Sistemas de Tempo Real: Microcontroladores
2013/14

Práctica 1: Primeros pasos con Arduino

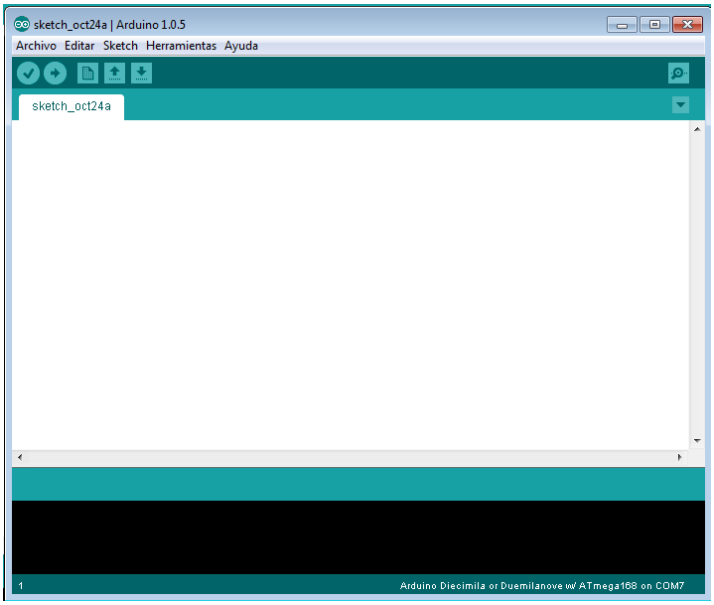
Objetivo

Llevar a cabo un primer contacto con el software/hardware de la plataforma Arduino. En concreto, se instalará el IDE y los drivers del Arduino, se aprenderá a manejar los pines digitales y se hará uso del puerto serie.

Instalación del IDE y los drivers

Antes de conectar el Arduino al PC es necesario instalar el entorno de desarrollo del mismo en los PCs del laboratorio. Para ello, en Windows 7, hay que seguir los siguientes pasos[1]:

1. Dirgirse a http://arduino.cc/en/Main/Software
2. Descargar la última versión disponible en .zip[2].
3. Extraer el .zip en un directorio donde tengamos permisos (e.g. en el escritorio).
4. Finalmente, verificar que no hay ningún problema al ejecutar el IDE (ejecutar "Arduino.exe"). Debería mostrarse una pantalla como la siguiente:



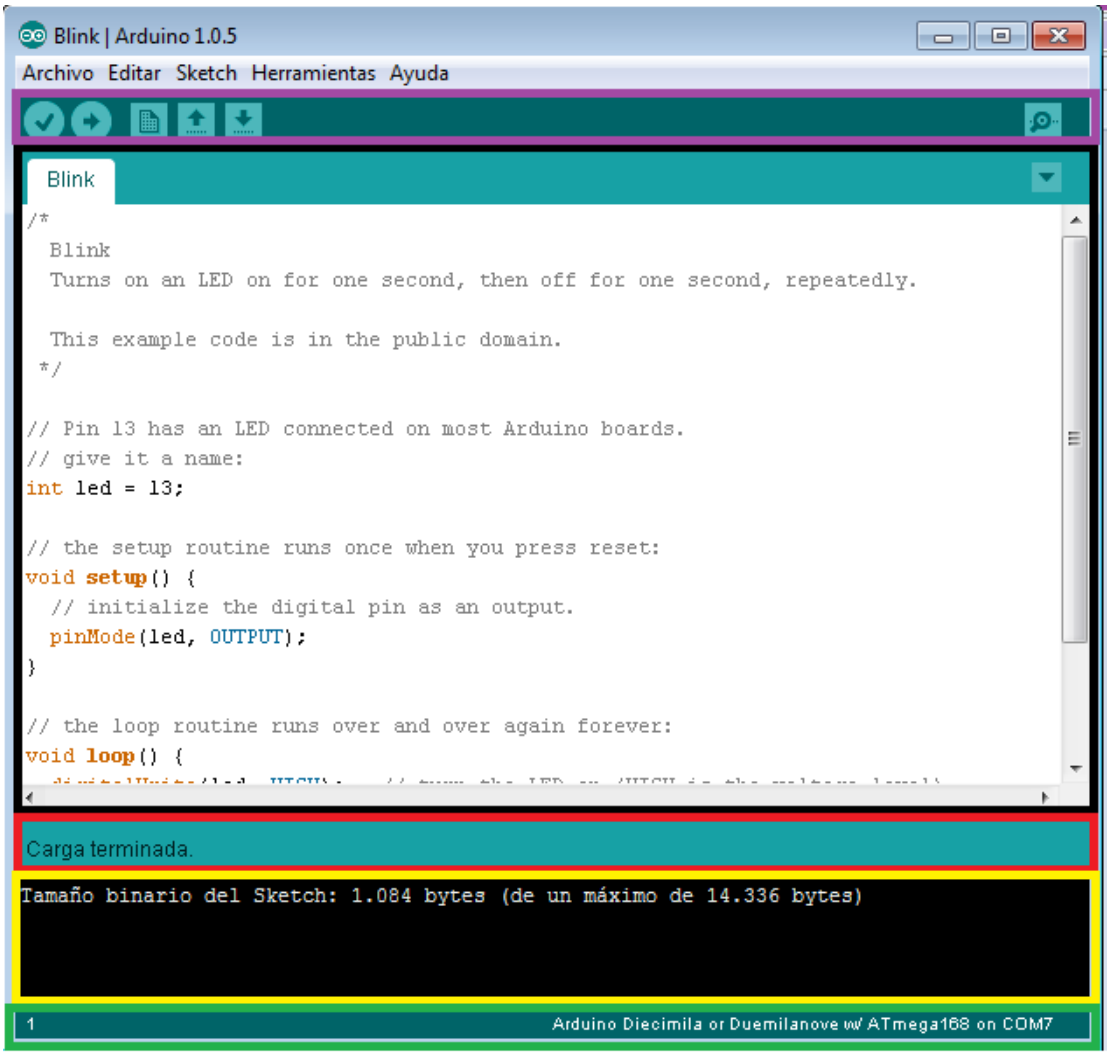
[1] Antes de proceder a la instalación es preciso verificar que tenemos al menos 360 MB de espacio libre.

[2] A la fecha de redacción de esta práctica, sería la versión 1.0.5 (no descargar la BETA).

Una vez funcionando el IDE, podemos conectar la placa Arduino a un puerto USB. Las placas Diecimila y Duemilanove **no requieren de drivers**, deberían ser detectadas automáticamente en Windows 7[3].

Estructura del IDE

El entorno de desarrollo de Arduino se compone de seis áreas, que, de la parte superior a inferior son:



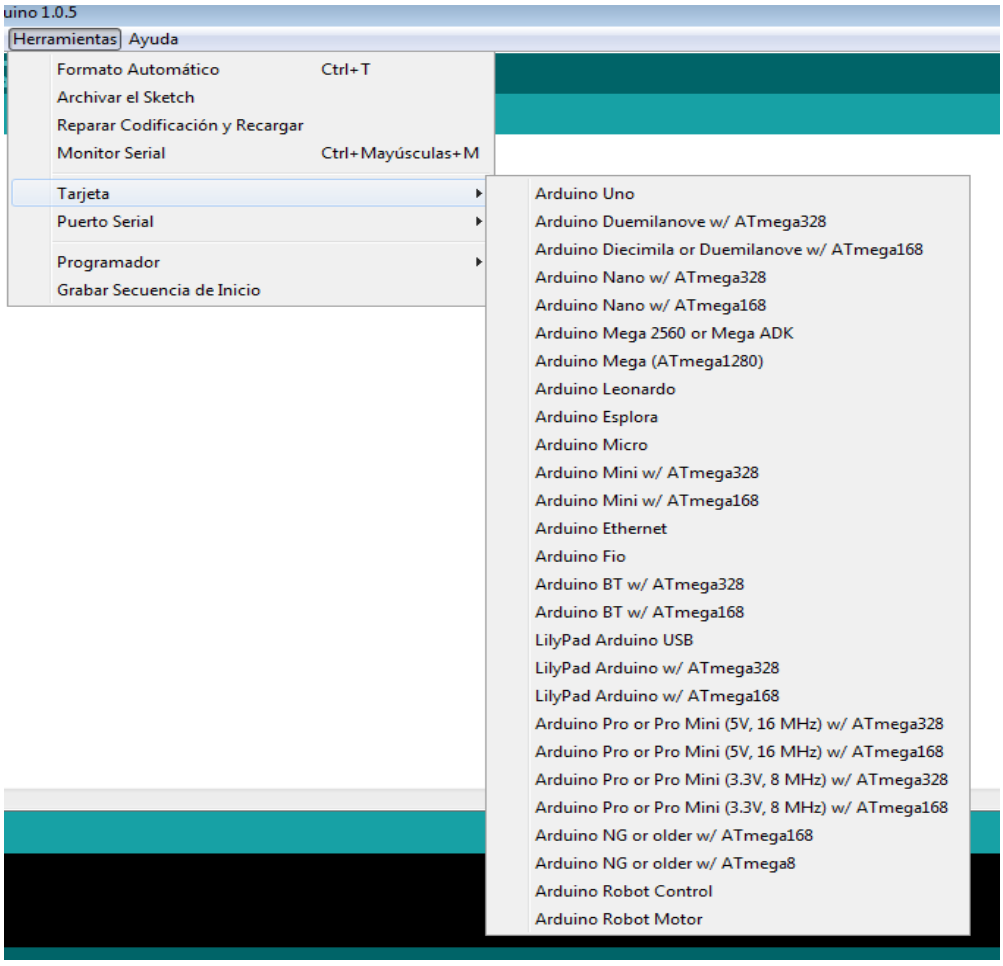
- La barra de menús principal.
- La zona de iconos de acceso rápido. Se recurrirá a ella principalmente para compilar y subir el código al Arduino.
- Dentro del rectángulo negro de la imagen anterior, se encuentra el área del código, que será donde se lleve a cabo la programación.

[3] En el caso de utilizar otra placa o SO puede ser necesario instalar los drivers antes de conectarla al ordenador.

- En la parte inmediatamente inferior (rectángulo rojo) está la barra del estatus de compilación/carga.
- Justo a continuación se encuentra el área de notificaciones (rectángulo amarillo).
- Finalmente, en la zona inferior está la barra de estatus de conexión, la cual indica la configuración hardware y de comunicaciones utilizada. Este punto es justo el que vamos a tratar en el siguiente apartado.

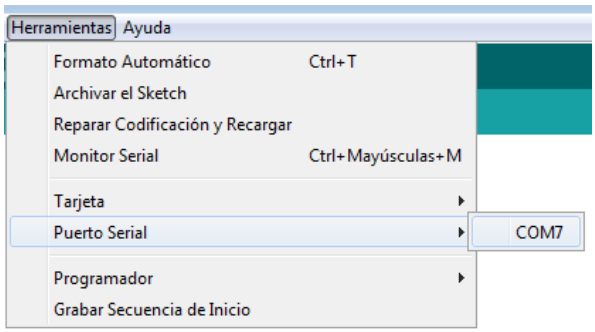
Configuración de la placa

Para configurar la placa tan sólo hay que seleccionar dos parámetros: el modelo de placa y el puerto serie. Para lo primero, hay que ir al menú "Herramientas" y dentro de "Tarjeta" escoger el modelo adecuado:



Atención: para el Duemilanove es necesario comprobar cuál es el modelo del micro que está insertado en el zócalo de la placa (ATmega168 o ATmega328).

Por otro parte, para escoger el puerto serie hay que ir a “Herramientas”-> “Puerto Serial” y seleccionar el puerto concreto:



El resto de parámetros deben de dejarse con sus valores por defecto.

Realizando un primer “Hola Mundo” con Arduino

Una vez configurada la placa, podemos comenzar a realizar pruebas con el Arduino. En primer lugar, vamos a hacer lo que se considera el “Hola Mundo” en microcontroladores: encender y apagar un led.

Para ello haremos uso de uno de los múltiples ejemplos incluidos con el IDE. Para cargarlo hay que ir a “Archivo->Ejemplos->01.Basics->Blink”. Se abrirá una nueva ventana del IDE con el código ya listo para ser compilado y cargado en el Arduino.

Antes de compilarlo y ejecutarlo vamos a estudiar el código:

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

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

```
// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;
```

```
// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}
```

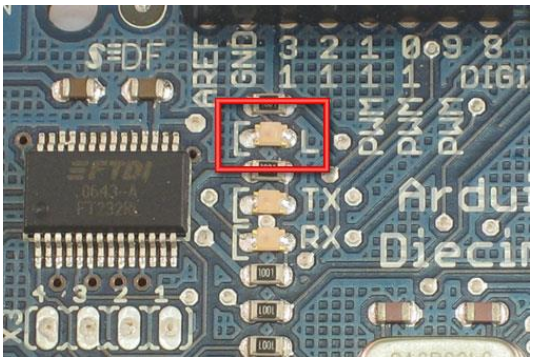
```
// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}
```

Este código se divide en tres partes:

- La **zona de declaración** de variables, constantes, funciones, etc…No es necesario que estas definiciones estén colocadas estrictamente al principio del código, pueden ir en cualquier parte del fichero.
- La **función *setup*()**. Es la primera función que ejecuta el microcontrolador. En ella se realizan las declaraciones de configuración del Arduino, las cuales sólo precisan ser ejecutadas una única vez.
- La **función *loop()***. Es la función principal del código y se ejecuta después de *setup()* a modo de bucle infinito (se ejecutará hasta que se retire la alimentación o se produzca un reset).

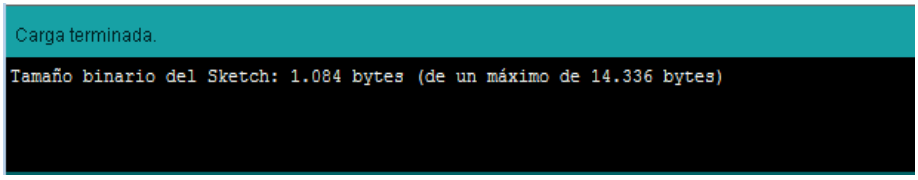
Respecto al código en sí:

- En la zona de declaración tan sólo se declara una variable de tipo *int* que indica el número de pin digital que se conmutará. En las placas Diecimila y Duemilanove, asociado a este pin 13 se encuentra un led que tiene serigrafíado a su lado la letra “L”.

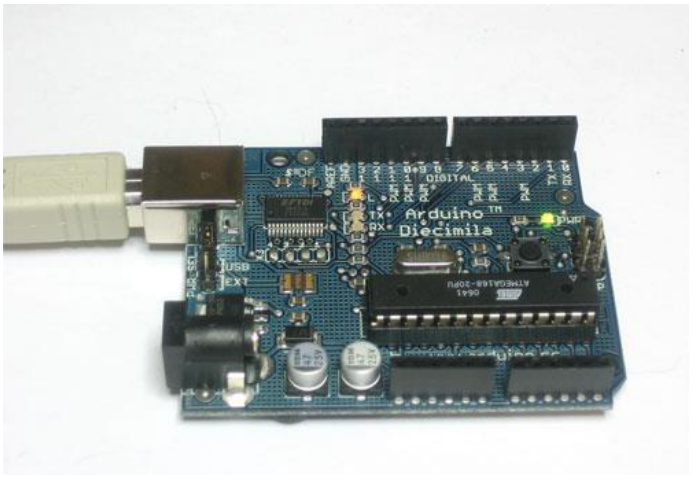


- En la función *setup()* se indica que se quiere configurar el pin 13 como salida para poder escribir valores lógicos 0 o 1 sobre el mismo (realmente, se podrá poner el pin a 0V o 5V).
- La función *loop()* apaga y enciende el led dejando intervalos de un segundo entre ambas operaciones. En concreto, *digitalWrite* permite cambiar el estado del pin entre HIGH y LOW, mientras que *delay* fuerza un retardo de milisegundos. Como se mencionaba anteriormente, esta función *loop()* se ejecutará en un bucle infinito.

Tras comprender el código, vamos a proceder a compilarlo. Para ello primero tenemos que pulsar sobre el icono de verificación (). Tras esto, en las áreas de notificaciones y de estatus de compilación/carga debería indicarse que la compilación se ha finalizado, así como el tamaño del binario compilado. A continuación, podemos proceder a realizar el downloading mediante el icono de carga (). Si ésta ha finalizado con éxito, así se indicará en el área del estatus de carga.



Unos segundos después de que finalice la carga se podrá ver como el led asociado al pin 13 comenzará a encenderse y apagarse a intervalos de un segundo.



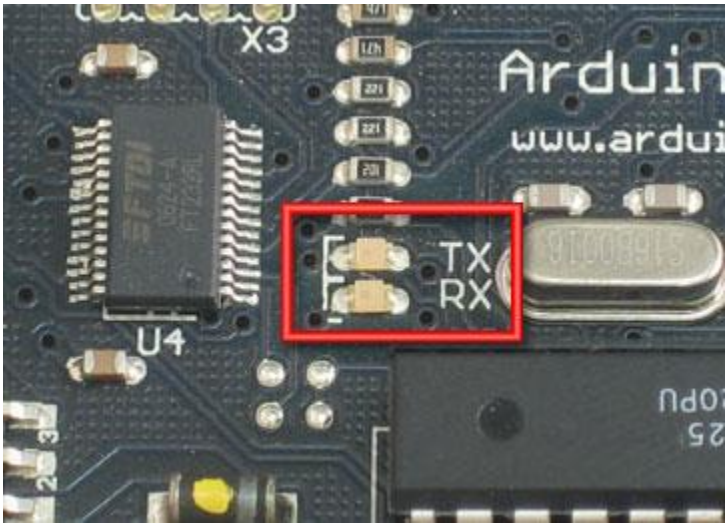
Modificando el código

La primera modificación directa que se puede hacer es cambiar los tiempos del delay. Probar a poner intervalos de 500 y 3.000 milisegundos para observar las diferencias en la frecuencia de parpadeo[4].

Control a través del puerto serie

Una modificación más avanzada consiste en alterar el código para que se pueda controlar el encendido/apagado a través de un PC conectado al Arduino. Para ello vamos a hacer uso de la librería *Serial*: nos conectaremos como clientes a través de una conexión serie al receptor serie del Arduino que recibirá los comandos y, si procede, ejecutará la acción de encender/apagar el led.

En primer lugar vamos a localizar donde se encuentran los pines y los leds asociados al puerto serie. Los pines del puerto serie se corresponden a las E/S digitales 0 y 1, y se encuentran etiquetados como RX/TX. Estos mismos pines se encuentran enganchados a un conversor serie-USB que nos evitará tener que incluir hardware adicional para comunicarnos con el micro. Los leds de este puerto se encuentran justo al lado del led asociado al pin 13 y cerca del conversor USB-serie (el chip FTDI):



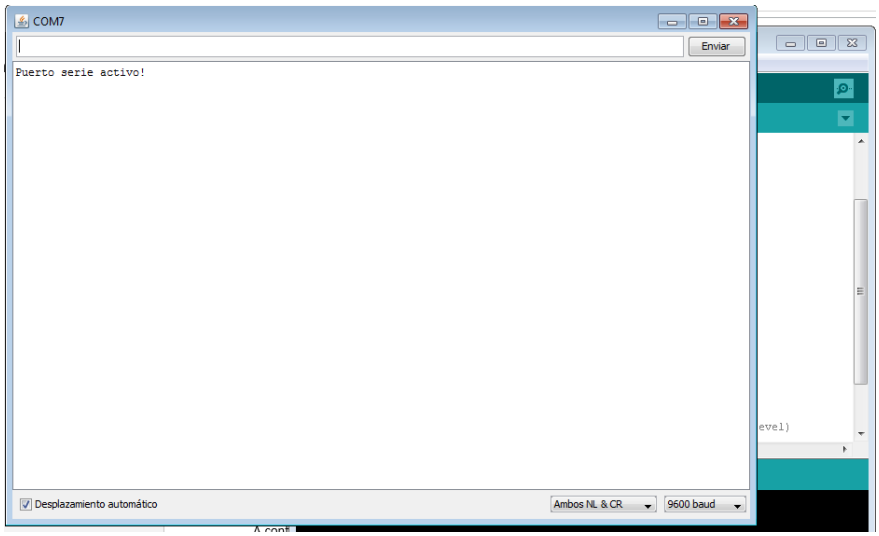
Una vez localizados, vamos a configurar el puerto serie por software y verificaremos que funciona correctamente. Para ello, vamos a alterar la función *setup()* del ejemplo anterior de la siguiente manera:

[4] El ejemplo “Blink” es de sólo lectura. En el momento de modificarlo y guardarlo se solicitará que se guarde con otro nombre.

```
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);

  Serial.begin(9600);
  Serial.println("Puerto serie activo!");

}
```

En este código la función *Serial.begin* sirve para activar y configurar la velocidad el puerto serie, indicándose que se quiere que funcione a 9.600 bps (baudios). La función *Serial.println* sirve para imprimir caracteres ASCII seguidos de un '\n'.

A continuación compilamos y cargamos el programa en el Arduino. Para comprobar que el puerto serie funciona correctamente vamos a usar el *Serial Monitor* que trae el IDE. Éste se encuentra en "Herramientas->Monitor Serial". Tras cargar el programa, al abrir este Serial Monitor deberíamos ver como se recibe el mensaje "Puerto serie activo!":



Asimismo, podemos ver cómo se comportan los leds del puerto serie cuando se transmiten datos desde el Arduino. Para ello, se puede incluir un *Serial.println* en el código de la función *loop()*: se puede ver el parpardeo del led TX cada vez que se envían datos desde el microcontrolador.

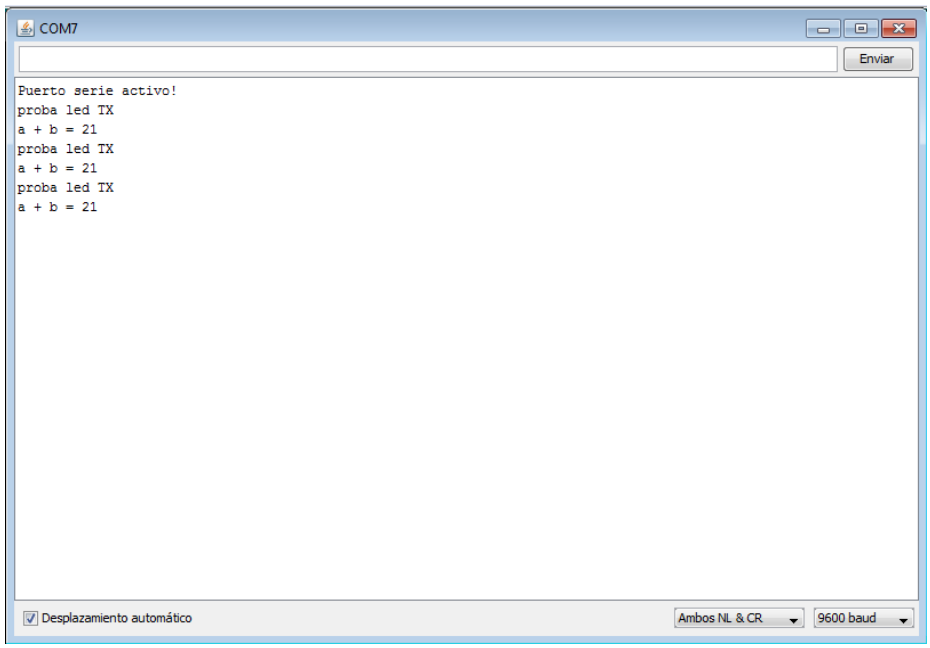
Existe también la función *Serial.print*, la cual funciona de manera similar a *Serial.println*, pero no añade el '\n' al final. Estas dos funciones admiten múltiples formatos que nos permiten enviar información de debugging que nos va a resultar muy útil durante el desarrollo[5]. Para comprobar simplemente su uso en un caso fácil, podemos declarar un par de variables enteras 'a' y 'b', y enviar el valor de su suma por el

[5] En este enlace se encuentra una recopilación de los formatos admitidos.

puerto serie. El código (incluyendo las modificaciones de los apartados anteriores), quedaría como sigue:

```
int led = 13;
int a = 17;
int b = 4;

void setup() {

  pinMode(led, OUTPUT);
  Serial.begin(9600);
  Serial.println("Puerto serie activo!");

}

void loop() {
  Serial.println("proba led TX");
  Serial.print("a + b = ");
  Serial.println(a + b);
  digitalWrite(led, HIGH);
  delay(1000);
  digitalWrite(led, LOW);
  delay(1000);
}
```

En el Serial Monitor se debería ver lo siguiente:



Antes de continuar con el control del puerto serie, conviene aclarar en este punto los tipos soportados por el lenguaje de programación de Arduino. Para ello vamos a hacer una prueba: cambiaremos en el código anterior el valor de 'a' de 17 a 70.000. ¿Qué se recibe ahora como suma de 'a' y 'b'?

Obviamente se ha producido un overflow debido a que 'a' es de tipo 'int': este tipo está limitado a 16 bits, con lo que el número más grande representable será 65535. Para evitar problemas similares, a continuación se exponen los tipos admitidos, su tamaño y su uso recomendado:

Tipo	Bytes	Rango	Uso
int	2	-32768 a 32767	Enteros positivos y negativos.
unsigned int	2	0 a 65535	Enteros positivos.
long	4	-2147483648 a -2147483648	Valores positivos y negativos grandes.
unsigned long	4	4294967295	Valores positivos grandes.
float	4	3.4028235E+38 a -3.4028235E+38	Números con decimales.
double	4	Idem a Float	En Arduino double es idéntico a float.
boolean	1	Falso(0) o True(1)	Valores booleanos.
char	1	-128 a 127	Caracteres o enteros entre -128 a 127.
byte	1	0 a 255	Idem a char pero para valores sin signo.
String	-	-	Arrays de char.

En el caso de los arrays, su uso es similar al de otros lenguajes. Un ejemplo de su declaración, inicialización y acceso sería:

```
Int inputPins[] = {1,2,3,4};

Int pin = inputPins[2];
```

Atención: es muy fácil incurrir en overflows en un microcontrolador al encontrarnos con tipos con menor tamaño que los equivalentes a los de una arquitectura basada en microprocesadores. Por ello, hay que tener siempre en cuenta el rango de valores de una variable, especialmente cuando en ella se va a almacenar el resultado de una operación que puede desbordar (e.g. multiplicación).

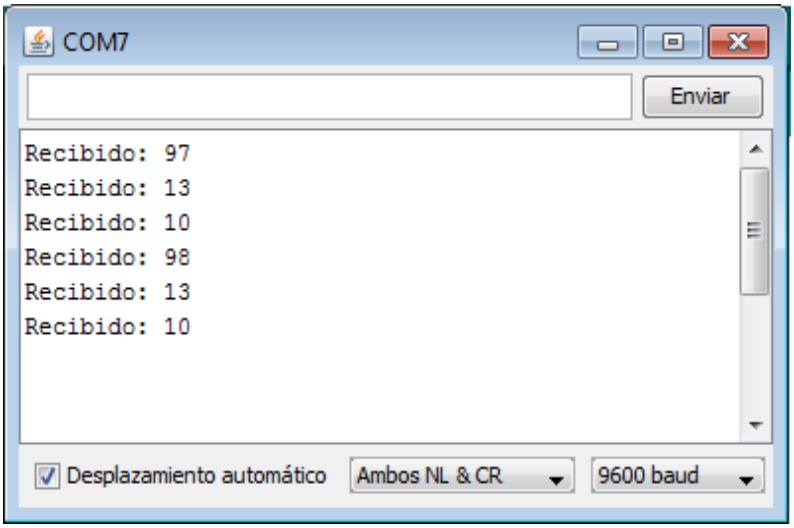
Igualmente hay que tener cuidado al mezclar elementos de distintos tipos (e.g. multiplicar un int por un float). Conviene especificar siempre con un *cast* el tipo que se quiere que tenga el resultado.

Realizado este inciso, podemos continuar con el software para el control del led a través del puerto serie. Hasta ahora sólo se ha visto cómo transmitir información desde el Arduino al ordenador. Para que sea el Arduino quien reciba la información es necesario hacer uso de la función *Serial.read()*. Dicha función permite que el micro lea byte a byte los datos recibidos a través del puerto serie.

Un primer ejemplo de esta función es un servidor de eco: cada carácter que reciba el microcontrolador por el puerto serie, lo volverá a enviar a través del mismo puerto serie al ordenador. Para ello, se utilizaría el siguiente código:

```
int byteRecibido = 0;

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


void loop() {
  if (Serial.available() > 0) {

    // lee el byte entrante:
    byteRecibido = Serial.read();

    //se hace el eco:
    Serial.print("Recibido: ");
    Serial.println(byteRecibido);
  }

}
```

Esto sería lo que se observaría en el Serial Monitor cuando se envían consecutivamente los caracteres ‘a’ y ‘b’ acompañados de ‘\r\n’:



Jugando con el buzzer

Un buzzer (zumbador, generalmente piezoeléctrico) permite emitir sonidos a distintas frecuencias. Será el primer actuador que utilicemos en prácticas. El modelo concreto que vamos a usar consta de dos pines: la patilla negativa va a GND y la positiva (marcada con un '+') va a un pin digital (para evitar conectar cableado adicional colocaremos el buzzer directamente sobre el zócalo del Arduino, utilizando GND y el pin digital 12).

Su uso es tremendamente sencillo. El siguiente código permite emitir una escala:

```
int pinBuzzer = 12;

int numTonos = 10;
int tonos[] = {261, 277, 294, 311, 330, 349, 370, 392, 415, 440};
          //{mid C,  C#,   D,  D#,   E,   F,  F#,   G,   G#,  A}

void setup()
{
  for (int i = 0; i < numTonos; i++)
  {
    tone(pinBuzzer, tonos[i]);
    delay(500);
   }

  noTone(pinBuzzer);
}

void loop()
{
}
```

La función *tone(pin, frecuencia)* permite emitir un tono de la frecuencia indicada a través de un pin. Al final, se trata de poner el pin indicado a 'ON' o 'OFF' un número de veces por segundo (por ejemplo, en tonos[1] sería 277 veces por segundo para generar C sostenido).

Enunciado de la práctica

A partir del código mostrado en esta práctica, implementar un programa que encienda el led asociado al pin 13 cuando se reciba por el puerto serie el carácter 'E' y que lo apague cuando reciba el carácter 'A'[6]. Asimismo, hacer que cuando se reciba el carácter 'S' se emita un sonido a través del buzzer. Enviar el código resultante (esta vez sin necesidad de memoria ajunta) a tiago.fernandez@udc.es.

[6] Para que el Serial Monitor no envíe el '\r\n', seleccionar en la parte inferior del Serial Monitor la opción "No hay fin de línea".