http://lacienciaparatodos.wordpress.com

PRÁCTICAS ROBÓTICA

ARDUINO

+

SHIELD MULTIFUNCIÓN

VISUALINO y CÓDIGO

Javier Fernández Panadero
Javierfpanadero@yahoo.com

ÍNDICE

Contenido

Javier Fernández Panadero
Javierfpanadero@yahoo.com

0. Objetivo

- Prácticas de **robótica desde cero**.
- Usaremos un **Shield multifunción** para tener la electrónica ya operativa y **centrarnos en la programación.**
- También pueden hacerse **sin el Shield**, simplemente cableando circuitos para pulsadores, LEDs, etc.
- Usaremos **Visualino** que es un lenguaje de bloques para simplificar la programación. Tiene la ventaja de **mostrar el código Arduino** al lado, de forma que podemos familiarizarnos para quien quiera profundizar
- Se han separado las propuestas de las soluciones para que sea más sencillo su uso con estudiantes.

A. ¿De qué hablamos aquí?

La robótica NO es sólo una automatización (como podría ser la cisterna del váter)

TRES ELEMENTOS CLAVE

a) Sensores

Nos darán **información del entorno** (luz-oscuridad, humedad, distancias, etc.) o del usuario (toca botones, escribe en el teclado, etc.).

b) Actuadores

Será **la forma en la que intervengamos en el mundo** (encendiendo luces, produciendo sonidos, enviando información al ordenador, moviendo motores, etc.)

c) Programación

Aquí es donde tenemos todo el poder. Con la información de los sensores **haremos que los actuadores se comporten como deseemos**, sin ninguna restricción más que nuestra imaginación.

B. Si no queréis usar Shield…

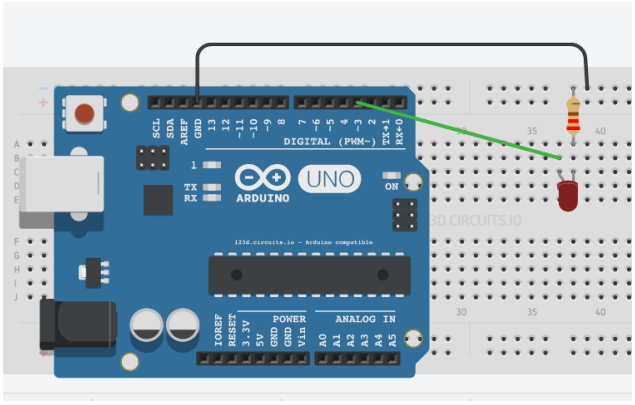
Tendréis que hacer pequeños circuitos para los LEDs y los pulsadores.

1. LEDs

Tenéis dos opciones (R=330Ω)

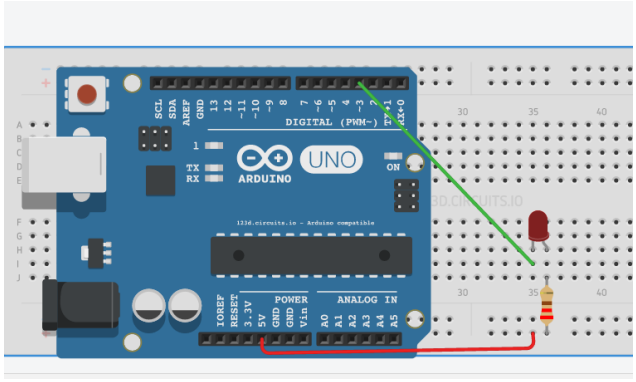
a) Lógica directa

Al dar una salida ALTA por el pin 3 de Arduino, el LED se encenderá.



b) Lógica inversa

Al dar salida BAJA por el pin 3 de Arduino, el LED se encenderá.

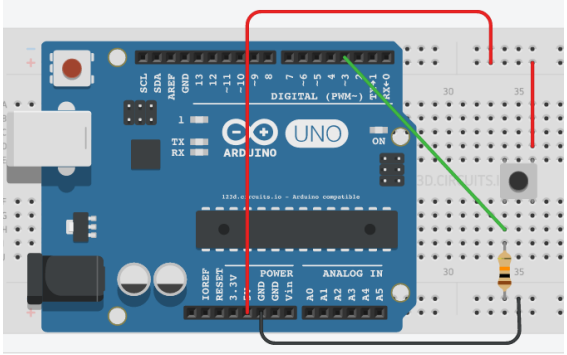


2. PULSADORES

Tenéis dos opciones (R=10kΩ)

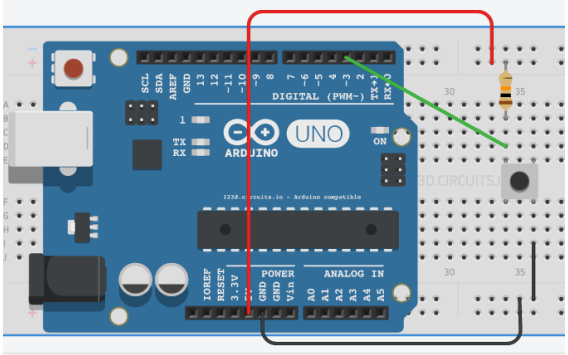
a) Lógica directa

Al accionar el pulsador llegará a Arduino una entrada ALTA (PULL DOWN)



b) Lógica inversa

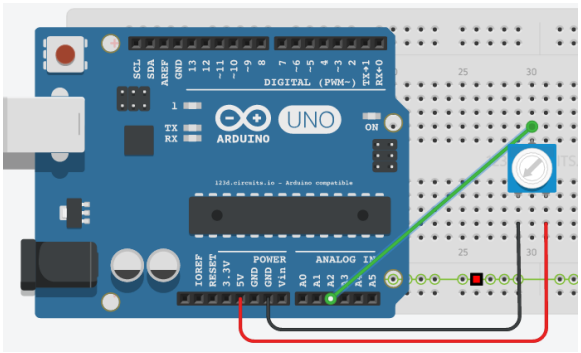
Al accionar el pulsador llegará a Arduino una entrada BAJA. (PULL UP)



NOTA: Configurando pinMode(n,INPUT_PULLUP) conecta una resistencia interna de (20K) conectada a +5V y solo hay que poner el pulsador a masa y al pin n

3. POTENCIÓMETRO

El potenciómetro nos sirve para dar una entrada de tensión variable entre 0V y 5V, para eso pondremos los terminales extremos a GND y 5V y el terminar central al pin analógico de entrada que queramos.



También se puede interpretar como la señal que daría un sensor.

4. ZUMBADOR

Dependerá del tipo que tengas, pero tanto el del Shield como el del kit de Arduino admiten la función tone() para producir sonidos de distinta duración y frecuencia.

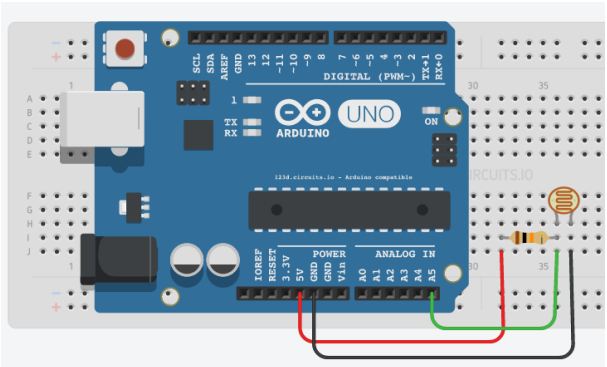
5. DISPLAY

En las prácticas usaremos los cuatro display de siete segmentos para dar pequeñas salidas de texto, valores de variables, etc.

Según vuestra disponibilidad podéis usar otros displays o, simplemente, el Monitor Serie.

6. OTROS COMPONENTES

Para otros componentes (LDRs, sensores de temperatura, servos, etc.) usaremos una placa protoboard y tendremos que hacer conexiones para las que usaremos cables hembra-macho, ya que los pins libres que nos quedan en el Shield son machos y por el otro lado iremos contra la placa. Por ejemplo, este sería un **típico divisor de tensión para leer una LDR.**



Javier Fernández Panadero
Javierfpanadero@yahoo.com

C. SHIELD MULTIFUNCIÓN

Se conecta sobre la placa Arduino, "tapándola" (hay otros muchos Shields con funciones distintas (WIFI, Relés, etc.).

En nuestro caso lo que queremos es tener entradas y salidas "cableadas gratis" para poder centrarnos en la programación.

ENTRADAS

1. **PULSADORES**

Tres pulsadores conectados a los pines analógicos A1, A2, A3 (en lógica INVERSA)

2. **POTENCIÓMETRO**

Conectado al pin analógico A0

SALIDAS

1. **LEDs**

Cuatro LEDs en los pines 10, 11, 12, 13. Los dos últimos NO admiten salida PMW ("analógica") en Arduino UNO. Conectados los cuatro en lógica INVERSA.

2. **ZUMBADOR**

Conectado al pin 3. Admite la función tone() y emite el sonido en "background", lo que quiere decir que sigue ejecutando instrucciones. Cuidado con mezclar tonos si no esperas.

3. **DISPLAY**

Constituido por cuatro "números" que podremos escribir con sencillez gracias a la función MFS.write() incluida en la librería del Shield y que admite texto, números y variables.

4. **PINES DISPONIBLES**

Quedan disponibles los pines A5, 9, 6 y 5 que usaremos para conectar otros sensores o actuadores que nos interesen. El primero nos permite entradas analógicas y 5, 6, y 9 admiten salidas PMW.

5. **OTROS**

Hay conexiones preparadas para otros sensores que no usaremos aquí. Para más detalle podéis consultar en la siguiente referencia: https://www.mpja.com/download/hackatronics-arduino-multi-function-shield.pdf

LIBRERÍA

Una **librería es un programa que se puede añadir** (#include) a los programas que queramos hacer.

La ventaja de utilizarlas es que dentro de ese código hay variables y, sobre todo, **funciones que podemos usar y nos evitamos programarlas.**

La librería podéis descargarla en este enlace, ya explicaremos cómo se incluye. http://files.cohesivecomputing.co.uk/MultiFuncShield-Library.zip Necesitas una versión reciente del IDE de Arduino (entorno de programación) para que pueda cargarse.

Hay otra librería extra que debes descargarte **TimerOne:**
https://code.google.com/archive/p/arduino-timerone/downloads

Esta librería puede usarse también para programar interrupciones (usuarios avanzados)

Como en estas prácticas queremos aprender a programar Arduino, no nos interesa mucho aprovecharnos de las funciones del Shield, pero sí usaremos una muy sencilla y que nos permitirá la utilización del display:

MFS.Write()

Como argumento pondremos **texto (entre comillas), números o variables.** En este último caso lo que se escribirá es el valor que tiene la variable.

Si queréis usar **el resto de funciones** podéis encontrarlas explicadas en la páginas 27, 28 y 29 del documento que ya hablamos https://www.mpja.com/download/hackatronics-arduino-multi-function-shield.pdf

CÓMO CONSEGUIR EL SHIELD

Lo podéis encontrar a muy bajo precio en muchas páginas, pero para los que necesitéis un proveedor que os mande factura, he encontrado precios razonables y muy buena disposición en estas dos tiendas:

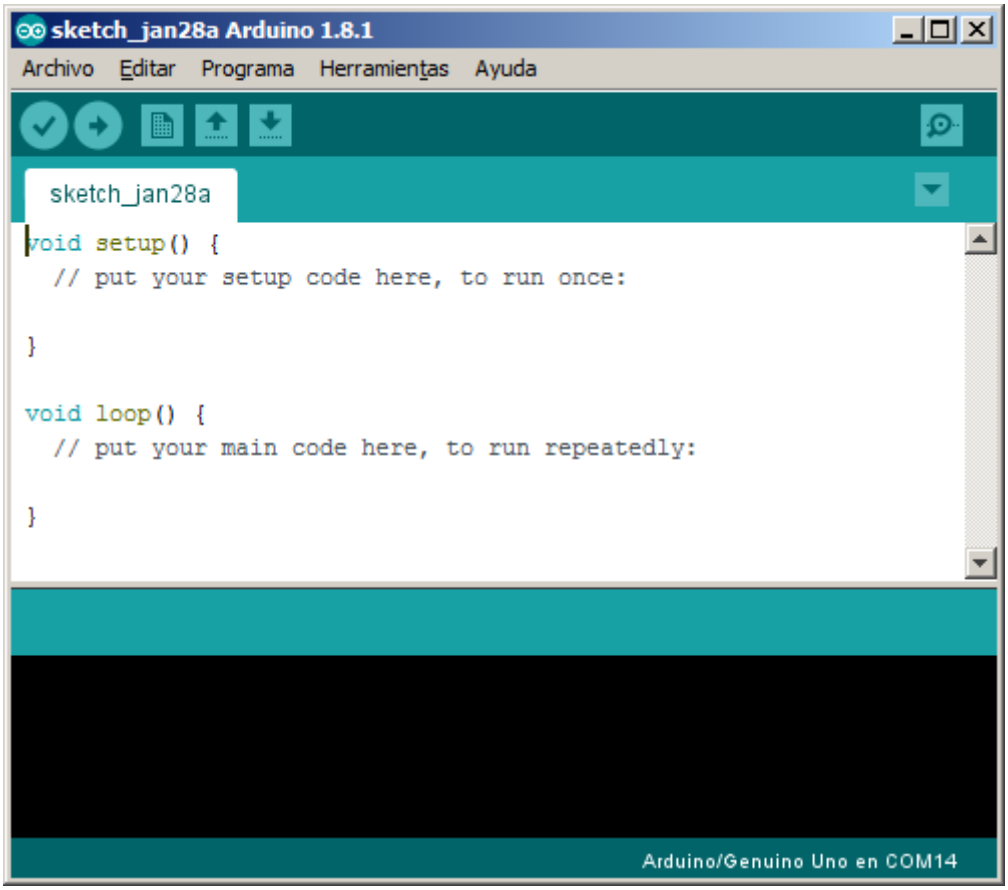
http://www.e-ika.com/shield-multifuncional-de-aprendizaje

https://www.electrohobby.es/es/shield/219-shield-multifuncion.html

¡¡HAY OTROS!!

Este no es el único entrenador disponible. Mira este qué exageración…

IDE ARDUINO

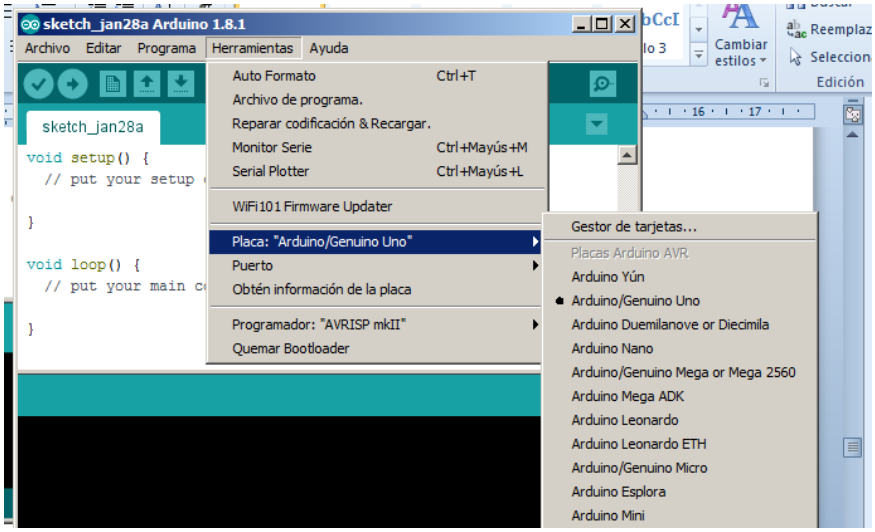


El IDE de Arduino es el entorno de programación donde escribiremos el código y desde donde podemos cargar los programas a la placa.

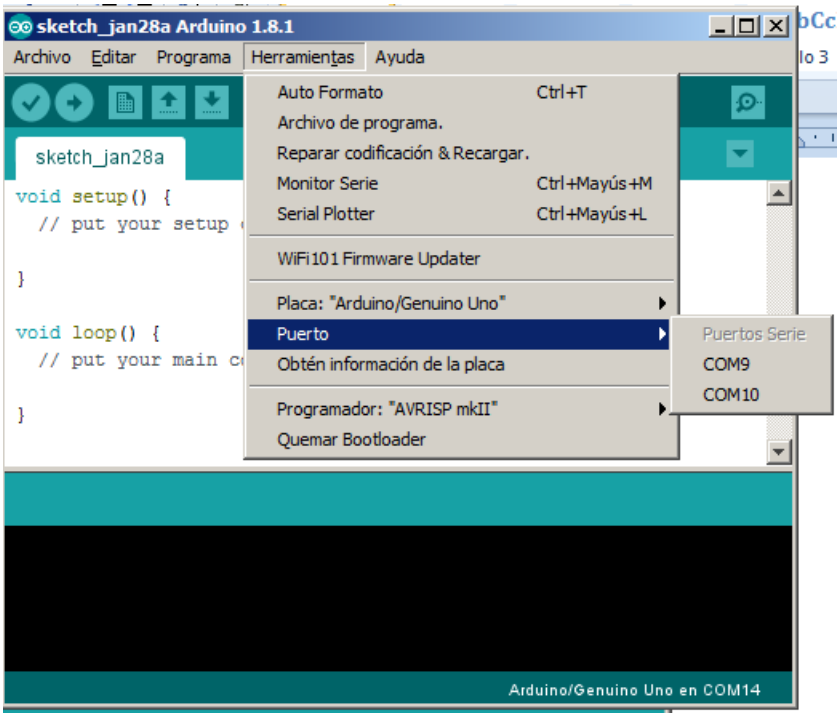
También podemos enviar y recibir datos de la placa mediante la herramienta Monitor Serie.

Cuando conectamos la placa al PC por su cable USB, esta se instala automáticamente, pero hay que “decirle” al IDE un par de cosas.

Lo primero, qué tipo de tarjeta, de toda la familia Arduino, hemos conectado.

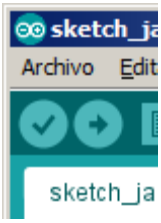


Después en qué puerto está conectada. La forma más fácil de saberlo es enchufar y desenchufar la placa y ver qué COM aparece. Con estas dos cosas la placa ya está en contacto con el PC. (Podría ser que también hubiera que elegir en ese mismo menú qué programador usa tu placa).



Lo siguiente sería escribir un programa y subirlo a la placa. Nosotros usaremos una herramienta visual, programaremos con bloques, la herramienta generará el código Arduino y lo copiaremos aquí.

El siguiente paso sería VERIFICAR que el código está correctamente escrito.



Después, SUBIRLO a la placa.

Esto se hace con los dos botones que hay arriba a la izquierda.

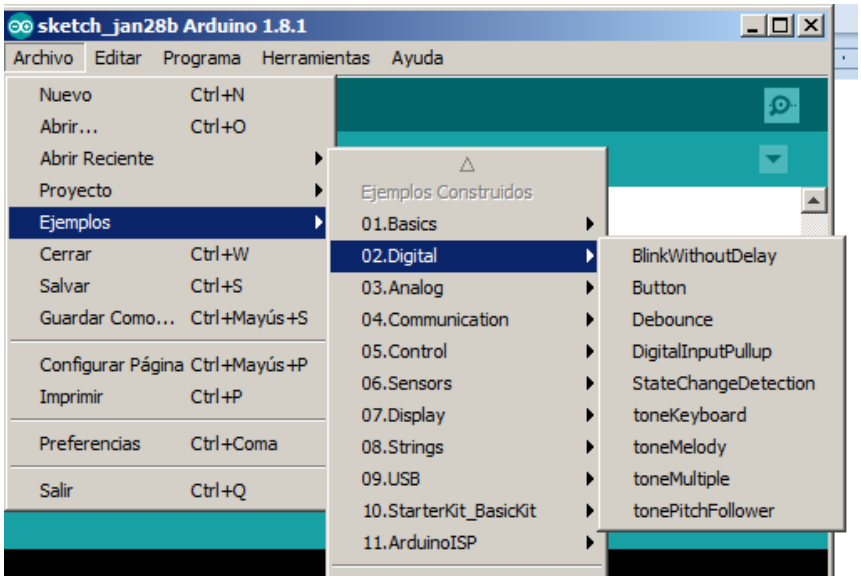
Dos cosas:

- Siempre que se sube se hace una verificación previa
- **Que la verificación de OK, no implica que el programa hace justo lo que tú quieres**, sino sólo que está correctamente escrito.

Cuando subes el programa parpadean unos LEDs en la placa y acto seguido comienza a ejecutarse automáticamente.

Si quieres que el programa se reinicie acciona el pulsador RESET de Arduino o del SHIELD.

El IDE ya tiene unos **ejemplos** que puedes ejecutar directamente



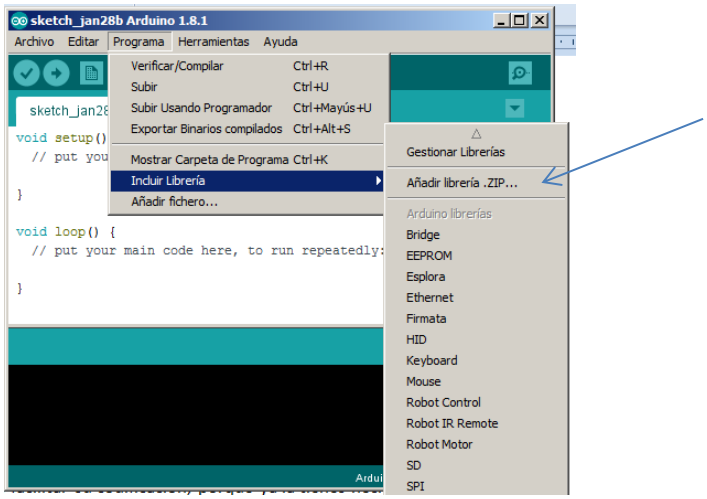
Mirar estos ejemplos es una estupenda forma de entender el código (están muy comentados).

También te aparecerán **ejemplos de las librerías que vayas instalando**, con lo que también será más fácil entenderlas.

INSTALAR LIBRERÍAS

Las librerías son pequeños programas donde se definen funciones y variables que pueden ser incluidas en tus programas para facilitar su codificación, porque ya están listas para usar.

Para **incluir librerías** en tu programa o para **instalar librerías** (Añadir librería .ZIP) que te descargues de Internet tienes que ir a este menú.



Nosotros tenemos que descargar e instalar **las librerías MFS y TimerONE**

http://files.cohesivecomputing.co.uk/MultiFuncShield-Library.zip

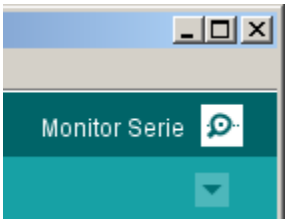
https://code.google.com/archive/p/arduino-timerone/downloads

MONITOR SERIE

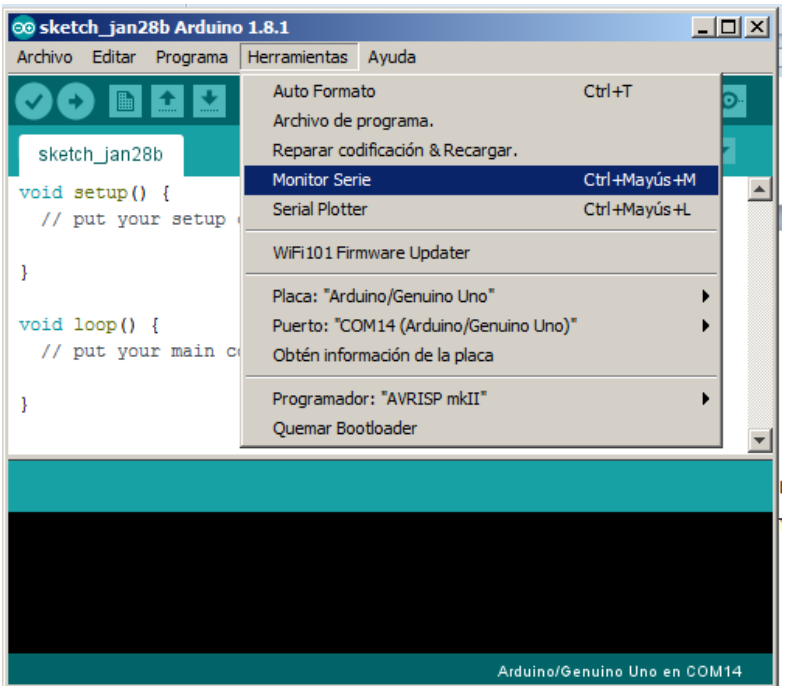
PUERTO SERIE

El puerto serie es una forma de comunicar Arduino con otros dispositivos, para enviar o recibir datos. El envío se lo haremos típicamente por teclado, aunque podría hacerse vía Bluetooth (mediante una app de terminal en un móvil, p.ej.) o a través de una WIFI.

Para acceder al monitor lo podemos hacer con el botón de la parte superior derecha



O por menú. Ahí también aparece el Serial Plotter que nos hará una representación gráfica en tiempo real si vamos mandando datos desde la placa



NOTA: Si quieres anular algunas instrucciones para probar cosas en tus programas, usa la función COMENTAR/DESCOMENTAR CÓDIGO en el menú contextual (botón derecho). Funciona con una o varias líneas.

VISUALINO

Visualino es un entorno gráfico para programar Arduino.

Podéis descargarlo, obtener documentación y mucho más aquí:

http://www.visualino.net/index.es.html

Es sencillo de usar, está basado en Blockly y puede usarse en español (muy similar a Scratch)

Muestra dos ventanas, en una se van colocando los bloques y en la otra se va generando código Arduino, de forma que nos vamos habituando al código y nos resultará más sencillo editarlo cuando sea necesario.

Según versiones y placas, **al intentar cargar el programa a la placa** (desde Visualino) puede que se haga directamente y empiece a funcionar o que se nos abra el IDE de Arduino y, en ese caso, procederemos a hacerlo desde allí.

A veces tenemos que editar el código (en texto), p.ej. para incluir la librería del SHIELD y añadir las instrucciones que llevarán sus funciones, podría hacerse en Visualino

Hay **muchos otros entornos,** bitblock por ejemplo (ideal si se usa BQ), Ardublock, etc.

A mí me gusta particularmente **S4A (Scratch for Arduino)**

Ventajas

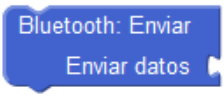
- Pueden verse los valores de las entradas, salidas y variables de forma muy sencilla sin tener que codificar la salida por el Monitor Serie u otro sistema.
- Puede programarse por eventos y tener código ejecutándose de forma simultánea (pseudoparalelismo), de forma que podemos tener un hilo que nos "vigile" un sensor, a la vez que otro hilo mueve unos motores, sin necesidad de integrar en una sola secuencia ambos conceptos.
- Puede hacerse programación orientada a objetos también de manera sencilla y clara.
- Muchos estudiantes lo conocen, así que el aprendizaje es más rápido.

También tiene desventajas

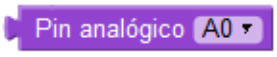
- Debe estar siempre conectado al PC (Es interpretado. A través de un programa que se carga en Arduino se interpretan las órdenes que llegan del PC).
- Tiene asignados los pines a tareas concretas obligatoriamente, salidas, entradas, analógicas y digitales. Por eso tampoco es compatible con esta Shield.
- No genera código Arduino.

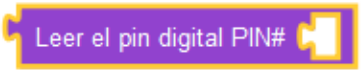
La forma de los bloques es importante y Visualino no te dejará ponerlos donde “no debes”.

Estos bloques son **ACCIONES** y puedes ponerse directamente en **REPETIR** (LOOP) o unos a continuación de otros



Estos otros bloques son **VALORES** y deben ponerse allí donde haya un “hueco” para su “punta”





PRÁCTICAS

Programación Actuadores

LEDs

1. Encender un LED

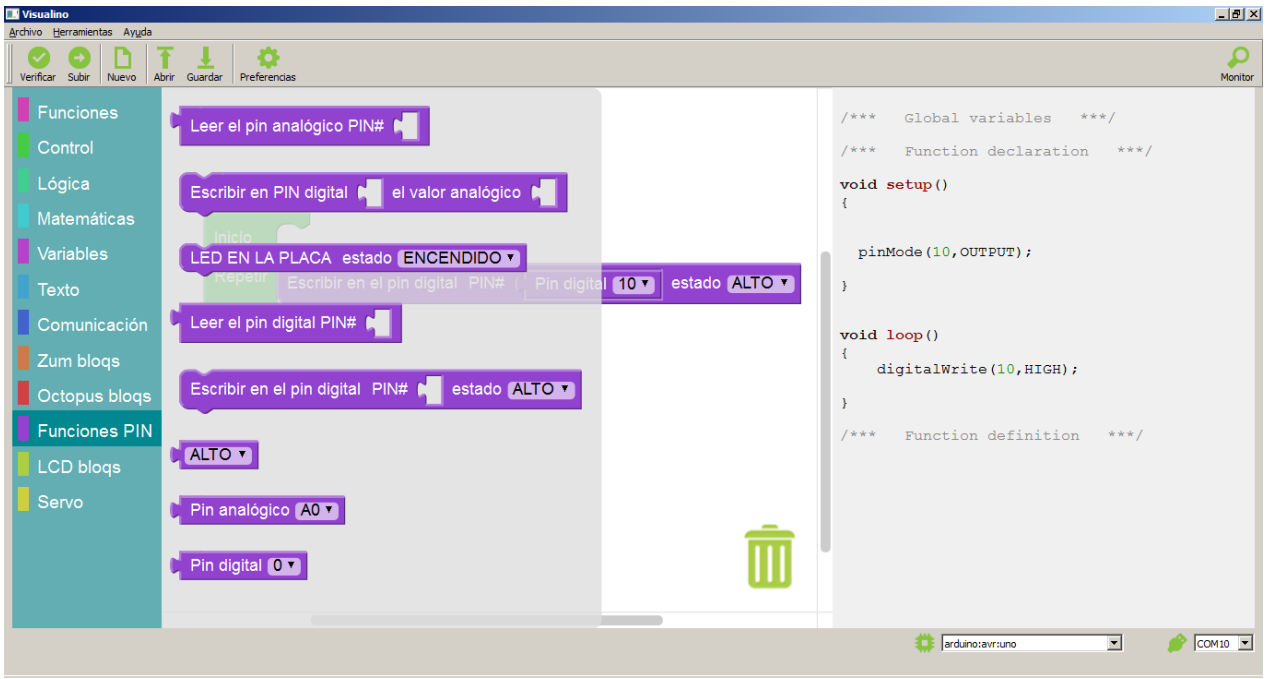
¡Nuestro primer programa!

Como se puede ver hay dos lugares donde podemos poner bloques.

Si queremos configurar algo o ejecutar una instrucción sólo una vez lo ponemos en INICIO

Si queremos que se repita una y otra vez la ejecución lo ponemos en REPETIR

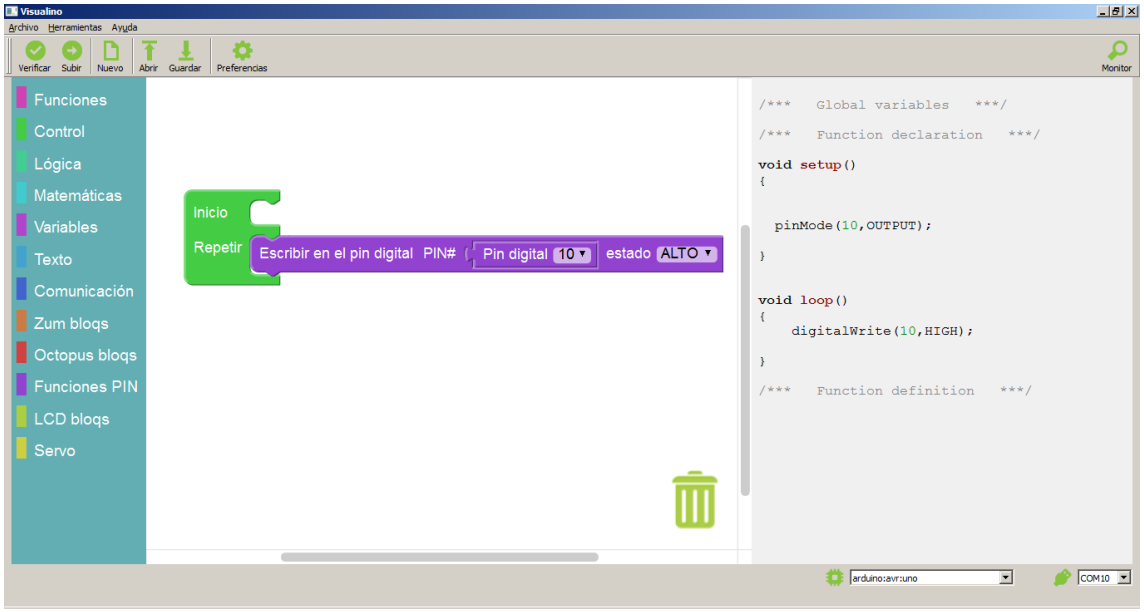
Encendamos uno de los LEDs (pines 10 al 13)



En el menú FUNCIONES PIN encontraremos los bloques necesarios

ESCRIBIR EN PIN DIGITAL

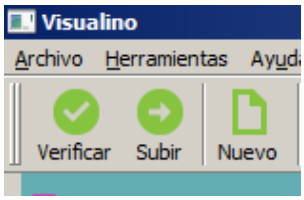
PIN DIGITAL



MUY IMPORTANTE: OBSERVA CÓMO VA APARECIENDO EL CÓDIGO EN LA PARTE DERECHA AL PONER LOS BLOQUES Y VERÁS SU SIGNIFICADO

Por ejemplo, se ha configurado el pin 10 como OUTPUT, ya que lo usamos como salida, si lo usásemos como entrada hubiera puesto INPUT.

Pulsa **VERIFICAR** para comprobar si tu código está correctamente escrito y **SUBIR** para cargarlo a la placa. Siempre que lo subas se hará una verificación previa.



RECUERDA: Correctamente escrito significa solamente que usas el lenguaje correctamente, no que el programa vaya a hacer lo que tú quieres. Por ejemplo, si dices “Dame el azúcar” estará escrito en perfecto español, pero si lo que querías era la sal, no funcionará como esperas.

Aunque Visualino puede configurarse para que suba directamente a la placa el programa, también es posible que se abra el IDE de Arduino y debas cargarlo desde allí.

PUEDE QUE EL LED NO SE ENCIENDA…

¿Es nuestra configuración de lógica directa o inversa? ¿Se encenderá cuando demos una salida ALTA o BAJA?

Prueba con ambos programas y lo verás. Ya dijimos que en SHIELD, para los LEDs, es INVERSA

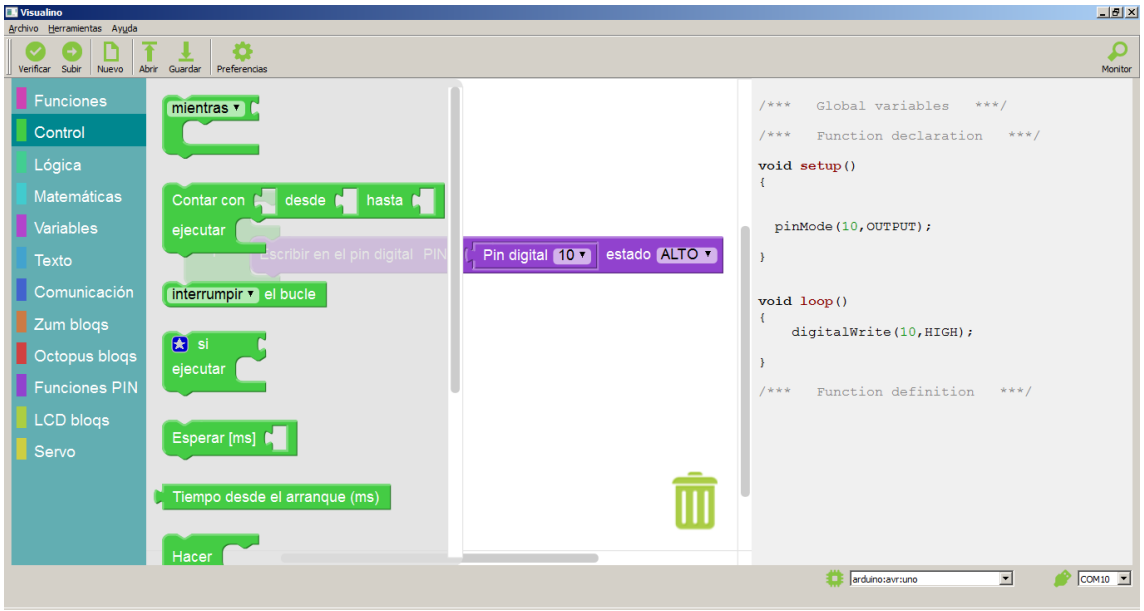
2. Encender un LED y apagarlo

Ya vimos en la práctica anterior cómo se encendía o se apagaba un LED

Ahora sólo necesitamos que esté un tiempo en cada estado.

Para esto usamos un bloque que tienes en el menú CONTROL.

Es el bloque ESPERAR (ms)



Deja que el LED esté medio segundo (500 ms) en cada estado.

FÍJATE EN QUE:

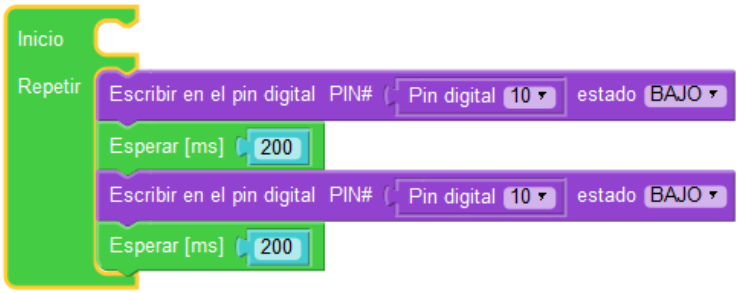
- Si no lo cambias tú, el estado del LED no cambiará (no se apaga solo).

- Las instrucciones en el bucle se repiten indefinidamente

Busca una manera de hacerlo.

Los valores numéricos están en el menú MATEMÁTICAS

SOLUCIÓN 2



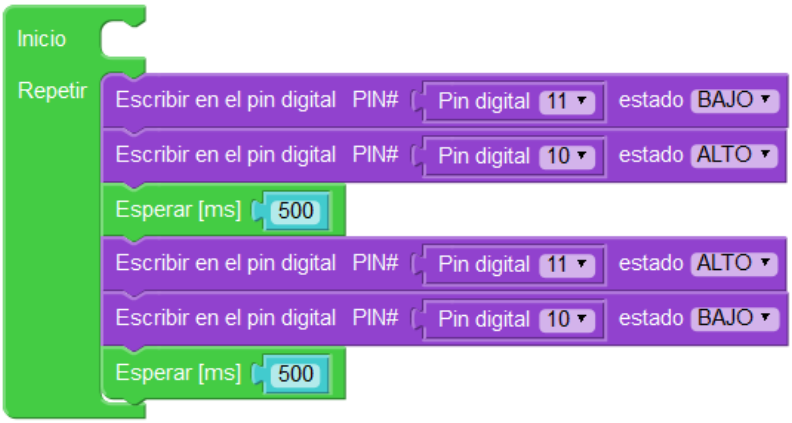

```
/***   Global variables   ***/

/***   Function declaration   ***/

void setup()
{
  pinMode(10,OUTPUT); // Configura el pin 10 como salida
}
void loop()
{
    digitalWrite(10,LOW);   // Enciende el LED (lógica inversa)
    delay(500);             // Espera durante medio segundo
    digitalWrite(10,HIGH);  // Apaga el LED (lógica inversa)
    delay(500);
}

/***   Function definition   ***/
```

3. Encender dos LED alternativamente

El tiempo que tarda en ejecutar una orden la placa es muy pequeño, así que instrucciones consecutivas nos parecerán simultáneas, por ejemplo, esto encenderá los dos LEDs “a la vez”.

```
digitalWrite(10,LOW);
digitalWrite(11,LOW);
```

SOLUCIÓN 3




```
/***   Global variables   ***/

/***   Function declaration   ***/

void setup()
{
  pinMode(11,OUTPUT);
  pinMode(10,OUTPUT);
}

void loop()
{
    digitalWrite(11,LOW);
    digitalWrite(10,HIGH);
    delay(500);
    digitalWrite(11,HIGH);
    digitalWrite(10,LOW);
    delay(500);

}

/***   Function definition   ***/
```

4. Coche fantástico

- NUESTRA PRIMERA VARIABLE
- NUESTRO PRIMER BUCLE (FOR)

Se trata de hacer un barrido de luces como el del famoso Coche Fantástico.

Vamos encendiéndolos hasta quedar todos encendidos, y después los apagamos uno a uno hasta que queden todos apagados.

Recuerda que instrucciones consecutivas son aparentemente simultáneas (en tiempo real)

Hazlo primero apagando y encendiendo los LEDs con una instrucción cada uno.

Ahora lo intentaremos hacer con un BUCLE:

Se trataría de decir, Enciende un led espera, apágalo, enciende el siguiente, etc.

Pero, ¿cómo indicamos, un led, el siguiente, otro más…?

Necesitamos una **VARIABLE**.

Una variable es como una caja donde voy a guardar un dato.

Hay que distinguir tres cosas

1. Nombre de la variable. P.ej.: EDAD
2. Valor que toma. P.ej.: 15
3. Tipo de datos que guarda: P.ej.: int (enteros)

En nuestro caso

Pensaremos en el LED 10, LED11, LED12 LED13

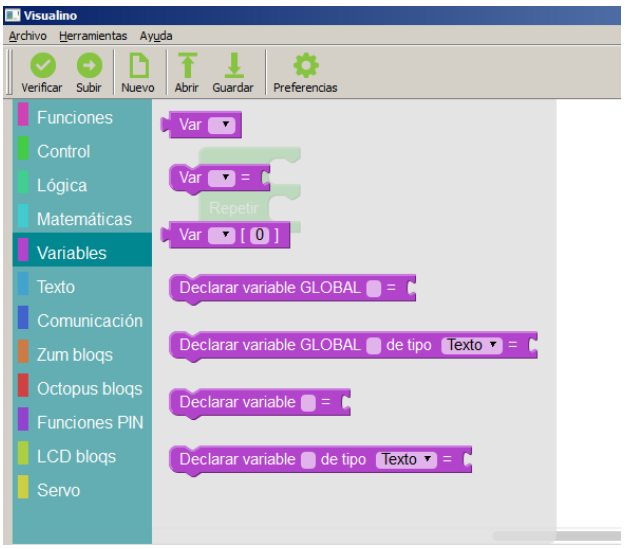
Que es lo mismo que decir LED i, y que la variable i puede tomar los valores del 10 al 13.

En el módulo CONTROL tienes el bloque adecuado.

En el primer hueco hay que poner una variable, en el segundo su valor inicial en el último su valor final.

Para eso hay que declarar la variable en el SETUP (INICIO), darle un nombre.

En su menú tienes todo lo relacionado con las variables.



Haz ahora otra variante en la que “una sola luz” va “moviéndose de izquierda a derecha. Eso implicará encender un LED y apagar el anterior, pero podemos hacer una cosa en una iteración y otra en la siguiente del bucle… A ver qué se te ocurre.

SOLUCIÓN 4-1


```
/***   Global variables   ***/
int i=0; // Así podemos usar esta variable en todo el código
/***   Function declaration   ***/

void setup()
{
       //Lo normal es definir aquí si los pines son de
       //entrada o salida, pero también se puede hacer
       //más adelante, siempre antes de usarlos.
       //Lo ponemos así, xq así lo hace Arduino
}
void loop()
{
    for (i = 10; i <= 13; i++) {
      pinMode(i,OUTPUT);    //ejecuta para cada valor de i
      digitalWrite(i,LOW);  //incluyendo 10 y 13
      delay(100);    // en este bucle los enciende
    }
    for (i = 13; i >= 10; i--) {
      pinMode(i,OUTPUT);
      digitalWrite(i,HIGH); //en este bucle los apaga
      delay(100);
    }

}

/***   Function definition   ***/
```

SOLUCIÓN 4-2


```
/***   Global variables   ***/
int i=0;

/***   Function declaration   ***/

void setup()
{

}


void loop()
{
    for (i = 10; i <= 13; i++) {
      pinMode(i,OUTPUT);
      digitalWrite(i,LOW);
      delay(100);
      pinMode(i,OUTPUT);
      digitalWrite(i,HIGH);
    }
    for (i = 12; i >= 10; i--) {
      pinMode(i,OUTPUT);
      digitalWrite(i,LOW);
      delay(100);
      pinMode(i,OUTPUT);
      digitalWrite(i,HIGH);
    }

}

/***   Function definition   ***/
```

Así es como lo escribe Visualino, pero es más sencillo y se entiende mejor si ponemos la configuración de los pines en el SETUP (INICIO). Fíjate que también los podemos configurar usando un bucle

```
/***   Global variables   ***/
int i=0;

/***   Function declaration   ***/

void setup()
{
   for (i = 10; i <= 13; i++) {
      pinMode(i,OUTPUT);    // usamos un bucle para configurar
   }
}

void loop()
{

/*Como ponemos el retardo en el medio las acciones que ocurren
"seguidas" son el apagado de un LED y el encendido del siguiente
al principio de la siguiente iteración*/

    for (i = 10; i <= 13; i++) {
      digitalWrite(i,LOW);
      delay(100);
      digitalWrite(i,HIGH);
    }
    for (i = 12; i >= 10; i--) {
      digitalWrite(i,LOW);
      delay(100);
      digitalWrite(i,HIGH);
    }

}

/***   Function definition   ***/
```

5. Encender varios LEDs a la vez

- MI PRIMERA FUNCIÓN

Ya hemos visto una manera de "resumir" o "encapsular" un conjunto de funciones con el bucle.

Ahora vamos a ver otra, las FUNCIONES

Una función es como un "conjuro". Tú lo llamas y pasa algo. Ocurre una acción.

Veamos ejemplos.

- Una función que encienda todos los LEDs. **encender**
- Una función que encienda tantos LEDs como le digas. **encenderLEDS**
- Una función que te dé el número pi. **dimePi**
- Una función que te devuelva la suma de dos números que tú le digas. **suma**

Si te fijas, no son todas iguales, unas pueden llamarse tal cual, harán una acción, pero no devuelven ningún valor. Por ejemplo: encender.

Otras necesitan que tú les des valores (argumentos) para poder funcionar. Por ejemplo: encenderLEDS y suma.

Otras devuelven valores. Por ejemplo: dimePi y suma.

Sabemos que algo es una función porque después de su nombre aparecen paréntesis.

Así debemos escribir: encender(), encenderLEDS(), dimePi() ,suma().

Dentro del paréntesis hay que poner los argumentos que necesita el "conjuro" para poder ejecutarse: encender(), encenderLEDS(n), dimePi(), suma(a,b) y así es como las "invocaremos".

Pero para definir las funciones también debemos poner delante si la función nos va a devolver un valor y de qué tipo es (si es un número entero, real (float), un carácter (char), un conjunto de caracteres (string), etc. Si la función no devuelve nada se pone void.

Por lo tanto, al definir nuestras funciones tendríamos que escribir

void encender() void encenderLEDS(n)

float dimePi() int suma(a,b)

Entendiendo que pi es un número real (float) y que la suma será entre enteros (int).

Esto simplifica mucho los programas porque los "atomiza" en pequeñas tareas y el programa principal se hace llamando a esas funciones.

También resultará más fácil de entender, si elegimos bien el nombre de las funciones.

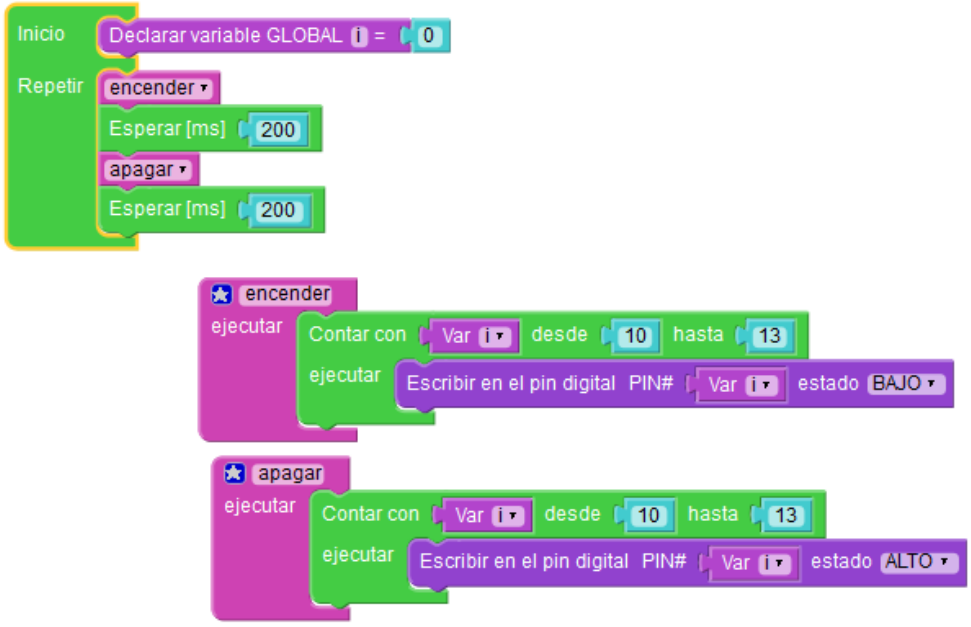
En el código Arduino las funciones pueden escribirse al principio o al final (fuera del SETUP y del LOOP) y el programa irá a "buscarlas" cuando sean invocadas.

Podemos definir variables dentro de las funciones para que sean usadas solamente allí, en lugar de declararlas al principio del todo como habíamos hecho. Esta es la distinción entre VARIABLES LOCALES Y GLOBALES.

Vamos con el ejercicio

Escribe una función que, al invocarla, encienda todos los LEDs y otra que apague todos los LEDs e invócalas en el LOOP para que parpadeen. Simplifica aún más utilizando bucles.

SOLUCIÓN 5




```
/***   Global variables   ***/
int i=0;

/***   Function declaration   ***/
void encender (); //anuncia las funciones q usará
void apagar ();

void setup()  // ya te darás cuenta que, tanto
{             // SETUP como LOOP son funciones

}

void loop()
{
    encender();       //llamamos a la función
    delay(200);
    apagar();         //llamamos a la función
    delay(200);
}

/***   Function definition   ***/
void encender () {                  //qué hace la función
  for (i = 10; i <= 13; i++) {
    pinMode(i,OUTPUT);              //esto podría ir en SETUP
    digitalWrite(i,LOW);
  }
 }
void apagar () {
  for (i = 10; i <= 13; i++) {
    pinMode(i,OUTPUT);
    digitalWrite(i,HIGH);
  }
 }
```

6. Escribir en el DISPLAY

Querríamos escribir en el display del SHIELD.

Si tuviéramos acceso a los segmentos del display podríamos ir encendiendo los que quisiéramos para hacer números, letras, figuras, etc., pero el acceso está regulado por unos integrados y por un buen follón.

Tenemos la suerte de que alguien ha programado las funciones necesarias, ahorrándonos ese trabajo. Sólo tenemos que saber cómo se llaman, si necesitan argumentos y si devuelven algún valor para poder invocarlas.

La función que vamos a usar puede ser muy compleja en su interior, pero es muy sencilla en su uso, sólo hay que escribir:

MFS.write();

Dentro del paréntesis pondremos un texto entre comillas (“hola”), números (4), o el nombre de una variable para que escriba su valor: MFS.write(x); escribiría el valor de x, no la x. para escribir x habría que decir MFS.write(“x”);

No podemos invocar una función sin cargar antes la librería que la contiene.

Aunque Visualino nos permite incluir ciertas librerías estándar, no la de nuestro SHIELD.

Por lo tanto, **siempre que usemos el display** en un programa **tendremos que retocar el código** en el IDE para poder incluir la librería y los MFS.write() que queramos, donde los queramos.

O usar este bloque para incluir líneas de código directamente: 

Prueba este ejemplo:

```
#include <TimerOne.h>
#include <Wire.h>
#include <MultiFuncShield.h>

void setup() {
 Timer1.initialize();
 MFS.initialize(&Timer1); // initialize multi-function shield library

 MFS.write("Hi");
}

void loop() {
}
```

Como veis hay dos librerías más además de MFS, eso es porque una librería como MFS **puede usar funciones que estén en otras librerías**… Total, que tienes que incluir todas las librerías relacionadas para que todo vaya bien.

En el SETUP tienes que incluir las dos líneas que ves para que pueda funcionar el display. A partir de ahí, sólo tienes que llamar MFW.write() cuando quieras.

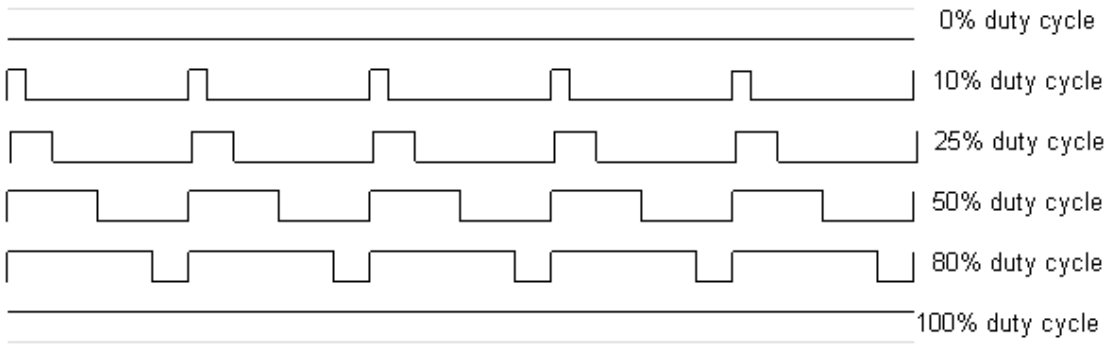
Fíjate que hemos llamado a la función en **el SETUP**, recuerda que ahí también podemos poner **acciones que sólo queremos que ocurran UNA VEZ**.

7. Encender un LED a distinto nivel (PMW)

Arduino da salidas digitales: ALTO/BAJO, 1/0, TRUE/FALSE…

Pero puede simularse una salida analógica con 256 niveles de intensidad (0-255).

Se llama PMW y se hace dando una salida que salta entre 0 y 5V, pero estando diferente tiempo en un valor y en otro. En promedio nos parecen valores intermedios de intensidad.



No todos los pines admiten este tipo de salida, lo sabemos porque al lado de su número aparece este símbolo ~

Para nuestros LEDs y nuestro Arduino UNO se puede hacer en el 10 y el 11. Dependerá de la tarjeta.

a) Primero haz un programa en el que un LED se ilumine a varios valores.
b) Después haz un programa en el que varíe entre encendido y apagado de manera suave (fade).

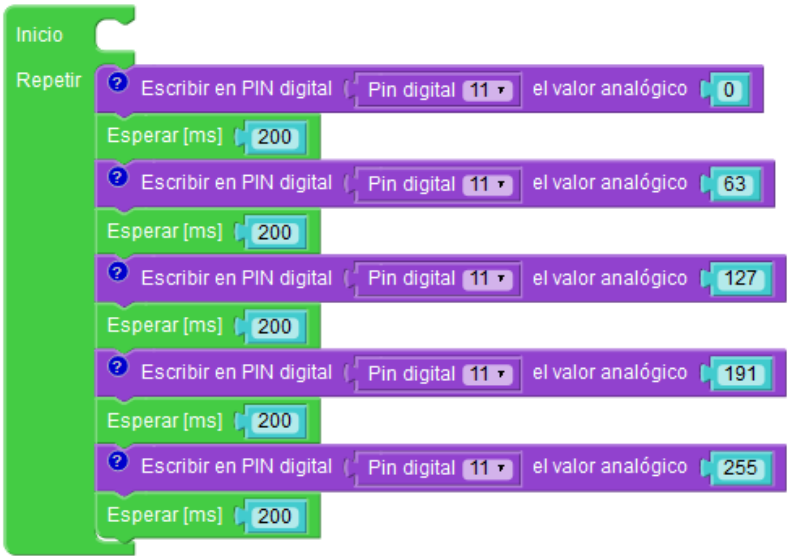
Añade a ambos programas las líneas necesarias para que se vea el valor de luminosidad en el display.

Hasta aquí casi todas las acciones sencillas que podemos hacer con los LEDs, ahora te toca a ti combinarlas para conseguir los efectos que desees.

Piensa en estos programas como “piezas” que puedes incluir en un programa mayor. De ti dependerá lo grande que quieras hacer el “puzle”.

OJO: El uso de PMW tira de interrupciones “Timer” y dará incompatibilidades con otras funciones como Servo (pines 9, 10), Tone (pines 3, 11) y SPI (pin 11) Estos pines son para Arduino UNO. Más info aquí https://www.luisllamas.es/salidas-analogicas-pwm-en-arduino/

SOLUCIÓN 7-1




```
#include <TimerOne.h>
#include <Wire.h>
#include <MultiFuncShield.h>
void setup() {

 Timer1.initialize();
 MFS.initialize(&Timer1); // initialize multi-function shield library

 MFS.write("Hi");
 delay(500);

 pinMode(11,OUTPUT);
}

void loop()
{
    analogWrite(11,0);
    MFS.write(0);
    delay(500);
    analogWrite(11,63);
    MFS.write(63);
    delay(500);
    analogWrite(11,127);
    MFS.write(127);
    delay(500);
    analogWrite(11,191);
    MFS.write(191);
    delay(500);
    analogWrite(11,255);
    MFS.write(255);
    delay(500);
}
```

Mejor si lo hacemos con un bucle. En la solución siguiente no hay opción… habría que hacer 255 repeticiones a mano.

SOLUCIÓN 7-2


```
#include <TimerOne.h>
#include <Wire.h>
#include <MultiFuncShield.h>

int i = 0;

void setup() {

 Timer1.initialize();
 MFS.initialize(&Timer1); // initialize multi-function shield library

 MFS.write("Hi");
 delay(500);

 pinMode(11,OUTPUT);
}

void loop()
{
    for (i = 255; i >= 0; i--) {
      analogWrite(11,i);
      MFS.write(i);
      delay(30);
     }
    for (i = 0; i <= 255; i++) {
      analogWrite(11,i);
      MFS.write(i);
      delay(30);
    }

}
```

ZUMBADOR

8. Producción de sonidos

El zumbador está conectado al pin digital 3.

Aunque en la librería del SHIELD hay funciones específicas, nosotros usaremos la de Arduino:

tone(pin,frecuencia,duración); Frecuencia en Hz, duración en ms.

Si no se especifica duración suena indefinidamente hasta que se manda otro sonido o bien se manda la orden de silenciar **noTone(pin).**

El sonido se ejecuta en background, eso quiere decir que el programa manda la orden de que suene y sigue ejecutando las órdenes siguientes. Si pones varios sonidos seguidos, se solaparán y no sonarán bien. Hay que incluir un delay de la misma duración que la nota para que la siguiente suene en su lugar correcto.

Estas son las frecuencias de las notas del piano (en Hz). No nos constan los límites de respuesta del zumbador del SHIELD ni del que uses, ni tu sensibilidad al sonido. Aconsejamos prudencia en el uso de frecuencias no fácilmente audibles fuera de (100-16 000 Hz). Volumen bajo, porfa.

Octavas	Cero	1	2	3	4	5	6	7	8
DO		33	65	131	262	523	1047	2093	4186
DO#		35	69	139	277	554	1109	2217	
RE		37	73	147	294	587	1175	2349	
RE#		39	78	156	311	622	1245	2489	
MI		41	82	165	330	659	1319	2637	
FA		44	87	175	349	698	1397	2794	
FA#		46	92	185	370	740	1480	2960	
SOL		49	98	196	392	784	1568	3136	
SOL#		52	104	208	415	831	1661	3322	
LA	28	55	110	220	440	880	1760	3520	
LA#	29	58	117	233	466	932	1865	3729	
SI	31	62	123	247	494	988	1976	3951	

a) Escribe una melodía
b) Haz un programa que vaya haciendo sonar frecuencias y mostrándolas en el display para ver límites de respuesta.

Si lo quieres ejecutar una vez, ponlo en el SETUP.

Se puede usar el bloque Zumbador en Visualino en el menú ZUM Bloqs. Hay dos, en uno tienes las notas de la escala natural preestablecidas, en el otro eliges tú la frecuencia libremente.

OJO: El uso de Tone tira de interrupciones "Timer" y dará problemas si usas PMW en pin 3 y pin 11 o si usas SPI en pin 11. Estos pines son para Arduino UNO. Más info aquí https://www.luisllamas.es/salidas-analogicas-pwm-en-arduino/

NOTA: Si cargas la librería **toneAC.h** puedes conseguir sonidos a distinto volumen:

Aquí puedes ver las instrucciones. Te permite fijar frecuencia, volumen y si quieres que se ejecute en background o no.

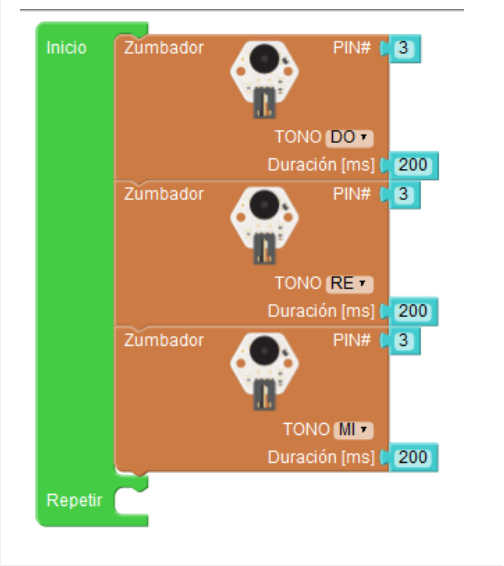
toneAC(frequency [, volume [, length [, background]]]) - Play a note.

- frequency - Play the specified frequency indefinitely, turn off with toneAC().
- volume - [optional] Set a volume level. (default: 10, range: 0 to 10 [0 = off])
- length - [optional] Set the length to play in milliseconds. (default: 0 [forever], range: 0 to 2^32-1)
- background - [optional] Play note in background or pause till finished? (default: false, values: true/false)

toneAC() - Stop output.
noToneAC() - Same as toneAC().

SOLUCIÓN 8-1

```
/***   Global variables   ***/

/***   Function declaration   ***/

void setup()
{
  tone(3,261,200);
  delay(200);
  tone(3,293,200);
  delay(200);
  tone(3,329,200);
  delay(200);

}

void loop()
{

}

/***   Function definition   ***/
```



SOLUCIÓN 8-2

```
#include <TimerOne.h>
#include <Wire.h>
#include <MultiFuncShield.h>

int i = 0;

void setup() {

 Timer1.initialize();
 MFS.initialize(&Timer1); //initialize shield library

 MFS.write("Hi");
 delay(500);

 }

void loop()
{
    for (i = 0; i <= 17000; i+=30) {
      tone(3,i,200);
      MFS.write(i);
      delay(200);
    }

}
```

Este código da un error en ejecución en la escritura en el display, que en sus librerías tiene en cuenta que se rebosa la capacidad de mostrar números.

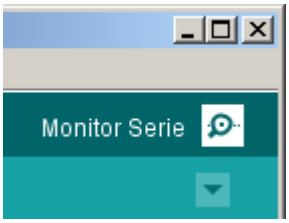
No olvidéis que para dar resultados con más detalle siempre tenemos el Monitor Serie.

PUERTO SERIE

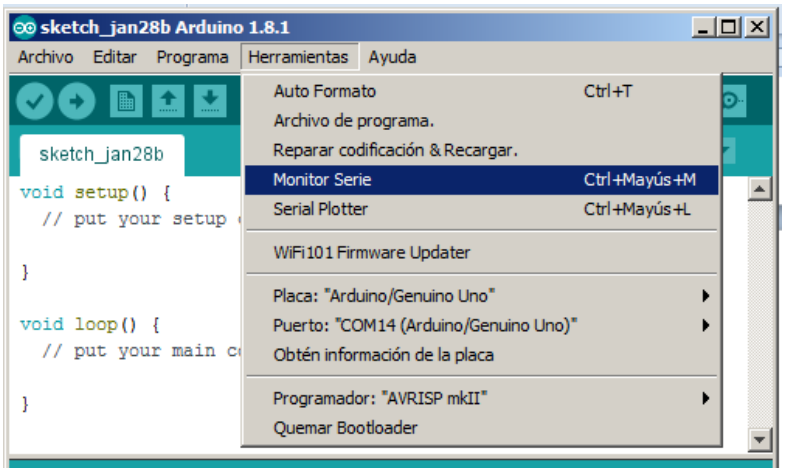
9. Salidas por el puerto Serie

El puerto serie es una forma de comunicar Arduino con otros dispositivos, para enviar o recibir datos. El envío se lo haremos típicamente por teclado, aunque podría hacerse vía Bluetooth (mediante una app de terminal en un móvil, p.ej.) o a través de una WIFI.

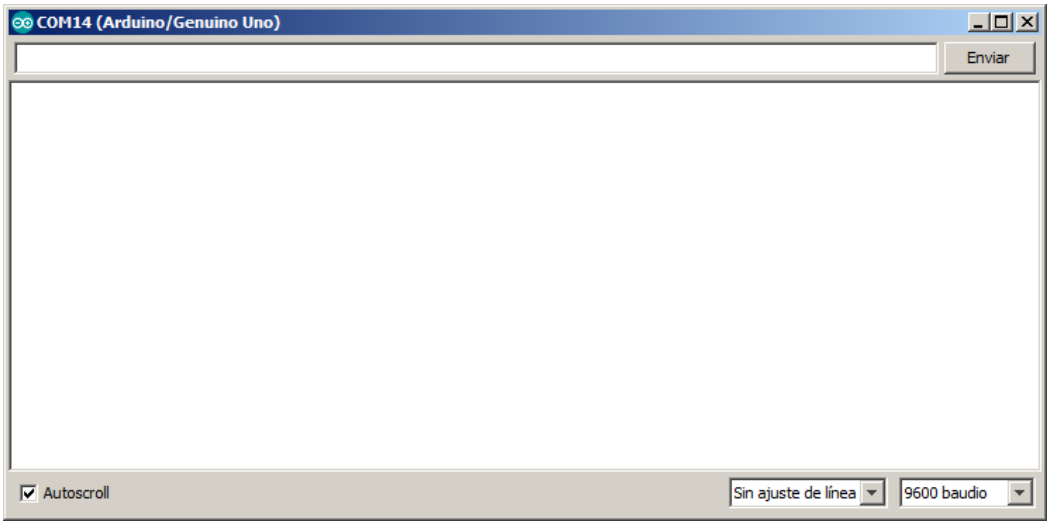
Para acceder al monitor lo podemos hacer con el botón de la parte superior derecha



O por menú. Ahí también aparece el Serial Plotter que nos hará una representación gráfica en tiempo real si vamos mandando datos desde la placa



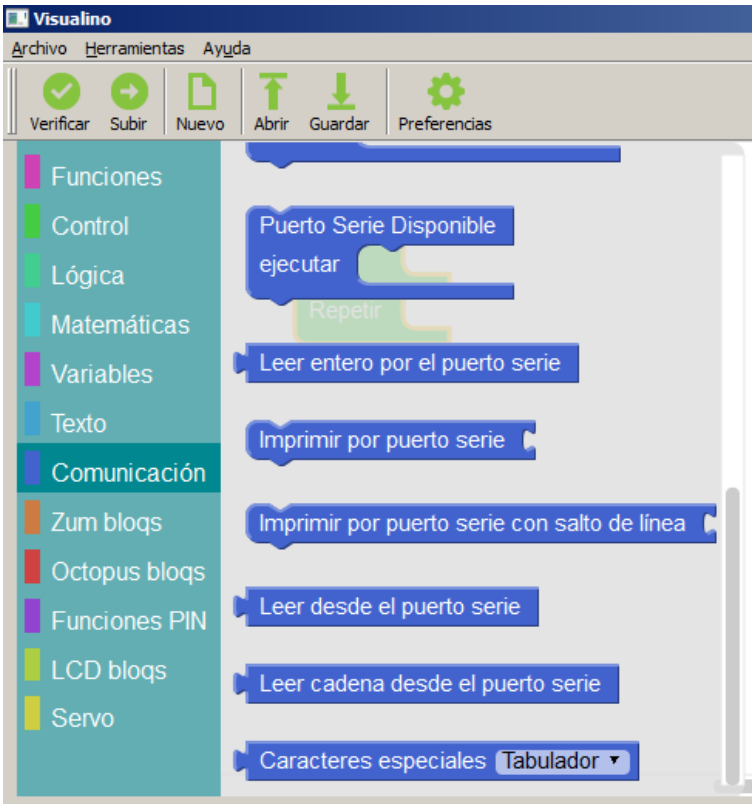
Esta es la apariencia que tiene en Monitor Serie



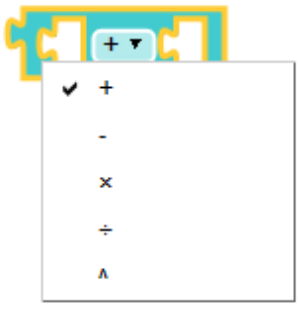
Debajo tenéis la velocidad de comunicación y opciones de visualización

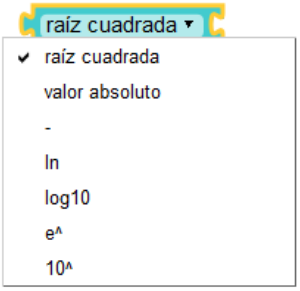
Arriba una caja de texto para escribir y enviar datos. En la pantalla se muestran los recibidos.

En Visualino podemos usar los comandos asociados en el menú COMUNICACIÓN



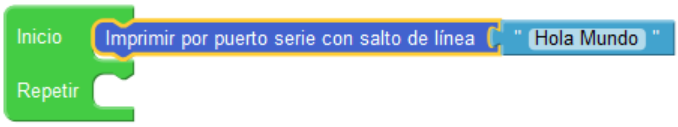
a) Escribe en el monitor Serie “Hola Mundo” una vez y luego repetidamente “Soy Yo”.
   - Mira lo que pasa con el autoscroll y añade un delay, si te parece
b) Escribe en el monitor Serie una cuenta atrás de diez a cero y la palabra “BOOM”.
   - Necesitarás una variable
   - Puedes añadir sonido
c) Escribe en el Serial Plotter la función raíz cuadrada de x (u otra que quieras) entre cero y 1000.
   - En el menú MATEMÁTICAS puedes hacer operaciones usando



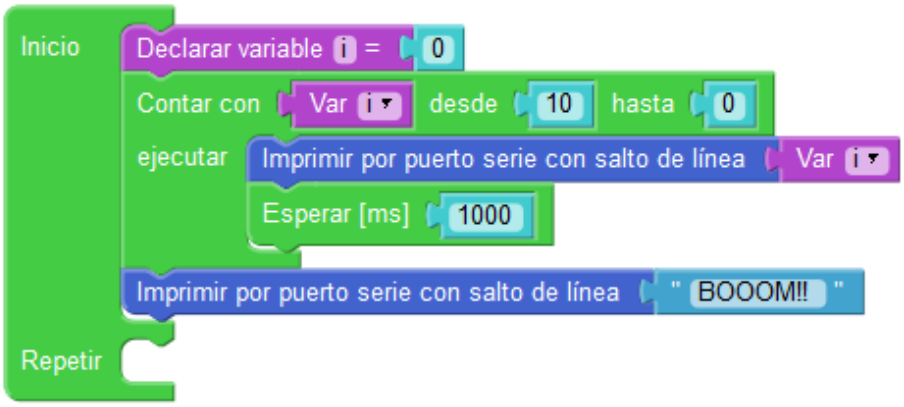


SOLUCIÓN 9-1

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

  Serial.print("Hola Mundo");
}

void loop()
{


}
```

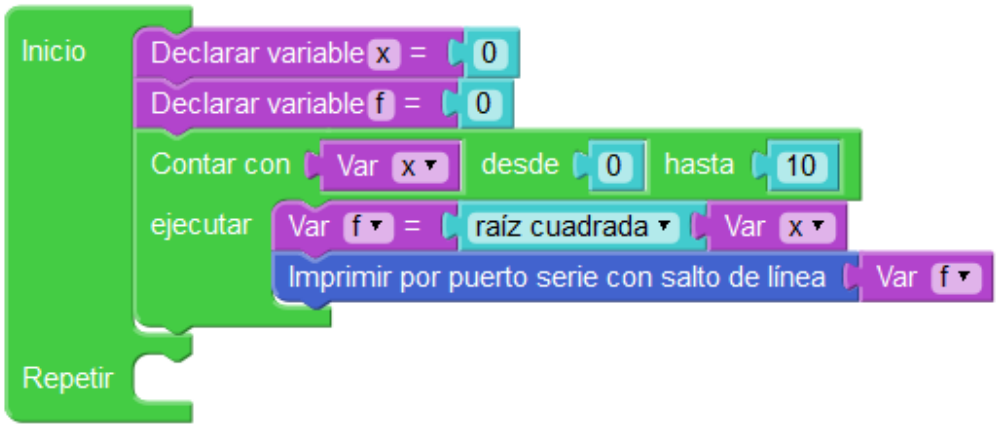


SOLUCIÓN 9-2



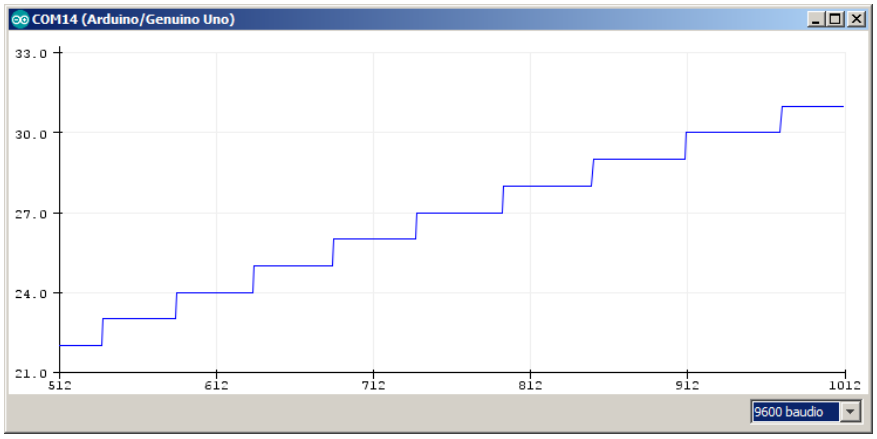

```
void setup()
{
  Serial.begin(9600);

    int i=0;
  for (i = 10; i >= 0; i--) {
    Serial.println(i);
    delay(1000);
  }
  Serial.println("BOOOM!!");

}

void loop()
{

}
```

SOLUCIÓN 9-3




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

  int x=0;
  int f=0;
  for (x = 0; x <= 1000; x++) {
    f=sqrt(x);
    Serial.println(f);
  }
}


void loop(){

}
```

Esto verás, se va dibujando según van llegando los datos.



NOTA: En el Serial Plotter pueden representarse varias series de valores a la vez si vas sacando los valores seguidos por comas, por ejemplo

23,54

23,48

Programación SENSORES

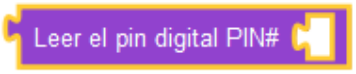
Pulsadores

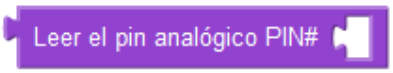
10. Enciende LED al pulsarse

¡¡NUESTRA PRIMERA DECISIÓN!!

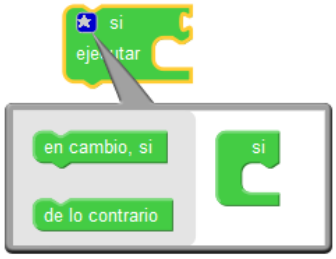
Hasta ahora generamos la salida en tiempo de programación, ahora vamos a reaccionar a lo que suceda "en el mundo" (en tiempo de ejecución). Será el usuario o los sensores los que disparen la respuesta.

Ahora necesitaremos "leer", por ejemplo el valor de los pulsadores. Recordemos que en este SHIELD están en lógica inversa.



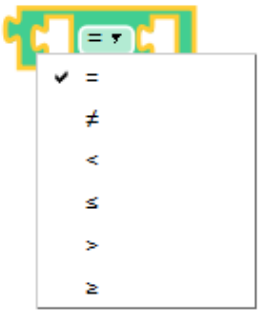


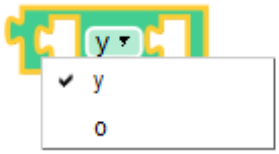
Para tomar la decisión usaremos un condicional



Tiene varios argumentos

1. CONDICIÓN LÓGICA que puede ser CIERTA O FALSA y que puede ser más o menos elaborada usando relaciones lógicas

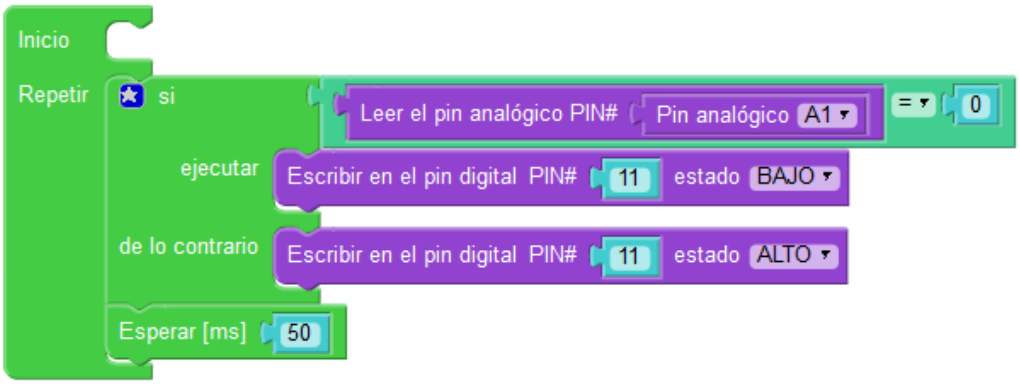




2. Acción a tomar cuando se cumple
3. Acción a tomar cuando no se cumple

a) Haz un programa en el que se encienda un LED al accionar un pulsador
b) Haz un programa en el que se encienda un LED al NO accionar un pulsador

SOLUCIÓN 10-1




```
void setup()
{
  pinMode(A1,INPUT);
  pinMode(11,OUTPUT);
}

void loop()
{
    if (analogRead(A1) == 0) {
      digitalWrite(11,LOW);
     }else {
      digitalWrite(11,HIGH);
     }
    delay(50);

}
```

El programa está constantemente ejecutando el LOOP de forma que mira una y otra vez si el botón está pulsado.

Si no apagásemos el LED en el caso de que lo encontrase sin pulsar, se quedaría pulsado después de la primera pulsación que se hiciera. Recordemos que sólo se ejecutan las órdenes que se dan, no se apaga por voluntad propia.

Como Arduino es muy rápido en el momento en el que se pulsa cuando está a punto de hacer contacto, puede que se lea varias veces, lo que se llama REBOTE.

Esto se puede solucionar por hardware o por software, el segundo será nuestro caso, añadiremos un pequeño delay antes del próximo ciclo y mejora. En el ejemplo Debounce del IDE tienes una forma de hacerlo bien.

SOLUCIÓN 10-2

Podríamos hacerlo de muchas maneras:

- Poniendo (igual a 1 en la condición del IF)
- Poniendo (distinto de 1 en la condición del IF)
- Cambiando ALTO por BAJO en la escritura del pin.

11. Pulsa para apagar, pulsa para encender

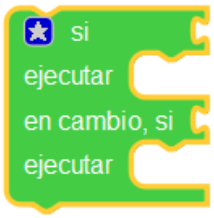
Aquí no nos sirve la misma opción, porque tenemos que tener en cuenta el estado del LED para ver qué acción llevamos a cabo (si encender o apagar).

Así que guardaremos en una variable el estado del led y guardaremos el estado del botón también en una variable para hacer más limpia la programación.

Recuerda añadir un pequeño retardo para evitar rebotes, y observa que, aun así, no funciona perfectamente todas las veces.

Usaremos el condicional, pero añadiremos una condición dentro de la primera condición, lo que se llama condicionales anidados. Por ejemplo:

- ¿Eres rubio?
- En caso de que no, ¿eres moreno?



También usaremos los bloques lógicos para hacer condiciones más elaboradas.

SOLUCIÓN 11

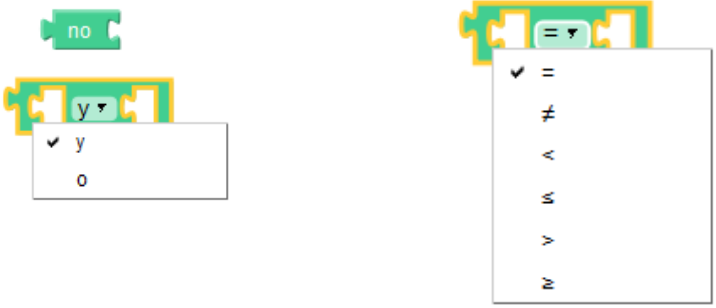

```
/***   Global variables   ***/
int estado_pulsador=1;
int estado_led=255;

/***   Function declaration   ***/

void setup()
{
  pinMode(A1,INPUT);
  pinMode(11,OUTPUT);
}


void loop()
{
    estado_pulsador=analogRead(A1);
    if ((estado_pulsador == 0) && (estado_led == 255)) {
      estado_led=0;
     }else if ((estado_pulsador == 0) && (estado_led == 0)) {
      estado_led=255;
     }
    analogWrite(11,estado_led);
    delay(100);

}

/***   Function definition   ***/
```

NOTA (avanzado): Si ponemos `#define led 2` esto no crea una variable ni ocupa espacio en memoria, será el preprocesador el que sustituya el valor antes de la compilación.

NOTA (avanzado): `if (cualquier cosa)` obliga a que se evalúe el paréntesis TRUE O FALSE, si es cero dará FALSE, en cualquier otro caso TRUE (incluso un carácter, p.ej.)

12. LED con combinación de pulsadores

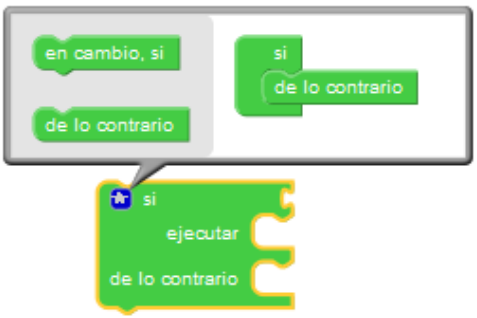
Usemos las funciones lógicas para combinar qué respuesta queremos dar con diferentes entradas. Combinando las funciones (AND, OR, NOT)



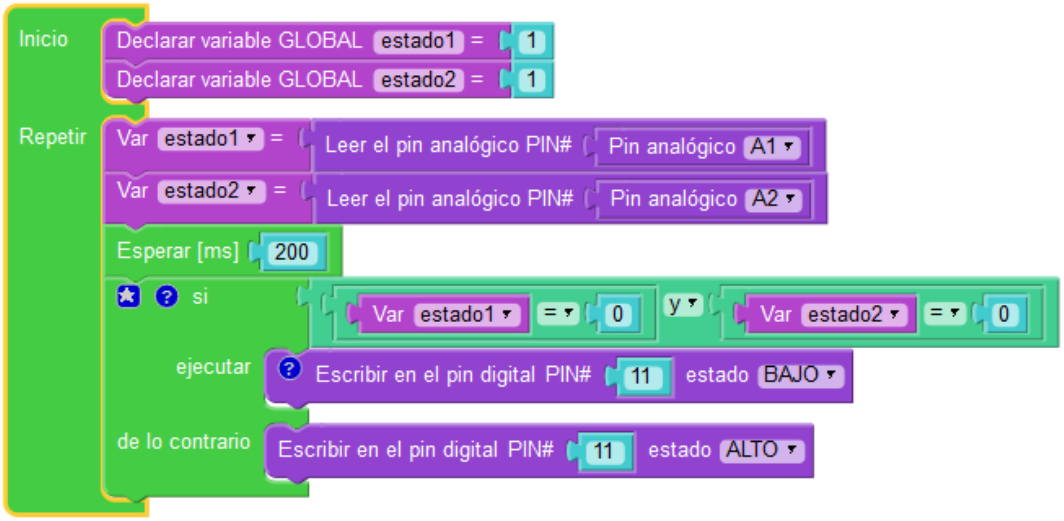
Por ejemplo:

a) Programa para que un LED se encienda cuando está pulsado A1 y A2 a la vez.
b) Retoca tu programa para que lo haga cuando sea A1 o A2
c) A1 y no A2, etc.

En este caso usa un condicional con opción “de lo contrario”. Eso te garantiza que alguna de las dos opciones se va a cumplir.



SOLUCIÓN 12




```
/***   Global variables   ***/
int estado1=0;
int estado2=0;

/***   Function declaration   ***/

void setup()
{
  pinMode(A1,INPUT);
  pinMode(A2,INPUT);
  pinMode(11,OUTPUT);
}

void loop()
{
    estado1=analogRead(A1);
    estado2=analogRead(A2);
    delay(200);
    if ((estado1 == 0) && (estado2 == 0)) {
      digitalWrite(11,LOW);
     }else {
      digitalWrite(11,HIGH);
     }

}

/***   Function definition   ***/
```

NOTA: Aunque nos resulta cómodo en Visualino poner el valor del pin en cada función, se aconseja definir una variable al principio del programa `int led 12` , por ejemplo, para poder entenderlo mejor o editar en un futuro sólo en ese punto si hay que cambiar el pin.

13. Mostrar estado del pulsador

Algo muy interesante es saber qué valores tenemos en las entradas, tanto para saberlo cuando estamos ejecutando, como cuando estamos diseñando y queremos saber qué está entrando.

a) Haz un programa que muestre el estado del botón por el Monitor Serie

Debe leerse:

Si no está pulsado: “El estado del pulsador es: 1023”

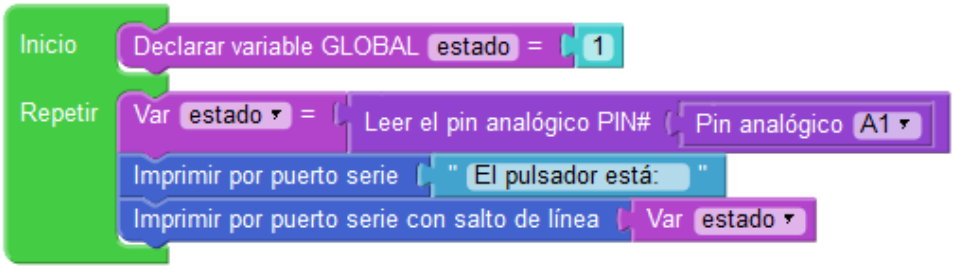
Si está pulsado: “El estado del pulsador es: 0”

La primera parte de la expresión es un texto, la última el valor de una variable.

Mira en el **serial plotter** también cómo se representa (en este caso, escribe sólo el valor).

b) También se podría mostrar por el display como vimos en otras prácticas

SOLUCIÓN 13



Por el puerto serie se pueden imprimir tanto texto como el valor de variables.

Hay un problema con Visualino y si se intentan insertar los caracteres especiales, como el tabulador, no genera el código correctamente, así que en estos bloques he puesto espacios entre la frase y el valor de la variable

```
/***   Global variables   ***/
int estado=1;

/***   Function declaration   ***/

void setup()
{

  pinMode(A1,INPUT);

  Serial.begin(9600);

}


void loop()
{
    estado=analogRead(A1);
    Serial.print("El pulsador está:");
    Serial.print("\t");
    Serial.println(estado);

}

/***   Function definition   ***/
```

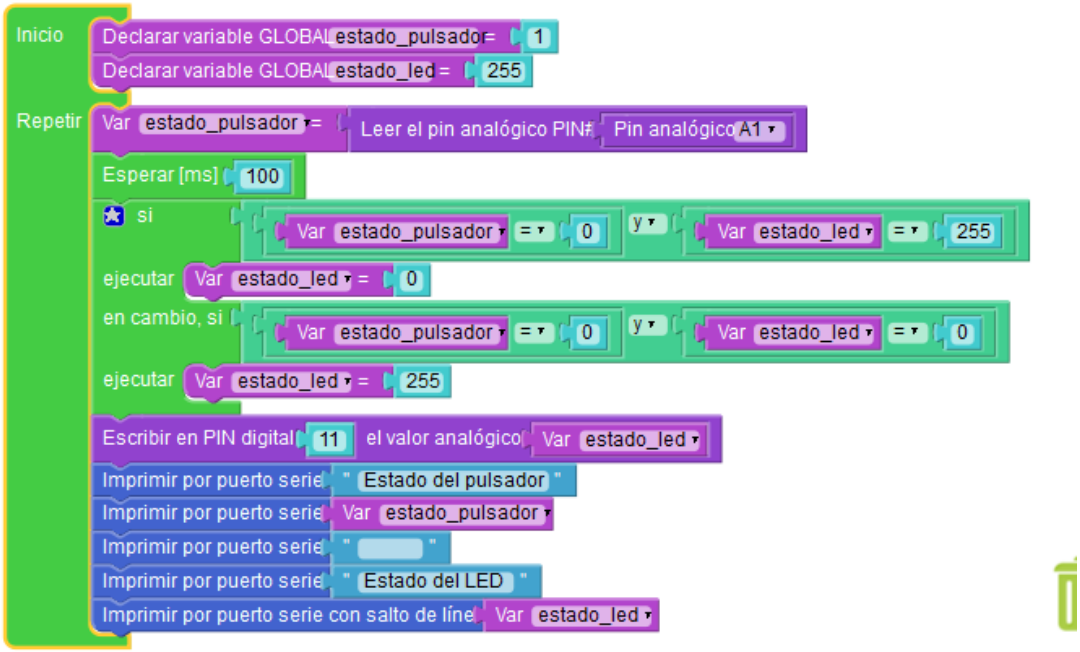
14. Mostrar estado del pulsador y del LED

Volveremos a hacer el circuito ON/OFF con cambio por pulsación y veremos que el estado del LED no es el del pulsador y por eso es necesario tener variables que nos indiquen el estado de las salidas, que no podemos leer directamente (como hacemos con las entradas).

Pulsa más rápido y más lento mirando el Monitor Serie y pensando en el delay que has puesto y entenderás mejor el concepto de “rebote”.

NOTA: Es posible saber el estado de una salida digital con la función digitalRead(), aunque esté definida como salida.

SOLUCIÓN 14



De nuevo, hubiéramos puesto un tabulador en lugar de espacios, pero no genera bien el código.

```
/***   Global variables   ***/
int estado_pulsador=0;
int estado_led=255;

void setup()
{
  pinMode(A1,INPUT);
  pinMode(11,OUTPUT);
  Serial.begin(9600);
}

void loop()
{
    estado_pulsador=analogRead(A1);
    delay(100);
    if ((estado_pulsador == 0) && (estado_led == 255)) {
      estado_led=0;
     }else if ((estado_pulsador == 0) && (estado_led == 0)) {
      estado_led=255;
     }
    analogWrite(11,estado_led);
    Serial.print("Estado del pulsador  ");
    Serial.print(estado_pulsador);
    Serial.print("\t");
    Serial.print("Estado del LED  ");
    Serial.println(estado_led);

}
```

15. Contar tiempos durante la ejecución. Función millis()

Saber el tiempo que ha pasado entre distintas acciones del usuario o activaciones de los sensores, puede ser muy importante.

La función millis() devuelve el tiempo en milisegundos desde el comienzo de la ejecución.

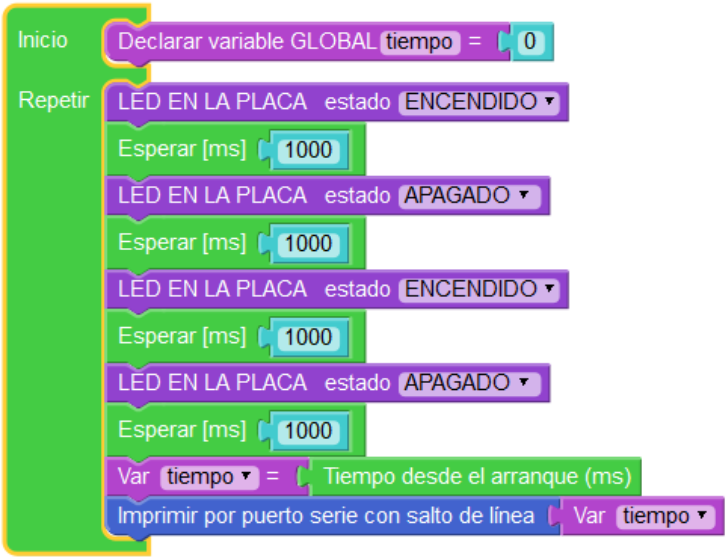
a) Empecemos haciendo un sencillo parpadeo doble y viendo el valor de millis() en ese momento, repetimos la operación y lo vamos escribiendo en el Monitor Serie.

Con esto veremos que ha pasado "casi" el tiempo de los delays, aunque de esa pequeña diferencia podemos deducir lo poquísimo que se tarda en ejecutar las acciones.

Al poco el tiempo comienza a ser negativo, cuando las variables "rebosan" no dan un error sino que siguen leyendo otras posiciones de memoria y darán salidas, aunque no tengan sentido.

Si queréis podéis guardar `millis()` en un `unsigned long` con mucha más capacidad y que es como lo hace Arduino.

SOLUCIÓN 15




```
/***   Global variables   ***/
int tiempo=0;

/***   Function declaration   ***/

void setup()
{
  pinMode(13,OUTPUT);
  Serial.begin(9600);
}


void loop()
{
    digitalWrite(13,HIGH);
    delay(1000);
    digitalWrite(13,LOW);
    delay(1000);
    digitalWrite(13,HIGH);
    delay(1000);
    digitalWrite(13,LOW);
    delay(1000);
    tiempo=millis();
    Serial.println(tiempo);
}

/***   Function definition   ***/
```

NOTA: Un uso interesante de la función millis() es para hacer "pausas" sin detener la ejecución del programa. Por ejemplo, si encendemos un LED, damos un delay, y luego lo apagamos. No se ejecuta nada durante ese delay.

La forma de hacerlo usando millis() es encenderlo, consultar millis, seguir tomando acciones y consultando millis() y, cuando se cumpla el intervalo que queríamos, mandar la orden de apagado.

El siguiente programa enciende el LED del pin12 y lo mantiene encendido diez segundos, pero mientras se va consultando si se acciona el pulsador del pin A1 que encenderá el LED del pin13.-

```
/***   Global variables   ***/
int estado=0;
int tiempo_final=0;
int tiempo_inicial;

/***   Function declaration   ***/

void setup()
{
  pinMode(12,OUTPUT);
  pinMode(13,OUTPUT);

  tiempo_inicial=millis();

  digitalWrite(12,LOW);
  digitalWrite(13,HIGH);

  pinMode(A1,INPUT);
}

void loop()
{
    tiempo_final=millis();
    if (tiempo_final - tiempo_inicial > 10000) {
      digitalWrite(12,HIGH);
     }
    estado=analogRead(A1);
    delay(100);
    if (estado == 0) {
      digitalWrite(13,LOW);
     }else {
      digitalWrite(13,HIGH);
     }
}

/***   Function definition   ***/
```

16. Esperando…

- PRIMER BUCLE CON CONDICIÓN DE PARO

Hemos hecho bucles en los que el número de repeticiones lo marcaba un FOR. Ese valor puede darse en la programación o tomarse de algún sensor, pero en muchas ocasiones nos interesa que se repita y repita una acción HASTA QUE ocurra algo o MIENTRAS QUE pasa cierta cosa, sin que nos importe el número de repeticiones, sino los eventos.

Por ejemplo:	Lava platos HASTA que no quede ninguno Lava platos MIENTRAS queden en el fregadero

El uso de MIENTRAS O HASTA lo marcará qué nos resulta más sencillo en la programación, pero podremos hacer bucles equivalentes.

La diferencia entre los dos bloques de la derecha:

1. Hacer-Mientras

   Primero hace una iteración y luego consulta la condición

2. Mientras- Hacer

   Primero consulta la condición y luego hace una iteración

Hay que cuidar los límites de las condiciones para no quedarse corto o pasarse (tiempo<10 o tiempo <=10). Con pequeños ejemplos y cuidado se puede ver si funciona como uno quiere.

Otra diferencia es que el primer bloque siempre ejecutará una acción y eso puede ser beneficioso en algunas circunstancias.

a) Haz un programa que encienda un LED y espere hasta que se pulse para apagarse (o que permanezca encendido mientras no se pulse)

SOLUCIÓN 16


```
/***   Global variables   ***/
int estado_pulsador=1023;

/***   Function declaration   ***/

void setup()
{

  pinMode(A1,INPUT);
  pinMode(11,OUTPUT);

    do {
    estado_pulsador=analogRead(A1);
    delay(100);
    digitalWrite(11,LOW);

  } while (estado_pulsador != 0);
  digitalWrite(11,HIGH);

}


void loop()
{
}
/***   Function definition   ***/
```

17. Medida del tiempo de reacción

Vamos a calcular el tiempo de reacción desde que se ilumina un LED hasta que se pulsa un botón. Para eso necesitaremos la función millis() que ya hemos visto.

Recordamos que la función millis() devuelve el tiempo en milisegundos desde el comienzo de la ejecución.

Para saber un intervalo de tiempo entre dos acciones, necesitamos dos variables.

Cuando sucede la primera acción metemos en tiempo1 el valor de millis()

Cuando sucede la segunda acción metemos en tiempo2 el valor de millis()

La resta de los dos tiempos es el intervalo que buscábamos.

Puedes hacer el programa con cuatro variables

a) Haz un programa que:
- Deje pasar un tiempo aleatorio entre dos y cuatro segundos (para que sorprenda)
- Encienda un LED y apunte el tiempo
- Espere HASTA que pulsemos un botón y apunte el tiempo
- Nos devuelva el tiempo transcurrido por el Monitor Serie (mira el plotter también)
- Se podría implementar que se vea en el display.

SOLUCIÓN 17


```
/***   Global variables   ***/
int estado=1023;
int tiempo_ini=0;
int tiempo_fin=0;
int tiempo_reaccion=0;

void setup()
{
  pinMode(11,OUTPUT);
  pinMode(A1,INPUT);
  Serial.begin(9600);
}

void loop()
{
    delay(random(2000,4000));
    estado=1023;
    digitalWrite(11,LOW);
    tiempo_ini=millis();
    while (estado == 1023) {
      tiempo_fin=millis();
      estado=analogRead(A1);
     }
    digitalWrite(11,HIGH);
    tiempo_reaccion=tiempo_fin - tiempo_ini;
    Serial.println(tiempo_reaccion);
}
```

Al hacerlo para el display da un error que se elimina con poner un pequeñísimo delay() después de la lectura del pulsador

```
#include <TimerOne.h>
#include <Wire.h>
#include <MultiFuncShield.h>

int estado=1023;
long tiempo_ini=0;
long tiempo_fin=0;
long tiempo_reaccion=0;

void setup()
{
  Timer1.initialize();
  MFS.initialize(&Timer1);

  pinMode(11,OUTPUT);
  pinMode(A1,INPUT);
  Serial.begin(9600);
}

void loop()
{
    delay(random(2000,4000));
    estado=1023;
    digitalWrite(11,LOW);
    tiempo_ini=millis();
    while (estado == 1023) {
      tiempo_fin=millis();
      estado=analogRead(A1);
      delay(5);
     }
    digitalWrite(11,HIGH);
    tiempo_reaccion=tiempo_fin - tiempo_ini;
    Serial.println(tiempo_reaccion);
    MFS.write((int)tiempo_reaccion);
}
```

18. Alarma.

- MI PRIMER BUCLE INFINITO… ¡APOSTA!

Ya vemos que WHILE nos permite detener el bucle principal del programa durante un tiempo no prefijado.

El problema de los bucles con condición de paro es que… no se llegue a cumplir la condición y que el programa se quede detenido allí para siempre.

Es importante fijarse en que **la variable que se evalúa en la condición tiene que cambiar su valor dentro del bucle,** porque si nada cambia… nada cambiará. Esto puede ser una comprobación que se haga siempre para intentar evitar quedarse "embuclao".

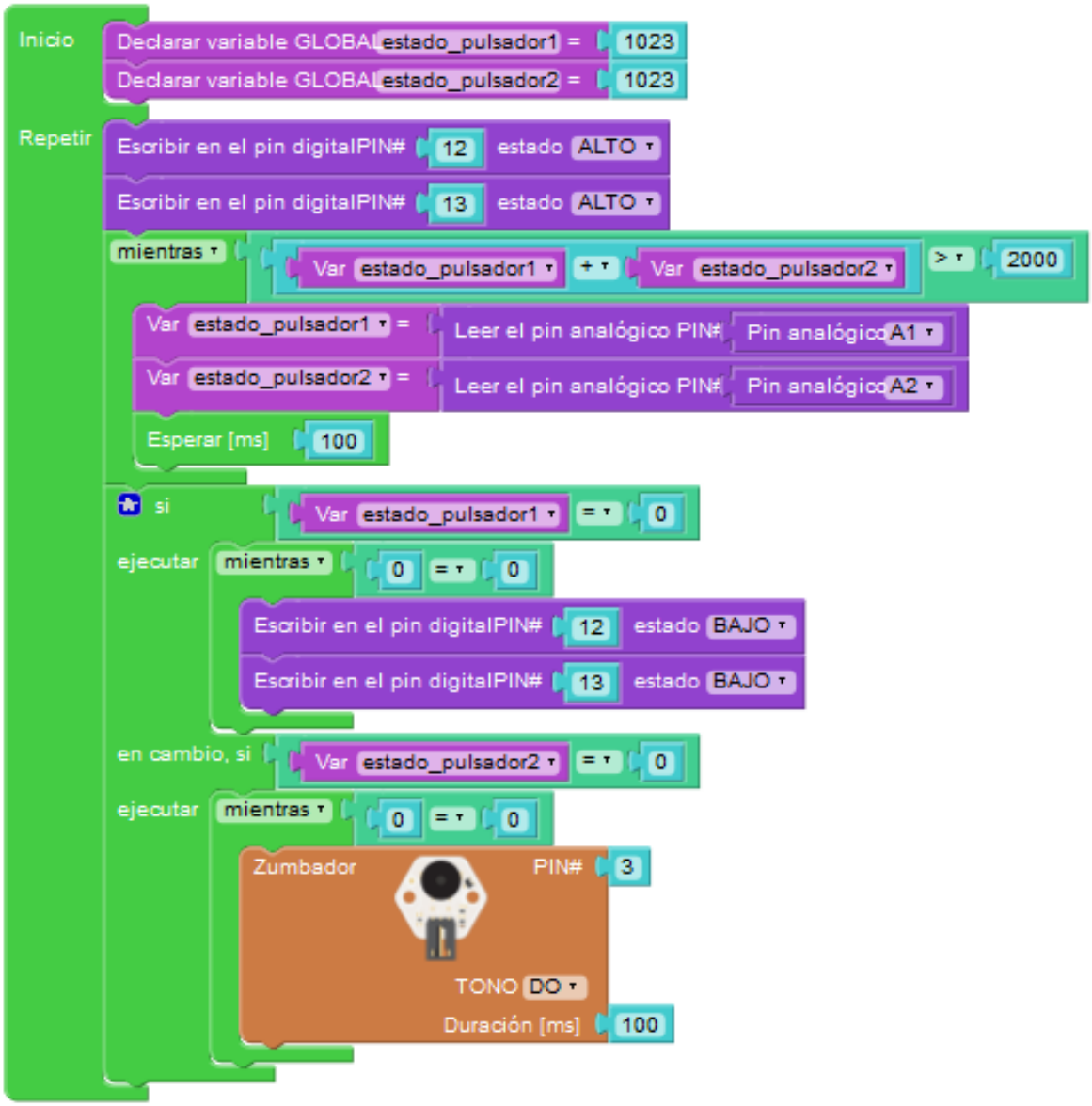
"Embuclarse" es, en realidad, algo que también **podemos usar a nuestro favor**, si en algún caso deseamos que nuestro sistema se quede parado y bloqueado en un punto: que no vuelva a evaluar ninguna variable, a mirar ningún sensor…

Un ejemplo claro es una alarma.

a) Haz un programa que encienda un LED si se pulsa el botón correcto y que haga sonar el zumbador si pulsas el botón incorrecto. Dejaremos parado el sistema en ambos casos.

Sobre todo, en el caso de fallo, no queremos que el usuario se "arrepienta" y deje de pulsar, de forma que la alarma dejaría de sonar. ¡Queremos que se quede atrapado!

SOLUCIÓN 18



Al inicializarse los pin, Arduino los pone a valor BAJO, eso hace que se nos enciendan "solos" porque nosotros trabajamos con lógica inversa. Así que tenemos que apagarlos explícitamente, o no configurarlos hasta justo antes de usarlos.

NOTA:

- Los pines digitales pueden configurarse en el LOOP.

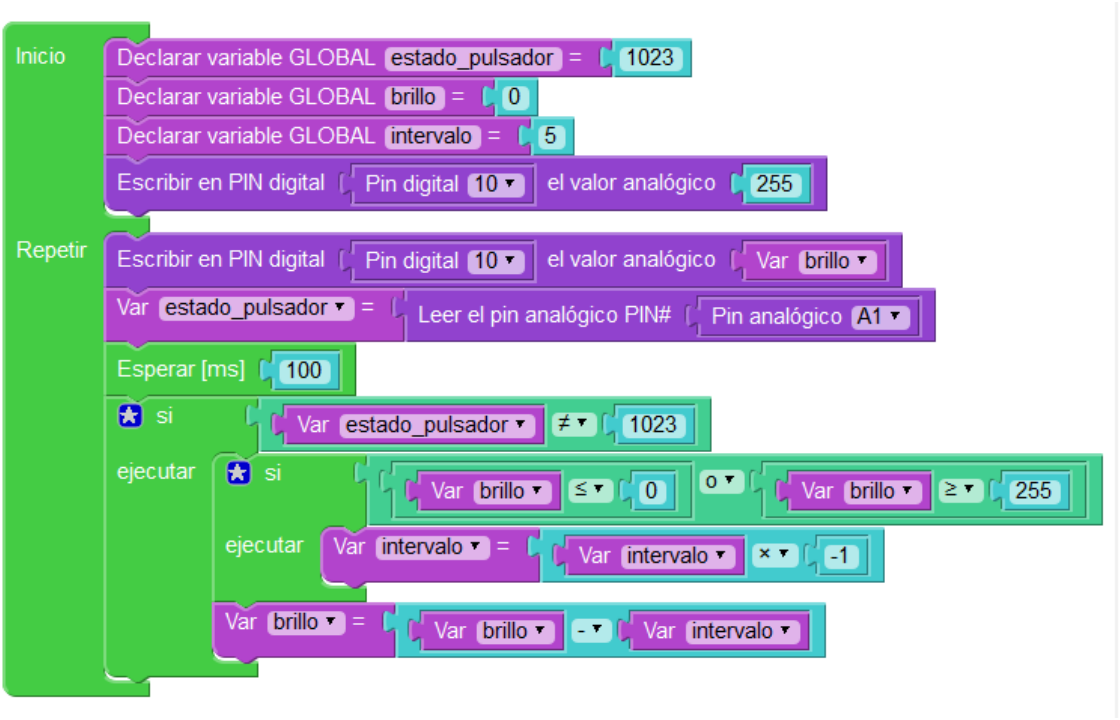
- No es necesario configurar los pines analógicos, responderán a lo que se les pida.

```
int estado_pulsador1=1023;
int estado_pulsador2=1023;

void setup()
{
  pinMode(12,OUTPUT);
  pinMode(13,OUTPUT);
  pinMode(A1,INPUT);
  pinMode(A2,INPUT);
}

void loop()
{
    digitalWrite(12,HIGH);
    digitalWrite(13,HIGH);
    while (estado_pulsador1 + estado_pulsador2 > 2000) {
      estado_pulsador1=analogRead(A1);
      estado_pulsador2=analogRead(A2);
      delay(100);
     }
    if (estado_pulsador1 == 0) {
      while (0 == 0) {
        digitalWrite(12,LOW);
        digitalWrite(13,LOW);
       }
     }else if (estado_pulsador2 == 0) {
      while (0 == 0) {
        tone(3,261,100);
        delay(100);
       }
     }

}

/***   Function definition   ***/
```

19. Cambio de luminosidad continua por pulsación

a) Haz un programa que vaya cambiando la luminosidad de un LED gradualmente mientras está accionando el pulsador. Si se suelta se queda en el punto en el que esté.
   - Usa variables para: brillo, la tasa de cambio de luz y el estado del pulsador
   - Pon el LED al valor del brillo y mira el estado del pulsador
   - Según esté el pulsador no haces nada o cambias luminosidad
   - Mira primero si está a tope (en cero o en 255) y cambia de signo el incremento
   - Incrementa (o decrementa) el brillo
   - Vuelta a empezar

SOLUCIÓN 19




```
/***   Global variables   ***/
int estado_pulsador=1023;
int brillo=0;
int intervalo=5;

/***   Function declaration   ***/

void setup()
{
  pinMode(10,OUTPUT);
  pinMode(A1,INPUT);

  analogWrite(10,255);
}

void loop()
{
    analogWrite(10,brillo);
    estado_pulsador=analogRead(A1);
    delay(100);
    if (estado_pulsador != 1023) {
      if ((brillo <= 0) || (brillo >= 255)) {
        intervalo=intervalo * -1;
       }
      brillo=brillo - intervalo;
     }
}

/***   Function definition   ***/
```

Queda bonito sacar por display el nivel de brillo

```
#include <TimerOne.h>
#include <Wire.h>
#include <MultiFuncShield.h>

/***   Global variables   ***/
int estado_pulsador=1023;
int brillo=0;
int intervalo=5;

/***   Function declaration   ***/

void setup()
{
  pinMode(10,OUTPUT);
  pinMode(A1,INPUT);

 Timer1.initialize();
 MFS.initialize(&Timer1);  //  initialize  multi-function  shield
library


  analogWrite(10,255);
}

void loop()
{
    analogWrite(10,brillo);
    estado_pulsador=analogRead(A1);
    delay(100);
    if (estado_pulsador != 1023) {
      if ((brillo <= 0) || (brillo >= 255)) {
        intervalo=intervalo * -1;
       }
      brillo=brillo - intervalo;
      MFS.write(brillo);
     }
}

/***   Function definition   ***/
```

20. Función sin retorno y con parámetros

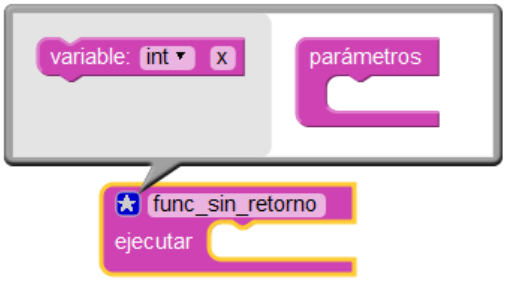
Ya hicimos dos funciones SIN retorno y SIN parámetros, que se limitaban a encender o apagar, respectivamente, todos los LEDs.

Ahora haremos los otros tres tipos de funciones

- SIN retorno y CON parámetros
- CON retorno y SIN parámetros
- CON retorno y CON parámetros

a) Haz un programa con una función SIN retorno y CON parámetros que controla dos luces

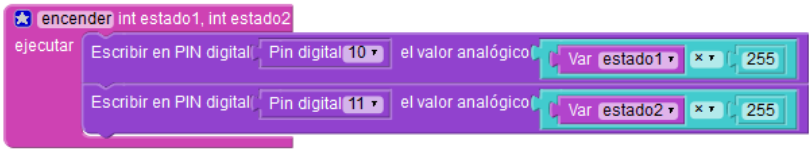
- void encender (estado1,estado2);



- Invócala unas cuantas veces con distintos valores para los parámetros
- Nótese que las variables estado1 y estado2 sólo existen dentro de la función, son VARIABLES LOCALES

Recordemos que **las funciones pueden definirse al principio o al final del código**, fuera del SETUP y del LOOP, y que **el sistema las buscará** cuando el programa las invoque.

SOLUCIÓN 20




```
/***   Global variables   ***/

/***   Function declaration   ***/
void encender (int estado1, int estado2);

void setup()
{
  pinMode(10,OUTPUT);
  pinMode(11,OUTPUT);
}

void loop()
{
    encender(0, 0);
    delay(200);
    encender(1, 0);
    delay(200);
    encender(0, 1);
    delay(200);
    encender(1, 1);
    delay(200);
}

/***   Function definition   ***/
void encender (int estado1, int estado2) {
  analogWrite(10,estado1 * 255);
  analogWrite(11,estado2 * 255);
 }
```

21. Función con retorno y sin parámetros

a) Haz una función que te dé el número de pulsadores que estén pulsados simultáneamente. No necesita parámetros, pero devolverá un número.

.

- **Cuando invoques a una función con retorno tienes que hacerlo desde una variable**, porque la función va a devolver un valor y tiene que tener una caja "donde caer".

SOLUCIÓN 21


Inicio
Declarar variable GLOBAL estado1 = 0
Declarar variable GLOBAL estado2 = 0
Repetir
Var estado1 = Leer el pin analógico PIN# Pin analógico A1
Var estado2 = Leer el pin analógico PIN# Pin analógico A2
Esperar [ms] 100
Imprimir por puerto serie con salto de líne numero_activos


numero_activos
ejecutar
Declarar variable suma = 0
Var suma = Var estado1 + Var estado2
si Var suma = 0
ejecutar Var suma = 2
en cambio, si Var suma = 1023
ejecutar Var suma = 1
en cambio, si Var suma = 2046
ejecutar Var suma = 0
devuelve Var suma

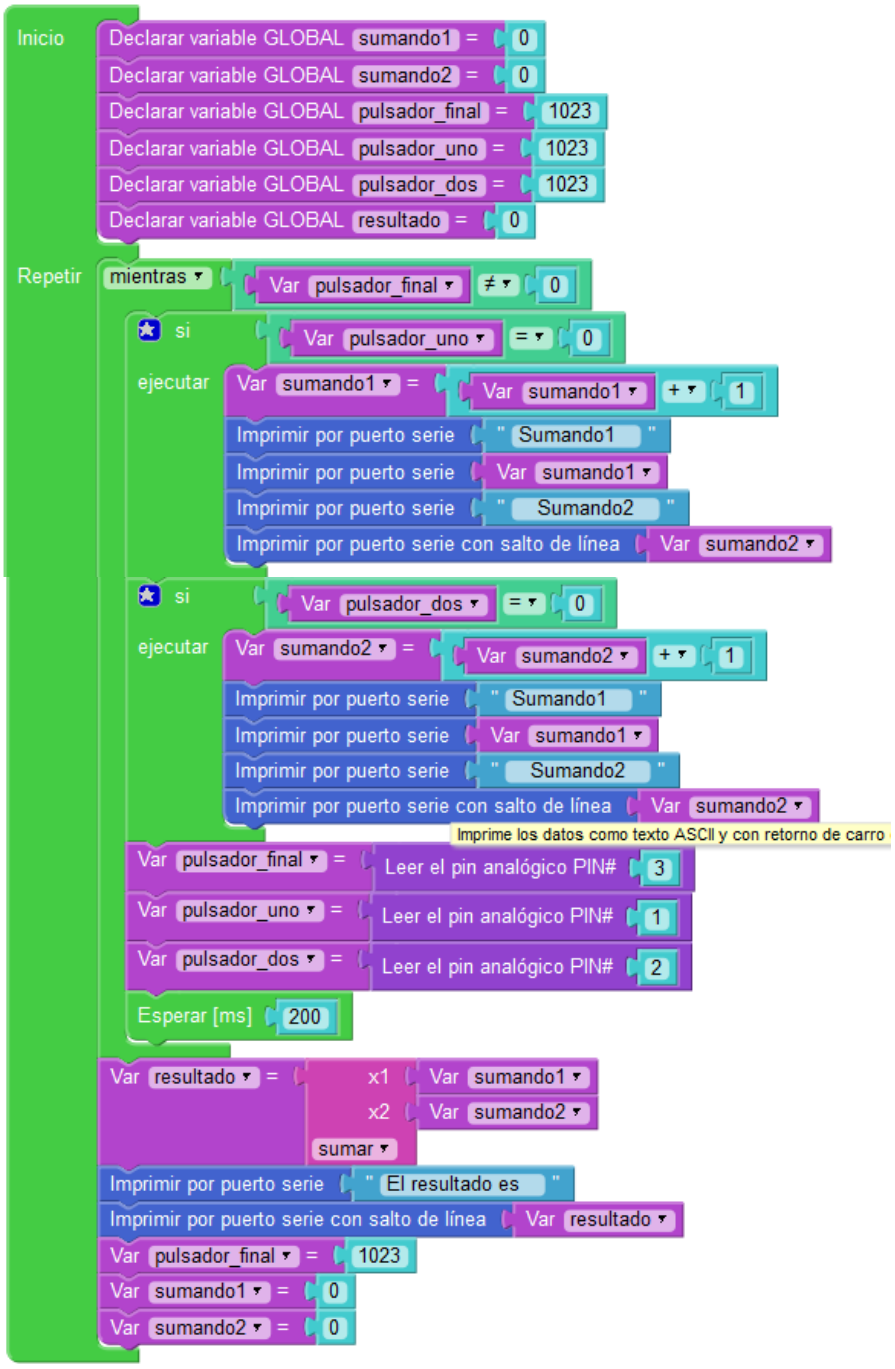
```
/***   Global variables   ***/
int estado1=0;
int estado2=0;

/***   Function declaration   ***/
int numero_activos ();

void setup()
{
  pinMode(A1,INPUT);
  pinMode(A2,INPUT);
  Serial.begin(9600);
}

void loop()
{
    estado1=analogRead(A1);
    estado2=analogRead(A2);
    delay(100);
    Serial.println(numero_activos());
}

/***   Function definition   ***/
int numero_activos () {
  int suma=0;
  suma=estado1 + estado2;
  if (suma == 0) {
    suma=2;
   }else if (suma == 1023) {
    suma=1;
   }else if (suma == 2046) {
    suma=0;
   }

  return suma;
 }
```

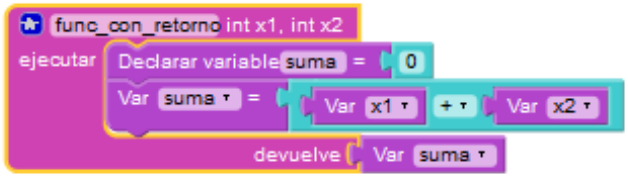
22. Función con retorno y con parámetros

a) Haz una función que sume dos números
   - El primero será el número de veces que se pulse el pulsador uno
   - El segundo lo mismo con el pulsador dos
   - El pulsador tres da el resultado por el puerto serie

- Con una función MIENTRAS esperamos a que se pulse el pulsador 3
- Con condicionales sumamos unidades a cada sumando, según el pulsador que se accione.
- Recuerda poner un retardo para disminuir los rebotes

SOLUCIÓN 22


Inicio
Declarar variable GLOBAL sumando1 = 0
Declarar variable GLOBAL sumando2 = 0
Declarar variable GLOBAL pulsador_final = 1023
Declarar variable GLOBAL pulsador_uno = 1023
Declarar variable GLOBAL pulsador_dos = 1023
Declarar variable GLOBAL resultado = 0
Repetir mientras Var pulsador_final ≠ 0
si Var pulsador_uno = 0
ejecutar Var sumando1 = Var sumando1 + 1
Imprimir por puerto serie " Sumando1 "
Imprimir por puerto serie Var sumando1
Imprimir por puerto serie " Sumando2 "
Imprimir por puerto serie con salto de línea Var sumando2
si Var pulsador_dos = 0
ejecutar Var sumando2 = Var sumando2 + 1
Imprimir por puerto serie " Sumando1 "
Imprimir por puerto serie Var sumando1
Imprimir por puerto serie " Sumando2 "
Imprimir por puerto serie con salto de línea Var sumando2
Imprime los datos como texto ASCII y con retorno de carro
Var pulsador_final = Leer el pin analógico PIN# 3
Var pulsador_uno = Leer el pin analógico PIN# 1
Var pulsador_dos = Leer el pin analógico PIN# 2
Esperar [ms] 200
Var resultado = x1 Var sumando1
x2 Var sumando2
sumar
Imprimir por puerto serie " El resultado es "
Imprimir por puerto serie con salto de línea Var resultado
Var pulsador_final = 1023
Var sumando1 = 0
Var sumando2 = 0


func_con_retorno int x1, int x2
ejecutar Declarar variable suma = 0
Var suma = Var x1 + Var x2
devuelve Var suma

```
/***   Global variables   ***/
int sumando1=0;
int sumando2=0;
int pulsador_final=1023;
int pulsador_uno=1023;
int pulsador_dos=1023;
int resultado=0;

/***   Function declaration   ***/
int func_con_retorno (int x1, int x2);

void setup()
{
  Serial.begin(9600);
  pinMode(3,INPUT);
  pinMode(1,INPUT);
  pinMode(2,INPUT);
}

void loop()
{
    while (pulsador_final != 0) { //no resultado hasta pulsar3
      if (pulsador_uno == 0) {
        sumando1=sumando1 + 1;    //+1 al sumando1 con pulsador1
        Serial.print("Sumando1   ");
        Serial.print(sumando1);
        Serial.print("   Sumando2   ");
        Serial.println(sumando2);
       }
      if (pulsador_dos == 0) {  //+1 al sumando2 con pulsador2
        sumando2=sumando2 + 1;
        Serial.print("Sumando1   ");
        Serial.print(sumando1);
        Serial.print("   Sumando2   ");
        Serial.println(sumando2);
       }
      pulsador_final=analogRead(3); //leemos pulsadores
      pulsador_uno=analogRead(1);
      pulsador_dos=analogRead(2);
      delay(200);                    // disminuimos rebote
     }
    resultado=sumar(sumando1, sumando2); //
    Serial.print("El resultado es   ");
    Serial.println(resultado);
    pulsador_final=1023;          // reiniciamos variables
    sumando1=0;
    sumando2=0;
}
/***   Function definition   ***/
int sumar (int x1, int x2) {
  int suma=0;
  suma=x1 + x2;

  return suma;
 }
```

23. Melodía

- Mi primer VECTOR

Vamos a aprovechar el zumbador para mostrar una manera de encapsular la información: los vectores, más general: las matrices. En inglés ARRAY

Puede ser una tira de valores. Por ejemplo

int mivector[3]= {3, 27, 32};

Esto es un array de enteros (int), pero puede hacerse con cualesquiera otros objetos.

Para referirnos a una de las componentes escribiremos

mivector[2]

En nuestro caso es igual a 32, porque **las componentes se numeran empezando con el cero.**

Los vectores nos facilitan su lectura y escritura de forma paramétrica, p. ej.

```
int pares[5];
void setup() {

  Serial.begin(9600);

  for(int i=0;i<5;i++){     // se escriben las componentes
  pares[i]=2*i+2;
  }

  for(int i=0;i<5;i++){     // se imprimen las componentes
  Serial.println(pares[i]);
  }
}

void loop() {

}
```

a) Hagamos una melodía, pero guardemos las frecuencias de las notas, las notas de la melodía y los silencios entre notas en arrays.

SOLUCIÓN 23

Vamos a analizar un programa estupendo de Antonio Guillermo Pérez Coronilla.

Guarda las frecuencias en vectores, uno para cada nota (DO, RE…). En cada vector habrá cinco octavas de esa nota.

La función nota() para hacerlas sonar, la tiene definida al final

```
void nota(int frec, int t)
{
    tone(spk,frec);      // suena la nota frec recibida
    delay(t);            // para después de un tiempo t
}
```

La primera variable nos da la frecuencia y la segunda la duración. En Visualino nos generaría el siguiente código (equivalente)

```
void nota(int frec, int t)
{
    tone(spk,frec,t);      // suena la nota frec recibida
    delay(t);            // para después de un tiempo t
}
```

Lo demás son invocaciones a notas y a silencios noTone(spk); delay(1000);

Vamos a escucharlo.

```
/*************************/
/*     popurri para Arduino     */
/*************************/

/*************** Antonio Guillermo Pérez Coronilla
**************/

/* declaración de variables */
int spk=3;                              // altavoz a GND y pin 3
int c[5]={131,262,523,1046,2093};       // frecuencias 4 octavas
de Do
int cs[5]={139,277,554,1108,2217};      // Do#
int d[5]={147,294,587,1175,2349};       // Re
int ds[5]={156,311,622,1244,2489};    // Re#
int e[5]={165,330,659,1319,2637};      // Mi
int f[5]={175,349,698,1397,2794};       // Fa
int fs[5]={185,370,740,1480,2960};     // Fa#
int g[5]={196,392,784,1568,3136};     // Sol
int gs[5]={208,415,831,1661,3322};   // Sol#
int a[5]={220,440,880,1760,3520};      // La
int as[5]={233,466,932,1866,3729};    // La#
int b[5]={247,494,988,1976,3951};      // Si

void nota(int a, int b);            // declaración de la función
auxiliar. Recibe dos números enteros.
```

```
void setup()
{
/*************************************/
/*                  HARRY POTTER                    */
/*************************************/
nota(b[2], 500);
nota(e[3],1000);
nota(g[3], 250);
nota(fs[3],250);
nota(e[3],1000);
nota(b[3],500);
nota(a[3],1250);
nota(fs[3],1000);
nota(b[2], 500);
nota(e[3],1000);
nota(g[3],250);
nota(fs[3],250);
nota(d[3],1000);
nota(e[3],500 );
nota(b[2],1000 );
noTone(spk); delay(1000);
nota(b[2], 500);
nota(e[3],1000);
nota(g[3], 250);
nota(fs[3],250);
nota(e[3],1000);
nota(b[3],500);
nota(d[4],1000);
nota(cs[4],500);
nota(c[4],1000);
nota(a[3],500);
nota(c[4],1000);
nota(b[3],250);
nota(as[3],250);
nota(b[2],1000);
nota(g[3],500);
nota(e[3],1000);
noTone(spk);
delay(2000);

/******************/
/*     STAR WARS      */
/******************/
/**** tema principal ****/
nota(d[1],150);noTone(spk);delay(50);
nota(d[1],150);noTone(spk);delay(50);
nota(d[1],150);noTone(spk);delay(50);
nota(g[1],900);noTone(spk);delay(150);
nota(d[2],900);noTone(spk);delay(50);
nota(c[2],150);noTone(spk);delay(50);
nota(b[1],150);noTone(spk);delay(50);
nota(a[1],150);noTone(spk);delay(50);
nota(g[2],900);noTone(spk);delay(150);
nota(d[2],900);noTone(spk);delay(100);
nota(c[2],150);noTone(spk);delay(50);
nota(b[1],150);noTone(spk);delay(50);
```

```
nota(a[1],150);noTone(spk);delay(50);
nota(g[2],900);noTone(spk);delay(150);
nota(d[2],900);noTone(spk);delay(100);
nota(c[2],150);noTone(spk);delay(50);
nota(b[1],150);noTone(spk);delay(50);
nota(c[2],150);noTone(spk);delay(50);
nota(a[1],1200);noTone(spk);delay(2000);

/**** marcha del imperio   ****/
nota(g[2],500);noTone(spk);delay(100);
nota(g[2],500);noTone(spk);delay(100);
nota(g[2],500);noTone(spk);delay(100);
nota(ds[2],500);noTone(spk);delay(1);
nota(as[2],125);noTone(spk);delay(25);
nota(g[2],500);noTone(spk);delay(100);
nota(ds[2],500);noTone(spk);delay(1);
nota(as[2],125);noTone(spk);delay(25);
nota(g[2],500);
noTone(spk);delay(2000);

/**********************/
/*     entre dos aguas     */
/**********************/
nota(a[1],400);noTone(spk);delay(400);
nota(e[1],400);noTone(spk);delay(400);
nota(a[1],400);noTone(spk);delay(200);
nota(e[1],200);noTone(spk);delay(200);
nota(a[1],200);noTone(spk);delay(200);
nota(as[1],100);noTone(spk);delay(100);
nota(b[1],400);noTone(spk);delay(400);
nota(fs[1],400);noTone(spk);delay(400);
nota(b[1],400);noTone(spk);delay(200);
nota(fs[1],200);noTone(spk);delay(200);
nota(b[1],200);noTone(spk);delay(200);
nota(as[1],100);noTone(spk);delay(100);
nota(a[1],400);noTone(spk);delay(400);
nota(e[1],400);noTone(spk);delay(400);
nota(a[1],400);noTone(spk);delay(400);
}

void nota(int frec, int t)
{
    tone(spk,frec);      // suena la nota frec recibida
    delay(t);                // para despues de un tiempo t
}

void loop()
{
}
```

PUERTO SERIE

Por el puerto serie también se pueden enviar datos a Arduino, con lo que se convierte en un "sensor" más.

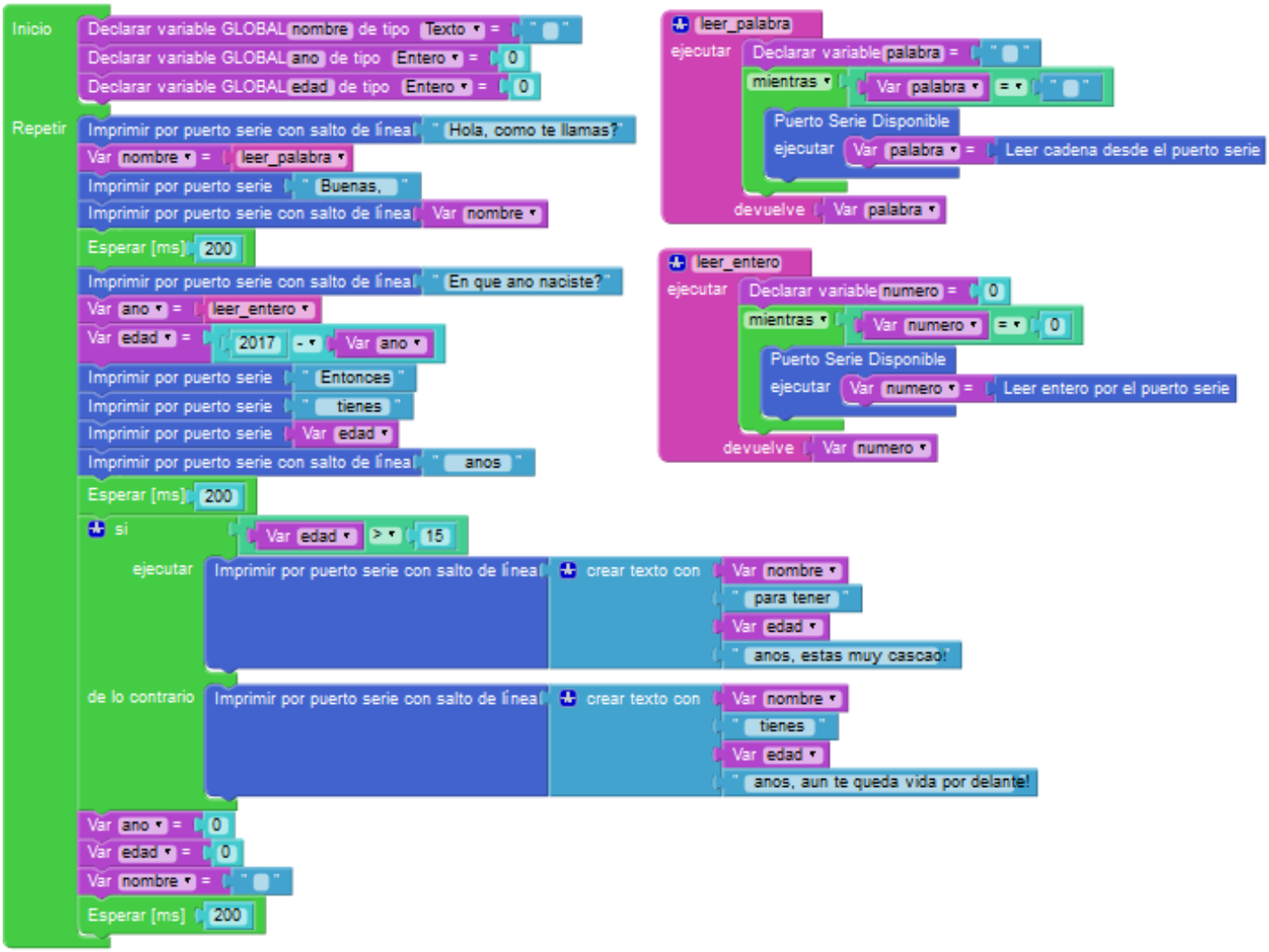
Usaremos el teclado, pero si añadimos un módulo Bluetooth o uno WIFI, también podían llegar datos por medio de terminales en móviles, PCs remotos, etc.

24. (Mini) Inteligencia Artificial

a) Hagamos un programa que hable con nosotros.
   - Básicamente se trata de que te haga preguntas
   - Espere tus respuestas
   - Guarde los valores que le das
   - Haga operaciones con ellas
   - Tome decisiones (condicionales)
   - Responda

Extendiendo esto lo que uno quiera, podría ir cubriendo cada vez un rango mayor de respuestas y "dar el pego" al menos durante un tiempo.

SOLUCIÓN 24



Es más sencillo generar una función sin parámetros y con retorno cada vez que queramos recoger un dato del usuario (una para texto y otra para enteros).

El bucle está esperando hasta que la variable tome un valor para pasar a la pregunta siguiente.

```
/***   Global variables   ***/
String nombre;
int ano;
int edad;

/***   Function declaration   ***/
String leer_palabra ();
int leer_entero ();

void setup()
{
  nombre="";
  ano=0;
  edad=0;

  Serial.begin(9600);

}

void loop()
{
    Serial.println("Hola, como te llamas?");
    nombre=leer_palabra();  // función que recoge el dato
    Serial.print("Buenas,   ");
    Serial.println(nombre);
    delay(200);
    Serial.println("En que ano naciste?");
    ano=leer_entero();      // función que recoge el dato
    edad=2017 - ano;
    Serial.print("Entonces");
    Serial.print("   tienes  ");
    Serial.print(edad);
    Serial.println("   anos  ");
    delay(200);
    if (edad > 15) {
      Serial.println(String(nombre) + String(" para tener ") +
String(edad) + String(" anos, estas muy cascao!"));
     }else {
      Serial.println(String(nombre) + String("  tienes  ") +
String(edad) + String(" anos, aun te queda vida por delante!"));
     }
    ano=0;
    edad=0;          //reiniciamos variables
    nombre="";
    delay(200);

}



//Sigue en página siguiente las funciones
```

```
/***   Function definition   ***/
String leer_palabra () {
  String palabra="";
  while (palabra == "") {   //espera a q se llene la variable
    if (Serial.available()>0){
      palabra=Serial.readString();

    }
   }

  return palabra;
 }
int leer_entero () {
  int numero=0;
  while (numero == 0) {     //espera a q se llene la variable
    if (Serial.available()>0){
      numero=Serial.parseInt();

    }
   }

  return numero;
 }
```

25. Elegir opción por teclado

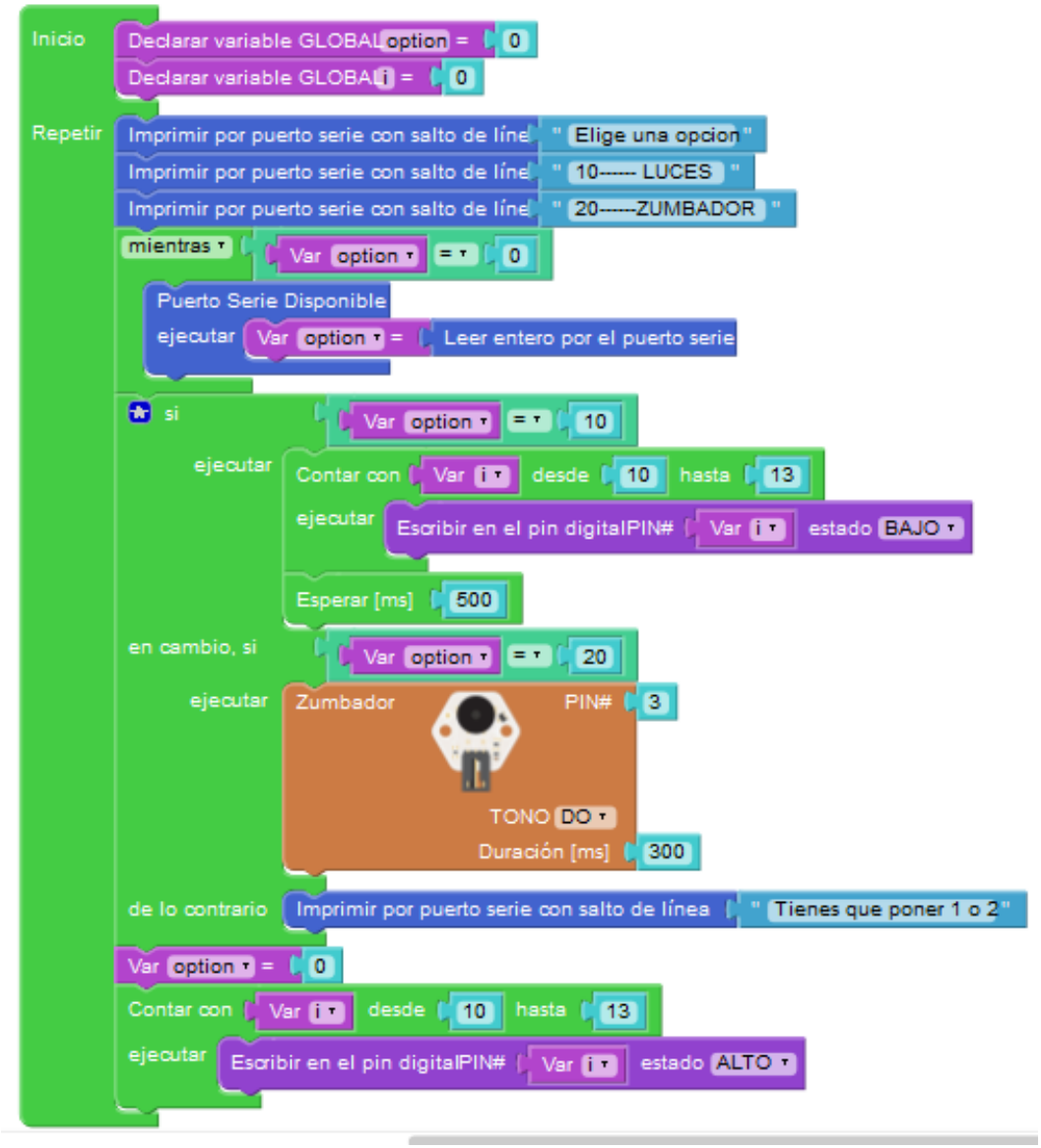
Esta debería ser una estructura típica switch/case aunque también se puede resolver con condicionales.

No hay bloque switch/case en Visualino, probablemente por ahorrarse la molestia de los errores con los tipos de variables, ya que switch/case sólo admite int y char.

Esta estructura se usa cuando se quieren separar casos en una variable que son disjuntos, por ejemplo, números distintos o letras distintas.

a) Hagamos un programa que
   - Nos pida un número
   - Si es un 1 encienda los LEDs
   - Si es un 2 haga sonar el zumbador
   - En cualquier otro caso, vuelva a preguntar.

SOLUCIÓN 25



De nuevo usamos un WHILE para mantener al programa esperando la respuesta, pero lo suyo hubiera sido.

```
switch (option){
   case 1:
      instrucciones;
      break;
   case 2:
      instrucciones;
      break;
   default:
      instrucciones;
      break;
}
```

La instrucción BREAK hace que salte fuera del SWITCH y siga la ejecución normal del programa.

PRUÉBALO!

```
/***   Global variables   ***/
int option=0;
int i=0;

/***   Function declaration   ***/

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

void loop()
{
    Serial.println("Elige una opcion");
    Serial.println("10------ LUCES");
    Serial.println("20------ZUMBADOR");
    while (option == 0) {
      if (Serial.available()>0){
        option=Serial.parseInt();

      }
     }
    if (option == 10) {
      for (i = 10; i <= 13; i++) {
        pinMode(i,OUTPUT);
        digitalWrite(i,LOW);
      }
      delay(500);
     }else if (option == 20) {
      tone(3,261,300);
      delay(300);
     }else {
      Serial.println("Tienes que poner 10 o 20");
     }
    option=0;
    for (i = 10; i <= 13; i++) {
      pinMode(i,OUTPUT);
      digitalWrite(i,HIGH);
    }
}

/***   Function definition   ***/
```

Si la variable hubiera sido char habría quedado

case ‘R’:

case ‘S’:

NOTA: Pongo estas letras y números tan raros en lugar de 1 y 2 o ‘a’ y ‘b’, para que no penséis que representan el orden de los casos, sino el valor de la variable para ese caso concreto.

POTENCIÓMETRO

El potenciómetro está conectado a una entrada analógica, pero los valores están discretizados entre 0 y 1023. Nos representaría cualquier sensor analógico que conectáramos a Arduino: LDR, Termistores, etc. A partir de la toma del valor la programación sería idéntica, independientemente del origen.

El potenciómetro que viene en el SHIELD es muy sensible. Es otra manera de decir que hay que darle un montón de vueltas para que varíe un poco (cuidado no te emociones y lo rompas).

26. Mostrar el valor de la entrada del potenciómetro en el display y en el Monitor Serie

Un ejercicio sencillo para ver cómo varía y tener controlado el componente.

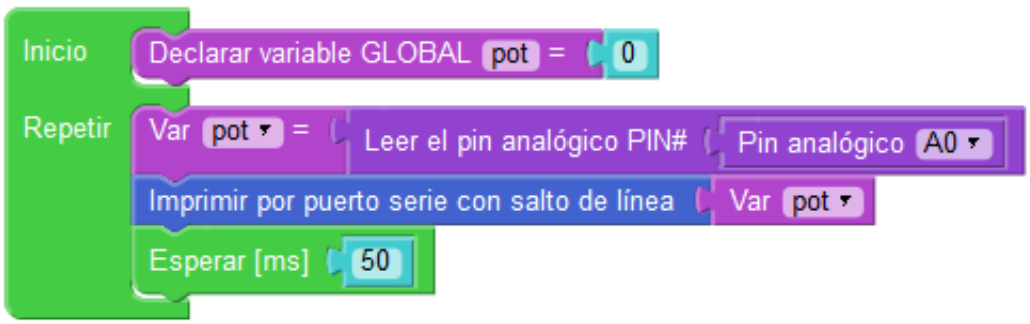
Los valores están entre 0 y 1023

Mostremos también gráficamente los valores por el Serial Plotter.

SOLUCIÓN 26-1

```
#include <TimerOne.h>
#include <Wire.h>
#include <MultiFuncShield.h>

int pot=0;

void setup()
{
  Timer1.initialize();
  MFS.initialize(&Timer1);
}

void loop()
{
    pot=analogRead(A0);
    MFS.write(pot);
}
```

SOLUCIÓN 26-2

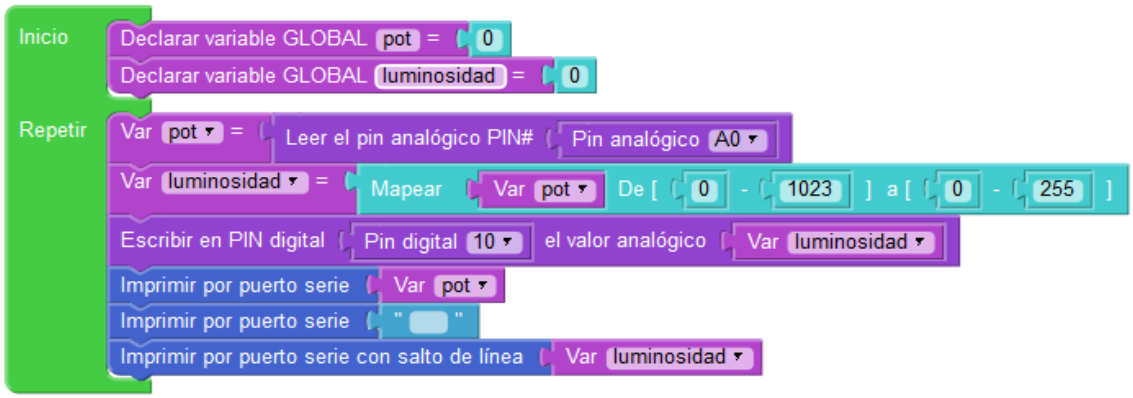



```
/***   Global variables   ***/
int pot=0;

/***   Function declaration   ***/

void setup()
{
  pinMode(A0,INPUT);
  Serial.begin(9600);
}

void loop()
{
    pot=analogRead(A0);
    Serial.println(pot);
    delay(50);
}

/***   Function definition   ***/
```

27. Control de la luminosidad de un LED "Continua" (MAPEO)

a) Hagamos un programa que varíe la luminosidad de un LED de forma continua.
   - Hay un problema. Queremos convertir la entrada en una salida, pero una va entre 0 y 1023 y otra entre 0 y 255. Para eso está la función MAPEAR.
   - map(variableOrigen, origen_inicio, origen_fin, destino_inicio, destino_fin);

SOLUCIÓN 27

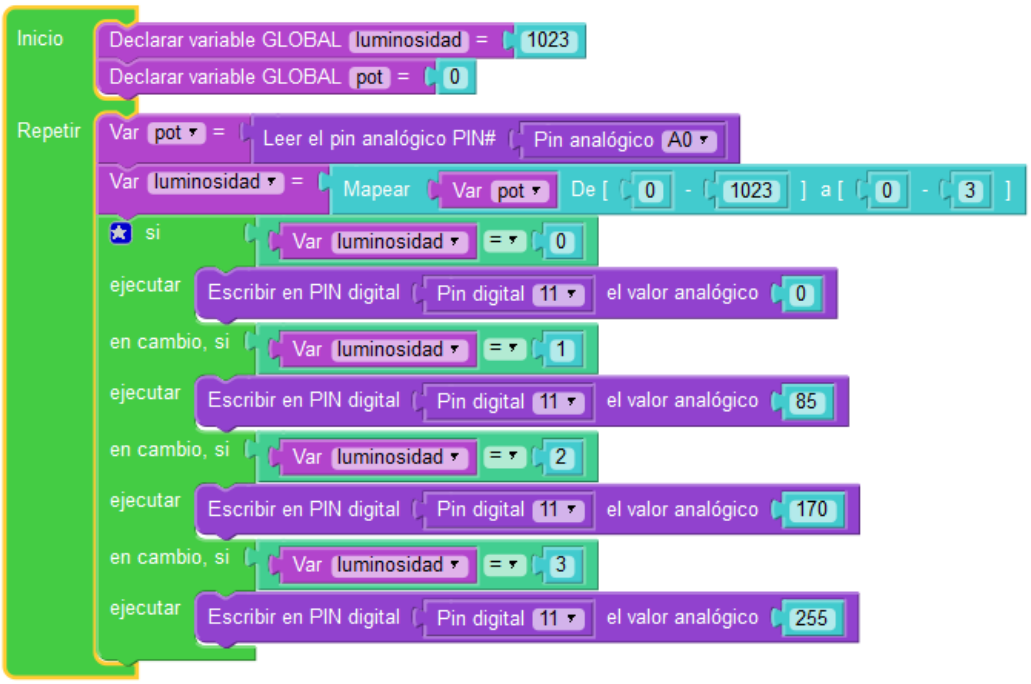



```
/***   Global variables   ***/
int pot=0;
int luminosidad=0;

/***   Function declaration   ***/

void setup()
{
  pinMode(A0,INPUT);
  pinMode(10,OUTPUT);
  Serial.begin(9600);
}

void loop()
{
    pot=analogRead(A0);
    luminosidad=map(pot,0,1023,0,255);
    analogWrite(10,luminosidad);
    Serial.print(pot);      // valores en Puerto serie
    Serial.print("\t");
    Serial.println(luminosidad);


}

/***   Function definition   ***/
```

28. Control de la luminosidad de un LED “Saltos” (MAPEO)

a) Hagamos un mapeo de nuevo, pero en pocos valores, sólo en 4.
   - Si la variable de destino es entera, la función map redondea y no hay problema alguno.

SOLUCIÓN 28

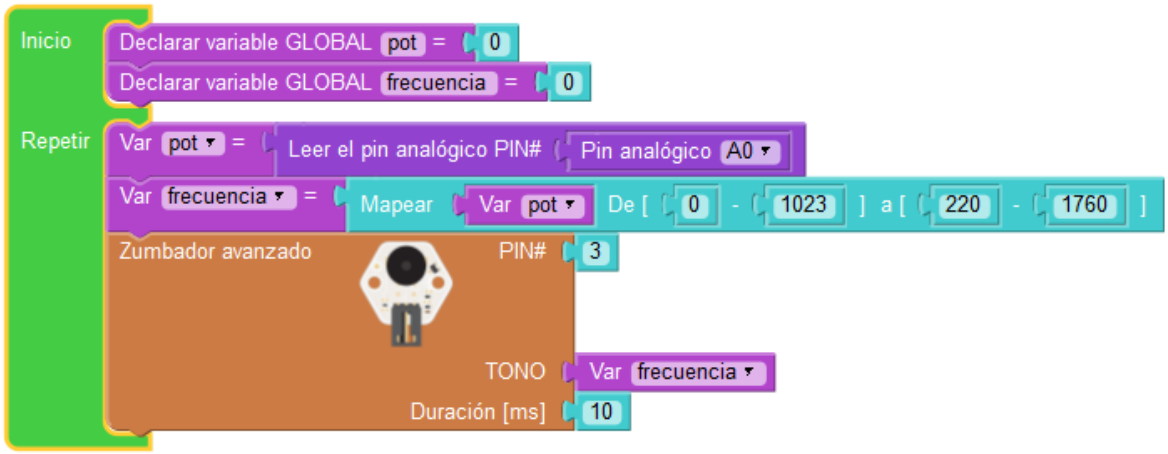



```
/***   Global variables   ***/
int luminosidad=1023;
int pot=0;

/***   Function declaration   ***/

void setup()
{
  pinMode(A0,INPUT);
  pinMode(11,OUTPUT);
}

void loop()
{
    pot=analogRead(A0);
    luminosidad=map(pot,0,1023,0,3);
    if (luminosidad == 0) {
      analogWrite(11,0);
     }else if (luminosidad == 1) {
      analogWrite(11,85);
     }else if (luminosidad == 2) {
      analogWrite(11,170);
     }else if (luminosidad == 3) {
      analogWrite(11,255);
     }
}
/***   Function definition   ***/
```

29. Control de la frecuencia con potenciómetro

a) Hagamos un programa que vaya cambiando la frecuencia con el giro del potenciómetro.

SOLUCIÓN 29




```
/***   Global variables   ***/
int pot=0;
int frecuencia=0;

/***   Function declaration   ***/

void setup()
{
  pinMode(A0,INPUT);
}


void loop()
{
    pot=analogRead(A0);
    frecuencia=map(pot,0,1023,220,1760);
    tone(3,frecuencia,10);
    delay(10);

}

/***   Function definition   ***/
```

Se oye cierto “chirrido” debido a los saltos de 10 milisegundos

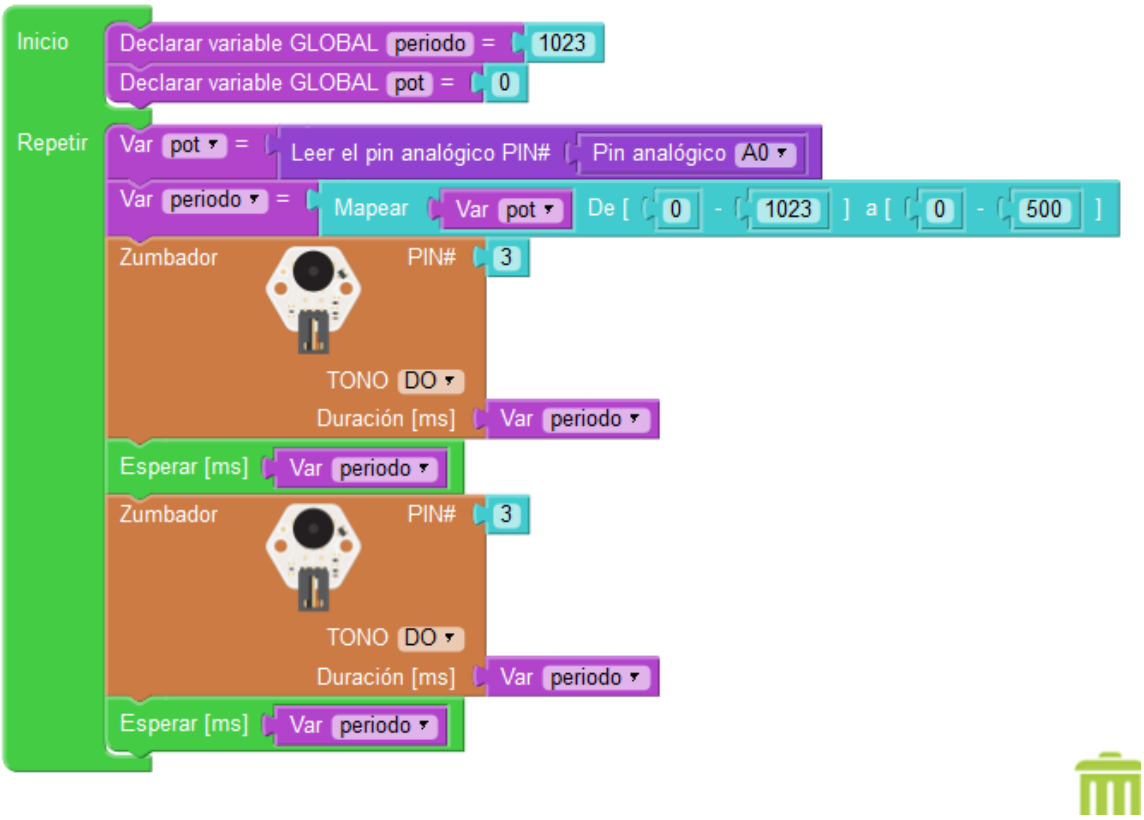
Si se quiere un sonido más continuo se puede editar el código y dejarlo así

```
    tone(3,frecuencia);
    // delay(10);
```

30. Alarma de choque

a) Hagamos un programa para evitar golpes al aparcar, por ejemplo.
   - Sonarán pitidos a intervalos más pequeños, dependiendo de la entrada del potenciómetro. Estaríamos simulando la señal que nos podría dar un sensor de ultrasonidos, por ejemplo.

SOLUCIÓN 30

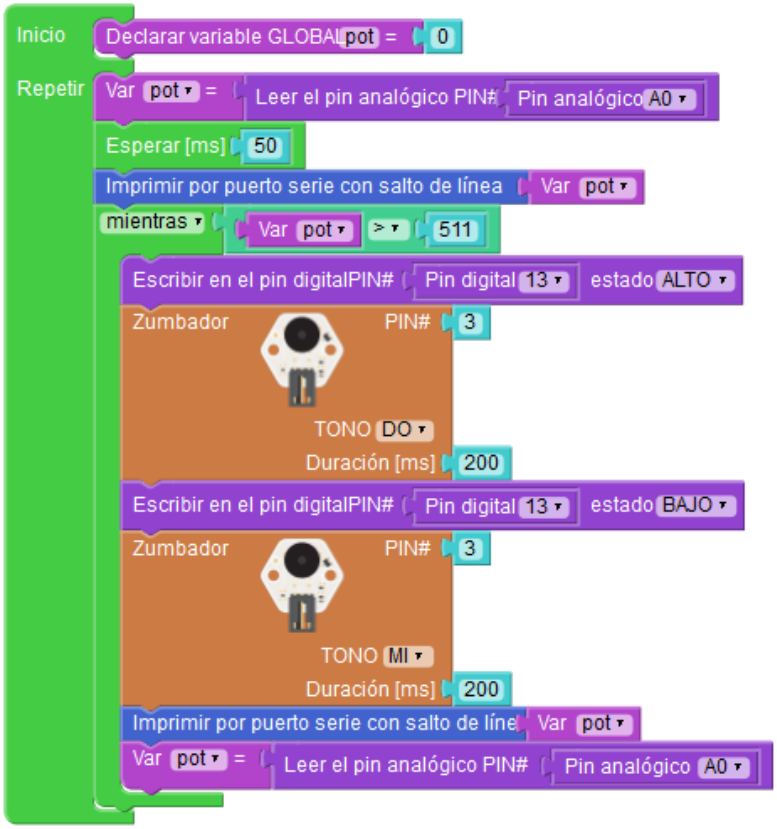



```
/***   Global variables   ***/
int periodo=1023;
int pot=0;

/***   Function declaration   ***/

void setup()
{
  pinMode(A0,INPUT);
}


void loop()
{
    pot=analogRead(A0);
    periodo=map(pot,0,1023,0,500);
    tone(3,261,periodo);
    delay(periodo);
    delay(periodo);
    tone(3,261,periodo);
    delay(periodo);
    delay(periodo);

}

/***   Function definition   ***/
```

31. Alarma de nivel

a) Hagamos un programa que dispare una alarma cuando el potenciómetro sobrepase un valor. Por ejemplo, parpadeo y pitidos.

SOLUCIÓN 31




```
int pot=0;

void setup()
{
  pinMode(A0,INPUT);
  pinMode(13,OUTPUT);
  Serial.begin(9600);
}

void loop()
{
    pot=analogRead(A0);
    delay(50);
    Serial.println(pot);

    while (pot > 511) {
      digitalWrite(13,HIGH);
      tone(3,261,200);
      delay(200);
      digitalWrite(13,LOW);
      tone(3,329,200);
      delay(200);
      Serial.println(pot);
      pot=analogRead(A0);
     }
}
```

32. Alarma de nivel adaptada

Las condiciones del entorno pueden ser muy diferentes y es muy deseable que un mismo programa pueda desenvolverse en situaciones donde los sensores den entradas en distinto rango en una ejecución y en otra.

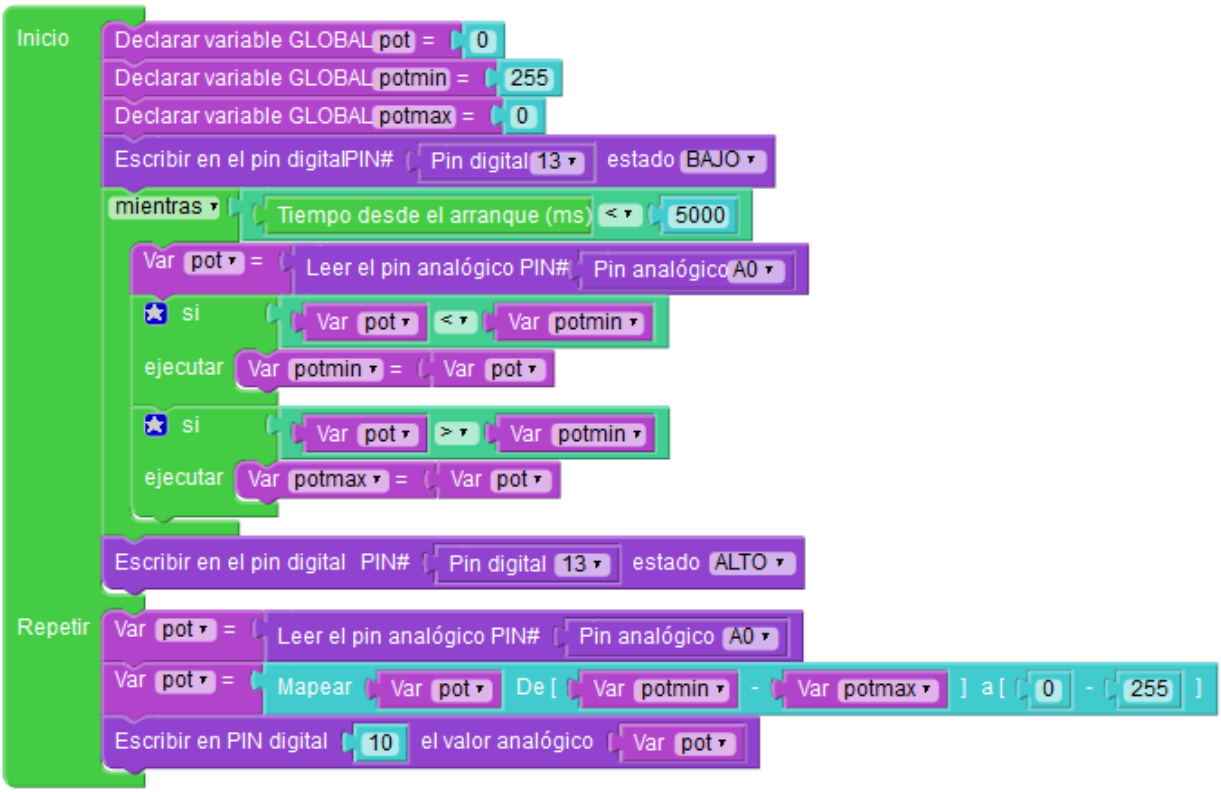
Esta es una solución elegante y automática para establecer el rango del sensor, que viene en los ejemplos del IDE.

Se establece el mínimo en 255 y el máximo en 0 y se dejan cinco segundos para que el sensor lea. Este periodo se señaliza con el LED 13

Si lee un valor mayor que cero se toma como nuevo máximo y si lee un valor menor que 255 se toma como nuevo mínimo. De esa forma los mínimos y máximo van moviéndose hasta alcanzar sus lugares.

Después se añadiría el WHILE correspondiente a la alarma.

SOLUCIÓN 32




```
/*Calibration

 Demonstrates one technique for calibrating sensor input.  The
 sensor readings during the first five seconds of the sketch
 execution define the minimum and maximum of expected values
 attached to the sensor pin.

 The sensor minimum and maximum initial values may seem
backwards.
 Initially, you set the minimum high and listen for anything
 lower, saving it as the new minimum. Likewise, you set the
 maximum low and listen for anything higher as the new maximum.

 The circuit:
 * Analog sensor (potentiometer will do) attached to analog
input 0
 * LED attached from digital pin 9 to ground

 created 29 Oct 2008
 By David A Mellis
 modified 30 Aug 2011
 By Tom Igoe

 http://www.arduino.cc/en/Tutorial/Calibration

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

```
// These constants won't change:
const int sensorPin = A0;    // pin that the sensor is attached
to
const int ledPin = 10;        // pin that the LED is attached to

// variables:
int sensorValue = 0;         // the sensor value
int sensorMin = 1023;        // minimum sensor value
int sensorMax = 0;           // maximum sensor value


void setup() {
  // turn on LED to signal the start of the calibration period:
  pinMode(13, OUTPUT);
  digitalWrite(13, LOW);

  // calibrate during the first five seconds
  while (millis() < 5000) {
    sensorValue = analogRead(sensorPin);

    // record the maximum sensor value
    if (sensorValue > sensorMax) {
      sensorMax = sensorValue;
    }

    // record the minimum sensor value
    if (sensorValue < sensorMin) {
      sensorMin = sensorValue;
    }
  }

  // signal the end of the calibration period
  digitalWrite(13, HIGH);
}

void loop() {
  // read the sensor:
  sensorValue = analogRead(sensorPin);

  // apply the calibration to the sensor reading
  sensorValue = map(sensorValue, sensorMin, sensorMax, 0, 255);

  // in case the sensor value is outside the range seen during
calibration
  sensorValue = constrain(sensorValue, 0, 255);

  // fade the LED using the calibrated value:
  analogWrite(ledPin, sensorValue);
}
```

Muy interesante el uso de la función CONSTRAINT que nos previene de que suban durante la ejecución los valores del sensor por encima del máximo medido en calibración.

SENSOR DE TEMPERATURA

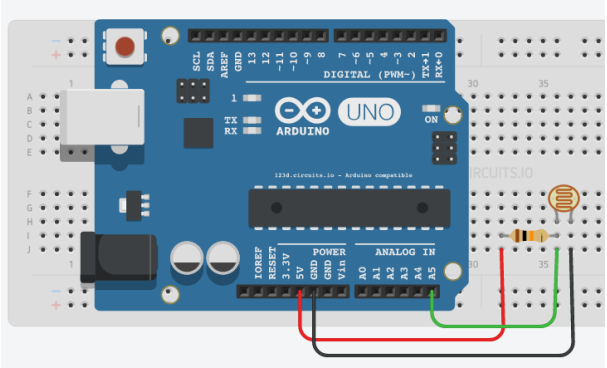
33. SENSOR DE TEMPERATURA TMP36

Este es el sensor que viene en el KIT Arduino UNO, hay otro similar y muy popular el LM35 (para el que además el Shield está preparado para conectarlo directamente, siempre que se habilite con el jumper J1).

Tienen distintos rangos y precisión. El primero más amplio, pero sólo una precisión de un grado; el segundo menos rango, pero más precisión. Ambos baratos y muy populares.

Tienen tres patillas, y son fáciles de confundir con transistores (cuidado).

Una patilla va a 5V otra a GND y la central da una señal de voltaje directamente, así **que no es necesario hacer un divisor de tensión** como en otros componentes pasivos (termistores, LDRs).



A partir de aquí podríamos hacer todo lo que hemos hecho con el potenciómetro, que está conectado como divisor de tensión, por lo que también proporciona una entrada de voltaje analógica.

Es interesante, para aplicaciones muy concretas, que con una sencilla fórmula puede transformarse el voltaje que dan en valores de temperatura.

LDR y TERMISTORES

34. LDRs y Termistores

Son componentes pasivos y, por lo tanto, tenemos que montar un circuito para que puedan proporcionar una entrada de tensión en un pin analógico.

La resistencia con la que se “comparen” debe ser similares o usar una resistencia variable que nos permita ajustar los valores de voltaje con el rango de la magnitud física a medir.

Esto es crucial cuando se hace todo por circuitería, pero con Arduino podemos tratar luego los datos y somos más elásticos en este sentido.

35. LCDs y otras hierbas

El cacharrismo es infinito y creciente.

En la práctica la mayoría de los aparatos vienen con una descripción de cómo debe hacerse el conexionado y librerías de funciones que podemos usar directamente con mucha comodidad, incluyendo los que vienen con su SHIELD.

Así que, para nuestro nivel de detalle, sólo tenemos que tratar en nuestro programa cómo son las entradas que proporcionan o cómo conseguir salidas por ellos.

MINISERVOS

36. INDICADOR DE AGUJA CON SERVO

Un servo es un motor que gira un ángulo entre 0 y 180 grados con buena precisión.

Puede ser muy interesante para

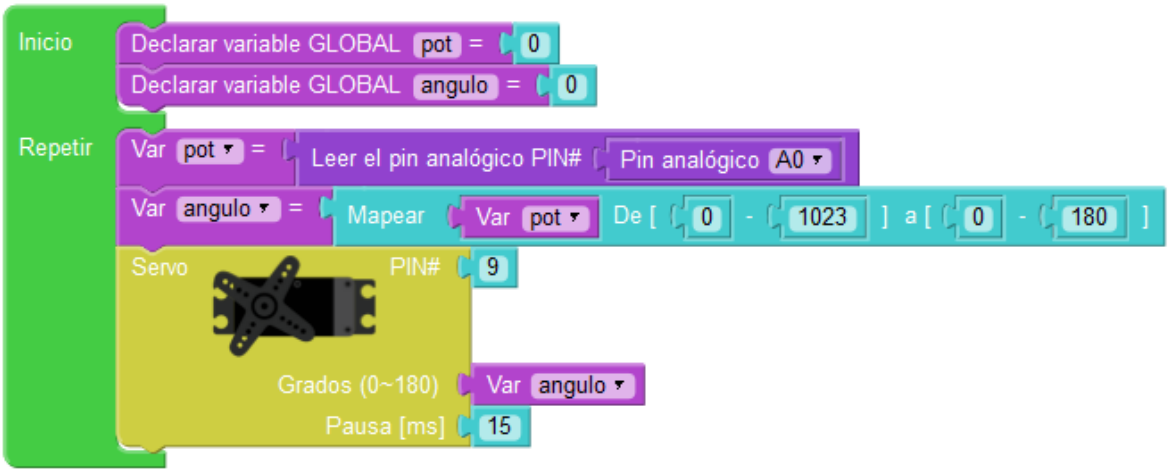
- Indicadores (mapeando valores de sensores o variables)
- Vallas
- Empujar cosas ligeras hacia rampas, etc.
- Brazos

En cualquier caso, es muy sencilla su escritura

a) Hagamos un programa que mapee un potenciómetro en los 180 grados del servo.

OJO: El uso de Servo tira de interrupciones “Timer” y dará incompatibilidades con otras funciones que usen los pines 9 y 10 con PMW. Estos pines son para Arduino UNO. Más info aquí https://www.luisllamas.es/salidas-analogicas-pwm-en-arduino/

SOLUCIÓN 36



Este es el código que genera Visualino.

Parece que crea un vector de nueve componentes y cada una sería un servo, este en particular es el 9.

```
#include <Servo.h>

Servo servos[13];

/***   Global variables   ***/
int pot=0;
int angulo=0;

/***   Function declaration   ***/

void setup()
{
  pinMode(A0,INPUT);
  servos[9].attach(9);
}

void loop()
{
    pot=analogRead(A0);
    angulo=map(pot,0,1023,0,180);
    servos[9].write(angulo);
    delay(15);

}

/***   Function definition   ***/
```

Es más habitual este código, donde miServo se instancia como un objeto de tipo Servo que puede utilizar todos los métodos (funciones) de estos. Por ejemplo, miServo.attach() para ver dónde lo conectamos, miServo.write() para ver qué ángulo se mueve, etc.

```
#include <Servo.h>

Servo miServo;

/***   Global variables   ***/
int pot=0;
int angulo=0;

/***   Function declaration   ***/

void setup()
{
  pinMode(A0,INPUT);
  miServo.attach(9);
}

void loop()
{
    pot=analogRead(A0);
    angulo=map(pot,0,1023,0,180);
    miServo.write(angulo);
    delay(15);
}

/***   Function definition   ***/
```

SERVOS DE ROTACIÓN CONTINUA

37. SERVOS DE ROTACIÓN CONTÍNUA

La lógica sería la misma, pero aquí el parámetro (también entre 0 y 180) nos da velocidad y sentido de giro, 90 sería parado.

Nos pueden servir para:

- Ruedas
- Ventiladores
- Cintas transportadoras

También existen los motores paso a paso para movimiento controlado, de nuevo tendréis que cargar sus librerías e invocar sus métodos. Lo encontraréis sencillo si habéis llegado hasta aquí.

RELÉS

38. RELÉS

El relé consigue el paso de control a potencia, desde nuestra placa podríamos conectar la televisión, la caldera, luces, motores… pero tiene el problema de necesitar bastante corriente y de mandar corriente en sentido contrario en los transitorios de encendido y apagado.

Esto no es un problema si se dispone de la circuitería adecuada (un transistor para amplificar, diodos, etc.), o, más cómodamente de los SHIELDs correspondientes.

En este último caso se trata simplemente de mandar la señal de 5V al relé, con la que activaremos o desactivaremos el aparato de potencia al abrirse o cerrarse los contactos del relé.

39. serialEvent()

Es una interesante función que transforma la filosofía con la que solemos programar en Arduino, que es secuencial.

Esta función **se pone fuera del SETUP y del LOOP** de forma que queda esperando a recibir un evento desde el puerto serie. Este evento puede llegar porque tecleemos algo, o bien accediendo a través de Bluetooth o WIFI a través de apps donde la filosofía es también menos secuencial y más de eventos.

Aquí un ejemplo:

En los comentarios puede leerse que esta función se ejecuta después de cada iteración del LOOP, así que cualquier retardo que produzca allí dejará también esperando a esta función:

“This routine is run between each time loop() runs, so using delay inside loop can delay response.”

```
/*
  Serial Event example

  When new serial data arrives, this sketch adds it to a String.
  When a newline is received, the loop prints the string and
  clears it.

  A good test for this is to try it with a GPS receiver
  that sends out NMEA 0183 sentences.

  Created 9 May 2011
  by Tom Igoe
  Modified 24 April 2013
  by Sean Alvarado

  This example code is in the public domain.

*/

String inputString = ""; // a string to hold incoming data
boolean stringComplete = false; // whether the string is complete

void setup() {
  // initialize serial:
  Serial.begin(9600);
  // reserve 200 bytes for the inputString:
  inputString.reserve(200);
}

void loop() {
  // print the string when a newline arrives:
  if (stringComplete) {
    Serial.println(inputString);
    // clear the string:
    inputString = "";
    stringComplete = false;
  }
}

/*
  SerialEvent occurs whenever a new data comes in the
  hardware serial RX. This routine is run between each
  time loop() runs, so using delay inside loop can delay
  response. Multiple bytes of data may be available.
*/
void serialEvent() {
  while (Serial.available()) {
    // get the new byte:
    char inChar = (char)Serial.read();
    // add it to the inputString:
    inputString += inChar;
    // if the incoming character is a newline, set a flag
    // so the main loop can do something about it:
    if (inChar == '\n') {
    stringComplete = true;
    }
  }
}
```

MUCHOS DISPOSITIVOS Y POCOS PINES…

Quedan muchas cosas en el tintero, como **el protocolo de comunicación I2C** que os permitirá **conectar muchos dispositivos en dos cables.** En la práctica si tenéis sensores o actuadores con esa tecnología. Deberéis cablearlos, cargar la librería correspondiente `Wire.h` , e invocar los métodos particulares de cada "cacharrito". En realidad sencillo, la dificultad se la llevó el que escribió la librería. (Gracias desde aquí).

Otra forma de tener varios dispositivos es usar **Shift Registers**. Aquí os lo dejo para el que quiera tirar del hilo.

INTERRUPCIONES

El control de las interrupciones hardware o software es muy interesante y os permite tener a Arduino "esperando eventos" o parando la programación para ejecutar una instrucción cada intervalo concreto.

PARA SITUACIONES MUY PRECARIAS…

… o para hacer pruebas…

Tenéis un simulador tanto de la electrónica como de la programación de Arduino en la web de Autodesk.

Podéis escribir código Arduino, usar el Monitor Serie, y usar los componentes en la simulación.

Muy útil

http://www.123dapp.com/circuits

PARA AMPLIAR

Para quien quiera profundizar desde los aspectos más básicos hasta lo más sofisticado y el mayor detalle tenéis un EXCELENTE tutorial en español en http://www.prometec.net/

FINALMENTE

Este es el fruto de unas cuantas horas de darse de cabezazos con hardware, software y textos… que han llegado a buen puerto gracias también a la ayuda de mucha gente y de toda la comunidad. La ciencia es una labor conjunta.

Quedan cosas en el tintero, pero no es este un documento que pretenda ser exhaustivo.

La intención de estas prácticas es que puedan ser usadas con facilidad por personas sin conocimientos previos y, sobre todo, por profesores de secundaria.

Espero que os sirvan y se agradecen comentarios que puedan mejorarlas. Iré actualizando el documento y lo tendréis a vuestra disposición en mi blog Lacienciaparatodos.wordpress.com.

Javier Fernández Panadero