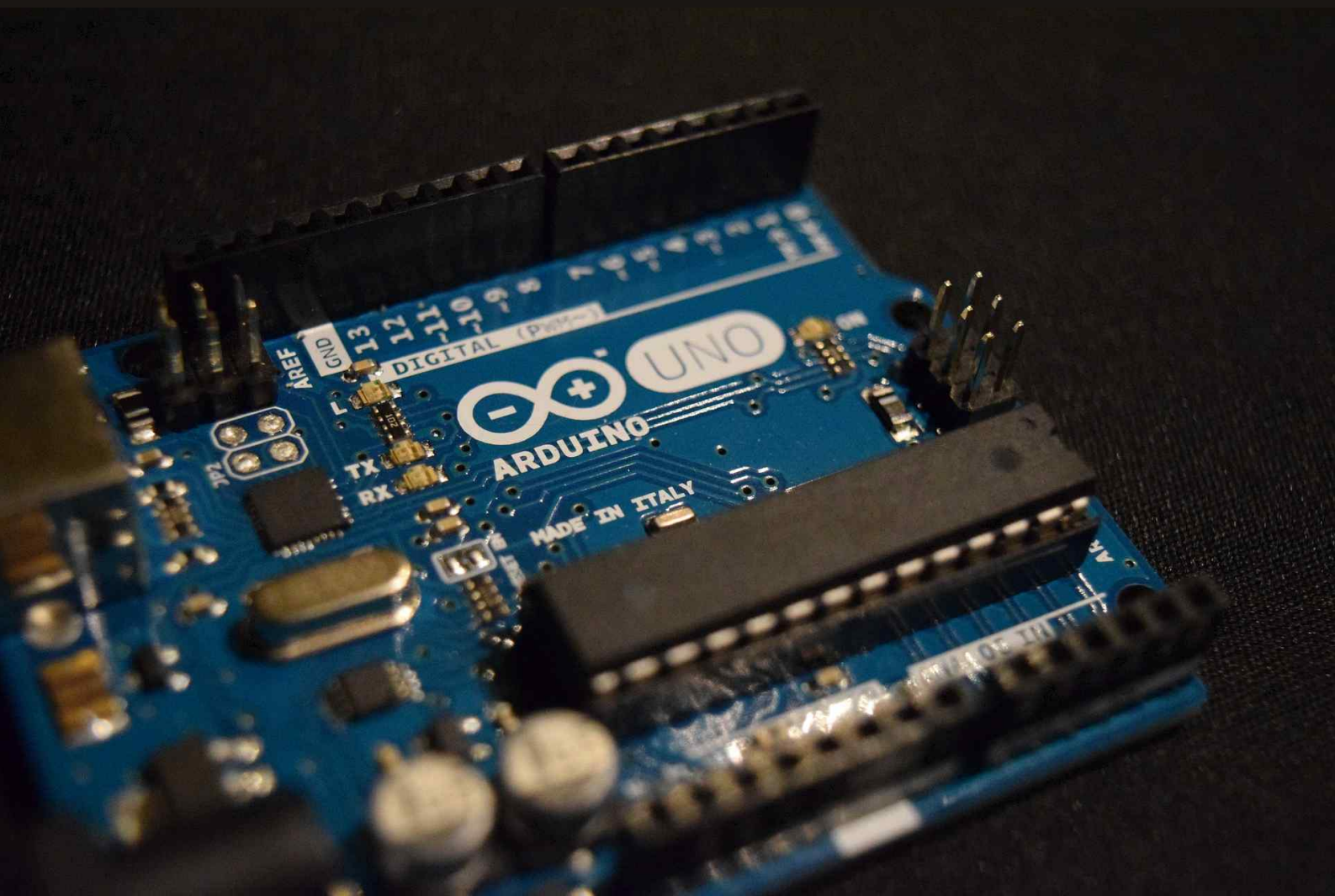
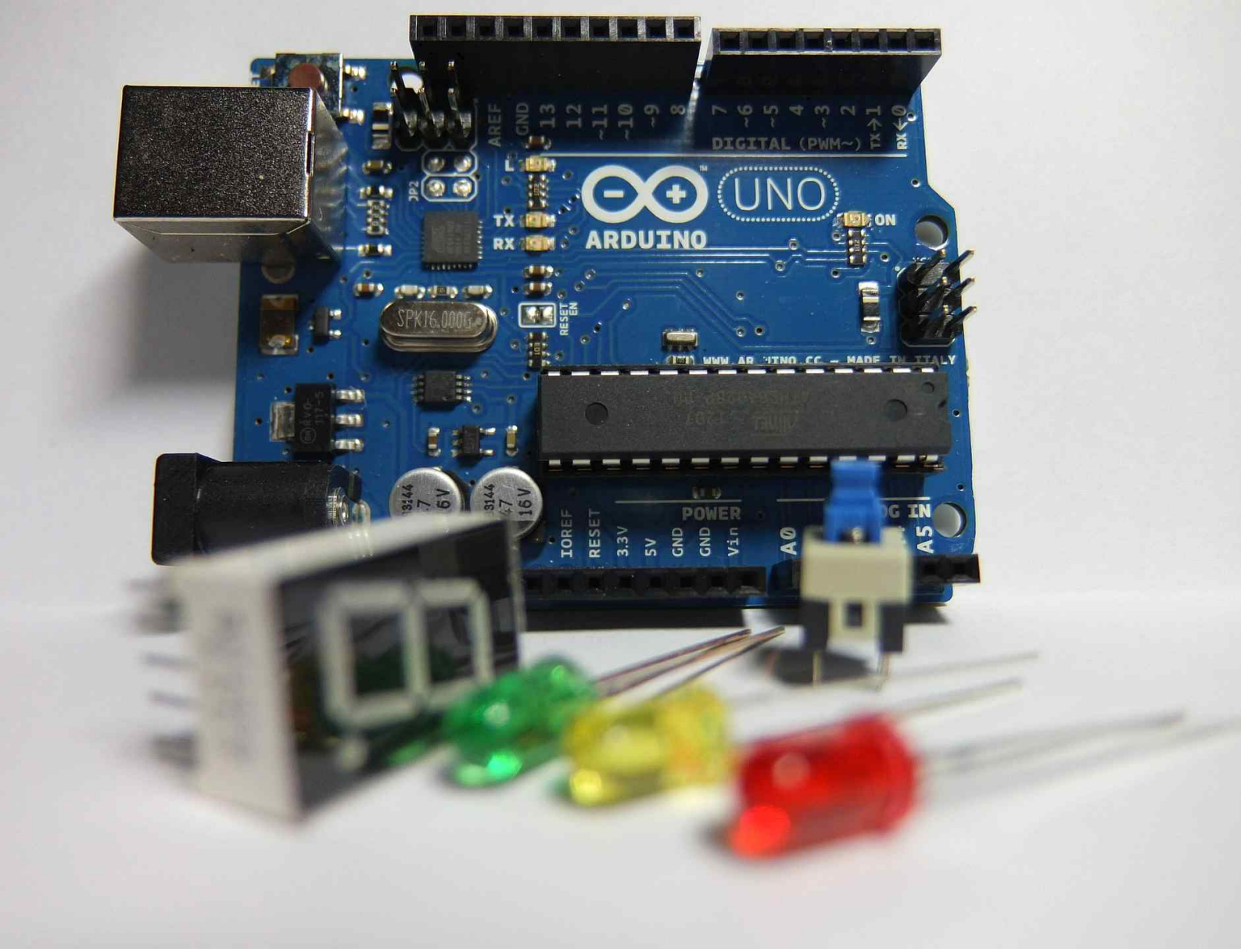
누구나 따라하는 아두이노 프로그램

- 초딩도 따라하는 아두이노 프로그램

C 언어를 처음부터 쉽게 공부하고 싶은 사람을 위한 무료 공개 강의 노트

저자 : 바람(eventia@gmail.com)





이 책의 저자는 "바람"입니다.

질문이 있으시면 아래 주소로 오셔서 질문하시면 이 책의 저자 "바람"이 시간날 때마다 답변해드리도록 하겠습니다.

인터넷 공개된 초판의 제작 시작일은 2016년 4월 29일입니다.

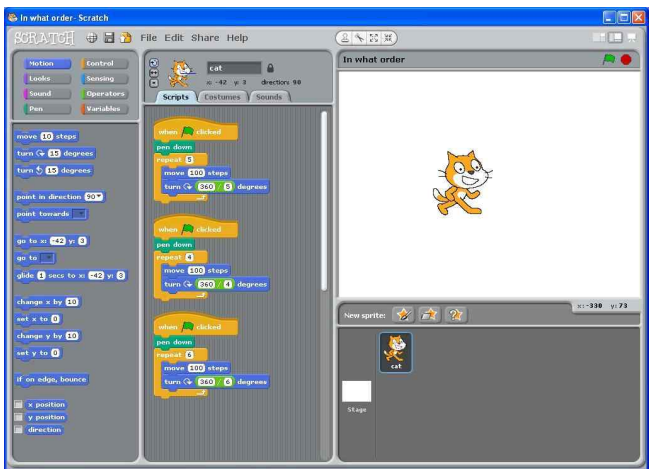
http://winduino.com

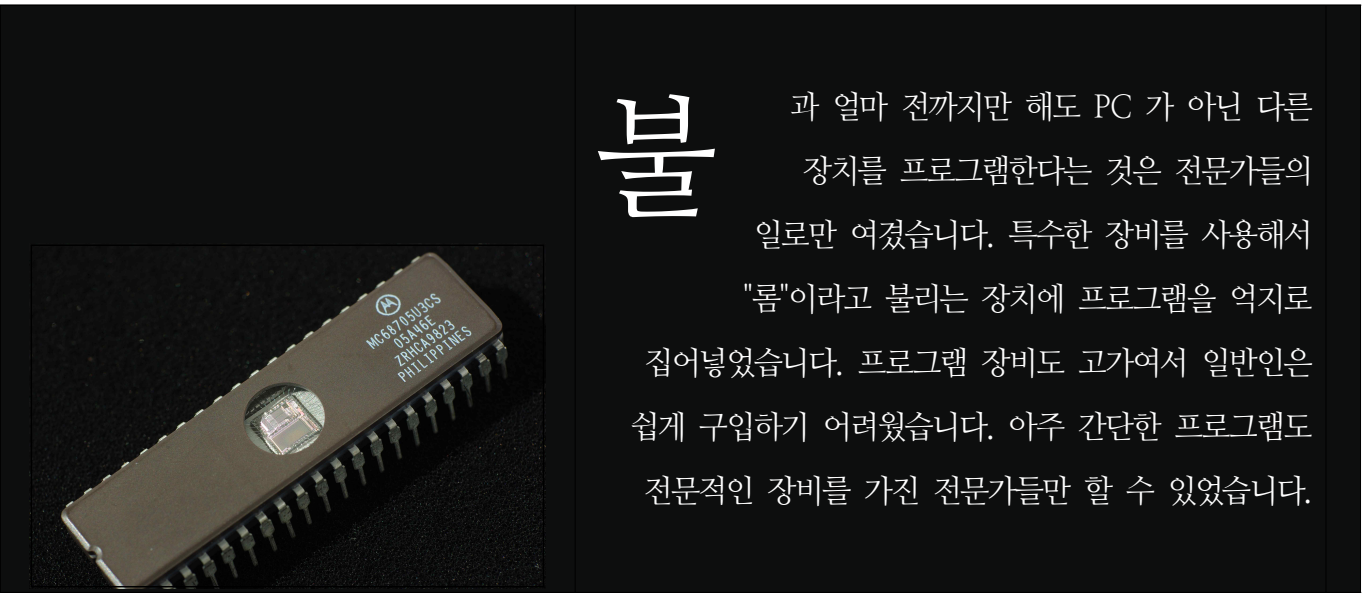
이 책의 목차

1장. 이 책의 사용설명서

1. 시작하면서

아두이노가 일반인들에게 알려지기 전에는 프로그램을 한다고 하면 보통 PC 에서 하는 프로그램을 의미했습니다. PC 에서 프로그램을 만들 때 사용하는 프로그래밍 언어는 많이 있습니다. 지금은 일반인도 프로그램할 수 있도록 쉬운 프로그램 언어들이 많이 나오고 있습니다. 스크래치같이 프로그램 언어는 레고 블록을 가져다 끼우는 식으로 프로그램을 할 수도 있습니다.





불과 얼마 전까지만 해도 PC 가 아닌 다른 장치를 프로그램한다는 것은 전문가들의 일로만 여겼습니다. 특수한 장비를 사용해서 "롬"이라고 불리는 장치에 프로그램을 억지로 집어넣었습니다. 프로그램 장비도 고가여서 일반인은 쉽게 구입하기 어려웠습니다. 아주 간단한 프로그램도 전문적인 장비를 가진 전문가들만 할 수 있었습니다.

하지만 아두이노는 이러한 흐름을 바꾸었습니다. 이제는 마이크로프로세서, 아두이노라는 것을 꽤 많은 사람들이 알고 있습니다. 그리고 꽤 많은 사람들이 그것을 다루어보고 싶어합니다.

여기서는 아두이노를 다루고 싶어하는 일반인을 대상으로 쉽게 프로그램할 수 있는 기초적인 프로그램공부를 하려고 합니다. 책은 문법적인 요소를 가능한 쉽게 설명하고 넘어갑니다. 간단한 예제를 통해 프로그램을 직접 하도록 했습니다. 이렇게 이 책의 끝까지 가면 적어도 스스로의 힘으로 아주 복잡하지 않은 아두이노프로그램이 가능할 것입니다.

이 책은 프로그램 실행속도를 조금 더 빠르게 하거나 조금 더 램을 적게 사용하는 것 같은 고급 기술을 가르치지 않습니다. 그 부분은 초급을 넘어서 중급 이상에서 다루어야 할 내용입니다. 아두이노를 처음 사용하는 것에 너무 많은 제약과 공부할 거리를 두지 않으려고 합니다. 우선 사용하면서 필요한 부분이 생기면 더 공부하면 됩니다. 더 공부하기 위해 우선 꼭 알아야만 할 것을 이 책은 다루고 있습니다.

더 많은 것을 해보고 싶다면 이 책을 넘어서서 더 많은 공부가 필요합니다. 그건 이 책을 모두 마친 다음에 스스로 판단하면 됩니다. 내가 원하는 것을 만들기에 충분하다면 굳이 더 깊은 공부를 하려고 발버둥칠 필요는 없습니다. 하지만 이정도로는 도저히 내가 원하는 것을 만들 수 없다면 그때는 이 책을 넘어서 심도 있게 프로그램을 더 깊이 공부하셔야합니다. 건투를 빕니다.

[추저리 추저리]

초등 대안학교를 다니는 아이가 있습니다. 제가 어릴 때 그랬던 것처럼 제 아이도 새로운 것을 보면 호기심을 주체할 수 없어합니다. 혼자서 일본 전자 잡지에 나온 Z80 보드를 만들어보겠다며 이리 저리 뛰어 다녔던 중학교 시절이 기억나면서 초등 4학년 아이에게 스크래치와 아두이노를 가르쳐 보았습니다.

이 책은 제 아이 같은 어린 초등학생 아이들도 공부할 수 있게 도움을 주려고 시작했습니다. 많은 것을 담기보다 가장 기초적인 것을 담으려고 했습니다. 하지만 역시 글을 쓰면서 많은 유혹을 받게 됩니다. 이것도 쓰고 싶고, 저것도 담고 싶어집니다. 하지만 아이를 가르치면서 제가 본 것은 아이는 배우기보다는 직접 자기의 손으로 하면서 더 많은 것을 알게 되는 것을 보았습니다. 가르치고 가르침 받은 것을 따라 하기보다는 궁금한 것을 질문하면서 오히려 더 빨리 지식을 습득하고 있었습니다.

2. 이 책의 사용법

아두이노를 처음 다루는 사람이라면 굳이 처음부터 끝까지 차분히 읽을 필요는 없습니다. 2장의 내용 설명이 이해가 되지 않아도 따라해 보세요.

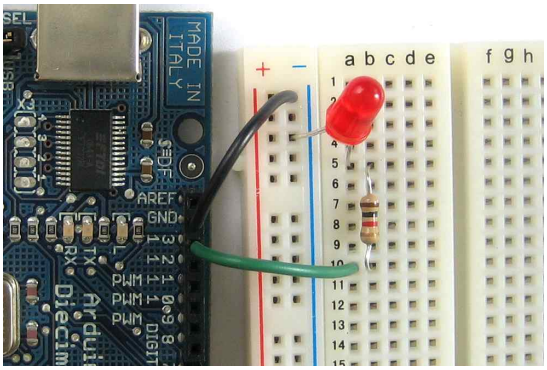
2장에서는 아두이노 IDE 설치부터 시작해서 LED 를 깜박거리는 가장 간단한 프로그램을 실행시킵니다. 불(LED)을 켜고 끌 수 있다면 불 대신 모터를 연결할 수도 있고, 다양한 장치를 사용할 수 있습니다.

[주저리 주저리]

초등 대안학교에서 다른 아이들도 스크래치와 아두이노를 그렇게 좋아했습니다. 대학생 이상, 전문가들의 것이라고만 알려졌던 MCU, 마이크로프로세서라는 말은 모르지만 그 아이들은 알아서 LED를 점멸시키면서 놀았습니다. Pc 에는 고양이가 뛰어다니고, 고양이가 발판을 밟으면 선풍기가 돌아가게 만들고 좋다고 낄낄댔습니다.

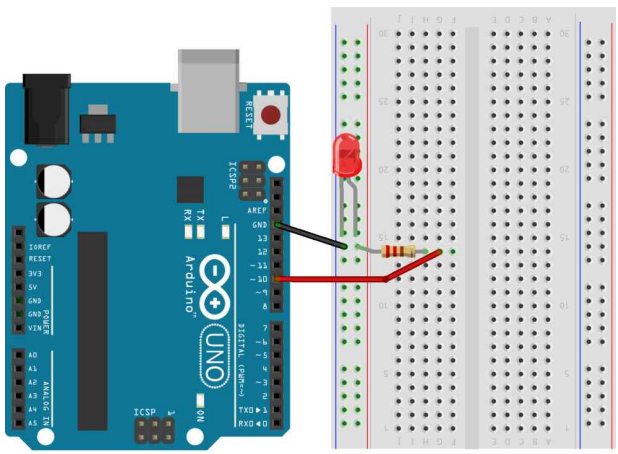
이 책은 초등학생들을 위한 글입니다. 이후로도 아이들을 가르치면서 필요한 내용과 일반인들을 위해 내용을 계속해서 공개하면서 무료로 배움을 가질 수 있도록 할 생각입니다.

이 책의 내용은 아두이노를 사용한 개인적 경험과 아두이노 홈페이지에 있는 다양한 예제들, 그리고 공개된 ebook인 arduino programming notebook을 참고했습니다. 영어로 된 arduino programming notebook 은 그 자체로 좋은 교재입니다. 영어만 익숙하다면 아이들이 직접 보아도 좋을 교재입니다.



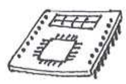
아두이노 보드에
LED 와 저항을 연결하여
불을 켜고 끌 수 있습니다.

fritzing 이라는 프로그램으로
그림을 그렸습니다.

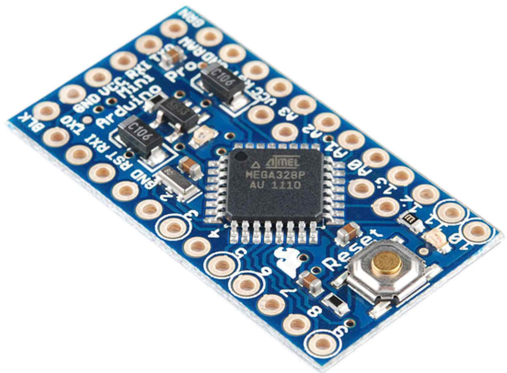


3. 아두이노와 프로그램

아두이노는 8비트 마이크로프로세서를 사용한 초소형컴퓨터입니다. 이 글을 쓰고 있는 2016년, 대부분의 가정에서 사용하는 컴퓨터는 CPU 속도가 아무리 느려도 1.6GHz 는 됩니다. 아두이노 우노의 속도가 16Mhz 인것과 비교해보면 100배 이상 차이가 납니다(1G = 1000M). 아두이노 우노의 램은 2k 입니다. 보통 컴퓨터의 램이 4G 에서 32G 인것과 비교하면 2000 배 이상 차이가 납니다(1M = 1000k). 쉽게 말해서 컴퓨터와 비교하면 아두이노는 매우 느리고 저장 공간도 매우 작다는 겁니다.

 PC	 Arduino
■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■■ 성능 100	■ 성능 1

하지만 이렇게 느린것 같아 보여도 아두이노는 꽤 많은 일을 처리할 수 있습니다. 사실 아두이노가 느리다는 것은 컴퓨터와 비교할 때 느리다는 것이지 실제로 우리가 처리하려는 일을 처리할 때 느리다고는 전혀 생각되지 않을 겁니다. 16MHz 라는 속도는 1초에 1천6백만개의 기본명령을 처리하는 속도입니다. 처음 PC 가 나왔을 때의 속도가 5MHz 가 채 되지 않았습니다. 이와 비교해서 느리지 않은 속도입니다. 아두이노가 느린 게 아니라 지금 PC 의 속도가 너무 빨라진 거죠.



가로와 세로가 43mm, 18mm 인
Arduino Pro mini 의 사진입니다.
이보다 더 작은 보드도 있답니다.

그럼 컴퓨터보다 느린 아두이노가 왜 필요할까요? 여러 가지 이유가 있습니다. 크기가 작아지고, 비용이 저렴해지고, 작은 전력으로 사용할 수 있기 때문입니다. 다양한 전자제품들안에 작은 미니 컴퓨터가 들어갑니다. 전기밥통 안에도 들어가고 세탁기 안에도 들어가고 에어컨 안에도 들어갑니다. 밥통에 들어가려면 크기가 작아야 하고, 가격도 저렴해야 합니다. 5만원이면 구입할 수 있는 전기밥솥에 가정용 PC 가

달려있다고 생각해보세요. 얼마나 크기도 커지고 가격이 비싸질까요? 아무리 커도 손바닥만한 크기의 미니컴퓨터가 필요한 이유입니다.

프로그램을 짜서 아두이노 안에 넣으며 아두이노는 작은 컴퓨터가 되어 간단한 동작을 알아서 하게 됩니다. 이 책은 아두이노를 사용하는 가장 기초적인 내용을 다루고 있습니다. 아두이노를 전혀 모르거나 기초가 전혀 없다면 이 책의 내용을 차근차근 보시면서 공부하시고 그 다음 자신이 원하는 것을 만들면서 다른 책을 공부하시면 됩니다.

영어로 책을 읽는 것이 가능하다면 무료로 공개된 아두이노 책들이 꽤 있습니다. 그 책들을 인터넷에서 검색해서 읽어보시기 바랍니다. 보통 구글에서 “free ebook arduino” 검색어로 검색하면 다 읽을 수 없을 만큼 많은 책들이 나옵니다. 안타깝게도 한글로 된 아두이노 책은 그렇게 쉽게 찾을 수 없습니다. 저작권이나 공유정신도 언제 한번 다뤄보고 싶습니다. 하지만 한국에서 이런 책을 쓰거나 프로그램을 만드는 사람들이 경제적으로 어려운 것도 현실입니다. 그래서 더 공유하지 못하고, 빈곤한 자료 속에서 계속해서 자신의 얼마 되지 않은 정보도 공개하지 못하는 악순환이 반복됩니다. 이 책 하나로 모든 것을 해결하지는 못하겠지만 앞으로 일반인들을 위한 자료를 계속해서 공유하기 위해 노력하려고 합니다.

4. 하드웨어와 소프트웨어

PC에서 프로그램을 하는 사람들은 프로그램에만 신경을 쓰면 됩니다. PC에는 키보드도 있고, 마우스도 있고, 모니터도 있고, 프린터도 있습니다. 프로그램을 짜면서 화면에 "안녕" 표시하고 싶으며 그냥 "안녕" 이라는 글씨가 나오게 프로그램을 하면 됩니다. LG 모니터를 쓸 때와 삼성 모니터를 쓸 때, 저렴한 모니터를 쓸 때 달라지지 않습니다. 똑같은 프로그램을 짜서 실행시키면 어떤 모니터를 사용하든지 상관없이 화면에 "안녕" 이라는 글씨가 나오게 됩니다.

하지만 아두이노는 조금 다릅니다. 아두이노에는 키보드도 없고, 마우스도 없고, 모니터는 더더욱 없습니다. 모니터같은 것을 달려고 알아보면 LCD 리는 것이 나옵니다. 그런데 이것이 한 줄짜리, 두 줄짜리, 네 줄짜리가 있습니다. 어떤 것은 그래픽LCD 라고 하면서 가로와 세로가 몇 개의 점이 찍히는지 그걸로 구분합니다. 아두이노와 LCD가 연결되는 방식은 또 다양합니다. 2줄짜리 I2C 라는 방식으로 연결되는 것이 있는가하면 어떤 것은 RS232 시리얼통신으로 연결됩니다. 4개의 데이터가 동시에 가는 4줄짜리 연결방식도 있고, 8개의 데이터가 동시에 가는 8줄짜리 연결방식(패러럴방식)도 있습니다. 뭐가 뭔지 머리만 아파집니다. 자 우선은 그런 걱정은 모두 버립시다. 더하기와 빼기도 못하는 아이가 미분방정식을 풀겠다고 대학수학책을 들여다보는 것은 칭찬할일이 아닙니다. 뭘 공부해야 할지를 모르고 있을 뿐입니다. 그때는 덧셈과 뺄셈을 공부할 수 있는 초등학교 저학년용 수학책과 연습장과 연필을 사주어야 합니다.

PC에서 프로그램을 할 때는 신경 쓸 필요 없던 것들이 아두이노 프로그램을 하려고 할 때는 신경을 쓰이게 합니다. 그래서 다른 사람은 잘 된다고 하는 프로그램소스를 그대로 가져와서 내 아두이노에 집어넣어서 실행시켰는데 잘 안 될 때가 자주 있습니다.

가끔 왜 안되는지를 묻는 글을 봅니다. "친구가 할 때는 됐는데 왜 안되죠?" 안되는 이유는 수 백 가지가 될 수도 있습니다. 하나하나 다른 점을 찾아보고 다른 점을 찾아서 고쳐가면서 실행해보면 언젠가는 될 겁니다. 아주 재수가 없으면 고장 난 아두이노를 붙잡고 시간만 버릴 수도 있고 생각지도 못하게 전원 어댑터가 맞지 않아서 그럴 수도 있습니다. 가능하면 아두이노와 주변 장치를 2개 사두세요.

아래에 나오는 모든 프로그램 예제를 실행시킬 때 반드시 전원연결과 나머지 주변장치들과의 연결을 확인하시기 바랍니다. 프로그램은 잘 실행이 되지만 LED 가 제대로 깜박이지 않는다든지 할 때 가장 큰 이유는 전원연결을 안했거나 배선을 잘못 연결한 경우가 아주 많습니다. 가끔 부품 이상이 원인일 수도 있지만 그런 경우는 경험상 100 번 중에 5번 이내입니다. 자 이제 시작해 봅시다.

2장. 초등학생도 따라할 수 있는 아두이노 따라잡기

1. 아두이노 IDE 설치

아두이노에 프로그램하기 위해서는 프로그램툴을 설치해야 합니다. 아두이노 공식 홈페이지로 가서 다운받은 후 설치를 진행합니다. 이 책에서는 윈도우환경에서 아두이노를 사용하는 것을 기준으로 설명하겠습니다. MAC 이나 LINUX 사용자들은 이미 이 책을 볼 필요가 없을 정도의 중급사용자로 판단됩니다. 그 정도 되면 알아서 아두이노 IDE 는 충분히 설치하실 수 있을 겁니다. 저는 PC 윈도우7, 윈도우10 에서 테스트하고, 라즈베리파이에서 아두이노 프로그램을 해봤습니다.

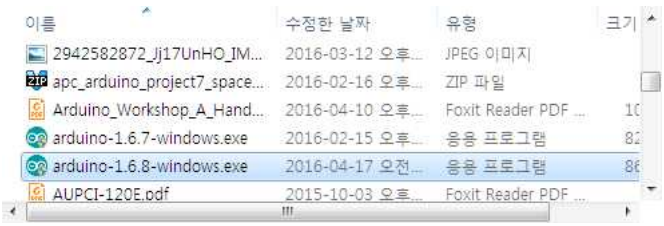
STEP 1. 아두이노 홈페이지에 방문해서 아두이노 프로그램(IDE)을 다운받습니다.

홈페이지 주소 : http://www.arduino.cc/

다운받을 프로그램 : https://www.arduino.cc/en/Main/Software

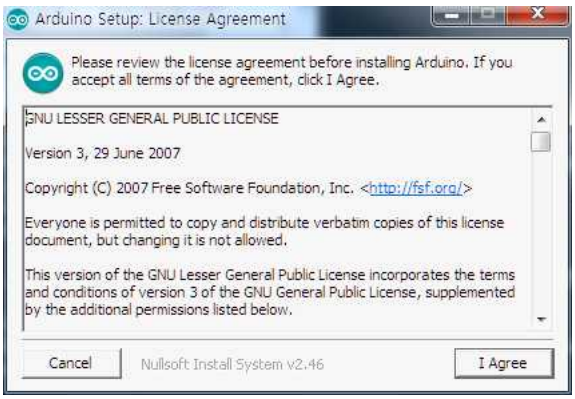
저 주소로 접속하면 다운받을 프로그램의 링크가 다시 나옵니다. 영어를 읽을 수 있으면 읽어서 그대로 따라하시면 됩니다. 아니면 아래에 있는 링크를 따라갑시다. 홈페이지가 수정될 때마다 그에 맞추어 아두이노 IDE 를 다운받는 방법을 자세히 설명해 두겠습니다.

STEP 2. 다운받은 프로그램을 PC 에 설치합니다.

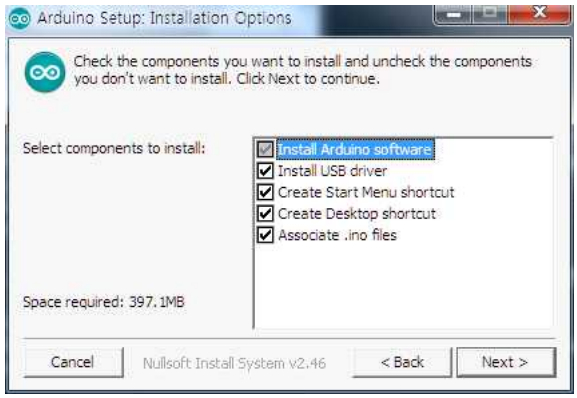


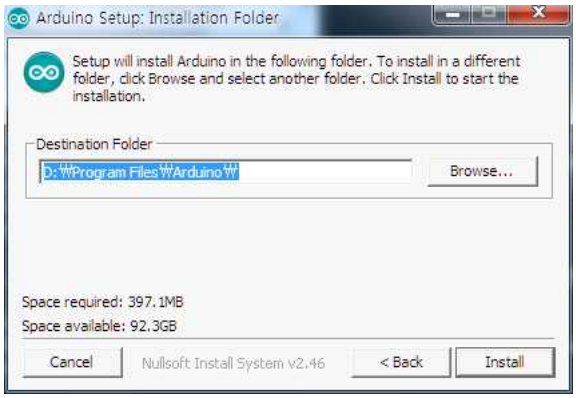
다운받은 파일은
arduino-1.6.8-windows.exe
입니다. 더블클릭해서 설치합니다.

[I Agree] 버튼을 클릭합니다.
설치가 시작됩니다.

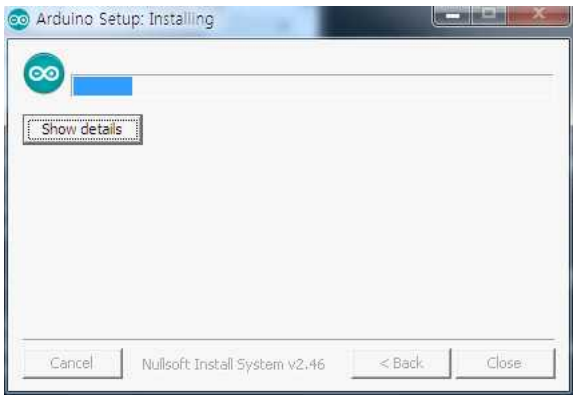


계속 진행합니다.
모두 선택해 주세요.

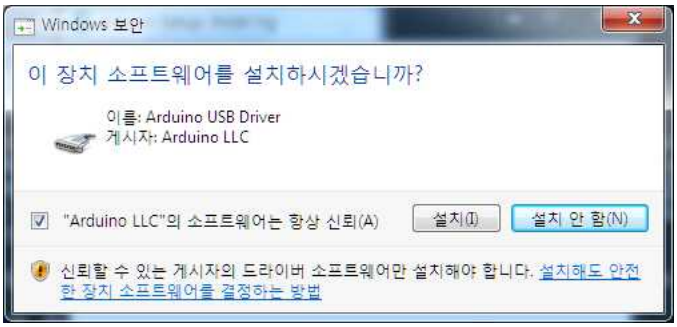




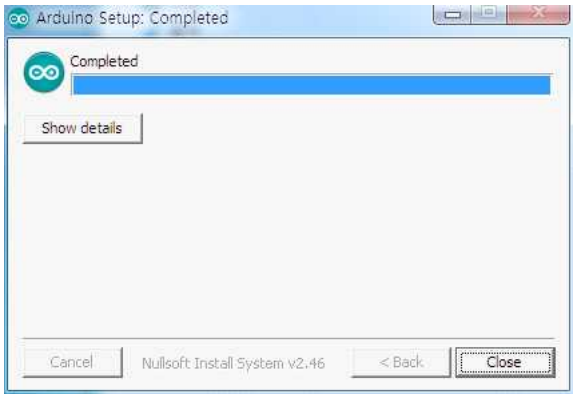
아두이노 IDE를 설치할
장소를 적으세요.



설치가 진행되고 있습니다.

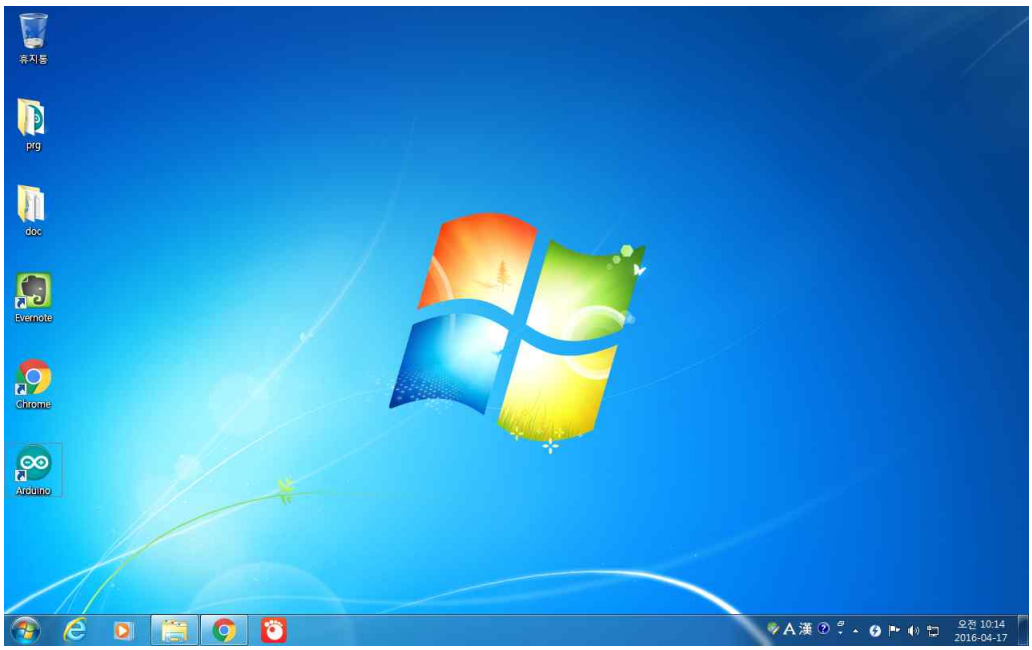


설치 중 USB 드라이버
설치를 묻습니다. 설치해 두세요.

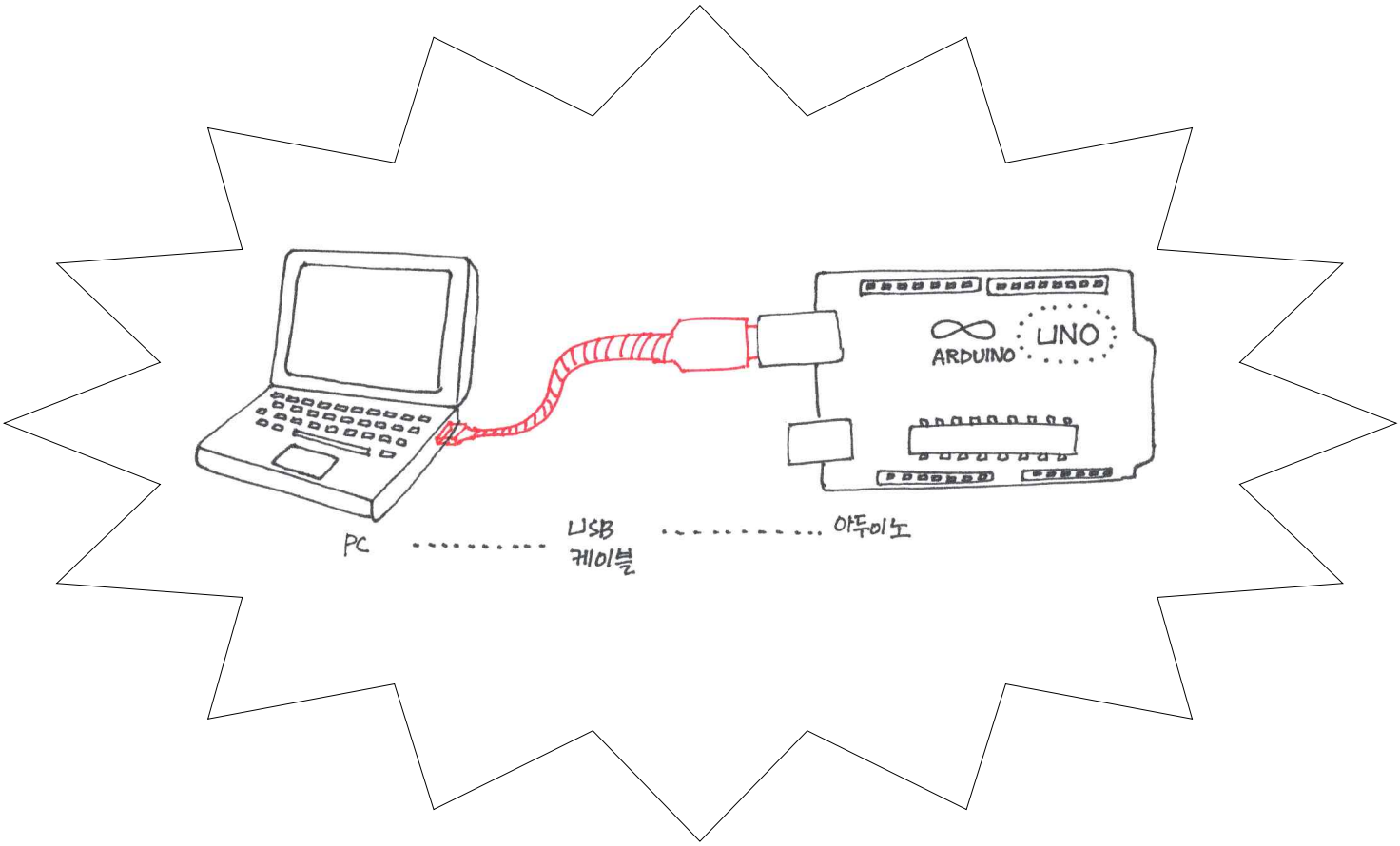


설치가 끝나면 Close를
선택해서 끝냅니다.

설치가 끝나면 바탕화면에 Arduino 아이콘이 설치됩니다.



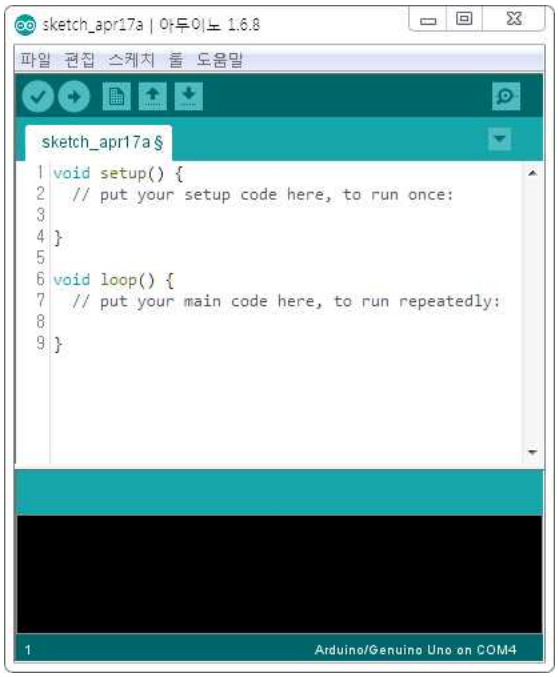
STEP 3. 아두이노와 PC를 USB 케이블로 연결합니다.



위 그림대로 연결하면 됩니다. 우선 PC 는 윈도우7 이상 (윈도우8, 윈도우10)을 기준으로 설명합니다. MAC 이나 리눅스를 사용하시는 분도 크게 다르지 않습니다만 별도의 설명은 여기서 하지 않겠습니다.

STEP 4. 설치된 아이콘을 더블클릭하면 IDE 가 실행됩니다.

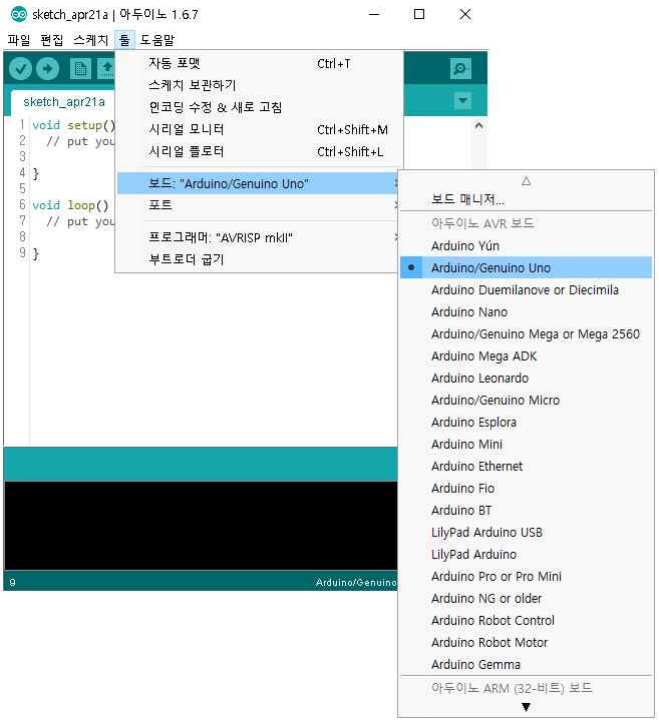
아이콘을 더블클릭하면 아두이노 IDE 가 실행됩니다.



STEP 5. IDE 내부에서 보드와 포트를 설정합니다.

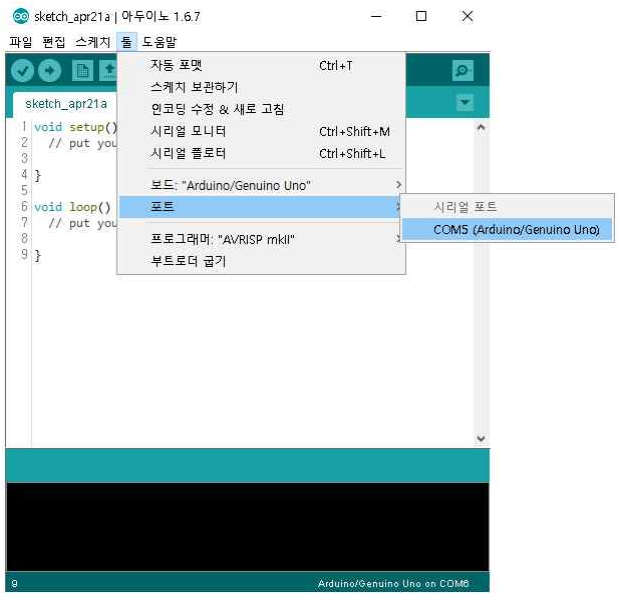
[툴] - [보드]

선택



[툴] - [포트]

선택



[툴] - [보드] 를 선택하고 그중 자신의 아두이노를 선택합니다. 아두이노 우노를 가지고 있다면 다음과 같이 Arduino/Genuino Uno 를 선택합니다.

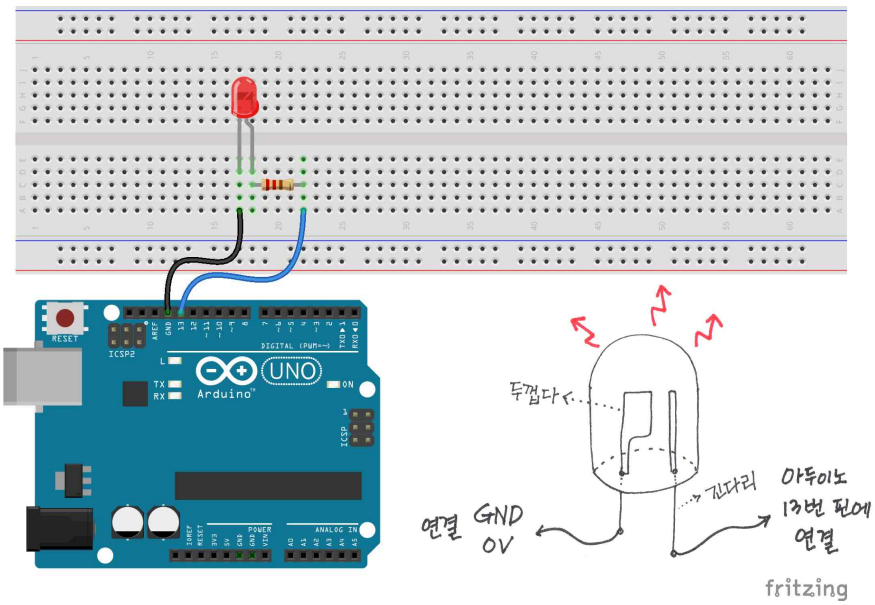
[툴] - [포트] 를 선택하고 자신의 아두이노가 연결된 포트를 선택합니다.

일반적으로 아두이노 IDE 를 설치할 때 아두이노 드라이버가 설치됩니다. 그래서 자동으로 이 부분에서 아두이노 포트가 보여집니다. 단, 저가의 아두이노 호환품 중 CH340 계열의 USB 칩을 사용하는 경우가 있는데, 그 경우 CH340 드라이버를 설치해 주어야 합니다. 구글에서 ch340 driver 를 검색한 후 찾아서 설치하시기 바랍니다. 자세한 내용은 홈페이지 http://electoy.tistory.com/211 을 참고하시면 됩니다.

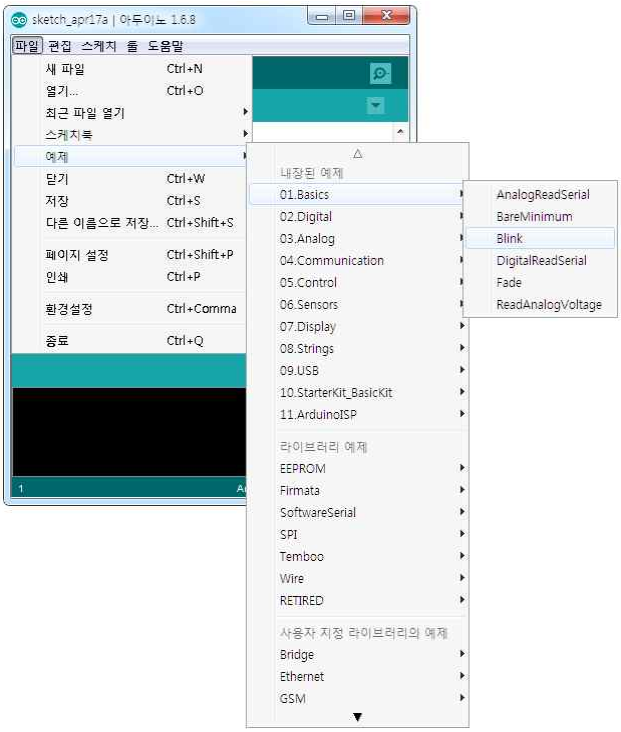
2. LED 켜고 끄는 프로그램 무조건 따라하기

아무것도 몰라도 괜찮습니다. 무조건 따라해 보세요. 무조건 따라했더니 LED 가 깜박이고 있다면 성공한 겁니다. 왜 그렇게 되는지 이해하는 것과 프로그램하는 방법을 다음 장에서 배우면 됩니다.

아두이노 우노를 준비하고 13번 핀과 GND 사이에 LED 와 저항을 다음 그림처럼 연결해 줍니다. LED 는 극성이 있습니다. 긴 다리와 짧은 다리를 구별해서 짧은 다리가 GND 에 연결되게 합니다. 혹시 다리를 잘라서 길이가 같아졌다면 원통 내부를 잘 보시면 한쪽이 두껍게 되어 있는 모습을 볼 수 있습니다. 두껍게 된 부분이 GND 에 연결됩니다.



아두이노 IDE 를 실행시킨 상태에서 **[파일]-[예제]-[01.Basics]-[Blink]** 를 선택해서 클릭합니다.



아두이노 우노를 PC 와 USB 케이블로 연결합니다.

[툴]-[보드]-[Arduino/Genuino Uno]를 클릭합니다.

[툴]-[포트] 에서 연결된 아두이노 우노의 포트를 선택합니다.

업로드를 위해 오른쪽화살표가 그려진 아이콘을 클릭합니다.

이제 아두이노 우노보드를 잘 보면 1초 간격으로 LED 가 깜박이는 것을 볼 수 있습니다.

여기까지 이상없이 진행되었다면 PC 에 문제가 없고, 아두이노에도 문제가 없고, 프로그램 컴파일도 잘되고 실행도 문제가 없는 것이 확인되었습니다. 이제부터 아두이노 프로그램을 본격적으로 시작해 보겠습니다.

3장. 아두이노 프로그램을 위한 쉬운 C언어 문법

1. 아두이노 기본 구조

아두이노 프로그램의 기본 구조는 2개의 몸통으로 구분됩니다. 첫번째 몸통은 실제 일을 하기 전 준비운동을 하는 단계입니다. setup() 이라고 하는 이름이 붙어있습니다. 두번째 몸통은 실제 일을 하는 단계입니다. loop() 라는 이름을 붙여두었습니다.

```
void setup()
{
    // 여기에 [준비운동]에 필요한 내용을 넣습니다.
}

void loop()
{
    // 여기에 [실제할 일]을 넣습니다.
}
```

여기 보면 setup() 과 loop()의 앞에 void 라는게 붙어 있습니다. 자세한 설명은 함수를 다룰 때 설명하겠습니다. 우선은 ***무조건*** 위의 모습을 기억해두세요.
[준비운동]과 [실제할 일]은 나눠집니다.

TIP

본래 C 언어는 main() 이라는 함수가 기본함수입니다.
main() 이라는 함수를 실행시키는 것이 C 언어입니다. main() 이라는 구조는 위에서 아래로 한번만 실행시키고 마치는 방식입니다. 그래서 그 안에서 for 나 while, if 등을 사용해서 프로그램을 구성합니다.

그런데 마이크로프로세서(이하 MCU)는 조금 다른 방식으로 프로그램 언어에 접근했습니다. 이를테면 MCU는 하드웨어를 준비하는 것은 한번이면 되지만 한번 시동이 걸린 다음부터는 전원이 꺼질 때까지 계속해야 하는 일을 합니다. 그래서 아두이노는 조금 특별하게 setup(), loop() 구조를 사용합니다. main() 안에 setup()을 한번 실행하고 하고 loop()을 무한 실행하게 두었습니다.

main() 함수와 setup(), loop() 의 관계는 다음과 같습니다. 지금은 이해가 안될 수도 있습니다. 그러면 그냥 넘어가세요.

```
main()
{
      setup();
      while(1) {
             loop();
      }
}
```

a. setup()

setup() 함수는 아두이노가 시작되는 처음단계에서 한번만 실행됩니다. 다음처럼 어떤 핀을 출력으로 사용한다면 먼저 그 핀을 출력으로 쓴다고 알려주는 역할로 사용됩니다.

```
1    void setup()
2     {
3        pinMode(13, OUTPUT);
4     }
```

여기서 절대 손대지 말아야 할 곳이 바로 첫째 줄과 둘째 줄, 그리고 맨 마지막 줄입니다. 첫째 줄은 setup() 함수를 시작한다는 뜻이고, 둘째 줄의 { 기호는 여기서부터 시작이라는 뜻입니다. 마지막 줄에 있는 } 를 만날 때까지 모든 것이 setup() 의 내용이 됩니다.

pinMode(13, OUTPUT);

pinMode() 함수는 아두이노의 핀을 입력으로 사용할 것인지 출력으로 사용할 것인지 선택하게 해줍니다. OUTPUT 은 출력으로 쓰겠다는 뜻입니다. 이제부터 아두이노의 13번 핀은 전류를 흘려서 바깥으로 내보낼 수 있습니다. 이 전류로 꼬마전구(LED)를 켜서 전기가 흐르는지, 흐르지 않는지를 볼 수 있습니다. 참고로 pinMode() 를 통해서 13번 핀을 출력으로 선택했다는 것이 13번 핀에 전류가 흐른다는 뜻은 아닙니다. 출력으로 잡혔고, 그 출력은 5V가 될 수도 있고, 0V 가 될 수도

있습니다.

항상 명령의 끝은 ; 로 마무리합니다. 즉, ";" 를 만나기 전까지는 프로그램은 아직 끝이 나지 않았다고 판단합니다. 처음 C언어 프로그램을 하는 사람들의 잦은 실수는 ; 를 제대로 사용하지 않는 것입니다.

setup() 은 처음부터 끝까지 한번만 실행하고 끝이 납니다.

b. loop()

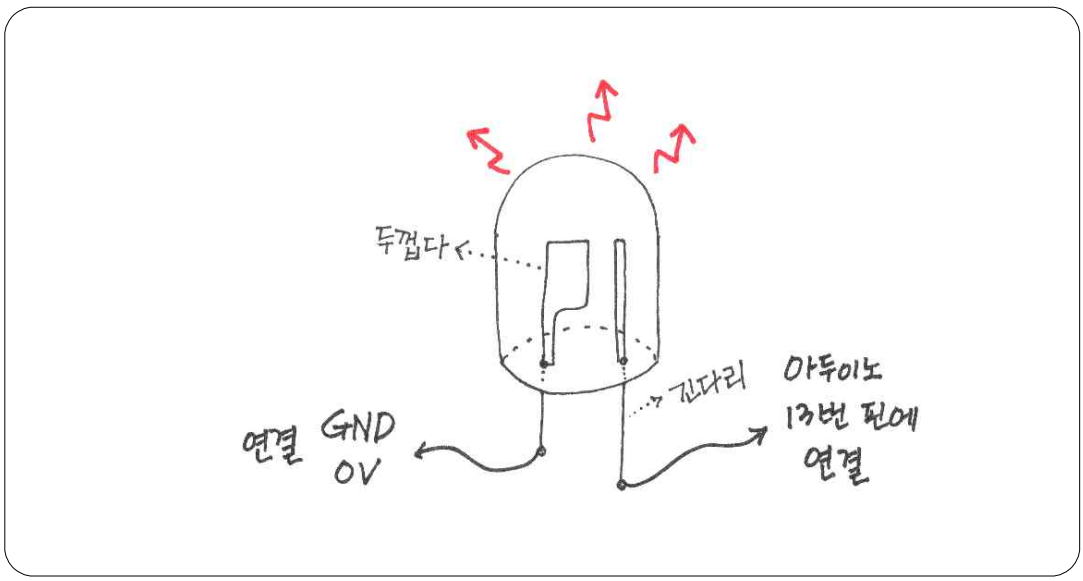
준비운동으로 13번 핀을 출력으로 만들었습니다. 그러면 13번 핀에 전기가 흐르게 합시다. 13번 핀에 꼬마전구(LED)를 달고 전기를 흐르게 해서 불이 들어오게 합시다. 이것은 loop() 에서 하면 됩니다.

```
void loop()
{
    digitalWrite(13, HIGH);
}
```

위에서 setup()을 통해 13번 핀을 출력으로 쓰겠다고 정했으니 loop() 에서는 그 13번 핀을 사용해서 전기를 흐르게 하든지 혹은 흐르지 않게 하든지 하면 됩니다. 아두이노의 13번 핀에 꼬마전구(LED)의 한쪽 발을 연결하고, 다른 한쪽 발은 0V에 연결합니다. 전기는 물과 같아서 높은 곳에서 낮은 곳으로 흐릅니다. 13번 핀에서 전기가 나와서 LED의 한쪽 발로 흘러들어가고, 그 전기가 다른 쪽 발로 나온 다음 0V 인 GND 로 흘러들어갑니다. 마치 산에 있는 물이 아래로 아래로 흘러 바다로 들어가듯이 전기는 이리저리 흘러 결국은 GND 로 흘러들어갑니다.

꼬마전구의 한쪽 발이 5V 에 연결되어있고, 다른 한쪽 발이 0V에 연결되어 있다면

5V에서 0V로 전기가 흐릅니다. 참고로 LED 라는 꼬마전구는 반도체입니다. LED 라는 반도체는 한쪽 방향으로는 전기가 흐르는 도체가 되지만 다른 반대 방향으로는 부도체가 됩니다. 다음 그림과 같이 잘 연결하세요. 반대로 연결하면 전기가 흐르지 않습니다. 전기가 흐르지 않으면 빛도 나지 않겠지요.



참고로 저항을 하나 연결해주면 더 좋습니다. 아두이노에서 나오는 전류가 약해서 LED 가 손상되지는 않습니다만, 아두이노에 직접 연결하는 경우가 아니라면 LED 에 너무 많은 전류가 흐르게 되면 LED 가 고장이 날 수 있습니다. 200 ohm 에서 500 ohm 사이의 저항을 하나 연결해 주면 좋습니다. 13번 핀과 LED 사이에 연결하거나 혹은 GND 와 LED 사이에 연결하거나 상관없습니다.

LED 와 저항 220Ω을 연결한 상태를 전기, 전자 기호로 나타낸 손 그림

digitalWrite() 함수는 ()안에 2개의 내용이 들어갑니다. 이걸 프로그래머들은 인자를 받는다고 표현합니다만, 굳이 어렵게 말할 필요는 없으니 앞으로 가능한 쉽게 풀어서 설명하겠습니다. **digitalWrite()** 안에 들어가는 2개의 내용은 첫째, 어떤 핀을 사용할까와 둘째 그 핀에 전기를 흐르게 할까 말까입니다.

13번 핀에 전기를 흐르게 하겠다(5V)면 13 과 HIGH 를 넣어주면 됩니다. 반대로 흐르지 않게 하겠다(OV)면 LOW 를 넣어주면 됩니다. HIGH 는 1 과 같고, LOW 는 0 과 같습니다. 일반적으로 전자회로를 다룰 때 1은 전기가 흐르는 것을 의미하고, 0은 전기가 흐르지 않는 것을 의미합니다.

loop() 는 처음부터 끝까지 실행한 다음 다시 처음으로 돌아갑니다. 이것을 끝없이 반복합니다. 아두이노에 전기가 공급되는 한 계속합니다.

그래서 위의 loop() 는 그냥 계속 13번에 연결된 LED 가 불이 켜진 채로 그대로 있습니다. 다음과 같이 바꿔봅시다.

```
void loop()
{
    digitalWrite(13, HIGH);
    digitalWrite(13, LOW);
}
```

이걸 실행하면 어떻게 될까요? loop() 는 처음부터 끝까지 실행한 후 다시 처음으로 돌아간다고 했으니까 13번 핀에 전기를 흘렀다(HIGH)가 그 다음에는 전기가 흐르지 않았다(LOW)를 계속 반복하게 됩니다. 그러면 LED 가 깜박일까요? 그냥 켜져 있습니다. 이유는 전기가 흐르고, 흐르지 않고를 반복하는 시간이 너무 짧기 때문에 눈에는 그냥 커져있는 것으로 보이는 것입니다. 나중에는 이걸 이용해서 LED 의 불빛의 밝기를 조절하는 것도 가능합니다.

그러면 깜박이게 하려면 어떻게 하면 될까요? 켜지고 꺼지는 사이에 충분히 눈으로 깜박거리는 것을 알 수 있도록 시간을 넣으면 됩니다. 아두이노에서는 기본적으로 `delay()` 라는 함수를 지원해줍니다.

`delay()` 라는 함수는 ()안에 mili second 인 1/1000 초의 배수를 넣게 됩니다. 즉, () 안에 1000 을 넣으면 1초 동안 아무것도 하지 않고 잠시 멈춰있게 됩니다. delay(“**시간**”) 는 주어진 “**시간**”동안 아무것도 하지 말고 정지해 있으라고 명령하는 것입니다.

```
void loop()
{
    digitalWrite(13, HIGH);
    delay(1000);
    digitalWrite(13, LOW);
}
```

어떻게 될까요?

3번째 줄에서 켜지고, 4번째 줄에서 1초 기다리고, 5번째 줄에서 다시 끕니다. 맨 처음으로 돌아갑니다.

깜박일 것 같지요? 하지만 실제로 실행해보면 깜박이지 않습니다. 계속 켜진 채로 있습니다. 왜일까요? 잠시 멈춰서 생각해 보세요. 이유는 다음 소스를 보면서 차이를 비교하면서 생각해보세요.

```
1  void loop()
2  {
3          digitalWrite(13, HIGH);
4          delay(1000);
5          digitalWrite(13, LOW);
6          delay(1000);
7  }
```

맨 아래에 delay(1000); 을 추가로 넣었습니다.

위에서부터 순서대로 실행 시켜 보겠습니다.

```
1  void loop()
2  {
3          digitalWrite(13, HIGH);
4          delay(1000);
5          digitalWrite(13, LOW);
6          delay(1000);
7  }
```

시작부분
    3번째 줄에서 **켭니다.**
    4번째 줄에서 **1초 기다립니다.**
    5번째 줄에서 **끕니다.**
    6번째 줄에서 **1초 기다립니다.**
시작부분으로 갑니다.

3번째 줄에서 켭니다.

4번째 줄에서 1초 기다립니다.

5번째 줄에서 끕니다.

6번째 줄에서 1초 기다립니다.

그리고 다시 3번째 줄, 프로그램의 처음 부분으로 갑니다.

첫 번째 소스도 이런 식으로 하나씩 순서대로 해석해 볼까요? 항상 프로그램은 위에서 아래로 하나씩 하나씩 순서대로 실행됩니다.

```
1  void loop()
2  {
3          digitalWrite(13, HIGH);
4          delay(1000);
5          digitalWrite(13, LOW);
6  }
```

시작부분
    켭니다.
    1초 기다립니다.
    끕니다.
시작부분으로 갑니다.

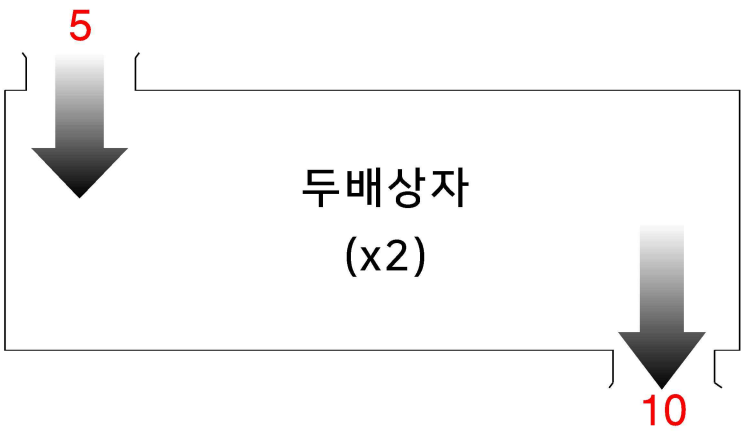
여기서는 켜고, **기다리고,** 끄고, 켜고, **기다리고,** 끄고... 이렇게 반복됩니다. 그런데 잘 보면 끈 다음 바로 켭니다. 즉, 꼬마전구를 끄자마자 바로 다시 켜버리는 것이죠.

이때 걸리는 시간은 매우 짧습니다. 너무 짧아서 꺼져있었다는 것을 사람의 눈이 인식할 수 없습니다. 이렇게 되면 꼬마전구는 눈으로 봐서는 계속 켜져 있는 것으로 보입니다. 처음에 원했던 깜박이는 동작을 하기는 하지만 눈으로 봐서는 알 수 없습니다. 그래서 맨 아래에 delay(1000); 을 추가로 넣었습니다. 그래야 끄고 다시 켜기 전에 1초를 기다리게 해서 꺼져 있는 상태를 눈으로 확인할 수 있게 됩니다.

```
1  void setup()
2  {
3      pinMode(13, OUTPUT);
4  }
5  void loop()
6  {
7      digitalWrite(13, HIGH);
8      delay(1000);
9      digitalWrite(13, LOW);
10     delay(1000);
11 }
```

c. 함수

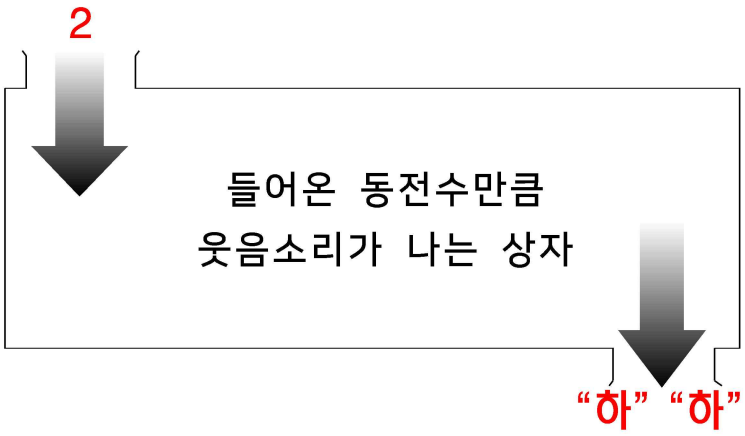
초등학교 교과서에 나오는 함수의 모습은 이렇습니다.



상자 그림이 있고 상자의 내부는 볼 수 없습니다. 이 상자는 들어가는 입구와 나오는 출구가 있습니다. 들어가는 입구로 어떤 수가 들어가면 나올때 2를 곱해서 나옵니다. 이 상자를 함수라고 하고, 이 함수상자의 이름은 "**두배상자**" 입니다.

함수는 기본적으로 들어가는 값이 있고, 나오는 값이 있습니다. 그리고 각각의 함수는 자신만의 이름이 있습니다. 여기까지가 초등학교 때 배웠던 함수의 기본입니다.

아두이노 프로그램에 사용되는 C언어도 함수라는 기능이 있습니다. 초등학교때 배운 함수와 비슷합니다. 조금 더 추가되었습니다. 들어오고 나가는 값 뿐만 아니라 함수는 어떤 특별한 행동을 할 수도 있습니다. 들어오는 값이나 나가는 값은 하나이거나 혹은 여러 개일 수 있습니다. 어떤 함수는 들어오는 값이나 나가는 값이 없습니다. 이를테면 "들어온 동전만큼 웃음 소리가 나는 상자" 와 같습니다.



위의 상자는 2 개의 동전을 넣으면 2번 "하", "하" 소리가 나는 상자입니다. 웃음소리를 듣고 싶으면 그만큼 동전을 넣으면 됩니다.

입력없이 출력만 있는 상자도 만들수 있습니다. 입력과는 상관없이 이 상자를 흔들기만하면 흔들때마다 10원씩 나오는 저금통같은 상자도 만들 수 있습니다.

게다가 입력도 출력도 없는 상자도 있습니다. 입력도 출력도 없지만 이 상자를 흔들면 음악소리가 나옵니다. 뮤직박스같은 상자입니다.

즉, 함수는 들어오는 것이 있고 나가는 것이 있는 것이 기본모습입니다. 하지만 위에서 설명한 것처럼 들어오는 것이 없을 수도 있고, 나가는 것이 없을 수도 있습니다. 나가는 것 혹은 들어오는 것이 없을 때 void 라는 단어를 씁니다. 함수 이름은 영어로 씁니다.

```
1 [나가는값] [함수이름] (들어오는 값들)
2 {
3     처리를 위한 명령문들...
4 }
```

이런 식으로 함수는 구성됩니다.

간단한 함수를 하나 만들어서 보겠습니다.

```
1 void laught()
2 {
3     Serial.println("HA HA");
4 }
```

입력으로 들어오는 것이 없기 때문에 **laught()** 에서 ()안은 비어있습니다.

출력으로 나가는 것이 없기 때문에 **void** 로 시작합니다.

함수 안에서 처리하는 것은 HA HA 라는 문장을 화면에 출력만 합니다.

아두이노 같은 C 언어를 사용하는 프로그램에서는 이렇게 만들어지는 함수를 사용합니다.

함수를 실행시킬 때는 함수의 이름을 쓰면 됩니다. 왼쪽에서 loop() 라는 함수 안에서 laught() 라는 함수의 이름을 쓰면 그 함수를 호출해서 실행시킵니다.

```
1  void setup() {
2    Serial.begin(9600);
3  }
4
5  void loop() {
6    laught();
7  }
8
9  void laught(){
10   Serial.println("HA HA");
11 }
```

프로그램의 실행 순서는 다음과 같습니다.

1 setup()을 실행합니다. 시리얼통신을 하도록 초기화합니다.

2 loop()을 실행시킵니다.

3 loop() 안에 있는 laught()를 실행시킵니다.

\# laught() 안에 있는 Serial.println("HA HA")을 실행합니다.

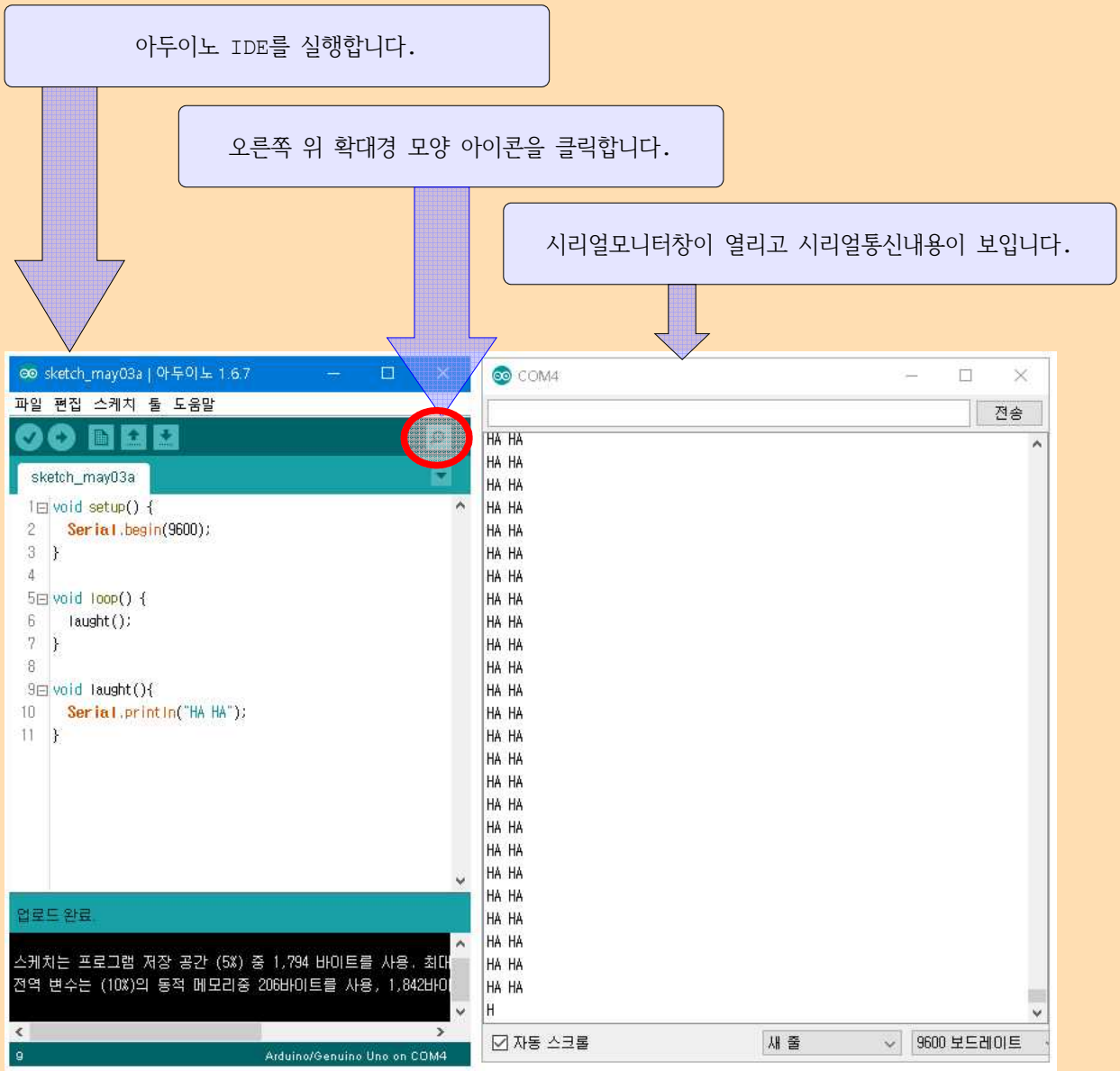
\# laught() 함수 끝까지 실행 한 다음 종료합니다.

\# laught() 가 종료되면 loop() 함수 6번 줄의 끝으로 갑니다.

\# loop() 가 종료됩니다.

4 다시 loop() 가 시작됩니다.

: 3번으로 갑니다. (# 표시된 줄을 계속 반복합니다.)


아두이노 IDE를 실행합니다.
오른쪽 위 확대경 모양 아이콘을 클릭합니다.
시리얼모니터창이 열리고 시리얼통신내용이 보입니다.
sketch_may03a | 아두이노 1.6.7
파일 편집 스케치 툴 도움말
sketch_may03a
1 void setup() {
2 Serial.begin(9600);
3 }
4
5 void loop() {
6 laught();
7 }
8
9 void laught(){
10 Serial.println("HA HA");
11 }
업로드 완료.
스케치는 프로그램 저장 공간 (5%) 중 1,794 바이트를 사용. 최대
전역 변수는 (10%)의 동적 메모리중 206바이트를 사용, 1,842바이
9
Arduino/Genuino Uno on COM4
COM4
전송
HA HA
H
자동 스크롤
새 줄
9600 보드레이트

d. { } 괄호의 의미

c 언어 문법에서 { } 괄호는 문단을 의미하게 됩니다. " { " 바로 다음이 시작이 되고, " } " 의 바로 앞이 끝이 됩니다. { } 로 둘러싸인 문단은 의미상 하나로 처리됩니다.

함수의 내용은 시작하는 { 과 끝나는 } 사이에 들어갑니다.

```
1 void laught()
2 {
3     Serial.println("HA HA");
4 }
```

e. 세미콜론 “ ; ”

세미콜론 “ ; ” 은 반드시 명령의 끝에 붙어야만 합니다. 한 줄에 명령을 하나만 써야한다는 제약은 없습니다. 여러 명령을 한 줄에 써도 되지만, 각 명령의 끝에 ; 를 붙여야 합니다.

초보자 뿐만 아니라 어느 정도 프로그램을 많이 한 사람들도 ; 를 빠트리는 실수를 합니다.

```
1  void loop()
2  {
3     digitalWrite(13, HIGH);
4     delay(1000);
5     digitalWrite(13, LOW);
6     delay(1000);
7  }
```

```
1  void loop()
2  {
3     digitalWrite(13, HIGH);  delay(1000);
4     digitalWrite(13, LOW);   delay(1000);
5  }
```

위의 왼쪽과 오른쪽은 완벽하게 같습니다. 때로는 읽기에 편하도록 하기 위해서 오른쪽처럼 쓰기도 합니다.

2. 주석

a. 여러줄 주석

프로그램을 하다보면 설명을 넣고 싶은 곳이 있습니다. 설명은 나중에 다시 이 프로그램을 살펴볼 때 예전에 했던 일을 왜 이렇게 했는지 기록하는 것입니다.

/* 이렇게 시작하고 */ 이렇게 끝납니다. /* 과 */ 사이에 있는 모든 것은 눈에는 보이지만 프로그램에서는 없는 것으로 간주합니다.

즉, 주석은 프로그램과 아무런 상관이 없습니다. 오직 주석은 나중에 소스를 보는 사람을 위한 설명입니다. 한달 후에 혹은 일년 후에 내가 만들었던 소스를 다시 수정하거나 비슷한 다른 프로그램을 만들려고 할 때 내가 만들었지만 생소한 기분을 느낍니다. 그럴 때 적어둔 설명(주석)을 보면서 내가 이 부분을 왜 이렇게 만들었는지 다시 알게 해 주는 기능입니다.

```
1  /*
2      이곳은 설명을 위한 주석입니다.
3      이곳에 쓰는 모든 것은 볼 수는 있지만
4      프로그램과는 무관합니다.
5  */
```

b. 한줄 주석

간단하게 한 줄 짜리 설명을 붙이고 싶을 때 설명의 앞에 // 를 넣으면 // 이후에 줄의 끝까지 나오는 모든 것은 주석이 됩니다. /* */ 과는 달리 줄의 끝 부분이나 혹은 한 줄로 설명할 수 있는 간단한 주석을 달 때 사용됩니다.

```
1   // 이 줄은 한줄짜리 설명입니다.
```

```
/*
   LED_BLINK.ino

   13번 핀에 달린 LED를 1초 간격으로 점멸하는 프로그램
   digitalWrite() 와 delay() 함수를 사용함
*/

void loop()
{
    digitalWrite(13, HIGH);   // 13번 핀을 HIGH 상태로 만든다
    delay(1000);              // 1초 간 딜레이를 준다
    digitalWrite(13, LOW);    // 13번 핀을 LOW 상태로 만든다
    delay(1000);              // 1초 간 딜레이를 준다
}
```

```
void loop()
{
    digitalWrite(13, HIGH);
    delay(1000);
    digitalWrite(13, LOW);
    delay(1000);
}
```

위의 주석 달린 소스와 아래에 주석을 제거한 소스는 아두이노에게는 완벽하게 동일하다. 단, 위의 소스처럼 주석을 적절히 적어 놓으면 나중에 시간이 지난 뒤 다시 프로그램을 수정하거나 소스를 사용할 때 도움이 됩니다.

3. 변수

변수라는 것은 "변할 수 있는 수"로 어떤 값을 저장할 수 있는 공간을 의미합니다. 이를테면 빈 박스와 같습니다. 이 박스에는 사과도 담을 수 있고, 배도 담을 수 있습니다. 어떤 박스는 크기가 작아서 계란 하나만 담을 수 있습니다. 또 어떤 박스는 플라스틱 밀폐용기로 되어 있어서 물을 담을 수 있습니다.

변수는 컴퓨터 메모리에 차지하는 공간을 의미합니다. 거기에 작은 숫자만 저장할 수 있는 변수도 있고, 큰 숫자를 저장할 수 있는 변수도 있습니다.

변수를 쓰기 위해서는 먼저 변수를 쓰겠다고 알려야 합니다. 이걸 '선언' 이라고 하는데, 어려운 용어를 기억할 필요는 없습니다.

```
int value;
```

이렇게 value 라는 변수를 쓰겠다고 알리고 만드는 것이 사용하는 것보다 먼저 있어야 합니다.

```
int value;
value = 13;
```

이렇게 만들어진 변수 value 에 원하는 값 13 을 넣었습니다. 이제부터 value 라는 상자를 들여다보면 거기엔 13 이라는 값이 담겨있게 됩니다.

a. 변수 선언

변수 선언은 변수를 쓰기 전에 반드시 먼저 해야 합니다.

```
int value;
```

처럼 쓰기 전에 변수를 만들어야 합니다.

```
int value = 120;
```

처럼 변수를 만들면서 변수의 값을 넣어주어도 됩니다.

변수의 종류는 **byte, char, int, long, float** 외에도 몇 가지가 더 있습니다. char 은 문자를 다룰 때, int 는 정수형 숫자를 다룰 때, long 은 정수형 숫자로 큰 수를 다룰 때, byte 는 작은 숫자로 양수만 다룰 때, float 은 실수형 숫자를 다룰 때 사용합니다. 변수의 사용법은 이후에 다루도록 하겠습니다.

b. 전역 변수, 지역 변수

위에서 선언된 변수는 어디에 있느냐에 따라 전역변수 또는 지역변수로 다시 분류됩니다. 함수 안에서 만들어졌으며 지역변수라고 불리고 그 함수 안에서만 사용이 됩니다.

```
1 void func()
2 {
3    int i = 10;
4    Serial.println(i);
5 }
```

그리고, 함수가 끝나면 지역변수는 사라집니다. 위에서 func() 함수 밖에서 i 를 사용하면 에러가 나옵니다.

반면, 전역변수는 함수 밖에서 만들어집니다.

```
1 int i = 10;
2 void func()
3 {
4    Serial.println(i);
5 }
```

함수가 끝나도 함수 밖에서 만들어진 전역변수는 계속 남아있습니다.

지역변수는 함수 안에서만 사용되므로 같은 이름이 다른 함수 안에서 새롭게 만들어지고, 사용될 수 있습니다.

```
1  void func_i()
2  {
3     int i = 10;
4     Serial.println(i);
5  }
6
7  void func_j()
8  {
9     int i = 20;
10    Serial.println(i);
11 }
```

전역변수는 사용에 주의해야 합니다.

전역변수와 지역변수가 사용되는 곳을 알아봅시다.

```
1  int globalvalue;
2  void setup(){
3    Serial.begin(9600);
4  }
5
6  void loop(){
7     int b;
8     for (int a=0; a<10; a++) {
9           Serial.println(a);
10       }
11 }
```

전역변수인 **globalvalue** 는 어디서나 사용할 수 있습니다. 지역변수인 **a** 는 for 문 안에서만 사용할 수 있습니다. 역시 또 다른 지역변수인 **b** 는 loop() 안에서만 사용할 수 있습니다.

```
    void loop(){                                    // 1
        int b=10;                                   // 2
        for (int a=0; a<10; a++) {                  // 3
              Serial.print("a=");                   // 4
              Serial.println(a);                    // 5
          }                                         // 6
          Serial.print("a=");                       // 7
          Serial.print(a);                          // 8
          Serial.print("  b=");                     // 9
          Serial.print(b);                          // 10
    }                                               // 11
```

위의 예제를 실행하면 8번째 줄에서 에러가 납니다. 변수 a 는 3번째 줄, for 문에 포함되어 있습니다. for 문의 { } 문단 안에서만 변수 a 는 사용할 수 있습니다. 즉, 3번 줄에서 6번 줄을 벗어나서는 a를 쓸 수 없습니다. 억지로 사용하려고 하면 에러를 발생합니다.

여러 함수 안에서 변수의 값을 읽거나 쓰려면 전역변수를 사용하면 됩니다. 물론 그 방법 말고도 다른 좋은 방법이 있습니다만 여기서는 전역변수를 사용하는 방법을 알려드립니다.

```
int globalvalue;                                    // 1
void setup(){                                       // 2
    Serial.begin(9600);                             // 3
    globalvalue = 10;                               // 4
}                                                   // 5
void loop(){                                        // 6
    int b=10;                                       // 7
    globalvalue++;                                  // 8
        Serial.print("globalvalue=");                // 9
        Serial.print(globalvalue);                  // 10
}                                                   // 11
```

함수 외부에서 선언된 전역변수 globalvalue 는 어떤 함수 안에서도 사용이 가능합니다. globalvalue 를 setup() 함수에서 초기 값을 준 뒤, loop()함수에서 반복될 때마다 1씩 증가시켜보았습니다. 이처럼 전역변수는 여러 함수에서 읽고 쓸 수 있습니다.

c. 변수의 종류

byte

byte 변수는 0 부터 255 까지의 정수 숫자를 저장할 수 있습니다. byte 변수는 1개의 바이트를 사용합니다. 1 바이트는 8개의 비트의 모음입니다. 일반적으로 아두이노 같은 마이크로프로세서는 8비트, 즉 1바이트를 기본단위로 합니다. 1비트는 전기가 흐르거나 흐르지 않는 하나의 정보를 보관할 수 있는 장소입니다. 1 비트는 0 또는 1을 보관할 수 있는 공간입니다. 정보가 보관된 곳을 1로 표시하고 비어있는 것을 0으로 표시하면 다음과 같습니다.

이진수	십진수	계산
0000 0000	0	$= 0\times2^7+ 0\times2^6+ 0\times2^5+ 0\times2^4+ 0\times2^3+ 0\times2^2+ 0\times2^1+ 0\times2^0$
0000 0001	1	$= 0\times2^7+ 0\times2^6+ 0\times2^5+ 0\times2^4+ 0\times2^3+ 0\times2^2+ 0\times2^1+ \mathbf{1}\times2^0$
0000 0010	2	$= 0\times2^7+ 0\times2^6+ 0\times2^5+ 0\times2^4+ 0\times2^3+ 0\times2^2+ \mathbf{1}\times2^1+ 0\times2^0$
0000 0011	3	$= 0\times2^7+ 0\times2^6+ 0\times2^5+ 0\times2^4+ 0\times2^3+ 0\times2^2+ \mathbf{1}\times2^1+ \mathbf{1}\times2^0$
0000 0100	4	$= 0\times2^7+ 0\times2^6+ 0\times2^5+ 0\times2^4+ 0\times2^3+ \mathbf{1}\times2^2+ 0\times2^1+ 0\times2^0$
…	…	…
1111 1101	253	$= \mathbf{1}\times2^7+ \mathbf{1}\times2^6+ \mathbf{1}\times2^5+ \mathbf{1}\times2^4+ \mathbf{1}\times2^3+ \mathbf{1}\times2^2+ 0\times2^1+ \mathbf{1}\times2^0$
1111 1110	254	$= \mathbf{1}\times2^7+ \mathbf{1}\times2^6+ \mathbf{1}\times2^5+ \mathbf{1}\times2^4+ \mathbf{1}\times2^3+ \mathbf{1}\times2^2+ \mathbf{1}\times2^1+ 0\times2^0$
1111 1111	255	$= \mathbf{1}\times2^7+ \mathbf{1}\times2^6+ \mathbf{1}\times2^5+ \mathbf{1}\times2^4+ \mathbf{1}\times2^3+ \mathbf{1}\times2^2+ \mathbf{1}\times2^1+ \mathbf{1}\times2^0$

8개의 방, 비트를 가지고 셀 수 있는 수는 0부터 255까지 총 256 개가 됩니다. byte 변수를 만들 때 다음과 같이 씁니다.

```
byte sample = 125;
```

byte 변수는 최대 255까지 담을 수 있습니다. 여기에 255 가 넘는 수가 들어오면 다시 0부터 시작됩니다.

```
byte sample = 255;          //   1111 1111
sample = sample + 1;        //   1111 1111 + 1

// 1 0000 0000 이 되고, 아래 8개의 비트만 변수에 저장
// sample 변수에 0000 0000 이 저장
```

```
byte sample = 0;
while(1) {
    sample++;
    Serial.println(sample);
}
```

위 내용을 실행시키면 0부터 255까지 화면에 나오고 다시 0부터 재 시작됩니다. sample 라는 변수가 byte 변수여서 8비트의 저장 공간을 가지고 있기 때문에 그 이상의 비트는 버려지게 됩니다.

char

char 변수는 키보드에 나와 있는 문자나 숫자 하나를 저장할 수 있습니다. 단, 한글을 저장할 수는 없습니다. 문자나 숫자를 저장할 때 ' ' 를 사용합니다. 8비트, 한 바이트짜리 저장공간을 사용합니다. 숫자 저장할 때는 127에서 -128까지 저장할 수 있습니다.

byte 와 char 는 동일한 저장 공간 크기인 1바이트를 가집니다. 단 byte 는 부호를 가지지 않아서 0부터 255까지 사용하지만 char 은 첫 번째 비트를 부호로 사용하기 때문에 7비트로 표시 가능한 0부터 127까지 사용합니다. 그리고 첫 비트가 1일 때는 음수를 표시합니다. -1부터 -128까지 가능합니다.

```
void setup(){
    Serial.begin(9600);
}
void loop(){
  char sample = 'A';
  Serial.println(sample);
}
```

char sample = 'A';

char sample = 65;

이진수	십진수	계산
XXXX XXXX	-128~127	X는 0 또는 1
= **X**×2^7+ **X**×2^6+ **X**×2^5+ **X**×2^4+ **X**×2^3+ **X**×2^2+ **X**×2^1+ **X**×2^0		

이진수	십진수	이진수	십진수
0000 0000	0	1000 0000	-128
0000 0001	1	1000 0001	-127
0000 0010	2	1000 0010	-126
0000 0011	3	1000 0011	-125
0000 0100	4	1000 0100	-124
…	…		
0111 1101	125	1111 1101	-3
0111 1110	126	1111 1110	-2
0111 1111	127	1111 1111	-1

TIP

[2진수와 음수를 나타내기 위한 2의 보수]

전기의 흐름, 즉 비트의 0과 1로만 숫자를 나타낼 수 있는 이진수로 디지털에서 숫자를 처리합니다. 2진수는 각 자리수마다 다음과 같은 방식으로 수를 셉니다. 8비트 이진수는 0000 0000부터 1111 1111 까지입니다. 이진수는 다음과 같이 10진수로 바꿀 수 있습니다.

이진수 : XXXX XXXX = $X\times2^7+ X\times2^6+ X\times2^5+ X\times2^4+ X\times2^3+ X\times2^2+ X\times2^1+ X\times2^0$
(X 는 0 또는 1)

일반적으로 음수는 숫자의 앞에 '-' 표시를 붙입니다. 하지만 디지털 이진수에서 음수를 표현하는 방법은 조금 특별합니다. 음수를 표시하기 위해 "2의 보수"라는 것을 사용합니다.
1 과 -1을 더하면 0이 됩니다. 이것을 이용해서 1과 더해서 0이 되는 수를 찾으면 됩니다. 1은 아두이노에서 이진수로 표현할 때 B 0000 0001 이 됩니다. 이진수 00000001에서 계속 증가하면 최종적으로 11111111까지 됩니다. 11111111에서 1이 증가하면 (1)000000000이 됩니다. 여기서 8비트를 넘어선 9비트째인 1은 버리고, 아래쪽 8개의 비트만 남게 되어 00000000 즉, 0이 됩니다. 다시 말해서 B11111111 에 1을 더하면 0이 됩니다. 그래서 1의 대한 음수표현인 -1은 디지털 이진수에서 11111111 이 됩니다.

0000 0001 과 1111 1111을 더하면 1 0000 0000 즉, 0이 됩니다. -1은 1111 1111입니다.
0000 0010 과 1111 1110을 더하면 1 0000 0000 즉, 0이 됩니다. -2는 1111 1110입니다.
0000 1111 과 1111 0001을 더하면 1 0000 0000 즉, 0이 됩니다. -15는 1111 0001입니다.

2의 보수를 이용한 음수표현은 이진수에서 1과 0을 서로 바꾼 다음 1을 더하면 됩니다.

-1을 표현해 보겠습니다. 1은 0000 0001입니다. 여기서 0과 1을 바꾸면 1111 1110 이 됩니다. 여기에 1을 더하면 최종적으로 1111 1111 이 됩니다.
-12를 표현해 보겠습니다. 12의 이진수는 0000 1100입니다. 0과 1을 바꾸면 1111 0011 이 됩니다. 여기에 1을 더해서 만들어진 1111 0100 이 -12입니다.

int

int 변수는 16비트, 2바이트를 사용합니다. int 변수가 표현할 수 있는 숫자는 -32,768 부터 32,765 까지의 정수 숫자입니다.
일반적으로 아두이노에서 정수 숫자를 사용할 때 가장 많이 사용하는 변수입니다.

```
int sample = 10000;
```

long

long 변수는 4바이트를 사용합니다. lng 변수로 표현할 수 있는 정수는 -2,147,483,648 부터 2,147,483,647 까지의 정수입니다.
int 로 표현할 수 있는 숫자보다 더 큰 숫자를 사용할 때 long 변수를 사용합니다.

```
int sample = 214748364;
int sample = 21474 * 9876;
```

float

float 변수는 -3.4028235E+38 부터 3.4028235E+38 까지의 실수 숫자를 저장합니다. float 변수는 4바이트를 사용합니다.
단 아두이노에서 실수 연산은 속도가 많이 느려지게 됩니다.
그리고 실수연산은 정확도가 많이 떨어집니다. 가능하면 float 같은

실수 변수를 사용하지 않는 것이 좋습니다.

float 변수는 총 32비트 중 부호 1비트, 지수부 8비트, 가수부 23비트를 사용합니다.

그래서 십진수로 따지면 7자리숫자가 유효숫자가 됩니다. float 변수는

$-3.4028235 \times 10^{38}$ ~ 3.4028235×10^{38} 사이의 숫자를 표현할 수 있습니다.

```
float sample = 3.1415;
```

String

엄밀하게는 변수는 아니지만 아두이노에서 String 은 마치 문자열을 저장하고 읽는 일을 하는 변수처럼 사용됩니다.

```
void setup(){
    Serial.begin(9600);
}
void loop(){
    String sample = "Remember 0416";
    Serial.print(sample);
}
```

4. 연산

a. "=" 의 의미

수학에서 "="은 보통 좌변과 우변이 같다는 것을 의미합니다. 하지만 프로그램에서 "=" 는 오른쪽에 있는 값을 왼쪽으로 넣으라는 명령입니다.

```
x = x + 10;
```

x 에 10 을 더한 다음 그 값을 x 에 저장합니다. 그러면 x 라는 저장공간, 변수는 처음 값보다 10이 커진 값으로 바뀝니다.

이것을 더 간단하게 이렇게 쓸 수 있습니다.

```
x += 10;
```

자주 사용하는 1 을 더하는 것과 1 을 빼는 것은 ++, -- 를 사용합니다.

```
x = x + 1;
x++;
```

위의 두 줄은 같은 의미입니다.

b. 기본 수학연산

+, -, *, /, % 이렇게 다섯 가지 기본 수학연산이 가능합니다.

+ : 더하기

- : 빼기

* : 곱하기

/ : 나누기

% : 나머지

위의 = 과 결합하여 다음과 같이 사용할 수 있습니다.

`result = result + a;`	`result += a;`	변수 result 에 a 를 더해서 저장합니다
`result = result - a;`	`result -= a;`	변수 result 에 a 를 빼서 저장합니다
`result = result * a;`	`result *= a;`	변수 result 에 a 를 곱해서 저장합니다
`result = result / a;`	`result /= a;`	변수 result 를 a 로 나누고 저장합니다
`result = result % a;`	`result %= a;`	변수 result 를 a 로 나눈 나머지를 저장합니다

c. 비교

크거나 작거나 같거나를 가지고 참과 거짓을 판별한다.

```
x == y     // x 가 y 와 같다.
```

만약 x 와 y 가 같은 값이라면 이 결과는 "진실"이 되고, x 와 y 가 다른 값이라면 이 결과는 "거짓"이 된다. 이 외에 다음과 같은 비교가 가능하다.

```
x > y      // x 가 y 보다 크다.
x >= y     // x 가 y 보다 크거나 같다.
x < y      // x 가 y 보다 작다.
x <= y     // x 가 y 보다 작거나 같다.
x != y     // x 와 y 가 다르다.
```

d. 논리

비교와 함께 자주 사용된다.

```
if ( (x > 0) && (x < 10) )  // x 가 0 보다 크다 그리고 x 가 10 보다 작다
```

&& : 논리곱, 오른쪽과 왼쪽이 모두 참일 때 결과는 참이 된다.
|| : 논리합, 오른쪽 또는 왼쪽, 둘 중 하나라도 참이면 결과는 참이다.
! : 부정, 참이면 거짓이, 거짓이면 참이 된다.

```
( x>0 && x<5 )   // x 가 0와 5 사이일 때 참
( x>10 || x <5 )  // x 가 10보다 크거나 5보다 작을 때 참
( !x )            // x 가 참이면 거짓이, 거짓이면 참이 된다.
```

"비교"와 "논리"는 보통 조건문과 함께 사용된다. if 또는 for 문 등과 함께 사용된다.

5. 상수

변수와 달리 상수는 항상 그대로 있는 수를 말합니다. 보통 변해서는 안 돼는 수를 미리 지정해두는 식으로 사용됩니다.

```
#define  SIZE  1024
```

#define 문을 사용, 프로그램의 가장 앞부분에서 사용할 수 있습니다. 보통은 가장 많이 사용되는 방법입니다.

또는 const 변수를 사용할 수 있습니다. 기존의 변수 앞에 const를 추가하면 그 변수는 읽을 수는 있지만 쓸 수 없는 상수가 됩니다.

```
const int SIZE 1024;
```

이렇게 const 를 붙여서 만들 수도 있습니다. 보통 상수는 변수와 다르다는 것을 나타내기 위해 이름을 대문자로 사용합니다. SIZE 라는 변수는 상수 1024를 저장합니다. 그리고 SIZE 는 읽을 수만 있지 쓸 수는 없습니다. 즉, 값이 1024 는 변경되지 않는 값입니다.

변수 대신 상수를 이용해서 프로그램을 하는 이유는 실수로 상수를 다른 값으로 변경시키는 것을 방지하기 위해서입니다. 함께 프로그램을 하는 다른 사람일 수도 있고, 혹은 어떤 함수에서 변수를 이용하면서 값을 변경시킬 수도 있습니다. 이러한 실수를 미연에 방지하기 위해 변수 대신 상수를 사용합니다.

6. 참과 거짓

(a == 12) 는 a 의 값을 읽어 와서 그것이 12 인지를 비교해 봅니다. 12라면 참을 12가 아니라면 거짓을 보냅니다. 참은 1 이 되고, 거짓은 0 이 됩니다.

```
if ( a == 12 )
{
   할일 ;
}
```

a 가 12 일 때 (a==12) 는 참(1)이 되고, 그래서 "할일"을 하게 됩니다.

```
if ( a == 12 )
{
   Serial.println(“Great! A=12”);
}
```

프로그램 안에 위 코드를 넣어주면 a 값이 12가 될 때 화면에

Great! A=12

라는 문구를 보여줍니다.

7. HIGH / LOW

아두이노에서 HIGH 는 1 을 의미하고 이는 ON 또는 5V 나 3.3V 의 전원이 들어가는 것을 의미합니다. 반대로 LOW 는 0 을 의미하고, OFF 나 0V 즉, GND 에 연결되었다는 것을 뜻합니다. 즉, 1 대신 HIGH를 사용하고 0 대신 LOW를 사용할 수 있습니다.

```
digitalWrite(13, HIGH);  // 13번 핀에서 5V(또는 3.3V) 를 내보냅니다.
digitalWrite(13, LOW);   // 13번 핀이 GND(0V) 에 연결됩니다.
```

```
digitalWrite(13, 1);  // 13번 핀에서 5V(또는 3.3V) 를 내보냅니다.
digitalWrite(13, 0);   // 13번 핀이 GND(0V) 에 연결됩니다.
```

위의 예에서 보듯이 HIGH 와 1은 동일하고 LOW 와 0 도 동일합니다. 1 과 0 대신 HIGH 와 LOW를 써서 이유는 사람이 보기 쉽게 알아볼 수 있도록 합니다.

8. if 조건문

a. if 문

if 조건문은 조건이 참 일 때만 실행합니다.

```
if (a==13)        // 조건부분
{
   // 조건이 참이면 이 부분을 실행합니다.
}
```

```
int condition =0;     // 전역변수를 만들고 값을 0으로 만든다.
void setup(){
   Serial.begin(9600);
}
void loop(){
  condition ++;       // 전역변수의 값을 1 증가
  if ((condition % 5)==0)  { // 전역변수를 5로 나눈 나머지가 0일 때만 아래 조건 실행
    Serial.print("condition = ");
    Serial.println(condition);       // 값을 화면에 프린트
  }
}
```

b. if , else 문

if 문과 함께 조건이 거짓일 때만 실행되는 else 도 있습니다.

```
if (조건)
{
   // 조건이 참일때 실행
}
else
{
   // 조건이 거짓일때 실행
}
```

나머지 연산자를 사용해서 13번 LED를 켜고 끄는 프로그램을 예로 들어보겠습니다.

```
int condition =0;
void setup(){
   Serial.begin(9600);
   pinMode(13, OUTPUT);
}
void loop(){
  condition ++;
  if ((condition % 10) >4)  {
    digitalWrite(13,HIGH); Serial.println("LED ON"); delay(100);
  }
  else {
    digitalWrite(13,LOW); Serial.println("LED OFF"); delay(100);
  }
}
```

if 와 else if 를 사용해서 [만약 ~ 이면] ... [그렇지 않고 만약 ~ 이면]... 의 구문을 만들 수 있습니다.

```
if (inputValue <=500)
{
    첫 번째 조건을 만족할 때 할일들;
}
else if (inputValue > 1000)
{
    두 번째 조건을 만족할 때 할일들;
}
else
{
    첫 번째, 두 번째 조건을 모두 만족시키지 못할 때 할일들;
}
```

9. for 반복문

다음과 같은 구문으로 일정횟수만큼 명령을 반복합니다.

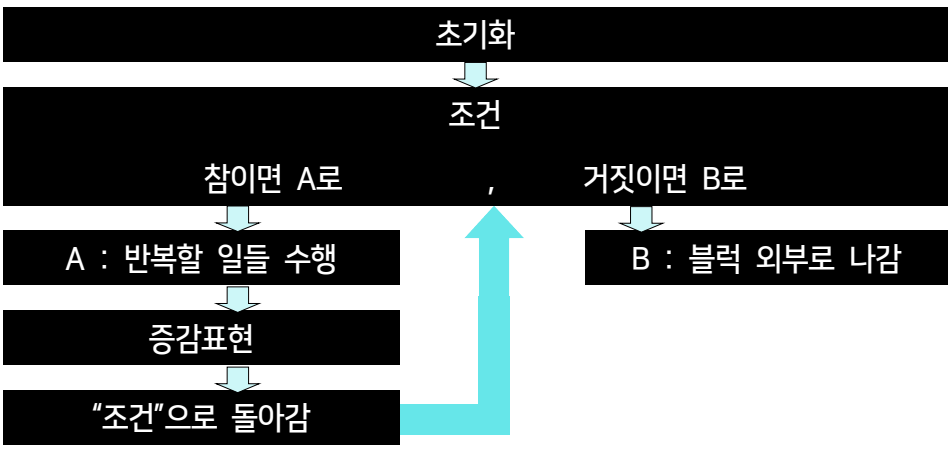
```
for ( 초기화; 조건; 증감표현)
{
    반복할 일들;
}
```

초기화는 변수를 만들어서 초기화 시킬 수도 있고, 외부에서 만들어진 변수를 사용할 수도 있습니다.

조건은 변수를 사용해서 조건에 맞을 때만 아래 블록 안에 있는 내용을 실행합니다.

증감표현은 블록 안의 내용을 끝까지 실행시킨 다음 실행됩니다.

실행순서는 다음과 같습니다.



아두이노 13번 핀에 에 붙어 있는 LED를 20번 반복해서 깜박이는 프로그램을 짜보면 다음과 같습니다.

```
for (int i = 0; i<20 ; i++)
{
   digitalWrite(13, HIGH);
   delay(500);
   digitalWrite(13, LOW);
   delay(500);
}
```

위의 for 문을 포함한 아두이노 전체 코드는 다음과 같습니다.

```
void setup(){
   pinMode(13, OUTPUT);
   for (int i =0; i<20 ; i++)
   {
      digitalWrite(13, HIGH);
      delay(500);
      digitalWrite(13, LOW);
      delay(500);
   }
}
void loop(){
}
```

loop()을 쓰지 않고 setup() 에 넣어 20번만 반복하게 프로그램 했습니다.

10. while 반복문

a. while 문

while 반복문은 조건이 만족되는 한 무한히 반복됩니다.

```
while ( i > 3 )
{
    명령문1;
    명령문2;
    명령문3;
}
```

변수 i 가 3보다 크면 { } 내부의 명령문1,2,3이 반복적으로 실행됩니다.

명령문3 이 실행된 다음 (i 〉 3) 의 조건을 만족하는지 판단 한 후 만족하면 다시 명령문1 부터 실행됩니다. (i 〉 3) 이 거짓이라면 while 문 블록을 벗어납니다.

b. do , while 문

```
while ( 조건 )
{
    명령문;
}
```

위와 같은 while 문은 조건이 거짓이면 블럭 내부의 명령문은 한번도 실행되지 않습니다. 명령문을 최소한 한번 실행시키기 위해서 조건을 뒤로 보내는 do ... while (조건) 이 있습니다.

```
do
{
    명령문;
} while(조건);
```

아날로그 센서의 값을 읽은 뒤 x 가 100 보다 작으면 계속해서 센서를 읽게 하는 예제는 다음과 같습니다.

```
1 do
2 {
3     x = readSensors(); // 센서값을 읽어서 x 에 저장한다.
4     delay(50);    // 50ms 동안 기다린다.
5 }  while(x<100);   // x 가 100 보다 작으면 반복한다.
```

11. 아두이노 내부 함수들

a. INPUT / OUTPUT

아두이노는 핀을 통해 외부에서 아두이노 안으로 신호가 들어올 수도 있고, 아두이노에서 바깥으로 신호가 나갈 수도 있습니다. 들어오도록 할때 INPUT 을 쓰고, 나가도록 할 때 OUTPUT 을 씁니다.

INPUT 과 **OUTPUT** 은 **pinMode()** 라는 함수에서 각 핀의 번호와 연결되어서 입력으로 사용할지, 출력으로 사용할지를 결정할 때 사용됩니다.

```
pinMode(13, OUTPUT); // 13 번 핀을 OUTPUT (출력)으로 사용합니다.
```

b. pinMode(pin, mode)

void setup() 안에서 특정 핀의 입력과 출력을 지정하기 위해 사용됩니다.

```
pinMode(13, OUTPUT); // 13번 핀을 출력으로 지정한다.
pinMode(12, INPUT);  // 12번 핀을 입력으로 지정한다.
pinMode(11, INPUT_PULLUP);  // 11번 핀을 입력으로 지정한다. 풀업된다.
```

여기서 풀업, **INPUT_PULLUP** 의 의미는 내부적으로 20kΩ 저항이 핀과 VCC (5V) 사이에 들어간다는 것을 의미합니다. OUTPUT 으로 지정해서 출력으로 만들었을 때 핀은 최대 40mA 까지 전류를 흘려보낼 수 있습니다. 이 전류는 LED 를 밝게하기에는 충분하지만 릴레이나 모터 등을 작동시키기에는 부족합니다. 외부 장치와 출력핀이 바로 연결된 때 과도한 전류가 흐르게 되면 아두이노의 ATMEGA MCU가 손상될 수 있습니다. OUTPUT 으로 지정된 핀에는 일반적으로 360 Ω 에서 1 kΩ 정도의 저항을 직렬로 연결해 주는 것이 좋습니다.

V = I x R

5V = I x 360

I = 5/360 = 14mA

전압, 전류, 저항의 관계를 **"오옴의 법칙"**이라고 합니다. 오옴의 법칙에서 전압은 전류와 저항의 곱으로 나타납니다. 5V 의 전압이 걸린 곳에 360Ω 저항을 달면 전류는 왼쪽의 공식에 따라 0.014A 즉, 14mA 가 나옵니다. 이정도면 LED 의 불을 켜서 전기가 흐르는지, 신호가 들어오는지 확인하기에는 적당합니다.

c. digitalRead(핀)

아두이노 입력으로 설정된 핀의 디지털값을 읽습니다. 일반적으로 핀에 걸린 전압이 VCC 이면 1 을, GND 이면 0 을 반환합니다.

```
value = digitalRead(12); // 12번 핀을 읽어서 value 변수에 저장
```

단, digitalRead()를 쓰기 전에 pinMode()를 사용해서 핀의 출력을 설정해 두어야합니다.

```
pinMode(12, INPUT);
value = digitalRead(12); // 12번 핀을 읽어서 value 변수에 저장
                         // value 에는 0 또는 1 이 저장됨
```

d. digitalWrite(핀, 값)

아두이노 출력핀에 정해진 값을 출력합니다. 값은 0 과 1 둘 중의 하나입니다.

```
digitalWrite(11, 1);  // 11번 핀에 1(+5V) 을 출력한다.
                      // 이것을 다음으로 쓸 수 있다.
digitalWrite(11, HIGH); // HIGH 는 1 과 같고, LOW 는 0 과 같다.
```

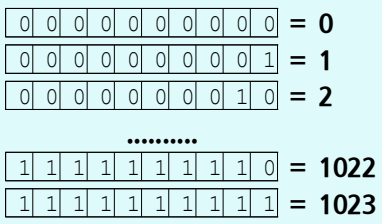
e. analogRead(핀)

아날로그 값을 읽습니다.

```
value = analogRead(A0); // 핀 A0 에 연결된 값을 읽어온다.
          // value 에 저장되는 값은 최소 0에서 최대 1023 사이 정수이다.
```

아두이노에서 아날로그 값을 읽는 것은 0V 에서 5V 사이의 어떤 전압값을 읽는 것입니다. 아두이노는 10 개의 비트로 값을 분해합니다. 10개의 비트는 2 의 10 승을 의미하고 이는 총 1024 가 됩니다. 즉, 최소값은 0 이고 최대값은 1023 이 됩니다.

아두이노 아날로그 입력을 받을 때 주의할 것은 아날로그 입력핀에 들어오는 전압이 0V에서 5V 사이가 되어야 합니다. 이 값을 벗어난 전압이 들어올 경우 아두이노 내부 회로가 망가질 수 있습니다.

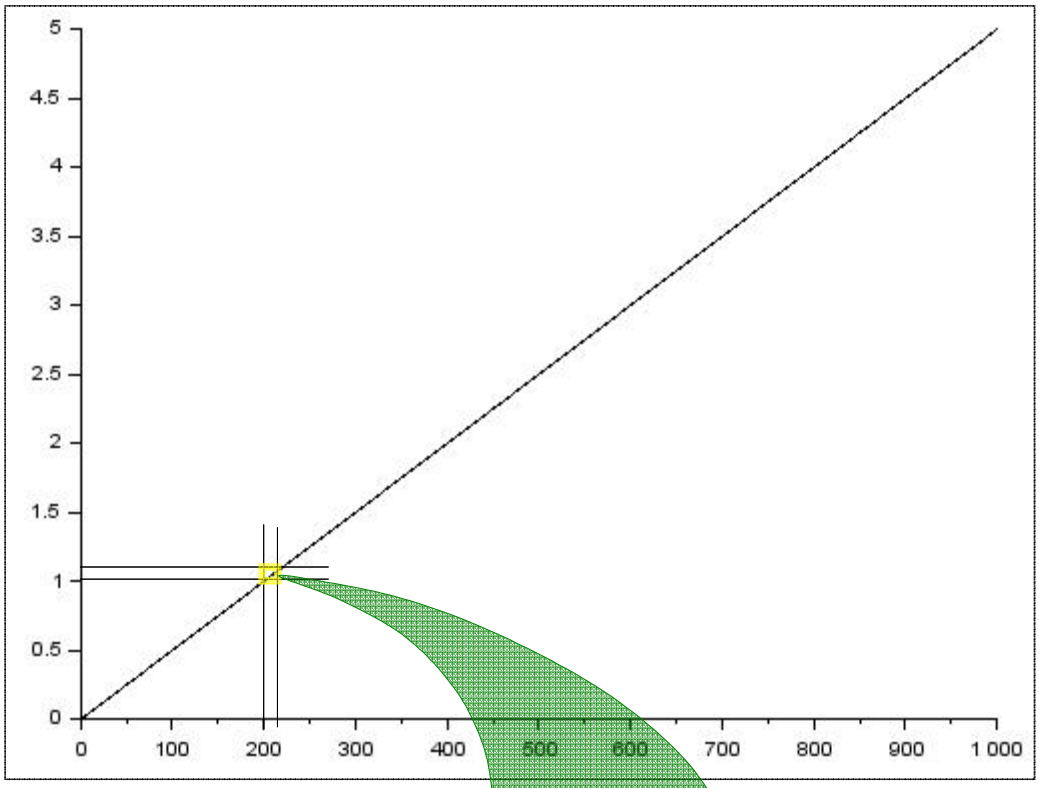


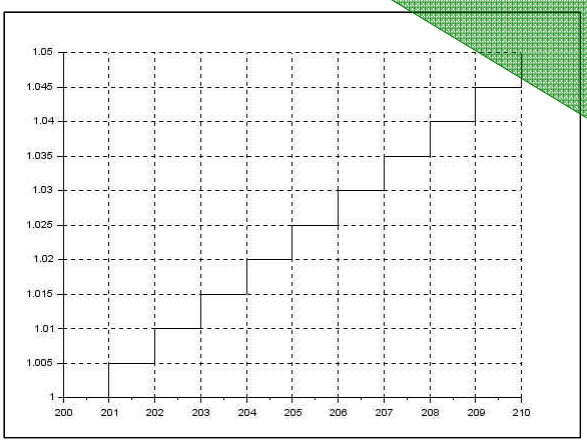
ADC 라고 불리는 아날로그-디지탈 변환장치는 변환시킬 수 있는 정도와 시간에 따라 분류됩니다. 변환시킬 수 있는 정도를 보통 비트(bits)로 표시하고, 변환에 걸리는 시간을 Hz 로 표시합니다.

아두이노 ADC 샘플링 시간은 다른 작업에 걸리는 시간을 모두 최소화할 때 9kHz 정도가 됩니다. 1초에 9,000번 까지 가능합니다.

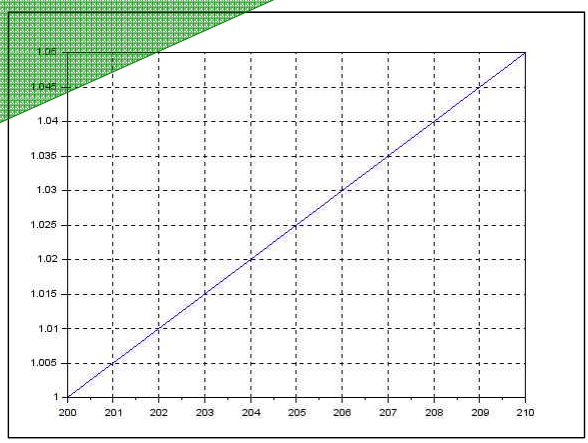
ADC 10 비트는 총 1024 (≒1000) 단계로 입력된 아날로그 값을 변환합니다. 이 경우 0.0048828V(≒0.005V) 즉, 4.89mV 단계로 입력되어지는 전압 값을 구별할 수 있습니다.

TIP





ADC를 거친 후 아두이노에서 받는 값

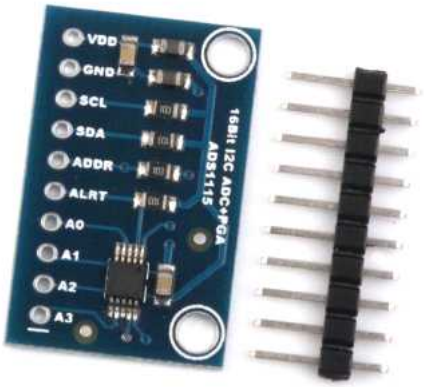


실제 아날로그 값

TIP

실제 아날로그 값은 0 과 1 사이를 무한히 나눌 수 있습니다. 0V 와 1V 사이는 무수히 많은 값들이 있습니다. 0.1, 0.2, 0.2012549, 0.20199859, 등 셀 수 없는 많은 값이 존재합니다. 그러한 값을 모두 읽어 들일 수 있으면 좋겠지요. 하지만 실제로 측정할 수 있는 방법이 없습니다. 읽을 수 있는 가능한 범위가 있을 뿐입니다. 아두이노에 있는 ATMEL 칩은 10 비트 단위의 정밀도를 가지고 아날로그값을 디지털로 변환합니다.

즉, 위에 있는 그래프처럼 아날로그는 1V 와 1.005V 사이에 무한한 연속적인 값을 가집니다. 하지만 아두이노에서는 1V 와 1.005V 사이에는 다른 값이 없습니다. 입력되는 아날로그값이 1V 와 1.005V 사이라면 아두이노에서 인식되는 값은 1V 가 됩니다. 그 다음 단계인 1.005V 와 1.010V 사이의 아날로그값이 입력되면 아두이노에서 인식되는 값은 1.005V 가 됩니다. 이렇게 총 1024 단계 ($=2^{10}$)를 가집니다.



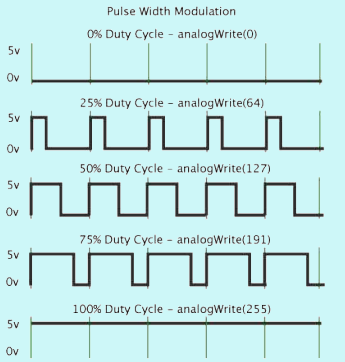
아두이노 보드와 연결해서 사용할 수 있는 16비트 ADC 보드입니다. 16비트로 ADC를 사용하기 때문에 2의 16승인 65536 단계로 나눠집니다. 아두이노가 1024 단계였던 것과 비교하면 64배 ($=2^6$) 큰 것입니다.

f. analogWrite(핀, 값)

핀에 아날로그 값을 씁니다. 이때 핀은 pwm 출력이 가능한 핀만 가능합니다. pwm 은 보드에서 번호 옆에 ~ 표시가 있는 핀만 사용할 수 있습니다. UNO 의 경우 **3,5,6,9,10,11** 핀이 **pwm** 이 가능하고, **analogWrite()** 로 값을 쓸 수 있습니다.

```
analogWrite(3,127); // 아날로그 출력핀 3 번에 127 의 값인 2.5V를 출력
```

쓸 수 있는 값은 0 부터 255 까지로 분해능이 8 비트입니다. 최대 256 단계까지 가능합니다. pwm 은 2ms 의 시간 안에서 256 단계로 나누어 일부는 1 을, 나머지는 0 을 출력하는 방식입니다. 이것을 2ms 의 주기로 반복합니다.



PWM 이란 Pulse Width Modulation 의 약자로 펄스의 폭을 조절하는 것입니다. 아두이노에서는 2ms 즉, 0.002 초 안에서 ON 과 OFF를 나눠서 나가는 출력을 조절합니다. 그래서 엄밀하게는 Digital Analog Converter 가 아니지만 모터를 제어하거나 온도를 제어하거나 LED 의 빛의 세기를 제어할 때는 PWM을 DAC 로 사용할 수 있습니다.
아두이노 보드에 있는 핀번호 옆에 물결(~)표시가 있는 핀이 PWM 출력이 가능한 핀입니다.

g. delay(시간)

delay 는 주어진 시간만큼 아무것도 하지 않고 기다립니다. 시간의 단위는 ms 이고, 1000ms 는 1초가 됩니다.

```
delay(1000); // 1000ms 즉 1초 동안 기다림
```

delay() 함수를 쓸 경우 주의할 점은 함수 실행 중인 시간동안 다른 작업을 하지 못한다는 것입니다. 만약 스위치입력을 받아야 한다면 delay() 함수의 사용을 잘 고려해서 써야 합니다. delay() 되는 시간에 스위치 작동을 시키면 아두이노에서는 스위치 입력을 인식하지 못할 수도 있습니다.

h. millis()

현재 시간을 반환합니다. 현재 시간은 아두이노에 전원이 들어온 뒤로 흐른 시간을 의미합니다. 시간의 단위는 ms입니다. milis() 함수를 사용해서 시간을 저장할 때는 unsigned long 타입의 변수를 사용합니다.

```
unsigned long value;
value = milis();
```

milis() 함수를 사용해서 아두이노가 중간에 정지하지 않는 delay(1000)을 만들 수 있습니다. 아래 소스를 분석해 보시면 delay()를 사용하지 않고도 동일한 효과를 내는 방법을 알 수 있습니다.

```
1  unsigned long previousMillis = 0;
2  unsigned long currentMillis = millis();
3
4    if (currentMillis - previousMillis >= 1000) {
5      previousMillis = currentMillis;
6      if (ledState == LOW) { ledState = HIGH; }
7      else { ledState = LOW; }
8    }
```

소스 분석은 이 책에서 다루고자하는 범위를 넘어섭니다. 따로 홈페이지에 올려두겠습니다.

12. 다양한 수학 함수들

mix(x,y) : 둘 중 작은 수를 반환한다.

max(x,y) : 둘 중 큰 수를 반환한다.

randomSeed(value) : random() 함수의 시작지점을 결정할 때 사용합니다.
보통 value 로 오픈된 상태의 analogRead() 값을 사용합니다.

random(max); 0 과 max 사이의 값 중 하나를 무작위로 만든다.

random(min,max); min 과 max 사이의 값 중 하나를 무작위로 만든다.

```
int number;
randomSeed(analogRead(A0)); // random() 초기화
number = random(1000); // 0 과 1000 사이의 값을 무작위(random)로 만듭니다.
number = random(50,200); // 50 과 200 사이의 값을 무작위(random)로 만듭니다.
```

analogRead(A0) 는 A0 핀에 들어오는 아날로그값 (0V ~ 5V)입니다. A0에 아무것도 연결하지 않은 상태에서 무작위의 값이 들어오면 그 값으로 random() 함수를 초기화시킵니다.

13. 디버깅

Serial.begin(rate) : setup() 함수의 내부에 시리얼통신 속도를 설정합니다.

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

Serial.print(data) : 데이터를 시리얼포트로 전송합니다. 일반적으로 아두이노 보드에 있는 어떤 값(문자열)을 PC 쪽으로 보낼때 사용합니다.

Serial.println() 은 Serial.print() 를 보낸 후 마지막에 줄바꿈 신호를 덧불입니다. 즉, Serial.println() 을 쓰면 문자를 시리얼통신 창에 쓴 다음 줄을 바꾸고 새줄의 첫번째 칸으로 커서가 이동한다.

```
void setup(){
  Serial.begin(9600);  // 속도는 9600
}
void loop(){
  Serial.print(analogValue); // analogValue 에 들어있는 값을 시리얼통신 전송
  Serial.println("전송하는 문자열"); // 문자열을 시리얼통신으로 전송
}
```

아두이노같은 MCU를 프로그램 할 때는 프로그램이 제대로 되었는지 아닌지 확인할 필요가 있습니다. 이런 확인 작업을 '디버깅'한다고 말합니다.
JTAG 와 같은 디버깅에 사용되는 용어와 장비들이 있습니다. 하지만 이러한 장비들과 프로그램은 매우 가격이 비쌉니다. 오른쪽 사진은 STM32 시리즈 MCU 에 사용된는 디버깅장비입니다.

그래서 프로그램 초기에 하던 방식으로 매우 간단하면서 저렴한 방식을 아두이노에서 채택했습니다. 그것이 시리얼통신을 이용한 모니터링방식입니다. 물론 JTAG 등의 전문적인 장비나 프로그램을 가지고 프로그램하는 것과 비교하면 매우 성능도 낮습니다. 하지만 가격이 따로 발생하지 않기 때문에 아두이노와 같은 복잡하지 않은 프로그램에서 사용하기는 적당합니다.

아두이노는 Serial.print(), Serial.println() 같은 문자열을 화면에 보여주는 함수를 사용해서 필요한 정보를 PC 에 보낼 수 있습니다. 이렇게 전달되는 정보를 화면으로 보며 어디까지 실행되었는지를 확인할 수 있습니다. 일반적으로 특별한 변수의 값을 보여주거나 혹은 어떤 부분이 실행이 되었는지 아닌지의 여부를 보여주는 용도로 사용됩니다.

```
int pushButton =2;
void setup() {
        Serial.begin(9600);
        pinMode(pushButton, INPUT_PULLUP);
        pinMode(13, OUTPUT);
}
void loop() {
        int buttonState = digitalRead(pushButton);
        Serial.print("buttonState = ");
        Serial.println(buttonState);
        if (buttonState) {
            digitalWrite(13, HIGH);
            Serial.println("Push SW ON");
        }
        else {
            digitalWrite(13, LOW);
            Serial.println("Push SW OFF");
        }
        delay(1);
}
```

Serial.begin(9600); 으로 PC 와 RS232 시리얼통신을 할 수 있도록 합니다.

Serial.print("buttonState = ");
Serial.println(buttonState);
buttonState 값을 화면에 보여줍니다.

buttonState 가 1 이면
Serial.println("Push SW ON");를 실행, 화면에 Push SW ON 을 보여줍니다.

buttonState 가 0 이면
Serial.println("Push SW OFF");를 실행, 화면에 Push SW OFF 을 보여줍니다.

디버깅을 위해서 Serial.println() 등의 함수를 쓸 때는 프로그램의 실행순서를 머릿속으로 따라가면서 의심이 드는 장소에 사용합니다.

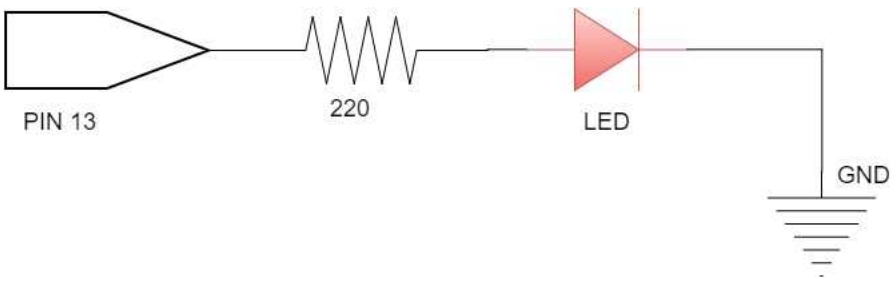
>> 혹시 버튼에 이상이 있지 않을까?
>> buttonState 값을 출력해서 확인합니다. 버튼이 고장이거나 선이 단선(겉으로는 연결되었지만 속은 끊어진 상태) 되었거나 혹은 연결이 잘 안되어 있는지 등을 확인할 수 있습니다.

>> 혹시 LED 가 고장인가?
>> buttonState 가 1 이어서 "Push SW ON"이 화면에 나왔는데 LED 가 켜지지 않는다면 LED 이상이거나 연결된 저항에 문제가 있거나 전선 연결에 문제가 있는 등의 다양한 문제를 점검해봐야 합니다.

>> if 조건문 작동은 잘되는가?
>> buttonState 값에 따라서 if 와 else 구문이 제대로 작동하고 있는가?

4장. 아두이노 프로그램 기본구조

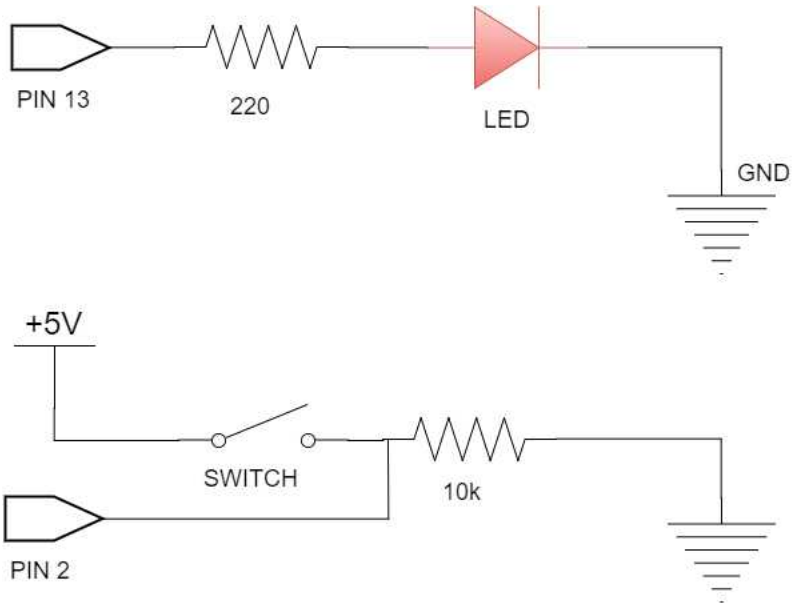
1. 디지털출력



아두이노의 가장 기초 프로그램으로 핀 하나의 출력을 ON 또는 OFF 로 합니다. 13번 핀을 출력으로 설정하고 1초 간격으로 ON 과 OFF 를 반복합니다.

```
1  int ledpin = 13;
2  void setup() {
3     pinMode(ledpin, OUTPUT);
4  }
5  void loop(){
6     digitalWrite(ledpin, HIGH);
7     delay(1000);
8     digitalWrite(ledpin, LOW);
9     delay(1000);
10 }
```

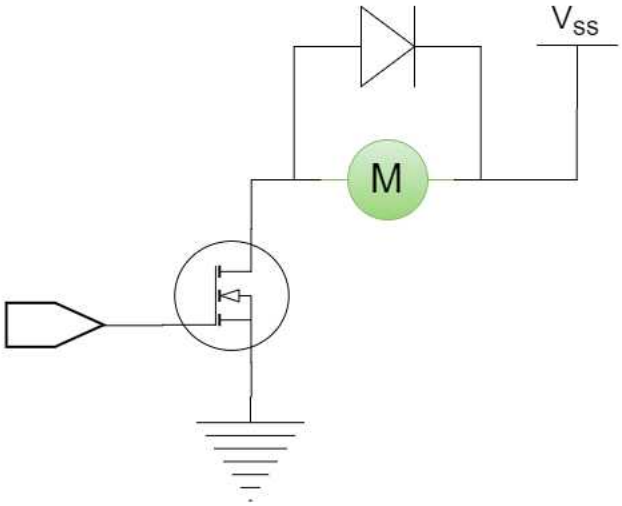
2. 디지털입력



디지털핀 2번을 입력으로 설정하고 스위치를 연결합니다.
연결된 스위치가 ON 일때 13번 LED 가 1초 간격으로 깜박이게 합니다.

```
1  int ledPin = 13;
2  int switchPin = 2;
3  void setup(){
4     pinMode(ledPin, OUTPUT);
5     pinMode(switchPin, INPUT);
6  }
7  void loop(){
8     if (digitalRead(switchPin)==HIGH){
9        digitalWrite(ledPin, HIGH);
10       delay(1000);
11       digitalWrite(ledPin, LOW);
12       delay(1000);
13    }
14 }
```

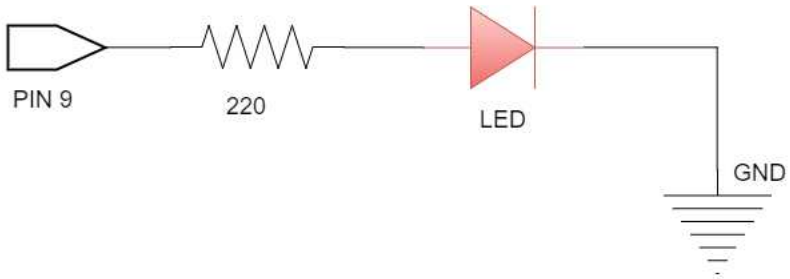
3. 대전류 출력



40mA 이상의 출력이 필요하면 MOSFET 등을 사용합니다. DC 모터는 회전하다 정지할 때 역기전력이 발생합니다. 역기전력을 상쇄시키기 위해 역방향으로 다이오드를 부착합니다. IRF510 을 사용한 DC 모터 사용 예제는 다음과 같습니다.

```
int outPin = 5;
void setup(){
    pinMode(outPin, OUTPUT);
}
void loop(){
    for (int i=0; i<=5; i++) {
        digitalWrite(outPin, HIGH);
        delay(250);
        digitalWrite(outPin, LOW);
        delay(250);
    }
    delay(1000);
}
```

4. PWM 출력

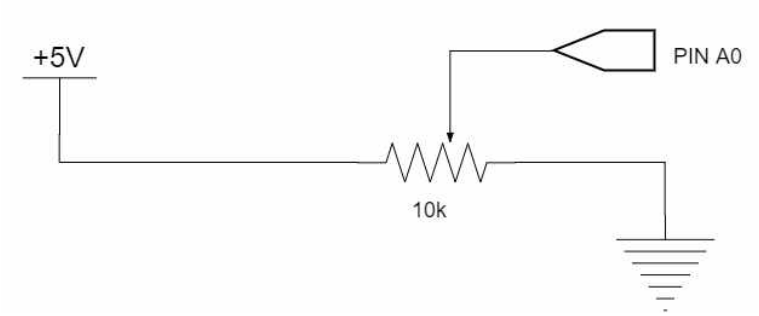


디지탈출력의 경우 OFF인 0 과 ON인 1 로 구분됩니다. OFF는 전압이 0V 인 경우를 나타내고 ON은 들어오는 전압 VCC 가 나가는 것을 의미합니다. PWM 은 짧은 시간 속에 ON 과 OFF 를 반복하는 것입니다. 아두이노에서 PWM은 2ms 주기 신호 안에 ON 과 OFF 를 일정비율로 넣습니다.

```
1 int ledPin = 9;
2 void setup() {
3 }
4 void loop(){
5     for (int i=9; i<=255; i++) {
6         analogWrite(ledPin, i);  delay(100);
7     }
8     for (int i=255; i>=0; i--) {
9         analogWrite(ledPin, i);  delay(100);
10    }
11 }
```

PWM 출력에는 **`analogWrite()`** 함수를 사용합니다. 예제는 PWM 으로 LED 의 불빛의 밝기를 조절한 것입니다. PWM 출력이 가능한 핀은 아두이노 보드의 핀 번호 옆에 물결표시(~)로 표시되어 있습니다.

5. 아날로그값 입력받기




```
1  int potPin = A0;
2  int ledPin = 13;
3  void setup(){
4     pinMode(ledPin, OUTPUT);
5  }
6  void loop(){
7     digitalWrite(ledPin, HIGH);
8     delay(analogRead(potPin));
9     digitalWrite(ledPin, LOW);
10    delay(analogRead(potPin));
11 }
12
```

아두이노에는 아날로그 값을 읽어들이는 ADC가 있습니다. ADC는 아날로그값을 디지탈로 변환해서 그 값을 읽어들입니다. 0V 에서 기준 전압까지를 총 1024 단계로 나누어 그 단계값을 계산합니다. analogRead() 함수를 사용하여 가변저항을 +5V와 GND(0V) 사이에 연결하고 그 중간부분에 위치한 전압값을 읽어 들입니다.

5장. 하드웨어 연결

아두이노와 같은 MCU 는 PC 프로그램과 달리 입력과 출력을 특징 지워 주어야 합니다. 쉽게 말해서 PC 는 키보드와 마우스라는 입력장치가 기본적으로 달려있습니다. 출력도 모니터와 프린터라는 출력장치가 있습니다. 이렇게 기본 입력장치와 출력장치가 이미 붙어 있기 때문에 PC 프로그램을 하는 사람들은 입, 출력 장치에 대해서 크게 신경을 쓰지 않습니다. 대부분 OS 수준에서 알아서 입, 출력 장치가 처리가 됩니다.

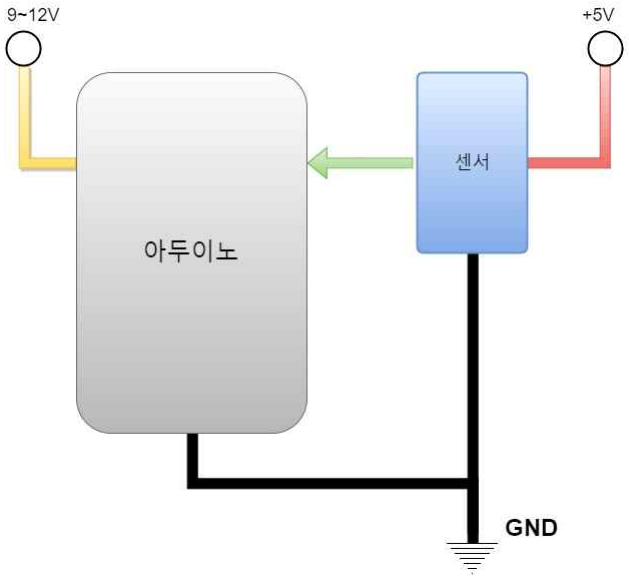
하지만 아두이노 같은 MCU 는 입력과 출력을 잘 지정해 주어야 합니다. 이를테면 스위치를 붙인다고 하더라도 디지털 13번에 붙일 것인지, 디지털 2번에 붙일 것인지, 아니면 디지털핀이 모자라서 아날로그 입력으로 주로 사용되는 A0 에 붙일 것인지를 결정하고, 그에 따라서 스위치와 센서 등을 아두이노와 제대로 연결해 주어야 합니다.

센서나 모터 등을 연결할 때 주의할 것을 몇 가지 이 장에서는 알려드리도록 하겠습니다.

1. 센서 연결

입력장치인 센서를 연결할 때 주의할 점은 다음과 같습니다.

아두이노와 센서의 GND를 연결합니다.



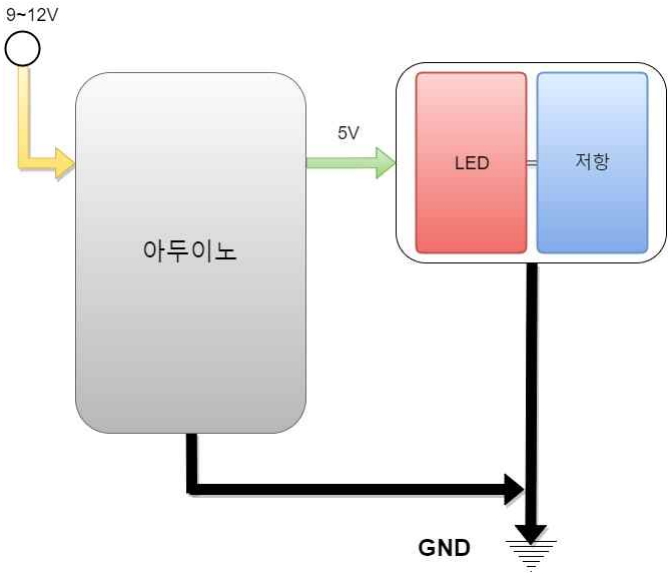
센서에서 나오는 값이 아두이노로 들어가야 합니다. 그러기 위해서 아두이노와 센서는 동일한 전압기준을 가져야 합니다. 이것을 GND 라고 이름을 붙입니다. 그리고 대부분의 경우 GND 는 0V 가 됩니다.

센서가 제대로 동작하기 위해서 센서에 전원 즉, 전기가 들어가야 합니다. 아두이노에 사용되는 대부분의 센서는 5V 전원을 필요로 합니다. 아두이노에서 나오는 5V를 사용하거나 혹은 다른 SMPS에서 5V를 만들어서 사용해도 됩니다. 그리고 센서와 아두이노의 GND 는 서로 연결해 둡니다. 그리고 센서에서 나오는 값과 센서의 핀, 아두이노에 연결되어 들어가는 핀을 꼭 확인해 둡시다. 종종 다른 사람이 짠 프로그램을 그대로 적용시키면서 하드웨어는 고치지 않아서 프로그램이 잘못된 것 같다고 소스코드만 질문게시판에 올리는 초보자들이 있습니다. 꼭 기억하세요. 하드웨어 연결 상태를 확인하지 않고 소스코드만 고쳐달라고 해서는 안됩니다. 아두이노는 하드웨어와 프로그램 소스 모두가 중요합니다.

2. 출력 연결

작은 전류만 허용되는 출력

일반적으로 가장 많이 사용되는 LED 연결을 예로 들어보겠습니다. LED 는 위에서 언급했듯이 20mA 정도의 전류를 흐르게 하는 것이 적절합니다. 그보다 많으면 LED 가 고장 날 수 있습니다. 너무 적은 전류는 빛이 약해서 켜진 상태인지 꺼진 상태인지 잘 분간이 안 될 수도 있습니다.



LED 와 같이 작은 전류만 허용되는 출력을 사용할 경우 저항을 꼭 함께 사용해야만 합니다. 오옴의 법칙을 이용해서 전압, 전류, 저항 관계식으로 허용 전류 이상이 출력으로 나오지 않게 하셔야 합니다. 이를테면 LED를 하나 켜고 끈다고 할 때,

V = I R을 쓰면,

(5-2)V = 0.02A x R …… LED 의 전압강하가 2V 정도 됩니다.
R = 3V / 0.02A = 150 …… 계산된 저항값은 150이 나옵니다.
150 Ω 저항을 사용하면 20mA 정도의 전류가 LED를 흐릅니다. 보통은 조금 안전하게 이보다 약간 큰 저항을 사용합니다. 200Ω에서 400Ω 정도의 저항을 사용하면 됩니다.

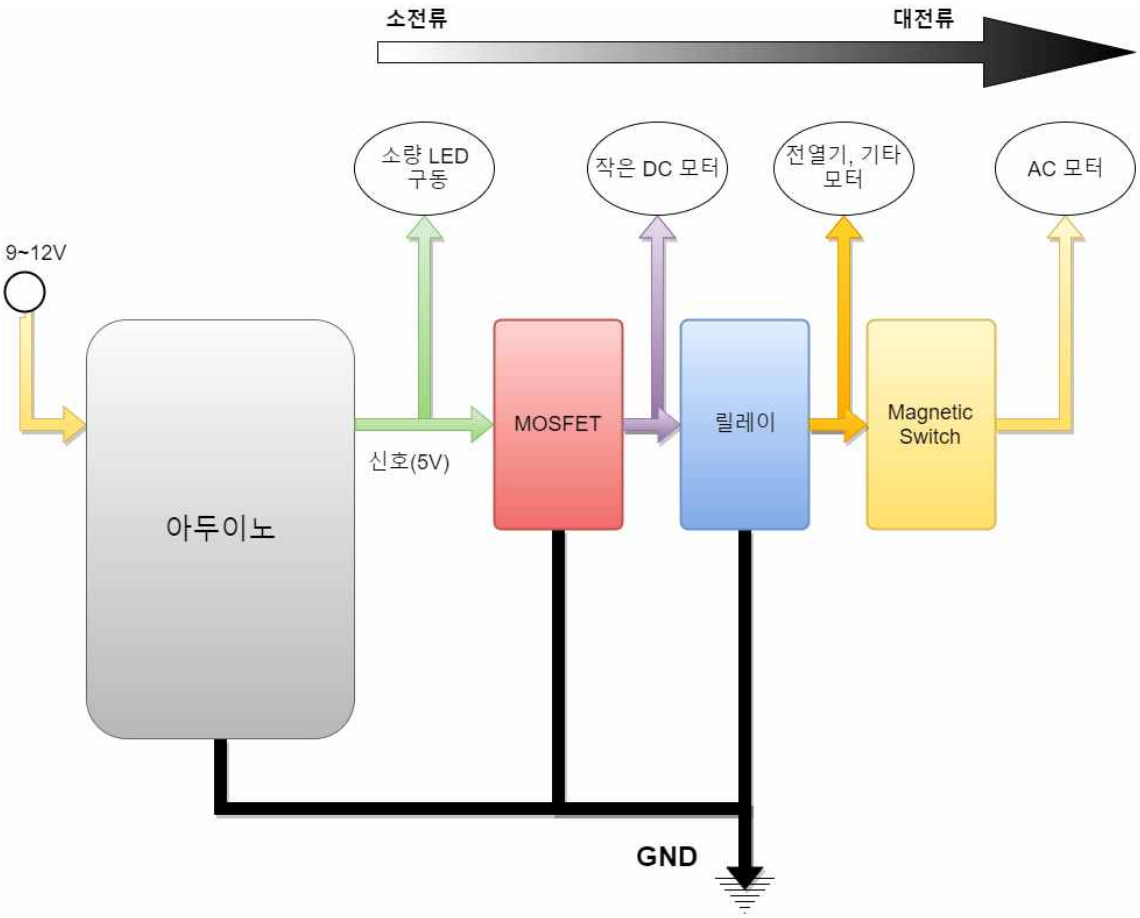
큰 전류가 필요한 출력

모터와 같은 어떤 동작을 하는 출력장치를 액추에이터라고 통칭해서 부릅니다. 이런 종류의 출력장치는 상당히 큰 전력과 전류를 필요로 합니다. 이런 출력장치를 아두이노에 바로 연결하게 되면 아두이노는 내부에서 출력할 수 있는 전원 이상을 끌어낼 수 없기 때문에 프로세서가 정지해 버립니다. 리셋이 되는 현상이 발생합니다.

이런 종류의 출력을 트랜지스터를 이용하거나 릴레이를 이용해서 출력을 증폭시킵니다. 4장 3항의 대전류 출력 예제는 트랜지스터(MOSFET)을 스위치로 이용해서 모터를 구동시킵니다. 트랜지스터를 이용한 증폭으로도 어려울 때는 릴레이를 사용합니다. 릴레이 역시도 사용가능한 정도가 있습니다. 그 이상의 큰 장치를 사용할 때는 그에 맞는 별도의 전기 지식이 필요합니다.

이 경우 M.C. 라고 불리는 Magnetic Contactor 또는 Magnetic Switch를 사용합니다. M.C. 사용은 전기 쪽 일로 3상 AC 모터를 사용할 일이 있을 때 주로 쓰게 됩니다. 일반적으로 아두이노로 작업을 할 때는 DC 모터를 사용하므로 M.C. 까지 신경

쓸 필요는 없습니다만 이것을 말씀드리는 이유는 사용하는 출력장치에 따라 다양한 전력이 필요하고 큰 전력과 전류를 사용해야만 하는 출력장치에는 별도의 출력 증폭용 장치가 필요하다는 것을 말씀드리기 위해서입니다.



6장. 프로그램 방법

막상 아두이노를 배우면 이것저것 많은 함수들을 배우게 됩니다. 그리고 다양한 예제들이 있습니다. 예제들을 보고 필요한 것들을 끼워서 붙이면 프로그램이 된다고 설명하는 글이 많습니다. 틀린 말은 아닙니다만 조금 생각할 부분이 있습니다.

한국어를 하든, 영어를 하든, 일본어나 중국어를 하든 말에는 법칙이 있습니다. 문법이라고 하지요. 문법뿐 아니라 글을 쓸 때 "기-승-전-결"이라는 일반적인 흐름이 있습니다. 프로그램도 그와 비슷합니다.

아주 쉽게 세 가지 단계로 프로그램을 하는 방법을 소개합니다. 복잡한 프로그램은 이렇게 하기는 어렵겠지만 우선 아두이노로 처음 하는 프로그램이라면 이 방법을 익혀 두시면 앞으로 많이 도움이 될 겁니다.

단계1 - [**입력**] 센서를 통한 입력값을 받는다.
단계2 - [**연산**] 받은 값을 사용해서 필요한 정보를 계산한다.
단계3 - [**출력**] 최종 계산 결과를 출력한다.

이게 끝입니다. 물론 조금 복잡하게 프로그램을 하다보면 단계3의 출력값이 다른 단계1의 입력값이 되기도 합니다.

단계1 에서는 주로 아두이노와 연결된 센서의 값을 읽어들입니다. 아두이노에서 사용하는 디지털 센서는 0과 1의 조합으로 된 값이 나오고, 아날로그 센서는 0V에서 5V 사이의 전압값이 나옵니다. 이 값의 범위를 벗어나면 이 범위에 들어맞도록 값을 고쳐줘야 합니다.

단계2에서는 들어온 값을 가지고 필요한 계산을 수행합니다. 이를테면 온도센서의 값으로 0V에서 5V사이의 값을 가져왔다면 그 값을 실제 우리가 알 수 있는 섭씨나 화씨 온도로 변환시킵니다. 계산식은 센서에서 나오는 값과 우리가 얻고자 하는 값사이의 관계식을 만들어서 씁니다. 보통 센서를 만드는 회사에서 이러한 내용을 문서를 만들어서 제공하고 이런 기술문서도 데이터시트에 포함됩니다.

단계3에서는 들어온 값으로 계산이 끝난 값을 가지고 출력을 결정합니다. 출력은 다양한 방법이 사용됩니다. 간단하게는 LED 의 불빛을 켜고 끄는 것도 가능하고, PC와 시리얼통신으로 연결해서 PC 모니터 상에 값을 보여줄 수도 있습니다. 별도로 마련된 LCD 모니터에 값을 보여줄 수도 있고, 따로 연결된 모터를 작동시키거나 릴레이를 작동시켜서 집의 보일러를 켜고 끌 수도 있습니다. 이러한 모든 것이 출력이 됩니다.

1. 입력

아두이노에서 주로 사용되는 입력은 digitalRead() 와 analogRead()입니다. 입력에 사용되는 함수를 쓰기 위해서 먼저 pinMode() 함수로 어떤 핀을 어떤 입력으로 쓸 것인지 결정해 주어야합니다.

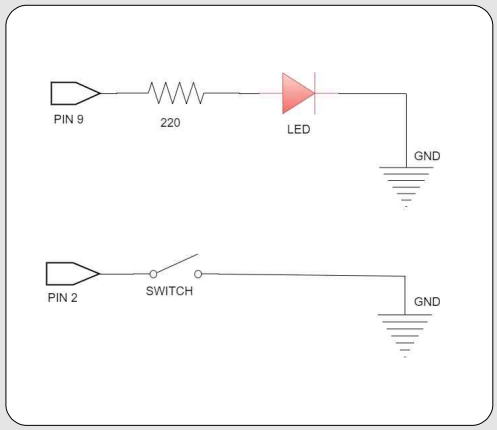
2. 연산

입력된 값을 가지고 원하는 결과를 얻을 수 있도록 다양한 연산을 수행합니다. 이 과정에서 다양한 수학함수들과 문자열함수들, 그리고 for, if 문등이 사용됩니다.

3. 출력

아두이노에서 주로 사용되는 출력은 Serial.print(), Serial.println(), digitalWrite(), analogWrite() 함수입니다. 물론 이 외에도 SD 카드에 저장하거나 LCD 창에 글자나 그림을 쓰는 등의 출력도 가능합니다. 하지만 이런 다양한 출력들도 내부를 잘 살펴보면 위의 4가지 함수를 조합해서 사용하는 경우가 대부분입니다.

Project : 아두이노의 2번 핀에 스위치를 연결하고 스위치를 ON상태로 했을 때 때 9번 핀에 연결된 LED 가 켜지고, 스위치를 OFF 상태로 했을 때 LED 가 꺼지게 하라.



[입력] 스위치 디지털 입력값을 PIN 2 로 받음
[연산]
〉〉 PIN 2 의 값을 buttonState 에 저장
〉〉 buttonState 값을 반전시킴
[출력] buttonState 의 값을 PIN 9 LED 로 출력

```
int buttonState;

void setup() {
    pinMode(2, INPUT_PULLUP);
    pinMode(9, OUTPUT);
}

void loop() {
    buttonState =!(digitalRead(2));
    digitalWrite(9, buttonState);
    delay(1);
}
```

주요함수	필요한 설정
[입력]	
digitalRead(2)	pinMode(2, INPUT_PULLUP);
[연산]	
buttonState = !digitalRead(2)	delay(1);
[출력]	
digitalWrite(9, buttonState);	pinMode(9, OUTPUT);

입력, 연산, 출력 관련 주요 함수를 순서대로 넣어주면 프로그램은 완성입니다.

먼저 [입력] 함수를 씁니다.

```
digitalRead(2)
```

그런데 이 문장을 사용하려면 먼저 이 함수를 사용할 수 있도록 2번 핀을 디지털입력으로 설정해 주어야 합니다. 그리고 설정은 한번만 하면 됩니다. 한번만 하면 되는 것은 setup() 에 넣어줍니다. 그리고 digitalRead(2) 는 계속 하는 것이니까 loop()에 넣어줍니다.

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

void loop() {
    digitalRead(2);
}
```

이제 [입력] 부분이 끝났으니 [연산] 부분으로 넘어갑니다. 연산에서는 읽은 값을 반전시켜 buttonState 에 저장합니다.

digitalRead(2) 를 반전시켜서 저장합니다.

```
buttonState = !(digitalRead(2));
```

그리고 loop() 함수를 한번씩 실행할 때마다 1ms 씩 멈추도록 합니다.

```
delay(1);
```

여기까지 한 것을 모아보면 다음과 같습니다.

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

void loop() {
    buttonState = !(digitalRead(2));
    delay(1);
}
```

이제 마지막으로 [출력] 부분을 프로그램합니다. 출력은 buttonState 의 값을 13번 핀으로 그대로 내보냅니다.

```
digitalWrite(9, buttonState);
```

13번 핀을 출력으로 사용하려면 먼저 13번 핀을 출력으로 쓰겠다고 아두이노에 알려줘야 합니다.

```
pinMode(9, OUTPUT);
```

출력 부분을 포함해서 전체 프로그램을 완성시켜 봅시다.

```
void setup() {
    pinMode(2, INPUT_PULLUP);
    pinMode(9, OUTPUT);
}

void loop() {
    buttonState = !(digitalRead(2));
    delay(1);
    digitalWrite(9, buttonState);
}
```

설정에 필요한 pinMode() 함수는 모두 1번만 처음에 실행시키면 되는 것이므로 모두 setip() 안에 넣습니다. 그리고 **digitalWrite(9, buttonState)** 는 입력값을 받고 연산을 거친 다음에 출력이 이루어져야 하기 때문에 loop() 함수의 맨 아래쪽에 넣습니다.

이렇게 프로그램이 완성되었습니다.

수고하셨습니다. 여기까지 공부하신다고 고생하셨습니다. 이후의 과정은 홈페이지를 통해 계속해서 공개하고 공유하도록 하겠습니다.

함께 이 책을 수정하고 지속적으로 업데이트해 나가기를 원하시는 분이 계시면 연락주시면 고맙겠습니다.

내용은 어떻게든 만들고 설명할 수는 있지만 문서로 만들고, 책처럼 만든다는 것이 쉽지 않았습니다. 별도의 편집프로그램없이 한글오피스 만으로 이 문서를 만들었습니다. 인디자인을 사용해보고 싶은 마음은 굴뚝같았지만 아직 사용해 본 적이 없어서 며칠 시도하다가 포기했습니다. 혹시 문서편집 프로그램을 사용하시면서 그래픽 작업도 가능한 분이 계시면 함께 이 책의 다음번 업데이트를 했으면 좋겠습니다.

정보는 공유될 때 풍성해집니다.

http://winduino.co.kr

2016년 5월 6일,

아직 찾지 못한 9명의 세월호 실종자를 기억하며,

"바람"이 아두이노를 공부하는 모든 분들에게 이 책을 드립니다.