



인하공업전문대학
INHA TECHNICAL COLLEGE

사물인터넷 5주차

인하공업전문대학 컴퓨터 정보과
김한결

사물인터넷 개요

■ 초연결 사회와 4차 산업혁명

• 초연결 사회

- 2008년 가트너에서 처음 사용
- 4차 산업혁명 시대의 특징 중 하나를 설명하는 말
- 시스템과 사물의 연결을 넘어 모든 사물과 데이터, 인간의 행동 등이 마치 거미줄처럼 촘촘하게 네트워크로 연결되는 사회
- IoT: 모든 사물 또는 시스템이 작게는 센서부터 크게는 도시 규모까지 모두 연결되는 기술기반

• 산업 혁명의 변화

- 1차 산업혁명: 18세기 증기기관의 발명으로 촉발되어 도시화로 이어짐
- 2차 산업혁명: 19세기 후반 전기에 의한 산업 구조의 변화로 대량생산이라는 새로운 체제로 접어든 시점
- 3차 산업혁명: 디지털 기술의 발전으로 이루어진 개인용 컴퓨터와 네트워크 등의 정보통신기술 혁신을 의미
- 4차 산업혁명: 로봇, 인공지능, 나노기술, 생명 공학, 자율주행 등 수많은 기술 혁신이 발생, IoT 데이터로 모든 것이 연결



그림 8-1 초연결 사회

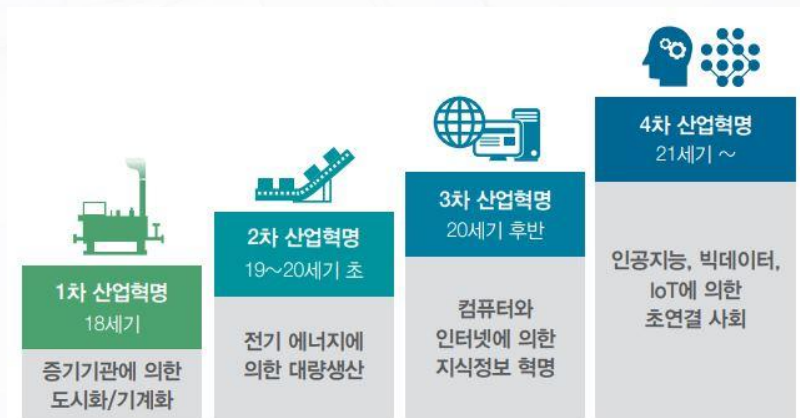


그림 8-2 산업혁명의 변화

사물인터넷 개요

■ IoT의 이해

• IoT의 개념

- 1999년에 케빈 애시톤이 처음 제안했고 이후 관련된 시장 분석 자료 발표를 통해 대중화된 용어
- IoT라는 용어는 제안된 후 아직도 발전을 거듭하며 보완 중이므로 기술적, 이론적으로 아직 완성되지 않은 상태
- 좁은 의미: 유비쿼터스 환경에서 사물끼리 통신을 주고받는 것
- 넓은 의미: 인간, 사물, 서비스가 인간의 명시적 개입 없이 상호 협력해 센싱, 네트워킹, 정보 처리 등의 지능적 관계를 형성 하는 사물 공간 연결망

• IoT의 요소

- 연결: 기기와 서비스 인프라를 묶는 역할을 하는 유·무선통신 혹은 네트워크 인프라
- 정보: 정보의 생산은 곧 주변 환경과 상태의 정보를 얻어내는 센싱 기술, 정보는 생성되는 지점에 의미가 있음
- 서비스: 사람에게 제공을 목적으로 정보를 묶고 가공해 제공하는 서비스, 서비스를 제공하기 위한 서비스 연동 인터페이스 기술

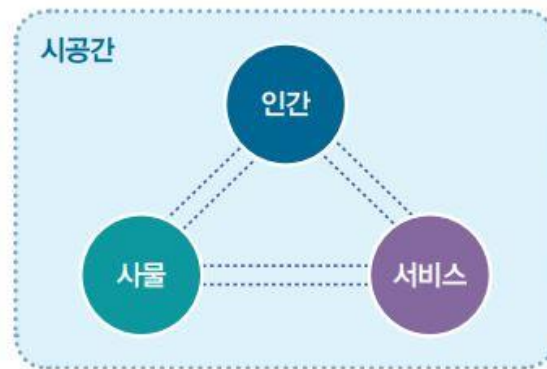


그림 8-3 IoT의 개념

■ IoT의 주요 기술

■ 센싱 기술

- 정보를 수집·처리·관리하고 이를 서비스로 구현하기 위한 환경을 지원하는 기술
- 물리적 센서는 표준화된 인터페이스와 정보처리 능력을 탑재한 스마트 센서로 발전
- 이미 센싱한 데이터로부터 특정 정보를 추출 하는 가상 센싱 기능도 구현

• 센싱기술의 특징

- 프로토콜의 이종성
 - » 디바이스는 상이한 플랫폼에서 동작하는데, 이때 디바이스 간 통신 프로토콜이 서로 다름
- 민감한 데이터의 수집으로 인한 정보 보안 문제
 - » 데이터의 대량 처리를 위해 '정보 보안'을 유지해야 함
- 자원 제약성이 있음
 - » 제한된 배터리 전력 수명으로 CPU나 메모리 등의 자원에 제약을 많이 받음
- 이동성이 있음
 - » IoT 기기는 이동성이 높아 네트워크 토폴로지가 동적으로 만들어지는 경우가 많음
 - » 센서 기기는 낮은 성능과 좁은 대역폭으로 연결성, 즉 데이터 품질이 좋지 않음

사물인터넷 개요

■ IoT의 주요 기술

■ 유·무선통신 및 네트워크 인프라 기술

• 근거리 통신기술

- 대표적인 근거리 통신기술은 WPAN으로, 지그비, 블루투스, NFC 등과 같은 무선통신기술이 해당

• 무선 이동통신기술

- 대표적인 무선 이동통신기술에는 Wi-Fi, 4G 롱텀 에볼루션인 LTE, 3GPP 릴리즈 8 등
- 2G 기술 : GSM, CDMA와 3G의 UMTS, WCDMA, HSDPA, Wibro
- 4G 기술 : 릴리즈8
- 5G 기술: 3GPP 릴리즈 15등
- 그 외: 위성 통신, 가시광 통신 등
- 차량특화기술: WAVE, DSRC
- 차세대 기술: 6LoWPAN, LoRa/NB-IoT 네트워크 등

표 8-1 저전력 IoT 네트워크의 예

영역	항목	주파수	커버리지	전송속도	표준
협대역	Wi-Fi	2.4Ghz, 5Ghz	20~100m	2~14Gbps	802.11b~ax
	지그비Zigbee	868,900~928Mhz	10~100m	250Kbps	802.15.4
	블루투스Bluetooth	2.4Ghz	10m	1~2.1Mbps	802.15.1
광대역	SigFox	8~900Mhz	~13Km	100bps	비표준
	LoRa	8~900Mhz	~10Km	10Kbps	비표준
	NB-IoT	200Khz	~15Km	200Kbps	3GPP 릴리즈 13

3GPP의 주요 프로젝트

프로젝트 명	주요 기술	주요 특징
Release 99	3G (UMTS/WCDMA)	음성 및 데이터 통합 서비스, 패킷 기반 전송 기술 도입
Release 8	4G (LTE)	고속 데이터 전송, IP 기반 네트워크, OFDMA 사용
Release 15	5G NR (New Radio)	초고속, 초저지연, 대용량 연결, mmWave 사용
Release 18	5G Advanced	고주파 대역 확장, 엣지 컴퓨팅 통합, IoT 지원 강화

■ IoT의 주요 기술 (부록)

▪ 좁은 대역폭 vs 넓은 대역폭 비교

구분	좁은 대역폭 (Narrowband)	넓은 대역폭 (Wideband)
주파수 대역	수 kHz에서 수십 kHz	수 MHz에서 수 GHz
데이터 전송 속도	저속 (수 kbps)	고속 (수 Mbps ~ Gbps)
전력 소모	적음	많음
전파 감쇠	적음	많음 (고주파수 대역에서 심함)
신호 전송 거리	비교적 멀리 전송 가능	가까운 거리 또는 중계기 필요
주 사용 분야	IoT, 원격 모니터링, 저전력 센서	스마트폰, 영상 스트리밍, 고속 인터넷
실시간 처리	어려움	가능

■ IoT의 주요 기술

■ 유·무선통신 및 네트워크 인프라 기술

• 유선통신기술

- 광대역 통합망, 시리얼 통신, 이더넷 등
- 이더넷 기반의 표준 IP 통신 방식: 기존의 폐쇄적이고 특정 환경에서만 동작하는 프로토콜을 대체 또는 변환해서 사용
- 개인 무선 네트워크 기술: 저전력 기반의 경량 데이터 전송 방식 게이트웨이를 통해 IP망 기기와 연결해 서비스를 완성

■ 서비스 인터페이스 기술

- 인간, 사물, 서비스를 결합하는 기능을 수행해 인간에게 제공되는 응용 서비스와 연동하는 역할을 수행

표 8-2 IoT의 3대 주요 기술

기술	설명	세부 기술 예
센싱 기술	사람의 오감을 대신하여 정보를 수집하는 도구로, 현재는 사람의 오감으로 인지 불가능한 영역까지 확장	온도, 습도, 조도, 가스, 초음파 등의 센서, 원격 감지, SAR 레이다, 위치, 모션 센서 등
유·무선통신 및 네트워크 인프라 기술	유·무선에 기반한 근거리 및 원거리 신호 전송 통신	WPAN, Wi-Fi, CDMA/LTE, 블루투스, 이더넷, BcN, 위성통신, 시리얼 통신, PLC 등
서비스 인터페이스 기술	센싱, 가공, 추출, 처리, 저장, 판단, 상황 인식, 인지, 보안, 개인정보 보호, 인증, 객체 정형화 등의 서비스를 연결하는 접점 기술	오픈 API, 오픈 플랫폼, 미들웨어, 데이터 마이닝, 웹 서비스, 소셜 네트워크 인터페이스 등

■ IoT 플랫폼

■ 디바이스 플랫폼

- 데이터 수집을 쉽게 하기 위해 대개 하드웨어부터 운영 소프트웨어까지 플랫폼으로 제공
- 디바이스 종단의 단말 플랫폼, 단말 간 연결과 서비스 서버 연결까지 제공하는 단말 연결 플랫폼이 포함
- 경량화된 제품이 많음

■ 연결 플랫폼

- 통신 인프라 기술을 기반으로 다양한 기기로부터 수집된 데이터를 클라우드에 적재하기 위한 통합된 방식의 연결 플랫폼
- 데이터 연결 플랫폼에 해당하는 프로토콜 기술은 CoAP, MQTT, AMQP 등

■ 데이터 플랫폼

- 데이터 플랫폼은 수집·적재된 데이터를 보관·처리하는 저장소와 데이터를 분석·처리하는 기술로 구분
- 개별 디바이스로부터 수집된 데이터에서 유사한 패턴과 현상을 도출하고 분류하는 다양한 분석 기술을 포함
- 분석 결과를 추상화하고 그림이나 도표로 시각화하는 데이터 분석을 지원하는 툴 조합을 제공

■ 서비스 플랫폼

- 스마트시티, 환경 기상, 에너지, 스마트 홈, 농업, 건강, 안전 등의 다양한 서비스에 지능형 기술을 적용하고 확장을 지원
- 모니터링 기술, 서비스 생태계 확장 기술, 서비스 인터페이스 기술로 나뉨

❖ 사물 인터넷의 개념?

- 개체나 센서, 일상용품이 네트워크에 연결되고 정보처리 능력을 갖게 되어, 인간의 개입이 거의 없이 정보를 생성, 교환, 소비하여 정해진 기능을 수행하는 것

- 1 센서 – 물리적 환경의 변화를 감지하고 데이터를 수집
- 2 엣지 디바이스 – 수집된 데이터를 처리하고 간단한 분석 수행
- 3 게이트웨이 – 엣지 디바이스와 서버(클라우드) 간의 중계 역할
- 4 서버/클라우드 – 데이터를 저장, 분석 및 시각화

사물인터넷 활용분야

❖ 사물 인터넷 활용 분야

스마트 홈



자동화된 가전제품,
보안 시스템

스마트 시티



교통 관리, 공공 안전,
환경 모니터링

스마트 팩토리



생산 공정 최적화,
장비 상태 모니터링

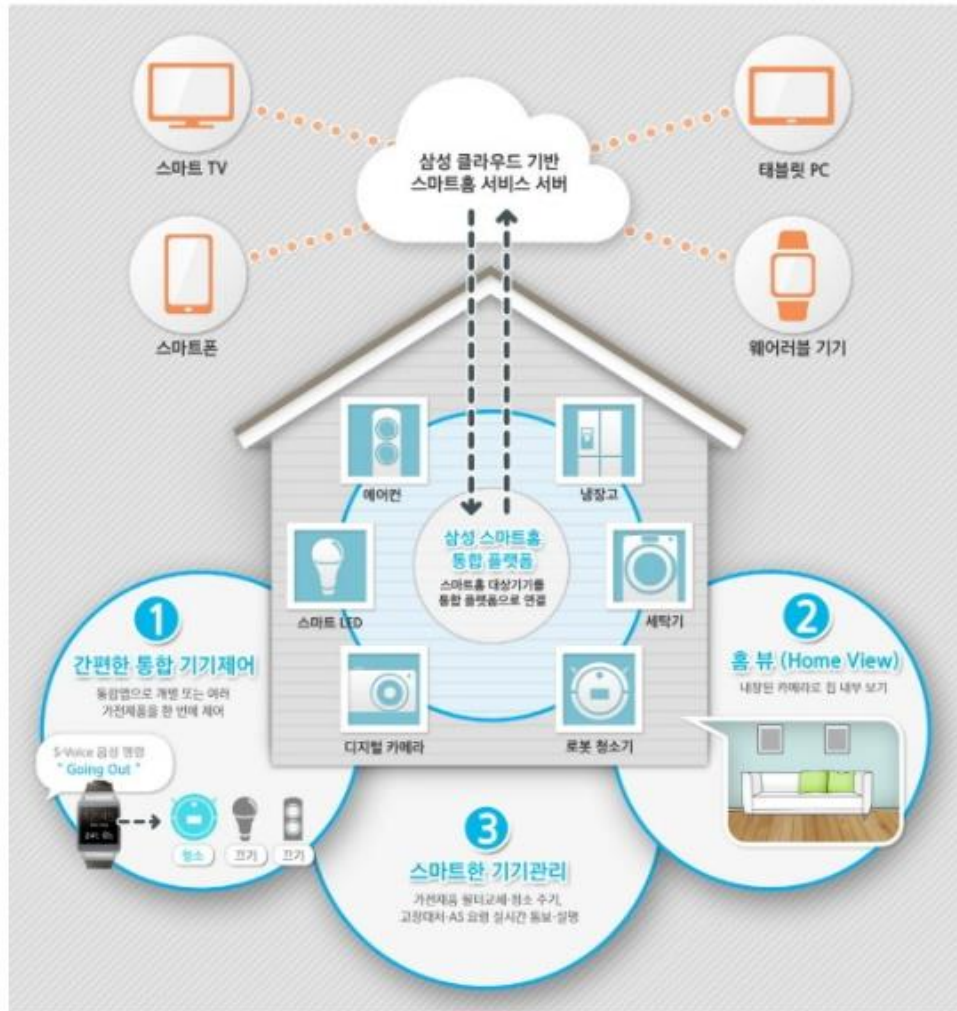
헬스케어



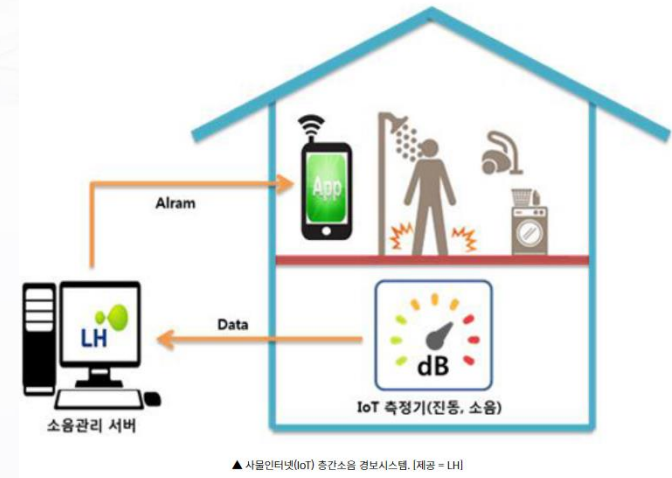
원격 의료, 웨어러블 디바이스

사물인터넷 활용분야

스마트 홈



출처 - <https://bahns.net/5673511>



출처 - <https://www.boannews.com/media/view.asp?id=517799>

사물인터넷 활용분야



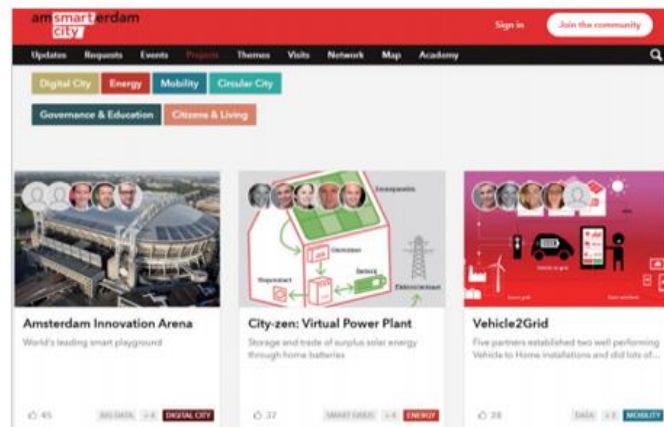
스마트 시티

【사업 개념도】



□ 주요 서비스

구분	서비스 사진	주요 내용
배리어프리 내비게이션		○ 지하철 등의 실내공간에서 교통약자의 이동경로 및 환승경로를 키오스크 또는 스마트폰을 통해 길안내 서비스를 제공
배리어프리 스테이션		○ 교통약자를 배려한 안내 및 편의시설을 갖춘 정류장이자 승차공유플랫폼 등을 이용할 수 있는 거점으로서의 역할
배리어프리 승차공유 플랫폼		○ 교통약자 유형 및 서비스 이용 목적에 맞는 교통 서비스(자전거배차, 무상카풀, DRT, 택시 등승)를 제공



< Amsterdam Smart City 홈페이지(출처:서울디지털재단 「스마트시티 사례집」) >



< POO WiFi (출처: <https://www.coloribus.com>) >

사물인터넷 활용분야

스마트 팩토리



출처 - <https://www.mmkorea.net/news/articleView.html?idxno=7461>



출처 - <https://www.hankyung.com/article/2018111319111>



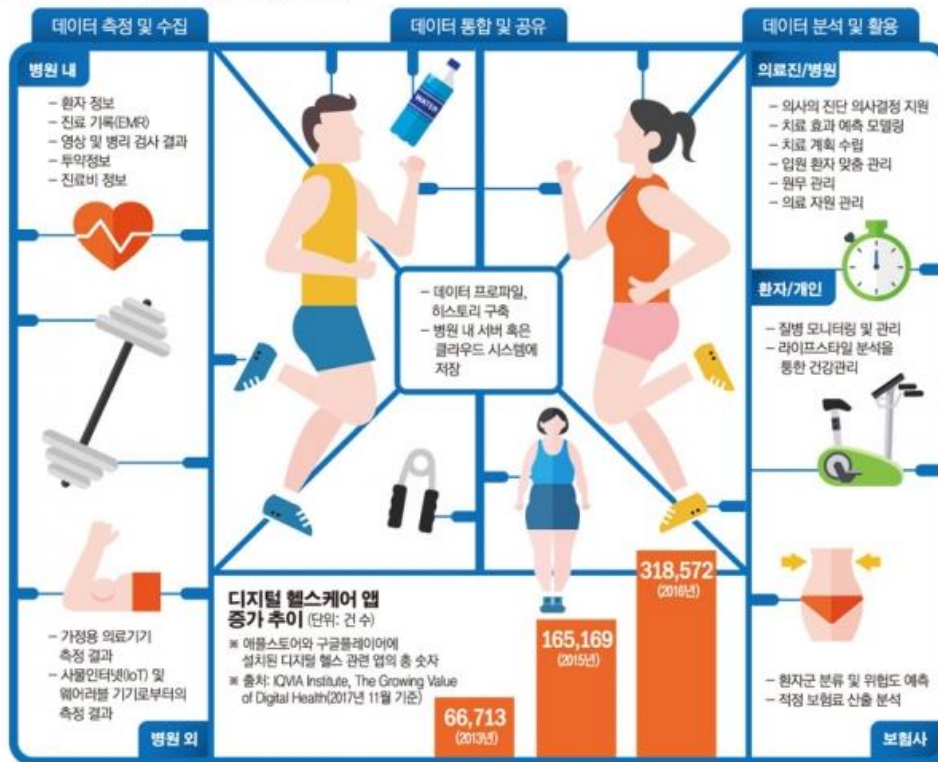
출처 - <https://live.lge.co.kr/2407-lg-smartfactory/>

사물인터넷 활용분야

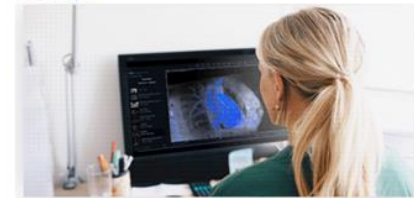


헬스케어

헬스케어 데이터 수집·통합·활용의 흐름



Arterys



2D and 4D Flow

Visualize and quantify blood flow precisely anywhere in the heart

Enabling a better patient experience, faster CMR and precise analysis Arterys Cardio AI is built on the most advanced web rendering technology for medical imaging. You can load studies of any size, from 2D PC to multiple 4D Flow datasets. Enjoy breathtaking graphics and automation for a fast, comprehensive view. You've got to see it - and experience it - to believe it.

Viz.ai



Auto-detect suspected diseases.
Accelerate time to diagnosis and treatment.

The all-in-one solution autodetects suspected diseases across a wide range of therapeutic areas in seconds.

[자료: 각 회사 홈페이지]

출처 : 의료기기뉴스라인
(<http://www.kmdianews.com>)

출처 - <https://www.etoday.co.kr/news/view/1649452>

Arduino C++ 프로그래밍

■ 변수 선언 형식

- 꼭 필요한 변수형

- C++의 변수형은 다양하지만 아두이노 프로그램 작성에 필요한 변수형은 한정적이므로 필수 변수형만 살펴보기로 한다.
- 아래 표의 변수형 만으로도 대부분의 프로그램을 작성할 수 있다.

변수형	용도	범위	크기(바이트)
bool	참과 거짓을 표현하는 정수	0,1	1
int	프로그램 내의 카운트용 정수	우노 : -32,768~32,767	2
		ESP8266 : 약 -21억~21억	4
unsigned long	가장 큰 정수. 시간 표시용	0~약 42억	4
float	소수점 있는 숫자	유효숫자 6자리	4
char	문자 한 개	-128~127	1
String	문자열을 보관하는 클래스	연속된 문자 묶음	가변

- int형 변수는 마이크로컨트롤러의 종류에 따라 값의 범위가 달라지므로 주의한다.

변수에 대한 이해

■ 변수 선언 형식

- unsigned long은 부호가 없는 long 형 변수라는 뜻이다. 아두이노에서 가장 큰 정수로 주로 **밀리초 단위 표현에 사용된다.**
- int나 long과 같은 정수 변수는 표현 영역의 각 절반으로 양수와 음수를 표시한다.
- char는 영숫자나 특수 문자 한 개를 표현한다. 실제로는 1바이트를 차지하는 **부호 있는 정수이며 연산도 가능하다.**
- **String**은 C++의 변수형이 아니라 입문자들이 쉽게 문자열을 다루도록 **아두이노에서 만든 클래스다.**

변수에 대한 이해

■ 변수 선언 형식

• 변수명

- 영문자, 숫자, 그리고 (underscore)를 섞어 변수명을 만든다. 숫자로 시작하면 안되며, 영문자는 대소문자를 구분한다.
- 중간에 빈칸이 있으면 안된다.

```
int cnt;  
int Cnt;    // cnt와 Cnt는 서로 다른 변수명  
int cnt23;  
int cnt_no;  
int _cnt;   // _로 시작 가능  
int 2cnt;   // 오류. 숫자로 시작  
int cnt no; // 오류. 중간에 빈칸이 있음
```

- 여러 단어로 변수명을 만들 때는 첫 단어는 소문자, 이후의 단어는 첫 글자만 대문자로 작성하는 **낙타등 표기법(camel case)** 사용

```
int totalNoOfCnt;
```

변수에 대한 이해

■ 변수의 적용 범위

• 전역 변수

- 변수를 `setup()`보다 앞에 선언하면 모든 영역에서 변수를 부를 수 있다. 이러한 변수를 **전역 변수(global variable)**라고 한다.
- 전역 변수는 메모리가 끝까지 유지되며 가용 메모리가 줄어든다.

• 지역 변수

- 대부분의 변수는 프로그램의 특정 함수에서만 사용된다. 이러한 변수를 **지역 변수(local variable)**라고 한다.
- 지역 변수는 선언된 함수 또는 블록 안에서만 통용된다.
- 메모리 활용면에서 유리하다.

변수에 대한 이해

■ 변수의 적용 범위

- 프로그램[3-2]를 작성하고 시리얼 모니터를 연 상태에서 [스케치] - [업로드]를 실행 후 출력 내용을 살펴보자.

프로그램 3-2 var-global-local

```
001 int i = 3; // 전역 변수
002 int j = 4;
003 void setup() {
004     int j = 5; // j(지역 변수)는 함수 setup() 내에서만 유효
005     Serial.begin(115200);
006     Serial.println();
007     if (j > 3) { // 지역 변수 j: 5
008         int i = j + 1; // i(지역 변수)는 if문 안에서만 유효
009         Serial.println(i); // i: 6
010     }
011     Serial.println(i); // 전역 변수 i: 3
012     Serial.println(j); // 지역 변수 j: 5
013 }
014
015 void loop() {
016
017 }
```


변수에 대한 이해

■ 변수의 속성 지정하기

- **const**

- 변수 선언 시 값이 중간에 바뀌지 않는다면 변수형 앞에 const를 붙이는게 좋다.

```
const int pinNo = 2;
```

```
pinNo = 2;    // 오류. 상수형 변수의 값은 변경할 수 없음
```

- **static**

- 함수 안의 지역 변수라도 값이 유지되어야 하는 경우가 있다. 이때 static을 붙여 변수를 선언한다. static 변수는 값을 바꿀 수 있다.
- **static**으로 선언된 변수는 메모리에서 전역 변수처럼 관리된다.

변수에 대한 이해

■ 변수의 속성 지정하기

- **static**

```
void displayOled() {  
    static bool first = true;  
    if (first) {  
        first = false;  
        Serial.println("first entry!");  
    }  
    else {  
        Serial.println("repeat entry!");  
    }  
}
```

- 함수 displayOled()가 반복적으로 호출되더라도 static 변수 first의 값은 유지된다.
- **static**으로 선언된 변수의 초기화는 최초 한번만 수행된다. 따라서 두 번째 수행부터 first의 값은 false이다.

숫자형 변수

■ bool

- bool 또는 boolean은 참(true, 1)과 거짓(false, 0)의 상태만 보관하는 특수한 숫자형 변수이다.
- bool형 변수는 1바이트를 차지하는 정수로 0과 1의 값을 갖는다.
- C++에서 **0은 거짓, 1은 참으로 예약되어** 있으며 아두이노에서는 추가로 **HIGH와 LOW를 각각 1과 0으로 지정했다.**

```
bool ledOn = LOW;    // LOW는 0이므로 ledOn은 거짓(0)
bool ledOn = false;  // false도 0이므로 ledOn은 거짓(0)
bool ledOff = HIGH;  // HIGH는 1이므로 ledOff는 참(1)
bool ledOff = true;  // true도 1이므로 ledOff는 참(1)
bool ledStatus = 3;  // 3을 지정해도 값은 1로 되어 참(1)
```

숫자형 변수

■ int

- 가장 간단히 처리할 수 있는 정수형

```
int onMil = 1500;  
int offMil = 500;
```

- ESP8266에서는 4바이트(32비트)로 이루어진 정수를 표현한다.
- 아두이노 우노는 2바이트(16비트) 정수를 표현하므로 주의한다.

구분	메모리	정수 범위
아두이노 우노	2바이트	$-32,768(-2^{15}) \sim 32,767(2^{15}-1)$
ESP8266	4바이트	$-2,147,483,648(-2^{31}) \sim 2,147,483,647(2^{31}-1)$

■ unsigned long

- long은 4바이트 정수 변수형, unsigned long은 부호 없이 정수를 표현하므로 0에서 약 42억까지 표현할 수 있다.
- ESP8266에서 int와 long은 각각 4바이트 정수형으로 같다.
- 주로 밀리초 단위의 시간을 보관하는 변수로 사용, 아래의 **mills()** 함수는 밀리초를 돌려주는 아두이노 내장 함수이다.

숫자형 변수

■ float

- 소수점을 표시하는 변수형이다. 4바이트 메모리에 숫자의 **부호, 유효숫자, 지수**를 분리하여 표현한다.

```
float temp = 25.5;
```

- 유효숫자가 10진수 기준 6자리 정도로 100만 이상의 수를 정밀하게 표현하는 것은 힘들다.
- 아두이노에서는 이 변수형을 처리하지 위해 부가 루틴이 요구되므로 필요할 때만 사용한다.
- 더 정밀한 큰 숫자를 취급해야 할 경우 double을 사용한다.

숫자형 변수

■ 간혹 사용되는 정수형

- 일반적으로 int를 사용하고 큰 수에는 unsigned long을 사용하지만 타인의 라이브러리를 이용할 경우도 있어, 다른 정수 표현도 알아두어야 한다.

- **byte**

- 1바이트로 부호없이 정수 표현, 0에서 255까지를 커버하며 범위가 정해져 있는 핀 번호 등에 사용하여 메모리를 절약한다.
- Unsigned char과 같은 의미이다.

```
byte ledPin = 2;
```

- **uint8_t**

- typedef라는 지시어를 이용해 정의한 데이터형으로 unsigned int 8bit type이란 뜻이다. Byte나 unsigned char와 같다.

```
uint8_t ledPin = 2;
```

숫자형 변수

■ 산술연산자

- 산술연산자는 숫자형 변수나 상수를 이용한 산술식을 위한 연산자이다.

산술연산자	의미
<code>a = b</code>	오른쪽 값을 왼쪽 변수로 복사
<code>a + b</code>	덧셈
<code>a - b</code>	뺄셈
<code>a * b</code>	곱셈
<code>a += b</code>	<code>a = a + b</code>
<code>a / b</code>	나눗셈
<code>a % b</code>	나머지
<code>a++</code>	값 사용 후 증가
<code>++a</code>	증가 후 값 사용
<code>a--</code>	값 사용 후 감소
<code>--a</code>	감소 후 값 사용

❖ `a++`와 `++a`는 명령어 안에서 사용될 때에는 값이 다르게 반환될 수 있으므로 주의한다.

■ 배열을 초기화하고 한꺼번에 다루기

- 배열은 선언할 때 초기값을 정하거나, 한꺼번에 다룰 수도 있다.

```
// 배열을 선언하고 초기화
float temp[3] = {23.5, 34.5, 34.2};

// 합계 변수를 선언하고 초기화
float sum = 0;

// 배열 전체의 합을 구함
for (int i = 0; i < 3; i++) {
    sum = sum + temp[i];
}

// 평균을 냄
float ave = sum / 3; // sum: 92.20, ave: 30.73
```

■ 배열을 초기화하고 한꺼번에 다루기

- **for문으로 모든 배열 요소 다루기**
- 배열을 통으로 처리할 때는 for문을 사용한다.

```
for ( 변수 선언문 ; 조건식; 매회 마지막 명령어 ) {  
    // 명령어 그룹  
}
```

- 처음에만 변수 선언문을 실행, 참이면 명령어 실행, 이후 조건식 테스트를 반복한다. 조건식이 거짓이면 수행을 마친다.

```
for (int i = 0; i < 3; i++) {  
    sum = sum + temp[i];  
}
```

- 앞의 문장은 아래와 같이 수행된다.

```
sum = sum + temp[0];  
sum = sum + temp[1];  
sum = sum + temp[2];
```

- 배열을 초기화하고 한꺼번에 다루기
 - 배열 초깃값 지정

```
변수형 배열명[n] = {초깃값0, 초깃값1, ... 초깃값n-1};
```

- n은 배열의 개수를 뜻한다.

```
float temp[3] = {23.5, 34.5, 34.2};
```

■ 배열을 초기화하고 한꺼번에 다루기

- 배열을 모두 0으로 초기화하는 방법
 - 아래와 같이 지정하면 모든 요소가 0으로 초기화 된다.

```
float temp[3] = {};
```

문자형 변수

■ char

- char는 c++에서 문자를 나타내는 변수형이다. char로 선언한 변수는 영숫자나 특문자를 보관하며 문자는 'a'와 같이 따옴표로 묶는다.
- 아두이노에서 한글은 크기가 3바이트로 char로 선언된 변수에 보관할 수 없다.

```
char c = 'a';
```

• 아스키(ASCII) 코드

- char형 변수는 1바이트의 부호가 있는 정수이므로 -128~127의 값을 가진다. 이 중 0~127 범위의 숫자는 문자 표시 용도로 쓰인다.
- 문자 표시 코드 체계를 아스키(ASCII, American Standard Code For Information Interchange) 코드라고 한다.

코드 범위(10진수)	용도	설명
0 ~ 31, 127	제어문자	특수한 제어 기능을 하며 눈에 보이지 않음
32~126	인쇄 가능 문자	프린터나 디스플레이 장치에 출력할 수 있는 눈에 보이는 문자
48~57	숫자	0~9
65~90	알파벳 대문자	A~Z
97~121	알파벳 소문자	a~z

문자형 변수

■ char

- 코드표의 제어 문자나 특별한 문자는 프로그램 코드에서 역슬래시(\)를 붙여 '\0' 과 같은 방식으로 표현한다.

표기	코드(10진수)	코드(16진수)	설명
\'	39	27	작은 따옴표
\"	34	22	큰 따옴표
\\	92	5c	역슬래시
\0	0	00	널(null) 문자, 문자열의 끝
\n	10	0a	새 줄(new line)
\r	13	0d	캐리지 리턴(carriage return) ¹
\t	9	09	수평 탭

```
char c1,c2,d1,d2,e1,e
```

```
c1 = '\0';
```

```
c2 = 0;
```

```
d1 = '\n';
```

```
d2 = 10;
```

```
e1 = 'a';
```

```
e2 = 97;
```

- 우측 상단 코드에서 변수 c1과 c2, d1과 d2, e1과 e2는 같은 값을 가진다.

문자형 변수

■ char

- char의 정수 범위

```
char c = 'a'; // 초깃값, 아스키 97
c = 'b'; // 값 복사, 아스키 98
c = c + 2; // c : 'd', 연산, 아스키 100

Serial.println(c); // d가 출력됨
Serial.println((int)c); // int형으로 변환. 100이 출력됨
```

- 상단의 표는 char의 활용들을 보여준다.
- char는 속성상 8비트의 부호가 있는 정수이므로 연산이 가능하다. Serial.println()에서 char형의 변수인 c는 문자로 출력되므로 정수로 바꾸고 싶다면 (int)c와 같이 형 변환 표시를 해야 한다.
- char 범위의 정수는 int8_t 변수형이다. 해당 변수형은 정수로 출력되므로 문자 출력을 원하면 하단과 같이 char형으로 선언한다.

```
int8_t c = 'a'; // 초깃값, 아스키 97
c = c + 1; //
Serial.println(c); // 숫자 98이 출력됨
Serial.println((char)c); // 문자 b가 출력됨
```


문자형 변수

■ char

- 16진수

- char 한 개를 2진수로 표현하면 8자리나 되어서 불편하다. 따라서 4 비트씩 16진수 2자리로 표현하는 방법을 사용한다.
- 10에서 15까지의 값에는 a에서 f까지의 문자를 사용한다.

10진수	2진수	16진수	10진수	2진수	16진수
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	a
3	0011	3	11	1011	b
4	0100	4	12	1100	c
5	0101	5	13	1101	d
6	0110	6	14	1110	e
7	0111	7	15	1111	f

문자형 변수

■ char

- 문자 a의 아스키 코드는 97이며 2진수로는 01100001이고 16진수로는 61이다. 프로그램에서는 다음과 같다.

```
char c1,c2,c3,c4;
```

```
c1 = 'a';
```

```
c2 = 97;
```

```
c3 = 0b01100001;
```

```
c4 = 0x61;
```

- 변수 c1, c2, c3, c4의 내용은 동일하다. 2진수 앞에는 0b를 붙이고 16진수 앞에는 0x를 붙여 표기한다.

문자형 변수

■ char 배열

- C++에는 문자열을 직접 표현하는 변수형은 없다.
- 문자열은 char 변수의 배열(array) 또는 char 포인터(pointer)를 사용하여 문자열을 표시한다.

선언 방법	용도
<code>char</code> 배열명[배열 크기]	char형 배열에 문자열을 직접 보관
<code>char *</code> 변수명	임의의 메모리에 문자열을 보관하고 문자열의 주소를 <code>char *</code> (char 포인터) 변수명에 지정함

- 최소한의 개념과 C++의 char 배열과 String을 서로 변환하는 방법만 알아도 대부분의 문제는 해결된다.

문자형 변수

■ char 배열

• 배열을 만드는 방법

- 변수형 뒤에 nm[8]과 같이 변수 개수를 지정한다.

```
char nm[8] = "gildong";
```

- char 배열 nm은 8개의 요소로 이루어져 있고, nm[0]~nm[7]로 8개이다. 요소의 위치를 나타내는 0~7은 첨자(index)라고 한다. 첨자는 0부터 시작하므로 마지막 첨자는 배열 크기보다 1이 작다.

```
char nm[8] = "gildong";  
char nm1[8] = {'g', 'i', 'l', 'd', 'o', 'n', 'g', '\0'};
```

- 보관하는 문자의 개수가 달라질 수 있다. 따라서 마지막 문자 다음에 '\0'를 삽입하여 문자열의 끝을 표시하고 실질적으로 일곱 문자를 보관할 수 있다.
- '\0'는 널(null)이라고 부르고 정수로는 0이다.

문자형 변수

■ char 배열

- 아래 nm과 nm1은 같다.

```
char nm[8]= "gildong";  
char nm1[8]= {'g','i','l','d','o','n','g','\0'};
```

- 배열 nm1처럼 요소 각각에 초깃값을 주기 힘들므로 배열 nm처럼 문자열으로 준다. char 배열에 초깃값 'gildong'과 같이 주면 마지막으로 '\0' 을 컴파일러가 알아서 추가한다.

```
char nm[8] = "gildong";
```

주소	330~333	배열명 nm은 첫 번째 배열 요소인 nm[0]의 주소를 나타내는 상수입니다.
내용	120	
변수	nm	

주소	120	121	122	123	124	125	126	127
내용	g	i	l	d	o	n	g	\0
변수	nm[0]	nm[1]	nm[2]	nm[3]	nm[4]	nm[5]	nm[6]	nm[7]

문자형 변수

■ char 배열

- char 배열은 문자열을 다룰 때에 사용한다.
- 배열 이름은 배열의 첫 요소의 주소를 나타내는 변수로 취급된다.
- 출력하면 첫 요소부터 '\0'까지 출력한다.

```
char nm[8]= "gildong";  
char nm1[8]= {'g','i','l','d','o','n','g','\0'};
```

```
Serial.println(nm); // gildong이 출력됨  
Serial.println(nm1); // gildong이 출력됨
```

- char 배열에서 개별 첨자를 아래와 같이 다룰 수 있다.

```
char nm[8] = "gildong";  
nm[3] = '\0';  
Serial.println(nm); // gil이 출력됨 nm[]: gil\0ong
```

문자형 변수

■ char 배열

- char 배열을 다루는 함수

- 아래 표는 문자열의 기본 함수이다. 표에서 '!'은 '또는'이란 뜻이다. cpy, cat, len, cmp는 각각 copy(복사), concate(연결), length(길이), compare(비교)에서 온 말이다.

용도	함수 사용	설명
복사	<code>strcpy(char배열, char배열 char포인터)</code>	첫 인수에 두 번째 인수의 값을 복사함 (첫 인수에 원래 있던 값은 없어짐)
연결	<code>strcat(char배열, char배열 char포인터)</code>	첫 인수에 두 번째 인수의 값을 덧붙임
길이	<code>strlen(char배열 char포인터)</code>	인수의 문자 개수를 돌려줌
비교	<code>strcmp(char배열 char포인터, char배열 char포인터)</code>	1, 0, -1의 숫자를 돌려줌 • 1 : 첫 인수가 클 때 • 0 : 두 인수가 같을 때 • -1 : 첫 인수가 작을 때

문자형 변수

■ char 포인터

- 포인터는 메모리 상의 특정 주소다. char*와 같이 *를 붙이면 변수형의 포인터 변수라는 의미다.
- char*는 메모리 어딘가의 문자열의 주소를 보관한다.
- char 포인터 변수와 char 배열명은 모두 포인터를 표시하지만 상수로 취급되는 char 배열명과 달리 char 포인터는 중간에 값을 바꿀 수 있다는 점이 다르다.

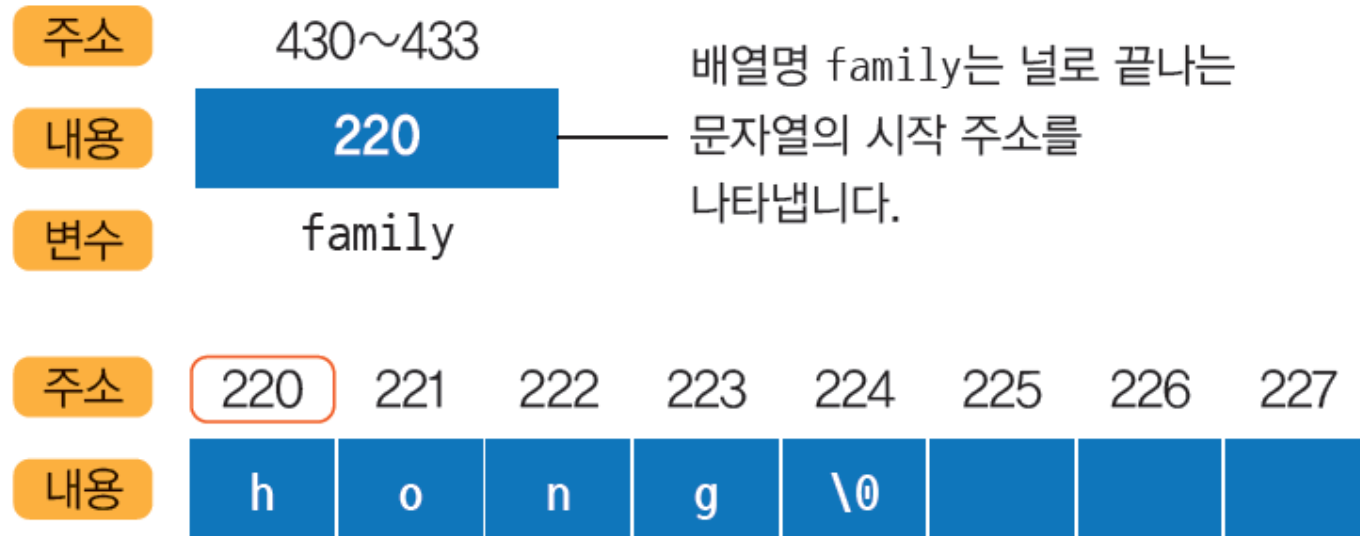
```
char nm[8] = "Gildong";  
char * family = "Hong";  
char * fullName;  
fullName = nm;           // fullName : Gildong  
fullName = family;       // fullName : Hong  
fullName = "Hong Gildong"; // fullName : Hong Gildong  
nm = "Hong Gildong";     // 오류. nm은 상수라서 변경 불가
```


문자형 변수

■ char 포인터

- C++에서 문자열 상수는 배열 첫 글자의 포인터를 의미한다. 메모리에는 'G','i','l','d','o','n','g','\0'문자의 저장소가 존재하며 'G'의 포인터를 문자열 상수인 것처럼 활용한다. 그래서 문자열을 전달받는 변수의 속성은 char *로 선언되어야 한다.

```
char * family;  
family = "hong";
```



문자형 변수

■ char 배열과 char 포인터의 사용 사례

- 초깃값을 갖는 문자열로 사용할 때
 - 외부 라이브러리에서 C++ 문자열을 요구하기도 한다. 이때는 아래와 같이 char 배열로 지정한다. 배열의 크기는 지정하지 않아도 컴파일러가 처리한다.

```
char ssid[] = "mySsid";    // ssid[7] = "mySsid"  
char pass[] = "myStrongPassword"; // pass[17] = "myStrongPassword"
```

- char 포인터를 사용해도 결과는 같다.

```
char * ssid = "mySsid";  
char * pass = "myStrongPassword";
```

- 값을 고정하기 위한 상수로 char 배열이나 char 포인터를 사용할 경우 맨 앞에 const를 붙이는 것이 좋다.

문자형 변수

■ char 배열과 char 포인터의 사용 사례

- 다른 함수에서 사용할 버퍼를 정의할 때
 - 뒤에 나오는 String 오브젝트에서는 아래와 같이 char 배열의 크기를 정한다.

```
char buf[16];  
String nmStr = "Gildong";  
nmStr.toCharArray(buf, 16);
```

- char buf[16]의 형식으로 16개 요소의 배열을 선언한다.
toCharArray()는 뒤에서 설명한다.

String 클래스

- 초보자가 C++로 문자열을 다루기는 힘들다. 아두이노에서는 쉽게 문자열을 다루기 위해 String 클래스를 만들었다.
- String 클래스는 문자열을 다루는 프로그램 묶음이다.

String 클래스

■ String 오브젝트 만들기

- String 클래스는 필요할 때마다 오브젝트를 만들어서 사용하면 된다. 오브젝트를 만들 때 초깃값을 지정할 수 있다.

```
String 오브젝트1;  
String 오브젝트2 = "초깃값";
```

- 예시

```
String fullName;  
String name = "Gildong";  
String family = "Hong";  
  
fullName = family + " " + name; // Hong Gildong  
fullName = String("Hong") + " " + "Gildong"; // hong gildong  
fullName = "Hong" + " " + "Gildong"; // 오류
```

String 클래스

■ String 오브젝트 만들기

- String 오브젝트에 하나의 문자열은 직접 할당할 수 있다.
- String 오브젝트는 **몇 개든 +로 연결**할 수 있으며. 연결할 대상 중 적어도 하나는 **String 오브젝트 속성**을 가져야 한다.
- **String은 클래스고 fullName, name, family는 오브젝트**이며 오브젝트를 다루는 다양한 함수(메소드)를 사용할 수 있다.

String 클래스

■ 다른 변수형을 스트링으로 변환하기

- 상수나 변수를 String으로 감싸도 스트링으로 변환된다.

```
String(문자 또는 문자열)
String(숫자)
String(변수)
String(float형 변수 또는 상수, 소수점 이하 자릿수)
```

- [프로그램3-4]처럼 다양한 변수형을 스트링으로 바꿔보자.

프로그램 3-4 String-concat

```
001 void setup() {
002   Serial.begin(115200);
003   Serial.println();
004
005   int h = 9;
006   String hStr = String(h); // "9"
007
008   int m = 8;
009   String mStr = String(m); // "8"
010   if (mStr.length() == 1) {
011     mStr = '0' + mStr; // "08"
012   }
```

① 8-12행:스트링에 .length()를 붙이면 길이를 알려준다.

길이 = 스트링.length();

String 클래스

■ 다른 변수형을 스트링으로 변환하기

```
013  
014   int s = 35;  
015   String sStr = String(s); // "35"  
016  
017   float t = 27.56;  
018   String tStr = String(t,1); // "27.6" (반올림)  
019  
020   String longStr = hStr + ":" + mStr + ":" + sStr  
021                   + ", " + tStr;  
022   Serial.println(longStr); // "9:08:35,27.6"  
023 }  
024  
025 void loop() {  
026  
027 }
```

실행 결과

9:08:35,27.6

② 17-18행:String()의 두 번째 인수로 소수점 이하 자릿수를 지정할 수 있다. 생략하면 소수점 2자리로 반올림한다.

스트링 = String(값, 소수점 이하 자릿수)

String 클래스

■ 스트링에서 값 골라내기

– 다음은 스트링에서 요소들을 골라내는 프로그램이다.

프로그램 3-5 String-substring-val

```
001 void setup() {  
002     Serial.begin(115200); // IDE 시리얼 모니터 맨 아래에서 속도 선택  
003     Serial.println();    // 이상한 글자 회피하기 위해 자동 줄바꿈  
004  
005     String longStr = "9:08:35,27.6"; // "hh:mm:ss,nn.n"  
006  
007     int p1 = longStr.indexOf(":");    // 처음부터  
008     int p2 = longStr.indexOf(":", p1+1); // p1+1부터  
009     int p3 = longStr.indexOf(",");    // 처음부터
```

① 5-9행:스트링에서 인수1의 위치를 알려준다.
인수2는 실행 위치로 생각하면 처음부터 실행한다.
인수1이 문자열의 첫 번째 글자면 0, 아니면 -1을 반환한다.

위치 = 스트링.indexOf(찾을 문자열); //
처음부터 찾을
위치 = 스트링.indexOf(찾을 문자열, 찾
기 시작하는 위치);

String 클래스

■ 스트링에서 값 골라내기

- 다음은 스트링에서 요소들을 골라내는 프로그램이다.

```
011 String hStr = longStr.substring(0,p1);    // p1 앞 글자까지
012 String mStr = longStr.substring(p1+1,p2); // p2 앞 글자까지
013 String sStr = longStr.substring(p2+1,p3); // p3 앞 글자까지
014 String tStr = longStr.substring(p3+1);    // p3+1부터 끝까지
015
```

② 11-14행:substring()은 정해진 범위의 문자열을 잘라 돌려준다.
서 인수2 앞 글자까지 잘라낸다.인수2가 생략되면 마지막까지 자른다.

인수1에

잘라낸 스트링 = 스트링.substring(시작 위치) // 끝까지 자름
잘라낸 스트링 = 스트링.substring(시작 위치, 이 숫자 앞까지 자름)

String 클래스

■ 스트링에서 값 골라내기

– 다음은 스트링에서 요소들을 골라내는 프로그램이다.

```
016   int h = hStr.toInt(); // 정수형으로 변환
017   int m = mStr.toInt(); // 정수형으로 변환
018   int s = sStr.toInt(); // 정수형으로 변환
019   float t = tStr.toFloat(); // float형으로 변환
020
021   Serial.println(h);
022   Serial.println(m);
023   Serial.println(s);
024   Serial.println(t);
025 }
026
027 void loop() {
028
029 }
```

③ 16-19행:toInt()와 toFloat()는 스트링을 int형과 float형의 값으로 반환, 온전한 숫자가 아니면 0을 반환한다.

int로 선언된 변수 = 스트링.toInt();
float로 선언된 변수 = 스트링.toFloat();

문자형 변수

■ 스트링을 문자열로 바꾸기

- 스트링 오브젝트의 메소드로 대부분의 문자열을 처리할 수 있으나 아래와 같이 결과를 문자열로 바꿔야 하는 경우가 생긴다.

프로그램 3-6 String-to-char-array

```
001 void setup() {  
002     Serial.begin(115200);  
003     Serial.println();  
004  
005     String nameStr = "gildong";  
006     char buf1[10];  
007     char buf2[10];  
008     char buf3[10];  
009  
010     nameStr.toCharArray(buf1,sizeof(buf1)); // sizeof(buf1): 10  
011     strcpy(buf2,buf1);  
012     strcpy(buf3,nameStr.c_str());  
013  
014     Serial.println(buf1);  
015     Serial.println(buf2);  
016     Serial.println(buf3);  
017 }  
018  
019 void loop() {  
020  
021 }
```

❶ 10행:메소드 toCharArray()는 스트링 nameStr을 문자열로 바꿔 char 배열인 buf1에 넣는다. 인수2는 인수1 buf1의 크기이다. sizeof(buf1)는 buf1의 길이를 나타낸다.

스트링.toCharArray
(바뀐 문자열이 들어갈
char 배열,
char 배열의 크기);

문자형 변수

■ 스트링을 문자열로 바꾸기

② 11-12행: 함수 `strcpy()`는 인수2인 **char 배열**을 인수1인 **char 배열**로 복사한다. 메소드 `c_str()`는 스트링을 (null로 끝나는) **char 배열** 형식으로 바꿔 돌려준다. `c_str()`은 스트링을 변환하여 바로 사용한다.

```
strcpy(복사 받을 char 배열, 복사할 char 배열);  
strcpy(복사 받을 char 배열, 복사할 스트링.c_str());
```

문자형 변수

■ 스트링을 활용한 추가 함수

```
016  
017     int h = hStr.toInt(); // 정수형으로 변환  
018     int m = mStr.toInt(); // 정수형으로 변환  
019     int s = sStr.toInt(); // 정수형으로 변환  
020     float t = tStr.toFloat(); // float형으로 변환  
021  
022     Serial.println(h);  
023     Serial.println(m);  
024     Serial.println(s);  
025     Serial.println(t);  
026 }  
027  
028 void loop() {  
029  
030 }
```

실행 결과

```
9  
8  
35  
27.60
```

■ 참과 거짓

- 참인 경우만 실행하는 조건문

– 이 장에서 배울 if문은 가장 기초적인 명령어다.

```
if (조건식) {  
    // 참일 때 수행할 명령어가 옴  
    // 명령어는 {}으로 묶음  
}
```

– If문 else문을 비롯한 선택문에 대해서는 2절에서 다룬다.

■ 참과 거짓

- 참과 거짓으로 나누어서 실행하는 조건문

– 조건식이 거짓인 경우에도 실행하려면 else문을 추가한다.

```
if (조건식) {  
    // 참일 때 수행할 명령어가 옴  
    // 명령어는 {}으로 묶음  
}  
else {  
    // 거짓일 때 수행할 명령어가 옴  
    // 명령어는 {}으로 묶음  
}
```


■ 비교연산자

- 조건식은 비교연산자로 표현되는 경우가 많다.

비교연산자	의미
$a == b$	a와 b가 같다
$a != b$	a와 b가 같지 않다
$a > b$	a가 b보다 크다
$a >= b$	a가 b보다 크거나 같다
$a < b$	a가 b보다 작다
$a <= b$	a가 b보다 작거나 같다

■ 논리연산자

- 논리 연산자는 참 또는 거짓인 두 값을 이용하여 새로운 참과 거짓을 찾는 연산자다.

논리연산자	의미	설명
!a	논리 부정(NOT)	참과 거짓을 서로 바꿈
a && b	논리적 AND	둘 다 참일 때 참
a b	논리적 OR	둘 중 하나가 참이면 참

- &과|를 두 개씩 겹쳐서 사용한다는 점을 주의한다. 한 개만 사용하면 비트연산자로 쓰여 다른 결과가 도출된다.
- 프로그램[4-1]을 올리고 결과를 확인한다.

■ 중첩된 조건문

- 선택이나 반복을 위한 논리식 안에 새로운 논리식을 중첩해 사용한다.

```
if (조건식) {  
    // 참일 때 수행할 명령어 그룹  
    if (조건식) {  
        // 참일 때 수행할 명령어 그룹  
    }  
    else {  
        // 거짓일 때 수행할 명령어 그룹  
    }  
}  
else {  
    // 거짓일 때 수행할 명령어 그룹  
    if (조건식) {  
        // 참일 때 수행할 명령어 그룹  
    }  
    else {  
        // 거짓일 때 수행할 명령어 그룹  
    }  
}
```

■ if문

- if와 else문을 통해 다양한 상황을 처리할 수 있다. 하나의 변수값을 기준으로 삼을 때는 switch문을 활용할 수 있다.

```
if (조건식) {  
    // 참일 때 수행할 명령어 그룹  
}
```

- 조건식이 참일 때에만 {} 안의 프로그램을 수행한다. 조건에 맞는 명령어가 하나이면 {}을 지정하지 않아도 되나, 오류를 피하기 위해 항상 {}를 사용하는 경우가 많다.

■ if문

- 다음은 시리얼 모니터의 입력을 확인하는 조건문이다.

```
if (Serial.available()) {  
    char c = Serial.read();  
    Serial.print(c);  
}
```

- Serial.available()은 버퍼에 도착한 문자 수를 정숫값으로 알려준다.
- Serial.read()는 시리얼 버퍼에서 한 글자를 읽어서 돌려준다.

- 입력 버퍼에 문자 5개가 도착하면 조건식은 참이 되어 {} 안의 명령어를 수행한다. **메소드 Serial.available()은 버퍼에 문자가 도착해 있으면 참이 된다.**

■ if else문

```
if (조건식) {  
    // 참일 때 수행할 명령어 그룹  
}  
else {  
    // 거짓일 때 수행할 명령어 그룹  
}
```

- 조건식의 참 거짓을 구분하여 {} 안의 프로그램을 수행한다.

■ if else if문

```
if (조건식) {  
    // 조건식이 참일 때 수행할 명령어 그룹  
}  
else if (조건식n) {  
    // 조건식n이 참일 때 수행할 명령어 그룹  
}
```

- 여러 개의 조건식을 차례로 따져 {} 안의 프로그램을 수행한다.

■ switch문

```
switch(정수형 변수) {  
    case 값1 :  
        // 변수가 값1과 같을 경우 수행할 명령어 그룹  
        break;  
    case 값n :  
        // 변수가 값n과 같을 경우 수행할 명령어 그룹  
        break;  
    default:  
        // 변수와 일치하는 값이 위에서는 없을 경우 수행할 명령어 그룹  
}
```

- 하나의 변수값을 비교하여 여러 값으로 나누어 수행할 때 사용한다.
- case문은 여러 개 사용할 수 있으며 break문으로 끝낸다. break문이 없는 경우 문장을 따라 계속 수행하므로 주의한다.
- switch문의 ()안에는 정수형 변수를 지정한다.

■ 삼항 연산자(? :)

- 여러 줄의 조건식을 한 줄로 표현할 때 사용한다.

```
변수 = 조건식 ? 참일 때 값 : 거짓일 때 값 ;
```

- 앞 문장은 아래와 동일하다.

```
if (조건식)  
    변수 = 참일 때 값;  
else  
    변수 = 거짓일 때 값;
```


■ 삼항 연산자(? :)

- 아래 예시의 value 값은 얼마인가?

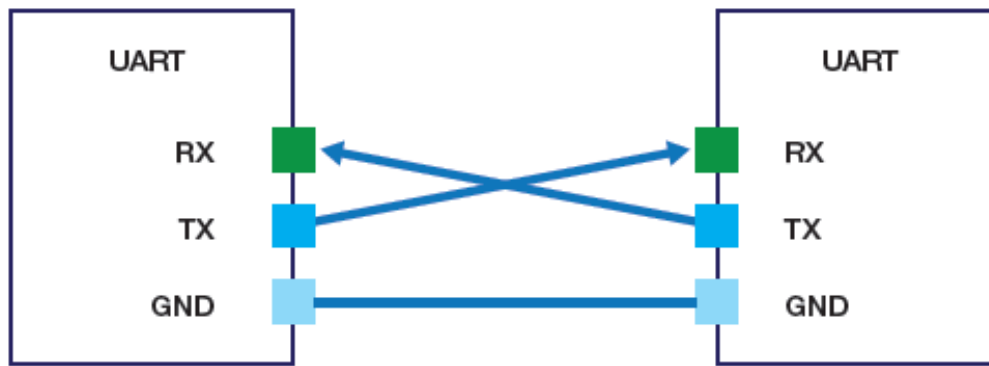
```
boolean status = true;  
int value;  
value = status ? LOW : HIGH;  
digitalWrite(LED_BUILTIN,value);
```

- status는 참이므로 LOW가 value에 전달되며 아래 프로그램과 동일한 의미를 지닌다.

```
boolean status = true;  
int value;  
if (status)  
    value = LOW;  
else  
    value = HIGH;  
digitalWrite(LED_BUILTIN,value);
```

■ 시리얼 통신이란?

정보처리기기는 기본으로 시리얼 통신을 지원한다. 두 기기 간에 통신 속도를 약속하고 송수신 회선을 각각 지정하여 데이터를 주고받는다.

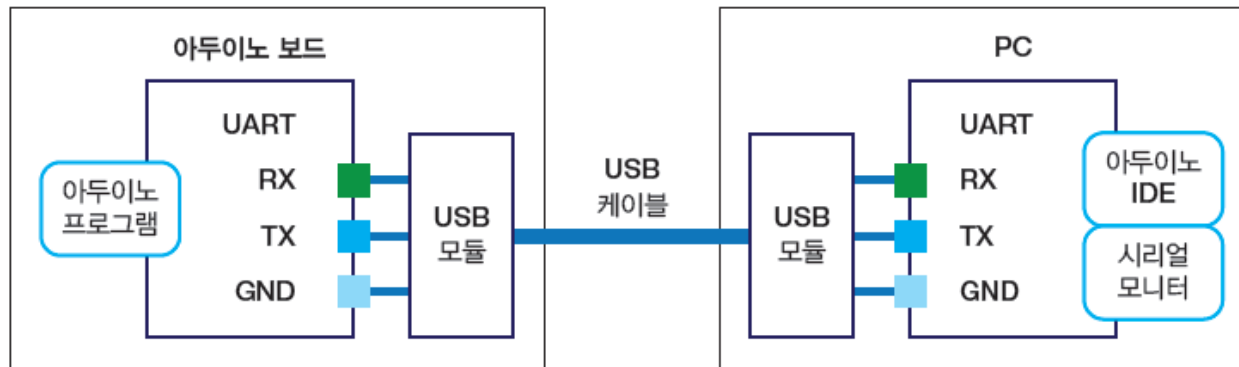


[그림 5-1] 시리얼 통신 개념도

- 보드에는 시리얼 통신을 지원하는 하드웨어인 UART(Universal Asynchronous Receiver/Transmitter)가 장착되어 있다.
- UART에는 TX(transmission)와 RX(reception)이 존재한다. TX는 다른 기기의 RX와 연결하고 RX는 다른 기기의 TX와 연결한다. GND는 서로 연결한다.

■ 아두이노 보드와 PC 간의 시리얼 통신

- 보드와 PC 모두 USB 모듈이 있으므로 USB 케이블로 연결한다.
- USB 모듈의 도움을 받지만 두 기기는 시리얼로 통신한다.



[그림 5-2] 아두이노 보드와 PC 연결 개념도

시리얼 모니터로 출력하기

- 보드에서 Serial 오브젝트를 이용하여 데이터를 전송하면 IDE의 시리얼 모니터는 수신된 내용을 화면에 표시한다.

[표 5-1] Serial의 출력 메소드

메소드	의미
<code>begin(송수신 비트 수)</code>	초기화하면서 초당 송수신 비트 수를 지정함 주로 115200을 지정하면 됨
<code>print(출력할 값)</code>	인수를 눈에 보이는 글자로 출력함
<code>println(출력할 값)</code>	인수를 눈에 보이는 글자로 출력하고 줄을 바꿈
<code>write(숫자 값)</code>	메모리상의 내용을 1바이트로 출력함
<code>printf(출력 포맷, 값 리스트)</code>	지정한 포맷에 맞춰서 여러 개의 값을 눈에 보이는 글자로 출력함

시리얼 모니터로 출력하기

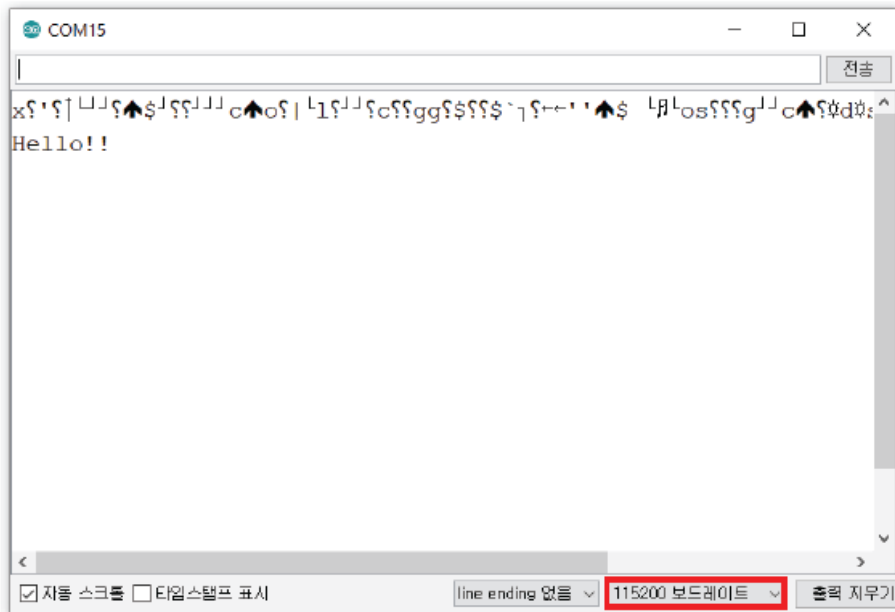
■ begin()

```
Serial.begin(통신 속도);
```

- Serial 오브젝트 사용을 위한 보드 레이트(baud rate) 지정에는 begin() 을 사용한다.

프로그램 5-1 serial-hello

```
001 void setup() {  
002   Serial.begin(115200); // IDE 시리얼 모니터 맨 아래에서 속도 선택  
003   Serial.println(); //쓰레기 글자를  
004   Serial.println("Hello!!");  
005 }  
006  
007 void loop() {  
008  
009 }
```



시리얼 모니터로 출력하기

■ print()

```
Serial.print(val);  
Serial.print(val,format);  
Serial.println(val);  
Serial.println(val, format);
```

- Serial.print()는 값을 문자, 문자열 또는 10진수 숫자로 출력한다.
- format으로 DEC, BIN, OCT, HEX를 지정하면 val의 값을 각각 10진수, 2진수, 8진수, 16진수로 출력한다.

시리얼 모니터로 출력하기

■ write()

```
Serial.write(byte형 val);  
Serial.write(문자|문자열);
```

- 다음은 print()와 write()의 출력 형식을 비교하는 프로그램이다.

프로그램 5-3 serial-write

```
001 void setup() {  
002   Serial.begin(115200);  
003   Serial.println();  
004  
005   Serial.println("-----문자");  
006   char c = 'a';  
007   Serial.println(c); // a  
008   Serial.write(c);   // a  
009  
010   Serial.println("\n-----문자열");  
011   char s[] = "abc";  
012   Serial.println(s); // abc  
013   Serial.write(s);   // abc  
014  
015   Serial.println("\n-----byte");  
016   byte b = 97; // 아스키 값 97 : 'a'
```

시리얼 모니터로 출력하기

■ write()

```
017 Serial.println(b); // 97
018 Serial.write(b);   // a
019 Serial.println();
020 }
021
022 void loop() {
023
024 }
```

실행 결과

```
-----문자
a
a
-----문자열
abc
abc
-----byte
97
a
```

- 문자, 문자열의 경우 결과가 같으나 숫자로 출력할 경우는 다르다.
- Serial.write()는 메모리 내용을 그대로 보낸다. 숫자 97은 아스키 코드로 'a'를 뜻한다.
- Serial.write()를 사용하여 숫자를 전송하면 데이터양이 줄어들지만 다루기 불편하다. 필요할 때만 사용한다.

시리얼 모니터로 출력하기

■ printf()

```
Serial.printf(format, 값 리스트 ...);
```

- Serial.printf()의 첫 인수는 출력을 위한 포맷으로 문자열로 표시한다. %로 시작하는 형식은 다음에 오는 값 리스트(인자)와 하나씩 대응된다. 이 방식은 Serial.print() 보다 직관적이다.

[표 5-2] 주요 포맷 형식

형식	출력 내용
%c	ASCII 문자
%d	부호 있는(signed) 정수
%nd	최소 n자리 정수, 자리가 남으면 빈칸(n은 숫자)
%0nd	최소 n자리 정수, 자리가 남으면 0으로 채움(n은 숫자)
%u	부호 없는(unsigned) 정수
%x	16진수로 표시
%0nx	최소 n자리 16진수, 자리가 남으면 0으로 채움(n은 숫자)
%f	부동 소수점수, 소수점 이하 6자리(2자리 아님)
%.nf	부동 소수점수, 반올림하여 소수점 이하 n자리(n은 숫자)
%s	문자열(인자로 문자열을 지정하며 스트링을 인자로 사용하려면 스트링.c_str()로 지정해야 함)

시리얼 모니터에서 입력 받기

- 아래는 이번 챕터에서 살펴볼 Serial과 관련된 메소드이다.

[표 5-3] Serial의 입력 메소드

메소드	의미
<code>begin(송수신 비트 수)</code>	초기화하면서 초당 송수신 비트 수를 지정
<code>available()</code>	입력 버퍼에 도착한 문자의 개수를 반환
<code>read()</code>	입력 버퍼에서 한 글자를 읽어서 반환
<code>parseInt()</code>	입력 버퍼에서 정수가 되는 부분까지 읽은 후 정수로 바꿔서 반환
<code>parseFloat()</code>	입력 버퍼에서 소수점 있는 수가 되는 부분까지 읽은 후 float로 바꿔서 반환
<code>readStringUntil(끝 표시 문자)</code>	끝 문자로 표시된 문자가 나오기 전까지 읽어서 지정한 배열에 삽입

시리얼 모니터에서 입력 받기

■ 입력한 내용을 그대로 출력하기

- 입력된 문자를 처리하는 방법

– 다음은 입력된 문자를 처리하는 전형적인 루틴이다.

```
while(Serial.available()) {  
    char c = Serial.read();  
    // 후속 처리  
}
```

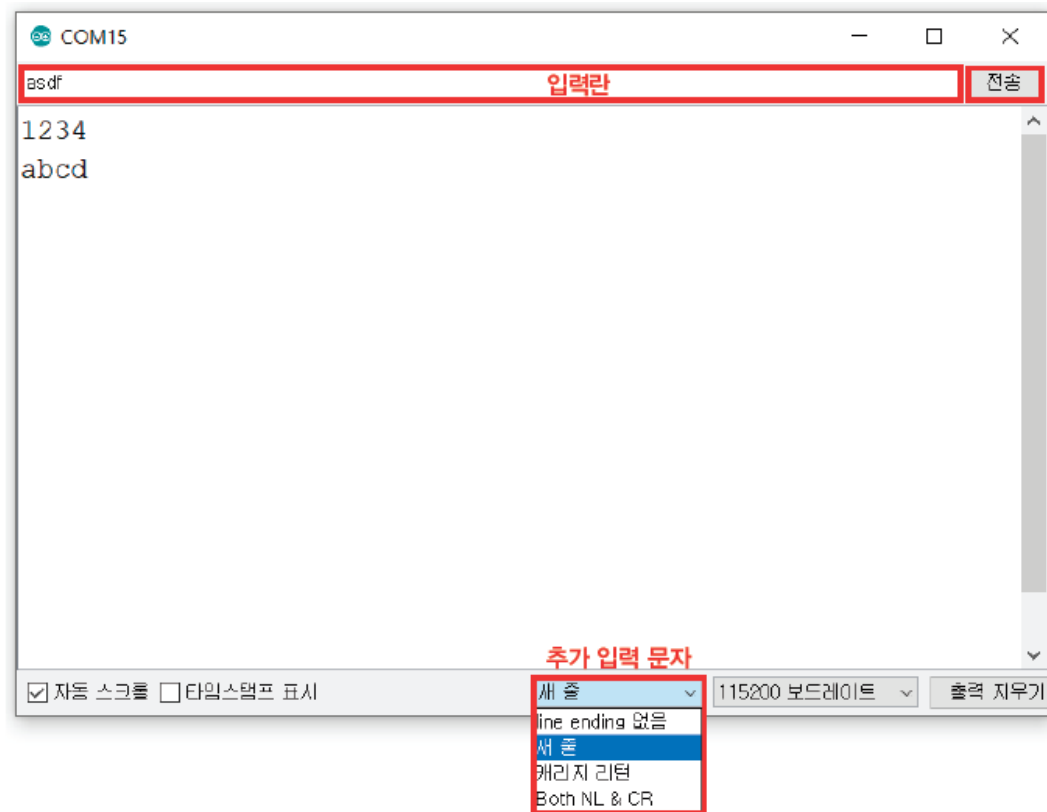
- 모니터에 데이터를 입력하면 보드의 수신 버퍼에 도착한다. 프로그램은 Serial.available()을 통해 문자수를 확인한다.
- 문자수가 0이 아니면 Serial.read()를 호출하여 하나씩 읽어온다.
- Serial.available()이 알려주는 숫자는 1씩 줄어드는데 0이 되면 while()을 벗어난다.

시리얼 모니터에서 입력 받기

■ 입력한 내용을 그대로 출력하기

- 추가 입력 문자 선택하기

[프로그램 5-5]를 실행한 후 시리얼 모니터를 열어 '추가 입력 문자'를 '새 줄'로 바꿉니다.



[그림 5-4] 시리얼 모니터에서 추가 입력 문자 선택하기

시리얼 모니터에서 입력 받기

■ 입력한 내용을 그대로 출력하기

입력란에 '1234'를 입력한 후 [전송] 버튼을 누르고, 'abcd'를 입력한 후에도 [전송] 버튼을 누릅니다.

실행 결과

1234

abcd

'1234'를 입력한 후 [전송] 버튼을 누르면 '1234'와 함께 '새 줄'을 나타내는 문자(아스키 코드 10)가 연달아 전송됩니다. 그래서 'abcd'는 다음 줄에 출력됩니다.

[표 5-4] 추가 입력 문자

추가 입력 문자	추가로 입력되는 문자
line ending 없음	추가 입력 문자 없음
새 줄	"\n" : New Line, Line Feed, 아스키 코드10
캐리지 리턴	"\r" : Carriage Return, 아스키 코드13
Both NL & CR	"\r\n" : Carriage Return + New Line

시리얼 모니터에서 입력 받기

■ int, float, 문자열 입력 받기

- key와 val
 - 시리얼 통신을 위해서는 데이터 형식을 먼저 정해야 한다.
 - 데이터 이름(key)과 값(val)을 한 쌍으로 주고받으면 편하다.
 - c라는 key와 123이라는 val은 'c123' 또는 'c 123'으로 입력할 수 있다.

[표 5-5] key, val 데이터 형식 설계

데이터	key	val의 데이터 형식	val의 변수 이름	입력 예
개수	c	int	cnt	c125 또는 c 125
온도	t	float	temp	t26,6 또는 t 26,6
이름	n	String, 문자 끝에 ;	name	ngildong;

시리얼 모니터에서 입력 받기

■ 시리얼 모니터에서 LED 제어하기

- 삼항 연산자

받은 값 = 조건식 ? 참일 때의 값 : 거짓일 때의 값;

- 삼항 연산자는 ?와 :이다.
- 여러 줄에 걸친 코드를 줄일 수 있으나 난해하지 않도록 주의한다.
- 뒤에서 배울 Node-RED의 스크립트에 반드시 써야하는 경우가 있다.

```
ledVal = val ? LED_ON : LED_OFF;
```

- 위 코드를 다음과 같이 바꿀 수 있다.

```
if (val == 1) {  
    ledVal = LED_ON;  
}  
else {  
    ledVal = LED_OFF;  
}
```

■ While문

```
while (조건식) {  
    // 참일 때 수행할 명령어 그룹  
}
```

- 조건식이 참일 때 {} 안의 프로그램을 수행, 다시 조건식을 테스트해서 **참일 경우 다시 수행**하는 과정을 반복한다.
- break문을 통해서 빠져나온다.
- continue문으로 나머지 코드를 무시하고 반복할 수 있다.

■ do while문

```
do {  
    // 이 명령어 그룹은 먼저 한 번 수행되고,  
    // 조건식이 참이면 계속 수행됨  
} while(조건식) ;
```

- 조건식을 마지막 while문에서 테스트한다. 따라서 {}안의 **프로그램은 최소 한 번 실행된다.**
- 코드 진행 중 벗어나고 싶을 때는 break문을 사용한다.
- 중간 코드를 무시하고 반복하고 싶으면 continue문을 사용한다.

■ for문

```
for ( 변수 선언문 ; 조건식; 매회 마지막 명령어 ) {  
    // 명령어 그룹  
}
```

- 처음 선언문을 실행 후 조건식이 참이면 명령어 그룹을 실행한다. 매회 마지막 명령어를 실행하고 조건식 테스트를 반복한다.
- 변수 선언문은 비워두고 ;으로 조건식과 구분할 수 있다.

■ for문

```
char in[11] = "input data";
```

```
char out[11];
```

```
int size = 11;
```

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

```
    out[i] = in[i];
```

```
}
```

– 변수 i는 for문의 {} 안에서만 유효한 지역 변수이다.

주기적으로 if 블록 실행하기

■ 1초마다 if 블록 실행하기

- setup()과 loop() 함수는 아두이노의 기본이다. setup()은 단 한번 수행되고, loop()는 반복적으로 수행된다.
- [프로그램 6-1]은 1초 동안의 loop의 실행 횟수를 세어서 출력한다.

프로그램 6-1 loop-count

```
001 void setup() {  
002     Serial.begin(115200);  
003     Serial.println();  
004 }  
005  
006 void loop() {  
007     static unsigned long loopCnt = 0;  
008     static unsigned long nextMil = millis() + 1000;
```

❶ 7~8행 : 지역 변수 **loopCnt**, **nextMil**을 선언했다. 지역 변수는 다음 실행 때 값을 잃어버리므로 값 유지를 위해 **static**을 사용한다.

millis()는 가동 시간을 밀리초 단위로 알려 주는 아두이노 내장 함수다.

nextMil은 1초 후를 나타내는 변수이므로 **millis()+1000**으로 초기화 한다.

주기적으로 if 블록 실행하기

■ 1초마다 if 블록 실행하기

```
009  
010  loopCnt++; // loopCnt = loopCnt + 1;  
011  
012  if (millis() > nextMil) {  
013      nextMil = millis() + 1000;  
014      Serial.println(loopCnt);  
015      loopCnt = 0;  
016  }  
017 }
```

실행 결과

```
...  
94557  
94653  
94657  
94671  
...
```

② 10행 : `loop()`가 실행될 때마다 `loopCnt`를 1씩 증가시킨다.

③ 12~16행 : 1초마다 if 블록을 수행한다.

조건식 (`millis() > nextMil`)이 참이 되면 `nextMil`은 1초 이후로 정해진다. 그래서 1초 후에 다시 조건식이 참이 된다. 실제로 약 94000회마다 참이 되고, 나머지 실행 시간은 거짓이므로 블록은 실행되지 않는다.

주기적으로 if 블록 실행하기

■ 주기적으로 if 블록 실행하기

- 다음은 블록을 주기적으로 실행하는 전형적인 코드이다.
- 변수 nextMil에 붙은 static에 유의한다.
- 아두이노 프로그램에서 자주 사용된다.

```
static unsigned long nextMil = millis() + 실행 간격;  
if (millis() > nextMil) {  
  nextMil = millis() + 실행 간격;  
  // 실행할 코드  
}
```

- 함수 내에서 변수 선언 시 꼭 첫 부분에 둘 필요는 없다.
- 변수는 사용 전에 선언하면 되며 선언 위치와 사용 위치는 가까울수록 읽기가 좋다.

LED 제어하기

- 아두이노 보드에는 여러 센서와 디스플레이 장치가 붙어있다.
- 프로그램 작성 시 장치들에 골고루 시간을 할당해야 한다.
- 특정 기능 수행에 로직을 묶어 놓아서 다른 일을 못하게 되면 프로그램이 **블로킹(Blocking)**되었다고 한다.
- 반대로 여러 기능을 수행하는 것은 **논블로킹(non-blocking)** 방식이며 아두이노에서도 논블로킹 방식으로 프로그래밍해야 한다.

- delay() 소프트웨어적으로 구현된 Busy-wait 방식 블로킹 함수

```
void delay(unsigned long ms) {  
    unsigned long start = millis();  
    while (millis() - start < ms) {  
        // 아무 것도 안 하고 그냥 기다림  
    }  
}
```

- millis() 타이머0이라는 하드웨어 타이머를 이용해서 시간 측정
- 하지만 delay() 자체는 CPU를 멈추고 루프 돌리는 소프트웨어적인 방식

타이머 사용하기

- 인터럽트(Interrupt)는 외부의 자극이나 시간에 의해 마이크로컨트롤러에 긴급히 전달되는 신호이다.
- 타이머는 마이크로컨트롤러에 포함된 하드웨어 장치로 보드의 종류에 따라 프로그램 작성 방법이 다르다..

타이머 사용하기

■ Simple Timer(아두이노 우노, ESP 계열)

- SimpleTimer는 하드웨어 타이머나 인터럽트 대신 millis() 함수 기반으로 작성한 통상적인 클래스다. 마이크로컨트롤러 종류와 상관없이 사용할 수 있다.

여기서 잠깐 Github에서 라이브러리 설치하기

구글에서 'SimpleTimer.h'를 검색하여 <https://github.com/jfturcot/SimpleTimer>을 눌러봅시다. 이때 나타나는 화면이 Github에서 라이브러리를 다운로드할 수 있는 일반적인 화면입니다.

- 1 상단 우측의 연두색 버튼인 [clone or download]를 누릅니다.
- 2 [Download ZIP]을 누릅니다.
- 3 [다른 이름으로 저장] 화면이 나오면 저장할 폴더를 지정하고 [저장] 버튼을 누릅니다.
- 4 IDE에서 [스케치] - [라이브러리 포함하기] - [.zip파일 포함하기...]를 실행합니다.
- 5 앞에서 다운로드한 파일을 선택하고 열기를 누르면 라이브러리가 설치됩니다.

타이머 사용하기

프로그램 6-6 loop-count-SimpleTimer

```
001 #include <SimpleTimer.h>
002 SimpleTimer timerCnt;
```

❶ 1~2행 : 앞에서 설치한 라이브러리는 1행과 같이 추가한다. 2행은 SimpleTimer를 이용해 오브젝트 timerCnt를 생성했다.

```
003
004 unsigned long loopCnt;
005
006 void timerCntFunc() {
007     Serial.println(loopCnt);
008     loopCnt = 0;
009 }
```

```
010
011 void setup() {
012     Serial.begin(115200);
013     Serial.println();
014
```

```
015     timerCnt.setInterval(1000, timerCntFunc); // milli-sec, func
016 }
```

❷ 15행 : 앞에서 선언한 timerCnt에 setInterval() 메소드를 붙여 실행한다. 첫번째 인수로 실행 간격을 밀리초로 지정하고 두 번째 인수로 함수를 지정한다. 이때 함수는 일반 함수다.

```
017
018 void loop() {
019     timerCnt.run();
```

```
020     loopCnt++; // loopCnt = loopCnt + 1;
021 }
```

❸ 19행 : loop()안의 모든 오브젝트에 대해 오브젝트.run을 해두어야 한다. 메소드가 실행될 때 조건이 충족되면 지정한 함수가 실행된다.

출처 - 한빛아카데미(주) 따라 하면서 배우는 사물 인터넷

타이머 사용하기

실행 결과

73085

73779

72828

- loop()의 반복 시기가 일정하지 않을 수 있다. 따라서 SimpleTimer 오브젝트에서 지정한 함수의 수행 시간도 달라질 수 있다.
- SimpleTimer의 주된 용도는 1초 이상의 간격으로 주기적으로 센서 데이터를 취득하는 것이다.
- **LED 제어와 같이 정확한 시간이 중요하면 Ticker를 사용하고, loop()의 수행 속도에 맞춰 센서 데이터를 취득하는 것이 중요한 경우에는 SimpleTimer를 사용한다.**

함수 만들기

■ 함수의 구조

- 함수(function)은 특정 기능을 수행하는 프로그램의 묶음이다.
- 함수의 기본 구조는 아래와 같다.

```
전달받을_데이터형 함수 이름(매개 변수_리스트) {  
    // 명령어 그룹  
    return 전달할_데이터형의_값이나_변수;  
}
```

- '전달받을_데이터형' 은 돌려받을 값이 void로 지정한다.
- '매개 변수_리스트' 는 매개 변수(parameter)가 필요 없으면 비운다.

```
void 함수 이름() {  
    // 명령어 그룹  
}
```

함수 만들기

■ 함수를 호출하는 방법

- 함수 이름만으로 호출하는 방법
- sum() 함수를 호출하여 두 수의 합을 구하는 프로그램이다.

프로그램 7-1 sum-no-parameter

```
001 int a1 = 2;
002 int a2 = 3;
003 int a3;
004
005 void setup() {
006     Serial.begin(115200);
007     Serial.println();
008
009     sum();
010     Serial.println(a3);
011 }
```

```
012
013 void loop() {
014 }
015
016 void sum() {
017     a3 = a1 + a2;
018 }
```

실행 결과

5

함수 만들기

■ 함수를 호출하는 방법

- 인수의 값을 복사하여 매개 변수로 전달하는 방법
- 다음 프로그램은 sum() 함수를 호출할 때 인수(argument)로 값을 전달하고 값을 돌려받는다.

프로그램 7-2 sum

```
001 int a1 = 2;
002 int a2 = 3;
003 int a3;
004
005 void setup() {
006     Serial.begin(115200);
007     Serial.println();
008
009     //아래 a1,a2는 인수(argument)임
010     a3 = sum(a1,a2);
011     Serial.println(a3);
012 }
```

```
013
014 void loop() {
015 }
016
017 //아래 a,b는 매개 변수(parameter)임
018 int sum(int a, int b) {
019     int c = a + b;
020     return c;
021 }
```

실행 결과

5

■ 함수를 호출하는 방법

- 인수의 값을 복사하여 매개 변수로 전달하는 방법
 - 호출 시 **인수의 변수형과 함수의 매개 변수형**은 일치해야 한다.
 - 함수를 호출할 때 인수의 값을 복사해서 함수의 매개 변수로 전달하므로 **두 변수는 서로 다른 영역**을 가진다.
 - 따라서 함수에서 매개 변수의 값을 바꿔도 호출할 때 사용한 인수의 값을 변하지 않는다.

함수 만들기

■ 함수를 호출하는 방법

- 사용한 인수를 매개 변수에서 참조하여 사용하는 방법
 - 함수 호출 시 인수로 사용한 변수값을 함수 내에서 수정하려면 함수의 매개 변수 다음에 '&'을 붙인다.
 - '&'은 변수의 주소를 전달하는 역할을 한다. [프로그램7-3]은 sum() 함수 출력 시 인수로 지정한 변수값을 함수에서 바꿔 출력하는 경우를 보여준다.

프로그램 7-3 sum-ampersand

```
001 int a1 = 2;
002 int a2 = 3;
003 int a3;
004
005 void setup() {
006     Serial.begin(115200);
007     Serial.println();
008
009     //아래 a1,a2,a3는 인수(argument)임
010     sum(a1,a2,a3);
011     Serial.println(a3);
012 }
```

```
013
014 void loop() {
015 }
016
017 //아래 a,b,c는 매개 변수(parameter)임
018 void sum(int a, int b, int& c) {
019     c = a + b;
020 }
```

실행 결과

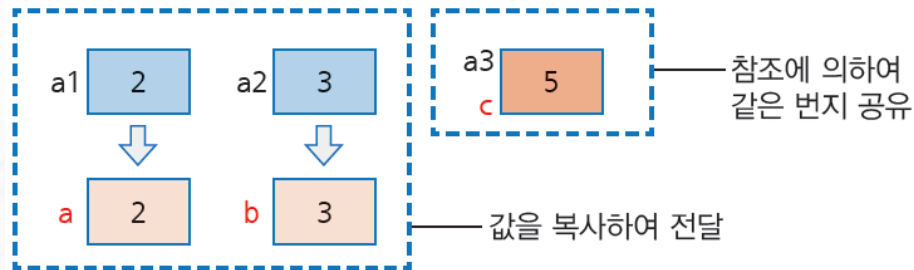
5

함수 만들기

■ 함수를 호출하는 방법

- 사용한 인수를 매개 변수에서 참조하여 사용하는 방법
- [프로그램7-3]에서 sum() 함수를 호출하면 a1과 a2를 더한 값이 a3에 온다.
- 18행에서 sum() 함수의 세 번째 매개 변수(parameter) c가 참조 형식으로 선언되었기 때문이다.

```
int a1 = 2;  
int a2 = 3;  
int a3;  
sum(a1,a2,a3);
```



```
void sum(int a, int b, int& c) {  
    c = a + b;  
}
```

[그림 7-1] 함수의 인수와 매개 변수

■ 함수 오버로딩

- 여러 함수가 유사 기능을 지니며 받는 변수형이나 매개 변수의 데이터형, 개수 등이 가른 경우 같은 이름을 사용하면 편하다.
- 같은 이름에 매개 변수형이나 개수를 다르게 갖는 것을 함수 오버로딩(function overloading)이라고 한다.
- [프로그램7-4]의 sum() 함수는 실제로는 다른 함수지만 같은 이름을 사용한다.

함수 만들기

프로그램 7-4 sum-overload

```
001 int a1 = 2;
002 int a2 = 3;
003 int a3;
004
005 float f1 = 2.5;
006 float f2 = 3.6;
007 float f3;
008
009 void setup() {
010     Serial.begin(115200);
011     Serial.println();
012
013     //아래 a1,a2는 인수(argument)임
014     a3 = sum(a1,a2);
015     Serial.println(a3);
```

```
016
017     //아래 f1,f2,f3은 인수(argument)임
018     sum(f1,f2,f3);
019     Serial.println(f3);
020 }
021
022 void loop() {
023 }
024
025 //아래 a,b는 매개 변수(parameter)임
026 int sum(int a, int b) {
027     int c = a + b;
028     return c;
029 }
030
031 //아래 a,b,c는 매개 변수(parameter)임
032 void sum(float a, float b, float& c) {
033     c = a + b;
034 }
```

실행 결과

5

6.10

■ 함수 프로토타입

- 함수는 돌려받을 변수형, 함수 이름, 매개 변수의 속성 정보나 개수를 미리 정해야 한다. 이런 함수의 요약 정보를 **함수 프로토타입 (function prototype)**이라고 한다.
- 함수 프로토타입이 선언되어 있으면 이후에도 편한 곳에서 함수를 사용하면 된다.
- 아두이노 IDE는 전처리 과정에서 프로토타입을 `setup()` 함수 앞 부분에 자동으로 넣어준다. 따라서 아두이노 IDE 환경에서는 프로토타입을 작성하지 않는다.
- C++의 기본 기능이 아니라 IDE의 부가 기능임을 주의한다.
- 함수 프로토타입을 꼭 작성해야 하는 경우
 - 오브젝트 생성 시 인수로 함수를 지정하면 `setup()`보다 앞이므로 반드시 함수 프로토타입을 선언한다.

클래스 만들기

- 클래스는 재사용할 수 있는 프로그램 묶음이다.
- 하나의 클래스는 공유할 수 있는 변수와 함수로 구성되어 있다.

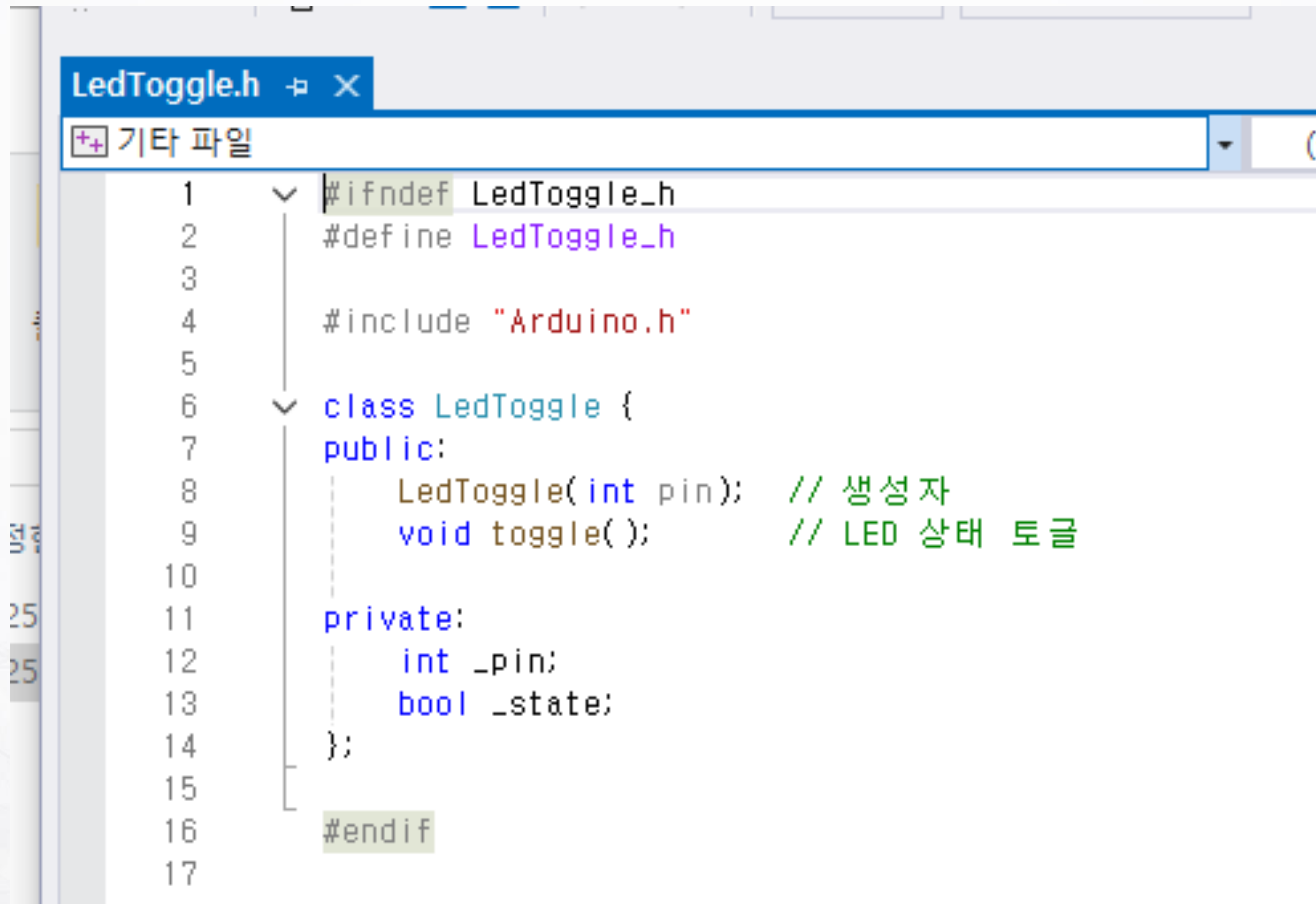
- LED Control 예제

myLED

```
void setup() {  
    pinMode(LED_BUILTIN, OUTPUT);  
}  
  
void loop() {  
    digitalWrite(LED_BUILTIN, HIGH);  
    delay(1000);  
    digitalWrite(LED_BUILTIN, LOW);  
    delay(1000);  
}
```

클래스 만들기

- Class 파일 작성법
- LedToggle.h



The screenshot shows a code editor window titled "LedToggle.h". The code is a C++ header file for a class named "LedToggle". It includes a preprocessor guard, a header guard, and an include for "Arduino.h". The class "LedToggle" has a public constructor "LedToggle(int pin)" and a public method "void toggle()". It also has private member variables "int _pin" and "bool _state". The code is as follows:

```
1  #ifndef LedToggle_h
2  #define LedToggle_h
3
4  #include "Arduino.h"
5
6  class LedToggle {
7  public:
8      LedToggle(int pin);    // 생성자
9      void toggle();        // LED 상태 토글
10
11  private:
12      int _pin;
13      bool _state;
14  };
15
16  #endif
17
```

아두이노 파일 구조

```
#ifndef LED_H
#define LED_H
// 원래 .h 파일 내용
#endif
```

- #ifndef는 #if not defined를 줄인 말이다. 처음 #include한 후부터는 #ifndef가 거짓이 되므로 #include는 동작하지 않는다.
- Led.h 파일의 중복 #include를 방지하기 위한 코드는 아래와 같다.

```
#ifndef LED_H
#define LED_H

#define LED_OFF 0
#define LED_ON 1
class Led {
};

Led::Led(int INpin, bool INonVal) {
}
#endif
```

클래스 만들기

class 명령어는 클래스를 정의한다.

```
class class_name {  
  
    public:  
        // constructor  
        class_name();  
        class_name(argument_list);  
  
        // 메소드 선언  
  
    private:  
  
        // 내부 사용 변수 선언  
  
}; // {}의 끝에 ;이 있음
```

- 클래스는 함수가 아니라 명령어이므로 끝에 세미콜론[;]을 붙인다.
- class_name은 첫문자를 대문자로 지칭한다.
오브젝트는 클래스와 구분하여 첫 글자를 소문자로 작성하면 좋다.

클래스 만들기

```
class class_name {  
  
    public:  
        // constructor  
        class_name();  
        class_name(argument_list);  
  
        // 메소드 선언  
  
    private:  
  
        // 내부 사용 변수 선언  
  
}; // {}의 끝에 ;이 있음
```

- {} 안은 public과 private를 붙여서 두 영역으로 나눈다.
- Public 영역은 클래스 외부에 알려져야 할 메소드의 프로토타입을 정의한다. private 영역에는 메소드 안에서 사용할 함수나 변수를 선언한다. 이 값은 외부에 알려지지 않는다.
- 클래스 이름과 같은 이름을 가진 특별한 함수를 생성자(constructor)라고 한다.

클래스 만들기

- cpp(소스) 파일 작성법
- LedToggle.cpp

LedToggle.cpp

기타 파일

```
1  #include "LedToggle.h"
2
3  LedToggle::LedToggle(int pin) {
4      _pin = pin;
5      _state = false;
6      pinMode(_pin, OUTPUT);
7      digitalWrite(_pin, LOW);
8  }
9
10 void LedToggle::toggle() {
11     _state = !_state;
12     digitalWrite(_pin, _state ? HIGH : LOW);
13 }
14
```

클래스 만들기

■ 클래스 작성하기

- 범위 지정 연산자(::) 사용하기

- class문 밖에서 클래스에 속한 메소드를 작성할 때는 이름 앞에 class_name::을 붙여 클래스의 소속을 표시한다.
- 범위 지정 연산자 (::) 사용을 제외하면 메소드 작성과 일반 함수 작성은 똑같다.

- 클래스의 장점

- 클래스를 별도 파일로 보관하면 재사용시 클래스를 선언하고 생성자, 메소드의 사용법만 알면 된다.

클래스 만들기

■ 클래스 작성하기

- 생성자

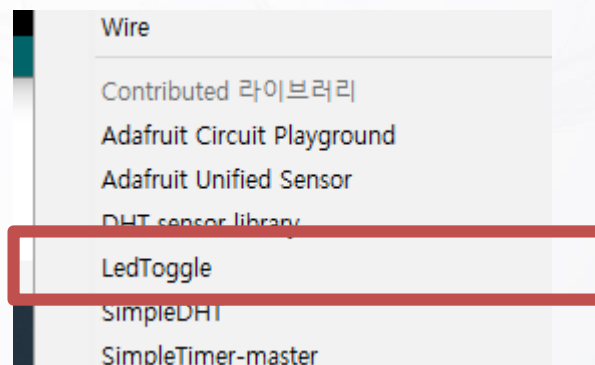
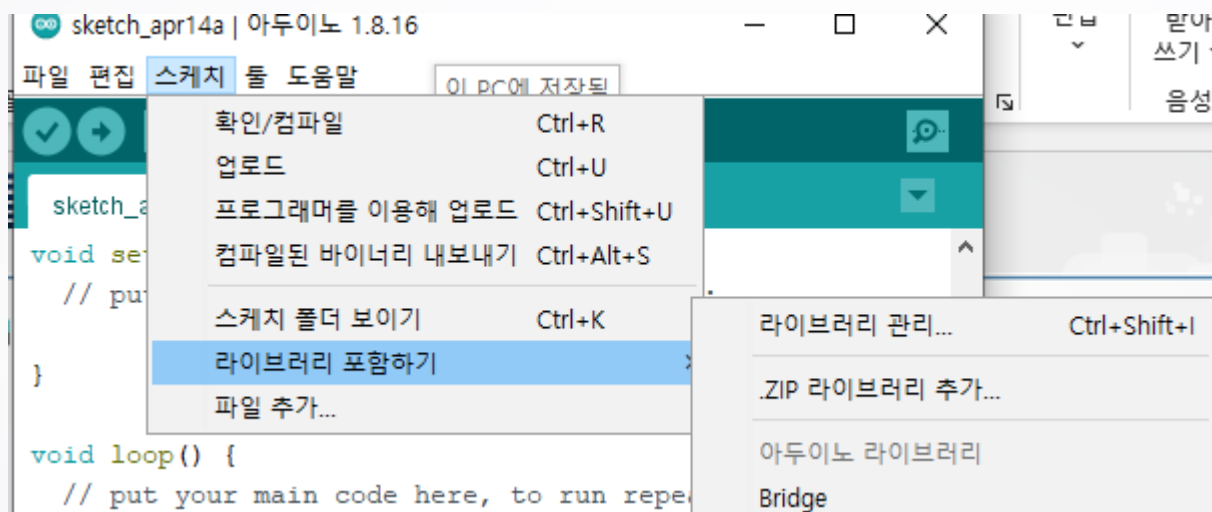
```
LedToggle led(LED_BUILTIN);
```

- 생성자 앞에는 void나 변수형이 붙지 않으므로 다른 메소드와 구분된다.
- 생성자는 주로 클래스 사용을 위해 오브젝트를 생성할 때 호출된다.
- 위와 같이 지정하면 생성자 `Led(int pin, bool onVal);`이 사용된다.
- 즉, 클래스 `Led`를 활용한 오브젝트 `led`가 생성된다.
- 관행적으로 메소드의 프로토타입과 변수만 `class`문 안에 선언하고 메소드의 코드는 `class`문이 끝난 다음에 작성한다.

클래스 만들기

■ 클래스 사용하기

- 클래스 포함하기



클래스 만들기

- 사용 예제 myled.ino 파일



```
myled | 아두이노 1.8.16
파일 편집 스케치 툴 도움말
[check] [run] [sketch] [upload] [download] 확인
myled $
#include <LedToggle.h>

LedToggle led(LED_BUILTIN);

void setup() {
}

void loop() {
    led.toggle();
    delay(500);
}
```

라이브러리 활용하기

■ 라이브러리 파일 구조

- 라이브러리가 Led.h와 Led.cpp 파일로 구성되어 있으면 다음 파일 구조를 가진다. 초록색 부분은 폴더를 의미한다.

```
Arduino>libraries
  Led
    examples
      led-sample1
        led-sample1.ino
    src
    Led.cpp
    Led.h
    keywords.txt
    library.json
    library.properties
    README.md
```

- Led.cpp와 Led.h 파일은 필수적이다. 라이브러리 폴더 이름은 기능을 잘 설명하도록 이름을 준다.
- library.properties에는 라이브러리 속성을 기록할 수 있다. 이 파일이 지정되어 있으면 라이브러리 폴더 아래에 폴더 src를 만들고 그 안에 .h 나 .cpp 파일을 둘 수도 있다.

웹 페이지 아두이노 라이브러리 정보 파일

keywords.txt, library.json, library.properties, README.md는 아두이노 라이브러리의 정보를 표시하는 파일입니다. 이러한 파일에 대한 설명은 다음 웹 페이지에서 제공합니다.

<https://iotmaker.kr/iotbook-arduino-lib-files>

**5주차 강의가 끝났습니다,
모두 고생하셨습니다.**

