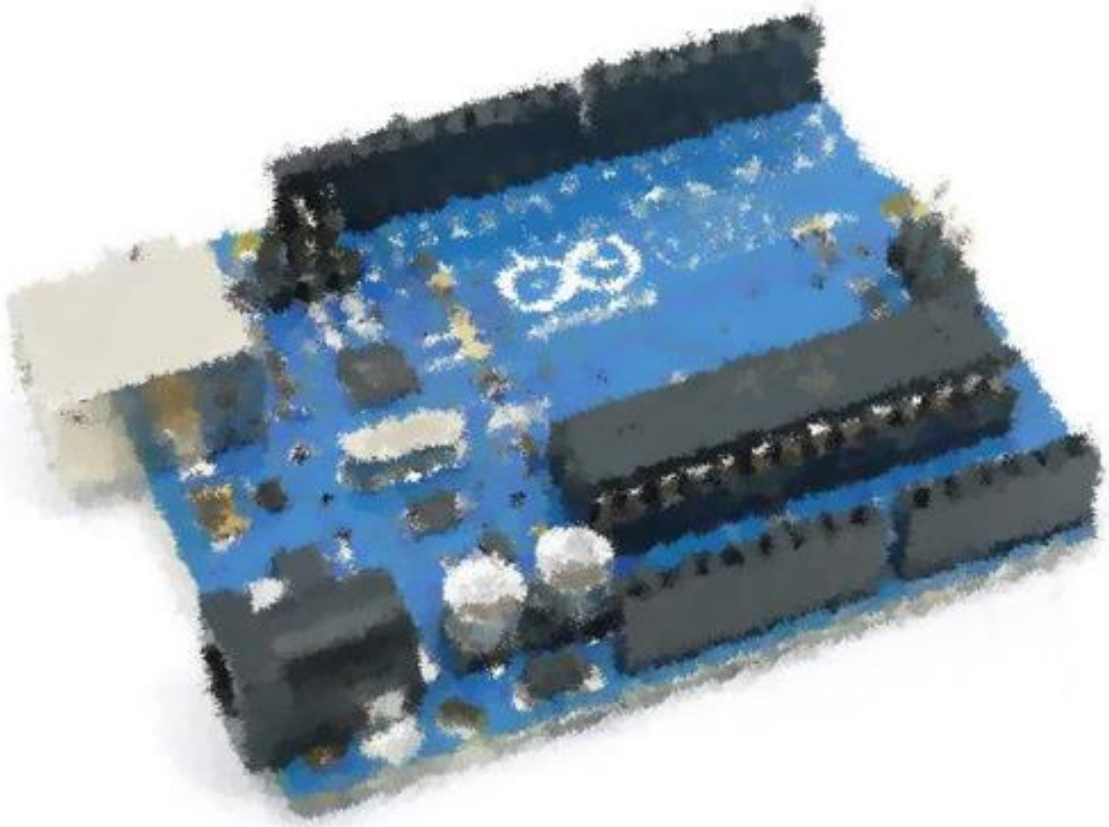
CAPÍTULO

2 nivel pardillo

Genuino ARDUINO



Daniel Gallardo García
Profesor de Tecnología
Jerez de la Frontera

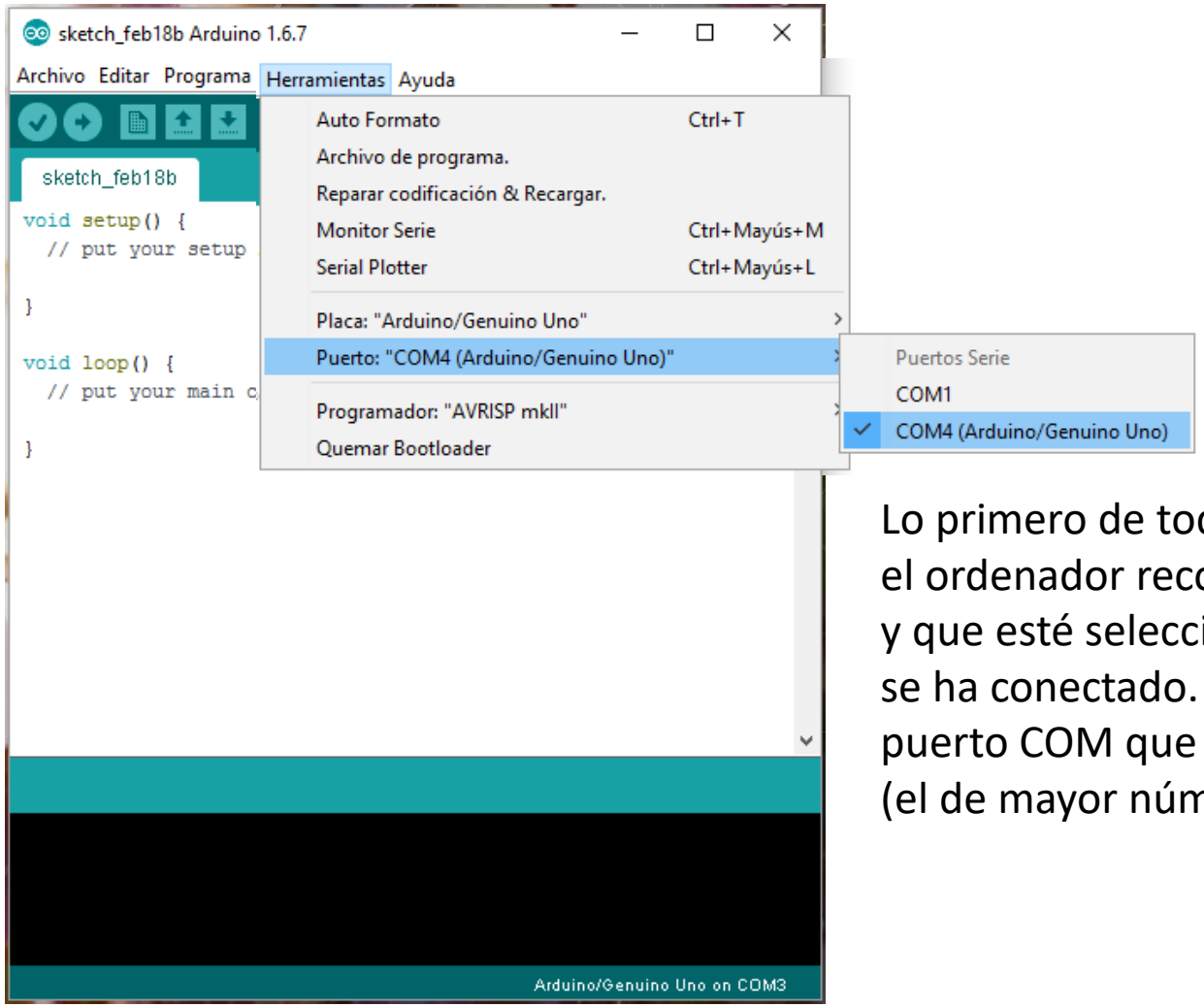
Índice

La IDE de Arduino
Variables
Configuración
Salidas Digitales
Estructuras: Condicional
Operadores lógicos
Estructuras: Bucle *for*
Operadores de incrementos
Salidas Analógicas
Entradas Digitales
Sensores Digitales: el pulsador
Comunicación con el PC
Entradas Analógicas
Sensores Analógicos: el potenciómetro
Sensores Analógicos: la LDR
Mapear datos
Restringir datos

Daniel Gallardo García
Profesor de Tecnología
Jerez de la Frontera

La IDE de Arduino

Utilizaremos Arduino 1.6.7 para programar desde el ordenador a nuestra placa Arduino a través de un cable USB.



Lo primero de todo es asegurarse de que el ordenador reconoce a la placa Arduino, y que esté seleccionado el puerto al que se ha conectado. Debemos seleccionar el puerto COM que nos aparece más abajo (el de mayor número).

Programando Arduino

Todo programa para Arduino presenta una **estructura básica**:

1ª parte `int x = 0;` **Declarar** las **variables globales**.

2ª parte `void setup() {…}` **Configuración** de Arduino (se ejecutará una sola vez).

3ª parte `void loop() {…}` **Comandos** que regirán el comportamiento de Arduino (se ejecutará en un bucle infinito y sin pausa).

Variables

Podemos entender el concepto de variable si la comparamos con una cajita o maletín donde podemos guardar cosas (números, letras, valores lógicos…), y más adelante poder utilizar lo que hay dentro de ella, cambiarle el contenido, etc.

Antes de utilizar una variable, debemos **declararla** (para que Arduino la reconozca) indicando qué **tipo** de contenido que va a almacenar y poniéndole un **nombre** y también podemos **inicializarla** (darle un valor inicial). Si no se inicializa, Arduino le asignará el valor 0 o nulo por defecto.

```
int x = 0;          /*declaro la variable x, que almacenará un número entero,
                    y almaceno (o asigno) el número 0 en ella */

int y;              /*declaro la variable y, que almacenará un número con
                    entero (que no puede tener decimales), pero no la
                    inicializo */

y = 7;              //asigno a la variable (que previamente he declarado)
                    el valor 7
```

Variables

Una variable es un valor que Arduino puede almacenar en su memoria, y que posteriormente podrá ser utilizado o modificado. Los **tipos de variables** más utilizados son:

`int` almacena un **número entero** entre -32769 y 32767 (2 bytes).

`long` **número entero** muy largo, entre -2147483648 y 2147483647 (4 bytes).

`float` **número decimal** con un rango entre $-3.4028235 \cdot 10^{38}$ y $3.4028235 \cdot 10^{38}$ (4 bytes).

Dominio de una variable:

- Si **declaro** una variable **al comienzo** del sketch (variable **global**), podré emplear dicha variable en cualquier momento (dentro de cualquier función o bloque del programa),

- Si **declaro** una variable **dentro de una función** (variable **local**), sólo se podrá utilizar en dicha función.

Configuración `void setup() {…}`

En este bloque de código habrá que especificar:

- Qué **pines** van a ser utilizados como **entradas** y cuáles como **salidas**:

```
pinMode(2, OUTPUT);          //utilizaré el pin 2 como salida Digital
pinMode(3, OUTPUT);          //utilizaré el pin ~3 como salida Digital o Analógica
pinMode(8, INPUT);           //utilizaré el pin 8 como entrada Digital

/* Los pines A0, A1, … A5 ya están preconfigurados como entrada Analógica, y
no será necesario configurarlos a menos que queramos utilizarlos como salida o
entrada digital */
```

- Si queremos establecer **comunicación** con el **ordenador**:

```
Serial.begin(9600);          //hay que especificar los bits/segundo (baudios)
```

- Si queremos **generar números pseudo-aleatorios**:

```
randomSeed(analogRead(A0)); /*inicia una generación de números aleatorios a
                              partir de la lectura analógica del pin A0 */
```

Bucle infinito

`void loop() {…}`

En este bloque de código se deberán escribir todas aquellas instrucciones, órdenes, primitivas, comandos o funciones necesarias **para que Arduino funcione según nuestro propósito**.

Iremos viendo cuáles son esas **funciones** durante el desarrollo de estos apuntes.

Salidas Digitales

`digitalWrite(4, HIGH);`

Podemos indicar a Arduino que en un pin de salida digital (del 0 al 13) coloque un "**0**" o un "**1**" lógico (que serán **0 V** o **5 V**) mediante los siguientes comandos, respectivamente:

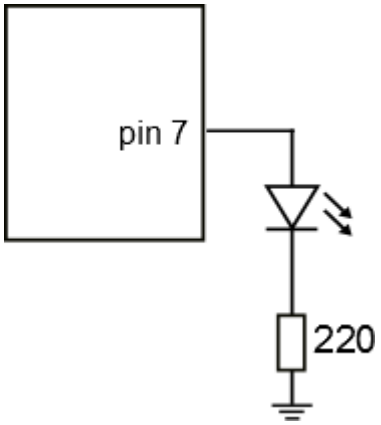
```
digitalWrite(4, LOW);       //pondrá 0 V en el pin 4

digitalWrite(4, HIGH);      //pondrá 5 V en el pin 4
```

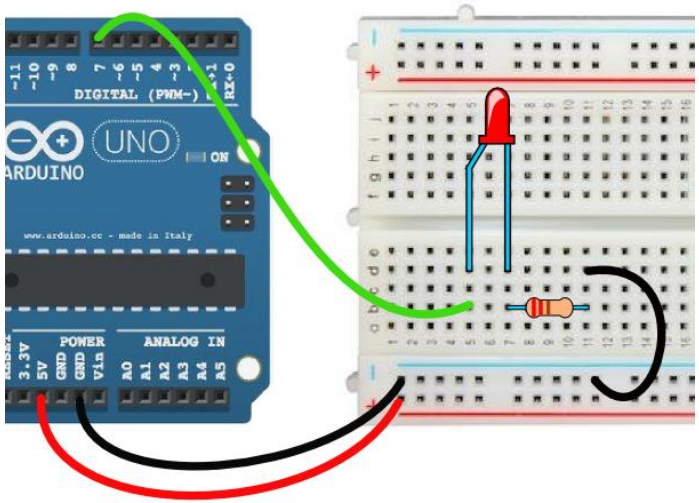
Si queremos que Arduino haga una **pausa**, y que durante ese tiempo no continúe leyendo el código del sketch, debemos utilizar la siguiente función:

```
delay(200);         //hace una pausa en el programa de 200 milisegundos
```

Ejemplo: Parpadeo de un LED

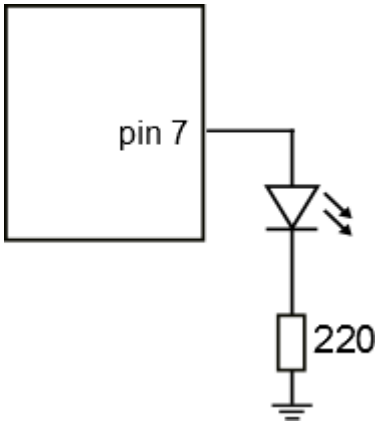
Conectaremos a un pin de salida digital un LED (y su respectiva resistencia), y haremos un sketch para que parpadee a intervalos de 1 s:

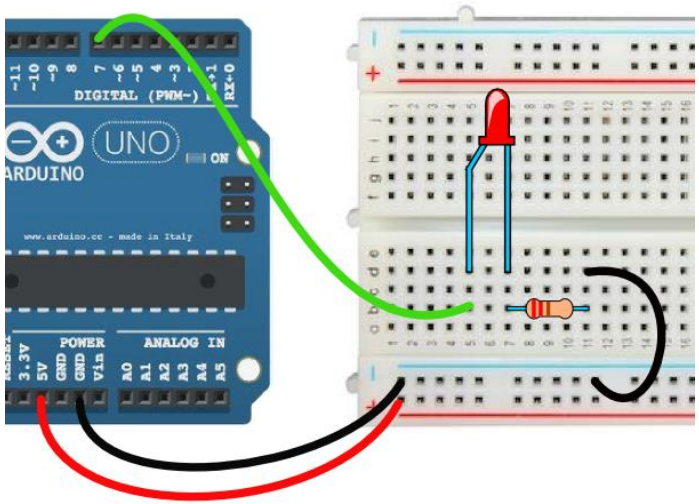




Ejemplo: Parpadeo de un LED

Conectaremos a un pin de salida digital un LED (y su respectiva resistencia), y haremos un sketch para que parpadee a intervalos de 1 s:






```
void setup() {
  pinMode(7, OUTPUT);       //configuro el pin 7 como salida
}

void loop() {
  digitalWrite(7, HIGH);    // enciendo el LED al poner el nivel HIGH en el pin
  delay(1000);              // espera 1 segundo
  digitalWrite(7, LOW);     // apaga el LED al poner el nivel LOW en el pin
  delay(1000);              // espera 1 segundo
}
//este bucle se repetirá infinitas veces
```

Estructuras: Condicional `if(x == 3) {…}`

Se utiliza esta estructura para hacer que Arduino ejecute ciertas órdenes en caso de cumplirse alguna condición, es decir: que presente un **comportamiento condicional**.

Dicha condición consiste en una **expresión booleana**, es decir, que solo hay dos posibilidades: que se cumpla o que no se cumpla, que sea verdadero o falso, que sea `true` o `false`. Veamos algunos ejemplos:

```
a == 3    //la variable a es igual a 3

b > 10    //la variable b es mayor de 10

c < 25    //la variable c es menor de 25

d >= -7   //la variable d es mayor o igual que -7

e <= 206  //la variable e es menor o igual que 206

f != 0    //la variable f es distinta de 0
```

No debemos confundir la asignación de una variable (=) con la expresión booleana (==):

```
x = 2     //guardo el número 2 en la variable x
x == 2    //comparo si lo que hay en la variable x es un 2 o no
```

Estructuras: Condicional_2 `if(x > 3) {…} else {…}`

Podemos hacer que Arduino tenga un comportamiento condicional con la siguiente estructura:

```
if(x < 10) {
  /*este bloque de código se ejecutará únicamente si el valor de la variable x
  es menor de 10, es decir, si la expresión booleana es verdadera */
}

if(digitalRead(2)) {
  /*este bloque de código se ejecutará únicamente si la lectura digital del
  pin 2 equivale a 1 o true */
}
```

Hay otra variante, donde le podemos decir a Arduino que tenga **otro comportamiento en el caso contrario**, es decir, en el caso en el que no se cumpla la condición:

```
if(x < 10) {
  //se ejecutará si el valor de la variable x es menor de 10
}
else {
  /*este bloque de código se ejecutará en el caso contrario (cuando x sea
  mayor o igual que 10 */
}
```

Estructuras: Condicional_3 `if() {…} else if {…}`

Con `if(){ } else(){ }` Arduino puede tomar dos comportamientos distintos (o coge un camino o coge el otro). Si quisiéramos que existiesen más opciones, debemos añadir nuevas condiciones de la siguiente manera:

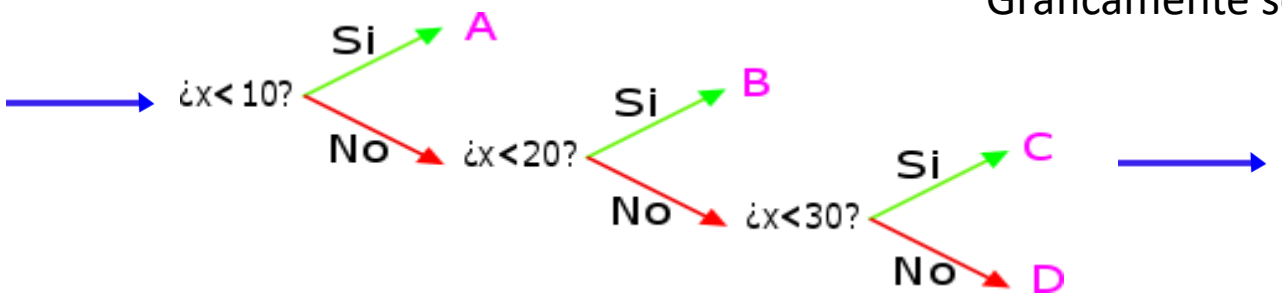
```
if(x < 10) {
  //se ejecutará si el valor de la variable x es menor de 10
}
else if(x < 20) {
  /*este bloque de código se ejecutará si el valor de x es mayor e igual
  que 10 y menor de 20 */
}
else if(x < 30) {
  /*este bloque de código se ejecutará si el valor de x es mayor e igual
  que 20 y menor de 30 */
}
else {
  /*este bloque de código se ejecutará si no ha cumplido ninguna de las
  condiciones anteriores (cuando x sea mayor o igual que 30) */
}
```

Estructuras: Condicional_3 `if() {…} else if {…}`

Con `if(){ } else(){ }` Arduino puede tomar dos comportamientos distintos (o coge un camino o coge el otro). Si quisiéramos que existiesen más opciones, debemos añadir nuevas condiciones de la siguiente manera:

```
if(x < 10) {                                                    A
  //se ejecutará si el valor de la variable x es menor de 10
}
else if(x < 20) {                                               B
  /*este bloque de código se ejecutará si el valor de x es mayor e igual
  que 10 y menor de 20 */
}
else if(x < 30) {                                               C
  /*este bloque de código se ejecutará si el valor de x es mayor e igual
  que 20 y menor de 30 */
}
else {                                                          D
  /*este bloque de código se ejecutará si no ha cumplido ninguna de las
  condiciones anteriores (cuando x sea mayor o igual que 30) */
}
```

Gráficamente sería algo como esto:



Operadores lógicos && || !

¿Cómo se expresa una condición como: *x debe se mayor que 3 y menor que 7*? ¿O *x debe ser o menor de 2 o mayor de 5*? Existen 3 operadores lógicos que pueden emplearse: `&&`, `||` y `!`

```
&&  /*si lo que queremos es que se cumplan simultáneamente más de una
    condición o expresión booleana. Equivale al operador lógico AND */

||  /*si lo que queremos es que se cumpla al menos una de entre varias
    condiciones o expresiones booleanas (| se consigue con Alt Gr + 1).
    Equivale al operador lógico OR */

!   /*si lo que queremos es que se cumpla lo contrario a unan condición.
    Equivale al operador lógico NOT */
```

Veamos algunos ejemplos:

```
if(x > 3 && x < 7) //se ejecutará si x está comprendido entre 3 y 7

if(x < 2 || x > 5) //se ejecutará si x es menor de 2 o mayor de 5

if(!digitalRead(2)) //se ejecutará si la lectura digital es 0 o false
```

Estructuras: Bucle for `for(int i=0; i<3; i++) {…}`

Esta estructura permite realizar un bucle un número de veces en función de una variable (por costumbre se le llamará `i`). Será necesario inicializar dicha variable, establecer una condición (que mientras se cumpla, se realizará el bucle), y expresar una iteración (incrementar un cierto valor a la variable).

```
for(int i=4; i<10; i=i+1) {    /*debo inicializar la variable i (y declararla
                    en caso de que no se trate de una variable global) */
  //mientras i sea menor de 10, ejecutará este bloque de código
  //por último se produce la iteración, e i incrementará en 1 su valor
}
  /*en este caso, realizará el bucle para los siguientes valores de i:
  4, 5, 6, 7, 8, y 9; es decir, realizará 6 veces el bucle. Además, puedo
  utilizar el valor de la variable i durante el bucle. Cuando se acabe el
  bucle, si la variable i es global, saldría con el valor de 10 */
```

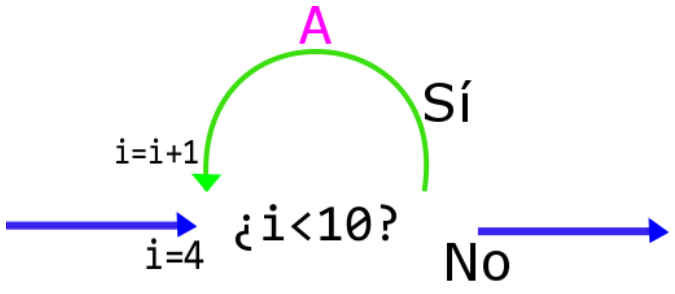
Como la variable `i` se ha declarado dentro del bucle `for`, no tendrá validez fuera de él (variable local).

Estructuras: Bucle for `for(int i=0; i<3; i++) {…}`

Esta estructura permite realizar un bucle un número de veces en función de una variable (por costumbre se le llamará `i`). Será necesario inicializar dicha variable, establecer una condición (que mientras se cumpla, se realizará el bucle), y expresar una iteración (incrementar un cierto valor a la variable).

```
for(int i=4; i<10; i=i+1) { A /*debo inicializar la variable i (y declararla
                    en caso de que no se trate de una variable global) */
  //mientras i sea menor de 10, ejecutará este bloque de código
  //por último se produce la iteración, e i incrementará en 1 su valor
}
  /*en este caso, realizará el bucle para los siguientes valores de i:
  4, 5, 6, 7, 8, y 9; es decir, realizará 6 veces el bucle. Además, puedo
  utilizar el valor de la variable i durante el bucle. Cuando se acabe el
  bucle, si la variable i es global, saldría con el valor de 10 */
```

Gráficamente sería algo como esto:



Como la variable `i` se ha declarado dentro del bucle `for`, no tendrá validez fuera de él (variable local).

Operadores de incrementos

i++

Dicha variable va **incrementándose** cada vez que se repite el `for`. El incremento puede expresarse de las siguientes maneras:

```
i = i + 5          //el valor de i se incrementa en 5
i += 5             //el valor de i se incrementa en 5 (es otra forma)
i = i + 1          //el valor de i se incrementa en 1
i += 1             //el valor de i se incrementa en 1 (es otra forma)
i++                //el valor de i se incrementa en 1 (sólo para incremento +1)

i = i - 1          //el valor de i disminuye en 1
i -= 1             //el valor de i disminuye en 1 (es otra forma)
i--                //el valor de i disminuye en 1 (sólo para incremento -1)

i = i * 3          //el valor de i se multiplica por 3
i *= 3             //el valor de i se multiplica por 3 (es otra forma)

i = i / 2          //el valor de i se divide entre 2
i /= 2             //el valor de i se divide entre 2 (es otra forma)
```

Ejemplo: Parpadeo de un LED, cada vez más rápido

Ahora se hace necesario utilizar variables:

```
int t = 1000;                 //será el tiempo de las pausas
int pinLed = 7;               //será el pin donde conecto el LED

void setup() {
  pinMode(pinLed, OUTPUT);
}

void loop() {
  digitalWrite(pinLed, HIGH);
  delay(t);
  digitalWrite(pinLed, LOW);
  delay(t);
  t -= 50;                    //después de cada ciclo, recorto 50 ms el tiempo
  if(t < 1) {                 //cuando el tiempo sea menor de 1 ms…
    t = 1000;                 //reinicio t a 1000 ms y…
    delay(2000);              //hago una pausa de 2 s
  }
}
```

Ejemplo: Parpadeo de un LED, cada vez más rápido

Ahora se hace necesario utilizar variables:

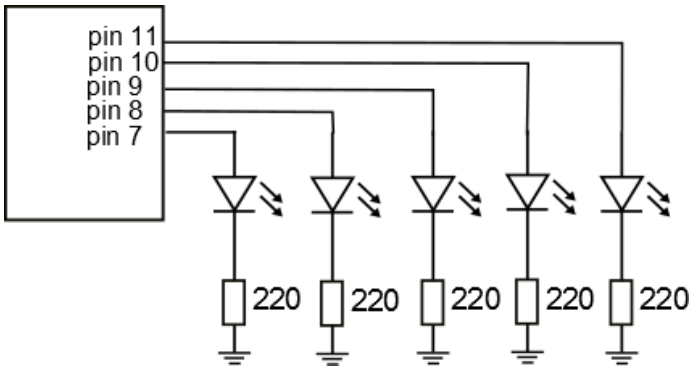
```
int t = 1000;                 //será el tiempo de las pausas
int pinLed = 7;               //será el pin donde conecto el LED

void setup() {
  pinMode(pinLed, OUTPUT);
}

void loop() {
  digitalWrite(pinLed, HIGH);
  delay(t);
  digitalWrite(pinLed, LOW);
  delay(t);
  t -= 50;                    //después de cada ciclo, recorto 50 ms el tiempo
  if(t < 1) {                 //cuando el tiempo sea menor de 1 ms…
    t = 1000;                 //reinicio t a 1000 ms y…
    delay(2000);              //hago una pausa de 2 s
  }
}
```

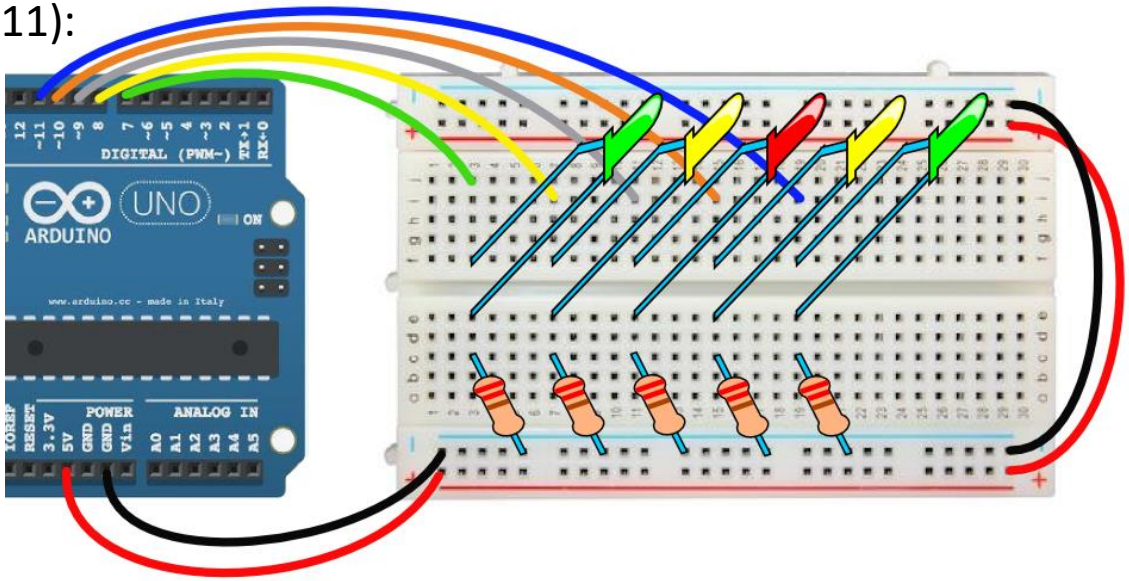
No siempre un incremento lineal (restando un poco cada vez) da buen resultado. Sustitúyelo por esto otro:

```
t  /= 1.1;          //cada vez se divide entre 1,1
```

Ejemplo: Luz que avanza por 5 LEDs

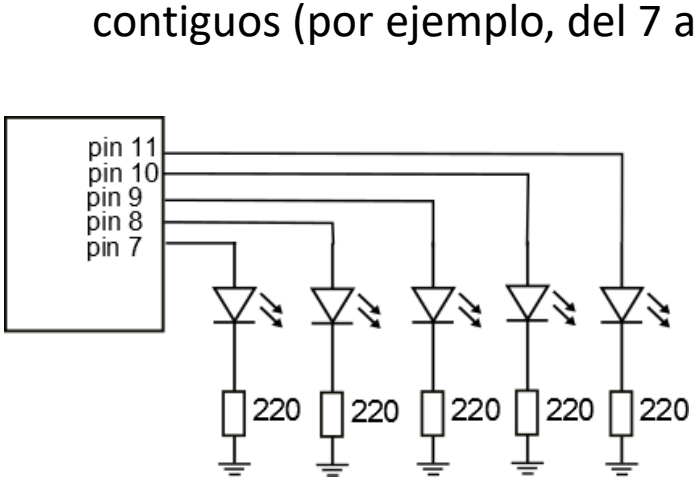
Conectaremos los 5 LEDs (con sus correspondientes resistencias) a 5 pines contiguos (por ejemplo, del 7 al 11):

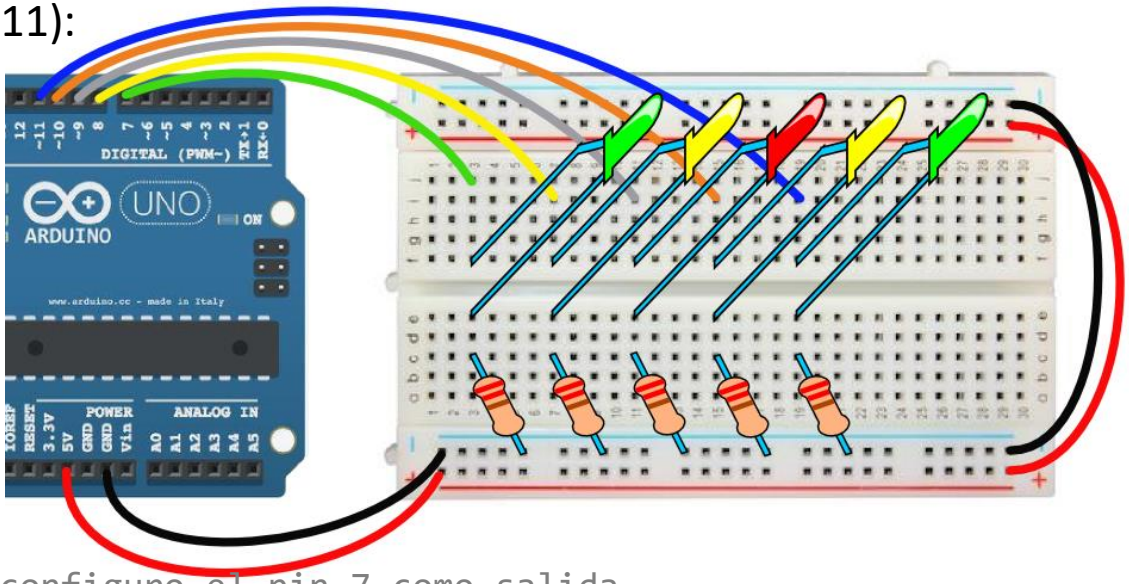




Ejemplo: Luz que avanza por 5 LEDs

Conectaremos los 5 LEDs (con sus correspondientes resistencias) a 5 pines contiguos (por ejemplo, del 7 al 11):

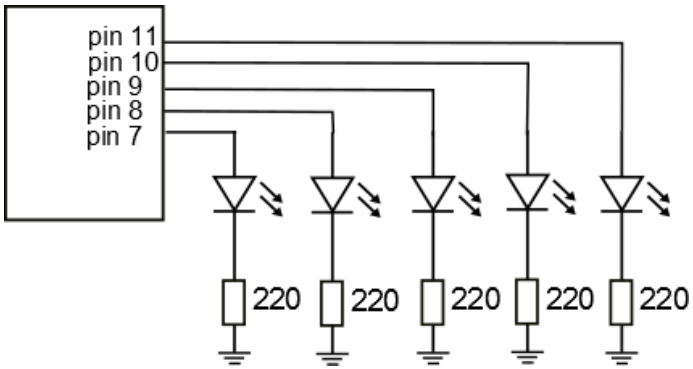


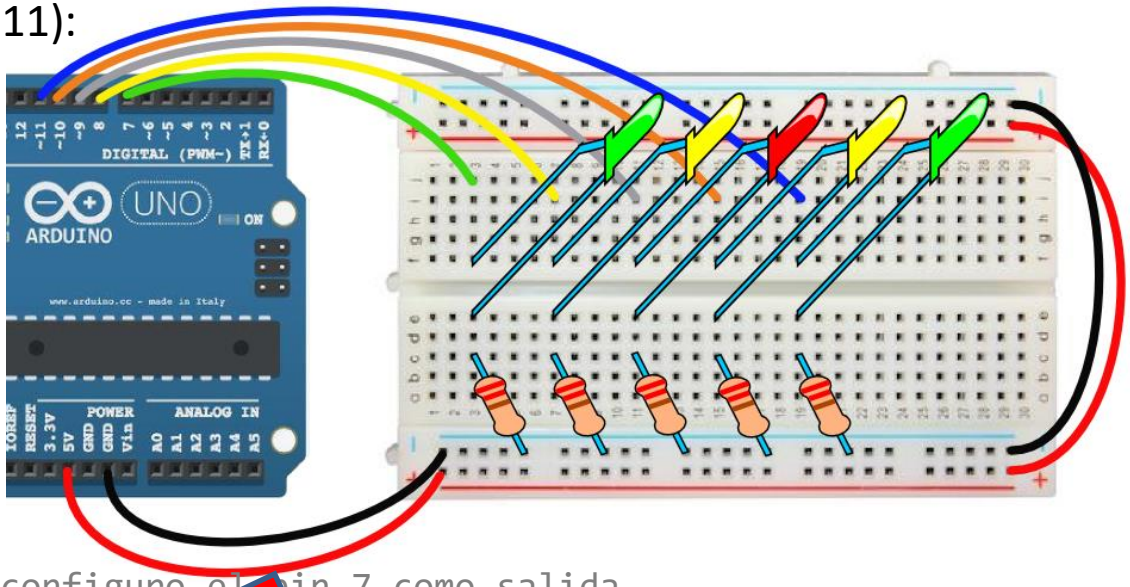



```
void setup() {
  pinMode(7, OUTPUT);       //configuro el pin 7 como salida
  pinMode(8, OUTPUT);       //configuro el pin 8 como salida
  pinMode(9, OUTPUT);       //configuro el pin 9 como salida
  pinMode(10, OUTPUT);      //configuro el pin 10 como salida
  pinMode(11, OUTPUT);      //configuro el pin 11 como salida
}

void loop() {
  digitalWrite(7, HIGH);    //enciendo el pin 7
  delay(1000);              //espero 1 s
  digitalWrite(7, LOW);     //apago el pin 7
  digitalWrite(8, HIGH);    //enciendo el pin 8
  delay(1000);              //espero 1 s
```

Ejemplo: Luz que avanza por 5 LEDs

Conectaremos los 5 LEDs (con sus correspondientes resistencias) a 5 pines contiguos (por ejemplo, del 7 al 11):



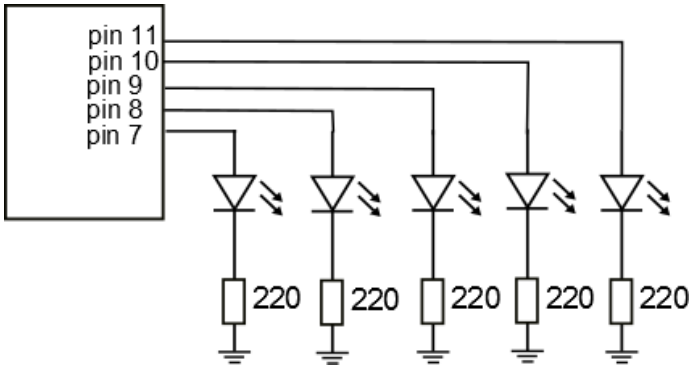



```
void setup() {
  pinMode(7, OUTPUT);      //configuro el pin 7 como salida
  pinMode(8, OUTPUT);      //configuro el pin 8 como salida
  pinMode(9, OUTPUT);      //configuro el pin 9 como salida
  pinMode(10, OUTPUT);     //configuro el pin 10 como salida
  pinMode(11, OUTPUT);     //configuro el pin 11 como salida
}

void loop() {
  digitalWrite(7, HIGH);   //enciendo el pin 7
  delay(1000);             //espero 1 s
  digitalWrite(7, LOW);    //apago el pin 7
  digitalWrite(8, HIGH);   //enciendo el pin 8
  delay(1000);             //espero 1 s
```

¡Esto huele a catetada!...

Ejemplo: Luz que avanza por 5 LEDs_2

Utilizaré el bucle `for` para acortar el código:




```
void setup() {
  for(int i=7; i<12; i++) pinMode(i, OUTPUT);
          //configuro los 5 pines en una sola línea de código
}

void loop() {
  for(int i=7; i<12; i++) {
    digitalWrite(i, HIGH);            //enciendo el LED i
    digitalWrite(i-1, LOW);           //apago el LED anterior
    delay(1000);
  }
  digitalWrite(11, LOW);     /*apago el último LED puesto que
                             nunca se llega a i-1 = 11 */
}
```

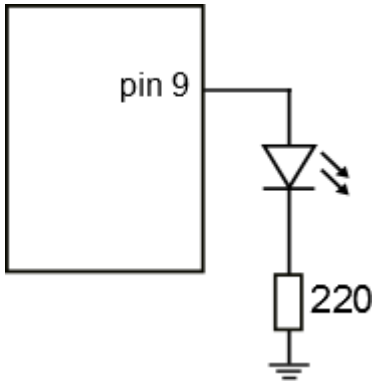
Salidas Analógicas `analogWrite(5, 128);`

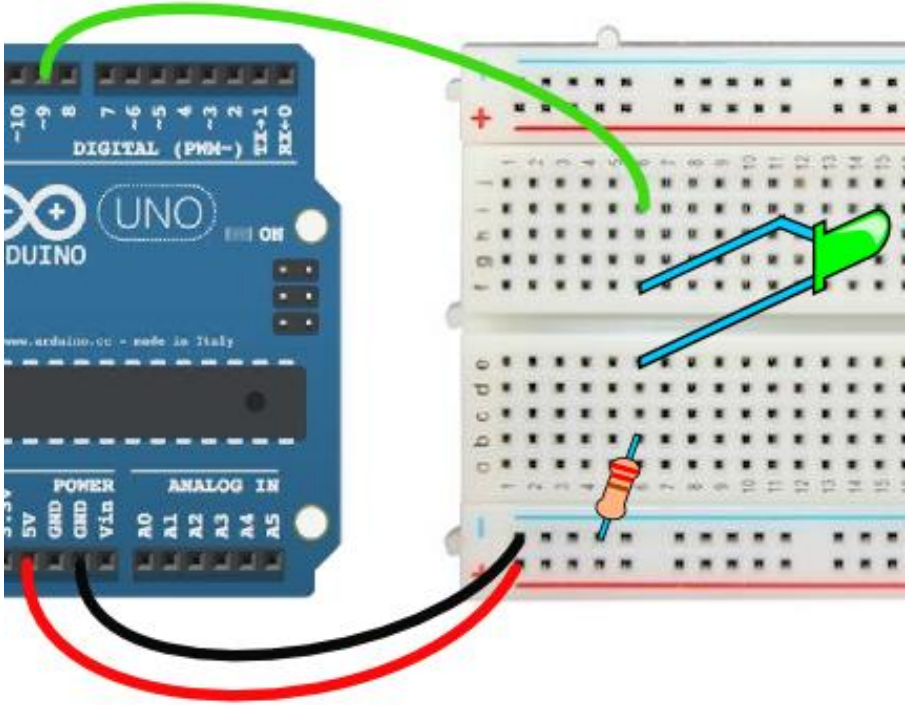
Podemos indicar a Arduino que en un pin de salida analógica (`~3`, `~5`, `~6`, `~9`, `~10` y `~11`) coloque un valor de tensión comprendido entre 0 V y 5 V, pudiendo seleccionar entre 256 valores intermedios posibles (de **0** a **255**), empleando la siguiente función:

```
analogWrite(11, 214);        /*pondrá en el pin ~11 un valor de salida de 214,
                               que equivale a unos 4,2 V */

analogWrite(3, 0);           /*pondrá en el pin ~3 un valor de salida de 0,
                               que equivale a 0 V o nivel LOW */
```

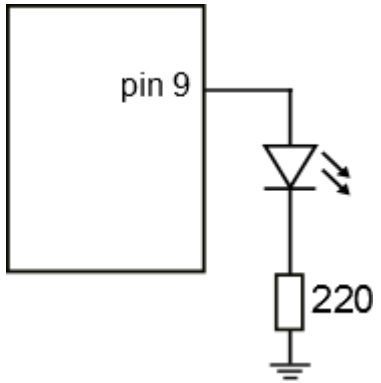
Ejemplo: Parpadeo gradual de un LED

Conectaremos a un pin de salida analógica (el ~9, por ejemplo) un LED (y su respectiva resistencia), y haremos un sketch para que se encienda y se apague de forma gradual:





Ejemplo: Parpadeo gradual de un LED_2




```
int brillo = 0;       //será el valor de la salida analógica
int incremento = 5;   //el brillo cambiará de 5 en 5
int pinLed = 9;

void setup() {
  pinMode(pinLed, OUTPUT);            //el pin 9 será la salida
}

void loop() {
  analogWrite(pinLed, brillo);        //coloca el valor brillo en el pin 9
  brillo += incremento;               //brillo aumenta su valor en 5
  if(brillo == 0 || brillo == 255) {  //si brillo llega a sus límites, pasamos de...
    incremento = -incremento;         //...crecer a decrecer, y viceversa
  }
  delay(30);                  /*hay que dar un pequeño tiempo entre valor y valor
                              de brillo para que la variación no sea instantánea */
}
```

Entradas Digitales

digitalRead(4);

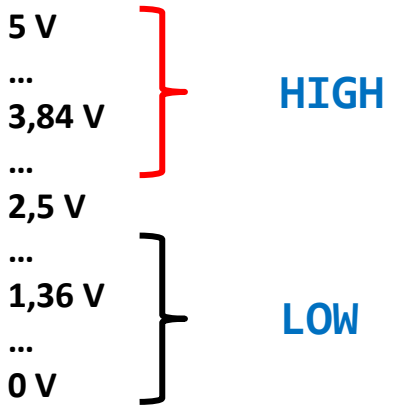
Podemos indicar a Arduino que lea en un pin de entrada digital (del 0 al 13) el valor digital que haya en él (LOW o HIGH) mediante la siguiente función:

```
val = digitalRead(6);        /*asignará a la variable val el valor de la
                              lectura digital del pin 6:
si val es utilizada como un número, podrá ser 0 (si es LOW) o 1 (si es HIGH),
si val es utilizada como una expresión booleana, podrá ser false (si es LOW)
                                                    o true (si es HIGH)*/
```

Recordemos que en el setup se debió haber configurado el pin 6 como entrada y haber declarado con anterioridad la variable val:

```
int val;

void setup() {
  pinMode(6, INPUT);
}
```

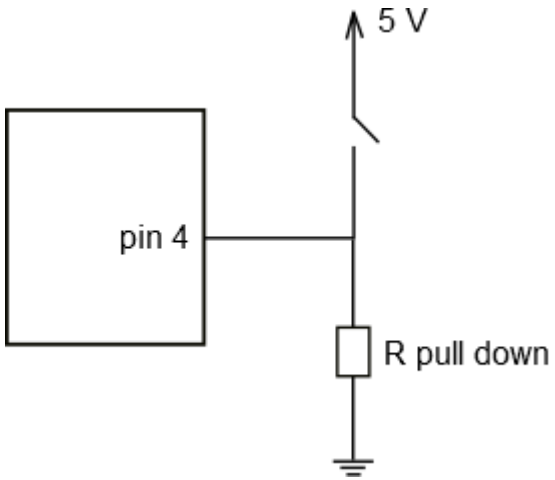
Si en el pin 6 hay **menos de 2,5 V** Arduino lo interpretará como una lectura de valor **LOW**, y si hay **más de 2,5 V** lo interpretará como un valor **HIGH**.

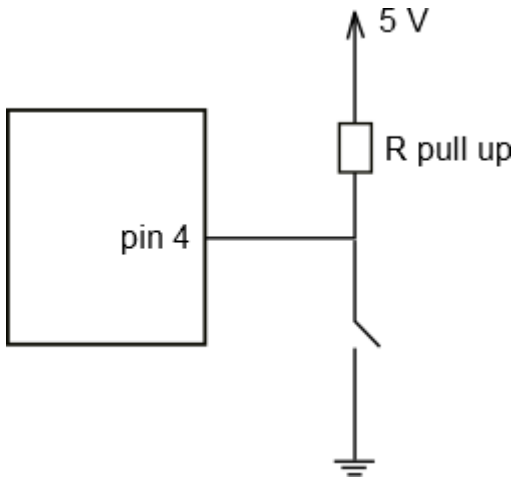


Sensores Digitales: el pulsador

Cualquier sensor que tenga dos posibles valores distintos puede emplearse como sensor digital. Un **pulsador** es un buen ejemplo.

Podemos hacer una entrada digital con un **pulsador** y una **resistencia de drenaje**, de 10K por ejemplo, con cualquiera de las siguientes configuraciones:



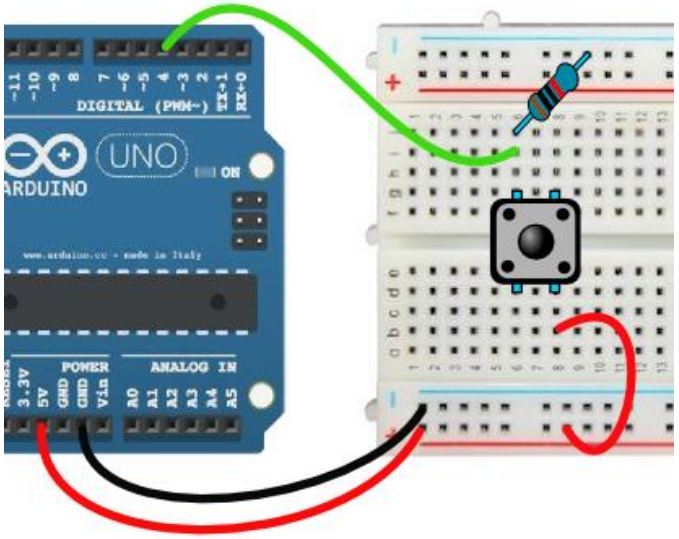


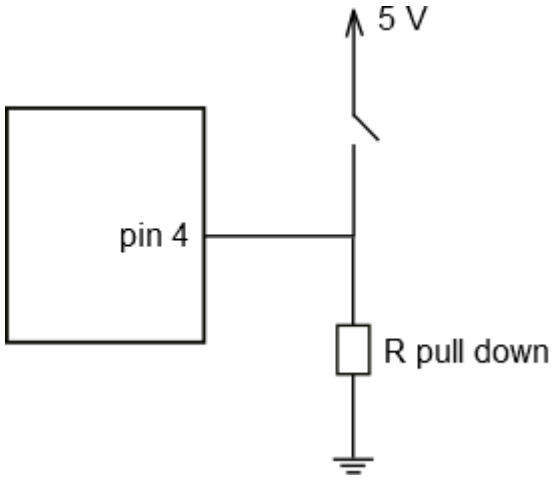
Apretado (ON) → `digitalRead(4) = HIGH`

Sin apretar (OFF) → `digitalRead(4) = LOW`

Apretado (ON) → `digitalRead(4) = LOW`

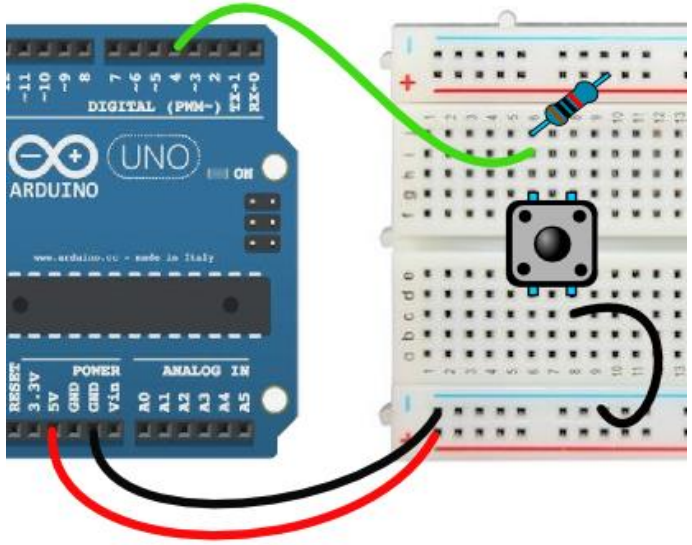
Sin apretar (OFF) → `digitalRead(4) = HIGH`

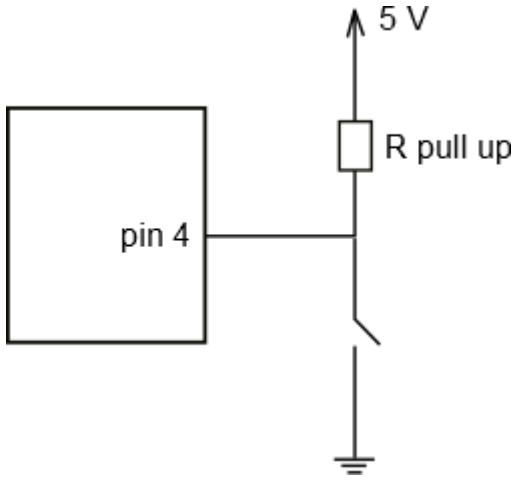




Apretado (ON) → `digitalRead(4)` = **HIGH**

Sin apretar (OFF) → `digitalRead(4)` = **LOW**

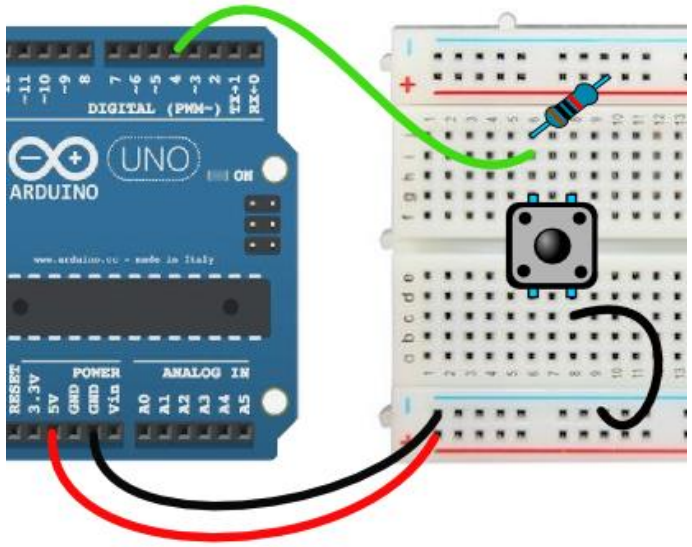


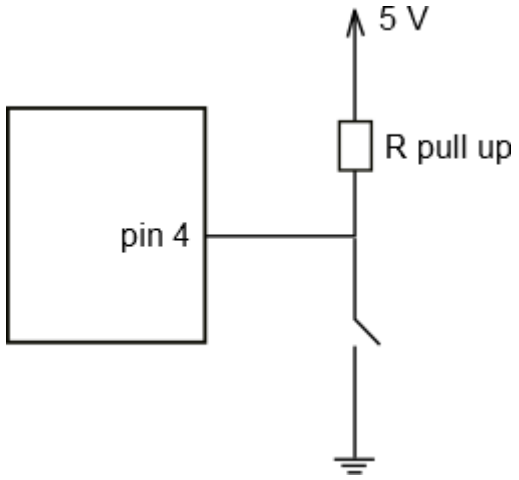


Apretado (ON) → `digitalRead(4)` = **LOW**

Sin apretar (OFF) → `digitalRead(4)` = **HIGH**

Esta segunda configuración (con la resistencia pull up) presenta una ventaja: Podemos **utilizar una resistencia interna del microprocesador ATmega**, y nos ahorraríamos el tener que ponerla.

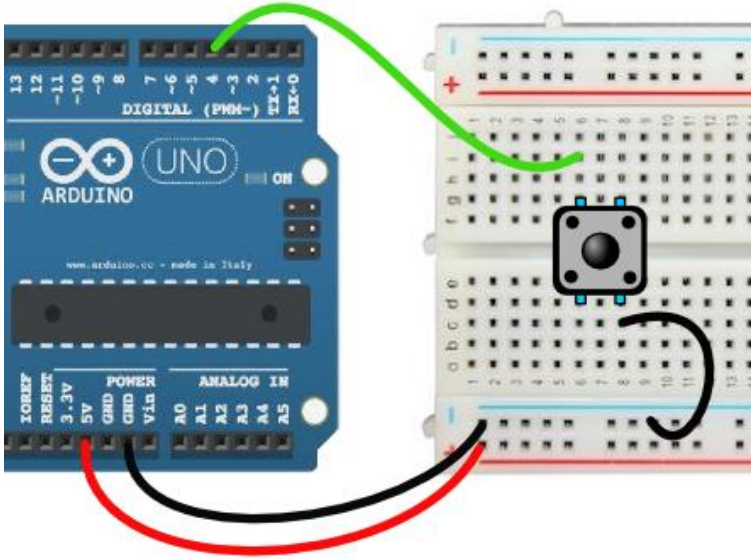




Apretado (ON) → `digitalRead(4) = LOW`

Sin apretar (OFF) → `digitalRead(4) = HIGH`

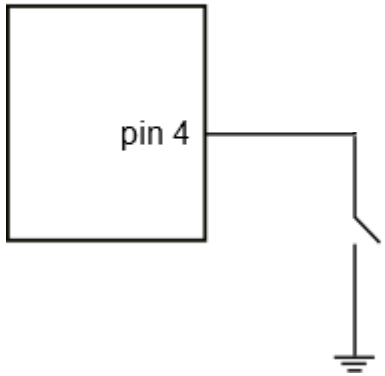
Esta segunda configuración (con la resistencia pull up) presenta una ventaja: Podemos **utilizar una resistencia interna del microprocesador ATmega**, y nos ahorraríamos el tener que ponerla.



Conectaríamos el pulsador como se muestra en la imagen:

Para hacer uso de dicha resistencia interna, debemos hacer una pequeña **modificación** a la hora de **configurar el pin de entrada**:

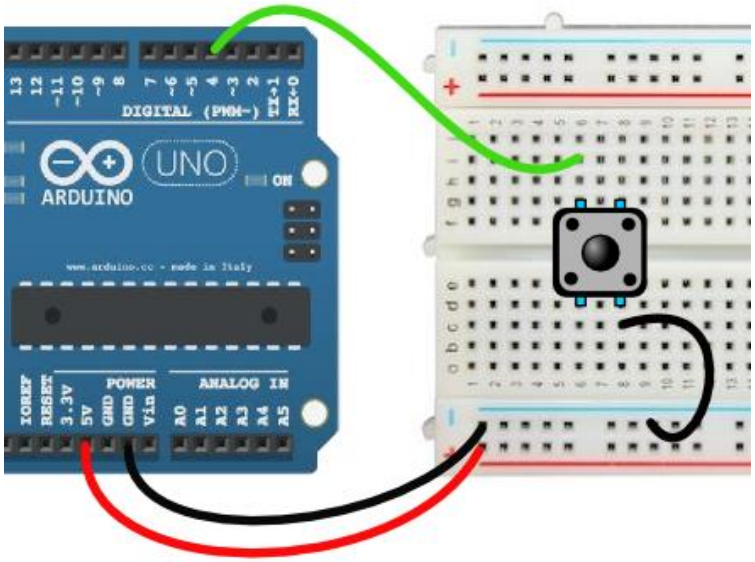
```
pinMode(4, INPUT_PULLUP);
```



Apretado (ON) → `digitalRead(4)` = **LOW**

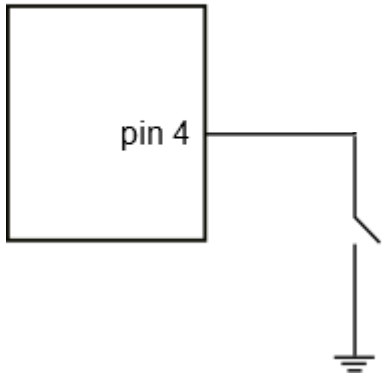
Sin apretar (OFF) → `digitalRead(4)` = **HIGH**

Debemos tener en cuenta que Arduino va a percibir todo el transitorio (sobreoscilación o bouncing) cada vez que el pulsador sea accionado, en el cambio de 5 a 0 V o de 5 a 0 V. Eso significa que **durante un muy breve periodo de tiempo, el valor en la entrada puede oscilar**.



Para hacer uso de dicha resistencia interna, debemos hacer una pequeña **modificación** a la hora de **configurar el pin de entrada**:

```
pinMode(4, INPUT_PULLUP);
```

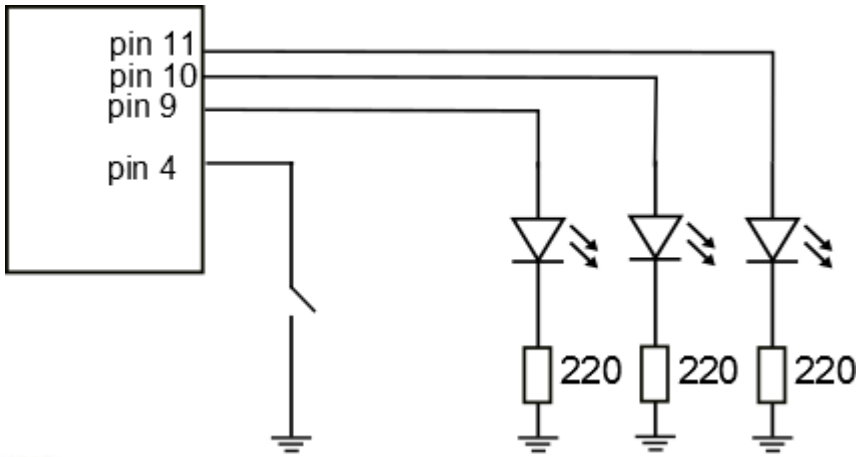


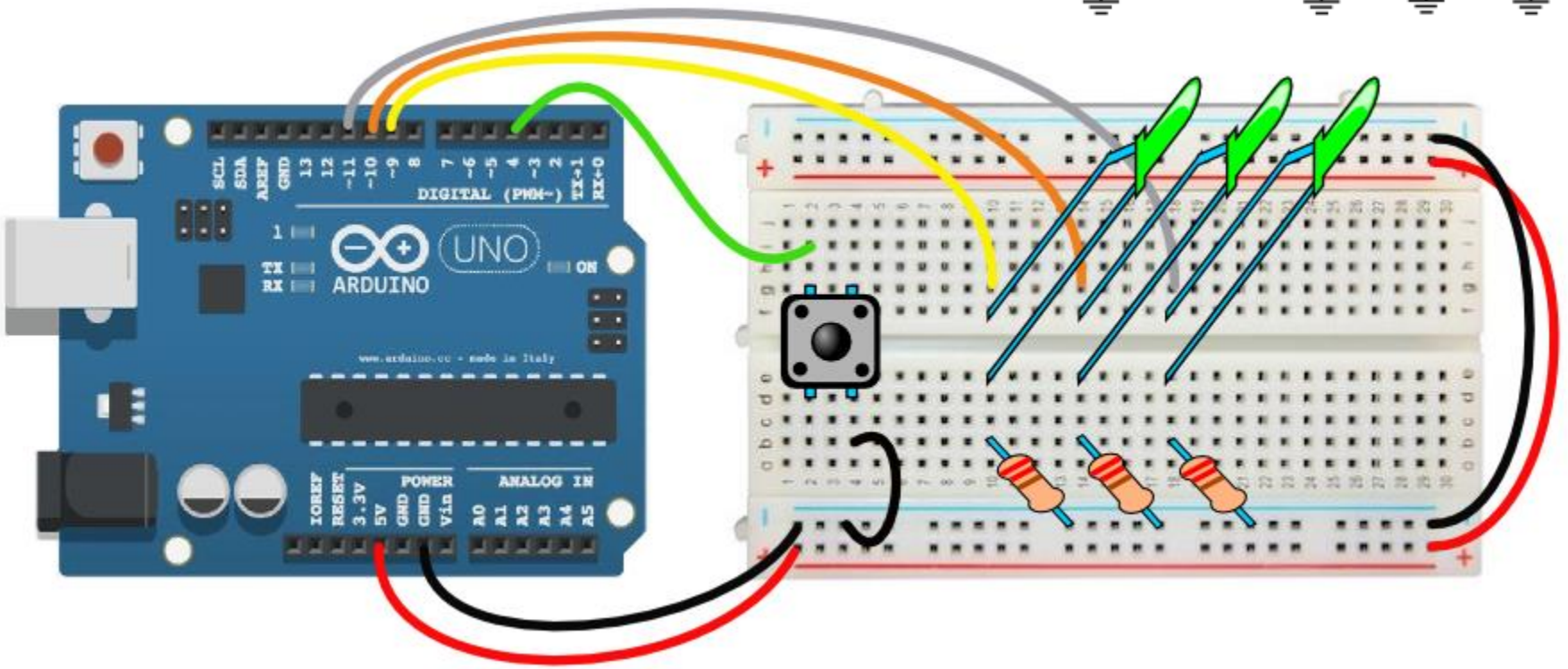
Apretado (ON) → `digitalRead(4)` = **LOW**

Sin apretar (OFF) → `digitalRead(4)` = **HIGH**

Ejemplo: Disparando ráfagas

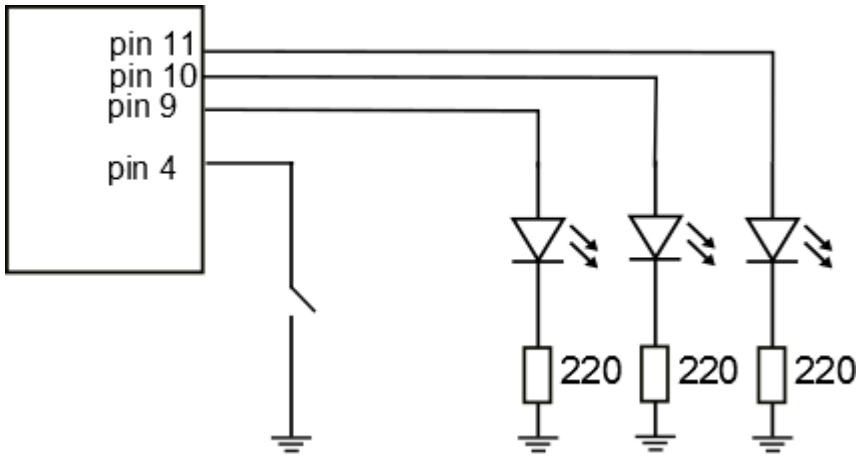
Cuando apretemos un pulsador, se desplazará una luz a través de 3 LEDs:





Ejemplo: Disparando ráfagas

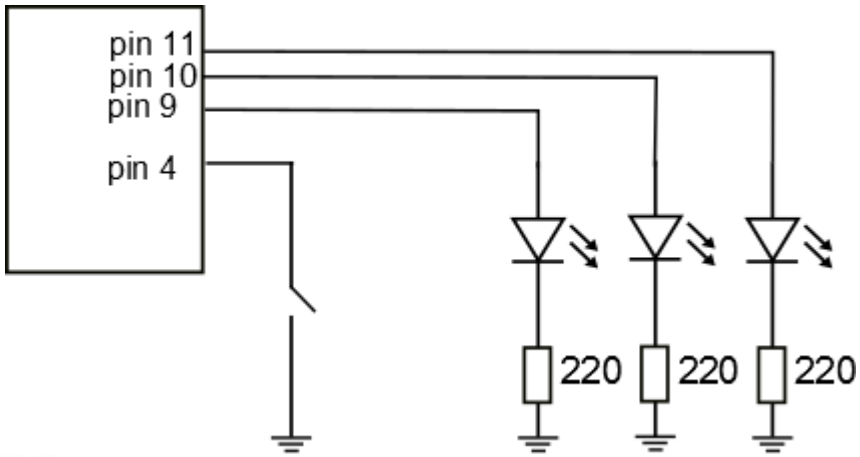
Cuando apretemos un pulsador, se desplazará una luz a través de 3 LEDs:

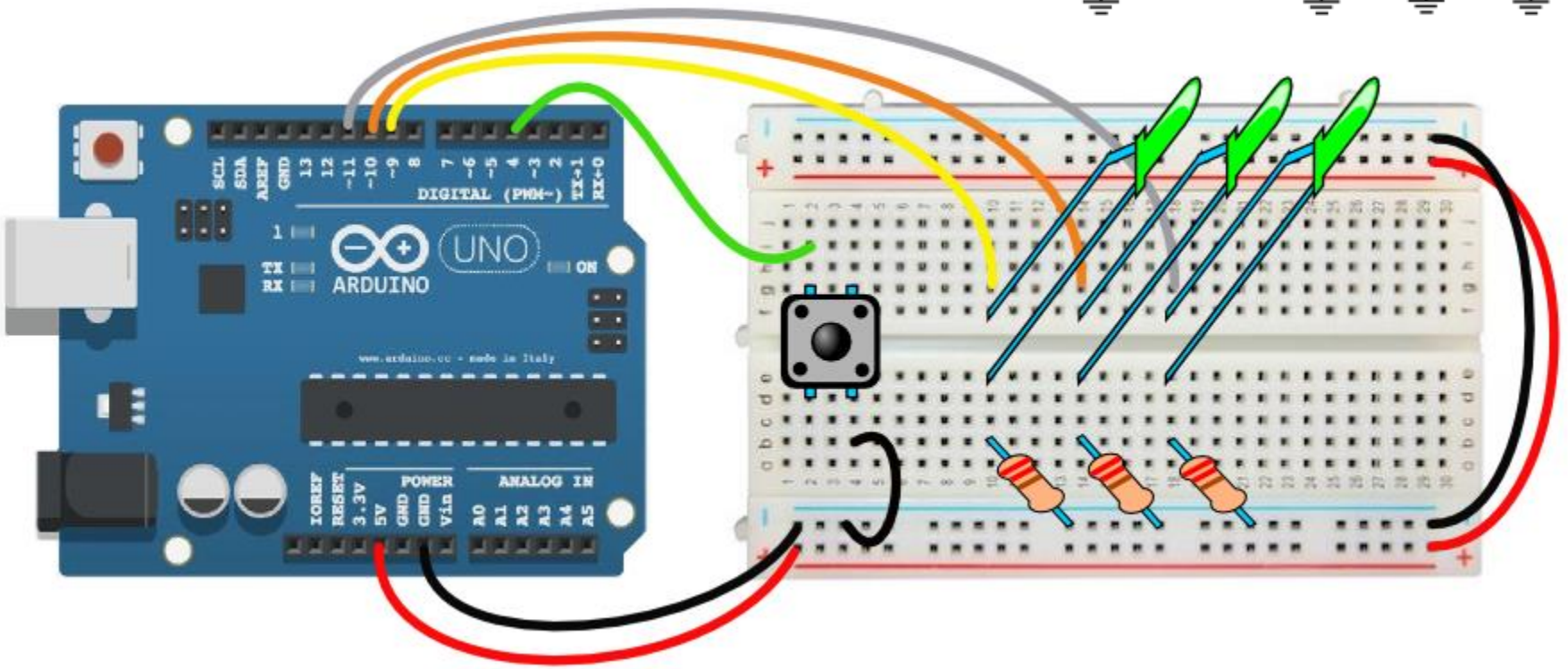



```
void setup() {
  for(int i=9; i<12; i++) pinMode(i, OUTPUT);
  pinMode(4, INPUT_PULLUP);
}

void loop() {
  if(!digitalRead(4)) {        //apretado da una lectura de LOW o false
    for(int i=9; i<12; i++) {
      digitalWrite(i, HIGH);
      digitalWrite(i-1, LOW);
      delay(100);
    }
    digitalWrite(11, LOW);
  }
}
```

Ejemplo: Contando 1, 2 y 3

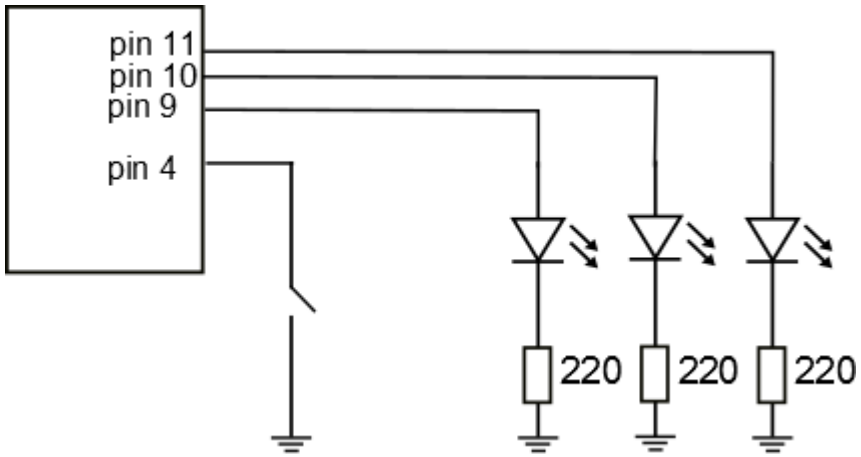
Cuando apretemos el pulsador, se irán encendiendo alternativamente cada uno de los LEDs:





Ejemplo: Contando 1, 2 y 3

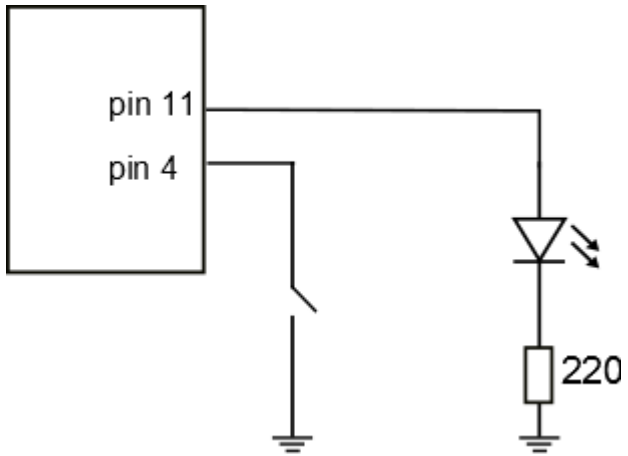
Cuando apretemos el pulsador, se irán encendiendo alternativamente cada uno de los LEDs:

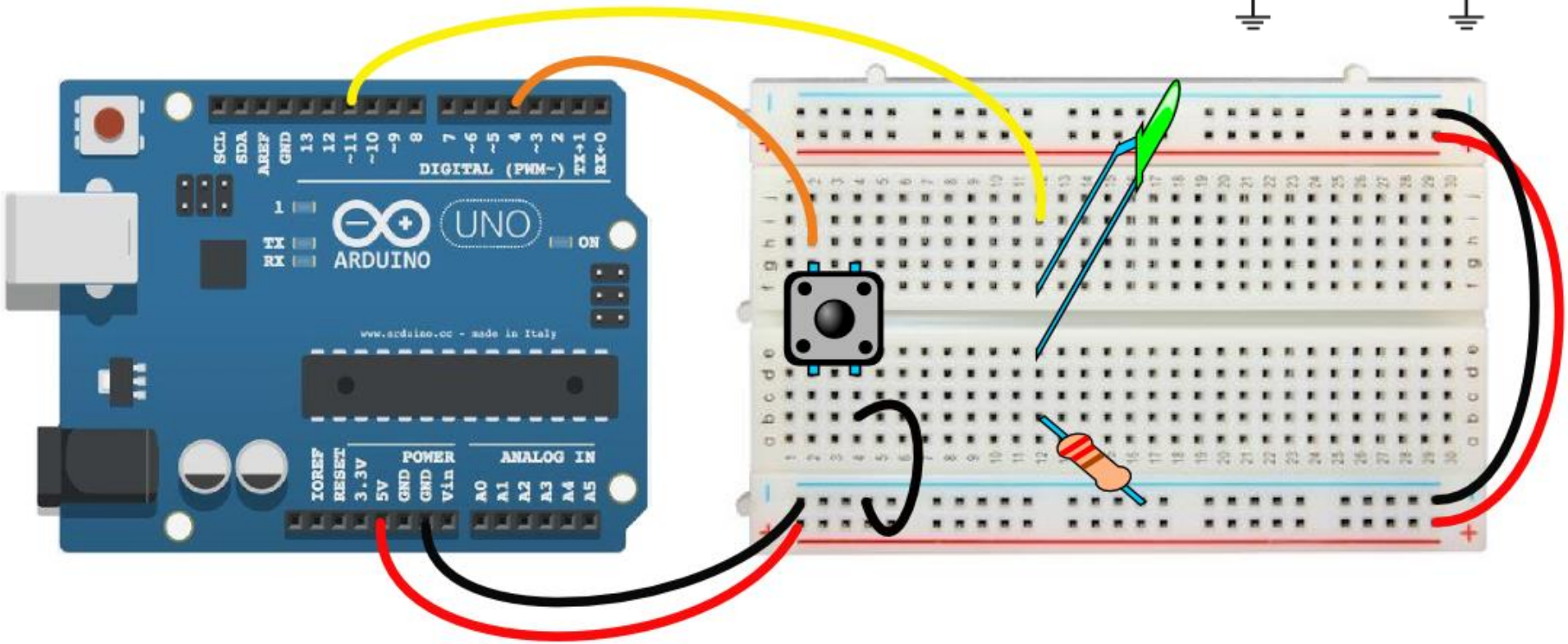



```
int x = 0;          //contaré las pulsaciones

void setup() {
  for(int i=9; i<12; i++) pinMode(i, OUTPUT);
  pinMode(4, INPUT_PULLUP);
}

void loop() {
  if(!digitalRead(4)) x++;              //cuando aprieto, el contador sube 1
  if(x > 0) digitalWrite(9, HIGH);      //enciendo el primer LED a partir de x>0
  if(x > 1) digitalWrite(10, HIGH);     //enciendo el segundo LED a partir de x>1
  if(x > 2) digitalWrite(11, HIGH);     //enciendo el tercer LED a partir de x>2
  if(x == 4) {                          //si x es 4 …
    for(int i=9; i<12; i++) digitalWrite(i, LOW);       //apago todos los LEDs …
    x = 0;                                              //y reinicio el contador
  }
  delay(200);       //espero un pequeño tiempo para separar las pulsaciones
}
```

Ejemplo: Pulsador como interruptor

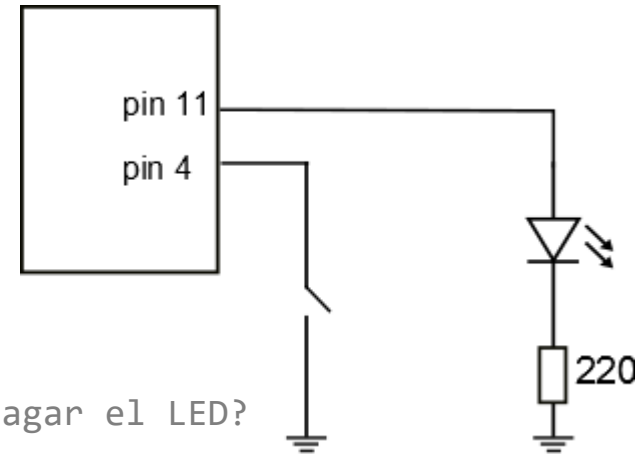
Cuando apretemos el pulsador, el LED pasará del estado OFF al ON y viceversa:





Ejemplo: Pulsador como interruptor

Cuando apretemos el pulsador, el LED pasará del estado OFF al ON y viceversa:




```
int val = 0;           //lectura del pulsador: ON u OFF
int valAnterior = 0;   //lectura justamente anterior
int estado = 0;        //¿qué toca ahora, encender o apagar el LED?

void setup() {
  pinMode(11, OUTPUT);
  pinMode(4, INPUT_PULLUP);
}

void loop() {
  val = digitalRead(4);
  if(val == LOW && valAnterior == HIGH) {   //se cumple cuando aprieto y suelto
    estado = 1 - estado;    //si he pulsado y soltado, que cambie el estado del LED
    delay(20);              //elimino los efectos de bouncing
  }
  valAnterior = val;        //el valor del pulsador pasa a ser valor pasado
  if(estado == 1) digitalWrite(11, HIGH);  //el LED encenderá cuando el estado sea 1
  else digitalWrite(11, LOW);              //en caso contrario, lo apaga
}
```

Comunicación con el PC `Serial.begin(9600);`

En muchas ocasiones es muy útil poder **visualizar** a través del ordenador los **valores de lectura en los pines de entrada y de salida** de Arduino. Asimismo, también puede ser necesario **mandar información** a Arduino desde el **teclado** del PC, o viceversa: **mandar información** desde Arduino hacia el ordenador para, por ejemplo, intervenir en un sketch de **Processing** . Veamos cómo poner en contacto ambos aparatos:

Comunicación con el PC `Serial.begin(9600);`

Dentro del `setup` debemos avisar que vamos a establecer dicha comunicación, utilizando la siguiente orden:

```
Serial.begin(9600);          /*hay que especificar los baudios (bits/s), que
                             por costumbre van a ser 9600 */
```

Luego, dentro del `loop` podemos utilizar las siguientes funciones para transmitir datos desde Arduino hacia el PC:

```
Serial.print(val);           //imprime el valor de la variable val

Serial.println(val);         //imprime el valor de val e inserta una línea nueva

Serial.print("hola amigos");          //imprime el texto "hola, amigos"

Serial.print('\t');                   //imprime una tabulación
```

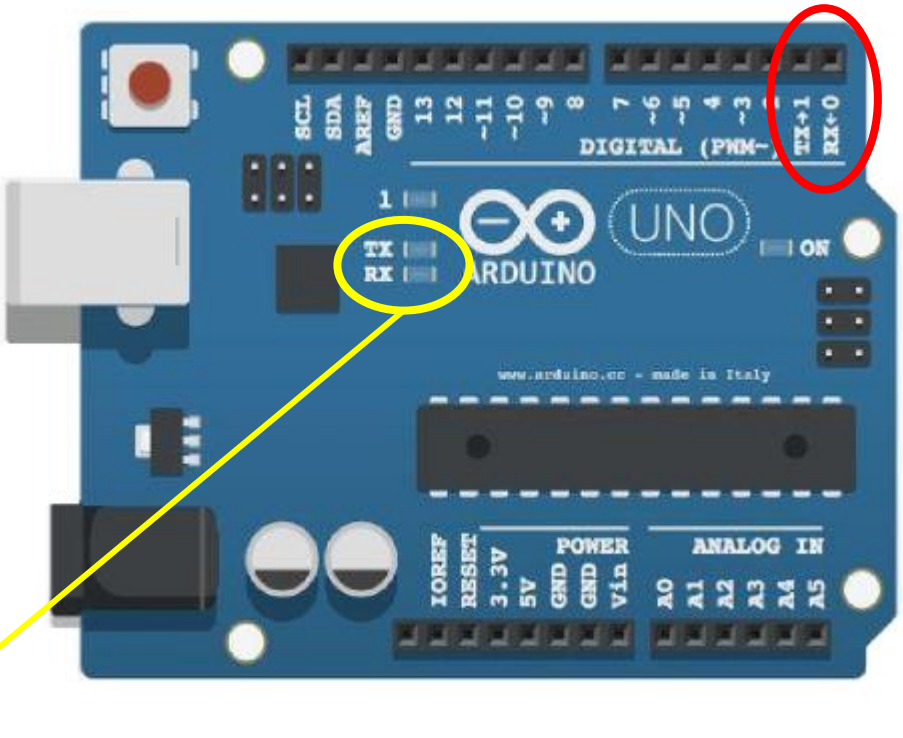
Comunicación con el PC_2

Las funciones para que Arduino reciba datos a través del teclado desde el PC son:

```
x = Serial.available();      /*asigna a x el número de bytes disponibles en el
                             puerto serie que aún no han sido leídos. Después de
                             haberlos leídos todos, la función Serial.available()
                             devuelve el valor 0 (false), hasta que no lleguen
                             nuevos datos al puerto serie */

val = Serial.read();         /*asigna a val el valor disponible en el puerto serie,
                             que lo introducimos desde el teclado del ordenador en
                             la zona de textos del Serial Monitor */

Serial.flush();              /*porque los datos pueden llegar al puerto serie a más
                             velocidad que la del proceso del programa, Arduino
                             puede guardar todos los datos de entrada en un buffer.
                             Si es necesario limpiar el buffer para llenarlo de
                             datos nuevos, debemos usar la función flush(); */
```

Comunicación con el PC_3

Cuando Arduino establece comunicación con el PC necesita utilizar los pines 0 y 1 (**RX** y **TX**) para recibir y/o transmitir datos, por lo tanto no debemos utilizarlos como entradas o salidas para nuestro circuito.



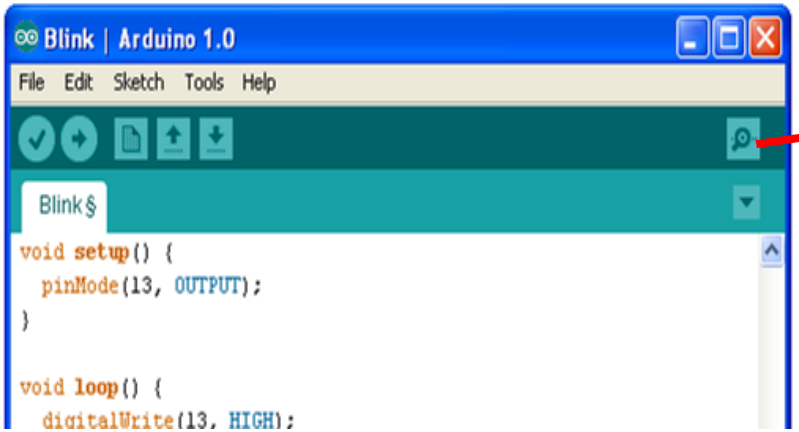
Estos LEDs indican si hay una transmisión de datos:

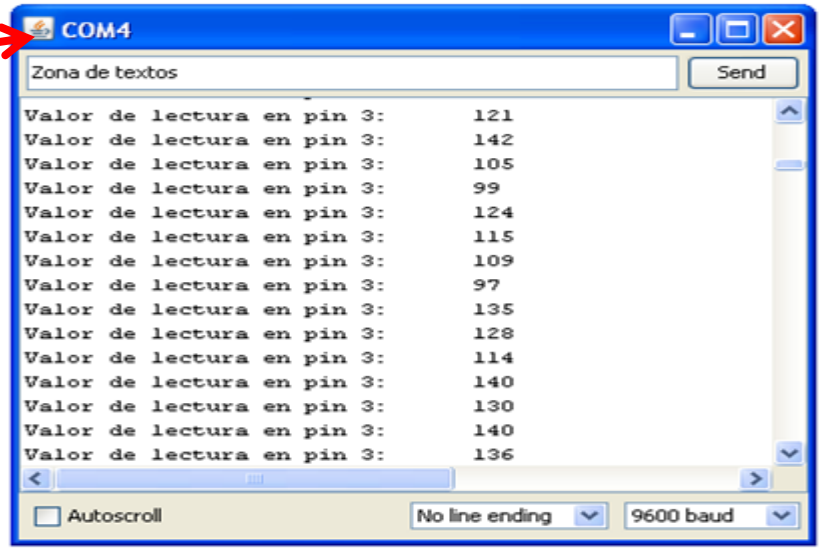
TX → de Arduino al PC

RX → del PC a Arduino

Comunicación con el PC_4

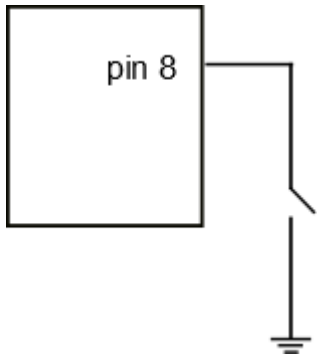
Para poder visualizar en pantalla los datos que Arduino va imprimiendo, o para poder introducir datos a Arduino a través del teclado, debemos abr ir la ventana de impresión en el siguiente botón (**Monitor Serie**):





Ejemplo: Mirando el estado de un pulsador

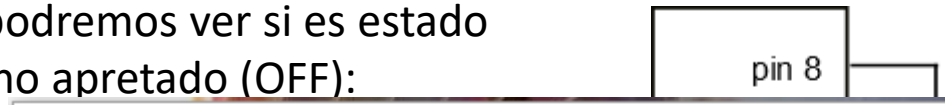
En la ventana de Serial Monitor podremos ver si es estado del pulsador es apretado (ON) o no apretado (OFF):

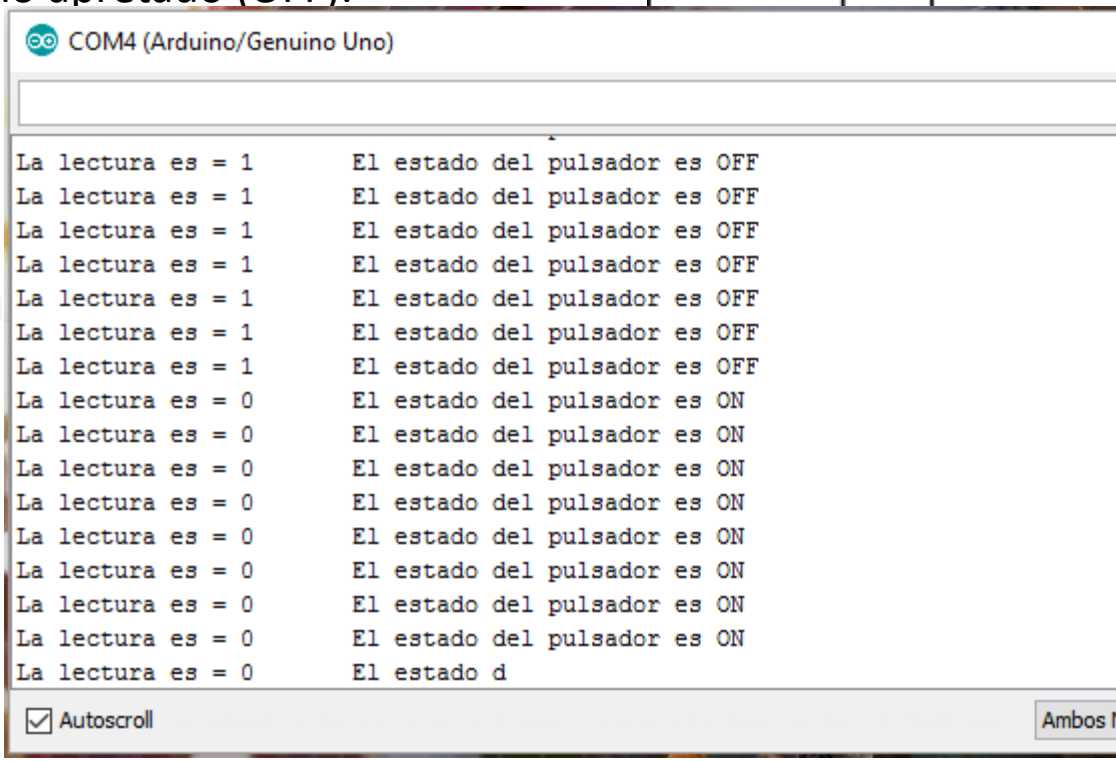



```
int val;           //será el valor de la lectura en el pin 8

void setup() {
  pinMode(8, INPUT_PULLUP);          //lo configuraré con resistencia pull up interna
  Serial.begin(9600);                //establezco comunicación por puerto serie
}

void loop() {
  val = digitalRead(8);               //asigno a val el valor de la lectura digital
  Serial.print("La lectura es = ");
  Serial.print(val);
  Serial.print('\t');                 //es una tabulación
  if(val) Serial.println("El estado del pulsador es OFF");
          //si val == true, es porque no estamos pulsando el pulsador
  else Serial.println("El estado del pulsador es ON");
          //el caso contrario (val == false) significaría que sí lo estamos apretando
}
```

Ejemplo: Mirando el estado de un pulsador

En la ventana de Serial Monitor podremos ver si es estado del pulsador es apretado (ON) o no apretado (OFF):

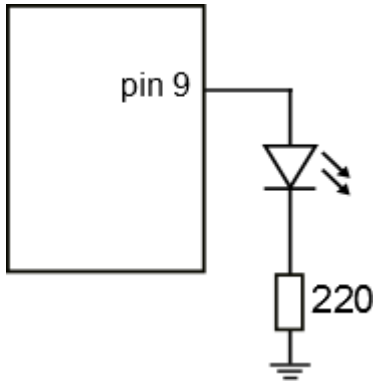





```
int val;           //será el valor d

void setup() {
  pinMode(8, INPUT_PULLUP);
  Serial.begin(9600);
}

void loop() {
  val = digitalRead(8);
  Serial.print("La lectura es = ");
  Serial.print(val);
  Serial.print('\t');                    //es una tabulación
  if(val) Serial.println("El estado del pulsador es OFF");
          //si val == true, es porque no estamos pulsando el pulsador
  else Serial.println("El estado del pulsador es ON");
          //el caso contrario (val == false) significaría que sí lo estamos apretando
}
```

Ejemplo: Control de un LED con el teclado

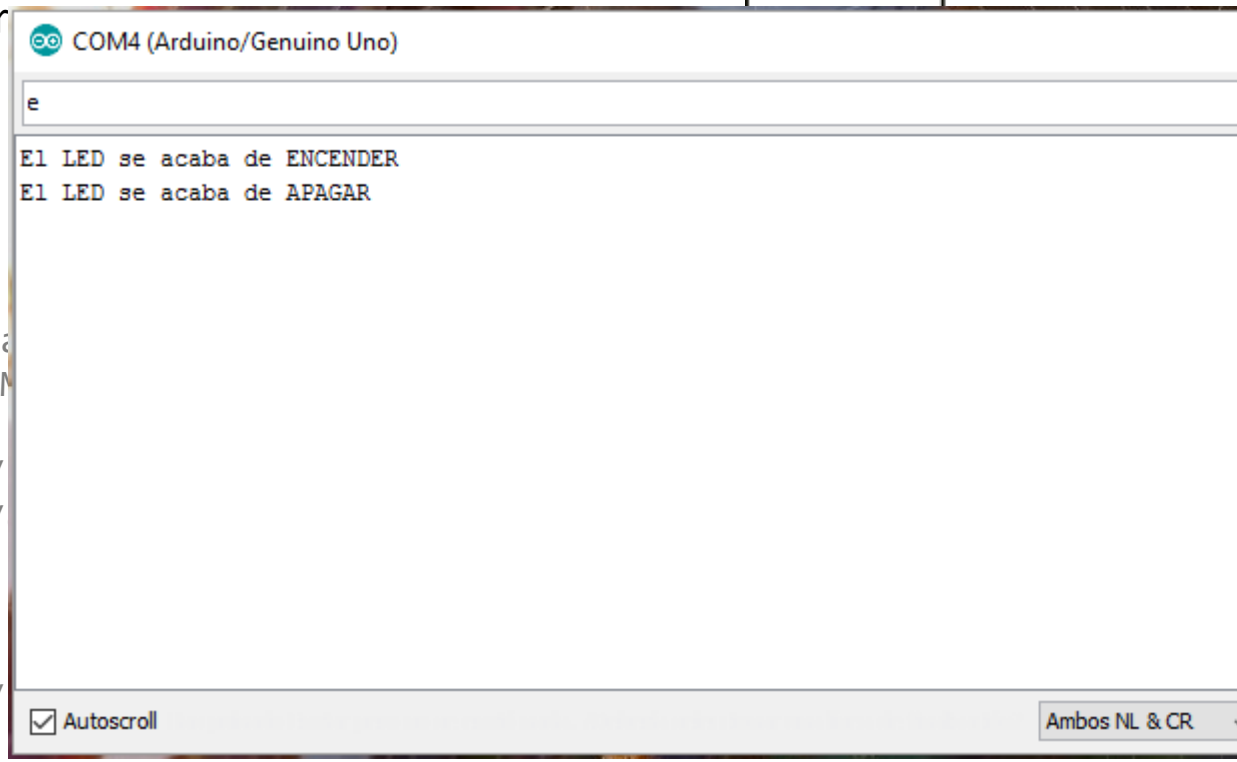
Encenderemos y apagaremos el LED a través de las teclas "e" para encender y "a" para apagar:




```
int val;           /*será el valor del último carácter introducido en la zona de texto
                   del Serial Monitor */
void setup() {
  pinMode(9, OUTPUT);       //conectaré el LED al pin 9
  Serial.begin(9600);       //establezco comunicación por puerto serie
}

void loop() {
  val = Serial.read();      //asigno a val el último carácter introducido
  if(val == 'e') {
    digitalWrite(7, HIGH);  //enciendo el LED
    Serial.println("El LED se acaba de ENCENDER");
  }
  if(val == 'a') {
    digitalWrite(7, LOW);   //apago el LED
    Serial.println("El LED se acaba de APAGAR");
  }
```

Ejemplo: Control de un LED con el teclado

Encenderemos y apagaremos el LED a través de las teclas "e" para encender y "a" par




```
int val;           /*será el va
                   del Serial M
void setup() {
  pinMode(9, OUTPUT);       //
  Serial.begin(9600);       //
}

void loop() {
  val = Serial.read();      //
  if(val == 'e') {
    digitalWrite(7, HIGH);  //enciendo el LED
    Serial.println("El LED se acaba de ENCENDER");
  }
  if(val == 'a') {
    digitalWrite(7, LOW);   //apago el LED
    Serial.println("El LED se acaba de APAGAR");
  }
```

Entradas Analógicas `analogRead(A1);`

Recordemos que las entradas analógicas en Arduino no hay que configurarlas en el setup. La lectura de una señal analógica de entrada podrá tener valores comprendidos entre **0** y **1023**, correspondientes a los valores intermedios de un rango de 0 a 5 V.

```
val = analogRead(A3);        /*asignará a la variable val el valor de la
                             lectura digital del pin A3, que será un número
                             comprendido entre 0 y 1023 */
```

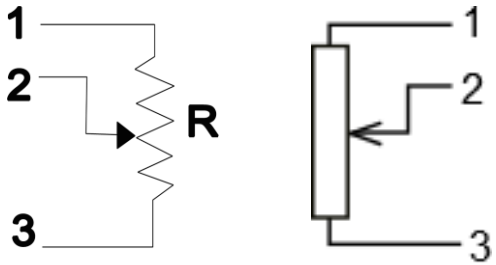
Evidentemente, para emplear una entrada analógica necesitamos un sensor analógico, es decir, que sus valores eléctricos varíen en un rango significativo, no limitándose a dos posibles valores. Sensores analógicos pueden ser:

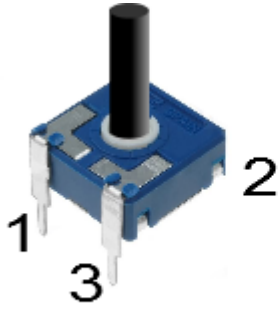
- Potenciómetro
- LDR (fotorresistencia)
- NTC (termoresistencia)
- sensor de sonido (piezoresistencia)
- sensor de ultrasonido
- etc…

Sensores Analógicos: El Potenciómetro

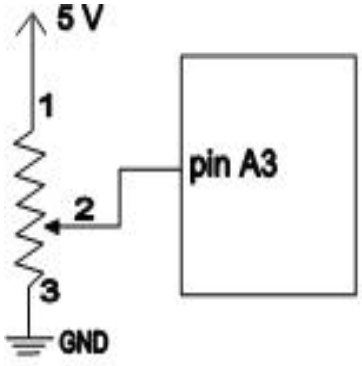
El potenciómetro es un sensor de entrada muy usado. Consta de una **resistencia variable**, que **dependerá de la posición de giro en que se encuentre su consola** (o mango). Es muy importante conectar el potenciómetro de manera correcta para evitar que se produzca algún cortocircuito (uniendo 5 V y 0 V sin ninguna resistencia de por medio).

Un potenciómetro posee tres patas. Su símbolo es el siguiente:





Podemos conseguir una entrada analógica de la siguiente manera:



Girado a tope en sentido antihorario → `analogRead(A3) = 1023`

…

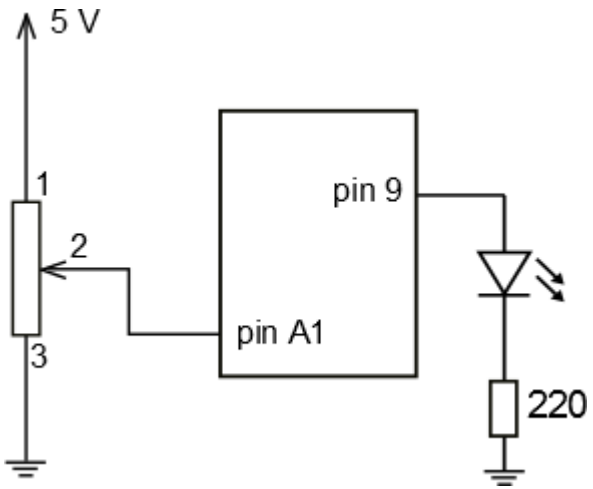
Posición centrada → `analogRead(A3) = 512`

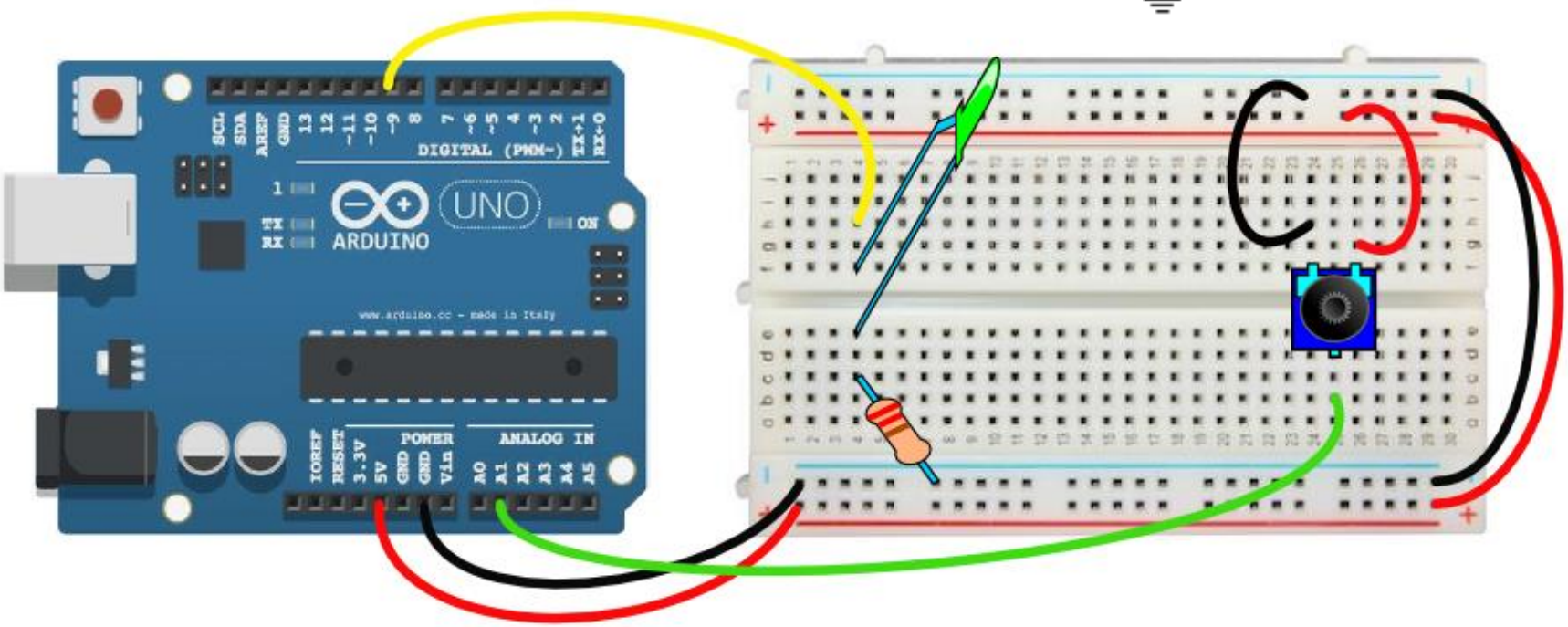
…

Girado a tope en sentido horario → `analogRead(A3) = 0`

Ejemplo: Control de un LED con potenciómetro

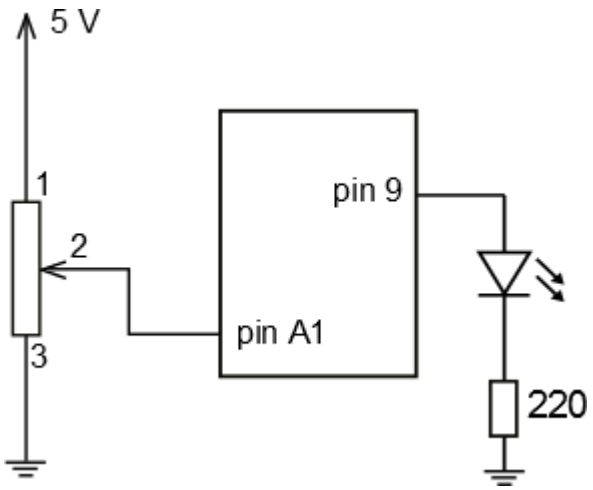
Encenderemos un LED con más o menos intensidad, según giremos más o menos un potenciómetro. Además, visualizaremos en el Serial Monitor en valor de la entrada.





Ejemplo: Control de un LED con potenciómetro

Encenderemos un LED con más o menos intensidad, según giremos más o menos un potenciómetro. Además, visualizaremos en el Serial Monitor en valor de la entrada.

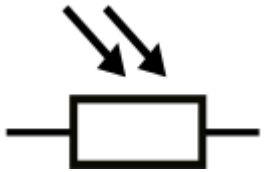



```
int val;
int brillo;

void setup() {
  pinMode(9, OUTPUT);
  Serial.begin(9600);                  //no hay que configurar el pin A1
}

void loop() {
  val = analogRead(A1);                //es el valor de lectura de la señal de entrada
  Serial.print("Lectura entrada analógica = ");
  Serial.print(val);                   //imprimirá dicha lectura
  Serial.print('\t');
  brillo = val / 4;                    //escalo los 1024 valores a 256 (divido /4)
  Serial.print("Valor del brillo del LED = ");
  Serial.println(brillo);              //imprimirá el valor del brillo
  analogWrite(9, brillo);              //encenderá el LED con una intensidad = brillo
}
```

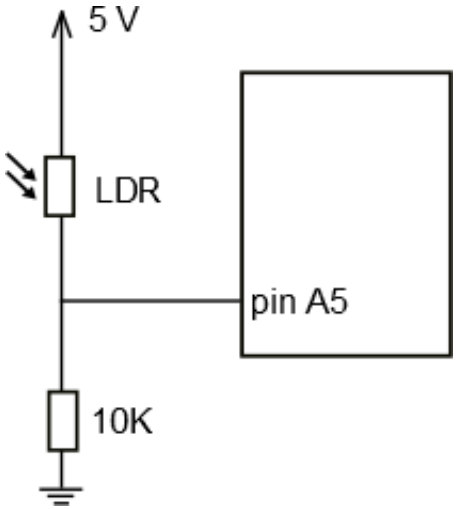
Sensores Analógicos: La LDR

Una LDR es una resistencia variable con la luz: cuanto mayor luz reciba, menos ohmios tendrá. Podemos utilizar una LDR para utilizarlo como sensor de luz.

Su símbolo es el siguiente:



Podemos conseguir una entrada analógica de la siguiente manera:



Mucha luz → `analogRead(A5) = valores altos (~ 500)`

…

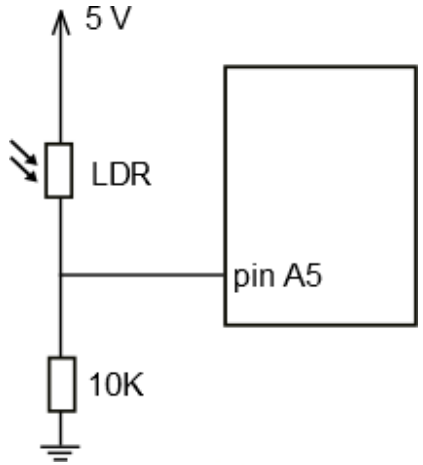
Luz moderada → `analogRead(A5) = valores medios (~ 300)`

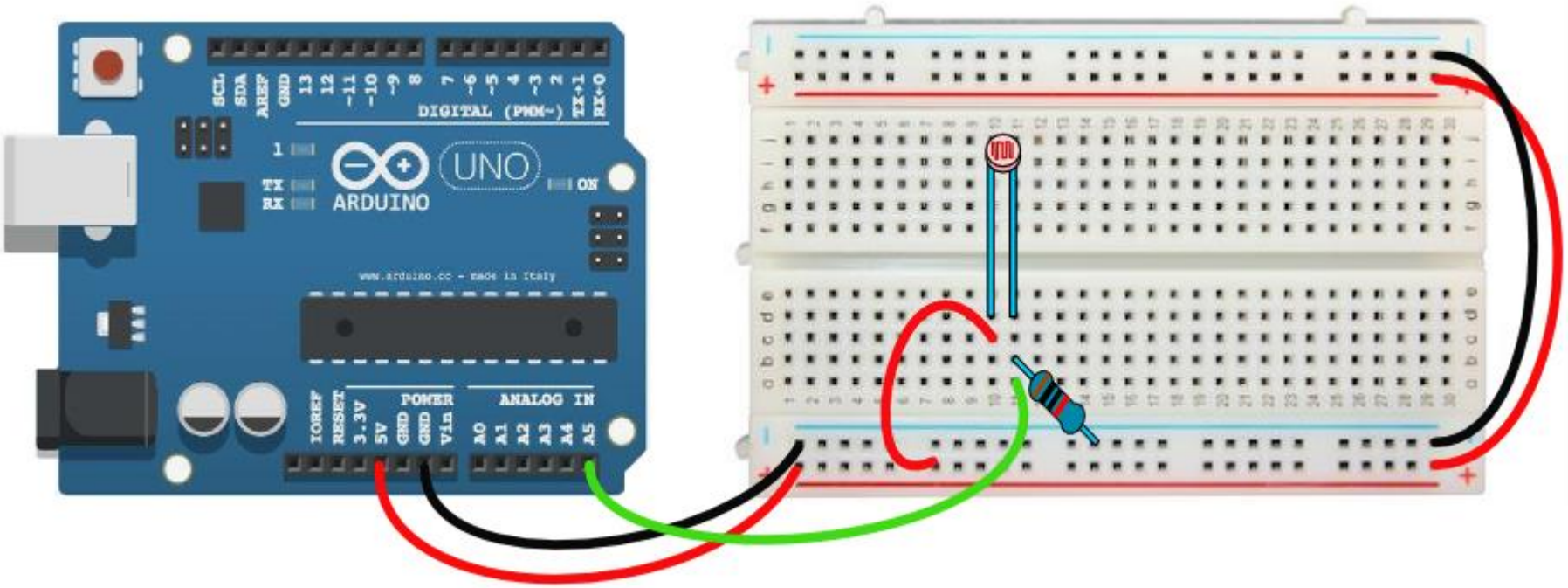
…

Oscuridad → `analogRead(A5) = valores bajos (~ 100)`

Ejemplo: Sensor de luz

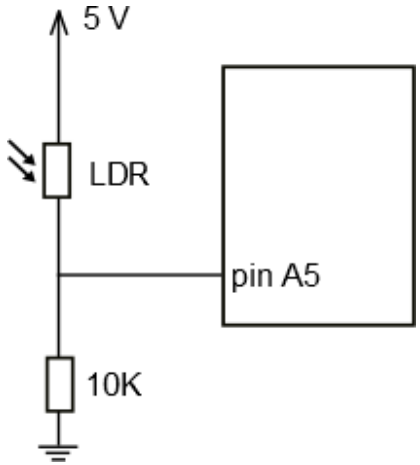
Utilizaremos una LDR para montar un sensor de luz. A través del Monitor Serie veremos qué lecturas llegan a Arduino con diferentes estados lumínicos.





Ejemplo: Sensor de luz

Utilizaremos una LDR para montar un sensor de luz. A través del Monitor Serie veremos qué lecturas llegan a Arduino con diferentes estados lumínicos.




```
int val;

void setup() {
  Serial.begin(9600);                 //no hay que configurar el pin A1
}

void loop() {
  val = analogRead(A5);               //es el valor de lectura de la señal de entrada
  Serial.print("Lectura entrada analógica = ");
  Serial.println(val);                //imprimirá dicha lectura
}
```

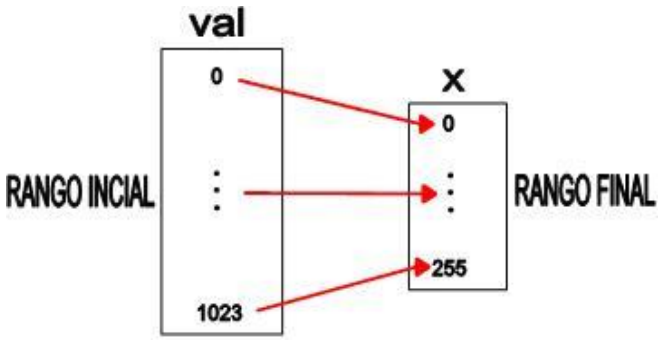
Mapear datos `map(val, 0, 1023, 0, 255);`

La función `map` es muy útil para utilizar una lectura de un pin de entrada y emplear ese dato para utilizarlo en un pin de salida analógico (es algo parecido a realizar una regla de tres).

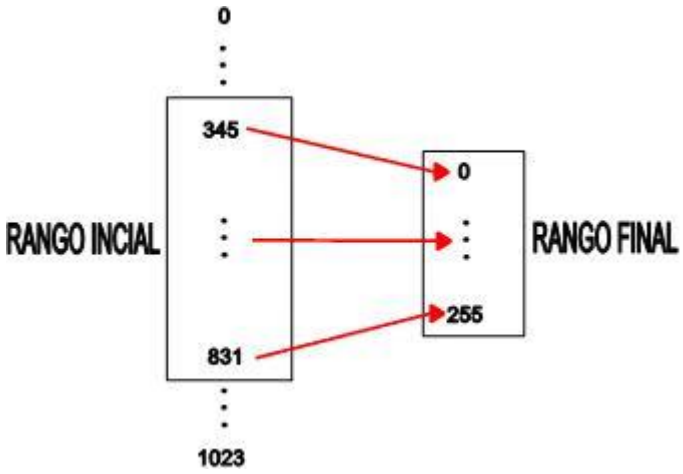
La función `map` toma un valor `val` dentro de rango de valores iniciales y lo mapea (lo escala) a un nuevo rango de valores finales:

```
x = map(val, minIncial, maxInicial, minFinal, maxFinal);
```

```
x = map(val, 0, 1023, 0, 255);
```




```
x = map(val, 345, 831, 0, 255);
```



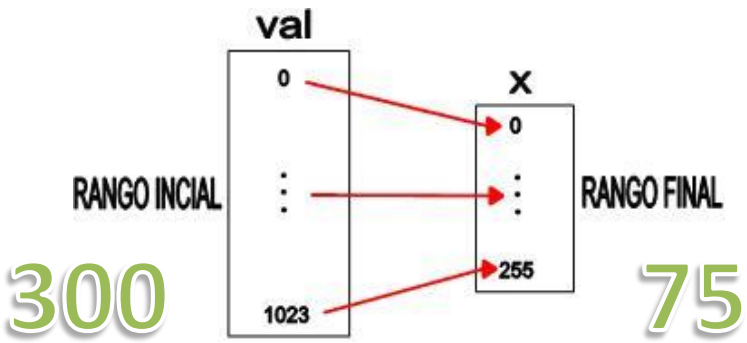
Mapear datos `map(val, 0, 1023, 0, 255);`

La función `map` es muy útil para utilizar una lectura de un pin de entrada y emplear ese dato para utilizarlo en un pin de salida analógico (es algo parecido a realizar una regla de tres).

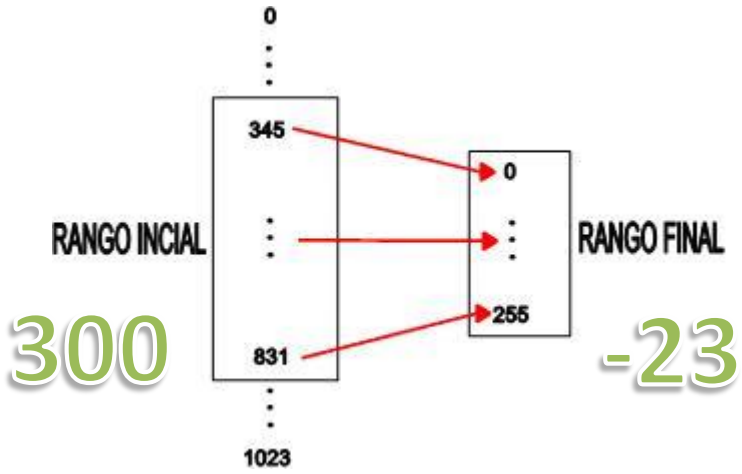
La función `map` toma un valor `val` dentro de rango de valores iniciales y lo mapea (lo escala) a un nuevo rango de valores finales:

```
x = map(val, minIncial, maxInicial, minFinal, maxFinal);
```

```
x = map(val, 0, 1023, 0, 255);
```




```
x = map(val, 345, 831, 0, 255);
```



Este ejemplo es equivalente a:

```
x = val / 4;
```

Así podré, por ejemplo, calibrar nuestros sensores de entrada (como la LDR).

Restringir datos `constrain(val, 0, 255);`

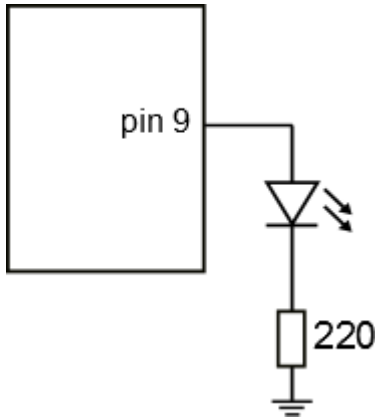
También puede ser muy útil, a la hora de evitar salirnos de los rangos habituales de uso, la función `constrain`:

```
x = constrain(val, 50, 200);          /*agina a la variable x el valor de la
                                      variable val, pero dicho valor será
                                      como poco de 50, y como mucho de 200 */
```

Restringir datos_2

Piensa, ¿qué crees que va a ocurrir en los siguientes programas?

```
int x = 0;

void setup() {
  pinMode(9, OUTPUT);
}

void loop() {
  analogWrite(9, x);
  x++;
  delay(50);
}
```

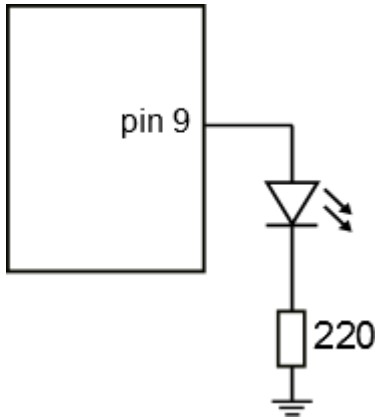



```
int x = 0;

void setup() {
  pinMode(9, OUTPUT);
}

void loop() {
  x = constrain(x, 0, 255);
  analogWrite(9, x);
  x++;
  delay(50);
}
```

Restringir datos_2

Piensa, ¿qué crees que va a ocurrir en los siguientes programas?

```
int x = 0;

void setup() {
  pinMode(9, OUTPUT);
}

void loop() {
  analogWrite(9, x);
  x++;
  delay(50);
}
```



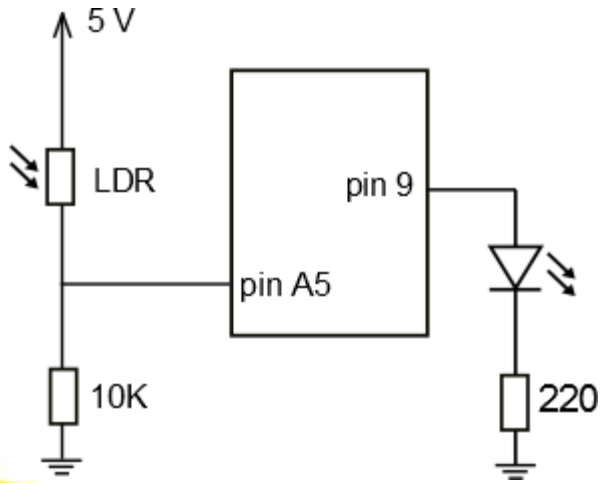

```
int x = 0;

void setup() {
  pinMode(9, OUTPUT);
}

void loop() {
  x = constrain(x, 0, 255);
  analogWrite(9, x);
  x++;
  delay(50);
}
```

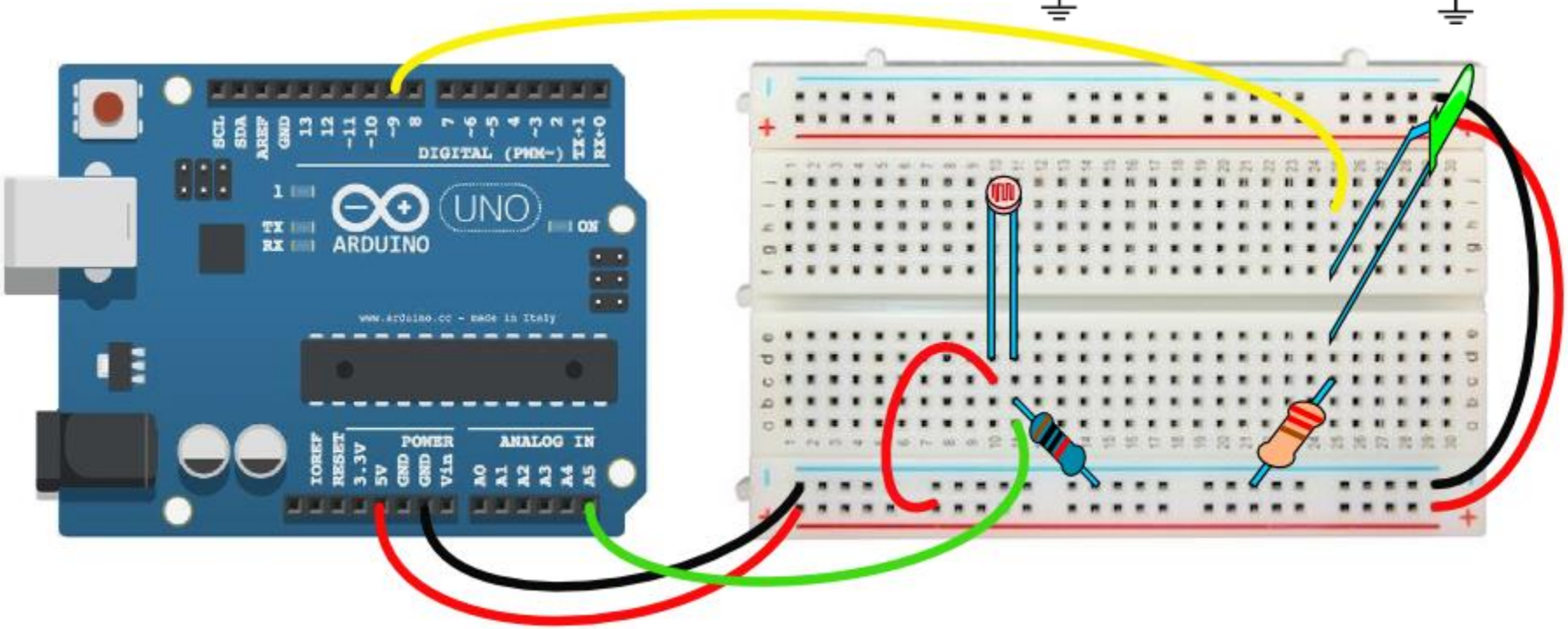
En el programa de la izquierda, el valor de la `x` "**rebosará**" del valor máximo admitido para la función `analogWrite`, que es `255`, con lo cual a partir de ese valor lo interpretará como volver a comenzar un nuevo rango de valores, es decir:
`x=256→0; x=257→1; x=258→2; …; x=510→254; x=511→255; x=512→0; x=513→1; …`

Y en el programa de la derecha, el valor de `x` estará siempre limitado a `255`, con lo que la salida analógica, una vez alcanzado ese valor, no cambiará, pese a la orden `x++;`

Ejemplo: Luz automática nocturna

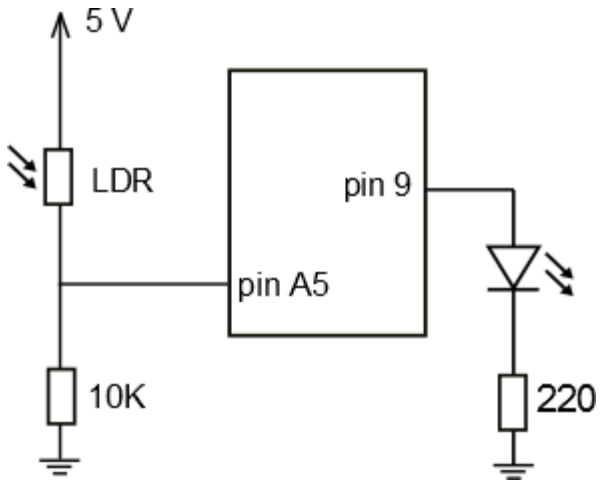
Haremos que se encienda un LED con una intensidad que sea proporcional al grado de oscuridad (cuanto más oscuro, más iluminará el LED).





Ejemplo: Luz automática nocturna

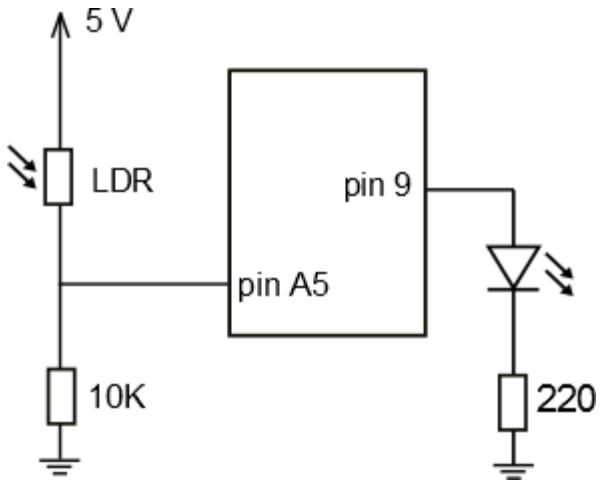
Haremos que se encienda un LED con una intensidad que sea proporcional al grado de oscuridad (cuanto más oscuro, más iluminará el LED).




```
int val;
int brillo;

void setup() {
  pinMode(9, OUTPUT);
  Serial.begin(9600);                  //no hay que configurar el pin A5
}

void loop() {
  val = analogRead(A5);                //es el valor de lectura de la señal de entrada
  Serial.print("Lectura entrada analógica = ");
  Serial.print(val);                   //imprimirá dicha lectura
  Serial.print('\t');
  brillo = map(val, 0, 1023, 0, 255);          //este mapeado será provisional!

  Serial.print("Valor del brillo del LED = ");
  Serial.println(brillo);              //imprimirá el valor del brillo
  analogWrite(9, brillo);              //encenderá el LED con una intensidad = brillo
}
```

Ejemplo: Luz automática nocturna

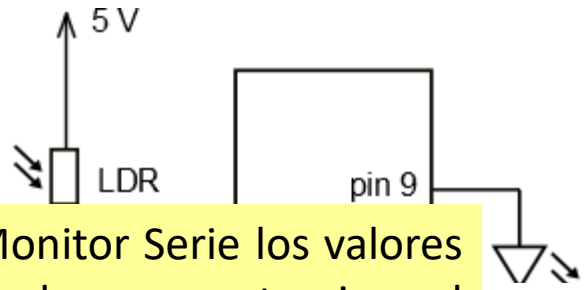
Haremos que se encienda un LED con una intensidad que sea proporcional al grado de oscuridad (cuanto más oscuro, más iluminará el LED).




```
int val;
int brillo;

void setup() {
  pinMode(9, OUTPUT);
  Serial.begin(9600);                  //no hay que configurar el pin A5
}

void loop() {
  val = analogRead(A5);                //es el valor de lectura de la señal de entrada
  Serial.print("Lectura entrada analógica = ");
  Serial.print(val);                   //imprimirá dicha lectura
  Serial.print('\t');
  brillo = map(val, 0, 1023, 0, 255);          //este mapeado será provisional!

  Serial.print("Valor del brillo del LED = ");
  Serial.println(brillo);              //imprimirá el valor del brillo
  analogWrite(9, brillo);              //encenderá el LED con una intensidad = brillo
}
```

Ejemplo: Luz automática nocturna

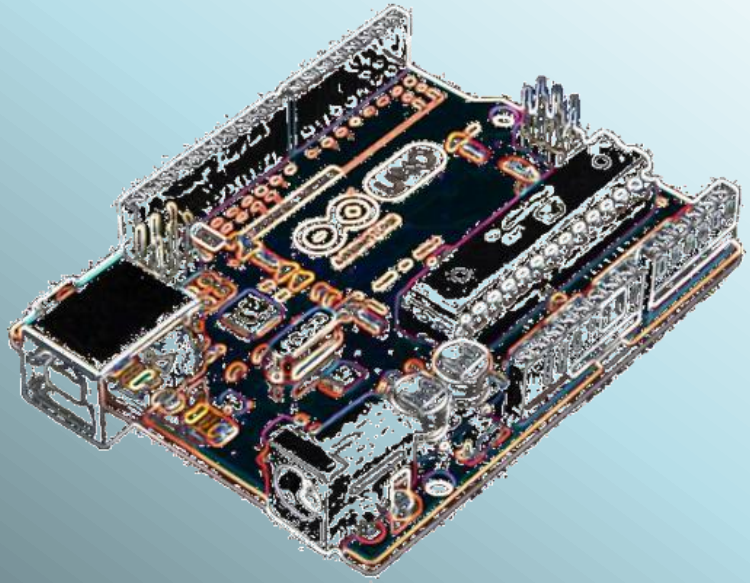
Haremos que se encienda un LED con una intensidad que sea proporcional al grado de oscuridad (cuanto más oscuro, más iluminará el LED).



Una vez que hayamos visto en el Monitor Serie los valores de lectura mínimos y máximos, podremos customizar el rango inicial para la función `map` (además, se observa que para valores bajos, el LED enciende más, con lo cual debemos intercambiar el orden del valor mínimo y máximo). Si por ejemplo valMin = 100, y valMax = 800, debemos poner:

```
int val;
int brillo;

void setup() {
  pinMode(9, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  val = analogRead(A5);              //es el valor de lectura de la señal de entrada
  Serial.print("Lectura entrada analógica = ");
  Serial.print(val);                 //imprimirá dicha lectura
  Serial.print('\t');
  brillo = map(val, 800, 100, 0, 255);        //este mapeado será el correcto
  brillo = constrain(brillo, 0, 255);         //así evitaré que se salga de (0,255)
  Serial.print("Valor del brillo del LED = ");
  Serial.println(brillo);            //imprimirá el valor del brillo
  analogWrite(9, brillo);            //encenderá el LED con una intensidad = brillo
}
```

FIN del Capítulo II



Daniel Gallardo García
Profesor de Tecnología
Jerez de la Frontera
danielprofedetecno@gmail.com