

Arduino Prime Kit

Manual

- 2019년 1학년 수업에서는 다음의 여러 가지 부품 중에서 1, 2학기로 나누어 실험에 필요한 부품과 모듈만 제공한다.
- 제공된 부품과 모듈은 파손이나 분실 시 실험을 진행하는데 어려움이 있으니 절대 파손이나 분실을 하지 않도록 조심한다.

아두이노 프라임 키트 (Arduino Prime Kit)

Objectives

본 장에서는 실습 키트 소개 및 특징을 살펴본다.

1. 아두이노 프라임 키트 소개
2. 아두이노 프라임 키트 실습 부품
3. 아두이노 프라임 키트 구성품

1. 아두이노 프라임 키트 소개

1-1. 아두이노 프라임 키트 소개



아두이노 프라임 키트는 Arduino Uno R3를 기반으로 한 아두이노 플랫폼 기반 프로그래밍 교육용 키트이다.

I/O 부품, 센서 모듈, 확장 쉴드, 각종 전자 소자를 활용해서 기초 전자 실습부터 각종 센서 모듈 제어까지 제어할 수 있으며 원하는 센서를 조합하여 원하는 기능의 프로젝트도 구성이 가능하다.

아두이노 프로그래밍에 대한 이해 및 동작원리를 알 수 있으며 다양한 센싱 실습으로 폭 넓은 프로젝트 구성이 가능한 키트이다.



[아두이노 프라임 키트 구성]





- 2017년 2학년 수업에서는 위의 여러 가지 부품 중에서 실험에 필요한 것을 1, 2학기로 나누어 제공한다.

1-2. 아두이노 프라임 키트 스펙

MCU 보드	<ul style="list-style-type: none"> - MCU : ATmega328P - Core Processor : 8bit AVR - Operating Voltage : 5V - Input Voltage(Recommended) : 7 ~ 12V - Input Voltage(limits) : 6 ~ 20V - Digital I/O Pins : 14 (6 provide PWM output) - Analog Input Pins : 6 - DC Current per I/O Pin : 40mA - DC Current for 3.3V Pin : 50mA - Flash Memory : 32KB (0.5KB used by bootloader) - SRAM : 2KB - EEPROM : 1KB - Clock Speed : 16MHz
센서 모듈	<ul style="list-style-type: none"> - 텍스트 LCD, 1채널 릴레이, RTC, RGB LED, 조이스틱 - 서보 모터, 스텝 모터, 7-Segment(FND), Dot Matrix - 사운드 센서, 온습도 센서, 수위측정 센서, 스위치 센서 - 광 센서, 기울기 센서, 온도 센서, 불꽃 감지 센서 - 적외선 수신기, 초음파 센서
통신 모듈	<ul style="list-style-type: none"> - RFID, 블루투스
셸드 모듈	<ul style="list-style-type: none"> - 이더넷 셸드, 우노 프로토타입 셸드, 센서 셸드

2. 아두이노 프라임 키트 실습 부품

2-1. 아두이노 프라임 키트 보드 & 쉴드

Main MCU		
	Arduino Uno R3	<ul style="list-style-type: none"> - ATmega328P-PU - 8bit AVR - 14 Digital Pins - 6 Analog Input Pins - 32KB Flash Memory / 2KB SRAM - 16MHz Clock Speed - 5V Operating Voltage
쉴드(SHIELD)		
	Ethernet Shield [SPI방식]	<ul style="list-style-type: none"> - Ethernet 통신용 쉴드 - Arduino Uno R3의 SPI핀과 연결 - Arduino를 인터넷에 연결
	Uno Prototype Shield	<ul style="list-style-type: none"> - 센서 또는 모터 등 작동 부품 연결 - 브레드 보드에 와이어 링으로 회로구성 - 와이어 링 결과물 보관 및 참조 용이
	Sensor Shield	<ul style="list-style-type: none"> - 디지털 및 아날로그 핀 단자 - I2C/UART 인터페이스 - 브레드 보드 없이 센서 및 부품을 plug & play 형식으로 연결 가능 - 점퍼를 사용해서 IIC, COM 선택

2-2. 아두이노 프라임 키트 구성 통신 모듈 & 전자 소자

[통신 모듈 2종]

Item		Item	
	블루투스 [Serial방식]		RFID 모듈, 태그 2종 [SPI방식]

[전자 소자 5종]

Item		Item	
	LED 3종 각 5개		저항 3종 1kΩ, 10kΩ, 220kΩ 각 10개씩
	가변 저항 [ADC방식]		키 버튼 4개 [GPIO방식]
	8비트 쉬프트 레지스터 칩		

2-3. 아두이노 프라임 키트 구성 센서 모듈

[센서 모듈 20종]

Item		Item	
	서보 모터 [PWM방식]		스텝 모터 [PWM방식]
	온습도센서 [ADC방식]		조이스틱 모듈 [ADC+GPIO방식]
	RGB LED [PWM방식]		7-세그먼트 [GPIO방식]
	키패드 [GPIO방식]		도트 매트릭스 [GPIO방식]
	초도 센서 [ADC방식]		초음파 센서 [GPIO방식]
	Text LCD [I2C방식]		사운드 감지 센서 [ADC방식]

[센서 모듈 20종]

Item		Item	
	적외선 수신기 [PWM방식]		온도 센서 [ADC방식]
	불꽃 감지 센서 [ADC방식]		1채널 릴레이 모듈 [GPIO방식]
	RTC 모듈 [방식]		수위 측정 모듈 [ADC방식]
	기울기 센서 [GPIO방식]		수동, 피에조 부저 [PWM/GPIO방식]

* 상기 이미지의 센서는 실제 모양과 다소 차이가 있을 수 있으며, 사전 예고 없이 변경될 수 있습니다. *

3. 아두이노 프라임 키트 액세서리

		
830 point 브레드보드 X1	400 point 브레드보드 X1	170 point 미니 브레드보드 X2
		
브레드보드 전원공급 장치 3.3V / 5V	브레드보드용 점퍼 와이어 65개	뒤풍 female to female 케이블 20cm 40개
		
뒤풍 male to female 케이블 20cm 40개	9V 배터리 전용 스냅	B타입 USB 케이블
		
이더넷 실드용 케이블	12V 1A 전원 어댑터	적외선 리모트 컨트롤러 (코인 배터리 포함)

* 상기 이미지의 액세서리는 실제 모양과 다소 차이가 있을 수 있으며, 사전 예고 없이 변경될 수 있습니다. *

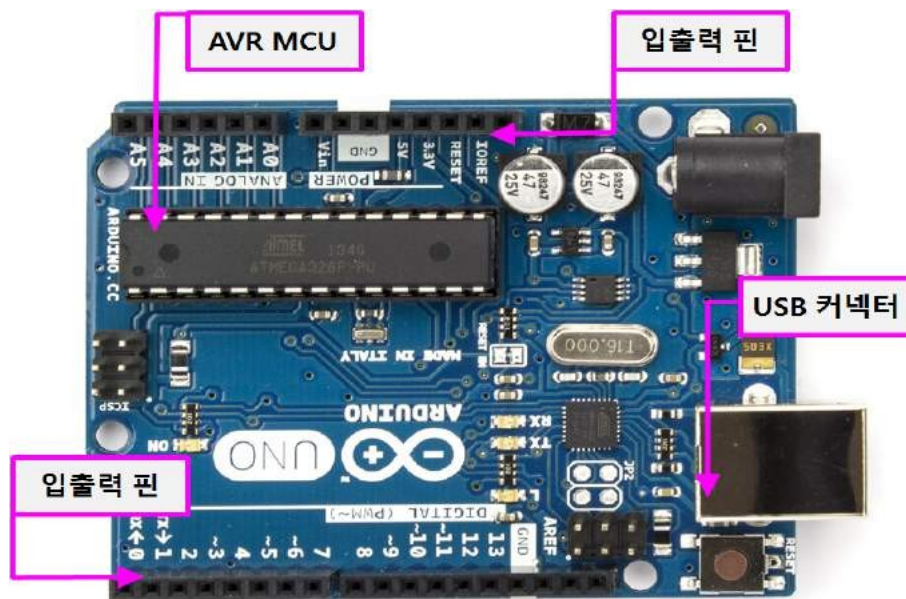
4. 아두이노 구성

2-1. 아두이노(Arduino) 구성 요소



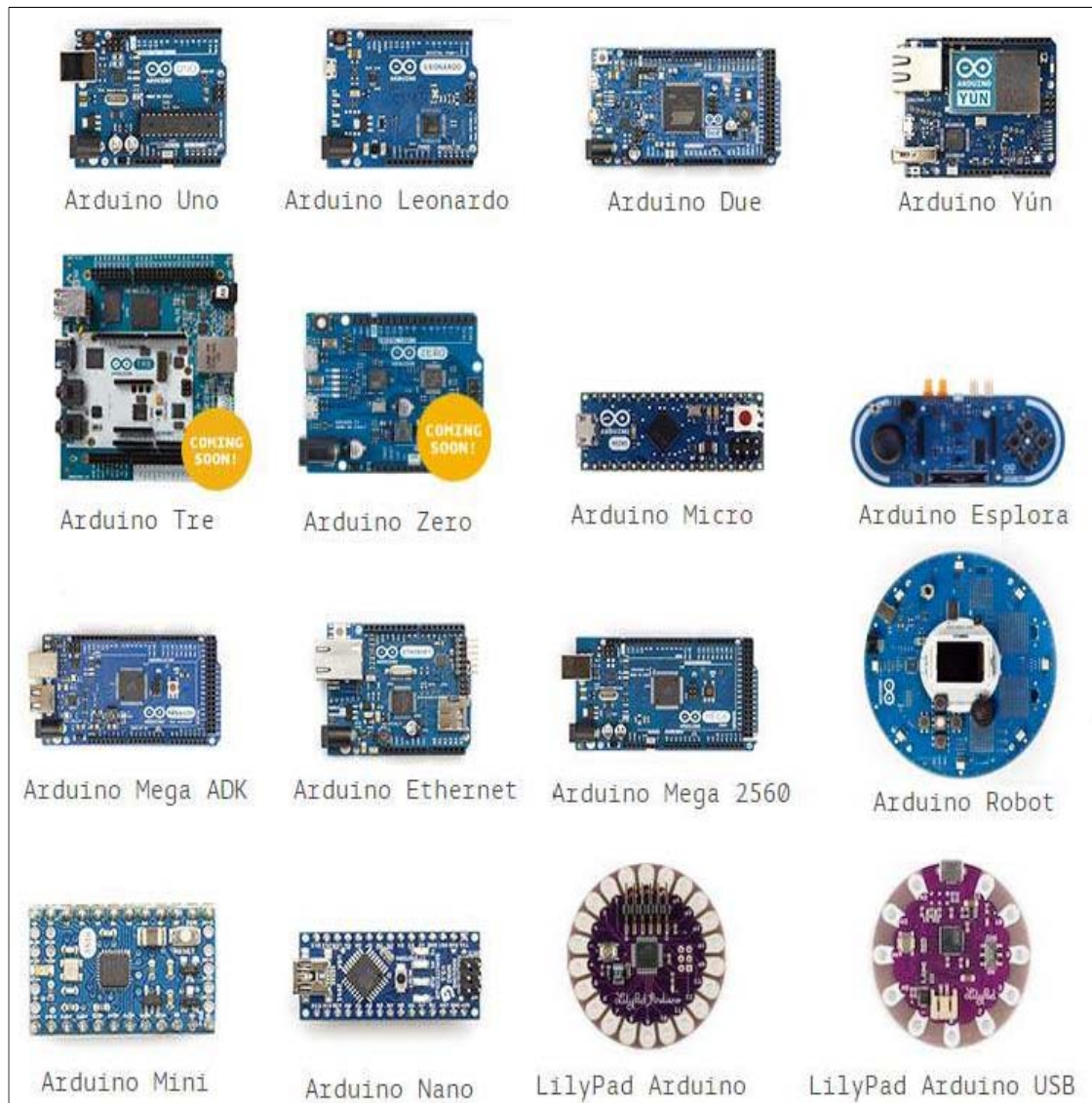
[Hardware Board]

Atmel사의 AVR 8비트, 32비트 Microcontroller와 USB 커넥터, 입출력 핀들로 구성되어 있다.



[Arduino Uno R3]

초기에는 8비트 AVR인 ATmel16, 32, 256등으로 개발이 되었고 최근 들어서는 32비트 ARM Cortex-M3 코어의 제품도 개발이 되고 있다.

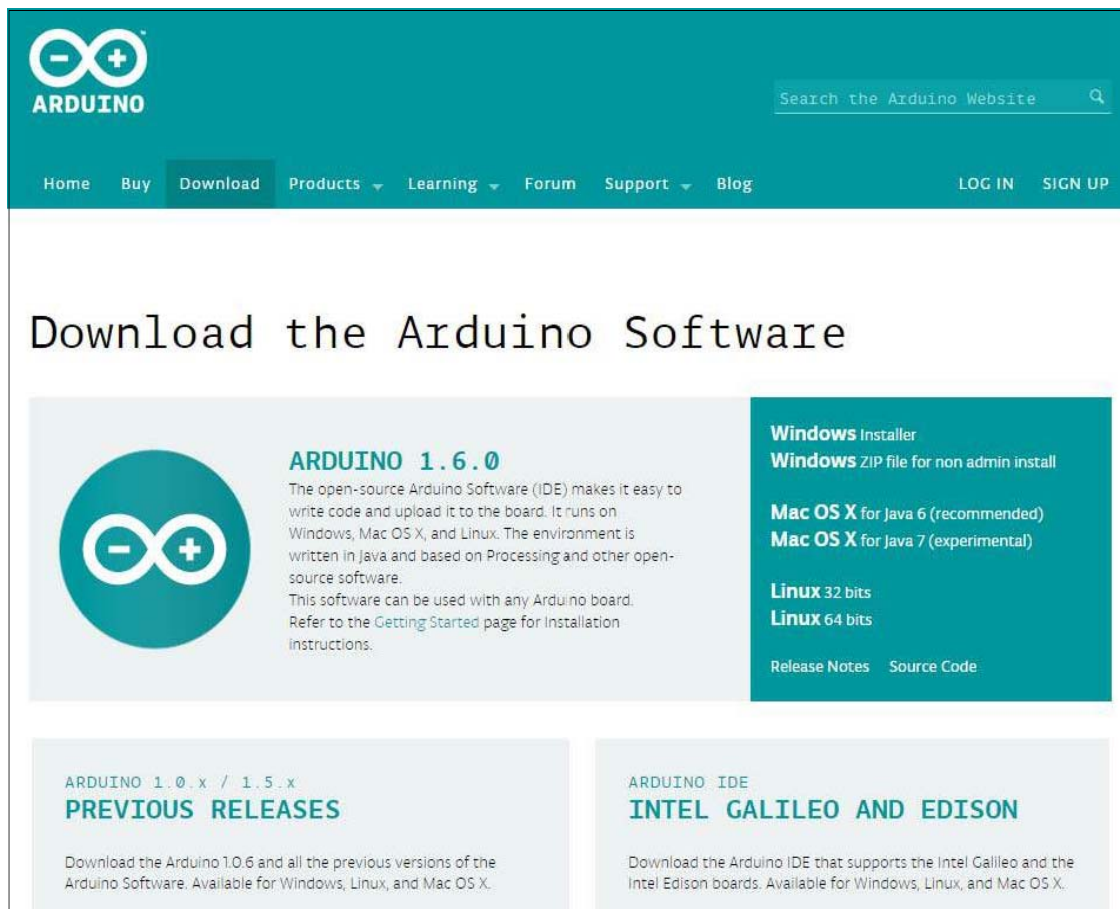


[다양한 Arduino board]

아두이노(Arduino)는 USB커넥터를 통해서 개발 PC와 연결이 되며 별도의 추가 장비 없이 개발 PC에서 작성한 프로그램을 보드의 플래시 메모리에 로드 할 수 있다. 일반 임베디드 프로그래밍의 프로그램 로딩의 복잡한 과정 및 추가 장비가 필요 없이 쉽게 개발 할 수 있다.

[Software IDE]

스케치(Sketch)라는 이름의 IDE는 아두이노(Arduino) 공식 사이트 arduino.cc에서 다운 받을 수 있다. 프로그램 개발, 컴파일, 업로드, 시리얼 통신 기능까지 제공이 된다. 이 툴 하나로 개발에 필요한 모든 것이 가능하다.



[아두이노 공식 사이트 - Download 부분]

현재 1.8.0 버전의 Sketch IDE가 발표되었다.

이전의 보드 종류에 따라서 다른 버전의 IDE를 사용해야 하지만, 1.6.0버전부터는 보드 종류와 상관없이 사용이 가능하다.

IDE는 개발 PC의 운영체제에 맞게 다운 받아 압축만 해제하면 바로 사용이 가능하다.

[Standard Library]

▶ Standard Library

대부분의 보드에서 사용하는 공통된 기능에 대해서는 라이브러리를 제공하고 있으며 이것을 표준 라이브러리라 한다.

- 위치 : Sketch IDE를 다운 받은 폴더 내에 libraries폴더
- 참고 : 공식 사이트 <http://arduino.cc/en/Reference/Libraries>
- 종류 : 13개 표준 라이브러리와 보드 종류별 라이브러리 제공

라이브러리 명	특징
EEPROM	영구 저장소 EEPROM에 데이터 쓰기 읽기 기능 제공
Ethernet	Ethernet shield와 아두이노 보드를 연결해서 Internet 기능 제공
Firmata	PC와 아두이노 보드 사이의 통신 기능 제공
GSM	GSM Shield와 아두이노 보드를 연결해서 GPRS 무선 통신 기능 제공
LiquidCrystal	LCD 제어 기능 제공
SD	SD Card 제어 기능 제공
Servo	Servo 모터 제어 기능 제공
SPI	SPI 통신 기능 제공
SoftwareSerial	일반 핀을 시리얼 통신용으로 사용할 수 있는 기능 제공
Stepper	Step 모터 제어 기능 제공
TFT	TFT 스크린 제어 기능 제공
WiFi	WiFi shield와 아두이노 보드를 연결해서 Wifi통신 기능 제공
Wire	TWI/I2C 통신 기능 제공

이 외에도 보드별 특별하게 제공되는 라이브러리들도 존재한다.

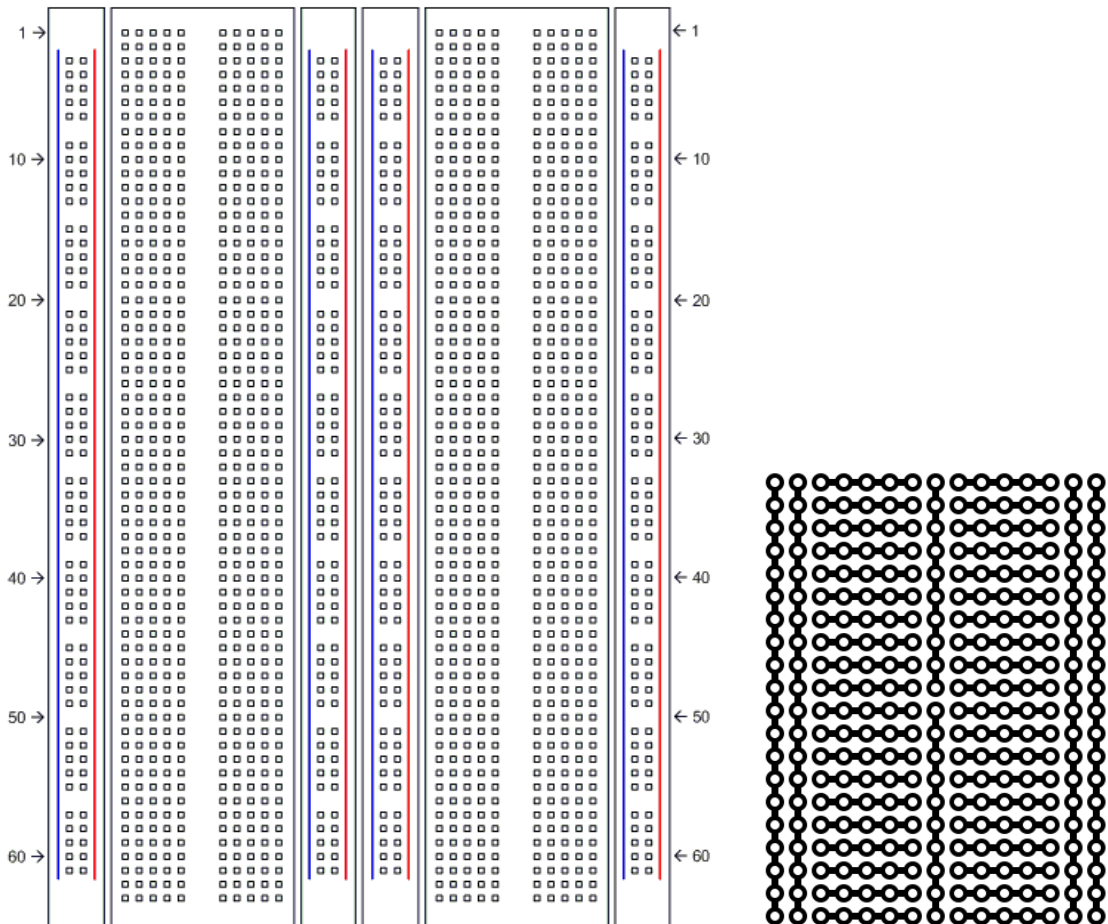
실험에 필요한 기초 지식

Objectives

본 장에서는 아두이노를 사용하여 실험을 위한 간단한 기초 지식을 배울 수 있다.

1. 빵판(브레드 보드) 활용법

▶ 빵판(브레드보드 : Breadboard)란?



[브레드 보드 외관 & 내부 모습]

빵판(브레드 보드)이란 회로의 개발을 위해 사용하는 기판으로 납땜을 해서 한번만 사용하는 회로가 아니라 여러 번 사용 할 수 있다.

위 그림에서 빨간 선으로 표시된 부분과 파란색 선으로 이루어진 부분은 일반적으로 전원 연결선으로 사용하고 빨간 선은 (+) 전원 공급, 파란 선은 (-)접지로 사용한다.

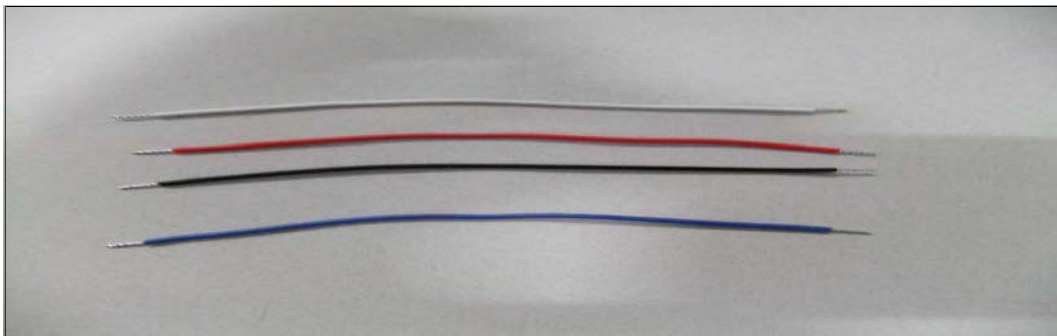
오른쪽 그림처럼 전원 연결 부분은 내부적으로 세로로 연결 되어 있고 나머지 부분은

가로로 연결되어 있다. 다시 말해서 가로줄은 전류가 통하지만 세로줄은 전류가 통하지 않는다. (전원 연결선은 세로만 연결되어 있음)

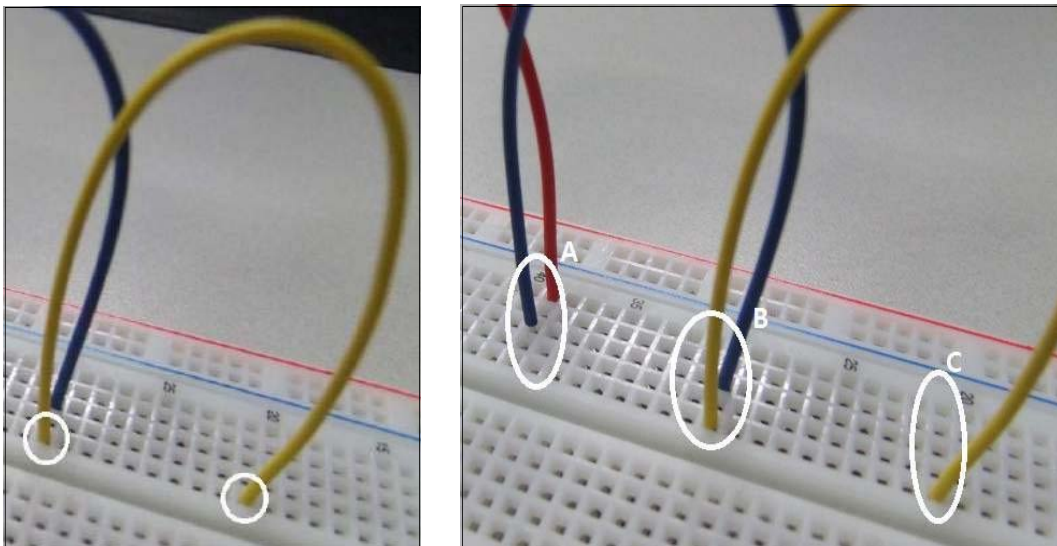
▶ 빵판(브레드보드 : Breadboard)에 부품 조립하기

▶ 전선 연결하기

우선 빵판(브레드보드)에 연결할 전선의 양 끝의 피복을 벗겨낸다. (실습 키트의 전선은 이미 피복이 벗겨진 상태임)

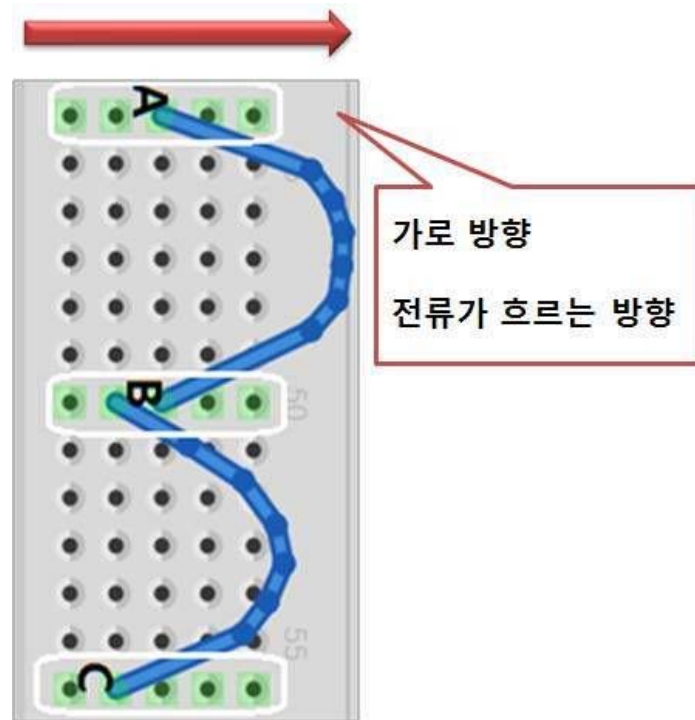


아래 그림처럼 피복이 벗겨진 부분을 브레드보드의 구멍에 끼워 넣는다.



[선 연결 - Wiring]

가로 줄에 전류가 흐르고 있어서 같은 가로 줄에 전선을 꽂으면 서로 연결되어서 전기가 통할 수 있는 상태이다.

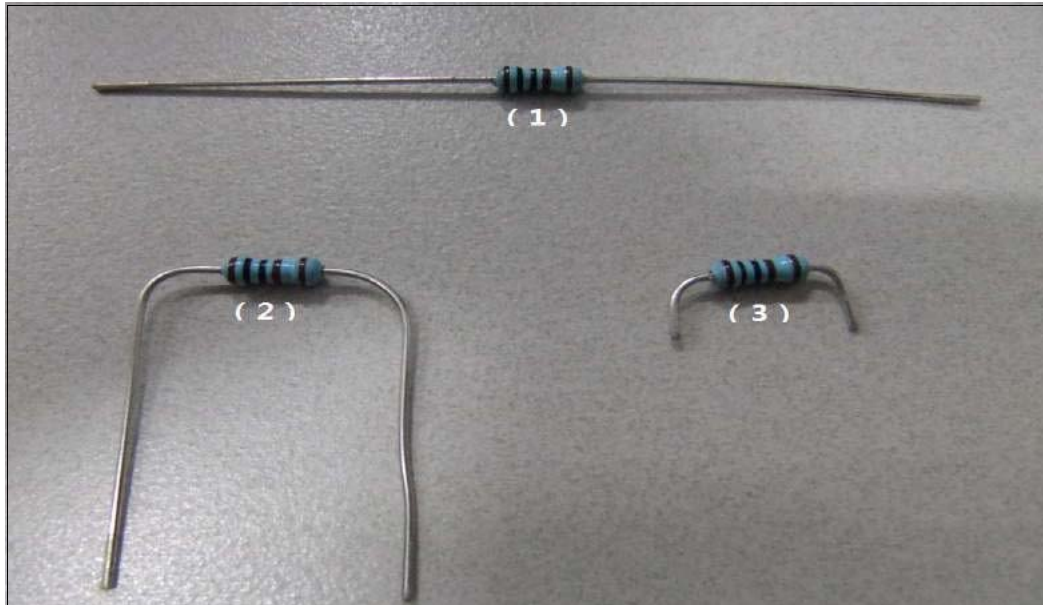


[연결하여 선 꽂기]

그래서 $A \rightarrow B \rightarrow C$ 전선은 모두 연결된 상태가 된다.

▶ 저항 소자 연결하기

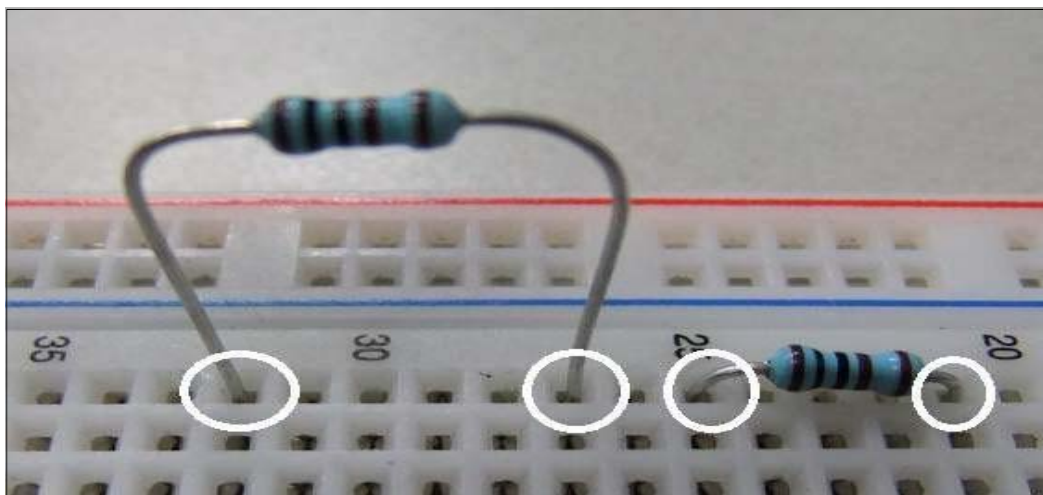
우선 준비된 저항을 (1) → (2) → (3) 의 순서로 모양을 만든다. 여기서 부품을 삽입할 때 (2), (3) 둘 다 사용해도 상관없다.



[저항 형태 만들기]

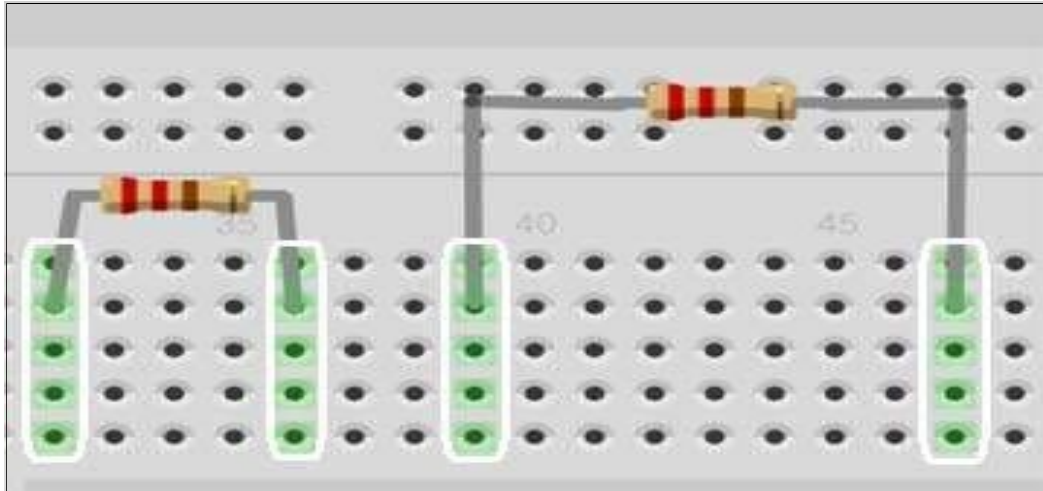
실제로 부품을 브레드 보드에 삽입하면 아래와 같은 형태로 삽입한다.

○ 모양의 부분에 실제 부품을 브레드 보드에 삽입한 부분이다.



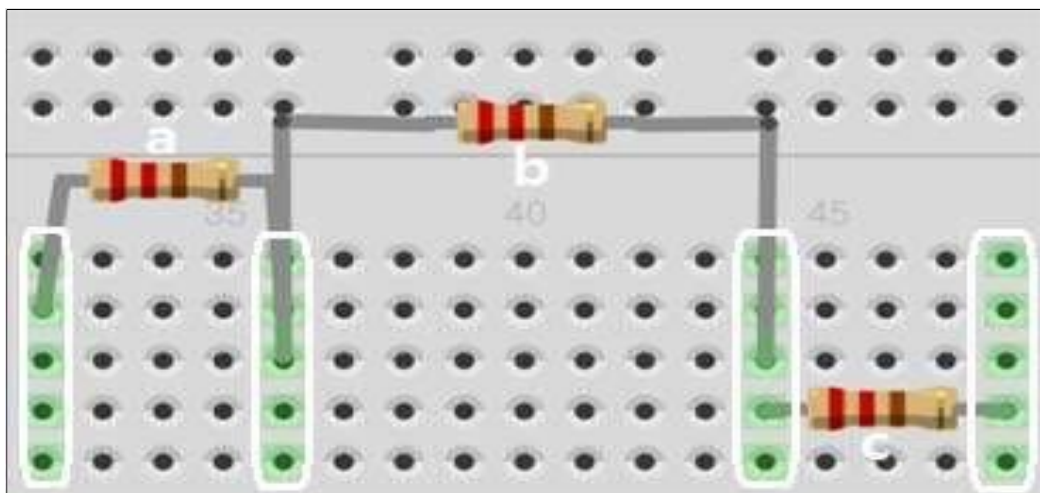
[저항 연결 하기]

내부적으로는 아래 그림처럼 저항이 조립된 상태고 5개의 슬롯 즉, 가로줄이 연결된 상태가 된다.



[저항 연결]

저항 부품을 5개 슬롯에 연결해서 조립한다면 아래처럼 연결 되었다고 볼 수 있다. 저항 a, b, c가 모두 직렬로 연결된 상태이다.



[저항의 직렬 연결]

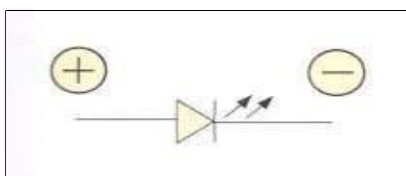
▶ LED 연결하기

LED 연결 시에는 극성에 주의해야 한다.



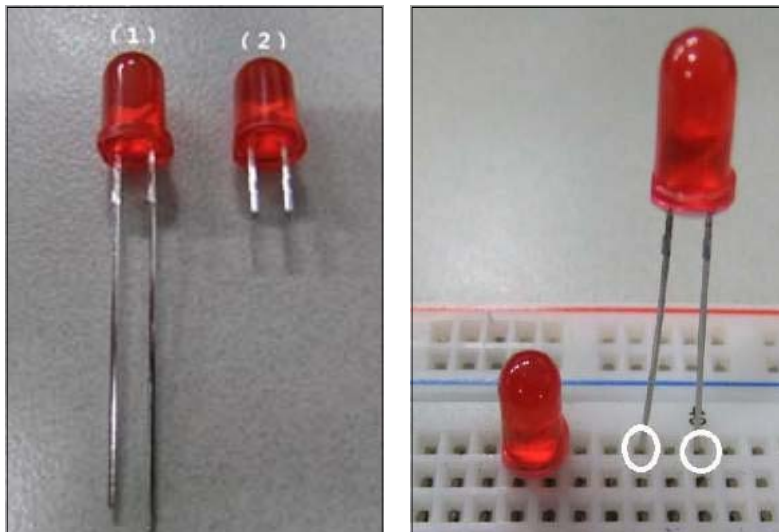
[LED의 극성 구분법]

위에 그림에서 다리가 짧은 쪽이 (—)극, 긴 쪽이 (+)극이다.
다리로 구분을 할 수 없을 경우 LED 캡 안의 ㄱ자 모양을 보고 판단한다. ㄱ자 모양이 (—)극, 반대쪽이 (+)극이 된다.]



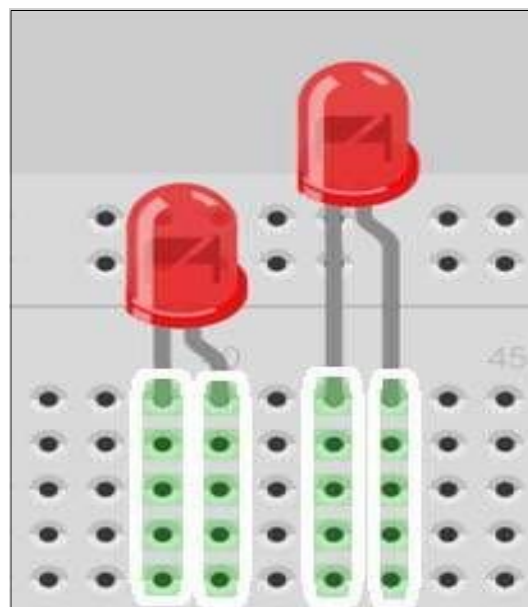
LED를 회로도에서 표시할 경우는 왼쪽 그림처럼 표시되며 극성을 구분할 수 있다.

LED 소자를 (1) → (2) 순서로 준비하자.



[LED 준비 및 브레드보드에 배선]

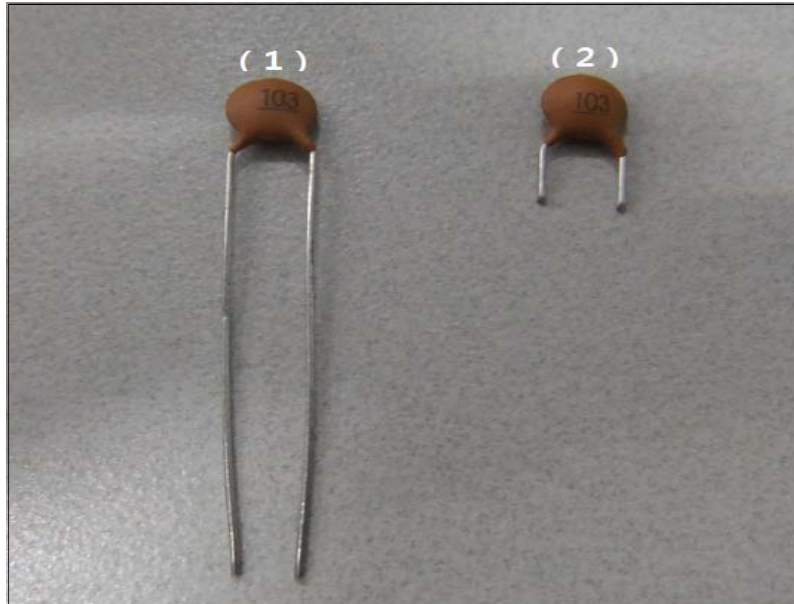
내부적으로는 아래의 그림처럼 연결되어 있다.



[LED 연결 내부]

▶ 콘덴서 연결하기

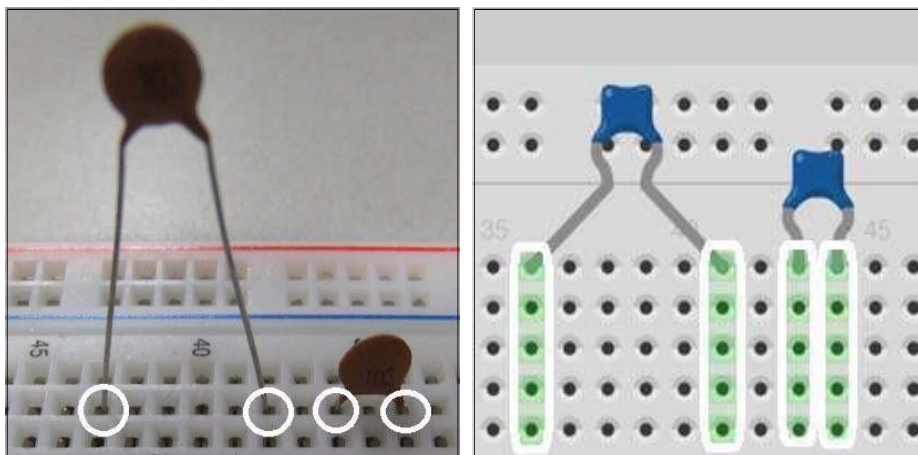
세라믹 콘덴서의 경우에는 극성이 없다.



[세라믹 콘덴서]

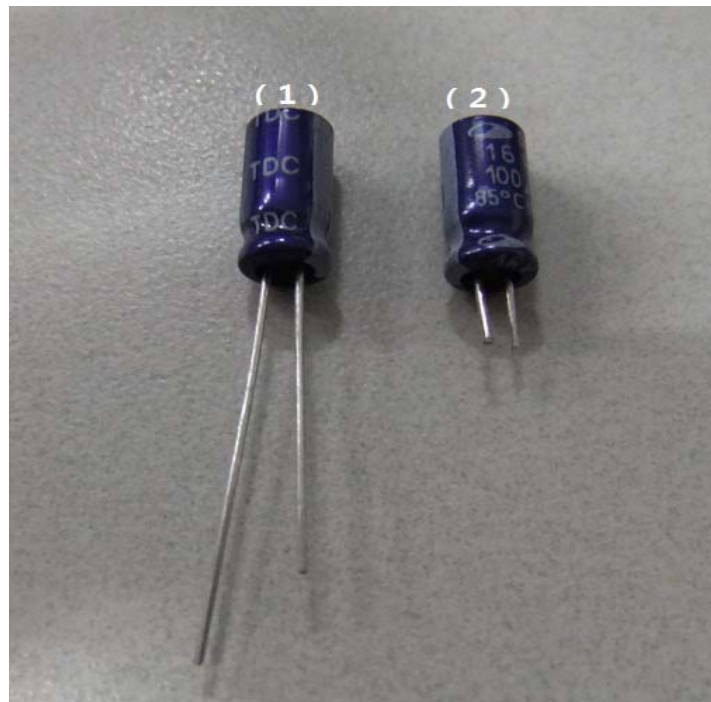
아래와 같이 방향에 상관없이 연결하면 된다.

실제 연결된 모습은 왼쪽과 같고 내부적으로는 아래와 같이 연결되어 있다.



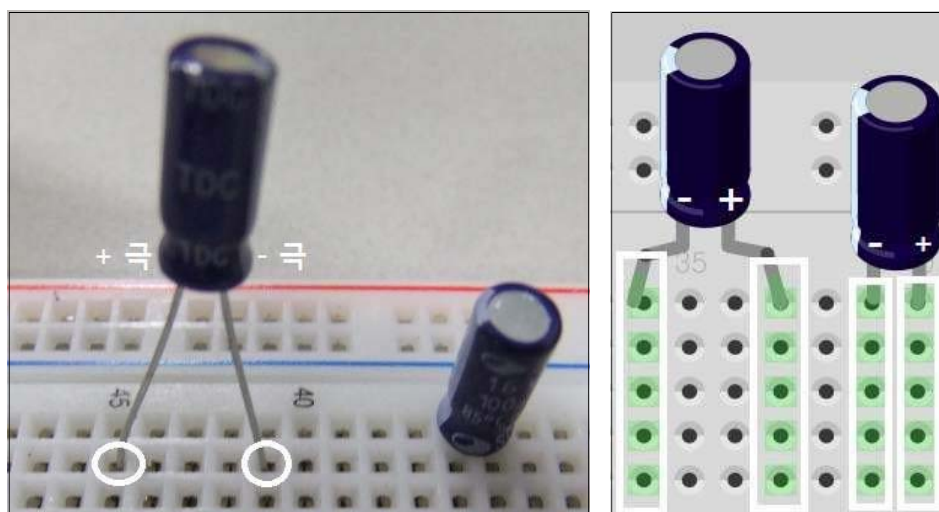
[세라믹 콘덴서 연결]

전해 콘덴서의 경우에는 극성이 있어서 사용 시 극성에 유의해야 한다.



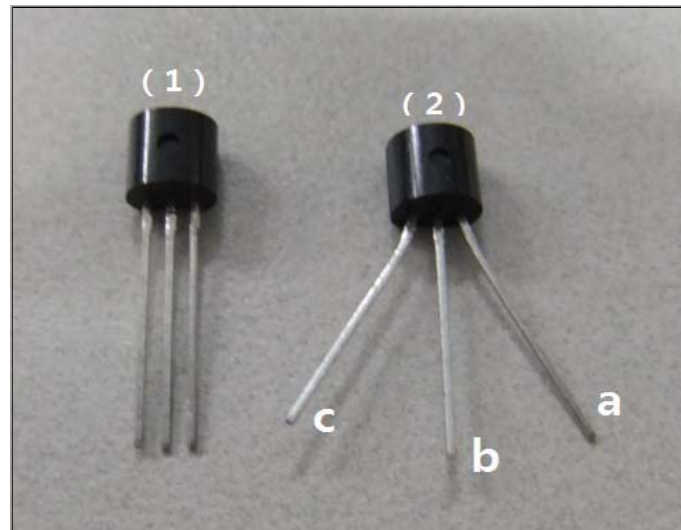
[전해 콘덴서]

최초 부품 (1)번에서 소자 다리가 짧은 쪽이 (-)극, 긴 쪽이 (+)극이다. 소자다리를 짧은 (2)의 경우에는 콘덴서 옆에 프린트된 것을 보고 확인을 할 수 있다. (-)라고 프린트 된 쪽이 (-)극이며 아무 표시가 되어 있지 않은 쪽이 (+)극이다.



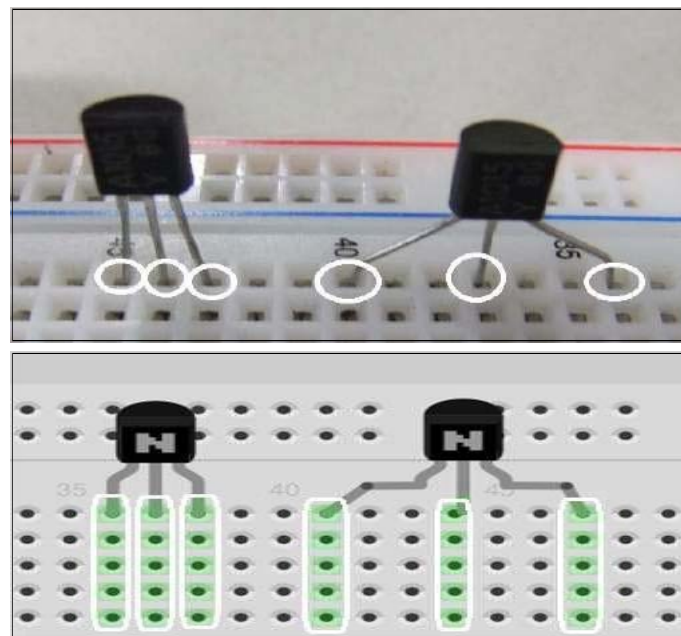
[전해 콘덴서 연결]

▶ 트랜지스터 연결하기



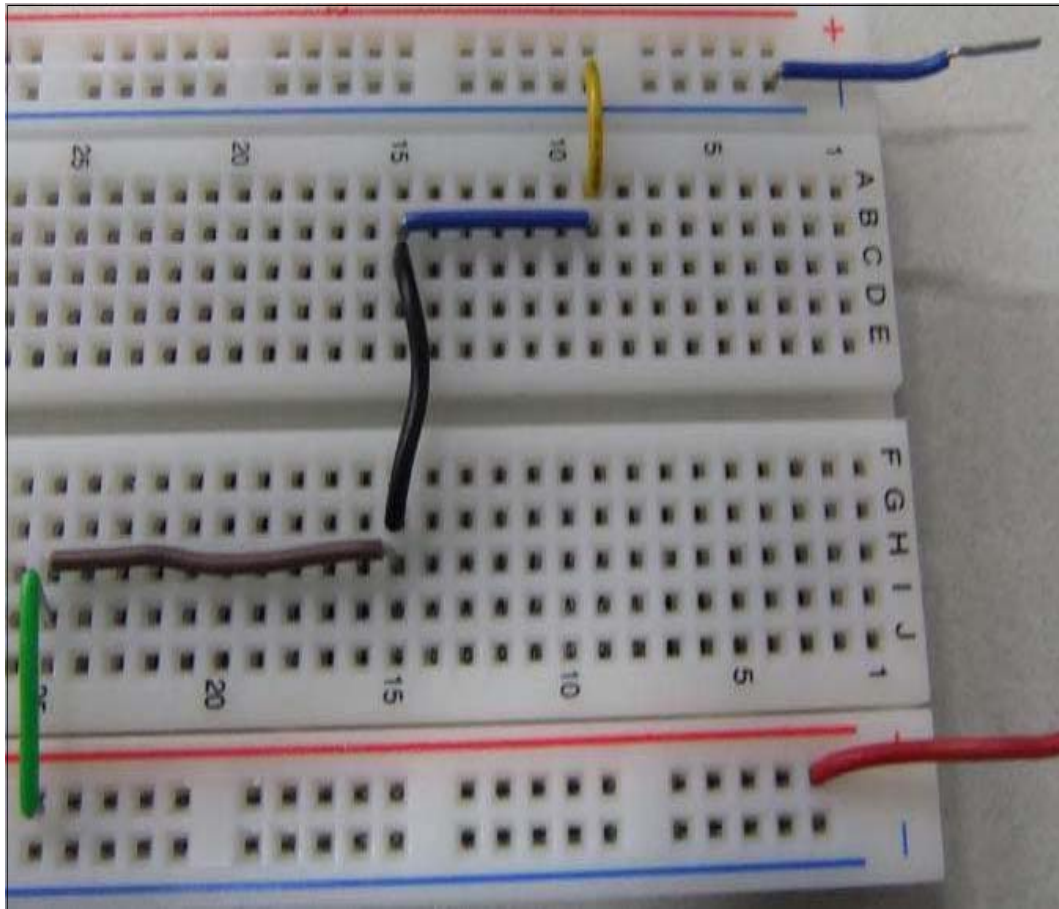
[트랜지스터]

트랜지스터의 경우에는 소자의 다리가 3개로 각각 에미터, 베이스, 콜렉터로 이루어져 있으며 각각의 트랜지스터마다 에미터, 베이스, 콜렉터의 위치가 다르다.



[트랜지스터 연결]

트랜지스터를 실제로 브레드 보드에 연결하면 위의 그림처럼 표현할 수 있다



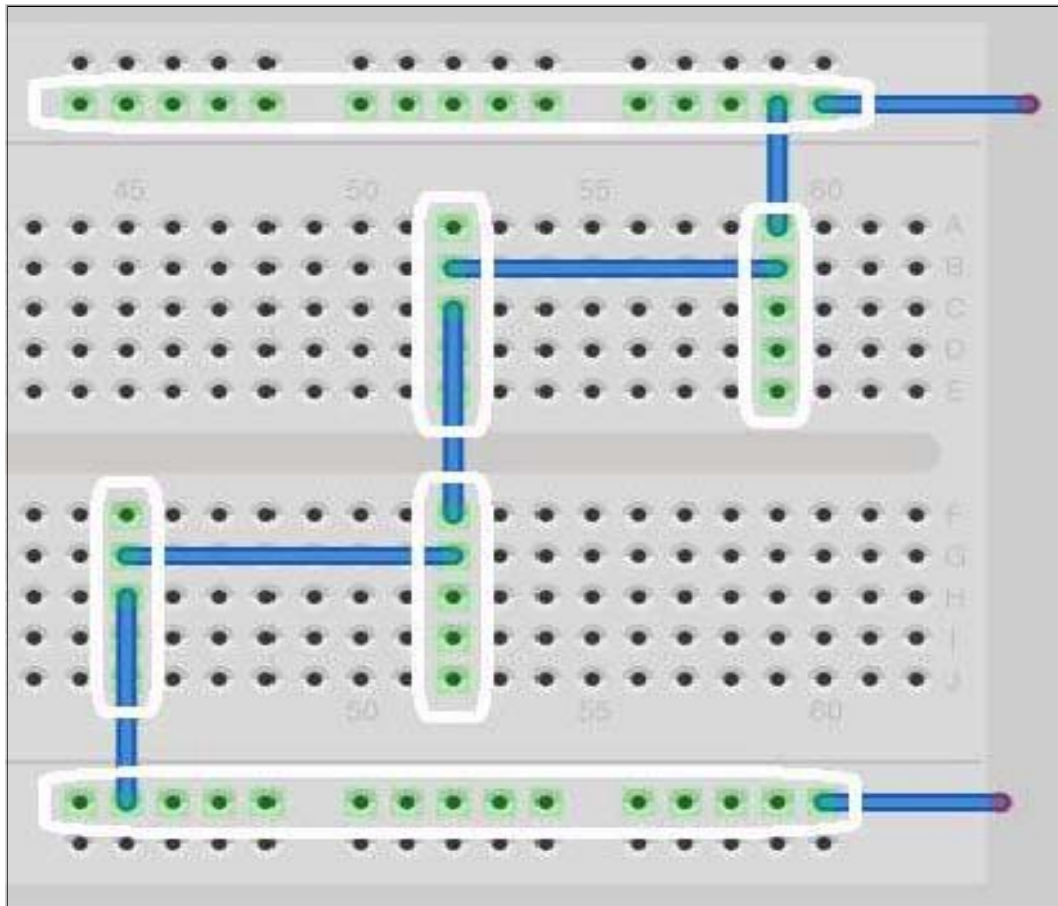
[브레드 보드 전선 연결]

그림처럼 전선이 연결되어 있을 때 전기를 흘려주면 내부적으로 어디까지 통할까?

빵판(브레드 보드)의 특성을 이해했다면 쉽게 파악할 수 있다.

그림에서 영문으로 쓰인 부분은 5개의 슬롯끼리만 연결 되어 있고 + , - 가 표시된 부분은 같은 라인끼리 연결 되어 있다.

아래의 그림은 내부적으로 연결된 상태를 표시한 것이다.

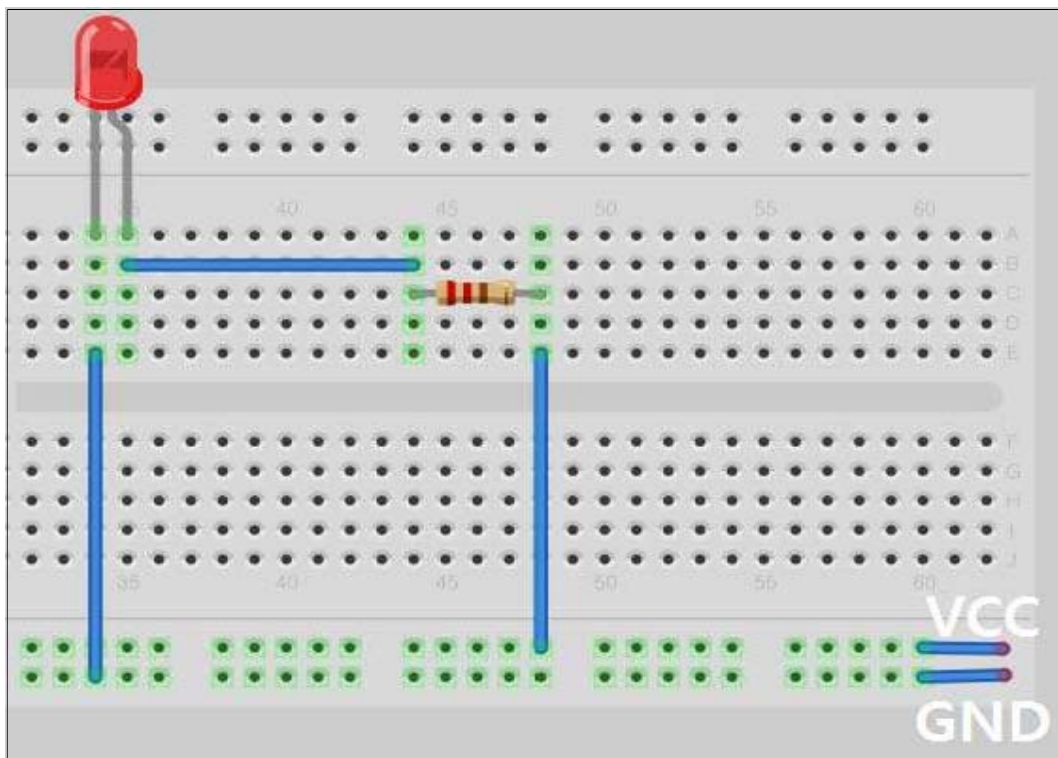


[브레드 보드 전선 연결]

▶ **빵판(브레드보드 : Breadboard)을 이용하여 LED 켜기**

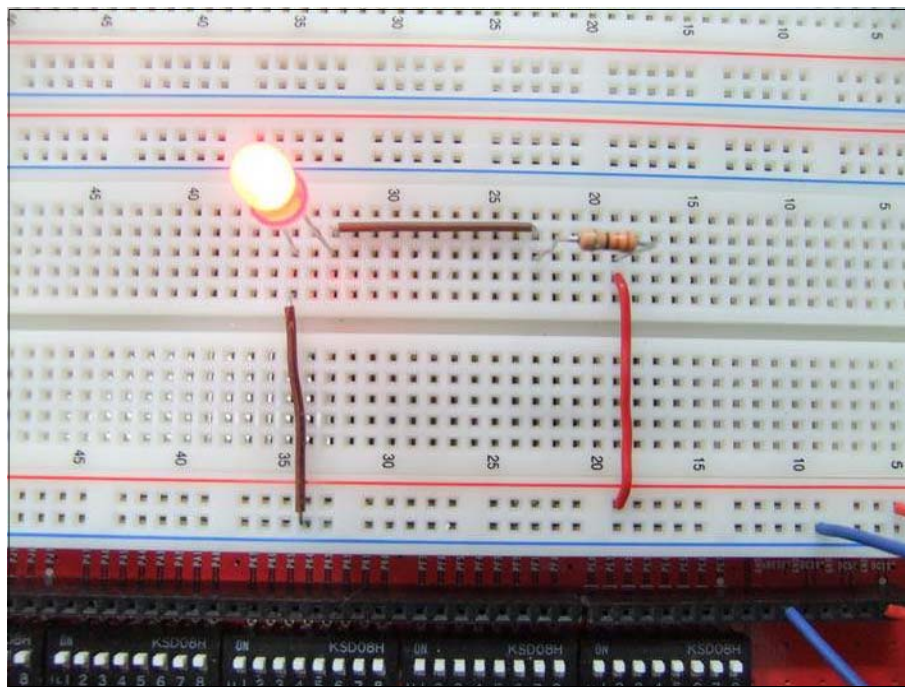
빵판(브레드보드)에 저항과 LED만을 통해서 LED에 불이 들어오도록 해보자.

내부적인 선의 연결과 저항의 역할 LED의 극성을 이해하였으니 충분히 가능할 것이다.



[브레드 보드를 활용한 LED 켜기]

위와 같이 빵판(브레드 보드)에 LED와 저항을 연결하여 LED가 켜지는지 확인해 보자.



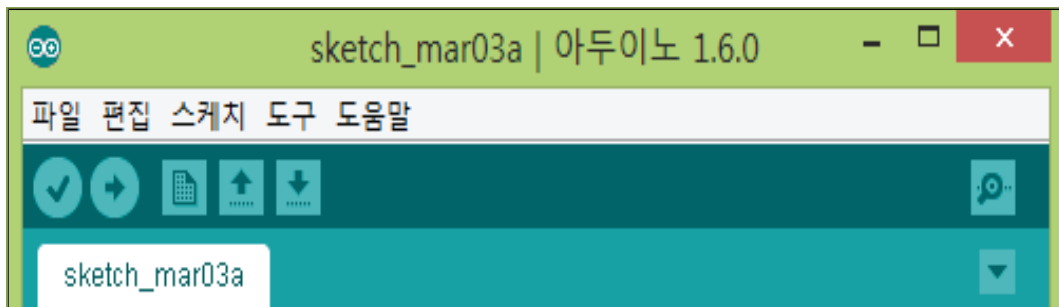
[브레드 보드로 LED 켜기]

Integration Development Environment

Objectives

본 장에서는 아두이노를 제어하기 위한 디바이스 드라이버 설치, 보드 세팅, 툴 설치를 살펴본다.

▶ 메뉴 아이콘 살펴보기



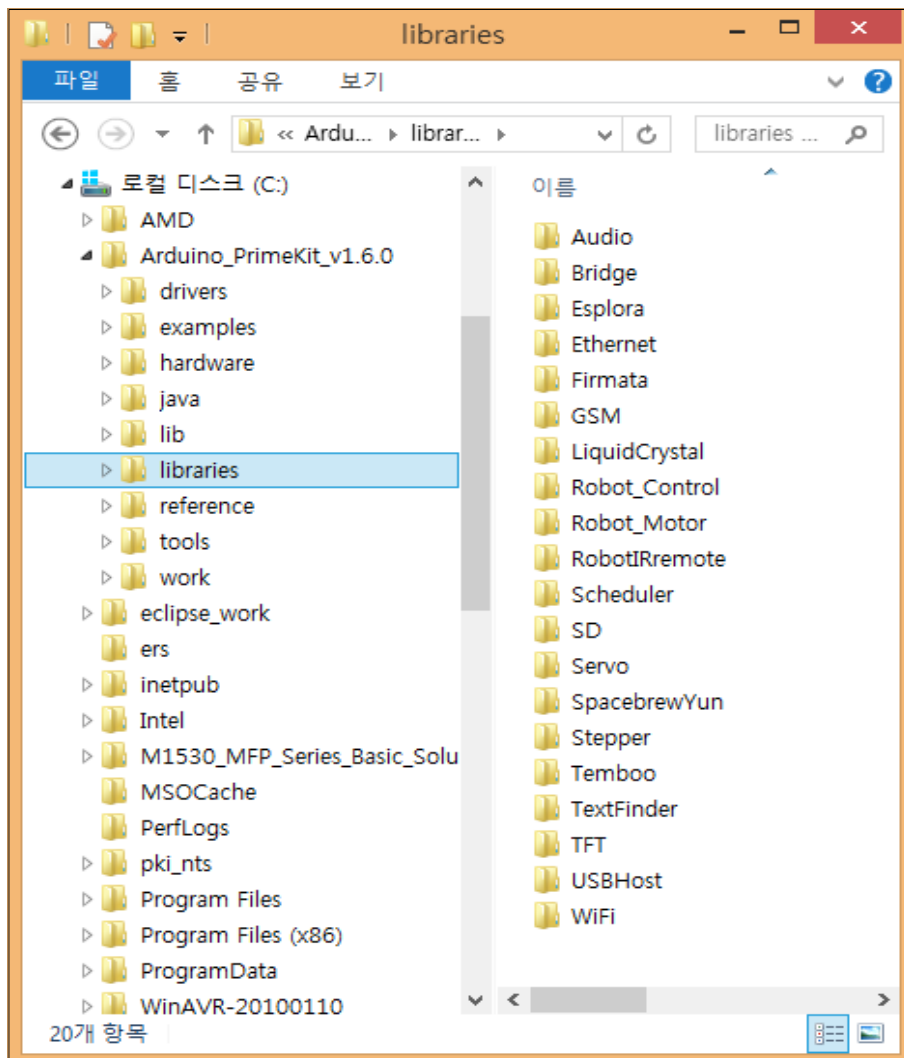
	확인	코드 오류 확인 및 소스 파일 컴파일
	업로드	컴파일 하지 않은 경우 컴파일도 수행 아두이노 보드에 소스코드 올리기
	새 파일	새로운 스케치 창 띄우기
	열기	예제 및 작성한 스케치 소스 리스트
	저장	소스 코드 저장
	시리얼 모니터	시리얼 통신 창 띄우기

▶ 아두이노 제공 라이브러리

스케치 IDE 구조를 살펴 볼 때 **libraries**폴더를 보았다.

위치 : ArduinoWlibraries

이 폴더에는 개발자가 스케치 프로그램 시에 사용하는 아두이노 표준 라이브러리와 특정 아두이노 보드에 특화된 라이브러리가 포함되어 있다.



[표준 라이브러리]	
EEPROM	EEPROM 제어 라이브러리
Ethernet	Ethernet Shield를 사용한 인터넷 연결 관련 라이브러리
Firmata	아두이노와 PC 사이의 통신 프로토콜 관련 라이브러리
GSM	GSM shield를 사용한 GSM/GRPS 네트워크 통신 라이브러리
LiquidCrystal	LCD 제어 라이브러리
SD	SD카드 제어 라이브러리
Servo	서보 모터 제어 라이브러리
SPI	SPI 통신 제어 라이브러리
SoftwareSerial	소프트웨어 시리얼 포트 제어 라이브러리
Stepper	스텝 모터 제어 라이브러리
TFT	TFT 스크린 제어 라이브러리
WiFi	WiFi Shield를 사용한 인터넷 연결 관련 라이브러리
Wire	TWI(I2C)통신 제어 라이브러리

[Arduino Due 보드에만 사용되는 라이브러리]	
Audio	SD카드로부터 오디오 파일 재생 관련 라이브러리
Scheduler	스케줄링 관련 라이브러리
USBHost	마이크, 키보드 같은 USB 주변 장치 통신 관련 라이브러리

[Arduino Esplora 보드에서만 사용되는 라이브러리]	
Esplora	Esplora기반 센서 제어 라이브러리

[Arduino Robot 보드에서만 사용되는 라이브러리]	
Robot	로봇 기능 제어 라이브러리

[Arduino Yun 보드에서만 사용되는 라이브러리]	
Bridge	Linux 프로세서와 Yun 사이의 통신 제어 라이브러리

[Arduino Leonardo, Micro, Due, Esplora 보드에서만 사용되는 라이브러리]	
Keyboard	USB 라이브러리로 PC와 연결된 키보드 제어 라이브러리
Mouse	USB 라이브러리로 PC와 연결된 마우스 제어 라이브러리

▶	외부 라이브러리 추가
---	--------------------

아두이노는 Open 소스로 다양한 센서 및 모듈 판매 회사에서 제공해 주는 라이브러리와 개발자들이 개발한 라이브러리를 추가해서 사용할 수 있다.

라이브러리를 추가하는 것은 두 가지 방법이 있다.

[방법 1] 아두이노 제공 라이브러리 폴더에 추가

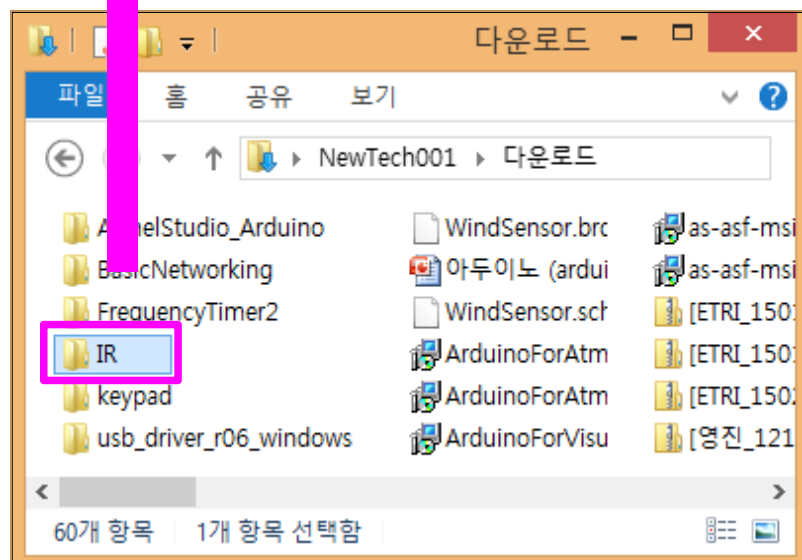
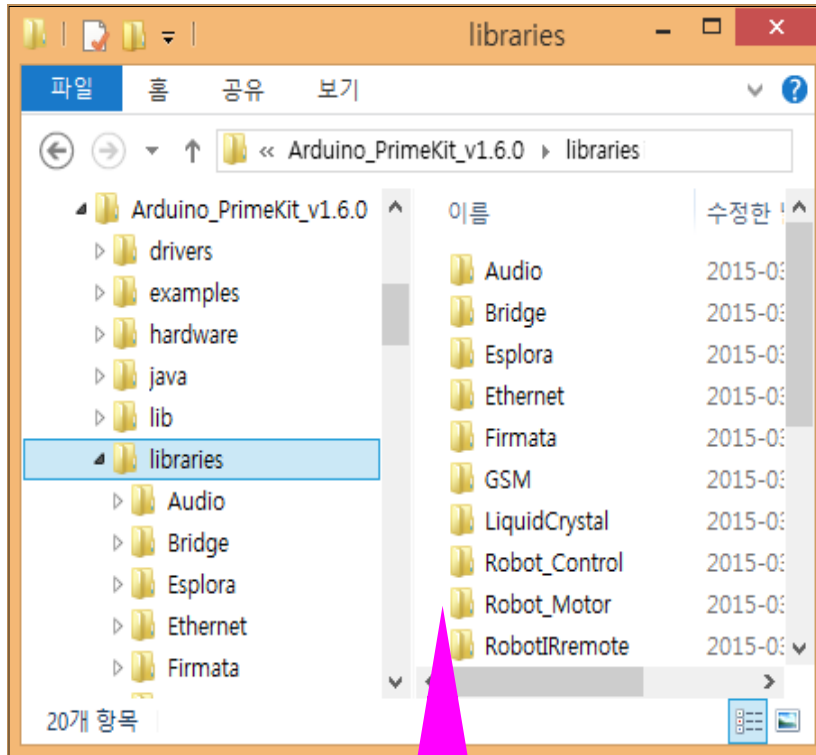
▶ 위치 : `Arduino_PrimeKit_v1.6.0\libraries`

[방법 2] 외부 라이브러리만 담는 폴더에 추가

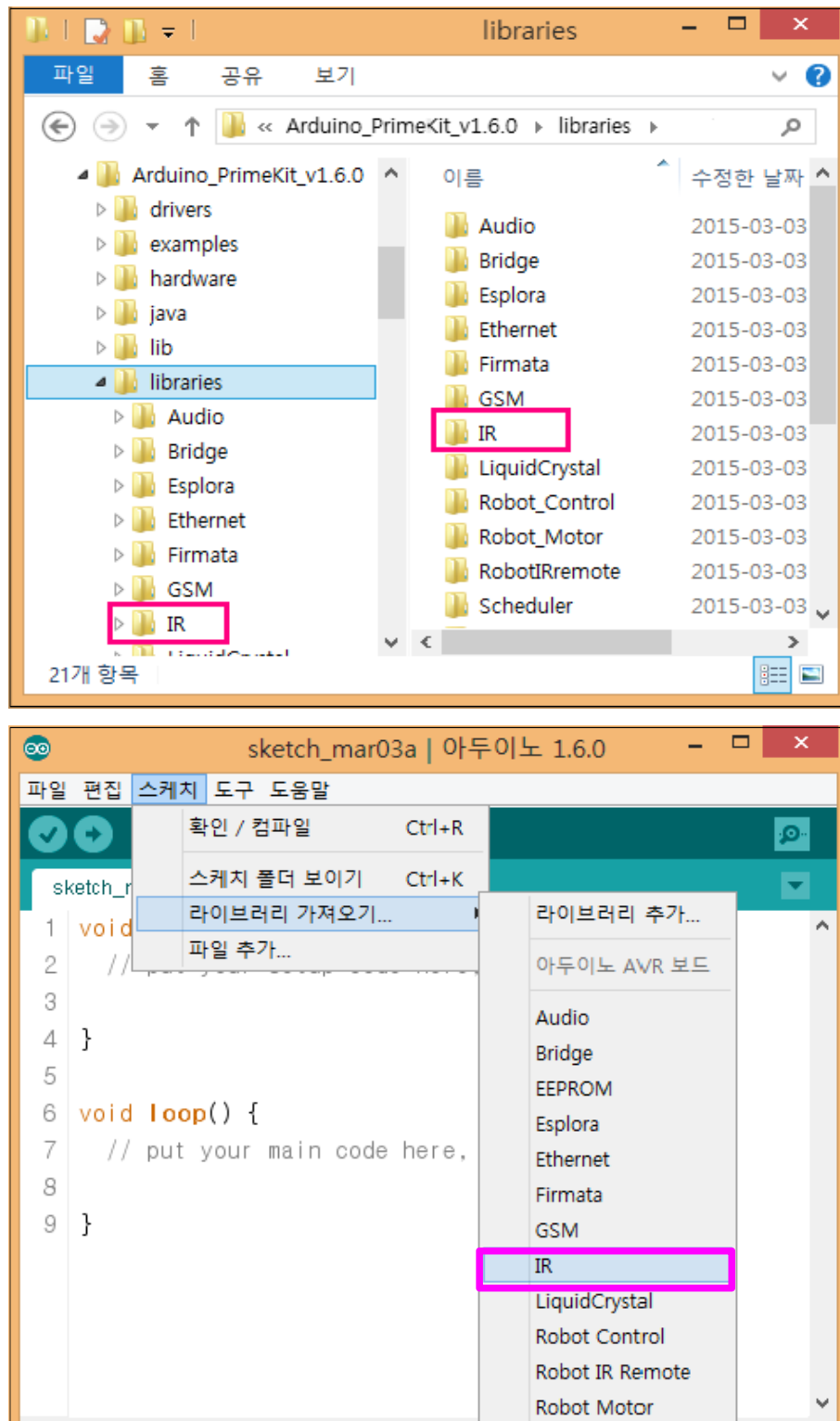
▶ 위치 : 스케치북 저장 폴더\libraries

[방법 1] 제공 라이브러리 폴더에 추가

(1-1) 스케치 IDE의 폴더 리스트에서 libraries폴더로 추가할 라이브러리를 복사한다.

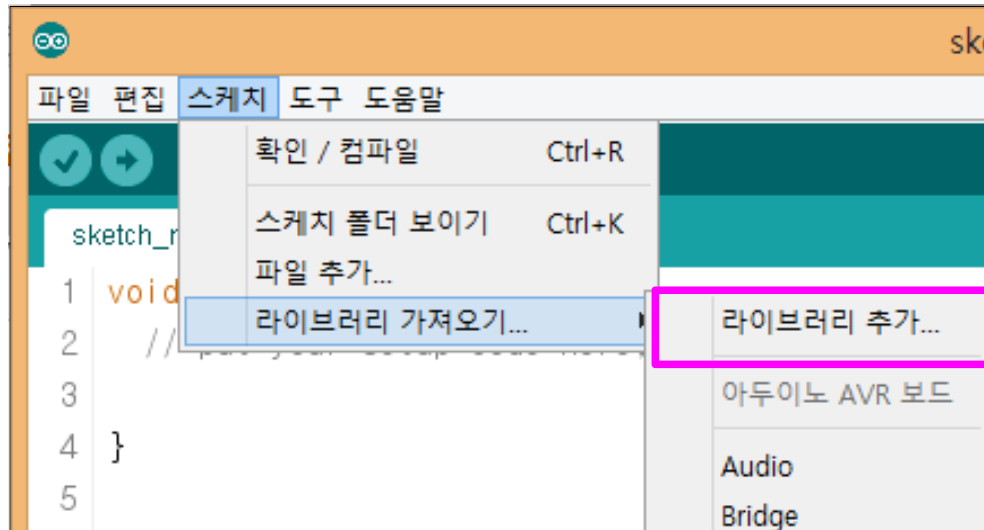


(1-2) 스케치 IDE에 추가된 라이브러리를 확인할 수 있다.

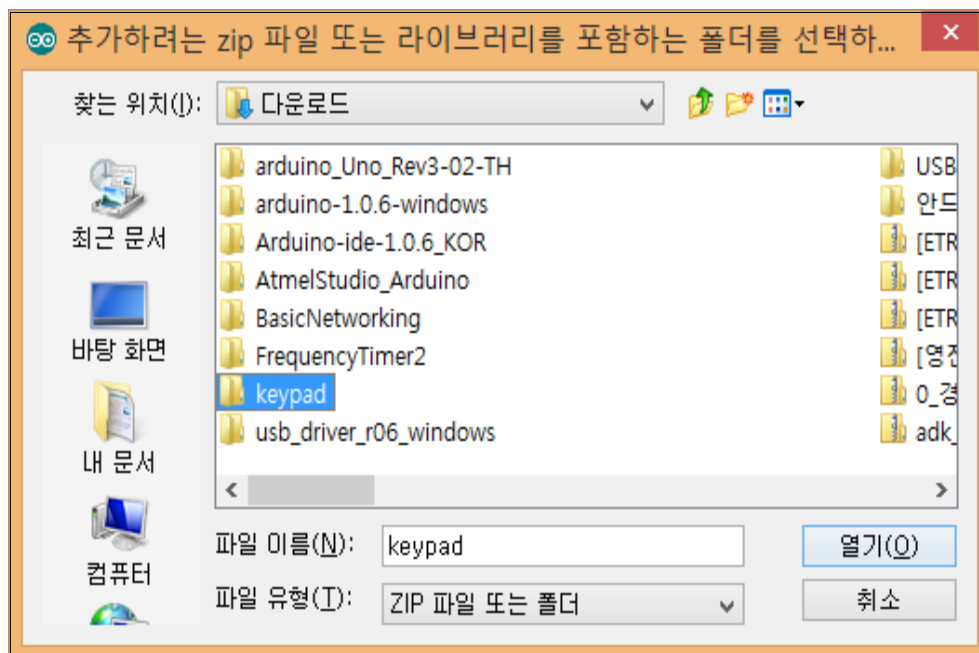


[방법 2] 외부 라이브러리만 담은 폴더에 추가

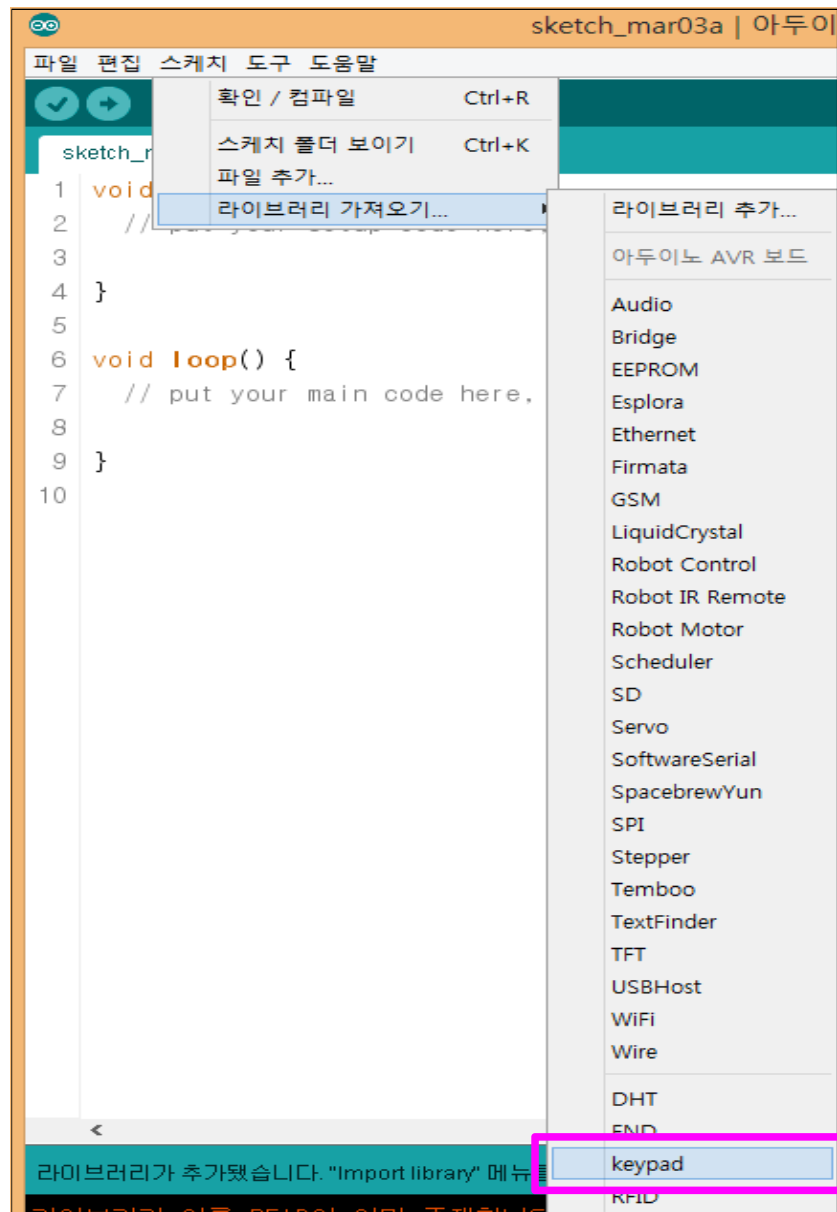
(2-1) 스케치 IDE에서 [스케치 -> 라이브러리 가져오기 -> 라이브러리 추가] 항목 선택

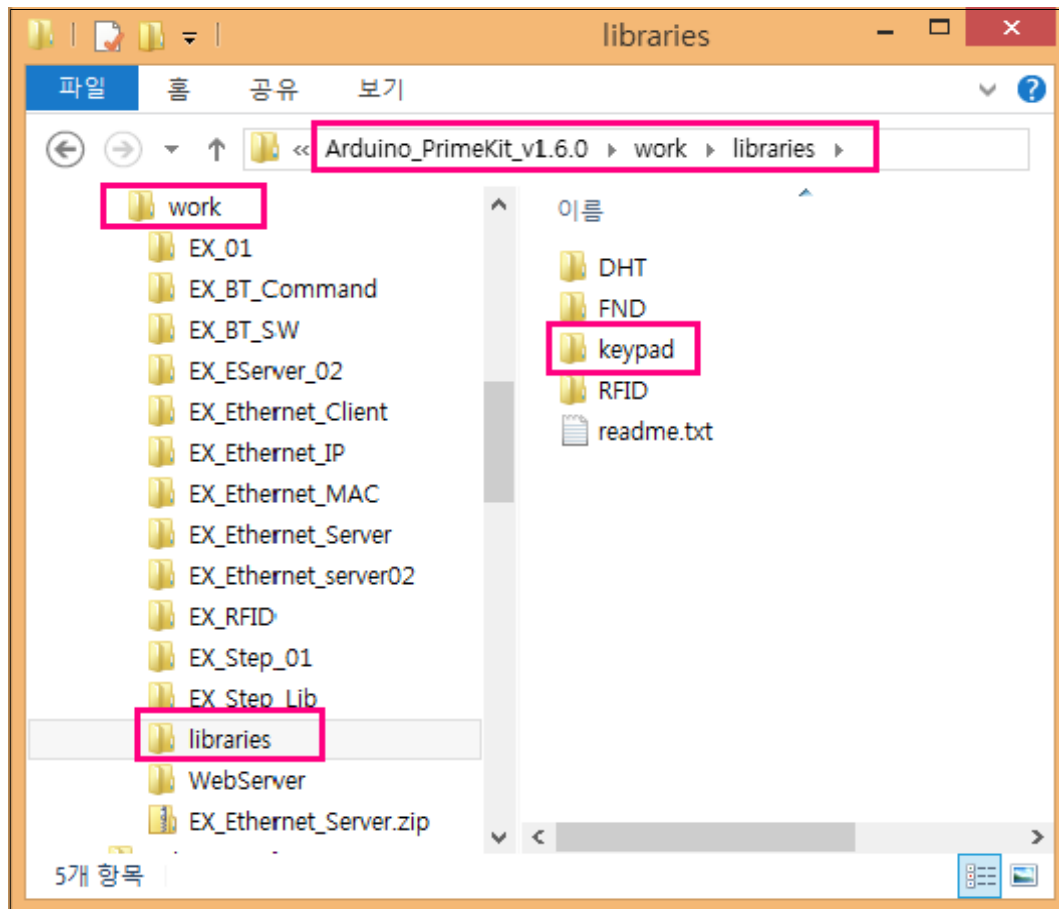


(2-2) 추가하고 싶은 라이브러리를 선택한다.



(2-3) 추가된 것을 확인할 수 있다.





[스케치 북 저장 폴더\libraries의 라이브러리]

1. 아두이노 스케치의 기본 구조와 문법

1-1. 프로그래밍 기본을 알자!!!

▶ 하드웨어 & 소프트웨어 & 프로그램

하드웨어(Hardware)는 사전적인 의미로 철물, 기계를 의미한다.

생활에서 사용하고 있는 전화, 냉장고, TV, 리모컨 등등 모든 전자기기들이 바로 하드웨어이다. 하지만, 이것은 형태만 있을 뿐 움직일 수 없다.

우리가 원하는 기능 즉, 전화를 걸고, 받고, 음식물을 보관하고, TV를 켜고 끄고 등등의 기능을 하도록 하기 위해서 필요한 것이 있다. 바로 소프트웨어(Software)다.

소프트웨어(Software)는 철물 덩어리가 어떤 기능을 할 수 있도록 순서와 절차에 맞게 지시를 내리는 명령들을 모아 놓은 것이다. 그리고 이것을 **프로그램(Program)**이라고 한다.

▶ 명령문의 끝 ; (세미콜론)

우리가 한글을 배울 때 한 문장이 끝나면 .(마침표)를 찍어준다.

프로그램에서도 명령이 끝났다는 것을 알려주기 위해서 ; **(세미콜론)**을 찍어준다.

세미콜론이 나오지 않으면 아직 명령어 문장이 끝나지 않은 것으로 인식한다.

```
int a = 10; <- 명령문 종료
```

```
int num = 23 <- 명령문 종료 안됨
```

```
;<- 여기서 종료 됨
```


▶ 명령문 집합 블록 { }

여러 개의 명령문이 모여서 하나의 기능을 수행할 때 명령문을 묶어주는 것을 블록 { } 이라 한다.

블록 { 명령문; } 안은 또 하나의 영역으로 변수를 선언할 수도 있고, 이런 변수를 지역 변수라고 한다.

```
void add(int a, int b, int cnt)
{
    int r = a + b;

    Serial.print(" Result = ");
    Serial.println(r);
}
```

3개의 명령문 블록

▶ 자료형(Data Type)

우리가 택배를 보낼 때 물건의 크기에 맞는 상자에 물건을 담아서 보낸다. 큰 상자에 작은 물건을 담아서 보내면 적당한 상자보다 상자 구매 비용이 더 발생할 것이고, 반대로 작은 상자에 물건을 담을 경우, 들어가지 않아서 물건을 잘라내고 담아야 할 것이다.

프로그램에서도 데이터를 저장하기 위해서 메모리(Memory)라는 공간이 있다. 이 공간이 무한대로 많은 것이 아니기 때문에 저장할 데이터에 맞는 공간만큼만 사용을 해야 한다. 그래서 데이터 별로 사용 가능한 최적의 크기를 지정해 두었는데 이것이 바로 **자료형(Data Type)**이라 한다.

타입 (TYPE)		특징	사용 예
논리형	boolean	- 크 기 : 1바이트 - 데이터 : true, false - 범 위 : 0, 1	boolean isOn=false;
문자형	char	- 크 기 : 1바이트 - 데이터 : 문자 한 개 - 범 위 : -128 ~ 127	char c='a';
	unsigned char	- 크 기 : 1바이트 - 데이터 : 문자 한 개 - 범 위 : 0 ~ 255	char c='a';
정수 수치형	byte	- 크 기 : 1바이트 - 데이터 : 정수 - 범 위 : 0 ~ 255	byte b=B1001;
	int	- 크 기 : 2바이트 - 데이터 : 정수 - 범 위 : -32,768 ~ 32,767	int pin=13;
	unsigned int word	- 크 기 : 2바이트 - 데이터 : 정수 - 범 위 : 0 ~ 4,294,967,295	unsigned int pin=13;
	long	- 크 기 : 4바이트 - 데이터 : 정수 - 범 위 : $-2^{32-1} \sim 2^{32-1}-1$	long value=186000L;
	unsinged long	- 크 기 : 4바이트 - 데이터 : 정수 - 범 위 : $0 \sim 2^{32}-1$	unsigned long time;
실수 수치형	float	- 크 기 : 4바이트 - 데이터 : 실수 - 범 위 : $-2^{32-1} \sim 2^{32-1}-1$ - 정밀도 : 소수점 이하 7	float f = 0.123;
	double	- 크 기 : 4바이트 - 데이터 : 실수 - 범 위 : $-2^{32-1} \sim 2^{32-1}-1$ - 정밀도 : 소수점 이하 14	double d = 0.121234;

▶ 상수(Constance)

우리가 사용하는 지역번호 053은 대구, 02는 서울이라고 정해 둔 것처럼 프로그램에 필요한 숫자 값들을 우리가 이해하기 쉬운 단어로 선언해 둔 것을 상수(Constance)라 한다. 그리고 상수는 프로그램 중간에 값을 변경할 수 없다.

상수는 프로그램 개발 시에 만들 수도 있고, 아두이노에서는 스케치 프로그램시에 사용될 상수를 미리 만들어서 선언해 두었다.

스케치 프로그램 폴더 내 "Arudino_Prime_v1.0.0\hardware\arduino\cores\arduino\WArduino.h"파일에 선언되어 있다.

```
//- 핀으로 보낼 데이터 값
#define HIGH                1    //- On
#define LOW                 0    //- Off

//- 핀의 입출력용도
#define INPUT               0    //- 입력 모드
#define OUTPUT              1    //- 출력 모드
```

```
void setup( ) {
    pinMode(13, OUTPUT);    //- 상수 사용
}
```

개발자가 프로그램 시에 새롭게 만들 수도 있다.

```
[ 방법 1 ]  const int PIN_NO = 13;           //- 상수 선언
[ 방법 2 ]  #define PIN_NUM 13             //- 매크로 상수 선언

void setup( ) {
    pinMode(PIN_NUM, OUTPUT);               //- 매크로 상수 사용
    pinMode(PIN_NO, OUTPUT);               //- 상수 사용
}
```

▶ 변수(Variable)

변수는 프로그램 중에 사용되는 데이터를 저장하는 공간이다.

우리가 메모를 하거나 연습장에 계산식을 적는 것처럼 프로그램에서 일을 하기 위해서 사용되는 데이터를 담아두는 연습장과 같은 것을 변수(Variable)라 한다.

이것은 상수와 반대로 언제든지 변경이 가능하다.

아두이노에서는 데이터를 메모리라는 공간에 저장한다. 그리고 메모리 공간을 효율적으로 사용하기 위해서 저장할 데이터 종류별로 사용가능한 메모리 크기를 정해서 자료형(Data Type)으로 만들어 두었다.

사용할 공간의 크기를 정하고 그 공간에 데이터를 저장하거나 꺼내 오기 위해서 이름을 붙여주어야 한다. 이것을 변수 선언이라 한다.

[변수 선언 및 초기화 형식]

```
데이터 크기   저장 공간 이름 ;           //- 변수 선언
데이터 크기   저장 공간 이름 = 데이터 ;   //- 변수 선언 + 초기화
```

```
int ledPin = 13;           //- 변수 선언 + 초기화
int count;                 //- 변수 선언

void setup( ) {
  pinMode( ledPin, OUTPUT );   //- 변수 사용
  count = 100;                //- 변수 초기화
}
```

▶ 배열(Array)

만약 한 사람에게 동일한 물건을 6개 보낸다고 생각해보자.

6개의 상자에 물건을 하나씩 담고, 6번 주소를 적어서 보내야 한다.

차라리 1개의 큰 상자에 6개 물건을 담고 주소를 한번 적어서 보낸다면 좀 더 수월할 것이다.

프로그램에서도 동일한 자료형의 데이터를 여러 개 저장해야 할 경우 배열(Array)이라는 것을 사용한다. 배열은 연속된 메모리 공간을 제공하고, 이 공간을 하나의 이름으로 사용할 수 있도록 했다.

대신에 여러 개의 데이터가 있으므로 식별하기 위해서 번호를 부여하고 번호로 데이터가 담긴 공간에 접근할 수 있다. 이 번호를 **첨자 또는 인덱스(Index)**라 한다.

예를 들어서 아파트 101동에는 동일한 구조의 집들이 10채가 있다. 이 10채의 집을 구별하기 위해서 101호, 201호, 301호,, 1001호 방식으로 번호가 부여되어 있다. 여기서 101동이 배열이름이 되고, 101 ~ 1001이 바로 첨자가 된다.

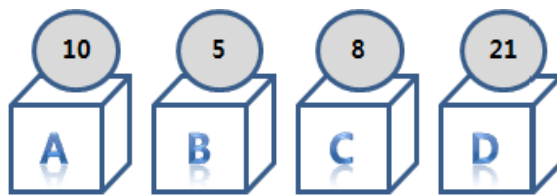
첨자는 배열의 크기를 의미하며 범위는 0 ~ (배열 크기-1)이다.

[배열 선언 및 사용]

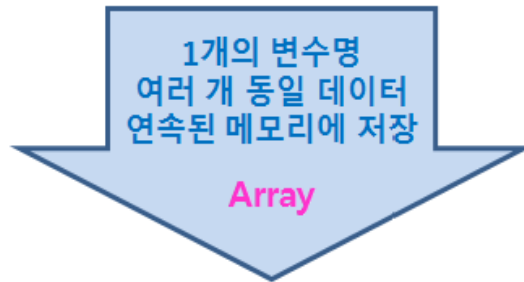
자료형 배열이름 [첨자]

```
int    values [ 4 ];           //- int형 4개의 데이터가 저장되는 공간
values [0] = 5;                //- 0번째 공간에 5 저장
values [1] = 10;               //- 1번째 공간에 10 저장
values [2] = 0;                //- 2번째 공간에 0 저장
values [3] = 4;                //- 3번째 공간에 4 저장
```

 [첨자 (인덱스)]



```
int A = 10;
int B = 5;
int C = 8;
int D = 21;
```



배열 이름 : A
배열 크기 : 4

* 선언 방법(1)

```
int A[4] = {10, 5, 8, 21}; // - 선언+초기화
```

* 선언 방법(2)

```
int A[4];           //- 선언
A[0] = 10;          //- 초기화
A[1] = 5;
A[2] = 8;
A[3] = 21;
```

▶ 문자열(string)

하나의 문자가 아닌 문장이나 단어 등 여러 개의 문자가 모여 있는 것을 문자열(string)이라 한다.

문자열은 쌍 따옴표("")로 감싸서 표기를 한다. 그리고 한 개 이상의 문자가 모여 있기 때문에 char 타입의 배열에 저장해야 한다.

주의할 것이 하나있다.

문자열은 기본적으로 끝에 **Null 종료 문자**라는 것이 붙는다. 왜냐

하면, 데이터 전달 시 문자열의 끝을 알기 위해서 사용한다.

그래서 문자열을 배열에 저장할 경우에는 '문자개수 + 널 종료 문자'로 배열 크기를 잡아 주어야 한다.

Null 종료 문자는 ASCII code 값은 0이고, 특수 문자 '\0'로 약속되어 있다.

```
char messages[10] = "Good";
char messages[ ] = "Happy"; c
har messages[8] = "arduino";          <= 7문자 + null 종료 문자
char str1[8] = { 'a', 'r', 'd', 'u', 'i', 'n', 'o' };          <= 문자 7개 저장
char str1[8] = { 'a', 'r', 'd', 'u', 'i', 'n', 'o', '\0' };      <= 문자열 저장
```


▶ String 객체

문자열은 char 타입 배열을 선언해서 저장을 한다.

많이 사용되는 문자열을 매번 char 타입 배열을 선언하지 않고 저장 및 사용하기 쉽도록 제공되는 것이 있다. 바로 String 객체이다.

객체(Object)란 객체지향언어의 기본 개념이다.

쉽게 생각을 한다면 사람, 강아지, 나무, 산 등등 세상에 존재하는 모든 것들이 객체가 된다. 우리는 이런 객체들이 가지는 모양, 성질, 기능들을 설명을 들으면 무슨 객체를 의미하는지 알 수가 있다.

예를 들어서 다리가 4개고 꼬리가 있고 아는 사람을 보면 꼬리를 흔들고 낯선 사람을 보면 짖는 것은 무엇일까? 바로 강아지다.

이렇게 객체가 가지는 모양, 성질, 기능, 역할 등을 가지고 세상에 존재하는 모든 사물들을 분류하고 인지할 수 있다. 이 개념을 프로그램에 적용한 것이 바로 객체지향언어이다.

String 객체는 문자열을 프로그램에서 쉽고 편하게 사용하기 위해서 문자열이 가지는 속성과 기능들을 하나로 묶어서 만든 자료형이다.

그래서 일반 변수를 사용할 때처럼 사용하면 된다.

[String 객체 선언 및 사용]	
객체명 변수명; 변수명	// - String객체 타입 변수 선언
수명 . 함수명();	// - String객체 제공 함수 사용
String msg = "Happy";	
String msg2 = "10";	
msg.substring(1);	// - "Happy"에서 인덱스가 1인 'a' 리턴
msg2.toInt();	// - "10"을 정수 10으로 변환

▶ 함수(Function)

프린터는 프린트를 한다. 계산기는 계산을 한다.

프로그램에서 특정 기능을 수행하는 코드를 모아 놓은 것을 **함수 Function**이라한다.

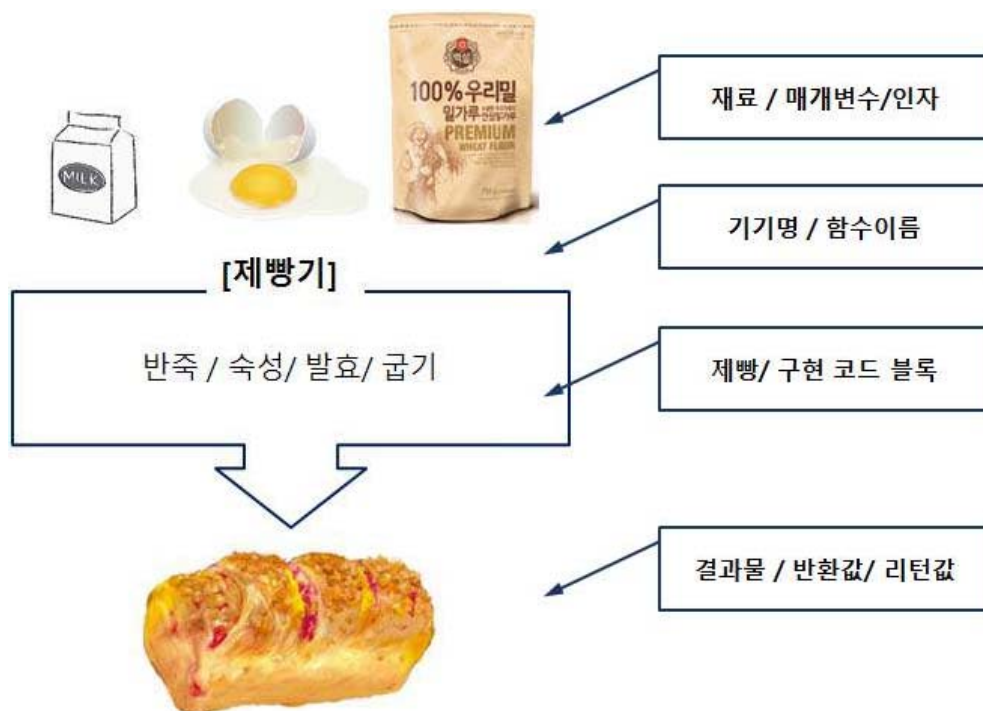
프린트를 할 때 마다 프린터기를 만들지 않는 것처럼 함수를 잘 만들어 두면 특정 기능 수행할 경우 만들어진 함수를 사용만 하면 된다.

함수를 사용함으로써 중복되는 코드가 줄어들고, 이미 만들어진 코드를 재사용할 수 있어서 좋다.

그럼 함수는 어떻게 만들까? 간단하다. 예

를 들어서 빵을 만드는 제빵기가 있다.

제빵기가 빵을 만들기 위해서는 밀가루, 달걀, 우유 등의 재료가 필요하고 이 재료를 제빵기 안에 넣어주면 제빵기가 빵을 만들어서 준다.



함수도 제빵기와 같다. 위의 그림과 같다.

[함수 선언]		
반환타입 함수이름(매개변수타입 매개변수이름) { // 기능 구현 코드 return 결과; // 결과 리턴(반환) } 		
함수기능	정수 2개를 더하기	<pre>int add(int a, int b) { int result = a + b; return result; }</pre>
함수이름	add	
매개변수	int a, int b	
반환타입	int	

함수가 기능을 수행 후에 결과물이 있을 수도 있고 없을 수도 있다.

만약 결과물이 없을 경우 사용하는 키워드가 있다. 바로 void이다.

함수를 선언할 경우 반환 결과가 없다면 void라고 반환타입에 작성하면 된다.

함수기능	메시지 출력	<pre>void displayMessage(String msg) { Serial.println(msg); }</pre>
함수이름	displayMessage	
매개변수	String msg	
반환타입	void	

만들어진 함수를 사용하는 것을 **함수 호출**이라 한다.

프로그램에서 필요한 경우 함수를 사용하면 되는데, 실생활에서도 누군가에게 일을 시키기 위해서 '누구야~ 청소 좀 해줘~.'라고 불러서 일을 시키는 것과 같다.

바로 함수이름을 불러 주면 된다.

[함수 호출]	
함수이름(매개변수이름);	
<pre>void displayMessage(String msg) { Serial.println(msg); }</pre>	<pre>int total =0; String msg="Total : "; for(int num=0; num<10; l++) { total += add(num, num*10); } msg += total; displayMessage(msg);</pre>
<pre>int add(int a, int b) { int result = a + b; return result; }</pre>	

▶ 전역 변수와 지역 변수

변수는 선언되는 위치에 따라서 변수의 사용 범위가 달라지는데 이것을 스코프(Scope)라하며 전역 변수, 지역 변수라 부른다.

	전역 변수(Global Variable)	지역 변수(Local Variable)
위치	함수 코드 블록 밖에 선언	함수 코드 블록 안에 사용
범위	같은 파일 안에 있는 모든 함수에서 사용 가능	다른 함수 사용 불가. 해당 함수에서만 사용 가능

```
int globalVar=10;

void setup( ) {
    globalVar = 11;
    localVar = 21;    // loop()함수 안에 선언된 변수, 다른 함수 사용불가
    ERROR 발생
}

void loop( ) {
    int localVar=20;

    localVar=21;
    globalVar=12;    //- 함수 밖에 선언되어 있는 전역변수으로써 사용 가능
}
```

▶ 연산자(Operator)

연산자(Operator)란? 프로그램에서 기능을 수행하기 위해서 계산에 사용되는 다양한 기호들이며 연산의 대상이 되는 것을 피연산자(Operation)이라 한다.

(예) $10 + 20$ 연산자 : $+$ 피연산자 : $10, 20$

프로그램에서 사용되는 다양한 연산자들을 살펴보자.

연산자	기호	역할	
대입	=	오른쪽에 있는 것을 왼쪽에 넣기	
		(예) int a = 10;	
산술	+, -, *	덧셈, 뺄셈, 곱셈, 나눗셈, 나머지 연산 수행	
	/, %	(예) 1+3, 10-8, 2*9, 10/3, 7%2	
관계	==, !=	피연산자 값의 관계 검사 후 true, false 반환	
	<, > <=, >=	(예) int a = 10; int b = 30; a == b -> false a != b -> true	
논리	&&,	관계 연산자와 같이 사용되어 두 개 이상의 조건식을 검사 후 true, false 반환	
		(조건A) && (조건B)	조건A, B 모두 참일 때 true
		(조건A) (조건B)	조건A, B 중 하나만 참이면 true
		(예) int a=10; int b=30; (a > b) && (a != b) -> false (a > b) (a != b) -> true	

연산자	기호	역할
증감	++, --	변수의 값을 1 증가 또는 1 감소시키는 연산자
		(예) int a = 10; a++; // a는 11 a--; // a는 10
복합	+=, -=	대입 연산자와 다른 연산자를 결합해 식을 간단히 하는 연산자
	*=, /= %=	(예) a = a+10; -> a += 10; b = b/2; -> b /= 2;

▶ 제어문 - 조건문

프로그램 실행 시에 조건에 따라서 프로그램의 실행 순서를 변경하기 위한 것이다.

[조건문 if~문]

기능	프로그램 실행 중 조건에 따라서 프로그램 실행 순서를 변경하는 것	
형태	if(조건){ A } 문	(조건식)이 참인 경우만 A 실행
	if(조건){ A } else{ B } 문	(조건식)이 참인 경우 A, 거짓이면 B 실행
	if(조건)~ else if(조건)~ 문	조건이 여러 개인 경우 사용 첫 번째, 두 번째 조건 반복 검사 후 실행
예시	<pre>if(val == 0) { digitalWrite(13, LOW); }</pre>	//- 조건이 참인 경우만 동작하는 경우
	<pre>if(val == 0) { digitalWrite(13, LOW); } else { digitalWrite(13, HIGH); }</pre>	//- 참인 경우의 동작 //- 거짓인 경우의 동작 * 참/거짓에 따라 다른 동작을 하는 경우
	<pre>if(val == 0){ digitalWrite(14, LOW); } else if(val == 1) { digitalWrite(13, LOW); } else { digitalWrite(13, HIGH); digitalWrite(14, HIGH); }</pre>	//- 조건이 0일 경우 동작 //- 조건이 1일 경우 동작 //- 그 외의 경우의 동작 * 조건 하나에 다양한 동작이 있는 경우

▶ 제어문 - 선택문

다양한 경우의 수가 존재하는 경우 사용되는 제어문이다.

[선택문 switch~case~문]

기능	하나의 조건에 여러 경우가 발생할 경우 사용 조건문의 if~ else if~ else~구문과 동일 기능이지만 코드는 훨씬 간결 단! 조건은 반드시 변수만 사용 가능	
형태	<pre>switch(조건 변수){ case 경우1: break; case 경우2: break; default: break; }</pre>	(조건 변수)의 각 경우에 따른 case문 실행 해당하는 경우가 없을 경우 default 실행
예시	<pre>switch(val) // 조건 변수 val에 여러 경우의 처리 { case 0: // val의 값이 0일 경우 digitalWrite(13, LOW); digitalWrite(14, LOW); break; case 1: // val의 값이 1일 경우 digitalWrite(13, LOW); digitalWrite(14, HIGH); break; case 2: // val의 값이 2일 경우 digitalWrite(13, HIGH); digitalWrite(14, LOW); break; default: // 이 외의 경우 digitalWrite(13, HIGH); digitalWrite(14, HIGH); break; }</pre>	

▶ 제어문 - 반복문

프로그램 실행 시에 동일한 코드를 계속적으로 실행해야 할 경우 사용된다.

중복 코드를 제거할 수 있다.

[반복문 for~문]

기능	지정된 횟수만큼 동일한 코드 반복 실행	
형태	<pre>for(시작; 완료; 증감){ //- 반복 코드 }문</pre>	반복의 횟수를 아는 경우에 사용 시작 - 반복 횟수 초기화 완료 - 반복 횟수 지정 증 감 - 반복 횟수 계산
예시	<pre>for(int i=0; i<10; i++) { //- 10번 반복 digitalWrite(13, HIGH); delay(1000); digitalWrite(13, LOW); delay(1000); }</pre>	

[반복문 while~문]

기능	조건이 true인 경우까지만 반복 수행	
형태	<pre>while(조건식) { // 반복할 내용 }</pre>	반복의 횟수를 알 수 없는 경우 (조건식)이 참일 경우만 수행됨
예시	<pre>int i=0; while(i<10) { digitalWrite(13, HIGH); delay(1000); digitalWrite(13, LOW); delay(1000); i++; }</pre>	

[반복문 do~ while~문]

기능	조건 검사 없이 우선 한 번 실행 후 조건 검사	
형태	<pre>do{ // 반복할 내용 }while(조건식);</pre>	무조건 한 번은 실행이 되는 형태
예시	<pre>int i=0; do{ digitalWrite(13, HIGH); delay(1000); digitalWrite(13, LOW); delay(1000); i++; }while(i<10);</pre>	

▶ 주석문

프로그램을 작성 시 코드에 대한 설명을 적어둔 것을 **주석(Comment)**이라 한다.
시간이 지난 뒤에 기능을 파악 및 수정 보완 작업이 수월해지기 때문이다.

컴파일 및 프로그램 실행에는 영향을 미치지 않는다.

한 줄 주석	//	<pre>[예] int a = 20; // boolean bOn = false; char name[10]="Tom";</pre>
여러 줄 주석	/* ~ */	<pre>[예] /* int a = 20; boolean bOn = false; char name[10] = "Tom"; */</pre>

1-2. 아두이노 스케치(Sketch) 프로그램

스케치(Sketch) 프로그램은 C, C++언어를 사용한 프로그램 개발 도구로써 개발, 컴파일, 실행까지 모두 가능하다.

일반적인 C, C++ 프로그램과는 조금 다른 구조로 main() 함수가 보이지 않는다.

대신 필수 함수 2개가 존재하며 반드시 구현해 주어야 한다.

▶ 스케치 프로그램의 기본 구조

```
//-----
//- 헤더파일 포함 : 라이브러리 사용을 위한 헤더 파일
//-----
#include "RgbLed.h"

//-----

//- 매크로 상수 : 핀 번호, 프로그램 제어 상수
//-----
#define PIN_NUM 10

//-----

//- 전역변수 : 객체타입 변수, 프로그램 제어 변수
//-----
boolean bOn = false;
RgbLed led;

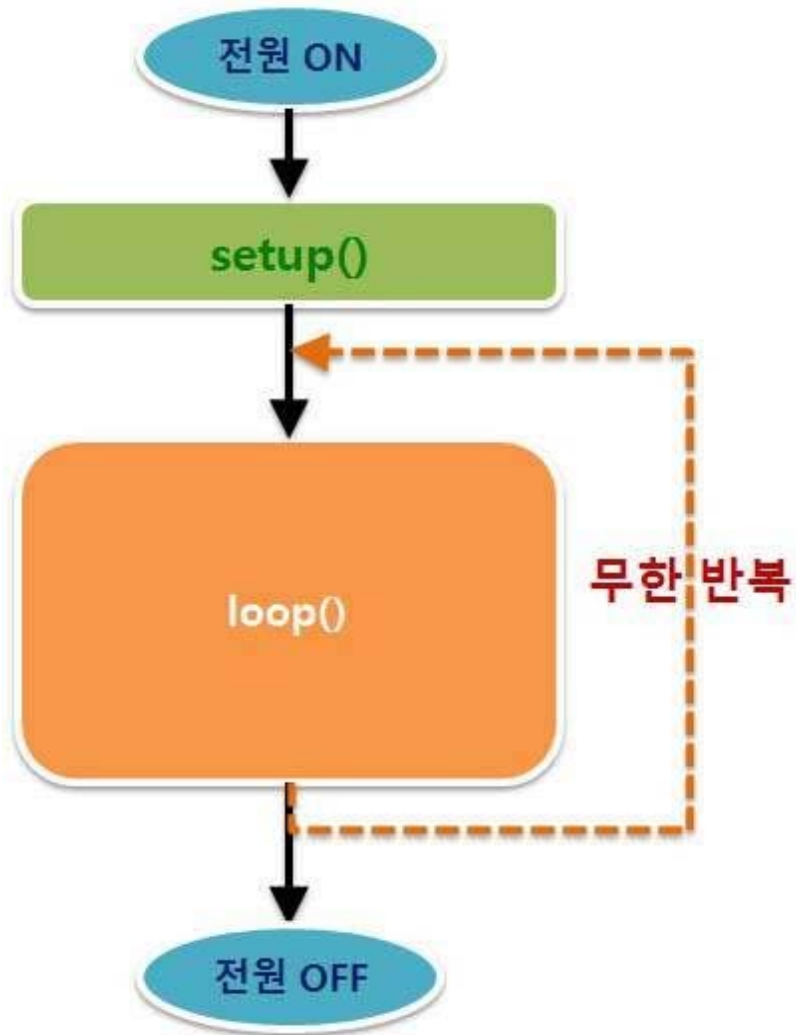
//-----

//- 초기화 함수 : 프로그램 초기화 및 설정, 전원 On된 후 1번만 실행
//-----
void setup() {
}

//-----

//- 기능 구현 함수 : 전원이 Off되기 전까지 무한 반복 수행
//-----
void loop() {
}
```

아두이노에 전원을 인가하면 내부적으로 코드에서 **setup()**이라는 함수를 찾아 함수의 내용을 실행한다. 그 이후, 다시 **loop()**라는 함수를 찾고 **loop()** 함수를 무한히 반복하게 된다.



2. 입력(Input) & 출력(Output) 이해

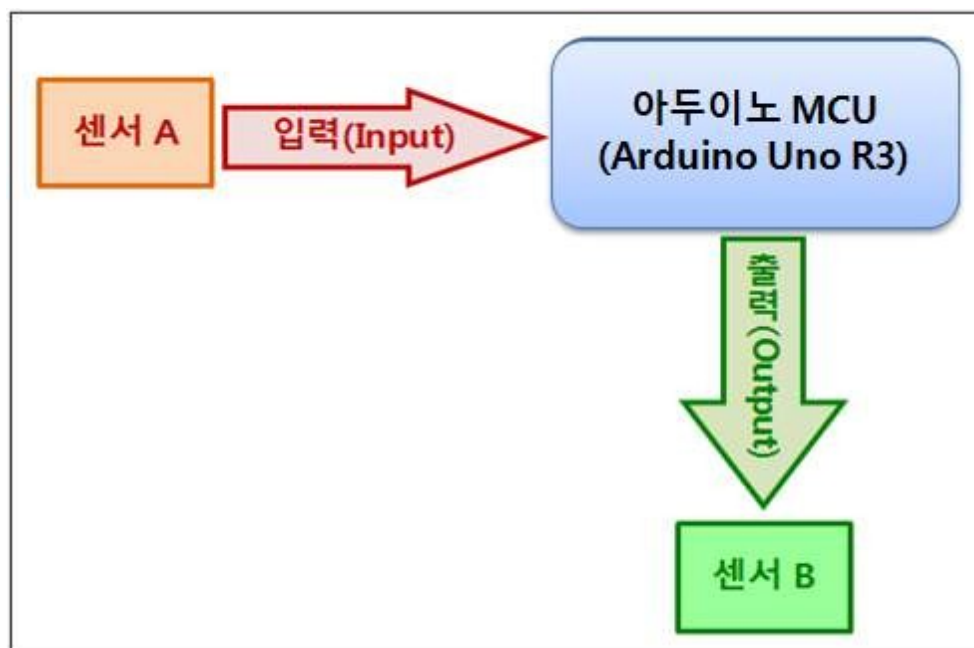
2-1. 입력 & 출력

▶ 입력(Input) & 출력(Output) 이란?

세상에는 맛있는 음식이 너무나 많이 있다.
우리는 매일 매일 무엇을 먹을까 고민을 한다. 행복하게 먹은 음식은 몸 안에서 영양분으로 흡수하고 필요 없거나 남는 것은 밖으로 내보낸다.

아두이노 또한 마찬가지 이다.
어떤 신호나 정보를 받아 들어서 어떤 일을 하기도 하고, 반대로 외부에서 요청이 와서 일을 수행하고 결과를 보내 주기도 한다.

이 때 외부에서 아두이노로 보내 주는 것을 **입력(Input)**이라하고 아두이노에서 외부에 보내주는 것을 **출력(Output)**이라 한다.



▶ 데이터(Data)

외부 센서 또는 내부 기기와 아두이노가 주고받는 것을 데이터(Data)라 한다.

예를 들어서 다음과 같은 것을 데이터라 한다.

- 외부 센서에서 측정된 값 (온도, 소음, 습도 등등)
- 외부 센서에서 아두이노로 전달하는 명령어
- 아두이노에서 외부 센서에 전달하는 명령어 또는 값
- 아두이노에서 외부 센서에 전달하는 연속된 값

데이터는 숫자, 문자, 정보, 명령어와 같은 다양한 형태이지만 기기가 인식하는 것은 전기적인 신호밖에 없다.

그럼 **전기적인 신호**란 무엇일까?

우리가 사용하는 한글을 배우기 이해하기 위해서 자음(ㄱ, ㄴ, ㄷ.....)과 모음(아, 야, 어.....)을 배우고 이것을 바탕으로 단어, 문장을 만들었다.

영어를 배우기 위해서 알파벳 26개를 배우고 알파벳을 조합해 단어, 문장을 만들었다.

전기적인 신호 또한 마찬가지다.

전기적인 신호는 전기가 켜져 있다와 꺼져 있다는 2가지만 존재한다.

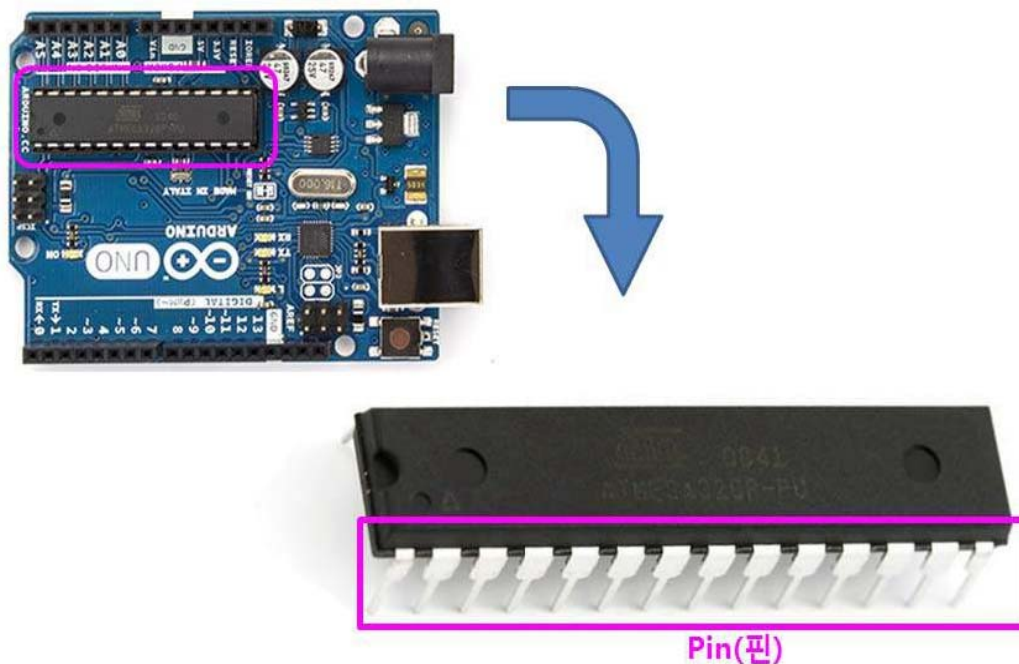
이 2가지를 기기가 인식하는 디지털 숫자 값과 전류의 세기를 나타내는 단어로 아래와 같이 나타낸다.

전원 켜 경우	전원을 끈 경우
1	0
HIGH	LOW

그래서 전자기기인 아두이노에 다양한 종류의 데이터를 입력하더라도 내부적으로는 0과 1의 디지털 숫자 값으로 번역해서 동작하게 된다.

▶ 데이터(Data) 이동 통로

외부 센서나 기기, 내부 기기와 아두이노는 어디로 데이터를 주고받을까?



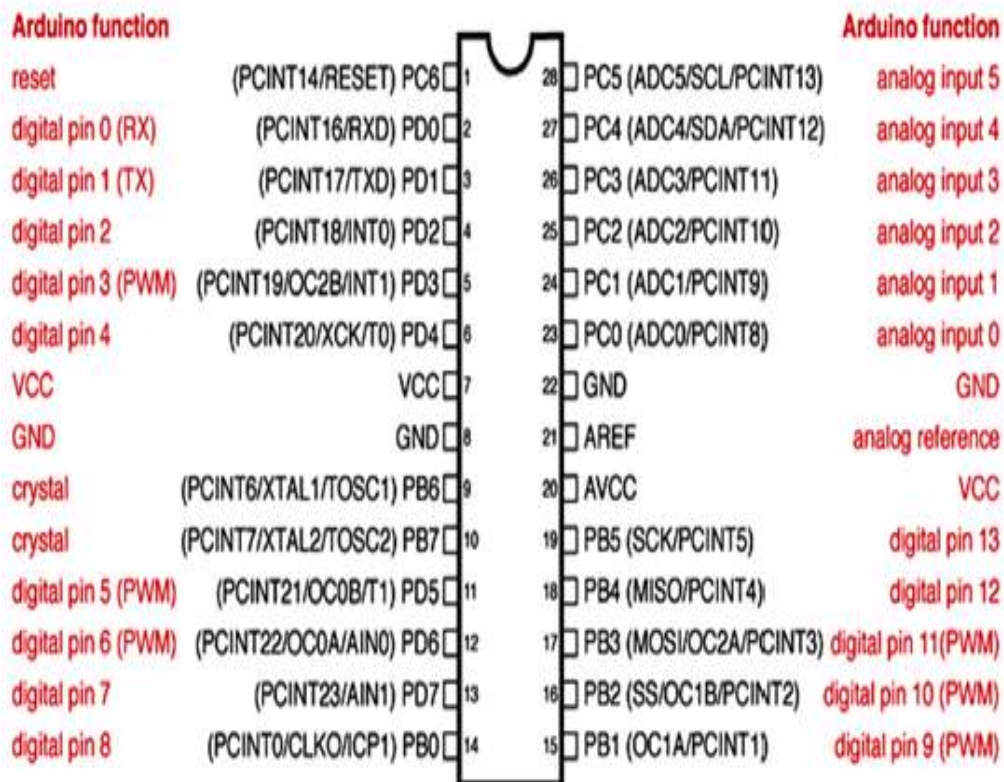
그림에서 보면 아두이노의 Arduino Uno R3의 ATmega328 MCU 주위에 많은 쇠 선들이 있다. 이 쇠 선을 **핀(Pin)**이라고 부르고 이 핀을 통해서 아두이노와 외부에 다양한 센서 및 기기들과 데이터를 주고받는다.

아두이노의 Arduino Uno R3의 ATmega328 MCU에는 총 28개의 핀이 존재한다. 각 핀은 기본적으로 어떤 기능을 할지 정해 놓았는데 이것을 **핀 맵(Pin Map)**이라 한다.

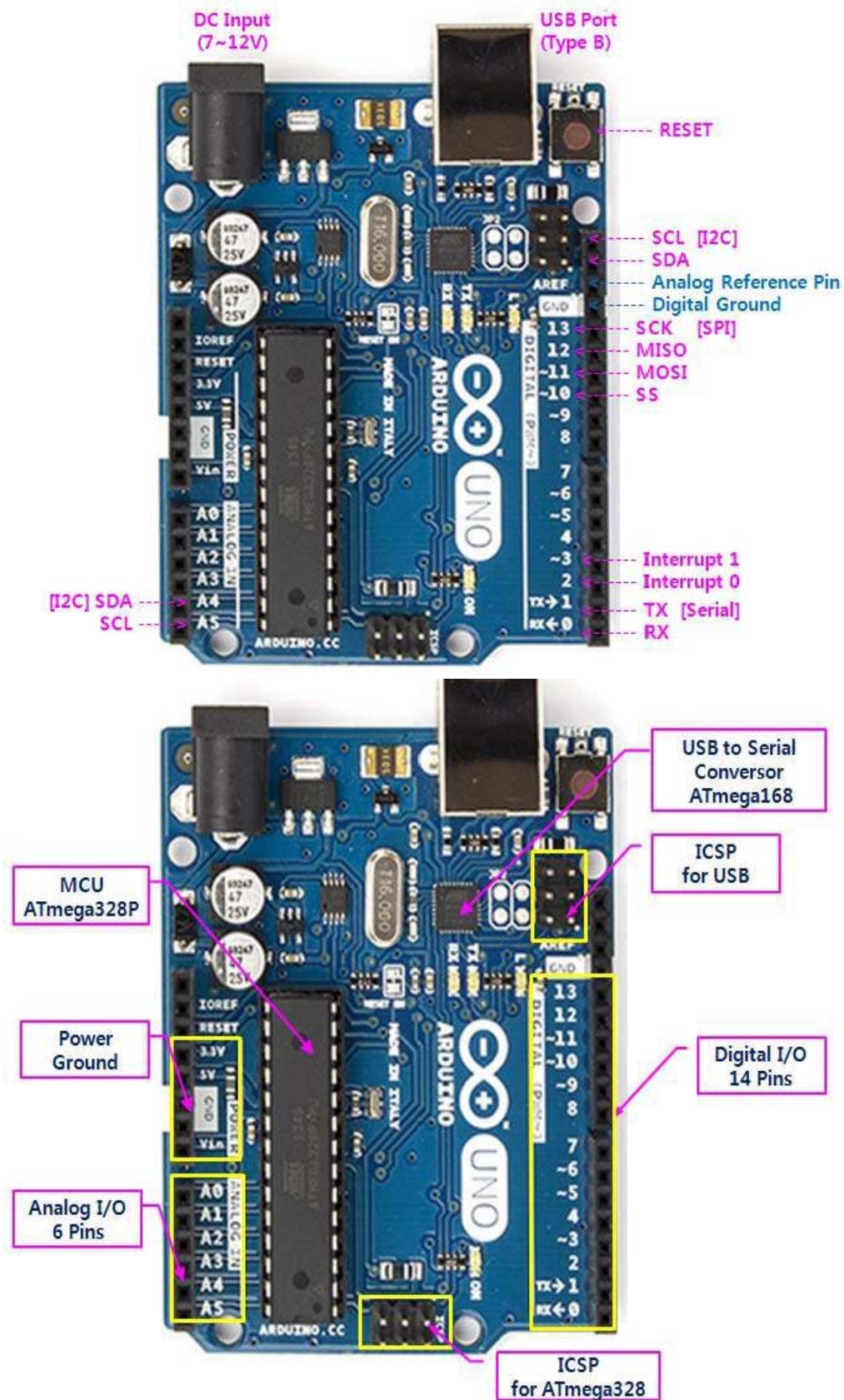
3. 아두이노 입력(Input)과 출력(Output)

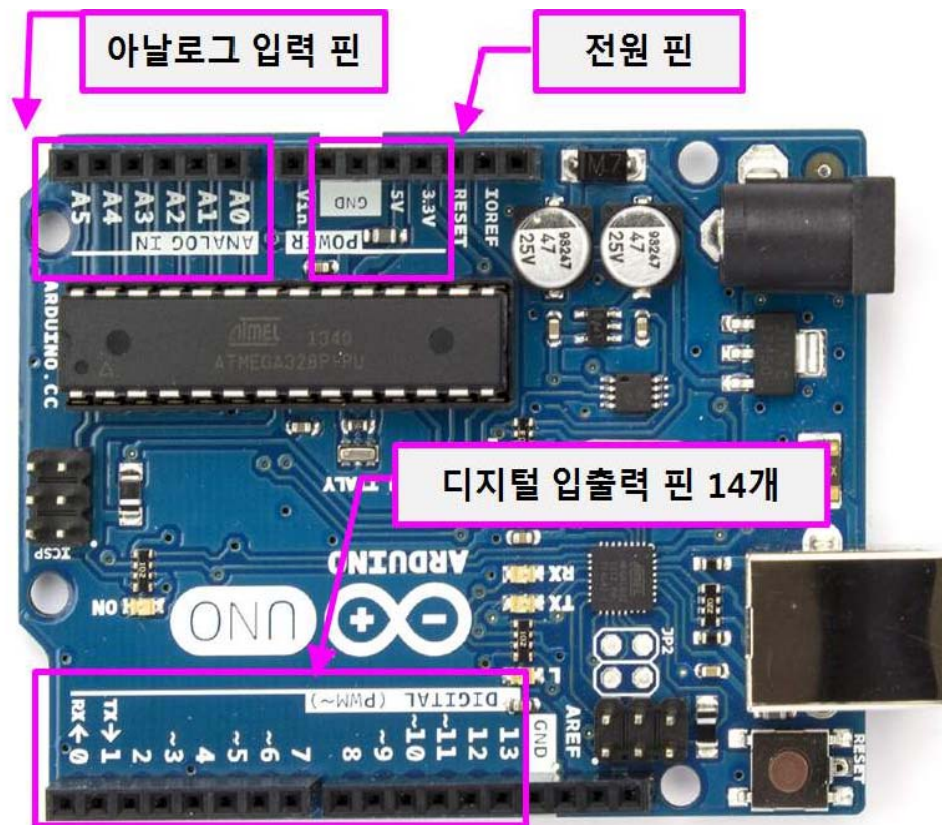
3-1. 입력 & 출력 핀 맵

전체 핀 개수	디지털 입출력 전용 핀	아날로그 입출력 전용 핀
28개	14개(PWM 6개 포함)	8개



[Arduino Uno R3 보드 요소 기능]





[Arduino Uno R3 입출력 핀 & 전원 핀]

이렇게 28개의 각 핀들은 이미 지정된 기능을 가지고 있고 각 핀을 사용하기 위해서는 기본적으로 2가지 설정을 해 주어야 한다.

1. 핀의 용도 설정 => 입력 또는 출력
2. 데이터 타입 설정 => 디지털 또는 아날로그

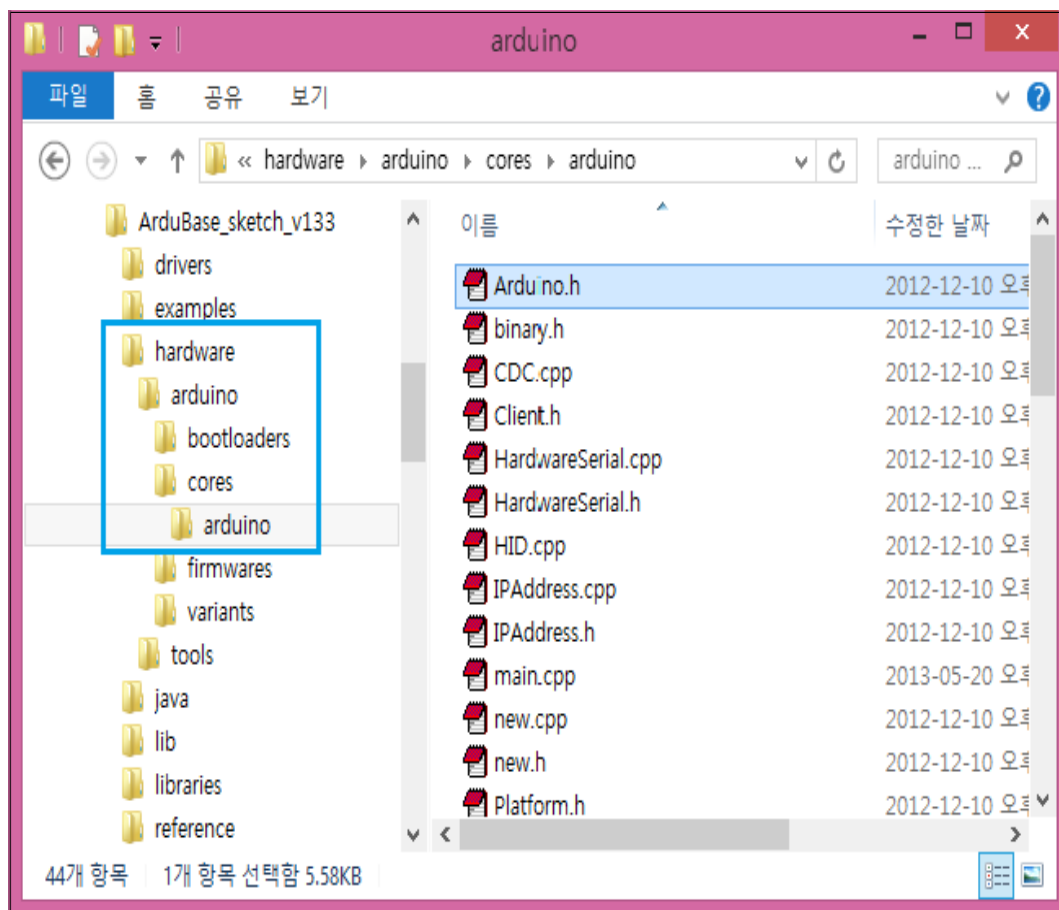
2가지 설정이 끝나면 해당 핀을 통해서 원하는 기능을 구현할 수가 있다.

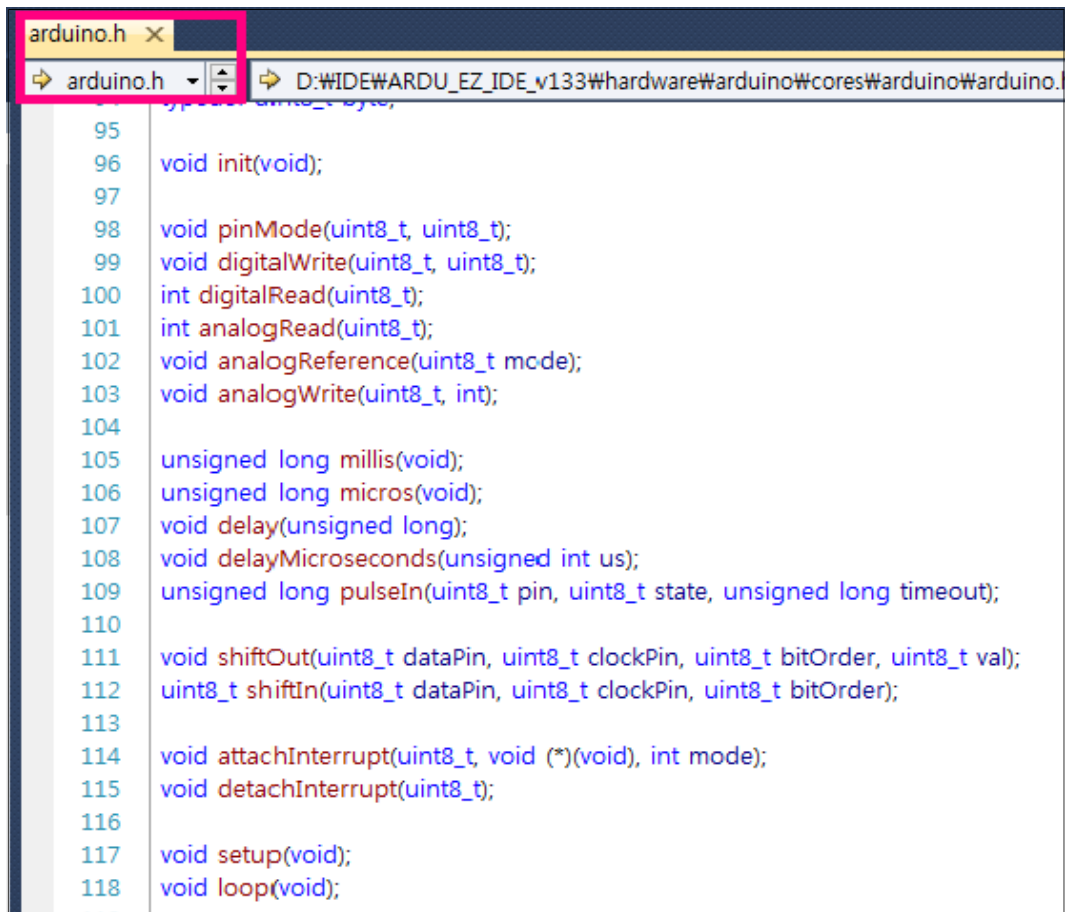
3-2. 입력 & 출력 라이브러리

▶ 라이브러리 위치

아두이노는 핀에 대한 입출력 데이터 처리를 위한 함수들을 제공하고 있다.

설치된 스케치 폴더 내에 'Arduino_Prime_v1.0.0\hardware\arduino\cores\arduino\Arduino.h' 파일에 함수들이 선언되어 있다.





```

95
96 void init(void);
97
98 void pinMode(uint8_t, uint8_t);
99 void digitalWrite(uint8_t, uint8_t);
100 int digitalRead(uint8_t);
101 int analogRead(uint8_t);
102 void analogReference(uint8_t mode);
103 void analogWrite(uint8_t, int);
104
105 unsigned long millis(void);
106 unsigned long micros(void);
107 void delay(unsigned long);
108 void delayMicroseconds(unsigned int us);
109 unsigned long pulseIn(uint8_t pin, uint8_t state, unsigned long timeout);
110
111 void shiftOut(uint8_t dataPin, uint8_t clockPin, uint8_t bitOrder, uint8_t val);
112 uint8_t shiftIn(uint8_t dataPin, uint8_t clockPin, uint8_t bitOrder);
113
114 void attachInterrupt(uint8_t, void (*)(void), int mode);
115 void detachInterrupt(uint8_t);
116
117 void setup(void);
118 void loop(void);

```

[hardware\Arduino\cores\Arduino\Arduino.h]

arduino.h 파일은 아두이노 프로그래밍의 기본 헤더 파일로써 프로그램 컴파일 시 자동 추가 되므로, #include "Arduino.h" 코드를 넣지 않아도 된다.

라이브러리 함수 - 입출력 함수		
▶	경로	hardware\arduino\cores\arduino\arduino.h
	파일 / 클래스	arduino.h / —

void pinMode(pinNum, mode)		
기능	해당 핀을 입력으로 사용할지 출력으로 사용할지 설정	
매개변수	pinNum	설정하고 싶은 핀 번호
	mode	입력(INPUT) 또는 출력(OUTPUT)설정
리턴 값	void	없음

void digitalWrite(pinNum, value)		
기능	지정된 핀에 디지털 값 출력	
매개변수	pinNum	출력할 핀 번호
	value	디지털 값 LOW(0V) 또는 HIGH(5V)
리턴 값	void	없음

void digitalRead(pinNum)		
기능	지정된 핀으로 입력된 디지털 값 읽어오기	
매개변수	pinNum	값을 읽어올 핀 번호
리턴 값	value	디지털 값 LOW(0V) 또는 HIGH(5V)

void analogWrite(pinNum, value)		
기능	지정된 핀에 아날로그 값 출력	
매개변수	pinNum	출력할 핀 번호
	value	0 ~ 255 값 : PWM으로 생성할 수 있는 파형 듀티비
리턴 값	void	없음

int analogRead(pinNum)		
기능	지정된 핀으로 입력된 아날로그 값을 읽어오기	
매개변수	pinNum	값을 읽어올 핀 번호
리턴 값	value	0 ~ 1023 범위 값 : 10비트 ADC 인식 범위

4. 아두이노 모니터링(Monitoring)

4-1. 모니터링이란?

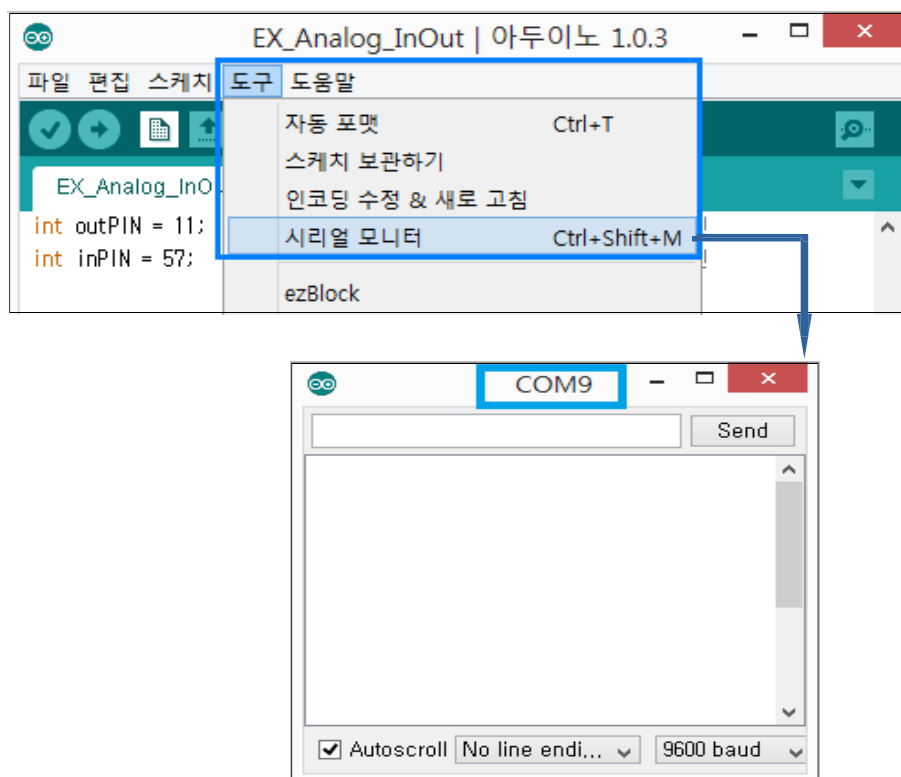
모니터링이라는 말은 한번 씩 들어 보았을 것이다.

제대로 일을 하고 있는지 몰래 가서 체크해 보는 것을 말한다. 쉽게 생각한다면 옛날에 암행어사와 같은 개념이다.

아두이노, 세탁기, 커피 메이커 등등 우리 주변에서 볼 수 있는 전자제품을 개발 할 때도 정상 동작 중인지 현재 어디까지 동작을 했는지 등등의 상태를 확인해야 할 경우가 발생한다.

이런 경우를 위해서 아두이노에서는 Serial Monitor라는 모니터링 프로그램을 제공하고 있다.

스케치 툴에서 실행시킬 수 있다.

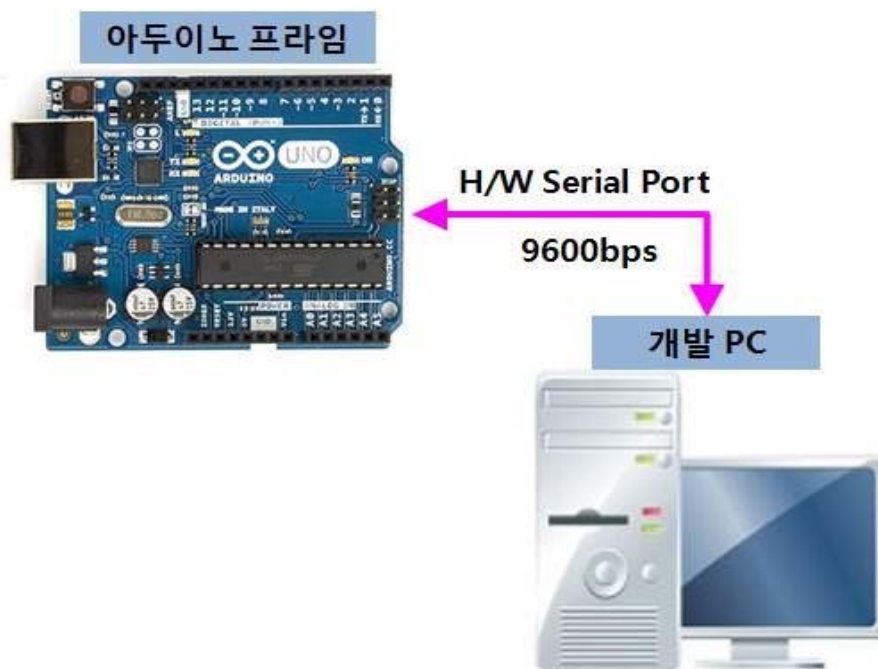


[시리얼 모니터 프로그램]

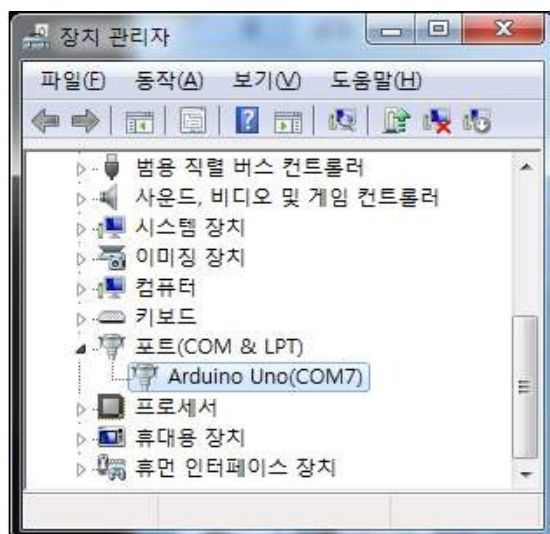
4-2. 아두이노 & 개발 PC 모니터링 연결

아두이노는 USB 케이블을 통해 개발 PC와 연결이 된다.

USB 케이블을 사용하면 전력공급 뿐만 아니라 연결된 개발 PC와 데이터를 주고받을 수도 있다. 이것을 **시리얼(Serial) 통신**이라 한다.



연결 후 개발 PC와 제대로 연결이 되었는지 어디에 연결이 되었는지 확인을 해 보자.



개발 PC의 7번 COM 포트에 연결된 것을 확인할 수 있다.

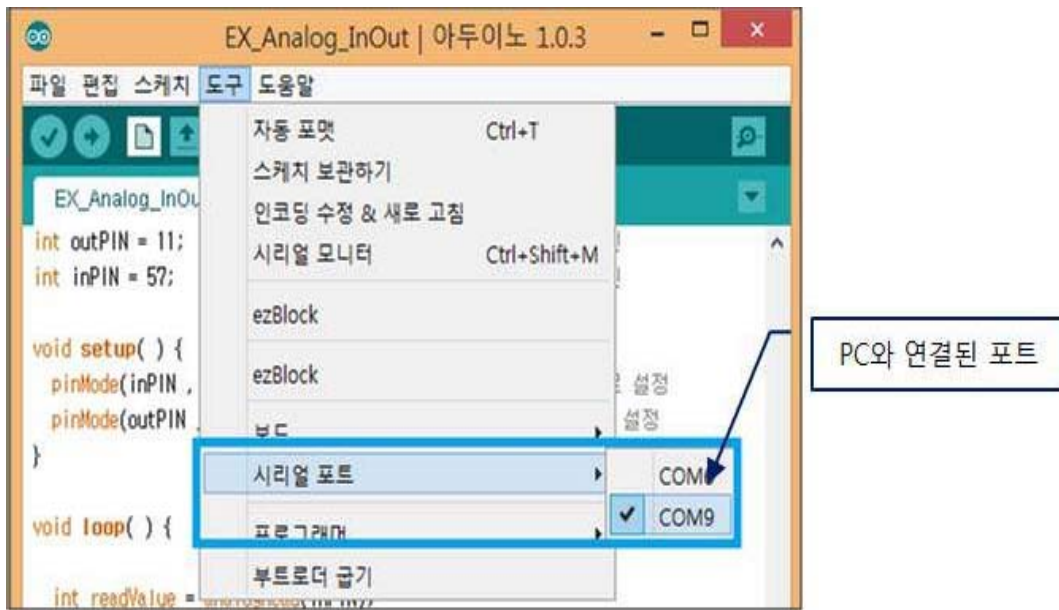
연결된 7번 COM 포트를 통해서 아두이노로부터 데이터를 받거나 보낼 것이다.

* 개인 PC와 연결된 포트 확인

시리얼 통신에 대한 자세한 사항은 뒤에서 더 자세히 배울 것이다.

개발 PC에서 연결을 확인했다면 아두이노의 시리얼 프로그램에서도 제대로 연결을 설정해 주어야 한다.

당연히 스케치 툴을 통해서 설정한다.



포트 설정 후 시리얼 모니터(Serial Monitor)를 실행해서 다음과 같이 확인 할 수 있다
그리고 하단의 메뉴로 개발 PC와 데이터를 주고받기 위한 속도를 설정 할 수도 있다.



4-3. 아날로그 신호 모니터링 하기

우리는 디지털 값과 아날로그 값을 읽어올 수 있는 방법을 배웠다.

디지털 입력의 경우 LOW(0) 또는 HIGH(1)의 값밖에 가지지 않기 때문에 LED를 켜거나 끄는 방법으로 값을 알 수 있을 것이다.

그렇다면 아날로그 입력 값은 어떻게 알 수 있을까?

0 ~ 1023 사이의 정확한 값을 눈으로 보고 싶다면 어떻게 해야 할까?

바로 시리얼 모니터(Serial Monitor)라는 모니터링 프로그램을 사용하면 된다.

연결하는 법과 시리얼 모니터(Serial Monitor) 실행시키는 방법은 앞에서 살펴보았다.

▶ 시리얼 모니터(Serial Monitor)에 출력하기

시리얼 모니터(Serial Monitor)에 출력하기 위해서는 다음과 같은 단계를 수행한다.
딱 2가지만 기억하면 된다. 아주 쉽다.

- ① 약속한 속도로 데이터를 주고받아서 처리해주는 시리얼(Serial) 객체 생성

void Serial.begin(int speed) - 지정된 speed로 PC와 데이터 주고받기 시작

예) Serial.begin(9600); - 초당 9600비트 데이터 주고받는 통신 시작

- ② 출력함수 println(), print()를 사용해서 원하는 데이터를 화면 출력

void Serial.print(data) - data를 시리얼 모니터 화면에 출력

void Serial.println(data) - data를 시리얼 모니터 화면에 출력 후 줄 바꿈

예) int readValue = analogRead(inPIN);

Serial.println(readValue);

입력받은 아날로그 데이터 값을 시리얼 모니터(Serial Monitor)에 출력해 보자.

[아날로그 데이터를 시리얼 모니터에 출력하기]		
01	int inPIN = A0;	//- 아날로그 신호 입력 핀
02		
03	void setup() {	
04	pinMode(inPIN , INPUT);	//- 입력으로 설정
05	Serial.begin(9600);	//- 통신 시리얼과 속도 설정
06	}	
07		
08	void loop() {	
09		
10	int readValue = analogRead(inPIN);	//- 아날로그 신호 측정
11		
12	Serial.println(readValue);	//- 읽어온 값 출력
13		
14	delay(100);	
15	}	

시리얼 모니터(Serial Monitor)를 실행해서 출력되는 값을 확인 할 수 있다.

