



Universidad de Granada

decsai.ugr.es

Teoría de la Información y la Codificación

Grado en Ingeniería Informática

Seminario - Introducción a Arduino.



DECSAI

**Departamento de Ciencias de la
Computación e Inteligencia Artificial**



UNIVERSIDAD
DE GRANADA

Teoría de la Información y la Codificación

Grado en Ingeniería Informática

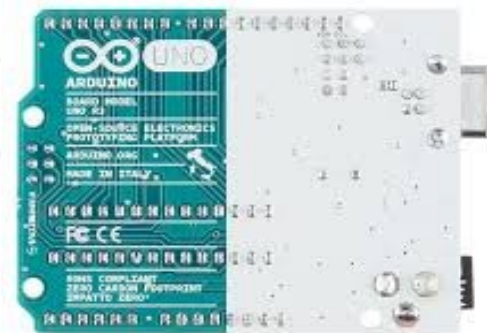


1. ¿Qué es Arduino?
2. Instalación para las prácticas
3. Comunicaciones en serie
4. Dispositivos GPIO



DECSAI

- **Arduino** es un conjunto de placas controladoras y un entorno de programación, Open Hardware y Open Software.
- Facilitan la elaboración de proyectos de electrónica, automatismo, control, domótica, etc.
- Existen varios modelos de Arduino como son Uno, Leonardo, Mega...
- En el laboratorio utilizaremos el modelo **Arduino Uno**, por ser el más económico, el de mayor flexibilidad y posibilidades en proporción a su precio, y porque tiene la capacidad suficiente para la construcción de los prototipos de las prácticas.



- El desarrollo de proyectos con Arduino conlleva el uso de uno o varios elementos que se integran en la placa: entradas, salidas, alimentación, comunicación y shields de extensión.
 - **Entradas:** son pines incrustados en la placa. Se utilizan para adquirir datos desde sensores u otros dispositivos externos. En la placa Arduino Uno son los pines digitales (del 0 al 13) y los analógicos (del A0 al A5).
 - **Salidas:** los pines de salidas se utilizan para el envío de datos a dispositivos externos o a actuadores. En este caso los pines de salida son los pines digitales (0 a 13), que pueden configurarse como entrada o como salida.
 - **Otros pines de interés:** TX (transmisión) y RX (lectura) también usados para comunicación en serie, RESET para resetear el sistema, Vin para alimentar la placa con fuentes de alimentación externa, y los pines ICSP para comunicación por puerto SPI.

- El desarrollo de proyectos con Arduino conlleva el uso de uno o varios elementos que se integran en la placa: entradas, salidas, alimentación, comunicación y shields de extensión.
- **Alimentación:** Considerados para la alimentación de sensores y actuadores, tales como los pines GND (del inglés GROUND, tierra), pines 5V que proporcionan 5 Voltios, pines 3.3V que proporciona 3.3 Voltios, los pines REF de referencia de voltaje. El pin Vin sirve para alimentar la placa de forma externa, aunque lo normal es alimentarlo por USB o por el conector de alimentación usando un voltaje de 5 a 12 Voltios.
- **Comunicación:** Lo más normal es utilizar comunicación serie por USB para cargar los programas en la placa o para enviar/recibir datos. No obstante, existen shields que permiten extender la capacidad de comunicación de la placa utilizando los pines ICSP (comunicación ISP), los pines 10 a 13 (también usados para comunicación ISP), los pines TX/RX o cualquiera de los digitales.

- El desarrollo de proyectos con Arduino conlleva el uso de uno o varios elementos que se integran en la placa: entradas, salidas, alimentación, comunicación y shields de extensión.
- **Shields (placas de expansión):** Son otras placas externas que se insertan sobre Arduino para extender sus capacidades. Algunas de las más comunes son las de Wi-Fi, sensores, actuadores (motores), Pantallas LCD, relés, matrices LED's, GPS, etc.



– Especificaciones técnicas:



Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
LED_BUILTIN	13
Length	68.6 mm
Width	53.4 mm
Weight	25 g

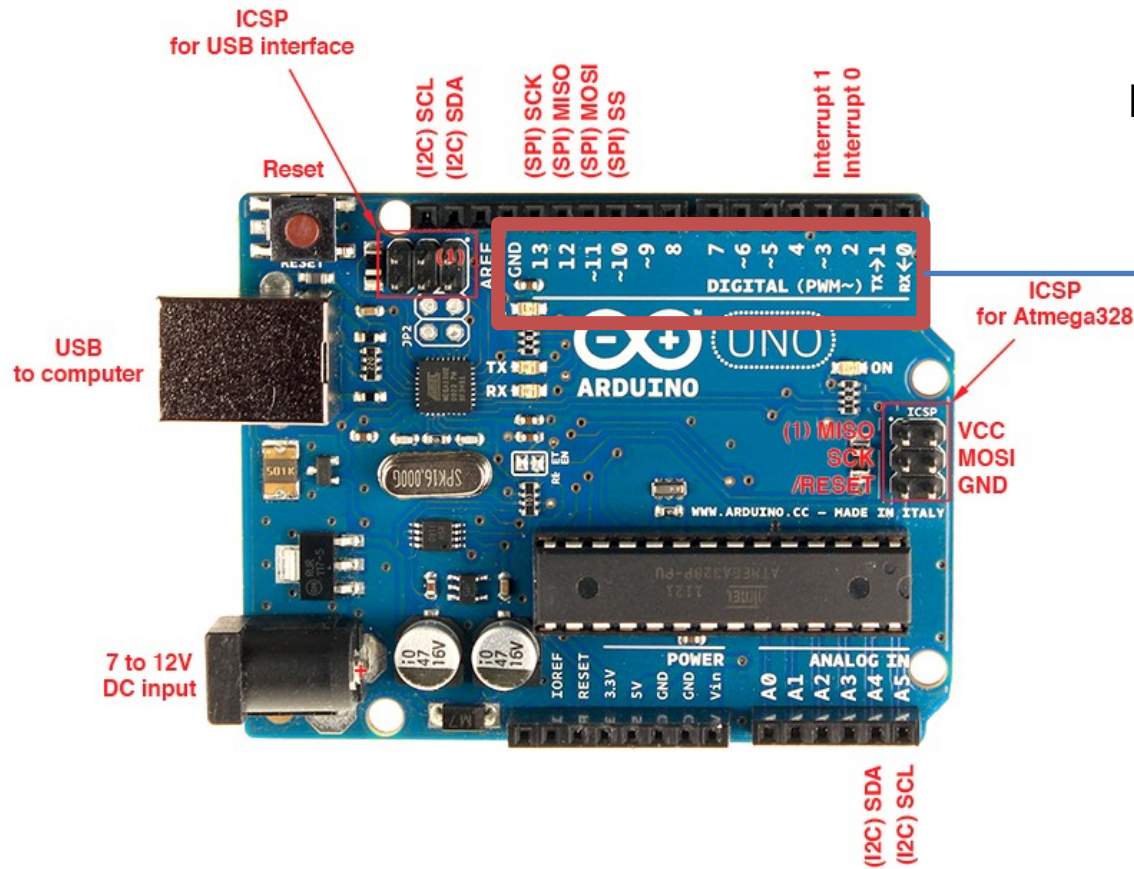
– Especificaciones técnicas:



- **Sólo 2KB de memoria RAM** (menos lo que quiten bibliotecas específicas)
- **Sólo 32KB de tamaño de programa** (memoria de sólo lectura)
- **A 16 MHz**

Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
LED_BUILTIN	13
Length	68.6 mm
Width	53.4 mm
Weight	25 g

- Mapa de pines de la placa Arduino Uno (estándar):



Pines de entrada/salida (I/O) digitales a usar en las prácticas

– Arduino es seguro, pero también puede romperse. Aquí enunciamos **las 8 formas más comunes de destruir una placa Arduino**:

1. Conectar dos pines entre es arriesgado si no se sabe qué se hace. sí Especialmente si uno es tierra y el otro voltaje, la placa quedará dañada.
2. Aplicar un voltaje superior a 5.5V a cualquier pin de entrada/salida.
3. Al usar alimentación por Vin, invertir la corriente (conectar el positivo al negativo y viceversa).
4. Conectar un voltaje superior al requerido en los pines de voltaje (por ejemplo, conectar 7V al pin de 5V o conectar 5V al pin de 3.3V).
5. Conectar el voltaje directamente a tierra.
6. Aplicar dos entradas de voltaje diferente para alimentar la placa (entrada externa Vin y entrada externa por el conector de alimentación).
7. Aplicar un voltaje superior a 13V al pin RESET.
8. Incluir en la placa una corriente superior a la soportada (la suma total de toda la corriente incluida en todos los pines de entrada/salida no puede superar los 200mA).

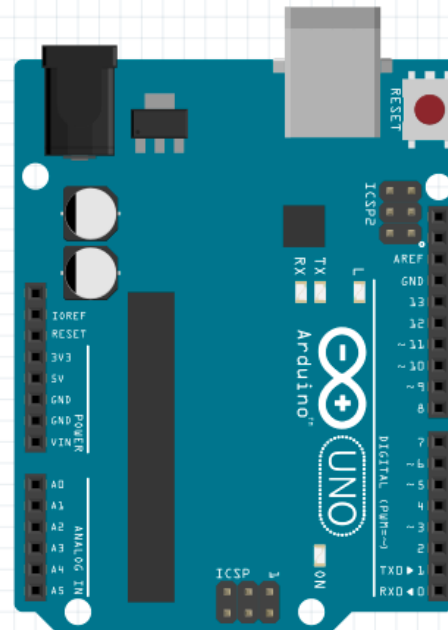
- La programación en Arduino se realiza en C. Las entradas y salidas se efectúan sobre los pines de la placa. Cada pin va asociado a una patilla del microcontrolador.
- **Aparte de las bibliotecas estándar, necesitaremos las bibliotecas específicas para microcontroladores AVR.**
- La correspondencia de los puertos con los pines de la placa Arduino Uno es la siguiente:

MCU : Atmega 328
Input voltage : 7V-12V
Operating voltage : 5V
CPU Speed : 16MHZ
Analog In/Out : 6/0
Digital IO/PWM : 14/6
EEPROM : 1KB
SRAM : 2KB
Flash : 32KB
UART : 1

Atmega168 Pin Mapping

Arduino function	Pin	MCU Pin	MCU Function	Arduino function
reset	1	PC6	(PCINT14/RESET)	analog input 5
digital pin 0 (RX)	2	PD0	(PCINT16/RXD)	analog input 4
digital pin 1 (TX)	3	PD1	(PCINT17/TXD)	analog input 3
digital pin 2	4	PD2	(PCINT18/INT0)	analog input 2
digital pin 3 (PWM)	5	PD3	(PCINT19/OC2B/INT1)	analog input 1
digital pin 4	6	PD4	(PCINT20/XCK/T0)	analog input 0
VCC	7	VCC		GND
GND	8	GND		analog reference
crystal	9	PB6	(PCINT6/XTAL1/TOSC1)	VCC
crystal	10	PB7	(PCINT7/XTAL2/TOSC2)	digital pin 13
digital pin 5 (PWM)	11	PD5	(PCINT21/OC0B/T1)	digital pin 12
digital pin 6 (PWM)	12	PD6	(PCINT22/OC0A/AIN0)	digital pin 11 (PWM)
digital pin 7	13	PD7	(PCINT23/AIN1)	digital pin 10 (PWM)
digital pin 8	14	PB0	(PCINT0/CLK0/ICP1)	digital pin 9 (PWM)
		15	PB1 (OC1A/PCINT1)	
		16	PB2 (SS/OC1B/PCINT2)	
		17	PB3 (MOSI/OC2A/PCINT3)	
		18	PB4 (MISO/PCINT4)	
		19	PB5 (SCK/PCINT5)	
		20	AVCC	
		21	AREF	
		22	GND	
		23	PC0 (ADC0/PCINT8)	
		24	PC1 (ADC1/PCINT9)	
		25	PC2 (ADC2/PCINT10)	
		26	PC3 (ADC3/PCINT11)	
		27	PC4 (ADC4/SDA/PCINT12)	
		28	PC5 (ADC5/SCL/PCINT13)	

Digital Pins 11, 12 & 13 are used by the ICSP header for MOSI, MISO, SCK connections (Atmega168 pins 17, 18 & 19). Avoid low-impedance loads on these pins when using the ICSP header.



ARDUINO PIN		MICROCONTROLLER PIN
0	-	PD0(RXD)
1	-	PD1(TXD)
2	-	PD2(INT0)
3	-	PD3(INT1)
4	-	PD4
5	-	PD5
6	-	PD6
7	-	PD7
8	-	PB0
9	-	PB1
10	-	PB2(SS')
11	-	PB3(MOSI)
12	-	PB4(MISO)
13	-	PB5(SCK)
A0	-	PC0
A1	-	PC1
A2	-	PC2
A3	-	PC3
A4	-	PC4(SDA)
A5	-	PC5(SCL)

Programar para AVR en C no es diferente de programar para Windows o para Linux. Sólo hacen falta las bibliotecas necesarias:

– **Bibliotecas y macros útiles:**

- **Biblioteca <avr/io.h>**: Contiene las definiciones de variables para direccionar puertos.
- **Biblioteca <util/delay.h>**: Contiene las definiciones de funciones para para la ejecución del programa por los instantes de tiempo requeridos.
- Macro **F_CPU**: Indica cada cuánto tiempo se refresca el tick del procesador.
- Macro **#define _BV(bit) (1 << (bit))**. Definida en <avr/io.h>: Se utiliza para transformar un bit a byte.

Un programa tipo en C para Arduino UNO comenzaría así:

```
// Utilizado para que el procesador pueda calcular el delay a partir del número de ticks del procesador
#define F_CPU 1000000UL

#include <avr/io.h>
#include <util/delay.h>
```

- Un ejemplo de programa en un fichero **main.cpp**:

```
// Utilizado para el cálculo de ms en _delay_ms
#define F_CPU 1000000UL

#include <avr/io.h>
#include <util/delay.h>

#define BLINK_DELAY_MS 1000

int main (void)
{
    /* Pin 0 del puerto B del micro puesto como salida */
    DDRB |= _BV(DDB0);

    while(1) {
        /* Mandamos señal de voltaje alto al pin 0 del puerto B */
        PORTB |= _BV(PORTB0);
        _delay_ms(BLINK_DELAY_MS);

        /* Mandamos señal de voltaje bajo al pin 0 del puerto B */
        PORTB &= ~_BV(PORTB0); // ~ es el NOT lógico a nivel de bits
        _delay_ms(BLINK_DELAY_MS);
    }
}
```

Qué vemos nuevo:

- Operadores a nivel de bits de C/C++:
 - $a \mid b$: Realiza el or a nivel de bits de a y b . Ejemplo:

unsigned char $a=1$, $b=4$, $c=a \mid b$; // c vale 5: 00000101

– 00000001 | 00000100 = 00000101

- $a \& b$: Realiza el and a nivel de bits de a y b . Ejemplo:

unsigned char $a=3$, $b=1$, $c=a \& b$; // c vale 1: 00000001

– 00000011 & 00000001 = 00000001

Los operadores tienen sus correspondientes $|=$, $\&=$.

- Operadores a nivel de bits de C/C++:
 - $\sim b$: Realiza el not a nivel de bits de b . Ejemplo:

unsigned char a= 1, c= \sim a; // c vale 254: 11111110

– **$\sim 00000001 = 11111110$**

- $A \ll n$: Desplaza todos los bits de A , n posiciones hacia la izquierda, introduciendo n 0's por la derecha.

– **$A = 1$; // 0b00000001 $A \ll 2$; // 0b00000100**

- $A \gg n$: Desplaza todos los bits de A , n posiciones hacia la izquierda, introduciendo n 0's por la derecha.

– **$A = 3$; // 0b00000011 $A \gg 1$; // 0b00000001**

- Operadores a nivel de bits de C/C++:
 - ¿Cómo poner un bit n a 1? Haciendo un OR a nivel de bits del dato con ese bit

unsigned char a= 8; a |= (1<<4); // a vale 00001000,

1<<4=00010000

Resultado: a=00011000

- ¿Cómo poner un bit n a 0? Haciendo un AND a nivel de bits del dato con el complementario de ese bit

unsigned char a= 12; a&= ~(1<<2); // a vale 00001100,

1<<2=00000100

~(1<<2)=11111011

Resultado: a=00001000

— Ejemplo de programa en C para AVR

```
// Utilizado para el cálculo de ms en _delay_ms
#define F_CPU 1000000UL
```

```
#include <avr/io.h>
#include <util/delay.h>
```

```
#define BLINK_DELAY_MS 1000
```

```
int main (void)
{
    /* Pin 0 del puerto B del micro puesto como salida */
    DDRB |= _BV(DDB0);

    while(1) {
        /* Mandamos señal de voltaje alto al pin 0 del puerto B */
        PORTB |= _BV(PORTB0);
        _delay_ms(BLINK_DELAY_MS);

        /* Mandamos señal de voltaje bajo al pin 0 del puerto B */
        PORTB &= ~_BV(PORTB0); // ~ es el NOT lógico a nivel de bits
        _delay_ms(BLINK_DELAY_MS);
    }
}
```

Atmega168 Pin Mapping

Arduino function				Arduino function	
reset	(PCINT14/RESET) PC6	1	28	PC5 (ADC5/SCL/PCINT13)	analog input 5
digital pin 0 (RX)	(PCINT16/RXD) PD0	2	27	PC4 (ADC4/SDA/PCINT12)	analog input 4
digital pin 1 (TX)	(PCINT17/TXD) PD1	3	26	PC3 (ADC3/PCINT11)	analog input 3
digital pin 2	(PCINT18/INT0) PD2	4	25	PC2 (ADC2/PCINT10)	analog input 2
digital pin 3 (PWM)	(PCINT19/OC2B/INT1) PD3	5	24	PC1 (ADC1/PCINT9)	analog input 1
digital pin 4	(PCINT20/XCK/T0) PD4	6	23	PC0 (ADC0/PCINT8)	analog input 0
VCC	VCC	7	22	GND	GND
GND	GND	8	21	AREF	analog reference
crystal	(PCINT6/XTAL1/TOSC1) PB6	9	20	AVCC	VCC
crystal	(PCINT7/XTAL2/TOSC2) PB7	10	19	PB5 (SCK/PCINT5)	digital pin 13
digital pin 5 (PWM)	(PCINT21/OC0B/T1) PD5	11	18	PB4 (MISO/PCINT4)	digital pin 12
digital pin 6 (PWM)	(PCINT22/OC0A/AIN0) PD6	12	17	PB3 (MOSI/OC2A/PCINT3)	digital pin 11 (PWM)
digital pin 7	(PCINT23/AIN1) PD7	13	16	PB2 (SS/OC1B/PCINT2)	digital pin 10 (PWM)
digital pin 8	(PCINT0/CLKO/CP1) PB0	14	15	PB1 (OC1A/PCINT1)	digital pin 9 (PWM)

Digital Pins 11, 12 & 13 are used by the ICSP header for MOSI, MISO, SCK connections (Atmega168 pins 17, 18 & 19). Avoid low-impedance loads on these pins when using the ICSP header.

— Pasos para la compilación y envío del programa a la placa:

1. Compilación:

```
avr-gcc -Os -mmcu=atmega328p -c -o main.o main.cpp
avr-gcc -mmcu=atmega328p main.o -o main
avr-objcopy -O ihex -R .eeprom main main.hex
```

2. Búsqueda del puerto de conexión de Arduino Uno *(nota: Arduino Uno tiene que estar conectado al PC por el puerto USB):*

```
lsusb
ls /dev/serial/by-id -l
```

3. Envío del programa a Arduino Uno:

```
sudo avrdude -F -V -c arduino -p ATMEGA328P -P /dev/ttyACM0 -b 115200 -U flash:w:main.hex
```

4. Envío para Windows: En lugar de /dev/ttyACM0, será COM2, COM3...

5. Envío para MAC: En lugar de /dev/ttyACM0, será /dev/tty.usbmodel1411, /dev/cu.usbmodel1411, ...



UNIVERSIDAD
DE GRANADA

Teoría de la Información y la Codificación

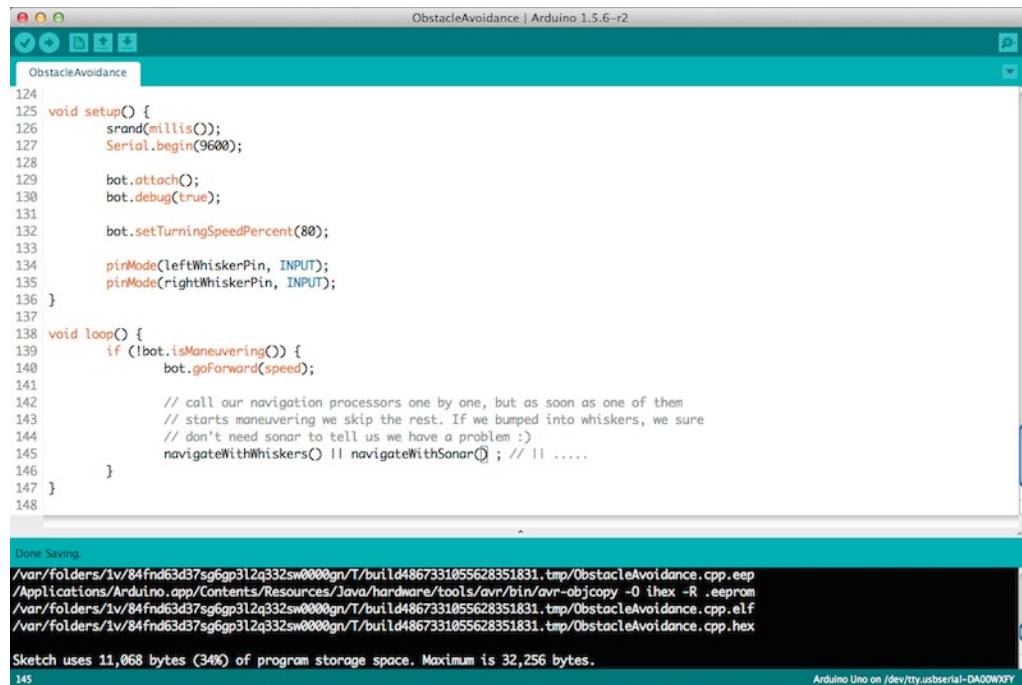
Grado en Ingeniería Informática

1. ¿Qué es Arduino?
- » 2. Instalación para las prácticas
3. Comunicaciones en serie
4. Dispositivos GPIO



DECSAI

- Arduino tiene su propio IDE de programación, con código C. Es el utilizado principalmente por aquellos que se quieren iniciar en Arduino como hobbie. No es un entorno profesional, aunque otros muchos microcontroladores profesionales han incorporado sus productos dentro de esta plataforma:



```

ObstacleAvoidance | Arduino 1.5.6-r2

ObstacleAvoidance
124
125 void setup() {
126     srand(millis());
127     Serial.begin(9600);
128
129     bot.attach();
130     bot.debug(true);
131
132     bot.setTurningSpeedPercent(80);
133
134     pinMode(leftWhiskerPin, INPUT);
135     pinMode(rightWhiskerPin, INPUT);
136 }
137
138 void loop() {
139     if (!bot.isManeuvering()) {
140         bot.goForward(speed);
141
142         // call our navigation processors one by one, but as soon as one of them
143         // starts maneuvering we skip the rest. If we bumped into whiskers, we sure
144         // don't need sonar to tell us we have a problem :)
145         navigateWithWhiskers() || navigateWithSonar(); // || .....
146     }
147 }
148

Done Saving.
/var/folders/1v/84fnd63d37sg6gp3l2q332sw00000gn/T/build4867331055628351831.tmp/ObstacleAvoidance.cpp.eep
/var/folders/1v/84fnd63d37sg6gp3l2q332sw00000gn/T/build4867331055628351831.tmp/ObstacleAvoidance.cpp.elf
/var/folders/1v/84fnd63d37sg6gp3l2q332sw00000gn/T/build4867331055628351831.tmp/ObstacleAvoidance.cpp.hex

Sketch uses 11,068 bytes (34%) of program storage space. Maximum is 32,256 bytes.
145
Arduino Uno on /dev/tty.usbserial-DA00WXY

```


- Lo primero que haremos será instalar el IDE de Arduino:

Usando gestores de paquetes (Ubuntu)
Desde web (demás sistema)

En Ubuntu:


`sudo apt-get install arduino`

- Para descargar el IDE de Arduino desde la web:

<https://www.arduino.cc/en/main/software>



Download the Arduino IDE



ARDUINO 1.8.5

The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open-source software.

This software can be used with any Arduino board. Refer to the [Getting Started](#) page for Installation Instructions.

Windows Installer, for Windows XP and up
Windows ZIP file for non admin install

Windows app Requires Win 8.1 or 10


Mac OS X 10.7 Lion or newer

Linux 32 bits
Linux 64 bits
Linux ARM

[Release Notes](#)
[Source Code](#)
[Checksums \(sha512\)](#)

HOURLY BUILDS

Download a **preview of the incoming release** with the most updated features and bugfixes.

BETA BUILDS

Download the **Beta Version** of the Arduino IDE with experimental features. This version should NOT be used in production.

- **Al instalar el IDE se abren automáticamente los puertos y permisos necesarios, en Windows**

– Post-instalación (Linux y MAC):

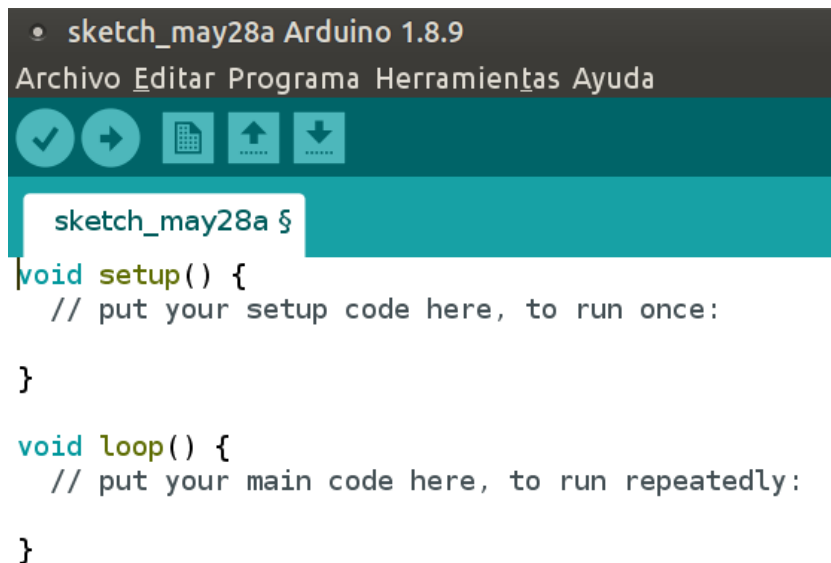
Si has instalado Arduino con apt-get, no tienes que hacer nada.

En caso contrario, **debes añadirte al grupo dialout:**

```
sudo adduser tu_usuario dialout
```

Ahora reinicia tu PC para que los cambios surtan efecto

- Al iniciar el IDE por primera vez, aparece una ventana como la siguiente:



```

• sketch_may28a Arduino 1.8.9
Archivo Editar Programa Herramientas Ayuda
[Checkmark] [Next] [File] [Up Arrow] [Down Arrow]
sketch_may28a §
void setup() {
  // put your setup code here, to run once:
}

void loop() {
  // put your main code here, to run repeatedly:
}

```

- Al iniciar el IDE por primera vez, aparece una ventana como la siguiente:



```

• sketch_may28a Arduino 1.8.9
Archivo Editar Programa Herramientas Ayuda

[Checkmark] [Right Arrow] [Document] [Upload Arrow] [Download Arrow]

sketch_may28a §

void setup() {
  // put your setup code here, to run once:
}

void loop() {
  // put your main code here, to run repeatedly:
}

```

- **Conectemos la placa Arduino con el cable USB al PC**
- **Pulsando sobre *Herramientas*→*Placa*, deberemos tener seleccionada la placa *Arduino/Genuino Uno* (es la placa de prácticas).**

- Al iniciar el IDE por primera vez, aparece una ventana como la siguiente:



```

• sketch_may28a Arduino 1.8.9
Archivo Editar Programa Herramientas Ayuda

[Checkmark] [Next] [File] [Up Arrow] [Down Arrow]

sketch_may28a §

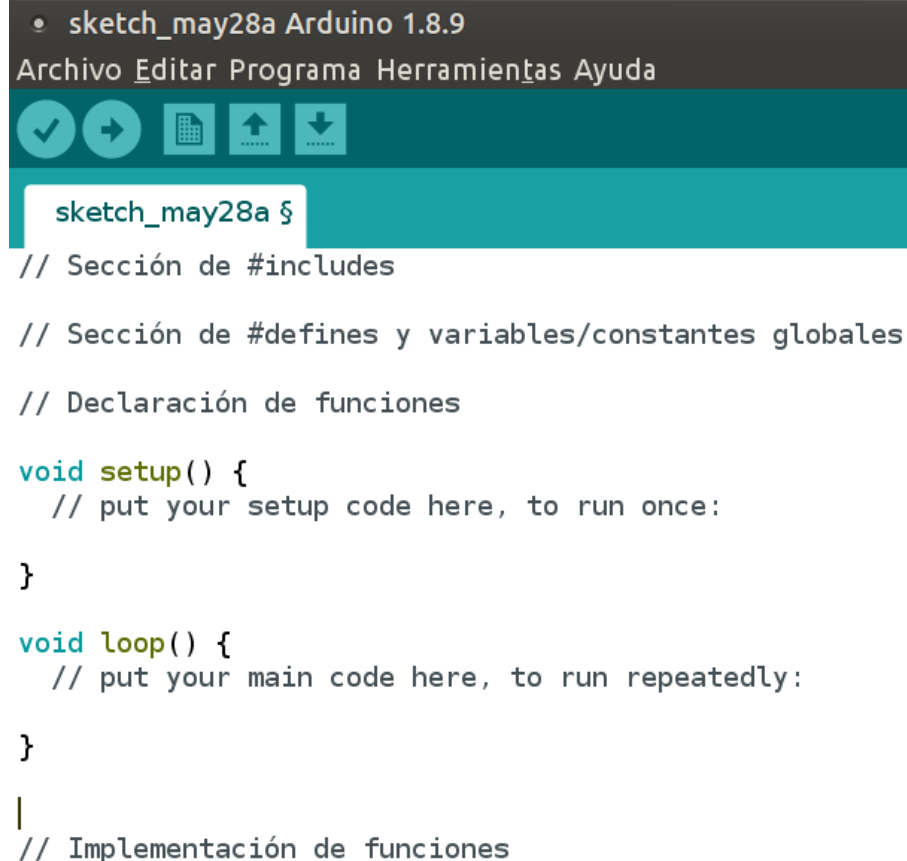
void setup() {
  // put your setup code here, to run once:
}

void loop() {
  // put your main code here, to run repeatedly:
}

```

- Pulsando sobre **Herramientas→Puerto**, deberemos tener seleccionado el puerto donde se encuentra conectado Arduino (normalmente, aparecerán sólo las opciones de puertos disponibles).

- Normalmente, la estructura de un programa Arduino será la siguiente, organizando todo el código en el fichero + bibliotecas:



```

• sketch_may28a Arduino 1.8.9
Archivo Editar Programa Herramientas Ayuda
[Checkmark] [Next] [File] [Up] [Down]
sketch_may28a §
// Sección de #includes

// Sección de #defines y variables/constantes globales

// Declaración de funciones

void setup() {
    // put your setup code here, to run once:
}

void loop() {
    // put your main code here, to run repeatedly:
}

|
// Implementación de funciones
  
```

El IDE de Arduino, con el programa de código de ejemplo genera automáticamente un programa **main.cpp** con la siguiente estructura:

```
// Sección de #includes

// Sección de #defines y variables/constantes globales

// Declaración de funciones

void setup();
void loop();

int main() {

    setup();

    while (true) {
        loop();
    }

    return 0;
}

// Implementación de funciones
```

El IDE de Arduino, con el programa de código de ejemplo genera automáticamente un programa **main.cpp** con la siguiente estructura:

```
// Sección de #includes
```

```
// Sección de #defines y variables/constantes globales
```

```
// Declaración de funciones
```

```
void setup();
void loop();
```

```
int main() {
```

```
    setup();
```

setup(): Función para inicializar variables, puertos, etc.

```
    while (true) {
```

```
        loop();
```

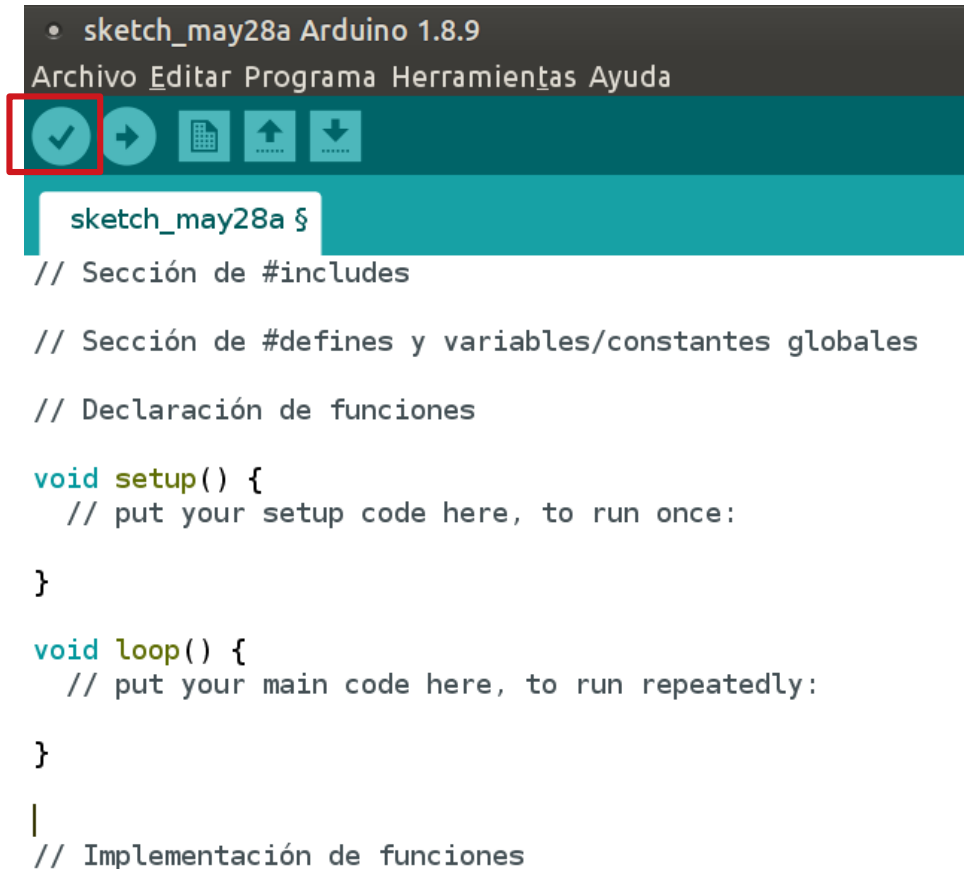
loop(): Función principal del programa, que se ejecuta en un bucle infinito

```
    }
    return 0;
```

```
}
```

```
// Implementación de funciones
```

- Compilación del programa en el PC: Pulsando sobre el botón de verificar programa.



```

• sketch_may28a Arduino 1.8.9
Archivo Editar Programa Herramientas Ayuda

[Verify] [Run] [New] [Open] [Save]

sketch_may28a §
// Sección de #includes

// Sección de #defines y variables/constantes globales

// Declaración de funciones

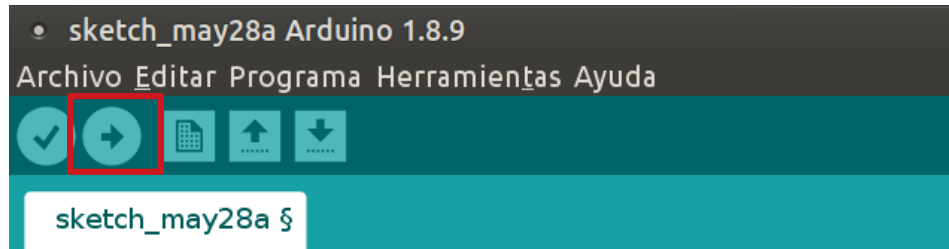
void setup() {
  // put your setup code here, to run once:
}

void loop() {
  // put your main code here, to run repeatedly:
}

|
// Implementación de funciones

```

- Compilación+envío+ejecución del programa en Arduino: Pulsando sobre el botón de enviar programa. **Una vez enviado, el programa comenzará a ejecutarse automáticamente en la plataforma Arduino.**



```

• sketch_may28a Arduino 1.8.9
Archivo Editar Programa Herramientas Ayuda
[Checkmark] [Upload] [New] [Open] [Save]
sketch_may28a $
// Sección de #includes

// Sección de #defines y variables/constantes globales

// Declaración de funciones

void setup() {
    // put your setup code here, to run once:
}

void loop() {
    // put your main code here, to run repeatedly:
}

|
// Implementación de funciones
  
```



UNIVERSIDAD
DE GRANADA

Teoría de la Información y la Codificación

Grado en Ingeniería Informática

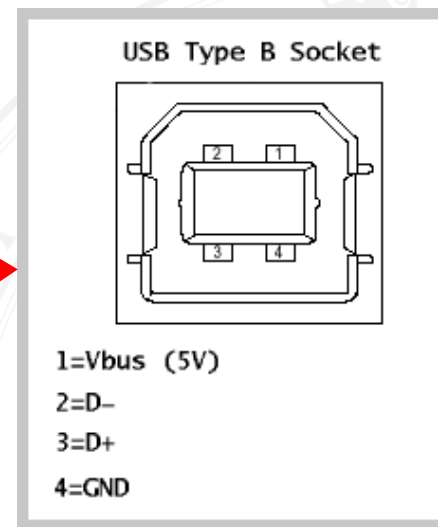
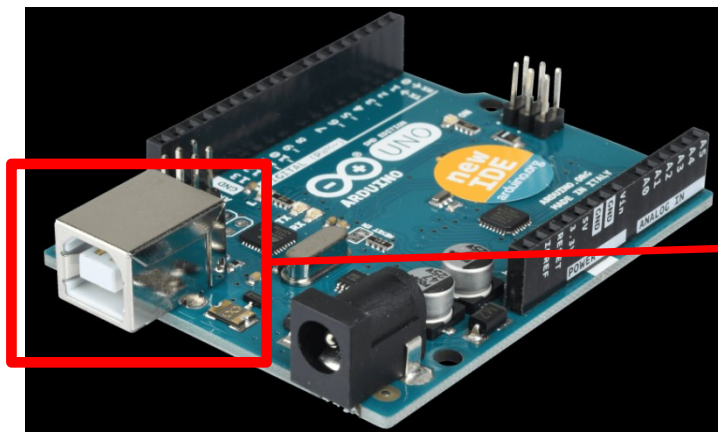
1. ¿Qué es Arduino?
2. Instalación para las prácticas
- » 3. Comunicaciones en serie
4. Dispositivos GPIO



DECSAI

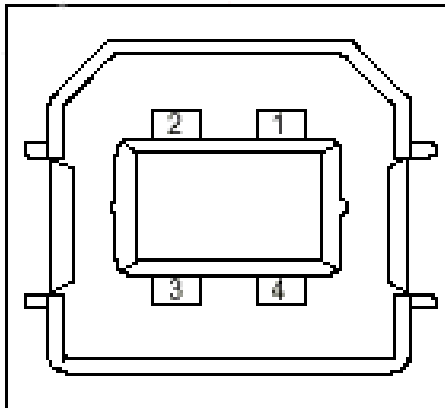
- Podemos comunicarnos con la placa Arduino de múltiples formas, aunque la mayoría de ellas requieren de placas de expansión (**shields**) adicionales:
 - Shield Wi-Fi
 - Shield Bluetooth
 - Pantallas táctiles
 - Etc.
- Todas las placas Arduino tienen posibilidad de comunicaciones por puerto serie (como mínimo un puerto serie). Así es como grabamos los programas en la memoria de la placa Arduino.
- Las **comunicaciones en serie** se realizan a través de **puertos UART**.

- El “**Bus Universal en Serie**” (**USB**) es un bus estándar que define las conexiones y protocolos para conectar, comunicar y alimentar periféricos y dispositivos electrónicos.
- En Arduino, **la comunicación por USB se realiza a través del puerto serie UART**. El conector utilizado en la placa es un **USB Tipo B**.



– Descripción del conector USB Tipo B:

USB Type B Socket



1=Vbus (5V)

2=D-

3=D+

4=GND

- **Pin 1:** Alimentación con un voltaje de 5V DC
- **Pines 2 y 3:** Transmisión de datos
- **Pin 4:** Toma de tierra

- La gestión del puerto UART en procesadores AVR (y en cualquier procesador, por regla general), requiere de una inicialización.
- Esta inicialización implica también indicar cuál es la velocidad (en baudios) a la que se va a trabajar con el puerto (tradicionalmente, la velocidad del puerto serie más estándar es 9600, aunque un puerto USB puede trabajar cómodamente a 115200. Dependiendo de qué plataforma se use, a mayores velocidades se puede obtener errores en las comunicaciones.
 - Además, como el puerto UART es gestionado a través de interrupciones hardware, es necesario activar el módulo de gestión de interrupciones SEI. **Las bibliotecas de Arduino realizan estas acciones automáticamente.**
- **Deberemos realizar estas operaciones sólo una vez al principio del programa (función `setup()`), e incluirlas en todos los programas que implementemos y que hagan uso de comunicaciones en serie por USB.**

- Las bibliotecas de Arduino proporcionan el objeto global **Serial**, para realizar comunicaciones en serie.

```
#ifndef __cplusplus
extern "C" {
#endif

extern SerialPort Serial;

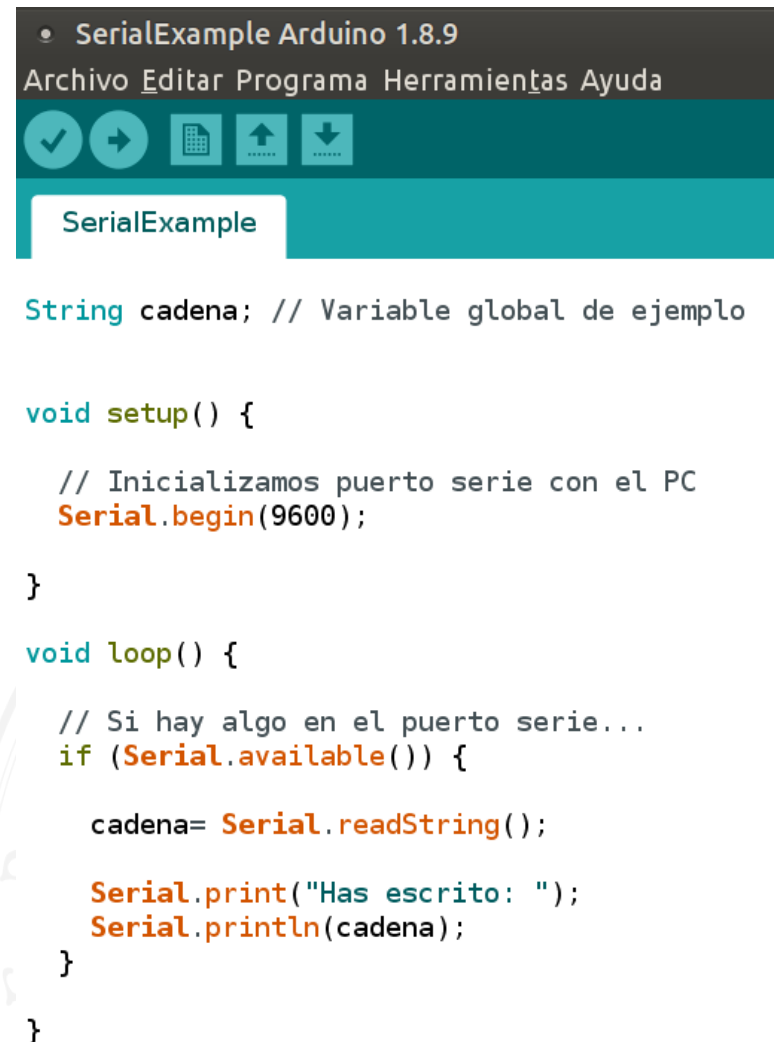
#ifdef __cplusplus
}
#endif
```

- Inicialización en la función **setup()**: **Serial.begin(baudios)**
- **Ejemplo**: **Serial.begin(9600);**

– Métodos útiles de Serial:

- **Serial.begin(baudios):** Inicializa las comunicaciones por el puerto serie (UART), a la velocidad dada en baudios. **Requerido en la función setup() si vamos a utilizar el objeto Serial en el programa.**
- **Serial.print("Cadena"):** Envía la cadena por comunicaciones serie.
- **Serial.println("Cadena"):** Envía la cadena por comunicaciones serie, y la termina con un '\n'.
- **uint8_t a= Serial.read():** Lee un byte desde comunicaciones serie.
- **Serial.readString():** Lee una cadena de caracteres desde puerto serie, y la devuelve en un objeto **String**. **OJO: String (con S mayúscula) es una clase propia de Arduino, similar a la clase string de C/C++. Pero no es la misma clase.**
- **bool a= Serial.available():** Devuelve true si hay datos disponibles para lectura por las comunicaciones serie. False en otro caso.

- Programa de ejemplo: Escribir el siguiente programa, y guardarlo con el nombre “SerialExample.ino” (la extensión .ino se guarda automáticamente por el IDE de Arduino):



```

SerialExample Arduino 1.8.9
Archivo Editar Programa Herramientas Ayuda

SerialExample

String cadena; // Variable global de ejemplo

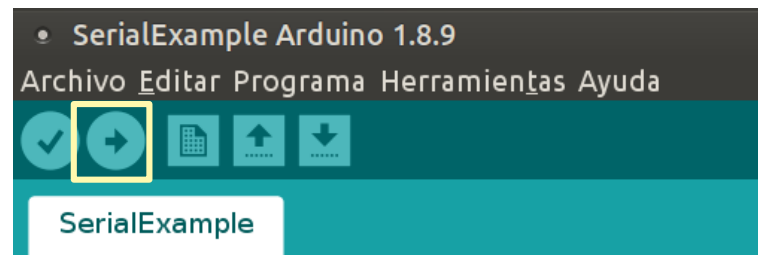
void setup() {
    // Inicializamos puerto serie con el PC
    Serial.begin(9600);
}

void loop() {
    // Si hay algo en el puerto serie...
    if (Serial.available()) {
        cadena= Serial.readString();

        Serial.print("Has escrito: ");
        Serial.println(cadena);
    }
}

```


- Compilaremos y enviaremos el programa a la plataforma. En cuanto se reciba (si hay éxito en el envío), empezará a ejecutarse automáticamente.



```
String cadena; // Variable global de ejemplo
```

```
void setup() {
```

```
    // Inicializamos puerto serie con el PC
    Serial.begin(9600);
```

```
}
```

```
void loop() {
```

```
    // Si hay algo en el puerto serie...
    if (Serial.available()) {
```

```
        cadena= Serial.readString();
```

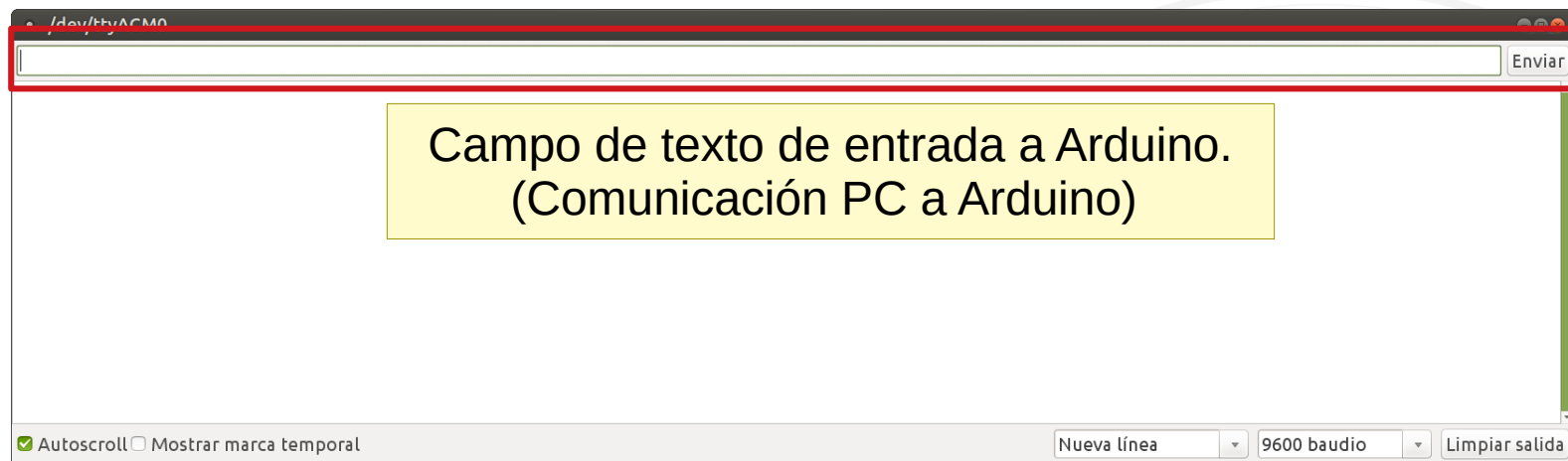
```
        Serial.print("Has escrito: ");
        Serial.println(cadena);
```

```
    }
```

```
}
```

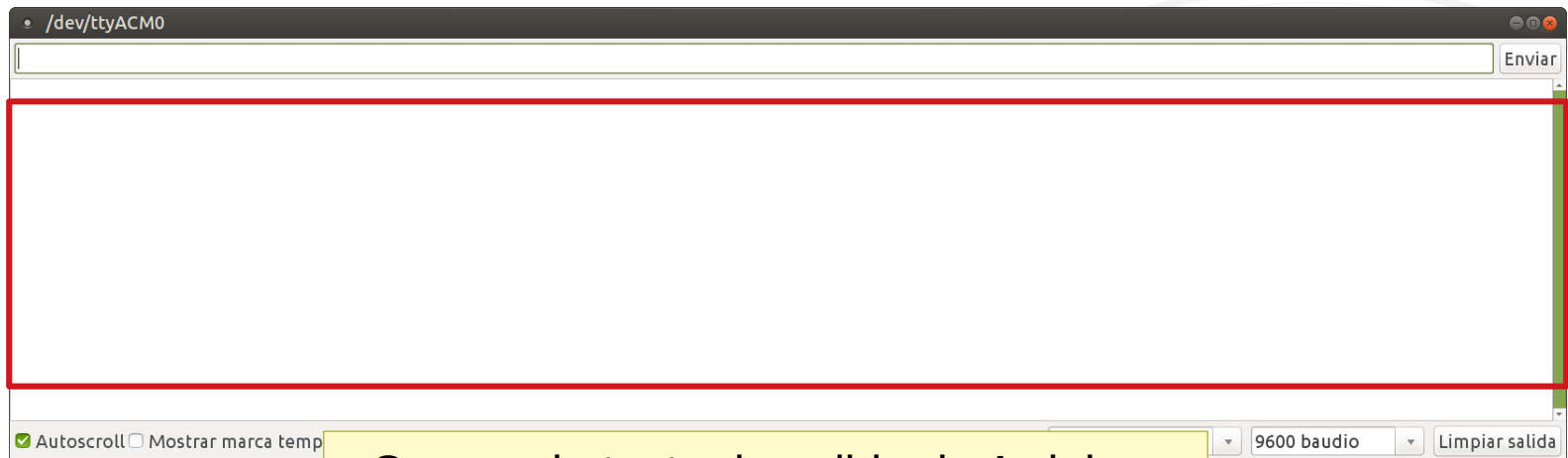
Para comunicarnos con Arduino por el puerto serie, es necesario abrir el **Monitor Serie (Serial Monitor)**. Se activa en las opciones de **Herramientas→Monitor Serie**.

- **IMPORTANTE:** Es necesario que en **Herramientas→Puerto** esté seleccionado el puerto correcto donde tenemos conectado a Arduino. En otro caso, el monitor serie no tendrá efecto.



Para comunicarnos con Arduino por el puerto serie, es necesario abrir el **Monitor Serie (Serial Monitor)**. Se activa en las opciones de **Herramientas→Monitor Serie**.

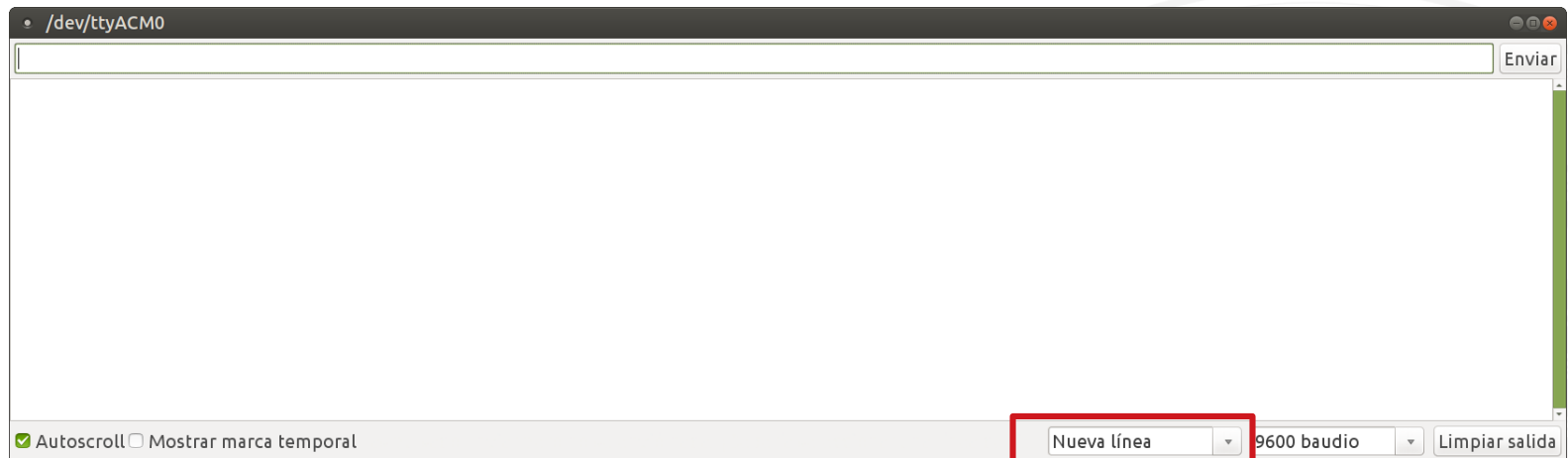
- **IMPORTANTE:** Es necesario que en **Herramientas→Puerto** esté seleccionado el puerto correcto donde tenemos conectado a Arduino. En otro caso, el monitor serie no tendrá efecto.



Campo de texto de salida de Arduino.
(Comunicación Arduino a PC)

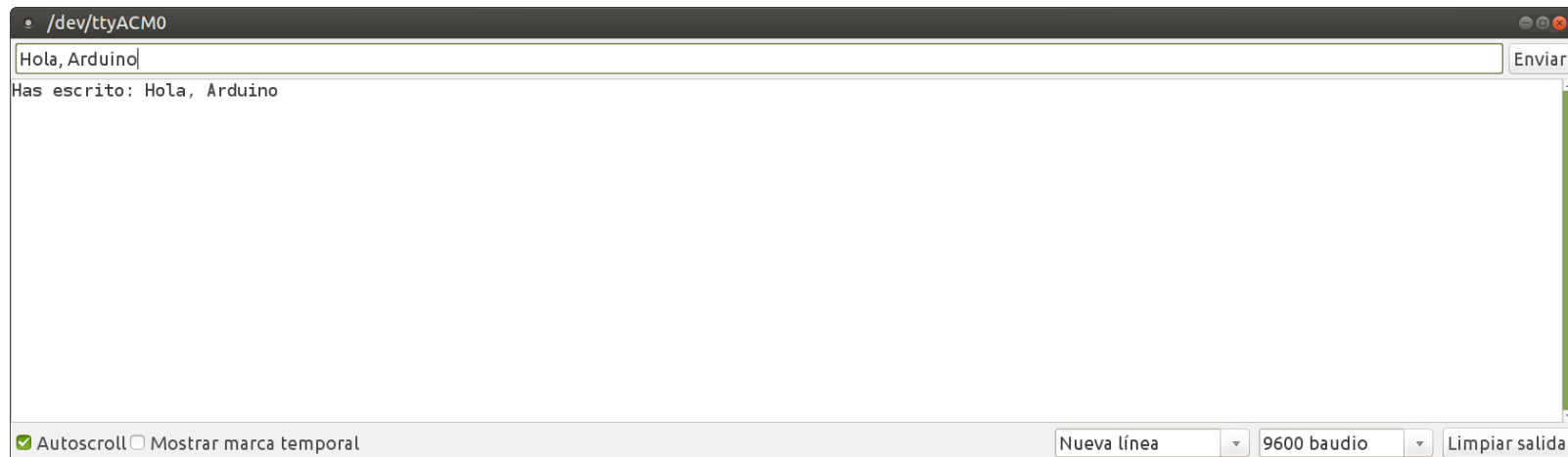
Para comunicarnos con Arduino por el puerto serie, es necesario abrir el **Monitor Serie (Serial Monitor)**. Se activa en las opciones de **Herramientas→Monitor Serie**.

- **IMPORTANTE:** Es necesario que en **Herramientas→Puerto** esté seleccionado el puerto correcto donde tenemos conectado a Arduino. En otro caso, el monitor serie no tendrá efecto.



Caracter a enviar automáticamente al final de la cadena. En las prácticas, no seleccionaremos ningún carácter de finalización.

Ejemplo: Escribir algo en el campo de texto y esperar a la respuesta de Arduino.



Si nada ocurre: ¿Has seleccionado correctamente el puerto de Arduino en *Herramientas* → *Puerto*? ¿Tienes conectado Arduino al PC? ¿Has enviado el programa de ejemplo?



UNIVERSIDAD
DE GRANADA

Teoría de la Información y la Codificación

Grado en Ingeniería Informática

1. ¿Qué es Arduino?
2. Instalación para las prácticas
3. Comunicaciones en serie
- » 4. Dispositivos GPIO



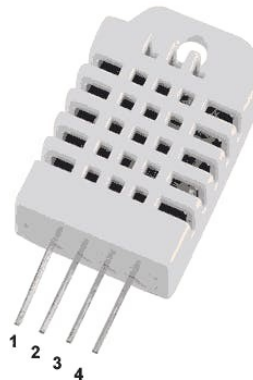
DECSAI

- **GPIO** = General Purpose Input/Output

Los pines de Arduino son GPIO, pines de entrada/salida de propósito general.

Se utilizan por sensores y actuadores para comunicarse con la placa Arduino

DHT22 pins	
1	VCC
2	DATA
3	NC
4	GND



Sensores: Se configuran como **INPUT**

Actuadores: Se configuran como **OUTPUT**

Algunas funciones de la API de Arduino para gestión de pines GPIO:

```
#define HIGH 0x1
#define LOW 0x0

#define INPUT 0x0
#define OUTPUT 0x1
#define INPUT_PULLUP 0x2
#define delay(ms) (_delay_ms(ms))

void pinMode(uint8_t, uint8_t);
void digitalWrite(uint8_t, uint8_t);
int digitalRead(uint8_t);
```



Ejemplos:

pinMode(número, OUTPUT) configura el pin **número** de Arduino como **SALIDA**.

pinMode(número, INPUT) configura el pin **número** de Arduino como **ENTRADA**.

Siempre hay que configurar los pines como entrada o como salida al principio de nuestro programa, en la función setup().

Algunas funciones de la API de Arduino para gestión de pines GPIO:

```
#define HIGH 0x1
#define LOW 0x0

#define INPUT 0x0
#define OUTPUT 0x1
#define INPUT_PULLUP 0x2
#define delay(ms) (_delay_ms(ms))

void pinMode(uint8_t, uint8_t);
void digitalWrite(uint8_t, uint8_t);
int digitalRead(uint8_t);
```



Ejemplos:

digitalWrite(número, HIGH) activa la salida por el pin número de Arduino (3.3v). Quedará así hasta que se desactive.

digitalWrite(8, LOW) desactiva la salida por el pin número de Arduino (0v).
Quedará así hasta que se active.

Sólo se debe aplicar digitalWrite sobre pines definidos como OUTPUT con pinMode

¡PELIGRO DE DAÑAR ARDUINO SI SE HACE DE OTRA FORMA!

Algunas funciones de la API de Arduino para gestión de pines GPIO:

```
#define HIGH 0x1
#define LOW 0x0

#define INPUT 0x0
#define OUTPUT 0x1
#define INPUT_PULLUP 0x2
#define delay(ms) (_delay_ms(ms))

void pinMode(uint8_t, uint8_t);
void digitalWrite(uint8_t, uint8_t);
int digitalRead(uint8_t);
```



Ejemplos:

int a= digitalRead(número) Lee valor de entrada (HIGH/LOW) que haya en el pin **número** (por ejemplo, desde un sensor), y lo devuelve.

Sólo se puede aplicar digitalRead sobre pines definidos como INPUT con pinMode

Algunas funciones de la API de Arduino para gestión de pines GPIO:

```
#define HIGH 0x1
#define LOW 0x0

#define INPUT 0x0
#define OUTPUT 0x1
#define INPUT_PULLUP 0x2
#define delay(ms) (_delay_ms(ms))

void pinMode(uint8_t, uint8_t);
void digitalWrite(uint8_t, uint8_t);
int digitalRead(uint8_t);
```

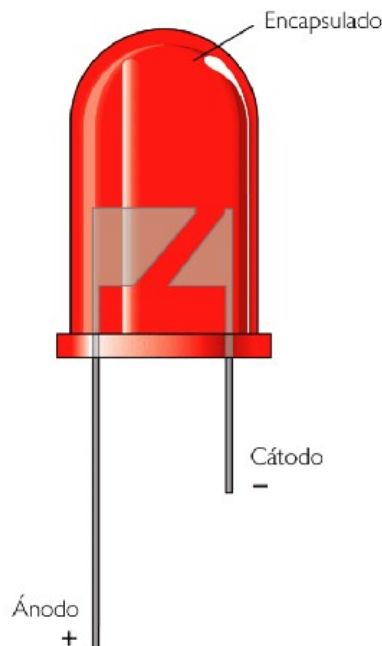


Ejemplos:

delay(1000) Provoca el bloqueo del programa por 1000 ms.

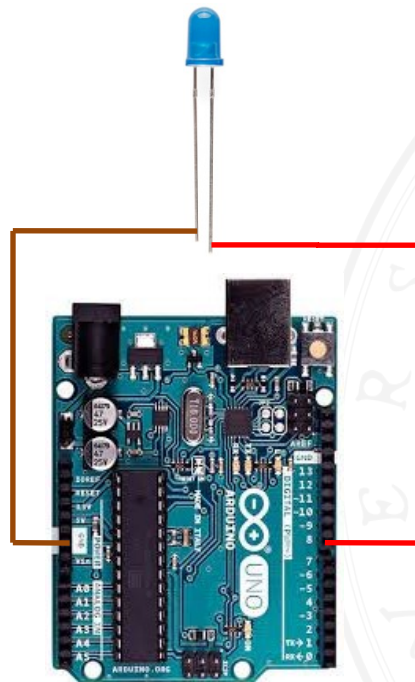
Prueba de funcionamiento de actuadores:

1. **Seleccione un diodo LED en el laboratorio, y 2 cables de conexión macho-hembra.**
2. **Localice el cátodo (polo negativo, alambre corto) y el ánodo (polo positivo, alambre largo) del LED.**



Prueba de funcionamiento de actuadores:

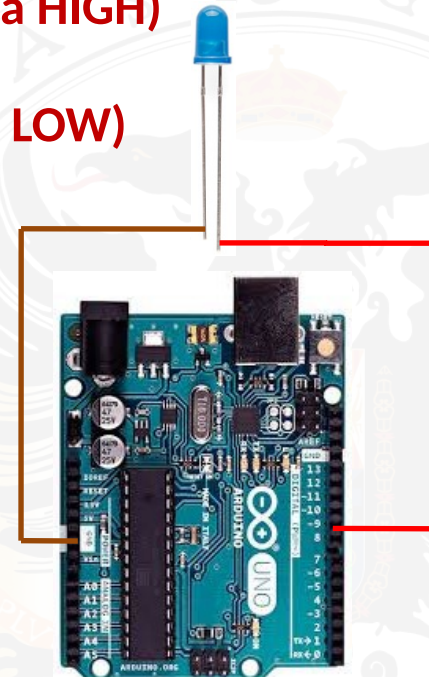
3. **Conecte el extremo hembra de un cable al cátodo del LED, y el extremo macho a un pin GND de la placa Arduino.**
4. **Conecte el extremo hembra de otro cable al ánodo del LED, y el extremo macho al **PIN DIGITAL 8**.**



– Prueba de funcionamiento:

Escriba un programa que:

- 1. Defina el pin 8 como salida (función setup)**
- 2. El código de la función *loop()* deberá hacer lo siguiente**
 - 1. Encender el LED (digital Write a HIGH)**
 - 2. Espere 1 segundo**
 - 3. Apague el LED (digital Write a LOW)**
 - 4. Espere 1 segundo**



- Programa del ejemplo: Al subir el programa a Arduino, el LED deberá parpadear encendiéndose y apagándose cada segundo.

LED

```
void setup() {

    // Configuramos pin 8 como salida digital
    pinMode(8, OUTPUT);

}

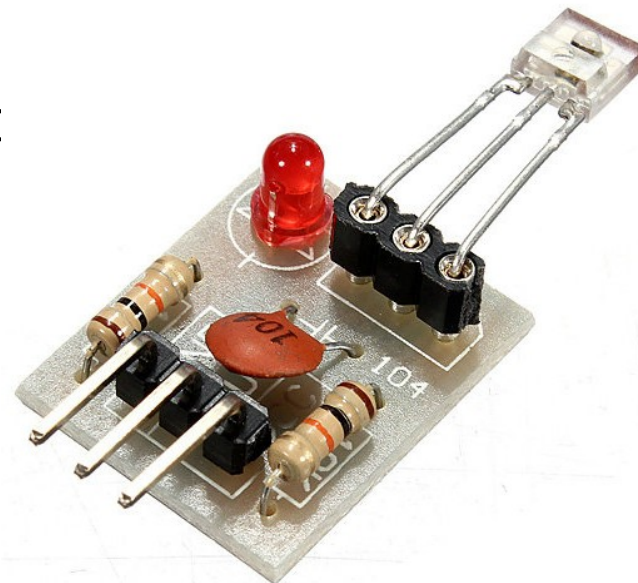
void loop() {

    // Enviamos corriente al pin 8 y esperamos 1seg.
    digitalWrite(8, HIGH);
    delay(1000);

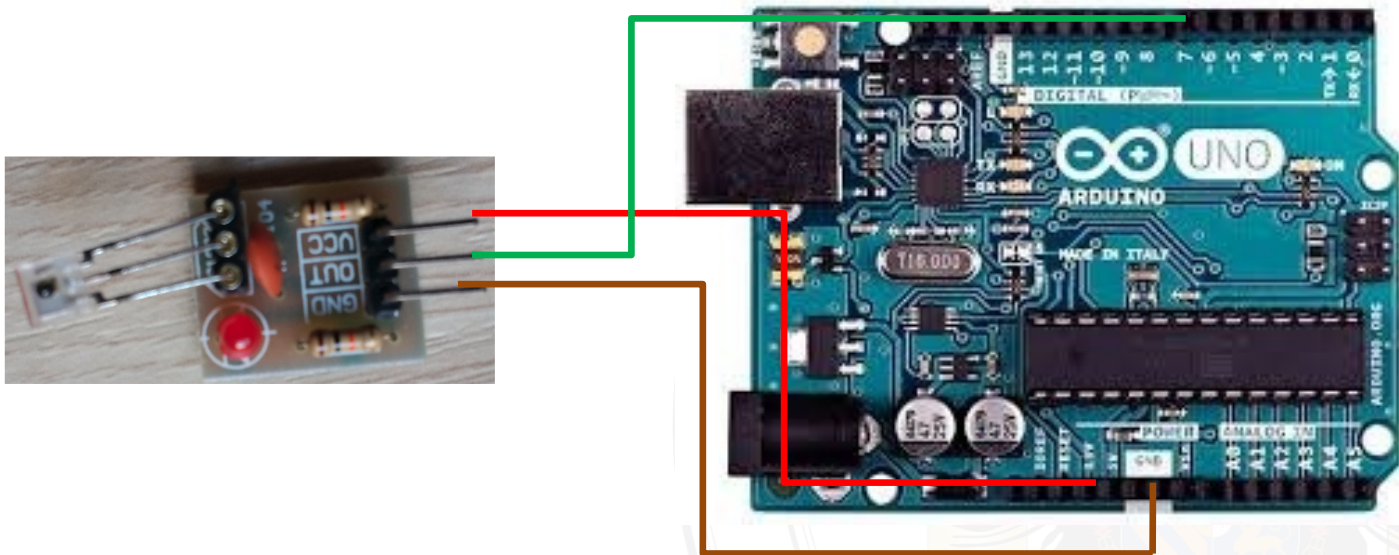
    // Desactivamos corriente en el pin 8 y esperamos 1seg.
    digitalWrite(8, LOW);
    delay(1000);

}
```

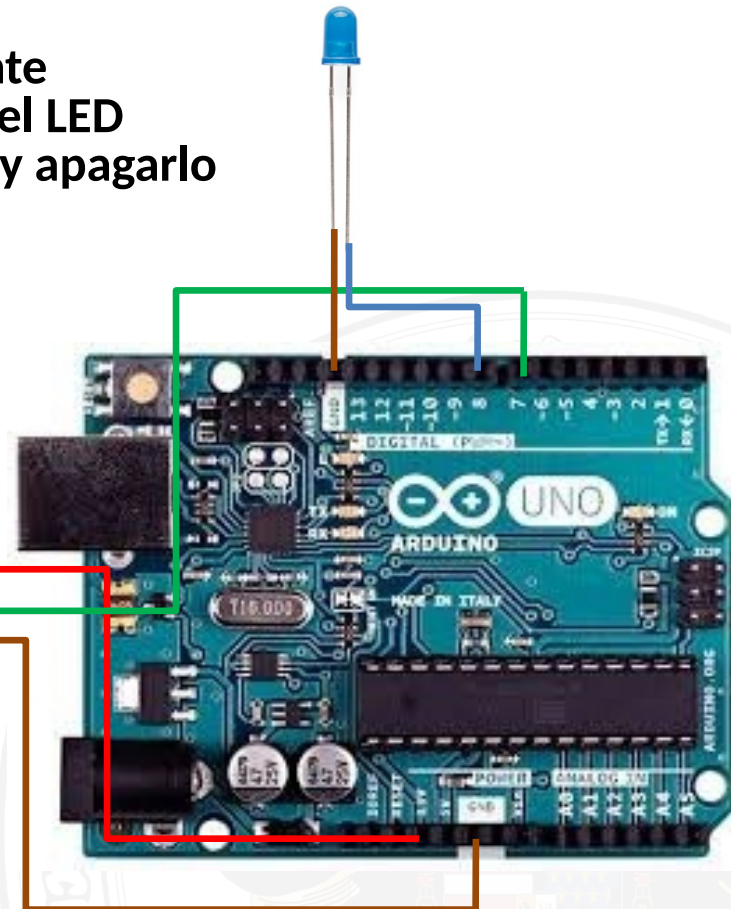
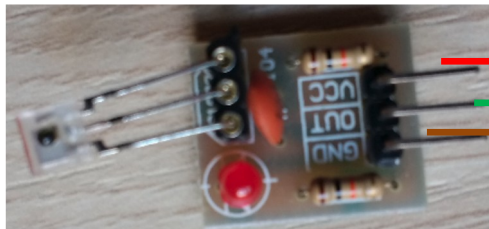
- Para construir los prototipos de emisión/recepción de información mediante láser, utilizaremos **fotorreceptores** capaces de captar la luz emitida mediante láser.
- El **Módulo receptor 2PCS** láser no modulador sensor Tubo es un fotorresistor que responde a la luz ambiental existente.
- Es un sensor que se conecta a un pin GPIO.
- Funciona a 3.3V.
- Tiene 3 patillas:
 - Conexión a 3.3V
 - Conexión a GND
 - Conexión a un pin digital (



- **Módulo receptor 2PCS** láser no modulador sensor Tubo. **Conexión con la placa:**
 - Utilizando cables de conexión de pines Macho-Hembra, conectaremos:
 - la patilla GND del sensor a tierra en la placa Arduino,
 - la patilla VCC a la alimentación de 3.3V de la placa, y
 - la salida del sensor VOUT al **pin 7** de E/S digital:



- También conectaremos un **LED al pin 8** de E/S digital Arduino
- Implementaremos el siguiente comportamiento: Encender el LED cuando el sensor reciba luz, y apagarlo en caso contrario.



– Prueba de funcionamiento (Programa *Fotorreceptor*):

Escriba un programa que:

1. Defina el pin 8 como salida y el pin 7 como entrada (función *setup*)
2. Inicialmente, el pin de salida debe estar a LOW (función *setup*)
3. Declave una variable global *encendido*, que inicialmente tenga el valor *false* (función *setup*).
4. El código de la función *loop()* deberá hacer lo siguiente:
 1. Si *digitalRead* lee HIGH desde el pin 7 y *encendido=false*, hacer:
 - *encendido=true*, *digitalWrite* HIGH al pin 8
 2. En otro caso, si *digitalRead* lee LOW desde el pin 7 y *encendido=true*, hacer:
 - *encendido= false*, *digitalWrite* LOW al pin 8

Prueba de funcionamiento (Programa *Fotorreceptor*):

Fotorreceptor §

```
// Para saber el el LED está encendido
bool encendido;

void setup() {

    // Pin de salida: LED
    pinMode(8, OUTPUT);

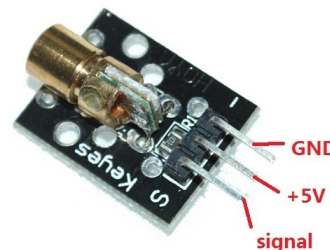
    // Pin de entrada: Fotorreceptor
    pinMode(7, INPUT);

    // Inicialmente no encender el LED
    encendido= false;
    digitalWrite(8, LOW);
}

void loop() {

    if (digitalRead(7) == HIGH) { // Lectura de luz en el fotorreceptor
        if (!encendido) { // Encendemos el LED si no está encendido
            encendido= true; digitalWrite(8, HIGH);
        }
    } else { // No se lee luz en el fotorreceptor
        if (encendido) { // Si está encendido el LED, se apaga
            encendido= false; digitalWrite(8, LOW);
        }
    }
}
```


- Para construir los prototipos de emisión/recepción de información mediante láser, utilizaremos **emisores láser** capaces de emitir luz en el rango visible con una longitud de onda de 650nm.
- El **Módulo emisor láser KY-008** es un emisor adecuado a nuestras necesidades.
- Es un emisor que se conecta a un GPIO de un puerto digital.
- Funciona a 5V.
- Su funcionamiento a nivel de programa equivale a encender/apagar un LED, como hemos visto en ejemplos previos.



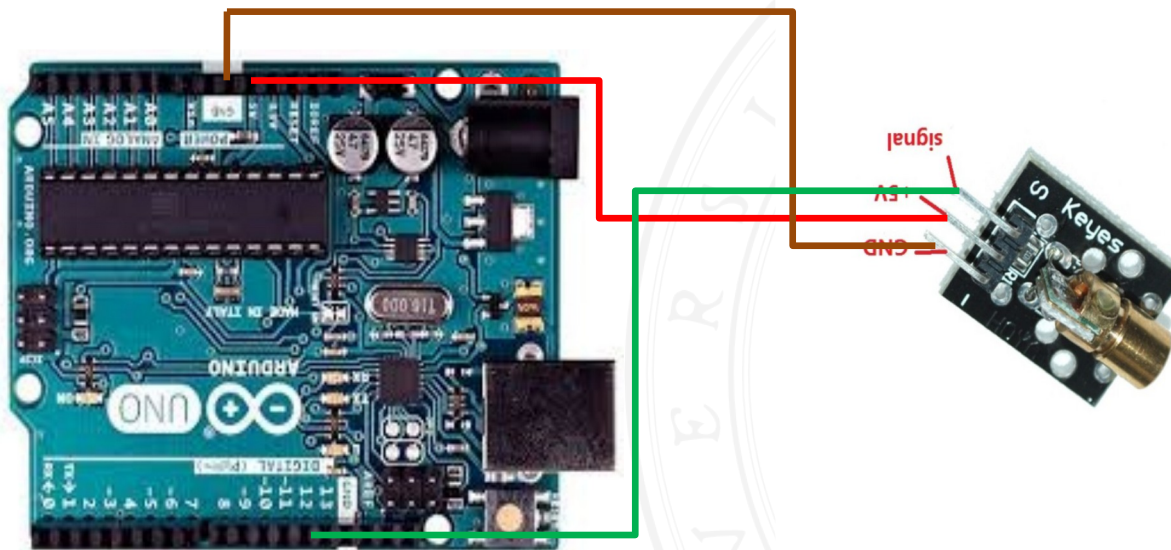
- **IMPORTANTE:** El emisor láser que se utiliza en las prácticas tiene una longitud de onda relativamente alta (650nm), por lo que no produce daños erosivos ni quemaduras. Sin embargo, **el láser no se debe orientar directamente a los ojos, pues estos sí pueden ser dañados** ante un estímulo de estas características.



– Ejemplo de funcionamiento:

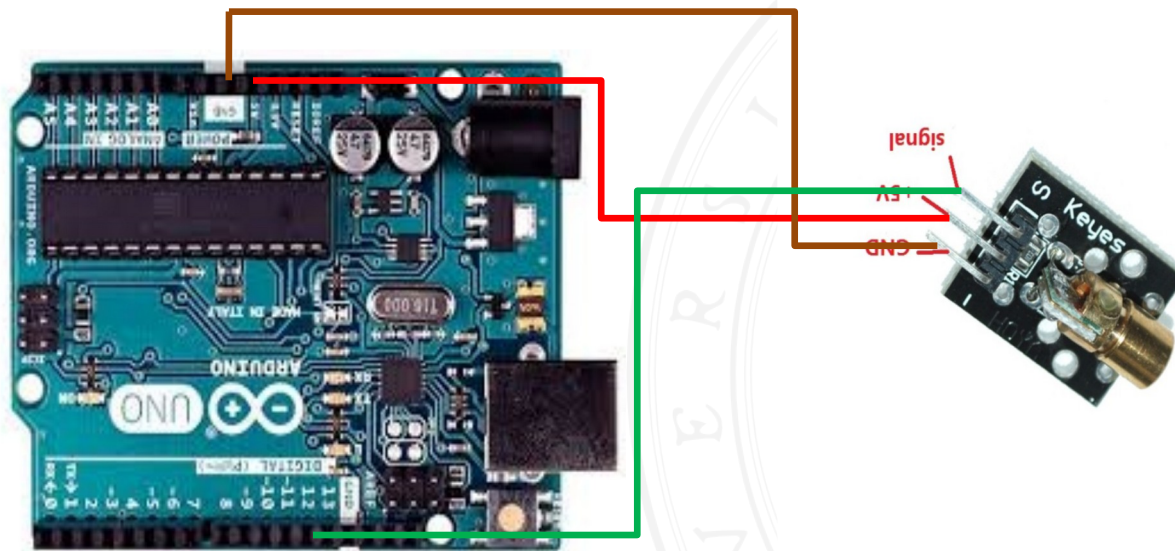
Utilizando el programa **Fotorreceptor** anterior, cambie el LED y sustitúyalo por el láser. Realice las siguientes conexiones (**¡CUIDADO: Desconecte Arduino antes de realizar cualquier modificación hardware!**):

- El pin S (Signal) del láser al pin 8 de Arduino
- El pin - (GND) del láser a cualquier pin GND de Arduino.
- El pin restante del láser al pin de 5V de Arduino



– Ejemplo de funcionamiento:

Resultado: El láser produce el mismo comportamiento que el LED por el que ha sido sustituido.





Universidad de Granada

decsai.ugr.es

Teoría de la Información y la Codificación

Grado en Ingeniería Informática

Seminario - Introducción a Arduino.



DECSAI

**Departamento de Ciencias de la
Computación e Inteligencia Artificial**