

# random[arq]

**Tutorial** (Versión en español – work in progress)

Por Juanma Sarrió

## ARDUINO + [P5, GRASSHOPPER, FIREFLY]

Arduino open source hardware developed by Massimo Banzi, David Cuartielles, Tom Igoe,  
Gianluca Martino & David Mellis:

<http://www.arduino.cc>

Processing developed by Ben Fry & Casey Reas:

<http://processing.org>

Grasshopper developed by David Rutten:

<http://www.grasshopper3d.com>

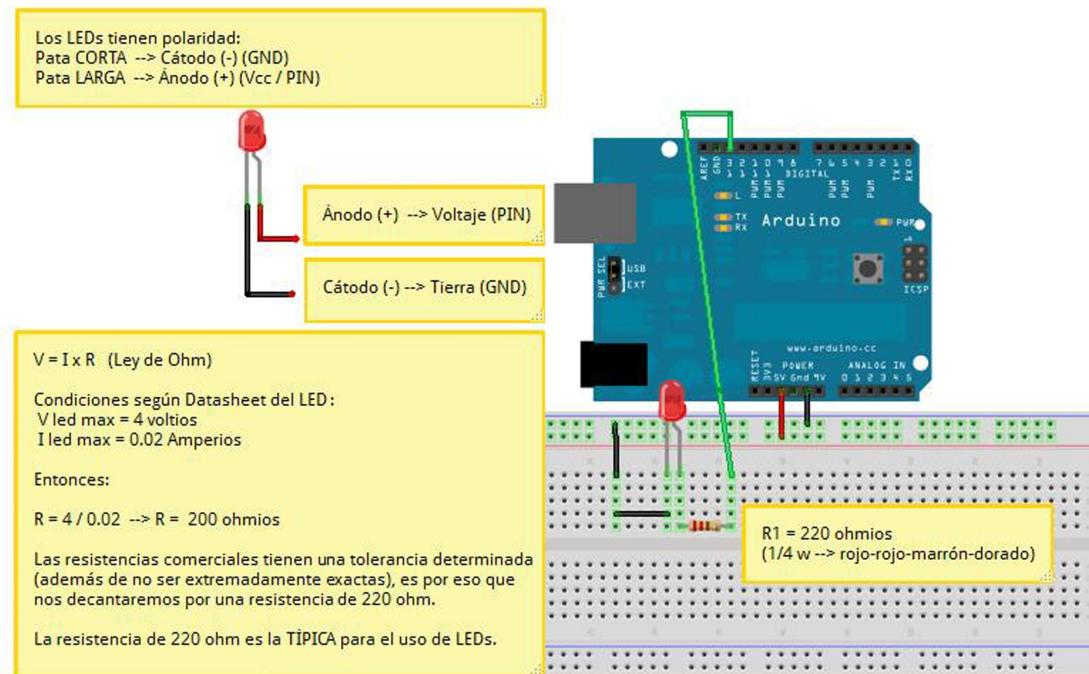
Firefly developed by Andy Payne & Jason K Johnson:

<http://www.fireflyexperiments.com/>

# EJEMPLOS BÁSICOS DE ARDUINO\_

## 01\_Blink

Esquema en Fritzing



Abrimos Arduino y una vez dentro de él, pulsamos el botón **OPEN** (El que tiene la flechita hacia arriba), vamos al submenú **DIGITAL** y abrimos el ejemplo **BLINK**.

### Código del ejemplo Blink de Arduino

```
int ledPin = 13; // LED conectado en el pin digital 13

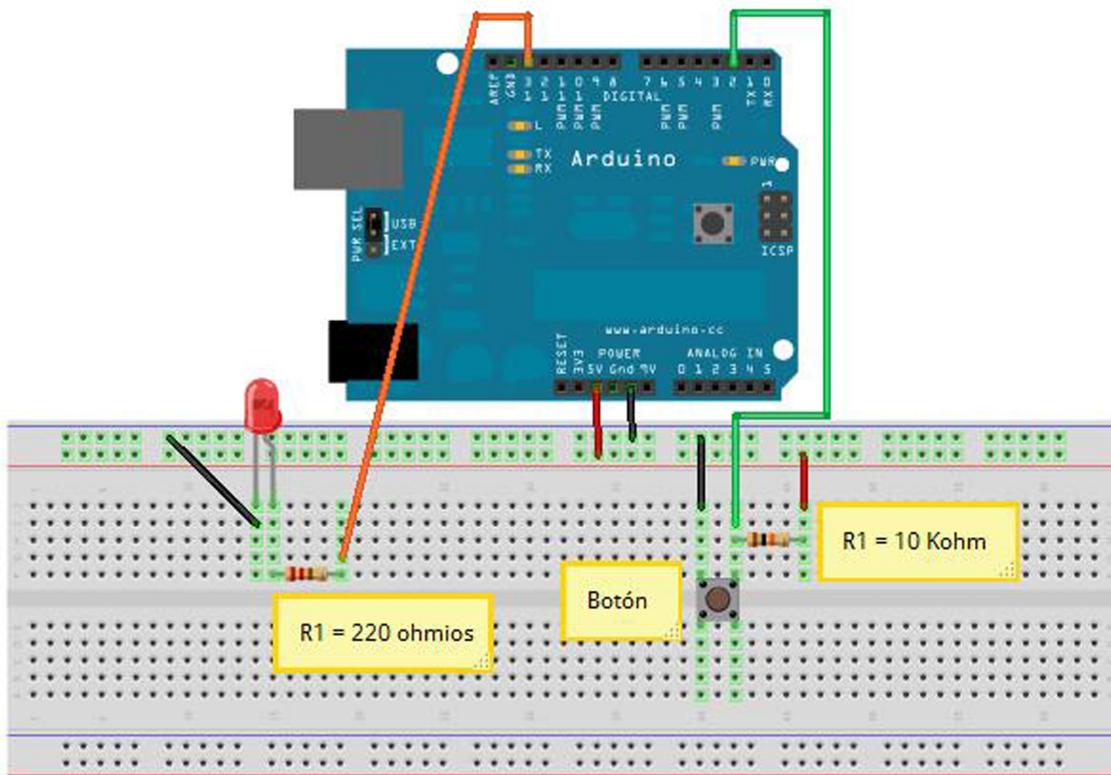
// El Setup () sólo se lee UNA vez (como en Processing).
void setup() {
  pinMode(ledPin, OUTPUT); // Declaramos el pin donde tenemos el Led como SALIDA
}

// El Loop () se lee constantemente.
void loop() {
  digitalWrite(ledPin, HIGH); // Encendemos el LED (HIGH → Le pasamos los 5 voltios)
  delay(1000); // Esperamos 1000 milisegundos (1 seg)
  digitalWrite(ledPin, LOW); // Apagamos el LED (LOW → Le pasamos 0 voltios)
  delay(1000); // Esperamos 1000 milisegundos (1 seg)
}
```

El uso de Leds es muy útil para monitorizar acciones y comprobar que funcionan conexiones o sensores. En este ejemplo, el LED sólo tiene dos posiciones, o encendido o apagado, ya que lo estamos controlando con un pin **DIGITAL** (que sólo admite esas dos posiciones). Sin embargo, un LED puede adoptar más estados como veremos más adelante.

## 02\_Button

Esquema en Fritzing



Pulsamos el botón **OPEN** (El que tiene la flechita hacia arriba), vamos al submenú **DIGITAL** y abrimos el ejemplo **BUTTON**.

### Código del ejemplo Button de Arduino

```
int buttonPin = 2;          // Botón conectado al pin DIGITAL 2
int ledPin = 13;            // LED conectado al pin Digital 13
int buttonState = 0;         // Variable que leerá el estado del Botón ( 0 = LOW ; 1 = HIGH)

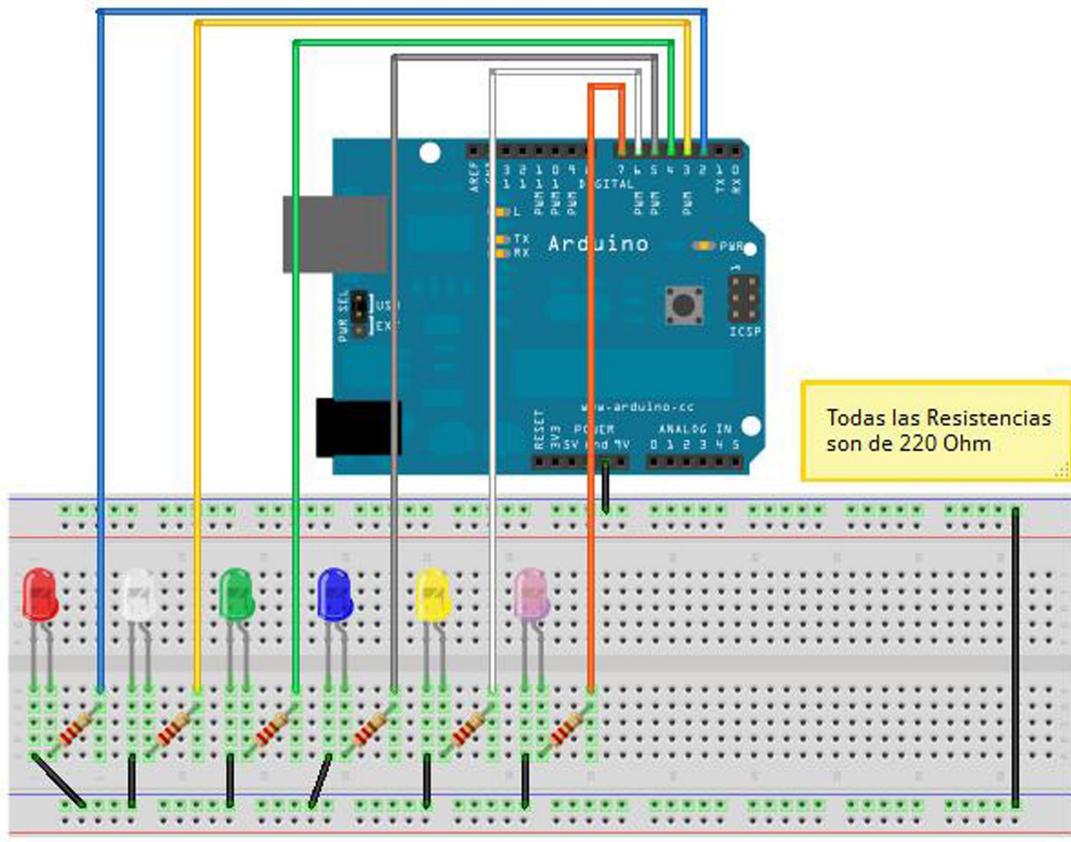
void setup() {
  pinMode(ledPin, OUTPUT);    // Declaramos el LED como SALIDA
  pinMode(buttonPin, INPUT);  // Declaramos el Botón como ENTRADA
}

void loop(){
  // Leemos (digitalmente) el estado del botón y lo guardamos en la variable buttonState
  buttonState = digitalRead(buttonPin);
  // Si el botón está pulsado (HIGH) ...
  if (buttonState == HIGH) {
    digitalWrite(ledPin, LOW); // Encendemos el LED (HIGH → 5 Voltios)
  }
  // ...en caso contrario (el botón no está pulsado → LOW)
  else {
    digitalWrite(ledPin, HIGH); // Apagamos el LED (LOW → 0 Voltios)
  }
}
```

Como vemos, en este sketch, no sucede nada, salvo que apretemos el botón y cambiemos el estado del botón, y a su vez, el de la variable buttonState.

## 03\_Coche Fantástico

Esquema en Fritzing



Pulsamos el botón **OPEN** (El que tiene la flechita hacia arriba), vamos al submenú **CONTROL** y abrimos el ejemplo **ARRAYS**.

### Código del ejemplo Arrays de Arduino

```
int timer = 100; // Variable para controlar el retardo del loop (100 milisegundos)
int ledPins[] = { 2, 7, 4, 6, 5, 3 }; // Array de los pines donde conectaremos los LEDs
int pinCount = 6; // Longitud de la Array
void setup() {
    int thisPin; // Variable temporal
    // La Array está numerada desde 0 hasta (pinCount - 1), es decir de 0 a 5
    // Usamos un bucle FOR para inicializar cada pin como SALIDA
    for (int thisPin = 0; thisPin < pinCount; thisPin++) {
        pinMode(ledPins[thisPin], OUTPUT);
    }
}

void loop() {
```

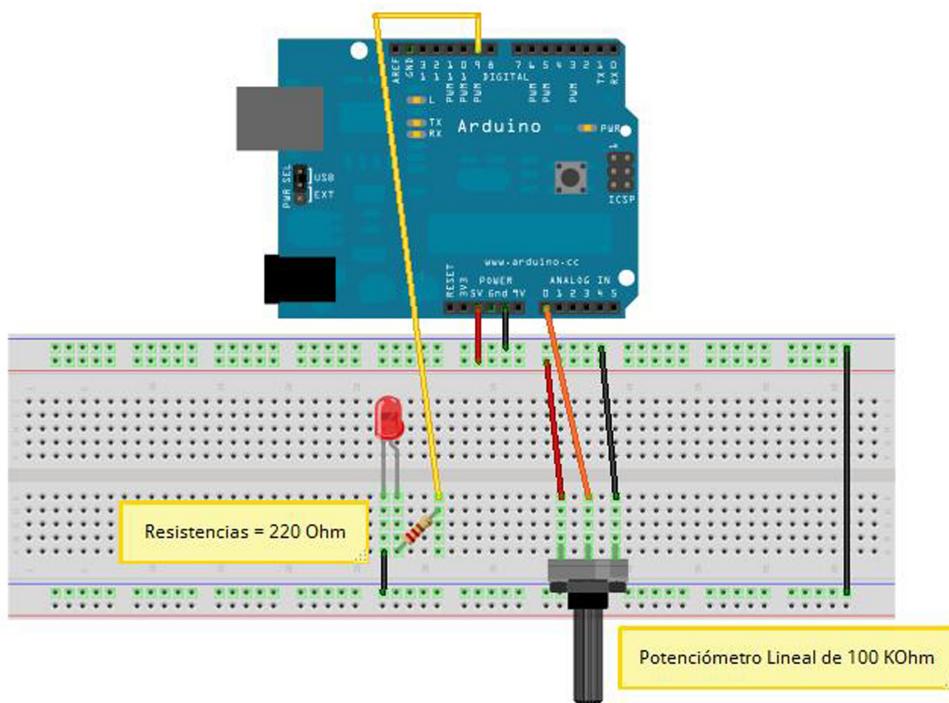
/\* Bucle FOR para ir encendiendo de uno en uno los leds. Funciona así:  
Para el PRIMER led...lo enciendo durante los milisegundos que he indicado con la variable  
timer...y lo apago...enciendo el SEGUNDO led, etc...

```
*/  
  
for (int thisPin = 0; thisPin < pinCount; thisPin++) {  
    digitalWrite(ledPins[thisPin], HIGH);  
    delay(timer);  
    digitalWrite(ledPins[thisPin], LOW);  
}  
  
/* Bucle FOR para ir encendiendo de uno en uno los leds. Funciona así:  
 Para el ÚLTIMO led...lo enciendo durante los milisegundos que he indicado con la variable  
 timer...y lo apago...enciendo el PENÚLTIMO led, etc...  
 */  
  
for (int thisPin = pinCount - 1; thisPin >= 0; thisPin--) {  
    // Encendemos el LED  
    digitalWrite(ledPins[thisPin], HIGH);  
    delay(timer);  
    // Apagamos el LED  
    digitalWrite(ledPins[thisPin], LOW);  
}  
}
```

Este es el primer sketch donde podemos inspirarnos para realizar una instalación (o parte de ella). Posteriormente, veremos cómo poder introducir una interacción con este sketch y que ya no esté “todo” programado, si no que reaccione ante determinada luz, sonido, vibración, temperatura, presencia de obstáculos,...

## 04\_Sensor Analógico

Esquema en Fritzing



Una vez dentro de Arduino, abriremos el ejemplo [04\\_Sensor\\_Analógico\\_Potenciómetro\\_SerialPrintln](#).

Código del ejemplo [04\\_Sensor\\_Analógico\\_Potenciómetro\\_SerialPrintln](#)

```
int potPin = 0;          // Potenciómetro en el pin ANALÓGICO 0
int ledPin = 9;           // LED en el pin PWM 9 (los pines PWM permiten pasar un rango como un tren
de pulsos  PWM → Pulse Width Modulation)
int valor;                // variable para almacenar el valor capturado desde el potenciómetro
int valorMapeado; // variable para almacenar el valor mapeado

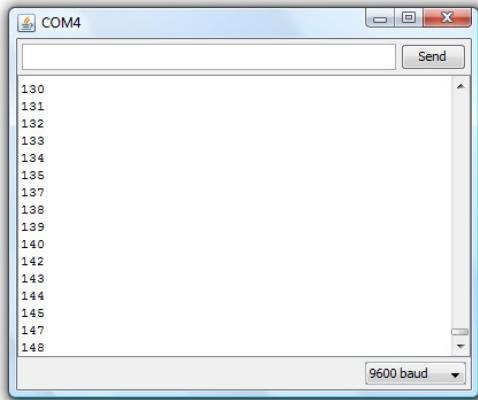
void setup() {
  Serial.begin(9600);      // Abrimos el puerto Serie de Arduino a una velocidad de 9600 baudios
  pinMode(ledPin, OUTPUT); // Declaramos el LED como SALIDA
}

void loop() {
// Leemos analógicamente el potenciómetro y lo guardamos en la variable valor
  valor = analogRead(potPin);

// Mapeamos el valor de la variable valor entre el rango original del potenciómetro (0 → 1023) y lo
pasamos al rango del LED ( 0 → 255 como en la definición de colores de Processing).
  valorMapeado = map(valor,0,1023,0,255);

// Escribimos analógicamente el valorMapeado en el LED
  analogWrite(ledPin,valorMapeado);
```

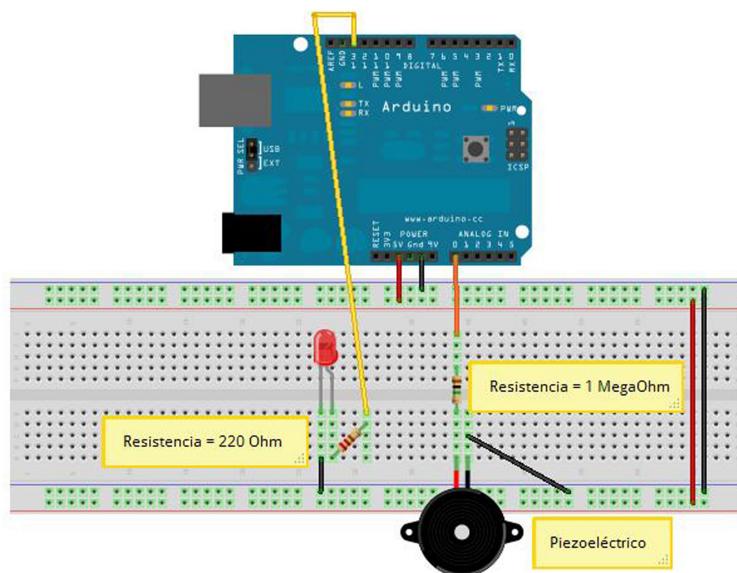
```
// Abrimos el Serial Monitor de Arduino y ahí podremos ver qué valores le vamos pasando al LED
Serial.println(valorMapeado);
}
```



En este ejemplo, vamos a empezar a ver como monitorizar operaciones o eventos en Arduino, gracias al **Puerto Serie**. Esta técnica nos permitirá, posteriormente mandar datos o acciones a otros programas como Processing, Grasshopper, Pure Data, Flash, o incluso a un Iphone/ Ipad o similar ... para dotar de mayor interactividad a nuestra instalación.

## 05\_Piezoeléctrico (Zumbador)

Esquema en Fritzing



```

int ledPin = 13;           // LED en el pin Digital 13
int knockSensor = 0;       // Piezoeléctrico en el pinAnalógico 0
int threshold = 100;        // Valor límite a partir del cual decidimos que se ha detectado un golpe o no
int sensorReading = 0;      // Variable que guarda la lectura del piezoeléctrico
int ledState = LOW;         // Variable que guarda el último estado del LED y que decide ON/OFF

void setup() {
    pinMode(ledPin, OUTPUT); // Declaramos el pin como SALIDA
    Serial.begin(9600);     // Abrimos el Puerto Serie a 9600 baudios
}

void loop() {
    // Leemos analógicamente el piezo y guardamos la lectura en sensorReading
    sensorReading = analogRead(knockSensor);

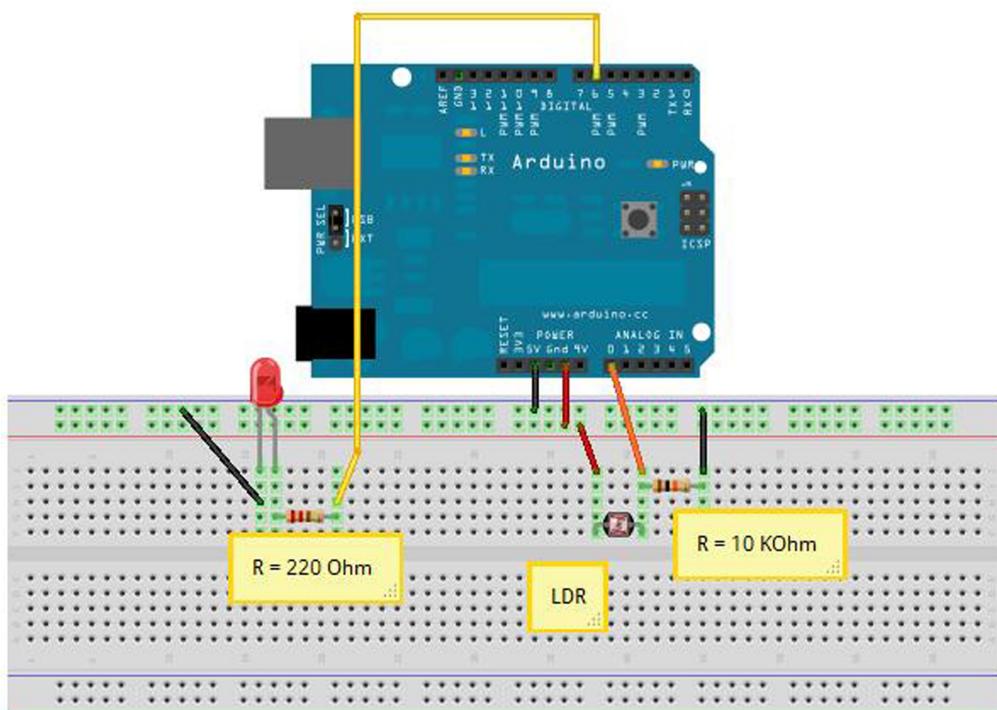
    /* Entramos en un bucle IF...
    Si la lectura guardada en sensorReading es MAYOR O IGUAL que el límite que hemos fijado con la
    variable threshold ....
    */
    if (sensorReading >= threshold) {
        ledState = !ledState;          // Invertimos el valor de ledState (de LOW a HIGH)
        digitalWrite(ledPin, ledState); // Escribimos digitalmente ese valor en el LED
    }
    // Escribimos en el Puerto Serie la palabra (es una String de texto) para saber que se ha cumplido la
    // acción
    Serial.println("Knock!");
}
delay(100); // Retrasamos 100 milisegundos que el programa vuelva a empezar otro bucle del loop
(). Esta orden es muy útil para evitar que bloqueemos el Puerto Serie.
}

```

Los piezoeléctricos son unos elementos muy útiles, ya que son micrófonos de contacto. Si le pasamos una vibración, nos puede devolver un voltaje, y si le pasamos voltaje, nos puede devolver un sonido. Son actuadores fundamentales en las instalaciones sonoras caseras, como veremos posteriormente, como puede ser un **Theremin**.

## 06\_Fotoresistencia (LDR)

Esquema en Fritzing



```

int LDR_pin = 0;          // LDR en el pin Analógico 0
int LED_pin = 6;          // LED en el pin Digital PWM 6
int LDR_val;              // Variable que almacena la lectura del LDR
int LDR_mapeado; // Variable mapeada para encender el LED

void setup() {
  Serial.begin(9600);      // Abrimos el Puerto Serie a 9600 baudios
  pinMode(LED_pin,OUTPUT);
  // Aunque no es necesario declarar los pines ANALOGOS como INPUTS, lo haremos
  pinMode(LDR_pin,INPUT);
}

void loop(){
  LDR_val = analogRead(LDR_pin);           // leemos el valor del LDR
  LDR_mapeado = constrain(LDR_val,0,255); // limitamos el valor del LDR al rango [0-255] para el LED
  analogWrite(LED_pin,LDR_mapeado);        // Le pasamos al LED el valor mapeado
  Serial.print("LDR mapeado = "); // escribimos en el Puerto Serie el valor con que se enciende el LED
  Serial.println(LDR_mapeado);
}

```

En este ejemplo se puede ver claramente la disposición de un **divisor resistivo**.

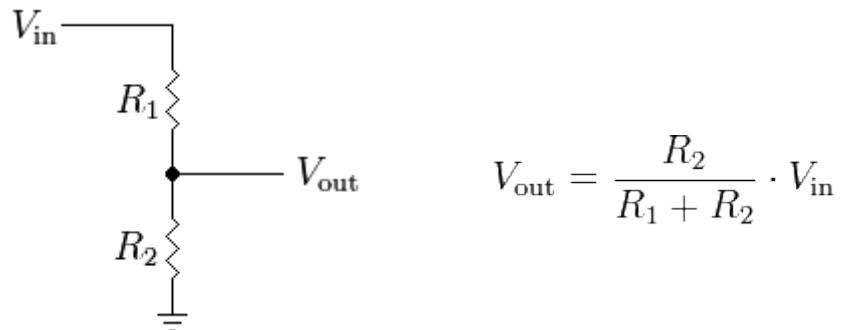
Un divisor de tensión es una configuración de un circuito eléctrico que reparte la tensión de una fuente entre una o más impedancias conectadas en serie.

Un divisor resistivo es un caso especial de divisor de tensión, en el que ambas impedancias son puramente resitivas.

Como Arduino sólo puede leer **voltajes**, y muchos sensores son resitivos, (varían la resistencia)

usamos estos divisores resistivos para leer este tipo de sensores.

### Funcionamiento de un Divisor Resistivo



Este ejemplo funciona exactamente igual si cambiamos la LDR por un **Thermistor** (NTC) (una resistencia por calor).

En este ejemplo podemos observar cómo funcionan las funciones **constrain()** y la diferencia entre **Serial.print ()** y **Serial.println ()**.

Las LDRs son muy útiles para construir instalaciones sensibles a la luz (o a la oscuridad) y también se pueden emplear para fabricar **barreras de luz** (colocamos un puntero láser enfocando constantemente a la LDR y cuando “algo” interrumpa la luz del láser, la LDR dispara un evento para detectar esa acción.)

## 07\_Sensor de Inclinación (TILT)

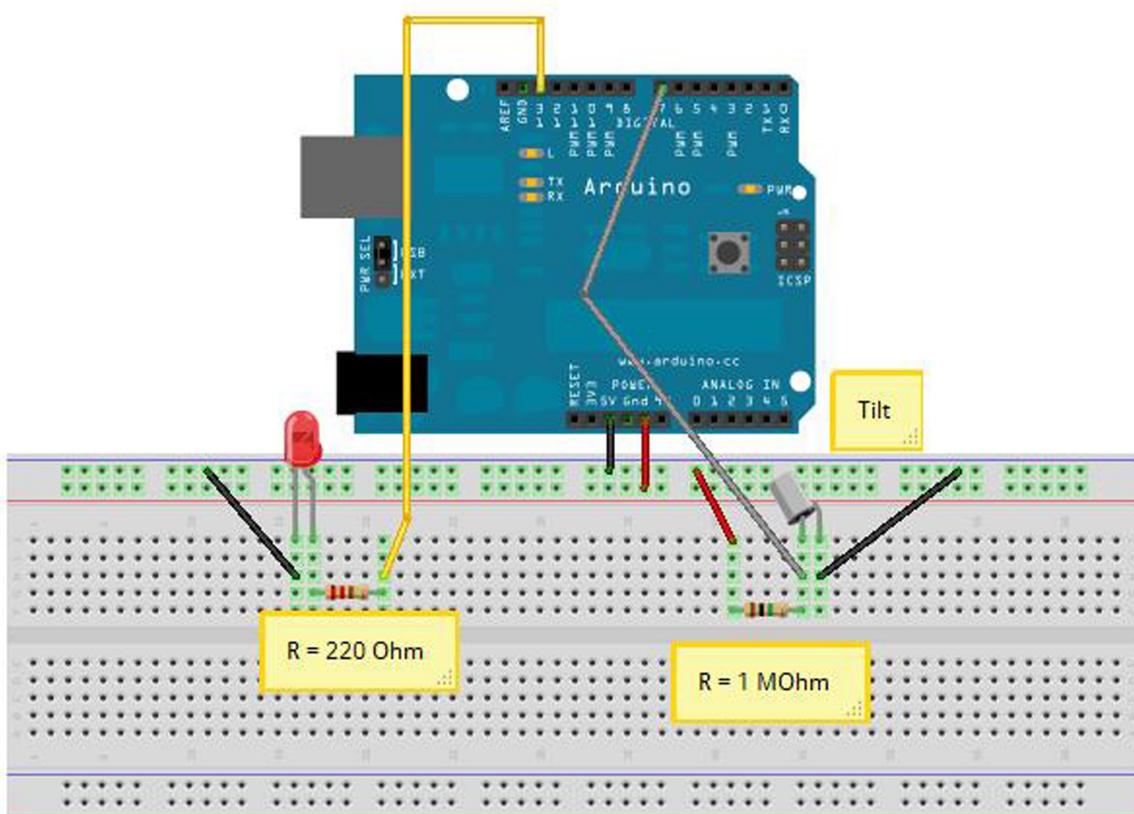
```

int ledPin = 13;           // PIN del LED
int inPin = 7;             // PIN del sensor
int value = 0;              // Valor del sensor

void setup() {
  pinMode(ledPin, OUTPUT);    // Inicializa el pin 13 como salida digital
  pinMode(inPin, INPUT);      // Inicializa el pin 7 como entrada digital
}

void loop() {
  value = digitalRead(inPin); // Lee el valor de la entrada digital
  digitalWrite(ledPin, value);
}

```



Esquema en Fritzing

Un sensor de inclinación es un componente que puede detectar la inclinación de un objeto. Sin embargo, no deja de ser un pulsador activado por un mecanismo físico. Este tipo de sensor es la versión básica de un interruptor de mercurio. El sensor tilt **NO** tiene polaridad.

Este sensor contiene una bola metálica en su interior que comuta los dos pines del dispositivo de encendido a apagado, y viceversa, si el sensor alcanza un cierto ángulo. Es decir, detecta si algo está arriba o abajo. Cuando el sensor detecta una vibración, enciende el LED.

Este actuador está presente en muchos juguetes infantiles, en los que se activan eventos cuando se acaricia o se golpea. Posteriormente, veremos cómo hackear un **Nunchuck** de una Wii para conectarlo a Arduino y ver el funcionamiento de sus 3 acelerómetros.

## 08\_Música con el TILT

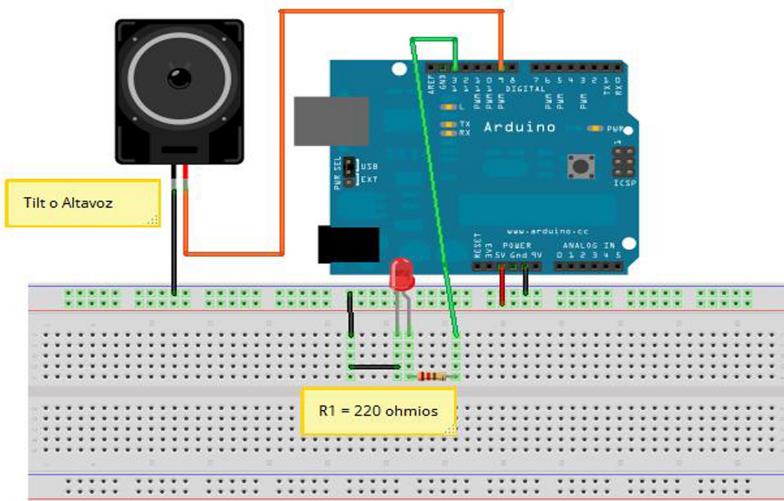
//El Archivo "pitches.h" contiene las frecuencias de las notas que queramos utilizar

```
#include "pitches.h"

int ledPin = 13;      // LED en el pin 13
int altavoz = 9;      // Altavoz o piezoelectrico en el pin PWM 9
// Array de bytes compuesto por caracteres para indicar qué nota debe sonar
byte letras[ ] ={ 'a', 's', 'd', 'f', 'g' };
// notas de la melodía de Encuentros en la 3ª Fase según el archivo "pitches.h"
int notas[ ] = {NOTE_A4, NOTE_C4, NOTE_GS3, NOTE_GS2, NOTE_DS3};
byte valor; // Valor en el que guardamos lo que le pasamos por Puerto Serie
// Byte que le pasamos desde el puerto Serie...
int byteQueIntroducimosPorPuertoSerie = -1;
int estadoLED = LOW; // Al principio el LED está OFF
int contador; // Variable contador para recorrer la Array de notas

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(altavoz, OUTPUT);
  Serial.begin(9600); // Abrimos Puerto Serie a 9600 baudios
}

void loop() {
  // Estamos atentos a la lectura desde el puerto serie para ver que letra escribimos
  byteQueIntroducimosPorPuertoSerie = Serial.read();
  // Si escribimos cualquier letra...
  if (byteQueIntroducimosPorPuertoSerie != -1) {
    // guardamos esa letra en la variable valor
    valor = byteQueIntroducimosPorPuertoSerie;
    // Escribimos la letra en el puerto Serie
    Serial.println(valor, BYTE);
    // Cambiamos el Estado del LED de OFF a ON o de ON a OFF
    estadoLED = !estadoLED;
    // Pasamos el nuevo estado al LED
    digitalWrite(ledPin, estadoLED);
  }
  // Desde 0 hasta recorrer toda la Array (de 0 a 4)...
  for (contador = 0; contador <= 4; contador++) {
    // ...y si a la letra 0 le corresponde el valor 0...
    // es decir...si letra 0 = 'a' ...
    if (letras[contador] == valor) {
      // Le pasamos al pin del altavoz el tono de la letra 0 ...
      tone(altavoz, notas[contador]);
    }
  }
}
```

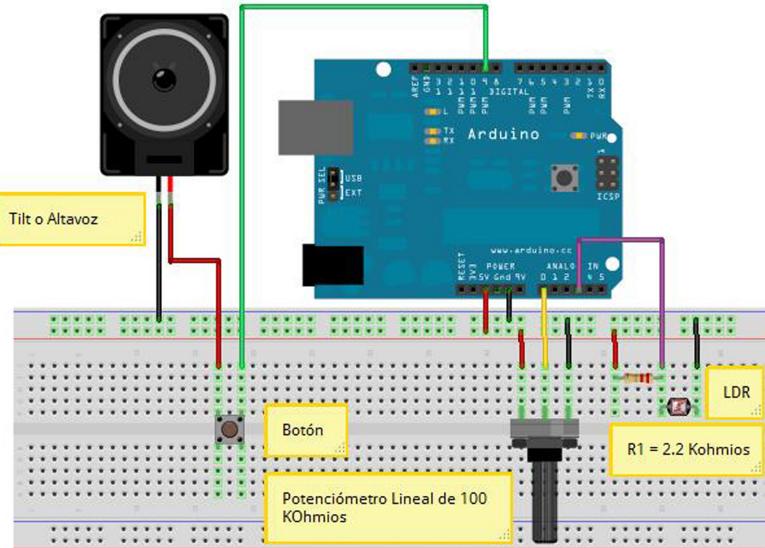


## Esquema en Fritzing

En este ejemplo, podemos ver la utilidad de la función `Serial.read()`, que nos permite introducir parámetros dentro de Arduino para disparar eventos.

## 09\_Theremin

Esquema en Fritzing



```
// PINES SENSORES
int LDR = 3; // LDR en el pin analogico 3
int altavoz = 9; // Altavoz en el pin PWM 9
int potenciometro = 0; // Potenciometro en el pin analogico 0

// VARIABLES PARA ALMACENAR VALORES
int valor = 0;
int valor_mapeado = 0;
int valor_tiempo = 0;

void setup(){
pinMode(LDR, INPUT);
pinMode(altavoz, OUTPUT);
pinMode(potenciometro, INPUT);
Serial.begin(9600);
}

void loop(){
// Guardamos la lectura de la LDR en la variable valor
valor = analogRead(LDR);
// Mapeamos la lectura del potenciometro entre un valor de 100 ---> 500 milisegundos
valor_tiempo = map(analogRead(potenciometro),0,1023,100,1000);
// Mapeamos la lectura del LDR con un rango de tonos de notas determinado
valor_mapeado = map(valor,0,1023,956,1915);
// Introducimos una función creada por nosotros para hacer sonar el altavoz
playTone(valor_mapeado,valor_tiempo);
// Imprimos el valor de la lectura de la LDR
Serial.println(valor,DEC);
// 10 milisegundos de retraso para evitar saturar el puerto serie
delay(10);
}

// Esta es la función que hemos creado. Hay que pasarle dos argumentos:
```

```
// el tono (las notas) y duration (el rango entre 10 y 1000 milisegundos)
void playTone(int tono, int duration) {
    for (long i = 0; i < duration * 1000L; i += tono * 2) {
        digitalWrite(altavoz, HIGH);
        delayMicroseconds(tono);
        digitalWrite(altavoz, LOW);
        delayMicroseconds(tono);
    }
}
```

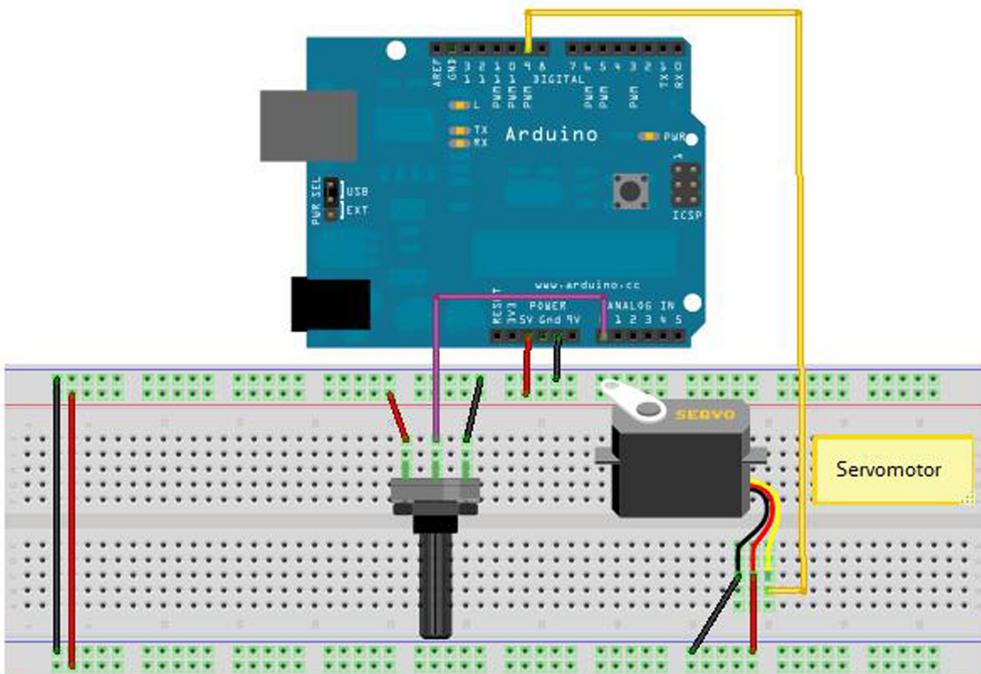
## 10\_Servomotor

```
// Para controlar nuestro Servo, podemos usar la librería Servo
// que ya tiene funciones predeterminadas
#include <Servo.h>
// Creamos un objeto Servo para controlar nuestro servo
Servo miServo;
// Potenciómetro en el Pin Analogico 0
int potenciometro = 0;
// Variable para guardar la lectura del potenciómetro
int valor;

void setup() {
  // Conectamos el servo en el Pin PWM 9
  miServo.attach(9);
}

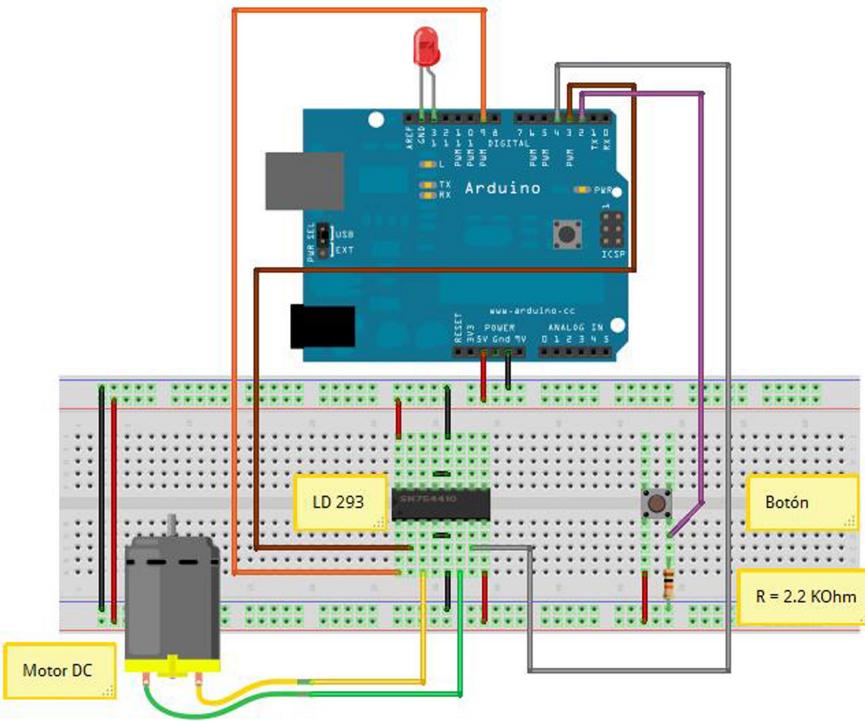
void loop() {
  // Leemos el valor del potenciómetro
  valor = analogRead(potenciometro);
  // Mapeamos su valor al rango del servo [0°-180°]
  valor = map(valor, 0, 1023, 0, 179);
  // Le pasamos al servo el valor escalado
  miServo.write(valor);
  // Retraso de 15 milisegundos para refrescar el servo
  delay(15);
}
```

### Esquema en Fritzing



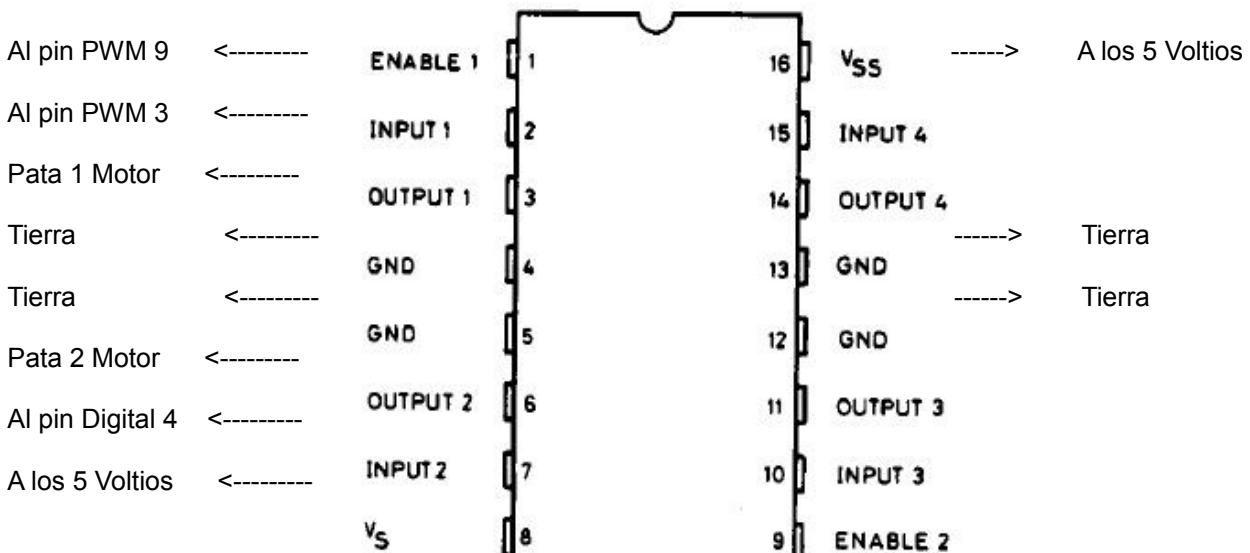
## 11\_Motor DC controlado con el chip LD293

Esquema en Fritzing



Los chips, generalmente, suelen tener una muesca (**semicírculo**) que nos indica la orientación de sus pines. El pin nº 1 de este chip será el que se encuentre a la izquierda de la muesca. El resto de la numeración será antihoraria, como se puede ver en la imagen inferior que muestra en detalle las conexiones del chip LD293.

L293D



Con este chip podemos conectar 2 motores DC, tan sólo con conectar el segundo motor de manera análoga al primero, pero en la parte derecha del chip.

S.6574

```
int botonPin = 2; // Boton al pin digital 2
int motorPin1 = 3; // Pata 1 del motor al pin digital 3
int motorPin2 = 4; // Pata 2 del motor al pin digital 4
int speedPin = 9; // Habilitamos el pin 1 del LD293 y va al PWM 9
int ledPin = 13; // LED

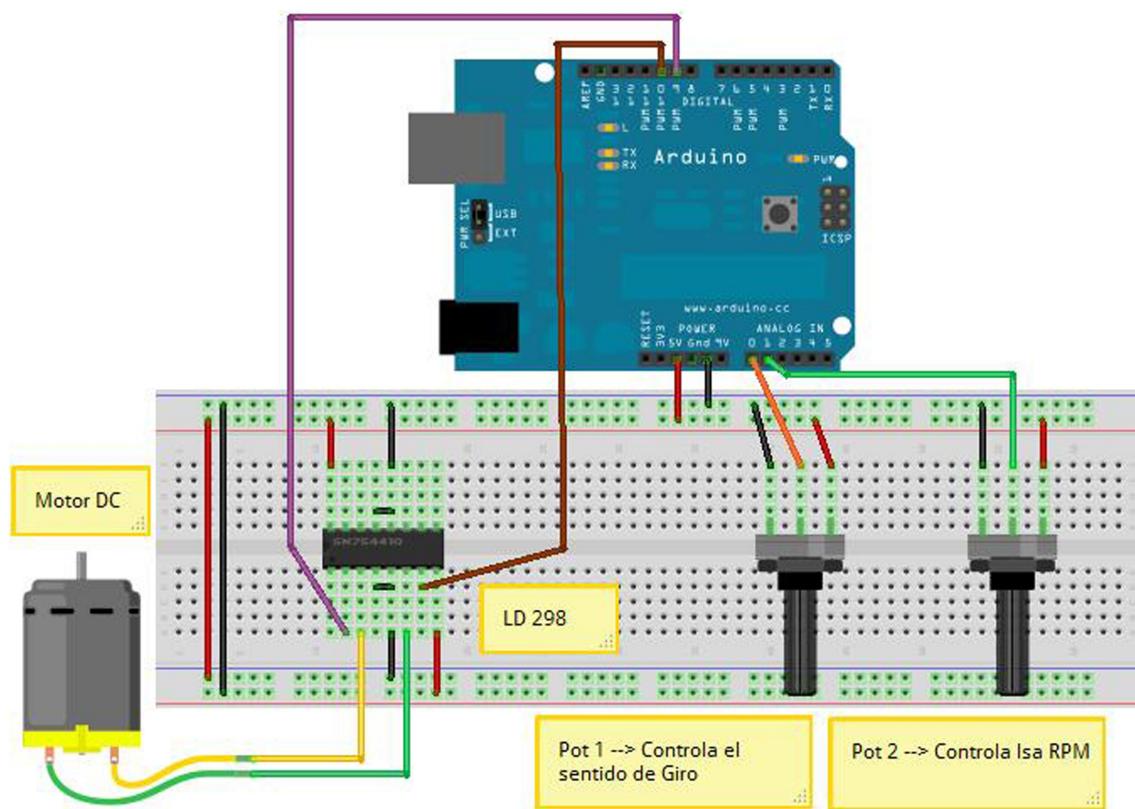
void setup() {
  pinMode(botonPin, INPUT); // Botón como ENTRADA
  pinMode(motorPin1, OUTPUT); // El resto, como SALIDA
  pinMode(motorPin2, OUTPUT);
  pinMode(speedPin, OUTPUT);
  pinMode(ledPin, OUTPUT);
  digitalWrite(speedPin, HIGH); // Arrancamos el motor desde el principio
  // Si el led parpadea 3 veces después de arrancar el motor, quiere
  // decir que Arduino se ha reiniciado debido a un cortocircuito
  blinkLED(ledPin, 3, 100);
}

void loop() {
  if (digitalRead(botonPin) == HIGH) { // Si presionamos el botón ...
    // El motor gira en sentido horario
    digitalWrite(motorPin1, LOW);
    digitalWrite(motorPin2, HIGH);
  }
  else {
    digitalWrite(motorPin1, HIGH); // En caso contrario ... el motor gira en sentido antihorario
    digitalWrite(motorPin2, LOW);
  }
}

// Este método nos informa del cortocircuito
void blinkLED(int quePin, int cuantasVeces, int miliSeg) {
  int i = 0;
  for (i = 0; i < cuantasVeces; i++) {
    digitalWrite(quePin, HIGH);
    delay(miliSeg/2);
    digitalWrite(quePin, LOW);
    delay(miliSeg/2);
  }
}
```

## 12\_Motor DC controlado con el chip LD293 y con 2 Potenciómetros que controlan su sentido de giro y sus RPM

Esquema en Fritzing



```

int valor; // variable para leer el pot 1
int valor1; // variable para leer el pot 2
int valor_mapeado; // variable para mapear el pot 2

int motor[ ] = {9, 10}; // Array de Pines PWM para el motor
int pot1 = 0; // Pot 1 al Pin 0 analogico
int pot2 = 1; // Pot 2 al Pin 1 analogico

void setup() {
  Serial.begin(9600);
  pinMode(pot1, INPUT); // Potenciometros como ENTRADA
  pinMode(pot2, INPUT);

  // Declaramos una variable temporal y recorremos la array del Motor como SALIDA
  int i;
  for(i = 0; i < 2; i++){
    pinMode(motor[i], OUTPUT);
  }
}

void loop() {

```

```
valor = analogRead(pot1);           // leemos el pot1
valor1 = analogRead(pot2);          // leemos el pot2
// mapeamos el pot2 para adecuarlo al rango del motor
valor_mapeado = map(valor1,0,1023,0,255);

// Si el valor de pot1 es MENOR que 512 (1/2 del pot)...
if(valor <= 512){
    Serial.print("Giro : Antihorario");
    // El motor RETROCEDE según la lectura mapeada del pot 2
    motorAtras(valor_mapeado);
}

// Si el valor de pot1 es MAYOR que 512 ...
if(valor > 512){
    Serial.print("Giro : Horario");
    // El motor AVANZA según la lectura mapeada del pot 2
    motorAdelante(valor_mapeado);
}

Serial.print("/t");
Serial.print("RPM : ");
Serial.println(valor_mapeado);
delay(50);
}

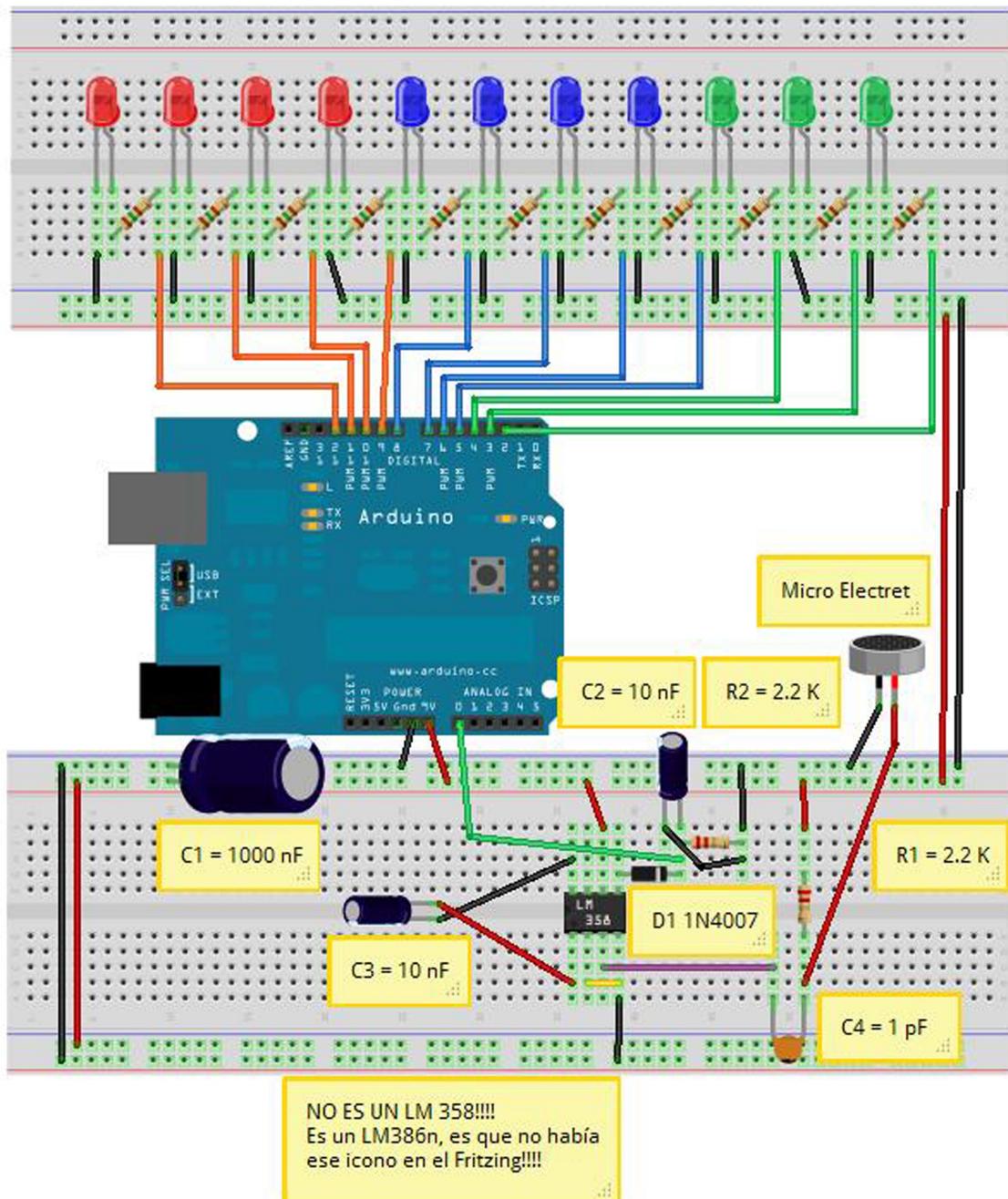
// Función Auxiliar para el AVANCE del Motor.
// Ponemos una pata del Motor a CERO y la otra con la lectura del pot 2
void motorAdelante(int RPM_1){
    analogWrite(motor[0], RPM_1);
    analogWrite(motor[1], 0);
}

// Función Auxiliar para el RETROCESO del Motor.
// Ponemos la pata contraria (la que estaba a CERO en el caso de avance del Motor) con la lectura del
// pot 2 y la otra a CERO
void motorAtras(int RPM_2){
    analogWrite(motor[0], 0);
    analogWrite(motor[1], RPM_2);
}
```

Hay que destacar que en las dos funciones auxiliares, hemos incorporado una variable (RPM\_1 y RPM\_2, respectivamente) que, en el **loop** ( ) se completan con la lectura del potenciómetro 2. Como se puede ver, la sintaxis de Arduino, es prácticamente idéntica a la de Processing. Incluso, para dejar el código más limpio, podríamos haber coladoado las dos funciones auxiliares en una nueva pestaña.

## 13\_Vúmetro con Micrófono Electret y 11 LEDs

Esquema en Fritzing



```
int val; // Variable para la lectura del micro
int microfono = 0; // Micro en el pin Analogico 0
int i; // Variable para recorrer la array de LEDs

int leds [] = { // Array de los Pines de los LEDs
 2,3,4,5,6,7,8,9,10,11,12};
```

```
// En este caso, el total es 11 LEDs (la array empieza en [0])
int longArrayLEDS = 10;

void setup(){
  Serial.begin(9600);
  pinMode(microfono, INPUT);      // Micro como SALIDA
  for(i = 0; i < longArrayLEDS; i++){ // Todos los LEDs como SALIDAS
    pinMode(leds[i], OUTPUT);
  }
}

void loop() {
  val = analogRead(microfono);    // Leemos el micro
  val = map(val,610,1023,0,255);   // Mapeamos para quitar el ruido

// Si el valor es MENOR que 15...apagamos todos los leds
  if(val < 15){
    for(i = 0; i <= longArrayLEDS; i++){
      digitalWrite(leds[i],LOW);
    }
  }

// Si el valor es MAYOR que 15 y MENOR que 23...encendemos el 1er led y el resto apagados
  if(val > 15 && val < 23){
    for(i = 0; i <= longArrayLEDS - 10; i++){
      digitalWrite(leds[i],HIGH);
    }
    for(i = 1; i <= longArrayLEDS; i++){
      digitalWrite(leds[i],LOW);
    }
  }

// Si el valor es MAYOR que 23...encendemos los 2 primeros leds y el resto apagados
  if (val >= 23){
    for(i = 0; i <= longArrayLEDS - 9; i++){
      digitalWrite(leds[i],HIGH);
    }
    for(i = longArrayLEDS - 9; i <= longArrayLEDS; i++){
      digitalWrite(leds[i],LOW);
    }
  }

// Si el valor es MAYOR que 46...encendemos los 3 primeros leds y el resto apagados
  if(val >= 46){
    for(i = 0; i <= longArrayLEDS - 8; i++){
      digitalWrite(leds[i],HIGH);
    }
    for(i = longArrayLEDS - 8; i <= longArrayLEDS; i++){
      digitalWrite(leds[i],LOW);
    }
  }

// Si el valor es MAYOR que 69...encendemos los 4 primeros leds y el resto apagados
  if(val >= 69){
    for(i = 0; i <= longArrayLEDS - 7; i++){
      digitalWrite(leds[i],HIGH);
    }
    for(i = longArrayLEDS - 7; i <= longArrayLEDS; i++){
      digitalWrite(leds[i],LOW);
    }
  }
}
```

```
}

// Si el valor es MAYOR que 92...encendemos los 5 primeros leds y el resto apagados
if(val >= 92){
    for(i = 0; i <= longArrayLEDS - 6; i++){
        digitalWrite(leds[i],HIGH);
    }
    for(i = longArrayLEDS - 6; i <= longArrayLEDS; i++){
        digitalWrite(leds[i],LOW);
    }
}

// Si el valor es MAYOR que 115...encendemos los 6 primeros leds y el resto apagados
if(val >= 115){
    for(i = 0; i <= longArrayLEDS - 5; i++){
        digitalWrite(leds[i],HIGH);
    }
    for(i = longArrayLEDS - 5; i <= longArrayLEDS; i++){
        digitalWrite(leds[i],LOW);
    }
}

// Si el valor es MAYOR que 138...encendemos los 7 primeros leds y el resto apagados
if(val >= 138){
    for(i = 0; i <= longArrayLEDS - 4; i++){
        digitalWrite(leds[i],HIGH);
    }
    for(i = longArrayLEDS - 4; i <= longArrayLEDS; i++){
        digitalWrite(leds[i],LOW);
    }
}

// Si el valor es MAYOR que 161...encendemos los 8 primeros leds y el resto apagados
if(val >= 161){
    for(i = 0; i <= longArrayLEDS - 3; i++){
        digitalWrite(leds[i],HIGH);
    }
    for(i = longArrayLEDS - 3; i <= longArrayLEDS; i++){
        digitalWrite(leds[i],LOW);
    }
}

// Si el valor es MAYOR que 184...encendemos los 9 primeros leds y el resto apagados
if(val >= 184){
    for(i = 0; i <= longArrayLEDS - 2; i++){
        digitalWrite(leds[i],HIGH);
    }
    for(i = longArrayLEDS - 2; i <= longArrayLEDS; i++){
        digitalWrite(leds[i],LOW);
    }
}

// Si el valor es MAYOR que 207...encendemos los 10 primeros leds y el último apagado
if(val >= 207){
    for(i = 0; i <= longArrayLEDS - 1; i++){
        digitalWrite(leds[i],HIGH);
    }
}
```

```
for(i = longArrayLEDS - 1; i <= longArrayLEDS; i++){
    digitalWrite(leds[i],LOW);
}

// Si el valor es MAYOR que 230...encendemos todos los leds
if(val >= 230){
    for(i = 0; i <= longArrayLEDS; i++){
        digitalWrite(leds[i],HIGH);
    }
}

Serial.println(val, DEC);
}
```

# CONEXIÓN DE ARDUINO CON PROCESSING

Este es un ejemplo muy sencillo de cómo enviar un **único** dato, a partir de la lectura analógica de un **Potenciómetro**, desde Arduino a Processing. Una vez que tengamos la señal disponible en Processing, la podremos usar para lo que nos interese, siempre teniendo en cuenta el rango de la variable, el tipo, ...

Veamos el sketch de **Arduino**:

```
int potPin = 0;           // Pot en el pin ANALÓGICO 0
int ledPin = 9;           // LED en el pin PWM 9
int valor;                // variable para almacenar el valor del pot
int valorMapeado;        // variable para almacenar el valor mapeado

void setup() {
  Serial.begin(9600); // Abrimos el puerto Serie de Arduino a una velocidad de 9600 baudios
  pinMode(ledPin, OUTPUT); // Declaramos el LED como SALIDA
}

void loop() {
// Leemos analógicamente el potenciómetro y lo guardamos en la variable valor
  valor = analogRead(potPin);

// Mapeamos el valor de la variable valor entre el rango original del potenciómetro (0 → 1023) y lo pasamos al rango del LED (0 → 255)
  valorMapeado = map(valor,0,1023,0,255);

// Escribimos analógicamente el valorMapeado en el LED
  analogWrite(ledPin,valorMapeado);

// Abrimos el Serial Monitor de Arduino y ahí podremos ver qué valores le vamos pasando al LED
  Serial.println(valorMapeado);
}
```

Antes de pasar al código de Processing, es importante tener en cuenta dos parámetros, para que la conexión Arduino – Processing funcione correctamente:

1 → En qué **Puerto** está conectado Arduino. En Windows, suele ser COM "y un número". En este ejemplo. Mi Arduino estaba conectado en el **COM4**.

2 → A qué **velocidad** está leyendo Arduino los datos. En este caso, a **9600 baudios** (es la medida estándar de apertura del Puerto Serie).

Veamos el sketch en **Processing**:

```
import processing.serial.*; // Importamos la librería que se ocupa de la lectura del puerto serie

Serial myPort; // Objeto Serial para inicializar el puerto serie
String buf = ""; // Cadena de texto, mas de una letra, en este caso esta vacia
int val; // Variable donde recibo el valor que me envia arduino
```

```

float diametro; // Diametro de mi circunferencia

void setup(){
    size(512,512);           // Tamaño de la ventana de visualización
    smooth();               // Suavizado de los bordes
    //imprimo solo una vez la lista de nombres de puertos series que tiene mi ordenador para ver donde
    //puedo leer
    println(Serial.list());
    // Puerto donde está conectado Arduino para poder leer los datos
    String portName = "COM4";
    // Asigno a mi objeto Serial el puerto que está conectado a Arduino y con la misma velocidad
    myPort = new Serial(this, portName, 9600);
}

void draw(){
    //si el puerto esta disponible...sirve para que no se sature la ventana de processing
    if ( myPort.available() > 0 ) {
        val = myPort.read(); //en la variable val, escribe lo que estás leyendo en el puerto
        if(val != 10) { //si val es distinto de 10, o sea...detecto donde cambio de linea con el return (10)
            buf += char(val); //...entonces...guardo el valor como una cadena de texto
        }else{
            diametro = float(buf); //y si no....el diametro vale 1023 menos la transformacion de buf, ahora
            //como float...
            buf = ""; //vacio la variable buf...este doble cambio de la variable es un truco para que
            //funcione bien en processing
        }
        //si no esta disponible...no hagas nada
    }

    background(255);           // Fondo de pantalla blanco
    noStroke();               // No dibujaremos Bordes
    fill(255,0,0);             // Relleno de color Rojo
    ellipse(width/2,height/2,diametro,diametro); // Circunferencia en mitad de la pantalla
    println(diametro);          // Imprimimos en el Puerto Serie el diámetro
}

```

Este es un ejemplo más complejo para enviar **múltiples** datos, a partir de las lecturas analógicas de dos **Potenciómetros** (que controlan dos Motores DC), desde Arduino a Processing. En este ejemplo, usaremos las lecturas de los potenciómetros para modificar el diámetro de una circunferencia y para variar su color de relleno.

Veamos el sketch de **Arduino**:

```

int valor1;           // variable para leer el pot 1
int valor2;           // variable para leer el pot 2
int valor_mapeado1; // variable mapear el pot 1
int valor_mapeado2; // variable mapear el pot 2

int motor1[] = { 9, 10}; // Array de Pins PWM para el motor1
int motor2[] = { 5, 6}; // Array de Pins PWM para el motor2

int pot1 = 0;          // Pot 1 al Pin 0 analogico

```

```
int pot2 = 1;           // Pot 2 al Pin 1 analogico

void setup() {
  Serial.begin(9600);
  pinMode(pot1, INPUT);      // Potenciómetros como ENTRADA
  pinMode(pot2, INPUT);
  // Declaramos una variable temporal y recorremos la array del Motor1 y del Motor2 como SALIDA
  int i;
  for(i = 0; i < 2; i++){
    pinMode(motor1[i], OUTPUT);
  }
  int t;
  for(t = 0; t < 2; t++){
    pinMode(motor2[t], OUTPUT);
  }
}

void loop() {
  valor1 = analogRead(pot1);    // leemos el pot1
  valor2 = analogRead(pot2);    // leemos el pot2
  // mapeamos los pots para adecuarlos al rango del motor
  valor_mapeado1 = map(valor1,0,1023,0,255);
  valor_mapeado2 = map(valor2,0,1023,0,255);

  // El motor1 AVANZA según la lectura mapeada del pot 1
  analogWrite(motor1[0], valor_mapeado1);
  analogWrite(motor1[1], 0);

  // El motor2 AVANZA según la lectura mapeada del pot 2
  analogWrite(motor2[0], valor_mapeado2);
  analogWrite(motor2[1], 0);

  //Serial.print("RPM Motor1 : ");

  // Imprimos un "0" al principio...ver comentarios en el sketch de Processing y los valores los separamos por COMAS
  Serial.print("0");
  Serial.print(",");
  Serial.print(valor_mapeado1);
  //Serial.print(",");
  //Serial.print("RPM Motor2 : ");
  Serial.print(",");
  Serial.println(valor_mapeado2);

  delay(20); // Retraso para no saturar el Puerto Serie
}
```

Veamos el sketch en **Processing**:

```
import processing.serial.*;

Serial puerto;          // Objeto Serial llamado port (donde leo los datos que envía Arduino)

int diametro;           // variable para modificar el diámetro
int rojoVerde;          // variable para modificar el color de relleno

void setup(){
```

```
size(600,480);           // Tamaño ventana visualizacion
smooth();                // Suavizado
noStroke();               // Sin bordes
background(0);            // Color de fondo Blanco
AbroPuertoSerie();
}

void draw(){
arduinoPuertoDisponible();
background(0);
fill(rojoVerde , 255-rojoVerde , 0);
ellipse(width/2 , height/2 , diametro , diametro);
}

////////////////// COMUNICACIÓN CON ARDUINO ///////////////////
void AbroPuertoSerie(){
// Imprimo una lista de todos los puertos serie disponibles
println(Serial.list());
// Abro el puerto serie de Arduino
puerto = new Serial(this, "COM4", 9600);
// No se leerá la función serialEvent() hasta que no aparezca un salto de línea!!!!
puerto.bufferUntil("\n");
}

void arduinoPuertoDisponible(){
// Mientras el puerto esté disponible...leo los datos de Arduino
while (puerto.available() > 0) {
  serialEvent(puerto.read());
}
}

void serialEvent(int Serial) {
// Almacenamos la String que nos manda arduino hasta el salto de línea
String inString = puerto.readStringUntil("\n");

// Si la string guardada NO es NULA ...
if (inString != null) {
  inString = trim(inString);    // Borramos de la string los espacios en blanco ...

// Dividimos la String por las COMAS y convertimos las subStrings en una array de números enteros
int[ ] sensores = int(split(inString, ","));

// Si la array tiene por lo menos 3 elementos. IMPORTANTE: el primer valor es el "0". Esto lo hago
// porque si el valor [0] es igual a 0 (null), genera problemas en las lecturas. Por eso no lo uso
// directamente!!!!!
if (sensores.length >=3) {
// Asignamos las distintas lecturas ordenadas a las variables que me interesan. Ver que no uso la
// posición sensores [0]!!!!!!!!!!!!!!
diametro = sensores[1];
rojoNegro = sensores[2];
}
print(diametro);
print(",");
println(rojoVerde);
}
}
```

# CONEXIÓN DE ARDUINO CON FIREFLY

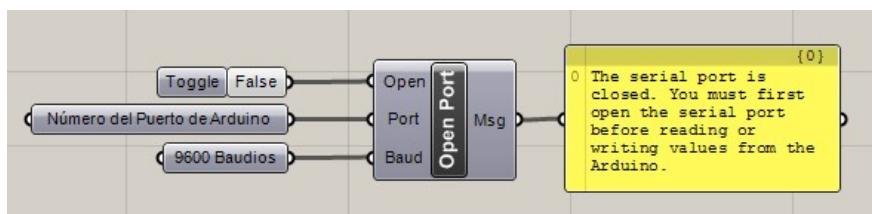
Hasta hace muy poco tiempo, la conexión entre Arduino y Grasshopper se realizaba a través de **scripts** realizados dentro de Grasshopper (GH), realizados por los propios autores de las definiciones. Sin embargo, hace unos meses, **Andy Payne**, del estudio LIFT Architects, y **Jason K. Johnson**, del estudio Future-Cities-Lab, lanzaron un plugging para GH llamado **FIREFLY**, que incorpora una serie de componentes para que realizar una conexión entre Arduino y GH sea más sencillo que anteriormente. Actualmente, está disponible la versión 1.003.

Link para descargar el plugin:

<http://www.fireflyexperiments.com/>

Las instrucciones para instalar este plugging las podeis encontrar en el pdf que hay adjunto al plugin, al igual que una pormenorizada descripción de las diferentes cajas, e incluso ejemplos. Sin embargo, voy a describir las cajas más importantes de este plugging para conectar Arduino y GH.

## 01\_Open Port



La caja Open Port tiene 3 Entradas y 1 Salida:

Entradas:

**Open** → Hay que conectarle un Toogle para poder cambiar de Falso a Verdadero la booleana que controla la apertura del Puerto.

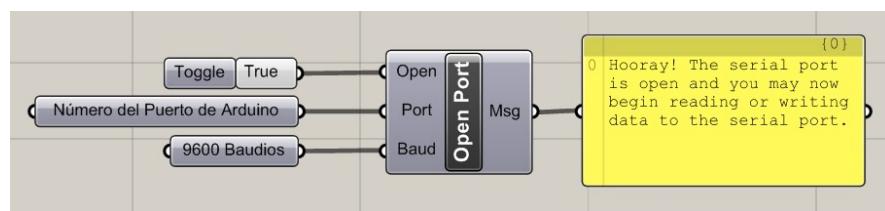
**Port** → Necesita un número entero para que pueda localizar qué puerto **COM#** está siendo utilizado por Arduino.

**Baud** → Precisa de otro número entero para que entienda a que velocidad está trabajando Arduino. Por defecto, esta entrada está a 9600 baudios.

Salida:

**Msg** → Con una nota, podemos comprobar que el puerto serie está **CERRADO**.

Si estuviera **ABIERTO**, nos aparecería lo siguiente:



## 02\_Generic Serial Read

Una vez que ya sabemos como abrir el Puerto Serie, vamos a empezar a recibir datos desde

Arduino con la pila llamada Generic Serial Read (se encuentra dentro del subMenú Utility). Para empezar a recibir datos, es **imprescindible**, tener operativa la pila de Open Port. Esta pila tiene 4 Entradas y 2 Salidas:

Entradas:

**Start** → Hay que conectarle un Toogle para poder cambiar de Falso a Verdadero la booleana que controla la recepción de los datos.

**Port** → Necesita un número entero para que pueda localizar qué puerto **COM#** está siendo utilizado por Arduino. Se puede ligar con la entrada de la pila de Open Port.

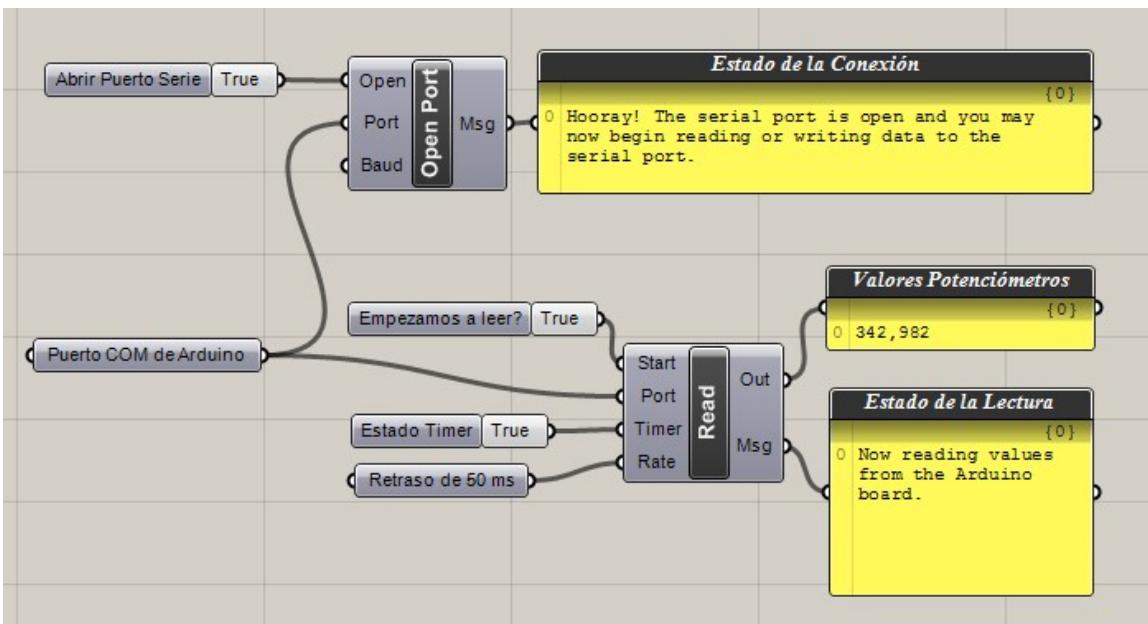
**Timer** → Hay que conectarle un Toogle para poder cambiar de Falso a Verdadero la booleana que controla el estado del Timer (Temporizador).

**Rate** → Precisa de un número entero para que controlar el tiempo (en milisegundos) de retardo con el que empezar cada nueva lectura de los sensores. Es aconsejable que sea el mismo que hemos usado en el delay () de Arduino.

Salidas:

**Out** → En esta salida podremos leer los datos que vamos recibiendo desde Arduino.

**Msg** → En ella, controlamos el estado de la lectura de datos desde Arduino.



Para la explicación del funcionamiento de esta pila, hemos usado un sketch en Arduino que manda la lectura de 2 potenciómetros.

Veamos el sketch de **Arduino**:

```

int potPin0 = 0; // Potenciómetro en el pin ANALÓGICO 0
int potPin1 = 1; // Potenciómetro en el pin ANALÓGICO 1
int valor0; // Variable para el pot 0
int valor1; // Variable para el pot 1
void setup() {
  Serial.begin(9600); // Abrimos el puerto Serie de Arduino a una velocidad de 9600 baudios
  
```

```

pinMode(potPin0, INPUT); // Declaramos los pots como ENTRADA
pinMode(potPin1, INPUT);
}
void loop() {
// Leemos analógicamente los pots y los guardamos en las variables valor
valor0 = analogRead(potPin0);
valor1 = analogRead(potPin1);
Serial.print(valor0);
Serial.print(",");
Serial.println(valor1);
delay(50);
}

```

### 03\_ Generic Serial Write

Instalamos la caja Open Port (con el COM# y la velocidad en Baudios) y ahora, vamos a empezar a enviar datos desde GH a Arduino con la pila llamada Generic Serial Write (se encuentra dentro del subMenú Utility). Para empezar a recibir datos, es **imprescindible**, tener operativa la pila de **Open Port**. Esta pila tiene 3 Entradas y 2 Salidas:

Entradas:

**Start** → Hay que conectarle un Toogle para poder cambiar de Falso a Verdadero la variable booleana que controla el estado del envío de datos.

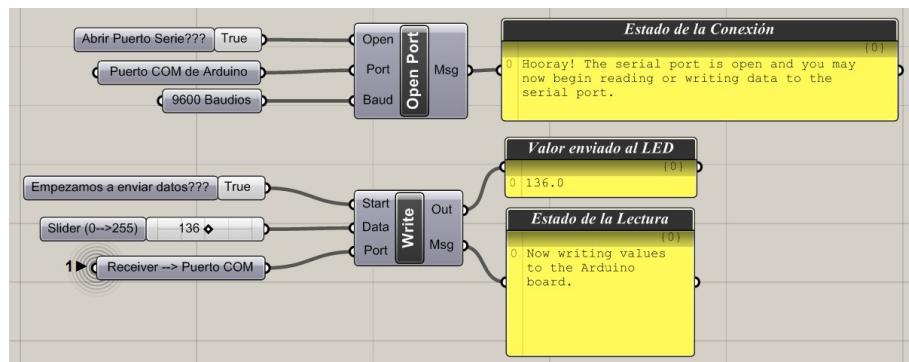
**Data** → String con el dato que vamos a enviar a Arduino.

**Port** → Necesita un número entero para que pueda localizar qué puerto **COM#** está siendo utilizado por Arduino. Se puede ligar con la entrada de la pila de Open Port.

Salidas:

**Out** → En esta salida podremos leer los datos que enviamos a Arduino.

**Msg** → En ella, controlamos el estado de la lectura de datos para Arduino.



Para la explicación del funcionamiento de esta pila, hemos usado, como base, el sketch [Firefly\\_DUEM\\_Firmata\\_1002.pde](#) para Arduino que podemos encontrar dentro de la carpeta [Firefly\\_Build 1.002 / Firefly Arduino Firmata 1.002 / Firefly\\_DUEM\\_Firmata\\_1002](#). Este sketch está realizado para usar la pila Duemilanove Write, pero retocándolo un poco y eliminando todo lo que no nos sirve, se puede utilizar para trabajar con esta pila.

Veamos el sketch de **Arduino**:

```
#define BUFFSIZE 128

char buffer[BUFFSIZE];
uint8_t bufferidx = 0;
uint16_t PWM6;      // Control del PWM6

char *parseptr;
char buffidx;
int ledPin = 6;     // Led al PWM6

void setup() {
  Serial.begin(9600);
}

void loop() {
  serialread(); // Llamamos a la Funcion serialRead
}

void serialread(){

  char c; // captamos un caracter del Puerto Serie
  if (Serial.available()) {
    c = Serial.read(); // Leemos ese caracter
    buffer[bufferidx] = c; // añadimos ese caracter al buffer

    if (c == '\n') { // Si leemos un SALTO DE LINEA...
      buffer[bufferidx+1] = 0; // ...terminamos de leer
      parseptr = buffer; // ...descargamos el buffer en la variable parseptr
      PWM6 = parsedecimal(parseptr); // Analizamos la variable parseptr

      // Analizamos las diferentes situaciones que podemos tener
      if (PWM6 == 1) { // Si es IGUAL a UNO ...
        digitalWrite(ledPin, HIGH); // Encendemos el LED
      }
      else if (PWM6 == 0){ // Si es IGUAL a CERO ...
        digitalWrite(ledPin, LOW); // Apagamos el LED
      }
      else{ // Si NO es ni UNO ni CERO ...
        analogWrite(ledPin, PWM6); // Escribimos analogicamente el valor en el LED
      }

      // Una vez terminada la lectura y el paso del valor al led, empezamos otra vez a leer
      bufferidx = 0; // Reiniciamos el buffer para la siguiente lectura
      return; // Devolvemos al buffer el CERO para no incrementar el Indice
    }
    // Hasta que no encontramos un SALTO DE LINEA, seguimos leyendo del buffer
    bufferidx++; // Incrementamos el indice para el siguiente caracter
    if (bufferidx == BUFFSIZE-1) { // Si llegamos al final del buffer, lo reseteamos por seguridad
      bufferidx = 0;
    }
  }
}

// Función para analizar los datos que enviamos desde Firefly
uint32_t parsedecimal(char *str) {
  uint32_t d = 0;
```

```

while (str[0] != 0) {
    if ((str[0] > '50') || (str[0] < '0'))
        return d;
    d *= 10;
    d += str[0] - '0';
    str++;
}
return d;
}

```

## 04\_ Binary Blink

Instalamos la caja Open Port (con el COM# y la velocidad en Baudios) y ahora, vamos a empezar a enviar datos desde GH a Arduino con la pila llamada Binary Blink (se encuentra dentro del subMenú Utility). Para empezar a recibir datos, es **imprescindible**, tener operativa la pila de Open Port. Esta pila tiene 3 Entradas y 1 Salida:

Entradas:

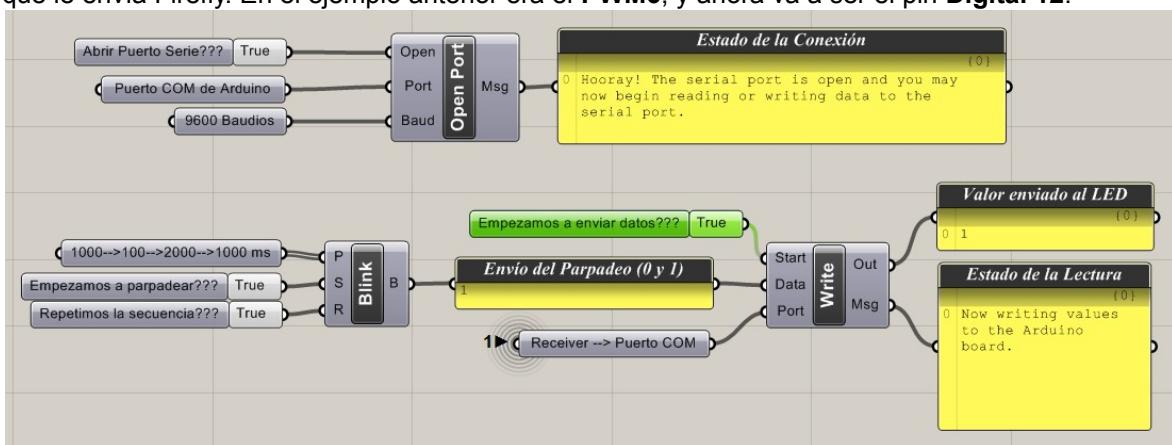
**P** → Hay que conectarle una lista de números enteros (en milisegundos) para crear la secuencia de parpadeos.

**S** → Hay que conectarle un Toogle para poder cambiar de Falso a Verdadero la booleana que controla el inicio del envío de datos para parpadear.

**R** → Hay que conectarle un Toogle para poder cambiar de Falso a Verdadero la booleana que controla si se repite o no la secuencia de parpadeo.

Salida:

**B** → La salida siempre nos dará o un **cero** o un **uno** para enviarlo a Arduino, según la secuencia de parpadeo determinada en la entrada “**P**”. Para la explicación del funcionamiento de esta pila, podemos volver a usar el sketch anterior, modificando solamente qué pin tiene que estar ahora a la escucha de los datos que le envía Firefly. En el ejemplo anterior era el **PWM6**, y ahora va a ser el pin **Digital 12**.



Veamos el sketch de Arduino:

```

#define BUFFSIZE 128

char buffer[BUFFSIZE];
uint8_t bufferidx = 0;
uint16_t DPin12; // Control del DIG 12

```

```
char *parseptr;
char buffidx;
int ledPin = 12; // Led al DIG 12

void setup() {
  pinMode(12, OUTPUT); // Led como SALIDA
  Serial.begin(9600);
}

void loop() {
  serialread(); // Llamamos a la Funcion serialRead
}

void serialread(){

  char c; // captamos un caracter del Puerto Serie
  if (Serial.available()) {
    c = Serial.read(); // Leemos ese caracter
    buffer[buffidx] = c; // añadimos ese caracter al buffer

    if (c == '\n') { // Si leemos un SALTO DE LINEA...
      buffer[buffidx+1] = 0; // ...terminamos de leer
      parseptr = buffer; // ...descargamos el buffer en la variable parseptr
      PWM6 = parsedecimal(parseptr); // Analizamos la variable parseptr

      // Analizamos las diferentes situaciones que podemos tener
      if (PWM6 == 1) { // Si es IGUAL a UNO ...
        digitalWrite(ledPin, HIGH); // Encendemos el LED
      }
      else if (PWM6 == 0){ // Si es IGUAL a CERO ...
        digitalWrite(ledPin, LOW); // Apagamos el LED
      }
      else{ // Si NO es ni UNO ni CERO ...
        analogWrite(ledPin, PWM6); // Escribimos analogicamente el valor en el LED
      }
    }

    // Una vez terminada la lectura y el paso del valor al led, empezamos otra vez a leer

    buffidx = 0; // Reiniciamos el buffer para la siguiente lectura
    return; // Devolvemos al buffer el CERO para no incrementar el Indice
  }
  // Hasta que no encontramos un SALTO DE LINEA, seguimos leyendo del buffer
  buffidx++; // Incrementamos el indice para el siguiente caracter
  if (buffidx == BUFFSIZE-1) { // Si llegamos al final del buffer, lo reseteamos por seguridad
    buffidx = 0;
  }
}

// Función para analizar los datos que envíamos desde Firefly
uint32_t parsedecimal(char *str) {
  uint32_t d = 0;

  while (str[0] != 0) {
    if ((str[0] > '50') || (str[0] < '0'))
      return d;
    d *= 10;
    str++;
  }
}
```

```

d += str[0] - '0';
str++;
}
return d;
}

```

## 05\_ Buffer

Instalamos la caja Open Port (con el COM# y la velocidad en Baudios) y ahora, vamos a empezar a enviar datos desde GH a Arduino con la pila llamada Buffer (se encuentra dentro del subMenú Utility). Para empezar a recibir datos, es **imprescindible**, tener operativa la pila de Open Port. Esta pila tiene 2 Entradas y 3 Salidas:

Entradas:

**V** → Son los valores enteros que nos pasa la pila “Write”.

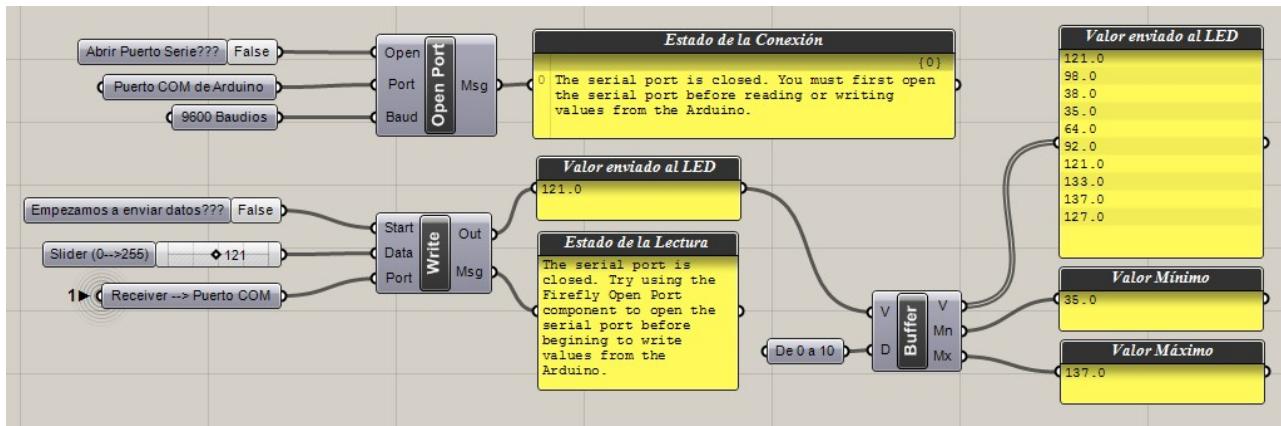
**D** → Hay que proporcionarle un dominio para el tamaño que queremos que tenga la tabla de datos. En este ejemplo es de 0 a 10.

Salidas:

**V** → Tabla de datos con los valores almacenados.

**Mn** → Valor Mínimo almacenado en la tabla.

**Mx** → Valor Máximo almacenado en la tabla.



Para la explicación del funcionamiento de esta pila, podemos volver a usar el sketch anterior, modificando solamente qué pin tiene que estar ahora a la escucha de los datos que le envía Firefly. Esta caja se puede usar tanto para leer datos desde arduino, como para enviar datos desde Firefly. Yo en este ejemplo la he usado para enviar datos a un LED analógicamente.

Veamos el sketch de Arduino:

```

#define BUFFSIZE 128

char buffer[BUFFSIZE];
uint8_t bufferidx = 0;
uint16_t PWM6; // Control del PWM6

char *parseptr;
char buffidx;

```

```
int ledPin = 6;      // Led al PWM6

void setup() {
  Serial.begin(9600);
}

void loop() {
  serialread(); // Llamamos a la Funcion serialRead
}

void serialread(){

  char c; // captamos un caracter del Puerto Serie
  if (Serial.available()) {
    c = Serial.read(); // Leemos ese caracter
    buffer[bufferidx] = c; // añadimos ese caracter al buffer

    if (c == '\n') { // Si leemos un SALTO DE LINEA...
      buffer[bufferidx+1] = 0; // ...terminamos de leer
      parseptr = buffer; // ...descargamos el buffer en la variable parseptr
      PWM6 = parsedecimal(parseptr); // Analizamos la variable parseptr
      // Analizamos las diferentes situaciones que podemos tener
      if (PWM6 == 1){ // Si es IGUAL a UNO ...
        digitalWrite(ledPin, HIGH); // Encendemos el LED
      }
      else if (PWM6 == 0){ // Si es IGUAL a CERO ...
        digitalWrite(ledPin, LOW); // Apagamos el LED
      }
      else{ // Si NO es ni UNO ni CERO ...
        analogWrite(ledPin, PWM6); // Escribimos analogicamente el valor en el LED
      }
    }

    // Una vez terminada la lectura y el paso del valor al led, empezamos otra vez a leer

    bufferidx = 0; // Reiniciamos el buffer para la siguiente lectura
    return; // Devolvemos al buffer el CERO para no incrementar el Indice
  }
  // Hasta que no encontramos un SALTO DE LINEA, seguimos leyendo del buffer
  bufferidx++; // Incrementamos el indice para el siguiente caracter
  if (bufferidx == BUFFSIZE-1){ // Si llegamos al final del buffer, lo reseteamos por seguridad
    bufferidx = 0;
  }
}

// Función para analizar los datos que enviamos desde GH
uint32_t parsedecimal(char *str) {
  uint32_t d = 0;

  while (str[0] != 0) {
    if ((str[0] > '5') || (str[0] < '0'))
      return d;
    d *= 10;
    d += str[0] - '0';
    str++;
  }
  return d;
}
```

## 06\_ Constrain

Instalamos la caja Open Port (con el COM# y la velocidad en Baudios) y ahora, vamos a empezar a enviar datos desde Firefly a Arduino con la pila llamada Constrain (se encuentra dentro del subMenú Utility). Para empezar a recibir datos, es **imprescindible**, tener operativa la pila de Open Port. Esta pila tiene 2 Entradas y 1 Salida:

Entradas:

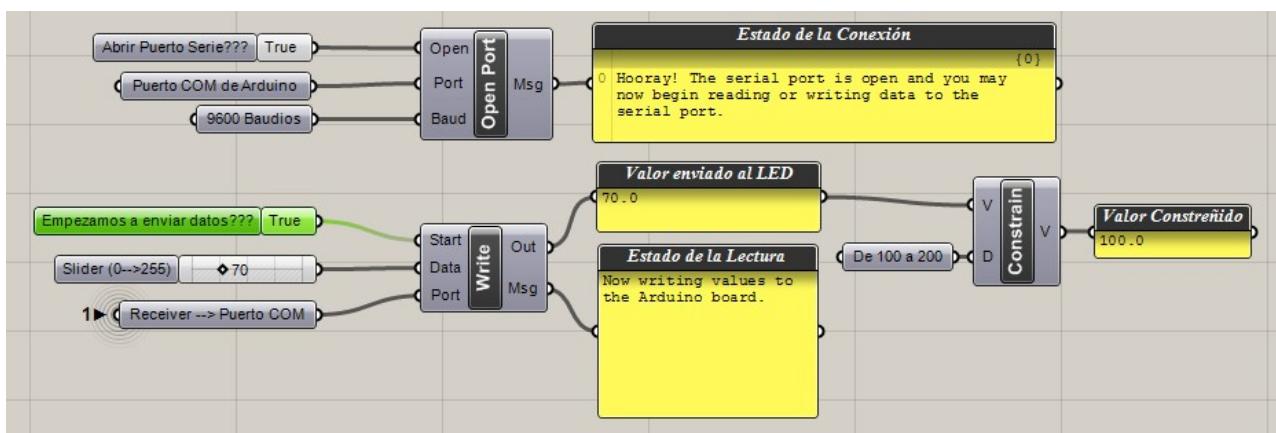
**V** → Son los valores enteros que nos pasa la pila “Write”.

**D** → Hay que proporcionarle un dominio para determinar los límites para el valor constreñido. En este ejemplo es de 100 a 200.

Salida:

**V** → Valor constreñido a los límites que le hemos puesto.

Observad, que el valor que le envío al LED es 70, pero debido a los límites que hemos puesto, el valor constreñido nunca puede ser MENOR que 100 ni MAYOR que 200. Esta caja puede ser muy útil para remapear datos tanto si los recibe Firefly, como si los recibe Arduino.



Para la explicación del funcionamiento de esta pila, podemos volver a usar el sketch anterior, modificando solamente qué pin tiene que estar ahora a la escucha de los datos que le envía Firefly. Esta caja se puede usar tanto para leer datos desde arduino, como para enviar datos desde Firefly. Yo en este ejemplo la he usado para enviar datos a un LED analógicamente.

Veamos el sketch de **Arduino**:

```
#define BUFFSIZE 128

char buffer[BUFFSIZE];
uint8_t bufferidx = 0;
uint16_t PWM6;      // Control del PWM6

char *parseptr;
char buffidx;
int ledPin = 6;     // Led al PWM6

void setup() {
```

```
Serial.begin(9600);
}

void loop() {
    serialread(); // Llamamos a la Funcion serialRead
}

void serialread(){

char c; // captamos un caracter del Puerto Serie
if (Serial.available()) {
    c = Serial.read(); // Leemos ese caracter
    buffer[bufferidx] = c; // añadimos ese caracter al buffer

    if (c == '\n') { // Si leemos un SALTO DE LINEA...
        buffer[bufferidx+1] = 0; // ...terminamos de leer
        parseptr = buffer; // ...descargamos el buffer en la variable parseptr
        PWM6 = parsedecimal(parseptr); // Analizamos la variable parseptr

        // Analizamos las diferentes situaciones que podemos tener
        if (PWM6 == 1) { // Si es IGUAL a UNO ...
            digitalWrite(ledPin, HIGH); // Encendemos el LED
        }
        else if (PWM6 == 0){ // Si es IGUAL a CERO ...
            digitalWrite(ledPin, LOW); // Apagamos el LED
        }
        else{ // Si NO es ni UNO ni CERO ...
            analogWrite(ledPin, PWM6); // Escribimos analogicamente el valor en el LED
        }
    }

    // Una vez terminada la lectura y el paso del valor al led, empezamos otra vez a leer

    bufferidx = 0; // Reiniciamos el buffer para la siguiente lectura
    return; // Devolvemos al buffer el CERO para no incrementar el Indice
}
// Hasta que no encontramos un SALTO DE LINEA, seguimos leyendo del buffer
bufferidx++; // Incrementamos el indice para el siguiente caracter
if (bufferidx == BUFFSIZE-1) { // Si llegamos al final del buffer, lo reseteamos por seguridad
    bufferidx = 0;
}
}

// Función para analizar los datos que enviamos desde GH
uint32_t parsedecimal(char *str) {
    uint32_t d = 0;

    while (str[0] != 0) {
        if ((str[0] > '5') || (str[0] < '0'))
            return d;
        d *= 10;
        d += str[0] - '0';
        str++;
    }
    return d;
}
```

## 07\_ Fader1

Instalamos la caja Open Port (con el COM# y la velocidad en Baudios) y ahora, vamos a empezar a enviar datos desde Firefly a Arduino con la pila llamada Fader1 (se encuentra dentro del subMenú Utility). Para empezar a recibir datos, es **imprescindible**, tener operativa la pila de **Open Port**. Esta pila tiene 4 Entradas y 1 Salida:

Entradas:

**V1** → Primer límite para el desvanecimiento del Led.

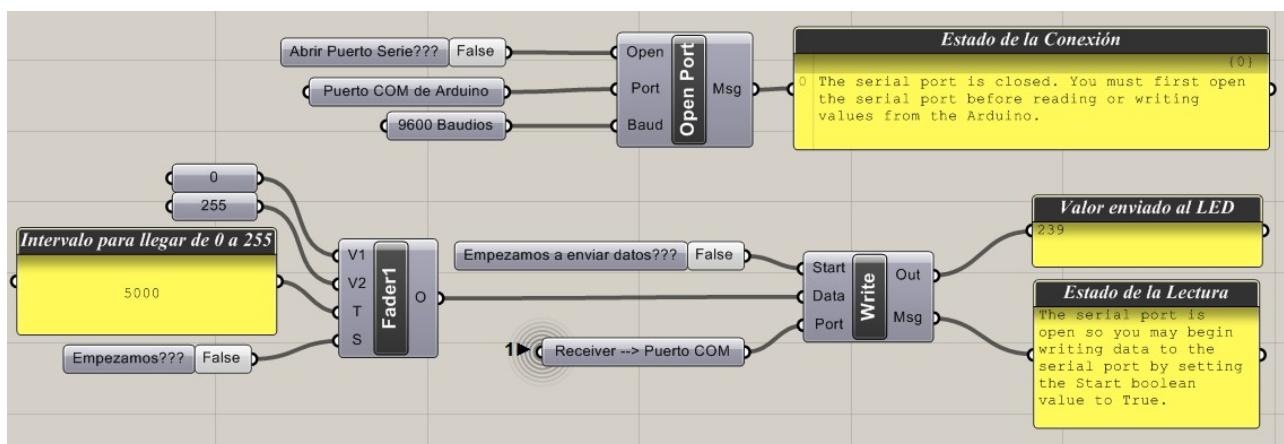
**V2** → Segundo límite para el desvanecimiento del Led.

**T** → Intervalo en milisegundos que tardará en hacer el desvanecimiento del Led desde el valor mínimo al máximo.

**S** → Hay que conectarle un Togle para poder cambiar de Falso a Verdadero la booleana que controla el inicio del desvanecimiento del Led.

Salida:

**O** → Valores de salida.



Para la explicación del funcionamiento de esta pila, podemos volver a usar el sketch anterior, modificando solamente qué pin tiene que estar ahora a la escucha de los datos que le envía Firefly. Esta caja se puede usar tanto para leer datos desde arduino, como para enviar datos desde Firefly. Yo en este ejemplo la he usado para enviar datos a un LED analógicamente.

Veamos el sketch de Arduino:

```
#define BUFFSIZE 128

char buffer[BUFFSIZE];
uint8_t bufferidx = 0;
uint16_t PWM6;      // Control del PWM6

char *parseptr;
char buffidx;
int ledPin = 6;     // Led al PWM6
void setup() {
  Serial.begin(9600);
}
```

```
void loop() {
    serialread(); // Llamamos a la Funcion serialRead
}

void serialread(){

    char c; // captamos un caracter del Puerto Serie
    if (Serial.available()) {
        c = Serial.read(); // Leemos ese caracter
        buffer[bufferidx] = c; // añadimos ese caracter al buffer

        if (c == '\n') { // Si leemos un SALTO DE LINEA...
            buffer[bufferidx+1] = 0; // ...terminamos de leer
            parseptr = buffer; // ...descargamos el buffer en la variable parseptr
            PWM6 = parsedecimal(parseptr); // Analizamos la variable parseptr

            // Analizamos las diferentes situaciones que podemos tener
            if (PWM6 == 1) { // Si es IGUAL a UNO ...
                digitalWrite(ledPin, HIGH); // Encendemos el LED
            }
            else if (PWM6 == 0){ // Si es IGUAL a CERO ...
                digitalWrite(ledPin, LOW); // Apagamos el LED
            }
            else{ // Si NO es ni UNO ni CERO ...
                analogWrite(ledPin, PWM6); // Escribimos analogicamente el valor en el LED
            }

            // Una vez terminada la lectura y el paso del valor al led, empezamos otra vez a leer
            bufferidx = 0; // Reiniciamos el buffer para la siguiente lectura
            return; // Devolvemos al buffer el CERO para no incrementar el Indice
        }
        // Hasta que no encontramos un SALTO DE LINEA, seguimos leyendo del buffer
        bufferidx++; // Incrementamos el indice para el siguiente caracter
        if (bufferidx == BUFFSIZE-1) { // Si llegamos al final del buffer, lo reseteamos por seguridad
            bufferidx = 0;
        }
    }
}

// Función para analizar los datos que envíamos desde Firefly
uint32_t parsedecimal(char *str) {
    uint32_t d = 0;

    while (str[0] != 0) {
        if ((str[0] > '50') || (str[0] < '0'))
            return d;
        d *= 10;
        d += str[0] - '0';
        str++;
    }
    return d;
}
```

## 08\_Fader2

Instalamos la caja Open Port (con el COM# y la velocidad en Baudios) y ahora, vamos a empezar a enviar datos desde Firefly a Arduino con la pila llamada Fader2 (se encuentra dentro del subMenú Utility). Para empezar a recibir datos, es **imprescindible**, tener operativa la pila de Open Port. Esta pila tiene 5 Entradas y 1 Salida:

Entradas:

**V1** → Primer límite para el desvanecimiento del Led.

**V2** → Segundo límite para el desvanecimiento del Led.

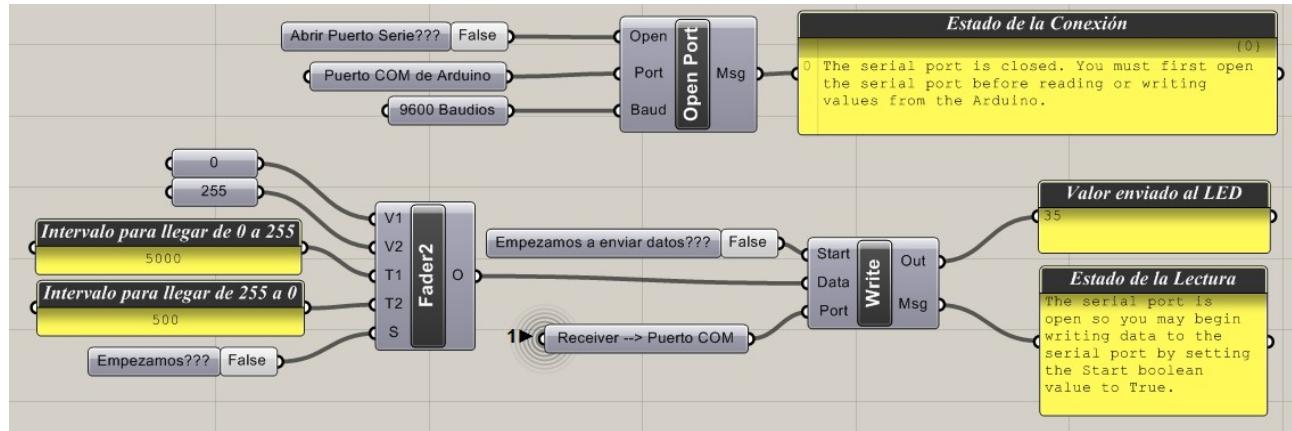
**T1** → Intervalo en milisegundos que tardará en hacer el desvanecimiento del Led desde el valor mínimo al máximo.

**T2** → Intervalo en milisegundos que tardará en hacer el desvanecimiento del Led desde el valor máximo al mínimo.

**S** → Hay que conectarle un Toogle para poder cambiar de Falso a Verdadero la booleana que controla el inicio del desvanecimiento del Led.

Salida:

**O** → Valores de salida.



El código utilizado en Arduino es idéntico al caso de la pila Fader1. Estas dos pilas pueden ser muy útiles para controlar Motores DC, controlando sus tiempos de apagado y encendido o su sentido de giro.

## 09\_ Smoothing

Instalamos la caja Open Port (con el COM# y la velocidad en Baudios) y ahora, vamos a empezar a enviar datos desde Arduino a Firefly con la pila llamada Smoothing (se encuentra dentro del subMenú Utility). Para empezar a recibir datos, es **imprescindible**, tener operativa la pila de **Open Port**. Esta pila tiene 2 Entradas y 1 Salida:

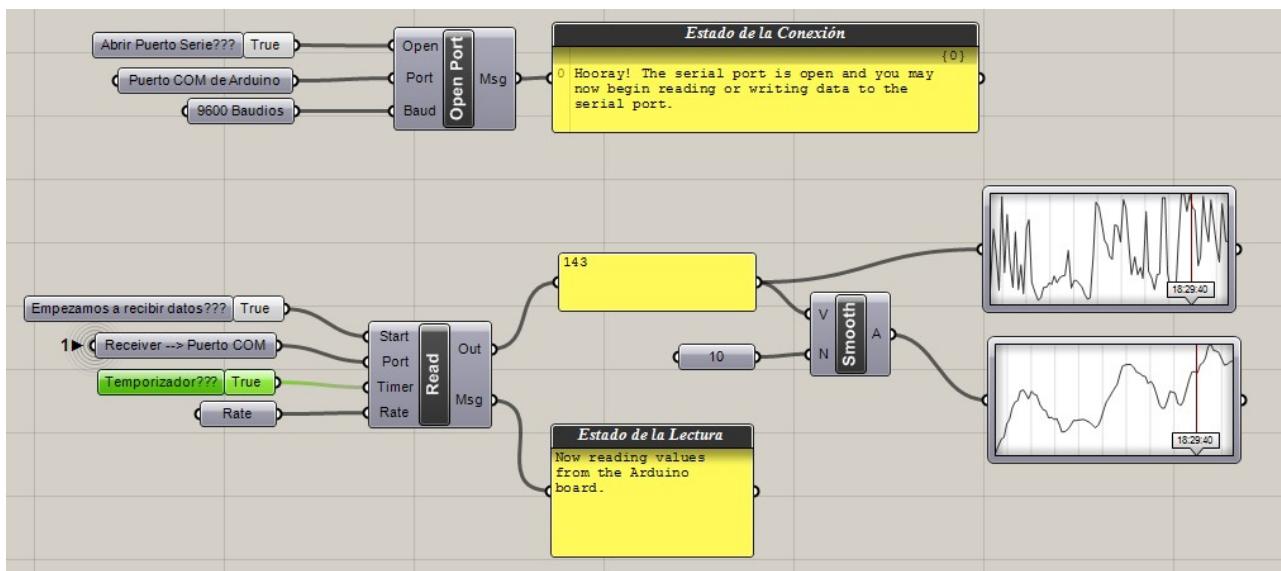
Entradas:

**V** → Valores de Entrada que provienen de la caja “Write”. En este caso, es la lectura de una Fotoresistencia (LDR).

**N** → Número de veces que suaviza las lecturas. Este parámetro funciona bien entre 10 y 20.

Salida:

**A** → Valores de Salida, una vez suavizados.



En la gráfica de arriba, se pueden ver los datos de salida directamente leídos desde Arduino, donde se pueden apreciar los picos de lectura. En la gráfica de abajo, vemos los datos suavizados gracias a la pila Smoothing. El rango de la LDR se ha mapeado en Arduino a un rango entre 0 y 255, para manipular el Led.

Para la explicación del funcionamiento de esta pila, he usado un sketch para leer una **LDR** (sensor analógico con rango entre 0 y 1023) y controlar esa lectura para controlar un LED analógicamente.

Veamos el sketch de **Arduino**:

```

int LDR_pin = 0;           // LDR en el pin Analógico 0
int LED_pin = 6;           // LED en el pin Digital PWM 6
int LDR_val;               // Variable que almacena la lectura del LDR
int LDR_mapeado;           // Variable mapeada para encender el LED

void setup() {
  Serial.begin(9600);      // Abrimos el Puerto Serie a 9600 baudios
  pinMode(LED_pin,OUTPUT);
  pinMode(LDR_pin,INPUT);
}

void loop(){
  
```

```

LDR_val = analogRead(LDR_pin);           // leemos el valor del LDR
LDR_mapeado = map(LDR_val,0,1023,0,255); // limitamos el valor del LDR a [0-255] para el LED
analogWrite(LED_pin,LDR_mapeado);       // Le pasamos al LED el valor mapeado
Serial.println(LDR_mapeado);           // escribimos en el Puerto Serie el valor con que se enciende el LED
delay(100);
}

```

## 10\_ Data Log

Instalamos la caja Open Port (con el COM# y la velocidad en Baudios) y ahora, vamos a empezar a enviar datos desde Arduino a Firefly con la pila llamada Data Log (se encuentra dentro del subMenú Utility). Para empezar a recibir datos, es **imprescindible**, tener operativa la pila de **Open Port**. Esta pila tiene 3 Entradas y 1 Salida:

Entradas:

**V** → Valores de Entrada. En este caso, los valores vienen de suavizarlos con la pila Smoothing

**R** → Hay que conectarle un Toogle para poder cambiar de Falso a Verdadero la booleana que controla el reseteo del guardado de datos.

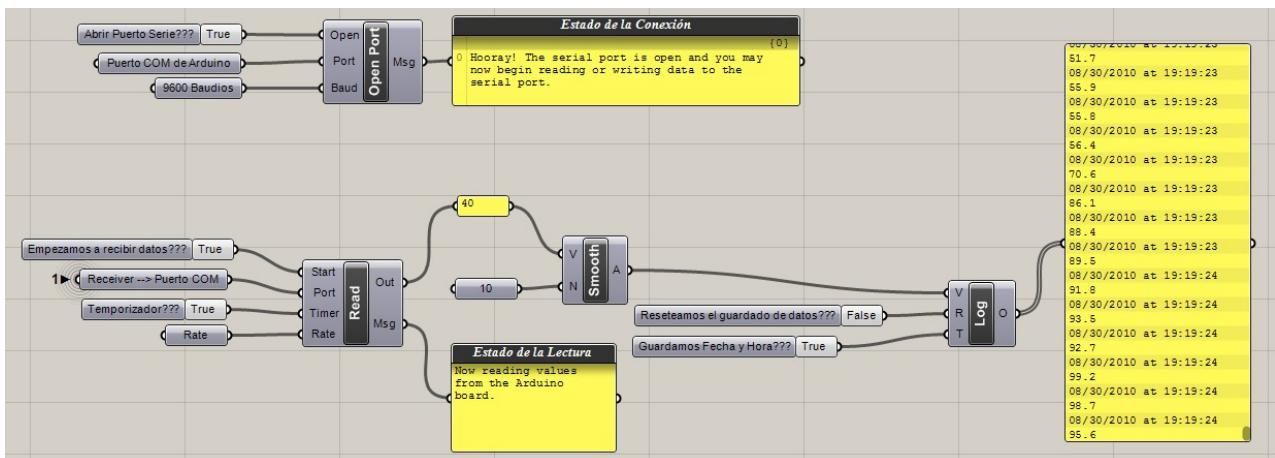
**T** → Hay que conectarle un Toogle para poder cambiar de Falso a Verdadero la booleana que controla si queremos que a la hora de guardar los datos, se incluya la Fecha y la Hora.

Salida:

**T** → Valores de Salida.

La pila Data Log puede sernos muy útil para almacenar lecturas de datos o para reproducir los datos que hemos recogido de una determinada acción y cargarlos para simular la misma acción. De ahí la entrada que nos permite guardar o no la fecha y hora de recepción de los datos.

Para la explicación del funcionamiento de esta pila, he usado el sketch para leer una **LDR** (sensor analógico con rango entre 0 y 1023) del ejemplo anterior de la pila Smoothing.



## 11\_ Wii Nunchuck

Instalamos la caja Open Port (con el COM# y la velocidad en Baudios) y ahora, vamos a empezar a enviar datos desde Arduino a Firefly con la pila llamada Wii Nunchuck (se encuentra dentro del subMenú Ot). Para empezar a recibir datos, es **imprescindible**, tener operativa la pila de **Open Port**. Esta pila tiene 4 Entradas y 8 Salidas:

Entradas:

**Start** → Hay que conectarle un Toogle para poder cambiar de Falso a Verdadero la booleana que controla la recepción de los datos.

**Port** → Necesita un número entero para que pueda localizar qué puerto **COM#** está siendo utilizado por Arduino. Se puede ligar con la entrada de la pila de Open Port.

**Timer** → Hay que conectarle un Toogle para poder cambiar de Falso a Verdadero la booleana que controla el estado del Timer (Temporizador).

**Rate** → Precisa de un número entero para que controlar el tiempo (en milisegundos) de retardo con el que empezar cada nueva lectura de los sensores. Es aconsejable que sea el mismo que hemos usado en el delay () de Arduino.

Salidas:

**AccX** → Devuelve el valor del Acelerómetro X.

**AccY** → Devuelve el valor del Acelerómetro Y.

**AccZ** → Devuelve el valor del Acelerómetro Z.

**JoyX** → Devuelve el valor del Joystick en el eje X.

**JoyY** → Devuelve el valor del Joystick en el eje Y.

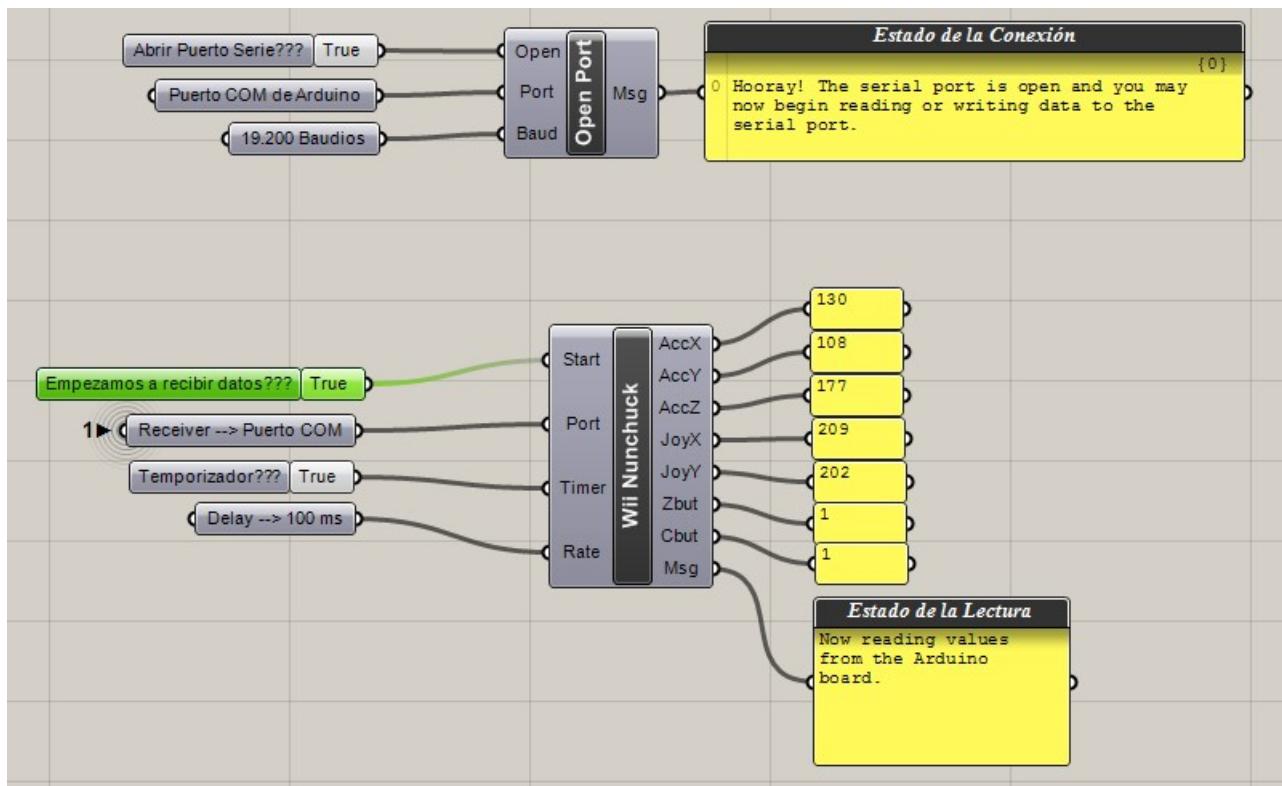
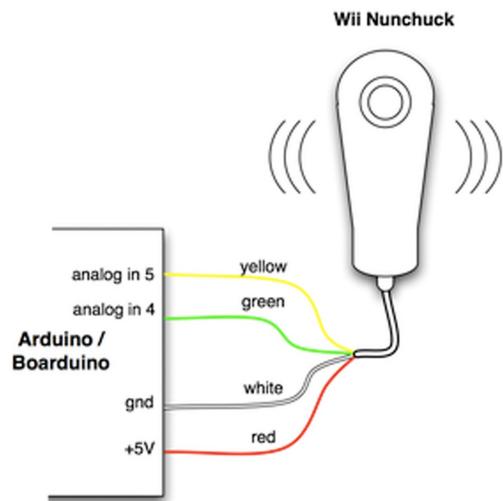
**Zbut** → Devuelve el valor del estado del Botón Z (por defecto es 0).

**Cbut** → Devuelve el valor del estado del Botón C (por defecto es 0).

**Msg** → En ella, controlamos el estado de la lectura de datos desde Arduino.

Esta pila nos puede ser muy útil, para capturar la lectura de los 3 acelerómetros del Nunchuck para incorporar la gravedad en una determinada instalación.

Lo primero que tenemos que tener claro, es cómo se comunica el Nunchuck con Arduino. Si cortamos el cable del Nunchuck y lo pelamos, observaremos que hay 4 cables: Blanco, Rojo, Verde y Amarillo (normalmente estos son los colores, aunque si nuestro Nunchuck no es original, sino una imitación, los colores pueden estar alterados). Ahora identificaremos los cables y veremos como se usan según nuestro sketch en Arduino.



Veamos ahora el código de **Arduino**:

```
#include <Wire.h>
#include <string.h>

#undef int
#include <stdio.h>
uint8_t outbuf[6]; // Array para almacenar los datos de salida
int cnt = 0; // Contador
```

```
void setup () {
    Serial.begin(19200);      // abrimos el puerto serie a 19200 baudios
    Wire.begin ();            // juntamos el bus de datos i2c con la direccion 0x52
    nunchuck_init ();         // Inicializamos el envío de datos del Nunchuck
}

void loop (){
    Wire.requestFrom (0x52, 6); // pide datos del nunchuck al dispositivo 0x52, y se recibirán 6 bytes
    // similar al Serial.available() nos indica que se estan recibiendo datos por el puerto.
    while (Wire.available ()) {
        outbuf[cnt] = nunchuk_decode_byte (Wire.receive());           // recibe el byte como un entero
        cnt++;                                                        // incrementamos el contador
    }

    // Si recibimos los 6 bytes, entonces los imprimimos
    if (cnt >= 5) {
        print();          // llamamos a la funcion print
    }

    cnt = 0;              // volvemos a dejar el contador en 0
    send_zero();          // manda el pedido para recibir mas bytes
    delay (100);
}
```

#### ////////// FUNCIONES AUXILIARES ///////////

```
void nunchuck_init () {
    Wire.beginTransmission (0x52);      // transmite al dispositivo 0x52
    Wire.send (0x40);                  // envia la direccion de la memoria
    Wire.send (0x00);                  // manda un cero
    Wire.endTransmission ();           // deja de transmitir
}

void send_zero () {
    Wire.beginTransmission (0x52);      // transmite al dispositivo 0x52
    Wire.send (0x00);                  // manda 1 byte, un cero
    Wire.endTransmission ();           // deja de transmitir
}

// Imprime la informacion que recibimos La informacion de los acelerometros que
// tiene una longitud de 10 bits. Entonces leemos 8 bits y le agregamos los
// ultimos 2 bits. Esto es por lo que se multiplican 2 * 2.
// Los bits que agregamos vienen en el 6to Byte que recibimos.

void print () {
    int joy_x_axis = outbuf[0];
    int joy_y_axis = outbuf[1];
    int accel_x_axis = outbuf[2] * 2 * 2;
    int accel_y_axis = outbuf[3] * 2 * 2;
    int accel_z_axis = outbuf[4] * 2 * 2;

    int z_button = 0;
    int c_button = 0;

    // byte outbuf[5] contiene los bits de los botones C y Z
    // tambien contiene los ultimos bits significativos de los acelerometros
    // por lo que tenemos que analizar cada bit del byte outbuf[5]
```

```
if ((outbuf[5] >> 0) & 1) {  
    z_button = 1;  
}  
if ((outbuf[5] >> 1) & 1) {  
    c_button = 1;  
}  
if ((outbuf[5] >> 2) & 1) {  
    accel_x_axis += 2;  
}  
if ((outbuf[5] >> 3) & 1) {  
    accel_x_axis += 1;  
}  
if ((outbuf[5] >> 4) & 1) {  
    accel_y_axis += 2;  
}  
if ((outbuf[5] >> 5) & 1) {  
    accel_y_axis += 1;  
}  
if ((outbuf[5] >> 6) & 1) {  
    accel_z_axis += 2;  
}  
if ((outbuf[5] >> 7) & 1) {  
    accel_z_axis += 1;  
}  
  
Serial.print("joystick Eje X = ");  
Serial.print(joy_x_axis, DEC);  
Serial.print("\t");  
  
Serial.print("joystick Eje Y = ");  
Serial.print(joy_y_axis, DEC);  
Serial.print("\t");  
  
Serial.print("acel. Eje X = ");  
Serial.print(accel_x_axis, DEC);  
Serial.print("\t");  
  
Serial.print("acel. Eje Y = ");  
Serial.print(accel_y_axis, DEC);  
Serial.print("\t");  
  
Serial.print("acel. Eje Z = ");  
Serial.print(accel_z_axis, DEC);  
Serial.print("\t");  
Serial.print("Boton Z = ");  
Serial.print(z_button, DEC);  
Serial.print("\t");  
  
Serial.print("Boton C = ");  
Serial.print(c_button, DEC);  
Serial.print("\t");  
  
Serial.println();  
}  
char nunchuk_decode_byte (char x) {  
    x = (x ^ 0x17) + 0x17;  
    return x;  
}
```

## 12\_ Pachube Read

Con esta pila, no tenemos que conectar Arduino, ni abrir Puerto Serie, ni nada. Trabaja directamente con **direcciones de Internet**. Esta pila tiene 2 Entradas y 8 Salidas:

Entradas:

**Feed** → Dirección de Internet de Pachube.

**API Key** → Para poder trabajar con esta pila, es necesario solicitarla a la web Pachube. Un API Key es como una contraseña para identificarnos.

Salidas:

**csv** → Nos proporciona un archivo .csv (comma-separated values) siempre y cuando la dirección con la que estemos trabajando nos lo pueda proporcionar.

**Msg** → Mensaje sobre la dirección a la que estamos conectados, con la fecha y hora.

**ID** → Identificación XML de cada elemento.

**Tag** → Etiqueta XML de cada elemento.

**Min** → Valores Mínimos.

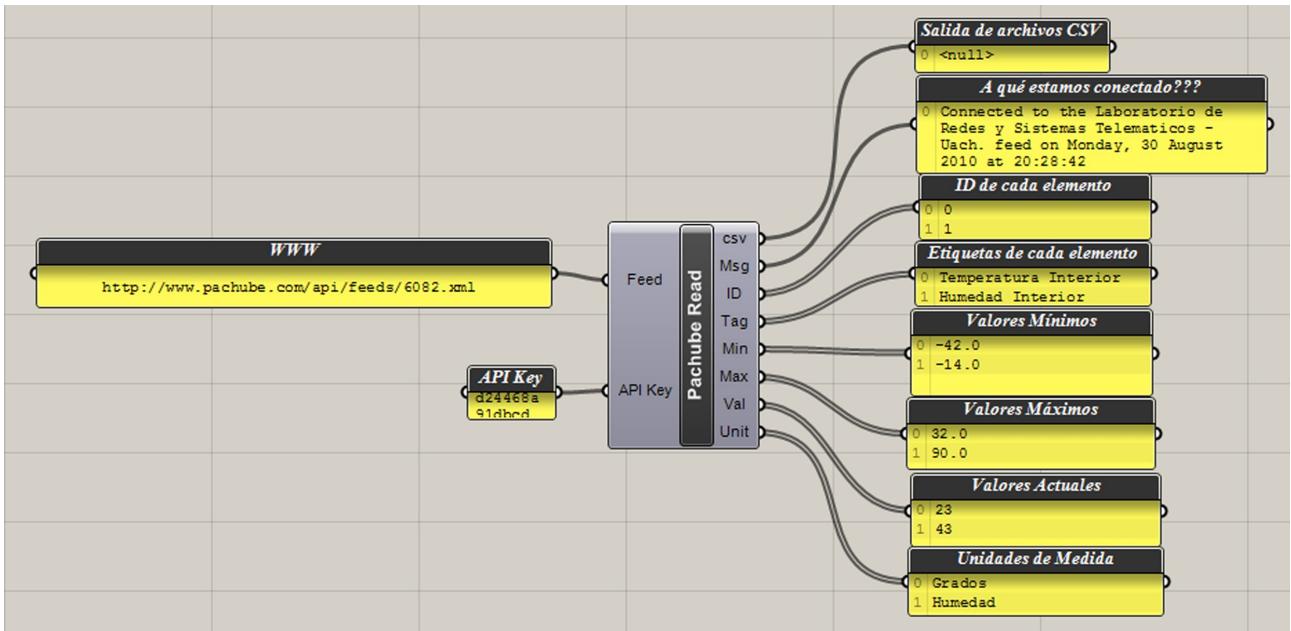
**Max** → Valores Máximos.

**Val** → Valores actuales.

**Unit** → Unidades de medida de los datos.

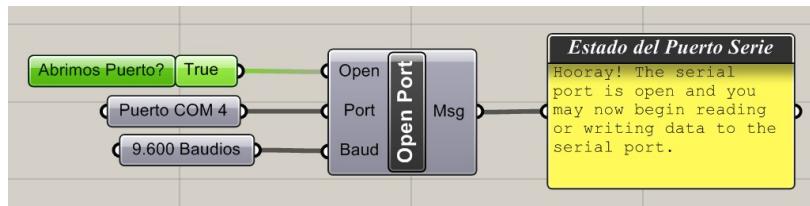
Esta pila nos puede servir para trabajar con datos desde Internet, monitorizando cualquier evento (Concentración de CO<sub>2</sub>, Temperaturas, Tráfico,...). En la web de Pachube podemos encontrar tutoriales sobre cómo trabajar con sus elementos.

En este ejemplo, he conectando Firefly con Pachube para leer datos via web desde el Laboratorio de Redes y Sistemas Telemáticos de la Facultad de Ciencias de la Ingeniería (Campus Miraflores, Universidad Austral de Chile).



Para terminar la parte de cómo conectar Arduino con Firefly, vamos a explicar el ejemplo que aparece al final del libro, pero metiendo algo más avanzado. Vamos a empezar con el ejemplo de recibir datos de Firefly en Arduino y controlar un **servomotor**, un Led **RGB**, tres Leds convencionales digitalmente y un led convencional analógicamente.

Primero, abriremos el Puerto Serie como hemos visto en todos los ejemplos anteriores, con la pila Open Port (nos aseguraremos que el puerto COM# y la velocidad de comunicación de Arduino, es la misma que en Firefly).



Una vez que ya podemos abrir el puerto Serie, vamos a trabajar con la pila **Duemilanove Write**. Observamos que tiene 11 Entradas y 2 Salidas:

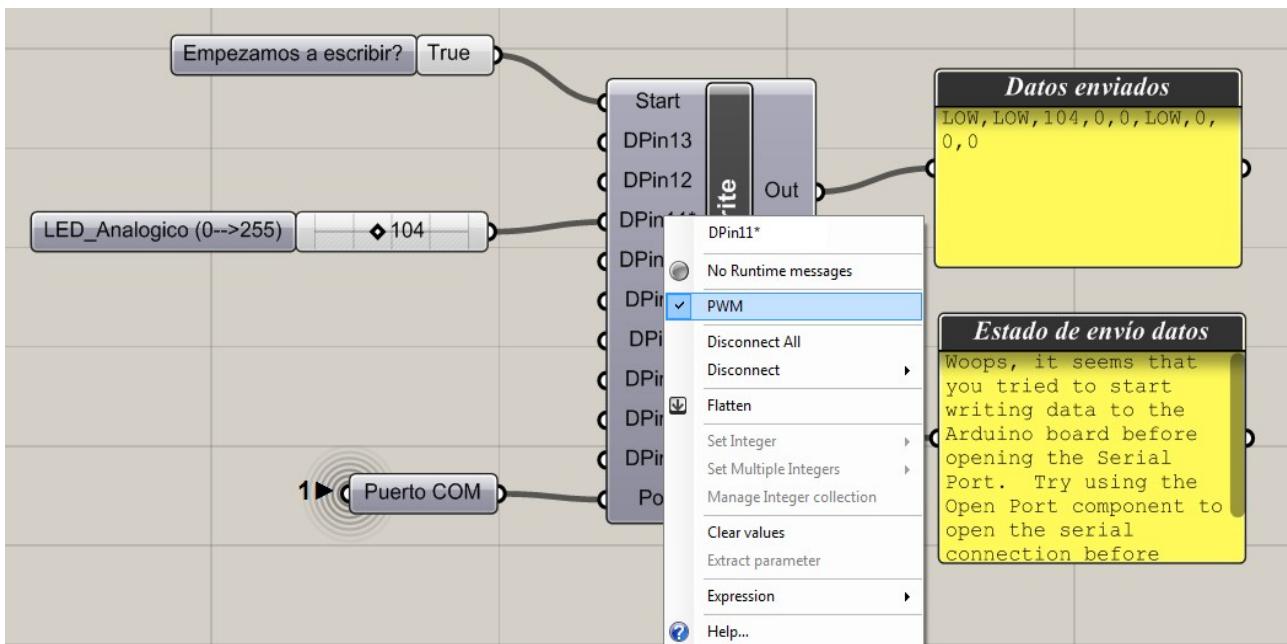
Entradas:

**Start** → Hay que conectarle un Toogle para poder cambiar de Falso a Verdadero la booleana que controla el estado de la Escritura a través del Puerto Serie.

**DPin13** → Necesita un número entero para controlar el pin Digital de Arduino nº 13. Le hemos conectado un slider de que sólo admite dos números o CERO o UNO, es decir, apagado o encendido.

**DPin12** → Necesita un número entero para controlar el pin Digital de Arduino nº 12. Trabaja igual que el Dpin13.

**DPin11\*** → Los pines que aparecen con un **asterisco (\*)**, nos indican que pueden trabajar como pines **Digitales** o como pines **PWM** (admiten un rango de lectura). Para que Firefly entienda cómo debe trabajar, debemos pinchar con el botón **derecho** sobre el pin con \* que queramos y ver el menú que se despliega.



Si la etiqueta **PWM** está **Activada**, ese pin admitirá un rango y trabajará como **PWM**, si por el contrario, está **desactivada**, trabajará como pin **Digital**. En la imagen anterior, le hemos agregado un Slider de números Enteros y con un rango entre 0 y 255 (para poder trabajar con un Led) y podemos observar como este pin PWM admite un rango y lo envía a la ventana de salida que lo mandará a Arduino. En nuestro ejemplo, utilizaremos este pin como PWM para el control analógico de un led convencional.

**DPin10\*** → Como lo usaremos como PWM para el control de un servo, activaremos la pestaña PWM de este pin y le conectaremos un slider de números enteros entre 0 y 180 que es el rango en el que trabaja el Servo.

**DPin9\*** → En este ejemplo no lo usaremos, pero puede trabajar como PWM o como Digital.

**DPin8** → Necesita un número entero para controlar el pin Digital de Arduino nº 8. Le hemos conectado un slider de que sólo admite dos números o CERO o UNO, es decir, apagado o encendido.

**DPin6\*** → Como lo usaremos como PWM para el control de un Led RGB, activaremos la pestaña PWM de este pin y le conectaremos un slider de números enteros entre 0 y 255 que es el rango en el que trabaja el Led. Este pin controla la escritura del color **AZUL** (B) del Led RGB.

**DPin5\*** → Como lo usaremos como PWM para el control de un Led RGB, activaremos la pestaña PWM de este pin y le conectaremos un slider de números enteros entre 0 y 255 que es el rango en el que trabaja el Led. Este pin controla la escritura del color **VERDE** (G del Led RGB).

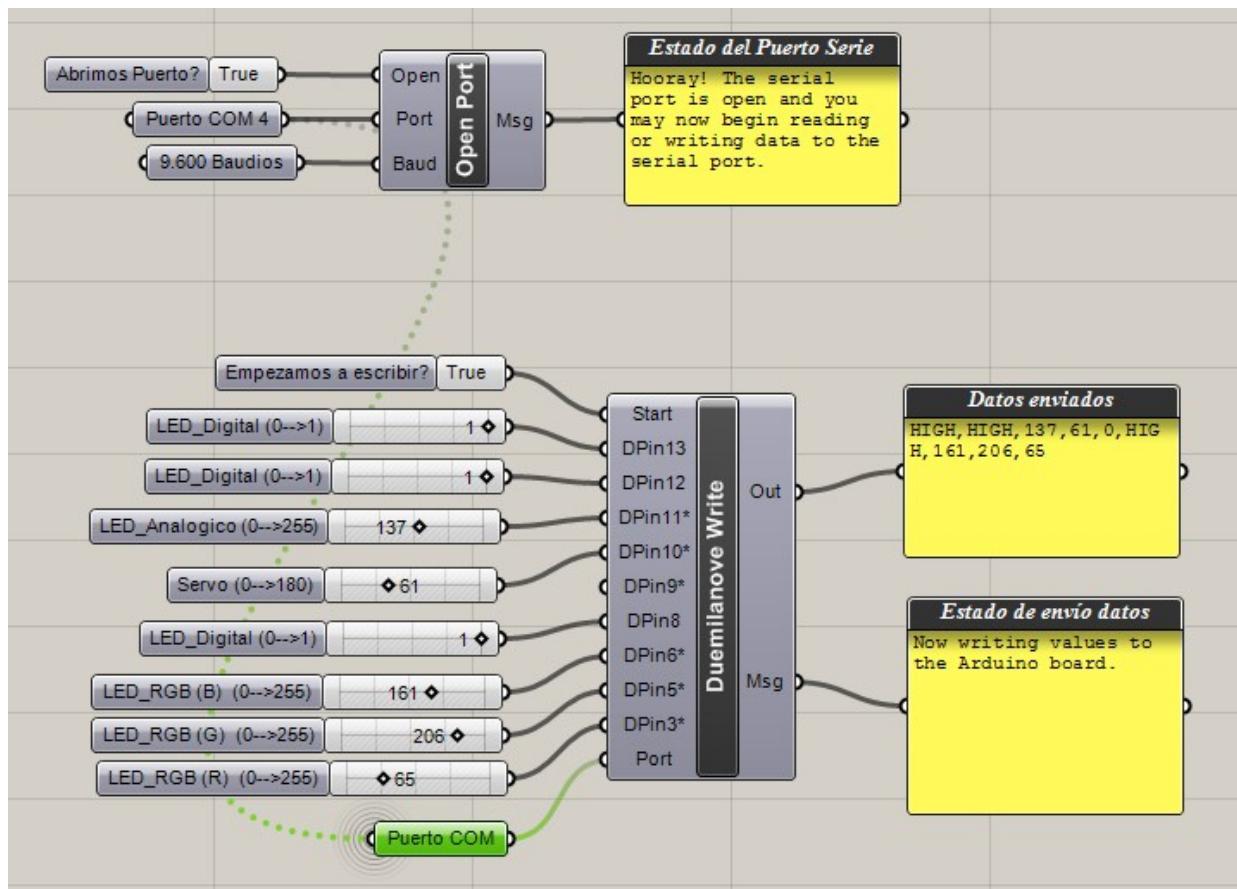
**DPin3\*** → Como lo usaremos como PWM para el control de un Led RGB, activaremos la pestaña PWM de este pin y le conectaremos un slider de números enteros entre 0 y 255 que es el rango en el que trabaja el Led. Este pin controla la escritura del color **AZUL** (R del Led RGB).

**Port** → Hay que conectarle un Receiver para que esté conectado al mismo puerto COM# que la pila de Open Port.

Salidas:

**Out** → Valores de Salida para mandarlos a Arduino.

**Msg** → Con una nota, podemos comprobar que el estado de Envío de los datos.



Ahora vamos a ver el código de **Arduino**:

```
#include <Servo.h>

#define BUFFSIZE 128

char buffer[BUFFSIZE];
uint8_t bufferidx = 0;
uint16_t DPin13, DPin12, PWM11, PWM10, PWM9, DPin8, PWM6, PWM5, PWM3;

char *parseptr;
char buffidx;
int ledAnalogicoPin = 11;
int ledDigitalPin1 = 13;
int ledDigitalPin2 = 12;
int ledDigitalPin3 = 8;
int ledRGB_R = 3;
int ledRGB_G = 5;
int ledRGB_B = 6;
Servo Servo1;

void setup() {
  pinMode(ledAnalogicoPin, OUTPUT);
  pinMode(ledDigitalPin1, OUTPUT);
  pinMode(ledDigitalPin2, OUTPUT);
  pinMode(ledDigitalPin3, OUTPUT);
  pinMode(ledRGB_R, OUTPUT);
```

```
pinMode(ledRGB_G, OUTPUT);
pinMode(ledRGB_B, OUTPUT);
Servo1.attach(10);           // Servo en el PWM 10
Serial.begin(9600);
}

void loop() {
    serialread();           // Llamamos a la Funcion serialRead
}

void serialread(){
    char c;                // captamos un caracter del Puerto Serie
    if (Serial.available()) {
        c = Serial.read();      // Leemos ese caracter
        buffer[bufferidx] = c; // añadimos ese caracter al buffer

        if (c == '\n') {        // Si leemos un SALTO DE LINEA...
            buffer[bufferidx+1] = 0; // ...terminamos de leer
            parseptr = buffer;    // ...descargamos el buffer en la variable parseptr

            DPin13 = parsedecimal(parseptr);
            parseptr = strchr(parseptr, ',')+1;

            DPin12 = parsedecimal(parseptr);
            parseptr = strchr(parseptr, ',')+1;

            PWM11 = parsedecimal(parseptr);
            parseptr = strchr(parseptr, ',')+1;

            PWM10 = parsedecimal(parseptr);
            parseptr = strchr(parseptr, ',')+1;

            PWM9 = parsedecimal(parseptr);
            parseptr = strchr(parseptr, ',')+1;

            DPin8 = parsedecimal(parseptr);
            parseptr = strchr(parseptr, ',')+1;

            PWM6 = parsedecimal(parseptr);
            parseptr = strchr(parseptr, ',')+1;

            PWM5 = parsedecimal(parseptr);
            parseptr = strchr(parseptr, ',')+1;

            PWM3 = parsedecimal(parseptr);

            if (DPin13 == 1) {
                digitalWrite(ledDigitalPin1, HIGH);
            }
            else {
                digitalWrite(ledDigitalPin1, LOW);
            }

            if (DPin12 == 1) {
                digitalWrite(ledDigitalPin2, HIGH);
            }
            else {
                digitalWrite(ledDigitalPin2, LOW);
            }
        }
    }
}
```

```
}

if (DPin8 == 1) {
    digitalWrite(ledDigitalPin3, HIGH);
}
else {
    digitalWrite(ledDigitalPin3, LOW);
}
// Analizamos las diferentes situaciones que podemos tener
if (PWM11 == 1){           // Si es IGUAL a UNO ...
    digitalWrite(ledAnalogicoPin, HIGH);    // Encendemos el LED
}
else if (PWM11 == 0){      // Si es IGUAL a CERO ...
    digitalWrite(ledAnalogicoPin, LOW);     // Apagamos el LED
}
else{                     // Si NO es ni UNO ni CERO ...
    analogWrite(ledAnalogicoPin, PWM11);    // Escribimos analogicamente el valor en el LED
}

if (PWM10 == 1){
    digitalWrite(10, HIGH);
}
else if (PWM10 == 0){
    digitalWrite(10, LOW);
}
else{
    Servo1.write(PWM10);
}
if (PWM6 == 1){           // Si es IGUAL a UNO ...
    digitalWrite(ledRGB_B, HIGH);    // Encendemos el LED
}
else if (PWM6 == 0){      // Si es IGUAL a CERO ...
    digitalWrite(ledRGB_B, LOW);    // Apagamos el LED
}
else{                     // Si NO es ni UNO ni CERO ...
    analogWrite(ledRGB_B, PWM6);    // Escribimos analogicamente el valor en el LED
}
if (PWM5 == 1){           // Si es IGUAL a UNO ...
    digitalWrite(ledRGB_G, HIGH);    // Encendemos el LED
}
else if (PWM5 == 0){      // Si es IGUAL a CERO ...
    digitalWrite(ledRGB_G, LOW);    // Apagamos el LED
}
else{                     // Si NO es ni UNO ni CERO ...
    analogWrite(ledRGB_G, PWM5);    // Escribimos analogicamente el valor en el LED
}

if (PWM3 == 1){           // Si es IGUAL a UNO ...
    digitalWrite(ledRGB_R, HIGH);    // Encendemos el LED
}
else if (PWM3 == 0){      // Si es IGUAL a CERO ...
    digitalWrite(ledRGB_R, LOW);    // Apagamos el LED
}
else{                     // Si NO es ni UNO ni CERO ...
    analogWrite(ledRGB_R, PWM3);    // Escribimos analogicamente el valor en el LED
}

// Una vez terminada la lectura y el paso del valor al led, empezamos otra vez a leer
```

```

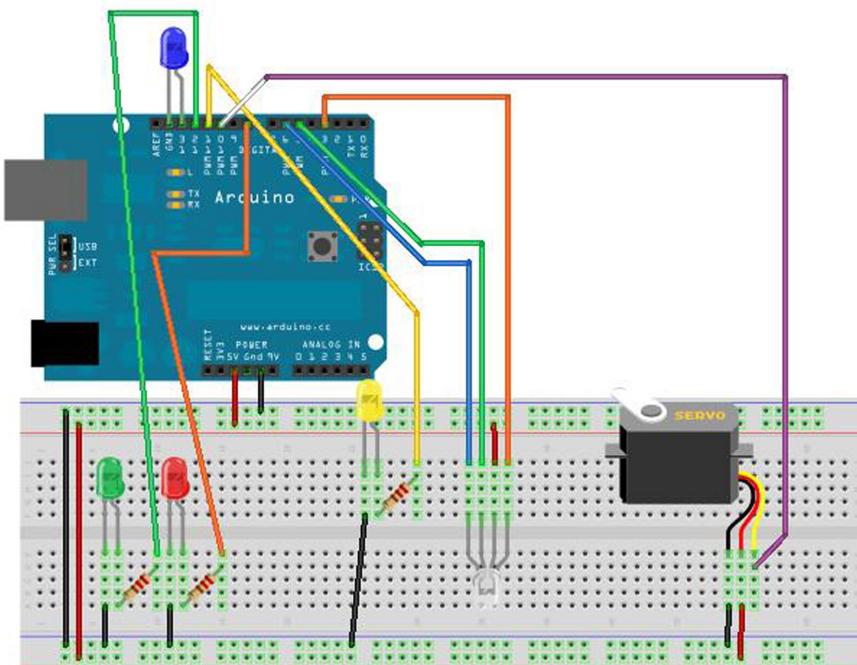
bufferidx = 0; // Reiniciamos el buffer para la siguiente lectura
return; // Devolvemos al buffer el CERO para no incrementar el Indice
}
// Hasta que no encontremos un SALTO DE LINEA, seguimos leyendo del buffer
bufferidx++; // Incrementamos el indice para el siguiente caracter
if(bufferidx == BUFFSIZE-1) { //Si llegamos al final del buffer, lo reseteamos por seguridad
    bufferidx = 0;
}
}

// Función para analizar los datos que enviamos desde GH
uint32_t parsedecimal(char *str) {
    uint32_t d = 0;

    while (str[0] != 0) {
        if ((str[0] > '50') || (str[0] < '0'))
            return d;
        d *= 10;
        d += str[0] - '0';
        str++;
    }
    return d;
}

```

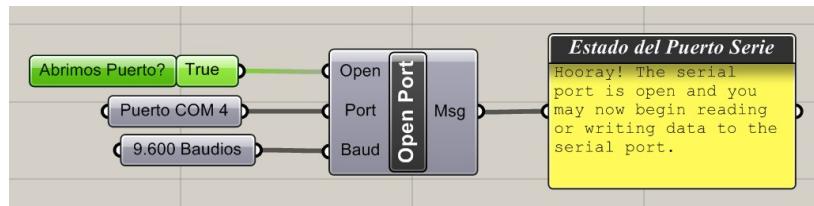
Aquí podemos ver cómo se realiza el esquema eléctrico en **Arduino**:



Ahora, Vamos a realizar un ejemplo de enviar datos desde Arduino a Firefly y controlar una **LDR** (que a su vez controla un Led analógicamente) y un **Botón** (que a su vez controla un led convencional digitalmente).

Primero, abriremos el Puerto Serie como hemos visto en todos los ejemplos anteriores, con la pila

Open Port (nos aseguraremos que el puerto COM# y la velocidad de comunicación de Arduino, es la misma que en Firefly).



Una vez que ya podemos abrir el puerto Serie, vamos a trabajar con la pila **Duemilanove Read**. Observamos que tiene 4 Entradas y 10 Salidas:

Entradas:

**Start** → Hay que conectarle un Toogle para poder cambiar de Falso a Verdadero la booleana que controla el estado de la Lectura a través del Puerto Serie.

**Port** → Necesita un número entero para que pueda localizar qué puerto **COM#** está siendo utilizado por Arduino. Se puede ligar con la entrada de la pila de Open Port.

**Timer** → Hay que conectarle un Toogle para poder cambiar de Falso a Verdadero la booleana que controla el estado del Timer (Temporizador).

**Rate** → Precisa de un número entero para que controlar el tiempo (en milisegundos) de retardo con el que empezar cada nueva lectura de los sensores. Es aconsejable que sea el mismo que hemos usado en el delay () de Arduino.

Salidas:

**APin0** → En este ejemplo, lo usaremos para recibir la lectura mapeada a un rango de 0-255 (para trabajar también con un led analógicamente) de una LDR.

**APin1** → En este ejemplo, no lo usaremos. Pero funciona igual que Apin0, recibiendo datos de un sensor analógico.

**APin2** → En este ejemplo, no lo usaremos. Pero funciona igual que Apin0, recibiendo datos de un sensor analógico.

**APin3** → En este ejemplo, no lo usaremos. Pero funciona igual que Apin0, recibiendo datos de un sensor analógico.

**APin4** → En este ejemplo, no lo usaremos. Pero funciona igual que Apin0, recibiendo datos de un sensor analógico.

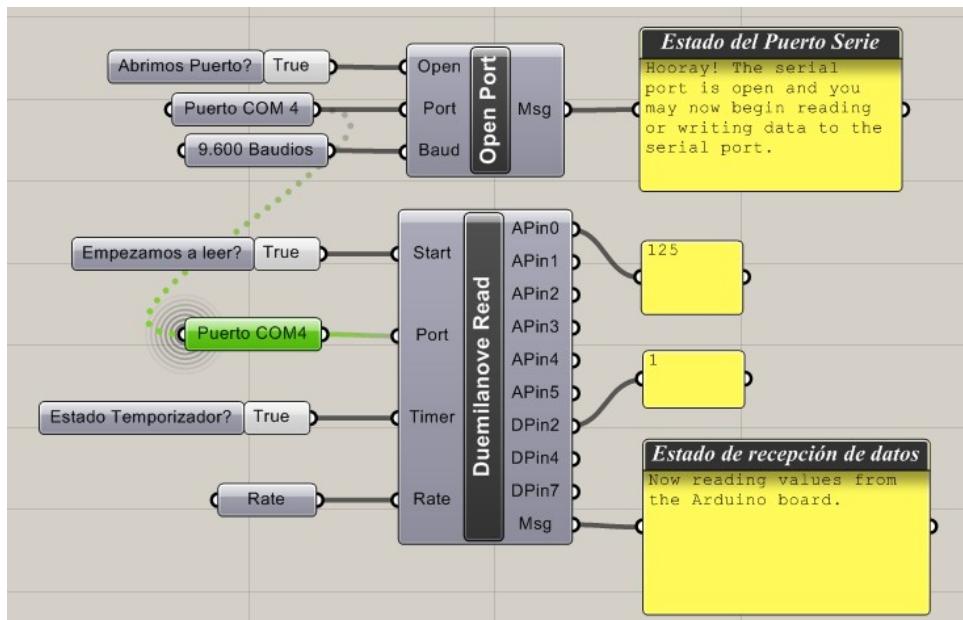
**APin5** → En este ejemplo, no lo usaremos. Pero funciona igual que Apin0, recibiendo datos de un sensor analógico.

**DPin2** → En este ejemplo, lo usaremos para recibir la lectura del estado de un Botón, que a su vez, controla si un led se enciende o no.

**DPin4** → En este ejemplo, no lo usaremos. Pero funciona igual que DPin2, recibiendo datos de un componente digital.

**DPin7** → En este ejemplo, no lo usaremos. Pero funciona igual que DPin2, recibiendo datos de un componente digital.

**Msg** → Con una nota, podemos comprobar que el estado de Recepción de los datos.



Ahora vamos a ver el código de **Arduino**:

```

int LDR_pin = 0;           // LDR al Analogico 0
int LED_pin = 11;          // El led que controla la LDR al PWM11
int LDR_val;               // Variable que almacena la lectura del LDR
int LDR_mapeado;           // Variable mapeada para encender el LED

int ledPin = 12;            // El led que controla el Botón al Digital 12
int buttonPin = 2;           // El Botón al Digital 2
int buttonState = 0;          // Variable que almacena el Estado del Botón

// Estos pines los definimos para que la pila en GH funcione bien, pero no los usaremos ahora
int APin1 = 0;
int APin2 = 0;
int APin3 = 0;
int APin4 = 0;
int APin5 = 0;

int DPin4 = 0;
int DPin7 = 0;

void setup() {
  pinMode(buttonPin, INPUT); // Botón y LDR como ENTRADAS
  pinMode(ledPin, OUTPUT);
  pinMode(LDR_pin, INPUT);   // Los 2 Leds como SALIDAS
  pinMode(LED_pin, OUTPUT);
  Serial.begin(9600);
}

void loop(){
  // Leemos el resto de pines, aunque en este ejemplo no los usamos.
  // Si no lo hacemos, la pila Read en GH puede funcionar mal.
  APin1 = analogRead(1);
  APin2 = analogRead(2);
}

```

```
APin3 = analogRead(3);
APin4 = analogRead(4);
APin5 = analogRead(5);

DPin4 = digitalRead(4);
DPin7 = digitalRead(7);

ldrYled(); // Llamamos a la función que trabaja con la LDR

botonLED(); // Llamamos a la función que trabaja con el Botón

Serial.print(LDR_mapeado);
Serial.print(",");
Serial.print(APin1);
Serial.print(",");
Serial.print(APin2);
Serial.print(",");
Serial.print(APin3);
Serial.print(",");
Serial.print(APin4);
Serial.print(",");
Serial.print(APin5);
Serial.print(",");
Serial.print(buttonState);
Serial.print(",");
Serial.print(DPin4);
Serial.print(",");
Serial.print(DPin7);
Serial.print(",");
Serial.println("eol");
delay(50);
}
```

#### FUNCTIONES AUXILIARES

```
void ldrYled(){
    LDR_val = analogRead(LDR_pin); // leemos el valor del LDR
    LDR_mapeado = constrain(LDR_val,0,255); // Limitamos el valor del LDR al rango [0-255] para el LED
    analogWrite(LED_pin,LDR_mapeado); // Le pasamos al LED el valor mapeado
}

void botonLED(){
    buttonState = digitalRead(buttonPin); // Leemos el estado del Botón
    // Si NO apretamos el botón, el Led está apagado
    if (buttonState == HIGH) {
        digitalWrite(ledPin, LOW);
    }
    else {
        // Si apretamos el botón, encendemos el Led
        digitalWrite(ledPin, HIGH);
    }
}
```

Aquí podemos ver cómo se realiza el esquema eléctrico en **Arduino**:

