

7. 데이터 입출력 응용

이전 몇 개의 장에서 아날로그 및 디지털 데이터의 입출력 방법에 대해 알아보았다. 또한 3장에서 직렬 통신을 위해 사용되는 Serial 클래스에 대해서 알아보았다. 이 장에서는 지금까지 설명한 내용들을 이용하여 간단한 스케치를 작성해 봄으로써 아두이노 또는 마이크로컨트롤러를 이용하여 간단한 시스템을 구성하고 이를 위해 필요한 스케치의 작성 방법에 익숙해지도록 한다. 또한 스케치 작성을 위해 흔히 사용되는 프로그래밍 기법들도 알아본다.

1. LED 모듈에 패턴 표시하기

이전 장에서 LED 모듈을 이용하는 방법에 대해 알아보았다. 먼저 보다 복잡한 패턴을 LED 모듈에 표시하는 방법에 대해 알아보자. 간단한 패턴의 경우, 예를 들어 8개의 LED 중 순서대로 하나의 LED만을 켜는 경우에는 간단한 계산만으로 가능하였다. 그림 1에서와 같이 8개의 LED에 6개의 패턴을 순서대로 나타내는 경우를 생각해 보자.

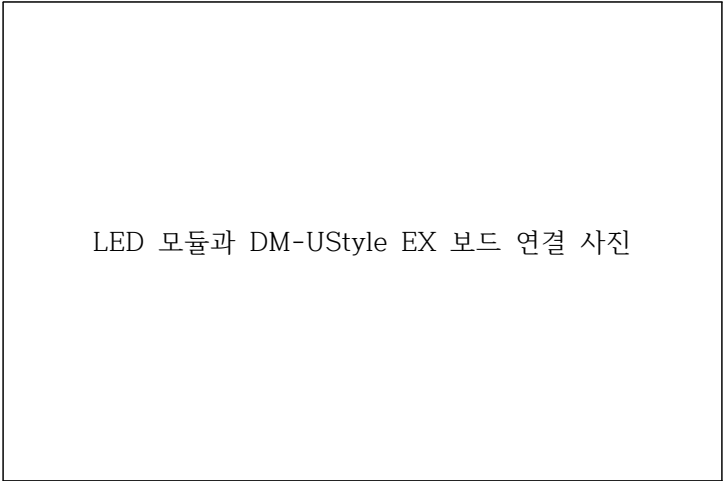
패턴	LED 1	LED 2	LED 3	LED 4	LED 5	LED 6	LED 7	LED 8
1	ON	OFF	ON	OFF	OFF	ON	OFF	ON
2	OFF	ON	OFF	ON	ON	OFF	ON	OFF
3	OFF	OFF	ON	OFF	OFF	ON	OFF	OFF
4	OFF	OFF	OFF	ON	ON	OFF	OFF	OFF
5	OFF	OFF	ON	OFF	OFF	ON	OFF	OFF
6	OFF	ON	OFF	ON	ON	OFF	ON	OFF

그림 1. LED 출력 패턴 ( : ON,  : OFF)

그림 1의 패턴은 간단한 계산으로는 켜고 끄는 위치를 결정하기가 쉽지 않다. 이처럼 복잡한 패턴의 경우 값을 계산하기보다는 패턴 자체를 저장하는 것이 일반적이다. 그림 1의 패턴은 2차원 배열로 다음과 같이 저장할 수 있다.

```
int patterns[6][8] = {
    {1, 0, 1, 0, 0, 1, 0, 1},
    {0, 1, 0, 1, 1, 0, 1, 0},
    {0, 0, 1, 0, 0, 1, 0, 0},
    {0, 0, 0, 1, 1, 0, 0, 0},
    {0, 0, 1, 0, 0, 1, 0, 0},
    {0, 1, 0, 1, 1, 0, 1, 0} };
```

스케치 1은 패턴 데이터를 이용하여 0.5초 간격으로 패턴의 점멸 상태를 바꾸는 스케치의 예이다. LED 모듈은 디지털 5번에서 12번까지 8개의 핀에 연결 하였다. 그림 1에 보이는 것처럼 패턴이 나타나는가?



LED 모듈과 DM-UStyle EX 보드 연결 사진

그림 2. LED 모듈 연결

스케치 1. LED 모듈에 패턴 나타내기 I

```
int patterns[6][8] = {
    {1, 0, 1, 0, 0, 1, 0, 1},
    {0, 1, 0, 1, 1, 0, 1, 0},
    {0, 0, 1, 0, 0, 1, 0, 0},
    {0, 0, 0, 1, 1, 0, 0, 0},
    {0, 0, 1, 0, 0, 1, 0, 0},
    {0, 1, 0, 1, 1, 0, 1, 0} };
```

```
int pins[] = {5, 6, 7, 8, 9, 10, 11, 12};    // LED 모듈 연결 핀
int index;                                  // 현재 출력 패턴 번호
int patterns_no = 6;                        // 패턴의 개수

void setup(){
    for(int i = 0; i < 8; i++){              // LED 모듈 연결 핀을 출력으로 설정
        pinMode(pins[i], OUTPUT);
    }
    index = 0;
}

void loop(){
    index = (index + 1) % patterns_no;      // 패턴의 반복 설정

    for(int i = 0; i < 8; i++){
        digitalWrite(pins[i], patterns[index][i]);
    }

    delay(500);
}
```

스케치 1의 경우 패턴을 저장하기 위해 int 형 배열을 사용하였다. int 형은 2바이트의 저장 공간을 필요로 하므로 스케치 1에서 패턴을 저장하기 위해서 총 48 바이트의 메모리를 사용한다. ATmega328의 경우 프로그램 설치를 위해 사용할 수 있는 플래시 메모리의 크기가 32KB로 그리 크지 않으므로 가능하면 메모리 사용량을 줄이는 것이 바람직하다. 메모리 사용량을 줄이기 위해 스케치 1에서 int 형으로 지정된 패턴을 비트 단위로 지정할 수 있다. 디지털 데이터의 경우 실제로 입출력되는 데이터는 비트 단위의 데이터이므로 특정 순간에 8개의 LED를 점멸하는 데이터는 8비트, 즉 1 바이트로 표현할 수 있다. 스케치 1에 사용된 int 단위의 패턴을 비트 단위로 표현하여 byte 형으로 나타내면 다음과 같다.

```
byte patterns[6] = {  
    0b10100101,  
    0b01011010,  
    0b00100100,  
    0b00011000,  
    0b00100100,  
    0b01011010 };
```

byte 형 데이터에서 각 위치의 비트값을 알아내기 위해서는 bitRead 함수를 사용할 수 있다.

```
bitRead(x, n)  
    x : 데이터  
    n : 읽어낼 비트 위치로 LSB부터 0에서 시작  
    반환값 : 0 또는 1의 비트값
```

이외에도 특정 위치에 비트 값을 기록하는 bitWrite, 특정 위치의 비트 값을 1로 만드는 bitSet, 특정 위치의 비트 값을 0으로 만드는 bitClear 등의 함수를 통해 비트 단위의 조작이 가능하다. 자세한 사용 방법은 부록의 함수 목록을 참고하면 된다.

스케치 2는 비트 단위의 패턴을 사용하여 스케치 1을 수정한 스케치이다. 결과는 스케치 1과 동일하며 프로그램이 업로드 된 후 출력되는 메시지를 통해 스케치의 크기가 줄어들었음을 확인할 수 있다.

스케치 2. LED 모듈에 패턴 나타내기 II

```
byte patterns[] = {  
    0b10100101,  
    0b01011010,  
    0b00100100,  
    0b00011000,  
    0b00100100,  
    0b01011010 };
```

```
int pins[] = {5, 6, 7, 8, 9, 10, 11, 12};    // LED 모듈 연결 핀
int index;                                   // 현재 출력 패턴 번호
int patterns_no = 6;                         // 패턴의 개수

void setup(){
    for(int i = 0; i < 8; i++){               // LED 모듈 연결 핀을 출력으로 설정
        pinMode(pins[i], OUTPUT);
    }
    index = 0;
}

void loop(){
    index = (index + 1) % patterns_no;

    for(int i = 0; i < 8; i++){
        boolean on_off = bitRead(patterns[index], i);
        digitalWrite(pins[i], on_off);
    }

    delay(500);
}
```

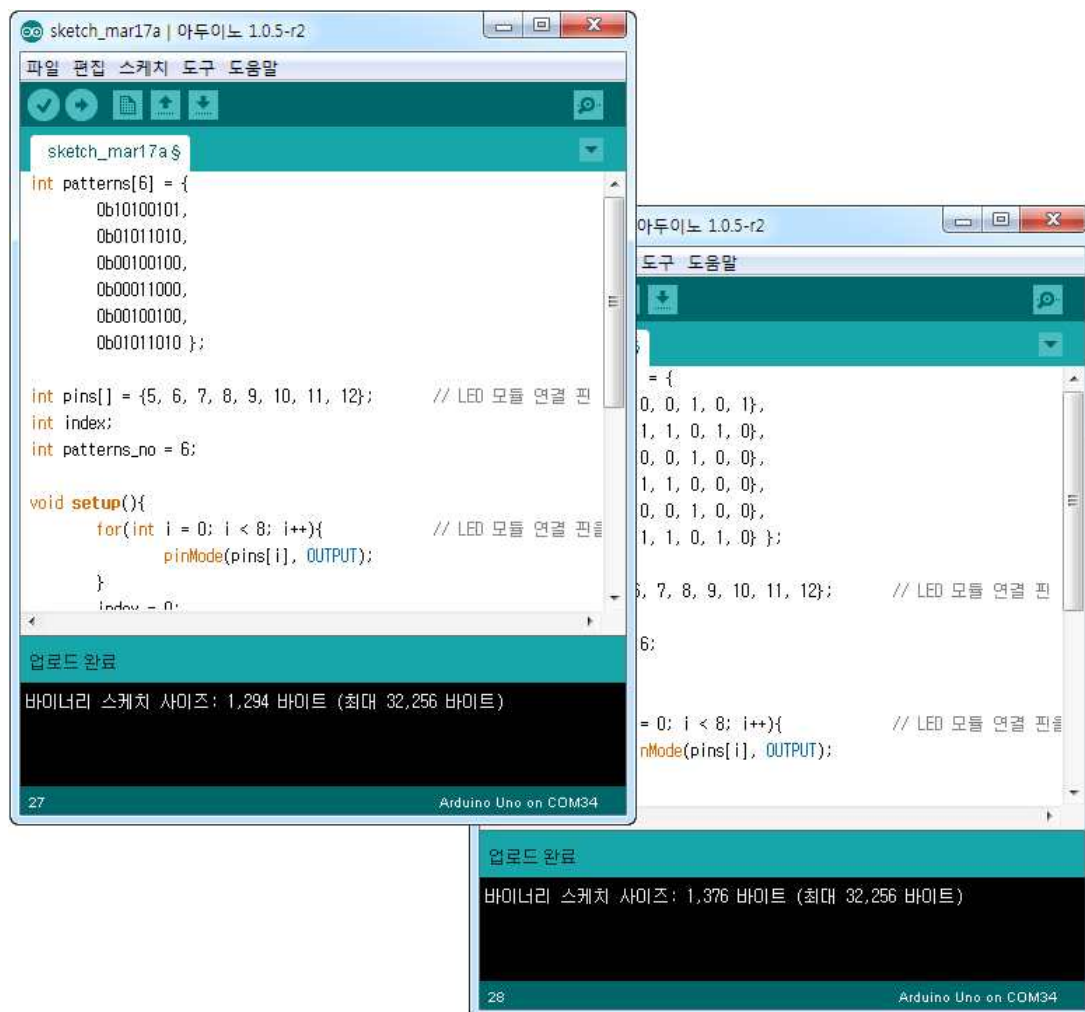


그림 3. 스케치 1과 스케치 2의 크기 비교

2. LED 모듈에 패턴 바꾸기

앞 절에서 LED 모듈에 패턴을 표시하는 방법을 살펴보았다. 여기에 버튼 입력을 추가하여 버튼을 통해 패턴의 표시 순서를 바꾸는 방법을 알아보자. 사용할 패턴은 간단하게 두 개의 인접한 LED가 켜지는 그림 4의 패턴을 사용한다. 출력 패턴은 이전의 스케치들과 마찬가지로 비트 표현으로 저장된 배열을 사용한다.

패턴	LED 1	LED 2	LED 3	LED 4	LED 5	LED 6	LED 7	LED 8
1	ON	ON	OFF	OFF	OFF	OFF	OFF	OFF
2	OFF	ON	ON	OFF	OFF	OFF	OFF	OFF
3	OFF	OFF	ON	ON	OFF	OFF	OFF	OFF
4	OFF	OFF	OFF	ON	ON	OFF	OFF	OFF
5	OFF	OFF	OFF	OFF	ON	ON	OFF	OFF
6	OFF	OFF	OFF	OFF	OFF	ON	ON	OFF
7	OFF	OFF	OFF	OFF	OFF	OFF	ON	ON
8	ON	OFF	OFF	OFF	OFF	OFF	OFF	ON

그림 4. LED 출력 패턴 ( : ON,  : OFF)

스케치 3은 그림 4의 패턴을 0.5초 간격으로 반복해서 출력하는 스케치의 예이다.

스케치 3. LED 모듈에 패턴 나타내기 III

```

byte patterns[] = {
  0b11000000,
  0b01100000,
  0b00110000,
  0b00011000,
  0b00001100,
  0b00000110,
  0b00000011,
  0b10000001 };

int pins[] = {5, 6, 7, 8, 9, 10, 11, 12}; // LED 모듈 연결 핀
int index; // 현재 출력 패턴 번호
int patterns_no = 8; // 패턴의 개수

void setup(){
  for(int i = 0; i < 8; i++){ // LED 모듈 연결 핀을 출력으로 설정
    pinMode(pins[i], OUTPUT);
  }
  index = 0;

```

```

}

void loop(){
    index = (index + 1) % patterns_no;

    for(int i = 0; i < 8; i++){
        boolean on_off = bitRead(patterns[index], i);
        digitalWrite(pins[i], on_off);
    }

    delay(500);
}

```

스케치 3의 경우 그림 4의 패턴을 1번에서 8번 순서로 반복해서 보여준다. 디지털 9번 핀에 풀다운 저항을 이용하여 스위치를 연결하고 스위치를 누르면 나타내는 패턴의 순서를 8번에서 1번 순서로 바꾸도록 해보자. 스위치가 눌러질 때마다 패턴의 순서를 토글 되는 것으로 한다. 스위치를 사용하므로 LED 모듈은 디지털 9번 핀을 제외한 5, 6, 7, 8, 10, 11, 12, 13번 핀에 연결한다.

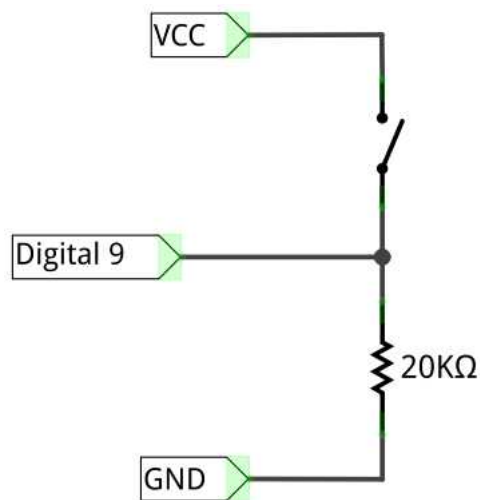


그림 5. 풀다운 저항 연결

스케치 4. 버튼에 의한 LED 모듈의 표현 패턴 순서 바꾸기 - delay 함수

```
byte patterns[] = {
    0b11000000,
    0b01100000,
    0b00110000,
    0b00011000,
    0b00001100,
    0b00000110,
    0b00000011,
    0b10000001 };

int pins[] = {5, 6, 7, 8, 10, 11, 12, 13}; // LED 모듈 연결 핀
int button_pin = 9; // 버튼 연결 핀
int index; // 현재 출력 패턴 번호
int patterns_no = 8; // 패턴의 개수
int step = 1; // 패턴 번호 증가 또는 감소

void setup(){
    for(int i = 0; i < 8; i++){ // LED 모듈 연결 핀을 출력으로 설정
        pinMode(pins[i], OUTPUT);
    }
    pinMode(button_pin, INPUT);
    index = 0;
}

void loop(){
    if(digitalRead(button_pin)){ // 버튼에 의해 증가 또는 감소 방향 토글
        step *= -1;
    }

    index += step; // 패턴 번호 증가 또는 감소
    // 경계 조건
    if(index < 0) index = patterns_no - 1;
    if(index >= patterns_no) index = 0;

    for(int i = 0; i < 8; i++){
        boolean on_off = bitRead(patterns[index], i);
```

```

        digitalWrite(pins[i], on_off);
    }

    delay(500);
}

```

스케치 4를 실행시키고 버튼을 눌러보면 패턴이 표시되는 방향이 바뀌는 것을 확인할 수 있다. 하지만 delay 함수를 사용하였으므로 버튼을 눌러도 패턴의 방향이 바뀌지 않는 경우가 있다. 스케치 4에서 delay 함수를 없애고 millis 함수를 사용하여 버튼이 눌리는 것을 즉각적으로 반영할 수 있도록 수정한 스케치가 스케치 5이다.

스케치 5. 버튼에 의한 LED 모듈의 표현 패턴 순서 바꾸기 - millis 함수

```

byte patterns[] = {
    0b11000000,
    0b01100000,
    0b00110000,
    0b00011000,
    0b00001100,
    0b00000110,
    0b00000011,
    0b10000001 };

int pins[] = {5, 6, 7, 8, 10, 11, 12, 13}; // LED 모듈 연결 핀
int button_pin = 9; // 버튼 연결 핀
int index; // 현재 출력 패턴 번호
int patterns_no = 8; // 패턴의 개수
int step = 1; // 패턴 번호 증가 또는 감소
unsigned long time_previous, time_current;
unsigned long INTERVAL = 500; // 패턴 변화 시간 간격. 0.5초

void setup(){
    for(int i = 0; i < 8; i++){ // LED 모듈 연결 핀을 출력으로 설정
        pinMode(pins[i], OUTPUT);
    }
}

```

```

    }
    pinMode(button_pin, INPUT);
    index = 0;
    time_previous = millis();
}

void loop(){
    if(digitalRead(button_pin)){                // 버튼에 의해 증가 또는 감소 방향 토글
        step *= -1;
        delay(500);                             // 디바운스
    }

    time_current = millis();
    if(time_current - time_previous > INTERVAL){
        time_previous = time_current;
        index += step;                          // 패턴 번호 증가 또는 감소
        // 경계 조건
        if(index < 0) index = patterns_no - 1;
        if(index >= patterns_no) index = 0;

        for(int i = 0; i < 8; i++){
            boolean on_off = bitRead(patterns[index], i);
            digitalWrite(pins[i], on_off);
        }
    }
}

```

3. 시리얼 모니터를 통한 출력 제어

시리얼 모니터를 통해 0에서 255 사이의 byte 형 데이터를 입력 받고 입력 받은 데이터를 이용하여 LED 모듈을 켜는 스케치를 작성해 보자. 입력 받은 데이터는 0에서 255 사이의 데이터로 제한하며 범위를 벗어나는 경우에는 잘못된 값임을 시리얼 모니터로 출력한다. 수신 받은 값은 이진수열로 변환하여 해당 비트가 1인 위치의 LED만을 켜고 나머지는 끈다. 시리얼 모니터에서 수신되는 데이터는 문자열로 하며 문자열의 끝을 표시하기 위해 New Line ('\n') 옵션을 시리얼 모니터에서 선택한다.

스케치 6. 시리얼 모니터로 LED 모듈 제어

```
int pins[] = {5, 6, 7, 8, 10, 11, 12, 13}; // LED 모듈 연결 핀
String buffer; // 수신 문자열 저장
boolean process; // 문자열 종료 여부

void setup(){
  for(int i = 0; i < 8; i++){ // LED 모듈 연결 핀을 출력으로 설정
    pinMode(pins[i], OUTPUT);
  }

  Serial.begin(9600);
  buffer = ""; // 수신 문자열은 빈 상태로 시작
  process = false; // 문자열 종료되지 않았음
}

void loop(){
  while(Serial.available() > 0){ // 수신된 문자가 존재하는 경우
    char data = Serial.read(); // 문자 단위로 읽기
    if(data == '\n'){ // 문자열이 종료된 경우
      process = true; // 지금까지 수신된 문자열 처리
      break;
    }
    else{
      buffer += data; // 문자열 이어 붙이기
    }
  }

  if(process){ // 문자열이 종료된 경우
    int number = buffer.toInt(); // 정수로 변환

    process = false;
    buffer = "";

    Serial.println("I got " + String(number));
  }
}
```

```
if(number < 0 || number > 255){  
    Serial.println("Wrong number !!"); // 잘못된 문자열 수신  
}  
else{  
    for(int i = 0; i < 8; i++){          // LED 점멸  
        boolean on_off = bitRead(number, i);  
        digitalWrite(pins[i], on_off);  
    }  
}  
}  
}
```

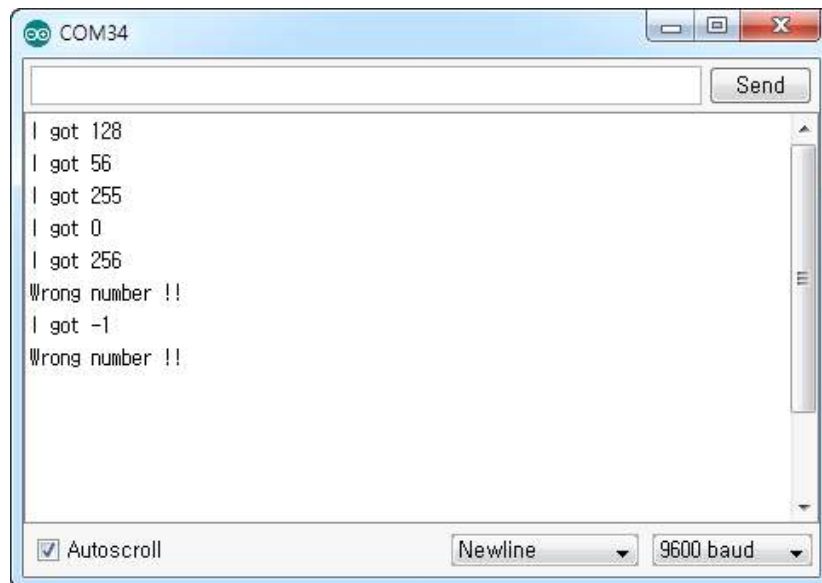


그림 6. 스케치 6 실행 결과

시리얼 모니터에서 수신된 문자열을 숫자로 변환하기 위해 스케치 6에서는 String 클래스의 toInt 멤버 함수를 사용하였다. toInt 함수는 문자열을 숫자로 변환하여 반환하며 변환에 실패한 경우에도 영(zero)을 반환한다. 따라서 숫자로 변환할 수 없는 문자열을 전송한 경우 LED 모듈의 모든 LED는 꺼지게 된다.

4. 2진수를 10진수로 변환하는 게임

LED 모듈과 시리얼 모니터를 이용하여 간단한 게임을 만들어 보자. 먼저 1에서 255 사이의 난수를 생성하고 이를 LED 모듈의 LED를 통해 이진수로 보여준다. 사용자는 LED 모듈에 나오는 이진수를 십진수로 계산하여 시리얼 모니터로 전송하고 전송된 숫자와 생성된 숫자의 일치 여부에 따라 메시지를 출력한다.

스케치 7. 이진수를 십진수로 계산하기 게임

```
int pins[] = {5, 6, 7, 8, 10, 11, 12, 13};    // LED 모듈 연결 핀
String buffer;                                // 수신 문자열 저장
boolean process;                              // 문자열 종료 여부

void setup(){
  for(int i = 0; i < 8; i++){                  // LED 모듈 연결 핀을 출력으로 설정
    pinMode(pins[i], OUTPUT);
  }

  Serial.begin(9600);
  buffer = "";
  process = false;

  randomSeed(analogRead(A5));                  // 난수 생성기 초기화
}

void loop(){
  int number = random(1, 256);                 // [1 255] 범위의 난수 생성
  for(int i = 0; i < 8; i++){                  // LED 모듈에 나타내기
    boolean on_off = bitRead(number, i);
    digitalWrite(pins[i], on_off);
  }

  Serial.println();
  Serial.println("Now, start to guess...");

  while(true){                                 // 문자열 입력 대기
    while(Serial.available() > 0){            // 수신 버퍼에 문자가 존재하는 경우
      char data = Serial.read();
    }
  }
}
```

```
if(data == '\n'){                                // 문자열 종료 문자인 경우
    process = true;
    break;
}
else{
    buffer += data;
}
}

if(process){                                     // 문자열 수신이 종료된 경우
    int guess = buffer.toInt();                 // 정수로 변환
    process = false;
    buffer = "";
    Serial.println("You entered " + String(guess));

    if(guess == number){                       // 생성된 난수와 입력한 숫자가 일치
        Serial.println("Okay, you are right.");
    }
    else{
        Serial.println("Sorry, do some more math.");
    }

    break;                                     // 새로운 난수 생성을 위해 종료
}
}
}
```

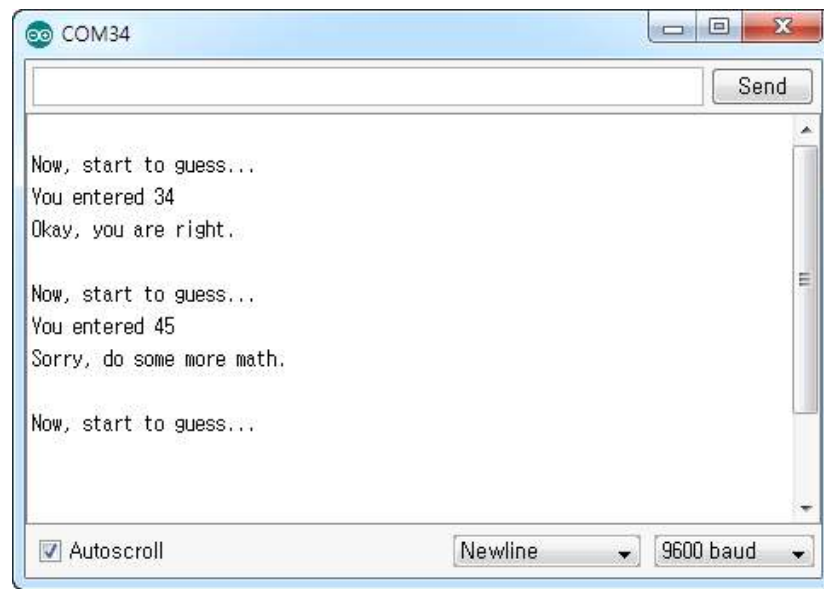


그림 7. 스케치 7 실행 결과