Introducción a Arduino I:

Jugando con Leds y pulsadores

"El verdadero progreso es el que pone la tecnología al alcance de todos."

(Henry Ford)

Índice:

Materiales: por alumno

- 1 x Arduino Uno, con su cable USB.
- 5 x Led (colores varios).
- 5x resistencias 220 ohm.
- 15 x cables de conexión.
- 2 x Pulsadores PCB
- 2 x Resistencias 10K
- 1 x Protoboard
- 1 x Ordenador (PC) por alumno.

Glosario:

ATmega8U2: Microcontrolador usado en las placas Arduino para establecer la conexión USB

Bit: es el acrónimo de *Binary digit* o dígito binario. Un bit es un dígito del sistema de numeración binario

Bootloader: Es un gestionador de arranque. Es una programación de que disponen todos los micros de Arduino para facilitar la comunicación con el PC y la programación.

Byte: secuencia de 8 bits contiguos.

CAD: *Computer Aided Design* o diseño asistido por ordenador

CAN: es un protocolo de comunicaciones desarrollado por la firma alemana Robert Bosch GmbH, basado en una topología bus para la transmisión de mensajes en ambientes distribuidos, además ofrece una solución a la gestión de la comunicación entre múltiples CPUs (unidades centrales de proceso).

Clavija: Parte macho de un conector.

CPU: *Central Proces Unit* o unidad central de proceso

Conversor A/D: Dispositivo electrónico encargado de convertir una señal analógica en una sucesión de bits o señal digital

Drivers: Son pequeños programas que permiten que el Sistema Operativo sepa utilizar las capacidades de un periférico. Software que se encarga de interactuar entre el sistema operativo y los dispositivos (hardware).

DFU: (Device Firmware Update), Actualización del Firmware del Dispositivo se trata de un modo que disponen muchos dispositivos que permite reprogramarle. Es posible reprogramar el dispositivo mientras esta encendido sin necesidad de programador externo.

EEPROM: son las siglas de *Electrically-Erasable Programmable Read-Only Memory* (ROM programable y borrable eléctricamente). En español se la suele denominar "E²PROM" y en inglés "E-Squared-PROM". Es un tipo de memoria no volátil que puede ser programado, borrado y reprogramado eléctricamente.

Firmware: Programación interna de un dispositivo que le permite controlar su propio hardware.

FLASH: La memoria flash es una forma desarrollada de la memoria EEPROM que permite que múltiples posiciones de memoria sean escritas o borradas en una misma operación de programación mediante impulsos eléctricos, frente a las anteriores que sólo permite escribir o borrar una única celda cada vez. Por ello, flash permite funcionar a velocidades muy

superiores cuando los sistemas emplean lectura y escritura en diferentes puntos de esta memoria al mismo tiempo.

I2C: es un bus de comunicaciones serie. Su nombre viene de *Inter-Integrated Circuit.* La principal característica de **I²C** es que utiliza dos líneas para transmitir la información: una para los datos y por otra la señal de reloj.

PAD: o huella de un componente es la forma que deben tener las pistas del circuito impreso para recibir el componente y así pueda ser soldado correctamente. Cada componente tiene una huella distinta.

Patilla: también llamado **terminal**. Son los contactos terminales de un conector o componente electrónico, fabricado de un material conductor de la electricidad. Estos se utilizan para poder realizar una conexión sin soldar.

PCB: *Printed Circuit Board* o circuito impreso, es un medio para sostener mecánicamente y conectar eléctricamente componentes electrónicos, a través de *rutas* o *pistas* de material conductor, grabados en hojas de cobre laminadas sobre un sustrato no conductor.

Pines: también llamados **terminales**. Son los contactos terminales de un conector o componente electrónico, fabricados de un material conductor de la electricidad. Estos se utilizan para poder realizar una conexión sin soldar.

Puerto COM: Un puerto serie o puerto serial es una interfaz de comunicaciones de datos digitales, frecuentemente utilizada por computadoras y periféricos, donde la información es transmitida bit a bit enviando un solo bit a la vez. Puerto muy usado hasta la aparición del USB (que es una versión del COM).

Resistencias Pull-up: Son resistencias que permiten que una entrada nunca quede desconectada. Son de valor alto (5K o 10K) y conectan la entrada con el negativo o positivo de modo que en ausencia de señal el valor en la entrada es bajo o alto respectivamente.

PWM: *Pulse-Width Modulation* en inglés o modulación por ancho de pulsos de una señal o fuente de energía es una técnica en la que se modifica el ciclo de trabajo de una señal periódica (por ejemplo sinusoidal o cuadrada) ya sea para transmitir información a través de un canal de comunicaciones o como control de la cantidad de energía que se envía a una carga.

RAM: (en inglés: *Random Access Memory* cuyo acrónimo es **RAM**) es la memoria desde donde el procesador recibe las instrucciones y guarda los resultados.

Rpm: revoluciones por minuto

SRAM: Static Random Access Memory o memoria estática de acceso aleatorio.

SMD: *Surface Mount Device* o dispositivos de montaje superficial. Se trata de los componentes electrónicos que se sueldan sobre la superficie de una placa electrónica sin necesidad de hacer agujeros

Socket: en inglés zócalo. Parte hembra de un conector.

SPI: (Serial Peripheral Interface Bus): Es un bus de comunicaciones serie estandarizado por Motorola que opera en modo full dúplex. Los dispositivos son esclavos/maestros donde el maestro inicia las comunicaciones. Este bus admite múltiples dispositivos conectados simultáneamente con distintos números de esclavo identificativos. Esta comunicación necesita 4 hilos.

Sketch: Un sketch es el nombre que usa Arduino para un programa. Es la unidad de código que se sube y ejecuta en la placa Arduino.

TTL: es la sigla en inglés de transistor-transistor logic, es decir, "lógica transistor a transistor". Es una familia lógica o lo que es lo mismo, una tecnología de construcción de circuitos electrónicos digitales. En los componentes fabricados con tecnología TTL los elementos de entrada y salida del dispositivo son transistores bipolares. Los niveles lógicos vienen definidos por el rango de tensión comprendida entre 0,2V y 0,8V para el estado L (bajo) y los 2,4V y Vcc para el estado H (alto).

TWI: I2C es un bus de comunicaciones en serie. Su nombre viene de Inter-Integrated Circuit. Se necesitan solamente dos líneas, la de datos (SDA) y la de reloj (SCL). Cada dispositivo conectado al bus tiene un código de dirección seleccionable mediante software. Habiendo permanentemente una relación Master/ Slave entre el micro y los dispositivos conectados. El bus permite la conexión de varios Masters, ya que incluye un detector de colisiones. El protocolo de transferencia de datos y direcciones posibilita diseñar sistemas completamente definidos por software. Los datos y direcciones se transmiten con palabras de 8 bits.

USB: *Universal Serial Bus* o bus universal serie, abreviado comúnmente USB, es un puerto que sirve para conectar periféricos a un ordenador.

UART: El corazón del sistema de comunicaciones serie es la UART, acrónimo de Universal Asynchronous Receiver-Transmitter. Es un chip cuya misión principal es convertir los datos recibidos del bus del PC en formato paralelo, a un formato serie que será utilizado en la transmisión hacia el exterior. También realiza el proceso contrario: transformar los datos serie recibidos del exterior en un formato paralelo entendible por el bus.

1 ¿Qué es FabLab León?

La misión de Fab Lab León es convertirse en un ecosistema que aliente y ayude a las personas a construir (casi) todo lo que puedan imaginar.

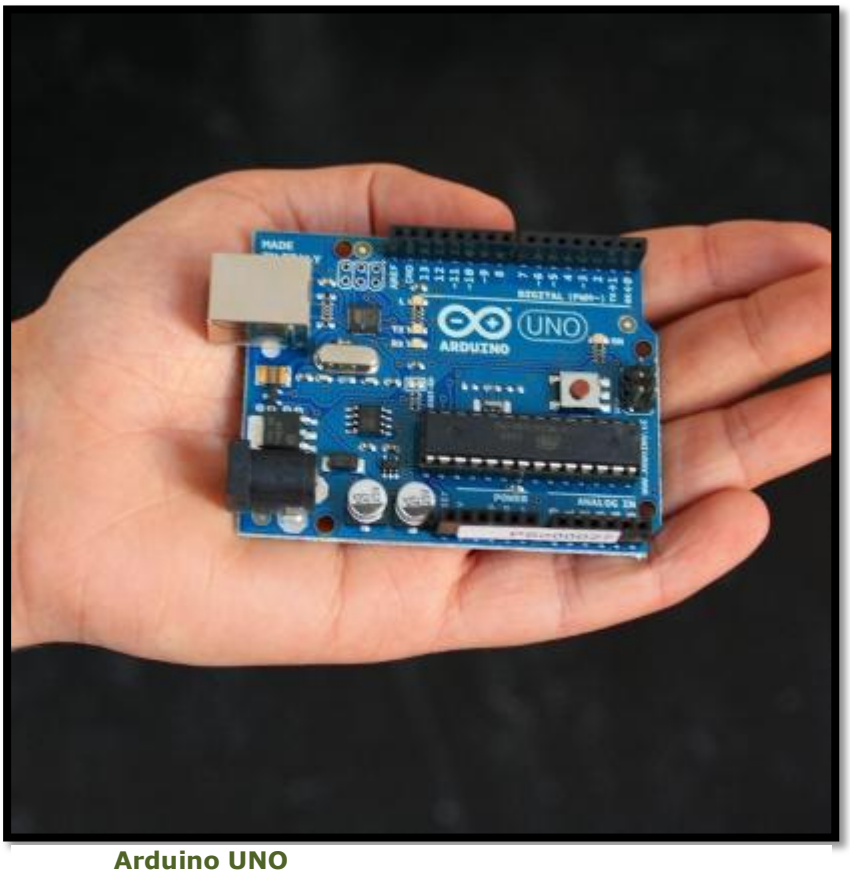
Un Fab Lab es un proyecto de la organización sin ánimo de lucro tMA, y consiste en un espacio de trabajo comunitario, a un coste razonable, que sirve como incubadora de investigación, de iniciativas creativas y de negocios, que permite el crecimiento personal, el desarrollo económico e incrementa las capacidades locales.

El concepto de Fab Lab refleja un nuevo paradigma de la fabricación en el que las personas definen los problemas, crean las soluciones y comercializan los productos.

El Fab Lab elimina barreras como el capital fundacional, el acceso a equipamiento y el acceso a expertos, alentando un cambio sistémico de los entornos educacionales, de innovación y de emprendimiento.

2 Plataforma electrónica Arduino UNO

Arduino es una plataforma de electrónica abierta para la creación de prototipos basada en software y hardware flexibles y fáciles de usar. Se creó para artistas, diseñadores, aficionados y cualquiera interesado en crear entornos u objetos interactivos.

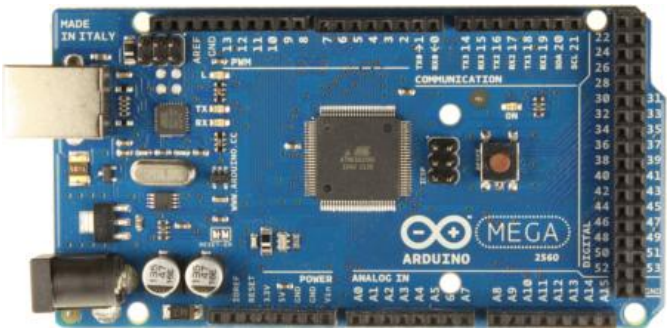


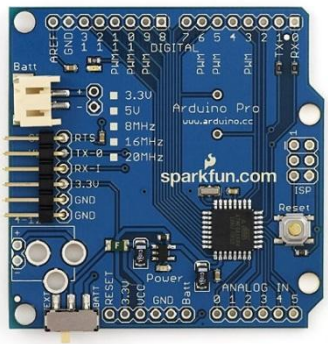
Arduino UNO

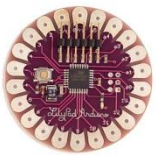
Arduino puede tomar información del entorno a través de sus pines de entrada de toda una gama de sensores y puede afectar aquello que le rodea controlando luces, motores y otros actuadores. El microcontrolador en la placa Arduino se programa mediante el lenguaje de programación Arduino (basado en Wiring) y el entorno de desarrollo Arduino (basado en Processing). Los proyectos hechos con Arduino pueden ejecutarse sin necesidad de conectar a un ordenador, si bien tienen la posibilidad de hacerlo y comunicar con diferentes tipos de software (p.ej. Flash, Processing, MaxMSP).

Las placas pueden ser hechas a mano o compradas montadas de fábrica; el software puede ser descargado de forma gratuita. Los ficheros de diseño de referencia (CAD) están disponibles bajo una licencia abierta, así pues eres libre de adaptarlos a tus necesidades.

Existen diversas tarjetas con diferentes microcontroladores, cada una con unas características concretas. Nosotros usaremos Arduino UNO por ser el buque insignia de la familia Arduino. Además una vez que eres capaz de comprender el funcionamiento de la UNO, ya puedes usar cualquiera de las demás.







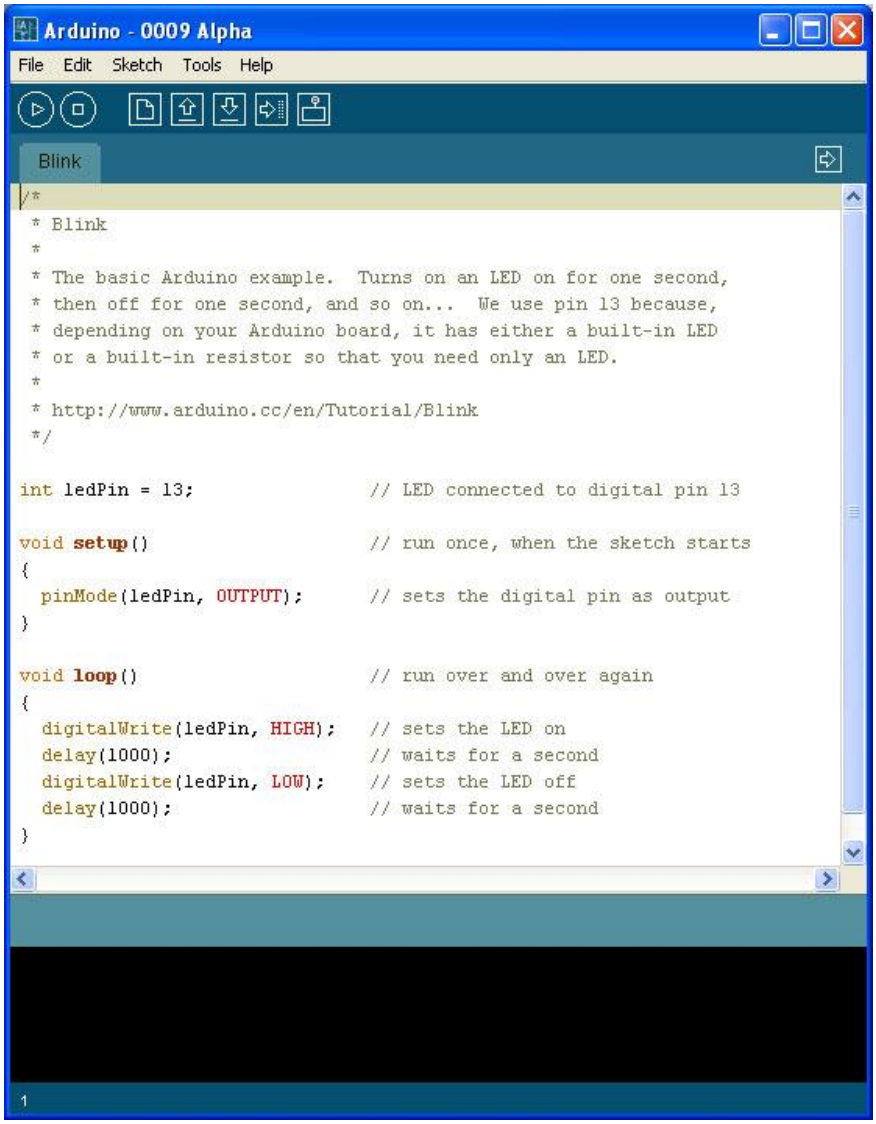
3 Software de Arduino

La Arduino Uno tiene una gran cantidad de posibilidades para la comunicación con un ordenar, u otros microcontroladores. El microcontrolador de esta placa (Atmega328) proporciona comunicación serie UART TTL (5V), que está disponible en los pines digitales 0 (RX) y 1 (TX). El ATmega8U2 integrado en la tarjeta permite transfiere esta comunicación sobre el USB y aparece con un puerto COM virtual para el software del ordenador. El firmware del 8U2 emplea los drivers estándar para el puerto USB, por lo tanto no son necesarios drivers externos. Sin embargo bajo Windows es necesario un archivo .inf. El software the Arduino incluye un monitor serie que permite el envío y recepción de datos de texto plano entre el ordenador y la tarjeta. Los leds de RX y TX se iluminaran durante la transmisión de los datos vía el chip conversor USB-a-serie y la conexión USB al ordenador

La librería SoftwareSerial permite la comunicación serie en cualquiera de los pines digitales de la Uno.

El ATmega328 también soporta comunicaciones I2C (TWI) y SPI. El software de Arduino incluye la librería Wire para simplificar el uso del bus I2C; ver la documentación para más información. Para la comunicación SPI se usa la librería SPI.

Encabezado del software de Arduino



Sencillez en la interfaz del software Arduino

3.1 Programación

El Arduino Uno puede ser programado con el software gratuito Arduino (descargable de su página web). Una vez descargado no es necesario instalarlo basta con descomprimir la carpeta y ejecutar arduino.exe.

A continuación se debe conectar la tarjeta con su cable USB. Windows tratara de buscar el controlador adecuado, como no lo encontrara, nosotros debemos de actualizar el controlador.

Para ello debemos acceder al administrador de dispositivos y hacer clic derecho sobre el dispositivo recién instalado, actualizamos el controlador eligiendo como carpeta donde se encuentra el driver la de Arduino (recién descomprimida).

Una vez hecho lo anterior estamos listos para comenzar a programar la tarjeta. Solo es necesario una vez en la interfaz seleccionar "Arduino Uno" en el menú Tools > Board.

El chip ATmega328 que usa Arduino Uno viene pre programado con un bootloader (consultar glosario) que permite introducirle nuevo código sin necesidad de un programador externo. Se comunica usando el protocolo STK500.

También es posible ignorar el bootloader y programar el microcontrolador a través del conector ICSP (In-Circuit Serial Programming).

El firmware que usa el ATmega8U2 también está disponible. Este chip está cargado con un bootloader DFU, que puede ser activado conectando el jumper de soldadura en la parte posterior de la placa y posteriormente reseteando el 8U2. Así después se podrá hacer la programación con un programador DFU (no usado en este taller).

5 Estructura básica de Lenguaje Arduino

La estructura básica del lenguaje de programación de Arduino es bastante simple y se compone de al menos dos partes. Estas dos partes necesarias, o funciones, encierran bloques que contienen declaraciones, estamentos o instrucciones.

```
void setup()
{
      instrucciones;
}
void loop()
{
      instrucciones;
}
```

En donde setup() es la parte encargada de recoger la configuración y loop() es la que contienen el programa que se ejecutará cíclicamente (de ahí el termino loop –bucle-). Ambas funciones son necesarias para que el programa funcione correctamente. La función de configuración debe contener la declaración de las variables. Es la primera función a ejecutar en el programa, se ejecuta sólo una vez, y se utiliza para configurar o inicializar pinMode (modo de trabajo de las E/S), configuración de la comunicación en serie y otras.

La función bucle (loop) siguiente contiene el código que se ejecutara continuamente (lectura de entradas, activación de salidas, etc.) Esta función es el núcleo de todos los programas de Arduino y la que realiza la mayor parte del trabajo.

5.1 setup()

La función setup() se invoca una sola vez cuando el programa empieza. Se utiliza para inicializar los modos de trabajo de los pines, o el puerto serie. Debe ser incluido en un programa aunque no haya declaración que ejecutar. Así mismo se puede utilizar para establecer el estado inicial de las salidas de la placa.

```
void setup()
{
pinMode(pin, OUTPUT); // configura el 'pin' como
salida
digitalWrite(pin, HIGH); // pone el ‘pin’ en estado
// HIGH
}
```

5.2 loop()

Después de finalizar setup(), la función loop() hace precisamente lo que sugiere su nombre, se ejecuta de forma cíclica, lo que posibilita que el programa este respondiendo continuamente ante los eventos que se produzcan en la placa.

```
void loop()
{
digitalWrite(pin, HIGH); // pone en uno (on, 5v) el´pin´
delay(1000); // espera un segundo (1000 ms)
digitalWrite(pin, LOW); // pone en cero (off, 0v.) el
delay(1000); // ´pin´
}
```

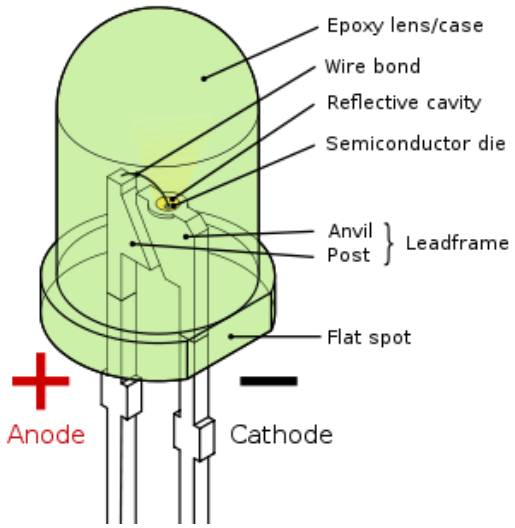
6 Ejemplos a implementar

6.1 Led intermitente

En la mayoría de los lenguajes de programación, el primer programa que escribes imprime en la pantalla del ordenador la frase "Hola Mundo". Ya que una placa Arduino no tiene una pantalla (ya la se la pondremos), podemos hacer parpadear un Led.

Las placas están diseñadas para que hacer parpadear un Led sea muy fácil. Algunas tienen un Led directamente incorporado en la placa para este pin. Si no lo tiene, usarlo es igual de sencillo solo necesitamos una resistencia de 220 Ohm y el Led.

Los Leds tienen polaridad, lo que significa que solo encenderán si están correctamente polarizados. El terminal más largo es el positivo y deberá estar conectado al pin elegido. El terminal corto debe conectarse con la tierra (GND), la capsula del LED tiene un borde plano en este extremo. Si el LED no enciende, trata de conectarlo de manera opuesta, intercambiando los terminales de posición (no dañaras el LED si lo conectas en sentido opuesto).

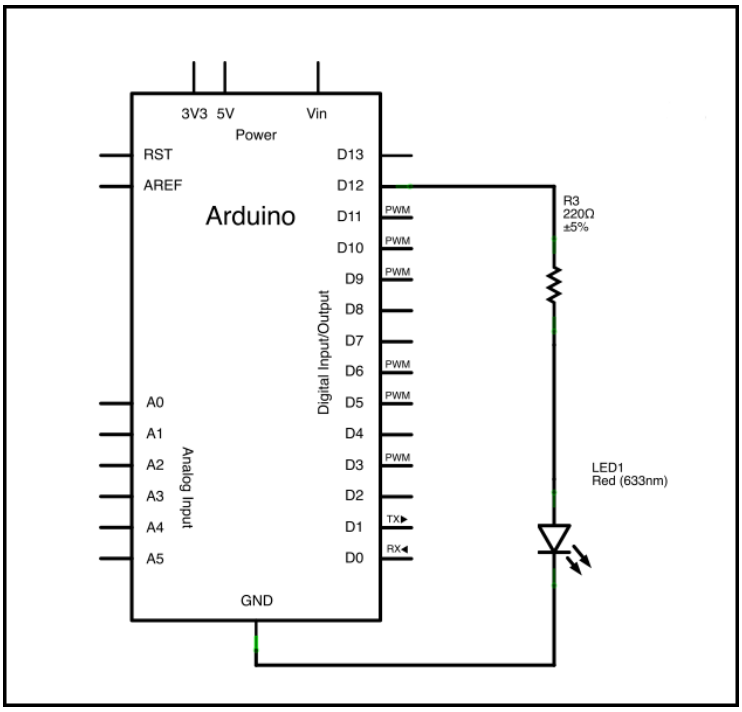


6.1.1 Código

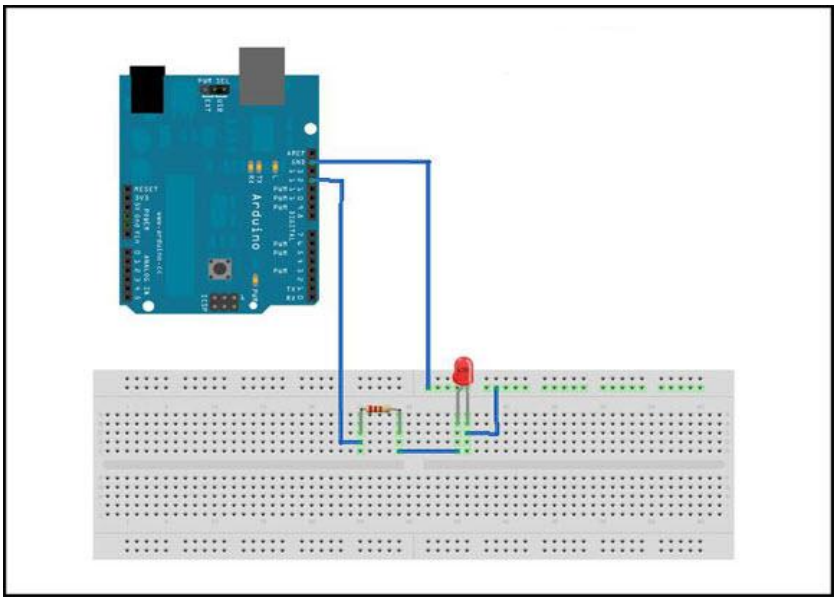
El código de ejemplo es sencillo:

```
/*
  Blink
  Turns on an LED on for one second, then off for one second,
repeatedly.

  This example code is in the public domain.
 */

void setup() {
  // initialize the digital pin as an output.
  // Pin 12 has an LED connected on most Arduino boards:
  pinMode(12, OUTPUT);
}

void loop() {
  digitalWrite(12, HIGH);   // set the LED on
  delay(1000);              // wait for a second
  digitalWrite(12, LOW);    // set the LED off
  delay(1000);              // wait for a second
}
```

6.1.2 Circuito



6.1.3 Montaje en Protoboard



6.1.4 Ejercicio 1:

Construir y programar una tira de 5 Leds que simule el efecto "coche fantástico"

6.2 Botones o pulsadores

Los pulsadores o switches conectan dos puntos de un circuito al ser pulsados.

Conecta tres cables a la placa Arduino. Los dos primeros, rojo y negro, conectan a las dos hileras largas verticales de los laterales de la placa universal (*breadboard*) para proporcionar acceso a la fuente de 5 voltios y a masa (*ground*). El tercer cable va desde el pin digital 2 a una patilla del pulsador. Esa misma patilla del pulsador se conecta a través de una resistencia *pull-down* (en este caso 10 KOhms) a masa. El otro extremo del pulsador se conecta a la fuente de 5 voltios.

Cuando el pulsador está abierto (sin pulsar) no hay conexión entre las dos patas del pulsador, de forma que el pin está conectado a tierra (a través de la resistencia *pull-down*) y leemos un LOW (bajo ó 0). Cuando el botón se cierra (pulsado), se establece la unión entre sus dos extremos, conectando el pin a 5 voltios, por lo que leemos un HIGH (alto ó 1).

También puedes cablear el circuito en sentido contrario, con una resistencia "pull-up" que mantenga la entrada en HIGH, y que pase a LOW cuando se pulse el botón. Así, el comportamiento del programa (sketch) se invertirá, con el LED normalmente encendido y apagado cuando se pulsa el botón.

Si se desconectas el pin digital de E/S del todo, el LED puede parpadear de forma errática. Esto se debe a la entrada es "flotante", es decir, al azar se tornará en HIGH o LOW. Por eso se necesita la resistencia *pull-up* o *pull-down* en el circuito.

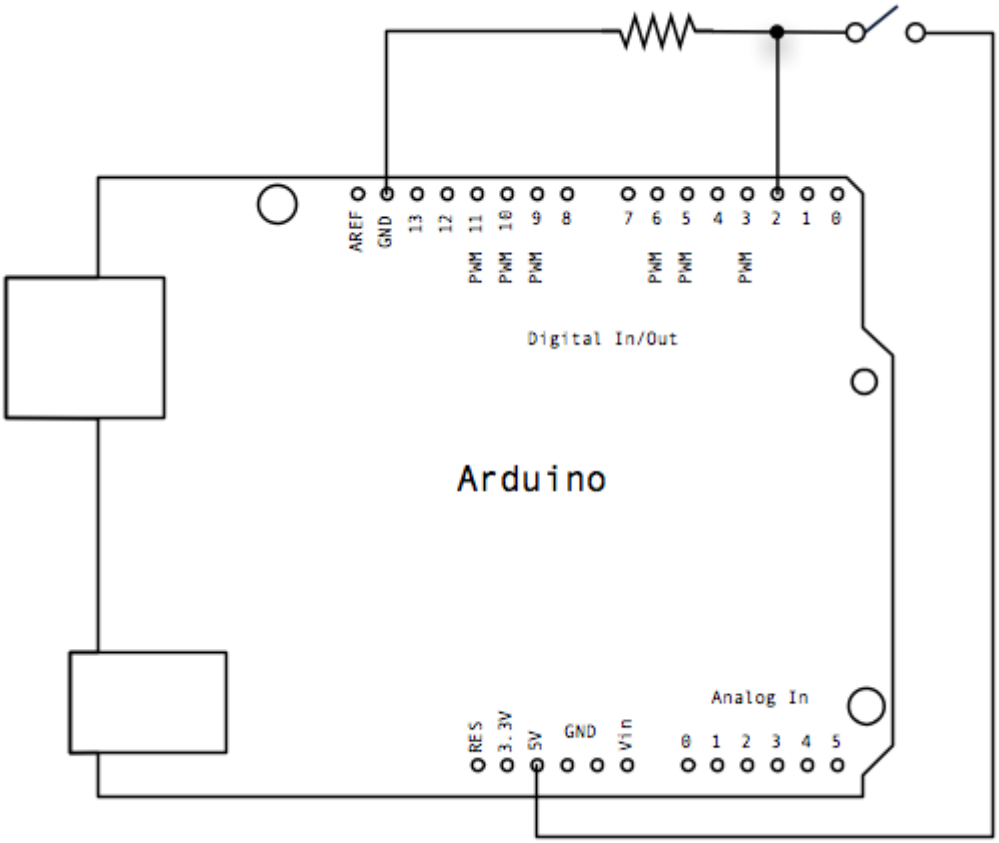
Combinando el primer ejemplo y este podemos controlar el led con el pulsador.

6.2.1 Código

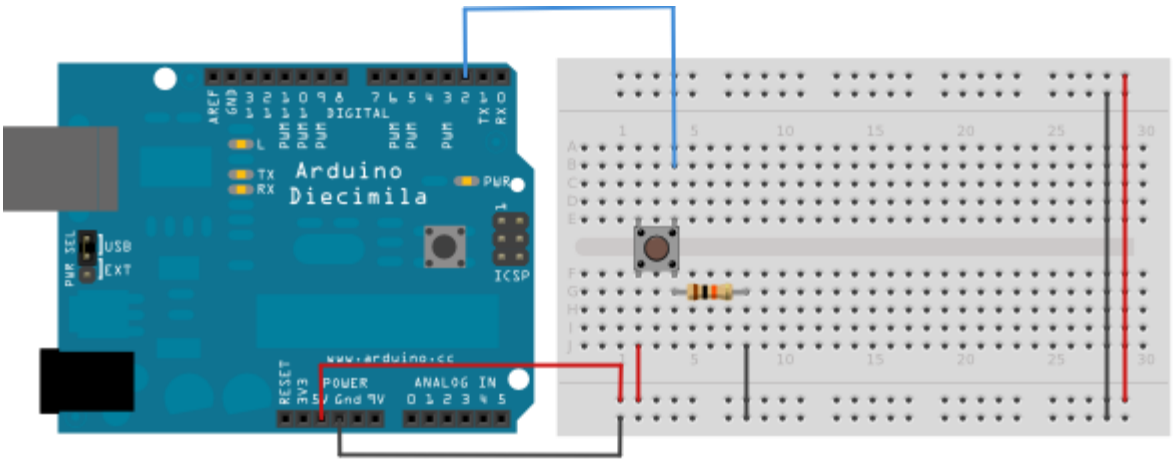
No te preocupes si ahora no comprendes todo el código, basta con que sepas modificar alguna parte y veas lo que va sucediendo.

```
/*
  Button
*/

// constants won't change. They're used here to
// set pin numbers:
const int buttonPin = 2;     // the number of the pushbutton pin
const int ledPin =  12;      // the number of the LED pin

// variables will change:
int buttonState = 0;         // variable for reading the pushbutton
status

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
}

void loop(){
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed.
  // if it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  }
  else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

6.2.2 Circuito



6.2.3 Montaje en Protoboard



6.2.4 Ejercicio 2:

A partir del ejercicio 1 hacer que la tira de leds se detenga al pulsar el botón.

6.2.5 Ejercicio 3 complementario:

Con los montajes anteriores colocar 2 pulsadores uno que encienda el efecto "coche fantástico" y otro que lo apague.

Páginas web de consulta:

http://www.arduino.cc/

http://www.fablableon.org/

http://www.arduinoacademy.com/

Si tienes alguna duda pregúntame:

Arduino y electrónica Fab Lab León: David Benéitez

david.beneitez@fablableon.org

7 Anexo: Guía de lenguaje Arduino.

7.1 Funciones

Una función es un bloque de código que tiene un nombre y un conjunto de instrucciones que son ejecutadas cuando se llama a la función. Son funciones setup() y loop() de las que ya se ha hablado. Las funciones de usuario pueden ser escritas para realizar tareas repetitivas y para reducir el tamaño de un programa. Las funciones se declaran asociadas a un tipo de valor. Este valor será el que devolverá la función, por ejemplo 'int' se utilizara cuando la función devuelva un dato numérico de tipo entero. Si la función no devuelve ningún valor entonces se colocara delante la palabra "void", que significa "función vacía". Después de declarar el tipo de dato que devuelve la función se debe escribir el nombre de la función y entre paréntesis se escribirán, si es necesario, los parámetros que se deben pasar a la función para que se ejecute.

```
tipo nombreFunción(parámetros)
{
instrucciones;
}
```

La función siguiente devuelve un numero entero, delayVal() se utiliza para poner un valor de retraso en un programa que lee una variable analógica de un potenciómetro conectado a una entrada de Arduino. Al principio se declara como una variable local, ´v´ recoge el valor leído del potenciómetro que estará comprendido entre 0 y 1023, luego se divide el valor por 4 para ajustarlo a un margen comprendido entre 0 y 255, finalmente se devuelve el valor ´v´ y se retornaría al programa principal.

```
int delayVal()
{
int v; // crea una variable temporal 'v'
v= analogRead(pot); // lee el valor del potenciómetro
v /= 4; // convierte 0-1023 a 0-255
return v; // devuelve el valor final
}
```

7.2 {} entre llaves

Las llaves sirven para definir el principio y el final de un bloque de instrucciones. Se utilizan para los bloques de programación setup(), loop(), if.., etc.

```
type función()
{
instrucciones;
}
```

Una llave de apertura "{ " siempre debe ir seguida de una llave de cierre " }" , si no es así el compilador dará errores. El entorno de programación de Arduino incluye una herramienta de gran utilidad para comprobar el total de llaves. Solo tienes que hacer clic en el punto de inserción de una llave abierta e inmediatamente se marca el correspondiente cierre de ese bloque (llave cerrada).

7.3 ; punto y coma

El punto y coma ";" se utiliza para separar instrucciones en el lenguaje de programación de Arduino. También se utiliza para separar elementos en una instrucción de tipo "bucle for" .

```
int x = 13; // declara la variable 'x' como tipo
// entero de valor 13
```

Nota: Si olvidáis poner fin a una línea con un punto y coma se producirá en un error de compilación. El texto de error puede ser obvio, y se referirá a la falta de un punto y coma, o puede que no. Si se produce un error raro y de difícil detección lo primero que debemos hacer es comprobar que los puntos y comas están colocados al final de las instrucciones.

7.4 /*... */ bloque de comentarios

Los bloques de comentarios, o comentarios multi-linea son áreas de texto ignorados por el programa que se utilizan para las descripciones del código o comentarios que ayudan a comprender el programa. Comienzan con / * y terminan con * / y pueden abarcar varias líneas.

```
/* esto es un bloque de comentario no se debe olvidar
cerrar los comentarios estos deben estar equilibrados */
```

Debido a que los comentarios son ignorados por el compilador y no ocupan espacio en la memoria de Arduino pueden ser utilizados con generosidad. También pueden utilizarse para "comentar" bloques de código con el propósito de anotar informaciones para depuración y hacerlo más comprensible para cualquiera.

Nota: Dentro de una misma línea de un bloque de comentarios no se puede escribir otra bloque de comentarios (usando /*..*/).

7.5 // línea de comentarios

Una línea de comentario empieza con / / y terminan con la siguiente línea de código. Al igual que los comentarios de bloque, los de línea son ignoradas por el programa y no ocupan espacio en la memoria.

```
// Esto es un comentario
```

Una línea de comentario se utiliza a menudo después de una instrucción, para proporcionar más información acerca de lo que hace esta o para recordarla más adelante.

7.6 Variables

Una variable es una manera de nombrar y almacenar un valor numérico para su uso posterior por el programa. Como su nombre indica, las variables son números que se pueden variar continuamente en contra de lo que ocurre con las constantes cuyo valor nunca cambia. Una variable debe ser declarada y, opcionalmente, asignarle un valor. El siguiente código de ejemplo declara una variable llamada variableEntrada y luego le asigna el valor obtenido en la entrada analógica del PIN2:

```
int variableEntrada = 0; // declara una variable y le
// asigna el valor 0
variableEntrada = analogRead(2); // la variable recoge
//el valor analógico del PIN2
```

'variableEntrada' es la variable en sí. La primera línea declara que será de tipo entero "int". La segunda línea fija a la variable el valor correspondiente a la entrada analógica PIN2. Esto hace que el valor de PIN2 sea accesible en otras partes del código. Una vez que una variable ha sido asignada, o re-asignada, usted puede probar su valor para ver si cumple ciertas condiciones, o puede utilizar directamente su valor. Como ejemplo ilustrativo veamos tres operaciones útiles con variables: el siguiente código prueba si la variable "entradaVariable" es inferior a 100, si es cierto se asigna el valor 100 a "entradaVariable" y, a continuación, establece un retardo (delay) utilizando como valor "entradaVariable" que ahora será como mínimo de valor 100:

```
if (entradaVariable < 100) // pregunta si la variable es
{ //menor de 100
entradaVariable = 100; // si es cierto asigna el valor
} //100
delay(entradaVariable); // usa el valor como retardo
```

Nota: Las variables deben tomar nombres descriptivos, para hacer el código más legible. Los nombres de variables pueden ser "contactoSensor" o "pulsador", para ayudar al programador y a cualquier otra persona a leer el código y entender lo que representa la variable. Nombres de variables como "var" o "valor", facilitan muy poco que el código sea inteligible. Una variable puede ser cualquier nombre o palabra que no sea una palabra reservada en el entorno de Arduino.

7.7 Declaración de variables

Todas las variables tienen que declararse antes de que puedan ser utilizadas. Para declarar una variable se comienza por definir su tipo como int (entero), long (largo), float (coma flotante), etc., asignándoles siempre un nombre, y, opcionalmente, un valor inicial. Esto solo debe hacerse una vez en un programa, pero el valor se puede cambiar en cualquier momento usando aritmética y reasignaciones diversas.

El siguiente ejemplo declara la variable entradaVariable como una variable de tipo entero "int", y asignándole un valor inicial igual a cero. Esto se llama una asignación.

```
int entradaVariable = 0;
```

Una variable puede ser declarada en una serie de lugares del programa y en función del lugar en donde se lleve a cabo la definición esto determinara en que partes del programa se podrá hacer uso de ella.

7.8 utilización de una variable

Una variable puede ser declarada al inicio del programa antes de la parte de configuración setup(), a nivel local dentro de las funciones, y, a veces, dentro de un bloque, como para los bucles del tipo if.. for.., etc. En función del lugar de declaración de la variable así se determinara el ámbito de aplicación, o la capacidad de ciertas partes de un programa para hacer uso de ella.

Una variable global es aquella que puede ser vista y utilizada por cualquier función y estamento de un programa. Esta variable se declara al comienzo del programa, antes de setup().

Una variable local es aquella que se define dentro de una función o como parte de un bucle. Solo es visible y solo puede utilizarse dentro de la función en la que se declaró.

Por lo tanto, es posible tener dos o más variables del mismo nombre en diferentes partes del mismo programa que pueden contener valores diferentes. La garantía de que solo una función tiene acceso a sus variables dentro del programa simplifica y reduce el potencial de errores de programación.

El siguiente ejemplo muestra como declarar a unos tipos diferentes de variables y la visibilidad de cada variable:

```
int value; // 'value' es visible para cualquier
función
void setup()
{
```

```
// no es necesario configurar nada en este ejemplo
}
void loop()
{
for (int i=0; i<20;) // 'i' solo es visible
{ // dentro del bucle for
i++
} // 'f' es visible solo
float f; // dentro de loop()
}
```

7.9 Tipos de datos

7.9.1 Byte

Byte almacena un valor numérico de 8 bits sin decimales. Tienen un rango entre 0 y 255.

```
byte unaVariable = 180; // declara 'unaVariable' como
// de tipo byte
```

7.9.2 Int

Enteros son un tipo de datos primarios que almacenan valores numéricos de 16 bits sin decimales comprendidos en el rango 32,767 to -32,768.

```
int unaVariable = 1500; // declara 'unaVariable' como
// una variable de tipo entero
```

Nota: Las variables de tipo entero "int" pueden sobrepasar su valor máximo o mínimo como consecuencia de una operación. Por ejemplo, si x = 32767 y una posterior declaración agrega 1 a x, x = x + 1 entonces el valor se x pasara a ser -32.768. (algo así como que el valor da la vuelta).

7.9.3 Long

El formato de variable numérica de tipo extendido "long" se refiere a números enteros (tipo 32 bits) sin decimales que se encuentran dentro del rango -2147483648 a 2147483647.

```
long unaVariable = 90000; // declara 'unaVariable' como
// de tipo long
```

7.9.4 Float

El formato de dato del tipo "punto flotante" "float" se aplica a los números con decimales. Los números de punto flotante tienen una mayor resolución que los de 32 bits con un rango comprendido 3.4028235E +38 a +38-3.4028235E.

```
float unaVariable = 3.14; // declara 'unaVariable' como
// de tipo flotante
```

Nota: Los números de punto flotante no son exactos, y pueden producir resultados extraños en las comparaciones. Los cálculos matemáticos de punto flotante son también mucho más lentos que los del tipo de números enteros, por lo que debe evitarse su uso si es posible.

7.9.5 arrays

Un array es un conjunto de valores a los que se accede con un número índice. Cualquier valor puede ser recogido haciendo uso del nombre de la matriz y el número del índice. El primer valor de la matriz es el que está indicado con el índice 0, es decir el primer valor del conjunto es el de la posición 0. Un array tiene que ser declarado y opcionalmente asignados valores a cada posición antes de ser utilizado.

```
int miArray[] = {valor0, valor1, valor2...}
```

Del mismo modo es posible declarar una matriz indicando el tipo de datos y el tamaño y posteriormente, asignar valores a una posición especifica:

```
int miArray[5]; // declara un array de enteros de 6
// posiciones
miArray[3] = 10; // asigna l valor 10 a la posición 4
```

Para leer de un array basta con escribir el nombre y la posición a leer:

```
x = miArray[3]; // x ahora es igual a 10 que está en
// la posición 3 del array
```

Las matrices se utilizan a menudo para estamentos de tipo bucle, en los que la variable de incremento del contador del bucle se utiliza como índice o puntero del array. El siguiente ejemplo usa una matriz para el parpadeo de un LED. Utilizando un bucle tipo for, el contador comienza en cero 0 y escribe el valor que figura en la posición de índice 0 en la serie que hemos escrito dentro del array parpadeo[], en este caso 180, que se envía a la salida analógica tipo PWM configurada en el PIN10, se hace una pausa de 200 ms y a continuación se pasa al siguiente valor que asigna el índice "i" .

```
int ledPin = 10; // LED en el PIN 10
byte parpadeo[] = {180, 30, 255, 200, 10, 90, 150, 60};
// array de 8 valores
void setup()
{
pinMode(ledPin, OUTPUT); // configura la salida
}
```

```
void loop()
{
for(int i=0; i<7; i++)
{
analogWrite(ledPin, parpadeo[i]);
delay(200); // espera 200ms
}
}
```

7.10 aritmética

Los operadores aritméticos que se incluyen en el entorno de programación son suma, resta, multiplicación y división. Estos devuelven la suma, diferencia, producto, o cociente (respectivamente) de dos operandos.

```
y = y + 3; x = x - 7; i = j * 6; r = r / 5;
```

La operación se efectúa teniendo en cuenta el tipo de datos que hemos definido para los operandos (int, dbl, float, etc..), por lo que, por ejemplo, si definimos 9 y 4 como enteros "int" , 9 / 4 devuelve de resultado 2 en lugar de 2,25 ya que el 9 y 4 se valores de tipo entero "int" (enteros) y no se reconocen los decimales con este tipo de datos.

Esto también significa que la operación puede sufrir un desbordamiento si el resultado es más grande que lo que puede ser almacenada en el tipo de datos. Recordemos el alcance de los tipos de datos numéricos que ya hemos explicado anteriormente. Si los operandos son de diferentes tipos, para el cálculo se utilizara el tipo más grande de los operandos en juego. Por ejemplo, si uno de los números (operandos) es del tipo float y otra de tipo integer, para el cálculo se utilizara el método de float es decir el método de coma flotante.

Elija el tamaño de las variables de tal manera que sea lo suficientemente grande como para que los resultados sean lo precisos que usted desea. Para las operaciones que requieran decimales utilice variables tipo float, pero sea consciente de que las operaciones con este tipo de variables son más lentas a la hora de realizarse el cómputo.

Nota: Utilice el operador (int) para convertir un tipo de variable a otro sobre la marcha. Por ejemplo, i = (int) 3,6 establecerá i igual a 3.

7.11 asignaciones compuestas

Las asignaciones compuestas combinan una operación aritmética con una variable asignada. Estas son comúnmente utilizadas en los bucles tal como se describe mas adelante. Estas asignaciones compuestas pueden ser:

```
x ++ // igual que x = x +1, o incremento de x en +1
x -- // igual que x = x - 1, o decremento de x en -1
```

```
x += y // igual que x = x + y, o incremento de x en +y
x -= y // igual que x = x - y, o decremento de x en -y
x *= y // igual que x = x * y, o multiplica x por y
x /= y // igual que x = x / y, o divide x por y
```

Nota: Por ejemplo, x * = 3 hace que x se convierta en el triple del antiguo valor x y por lo tanto x es reasignada al nuevo valor.

7.12 operadores de comparación

Las comparaciones de una variable o constante con otra se utilizan con frecuencia en las estructuras condicionales del tipo if.. para testear si una condición es verdadera. En los ejemplos que siguen en las próximas páginas se verá su utilización práctica usando los siguientes tipo de condicionales:

```
x == y // x es igual a y
x != y // x no es igual a y
x < y // x es menor que y
x > y // x es mayor que y
x <= y // x es menor o igual que y
x >= y // x es mayor o igual que y
```

7.13 operadores lógicos

Los operadores lógicos son usualmente una forma de comparar dos expresiones y devolver un VERDADERO o FALSO dependiendo del operador. Existen tres operadores lógicos, AND (&&), OR (||) y NOT (!), que a menudo se utilizan en estamentos de tipo if: Lógica AND:

```
if (x > 0 && x < 5) // cierto sólo si las dos
expresiones // son ciertas
```

Lógica OR:

```
if (x > 0 || y > 0) // cierto si una cualquiera de las
// expresiones es cierta
```

Lógica NOT:

```
if (!x > 0) // cierto solo si la expresión es
// falsa
```

7.14 constantes

El lenguaje de programación de Arduino tiene unos valores predeterminados, que son llamados constantes. Se utilizan para hacer los programas más fáciles de leer. Las constantes se clasifican en grupos.

7.15 cierto/falso (true/false)

Estas son constantes booleanas que definen los niveles HIGH (alto) y LOW (bajo) cuando estos se refieren al estado de las salidas digitales. FALSE se asocia con 0 (cero), mientras que TRUE se asocia con 1, pero TRUE también puede ser cualquier otra cosa excepto cero. Por lo tanto, en sentido booleano, -1, 2 y -200 son todos también se definen como TRUE. (esto es importante tenerlo en cuanta).

```
if (b == TRUE);
{
ejecutar las instrucciones;
}
```

7.16 High/Low

Estas constantes definen los niveles de salida altos o bajos y se utilizan para la lectura o la escritura digital para las patillas. ALTO se define como en la lógica de nivel 1, ON, o 5 voltios, mientras que BAJO es lógica nivel 0, OFF, o 0 voltios.

```
digitalWrite(13, HIGH); // activa la salida 13 con un
// nivel alto (5v.)
```

7.17 Input/Output

Estas constantes son utilizadas para definir, al comienzo del programa, el modo de funcionamiento de los pines mediante la instrucción pinMode de tal manera que el pin puede ser una entrada INPUT o una salida OUTPUT.

```
pinMode(13, OUTPUT); // designamos que el PIN 13 es
// una salida
```

7.18 Control de flujo

7.18.1 if (si condicional)

if es un estamento que se utiliza para probar si una determinada condición se ha alcanzado, como por ejemplo averiguar si un valor analógico está por encima de un cierto número, y ejecutar una serie de declaraciones (operaciones) que se escriben dentro de llaves, si es verdad. Si es falso (la condición no se cumple) el programa salta y no ejecuta las operaciones que están dentro de las llaves, El formato para if es el siguiente:

```
if (unaVariable ?? valor)
{
ejecuta Instrucciones;
}
```

En el ejemplo anterior se compara una variable con un valor, el cual puede ser una variable o constante. Si la comparación, o la condición entre paréntesis se

cumple (es cierta), las declaraciones dentro de los corchetes se ejecutan. Si no es así, el programa salta sobre ellas y sigue.

Nota: Tenga en cuenta el uso especial del símbolo '=', poner dentro de if (x = 10), podría parecer que es válido pero sin embargo no lo es ya que esa expresión asigna el valor 10 a la variable x, por eso dentro de la estructura if se utilizaría X==10 que en este caso lo que hace el programa es comprobar si el valor de x es 10.. Ambas cosas son distintas por lo tanto dentro de las estructuras if, cuando se

pregunte por un valor se debe poner el signo doble de igual "==" .

7.18.2 if... else (si..... sino ..)

if… else viene a ser un estructura que se ejecuta en respuesta a la idea "si esto no se cumple haz esto otro" . Por ejemplo, si se desea probar una entrada digital, y hacer una cosa si la entrada fue alto o hacer otra cosa si la entrada es baja, usted escribiría que de esta manera:

```
if (inputPin == HIGH)
{
instruccionesA;
}
else
{
instruccionesB;
}
```

Else puede ir precedido de otra condición de manera que se pueden establecer varias estructuras condicionales de tipo unas dentro de las otras (anidamiento) de forma que sean mutuamente excluyentes pudiéndose ejecutar a la vez. Es incluso posible tener un número ilimitado de estos condicionales. Recuerde sin embargo que solo un conjunto de declaraciones se llevara a cabo dependiendo de la condición probada:

```
if (inputPin < 500)
{
instruccionesA;
}
else if (inputPin >= 1000)
{
instruccionesB;
{
else
{
instruccionesC;
}
```

Nota: Un estamento de tipo if prueba simplemente si la condición dentro del paréntesis es verdadera o falsa. Esta declaración puede ser cualquier declaración valida. En el anterior ejemplo, si cambiamos y ponemos (inputPin == HIGH). En

este caso, el estamento if solo comprobaría si la entrada especificada esta en nivel alto (HIGH), o +5v.

7.18.3 for

La declaración for se usa para repetir un bloque de sentencias encerradas entre llaves un número determinado de veces. Cada vez que se ejecutan las instrucciones del bucle se vuelve a testear la condición. La declaración for tiene tres partes separadas por (;), vemos el ejemplo de su sintaxis:

```
for (inicialización; condición; expresión)
{
Instrucciones;
}
```

La inicialización de una variable local se produce una sola vez y la condición se testea cada vez que se termina la ejecución de las instrucciones dentro del bucle. Si la condición sigue cumpliéndose, las instrucciones del bucle se vuelven a ejecutar. Cuando la condición no se cumple, el bucle termina.

El siguiente ejemplo inicia el entero i en el 0, y la condición es probar que el valor es inferior a 20 y si es cierto i se incrementa en 1 y se vuelven a ejecutar las instrucciones que hay dentro de las llaves:

```
for (int i=0; i<20; i++) // declara i y prueba si es
{ // menor que 20, incrementa i.
digitalWrite(13, HIGH); // enciende el pin 13
delay(250); // espera ¼ seg.
digitalWrite(13, LOW); // apaga el pin 13
delay(250); // espera ¼ de seg.
}
```

Nota: El bucle en el lenguaje C es mucho más flexible que otros bucles encontrados en algunos otros lenguajes de programación, incluyendo BASIC. Cualquiera de los tres elementos de cabecera puede omitirse, aunque el punto y coma es obligatorio. También las declaraciones de inicialización, condición y expresión puede ser cualquier estamento valido en lenguaje C sin relación con las variables declaradas. Estos tipos de estados son raros pero permiten disponer soluciones a algunos problemas de programación raras.

7.18.4 while

Un bucle del tipo while es un bucle de ejecución continua mientras se cumpla la expresión colocada entre paréntesis en la cabecera del bucle. La variable de prueba tendrá que cambiar para salir del bucle. La situación podrá cambiar a expensas de una expresión dentro el código del bucle o también por el cambio de un valor en una entrada de un sensor.

```
while (unaVariable ?? valor)
{
ejecutarSentencias;
}
```

El siguiente ejemplo testea si la variable "unaVariable" es inferior a 200 y, si es verdad, ejecuta las declaraciones dentro de los corchetes y continuara ejecutando el bucle hasta que 'unaVariable' no sea inferior a 200.

```
While (unaVariable < 200) // testea si es menor que 200
{
instrucciones; // ejecuta las instrucciones
// entre llaves
unaVariable++; // incrementa la variable en 1
}
```

7.18.5 do... while

El bucle do while funciona de la misma manera que el bucle while, con la salvedad de que la condición se prueba al final del bucle, por lo que el bucle siempre se ejecutara al menos una vez.

```
do
{
Instrucciones;
} while (unaVariable ?? valor);
```

El siguiente ejemplo asigna el valor leído leeSensor() a la variable 'x', espera 50 milisegundos, y luego continúa mientras que el valor de la 'x' sea inferior a 100:

```
do
{
x = leeSensor();
delay(50);
} while (x < 100);
```

7.19 e/s digitales

7.19.1 pinMode(pin, mode)

Esta instrucción es utilizada en la parte de configuración setup () y sirve para configurar el modo de trabajo de un PIN pudiendo ser INPUT (entrada) u OUTPUT (salida).

```
pinMode(pin, OUTPUT); // configura 'pin' como salida
```

Los terminales de Arduino, por defecto, están configurados como entradas, por lo tanto no es necesario definirlos en el caso de que vayan a trabajar como entradas. Los pines configurados como entrada quedan, bajo el punto de vista eléctrico, como entradas en estado de alta impedancia. Estos pines tienen a nivel interno una resistencia de 20 KΩ a las que se puede acceder mediante software. Estas resistencias se acceden de la siguiente manera:

```
pinMode(pin, INPUT); // configura el 'pin' como
// entrada
digitalWrite(pin, HIGH); // activa las resistencias
// internas
```

Las resistencias internas normalmente se utilizan para conectar las entradas a interruptores. En el ejemplo anterior no se trata de convertir un pin en salida, es simplemente un método para activar las resistencias interiores. Los pines configurado como OUTPUT (salida) se dice que están en un estado de baja impedancia estado y pueden proporcionar 40 mA (miliamperios) de corriente a otros dispositivos y circuitos. Esta corriente es suficiente para alimentar un diodo LED (no olvidando poner una resistencia en serie), pero no es lo suficiente grande como para alimentar cargas de mayor consumo como relés, solenoides, o motores.

Un cortocircuito en las patillas Arduino provocara una corriente elevada que puede dañar o destruir el chip Atmega. A menudo es una buena idea conectar en la OUTUPT (salida) una resistencia externa de 470 o de 1000 Ω.

7.19.2 digitalRead(pin)

Lee el valor de un pin (definido como digital) dando un resultado HIGH (alto) o LOW (bajo). El pin se puede especificar ya sea como una variable o una constante (0-13).

```
valor = digitalRead(Pin); // hace que 'valor sea igual
// al estado leído en ´Pin´
```

7.19.3 digitalWrite(pin, value)

Envía al ´pin´ definido previamente como OUTPUT el valor HIGH o LOW (poniendo en 1 o 0 la salida). El pin se puede especificar ya sea como una variable o como una constante (0-13).

```
digitalWrite(pin, HIGH); // deposita en el 'pin' un
valor // HIGH (alto o 1)
```

El siguiente ejemplo lee el estado de un pulsador conectado a una entrada digital y lo escribe en el ´pin´de salida LED:

```
int led = 13; // asigna a LED el valor 13
int boton = 7; // asigna a botón el valor 7
int valor = 0; // define el valor y le asigna el
// valor 0
void setup()
{
pinMode(led, OUTPUT); // configura el led (pin13) como
salida
pinMode(boton, INPUT); // configura botón (pin7) como
entrada
}
void loop()
```

```
{
valor = digitalRead(boton); //lee el estado de la
// entrada botón
digitalWrite(led, valor); // envía a la salida ´led´el
} // valor leído
```

7.20 E/s analógicas

7.20.1 analogRead(pin)

Lee el valor de un determinado pin definido como entrada analógica con una resolución de 10 bits. Esta instrucción solo funciona en los pines (0-5). El rango de valor que podemos leer oscila de 0 a 1023.

```
valor = analogRead(pin); // asigna a valor lo que lee
// en la entrada ´pin'
```

Nota: Los pines analógicos (0-5) a diferencia de los pines digitales, no necesitan ser declarados como INPUT u OUPUT ya que son siempre Inputs.

7.20.2 analogWrite(pin, value)

Esta instrucción sirve para escribir un pseudo-valor analógico utilizando el procedimiento de modulación por ancho de pulso (PWM) a uno de los pines de Arduino marcados como "pin PWM". El más reciente Arduino, que implementa el chip ATmega168, permite habilitar como salidas analógicas tipo PWM los pines 3, 5, 6, 9, 10 y

11. Los modelos de Arduino más antiguos que implementan el chip ATmega8, solo tiene habilitadas para esta función los pines 9, 10 y 11. El valor que se puede enviar a estos pines de salida analógica puede darse en forma de variable o constante, pero siempre con un margen de 0-255.

```
analogWrite(pin, valor); // escribe 'valor' en el 'pin'
// definido como analógico
```

Si enviamos el valor 0 genera una salida de 0 voltios en el pin especificado; un valor de 255 genera una salida de 5 voltios de salida en el pin especificado. Para valores de entre 0 y 255, el pin saca tensiones entre 0 y 5 voltios - el valor HIGH de salida equivale a 5v (5 voltios). Teniendo en cuenta el concepto de señal PWM, por ejemplo, un valor de 64 equivaldrá a mantener 0 voltios de tres cuartas partes del tiempo y 5 voltios a una cuarta parte del tiempo; un valor de 128 equivaldrá a mantener la salida en 0 la mitad del tiempo y 5 voltios la otra mitad del tiempo, y un valor de 192 equivaldrá a mantener en la salida 0 voltios una cuarta parte del tiempo y de 5 voltios de tres cuartas partes del tiempo restante. Debido a que esta es una función de hardware, en el pin de salida analógica (PWN) se generara una onda constante después de ejecutada la instrucción analogWrite hasta que se llegue

a ejecutar otra instrucción analogWrite (o una llamada a digitalRead o digitalWrite en el mismo pin).

Nota: Las salidas analógicas a diferencia de las digitales, no necesitan ser declaradas como INPUT u OUTPUT..

El siguiente ejemplo lee un valor analógico de un pin de entrada analógica, convierte el valor dividiéndolo por 4, y envía el nuevo valor convertido a una salida del tipo PWM o salida analógica:

```
int led = 10; // define el pin 10 como ´led´
int analog = 0; // define el pin 0 como ´analog´
int valor; // define la variable ´valor´
void setup(){} // no es necesario configurar
// entradas y salidas
void loop()
{
valor = analogRead(analog); // lee el pin 0 y lo
asocia a //la variable valor
valor /= 4; //divide valor entre 4 y lo
//reasigna a valor
analogWrite(led, value); // escribe en el pin10 valor
}
```

7.21 Control del tiempo

7.21.1 delay(ms)

Detiene la ejecución del programa la cantidad de tiempo en ms que se indica en la propia instrucción. De tal manera que 1000 equivale a 1seg.

```
delay(1000); // espera 1 segundo
```

7.21.2 millis()

Devuelve el número de milisegundos transcurrido desde el inicio del programa en Arduino hasta el momento actual. Normalmente será un valor grande (dependiendo del tiempo que esté en marcha la aplicación después de cargada o después de la última vez que se pulso el botón "reset" de la tarjeta).

```
valor = millis(); // valor recoge el número de
// milisegundos
```

Nota: Este número se desbordara (si no se resetea de nuevo a cero), después de aproximadamente 9 horas.

7.22 Matemáticas

7.22.1 min(x, y)

Calcula el mínimo de dos números para cualquier tipo de datos devolviendo el número más pequeño.

```
valor = min(valor, 100); // asigna a valor el más
// pequeños de los dos
// números especificados.
```

Si 'valor' es menor que 100 valor recogerá su propio valor si ´valor´ es mayor que 100 valor pasara a valer 100.

7.22.2 max(x, y)

Calcula el máximo de dos números para cualquier tipo de datos devolviendo el número mayor de los dos.

```
valor = max(valor, 100); // asigna a valor el mayor de
// los dos números 'valor' y
// 100.
```

De esta manera nos aseguramos de que valor será como mínimo 100.

7.23 aleatorios

7.23.1 randomSeed(seed)

Establece un valor, o semilla, como punto de partida para la función random().

```
randomSeed(valor); // hace que valor sea la semilla del
// random
```

Debido a que Arduino es incapaz de crear un verdadero numero aleatorio, randomSeed le permite colocar una variable, constante, u otra función de control dentro de la función random, lo que permite generar números aleatorios "al azar". Hay una variedad de semillas, o funciones, que pueden ser utilizados en esta función, incluido millis () o incluso analogRead () que permite leer ruido eléctrico a través de un pin analógico.

7.23.2 random(max), random(min, max)

La función random devuelve un número aleatorio entero de un intervalo de valores especificado entre los valores min y max.

```
valor = random(100, 200); // asigna a la variable
// 'valor' un numero aleatorio
// comprendido entre 100-200
```

Nota: Use esta función después de usar el randomSeed().

El siguiente ejemplo genera un valor aleatorio entre 0-255 y lo envía a una salida analógica PWM :

```
int randNumber; // variable que almacena el valor
// aleatorio
int led = 10; // define led como 10
void setup() {} // no es necesario configurar nada
void loop()
{
randomSeed(millis()); // genera una semilla para
// aleatorio a partir
// de la función millis()
randNumber = random(255); // genera número aleatorio
// entre 0-255
analogWrite(led, randNumber); // envía a la salida
// led de tipo PWM el
// valor
delay(500); // espera 0,5 seg.
}
```

7.24 comunicación serie

7.24.1 Serial.begin(rate)

Abre el puerto serie y fija la velocidad en baudios para la transmisión de datos en serie. El valor típico de velocidad para comunicarse con el ordenador es 9600, aunque otras velocidades pueden ser soportadas.

```
void setup()
{
Serial.begin(9600); // abre el Puerto serie
} // configurando la velocidad en 9600 bps
```

Nota: Cuando se utiliza la comunicación serie los pines digital 0 (RX) y 1 (TX) no puede utilizarse al mismo tiempo.

7.24.2 Serial.println(data)

Imprime los datos en el puerto serie, seguido por un retorno de carro automático y salto de línea. Este comando toma la misma forma que Serial.print(), pero es mas fácil para la lectura de los datos en el Monitor Serie del software.

```
Serial.println(analogValue); // envía el valor
// 'analogValue' al
// puerto
```

Nota: Para obtener más información sobre las distintas posibilidades de Serial.println () y Serial.print () puede consultarse el sitio web de Arduino.

El siguiente ejemplo toma de una lectura analógica pin0 y envía estos datos al ordenador cada 1 segundo.

```
void setup()
{
Serial.begin(9600); // configura el puerto serie a
// 9600bps
}
void loop()
{
Serial.println(analogRead(0)); // envía valor analógico
delay(1000); // espera 1 segundo
}
```

7.24.3 Serial.print(data, data type)

Vuelca o envía un número o una cadena de caracteres, al puerto serie. Dicho comando puede tomar diferentes formas, dependiendo de los parámetros que utilicemos para definir el formato de volcado de los números.

Parámetros data: el número o la cadena de caracteres a volcar o enviar.

Data type: determina el formato de salida de los valores numéricos (decimal, octal, binario, etc...) DEC, OCT, BIN, HEX, BYTE , si no se pone nada vuelca ASCII

Ejemplos:

```
Serial.print(b) // Vuelca o envía el valor de b como
// un número decimal en caracteres
// ASCII.
int b = 79;
Serial.print(b); // imprime la cadena "79".
Serial.print(b, HEX); // Vuelca o envía el valor de
// b como un número hexadecimal
// en caracteres ASCII "4F".
Serial.print(b, OCT); // Vuelca o envía el valor de
// b como un número Octal en
// caracteres ASCII "117".
Serial.print(b, BIN) // Vuelca o envía el valor de
// b como un número binario en
// caracteres ASCII "1001111".
Serial.print(b, BYTE); // Devuelve el carácter "O",
// el cual representa el
// carácter ASCII del valor
// 79. (Ver tabla ASCII).
Serial.print(str); //Vuelca o envía la cadena de
// caracteres como una cadena ASCII.
Serial.print("Hello World!"); // vuelca "Hello World!".
```

7.24.4 Serial.avaible()

Devuelve Un entero con el número de bytes disponibles para leer desde el buffer serie, o 0 si no hay ninguno. Si hay algún dato disponible, SerialAvailable() será mayor que 0. El buffer serie puede almacenar como máximo 64 bytes.

```
int Serial.available() // Obtiene un número entero
// con el número de bytes
// (caracteres) disponibles
// para leer o capturar desde
// el puerto serie
```

Ejemplo

```
int incomingByte = 0; // almacena el dato serie
void setup() {
Serial.begin(9600); // abre el puerto serie, y le asigna
// la velocidad de 9600 bps
}
void loop() {
if (Serial.available() > 0) // envía datos sólo si
{ // los recibe:
incomingByte = Serial.read();// lee el byte de entrada:
//lo vuelca a pantalla
Serial.print("I received: ");
Serial.println(incomingByte, DEC);
}
}
```

7.24.5 Serial.Read()

Lee o captura un byte (un carácter) desde el puerto serie. Devuelve el siguiente byte (carácter) desde el puerto serie, o -1 si no hay ninguno.

Ejemplo:

```
int incomingByte = 0; // almacenar el dato serie
void setup() {
Serial.begin(9600); // abre el puerto serie, y le asigna
// la velocidad de 9600 bps
}
void loop() {
if (Serial.available() > 0)// envía datos sólo si los
{ // recibe
incomingByte = Serial.read(); // lee el byte de
// entrada y lo vuelca
Serial.print("I received: "); // a pantalla
Serial.println(incomingByte, DEC);
}
}
```