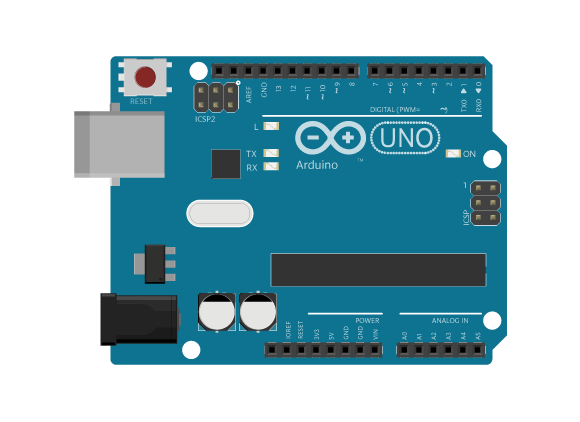
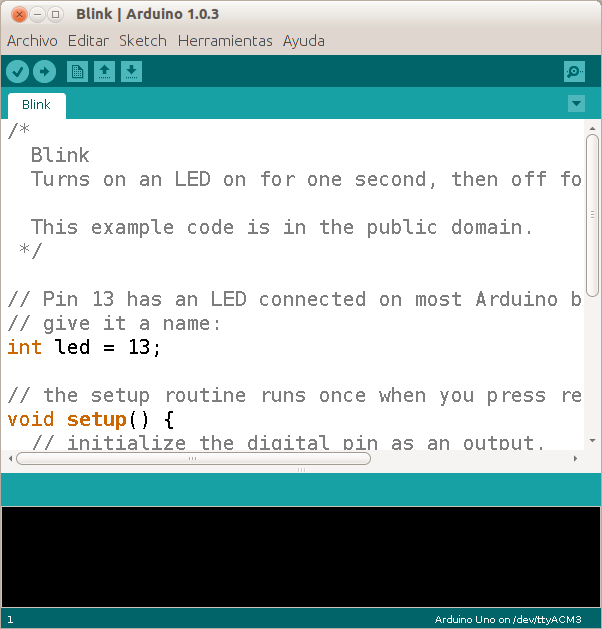
Qué es Arduino

Las placas Arduino son pequeños ordenadores con los que puedes leer información de diferentes sensores, así como controlar luces, motores y muchas otras cosas. La gran mayoría de los sistemas que nos rodean son ordenadores de diferentes tamaños. Los ordenadores no necesitan tener teclado ni pantalla. Hay ordenadores en el microondas de la cocina, dentro de los ascensores para detectar qué botón pulsas y, en los coches. Hay más de 70 ordenadores similares a Arduino… hay ordenadores por todas partes.



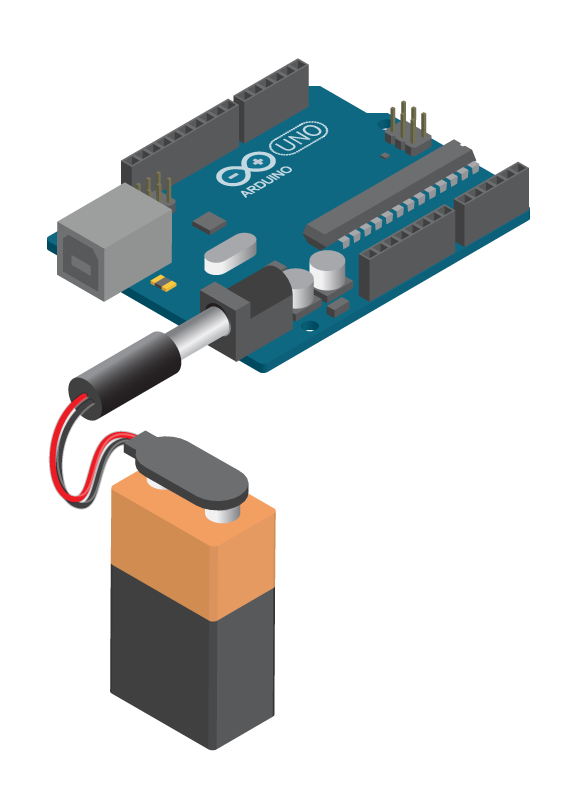
Puesto que Arduino, a diferencia del ordenador que usas normalmente, no tiene pantalla ni teclado, se necesita un programa externo ejecutado en otro ordenador para poder escribir programas para la placa Arduino. Éste software es lo que llamamos Arduino IDE. IDE significa “Integrated Development Environment” (Entorno de Desarrollo Integrado), y es un término común para llamar a este tipo de desarrollo de software. Escribes tu programa en el IDE, lo cargas en el Arduino, y el programa se ejecutará en la placa.



El IDE de Arduino es muy sencillo y parecido a Processing. Hay una sencilla barra de herramientas que puedes utilizar para:

* https://farm6.staticflickr.com/5502/12163728663_46829b344c_o.png Verificar si tu programa va a funcionar.
* https://farm4.staticflickr.com/3668/12163909654_851025fa23_o.png Cargar el programa a la placa de Arduino.
* https://farm8.staticflickr.com/7369/12164151396_4bc17181be_o.png Crear un programa nuevo.
* https://farm8.staticflickr.com/7396/12164151796_6e9e4ba2ab_o.png Abrir un programa.
* https://farm6.staticflickr.com/5521/12163910454_a4170cbe2c_o.png Guardar el programa en el disco duro del ordenador.
* https://farm8.staticflickr.com/7373/12163730443_f52ca8b9e0_o.png (En la parte derecha de la barra de herramientas se encuentra el Monitor Serial) abre una ventana de comunicación con la placa Arduino.

Las placas Arduino se conectan a tu ordenador utilizando un cable USB, al igual que cualquier otro periférico, como la impresora, el teclado o incluso, un mando de videojuegos. Arduino necesita estar conectado al ordenador a través del cable USB para cargar un programa. El cable USB sirve también para suministrar energía a la placa, pero también puedes alimentarla usando una fuente de energía externa, como una batería o un transformador apropiado.



Cuando programes tu Arduino, debes asegurarte de que el IDE está configurado correctamente para la placa que estés utilizando. Compruébalo en el menú ‘Herramientas → Puerto Serial’ para ver que el puerto esté configurado correctamente, y en ‘Herramientas → Tarjeta’ para comprobar que esté asignado ‘Arduino Uno’.

**Nota: para saber a qué Puerto Serial está conectado tu Arduino, sigue estos pasos:**

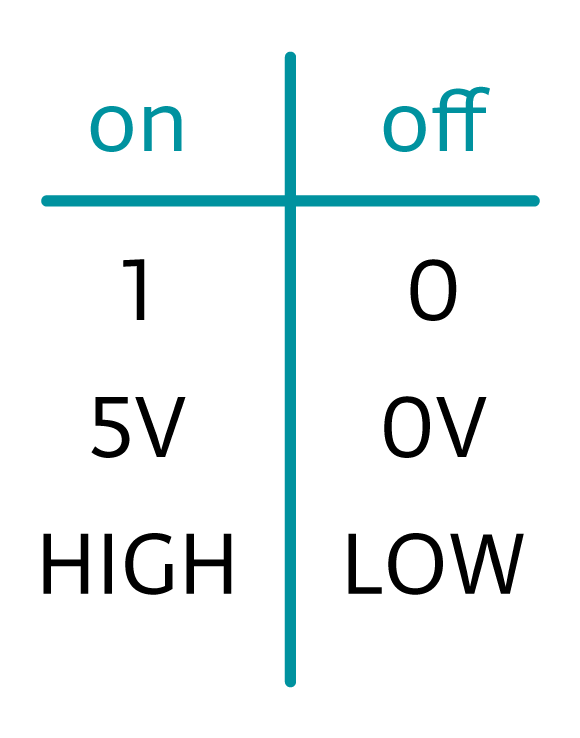
1. Desconecta Arduino del ordenador.
2. Comprueba ‘Herramientas → Puerto Serial’ para ver si está disponible.
3. Conecta Arduino al ordenador.
4. Entra en ‘Herramientas → Puerto Serial’ de nuevo y verás un puerto nuevo, ése es tu Arduino.
5. Selecciona ese puerto.

# Señales digitales

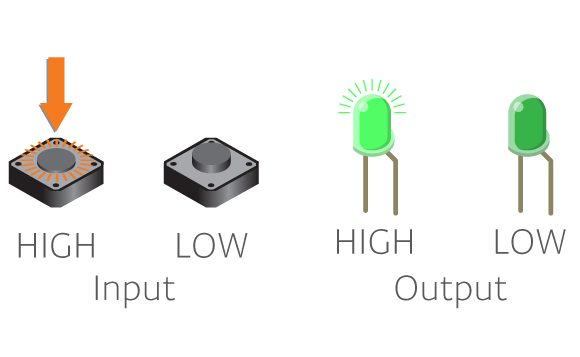
El alfabeto Español consta de 27 símbolos, más otros tantos para las mayúsculas, y 10 símbolos para los números. A diferencia de los humanos, los ordenadores trabajan y se comunican con sólo 2 símbolos; ’1′ y ’0′. Esto es lo que llamamos señales digitales. Usando una combinación de éstos símbolos, las máquinas digitales puedes representar todo lo que hay en el Universo.

Una característica común a todos los ordenadores es que utilizan la lógica binaria. Esto quiere decir que pueden solucionar múltiples operaciones utilizando solo dos símbolos básicos para representar toda la información. Los seres humanos, por ejemplo, utilizamos muchos símbolos. El abecedario español, tiene 27 símbolos en minúscula, más las mismas en mayúscula, además de 10 símbolos numéricos. Los ordenadores solo utilizan dos: ’1′ y ’0′. Combinando estos dos números, las máquinas digitales pueden representar prácticamente todo lo que hay en el Universo.

Arduino representa un ’1′ con 5 Voltios, y un ’0′ con 0 Voltios. Cuando escribes un programa para Arduino, tú representas un ’1′ escribiendo HIGH y ’0′ escribiendo LOW. Cuando compilas el código, lo que realmente pasa es que lo que has escrito con símbolos humanos se traduce a unos y ceros, el lenguaje que los ordenadores entienden.



Al hablar de señales digitales y Arduino, hablamos de entradas y salidas. Una entrada digital significa que Arduino está recibiendo datos digitales de un sensor, por ejemplo, un botón. Cuando leemos desde un botón, Arduino recibirá bien 5V, HIGH o 0V, LOW, dependiendo si el botón está pulsado o no. Una salida digital significa que Arduino está mandado datos digitales a un actuador, como por ejemplo, un LED. Para encender un LED, Arduino manda 5V, HIGH; y para apagarlo, manda 0V, LOW.



## Lógica binaria

Otra característica común en todos los ordenadores es que utilizan lógica binaria. Lógica binaria significa que sólo hay dos posibilidades. Puesto que un ordenador utiliza sólo dos símbolos, ’0′ y ’1′, decimos que ellos también emplean la lógica binaria. En este caso, ’1′ representa TRUE, mientras que ’0′ representa FALSE. Esto es así porque la lógica binaria también puede ser usada para hacer preguntas como ‘¿Ha alcanzado la temperatura 20 grados?’, la respuesta es o bien cierta, o bien falsa, y por tanto puede ser representada por ’0′ ó ’1′.

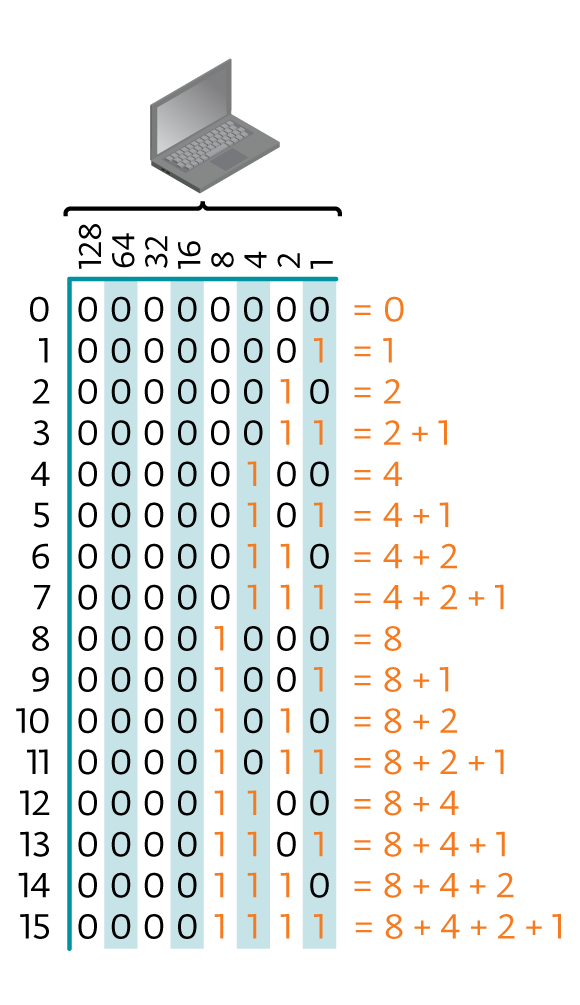
Cuenta en binario

Los seres humanos tenemos un total de 10 símbolos para representar los números: del 0 al 9. Combinándolos, podemos representar cualquier número – 13, 648, 2015, etc. De la misma forma que cualquier cosa en el Universo puede ser representada por unos y ceros. Sólo se necesita un sistema para combinarlos.

Imagina cuatro interruptores que pueden estar encendidos o apagados. Cada interruptor tiene su propio valor, ’8′,’4′,’2′ y ’1′. Cuando todos están apagados, representan ’0′ – 0000. Para representar el número uno, simplemente encendemos el interruptor ’1′ – 0001. Para representar el número dos, encendemos el interruptor del ’2′ – 0010. Ahora piensa cuidadosamente – ¿Cómo puedo representar el número tres? Basta con encender el interruptor del ’1′ y el del ’2′ – 0011; ya que 2 más 1 son 3.

¿Empiezas a entender el funcionamiento detrás del sistema binario? Probemos a obtener el número siete. Encendemos los interruptores ’4′, ’2′ y ’1′ – 0111. ¿Y para el número 11? Encendemos los interruptores ’8′, ’2′ y ’1′ – 1011.

Mira en la siguiente tabla para entender como funciona:



﻿

﻿

# Blink

Durante la introducción a la programación vimos como hacer pequeños programas y animaciones usando el ordenador. Encendíamos y apagábamos los píxels en la pantalla del ordenador. Como sabes, la placa Arduino no tiene pantalla, pero tiene un LED – una pequeña lámpara que puede encenderse y apagarse fácilmente usando un programa. Se puede decir que Arduino viene con una pantalla de un solo pixel. Ese LED en placa, está conectado al Pin digital 13. Como puedes ver en la placa, todos los pins están numerados y agrupados por funcionalidad. Hay un grupo de 14 pins (numerados del 0 al 13) que son pins digitales y luego otro grupo de 6 pins (etiquetados de A0 a A5) que son los analógicos. Más adelante explicaremos los pins analógicos, ahora nos centraremos en los digitales. Veamos cómo controlar el LED en tu Arduino utilizando un comando sencillo. El primer ejemplo es el que llamamos Blink, que significa encender y apagar el LED repetidamente. Al igual que los programas de Processing que siempre necesitan tener una función setup() y una función draw(), los programas Arduino necesitan las funciones setup() y loop():

* setup(): Esta parte del programa sólo se ejecuta al principio. Aquí podrás configurar las funcionalidades de los pins, ya sean inputs (entradas) u outputs (salidas), o si van a realizar una función más compleja como enviar señales a través del cable USB.
* loop(): Esta parte funcionará infinitamente (o hasta que desconectes la fuente de alimentación). Los comandos en el loop serán ejecutadas en orden, una tras otra. Cuando llegamos al último comando, comenzará de nuevo desde el principio.

El programa Blink más básico funciona de la siguiente manera:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | void setup() {    pinMode(13, OUTPUT);  }    void loop() {    digitalWrite(13, HIGH);    delay(1000);    digitalWrite(13, LOW);    delay(1000);  } |

Aquí puedes ver tres comandos diferentes:

* pinMode(pinNumber, INPUT | OUTPUT | INPUT\_PULLUP): Se utiliza para determinar si el Pin digital en tu Arduino está escribiendo (OUTPUT) o leyendo (INPUT | INPUT\_PULLUP) las señales desde/hacia el entorno.
* digitalWrite(pinNumber, HIGH | LOW): Se utiliza para hacer que un Pin digital concreto escriba 5 voltios(HIGH) ó 0 voltios (LOW) a una salida.
* delay(time): Detiene el programa durante cierta cantidad de tiempo. El tiempo se expresa en milisegundos. Si quieres detener el programa durante 2 segundos, deberías escribir delay(2000).

Recordemos también lo que vimos durante la semana de Processing:

* Cada línea termina con punto y coma ‘;’.
* Los bloques de código están contenidos entre llaves ‘{ }’.
* Las funciones comienzan con una definición como void.
* Las funciones tienen parámetros que figuran entre paréntesis ‘( )’, y pueden tener tantos parámetros como necesites.

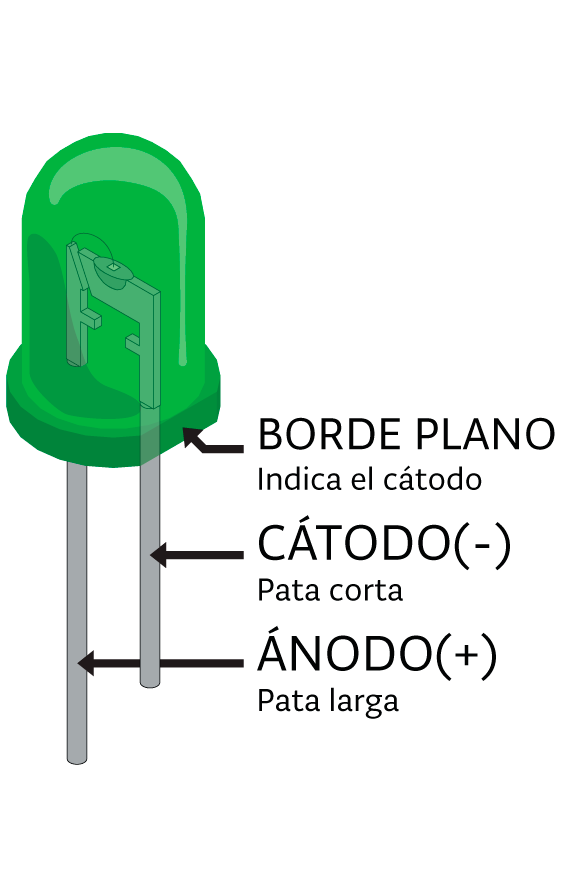
Pero volviendo a lo que hace el programa, éste enciende el LED marcado con la ‘L’ en tu Arduino, espera 1 segundo, lo apaga y espera otro segundo. Luego esto lo repite una y otra vez hasta el infinito. Cárgalo en la placa utilizando la flecha derecha (el segundo botón de la barra de herramientas) y comprueba cómo funciona.

## ¡Sigue experimentando!

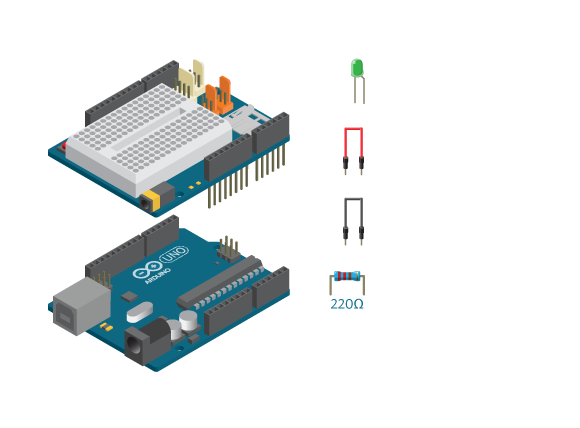
* Ahora puedes probar a ver qué sucede cuando cambias los tiempos dentro de la función delay. ¿Qué ocurre si haces los delays más pequeños?, ¿la luz parpadea más rápido o más lento?
* También puedes utilizar la luz para simular los latidos del corazón. Comprueba tus latidos midiendo tu pulso, notarás que siempre obtienes dos pulsaciones, algo así como tock tock… pausa… tock tock… pausa… ¿Puedes simularlo con un LED? (SUGERENCIA: puedes copiar y pegar el código).
* Por último, ¿qué sucede cuando el delay se convierte a un intervalo verdaderamente pequeño? Digamos que lo cambias a delay(10), ¿todavía puedes ver el parpadeo de la luz?

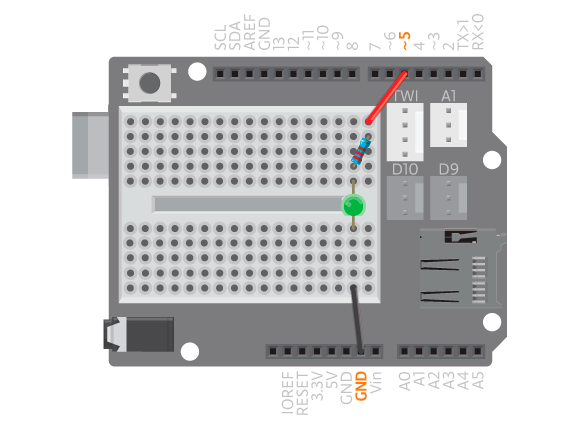
## Añádele tus propios LEDs

¿Quieres añadirle al circuito tus propios LED? Cuando trabajas con un LED, es importante que recuerdes que tienen polaridad. Los LED están construidos para mostrarla. El Pin largo, se denomina ánodo, y es el positivo, cuando lo conectamos al circuito, éste debe estar conectado al extremo positivo del alimentador. El Pin corto, o cátodo, debe estar conectado a tierra (0 voltios).



Los LED no se romperán si los conectas con la polaridad opuesta, simplemente no funcionarán. Sin embargo, por los LED no puede circular mucha corriente, para asegurarte de que no se quemen, debes poner una resistencia delante de ellos. El valor de la resistencia suele variar, en nuestro caso, utilizaremos 220 ohm para los LED. Para hacer más fácil la construcción de circuitos, usaremos la Shield Básica Educativa. Una vez montada en tu Arduino, la shield amplia las capacidades de la placa, además de añadirle características extra. Para hacer el circuito con un LED, haremos uso de la breadboard.





Si haces un circuito como el que hemos mostrado anteriormente, tendrás que cambiar el programa Blink para direccionarlo al Pin número 5 en vez de al Pin número 13. El programa puede parecerse al siguiente:

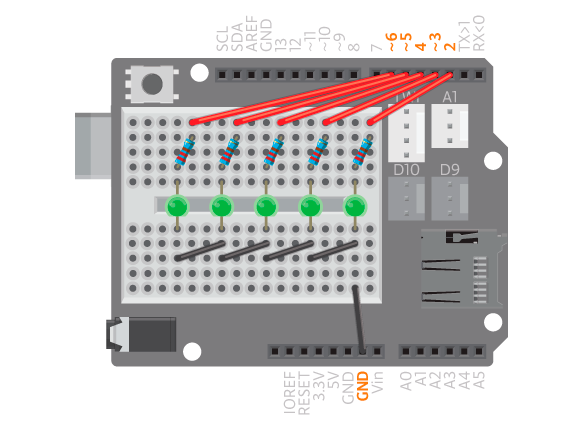
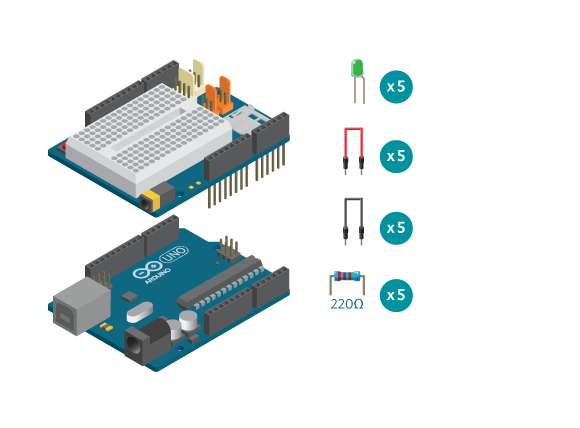
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | void setup() {    pinMode(5, OUTPUT);  }    void loop() {    digitalWrite(5, HIGH);    delay(1000);    digitalWrite(5, LOW);    delay(1000);  } |

También puedes utilizar variables que contengan el número del Pin, así será más fácil cambiar el programa en caso de que sea necesario.

Puede que recuerdes las variables de la primera semana del curso. Sólo recordarte que las variables son lugares dentro de la memoria del ordenador donde podemos almacenar números, caracteres o incluso cadenas de texto (strings) completas. La variables tienen un tipo (como int para número y strings para cadenas de texto) y un nombre al que le puedes asignar lo que quieras. En este caso, queremos que una variable almacene el número que representa el Pin que estamos cambiado con nuestro programa. Escribiendo el programa de esta forma, sólo tienes que reemplazar el número del Pin en un único lugar del código, en lugar de en tres si decidiéramos conectar el LED en otro pin de nuevo. Veamos cómo sería:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | int ledPin = 5;    void setup() {    pinMode(ledPin, OUTPUT);  }    void loop() {    digitalWrite(ledPin, HIGH);    delay(1000);    digitalWrite(ledPin, LOW);    delay(1000);  } |

ledPin es una variable. En este caso, los valores posibles van de 0 a 13: cualquiera de los pins digitales de la placa Arduino.

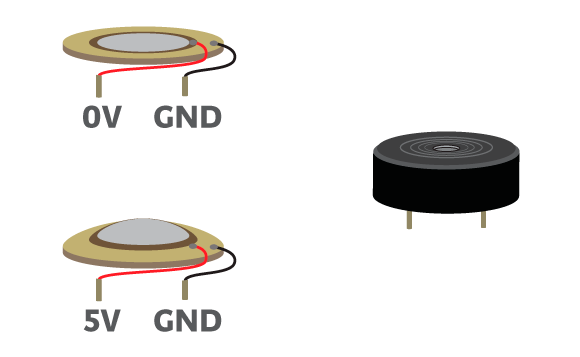
Por supuesto, es posible conectar y controlar más de un solo LED a tu Arduino. En los proyectos de esta semana verás que al usar varios LEDs obtenemos lo que llamamos VU-Metro. Un VU-Metro es una linea de LEDs agrupados. Existen varias funciones que hemos preparado para que puedas controlar muchos LEDs más fácilmente.

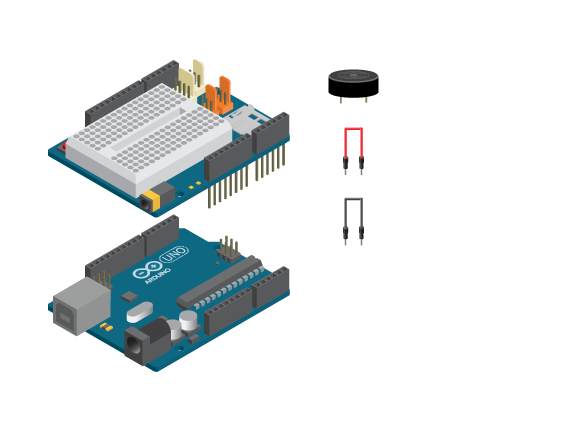
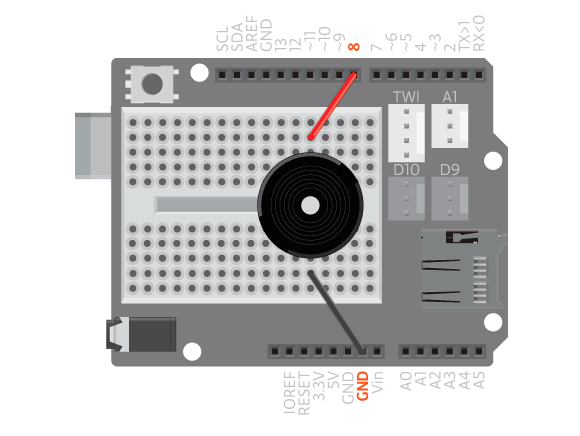
**Nota: Te recomendamos que no utilices los pines 0 ni 1 a no ser que realmente los necesites. Estos pines se utilizan para que Arduino envíe información al ordenador, si les conectas cualquier cosa, Arduino no será capaz de comunicarse y por lo tanto, no será posible reprogramarlo.**

# Beep

Vamos a jugar un poco con sonidos. Como probablemente ya sabrás, el sonido son vibraciones. El sonido de una guitarra por ejemplo, se produce por las vibraciones de las cuerdas. Por lo tanto, para producir sonido con Arduino necesitamos algo que genere vibraciones. En el siguiente experimento haremos esto mediante un zumbador piezoeléctrico que pite.

## El Zumbador Piezoeléctrico

  
El micrófono de contacto, también conocido como piezo zumbador, es un componente electrónico formado a partir de la combinación de dos discos de distintos materiales. Uno de ellos es metálico y el otro, generalmente es de cerámica, y ambos tienen propiedades piezoeléctricas. Cuando se le aplica un voltaje al componente, los materiales se repelen produciendo un “click” audible (chasquido). Poniendo a cero la diferencia de tensión, hará que los materiales vuelvan a su posición inicial, produciendo de nuevo un sonido de “click”.

Conecta el zumbador al pin digital 8 y a tierra. No importa cual de los pines está conectado a que pin. Para hacer vibrar al piezoeléctrico ahora, necesitamos hacer que el pin digital cambie entre HIGH y LOW repetidamente, esto es, alternar entre 5V y 0V. Se parece bastante a hacer parpadear a un LED, ¿verdad?. De hecho, el programa para que el zumbador suene es muy parecido al ejemplo “Blink”.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | int speakerPin = 8;    void setup() {    pinMode(speakerPin, OUTPUT);  }    void loop() {    digitalWrite(speakerPin, HIGH);    delay(1000);    digitalWrite(speakerPin, LOW);    delay(1000);  } |

Como has podido comprobar, la única diferencia en este programa es que usamos la variable speakerPin en vez de ledPin.

Este programa hace que el material dentro del piezoeléctrico se repela y vuelva a su posición normal en un intervalo de 2 segundos. Lo que escuchas es un “click” cada segundo. Pero para poder generar un tono, necesitamos hacer que el piezoeléctrico oscile más rápido. Podemos conseguirlo simplemente reduciendo el tiempo de retraso. Prueba cambiando el delay a un milisegundo:

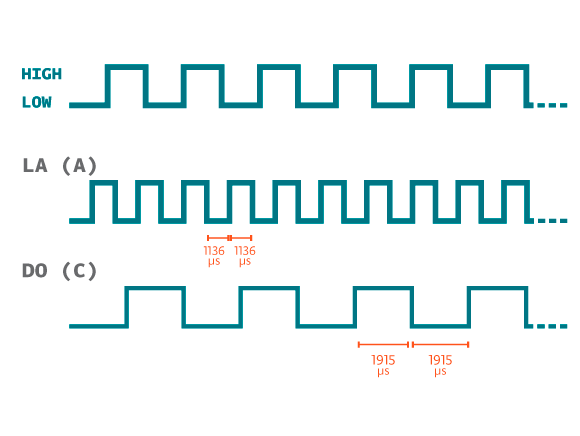
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | int speakerPin = 8;    void setup() {    pinMode(speakerPin, OUTPUT);  }    void loop() {    digitalWrite(speakerPin, HIGH);    delay(1);    digitalWrite(speakerPin, LOW);    delay(1);  } |

Si has conectado el zumbador correctamente, deberías estar escuchando un sonido muy fuerte saliendo de tu Arduino. La única manera de detenerlo es o bien desconectarlo de la placa, o quitando el cable del pin 8.

## Tonos

Ahora ya sabes como generar sonido con Arduino y un zumbador piezoeléctrico cambiando el pin entre ’0′ y ’1′ un determinado número de veces por segundo. Esta cantidad es la frecuencia, y obtendrás diferentes tonos dependiendo de su valor. A alta frecuencia, esto es, más oscilaciones por segundo, obtendrás sonidos muy agudos, mientras que a baja frecuencia, obtendrás sonidos más graves.

Los tonos tienen nombres: Do (C), Re (D), Mi (E), Fa (F), Sol (G), La (A), Si (B). De estos tonos, Do (C) tiene la frecuencia más baja a 262 Hercios, mientras que Si (B) tiene la más alta a 494 Hz. Esto significa que necesitamos hacer oscilar al pin 262 veces por segundo para que suene un Do (C) y 494 veces por segundo para que suene un Si (B).

  
Si quieres hacer sonar tonos diferentes, tienes que cambiar la cantidad de tiempo en que el pin está en ON y OFF. Basta con cambiar el valor de la función delay(1). Sólo hay un problema: para tocar los tonos, deben ser controlados con más precisión. En vez de utilizar la función delay, deberías usar la función delayMicroseconds. Como recordarás, el delay hace una pausa del orden de milisegundos, mientras que el delayMicroseconds la hace en microsegundos, es decir, 1000 veces menor que un milisegundo.

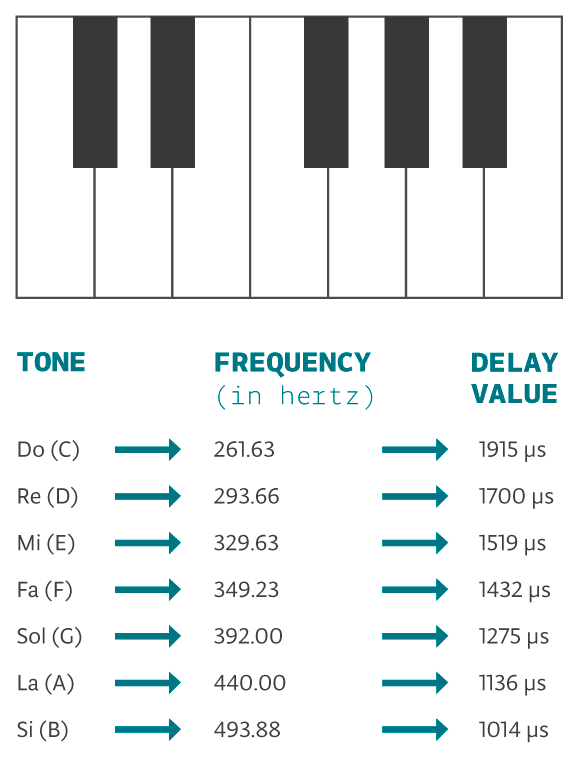
La siguiente ecuación describe la relación entre las dos funciones:

|  |  |
| --- | --- |
| 1 | delay(1) = 1000 \* delayMicroseconds(1) = delayMicroseconds(1000) |

Esto quiere decir que el programa que has utilizado para tocar Beep, también se puede escribir como:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | int speakerPin = 8;    void setup() {     pinMode(speakerPin, OUTPUT);  }    void loop() {     digitalWrite(speakerPin, HIGH);     delayMicroseconds(1000);     digitalWrite(speakerPin, LOW);     delayMicroseconds(1000);  } |

Esto nos muestra un aspecto básico de la programación: ¡puedes hacer lo mismo de distintas maneras! Este programa y el anterior hacen lo mismo. Si quieres que tu zumbador piezoeléctrico toque diferentes tonos, sólo tienes que cambiar el valor ’1000′ por los otros valores en el delay de la tabla de abajo.



## Melodías

Una melodía es una combinación de notas. Sin embargo, si quisiéramos tocar una melodía en un zumbador cómo hemos hecho en los ejemplos anteriores, no acabaríamos nunca. En lugar de tener que calcular el tiempo del delay dependiendo de la frecuencia, y decidir cuantas veces hacer oscilar al pin dependiendo de la duración de la nota, hemos hecho una función llamada play(). Puedes leer acerca de cómo usarla para tocar melodías [aquí](http://andalucia.verkstad.cc/es/course-literature/melodia/).

# Entradas digitales

Al igual que puedes asignar ’0′ ó ’1′ a un pin en Arduino, puedes leer el valor generado por un sensor que esté conectado a él. Al trabajar con pines digitales, utilizaremos sensores que solo pueden estar en ON y OFF.

Un simple cable es un buen ejemplo de un INPUT (entrada) digital. Puedes conectar un cable a, por ejemplo, el pin 5 y encenderlo conectando la otra parte del cable o bien a 5 voltios o a GND (0 voltios) en el conector de tu Arduino.

¡Prueba esto! Coge unos de los conectores de tu kit, conéctalo al pin 5 y programa el siguiente código en tu Arduino:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | int inputPin = 5;  int ledPin = 13;    void setup() {    pinMode(inputPin, INPUT);    pinMode(ledPin, OUTPUT);  }    void loop() {    if (digitalRead(inputPin) == HIGH) {      digitalWrite(ledPin, HIGH);    }    else {      digitalWrite(ledPin, LOW);    }  } |

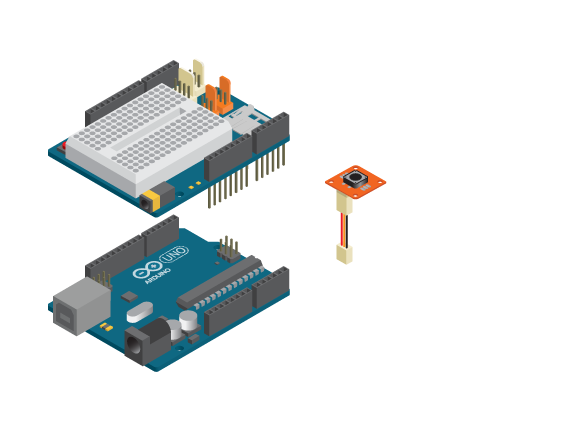
Al insertar el cable en el interior de la cabecera de 5 voltios, verás que el LED en el Pin 13 se enciende y, cuando lo insertas en la cabecera marcada como GND, el LED debería quedarse en OFF (se debería apagar).

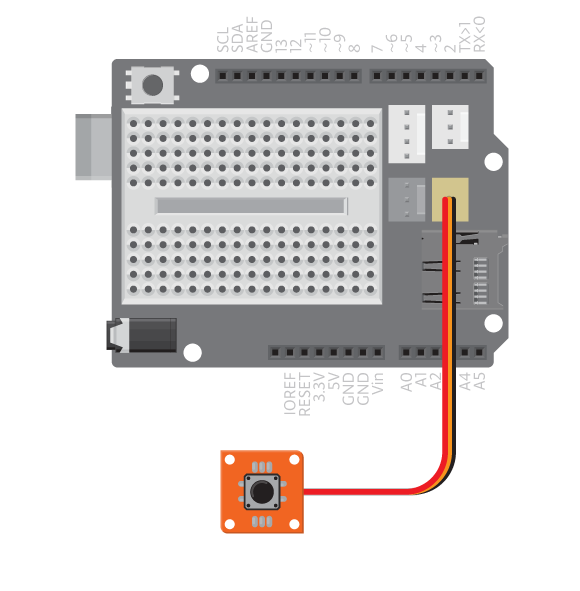
Este programa introduce algunas cosas nuevas:

* Primero, vemos que puedes utilizar muchas variables, en este caso, tenemos una para el LED ledPin y otra para el cable inputPin.
* Al igual que un pinMode se puede utilizar para asignar un pin como OUTPUT, puedes asignar el pin como INPUT.
* Hay un comando, digitalRead(inputPin), que leerá el valor del voltio en el inputPin y devolverá si está a HIGH o LOW. Devolverá HIGH cuando el cable esté a 5V y LOW cuando se conecte a GND.
* El programa utiliza una instrucción condicional if para comprobar el valor del inputPin.
* Cuando quieres comparar dos valores en un programa, debes utilizar dos veces el signo ‘igual a’, es decir, ==. Esto hace que el ordenador entienda que los quieres comparar (en lugar de asignar). Este “doble igual” es lo que llamamos un operador. Hay otros operadores que comparan otro tipo de valores.

## Botón

Vamos a probar en lugar de ello con un botón. Para este ejemplo necesitarás montar la Shield Básica Educativa en Arduino y conectar un botón tinkerKit al conector D9 con un cable tinkerKit.





Puedes usar el mismo código del ejemplo anterior, pero necesitarás cambiar el valor del inputPin a 9. Esto es porque el conector D9 está de hecho, conectado al pin 9.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | int inputPin = 9;  int ledPin = 13;    void setup() {    pinMode(inputPin, INPUT);    pinMode(ledPin, OUTPUT);  }    void loop() {    if (digitalRead(inputPin) == HIGH) {      digitalWrite(ledPin, HIGH);    }    else {      digitalWrite(ledPin, LOW);    }  } |

Como puedes ver, el LED se enciende cuando el botón está pulsado, y se apaga cuando lo sueltas.

## ¡Sigue experimentando!

* Escribe un programa que haga sonar un sonido al pulsar el botón.

**Nota: Recuerda que el conector D9 está conectado al pin digital 9. Si tienes algo a D9, no podrás usarlo para conectar cualquier otra cosa.**