합니다.
약 300ms 시간 대기합니다.
"uartString"를 초기화합니다.
"currentStep = 2"로 설정을 진행합니다.
"if-elif"문 종료.

"currentStep = 2"에 의해 "elif currentStep == 2:"부분부터 다시 실행됩니다.

※ "atd"명령어 뒤의 숫자(083a5c1f015b)는 연결할 펌테크 드론에 따라 다르게 입력되어야 합니다.



① 라즈베리파이 확장보드에 조이스틱 쉴드를 장착합니다.
② 조이스틱 쉴드에 BLE 장치를 장착합니다.
③ 라즈베리파이 확장보드의 전원을 ON 합니다.
④ Python 2(IDLE) 프로그램으로 Joystick_13_Check-Message_02.py를 오픈합니다.
⑤ Python 2 프로그램을 실행합니다.(Run Module 실행 또는 F5를 누릅니다.)
⑥ Python Shell이 실행됩니다.
⑦ 펌테크 드론의 전원을 ON 합니다.
⑧ 4번 스위치 눌러서 블루투스 연결을 진행합니다. (연결되면 **"조이스틱 쉴드"**의 Status LED가 ON 됩니다.)
⑨ 출력되는 디버깅 데이터를 확인합니다.
⑩ 5번 스위치 눌려서 블루투스 연결 종료를 진행합니다.
⑪ 출력되는 디버깅 데이터를 확인합니다.
⑫ Python Shell에서 "Ctrl+C" 입력하여 프로그램을 종료합니다.
⑬ 라즈베리파이 확장보드의 전원을 OFF 합니다.
⑭ 펌테크 드론의 전원을 OFF 합니다.

24. Joystick_14_DataPacket_01

펌테크 드론을 제어하기 위해 라즈베리파이 **"조이스틱 쉴드"**에서 송신해야 할 "데이터 패킷"을 테스트합니다.

BLE Peripheral 장치는 특정한 기능을 제공하기 위해서 Service를 제공합니다.
펌테크 드론의 특정한 기능은 "비행 Service"입니다.
(참고 1: 펌테크의 FBL780_Serial Peripheral의 특정한 기능은 "시리얼 Service"입니다.)
(참고 2: 펌테크의 FBL780_H Peripheral의 특정한 기능은 "시리얼 Service / PIO Service / ADC Service / Config Service" 입니다.)

BLE Peripheral 장치의 특정한 기능을 제어(데이터 송신/수신)하기 위해서는 "핸들"을 사용합니다.
(펌테크 홈페이지에서 FBL780BC_H 제품의 **"FBL780_Appendix_1~3.pdf"** 문서를 참조바랍니다.)

펌테크 드론의 "비행 Service"를 제어하기 위해서는 "0x0006"의 "핸들 번호"를 사용합니다. 즉 핸들번호 "0006"에 드론 제어를 위해 정해진 프로토콜을 송신합니다.

BLE Peripheral 장치를 제어하기 위해서는 BLE Central 장치를 사용해야 합니다.
펌테크 드론을 제어하기 위해서는 펌테크 BLE 장치인 FBL780_Serial을 Central로 설정하여 사용합니다.

FBL780_Serial을 Central로 설정하고 라즈베리파이 **"조이스틱 쉴드"**에 장착하여 펌테크 드론 제어를 위한 "데이터 패킷"을 송신합니다. (**Central 설정은 "19. Joystic_10_Serial_02"에서 진행했습니다.**)

펌테크 드론 제어를 위한 프로토콜의 "데이터 패킷" 구조는 아래와 같습니다. (**총 8바이트 사용**)

startBit	commandBit	roll	pitch	yaw	throttle	operationBit	checkSum
F0	A1	0~200	0~200	0~200	0~200	5	0

startBit / commandBit: 고정 값 (0xF0, 0xA1)
roll: 100을 기준으로 100보다 크면 오른쪽으로 이동하고, 100보다 작으면 왼쪽으로 이동합니다. (0~200)
pitch: 100을 기준으로 100보다 크면 전진하고, 100보다 작으면 후진합니다.(0~200)
yaw: 100을 기준으로 100보다 크면 우회전하고, 100보다 작으면 좌회전합니다.(0~200)
throttle: 0에서 시작해서 증가하면 상승하고, 감소하면 하강합니다.(0~200)
operationBit: 1(높이 수동 조정) / 5(높이 자동 조정)
checkSum: 송신 데이터의 오류검출에 사용합니다.
checkSum = commandBit + roll + pitch + yaw + throttle + operationBit

※ "높이 수동 조정"의 경우, 반드시 Safe Track에서 테스트를 진행하기 바랍니다.

※ "높이 수동 조정"의 경우 드론 제어가 힘들기 때문에 드론 파손의 위험이 많습니다.

```
import serial
import RPi.GPIO as GPIO
import time

#-----------------------------------------------
ser = serial.Serial('/dev/ttyS0',9600,timeout=0.001)

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

#-----------------------------------------------
switch_list = [22, 27, 17, 23, 24, 25]

for i in range(6):
    GPIO.setup(switch_list[i], GPIO.IN)

print("\nRaspberryPi Data Packet 01\n")

time.sleep(0.5)
ser.write("atd")
ser.write("083a5c1f015b")
ser.write("\r")
time.sleep(0.5)

while True:
    if GPIO.input(switch_list[0]) == 0:
        ser.write("at+writeh0006")

        ser.write("f0")
        ser.write("a1")
        ser.write("64")
        ser.write("64")
        ser.write("64")
        ser.write("78")
        ser.write("01")
        ser.write("46")

        ser.write("\r")
        time.sleep(0.3)
```

"serial"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.
"PIO"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.
"time"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.
"serial" 라이브러리를 사용하기 위해 "인자"를 생성합니다.
생성된 하드웨어 시리얼 포트는 9600bps로 동작됩니다.
"PIO" 포트의 번호 사용 방식을 "BCM"으로 설정합니다.
"PIO" 포트의 사용 중에 프로그램을 종료하는 경우 발생되는 warning을 출력되지 않도록 설정합니다.

"PIO" 포트로 사용할 포트를 'switch_list"라는 리스트에 설정합니다.
GPIO.setup()을 사용하여 6개의 디지털 포트를 입력으로 설정합니다.

출력하고자 하는 디버깅 메시지를 "print()"에 입력합니다.
여기서는 "RaspberryPi Data Packet 01"을 입력하였습니다.

BLE 장치가 시작되기를 기다리기 위해 약 500ms 대기합니다.
하드웨어 시리얼 포트로 "atd083a5c1f015b₩r"을 출력합니다.
BLE 장치가 연결되기를 기다리기 위해 약 500ms 대기합니다.

잠시 후 펌테크 드론과 연결되면 **"조이스틱 쉴드"**의 Status LED가 ON 됩니다.

프로그램의 지속적인 반복 수행을 위해 "while True:"를 사용합니다.

"if GPIO.input(switch_list[0]) == 0"을 사용하여 디지털 0번째 포트가 High 상태에서 Low 상태로 변경되었는지 체크를 합니다. (포트의 시작은 0번째 포트부터 시작합니다.)
0번째 포트가 Low 상태로 변경된 경우, 하드웨어 시리얼 포트로 "at+writeh0006"을 출력합니다. 이것은 펌테크 드론의 0006 핸들에 HEX로 데이터를 전달하겠다는 AT Command입니다.

하드웨어 시리얼 포트로 "f0"를 출력합니다. 이것은 펌테크 드론으로 startBit를 송신하는 것입니다.

하드웨어 시리얼 포트로 "a1"을 출력합니다. 이것은 펌테크 드론으로 commandBit를 송신하는 것입니다.

하드웨어 시리얼 포트로 "64"를 출력합니다. 이것은 펌테크 드론으로 roll을 송신하는 것입니다.
(0x64 = 100으로 좌/우 이동 안 함)

하드웨어 시리얼 포트로 "64"를 출력합니다. 이것은 펌테크 드론으로 pitch를 송신하는 것입니다.
(0x64 = 100으로 전/후 이동 안 함)

하드웨어 시리얼 포트로 "64"를 출력합니다. 이것은 펌테크 드론으로 yaw를 송신하는 것입니다.
(0x64 = 100으로 좌회전/우회선 안 함)

하드웨어 시리얼 포트로 "78"을 출력합니다. 이것은 펌테크 드론으로 throttle을 송신하는 것입니다.
(0x78 = 120으로 120의 세기로 상승 하라는 뜻입니다.)

하드웨어 시리얼 포트로 "01"을 출력합니다. 이것은 펌테크 드론으로 "높이 수동 조정"을 송신하는 것입니다.

하드웨어 시리얼 포트로 "46"을 출력합니다. 이것은 펌테크 드론으로 checkSum을 송신하는 것입니다.
(checkSum = 0xa1 + 0x64 + 0x64 + 0x64 + 0x78 + 0x01 : 14. Joystick_07_Checksum_02 참고)

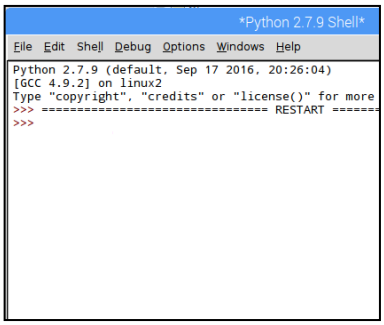
하드웨어 시리얼 포트로 "₩r"을 출력합니다. 이것은 데이터의 입력이 완료되었다는 AT Command입니다.

약 300ms 시간 동안 대기합니다.

디지털 0번째 포트가 Low 상태로 변경된 동안 **"조이스틱 쉴드"**를 통해 펌테크 드론으로 제어를 위한 프로토콜이 약 300ms 간격으로 계속 송신됩니다.

디지털 0번째 포트의 Low 상태 변경이 해제하면 펌테크 드론은 바로 하강합니다.

※ "atd"명령어 뒤의 숫자(083a5c1f015b)는 연결할 펌테크 드론에 따라 다르게 입력되어야 합니다.



① 라즈베리파이 확장보드에 조이스틱 쉴드를 장착합니다.
② 조이스틱 쉴드에 BLE 장치를 장착합니다.
③ 라즈베리파이 확장보드의 전원을 ON 합니다.
④ 펌테크 드론의 전원을 ON 합니다. (파이썬 프로그램 실행 전)
⑤ Python 2(IDLE) 프로그램으로 Joystick_14_DataPacket_01.py를 오픈합니다.
⑥ Python 2 프로그램을 실행합니다.(Run Module 실행 또는 F5를 누릅니다.)
⑦ 잠시 후 블루투스 연결이 진행됩니다. (연결되면 **"조이스틱 쉴드"**의 Status LED가 ON 됩니다.)
⑧ 0번 스위치 눌러서 펌테크 드론 제어 프로토콜을 송신합니다.
⑨ 펌테크 드론이 상승합니다. 단, 0번 스위치 지속적으로 누름 상태여야 합니다.
⑩ 0번 스위치 누름 해제 시 펌테크 드론은 하강합니다.
⑪ Python Shell에서 "Ctrl+C" 입력하여 프로그램을 종료합니다.
⑫ 라즈베리파이 확장보드의 전원을 OFF 합니다.
⑬ 펌테크 드론의 전원을 OFF 합니다.

25. Joystick_14_DataPacket_02

이전에 테스트한 결과에 의하면, 펌테크 드론을 제어하기 위해서는 "데이터 패킷"을 일정한 간격으로 지속적으로 송신해야 합니다. (이전에는 "0번" 스위치를 누르고 있는 동안 "데이터 패킷"이 지속적으로 송신됩니다.)

이번에는 펌테크 드론과 연결된 이후 일정한 간격으로 "데이터 패킷"이 지속적으로 송신되도록 합니다. 그리고 throttle (상승/하강) 제어를 함수로 만들고 "상승/하강"이 제어되도록 합니다.

throttle(상승/하강)값이 변경되면 checkSum도 변경됨으로 checkSum 계산도 함수를 이용하여 계산하도록 합니다.

```
import serial
import RPi.GPIO as GPIO
import time

#--------------------------------------------------
ser = serial.Serial('/dev/ttyS0',9600,timeout=0.001)

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

startBit = 0xf0
commandBit  = 0xa1
roll  = 100
pitch = 100
yaw = 100
throttle = 0
operationBit = 0x01
checkSum = 0

#--------------------------------------------------
def checkThrottle():
    global throttle

    if GPIO.input(switch_list[1]) == 0:
        if throttle > 59:
            throttle -= 20
        elif throttle > 3:
            throttle -= 4

    if GPIO.input(switch_list[0]) == 0:
        if throttle < 20:
            throttle = 20
        elif throttle < 181:
            throttle += 20

def checkCRC():
    global commandBit
    global roll
    global pitch
    global yaw
    global throttle
    global operationBit
    global checkSum

    checkSum = commandBit + roll + pitch + yaw + throttle + operationBit
    checkSum = checkSum & 0x00ff
```

"serial"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.
"PIO"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.
"time"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.
"serial" 라이브러리를 사용하기 위해 "인자"를 생성합니다.
생성된 하드웨어 시리얼 포트는 9600bps로 동작됩니다.
"PIO" 포트의 번호 사용 방식을 "BCM"으로 설정합니다.
"PIO" 포트의 사용 중에 프로그램을 종료하는 경우 발생되는 warning을 출력되지 않도록 설정합니다.

펌테크 드론 제어를 위한 "데이터 패킷"을 변수로 정의합니다.
startBit / commandBit / roll / pitch / yaw / throttle / operationBit / checkSum

"checkThrottle()"이름으로 함수를 정의합니다.
이 함수는 "0번"스위치와 "1번"스위치의 눌림을 체크합니다.

"1번"스위치가 눌린 경우, throttle(상승/하강) 값을 감소시킵니다.
"0번"스위치가 눌린 경우, throttle(상승/하강) 값을 증가시킵니다.

"checkCRC()"이름으로 함수를 정의합니다.
이 함수는 펌테크 드론 제어를 위한 "데이터 패킷"의 checkSum을 계산합니다. checkSum 계산은 "데이터 패킷"으로 사용하는 변수를 이용합니다.

```
switch_list = [22, 27, 17, 23, 24, 25]

for i in range(6):
    GPIO.setup(switch_list[i], GPIO.IN)

print("\nRaspberryPi Data Packet 02\n")

time.sleep(0.5)
ser.write("atd")
ser.write("083a5c1f015b")
ser.write("\r")
time.sleep(0.5)

while True:
    checkThrottle()
    checkCRC()

    ser.write("at+writeh0006")

    ser.write("f0")
    ser.write("a1")
    ser.write("64")
    ser.write("64")
    ser.write("64")

    if throttle < 0x10:
        ser.write('0'+hex(throttle)[2:4])
    else:
        ser.write(hex(throttle)[2:4])

    ser.write("01")

    if checkSum < 0x10:
        ser.write('0'+hex(checkSum)[2:4])
    else:
        ser.write(hex(checkSum)[2:4])

    ser.write("\r")
    time.sleep(0.1)
```

"PIO" 포트로 사용할 포트를 'switch_list"라는 리스트에 설정합니다.
GPIO.setup()을 사용하여 6개의 디지털 포트를 입력으로 설정합니다.

출력하고자 하는 디버깅 메시지를 "print()"에 입력합니다.
여기서는 "RaspberryPi Data Packet 02"를 입력하였습니다.

BLE 장치가 시작되기를 기다리기 위해 약 500ms 대기합니다.
하드웨어 시리얼 포트로 "atd083a5c1f015b₩r"을 출력합니다.
BLE 장치가 연결되기를 기다리기 위해 약 500ms 대기합니다.
잠시 후 펌테크 드론과 연결되면 **"조이스틱 쉴드"**의 Status LED가 ON 됩니다.

프로그램의 지속적인 반복 수행을 위해 "while True:"를 사용합니다.
"checkThrottle()"함수를 콜해서 throttle값을 계산합니다.
"checkCRC()"함수를 콜해서 checkSum값을 계산합니다.

하드웨어 시리얼 포트로 "at+writeh0006"을 출력합니다. 이것은 펌테크 드론의 0006 핸들에 HEX로 데이터를 전달하겠다는 AT Command입니다.
하드웨어 시리얼 포트로 "f0"를 출력합니다. 이것은 펌테크 드론으로 startBit를 송신하는 것입니다.
하드웨어 시리얼 포트로 "a1"을 출력합니다. 이것은 펌테크 드론으로 commandBit를 송신하는 것입니다.
하드웨어 시리얼 포트로 "64"를 출력합니다. 이것은 펌테크 드론으로 roll을 송신하는 것입니다. (0x64 = 100으로 좌/우 이동 안 함)
하드웨어 시리얼 포트로 "64"를 출력합니다. 이것은 펌테크 드론으로 pitch를 송신하는 것입니다. (0x64 = 100으로 전/후 이동 안 함)
하드웨어 시리얼 포트로 "64"를 출력합니다. 이것은 펌테크 드론으로 yaw를 송신하는 것입니다.
(0x64 = 100으로 좌회전/우회선 안 함)
하드웨어 시리얼 포트로 throttle을 출력합니다. 이것은 펌테크 드론으로 throttle을 송신하는 것입니다.
하드웨어 시리얼 포트로 "01"을 출력합니다. 이것은 펌테크 드론으로 "높이 수동 조정"을 송신하는 것입니다.
하드웨어 시리얼 포트로 checkSum을 출력합니다. 이것은 펌테크 드론으로 checkSum을 송신하는 것입니다.
(checkSum = commandBit + roll + pitch + yaw + throttle + operationBit)
하드웨어 시리얼 포트로 "₩r"을 출력합니다. 이것은 데이터의 입력이 완료되었다는 AT Command입니다.
약 100ms 시간 동안 대기합니다.

※ write(hex(throttle)) / write(str(throttle))의 차이점

write(hex(throttle)) => 실제 값(숫자)을 HEX 타입의 문자로 변경하여 출력합니다.
write(str(throttle)) => 실제 값을 10진수 타입의 문자로 변경하여 출력합니다.
HEX와 10진수는 10진수 10의 값부터 표시 방법이 달라집니다.
"joystick_14_DataPacket_02-1.py"를 실행하여 차이점 확인.

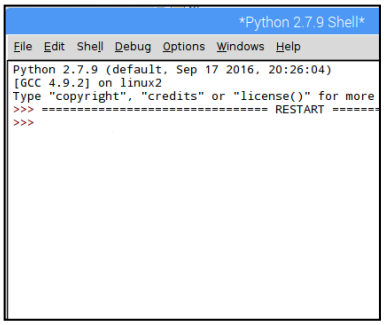
※ write(hex(throttle)) / write(hex(throttle) [2:4])의 차이점

write(hex(throttle)) => 실제 값(숫자)을 HEX 타입의 문자로 변경하여 출력합니다.
HEX 타입을 나타내기 위해 "0x"가 문자로 추가되어 출력합니다.
write(hex(throttle)[2:4]) => 실제 값(숫자)을 HEX 타입의 문자로 변경하여 출력합니다.
HEX 타입을 나타내는 "0x" 이후 문자부터 출력합니다.
"joystick_14_DataPacket_02-2.py"를 실행하여 차이점 확인.

※ write(hex(throttle)[2:4]) / write('0'+hex(throttle) [2:4])의 차이점

16진수(HEX) 0 ~ f 까지 값은 한 자리수로 인식됩니다. 이것을 문자로 변경해도 한 자리로 인식됩니다.
16진수(HEX)를 두 자리수로 처리하기 위해 0 ~f 까지 값인 경우 write('0'+hex(throttle)[2:4])를 사용하여 인위적으로 문자 "0"을 추가해 줍니다.
"joystick_14_DataPacket_02-3.py"를 실행하여 차이점 확인.

※ "atd"명령어 뒤의 숫자(083a5c1f015b)는 연결할 펌테크 드론에 따라 다르게 입력되어야 합니다.



① 라즈베리파이 확장보드에 조이스틱 쉴드를 장착합니다.
② 조이스틱 쉴드에 BLE 장치를 장착합니다.
③ 라즈베리파이 확장보드의 전원을 ON 합니다.
④ 펌테크 드론의 전원을 ON 합니다. (파이썬 프로그램 실행 전)
⑤ Python 2(IDLE) 프로그램으로 Joystick_14_DataPacket_02.py를 오픈합니다.
⑥ Python 2 프로그램을 실행합니다.(Run Module 실행 또는 F5를 누릅니다.)
⑦ 잠시 후 블루투스 연결이 진행됩니다. (연결되면 **"조이스틱 쉴드"**의 Status LED가 ON 됩니다.)
⑧ 0번 스위치 눌러서 throttle 값을 증가시킵니다.
⑨ 펌테크 드론이 상승합니다.
⑩ 1번 스위치 눌러서 throttle 값을 감소시킵니다.
⑪ 펌테크 드론이 하강합니다.
⑫ Python Shell에서 "Ctrl+C" 입력하여 프로그램을 종료합니다.
⑬ 라즈베리파이 확장보드의 전원을 OFF 합니다.
⑭ 펌테크 드론의 전원을 OFF 합니다.

26. Joystick_14_DataPacket_03

ser.write("f0")나 ser.write("64")의 경우, 변경이 불가능한 문자를 출력하는 것입니다.

조금 더 유용한 프로그램 작성을 위해 변경이 불가능한 문자를 출력하지 말고 변경이 가능한 변수에 저장된 숫자를 문자로 변경하여 출력하는 방식을 사용합니다.

```
import serial
import RPi.GPIO as GPIO
import time

#--------------------------------------------------
ser = serial.Serial('/dev/ttyS0',9600,timeout=0.001)

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

startBit = 0xf0
commandBit  = 0xa1
roll  = 100
pitch = 100
yaw = 100
throttle = 0
operationBit = 0x01
checkSum = 0

#--------------------------------------------------
def checkThrottle():
    global throttle

    if GPIO.input(switch_list[1]) == 0:
        if throttle > 59:
            throttle -= 20
        elif throttle > 3:
            throttle -= 4

    if GPIO.input(switch_list[0]) == 0:
        if throttle < 20:
           throttle = 20
        elif throttle < 181:
            throttle += 20

def checkCRC():
    global commandBit
    global roll
    global pitch
    global yaw
    global throttle
    global operationBit
    global checkSum

    checkSum = commandBit + roll + pitch + yaw + throttle + operationBit
    checkSum = checkSum & 0x00ff
```

"serial"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.

"PIO"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.

"time"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.

"serial" 라이브러리를 사용하기 위해 "인자"를 생성합니다.

생성된 하드웨어 시리얼 포트는 9600bps로 동작됩니다.

"PIO" 포트의 번호 사용 방식을 "BCM"으로 설정합니다.

"PIO" 포트의 사용 중에 프로그램을 종료하는 경우 발생되는 warning을 출력되지 않도록 설정합니다.

펌테크 드론 제어를 위한 "데이터 패킷"을 변수로 정의합니다.
startBit / commandBit / roll / pitch / yaw / throttle / operationBit / checkSum

"checkThrottle()"이름으로 함수를 정의합니다.
이 함수는 "0번"스위치와 "1번"스위치의 눌림을 체크합니다.
"1번"스위치가 눌린 경우, throttle(상승/하강) 값을 감소시킵니다.
"0번"스위치가 눌린 경우, throttle(상승/하강) 값을 증가시킵니다.

"checkCRC()"이름으로 함수를 정의합니다.
이 함수는 펌테크 드론 제어를 위한 "데이터 패킷"의 checkSum을 계산합니다. checkSum 계산은 "데이터 패킷"으로 사용하는 변수를 이용합니다.

```
switch_list = [22, 27, 17, 23, 24, 25]

for i in range(6):
    GPIO.setup(switch_list[i], GPIO.IN)

print("\nRaspberryPi Data Packet 03\n")

time.sleep(0.5)
ser.write("atd")
ser.write("083a5c1f015b")
ser.write("\r")
time.sleep(0.5)

while True:
    checkThrottle()
    checkCRC()

    ser.write("at+writeh0006")

    ser.write(hex(startBit)[2:4])
    ser.write(hex(commandBit)[2:4])
    ser.write(hex(roll)[2:4])
    ser.write(hex(pitch)[2:4])
    ser.write(hex(yaw)[2:4])

    if throttle < 0x10:
        ser.write('0'+hex(throttle)[2:4])
    else:
        ser.write(hex(throttle)[2:4])

    ser.write("01")

    if checkSum < 0x10:
        ser.write('0'+hex(checkSum)[2:4])
    else:
        ser.write(hex(checkSum)[2:4])

    ser.write("\r")
    time.sleep(0.1)
```

"PIO" 포트로 사용할 포트를 'switch_list"라는 리스트에 설정합니다.
GPIO.setup()을 사용하여 6개의 디지털 포트를 입력으로 설정합니다.

출력하고자 하는 디버깅 메시지를 "print()"에 입력합니다.
여기서는 "RaspberryPi Data Packet 03"을 입력하였습니다.

BLE 장치가 시작되기를 기다리기 위해 약 500ms 대기합니다.
하드웨어 시리얼 포트로 "atd083a5c1f015b₩r"을 출력합니다.
BLE 장치가 연결되기를 기다리기 위해 약 500ms 대기합니다.
잠시 후 펌테크 드론과 연결되면 **"조이스틱 쉴드"**의 Status LED가 ON 됩니다.

프로그램의 지속적인 반복 수행을 위해 "while True:"를 사용합니다.

"checkThrottle()"함수를 콜해서 throttle값을 계산합니다.
"checkCRC()"함수를 콜해서 checkSum값을 계산합니다.

하드웨어 시리얼 포트로 "at+writeh0006"을 출력합니다. 이것은 펌테크 드론의 0006 핸들에 HEX로 데이터를 전달하겠다는 AT Command입니다.

하드웨어 시리얼 포트로 startBit를 출력합니다. 이것은 펌테크 드론으로 startBit를 송신하는 것입니다.

하드웨어 시리얼 포트로 commandBit를 출력합니다. 이것은 펌테크 드론으로 commandBit를 송신하는 것입니다.

하드웨어 시리얼 포트로 roll을 출력합니다. 이것은 펌테크 드론으로 roll을 송신하는 것입니다.

하드웨어 시리얼 포트로 pitch를 출력합니다. 이것은 펌테크 드론으로 pitch를 송신하는 것입니다.

하드웨어 시리얼 포트로 yaw를 출력합니다. 이것은 펌테크 드론으로 yaw를 송신하는 것입니다.

하드웨어 시리얼 포트로 throttle을 출력합니다. 이것은 펌테크 드론으로 throttle을 송신하는 것입니다.

하드웨어 시리얼 포트로 operationBit를 출력입니다. 이것은 펌테크 드론으로 "높이 수동 조정"을 송신하는 것입니다.

하드웨어 시리얼 포트로 checkSum을 출력합니다. 이것은 펌테크 드론으로 checkSum을 송신하는 것입니다.
(checkSum = commandBit + roll + pitch + yaw + throttle + operationBit)

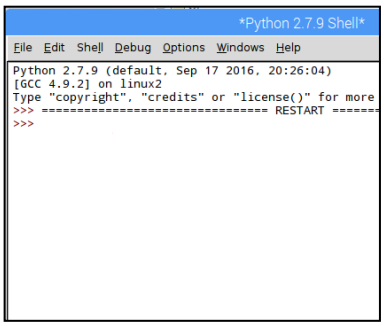
하드웨어 시리얼 포트로 "₩r"을 출력합니다. 이것은 데이터의 입력이 완료되었다는 AT Command입니다.

약 100ms 시간 동안 대기합니다.

※ roll / pitch / yaw의 경우, 아직은 변하지 않고 100(0x64)을 출력하기 때문에 문자 "0"을 출력하는 부분이 추가되지 않았습니다.

※ operationBit의 경우, 1(0x01)을 출력하기 때문에 문자 "0"을 조건 없이 출력하도록 합니다.

※ "atd"명령어 뒤의 숫자(083a5c1f015b)는 연결할 펌테크 드론에 따라 다르게 입력되어야 합니다.



① 라즈베리파이 확장보드에 조이스틱 쉴드를 장착합니다.
② 조이스틱 쉴드에 BLE 장치를 장착합니다.
③ 라즈베리파이 확장보드의 전원을 ON 합니다.
④ 펌테크 드론의 전원을 ON 합니다. (파이썬 프로그램 실행 전)
⑤ Python 2(IDLE) 프로그램으로 Joystick_14_DataPacket_03.py를 오픈합니다.
⑥ Python 2 프로그램을 실행합니다.(Run Module 실행 또는 F5를 누릅니다.)
⑦ 잠시 후 블루투스 연결이 진행됩니다. (연결되면 **"조이스틱 쉴드"**의 Status LED가 ON됩니다.)
⑧ 0번 스위치 눌러서 throttle 값을 증가시킵니다.
⑨ 펌테크 드론이 상승합니다.
⑩ 1번 스위치 눌러서 throttle 값을 감소시킵니다.
⑪ 펌테크 드론이 하강합니다.
⑫ Python Shell에서 "Ctrl+C" 입력하여 프로그램을 종료합니다.
⑬ 라즈베리파이 확장보드의 전원을 OFF 합니다.
⑭ 펌테크 드론의 전원을 OFF 합니다.

27. Joystick_14_DataPacket_04

이번에는 펌테크 드론 pitch(전진/후진) 제어를 함수로 만들고 "전진/후진"이 제어되도록 합니다.

```
#-------------------------
import serial
import RPi.GPIO as GPIO
import time
import spidev

#-------------------------
ser = serial.Serial('/dev/ttyS0',9600,timeout=0.001)
spi = spidev.SpiDev()
spi.open(0,0)
spi.max_speed_hz = 1000000

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

startBit = 0xf0
commandBit  = 0xa1
roll  = 100
pitch = 100
yaw = 100
throttle = 0
operationBit = 0x01
checkSum = 0

#-------------------------
def analog_read(channel):
    data = spi.xfer2([1, (0x08 + channel) << 4, 0])
    adc_out = ((data[1] & 0x03) << 8) + data[2]
    time.sleep(0.01)

    return adc_out

def checkThrottle():
    global throttle

    if GPIO.input(switch_list[1]) == 0:
        if throttle > 59:
            throttle -= 20
        elif throttle > 3:
            throttle -= 4

    if GPIO.input(switch_list[0]) == 0:
        if throttle < 20:
            throttle = 20
        elif throttle < 181:
            throttle += 20

def checkPitch():
    global pitch

    secondPitch = analog_read(3)

    if secondPitch < 400:
        pitch = 50
    elif secondPitch > 600:
        pitch = 150
    else:
        pitch = 100
```

"serial"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.
"PIO"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.
"time"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.
"SPI"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.
"serial" 라이브러리를 사용하기 위해 "인자"를 생성합니다.
생성된 하드웨어 시리얼 포트는 9600bps로 동작됩니다.
"SPI" 라이브러리를 사용하기 위해 "인자"를 생성합니다.
생성된 "인자"를 사용하여 라이브러리 초기화를 진행합니다.
최대 클럭 스피드를 1MHz로 설정합니다.
"PIO" 포트의 번호 사용 방식을 "BCM"으로 설정합니다.
"PIO" 포트의 사용 중에 프로그램을 종료하는 경우 발생되는 warning을 출력되지 않도록 설정합니다.
펌테크 드론 제어를 위한 "데이터 패킷"을 변수로 정의합니다.
startBit / commandBit / roll / pitch / yaw / throttle / operationBit / checkSum
"analog_read()" 이름으로 함수를 정의합니다.
이 함수는 ADC 변환 칩에 입력된 ADC 데이터를 SPI 포트로 읽어옵니다.
주어진 채널에 해당하는 ADC 값을 읽어 10비트 디지털 값으로 리턴합니다.
"checkThrottle()"이름으로 함수를 정의합니다.
이 함수는 "0번"스위치와 "1번"스위치의 눌림을 체크합니다.
"1번"스위치가 눌린 경우, throttle(상승/하강) 값을 감소시킵니다.
"0번"스위치가 눌린 경우, throttle(상승/하강) 값을 증가시킵니다.
"checkPitch()" 이름으로 함수를 정의합니다.
이 함수는 "analog_read()" 함수를 이용하여 ADC 3번 채널 변환 값을 읽습니다
"ADC 3번"채널 값이 400보다 작은 경우, pitch(전진/후진) 값을 50으로 설정합니다.
"ADC 3번"채널 값이 600보다 큰 경우, pitch(전진/후진) 값을 150으로 설정합니다.
"ADC 3번"채널 값이 400 ~ 600 사이인 경우, pitch(전진/후진) 값을 100으로 설정합니다.

```
def checkCRC():
    global commandBit
    global roll
    global pitch
    global yaw
    global throttle
    global operationBit
    global checkSum

    checkSum = commandBit + roll + pitch + yaw + throttle + operationBit
    checkSum = checkSum & 0x00ff
```

"checkCRC()"이름으로 함수를 정의합니다.
이 함수는 펌테크 드론 제어를 위한 "데이터 패킷"의 checkSum을 계산합니다. checkSum 계산은 "데이터 패킷"으로 사용하는 변수를 이용합니다.

```
switch_list = [22, 27, 17, 23, 24, 25]

for i in range(6):
    GPIO.setup(switch_list[i], GPIO.IN)

print("\nRaspberryPi Data Packet 04\n")

time.sleep(0.5)
ser.write("atd")
ser.write("083a5c1f015b")
ser.write("\r")
time.sleep(0.5)

while True:
    checkThrottle()
    checkPitch()
    checkCRC()

    ser.write("at+writeh0006")

    ser.write(hex(startBit)[2:4])
    ser.write(hex(commandBit)[2:4])
    ser.write(hex(roll)[2:4])
    ser.write(hex(pitch)[2:4])
    ser.write(hex(yaw)[2:4])

    if throttle < 0x10:
        ser.write('0'+hex(throttle)[2:4])
    else:
        ser.write(hex(throttle)[2:4])

    ser.write("01")

    if checkSum < 0x10:
        ser.write('0'+hex(checkSum)[2:4])
    else:
        ser.write(hex(checkSum)[2:4])

    ser.write("\r")
    time.sleep(0.1)
```

"PIO" 포트로 사용할 포트를 'switch_list"라는 리스트에 설정합니다.
GPIO.setup()을 사용하여 6개의 디지털 포트를 입력으로 설정합니다.
출력하고자 하는 디버깅 메시지를 "print()"에 입력합니다.
여기서는 "RaspberryPi Data Packet 04"를 입력하였습니다.
BLE 장치가 시작되기를 기다리기 위해 약 500ms 대기합니다.
하드웨어 시리얼 포트로 "atd083a5c1f015b₩r"을 출력합니다.
BLE 장치가 연결되기를 기다리기 위해 약 500ms 대기합니다.
잠시 후 펌테크 드론과 연결되면 **"조이스틱 쉴드"**의 Status LED가 ON 됩니다.

프로그램의 지속적인 반복 수행을 위해 "while True:"를 사용합니다.

"checkThrottle()"함수를 콜해서 throttle값을 계산합니다.
"checkPitch()"함수를 콜해서 pitch값을 계산합니다.
"checkCRC()"함수를 콜해서 checkSum값을 계산합니다.

하드웨어 시리얼 포트로 "at+writeh0006"을 출력합니다. 이것은 펌테크 드론의 0006 핸들에 HEX로 데이터를 전달하겠다는 AT Command입니다.
하드웨어 시리얼 포트로 startBit를 출력합니다. 이것은 펌테크 드론으로 startBit를 송신하는 것입니다.
하드웨어 시리얼 포트로 commandBit를 출력합니다. 이것은 펌테크 드론으로 commandBit를 송신하는 것입니다.
하드웨어 시리얼 포트로 roll을 출력합니다. 이것은 펌테크 드론으로 roll을 송신하는 것입니다.
하드웨어 시리얼 포트로 pitch를 출력합니다. 이것은 펌테크 드론으로 pitch를 송신하는 것입니다.
하드웨어 시리얼 포트로 yaw를 출력합니다. 이것은 펌테크 드론으로 yaw를 송신하는 것입니다.

하드웨어 시리얼 포트로 throttle을 출력합니다. 이것은 펌테크 드론으로 throttle을 송신하는 것입니다.
하드웨어 시리얼 포트로 operationBit를 출력합니다. 이것은 펌테크 드론으로 "높이 수동 조정"을 송신하는 것입니다.

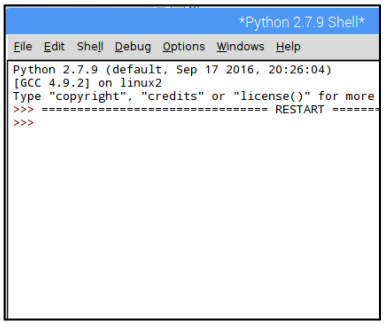
하드웨어 시리얼 포트로 checkSum을 출력합니다. 이것은 펌테크 드론으로 checkSum을 송신하는 것입니다.
(checkSum = commandBit + roll + pitch + yaw + throttle + operationBit)
하드웨어 시리얼 포트로 "₩r"을 출력합니다. 이것은 데이터의 입력이 완료되었다는 AT Command입니다.
약 100ms 시간 동안 대기합니다.

※ pitch의 경우, 50(0x32) / 100(0x64) / 150(0x96)값만 가지도록 했기 때문에 문자 "0"을 출력하는 부분이 추가되지 않았습니다.

※ roll / yaw의 경우, 아직은 변하지 않고 100(0x64)를 출력하기 때문에 문자 "0"을 출력하는 부분이 추가되지 않았습니다.

※ operationBit의 경우, 1(0x01)을 출력하기 때문에 문자 "0"을 조건 없이 출력하도록 합니다.

※ "atd"명령어 뒤의 숫자(083a5c1f015b)는 연결할 펌테크 드론에 따라 다르게 입력되어야 합니다.



① 라즈베리파이 확장보드에 조이스틱 쉴드를 장착합니다.
② 조이스틱 쉴드에 BLE 장치를 장착합니다.
③ 라즈베리파이 확장보드의 전원을 ON 합니다.
④ 펌테크 드론의 전원을 ON 합니다. (파이썬 프로그램 실행 전)
⑤ Python 2(IDLE) 프로그램으로 Joystick_14_DataPacket_04.py를 오픈합니다.
⑥ Python 2 프로그램을 실행합니다.(Run Module 실행 또는 F5를 누릅니다.)
⑦ 잠시 후 블루투스 연결이 진행됩니다. (연결되면 **"조이스틱 쉴드"**의 Status LED가 ON 됩니다.)
⑧ 0번 스위치 눌러서 throttle 값을 증가시킵니다.
⑨ 펌테크 드론이 상승합니다.
⑩ "ADC 3번"채널 값 읽어서(조이스틱 앞으로 조정) pitch 값을 변경시킵니다.
⑪ 펌테크 드론이 전진합니다. (앞으로 기울어짐)
⑫ "ADC 3번"채널 값 읽어서(조이스틱 뒤로 조정) pitch 값을 변경시킵니다.
⑬ 펌테크 드론이 후진합니다. (뒤로 기울어짐)
⑭ "ADC 3번"채널 값 읽어서(조이스틱 중앙으로 조정) pitch 값을 변경시킵니다.
⑮ 펌테크 드론이 수평이 됩니다. (기울어짐 없음)
⑯ 1번 스위치 눌러서 throttle 값을 감소시킵니다.
⑰ 펌테크 드론이 하강합니다.
⑱ Python Shell에서 "Ctrl+C" 입력하여 프로그램을 종료합니다.
⑲ 라즈베리파이 확장보드의 전원을 OFF 합니다.
⑳ 펌테크 드론의 전원을 OFF 합니다.

28. Joystick_14_DataPacket_05

이번에는 펌테크 드론 roll(좌이동/우이동) 제어를 함수로 만들고 "좌이동/우이동"이 제어되도록 합니다.

```
#-------------------------
import serial
import RPi.GPIO as GPIO
import time
import spidev

#-------------------------
ser = serial.Serial('/dev/ttyS0',9600,timeout=0.001)
spi = spidev.SpiDev()
spi.open(0,0)
spi.max_speed_hz = 1000000

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

startBit = 0xf0
commandBit  = 0xa1
roll  = 100
pitch = 100
yaw = 100
throttle = 0
operationBit = 0x01
checkSum = 0

#-------------------------
def analog_read(channel):
    data = spi.xfer2([1, (0x08 + channel) << 4, 0])
    adc_out = ((data[1] & 0x03) << 8) + data[2]
    time.sleep(0.01)

    return adc_out

def checkThrottle():
    global throttle

    if GPIO.input(switch_list[1]) == 0:
        if throttle > 59:
            throttle -= 20
        elif throttle > 3:
            throttle -= 4

    if GPIO.input(switch_list[0]) == 0:
        if throttle < 20:
            throttle = 20
        elif throttle < 181:
            throttle += 20
```

"serial"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.

"PIO"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.

"time"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.

"SPI"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.

"serial" 라이브러리를 사용하기 위해 "인자"를 생성합니다.
생성된 하드웨어 시리얼 포트는 9600bps로 동작됩니다.

"SPI" 라이브러리를 사용하기 위해 "인자"를 생성합니다.
생성된 "인자"를 사용하여 라이브러리 초기화를 진행합니다.
최대 클럭 스피드를 1MHz로 설정합니다.

"PIO" 포트의 번호 사용 방식을 "BCM"으로 설정합니다.
"PIO" 포트의 사용 중에 프로그램을 종료하는 경우 발생되는 warning을 출력되지 않도록 설정합니다.

펌테크 드론 제어를 위한 "데이터 패킷"을 변수로 정의합니다.
startBit / commandBit / roll / pitch / yaw / throttle / operationBit / checkSum

"analog_read()" 이름으로 함수를 정의합니다.
이 함수는 ADC 변환 칩에 입력된 ADC 데이터를 SPI 포트로 읽어옵니다.
주어진 채널에 해당하는 ADC 값을 읽어 10비트 디지털 값으로 리턴합니다.

"checkThrottle()"이름으로 함수를 정의합니다.
이 함수는 "0번"스위치와 "1번"스위치의 눌림을 체크합니다.
"1번"스위치가 눌린 경우, throttle(상승/하강) 값을 감소시킵니다.
"0번"스위치가 눌린 경우, throttle(상승/하강) 값을 증가시킵니다.

```
def checkPitch():
    global pitch

    secondPitch = analog_read(3)

    if secondPitch < 400:
        pitch = 50
    elif secondPitch > 600:
        pitch = 150
    else:
        pitch = 100

def checkRoll():
    global roll

    secondRoll = analog_read(2)

    if secondRoll < 400:
        roll = 50
    elif secondRoll > 600:
        roll = 150
    else:
        roll = 100
```

"checkPitch()" 이름으로 함수를 정의합니다.
이 함수는 "analog_read()" 함수를 이용하여 ADC 3번 채널 변환 값을 읽습니다.
"ADC 3번"채널 값이 400보다 작은 경우, pitch(전진/후진) 값을 50으로 설정합니다.
"ADC 3번"채널 값이 600보다 큰 경우, pitch(전진/후진) 값을 150으로 설정합니다.
"ADC 3번"채널 값이 400 ~ 600 사이인 경우, pitch(전진/후진) 값을 100으로 설정합니다.

"checkRoll()" 이름으로 함수를 정의합니다.
이 함수는 "analog_read()" 함수를 이용하여 ADC 2번 채널 변환 값을 읽습니다.
"ADC 2번"채널 값이 400보다 작은 경우, roll(좌이동/우이동) 값을 50으로 설정합니다.
"ADC 2번"채널 값이 600보다 큰 경우, roll(좌이동/우이동) 값을 150으로 설정합니다.
"ADC 2번"채널 값이 400 ~ 600 사이인 경우, roll(좌이동/우이동) 값을 100으로 설정합니다.

```
def checkCRC():
    global commandBit
    global roll
    global pitch
    global yaw
    global throttle
    global operationBit
    global checkSum

    checkSum = commandBit + roll + pitch + yaw + throttle + operationBit
    checkSum = checkSum & 0x00ff
```

"checkCRC()"이름으로 함수를 정의합니다.
이 함수는 펌테크 드론 제어를 위한 "데이터 패킷"의 checkSum을 계산합니다. checkSum 계산은 "데이터 패킷"으로 사용하는 변수를 이용합니다.

```
switch_list = [22, 27, 17, 23, 24, 25]

for i in range(6):
    GPIO.setup(switch_list[i], GPIO.IN)

print("\nRaspberryPi Data Packet 05\n")

time.sleep(0.5)
ser.write("atd")
ser.write("083a5c1f015b")
ser.write("\r")
time.sleep(0.5)

while True:
    checkThrottle()
    checkPitch()
    checkRoll()
    checkCRC()

    ser.write("at+writeh0006")

    ser.write(hex(startBit)[2:4])
    ser.write(hex(commandBit)[2:4])
    ser.write(hex(roll)[2:4])
    ser.write(hex(pitch)[2:4])
    ser.write(hex(yaw)[2:4])

    if throttle < 0x10:
        ser.write('0'+hex(throttle)[2:4])
    else:
        ser.write(hex(throttle)[2:4])

    ser.write("01")

    if checkSum < 0x10:
        ser.write('0'+hex(checkSum)[2:4])
    else:
        ser.write(hex(checkSum)[2:4])

    ser.write("\r")
    time.sleep(0.1)
```

"PIO" 포트로 사용할 포트를 'switch_list"라는 리스트에 설정합니다.
GPIO.setup()을 사용하여 6개의 디지털 포트를 입력으로 설정합니다.
출력하고자 하는 디버깅 메시지를 "print()"에 입력합니다.
여기서는 "RaspberryPi Data Packet 05"를 입력하였습니다.
BLE 장치가 시작되기를 기다리기 위해 약 500ms 대기합니다.
하드웨어 시리얼 포트로 "atd083a5c1f015b₩r"을 출력합니다.
BLE 장치가 연결되기를 기다리기 위해 약 500ms 대기합니다.
잠시 후 펌테크 드론과 연결되면 **"조이스틱 쉴드"**의 Status LED가 ON 됩니다.

프로그램의 지속적인 반복 수행을 위해 "while True:"를 사용합니다.

"checkThrottle()"함수를 콜해서 throttle값을 계산합니다.
"checkPitch()"함수를 콜해서 pitch값을 계산합니다.
"checkRoll()"함수를 콜해서 roll값을 계산합니다.
"checkCRC()"함수를 콜해서 checkSum값을 계산합니다.

하드웨어 시리얼 포트로 "at+writeh0006"을 출력합니다. 이것은 펌테크 드론의 0006 핸들에 HEX로 데이터를 전달하겠다는 AT Command입니다.
하드웨어 시리얼 포트로 startBit를 출력합니다. 이것은 펌테크 드론으로 startBit를 송신하는 것입니다.
하드웨어 시리얼 포트로 commandBit를 출력합니다. 이것은 펌테크 드론으로 commandBit를 송신하는 것입니다.
하드웨어 시리얼 포트로 roll을 출력합니다. 이것은 펌테크 드론으로 roll을 송신하는 것입니다.
하드웨어 시리얼 포트로 pitch를 출력합니다. 이것은 펌테크 드론으로 pitch를 송신하는 것입니다

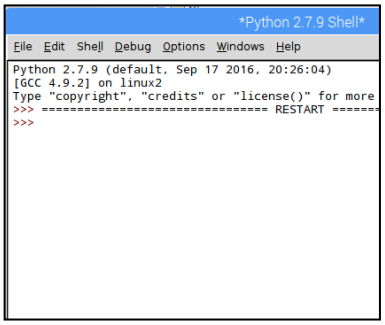
.
하드웨어 시리얼 포트로 yaw를 출력합니다. 이것은 펌테크 드론으로 yaw를 송신하는 것입니다.
하드웨어 시리얼 포트로 throttle을 출력합니다. 이것은 펌테크 드론으로 throttle을 송신하는 것입니다.
하드웨어 시리얼 포트로 operationBit를 출력합니다. 이것은 펌테크 드론으로 "높이 수동 조정"을 송신하는 것입니다.
하드웨어 시리얼 포트로 checkSum을 출력합니다. 이것은 펌테크 드론으로 checkSum을 송신하는 것입니다.
(checkSum = commandBit + roll + pitch + yaw + throttle + operationBit)
하드웨어 시리얼 포트로 "₩r"을 출력합니다. 이것은 데이터의 입력이 완료되었다는 AT Command입니다.
약 100ms 시간 동안 대기합니다.

※ pitch / roll의 경우, 50(0x32) / 100(0x64) / 150(0x96)값만 가지도록 했기 때문에 문자 "0"을 출력하는 부분이 추가되지 않았습니다.

※ yaw의 경우, 아직은 변하지 않고 100(0x64)를 출력하기 때문에 문자 "0"을 출력하는 부분이 추가되지 않았습니다.

※ operationBit의 경우, 1(0x01)을 출력하기 때문에 문자 "0"을 조건 없이 출력하도록 합니다.

※ "atd"명령어 뒤의 숫자(083a5c1f015b)는 연결할 펌테크 드론에 따라 다르게 입력되어야 합니다.



① 라즈베리파이 확장보드에 조이스틱 쉴드를 장착합니다.
② 조이스틱 쉴드에 BLE 장치를 장착합니다.
③ 라즈베리파이 확장보드의 전원을 ON 합니다.
④ 펌테크 드론의 전원을 ON 합니다. (파이썬 프로그램 실행 전)
⑤ Python 2(IDLE) 프로그램으로 Joystick_14_DataPacket_05.py를 오픈합니다.
⑥ Python 2 프로그램을 실행합니다.(Run Module 실행 또는 F5를 누릅니다.)
⑦ 잠시 후 블루투스 연결이 진행됩니다. (연결되면 **"조이스틱 쉴드"**의 Status LED가 ON 됩니다.)
⑧ 0번 스위치 눌러서 throttle 값을 증가시킵니다.
⑨ 펌테크 드론이 상승합니다.
⑩ "ADC 2번"채널 값 읽어서(조이스틱 오른쪽으로 조정) roll 값을 변경시킵니다..
⑪ 펌테크 드론 우측으로 이동합니다. (우측으로 기울어짐)
⑫ "ADC 2번"채널 값 읽어서(조이스틱 왼쪽으로 조정) roll 값을 변경시킵니다.
⑬ 펌테크 드론 좌측으로 이동합니다. (좌측으로 기울어짐)
⑭ "ADC 2번"채널 값 읽어서(조이스틱 중앙으로 조정) roll 값을 변경시킵니다.
⑮ 펌테크 드론 수평을 유지합니다. (기울어짐 없음)
⑯ 1번 스위치 눌러서 throttle 값을 감소시킵니다.
⑰ 펌테크 드론이 하강합니다.
⑱ Python Shell에서 "Ctrl+C" 입력하여 프로그램을 종료합니다.
⑲ 라즈베리파이 확장보드의 전원을 OFF 합니다.
⑳ 펌테크 드론의 전원을 OFF 합니다.

29. Joystick_14_DataPacket_06

이번에는 펌테크 드론 yaw(좌회전/우회전) 제어를 함수로 만들고 "좌회전/우회전"이 제어되도록 합니다.

※ yaw(좌회선/우회전)는 "Safety Track"에서는 테스트가 불가능합니다.

비상버튼을 체크해서 모터 회전을 0으로 설정하도록 합니다.

"데이터 패킷" 송신 부분을 함수로 구성합니다.

```
#-------------------------
import serial
import RPi.GPIO as GPIO
import time
import spidev

#-------------------------
ser = serial.Serial('/dev/ttyS0',9600,timeout=0.001)
spi = spidev.SpiDev()
spi.open(0,0)
spi.max_speed_hz = 1000000

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

startBit = 0xf0
commandBit  = 0xa1
roll  = 100
pitch = 100
yaw = 100
throttle = 0
operationBit = 0x01
checkSum = 0

#-------------------------

def analog_read(channel):
    data = spi.xfer2([1, (0x08 + channel) << 4, 0])
    adc_out = ((data[1] & 0x03) << 8) + data[2]
    time.sleep(0.01)

    return adc_out

def checkThrottle():
    global throttle

    if GPIO.input(switch_list[1]) == 0:
        if throttle > 59:
            throttle -= 20
        elif throttle > 3:
            throttle -= 4

    if GPIO.input(switch_list[0]) == 0:
        if throttle < 20:
            throttle = 20
        elif throttle < 181:
            throttle += 20
def checkPitch():
    global pitch

    secondPitch = analog_read(3)

    if secondPitch < 400:
        pitch = 50
    elif secondPitch > 600:
        pitch = 150
    else:
        pitch = 100
```

"serial"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.
"PIO"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.
"time"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.
"SPI"에 관련된 외부 함수를 사용하기 위해 라이브러리를 추가합니다.

"serial" 라이브러리를 사용하기 위해 "인자"를 생성합니다.
생성된 하드웨어 시리얼 포트는 9600bps로 동작됩니다.

"SPI" 라이브러리를 사용하기 위해 "인자"를 생성합니다.
생성된 "인자"를 사용하여 라이브러리 초기화를 진행합니다.
최대 클럭 스피드를 1MHz로 설정합니다.

"PIO" 포트의 번호 사용 방식을 "BCM"으로 설정합니다.
"PIO" 포트의 사용 중에 프로그램을 종료하는 경우 발생되는 warning을 출력되지 않도록 설정합니다.
펌테크 드론 제어를 위한 "데이터 패킷"을 변수로 정의합니다.
startBit / commandBit / roll / pitch / yaw / throttle / operationBit / checkSum

"analog_read()" 이름으로 함수를 정의합니다.
이 함수는 ADC 변환 칩에 입력된 ADC 데이터를 SPI 포트로 읽어옵니다.
주어진 채널에 해당하는 ADC 값을 읽어 10비트 디지털 값으로 리턴합니다.

"checkThrottle()"이름으로 함수를 정의합니다.
이 함수는 "0번"스위치와 "1번"스위치의 눌림을 체크합니다.
"1번"스위치가 눌린 경우, throttle(상승/하강) 값을 감소시킵니다.
"0번"스위치가 눌린 경우, throttle(상승/하강) 값을 증가시킵니다.

"checkPitch()" 이름으로 함수를 정의합니다.
이 함수는 "analog_read()" 함수를 이용하여 ADC 3번 채널 변환 값을 읽습니다.
"ADC 3번"채널 값이 400보다 작은 경우, pitch(전진/후진) 값을 50으로 설정합니다.
"ADC 3번"채널 값이 600보다 큰 경우, pitch(전진/후진) 값을 150으로 설정합니다.
"ADC 3번"채널 값이 400 ~ 600 사이인 경우, pitch(전진/후진) 값을 100으로 설정합니다.

```
def checkRoll():
    global roll

    secondRoll = analog_read(2)

    if secondRoll < 400:
        roll = 50
    elif secondRoll > 600:
        roll = 150
    else:
        roll = 100

def checkYaw():
    global yaw

    if GPIO.input(switch_list[2]) == 0:
        yaw = 50
    elif GPIO.input(switch_list[3]) == 0:
        yaw = 150
    else:
        yaw = 100

def checkEmergency():
    global roll
    global pitch
    global yaw
    global throttle

    if GPIO.input(switch_list[4]) == 0:
        throttle = 0
        roll = 100
        pitch = 100
        yaw = 100

def sendDroneCommand():
    ser.write("at+writeh0006")

    ser.write(hex(startBit)[2:4])
    ser.write(hex(commandBit)[2:4])
    ser.write(hex(roll)[2:4])
    ser.write(hex(pitch)[2:4])
    ser.write(hex(yaw)[2:4])

    if throttle < 0x10:
        ser.write('0'+hex(throttle)[2:4])
    else:
        ser.write(hex(throttle)[2:4])

    ser.write('0'+hex(operationBit)[2:4])

    if checkSum < 0x10:
        ser.write('0'+hex(checkSum)[2:4])
    else:
        ser.write(hex(checkSum)[2:4])

    ser.write("\r")
```

"checkRoll()" 이름으로 함수를 정의합니다.
이 함수는 "analog_read()" 함수를 이용하여 ADC 2번 채널 변환 값을 읽습니다.
"ADC 2번"채널 값이 400보다 작은 경우, roll(좌이동/우이동) 값을 50으로 설정합니다.
"ADC 2번"채널 값이 600보다 큰 경우, roll(좌이동/우이동) 값을 150으로 설정합니다.
"ADC 2번"채널 값이 400 ~ 600 사이인 경우, roll(좌이동/우이동) 값을 100으로 설정합니다.

"checkYaw()" 이름으로 함수를 정의합니다.
이 함수는 "2번"스위치와 "3번"스위치의 눌림을 체크합니다.
"2번"스위치가 눌린 경우, yaw(좌회전/우회전) 값을 50으로 설정합니다.
"3번"스위치가 눌린 경우, yaw(좌회전/우회전) 값을 150으로 설정합니다.
"2번 또는 3번"스위치가 눌리지 않은 경우, yaw(좌회전/우회전) 값을 100으로 설정합니다.

"checkEmergency()" 이름으로 함수를 정의합니다.
이 함수는 "4번"스위치의 눌림을 체크합니다.
"4번"스위치가 눌린 경우, "throttle = 0 / roll = 100 / pitch = 100 / yaw = 100"으로 설정합니다.

"sendDroneCommand()" 이름으로 함수를 정의합니다.
이 함수는 펌테크 드론 제어 데이터 패킷을 송신합니다.
하드웨어 시리얼 포트로 "at+writeh0006"을 출력합니다. 이것은 펌테크 드론의 0006 핸들에 HEX로 데이터를 전달하겠다는 AT Command입니다.
하드웨어 시리얼 포트로 startBit를 출력합니다. 이것은 펌테크 드론으로 startBit를 송신하는 것입니다.
하드웨어 시리얼 포트로 commandBit를 출력합니다. 이것은 펌테크 드론으로 commandBit를 송신하는 것입니다.
하드웨어 시리얼 포트로 roll을 출력합니다. 이것은 펌테크 드론으로 roll을 송신하는 것입니다.
하드웨어 시리얼 포트로 pitch를 출력합니다. 이것은 펌테크 드론으로 pitch를 송신하는 것입니다.
하드웨어 시리얼 포트로 yaw를 출력합니다. 이것은 펌테크 드론으로 yaw를 송신하는 것입니다.

하드웨어 시리얼 포트로 throttle을 출력합니다. 이것은 펌테크 드론으로 throttle을 송신하는 것입니다.
하드웨어 시리얼 포트로 operationBit를 출력합니다. 이것은 펌테크 드론으로 "높이 수동 조정"을 송신하는 것입니다.
하드웨어 시리얼 포트로 checkSum을 출력합니다. 이것은 펌테크 드론으로 checkSum을 송신하는 것입니다.
(checkSum = commandBit + roll + pitch + yaw + throttle + operationBit)
하드웨어 시리얼 포트로 "₩r"을 출력합니다. 이것은 데이터의 입력이 완료되었다는 AT Command입니다.

```
def checkCRC():
    global commandBit
    global roll
    global pitch
    global yaw
    global throttle
    global operationBit
    global checkSum

    checkSum = commandBit + roll + pitch + yaw + throttle + operationBit
    checkSum = checkSum & 0x00ff
```

"checkCRC()"이름으로 함수를 정의합니다.
이 함수는 펌테크 드론 제어를 위한 "데이터 패킷"의 checkSum을 계산합니다. checkSum 계산은 "데이터 패킷"으로 사용하는 변수를 이용합니다.

```
switch_list = [22, 27, 17, 23, 24, 25]

for i in range(6):
    GPIO.setup(switch_list[i], GPIO.IN)

print("\nRaspberryPi Data Packet 06\n")

time.sleep(0.5)
ser.write("atd")
ser.write("083a5c1f015b")
ser.write("\r")
time.sleep(0.5)

while True:
    checkThrottle()
    checkPitch()
    checkRoll()
    checkYaw()
    checkEmergency()
    checkCRC()

    sendDroneCommand()
    time.sleep(0.1)
```

"PIO" 포트로 사용할 포트를 'switch_list"라는 리스트에 설정합니다.
GPIO.setup()을 사용하여 6개의 디지털 포트를 입력으로 설정합니다.
출력하고자 하는 디버깅 메시지를 "print()"에 입력합니다.
여기서는 "RaspberryPi Data Packet 06"을 입력하였습니다.
BLE 장치가 시작되기를 기다리기 위해 약 500ms 대기합니다.
하드웨어 시리얼 포트로 "atd083a5c1f015b₩r"을 출력합니다.
BLE 장치가 연결되기를 기다리기 위해 약 500ms 대기합니다.
잠시 후 펌테크 드론과 연결되면 **"조이스틱 쉴드"**의 Status LED가 ON 됩니다.

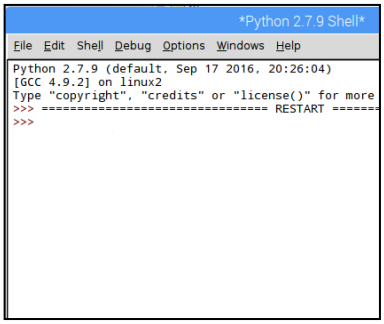
프로그램의 지속적인 반복 수행을 위해 "while True:"를 사용합니다.

"checkThrottle()"함수를 콜해서 throttle값을 계산합니다.
"checkPitch()"함수를 콜해서 pitch값을 계산합니다.
"checkRoll()"함수를 콜해서 roll값을 계산합니다.
"checkYaw()"함수를 콜해서 yaw값을 계산합니다.
"checkEmergency()"함수를 콜해서 비상상태를 체크합니다.
"checkCRC()"함수를 콜해서 checkSum값을 계산합니다.
"sendDroneCommand()" 함수를 콜해서 펌테크 드론 제어용 "데이터 패킷"을 송신합니다.
약 100ms 시간 동안 대기합니다.

※ pitch / roll / yaw의 경우, 50(0x32) / 100(0x64) / 150(0x96)값만 가지도록 했기 때문에 문자 "0"을 출력하는 부분이 추가되지 않았습니다.

※ operationBit의 경우, 1(0x01)을 출력하기 때문에 문자 "0"을 조건 없이 출력하도록 합니다.

※ "atd"명령어 뒤의 숫자(083a5c1f015b)는 연결할 펌테크 드론에 따라 다르게 입력되어야 합니다.



① 라즈베리파이 확장보드에 조이스틱 쉴드를 장착합니다.
② 조이스틱 쉴드에 BLE 장치를 장착합니다.
③ 라즈베리파이 확장보드의 전원을 ON 합니다.
④ 펌테크 드론의 전원을 ON 합니다. (파이썬 프로그램 실행 전)
⑤ Python 2(IDLE) 프로그램으로 Joystick_14_DataPacket_06.py를 오픈합니다.
⑥ Python 2 프로그램을 실행합니다.(Run Module 실행 또는 F5를 누릅니다.)
⑦ 잠시 후 블루투스 연결이 진행됩니다. (연결되면 **"조이스틱 쉴드"**의 Status LED가 ON 됩니다.)
⑧ 0번 스위치 눌러서 throttle 값을 증가시킵니다.
⑨ 펌테크 드론이 상승합니다.
⑩ 2번 스위치 눌러서 yaw 값을 변경시킵니다.
⑪ 펌테크 드론이 좌회전합니다. (Safety Track에서 확인 불가능)
⑫ 3번 스위치 눌러서 yaw 값을 변경시킵니다.
⑬ 펌테크 드론이 우회전합니다. (Safety Track에서 확인 불가능)
⑭ 1번 스위치 눌러서 throttle 값을 감소시킵니다.
⑮ 펌테크 드론이 하강합니다.
⑯ Python Shell에서 "Ctrl+C" 입력하여 프로그램을 종료합니다.
⑰ 라즈베리파이 확장보드의 전원을 OFF 합니다.
⑱ 펌테크 드론의 전원을 OFF 합니다.

30. Joystick_14_DataPacket_07

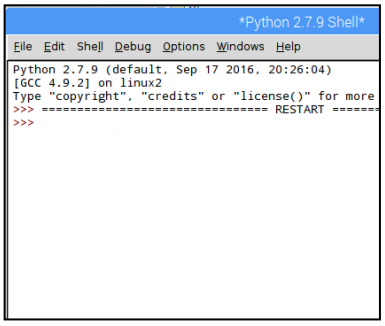
펌테크 드론을 "높이 자동 조정" 모드로 동작시킵니다.

※ "높이 자동 조정" 모드는 "Safety Track" 없이 펌테크 드론을 제어합니다.
※ "높이 자동 조정" 모드는 펌테크 드론의 높이를 임의로 조정할 수 없습니다.

```
import serial
import RPi.GPIO as GPIO
import time
import spidev

#
ser = serial.Serial('/dev/ttyS
spi = spidev.SpiDev()
spi.open(0,0)
spi.max_speed_hz = 100000

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

startBit = 0xf0
commandBit  = 0xa1
roll  = 100
pitch = 100
yaw = 100
throttle = 0
operationBit = 0x05
checkSum = 0
```

"높이 자동 조정" 모드로 동작시키는 경우, operationBit를 0x05로 변경만 하면 됩니다.

※ "높이 자동 조정" 모드 동작 시, 펌테크 드론을 상승 시켰을 때 프로팰러가 느리게 돌면, 비상버튼(PIO 4 포트)을 길게 눌러 "데이터 패킷"을 초기화한 후에 다시 상승 시킵니다.



① 라즈베리파이 확장보드에 조이스틱 쉴드를 장착합니다.
② 조이스틱 쉴드에 BLE 장치를 장착합니다.
③ 라즈베리파이 확장보드의 전원을 ON 합니다.
④ 펌테크 드론의 전원을 ON 합니다. (파이썬 프로그램 실행 전)
⑤ Python 2(IDLE) 프로그램으로 Joystick_14_DataPacket_07.py로 오픈합니다.
⑥ Python 2 프로그램을 실행합니다.(Run Module 실행 또는 F5를 누릅니다.)
⑦ 잠시 후 블루투스 연결이 진행됩니다. (연결되면 **"조이스틱 쉴드"**의 Status LED가 ON 됩니다.)
⑧ 0번 스위치 눌러서 throttle 값을 증가시킵니다.
⑨ 펌테크 드론이 상승합니다. (**1회만 눌러도 자동으로 높이가 조정됩니다. => Safety Track 끝까지 올라갑니다.**)
⑩ 1번 스위치 눌러서 throttle 값을 감소시킵니다.
⑪ 펌테크 드론이 하강합니다. (**1회만 눌러도 자동으로 높이가 조정됩니다. => Safety Track 끝까지 내려옵니다.**)
⑫ Python Shell에서 "Ctrl+C" 입력하여 프로그램을 종료합니다.
⑬ 라즈베리파이 확장보드의 전원을 OFF 합니다.
⑭ 펌테크 드론의 전원을 OFF 합니다.

31. Joystick_20_Drone_Shield

"높이 자동 조정"모드로 변경하고, **"if ~ elif문"**을 이용하여 **"PIO 버튼 체크"**를 통한 블루투스 연결 / 종료, 블루투스 연결 완료 **"응답 메시지 체크"**, **"PIO 버튼과 ADC(조이스틱) 입력"**을 체크하여 펌테크 드론을 제어 하도록 구성하고 프로그램 효율을 위한 **"함수"**를 구성하면 펌테크 드론 제어용 **"최종 조이스틱 쉴드 프로그램"**이 완성됩니다.

최종 소스 코드 설명은 홈페이지의 드론 항목의 “” 문서를 참조하시기 바랍니다.