

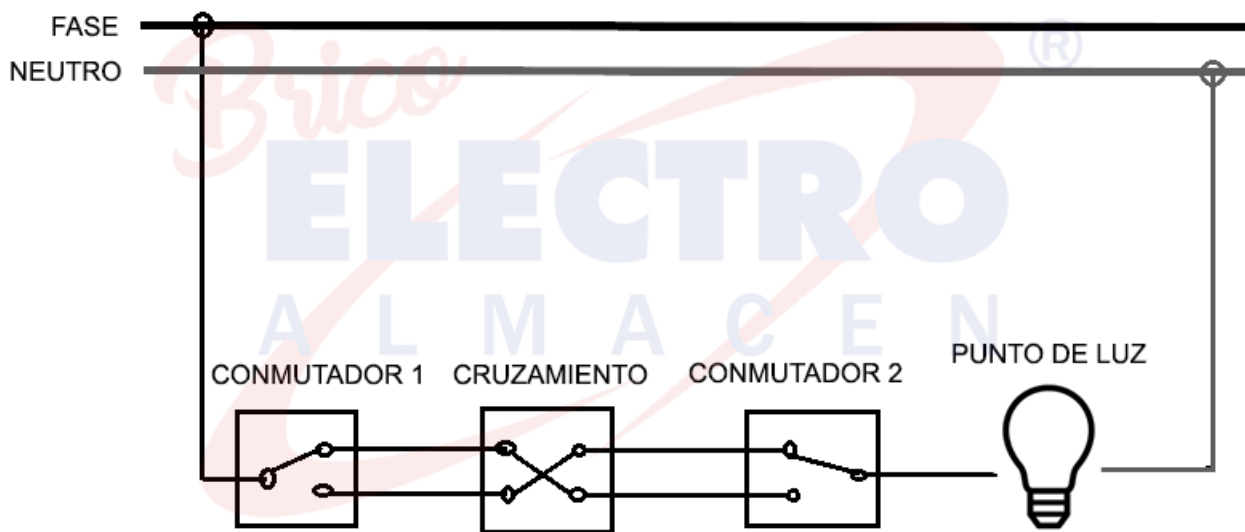
## Table of Contents

1	Introducción.....	2
1.1	La plataforma Arduino.....	4
1.2	El software (IDE).....	7
1.3	Programación de un dispositivo interactivo.....	8
1.4	Práctica 01: LED intermitente.....	9
1.4.1	Estructura de un sketch.....	16
1.4.2	Análisis del sketch práctica 01.....	18
1.4.3	Mediciones práctica 01.....	21
1.5	Práctica 02: LED encendido por un pulsador mientras se mantiene pulsado.....	22
1.5.1	Mediciones práctica 02.....	24
1.5.2	If ... else.....	24
1.6	Práctica 03: LED encendido por un pulsador con función de interruptor.....	26

# 1 Introducción

No hace mucho, los circuitos eléctricos y electrónicos se construían utilizando componentes de una sola función, como por ejemplo interruptores, conmutadores o cruzamientos.

Con este tipo de componentes no programables, cada circuito estaba cableado para realizar una función específica, por ejemplo encender y apagar una lámpara desde distintos lugares. Cambiar la función de un circuito exigía modificar el cableado y sustituir componentes.



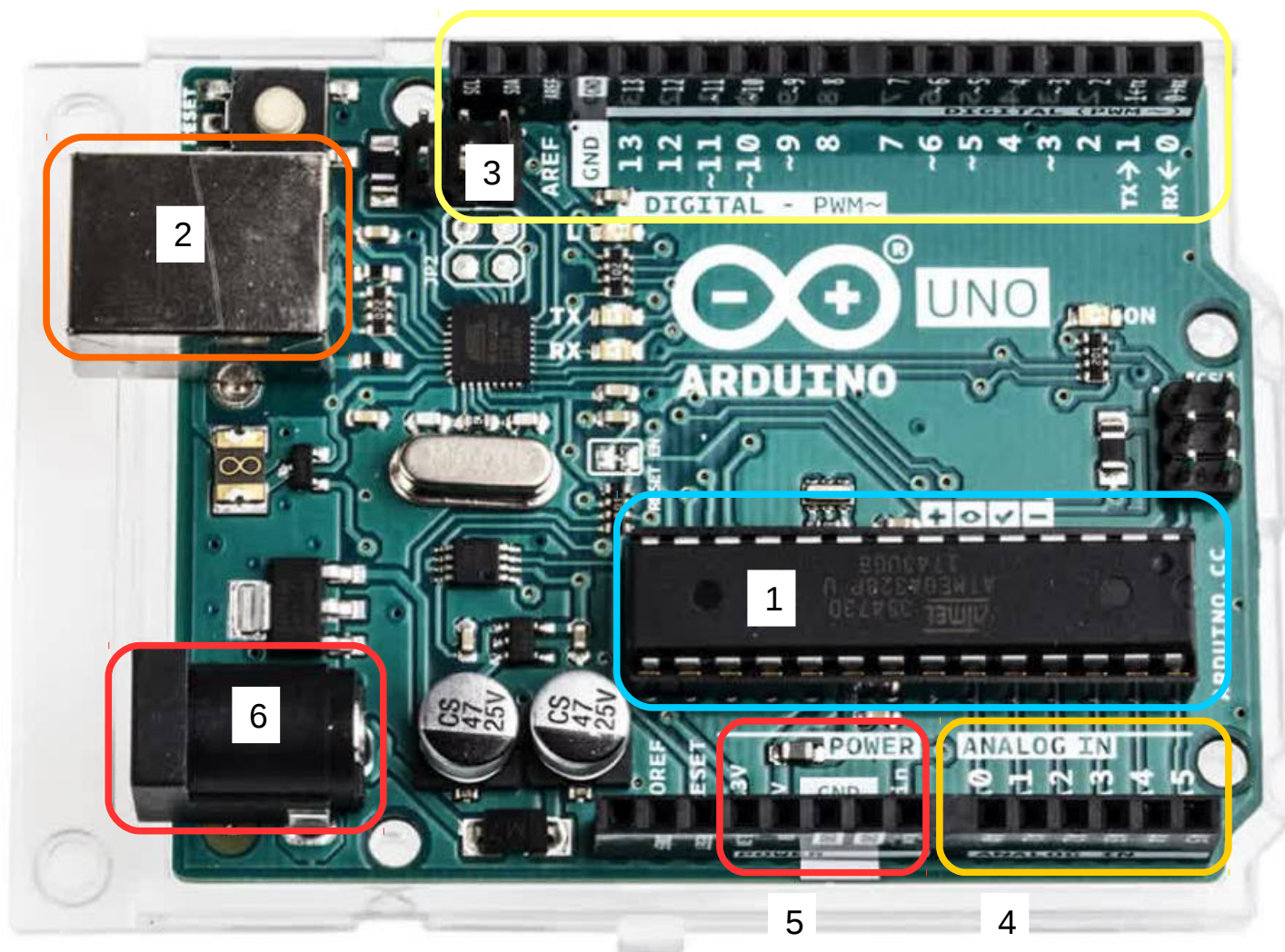
Con la aparición de las tecnologías digitales y los microprocesadores, estas funciones, que antes se implementaban mediante el cableado y las características específicas de cada componente, fueron sustituidas por componentes programables que permiten realizar multitud de funciones.



Un dispositivo con microprocesador permite realizar las funciones de diversos dispositivos con componentes no programables

## 1.1 La plataforma Arduino

El hardware del sistema arduino es una placa base con el microcontrolador, las entradas, salidas y tomas de alimentación.



1. Microcontrolador (ATmega328)
2. Conexión a ordenador, USB B, alimentación (5V) y datos.
3. Salidas/Entradas digitales
4. Entradas analógicas
5. Tomas GND (ground) de tierra (negativo) y tomas de 5 V y 3,3 V
6. Toma de alimentación 6 V a 12 V, para conector de barril

El microcontrolador es programable desde un ordenador. Para la conexión se utiliza un cable de USB A a USB B.



USB A

USB B



La web <https://store.arduino.cc/en-es/collections/boards-modules> presenta una gran variedad de placas Arduino. Para las prácticas propuestas en este documento se utilizará un Arduino modelo UNO.

El Arduino UNO dispone de:

#### **14 pines digitales (pines 0 a 13)**

Estos pueden funcionar como entradas o salidas, según se especifique en el programa (sketch) que ejecutará el microcontrolador. La entrada o salida digital, reconoce o muestra, dos estados, 5 V o 0V.

#### **6 pines analógicos (pines 0 a 5) para señales de entrada**

Una entrada analógica reciben una señal, por ejemplo de tensión, suministradas por un sensor, y la convierte en un número entre 0 y 1023 (convertor analógico digital de 10 bit).

#### **6 pines analógicos (pines 3, 5, 6, 9, 10 y 11) para señales de salida**

Se trata de 6 de los pines digitales, que pueden ser reprogramados como salidas analógicas.

Cuando la placa está conectada a un ordenador mediante el cable USB A/B, es alimentada desde la toma USB. Para alimentar la placa sin utilizar un cable USB, se puede utilizar el pin Vin, y se recomienda una tensión de 6 V a 12 V. Con una fuente de alimentación que disponga de un conector de barril de 2,1 mm con polo positivo en el centro (interior), también puede alimentar, utilizando la toma para conector de barril.



## 1.2 El software (IDE)

IDE (Integrated Development Environment) es un programa que se debe instalar en el ordenador, para crear los programas específicos del Arduino, llamados sketches.

El lenguaje de programación utilizado en los sketches se llama Processing Language ([www.processing.org](http://www.processing.org)).

Los pasos a seguir para la programación del Arduino son los siguientes:

1. Conectar la palca Arduino al puerto USB del ordenador.
2. Escribir un sketch definiendo el funcionamiento de los componentes del circuito.
3. Descargar el sketch al Arduino y esperar unos segundos al reinicio.
4. El Arduino ejecuta el sketch.

### 1.3 Programación de un dispositivo interactivo

Un dispositivo interactivo es un circuito electrónico capaz de percibir el entorno mediante sensores (componentes electrónicos que convierten las mediciones del mundo real en señales eléctricas).

El dispositivo procesa la información que obtiene de los sensores y puede reaccionar mediante actuadores, componentes electrónicos capaces de convertir una señal eléctrica en una acción física. La reacción está determinada por el programa que ejecuta el microcontrolador.

Ejemplos de <b>sensores</b>	Ejemplos de <b>actuadores</b>
Sonda de temperatura	Dispositivos luminosos (lámparas, LEDs)
Sonda de humedad	Pantalla LCD
Detector de gases hidrocarburos	Dispositivos acústicos (altavoz, zumbador )
Detector de monoxido de carbono	Dispositivos interruptores (transistor, relé)
Sonda Hall (detector de campo magnético)	Motor
Detector de final de carrera	



## 1.4 Práctica 01: LED intermitente

La primera práctica será montar un circuito con un LED que se encenderá y apagará de forma intermitente.

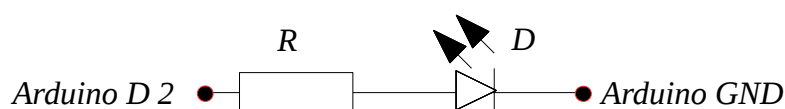
El LED es un componente con polaridad, su contacto más largo se debe conectar al polo positivo y el más corto al negativo. En caso contrario bloqueará el paso de la corriente, sin encenderse.



Además se debe limitar la corriente que circula por el LED a aproximadamente 20 mA. Si se utiliza la tensión de alimentación de 5 V disponible en la placa Arduino, la resistencia necesaria para

limitar la corriente será de al menos  $R = \frac{E}{I} = 5 \frac{V}{0,02 A} = 250 \Omega$ . Se pueden utilizar resistencias de valores bastante más altos, por ejemplo de  $1 k\Omega$  para proteger el LED.

En esta práctica, se elige la salida digital 2 para dar señal al LED. El esquema de conexión es el siguiente:

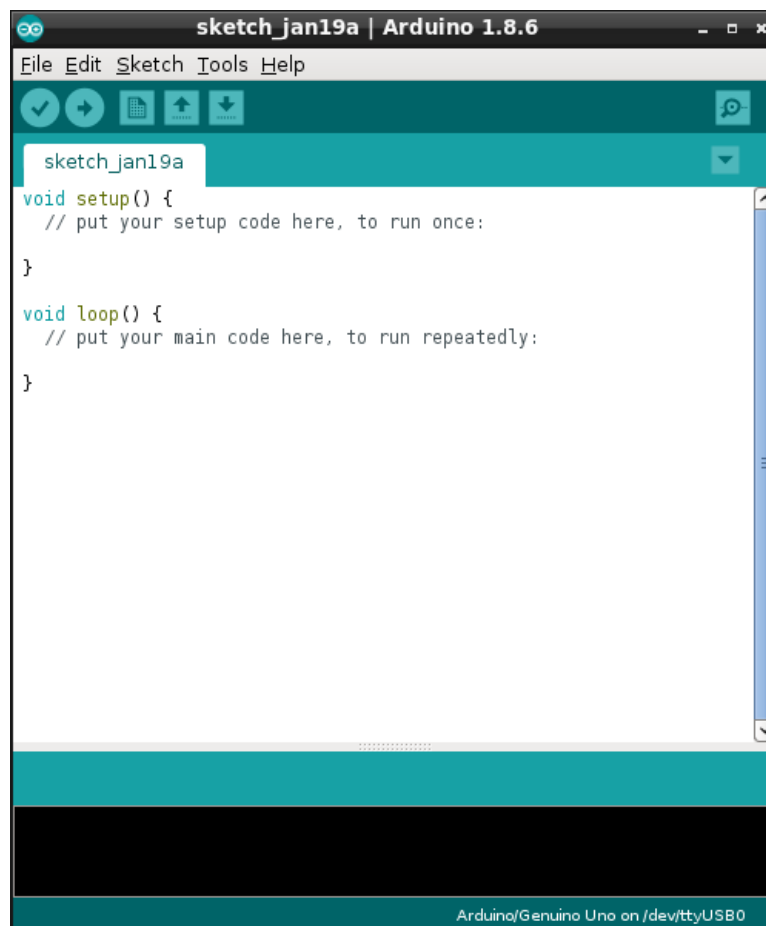


Una vez conectados los componentes se crea el código que ejecutará el microcontrolador.

Para ello se abre la IDE, haciendo doble click sobre el icono.



Seleccionar File>New y aparecerá un sketch nuevo



En primer lugar se guardará el nuevo sketch con el nombre “LED\_intermitente”. Para ello seleccionar File>Save as ...

A continuación se debe introducir el siguiente código:

```
                                // Práctica 01 : LED _intermitente

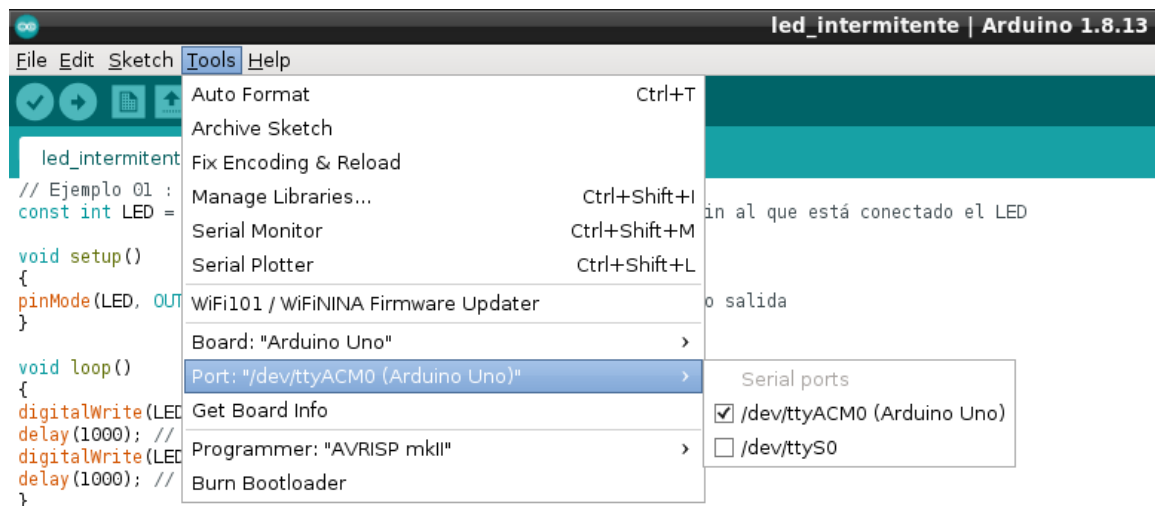
const int LED = 2;             // LED recibe el valor 2, que es el número del pin al que está
                                // conectado el LED

void setup()
{
  pinMode(LED, OUTPUT); // define el pin número LED, es decir 2, como salida
}

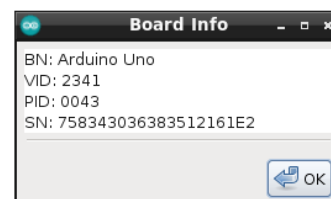
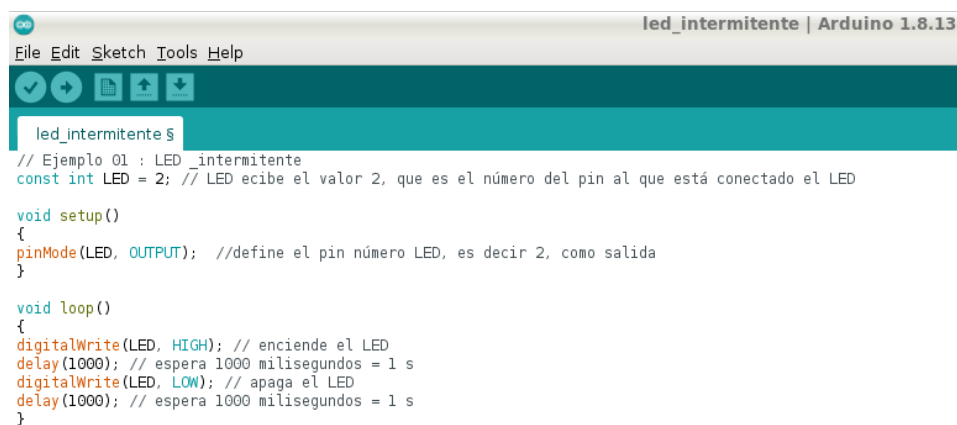
void loop()
{
  digitalWrite(LED, HIGH); // enciende el LED
  delay(1000);             // espera 1000 milisegundos = 1 s
  digitalWrite(LED, LOW);  // apaga el LED
  delay(1000);             // espera 1000 milisegundos = 1 s
}
```

Ahora se comprueba que el ordenador reconoce que la placa Arduino conectada y se puede comunicar con ella.

Tools>Port → marcar la opción Arduino 1



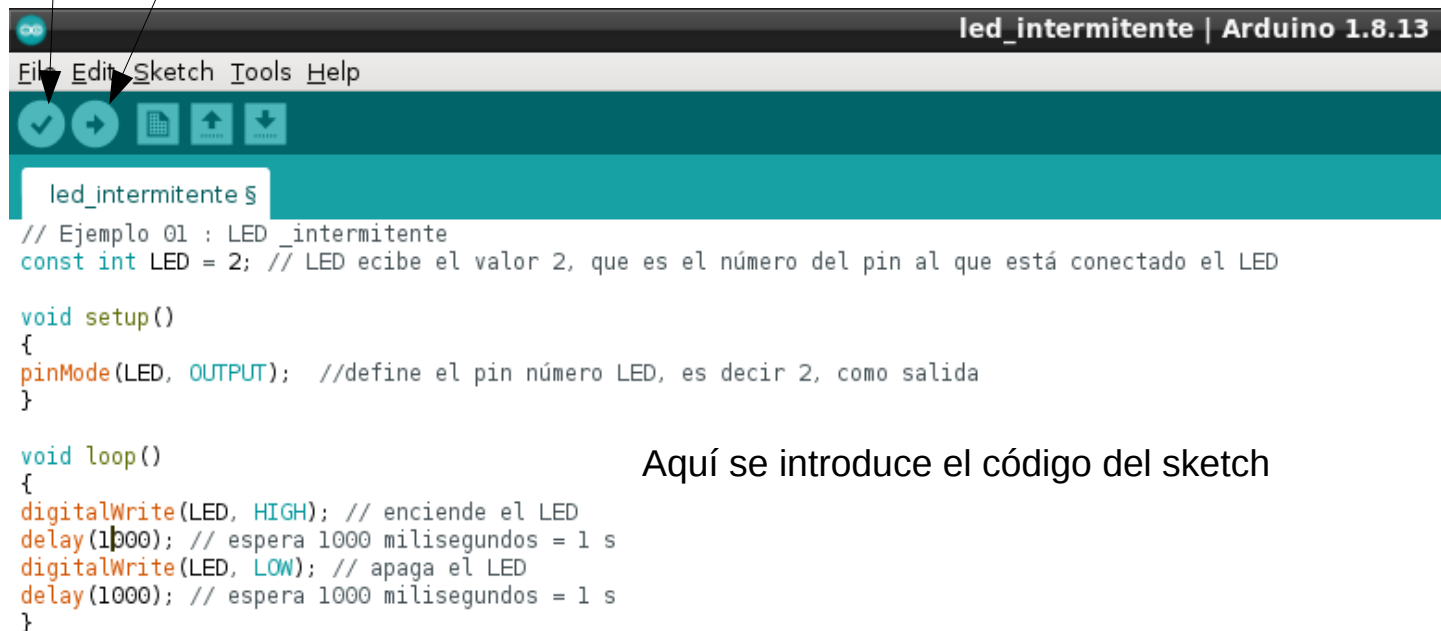
Seguidamente TOOLS>Get Board info → muestra los datos de la placa Arduino



La comunicación entre el ordenador y la placa Arduino funciona. La placa ha sido reconocida e identificada.


Verifica el código

Descarga el código al Arduino

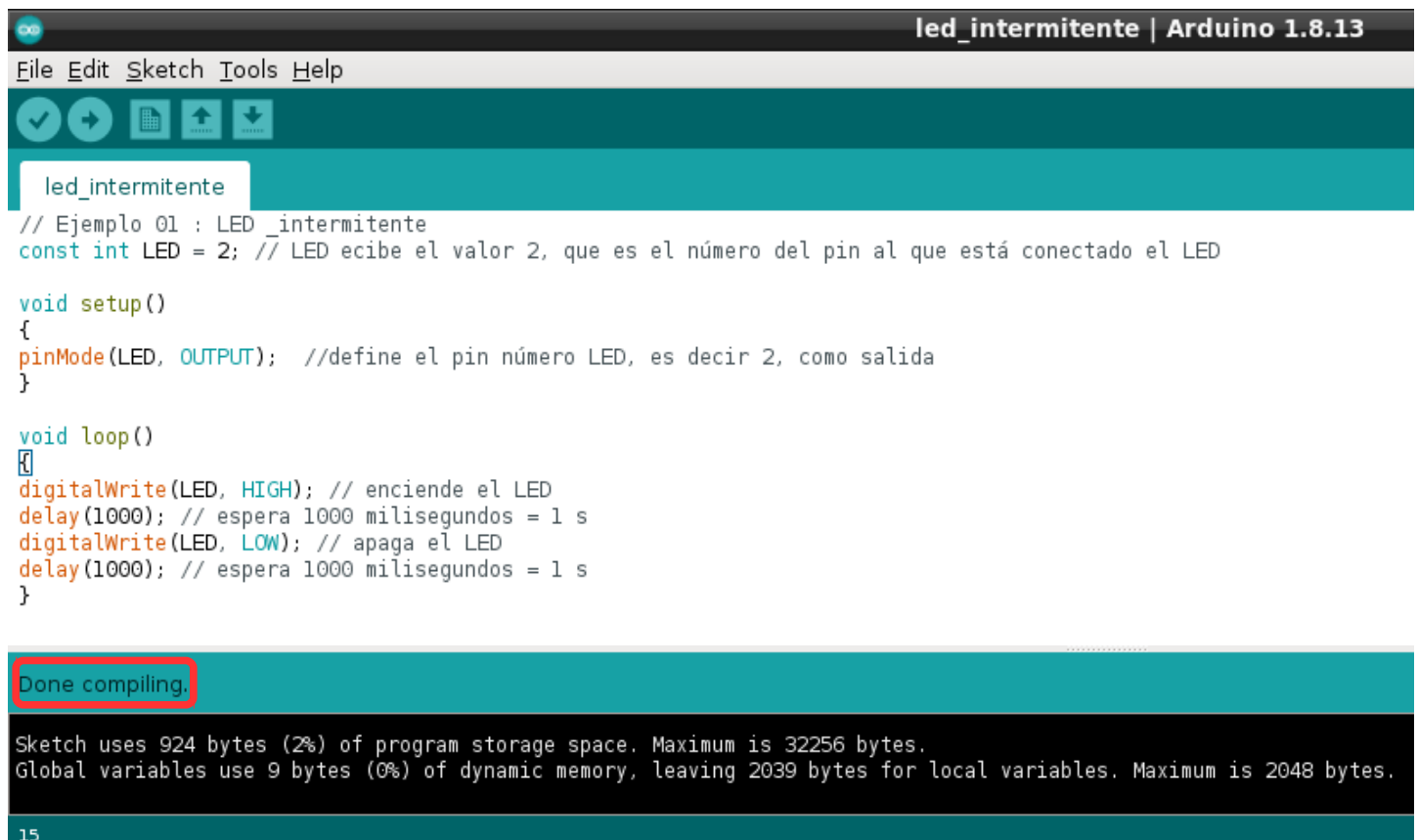


Aquí se introduce el código del sketch

La imagen muestra la IDE con el primer sketch introducido.

Una vez introducido el código en la IDE, conviene verificar que no contiene errores. Para verificar el código se pulsa sobre el botón de verificación. 

Si el código es correcto, se mostrará el mensaje “Done compiling”



```
led_intermitente | Arduino 1.8.13
File Edit Sketch Tools Help

led_intermitente
// Ejemplo 01 : LED _intermitente
const int LED = 2; // LED recibe el valor 2, que es el número del pin al que está conectado el LED

void setup()
{
  pinMode(LED, OUTPUT); //define el pin número LED, es decir 2, como salida
}

void loop()
{
  digitalWrite(LED, HIGH); // enciende el LED
  delay(1000); // espera 1000 milisegundos = 1 s
  digitalWrite(LED, LOW); // apaga el LED
  delay(1000); // espera 1000 milisegundos = 1 s
}

Done compiling.

Sketch uses 924 bytes (2%) of program storage space. Maximum is 32256 bytes.
Global variables use 9 bytes (0%) of dynamic memory, leaving 2039 bytes for local variables. Maximum is 2048 bytes.

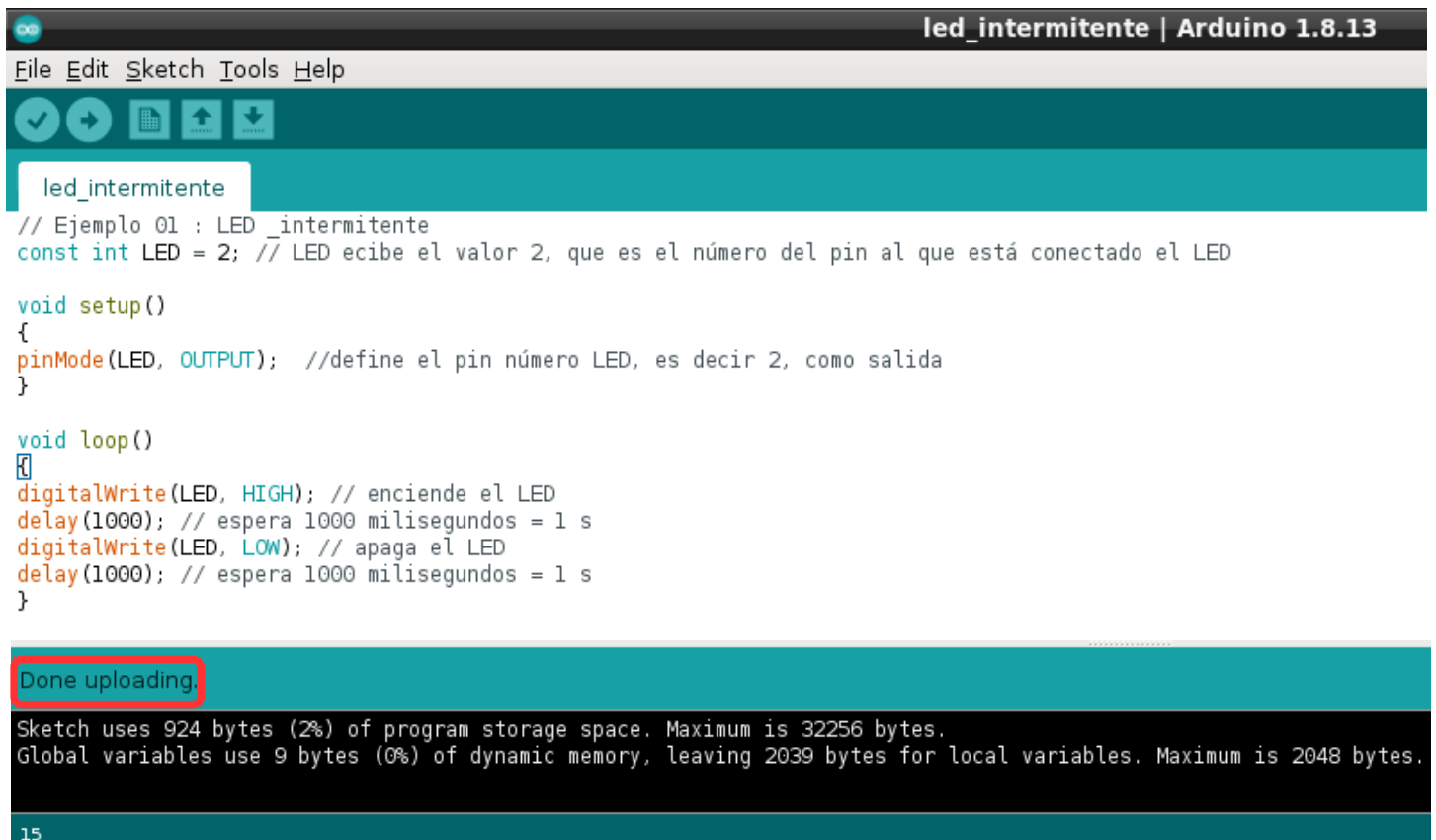
15
```

El sketch ha sido compilado, es decir, traducido al lenguaje de máquina que el microcontrolador puede interpretar.

Habiendo verificado que el sketch se ha compilado sin errores, es el momento de descargarlo a la placa Arduino, pulsando el botón de descarga 

Al descargar un sketch, el Arduino interrumpe el programa que esté ejecutando y se resetea, almacenando en la memoria el nuevo sketch que recibe desde la IDE a través del puerto USB. Una vez almacenado el sketch, inicia su ejecución.

Durante la descarga, los LEDs RX y TX situados en la placa Arduino parpadean, indicando que el ordenador y la placa Arduino están comunicando. Una vez finalizada la descarga, aparecerá el mensaje “Done uploading”.



```
led_intermitente | Arduino 1.8.13
File Edit Sketch Tools Help
led_intermitente
// Ejemplo 01 : LED _intermitente
const int LED = 2; // LED recibe el valor 2, que es el número del pin al que está conectado el LED

void setup()
{
  pinMode(LED, OUTPUT); //define el pin número LED, es decir 2, como salida
}

void loop()
{
  digitalWrite(LED, HIGH); // enciende el LED
  delay(1000); // espera 1000 milisegundos = 1 s
  digitalWrite(LED, LOW); // apaga el LED
  delay(1000); // espera 1000 milisegundos = 1 s
}

Done uploading.
Sketch uses 924 bytes (2%) of program storage space. Maximum is 32256 bytes.
Global variables use 9 bytes (0%) of dynamic memory, leaving 2039 bytes for local variables. Maximum is 2048 bytes.
15
```

Una vez que el Arduino ha sido programado con un sketch, el código queda guardado en la memoria y se mantiene en ella, hasta que se descargue un nuevo sketch. La memoria guarda un sketch incluso si la placa se desconecta de la alimentación, equivale al disco duro de un ordenador.

Si el sketch ha sido descargado correctamente, el LED conectado debería parpadear en intervalos de 1 s.

### 1.4.1 Estructura de un sketch

El microcontrolador del Arduino ejecuta el sketch de arriba abajo, es decir, la primera línea de código es la primera que lee y ejecuta. A continuación ejecuta la segunda línea y así sucesivamente, hasta llegar a la última línea. Cada línea de código es una instrucción que debe acabar con el símbolo de punto y coma ;

Las instrucciones se agrupan escribiéndolas entre llaves ,

```
{  
instrucción 1;  
instrucción 2;  
instrucción 3;  
}
```

El sketch de la práctica 01 muestra dos bloques de código entre llaves.

Estos bloques se llaman funciones y los nombres que las identifican son void setup () y void loop ().

Un sketch precisa contener al menos estas dos funciones. La función void setup () siempre precede a void loop (), ya que en void setup () se define el comportamiento o funcionamiento de componentes del circuito. En la práctica 01, la función void setup () define que el pin 2 es una salida, utilizando la instrucción:

```
pinMode(LED, OUTPUT);
```

La función void setup se ejecuta una única vez, inicializando los componentes del circuito, para que a continuación, la función void loop () se repita continuamente, mientras se mantenga alimentado el Arduino.



La función void loop () es la que contiene las instrucciones que se repiten sucesivamente (loop significa bucle). En la práctica 01 el bloque de instrucciones correspondiente a void loop () es :

```
digitalWrite(LED, HIGH);           // enciende el LED
delay(1000);                        // espera 1000 milisegundos = 1 s
digitalWrite(LED, LOW);            // apaga el LED
delay(1000);                       // espera 1000 milisegundos = 1 s
```

A diferencia de un ordenador, el Arduino no puede ejecutar multiples programas simultáneamente, ni finalizar la ejecución de un programa. Al alimentar el Arduino, su programa se ejecuta, la única manera de pararlo es desconectando la alimentación, o descargando un nuevo sketch.

Los textos iniciados por // son ignorados por el Arduino, y sirven como comentarios, que aclaran el funcionamiento del sketch a quien lo aplique.

### 1.4.2 Análisis del sketch práctica 01

A continuación se analiza el sketch práctica 01 línea a línea.

**Línea 1:** // Práctica 01 : LED\_intermitente

Comentario inicial que identifica el sketch a modo de título.

**Línea 2:** `const int LED = 2;` // LED recibe el valor 2, que es el número del pin al que está

```
// conectado el LED
```

El primer paso en un sketch es definir constante y variables. En este caso se define que la constante LED, tiene el valor 2. Una constante siempre mantiene el valor con la que fue definida. Cada vez que aparece LED en una instrucción, es sustituido por el número 2.

int significa integer, e identifica a la constante LED como un número entero.

### Línea 3: void setup()

Declara la función `setup`. Esta función inicializa componentes del sistema y sólo se ejecuta una vez.

**Línea 4: {**

La llave abierta marca el comienzo del bloque de instrucciones de la función setup.

**Línea 5:** pinMode(LED, OUTPUT); //define el pin número LED, es decir 2, como salida

Esta es la única instrucción de la función setup. Define el pin identificado por LED, y LED es el número 2, como un pin de salida.

Al tratarse de una salida digital, el valor del pin 2 puede ser HIGH, que significa que en el pin hay una tensión de 5 V respecto a GND, o LOW, que significa que el pin presenta una tensión de 0V respecto a GND.

Pinmode es una función, que precisa las dos constantes LED y OUTPUT como argumentos. Esta función asigna al pin identificado por la primera constante, su modo de funcionamiento, que puede ser como pin con señal de salida (OUTPUT), o pin con señal de entrada (INPUT).

**Línea 6: }**

La llave cerrada marca el final del bloque de instrucciones de la función setup.

**Línea 7: void loop()**

Declara la función loop. Las instrucciones de esta función se ejecutan en bucle (sucesivamente), hasta desconectar el Arduino o cargar un nuevo sketch.

**Línea 8: {**

La llave abierta marca el comienzo del bloque de instrucciones de la función loop.

**Línea 9: digitalWrite(LED, HIGH); // enciende el LED**

La función digitalWrite asigna un estado a un pin. En este caso asigna el estado HIGH al pin identificado por la constante LED, que es 2. El pin 2 presenta una tensión de 5V respecto a GND. El diodo se enciende.

**Línea 10:** `delay(1000); // espera 1000 milisegundos = 1 s`

La función `delay` introduce un tiempo de espera de 1000 milisegundos, antes de que el microprocesador pase a ejecutar la siguiente instrucción. El diodo se mantiene encendido 1 s.

**Línea 11:** `digitalWrite(LED, LOW); // apaga el LED`

La función `digitalWrite` asigna un estado a un pin. En este caso asigna el estado `LOW` al pin identificado por la constante `LED`, que es 2. El pin 2 presenta una tensión de 0V respecto a GND. El diodo se apaga.

**Línea 12:** `delay(1000); // espera 1000 milisegundos = 1`

La función `delay` introduce un tiempo de espera de 1000 milisegundos, antes de que el microprocesador pase a ejecutar la siguiente instrucción. El diodo se mantiene apagado 1 s.

**Línea 13:** `}`

La llave cerrada marca el final del bloque de instrucciones de la función `loop`.

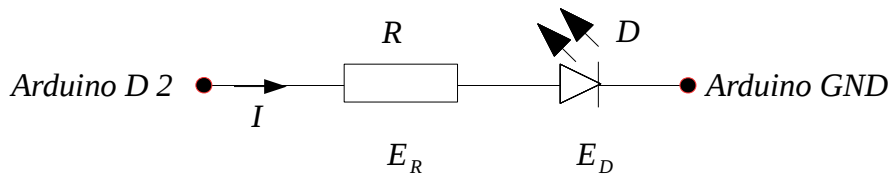
Desde esta línea, el microprocesador vuelve a ejecutar la primera instrucción del bloque `loop`.

El funcionamiento del sketch práctica 01 se puede resumir como sigue:

- El pin 2 es definido como pin de salida (esto sólo ocurre una vez, al inicio del sketch).
- Inicio de la ejecución de las instrucciones en bucle
- El diodo conectado al pin 2 se enciende.
- Pausa de 1 s.
- El diodo conectado al pin 2 se apaga.
- Pausa de 1 s.
- Vuelta al inicio del bucle

### 1.4.3 Mediciones práctica 01

Mide los valores de las tensiones y la corriente.



$$E_R =$$

$$E_D =$$

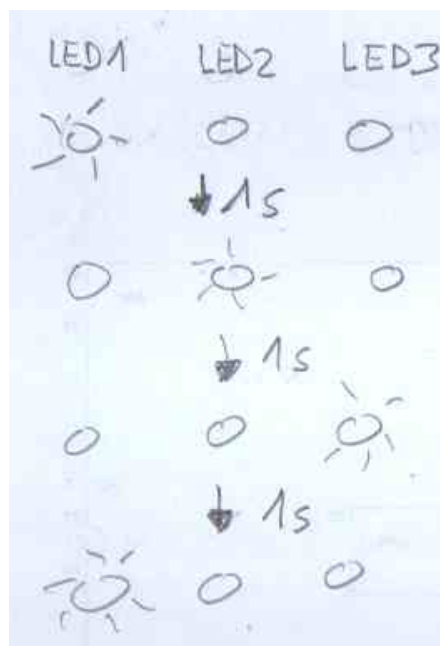
$$I =$$

$$R =$$

#### Ejercicio 1:

Crea un sketch que encienda secuencialmente 3 LEDs, con una pausa de 1 segundo entre el encendido y apagado.

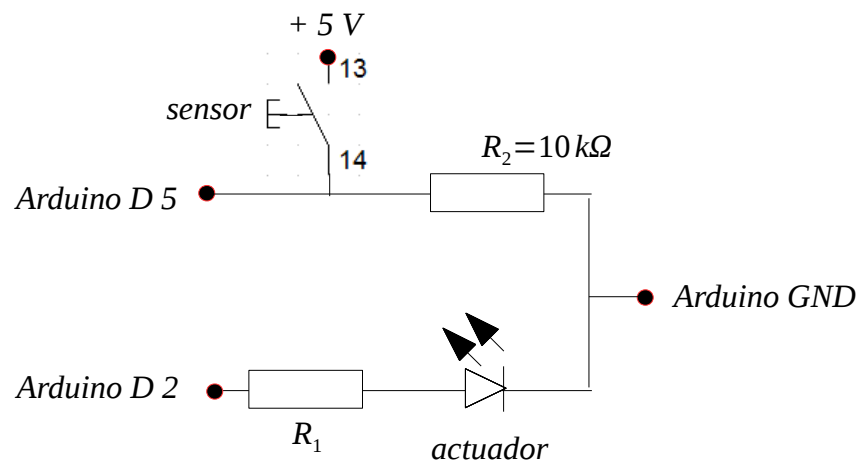
Dibuja el esquema de montaje de los 3 LEDs.



## 1.5 Práctica 02: LED encendido por un pulsador mientras se mantiene pulsado

En esta práctica se enciende un LED mientras se mantenga accionando un pulsador. A falta de pulsador se pueden utilizar dos conductores que se juntan cerrando el circuito.

Con el Arduino se precisará un sensor, que será un pulsador, para dar una señal de entrada, que mientras dure, encienda del diodo.



Para conocer el estado del pulsador, se utilizará la función `digitalRead()`.

La función `digitalRead()` devuelve el valor LOW en caso de no detectar tensión en el pin que le ha sido asignado y el valor HIGH en caso de detectar tensión en el pin. Es decir, `digitalread()` permite saber qué es lo que está pasando en el exterior, si la entrada digital está o no recibiendo señal y devuelve un valor que se puede guardar en la memoria y ser utilizado por el microcontrolador para determinar qué instrucción ejecutar.

A continuación se debe crear un nuevo sketch, llamado practica\_02 e introducir el siguiente código:

// práctica 02: Encender un LED con un pulsador

```
const int LED = 2;           // LED recibe el valor 2, que es el número del pin al que  
                             // está conectado el LED
```

```
const int PULSADOR = 5;     // PULSADOR recibe el valor 5, que es el número del  
                             // pin al que está conectado el pulsador
```

```
int val = 0;                 // val guardará el estado del pulsador, 0 sin señal de  
                             // tensión, 1 con señal de tensión
```

**void setup()**

```
{  
pinMode(LED, OUTPUT);       // define el pin número LED, es decir 2, como salida  
  
pinMode(PULSADOR, INPUT);   // define el pin número PULSADOR, es decir 5, como  
                             // entrada  
}
```

**void loop()**

```
{  
val = digitalRead(PULSADOR); // detecta y guarda el estado del pulsador  
                             // (abierto o cerrado)  
  
if (val == 1) { digitalWrite(LED, 1); } else { digitalWrite(LED, 0); }  
// si el valor de val es 1, da salida de 5 V a pin 2 y el diodo se enciende  
// si el valor de val es 0, da salida de 0 V a pin 2 y el diodo se apaga  
}
```

Una vez introducido el código y guardado el sketch como practica\_02, se conecta el ordenador con el Arduino mediante el cable USB y se descarga el sketch.

### 1.5.1 Mediciones práctica 02

Medir tensión en la salida digital 2 con el pulsador accionado y sin accionar.

A continuación se hace un vídeo mostrando el funcionamiento del circuito (encendido y apagado del LED y valor de tensión en la salida digital 2, pulsando y sin pulsar). El vídeo debe mandarse a [pposada@cifpnauticopesquera.es](mailto:pposada@cifpnauticopesquera.es)

### 1.5.2 If ... else

**If** (condición) { ← esta llave marca el inicio el grupo de instrucciones de **if**  
instrucción 1;  
instrucción 2;  
} ← esta llave marca el final del grupo de instrucciones de **if**  
**else** { ← esta llave marca el inicio el grupo de instrucciones de **else**  
instrucción 3;  
instrucción 4;  
} ← esta llave marca el final del grupo de instrucciones de **else**

El comando **if** (si) siempre lleva asociada una condición. Si la condición se cumple ejecuta las instrucciones agrupadas por las llaves.

Si la condición no se cumple, salta a **else** (si no) y se ejecutan las instrucciones agrupadas en las llaves correspondientes a **else**.



En el sketch de la práctica 02 aparece el siguiente uso del comando **if**:

```
if (val == 1) { digitalWrite(LED, 1); } else { digitalWrite(LED, 0); }
```

La condición que se tiene que cumplir, para que se ejecute la instrucción de **if**

```
digitalWrite(LED, 1);      //enciende el LED, instrucción de if  
es (val == 1).
```

Si (val == 0), se salta a **else**, sin ejecutar la instrucción de **if** y se ejecuta la instrucción de **else**.

```
digitalWrite(LED, 0);      //apaga el LED, instrucción de else
```

Para comparar dos valores se utiliza el símbolo ==, mientras que para asignar un valor, por ejemplo a una constante o a una variable, se utiliza el símbolo =.

El comando **if** puede aparecer sin else. En este caso, se ejecutarán las instrucciones correspondientes a **if**, si se cumple la condición. Si no se cumple, se saltarán las instrucciones de **if** y se ejecutará la instrucción que siga tras la llave que marca el final de **if**.

## **1.6 Práctica 03: LED encendido por un pulsador con función de interruptor**

En esta práctica se enciende o apaga un LED accionando un pulsador. Con cada pulsación se cambia el estado del LED entre apagado y encendido.

Para mantener encendido el LED tras una pulsación será necesario algún tipo de memoria, que “recuerde” que el pulsador ha sido accionado una vez que haya dejado de pulsarse. Para recordar una pulsación se utilizará una variable. Al declarar una variable, se reserva un lugar en la memoria del Arduino, donde se podrá guardar un valor. Es necesario indicar de qué tipo de valor se trata, si de un número entero (integer), o texto

Estos apuntes son una adaptación de “[Getting Started with Arduino](#)”, autor Massimo Banzi.