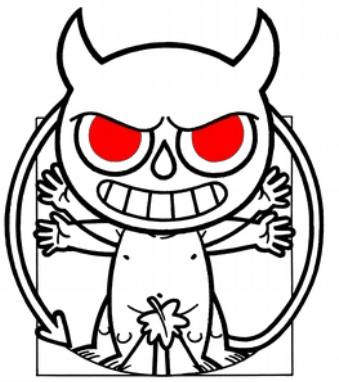
ArduPRENDE

Guía de usuario

Diableco.Solutions



Diableco.com

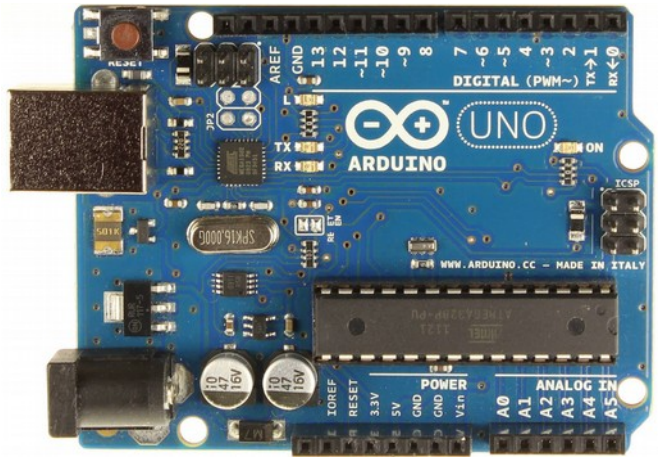
Índice

Arduino

Se trata de una plataforma open source que permite utilizar hardware de una forma rápida y estandarizada, con lo que con unos mínimos conocimientos de hardware se puede empezar a programar rápidamente.

El lenguaje de programación utilizado es una implementación de Wiring. Se podría decir que el código es C/C++ pero al tener ciertas particularidades no compila directamente como C++, aunque el IDE internamente sí convierte el código a un C++ compilable.

El entorno de desarrollo (Arduino IDE) está basado en Processing (al igual que lo está Wiring), y también es open source aunque en la web oficial de Arduino se proporciona el programa listo para descargar. El Arduino IDE incluye decenas de códigos de ejemplo integrados con los que se puede probar los componentes de hardware y librerías de código más populares. Utiliza, de forma transparente: *AVR Libc* para compilar y *avrdude* para programar el microcontrolador.



Las placas (boards) oficiales de Arduino están totalmente soportadas en el IDE ya que todas tienen un apartado concreto en la sección de Herramientas/Placa en Arduino IDE. Las placas integran lo más básico: microcontrolador y conexiones de entrada/salida. La gran mayoría también tienen la comunicación con el ordenador (USB) y un regulador de tensión (cada placa especifica qué rangos es capaz de soportar). Comparativa de diversas placas tanto oficiales como compatibles: https://learn.sparkfun.com/tutorials/arduino-comparison-guide

Las “Shield” son unas placas expansoras que permiten ampliar las posibilidades de las placas, suelen consistir en una segunda placa (sin microcontrolador) que añade sensores, módulos de comunicación o de interacción.

Arduino IDE

En la web oficial se puede descargar: http://arduino.cc/en/Main/Software

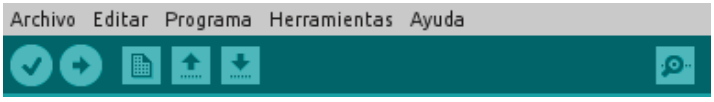
Para sistemas Windows se recomienda utilizar el instalador (*Installer*) y en Linux se recomienda instalar del repositorio oficial de la distribución que se esté utilizando, pero si el repositorio aún tiene una versión inferior a la 1.6 una vez instalado se recomienda descargar la versión correspondiente (32bit o 64bit) descomprimirla y usar esa en lugar de la del repositorio.

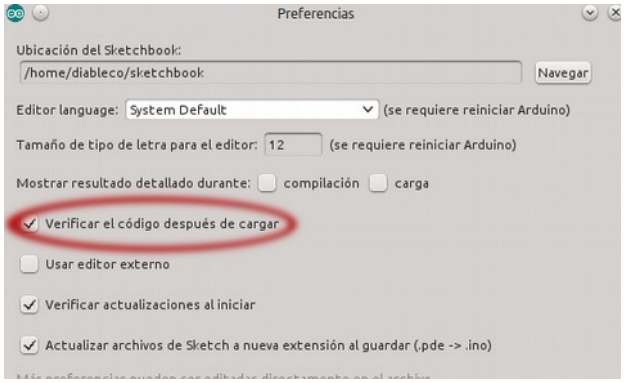
Antes de programar, es muy importante el tipo de placa y el puerto al que está conectado:

- Herramientas => Placa => Arduino Uno
- Herramientas => Puerto => El que corresponda

Para saber qué puerto es el correcto hay un sencillo truco: desconectar el USB, entrar a ese menú y ver los puertos existentes, a continuación volver a conectar el USB y el puerto nuevo es el que hay que elegir. En la parte inferior derecha del Arduino IDE aparece la placa y el puerto elegidos.

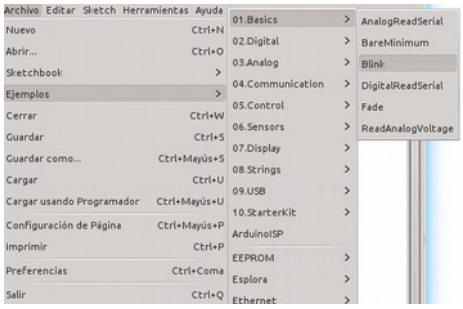
Las opciones básicas se encuentran como iconos en la parte superior: **compilar y subir** (flecha derecha), la comunicación por **puerto serie** (lupa) y botón de **guardar** (flecha hacia abajo):





Para hacer pruebas y que la subida de información a la placa Arduino sea más rápida se recomienda desactivar la verificación de código en **Archivo=>Preferencias**. Eso significa que se sube el código al Arduino y no se comprueba si fue exitosa. **MUY IMPORTANTE volver a activar** esta opción cuando se quiera programar un programa definitivo o cuando aparezcan problemas extraños.

También es muy útil la sección **Archivo=>Ejemplos** con los que se pueden probar las operaciones más sencillas de una forma muy rápida, las más destacadas: 1>Blink, 2>Button, 3>Fading, 4>ASCIITable y 3>AnalogInOutSerial.



Lenguaje C

En código C las instrucciones que se escriben se llaman "sentencias" y siempre deben finalizar en ; (que equivale al punto de un texto escrito), en un texto literario la falta del punto final no impediría su correcta lectura, pero en programación es imprescindible que esté todo escrito correctamente, por lo que la falta del punto y coma hará aparecer un error en el código (usualmente ese error aparece en la siguiente línea, así que cuando veamos un error en una línea que parezca estar bien, posiblemente se deba a la falta de “;” en la línea anterior).

El código C nos ayuda a los humanos a programar de forma más cómoda: es importante destacar que pese a que nuestros programas estén escritos en lenguaje C el compilador “traducirá” nuestro código a algo que el microcontrolador entienda (ensamblador=código máquina), y eso son unas sencillas instrucciones en unos y ceros.

Comentarios

Para poder entender mejor el código fuente usaremos frecuentemente comentarios en nuestros programas. Los comentarios son texto que ve el usuario pero Arduino ignora por completo Para ello existen 2 tipos, una línea y varias.

Comentarios de una línea:

```
int hola=0; //Inicializada la variable "hola" a cero
```

Este tipo de comentarios empieza por // y se considera comentario todo lo que venga detrás hasta un salto de línea.

Comentarios de varias líneas:

```
/* Mi primer programa
    Creado: ahora mismo
    Por: Diableco */
```

Todo el texto que se "encierre" entre /* y */ se considera comentario.

Consejo: Utiliza el comentario de 1 línea siempre que sea posible, de esta forma se puede usar el de varias líneas cuando se quiere “anular” un trozo de código (por ejemplo para hacer una prueba) sin tener que borrar ese código “anulado”.

Variables

La forma habitual de crear una variable es la siguiente:

```
int hola=0;
```

Primero se indica el **tipo del dato** "`int`" (en este ejemplo un entero), después el **nombre** de la variable "`hola`" siendo sensible a mayúsculas (es decir "hola" y "Hola" son diferentes) y opcionalmente se puede asignar un **valor inicial** a la variable.
Pese a ser opcional, es una buena práctica la de asignar un valor inicial para evitar comportamientos inesperados en nuestros programas. Del mismo modo es muy útil poner un nombre a la variable que nos sirva para identificar qué es lo que hace.
Los tipos de variable disponibles son:

- **8bit**:
  - **boolean**: sólo 2 valores posibles true/false
  - **byte**: números sin signo entre 0 y 255 (incluidos)
  - **char**: números con signo entre -128 y 127, el compilador en algunas situaciones interpretará como un carácter dando resultados inesperados
  - **unsigned char**: se recomienda utilizar “byte” en su lugar ya que son similares
- **16bit**:
  - **word**: números sin signo entre 0 y 65535 (incluidos)
  - **int**: números con signo entre -32768 y 32767, es el tipo más utilizado
  - **unsigned int**: se recomienda utilizar “word” en su lugar
- **32bit**:
  - **long**: números con signo entre -2 147 483 648 y 2 147 483 647 (dos millardos)
  - **unsigned long**: números sin signo entre 0 y 4 294 967 295 (cuatro millardos), el uso más habitual de este tipo es para almacenar el valor de la función *millis()*, que es el número de milisegundos que lleva el programa ejecutándose.
  - **float**: con signo entre $-3.4028235 \times 10^{38}$ y 3.4028235×10^{38}. En Arduino la coma flotante no es nativa y el compilador tendrá que hacer mucho trabajo, evitar utilizar este tipo para no comprometer el rendimiento de nuestros programas.

Una vez declarada la variable "`hola`" se puede consultar su valor en cualquier momento:

```
Serial.Print(hola); //Imprime valor de "hola" en el puerto serie
```

O modificar su valor:

```
hola=1; //Modificación del valor "hola" al valor fijo 1
hola=otravariable; //Ahora "hola" tiene el valor de otravariable
```

Nótese que no hay que poner "`int`" cuando se modifique su valor.

Nota: las variables sólo son accesibles dentro de su ámbito, eso significa que si creamos una variable dentro de una función, su valor no será accesible fuera de ella. Pese a ello, se pueden crear "variables globales", es decir que se pueden usar a lo largo de todo el programa, para hacer una variable global bastará con crearla al principio del código, fuera de las funciones loop() y setup().

Funciones

Son agrupaciones de sentencias y se utilizan para ejecutar "bloques" de código. Su función habitual es para no repetir un trozo de código varias veces, o para simplificar la lectura del código. Las funciones pueden recibir valores y pueden devolver valores, aunque no es obligatorio.

Las funciones que no devuelven valores son declaradas con un "`void`" delante:

```
void escribirTexto(caracteres){
     Serial.Println(); //Nueva línea de texto
     Serial.Print(caracteres);
}
```

Por ejemplo una función "`Suma()`" necesitaría recibir los 2 números y devolvería el resultado de la operación. Así que cuando sí devuelven un valor se declaran precedidas del tipo de dato que tiene dicho valor, por ejemplo:

```
int Suma(int valorA, int valorB) {
     int temporal=0;
     temporal=valorA+valorB;
     return temporal;
}
```

Para ejecutar una función se escribirá su nombre seguida de un paréntesis, si necesita valores se introducirán en dicho paréntesis, si devuelve un valor se trata como una variable, es decir se asume que Suma() tiene un valor como si se tratase de una variable:

```
totalSensores = Suma(sensor1, sensor2);
```

Cuando se “pasan” valores a una función éstos han de ser del mismo tipo que el que la función espera recibir, en nuestro ejemplo “`sensor1`” y “`sensor2`” tienen que ser de tipo entero (`int`) porque “`valorA`” y “`valorB`” están declarados como enteros dentro de la función.

Otro ejemplo, con valores preestablecidos:

```
hola = Suma(2,3); //El valor final de "hola" será 5
```

Los nombres de las funciones (al igual que ocurría con las variables) son sensibles a mayúsculas: "`MiFuncion()`" es diferente de "`mifuncion()`", "`MIFUNCION()`" o de "`MiFuNcIoN()`".

En la medida de lo posible, siempre se usará un nombre en una función que explique qué es lo que hace. En nuestro ejemplo "`Suma()`" resume perfectamente lo que hace la mencionada función.

Sentencia condicional

Es un grupo de instrucciones que se ejecutarán sólo en el caso de que se cumpla una serie de condiciones.

Por ejemplo si quisiéramos escribir en el puerto serie si un número es negativo o positivo sería (asumiendo que "número" es un valor que cambia a lo largo del programa):

```
int NUMERO; //Valores entre -32768 y 32767
//[…]Código donde NUMERO va cambiando de valor[…]
if (NUMERO>0){
      Serial.println("Número positivo.");
}
else if (NUMERO==0){
      Serial.println("Número es cero.");
}
else{
      Serial.println("Número negativo.");
}
```

Tanto el "`else if`" como el "`else`" son opcionales, pero muy utilizados. "`else if`" sirve para añadir condiciones que no cumplan el primer "`if`" y finalmente las situaciones que no se han satisfecho son contempladas en el "`else`".

Operadores de comparación

Existe una serie de operadores para cuando se quieren comparar 2 valores, el resultado de la comparación sólo puede ser `TRUE` (en caso de cumplirse) o `FALSE` en caso de no ser válida.

- == "Igual que": ambos valores serán idénticos (ojo, es diferente al "=").
- **!**= "Distinto de": los valores no son iguales (es decir, son diferentes).
- < "Menor que": el valor de la izquierda es menor que el de la derecha.
- <= "Menor o igual": similar al anterior, pero incluye el caso de que sean iguales.
- > "Mayor que": el valor de la izquierda es mayor del de la derecha.
- >= "Mayor o igual": similar al anterior incluyendo el caso de ser idénticos.

Es importante que los valores a comparar sean del mismo tipo, pues si se compara un "`int`" con un "`byte`" posiblemente sucedan comportamientos inesperados.

Operadores lógicos

Los operadores lógicos devolverán también el resultado `TRUE`/`FALSE` y son "**&&**" para el operador AND y "||"para el operador OR. Son útiles para hacer cumplir 2 condiciones, por ejemplo:

```
if ( (NUMPOSITIVO > 0) && (NUMNEGATIVO < 0) ) { TodoCorrecto(); }
```

Los paréntesis no son obligatorios, pero sí altamente recomendables para entender mejor el código.

Switch

Su funcionamiento es equivalente al anidamiento de diversos "`else if`" pero permite al humano programador crear programas más legibles. Se utiliza cuando se precisan muchos tipos de acción ante diversos valores iniciales. Pese a que Switch sólo permite comparar con valores exactos de la variable es bastante habitual dicho tipo de comparación.

Al igual que ocurre en el "`if`", si al final de las comparaciones no se han satisfecho ninguna de las comparaciones se ejecutará un bloque de código denominado "`switch`".

Su forma es:

```
switch (VARIABLE_A_ANALIZAR) {
	case 0:
		//Código cuando VARIABLE_A_ANALIZAR==0
	break;
	case xxxx:
		//Código cuando la variable tenga este valor
	break;
	case yyyy:
		//Código cuando la variable tenga este valor
	break;
	default:
		//Código ejecutado cuando ninguna condición anterior fuese válida
}
```

Como se puede ver consiste en un "`switch (VAR)`" y diferentes "`case X:`" cerrado por "`break;`", siendo imprescindible este último break; para que el siguiente "`case X:`" sea válido. Al final un "`default:`" para las situaciones que no se han satisfecho con los "`case X:`" precedentes, pese a ser no ser obligatorio puede estar vacío y no tener instrucciones asociadas aunque no se recomienda, pues se supone que "`switch`" se usa cuando se necesita que "ocurra algo" e incluso aunque se contemplen todas las posibles combinaciones es bueno tenerlo preparado para errores inesperados.

Imaginemos que disponemos de un selector que nos da 4 valores posibles (0 a 3) y queremos actuar sobre un motor en función de qué opción hemos elegido, podríamos utilizar un "`if`" para ello, pero también el "`switch`" nos permite realizar las mismas operaciones:

```
switch (BOTONERA) {
	case 0:
		MotorAdelante();
	break;
	case 1:
		MotorRetroceso();
	break;
	default:
		ApagarMotor();
}
```

Esta forma nos permitiría de forma fácil añadir nuevas funcionalidades en el código, pues con añadir "`case 2:`" no sería necesario modificar el hardware simplemente añadir unas pocas líneas nuevas de código.

Bucles

Los condicionales servían para tomar decisiones, pero cuando es necesario repetir varias veces las mismas operaciones existen los bucles que consisten en estar ejecutando el código que tengan dentro hasta que se cumpla una condición.

while

En este caso estará ejecutándose el código hasta que la condición se vuelva "`FALSE`" son muy útiles cuando queremos esperar a que "ocurra" algo antes de seguir ejecutando más programa, típicamente esperar a que un botón sea pulsado para empezar a funcionar, por ejemplo:

```
byte estadoBoton = digitalRead(pinBOTON);
while(estadoBoton != 1){
      ParpadeoLED();
      estadoBoton = digitalRead(pinBOTON); //Actualizar estado, importante
}
```

Un error común con los bucles "`while`" es no actualizar el estado de la variable usada en la condición dentro del propio "`while`" dando lugar a bucles infinitos (es decir un bucle que nunca termina y por lo tanto bloqueando el resto del código).

for

Pese a que el funcionamiento es similar al "while", el "`for`" se usa habitualmente cuando queremos repetir 'n' veces una porción de código, ya que este tipo de bucle tiene 3 "parámetros" de inicialización: inicialización, comparación e incremento:

- Variable/s: Aquí se inicializa la variable que cambia en cada iteración.
- Comparación: Cada vez que sea "`FALSE`" volverá a ejecutarse el bucle, si la comparación se volviese "`TRUE`" se terminaría la ejecución del bucle.
- Incremento: Cuando la comparación fue "`FALSE`" se cambiará el valor de la variable inicial (de esta forma, en algún momento la comparación se volverá verdadera).

En este ejemplo se mostrará por puerto serie 10 veces un texto:

```
Serial.print("Sé contar: ");
for (i=0; i<10; i++){
      Serial.print(i);
      if (i!=9) { Serial.print(", "); } //Evita la última coma
}
Serial.print(".");
```

Viéndose por consola: "**Sé contar: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.**" Nótese que la comparación se ha hecho con el número 10, pero en realidad sólo se muestra hasta el número 9. Es un error muy común con el bucle "`for`": cuando se cumpla la condición el código no se ejecuta. Para solucionar la duda de si hay que poner 11 para que muestre el número 10 es más sencillo utilizar "<=" en lugar del "<", de esta forma está contemplado el valor escrito.

Programación

En esta sección veremos los elementos básicos del manejo de Arduino que consisten en entrada/salida tanto digital como analógica. También se verá el puerto serie para poder ver lo que ocurre dentro del microcontrolador sin necesidad de ninguna interfaz (como una pantalla).

Básico: LED

ArduPRENDE tiene 2 LED normales y un tercer LED RGB (en su interior tiene 3 LED):

- PIN Arduino: D7 – LED izquierdo (color amarillo).
- PIN Arduino: D8 – LED derecho (color verde).
- LED RGB: Pines Arduino: D5 (rojo), D6 (verde) y D9 (azul).
- PIN Arduino D13 – En la placa del Arduino Uno hay otro LED, marcado como "L".

Si en Arduino IDE usamos el ejemplo Básico llamado "**Blink**" haremos que el LED del Arduino Uno parpadee con una frecuencia de medio hercio, cambiar el valor 13 por el 7 o el 8.

Las funciones esenciales para trabajar con los LED son: `pinMode()` y `digitalWrite()`.

Para indicar que un PIN concreto es una salida se utiliza `pinMode()` y ha de ir en el Setup():

```
pinMode(7, OUTPUT);
```

Siendo el primer valor el número de PIN y el segundo el tipo (OUTPUT=Salida).

Una vez establecido como salida, con digitalWrite(), se puede poner tensión alta o tensión baja. Cuando está conectado un LED eso significaría "encender" y "apagar" la luz que emite:

```
digitalWrite(7, HIGH); //Encender LED amarillo
digitalWrite(7, LOW); //Apagar LED amarillo
```

Jugando con los LED

1. Cambia la frecuencia de parpadeo del LED a 2Hz (cambiar el 13 por 7 u 8).
2. Haz que, al menos, 3 LED se enciendan de forma consecutiva. Ahora simula un semáforo.
3. Escribe tu nombre en código morse. Ayuda <https://duckduckgo.com/?q=morse+Hola> (cambia "Hola" por tu nombre).

Básico: Botón

ArduPRENDE viene con 2 botones pulsadores, situados en los extremos. Su estado puede se leído en nuestros programas, su distribución es:

- Pulsador izquierdo – PIN Arduino: D2
- Pulsador derecho – PIN Arduino: D4

Para probar su funcionamiento usaremos el ejemplo Digital "**Button**" de Arduino IDE y cambiaremos el valor de la variable "buttonPin" a un 2 o un 4 para hacerlo corresponder con alguno de nuestros botones.

Para poder utilizar los botones las funciones básicas son `pinMode()` y `digitalRead()`.

Primero ha de configurarse, y para ello se introducirá dentro de Setup():

```
pinMode(2, INPUT);
```

Siendo el primer valor el número de puerto, mientras que el segundo valor indica que es de tipo entrada digital (INPUT=entrada).

Una vez configurado el puerto como entrada, se podrá consultar el valor mediante:

```
estadoPULSADOR = digitalRead(2);
```

La función puede dar los valores "`LOW`" o "`HIGH`" y se recomienda guardar el valor en una variable intermedia y no utilizarlo directamente (si se utiliza no pasa nada).

IMPORTANTE: Nunca poner los pines de los botones (2 y 4) como salida digital, podría dañar el microcontrolador.

Jugando con los botones

1. Haz que un botón encienda el LED (D7 ó D8) y el otro botón lo apague.
2. Repite el anterior ejercicio, pero ahora el botón ejecutará una función que actúa en el LED.
3. Un botón enciende consecutivamente los LED (y los mantiene encendidos), el otro botón apagará progresivamente los LED (D7, D8 y D9).

Puerto serie

El microprocesador del Arduino Uno lleva un puerto serie (UART o USART son otras denominaciones) en los PIN D0 y D1. Pero además utiliza esta comunicación con el ordenador mediante un segundo microprocesador, eso significa que podremos “hablar” con el ordenador y/o utilizar componentes externos que usen puerto serie pero ambas comunicaciones interferirán entre ellas, es decir, en algunas ocasiones será necesario desconectar el dispositivo externo para poder reprogramar el Arduino Uno. Para leer lo que ocurre en el puerto serie, en Arduino IDE se encuentra en “Herramientas // Monitor serie” (o en el icono de lupa de la parte superior derecha).

Para iniciar las comunicaciones del puerto serie se usará la siguiente función dentro del setup():

```
Serial.begin(9600);
```

Siendo el número la velocidad en baudios, los posibles valores son: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600 o 115200 (9600 es el más común).

En el Arduino Uno hay un par de LED marcados como RX/TX se encienden cuando hay comunicación del puerto serie (ya sea con el ordenador o con un dispositivo) y eso incluye la carga de nuevos programas desde el Arduino IDE.

Las funciones principales son las siguientes:

```
Serial.print(variable);
Serial.print("Texto fijo");
Serial.println("Texto fijo, pero al final añade una nueva línea");
```

Para probar que todo funciona correctamente abriremos el ejemplo “04.Communication // ASCIITable” y al pulsar en el botón del puerto serie (situado en la parte derecha) veremos un listado de caracteres con su valor decimal y binario correspondiente.

Nota: Si en la consola del puerto serie vemos caracteres extraños significará que la velocidad de comunicación no es la correcta, asegúrate de que `Serial.begin(xxxx)` y la velocidad en baudios coinciden.

Charlando con Arduino

1. Ver el estado de un botón. Ayuda: Usar ejemplo “01.Basics // DigitalReadSerial” (cambiando variable “buttonPin” al valor 2 o el 4).
2. Muestra por el puerto serie del USB cuántas veces han sido pulsados los botones.
3. Comunícate con otra placa utilizando el puerto serie, recuerda que para que funcione hay que conectar un RX de una con un TX de la otra y viceversa (además de unir GND).

Puertos analógicos

El microcontrolador del Arduino Uno tiene 10bit de resolución en sus puertos analógicos y trabaja a 5V. Eso significa que las mediciones se harán entre 0 y 5V divididas en 1024 “trozos”, en otras palabras que seremos capaces de detectar cambios de tensión de unos 5mV.

El Arduino Uno tiene 6 puertos analógicos denominados A0 hasta A5. Como A4 y A5 se pueden utilizar para el I²C se recomienda dejarlos libres en la medida de lo posible.

La función para leer datos analógicos, `analogRead()` devuelve un número entre 0 y 1023:

```
int valor;
valor = analogRead(analogPin);
```

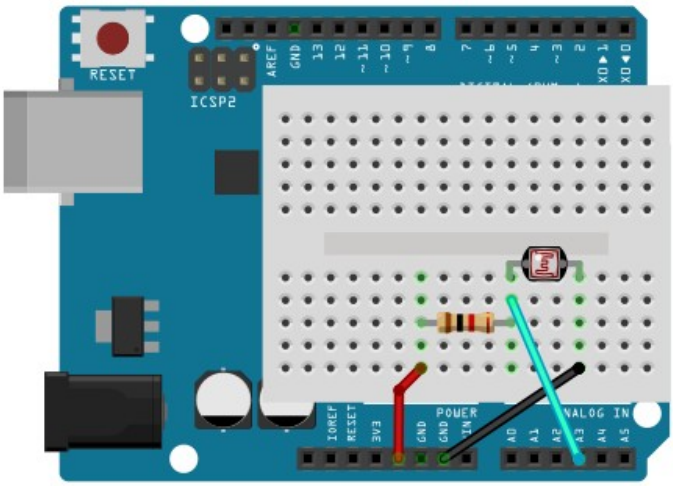
Fotorresistencia (LDR)

También conocidas como LDR por sus siglas en inglés consiste en un material semiconductor que varía su resistencia en función de la luz que incide sobre su superficie lo habitual es que sean unas pocas decenas de ohmnios cuando está incidiendo mucha luz a un valor muy elevado (del orden de megaohmios) cuando está en un lugar oscuro. No tienen polaridad así que no importa la posición de conexión, pero el montaje requiere una resistencia para evitar problemas.

Con el siguiente ejemplo se verá en el puerto serie el valor de la fotorresistencia:

```
int pinLDR = A2;
int valorLDR = 0;
void setup()
{   Serial.begin(9600);   }

void loop()
{
      valorLDR = analogRead(pinLDR);
      Serial.println(valorLDR);
      delay(250);
}
```



Con mucha luz (flash del móvil o linterna) se estará cerca de 0 y para mucha oscuridad (tapándolo) serán valores cercanos a 1023.

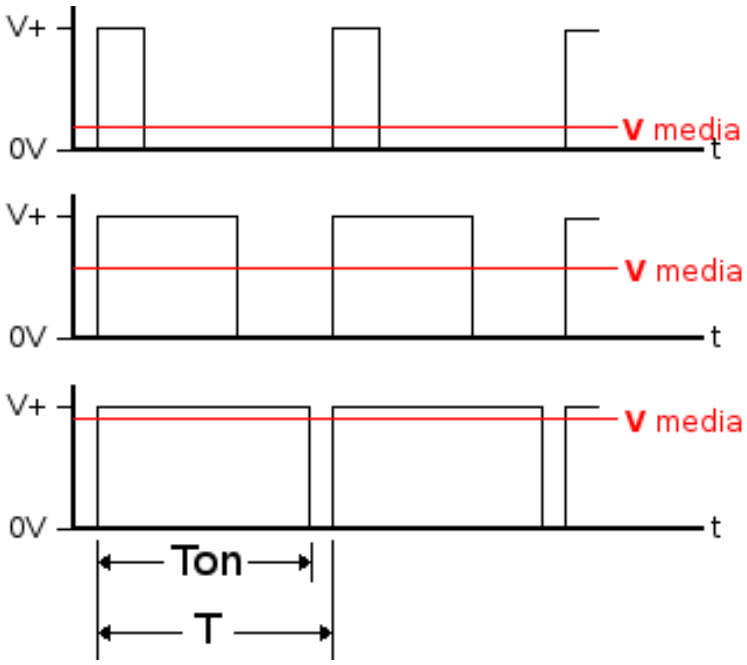
La “visión” de la electrónica

1. Enciende el LED amarillo (D7) cuando esté oscuro y el LED verde (D8) cuando exista luz (es decir si el valor es superior o inferior a 512 enciende uno u otro LED).
2. Repetir pero ahora usando los 5 LED disponibles: si es inferior a 100 encender D7, si supera el valor 300 encender D5, para 500 encender D6, en 700 encender D9 y mayor a 900 encender D8 (recuerda apagar los LED para el caso de mucha luz a oscuridad).
3. Haz que un LED parpadee en función de la luz que reciba la LDR. Ayuda: ejemplo 03.Analog=>AnalogInput (cambiando `sensorPin` por el que corresponda y `ledPin` por 5, 6, 7, 8 ó 9).

Salida Analógica: PWM

PWM: Viene de las siglas en inglés de Modulación por Ancho de Pulso. Se utiliza para controlar la potencia que se entrega a la carga (en nuestro caso la intensidad del LED RGB y el zumbador).

Debido a que la tensión de alimentación no se puede variar se alimentan los dispositivos con un PWM que consiste en pulsos de diferente ancho (modulación) que al tener mayor ancho generan una tensión media mayor. Y al ser más estrechos se tiene una tensión media inferior. De esta forma, alimentando la carga **con una tensión media variable se consigue variar su comportamiento de forma proporcional**.



Los casos extremos serían no tener un pulso, es decir que la media sean 0V y que no exista ningún "hueco" siendo ese valor máximo la tensión de alimentación (5V).

En la imagen se muestran 3 ejemplos de cómo varía la tensión media en función del "ancho" del pulso, en la placa habrá 5V de V+.

Conviene destacar que son datos digitales, pero desde el punto de vista de la carga (zumbador o el LED RGB) es una variable analógica pues "trabajan" con el valor medio del resultado del PWM, es decir, un valor medio analógico.

Para conseguir una salida de PWM se actúa sobre *Ton*, es decir el tiempo que estará la tensión funcionando. El valor será entre 0 y 255:

```
int numeroPIN=5; //LED rojo del RGB
byte valorPWM=150;
analogWrite(numeroPIN, valorPWM);
```

Ejercicios analógicos

1. Haz que el LED de Arduino (D13) reciba un PWM de 255 a 0, ¿qué ocurre con la intensidad lumínica?. Dicho LED está en la placa del Arduino Uno .
2. Manteniendo pulsado un botón se sube el valor del PWM al zumbador (D10), con el otro botón se baja el valor. Repetir pero ahora cambia el valor del PWM en los 3 pines del RGB (mostrará un color blanco), es decir, introducir el mismo valor en: D5, D6 y D9.
3. Haz que el color blanco del RGB sea más intenso en función del valor de la LDR (es decir si está oscuro el blanco será mínimo y si está muy iluminado alumbrará al máximo).

Zumbador

Se trata de un par de elementos: un electroimán y una lámina de acero, cuando se hace pasar una corriente eléctrica por la bobina del electroimán produce un campo magnético que hace vibrar la lámina de acero produciendo un tono agudo. En inglés se los conoce como "*buzzer*" ya que originalmente estaban construidos como una campana que vibraba pero no tenía badajo y generaban un sonido áspero.

En función de la tensión que reciba el zumbador producirá un tono diferente, por lo que si se utiliza el PWM para cambiar la tensión se pueden producir diferentes tonos. El las ondas de sonido no son más que vibraciones en la presión del aire, por ejemplo la nota musical "Fa" en la octava prima se corresponde con un tono de unos 349Hz, es decir, que el zumbador tendría que vibrar 349 veces por segundo para reproducir dicha nota.

El zumbador no está conectado directamente a la salida del Arduino, hay un **transistor** que amplifica esa señal, de esta forma se protege la salida del Arduino. A la hora de programar no hay diferencia entre tener o no el transistor. Para su manejo, el código es idéntico a una salida analógica:

```
int numeroPIN=10; //PIN del zumbador
byte valorPWM=150;
analogWrite(numeroPIN, valorPWM);
```

Transistor

Se trata de un dispositivo electrónico semiconductor con 3 "patas" denominadas base, colector y emisor. Su funcionamiento básico consiste en un circuito gobernado por la base. En otras palabras, la unión colector/emisor es un circuito abierto (no pasa la corriente) según la señal que esté recibiendo la base. Cuando se utiliza como amplificador permite que en la unión colector/emisor circule una corriente más elevada de la que circularía por la base.

Aunque hay una función que ayuda a utilizar el zumbador: `tone(numeroPIN, Frecuencia)` y `noTone(numeroPIN)` para apagarlo. Esta función permite poner la frecuencia en hercios:

```
int frecuenciaLa=349;
tone(10, frecuenciaLa);
delay(250);
noTone(10);
```

Opcionalmente se puede escribir un segundo parámetro que sería la duración del tono, quedando como `tone(numeroPIN, Frecuencia, TiempoEnMilisegs)` El ejemplo anterior quedaría de la siguiente forma:

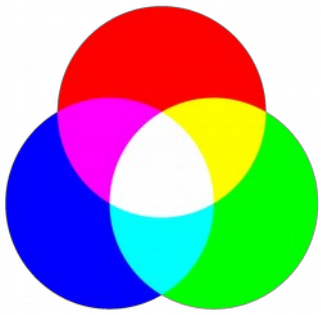
```
tone(10, 349, 250); // Equivalente al ejemplo anterior
```

Gritando con Arduino

1. Haz que el ejemplo "10.StarterKit=>p06_LightTheremin" funcione al pulsar uno de los botones y que deje de funcionar al pulsar el otro. Recuerda cambiar: `ledPin` por 8, el `analogRead` por el que corresponda a la LDR y el primer número (el 8) de la función `tone()` tiene que ser un 10.

2. Modifica la melodía del ejemplo "02.Digital=>toneMelody", Recuerda cambiar en la función `tone()` y en la `noTone()` el número 8 por el correcto, el número 10.

LED RGB



Cualquier pantalla LCD tiene cientos de miles de estos dispositivos pero de un tamaño minúsculo. Se trata de un dispositivo que en su interior hay 3 LED de colores. El funcionamiento se basa en la cantidad de luz que emite cada uno de los 3 colores.

Las siglas RGB provienen de las iniciales (en inglés) de los 3 colores que hay en su interior. Siendo dichos colores el rojo, verde y azul. Al combinar esos 3 colores de luz primarios se obtienen los diferentes colores, por ejemplo al mezclar el azul y el verde se obtiene el cian o con azul y rojo sale magenta. Cuando los 3 tienen la misma intensidad será el color blanco.

Como hemos visto una forma eficaz de obtener valores medios de tensión es mediante PWM y así es como se obtienen los diferentes colores. La conexión es la siguiente:

- R=Rojo – PIN Arduino: D5
- G=Verde – PIN Arduino: D6
- B=Azul – PIN Arduino: D9

Cabe destacar que la conexión de los LED es de cátodo común y eso implica que su funcionamiento es a la inversa, eso se traduce en que un valor de PWM igual a cero la luminosidad será la máxima y para un valor de 100% del PWM, que en Arduino significa un `analogWrite()` de valor 254, su luminosidad será mínima (concretamente estarán apagados).

El código será igual que para manejar un PWM, por ejemplo mostremos varios colores:

```
analogWrite(5, 100); //Color Rojo, casi a media intensidad
delay(500);
analogWrite(6, 100); //Color amarillo (pues el rojo sigue a 100)
delay(500);
analogWrite(9, 100); //Color blanco (rojo y verde siguen a 100)
delay(500);
```

Colorines del Arduino

1. Pon los 3 colores a la mínima intensidad pues de esta forma se pueden ver los 3 LED que componen el LED.
2. Muestra todas las intensidades de luz que permite cada uno de los 3 colores. Consejo: utiliza un bucle `for()` para simplificar la tarea.
3. Haz que se vea la siguiente secuencia de colores: rojo, verde, azul, cian, magenta, amarillo y blanco. Consejo: Usa un valor intermedio de intensidad en el PWM un valor de 200.

I²C – Wire

Para utilizar el bus I²C hay un premisa que se puede resumir en "**uno escribe y el resto leen**".

Su nombre proviene de Inter-Integrated Circuit y fue creado originalmente por Philips Semiconductors en 1982 y publicada la especificación para su uso por otros fabricantes en 1992.

Consiste en un bus de 2 líneas (Reloj y Datos) por el que viajan paquetes de bits (8 en el caso de Arduino). Para diferenciar cada uno de los dispositivos, en Arduino, se utilizan 7bit (eso significa que puede haber hasta 127 dispositivos diferentes) siendo el último bit el que determina si el maestro va va a escribir o leer en el bus. La hoja de características del dispositivo que conectemos nos dirá cuál es la dirección del mismo y si es necesario enviar comandos así como el formato de los datos que nos dará dicho dispositivo, en Arduino utilizaremos la librería Wire nos facilitará enormemente la tarea. Para conocer más sobre el procotolo I²C se recomienda este tutorial (en inglés): https://learn.sparkfun.com/tutorials/i2c

Arduino Uno tiene las 2 líneas del BUS en los pines A4 y A5:

- A5: **SCL** (línea del reloj).
- A4: **SDA** (línea para los datos en serie).

Cuando se quiera utilizar la librería Wire, en nuestros programas hay que añadir un fichero antes de la función setup():

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

Y dentro de la función `setup()` hay que inicializar la comunicación en el puerto, como parámetro se introducirá el número (entre 1 y 127) que usaremos como esclavo. Si no se proporciona ningún número, el Arduino será el maestro del bus I²C.

```
Wire.begin(direccion); //entre 1 y 127, vacío para ser Maestro
```

Desde este momento se pueden usar las funciones correspondientes al I²C:

```
requestFrom()
beginTransmission()
endTransmission()
write()
available()
read()
onReceive()
onRequest()
```

Se recomienda visitar la web oficial: http://www.arduino.cc/en/Reference/Wire

Para los 2 siguientes ejemplos (se encuentran en el menú Wire de los ejemplos de Arduino IDE) se requieren 2 Arduinos pues uno será el maestro y el otro el esclavo (importante: unir también GND).

Ejemplo Wire: master_reader / slave_sender

Se encuentran en Archivo=>Ejemplos=>Wire=>master_reader y slave_sender

En este ejemplo el Arduino que hace de…

- …maestro pide 4 byte al esclavo número 2.
- …esclavo responde con 4 byte, por ejemplo: “hola”.

Arduino MAESTRO – En el setup()

```
Wire.begin(); //Será el maestro en el bus I²C
```

Arduino MAESTRO – En el loop()

```
Wire.requestFrom(2, 4);       //Se piden 4 byte al esclavo número 2
while (Wire.available())      //Esperando a que termine
{
      char c = Wire.read();   //Recibir cada byte como un "char"
      Serial.print(c);        //Mostrar resultado por consola
}
```

Arduino ESCLAVO – En el setup()

```
Wire.begin(2);  //Unirse al I²C como esclavo número 2
Wire.onRequest(funcionDeRespuesta); //Estará fuera del loop()
```

Arduino ESCLAVO – En el loop()

Vacío, no requiere código.

Arduino ESCLAVO – Fuera de loop, (por ejemplo al final)

```
void funcionDeRespuesta()
{
      Wire.write("hola"); //Cuando Maestro pida, se responde con 4byte
}
```

Ejemplo Wire: master_writer / slave_receiver

Se encuentran en Archivo=>Ejemplos=>Wire=>master_writer y slave_receiver

En este ejemplo el Arduino que hace de…

- …maestro inicia una transmisión de datos al esclavo 4, le enviará 5 bytes de texto y al final un byte de tipo entero (`int`).
- …esclavo, al recibir una orden del maestro empieza a leer bytes en modo char (mientras los recibe los va enviando al puerto serie), el último byte lo lee como un número entero (`int`).

Arduino MAESTRO – En el setup()

```
Wire.begin(); //Maestro
```

Arduino MAESTRO – En el loop()

```
Wire.beginTransmission(4);    //Enviar datos al nº4
      Wire.write("x es ");    //Enviar 5 bytes
      Wire.write(x);          //Enviar 1 byte
Wire.endTransmission();       //Finalizar transmisión
x++;  //Incrementar valor
```

Arduino ESCLAVO – En el setup(), es igual que el otro ejemplo, sólo cambia el número

```
Wire.begin(4);  //Unirse al I2C como esclavo número 4
Wire.onRequest(funcionDeRespuesta);
```

Arduino ESCLAVO – En el loop()

Vacío, no requiere código

Arduino ESCLAVO – Fuera de loop, (por ejemplo al final). Cuando el maestro llame, se leen todos los datos que él nos envíe.

```
void funcionDeRespuesta()
{
      while (1 < Wire.available()) //Leer todo, salvo el último
      {
            char c = Wire.read();   //Recibido como carácter (char)
            Serial.print(c);        //Mostrar dato
      }
      int x = Wire.read();          //El último leído como entero (int)
      Serial.println(x);            //Mostrar dato
}
```

Preguntas frecuentes

Al abrir el Arduino IDE no conecta con el Arduino Uno

Posiblemente el número de puerto sea diferente a la última vez, revísalo.

El puerto no se muestra en Tools//Board

Cierra todas las ventanas de Arduino IDE, desconecta el Arduino del ordenador, espera unos segundos y vuelve a conectar el USB, finalmente abre el IDE y comprueba que ahora sí se muestra.

¿Cómo averiguo a qué puerto está conectado?

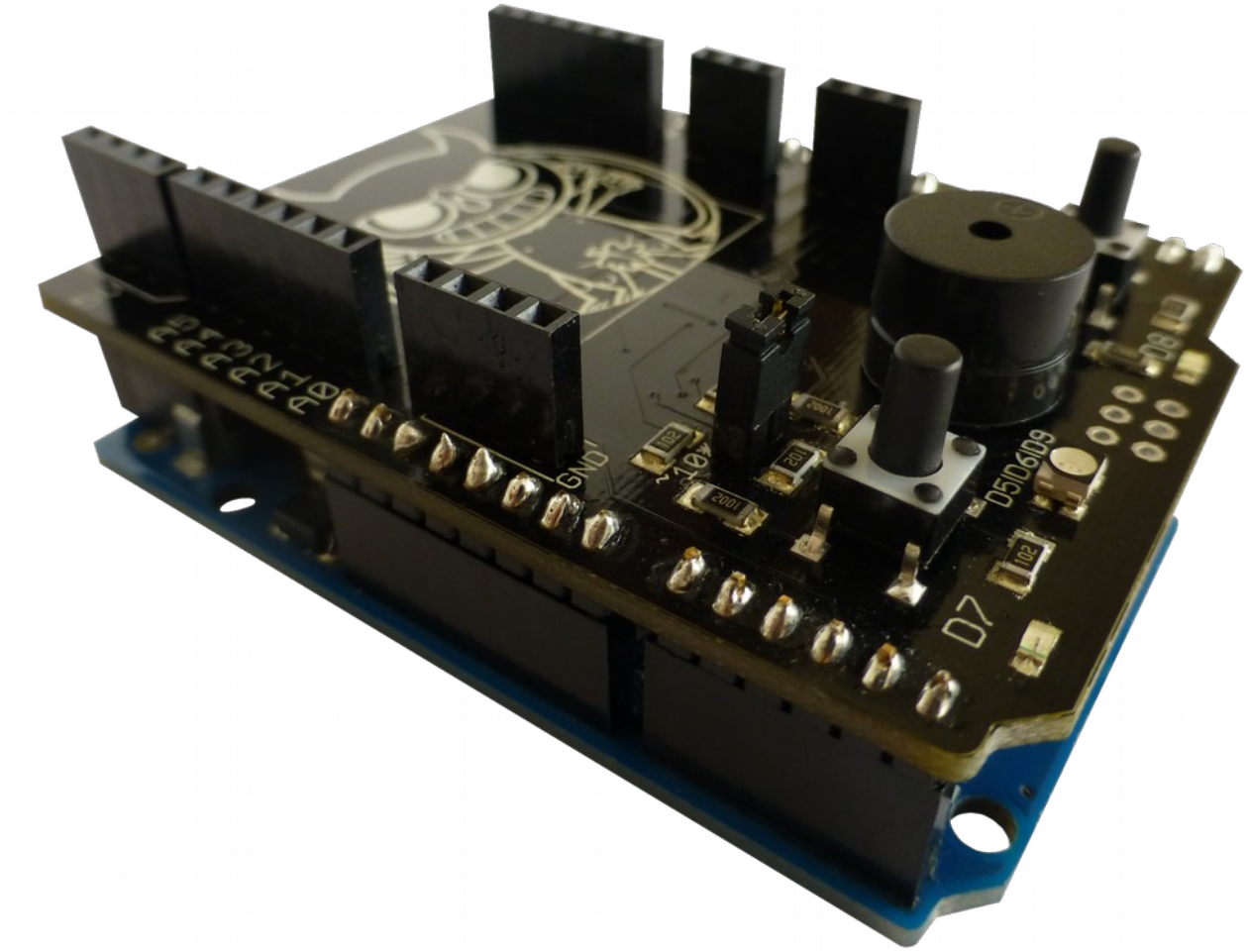
En la mayoría de portátiles ya hay puertos COM y como la única información es el número la forma más rápida es mirar el menú antes de conectarlo y volver a mirar cuando se conecte, el nuevo puerto será donde está conectada la placa. En Windows, la otra forma es ir a “Dispositivos e impresoras”, pulsar en Propiedades del dispositivo y en la pestaña “Hardware” se puede ver el puerto COM asociado al final del nombre del dispositivo.

¿Hay que expulsar de forma segura el USB?

No, pues al tratarse de un microcontrolador su memoria no se borrará ni se corromperá. Aunque se recomienda encarecidamente que no se desconecte el USB durante el breve periodo de tiempo en el que se está programando la placa.

Correspondencia de conexiones

¿Qué hay?	Arduino IDE	Tipo	NOTAS
RX	0	I/O digital + comunicaciones	Comunicación serie: RX
TX	1	I/O digital + comunicaciones	Comunicación serie: TX
Botón izquierdo	2	Entrada digital	Siempre como entrada
I/O D3	3	PWM + I/O digital	
Botón derecho	4	Entrada digital	Siempre como entrada
RGB rojo	5	Salida PWM	Siempre como salida
RGB verde	6	Salida PWM	Siempre como salida
LED amarillo	7	Salida digital	Siempre como salida
LED verde	8	Salida digital	Siempre como salida
RGB azul	9	Salida PWM	Siempre como salida
Zumbador	10	I/O digital + PWM	Quitar jumper para usar I/O
MOSI	11	PWM + I/O digital + comunic.	Comunicación SPI: MOSI
MISO	12	I/O digital + comunicaciones	Comunicación SPI: MISO
SCK	13	I/O digital + comunicaciones	Comunicación SPI: SCK
I/O A0	A0	I/O analógica	
I/O A1	A1	I/O analógica	
I/O A2	A2	I/O analógica	
I/O A3	A3	I/O analógica	
I/O A4	A4	I/O analógica + comunicaciones	Comunicación I^2C: SDA
I/O A5	A5	I/O analógica + comunicaciones	Comunicación I^2C: SCL
5V		Alimentación, parte positiva	
GND		Alimentación, parte negativa	También llamado “masa”


GND
D5D6D9
D7