Curso de supervivencia con ARDUINO.

Índice de contenido

Introducción

El objetivo de este pequeño panfleto no es otro que reivindicar dos cosas:

Desde el punto de vista de los profesores, el derecho a disfrutar con lo que hacemos, dando clases creativas e intentando insuflar a nuestros alumnos la ilusión y el gusto por la resolución de retos tecnológicos.

Desde el punto de vista de los alumnos, la necesidad de participar en clases interactivas, que saquen lo mejor de nosotros y que nos permitan explotar nuestra creatividad e ilusión.

Esto no es un curso de electrónica, yo personalmente, tengo más bien poca idea de lo que se sale de la electrónica digital más básica. Puedo decir con gran orgullo que muchos de mis alumnos, al terminar su segundo curso, saben más de electrónica que un servidor (En mi descargo decir que yo no doy clase en ningún módulo específico de electrónica ;))

Tampoco es un curso de programación, aquí sí que me puedo medir con algunos, disfruto desarrollando cosas en un número relativamente amplio de lenguajes y plataformas, pero esto tiene que ver sólo un poco con programación.

Esto es un curso de ARDUINO, si todavía no sabes lo que es ¡prepárate! Al final de estas páginas vas a estar haciendo que tus ideas interactúen con el mundo físico que te rodea. Arduino es … ¡La leche!

¿Qué es Arduino?

Es una plataforma para el desarrollo de productos electrónicos. Enfocada a un público no experto (artistas, entusiastas de la electrónica, …)

Es hardware libre, por tanto todos sus diseños son abiertos y pueden reutilizarse, mejorarse, …

Dispone de un IDE (Entorno de desarrollo) cómodo y flexible.

Todo lo que necesitas para empezar a desarrollar con Arduino lo puedes encontrar en la web: http://www.arduino.cc o en su versión en español: http://www.arduino.cc/es/

Si estás impaciente por empezar a desarrollar vete directamente a la sección de descargas de cualquiera de las dos y selecciona el ejecutable adecuado para tu plataforma: Linux, MacOS o Windows.

¿Para qué sirve Arduino?

Arduino es una placa con un microcontrolador de la marca Atmel y con toda la circuitería de soporte, que incluye, reguladores de tensión, un puerto USB (En los últimos modelos, aunque el original utilizaba un puerto serie) conectado a un módulo adaptador USB-Serie que permite programar el microcontrolador desde cualquier PC de manera cómoda y también hacer pruebas de comunicación con el propio chip.

Un arduino dispone de 14 pines que pueden configurarse como entrada o salida y a los que puede conectarse cualquier dispositivo que sea capaz de transmitir o recibir señales digitales de 0 y 5 V.

También dispone de entradas y salidas analógicas. Mediante las entradas analógicas podemos obtener datos de sensores en forma de variaciones continuas de un voltaje. Las salidas analógicas suelen utilizarse para enviar señales de control en forma de señales

PWM.

Características técnicas de un ARDUINO UNO.

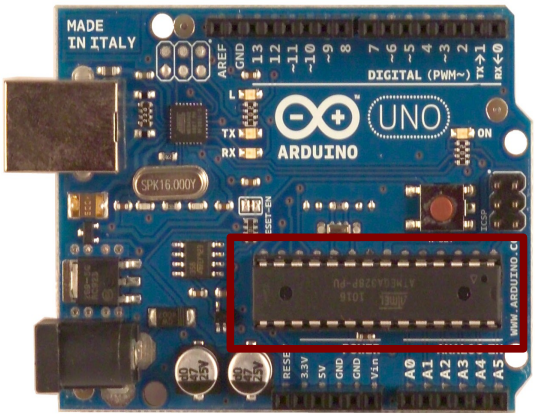


Ilustración 1: Arduino UNO con microcontrolador en formato DIP

Arduino UNO es la última versión de la placa, existen dos variantes, la Arduino UNO convencional y la Arduino UNO SMD. La única diferencia entre ambas es el tipo de microcontrolador que montan. La primera es un microcontrolador Atmega en formato DIP y la segunda dispone de un microcontrolador en formato SMD. Para entendernos, el formato DIP es mucho más grande que el formato SMD, que se suelda a la superficie de la placa.

Nosotros nos hemos decantado por la primera porque nos permite programar el chip sobre la propia placa y después integrarlo en otros montajes.

Si tu intención es usar directamente la propia placa en tus prototipos, cualquiera de las dos versiones es .

Resumen de características Técnicas:

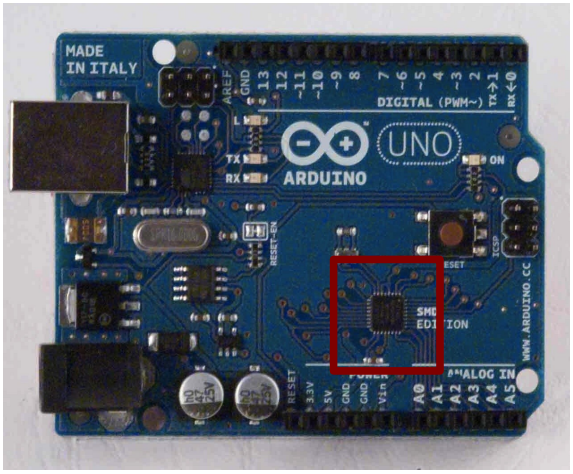


Ilustración 2: Arduino UNO con microcontrolador en formato SMD

Microcontrolador	Atmega328
Voltaje de operación	5V
Voltaje de entrada (Recomendado)	7 – 12V
Voltaje de entrada (Límite)	6 – 20V
Pines para entrada- salida digital.	14 (6 pueden usarse como salida de PWM)
Pines de entrada analógica.	6
Corriente continua por pin IO	40 mA
Corriente continua en el pin 3.3V	50 mA
Memoria Flash	32 KB (0,5 KB ocupados por el bootloader)
SRAM	2 KB
EEPROM	1 KB
Frecuencia de reloj	16 MHz

Entrada y salida:

Cada uno de los 14 pines digitales se puede usar como entrada o como salida. Funcionan a 5V, cada pin puede suministrar hasta 40 mA. La intensidad máxima de entrada también es de 40 mA.

Cada uno de los pines digitales dispone de una resistencia de pull-up interna de entre 20KΩ y 50 KΩ que está desconectada, salvo que nosotros indiquemos lo contrario.

Arduino también dispone de 6 pines de entrada analógicos que trasladan las señales a un conversor analógico/digital de 10 bits.

Pines especiales de entrada y salida:

- RX y TX: Se usan para transmisiones serie de señales TTL.
- Interrupciones externas: Los pines 2 y 3 están configurados para generar una interrupción en el atmega. Las interrupciones pueden dispararse cuando se encuentra un valor bajo en estas entradas y con flancos de subida o bajada de la entrada.
- PWM: Arduino dispone de 6 salidas destinadas a la generación de señales PWM de hasta 8 bits.
- SPI: Los pines 10, 11, 12 y 13 pueden utilizarse para llevar a cabo comunicaciones SPI, que permiten trasladar información full dúplex en un entorno Maestro/Esclavo.
- I^2C: Permite establecer comunicaciones a través de un bus I^2C. El bus I^2C es un producto de Phillips para interconexión de sistemas embebidos. Actualmente se puede encontrar una gran diversidad de dispositivos que utilizan esta interfaz, desde pantallas LCD, memorias EEPROM, sensores...

¿Cómo alimentar un Arduino?

Puede alimentarse directamente a través del propio cable USB o mediante una fuente de alimentación externa, como puede ser un pequeño transformador o, por ejemplo una pila de 9V. Los límites están entre los 6 y los 12 V. Como única restricción hay que saber que si la placa se alimenta con menos de 7V, la salida del regulador de tensión a 5V puede dar menos que este voltaje y si sobrepasamos los 12V, probablemente dañaremos la placa.

La alimentación puede conectarse mediante un conector de 2,1mm con el positivo en el centro o directamente a los pines Vin y GND marcados sobre la placa.

Hay que tener en cuenta que podemos medir el voltaje presente en el jack directamente desde Vin. En el caso de que el Arduino esté siendo alimentado mediante el cable USB, ese voltaje no podrá monitorizarse desde aquí.

Foto de familia: Las variantes de Arduino.

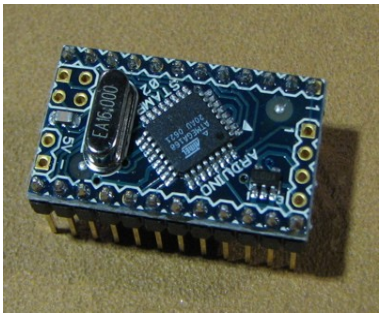


Ilustración 5: Arduino Mini

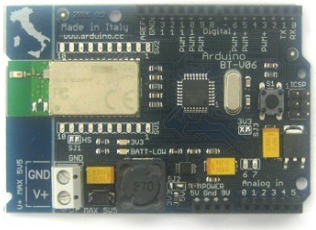


Ilustración 4: Arduino Bluetooth

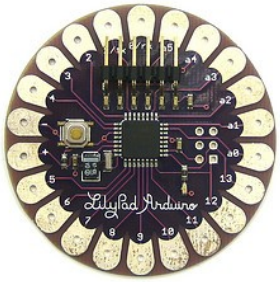


Ilustración 3: Arduino Lilypad

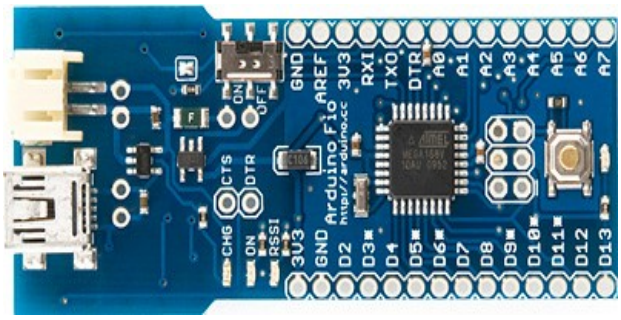


Ilustración 7: Arduino FIO

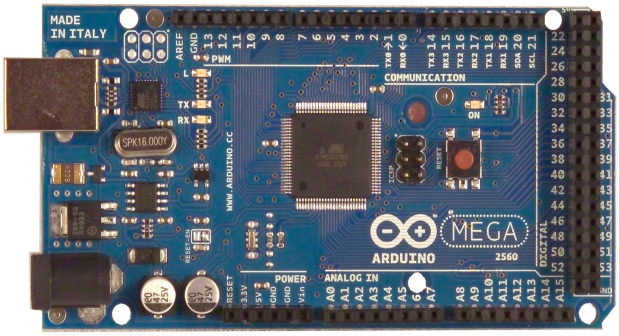


Ilustración 8: Arduino UNO MEGA

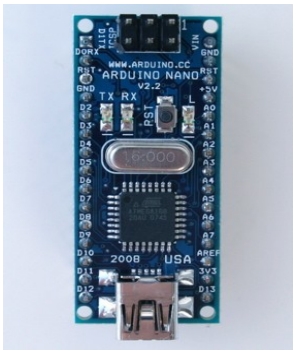


Ilustración 9: Arduino N

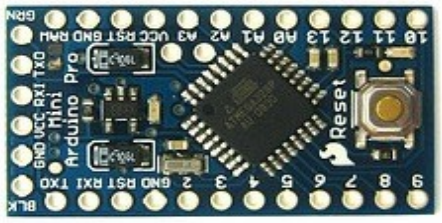


Ilustración 6: Arduino PRO Mini

Ilustración 10: Arduino PRO

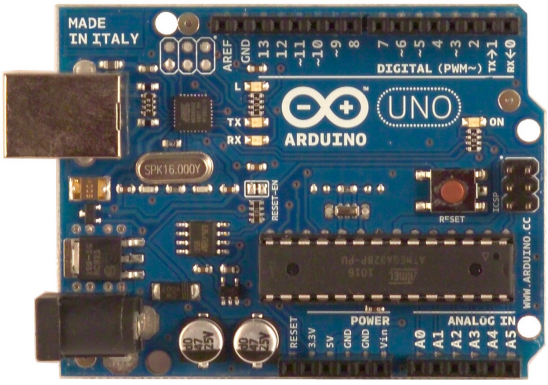


Ilustración 11: Arduino UNO

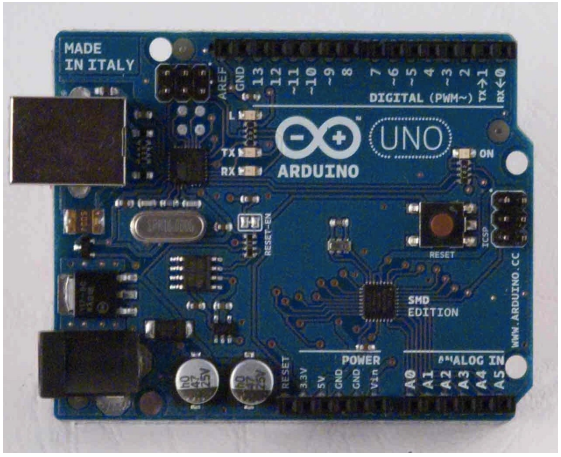


Ilustración 12: Arduino UNO SMD

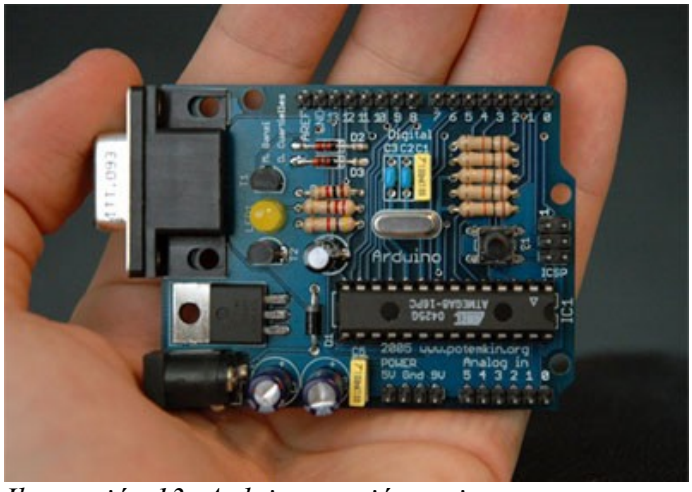
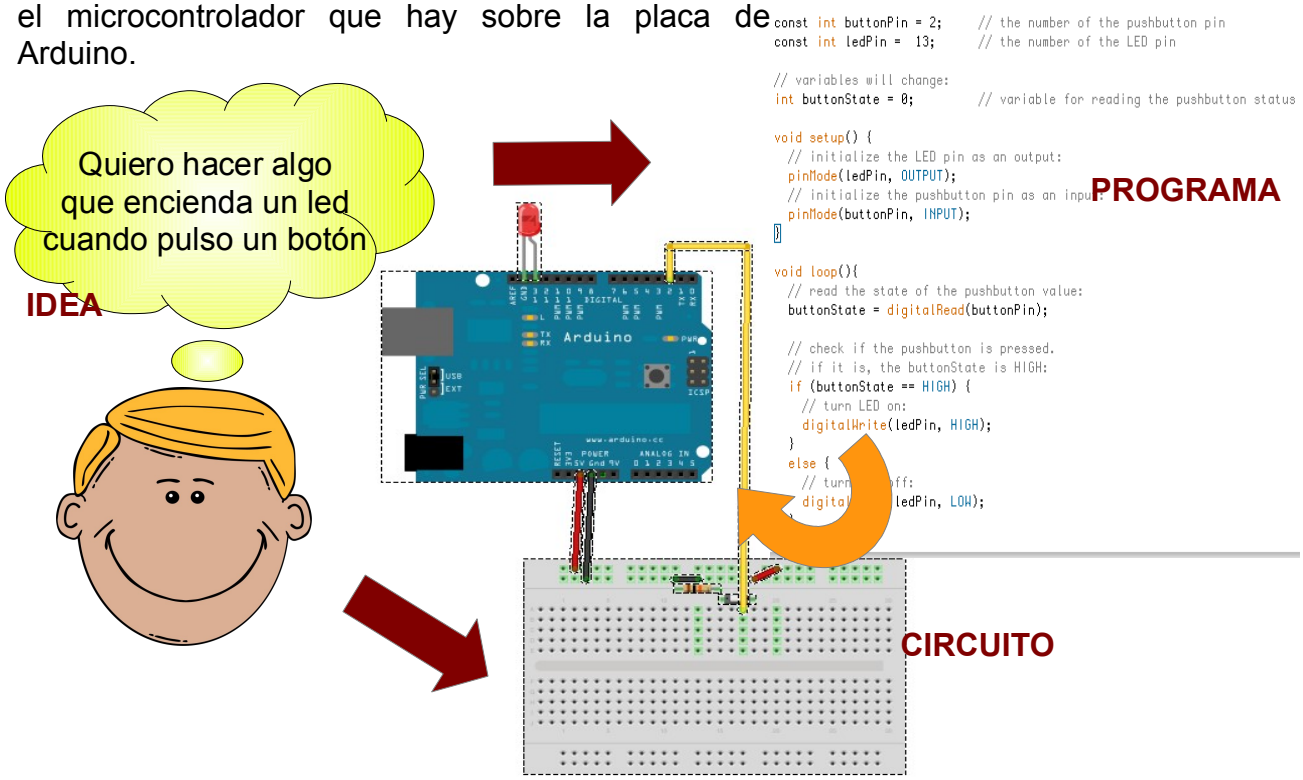


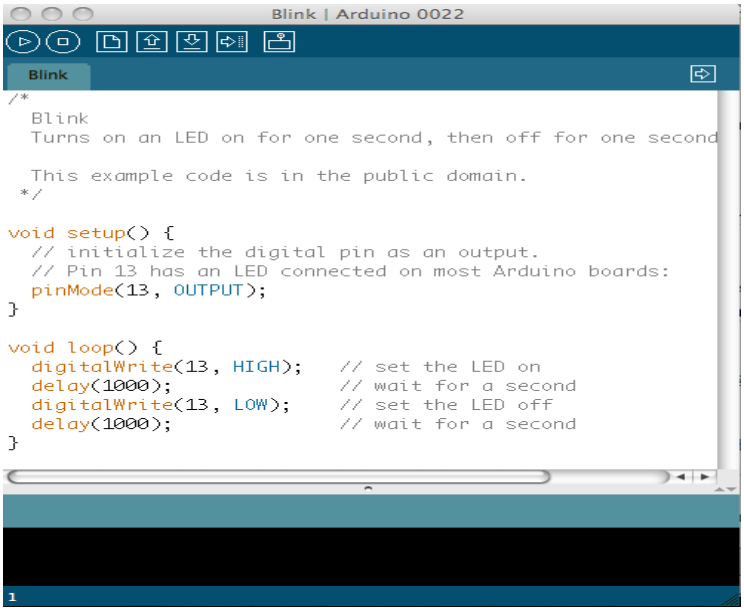
Ilustración 13: Arduino versión serie

El IDE Arduino.

Cuando trabajamos con Arduino, lo que hacemos realmente es mezclar un circuito con una idea que plasmamos en un programa. Este programa lo grabamos en un chip que es el microcontrolador que hay sobre la placa de Arduino.



Las siglas IDE vienen de Integrated Development Environment, lo que traducido a nuestro idioma significa Entorno de Desarrollo Integrado. En el caso de Arduino se trata de una ventana en la que podremos editar los programas que vamos a cargar en la placa y una serie de botones que nos permitirán llevar a cabo operaciones como la verificación de que nuestro programa es correcto o programar el microcontrolador.

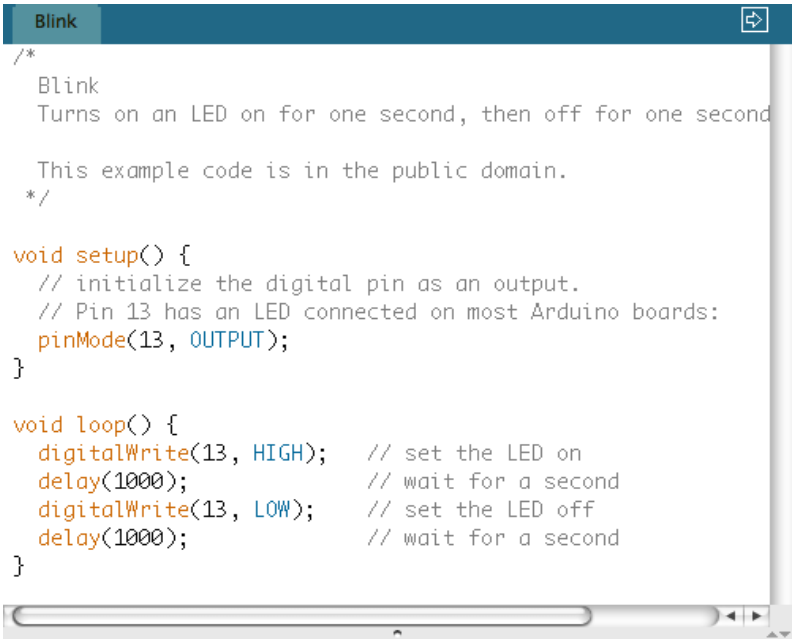


La ventana se divide en cuatro grandes áreas:

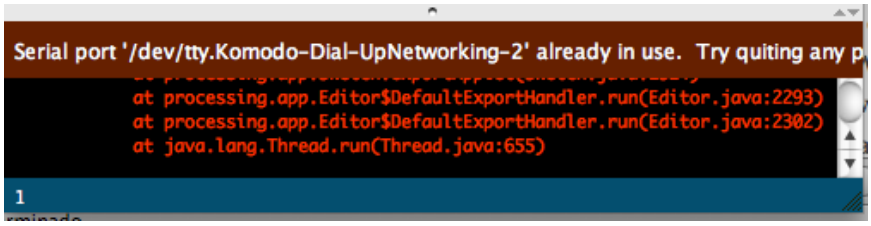
- La barra de menús.

- La barra de botones.

- El editor.



- Y la barra de mensajes.



Primera práctica: Configurar la placa y el puerto serie.

Lo primero que tenemos que hacer a la hora de empezar a trabajar con Arduino es “explicarle” al IDE dónde puede encontrar la placa y de qué variante de Arduino se trata.

Una vez hemos conectado la placa al puerto USB de nuestro ordenador, haremos click sobre la opción “Tools” y seleccionaremos la placa que hemos conectado.

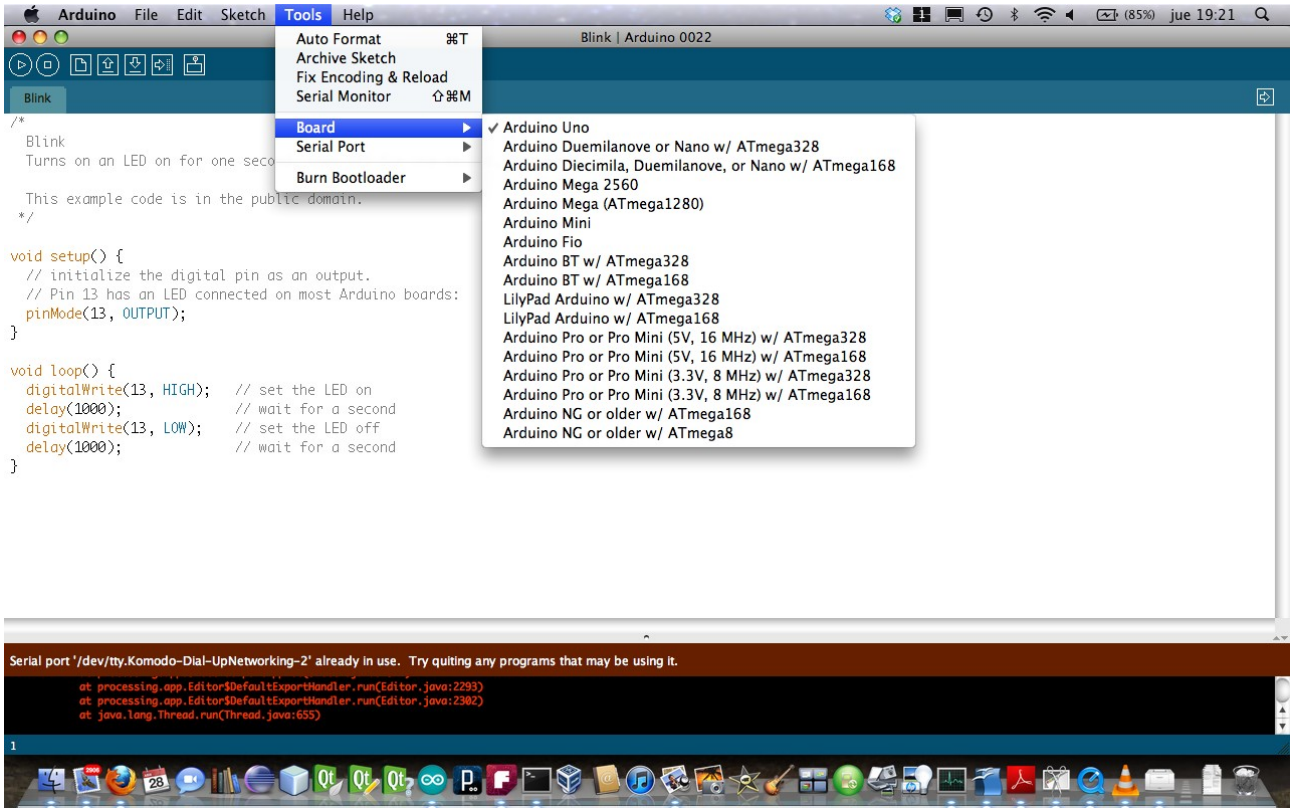


Ilustración 14: Configuración del tipo de placa

A continuación, tendremos que indicar en qué puerto está conectado. Para ello iremos al siguiente menú.

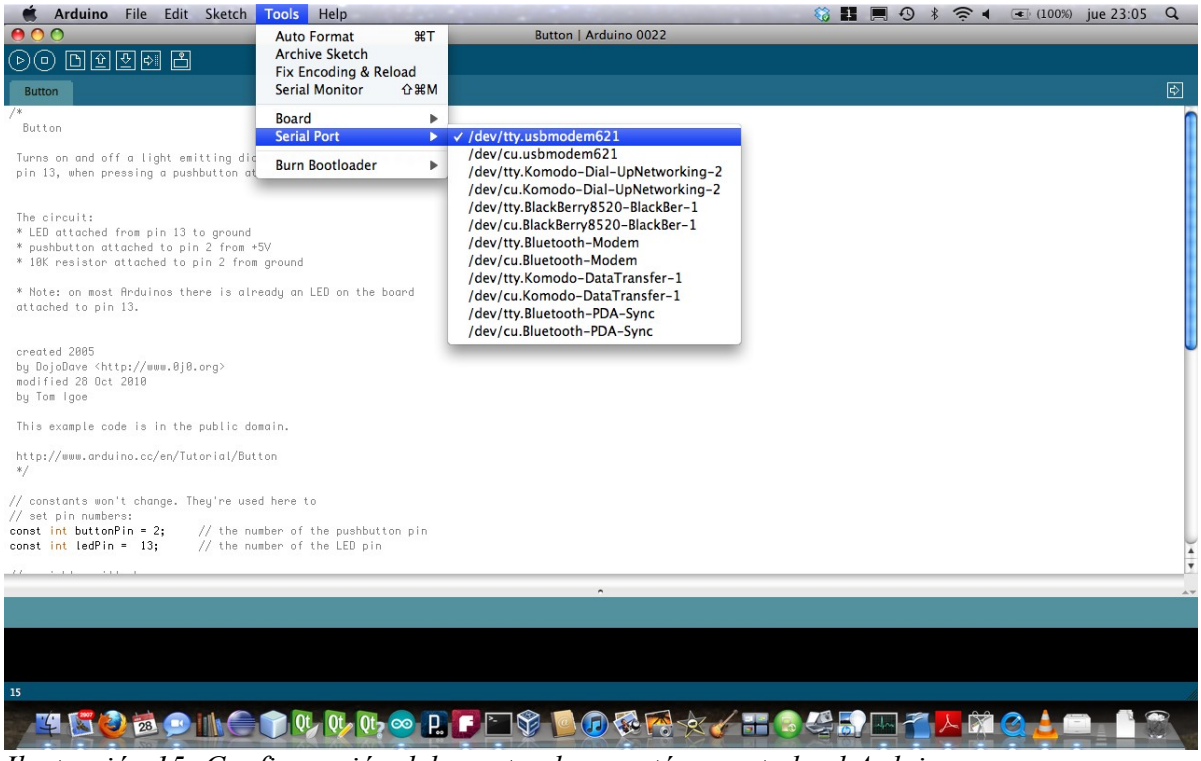


Ilustración 15: Configuración del puerto al que está conectado el Arduino

Aquí la cuestión comienza a complicarse un poco. Dependiendo del sistema operativo que utilices, en este menú encontrarás unas opciones u otras. Te reseño las opciones más comunes:

- En las máquinas con Linux y con Mac OS X, encontrarás una lista con nombres con esta pinta: /dev/tty.usbXXXX
- En las máquinas con Windows lo que verás será una lista de puertos denominados COMX, donde X es un número.

En ambos casos, el puerto suele aparecer cuando conectas el Arduino mediante el cable USB y desaparecer cuando éste ya no está conectado. Simplemente tendrás que probar.

Y recuerda: Prueba, prueba y vuelve a probar, no se va a romper la placa, **¡no tengas miedo!**

Al conectar la placa por primera vez el pequeño diodo que se encuentra cerca del PIN 13 debería comenzar a parpadear periódicamente, siempre y cuando no se haya cargado ningún otro programa en la placa. En cualquier caso, el diodo verde marcado como ON debe mantenerse encendido de manera contínua.

¡Felicidades, has terminado la práctica 1!

Programación de un Arduino.

Vamos a comenzar a programar Arduinos ¡Por fin!

Arduino utiliza una mezcla curiosa de lenguajes de programación. Está implementado en Java, pero para programar los chips utiliza C++.

Nosotros no pretendemos conseguir que en un taller de un par de horas la gente se convierta por arte de magia en grandes programadores, pero sí que sean capaces de hacer sus primeros pinitos mediante proyectos sencillos y que esto pueda favorecer que les pique el gusanillo.

Un programa diseñado para ejecutarse sobre un Arduino se conoce como sketch, que podríamos traducir como “boceto” o “borrador”. Un sketch siempre tiene la misma estructura.

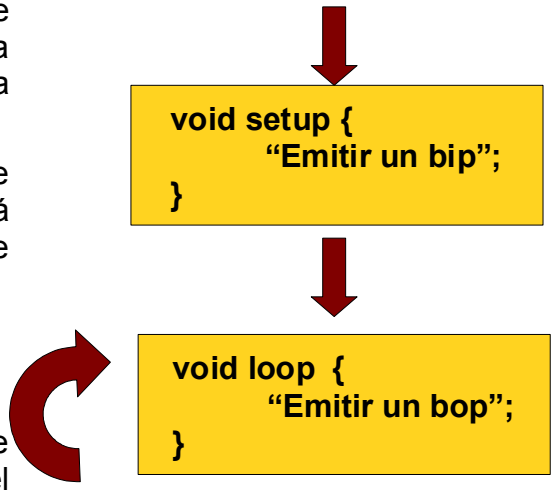
```
void setup() {
  // Lo que pongas entre estas llaves se ejecutara
  // una unica vez, al arrancar.

}

void loop() {
  // Lo que pongas entre estas llaves se ejecutara
  // en un bucle infinito despues de ejecutar el contenido de setup.


}
```

Ilustración 16: Sketch mínimo para Arduino

Lo que se escriba entre las llaves que acompañan al nombre setup, se ejecuta una única vez siempre que se encienda o resetée la placa.

Lo que se escriba dentro de las llaves que acompañan al nombre loop se ejecutará constantemente hasta que se apague o resetée la máquina.

Para entendernos, un pequeño ejemplo gráfico.



Si tuviéramos un Arduino capaz de entender este programa, al encenderlo, es decir, al conectarle el cable USB o una pila de 9V. Primero se ejecutaría el contenido de la función setup y a continuación comenzaría a repetirse indefinidamente el contenido de la función loop. Por lo tanto, lo que vería el dueño del Arduino sería:

Bip

Bop

Bop

Bop

Bop

.

.

.

Hasta que a alguien se le ocurriera desconectar el cable o la pila del Arduino.

Control de entradas y salidas digitales.

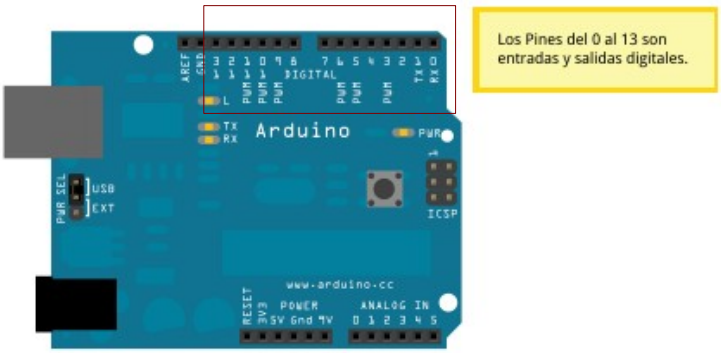


Ilustración 17: Pines digitales de Arduino

Arduino dispone de 14 pines de entradas y salidas digitales. Pueden configurarse como pines de entrada o de salida. Veamos un ejemplo.

Práctica 2: Encendido de un LED.

Un diodo LED es un dispositivo electrónico que es capaz de producir luz, requiere que se preste atención a la polaridad, es decir, debe tenerse en cuenta que una pata debe estar conectada a un punto del circuito con una tensión más positiva que la otra.

El truco es recordar:

- Un LED dispone de dos patillas, una más larga y otra más corta.
- La pata más larga debe estar conectada a la parte más positiva del circuito.
- El voltaje entre las patas del LED debe estar dentro de los límites que indica el fabricante.

Los LEDS suelen encenderse en torno a los 3V. Para limitar la tensión en los extremos de un LED debemos colocar una resistencia. Si no lo hacemos podríamos llegar a quemarlos.

La idea:

Quiero que un led se encienda y parpadée.

El circuito.

Vamos a utilizar una salida cualquiera del Arduino, en principio, la patilla 12. A esta patilla vamos a conectar un LED. Para que encienda con normalidad y no dañarlo, vamos a colocarle, en serie, una resistencia, yo he representado una de 2K2Ω.

Fíjate en la imagen.

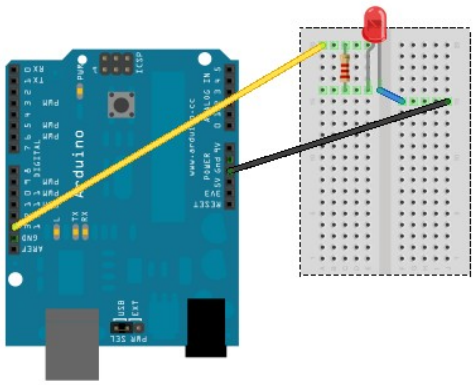


Ilustración 18: Circuito de la práctica 1

Hay que recordar que la pata más larga del LED debe conectarse a una parte “más positiva” del circuito, mientras que la pata más corta la conectaremos a tierra utilizando uno de los pines etiquetados como GND (Ground) del Arduino

El programa:

```
void setup() {
  // Ponemos el pin 12 en modo salida.
  pinMode(12, OUTPUT);

}

void loop() {
  // Lanzamos 5 Voltios por el pin 12.
  digitalWrite(12, HIGH);
  // Esperamos 1 segundo.
  delay(1000);
  // Dejamos el puerto 12 a 0 V.
  digitalWrite(12, LOW);
  // Esperamos 1 segundo.
  delay(1000);
}
```

Recuerda, lo que se escribe entre las llaves de la función setup se ejecuta una única vez en el arranque de la placa. Lo que coloquemos entre las llaves de la función loop se ejecutará una vez tras otra hasta que apaguemos el Arduino.

Te estarás preguntando ¿qué porras pone ahí? Vamos a traducir un poco.

pinMode(numero de pin, entrada o salida)

pinMode es una función, es un trozo de código que alguien programó para que no tuvieras que hacerlo tú, así que después de mostrarle el respeto y agradecimiento adecuado vamos a ver para qué sirve.

Esta función configura uno de los pines digitales como entrada o salida. Si recuerdas tenemos catorce pines digitales, el primero es el 0 y el último es el 13. Además existen dos posibles configuraciones para cada pin, puede estar configurado como entrada INPUT o como salida OUTPUT.

Así que para configurar el puerto 12 como salida tendremos que escribir:

- pinMode(12, OUTPUT);

Si lo quisiéramos configurar como entrada el pin 11 tendríamos que escribir:

- pinMode(11, INPUT);

Cosas importantes:

¡Ojo!, el pin que quiero configurar y la configuración que quiero que se le aplique están separados por una coma "," ¡No te olvides de ella!

El punto y coma del final ";" también es importante.

Más importante aún, cuando programes un Arduino recuerda que debe estar desconectado del resto del circuito.

digitalWrite(número de pin, estado alto o estado bajo)

A estas alturas, ya tienes que haberte percatado de que esto tiene que ver con señales digitales. Como sabes, las señales digitales binarias, representan dos estados un estado bajo, también conocido como 0, apagado u OFF y un estado alto también conocido como 1, encendido u ON. También sabrás que el estado alto o HIGH se representa con 5V (Aunque las placas que se alimentan a 3.3V devolverán esto como valor alto) y que el estado bajo o LOW se representa con 0V.

DigitalWrite necesita dos parámetros, el primero, una vez más es el número de pin digital y el siguiente es el estado que queremos mantener en ese pin, por lo tanto.

Si quiero enviar un valor alto en el pin 12 tendré que escribir:

- digitalWrite(12, HIGH);

Si quiero tener 0V en el pin 10 escribiré:

- digitalWrite(10, LOW);

delay(milisegundos)

Delay es una función más sencillita que el resto, hace que toda la ejecución de Arduino pare durante los milisegundos que le indiquemos como argumento.

Por lo tanto, si quiero esperar medio segundo escribiré:

- delay(500);

Si quiero esperar diez segundos escribiré:

- delay(10000);

Bueno, vale, el programa ya está listo ¿Ahora qué hacemos? Es bien sencillo. Por un lado pulsaremos sobre el botón que tiene el símbolo de play de tu DVD, minicadena o reproductor multimedia.

Este botón hará que tu sketch se compile (Se transforme a un lenguaje que "entiende" el microcontrolador) Si hubiera cualquier error, nos aparecerá un texto en rojo en la parte baja de la ventana.

Si todo ha ido bien, podemos “subir” el programa a la placa. Lo que haremos es enviar a través del cable USB el programa traducido y lo grabará en el chip del microcontrolador.

Este proceso es realmente complicado, sólo hay que hacer click en otro botón. Concretamente en el que se muestra en el gráfico a continuación.

Sobre la placa hay dos pequeños LEDS, etiquetados con los nombres, TX y RX que deberían comenzar a parpadear

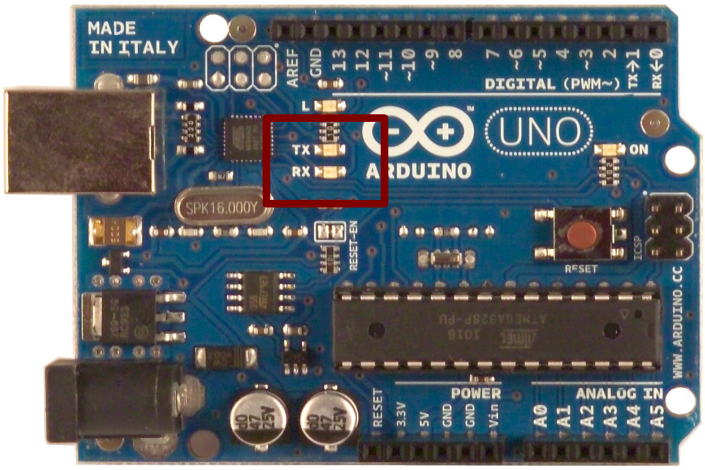


Ilustración 19: En el recuadro, los LEDS RX y TX

Actividades:

1. Haz el mismo montaje anterior pero utilizando el pin 7 para conectar el diodo LED.

2. Con el montaje anterior haz que el diodo esté encendido 1 segundo y apagado 2.

3. Intenta ahora hacer que tres leds se apaguen y parpadéen de manera secuencial, ¿a quién no le gustaba el efecto del coche fantástico?

Práctica 3: El LED L integrado en la placa.

Si te fijas, en la placa, justo enfrente del pin digital número 13 tienes un pequeño LED que tiene serigrafiada una letra “L” justo al lado. Es un diodo que está conectado directamente al pin 13 y que no necesita de ningún componente adicional. Podemos utilizarlo en nuestros montajes para mostrarnos si la placa está teniendo actividad o no. La manera de programarlo es exactamente la misma que en los casos anteriores.

La idea.

Quiero lograr que parpadée el LED L integrado en la placa.

El circuito.

Para desarrollar esta idea sólo hace falta la placa de arduino y un cable USB

El programa.

```
void setup() {
  // Ponemos el pin 13 en modo salida.
  pinMode(13, OUTPUT);

}

void loop() {
  // Lanzamos 5 Voltios por el pin 13.
  digitalWrite(13, HIGH);
  // Esperamos 1 segundo.
  delay(1000);
  // Dejamos el puerto 13 a 0 V.
  digitalWrite(13, LOW);
  // Esperamos 1 segundo.
  delay(1000);
}
```

Práctica 4: Lectura de entradas digitales.

Vale, ya encendemos algunos cachivaches, pero ...¿cómo nos enteramos de lo que pasa fuera de la placa?

Del mismo modo que podemos configurar un pin como salida, también podemos configurarlo como entrada. Y, por supuesto, también podemos leer lo que pasa en esa entrada.

La manera más sencilla de interactuar con el mundo exterior es mediante pulsadores. Un pulsador tiene la siguiente pinta:

Las patillas se conectan dos a dos

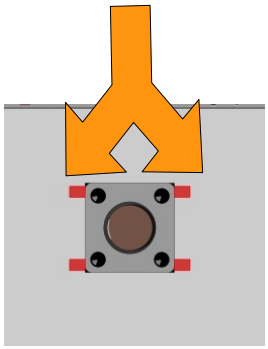


Ilustración 20: Un mini pulsador

La idea.

Quiero que un LED se encienda siempre que pulse un botón.

El circuito.

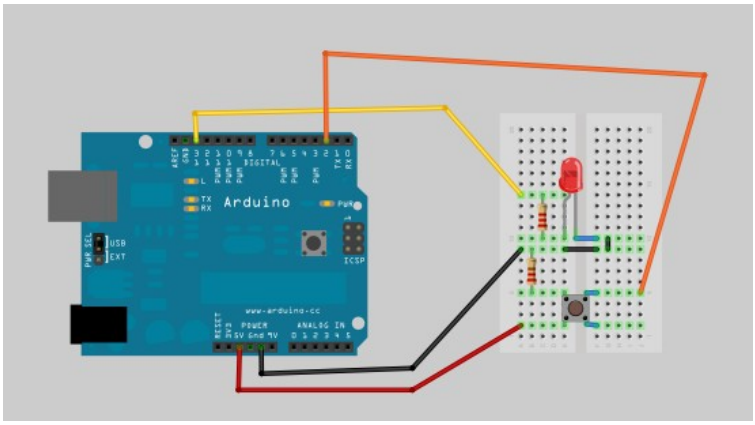


Ilustración 21: Micro pulsador con resistencia de pull-down.

El circuito es un poquito más complejo que el anterior. Hemos añadido un par de cables más, una resistencia de 220Ω y un pulsador. La idea es que cuando el pulsador se cierra obtenemos 5V en el pin 2, que tendremos que configurar como entrada. Mientras el pulsador permanezca inalterado obtendremos 0V a la entrada.

Hasta ahí todo correcto, pero … ¿Esa resistencia nueva para qué es?

Vamos primero con el problema que resuelve y después nos centramos en cómo resolverlo.

Si nosotros conectásemos un Arduino y leyésemos una entrada digital que no tiene nada conectado podríamos ver que, en ocasiones, los valores a la entrada fluctúan.

La manera de evitar esto es con una resistencia de pull-down o una resistencia de pull-up. Una **resistencia de pull-down** lo que hace es garantizar que cuando el botón no está pulsado, se lea siempre un 0 lógico, es decir, fija la tensión a 0 V. Por el contrario, una **resistencia de pull-up** garantiza que el valor leído es un 1 lógico, por tanto 5V.

En nuestro caso, hemos decidido utilizar la resistencia de 220Ω como resistencia de pull-down.

El programa.

```
int boton = 0;

void setup() {
  // Ponemos el pin 13 en modo salida.
  pinMode(13, OUTPUT);
  // Ponemos el pin 2 en modo entrada.
  pinMode(2, INPUT);
}

void loop() {
  // Leemos el estado del boton conectado al pin 2.
  boton = digitalRead(2);
  // Si el valor es "Alto", es decir 5V. Encendemos el led.
  if (boton == HIGH) {
    digitalWrite(13, HIGH);
  }
  // Si el valor es "Bajo", es decir 0V. Apagamos el led.
  else {
    digitalWrite(13, LOW);
  }
}
```

Algunas cosas nuevas: ¿qué es eso que aparece en el comienzo del archivo?

La primera linea es una declaración de variables, vamos a verla más de cerca:

```
int boton = 0;
```

Una variable es un pequeño (o no tan pequeño) espacio en memoria en el que podemos guardar algo, ese algo puede ser un número, una letra, un montón de números, un montón de letras o un estado, por ejemplo: Verdadero o Falso.

En color naranja aparece el **tipo de datos** de las variables que estamos definiendo. El tipo de datos de una variable indica qué tipo de información podemos guardar dentro de la misma. De momento nos vamos a centrar sólo en este, ya iremos tirando de la manta y desvelando el resto.

```
int boton = 0;
```

El tipo de datos int representa un número entero, es decir un número sin coma. En nuestro pequeño microcontrolador podemos representar números enteros, concretamente todos los que se puedan representar con dos bytes, es decir: desde -32768 hasta 32767.

Te estarás preguntando ¿y si quiero utilizar números mayores …? ¡Vaya castaña! Evidentemente se puede, y números con parte decimal con bastante precisión, ¡tranquilo pequeño padawan!. El espectáculo no ha hecho más que comenzar.

La definición de variables siempre incluye un tipo de datos, el nombre que le queremos

dar a la variable y, opcionalmente, un valor con el que queremos que se inicialice. En nuestro caso, hemos decidido inicializar la variable con un valor de 0.

Si no inicializas la variable no puedes estar seguro del valor que tendrá en el primer momento, antes de que le asignes tú uno y eso puede ser problemático.

digitalRead(número de pin)

Esta es la primera de las funciones que tratamos que devuelve un valor, a estas alturas ya tendrás más que claro qué hace. Lee el estado del pin que le indicamos como argumento y lo guarda en la variable que le pongamos delante. Por ejemplo:

- boton = digitalRead(2);

Leerá el estado del pin 2 y guardará el valor leído en la variable boton que hemos definido como int en los pasos anteriores.

¿Pero qué se va a guardar en la variable? Pues sencillo, se pueden guardar sólo dos valores, que además ya conocemos de algún paso anterior:

- Devolverá HIGH si en la entrada hay tres voltios o más.
- Devolverá LOW si en la entrada hay menos de tres voltios.

Cosas importantes:

Las entradas analógicas también pueden utilizarse como pines digitales de entrada, serían los pines 14 al 19.

if – else

La otra cosa rara que aparece en el código es eso que pone **if** y un poco más abajo **else** si andas un poco bien de inglés sabrás lo que significan estas dos palabras, significan “si” y “sino”.

No hay que ser una lumbrera para darse cuenta de para qué pueden usarse. En programación se llama a estas cosas sentencias de control, sirven para que el programa lleve a cabo unas tareas si se da una condición y otras si no se da.

Si traducimos a lenguaje humano lo que hemos escrito en el programa podríamos decir:

```
Si (el valor de la variable boton es HIGH) {
        Pon 5 voltios en el pin 13.
}
Si no {
        Pon 0 voltios en el pin 13.
}
```

Tenemos que saber algunas cosas antes de continuar:

Las comprobaciones que queremos hacer para decidir si hacemos unas tareas u otras deben estar entre paréntesis detrás de la palabra if.

Podemos poner tantos bloques if como queramos, pero el último de todos debe ser el bloque else.

Puede haber programas en los que aparezca un bloque if pero que no termine con un bloque else.

```
if (condicion) {
        .
        Lo que queremos que ocurra si se cumple la condición.
        .
}
else {
        .
        Lo que queremos que ocurra si no se cumple la condición.
        .
}
```

Para hacer comprobaciones sobre variables disponemos de algunos operadores:

- a == b: Combrueba si el contenido almacenado dentro de la variable a es igual que el de la variable b.
- a != b: Comprueba si el contenido almacenado dentro de la variable a es distinto que el de la variable b.
- a < b: Comprueba si el contenido almacenado dentro de la variable a es menor que el de la variable b.
- a > b: Comprueba si el contenido de la variable a es mayor que el de la variable b.
- a <= b: Comprueba si el contenido de la variable a es menor o igual que el de la variable b.
- a >= b: Comprueba si el contenido de la variable a es mayor o igual que el de la variable b.

Cosas importantes:

¡Ni se te ocurra desmontar el circuito, vamos a usarlo dentro de un momento.!

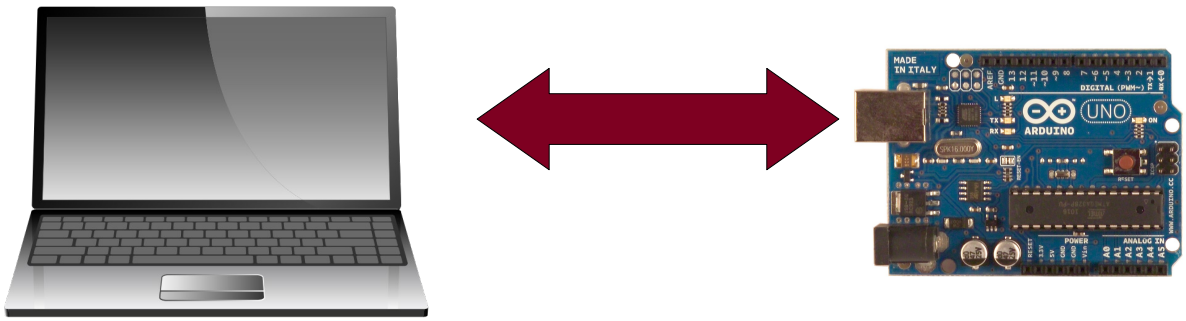
Actividades:

1. Haz el mismo montaje anterior, pero conecta dos LEDS, uno en el pin 12 y otro en el pin 13 y haz que se enciendan de manera alternativa cuando presiones el pulsador, esto es: Cuando el diodo del pin 12 está encendido, el diodo del pin 13 debe estar apagado y viceversa.

Comunicaciones serie.

¿No sería genial poder hablar con tu Arduino? No, no me estoy volviendo loco y tampoco me han abandonado mis amigos. ¡Puedes enviar información desde tu Arduino hacia tu ordenador o hacia otros dispositivos! ¡También puedes enviar datos a tu Arduino desde tu ordenador!

Vamos a ver cómo se hace.



Arduino no es capaz de “entender” una transmisión USB, el protocolo de comunicaciones USB es bastante complicado y, por lo general, la mayoría de microcontroladores no son capaces de trabajar con él. Sin embargo podemos comunicarnos con nuestra placa a través del cable USB porque uno de los chips que están soldados sobre ella es un conversor de protocolos, que transforma la información enviada a través del cable USB en información enviada mediante un puerto serie.

No dejes de leer todavía, te prometo que no te voy a soltar otro rollo informático, además el protocolo serie es sencillito.

Cuando dos dispositivos son capaces de comunicarse mediante un puerto serie, se envían la información en pequeños trozos unos detrás de otros. Por ejemplo, si quiséramos enviar la palabra “Hola” entre dos aparatos, comenzaría a enviarse H, O, L, A.

En una comunicación serie, uno de los factores más importantes es la velocidad de transmisión, que se mide en bps (bits por segundo) Una velocidad de 9600 bps significa que se estarán enviando 9600 bits por segundo.

Dos de los pines digitales de Arduino se utilizan para la transmisión y recepción serie, son los pines 0 y 1.

Hay que tener en cuenta que mientras usemos estos dos pines para comunicaciones serie no podremos utilizarlos como pines para entrada/salida digital.

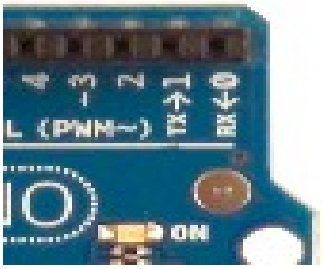


Ilustración 22: Pines Serie RX y TX

Si recuerdas hace poco, en la primera práctica, decíamos que cuando se grababa el programa en el chip mediante el botón upload parpadeaban dos pequeños LEDS que se llamaban RX y TX, evidentemente, esto no es casualidad.

El pin RX en una comunicación serie es por el que se recibe la información y el pin TX es por el que se transmite.

Cuando comencemos a transmitir y a "hablar" con otros dispositivos veremos que nuestros LEDS parpadearán indicándonos que está llevándose a cabo una transmisión.

Bueno, menos palabrería y más trabajar, que se nos va el tiempo y tenemos que cambiar el planeta a base de hardware libre.

Práctica 5: Transmisión serie.

La idea.

Quiero que mi arduino me mande un mensaje vía puerto serie cada vez que alguien pulse el botón.

El circuito.

Estarás diciendo … ¡Vaya timo! Si es el mismo circuito que antes.

Pues sí, te lo había avisado.

De momento vamos a "hablar" con nuestro Arduino a través del cable USB. Dentro de un ratito estaremos haciendo hablar a dos Arduinos entre si.

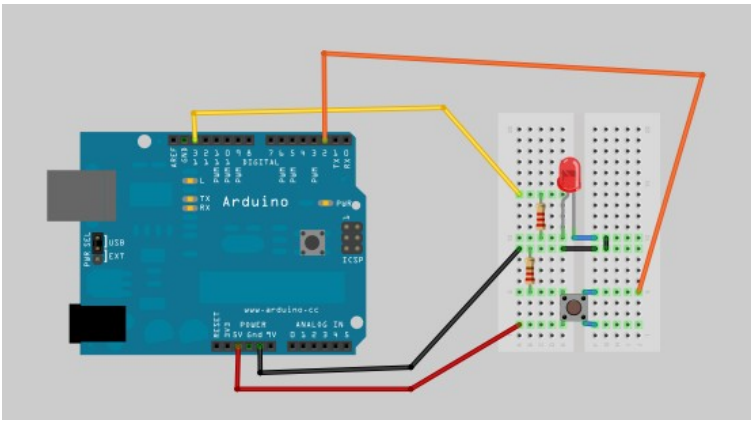


Ilustración 23: Micro pulsador con resistencia de pull-down.

El programa.

```
int boton = 0;

void setup() {
  // Ponemos el pin 13 en modo salida.
  pinMode(13, OUTPUT);
  // Ponemos el pin 2 en modo entrada.
  pinMode(2, INPUT);
  // Inicializamos el puerto serie para que transmita a 9600 bps.
  Serial.begin(9600);
}

void loop() {
  // Leemos el estado del boton conectado al pin 2.
  boton = digitalRead(2);
  // Mostramos por el puerto serie el valor que hemos leido.
  Serial.println(boton);
  // Si el valor es "Alto", es decir 5V. Encendemos el led.
  if (boton == HIGH) {
    digitalWrite(13, HIGH);
  }
  // Si el valor es "Bajo", es decir 0V. Apagamos el led.
  else {
    digitalWrite(13, LOW);
  }
}
```

Como no podía ser de otra manera, más cosas nuevas. Si te fijas sólo aparecen dos lineas distintas, vamos a destriparlas.

Serial.begin(velocidad en bps)

Esta función (Que, aunque no es una función, vamos a llamar así para no liarnos más) inicializa el puerto serie interno del microcontrolador. Y le indica a qué velocidad vamos a transmitir.

Entre los paréntesis podemos poner prácticamente cualquier valor, aunque para conmunicarnos con el ordenador con facilidad solemos utilizar alguno de los siguientes: 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, 115200.

No perdamos de vista lo que significan esos numeritos, cuanto más grandes, más bits por segundo pasarán por el cable.

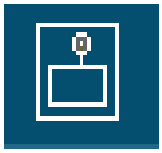
Serial.println(Dato que quiero enviar)

Esta es la primera función que veremos que nos permite enviar información, además tiene una peculiaridad, todos los datos que enviemos estarán seguidos por un salto de línea, es decir, el siguiente dato se escribirá en la línea de debajo.

El programa se compilará/verificará y se subirá a la placa de la misma manera que hasta ahora

Pero añadiremos un paso adicional para poder ver lo que nuestra placa de Arduino nos quiere decir a través del cable.

Hay otro botón que abre el monitor serie, que es una ventana en la que podremos leer lo que Arduino nos envía por el cable.



Al pulsar este botón se abrirá la siguiente ventana:

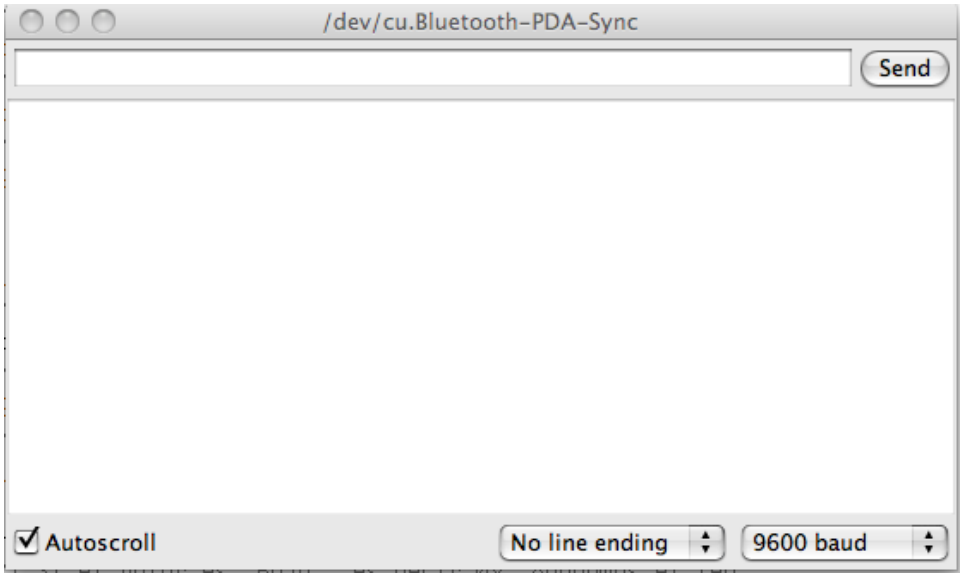


Ilustración 24: Ventana del monitor serie

Si te fijas, en la parte inferior derecha dispones de un menú desplegable en el que podrás seleccionar la velocidad de transmisión, en este caso es correcta y la dejamos así.

Bueno, la verdad es que estos mensajes tampoco son demasiado espectaculares ¿no?. Vamos a mejorar esto un poco.

Práctica 6: Transmisión serie mejorada.

La idea.

Vale, el mensajito de la práctica anterior está bien pero… Quiero que Arduino me mande un mensaje para humanos, no un mensaje para máquinas.

El circuito.

En serio, te prometo que me encanta currar, no es por falta de motivación, ni una excusa para no hacer otro esquema, simplemente es que con ese montaje nos vale.

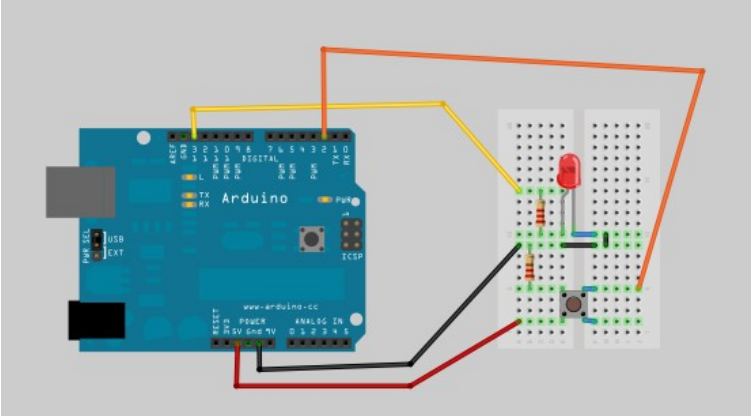


Ilustración 25: Micro pulsador con resistencia de pull-down.

El programa.

```
int boton = 0;
int contador = 0;

void setup() {
  // Ponemos el pin 13 en modo salida.
  pinMode(13, OUTPUT);
  // Ponemos el pin 2 en modo entrada.
  pinMode(2, INPUT);
  // Inicializamos el puerto serie para que transmita a 9600 bps.
  Serial.begin(9600);
}

void loop() {
  // Leemos el estado del boton conectado al pin 2.
  boton = digitalRead(2);
  // Si el valor es "Alto", es decir 5V. Encendemos el led.
  if (boton == HIGH) {
    digitalWrite(13, HIGH);
    contador = contador + 1;
    // Mostramos por el puerto serie el valor que hemos leido.
    Serial.print("El boton ha sido pulsado ");
    Serial.print(contador);
    Serial.println(" veces");
  }
  // Si el valor es "Bajo", es decir 0V. Apagamos el led.
  else {
    digitalWrite(13, LOW);
  }
}
```

Este programa tiene pocas cosas nuevas, si te fijas aparece una nueva declaración de variables, concretamente:

int contador = 0;

Como ya sabes, estamos definiendo una variable que almacenará un número entero, en este caso, la hemos llamado contador y la hemos inicializado con un valor 0.

Cosas importantes:

Es muy buena idea que los nombres de las variables sean descriptivos, esto es, que representen el valor que contienen. Pero no te pases, que no se te vaya a ocurrir definir la variable:

```
int contador_que_voy_a_usar_para_encender_un_LED = 0;
```

Un poco más abajo hacemos lo siguiente:

```
contador = contador + 1;
```

Lo que quiere decir esa linea es sencillo si lo piensas un poco. En la variable contador, guarda el valor que tiene ahora mismo más uno.

Si te fijas, esa linea está puesta justo detrás de la sentencia **if** que controlaba el valor leído en el pin al que está conectado el botón.

Si lo leemos todo en conjunto veremos que lo que pretende hacer el programa es:

Si el botón ha sido pulsado

Enciende el LED del pin 13

Incrementa el contador en una unidad

Manda por el puerto serie un mensaje.

Serial.print(Dato que quiero enviar)

Justo en este punto, tenemos una nueva instrucción que nos permite enviar información a través del puerto serie. La única diferencia entre esta instrucción y la anterior es que ésta no termina en “ln”. Lo que significa que cuando se envían los datos no se salta de linea al representarlos.

Si en un programa de Arduino aparecieran las siguientes sentencias:

```
Serial.print(“Uno”);
Serial.print(“Dos”);
```

Lo que recibiría por su puerto serie el dispositivo que tuviera conectado sería:

UnoDos.

Si lo que deseásemos fuera que aparecieran en la misma línea pero separados, tendríamos que incluir un espacio entre las comillas de alguno de los dos mensajes, por

ejemplo:

```
Serial.print(“Uno ”);
Serial.print(“Dos”);
```

Esto daría como resultado.

```
Uno Dos.
```

Por último, si quisiéramos que un mensaje apareciera en una línea y otro en la siguiente tendríamos que hacer:

```
Serial.println(“Uno”);
Serial.println(“Dos”);
```

Lo que terminaría siendo representado como:

```
Uno
Dos
```

Otra cosa a tener en cuenta, podemos enviar el valor que hay guardado en una variable o podemos enviar un texto. Debemos recordar que el texto que queramos enviar debe estar situado entre dobles comillas, es decir:

- **Correcto:** `Serial.print(“Este es el texto que quiero enviar”);`
- **Incorrecto:** `Serial.print(Este es el texto que quiero enviar);`

En el programa que tenemos entre manos somos más rebuscados, hacemos lo siguiente:

```
Serial.print(“El boton ha sido pulsado ”);
Serial.print(contador);
Serial.println(“ veces”);
```

Lo que hace esto es intercalar el valor de la variable contador entre dos mensajes compuestos por letras ¡Por eso están entre comillas!

Por ejemplo: Si el valor de contador fuera 12, en el dispositivo que recibe la transmisión veríamos:

```
El boton ha sido pulsado 12 veces
```

Cosas importantes:

Algunos estarán pensando … mira al tío este que está siempre con lo de las faltas de ortografía. En líneas generales es mala idea utilizar tildes dentro de tu código, es más, en general no se representarán en la ventana del editor.

¡Que conste que sé que botón lleva tilde!

Puede estar ocurriendo otra cosa, es posible que cada pulsación del botón genere más de un mensaje. Esto ocurre porque, aunque puedas percibir una única pulsación, a nivel electrónico se producen pequeñas variaciones, que duran microsegundos, y que hacen que al no estar estabilizada la entrada, Arduino pueda confundirse.

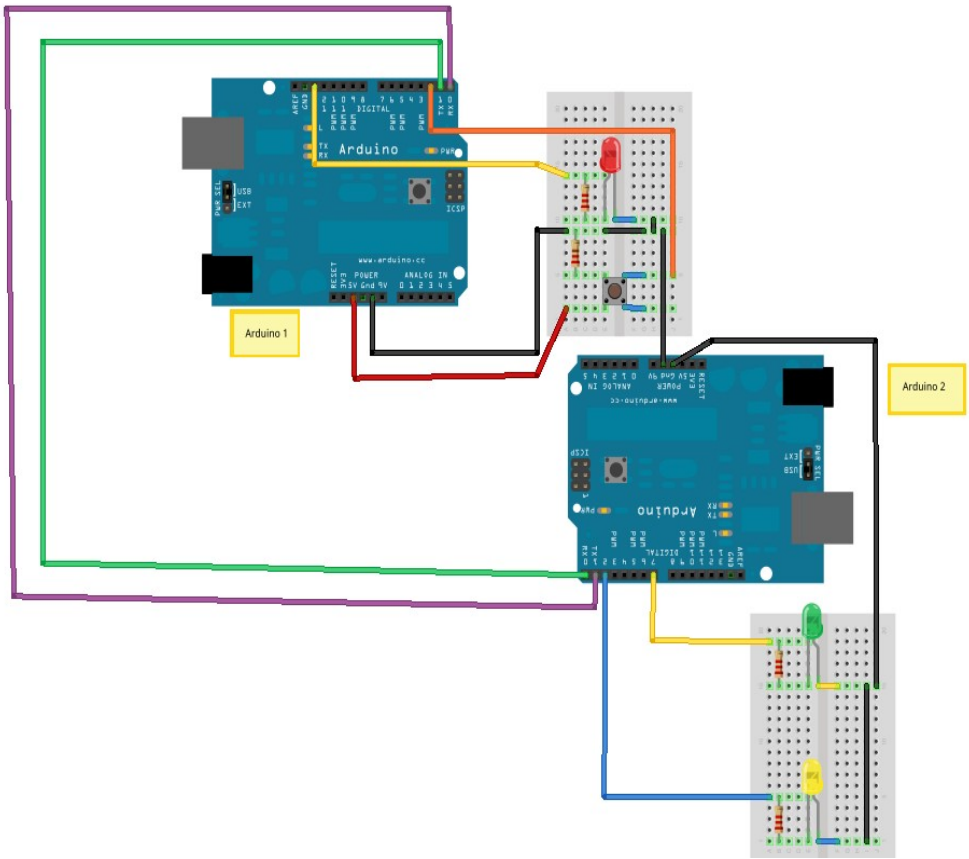
La solución más sencilla es añadir un pequeño retraso de un par de milisegundos, de manera que, cuando vuelva a leerse el estado de la entrada ya se haya estabilizado.

Práctica 7: Hablando con otros Arduinos.

La idea.

Quiero que mi Arduino le cuente a otro lo que pasa con una de sus entradas y que esto provoque que el segundo Arduino reaccione encendiendo varios LEDS.

El circuito.



Pues sí, la cosa se ha complicado un poquito más, además vas a tener que hacer esta práctica en equipo.

La vida es dura y hay todo un mundo más allá del teclado, así que si no conoces a la persona que hay sentada al lado tuyo ya sabes, preséntate, que la gente que usa hardware libre, aparte de freaky es muy educada.

Les recomiendo que monten un circuito cada uno y después unan las líneas comunes, son sólo tres, una línea que va desde el pin TX del Arduino 1 al pin RX del Arduino 2, otra linea que va desde el pin TX del Arduino 2 al pin RX del Arduino 1. Por último conectaremos las tierras de los dos para garantizar que tienen una referencia común.

Los programas.

En esta práctica tendremos que desarrollar dos programas, uno para el Arduino 1 y otro para el Arduino 2.

Vamos con el primero, que será una pequeña modificación del que hemos estado utilizando en la práctica anterior.

```
int boton = 0;
int contador = 0;

void setup() {
  // Ponemos el pin 13 en modo salida.
  pinMode(13, OUTPUT);
  // Ponemos el pin 2 en modo entrada.
  pinMode(2, INPUT);
  // Inicializamos el puerto serie para que transmita a 9600 bps.
  Serial.begin(9600);
}

void loop() {
  // Leemos el estado del boton conectado al pin 2.
  boton = digitalRead(2);
  // Si el valor es "Alto", es decir 5V. Encendemos el led.
  if (boton == HIGH) {
    digitalWrite(13, HIGH);
    contador = contador + 1;
    // Mostramos por el puerto serie el valor que hemos leido.
    Serial.println(contador);
  }
  // Si el valor es "Bajo", es decir 0V. Apagamos el led.
  else {
    digitalWrite(13, LOW);
  }
}
```

Lo único que enviaremos por el puerto serie, en este caso, será el número de pulsaciones que hemos recibido hasta el momento.

Para hacernos un esquema mental: Cada vez que alguien presione el pulsador, se volverá a encender el LED y enviaremos, por nuestro puerto serie un mensaje que contendrá, exclusivamente el número de pulsaciones que hemos recibido.

Ahora el programa para el Arduino 2, éste sí que tiene cosas nuevas.

```
int anterior = 0;
int recibido = 0;

void setup() {
  // Ponemos los pines 2 y 7 en modo salida.
  pinMode(2, OUTPUT);
  pinMode(7, OUTPUT);
  // Inicializamos el puerto serie para que transmita a 9600 bps.
  Serial.begin(9600);
}

void loop() {
  // Compruebo si me han enviado algo.
  if (Serial.available() > 0) {
    recibido = Serial.read()
    // Si el numero que me han enviado es mayor que
    // lo recibido anteriormente hago parpadear los LEDS.
    if (recibido > anterior) {
      anterior = recibido;
      digitalWrite(2, HIGH);
      delay(500);
      digitalWrite(2, LOW);
      digitalWrite(7, HIGH);
      delay(500);
      digitalWrite(7, LOW);
    }
  }
}
```

Serial.available()

De nuevo vamos a llamar a esto función aunque no lo sea, esta “función” nos indica cuántos bytes se han recibido en el puerto serie.

Si obtenemos un 0 cuando consultamos, significa que nadie nos ha mandado ni un triste mensajito. Si obtenemos un número mayor que cero significa que nos han enviado algo y podemos leerlo.

Serial.read()

Esta “función”, que tampoco es una función, nos devuelve el primer byte de los que se hayan recibido. Al leerlo lo elimina, por tanto, si volviéramos a leer recibiríamos el siguiente valor recibido.

Cosas importantes:

Arduino tiene muy poco espacio para mensajes recibidos, puede almacenar 128 bytes, en los que caben 128 números enteros o 128 letras.

Tenemos que tener cuidado de leer constantemente los datos que se nos envían e intentar no pasarnos enviando datos a otros Arduinos.

Si en algún momento necesitamos vaciar esos 128 bytes, existe otra función que podremos utilizar, se trata de **Serial.flush()**

Hay otra función, que nunca he utilizado, relacionada con el puerto serie, se llama **Serial.peek()**. Lo que hace es devolvernos el primer valor recibido pendiente de leer, pero no lo elimina. Sucesivas ejecuciones de esta función, nos devolverán el mismo valor.

E/S analógica.

Un Arduino UNO tiene seis entradas analógicas que están conectadas a sendos conversores analógico/digitales de 10 bits. Realmente, si le hablas a tus amigos con palabras de estas los vas a dejar alucinados, lo que significan tampoco es tan complejo, vamos a verlo.

Las señales digitales, que son con las que llevamos jugando desde hace un rato, sólo pueden tomar un conjunto limitado de valores, en el caso de las señales digitales binarias, 0 y 1.

Una señal analógica es una señal continua, es decir, puede tomar infinitos valores entre dos puntos. Por lo tanto si mido en un instante, puede estar tomando un valor 0 y en un par de milisegundos haber cambiado y tomar un valor 0,02.

El cerebro de Arduino, nuestro microcontrolador Atmega, es un pequeño ordenador, por lo que está habituado a trabajar sólo con ceros y unos. Los conversores analógico/digitales, lo que hacen es traducir esas señales que pueden tomar prácticamente cualquier valor, en otras que puede entender el microcontrolador.

El hecho de que nuestros conversores analógico-digitales sean de 10 bits, lo único que

significa, es que van a poder distinguir entre 1024[1] valores distintos. Si no te aclaras con esto, no te preocupes y confía en mi.

Por lo tanto, si tenemos un dispositivo que nos envía información codificada como cambios de voltaje y la diferencia de potencial máxima que puede establecer entre una entrada analógica de Arduino y tierra es de 5V, nuestro Arduino podrá informarnos de cómo evoluciona esa señal en intervalos de:

$$\frac{5}{1024} = 0{,}0049\,V = 4{,}9\,mV$$

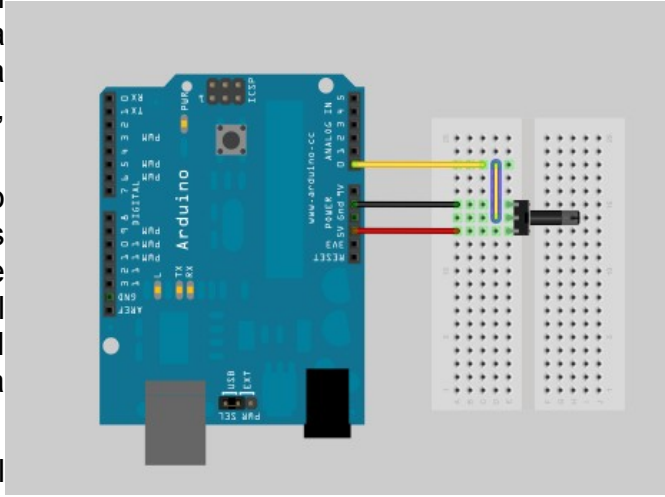
Menos rollo y vamos a ver cómo se hace.

Práctica 8: Monitorización de entradas analógicas con Arduino.

La idea.

Quiero que Arduino me informe de la diferencia de potencial que existe entre las patitas de un potenciómetro.

El circuito.



El circuito es bastante sencillo, un potenciómetro es una resistencia variable, a medida que giramos el mando, la resistencia va cambiando, y por tanto, también la tensión entre sus extremos.

La manera de conectar un potenciómetro es la siguiente, se pone uno de los extremos a tierra y otro a un voltaje conocido, nosotros aprovecharemos el hecho de que Arduino tiene un pin por el que podemos obtener 5V con respecto a tierra.

Finalmente, la pata central del potenciómetro se conecta directamente a una de las entradas analógicas, nosotros hemos decidido utilizar la entrada analógica 0.

El programa.

Seguro que continúas pensando... ¿Pero esto no iba a ser super complicado?

Pues no, de complicado nada de nada.

La verdad es que este ejemplo es bastante sencillito, pero nos puede servir para construir nuevos ejemplos más complejos.

```
int valor = 0;

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

void loop() {
  valor = analogRead(0);
  Serial.println(valor);
  delay(500);
}
```

1 Con 10 bits pueden representarse, como máximo 1024 valores diferente valores serán los comprendidos entre 0 y 1023, ambos incluídos.

Ahora mismo ya deberías ser capaz de imaginar qué es lo que debe ocurrir con este código.

Tenemos una nueva definición de variables, y ¡Caramba! De nuevo es un entero y lo inicializamos a cero.

Configuramos un puerto serie a una velocidad de 9600 bps, por lo tanto, seguro que estamos pensando en ver qué tiene que decirnos nuestro Arduino.

Después, dentro de loop, que si recuerdas era esa parte del código que se repetía una vez tras otra, tenemos una función que aún desconocemos. Con un poco de imaginación y otro poco de conocimiento de la lengua de Shakespeare podemos adivinar para qué sirve, lee una entrada analógica y nos devuelve el valor de la misma. Recuerda que estos valores siempre estarán comprendidos entre 0 y 1023.

Finalmente, sea cual sea ese valor leído lo enviamos por el puerto serie a quién lo quiera escuchar.

¡Ah, me olvidaba! Y esperamos 500 milisegundos.

Fácil ¿no?

Realmente, sólo hay una función nueva, destripémosla un poco.

analogRead(pinAnalógico)

Como ya hemos indicado, esta función devuelve el valor presente en una entrada analógica. Un arduino UNO tiene seis entradas analógicas, por lo tanto, si queremos leer el estado del pin analógico 3 escribiremos:

- analogRead(3)

Si quisiéramos leer el pin analógico 5, cambiaríamos la expresión anterior por:

- analogRead(5)

Práctica 9: Entradas y salidas analógicas.

La práctica anterior tiene que servir para algo más que para que Arduino nos muestre un número por el puerto serie.

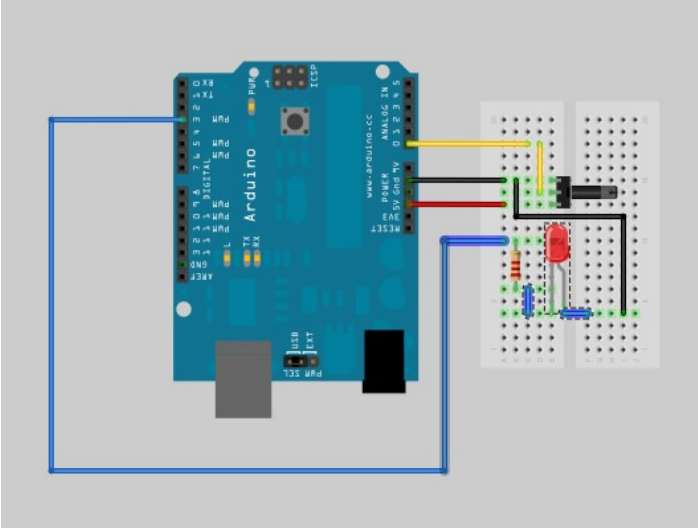
Como seguro que eres una persona avispada, habrás caído en la cuenta de que los potenciómetros se parecen sospechosamente a los mandos de control de volumen de los equipos de sonido ¿o no? Si en un equipo de sonido podemos utilizar un potenciómetro para cambiar el volumen de una salida, seguramente podremos inventar algo con esto …

La idea.

Quiero cambiar la intensidad lumínica de uno de mis leds con un potenciómetro.

El circuito.

Mantenemos el circuito de la práctica anterior y añadimos un led con su correspondiente resistencia.



El programa.

Tal y como hemos reutilizado el circuito anterior, también vamos a reutilizar el código, vamos a añadir un par de funciones nuevas y a eliminar el delay.

En este programa hemos introducido dos funciones muy importantes, map y analogWrite. Veamos cuál es su utilidad.

```
int valor = 0;

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

void loop() {
  valor = analogRead(0);
  Serial.println(valor);
  valor = map(valor, 0, 1023, 0, 255);
  analogWrite(3, valor);
}
```

analogWrite(pinPwm, valor)

Esta función es tremendamente interesante para todos aquellos que nos hemos peleado con la programación de microcontroladores en ensamblador[2], si no sabes lo que es, ¡Ni preguntes!

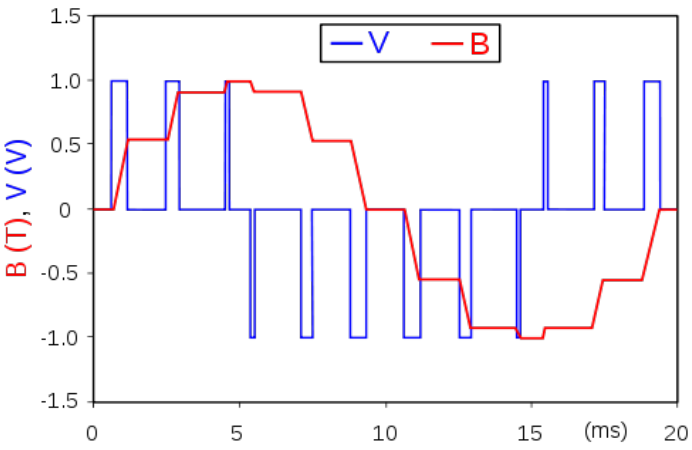


Ilustración 26: Modulación por amplitud de pulsos.

Las siglas PWM provienen de Pulse Wide Modulation, o lo que es lo mismo, modulación por amplitud de pulsos. Como esto no es un curso de electrónica sino un curso sobre Arduino, quédate sólo con la siguiente idea:

Mediante una señal PWM, que es una señal digital en la que se envían ceros o unos más o menos largos, podemos simular una salida analógica y hacer creer al cachivache que tengamos conectado a esa salida que lo que está recibiendo es una variación "suave" de voltaje.

La primera particularidad de esta función es que no puede utilizarse con cualquier pin, sólo con los que tienen dibujado a su lado este símbolo "~". Si echas un vistazo detectarás enseguida los pines que pueden generar señales PWM, son los: 3, 5, 6, 9, 10, 11.

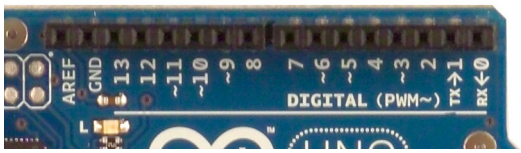


Ilustración 27: Detalle en el que se aprecian los pines con capacidad para generar señales PWM.

La segunda característica curiosa es que sólo podemos enviar valores entre 0 y 255.

map(valor, desdeBajo, desdeAlto, aBajo, aAlto)

Supongamos que tenemos una variable que puede tomar valores entre 0 y 1023 y que necesitamos adaptar esos valores a otro rango, por ejemplo, entre 0 y 255, de manera que si originalmente tenemos un 0 a la entrada, tendremos un 0 a la salida, pero si tenemos un 1023 a la entrada, generaremos un 255 a la salida.

A este proceso se le conoce como interpolación y la función map lo automatiza por nosotros.

Como habrás supuesto, lo que hace en nuestro programita es "adaptar" los valores que obtenemos al leer la entrada del osciloscopio, para que puedan utilizarse al generar la PWM.

2 El ensamblador o lenguaje ensamblador es la manera de comunicarnos y diseñar programas más cercana al lenguaje que entiende el procesador. Conozco a poca gente que le apasione, aunque alguno hay.

Servos y sensores de luminosidad (LDR)

Bueno, esto va tocando a su fin, vamos a intentar verle la utilidad a esto de las entradas analógicas.

A continuación vamos a utilizar un componente que se conoce como resistencia LDR, realmente es similar a un potenciómetro, pero que varía su resistencia en función de la luz que percibe.

Este es un ejemplo muy barato de sensor, la mayoría de sensores actúan como una LDR, varían un voltaje entre sus extremos que podemos “leer” en alguna de las entradas analógicas de nuestro Arduino.

Por otro lado tenemos los servomotores, también conocidos como servos.

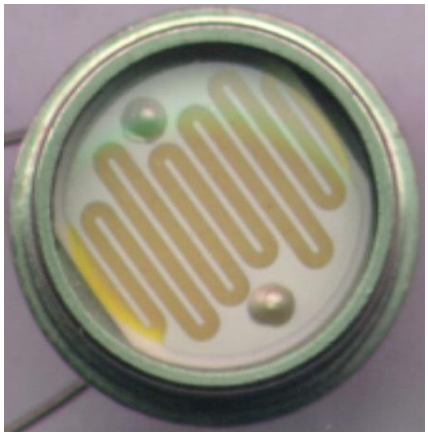


Ilustración 28: Resistencia LDR

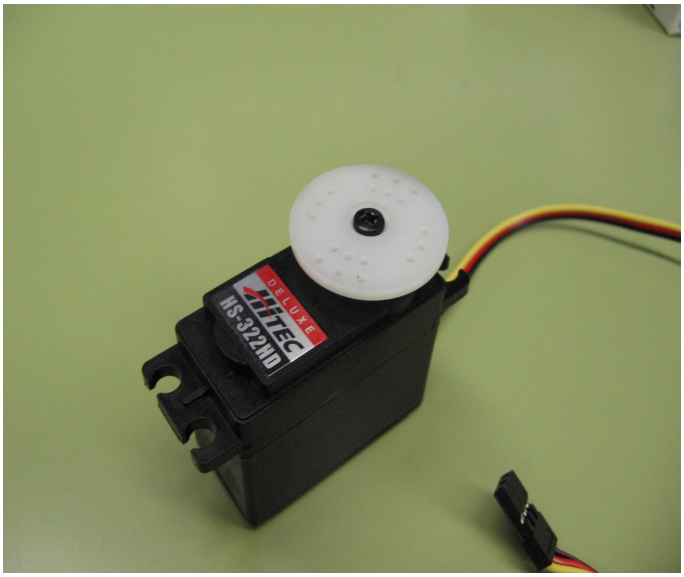


Ilustración 29: Fotografía de un servomotor

Un servo es un motor de corriente contínua metido dentro de una carcasa en la que también se encuentra una cierta lógica de control. Gracias a esa lógica de control, nosotros podemos indicarle qué ángulo queremos que gire y él, obedientemente, cumplirá con nuestros deseos.

Dentro de la carcasa de un servo también encontraremos unos engranajes que son los que garantizan una cierta fuerza a la hora de ejecutar el giro.

Los servos giran más lentamente que los motores de corriente continua normales, pero tienen mucha más fuerza de giro.

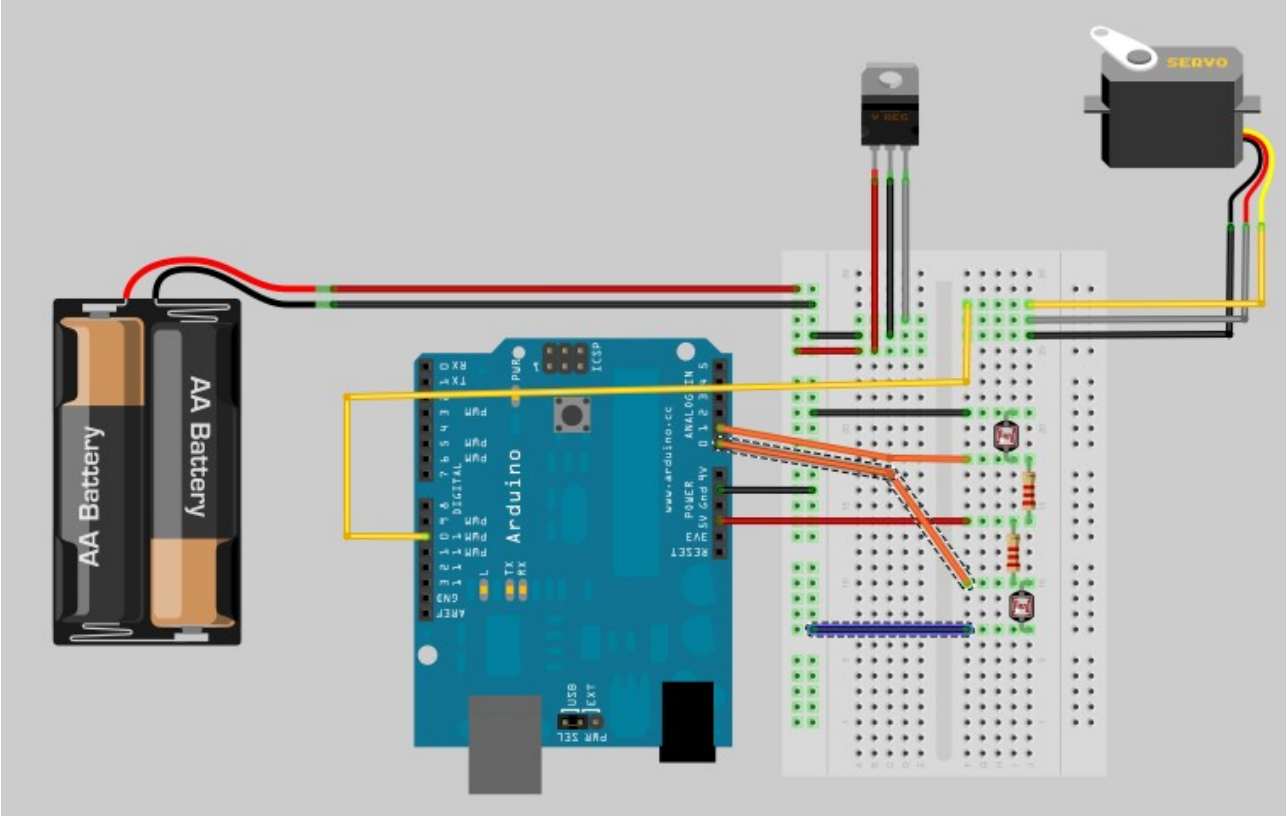
Los servos suelen disponer de tres cables, uno negro que debe conectarse a tierra, uno rojo que suele conectarse a 5V y un tercero que suele ser blanco o amarillo que es por el que se envía la señal de control.

Práctica 10: Control de un servo mediante entradas analógicas.

La idea.

Quiero diseñar un dispositivo que se oriente hacia el lugar más oscuro según los valores de dos sensores de luminosidad.

El circuito.



Sí, este circuito ya empieza a parecer un poco más complicado. Veamos de qué se trata.

En primer lugar llama la atención las pilas. Por lo general, la mayoría de dispositivos que solemos utilizar y conectar a Arduino pueden alimentarse a través de la propia placa y mediante el puerto USB del ordenador que utilizas para programar.

Sin embargo, cuando entran en juego motores cambia bastante el escenario. Los motores de corriente contínua y los propios servos suelen tener consumos elevados de corriente. Como ya hemos visto, nuestro Arduino les puede proveer, como máximo, con 40 mA. Por lo tanto, no nos queda otra que utilizar una fuente de alimentación alternativa.

Siempre que vayamos a utilizar varias fuentes de alimentación, es muy importante que nos acordemos de unir las tierras de las mismas, para garantizar que todas tienen un mismo nivel de referencia.

El otro problema que tenemos es que no suelen encontrarse con facilidad pilas de 5V, aunque es más fácil encontrarlas de 1'5V o 9V. Es por eso que recurrimos a un circuito regulador de voltajes. En nuestro caso, un 7805. Se parece bastante a un transistor, pero tiene una función muy distinta.

Si pones el integrado de frente a ti, de modo que puedas leer las letras que tiene sobreimpresas, las patas en sentido de lectura tienen las siguientes funciones:

- 1 Entrada de voltaje: Debemos conectarla al positivo de la fuente de alimentación.
- 2 Tierra: Debemos conectarla al negativo de la fuente de alimentación.
- 3 Salida de voltaje: Entre esta pata y tierra dispondremos de 5V.

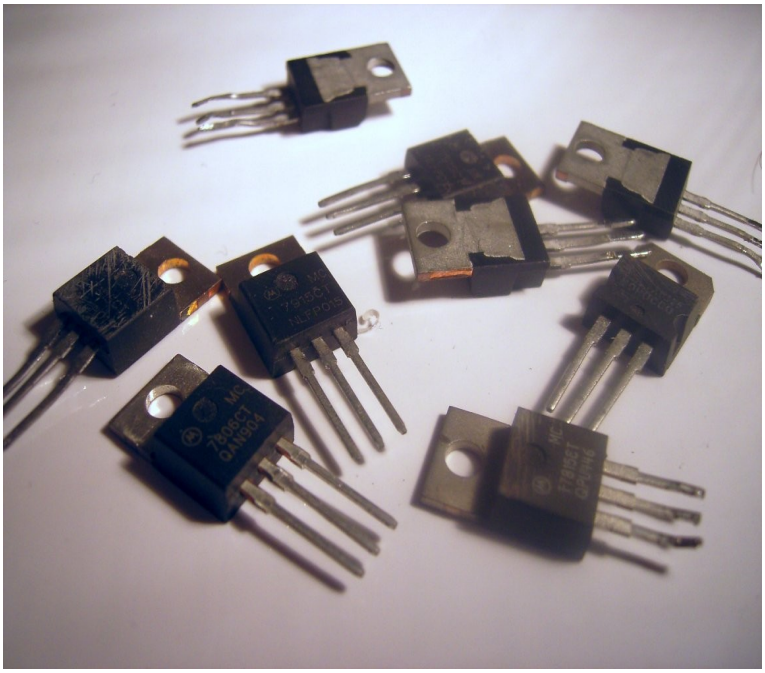


Ilustración 30: Reguladores de voltaje.

En el caso de las LDR, el circuito que hemos montado es todo un clásico en el mundo de la electrónica y se conoce como divisor de tensión. Como esto no es un curso de electrónica, no les voy a contar de qué va, pero saben que tienen toda la información a un click de distancia.

El programa.

Este programa sí que tiene sustancia, desde la primera línea comenzaremos a ver cosas desconocidas.

```
#include <Servo.h>

int valorLDR_Dcha = 0;
int valorLDR_Izda = 0;
int valorServo = 0;

Servo servo1;

void setup() {
  Serial.begin(9600);
  servo1.attach(10);
}

void loop() {
  valorLDR_Izda = analogRead(1);
  Serial.print("LDR Izquierda: ");
  Serial.print(valorLDR_Izda);
  Serial.print("          ");
  valorLDR_Dcha = analogRead(0);
  Serial.print("LDR Derecha: ");
  Serial.println(valorLDR_Dcha);
  if (valorLDR_Izda < valorLDR_Dcha) {
    valorServo = valorServo - 10;
    if (valorServo < 0) {
      valorServo = 0;
    }
  }
  else {
    valorServo = valorServo + 10;
    if (valorServo > 179) {
      valorServo = 179;
    }
  }
  servo1.write(valorServo);
}
```

Prestando un poco de atención no es difícil percatarse de esa línea extraña que pone algo así como esto:

- #include <Servo.h>

Esta línea incluye el contenido de una librería.

Además, no la he escrito yo, cuando queremos incluir el contenido de una librería, habitualmente pulsaremos en la barra de menús donde pone **Sketch**. En el menú desplegable seleccionaremos **Import Library** y de todas las librerías disponibles importaremos la que necesitemos.

Las librerías son conjuntos de funciones que alguien ha programado por tí para que no tengas que estresarte demasiado.

En nuestro caso, esta librería incluye la definición de una clase de objetos. Hasta ahora hemos estado llamando funciones a todo lo que escribíamos, si tenía paréntesis. Y hasta ahora yo he estado diciendo que lo estábamos nombrando mal.

Ha llegado el momento de contarles la verdad. Muchas de las cosas que hemos utilizado no eran funciones, sino métodos de un objeto.

Cierra los párpados que con esa cara de asombro la gente no va a querer hablar contigo. Sí, sí, esto tiene mucho que ver con eso que seguro que has oído alguna vez y que los freakys llaman: “Programación orientada a objetos”

Pero como esto no es un curso de programación lo vamos a dejar ahí. Lo único que necesito que sepas es que un objeto es un mecanismo para agrupar de manera lógica funciones.

Nosotros podemos crear variables que tengan como tipo un objeto, aunque en este caso, si somos muy puristas, tendremos que llamarlas instancias de un objeto.

Las primeras tres variables no tienen secretos para nosotros a estas alturas:

```
int valorLDR_Dcha = 0;
int valorLDR_Izda = 0;
int valorServo = 0;
```

Estas tres líneas declaran tres variables de tipo entero y las inicializan a cero.

Pero la siguiente es harina de otro costal, échale un vistazo:

```
Servo servo1;
```

Lo que hemos hecho es declarar una instancia del objeto Servo. Esta instancia se llamará, a partir de ahora, servo1.

También ha cambiado algo dentro de setup. Aparece una línea curiosa, concretamente la siguiente:

```
servo1.attach(10);
```

Esta línea significa que tenemos un servo conectado al pin 10.

Cosas importantes:

Sólo podremos conectar servos a las salidas analógicas de arduino, básicamente porque en el control de servos se utiliza una señal PWM.

La última linea rara de esta práctica es la siguiente:

```
servo1.write(valorServo);
```

Efectivamente, esto envía un valor al servo codificado como una señal PWM. A pesar de que tampoco sería muy complejo de hacer “a mano”, lo cierto es que estar calculando amplitudes de pulsos no es lo que más disfruto haciendo, y eso es lo que debió pensar la persona que escribió la librería servo. El valor que tenemos que enviar es el ángulo que queremos que gire.

Los servos, por lo general, sólo giran 180º así que podremos poner valores comprendidos entre 0 y 179.

Pachube.

Estamos llegando al final de esto y no, no me he equivocado al escribir lo anterior. Pachube es un servicio que permite monitorizar el estado de sensores a través de internet y cómo no, podemos utilizarlo con Arduino.

Pachube es accesible a través de la dirección www.pachube.com es un servicio que requiere que te registres, pero dispone de funciones gratuitas que nos pueden permitir estudiar el estado de un Arduino remotamente e interactuar con él.

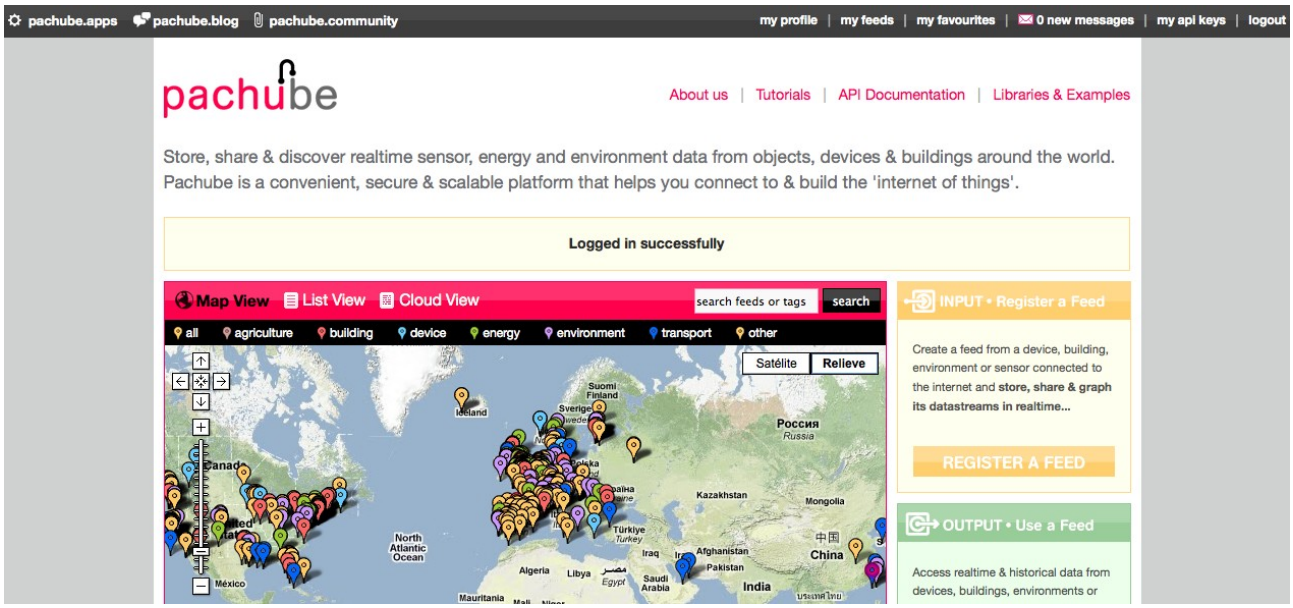


Ilustración 31: Página principal de Pachube

Ilustración 32: Un feed Pachube de la cuesta.

La familia de Arduino.

Fritzing.

Fritzing es la herramienta con la que he diseñado todos los esquemas de circuitos de este curso.

Dispone de una interfaz muy intuitiva y permite contribuir con el proyecto creando nuevos componentes que podrán utilizarse, a partir de ese momento, en la aplicación.

Es software libre y además sigue en desarrollo.

Con Fritzing es posible hacer diagramas basados en protoboards, esquemas electrónicos clásicos y diseño básico de PCB's con autorouting de pistas.

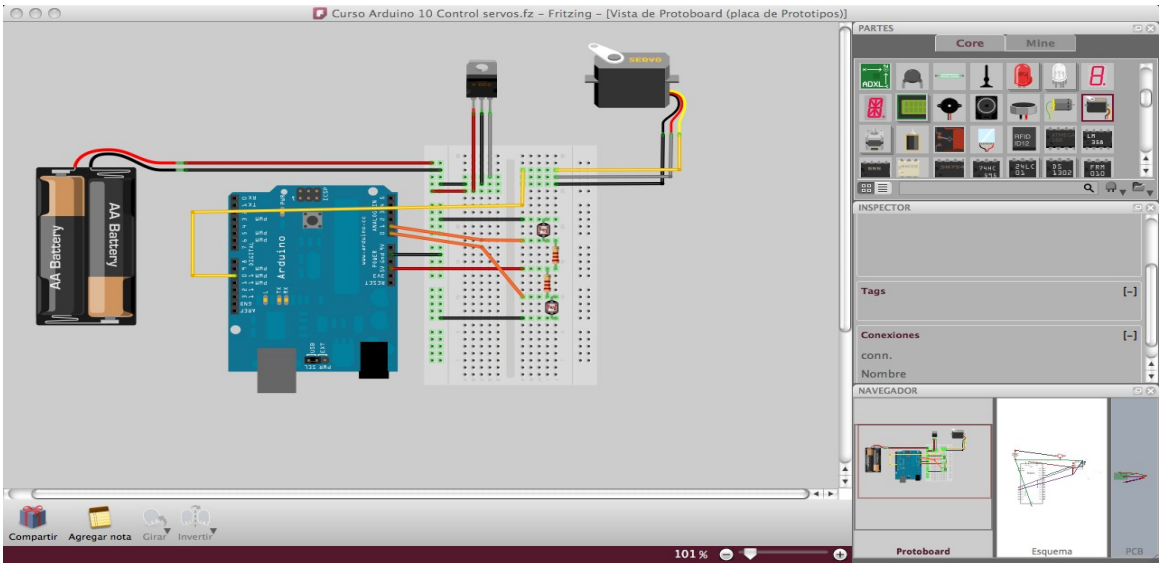


Ilustración 33: Pantalla principal de Fritzing, vista protoboard

Processing.

Processing es uno de los abueletes de Arduino, además, bastará que veas la interfaz de usuario para que comprendas la relación familiar.

Processing es un lenguaje de programación orientado a la representación gráfica, permite hacer representaciones en 2D y 3D y como complemento de Arduino (o viceversa) no tiene precio.

A continuación les incluyo una imagen generada mediante Processing y el código correspondiente.

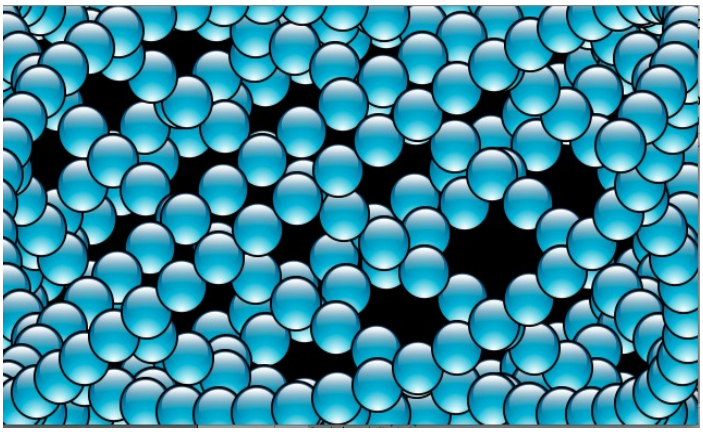


Ilustración 34: Imagen generada mediante Processing

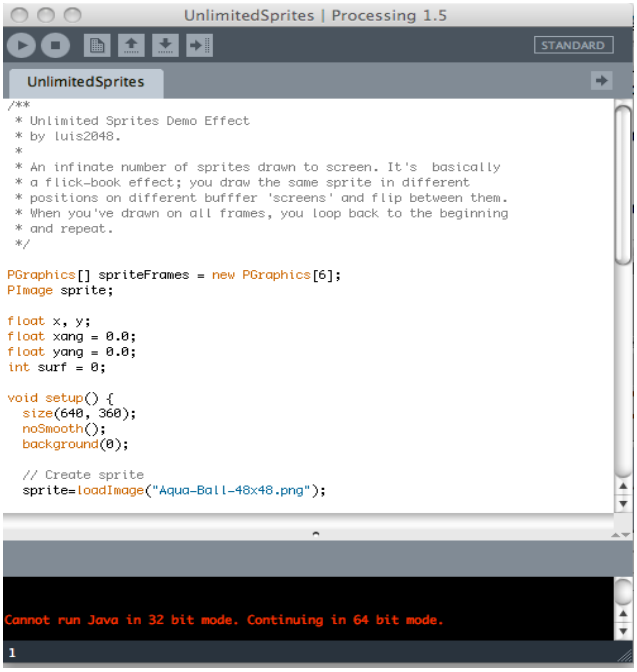


Ilustración 35: Ventana de Processing.

¿Cómo usar Arduino en docencia?

Robótica básica.

Fabricación de elementos útiles para el taller.

Sistemas de comunicaciones.

Domótica.

Telemática.

Televisión y videojuegos, sonido.

Idiomas.

Participación en proyectos internacionales.

¿Dónde buscar ayuda?

1. Página web oficial de Arduino: http://arduino.cc/es/Reference/HomePage
2. La Comunidad de Arduino, sobre todo a través de los foros y listas de correo. Si de algo pueden estar orgullosos en Arduino.cc es de haber generado uno de los ecosistemas más entusiastas y colaboradores del panorama del hardware y software abierto.

Recursos.

1. Make Magazine.
2. Blog de Bricogeek.
3. Sparkfun.
4. Video tutoriales youtube.

Referencias y Webografía

- Página web oficial de Arduino: http://www.arduino.cc
- Laboratorio de creación e innovación tecnológica de Canarias (Canarnova): https://groups.google.com/group/canarnova?hl=es
- Blog de bricogeek: http://blog.bricogeek.com/
- Sparkfun electronics: http://www.sparkfun.com/

- Make Magazine: http://makezine.com/
- Fritzing: http://fritzing.org/
- KiCAD: http://kicad.sourceforge.net/wiki/Main_Page

Recursos gráficos utilizados en este curso:

- La imagen del coche fantástico procede de la Wikipedia.
- El rostro de la sección sobre el IDE de Arduino ha salido de opencliparts.org
- La imagen de un ordenador que aparece en la sección sobre comunicaciones serie también la he obtenido de opencliparts.org http://www.openclipart.org/detail/129931
- La gráfica PWM ha sido extraída de: http://upload.wikimedia.org/wikipedia/commons/thumb/8/8e/PWM%2C_3-level.svg/350px-PWM%2C_3-level.svg.png
- La fotografía de la resistencia LDR proviene de la Wikipedia: http://upload.wikimedia.org/wikipedia/commons/a/a2/Fotocelda.jpg
- La fotografía del servo proviene de la Wikipedia: http://upload.wikimedia.org/wikipedia/commons/d/d2/Servomotor_01.jpg
- La fotografía de los reguladores de voltaje proviene de la Wikipedia: http://upload.wikimedia.org/wikipedia/commons/d/d0/7800_IC_regulators.jpg

He acabado este curso el día después del XXXIV Cross María Auxiliadora, lo que significa que ayer cronometramos a 2317 participantes.

El software con el que se hizo también es software libre.

El mundo se cambia cambiándolo.