Magia

Esta semana vamos a construir algunos proyectos relacionados con la magia. Para alucinar del todo, os juntaréis en grupos y construiréis uno de los proyectos más increíbles pero también, de los más sencillos. Serás un músico reproduciendo música con diferentes objetos al azar, un artista que experimenta con el arte temporal o simplemente crearás un monstruo que proteja tus galletas.

Para realizar el trabajo, podrás adentrarte en el reino de las señales analógicas. Aprenderás los conceptos básicos de lectura y escritura analógica, además de jugar con sensores que dan valores analógicos. También podrás conocer cómo funciona el Puerto Serial, que le da la potencia a Arduino para comunicarse con el ordenador, ampliando sus capacidades.

# Leyendo en analógico

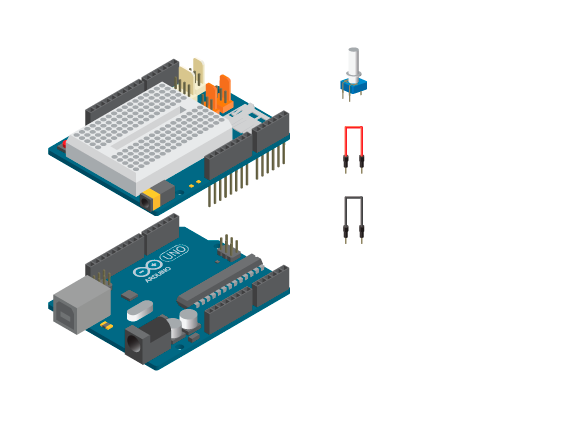
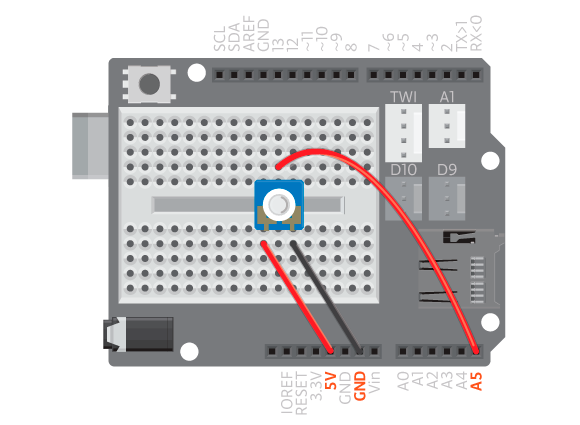
Como has aprendido anteriormente, las señales digitales solo tienen dos estados opuestos: 1 ó 0. Si presionas un botón, su estado cambiará de uno a otro. Un LED está encendido o apagado.

El mundo real sin embargo, no es digital. El agua por ejemplo, no sólo está caliente o fría, puede también estar templada. Para poder medir eso, y otras cosas del mundo real, no podemos usar sólo señales digitales. En lugar de ello, empleamos señales analógicas.

En vez de 2 estados opuestos, las señales analógicas tienen niveles continuos. Así que si tienes un sensor de luz, puedes obtener muchos valores diferentes que expresan cómo de iluminada está la habitación, y no sólo clara/oscura. O en el caso de un termómetro, te dice la temperatura mediante un número, en lugar de fría/caliente.

Con Arduino, puedes obtener los valores analógicos de los Pins analógicos. Sobre la placa puedes ver un grupo de Pins marcados como analog in, que llevan el nombre de A0 a A5. Cuando se les aplica tensión, ellos reportan valores de 0 a 1023; de este modo, cuando no hay voltaje en un Pin, la lectura es 0. Si le aplicas 5V, la lectura será 1023. Con una entrada de 2.5V, te dará 512. Para leer estos valores desde un Pin analógico tienes que utilizar la función analogRead(), en lugar de digitalRead().

Para seguir explicando las señales analógicas, necesitamos introducir el potenciómetro. Un potenciómetro es un control que funciona mediante el giro de un brazo. Por ejemplo, el regulador de volumen de un estéreo es un potenciómetro. Con dos Pins exteriores al potenciómetro, conectados a GND y 5V respectivamente, puedes utilizar el potenciómetro para controlar la cantidad de tensión que quieres que haya en el Pin central, siempre dentro del rango de 0V a 5V.

Aquí vamos a experimentar un poco usando un potenciómetro para controlar el parpadeo de un LED. Conecta el Pin central del potenciómetro al Pin analógico A5, los otros Pins conéctalos a 5V y GND. Usaremos el LED on-board, incluido en la placa, como salida.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | int ledPin = 13;    void setup() {    pinMode(ledPin, OUTPUT);  }    void loop() {    int val = analogRead(A5);      digitalWrite(ledPin, HIGH);    delay(val);    digitalWrite(ledPin, LOW);    delay(val);  } |

Puedes encontrar un nuevo comando:

* analogRead (pinNumber): Este comando coge la lectura de un Pin analógico especificado por la variable pinNumber, que puede ser desde A0 hasta A5. El valor leído variará entre 0 (0V) y 1023 (5V).
* Los pines analógicos sólo se pueden usar como entradas, por lo que no es necesario declarar el pin mode en el setup.

La lectura del potenciómetro conectado al pin analógico A5 cambiará el tiempo de retardo, delay, y por lo tanto la velocidad con la que parpadea el LED on-board del pin 13. Cuando gires el brazo del potenciómetro, podrás ver el cambio en la frecuencia de parpadeo.

## ¡Sigue experimentando!

* El valor de analogRead() tiene 1024 niveles – entre 0 y 1023. ¿Se te ocurre una manera de procesar un valor utilizando menos niveles? Imagina que sólo necesitamos 10 valores diferentes, manteniendo 0V como 0 pero 5V como 10, en lugar de 1023.(SUGERENCIA: Busca una función llamada map() en la referencia de Arduino.)
* ¿Qué sucede si reduces el número de niveles de analogRead a solamente dos valores, 0 y 1? Intenta aplicarlo al ejemplo del botón digital. ¿Sientes que has entendido la relación entre las señales analógicas y digitales?

# Escribiendo en analógico

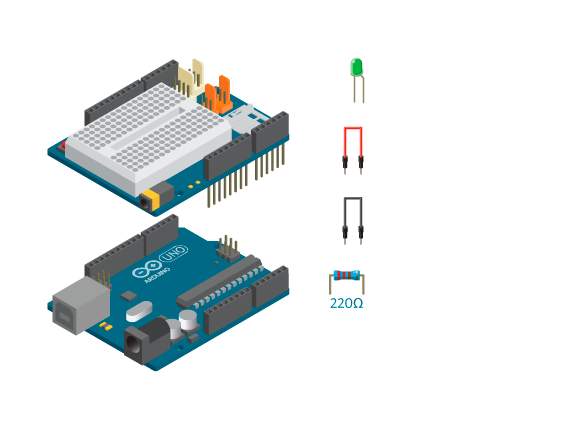
Al igual que puedes leer señales analógicas, también puedes generar valores analógicos. Arduino utiliza Pins con Modulación Controlada por Pulsos, PWM (del inglés Pulse Width Modulation) para enviar los valores analógicos. Echa un vistazo a los Pins digitales que llevan al lado un símbolo de corriente (~).

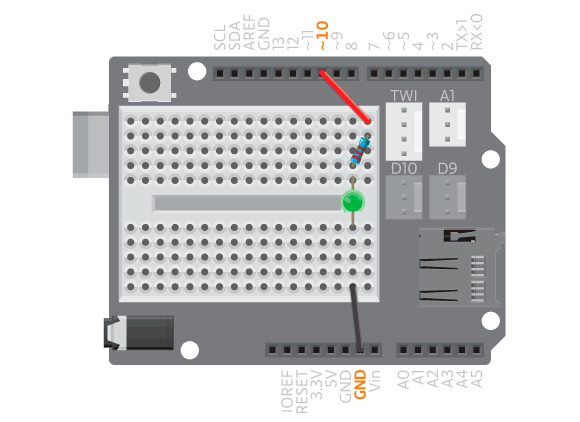
Cuando se utiliza un Pin digital normal para escribir HIGH o LOW, se obtiene bien 5V o 0V. Estos Pins PWM tienen una habilidad diferente, puedes utilizarlos para obtener un nivel de tensión en cualquier punto entre 5V y 0V. Con esto, puedes atenuar un LED sin problemas entre ON y OFF.

Para utilizar esta habilidad especial de los Pins PWM, tendrás que usar la función analogWrite(). Se necesitan dos parámetros, el número del Pin PWM, y el nivel de salida. El nivel de salida es un número entre 0 y 255. 0 es igual a digitalWrite(pin, LOW), 255 es igual a ejecutar digitalWrite(pin, HIGH) y puedes llegar a cualquier punto entremedias.

**Nota: Ten cuidado con la diferencia entre analogRead y analogWrite: analogRead tiene 1024 niveles de lectura, mientras que analogWrite sólo tiene 256. analogRead utiliza Pins analógicos, mientras que analogWrite utiliza Pins digitales tipo PWM.**

Veamos un ejemplo. Coge un LED y una resistencia de 220 ​​Ohmios (con líneas de color rojo, rojo, marrón). Conecta el Pin largo del LED a la resistencia y el otro Pin de la resistencia al pin digital 10. Conecta el brazo corto del LED a GND.





No es necesario llamar a la función interna pinMode cuando utilizas analogWrite. Así que puedes dejar la configuración vacía aquí.

Después de cargar el código de abajo, verás que el LED se va iluminando poco a poco hasta llegar a su máxima intensidad, y luego se repite.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | int ledPin = 10;  int fade = 0;    void setup() {    // nothing here  }    void loop() {    analogWrite(ledPin, fade);    delay(10);    fade = fade + 10;    if (fade > 255) fade = 0;  } |

Aquí se ha utilizado un nuevo comando:

* analogWrite(pinNumber, fadeLevel): escribe una señal analógica a un Pin PWM. pinNumber es el número de Pin PWM que estás utilizando, fadeLevel es un número entre 0 y 255. 0 es igual a digital LOW, 255 es igual a digital HIGH.

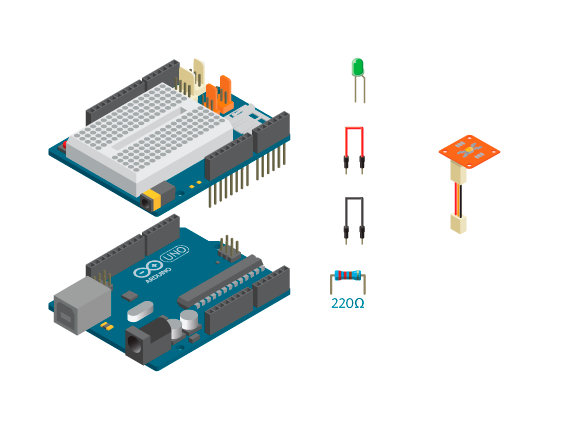
## ¡Sigue experimentando!

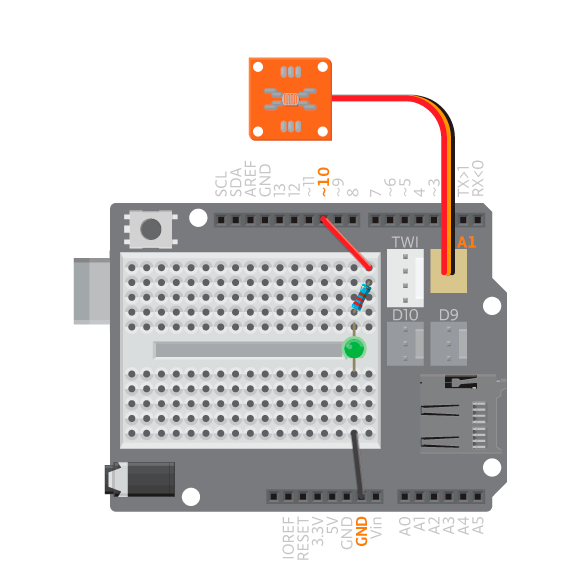
1. Intenta hacer que el LED se desvanezca a negro después de que alcance el brillo completo, en lugar de quedarse oscuro de repente. A esto lo llamamos una luz de respiración (breath).
2. ¿Se puede cambiar la velocidad de la “respiración” para que se desvanezca más rápido o más lento?
3. ¿Se puede utilizar un potenciómetro para controlar el LED? Recuerda la diferencia entre analogRead() y analogWrite().

# LDR

Ahora vamos a aprender acerca de un sensor analógico real. El LDR, es una resistencia dependiente de la luz (sus siglas provienen del inglés Light Dependent Resistor), detecta la luminosidad y según la cantidad de luz que le llegue, el sensor devolverá un valor analógico diferente. Puedes hacer cosas muy interesantes con ella, como una lámpara que se enciende automáticamente cuando la habitación se oscurece o que un robot siga una linterna… pero vamos a empezar por lo básico.

Consigue un sensor LDR TinkerKit y un cable TinkerKit y conéctalos entre sí. Luego conecta el otro extremo del cable al puerto analógico de tres pins TinkerKit A1 en la shield Básica Educativa.





Ahora coge un LED y una resistencia de 220 ohm, conecta el LED al Pin 10 como se describió en ejercicios anteriores.

Sube el código que sigue, y experimenta un poco con el sensor LDR. Apúntalo a la luz o cúbrelo con tu mano. Observa cómo se comporta el LED.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | int ledPin=10;  int ldrPin=A1;    void setup() {    //nothing here  }    void loop() {    int ldrValue=analogRead(ldrPin);    int ledValue=map(ldrValue,0,1023,0,255);      analogWrite(ledPin, ledValue);    delay(10);  } |

Los comandos usados son:

* map(valor, desdeBajo, desdeAlto, hastaBajo, hastaAlto): Reasigna un número de un rango a otro. valor es el valor que queremos reasignar.desdeBajo y desdeAlto son el valor mínimo y máximo que el  valor puede tomar. En este caso desde 0 a 1023. hastaBajo y hastaAlto son el mínimo y el máximo valor que queremos. En este caso, desde 0 hasta 255.

Utilizamos analogRead() para leer el valor del LDR, cuyo rango va de 0 a 1023. Entonces usamos la función map() para transformar dicho valor de forma que el nuevo sea proporcional a un valor entre 0 y 255. Si ldrValue es 1023, ledValue será 255, si ldrValue es 512 ledValue será 127. Una vez el valor haya sido reasignado, lo utilizamos para encender el LED. El LED se hará oscuro o luminoso dependiendo de la cantidad de luz que el LDR detecte.

## ¡Sigue experimentando!

* Intenta hacer la lámpara automática que hemos mencionado anteriormente. Cuando la lectura del LDR es más baja que un valor concreto o umbral, la luz se enciende. De lo contrario, se apaga. Utiliza un LED para simular la lámpara.
* ¿Recuerdas el ejemplo Beep de la semana pasada? ¡Haz que funcione con un LDR! Recuerda asignar los valores correctamente.

## Calibración de Sensores

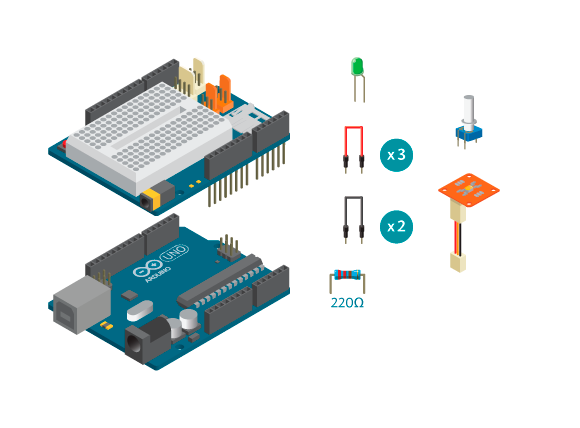
El LDR es un sensor fuertemente influenciado por configuraciones exteriores. Es decir, un proyecto con LDR funcionará de manera diferente en diferentes sitios porque la configuración lumínica es diferente. En ocasiones funcionará a la primera pero a veces tendremos que calibrarla para que funcione en lugares diferentes.

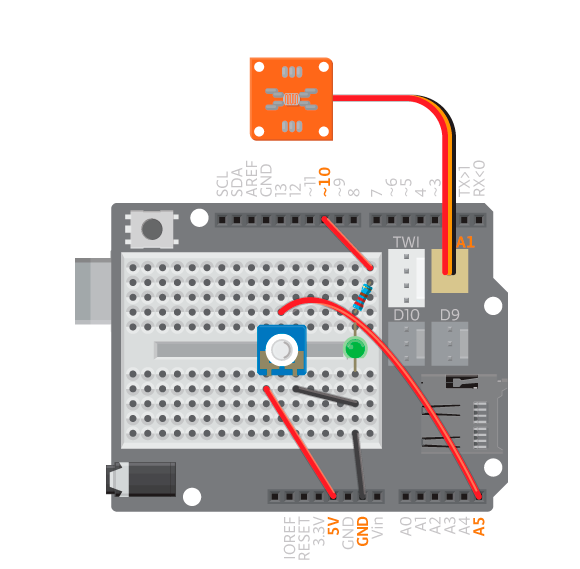
Coge la lámpara automática como ejemplo. Digamos que la queremos utilizar para otro propósito. Esta vez, se encenderá si le colocas un trozo de papel sobre la LDR.

Necesitarás experimentar un poco, intenta buscar el umbral para un trozo de papel. ¿Has conseguido que funcione? Muy bien, mueve el LDR por debajo de la mesa, ¿todavía funciona?

Si es que no, probablemente ya sabemos cuál es el problema. El papel no ha cambiado mucho la lectura de la LDR, mientras que la iluminación de tu habitación sí lo ha hecho (al mover el sensor bajo de la mesa). Por lo que necesitarás encontrar una manera mejor de cambiar el umbral dinamicamente.

Ahora busca un potenciómetro y conéctalo a la entrada analógica A5 como en ejercicios anteriores. Reasigna el valor del analogRead() del potenciómetro a 0~255 y utiliza esto como el umbral.





|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | int ledPin=10;  int ldrPin=A1;  int potPin=A5;    void setup() {    pinMode(ledPin,OUTPUT);  }    void loop() {    int ldrValue=analogRead(ldrPin);    int threshold=analogRead(potPin);      if(ldrValue>threshold){      digitalWrite(ledPin,LOW);    }    else{      digitalWrite(ledPin,HIGH);    }      delay(10);  } |

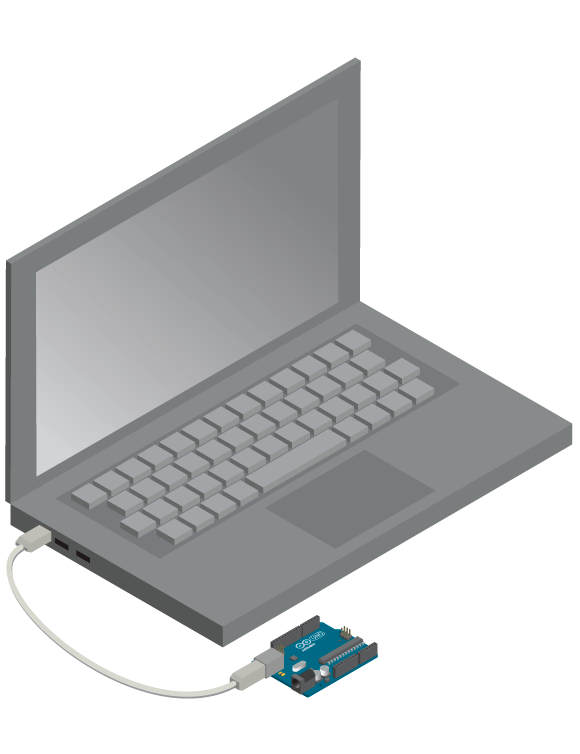
Ahora, en lugar de ajustar el umbral en el código, puedes hacerlo girando el potenciómetro hasta que el LDR detecte el papel. Prueba a ponerlo otra vez bajo de la mesa y verás cómo de simple y práctico puede ser un potenciómetro que reasigne los valores.

Acabas de aprender una manera de calibrar sensores analógicos, ¡utilízalo cuando construyas tus propios proyectos!

# ﻿Puerto serie

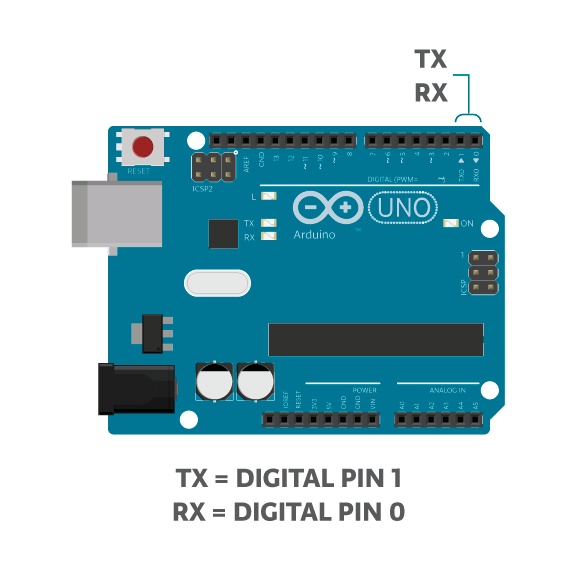
Las placas Arduino se conectan a tu ordenador usando un cable USB. El modo en que las placas “hablan” con el ordenador consiste en una técnica llamada *Puerto Serial* o *Puerto Serie*. Este se puede usar para intercambiar datos relativamente complicados entre Arduino y el ordenador. En lugar de señales digitales o analógicas puedes enviar o recibir texto (string).

Mediante comunicación serie puedes comunicarte también con otros programas. Puedes por ejemplo, utilizar Arduino para leer el estado de un botón y mandar los datos a un sketch de processing que cambie el color de la pantalla cuando el botón esté presionado.



El puerto serie usa los Pins digitales 0 y 1:

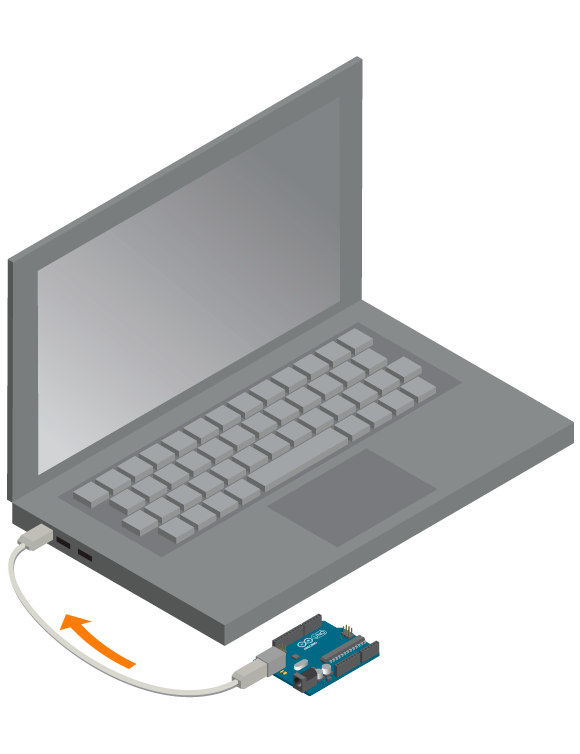
* El Pin 0, RX o recepción, por aquí llegan los datos a Arduino.
* El Pin 1, TX o transmisión, por aquí salen los datos desde Arduino.

Por lo tanto, no uses la función digitalRead() ni digitalWrite() en esos Pins si vas a utilizar comunicación serie.  


# Enviando al ordenador

## Instrucciones

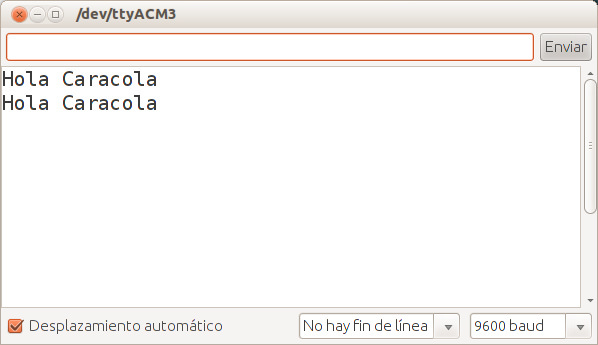
Para enviar un mensaje al ordenador, necesitarás tres comandos diferentes: Serial.begin(), Serial.println() o Serial.print().



Carga el código de abajo en tu placa y haz click en Herramientas -> Monitor Serial en el IDE de Arduino.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | void setup() {    Serial.begin(9600);  }    void loop() {    Serial.println("Hola Caracola");    delay(1000);  } |

Verás una pequeña ventana con el texto “Hola Caracola”, apareciendo una vez por segundo. Esta ventana muestra todo lo enviado a tu ordenador a través del puerto serial de Arduino.

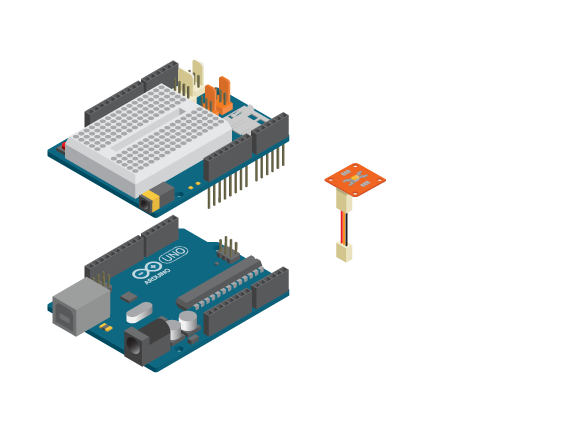


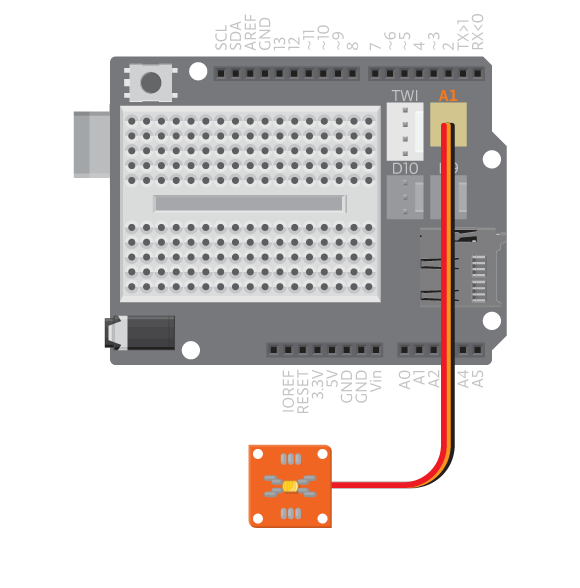
Los comandos utilizados arriba son:

* Serial.begin(velocidad): Inicia la comunicación serial. Velocidad ​​es lo rápido que los mensajes son transmitidos. Es importante que tanto el emisor (en este caso la placa Arduino) y el receptor (en este caso el monitor serial) se comuniquen a la misma velocidad. De otra manera, sería como dos personas que intentan hablar, uno chino y el otro en español. Por ahora, utiliza 9600. Esta es la velocidad más común cuando se trabaja con Arduino y Processing.
* Serial.println(mensaje): Muestra un mensaje a través del puerto serial, con un salto de línea al final. Así que cada vez que llames a la función Serial.println(), el mensaje siguiente se iniciará en una nueva línea. El mensaje es una cadena de texto (tipo String), reemplázala con el texto que quieras enviar.
* Serial.print(mensaje): Muestra un mensaje a través del puerto serial. Así que cada vez que llames a Serial.print(), el siguiente mensaje aparecerá justo después, sin nueva línea. El mensaje es una cadena de texto reemplázala con el texto que quieras enviar.

## Enviando valores del LDR

La cosa más importante en la que utilizarás la comunicación serial es para testear tus programas. De esta manera, más que mandar mensajes estáticos por el puerto serial, puedes mandar valores dinámicos que cambien con el tiempo. Esto es útil cuando quieras usar un sensor analógico pero que no sabes exactamente qué valores lee.





¿Te acuerdas del ejemplo anterior donde utilizábamos un potenciómetro para calibrar nuestro programa LDR? En lugar de ello, ahora vamos a mostrarte cómo usar la comunicación serial para hacerlo.

En este ejemplo leeremos el valor analógico del LDR e imprimiremos el valor por el monitor serial. Conecta un TinkerKit LDR al puerto de tres pines A1. Carga el siguiente código y abre el monitor serial.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | void setup() {    Serial.begin(9600);  }    void loop() {    int sensorValue = analogRead(A1);      Serial.println(sensorValue);    delay(100);  } |

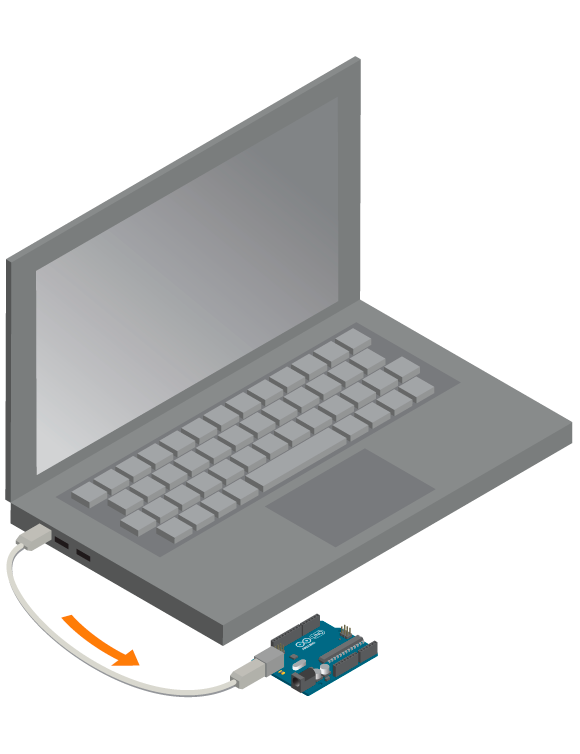
No se necesita ningún nuevo comando aquí. Leemos el valor del LDR conectado al puerto A1 e imprimimos dicho valor por el monitor serial. El delay() está aquí sólo para hacer más fácil la lectura de los datos. Cuando tengas una buena idea sobre qué valores obtendrás, sabrás que valor utilizar como umbral.

**Nota: Acuérdate de que el conector de tres pins A1 está realmente conectado al pin A1, por lo que no puedes usar ningún otro sensor que ese pin analógico.**

﻿

# Recibiendo del ordenador

Puedes utilizar el comando Serial.read() para obtener datos que llegan a Arduino a través del puerto serial.



El siguiente ejemplo muestra cómo utilizar el ordenador para controlar un LED con Arduino. Utilizaremos el on-board LED, por lo que todo lo que necesitas es cargar este código:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | int ledPin=13;  int incomingByte;    void setup() {    Serial.begin(9600);    pinMode(ledPin,OUTPUT);  }  void loop() {    if(Serial.available()>0){      incomingByte=Serial.read();        if(incomingByte=='H'){        digitalWrite(ledPin, HIGH);      }      if(incomingByte=='L'){        digitalWrite(ledPin,LOW);      }      }  } |

Abre el monitor del puerto serie desde Herramientas -> Monitor Serial o con el icono al final de la barra de tareas. Hay un cuadro de entrada de texto con el botón enviar al lado. Escribe H en el cuadro de entrada y haz clic en el botón enviar. Deberías ver que se enciende el LED conectado al Pin 13 de tu Arduino. Teclea L y envía. El LED se apagará. Asegúrate de utilizar mayúsculas.

Los comandos utilizados son:

* Serial.available(): Comprueba si hay señales procedentes del puerto serial. Devuelve bien true (verdadero) o false (falso). Lee las señales sólo cuando estén disponibles.
* Serial.read(): Lee un byte del puerto serie.

Sólo queremos leer del puerto serial cuando haya datos entrantes. Por tanto, primero comprobamos con Serial.available si hay algo que leer. En ese caso, leemos con Serial.read() y lo guardamos en incomingByte. Luego comprobamos si incomingByte es igual a ‘H’ o ‘L’ y encendemos o apagamos el LED conforme a esto.

## ¡Sigue experimentando!

1. Consulta cualquiera de los ejemplos en Archivo –> Ejemplos –> Castilla –> Help, casi todos hacen uso del puerto serie. Por ejemplo, controla el programa breath light con las señales del puerto serial.
2. Si dispones de dos baterías de 9V, haz que un Arduino controle al otro. El Arduino que recibe las señales debe tener el código anterior. El Arduino que envía debe mandar las señales “H” o “L” alternativamente.