# 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

C main() .
main() C . main()
. for while, if
.

( MCU) .
MCU

.
setup(), loop() . main() setup()
loop() .

main() setup(), loop() .
. .

```
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()

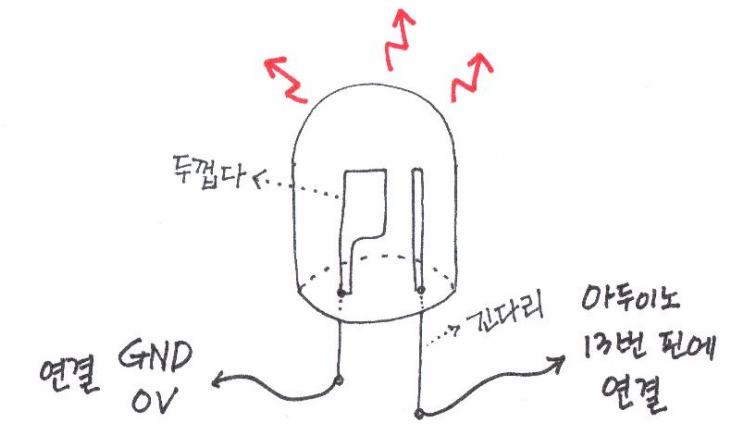
준비운동으로 13번 핀을 출력으로 만들었습니다. 그러면 13번 핀에 전기가 흐르게 합시다. 13번 핀에 꼬마전구(LED)를 달고 전기를 흐르게 해서 불이 들어오게 합시다. 이것은 loop() 에서 하면 됩니다.

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

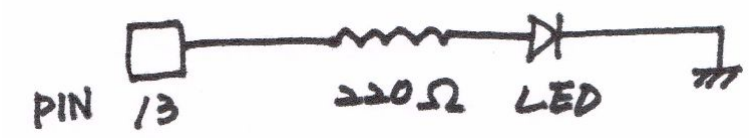
위에서 setup()을 통해 13번 핀을 출력으로 쓰겠다고 정했으니 loop() 에서는 그 13번 핀을 사용해서 전기를 흐르게 하든지 혹은 흐르지 않게 하든지 하면 됩니다. 아두이노의 13번 핀에 꼬마전구(LED)의 한쪽 발을 연결하고, 다른 한쪽 발은 0V에 연결합니다. 전기는 물과 같아서 높은 곳에서 낮은 곳으로 흐릅니다. 13번 핀에서 전기가 나와서 LED의 한쪽 발로 흘러들어가고, 그 전기가 다른 쪽 발로 나온 다음 0V 인 GND 로 흘러들어갑니다. 마치 산에 있는 물이 아래로 아래로 흘러 바다로 들어가듯이 전기는 이리저리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

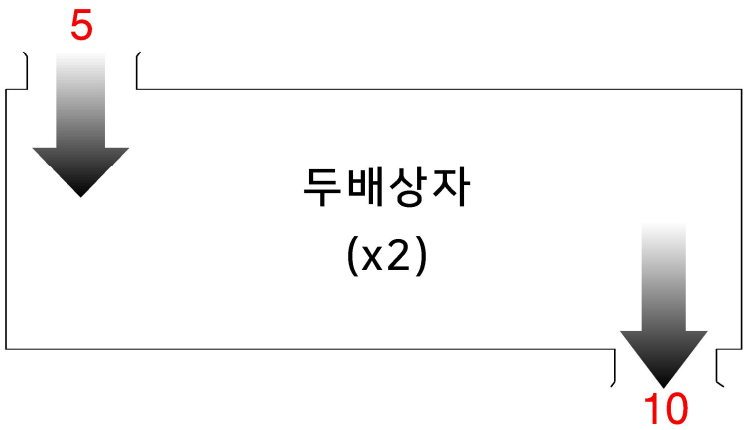
여기서는 켜고, **기다리고**, 끄고, 켜고, **기다리고**, 끄고... 이렇게 반복됩니다. 그런데 잘 보면 끈 다음 바로 켭니다. 즉, 꼬마전구를 끄자마자 바로 다시 켜버리는 것이죠.

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

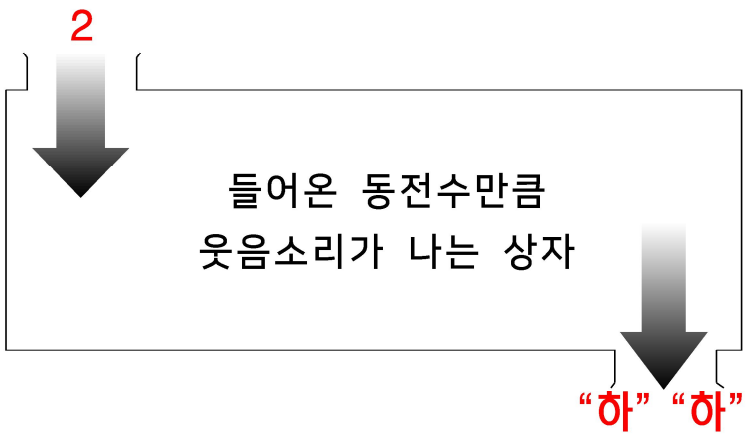
초등학교 교과서에 나오는 함수의 모습은 이렇습니다.




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.

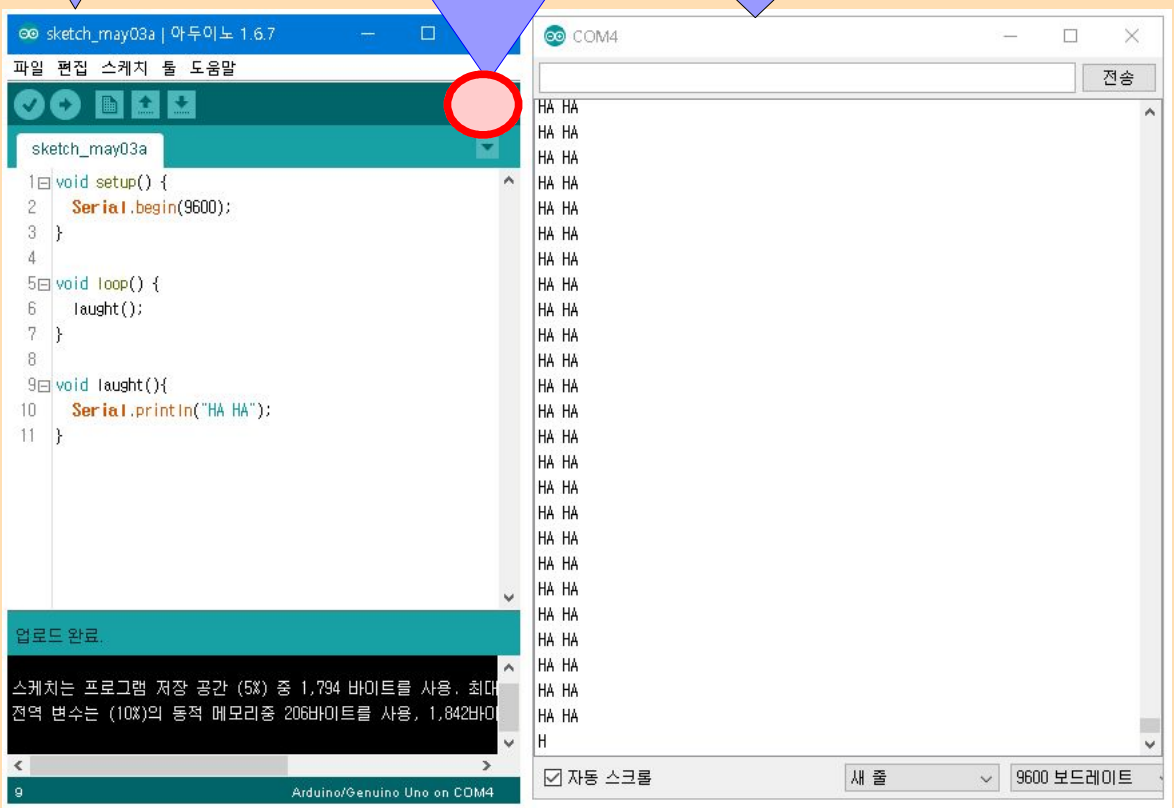
4 다시 loop() 가 시작됩니다.

: 3번으로 갑니다. (# 표시된 줄을 계속 반복합니다.)

아두이노 IDE를 실행합니다.

오른쪽 위 확대경 모양 아이콘을 클릭합니다.

시리얼모니터창이 열리고 시리얼통신내용이 보입니다.



## 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.