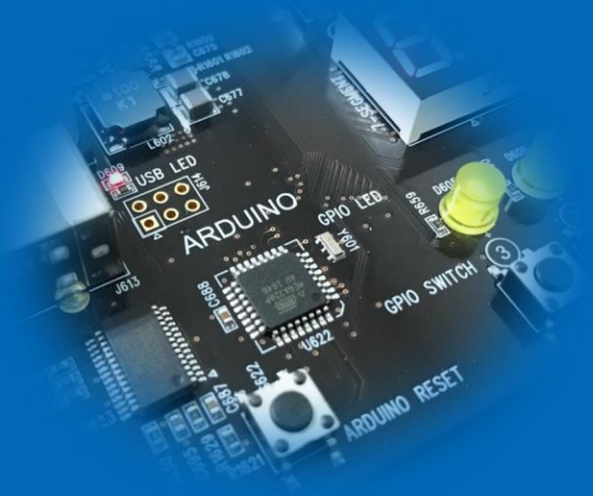
코딩키트

블록 코딩에서 사물인터넷 코딩까지
쉽고 재미있게 배울 수 있어요.

블록 코딩 / 아두이노 코딩 / 라즈베리파이 코딩 / 사물인터넷 코딩

코딩북3 - 아두이노 코딩



SISO DREAM
Silicon to Software

(주)시소드림 SISODREAM Inc.

- 본 문서의 저작권 -

발행일 : 2015 년 8 월 28 일 (초판), 2017 년 3 월 8 일 (개정판)
발행처 : ㈜시소드림
연락처 : ck@sisodream.com / 031-757-7755

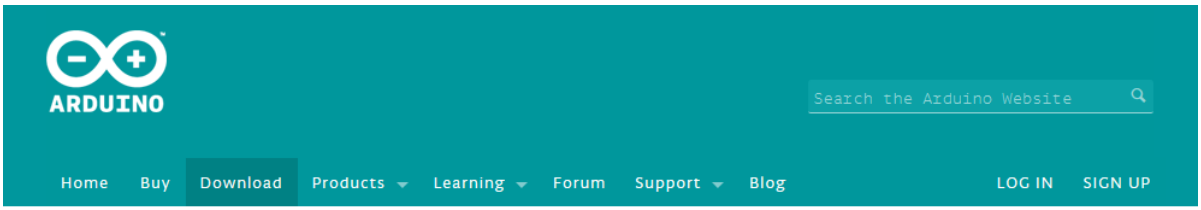
목 차

[아두이노 프로그램 설치 및 실행]

코딩키트에는 매우 간단한 컴퓨터인 아두이노(Arduino)가 장착되어 있습니다. 그래서 컴퓨터에 아두이노 프로그램을 설치해야 합니다. 다음 사이트에서 프로그램을 다운 받습니다. (만약 컴퓨터에 아두이노 프로그램이 설치되어 있다면 이 부분은 지나치셔도 됩니다.)

http://www.arduino.cc/en/Main/Software

사이트를 방문하면 다음과 같은 화면이 보입니다.

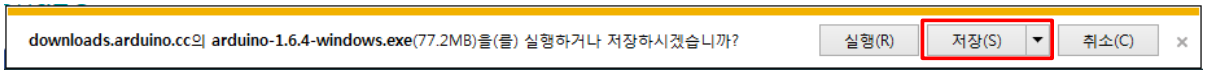


여기서 붉은색 박스 안의 Windows Installer 를 클릭합니다.

그러면 다음과 같은 화면을 볼 수 있습니다.

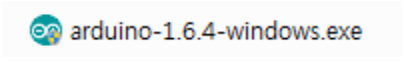
여기서 붉은색 박스 안의 JUST DOWNLOAD 를 클릭합니다.

그러면 다음과 같은 창이 브라우저 하단에 보입니다.



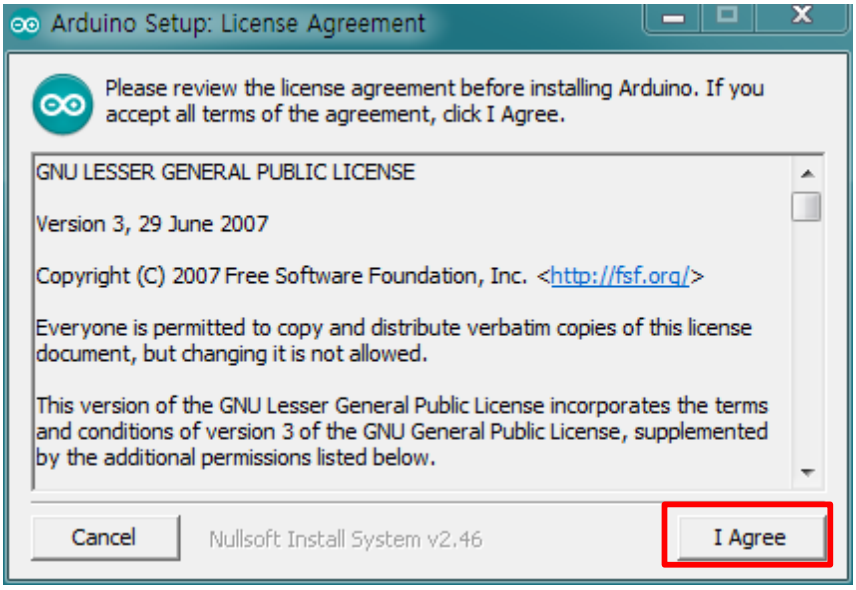
여기서 붉은색 박스 안의 저장 버튼 중 오른쪽에 있는 아래쪽 방향 화살표를 누르면 "다른 이름으로 저장하기" 창이 나옵니다. 그러면 원하는 폴더를 선택하고 저장합니다.

저장한 폴더를 보면 다음과 같은 파일이 있습니다.

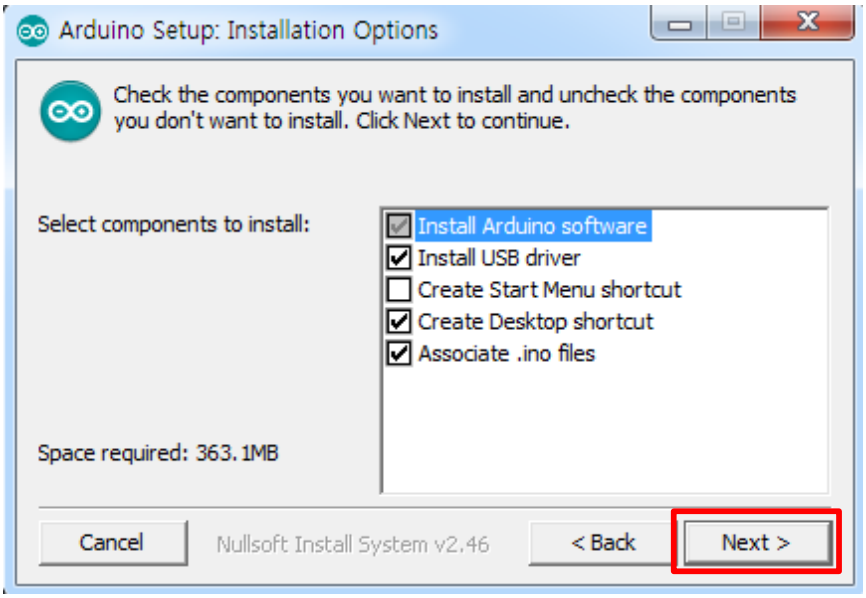


이 파일을 더블 클릭하여 실행합니다. 이제부터 아두이노 프로그램을 설치하는 것입니다.

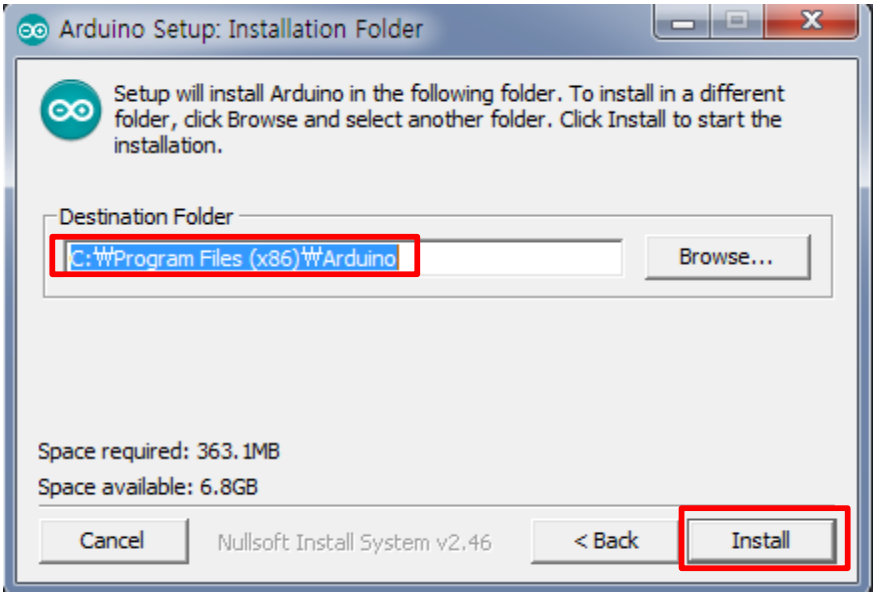
첫 화면은 다음과 같습니다.



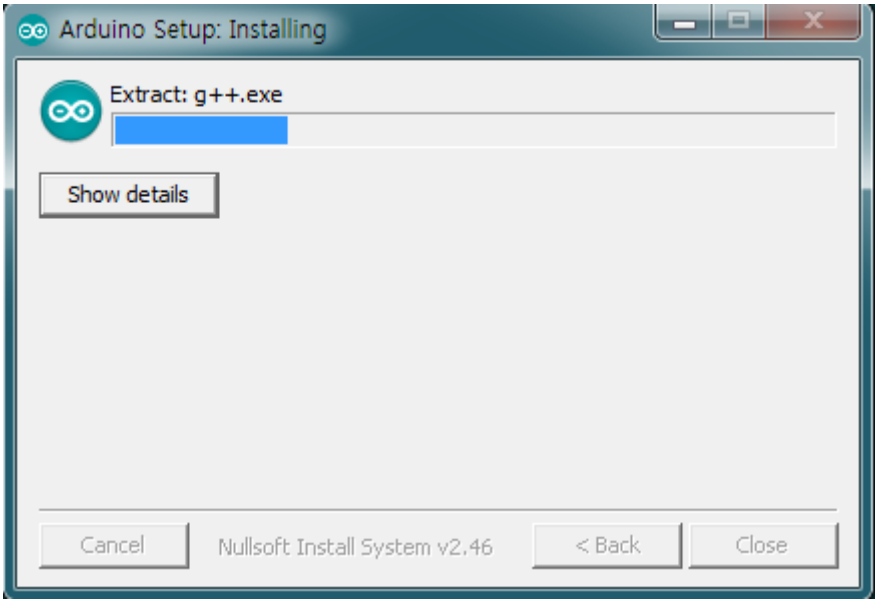
붉은색 박스 안의 "I Agree" 버튼을 클릭합니다. 그러면 다음과 같이 화면이 보입니다.



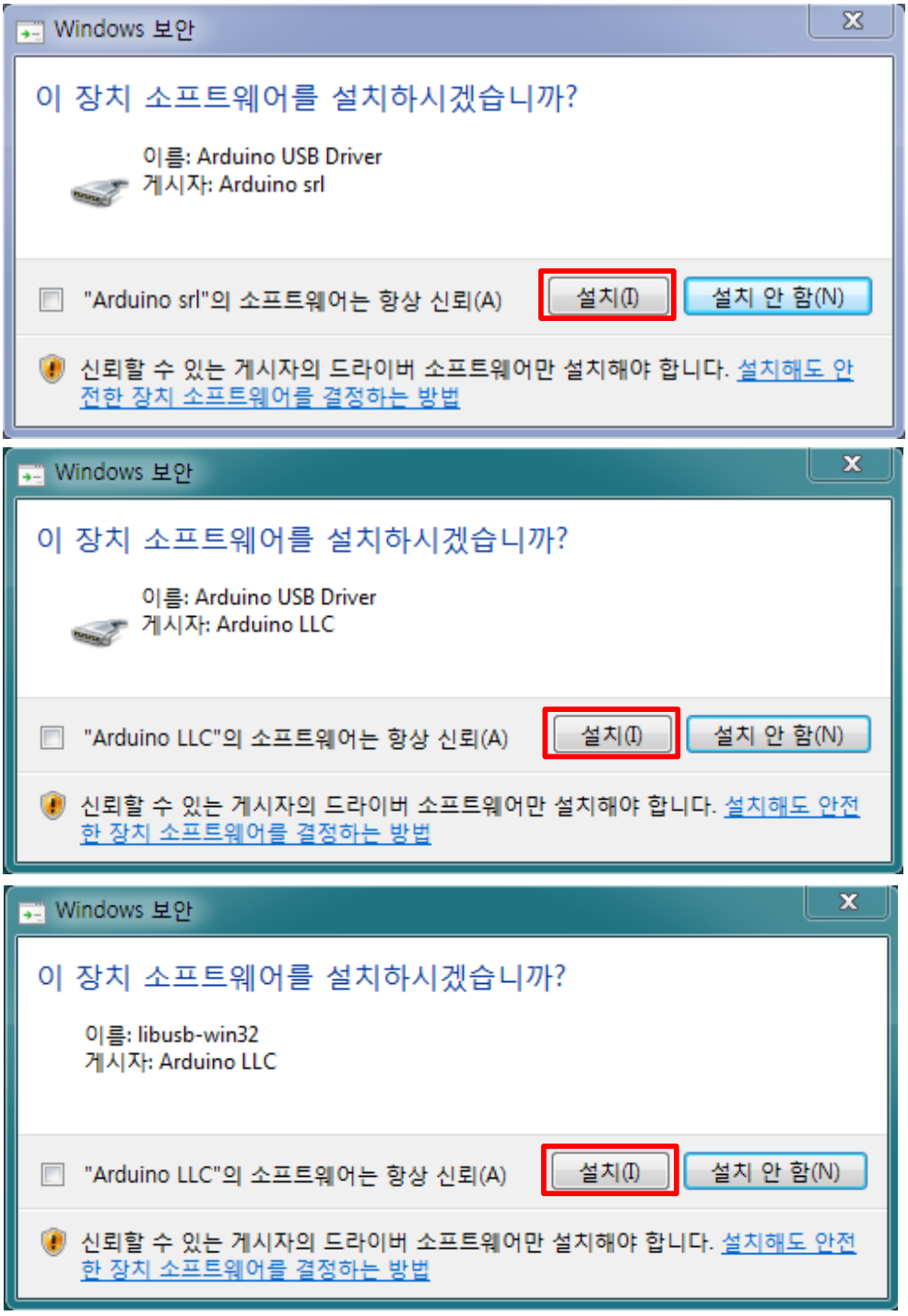
설치할 항목을 선택 합니다. 모든 것을 다 선택하셔도 되고, 위의 그림과 같이 선택하셔도 됩니다. "Next" 버튼을 클릭합니다.



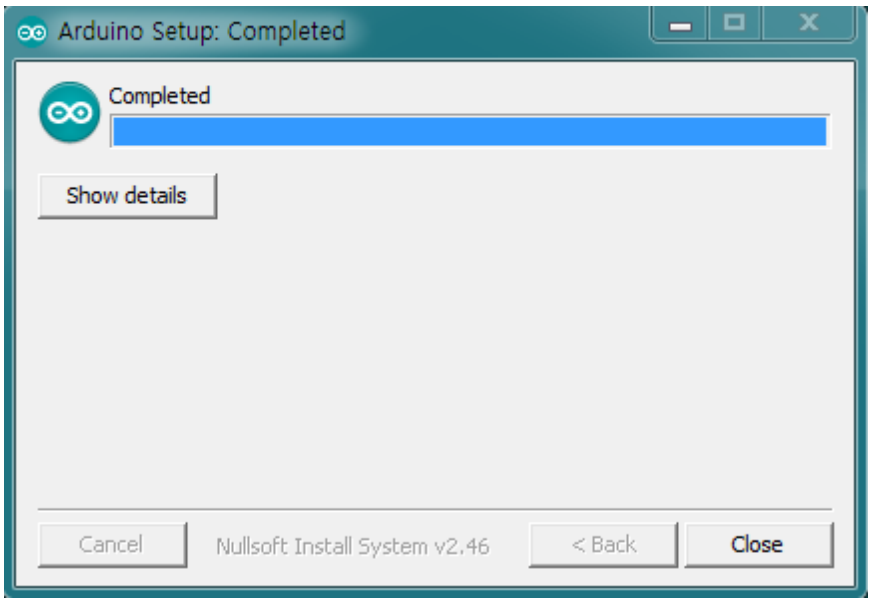
설치할 폴더를 쓰고 "Install" 버튼을 누릅니다. 이 때 기본 폴더를 선택하셔도 되고, "Browser..." 버튼을 이용하여 원하는 설치 폴더를 선택할 수 있습니다. 다음과 같은 설치 중인 화면이 보입니다. 약 2 ~ 3 분 정도 시간이 소요됩니다.



다음과 같이 중간에 설치를 물어 보는 창에서는 모두 "설치" 버튼을 클릭합니다. 이 창들은 모두 나올 수도 있고 이 중 한두 개만 나올 수도 있습니다.

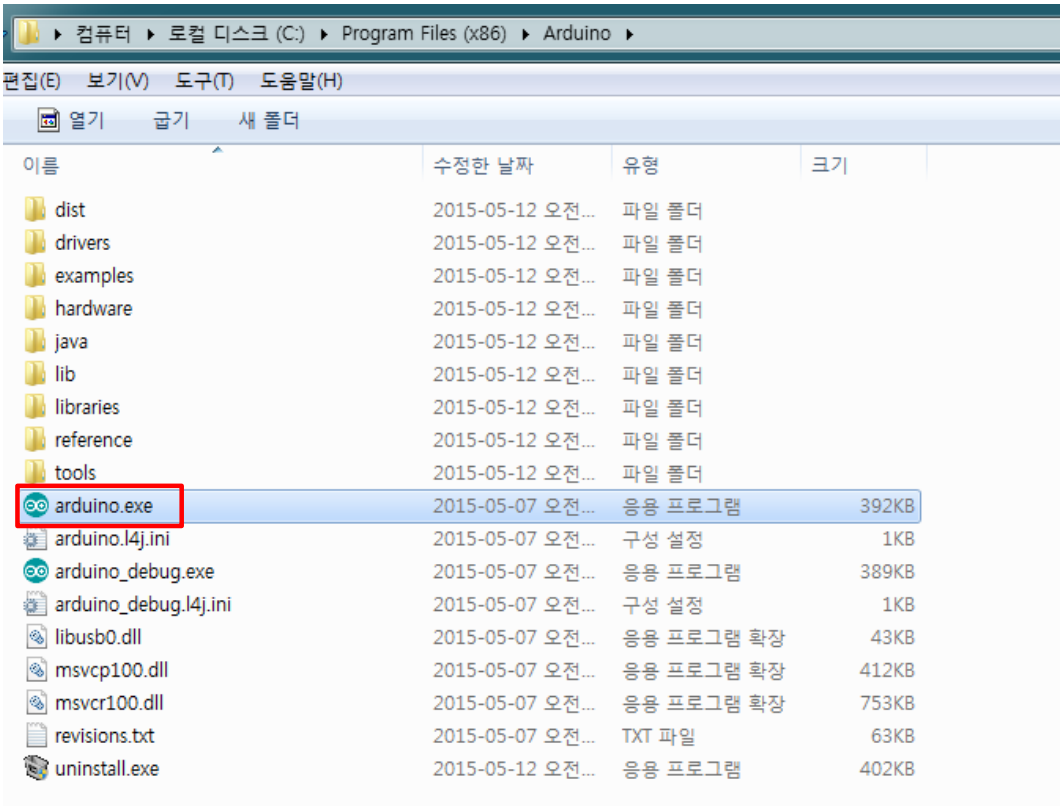


설치가 완료되면 다음과 같이 설치 완료 창이 표시됩니다.

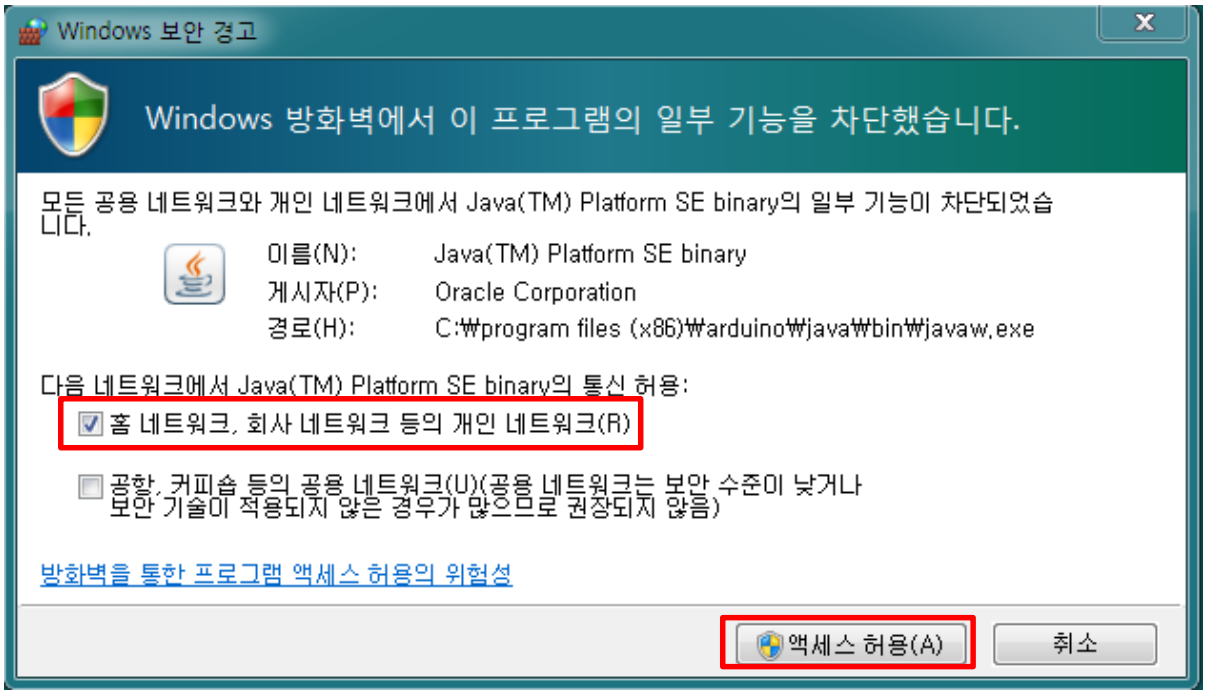


이제 바탕화면의 아두이노 아이콘을 더블 클릭하여 아두이노 프로그램을 실행합니다.

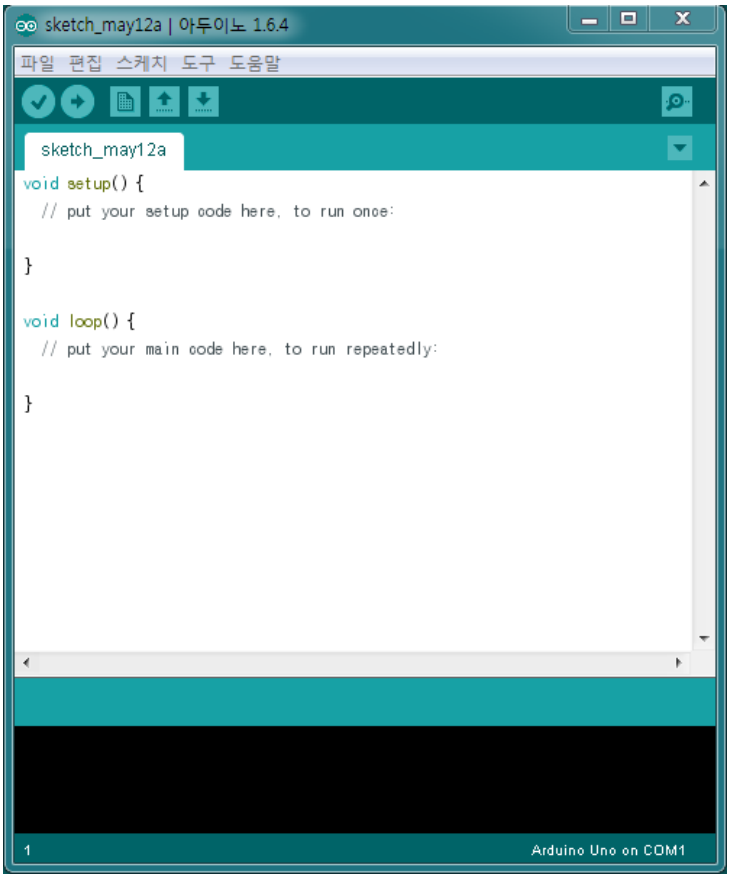
혹시 바탕화면에 아이콘이 없으면 설치 폴더로 가서 arduino.exe 를 실행합니다.



실행 중에 다음과 같은 창이 보이면 "홈 네트워크, 회사 네트워크 등의 개인 네트워크" 만을 체크하고, "액세스 허용" 버튼을 누릅니다.



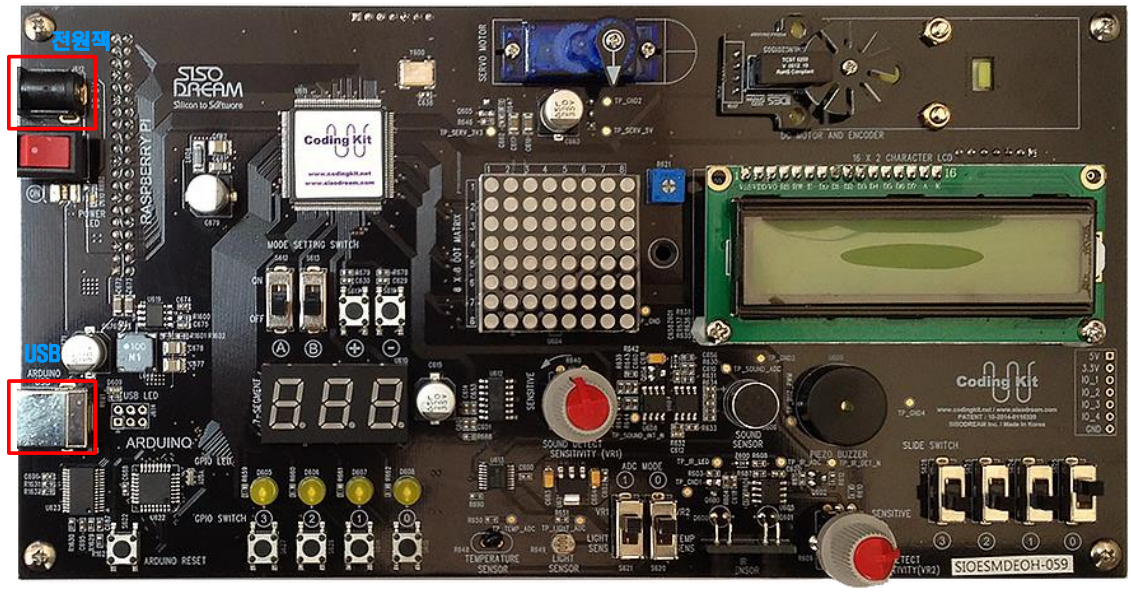
이제 다음과 같이 아두이노 프로그램이 보입니다.



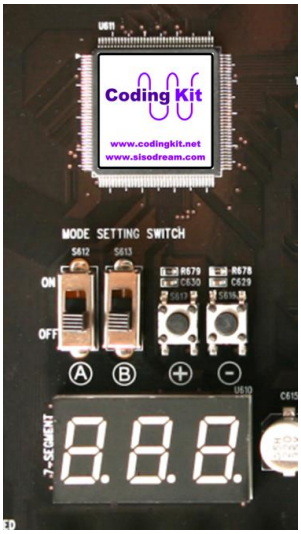
"void setup()" 과 "void loop()" 크게 두 부분으로 나뉘어 있는데요. 편의상 셋업 작업 지시서와 루프 작업 지시서로 이름 붙이겠습니다. 셋업 작업 지시서에 있는 부분은 컴퓨터가 처음 한 번만 수행합니다. 루프 지시서에 있는 부분은 컴퓨터가 계속해서 반복적으로 수행합니다. **즉, 컴퓨터는 셋업 작업 지시서의 내용을 루프 작업 지시서 보다 먼저 처음에 딱 한번만 수행합니다. 그 이후에 루프 작업 지서서의 내용을 계속해서 반복적으로 수행합니다.**

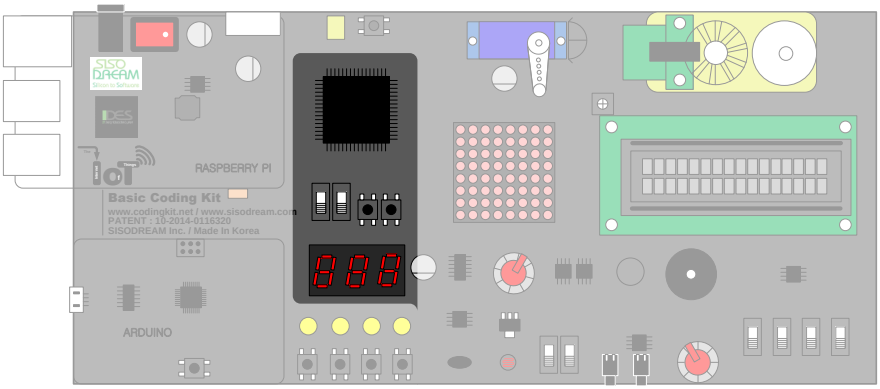
[코딩키트 준비 및 스위칭 시스템]

먼저 아답터를 전원 잭에 연결합니다. 그리고 전원 스위치를 눌러 코딩키트에 전기가 들어가도록 합니다. 그리고 USB 케이블을 PC 에 연결합니다.



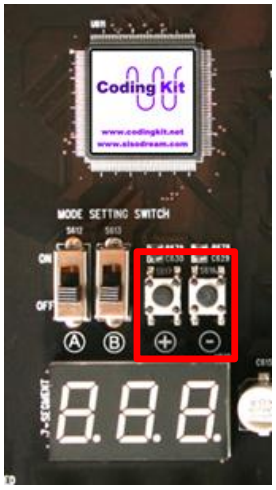
그러면 세븐세그먼트가 깜박일 것입니다. 다음 그림에서 A, B 스위치를 모두 아래로 내리면 코딩키트는 아두이노 코딩을 할 준비가 된 것입니다.





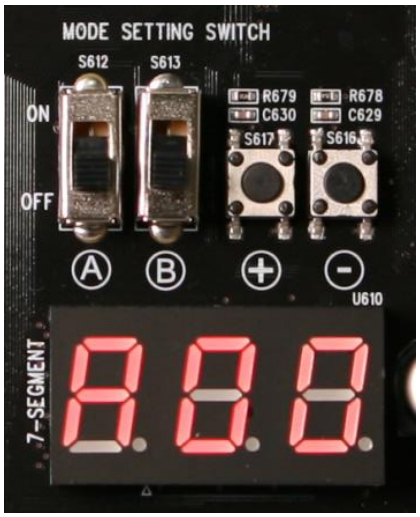
다음 그림과 같이 가장 왼쪽의 세븐세그먼트에 "A" 가 표시될 것입니다. 이것은 아두이노 코딩을 의미합니다.

다음 그림의 붉은색 박스 안의 + 버튼을 누르면 두번째, 세번째 세븐세그먼트의 숫자가 올라갑니다. - 버튼을 누르면 숫자가 내려갑니다. 이렇게 하여 연결 번호를 설정합니다.

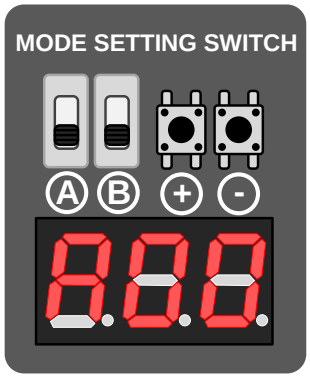


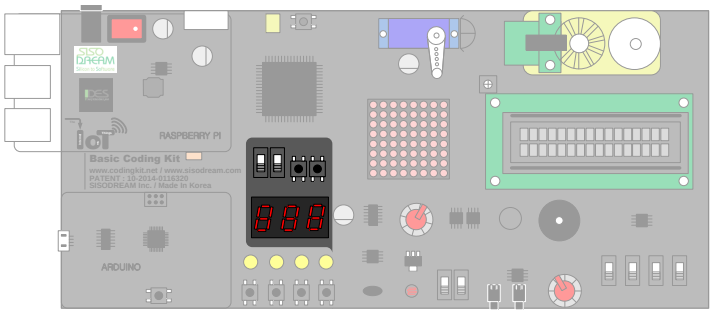
이 연결 번호와 아두이노 표시를 코딩키트에서는 스위칭 ID 라고 합니다. 더 자세한 사항은 "코딩키트 소개" 편을 참고 바랍니다. 이 스위칭 ID는 A00, A01, A02, ... 으로 표시가 됩니다. 이것은 세븐세그먼트에 표시되는 것과 동일합니다. 이제 여러분이 + 혹은 - 버튼을 1~2 초간 계속 누르고 있으면 세븐세그먼트의 깜박임이 멈추고 연결이 이루어집니다. 스위칭 시스템 설정이 완료된 것입니다.

지금부터 할 아두이노 코딩 연습에서의 스위칭 ID 는 "A00" 입니다. 다음 사진과 같이 설정해 주시기 바랍니다. 이렇게 하기 위해서는 A, B 스위치는 모두 내리고 +, - 버튼을 이용하여 숫자는 "00" 으로 맞춥니다. 그리고 + 혹은 – 버튼을 1~2초간 눌러 세븐세그먼트의 깜박임이 멈추도록 합니다.



위 사진의 스위치 위치나 세븐세그먼트의 값이 잘 확인이 안 되시면 아래 그림을 참조하십시요.





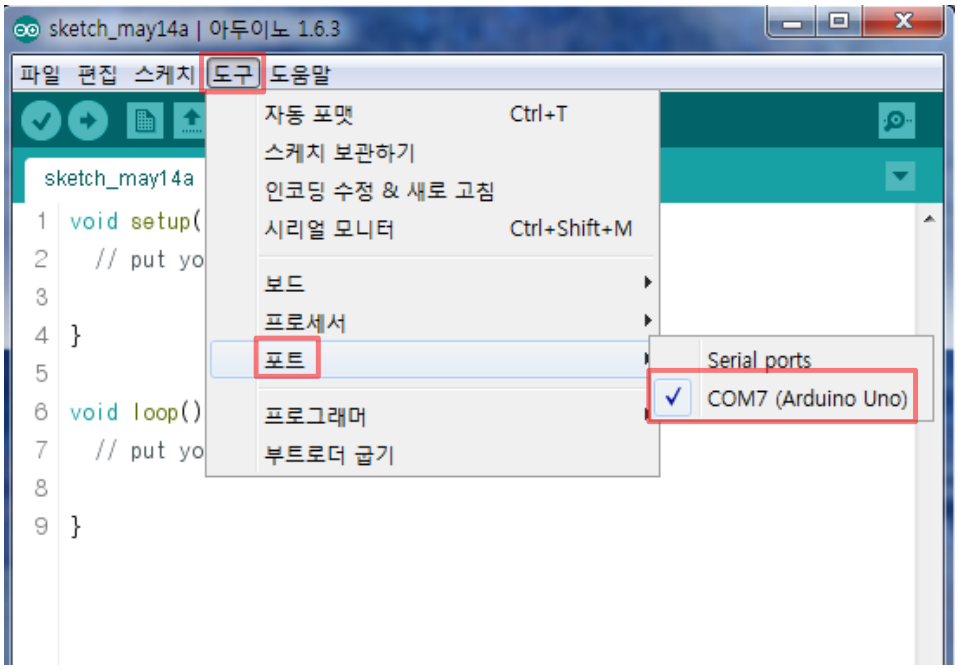
위 그림 중 오른쪽의 그림은 왼쪽의 그림이 보드에서 어느 위치에 해당하는 가를 표시한 것입니다.

각각의 스위칭 ID에 어떤 연결이 되어 있는지에 대해서는 [부록 A]에 정리해 두었습니다.

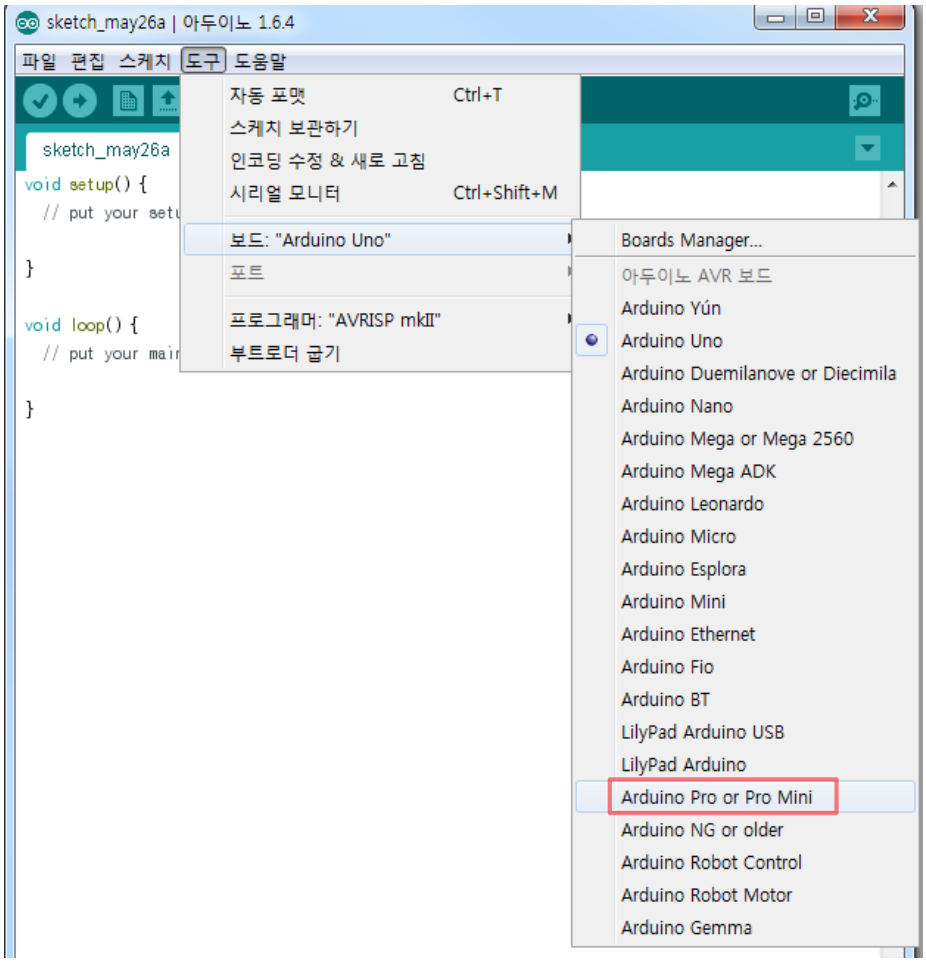
코딩키트의 준비가 완료 되면 다음과 같이 아두이노 프로그램의 "도구 → 포트" 메뉴에서 해당 COM 포트를 확인할 수 있다. 이 포트를 선택해 줍니다. 그림에서는 COM7 이지만 다른 포트로 잡혀 있다면 다른 포트를 선택하여 주십시요.

Note : 위의 설명은 Windows7 을 기준으로 설명하였습니다. Windows8, Windows10 에서의 사용은 코딩 사이트의 관련 게시물을 참고해 주십시요. 코딩키트는 Windows7, Windows8, Windows10 만을 지원합니다.

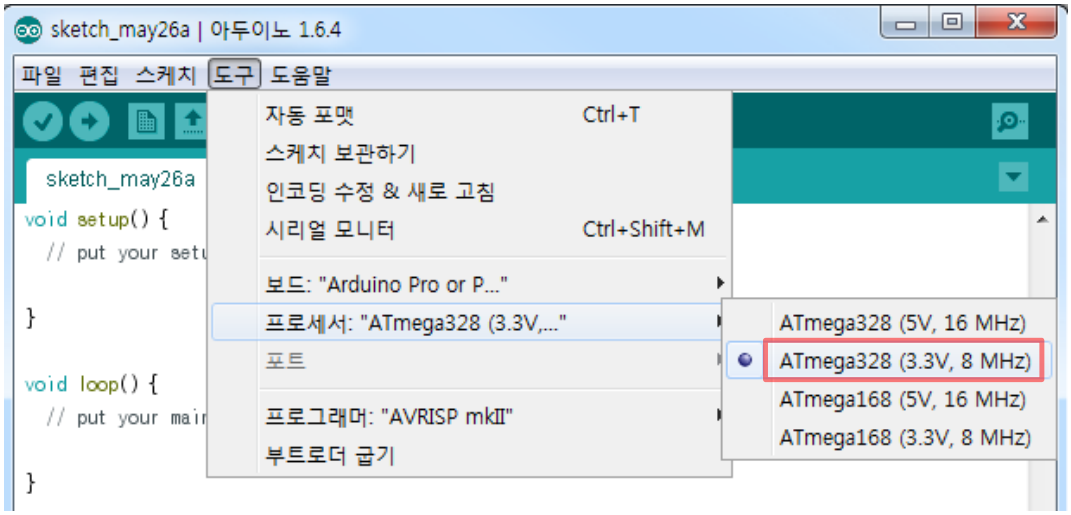
Note : COM 포트가 보이지 않으면 드라이버 설치가 제대로 되었는지 확인 바랍니다. 드라이버 설치 내용은 "코딩키트 소개" 편을 참고해 주십시요. 만약 한참을 기다려도 드라이버 소프트웨어가 자동으로 설치되지 않을 경우 코딩 사이트에 문의해 주십시요.



다음 그림과 같이 "도구 → 보드 → Arduino Pro or Pro Mini" 를 선택합니다.



다음 그림과 같이 "도구 → 프로세서 → ATmega328 (3.3V, 8MHz)" 를 선택합니다.



이제 아두이노 프로그램에 관한 코딩 준비는 다 되었습니다.

이제 준비를 다 하였으니 코딩을 해 보겠습니다. 우리가 하려는 것은 "코딩키트 소개편"에서 예를 들었던 전등에 불이 들어와 3초 동안 켜 있다가 꺼지는 작업 지시서를 작성하는 것입니다. 코드는 다음과 같았습니다.

```
(LAMP, ON);
Delay(3);
(LAMP, OFF);
```

이것을 아두이노 문법에 맞게 쓰면 다음과 같습니다. 크게 다를 것은 없습니다.

```
void setup() {
  // put your setup code here, to run once:
  CK_InitLamp3SecOn();
  digitalWrite(LAMP, ON);
  Delay(3);
  digitalWrite(LAMP, OFF);
}

void loop() {
  // put your main code here, to run repeatedly:
}
```

붉은색 부분이 위에서 얘기한 작업 지시서의 내용입니다. 그 이외의 부분들은 컴퓨터 언어의 문법을 맞추어 주기 위한 부분입니다. "CK_InitLamp3SecOn();" 은 미리 작성한 부분으로 이 코드를 실행시키기 전에 추가할 것입니다. (LAMP, ON) 앞에 있는 digitalWrite 라는 것은 LAMP 에 ON 이라는 디지털 신호를 쓰라는 것으로 나중에 좀 더 살펴 보겠습니다.

이제 이 코드를 코딩키트 (Coding Kit) 에서 동작시켜 보기 전에 다음과 같은 코드를 위의 코드의 앞에 추가하겠습니다.

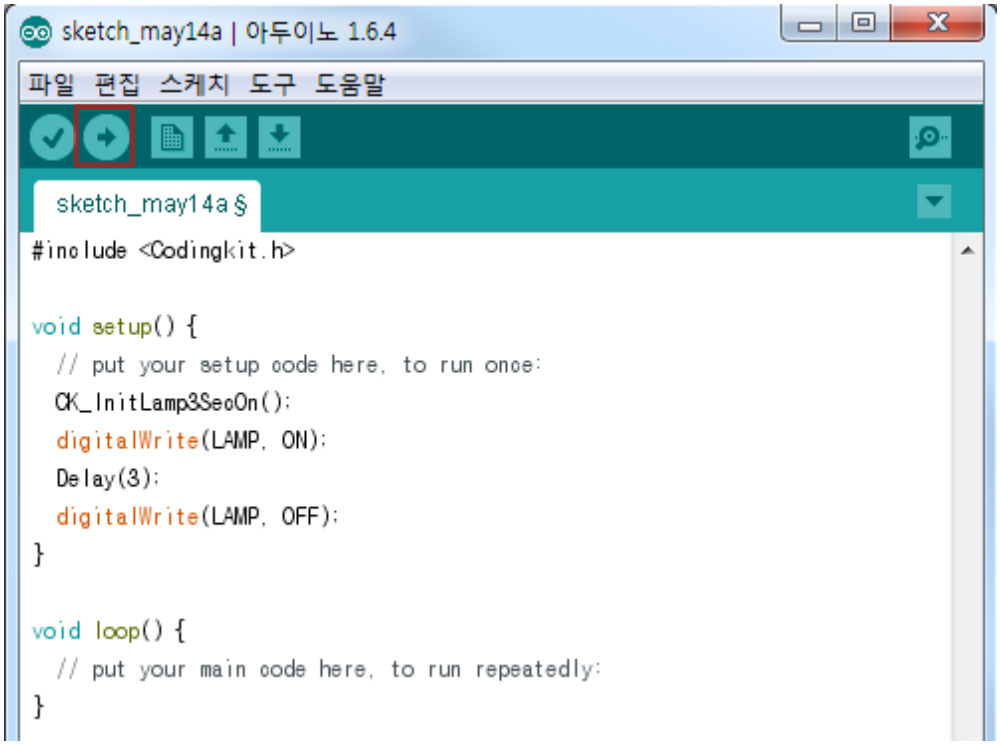
```
#define LAMP 2
#define ON 1
#define OFF 0
#define Delay(x) delay(x*1000)

void CK_InitLamp3SecOn() {
   pinMode(LAMP, OUTPUT);
}
```

그러면 전체 코드는 다음과 같습니다.

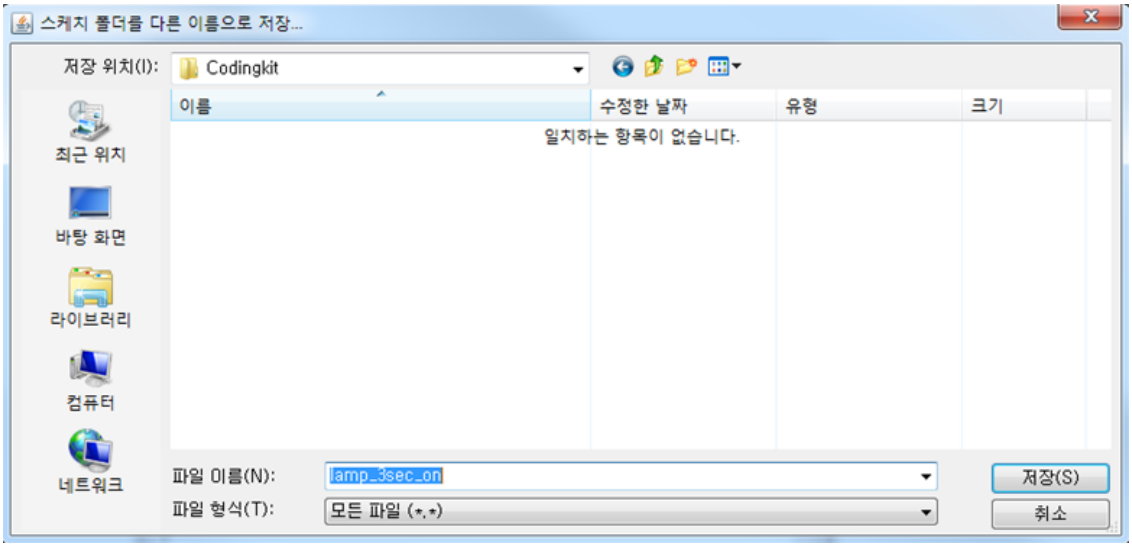
```
#define LAMP 2
#define ON 1
#define OFF 0
#define Delay(x) delay(x*1000)
```

```
void CK_InitLamp3SecOn() {
   pinMode(LAMP, OUTPUT);
}

void setup() {
  // put your setup code here, to run once:
  CK_InitLamp3SecOn();
  digitalWrite(LAMP, ON);
  Delay(3);
  digitalWrite(LAMP, OFF);
}

void loop() {
  // put your main code here, to run repeatedly:
}
```

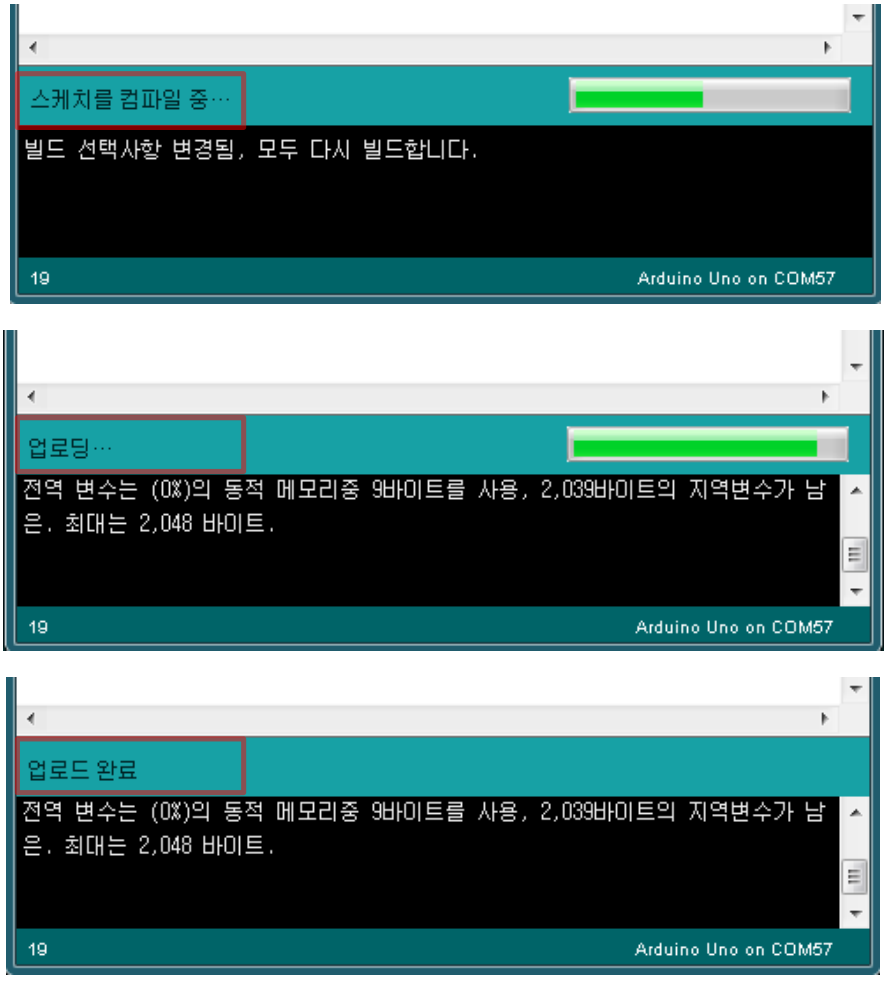
아두이노 프로그램에서 다음 그림의 붉은색 박스 표시와 같은 오른쪽 화살표 버튼을 누릅니다.



그러면 현재 코드를 저장하라는 창이 뜹니다. 적당한 폴더를 선택한 후 "lamp_3sec_on" 이라는 이름으로 저장합니다. 다른 이름으로 하셔도 좋습니다.



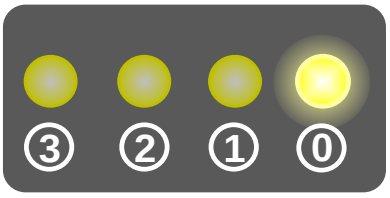
저장 버튼을 누르면 코드가 코딩키트로 전달이 됩니다. 여러분이 코딩한 작업 지시서가 컴퓨터에게 전달된 것입니다. 아두이노에서는 이것을 업로드라고 합니다. 다음 그림과 같이 아두이노 프로그램 하단에 컴파일과 업로드 진행 상태바가 보입니다.

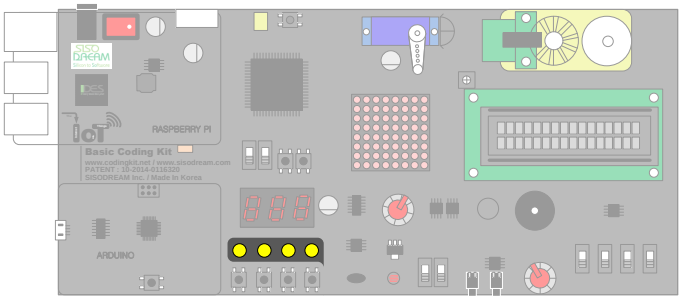


첫 화면에 컴파일 중이라는 메시지가 보이시죠? 이것이 바로 컴퓨터가 여러분이 작성한 코드를 컴파일 하

는 중인 것입니다. 중간 화면은 업로드 중이라는 것이고, 최종적으로 "업로드 완료"를 표시합니다. 이렇게 하여 여러분이 작성한 작업 지시서가 컴퓨터로 전송이 완료 되었습니다. 그럼 컴퓨터에서는 그 작업 지시서대로 일을 해야겠지요? 그럼 눈으로 직접 확인하시지요.

코딩키트에서 다음과 같이 전등이 켜진 것을 확인할 수 있습니다. 3초 후에 꺼집니다.

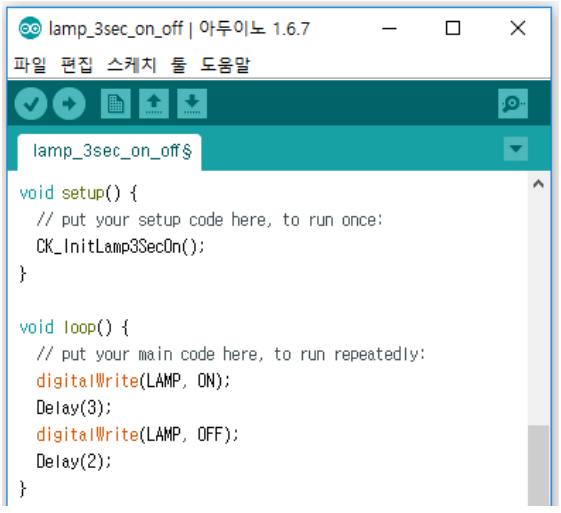




그런데, 전등이 너무 빨리 켜졌다가가 꺼져 버렸습니다. 그래서 다음과 같이 셋업 작업 지시서가 아닌 루프 작업 지시서에 쓰겠습니다. 그럼 계속 반복하겠지요? 그런데 한가지 문제는 루프를 계속 반복하면서 꺼진 후에 바로 켜지기 때문에 여러분이 꺼진 것을 확인할 수 없습니다. 그래서 꺼진 후에서 2초를 기다리는 작업 문구를 추가하겠습니다.

코드는 다음과 같습니다.

```
void setup() {
  // put your setup code here, to run once:
  CK_InitLamp3SecOn();
}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(LAMP, ON);
  Delay(3);
  digitalWrite(LAMP, OFF);
  Delay(2);
}
```



업로드 버튼()을 클릭 하시면 코드가 코딩키트로 전달되고, 코딩키트에서 3초 켜지고, 2초 꺼지는 것을 무한 반복하는 것을 보실 수 있습니다.

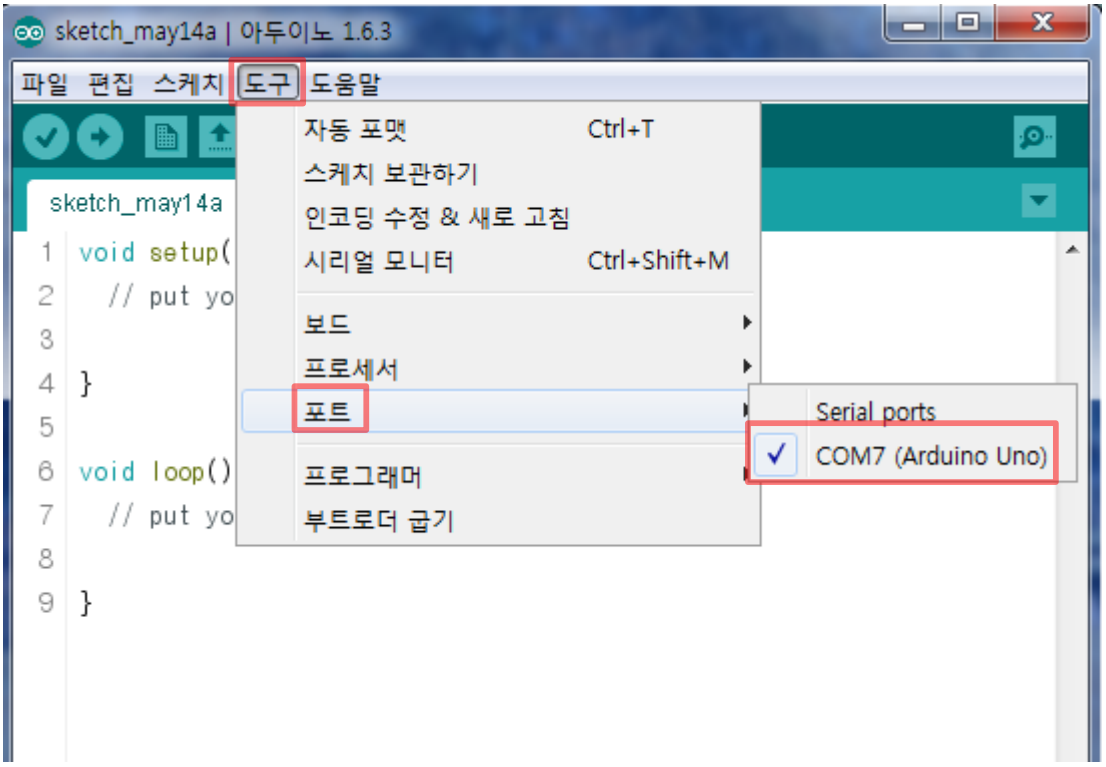
다음 코드를 추가하는 것 잊지 마세요.

```
#define LAMP 2
#define ON 1
#define OFF 0
```

```
#define Delay(x) delay(x*1000)

void CK_InitLamp3SecOn() {
  pinMode(LAMP, OUTPUT);
}
```

이렇게 하여 여러분의 첫번째 코딩이 잘 마무리 되었습니다. 매우 쉽고 재밌죠. 이번에는 첫번째라서 준비할 것들이 많았습니다. 앞으로는 더 쉽고 재미있을 것입니다. 이제부터 서두르지 않고 차근차근 잘 따라오시면 어렵지 않게 컴퓨터 코딩의 세계에 입문하실 수 있습니다. 코딩은 어렵고, 전문가만이 할 수 있다는 생각은 하지 마세요. 자신감을 갖고 천천히 따라 하시면 금방 코딩을 쉽고 재미있게 잘 하실 수 있습니다. 이제부터 정말 재미있는 코딩의 세계에 푹 빠져드실 것입니다. 잠자는 것도 잊고 코딩할 수도 있어요!! Let's Go ~~~

Note : 만약 코드가 업로드 되지 않으면 다음과 같이 "도구 → 포트" 메뉴에서 해당 포트를 연결해 줍니다.

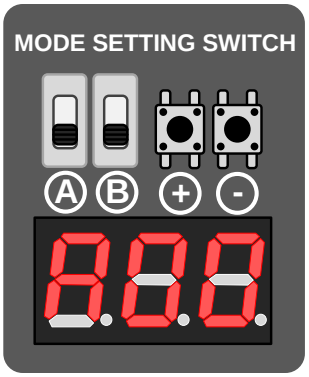


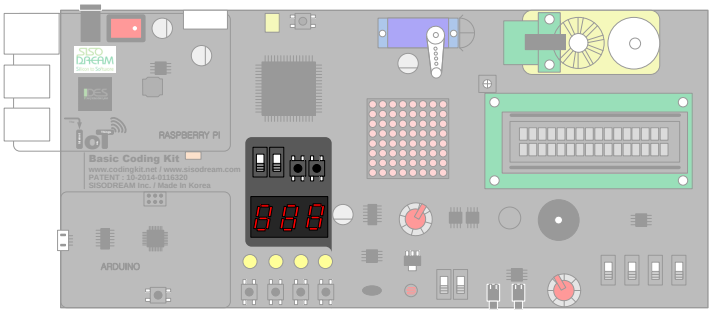
그림에서는 컴포트 7(COM7)번으로 되어 있습니다. 그런데, 이 번호는 PC 의 환경에 따라서 달라질 수 있습니다. 해당 포트를 선택해 주시면 됩니다.

Note : 코딩키트가 처음 사용하는 제품이라면 위의 예제와 같이 동작을 할 것입니다. 그렇지 않고 사용하였던 제품이라면 현재 아두이노에 다른 소프트웨어가 동작하고 있기 때문에 처음에 LED 가 켜 있을 수 있습니다. 그래도 올바르게 동작이 된다면 3초 후에는 LED 가 꺼질 것입니다. 이렇게 LED 가 꺼진 것을 확인하였다면, 스위치를 다시 한번 껐다가 켜 보십시요. 그러면 LED 가 꺼져 있다가 잠시 후에 켜질 것입니다.

그리고 3초 후에 꺼집니다.

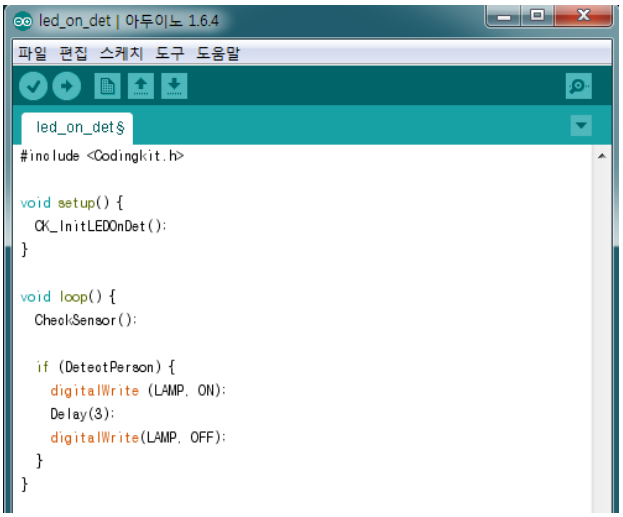
Note : 혹시 잘 되지 않는다면 다음 그림과 같이 스위치와 세븐세그먼트(7-Segment)가 설정되어 있는지를 확인합니다. 다르게 되어 있다면 스위치를 그림과 같이 모두 내리고 전원 스위치를 껐다가 켭니다. 그리고 + 버튼을 계속 눌러 A00 이 깜박이지 않고 고정되어 있게 하시고 코드를 다시 한 번 더 업로드해 주세요.





그럼 이제 아파트 복도나 계단에 있는 사람이 다가오면 켜지는 전등을 만들어 볼까요? 코드는 다음과 같습니다. 앞에서 얘기한 부분이 그대로 반영되었습니다.

```
void setup() {
  CK_InitLampOnDet();
}

void loop() {
  CheckSensor();

  if (DetectPerson) {
    digitalWrite (LAMP, ON);
    Delay(3);
    digitalWrite(LAMP, OFF);
  }
}
```



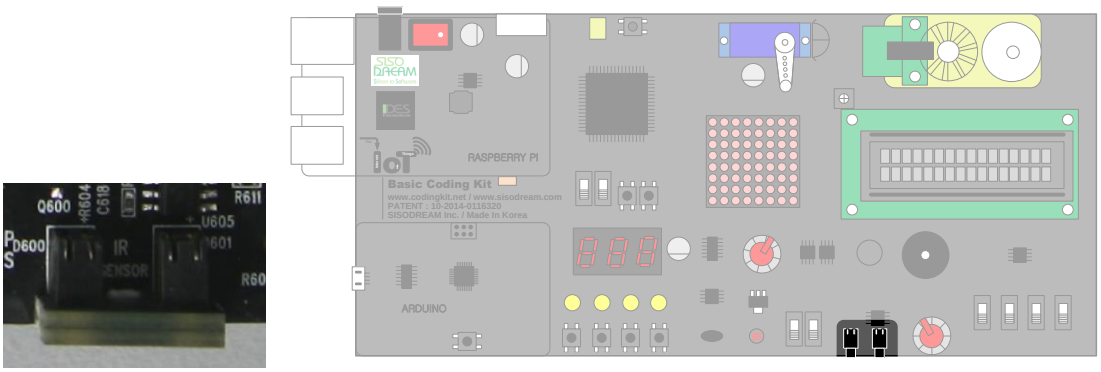
셋업 지시서에 CK_InitLEDOnDet() 라는 초기화 지시서를 포함합니다. 그리고 루프 지시서에서 CheckSensor() 라고 해 줍니다. 이것은 보통 아파트 복도나 계단에 있는 사람이 다가오면 불이 켜지는 등을 센서 등이라고 하지요. 이것은 센서라는 것이 사람이 다가오는 것을 알려주기 때문입니다. 그래서 이 코드에서도 CheckSensor() 라고 해서 "센서를 체크해서 사람이 다가오는 것을 알려 줘." 하는 뜻입니다. 그리고 만약 사람이 감지되면 DetectPerson 이라는 것으로 표시를 합니다. 그래서 DetectPerson 이 되면 전등에 불이 켜지고 3초를 유지한 후에 불이 꺼집니다.

이 코드를 업로드 버튼을 눌러 코딩키트로 전달하기 전에 다음과 같은 코드를 이 코드 위에 추가합니다.

```
#define LAMP 2
#define ON 1
#define OFF 0
#define Delay(x) delay(x*1000)

#define IR_LED A5
#define IR_DETECT A0

void CK_InitLamp3SecOn() {
    pinMode(LAMP, OUTPUT);
    pinMode(IR_LED, OUTPUT);
}

int DetectPerson;

void CK_InitLEDOnDet() {
    CK_InitLamp3SecOn();
    digitalWrite(IR_LED, ON);
    DetectPerson = 0;
}

void CheckSensor() {
    if (analogRead(IR_DETECT) > 450)
        DetectPerson = 1;
    else
        DetectPerson = 0;
}
```

이 코드는 조금 깁니다. 아직까지는 모르고 넘어가셔도 괜찮습니다. 아두이노 코딩을 공부해 가면서 차차 알아갈 것입니다.

이제 업로드 버튼을 눌러 업로드합니다. 이것이 완료되면 다음 그림과 같이 코딩키트의 센서 앞에 손을 가까이 하면 전등에 불이 들어오는 것을 확인 할 수 있습니다. 그리고 3초 후에 불이 꺼집니다.



< 예제 코드 : **3 초간 전등 켜기** >

< 코드 위치 : Coding Kit → **lamp_3sec_on >**

앞으로 위와 같이 예제 코드의 위치를 표시하겠습니다.

< 예제 코드 : **3 초간 전등 켜고 2 초간 전등 끄기** >

< 코드 위치 : Coding Kit → **lamp_3sec_on_off >**

< 예제 코드 : **물체를 감지하면 전등 켜기** >

< 코드 위치 : Coding Kit → **lamp_on_det >**

이 교재에는 다음과 같은 항목들이 포함되어 있습니다.

< 예제 코드 : >

컴퓨터 언어의 문법을 설명하거나 디바이스를 컨트롤하기 위한 예제 코드에 대한 설명입니다.

< 문법 설명 : >

컴퓨터 언어의 문법에 대해서 따로 정리합니다.

< 연습 문제 : >

직접 코딩을 해 보는 연습 문제입니다. 코딩 실력 향상에는 직접해보는 것이 최고입니다.

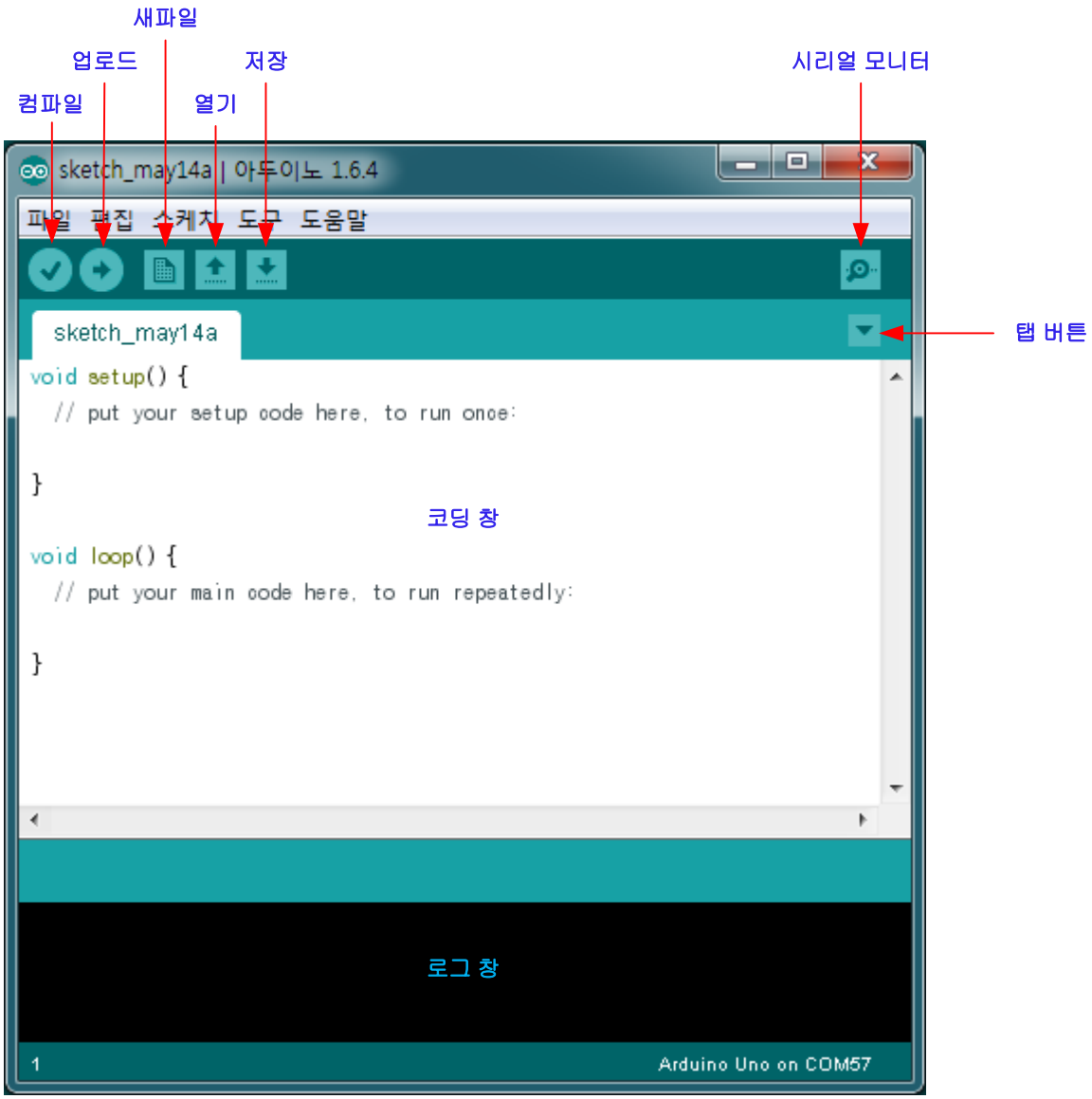
< 코드 위치 : >

코딩북의 모든 예제 및 연습 문제에 있는 원본 소스 코드에 대한 위치를 알려 줍니다.

앞으로 아두이노를 이용한 다양한 예제를 코딩키트에서 실행해 보겠습니다. 이러한 예제를 활용하여 매우 쉽게 설명하겠습니다. 부디 지치지 마시고 차근차근 따라오시기를 부탁드립니다. 여러분이 이 교재를 모두 마스터 하셨을 때 여러분의 코딩 실력은 매우 향상되어 있을 것입니다. 그리고 앞으로 C-언어 같이 좀 더 어려운 언어를 쉽게 익히실 수 있을 것입니다. 그리고 코딩키트와 같이 함으로서 LED, 모터, 센서 같은 디바이스에 한 층 더 친숙해 지실 수 있을 것입니다. 이렇게 실력을 쌓는다면 앞으로 여러분이 상위 학교로의 진학이나 취업하는데 많은 보탬이 되리라 확신합니다.

[아두이노 프로그램 더 알아보기]

현재까지 사용한 아두이노 프로그램에 대해서 간략히 알아 보겠습니다.



컴파일 : 코드에 문법적으로 오류가 없는지를 확인하고 코드를 업로드 할 수 있는 소프트웨어로 만들어 줍니다. 주로 업로드는 하지 않고 문법적으로 오류가 없는 지를 확인하는 용도로 주로 사용됩니다.

업로드 : 컴파일을 하고 컴파일이 완료되면 코드를 업로드 합니다. 이 버튼은 컴파일과 업로드를 동시에 수행합니다.

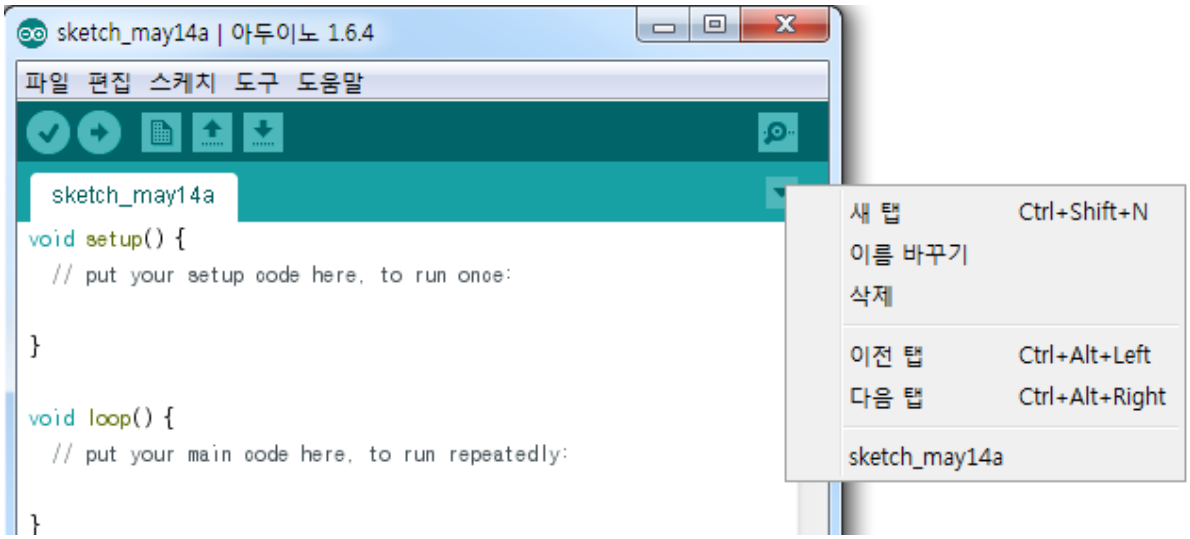
새파일 : 아두이노 프로그램 새로운 창이 나타나면서 새 파일이 열립니다. 즉, 새로운 코딩을 할 수 있도록 합니다.

열기 : 기존의 아두이노 코드 파일을 엽니다. 그리고 빠르게 예제 파일들을 읽어 올 수도 있습니다.

저장 : 현재 코드를 저장합니다.

시리얼 모니터 : 아두이노 보드와 통신을 합니다. 나중에 관련 예제를 다룰 때 더 자세히 다루겠습니다. 예제를 보시면 훨씬 더 이해가 빠를 것입니다.

탭 버튼 : 탭 버튼을 누르면 그림과 같이 팝업 메뉴가 보입니다. 여기서 새 탭은 새로운 이름으로 탭이 하나 더 추가가 됩니다. 이렇게 추가된 탭에는 이전 탭에 있던 내용을 옮겨 오거나 새로운 내용을 쓸 수 있습니다. 중요한 것은 이 모든 것이 다 하나의 파일처럼 컴파일 된다는 것입니다. 즉, 하나의 파일이 너무 길어 읽기가 힘들거나 할 때 나누어 쓸 목적으로 만든 기능입니다.



코딩 창 : 제목 그대로 코딩을 하는 창입니다.

로그 창 : 컴파일 할 때 에러를 표시하거나, 아두이노 프로그램에서 사용자에게 무엇인가 알릴 것이 있을 때 이용하는 창 입니다.

< 문법 설명 : **코드 주석** >

코딩에서는 보통 주석(Comment)이라고 부르는 것이 있습니다. 이것은 실제로 수행되는 코드는 아니고 단지 코드를 보는 사람을 위한 설명, 주석 같은 것입니다. 이 컴퓨터 언어라는 것이 작업 지시서와 같다고 했습니다. 그런데, 그 작업 지시서는 사람이 만들고 수행은 컴퓨터가 합니다. 그래서 이 작업 지시서를 나중에 수정하거나 내용을 추가하는 일들은 사람이 해야 합니다. 그 때 이전의 작업 지시서에 대한 설명이나 주석이 잘 되어 있으면 훨씬 더 알아보기 쉽고 수정, 추가 작업이 용이 하겠지요. 그래서 코딩할 때 주석을 잘 써 두는 습관을 들이면 좋습니다.

주석은 문법적으로 다음과 같이 표시 합니다.

```
void setup() {
  // put your setup code here, to run once:
}
```

바로 아두이노 프로그램을 실행 시키면 위의 붉은색 부분처럼 "//" 기호 이후부터 그 줄의 끝까지가 주석인 것입니다. 다음과 같이 /* 와 */ 로 묶는 것도 주석을 만드는 또 다른 방법입니다.

```
void setup() {
  /* put your setup code here, to run once: */
}
```

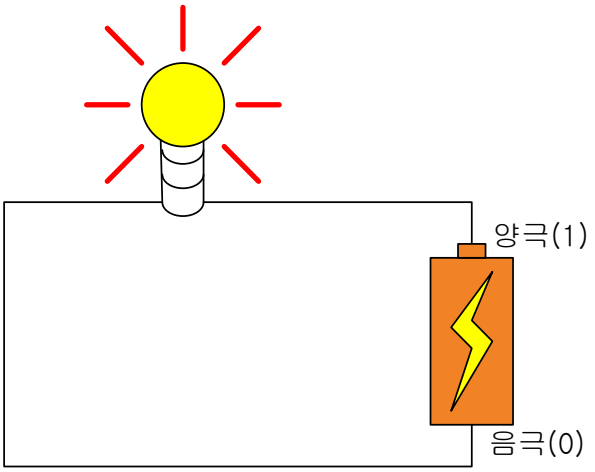
두 가지 주석 방법이 조금 다른 용도로 사용됩니다. "//" 기호는 보통 한 줄 안에서 처음부터 혹은 중간부터 끝까지 주석 처리할 때 많이 사용됩니다. /* 와 */ 은 한 줄 안에서는 중간 부분을 주석 처리할 때 사용되거나 또는 여러 줄을 한꺼번에 주석 처리할 때 사용됩니다.

[코딩으로 디바이스 컨트롤 하기]

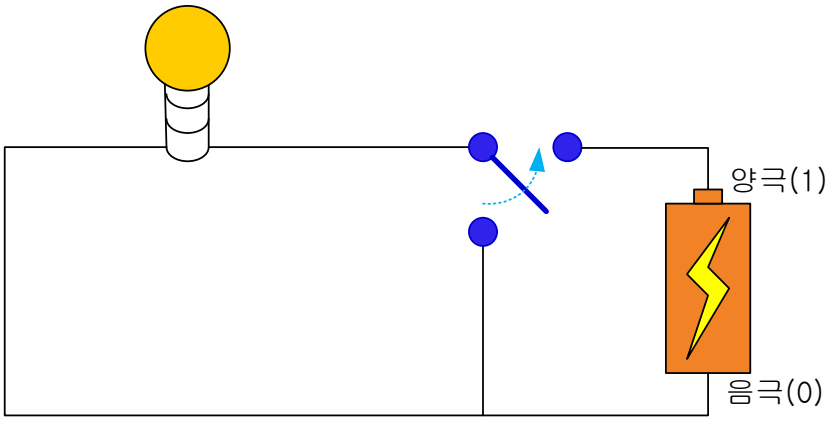
아두이노와 같은 소형 컴퓨터는 위에서 예로 든, 사람이 다가오면 켜지는 센서등과 같이 간단한 기능을 끊임 없이, 전원만 연결되어 있다면 1년 365일 수행하는 그런 제품에 많이 사용됩니다. 그래서 아두이노 프로그램에 루프 작업 지시서 부분이 있는 것입니다. 이 루프는 전원을 끄지 않으면 언제까지고 계속해서 반복 수행이 됩니다. 위의 프로그램을 업로드해 두고 아직 코딩키트를 끄지 않았다면 지금 센서에 손을 가까이 해 보십시요. 또 전등에 불이 들어 오지요? 참, 성실한 놈입니다.

그러면 이제 이 아두이노 같은 컴퓨터가 어떻게 전등과 같은 디바이스들을 컨트롤하는지에 대해서 알아보도록 하겠습니다.

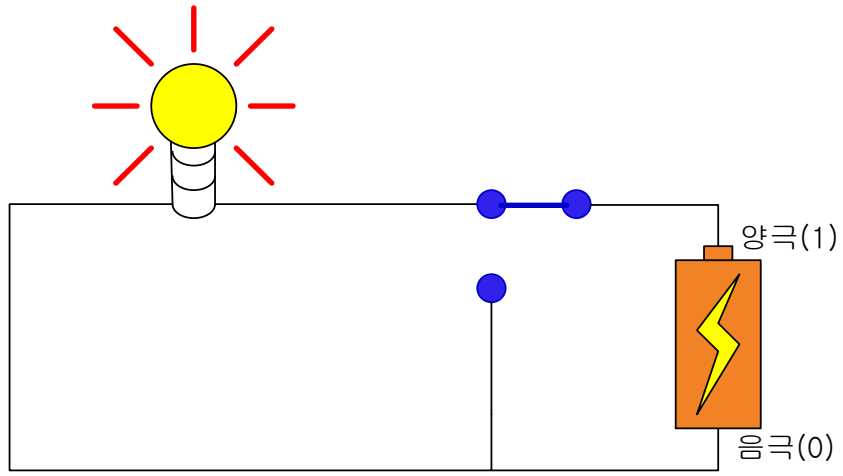
다음은 건전지와 꼬마 전구가 연결되어 있는 그림입니다. 그림과 같이 건전지의 양극과 음극을 전구의 양극과 음극에 연결하면 전구가 켜 집니다. 여기서 건전지의 양극과 음극에서 나오는 전기 값을 숫자로 표시하겠습니다. 양극의 전기 값은 1 로, 음극의 전기 값은 0 으로 표시합니다. 그러면 건전지에서 전구의 양극에 1 값을 주고 음극에 0 값을 주면 (연결하면) 전구는 켜집니다.



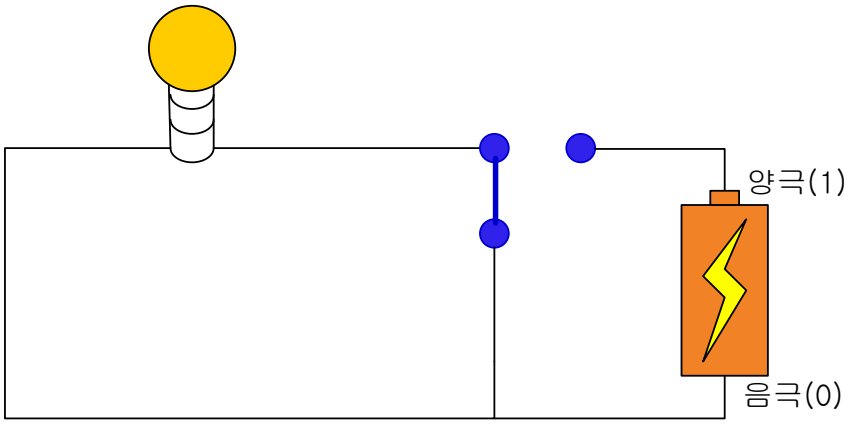
그런데 다음 그림과 같이 꼬마 전구와 건전지 사이에 스위치를 연결합니다. 이 스위치는 건전지에서 1 이 나오는 곳과 0 이 나오는 곳을 각각 연결할 수 있습니다.



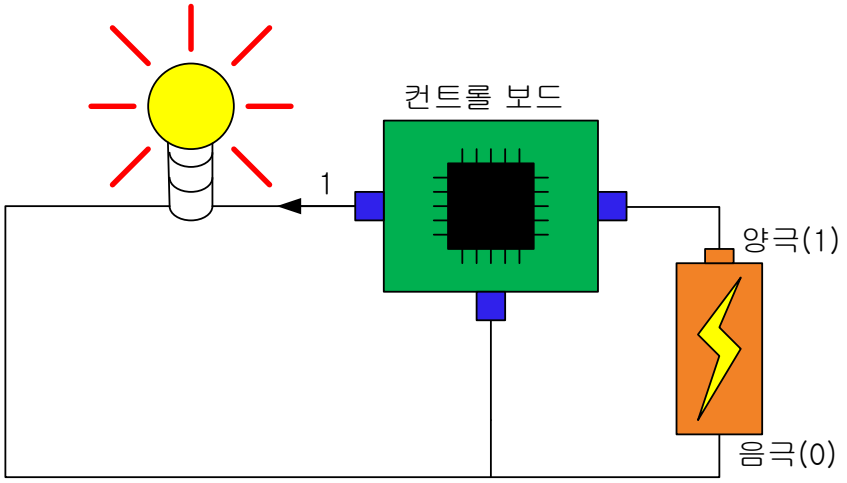
다음 그림과 같이 이 스위치를 건전지의 1 이 나오는 곳에 연결하면 전구는 켜집니다.

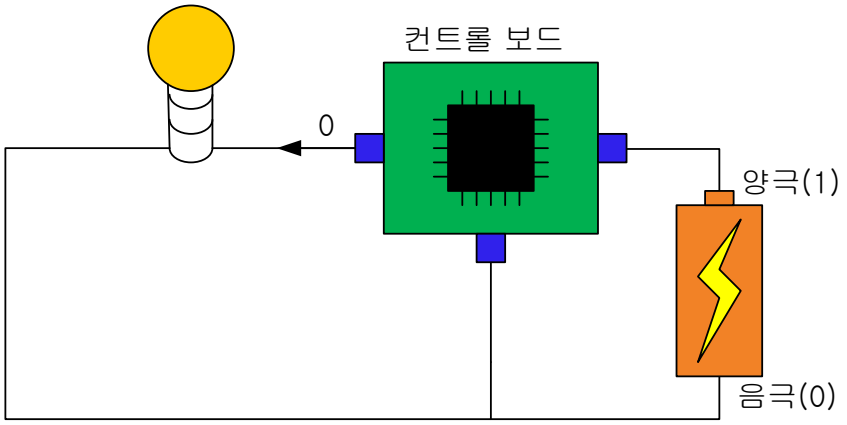


다음 그림과 같이 이 스위치를 건전지의 0 이 나오는 곳에 연결하면 전구는 꺼집니다.

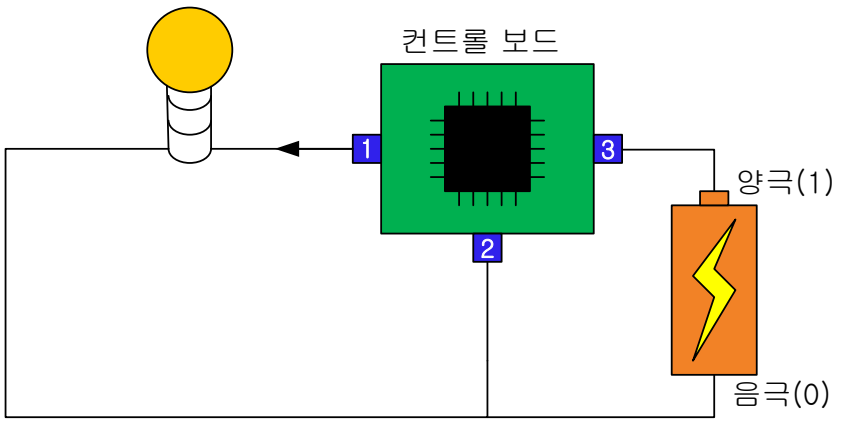


이 스위치 대신 다음과 같이 컨트롤 보드를 연결합니다. 이 컨트롤 보드는 앞에서 본 아두이노와 같이 여러분이 코딩한 프로그램으로 컨트롤 할 수 있는 보드입니다. 이 프로그램이 위의 스위치처럼 건전지의 1 값, 혹은 0 값을 전구에 연결합니다. 아래 그림과 같이 1 값을 꼬마 전구에 주면 전구는 켜집니다. 0 값을 주면 전구가 꺼집니다.



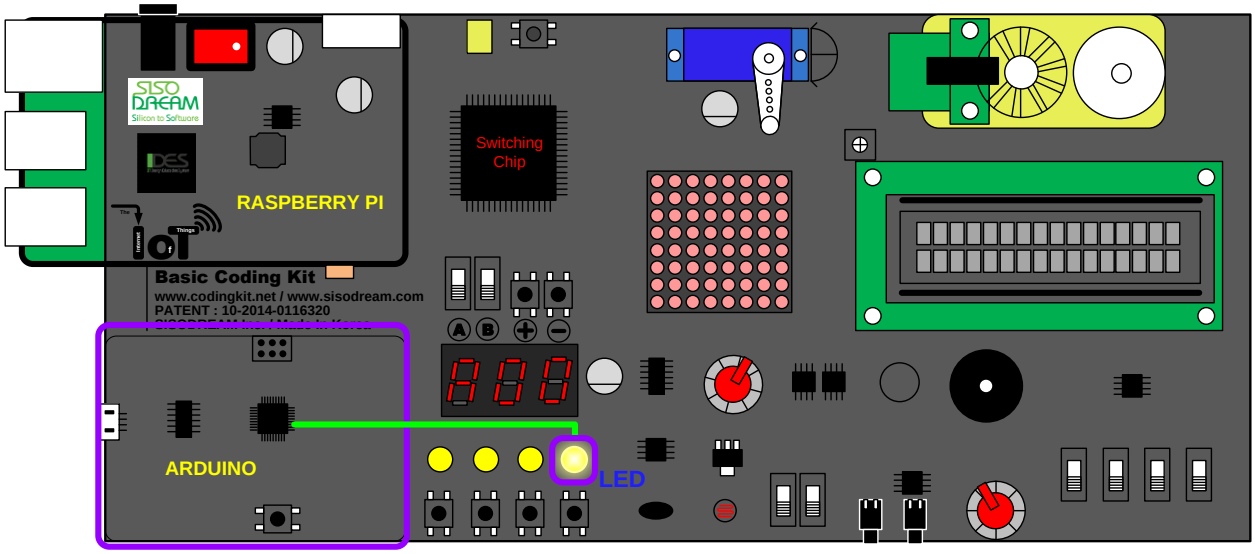


그림의 컨트롤 보드에서 전구 또는 건전지와 연결된 파란색 부분을 핀이라고 합니다. 이 핀은 컨트롤 보드와 컨트롤 보드가 컨트롤 할 수 있는 부품들이 연결되는 통로 역할을 합니다. 다음 그림과 같이 컨트롤 보드는 자신의 핀들에 번호를 붙입니다.



컨트롤 보드의 코딩을 할 때 이 핀 번호에 1 혹은 0 값을 써서 연결되어 있는 부품에 값을 전달합니다. 또는 연결되어 있는 부품에서 전달하는 값을 핀으로 입력 받습니다. 위 그림에서 컨트롤 보드는 전등으로 값을 주는 것이고, 건전지로부터 값을 받는 것입니다.

이제 컨트롤 보드를 아두이노로 바꾸고 꼬마 전구는 LED 로 바꾸겠습니다. 그러면 다음 그림과 같이 코딩키트가 만들어 지고, 여러분이 아두이노 프로그램에서 코딩을 하여 코딩키트를 컨트롤 할 수 있는 것입니다.



아두이노는 코딩키트의 왼쪽 하단의 보라색 박스 안에 있습니다. 이 아두이노 핀 중 하나에 LED 가 연결되어 있습니다. 이것을 연두색 선으로 표시하였습니다. 좀 더 설명드리면, 코딩키트에서는 아두이노가 아답터에 연결되어 있습니다. 이것은 아두이노가 건전지에 연결되어 있는 것과 같은 것으로 생각하시면 됩니다. 아두이노는 LED 를 켜기 위해서 아답터로부터 받은 1 을 LED 로 보냅니다. 그러면 LED 가 켜지는 것입니다. 그런데, 아두이노로 LED 를 켜기 위해서는 스위치가 건전지에서 받은 1 을 꼬마 전구에 전달하는 것과 같이 아두이노는 아답터에서 받은 1을 LED 에 전달하는 코딩을 해야 합니다. 그 코드는 다음과 같습니다.

```
(LED, 1)
```

반대로 LED 를 끄기 위해서는 다음과 같이 0 을 전달합니다.

```
(LED, 0)
```

그런데, 아두이노는 아답터에서 받은 1 과 0 을 출력할 수 있는 핀이 여러 개 있습니다. 이 중 LED는 2번 핀에 연결되어 있습니다. 그래서 다음과 같이 LED 가 몇 번 핀이라는 것을 알려 줍니다.

```
LED = 2
```

여기서 또 한가지 정의해 주어야 할 것이 아두이노에서 1 또는 0 값을 LED 에 써 준다는 의미로 다음과 같이 씁니다.

```
digitalWrite(LED, 1)
```

여기서 그냥 "Write" 가 아니고 "digitalWrite" 입니다. 디지털이라는 수식어가 붙는데, 이것은 전달해 줄

수 있는 값이 1 과 0 으로 한정되어 있어서 그럽니다. 즉, LED 에는 1 혹은 0 만을 쓸 수 있습니다. 이것과 관련하여 나중에 디지털과 아날로그 신호에 대해서 자세히 설명하겠습니다.

그리고 컨트롤 보드에서 1 혹은 0 값이 핀으로 나가서 LED 를 켜거나 끕니다. 즉, 1 또는 0 값이 핀으로 나간다는 의미로 다음과 같이 LED 는 Output 이라고 정의해 둡니다. 출력이라고 하지요.

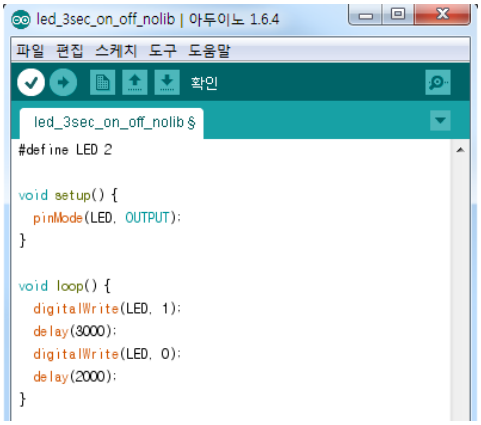
```
(LED, OUTPUT)
```

이렇게 핀이 입력으로 쓰이는지 출력으로 쓰이는지가 전기 회로에서는 매우 중요합니다. 그래서 어떤 핀을 사용하기 전에는 반드시 입력인지 출력인지를 정의 합니다. 이 정의는 핀의 모드를 정하는 것이라고 해서 pinMode 라고 쓰고 핀 이름을 쓰고, 입력 또는 출력을 명기합니다. 정확히는 다음과 같이 코딩합니다.

```
pinMode(LED, OUTPUT);
```

그래서 이상의 내용들을 바탕으로 이전에 했던 전등을 3초간 켜고 2초가 끄는 예제와 비슷한 LED 를 3초간 켜고 2초간 끄는 예제를 해 보겠습니다.

```
#define LED 2

void setup() {
  pinMode(LED, OUTPUT);
}

void loop() {
  digitalWrite(LED, 1);
  delay(3000);
  digitalWrite(LED, 0);
  delay(2000);
}
```



몇 가지 특이 점을 살펴 보면 LED 가 2번 핀이라는 의미로 "#define LED 2" 라고 했습니다. 그리고 또 전에는 Delay(3), Delay(2) 하던 것을 delay(3000), delay(2000)으로 사용하였습니다. 이것은 Delay 는 괄호 안의 수가 1 초 단위이고 delay 는 괄호 안 수의 단위가 1000 분의 1 초이기 때문입니다. 그래서 delay(3000) 과 Delay(3) 은 똑같이 3 초의 시간을 기다리는 것입니다.

그리고 이 코드에서는 이전에 있었던 "#include <Codingkit.h>" 가 빠졌습니다. 이것은 코딩키트를 사용하기 위하여 미리 준비해 둔 "Codingkit.h" 와 "Codingkit.cpp" 파일을 가지고 와라 하는 뜻입니다. 그런데, 이번에는 "#include <Codingkit.h>" 가 빠졌다는 것은 미리 준비된 코드 없이 여러분 스스로 모든 코드를 작성했다는 뜻입니다. 진정으로 여러분이 처음 작성한 아두이노 코드가 되겠습니다.

이제 업로드 버튼을 눌러 여러분이 코딩한 코드의 동작을 코딩키트로 확인해 보십시요.

여기서 한 가지 더 알아 두셔야 할 것이, 이렇게 어떤 핀에 1 혹은 0 값을 바로 전달해 줄 수 있는 핀을

GPIO (General Purpose Input Output) 라고 합니다.

이번 장에서는 여러분의 코딩이 어떻게 하드웨어 디바이스를 컨트롤 하는지에 관해서 공부했습니다. 다음장 부터는 아두이노 프로그램 언어의 문법을 배우면서 본격적으로 컴퓨터 언어의 세계로 들어 가겠습니다. 앞으로도 열심히 해 줄 것을 부탁드립니다.

< 예제 코드 : **LED 3 초간 켜기** >

< 코드 위치 : Coding Kit → **led_3sec_on_off** >

< 문법 설명 : **#define** >

"#define" 은 무엇을 무엇으로 정의하는 것입니다. 다음과 같이 쓰면 A 를 B 라고 정의 하는 것입니다.

```
#define A B
```

그리고 실제로도 이렇게 쓴 다음부터는 A 는 B 로 모두 바뀝니다. 실제로 컴퓨터가 컴파일 할 때 A로 되어 있는 것은 모두 B 로 바꾸어 버립니다. 그래서 B 를 쓰고 싶은 곳에 A 를 쓰면 B가 됩니다. 그러면, 그냥 B 를 쓰면 되지 굳이 왜 A 로 정의해서 바꿀까요? 그것은 사람들이 보고 이해하기 쉽게 하려고 하는 것입니다. 이런 것을 가독성이 좋다고 합니다. 이것은 앞에 나온 코드들을 보면 쉽게 이해할 수 있을 것입니다.

```
#define LED 2

void setup() {
  pinMode(LED, OUTPUT);
}

void loop() {
  digitalWrite(LED, 1);
  delay(3000);
  digitalWrite(LED, 0);
  delay(2000);
}
```

위의 코드를 "#define LED 2" 를 빼고 해 보겠습니다. 그럼 LED 라고 되어 있는 부분을 모두 숫자 2로 바꾸어야 합니다.

```
void setup() {
  pinMode(2, OUTPUT);
}

void loop() {
  digitalWrite(2, 1);
```

```
  delay(3000);
  digitalWrite(2, 0);
  delay(2000);
}
```

이렇게 코딩하여 아두이노 프로그램에서 코드를 업로드해 보겠습니다. 어떻습니까? 코딩키트에서 잘 동작하지요. 하지만, 보기에는 좋지 않습니다. 단순히 숫자 2 라고 하는 것보다는 LED 라고 하면 "아! LED 에 1 을 쓰는구나!" 하고 이 코드를 읽는 사람이 쉽게 코드를 이해할 수 있습니다. 그러면 여기서 LED 를 켜는 것이니깐, 다음과 같이 1 을 ON 으로 정의하고, 0 을 OFF 로 정의하는 것은 어떨까요?

```
#define LED 2
#define ON 1
#define OFF 0

void setup() {
  pinMode(LED, OUTPUT);
}

void loop() {
  digitalWrite(LED, ON);
  delay(3000);
  digitalWrite(LED, OFF);
  delay(2000);
}
```

어때요? 훨씬 더 보기가 좋지요. 보기 좋은 떡이 먹기도 좋다고, 이렇게 해 두면 보기도 좋지만 또 한가지 중요한 이점이 있습니다. 만약 여러분이 좋은 직장에 취직을 해서 멋지게 LED 대신 숫자 2 를 쓴 코드를 컴파일하여 소프트웨어로 만들어 직장 상사에게 보여 줬습니다. 그런데, 상사가 "어! 나는 2 번 핀에 연결된 LED 를 켜는 것 보다는 3 번 핀에 연결된 LED 를 켜고 싶은데!" 그랬습니다. 그러면 여러분은 코드를 어떻게 수정할 겁니까? 2 번이라고 쓰인 곳을 모두 찾아가 3 번으로 바꾸어야겠지요.

```
void setup() {
  pinMode(3, OUTPUT);
}

void loop() {
  digitalWrite(3, 1);
  delay(3000);
  digitalWrite(3, 0);
  delay(2000);
}
```

하지만, 숫자 2를 LED 라고 한 것은 어떻게 하면 될까요? 간단합니다. "#define LED 2" 에서 숫자 2 에서 3 으로 바꾸면 됩니다.

```
#define LED 3
```

```
void setup() {
  pinMode(LED, OUTPUT);
}

void loop() {
  digitalWrite(LED, 1);
  delay(3000);
  digitalWrite(LED, 0);
  delay(2000);
}
```

어때요? 정말 멋지죠? 이렇게 "#define" 은 정말 유용하게 쓰입니다. 잘 알아 두셨다가 요긴하게 사용하세요.

한 가지 더! 여기서 끝이 아닙니다. 또 있습니다. 코드를 보면 delay(3000), delay(2000) 이것이 보이지요. 우리는 3초, 2초 이렇게 보기 쉽게 하고 싶은데, 방법이 없을까요? 왜 없겠습니까? 이럴 때도 #define 을 이용한답니다. 일단 코드를 한 번 볼까요?

```
#define LED 2
#define ON 1
#define OFF 0
#define Delay(x) delay(x*1000)

void setup() {
  pinMode(LED, OUTPUT);
}

void loop() {
  digitalWrite(LED, ON);
  Delay(3);
  digitalWrite(LED, OFF);
  Delay(2);
}
```

붉은색 코드 부분 중 "#define Delay(x) delay(x*1000)" 는 ('*' 표시는 곱하기를 의미합니다.) Delay의 괄호 안의 값에 곱하기 1000 을 하라는 뜻입니다. 그래서 Delay(3) 은 delay(3*1000) 과 같습니다.

"#define Delay(x) delay(x*1000)" 에서 x 는 이 값을 곱하기 1000 하라는 의미로 사용된 것입니다. 주로 x 를 사용하는데, 다른 문자를 사용하셔도 좋습니다.

한가지 더 말씀 드릴 것은 #define 구문에서 B 를 A 로 정의하는 "#define A B" 에서 A는 주로 대문자로 써 줍니다. 그렇게 하면 코드를 볼 때 "아! 이것은 #define 한 것이구나!" 하고 직감적으로 알 수 있습니다. 그래서 "#define led 2" 보다는 "#define LED 2" 해 주는 것이 좋습니다. "#define Delay(x)" 도 "#define DELAY(x)" 하는 더 좋을 수도 있지만, Delay(x) 해도 문법적으로 틀리지 않습니다.

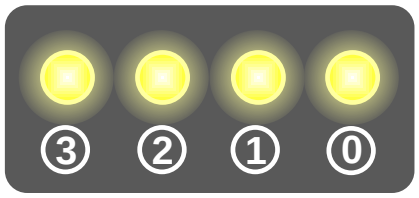
#define 구문은 자주 사용되는 매우 유용한 구문입니다.

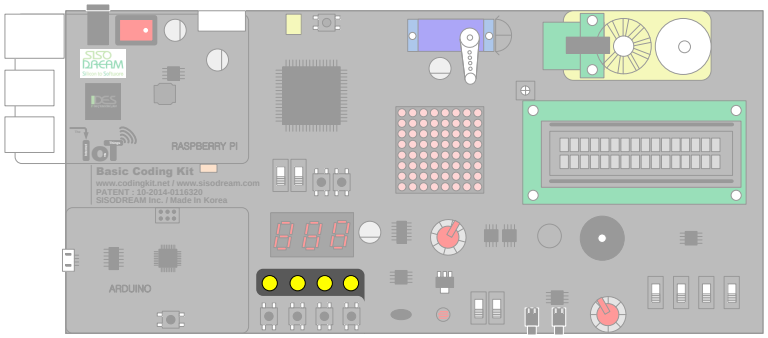
< 연습 문제 : **LED 4 개 1 초 간격으로 깜박이기** >

코딩키트에 있는 LED 4 개를 모두 1초 동안 켜고, 1초 동안 끄는 코딩을 해 봅니다. 즉, LED 모두 1초 간격으로 깜박이는 예제입니다. LED 각각의 핀 번호는 다음과 같습니다.

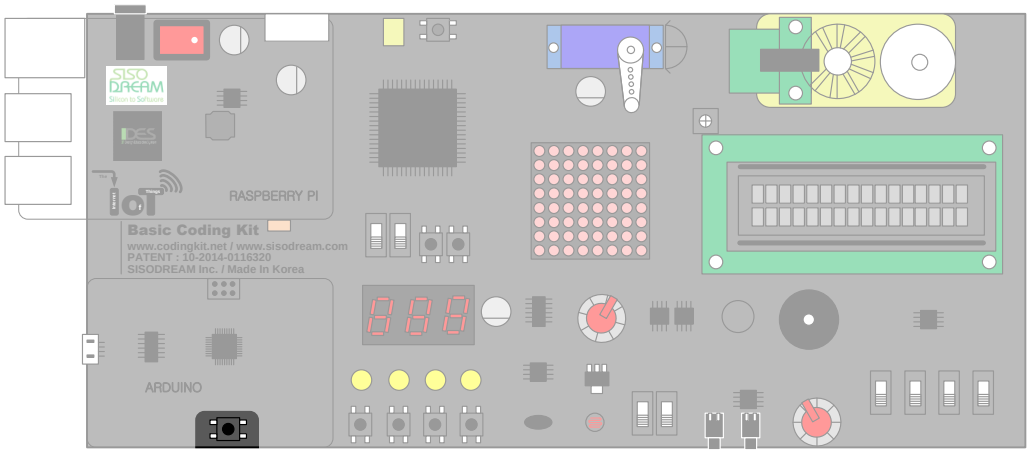
```
#define LED_0 2
#define LED_1 3
#define LED_2 7
#define LED_3 8
```

< 코드 위치 : Coding Kit → **led4_on_off >**





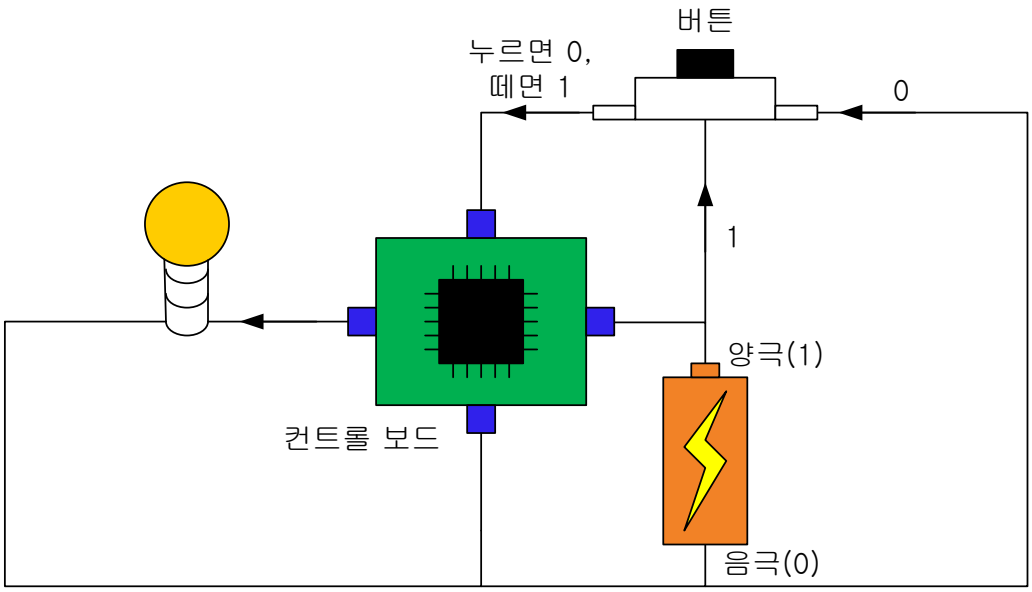
Note : 코딩키트에는 다음 위치에 아두이노 리셋 스위치가 있습니다.



여러분이 아두이노 코드를 다시 실행하고 싶을 때 눌러주시면 처음부터 코드를 다시 실행합니다. 즉 setup() 함수 처음부터 코드를 다시 실행합니다.

[신호 입력 받아 출력하기]

이번에는 버튼을 누르면 LED 가 켜지는 코딩을 해 보겠습니다. 버튼은 다음 그림과 같이 연결되어 있습니다.



버튼은 건전지의 1 혹은 0 값을 컨트롤 보드에 전달합니다. 버튼을 누르면 0 값이 컨트롤 보드에 전달되고, 떼면 1 값이 컨트롤 보드에 전달됩니다. 컨트롤 보드 입장에서는 버튼의 출력은 자신에게 들어오는 입력입니다. 그래서 핀 모드는 다음과 같이 코딩합니다.

```
pinMode(BUTTON, INPUT);
```

버튼과 연결된 핀의 값을 읽는 것은 다음과 같이 코딩합니다. 즉, 입력되는 값을 읽는 것입니다.

```
digitalRead(BUTTON);
```

이것은 digitalWrite 가 어떤 핀에 값을 쓰라는 것과는 반대로 어떤 핀의 값을 읽어 오라는 뜻입니다.

이 값에 따라서 LED 를 켤지 끌지를 결정해야 합니다. 그것은 다음과 같이 코딩합니다.

```
if (digitalRead(BUTTON) == BUTTON_PRESS)
  digitalWrite(LED, ON);
else
  digitalWrite(LED, OFF);
```

"if (digitalRead(BUTTON) == BUTTON_PRESS)" 는 BUTTON 핀에서 읽은 값이 버튼을 누른 값과 같냐? 하는 의미입니다. **여기서 "같냐?" 라는 뜻의 if 뒤의 괄호 안의 등호(=) 표시가 2개의 등호(==)라는 것을 꼭 기억해 주세요.** if 는 "만약 괄호 안의 값이 맞다면 다음에 나오는 문장을 수행해라" 하는 뜻입니다. 그래서 우리의 코드 "if (digitalRead(BUTTON) == BUTTON_PRESS)" 은 "버튼에 연결된 핀의 값을 읽어 보니 버튼이 눌린 값과 같다면" 의 뜻입니다. 같다면 다음 코드 "digitalWrite(LED, ON)"과 같이 LED 를 켜 줍니

다.

여기서 else 는 "if 문이 맞지 않다면" 입니다. 즉, "버튼에 연결되어 있는 핀의 값을 읽어 보니 버튼이 눌린 값과 다르면" 입니다. 그래서 else 는 다르면 다음에 나오는 문장을 수행해라 입니다. 여기서는 "digitalWrite(LED, OFF)" 입니다. 즉, LED 를 끄라는 의미입니다. "if ... else ..." 구문에 대해서는 나중에 또 설명하겠습니다.

이렇게 하면 우리가 원하는 "버튼이 눌리면 LED 를 켜는" 코드가 완성된 것입니다. 여기에 다음과 같이 몇 가지를 정의해 주면 됩니다.

```
#define LED 2
#define BUTTON 12
```

LED 의 핀 번호는 2번이고, 버튼의 핀 번호는 12번입니다.

버튼이 눌리면 0 값을 전달하고 떼면 1 값을 전달하기 때문에 다음과 같이 정의 합니다.

```
#define BUTTON_PRESS 0
```

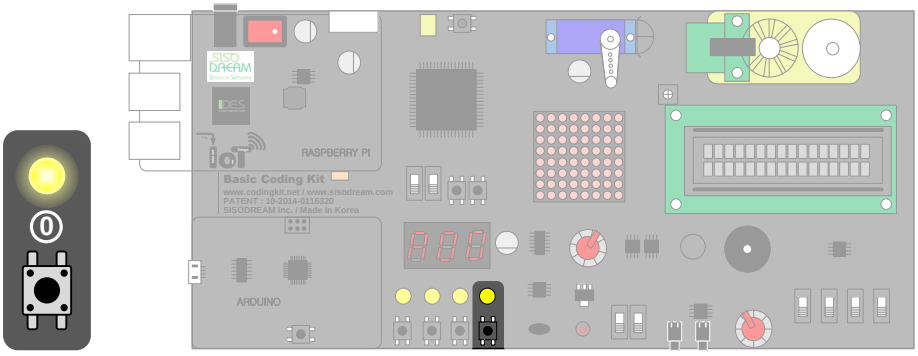
이상의 내용을 바탕으로 코딩하면 다음과 같습니다.

```
#define LED 2
#define BUTTON 12
#define BUTTON_PRESS 0

#define ON 1
#define OFF 0

void setup() {
  pinMode(LED, OUTPUT);
  pinMode(BUTTON, INPUT);
}

void loop() {
  if (digitalRead(BUTTON) == BUTTON_PRESS)
    digitalWrite(LED, ON);
  else
    digitalWrite(LED, OFF);
}
```

< 예제 코드 : **버튼이 눌리면 LED 켜기** >

< 코드 위치 : Coding Kit → **led_on_off_but >**

코딩키트에서 수행해 보십시요. 버튼을 눌러 LED 가 잘 켜지는지 확인해 보세요.



< 문법 설명 : **코드 묶음 (범위)** >

버튼을 누르면 LED 4개가 모두 켜지는 예제를 해 보겠습니다. 코드는 LED 한 개를 켜고 끄는 것과 매우 비슷합니다.

```
#define LED_0 2
#define LED_1 3
#define LED_2 7
#define LED_3 8

#define BUTTON 12
#define BUTTON_PRESS 0
#define ON 1
#define OFF 0
#define Delay(x) delay(x*1000)

void setup() {
  pinMode(LED_0, OUTPUT);
  pinMode(LED_1, OUTPUT);
  pinMode(LED_2, OUTPUT);
  pinMode(LED_3, OUTPUT);
  pinMode(BUTTON, INPUT);
}

void loop() {
  if (digitalRead(BUTTON) == BUTTON_PRESS) {
    digitalWrite(LED_0, ON);
    digitalWrite(LED_1, ON);
    digitalWrite(LED_2, ON);
    digitalWrite(LED_3, ON);
  }
  else {
    digitalWrite(LED_0, OFF);
    digitalWrite(LED_1, OFF);
    digitalWrite(LED_2, OFF);
    digitalWrite(LED_3, OFF);
  }
}
```

코드 중 붉은색 부분은 중괄호로 묶여 있습니다. 이것은 그 위의 if 뒤의 괄호 안의 값이 맞으면 중괄호 안의 모든 구문이 다 수행된다는 의미입니다. 이렇게 중괄호를 사용하면 어떤 구문들이 같은 조건에서 수행된다는 의미입니다. else 이후의 코드에서도 중괄호가 사용되었습니다. 크게는 setup() 부분과 loop() 부분에도 중괄호가 사용되었습니다.

< 예제 코드 : **버튼이 눌리면 LED 4 개 켜기** >

< 코드 위치 : Coding Kit → **led4_on_off_but >**

< 문법 설명 : **변수와 변수 타입** >

이제 컴퓨터 언어에서 매우 중요한 개념인 변수에 대해서 알아 보겠습니다. 수학 시간에 변수에 대해서 배웠죠? 그것과 같은 것입니다. 어떤 값을 저장해 두는 공간인 것입니다. 그런데 보통 이 공간을 어떤 문자로 표현하지요. 수학 시간에 배운 변수도 보통 x, y 같은 문자를 많이 쓰잖아요. 이전 예제 코드 중에 다음 코드를 한 번 볼까요?

```
if (digitalRead(BUTTON) == BUTTON_PRESS)
  digitalWrite(LED, ON);
else
  digitalWrite(LED, OFF);
```

여기서 digitalRead(BUTTON) 에서 전달되는 값을 변수에 저장해서 사용해 보도록 하겠습니다.

```
int button_value;
button_value = digitalRead(BUTTON);

if (button_value == BUTTON_PRESS)
  digitalWrite(LED, ON);
else
  digitalWrite(LED, OFF);
```

먼저 button_value 를 다음과 같이 "int button_value" 라고 선언 합니다. 여기서 int 라는 것은 button_value 라는 변수가 정수라는 뜻입니다. int 는 영문으로 정수라는 뜻의 integer 의 약자 입니다. int 와 같은 것들을 변수 타입이라고 합니다. 변수 타입을 정하는 것은 해당 변수의 범위가 어떻게 되는 지를 컴퓨터에게 미리 알려 주기 위한 것이지요. 이렇게 해야 하는 이유는 컴퓨터는 변수를 선언하면 그 변수가 저장되어야 할 위치와 공간의 크기를 정합니다. 그래야 그 곳에 변수를 저장할 수 있기 때문입니다.

좀 더 구체적으로 얘기를 하면 (어려우면 그냥 지나치셔도 됩니다.) 컴퓨터에는 메모리가 있습니다. 컴퓨터에서는 코드와 정보 모두를 메모리에 저장합니다. 그런데, 코드에서 변수 선언을 하면 컴퓨터는 그 변수가 저장될 위치를 정합니다. 이 때 변수의 크기도 같이 정해 줍니다. 그런데, 그 변수가 만약 0 ~ 100 까지만 변한다고 하면 그 크기만큼만 메모리 공간을 잡아 주면 됩니다. 쓸 때 없이 101 ~ 10000 까지도 저장할 수 있는 큰 공간을 잡을 필요는 없는 것입니다. 그래서 이렇게 변수의 타입을 정해 주면 컴퓨터에서는 변수 타입에 해당 하는 크기의 메모리 공간만을 사용합니다. 알뜰하게 메모리 공간을 사용할 수 있는 것입니다.

그래서 변수 타입을 정해 주었습니다. 그 변수의 타입보다 더 큰 값을 그 변수에 대입하면 컴퓨터에서는

컴파일 할 때 에러를 내면서 컴파일 할 수 없다고 합니다. 그래서 변수를 선언할 때는 주의해서 선언해야 합니다. 하지만, 변수가 어떤 크기를 가지는지를 잘 모르거나 메모리 공간을 절약할 필요가 없으면 보통 int 로 선언해 줍니다. int 는 -32768 에서 32767 까지 충분히 큰 값의 범위를 가지기 때문에 여러분이 사용하는 대부분의 값은 충분히 저장할 수 있습니다.

변수 타입은 다음과 같은 종류가 있습니다.

변수 타입	숫자 범위	설명	바이트
int	-32768 ~ 32767	정수를 나타냅니다.	2
char	-128 ~ 127	주로 문자를 나타냅니다.	1
long	-2147483648 ~ 2147483647	매우 큰 정수를 나타냅니다.	4

위의 변수들은 음수와 양수를 모두 포함하고 있습니다. 다음과 같이 0 과 양수만으로 이루어지는 변수 타입은 위의 변수 타입에 unsigned 라는 것을 붙이면 됩니다. 이것은 양수, 음수라는 부호가 없다는 뜻입니다.

변수 타입	숫자 범위	설명	바이트
unsigned int	0 ~ 65535	양의 정수를 나타냅니다.	2
unsigned char	0 ~ 255	작은 양의 정수를 나타냅니다.	1
unsigned long	0 ~ 4294967295	매우 큰 양의 정수를 나타냅니다.	4

이 이외에도 변수 타입은 소수점 아래까지 표현할 수 있는 실수형 변수 타입을 포함하여 몇 가지가 더 있습니다. 이에 대해서는 나중에 더 설명하겠습니다. 한꺼번에 너무 많이 하면 힘들 것 같아 이번 장에서는 여기까지만 하겠습니다.
그러면 다시 코드로 돌아 가겠습니다.

```
int button_value;

button_value = digitalRead(BUTTON);

if (button_value == BUTTON_PRESS)
  digitalWrite(LED, ON);
else
  digitalWrite(LED, OFF);
```

button_value 라고 변수를 선언했습니다. 이 변수에 무엇인가를 저장해 볼까요. 저장은 다음과 같이 합니다.

```
button_value = digitalRead(BUTTON);
```

저장 공간을 왼쪽에 쓰고 저장할 값을 오른쪽에 씁니다. 이 코드에서는 버튼 값을 읽어 들여 button_value

에 저장합니다. 그러면 button_value 는 버튼이 눌렸는지 안 눌렸는지를 저장하고 있습니다. 그리고 이 이후의 문장들에서는 digitalRead(BUTTON) 대신 button_value 를 사용하면 됩니다. 컴퓨터 입장에서도 매번 버튼 값을 읽어 오는 대신 바로 button_value 를 사용하니 얼마나 편하겠습니까. 코드는 다음과 같습니다.

```
#define LED 2
#define BUTTON 12
#define BUTTON_PRESS 0

#define ON 1
#define OFF 0
#define Delay(x) delay(x*1000)

void setup() {
  pinMode(LED, OUTPUT);
  pinMode(BUTTON, INPUT);
}

void loop() {
  int button_value;
  button_value = digitalRead(BUTTON);

  if (button_value == BUTTON_PRESS)
    digitalWrite(LED, ON);
  else
    digitalWrite(LED, OFF);
}
```

< 예제 코드 : **버튼이 눌리면 LED 켜기** >

< 코드 위치 : Coding Kit → **led_on_off_but_var** >

한가지 더 알아 두실 것은 위의 코드는 다음과 같이 변수의 선언과 값 할당을 분리하였습니다.

```
int button_value;
button_value = digitalRead(BUTTON);
```

그런데, 다음과 같이 선언과 값 할당을 한번에 할 수도 있습니다.

```
int button_value = digitalRead(BUTTON);
```

변수의 선언의 한 번만 해야 합니다. 그래서 다음과 같이 값을 할당 할 때마다 선언하면 에러가 발생합니다.

```
int button_value = digitalRead(BUTTON_0);
int button_value = digitalRead(BUTTON_1);
```

다음과 같이 같은 변수는 한 번만 선언하여야 합니다.

```
int button_value = digitalRead(BUTTON_0);
```

```
button_value = digitalRead(BUTTON_1);
```

< 문법 설명 : **if 문** >

앞의 코드들에서 자주 보았던 "if 문" 에 대해서 알아 보겠습니다.

```
if (button_value == BUTTON_PRESS)
  digitalWrite(LED, ON);
else
  digitalWrite(LED, OFF);
```

위의 코드는 if 이후의 괄호의 내용이 맞으면 다음 문장을 수행해라 하는 뜻입니다. 즉, "버튼이 눌린 것이 맞으면 LED 를 켜라" 하는 뜻이지요. 여기서 "~하면 ~해라" 하기 때문에 이것을 조건문이라고 합니다. 여러분이 영어 시간에 if 로 시작하는 것을 조건문이라고 배웠지요. 컴퓨터 언어도 사람이 쓰는 언어랑 비슷합니다. 이유는 이 컴퓨터 언어라는 것이 사람이 컴퓨터에게 말하는데 사용되는 것이기 때문에 사람의 언어랑 매우 비슷한 것입니다.

if 문 뒤의 괄호 안에는 주로 조건식이 쓰이는데, 다음과 같은 기호들을 사용한 조건식을 사용합니다. 이와 같은 조건식의 기호들을 "관계 연산자" 라고도 합니다.

기호	의미
==	같다
!=	같지 않다
<	작다
<=	작거나 같다
>	크다
>=	크거나 같다

여기서부터는 수학 시간에 배운 명제의 참, 거짓으로 설명하겠습니다. 참, 거짓이 어렵거나 대단한 것은 아니구요. 맞으면 참이고 틀리면 거짓인 것입니다. 다음 코드에서 if 문 괄호 안의 조건식이 참이 되면 다음 문장을 수행하고 거짓이면 수행하지 않습니다.

```
if (button_value == BUTTON_PRESS)
  digitalWrite(LED, ON);
else
  digitalWrite(LED, OFF);
```

button_value 가 BUTTON_PRESS 라는 값과 같아 이 조건식이 참이 되면 "digitalWrite(LED, ON)" 이 수행이 됩니다. 조건식이 거짓이면 else 이후의 문장 "digitalWrite(LED, OFF)"를 수행합니다.

< 연습 문제 : **버튼이 눌리지 않았을 때 LED 켜기** >

여기서 간단한 예제를 하나 해 볼까요? 버튼이 눌리지 않으면 LED 가 켜지고 버튼이 눌리면 LED 가 꺼지는 예제를 해 보겠습니다. "!=" 기호를 사용하면 간단히 해결될 것 같습니다. 직접 한 번 해 보시지요.

< 코드 위치 : Coding Kit → **led_on_but_unpressed** >

여기서 if 문에 대해서 더 자세히 알아 볼까요? if 문은 문법적으로 다음과 같이 쓸 수 있습니다.

```
if (조건식_1)
  문장_1;
else if (조건식_2)
 문장_2;
else if (조건식_3)
 문장_3;
else
 문장_4;
```

if 혹은 else if 다음의 괄호 안의 조건식이 참이면 바로 다음 문장(문장_1, 2, 3)을 수행합니다. 즉, 조건식_1 이 참이면 문장_1을, 조건식_1이 거짓이고 조건식_2가 참이면 문장_2를, 조건식_1, 2가 거짓이고 조건식_3 이 참이면 문장_3을 수행합니다. if, else if 다음 괄호 안의 값들이 모두 거짓이면 else 다음의 문장(문장_4)을 수행합니다. 여기서 else if 는 단 하나도 안 쓰일 수도 있고 매우 많이 쓰일 수도 있습니다. else 도 단 한번만 쓰일 수도 있고 안 쓰일 수도 있습니다.

다음과 같이 중괄호를 사용하여 중복적으로 사용할 수도 있습니다.

```
if (조건식_1) {
 문장_1;
 문장_2;
}
else if (조건식_2) {
  if (조건식_3)
    문장_3;
  else if (조건식_4)
    문장_4;
  else
    문장_5;
}
```

위의 코드에서 조건식_1 이 참이면 중괄호로 묶인 문장_1 과 문장_2 가 수행 됩니다. 조건식_1 이 거짓이

고 조건식_2 가 참이면 그 이후의 중괄호 안에 있는 if 문 묶음이 수행됩니다.

여기서 말하는 참과 거짓을 컴퓨터의 숫자로 얘기하면 참은 0 이 아닌 모든 수입니다. 거짓은 0 입니다. 그래서 다음과 같이 쓰면 if 다음의 문장이 수행됩니다.

```
if (100)
  문장_1;
else
  문장_2;
```

다음과 같이 쓰면 else 이후의 문장이 수행이 됩니다.

```
if (0)
  문장_1;
else
  문장_2;
```

물론 위의 두 코드와 같이는 잘 쓰지 않습니다. 이렇게 if 문을 정리해 보았습니다. if 문은 프로그램 언어에서 매우 많이 쓰는 조건문입니다. 잘 배워두시길 바랍니다.

< 문법 설명 : **논리 연산자** >

이번에는 버튼 2개를 이용한 예제를 해 보겠습니다. 버튼 2개가 모두 눌려야만 LED 가 켜지는 예제입니다. 먼저 코드를 한 번 볼까요?

```
#define LED 2

#define BUTTON_0 12
#define BUTTON_1 13

#define BUTTON_PRESS 0
#define ON 1
#define OFF 0

void setup() {
  pinMode(LED, OUTPUT);

  pinMode(BUTTON_0, INPUT);
  pinMode(BUTTON_1, INPUT);
}

void loop() {
  int but0_value;
  int but1_value;

  but0_value = digitalRead(BUTTON_0);
```

```
  but1_value = digitalRead(BUTTON_1);

  if ((but0_value == BUTTON_PRESS) && (but1_value == BUTTON_PRESS))
    digitalWrite(LED, ON);
  else
    digitalWrite(LED, OFF);
}
```

setup() 에서는 LED 는 출력 모드로, 버튼은 입력 모드로 하였습니다. loop() 에서는 but0_value 와 but1_value 를 선언해서 각각 버튼 값을 읽어 저장해 두었습니다. 그 다음 문장부터가 중요합니다.

"if ((but0_value == BUTTON_PRESS) && (but1_value == BUTTON_PRESS))" 는 "버튼 0번이 눌렸고, 버튼 1번도 눌렸으면" 하는 뜻입니다. 여기서 && 표시는 "AND 연산자" 또는 "논리곱" 이라고 불리며, && 표시 양쪽의 값이 모두 참이면 결과 값으로 참을 내어 줍니다. if 문은 괄호 안의 값이 참이면 바로 다음 문장을 수행하고 거짓이면 else 이후의 문장을 수행하거나 else 가 없으면 if 문을 그냥 끝냅니다.

AND 연산자는 다음 표와 같은 결과 값을 출력합니다. 참, 거짓은 ture, false 로 쓰겠습니다.

A	B	A && B
false	false	false
false	ture	false
ture	false	false
ture	ture	ture

이런 표를 진리표(Truth Table)라고 합니다. 표를 좀 더 살펴 보면 A, B 둘 다 참이면 참이 되고, 하나라도 거짓이면 거짓입니다. 그래서 위의 코드의 다음 부분에서 버튼이 모두 눌려져야만 LED 가 켜 지는 것입니다.

```
if ((but0_value == BUTTON_PRESS) && (but1_value == BUTTON_PRESS))
  digitalWrite(LED, ON);
else
  digitalWrite(LED, OFF);
```

이제 코딩키트롤 업로드하여 실행 시켜 보겠습니다. 버튼을 하나씩 눌러 보십시요. 절대로 LED 가 켜지지 않습니다. 두 개를 동시에 누르면 드디어 LED 가 켜 집니다.

< 예제 코드 : AND 연산자를 이용한 LED 켜기 >

< 코드 위치 : Coding Kit → **led_on_but_and >**

AND 가 있으면 OR 가 있고, 논리곱이 있으면 논리합이 있겠지요. 그래서 여기서는 OR 연산자, 논리합에

대해서도 알아보고 가겠습니다. OR 연산자는 "||" 기호를 씁니다. OR 연산자에 대한 진리표는 다음과 같습니다.

A	B	A || B
false	false	false
false	ture	ture
ture	false	ture
ture	ture	ture

표를 보면 A 나 B 둘 중 하나가 참이면 결과 값은 참이 되는군요.

< 연습 문제 : **OR 연산자를 이용한 LED 켜기** >

OR 연산자를 이용한 간단한 예제를 하나 해 보겠습니다. 버튼 둘 중 하나라도 눌리면 LED 가 켜지는 예제를 해 보겠습니다. OR 연산자를 이용해서 직접 한 번 해 보십시요.

< 코드 위치 : Coding Kit → **led_on_but_or >**

AND 와 OR 연산자 이외에 논리 연산자가 하나 더 있습니다. 바로 NOT 연산자 또는 "논리 부정" 입니다. 이 연산자는 AND 와 OR 연산자가 좌우에 두 개의 값을 연산하는 것과는 달리 자신의 오른쪽의 값만을 반대로 만듭니다. 기호는 "!" 이고 진리표는 다음과 같습니다.

A	!A
false	true
true	false

간단한 예제를 하나 해 보겠습니다. 전에 버튼이 눌리면 LED 가 켜지는 코드를 작성했습니다. 그 때는 다음과 같이 하였는데, NOT 연산자를 사용하면 매우 간단하게 바꿀 수 있습니다.

```
#define LED 2
#define BUTTON 12
#define BUTTON_PRESS 0

#define ON 1
#define OFF 0

void setup() {
```

```
  pinMode(LED, OUTPUT);
  pinMode(BUTTON, INPUT);
}

void loop() {
  if (digitalRead(BUTTON) == BUTTON_PRESS)
    digitalWrite(LED, ON);
  else
    digitalWrite(LED, OFF);
}
```

위의 코드에서는 if 문을 사용하였는데, NOT 연산자를 사용하여 다음과 같이 버튼의 값을 LED 로 바로 전달합니다.

```
#define LED 2
#define BUTTON 12

void setup() {
  pinMode(LED, OUTPUT);
  pinMode(BUTTON, INPUT);
}

void loop() {
  int but_val = digitalRead(BUTTON);
  digitalWrite(LED, !but_val);
}
```

위의 코드에서 보면 버튼 값을 읽어 but_val 변수에 저장합니다. 아시다시피 버튼은 눌리면 0 이고 눌리지 않으면 1 입니다. 그래서 but_val 에 NOT 연산자를 취하면 버튼이 눌렸을 때 1 이 되고, 눌리지 않았을 때 0 이 됩니다. 이 값을 그대로 LED 전달하면 LED 는 버튼이 눌렸을 때 켜지고, 버튼이 눌리지 않았을 때 꺼집니다.

< 예제 코드 : **NOT 연산자를 이용하여 버튼이 눌렸을 때 LED 켜기** >

< 코드 위치 : Coding Kit → **led_but_not >**

다음은 코딩키트의 버튼 2개와 LED 4개를 이용한 예제입니다. 여기서 버튼 2개가 어떻게 눌리느냐에 따라서 켜지는 LED 수를 다르게 하는 예제를 해 보겠습니다. 다음은 버튼 2개가 눌리는 경우와 이에 따라 LED 가 켜지는 경우를 나타내는 표입니다.

BUTTON 1	BUTTON 0	LED 3	LED 2	LED 1	LED 0
안 눌림	안 눌림	OFF	OFF	OFF	ON
안 눌림	눌림	OFF	OFF	ON	ON

눌림	안 눌림	OFF	ON	ON	ON
눌림	눌림	ON	ON	ON	ON

표의 의미는 버튼이 모두 안 눌리면 LED 0 만 켜집니다. 모두 눌리면 LED 가 모두 켜집니다. 나머지도 이와 비슷한 의미입니다. 위의 표와 같이 동작하도록 코딩해 보겠습니다. 조금 어려운 것 같지만 "!=" 와 "&&" 를 잘 이용하면 그렇게 어렵지 않습니다. 코드는 다음과 같습니다.

```
#define LED_0 2
#define LED_1 3
#define LED_2 7
#define LED_3 8

#define BUTTON_0 12
#define BUTTON_1 13

#define BUTTON_PRESS 0
#define ON 1
#define OFF 0

void setup() {
  pinMode(LED_0, OUTPUT);
  pinMode(LED_1, OUTPUT);
  pinMode(LED_2, OUTPUT);
  pinMode(LED_3, OUTPUT);

  pinMode(BUTTON_0, INPUT);
  pinMode(BUTTON_1, INPUT);
}

void loop() {
      int but0_value;
      int but1_value;

      but0_value = digitalRead(BUTTON_0);
      but1_value = digitalRead(BUTTON_1);

      if ((but1_value != BUTTON_PRESS) && (but0_value != BUTTON_PRESS)) {
                digitalWrite(LED_0, ON);
                digitalWrite(LED_1, OFF);
                digitalWrite(LED_2, OFF);
                digitalWrite(LED_3, OFF);
      }
      else if ((but1_value != BUTTON_PRESS) && (but0_value == BUTTON_PRESS)) {
                digitalWrite(LED_0, ON);
                digitalWrite(LED_1, ON);
                digitalWrite(LED_2, OFF);
                digitalWrite(LED_3, OFF);
      }
      else if ((but1_value == BUTTON_PRESS) && (but0_value != BUTTON_PRESS)) {
```

```
                digitalWrite(LED_0, ON);
                digitalWrite(LED_1, ON);
                digitalWrite(LED_2, ON);
                digitalWrite(LED_3, OFF);
        }
        else if ((but1_value == BUTTON_PRESS) && (but0_value == BUTTON_PRESS)) {
                digitalWrite(LED_0, ON);
                digitalWrite(LED_1, ON);
                digitalWrite(LED_2, ON);
                digitalWrite(LED_3, ON);
        }
    }
```

코드를 보면 if 문 안에서 "!=" 는 눌리지 않았으면 이고, "==" 눌렸으면 입니다. 이 부분만 이해하면 전체적으로 어려운 부분은 없습니다. 코드가 좀 길지요. 하지만 대부분 반복되는 부분입니다.

< 예제 코드 : **버튼 2 개의 눌림에 따른 LED 4 개 켜기** >

< 코드 위치 : Coding Kit → **led4_on_off_but2 >**

< 연습 문제 : **버튼 2 개의 눌림에 따른 LED 4 개 켜기** >

버튼 2 개가 다음과 같이 눌렸을 때 LED 가 다음과 같이 켜지는 예제를 해 봅니다.

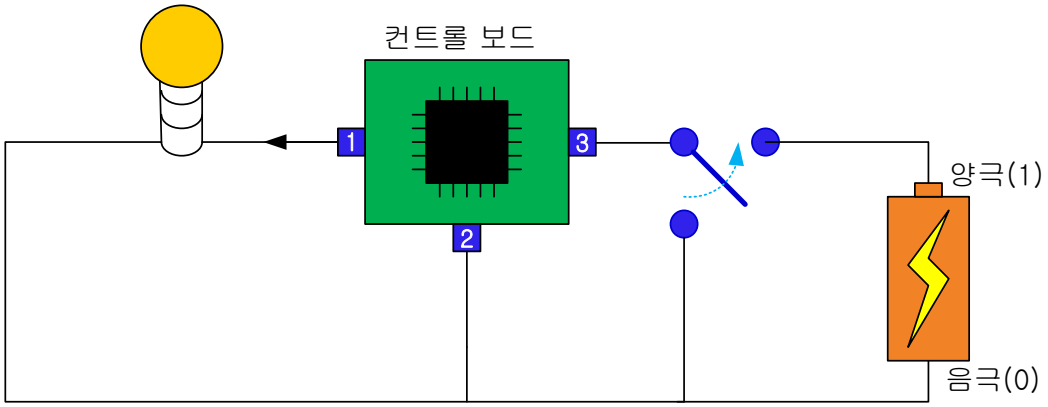
BUTTON 1	BUTTON 0	LED 3	LED 2	LED 1	LED 0
안 눌림	안 눌림	OFF	OFF	OFF	OFF
안 눌림	눌림	OFF	OFF	ON	ON
눌림	안 눌림	ON	ON	OFF	OFF
눌림	눌림	ON	ON	ON	ON

버튼 0 이 눌리면 LED 0 과 LED 1 번만 켜 집니다. 버튼 1 이 눌리면 LED 2 와 LED 3 번이 켜집니다. 한 번 잘 생각해 보시고, 나중에 제가 코딩한 것과도 비교해 보세요.

< 코드 위치 : Coding Kit → **led4_on_left_right >**

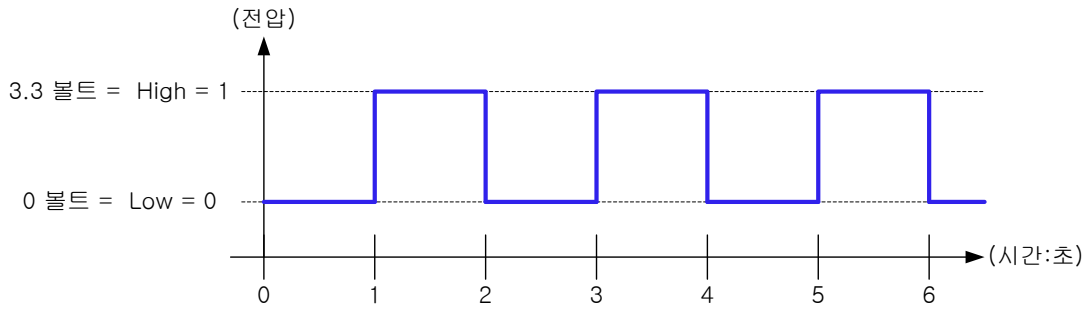
[전기 신호를 표시하는 방법과 PWM(Pulse Width Modulation)]

컴퓨터는 기본적으로 0 과 1, 두 숫자만을 사용합니다. 0 과 1 을 다르게 표현하는 여러 가지가 있습니다. 이 중에는 위에서 언급한 참과 거짓도 있고요. LED 를 켰다 끄는 ON, OFF 도 있습니다. 또 하나는 전기 신호의 레벨을 표시하는 High, Low 도 있습니다. 여기서는 High, Low 에 대해서 알아보겠습니다. 전에 보았던 다음 그림을 한번 더 볼까요?



위 그림의 건전지는 전압이 몇 볼트일까요? 3.3 볼트라고 가정해 보겠습니다. 그럼 스위치가 건전지의 양극에 접촉이 되면 컨트롤 보드로는 3.3 볼트가 입력이 됩니다. 음극에 접촉이 되면 0 볼트가 입력이 되겠지요. 이 전압을 바탕으로 컨트롤 보드의 모든 동작이 이루어집니다. 그래서 컨트롤 보드에서 LED 를 켜기 위해 출력해 주는 전압도 3.3 볼트가 되는 것입니다. 그런데, 코딩을 할 때는 3.3 볼트가 중요한 것은 아니고 이것을 그냥 숫자 1 로 사용하고, 0 볼트는 숫자 0 으로 사용합니다. 여기서 1 은 또다시 참, ON, High, ... 이런 값으로 대체해서 사용합니다. 0 도 마찬가지로 거짓, OFF, Low, ... 이런 값으로 대체해서 편리하게 사용합니다. 여러분은 코딩할 때 이런 값들을 편리하게 사용하면서 가독성이 좋게 코딩하시면 됩니다. 그런데, 왜 여기서 이런 3.3 볼트니 하는 전압 값을 이야기 할까요? 그것은 전기 신호를 표시할 때 주로 전압으로 표시를 하고, 그 전압 표시가 코딩키트와 같은 하드웨어를 활용한 코딩과 관련이 있어 그럽니다.

전기 신호는 시간의 흐름에 따라 변합니다. 즉, 그림에서 스위치를 1 초 간격으로 건전지의 양극과 음극을 접촉해 보겠습니다. 그 때 컨트롤 보드에 입력되는 전압의 변화를 그래프로 그려 보면 다음 그림과 같을 것입니다.



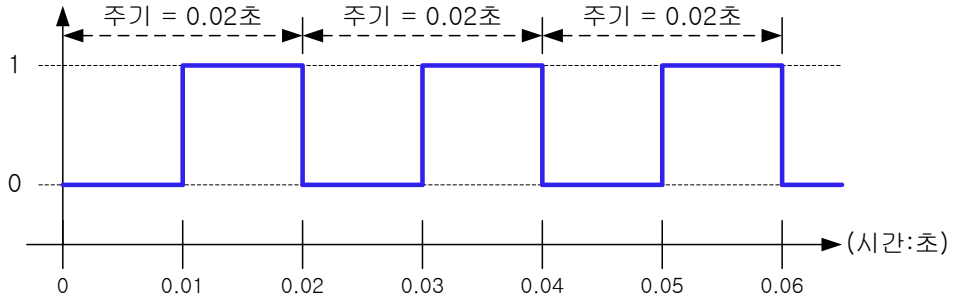
전기 신호는 보통 위의 그림과 같이 표시를 합니다. 그리고 이것을 전기 신호의 파형(Wave)이라고 합니다.

이 신호를 그대로 LED 로 출력을 하면 어떻게 될까요? 그러면 LED 가 1 초 주기로 깜박됩니다. 그러면 0.1 초, 0.01 초, 0.001 초 주기로 깜박되면 LED 는 어떻게 보일까요? 글쎄요? 뭘 망설입니까? 바로 코딩해 보면 알겠지요. 그럼 다음과 같이 코딩해 보도록 하겠습니다. 버튼 2 개가 모두 안 눌리면 1초, 버튼 0 번 한 개만 눌리면 0.1초, 버튼 1 번 한 개만 눌리면 0.01 초, 버튼 2 개 모두 눌리면 0.001 초 간격으로 LED 가 깜박이는 코드를 해 보겠습니다.

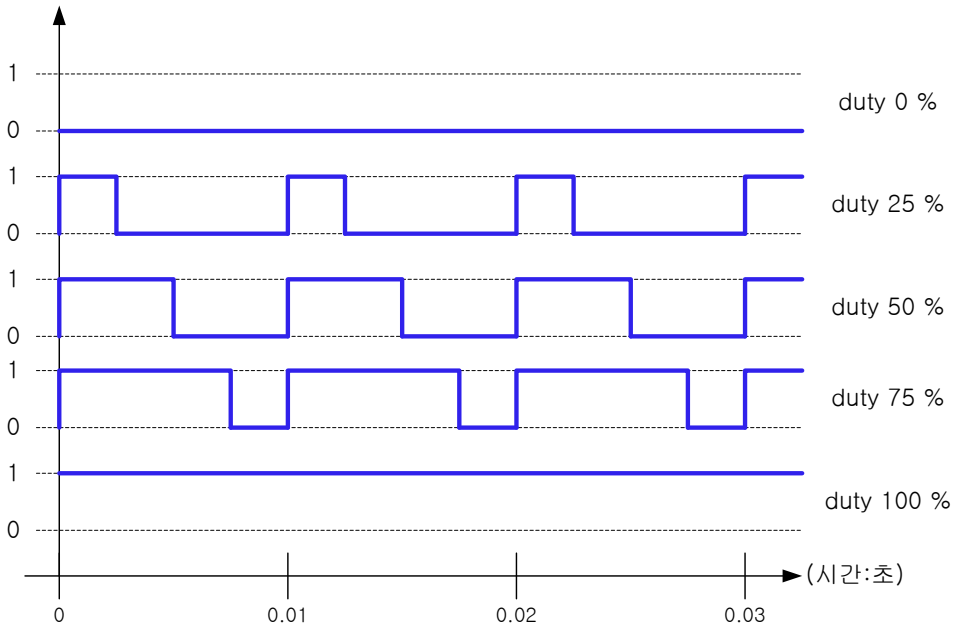
```
#define LED 2

#define BUTTON_0 12
#define BUTTON_1 13

#define BUTTON_PRESS 0
#define ON 1
#define OFF 0

void setup() {
  pinMode(LED, OUTPUT);

  pinMode(BUTTON_0, INPUT);
  pinMode(BUTTON_1, INPUT);
}

void loop() {
  int but0_value = digitalRead(BUTTON_0);
  int but1_value = digitalRead(BUTTON_1);

  if ((but1_value != BUTTON_PRESS) && (but0_value != BUTTON_PRESS)) {
    digitalWrite(LED, ON);
    delay(1000);
    digitalWrite(LED, OFF);
    delay(1000);
  }
  else if ((but1_value != BUTTON_PRESS) && (but0_value == BUTTON_PRESS)) {
    digitalWrite(LED, ON);
    delay(100);
    digitalWrite(LED, OFF);
    delay(100);
  }
  else if ((but1_value == BUTTON_PRESS) && (but0_value != BUTTON_PRESS)) {
    digitalWrite(LED, ON);
    delay(10);
    digitalWrite(LED, OFF);
    delay(10);
  }
  else if ((but1_value == BUTTON_PRESS) && (but0_value == BUTTON_PRESS)) {
    digitalWrite(LED, ON);
    delay(1);
```

```
    digitalWrite(LED, OFF);
    delay(1);
  }
}
```

코드에서 delay() 의 괄호 안의 값은 1000 분의 1 초를 표시한다고 했습니다. 그래서 delay(1000) 은 1 초, delay(100) 은 0.1 초, delay(10) 은 0.01 초, delay(1) 은 0.001 초를 표시합니다. 그래서 LED 를 켜 보면, 0.01 초와 0.001 초는 거의 켜져 있는 것과 같이 보입니다. 그래도 자세히 보면 0.01 초는 약간 흔들려 보입니다. 실제로는 LED 가 계속해서 켜졌다 꺼졌다를 반복하고 있는 것입니다. 여기서 0.01 초의 파형을 그려보면 다음과 같습니다.



이 그림에서는 정확한 전압 값을 쓰지 않고 코딩할 때 사용하는 0 과 1 을 썼습니다. 위와 같이 어떤 **고정된 시간 안에서** 반복적으로 0 과 1 사이를 움직이는 신호를 PWM(Pulse Width Modulation)이라고 합니다. 위 그림에서 고정된 시간은 0.02 초 입니다. 이렇게 정해진 시간을 주기라고 합니다. PWM 신호는 주기 안에서 1 인 구간의 길이와 0 인 구간의 길이를 다르게 하여 어떤 의미 있는 신호를 만들어 냅니다. 다음과 같은 파형들을 볼까요?



위 그림은 여러가지 PWM 신호들의 파형입니다. 신호의 1 인 구간이 주기에서 몇 % 를 자치하냐를 Duty

Ratio 라고 합니다. Duty Ratio 를 엔지니어들 사이에서는 듀티비라고 부릅니다. 듀티비라는 용어가 영어와 우리말이 적당히 섞인 말인데, 통상적으로 그렇게 부르고 있으니 우리도 그렇게 하지요. 그림에서 제일 위의 파형은 1 값이 전혀 없기 때문에 듀티비가 0 % 이고, 그 다음부터는 1 값에 따라서 듀티비를 나타냅니다. 제일 아래 파형은 주기가 모두 1 이기 때문에 듀티비가 100 % 입니다. 중간의 파형은 1 값이 전체에서 반이기 때문에 듀티비가 50 % 입니다.

< 예제 코드 : **LED 를 주기적으로 매우 빠르게 켜기 (신호 파형 이해)** >

< 코드 위치 : Coding Kit → **led_wave_but2 >**

이제 PWM 신호를 이용해서 LED 의 밝기를 조절해 보겠습니다. 다음 코드는 LED 의 밝기를 PWM 신호를 이용하여 서서히 밝아지게 하는 코드입니다. 지금까지는 첫번째 LED 를 사용하였는데, 이번에는 두번째 LED 를 사용합니다.

```
#define LED 3

#define ON 1
#define OFF 0

void setup() {
  pinMode(LED, OUTPUT);
}

void loop() {
  // LED Off
  digitalWrite(LED, OFF);
  delay(1000);

  // Duty Ratio : 20 %
  for (int i = 0; i < 100; i++) {   // delay : 100 * (2 + 8) = 1000
     digitalWrite(LED, ON);
     delay(2);
     digitalWrite(LED, OFF);
     delay(8);
  }

  // Duty Ratio : 40 %
  for (int i = 0; i < 100; i++) {
     digitalWrite(LED, ON);
     delay(4);
     digitalWrite(LED, OFF);
     delay(6);
  }

  // Duty Ratio : 60 %
```

```
    for (int i = 0; i < 100; i++) {
      digitalWrite(LED, ON);
      delay(6);
      digitalWrite(LED, OFF);
      delay(4);
    }

    // Duty Ratio : 80 %
    for (int i = 0; i < 100; i++) {
      digitalWrite(LED, ON);
      delay(8);
      digitalWrite(LED, OFF);
      delay(2);
    }

    // LED ON
    digitalWrite(LED, ON);
    delay(1000);
  }
```

loop() 안의 코드를 보면 일단 LED 를 꺼 주는 코드를 썼습니다. 그 다음은 주석으로 듀티비가 20 % 라고 하고 그 다음으로 for 로 시작하는 구문이 있습니다. 이것은 반복문 중의 하나인 for 문인데요. 새롭게 보는 for 문을 설명드리고 계속 진행하겠습니다.

< 예제 코드 : **PWM 신호로 LED 켜기** >

< 코드 위치 : Coding Kit → **led_pwm >**

< 문법 설명 : **for 문** >

for 문은 다음과 같이 문법적으로 쓸 수 있습니다.

```
for (인덱스_초기값 ; 인덱스_조건식 ; 인덱스_증가) {
  문장_1;
  문장_2;
}
```

for 문은 **괄호 안에서 정의한 수**만큼 문장들을 반복해서 수행하는 것입니다. 여기서 중요한 것은 괄호 안에서 정의한 수라는 것인데요. 그러면 먼저 for 문의 괄호 안의 내용부터 살펴 보겠습니다. 괄호 안에는 세미콜론(;)으로 분리되는 3개의 항목이 있습니다. 이 3개의 항목은 모두 **인덱스**라는 것을 포함하며, 이 인덱스를 이용하여 for 문의 반복 횟수를 결정합니다. 다음 예제를 보면서 설명하겠습니다.

```
int i;
for (i = 0; i <= 3; i = i + 1) {
```

```
  문장1;
  문장2;
  ...
}
```

위 코드에서 변수 i 를 선언하였습니다. 이 변수 i 가 인덱스인 것입니다. for 문 괄호 안의 첫번째 항목을 보면 i 의 초기값을 정의 합니다. 여기서는 **0 으로 정의합니다.** 그 다음으로 두번째 항목인 "i <= 3" 라는 조건식에 대입해 봅니다. 그러면 "0 <= 3" 라는 식이 참 됩니다. 참이 되면 for 문 안의 문장들을 수행합니다. 그 다음으로 세번째 항목으로 갑니다. 세번째 항목은 i 에 1 을 더해서 i 에 저장합니다. i 의 초기값이 0 이므로 1 을 더하면 **1 이 됩니다.** 이제 i 값은 1 이 되었고 이것을 두번째 항목인 " i <= 3" 라는 조건식에 대입해 봅니다. 그러면 "1 <= 3" 라는 식은 참이 됩니다. 이 값이 참이면 for 문 안의 문장들을 수행합니다. 여기서 for 문 안의 문장이라는 것은 위 코드에서 붉은색 중괄호 안의 문장을 말합니다. 이렇게 문장을 수행한 이후에는 for 문 괄호 안의 세번째 항목인 "i = i + 1" 을 수행합니다. 이전에 i 는 1 이였으므로 1 을 더하면 이제 **2 가 됩니다**. 이것을 두번째 항목인 3보다 작냐? 를 해 보면 2 는 3 보다 작으므로 결과값은 참입니다. 그래서 for 문 안의 문장을 또 수행합니다. 그리고 다시 " i = i + 1" 을 수행하여 값은 **3 이 됩니다**. 그럼 또 문장을 수행하고 " i = i + 1" 을 합니다. 이제 i 는 **4 가 됩니다**. 이 4 를 조건식에 대입해서 보면 "4 <= 3" 는 거짓입니다. 그래서 for 문 안의 문장을 수행하지 않고 for 문의 반복도 끝을 냅니다. for 문은 이런식으로 동작을 합니다.

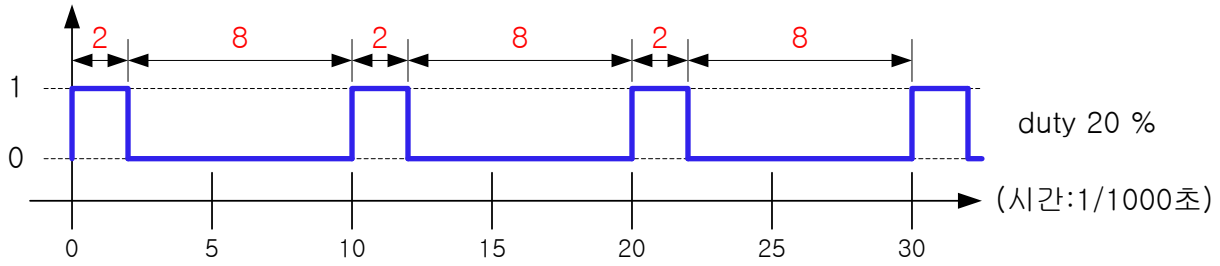
위에서 변수 i 는 0, 1, 2, 3 으로 변합니다. "i = i + 1" 은 다음과 같이 "i++" 라고도 씁니다. 둘 다 i 를 1 씩 증가한다는 뜻입니다. 변수 i 의 선언은 for 문의 괄호 안에서 해도 됩니다. 이상을 반영한 코드는 다음과 같습니다. for 문은 일반적으로 이렇게 사용합니다.

```
for (int i = 0; i <= 3; i ++) {
  문장;
}
```

이제 코드로 다시 돌아가면 다음은 첫번째 for 문입니다.

```
// Duty Ratio : 20 %
  for (int i = 0; i < 100; i++) {   // delay : 100 * (2 + 8) = 1000
     digitalWrite(LED, ON);
     delay(2);
     digitalWrite(LED, OFF);
     delay(8);
  }
```

이 for 문 안의 문장은 먼저 LED 를 켭니다. 그리고 delay(2) 의 코드에서 의해서 2초 안 LED 를 켠 상태로 유지합니다. 다음 코드는 LED를 끄고 8 초 동안 유지합니다. 이렇게 만든 PWM 신호는 전체 시간 10 에서 켜는 시간이 2 이기 때문에 듀티비가 20 % 입니다. 이 코드가 for 문에 의해서 100 번 반복되는 것입니다. 이렇게 만들어진 듀티비가 20 % 인 파형은 다음 그림과 같습니다.



delay() 의 단위는 0.001 초이기 때문에 이 파형은 1 인 구간이 0.002 초이고 0 인 구간이 0.008 초입니다. 이 신호를 for 문에 의해서 100 번을 반복하기 때문에 전체 동작 시간은 다음과 같은 공식에 의해서 1 초입니다.

(0.002 + 0.008) x 100 = 1

이것과 같이 1 초 동안 듀티비가 40%, 60%, 80% 인 신호가 그 다음부터 for 문을 사용해서 만들어집니다. 이렇게 LED 에 듀티비가 점점 상승하는 신호를 보내 주면 LED 는 단계적으로 밝아지게 됩니다. 이제 코딩키트에서 실행시켜 보십시요. 실제로 LED 가 단계적으로 밝아지는 것을 확인할 수 있습니다.

위의 예제에서 모든 for 문은 듀티비를 틀리게 하는 delay() 의 괄호 안의 값만 틀리고 다른 코드는 모두 같습니다. 그래서 다음과 같이 for 문 안에 for 문을 사용하여 코드를 간단하게 만들어 보겠습니다. 동작은 위의 예제와 정확히 일치하지는 않습니다.

```
#define LED 3

#define ON 1
#define OFF 0

void setup() {
  pinMode(LED, OUTPUT);
}

void loop() {
  for (int i = 0; i < 10; i++) {
    for (int j = 0; j < 100; j++) {
      digitalWrite(LED, ON);
      delay(i);
      digitalWrite(LED, OFF);
      delay(10 - i);
    }
  }
}
```

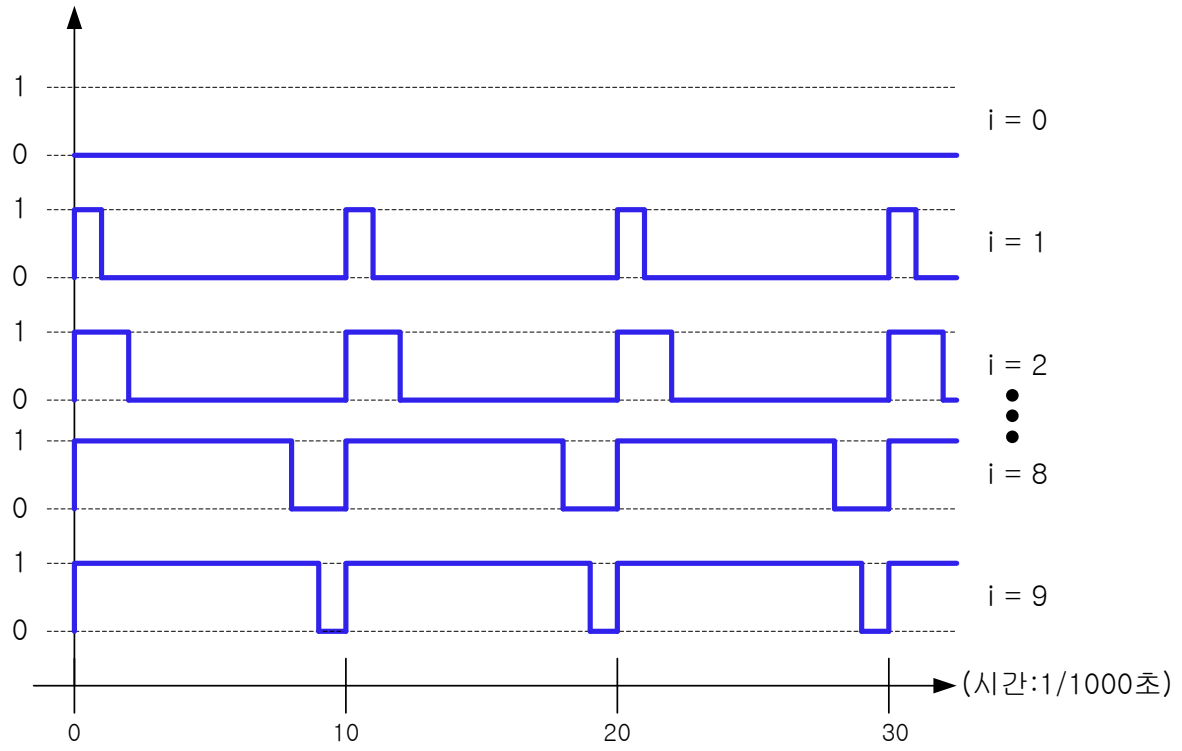
여기서는 for 문을 두 번을 반복했는데요. 두 번 반복했다고 해서 어려울 것은 없습니다. 그냥 for 문 안의

for 문도 그냥 이전 for 문 안에서 수행해야할 문장일 뿐입니다. 코드을 더 살펴보면, 첫번째 for 문은 인덱스 i 가 0 에서 9 까지 변해서 10 번 수행 하는 것이고, 두번째 for 문은 인덱스 j 가 0 에서 99 까지 변해서 100 번 수행하는 것입니다. 붉은색 코드들이 PWM 신호를 만듭니다. 일단 "digitalWrite(LED, ON)" 으로 LED 를 켭니다. 그리고 LED 가 켜져 있는 시간을 "delay(i)" 로 만들어 줍니다. 여기서 변수 i 는 for 문의 인덱스로 사용된 변수 i 입니다. 그래서 이 값은 "for (int i = 0; i < 10; i++)" 의 코드에 의해서 0 ~ 9 까지 변합니다. 그러면 LED 가 켜지는 시간은 0 ~ 9 까지 변합니다.

그리고 "digitalWrite(LED, OFF)" 으로 LED 를 끕니다. 그리고 LED 가 꺼지는 시간은 "delay(10 - i)" 로 했습니다. 그러면 이 시간은 i 가 0 ~ 9 까지 변할 때 10, 9, 8, ..., 1 로 변합니다.

그래서 LED 가 켜지는 시간은 0 ~ 9 까지 변하고 LED 가 꺼지는 시간은 10 ~ 1 까지 변합니다. 그러면 같은 시간 동안에 LED 가 켜지는 시간은 점점 늘어나고 LED 가 꺼지는 시간은 점점 줄어듭니다. 그러면 LED 는 서서히 밝아지게 되는 것입니다.

이렇게 해서 LED 로 출력할 PWM 신호가 다음과 같이 만들어 집니다.



각각은 1 초동안 유지가 됩니다. 이 1 초는 다음과 같은 식에 의해서 계산이 됩니다.

i + (10 - i) x 100 = 10 x 100 = 1000

위의 식에서 i 와 10 - i 는 delay 의 괄호 안의 값이고, 100 은 " for (int j = 0; j < 100; j++)" 에 의해서 반복되는 횟수입니다. 여기서 delay 의 단위는 0.001 초이기 때문에 1000 x 0.001 초 하여 1 초가 됩니다.

위와 같이 1 인 구간이 변한다는 것은 LED 가 켜지는 구간이 변한다는 것입니다. 그리고 이 시간은 매우 짧기 때문에 LED 의 밝기가 변하는 것을 보실 수 있습니다. 그러면 위의 예제 코드를 한 번 실행해 볼까요?

< 예제 코드 : 2 중 **for 문을 이용한 PWM 신호 만들기** >

< 코드 위치 : Coding Kit → **led_pwm_for2 >**

이번 예제와 코드 설명, 문법 설명은 길고 어려웠습니다. 본인이 완벽히 이해가 되지 않으셨다고 해서 크게 걱정할 것은 없습니다. 이번에는 어려운 부분이어서 많은 분들이 이해하기 힘들었을 것입니다. 앞으로 코딩키트와 함께 계속해서 공부하신다면 프로그램 언어에 익숙해 지면서 자연스럽게 이해가 될 것입니다. 여러분이 영어, 중국어 등을 공부할 때도 처음에 말이 입에 잘 붙지도 않고 잘 들리지도 않던 것이 계속해서 공부해 나가다 보면 익숙해져서 잘 들리고 잘 말할 수 있습니다. 코딩도 마찬가지입니다. 꾸준히 공부해 나가다 보면 재미가 생기고 익숙해져서 잘 하게 될 수 있습니다. 그리고 영어, 중국어를 배우는 것보다 나은 점은 그 언어들보다는 훨씬 쉽고 재미있다는 것입니다. 그러니 걱정 마시고 잘 따라와 주시길 부탁드립니다.

[analogWrite() 를 이용하여 PWM 신호 만들기]

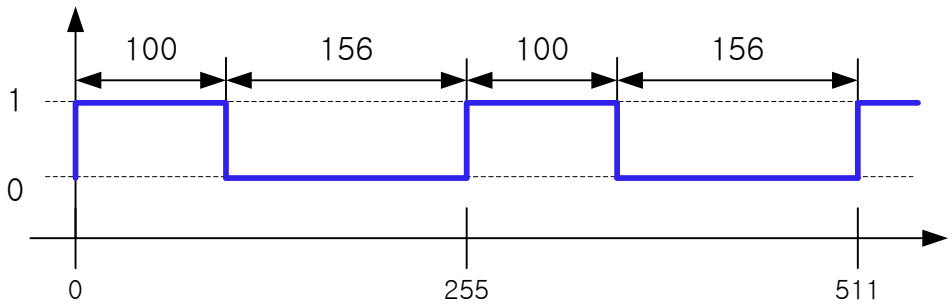
위에서 digitalWrite() 를 이용하여 PWM 신호를 만들었습니다. 그런데, 여기서 한 가지 알고 가야할 것이 위와 같이 하면 PWM 신호가 정확한 시간을 맞추어 나오지는 않습니다. 그것은 "delay()" 라는 문장 수행 이후나 이전에 다른 문장들을 수행하는데, 여기에도 시간이 걸리기 때문에 정확한 PWM 신호를 만들 수는 없습니다. 그래서 아두이노에서는 digitalWrite() 대신 analogWrite() 이용하면 정확한 PWM 신호를 마들 수 있습니다. 이것은 다음과 같이 사용합니다.

```
analogWrite(LED, value);
```

digitalWrite() 에서는 다음과 같이 핀에 0 혹은 1 값만을 줄 수 있었습니다.

```
digitalWrite(LED, 0)
digitalWrite(LED, 1)
```

하지만 analogWrite() 에서는 value 위치에 값을 0 ~ 255 까지 숫자를 줄 수 있습니다. 이 숫자의 의미는 PWM 신호 중 1 인 구간에 대한 시간입니다. 이 값을 최대 255 로 하면 1 인 구간이 PWM 신호 전체 구간이 되는 것이고 0 으로 하면 1 인 구간이 없는 것입니다. 혹은 이 값을 100 으로 하면 255 등분 한 중에 100 만큼이 1 이고 나머지가 0 입니다. 이것을 그림으로 그리면 다음과 같습니다.

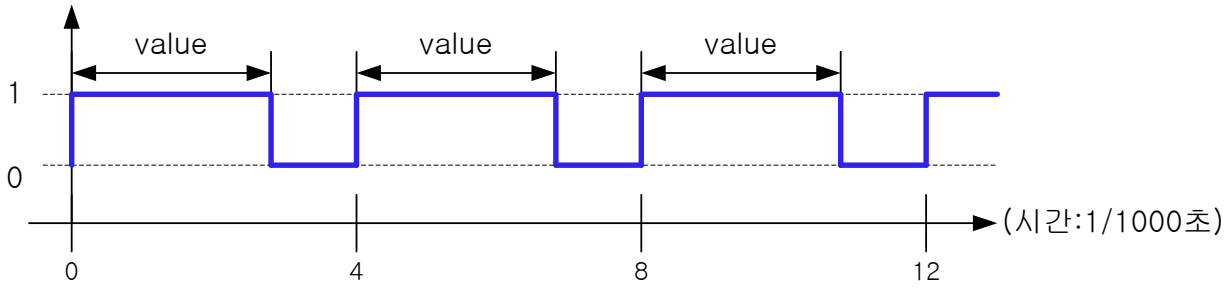


그런데, 왜 값을 0 ~ 100 도 아니고 0 ~ 255 를 사용했을까요? 그것은 컴퓨터는 2 진수를 사용하는데, 0 ~ 255 는 2 진수 8 자리 값입니다. 이 값은 여러분이 많이 들어 보셨던 바이트(Byte)입니다. 그래서 0 ~ 255 를 사용하는 것입니다. 2 진수니, 바이트니 하는 부분은 나중에 자세히 더 설명하겠습니다.

이렇게 하여 analogWirte() 를 사용하면 PWM 신호의 1 인 구간과 0 인 구간을 자유롭게 조정하여 만들 수 있습니다. 즉, 듀티비를 조정할 수 있는 것입니다.

PWM 신호의 1 인 구간 시간과 0 인 구간의 시간을 합친 것을 **주기**라고 한다고 했습니다. 즉, PWM 신호는 이 주기를 계속해서 반복하는 신호인 것입니다. 아두이노의 analogWrite() 를 이용하여 만들어진 PWM 신호 주기의 실제 시간은 0.004 초입니다. 그래서 analogWrite() 를 이용하여 만든 신호는 0.004 초 안에서 1 인 구간과 0 인 구간을 나누는 것입니다. 이것을 나눌 때 0 ~ 255 값 중 어떤 값을 쓰면 그 값만큼 1 인 구간의 시간이 결정되는 것입니다.

주기가 0.004 초라는 것은 위 그림에서 0 ~ 255 구간의 시간이 0.004 초라는 의미입니다. 주기가 0.004 초인 신호의 파형은 다음과 같습니다.



그러면 이제 analogWrite() 를 이용하여 LED 의 밝기를 조절하는 코딩을 해 보겠습니다.

```
#define LED 3

#define ON 1
#define OFF 0

void setup() {
  pinMode(LED, OUTPUT);
}

void loop() {
  for (int i = 0; i < 256; i++) {
    analogWrite(LED, i);
    delay(10);
  }
  digitalWrite(LED, OFF);
  delay(500);
}
```

위 코드에서 for 문의 인덱스 i 는 0 에서 255 까지 변합니다. 그래서 다음과 같은 analogWrite() 가 수행이 됩니다.

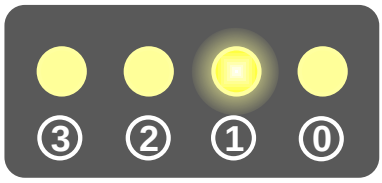
```
analogWrite(LED, 0);
analogWrite(LED, 1);
analogWrite(LED, 2);
.
.
.
analogWrite(LED, 254);
analogWrite(LED, 255);
```

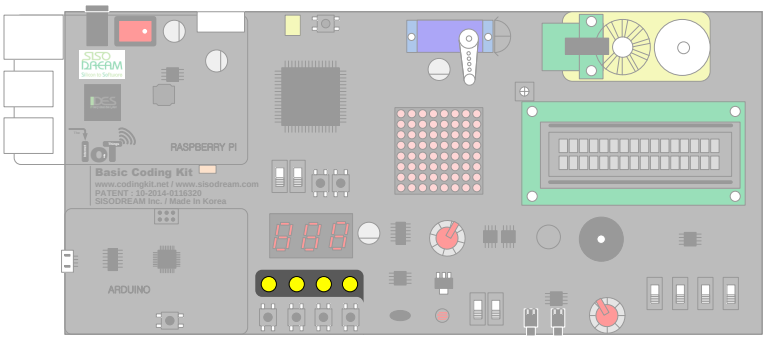
이러한 코드들이 수행이 되면 출력되는 PWM 신호의 1 인 구간은 0 ~ 255 까지 변하면서 LED 의 밝기를 변화 시킵니다.

for 문에서 delay(10) 이라고 해서 0.01 초의 시간을 주었는데, 이것은 지금의 밝기로 0.01 초를 유지하라는 뜻입니다. 이것은 현재의 PWM 신호를 0.01 초 유지하라고 하는 뜻과도 같습니다.

이제 아두이노 프로그램으로 업로드 하여 코딩키트에서 확인해 보면 LED 의 밝기 변화가 좀 더 자연스러운 것을 확인할 수 있을 것입니다.

그리고 여기서 한 가지 더 말씀 드릴 것은 이번 예제는 특이하게 코딩키트의 0 번 LED 를 사용하지 않고 1 번 LED 를 사용했습니다.





그 이유는 아두이노에서는 모든 핀으로 analogWrite() 를 사용하여 PWM 신호를 출력 할 수 있는 것은 아닙니다. 3, 9, 10, 11 번 핀과 5, 6 번 핀을 PWM 출력으로 사용할 수 있습니다. 이 중 방금 예제에서 사용했던 3 번 핀은 주기가 0.004 초였습니다. 9, 10, 11 번 핀도 마찬가지로 주기가 0.004 초입니다. 5, 6 번은 PWM 신호의 주기가 0.002 초로 더 짧습니다. 이 주기를 잘 고려하여 사용하시면 됩니다.

< 예제 코드 : **analogWrite() 함수를 이용하여 PWM 신호 만들기** >

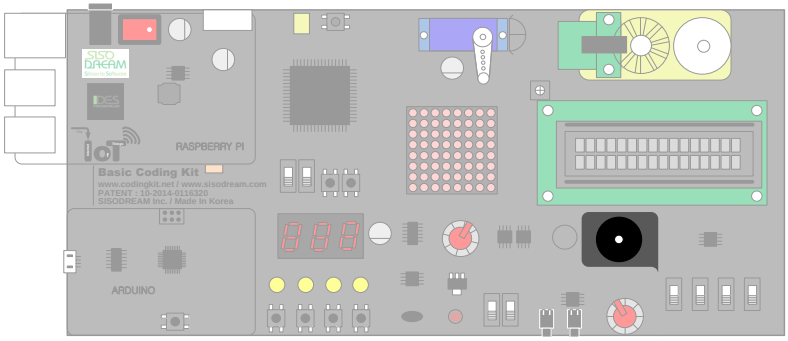
< 코드 위치 : Coding Kit → **led_pwm_analog >**

Note : 코딩키트의 아두이노는 3.3V, 8MHz 를 사용합니다. 그래서 analogWrite 핀의 PWM 주기가 0.002 초와 0.004 초입니다. 아두이노 5V, 16MHz 제품에서는 3, 9, 10, 11 번 핀의 PWM 주기는 0.002 초이고, 5, 6 번 핀의 PWM 주기는 0.001 초입니다. 코딩키트에 장착된 아두이노보다 주기가 반입니다.

[부저(Buzzer) 소리 내기]

코딩키트에는 다양한 디바이스가 있습니다. 그런데, 여태까지는 주구장창 LED 와 버튼만 했습니다. 여러분! LED 와 버튼 이제 지겹지요? 그래도 꾹 참고 여기까지 따라와 주셔서 감사합니다. 지금까지의 LED 와 버튼 예제로 여러분은 코딩의 많은 것을 배우셨습니다. 기초 체력이 튼튼해지신 겁니다. 이제부터는 그 기초 체력을 바탕으로 더 재미있고 신나는 코딩을 해 보겠습니다.

이번에 사용해 볼 디바이스는 부저(Buzzer)입니다. 부저는 다음 사진과 같은 모습니다. PWM 신호를 주면 삐~~~ 소리를 냅니다.



그럼 부저 예제를 당장 한번 해 보겠습니다.

```
#define BUZ 5
#define Delay(x) delay(x*1000)
#define BUZ_OFF 0

void setup() {
  pinMode(BUZ, OUTPUT);

  analogWrite(BUZ, 127);
  Delay(5);
  analogWrite(BUZ, BUZ_OFF);
}

void loop() {
}
```

부저는 아두이노의 5 번 핀에 연결되어 있습니다. 그리고 부저는 계속해서 울어대면 매우 시끄럽습니다. 그래서 5초만 울리게 했습니다. analogWrite(BUZ, 127) 하면 PWM 신호는 0 ~ 255 중 0 ~ 127 만을 1 로 하고 나머지 128 ~ 255 를 0 으로 합니다. 즉, 듀티비가 50 % 인 PWM 신호입니다. 코드를 업로드해 보면 5초간 삐~~~ 소리가 납니다.

< 예제 코드 : **부저 소리 내기** >

< 코드 위치 : Coding Kit → **buz_pwm >**

여러분이 앞에서 PWM 신호에 대해서 많은 공부를 해 두셔서 설명할 것이 별로 없습니다. 한 가지 주의하실 것은 anlogWrite(BUZ, 255) 혹은 digitalWrite(BUZ, 1) 등의 코드로 부저에 계속 1 이 유지되는 신호를 주면 부저가 서서히 상태가 안 좋아 지면서 망가질 수가 있습니다. 그래서 코딩키트에서는 2 초 이상 부저에 1 값을 계속해서 주면 이 값을 자동으로 0 으로 떨어지게 만들었습니다. 그래서 코딩키트를 사용하시는 여러분은 이런 걱정은 안 하셔도 됩니다. 그런데, 주기가 2 초 이상인 아주 느린 PWM 신호를 만들어서 부저에 넣어 주고 싶을 때는 어떻게 하나 하시는 분이 계실 수 있습니다. 하지만 그렇게 주파수가 느린 신호는 부저에서 소리를 제대로 내지 못 합니다. 그래서 그런 신호는 부저에 아무 소용이 없습니다.

< 연습 문제 : **버튼이 눌릴 때만 부저 소리 내기** >

그런데 삐~~~ 소리가 너무 시끄럽죠. 그래서 버튼을 연결해 보겠습니다. 버튼이 눌릴 때만 소리를 냅니다.

```
#define BUZ 5
#define BUTTON 12

#define BUZ_OFF 0
#define BUTTON_PRESSED 0

void setup() {
  pinMode(BUZ, OUTPUT);
  pinMode(BUTTON, INPUT);
}
```

그런데, 코드가 setup() 부분만 있지요. 나머지 loop() 부분은 직접 작성해 보십시요. 여태까지 많이 해 왔던 버튼 관련 예제를 활용하면 매우 쉽게 하실 수 있을 것입니다. 꼭 직접 코딩해 보십시요. 그래야 실력이 향상됩니다.

< 코드 위치 : Coding Kit → **buz_but >**

PWM 신호의 1인 부분을 더 길게 해 주면 즉, 듀티비를 크게 하면, 부저를 울리는 소리가 더 커집니다. 그래서 버튼을 누르고 있으면 계속 소리가 커지는 예제를 해 보겠습니다.

```
#define BUZ 5
#define BUTTON 12
```

```
#define BUZ_OFF 0
#define BUTTON_PRESSED 0

void setup() {
  pinMode(BUZ, OUTPUT);
  pinMode(BUTTON, INPUT);
}

int buz_vol = 0;

void loop() {
  if (digitalRead(BUTTON) == BUTTON_PRESSED) {
    analogWrite(BUZ, buz_vol);
    if (buz_vol >= 200)
      buz_vol = 200;
    else
      buz_vol = buz_vol + 1;
  }
  else {
    analogWrite(BUZ, BUZ_OFF);
    buz_vol = 0;
  }
  delay(500);
}
```

< 문법 설명 : **전역 변수, 지역 변수** >

먼저 부저 볼륨으로 사용될 buz_vol 이란 변수를 선언합니다. 이 변수는 이전에는 다음과 같이 loop() 안에 변수를 선언한 것과는 다르게 loop() 밖에서 선언했습니다.

```
void loop() {
  int button_value;
  button_value = digitalRead(BUTTON);

  if (button_value == BUTTON_PRESS)
    digitalWrite(LED, ON);
  else
    digitalWrite(LED, OFF);
}
```

이 둘 사이에는 어떤 차이가 있을까요? 그럼 부저 예제에서도 변수를 loop() 안에 선언해 볼까요?

```
void loop() {
  int buz_vol = 0;

  if (digitalRead(BUTTON) == BUTTON_PRESSED) {
    analogWrite(BUZ, buz_vol);
    if (buz_vol >= 200)
      buz_vol = 200;
```

```
    else
      buz_vol = buz_vol + 1;
  }
  else {
    analogWrite(BUZ, BUZ_OFF);
    buz_vol = 0;
  }
  delay(500);
}
```

이렇게 코딩을 하면 부저에서 소리가 날까요? 안 날까요? 당연히 안 나겠죠. loop() 안을 계속해서 반복할 때 buz_vol 은 붉은색 코드가 수행되면서 계속해서 0 으로 됩니다. 그러니 "analogWrite(BUZ, buz_vol)" 에서 buz_vol 값이 0 이 되어 아무 소리도 나지 않게 되지요. 그래서 buz_vol 을 loop() 밖에 선언하는 것입니다. 이렇게 loop() 밖에 선언하는 것을 전역 변수 (Global Variable) 이하고 하고, loop() 안에 선언하는 것을 지역 변수 (Local Variable) 이라고 합니다.

그 다음 if 구문은 버튼이 눌렸는지를 체크합니다. 눌렸으면 buz_vol 값을 1 씩 증가시키는데, 200 이 넘어가면 buz_vol 값을 200 으로 고정합니다. 이렇게 200 이 못 넘어가게 하는 것은 다음 코드입니다.

```
if (buz_vol >= 200)
  buz_vol = 200;
else
  buz_vol = buz_vol + 1;
```

buz_vol 값은 최대 255 까지 사용할 수 있습니다. 그런데, 여기서 200 으로 막은 것은 부저에 PWM 신호를 0 과 1 값이 계속해서 반복되는 신호를 만들기 위한 것입니다. 계속 0 은 없는 1 만을 유지하는 값을 오랫동안 부저에 입력해 주면 부저가 손상을 입을 수가 있습니다. 그래서 이런 코드를 추가한 것입니다.

부저가 눌리지 않으면 다음 코드와 같이 "analogWrite(BUZ, BUZ_OFF)" 하여 부저에서 소리가 나지 않게 합니다. 그리고 buz_vol 에 0 (buz_vol = 0) 값을 주어 볼륨을 0 으로 합니다.

```
else {
  analogWrite(BUZ, BUZ_OFF);
  buz_vol = 0;
}
```

여기서 한 가지 더 알아두셔야 할 것은 부저의 볼륨은 buz_vol 변수 값이 0 ~ 10 정도일 때만 귀로 확연한 차이를 느낄 수 있고 그 이상은 거의 비슷합니다. 그래서 buz_vol 변수 값이 0 ~ 10 일 때만 부저 소리가 점점 커지는 것을 느낄 수 있고 그 이상의 값(11 ~ 200)에서는 부저 소리의 차이를 별로 못 느끼실 것입니다.

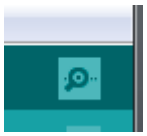
< 예제 코드 : **부저 볼륨 조절하기** >

< 코드 위치 : Coding Kit → **buz_vol_up >**

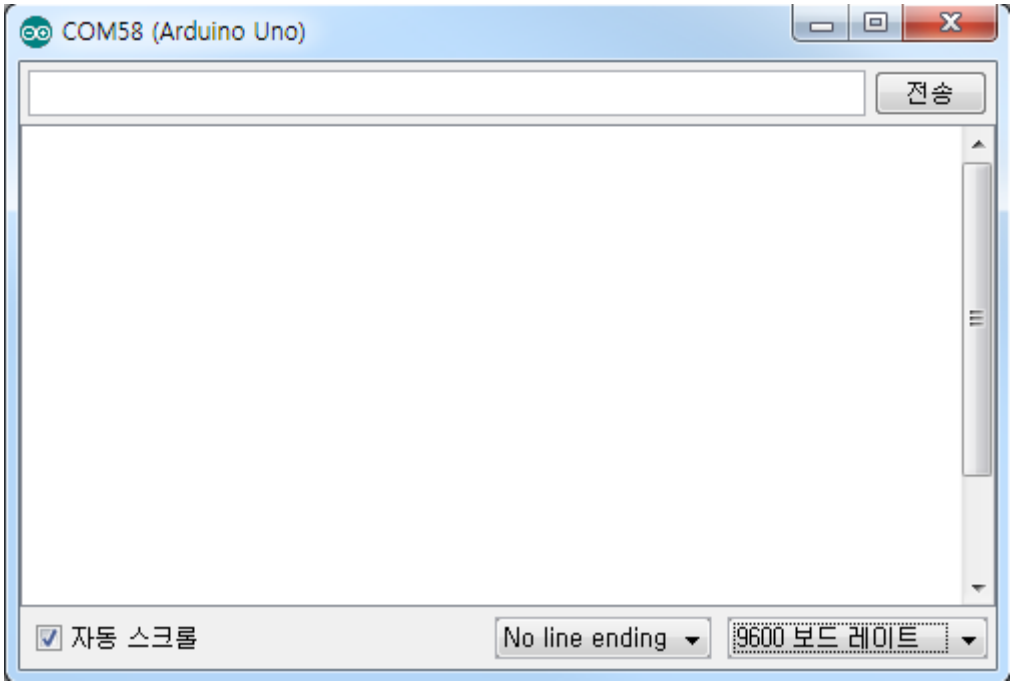
[아두이노 프로그램의 시리얼 모니터 활용하기]

위의 부저 예제 코드에서 변수 buz_vol 값이 변하면서 소리의 변화를 귀로 들었는데요. 이 값을 직접 눈으로 보면 더 좋겠지요. 혹시, 그런 방법이 있을까요? 있으면 좋겠지요? 있으면 좋겠다는 생각을 여러 사람이 했나 봅니다. 그래서 있습니다. 이제부터는 그 방법을 한 번 알아 보겠습니다.

아두이노 프로그램 오른쪽 상단에 보면 돋보기 모양이 있습니다.



이걸 시리얼 모니터 (Serial Monitor) 라고 하는데요. 한 번 눌러 볼까요?

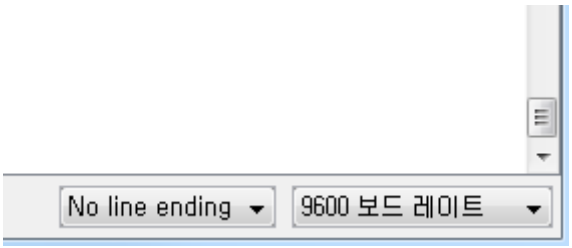


위와 같은 화면이 나오고요 다음과 같이 코딩하면 이 화면에 여러분이 원하시는 내용이 출력될 것입니다.

```
#define BUZ 5
#define BUTTON 12

#define BUZ_OFF 0
#define BUTTON_PRESSED 0

#define BUZ_VOL_MAX 10

void setup() {
  pinMode(BUZ, OUTPUT);
  pinMode(BUTTON, INPUT);
```

```
  Serial.begin(9600);
}

int buz_vol = BUZ_VOL_MAX / 2;

void loop() {
 if (digitalRead(BUTTON) == BUTTON_PRESSED) {
  analogWrite(BUZ, buz_vol);
  if (buz_vol >= BUZ_VOL_MAX)
   buz_vol = BUZ_VOL_MAX;
  else
   buz_vol = buz_vol + 1;
 }
 else {
  analogWrite(BUZ, BUZ_OFF);
  buz_vol = 0;
 }

 Serial.print("Volume : ");
 Serial.println(buz_vol);

 delay(500);
}
```

setup() 함수에서 "Serial.begin(9600)" 이라고 초기화를 해 줍니다. 이것은 내가 시리얼 모니터를 사용할 것이라고 초기화해 주는 것입니다. 이것은 아두이노와 PC 가 시리얼 통신이라는 통신 프로토콜로 연결하겠다고는 하는 것입니다. 괄호 안의 9600 이라는 숫자는 이 시리얼 통신의 속도입니다. 이 속도는 아래 그림과 같이 아두이노 프로그램 오른쪽 하단의 9600 보드 레이트와 맞춰 줘야 합니다.

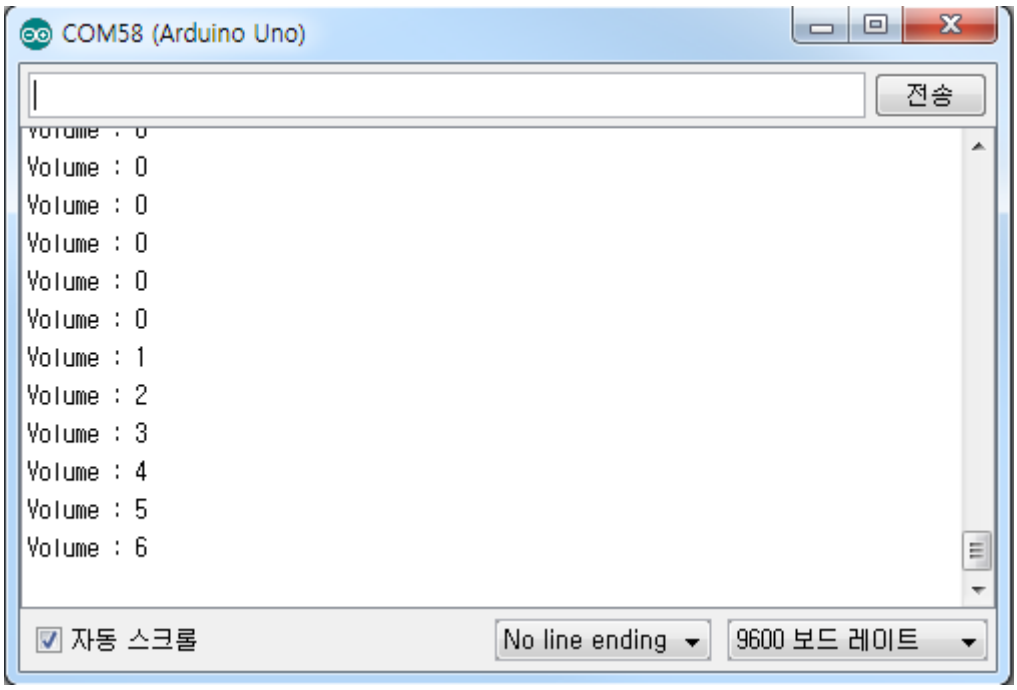


이 9600 보드 레이트를 클릭하면 속도를 바꿀 수 있습니다. 아두이노 프로그램에서 속도를 바꾸었다면 코드에서도 "Serial.begin(9600)" 에서 9600 이라는 숫자를 아두이노 프로그램의 속도 숫자와 맞추어 주어야 합니다.

"Serial.print()" 또는 "Serial.println()" 함수를 이용하여 시리얼 모니터로 출력하고 싶은 내용을 쓸 수 있습니다. "Serial.print()" 와 "Serial.println()" 의 차이점은 끝에 줄바꿈을 해 주느냐 안 해 주는냐 차이입니다. "Serial.println()" 은 줄 바꿈을 해 주고 "Serial.print()" 줄 바꿈을 해 주지 않습니다. 위의 코드를 실행하여 시리얼 모니터에 출력되는 값을 보면 더 잘 이해할 수 있을 것입니다.

아래의 그림을 보면 "Volume : " 다음에는 줄이 바뀌지 않았고 볼륨 값의 숫자 다음에는 줄이 바뀌어 새

줄에서 "Volume : " 이 출력됩니다.



그리고 다음과 같이 괄호 안의 값이 하나는 큰 따옴표 (" ") 로 싸여 있고, 하나는 그냥 쓰고 있습니다.

```
Serial.print("Volume : ");
Serial.println(buz_vol);
```

이 차이는 무엇일까요? 만약 위의 두 함수의 인자가 변수라면 큰 따옴표가 없어도 되고, 변수가 아니고 실제 시리얼 모니터로 출력할 문자라면 위와 같이 큰 따옴표로 감싸 줘야 합니다. 즉, 문자를 그대로 출력하고 싶으면 큰 따옴표를 사용합니다. 그러면 다음과 같이 하면 어떻게 될까요?

```
Serial.println("buz_vol");
```

그냥 buz_vol 이라고 출력이 될 것입니다. 그래서 변수를 출력하고 싶다면 큰따옴표 없이 변수 이름을 다음과 같이 바로 써 줍니다.

```
Serial.println(buz_vol);
```

이제 코드를 업로드하고 아두이노 프로그램에서 시리얼 모니터를 눌러 보십시요. 그럼 시리얼 모니터 창이 나타나고 계속해서 "Volum : 0" 이라고 찍힐 것입니다. 이 때 버튼을 눌러 주면 삐~~~ 소리가 나면서 위의 그림과 같이 시리얼 모니터 창에서 숫자가 올라가는 것이 보일 것입니다.

시리얼 모니터는 이렇게 사용하시면 됩니다. 이 시리얼 모니터는 지금과 같이 코드가 실행되고 있는 컴퓨터의 상태를 볼 수도 있고요. 코딩을 하였는데 코딩 의도대로 컴퓨터가 동작하지 않을 때 코드의 어디가 잘 못 되었는지를 찾는 것에도 많이 이용됩니다. 이렇게 코드의 잘 못 된 부분을 찾고 고치는 과정을 **디버깅**이라고 합니다. 위의 예제에서도 코드를 작성해서 업로드를 했는데, 소리가 안 난다고 해 봐요. 그럼 코

드의 어디가 잘 못 됐는지, 아니면 컴퓨터의 부저가 고장나서 소리가 안 나는지를 알 수는 없습니다. 이 때 시리얼 모니터를 이용하여 볼륨의 숫자가 제대로 나오는지를 볼 수 있습니다. 만약 이 숫자가 제대로 안 나오면 코드를 수정해야 하고, 숫자는 잘 나오는데 소리가 안 난다면 컴퓨터의 부저가 고장이 났는지를 의심해 볼 수 있습니다.

이번에는 부저의 볼륨은 올렸다 내렸다 하는 예제를 한번 해 보겠습니다. 버튼 1 번을 누르면 볼륨이 올라가고 0 번을 누르면 볼륨이 내려가는 예제를 해 보겠습니다. **볼륨의 값을 바꾸는 변수 buz_vol 은 값이 0 ~ 10 까지만 변하는 것으로 하십시요.** 그래야 코딩키트에서 실행했을 때 소리의 변화를 잘 느낄 수 있습니다.

```
#define BUZ 5
#define BUT_VOL_DOWN 12
#define BUT_VOL_UP 13

#define BUZ_OFF 0
#define BUTTON_PRESSED 0

void setup() {
  pinMode(BUZ, OUTPUT);
  pinMode(BUT_VOL_DOWN, INPUT);
  pinMode(BUT_VOL_UP, INPUT);
}

int buz_vol = 5;
```

두개의 버튼을 BUT_VOL_DOWN 과 BUT_VOL_UP 으로 정의합니다. 그리고 "int buz_vol = 5" 와 같이 중간 값으로 초기화합니다. 이 이후의 loop() 코드를 작성해 보십시요.

< 예제 코드 : **부저의 볼륨 올리고 내리기** >

< 코드 위치 : Coding Kit → **buz_vol_up_down >**

< 문법 설명 : **함수(1)** >

지금까지 PWM 신호를 만들기 위해서 analogWrite() 를 사용하였는데요. 이러한 것들을 함수라고 합니다. 함수라는 것은 수학 시간에 배운 함수와 비슷합니다. 어떤 코드를 실행하는 실행 단위 정도로 알아 두시면 됩니다. 그리고 이렇게 함수를 만들어 두면 코드의 여기 저기서 가져다가 사용할 수 있습니다. 아두이노 프로그램을 처음 실행 시키면 보이는 "void setup() { ... }" 과 "void loop() { ... }" 도 함수입니다. 다음은 함수를 이용하여 버튼을 눌러 부저 소리의 크기를 조절하는 예제입니다. 위의 예제와 비슷한데, 특이 사항은 버튼을 모두 누르지 않으면 소리가 서서히 줄어드는 것입니다.

```
#define BUZ 5
#define BUT_VOL_DOWN 12
#define BUT_VOL_UP 13

#define BUZ_OFF 0
#define BUTTON_PRESSED 0

void setup() {
  pinMode(BUZ, OUTPUT);
  pinMode(BUT_VOL_DOWN, INPUT);
  pinMode(BUT_VOL_UP, INPUT);
}

int buz_vol = 5;

void buz_vol_up () {
    analogWrite(BUZ, buz_vol);
    if (buz_vol >= 10)
      buz_vol = 10;
    else
      buz_vol = buz_vol + 1;
}

void buz_vol_down () {
    analogWrite(BUZ, buz_vol);
    if (buz_vol <= 0)
      buz_vol = 0;
    else
      buz_vol = buz_vol - 1;
}

void loop() {
  if (digitalRead(BUT_VOL_UP) == BUTTON_PRESSED) {
    buz_vol_up();
  }
  else if (digitalRead(BUT_VOL_DOWN) == BUTTON_PRESSED) {
    buz_vol_down();
  }
  else if ((digitalRead(BUT_VOL_UP) != BUTTON_PRESSED) &&
          (digitalRead(BUT_VOL_DOWN) != BUTTON_PRESSED)) {
    buz_vol_down();
  }
  delay(500);
}
```

위의 코드에서 buz_vol_up() 함수와 buz_vol_down() 함수를 이용하였습니다. 예제를 보시면 이전에 loop() 안에 있던 코드를 함수의 형태를 갖춰 밖으로 뺀 것입니다. 이렇게 함수의 내용을 기술해 두는 것을 "함수 정의" 라고 합니다.

loop() 함수 안에서 buz_vol_up() 과 buz_vol_down() 함수를 사용했습니다. 이렇게 어떤 함수를 사용하는

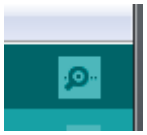
것을 불러 쓴다는 의미로 "함수 호출" 이라고 합니다.

위의 코드에서 보면 buz_vol_down() 함수를 BUT_VOL_DOWN 이 눌렸을 때와 버튼 둘 모두 안 눌렸을 때에 두 번 사용했습니다. 이것이 함수의 장점입니다. 여러 번 코딩해야 할 것을 이렇게 한 번만 코딩하고 여러 곳에서 불러다 쓰면 편리하게 코딩할 수 있겠지요. 함수는 조금 어려운 부분이니 추후에 자세히 더 알아보고 일단은 이렇게 사용한다는 정도만 알고 넘어 가겠습니다.

< 예제 코드 : **함수를 이용하여 부저 볼륨 올리고 내리기** >

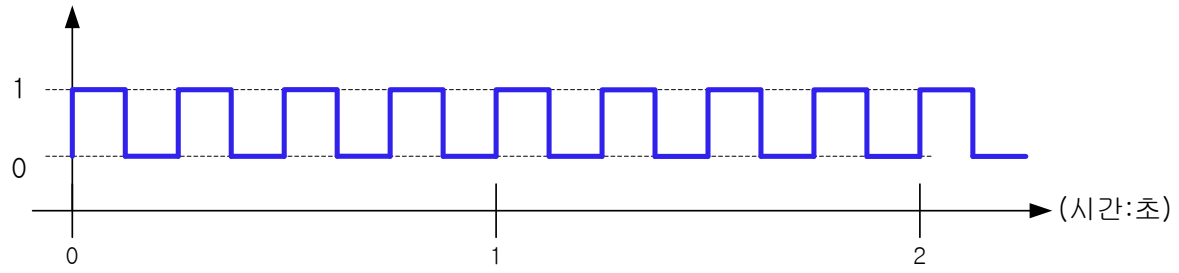
< 코드 위치 : Coding Kit → **buz_func >**

Note : 아두이노의 시리얼 모니터를 실행하면 리셋 스위치를 누른 것과 같이 처음부터 수행을 합니다. 즉 아래 그림의 시리얼 모니터 아이콘을 클릭하면 시리얼 모니터가 실행되면서 setup() 함수의 처음부터 다시 실행이 됩니다.



[주파수]

PWM 파형이 1 초에 몇 번 반복되느냐를 주파수라고 합니다. 그리고 그 반복되는 횟수를 헤르츠(Hertz) 라고 하고 Hz 로 표시합니다. 다음 그림과 같이 1 초에 4 번 반복이 되면 주파수는 4 Hz 입니다.



그러면 주기가 0.001 초인 신호는 주파수가 몇 Hz 일까요? 1 나누기 0.001 초 하면 1000 이 되니깐 1000 Hz 입니다. 이것을 1 kHz (킬로헤르츠) 라고도 합니다. 주파수를 비롯한 컴퓨터의 숫자들은 1000 을 단위로 한 이름이 있습니다. 그것은 다음 표와 같습니다.

이름	기호	값	10의 승수
테라 (Tera)	T	1,000,000,000,000	10^{12}
기가 (Giga)	G	1,000,000,000	10^{9}
메가 (Mega)	M	1,000,000	10^{6}
킬로 (Kilo)	k	1,000	10^{3}
-	-	0	0
밀리 (Milli)	m	0.001	10^{-3}
마이크로 (Micro)	μ (u)	0.000001	10^{-6}
나노 (Nano)	n	0.000000001	10^{-9}
피코 (Pico)	p	0.000000000001	10^{-12}

코딩키트에 장착된 아두이노의 PWM 핀 중 5, 6 번 핀은 주기가 0.002 초이므로 500 Hz 이고 나머지 PWM 신호 핀들은 주기가 0.004 초이므로 250 Hz 입니다. 그런데, 실제로는 이것과는 조금 차이가 있어 각각 490 Hz, 245 Hz 입니다.

지금까지는 부저에 입력되는 PWM 신호의 듀티비만을 조정하여 소리의 크기를 바꾸어 보았습니다. 이제는 PWM 신호의 주파수를 바꾸어 보겠습니다. 그러면 소리가 어떻게 변하는지 알아 보겠습니다. 그런데, 여기에 analogWrite() 함수를 쓰는 것은 더 이상 무리입니다. 왜냐하면, analogWrite() 함수를 사용하면 주파수는 490 Hz 와 245 Hz 로 고정이 됩니다. 즉, 주파수를 바꿀 수 없다는 말입니다. 그래서 아두이노에서 제공하는 tone() 이라는 함수를 사용하겠습니다.

사람이 들을 수 있는 주파수는 20 ~ 20,000 Hz 입니다. tone() 함수를 이용하여 낮은음에서 높은음으로

변하는 코딩을 해 보겠습니다.

```
#define BUZ 5

void setup() {
  pinMode(BUZ, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  for (int i = 20; i < 4000; i = i + 100) {
    tone(BUZ, i, 500);
    Serial.print("Freq : ");
    Serial.println(i);
    delay(500);
  }
  delay(3000);
}
```

먼저 tone 함수를 설명하겠습니다. tone 함수는 3 개의 인자를 받습니다. 첫번째는 어떤 핀으로 PWM 신호를 출력할 것이냐이고 두번째 인자는 주파수를 나타냅니다. 세번째 인자는 이 신호를 얼마나 오랫동안 출력할 것이냐 하는 것입니다. 즉, 얼마나 오랫동안 소리를 낼 것이냐 하는 이야기죠. 이 시간의 단위는 0.001 초입니다. 위의 코드 "tone(BUZ, i, 500)" 을 보면 BUZ 에 i 라는 주파수로 0.5 초간 소리를 내겠다는 뜻입니다. i 값은 for 문의 인덱스입니다. 이 인덱스는 20 에서 시작해서 100 씩 증가하여 4000 까지 변합니다. 정확히 4000 은 아니고 3920 입니다. 그 이유는 여러분 스스로 생각해 보세요. 별로 어려운 이야기는 아닙니다.

그래서 tone() 함수의 주파는 20 Hz ~ 4000 Hz 까지 변합니다. 그런데, 앞에서 사람이 들을 수 있는 소리는 20 Hz ~ 20,000 Hz 라고 했는데, 왜 4000 Hz 까지 일까요. 그것은 코딩키트에 장착되어 있는 부저의 특성이 4000 Hz 까지만 소리를 낼 수 있게 되어 있어서 그럽니다. 물론 이 이상으로 PWM 신호를 출력할 수 있지만 그렇게 하면 소리가 이상하게 들릴 것입니다. 그리고 계속해서 그렇게 무리하게 사용하면 부저가 망가질 수도 있습니다. 4000 Hz 로 실제 소리를 들어 보면 매우 음이 높은 소리라는 것을 알 수 있을 것입니다.

< 예제 코드 : **부저 입력 신호의 주파수를 바꾸어 음의 높낮이 변경하기** >

< 코드 위치 : Coding Kit → **buz_tone >**

코드에서 이 주파수 값을 시리얼 모니터로 볼 수 있도록 해 두었습니다. 이제 코드를 업로드해 볼까요? 소리가 잘 나오나요? 시리얼 모니터 아이콘을 클릭하여 주파수 값을 관찰할 수도 있습니다. 그러면 어떤 주파수에서 어떤 소리가 나는지 알 수 있을 것입니다. 소리를 들어보면 낮은 음에서 계속해서 높은음으로 올라가지요? 끝에 가서는 마치 이렇게 음이 올라가다가 꼭 터질 것만 같습니다. 걱정하지 마세요. 터지지

는 않아요. 그리고 이렇게 계속해서 시끄러운 소리가 반복되서 나오면 끄고 싶죠. 끄고 싶으시면 + 나 - 버튼을 누르세요. 그러면 세븐세그먼트가 깜박이면서 부저로 연결된 선을 끊기 때문에 더 이상 소리가 나지 않습니다. 연결을 다시 하고 싶으면 다음과 같이 합니다.

< 연습 문제 : **버튼을 누르면 음이 올라가는 부저 소리** >

위와 같은 예제에서는 부저 소리가 계속 나니깐 너무 시끄럽죠? 그래서 이번에는 버튼을 눌렀을 때만 소리가 나는 예제를 해 보겠습니다. 그런데, 버튼을 눌러 소리가 날 때는 계속해서 음이 올라가고 있어야 합니다. 즉, 버튼을 눌러 소리를 1000 Hz 까지 만들었다가 버튼을 떼면 소리가 나지 않다가 다시 버튼을 눌러 소리가 날 때는 1000 Hz 부터 시작해야 한다는 뜻입니다.

그리고 음의 최대는 4000 Hz 를 넘지 않도록 합니다. 4000 Hz 가 넘어 가려고 하면 다시 20 Hz 에서 시작합니다. 핀 정의와 초기화는 다음과 같습니다. loop() 함수의 코드를 작성해 보세요.

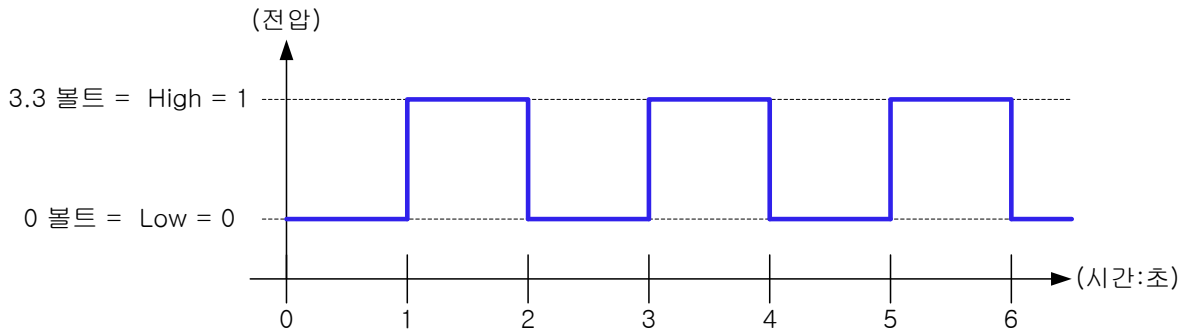
```
#define BUZ 5
#define BUTTON 12
#define BUTTON_PRESSED 0

void setup() {
  pinMode(BUZ, OUTPUT);
  pinMode(BUTTON, INPUT);
  Serial.begin(9600);
}

int i = 20;

void loop() {
}
```

< 코드 위치 : Coding Kit → **buz_tone_but** >

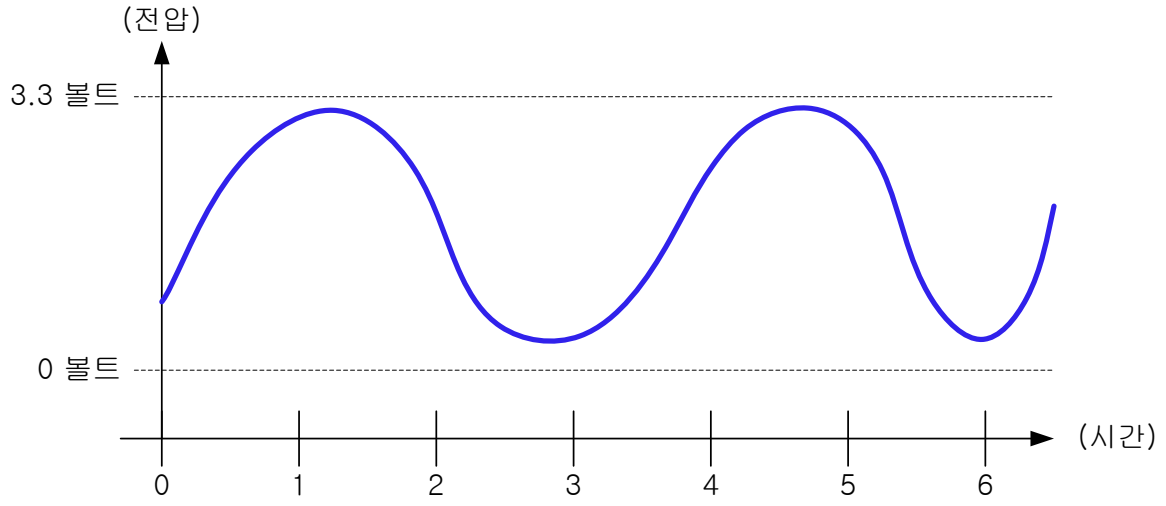
[디지털 VS 아날로그]

지금까지는 여러분은 디지털 신호만을 이용하는 코딩을 하였습니다. 아두이노에서는 PWM 신호도 아날로그 출력이라고 하는데, 그걸 아날로그라고 하기에는 좀 많이 부족합니다. 그럼 이제 아날로그 신호에 대해서 공부해 보겠습니다. 우리 아직 디지털도 안 배운 것 같은데요? 하시는 분들 계실 것 같은데, 아날로그를 공부하다보면 자연스럽게 디지털은 알게 되니 걱정 마세요. 그리고 지금까지 여러분이 0 또는 1, High 또는 Low, ON 또는 OFF, true 또는 false 했던 것들이 모두 디지털 신호입니다.

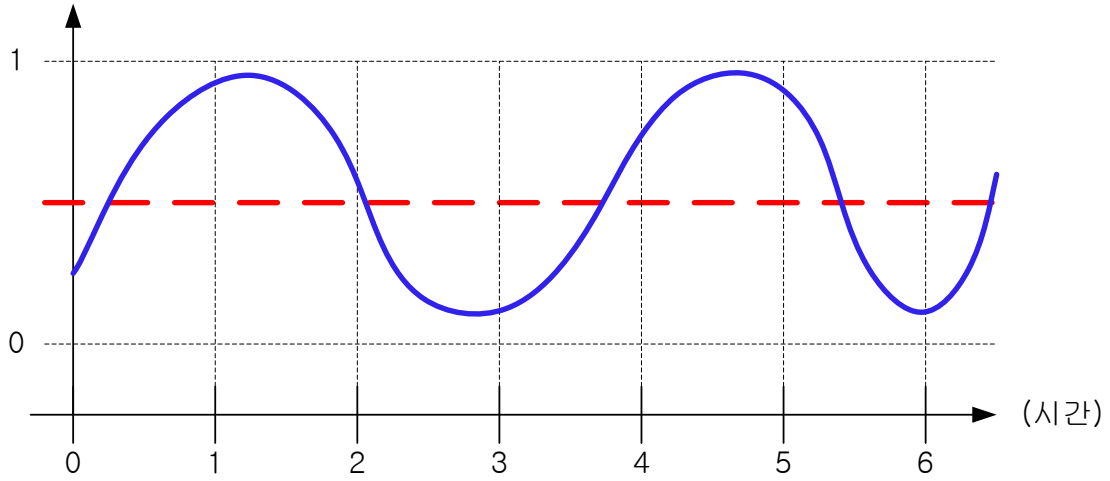
그러면 본격적으로 아날로그에 대해서 알아보겠습니다. 아날로그는 "길이", "각도" 와 같이 연속적으로 변하는 값을 말할 때 주로 사용 됩니다. 그래서 자동차의 속도를 바늘의 각도로 표시해 주는 속도계, 온도를 수은주의 길이로 표시하는 온도계 등을 아날로그 속도계, 아날로그 온도계라고 부르기도 합니다. 이와는 반대로 숫자로 자동차의 속도를 표시하거나 온도를 표시하는 것을 디지털 속도계, 디지털 온도계라고 부릅니다. 여러분은 지금까지 다음 그림과 같이 0 V 와 3.3 V 를 왔다 갔다 하는 신호의 파형을 주로 보았습니다. 이런 신호를 디지털 신호라고 하지요.



아날로그 신호의 파형은 다음과 같습니다.



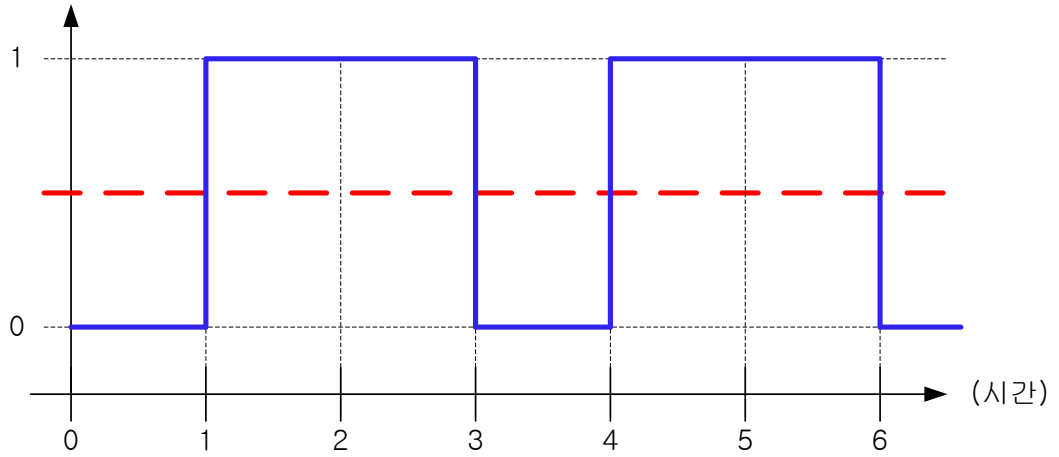
위 그림에서 신호가 연속적으로 변하는 것을 볼 수 있습니다. 그러면 이렇게 변하는 신호들을 어떻게 이용하여 코딩을 할 수 있을 까요? 코딩을 할 때는 모든 신호 값을 숫자로 표시해야 합니다. 그래서 위의 신호도 다음 그림과 같이 숫자로 바꿉니다.



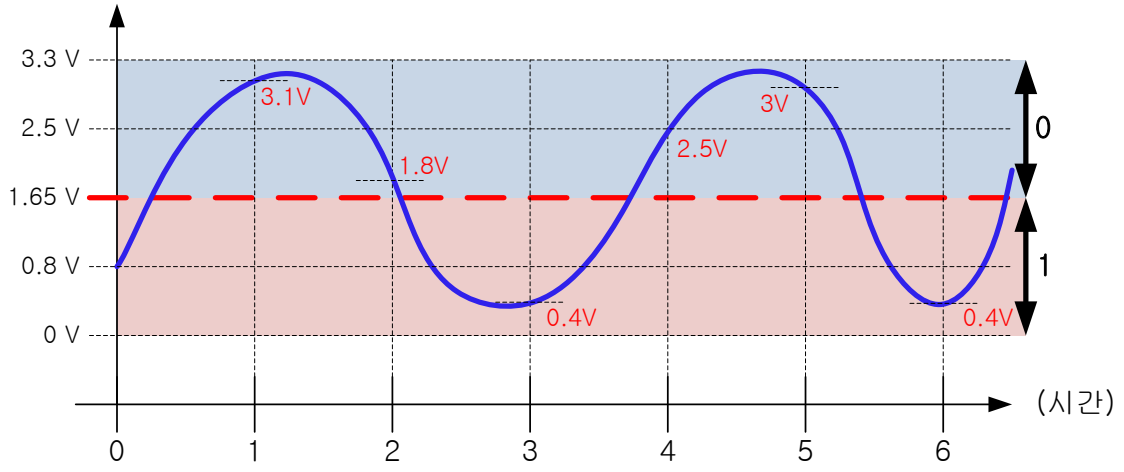
신호가 바뀌어지는 숫자는 0 과 1 입니다. 그래서 신호가 붉은색 점선 위쪽으로 올라가면 1 이고 신호가 붉은색 점선 아래로 내려오면 0 입니다. 그래서 각 시간마다 신호의 값은 다음과 같습니다.

시간	값
0	0
1	1
2	1
3	0
4	1
5	1
6	0

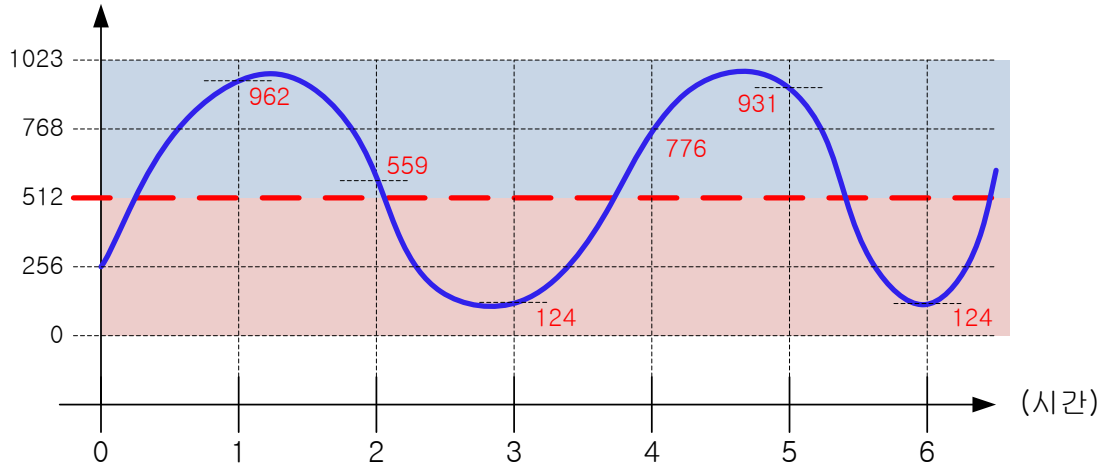
그러면 이 표의 값을 파형으로 그려 볼까요.



위의 그림은 주로 우리가 보아왔던 그림인데요. 바로 디지털 신호입니다. 이렇게 하면 아날로그 신호를 디지털 신호로 바꿀 수가 있고, 이렇게 바뀐 신호를 코딩에 활용하면 됩니다. 그런데 한가지 좀 아쉬운 것은 아날로그 신호를 디지털 신호로 바꾸는데, 바뀌어진 값이 너무 단순화 된 것 같습니다. 무슨 얘기냐면, 아날로그 신호를 디지털 신호로 바꿀 때 0 볼트에서 3.3 볼트 사이에 아랫쪽 반은 디지털 신호의 0 으로 바꾸고, 위쪽 반은 디지털 신호의 1 로 바꿉니다. 그림과 같이 중간 값 1.65 볼트의 위쪽인 3.1 볼트, 1.8 볼트, 2.5 볼트, 3 볼트는 모두가 다 1 인 것입니다. 반대로 중간 값 1.65 볼트 아래의 값인 0.8 볼트, 0.4 볼트는 모두가 다 0 인 것이지요. 이렇게 많은 값들이 단순하게 0 과 1 로 변해 버리는 것입니다.



만약 어떤 신호를 보내는 사람이 3.1 볼트와 1.8 볼트를 다른 의미의 신호로 사용했는데, 받는 사람이 이 신호를 모두 디지털 신호 1 로 만들어 버린다면 3.1 볼트와 1.8 볼트의 다른 의미의 신호를 구분할 수가 없어져 버립니다. 그래서 아날로그 신호는 다음 그림과 같이 좀 더 세분화하여 사용합니다.



아날로그 0 볼트에서 3.3 볼트까지의 값을 디지털 0 에서 1023 까지 값으로 바꾼 것입니다. 여기서 1023 은 2 진수 10 자리로 표시할 수 있는 최대 값입니다. 2 진수 10 자리를 컴퓨터 용어로 10 비트라고 합니다. 아날로그 신호를 0 ~ 1023 의 디지털 값으로 변환한다면 3.1 볼트와 1.8 볼트를 구분할 수 있습니다. 이

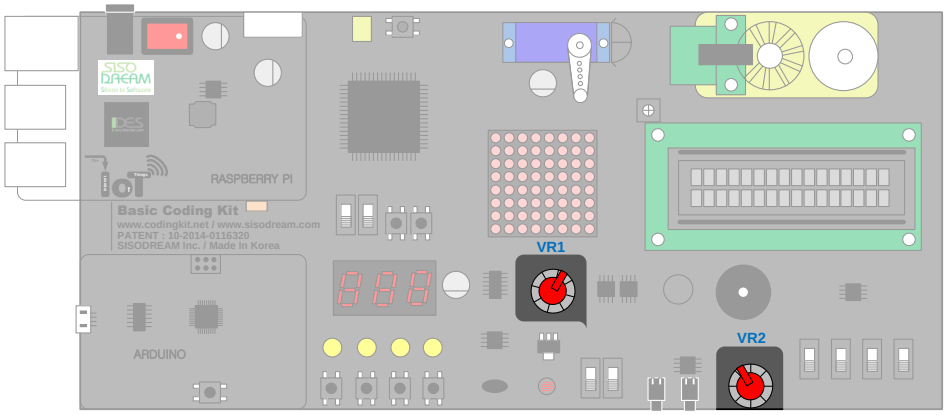
값들을 표로 표시해 보겠습니다.

시간	아날로그 값	디지털 값 (0, 1)	디지털 값 (0 ~ 1023)
0	0.8 V	0	256
1	3.1 V	1	962
2	1.8 V	1	559
3	0.4 V	0	124
4	2.5 V	1	776
5	3 V	1	931
6	0.4 V	0	124

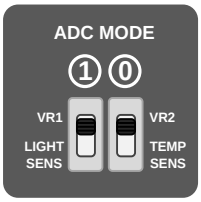
위의 표를 보면 단순히 0 과 1 로 바꾼 것보다 0 ~ 1023 으로 바꾼 것이 신호를 훨씬 더 정밀하게 바꾼 것입니다. 그럼 이제 코딩키트를 활용하여 아날로그 신호를 입력 받는 코드를 작성해 보겠습니다. 아두이노는 아날로그 값 입력을 받는 전용 핀이 있습니다. A0 ~ A5 입니다. 핀 번호에 붙은 "A" 는 아날로그(Analog)의 "A" 입니다. 그렇다고 해서 이 핀들은 디지털로 사용할 수 없는 것은 아닙니다. 디지털 핀으로도 사용이 가능합니다. 그래서 코딩키트에서도 A0 ~ A3 만 아날로그 입력 핀으로 사용하고 A4, A5 는 디지털 입력으로 사용합니다. 그렇다면 디지털 핀을 아날로그 핀으로 사용할 수 있을까요? 아쉽게도 그렇게는 할 수 없습니다. 이유는 아날로그 신호를 받아서 사용하려면 ADC(Analog to Digital Converter)라는 아날로그 신호를 디지털 신호로 바꾸는 내부 장치가 있어야 합니다. 아두이노의 아날로그 핀 내부에는 ADC 가 장착되어 있어 아날로그 신호를 받아서 사용할 수가 있는 것입니다. 디지털 핀에는 ADC 가 없습니다. 그래서 아날로그 핀으로 사용할 수가 없습니다.

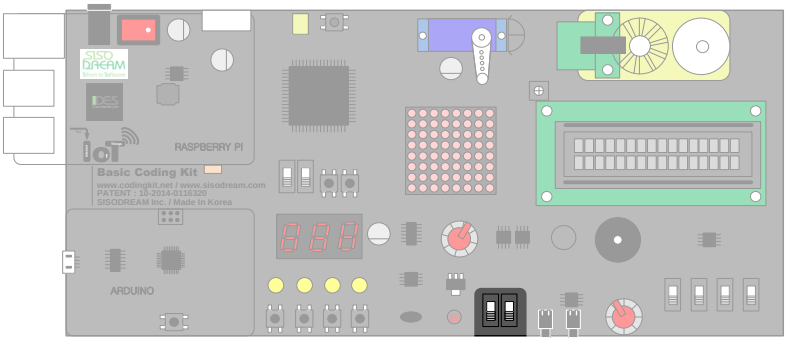
이 ADC 라는 장치가 위에서 설명한 0 볼트에서 3.3 볼트까지 변하는 값을 0 에서 1023 값으로 바꾸어 주는 것입니다. 아두이노 칩의 아날로그로 사용되는 핀에는 모두 이 ADC 장치가 장착되어 있습니다. 이 ADC 에 대해서는 앞으로 배우게 될 많은 코드 예제를 활용하면 훨씬 더 이해가 쉬울 것입니다. 그러니 아직 완벽하게 이해가 되지 않았다고 해서 걱정할 것은 없습니다. 그럼 이제 예제를 해 볼까요?

코딩키트에는 다음 그림과 같이 다이얼을 돌리는 장치가 있습니다. 이 장치를 "가변저항 (Variable Resistor)" 이라고 합니다. 이 가변저항이라는 것은 말 그대로 저항 값을 바꾸는 것입니다. 가변저항은 아두이노로 입력되는 전압 값을 바꾸는 장치 정도로만 알아 두십시요. 즉, 0 볼트에서 3.3 볼트의 아날로그 값을 아두이노로 입력해 주는 장치로 알고 있으면 됩니다. 이 값이 아두이노의 아날로그 핀에 장착된 ADC 에 의해서 0 ~ 1023 의 값으로 바뀌는 것입니다. 코딩키트에는 그림과 같이 가변 저항이 2 개가 있습니다. 각각 VR1, VR2 라고 합니다.



가변저항을 사용하기 위해서는 코딩키트의 ADC Mode 스위치를 모두 위로 올려 줍니다.



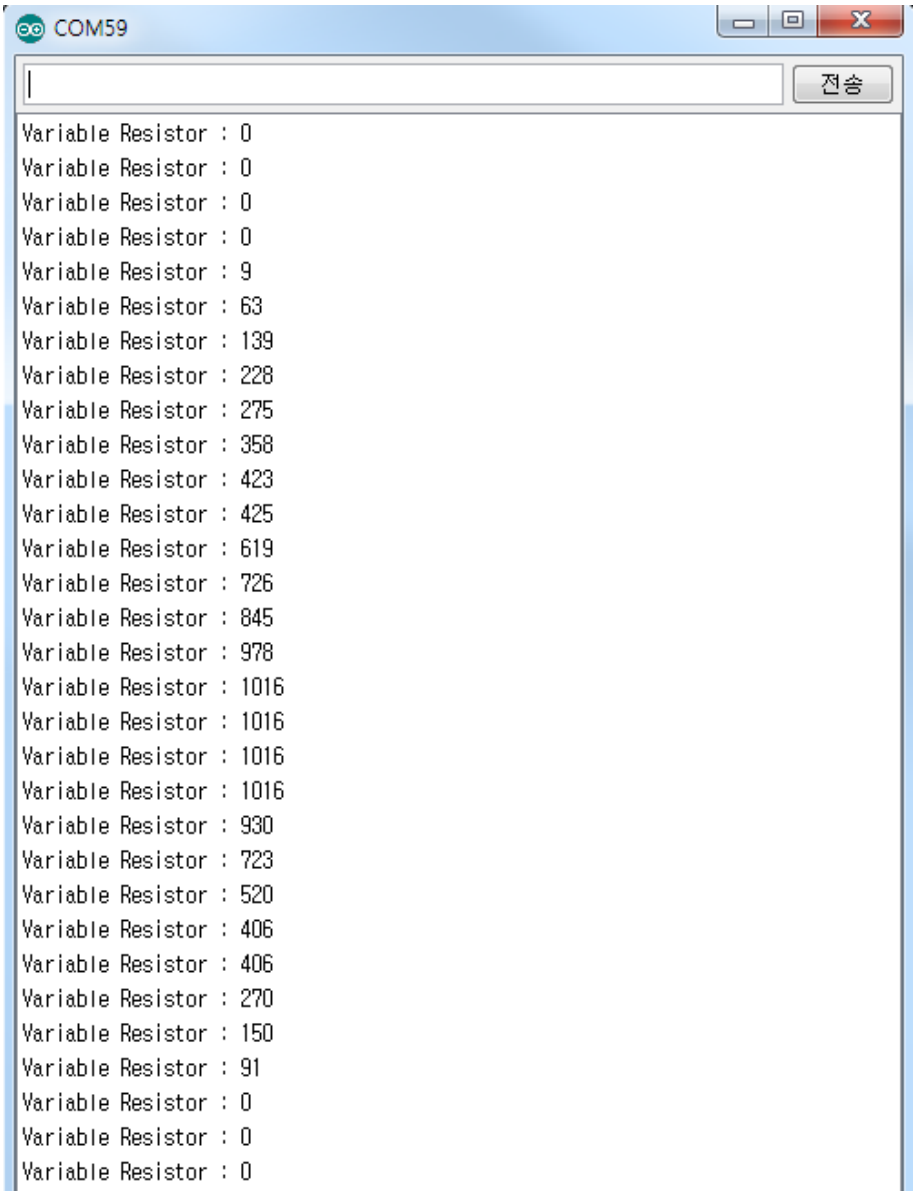


다음과 같이 가변저항에서 입력된 아날로그 값을 시리얼 모니터로 출력하는 코드를 작성해 보겠습니다.

```
#define VAR_RES_1 A3

void setup() {
  Serial.begin(9600);
}

void loop() {
  int val_var_res_1 = analogRead(VAR_RES_1);
  Serial.print("Variable Resistor : ");
  Serial.println(val_var_res_1);
  delay(500);
}
```

아날로그 핀 A3 에 가변저항이 연결되어 있습니다. 이 값을 analogRead() 함수를 이용하여 val_var_res_1 변수에 저장합니다. 저장된 값을 시리얼 모니터로 출력합니다.

이제 코드를 업로드합니다. 가변저항 1 번을 오른쪽으로 돌리면 값이 커지고 왼쪽으로 돌리면 값이 작아집니다. 시리얼 모니터로 보면 다음과 같습니다.



< 예제 코드 : **가변 저항에서 입력되는 아날로그 값 모니터 하기 >**

< 코드 위치 : Coding Kit → **vr_mon >**

가변저항 값은 최대가 1023 이라고 했는데, 시리얼 모니터로 값을 보면 최대값이 1016 입니다. 이것은 가변저항으로는 전압을 3.3 볼트까지는 올리지 못하고 약간 낮게 올라가기 때문에 1016 까지 나오는 것입니다. 약간의 오차라고 생각해 주시고 이것에 맞게 코딩해 주시면 됩니다. 그래서 최대값을 1016 으로 생각하고 코딩해 주셔도 되고, 크게 문제가 되지 않는다고 생각하면 그냥 1023 을 최대값으로 하여도 됩니다. 그 때 그 때 사정에 맞게 코딩하시면 됩니다. 1016 이라는 이 값은 보드마다 약간의 편차가 있을 수 있습니다. 즉, 여러분이 사용하시는 보드는 최대값이 1016 보다 약간 클수도 있고 작을 수도 있습니다. 이 값들을 시리얼 모니터로 관찰한 후에 알맞게 보정해 주시면 됩니다. 실제 산업계에서도 하드웨어의 오차를 소프트웨어로 보정하는 일이 빈번하게 있습니다. 그리고 그 보정을 얼마나 잘 하느냐도 엔지니어의 역량입니다.

여기서 이 1023 값이 나와야 하는데 1016 이 나오는 오차가 실제 전압으로는 얼마는 되는지 계산해 볼까요? 먼저 3.3 볼트를 1024 로 나누면 0.0032 볼트입니다. 1023 - 1016 은 7 입니다. 그래서 7 곱하기 0.0032 볼트하면 0.0225 볼트가 됩니다. 아주 미세한 오차임을 알 수 있습니다.

이렇게 다이얼 식으로 된 가변저항을 보니 어떤 예제가 하고 싶습니까? 부저의 소리를 조절해 보고 싶지요. 저도 그렇습니다만, 그것보다는 부저의 주파수를 바꾸어 음의 높낮이를 조절해 보는 것을 좋을 것 같습니다. 이유는 부저의 볼륨 값이 0 ~ 10 정도만을 움직이기 때문에, 그 보다는 훨씬 더 많이 움직이는 주파수가 알맞을 것 같습니다. 그래서 다음과 같이 코딩해 보았습니다.

```
#define BUZ 5
#define VAR_RES_1 A3

void setup() {
  Serial.begin(9600);
}

void loop() {
  int val_var_res_1 = analogRead(VAR_RES_1);
  tone(BUZ, val_var_res_1, 500);
  Serial.print("Variable Resistor : ");
  Serial.println(val_var_res_1);
  delay(500);
}
```

가변저항의 값을 읽어서 그대로 tone() 함수에 넣어 주기만 하면 간단하게 끝납니다. 쉽죠? 그럼 이 코드를 업로드 하여 코딩키트에서 실행해 보시겠어요. 그리고 가변저항을 이리 저리 돌려 보십시요. 그러면 부저의에서 음의 높낮이가 변하는 소리가 들릴 것입니다.

< 예제 코드 : **가변 저항을 이용하여 부저 음의 높낮이 조절하기** >

< 코드 위치 : Coding Kit → **vr_buz_frq >**

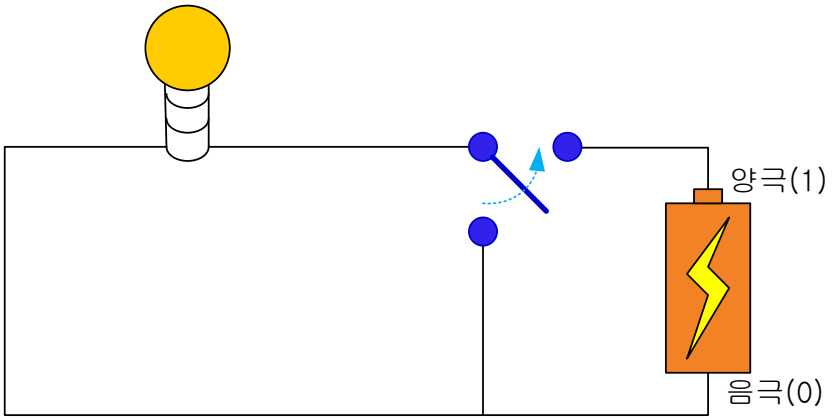
< 문법 설명 : **2 진수, 8 진수, 10 진수, 16 진수 그리고 비트, 바이트** >

사람은 10 진수를 사용합니다. 수를 셀 때 0 ~ 9 까지 세고 그리고 한 자리 올라가서 10부터 세는 것을 10 진수라고 합니다. 수를 셀 때 0 ~ 9 까지 수만 사용하는 것이지요. 그래서 2 진수는 0 과 1 만을 사용하여 수를 셉니다. 8 진수는 0 ~ 8, 16 진수는 0 ~ 15 까지를 이용하여 수를 셉니다. 그런데, 16 진수의 10 ~ 15 는 어떻게 표시를 할까요? 이것은 A ~ F 로 표시를 합니다. 그래서 2, 8, 10, 16진수는 다음 표와 같이 수를 셉니다.

2 진수	8 진수	10 진수	16 진수
0	0	0	0
1	1	1	1
10	2	2	2
11	3	3	3
100	4	4	4
101	5	5	5
110	6	6	6
111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	10	A
1011	13	11	B
1100	14	12	C
1101	15	13	D
1110	16	14	E
1111	17	15	F
1000	20	16	10
1001	21	17	11
1010	22	18	12
1011	23	19	13
1100	24	20	14

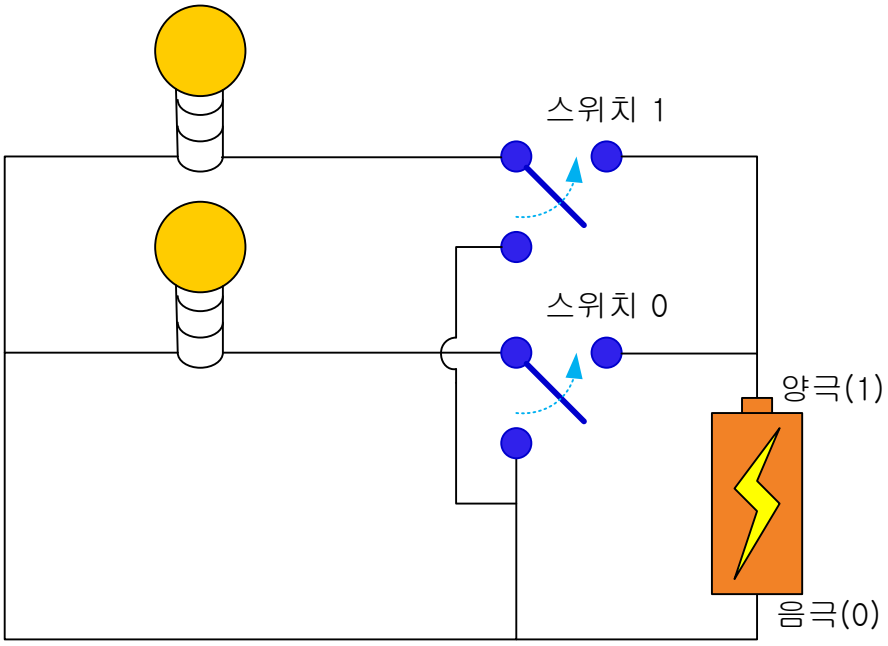
오래 전부터 사람의 손가락이 10 개라는 이유로 사람들은 10진수를 매우 익숙하게 사용하였습니다. 그런데 컴퓨터에서는 논리적인 이분법과 잘 맞아 떨어지는 2 진수가 매우 적합합니다. 또한 전기적인 소자의

특성으로서도 2 진수가 적합니다. 이 부분에 대해서는 좀 더 설명해 볼까요? 전에 우리가 봤던 다음 그림을 다시 보겠습니다.



위의 그림에서 전구에는 스위치의 연결에 따라서 값이 1 혹은 0 이 입력됩니다. 이것이 바로 2 진수의 한 자리 입니다. 2 진수의 한 자리를 비트(Bit)라고 부릅니다.

그럼 다음 그림과 같이 스위치가 2 개가 있는 것은 어떨까요?



이 스위치 2 개로 표시할 수 있는 2 진수 수는 다음과 같습니다.

스위치 1	스위치 0	2 진수	10 진수
0	0	00	0
0	1	01	1
1	0	10	2

1	1	11	3

10 진수로는 0 ~ 3 까지입니다. 이렇게 스위치가 늘어날수록 표현할 수 있는 수는 늘어납니다. 이것을 비트수가 늘어날수록 표현할 수 있는 수는 늘어난다고 할 수 있습니다. 비트수에 따라 표현할 수 있는 최대값은 다음과 같습니다.

비트수	2 진수	10 진수	16 진수
1	1	1	1
2	11	3	3
3	111	7	7
4	1111	15	F
5	11111	31	1F
6	111111	63	3F
7	1111111	127	7F
8	11111111	255	FF
9	111111111	511	1FF
10	1111111111	1023	3FF
16	...	65,535	FFFF
32	...	4,294,967,295	FFFFFFFF

이 값에 더하기 1 을 하면 각 비트에서 표현할 수 있는 수는 개수가 나옵니다. 즉, 2 비트인 경우에는 0, 1, 2, 3 을 표현할 수 있습니다. 여기서 최대값은 3 이고 여기에 1 을 더한 4 는 표현할 수 있는 수의 개수입니다. 즉, 0, 1, 2, 3 이렇게 4 개라는 것입니다.

8 비트를 1 바이트라고 합니다. 그래서 16 비트는 2 바이트, 32 비트는 4 바이트라고 합니다.

위의 표에서 익숙한 숫자 몇 개가 있을 것입니다. 여러분이 analogWrite() 함수를 사용할 때 두 번째 인자로 넘겨주는 값의 범위는 0 ~ 255 였습니다. 이 말은 두 번째 인자는 8 비트를 사용한다는 뜻입니다. 앞에서 ADC 의 범위는 0 ~ 1023 이라고 하였습니다. 이것은 10 비트를 사용한다는 뜻입니다. 이제 왜 앞의 코드들에서 255, 1023 이런 이상한 숫자를 사용하는지를 알겠지요. 이것이 바로 컴퓨터에서 사용하는 2 진수 숫자들인 것입니다. 여기서 한 가지 더 알아두어야할 것은 16 진수입니다. 사실 컴퓨터는 2 진수를 사용하기 때문에 16 진수가 의미가 없습니다. 하지만 사람이 코딩하는 것이기 때문에 16 진수를 사용하는 것입니다. 위의 표에서도 보시면 2 진수로는 16 비트를 쓰는데도 16 자리가 필요합니다. 하지만 16 진수를 쓰면 간단히 4 자리로 표현할 수 있습니다. 16 진수가 새로운 숫자는 아니고 2 의 4 자리가 모인 것입니다. 4 비트인 것이지요. 앞으로 여러분이 소프트웨어 엔지니어가 되시면 여러가지 코딩을 하시게 될 것입니다. 그런데, 사람이 많이 보게 되는 PC 프로그램을 코딩하신다면 10 진수를 많이 사용하시게 될 것이

고 전기전자 장치를 컨트롤하는 프로그램을 코딩하신다면 16 진수를 많이 사용하게 될 것입니다.

코딩에서 2 진수는 앞에 0b 를 붙이고 16 진수는 0x 를 붙입니다. 아무것도 붙이지 않는 것은 10 진수입니다. 다음은 모두 같은 코드 입니다.

```
int bin_number = 0b10100;
int dec_number = 20;
int hex_number = 0x14;
```

각 변수에 여러 수 표현으로 20 을 할당한 것입니다.

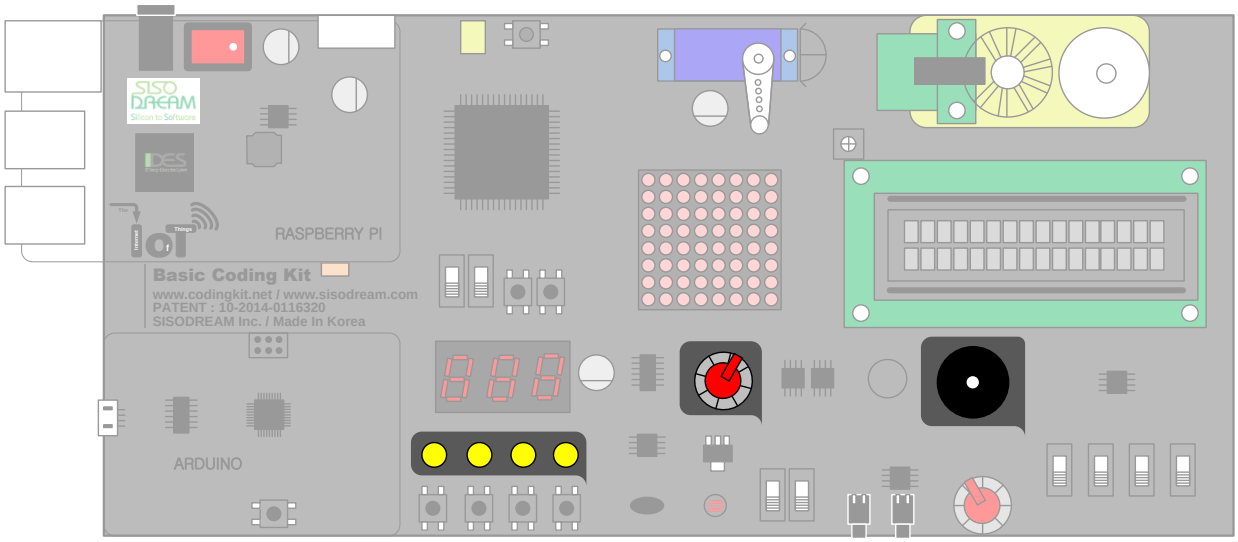
< 연습 문제 : **가변 저항을 이용하여 부저 음의 높낮이 조절 및 LED 켜기** >

가변저항을 돌리면 부저 소리의 높낮이가 바뀔 뿐 아니라 소리가 높아질수록 LED 도 많이 켜지는 예제를 한 번 해 보겠습니다.

```
#define BUZ 5
#define VAR_RES_1 A3
#define LED_0 2
#define LED_1 3
#define LED_2 7
#define LED_3 8
#define ON 1
#define OFF 0

void setup() {
  Serial.begin(9600);
  pinMode(LED_0, OUTPUT);
  pinMode(LED_1, OUTPUT);
  pinMode(LED_2, OUTPUT);
  pinMode(LED_3, OUTPUT);
}
```

위와 같이 부저, 가변저항, LED 가 정의 되었고, setup() 함수로 초기화 하였습니다. 여러분이 loop() 함수 부분을 코딩해 주십시요. LED 는 가변저항 값 0 ~ 1023 까지를 4 등분하여 값이 커질수록 LED 를 하나씩 더 켜 주면 됩니다. 이제 코드를 업로드하여 코딩키트에서 확인해 봅니다.

< 코드 위치 : Coding Kit → **vr_buz_frq_led >**



< 문법 설명 : **map() 함수** >

가변저항 값은 0 ~ 1023 까지 변하지만, 부저의 주파수는 20 ~ 4000 까지 변할 수 있습니다. 그래서 가변저항의 0 ~ 1023 의 숫자 범위를 부저의 주파수 20 ~ 4000 까지의 숫자 범위로 변환해 주는 map() 함수를 사용해 보겠습니다. map() 함수는 매핑(mapping)을 의미합니다. map() 함수는 다음 예제와 같이 사용합니다.

```
#define BUZ 5
#define VAR_RES_1 A3

void setup() {
  Serial.begin(9600);
}

void loop() {
  int val_var_res_1 = analogRead(VAR_RES_1);
  int val_mapped = map(val_var_res_1, 0, 1023, 20, 4000);
  tone(BUZ, val_mapped, 500);
  Serial.print("Variable Resistor : ");
  Serial.println(val_mapped);
  delay(500);
}
```

위의 코드를 보면 일단 가변저항 값을 val_var_res_1 에 저장합니다. 그리고 map() 함수를 활용하여 0 ~ 1024 까지의 값을 20 ~ 4000 까지의 값으로 바꾸어 val_mapped 함수에 저장합니다. map() 함수는 다음과 같이 정의 합니다.

```
new_value = map(old_value, from_min, from_max, to_min, to_max);
```

old_value 값의 from_min ~ from_max 값의 범위를 to_min ~ to_max 값으로 변환하여 new_value 에 저

장합니다. 잘 이해가 안 가실수 있는데, 그래서 예를 한 번 들어보겠습니다. 30 ~ 50 의 값을 0 ~ 100 의 값으로 바꾸어 보겠습니다. 그러면 코딩은 다음과 같이 될 것입니다.

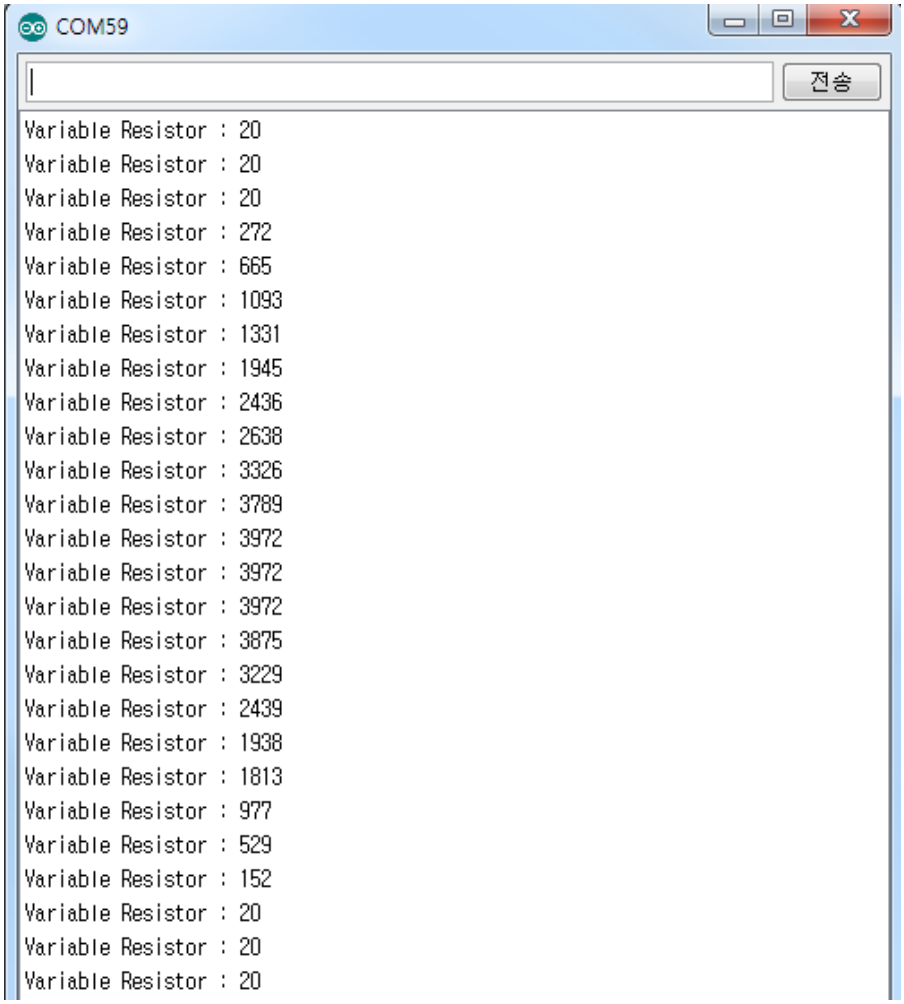
```
new_val = map(old_value, 30, 50, 0, 100)
```

이 코딩에 의해서 값들은 다음과 같이 변합니다.

old_value	new_val
30	0
31	5
32	10
33	15
34	20
35	25
36	30
37	35
38	40
39	45
40	50
41	55
42	60
43	65
44	70
45	75
46	80
47	85
48	90
49	95
50	100

각각의 입력 값들이 새로운 범위에 해당 값으로 매핑되는 것을 알 수 있습니다.

이제 코드를 업로드하여 코딩키트에서 실행해 보면 부저의 소리가 이전 보다 훨씬 더 높이 올라가는 것을 알 수 있을 것입니다. 그리고 시작도 20 Hz 에서 시작하여 아주 낮은 주파수의 소리가 들립니다. 이것을 시리얼 모니터로 보면 다음과 같습니다.



< 예제 코드 : **map() 함수를 이용하여 가변 저항 값을 부저 주파수 값으로 변환하기 >**

< 코드 위치 : Coding Kit → **vr_buz_frq_map >**

map() 함수는 앞으로도 많이 사용하게 될 것입니다. 이 함수를 사용하면 어떤 범위의 값을 다른 범위의 값으로 편리하게 변환해 줄 수 있습니다.

< 문법 설명 : **함수(2)** >

이번에 map() 함수를 배웠는데요. 여기서 함수의 일반적인 특징 몇 가지를 더 알아보고 가겠습니다. 함수는 어떤 입력을 받아서 함수 안의 문장들로 어떤 연산을 수행한 후에 결과 값을 출력해 주는 역할을 합니다. 이 결과 값은 변수에 저장되거나 이 함수를 호출하는 코드에서 연산에 직접 사용될 수 있습니다. 이 출력값을 함수를 호출한 코드에 돌려준다고 해서 반환값이라고 합니다. 또한 이렇게 값을 함수를 호출한 코드로 반환값을 전달하는 것을 반환한다고 합니다. 이 내용을 좀 더 구체적으로 설명하겠습니다.

함수 정의는 다음과 같이 합니다. - 이전에 함수를 배울 때 함수 정의는 함수의 수행 내용을 기술해 주는 것이라고 배웠습니다.

```
반환값_변수_타입 함수_이름 (매개변수_타입 매개변수) {
  문장_1;
  return 반환값;
}
```

매개변수라는 입력 값으로 문장_1 을 수행합니다. 그리고 결과 값을 반환하는데, 결과 값은 "return(반환값)" 으로 반환합니다. 좀 더 이해하기 쉽게 map() 함수를 예로 들겠습니다. map() 함수의 함수 정의는 다음과 같습니다.

```
int map(int x, int in_min, int in_max, int out_min, int out_max)
{
  int out_val = (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
  return out_val;
}
```

map() 함수의 매개변수는 "int x, int in_min, int in_max, int out_min, int out_max" 입니다. 이 값들이 "(x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min" 문장을 수행하고 그 값을 out_val 변수에 저장합니다. 이 값이 return 에 의해서 반환 되고, 이 반환 값의 타입은 int 라는 변수 타입니다.

이 함수는 다음과 같이 사용되었습니다.

```
int val_mapped = map(val_var_res_1, 0, 1023, 20, 4000);
tone(BUZ, val_mapped, 500);
```

이와 같이 함수의 매개변수에 값을 전달하는 것을 인자를 전달한다고 합니다. 그리고 이렇게 함수를 불러서 사용하는 것을 함수 호출이라고 한다고 전에 배웠습니다. map() 함수의 반환(return)값은 val_mapped 변수에 저장이 됩니다. 위와 같이 변수에 저장하여 사용할 수도 있고 다음과 같이 연산에 직접 사용할 수도 있습니다.

```
tone(BUZ, map(val_var_res_1, 0, 1023, 20, 4000), 500);
```

함수라는 것은 컴퓨터 언어에서 매우 중요한 개념입니다. 잘 이해하고 유용하게 사용하세요.

한 가지 더 알아 두셔야 할 것은 setup() 함수와 loop() 함수 앞에는 void 라고 쓰여 있습니다. 무심결에 계속 쓰고는 있었는데, 무슨 의미인지 잘 모르셨죠? 이것은 반환할 값이 없다는 뜻입니다. 즉, setup() 함수와 loop() 함수는 반환(return)할 값이 없다는 것입니다. 그래서 여러분도 함수를 정의할 때 반환할 값이 없으면 앞에 void 를 꼭 써 주십시요.

< 연습 문제 : **가변저항으로 LED 밝기 조절하기** >

이번에는 가변저항으로 LED 의 밝기를 조절해 보겠습니다. 다음 코드가 전에 LED 밝기를 조절했던 코드입니다.

```
#define LED 3

#define ON 1
#define OFF 0

void setup() {
  pinMode(LED, OUTPUT);
}

void loop() {
  for (int i = 0; i < 256; i++) {
    analogWrite(LED, i);
    delay(10);
  }
  digitalWrite(LED, OFF);
  delay(500);
}
```

위의 코드에서 analogWrite(LED, i) 는 LED 에 PWM 신호를 주는데, i 값 만큼이 1 이고 나머지는 0 이 되는 PWM 신호를 LED 에 주어 LED 의 밝기를 조절합니다. 여기서 i 값은 0 ~ 255 까지 변합니다. 그런데, 가변저항은 0 ~ 1023 까지 변합니다. 그럼 map() 함수를 사용하고 그 결과 값을 위의 코드에서 i 에 대체하면 됩니다. 그리고 for 문은 제거해 주시고요.

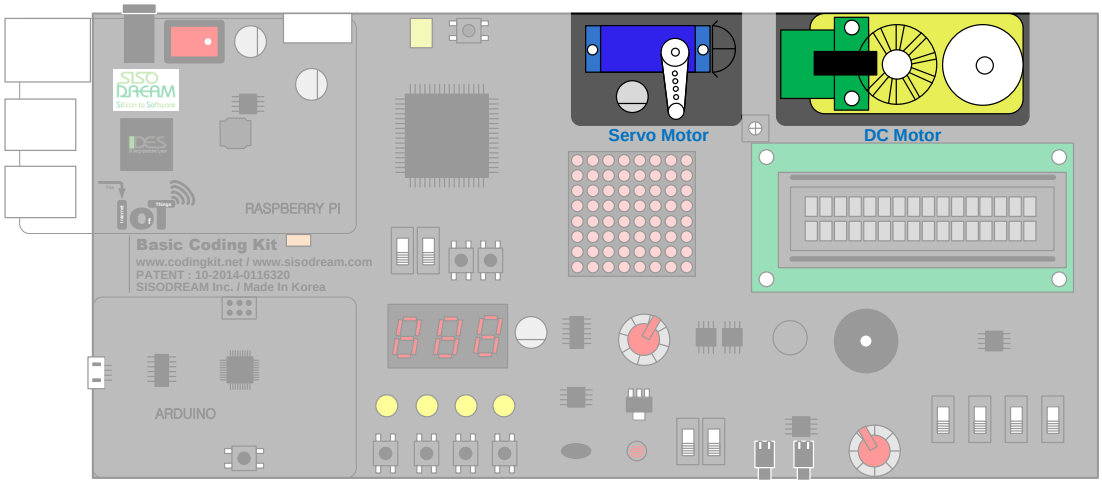
```
#define LED 3
#define VAR_RES_1 A3

void setup() {
  Serial.begin(9600);
}
```

setup() 함수는 위와 같습니다. loop() 함수는 직접 작성해 보세요.

< 코드 위치 : Coding Kit → **vr_led** >

[DC 모터 돌리기]

이제 또 새로운 디바이스를 다루어 볼까요? 이번에는 모터(Motor)를 다루어 보겠습니다. 코딩키트에는 DC 모터와 서보 모터 (Servo Motor) 가 있습니다. DC 모터는 여러분이 잘 알고 계시는 장난감 자동차 바퀴 등에 쓰이는 빙글빙글 돌아가는 모터고요. 서보 모터는 일정한 각도만을 움직이는 모터입니다. 주로 로보트를 만들 때 관절등에 사용됩니다. 예를 들어 로보트의 팔꿈치에 서보 모터를 장착하면 로보트의 팔이 어느 범위 이내의 각도 안에서만 동작하게 됩니다.



먼저 DC 모터를 다루어 보겠습니다. DC 모터를 컨트롤하는데는 다음과 같이 3 개의 신호가 사용됩니다.

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

DCMOT_EN (DC Motor Enable) 은 DC 모터를 동작하게 하는 DC 모터 활성화 신호입니다. 이 신호에 1 값을 전달하면 DC 모터가 활성화 됩니다. DCMOT_FWD (DC Motor Forward) 와 DCMOT_BWD (DC Motor Backward) 는 DC 모터의 회전 방향을 결정합니다. 둘 중 하나가 1 이 되면 그 방향으로 회전합니다. DCMOT_FWD 는 정방향으로 시계 반대 방향으로 회전합니다. DCMOT_BWD 는 역방향으로 시계 방향으로 회전합니다.

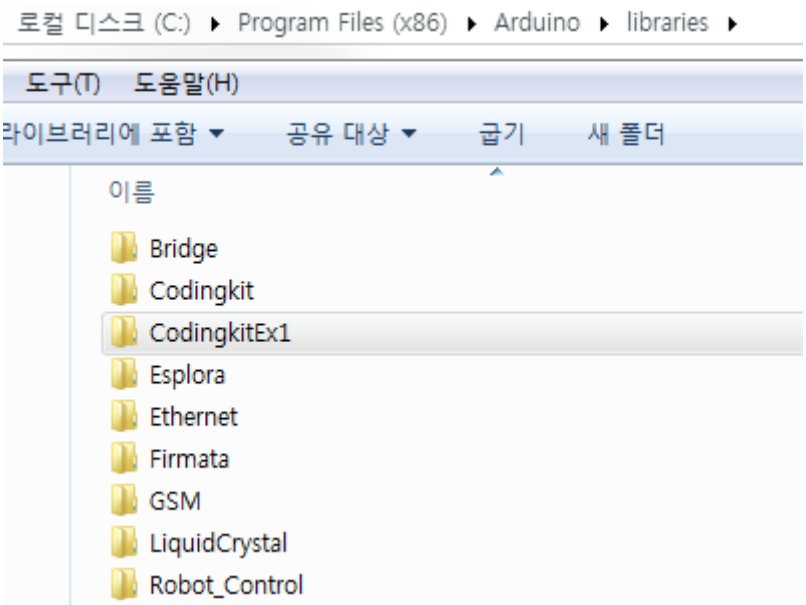
그러면 DC 모터가 정방향으로 회전하는 코딩을 해 보겠습니다.

```
#define DCMOT_EN    6
```

```
#define DCMOT_FWD 10
#define DCMOT_BWD 11

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

  digitalWrite(DCMOT_EN, 1);
  digitalWrite(DCMOT_FWD, 1);
  digitalWrite(DCMOT_BWD, 0);
}

void loop() {
}
```

코드에 대해서는 길게 설명하지 않아도 될 것 같습니다. DC 모터에 대해서 한 가지 더 알아 두셔야 할 것은 DCMOT_EN 은 1 이고 DCMOT_FWD, DCMOT_BWD 는 모두 0 이면 DC 모터는 돌지 않습니다. 또한 DCMOT_FWD, DCMOT_BWD 두 신호 모두 1 이어도 DC 모터는 돌지 않습니다.

이번 예제부터는 Coding Kit Ex1 라이브러리를 추가해 주어야 합니다. CodingkitEx1.zip 파일을 아두이노 설치폴더의 "libraries" 폴더 아래에 압축을 풀어 주십시요. 다음 그림과 같은 구조로 CodingkitEx1 폴더를 위치해 두어야 합니다.



< 예제 코드 : **DC 모터 정방향으로 돌리기** >

< 코드 위치 : Coding Kit Ex1 → **dcmot_en** >

< 연습 문제 : **버튼으로 DC 모터 방향 바꾸기** >

버튼을 누르면 DC 모터의 방향이 바뀌는 예제를 해 보겠습니다.

setup() 함수는 위와 같습니다. 여러분이 loop() 함수의 내용을 작성해 주십시요.

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

#define BUTTON_0 12
#define BUTTON_1 13

void setup() {
  pinMode(DCMOT_EN, OUTPUT);
  pinMode(DCMOT_FWD, OUTPUT);
  pinMode(DCMOT_BWD, OUTPUT);
  pinMode(BUTTON_0, INPUT);
  pinMode(BUTTON_1, INPUT);

  digitalWrite(DCMOT_EN, 1);
}
```

전에 배운 논리 연산자 중 "NOT 연산자" 를 사용하시면 매우 간단한 코드가 될 것입니다.

< 코드 위치 : Coding Kit Ex1 → **dcmot_dir_but** >

DC 모터의 신호 중 DCMOT_FWD 와 DCMOT_BWD 신호는 PWM 신호를 입력으로 받을 수 있습니다. PWM 신호에 따라서 DC 모터의 회전 속도가 달라 집니다. 그러면 우리가 전에 했던 PWM 신호로 LED 의 밝기를 조절했던 코드의 일부를 한 번 볼까요?

```
void loop() {
  for (int i = 0; i < 256; i++) {
    analogWrite(LED, i);
    delay(10);
  }
  digitalWrite(LED, OFF);
  delay(500);
}
```

여기서 analogWrite() 함수를 이용하여 PWM 신호의 1 인 구간을 조정했습니다. 이것을 듀티비를 조정했다고 합니다. 이 코드와 유사하게 DC 모터의 정방향 속도를 바꾸어 보겠습니다.

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

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

  digitalWrite(DCMOT_EN, 1);
  digitalWrite(DCMOT_BWD, 0);

  Serial.begin(9600);
}

void loop() {
  for (int i = 0; i < 256; i++) {
    analogWrite(DCMOT_FWD, i);
    Serial.print("PWM : ");
    Serial.println(i);
    delay(100);
  }
  digitalWrite(DCMOT_FWD, 0);
  delay(1000);
}
```

LED 밝기 조절하는 코드와 마찬가지로 analogWrite() 함수를 사용하였고, 약간의 시간 조정을 하였습니다. 그리고 PWM 값을 위해 시리얼 모니터 코딩을 하였습니다. 시리얼 모니터로 값을 보면 DC 모터는 PWM 신호 값이 50 정도가 되면 동작하기 시작합니다. 그 이하의 값으로는 DC 모터를 돌리기에는 어려운가 봅니다. 코드를 작성하여 업로드해 코딩키트에서 실행해 보십시요. DC 모터가 서서히 속력을 내는 것이 자동차의 가속 페달을 서서히 밟았을 때와 비슷한 소리를 내는 것 같지 않습니까? 재미있네요.

< 예제 코드 : **PWM 신호를 활용한 DC 모터 속도 바꾸기** >

< 코드 위치 : Coding Kit Ex1 → **dcmot_pwm >**

이번에는 버튼을 자동차 가속 페달로 사용하고 모터를 자동차의 바퀴로 생각하는 코딩을 해 볼까요? 자동차가 우리의 코딩키트보다는 훨씬 더 복잡하겠지만, 원리는 비슷하겠지요. 그럼 당장 해 봅시다.

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

#define BUTTON 12

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

```
  pinMode(BUTTON, INPUT);

  digitalWrite(DCMOT_EN, 1);
  digitalWrite(DCMOT_BWD, 0);

  Serial.begin(9600);
}

int dcmot_speed = 50;

void loop() {
  analogWrite(DCMOT_FWD, dcmot_speed);

  if (!digitalRead(BUTTON)) {
    if (dcmot_speed >= 255)
        dcmot_speed = 255;
      else
        dcmot_speed++;
  }
  else {
    if (dcmot_speed <= 50)
        dcmot_speed = 50;
      else
        dcmot_speed--;
  }

  Serial.print("DCMOT Speed : ");
  Serial.println(dcmot_speed);
  delay(50);
}
```

속도 변수 dcmot_speed 의 초기값은 50 으로 했습니다. 그러면 DC 모터가 슬슬 움직입니다. 이것은 자동차의 가속 페달을 밟지 않아도 꾸물꾸물 가는 것과 같습니다. 버튼을 누르면 가속 페달을 밟은 것과 같습니다. 그래서 계속해서 속도가 올라 가지요. 하지만 최고속도 255 를 넘으면 안 됩니다. 이것은 이 값을 analogWrite() 함수에 주기 때문에 255 를 최대로 잡은 것입니다. 누르고 있던 버튼을 떼면 속도는 계속 감속합니다. 제일 낮은 속도는 50 입니다. 여기서 사용된 "++" 는 왼쪽의 변수 값에 1 을 더해서 그 변수에 다시 저장하는 다음 코드와 같은 것입니다.

```
dcmot_speed = dcmot_speed + 1;
```

비슷하게 "--" 는 왼쪽의 변수 값에 1 을 뺀 다음 그 변수에 다시 저장하는 다음 코드와 같은 것입니다.

```
dcmot_speed = dcmot_speed - 1;
```

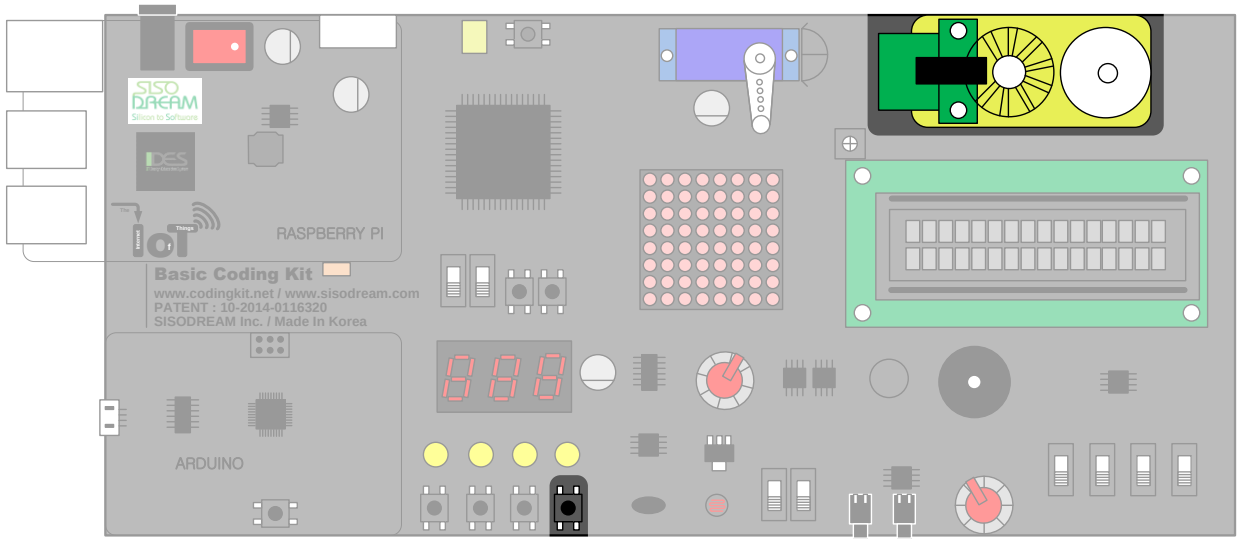
이제 한 번 달려 볼까요? 코드를 업로드하여 코딩키트에서 실행 시켜 봅니다. 붕붕 하고 버튼을 눌러 보십시요.

여기서 delay(50) 의 값을 작게 하면 좀 더 빠르게 가속할 수 있습니다. 속도가 얼마나 되는지 보려면 시리

얼 모니터를 이용합니다. 나중에는 좀 더 공부를 하고 나면 속도를 세븐세그먼트에 표시하면 디지털 속도계가 됩니다. 서보 모터에 표시하면 아날로그 속도계가 되겠지요. 나중에 그런 예제들을 다 해 볼 것입니다. 기대하고 계세요.

< 예제 코드 : **버튼과 DC 모터를 이용한 자동차 가속 페달** >

< 코드 위치 : Coding Kit Ex1 → **dcmot_speed_but >**



이번에는 가변 저항을 이용하여 속도를 한 번 바꾸어 볼까요? 다음과 같이 코딩해 보았습니다.

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

#define VAR_RES_1 A3

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

  digitalWrite(DCMOT_EN, 1);
  digitalWrite(DCMOT_FWD, 0);
  digitalWrite(DCMOT_BWD, 0);

  Serial.begin(9600);
}

int dcmot_speed = 0;
```

```
void loop() {
  int val_var_res_1 = analogRead(VAR_RES_1);
  dcmot_speed = map(val_var_res_1, 0, 1023, 0, 255);

  Serial.print("Variable Resistor : ");
  Serial.print(val_var_res_1);

  analogWrite(DCMOT_FWD, dcmot_speed);
  Serial.print(" Speed : ");
  Serial.println(dcmot_speed);

  delay(100);
}
```

먼저 가변저항을 A3 핀으로 정의 (#define VAR_RES_1 A3) 하였구요. 가변저항 값을 읽어 val_var_res_1 에 저장하였습니다. 이 값은 0 ~ 1023 까지 변하는데, map() 함수를 이용하여 analogWrite() 함수에 입력할 수 있는 0 ~ 255 값으로 바꾸어 변수 dcmot_speed 에 저장하였습니다. 이 값을 analogWrite() 함수를 이용하여 DCMOT_FWD 에 전달하면 PWM 신호가 DC 모터에 전달됩니다. 이제 업로드하고 코딩키트의 가변저항을 돌려 보면 DC 모터의 속도 바뀌면서 회전합니다. 자동차 소리처럼 붕붕 소리를 낼 수도 있답니다.

< 예제 코드 : **가변 저항을 이용한 속도 바꾸기** >

< 코드 위치 : Coding Kit Ex1 → **dcmot_speed_vr >**

< 연습 문제 : **가변 저항을 이용한 속도와 방향 바꾸기** >

이번에는 가변저항을 이용하여 속도만 바꾸는 것이 아니라 방향까지 바꿉니다. 가변저항의 값을 반으로 나누어 가변저항 값이 중간이면 DC 모터는 움직이지 않습니다. 하지만 중간에서 오른쪽 방향으로 더 돌리면 DC 모터는 정방향으로 돕니다. 가변저항을 오른쪽으로 더 돌리면 속도는 올라갑니다. 다시 가변저항으로 왼쪽으로 돌리면 속도는 내려갑니다. 중간에 오면 DC 모터는 다시 멈춥니다. 중간에서 왼쪽으로 돌리면 역방향으로 회전합니다. 왼쪽으로 더 돌리면 속도는 올라갑니다. 왼쪽으로 끝까지 돌리면 최대의 속도가 됩니다. 이렇게 가변저항을 이용하여 DC 모터의 속도와 방향을 바꾸는 예제를 해 보겠습니다.

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

#define VAR_RES_1 A3

void setup() {
  pinMode(DCMOT_EN, OUTPUT);
```

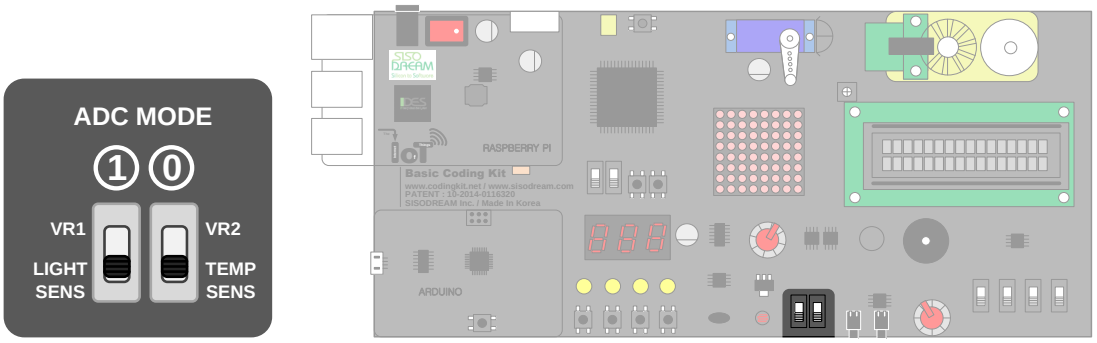
```
    pinMode(DCMOT_FWD, OUTPUT);
    pinMode(DCMOT_BWD, OUTPUT);

    digitalWrite(DCMOT_EN, 1);
    digitalWrite(DCMOT_FWD, 0);
    digitalWrite(DCMOT_BWD, 0);

    Serial.begin(9600);
}
```

핀 정의와 setup() 함수는 위와 같습니다. loop() 함수의 내용은 직접 작성해 보십시요. 설명이 길었듯이 조금 어렵습니다. 하지만 어려운만큼 재밌습니다. 지금까지 잘 따라오신 분이라면 이제 코딩키트를 활용한 코딩의 세계에 푹 빠졌을 것입니다.

< 코드 위치 : Coding Kit Ex1 → **dcmot_speed_vr_dir >**

[밝기 센서]

지금까지는 아날로그 입력으로 가변저항에서 오는 값만을 받았는데요. 이제부터는 센서에서 오는 아날로그 값을 받아 보겠습니다. 코딩키트에서는 가변저항과 밝기, 온도 센서가 같은 아두이노의 아날로그핀을 공유합니다. 그래서 다음과 같은 모드 스위치를 통해서 가변저항을 쓸 것인 센서를 쓸 것인지를 결정하여야 합니다.



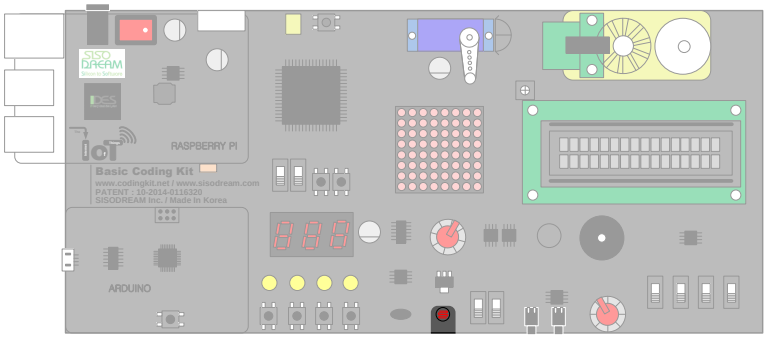
이 모드 스위치는 2 개이고 각각은 다음 표와 같이 쓰입니다. 스위치 1 은 가변저항 1 과 밝기 센서 중 하나를 선택하고, 스위치 0 은 가변저항 2 와 온도 센서를 선택합니다.

ADC Mode Switch 1	아두이노 아날로그 핀 (A3)
Up	가변저항 1 (VR1)
Down	밝기 센서

ADC Mode Switch 0	아두이노 아날로그 핀 (A2)
Up	가변저항 2 (VR2)
Down	온도 센서

센서라는 것은 잘 알겠지만 밝기, 온도, 거리 등, 측정하려는 대상의 상태를 전기 신호로 바꾸어 주는 장치입니다. 코딩키트에는 밝기, 온도, 소리, 적외선 센서 4 개가 있습니다. 이 센서에서 전달되는 값은 위에서 배운, 아두이노 안의 ADC 를 거쳐서 **0 ~ 1023** 까지의 값으로 들어 옵니다. 이 값을 이용하여 여러분이 코딩을 하는 것입니다. 이러한 코딩을 통하여 각 센서는 어떤 특징이 있는지, 어떤 분야에 어떻게 사용하면 좋은지도 알아 보겠습니다.

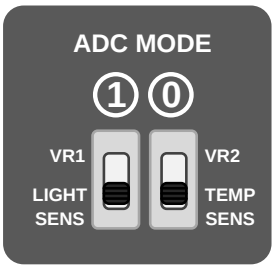
처음으로는 밝기 센서를 다뤄 보겠습니다. 밝기 센서는 다음 사진과 같습니다.



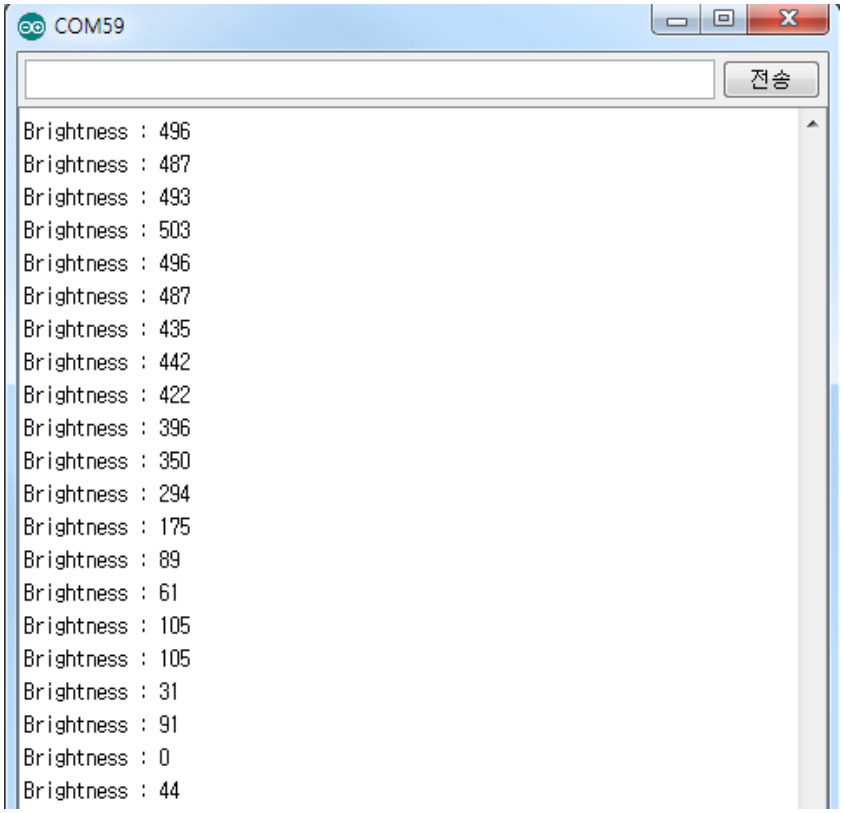
밝기 센서는 아날로그 핀 A3 에 연결되어 있습니다. 이 값을 시리얼 모니터로 보겠습니다. 코드는 다음과 같습니다.

```
#define SENSOR_LIGHT A3

void setup() {
  Serial.begin(9600);
}

void loop() {
  int brightness = analogRead(SENSOR_LIGHT);
  Serial.print("Brightness : ");
  Serial.println(brightness);
  delay(500);
}
```

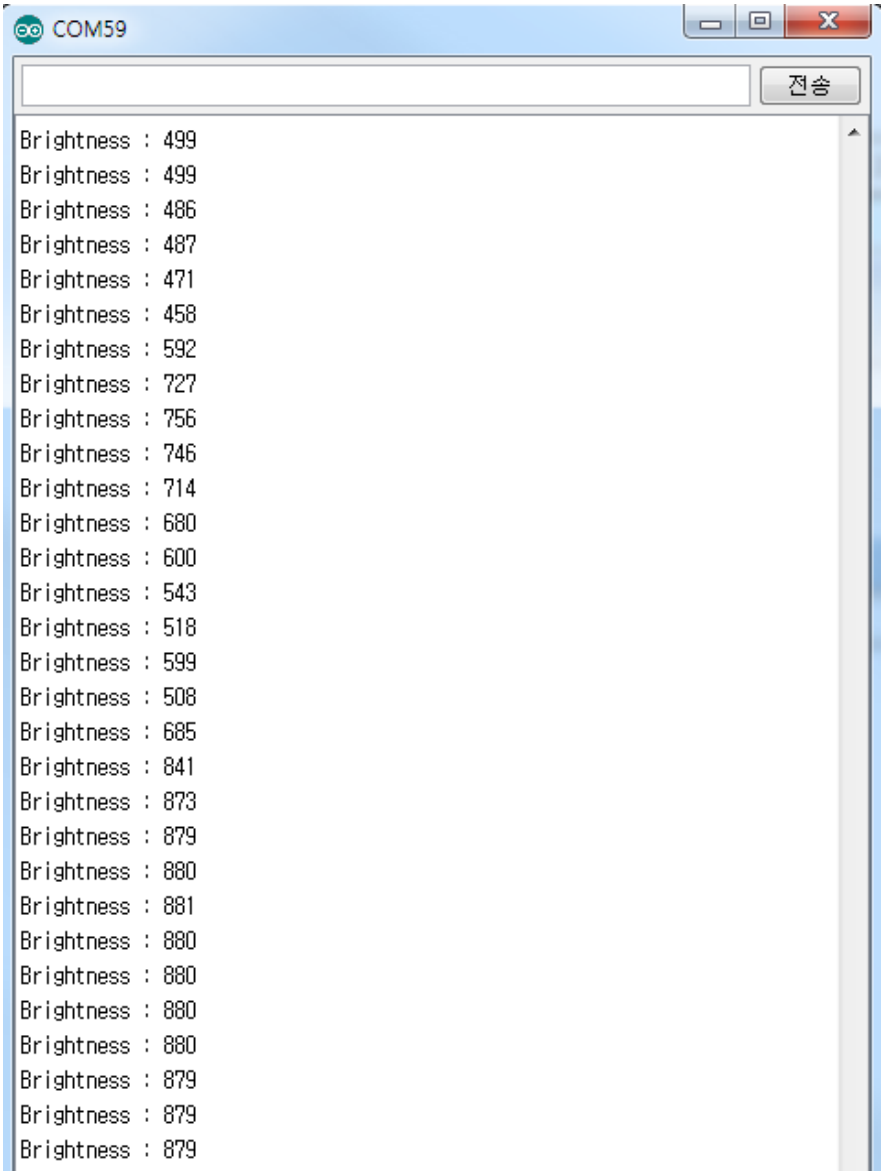
가변저항 값을 읽는 코드와 별로 다르지 않습니다. 그럼 업로드하기 전에 코딩키트에서 ADC MODE 스위치 중 1 번 스위치를 다음과 같이 아래로 내려야 합니다.



이제 코드를 업로드 합니다. 그리고 시리얼 모니터를 켭니다. 그러면 값이 대략적으로 470 ~ 500 정도로 나올 것입니다. 이것은 현재 실험하고 있는 장소의 밝기에 따라서 많은 편차가 있으니 470 ~ 500 사이가 안 나온다고 해서 코딩키트가 잘 못 됐는니, 망가졌느니 하지는 마세요. 이제 밝기 센서를 손으로 가려 보십시요. 그러면 값이 내려갈 것입니다. 손전등이나 휴대폰의 플래시를 이용하여 더 밝게 해 보십시요. 그러면 값이 올라갈 것입니다. 다음은 손으로 가려 어둡게 했을 때의 값의 변화 입니다.



다음은 손전등을 밝기 센서에 비춰 값을 크게 한 것입니다.



< 예제 코드 : **밝기 센서 값 입력 받기** >

< 코드 위치 : Coding Kit Ex1 → **sensor_light >**

< 연습 문제 : **밝기 센서를 이용하여 어두워지면 켜지는 가로등 만들기** >

거리의 가로등은 해가 져서 어두워지면 자동으로 켜집니다. 이것이 바로 밝기 센서를 이용한 것입니다. 여러분은 밝기 센서와 LED 를 이용하여 거리의 가로등과 똑같이 해 보겠습니다. 밝기 센서가 어두워지는 것을 감지하면 LED 를 켜는 것입니다. 밝기 센서에서 입력되는 밝기의 값에 따라 LED 가 켜지게 하는 것입니다. 예를 들어 밝기의 값이 100 이하면 LED 4개를 모두 켜고, 보통 밝기인 400 이상이 되면 LED 를 모두 끕니다. 이런 예제를 한 번 해 보겠습니다.

핀 정의와 setup() 함수는 다음과 같습니다.

```
#define SENSOR_LIGHT A3

#define LED_0 2
#define LED_1 3
#define LED_2 7
#define LED_3 8

#define ON  1
#define OFF 0

void setup() {
  Serial.begin(9600);

  pinMode(LED_0, OUTPUT);
  pinMode(LED_1, OUTPUT);
  pinMode(LED_2, OUTPUT);
  pinMode(LED_3, OUTPUT);
}
```

loop() 함수는 여러분이 직접 코딩해 보십시요. 밝기 센서 값을 이용하고 이 값이 범위에 따라 몇 개의 LED 를 켤 것인가는 if 문을 사용하면 쉽게 코딩할 수 있을 것입니다. LED 가 몇 개 켜질지 하는 경계를 밝기 센서 값으로 해야 합니다. 그런데, 이것은 현재 밝기 센서가 있는 곳이 밝은 곳이냐 어두운 곳이냐에 따라서 달라질 것입니다. 이 값은 여러분이 코딩을 하고 실행을 시켜 보면서 조금씩 바꾸어 가시면 됩니다. 이런걸 "튜닝" 이라고 합니다.

< 코드 위치 : Coding Kit Ex1 → **sensor_light_led >**

밝기 센서의 값이 작아지면, 즉, 어두워지면 LED 를 더 밝게 켜는 PWM 신호를 LED 에 주는 예제를 해 보겠습니다. 어두워지면 어두워질수록 LED 는 밝게 켜지고 밝으면 밝을수록 LED 는 흐리게 켜집니다.

```
#define SENSOR_LIGHT A3
#define LED 3

#define BRIGHT_MAX 560

void setup() {
  Serial.begin(9600);

  pinMode(LED, OUTPUT);
}

void loop() {
  int brightness = analogRead(SENSOR_LIGHT);
  Serial.print("Brightness : ");
```

```
  Serial.println(brightness);

  if (brightness >= BRIGHT_MAX)
    analogWrite(LED, 0);
  else
    analogWrite(LED, map(brightness, BRIGHT_MAX, 0, 0, 255));

  delay(500);
}
```

먼저 밝기 센서의 값을 시리얼 모니터로 출력하는 예제를 활용하여 현재의 밝기 값을 얻습니다. 제가 하는 곳에서는 밝기 값이 560 입니다. 이 값을 최대 밝기라는 뜻으로 BRIGHT_MAX 로 정의하였습니다. 그리고 loop() 함수에서 이 값보다 더 큰 값이면 LED 를 끕니다. 그리고 이것보다 어두운 밝기는 map() 함수를 활용하여 0 ~ 255 범위로 만들고 이 값을 analogWrite() 함수를 활용하여 LED 에 PWM 신호를 전달합니다. 이 코드 예제에서 배울 점은 BRIGHT_MAX 라는 값을 정하고 이 값을 한계 값으로 하는 if 문을 코딩하는 것입니다. analogWrite() 함수를 활용하여 LED 에 PWM 신호를 전달하기 전에 한계 값 (BRIGHT_MAX) 이상은 if 문을 활용하여 처리해 주는 것입니다. 이렇게 먼저 예외 값은 정리하고 진행을 해야 코드를 쉽게 작성할 수 있습니다. 일반적인 코딩 작성자는 LED 를 켜는 것에 초점촛 맞추기 때문에 LED 를 어떻게 켤까를 먼저 고민합니다. 하지만 코딩에서는 예외 사항을 먼저 정리하고 코딩하는 것이 좋습니다. 앞으로의 코딩 과정에서도 이러한 경우가 많이 있을 것입니다. 그 때마다 "예외 사항을 먼저 정리하라" 는 것을 기억해 주십시요. 이러한 코딩 기법은 중요한 부분이니 잘 알아두셨다가 필요한 곳에 잘 활용하면 좋을 것입니다.

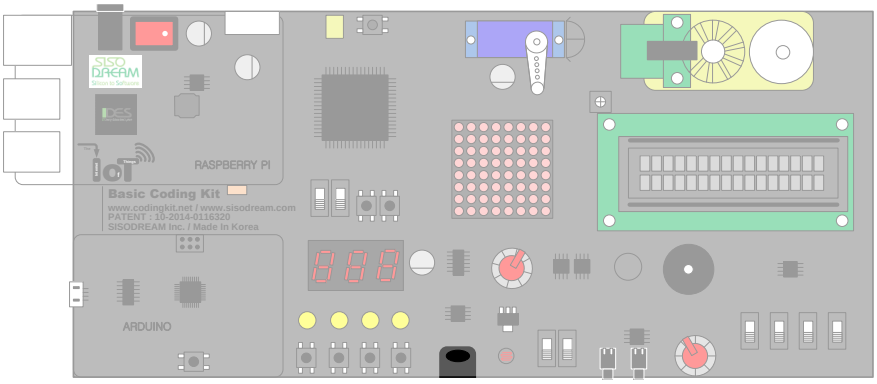
위 코드에서 한 가지 더 중요하게 봐야 할 것은 map() 함수입니다. 일반적인 map() 함수는 작은 값은 작은 값에 매핑되고 큰 값은 큰 값에 매핑되었습니다. 그런데, 위의 코드 "map(brightness, BRIGHT_MAX, 0, 0, 255))" 에서는 반대로 매핑하였습니다. 이렇게 할 수도 있다는 것과 경우에 따라서는 이렇게 사용하면 편리하다는 것을 잘 알아 두십시요.

< 예제 코드 : **밝기 센서를 이용한 어두워질수록 밝아 지는 LED** >

< 코드 위치 : Coding Kit Ex1 → **sensor_light_led_pwm >**

[온도 센서]

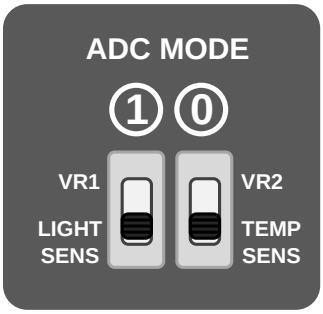
이번에는 온도 센서를 다뤄보도록 하겠습니다. 코딩키트의 온도 센서는 다음과 같습니다.



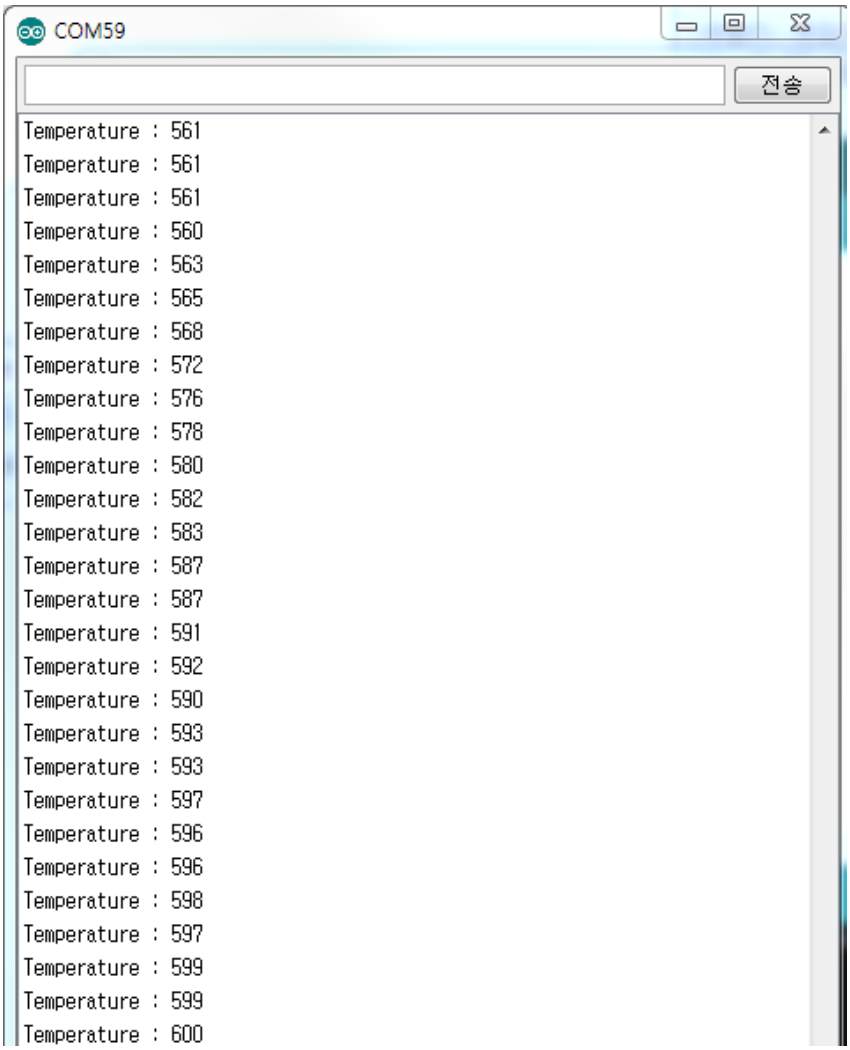
온도 센서는 온도를 측정하여 전기 신호로 바꾸어 주는 역할을 합니다. 코딩은 밝기 센서와 비슷합니다.

```
#define SENSOR_TEMP A2

void setup() {
  Serial.begin(9600);
}

void loop() {
  int temp = analogRead(SENSOR_TEMP);
  Serial.print("Temperature : ");
  Serial.println(temp);
  delay(500);
}
```

위의 코드를 보면 밝기 센서와 매우 비슷하다는 것을 아실 수 있을 것입니다. LIGHT 를 TEMP 로만 바꾸면 됩니다. 코드를 업로드 하기 전에 ADC MODE 스위치 0 번을 아래로 내려주셔야 합니다. 밝기 센서는 ADC MODE 스위치 1 번이고 온도 센서는 스위치 0 번입니다. 스위치를 내렸다면 이제 코드를 업로드합니다. 그리고 시리얼 모니터를 열어 온도 값이 얼마인지를 읽어 봅니다.



제가 한 곳에서는 온도 값이 550 정도가 측정이 됩니다. 온도 센서를 손가락 엄지와 검지를 이용하여 꼭 쥐고 있으면 온도 값이 올라가는 것을 확인할 수 있을 것입니다.



이것은 온도 센서의 값이고 실제로 우리가 알고 있는 20도, 30도 하는 온도 값은 아닙니다. 이 값과 온도계로 현재의 온도를 측정한 이후에 두 값의 관계를 이용하여 온도를 구하여야 합니다.

< 예제 코드 : **온도 센서 값 입력 받기** >

< 코드 위치 : Coding Kit Ex1 → **sensor_temp >**

온도가 이렇게 올라가면 어떻게 할까요? 선풍기를 켜야겠지요. 그래서 이번에는 온도 센서와 선풍기 역할을 할 DC 모터를 연결해 보겠습니다. 그 전에 위의 온도 센서 예제 코드를 활용하여 현재 온도 값이 얼마이고 손가락으로 쥐었을 때 얼마까지 온도가 올라가는지를 측정해 두십시요. 제가 있는 환경에서는 560 ~ 600 까지 변합니다. 그러니깐 저는 이 기준으로 코딩을 하겠습니다. 즉, 560 일 때는 DC 모터가 회전하

지 않고, 600 이 되면 최대로 회전을 하는 것이지요.

```
#define SENSOR_TEMP A2

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

#define TEMP_MIN 560
#define TEMP_MAX 600

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

  digitalWrite(DCMOT_EN, 1);
  digitalWrite(DCMOT_FWD, 0);
  digitalWrite(DCMOT_BWD, 0);

  Serial.begin(9600);
}

void loop() {
  int temp = analogRead(SENSOR_TEMP);

  if (temp <= TEMP_MIN)
    digitalWrite(DCMOT_FWD, 0);
  else if (temp >= TEMP_MAX)
    digitalWrite(DCMOT_FWD, 1);
  else
    analogWrite(DCMOT_FWD, map(temp, TEMP_MIN, TEMP_MAX, 0, 255));

  Serial.print("Temperature : ");
  Serial.println(temp);
  delay(500);
}
```

이전 코드에서 측정한 온도의 최소값과 최대값을 TEMP_MIN, TEMP_MAX 로 정의 합니다. 그래서 loop() 함수에서 if 문을 보면 온도 값이 TEMP_MIN 보다 작으면 DC 모터가 아예 돌지 않게 합니다. 그리고 TEMP_MAX 보다 크면 최대로 빨리 돌 수 있도록 합니다. 그리고 이 사이에 값은 map() 함수를 이용하여 0 ~ 255 값으로 만들어 줍니다. 여러분의 코딩키트가 있는 곳의 온도에 따라서 TEMP_MIN, TEMP_MAX 값은 바꾸어 주면 됩니다.

이 예제에서도 미리 TEMP_MIN, TEMP_MAX 라는 한계값을 정하고 loop() 함수의 if 문에서 한계값 처리를 미리 해 주었습니다. 잘 봐 두세요.

이 코드에서 DCMOT_FWD 에 PWM 신호를 주어 DC 모터의 회전 속도를 조정하였습니다. 그런데, 중간에

analogWrite() 대신 digitalWrite() 를 사용하여 DCMOT_FWD 에 1 을 주어 DC 모터의 회전 속도를 최대로 하였습니다. 그래서 PWM 신호의 모든 구간을 0 이 없는 1 값으로 채우는 코드는 analogWrite() 와 digitalWrite() 두 가지로 다 표현할 수 있습니다. 다음과 같은 코드입니다.

```
digitalWrite(DCMOT_FWD, 1);
analogWrite(DCMOT_FWD, 255);
```

그럼 PWM 신호의 모든 구간을 1 이 없는 0 값으로 모두 채우는 코드는 어떻게 작성할까요? 다음과 같습니다.

```
digitalWrite(DCMOT_FWD, 0);
analogWrite(DCMOT_FWD, 0);
```

이렇게 위와 같이 PWM 신호 중 0 과 1 이 반복되는 신호가 아닌 0 혹은 1 로 모두 채워지는 신호를 DC(Direct Current) 신호라고 합니다. 이 용어도 잘 알아 두시면 좋습니다. 이 DC 는 AC(Alternating Current), DC 할 때 DC 입니다. 즉, 교류, 직류할 때 직류를 나타내는 것이지요. 실제로 교류 신호는 여기서 배우는 PWM 신호와 같지는 않지만 반복적으로 변합니다. DC 는 어느 일정 전압을 계속해서 유지하고 있는 것입니다.

< 예제 코드 : **온도 센서를 이용하여 자동으로 선풍기 돌리기** >

< 코드 위치 : Coding Kit Ex1 → **sensor_temp_dcmot >**

< 연습 문제 : **온도 센서를 이용하여 선풍기와 난로 컨트롤 하기** >

온도가 너무 높으면 선풍기를 켜야 하고 온도가 너무 낮으면 난로를 켜야 합니다. 이번 예제에서는 온도 값에 따라 선풍기 역할을 할 DC 모터를 돌리거나, 난로 역할을 할 LED 를 켜 보도록 하겠습니다. 그렇게 하기 전에 위의 예제처럼 미리 TEMP_MIN, TEMP_MAX 를 정하고, 선풍기도 켜지 않고 난로도 켜지 않는 중간 값 온도를 TEMP_MID 라고 정의해 둡니다.

정의 부분은 다음과 같습니다.

```
#define SENSOR_TEMP A2

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

#define LED_0 2
#define LED_1 3
#define LED_2 7
#define LED_3 8
```

```
#define ON  1
#define OFF 0

#define TEMP_MIN 560
#define TEMP_MAX 600
#define TEMP_MID (TEMP_MIN + ((TEMP_MAX - TEMP_MIN)/2))

#define TEMP_LED_INT ((TEMP_MID - TEMP_MIN) / 4)
```

위의 코드에서 TEMP_MIN 이 560 이고, TEMP_MAX 가 600 이면 TEMP_MID 값은 560 과 600 의 중간 값인 580 입니다. 이 값을 직접 쓰셔도 괜찮지만 위와 같이 수식으로 써 주는 것이 더 좋습니다. 그 이유는 여러분의 온도 센서의 주변 온도에 따라서 TEMP_MIN 과 TEMP_MAX 를 바꾸면 TEMP_MID 값은 자동으로 바뀌기 때문입니다. 마치 엑셀 프로그램을 사용하실 때 수식을 이용하는 것과 같은 이치입니다.

그 아래에 다음과 같은 또 하나의 수식이 더 있는데요.

```
#define TEMP_LED_INT ((TEMP_MID - TEMP_MIN) / 4)
```

이것은 TEMP_MID 값과 TEMP_MIN 사이를 4 등분하여 TEMP_LED_INT (LED Interval) 로 정의한 것입니다. 이것은 왜 이렇게 했을까요? 그것은 LED 4 개를 온도에 따라서 켜기 위한 것입니다. 온도가 많이 내려가면 LED 를 더 많이 켜 주는 것입니다. 그런데, LED 가 4 개여서 4 등분한 것이고 그 값에 따라서 다음과 같은 표로 LED 를 켜 줍니다.

온도	켜지는 LED 개수
TEMP_MIN 미만	4
TEMP_MIN 이상 (TEMP_MIN + TEMP_LED_INT) 미만	3
(TEMP_MIN + TEMP_LED_INT) 이상 (TEMP_MIN + (TEMP_LED_INT x 2)) 미만	2
(TEMP_MIN + TEMP_LED_INT x 2) 이상 (TEMP_MIN + (TEMP_LED_INT x 3)) 미만	1
(TEMP_MIN + TEMP_LED_INT x 3) 이상 TEMP_MID 미만	0

setup() 함수는 다음과 같습니다.

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

  digitalWrite(DCMOT_EN, 1);
  digitalWrite(DCMOT_FWD, 0);
  digitalWrite(DCMOT_BWD, 0);

  pinMode(LED_0, OUTPUT);
  pinMode(LED_1, OUTPUT);
  pinMode(LED_2, OUTPUT);
  pinMode(LED_3, OUTPUT);
```

```
    Serial.begin(9600);
    Serial.println(TEMP_MID);
    Serial.println(TEMP_LED_INT);
  }
```

조금 어렵기는 하겠지만 loop() 함수는 직접 작성해 보십시요. 조금 어려워서 시간이 좀 걸리 수도 있지만 꼭 해 보십시요. 그리고 다 하셨으면 제가 작성한 정답 코드와도 꼭 비교해 보십시요. 그리고 어떤 차이가 있는지 확인해 보십시요. 이러한 과정을 계속 거치다 보면 여러분의 코딩 실력은 월등히 상승할 것입니다. 앞으로는 여러분의 코딩 실력을 믿고 조금 어려운 연습 문제들도 많이 내도록 하겠습니다.

< 코드 위치 : Coding Kit Ex1 → **sensor_temp_dcmot_led** >

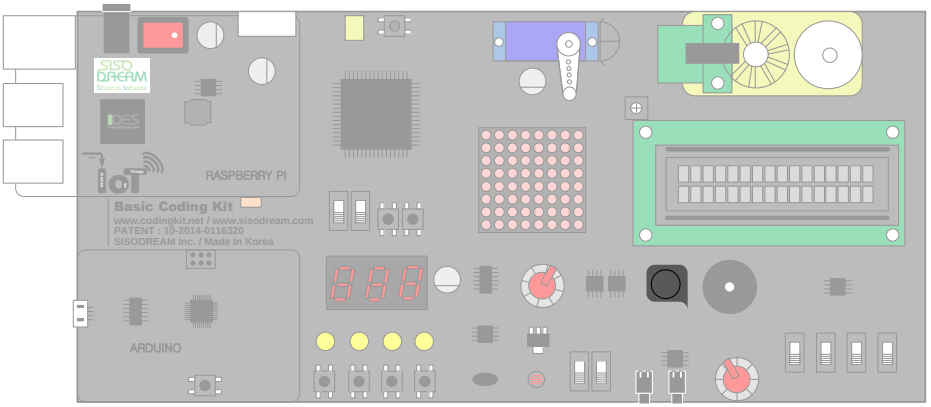
지금까지는 온도를 실제의 온도 값으로 하지 않고 센서에서 전달하는 값으로 하였습니다. 이것을 실제 온도로 바꾸려면 온도 센서의 값과 실제 온도를 측정하여 대략적으로 매칭 되는 값을 찾아가는 과정이 있어야 합니다. 그렇게 하여 찾은 값은 대략적으로 다음 표와 같습니다.

온도 센서 값	온도
237	0
286	5
339	10
395	15
454	20
512	25
569	30
630	35

이 표의 값을 잘 활용하여 온도를 표시하는 예제를 많이 만들어 보도록 하세요. 값을 보면 센서 입력 값 0 ~ 1023 범위의 중간 값들이 주로 있습니다. 이것은 이 온도 센서가 -40 도에서 120 도까지 측정할 수 있어서 표에서 보이는 0 도에서 35 도 사이는 위와 같은 값인 것입니다. 이 온도 센서의 오차는 ±5% 입니다.

[소리 센서]

소리 센서에 대해서 알아보도록 하겠습니다. 소리 센서는 마이크와 같은 것입니다. 외부의 소리가 크면 클수록 소리 센서의 값은 올라 갑니다.

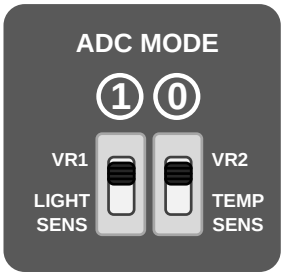


소리 센서 관련 코딩을 해 보겠습니다. 코딩키트에는 부저가 있어서 소리 센서에서 부저의 소리가 얼마나 되는지를 측정해 보겠습니다. 부저의 소리는 가변저항으로 조절해 보겠습니다. 코드는 다음과 같습니다.

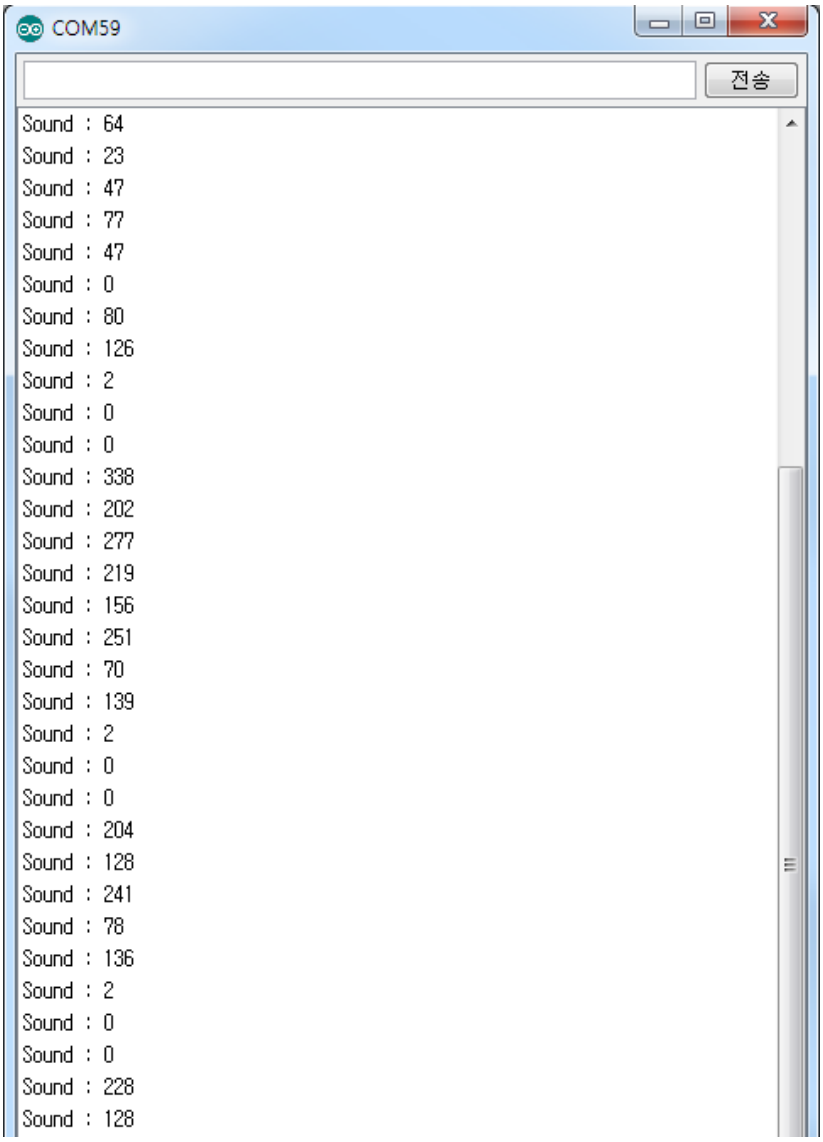
```
#define SENSOR_SOUND A1
#define BUZ 5
#define VAR_RES_1 A3

void setup() {
  Serial.begin(9600);
}

void loop() {
  int val_var_res_1 = analogRead(VAR_RES_1);
  int buz_vol = map(val_var_res_1, 0, 1023, 0, 10);
  analogWrite(BUZ, buz_vol);

  int sound = analogRead(SENSOR_SOUND);
  Serial.print("Sound : ");
  Serial.println(sound);
  delay(500);
}
```

map() 함수를 이용하여 가변저항의 값 0 ~ 1023 을 0 ~ 10 으로 변경하였습니다. 이 값을 analogWrite() 함수의 인자로 사용하여 부저의 소리 크기를 조절합니다. 이 값이 소리 센서에 입력됩니다. 이 값을 시리얼 모니터로 출력합니다.

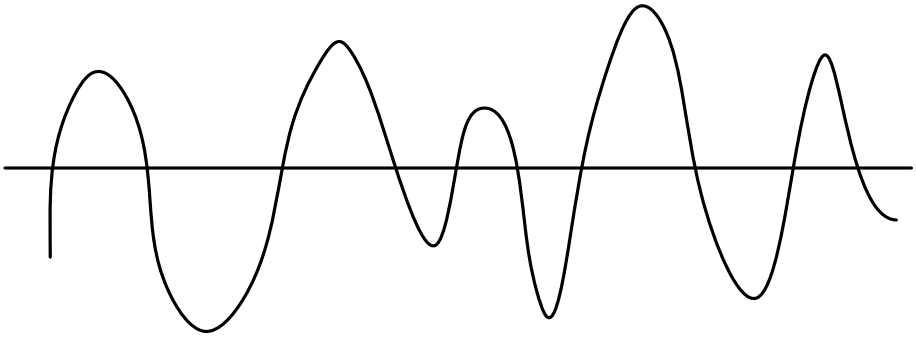
이제 코드를 업로드해 보겠습니다. 코딩키트의 가변저항을 사용할 것이므로 ADC 모드 스위치는 위로 올려 주십시요.



부저의 소리가 들리고 시리얼 모니터로 소리 센서 값을 확인해 보겠습니다.



위 그림과 같이 값이 들쭉 날쭉 합니다. 왜 그럴까요? 소리는 다음 그림과 같이 매 순간 순간 소리의 크기가 변합니다. 그래서 소리 센서의 값은 들쭉 날쭉한 것입니다.



< 예제 코드 : **소리 센서 값 입력 받기** >

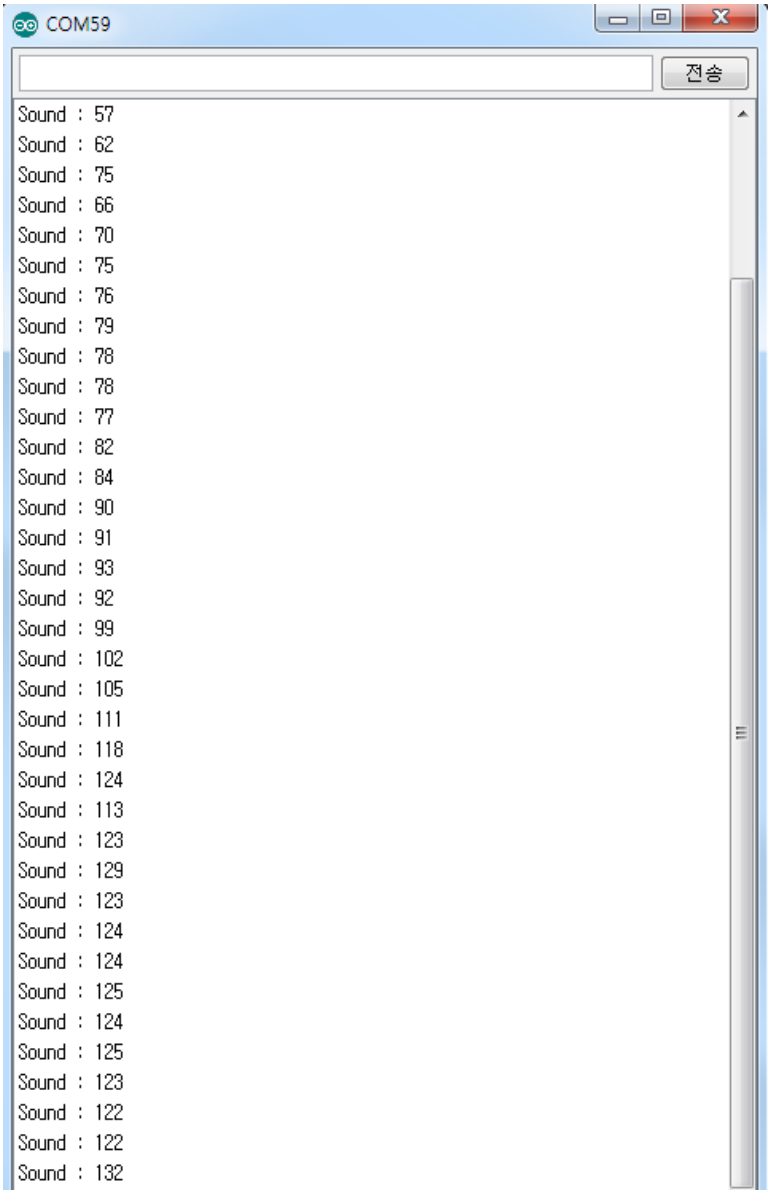
< 코드 위치 : Coding Kit Ex1 → **sensor_sound** >

소리의 크기를 제대로 측정하려면 평균값을 구해야 합니다. 한 동안 소리의 값들을 모으고 그 값들의 평균을 구하여 시리얼 모니터로 출력해 보겠습니다. 그래서 코드를 다음과 같이 수정했습니다.

```
int count = 0;
unsigned long sound_sum = 0;

void loop() {
  int val_var_res_1 = analogRead(VAR_RES_1);
  int buz_vol = map(val_var_res_1, 0, 1023, 0, 10);
  analogWrite(BUZ, buz_vol);

  int sound = analogRead(SENSOR_SOUND);

  if (count == 1000) {
    sound_sum = sound_sum / 1000;
    Serial.print("Sound : ");
    Serial.println(sound_sum);
    sound_sum = 0;
    count = 0;
  }
  else {
    sound_sum = sound_sum + sound;
    count++;
  }
}
```

loop() 함수에서 1000 번을 카운트하고 여기서 소리 값들을 전부 더해서 변수 sound_sum 에 저장합니다. if 문을 이용하여 변수 count 값이 1000 이 되면 sound_sum 을 1000 으로 나누어 소리 값의 평균을 구합니다. 이 값을 시리얼 모니터로 출력합니다. count 값이 1000 이 아닐 때는 count 를 증가시키면서 열심히 소리 값을 모읍니다. 시리얼 모니터 값은 다음과 같습니다. 훨씬 더 균일하게 소리 값이 측정되는 것을 알 수 있습니다.

위와 같은 코드도 잘 봐 두십시요. 카운트를 하고 그 카운트 값이 어떤 값에 도달했을 때 어떤 조치를 취하는 경우의 코드입니다. 하드웨어를 컨트롤할 때 많이 사용되는 경우입니다. 이와 같이 여러분이 코딩키트와 함께 코딩을 배우면 실제 관련 기업의 코딩 업무에서 많이 사용되는 코드들을 접하는 것입니다. 이러한 코드들을 많이 접한 여러분이 취직해서 비슷한 일을 한다면 일을 매우 잘 하겠죠? 그러니 열심히 코딩 공부해 주시길 부탁드립니다.

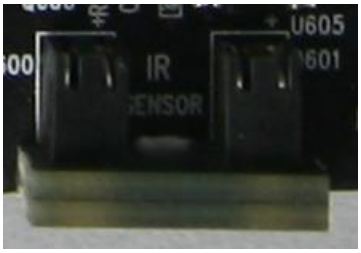


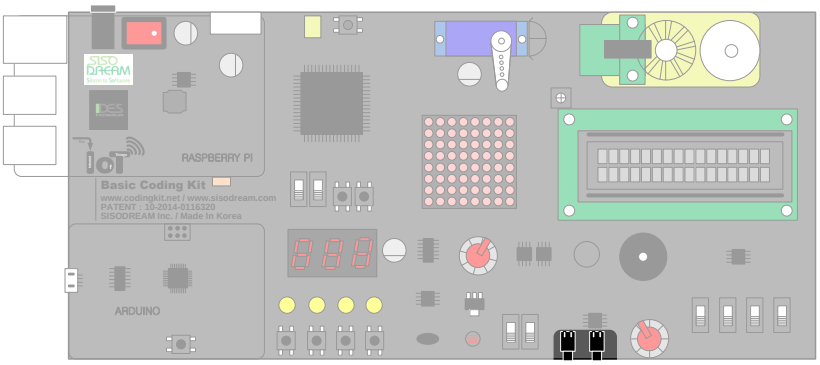
< 예제 코드 : **소리 센서 값 평균 내어 사용하기** >

< 코드 위치 : Coding Kit Ex1 → **sensor_sound_sum >**

[적외선 센서]

적외선(Infrared Rays) 센서를 이용하면 근처에 어떤 물체가 있는지를 알 수 있습니다. 적외선 센서는 발광부와 수광부가 있습니다. 발광부에서 적외선를 발사하면 이 발사된 적외선이 물체에 맞고 반사되어 되돌아 옵니다. 이렇게 되돌아 온 신호를 수광부에서 감지해서 그 값을 전기 신호로 바꾸어 전달합니다. 물체가 가까이 있으면 가까이 있을수록 그 전기 신호 값은 커집니다.





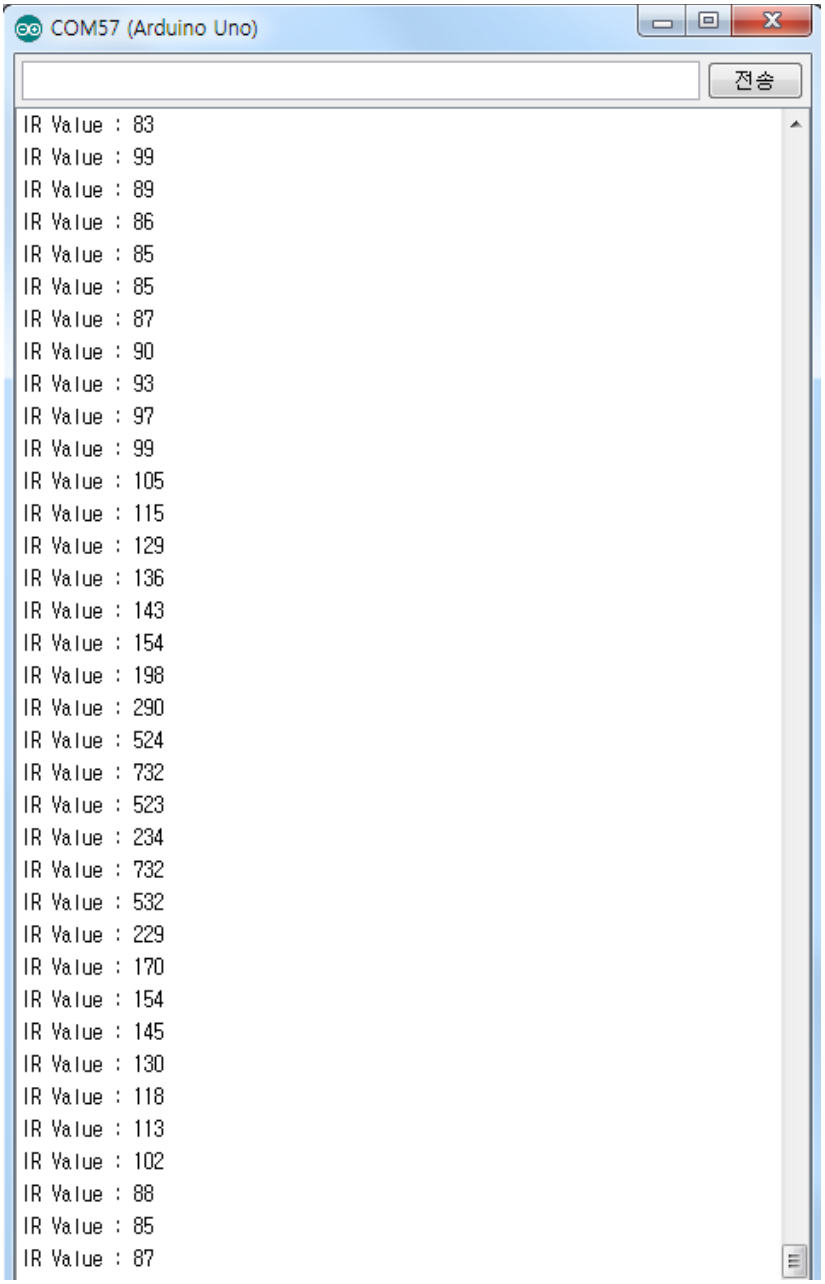
위의 적외선 센서를 보면 발광부와 수광부를 가림막을 씌어 두었습니다. 이것은 자연 상태에서도 적외선은 존재하기 때문에 이러한 적외선들이 적외선 센서로 들어오는 것을 막기 위해 가림막을 씌어 둔 것입니다.

코딩키트를 이용하여 예제를 해 보겠습니다. 코딩은 다음과 같이 합니다.

```
#define SENSOR_IR     A0
#define SENSOR_IR_LED A5

#define ON  1
#define OFF 0

void setup() {
  pinMode(SENSOR_IR_LED, OUTPUT);
  digitalWrite(SENSOR_IR_LED, ON);
  Serial.begin(9600);
}

void loop() {
  int ir_value = analogRead(SENSOR_IR);
  Serial.print("IR Value : ");
  Serial.println(ir_value);
  delay(500);
}
```

적외선 센서는 발광부와 수광부가 있다고 했습니다. 발광부에서 적외선을 발사하는데, 이 적외선은 적외선 LED 를 사용합니다. 그래서 SENSOR_IR_LED 를 정의하였습니다. 수광부는 SENSOR_IR 로 정의하였습니다. setup() 함수에서 SENSOR_IR_LED 를 켜서 적외선를 발사합니다. 이렇게 발사된 적외선은 물체에 반

사되어 다시 되돌아 옵니다. 되돌아온 적외선을 수광부에서 받는데, 이것은 analogRead() 함수를 활용하여 ir_value 변수에 저장합니다. 이 값을 시리얼 모니터로 관찰합니다. 적외선의 반사량은 물체의 색이나 반사면의 형태에 따라서도 달라질 수 있습니다. 그리고 자연 상태에서도 적외선이 존재하기 때문에 수광부에서는 그러한 값들도 취득하게 됩니다. (위에서 설명한 가림막으로도 100 % 막아지는 것은 아닙니다.) 그래서 이 적외선 센서도 튜닝이라는 과정을 거쳐 용도에 맞게 잘 사용하여야 합니다. 위의 코드를 업로드해 보겠습니다. 시리얼 모니터로 보면 다음과 같습니다.



적외선 센서 앞에 어떠한 물체도 없을 때의 값은 85 근처의 값입니다. 1 미터 가량 앞에 흰색 종이를 위치시켰다가 적외선 센서 앞으로 서서히 가까이 위치시키면 적외선 센서 값이 점점 커집니다. 그래서 최대값은 732 까지 올라간 것이 보입니다. 적외선 센서의 또 한가지 특징은 너무 가까이 다가오면 반대로 값이

작아지는 경향이 있습니다. 그래서 위의 시리얼 모니터를 보면 값이 732 였다가 234 까지 작아는 것을 볼 수 있습니다. 이렇게 값이 다시 작아지는 시점은 대략 1 cm 전후인것 같습니다. 이 값도 약간의 편차가 있으니 감안해서 사용하길 바랍니다. 다시 흰색 종이를 멀리하면 적외선 센서 값은 다시 작아 집니다.

수광부와 발광부가 평행하게 배치되는 적외선 센서의 구조 상 적외선이 물체에 반사되어 돌아오기 위해서는 일정 범위 이내의 반사 각이 유지되어야 합니다. 또한 물체가 적외선 센서에 너무 가까이 있는 경우에는 적외선이 물체에 반사되어도 수광부로 돌아오지 못하기 때문에 값이 작아집니다.

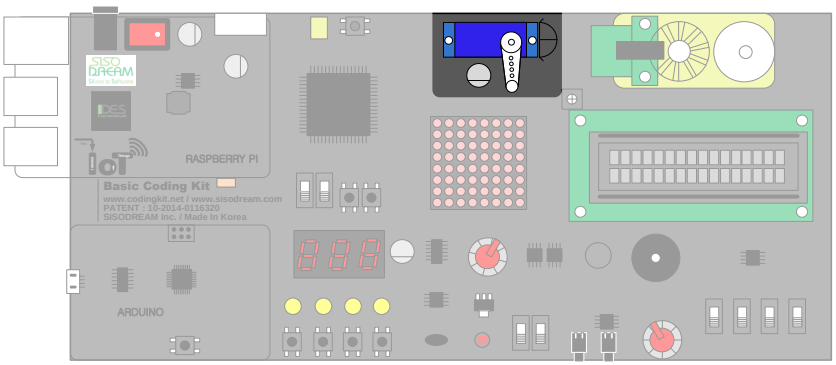
여기서 흰색 종이를 사용하는 것은 적외선은 흰색에서 가장 반사가 잘 되기 때문입니다. 하지만 손이나 기타 다른 물체를 사용 하셔도 괜찮습니다. 하지만 검은 색 물체는 적외선을 흡수하기 때문에 실험이 되지 않습니다.

< 예제 코드 : **적외선 센서 사용하기** >

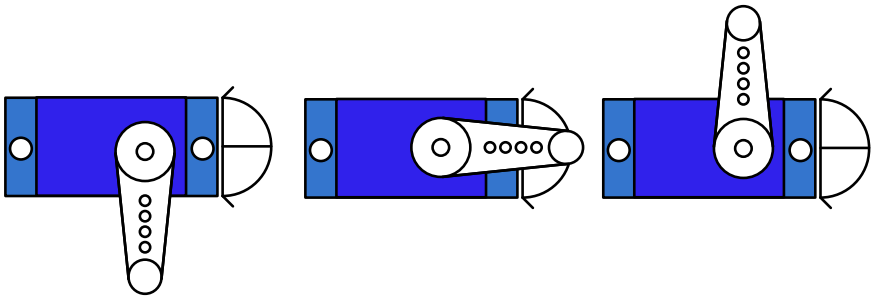
< 코드 위치 : Coding Kit Ex1 → **sensor_ir >**

[서보 모터]

이제부터는 서보 모터(Servo Motor)를 동작시켜 보겠습니다. 서보 모터는 다음 사진과 같고 PWM 신호를 받아서 그 신호에 따라 서보 모터의 혼(Horn(뿔), 사진 참조)을 어떤 각도로 움직입니다. 서보 모터의 혼을 서보혼(Servo Horn)이라고 부릅니다.

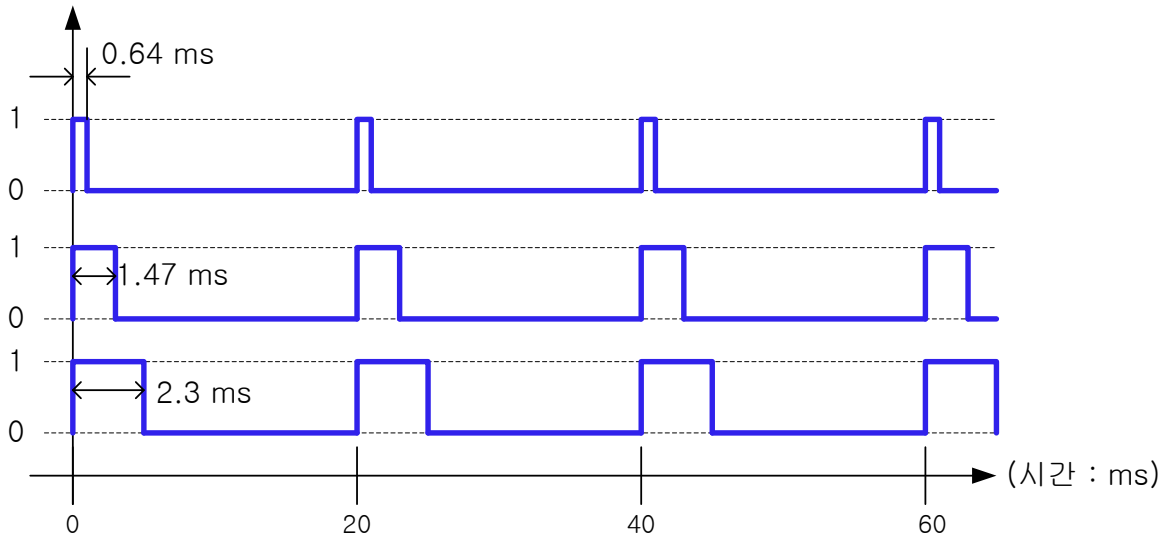


서보 모터는 입력 받는 PWM 신호의 듀티비에 따라서 다음과 같이 서보혼이 어떤 각도로 움직입니다.



서보 모터에 입력되는 PWM 신호의 주기는 20 ms 이고 PWM 신호 중 1 인 구간이 0.64 ms 에서 2.3 ms 범위 내에서 움직입니다. 여기서 ms 는 밀리세크(Mili-second)의 약자이고 1000 분의 1 초입니다. 즉, 1ms 는 0.001 초이고 20ms 는 0.02 초 입니다. 여기서 20 ms 중 0.64 ms 가 1 이면 듀티비는 3.2% 입니다. 2.3 ms 가 1 이면 듀티비는 11.5% 입니다. 이러한 듀티비에 따라서 서보혼이 어떤 각도로 돌아 갑니다. 여기서 서보 모터에 입력되는 PWM 신호의 1 인 구간의 시간이 0.64 ms 이면 0 도이고 2.3 ms 이면 180 도입니다. 서보 모터는 이렇게 PWM 신호의 1 인 구간의 값에 따라서 0 도 ~ 180 도까지 변합니다. 그러면

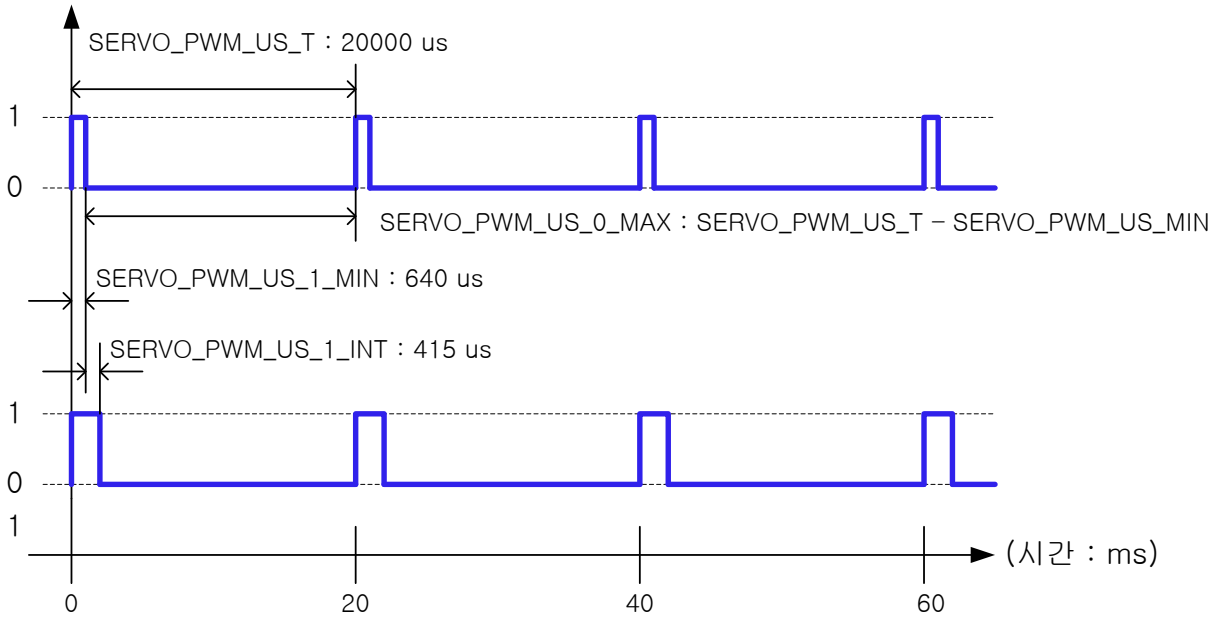
1.47 ms 는 90 도가 될 것입니다. 그래서 서보 모터에 입력되는 PWM 신호는 다음 그림과 같습니다.



위의 그림은 듀티비보다는 시간을 중심으로 그린 것입니다. 모통 서보 모터의 설명이 이렇게 되어 있서 여기서도 시간으로 설명 드렸습니다. 즉, PWM 신호의 주기 20 ms 중 1 인 구간이 0.64 ms, 1.47 ms, 2.3 ms 이렇게 그린 것입니다. 각각은 서보혼을 0 도, 90 도, 180 도로 돌립니다. 그러면 digitalWrite() 함수를 이용하여 PWM 신호를 만들어 서보 모터를 동작시켜 보겠습니다. 먼저 서보 모터에 연결된 핀과 PWM 의 시간을 정의 합니다.

```
#define SERVO_MOT 9

#define SERVO_PWM_US_T     20000
#define SERVO_PWM_US_1_MIN  640
#define SERVO_PWM_US_1_INT  415
#define SERVO_PWM_US_0_MAX  (SERVO_PWM_US_T - SERVO_PWM_US_1_MIN)
```

이 예제에서는 지금까지 사용했던 delay() 함수 이외에 delayMicroseconds() 함수를 사용하겠습니다. delay() 함수가 1,000 분의 1 초인 ms 단위라면 delayMicroseconds() 함수는 1,000,000 분의 1 초인 us (Micro-second) 단위입니다. 그래서 서보 모터의 PWM 신호의 주기인 20 ms 는 20,000 us 이고 이 값을 SERVO_PWM_US_T(T 는 주로 주기를 표시 함)로 정의하였습니다. PWM 신호의 1 인 구간의 최소값인 0.64 ms 는 640 us 로 SERVO_PWM_US_1_MIN 으로 정의 하였고, SERVO_PWM_US_T 에서 SERVO_PWM_US_1_MIN 을 뺀 PWM 신호의 0 인 구간의 최대값을 SERVO_PWM_US_0_MAX 로 정의하였습니다. SERVO_PWM_US_1_INT(INT 는 Interval 의 약자)는 서버혼의 각도를 45도 증가시키고자 할 때 PWM 신호의 1 인 구간의 증가분을 정의한 것입니다. 즉, 이 값이 증가하면 서보혼도 45도 만큼의 각도를 증가하여 움직입니다. 위의 정의를 PWM 신호 그림으로 나타내면 다음과 같습니다.



setup() 함수에서는 SERVO_MOT 의 핀 모드를 출력으로 하였습니다.

```
void setup() {
  pinMode(SERVO_MOT, OUTPUT);
}
```

loop() 함수에서는 위의 시간 정의를 이용하여 PWM 신호를 생성합니다.

```
void loop() {
  for (int i = 0; i < 5; i++) {
    for (int j = 0; j < 100; j++) {
      digitalWrite(SERVO_MOT, 1);
      delayMicroseconds(SERVO_PWM_US_1_MIN + (SERVO_PWM_US_1_INT * i));
      digitalWrite(SERVO_MOT, 0);
      delayMicroseconds(SERVO_PWM_US_0_MAX - (SERVO_PWM_US_1_INT * i) - 10000);
      delay(10);
    }
  }
  delay(1000);
}
```

SERVO_PWM_US_1_INT 값은 for 문의 인덱스 i 와 곱해져서 PWM 신호의 1 인 구간이 그 값의 배수만큼 증가합니다. 이렇게 만들어진 PWM 신호는 "for (int j = 0; j < 100; j++)" 에 의해서 100 번씩 반복됩니다. 그러면 100번씩 반복되는 동안은 PWM 신호는 고정된 파형을 그릴 것이고 서보 모터는 어떤 각도에 멈추어 있을 것입니다.

위 코드에서 붉은색 부분은 PWM 신호의 0 인 구간을 나타낸 것입니다. delayMicroseconds() 함수 인자의 최대값은 16383 입니다. 그런데, 우리는 여기에 최대 19360 (20000 - 640) 을 입력해야 합니다. 그래서

delayMicroseconds() 함수에서 10000 을 빼서 delay(10) 으로 바꾸었습니다. 10000 us 는 10 ms 와 같으므로 이렇게 처리한 것입니다.

< 예제 코드 : **디지털 신호로 서보 모터 컨트롤하기** >

< 코드 위치 : Coding Kit Ex1 → **servo_pwm_digital** >

이제 코드를 업로드해 볼까요. 어떻습니까? 서보 모터가 잘 동작하나요? 각도는 맞게 가는 것 같은데, 간혹 이상하게 서보 모터에서 이상한 소리가 나면서 안정적이지 않고 서보혼이 떨리기도 하고 하지요. 이것은 여러 원인이 있을 수 있지만 그 중 하나가 digitalWrite() 함수와 delayMicrosendonds() 함수로는 정확한 PWM 신호를 못 만들기 때문입니다. 이렇게 만들어진 PWM 신호는 오차가 많이 발생합니다. 그래서 서보 모터가 안정적으로 동작하지 못하는 것입니다. 그래서 부저나 DC 모터에서도 digitalWrite() 함수를 사용하지 않고 analogWrite() 함수나 tone() 함수 등을 사용하여 PWM 신호를 만들었던 것입니다. 그러면 서보 모터에서도 analogWrite() 함수나 부저에서 사용했던 tone() 함수를 사용할 수 있을까요? 먼저 analogWrite() 함수부터 알아보면, analogWrite() 함수는 주기가 0.002 초와 0.004 초로 고정되어 있어 주기가 0.02 초인 서보 모터의 PWM 신호를 만들 수 없습니다. tone() 함수는 주기를 원하는만큼 조절할 수 있어 서버 모터의 PWM 신호의 주기인 0.02 초의 주기를 만들 수 있습니다. 하지만 듀티비가 50% 로 고정되어 있어 듀티비를 바꾸어서 서보혼의 각도를 바꾸는 서보 모터 입력 PWM 신호를 만들 수는 없습니다. 이러한 이유로 analogWrite() 함수와 tone() 함수로는 서보 모터를 제대로 컨트롤 하기에는 부족합니다. 그래서 아두이노에서는 서보 모터를 위한 새로운 라이브러리를 제공합니다. 프로그램 언어에서 라이브러리라는 것은 미리 작성해둔 함수나 정의 코드의 모음입니다. 아두이노의 서보함수 라이브러리는 코딩키트에서 사용하고 있는 서보 모터와 각도를 정하는 PWM 신호의 듀티비가 조금 다릅니다. 그래서 이 값을 조정해야 합니다. 다음 폴더에서 Servo.h 파일을 엽니다.

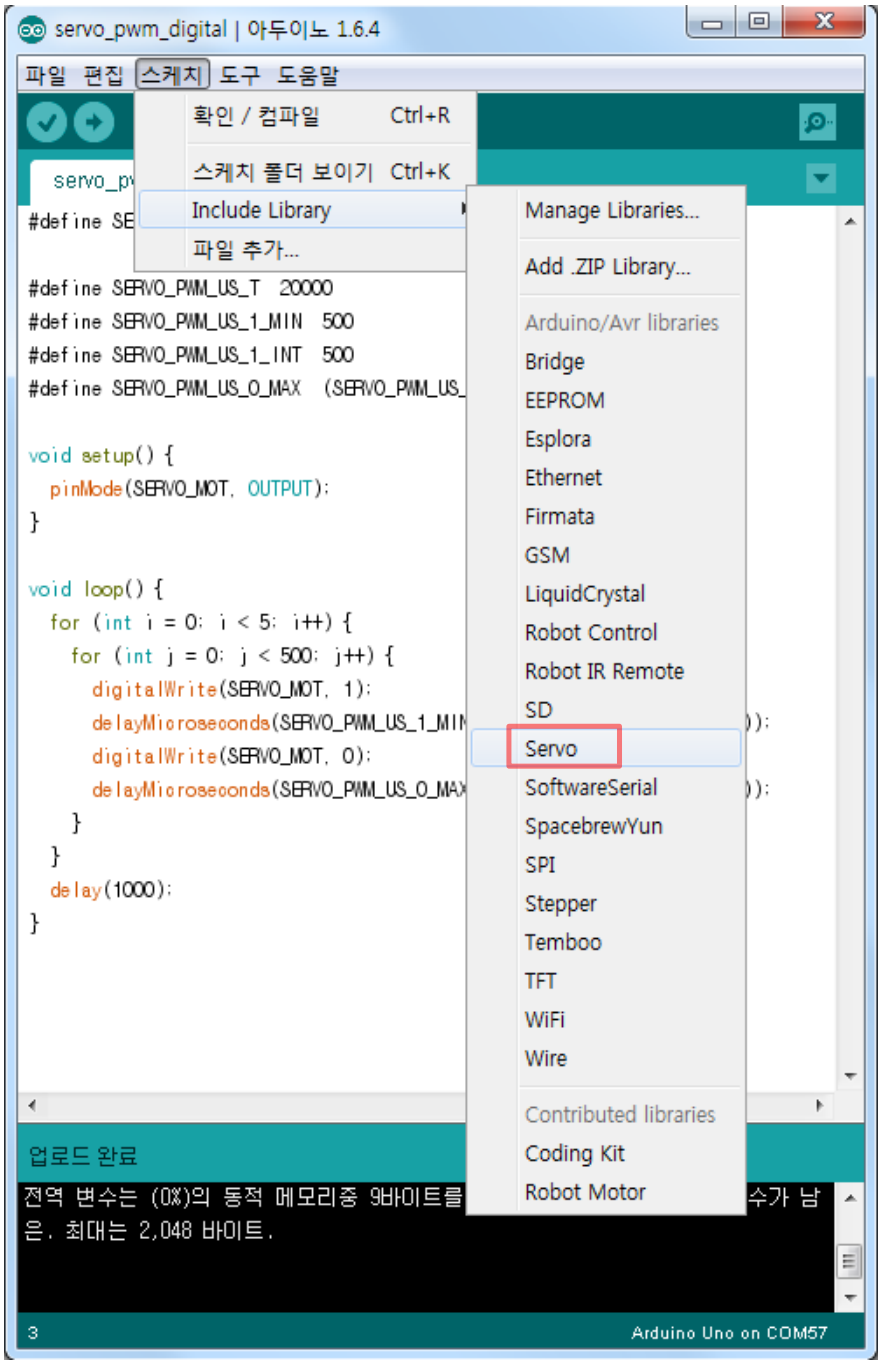
C:₩Program Files (x86)₩Arduino₩libraries₩Servo₩src

Servo.h 파일의 다음 항목을 수정합니다.

```
//#define MIN_PULSE_WIDTH       544     // the shortest pulse sent to a servo
//#define MAX_PULSE_WIDTH       2400    // the longest pulse sent to a servo
#define MIN_PULSE_WIDTH       640     // the shortest pulse sent to a servo
#define MAX_PULSE_WIDTH       2300    // the longest pulse sent to a servo
```

PWM 신호의 1인 구간(Pulse)의 최소값과 최대값을 조정하는 것입니다. 만약 해당 폴더 및 파일의 권한 문제 때문에 바로 수정이 안 될 경우에는 파일을 다른 폴더로 옮겨서 수정한 이후에 다시 제자리로 돌려둡니다.

서모 보터를 위한 라이브러리는 다음 그림과 같이 아두이노 프로그램에서 "스케치 → Include Library → Servo" 메뉴를 선택하여 추가할 수 있습니다.



이 라이브러리를 추가하면 코드의 제일 위에 다음 코드가 추가 됩니다.

```
#include <Servo.h>
```

이것은 Servo 라이브러리를 사용하겠다는 것입니다. 그래서 굳이 "스케치 → Include Library → Servo" 메뉴를 선택하지 않아도 코드에 "#include <Servo.h>" 만 추가해 주면 서보 모터를 동작실킬 수 있는 라이브러리를 사용할 수 있습니다. 이제 이 Servo 라이브러리를 이용하여 앞에서 했던 예제와 같이 서보 모터의 각도를 움직여 보도록 하겠습니다.

```
#include <Servo.h>

Servo ck_servo;   // Coding Kit Servo

#define SERVO_MOT 9

#define SERVO_ANGLE_MIN  0
#define SERVO_ANGLE_INT  45

void setup() {
  ck_servo.attach(SERVO_MOT);
}

void loop() {
  for (int i = 0; i < 5; i++) {
    ck_servo.write(SERVO_ANGLE_MIN + (SERVO_ANGLE_INT * i));
    delay(2000);    // 20ms * 100 = 2000 ms
  }
  delay(1000);
}
```

먼저 Servo 라이브러리의 함수들을 사용하기 위해서는 Servo 라는 변수 타입의 변수를 선언해 주어야 합니다. 그런데, Servo 라는 변수 타입도 있었나? 하는 의문이 드실 것입니다. 이 변수 타입은 Servo 라이브러리에서 정의한 변수 타입인 것입니다. 변수 타입은 스트럭처, 클래스 등의 여러 가지 방법으로 정의하여 사용할 수 있습니다. 이러한 방법은 매우 어려운 부분으로 여러분이 앞으로 코딩의 전문가가 되었을 때 꼭 해 보도록 하십시요. Servo 변수 타입으로 ck_servo 변수를 정의하였습니다. 이렇게 하면 Servo 라이브러리에서 정의한 함수들을 사용할 수 있습니다.

Servo 라이브러리의 attach() 라는 함수를 사용하려고 합니다. 그런데, 이 때는 ck_servo.attach() 라고 써서 Servo 변수 타입으로 정의한 변수 ck_servo 에 있는 함수를 사용한다고 알려 주는 것입니다. attach() 함수의 인자로는 서보 모터에 연결된 핀 번호를 알려 줍니다.

loop() 함수에서는 ck_servo.write() 함수를 사용하였습니다. 이 함수는 서보 모터의 각도를 인자로 받습니다. 이렇게 각도를 입력해 주면 해당 각도에 맞는 PWM 신호를 생성해 주고 이렇게 생성된 PWM 신호에 의해서 서보 모터는 해당 각도로 움직이는 것입니다.

이 각도는 0 도 ~ 180 도를 움직입니다. 가장 작은 각도인 0 도를 SERVO_ANGLE_MIN 으로 정의하였습니다. 서보 모터를 45 도 단위로 움직이기 위해서 SERVO_ANGLE_INT 을 45 로 정의 하였습니다. 이 정의 값들은 loop() 함수의 for 문에서 사용되었습니다. 이렇게 하면 서보 모터는 0 도, 45 도, 90 도, 180 도를 움직입니다.

이제 코드를 업로드 하여 서보 모터의 동작을 확인합니다. 이전에 digitalWrite() 함수와 delayMicroseconds() 함수를 활용한 서보 모터의 동작보다 훨씬 더 원활히 움직이는 것을 확인할 수 있을

것입니다. 이렇게 Servo 라이브러리를 사용하니까 서보 모터를 동작시키는 것이 참 쉽죠? 하지만 여러분은 코딩 공부를 하는 것이기 때문에 digitalWrite() 함수를 이용하여 서보 모터를 동작시키는 코드도 잘 알아두도록 하십시요.

< 예제 코드 : **Servo 라이브러리를 사용하여 서보 모터 동작 시키기** >

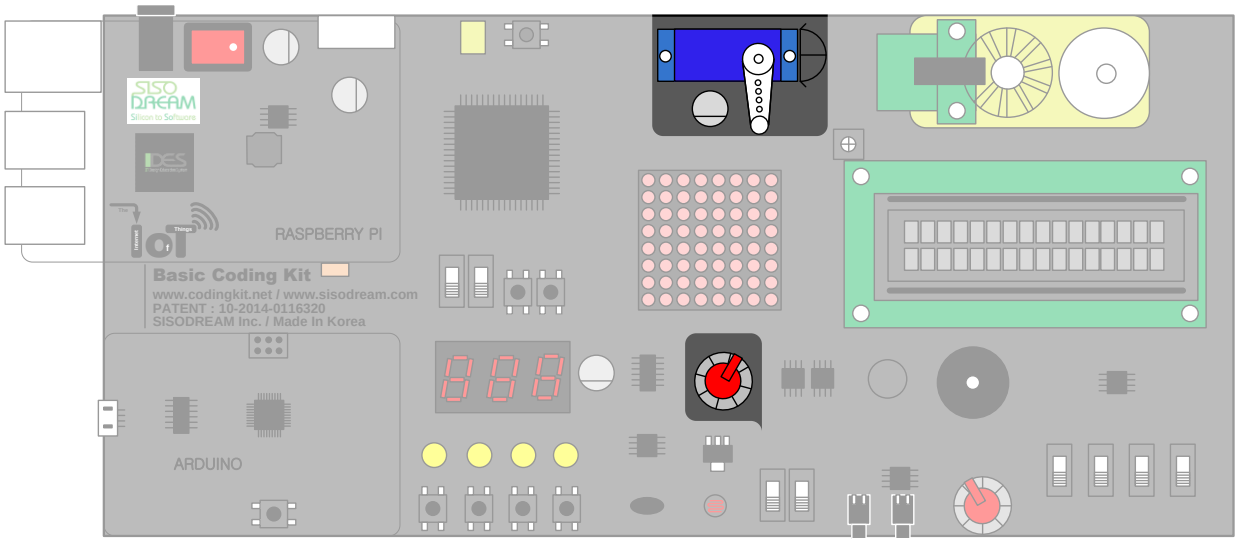
< 코드 위치 : Coding Kit Ex1 → **servo_pwm_lib >**

< 연습 문제 : **가변 저항으로 서보 모터 컨트롤 하기** >

간단한 예제 하나 해 볼까요? 가변 저항을 돌리면 서보 모터의 각도가 돌아가는 예제를 해 보겠습니다. 가변 저항의 값은 0 ~ 1023 까지 변하는데, 서보 모터는 0 ~ 180 도 까지 변합니다. 이것은 map() 함수를 활용하면 되겠지요. 정의와 setup() 함수는 다음과 같습니다.

```
#include <Servo.h>

Servo ck_servo;   // Coding Kit Servo

#define SERVO_MOT    9
#define VAR_RES_1    A3

void setup() {
  ck_servo.attach(SERVO_MOT);
  Serial.begin(9600);
}
```

loop() 함수는 여러분이 해 보십시요. 어렵지 않아요.

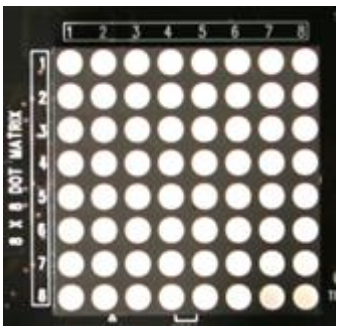


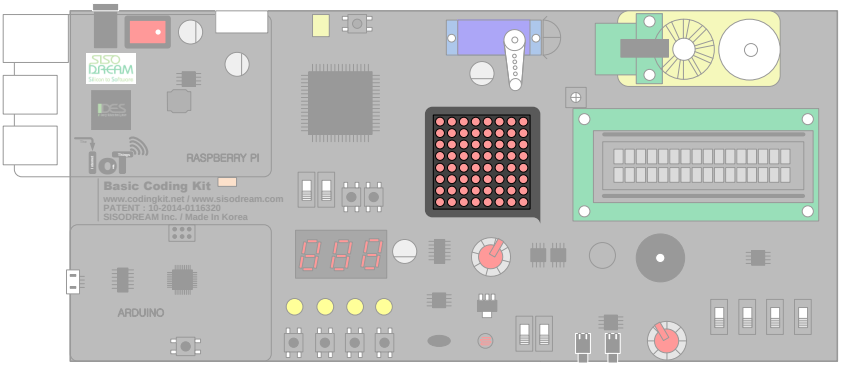
가변 저항을 사용하실 때는 ADC 모드 스위치를 위로 올리는 것 잊지 마세요.

< 코드 위치 : Coding Kit Ex1 → **servo_vr** >

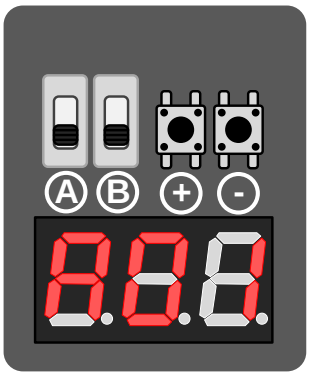
[도트매트릭스(Dotmatrix)에 하트 그리기]

도트매트릭스는 아래 사진과 같이 LED 를 여러 개 촘촘히 모아둔 모양입니다.





이 LED 를 하나 하나씩 별도로 켤 수 있기 때문에 원하는 문자나 그림을 표시할 수 있습니다. 도트매트릭스의 스위칭 ID A01 입니다. 스위치와 버튼을 이용하여 스위칭 ID 를 A01 로 바꾸겠습니다.

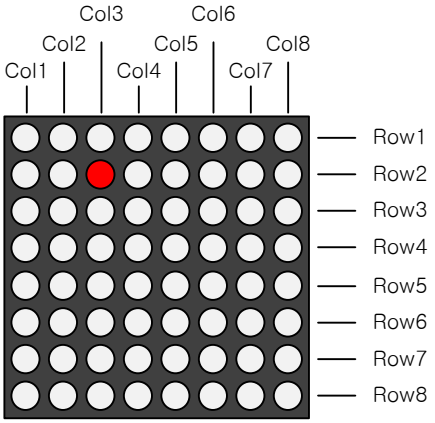


그런데, 이 LED 하나 하나가 다 아두이노의 핀에 연결되어 컨트롤하는 것은 아닙니다. 도트매트릭스에는 가로, 세로 8x8 해서 64 개의 LED 가 들어 있습니다. 이것은 가로줄 8 개와 세로줄 8 개로 구성이 됩니다. 아두이노에 가로줄 8 개가 8 핀에 연결이 되고, 세로줄 8 개가 8 핀에 연결이 됩니다. 그래서 총 소요되는 핀의 수는 16 핀입니다. 그 핀들은 다음과 같이 정의 합니다.

```
#define DM_ROW1     1
#define DM_ROW2     7
#define DM_ROW3     10
#define DM_ROW4     6
#define DM_ROW5     4
#define DM_ROW6     5
#define DM_ROW7     12
#define DM_ROW8     13

#define DM_COL1     0
#define DM_COL2     11
#define DM_COL3     8
#define DM_COL4     2
```

```
#define DM_COL5          9
#define DM_COL6          3
#define DM_COL7          A4
#define DM_COL8          A5
```

위에서 정의된 핀들은 도트매트릭스와는 아래와 같이 연결됩니다.



도트매트릭스의 각 LED 는 가로줄(Row)과 새로줄(Col)이 교차하는 곳의 LED 를 가로 줄 번호와 새로줄 번호로 켜 줍니다. 예를 들어 다음과 같이 코딩하면 위 그림에서 붉은색 부분이 켜집니다.

```
digitalWrite(DM_ROW2, 1);
digitalWrite(DM_COL3, 0);
```

위와 같이 가로 줄의 2 번에 1 을 쓰고 세로 줄 3 번에 0 을 쓰면 해당 위치가 켜집니다. 그리고 여기서 한 가지 더 알 수 있는 것은 교차 점의 LED 를 켜 주려면 가로 줄에는 1 을, 세로 줄에는 0 을 써 줘야 한다는 것입니다. 그래서 이것을 다음과 같이 정의해 줍니다.

```
#define ROW_ON    1
#define ROW_OFF   0

#define COL_ON    ROW_OFF
#define COL_OFF   ROW_ON
```

이 정의를 이용하면 위의 코드는 다음과 같이 바꿀 수 있습니다.

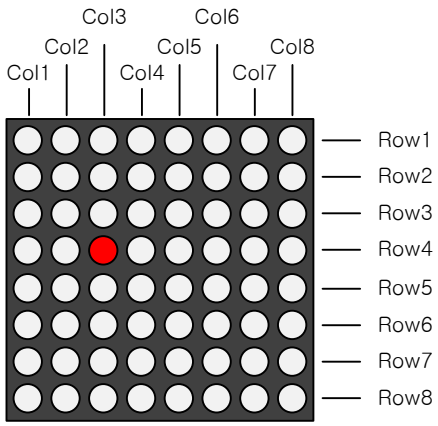
```
digitalWrite(DM_ROW2, ROW_ON);
digitalWrite(DM_COL3, COL_ON);
```

어때요? 훨씬 더 알아 보기 좋죠.

위의 코드에 의해서 가로 2번 줄과 세로 3번 줄은 모두 ON 상태가 되고, 두 줄이 만나는 교차점인 (2, 3)의 LED 가 켜집니다.

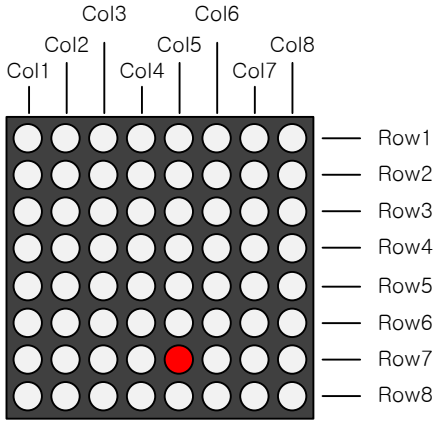
몇 가지 더 예를 들어 보겠습니다. 다음과 같이 가로 4번 줄이 ON 상태가 된다면, 가로 4번과 세로 3번줄이 만나는 교차점 (4, 3)의 LED 가 켜지게 됩니다.

```
digitalWrite(DM_ROW4, ROW_ON);
digitalWrite(DM_COL3, COL_ON);
```



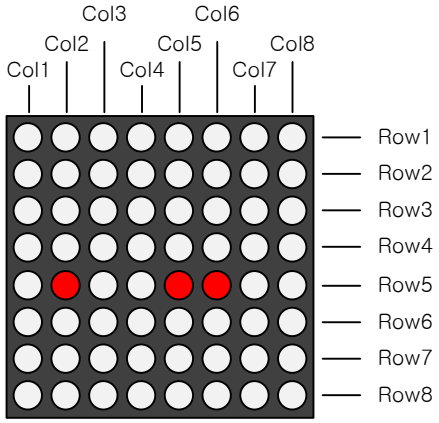
다음과 같이 가로 7번줄과 세로 5번줄이 ON 이 되면 교차점 (7, 5)에 LED 가 켜지게 되는 겁니다.

```
digitalWrite(DM_ROW7, ROW_ON);
digitalWrite(DM_COL5, COL_ON);
```



다음과 같이 하면 한 줄에 여러 개의 LED 를 켤 수도 있습니다. 다음은 (5, 2), (5, 5), (5, 6) 을 켜는 것입니다.

```
digitalWrite(DM_ROW5, ROW_ON);
digitalWrite(DM_COL2, COL_ON);
digitalWrite(DM_COL5, COL_ON);
digitalWrite(DM_COL6, COL_ON);
```



그러면 이제 도트매트릭스의 모든 LED 를 켜는 예제를 해 보겠습니다. 코드는 다음과 같습니다.

```
#define DM_ROW1          1
#define DM_ROW2          7
#define DM_ROW3          10
#define DM_ROW4          6
#define DM_ROW5          4
#define DM_ROW6          5
#define DM_ROW7          12
#define DM_ROW8          13

#define DM_COL1          0
#define DM_COL2          11
#define DM_COL3          8
#define DM_COL4          2
#define DM_COL5          9
#define DM_COL6          3
#define DM_COL7          A4
#define DM_COL8          A5

#define ROW_ON    1
#define ROW_OFF   0

#define COL_ON    ROW_OFF
#define COL_OFF   ROW_ON

void setup() {
  pinMode(DM_ROW1, OUTPUT);
  pinMode(DM_ROW2, OUTPUT);
  pinMode(DM_ROW3, OUTPUT);
  pinMode(DM_ROW4, OUTPUT);
  pinMode(DM_ROW5, OUTPUT);
  pinMode(DM_ROW6, OUTPUT);
  pinMode(DM_ROW7, OUTPUT);
  pinMode(DM_ROW8, OUTPUT);
```

```
  pinMode(DM_COL1, OUTPUT);
  pinMode(DM_COL2, OUTPUT);
  pinMode(DM_COL3, OUTPUT);
  pinMode(DM_COL4, OUTPUT);
  pinMode(DM_COL5, OUTPUT);
  pinMode(DM_COL6, OUTPUT);
  pinMode(DM_COL7, OUTPUT);
  pinMode(DM_COL8, OUTPUT);

  digitalWrite(DM_ROW1, ROW_ON);
  digitalWrite(DM_ROW2, ROW_ON);
  digitalWrite(DM_ROW3, ROW_ON);
  digitalWrite(DM_ROW4, ROW_ON);
  digitalWrite(DM_ROW5, ROW_ON);
  digitalWrite(DM_ROW6, ROW_ON);
  digitalWrite(DM_ROW7, ROW_ON);
  digitalWrite(DM_ROW8, ROW_ON);

  digitalWrite(DM_COL1, COL_ON);
  digitalWrite(DM_COL2, COL_ON);
  digitalWrite(DM_COL3, COL_ON);
  digitalWrite(DM_COL4, COL_ON);
  digitalWrite(DM_COL5, COL_ON);
  digitalWrite(DM_COL6, COL_ON);
  digitalWrite(DM_COL7, COL_ON);
  digitalWrite(DM_COL8, COL_ON);
}

void loop() {
}
```

별로 하는 것도 없는데, 코드가 너무 길죠. 이것은 도트매트릭스에 연결되는 핀이 16 개나 되어 너무 많아서 그럽니다. 각 핀들을 정의해 주고, 그 핀들을 일일이 핀 모드 설정하고, 값 주고 하는 작업들이 많은 것이지요. 이것을 짧게 할 수 없을까요? 있겠죠. 있으니깐 얘기를 꺼내겠죠.

스위칭 ID 는 A01 로 바꾸어 준 다음 코딩키트에 업로드하여 동작을 확인해 보세요.

< 예제 코드 : **도트매트릭스 켜기** >

< 코드 위치 : Coding Kit Ex1 → **dotmat >**

< 문법 설명 : **배열** >

배열을 이용하면 위의 긴 코드를 짧게 끝낼 수 있습니다. 배열이라는 것이 무엇이냐면, 어떤 값들을 모아 둔 것을 배열이라고 합니다. 그러면 위의 코드에서 가로줄 핀을 다음과 같이 모아 보겠습니다.

```
int dm_row[8] = {
  DM_ROW1, DM_ROW2, DM_ROW3, DM_ROW4,
  DM_ROW5, DM_ROW6, DM_ROW7, DM_ROW8};
```

위와 같이 하는 것을 배열 변수 선언이라고 합니다. 배열 변수를 선언하는 문법식은 다음과 같습니다.

```
변수_타입 변수_이름[원소수] = { 원소_1, 원소_2, ..., 원소_N };
```

먼저 변수 타입을 써 줍니다. 위에서는 int 로 해 주었습니다. 그 다음 변수 이름을 써 주고 대괄호([ ]) 안에 배열에 포함되는 원소의 수를 적어 줍니다. 배열에 포함되는 항목을 원소라고 합니다. 위 코드에서는 원소수를 8 로 해 주었군요. 이것은 가로줄이 8 핀이기 때문입니다. 그 다음 등호(=)를 하고 중괄호({ }) 안에 각 원소을 나열합니다. 여기서는 가로줄 핀들을 적어주었습니다. 원소수가 N 이라면 마지막 원소는 원소_N 으로 표기합니다.

배열에 포함되어 있는 각 원소을 사용하려면 인덱스라는 것을 활용합니다. 배열 안의 원소는 첫번째 원소부터 0 번으로 해서 1 씩 증가하는 인덱스를 붙여 줍니다. 위의 코드에서는 DM_ROW1 이 인덱스 0 번, DM_ROW2 가 인덱스 1 번으로 해서 마지막 원소까지 인덱스를 붙입니다. 그리고 "배열_변수[인덱스]" 를 이용하여 배열에 포함된 해당 항목을 사용합니다. 예를 들어 위의 코드에서 dm_row[0] 이라고 하면 DM_ROW1 을 가리키는 것이 됩니다. dm_row[3] 하면 DM_ROW4 가 되는 것이지요. 그러면 위에서 보았던 다음 코드는 배열 변수 dm_row 를 활용하면 어떻게 바꿀 수 있을까요.

```
digitalWrite(DM_ROW2, ROW_ON);
```

DM_ROW2 는 dm_row[1] 이므로 다음과 같이 쓸 수 있습니다.

```
digitalWrite(dm_row[1], ROW_ON);
```

그러면 이제 배열을 어떻게 사용하는지는 알았고, 어떻게 배열을 활용하여 코드를 줄일 수 있는지를 알아보겠습니다. 다음과 같이 핀 모드를 설정하는 코드를 줄여 보겠습니다.

```
pinMode(DM_ROW1, OUTPUT);
pinMode(DM_ROW2, OUTPUT);
pinMode(DM_ROW3, OUTPUT);
pinMode(DM_ROW4, OUTPUT);
pinMode(DM_ROW5, OUTPUT);
pinMode(DM_ROW6, OUTPUT);
pinMode(DM_ROW7, OUTPUT);
pinMode(DM_ROW8, OUTPUT);
```

일단 위의 코드를 핀 이름 대신 배열을 사용하여 바꾸면 다음과 같습니다.

```
pinMode(dm_row[0], OUTPUT);
pinMode(dm_row[1], OUTPUT);
pinMode(dm_row[2], OUTPUT);
pinMode(dm_row[3], OUTPUT);
pinMode(dm_row[4], OUTPUT);
```

```
pinMode(dm_row[5], OUTPUT);
pinMode(dm_row[6], OUTPUT);
pinMode(dm_row[7], OUTPUT);
```

위의 코드는 for 문을 이용하면 다음과 같이 간단하게 끝납니다.

```
for (int i = 0; i < 8; i++) {
  pinMode(dm_row[i], OUTPUT);
}
```

for 문의 인덱스 i 는 0 에서 7 까지 변합니다. 그래서 pinMode(dm_row[i], OUTPUT); 에서 dm_row[i] 는 dm_row[0], dm_row[1], … 이렇게 실행되면서 모든 가로핀의 핀 모드를 설정해 나갑니다. 그러면 위의 도트메트릭스 전체 LED 를 켜는 코드는 다음과 같이 바꿀 수 있습니다.

```
void setup() {
  for (int i = 0; i < 8; i++) {
    pinMode(dm_row[i], OUTPUT);
    pinMode(dm_col[i], OUTPUT);
  }
  for (int i = 0; i < 8; i++) {
    digitalWrite(dm_row[i], ROW_ON);
    digitalWrite(dm_col[i], COL_ON);
  }
}
```

어때요? 배열과 for 문을 활용하니 매우 짧은 코드로 바뀌었죠. 이게 바로 코딩 공부의 재미고, 코딩 실력 향상입니다. 더불어 위와 같이 코드를 줄이기 위한 노력이 분석적이고 논리적인 사고도 발달시킬 것입니다.

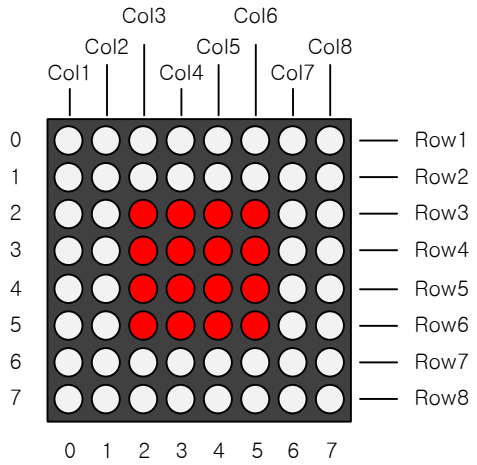
< 예제 코드 : **배열을 이용한 도트매트릭스 켜기** >

< 코드 위치 : Coding Kit Ex1 → **dotmat_array >**

이제 코드를 업로드해 보도록 하겠습니다.

< 연습 문제 : **도트매트릭스 일부 켜기** >

다음과 같이 도트매트릭스 가운데 4x4 를 켜 보도록 하겠습니다.



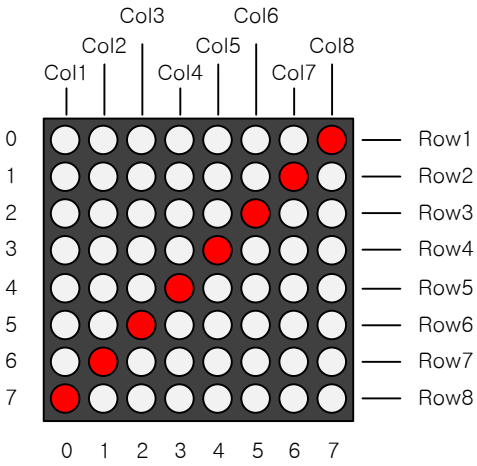
도트매트릭스 전체를 켜는 코드의 setup() 함수에서 for 문 안에 if 문을 활용하여 인덱스 i 가 어떤 범위 안에 일 때만 도트매트릭스를 켜고 그 이외의 인덱스 값일 때는 도트매트릭스를 꺼주는 코딩을 해 주시면 쉽게 할 수 있을 것입니다. 그리고 도트매트릭스를 끄는 것은 다음과 같이 해 주면 됩니다.

```
digitalWrite(dm_row[i], ROW_OFF);
digitalWrite(dm_col[i], COL_OFF);
```

이제 한 번 직접 코딩해 보시죠. Col3, 4, 5, 6 그리고 Row3, 4, 5, 6 이 만나는 교차점만 ON이 되게 해주면 됩니다.

< 코드 위치 : Coding Kit Ex1 → **dotmat_4x4_on** >

도트매트릭스에 이렇게 사각형이 아닌 어떤 모양을 표현하기 위해서는 도트매트릭스를 한줄씩 켜 주어야 합니다. 그 이유를 설명하기 위해 예를 들면 다음과 같은 대각선을 도트매트릭스에 켜 보도록 하겠습니다.



첫번째 줄을 켜는 시간 동안에는 Row1 과 Col8 만을 ON 시켜 (Row1, Col8) 좌표의 LED 만 켜지게 합니다. 두번째 줄을 켜는 시간 동안에는 Row2 과 Col7 만을 ON 시켜 (Row2, Col7) 좌표의 LED 만 켜지게 합니다. 이렇게 하려면 한 줄씩 컨트롤해야 합니다. 그리고 해당 줄에서는 하나의 LED 만 켜집니다. 그래서 다음과 같이 한 줄에 하나의 LED 만 켜는 함수를 만들었습니다.

```
void dm_on_1(int row, int col) {
  dm_off_all();
  digitalWrite(dm_row[row], ROW_ON);
  digitalWrite(dm_col[col], COL_ON);
}
```

먼저 다음과 같은 dm_off_all() 함수를 호출하여 도트매트릭스의 모든 LED 를 꺼 줍니다.

```
void dm_off_all() {
  for (int i = 0; i < 8; i++) {
    digitalWrite(dm_row[i], ROW_OFF);
    digitalWrite(dm_col[i], COL_OFF);
  }
}
```

모든 LED 를 꺼 준 다음에는 위의 붉은색 코드처럼 해당하는 가로, 세로줄의 LED 를 켜 주기 위해서 배열로 선언된 dm_row 와 dm_col 에 매개변수로 받은 row, col 을 인덱스로 하여 해당 LED 를 켜 줍니다. 이 두함수를 이용하여 대각선의 LED 를 켜는 loop() 함수 코드는 다음과 같습니다.

```
void loop() {
  dm_on_1(0, 7);
  delay(2);
  dm_on_1(1, 6);
  delay(2);
  dm_on_1(2, 5);
  delay(2);
  dm_on_1(3, 4);
  delay(2);
  dm_on_1(4, 3);
  delay(2);
  dm_on_1(5, 2);
  delay(2);
  dm_on_1(6, 1);
  delay(2);
  dm_on_1(7, 0);
  delay(2);
}
```

dm_on_1() 함수에 켜 주고 싶은 LED 의 위치를 알려 줍니다. 이 위치는 1 이 아닌 0 부터 시작합니다. 그것은 dm_on_1() 함수에서 배열에 이 값을 바로 주기 때문입니다. 배열은 인덱스가 0 부터 시작합니다.

코딩키트에서 실행해 보면 한줄씩 켜 주었는데도 모두 켜진 것처럼 보이지요. 이것은 한줄씩 켜 있는 시

간을 매우 짧게 하여 모든 줄이 빠르게 바뀌면서 켜 지면 모두 켜진 것처럼 보입니다.

< 예제 코드 : **도트매트릭스의 대각선 LED 만 켜기** >

< 코드 위치 : Coding Kit Ex1 → **dotmat_diag >**

위의 코드의 loop() 함수에 반복되는 코드가 많습니다. 그래서 그 부분을 for 문으로 바꾸어 보았습니다. 이렇게 코드를 수정하면 dm_on_1() 함수도 필요 없습니다.

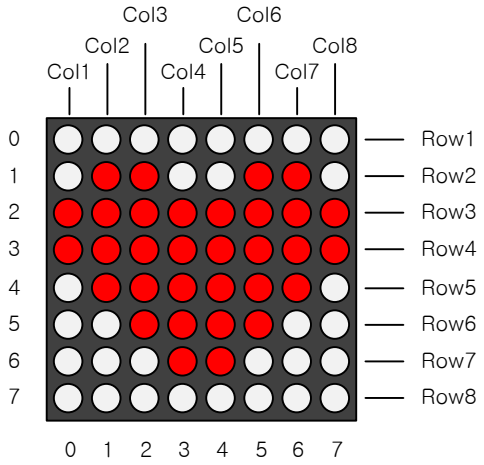
```
void loop() {
  for (int i = 0; i < 8; i++) {
    dm_off_all();
    digitalWrite(dm_row[i], ROW_ON);
    digitalWrite(dm_col[7-i], COL_ON);
    delay(2);
  }
}
```

코드를 길게 설명드리지는 않겠습니다. 여러분이 직접 분석해 보십시요. 코드를 분석하는 것도 좋은 공부입니다. 혹시 이해가 되지 않으시면 코딩 사이트로 질문해 주십시요.

< 예제 코드 : **for 문을 이용한 도트매트릭스의 대각선 LED 만 켜기** >

< 코드 위치 : Coding Kit Ex1 → **dotmat_diag_for >**

이렇게 단순한 형태 말고 다음과 같은 하트 모양을 한 번 켜 보도록 하겠습니다.



하트 모양을 켜는 예제는 앞의 예제들 보다 조금 어렵습니다. 하지만 이 예제를 해 보면 도트매트릭스에

어떤 모양도 만들수 있는 방법이 생깁니다. 그럼 코드를 보겠습니다. 도트매트릭스 핀 정의한 부분은 제외합니다.

```
#define ROW_ON    1
#define ROW_OFF   0

#define COL_ON    ROW_OFF
#define COL_OFF   ROW_ON

int dm_row[8] = {
  DM_ROW1, DM_ROW2, DM_ROW3, DM_ROW4,
  DM_ROW5, DM_ROW6, DM_ROW7, DM_ROW8};

int dm_col[8] = {
  DM_COL1, DM_COL2, DM_COL3, DM_COL4,
  DM_COL5, DM_COL6, DM_COL7, DM_COL8};

int dm_heart[8][8] = {
  {1, 1, 1, 1, 1, 1, 1, 1},
  {1, 0, 0, 1, 1, 0, 0, 1},
  {0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 0},
  {1, 0, 0, 0, 0, 0, 0, 1},
  {1, 1, 0, 0, 0, 0, 1, 1},
  {1, 1, 1, 0, 0, 1, 1, 1},
  {1, 1, 1, 1, 1, 1, 1, 1}
};

void setup() {
  for (int i = 0; i < 8; i++) {
    pinMode(dm_row[i], OUTPUT);
    pinMode(dm_col[i], OUTPUT);
    digitalWrite(dm_col[i], 0);
    digitalWrite(dm_row[i], ROW_OFF);
  }
}

void dm_off_all() {
  for (int i = 0; i < 8; i++) {
    digitalWrite(dm_row[i], ROW_OFF);
    digitalWrite(dm_col[i], COL_OFF);
  }
}

void loop() {
  for (int i = 0; i < 8; i++) {
    dm_off_all();                    // All Off
    digitalWrite(dm_row[i], ROW_ON);    // Row On
    for (int j = 0; j < 8; j++) {       // Col On
      digitalWrite(dm_col[j], dm_heart[i][j]);
    }
```

```
    delay(2);
  }
}
```

위의 코드를 보면 먼저 눈에 확 들어오는 부분이 붉은색 부분 안에 파란색 하트 모양이 숫자 0 으로 찍혀 있는 것을 볼 수 있을 것입니다. 앞으로는 여러분이 이 부분에 어떤 모양을 숫자 0 으로 그리시면 그 모양이 도트 매트릭스에 켜질 것입니다.

< 문법 설명 : **2 차원 배열** >

위의 코드를 이해하기 위해서는 먼저 배열에 대해서 조금 더 공부를 해야 합니다. 우리가 전에 배웠던 것은 1 차원 배열이고 지금 설명드릴 것은 2 차원 배열입니다. 그렇게 어려운 것은 아니고 배열 속에 배열이라고 생각하시면 됩니다. 다음과 같이 정의 합니다.

```
변수_타입 변수_이름[원소그룹수(M)][원소수(N)] =
 { { 원소_1_1, 원소_1_2, ..., 원소_1_N },
  { 원소_2_1, 원소_2_2, ..., 원소_2_N },
     .
     .
     .
  { 원소_M_1, 원소_M_2, ..., 원소_M_N },
```

위의 정의에서 원소 그룹의 수는 M 이고 원소의 수는 N 입니다. 2 차원 배열은 1 차원 배열들이 여러 개 묶인 배열이라고 생각하시면 됩니다. 그리고 위의 코드 예제를 보시면 이해하시기 쉬울 것입니다. 2 차원 배열을 행렬이라고도 부릅니다. 위의 코드에서는 8x8 행렬을 변수 dm_heart[8][8] 로 선언하였습니다. 이 차원 배열의 각 원소를 사용하는 것은 1 차원 배열이 "변수[N]" 으로 하나의 인덱스를 사용하였다면 2 차원 배열은 "변수[M][N]" 으로 인덱스 2 개를 사용합니다. 이제 loop() 함수의 코드를 보겠습니다.

```
void loop() {
  for (int i = 0; i < 8; i++) {
    dm_off_all();                          // 1. All Off
    digitalWrite(dm_row[i], ROW_ON);    // 2. Row On
    for (int j = 0; j < 8; j++) {          // 3. Col On
      digitalWrite(dm_col[j], dm_heart[i][j]);
    }
    delay(2);
  }
}
```

위의 코드는 크게 3 부분으로 나뉩니다. 첫번째로는 dm_off_all() 함수를 이용하여 모든 LED 를 끕니다. 두번째는 가로줄을 컨트롤합니다. 가로줄은 0 번부터 7 번까지가 계속해서 반복적으로 한 줄씩 켜 집니다. 이것을 위해서 i 를 인덱스로 하는 첫번째 for 문을 사용하였습니다. 이 for 문에서 인덱스 i 는 0 에서 7 까

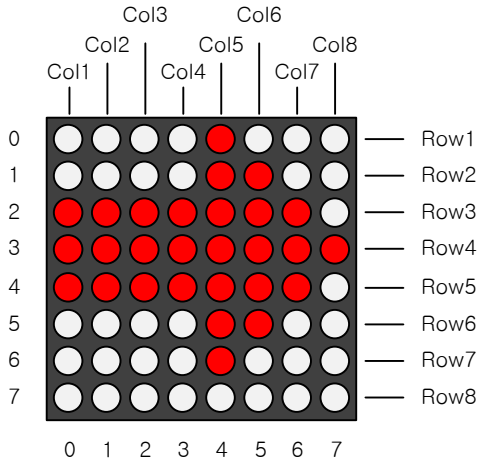
지 변합니다. 그리고 "digitalWrite(dm_row[i], ROW_ON)" 으로 코딩을 하여 해당 인덱스의 줄을 켜 주는 것입니다. 그렇게 선택된 가로줄은 두번째 j 를 인덱스로 하는 for 문에서 선택된 줄에서 어떤 LED 가 켜질지를 결정해 줍니다. 이것을 결정할 때 dm_hear 배열이 사용되는 것입니다. 그리고 이렇게 한 줄이 켜지는 시간은 delay(2) 로 하여 0.002 초간 켜져 있습니다. 이렇게 짧은 시간 동안만 켜게 되면 자연스럽게 도트매트릭스에 하트가 계속 켜 있는 것처럼 보입니다.

< 예제 코드 : **도트매트릭스로 하트 그리기** >

< 코드 위치 : Coding Kit Ex1 → **dotmat_heart >**

< 연습 문제 : **도트매트릭스로 화살표 그리기** >

이제 여러분은 도트매트릭스에 어떤 모양이든 그릴 수 있습니다. 그러면 다음과 같은 화살표 모양을 그리는 예제를 해 보도록 하겠습니다.



2 차원 배열의 코드만 바꾸면 됩니다. 매우 쉽죠.

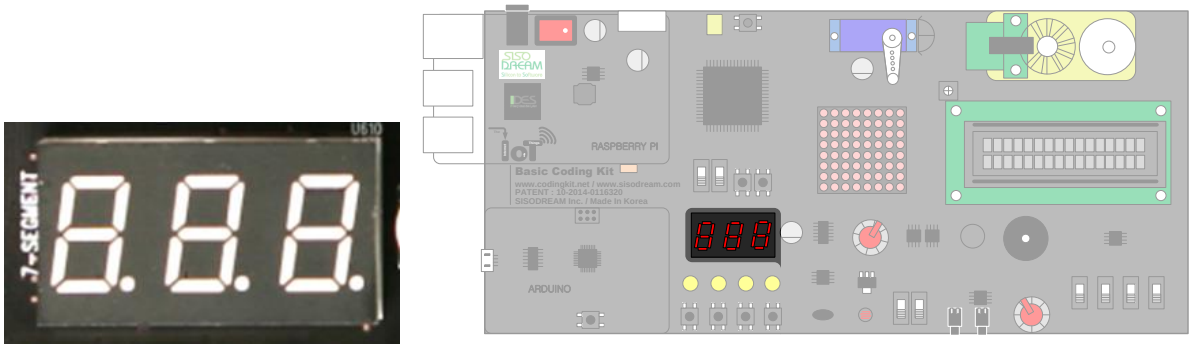
< 사진 : 도트매트릭스 화살표 >

< 코드 위치 : Coding Kit Ex1 → **dotmat_arrow >**

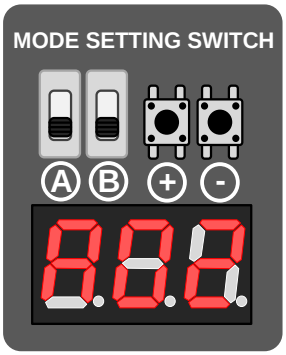
도트매트릭스에는 애니메이션을 표현하는 등의 재미있는 예제가 많습니다. 하지만 도트매트릭스만 하면 너무 지겨울 것 같아서 도트매트릭스 예제는 이만 넘어 가고, 추가적인 것은 코딩 사이트에서 만나도록 하겠습니다.

[세븐세그먼트에 숫자 표시 하기]

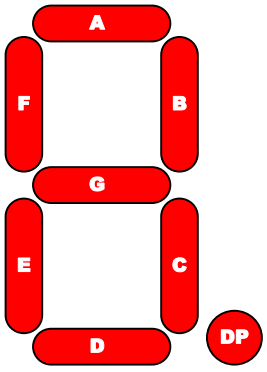
이번에는 세븐세그먼트(7-Segmnet)에 대해서 알아보도록 하겠습니다. 이 세븐세그먼트는 도트매트릭스와 비슷하게 LED 여러 개가 모여 있는 것입니다.



세븐세그먼트의 스위칭 ID 는 A02 입니다. 그리고 세븐세그먼트는 스위칭 ID 표시로도 사용되는 관계로 스위치와 버튼을 스위칭 ID A02 에 맞춘 다음 + 나 - 버튼을 오래 눌러 설정을 완료하면 세븐세그먼트는 꺼지거나 이전에 아두이노에 있던 코드가 수행이 될 것입니다. 그래서 조금 이상하게 동작할 수도 있습니다. 놀라지 마시고 그냥 코딩하고 업로드하여 사용하시면 됩니다.



세븐세그먼트는 이름처럼 숫자나 문자를 7개의 LED 로 표현합니다. 세븐세그먼트에서는 LED 보다는 세그먼트라는 말이 더 어울립니다. 각각의 세그먼트는 A 에서 G 까지 7 개의 문자로 이름을 붙입니다. 그리고 여기에 점을 하나 더 추가합니다. 이것은 소수점 이라는 뜻으로 DP(Decimal Point)로 표시합니다.



세븐세그먼트는 FND (Flexible Numeric Display) 라고도 합니다. 앞으로는 세븐세그먼트 대신 FND 라는 말을 사용하겠습니다.

FND 를 모두 켜는 예제를 해 보겠습니다.

```
#define FND_A    2
#define FND_B    3
#define FND_C    4
#define FND_D    5
#define FND_E    6
#define FND_F    7
#define FND_G    8
#define FND_DP   9
#define FND_SEL_1 10
#define FND_SEL_2 11
#define FND_SEL_3 12

#define FND_ON     0
#define FND_OFF    1
#define FND_SEL_ON  0
#define FND_SEL_OFF 1

int fnd[8] = {
  FND_A, FND_B, FND_C, FND_D,
  FND_E, FND_F, FND_G, FND_DP};

void setup() {
  for (int i = 0; i < 8; i++) {
    pinMode(fnd[i], OUTPUT);
  }

  pinMode(FND_SEL_1, OUTPUT);
  pinMode(FND_SEL_2, OUTPUT);
  pinMode(FND_SEL_3, OUTPUT);

  for (int i = 0; i < 8; i++) {
    digitalWrite(fnd[i], FND_ON);
  }

  digitalWrite(FND_SEL_1, FND_SEL_ON);
  digitalWrite(FND_SEL_2, FND_SEL_ON);
  digitalWrite(FND_SEL_3, FND_SEL_ON);
}

void loop() {
}
```

FND 의 세그먼트 A ~ G, DP 는 FND_A ~ FND_G, FND_DP 로 정의합니다. 그리고 코딩키트에는 FND 3 개가 있습니다. 그래서 위의 코드에서 FND_SEL_1, FND_SEL_2, FND_SEL_3 으로 3 개의 FND 를 선택하는

핀을 정의하였습니다. 이 핀들에 의해서 FND 각각을 켜거나 끕니다. FND_A ~ FND_G, FND_DP 를 켜거나 끄는 것을 FND_ON, FND_OFF 로 정의하였습니다. FND_SEL_1, 2, 3 신호를 켜거나 끄는 것은 FND_SEL_ON, FND_SEL_OFF 로 정의하였습니다. FND_A ~ FND_G, FND_DP 를 배열 변수 fnd 로 정의하였습니다. 이 배열을 이용하여 setup() 함수에서는 핀 모드를 설정하였고, 붉은색 코드와 같이 코딩하여 모든 FND 를 켜 주었습니다. 이 코드를 업로드하면 코딩키트에서 다음과 같이 켜 지는 것을 볼 수 있습니다.

< 예제 코드 : **세븐세그먼트 켜기** >

< 코드 위치 : Coding Kit Ex1 → **fnd** >

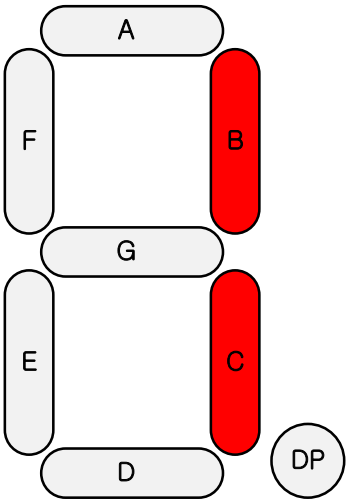
FND 에는 주로 숫자를 표시합니다. 그래서 숫자 1, 2, 3 을 3 개의 FND 에 켜는 예제를 해 보겠습니다. 다음과 같이 코딩을 합니다. 위의 예제에서 정의한 부분은 뺐습니다.

```
#define FND_ON_DELAY 2

int fnd[8] = {
  FND_A, FND_B, FND_C, FND_D,
  FND_E, FND_F, FND_G, FND_DP};

int fnd_1[8] = {1, 0, 0, 1,   1, 1, 1, 1};
int fnd_2[8] = {0, 0, 1, 0,   0, 1, 0, 1};
int fnd_3[8] = {0, 0, 0, 0,   1, 1, 0, 1};

void fnd_off() {
  digitalWrite(FND_SEL_1, FND_SEL_OFF);
  digitalWrite(FND_SEL_2, FND_SEL_OFF);
  digitalWrite(FND_SEL_3, FND_SEL_OFF);
}

void setup() {
  for (int i = 0; i < 8; i++) {
    pinMode(fnd[i], OUTPUT);
  }

  pinMode(FND_SEL_1, OUTPUT);
  pinMode(FND_SEL_2, OUTPUT);
  pinMode(FND_SEL_3, OUTPUT);
```

```
  fnd_off();
}

void loop() {
  // 첫번째 FND에 숫자 1 쓰기
  fnd_off();
  for (int i = 0; i < 8; i++) {
    digitalWrite(fnd[i], fnd_1[i]);
  }
  digitalWrite(FND_SEL_1, FND_SEL_ON);
  delay(FND_ON_DELAY);

  // 두번째 FND에 숫자 2 쓰기
  fnd_off();
  for (int i = 0; i < 8; i++) {
    digitalWrite(fnd[i], fnd_2[i]);
  }
  digitalWrite(FND_SEL_2, FND_SEL_ON);
  delay(FND_ON_DELAY);

  // 세번째 FND에 숫자 3 쓰기
  fnd_off();
  for (int i = 0; i < 8; i++) {
    digitalWrite(fnd[i], fnd_3[i]);
  }
  digitalWrite(FND_SEL_3, FND_SEL_ON);
  delay(FND_ON_DELAY);
}
```

먼저 fnd_off() 함수를 정의하였습니다. fnd_off() 함수를 보면 3 개의 FND 를 모두 끄는 코드가 있습니다. setup() 함수에서 초기에 FND 3 개를 모두 끄기 위해 fnd_off() 함수를 불러 씁니다.

이 3 개의 FND 는 FND_A ~ FND_G, FND_DP 신호을 공유하기 때문에 도트매트릭스와 비슷하게 각각 한 개씩 켜 주어야 합니다. 어느 순간이든 FND 3 개중 하나만을 선택하여 그 FND 를 컨트롤 합니다. loop() 함수에서 FND 3개를 각각 컨트롤하는데, 첫번째 FND 에는 숫자 1 을 써 주어야 합니다. 숫자 1 에 대한 각각의 세그먼트를 켜기 위한 정의는 배열 변수 fnd_1 에 정의해 두었습니다. 이 변수를 보면 세그먼트 A ~ G 중에 B, C 만을 켜고 나머지는 다 껐습니다. 켜기 위해서는 0 을 써 주고 끌때는 1 을 써 줍니다. 숫자 1 은 다음 그림과 같습니다.



위 그림을 배열 변수로는 다음과 같이 정의하였습니다.

```
int fnd_1[8] = {1, 0, 0, 1,   1, 1, 1, 1};
```

loop() 함수에서 첫번째 FND 에 숫자 1 을 써 주는 코드는 다음과 같습니다.

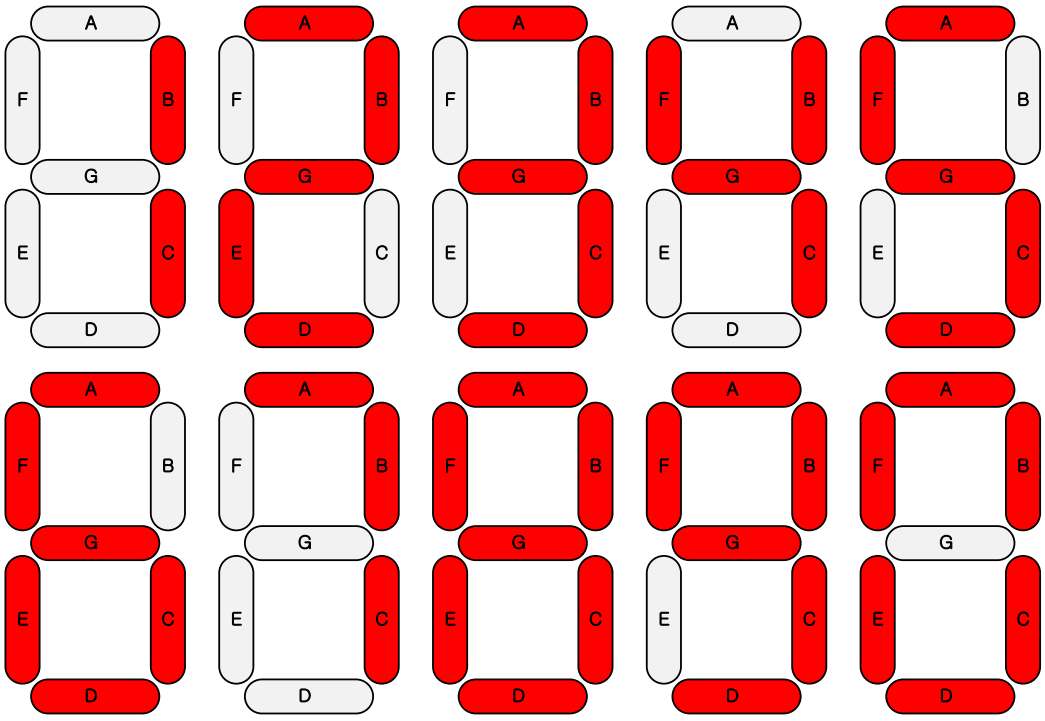
```
fnd_off();
for (int i = 0; i < 8; i++) {
  digitalWrite(fnd[i], fnd_1[i]);
}
digitalWrite(FND_SEL_1, FND_SEL_ON);
delay(FND_ON_DELAY);
```

일단 fnd_off() 함수를 이용하여 FND 를 모두 꺼 줍니다. 그리고 for 문을 이용하여 숫자 1 을 정의한 배열 변수 fnd_1 을 FND_A ~ FND_G, FND_DP 에 할당해 줍니다. 다음으로 첫번째 FND 만 켜 주기 위해서 "digitalWrite(FND_SEL_1, FND_SEL_ON)" 합니다. 그리고 약간의 시간을 주어 그 시간 동안은 첫번째 FND 만 켜 있도록 합니다. 두번째, 세번째 FND 를 켜는 것도 같은 방법으로 코딩합니다. 도트매트릭스와 같이 각각의 FND 를 아주 짧은 시간 동안만 켜면서 빠르게 스위칭하면 전체가 다 켜 있는 것처럼 보입니다. 이제 코드를 업로드 하여 코딩키트에서 확인합니다.

< 예제 코드 : **세븐세그먼트에 숫자 1, 2, 3 표시하기** >

< 코드 위치 : Coding Kit Ex1 → **fnd_123** >

이번에는 가변 저항 값을 FND 에 표시하는 예제를 해 보겠습니다. 가변 저항 값을 FND 에 표시 하려면 0 ~ 9 까지 숫자를 FND 에 표시해야 합니다. 숫자는 다음 그림과 같이 표시합니다.

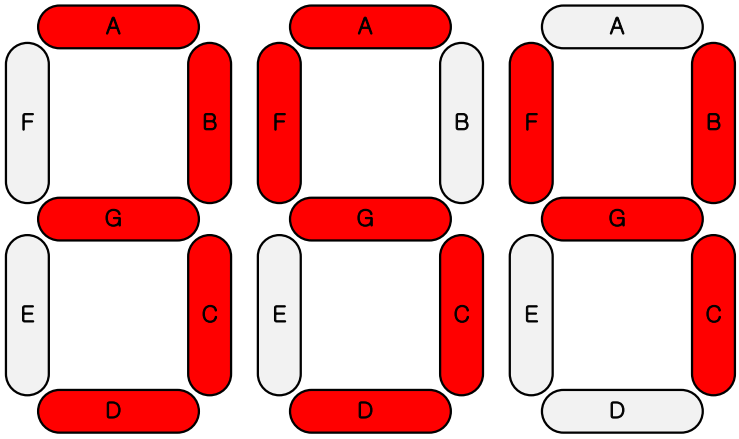


이 숫자들을 표시하기 위해서 배열 변수 fnd_0 ~ fnd_9 를 다음과 같이 정의 합니다.

```
int fnd_0[8] = {0, 0, 0, 0,   0, 0, 1, 1};
int fnd_1[8] = {1, 0, 0, 1,   1, 1, 1, 1};
int fnd_2[8] = {0, 0, 1, 0,   0, 1, 0, 1};
int fnd_3[8] = {0, 0, 0, 0,   1, 1, 0, 1};
int fnd_4[8] = {1, 0, 0, 1,   1, 0, 0, 1};
int fnd_5[8] = {0, 1, 0, 0,   1, 0, 0, 1};
int fnd_6[8] = {0, 1, 0, 0,   0, 0, 0, 1};
int fnd_7[8] = {0, 0, 0, 1,   1, 1, 1, 1};
int fnd_8[8] = {0, 0, 0, 0,   0, 0, 0, 1};
int fnd_9[8] = {0, 0, 0, 0,   1, 0, 0, 1};
```

가변 저항의 값은 0 ~ 1023 까지 변하는데, FND 는 0 ~ 999 까지 표현할 수 있습니다. 그래서 가변 저항 값은 map() 함수를 이용하여 0 ~ 999 로 바꾸어 줍니다.

FND 3 개는 100 자리, 10 자리, 1 자리를 표시합니다. 예를 들어 354 라는 숫자는 다음과 같이 표시가 되겠죠.



그럼 이 숫자를 100, 10, 1 자리로 나누어야 합니다. 이것을 위해서는 다음과 같은 코드가 필요합니다.

```
int fnd1_num = vr_map / 100;          // 100의 자리
int fnd2_num = (vr_map % 100) / 10;  // 10의 자리
int fnd3_num = (vr_map % 10);         // 1의 자리
```

여기서 vr_map 이라는 값이 354 라면 fnd1_num 에는 354 를 100 으로 나누어 나머지는 버리고 몫만 취합니다. 그러면 3 이 되겠지요. fnd2_num 을 구하는 두번째줄 코드에는 "%" 기호가 있습니다. 이 기호의 뜻은 왼쪽에 있는 수를 오른쪽에 있는 수로 나누었을 때 나머지를 쓰겠다는 의미입니다. 그래서 354 를 100 으로 나눈 나머지는 54 입니다. 이것을 10 으로 나눈 몫은 5 가 되겠지요. 이렇게 해서 10 의 자리 숫자를 구했습니다. 나머지의 1 의 자리 숫자는 10 으로 나눈 나머지를 취하면 되겠습니다. 이렇게 해서 100, 10, 1 의 자리를 구하여 각각의 변수에 저장하였습니다.

위에서 100, 10, 1 자리의 숫자도 구하고, 0 ~ 9 까지 숫자를 FND 에 어떻게 표시할지도 배열 변수로 정의하였습니다. 그러면 loop() 함수에서는 다음과 같이 하여 FND 에 숫자 값을 출력합니다.

```
sel_fnd_num(fnd1_num);
fnd_off();
for (int i = 0; i < 8; i++) {
  digitalWrite(fnd[i], fnd_num[i]);
}
digitalWrite(FND_SEL_1, FND_SEL_ON);
delay(FND_ON_DELAY);
```

여기서 함수 sel_fnd_num() 은 어떤 숫자를 입력으로 받아 그 숫자에 해당하는 FND 세그먼트 배열 변수를 출력해 주는 함수 입니다. 코드는 다음과 같습니다.

```
void sel_fnd_num(int num) {
  switch (num) {
    case 0 : for (int i = 0; i < 8; i++) fnd_num[i] = fnd_0[i]; break;
    case 1 : for (int i = 0; i < 8; i++) fnd_num[i] = fnd_1[i]; break;
    case 2 : for (int i = 0; i < 8; i++) fnd_num[i] = fnd_2[i]; break;
```

```
    case 3 : for (int i = 0; i < 8; i++) fnd_num[i] = fnd_3[i]; break;
    case 4 : for (int i = 0; i < 8; i++) fnd_num[i] = fnd_4[i]; break;
    case 5 : for (int i = 0; i < 8; i++) fnd_num[i] = fnd_5[i]; break;
    case 6 : for (int i = 0; i < 8; i++) fnd_num[i] = fnd_6[i]; break;
    case 7 : for (int i = 0; i < 8; i++) fnd_num[i] = fnd_7[i]; break;
    case 8 : for (int i = 0; i < 8; i++) fnd_num[i] = fnd_8[i]; break;
    case 9 : for (int i = 0; i < 8; i++) fnd_num[i] = fnd_9[i]; break;
  }
}
```

< 문번 설명 : **switch 문** >

위의 코드에서 switch 문을 활용하였습니다. switch 문은 어떤 변수에 따라서 수행하는 문장이 틀려집니다. 정의식은 다음과 같습니다.

```
switch (변수) {
  case 변수값_1 :
    문장_1;
  case 변수값_2 :
    문장_2;
 case 변수값_3 :
    문장_3;
  .
  .
  .
  case 변수값_N :
     문장_N;
  default :
    문장_default;
}
```

switch() 문은 괄호 안 변수의 값에 따라서 해당 case 이후의 모든 case 문장이 수행됩니다. 예를 들어 변수가 변수값_2 와 같으면 "case 변수값_2" 다음 문장이 수행되고, "case 변수값_3" 문장이 수행되고 그 이후에 나오는 모든 "case 변수값" 문장이 수행되고 최종적으로 "defaut : "다음의 문장도 수행됩니다. default 는 변수가 모든 "case 변수값" 과 일치하지 않을 때 수행됩니다. default 문장은 생략 가능합니다.

break 를 사용하면 중간에서 수행되는 case 를 멈출 수 있습니다. 예를 들어 다음과 같은 경우을 보겠습니다.

```
switch (변수) {
 case 변수값_1 :
   문장_1;
```

```
    case 변수값_2 :
      문장_2;
    case 변수값_3 :
      문장_3;
      break;
  case 변수값_4 :
      문장_4;
  default :
      문장_default;
  }
```

위의 코드에서 변수가 변수값_2 와 같다면 문장_2 와 문장_3 만을 수행하고 그 이후의 문장_4 문장_default 는 수행하지 않습니다. 그것은 문장_3 다음의 break 때문입니다.

위의 코드 sel_fnd_num() 함수의 switch 문에서는 함수의 입력 num 이 switch 문의 변수로 사용되었습니다. 이 num 이 0 이면 fnd_num 이라는 배열 변수에 fnd_0 배열 변수의 값을 할당합니다.

```
switch (num) {
  case 0 : for (int i = 0; i < 8; i++) fnd_num[i] = fnd_0[i]; break;
```

그리고 바로 break 를 이용하여 다음 문장이 수행되지 못하게 하였습니다. 이와 같이 1 ~ 9 도 처리하였습니다. 이렇게 하면 어떤 숫자도 FND 로 표시할 수 있겠지요. 그리고 case 문 한줄에 해당 case 에서 실행된는 모든 코드를 다 썼는데요. 이렇게 하면 비슷한 여러줄을 코딩할 때 보기도 좋고 고치기도 좋아 예외적으로 이렇게 썼습니다.

가변저항이 연결된 핀은 다음과 같이 정의 됩니다.

```
#define VAR_RES_1 A3
```

이상의 코드들을 활용하여 가변저항 값을 FND 로 표시하는 코드를 직접 구성해 보십시요. 재밌을 것입니다. 그리고 방금 소개된 코드는 매우 좋은 코드입니다. 이렇게 코드를 깔끔하고 구조적으로 안정적으로 코딩해 두면 나중에 코드를 고치기도 좋고 보기도 매우 좋습니다. 코딩키트의 예제 중에는 이렇게 멋진 코드들이 많이 있습니다. 여러분도 이런 코드들을 많이 보고 익혀서 여러분의 실력으로 만든다면 잘 나가는 컴퓨터 프로그래머가 되는 것은 금방입니다. 더 많은 코드들은 코딩 사이트에서 만나 보세요.

< 예제 코드 : **가변 저항 값을 FND 에 표시하기** >

< 코드 위치 : Coding Kit Ex1 → **fnd_vr** >

< 문법 설명 : **산술 연산자** >

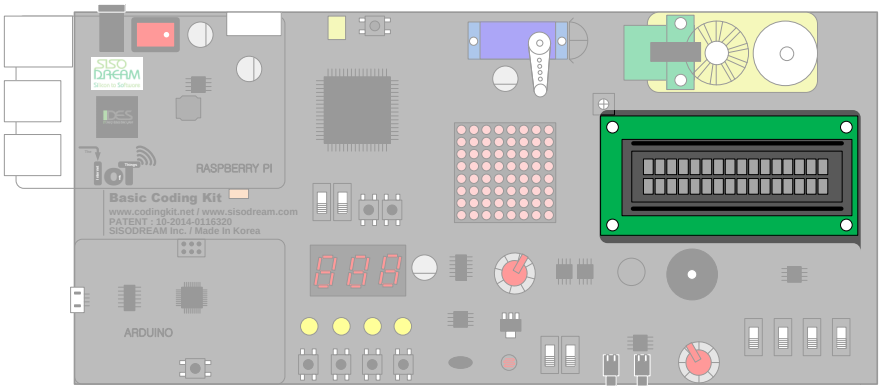
여기서 잠깐 산술 연산자에 대해서 배우고 가겠습니다. 산술 연산자란 더하기, 빼기, 곱하기, 나누기 등을 말합니다. 산술 연사자에는 다음과 같은 연산자가 있습니다.

기호	의미
+	더하기
-	빼기
*	곱하기
/	나누기
%	나머지
++	1 증가
--	1 감소

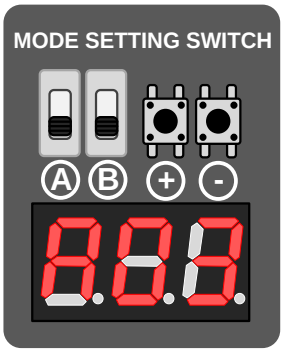
산술 연사자는 위의 표와 같고, 굳이 길게 설명 들이지 않아도 잘 아실 것입니다. 그리고 몇 가지는 이미 벌써 다 설명드렸습니다.

[Character LCD 에 문자 쓰기]

캐릭터 LCD(Character Liquid Crystal Display)라는 것은 문자를 출력하기 알맞게 제작된 LCD 입니다. 코딩키트의 캐릭터 LCD 는 16 개의 문자를 2 줄에 출력할 수 있는 캐릭터 LCD 가 장착되어 있습니다. 보통 16x2 캐릭터 LCD 라고 합니다.



캐릭터 LCD 를 쓰기 위해서는 스위칭 ID 를 A03 으로 맞추어야 합니다.



캐릭터 LCD 는 지금처럼 컬러 LCD 등의 좋은 LCD 들이 개발되기 오래 전부터 많이 사용되어 온 디바이스입니다. 사용하기도 쉽고 가격도 저렴하여 아직도 산업계 전반에 많이 사용되고 있습니다. 아두이노에서는 이 캐릭터 LCD 를 더욱 쉽게 사용하기 위해서 라이브러리를 제공하고 있습니다. 그래서 캐릭터 LCD 를 사용하기 위해서는 다음과 같은 코드를 추가해야 합니다.

```
#include <LiquidCrystal.h>
```

LCD 관련 핀들은 다음과 같은 핀들을 정의합니다.

```
#define LCD_EN    9   // Enable
#define LCD_RS    10  // Register Select
#define LCD_D0    11
#define LCD_D1    A5
#define LCD_D2    12
#define LCD_D3    A4

#define LCD_B_LIGHT 13
```

이러한 핀들에 대한 구체적인 내용은 코딩 사이트의 하드웨어 관련 게시물을 참고해 주세요. 일단은 다음과 같은 용도로 알고 넘어 가겠습니다. 우리에게는 코딩이 중요하니깐요!

핀 이름	용도
LCD_EN	LCD Eanble
LCD_RS	LCD 의 설정 신호 표시
LCD_D0 ~ 3	LCD 데이터 0 ~ 3
LCD_B_LIGHT	LCD Back Light

이제 LCD 에 "Coding Kit" 라는 문자를 출력하는 코드를 작성해 보겠습니다. 위에 있는 라이브러리 포함과 핀 정의 코드 이외의 코드는 다음과 같습니다.

```
LiquidCrystal ck_lcd(LCD_RS, LCD_EN, LCD_D0, LCD_D1, LCD_D2, LCD_D3);

void setup() {
  pinMode(LCD_B_LIGHT, OUTPUT);
  digitalWrite(LCD_B_LIGHT, 1); // LCD Back Light ON

  ck_lcd.begin(16, 2);
  ck_lcd.print("Coding Kit!");
}

void loop() {
}
```

먼저 "LiquidCrytal ck_lcd()" 라고 하여 LCD 를 정의합니다. 정의할 때 LCD 에 연결되는 핀들을 입력해 줍니다. 이렇게 정의된 LCD 가 setup() 함수에서 ck_lcd.begin() 함수를 통해서 초기화 됩니다. 초기화는 16x2 LCD 를 초기한다는 의미로 ck_lcd.begin(16, 2) 로 해 줍니다. 그리고 ck_lcd.print() 함수를 활용하여 LCD 에 문자를 표시합니다. 이제 업로드 하여 코딩키트의 캐릭터 LCD 에 "Coding Kit" 라는 문자가 표시되는 것을 확인합니다.

< 예제 코드 : **캐릭터 LCD 에 문자 출력하기** >

< 코드 위치 : Coding Kit Ex1 → **charlcd >**

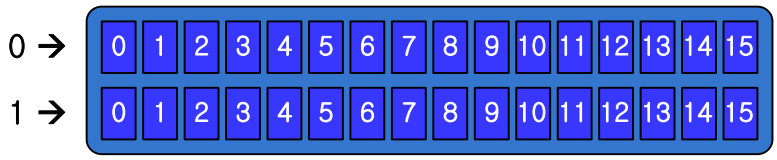
이번에는 캐릭터 LCD 에 온도 센서 값을 표시해 보도록 하겠습니다. 다음 사진과 같이 첫번째 줄에는 온도 센서의 값을 두번째 줄에는 온도가 올라갈수록 '>' 표시가 늘어나도록 하겠습니다.

코드는 다음과 같습니다.

```
#define SENSOR_TEMP A2

LiquidCrystal ck_lcd(LCD_RS, LCD_EN, LCD_D0, LCD_D1, LCD_D2, LCD_D3);

void setup() {
  pinMode(LCD_B_LIGHT, OUTPUT);
  digitalWrite(LCD_B_LIGHT, 1); // LCD Back Light ON

  ck_lcd.begin(16, 2);   // COL x ROW
}

#define TEMP_MIN    530

void loop() {
  int temp = analogRead(SENSOR_TEMP);
  ck_lcd.clear();

  // The 1st Line
  ck_lcd.setCursor(0, 0);
  ck_lcd.print("TEMP : ");
  ck_lcd.setCursor(7, 0);
  ck_lcd.print(temp);

  // The 2nd Line
```

```
  int num_char = (temp - TEMP_MIN)/5;
  if (num_char >= 15)
    num_char = 15;

  for (int i = 0; i < num_char; i++) {
    ck_lcd.setCursor(i, 1);
    ck_lcd.print(">");
  }

  delay(500);
}
```

위의 코드에서 온도 센서의 최소 값은 530 으로 정의 하였습니다. 이 값은 현재 코딩키트가 있는 곳의 온도에 따라서 달라질 수 있으니 여러분께서는 코드를 실행하시기 전에 적당히 조정하십시요. loop() 함수를 보면 ck_lcd.clear() 라는 함수가 있습니다. 이 함수는 LCD 의 모든 값을 지우는 역할을 합니다. 그 다음으로 setCursor() 하는 함수가 있습니다. 이 함수는 현재 LCD 에 문자가 출력될 위치를 잡아 줍니다. setCursor() 의 첫번째 인자는 세로 줄의 인덱스이고 두번재 인자는 가로 줄의 인덱스 입니다. LCD 의 가로 세로 인덱스는 다음 그림과 같습니다.



초기에는 커서의 위치를 (0, 0) 으로 잡아 줍니다. 먼저 "TEMP : " 라는 문자를 출력하고 커서를 (7, 0) 으로 옮깁니다. 옮겨진 위치에 온도 센서 값을 표시합니다.

두번째 줄은 현재 온도에서 최소 온도 값을 뺍니다. 이것을 5 로 나누면 ">" 를 표시해야 하는 수를 구할 수 있습니다. 이 값은 변수 num_char 에 저장됩니다. for 문을 이용하여 이 값만큼 ">" 를 표시합니다. 이 때 커서를 한 칸씩 이동시키면서 표시해 줍니다.

"if (num_char >= 15) num_char = 15;" 의 코드는 캐릭터 LCD 의 세로 인덱스의 최대가 15 이기 때문에 변수 num_char 값이 15 을 넘어가지 않도록한 코드입니다.

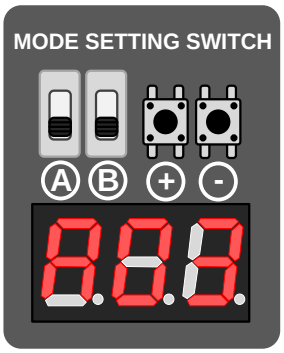
이제 코드를 업로드하여 코딩키트에서 테스트해 보겠습니다. 온도 센서를 엄지와 검지로 꼭 쥐고 있으면 온도가 서서히 올라가는 것을 볼 수 있을 것입니다. 두번째 줄의 ">" 표시도 계속 늘어납니다.

< 예제 코드 : **캐릭터 LCD 에 온도와 온도 상승 그래프 표시하기** >

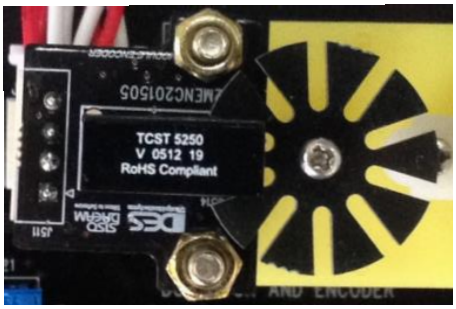
< 코드 위치 : Coding Kit Ex1 → **charlcd_temp** >

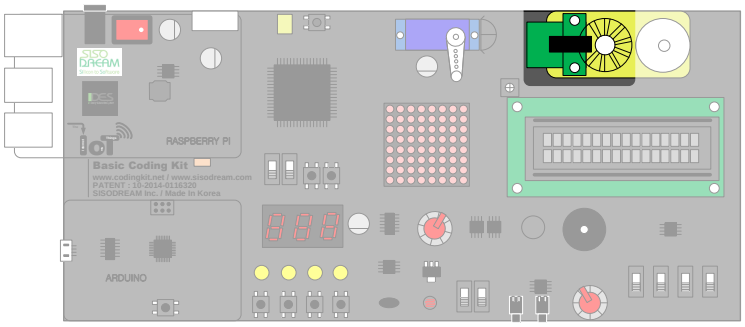
[DC 모터 인코더와 인터럽트 이해하기]

이번장의 스위칭 ID 는 A03 입니다.



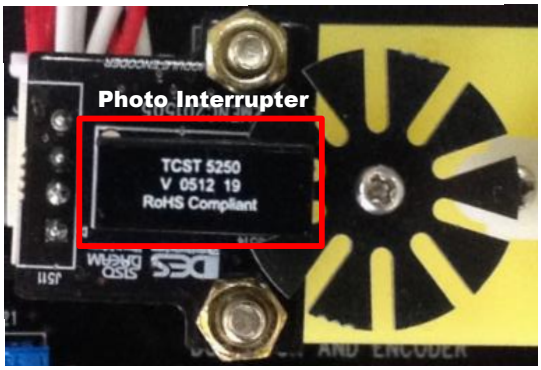
코딩키트의 DC 모터에는 인코더 (Encoder) 라는 것이 붙어 있습니다. 이것이 무엇을 하는 것일까요? 이것은 DC 모터가 얼마나 회전하고 있는지를 체크하는 것입니다.



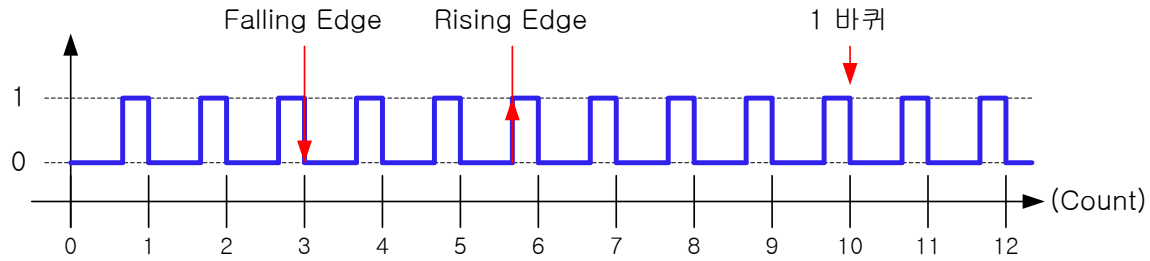


이 인코더는 크게 휠 (Wheel) 과 포토 인터럽터 (Photo Interrupter) 라는 센서로 나누어져 있는데, 그 중 휠은 다음 그림과 같이 10 개의 날개를 가지고 있습니다.

이 날개가 다음 그림과 같은 포토 인터럽터 센서의 안쪽을 지나갑니다.



이 포토 인터럽터는 여러분이 전에 배웠던 적외선 센서와 같은 원리입니다. 그런데, 전에 배웠던 적외선 센서가 발광부와 수광부가 같은 방향으로 되어 있어 반사파를 수신하는 대신 이 **포토 인터럽터는 발광부와 수광부가 마주 보고 있어 그 사이를 어떤 물체가 지나가게 되면 수신이 안 되고 물체가 없으면 수신이 되는 것입니다.** 그래서 수신 측에는 다음과 같은 신호가 전달 됩니다.



위의 그림에서 신호가 0 으로 떨어진 부분이 센서 사이에 휠의 날개가 들어간 상태이고 1 로 올라간 부분이 센서 사이에 휠의 날개가 없는 상태인 것입니다. 이 휠의 날개가 10 개이기 때문에 센서 수광부에서 신호 값이 0 으로 내려가는 부분을 10 번 카운트하면 DC 모터가 한 바퀴 돌았다는 것을 알 수 있습니다. 여기서 1 로 올라가는 부분을 Rising Edge 라고 합니다. 반대로 0 으로 떨어지는 부분을 Falling Edge 라고 합니다. Rising Edge 든 Falling Edge 든 10 번만 카운트 하면 DC 모터가 한 바퀴 돈 것을 알 수 있습니다.

그렇다면 DC 모터가 1 분에 몇 바퀴를 돌까요? 이런 것을 분당 회전수, 영어로는 RPM(Revolutions Per Minute)이라고 하지요. 그럼 바로 DC 모터의 RPM 을 측정하는 예제를 해 보겠습니다. DC 모터와 인코더는 다음과 같이 정의합니다. 여기에 DC 모터의 속도를 조정할 수 있는 가변저항도 정의 하였습니다.

```
#define DCMOT_EN          4
#define DCMOT_FWD         5
#define DCMOT_BWD         6
#define DCMOTENC_EN       7
#define DCMOTENC_DATA 2
#define VAR_RES_1         A3
```

이중 DC 모터 인코더 관련 핀들만 설명하겠습니다. DCMOTENC_EN 신호는 DC 모터 인코더를 켜 주는 Enable 신호입니다. DCMOTENC_DATA 는 포토 인터럽터 센서의 수광부에서 데이터를 받는 신호입니다. 이 신호는 위의 그림의 파형과 같은 신호가 입력됩니다.

먼저 전역 변수 몇 개를 정의하고 초기화 하였습니다. 이 변수들에 대해서는 코드에서 사용될 때 설명하겠습니다.

```
int get_rpm = 0;
int cnt = 0;
unsigned long now = 0;
unsigned long previous = 0;
unsigned long interval = 1;
```

다음과 같이 setup() 함수를 코딩하였습니다.

```
void setup() {
  pinMode(DCMOT_EN, OUTPUT);
  pinMode(DCMOT_FWD, OUTPUT);
  pinMode(DCMOT_BWD, OUTPUT);
  pinMode(DCMOTENC_EN, OUTPUT);
  pinMode(DCMOTENC_DATA, INPUT);

  digitalWrite(DCMOT_EN, 1);
  digitalWrite(DCMOT_FWD, 0);
  digitalWrite(DCMOT_BWD, 0);
  digitalWrite(DCMOTENC_EN, 1);

  Serial.begin(9600);

  attachInterrupt(0, enc_cnt, FALLING);

  previous = millis();
}
```

DCMOTENC_EN 신호는 1 로 하여 DC 모터 인코더를 켜 주었구요. DCMOTENC_DATA 신호는 입력으로 하여 신호를 받을 준비를 했습니다. 시리얼 모니터를 설정하였구요. 그리고 마지막에 못 보던 함수가 있습니다. attachInterrupt() 라는 함수 입니다. 여기서 인터럽트 (Interrupt) 라는 용어를 알아 보고 가겠습니다. 인터럽트라는 것은 단어 뜻 그대로 끼어드는 것입니다. 코딩된 모든 프로그램의 코드는 기본적으로 첫번째 줄부터 순서대로 수행이 됩니다. 그런데 이 인터럽트라는 것은 코드가 순서대로 수행 되고 있는 중간에 갑자기 끼어듭니다. 코드 실행 중에 예기치 않은 어떤 일이 생기면 그것을 처리하기 위해서 중간에 끼어들기를 하는 것입니다. 이걸 프로그램쪽 언어로 말하면 이벤트 (Event)가 발생하면 인터럽트, 즉 끼어들기를 한다고 합니다. 이런 끼어들기, 인터럽트의 예를 들어 보겠습니다. 여러분이 스마트폰으로 재미있는 게임을 하고 있었습니다. 그런데 갑자기 전화가 와서 게임은 중단되고 벨 소리가 울립니다. 여기서 전화가 오는 것이 예기치 않은 어떤 일이 발생된 것이고 스마트폰 코드는 이렇게 전화가 온 것을 인터럽트로 처리합니다. 즉, 게임 코드가 실행되는 도중에 전화가 오면 전화가 왔을 때 벨소리 등을 울려야 하는 전화 수신 처리 코드를 실행하는 것입니다. 여기서 전화가 온 것이 이벤트가 발생한 것이고, 게임 코드 중간에 벨소리 코드를 실행하는 것이 인터럽트입니다. 이렇게 평상시 순조롭게 코드가 수행되고 있던 중간에 갑자기 급한 일이 생길 때 그것을 먼저 처리해 주기 위해서 인터럽트 같은 것이 만들어진 것입니다.

인터럽트에는 어떤 이벤트가 발생했을 때 처리해 줄 코드가 필요합니다. 여기서 처리 코드는 일반적으로 함수가 쓰입니다. 이 함수를 인터럽트 서비스 루틴 (ISR : Interrupt Service Routine) 이라고 합니다. 이 인터럽트 서비스 루틴을 앞으로는 ISR 이라고 하겠습니다. 위의 스마트폰 예에서는 이벤트는 전화가 온 것이겠죠. 그리고 ISR 은 벨을 울리거나 하여 사용자에게 전화가 왔음을 알리는 함수 코드가 될 것입니다.

이제 우리의 예제 코드로 돌아가서 attachInterrupt() 함수를 계속 보겠습니다. 이 함수는 이름 그대로 여러분의 코드에 인터럽트를 붙여 주는 것입니다. 이 함수는 다음과 같이 정의합니다.

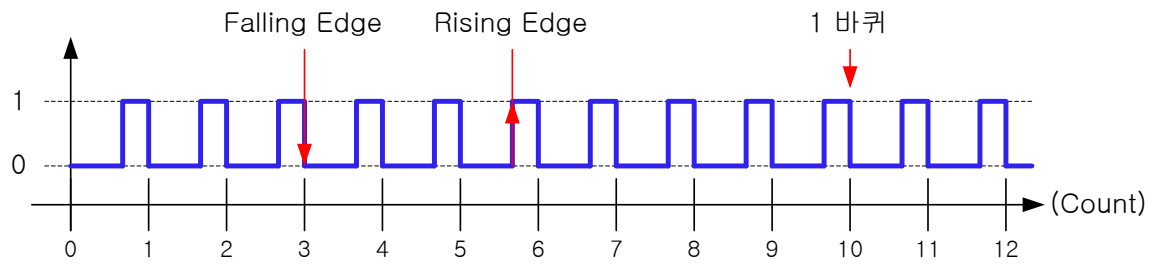
attachInterrupt(인터럽트_이벤트_핀_ID, ISR, 이벤트_모드)

첫번째 인자는 인터럽트_이벤트_핀_ID 입니다. 이것은 이벤트를 받을 핀을 의미하는 것입니다. 즉, 핀으로 이벤트가 발행했다는 통지를 받는 것입니다. 예를 들면 이 핀의 값이 계속해서 0 이었다가 갑자기 1 로 바뀝니다. 그러면 그것을 이벤트로 여기고 인터럽트 서비스 루틴을 실행합니다.

그런데 핀 번호가 아니고 핀 ID 입니다. 그것은 아두이노가 여러 제품이 있는데, 제품마다 조금씩 인터럽트로 사용되는 핀이 달라서 그럽니다. 그래서 핀 번호가 아닌 핀 ID 를 사용합니다. 코딩키트의 아두이노에서 인터럽트로 쓰일 수 있는 핀은 2 번과 3 번입니다. 즉, 2번과 3 번핀을 제외하고는 인터럽트로 사용할 수 없습니다. 2 번 핀의 ID 는 0 번이고, 3 번 핀의 ID 는 1 번입니다. 그래서 여러분의 인터럽트 이벤트 핀이 2 번에 연결되어 있으면 attachInterrupt() 함수의 첫번째 인자를 0 으로 하고, 3 번에 연결되어 있으면 1 로 합니다. attachInterrupt() 함수의 두번째 인자는 ISR 입니다. 즉, 이 인터럽트를 처리해 줄 함수 입니다. 마지막 인자는 이벤트의 모드입니다. 모드는 다음과 같습니다.

이벤트 모드	설명
LOW	값이 0 이 유지되는 동안 계속해서 이벤트 발생
CHANGE	값이 바뀔 때만 이벤트 발생
RISING	값이 0 에서 1 로 바뀔 때만 이벤트 발생
FALLING	값이 1 에서 0 으로 바뀔 때만 이벤트 발생

모드 중 RISING 과 FALLING 은 다음 그림에서 표시한 순간에 이벤트가 발생되는 것입니다.



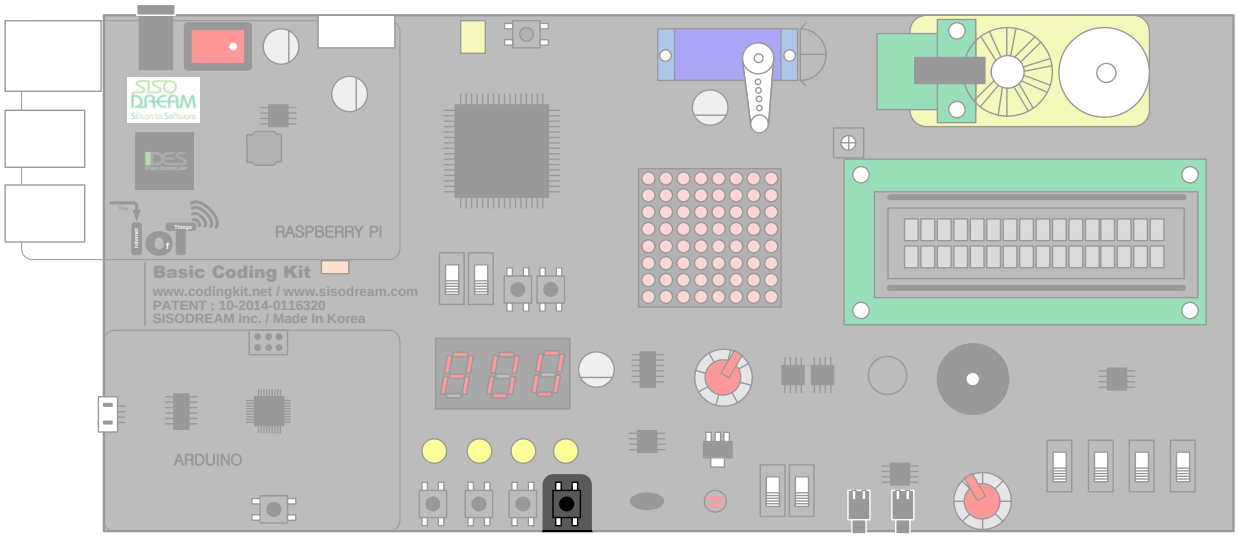
그리고 LOW 는 인터럽트 핀의 값이 0 인 동안에는 어떤 시간 간격으로 계속해서 인터럽트가 발생하는 것입니다. CHANG 는 값이 바뀔 때, 즉, 0 에서 1 로, 1 에서 0 으로 바뀔 때 인터럽트가 발생합니다. Falling

Edge, Rising Edge 모두에서 인터럽트가 발생하는 것이라고 생각하면 됩니다.

여기서 잠깐 위의 코드를 계속 설명하기 전에 인터럽트를 이용한 간단한 예제를 하나 해 보고 가겠습니다. 그러면 인터럽트에 대해서 훨씬 더 쉽게 이해할 수 있을 것입니다. 다음은 DC 모터가 정방향으로 3 초, 역방향으로 3 초씩 도는 것을 계속해서 반복하는 코드입니다. 여기에 버튼이 눌리는 인터럽트가 들어오면 DC 모터가 도는 것을 멈춥니다.

```
#define DCMOT_EN      4
#define DCMOT_FWD     5
#define DCMOT_BWD     6

#define INTR_BUT      3

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

  digitalWrite(DCMOT_EN, 1);
  digitalWrite(DCMOT_FWD, 0);
  digitalWrite(DCMOT_BWD, 0);

  attachInterrupt(1, intr_but, LOW);
}

void intr_but() {
  digitalWrite(DCMOT_FWD, 0);
  digitalWrite(DCMOT_BWD, 0);
}

void loop() {
  digitalWrite(DCMOT_FWD, 1);
  digitalWrite(DCMOT_BWD, 0);
  delay(3000);
  digitalWrite(DCMOT_FWD, 0);
  digitalWrite(DCMOT_BWD, 1);
  delay(3000);
}
```

위의 코드에서 attachInterrupt() 함수를 사용하여 인터럽트 ID 1 번을 사용하겠다고 선언합니다. 인터럽트 ID 1 번이면 아두이노의 3 번 핀을 사용하는 것인데 3 번 핀에는 다음 위치의 버튼이 연결되어 있습니다.



그래서 이 버튼을 누르면 인터럽트가 걸리는 것입니다. attachInterrupt() 함수의 두번째 인자는 이벤트가 발생했을 때 어떤 함수로 처리할 것인가를 알려주는 것입니다. 당연히 여기에는 DC 모터를 멈추는 코드가 들어 있는 함수가 있겠죠. 그래서 intr_but 이라는 함수 이름을 인자로 주었고 intr_but() 함수의 내용은 다음과 같이 DC 모터를 멈추는 코드가 있습니다.

```
void intr_but() {
  digitalWrite(DCMOT_FWD, 0);
  digitalWrite(DCMOT_BWD, 0);
}
```

그리고 attachInterrupt() 함수의 마지막 인자는 이벤트가 어떻게 발생했을 때 인터럽트를 할까? 하는 인자입니다. 여기서는 LOW 라고 써 주었습니다. 이것은 값이 0 인 동안을 말하는 것입니다. 앞에서 여러 번 말씀 드렸지만 코딩키트의 버튼은 평상시 1 값을 유지하다가 누르면 값이 0 으로 떨어집니다. 그래서 이 attachInterrupt() 함수의 마지막 인자에는 LOW 를 주어야 버튼이 눌려 있는 동안에 계속해서 인터럽트를 발생시킬 수 있는 것입니다.

loop() 함수는 간단하게 DC 모터를 정방향 3 초, 역방향으로 3 초간 계속해서 돌리는 코드입니다. 그럼 이제 코딩키트에서 실행시켜 보십시요. 그러면 DC 모터가 정방향, 역방향을 반복해서 돌고 있을 것입니다. 그 때 버튼을 누르면 DC 모터가 멈추었다가 다시 동작하는 것을 볼 수 있습니다.

버튼을 떼었는데도 바로 DC 모터가 돌지 않는 것은 인터럽트가 loop() 함수 코드의 delay() 함수 중간에 걸렸을 경우에는 인터럽트 함수를 다 끝낸 이후에 그 delay() 함수에서 보냈던 시간을 마저 다 보내고 나서 다음 코드를 수행하기 때문에 그런 것입니다.

< 예제 코드 : DC 모터 회전 중간에 인터럽트 걸기 >

< 코드 위치 : Coding Kit Ex1 → **dcmot_intr** >

이제 다시 DC 모터 인코더 코드 설명으로 돌아 오겠습니다. 다음과 같이 attachInterrupt() 함수를 사용하였습니다.

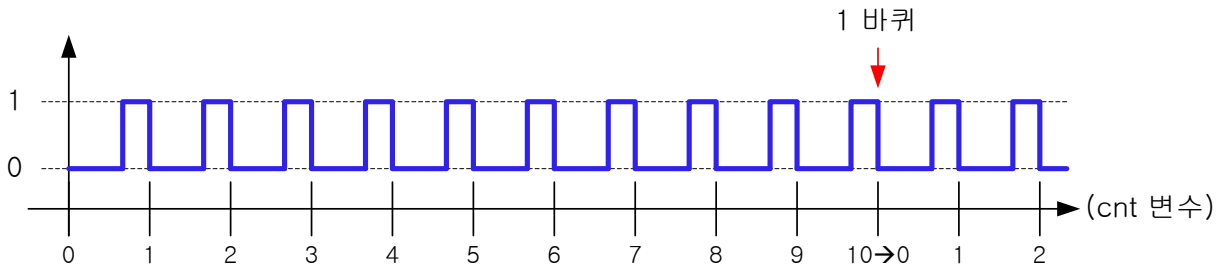
```
attachInterrupt(0, enc_cnt, FALLING);
```

이벤트 핀 ID 가 0 번이라는 것은 2 번 핀을 이용하겠다는 것이고 2 번 핀은 DCMOTENC_DATA 로 정의되어 있습니다. ISR 은 enc_cnt 함수 입니다. 이벤트 모드는 FALLING 으로 이 신호가 1 에서 0 으로 떨어질 때만 이벤트가 발행하고 그 때만 enc_cnt 함수가 호출되어 수행이 됩니다. 코드의 수행 순서가 아직 잘 이해가 되지 않으시는 분은 이 장의 마지막에 코드 수행 순서에 대해서 좀 더 자세히 설명드리겠습니다. 그 부분을 참고해 주십시요.

enc_cnt() 함수는 다음과 같습니다.

```
void enc_cnt() {
  cnt++;
  if (cnt == 10) {
    now = millis();
    interval = now - previous;
    previous = now;
    cnt = 0;
    get_rpm = 1;
  }
}
```

위의 코드에서 cnt 변수는 인코더의 날개를 카운트하여 저장하는 변수 입니다. enc_cnt() 함수는 DCMOTENC_DATA 신호의 Falling Edge 에서만 호출되어지기 때문에 여기서 cnt 값을 증가하면 그것이 인코더의 날개를 카운트하는 것과 같습니다. if 문에서 이 cnt 값이 10 인지를 체크하는 것은 10 이 되면 한 바퀴를 돌았다는 것이기 때문입니다. 이 if 문에서 체크하는 순간은 다음 그림에서 1 바퀴라고 표시된 부분입니다. 이 부분에서 cnt 값은 10 에서 0 으로 바뀝니다.



위의 코드에서 붉은색 코드에 millis() 라는 함수가 있습니다. 이 함수는 코드에서 시간을 측정할 때 사용되는 함수입니다. 이 함수는 프로그램이 시작했을 때부터 현재의 코드가 수행될 때까지의 시간을 알려 줍니다. 그래서 "now = millis()" 라고 하면 now 라는 변수에 프로그램이 시작했을 때부터 현재까지의 시간이 저장됩니다. interval 변수는 now 에서 previous 를 뺀 값을 저장합니다. 여기서 previous 는 이전 인터럽트에서의 시간이 저장되어 있습니다. 초기에는 setup() 함수에서 "previous = millis();" 하여 시간을 저장하였습니다. 다음 줄의 "previous = now;" 코드가 이전 인터럽트에서 시간을 저장하는 코드입니다. 그러면 previous 는 이전에 cnt 가 10 일 때의 값이 저장되어 있는 것입니다. 그래서 now 에서 previous 를 뺀 interval 값은 인코더의 휠이 한 바퀴 도는 시간이 됩니다. 그리고 cnt 값은 다시 0 으로 초기화 합니다. get_rpm 변수는 이제 휠이 한 바퀴 도는 interval 값을 구했으니 이것으로 RPM 값을 계산해라 하는 표시입니다.

여기서 now 와 previous 를 unsigned long 변수 타입으로 한 것은 millis() 함수가 unsigned long 변수 타입을 리턴하기 때문입니다.

이제 loop() 함수를 보도록 하겠습니다.

```
void loop() {
  int val_var_res_1 = analogRead(VAR_RES_1);
  int dcmot_speed = map(val_var_res_1, 0, 1023, 0, 255);
  analogWrite(DCMOT_FWD, dcmot_speed);

  if (get_rpm == 1) {
    get_rpm = 0;
    Serial.print("RPM : ");
    Serial.println((1000.0 * 60.0)/(float)interval);
  }
}
```

가변저항의 값을 받아 DC 모터의 속도를 조정합니다. 그 다음 if 문에서는 get_rpm 신호를 체크하고 있다가 이 신호가 1 이 되면 RPM 값을 계산하여 시리얼 모니터로 출력합니다. loop() 함수는 계속해서 반복을 하면서 get_rpm 값이 1 인지를 체크합니다. get_rpm 값은 인터럽트 서비스 루틴 함수, 즉, ISR 에서 cnt 값이 10 이 되면 1 로 되는 변수 입니다. loop() 함수에서는 반복적으로 get_rpm 값을 체크합니다. 그래야 어떤 순간에 RPM 값을 계산하여 출력할지를 알 수 있기 때문입니다.

위의 붉은색 부분이 RPM 값을 구하는 코드입니다. RPM 계산은 1 분을 interval 로 나누면 됩니다. 그런데, 분모에 있는 변수 interval 의 단위가 1000 분의 1 초이기 때문에 "1000.0 * 60.0" 을 interval 변수로 나눕니다.

< 문법 설명 : **변수 타입 변경과 상수의 실수 표현** >

위의 코드에서 (float)interval 의 코드는 값의 계산을 정수가 아닌 실수로 해 주겠다는 뜻입니다. 그래서 변수 타입을 정수 (unsigned long) 에서 실수 (float) 로 바꾸었습니다. 이렇게 변수의 타입은 바꿀 수가 있습니다. 변수 타입을 바꾸는 정의는 다음과 같습니다.

(바뀔_변수_타입)변수

변수의 앞에 괄호를 하고 괄호 안에 바꾸어 줄 변수를 타입을 쓰면 됩니다.

위의 코드에서 상수들도 1000, 60 으로 쓰지 않고 1000.0, 60.0 으로 써서 실수 계산을 했습니다. 상수에 소수점과 0 을 쓰면 "실수로 계산해 주세요." 하고 컴퓨터에 작업 지시를 내리는 것입니다.

여기서 또 한가지 더 알아두실 것은 어떤 연산을 할 때는 각 연산에 사용되는 인자들의 변수 타입을 동일하게 맞추어 주어야 한다는 것입니다.

위의 코드의 if 문 안에서 get_rpm 변수를 다시 0 으로 클리어 합니다. 그래야 loop() 함수를 반복할 때 인터럽트가 다시 발생해서 enc_cnt() 함수에서 get_rpm 신호를 다시 1 로 해 주기 전에는 if 문을 다시 수행하지 않게 됩니다. 이와 같은 코딩 기법들은 잘 보고 익혀 두십시요. 두고 두고 여러분이 코딩하는 동안에 아주 요긴하게 잘 이용될 것입니다.

< 예제 코드 : **DC 모터 인코더 사용하기 및 인터럽트 이해하기** >

< 코드 위치 : Coding Kit Ex1 → **dcmotenc >**

이제 인터럽트와 코드 실행 순서에 관해서 자세히 설명드리겠습니다.

```
 1 void enc_cnt() {
 2   cnt++;
 3   if (cnt == 10) {
 4     now = millis();
 5     interval = now - previous;
 6     previous = now;
 7     cnt = 0;
 8     get_rpm = 1;
 9   }
10 }
11
12 void loop() {
13   int val_var_res_1 = analogRead(VAR_RES_1);
14   int dcmot_speed = map(val_var_res_1, 0, 1023, 0, 255);
15   analogWrite(DCMOT_FWD, dcmot_speed);
16
17   if (get_rpm == 1) {
18     get_rpm = 0;
19     Serial.print("RPM : ");
20     Serial.println((1000.0 * 60.0)/(float)interval);
21   }
22 }
```

위의 코드는 이전 코드에서 setup() 함수는 뺀 나머지 코드들입니다. 각 코드에 줄 번호를 붙였습니다. 이 코드는 loop() 함수가 계속해서 반복적으로 수행이 됩니다. 그러면 loop() 함수의 안인 13 ~ 20 번 줄이 반복적으로 수행될 것입니다. 초기에는 변수 get_rpm 값이 0 이기 때문에 if 문 안인 18, 19, 20 번 줄은 수행이 안 될 것입니다. 그래서 13, 14, 15, 17 번 줄이 계속해서 반복적으로 수행이 될 것입니다. 만약 처음 인터럽트가 13 번 줄을 수행한 이후에 발생했다면 코드는 어떻게 수행이 될까요? 아마도 13, (인터럽트 발생), 2, 3 이 수행될 것입니다. 아직까지는 cnt 값이 10 이 아닐 것이기 때문입니다. 2 번 줄을 수행했으니 cnt 값은 1 이 되었겠지요. 이제 다시 loop() 함수로 돌아와서 13번 다음 줄인 14 번 줄부터 수행할 것입니다. 정리하면 13, (인터럽트 발생), 2, 3, 14, 15, 17, (다시 loop() 함수 반복), 13, 14, 15, 17, ... 이렇게 수행이 될 것입니다.

다시 또 인터럽트가 발생합니다. 이번에는 15 번 다음에 발생했다고 가정해 볼께요. 그러면 13, 14, 15, (인터럽트 발생), 2(cnt 2 로 증가), 3, 17, (loop() 함수 반복), 13, 14, 15, 17, ... 이렇게 코드는 계속해서 수행이 됩니다. 이렇게 코드가 수행이 되면서 인터럽트가 발생하고 해서 이제 cnt 값이 9 가 되었고, loop() 함수의 14 번 줄 다음에 인터럽트가 발생했다고 가정하겠습니다. 그러면 코드의 실행 순서는 13, 14, (인터럽트 발생), 2(cnt 10 으로 증가), 3(if 문 조건식 참), 4, 5, 6, 7, 8(get_rpm = 1, if 문 마지막 줄), 13(loop() 함수), 14, 15, 17(if 문 조건식 참), 18, 19, 20, (loop() 함수 반복), 13, 14, 15, 17(조건식 거짓), 13, 14, ... 이렇게 수행이 될 것입니다. 이제 인터럽트가 발생이 되면 코드가 어떻게 실행이 되는지 잘 아셨을 것입니다. 아직도 이해가 안 가신다면 차근차근 다시 한 번 더 따져 보십시요. 그리고 코드를 여기저기 바꾸어 가면서 테스트해 보시면 이해가 되실 것입니다. 그래도 이해가 안 가시는 부분은 코딩 사이트를 이용하여 질문 남겨 주십시요.

이번에는 자동차 속도계를 만들어 보도록 하겠습니다. 이 속도계에는 속도와 현재 주행 거리를 표시해 줍니다. 이것을 캐릭터 LCD 에 표시해 보겠습니다. DC 모터에 자동차 바퀴에 장착되어 있고, 한 바퀴 돌면 1.21 m 를 앞으로 간다고 가정합니다. 한 바퀴 돌 때 2 초가 걸린다고 하면 이 자동차의 속도는 시속으로 어떻게 될까요? 다음 식과 같습니다.

(3600초 x 1.21 m) / 2 초 = 2178 m = 2.178 km

여기서 3600초는 1 시간입니다. 즉, 시속 2.178 km 입니다. 속도가 매우 느린 것을 보니 장난감 자동차인가 봅니다? 이 식을 기초로 하여 공식을 정의 하면 다음과 같습니다.

(1시간) x (한 바퀴 돌 때 가는 거리) / (한 바퀴 도는 시간)

코드의 대부분은 위의 예제 코드와 같고 다음의 loop() 함수 중 붉은색 부분만 다릅니다.

```
#define DIST_1_CIR 1.21

float distance = 0;

void loop() {
  int val_var_res_1 = analogRead(VAR_RES_1);
  int dcmot_speed = map(val_var_res_1, 0, 1023, 0, 255);
  analogWrite(DCMOT_FWD, dcmot_speed);

  if (get_rpm == 1) {
    get_rpm = 0;
    float speed = (3600.0 * DIST_1_CIR)/(float)interval;
    ck_lcd.setCursor(0, 0);
    ck_lcd.print("SPEED : ");
    ck_lcd.setCursor(8, 0);
    ck_lcd.print(speed);
    ck_lcd.setCursor(0, 1);
    ck_lcd.print("DIST : ");
    ck_lcd.setCursor(7, 1);
    distance += DIST_1_CIR;
    ck_lcd.print(distance);
  }
}
```

위의 공식을 이용하여 속도를 구해서 LCD 의 첫번째 줄에 표시합니다. 단위는 시속, Km (킬로미터) 입니다. 두번째 줄에는 거리를 계속 더해서 표시해 줍니다. 단위는 m (미터) 입니다. 코드 중에 "+=" 기호가 있습니다. 이 기호는 오른쪽 변수에 왼쪽의 값을 더해서 다시 오른쪽 변수에 더하라는 뜻입니다. 그래서 다음 두 코드는 같습니다.

```
distance += 1.21;
```

```
distance = distance + 1.21;
```

"+=" 이외에 "-=", "*=" 등 대부분의 연산자가 가능합니다.

이제 업로드하여 코드를 실행해 보십시요.

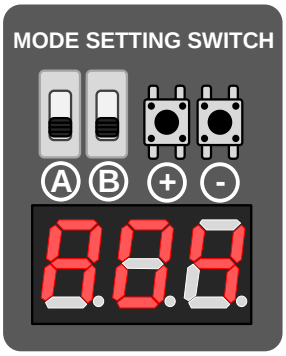
< 예제 코드 : **DC 모터 인코더를 이용한 디지털 속도계 만들기** >

< 코드 위치 : Coding Kit Ex1 → **dcmotenc_speed >**

[SPI 확장 모드를 이용한 도트매트릭스와 세븐세그먼트 컨트롤]

도트매트릭스는 코딩키트에 있는 디바이스 중 가장 많은 16 핀을 사용합니다. 세븐세그먼트도 11 핀을 사용합니다. 아두이노에서 디지털 입출력으로 최대한 사용할 수 있는 핀은 16 핀입니다. 이 중 2 핀은 시리얼 모니터로 사용되기 때문에 시리얼 모니터를 제외하면 사용할 수 있는 핀은 14 핀입니다. 그래서 이렇게 많은 핀을 사용하는 디바이스들을 SPI (Serial Peripheral Interface) 라는 인터페이스를 활용하여 4 핀으로 줄여보도록 하겠습니다. 이 4 핀을 이용하여 대부분의 디바이스를 컨트롤할 수 있습니다. 4 핀을 여러 핀으로 확장하고, SPI 인터페이스를 사용한다고 해서 이것을 SPI 확장 모드라고 합니다.

SPI 확장 모드의 스위칭 ID 는 A04 입니다. 그래서 이번 장의 스위칭 ID 는 A04 입니다.



스위치 ID 를 A04 로 하면 세븐세그먼트가 꺼집니다. 그것은 SPI 확장 모드에서 세븐세그먼트를 컨틀롤할 수 있기 때문에 꺼지는 것입니다. 즉, 그 이전까지는 스위칭 ID 를 컨트롤하는 하는 부분에서 세븐세그먼트를 컨트롤합니다. 그런데 이제부터는 여러분이 작성하는 코드가 세븐세그먼트를 컨트롤하게 되는 것입니다. 이제 여러분이 코딩을 하여 세븐세그먼트에 어떤 표시를 할 수 있는 것입니다.

먼저 SPI 란 무엇인지 알아보겠습니다. SPI 란 디바이스간의 통신을 가능하게 해 주는 상호 약속인 것입니다. 이런 것을 프로토콜(Protocol) 또는 인터페이스(Interface)라고 합니다. 일반적으로 SPI 같이 보드 위에서 디바이스 간의 통신은 인터페이스라고 주로 부릅니다. 이런 인터페이스 방식은 직렬(Serial)과 병렬(Parallel)로 나뉘어 집니다. 여기서 직렬과 병렬의 구분은 통신에 사용되는 데이터가 몇 비트인지에 따라서 나누어 집니다. 직렬은 보통 1 비트나 2 비트이고 병렬은 보통 8 비트이상입니다.

인터페이스는 데이터 신호 이외에 컨트롤 신호가 포함이 됩니다. 이 컨트롤 신호에는 보통 칩 선택 (Chip Select : CS) 신호, 클럭 (Clock) 신호, 읽기 쓰기 선택 (Read/Write : RW) 신호가 있습니다. SPI 도 비슷한 컨트롤 신호로 구성이 됩니다.

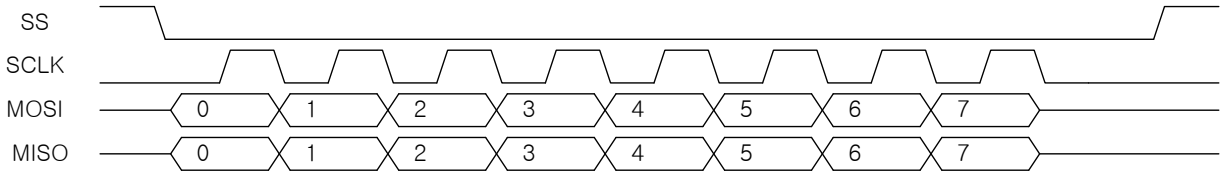
또한 디바이스 간의 인터페이스에는 마스터(Master), 슬래이브(Slave) 개념을 도입합니다. 인터페이스의 주최가 되어 통신을 시작과 끝을 결정하고 클럭 신호를 생성해 내는 등의 기능을 담당하는 쪽을 마스터라고 하고, 마스터의 신호에 따라서 응답을 하는 등의 수동적인 동작을 하는 쪽을 슬래이브라고 합니다. SPI 도 이 마스터, 슬래이브 개념을 도입합니다.

코딩키트에서는 아두이노, 라즈베리파이가 SPI 의 마스터가 되고, 코딩키트에 장착되어 있는 디바이스들이 슬래이브가 되는 것입니다.

SPI 신호는 다음과 같이 4 개의 신호로 구성됩니다.

신호 이름	방향	설 명
SS (Slave Select)	M→S	칩 선택 신호와 비슷한 신호로 슬래이브를 선택하는 신호
SCLK (Serial Clock)	M→S	클럭
MOSI (Master Output Slave Input)	M→S	마스터에서 출력되어 슬래이브로 입력되는 데이터 신호
MISO (Master Input Slave Output)	S→M	슬래이브에서 출력되어 마스터로 입력되는 데이터 신호

이 신호들의 파형은 다음과 같이 동작합니다.



먼저 SS 신호를 0 으로 내립니다. 그리고 SS 신호가 0 인 구간에 클럭을 보냅니다. 이 클럭 신호에 따라서 MOSI 나 MISO 신호를 주고 받습니다.

이제 이 SPI 를 이용하여 도트매트릭스나 세븐세그먼트를 컨트롤해 보겠습니다. 먼저 도트매트릭스를 컨트롤하는 코드를 보겠습니다.

```
#include <SPI.h>

#define SPI_SS 10

#define SPI_CMD_CLEAR         0x00
#define SPI_CMD_DM_CL_SEL   0x00
#define SPI_CMD_DM_ROW      0x04
#define SPI_CMD_DM_COL      0x05

#define SELECT_DOTMATRIX    0

void setup() {
  pinMode(SPI_SS, OUTPUT);
  SPI.begin();
  spiCmd(SPI_CMD_CLEAR, 0x00);
  spiCmd(SPI_CMD_DM_CL_SEL, SELECT_DOTMATRIX);
}

void spiCmd(int cmd, int data) {
  digitalWrite(SPI_SS, 0);
  SPI.transfer(cmd);
  SPI.transfer(data);
```

```
  digitalWrite(SPI_SS, 1);
}

void loop() {
  spiCmd(SPI_CMD_DM_ROW, 0xff);
  spiCmd(SPI_CMD_DM_COL, 0x00);
  delay(1000);
}
```

SPI 를 사용하기 위해서는 코드의 "#include <SPI.h>" 와 같이 SPI 라이브러리를 포함하여야 합니다. 포함된 SPI 라이브러리를 사용하기 위해서 setup() 함수에서 SPI.begin() 함수로 SPI 를 초기화 하였습니다. 아두이노에서는 SPI 의 신호에 다음 핀들을 이용합니다.

신호 이름	핀번호
SS (Slave Select)	10
SCLK (Serial Clock)	13
MOSI (Master Output Slave Input)	11
MISO (Master Input Slave Output)	12

SS 신호는 10 번 핀 대신 다른 핀을 사용해도 됩니다. 이 핀은 그냥 일반 핀을 digitalWrite() 함수로 사용하는 것과 같게 사용합니다. setup() 함수에서 이 핀의 핀 모드를 설정하고 spiCmd() 함수에서 digitalWrite() 함수로 값을 쓰는 것을 볼 수 있습니다.

코딩키트의 디바이스 컨트롤용 SPI 신호는 2 바이트로 구성됩니다. 첫번째 바이트는 커맨드(Command)라고 하여 다음에 나올 데이터가 어떤 종류의 데이터인지를 알려 줍니다. 두번째 바이트 데이터 값으로 주로 핀의 값을 전달합니다. 이것을 구현한 코드가 spiCmd() 함수 입니다. 이 함수를 보면 먼저 SS 신호에 0 을 주어 SPI 인터페이스의 시작을 슬래이브에 알립니다. 앞에서도 말씀드렸듯이 아두이노는 지금 SPI 마스터의 역할을 수행 중입니다. 그래서 SPI 마스터 코드를 작성하는 것입니다. SS 신호를 0 으로 떨어뜨려 SPI 통신을 시작하는 것은 SPI 마스터의 역할입니다. 다음의 SPI.transfer() 함수는 괄호 안의 값을 슬래이브로 보내는 역할을 합니다. SPI.transfer(cmd) 하면 cmd 변수 값을 슬래이브로 보냅니다. SPI.transfer(data) 하면 data 변수 값을 슬래이브로 보냅니다. 이렇게 커맨드(cmd)와 데이터(data)를 보내면 슬래이브에서는 이것을 받아 해당 디바이스를 컨트롤합니다. 커맨드와 데이터의 전송이 완료되면 SS 신호를 1 로 올립니다.

loop() 함수를 보기 전에 정의해둔 커맨드를 보겠습니다. 이 커맨드에 따라서 디바이스들이 컨트롤 되는 것입니다.

커맨드	코드 (16진수)	설명
SPI_CMD_CLEAR	0x00	디바이스들을 초기화 합니다.
SPI_CMD_DM_ROW	0x04	도트매트릭스의 가로줄 설정

SPI_CMD_DM_COL	0x05	도트매트릭스의 세로줄 설정

SPI_CMD_CLEAR 는 모든 디바이스를 초기화 합니다. 즉, 도트매트릭스, 세븐세그먼트를 모두 끕니다. 이 이외의 디바이스도 모두 꺼서 초기화를 합니다. SPI_CMD_DM_ROW 는 도트매트릭스의 가로줄을 설정합니다. 그래서 loop() 함수에서 "spiCmd(SPI_CMD_DM_ROW, 0xff)" 하면 가로줄을 모두 1 로 설정하는 것입니다. 이것을 SPI 를 이용하지 않은 도트매트릭스 코드에서는 다음과 같이 하였습니다.

```
digitalWrite(DM_ROW1, 1);
digitalWrite(DM_ROW2, 1);
digitalWrite(DM_ROW3, 1);
digitalWrite(DM_ROW4, 1);
digitalWrite(DM_ROW5, 1);
digitalWrite(DM_ROW6, 1);
digitalWrite(DM_ROW7, 1);
digitalWrite(DM_ROW8, 1);
```

이것과 같은 작업을 SPI 를 이용하여 하려고 하는 것입니다. 위의 코드에서 DM_ROW1 ~ DM_ROW8 은 SPI_CMD_DM_ROW 를 슬래이브로 전달함으로서 이 핀들이 선택되는 것이고 모두 1 을 써 주는 것은 0xff 의 데이터 값을 슬래브로 전달함으로서 실행이 됩니다. 이 코드는 다음과 같은 것입니다.

```
spiCmd(SPI_CMD_DM_ROW, 0xff);
```

그러면 세로줄에 다음과 같이 쓰는 코드는 SPI 를 이용한 코드에서는 어떻게 할까요?

```
digitalWrite(DM_COL1, 1);
digitalWrite(DM_COL2, 0);
digitalWrite(DM_COL3, 1);
digitalWrite(DM_COL4, 1);
digitalWrite(DM_COL5, 0);
digitalWrite(DM_COL6, 1);
digitalWrite(DM_COL7, 1);
digitalWrite(DM_COL8, 1);
```

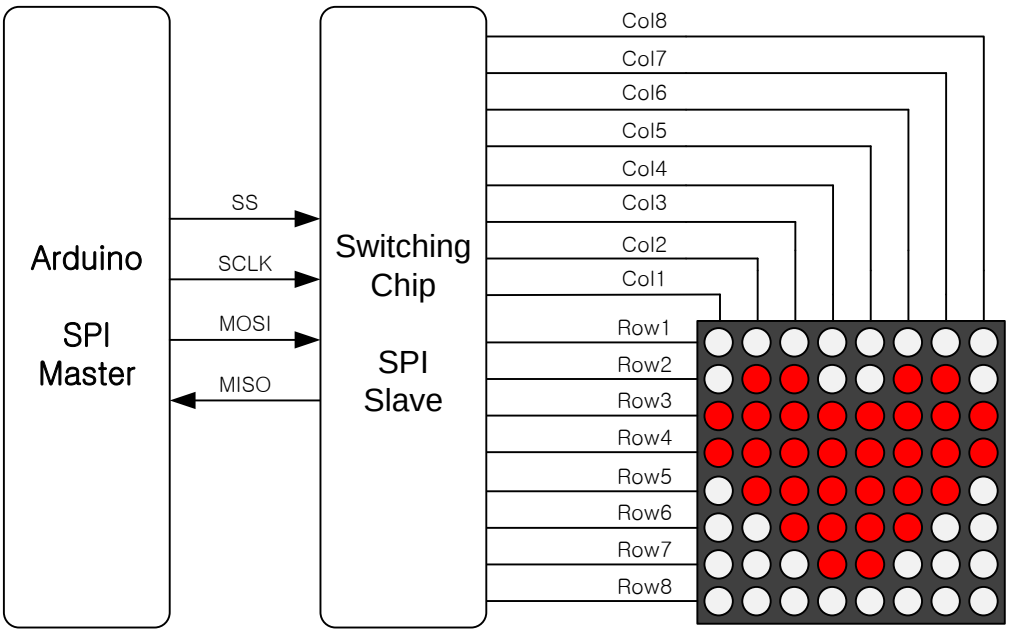
위의 코드는 다음과 같이 바뀝니다.

```
spiCmd(SPI_CMD_DM_COL, 0b10110111);
```

이번에는 2 진수를 사용해 보았습니다. 16 진수로는 0xb7 입니다. 이렇게 하면 세로줄에 0xb7 를 쓰게 도는 것입니다. 가로줄 또는 세로줄에 해당하는 핀 8 개를 8 비트 신호로 처리한 것입니다. 전에는 8 핀을 처리하기 위해서 배열을 사용하였습니다. 이번에는 8 비트로 처리한 것입니다.

이렇게 SPI 를 이용하여 도트매트릭스를 사용하면 아두이노의 핀 소모를 확실히 줄일 수 있습니다. 전에는 16핀을 이용하여 도트매트릭스를 컨트롤하였다면 이제는 4 핀만을 사용합니다. 남는 12 핀은 LED 나 모터 등에 연결하여 사용할 수 있는 것입니다.

그리고 한 가지 더 알아 두셔야할 것은 여기서 SPI 의 슬래이브는 어떤 디바이스일까요? 그것은 바로 코딩키트에 있는 스위칭 칩입니다. 이 칩에 SPI 슬래이브가 있고 이 슬래이브가 아두이노나 라즈베리파이의 마스터와 인터페이스하는 것입니다. 이 인터페이스로 받은 신호를 분석하여 디바이스를 컨트롤하는 것입니다. 여기서 사용된 도트매트릭스는 다음과 같이 컨트롤 되는 것입니다.



스위칭 칩에서는 아두이노에서 보내는 SPI 신호를 분석하여 이 신호에서 커맨드와 데이터를 찾아냅니다. 그리고 커맨드 중에 SPI_CMD_DM_ROW 가 입력되면 도트매트릭스의 가로줄을 컨트롤합니다. SPI_CMD_DM_COL 이 입력되면 세로줄을 컨트롤합니다.

이제 업로드하여 코딩키트에서 확인해 보세요. 도트매트릭스가 전부 켜진 것을 볼 수 있을 것입니다.

< 예제 코드 : **SPI 인터페이스를 이용하여 도트매트릭스 컨트롤하기** >

< 코드 위치 : Coding Kit SPI → **spi_dotmat >**

여기서 한 가지 더 설명 드릴 것이 있습니다. 코딩키트는 구조상 도트매트릭스와 캐릭터 LCD 를 같이 사용하지 못합니다. 그래서 다음과 같은 코드로 도트매트릭스를 사용할 것인지 캐릭터 LCD 를 사용할 것인지를 정해 주어야 합니다.

```
#define SPI_CMD_CLEAR        0x00
#define SPI_CMD_DM_CL_SEL    0x00
#define SPI_CMD_DM_ROW       0x04
#define SPI_CMD_DM_COL       0x05

#define SELECT_DOTMATRIX     0

void setup() {
```

```
  pinMode(SPI_SS, OUTPUT);
  SPI.begin();
  spiCmd(SPI_CMD_CLEAR, 0x00);
  spiCmd(SPI_CMD_DM_CL_SEL, SELECT_DOTMATRIX);
}
```

위의 코드 중 붉은 색 부분이 도트매트릭스를 선택하는 부분입니다. SPI 커맨드는 SPI_CMD_DM_CL_SEL 입니다. 이 커맨드에 데이터를 0 으로 하면 도트매트릭스가 선택이 되고 1 로 하면 캐릭터 LCD 가 선택이 됩니다. 그래서 SELECT_DOTMATRIX 는 0 으로 정의(#define SELECT_DOTMATRIX 0)하였습니다.

이번에는 도트매트릭스에 하트를 그려보겠습니다. 코드는 다음과 같습니다.

```
#include <SPI.h>

#define SPI_SS 10

#define SPI_CMD_CLEAR        0x00
#define SPI_CMD_DM_CL_SEL 0x00
#define SPI_CMD_DM_ROW     0x04
#define SPI_CMD_DM_COL      0x05

#define SELECT_DOTMATRIX    0

void spiCmd(int cmd, int data) {
  digitalWrite(SPI_SS, 0);
  SPI.transfer(cmd);
  SPI.transfer(data);
  digitalWrite(SPI_SS, 1);
}

#define ROW_ON    1
#define ROW_OFF   0

int dm_heart[8] = {
  0b11111111,
  0b10011001,
  0b00000000,
  0b00000000,
  0b10000001,
  0b11000011,
  0b11100111,
  0b11111111
};

void setup() {
  pinMode(SPI_SS, OUTPUT);
  SPI.begin();
  spiCmd(SPI_CMD_CLEAR, 0x00);
  spiCmd(SPI_CMD_DM_CL_SEL, SELECT_DOTMATRIX);
```

```
}

void loop() {
  for (int i = 0; i < 8; i++) {
    spiCmd(SPI_CMD_DM_ROW, 0x1 << i);
    spiCmd(SPI_CMD_DM_COL, dm_heart[i]);
    delay(2);
  }
}
```

코드가 매우 간단하지요? 코드 중간의 붉은색 부분의 2 진수 배열 부분이 보일 것입니다. 여기에 0 으로 하트 모양이 파란색으로 보이시지요. 앞에서 2 차원 배열로 했던 것을 1 차원 배열의 2 진수를 이용하여 8 비트를 처리하였습니다. 이 8 비트가 세로줄 각각에 입력되는 것입니다.

loop() 함수의 for 문의 도트매트릭스 가로줄 설정 부분에서 "0x1 << i" 라는 코드가 보입니다. 이것은 쉬프트(Shift) 연산자입니다. 0x1 값을 좌로 i 만큼 쉬프트하라는 뜻입니다. 그럼 i 값이 0 에서 7 까지 변할 때 이 값은 어떻게 변하는지 보겠습니다.

인덱스 i	0x1 << i (2 진수)
0	0b00000001
1	0b00000010
2	0b00000100
3	0b00001000
4	0b00010000
5	0b00100000
6	0b01000000
7	0b10000000

위의 표와 같은 값을 가로줄에 써 주게 되면 가로줄이 한 줄씩 켜지게 됩니다. 그리고 그렇게 켜지는 가로줄에 맞춰 세로줄의 값을 하트모양으로 그려주게 되면 도트매트릭스에 하트 모양이 생기게 되는 것입니다.

< 예제 코드 : **SPI 인터페이스를 이용하여 도트매트릭스에 하트 그리기** >

< 코드 위치 : Coding Kit SPI → **spi_dotmat_heart >**

이번에는 SPI 를 이용하여 세븐세그먼트를 컨트롤해 보겠습니다. 세븐세그먼트는 FND 라고도 부릅니다. 도트매트릭스 코드를 이해하셨다면 FND 코드도 쉽게 이해하실 것입니다. 그래서 코드 먼저 보도록하겠습니다.

```
#include <SPI.h>

#define SPI_SS 10

#define SPI_CMD_CLEAR   0x00
#define SPI_CMD_FND_SEG 0x06
#define SPI_CMD_FND_SEL 0x07

void spiCmd(int cmd, int data) {
  digitalWrite(SPI_SS, 0);
  SPI.transfer(cmd);
  SPI.transfer(data);
  digitalWrite(SPI_SS, 1);
}

void setup() {
  pinMode(SPI_SS, OUTPUT);
  SPI.begin();
  spiCmd(SPI_CMD_CLEAR, 0xff);
}


void loop() {
  spiCmd(SPI_CMD_FND_SEG, 0x00);
  spiCmd(SPI_CMD_FND_SEL, 0x00);
  delay(1000);
}
```

코드만 보시고도 눈치빠른 분들은 바로 이해하셨을 것입니다. "도트매트릭스에서는 SPI_CMD_DM 이라는 커맨드를 썼는데, FND 에서는 SPI_CMD_FND 라는 커맨드를 쓰는구나!" 하고 말입니다. 그렇습니다. FND 에서는 SPI_CMD_FND_SEG 와 SPI_CMD_FND_SEL 라는 커맨드를 사용합니다. 이 중 SPI_CMD_FND_SEG 는 FND 의 각 세그먼트를 컨트롤하는 것이고, SPI_CMD_FND_SEL 은 세개의 FND 중 하나를 선택하는 것입니다. 그래서 이렇게 두 개의 커맨드만 있으면 FND 를 컨트롤할 수 있습니다. 이 중 SPI_CMD_FND_SEG 커맨드의 데이터는 8 비트를 이용하여 8 개의 세그먼트를 컨트롤하고 SPI_CMD_FND_SEL 커맨드의 데이터는 3 비트만을 이용하여 FND 3 개 중 몇 개를 선택하는 선택 신호를 컨트롤 합니다. 이 코드의 loop() 함수에서는 SPI_CMD_FND_SEG 와 SPI_CMD_FND_SEL 커맨드 모두에 0 을 주어서 FND 가 모두 켜지게 하였습니다. 이제 업로드해 보시면 다음과 같습니다.

< 예제 코드 : **SPI 인터페이스를 이용하여 세븐세그먼트 컨트롤하기** >

< 코드 위치 : Coding Kit SPI → **spi_fnd >**

이번에는 FND 각각에 숫자 1, 2, 3 을 쓰는 코드를 보겠습니다.

```
#include <SPI.h>
```

```
#define SPI_SS 10

#define SPI_CMD_CLEAR   0x00
#define SPI_CMD_FND_SEG 0x06
#define SPI_CMD_FND_SEL 0x07

#define FND_SPI_SEL_1  0b011
#define FND_SPI_SEL_2  0b101
#define FND_SPI_SEL_3  0b110

#define FND_ON_DELAY 2

void spiCmd(int cmd, int data) {
  digitalWrite(SPI_SS, 0);
  SPI.transfer(cmd);
  SPI.transfer(data);
  digitalWrite(SPI_SS, 1);
}

// Turn ON FND
// 0 assign to Segment and 0 assign to Select

int fnd_0 = 0b00000011;
int fnd_1 = 0b10011111;
int fnd_2 = 0b00100101;
int fnd_3 = 0b00001101;
int fnd_4 = 0b10011001;
int fnd_5 = 0b01001001;
int fnd_6 = 0b01000001;
int fnd_7 = 0b00011111;
int fnd_8 = 0b00000001;
int fnd_9 = 0b00001001;

void setup() {
  pinMode(SPI_SS, OUTPUT);
  SPI.begin();
  spiCmd(SPI_CMD_CLEAR, 0xff);
}

void loop() {
  spiCmd(SPI_CMD_FND_SEG, fnd_1);
  spiCmd(SPI_CMD_FND_SEL, FND_SPI_SEL_1);
  delay(FND_ON_DELAY);

  spiCmd(SPI_CMD_FND_SEG, fnd_2);
  spiCmd(SPI_CMD_FND_SEL, FND_SPI_SEL_2);
  delay(FND_ON_DELAY);

  spiCmd(SPI_CMD_FND_SEG, fnd_3);
  spiCmd(SPI_CMD_FND_SEL, FND_SPI_SEL_3);
```

```
    delay(FND_ON_DELAY);
  }
```

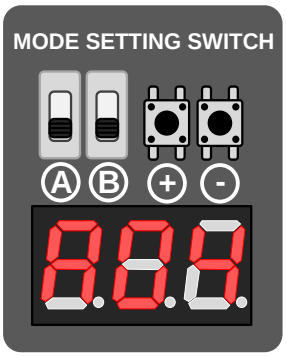
전에 SPI 를 사용하지 않고 했던 코드에서 숫자를 표시하는 배열을 2 진수의 8 비트 데이터로 바꾸었습니다. 그리고 배열 값을 일일이 digitalWrite() 함수로 썼던 것을 간단하게 spiCmd() 함수로 해결하였습니다. 이전의 배열 코드와 이전의 SPI 를 이용한 코드들과 같이 이 코드를 보시면 어렵지 않게 해결하실 수 있을 것입니다. 혹시 이해가 안 되는 부분은 코딩키트 사이트에 질문 남겨 주세요.

< 예제 코드 : **SPI 인터페이스를 이용하여 세븐세그먼트에 1, 2, 3 쓰기** >

< 코드 위치 : Coding Kit SPI → **spi_fnd_123 >**

[SPI 핀 확장 모드를 사용하여 여러 디바이스 연결하기]

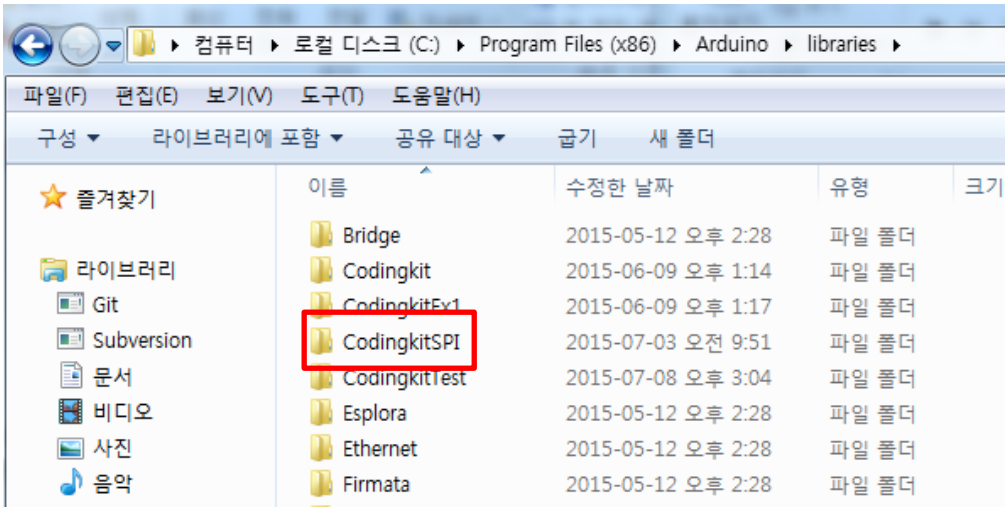
이번장의 스위칭 ID 는 A04 입니다.



SPI 핀 확장 모드를 이용하면 도트매트릭스와 세븐세그먼트뿐만 아니라 좀 더 많은 디바이스를 연결할 수 있습니다. 다음은 핀 확장 모드의 SPI 커맨드들입니다.

Command	코드 (16진수)	설명
SPI_CMD_CLEAR	0x00	디바이스들을 초기화 합니다.
SPI_CMD_LED	0x01	LED 값 설정
SPI_CMD_DM_ROW	0x04	도트매트릭스의 가로줄 설정
SPI_CMD_DM_COL	0x05	도트매트릭스의 세로줄 설정
SPI_CMD_FND_SEG	0x06	FND 세그먼트 값 설정
SPI_CMD_FND_SEL	0x07	각 FND 선택 신호 설정
SPI_CMD_BUTTON	0x10	버튼 값 받음
CMD_SWITCH	0x11	스위치 값 받음

CodingkitSPI 라이브러리를 이용하면 더 쉽게 코딩을 할 수 있습니다. 이 라이브러리는 코딩 사이트에서 다운로드 받아 압축을 해제한 뒤에 다음 그림과 같이 아두이노의 설치 폴더에 있는 libraries 폴더에 두시면 됩니다.



다음은 LED 를 컨트롤하는 예제입니다.

```
#include <CodingkitSPI.h>

void setup() {
  ck_spiBegin();
}

void loop() {
  ck_spiWr(SPI_CMD_LED, 0x0f);
  delay(1000);
  ck_spiWr(SPI_CMD_LED, 0x07);
  delay(1000);
  ck_spiWr(SPI_CMD_LED, 0x03);
  delay(1000);
  ck_spiWr(SPI_CMD_LED, 0x01);
  delay(1000);
  ck_spiWr(SPI_CMD_LED, 0x00);
  delay(1000);
}
```

CodingkitSPI 라이브러리를 사용하기 위해서 CodingkitSPI.h 를 include 했습니다. setup() 함수의 ck_spiBegin() 은 SPI 확장 핀 모드를 초기화 하는 함수 입니다. ck_spiWr() 함수는 SPI 커맨드와 데이터를 코딩키트로 전달하여 각 디바이스들을 컨트롤하는 함수입니다. ck_spiWr() 함수는 다음과 같이 정의 됩니다.

```
void ck_spiWr(커맨드, 데이터);
```

여기서 커맨드는 위의 표와 같습니다. 각 커맨드에 따라서 해당 데이터가 달라집니다. 데이터는 8 비트로 구성되어 있는데, 이것을 모든 비트를 다 사용할 수도 있고 일부만 사용할 수도 있습니다. 각 커맨드에 따른 데이터는 다음과 같습니다.

Command	Bits	Data	R/W	ON
SPI_CMD_CLEAR	X	X	W	X
SPI_CMD_LED	[3:0]	LED_3, LED_2, LED_1, LED_0	W	1
SPI_CMD_DM_ROW	[7:0]	Row1, Row2, ..., Row7, Row8	W	1
SPI_CMD_DM_COL	[7:0]	Col1, Col2, ..., Col7, Col8	W	0
SPI_CMD_FND_SEG	[7:0]	A, B, C, D, E, F, G, DP	W	0
SPI_CMD_FND_SEL	[2:0]	sel1, sel2, sel3	W	0
SPI_CMD_BUTTON	[3:0]	BUT_3, BUT_2, BUT_1, BUT_0	R	0
SPI_CMD_SWITCH	[3:0]	SW_3, SW_2, SW_1, SW_0	R	1

첫번째 CLEAR 커맨드의 데이터는 아무 의미가 없습니다. 이 커맨드가 입력되면 모든 디바이스 연결이 끊어지고 모든 디바이스는 동작을 하지 않는 상태가 됩니다. LED 커맨드는 LED 를 컨트롤 합니다. 여기에 사용되는 데이터는 하위 4 비트입니다. 이것을 [3:0] 으로 표시했습니다. 3, 2, 1, 0 번 비트를 나타내는 것입니다. 각 비트는 순서에 맞게 LED_3, LED_2, LED_1, LED_0 을 나타냅니다. 그래서 각각의 LED 를 켜는 코드는 다음과 같습니다.

```
ck_spiWr(SPI_CMD_LED, 0b0001);     // 0 번 LED ON
ck_spiWr(SPI_CMD_LED, 0b0010);     // 1 번 LED ON
ck_spiWr(SPI_CMD_LED, 0b0100);     // 2 번 LED ON
ck_spiWr(SPI_CMD_LED, 0b1000);     // 3 번 LED ON
```

각 비트를 1 로 하면 해당 LED 가 켜집니다. 데이터 값은 주로 2 진수나 16 진수를 사용합니다. 그러면 해당 비트의 값을 쉽게 표현할 수 있습니다. 16 진수로 하면 다음과 같습니다.

```
ck_spiWr(SPI_CMD_LED, 0x1);        // 0 번 LED ON
ck_spiWr(SPI_CMD_LED, 0x2);        // 1 번 LED ON
ck_spiWr(SPI_CMD_LED, 0x4);        // 2 번 LED ON
ck_spiWr(SPI_CMD_LED, 0x8);        // 3 번 LED ON
```

제가 굳이 또 16 진수의 코드를 보여드리는 이유는 코딩을 할 때 16 진수를 꽤 많이 사용합니다. 특히 하드웨어를 다루면서 비트 연산을 할 때는 더욱 더 그렇습니다. 2 진수를 쓰면 더 보기 좋긴 하겠지만 2 진수는 값을 표현하는데, 코드가 너무 길어집니다. 그래서 16 진수를 주로 사용합니다.

그래서 위의 코드에서 다음과 같이 0x0f, 0x07, 0x03, 0x01, 0x00 을 쓴 것은 어떤 의미일까요?

```
ck_spiWr(SPI_CMD_LED, 0x0f);
ck_spiWr(SPI_CMD_LED, 0x07);
ck_spiWr(SPI_CMD_LED, 0x03);
```

```
ck_spiWr(SPI_CMD_LED, 0x01);
ck_spiWr(SPI_CMD_LED, 0x00);
```

모든 LED 를 켰다가 왼쪽의 LED 부터 하나씩 꺼 주는 것입니다.

위의 표의 마지막 항목에 있는 "ON" 열의 의미는 ON 이 되기 위해서는 0 혹은 1 값 중 어떤 값을 써 주어야 하느냐는 의미입니다. LED 를 ON 할 때는 1 을 써 주어야 합니다. 그래서 LED 커맨드의 ON 항목은 1 인 것입니다. 버튼은 눌리면 0 입니다. 그래서 ON 항목이 0 인 것입니다.

< 예제 코드 : **SPI 를 이용한 LED 켜기** >

< 코드 위치 : Coding Kit SPI → **spi_led >**

도트매트릭스와 FND 는 이전 장에서 설명드렸습니다. 이제 버튼과 스위치가 남았습니다. 버튼과 스위치는 입력입니다. 즉, 값을 읽어와야 한다는 뜻입니다. 그래서 ck_spiRd() 함수를 사용합니다. 이 함수는 다음과 같이 정의 합니다.

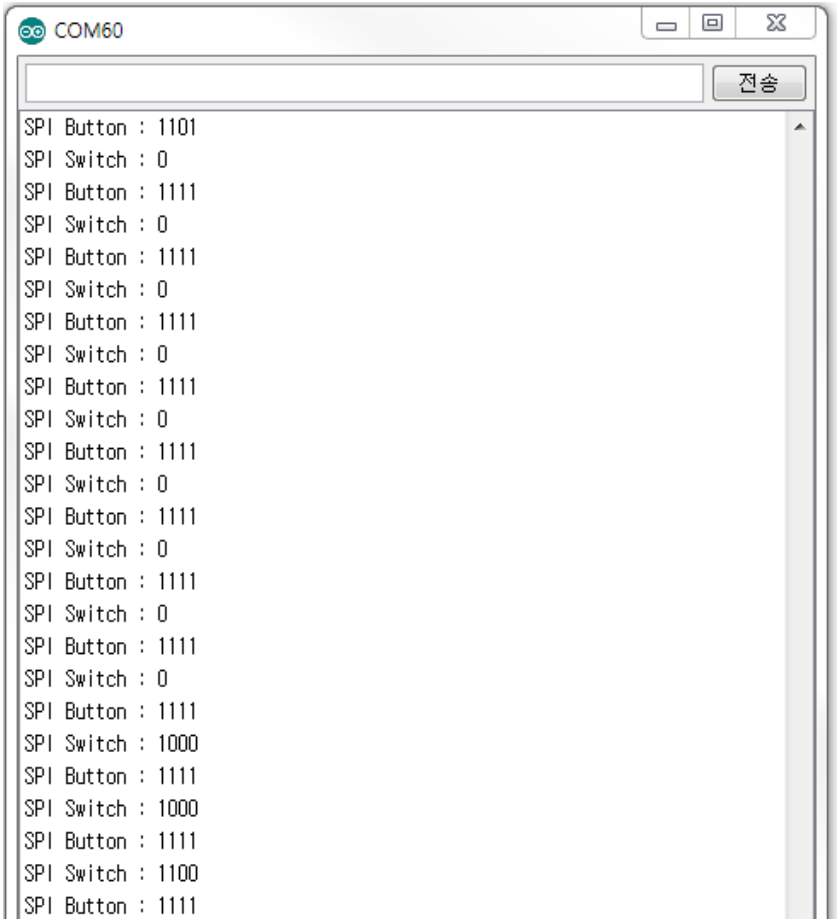
```
int ck_spiRd(커맨드);
```

이 함수는 커맨드를 받아서 해당 커맨드에 해당하는 데이터를 반환합니다. 커맨드가 SPI_CMD_BUTTON 이라면 4 개의 버튼 값을 위의 표와 같이 반환해 줍니다. 만약 반환 값이 0b1011 이면 2 번 버튼만 눌린 것이 됩니다. 커맨드가 SPI_CMD_SWITCH 이면 4 개의 스위치 값을 표와 같이 반환해 줍니다.

다음 예제는 버튼과 스위치의 상태를 시리얼 모니터에 표시하는 것입니다.

```
#include <CodingkitSPI.h>

void setup() {
  ck_spiBegin();
  Serial.begin(9600);
}

void loop() {
  Serial.print("SPI Button : ");
  Serial.println(ck_spiRd(SPI_CMD_BUTTON), BIN);
  Serial.print("SPI Switch : ");
  Serial.println(ck_spiRd(SPI_CMD_SWITCH), BIN);
  delay(500);
}
```

코딩 키드에 업로드하고 시리얼 모니터를 보면 다음과 같습니다.



버튼을 눌러 보거나 스위치를 위 아래로 내려 보십시요. 위 코드 중 다음의 BIN 이라고 쓴 것은 2 진수로 값을 출력하라는 뜻입니다.

Serial.println (ck_spiRd (SPI_CMD_BUTTON), BIN)

비슷하게 HEX 는 16 진수 OCT 는 8 진수로 출력하라는 의미입니다.

< 예제 코드 : **SPI 를 이용하여 버튼과 스위치 값 읽기** >

< 코드 위치 : Coding Kit SPI → **spi_but_switch** >

버튼 값 각각을 읽을 수 있는 함수도 있습니다.

int ck_spiRdBut(버튼_번호)

버튼_번호를 인자로 넘겨 주면 해당 버튼이 눌렸는지 안 눌렸는지를 0 또는 1 값을 반환해 줍니다.

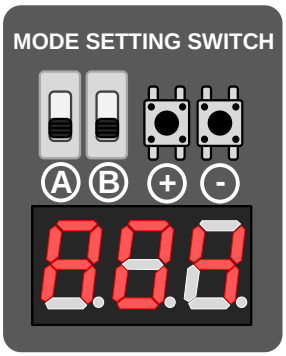
비슷하게 스위치 값 각각을 읽을 수 있는 함수도 있습니다.

int ck_spiRdBut(스위치_번호)

이 두 함수는 버튼 혹은 스위치 각각이 따로 컨트롤 되는 일이 많아서 만들어 둔 것입니다.

[SPI 핀 확장 모드를 사용하여 캐릭터 LCD 에 문자 쓰기]

이번장의 스위칭 ID 는 A04 입니다.



SPI 핀 확장 모드를 이용하여 캐릭터 LCD 를 사용하기 위해서는 LiquidCrystal_spi 라이브러리를 include 해야 합니다. 다음은 LCD 에 "Coding Kit!" 라고 표시하는 코드입니다.

```
#include <LiquidCrystal_spi.h>

LiquidCrystal_spi ck_lcd;

void setup() {
  ck_spiWr(SPI_CMD_DM_CL_SEL, SELECT_CHAR_LCD); // Select Character LCD
  ck_lcd.begin(16, 2);
  ck_lcd.print("Coding Kit!");
}

void loop() {
}
```

이전에 SPI 핀 확장 모드가 아닌 코드와 비교해 보면 매우 비슷한 것을 알 수 있습니다.

```
LiquidCrystal ck_lcd(LCD_RS, LCD_EN, LCD_D0, LCD_D1, LCD_D2, LCD_D3);

void setup() {
  pinMode(LCD_B_LIGHT, OUTPUT);
  digitalWrite(LCD_B_LIGHT, 1); // LCD Back Light ON

  ck_lcd.begin(16, 2);
  ck_lcd.print("Coding Kit!");
}

void loop() {
}
```

위의 코드와 비교해 보시면 LCD 에 여러 핀을 입력하여 초기화 하는 것은 SPI 확장 모드에서는 간단하게 선언해 주었습니다. Back Light 를 선언해 주는 부분도 SPI 확장 모드에서는 라이브러리에서 자동으로 해

주어 코드에서는 생략이 됩니다.

이제 코딩키트에 업로드하여 LCD 에 "Coding Kit!" 가 표시되는 것을 확인 봅니다.

< 예제 코드 : **SPI 확장 모드에서 캐릭터 LCD 에 문자 쓰기** >

< 코드 위치 : Coding Kit SPI → **spi_charlcd >**

SPI 확장 모드에서는 ck_lcd.print() 함수 이외의 함수들도 동일하게 사용할 수 있습니다. 다음은 이전에 했던 것과 같은 코드인 온도 센서의 값을 LCD 에 표시하고 온도가 올라가는 것을 ">" 기호로 표시하는 코드입니다. 얼마나 비슷한지 비교해 보십시요.

```
#include <LiquidCrystal_spi.h>

#define SENSOR_TEMP A2

LiquidCrystal_spi ck_lcd;

void setup() {
 ck_spiWr(SPI_CMD_DM_CL_SEL, SELECT_CHAR_LCD); // Select Character LCD
 ck_lcd.begin(16, 2);  // COL x ROW
}

#define TEMP_MIN    530

void loop() {
  int temp = analogRead(SENSOR_TEMP);
  ck_lcd.clear();

  // The 1st Line
  ck_lcd.setCursor(0, 0);
  ck_lcd.print("TEMP : ");
  ck_lcd.setCursor(7, 0);
  ck_lcd.print(temp);

  // The 2nd Line
  int num_char = (temp - TEMP_MIN)/5;
  if (num_char >= 15)
    num_char = 15;

  for (int i = 0; i < num_char; i++) {
    ck_lcd.setCursor(i, 1);
    ck_lcd.print(">");
  }

  delay(500);
}
```

코드가 매우 비슷하여 길게 설명드리지는 않겠습니다.

그러면 이렇게 SPI 확장 모드를 사용하는 이유는 무엇일까요? 그것은 핀 수를 대폭 줄일 수 있어서 입니다. SPI 확장 모드를 사용하면 아두이노에 코딩키트에 있는 모든 디바이스를 연결할 수 있습니다. 단, 한 가지 제약 사항은 도트매트릭스와 캐릭터 LCD 를 동시에 쓸 수 없다는 것뿐 입니다. 그러면 이제 여러분은 SPI 확장 모드를 활용하여 코딩키트의 모든 디바이스를 커트롤하는 실습을 해 볼 수 있는 것입니다.

< 예제 코드 : **SPI 확장 모드에서 캐릭터 LCD 에 온도 표시하기** >

< 코드 위치 : Coding Kit SPI → **spi_charlcd_temp >**

[문법 정리]

여기까지 잘 따라와 주신 여려분 감사합니다. 여기까지 착실히 공부해 오신 분이라면 처음 코딩키트를 접하셨을 때보다 매우 많이 코딩 실력이 늘었을 것입니다. 이 코딩키트에 나와 있는 부분들이 기초적인 부분이고 앞으로 배워야 할 부분이 많습니다. 하지만 어떤 일이든 기초가 중요합니다. 여기서 했던 기초 코딩 공부가 앞으로 여러분의 코딩 공부 또는 취직해서 코딩 일을 할 때 든든한 밑거름이 될 것입니다. 그래서 코딩의 기본이라고도 할 수 있는 기초를 어떻게 배우냐는 것은 매우 중요합니다. 코딩키트가 여러분의 코딩 인생에 매우 중요한 기본이 되었으면 합니다.

이제 끝으로 중요하지만 기회가 없어 설명하지 못한 컴퓨터 언어의 문법 설명 몇 가지를 하겠습니다.

< 문법 설명 : **while 문** >

while 문은 loop 함수와 같이 계속해서 반복을 하는 구문입니다. 다음과 같이 사용합니다.

```
while (조건식) {
  문장;
}
```

괄호 안의 조건식이 참이면 중괄호 안의 문장을 수행합니다. 만약 다음과 같이 쓰면 문장을 문한 반복해서 계속 수행합니다.

```
while(1) {
  문장;
}
```

while 문의 괄호 안에 1 을 쓰면 계속해서 참이되어 문장을 무한반복하여 계속 수행합니다. 마치 loop() 함수와 같은 것입니다. 보통 조건식은 다음과 같이 하여 어느정도 수행하여 while 문을 빠져나가도록 합니다.

```
i = 0;
sum = 0;
while(i < 10) {
  sum += i;
  i++
}
```

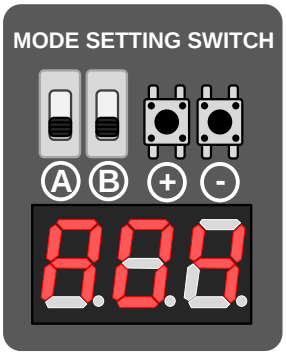
위의 코드는 0 부터 9 까지 더하는 코드입니다.

< 연습 문제 : **while 문을 이용하여 setup() 함수에서 버튼 체크하여 도트매트릭스 켜기** >

코딩키트에서 테스트해 볼 수 있는 예제를 하나 해 보겠습니다. 여자 친구에게 고백을 하기 위해서 도트매트릭스에 하트가 들어오게 하고 싶습니다. 그런데, 처음부터 켜는 것은 아니고 여자 친구가 문을 열고 들어 오면 켜는 것으로 해 볼까요? 그래서 setup() 함수 안에서 while 문으로 기다리다가 버튼이 눌리면 도트매트릭스에 하트를 켜는 것입니다.

```
while(ck_spiRdBut(0) != BUTTON_PRESSED) {
  delay(100);
}
while(1)
  dotmatrix_heart();
```

나머지는 여러분이 직접 작성해 보세요. 도트매트릭스는 SPI 를 이용하여 컨트롤합니다. 그래서 스위칭 ID 는 A04 입니다.



< 코드 위치 : Coding Kit SPI → **spi_dotmat_heart_button** >

< 문법 설명 : **break 와 continue** >

바로 앞에서 배운 while 문과 for 문 등은 어떤 조건이 맞으면 계속해서 반복적으로 수행을 합니다. 이런 구문을 loop 문이라고 부릅니다. 이제 loop 문을 컨트롤하는 구문에 대해서 알아 보도록하겠습니다. loop 문을 끝내는 것은 조건식이 거짓이면 끝이 납니다. 그런데 조건식과 상관 없이 빠져나올 수 있는 방법이 있습니다. 그것이 이제 배울 break 입니다.

```
while (조건식1) {
 문장1;
 if (조건식2)
  break;
 문장2;
```

```
}
문장3;
```

위의 코드에서 조건식1이 참이면 while 문을 계속해서 반복적으로 수행을 합니다. 그러다가 if 문의 조건식2가 참이면 if 문 안의 break 문을 수행합니다. 이 break 문을 만나면 while 문 안의 나머지 코드를 수행하지 않고 바로 빠져나와 while 문 다음의 문장3을 수행합니다. 즉 loop 문 안에서 break 를 만나면 loop 문을 빠져나옵니다.

다음은 continue 문에 대한 설명입니다.

```
while (조건식1) {
  문장1;
  if (조건식2)
    continue;
  문장2;
}
문장3;
```

위의 코드에서 while 문을 반복적으로 수행하다가 if 문의 조건식2가 참이 되어 continue 를 수행합니다. 이 continue 는 그 다음의 문장을 수행하지 않고 while 문을 계속 반복하는 것입니다. 즉, continue 다음의 문장2 는 수행되지 않고 while 문을 계속 반복합니다. 다시 정리하면 조건식1(참) → 문장1 → 조건식2(거짓) → 문장2 → 조건식1(참) → 문장1 → 조건식2(참) → continue (문장2 수행 안 함) → 조건식1(참) → 문장1 … 이렇게 continue 는 loop 문을 빠져 나가는 것이 아니고 loop 문 안의 나머지 문장을 더 이상 수행하지 않고 loop 문을 반복하는 것입니다.

이제 예제를 보면서 확실히 이해해 보도록하겠습니다. 다음은 스탑워치(Stopwatch)를 구현한 코드입니다. 시작 버튼을 누르면 시간이 흐르기 시작하고 버튼 리셋 버튼을 누르면 리셋이 되서 시간은 다시 0 이 됩니다. 이 시간이 흐르는 것은 FND 에 표시가 됩니다. 스위치를 ON 하면 시간은 계속 흐르고 OFF 하면 시간은 흐르지 않습니다. 코드는 다음과 같습니다.

```
#define BUT_START   0
#define BUT_RESET   1
#define SWITCH_STOP 0

#define BUTTON_PRESSED 0
#define SWITCH_ON      1

int count = 0;
```

```
void loop() {
  if (ck_spiRdBut(BUT_START) == BUTTON_PRESSED) {
    while (1) {
      if (count >= 999 * 10)
        count = 0;

      fnd_on(count/10);

      if (ck_spiRdBut(BUT_RESET) == BUTTON_PRESSED) {
        count = 0;
        break;
      }

      if (ck_spiRdSwitch(SWITCH_STOP) == SWITCH_ON) {
        continue;
      }
      count++;
    }
  }
  fnd_on(count);
}
```

loop() 함수에서 시작 버튼이 눌렸는지를 체크합니다. 시작 버튼이 눌렸다면 count 변수를 증가합니다. 그런데, 999 가 되면 FND 는 더 이상 표시하지 못하므로 999 다음은 0 부터 다시 시작합니다. 그런데 자세히 보면 999 가 아니라 999 * 10 을 해서 9990 입니다. 왜 그랬을까요? 그것은 이렇해 주면 시간이 조금 천천히 흐르게 FND 에 표시가 됩니다. 그래서 그 아래에 FND 에 시간을 표시해 주는 fnd_on() 함수에 인자로 count 를 10 으로 나눈 값을 전달해 줍니다. 그러면 count 는 0 ~ 9990 까지 변하지만 FND 에는 이것보다 10 배 느리게 숫자가 표시됩니다. 나중에 곱하기 10 과 나누기 10 을 뺀 코드도 테스트 해 보십시요.

리셋 버튼이 눌렸으면 count 는 0 으로 리셋되고 break 문이 실행되어 while(1) 루프를 빠져 나갑니다. 그리고 붉으색의 fnd_on(count) 문장을 실행합니다. 이 때 세븐세그먼트에는 0 이 표시되면서 리셋이 되는 것입니다. 다시 loop() 함수의 처음을 시작하는데, 이 때 시작 버튼이 눌렸음을 다시 체크합니다. 시작 버튼이 다시 눌리면 while(1) 루프를 다시 돌면서 count 값은 올라가고 FND 에 표시됩니다. 이러던 중 멈춤 스위치(SWITCH_STOP)가 ON 이 됩니다. 그러면 이 때 count 문이 수행이 됩니다. 그러면 더 이상 그 아래의 count++ 문장은 수행되지 않고 while(1) 루프를 계속 반복합니다.

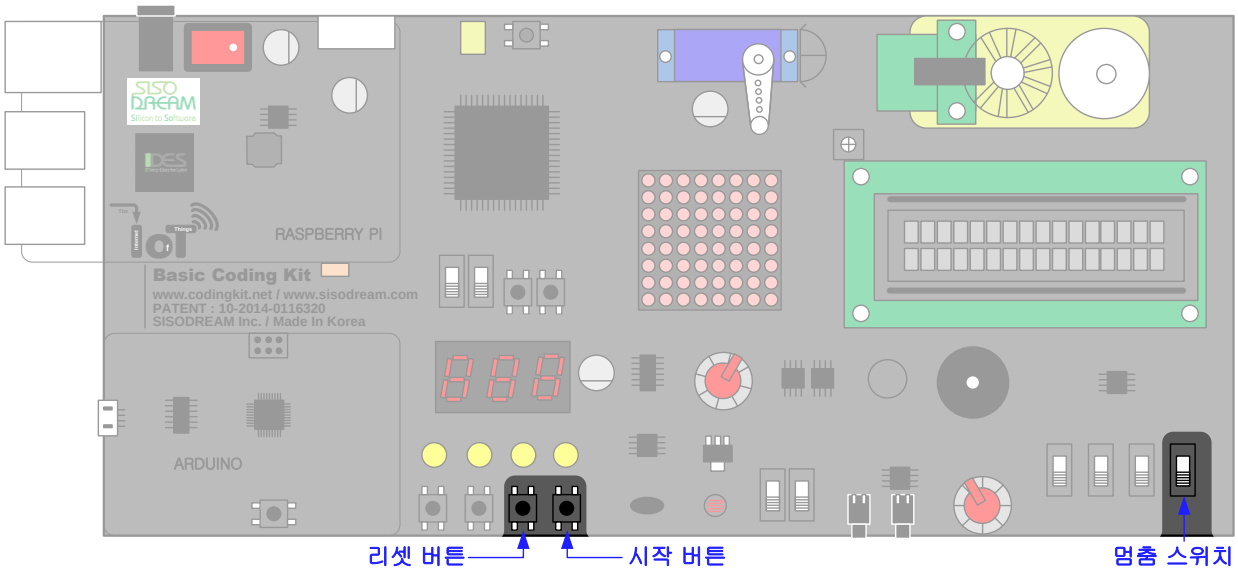
```
if (ck_spiRdSwitch(SWITCH_STOP) == SWITCH_ON) {
  continue;
}
count++;
```

그러면 FND 에 시간은 멈추고 더 이상 증가를 하지 않습니다.

정리하면 break 는 while(1) 루프를 완전히 빠져 나오는 것이고 continue 는 while(1) 루프는 빠져 나오지

않지만 자신 다음의 문장들은 더 이상 수행하지 않고 while(1) 루프를 계속 반복합니다. 이 break 와 continue 는 많이 사용되는 구문들이므로 예제와 함께 잘 알아 두십시요.

코딩키트에서 스위칭 모드는 A04 입니다. 다음과 같이 시작 버튼은 0 번 버튼이고 리셋 버튼은 1 번 버튼입니다. 멈춤 스위치는 0 번 스위치입니다.



< 예제 코드 : **break 와 continue 구문을 활용한 스탑워치** >

< 코드 위치 : Coding Kit SPI → **spi_fnd_stopwatch >**

< 문법 설명 : **키워드(Keyword)** >

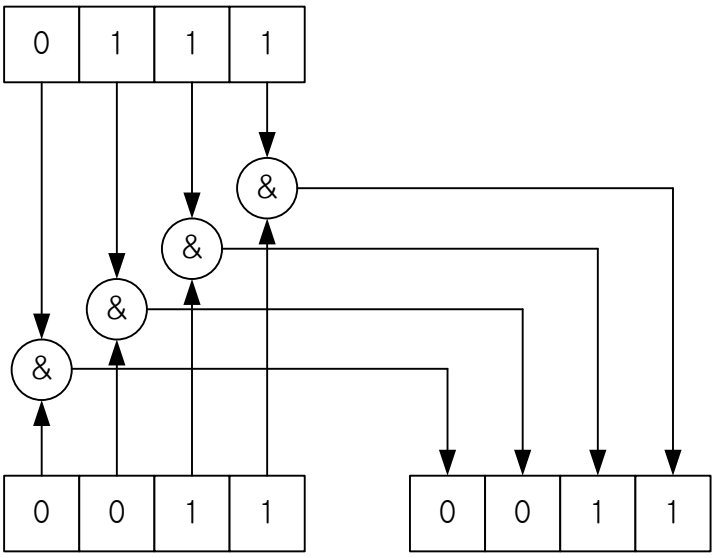
키워드라는 것은 프로그램 언어에서 미리 예약해 둔 단어를 말합니다. 이 키워드는 프로그램 언어에서 변수 이름이나 정의 등으로는 사용할 수 없습니다. 이 키워드의 예로는 if, for, while, case 등이 있습니다. 이 키워드는 미리 지정해 두었다고 하여 예약어라고도 합니다.

< 문법 설명 : **비트 연산자** >

비트 연산자(Bitwise Operator)는 논리 연산자와 매우 비슷합니다. 입력되는 값들을 AND, OR, NOT 을 합니다. 그런데 한 가지 차이는 논리 연산자가 전체 값을 하나로 사용하여 출력을 0 또는 1 로 한다면 비트 연산자는 각 비트들을 연산하여 여러 비트를 만듭니다. 나중에 논리 연산자와 비트 연산자를 비교하기로 하고 일단 비트 연산자는 다음 표와 같습니다.

비트 연산자	기호	설명	예제
AND 비트 연산	&	각 비트를 AND	0b0111 & 0b0011 = 0b0011
OR 비트 연산	|	각 비트를 OR	0b0111 | 0b0011 = 0b0111
XOR 비트 연산	^	각 비트가 다르면 1, 같으면 0	0b0111 ^ 0b0011 = 0b0100
NOT 비트 연산	~	각 비트를 NOT	~0b0111 = 0b1000

비트 연산자는 위의 표와 같이 AND, OR, XOR, NOT 이 있습니다. 다음 그림과 같이 각 비트의 자리를 잘 맞추어 계산을 합니다. 0 번 비트는 0 번 비트끼리, 1 번 비트는 1 번 비트끼리 자리를 잘 맞춥니다.



AND 비트 연산자는 모두 1 이면 1 이고 하나라도 0 이면 0 입니다. OR 비트 연산자는 모두 0 이면 0 이고 하나라도 1 이면 1 입니다. XOR 비트 연산자는 각 비트의 값이 다르면 1 이고 같으면 0 입니다. NOT 비트 연산자는 각 비트의 값을 뒤집습니다. 즉, 1 이면 0 이 되고 0 이면 1 이 됩니다.

그러면 논리 연산자와 차이는 무엇일까요? 여태까지 무심결에 논리 연산자를 사용하였는데, 이 비트 연산자와 차이가 있단 말이야? 이런 생각을 하시는 분이 계실 것입니다. 차이를 말씀 드리겠습니다. 논리 연산자는 전체의 값을 하나로 보고 출력을 0 혹은 1 을 출력합니다. 비트 연산자처럼 여러 값을 출력하는 것이 아니고 단지 0 혹은 1 만 출력합니다. 여기서 AND 논리 연산자의 진리표를 다시 보도록 하겠습니다.

A	B	A && B
false	false	false
false	ture	false
ture	false	false
ture	ture	ture

다음 표에서 ture 와 false 는 숫자로 보면 어떤 값일까요? false 는 항상 0 을 표시합니다. true 는 상황이

조금 다릅니다. 붉은색으로 된 A, B 의 ture 는 0 이 아닌 모든 값입니다. 즉, 1, 10, 99, 567, 103902, … 이렇게 0 이 아닌 모든 값이 ture 입니다. 하지만 오른쪽의 A && B 의 항목에 있는 검은색의 ture 는 1 을 말하는 것입니다. 1, 10, 99, 567, 103902, … 이런 숫자들이 아니고 그냥 유일하게 1 입니다. 위에서 논리 연산자는 0 혹은 1 값만 출력한다고 했습니다. 그래서 여기서 ture 는 1 인 것입니다. 제 말이 믿기지 않으시면 코드로 확인해 보세요. 이와는 달리 AND 비트 연산은 각각의 비트 값을 연산하는 것이기 때문에 결과 값이 0 과 1 만 나오는 것이 아니고 다양한 값이 나올 수 있습니다.

다음과 같은 예제를 참고 바랍니다.

```
0b0111 & 0b0011 = 0b0011 = 3
0b0111 && 0b0011 = 0b0001 = 1
0b0111 | 0b0011 = 0b0111 = 7
0b0111 || 0b0011 = 0b0001 = 1
~0b0111 = 0b1000 = 8
!0b0111 = 0b0000 = 0
0b0111 ^ 0b0011 = 0b0100 = 4
```

비트 연산자는 어떤 비트 위치에 1 이 있는지 확인하는 코드에 종종 사용됩니다. 다음과 같이 하면 2 번 비트에 1 이 있는지 확인할 수 있습니다.

```
if (val & 0b0100)
  문장1;
else
  문장2;
```

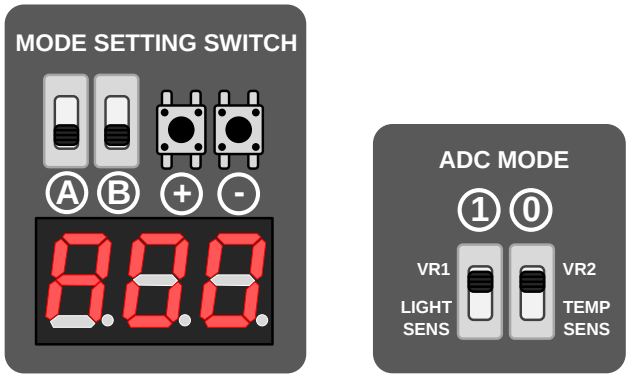
val 변수 값의 2 번 비트가 1 이면 if 문의 문장1 이 수행되고 2 번 비트에 1 이 없으면 else 문의 문장2 가 수행이 됩니다.

< 연습 문제 : **가변저항의 각 비트값에 따른 LED 켜기** >

이것을 이용해서 가변저항의 값중 0 번 비트가 1 이면 0 번 LED 가 켜지고, 1 번 비트가 1 이면 1 번 LED 가, 2 번 비트가 1 이면 2 번 LED 가 3 번 비트가 1 이면 3 번 LED 가 켜지는 예제를 작성해 보세요.

< 코드 위치 : Coding Kit Ex1 → **bitop_vr_led** >

코딩을 다 하셨으면 코딩키트에서 실행해 보시기 전에 스위칭 ID 는 A00 으로 해 주시고 ADC 모드 스위치 중 1 번은 위로 올려 주십시요.



비트 연산자는 각 비트를 마스킹(Masking)할 때도 주로 사용됩니다. 마스킹이란 어떤 값을 0 이나 혹은 1 로 강제로 만드는 것입니다. 다음 예제를 보시면 쉽게 이해하실 수 있을 것입니다.

```
0b1111 & 0b1100 = 0b1100
0b0111 & 0b1100 = 0b0100
0b0011 & 0b1100 = 0b0000
0b0001 & 0b1100 = 0b0000
0b0000 & 0b1100 = 0b0000
```

위의 예에서 붉은색의 0b1100 이 마스킹 비트입니다. 이 마스킹 비트와 AND 비트 연산을 하면 0 번과 1 번 비트는 항상 0 으로 만들어지고 2 번과 3 번 비트는 원래의 값으로 그대로 출력합니다. 이것은 0 번과 1 번 비트를 0 값으로 마스킹한 것입니다. 다음과 같이 하면 0 번과 1 번을 1 값으로 마스킹합니다.

```
0b1111 | 0b0011 = 0b1111
0b0111 | 0b0011 = 0b0111
0b0011 | 0b0011 = 0b0011
0b0001 | 0b0011 = 0b0011
0b0000 | 0b0011 = 0b0011
```

OR 비트 연산자를 사용하여 코딩하였습니다.

[부록 A : 스위칭(Switching) ID]

코딩키트는 스위칭 시스템을 이용하여 여러가지 연결을 설정할 수 있습니다. 이렇게 연결된 각각의 설정은 스위칭 ID 로 표시합니다. 다음은 코딩키트의 아두이노에서 스위칭 ID 에 따른 디바이스 연결 표입니다. 표에서 붉은색으로 표시된 숫자는 파란색으로 표시한 스위칭 ID 에서 각각의 디바이스에 연결된 아두이노의 핀 번호입니다. 여러분은 아두이노에서 이 번호로 코딩을 하시면 됩니다. 확인(√) 표시는 SPI 확장 모드로 연결하여 더 이상 핀 수는 소요되지 않지만 연결은 가능하다는 표시입니다. Device Control SPI 항목이 SPI 확장 모드를 의미합니다.

Arduino 디바이스	핀수	Switching ID 핀 번호						
		A00	A01	A02	A03	A04	A05	A06
LED 0	1	2				√	10	2
LED 1	1	3				√	11	3
LED 2	1	7				√	12	7
LED 3	1	8				√	13	8
Button 0	1	12		13	3	√		12
Button 1	1	13		A4	8	√		13
Button 2	1					√	2	6
Button 3	1					√	3	9
Buzzer	1	5				5	5	5
DC Motor Enable	1	6			4	8		
DC Motor Forward	1	10			5	3	6	
DC Motor Backward	1	11			6	6		
DC Motor Encoder Enable	1				7	7		
DC Motor Encoder Data	1				2	2		
Sound Sensor Detect	1	4				4		
IR Sensor LED	1	A5		A5		A5		4
IR Sensor Detect	1	A4				A4		
Servo Motor	1	9				9	4	
Dotmatrix	16		*1			√		
7-Segment	11			*2		√		
Device Control SPI	4					*4		
Character LCD EN, RS, Data	6				*3			
Character LCD Back Light	1							
Switch 0	1					√		10
Switch 1	1					√		11
Switch 2	1					√		A4
Switch 3	1					√		A5
Serial Monitor	2	0,1		0,1	0,1	0,1	0,1	0,1

위의 표에서 *1, *2, *3, *4 는 여러핀이 연결된 것을 나타내며 그 내용은 다음과 같습니다.

*1 : Dotmatrix	*2 : FND	*3 : Char LCD	*4 : SPI
DM_COL1 : 0	FND_A : 2	LCD_EN : 9	SPI_SS : 10
DM_ROW1 : 1	FND_B : 3	LCD_RS : 10	SPI_MOSI : 11
DM_COL4 : 2	FND_C : 4	LCD_D0 : 11	SPI_MISO : 12
DM_COL6 : 3	FND_D : 5	LCD_D2 : 12	SPI_SCLK : 13
DM_ROW5 : 4	FND_E : 6	LCD_D3 : 14	
DM_ROW6 : 5	FND_F : 7	LCD_D1 : 15	
DM_ROW4 : 6	FND_G : 8		
DM_ROW2 : 7	FND_DP : 9		
DM_COL3 : 8	FND_DIG1 : 10		
DM_COL5 : 9	FND_DIG2 : 11		
DM_ROW3 : 10	FND_DIG3 : 12		
DM_COL2 : 11			
DM_ROW7 : 12			
DM_ROW8 : 13			
DM_COL7 : A4			
DM_COL8 : A5			

아날로그 입력을 갖는 센서 및 가변저항은 고정된 핀에 연결됩니다. 이 해당 핀은 다음과 같습니다.

핀 번호	센서	가변저항	Note
A0	적외선 센서	X	-
A1	소리 센서	X	-
A2	온도 센서	가변저항 2	ADC 모드 스위치 0 번 : Up - 가변저항, Down - 센서
A3	밝기 센서	가변저항 1	ADC 모드 스위치 1 번 : Up - 가변저항, Down - 센서

디바이스가 어떤 핀에 연결되는지는 다음과 같이 정의하여 사용하시면 됩니다. 이 부분을 코드에 복사하시어 사용하셔도 됩니다. 그러면 편리할뿐만 아니라 코드의 에러도 줄일 수 있습니다.

[Release Note]

V1.0 : 2015. 8. 28

첫번째 버전 Release

V1.1 : 2015. 12. 9

(P.24) 라이브러리 폴더 이름

(P.25) 64 비트, 32 비트 운영체제에 따른 아두이노 폴더의 위치

(P.215 ~ P.217) 14 → A4, 15 → A5

V1.2 : 2016. 6. 12

(P.10) 코딩 보드 Short 버전 사진 추가

(P.30) 내용 추가

(P.41) 함수 이름 수정 : CK_InitLampOnDet() → CK_InitLEDOnDet()

(P.76), (P.103), (P.146) 오타 수정

V1.3 : 2016. 7. 7

(P.11) 20. 번을 "스위치 4 개" 로 변경

V2.0 : 2017. 6. 30

버전 2.0 출시
서버모터 범위 조정 - 소스코드 수정(servo_pwm_digital), 라이브러리 수정(Servo.h)