

2. Montajes básicos con Arduino

- Montajes básicos con Arduino
- Conexiones digitales
- Conexiones analógicas
- Señales PWM
- Mapeo

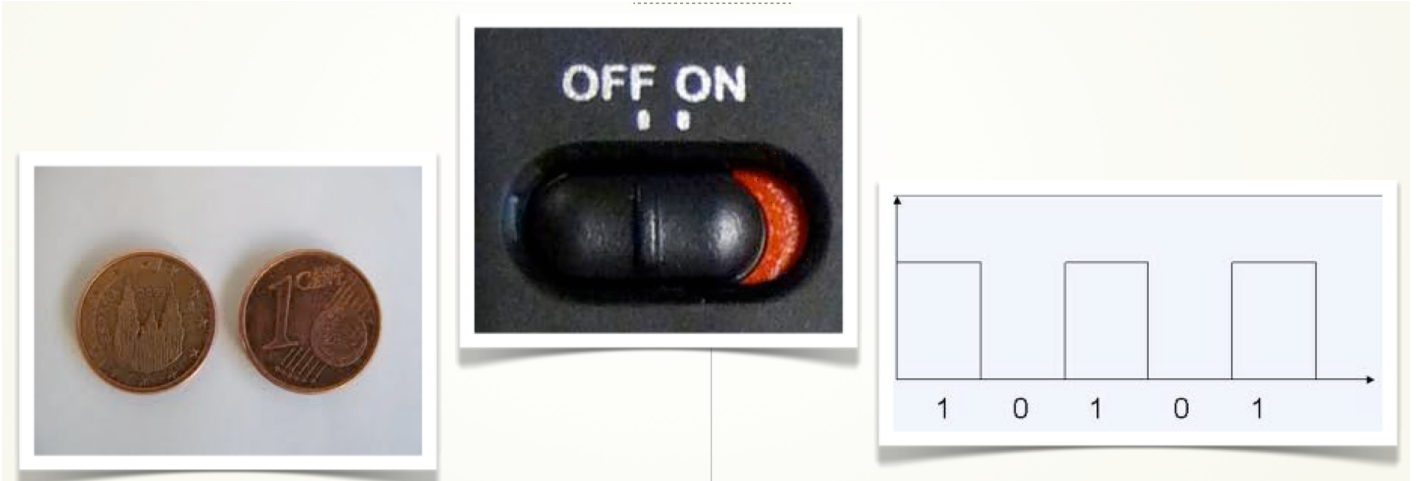
Montajes básicos con Arduino

Antes de empezar a realizar pequeños proyectos, vamos a dar un repaso a la plataforma Arduino, con las prácticas básicas que necesitarás para comprender los conceptos y avanzar en el libro .

<https://giphy.com/embed/Vo5Zvl0TeKL4Y>

[via GIPHY](#)

Conexiones digitales



En este apartado aprenderemos el funcionamiento básico de las entradas y salidas digitales de la placa Arduino. Si observamos bien la placa, vemos que hay 13 pines digitales.

En este caso la señal digital no es más que un valor discreto de entre dos posibles, que si en rigor se asocian a tensiones, nosotros por sencillez los asociaremos a dos valores que serán ,apagado o encendido o lo que es lo mismo LOW ó HIGH.

Así si asignamos un 0 al pin digital 4, es lo mismo que decir que ese pin, o mejor dicho, lo que esté conectado a ese pin estará apagado si le asignamos un 1, estamos diciendo que estará encendido.

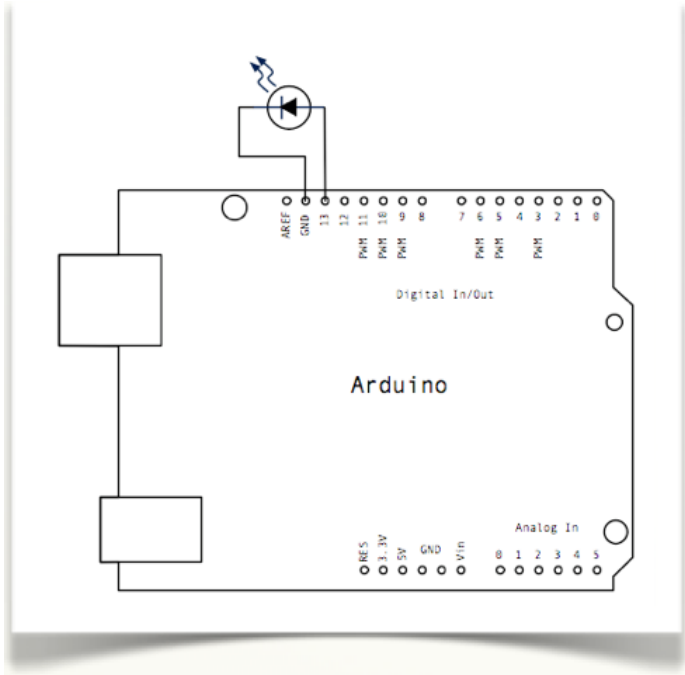
Entonces, ¿Con los 13 pines digitales de Arduino , podríamos actuar para controlar 13 bombillas? . Si, aunque Arduino es aún más potente ya que aún podemos usar los 5 pines analógicos también como salidas digitales. Veamos como.

Montaje 1: LED parpadeante sin EDUBASICA

Vamos a controlar el encendido y apagado de un led conectado al pin13 de Arduino.

¿Por qué el pin13 y no otro? Podríamos hacerlo con otro, pero el pin13 tiene asociado un led en la placa justo debajo de el y así nos evitamos tener que montar. Si pusiéramos un pin polarizado correctamente entre el pin13 y GND también funcionaría. El pin13 tiene también una resistencia que hace posible conectarle un led directamente, si hacemos el montaje con otro pin debemos añadir esta resistencia de 10Kohm entre el led y el pin.

Acuérdate: La pata más larga del LED es el (+) por lo tanto en el D13 y el corto (-) en GND.



<https://create.arduino.cc/editor/javierquintana/27aca596-db6d-4a41-a100-faf60cefa56e/preview>

<https://create.arduino.cc/editor/javierquintana/27aca596-db6d-4a41-a100-faf60cefa56e/preview?embed>

Todo lo que está entre las llaves de **loop()**, se ejecuta indefinidamente. Así vemos un efecto de led parpadeante ya que si analizamos las líneas del código vemos que el proceso es:

- Encendemos.
- Esperamos un segundo.
- Apagamos.
- Esperamos un segundo.

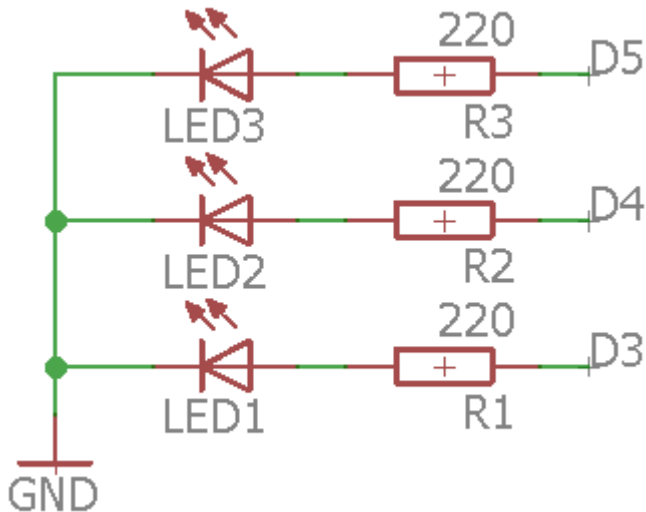
¡Atrevámonos y cambiemos los tiempos de parada!

<https://www.youtube.com/embed/EFfSLvIF9rY>

Montaje 2: LED EDUBASICA parpadeante

Igual que en el caso anterior, pero vamos a utilizar un LED de la shield de Edubásica, tenemos tres opciones:

- LED VERDE pin 3
- LED AMARILLO pin 4
- LED ROJO pin 5



El programa es igual que el anterior, pero cambiando el número del pin:

<https://create.arduino.cc/editor/javierquintana/2c75c843-72c9-4dc4-b01a-410e318f1080/preview>

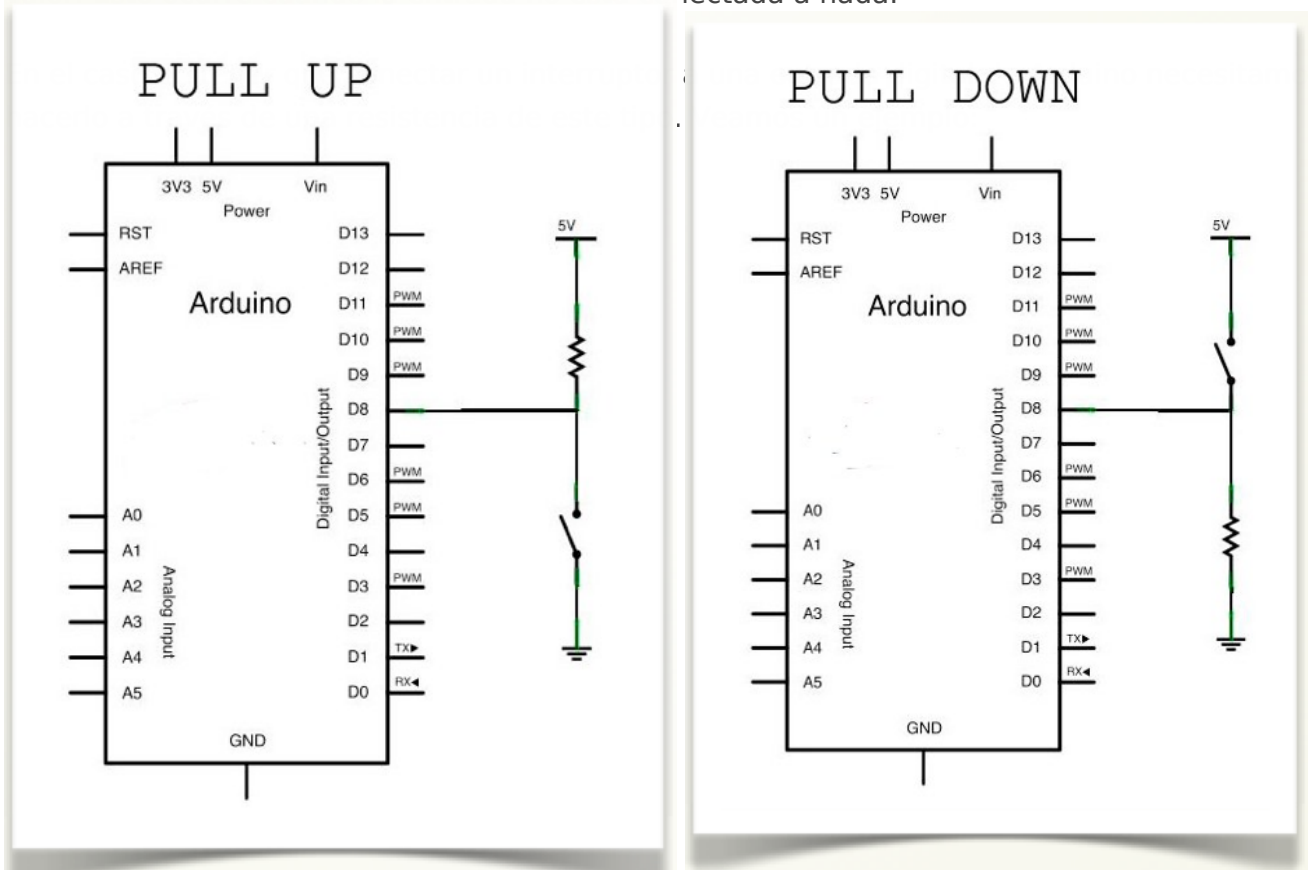
<https://create.arduino.cc/editor/javierquintana/2c75c843-72c9-4dc4-b01a-410e318f1080/preview?embed>

<https://www.youtube.com/embed/UHttCda49Vo?rel=0>

Resistencias pull-up y pull-down

En los proyectos con dispositivos digitales, caso de la placa Arduino, reciben señales de entradas digitales del exterior. Estas señales externas sirven para activar o desactivar un circuito, recibir información del estado de un sensor,...

Las resistencias “pull-up” y “pull-down” son resistencias que se ponen en las entradas digitales para fijar un valor por defecto, nivel alto (“1”) o nivel bajo (“0”), cuando no se detecta ningún valor. Esto ocurre cuando la entrada no está conectada a nada.

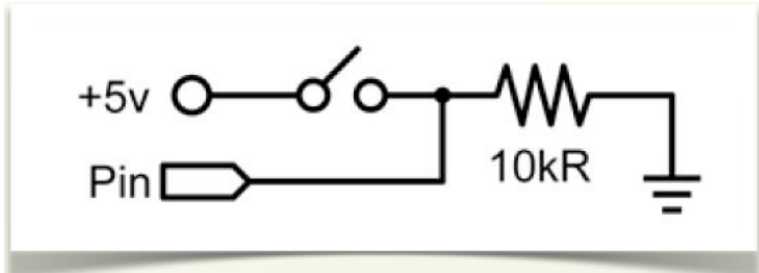


La resistencia “**pull-up**” establece un nivel alto (1) en la entrada digital en el estado de reposo del circuito. Un circuito con una entrada “pull-up” sería el siguiente.



La resistencia **“pull-down”** establece un nivel bajo (0) en la entrada digital en el estado de reposo del circuito. Este tipo de circuito es el más empleado en las entradas digitales para evitar lecturas erróneas debido a ruidos externos y consumo de energía. La resistencia suele ser de 10 kΩ y el circuito con esta configuración sería el siguiente.

Un ejemplo de circuito **“pull-down”** lo tenemos en la placa EduBásica en el pin digital D2, preparado para configurarlo como entrada, tiene conectado un pulsador y una resistencia pull-down. El esquema del circuito es el siguiente



El funcionamiento de este circuito que está conectado al pin digital D2 como entrada es detectar si el pulsador está pulsado o no.

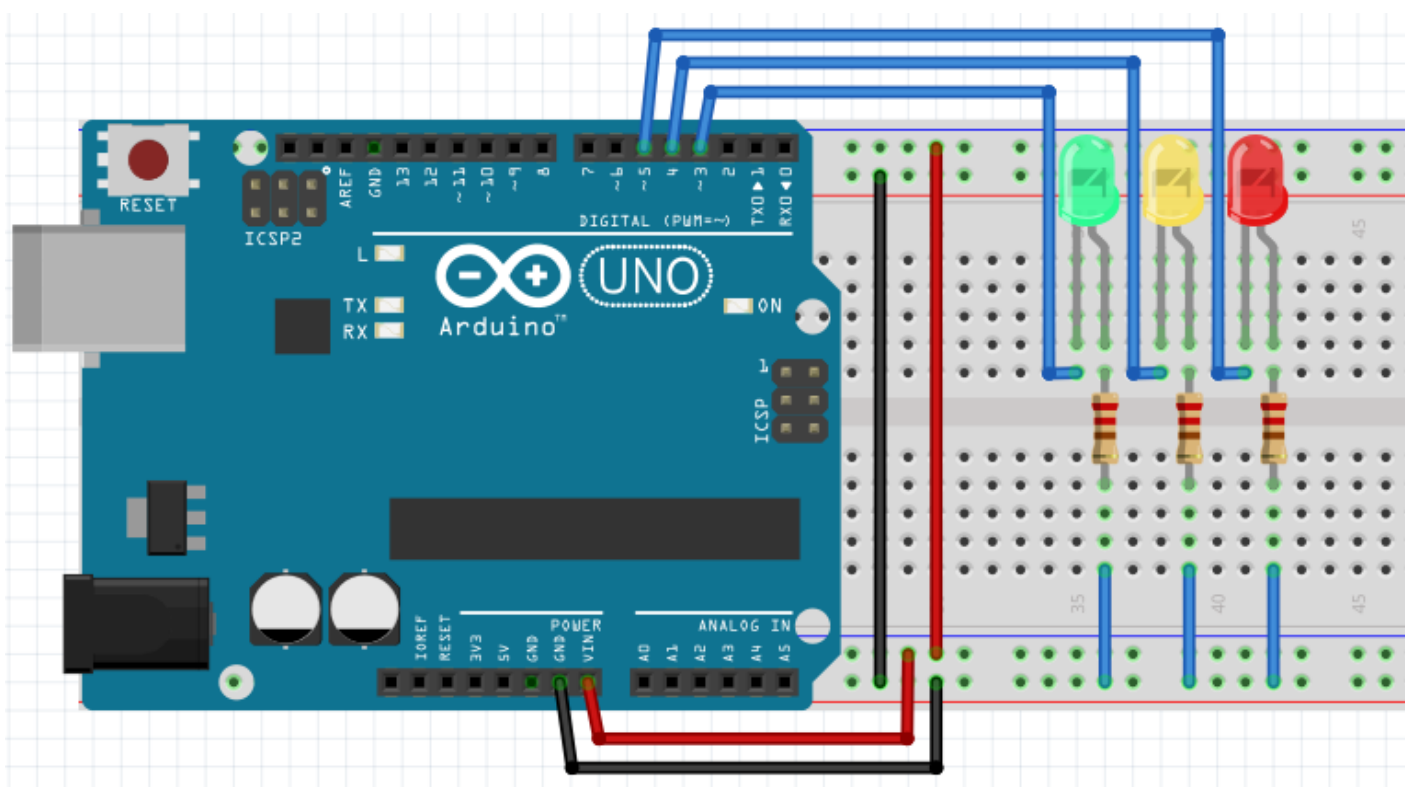
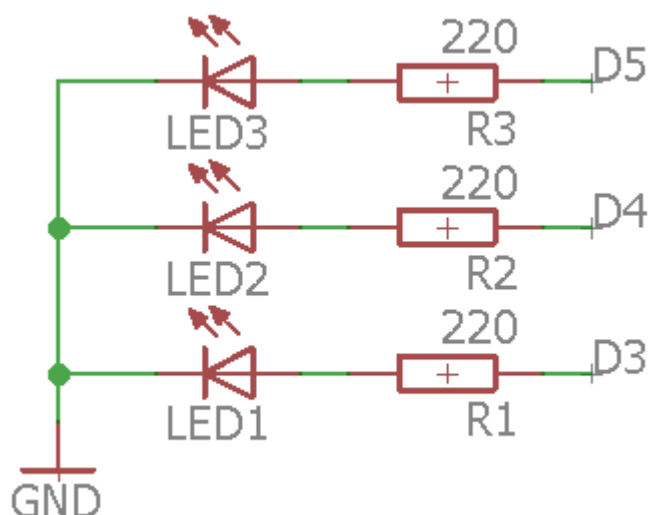
- Si el pulsador no está pulsado en Pin, que está conectado a D2, tenemos 0V por no pasar corriente entre el pin D2 y masa. Por tanto, corresponde a un nivel bajo o “0” lógico.
- Si el pulsador está pulsado en el pin D2 tenemos 5V, que corresponde a un nivel alto o “1” lógico.
- Si en el anterior circuito no se pone la resistencia de 10KΩ y el pulsador está abierto con el propósito de tener un nivel bajo porque no hay tensión, puede ocurrir y de manera aleatoria que el pin D2 lea un nivel alto por alguna interferencia producida por motores eléctricos, bobinas de un relé u otro dispositivo de nuestro proyecto.

Montaje 3: SEMAFORO CON EDUBASICA

Montaremos un semáforo con los tres leds de la EduBásica. Aquí es muy fácil, no hay que hacer ningún montaje.

Montaje 3: SEMÁFORO SIN EDUBASICA

La EduBásica es opcional y podemos montar el circuito correspondiente con una protoboard, pero EduBásica nos ahorra trabajo. Necesitamos añadir una resistencia entre el pin y el led, para evitar que el led se funda.



en este caso tienes libertad de utilizar D3 D4 D5 o otros que quieras.

Visualización de datos por el puerto serie. Serial.print

Queremos enseñarte un nuevo comando: **Serial.print**.

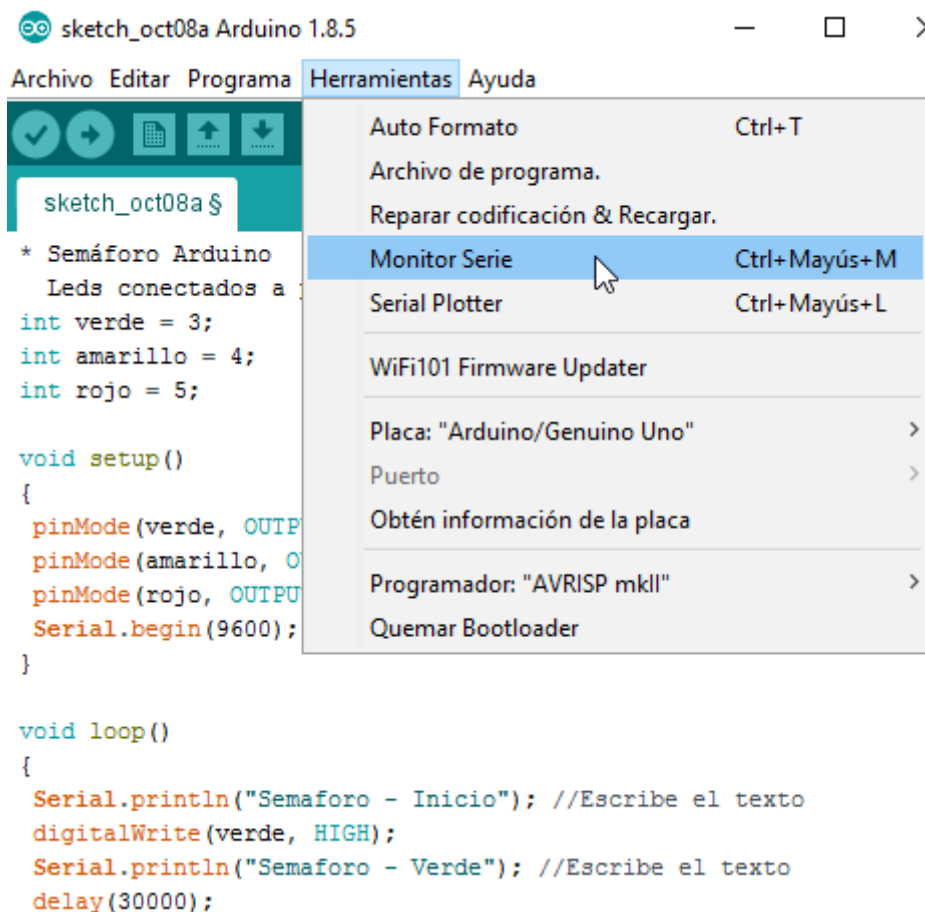
Este comando nos manda un texto al puerto serial por el que nos comunicamos con Arduino. De esta manera podemos depurar un programa sabiendo siempre por que línea está.



Para que funcione debemos tener en cuenta que:

- Hay que inicializar Serial. Esto se hace poniendo **Serial.begin(9600)** dentro de la rutina de setup(). 9600 se refiere a la velocidad que se comunicará.
- **Serial.print("xxx")** escribe lo que ponemos entre comillas tal cual.
- **Serial.print(x)** escribe el valor que contenga la variable x.
- **Serial.println()** es similar a lo anterior pero después añade un salto de línea.

Para ver lo que nuestro Arduino nos comunica por Serial, abrimos el monitor Serial que tenemos en el programa Arduino:





<https://create.arduino.cc/editor/javierquintana/28bf7eaa-441b-46fa-8a1c-7db91075f6ac/preview>

<https://create.arduino.cc/editor/javierquintana/28bf7eaa-441b-46fa-8a1c-7db91075f6ac/preview?embed>

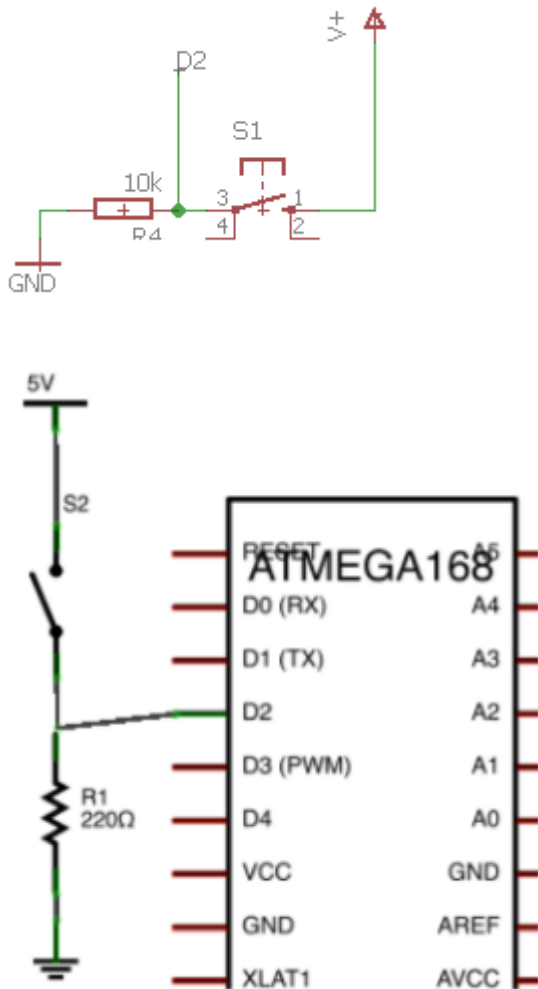
Montaje 4: Pulsador

Hasta ahora hemos visto como programar Arduino para que ejecute repetitivamente acciones, pero este actuaba de manera autónoma y nosotros sólo podíamos observar. Pero podemos interactuar con Arduino, por ejemplo, realizando una acción cuando activamos un pulsador. .

En este ejemplo, vamos a encender un led D3 verde cuando actuamos sobre el pulsador.

Montaje 4: Pulsador sin EDUBASICA

Utilizamos por ejemplo el pin 2 corresponde al pulsador y el pin 3 al led verde, solo nos queda cargar el programa y probar.



Montaje 4: Pulsador con EDUBASICA

Con la misma estructura (D2 es donde está el pulsador en EDUBASICA y D3 el LED VERDE) no hace falta realizar ningún cableado.

Programa 4. Comando digitalRead ...

Aparece un comando nuevo “**digitalRead(buttonPin)**” . Retorna el valor del pin que se ha configurado como entrada y al igual que en el caso de los pines que se configuran como salida, puede tener dos valores HIGH y LOW.

Si es HIGH significa que este pin está unido a la señal de 5v, si es LOW significa que está unido a 0v.

¿Por qué cuando el pulsador está en OFF D2 está a 0V? Porque está en PULL-DOWN

<https://create.arduino.cc/editor/javierquintana/911ef749-947b-424b-9f88-36e10e88a877/preview>

<https://create.arduino.cc/editor/javierquintana/911ef749-947b-424b-9f88-36e10e88a877/preview?embed>

Programa 4. con 3 leds

Otra opción es utilizar este programa donde se ve más visual:

<https://create.arduino.cc/editor/javierquintana/701c5f35-8022-4b66-bca2-c99ffd7bb906/preview>

<https://create.arduino.cc/editor/javierquintana/701c5f35-8022-4b66-bca2-c99ffd7bb906/preview?embed>

Conexiones analógicas

Las entradas analógicas se utilizan para leer la información de la magnitud física que nos proporciona los sensores de temperatura, luz, distancia,... La tensión que leemos del sensor nos la proporciona un circuito asociado a dicho sensor en un rango de valores de tensión continua entre 0V y 5V.



La placa de Arduino tiene 6 entradas analógicas marcadas como "A0", "A1", ..., "A5" que reciben los valores continuos en un rango de 0V a 5V, pero la placa Arduino trabaja sólo con valores digitales, por lo que es necesario una conversión del valor analógico leído a un valor digital. La conversión la realiza un circuito analógico/digital incorporado en la propia placa.

El conversor A/D de la placa tiene 6 canales con una resolución de 10 bits. Estos bits de resolución son los que marcan la precisión en la conversión de la señal analógica a digital, ya que cuantos



más bits tenga más se aproxima al valor analógico leído. En el caso de la placa Arduino el rango de los valores analógicos es de 0 a 5 V y con los 10 bits de resolución se puede obtener de 0 a 1023 valores digitales y se corresponde cada valor binario a $(5V/1024)$ 5 mV en el rango analógico.

En estas condiciones son suficientes para hacer muchos proyectos tecnológicos. En el caso de necesitar mayor resolución y como no podemos aumentar el número de bits de conversor A/D se puede variar el rango analógico utilizando el voltaje de referencia V_{ref} .

Las entradas analógicas tienen también la posible utilización como pines de entrada-salida digitales, siendo su enumeración desde 14 al 19.

En este manual dedicamos un capítulo completo a analizar la forma en que Arduino lee magnitudes analógicas. Asimismo veremos numerosos montajes en los que se utilizan estos pines para lectura de sensores.

Señales PWM

Arduino tiene entradas analógicas y digitales. Pero salidas **sólo digitales**.

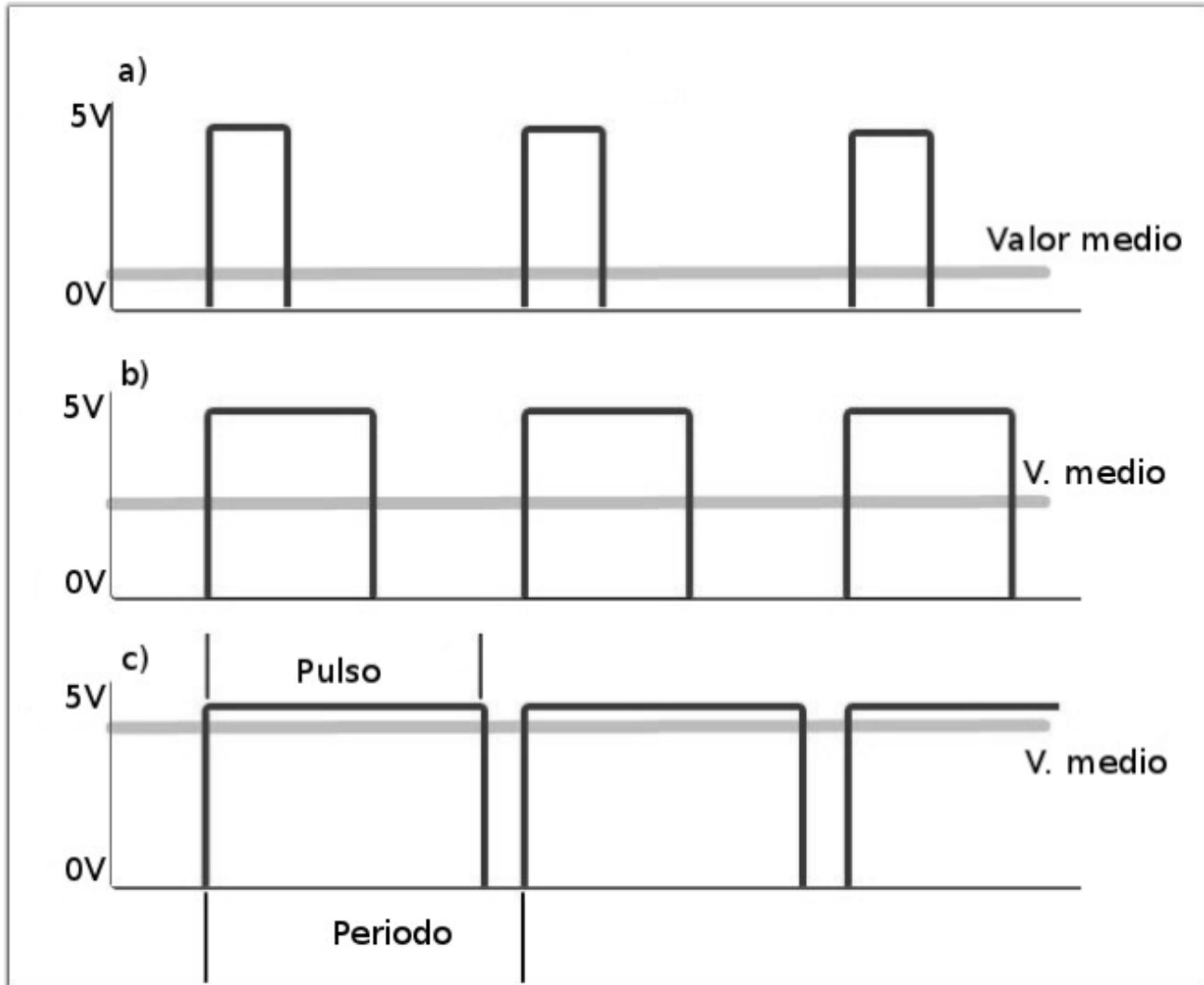
Para **simular** una salida **analógica** entre 0V y 5V se utilizan señales digitales PWM. En Arduino sólo tiene 6 salidas pseudo-analógicas. En los pines digitales 3, 5, 6, 8, 10 y 11 son PWM

¿Qué es PWM?

La señal PWM (*Pulse Width Modulation, Modulación de Ancho de Pulso*) es una señal que utiliza el microcontrolador para generar una señal continua sobre el proceso a controlar. Por ejemplo, la variación de la intensidad luminosa de un led, el control de velocidad de un motor de corriente continua,...

Para que un dispositivo digital, microcontrolador de la placa Arduino, genere una señal continua lo que hace es emitir una señal cuadrada con pulsos de frecuencia constante y tensión de 5V. A continuación, variando la duración activa del pulso (ciclo de trabajo) se obtiene a la salida una señal continua variable desde 0V a 5V.

Veamos gráficamente la señal PWM:



Los pines digitales de la placa Arduino que se utilizan como salida de señal PWM generan una señal cuadrada de frecuencia constante (490Hz), sobre esta señal periódica por programación podemos variar la duración del pulso como vemos en estos 3 casos:

- La duración del pulso es pequeña y la salida va a tener un valor medio de tensión bajo, próximo a 0V.
- La duración del pulso es casi la mitad del período de la señal, por tanto, la salida va a tener un valor medio de tensión próximo a 2,5V.
- La duración del pulso se aproxima al tiempo del período y el valor medio de tensión de salida se aproxima a 5V.

Para ejecutar una señal PWM, es simplemente **`analogWrite(analogOutPin, outputValor);`** donde



- analogOutPin es el número del Pin PWM, acuérdate que sólo puede ser uno de estos 6 : **3, 5, 6, 8, 10 y 11**
- outpuValor es el valor de la señal PWM pero **ojo desde 0 a 255** es decir
 - si quieres el valor de 0V tienes que poner 0
 - si quieres el valor de 5V tienes que poner 255
 - si quieres poner un valor medio, haz una regla de tres, por ejemplo 2.5V tienes que poner $255/2=127$ o 128 da igual

Mapeo

¿Qué es eso de "mapeo"?

En la jerga robótica, dicho pronto y mal pero para que se entienda, mapear significa hacer un **cambio de escala**

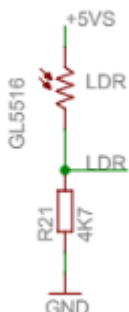
¿Cuándo se dan esas situaciones?

SITUACION A : Queremos leer un valor de entrada analógica en un Arduino, por lo tanto va de 0-1023 y queremos que se copie en una salida digital PWM de Arduino que va de 0-255

SITUACION B : Queremos leer un valor de entrada analógica en un Arduino, por lo tanto va de 0-1023 e interpretarlo en sus valores de voltios. Si suponemos que la placa se alimenta a 5V la variable de salida irá desde 0 a 5V

SITUACION C : Queremos leer el valor de un LDR, que tapándolo nos da 917 e iluminándolo al máximo es 1023, lo queremos copiar en una salida digital PWM, o sea que la salida va desde 0 a 255

Nota: El mínimo de 917 (puede ser otro número, es un valor experimental) es debido a que los LDR van montados en un divisor de tensión como el de la figura, y la resistencia de abajo, siempre se queda algo de tensión





SITUACION D : Queremos según el valor de un joystick conectado a las entradas analógicas de un Arduino (esto pasa en Echidna) se representen en la pantalla de Scratch 2*220 por 2*180, es decir

- Eje X : el potenciómetro (vamos a llamarlo *potx*) va de 0 a 1023 y la salida (*ejex*) va de -220 a 220
- Eje Y : el potenciómetro (vamos a llamarlo *poty*) va de 0 a 1023 y la salida (*ejey*) va de -180 a 180

SITUACION E: Ídem pero no con el potenciómetro, sino con el acelerómetro (vamos a llamarlo *acel*) que va 250 a 500

SITUACION F : Queremos leer un valor de entrada analógica en un Arduino, por lo tanto va de 0-1023 y queremos que se copie en una salida de un servo, por lo tanto lo que necesita es un ángulo que va de 0-180

SITUACION G : Ídem que F pero una raspberry por lo tanto GPI va de 0-65.535

¿Cómo se consigue mapear?

- Si programas con código Arduino IDE, tienes la instrucción **map**
- Si no tienes map, por ejemplo, programas con bloques gráficos tipo Scratch, lo tienes que hacer a mano
 - ¿Cómo? Con la ecuación de una recta

Para entendernos :

- **X** será el valor de entrada que tiene unos valores límites **X₁** e **X₂**
- **Y** es la variable de salida que queremos y que tiene otros valores límites **Y₁** e **Y₂**

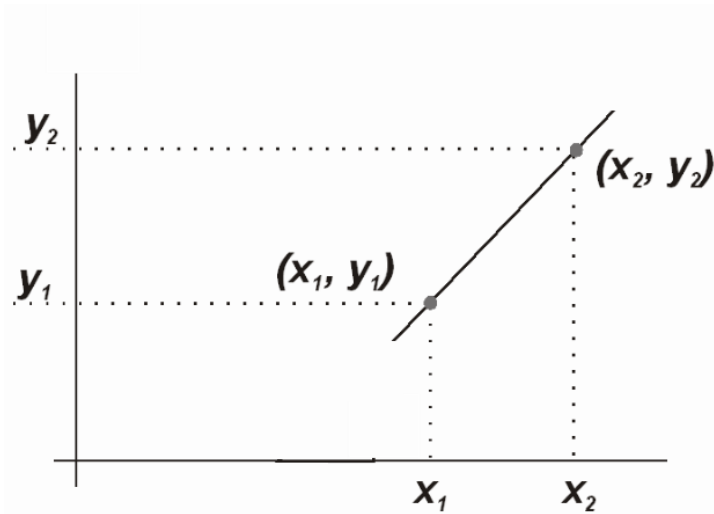
Luego y tiene esta ecuación :

$$y = y_1 + m * (x - x_1)$$

donde m es

$$m = \frac{x_2 - x_1}{y_2 - y_1}$$

Gráficamente



¿Me lo puedes hacer para cada situación anterior?

Si claro:

SITUACION A : Queremos leer un valor de entrada analógica en un Arduino, por lo tanto va de 0-1023 y queremos que se copie en una salida digital PWM de Arduino que va de 0-255

- Límites de las variables :
 - X de 0-1023
 - Y de 0-255
- Con la instrucción map : $Y = \text{map}(X, 0, 1023, 0, 255);$
- Sin la instrucción map $Y = 0.25 * X$ pues $255/1023 = 0.25$ también podemos escribir $Y = X/4$

SITUACION B : Queremos leer un valor de entrada analógica en un Arduino, por lo tanto va de 0-1023 e interpretarlo en sus valores de voltios. Si suponemos que la placa se alimenta a 5V la variable de salida irá desde 0 a 5V

- Límites de las variables :
 - X de 0-1023
 - Y de 0-5
- Con la instrucción map : $Y = \text{map}(X, 0, 1023, 0, 5);$
- Sin la instrucción map $Y = 0.0048 * X$ pues $5/1023 = 0.0048$ o también podemos escribir $Y = X/204$ que queda mejor pues $1023/5=204$ aprox.

SITUACION C : Queremos leer el valor de un LDR, que tapándolo nos da 917 e iluminándolo al máximo es 1023, lo queremos copiar en una salida digital PWM, o sea que la salida va desde 0 a 255

- Límites de las variables :
 - X de 917-1023
 - Y de 0-255
- Con la instrucción map : $Y = \text{map}(X, 917, 1023, 0, 255);$
- Sin la instrucción map $Y = 2.4 * X$ pues $255/(1023-917) = 2.4$

SITUACION D : Queremos según el valor de un joystick conectado a las entradas analógicas de un Arduino (esto pasa en Echidna) se representen en la pantalla de Scratch 2*220 por 2*180, es decir

- Eje X : el potenciómetro (vamos a llamarlo *potx*) va de 0 a 1023 y la salida (*ejex*) va de -220 a 220
- Eje Y : el potenciómetro (vamos a llamarlo *poty*) va de 0 a 1023 y la salida (*ejey*) va de -180 a 180

- EJEX
 - Límites de las variables :
 - *potx* de 0-1023
 - *ejex* de -220 a +220
 - Con la instrucción map : $\text{ejex} = \text{map}(\text{potx}, 0, 1023, -220, 220);$
 - Sin la instrucción map $\text{ejex} = -220 + 0.43 * \text{potx}$ pues $(220 - (-220))/1023 = 0.43$
- EJY
 - Límites de las variables :
 - *poty* de 0-1023
 - *ejey* de -180 a +180
 - Con la instrucción map : $\text{ejey} = \text{map}(\text{poty}, 0, 1023, -180, 180);$
 - Sin la instrucción map $\text{ejey} = -180 + 0.35 * \text{poty}$ pues $(180 - (-180))/1023 = 0.35$

SITUACION E: Ídem pero no con el potenciómetro, sino con el acelerómetro (vamos a llamarlo *acel*) que va 250 a 500

- EJEX
 - Límites de las variables :
 - acelerómetro *acel* de 250-500
 - *ejex* de -220 a +220
 - Con la instrucción map : $\text{ejex} = \text{map}(\text{acel}, 250, 500, -220, 220);$



- Sin la instrucción map ejey = -220 + 1.76*(acel-250) pues $(220 - (-220)) / (500 - 250) = 1.76$
- EJEY
 - Límites de las variables :
 - acelerómetro acel de 250-500
 - ejeY de -180 a +180
 - Con la instrucción map : ejey = map(acel, 250 500, -180, 180);
 - Sin la instrucción map ejey = -180 + 1.44*(acel-250) pues $(180 - (-180)) / (500 - 250) = 1.44$

SITUACION F : Queremos leer un valor de entrada analógica en un Arduino, por lo tanto va de 0-1023 y queremos que se copie en una salida de un servo, por lo tanto lo que necesita es un ángulo que va de 0-180

- Límites de las variables :
 - X de 0-1023
 - Y de 0-180
- Con la instrucción map : Y = map(X, 0, 1023, 0, 180);
- Sin la instrucción map Y = 0.17* X pues $180/1023 = 0.17$ también podemos escribir Y = X/5.7 pues $1023/180 = 5.7$

SITUACION G : Idem que F pero una raspberry por lo tanto GPI va de 0-65535

- Límites de las variables :
 - X de 0-65535
 - Y de 0-180
- Con la instrucción map : Y = map(X, 0, 65535, 0, 180);
- Sin la instrucción map Y = 0.00274* X pues $180/65535 = 0.00274$ pero es más cómodo al revés Y = X/364 pues $65535/180 = 364$