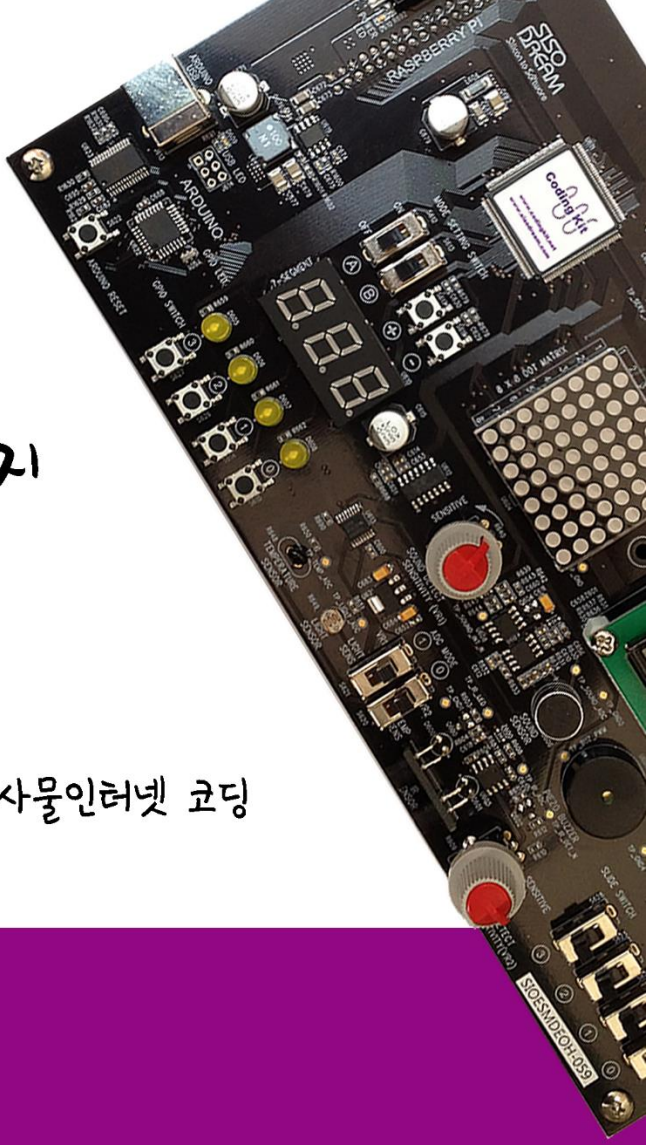
코딩키트



블록 코딩에서 사물인터넷 코딩까지
쉽고 재미있게 배울 수 있어요.

블록 코딩 / 아두이노 코딩 / 라즈베리파이 코딩 / 사물인터넷 코딩

코딩북5 - 코딩키트 확장판

사물인터넷(IoT) 코딩

아두이노와 라즈베리파이 연동 코딩

SISO DREAM
Silicon to Software

(주)시소드림 SISODREAM Inc.

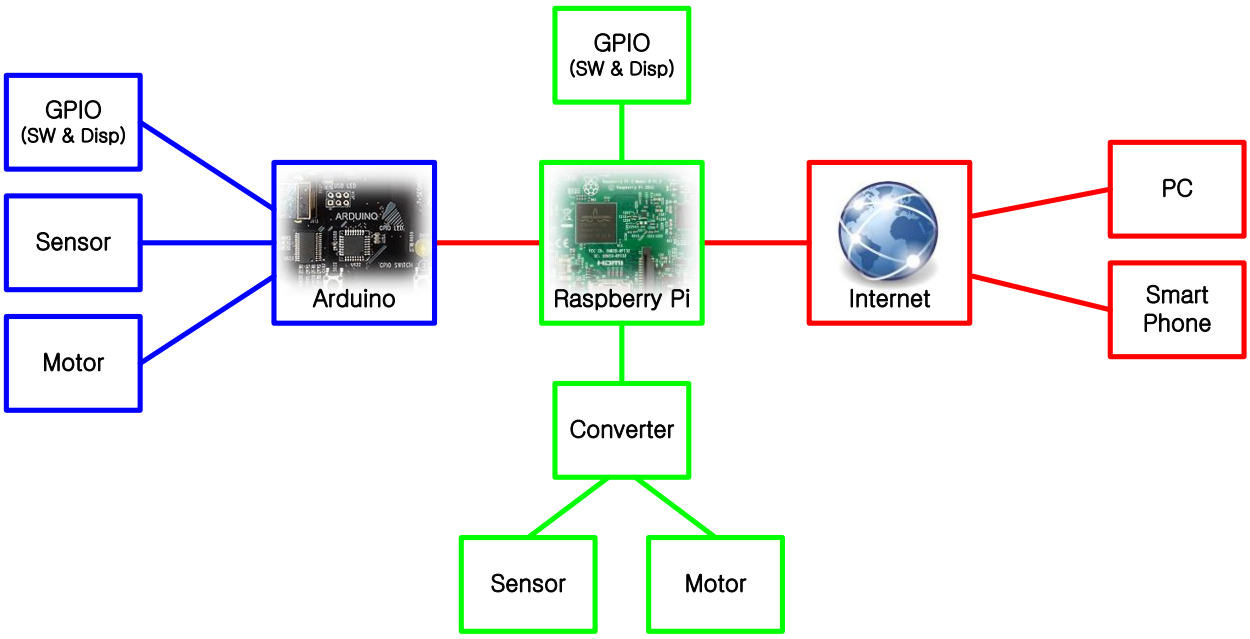
- 본 문서의 저작권 -

발행일 : 2017 년 7 월 26 일 (초판)
발행처 : ㈜시소드림
연락처 : ck@sisodream.com / 031-757-7755

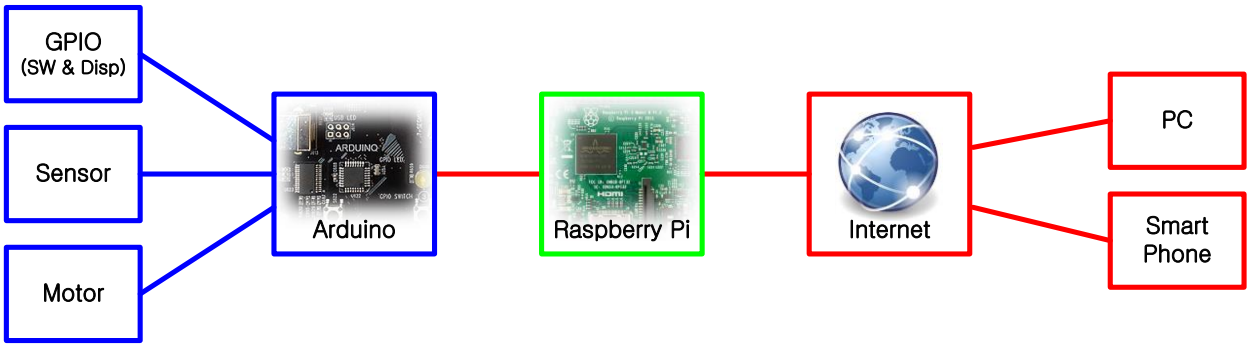
목 차

[코딩키트 확장판 소개]

지금까지 코딩키트로 아두이노 코딩과 라즈베리파이 코딩을 배웠습니다. 이제부터는 지금까지 배웠던 아두이노와 라즈베리파이 코딩을 바탕으로 더 다양하고 재미있는 코딩을 배우려고 합니다.

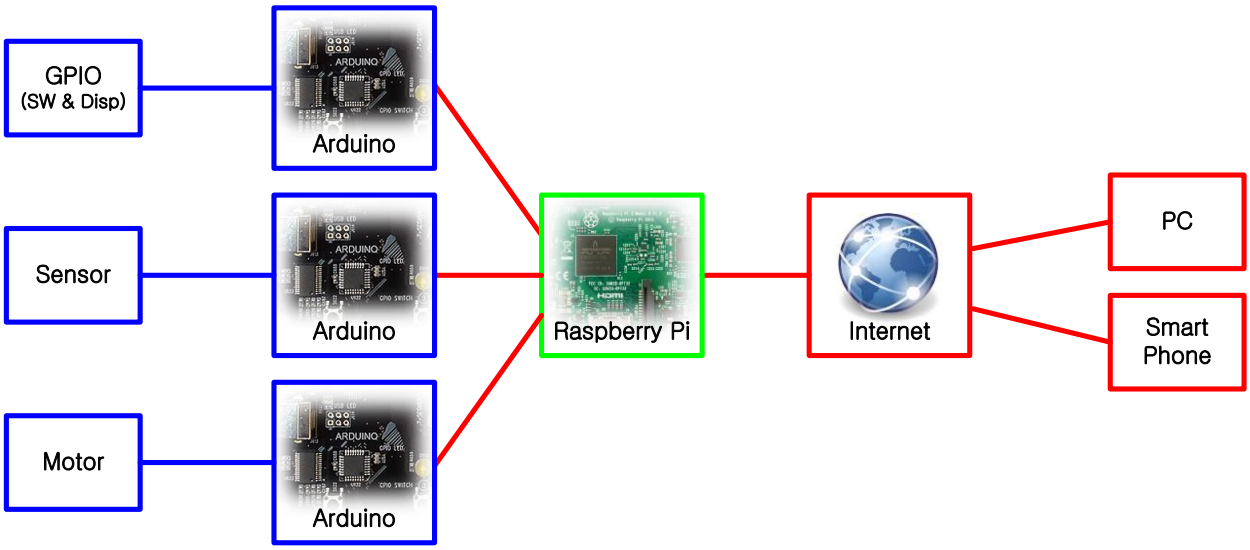


위의 그림에서 파란색 부분은 아두이노 코딩입니다. 아두이노에서 GPIO 컨트롤하는 법을 배웠고, 센서의 입력을 받았으며 모터를 컨트롤하였습니다. 연두색 부분은 라즈베리파이 코딩입니다. 이 두 부분을 바탕으로 이번에 붉은색 부분을 공부해 볼 것입니다. 먼저 아두이노와 라즈베리파이가 서로 통신하는 코딩을 공부합니다. 그리고 라즈베리파이를 인터넷에 연결하여 PC 또는 스마트폰과 통신하는 방법을 공부합니다. 요즘 한참 이슈가 되고 있는 사물인터넷(Internet of Things : IoT) 코딩을 공부하는 것입니다. 사물인터넷 코딩의 포맷은 다음 그림과 같습니다.



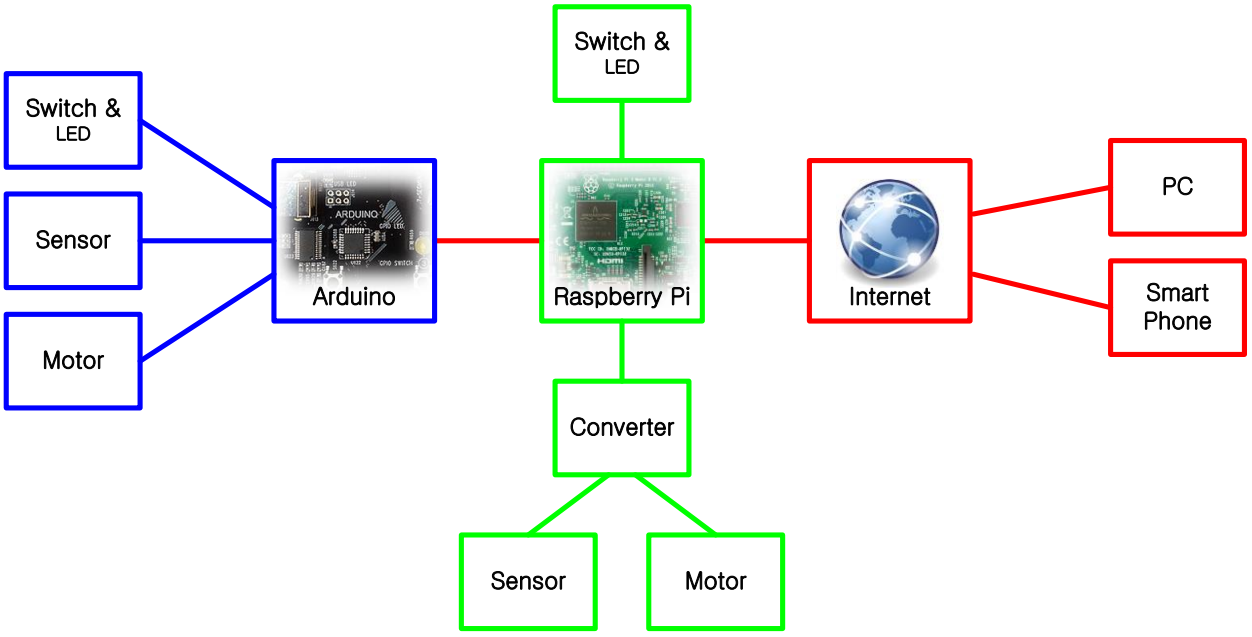
각종 디바이스는 아두이노에 연결되고 인터넷은 라즈베리파이에 연결됩니다. 이것은 아두이노와 라즈베리파이의 특징을 잘 활용한 구조입니다. 라즈베리파이는 매우 복잡한 고성능의 컴퓨터입니다. 또한 응용 소프트웨어(Application Software)가 하드웨어 위에서 원활하게 동작할 수 있도록 OS(Operating System) 도 설치되어 있습니다. 그리고 유무선 인터넷에 연결할 수 있도록 해 주는 하드웨어가 장착되어

있습니다. 그래서 기본적으로 라즈베리파이는 바로 인터넷에 연결하여 사용할 수 있습니다. 하지만 디바이스를 바로 컨트롤하는 일에는 조금 어려움을 겪고 있습니다. 응용 소프트웨어에서 디바이스까지 연결하는데는 중간에 OS 라는 것이 있어서 항상 물어보아야 합니다. 이 때 시간 지연이 있고 이것이 매우 짧은 시간이긴 하지만 디바이스 컨트롤에 미묘한 차이를 만들어냅니다. 이에 반해서 아두이노는 고성능은 아니지만 디바이스 컨트롤에만 집중하고 있기 때문에 디바이스를 바로 바로 컨트롤할 수 있고 센서등의 디바이스의 변화에 즉각적으로 대처할 수 있습니다. 또한 아두이노에 쓰이는 CPU 는 가격도 저렴하여 실제 산업 현장에서는 어떤 디바이스 하나에 아두이노 CPU 하나를 사용하여 그 CPU 에서는 해당 디바이스에만 집중하도록 합니다. 그래서 산업 현장에서는 다음 그림과 같이 여러대의 저가 아두이노 CPU 와 하나의 고가 라즈베리파이 CPU 를 연결하여 사물인터넷을 구현하는 경우도 많습니다.



위의 그림에서는 각각의 아두이노가 다른 디바이스에 연결되어 있는데, 이 디바이스들이 모두 같은 경우도 많습니다.

다시 한번 더 다음 그림을 살펴 보며 아두이노와 라즈베리파이 특징에 대해서 추가 설명을 하겠습니다.

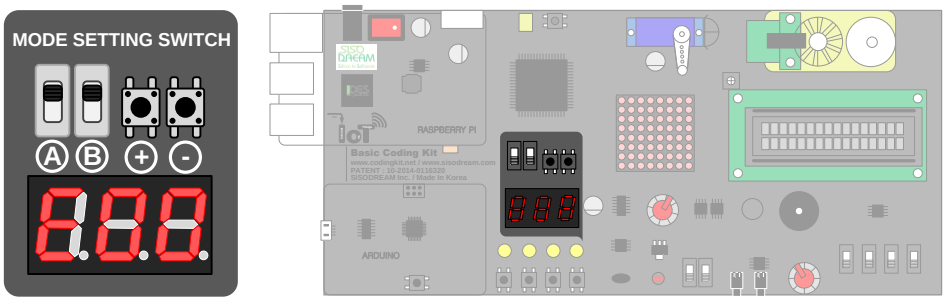


위의 그림을 잘 보면 아두이노는 센서 및 모터를 바로 연결하지만 라즈베리파이는 센서와 모터를 바로 연결하지는 못하고 컨버터(Converter)라는 것을 거칩니다. 이것은 앞에서 설명했듯이 라즈베리파이는 매우 복잡한 CPU 에 OS 가 설치되어 있어 즉각적으로 디바이스를 컨트롤할 수 없기 때문에 컨버터라는 것을 사용하고 이 컨버터가 위에서 설명한 아두이노의 역할을 하는 것입니다. 라즈베리파이 코딩을 배울 때 이 컨버터는 코딩키트 보드에 장착되어 있었습니다.

앞으로 여러분은 아두이노와 라즈베리파이가 통신하는 것을 배우고 라즈베리파이가 인터넷에 접속하여 필요한 정보를 가지고 오는 것을 배울 것입니다. 사물인터넷의 실질적인 예제를 통하여 사물인터넷의 개념을 이해하고 구현할 수 있는 코드도 배우겠습니다.

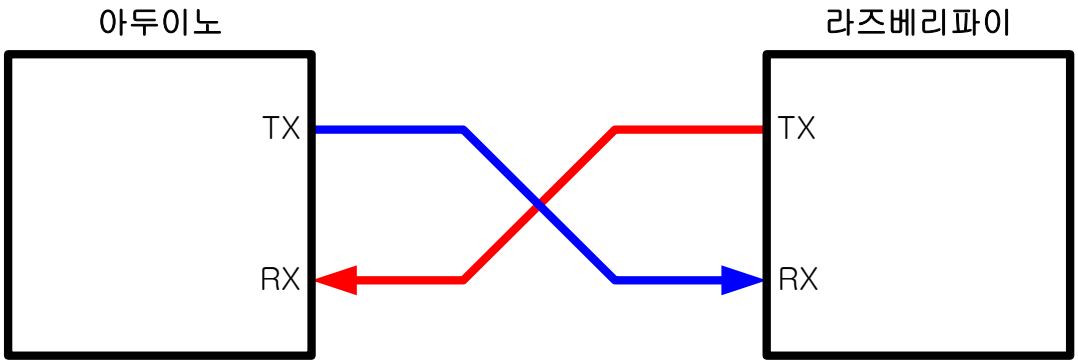
[아두이노와 라즈베리파이 시리얼통신하기]

라즈베리파이 교재를 참고하여 라즈베리파이를 코딩키트에 연결합니다. 라즈베리파이를 원격으로 연결하기 위해서는 라즈베리파이와 PC 를 크로스 케이블로 연결합니다. 라즈베리파이를 원격이 아닌 직접 컨트롤하기 위해서는 라즈베리파이에 모니터, 키보드, 마우스를 연결합니다. 아두이노는 PC 와 USB 케이블로 연결합니다. 이제 전원을 올리고 그림과 같이 스위칭 ID 를 E00 으로 선택합니다.

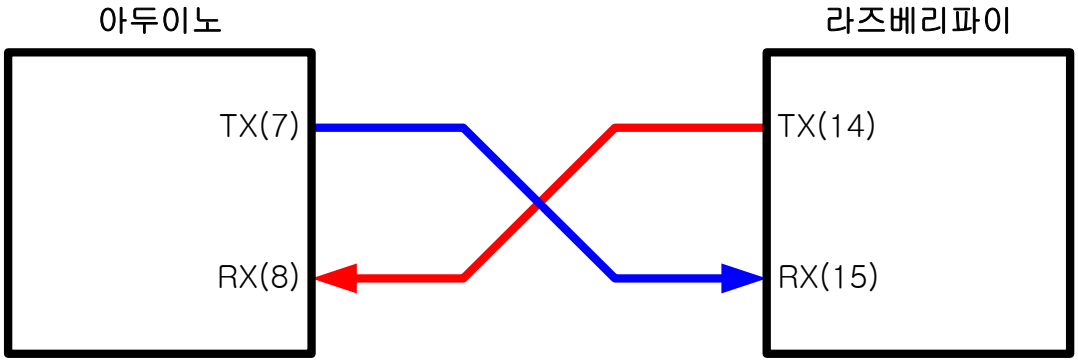


스위칭 ID 를 E00 으로 선택하면 아두이노와 라즈베리파이는 UART (Universal asynchronous receiver/transmitter) 로 연결되어 있습니다. 보통 시리얼(Serial) 통신이라고 하는 통신 규약입니다. 코딩키트에서는 이 통신 규약을 이용하여 아두이노와 라즈베리파이가 서로 통신을 할 것입니다. 시리얼 통신은 여러분이 아두이노를 공부하실 때 많이 들어보셨을 것입니다. 아두이노 프로그램에서 시리얼 모니터를 사용해 보셨지요. 이 시리얼 모니터도 아두이노와 PC 사이에 시리얼 통신을 하여 데이터를 주고 받는 것입니다. 관련하여서는 아두이노 코딩 교재를 참고바랍니다.

UART 는 2 개의 신호선으로 구성이 됩니다. 한 개는 데이터를 보내는 신호선, 다른 하나는 데이터를 받는 신호선입니다. 아두이노에서 보내는 신호핀(Transmitter Pin : TX)은 라즈베리파이의 받는 신호핀(Receiver Pin : RX)에 연결합니다. 반대로 라즈베리파이에서 보내는 신호핀은 아두이노에서 받는 신호핀에 연결합니다.



코딩키트에서 각각의 핀 번호은 다음과 같습니다.



그럼 먼저 아두이노에서 라즈베리파이로 데이터를 전송하는 코드를 작성해 보겠습니다. 그렇게 하려면 아두이노에서는 시리얼 포트로 데이터를 내보내는 코드를 작성하고 라즈베리파이에서는 시리얼 포트로 데이터를 받는 코드를 작성해 보겠습니다.

아두이노 코드는 다음과 같습니다.

```
#include <SoftwareSerial.h>

#define SW_SERIAL_TX 7
#define SW_SERIAL_RX 8

#define CK_SERIAL_BAUDRATE 19200

SoftwareSerial ckSerial(SW_SERIAL_RX, SW_SERIAL_TX);

void setup() {
  ckSerial.begin(CK_SERIAL_BAUDRATE);
}

void loop() {
  ckSerial.print("Hello");
  delay(1000);
}
```

아두이노에서 PC 연결되는 시리얼 포트의 핀 번호는 0, 1 번입니다. 이 시리얼 포트를 이용하여 라즈베리파이와 연결할 수도 있습니다. 하지만, 그렇게 하면 아두이노 코드를 업로드할 때 충돌이 생깁니다. 그래서 다른 시리얼 포트를 사용하는데, 아두이노에서는 별도의 하드웨어 시리얼 포트가 있지는 않습니다. 하지만, 소프트웨어적으로 만들어 놓은 시리얼 포트가 있습니다. 이것을 소프트웨어 시리얼 (Software Serial) 포트라고 부릅니다. 코딩키트에서는 이 소프트웨어 시리얼 포트를 이용하여 라즈베리파이와 통신을 하겠습니다.

위의 코드를 보면 이 소프트웨어 시리얼을 사용하기 위해서 “SoftwareSerial.h” 파일을 include 했습니다. 아두이노의 디지털 GPIO 핀들은 모두 소프트웨어 시리얼로 사용할 수 있습니다. 이번 예제에서는 7 번과 8 번을 소프트웨어 시리얼 포트로 사용하겠습니다. 그 중 전송 포트로는 7 번을, 수신 포트로는 8 번을 사용하겠습니다. 그리고 이 포트들을 이용하여 다음과 같이 소프트웨어 시리얼 포트를 선언합니다.

```
SoftwareSerial ckSerial(SW_SERIAL_RX, SW_SERIAL_TX);
```

코딩키트 시리얼이라는 뜻으로 “ckSerial” 이라고 명명했습니다. “setup()” 함수에서 ckSerial 을 초기화 합니다. 초기화 할 때 시리얼 통신의 속도를 정합니다. 이 속도를 보통 보드레이트(Baudrate)라고 부릅니다. 예제 코드에서 이 보드레이트을 19200 으로 미리 정해 두었습니다. 다음과 같이 이 값으로 시리얼 포트는 초기화 됩니다.

```
#define CK_SERIAL_BAUDRATE 19200

ckSerial.begin(CK_SERIAL_BAUDRATE);
```

“loop()” 함수를 보시면 ckSerial 보트로 데이터를 전송하는 코드를 볼 수 있습니다. 다음과 같이 아두이노 교재에서 배웠던 시리얼 모니터를 사용하는 것과 같이 “print()” 함수를 사용합니다.

```
ckSerial.print("Hello");
```

이렇게 하면 “Hello” 라는 단어가 라즈베리파이로 전송이 됩니다. 그러면 이제 라즈베리파이 코드를 작성하여 이 단어를 받아 보겠습니다.

라즈베리파이 코드는 다음과 같습니다.

```
import serial

port = serial.Serial("/dev/ttyAMA0", baudrate=19200, timeout=3.0)

while True:
   msg = port.read(5)
   print "Message from Arduino : ", msg
```

라즈베리파이 파이썬 코드도 아두이노 코드와 문법만 다르지 코드 구조는 비슷합니다. 먼저 시리얼포트를 사용하기 위해서 “serial” 라이브러리를 import 합니다. 다음으로는 아두이와 마찬가지로 시리얼 포트를 선언하면서 동시에 초기화를 합니다. 코드는 다음과 같습니다.

```
port = serial.Serial("/dev/ttyAMA0", baudrate=19200, timeout=3.0)
```

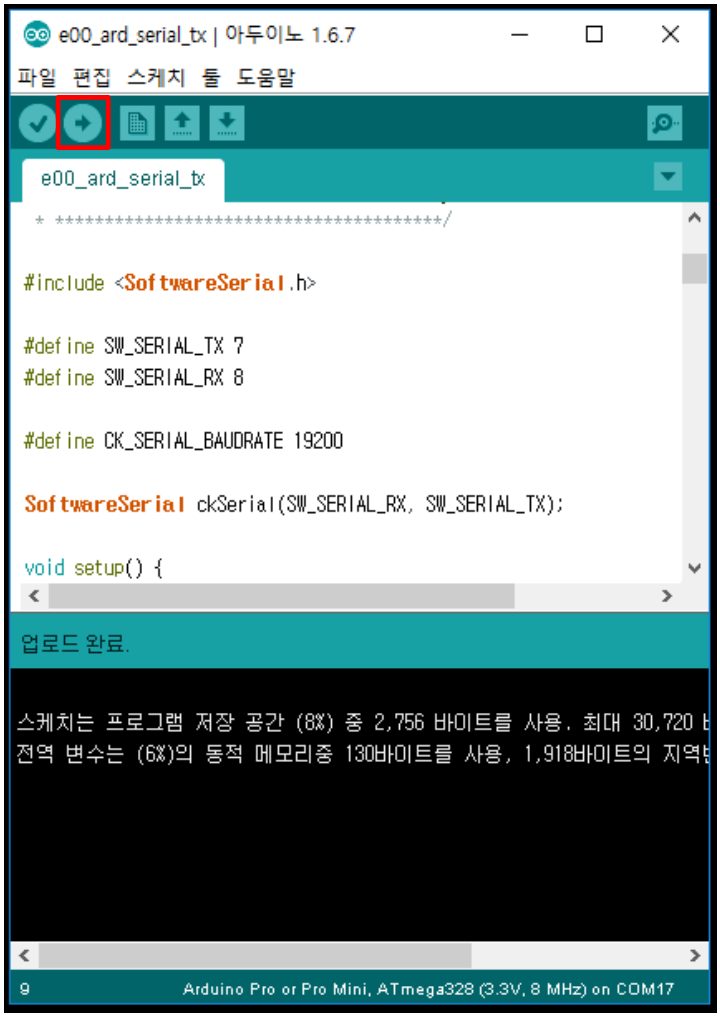
시리얼 포트를 초기화하는 함수로는 위의 코드와 같이 “serial.Serial()” 함수를 사용합니다. 이 함수의 첫번째 파라미터는 사용할 시리얼 포트가 연결된 파일의 이름입니다. 코딩키트의 라즈베리파이에서는 이 파일로 “/dev/ttyAMA0” 를 사용할 것입니다. 두번째 파라미터는 시리얼 포트의 속도입니다. 아두이노의 코드와 같은 속도를 설정해 주어야 합니다. 이 속도가 다르면 서로 통신을 할 수 없습니다. 마지막 세번째 파라미터로는 타임아웃(timeout)이 있습니다. 타임아웃은 이 시간동안 기다려도 상대편으로부터 어떤 데이터도 오지 않으면 기다리는 것을 멈추라는 의미입니다. “timeout=3.0” 은 3 초를 기다리라는 뜻입니다.

다음의 “while” 구문은 아두이노의 “loop()” 함수와 같은 기능을 합니다. “while” 구문 안에 “port.read(5)” 함수가 있습니다. 이 함수의 의미는 시리얼 포트로 5 바이트(Byte)의 데이터를 읽어 오라는 의미입니다. 그래서 이 함수는 시리얼 포트의 RX 핀으로 5 바이트의 데이터가 올 때까지 기다립니다. 5 바이트를 모두 받으면 함수를 끝내고 받은 데이터를 넘겨 줍니다. 초기화 함수 “serial.Serial()” 의 타임아웃 값은 이

함수에서 사용되는데, 만약 5 바이트가 타임아웃 시간인 3 초 안에 오지 않으면 이 함수를 데이터 5 바이트를 모두 다 못 받고 끝을 냅니다.

여기서 5 바이트는 아두이노에서 보내줄 “Hello”라는 단어의 글자수 입니다. 문자를 표시하는 아스키(ASCII) 코드는 영문 한 글자당 1 바이트를 사용하므로 “Hello” 라는 5 글자 단어는 5 바이트를 씁니다.

이렇게 하여 아두이노 코드와 라즈베리파이 코드가 모두 완성되었습니다. 이제 아두이노 코드를 아두이노 프로그램에서 업로드 아이콘을 눌러 업로드합니다.



라즈베리파이에서는 다음과 같이 하여 코드는 실행을 시킵니다. 그리고”Hello” 라는 단어를 잘 받아 출력한 것을 확인합니다.

```
pi@raspberrypi:~/e00_rsp_example $ sudo python e00_rsp_serial_rx.py
Message from Arduino :  Hello
Message from Arduino :  Hello
Message from Arduino :  Hello
Message from Arduino :  Hello
Message from Arduino :  Hello
```

라즈베리파이에서 디바이스를 파일에 연결하는 것과 관련하여 약간의 부연 설명을 드립니다. 라즈베리파이는 리눅스(Linux) 계열의 OS 를 사용합니다. 이 리눅스는 하드웨어 디바이스를 파일에 연결하여 사용합니다. 라즈베리파이에는 2 개의 시리얼 포트 디바이스가 있습니다. 코딩키트의 라즈베리파이에서는 이 2 개의 시리얼 포트 중 하나를 이전 예제에서 사용한 "/dev/ttyAMA0" 파일에 연결해 두었습니다. 나머지 하나는 "/dev/ttyS0" 에 연결해 두었습니다. 이것은 다음과 같은 리눅스 명령어 확인하실 수 있습니다.

```
pi@raspberrypi:~ $ ls -l /dev |grep serial
lrwxrwxrwx 1 root root           7 Mar 29 09:15 serial0 -> ttyAMA0
lrwxrwxrwx 1 root root           5 Mar 29 09:15 serial1 -> ttyS0
```

< 예제 코드 : 시리얼 포트를 이용하여 아두이노에서 라즈베리파이로 데이터 전송 >

< 아두이노 코드 : e00_ard_serial_tx.ino >

< 라즈베리파이 코드 : e00_rsp_serial_rx**.py** >

이번에는 라즈베리파이에서 아두이노로 데이터를 전송하는 것을 해 보겠습니다. 전송하는 쪽인 라즈베리파이 코드를 먼저 보겠습니다.

```
import serial
import time

port = serial.Serial("/dev/ttyAMA0", baudrate=19200, timeout=3.0)

while True:
    port.write("Codingkit\n")
    time.sleep(1)
```

이전에 보았던 라즈베리파이에서 수신하는 코드와 비슷합니다. 수신하는 코드에서는 "read()" 함수를 사용하였다면 송신하는 코드에서는 "write()" 함수를 사용합니다. "write()" 함수의 인수로 전송하고자 하는 데이터를 써 주면 되는 것입니다. 이번에는 "Codingkit" 라고 전송합니다. 끝에 "\n" 을 붙인 것은 이 단어를 다 쓰고 줄을 바꾸라는 의미입니다. 마지막에 1 초 시간 지연을 하였습니다. 이렇게 라즈베리파이 코드는 매우 간단하게 끝이 났습니다.

그럼 이제 이 데이터를 수신하는 아두이노 코드를 보겠습니다.

```
#include <SoftwareSerial.h>

#define SW_SERIAL_TX 7
#define SW_SERIAL_RX 8
```

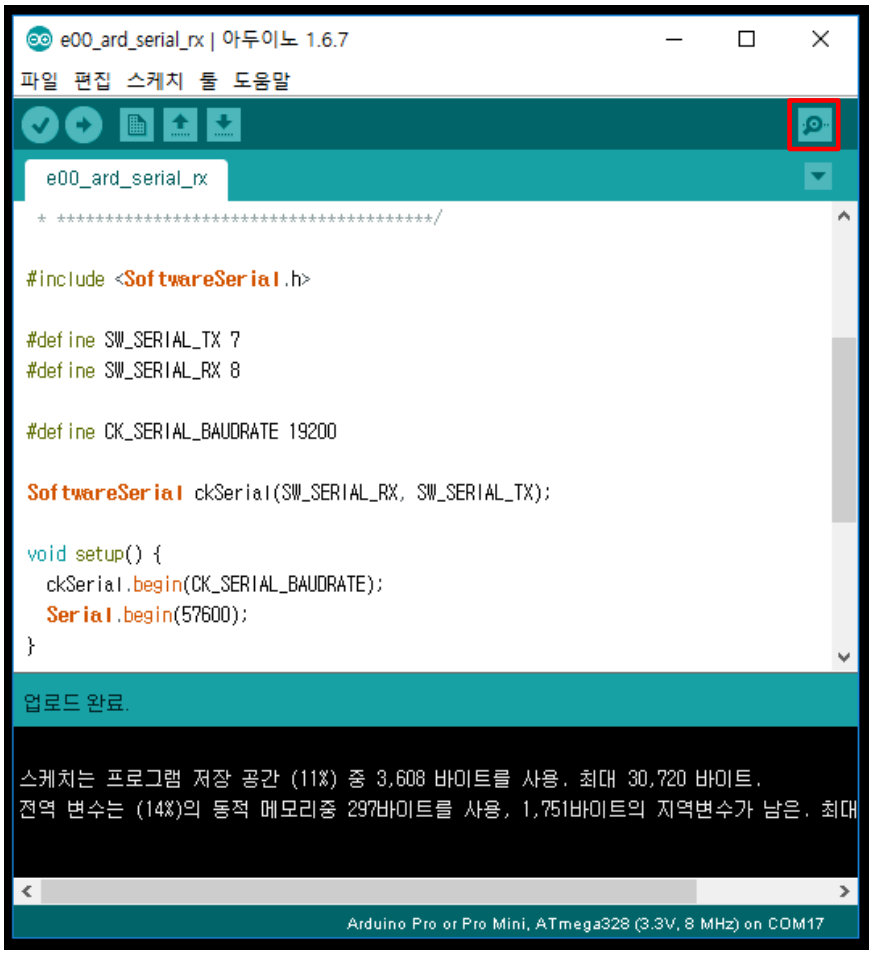
```
#define CK_SERIAL_BAUDRATE 19200

SoftwareSerial ckSerial(SW_SERIAL_RX, SW_SERIAL_TX);

void setup() {
  ckSerial.begin(CK_SERIAL_BAUDRATE);
  Serial.begin(57600);
}

void loop() {
  int rxSize = ckSerial.available();
  if (rxSize > 0) {
    for (int i = 0; i < rxSize; i++) {
      char c = ckSerial.read();
      Serial.print(c);
    }
  }
}
```

아두이노에서는 라즈베리파이로부터 받은 데이터를 PC 의 시리얼 모니터로 보내서 받은 데이터를 확인할 것입니다. 그래서 다음과 같이 시리얼 모니터용 시리얼 포트를 초기화 하였습니다.

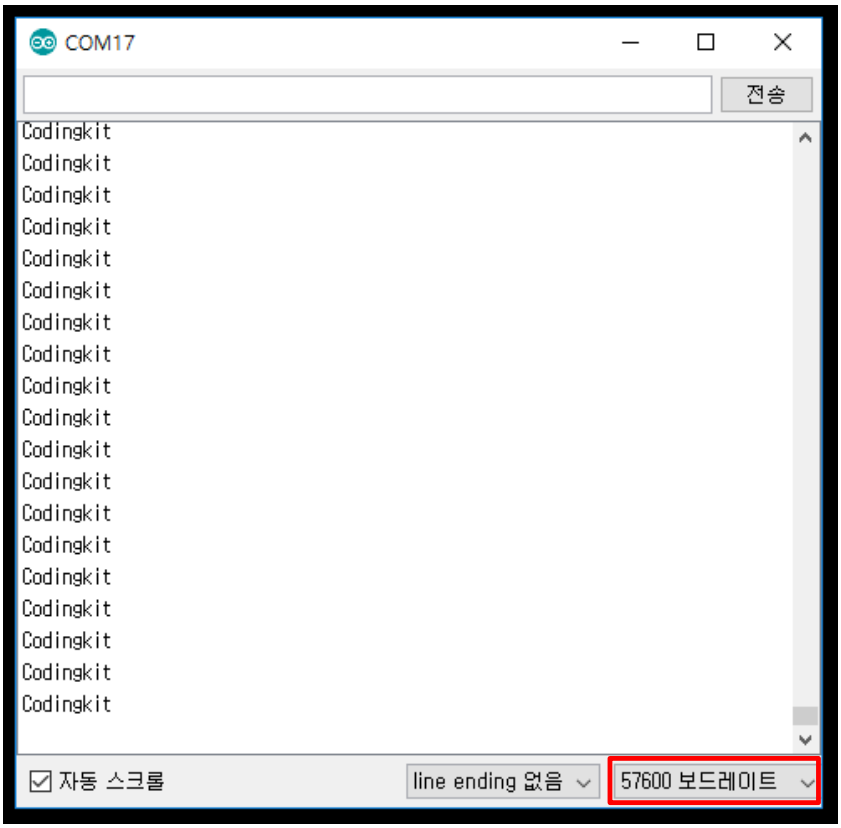
```
Serial.begin(57600);
```

“loop()” 함수를 살펴보겠습니다. “ckSerial.available()” 함수는 시리얼 포트로 받은 데이터의 크기를 바이트 단위로 반환합니다. 예제 코드에서는 이 값을 “rxSize” 변수에 저장하였습니다. 이 변수 값이 0 보다 크면 시리얼 포트로 수신한 데이터가 있다는 의미입니다. 그래서 “if” 구문에서 이 조건을 검사합니다. 만약 0 보다 크면 “for” 문을 이용하여 수신한 데이터를 “rxSize” 변수 크기만큼 읽어 드립니다. 이것이 “for” 문 코드로 작성되어 있습니다. “ckSerial.read()” 함수를 이용하면 한 바이트씩 데이터를 읽어들일 수 있습니다. 이렇게 읽은 데이터는 “c” 변수에 저장이 되고 이 변수는 “Serial.print()” 함수를 이용하여 시리얼 모니터로 출력이 됩니다.

이제 코드를 코딩키트로 업로드합니다. 라즈베리파이 코드를 실행합니다. 아두이노 프로그램에서 다음 아이콘을 눌러 시리얼 모니터를 실행합니다.



시리얼 모니터에 다음과 같이 출력되는 것을 확인하실 수 있을 것입니다.



혹시 글자로 제대로 표시되지 않는다면 위의 그림에서 붉은 박스로 표시한 보드레이트가 “57600 보드레이트” 로 되어 있는지 확인합니다.

< 예제 코드 : 시리얼 포트를 이용하여 라즈베리파이에서 아두이노로 데이터 전송 >

< 아두이노 코드 : e00_ard_serial_rx.ino **>**

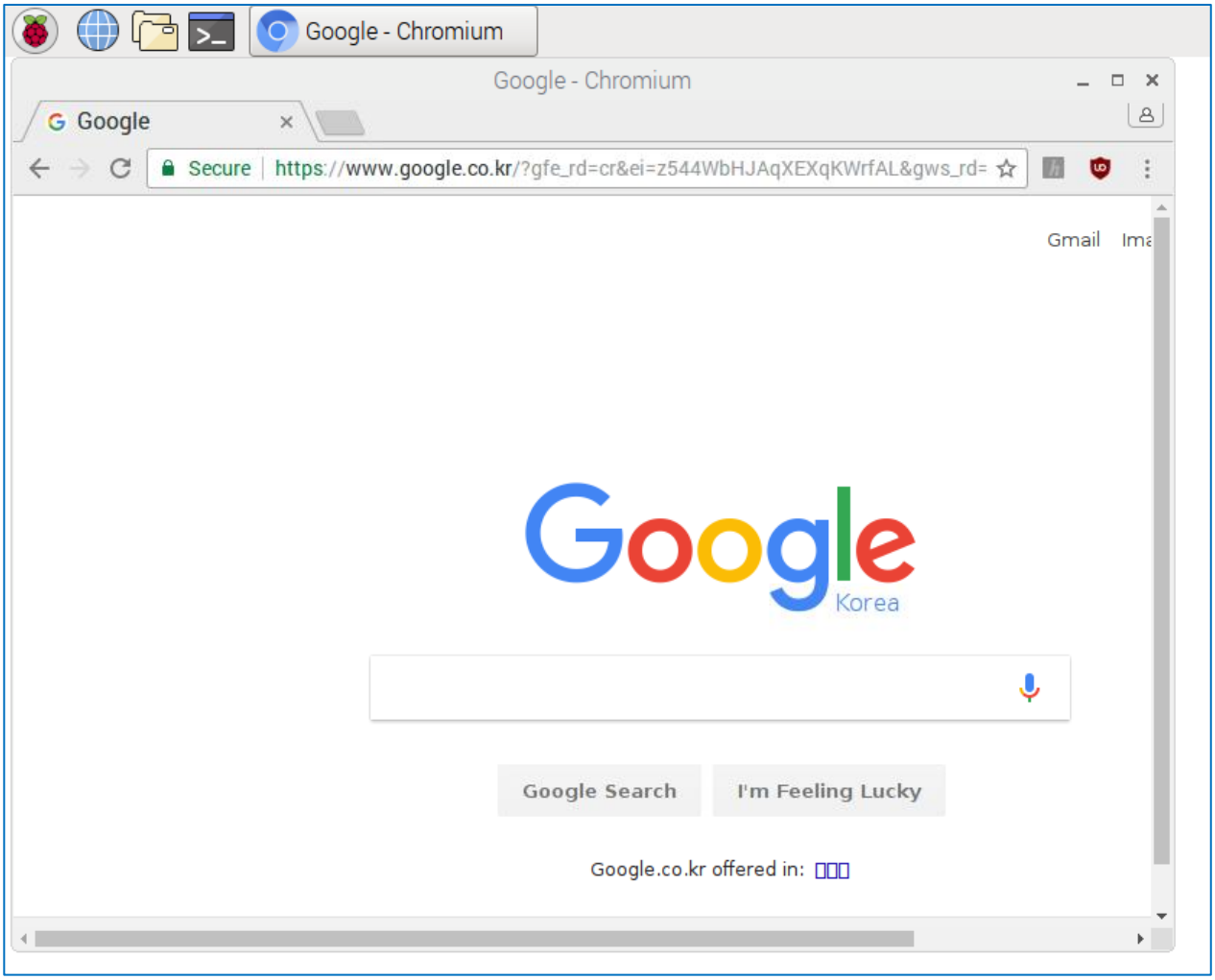
< 라즈베리파이 코드 : e00_rsp_serial_tx**.py >**

[인터넷 접속 파이썬 코드]

라즈베리파이3의 보드에는 유선랜과 무선랜(WiFi) 둘 다 장착되어 있습니다. 그래서 라즈베리파에서 바로 인터넷에 연결할 수 있습니다. 코딩키트에서 제공해 드리는 SD 카드에는 라즈베리파이3의 무선랜을 통하여 인터넷에 연결하는 설정이 되어 있습니다.

다음 그림과 같이 라즈베리파이 OS 화면의 우측 상단의 WiFi 아이콘을 눌러 무선공유기와 연결합니다. 암호가 필요하다면 암호를 입력합니다.

아래 그림과 같은 라즈베리파이 OS 화면에서 좌측상단의 지구본모양 아이콘을 클릭하면 웹브라우저를 띄울 수 있습니다. 주소창에 www.google.co.kr 을 입력하면 다음과 같은 화면을 볼 수 있습니다.



브라우저에서 보이는 인너텟 화면은 웹서버에 저장된 파일입니다. 브라우저, 웹서버 등에 관해서는 다음에 더 자세히 설명 드리겠습니다.

다음과 같이 브라우저의 주소창에 “http://www.ckiot.net/hello.html” 을 입력해 보세요. 그러면 다음과 같이 “Hello World!!” 라고 보여줄 것입니다.

이것은 웹서버에 hello.html 이라는 파일이 있는 것이고 이 파일은 브라우저에 위와 같은 화면 출력될 수 있도록 하는 내용을 담고 있을 것입니다. 라즈베리파이에서 이 파일의 내용을 가지고 와서 볼 수 있습니다. 그 코드는 다음과 같습니다.

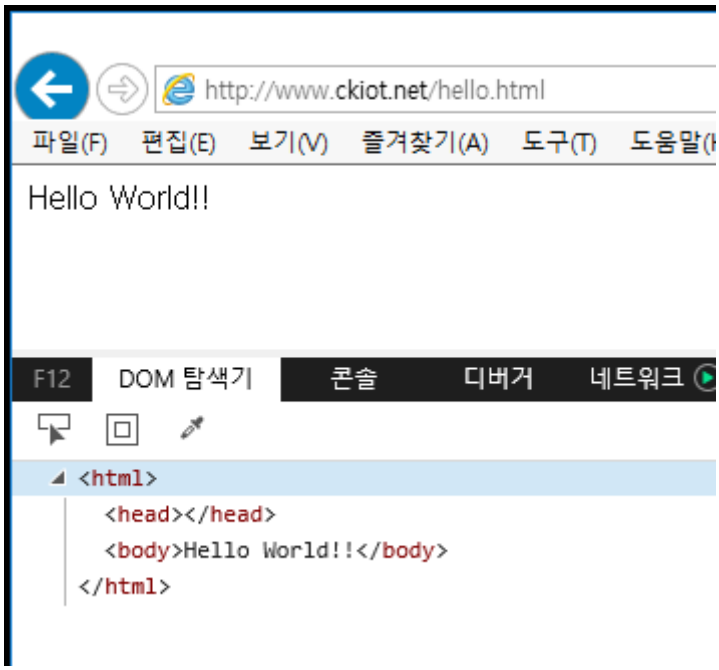
```
import urllib

read_file = urllib.urlopen("http://www.ckiot.net/hello.html")
file_data = read_file.read()
print file_data
read_file.close()
```

이 코드를 실행시키면 다음과 같이 출력됩니다.

```
pi@raspberrypi:~/ck_iot $ sudo python ck_iot_file_hello.py
<html>
<body>
Hello World!!
</body>
</html>
```

웹서버에는 “hello.html” 파일이 있고 이 파일의 내용은 위와 같다는 것을 알 수 있습니다. 다음과 같이 인터

넷 익스프로러 브라우저에서 F12 키를 눌러 “hello.html” 파일의 내용을 확인할 수도 있습니다.



파이썬 코드로 확인한 내용과 같은 것을 알 수 있습니다.

이제 파이썬 코드를 분석해 보겠습니다.

```
import urllib

read_file = urllib.urlopen("http://www.ckiot.net/hello.html")
file_data = read_file.read()
print file_data
read_file.close()
```

위의 코드에서 “urllib” 라이브러리 모듈은 URL 로 정의된 파일을 찾아서 그 파일의 정보와 내용을 볼 수 있도록 해 주는 파이썬 모듈입니다. URL(Uniform Resource Locator)이란 일반적으로 인터넷상에서 주소를 나타냅니다. 이 주소를 통해서 인터넷상의 대부분의 파일을 가리킬 수 있습니다.

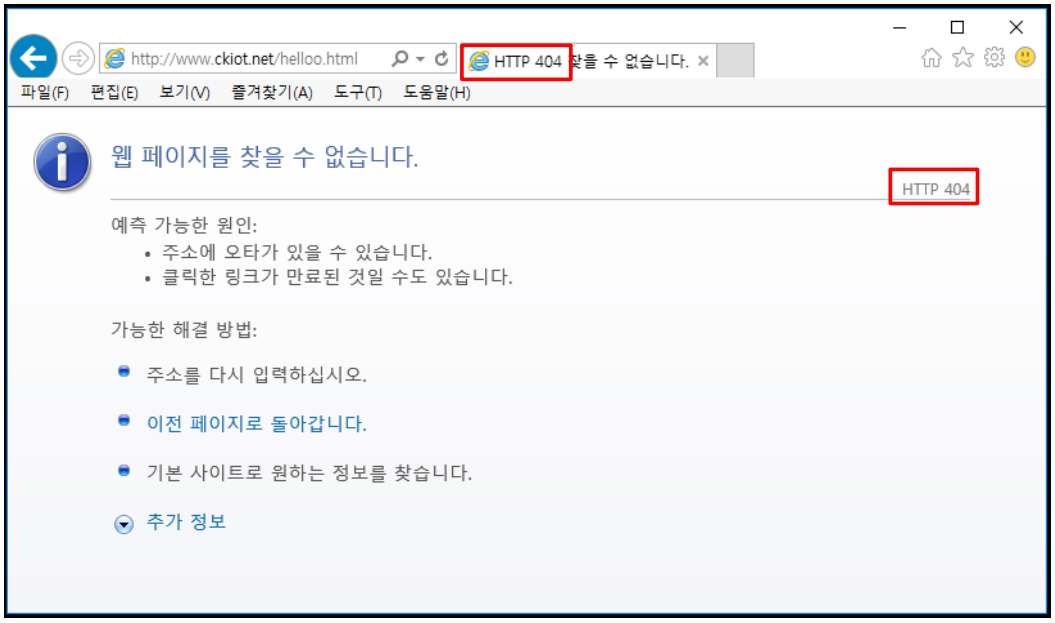
urllib.urlopen() 함수는 함수 인자로 url 주소를 받고 이 주소의 인터넷 사이트를 연결합니다. urllib 의 read() 함수는 url 주소 파일의 내용을 읽어 옵니다. close() 함수는 urlopen() 함수의 반대로 연결된 인터넷 사이트를 끊습니다.

Urllib 모듈의 주요 함수로는 다음과 같은 것들이 있습니다.

geturl() : 인터넷 주소를 가지고 옵니다.

infor() : 헤더(Header) 정보를 가지고 옵니다.

getcode() : 코드 값을 가지고 옵니다. 정상일 때는 200 이고 해당 파일이 없을 때는 404 값을 가지고 옵니다. 다음과 같이 주소를 잘 못 입력하면 404 코드 값이 출력됩니다.



위의 함수들을 다음 파이썬 코드로 확인해 보겠습니다.

```
import urllib

read_file = urllib.urlopen("http://www.ckiot.net/hello.html")
file_data = read_file.read()

print "read"
print file_data

print "------------------"
print "geturl"
print read_file.geturl()

print "------------------"
print "info"
print read_file.info()

print "------------------"
print "getcode"
print read_file.getcode()

read_file.close()
```

위의 코드를 실행시키면 다음과 같습니다.

```
pi@raspberrypi:~/ck_iot $ sudo python ck_iot_urllib.py
read
<html>
<body>
Hello World!!
</body>
</html>

------------------
geturl
http://www.ckiot.net/hello.html
------------------
info
Server: nginx
Date: Tue, 02 May 2017 02:28:51 GMT
Content-Type: text/html
Connection: close
Vary: Accept-Encoding
P3P: CP='NOI CURa ADMa DEVa TAIa OUR DELa BUS IND PHY ONL UNI COM
E'

------------------
getcode
200
```

주소를 다음과 같이 잘못 입력하면 결과는 다르게 출력됩니다.

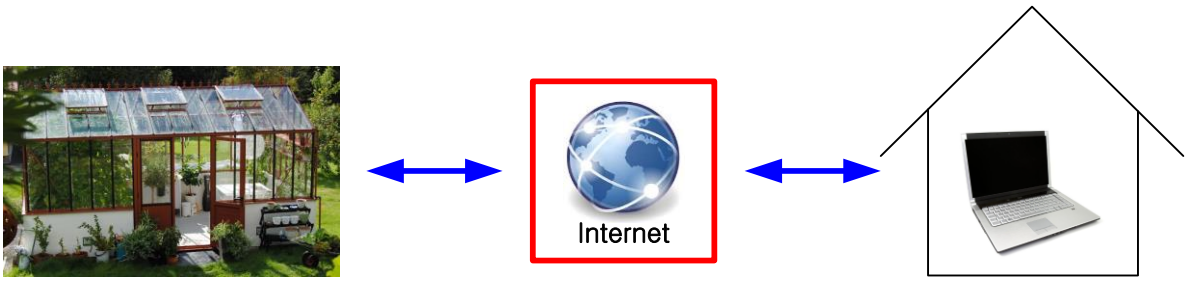
read_file = urllib.urlopen("http://www.ckiot.net/helloo.html")

```
pi@raspberrypi:~/ck_iot $ sudo python ck_iot_urllib.py
read
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>404 Not Found</TITLE>
</HEAD><BODY>
<H1>Not Found</H1>
The requested URL /helloo.html was not found on this server.<P>
</BODY></HTML>

------------------
geturl
http://www.ckiot.net/helloo.html
------------------
info
Server: nginx
Date: Tue, 02 May 2017 02:33:28 GMT
Content-Type: text/html; charset=iso-8859-1
Connection: close
Vary: Accept-Encoding

------------------
getcode
404
```

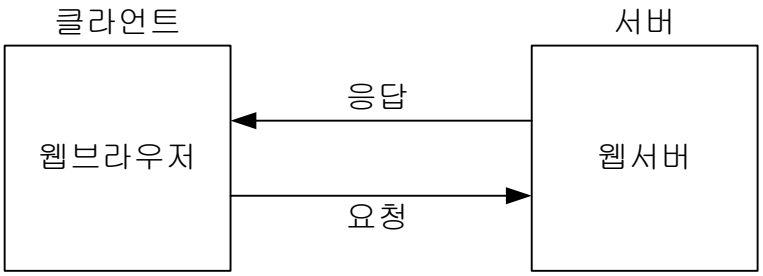
제일 마지막 줄에 “404” 를 확인할 수 있습니다.

[사물인터넷 그리고 클라언트와 웹서버]

사물인터넷 (IoT : Internet of Things)이라고 하는 것은 인터넷을 통하여 사물을 제어하거나 사물에서 보내 온 정보를 인터넷을 통하여 받아 보는 것입니다. 예를 들면 다음과 같이 먼 곳에 온실이 있고 IoT 기술을 이용하여 이 온실의 온도를 집에서 컨트롤하는 것입니다. 온실의 온도를 인터넷을 통하여 집에 있는 PC 로 볼 수 있습니다. 이 온도가 너무 낮으면 집에서 인터넷을 통하여 온실에 있는 보일러를 켜서 온도를 올릴 수 있습니다. 그러다가 온도가 적정 온도까지 올라가면 보일러를 끄면 되겠지요. 반대로 온도가 너무 높으면 환풍기를 돌려서 온도를 낮출 수도 있습니다. 이 구성은 다음 그림과 같습니다.

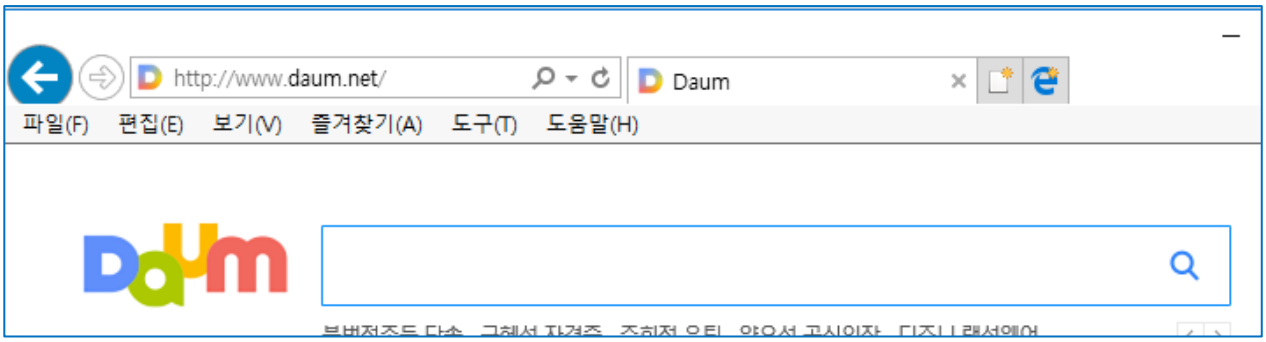


위의 그림에서 인터넷이라는 것은 여러분이 잘 알고 계시는 그 인터넷입니다. 이 인터넷이라는 것도 컴퓨터들이 연결되어 있는 네트워크입니다. 그 네트워크 중에서 전 세계의 모든 컴퓨터를 연결하려고 하는 가장 큰 네트워크입니다. 그림에서는 온실과 집의 컴퓨터를 인터넷 밖에 그렸지만 실제로는 이것들도 인터넷에 포함된 것입니다. 사물인터넷을 구현하기 위해서 우리는 이 인터넷에서 동작하는 웹(Web)을 다룰 것입니다. 이 웹은 기본적으로 서버와 클라이언트 구조로 구성되어 있으며 상호 데이터를 주고 받습니다.



클라이언트는 PC 에 설치되어 있는 웹브라우저라고 생각하면 됩니다. 웹서버는 클라이언트의 요청을 처리해 주는 서비스 시스템이라고 생각하시면 됩니다. 클라이언트와 서버 사이의 통신은 클라이언트에서 서버에 요청을 하면 서버에서 이 요청에 대한 응답을 해 주는 것으로 이루어집니다. 클라이언트의 요청에 대한 예는 다음과 같이 웹브라우저의 주소창에 특정 주소(http://www.daum.net)를 입력하는 것입니다.

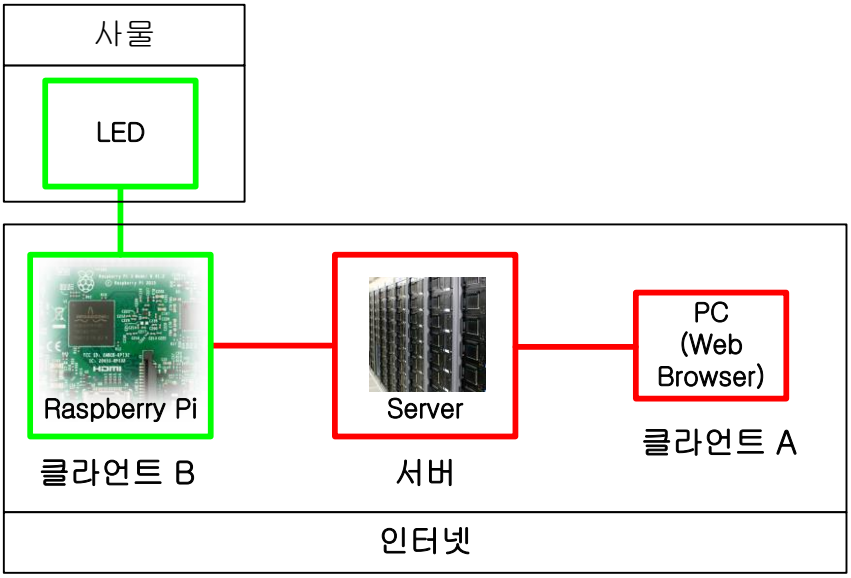
이렇게 특정 주소를 입력하는 하는 요청은 그 주소의 서버에게 해당 주소의 홈페이지 소스를 전달해 달라는 요청입니다. 이 요청을 받은 서버의 웹서버는 홈페이지 소스를 전달해 주는 것으로 응답을 합니다. 클라이언트의 웹브라우저에서는 서버에서 보낸 응답 소스를 다음 그림과 같이 웹프라우저를 통하여 보여줍니다.



IoT 를 웹에서 구현하기 위해서는 이 클라이언트와 서버의 개념을 잘 이해하셔야 합니다. 이제 우리는 이 클라이언트와 서버의 개념을 활용하여 IoT 를 구현해 볼 것입니다. 우리가 구현하려고 하는 IoT 의 구성은 하나의 서버에 두개의 클라이언트가 연결되어 있습니다. 클라이언트 중 하나는 PC 이고 다른 하나는 라즈베리파이입니다.

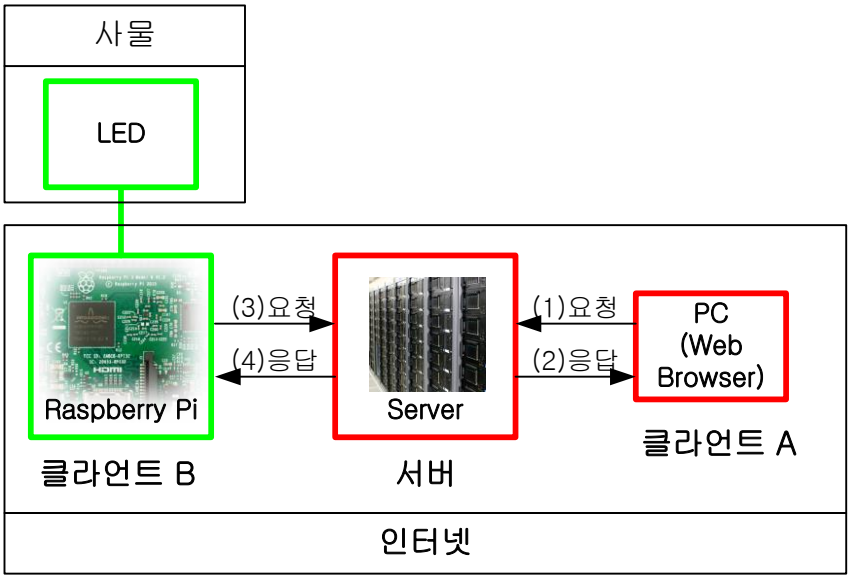
라즈베리파이에 사물이 연결될 것이고 PC 에서 이 사물들을 제어하거나 사물의 상태를 파악할 것입니다. 라즈베리파이는 코딩키트에 장착되어 여러가지 사물들과 연결될 것입니다. 더 나아가 아두이노와도 연결되어 더욱 더 다양한 IoT 예제를 다루어 볼 것입니다.

첫번째로 해 볼 매우 간단한 IoT 예제는 PC 에서 코딩키트의 LED 를 켜고 끄는 것입니다. 이것을 IoT 개념으로 생각하면 다음 그림과 같이 LED 가 사물이 되고 나머지 서버, 라즈베리파이, PC 는 인터넷이 되는 것입니다.



인터넷은 모든 컴퓨터가 연결된 커다란 네트워크라고 했습니다. PC 와 라즈베리파이 모두 인터넷에 연결된 컴퓨터이기 때문에 이것들까지 포함하여 인터넷인 것이고 여기에 연결된 사물이 LED 인 것입니다.

이것을 클라이언트의 요청을 서버에서 응답하는 관계로 보여주면 다음 그림과 같습니다.



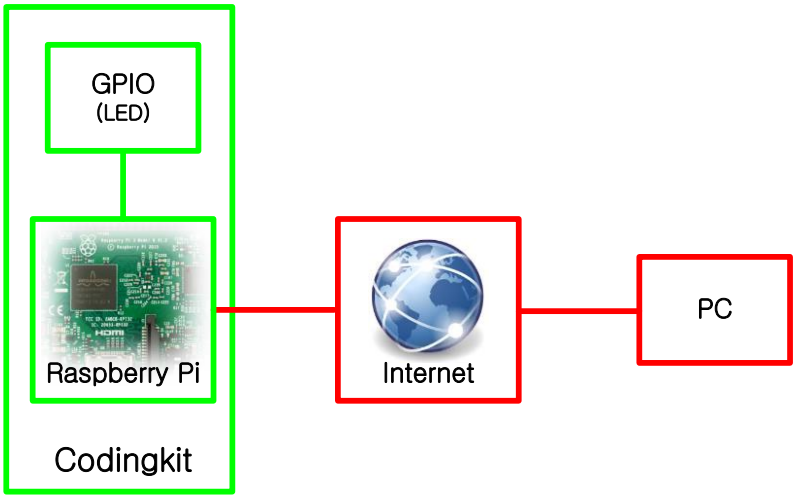
클라이언트 A 인 PC 의 웹프라우저에서는 서버에 접속하여 LED 의 상태를 켜짐으로 변경하라는 요청(1)을

합니다. 이 요청을 받은 서버는 LED 의 상태를 켜짐으로 변경하고 이 상태를 저장합니다. 그리고 이렇게 LED 의 상태를 변경하고 저장하였다고 응답(2)을 클라이언트 A 에 보내줍니다. 클라이언트 B 인 라즈베리파이는 서버에 LED 의 상태를 보내 줄 것을 요청(3)합니다. 이것에 대한 응답(4)으로 서버에서는 저장하고 있던 LED 의 상태를 보내줍니다. 클라이언트 A 에서는 LED 의 상태를 변경하고 싶을 때만 서버에 접속하여 LED 상태 변경 요청을 합니다. 클라이언트 B 에서는 LED 의 상태가 언제 변경될지 알지 못하므로 주기적으로 서버에 접속하여 LED 의 상태를 체크합니다.

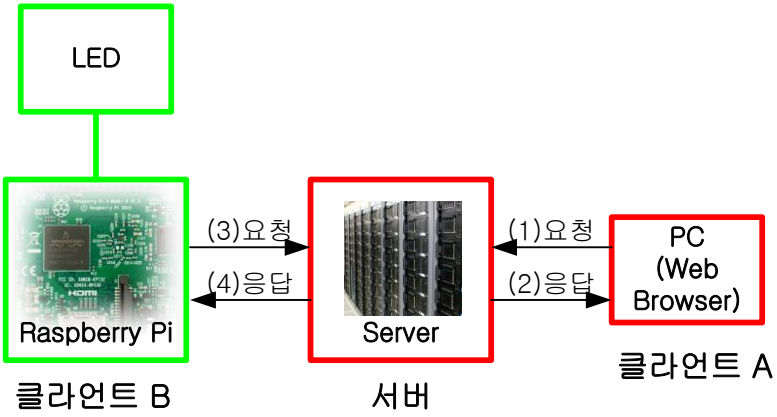
이 때 LED 의 상태를 저장하는 서버의 장치를 데이터베이스(Database : DB)라고 합니다. 이 DB 는 어떤 정보를 표처럼 만들어 저장합니다.

[인터넷을 통하여 코딩키트 LED 깜박이기]

이제 앞에서 설명한 것과 같이 PC 에서 인터넷을 통하여 라즈베리파이에 연결된 LED 를 켜고 끄는 코딩을 해 볼 것입니다.



위의 그림을 다음과 같이 클라이언트와 서버, 요청과 응답의 관계로 다시 그릴 수 있습니다.



위 그림의 구성 요소 중 클라이언트 들은 현재 여러분이 가지고 계신 PC 와 라즈베리파이가 장착된 코딩키트를 활용하시면 됩니다. 그리고 서버의 주소는 다음과 같습니다. 앞의 “Hello World” 예제에서 사용했던 서버입니다.

http://www.ckiot.net

이 서버는 코딩키트에서 미리 준비해 둔 서버입니다. 서버 설정 관련 내용은 코딩 사이트의 관련 게시물을 참고해 주세요.

클라이언트 A 인 PC 에서는 브라우저에 서버 주소(http://www.ckiot.net)을 입력하고 서버에 연결합니다. 그러면 다음과 같은 화면을 볼 수 있습니다.

서버에 접속하여 코딩키트의 LED 를 컨트롤하려면 먼저 로그인 하여야 합니다. 누구나 자신의 코딩키트에 접속하여 제어를 하면 안 되겠죠. 보안을 위해서 로그인 합니다. 로그인 ID 는 “ck0”, “ck1”, “ck2”, “ck3”, …, “ck254”, “ck255”까지 256개의 ID가 있습니다. 이 중 ID 하나를 선택하여 입력하면 됩니다. 암호는 라즈베리파이에서 “ckiot_get_pw.py” 파일을 다음과 같이 실행하면 됩니다. 이 파일은 “ckiot” 디렉토리 아래에 있습니다.

```
pi@raspberrypi:~/ckiot $ sudo python ckiot_get_pw.py
```

이 파일을 실행시키면 오늘 날짜가 맞는지 확인하는 질문을 합니다. 맞으면 “y” 를 입력하고 틀리면 “n” 을 입력합니다. “y” 를 입력하고 나면 ID Number 를 물어 봅니다. 이 때는 0 ~ 255 중 원하는 ID 숫자를 입력합니다.

```
pi@raspberrypi:~/ckiot $ sudo python ckiot_get_pw.py
Today is 20170712 , correct?
Enter "y" or "n" y
Input ID Number (0 ~ 255)55
ID Number : 55
****************************************
*  ID : ck55                           *
*  Password : 31809                    *
****************************************
```

마지막에 ID 와 암호(Password)를 알려 줍니다. 이 ID 와 암호를 이용하여 로그인 합니다. 물어보는 날짜가 틀리면 “n” 을 입력하고 맞는 날짜를 다음과 같이 입력해 줍니다.

```
pi@raspberrypi:~/ckiot $ sudo python ckiot_get_pw.py
Today is 20170712 , correct?
Enter "y" or "n" n
Input Date (ex : 20170101)20170709
Date : 20170709
Input ID Number (0 ~ 255)55
ID Number : 55
****************************************
*  ID : ck55                           *
*  Password : 16791                    *
****************************************
```

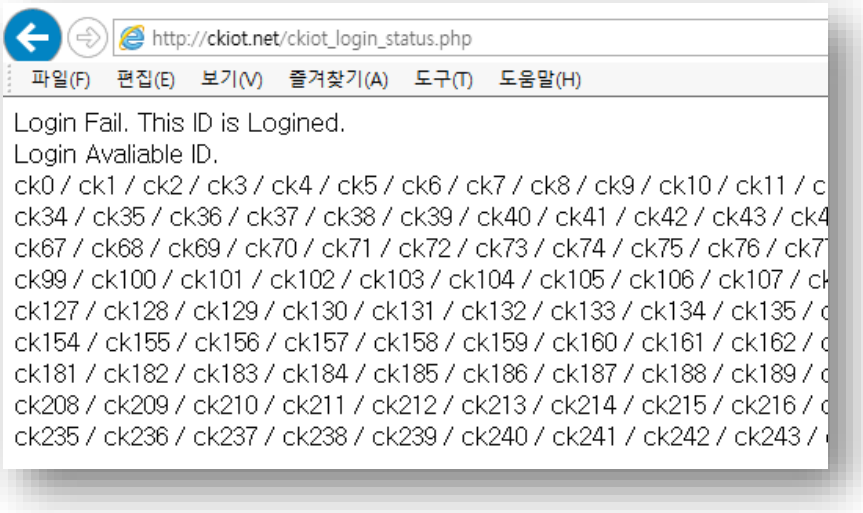
새로 입력된 날짜를 이용하여 ID 와 암호를 알려줍니다.

이렇에 생성되는 암호는 주기적(대략 5일정도)으로 바뀌는 것이기 때문에 로그인 하실 때 암호가 맞지 않다고 하면 다시 받아서 사용해야 합니다.

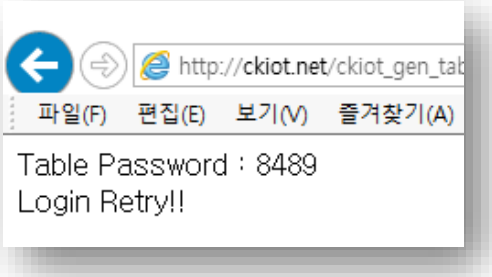
로그인 항목 중 “Table Password” 라는 것이 있는데, 이것은 서버의 데이터베이스에 코딩키트의 상태를 저장하기 위한 표(Table)를 만들어 두는데, 다른 사람이 접근하지 못 하도록 암호를 만들어 두는 것입니다. 암호는 서버에서 자동으로 만들어 집니다. 처음으로 로그인하는 것이어서 아직 데이터베이스에 표가 생성되어 있지 않다면 “Table Password” 는 “0” 으로 합니다.

이 ID와 암호를 이용하여 다음과 같이 로그인 합니다. 표 암호(Table Password)는 0 으로 하여 데이터베이스에 표를 생성합니다.

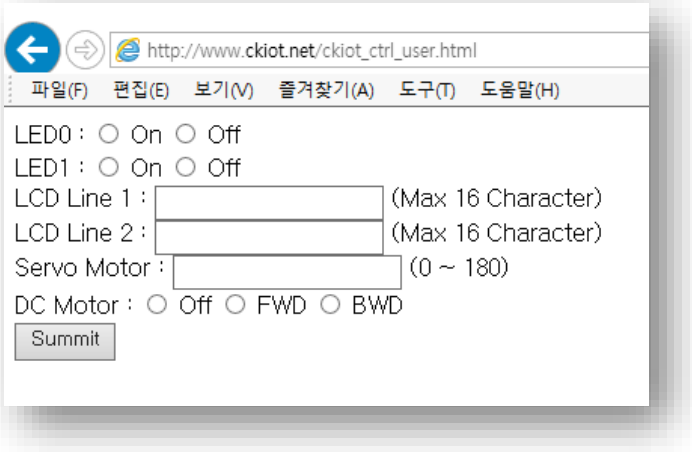
이 ID 를 누군가가 이미 사용하고 있다면 다음과 같은 메시지가 보입니다.



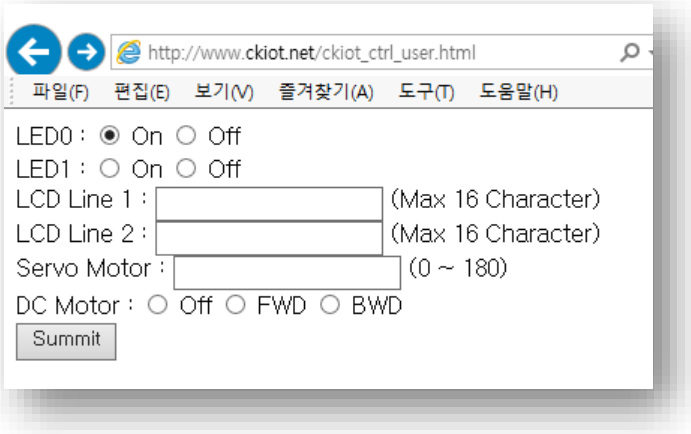
누군가가 이 ID 를 사용하고 있어서 로그인에 실패했다는 메시지를 보여 주면서 현재 사용 가능한 ID 의 목록을 보여 줍니다. 이 목록 중에 있는 다른 ID 를 사용하여 다시 로그인 합니다. 다른 ID 를 사용하여 로그인에 성공하였다면 다음과 같이 Table Password 를 알려주면서 다시 로그인 하라고 합니다.



이 Table Password 를 이용하여 다시 로그인 합니다. 그러면 다음과 같이 코딩키트에서 제어할 수 있는 디바이스들의 목록이 보입니다.



여기서 LED 를 켜고 싶으면 다음과 같이 LED 항목의 On 에 체크를 합니다. 그리고 “Summit” 버튼을 누릅니다.



우리가 구현하려고 하는 IoT 의 구성은 하나의 서버에 두개의 클라이언트가 연결되어 있습니다. 하나는 PC 이고 다른 하나는 라즈베리파이가 장착된 코딩키트입니다. PC 에서 코딩키트의 LED 를 켜거나 끄는 상태를 서버에 저장합니다. 그러면 코딩키트에서는 서버에 접속하여 현재 LED 의 상태를 불러와서 그 상태대로 LED 를 켜거나 끄면 되는 것입니다.

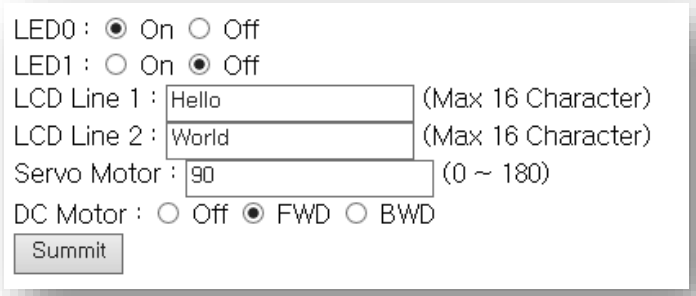
서버에는 데이터베이스 (Database : DB)라는 것이 있어서 어떤 정보를 표로 만들어 저장합니다. 코딩키트에 있는 디바이스들도 표를 만들어 저장합니다. DB 의 코딩키트 디바이스 표는 다음과 같이 구성되어 있습니다.

device	value
led0	0
led1	0
but0	1
but1	1
lcd_line1	
lcd_line2	
servo	0
dcmot	0
sensor_light	0
sensor_temp	0
sensor_sound	0
sensor_ir	0
var_res	0

위의 표에서 device 항목은 코딩키트의 각각의 디바이스를 나타내고, value 항목은 각 디바이스의 값을 저장합니다. 표에 대한 자세한 설명은 다음과 같습니다.

device	value	설명
led0	0, 1	LED 0 : On = 0, Off = 1
led1	0, 1	LED 1 : On = 0, Off = 1
but0	0, 1	Button 0 : 눌림 : 0, 안 눌림 : 1
but1	0, 1	Button 1 : 눌림 : 0, 안 눌림 : 1
lcd_line1	문자열	캐릭터 LCD 첫번째 라인, 최대 16 글자
lcd_line2	문자열	캐릭터 LCD 두번째 라인, 최대 16 글자
servo	0 ~ 180	서보모터 각도
dcmot	0, 1, 2	DC 모터 : 멈춤 = 0, 정방향 회전 = 1, 역방향 회전 = 2
sensor_light	0 ~ 1023	밝기 센서
sensor_temp	0 ~ 1023	온도 센서
sensor_sound	0 ~ 1023	소리 센서
sensor_ir	0 ~ 1023	적외선 센서
var_res	0 ~ 1023	가변 저항

PC 의 웹브라우저에서 다음과 같이 설정하고 Summit 버튼을 누르면 웹서버의 DB 의 표는 다음과 같이 변경되어 저장됩니다.

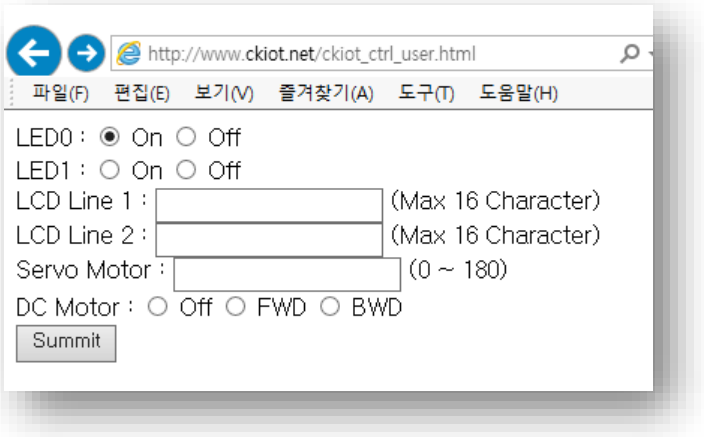


device	value
led0	1
led1	0
but0	1
but1	1
lcd_line1	Hello
lcd_line2	World
servo	90
dcmot	1
sensor_light	0
sensor_temp	0
sensor_sound	0
sensor_ir	0
var_res	0

이것을 PC 의 웹브라우저에서 다음과 같이 표시해 줍니다.

PC 웹브라우저 클라이언트에서 설정할 수 있는 디바이스는 LED0, .. , DC Motor 같이 PC 에서 코딩키트의 디바이스를 제어할 수 있는 것들입니다. 즉, 데이터의 전달이 PC 웹브라우저 → 인터넷 → 코딩키트로 되는 디바이스들만 PC 의 웹브라우저에서 제어합니다. 그래서 코딩키트 → 인터넷 → PC 로 데이터가 전달되는 디바이스는 사용자가 코딩키트를 제어하고 이것을 라즈베리파이가 전달합니다.

다시 예제로 돌아와서 다음과 같이 LED0 만 On 으로 설정하고 Summit 버튼을 누릅니다.



그러면 다음과 같이 현재 상태를 표시해 줍니다.

이제 라즈베리파이에서 웹서버의 DB 에 저장된 표를 읽어 오는 코딩을 해 보겠습니다. 코드는 다음과 같습니다.

```
import cookielib, urllib2, urllib

# URL Open
cj = cookielib.CookieJar()
opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(cj))
urllib2.install_opener(opener)

# Login CK IoT Server
user_id = "ck55"
password = "5224"
tbl_pw = “9113”
```

```
params = urllib.urlencode({"id":user_id, "pw":password, “tbl_pw”:tbl_pw})
req = urllib2.Request("http://www.ckiot.net/ckiot_login_device.php", params)
res = opener.open(req)

read_msg = res.read()
print read_msg

login_ok = int(read_msg[0])

if (login_ok == 0): #Login Fail
   print read_msg[1:]
else:
   print "Login OK"
   req = urllib2.Request("http://www.ckiot.net/ckiot_table.php")
   res = opener.open(req)
   read_msg = res.read()
   read_msg = read_msg[1:]
   print read_msg
```

먼저 cookielib, urllib2, urllib 라이브러리를 Import 합니다. 이 라이브러리들을 통해서 웹서버에 접속하고 웹서버에 데이터를 요청합니다. cookielib 라이브러리는 웹서버의 쿠키(Cookie) 데이터를 요청하여 받아오는데 사용됩니다. 쿠키라는 것은 웹서버에 기록되어 있는 작은 데이터들을 말하는 것입니다. urllib 라이브러리는 앞에서 설명 드렸고, urllib2 는 urllib 와 비슷한 기능을 가지면서 조금 더 개선된 것이라고 생각하시면 됩니다.

```
# URL Open
cj = cookielib.CookieJar()
opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(cj))
urllib2.install_opener(opener)
```

위와 같이 하여 웹서버에 접속할 준비를 합니다. PC 클라이언트에서 했던 것처럼 다음과 같이 웹서버에 로그인 합니다.

```
# Login CK IoT Server
user_id = "ck55"
password = "5224"
tbl_pw = “9113”

params = urllib.urlencode({"id":user_id, "pw":password, “tbl_pw”:tbl_pw })
req = urllib2.Request("http://www.ckiot.net/ckiot_login_device.php", params)
res = opener.open(req)
```

PC 클라이언트와 마찬가지로 ID 는 ck1, ck2, …, ck255 까지이며 이 중 하나를 선택하여 사용하면 됩니다. 암호(password)는 ckiot_get_pw.py 파일을 실행시켜 구해오면 됩니다. Table Password(tbl_pw)는 PC 의 웹브라우저에서 로그인 할 때 Table Password 값에 0 을 입력해 구해옵니다. user_id, password, tbl_pw 변수를 ulrlib.urlencode() 함수를 이용하여 팩킹해둡니다. 이렇게 팩킹된 값을 urllib2.Request() 함수를 이용하여 웹서버로 보낼 준비를 합니다. 웹서버로 보내기 위해서 웹서버의 파일 주소도 같이 적어줍니다. 웹서버에서 이 로그인을 처리해 줄 파일은 ckiot_login_device.php 라는 파일입니다. 이 파일은 PHP 언어로 코딩된 파일입니다. PHP 언어는 현재 웹서버 코딩 언어로 가장 많이 사용되는 컴퓨터 언어입니다. 이제 로그인 요청을 opener.open() 함수를 이용하여 웹서버에 보냅니다. 이 요청에 대해서 웹서버가 응답을 보내옵니다. 이 응답은 res(response : 응답) 변수에 저장이 됩니다.

웹서버의 응답 내용을 다음과 같은 코드를 이용하여 읽어들이고 출력해 볼 수 있습니다.

```
read_msg = res.read()
print read_msg
```

read_msg 의 내용은 로그인 ID 와 암호가 맞으면 1 이고 맞지 않으면 0 입니다. 그래서, 다음과 같은 코드를 이용하여 로그인이 잘 되었는지 확인할 수 있습니다.

```
login_ok = int(read_msg[0])

if (login_ok == 0): #Login Fail
   print read_msg[1:]
else:
   print "Login OK"
```

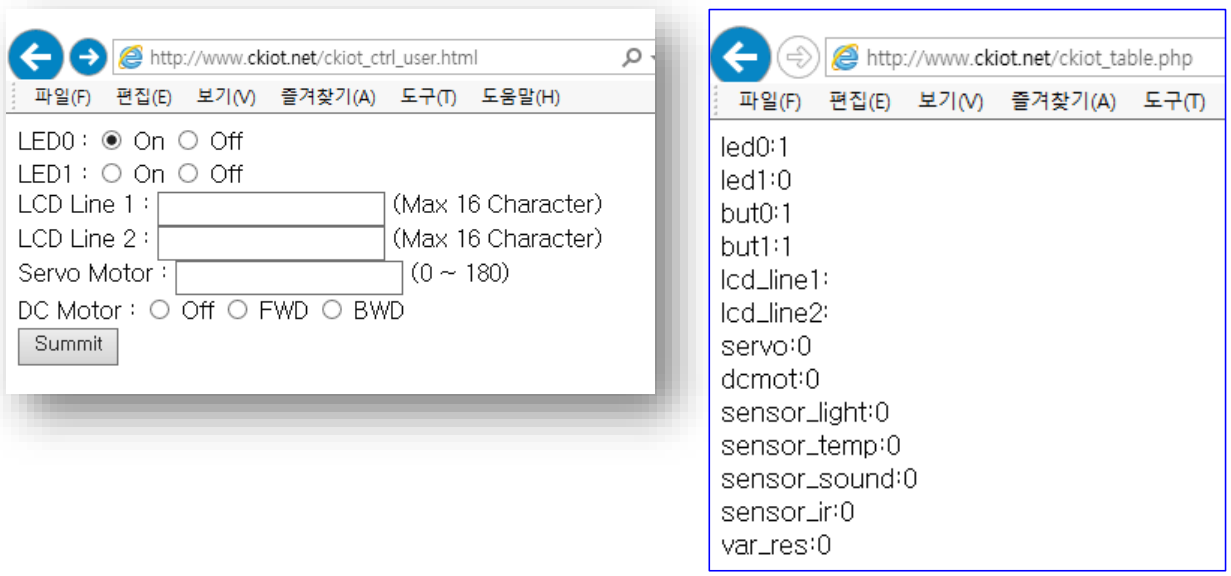
로그인 실패(Login Fail)일 때 read_msg 를 더 출력하는 이유는 read_msg 는 리스트인데 이 중 첫번째는 0 또는 1 로서 로그인에 대한 성공 실패를 나타내고, 그 다음 문자부터는 만약 로그인에 실패했을 때 원인을 보여주는 에러 메세지입니다.
로그인에 성공하였다면 다음과 같이 두 번째 요청을 합니다. 이 요청은 웹서버의 DB 에서 코딩키트 디바이스 관련 표를 보내달라는 요청입니다.

```
req = urllib2.Request("http://www.ckiot.net/ckiot_table.php")
res = opener.open(req)
read_msg = res.read()
read_msg = read_msg[1:]
print read_msg
```

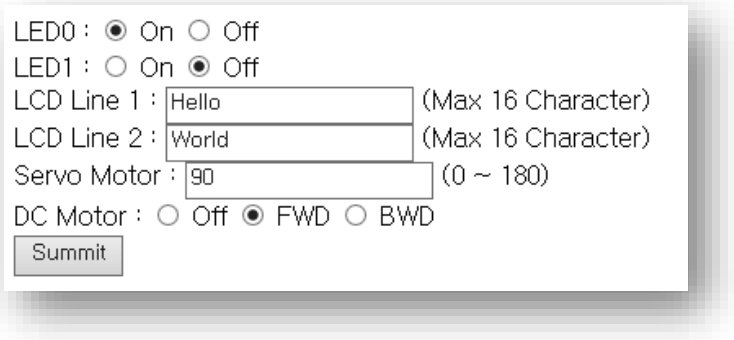
이 요청은 웹서버의 ckiot_table.php 파일에서 처리합니다. 이 파일은 웹서버에서 각 디바이스의 상태가 저장되어 있는 표를 다루는 파일입니다. print read_msg 의 출력 값은 다음과 같습니다.

```
pi@raspberrypi:~/ckiot $ sudo python e00_ckiot_dev_db.py
1
Login OK
led0:1.led1:0.lcd_line1:.lcd_line2:.servo:0.dcmot:0.but0:1.but1:1.sensor_light:0
.sensor_temp:0.sensor_sound:0.sensor_ir:0.var_res:0
```

출력이 잘 정리되어 보이지는 않지만, 이것은 다음과 같이 PC 클라언트에서 설정한 값들에 대한 출력입니다.



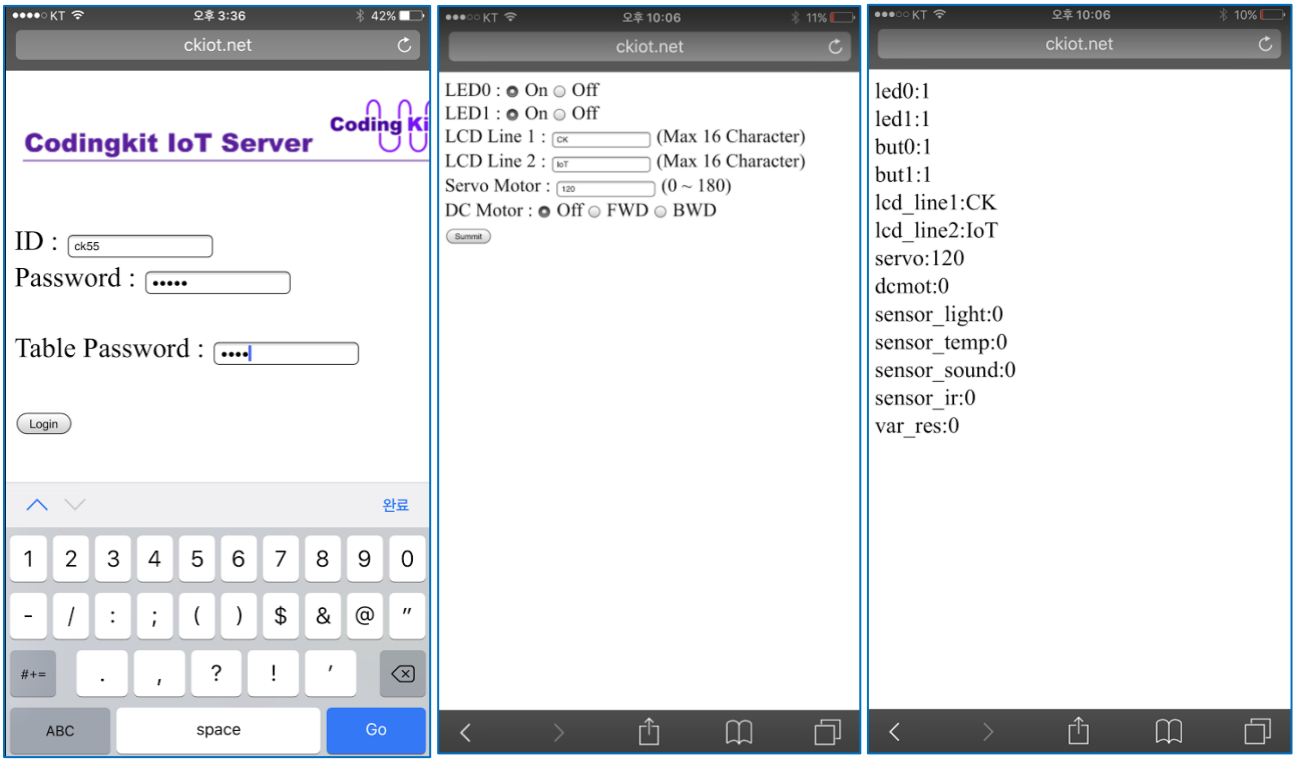
PC 에서 다음과 같이 설정을 바꾸어 보도록 하겠습니다.



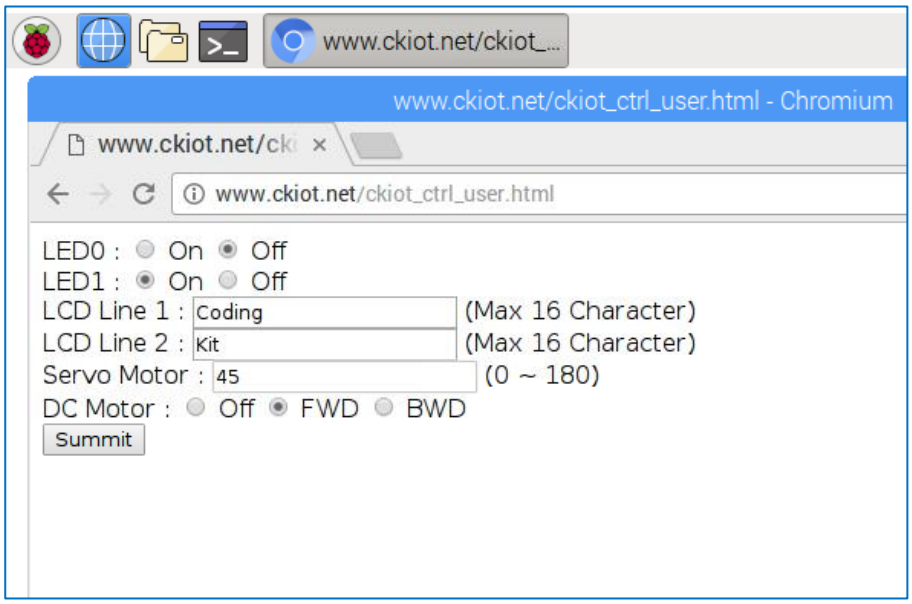
그러면 라즈베리파에서 파이썬 코드는 다음과 같이 출력이 됩니다.

```
pi@raspberrypi:~/ckiot $ sudo python e00_ckiot_dev_db.py
1
Login OK
led0:1.led1:0.lcd_line1:.lcd_line2:.servo:0.dcmot:0.but0:1.but1:1.sensor_light:0
.sensor_temp:0.sensor_sound:0.sensor_ir:0.var_res:0
```

앞에서 이야기한 PC 클라이언트라고 한 것을 휴대폰으로 바꾸어 테스트해 볼 수도 있습니다.



라즈베리파이의 웹브라우저에서도 가능합니다.



< 예제 코드 : 웹서버 디바이스 DB 내용 불러오기 >

< 라즈베리파이 코드 : e00_ckiot_dev_db**.py** >

파이썬 코드의 출력이 보기가 좋지 않습니다. 그래서 정리하는 코드를 작성해 보도록 하겠습니다.

```
if (login_ok == 0): #Login Fail
   print read_msg[1:]
else:
   print "Login OK"
   req = urllib2.Request("http://www.ckiot.net/ckiot_table.php")
   res = opener.open(req)
   read_msg = res.read()
   read_msg = read_msg[1:]
   #print read_msg
   dev_list = read_msg.split('.')
   for dev_name_val in dev_list:
      dev_name_val_list = dev_name_val.split(':')
      dev_name = dev_name_val_list[0]
      dev_val = dev_name_val_list[1]
      print dev_name, " : ", dev_val
```

위의 코드의 앞 부분은 이전 코드와 같습니다.

다음과 같은 이전 코드의 출력을 보면 디바이스별로 점(.)으로 구분된 것을 볼 수 있을 것입니다.

```
pi@raspberrypi:~/ckiot $ sudo python ckiot_dev_prt.py
res :  1
Login OK
led0:1.led1:0.but0:1.but1:1.lcd_line1:Hello.lcd_line2:World.servo:90.dcmot:1.sen
sor_light:0.sensor_temp:0.sensor_sound:0.sensor_ir:0.var_res:0
```

그래서 read_msg.split(.) 함수를 이용하여 점(.) 으로 구분하여 리스트에 저장합니다. 이렇게 구분된 리스트를 출력해 보면 다음과 같습니다.

```
['led0:0', 'led1:0', 'but0:1', 'but1:1', 'lcd_line1:', 'lcd_line2:', 'servo:0', 'dcmot:0',
'sensor_light:0', 'sensor_temp:0', 'sensor_sound:0', 'sensor_ir:0', 'var_res:0']
```

for 문에서 이 리스트의 항목을 개별적으로 나누고 각각의 항목에서 디바이스의 이름(dev_name)과 디바이스의 값(dev_val)으로 구분합니다. 이 때, split() 함수를 이용하여 콜론(:)으로 구분합니다. 이 코드를 실행시켜 출력을 보면 다음과 같이 각각의 디바이스별로 해당 값이 출력되는 것을 볼 수 있습니다.

```
led0   :  0
led1   :  0
but0   :  1
but1   :  1
lcd_line1   :
lcd_line2   :
servo   :  0
dcmot   :  0
sensor_light   :  0
sensor_temp   :  0
sensor_sound   :  0
sensor_ir   :  0
var_res   :  0
```

< 예제 코드 : 웹서버 디바이스 DB 내용 불러오기와 출력하기 >

< 라즈베리파이 코드 : e00_ckiot_dev_db_prt.py >

이제 이렇게 불러온 DB 의 내용에 따라서 실제 디바이스를 제어하는 코딩을 해 보겠습니다. 먼저 간단하게 LED 를 켜고 끄는 코딩을 해 보겠습니다. 간단히 요약하면 이전의 코드에 LED 에 해당하는 GPIO 를 추가하고 읽어드린 DB 의 내용 중 LED 부분을 찾아 그 값에 맞게 LED 에 해당하는 GPIO 컨트롤을 해 주면 됩니다. 코드는 다음과 같습니다.

```
import cookielib, urllib2, urllib
import RPi.GPIO as GPIO

# Set GPIO
GPIO.setmode(GPIO.BCM)
LED2 = 24
GPIO.setup(LED2, GPIO.OUT)

# URL Open
cj = cookielib.CookieJar()
opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(cj))
urllib2.install_opener(opener)

# Login CK IoT Server
user_id = "ck55"
password = "5224"
tbl_pw = “9113”

params = urllib.urlencode({"id":user_id, "pw":password, “tbl_pw”:tbl_pw })
req = urllib2.Request("http://www.ckiot.net/ckiot_login_device.php", params)
res = opener.open(req)

read_msg = res.read()
print read_msg

login_ok = int(read_msg[0])

if (login_ok == 0): #Login Fail
   print read_msg[1:]
else:
   print "Login OK"
   req = urllib2.Request("http://www.ckiot.net/ckiot_table.php")
   res = opener.open(req)
   read_msg = res.read()
   read_msg = read_msg[1:]
   #print read_msg
   dev_list = read_msg.split('.')
   for dev_name_val in dev_list:
      dev_name_val_list = dev_name_val.split(':')
      dev_name = dev_name_val_list[0]
      dev_val = dev_name_val_list[1]
      print dev_name, " : ", dev_val
      if (dev_name == "led0"):
         if (dev_val == '1'):
            GPIO.output(LED2, 1)
```

```
else:
    GPIO.output(LED2, 0)
```

붉은색 부분이 새롭게 추가된 부분입니다. 먼저 GPIO 라이브러리를 import 합니다. 그리고 LED 에 사용될 GPIO 를 설정합니다. 이 코드에서는 두번째 LED(LED2)를 사용할 것입니다. 이유는 코딩키트에서 라즈베리파이에는 두번째와 세번째 LED(LED2, LED3)가 연결되어 있습니다. 영번째(제일 오른쪽 LED)와 첫번째 LED(LED0, LED1)는 아두이노에 연결되어 있습니다.

코드의 제일 아래쪽 if 문에서 dev_name 변수가 “led0” 인 것을 검색합니다. 이 항목이 LED 항목입니다. “led0” 의 값이 ‘1’ 이면 LED 를 켜고 값이 ‘0’ 이면 LED 를 끕니다. 다음과 같이 PC 클라이언트에서 LED 를 켜는 것으로 설정하고 이 코드를 실행시켜 LED 가 켜지는 것을 확인합니다.

그런데 이렇게 하면 PC 클라이언트에서 LED 를 켜고 끄는 설정을 바꾼 이후에 라즈베리파이 클라이언트에서 파이썬 코드를 실행해야 합니다. 그래서 라즈베리파이 코드를 다음과 같이 무한 반복하게 만들어 PC 클라이언트에서의 LED 제어가 바로 바로 즉각적으로 반영되도록 합니다.

```
import time

(중간 생략)

count = 0
if (login_ok == 0): #Login Fail
    print read_msg[1:]
else:
    print "Login OK"
    req = urllib2.Request("http://www.ckiot.net/ckiot_table.php")
    while 1:
        res = opener.open(req)
        read_msg = res.read()
        read_msg = read_msg[1:]
        dev_list = read_msg.split('.')
```

```
for dev_name_val in dev_list:
    dev_name_val_list = dev_name_val.split(':')
    dev_name = dev_name_val_list[0]
    dev_val = dev_name_val_list[1]
    #print dev_name, " : ", dev_val
    if (dev_name == "led0"):
        if (dev_val == '1'):
            GPIO.output(LED2, 1)
        else:
            GPIO.output(LED2, 0)
count = count + 1
print count
time.sleep(1)
```

붉은색 부분이 새롭게 추가된 부분입니다. 중간에 while 문을 사용하여 무한 반복을 구현하였습니다. 반복 사이에 1 초의 시간 지연을 주었습니다. 이 시간 지연을 위해 시간(time) 라이브러리를 추가하였습니다. count 변수를 증가하고 이 변수를 출력하므로서 얼마나 반복하고 있는지를 알 수 있습니다. 이제 파이썬 코드를 실행시키고, PC 클라언트에서 LED 를 켜고 끄고를 테스트해 봅니다. 코딩키트의 LED 가 잘 켜지고 잘 꺼지나요? LED 가 약간의 시간 차이를 가지고 반응합니다. 이것은 time.sleep(1) 코드에 의한 1 초 지연 때문입니다. 스마트폰등을 이용하여 제어해 볼 수도 있습니다.

IoT(사물이터넷)를 실제 구현해 보니 재미있지요? IoT 에 대해서 많이 이야기들을 하지만 실제로 잘 정리되어 구현하는 예는 찾기 힘듭니다. IoT 를 구현하는데에는 여러가지 IT 기술들이 종합적으로 포함되어 있어 쉽게 정리하기가 어렵기 때문입니다. 여러분은 IoT 기술 설명과 예제가 잘 정리되어 있는 코딩키트를 활용하여 IoT 기술을 충분히 습득하기를 바랍니다. 이제 앞으로 더 재미있는 IoT 예제를 해 볼 것입니다. 많은 기대 바랍니다.

< 예제 코드 : 인터넷을 통하여 LED 깜박이기 >

< 라즈베리파이 코드 : e00_ckiot_led.py >

< 라즈베리파이 코드 : e00_ckiot_led_loop.py >

[버튼이 눌린 것을 웹서버 DB 에 저장하기]

이전 장에서는 웹서버의 DB 에 저장되어 있는 LED 상태를 읽어와서 코딩키트의 LED 를 켜거나 끄는 예제를 해 보았습니다. 이번에는 반대로 코딩키트의 버튼 상태를 웹서버의 DB 에 저장하는 것입니다.

코드는 이전 LED 예제와 비슷합니다.

```
import cookielib, urllib2, urllib
import RPi.GPIO as GPIO
import time

# Set GPIO
GPIO.setmode(GPIO.BCM)
BUT2 = 6
GPIO.setup(BUT2, GPIO.IN)

# URL Open
cj = cookielib.CookieJar()
opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(cj))
urllib2.install_opener(opener)

# Login CK IoT Server
user_id = "ck55"
password = "5224"
tbl_pw = “9113”

params = urllib.urlencode({"id":user_id, "pw":password, “tbl_pw”:tbl_pw })
req = urllib2.Request("http://www.ckiot.net/ckiot_login_device.php", params)
res = opener.open(req)

read_msg = res.read()
print read_msg

login_ok = int(read_msg[0])

count = 0
if (login_ok == 0): #Login Fail
   print read_msg[1:]
else:
   print "Login OK"
   while 1:
      but0 = GPIO.input(BUT2)
      params = urllib.urlencode({"but0":but0})
      req = urllib2.Request("http://www.ckiot.net/ckiot_table.php", params)

      res = opener.open(req)
      read_msg = res.read()
      read_msg = read_msg[1:]
      dev_list = read_msg.split('.')
      for dev_name_val in dev_list:
         dev_name_val_list = dev_name_val.split(':')
         dev_name = dev_name_val_list[0]
         dev_val = dev_name_val_list[1]

         if (dev_name == "but0"):
            if (dev_val == '1'):
```

```
            print "but0 : 1"
        else:
            print "but0 : 0"

    count = count + 1
    print count
    time.sleep(1)
```

붉은색 코드 부분이 달라지는 부분입니다. 먼저 버튼에 해당하는 GPIO 설정을 합니다. while 문에서 버튼의 상태를 체크합니다. 버튼의 상태를 다음 코드를 통해서 웹서버로 보냅니다.

```
but0 = GPIO.input(BUT2)
params = urllib.urlencode({"but0":but0})
req = urllib2.Request("http://www.ckiot.net/ckiot_table.php", params)
```

이렇게 전송된 버튼의 값은 바로 웹서버의 DB에 저장이 됩니다. 웹서버 DB에 저장된 값을 읽어와 출력합니다. 출력은 다음과 같습니다.

```
pi@raspberrypi:~/ckiot $ sudo python e00_ckiot_but_loop.py
1
Login OK
but0 : 1
1
but0 : 1
2
but0 : 0
3
but0 : 0
4
but0 : 1
5
but0 : 1
6
but0 : 1
7
```

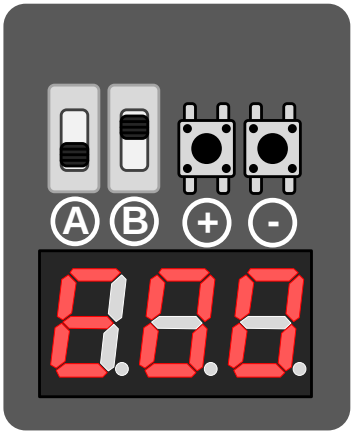
버튼(but0) 값이 처음에는 눌리지 않은 상태 1 을 보여줍니다. 버튼이 눌리면 0 으로 보여줍니다. 다시 버튼 누른 것을 떼면 1 로 출력됩니다.

< 예제 코드 : 버튼이 눌린 것을 웹서버 DB 에 저장하기 >

< 라즈베리파이 코드 : e00_ckiot_but_loop.py >

[인터넷을 통하여 아두이노에 연결된 DC 모터 돌리기]

현재 코딩키트의 스위칭 ID는 다음과 같이 E00 입니다.



E00 일 때 아두이노와 라즈베리파이의 전체 핀 연결 표는 다음과 같습니다. 각각의 디바이스는 아두이노(**Ard**) 혹은 라즈베리파이(**Raspi**)에 연결되어 있습니다. 붉은색 번호가 핀 번호입니다.

디바이스	Ard	Raspi
UART (Ard TX --> Raspi RX)	7	15
UART (Ard RX <-- Raspi TX)	8	14
LED 0	A4	
LED 1	A5	
LED 2		24
LED 3		25
Button 0	12	
Button 1	13	
Button 2		6
Button 3		16
Buzzer	5	
DC Motor Enable	6	
DC Motor Forward	10	
DC Motor Backward	11	
DC Motor Encoder Enable		20
DC Motor Encoder Data		√
Sound Sensor Detect		4
IR Sensor LED	4	
IR Sensor Detect		12

Servo Motor	9	
Dotmatrix		√
7-Segment		
Device Control SPI		*R1
Character LCD EN, RS, Data		√
Character LCD Back Light		√
Switch 0		26
Switch 1		27
Switch 2		2
Switch 3		3
Serial Monitor	0,1	
Ard --> Raspi Connect	3	23
Raspi --> Ard Connect	2	22
ADC SPI (Raspi Only)		*R2

위의 표에서 *R1 과 *R2 는 라즈베리파이의 SPI 연결로서 다음표와 같이 SPI 핀들이 연결되어 있습니다.

*R1 : SPI(Device)	*R2 : SPI (ADC)
SPI_SS : 7	SPI_SS : 8
SPI_MOSI : 10	SPI_MOSI : 10
SPI_MISO : 9	SPI_MISO : 9
SPI_SCLK : 11	SPI_SCLK : 11

전체 핀 연결 표를 보시면 DC 모터는 아두이노에 연결되어 있습니다. 그래서 인터넷을 통하여 DC 모터를 컨트롤하기 위해서는 다음과 같은 작업들이 필요합니다.

(1) PC 클라이언트의 브라우저에서 DC 모터를 다음과 같이 설정합니다.

(2) 이 값은 웹서버의 DB 에 저장됩니다.

(3) 라즈베리파이에서는 웹서버의 DB 에서 DC 모터의 설정을 읽어옵니다.

(4) 라즈베리파이는 웹서버에서 읽어 온 DC 모터의 설정을 UART 를 통하여 아두이노로 전달합니다.

(5) 아두이노에서는 라즈베리파이에서 받은 DC 모터의 설정과 같게 DC 모터를 동작시킵니다.

라즈베리파이에서 웹서버의 DB 에서 DC 모터 관련 내용을 읽어 오는 코드입니다.

```
import cookielib, urllib2, urllib
import RPi.GPIO as GPIO
import time
import serial

# Set Serial
port = serial.Serial("/dev/ttyAMA0", baudrate=19200, timeout=3.0)

# Set GPIO
GPIO.setmode(GPIO.BCM)
LED2 = 24
GPIO.setup(LED2, GPIO.OUT)

# URL Open
cj = cookielib.CookieJar()
opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(cj))
urllib2.install_opener(opener)

# Login CK IoT Server
user_id = "ck55"
password = "5224"
tbl_pw = “9113”

params = urllib.urlencode({"id":user_id, "pw":password, “tbl_pw”:tbl_pw })
req = urllib2.Request("http://www.ckiot.net/ckiot_login_device.php", params)
res = opener.open(req)

read_msg = res.read()
print read_msg

login_ok = int(read_msg[0])

count = 0
if (login_ok == 0): #Login Fail
   print read_msg[1:]
else:
   print "Login OK"
   req = urllib2.Request("http://www.ckiot.net/ckiot_table.php")
   while 1:
      res = opener.open(req)
      read_msg = res.read()
      read_msg = read_msg[1:]
      dev_list = read_msg.split('.')
      for dev_name_val in dev_list:
         dev_name_val_list = dev_name_val.split(':')
         dev_name = dev_name_val_list[0]
         dev_val = dev_name_val_list[1]
         print dev_name, " : ", dev_val
         # $dev_name:dev_val.
         if (dev_name == "dcmot"):
            port.write("$dcmot")
            if (dev_val == '0'):
               port.write(":Off.")
            elif (dev_val == '1'):
               port.write(":FWD.")
            elif (dev_val == '2'):
```

```
            port.write(":BWD.")
        count = count + 1
        print count
        time.sleep(1)
```

붉은색 부분이 새롭게 추가된 부분입니다. 먼저 아두이노와 시리얼 통신을 하기 위해서 serial 라이브러리를 Import 합니다. 다음으로 시리얼포트를 정의합니다. 라즈베리파이와 아두이노 사이에 디바이스 이름과 디바이스 값을 주고 받기 위해서 다음과 같은 약속을 합니다.

$dev_name:dev_val.

디바이스 이름앞에는 달러 표시($)를 붙이고 그 디바이스의 값에는 콜론(:)을 붙입니다. 디바이스 이름과 디바이스 값의 쌍이 끝나면 마지막에 점(.)을 씁니다. 예를 들어 DC 모터의 값이 정방향(FWD) 일 때는 다음과 같이 써 줍니다.

$dcmot:FWD.

위의 코드에서는 웹서버에서 DC 모터 관련 항목을 읽어와서 아두이노와 시리얼 통신으로 DC 모터 이름과 값을 전달합니다. 웹서버에서는 DC 모터가 정방향일 때는 1, 역방향일 때는 2, 멈춤 상태일 때는 0값을 전달해 줍니다. 라즈베리파이에서는 이 0, 1, 2 값을 "Off", "FWD", "BWD" 로 바꾸어 아두이노로 전달합니다.

이렇게 라즈베리파이에서 시리얼통신으로 전달해 준 디바이스 이름과 디바이스 값을 아두이노에서는 어떻게 받아서 어떻게 처리하는지를 공부해 보겠습니다. 다음과 같이 소프트웨어 시리얼(Sortware Serial) 라이브러리를 Include 하고 정의합니다.

```
#include <SoftwareSerial.h>

#define SW_SERIAL_TX 7
#define SW_SERIAL_RX 8

#define CK_SERIAL_BAUDRATE 19200

SoftwareSerial ckSerial(SW_SERIAL_RX, SW_SERIAL_TX);
```

라즈베리파이와 아두이노 사이에 데이터 전달은 다음과 같이 한다고 하는 것을 기억하고 계실 것입니다.

$dev_name:dev_val.

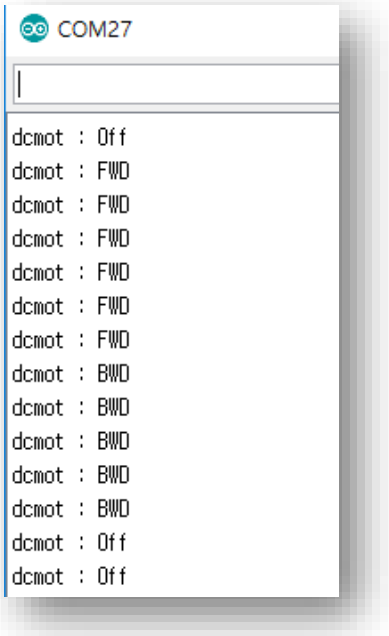
그래서 loop() 내에서 다음과 같이 '$', ':', '.' 을 검색합니다.

```
if (c == '$') {
   devNameEn = 1;
   devValEn = 0;
   rxCnt = 0;
  }
  else if (c == ':') {
   devNameEn = 0;
```

```
  devValEn = 1;
  rxCnt = 0;
 }
 else if (c == '.') {
  devNameEn = 0;
  devValEn = 0;
  rxCnt = 0;
  Serial.print(devName);
  Serial.print(" : ");
  Serial.println(devVal);
 }
 else {
  if (devNameEn)
    devName[rxCnt] = c;
  else if (devValEn)
    devVal[rxCnt] = c;
  rxCnt++;
 }
```

'$' 뒤에는 디바이스 이름이 입력되므로 devNameEn 변수를 1 로 해 줍니다. ':' 다음에는 디바이스 값이 입력되므로 devValEn 변수를 1 로 해 줍니다. '.' 는 입력의 끝을 의미하므로 모든 변수를 0 으로 초기화해 주고 입력된 디바이스 이름과 값을 출력합니다. if 문의 마지막 끝의 else 문에서는 지금 입력되는 값을 저장합니다. 여기서 저장할 때 디바이스 이름인지 디바이스 값인지는 devNameEn 변수와 devValEn 변수를 보고 알 수 있습니다. 문자가 하나씩 입력될 때마다 rxCnt 값을 1 씩 증가해 주면서 devName 과 devVal 배열에 저장합니다.

이제 모든 코드들을 실행해 볼 텐데요. 라즈베리파이의 코드를 먼저 실행합니다. 그리고 아두이노 코드를 실행합니다. 아두이노에서 시리얼 모니터를 열어 줍니다. 보드레이트는 57600 으로 합니다. PC 또는 스마트폰으로 ckiot 서버(http://www.ckiot.net)를 접속하고 로그인한 이후에 DC 모터를 정지(Off), 정방향(FWD), 역방향(BWD) 등을 선택해 봅니다. 그러면 다음과 같이 출력됩니다.



이번에는 실제 DC 모터를 돌려보겠습니다. 먼저 다음과 같이 DC 모터 핀 번호를 정의합니다.

```
#define DCMOT_EN        6
#define DCMOT_FWD       10
#define DCMOT_BWD       11
```

setup() 함수는 다음과 같습니다.

```
void setup() {
  pinMode(DCMOT_EN, OUTPUT);
  pinMode(DCMOT_FWD, OUTPUT);
  pinMode(DCMOT_BWD, OUTPUT);
  digitalWrite(DCMOT_EN, 1);

  ckSerial.begin(CK_SERIAL_BAUDRATE);
  Serial.begin(57600);
  rxCnt = 0;
}
```

다음과 같이 데이터 전달의 마침을 뜻하는 '.' 이 입력된 이후에 디바이스 값을 뜻하는 devVal 변수를 체크하여 모터를 돌려줍니다.

```
else if (c == '.') {
   devNameEn = 0;
   devValEn = 0;
   rxCnt = 0;
   Serial.print(devName);
   Serial.print(" : ");
   Serial.println(devVal);
   if (!strcmp(devVal, "Off")) {
      digitalWrite(DCMOT_FWD, 0);
      digitalWrite(DCMOT_BWD, 0);
   }
   else if (!strcmp(devVal, "FWD")) {
      digitalWrite(DCMOT_FWD, 1);
      digitalWrite(DCMOT_BWD, 0);
   }
   else if (!strcmp(devVal, "BWD")) {
      digitalWrite(DCMOT_FWD, 0);
      digitalWrite(DCMOT_BWD, 1);
   }
}
```

여기서 strcmp() 라는 함수를 알아보겠습니다. 이 함수는 문자열끼리 비교하는 것입니다. strcmp(devVal, "Off") 는 devVal 에 저장된 문자열과 "Off" 문자열을 비교하는 것입니다. 그래서 두 문자열이 같으면 0 을 반환해 줍니다. 1 이 아닌 0 을 반환해 주기 때문에 strcmp() 함수 앞에 '!' 연산자를 붙입니다. 이제 코드를 실행해 봅니다. 그리고 PC 또는 스마트폰에서 DC 모터를 컨트롤 합니다.

< 예제 코드 : 인터넷을 통하여 아두이노에 연결된 DC 모터 돌리기 >

< 라즈베리파이 코드 : e00_ckiot_dcmot_serial_tx.py >

< 아두이노 코드 : e00_ckiot_serial_rx.ino >

< 아두이노 코드 : e00_ckiot_ serial_rx_dcmot.ino >

[가변 저항값 웹서버에 전달하기]

아두이노에 연결된 가변저항의 값을 웹서버로 전송하는 코딩을 해 보겠습니다. 순서는 다음과 같습니다.

(1) 아두이노에서 가변저항 값을 읽습니다.

(2) 이 값을 시리얼 통신을 통해서 라즈베리파이로 보냅니다.

(3) 라즈베리파이에서는 받은 값을 웹서버로 전송합니다.

그럼 먼저 아두이노 코드를 살펴 보도록 하겠습니다. DC 모터 예제에서와 같이 아두이노와 라즈베리파이 사이의 전송은 다음과 같은 규약에 따릅니다.

$dev_name:dev_val.

그래서 loop() 함수의 코드는 다음과 같습니다.

```
void loop() {
  int vrVal = analogRead(VAR_RES_1);
  ckSerial.print("$");
  ckSerial.print("var_res");
  ckSerial.print(":");
  ckSerial.print(vrVal);
  ckSerial.print(".");
  delay(1000);
}
```

가변저항의 값을 읽어 "vrVal" 변수에 저장합니다. 이 값을 소프트웨어 시리얼 함수인 ckSerial.print() 를 이용하여 라즈베리파이 쪽으로 보냅니다. 이 함수가 여러 개 있는데 예를 들어 가변저항의 값이 356일 경우에 다음과 같이 보내려고 하는 것입니다.

$var_res:356.

가변저항의 이름(var_res)과 값을 제일 앞에 달러($), 중간에 콜론(:), 끝에 점(.)을 포함하여 같이 보냅니다. 이렇게 아두이노 코드는 간단히 끝납니다. 그리고 가변저항의 아날로그 핀 번호는 다음과 같이 A3 로 정의 합니다.

```
#define VAR_RES_1 A3
```

라즈베리파이 코드는 조금 더 복잡합니다. 그런데, 이전의 DC 모터 예제의 아두이노 코드와 비슷합니다. 아두이노와 시리얼 통신을 통해서 한 문자씩 읽어와 msg 변수에 저장합니다.

```
msg = port.read(1)
```

이 문자중에서 달러($)를 찾아지면 디바이스 이름(dev_name) 변수에 입력되는 문자들을 저장할 것입니다. 그래서 dev_name_en 변수를 1 로 합니다. 이 문자 중에서 콜론(:)이 찾아지면 디바이스 값(dev_val) 변수에 입력되는 문자들을 저장할 것입니다. 그래서 dev_val_en 변수를 1 로 합니다.

```
if (msg == '$'):
        dev_name_en = 1
        dev_val_en = 0
    elif (msg == ':'):
        dev_name_en = 0
        dev_val_en = 1
```

점(.) 을 찾으면 디바이스 이름과 값이 모두 저장된 것을 뜻합니다. 그래서 이 때 디바이스 이름과 값을 웹서버로 보냅니다.

```
elif (msg == '.'):
        dev_name_en = 0
        dev_val_en = 0
        #print dev_name, "----------", dev_val
            params = urllib.urlencode({"var_res":dev_val})
            req = urllib2.Request("http://www.ckiot.net/ckiot_table.php", params)
        res = opener.open(req)
```

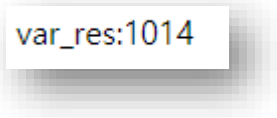
웹서버의 DB 값을 읽어와서 출력합니다. 이것을 다음과 같이 무한 반복하면 코드가 완성됩니다.

```
while 1:
    msg = port.read(1)
    if (msg == '$'):
        dev_name_en = 1
        dev_val_en = 0
    elif (msg == ':'):
        dev_name_en = 0
        dev_val_en = 1
    elif (msg == '.'):
        dev_name_en = 0
        dev_val_en = 0
        #print dev_name, "----------", dev_val
            params = urllib.urlencode({"var_res":dev_val})
            req = urllib2.Request("http://www.ckiot.net/ckiot_table.php", params)
        res = opener.open(req)
        read_msg = res.read()
        read_msg = read_msg[1:]
        dev_list = read_msg.split('.')
        for dev_name_val in dev_list:
            dev_name_val_list = dev_name_val.split(':')
            dev_name = dev_name_val_list[0]
            dev_val = dev_name_val_list[1]
            if (dev_name == "var_res"):
                print dev_name, " : ", dev_val
        dev_name = ""
        dev_val = ""
    else:
        if (dev_name_en == 1):
            dev_name += msg;
        elif (dev_val_en == 1):
            dev_val += msg;
```

아두이노 코드는 업로드하고 라즈베리파이 코드는 실행시킵니다. 그리고 코딩키트의 가변저항을 돌려보십시요. 다음과 같은 가변저항 값이 출력되는 것을 보실 수 있을 것입니다.

```
pi@raspberrypi:~/ckiot $ sudo python e00_ckiot_rx_vr.py
1
Login OK
var_res  :  125
var_res  :  125
var_res  :  110
var_res  :  0
var_res  :  0
var_res  :  61
var_res  :  262
var_res  :  469
var_res  :  597
var_res  :  847
var_res  :  957
var_res  :  1015
```

PC 의 웹브라우저에서도 가변저항 값이 변하는 것을 확인할 수 있습니다. 웹브라우저의 새로고침 버튼을 눌러 보세요.



< 예제 코드 : 가변저항 값 웹서버에 전달하기 >

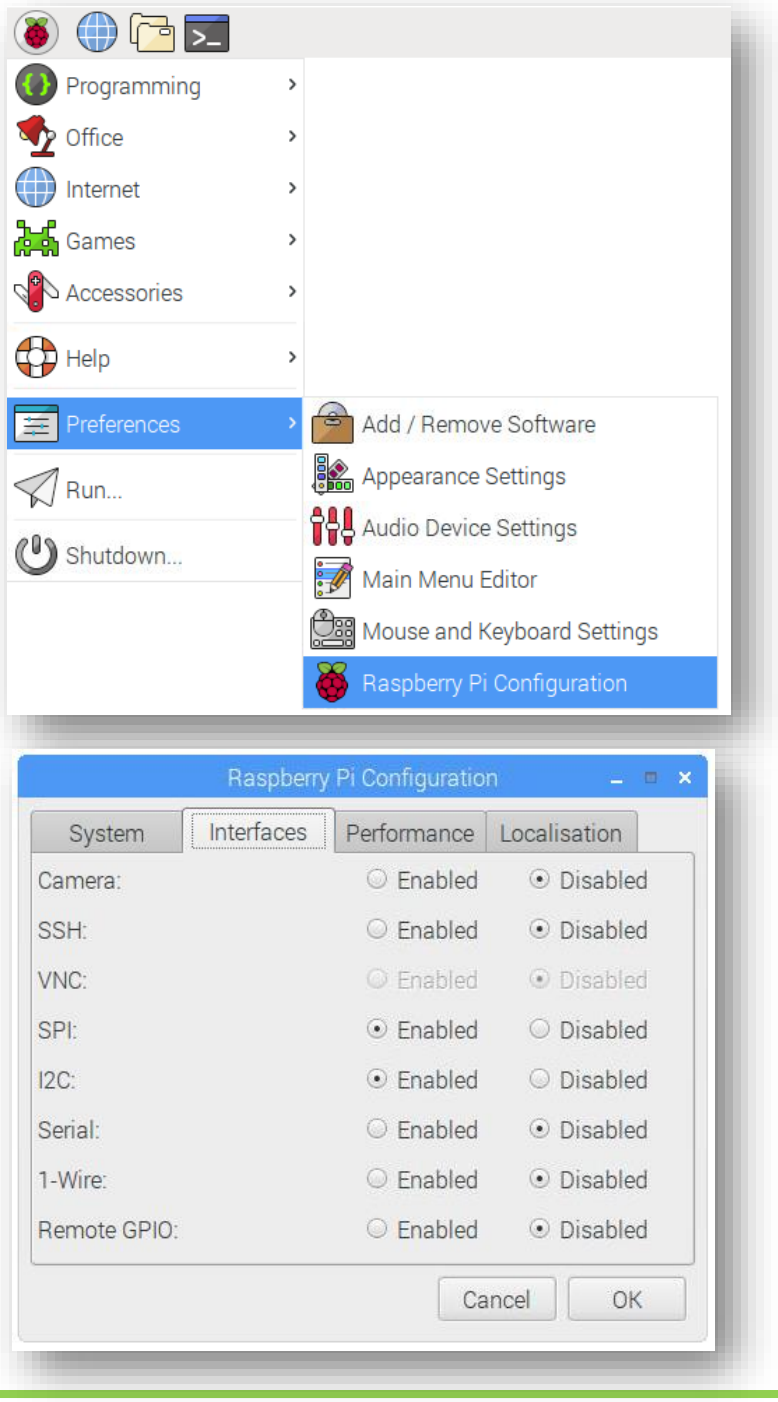
< 라즈베리파이 코드 : e00_ckiot_rx_vr.py >

< 아두이노 코드 : e00_ckiot_vr_tx.ino >

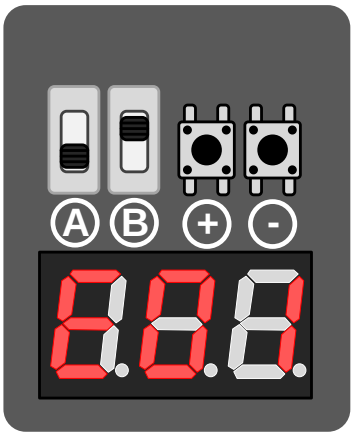
[라즈베리파이와 아두이노 I²C 통신하기]

I²C(Inter-Integrated Circuit)는 아이스퀘어씨(I Square C) 또는 아이투씨(I Two C)라고 읽습니다. 이것을 TWI(Two Wire Interface)라고도 합니다. 필립스에서 개발한 시리얼 통신으로 PCB(Printed Circuit Board)에서 저속의 디바이스를 연결할 때 사용합니다. 장점으로는 두 선을 사용하여 통신을 함으로 연결해야 할 선이 매우 적다는 것입니다. 단점으로는 속도가 매우 느립니다.

지금부터 라즈베리파이와 아두이노 간에 I²C 통신을 하겠습니다. 그럼 먼저 라즈베리파이에서 다음과 같이 I²C 를 활성화 합니다. 코딩키트를 사용하신다면 벌써 활성화가 되어 있을 것입니다.



코딩키트의 스위칭 ID는 다음과 같이 E01 로 변경합니다.



스위칭 ID, E01 에서 라즈베리파이는 2 번과 3 번을 사용하여 I²C 통신을 합니다. 라즈베리파이의 코드는 다음과 같습니다.

```
import smbus
import time

bus = smbus.SMBus(1)

address = 0x04

cnt = 0
while 1:
    bus.write_byte(address, cnt)
    print cnt
    if (cnt == 255):
        cnt = 0
    else:
        cnt = cnt + 1
    time.sleep(1)
```

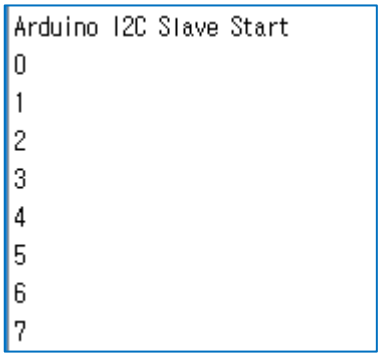
먼저 smbus 를 import 합니다. 라즈베리파이에서는 이 SMBus(System Management Bus) 라이브러리의 함수를 이용하여 I²C 로 데이터를 전송하거나 받습니다. 라즈베리파에는 I²C 포트가 2 개가 있습니다. 우리는 0 번과 1 번 중 1 번을 사용할 것입니다. 그래서 SMBus(1) 라고 합니다. I²C 의 Slave 주소는 0x04 로 합니다. while 문에서 cnt 가 1 초(time.sleep(1))초 간격으로 증가합니다. 이 값을 아두이노에 보낼 것인데 bus.write_byte() 함수를 사용하였습니다. 이 함수는 주소와 데이터를 인자로 받습니다. 데이터는 정수 값만 가능합니다. cnt 변수는 0~255 사이에서 변합니다. 이것은 전송을 한 바이트 내에서 하기 때문입니다.

I²C Slave 로 동작하는 아두이노 코드는 다음과 같습니다.

```
#include <Wire.h>

#define I2C_SLAVE_ADDRESS 0x04

void setup() {
  Serial.begin(57600);
  Wire.begin(I2C_SLAVE_ADDRESS);
```

```
  Serial.println("Arduino I2C Slave Start");
  Wire.onReceive(rxData);
}

void loop() {
  delay(100);
}

void rxData(int byteCount){
 while(Wire.available()) {
   char rxMsg = Wire.read();
   Serial.println(rxMsg, HEX);
 }
}
```

Wire.h 를 Include 합니다. 여기에 있는 함수들을 사용합니다. I²C Slave 의 주소는 라즈베리파에서 정의한 주소와 같은 주소인 0x04 를 사용합니다. setup() 함수에서는 시리얼모니터를 사용하기 위하여 시리얼 통신을 초기화 해 줍니다. 그리고 I²C 를 사용하기 위하여 Wire.begin() 함수를 사용합니다. 이 때 Slave 주소를 입력하면 아두이노는 I²C 의 Slave 모드가 되고 입력되 주소가 Slave 주소로 사용됩니다. 여기에서 I²C 통신으로 어떤 데이터가 들어왔을 때 사용하는 함수를 Wire.onReceive() 함수를 이용하여 정의해 줍니다. rxData 함수를 사용할 것입니다. loop() 함수에서는 딱히 하는 일이 없습니다. rxData() 함수에서는 Wire.available() 함수를 사용합니다. 이 함수는 I²C 통신을 통하여 어떤 데이터가 전송이 되면 그 데이터의 바이트 수만큼 반환해 줍니다. 이 값을 while 문에서 체크하여 Wire.read() 함수를 이용하여 rxMsg 변수에 저장합니다. 이 변수의 값을 시리얼 모니터로 전송합니다. 그러면 시리얼 모니터에는 다음과 같이 출력이 됩니다.



라즈베리파이에서 전송한 값이 출력되는 것을 확인할 수 있습니다.

아두이노 코드에서 while 문에서는 계속해서 체크하는 Wire.available() 함수는 현재 I²C 버퍼에 남아 있는 데이터의 바이트 양을 반환합니다. 이 양은 I²C 통신을 통해서 마스터로부터 받은 값이 채워지면 증가하고 Slave 인 아두이노에서 Wire.read() 함수를 통해서 한 바이트를 읽어가면 감소합니다.

이번에는 데이터를 아두이노에서 라즈베리파이로 보내겠습니다. 여전히 마스터는 라즈베리파이입니다. 그래서 라즈베리파이에서 아두이노의 데이터를 읽어 오는 것입니다. 라즈베리파이의 코드는 다음과 같습니다.

```
import smbus
import time

bus = smbus.SMBus(1)

address = 0x04

cnt = 0
while 1:
    num = bus.read_byte(address)
    print num, "(", cnt, ")"
    cnt = cnt + 1
    time.sleep(1)
```

이전의 라즈베리파이에서 아두이노로 데이터를 보내는 코드와 매우 비슷합니다. 거기서 bus.write_byte() 함수를 사용했다면 여기서는 bus.read_byte() 함수를 사용했다는 것만 다릅니다. 이 값을 num 변수로 받아서 출력합니다.

아두이노 코드는 다음과 같습니다.

```
#include <Wire.h>

#define I2C_SLAVE_ADDRESS 0x04

int cnt = 0;

void setup() {
  Serial.begin(57600);
  cnt = 0;

  Wire.begin(I2C_SLAVE_ADDRESS);
  Serial.println("Arduino I2C Slave Start");
  Wire.onRequest(txData);
}

void loop() {
  delay(100);
}

void txData(){
   Wire.write(cnt);
   if (cnt == 255)
            cnt = 0;
   else
            cnt++;
}
```

이전 코드의 setup() 함수에서 Wire.onReceive() 함수를 정의 했다면 이번 코드에서는 Wire.onRequest() 함수를 정의합니다. 이 함수의 인자로 마스터에서 데이터를 읽어 가는 신호가 들어 올 때 사용할 함수를 설정해 줍니다. 이 함수는 txData() 입니다. 이 함수에서는 Wire.write() 함수를 이용해서 마스터인 라즈베리파이 쪽으로 보낼 데이터를 정합니다. 카운터를 만들어 cnt 변수에 저장하고 이것을 Wire.write() 함수를 이용하여 보냅니다. 이렇게 보내준 값을 라즈베리파이에서 출력하면 다음과 같습니다.

```
pi@raspberrypi:~/e00_rsp_example $ sudo python e00_rsp_i2c_rx.py
0 ( 0 )
1 ( 1 )
2 ( 2 )
3 ( 3 )
4 ( 4 )
5 ( 5 )
6 ( 6 )
7 ( 7 )
8 ( 8 )
9 ( 9 )
10 ( 10 )
11 ( 11 )
12 ( 12 )
```

이렇게 하여 I²C 통신을 이용하여 라즈베리파이와 아두이노 사이에 데이터를 주고 받는 실습을 해 보았습니다.

< 예제 코드 : 라즈베리파이와 아두이노 I²C 통신하기 >

< 라즈베리파이 코드 : e01_rsp_i2c_tx.py >

< 아두이노 코드 : e01_ard_i2c_rx.ino >

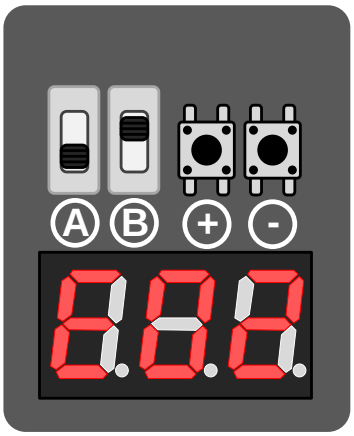
< 라즈베리파이 코드 : e01_rsp_i2c_rx.py >

< 아두이노 코드 : e01_ard_i2c_tx.ino >

[라즈베리파이와 아두이노 SPI 통신하기]

SPI (Serial Peripheral Interface) 는 이전의 아두이노와 라즈베리파이 교재에서 많이 해 보셔서 잘 아실 것입니다.

코딩키트의 스위칭 ID는 다음과 같이 E02 로 변경합니다.



이번에 할 SPI 통신 예제에서는 라즈베리파이를 마스터로 하고 아두이노를 슬래이브로 하는 예제를 해 보겠습니다. 먼저 라즈베리파이 코드를 보겠습니다.

```
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
import spidev
import time

SPI_SS = 19

GPIO.setup(SPI_SS, GPIO.OUT)
GPIO.output(SPI_SS, 1)

ck_spi = spidev.SpiDev()
ck_spi.open(0, 1)

cnt = 0
while 1:
    GPIO.output(SPI_SS, 0)
    ck_spi.xfer([cnt])
    GPIO.output(SPI_SS, 1)
    cnt = cnt + 1
    time.sleep(1)
```

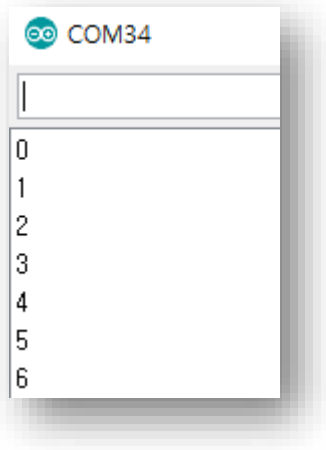
전에 많이 했던 코드라서 자세한 설명은 빼겠습니다. 자세한 내용은 아두이노나 라즈베리파이 교재를 참고해 주세요. 이번에 사용할 SPI 에서는 슬래이브 선택(Slave Select : SS) 신호를 GPIO 19 번으로 사용하는 것만이 다른 점입니다. while 구문을 보면 GPIO 출력으로 슬래이브 설정을 0 으로 해 줍니다.

그리고 ck_spi.xfer() 함수를 사용해서 데이터를 아두이노로 전송합니다. 이 데이터는 1 씩 증가하는 카운터 값입니다. ck_spi.xfer() 함수에 데이터를 보낼 때는 리스트 형태로 보내야 하기 때문에 대괄호([cnt])를 이용합니다.

다음은 아두이노 코드입니다.

```
#include <SPI.h>

void setup() {
  Serial.begin(57600);

  SPCR |= _BV(SPE);          // SPI Enable
  SPI.attachInterrupt();
}

ISR(SPI_STC_vect) {
   Serial.println(SPDR);
}

void loop() {
   delay(100);
}
```

이전의 아두이노 교재에서는 SPI 의 마스터로 동작하였는데, 이번에는 슬래이브로 동작합니다. 아두이노의 SPI 는 초기에 슬래이브 모드로 설정이 되어 있습니다. 그래서 SPCR(SPI 컨트롤 레지스터)로 SPI 를 활성화만 시켜주면 됩니다. 그리고 SPI 통신으로 어떤 데이터가 들어 왔을 때 인터럽트를 발생할 수 있게 SPI.attachInterrupt() 함수 실행해 줍니다. 이 인터럽트에 의해서 실행되는 함수는 ISR(SPI_STC_vect) 함수입니다. 이 함수 내에서 라즈베리파이로 받은 값을 시리얼 모니터로 출력합니다. 라즈베리파이에서 받은 값은 SPDR(SPI 데이터 레지스터)에 저장이 됩니다. 다음은 아두이노의 시리얼 모니터에 카운트 값이 출력된 모습니다.



이번에는 아두이노에서 보내고 라즈베리파이에서 받는 것을 해 보겠습니다. 라즈베리파이 코드는 다음과 같습니다.

```
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
```

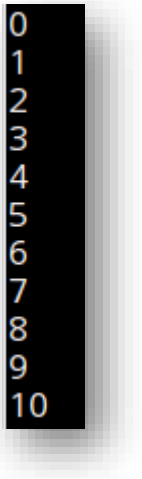
```
import spidev
import time

SPI_SS = 19

GPIO.setup(SPI_SS, GPIO.OUT)
GPIO.output(SPI_SS, 1)

ck_spi = spidev.SpiDev()
ck_spi.open(0, 1)

while 1:
    GPIO.output(SPI_SS, 0)
    rxData = ck_spi.xfer([0xff])
    GPIO.output(SPI_SS, 1)
    print rxData[0]
    time.sleep(1)
```

SPI 는 보는 것과 받는 것을 동시에 합니다. 그래서 데이터를 받으려면 보내야 합니다. 코드를 보면 ck_spi.xfer([0xff]) 0xff 라는 데이터를 보내는 것입니다. 보내는 것과 동시에 받기 때문에 rxData 변수에 아두이노로부터 받은 데이터가 저장됩니다. 받는 데이터들도 리스트로 저장이 됩니다. 그래서 “print rxData[0]” 으로 합니다.

아두이노 코드는 다음과 같습니다.

```
#include <SPI.h>

int cnt = 0;

void setup() {
    Serial.begin(57600);

    pinMode(MISO, OUTPUT);
    SPCR |= _BV(SPE);          // SPI Enable
    SPI.attachInterrupt();

    SPDR = cnt;
}

ISR(SPI_STC_vect) {
    Serial.println(cnt);
    cnt ++;
    SPDR = cnt;
}

void loop() {
    delay(100);
}
```

SPI 는 받는 것과 동시에 보냅니다. 그래서 SPDR(SPI 데이터 레지스터)에 어떤 데이터를 저장해 두면 이 데이터가 전송이 됩니다. setup() 함수에서 미리 SPDR 에 데이터를 쓴 것은 미리 써 두어야 라즈베리파이에서 데이터를 보낼 때 미리 써 둔 데이터를 전송합니다. 라즈베리파이에서 SPI 를 통하여 받은 값을 출력한 것은 다음 그림과 같습니다.



그럼 이제 데이터를 라즈베리파이와 아두이노에서 동시에 주고 받는 예제를 해 보겠습니다. 라즈베리파이 코드는 다음과 같습니다.

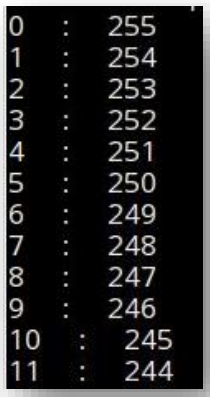
```
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
import spidev
import time

SPI_SS = 19

GPIO.setup(SPI_SS, GPIO.OUT)
GPIO.output(SPI_SS, 1)

ck_spi = spidev.SpiDev()
ck_spi.open(0, 1)

cnt = 0;
while 1:
   rxData = 0;
   GPIO.output(SPI_SS, 0)
   rxData = ck_spi.xfer([cnt])
   GPIO.output(SPI_SS, 1)
   print cnt, " : ", rxData[0]
   cnt = cnt + 1
   time.sleep(1)
```

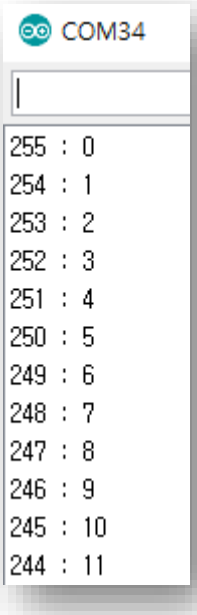
이전 코드와 매우 비슷합니다. 차이가 있다면 이전 코드에서는 0xff 라는 값으로 고정해서 보냈다면 이번에는 보내고자 하는 데이터를 보냅니다. 이 예제에서는 0 에서부터 1 씩 증가하는 카운트 값을 보냈습니다. 그리고 받음 값은 rxData 리스트에 저장하고 출력합니다.

아두이노 코드는 다음과 같습니다.

```
#include <SPI.h>

unsigned char cnt = 255;

void setup() {
  Serial.begin(57600);

  pinMode(MISO, OUTPUT);
  SPCR |= _BV(SPE);         // SPI Enable
  SPI.attachInterrupt();

  SPDR = cnt;
}

ISR(SPI_STC_vect) {
  Serial.print(cnt);
  Serial.print(" : ");
  Serial.println(SPDR);
  cnt--;
  SPDR = cnt;
}

void loop() {
  delay(100);
}
```

아두이노 코드도 이전 코드들과 매우 비슷합니다. SPDR 에 255 에서 1 씩 감소하는 카운터 값을 저장해 둡니다. 그러면 이 값이 라즈베리파이에서 데이터를 받으면서 전송이 됩니다. 라즈베리파이와 아두이노의 출력은 다음과 같습니다.





< 예제 코드 : 라즈베리파이와 아두이노 SPI 통신하기 >

< 라즈베리파이 코드 : e02_rsp_spi_tx.py >

< 아두이노 코드 : e02_ard_spi_rx.ino >

< 라즈베리파이 코드 : e02_rsp_spi_rx.py >

< 아두이노 코드 : e02_ard_spi_tx.ino >

< 라즈베리파이 코드 : e02_rsp_spi_tx_rx.py >

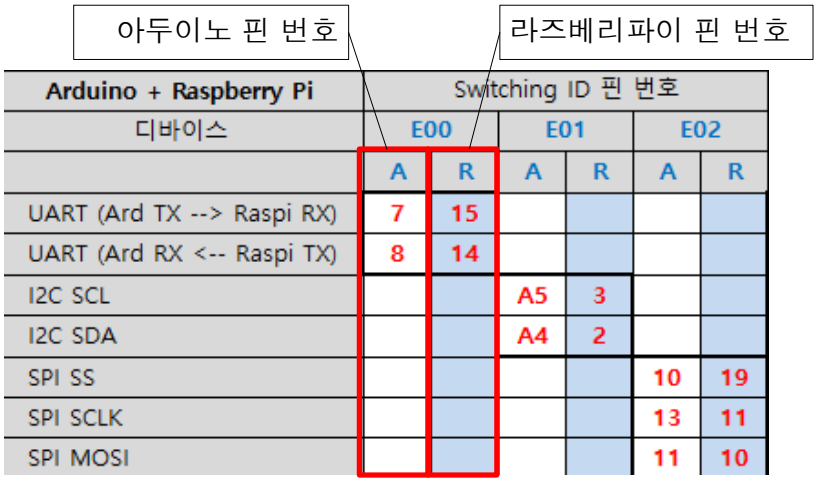
< 아두이노 코드 : e02_ard_spi_tx_rx.ino >

[부록 A : 코딩키트 확장판에서 스위칭(Switching) ID]

다음은 코딩 키트의 확장판에서 스위칭 ID 에 따른 디바이스 연결 표입니다.

Arduino + Raspberry Pi	Switching ID 핀 번호					
디바이스	E00		E01		E02	
	A	R	A	R	A	R
UART (Ard TX --> Raspi RX)	7	15				
UART (Ard RX <-- Raspi TX)	8	14				
I2C SCL			A5	3		
I2C SDA			A4	2		
SPI SS					10	19
SPI SCLK					13	11
SPI MOSI					11	10
SPI MISO					12	9
LED 0	A4		√			14
LED 1	A5		√			15
LED 2		24	√			24
LED 3		25	√			25
Button 0	12		√			21
Button 1	13		√			13
Button 2		6	√			6
Button 3		16	√			16
Buzzer	5		5		5	
DC Motor Enable	6			17		17
DC Motor Forward	10			√		√
DC Motor Backward	11			√		√
DC Motor Encoder Enable		20		20		20
DC Motor Encoder Data		√		√		√
Sound Sensor Detect		4		4		4
IR Sensor LED	4		4		4	
IR Sensor Detect		12		12		12
Servo Motor	9		9			√
Dotmatrix		√	√			√
7-Segment						
Device Control SPI		*R1	*A1			*R1
Character LCD EN, RS, Data		√	√		*A2	
Character LCD Back Light		√	√		2	
Switch 0		26	√			26
Switch 1		27	√			27
Switch 2		2	√			2
Switch 3		3	√			3
Serial Monitor	0,1		0,1		0,1	
Ard --> Raspi Connect	3	23	3	23	3	23
Raspi --> Ard Connect	2	22	2	22		
Ard --> Raspi Connect			6	21		
Raspi --> Ard Connect			7	24		
Ard --> Raspi Connect			8	25		
ADC SPI (Raspi Only)		*R2		*R2		*R2

표에서 붉은색으로 표시된 숫자는 파란색으로 표시한 스위칭 ID 에서 각각의 디바이스에 연결된 아두이노와 라즈베리파이의 핀 번호입니다. 파란색 A 로 표시된 열은 아두이노의 핀 번호를 의미하며 파란색 R 로 표시된 열은 라즈베리파이의 핀 번호를 의미합니다.



Arduino + Raspberry Pi	Switching ID 핀 번호					
디바이스	E00		E01		E02	
	A	R	A	R	A	R
UART (Ard TX --> Raspi RX)	7	15				
UART (Ard RX <-- Raspi TX)	8	14				
I2C SCL			A5	3		
I2C SDA			A4	2		
SPI SS					10	19
SPI SCLK					13	11
SPI MOSI					11	10

라즈베리파이 코딩은 R 로 표시된 열의 핀 번호를 이용하시고 아두이노 코드는 A 로 표시된 열의 핀 번호를 이용하여 코딩하시면 됩니다. 확인(√) 표시는 SPI 확장 모드로 연결하여 더 이상 핀 수는 소요되지 않지만 연결은 가능하다는 표시입니다.

위의 표에서 *R1, *R2 는 라즈베리파이에 여러핀이 연결된 것을 나타내며 그 내용은 다음과 같습니다.

*R1 : SPI(Device)	*R2 : SPI (ADC)
SPI_SS : 7	SPI_SS : 8
SPI_MOSI : 10	SPI_MOSI : 10
SPI_MISO : 9	SPI_MISO : 9
SPI_SCLK : 11	SPI_SCLK : 11

위의 표에서 *A1, *A2 는 아두이노에 여러핀이 연결된 것을 나타내며 그 내용은 다음과 같습니다.

*A1 : SPI	*A2 : Char LCD
SPI_SS : 10	LCD_EN : 9
SPI_MOSI : 11	LCD_RS : 6
SPI_MISO : 12	LCD_D0 : 7
SPI_SCLK : 13	LCD_D1 : A5
	LCD_D2 : 8
	LCD_D3 : A4

[Release Note]

V2.0 : 2017. 7. 18

-. 첫번째 버전 Release

V2.1 : 2018. 2. 5

-. (2018. 1. 27) 7 페이지 : 변경

Mode Setting Switch 그림 수정 – 스위치가 모두 위로 올라가야 하는데, 하나만 올라가 있었음

-. (2018. 1. 27) 바닥글 추가

-. (2018. 2. 5) 15페이지 : 변경

인터넷 연결 관련 내용 수정

-. (2018. 2. 5) 32페이지 : 변경

-. (2018. 2. 5) 39페이지 : 변경