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.

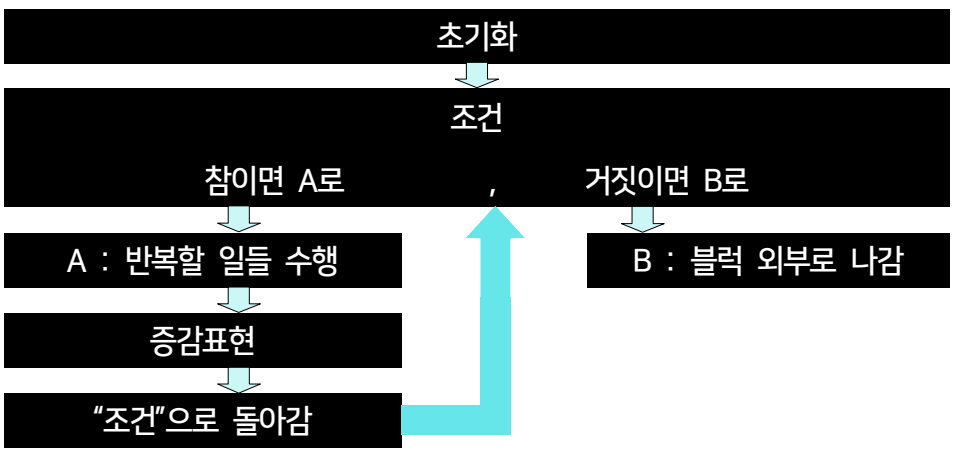
```
for ( 초기화; 조건; 증감표현)
{
    반복할 일들;
}
```

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

5(V) = I(A) x 360(Ω)

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.
```

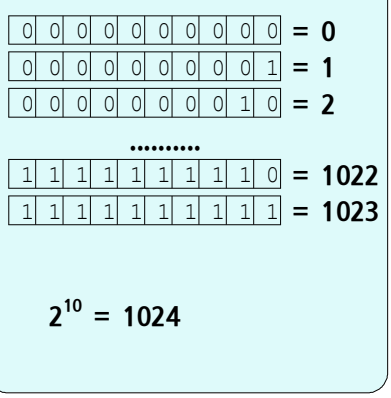
e. analogRead(핀)

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.

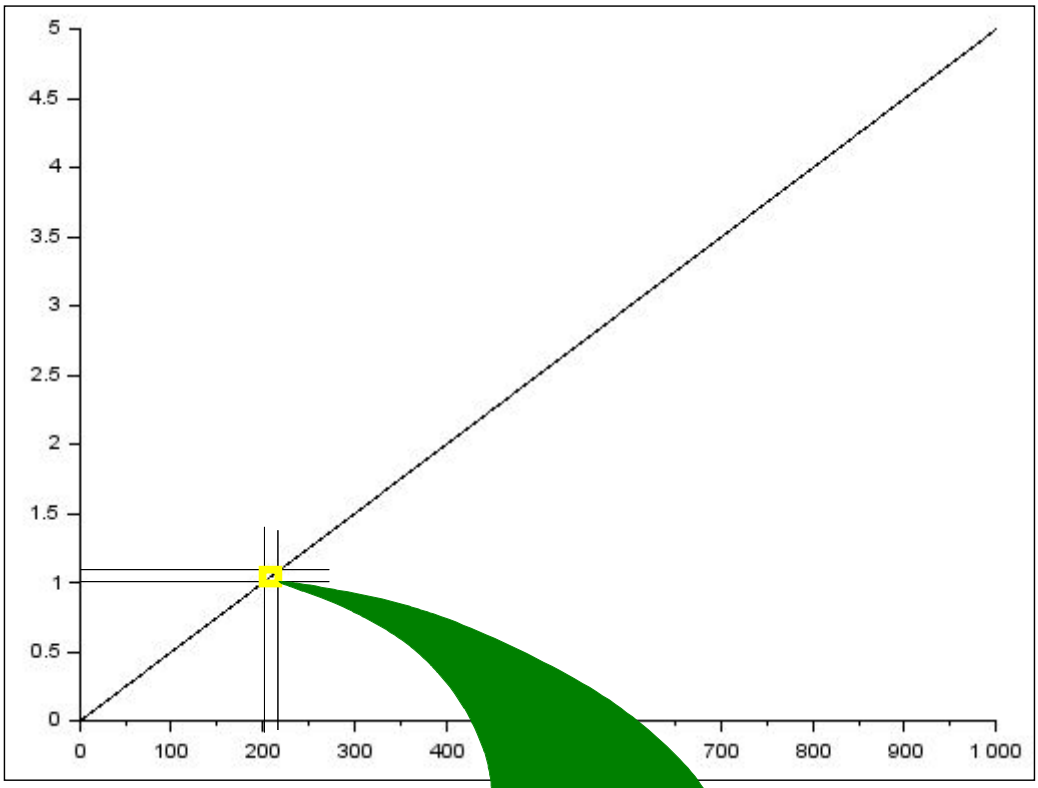


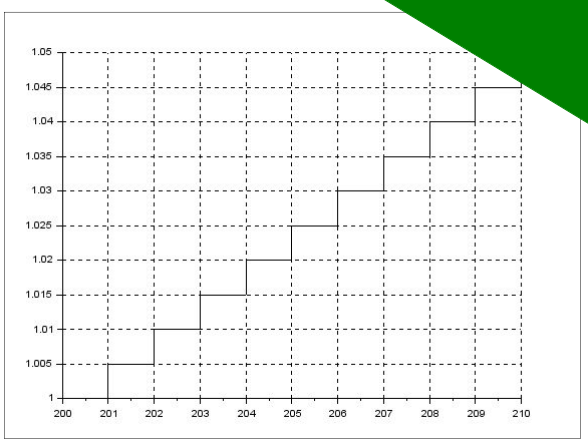
ADC 라고 불리는 아날로그-디지탈 변환장치는 변환시킬 수 있는 정도와 시간에 따라 분류됩니다. 변환시킬 수 있는 정도를 보통 비트(bits)로 표시하고, 변환에 걸리는 시간을 Hz 로 표시합니다.

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

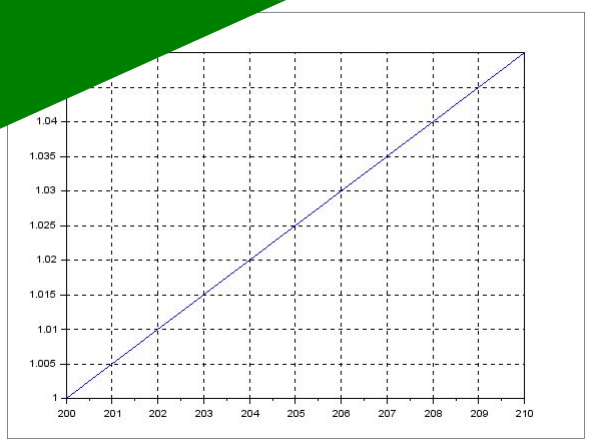
ADC 10 비트는 총 1024 (≒1000) 단계로 입력된 아날로그 값을 변환합니다. 이 경우 0.0048828V(≒0.005V) 즉, 4.89mV 단계로 입력되어지는 전압 값을 구별할 수 있습니다.

TIP





ADC를 거친 후 아두이노에서 받는 값

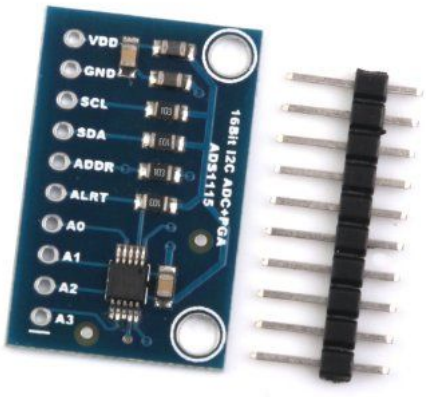


실제 아날로그 값

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.



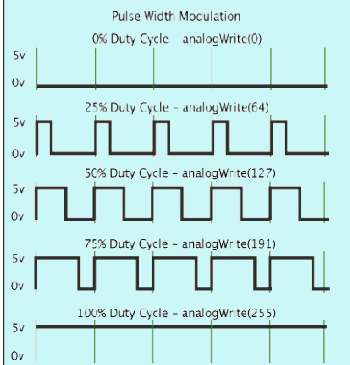
아두이노 보드와 연결해서 사용할 수 있는 16비트 ADC 보드입니다. 16비트로 ADC를 사용하기 때문에 2의 16승인 65536 단계로 나눠집니다. 아두이노가 1024 단계였던 것과 비교하면 64배 ($=2^6$) 큰 것입니다.

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?