E. E. S. T. Nº 5 – TIGRE

ELECTRÓNICA

SISTEMAS DE CONTROL

Arduino

TRABAJO PRÁCTICO Nº 7

TEMA: Comunicación serial, sensores y salidas de potencia.

Función map(). Instrucción switch..case

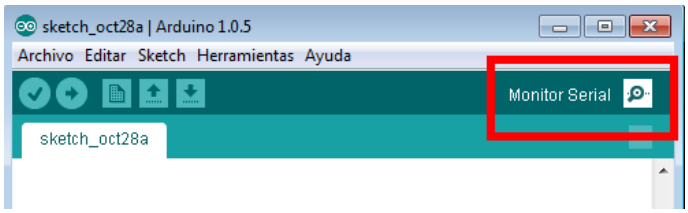
ALUMNO:	**Curso:**
FECHA DE ENTREGA:	
FECHA DE APROBACIÓN:	
CALIFICACIÓN:	FIRMA:

Comunicación serie: Serial

La interfaz serie se utiliza para la comunicación entre la placa Arduino y un ordenador u otros dispositivos. Todas las placas Arduino tienen al menos un puerto serie (también conocido como **UART** o **USART**). El Arduino se puede comunicar a través de:

- los pines digitales **0** (RX) y **1** (TX) con cualquier dispositivo con comunicación serie, en este caso no pueden usarse estos pines como entrada o salida digital
- o el cable USB para comunicarse con una computadora personal.

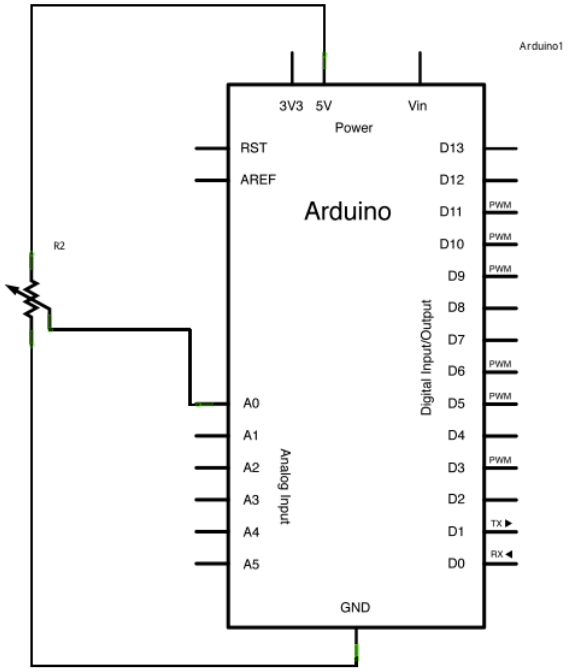
El entorno Arduino posee una **herramienta** interesante que, si está configurada correctamente, nos permite **enviar** y **recibir** datos en tiempo real con la placa Arduino: se trata del **Monitor Serial**.



El botón del Monitor Serial está a la **derecha** en la barra de botones.

Para aprender a utilizar **Serial** vamos a conectar un **potenciómetro** a la entrada **`A0`** como ya lo hemos hecho anteriormente: **uno** de los extremos al **positivo** +5V, el **otro** extremo a **GND** y el punto **medio** a la entrada **analógica** `A0`.

Para las comprobaciones utilizaremos el ejemplo ***AnalogReadSerial*** que se encuentra en Ejemplos – 01.Basics.



AnalogReadSerial §

```
/*
  AnalogReadSerial
  Reads an analog input on pin 0, prints the result to the serial monitor.
  Attach the center pin of a potentiometer to pin A0, and the outside pins to +5V and ground.

 This example code is in the public domain.
 */

// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}

// the loop routine runs over and over again forever:
void loop() {
  // read the input on analog pin 0:
  int sensorValue = analogRead(A0);
  // print out the value you read:
  Serial.println(sensorValue);
  delay(1);        // delay in between reads for stability
}
```

La comunicación serie se configura en **`void setup()`** través de **`Serial.begin`**.

```
// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}
```

`begin()`

Descripción

Establece la velocidad de datos en bits por segundo (baudios) para la transmisión de datos en serie. Para comunicarse con el computador, utilice una de estas velocidades: 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600 o 115200. Sin embargo, puede especificar otras velocidades si a través de los pines 0 y 1 conecta un componente que requiera una velocidad de transmisión en particular.

Sintaxis

`Serial.begin(speed);`

Parámetros

speed: Velocidad en bits por segundo (baudios).

Devuelve

Nada.

Por otro lado, para poder **obtener** los datos a través del **Monitor Serial** se utiliza la función **`Serial.println`**.

```
// read the input on analog pin 0:
int sensorValue = analogRead(A0);
// print out the value you read:
Serial.println(sensorValue);
```

`println()`

Descripción

Escribe los datos al puerto serie como texto ASCII seguido de un retorno de carro (ASCII 13, o '\r') y un carácter de avance de línea (ASCII 10, o '\n'). Este comando tiene la misma forma que **`Serial.print()`**.

Sintaxis

`Serial.println(val);`

`Serial.println(val, format);`

Parámetros

val: el valor a imprimir de cualquier tipo. Se transforma siempre a caracteres ASCII.

format: especifica el número de la base (para números enteros) o el número de posiciones decimales (para números de coma flotante o tipo "float")

Devuelve

Nada.

Más detalles de println

Tanto **`print`** como **`println`** **envían** los datos al puerto serie como texto **ASCII**. Estos comandos pueden tomar muchas formas. Los números **enteros** son impresos mediante un juego de caracteres ASCII para **cada** dígito. Los valores de tipo **`float`** son impresos en forma de dígitos ASCII con **dos decimales** por defecto. Los valores tipo **`byte`** se envían como un **único** carácter. Los caracteres y las cadenas se envían **tal cual**. Ejemplos:

```
Serial.println(78);                 // imprime "78"
Serial.println(1.23456);            // imprime "1.23"
Serial.println(byte(78));           // imprime "N" (cuyo ASCII es 78)
Serial.println('N');                // imprime "N"
Serial.println("Hello world.");     // imprime "Hello world."
```

Un **segundo** parámetro **opcional** especifica la base (formato) a usar. Los valores permitidos son:

Para valores enteros:

- BYTE (representa en carácter que le corresponda según su código ASCII)
- BIN (binarios o números de base 2)
- OCT (octales o números de base 8)
- DEC (decimales o números de base 10)
- HEX (hexadecimales o números de base 16).

Para números de coma flotante: este parámetro especifica el numero de posiciones decimales a usar.

Ejemplos

```
Serial.println(78, BYTE);           // imprime "N"
Serial.println(78, BIN);            // imprime "1001110"
Serial.println(78, OCT);            // imprime "116"
Serial.println(78, DEC);            // imprime "78"
Serial.println(78, HEX);            // imprime "4E"
Serial.println(1.23456, 0);         // imprime "1"
Serial.println(1.23456, 2);         // imprime "1.23"
Serial.println(1.23456, 4);         // imprime "1.2346"
```

Actividad

EJ-0701) Conecte en el protoboard el **potenciómetro** de acuerdo al modelo de hardware mostrado, **compruebe** el funcionamiento del sketch ***AnalogReadSerial*** abriendo el **Monitor Serial** y observe la **variación** de los valores mostrados cuando varía la **posición** del cursor del potenciómetro.

Sensor de luminosidad: fotorresistencia o LDR

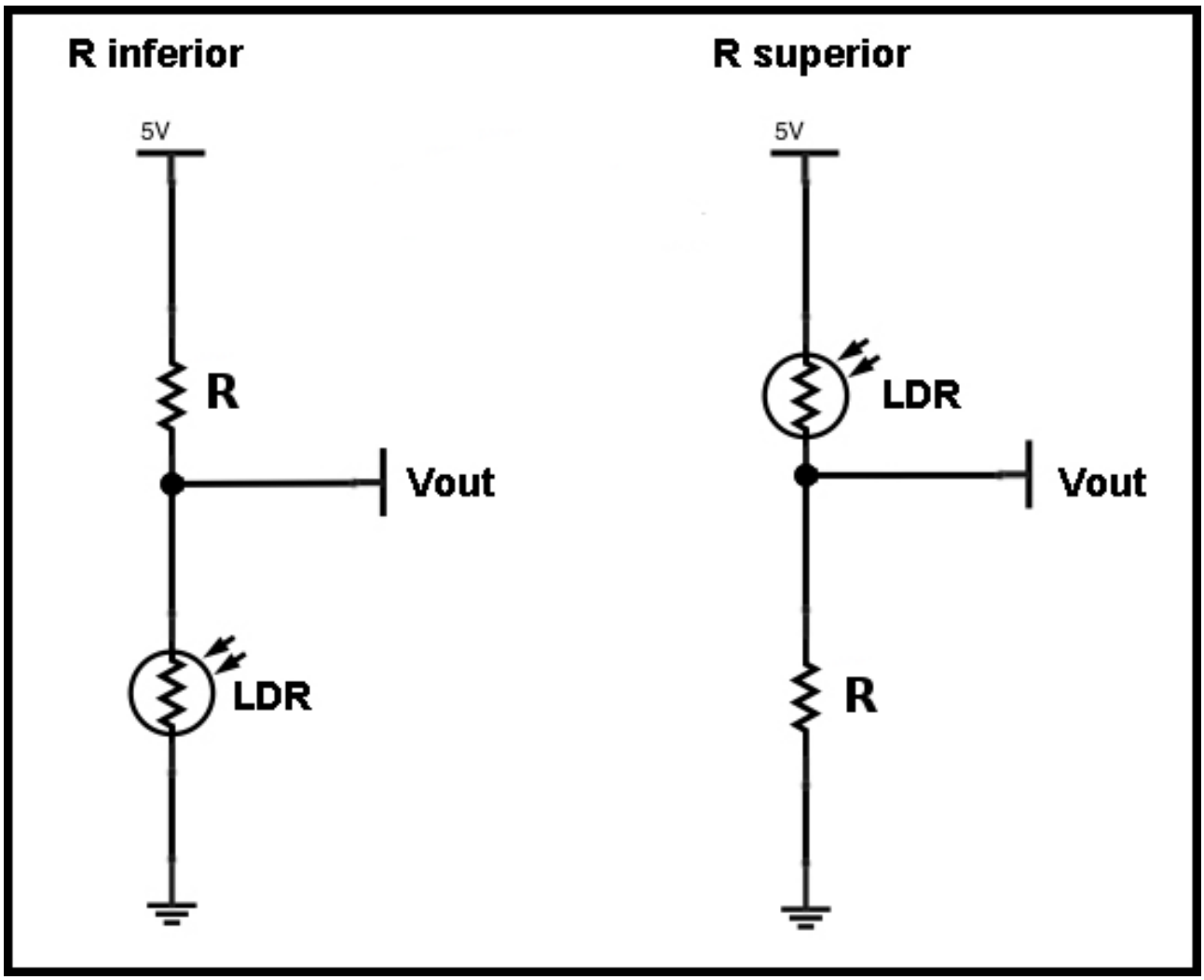
Hasta el momento hemos empleado un potenciómetro como entrada analógica variable, es hora de emplear otros elementos y comenzaremos con una **fotorresistencia**.

Una **fotorresistencia** es un componente electrónico cuya resistencia **disminuye** con el **aumento** de la **intensidad** de luz incidente. Puede también ser llamado **fotorresistor**, **fotoconductor**, **célula fotoeléctrica** o resistor dependiente de la luz, cuyas siglas, **LDR**, se originan de su nombre en inglés light-dependent resistor. Su cuerpo está formado por una célula o celda y dos patillas. En la imagen de la derecha se muestra su **símbolo** eléctrico.

El valor de **resistencia** eléctrica de un LDR es **bajo** cuando haya **luz incidiendo** en él (puede descender hasta 50 ohms) y muy **alto** cuando está a **oscuras** (varios megaohmios).

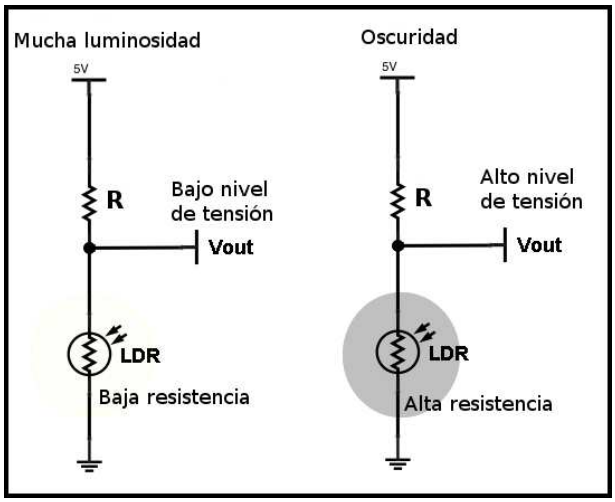
Un LDR se utiliza siempre acompañado de otra resistencia como divisor de tensión. Puede ser conectado de dos maneras diferentes: como resistencia **inferior** o como resistencia **superior**.

El **modo** en que conectemos el LDR provocará un **comportamiento diferente** en la entrada analógica de acuerdo a si hay presente luz o no en la fotocelda.



LDR como resistencia inferior

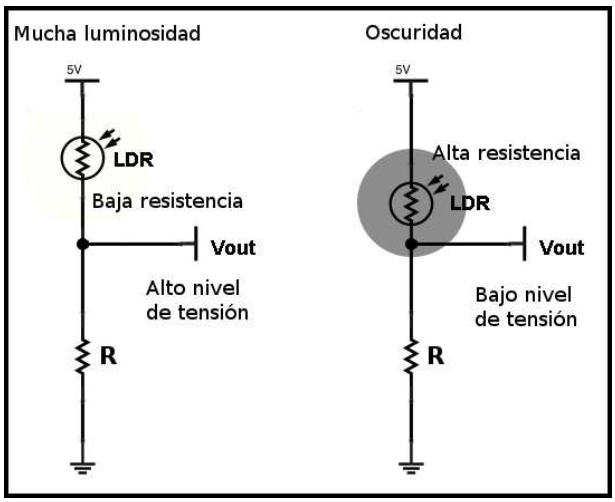
El comportamiento lo podemos observar en la imagen de la derecha.



Cuando hay **mucha** luz incidente en la fotocelda la resistencia será **baja** facilitando el camino de la corriente a tierra y provocando un **bajo** nivel de tensión en **Vout**.

En cambio, con **poca** luz u **oscuridad**, la resistencia **aumentará** oponiéndose al paso de la corriente con lo cual provocará que en Vout haya un nivel de tensión **elevado**: cuanta más alta sea la resistencia del LDR más cercano a 5V será Vout.

LDR como resistencia superior

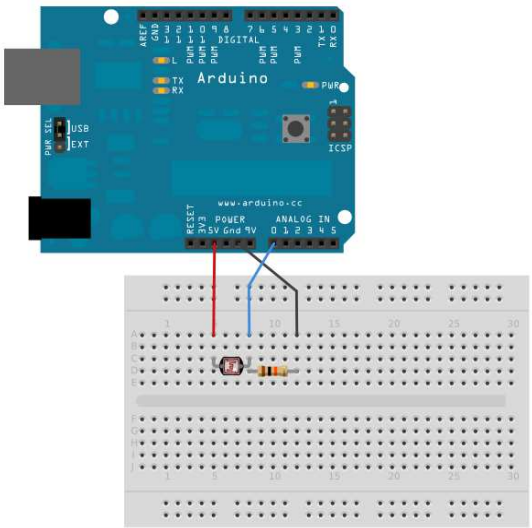


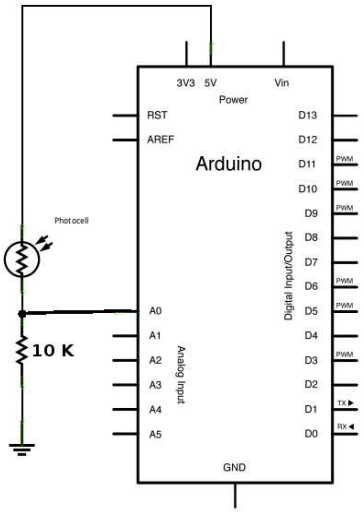
El **comportamiento** de Vout cuando el LDR está conectado como resistencia superior es **inverso**.

En Vout tendremos una **tensión alta** cuando esté completamente **iluminado**, ya que se comportará prácticamente como un cortocircuito, con una resistencia muy baja de 50Ω o 100Ω. Es como si estuviera la entrada conectada directamente a 5 V.

En **oscuridad** la resistencia será muy alta y también será alta la caída de **tensión** provocando que Vout sea **bajo**.

En el ejemplo que analizaremos haremos esta última conexión ya que interpretaremos un valor alto en entrada analógica como un elevado nivel de luminosidad. A continuación le muestro como conectar el LDR al Arduino.





Actividad

EJ-0702) Conecte en el protoboard el **LDR** de acuerdo al modelo de hardware mostrado y vuelva a ejecutar el sketch ***AnalogReadSerial*** junto con el monitor serial.

Para poder utilizar este sensor en el siguiente ejercicio deberá observar y tomar nota de ciertos valores mostrados en 4 situaciones del LDR:

- cuando esté totalmente tapado.
- con la mano cubriendo en sensor impidiendo la llegada de luz pero sin tocarlo.
- con la mano un poco más lejos del sensor.
- con el sensor sin ningún obstáculo.

Tome nota de los valores mínimos y máximos porque dependerá del ambiente en donde haga las pruebas. Si es necesario, aumente el delay del sketch para poder apreciarlos mejor.

Función maps() y estructura switch

El siguiente ejemplo que vamos a analizar es el que se encuentra en ***Ejemplos – 05.Control – switchCase.***

Tiene este encabezado:

```
/*
  Switch statement

 Demonstrates the use of a switch statement.  The switch
 statement allows you to choose from among a set of discrete values
 of a variable.  It's like a series of if statements.

 To see this sketch in action, but the board and sensor in a well-lit
 room, open the serial monitor, and and move your hand gradually
 down over the sensor.

 The circuit:
 * photoresistor from analog in 0 to +5V
 * 10K resistor from analog in 0 to ground

 created 1 Jul 2009
 modified 9 Apr 2012
 by Tom Igoe

 This example code is in the public domain.
```

Este ejemplo muestra mediante diferentes **mensajes** los cambios producidos en la **luminosidad** que le llega al LDR. Para ello se emplea en el sketch una estructura denominada **`switch..case`** que analizaremos luego.

Para que funcione correctamente debe **conectar** la fotocelda de la manera que lo hicimos en el ejercicio anterior y tener a mano los **valores** que anotó en su experimentación.

Luego del encabezado se definen las constantes y se configura la comunicación serie.

```
// these constants won't change. They are the
// lowest and highest readings you get from your sensor:
const int sensorMin = 0;      // sensor minimum, discovered through experiment
const int sensorMax = 600;    // sensor maximum, discovered through experiment

void setup() {
  // initialize serial communication:
  Serial.begin(9600);
}
```

El autor del programa aclara que los valores de **`sensorMin`** y de **`sensorMax`** los descubrió por experimentación. Esos valores los debe modificar por los que encontró Ud. porque son **dependientes** del LDR utilizado y de la **luz** ambiente del lugar. Como dijimos, luego se configura la velocidad de transmisión a 9600 bps.

El siguiente paso, ya dentro del programa principal, es hacer la **lectura** del valor del **sensor**. A través de la función **`analogRead`** se lee el valor de la entrada analógica **`A0`** y se lo almacena en la variable **`sensorReading`**.

```
void loop() {
  // read the sensor:
  int sensorReading = analogRead(A0);
  // map the sensor range to a range of four options:
  int range = map(sensorReading, sensorMin, sensorMax, 0, 3);
```

Como hay un **amplio** rango de valores posibles del valor leído, se hace el uso de la función **`map`** para reducir ese rango a solamente **cuatro**. Veamos como funciona **`map`**:

`map()`

Descripción

Re-mapea un número desde un rango hacia otro. Esto significa que, un valor con respecto al rango **`desdeInf - desdeSup`** será mapeado al rango **`hastaInf-hastaSup`**.

Sintaxis

`map(valor, desdeInf, desdeSup, hastaInf, hastaSup);`

Parámetros

`valor`: el número (valor) a mapear.

`desdeInf`: el límite **inferior** del rango **actual** del valor.

`desdeSup`: el límite **superior** del rango **actual** del valor.

`hastaInf`: límite **inferior** del rango **deseado**.

`hastaSup`: límite **superior** del rango **deseado**.

Devuelve

El valor mapeado.

Ejemplo:

```
/* Mapea un valor análogo a 8 bits (0 a 255) */
void setup() {}

void loop()
{
  int val = analogRead(0);          // lee un valor rango 0..1023
  val = map(val, 0, 1023, 0, 255);  // mapea el valor al rango 0..255
  analogWrite(9, val);             // para usarse con analogWrite
}
```

¿Para qué se hace todo esto? Respuesta: en este ejemplo para utilizarlo para mostrar cuatro mensajes diferentes.

Unas líneas antes observamos que la función **`map`** fue utilizada de esta manera:

```
int range = map(sensorReading, sensorMin, sensorMax, 0, 3);
```

Esto hace que el valor leído (**`sensorReading`**) que va a estar comprendido entre los valores que UD experimentó (**`sensorMin`** y **`sensorMax`**) se van a **mapear** a **cuatro** valores posibles: 0, 1, 2 y 3. En pocas palabras los valores **bajos** del LDR se van a transformar en valores **0**, los más **altos** en valor **3** y los **intermedios** se repartirán entre los valores **1** y **2**.

Una manera se utilizar estos valores es utilizar la instrucción **`if`**. En este caso se necesitarían cuatro **`if`**. En estas situaciones es más **práctico** utilizar la instrucción **`switch..case`**. Veamos como:

```
  // do something different depending on the
  // range value:
  switch (range) {
  case 0:    // your hand is on the sensor
    Serial.println("dark");
    break;
  case 1:    // your hand is close to the sensor
    Serial.println("dim");
    break;
  case 2:    // your hand is a few inches from the sensor
    Serial.println("medium");
    break;
  case 3:    // your hand is nowhere near the sensor
    Serial.println("bright");
    break;
  }
  delay(1);        // delay in between reads for stability
}
```

switch..case

Descripción

Controla el flujo de los programas permitiendo a los programadores especificar diferentes códigos que deberían ser ejecutados en función de varias condiciones. En particular, una sentencia **switch** compara el **valor** de una **variable** con el valor especificado en las sentencias **case**. Cuando se encuentra una sentencia **case** cuyo valor coincide con dicha variable, el código de esa sentencia se **ejecuta**.

La palabra clave **break** sale de la sentencia **switch**, y es usada típicamente al final de cada **case**. Sin una sentencia break, la sentencia **switch** continuaría ejecutando las siguientes expresiones hasta encontrar un **break** o hasta llegar al final de la sentencia **switch**.

Sintaxis

```
switch (var) {
 case etiqueta:
   // sentencias
   break;
 case etiqueta:
   // sentencias
   break;
 default:
   // sentencias
}
```

Parámetros

var: la variable cuyo valor se compara con los varios "case".

etiqueta: un valor para comparar con la variable.

Ejemplo

```
switch (var) {
   case 1:
     //hacer algo cuando var sea igual a 1
     break;
   case 2:
     //hacer algo cuando var sea igual a 2
     break;
   default:
     // si nada coincide, se ejecutan las instrucciones "default"
     // "default" es opcional
 }
```

En el ejemplo mostrado, según el valor leído de la fotocelda, el Arduino emitirá un mensaje a través de la comunicación serie, en base a cuatro posibilidades:

- dark
- dim
- medium
- light

Actividad

EJ-0703) Con el LDR conectado cargue el ejemplo: ***Ejemplos – 05.Control – switchCase*** y si hizo bien la calibración debería mostrar correctamente los mensajes en las cuatro situaciones.

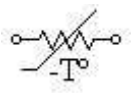
EJ-0704) Basado en el ejemplo anterior escriba un programa para controlar la cantidad de LEDs encendidos (hasta 4) de acuerdo al nivel de iluminación del LDR. Si el sensor está iluminado totalmente con una linterna deben estar **todos** apagados.

EJ-0705) Controle el brillo de un LED de acuerdo al valor de luminosidad: cuanto más oscuro, más brillante deberá estar el LED.

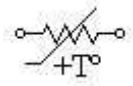
Sensor temperatura: termistor

Un termistor es un **sensor resistivo** de **temperatura**. Su funcionamiento se basa en la variación de la resistividad que presenta un semiconductor con la temperatura. El término termistor proviene de **Therm**ally Sensitive Re**sistor**. Existen dos tipos de termistores:

NTC (Negative Temperature Coefficient) – coeficiente de temperatura negativo. A la derecha se muestra su símbolo.



PTC (Positive Temperature Coefficient) – coeficiente de temperatura positivo. El símbolo del PTC lo vemos en la imagen de la derecha.



Cuando la **temperatura aumenta**, los tipo **PTC aumentan** su **resistencia** y los **NTC** la **disminuyen**.

Los termistores pueden venir en diferentes formatos:

				
Perla	SMD	Disco	Axial	Sonda

Sin entrar en demasiados detalles, un termistor se comporta igual que un LDR pero con respecto a la temperatura.

Vamos a utilizar una NTC, es decir, una resistencia con coeficiente de temperatura negativo respecto a la variación de su resistencia, esto es que a más temperatura, aumentará la concentración de portadores, lo que hará que su resistencia sea menor.

Un NTC de 10K quiere significar que su resistencia es de 10K a temperatura ambiente (25°C) y como **no tiene** una respuesta lineal, los fabricantes proveen tablas.

Ohm / Temperature figure

TMP.	-50°C	-40°C	-30°C	-20°C	-10°C	0°C	10°C	20°C	25°C	30°C
Ω	725K	360.9K	188.2K	102.3K	57670	33630	20240	12560	10000	8013

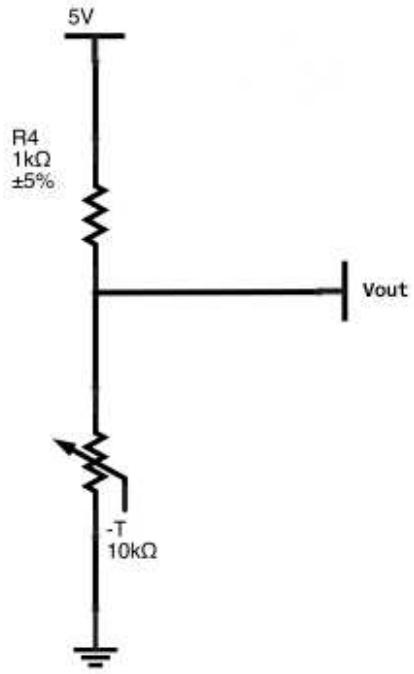
TMP.	40°C	50°C	60°C	70°C	80°C	90°C	100°C	110°C	120°C	130°C
Ω	5241	3507	2400	1677	1194	865.2	637	473.1	361	277.4

Conexión del NTC

La manera de conectar este sensor a nuestro circuito es formando un **divisor** de **tensión** en el que nuestro NTC será la resistencia inferior con su salida a una entrada analógica.

Siempre hay que limitar la corriente porque si llegara a ser muy alta afectaría el valor de la resistencia del sensor y por lo tanto las lecturas serían erróneas.

En cuanto a la otra resistencia que formará el divisor de tensión, se puede usar una de 1KΩ, esto es así para aprovechar el rango de muestreo que nos proporciona Arduino con un consumo de corriente limitado, veamos rápidamente el motivo.



Si recordamos la ecuación del divisor de tensión:

$$Vout = \frac{Rinferior}{Rinferior + Rsuperior}$$

Con este esquema $R_{superior}$ es fija de 1K y como la $R_{inferior}$ variará con la temperatura, esa variación será reflejada en Vout. Por ejemplo, haciendo unos cálculos y utilizando la tabla de arriba tenemos:

Temperatura	R del NTC	Vout
25° C	10000 ohm	4.546 V
60° C	2400 ohm	3.529 V
100° C	637 ohm	1.946 V

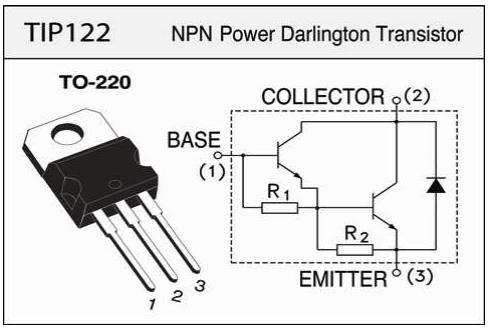
Como observamos, valores bajos de tensión nos indicarían temperaturas altas.

Este esquema nos serviría para detectar, por ejemplo, un exceso de temperatura y tomar una acción al respecto: **encender** un ventilador para bajarla.

Pero ¡**cuidado!** no se le ocurra conectar un ventilador directamente al Arduino porque necesitará más corriente que la que provee un pin de salida.

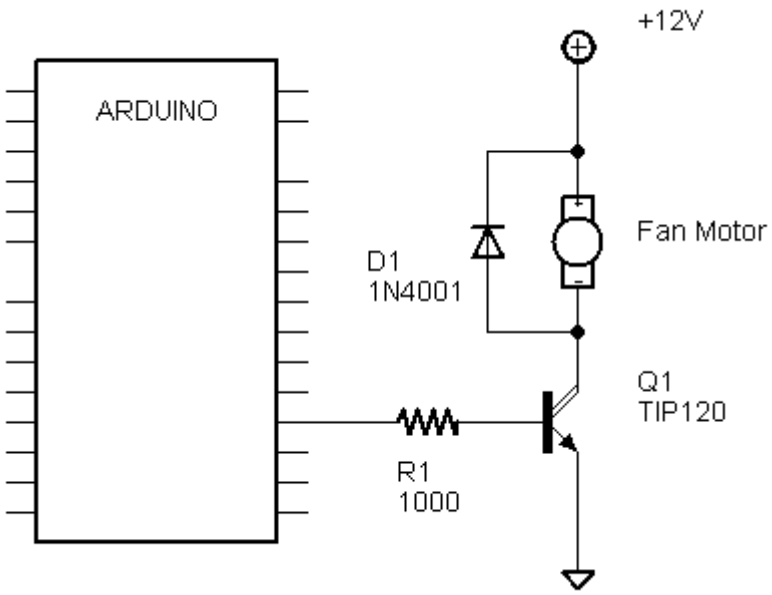
Un circuito que maneja más corriente

Una solución para controlar con el Arduino un dispositivo con alimentación CC que necesite una corriente más alta, es usar un transistor NPN Darlington diseñado para aplicaciones de conmutación lineales de potencia media: el TIP122. Este componente se puede utilizar para alimentar dispositivos tales como motores, relés y ventiladores, donde la única operación de control de operación necesaria sea ON - OFF.



La base del transistor (PIN1) se conecta a una salida digital de Arduino a través de una resistencia de 1Kohm . El emisor (PIN3) se conecta a tierra y el colector (PIN2) se conecta a un extremo de la bobina del dispositivo que quiere controlar. El otro extremo de la bobina está conectado a la fuente de alimentación de 12 VCC externa (la tierra de esta fuente de alimentación se conecta a una tierra común con el Arduino. Esto es necesario para el transistor funcione.

Es muy importante colocar el diodo en paralelo a la bobina del dispositivo que está siendo manejado para proteger el del pico de potencial de tensión que se genera cuando se desconecta la carga inductiva.



Actividad

EJ-0706) Arme el hardware correspondiente y desarrolle un programa que mantenga encendido un LED de color **verde** a una temperatura **ambiente** o **inferior** a ella, un LED **amarillo** cuando la temperatura sea un poco superior pero que no llegue por ejemplo a los **80 °C** y que se encienda el LED **rojo** cuando la temperatura sea elevada.

EJ-0707) Genere un programa que realice un control **ON-OFF** de temperatura. A una determinada temperatura detectada mediante un **NTC** que se **encienda** un **ventilador** para bajarla. El ventilador deberá **apagarse** cuando se detecte nuevamente la temperatura **ambiente**.