

# Introducción a la Robótica Móvil

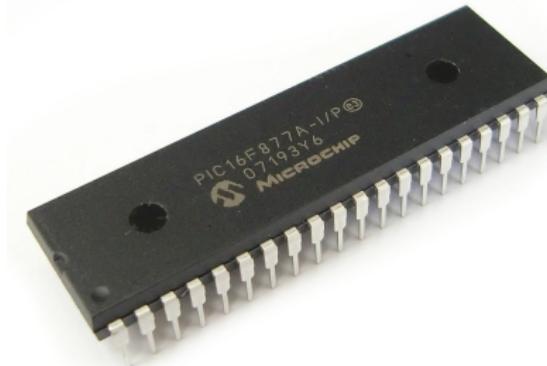
Primer cuatrimestre de 2018

Departamento de Computación - FCEyN - UBA

Miccontroladores - clase 2

Introducción a Arduino

# Microcontroladores



- Circuito integrado programable.
- Arquitectura Hardvard (por lo general).
- Memoria de datos (SRAM) y memoria de programa (EEPROM, Flash).
- Pines de E/S .
- Módulos para control de periféricos y para comunicación.
- Varios modelos y marcas: PIC (Microchip), MSP430 (Texas Instruments), AVR (Atmel), Cortex-M (ARM).

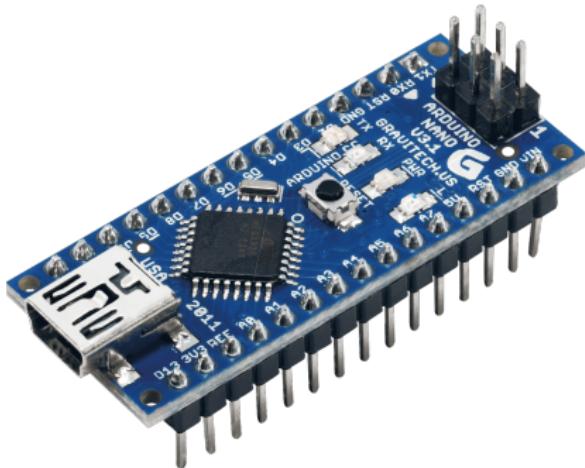
# Microcontrolador vs Microprocesador

## Microcontrolador

- Resuelve bien tareas específicas, e.g. lavarropas, microondas, etc.
- Típicamente RISC de 8 o 16 bits con arquitectura Harvard.
- Los datos se almacenan en SRAM, el programa en EEPROM o Flash.
- Un programa que se repite indefinidamente.
- No tienen un Sistema Operativo (o a lo sumo tienen un RTOS)
- Memoria y E/S on board.
- Bajo costo, menor tamaño y poco consumo.

## Microprocessor

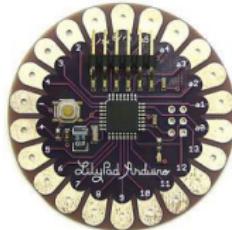
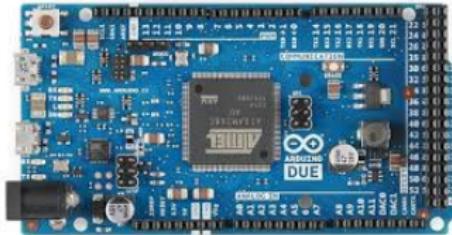
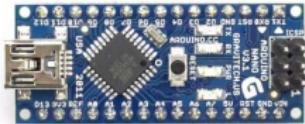
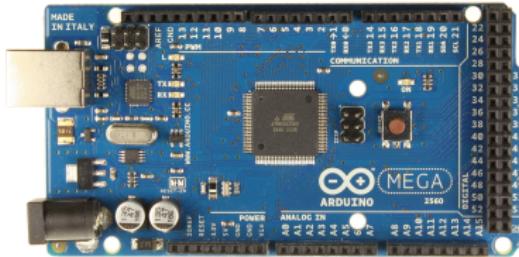
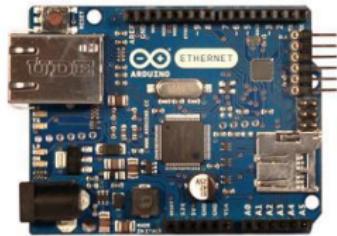
- Concebidos para propósito general.
- Típicamente CISC de 32 o 64 bits con arquitectura Von Neumann.
- Los datos y los programas se almacenan en DRAM (DDR2/3/4).
- Muchos procesos corriendo en paralelo.
- Tienen Sistemas Operativos muy complejos.
- Memoria y E/S externos, por ejemplo placas de video, red, etc.
- Más caro, de mayor tamaño y consumo.



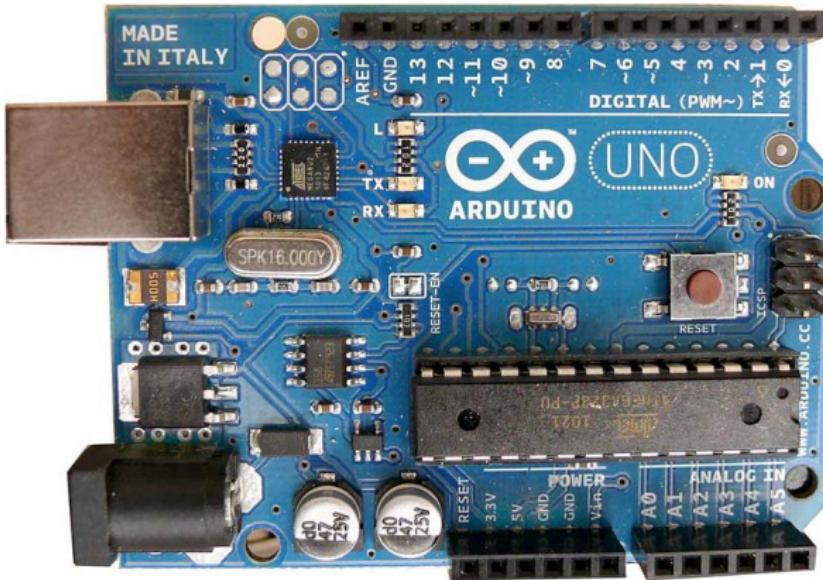
## ¿Qué es?

- Una empresa.
- Una placa (hardware).
- Una IDE (software).
- Open source.
- Programable por USB.
- Lenguaje de programación basado en Wiring/Processing
- Win-Mac-Linux.
- Barato.
- Muy popular.

# Muchos modelos de Arduinos



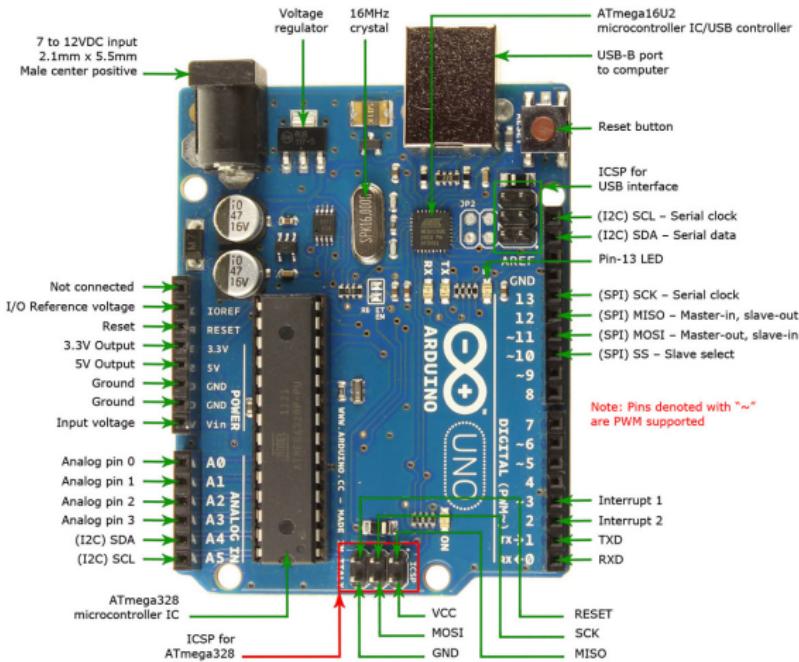
# Arduino UNO



## Especificaciones

- Basado en ATmega328.
  - RISC, 8 bits, 1 MIPS.
  - 32 KB Flash.
  - 2 KB SRAM.
  - 6 A/D 10 bits.
  - 14 E/S digitales.

# Esquema del Arduino UNO



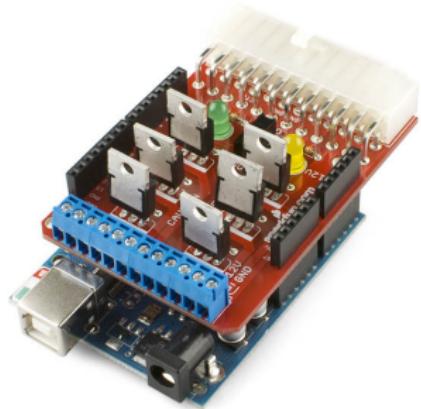
## ¿Qué podemos hacer?

- E/S digital: controlar leds, switches, PWMs.
- E/S analógica: leer sensores.
- E/S serial, I2C, SPI: e.g. GPS.
- Con shields: muchas otras cosas más.

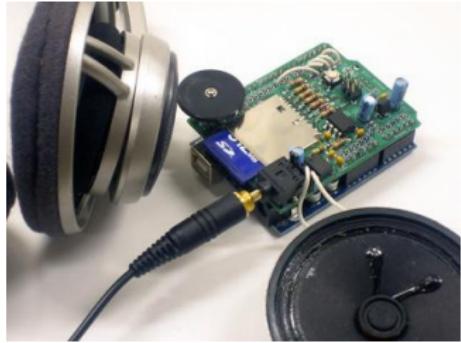
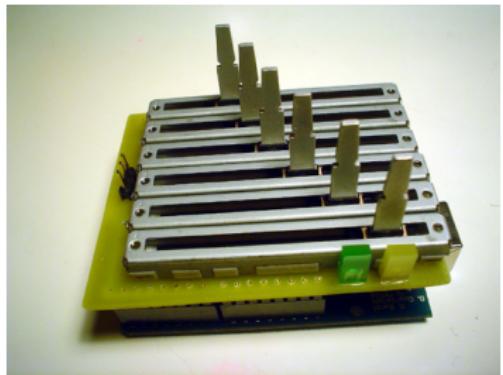
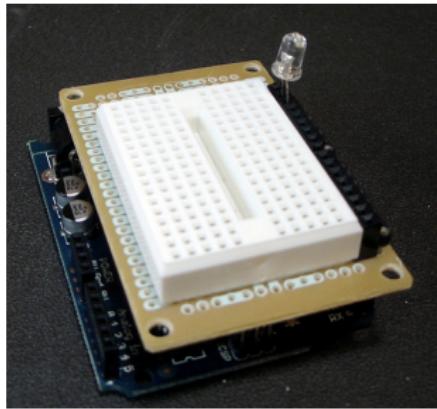
# Terminología de Arduino

- Micontrolador.
- Pines E/S.
- Shields
- Sensores y actuadores.
- Sketch
- Bibliotecas (mal dicho librerías) de funciones.

# Shields



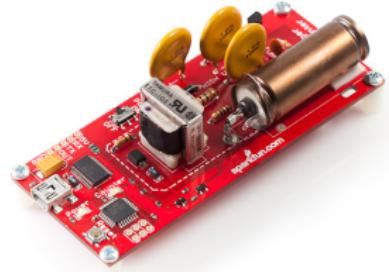
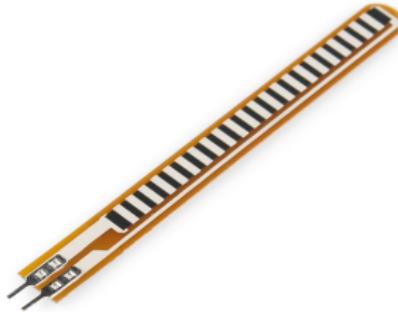
# Shields



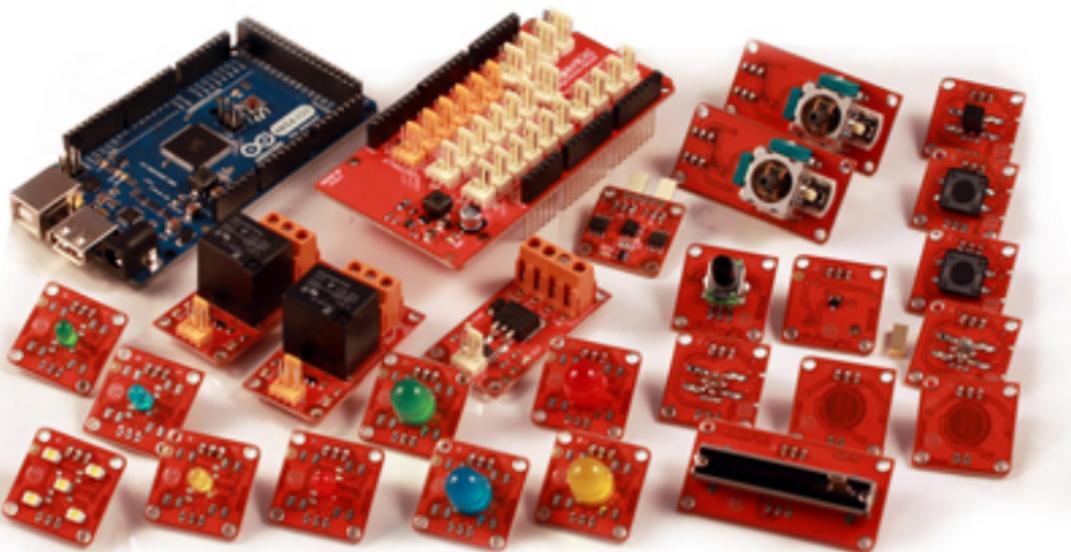
# Shields



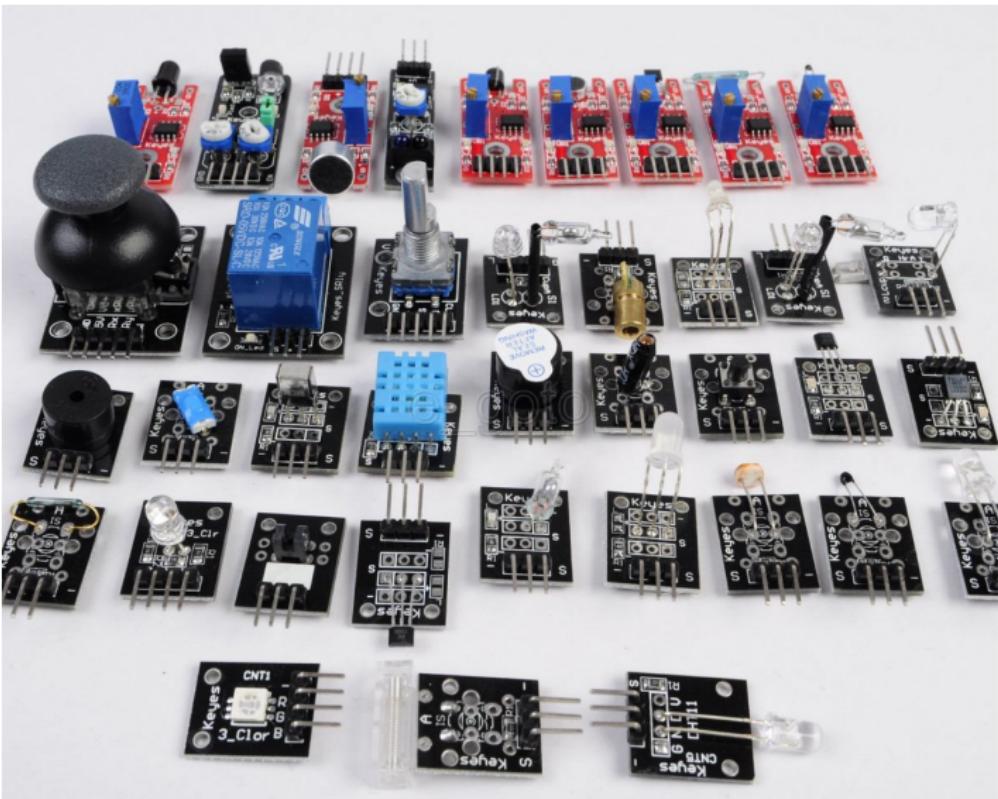
# Sensores



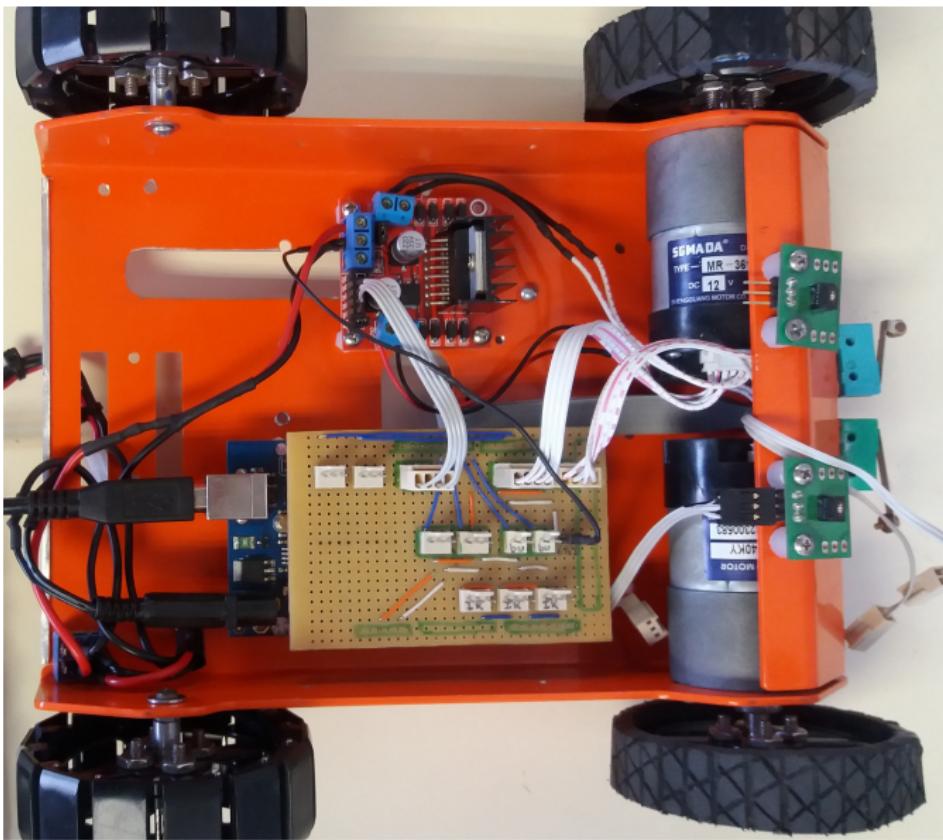
# Kits Arduinos



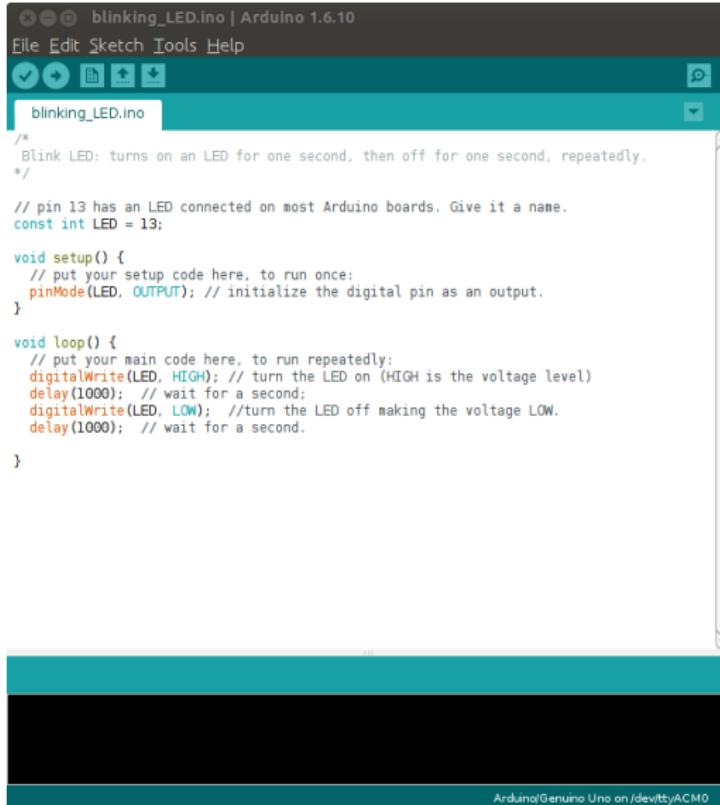
## Kits Arduinos



# Arduino en la vida real



# Sketch



The screenshot shows the Arduino IDE interface with the following details:

- Title Bar:** Shows "blinking\_LED.ino | Arduino 1.6.10".
- Menu Bar:** Includes File, Edit, Sketch, Tools, Help.
- Toolbar:** Contains icons for Save, Undo, Redo, Open, Upload, and Download.
- Code Editor:** Displays the C++ code for the "blinking\_LED.ino" sketch. The code is as follows:

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

// pin 13 has an LED connected on most Arduino boards. Give it a name.
const int LED = 13;

void setup() {
  // put your setup code here, to run once:
  pinMode(LED, OUTPUT); // initialize the digital pin as an output.
}

void loop() {
  // put your main code here, to run repeatedly:
  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 making the voltage LOW.
  delay(1000); // wait for a second.
}
```

**Status Bar:** At the bottom right, it says "Arduino/Genuino Uno on /dev/ttyACM0".

# Simulador Tinkercad

<https://www.tinkercad.com>

The screenshot shows the Tinkercad interface. At the top, there's a blue header bar with the text "Ready to build a maker mindset at your school? Join our webinar to learn how." Below the header, the Tinkercad logo is on the left, followed by navigation links: "TINKERCAD FOR...", "GALLERY", "COMMUNITY", "LEARN", and "TEACH". On the right side of the header is a search bar and a user profile icon.

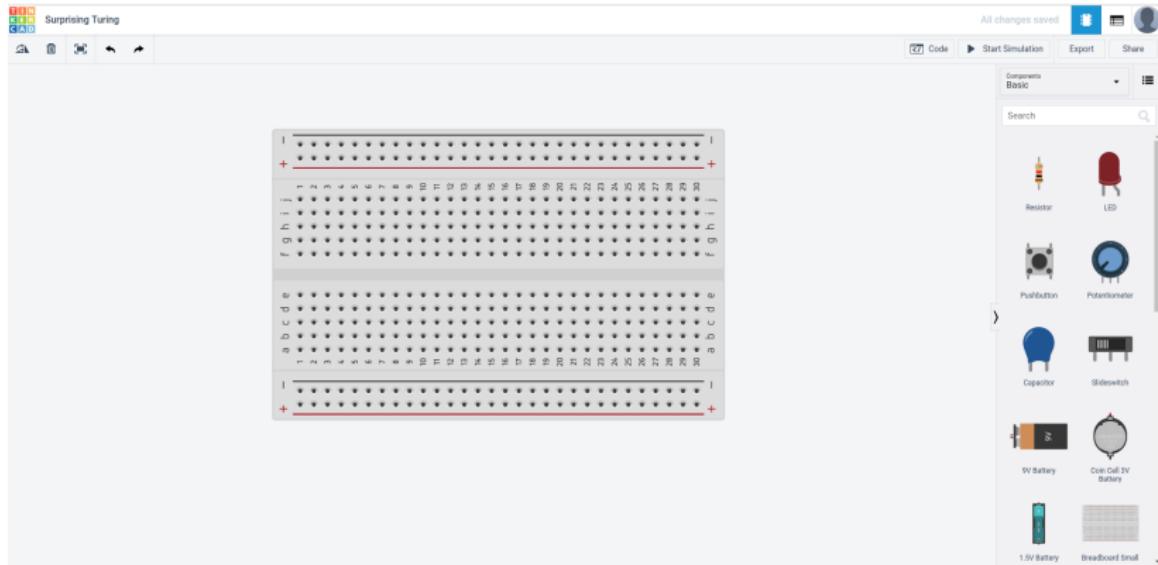
The main workspace is titled "Circuits". It features a "Create new Circuit" button in a green box. Below it is a preview area showing a breadboard with various components like resistors, capacitors, and a microcontroller. A call-to-action button "Try Circuits" is located below the preview.

On the left sidebar, there's a user profile picture for "pdecris" and a search bar labeled "Search designs...". Below that, there are tabs for "3D Designs", "Circuits" (which is highlighted in blue), "Lessons", "Projects", and a "Create project" button. Further down the sidebar are "Tweets" and a "Follow" button, along with a Twitter feed for the @tinkercad account. A message about student ownership of learning spaces is also present.

- Diseño 3D.
- Simulación de circuitos simples.
- Arduino (distintos modelos):
  - Programación! =D
  - Mucho más lento D=
- Comunidad: se pueden ver proyectos contribuídos por otros

**Importante:** Ya deberían haber creado una cuenta! (sino háganlo)

# La simulación: algunos componentes, el protoboard



- Facilita las conexiones de componentes, sin soldar o hacer una placa.
- Tiene filas y columnas internamente interconectadas.
- 2 pares de líneas de alimentación independientes (positivo y tierra).
- 2 grupos de columnas de propósito general.

# La simulación: Arduino, programación

The screenshot shows a Tinkercad simulation environment for an Arduino Uno R3. At the top, there's a toolbar with icons for file operations, undo, redo, and a search bar. The main area features a 3D model of an Arduino Uno R3 board with its pins labeled. A USB cable is connected to the board. To the right of the board is the code editor window, which contains the following Arduino sketch:

```
/*
  This program blinks pin 13 of the Arduino (the
  built-in LED)
*/

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

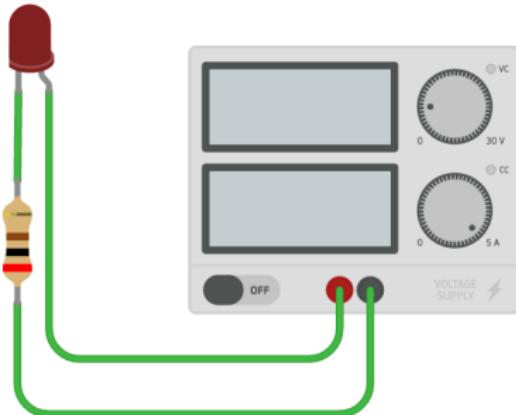
// The setup routine runs once when you press reset:
void setup()
{
  // Initialize the digital pin as an output:
  pinMode(led, OUTPUT);
}

// The loop routine runs over and over again forever:
void loop()
{
  // turn the LED on (HIGH is the voltage level)
  digitalWrite(led, HIGH);
  delay(1000); // Wait for 1000 milliseconds = 1 second
  // turn the LED off (LOW is the voltage level)
  digitalWrite(led, LOW);
  delay(1000); // Wait for 1000 milliseconds = 1 second
}
```

Below the code editor is the serial monitor window, which is currently empty.

(click para simulación)

# Hola mundo! (version electrónica)

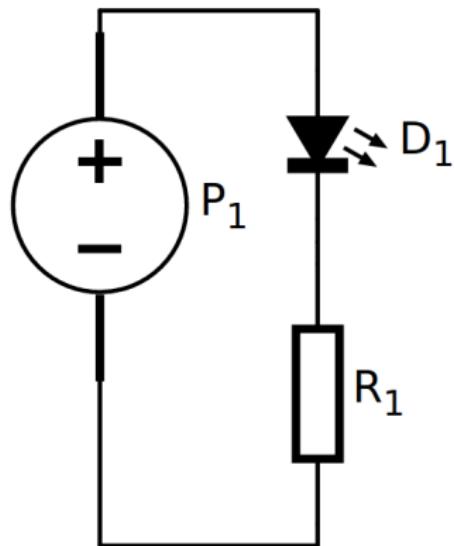


(click para simulación)

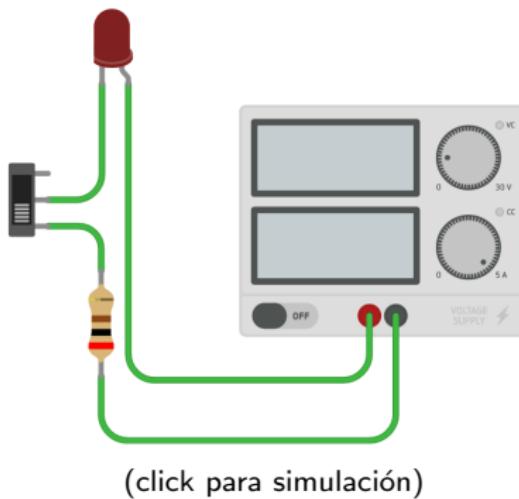
- Fuente: puedo elegir el voltaje que saca y limitar la corriente que ofrece
- LED: diodo emisor de luz. Podemos suponer que no ofrece resistencia (se “come” toda la corriente que puede).
- Resistencia: evita que el LED se queme (limita la corriente), lo elijo según el voltaje de la fuente (generalmente 5V)

# Hola mundo! (esquemático)

- Diagrama formal para describir circuitos.
- Lectura directa y sencilla.
- Abstacta de ciertos detalles (ej: color del LED).



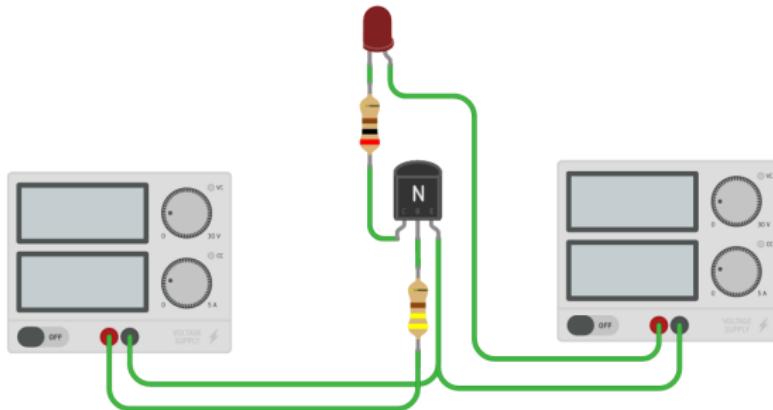
# Tomemos el control (que el último apague la luz...)



- Switch: según la posición de la llave, interconecta dos pines para cerrar el circuito.
- Control del LED: en una posición el circuito está abierto (no circula corriente), en la otra sí.

# Un interruptor con lógica digital

Quiero controlar el LED con una señal *lógica* (un voltaje que sea 5V o 0V), no un interruptor mecánico (somos muy perezosos)

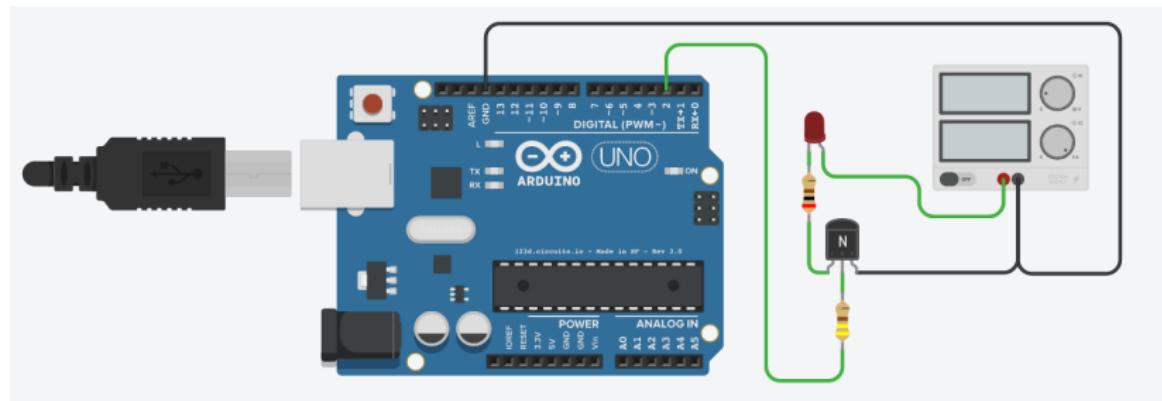


- Transistor: tiene tres pines: según el voltaje (en realidad corriente, por eso la resistencia) que vea en el pin del medio, deja pasar la corriente.
- Segunda fuente: la usamos para simular la señal digital, la corriente que ofrece es despreciable.

# Un interruptor todavía más complicado

Cómo prender una lamparita, según un programador...

Usamos una señal de un microcontrolador para controlar el LED (o cualquier otra cosa!). Ahora el LED se prende solo!



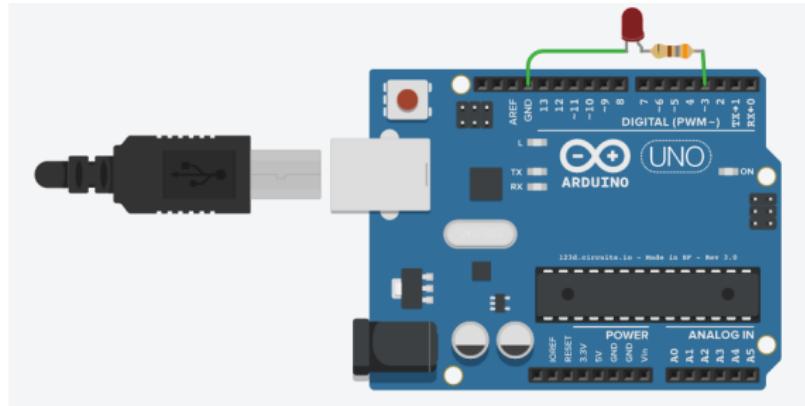
(click para simulación)

→ Con Arduino podemos controlar el valor lógico del pin, que a su vez controla el circuito del LED + fuente.

**Importante:** notar que las masas (0V o *ground*) siempre van juntas. Los voltajes son relativos a una referencia que debe ser común!

# Simplifiquemos un poco

El pin está controlado por un transistor, capaz de otorgar una corriente pequeña (10-40mA) que para el LED alcanza. Nos evitamos la fuente y el transistor.



(click para simulación)

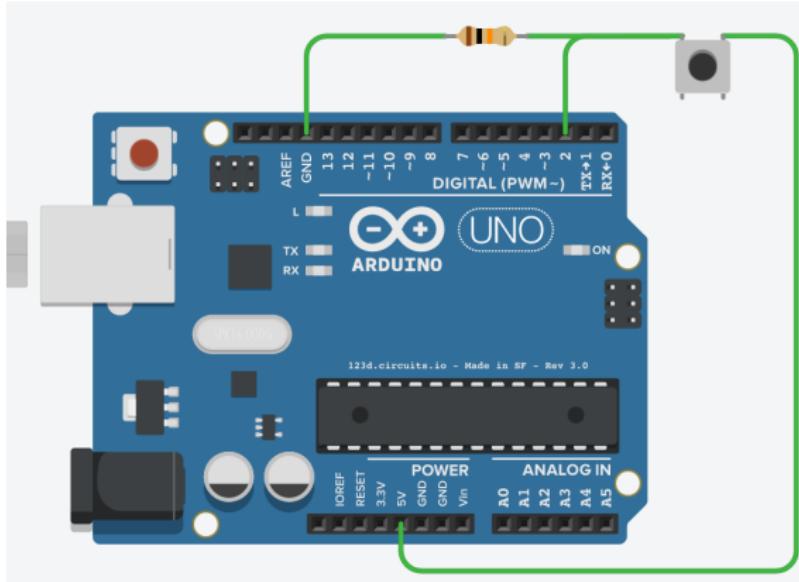
# ¿Y el código?

El Arduino está programado para parpadear el LED apenas lo enchufamos y el programa empieza a correr (sketch)

```
1 // el numero de PIN al que conectamos el LED
2 int led = 3;
3
4 // esta función corre una vez al inicio
5 void setup() {
6     pinMode(led, OUTPUT); /* configurar al pin como salida (vamos a sacar una señal por acá) */
7 }
8
9 // esta función corre como un ciclo infinito
10 void loop() {
11     digitalWrite(led, HIGH);    // escribo una señal "HIGH" (5 volts)
12     delay(1000);              // espero un segundo
13     digitalWrite(led, LOW);    // escribo una señal "LOW" (0 volts)
14     delay(1000);              // espero un segundo
15 }
```

# ¿Cómo hago para leer una señal?

Un pin puede configurarse como “input”, para leer si hay 0 o 5 volts.

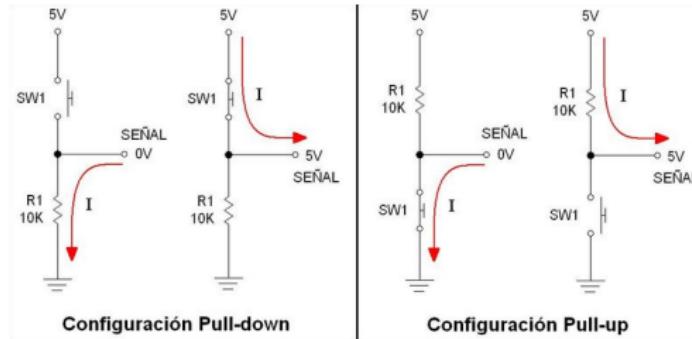


(click para simulación, ver código)

**Pregunta:** ¿por qué hay una resistencia?

# Cómo evitar que el voltaje “flete”

- Cuando una entrada no está conectada a nada (botón sin apretar) estaría leyendo “ruido”.
- Quiero que las entradas tengan valores siempre definidos: cableo la señal a tierra/positivo.



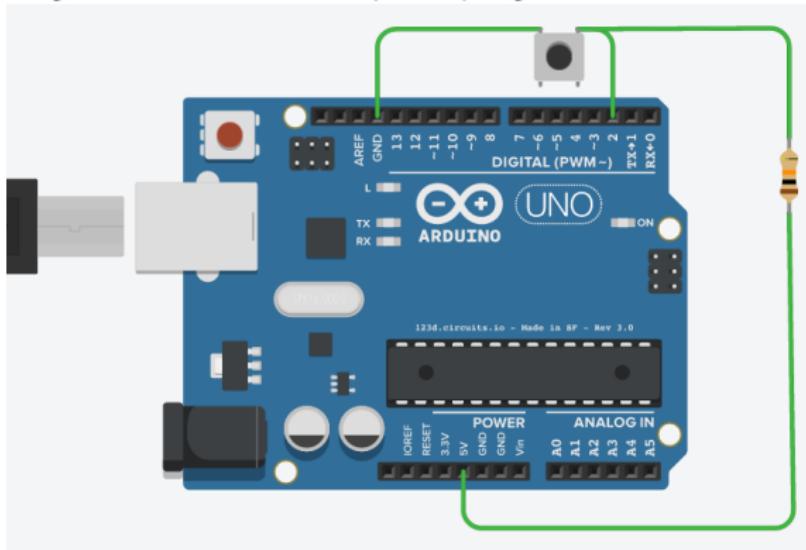
Y, entonces, ¿por qué la resistencia?

**Respuesta corta:** estaría poniendo en “corto” el circuito (evito pedir corriente excesiva a la fuente y “quemarla”)

**Respuesta larga:** quiero predeterminar la señal en un valor conocido, pero que otra señal pueda tomar precedencia (lo que quiero leer)

# Lectura digital, ¿y con un pull-up?

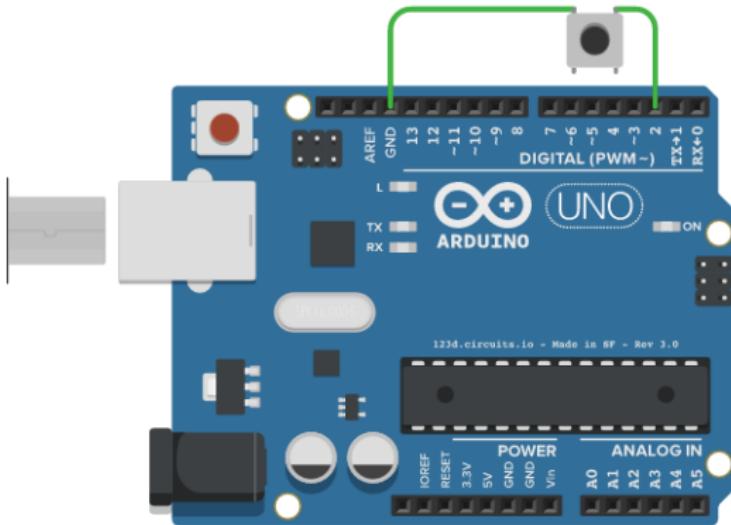
¿Cómo sería con un pull-up? ¿Qué cambiaría?



(click para simulación, ver código)

# Lectura digital, ¿y más simple todavía?

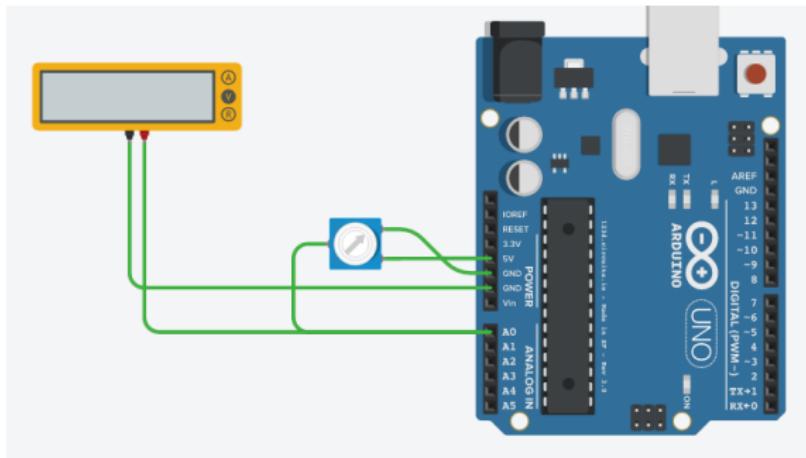
El Arduino en general tiene *pull-ups* internos.



(click para simulación, ver código)

# No todo es digital: conversión A/D

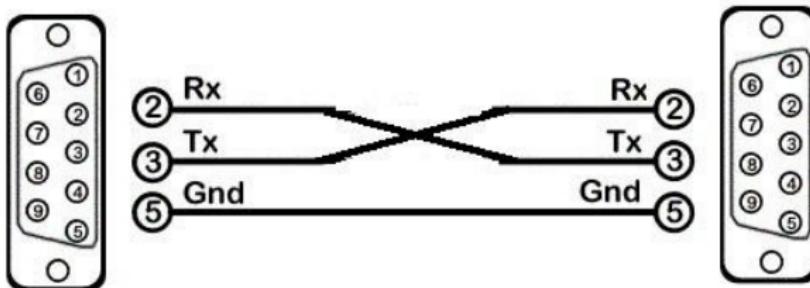
Existen pines especiales (A0-A5) que permiten leer voltajes continuos (no digitales) entre 5V y GND. El valor en volts se convierte a un número binario.



(click para simulación, ver código)

- Potenciómetro: resistencia *variable*, permite obtener un voltaje variable (no nos preocupemos por ahora cómo)

# Comunicación serial: el afamado RS232



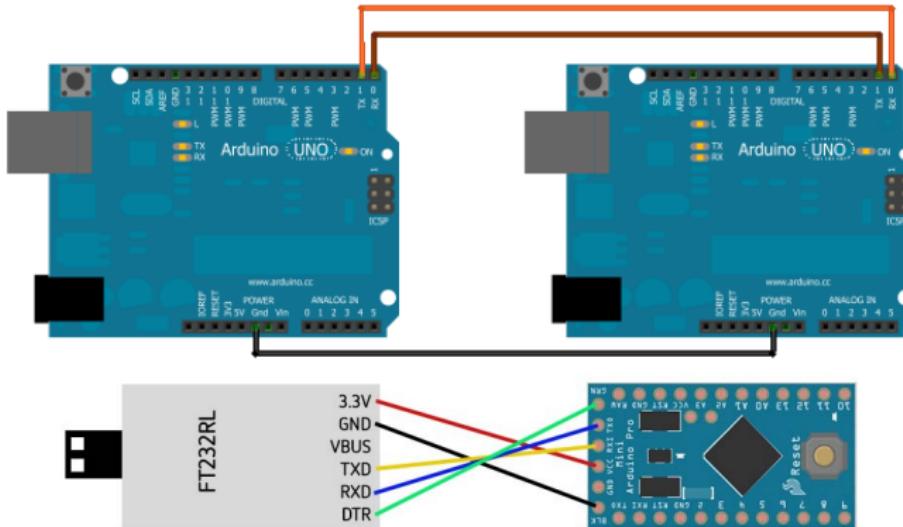
**DB9 hembra**  
visto desde atrás

**DB9 hembra**  
visto desde atrás

- RS232: Protocolo de comunicación serial (es decir, mando los bits de a uno por un cable), relativamente simple.
- Hardware: como mínimo RX, TX y GND (luego hay otras señales, pero no son estrictamente necesarias)

# Comunicación serial en Arduino

- Hardware serial: pines TX, RX que manejan las señales 0-5V
- Serial emulado: dispositivo USB que le hace creer a la PC que tiene un puerto serie (ejemplo: adaptador serial-usb)
- Conector USB arduino!



# Comunicación serial en Arduino

- Setup: inicializar el puerto serie, indicando la velocidad de comunicación (baudios / bits por segundo)
- Uso: imprimo usando Serial.print/println (tambien puedo leer)

The screenshot shows the Arduino IDE interface. At the top, there are tabs for "Upload & Run", "Libraries", "Download Code", "Debugger", and "Serial Monitor". The "Serial Monitor" tab is highlighted in blue. Below the tabs, the board selection dropdown shows "1 (Arduino Uno R3)". The code editor contains the following sketch:

```
1 void setup()
2 {
3     Serial.begin(9600);
4 }
5
6 void loop() {
7     Serial.println("Hola mundo!");
8 }
```

The "Serial Monitor" window displays the output of the code, which is the string "Hola mundo!" repeated 14 times.

mundo!
Hola mundo!
Hola mundo!
Hola mundo!
Hola mundo!
Hola mundo!
Hola mundo!
Hola mundo!
Hola mundo!
Hola mundo!
Hola mundo!
Hola mundo!
Hola mundo!
Hola mundo!

# Arduino: muy breve resumen de funciones

`analogRead(pin)` devuelve un numero entero de 10 bits (0-1023),  
proporcional al voltaje del pin **analógico**  
**nota:**  $A_n \rightarrow \text{pin} = n$

`pinMode(pin, modo)` selecciona un pin **digital** como INPUT,  
INPUT\_PULLUP o OUTPUT

`digitalWrite(pin, valor)` escribe un valor HIGH o LOW en el pin  
**digital** (poner como OUTPUT!)

`digitalRead(pin)` devuelve un valor HIGH o LOW segun lo que se sense  
en pin **digital** (poner como INPUT!)

`delay(ms)` pausa la ejecución durante el tiempo indicado en  
milisegundos

`Serial.begin(velocidad)` configura el puerto serial (usar 9600 en el  
simulador). Hacer en el setup!

`Serial.print / Serial.println` imprime (sin dejar y dejando un  
salto de línea) un número, string, etc.

# Taller 1: basta de tanta cháchara, quiero jugaaaar!

Bajar consigna desde la página de la materia!