

Diseño con Arduino – Ofimega – Guía rápida de introducción

Arduino es una plataforma de hardware libre basada en una sencilla placa de entradas y salidas simples y un entorno de desarrollo que implementa el lenguaje de programación *Processing/Wiring* similar al C.

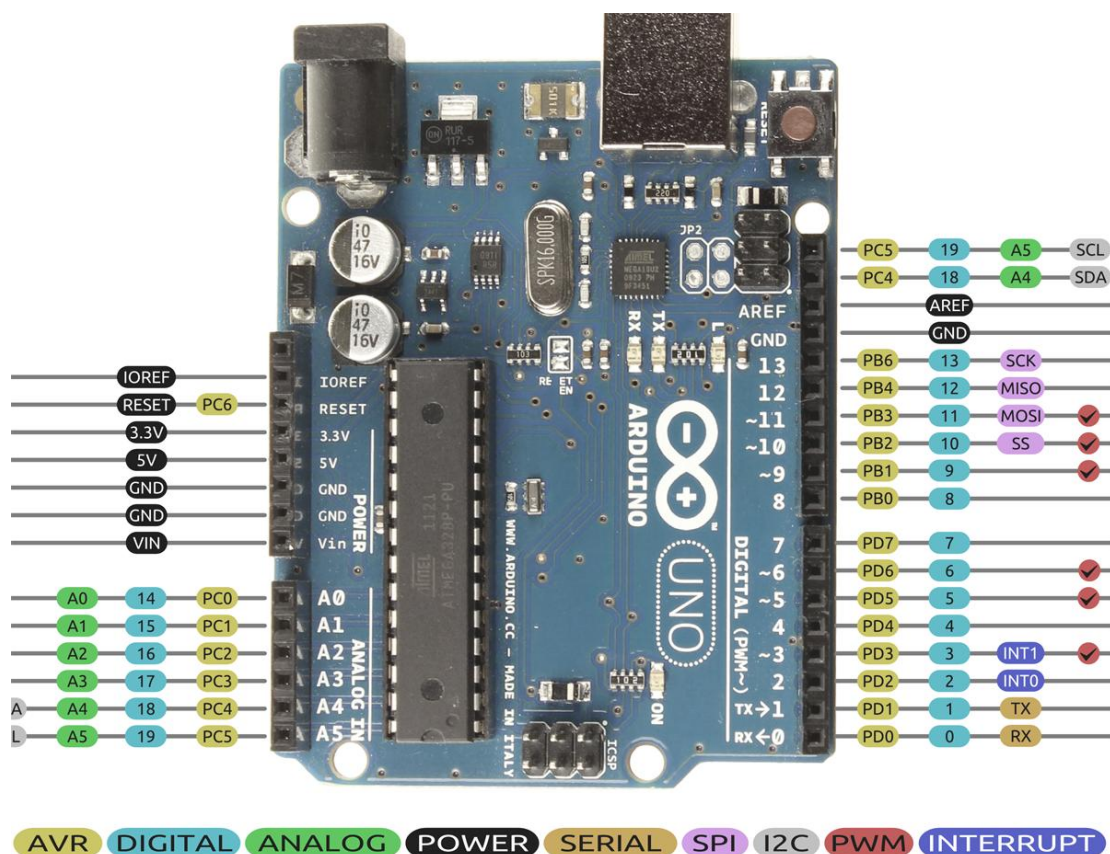
Arduino se puede utilizar para desarrollar objetos interactivos autónomos o puede ser conectado a software del ordenador.

Las plataformas Arduino están basadas en los microcontroladores Atmega, chips sencillos y de bajo coste que permiten el desarrollo de múltiples diseños. Al ser open-hardware, tanto su diseño como su distribución son libres.

Características:

- ▶ 14 Pines de entrada/salida digitales: 0 – 13
- ▶ (de los cuales 6 pueden ser salidas de pulso modulable PWM: 3, 5, 6, 9, 10, 11)
- ▶ 6 Pines de entrada analógicos: Analog Input Pins A0 – A5
- ▶ Voltaje de entrada (recomendado): 7-12V Voltaje de operación: 5V
- ▶ Voltaje de entrada (mín. y máx.) 6-20V
- ▶ Corriente de E/S (por pin) 40 mA
- ▶ Memoria Flash 32 KB SRAM 2 KB - Microcontrolador ATmega328 - EEPROM 1 KB -
- ▶ Frecuencia de reloj 16 MHz
- ▶ Transmisión en serie: pines 0 (Rx) y 1 (Tx)

Diagrama de pines de Arduino UNO:



Conectar Arduino con el PC y probar programa:

- ▶ Consigue un Arduino y un cable USB
- ▶ Descarga el IDE de Arduino: <http://arduino.cc/es/Main/Software> o de la tienda de Windows
- ▶ Conecta la placa: usando el cable USB (LED verde de alimentación PWR encendido)
- ▶ Instala los drivers: deberían descargarse e instalarse automáticamente
- ▶ Ejecuta la Aplicación Arduino: doble click en la aplicación Arduino
- ▶ Abre el ejemplo Blink: programa ejemplo para hacer parpadear un LED: File > Examples > Digital > Blink.
- ▶ Selecciona tu placa: Menu: Tools > Board. Para chip ATmega 328 : Duemilanove or Nano w/ ATmega328
- ▶ Selecciona tu puerto serie: Tools | Serial Port (probable es que sea COM3)
- ▶ Sube el sketch a la placa: pulsa sobre el botón "Upload" parpadear los led RX y TX y luego el led de la placa conectado al pin 13 (L) comienza a parpadear (con un color naranja). Funcionamiento OK

Ejemplo de sketch (programa)

Un sketch es el nombre que usa Arduino para un programa en código.

Funciones:

```
void setup()          //-> Función, de nombre, "setup" de parámetros vacíos ( ) y que no devuelve nada (void)
{
  pinMode(ledPin, OUTPUT); // -> llamada a una función previamente definida a la que se le pasan 2 valores
}
```

pinMode(): configura un pin como entrada o salida. Para utilizarla, le pasas el número del pin que vas a configurar y la constante INPUT o OUTPUT. Como input, un pin puede detectar el estado de un *sensor*, y como salida, puede manejar un *actuador*, como un LED.

digitalWrite(): envía un valor a un pin. Ejemplo: digitalWrite(ledPin, HIGH) -> modifica ledPin (pin 13) como HIGH, o 5 volts. Enviando LOW a un pin lo conecta a tierra, o 0 volts.

delay(1000): Hace a Arduino esperar 1000 milisegundos o 1 segundo.

setup(): Es llamada una vez, al inicio del sketch (on create) para definir los pins o inicializar bibliotecas

loop(): Es llamada repetidamente y es el corazón de la mayoría de los sketches. Necesitas incluir ambas funciones en tu sketch, aun cuando no las necesites para nada.

Páginas de interés

- Véase más en: <http://www.arduino.cc/es>
- Luchador de sumo: <http://www.austral.edu.ar/ingenieria/files/2012/06/Orientaciones.pdf>
- Para referencia del lenguaje: http://arduino.cc/es/Reference/HomePage#.Uzv1ePl_vX4
- Ejemplos: http://arduino.cc/es/Tutorial/HomePage#.Uzv1s_l_vX4
- Programación arduino visual en modo bloques: <https://github.com/gasolin/BlocklyDuino>
- Simulador de Arduino: VirtualBreadboard: <http://www.virtualbreadboard.com/>
- Simulador on line libre: <http://123d.circuits.io/>
- Tutoriales Introducción: <http://www.prometec.net/>
- Tutoriales: <http://txapuzas.blogspot.com.es/2011/10/paperrobot-chasis-para-robot-con.html>

MANUAL LENGUAJE DE PROGRAMACIÓN ARDUINO

1. ESTRUCTURA

- setup () (*inicialización*)
- loop () (*bucle*)

Estructuras de control

- if (*comparador si-entonces*)
- if...else (*comparador si...sino*)
- for (*bucle con contador*)
- switch case (*comparador múltiple*)
- while (*bucle por comparación booleana*)
- do... while (*bucle por comparación booleana*)
- break (*salida de bloque de código*)
- continue (*continuación en bloque de código*)
- return (*devuelve valor a programa*)

Operadores

- = (asignación)
- + (suma)
- - (resta)
- * (multiplicación)
- / (división)
- % (resto)
- == (igual a)
- != (distinto de)
- < (menor que)
- > (mayor que)
- <= (menor o igual que)
- >= (mayor o igual que)
- && (y)
- || (o)
- ! (negación)
- ++ (incrementa)
- -- (decremento)
- += (composición suma)
- -= (composición resta)
- *= (composición multiplicación)
- /= (composición división)

2. VARIABLES

Constantes

- HIGH | LOW
- INPUT | OUTPUT
- true | false
- Const

Tipos de Datos

- boolean (*booleano*)
- char (*carácter*)
- byte
- int (*entero*)
- unsigned int (*entero sin signo*)
- long (*entero 32b*)
- unsigned long (*entero 32b sin signo*)

- float (*en coma flotante*)
- double (*en coma flotante de 32b*)
- string (*cadena de caracteres*)
- array (*cadena*)
- void (*vacío*)

3. FUNCIONES

E/S Digitales

- pinMode ()
- digitalWrite ()
- digitalRead ()

E/S Analógicas

- analogRead ()
- analogWrite () - PWM (*modulación por ancho de pulso*)

E/S Avanzadas

- tone ()
- noTone ()
- shiftOut ()
- pulseIn ()

Tiempo

- millis ()
- micros ()
- delay ()
- delayMicroseconds ()

Matemáticas

- min () (*mínimo*)
- max () (*máximo*)
- abs () (*valor absoluto*)
- constrain () (*limita*)
- map () (*cambia valor de rango*)
- pow () (*eleva a un número*)
- sq () (*eleva al cuadrado*)
- sqrt () (*raíz cuadrada*)

Trigonometría

- sin () (*seno*)
- cos () (*coseno*)
- tan () (*tangente*)

Aúmeros Aleatorios

- randomSeed ()
- random ()

Comunicación

- Serial
- begin ()
- end ()
- available ()
- read ()
- flush ()
- print ()
- println ()
- write ()

1. ESTRUCTURA:

SETUP()

La función setup() se establece cuando se inicia un programa -sketch. Se emplea para iniciar variables, establecer el estado de los pins, inicializar librerías, etc. Esta función se ejecutará una única vez después de que se conecte la placa Arduino a la fuente de alimentación, o cuando se pulse el botón de reinicio de la placa.

```
int buttonPin = 3;
void setup()
{
    Serial.begin(9600);
    pinMode(buttonPin, INPUT);
}
void loop()
{
    // aquí va el bucle
}
```

LOOP()

Esta función se ejecuta consecutivamente, permitiendo al programa variar y responder en cada instante. Úsala para controlar de forma activa la placa Arduino.

```
int buttonPin = 3; // setup inicializa la comunicación serial y el buttonPin
void setup()
{
    beginSerial(9600);
    pinMode(buttonPin, INPUT);
}
// loop obtiene el estado del pin del botón cada vez y de estar presionado,
// lo comunica por serial.
void loop()
{
    if (digitalRead(buttonPin) == HIGH)
        serialWrite('H');
    else
        serialWrite('L');
    delay(1000);
}
```

ESTRUCTURAS DE CONTROL

Operadores de Comparación (comparadores):

$x == y$	(x es igual a y)	$x > y$	(x es mayor a y)
$x != y$	(x no es igual a y)	$x <= y$	(x es menor o igual a y)
$x < y$	(x es menor a y)	$x >= y$	(x es mayor o igual a y)

Atención: No confundir un signo de igual solo de asignación, con el igual doble de comparación.

IF / ELSE

if/else permite separar las acciones en dos casos; el caso verdadero o falso según se cumpla o no la comparación.

Sintaxis: if (comparación) {acción_verdadera} else {acción_falsa}

else if permita realizar múltiples comprobaciones en una misma estructura de condiciones

```
if (pinCincoInput < 500)
{
    // acción A
}
else
{
    // acción B
}
```

```
if (pinCincoInput < 500)
{
    // ejecutar A
}
else if (pinCincoInput >= 1000)
{
    // ejecutar B
}
else
{
    // ejecutar C
}
```

Entonces un bloque else if puede ser usado con o sin else al final.

Pueden emplearse If-else anidados de modo que la cantidad de declaraciones else if, y sus ramificaciones son ilimitadas.

Otra forma de expresar ramificaciones (*branching* en inglés), es con la declaración switch case

SENTENCIA SWITCH / CASE

Como las sentencias **if**, **switch...case** permite agrupar diferentes códigos que deberían ser ejecutados en función de varias condiciones.

La palabra clave **break** sale de la sentencia switch, y es usada típicamente al final de cada case.

Ejemplo

```
switch (var) {
  case 1:
    //hacer algo cuando sea igual a 1
    break;
  case 2:
    //hacer algo cuando sea igual a 2
    break;
  default:
    // si nada coincide, ejecuta el "default"
    // el "default" es opcional
}
```

Sintaxis

```
switch (var) {
  case etiqueta:
    // sentencias
    break;
  case etiqueta:
    // sentencias
    break;
  default:
    // sentencias
}
```

BUCLES

FOR

La declaración **for** es usada para repetir un bloque encerrado entre llaves un número determinado de veces. El bucle for tiene tres partes o argumentos en su inicialización:

```
for (initialization; condition;
    increment) {
    //función(es);
}
```

La *initialization*, o inicialización, se produce sólo la primera vez. Cada vez que se va a repetir el bucle, se revisa la *condition*, o condición: si es cierta, el bloque de funciones (y el incremento del contador) se ejecutan, y la condición vuelve a ser comprobada de nuevo. Si la condición es falsa, el bucle termina.

El bucle for, en C, es mucho más flexible que en otros lenguajes. Un ejemplo es hacer apagarse/encenderse un LED poco a poco,

```
/* Variar la intensidad de un LED usando un salida PWM
int PWMpin = 10; En el pin 10 hay un LED en serie con una
resistencia de 470 ohmios */
void setup()
{
    // no es necesario nada aquí
}
void loop()
{
    for (int i=0; i <= 255; i++){
        analogWrite(PWMpin, i);
        delay(10);
    }
}
```

```
void loop()
{
    int x = 1;
    for (int i = 0; i > -1; i = i + x){
        analogWrite(PWMpin, i);
        if (i = 255) x = -1; // cambia de signo para apagarlo
        delay(10);
    }
}
```

WHILE

Grupo de código que se ejecuta continuamente, hasta que la expresión de dentro del paréntesis, (), pasa a ser falsa.

Sintaxis

```
while(expresion){
  // sentencia(s)
}
```

Ejemplo

```
var = 0;
while(var < 200){
  // haz algo repetitivo 200 veces
  var++;
}
```

DO - WHILE

El bucle "do" trabaja de la misma manera que el bucle "while", con la excepción de que la condición se comprueba al final del bucle, por lo que este bucle se ejecuta "siempre" al menos una vez.

Sintaxis:

```
do
{
    // bloque de instrucciones
} while (condición);
```

Ejemplo:

```
do
{
    delay(50); // espera a que los sensores se estabilicen
    x = readSensors(); // comprueba los sensores
} while (x < 100); //si se cumple la condición se repite
```

BREAK

Break es usado para salir de los bucles **do**, **for**, o **while**, pasando por alto la condición normal del bucle. Es usado también para salir de una estructura de control **switch**.

```
for (x = 0; x < 255; x ++)  
{  
  digitalWrite(PWMPin, x);  
  sens = analogRead(sensorPin);  
  if (sens > threshold){ // bail out on sensor detect  
    x = 0;  
    break; // sale del bucle for.  
  }  
  delay(50);  
}
```

CONTINUE

Omite el resto de iteraciones de un bucle (**do**, **for**, o **while**). Salta el bucle a la siguiente iteración.

```
for (x = 0; x < 255; x ++)  
{  
  if (x > 40 && x < 120){ // crea un salto en estos valores  
    continue;  
  }  
  digitalWrite(PWMPin, x); }  
}
```

RETURN

Termina una función y devuelve un valor a la función que la llama. Puede no devolver nada.

Sintaxis

```
return;  
return valor; // ambas formas son correctas
```

Parámetros

valor: cualquier variable o tipo constante

Ejemplos

Una función que compara la entrada de un sensor a un umbral

```
int comprobarSensor(){  
  if (analogRead(0) > 400) {  
    return 1;  
  }  
  else{  
    return 0;  
  }  
}
```

La palabra clave *return* también es útil para depurar una sección de código sin tener que comentar una gran cantidad de líneas de código posiblemente incorrecto.

```
void loop(){  
  // -->código bueno aquí  
  return;  
  // --> resto del programa del que se desconfía, nunca será ejecutado por estar detrás de return  
}
```

SINTAXIS

- › **; PUNTO Y COMA** Utilizado para terminar una declaración. **Ejemplo:** `int a = 13;`
- › **{ } LLAVES:** Se utilizan en diferentes construcciones para agrupar o englobar un conjunto de código.
- › **COMENTARIOS:** son líneas en el programa para aclarar el funcionamiento del programa
`// para una línea /* para varias líneas */`

OPERADORES

- › **= OPERADOR DE ASIGNACIÓN:** Guarda el valor en la derecha del símbolo "="
- › **% (MÓDULO):** Calcula el resto de la división entre dos enteros.
- › **&& (AND lógico):** Verdadero sólo si ambos operadores son Verdadero.
- › **|| (OR lógico):** Verdadero si alguno de los dos operadores es Verdadero.
- › **! (NOT):** Verdadero si el operador es Falso.
- › **++ (incremento) / -- (disminución):** Incrementa o disminuye una variable

CONSTANTES: Se usan para facilitar la lectura de los programas. Suelen ser valores fijos

- › **Constantes Booleanas verdadero (true) y falso (false):** para representar si algo es cierto o falso en Arduino: **true**, y **false**. Se escriben en minúsculas, al contrario que HIGH, LOW, INPUT, y OUTPUT.
- › **Constantes numéricas:** Normalmente, las constantes *integer* son tratadas como enteros EN base 10 (decimales), pero se puede utilizar notación especial (formateadores) para usar en otras bases.

Base	Ejemplo	Comentario
10 (decimal)	123	Ninguno.
2 (binario)	B1111011	Antecede "B" Sólo funciona con valores de 8 bits
8 (octal)	0173	Antecede "0" Caracteres 0 al 7 válidos.
16 (hexadecimal)	0x7B	Antecede "0x" Caracteres 0-9,A-F, a-f válidos.

VARIABLES. Tipos de datos

- › **BOOLEANOS:** Un **booleano** sólo puede tomar dos valores, Verdadero o Falso.
- › **CHAR:** Ocupa un byte de memoria y almacena un valor de carácter. Los caracteres literales se escriben con comillas simples: 'A' (para varios caracteres -strings- utiliza dobles comillas "ABC").
- › **BYTE:** Un byte almacena un número sin signo hasta 255.
- › **INT:** Integers (Números enteros) entre -32,768 hasta 32,768
- › **UNSIGNED INT** Sólo valores positivos, entre 0 y 65.535 ($2^{16} - 1$).
- › **LONG** Tamaño extendido para almacenar números de 32 bits (4 bytes), de $\pm 2.147.483.647$
- › **UNSIGNED LONG** Extendidas para almacenar números. No almacenan números negativos,
- › **FLOAT** Tipo variable para los números en coma flotante (número decimal).
- › **DOUBLE:** Número en coma flotante de doble precisión. Ocupa 4 bytes.
- › **STRING:** Arrays de caracteres (tipo char) que terminan con el caracter NULL.
- › **ARRAYS:** Matriz o colección de variables que son accedidas mediante un número de índice.
- › **VOID:** *void* (vacío) Indica que se espera que **no** devuelva información a la función donde fue llamada.

3. FUNCIONES: E/S DIGITALES

PINMODE(): Configura el pin especificado para comportarse como una entrada o una salida.

DIGITALWRITE(): Escribe un valor HIGH o LOW hacia un pin digital.

Si el pin ha sido configurado como OUTPUT con pinMode(), su voltaje será establecido al correspondiente valor: 5V (o 3.3V en tarjetas de 3.3V) para HIGH, 0V (tierra) para LOW.

Si el pin es configurado como INPUT, escribir un valor de HIGH con digitalWrite() habilitará una resistencia interna de 20K conectada en *pullup*. Escribir LOW invalidará la resistencia.

DIGITALREAD() Lee el valor de un pin digital especificado, HIGH o LOW.

ANALOGREAD(): Lee el valor de tensión en el pin analógico especificado entre 0 y 5 voltios convirtiéndolos en un número entero entre 0 y 1023.

ANALOGWRITE(): Escribe un valor analógico por pulso (PWM) en un pin valores hasta 256.

TONE(): Genera una onda cuadrada a frecuencia especificada en un pin. La duración especificada

NOTONE(): Detiene la generación de la señal cuadrada que se activa al hacer uso de la función tone().

SHIFTOUT(): Desplaza un byte de datos bit a bit. Empieza desde el bit más significativo (más izquierda)

PULSEIN(): Lee un pulso (ya sea HIGH —alto— o LOW —bajo—) en un pin.

MIN(X, Y) Calcula el mínimo de dos números.

MAX(X, Y) Calcula el máximo de dos números.

ABS(X) Calcula el valor absoluto de un número.

CONSTRAIN(X, A, B) Restringe un número a un rango definido.

MAP(VALUE, FROMLOW, FROMHIGH, TOLOW, TOHIGH)

Re-mapea un número desde un rango hacia otro. Ésto significa que, un valor (*value*) con respecto al rango *fromLowfromHight* será mapeado al rango *toLow-toHigh*.

POW(BASE, EXPONENTE): Calcula un número elevado a otro número.

SQ(X) Calcula el cuadrado de un número: el número multiplicado por el mismo.

SQRT(X): Calcula la raíz cuadrada de un número.

SIN(RAD): Calcula el seno de un ángulo (en radianes). El resultado será entre -1 y 1.

COS(RAD): Calcula el coseno de un ángulo (en radianes). El resultado estará entre -1 y 1.

TAN(RAD): Calcula el coseno de un ángulo (en radianes)

RANDOMSEED(SEED): randomSeed() inicializa el generador de números pseudoaleatorios,

RANDOM(): La función *random* genera números pseudoaleatorios.

4. COMUNICACIÓN

SERIAL Se utiliza para la comunicación entre la placa Arduino por puerto serie / USB.

BEGIN(): Establece la velocidad de datos en bits por segundo (baudios) para la transmisión de datos

END(): Desactiva la comunicación serie, permitiendo a los pines RX y TX ser usados como pines digitales.

AVAILABLE(): Devuelve el número de bytes disponibles para ser leídos por el buffer del puerto serie.

READ(): Lee los datos entrantes del puerto serie.

FLUSH(): Vacía el búfer de entrada de datos en serie.

PRINT(): Imprime los datos al puerto serie como texto ASCII.

PRINTLA(): Imprime los datos al puerto serie como texto ASCII seguido de un retorno de carro

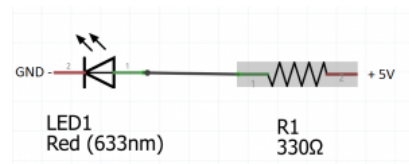
WRITE() Escribe datos binarios en el puerto serie. Se envían como un byte o una serie de bytes.

TUTORIALES BÁSICOS.

Lección 1: Encender un LED externo:

Material requerido:

Arduino Uno o similar - Protoboard. - Un diodo LED - Una resistencia de 220 o 330 Ohmios – Cables



Cálculo de la Resistencia: $V = I \cdot R$

Para una intensidad de unos 20 mA: $R = V / I \rightarrow 5 / 0.02 = 250 \Omega$

Como no tenemos de 250, utilizaremos una de entre 220 Ω o de 330 Ω

Colores: 220: rojo-rojo-marrón
330: naranja-naranja-marrón →

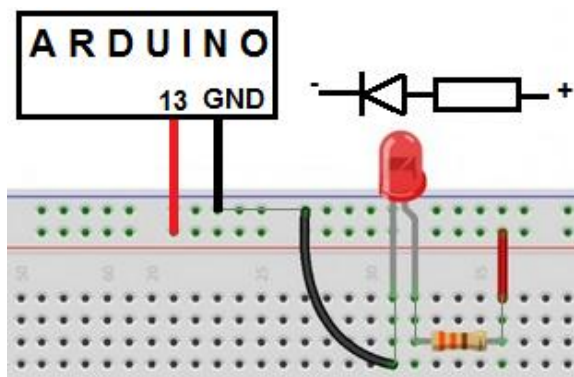
Color	figura significativa			Multiplicador	Tolerancia (%)
	banda 1	banda 2	banda 3		
Negro	0	0	0	x 1	
Marron	1	1	1	x 10	± 1% (F)
Rojo	2	2	2	x 100	± 2% (G)
Naranja	3	3	3	x 1K	
Amarillo	4	4	4	x 10K	
Verde	5	5	5	x 100K	± 0.5% (D)
Azul	6	6	6	x 1M	± 0.25% (C)
Violeta	7	7	7	x 10M	± 0.1% (B)
Gris	8	8	8	x 100M	± 0.05% (A)
Blanco	9	9	9	x 1G	
Oro			solo para 5 y 6 bandas	x 0.1	± 5% (J)
Plata				x 0.01	± 10% (K)
Nada					± 20% (M)



Conexión en la protoboard:

Cuando nuestro programa ponga un valor de HIGH (5V) en el pin 13 permitirá el flujo de corriente por el circuito iluminando el LED.

Con LOW sencillamente el circuito estará apagado, sin tensión



Programación:

// Primer Programa sketch: Blink

```
void setup() {
  pinMode(13, OUTPUT);
}
void loop() {
  digitalWrite(13, HIGH); // enciende
  delay(1000);           // retardo 1
  segundo
  digitalWrite(13, LOW); // apaga LOW
  delay(1000);           // retardo 1
  segundo
}
```

// Segundo Programa sketch

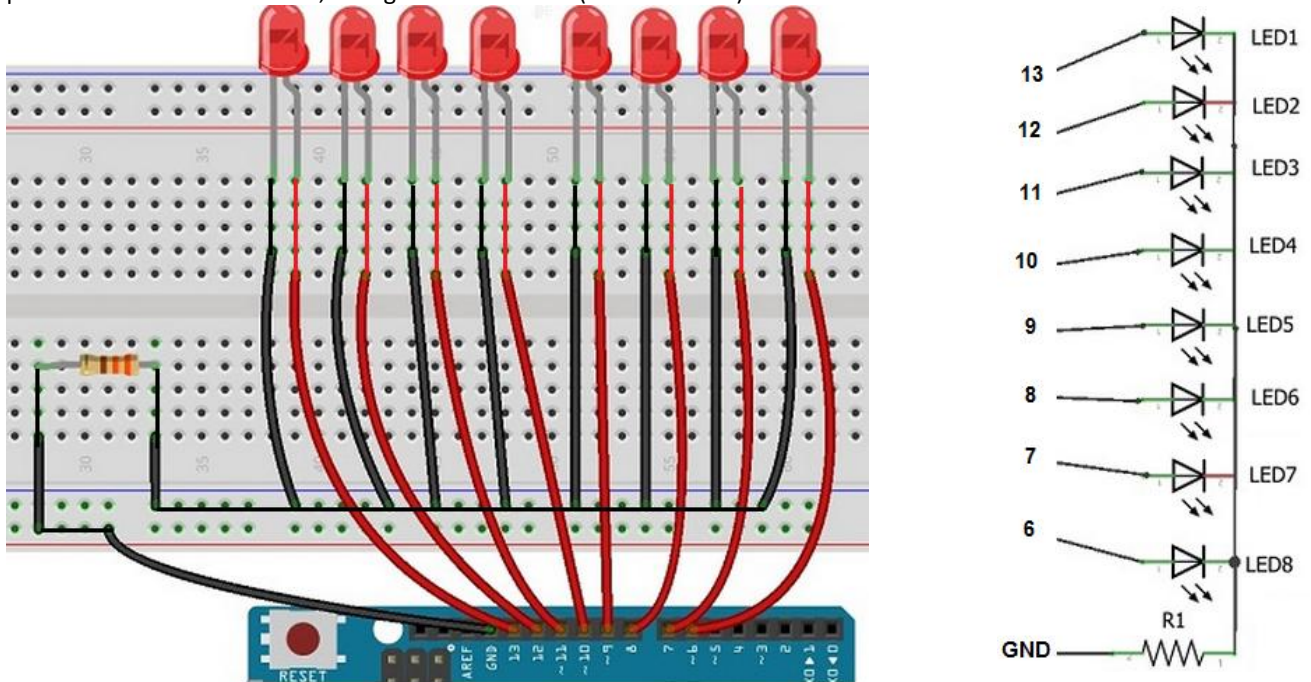
// parpadea tres veces y luego queda encendido

```
int contar=0; // variable para contar
void setup() {
  pinMode(13, OUTPUT); // inicializa pin
}
void loop() {
  while (contar<3)
  {
    digitalWrite(13, HIGH); // LED on
    delay(500);             // retardo
    digitalWrite(13, LOW);  // LED off
    delay(500);             // retardo
    contar++; //incrementa
  }
  digitalWrite(13, HIGH);
}
```

Lección 2: Control de varias salidas por LED (El coche fantástico)

Objetivo: ocho leds que se enciendan secuencialmente.

Conexión en la Protoboard: Utilizaremos los ocho pines del 6 al 13 como salida de 8 leds que vuelven, pasando todos por una resistencia de 220 Ω, al negativo tierra GND (ánodo común).



Utilizaremos dos veces la instrucción **for**, que nos permite repetir un bloque de instrucciones un número determinado de veces.

El primer ciclo **for** hace que las luces se encienda en secuencia desde la 6 hasta la 13. El segundo bucle entra a continuación empezando con la luz 12 (para no repetir la 13) y finalizando con la 7(para no repetir la 6), y vuelta a empezar.

En el segundo bucle **for** hemos hecho una cuenta atrás diciéndole a la variable que se decrementara en uno en cada iteración mediante la instrucción `sale--`

En realidad, la variable `sale` no era necesario haberla declarado antes ya que se puede crear dentro de la propia instrucción `for`

Para finalizar es bueno “apagar” los pines de salida para no interferir en el siguiente ejercicio:

```
void loop(){
  for(sale=6;sale<14;sale++)
    digitalWrite(sale, LOW);
}
```

```
int sale=0;
void setup(){
  for(sale=6;sale<14;sale++)
    pinMode(sale, OUTPUT); // pines 6 a 13 como salida
}
void loop(){
  for(sale=6;sale<14;sale++) // cuenta de 6 a 14
  {
    digitalWrite(sale, HIGH);
    delay(100);
    digitalWrite(sale, LOW);
    delay(100);
  }
  for(sale=14;sale>6;sale--) // cuenta atrás
  {
    digitalWrite(sale, HIGH);
    delay(100);
    digitalWrite(sale, LOW);
    delay(100);
  }
}
```

Variante para 4 leds con incremento de tiempo:

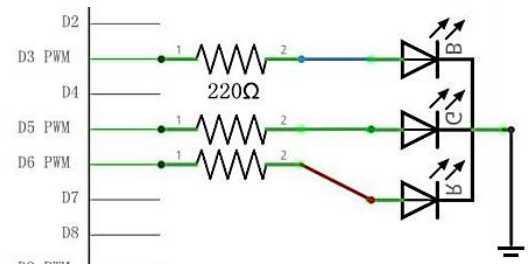
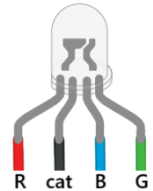
```
int sale=0;
int tiempo=100;
void setup(){
  for(sale=6;sale<10;sale++) //conectar pines de 6 a 9
    pinMode(sale, OUTPUT); // pines 6 a 9 como salida
}
void loop(){
  for(sale=6;sale<10;sale++) // cuenta de 6 a 9
  {
    digitalWrite(sale, HIGH);
    delay(tiempo);
    digitalWrite(sale, LOW);
    delay(tiempo);
  }
  tiempo=tiempo+20;
}
```

Lección 3: Control de LED RGB. Pulsos analógicos

Internamente son tres leds: rojo-verde y azul, todos en uno con ánodo o cátodo común.

Podemos variar su intensidad desde los pines de pulsos analógicos de salida, marcados con ~ en el Arduino.

El ánodo común es el segundo desde la parte plana.



```
/** ** LEDRGB ** **
```

```
// Define Pines
#define BLUE 3
#define GREEN 5
#define RED 6

void setup()
{
  pinMode(RED, OUTPUT);
  pinMode(GREEN, OUTPUT);
  pinMode(BLUE, OUTPUT);
  digitalWrite(RED, HIGH); //rojo on
  digitalWrite(GREEN, LOW);
  digitalWrite(BLUE, LOW);
}
// variables para el loop
int redVal;
int greenVal;
int blueVal;

void loop()
{
  #define tiempo 10 // constante tiempo

  redVal = 255; // rojo a tope
  greenVal = 0;
  blueVal = 0;

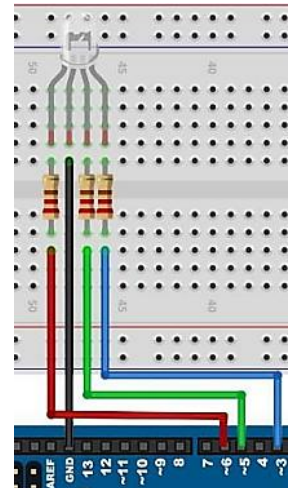
  for(int i = 0; i < 255; i += 1)
  {
    redVal -= 1; //baja el rojo
    greenVal += 1; //sube el verde
```

```
analogWrite(RED, redVal);
analogWrite(GREEN, greenVal);
delay(tiempo);
}

redValue = 0;
greenValue = 255; // verde a
tope
blueValue = 0;

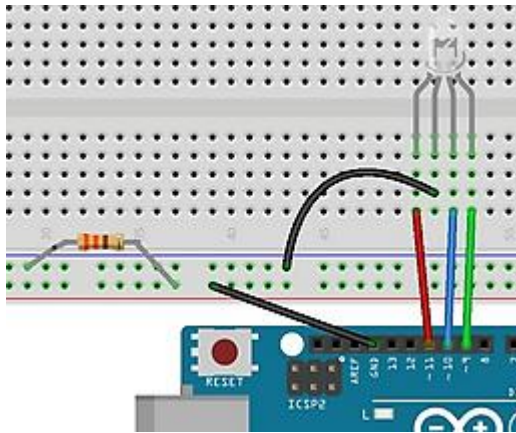
for(int i = 0; i < 255; i += 1)
// baja verde y sube azul
{
  greenVal -= 1;
  blueVal += 1;
  analogWrite(GREEN, greenVal);
  analogWrite(BLUE, blueVal);
  delay(tiempo);
}

redVal = 0;
greenVal = 0;
blueVal = 255; //azul a tope
for(int i = 0; i < 255; i += 1)
{
  blueValue -= 1; // baja azul
  redValue += 1; //y sube rojo
  analogWrite(BLUE, blueValue);
  analogWrite(RED, redValue);
  delay(tiempo);
}
}
```



Lección 4: Colores en LED RGB. Salidas analógicas en pines digitales y usar una función.

- El pin más largo de un LED RGB de ánodo común es el GND.
- Al lado de GND hay dos pines a un lado y uno solitario al otro. Por lo normal el solitario es el rojo. Así pues el pin out (patillaje) de un RGB LED suele ser: **R - GND - G - B**



Arduino permite escribir valores analógicos de 0 a 255 en los pines digitales si son del tipo pulso PWM.

Función color: Creamos la función **color** y le pasamos como parámetros los tres niveles de color. La función **random(N)** devuelve un valor al azar comprendido entre 0 y N y lo usaremos para generar colores aleatorios:

```
Color(random(255), random(255), random(255))
```

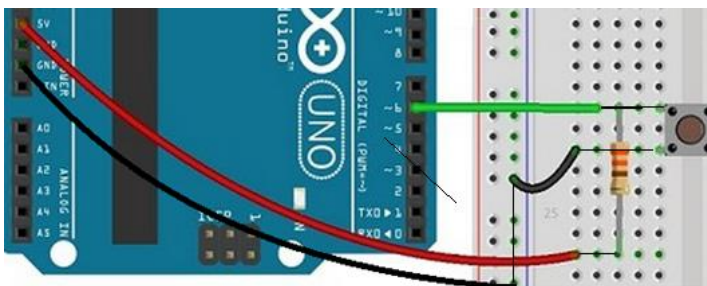
```
void setup()
{
  for (int i =9 ; i<12 ; i++)  pinMode(i, OUTPUT);
}

void Color(int R, int G, int B)
{
  analogWrite(9 , R) ; // Red - Rojo
  analogWrite(10, G) ; // Green - Verde
  analogWrite(11, B) ; // Blue - Azul
}

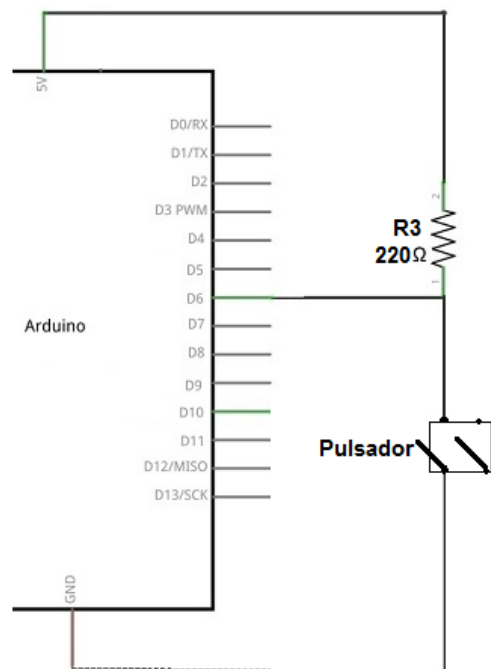
void loop() //-> modo test
{
  Color(255 ,0 ,0) ;
  delay(500);
  Color(0,255 ,0) ;
  delay(500);
  Color(0 ,0 ,255) ;
  delay(500);
  //Color(random(255), random(255), random(255));
  //delay(1000);
}

void loop() //-> Variante para colores aleatorios...
{
  Color(random(255), random(255), random(255)) ;
  delay(500);
}
```

Lección 4: Detección por entrada digital. Pulsador a negativo



```
void setup()
{
  pinMode( 13, OUTPUT) ; // Pin 13 como salida
  pinMode( 6 , INPUT) ; //Pin 6 como entrada
}
void loop()
{
  int valor = digitalRead(6) ; // leemos el valor
  digitalWrite(13, valor) ;
}
```



Mientras no pulsemos el pin 6 de Arduino tendrá una lectura de tensión alta (HIGH). En cambio cuando pulsemos el pulsador cerraremos el circuito del pin 6 a Ground con lo que leerá tensión baja, LOW y nuestro programa apagará la luz del LED 13.

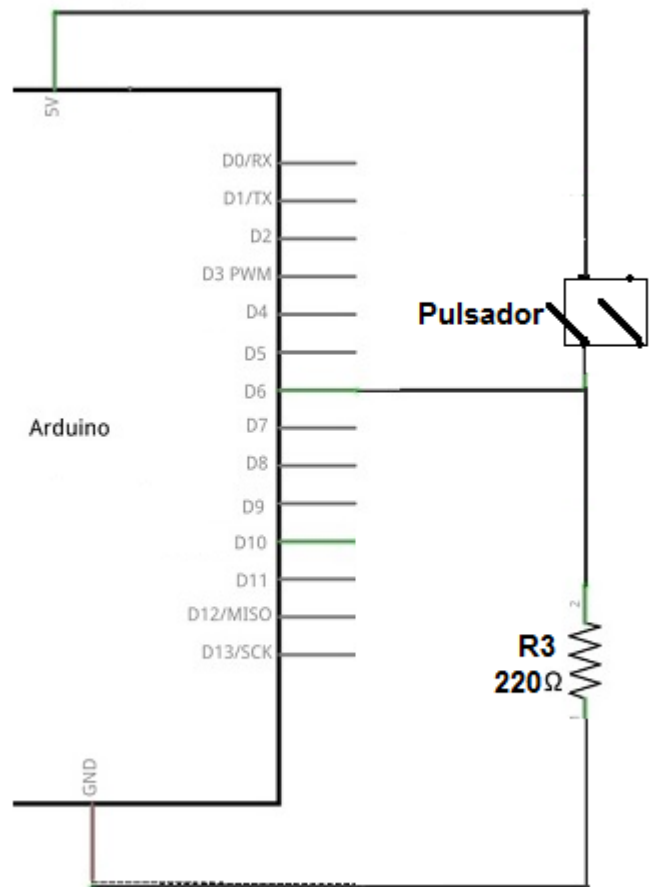
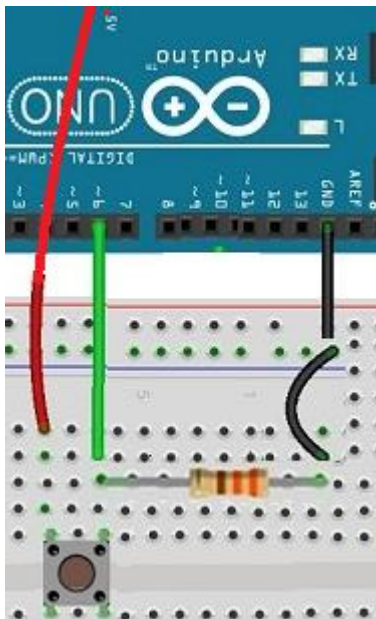
Si no pudiéramos la resistencia R3, al pulsar S1 leeríamos correctamente LOW en el pin 6. Pero al dejar de pulsar S1 el pin 6 estaría en un estado flotante, que es ni HIGH ni LOW sino indeterminado. Como esto es inaceptable en circuitos digitales forzamos una lectura alta con R3.

A esta resistencia que fuerza el valor alto en vacío se le conoce como *pullup*. Si la conectáramos a masa para forzar una lectura a Ground se le llamaría pulldown. Esta resistencia es clave para que las lecturas del pulsador sean consistentes.

Pulsación entrada por positivo:

Mientras no pulsemos el pin 6 de Arduino tendrá una lectura de tensión baja a Tierra. En cambio, cuando pulsemos el pulsador cerraremos el circuito del pin 6 a +5V con lo que leerá tensión alta y nuestro programa encenderá la luz del LED 13.

El programa es el mismo que el anterior pero su lógica es negada respecto al anterior.



Variación con Interruptor magnético:

Equivale a un interruptor con contacto normalmente cerrado y se abre al separar la pieza por magnetismo.



Lección 5: Interruptor de 3 contactos y 2 estados: NC y NA. Circuito de seguridad negado

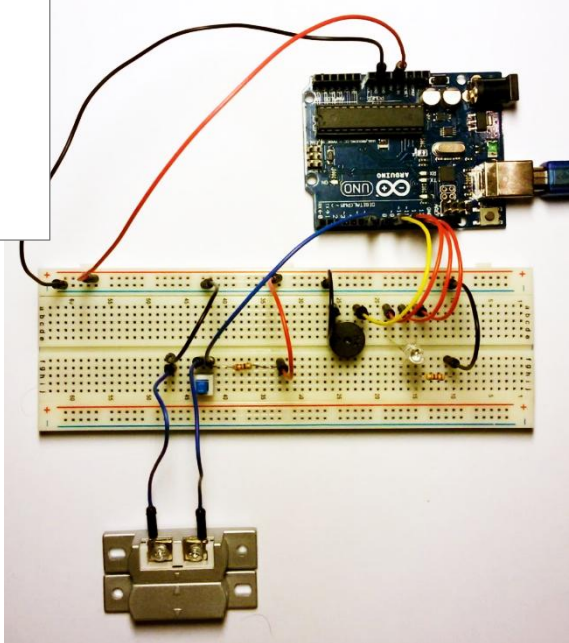
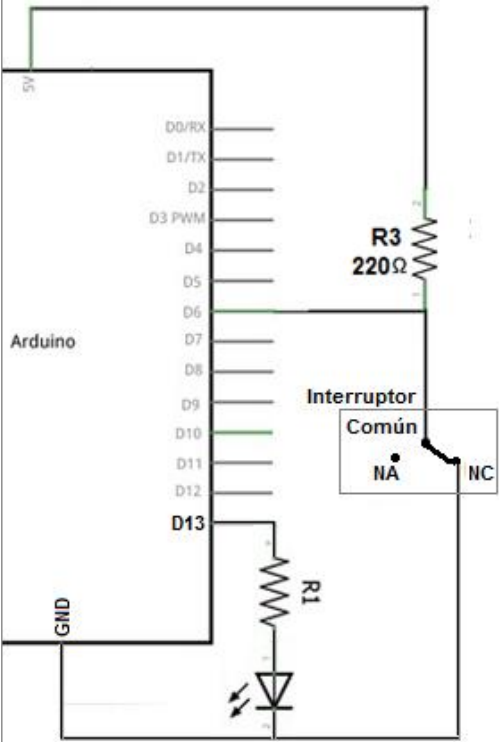
La lógica negada se emplea en circuitos de seguridad y alarmas: Un circuito cerrado llamado serie, está alimentando constantemente el pin de entrada con señal alta. Si este circuito sufre un corte, deja de recibir la señal alta y entonces activa el estado de alarma.

Utilizaremos un interruptor de 3 contactos. El pin central o común en estado normal (sin pulsar) tiene un contacto cerrado y el otro abierto (utilizaremos el estado normalmente cerrado NC). Al pulsar abriremos el contacto normalmente cerrado y cerraremos el normalmente abierto.

```

int estado=0; //estado de aviso por serie abierta
int alarma=0; //estado de alarma
int const tiempo_espera=10; //regula tiempo de espera

void setup()
{
pinMode( 13, OUTPUT) ; // Pin 13 salida azul
pinMode( 12, OUTPUT) ; // 12 salida roja
pinMode( 11, OUTPUT) ; // 12 salida verde
pinMode( 6 , INPUT) ; //Pin 6 entrada
digitalWrite(11, HIGH) ;
}
void loop()
{
estado = digitalRead(6) ; // leemos estado
if (estado==1) //aviso abierto
{
if (alarma==0) { //si no hay alarma avisa parpadeando...
int i=0;
for (i=0;i<tiempo_espera;i++){
digitalWrite(13, HIGH); // LED on
delay(500); // retardo
digitalWrite(13, LOW); // LED off
delay(500); // retardo
estado = digitalRead(6) ; // vuelve a leer estado
if (estado==0) break; //oportunidad para apagar aviso
}
}
estado = digitalRead(6) ; // vuelve a leer estado
if (estado==1) { //si sigue abierto activa alarma
digitalWrite(13, HIGH) ;
digitalWrite(12, HIGH) ;
alarma=1;
}
}
else
{
if (alarma==0) digitalWrite(13, LOW) ;
}
}
    
```



Variaciones:

Interruptor magnético:

Equivalo a un interruptor con contacto normalmente cerrado y se abre al separar la pieza por magnetismo.

LED RGB: En el programa mantén LED verde siempre encendido. Utiliza el parpadeo de aviso con el LED azul y la señal de alarma con el red rojo. (Vésase ejercicio de LED RGB)

Lección 6: Activación de un servomotor. Librería servo.

Un servo es un motor diseñado para mover su eje un ángulo fijo en respuesta a una señal de control. Útil en giros de dirección como el timón de un barco o subir una barrera que no suelen superar los 180 °.

Arduino dispone de una librería estándar con funciones para servos: Servo.h.

Pero para poder utilizar sus funciones debemos crear, lo que se conoce como un objeto del tipo Servo que llamaremos servo1. Así podremos usar las funciones más frecuentes como attach y write: servo1.attach(pin) asigna el pin de control y servo1.write(pos) escribe el valor de salida por el pin de control.

C++ es un lenguaje orientado a objetos, esto significa que podemos poner un tipo de objeto en nuestro programa (como un servo) sin más que declarar un nuevo objeto del tipo que deseamos. En la jerga de la programación se llama crear una instancia, o instanciar un nuevo objeto.

Primer programa: Hace que el servo vaya barriendo un ángulo en grados y moviendo el servo a esa posición

Segundo programa: El servo se mueve siguiendo al potenciómetro. Para ello hay que entender que el valor que leamos en la puerta A0 está comprendido entre 0 y 1024, y que estos valores tienen que distribuirse entre 0 y 180°.

Así que para calcular el ángulo correspondiente basta con hacer: $angulo = \frac{180}{1024} * lectura\ A0$

La función map(), que hace exactamente eso de un modo cómodo, librándonos de los float.

angulo = map(analogRead(A0), 0, 1024, 0, 180); Que quiere decir: Haz la proporción de los valores que leas en A0, entre 0 y 1024, en un valor comprendido entre 0 y 180 y asígnale ese valor al ángulo.

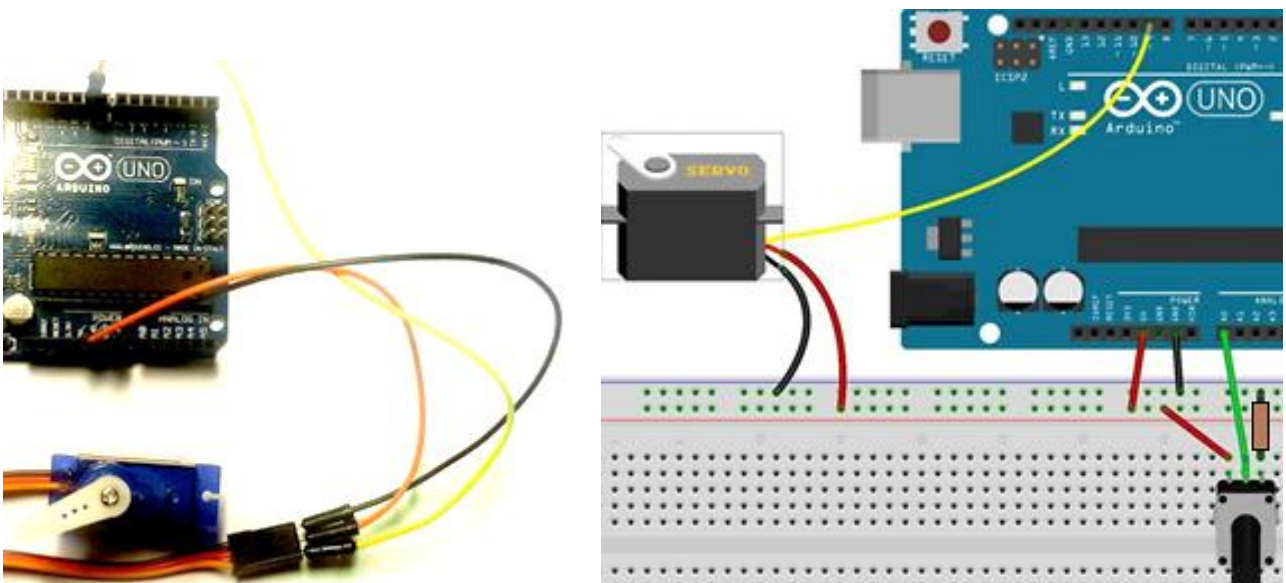
```
//primer programa: servo oscilante
//-----
#include <Servo.h>          // Incluir la librería Servo
Servo servo1;             // Crear objeto tipo Servo llamado servo1
int angulo = 0 ;

void setup()
{
  servo1.attach(9);       // Conectar servo1 al pin 9
}
void loop()
{
  for(angulo = 0; angulo <= 180; angulo++) { //incrementa
    servo1.write(angulo);
    delay(25);
  }
  for(angulo = 180; angulo >=0; angulo--) { //decrementa
    servo1.write( angulo );
    delay(25);
  }
}
```

```
//Servo regulado ángulo por potenciómetro con map
//-----
#include <Servo.h>        // Incluir la librería Servo
Servo servo1;           // Crear un objeto tipo Servo llamado servo1
int angulo = 0 ;

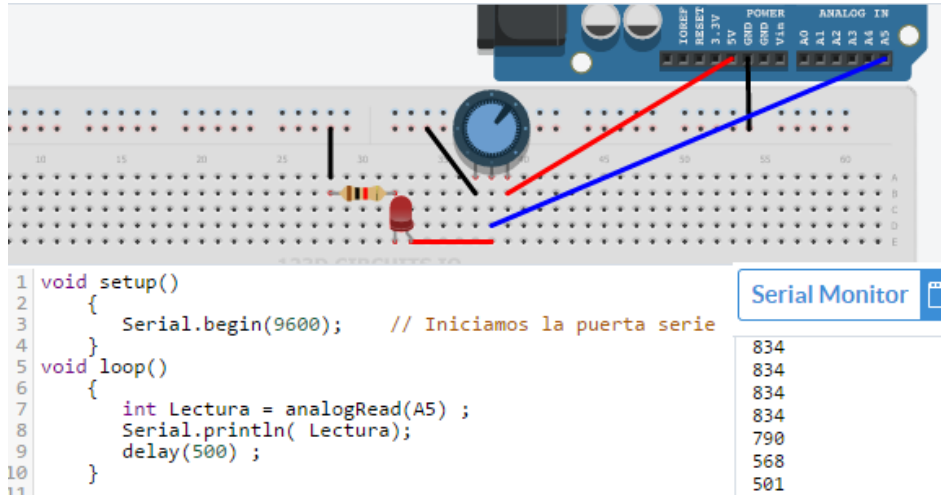
void setup()
{
  servo1.attach(9); // Conectar servo1 al pin 9
} //suele ser el cable amarillo o blanco

void loop()
{
  angulo = map( analogRead(A0), 0, 1024, 0, 180);
  servo1.write(angulo);
  delay(250);
}
```



Lección 7 a: Entradas analógicas. Potenciómetros.

Un potenciómetro es, simplemente, un mecanismo para proporcionar una resistencia variable. En este caso utilizaremos un potenciómetro de 10 kΩ que regulará la intensidad del LED, pero por seguridad, añadiremos al LED una resistencia de 220 Ω. El programa nos permitirá leer los valores del potenciómetro por la entrada analógica A5 que tomará una muestra del valor en Voltios a 9600 baudios cada medio segundo y mostrará esa lectura cada medio segundo. Este circuito está probado en el emulador *TinkerCad* antes 123d.circuits.io o Virtual Breadboard



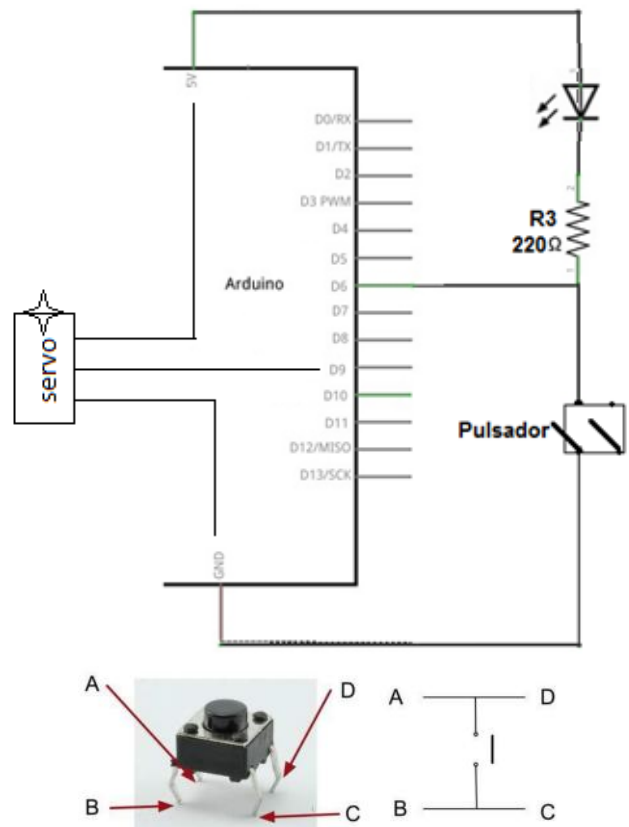
7b. Servomotor para barra de subida al activar un pulsador de flanco descendente

Un *servo* es un motor diseñado para mover su eje un ángulo fijo en respuesta a una señal de control. Útil en giros de dirección como el timón de un barco o subir una barrera que no suelen superar los 180 °.

Arduino dispone de una *librería* estándar que ya incorpora funciones para servos: **Servo.h**.

Pero para poder utilizar sus funciones debemos crear, lo que se conoce como un *objeto* del tipo *Servo* que llamaremos servo1. Así podremos usar las funciones más frecuentes como attach y write: servo1.attach(pin) asigna el pin de control y servo1.write(pos) escribe el valor de salida por el pin de control.

```
#include <Servo.h> // Incluir la librería Servo
#include <Serial.h> // Incluir la librería Serial
Servo servo1; // Crear objeto tipo Servo llamado servo1
int angulo = 0 ;
void setup()
{
  servo1.attach(9) ; // Conectar servo1 al pin 9
  pinMode(6, INPUT) ;
  angulo=0; //inicializa a cero
  servo1.write( angulo);
  Serial.begin(9600); // Iniciamos la comunicación serie
}
void loop()
{
  if ((digitalRead(6)==LOW) && (angulo==0))
  {
    Serial.println("subiendo");
    for(angulo = 0; angulo <= 90; angulo++)
  //sube
  {
    servo1.write(angulo);
    delay(25);
  }
  delay(2000);
}
if (angulo>0) //decrementa
{
  servo1.write( angulo--);
  delay(25);
}
}
```



7c. Servomotor controlado desde sensor flex.

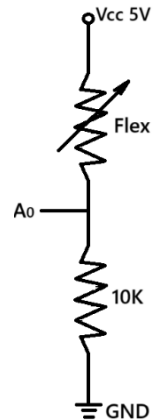
Sensor de flexión: Es un de sensor que actúa como una resistencia variable al variar la cantidad de curvatura o deformación. Se puede sustituir por un potenciómetro.

El conexionado eléctrico lo hemos hecho como un divisor de tensión entre 0 y 5V recogiendo su voltaje por A0.

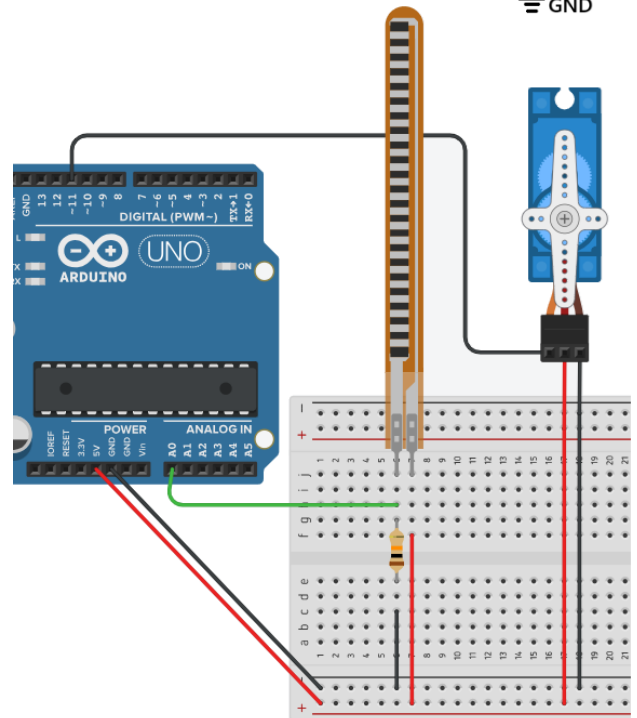
Cuando el sensor flexible está recto, su resistencia es baja y proporciona por la entrada analógica

(pin A0) un voltaje que se traducirá en una señal PWM en el pin de salida analógica 11 para girar el servomotor al ángulo deseado.

A medida que el sensor flexible se dobla lentamente, su resistencia aumenta. Por lo tanto, la entrada de voltaje analógico en el pin A0 también aumenta. y el ancho del pulso DWM de salida por el pin 11 será mayor y girará el servo. Para calibrar los valores de 0 a 180 ° utilizamos la función `map(señal, max1, min1,max2,min2)`.

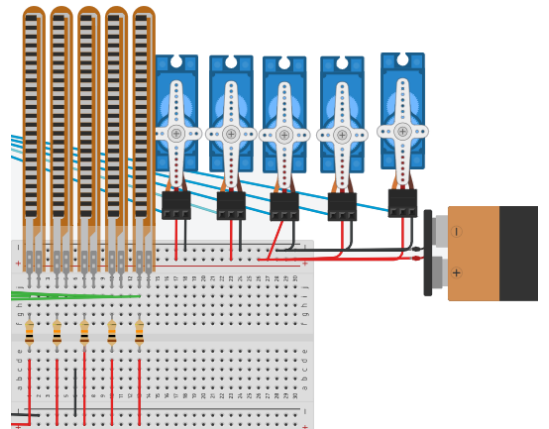


```
#include <Servo.h>
Servo servo1;
const int flexPin = A0;
void setup() {
  servo1.attach(11);
  Serial.begin(9600);
}
void loop() {
  int flexposition;
  int servoposition;
  flexposition=analogRead(flexPin);
  servoposition=map(flexposition, 256, 60, 0, 180);
  servoposition=constrain(servoposition, 0, 180);
  delay(100);
}
```



Mano robótica:

En el caso que quieras utilizar un guante para activar los sensores flex y 5 servomotores para mover la mano, Arduino no tendrás suficiente amperaje para los 5 servomotores. En tal caso puedes optar por utilizar una fuente de alimentación independiente y/o transistores como se explica en la siguiente lección.



Lección8: Transistor y motor.

Los pines de salida de Arduino permiten un máximo de 40 mA que no es suficiente para alimentar un motor de continua. Actuaremos a través de un transistor que nos regula y amplifica la señal de activación llamada corriente de base, como si de un grifo de presión se tratase, aumentando la señal entre el colector y el emisor.

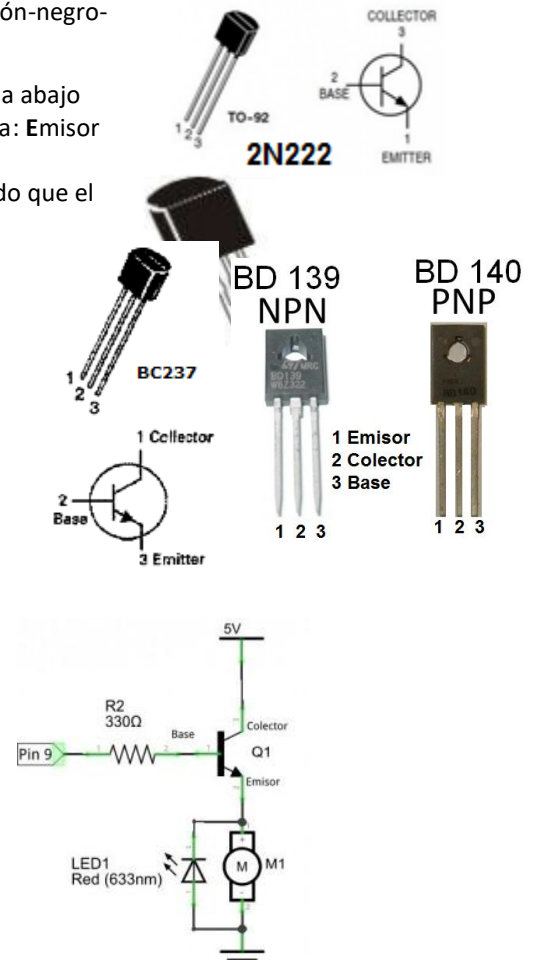
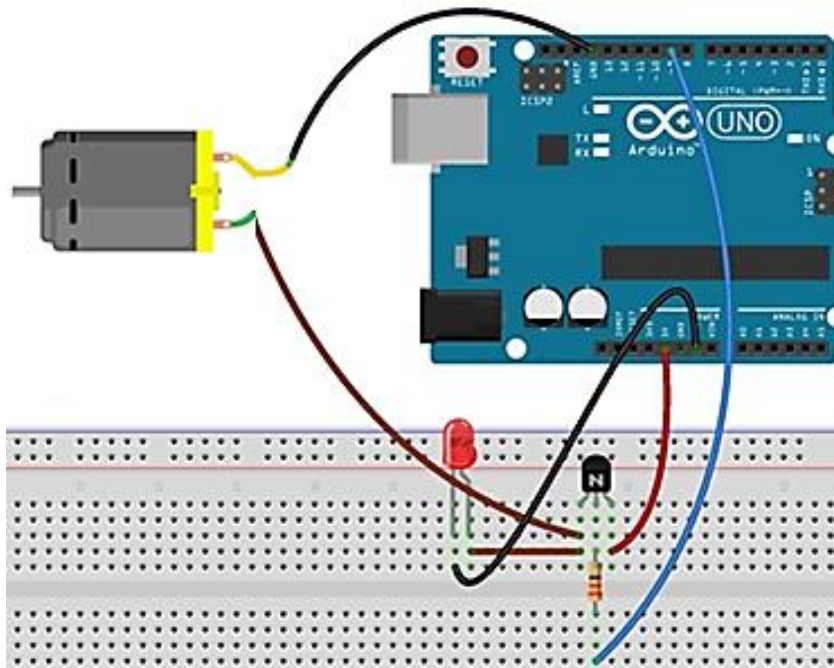
Transistor es un acrónimo del inglés: Transfer Resistor. Indica que la resistencia entre Emisor y Colector sea proporcional a la señal de control que inyectamos en la Base. Una ventaja de usar un transistor es que aísla eficazmente el circuito de control de la base, de la carga de salida del emisor.

Los transistores pueden ser del tipo PNP o NPN como los BD 139 y BD 140. La serie B son de silicio. BC para baja frecuencia baja potencia y serie BD para media.

La resistencia de base puede ser de entre 220 Ω (rojo-rojo-marrón) a 1 K (marrón-negro-rojo)

Para identificar cada pin del transistor, sostened el transistor con las patas hacia abajo mirando a la cara plana, donde esta rotulado el nombre. De izquierda a derecha: Emisor – Colector – Base o Colector – Emisor - Base

Mejor añadir un diodo al circuito en paralelo para que proteja el transistor, dado que el motor tiene carga inductiva de retorno.



Programas:

```
//Sin variar la velocidad:

const int control = 9 ; //constante entera

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

void loop()
{
  digitalWrite(control, HIGH); //salida digital
  delay(2000);
  digitalWrite(control, LOW);
  delay(1000);
}
```

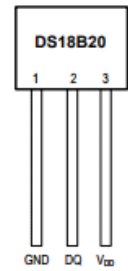
```
// variando la velocidad:

const int control = 9 ;
void setup()
{
  pinMode(control, OUTPUT) ;
}

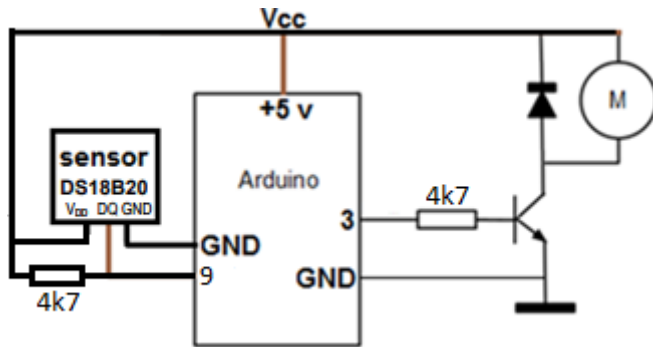
void loop()
{
  for ( int n = 0 ; n < 255 ; n++)
  {
    analogWrite (control, n) ; //salida analógica
    delay() ;
  }
}
```


Control de ventilador por sensor de temperatura digital

Esta práctica utiliza el sensor de temperatura digital DS18B20, ideal para la lectura de temperatura en líquidos por su encapsulado. Este sensor transmite los datos por el pin DQ en el protocolo 1-Wire. Puede medir temperaturas entre -55°C y 125°C. Permite la lectura de diferente precisión.



El transistor que nos regula y amplifica la señal de base es de media potencia DB137 o DB139 a través de una resistencia de 0,5 KΩ o de 470 Ω



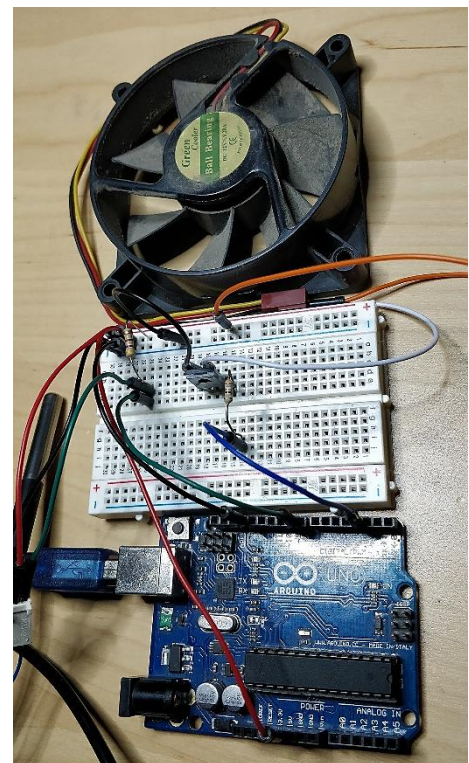
Modelo: BD137 / BD139
 Tipo: NPN
 Vmax: 60V / 80V
 Imax: 1500mA
 HFE min: 40

Programa: Debemos incluir las librerías OneWire (que implementa el protocolo 1-Wire) y DallasTemperature (para enviar los comandos adecuados), a través del Gestor de Librerías.

Instalar estas librerías desde el menú: Programa – Incluir librería – Gestor de bibliotecas.

```

#include <OneWire.h>
#include <DallasTemperature.h>
const int pinDatosDQ = 9; // Pin donde se conecta el bus 1-Wire
const int motor1 = 3; //pin sortida analogica motor 1
const int motor2 = 5; //pin sotida analogica motor 2
// Instancia a las clases OneWire y DallasTemperature
OneWire oneWireObjeto(pinDatosDQ);
DallasTemperature sensorDS18B20(&oneWireObjeto);
void setup()
{
  pinMode(motor1,OUTPUT); //sortida del motor
  Serial.begin(9600); // Iniciamos la comunicación serie
  sensorDS18B20.begin(); // Iniciamos el bus 1-Wire
}
void loop() {
  //----- lectura de sensores
  float temp0,temp1;
  int vMotor1,vmotor2;
  sensorDS18B20.requestTemperatures(); // Mandamos comandos
  temp0=sensorDS18B20.getTempCByIndex(0); // Leemos sensor 0
  temp1=sensorDS18B20.getTempCByIndex(1); // Leemos sensor 1
  Serial.print("Temperatura sensor 0: ");
  Serial.print(temp0);
  Serial.print(" - Temperatura sensor 1: ");
  Serial.print(temp1);
  Serial.print("\n");
  //----- control ventiladores -----
  vMotor1=temp0*5;
  analogWrite(motor1, vMotor1);
  Serial.print("V Motor 0: ");
  Serial.print(vMotor1);
  Serial.print("\n");
  delay(1000);
}
    
```



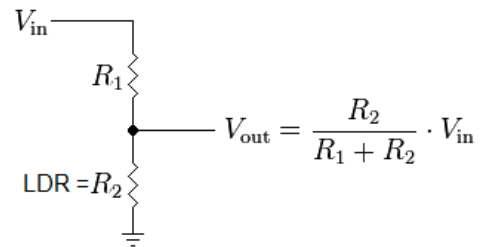
Lección10: seguidor de luz

Un fotoresistor, o LDR (light-dependent resistor) es un dispositivo cuya resistencia varía en función de la luz recibida. Su comportamiento es el siguiente: Mas luz = menor resistencia eléctrica. Menos luz = mayor resistencia eléctrica. Son de baja precisión, y varían con la temperatura.

Los valores típicos son de 1 MΩ en total oscuridad, a 50-100 Ω en luz brillante.

Si la entrada analógica lee de 0 a 5V en un rango de 1 a 1024.

Podemos utilizar como divisor de tensión una resistencia de entre 220 Ω a 500 Ω.



El ejemplo 1 emplea una entrada analógica para activar el LED integrado en la placa si supera un cierto umbral (threshold). El ejemplo 2 comprueba, a través del monitor serie, el valor de dos LDR.

```
//---- Test 1 sensor enciende el led 13 ----
const int LEDPin = 13;
const int LDRPin = A0;
const int umbral = 900;

void setup() {
  pinMode(LEDPin, OUTPUT);
  pinMode(LDRPin, INPUT);
}

void loop() {
  int input = analogRead(LDRPin);
  if (input > umbral) {
    digitalWrite(LEDPin, HIGH);
  }
  else {
    digitalWrite(LEDPin, LOW);
  }
}
```

```
// --- Medida valor 2 Sensores LDR por serial

void setup() {
  Serial.begin(9600);
}

void loop() {
  int sensor_iz = analogRead(A0);
  int sensor_der = analogRead(A1);
  Serial.print("\n\nIzquierdo: ");
  Serial.println(sensor_iz);
  Serial.print("Derecho: ");
  Serial.println(sensor_der);
  delay(700);
}
```

Ejemplo 3: Indicador de nivel Luminosos con 3 LEDs:

```
int pinLed1 = 2; // Pin donde se conectan los leds
int pinLed2 = 3;
int pinLed3 = 4;
int pinLDR = A0; // Pin de entrada para el LDR

int valorLDR = 0; // Variable para el valor del LDR

void setup()
{
  pinMode(pinLed1, OUTPUT); // Configuramos salidas
  pinMode(pinLed2, OUTPUT);
  pinMode(pinLed3, OUTPUT);
  Serial.begin(9600); // Configurar el puerto serial
}

void loop()
{
  digitalWrite(pinLed1, LOW); // Apagar los leds
  digitalWrite(pinLed2, LOW);
  digitalWrite(pinLed3, LOW);
```

```
valorLDR= analogRead(pinLDR); // Guardamos valor A0
Serial.println(valorLDR); //muestra en monitor ser.

// Encender los leds según el valor
if(valorLDR > 300)
{
  digitalWrite(pinLed1, HIGH);
}
if(valorLDR > 600)
{
  digitalWrite(pinLed2, HIGH);
}
if(valorLDR > 900)
{
  digitalWrite(pinLed3, HIGH);
}
delay(200); // Esperar antes de actualizar
```

Lección10b: seguidor de luz 2

Sensores: 2 fotorresistencias LDR como divisor de tensión con resistencia de 220 Ω

Actuadores: 2 Motores DC controlados por transistor BD139 con resistencia 1K de base + 2 diodos protección opcional

```
// prueba test de motores:
const int motor_iz = 3 ;
const int motor_der = 5 ;

void setup()
{
  pinMode(motor_iz, OUTPUT);
  pinMode(motor_der, OUTPUT);
}

void adelante()
{
  analogWrite(motor_iz,200) ; //salida anal. iz
  analogWrite(motor_der,200) ; //salida anal. der
}

void gira_izquierda()
{
  analogWrite(motor_iz,0) ; //salida anal. iz
  analogWrite(motor_der,200) ; //salida anal. der
}

void gira_derecha()
{
  analogWrite(motor_iz,200) ; //salida anal. iz
  analogWrite(motor_der,0) ; //salida anal. der
}

void para()
{
  analogWrite(motor_iz,0) ; //salida anal. iz
  analogWrite(motor_der,0) ; //salida anal. der
}

void loop()
{
  for ( int n = 0 ; n < 100 ; n++)
  {
    adelante();
    delay(1000);
    gira_izquierda();
    delay(1000);
    gira_derecha();
    delay(1000);
    para();
    delay(1000);
  }
}
```

```
// seguidor de luz:
const int motor_iz = 3 ;
const int motor_der = 5 ;
const int vel = 250; //velocidad motores
const int sens_umbral=990; //umbral de sensibilidad
void setup()
{
  Serial.begin(9600);
  pinMode(motor_iz, OUTPUT);
  pinMode(motor_der, OUTPUT);
}

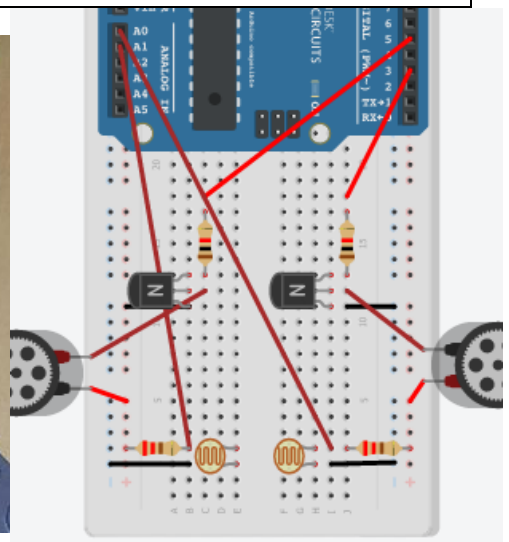
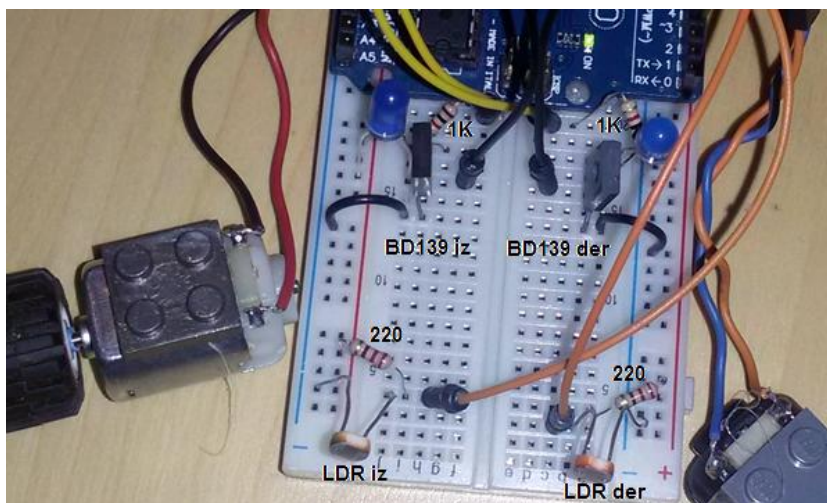
void adelante(int vel)
{
  analogWrite(motor_iz,vel) ; //salida iz
  analogWrite(motor_der,vel) ; //salida der
}

void gira_izquierda(int vel)
{
  analogWrite(motor_iz,0) ;
  analogWrite(motor_der,vel) ; //salida der
}

void gira_derecha(int vel)
{
  analogWrite(motor_iz,vel) ; //salida iz
  analogWrite(motor_der,0) ;
}

void para()
{
  analogWrite(motor_iz,0) ; //salida iz
  analogWrite(motor_der,0) ; //salida der
}

void loop()
{
  int sensor_iz = analogRead(A0) ;
  int sensor_der = analogRead(A1) ;
  Serial.println(sensor_iz);
  Serial.println(sensor_der);
  if ((sensor_iz<sens_umbral) &&
  (sensor_der<sens_umbral)) adelante(vel);
  else if (sensor_iz<sens_umbral) gira_izquierda(vel);
  else if (sensor_der<sens_umbral) gira_derecha(vel);
  else if ((sensor_iz>sens_umbral) &&
  (sensor_der>sens_umbral)) para();
  delay(300);
  para();
  delay(500);
}
```

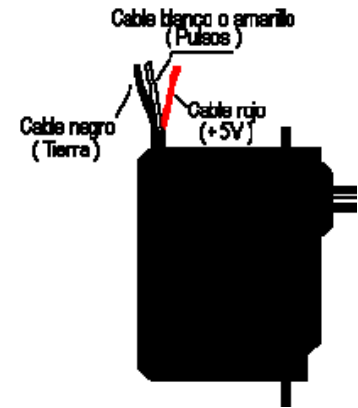


Lección11: Servomotores de 360°. Rotación continua.

Los servomotores de rotación continua giran 360°, mientras que los servos estándar sólo giran 180°. Éstos permiten controlar la dirección y velocidad de giro mediante pulsos analógicos (PWM) de 0 a 180. Se usa la librería *Servo.h*

El servomotor de rotación continua tiene 3 cables. Dos de alimentación (rojo (+), negro (-)), entre 5V y 7.5V, y uno de señal de control (blanco o amarillo), que se conecta al pin de la placa Arduino con el comando: `servo.attach(pin)`

La velocidad varía entre 0 y 180 ° siendo 90° en valor de paro central. Se envía con el comando `servo.write(velocidad)`. Probamos los siguientes programas de test:



```
//----- TEST 2 MOTORES -----
#include <Servo.h> //utiliza la librería para servos

void mueve_motores(int vel_a, int vel_b); // Función
Servo motor_izq; // Servo de la rueda izquierda
Servo motor_der; // Servo de la rueda derecha

void setup() //inicializacion
{
  motor_der.attach(9); // servo derecho a pin 9
  motor_izq.attach(10); // servo izquierdo a pin 10
}

void loop()
{
  mueve_motores(50,50); //adelante
  mueve_motores(0,0); //para
  mueve_motores(-50,-50); //atras
  mueve_motores(0,0); //para
}

//Función mover motores en el rango de -100 a 100
void mueve_motores(int vel_a, int vel_b)
{
  int tiempo=1000; //var tiempo establece a 1000
  motor_der.write(map(vel_a, -100, 100, 0, 180));
  motor_izq.write(map(vel_b, 100, -100, 0, 180));
  delay(tiempo);
}
```

Ejercicios de ampliación para control de un servo:

//----- 1 Servo regulado por potenciómetro -----

```
#include <Servo.h> // Incluir la librería Servo para usar attach y write
Servo servo1; // Crear un objeto del tipo o clase Servo llamado servo1
int angulo = 0 ;
void setup()
{
  servo1.attach(9) ; // Conectar servo1 al pin 9
}
void loop()
{
  angulo = map( analogRead(A0), 0, 1024, 0, 180); //lee por el pin A0
  servo1.write(angulo);
  delay(250);
}
```

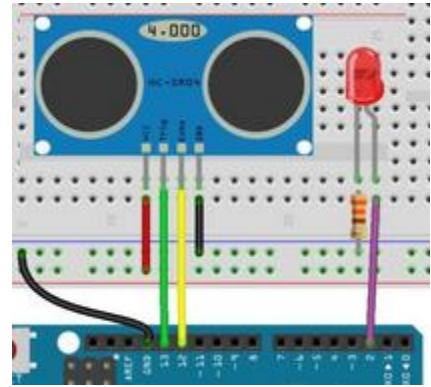
//----- 1 Servo regulado por Serial del PC -----

```
#include <Servo.h> // Incluimos la librería Servo
Servo servo_mot; // Creamos un objeto tipo Servo llamado servo_mot
int vel = 0 ; // variable entera para control de velocidad
void setup()
{
  servo_mot.attach(9) ; // Conectar cable servo de control al pin 9
  Serial.begin(9600); //activamos la comunicación serie con el PC
  Serial.println ("Q sube vel, Z baja vel, A para:");
}
void loop()
{
  int tecla= Serial.read(); // lee un caracter por el puerto serie
  if(tecla!= -1) // si es alguna tecla... (si no es ninguna, por defecto, vale -1)
  {
    switch(tecla) { //activa los posibles casos según el valor
      case'q': vel = vel+2; break; // si la tecla es q aumenta vel
      case'a': vel=0 ; break; // si la tecla es a para
      case'z': vel=vel-2; break; // si la tecla es z y vel>0 disminuye velocidad
    }
    servo_mot.write(map(vel, 0, 100, 90, 180)); //cambia valores rango 0 a 100 a angulo 90 a 180
    delay(20);
  }
}
```

Lección12: SENSOR ULTRASÓNICO DE DISTANCIA

Principio básico del sensor por ultrasonidos HC-SR04: Enviar pulsos ultrasónicos y escuchar el eco de retorno. Midiendo este tiempo, podemos calcular la distancia hasta el obstáculo. Para ello el HC-SR04 dispone de 4 pines; los dos de los extremos son de alimentación y los dos centrales son *trigger* y *echo* (el disparador y el receptor de señal).

Como el eco mide el tiempo que tarda el pulso en ir y venir la distancia recorrida será la mitad de lo que tarda la velocidad del sonido que es de 340 m/s. (También es posible utilizar una librería externa no incluida en el IDE).

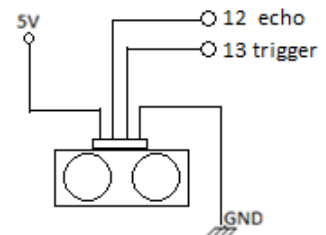


- ▶ Tiempo en seg./m = 1/340 = 0,00291
- ▶ Tiempo en Microseg/cm. = 29,1
- ▶ distancia = tiempo / (2*29,1) ida y vuelta o distancia = tiempo·340·10⁴ / 2

```
#define trig 13 //pin 13 al trigger
#define echo 12 //pin 12 al echo
#define led 2

void setup()
{
  Serial.begin (9600); // activamos monitor serie
  pinMode(trig, OUTPUT); //configuramos pines
  pinMode(echo, INPUT);
  pinMode(led, OUTPUT);
}

void loop()
{
  long duracion, distancia ; // variables de números grandes
  digitalWrite(trig, LOW); // Nos aseguramos de trigger está desactivado
  delayMicroseconds(2); // Para asegurarnos de que trigger esta LOW
  digitalWrite(trig, HIGH); // Activamos el pulso de salida
  delayMicroseconds(10); // Esperamos 10µs con el pulso activo
  digitalWrite(trig, LOW); // Cortamos el pulso y a esperar el echo
  duracion = pulseIn(echo, HIGH); // Tiempo en recibir el pulso de vuelta
  distancia = duracion / 2 / 29.1; // Convertimos tiempo a distancia
  if (distancia<200) Serial.println("Distancia:" +String(distancia) + " cm.");
  else Serial.println("Distancia >200 cm."); //monitor serie
  int Limite = 200 ; // Medida en vacío del sensor
  if ( distancia < Limite)
    digitalWrite ( led , HIGH) ; //encendemos el led, opcional: sirena()
  else
    digitalWrite( led , LOW) ; //apagamos el led
  delay (500) ; // Para limitar el número de mediciones
}
```



Ampliaciones propuestas:

Activar la función sirena() al encender el led:

```
//Conectar zumbador en pin 11 y
añadir pinMode(11, OUTPUT);

void sirena()
{
  tone(11, 2000, 100);
  delay(200);
  tone(11, 1500, 100);
  delay(200);
}
```



Abrir una barrera al encender el led:

```
//incluir <Servo.h> servo1.attach(9)
Void barrera(){
Serial.println("subiendo");
for(angulo = 0; angulo <= 90; angulo++) //sube
{
  servo1.write(angulo);
  delay(25);
}
delay(2000);
}
if (angulo>0) //decrementa
{
  servo1.write( angulo--);
  delay(25);
}}
```

Ampliaciones propuestas: Coche con sensor de ultrasonidos

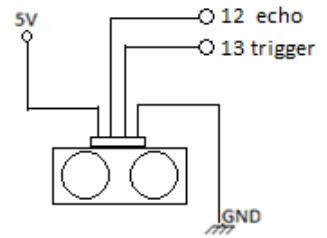
//----- MOTORES Y ULTRASONIDO -----

```
#include <Servo.h> //utiliza la librería para servos
void mueve_motores(int vel_a, int vel_b); // declara función
void cambiadireccion(); // declara función
void adelante(); // declara función
#define trig 13 //pin 13 al trigger
#define echo 12 //pin 12 al echo
Servo motor_izq; // Servo de la rueda izquierda
Servo motor_der; // Servo de la rueda derecha

void setup() //inicializacion
{
  Serial.begin(9600); // activamos monitor serie
  motor_der.attach(9); // servo derecho a pin 9
  motor_izq.attach(10); // servo izquierdo a pin 10
  pinMode(trig, OUTPUT); //pin 13 trigger salida
  pinMode(echo, INPUT); //pin 12 echo entrada
}
void loop()
{
  long duracion, distancia ; // variables de números grandes
  digitalWrite(trig, LOW); // Nos aseguramos de trigger está desactivado
  delayMicroseconds(2); // Para asegurarnos de que trigger esta LOW
  digitalWrite(trig, HIGH); // Activamos el pulso de salida
  delayMicroseconds(10); // Esperamos 10µs con el pulso activo
  digitalWrite(trig, LOW); // Cortamos el pulso y a esperar el echo
  duracion = pulseIn(echo, HIGH) ; // Tiempo en recibir el pulso de vuelta
  distancia = duracion / 2 / 29.1; // Convertimos tiempo a distancia
  if (distancia<200) Serial.println("Distancia:" +String(distancia) + " cm.");
  else Serial.println("Distancia >200 cm."); //monitor serie
  int Limite = 50 ; // distancia mínima
  if ( distancia < Limite)
    cambiadireccion(); //llama a función cambia direccion
  else
    adelante(); //llama a función adelante
  delay(500) ; // Para limitar el número de mediciones
}

//-----funciones -----
void adelante()
{
  mueve_motores(50,50); //llama a función mueve_motores: avanza a velocidad 50 derecha y 50 izquierda
}
void cambiadireccion()
{
  mueve_motores(0,0); //para
  mueve_motores(50,-10); //gira a la derecha
}

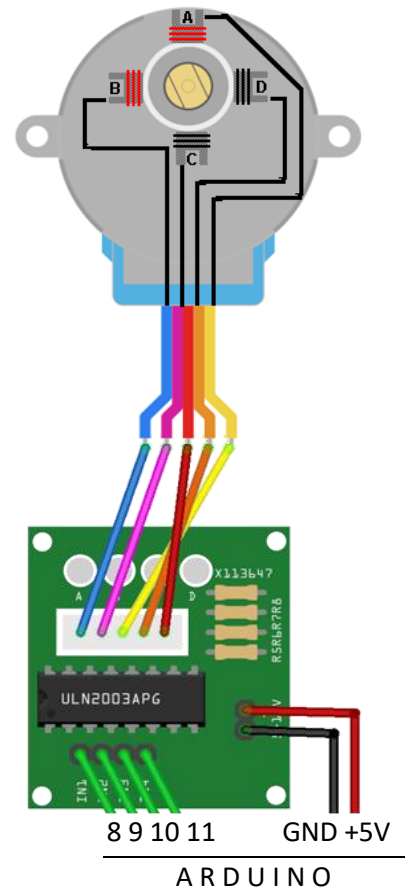
void mueve_motores(int vel_a, int vel_b)
{
  int tiempo=1000; //var tiempo establece a 1000
  motor_der.write(map(vel_a, -100, 100, 0, 180));
  motor_izq.write(map(vel_b, 100, -100, 0, 180));
  delay(tiempo);
}
```



Motor paso a paso

Hoja de características básicas 28BYJ-48:

- Motor paso a paso con 5 cables (4 bobinas unipolar A-B-C-D)
- Voltaje de funcionamiento 5V o 12V
- Este tipo de motores lleva un driver o circuito de potencia con:
 - 4 LEDs que indican cuando se excita una bobina
 - 4 resistencias para proteger los LEDs
 - Chip ULN2003 que contiene 3 transistores Darlington
 - 4 entradas para el controlador, por donde entran los pulsos.
 - Jumpers para seleccionar el voltaje de funcionamiento (5V o 12V)
 - Hay que puentear el voltaje que no se utiliza
- Cada paso avanza 5,625°
- Caja reductora mediante engranajes 1/64
- Se consigue un paso de $5,625/64 = 0,088^\circ$
- Resistencia del bobinado de 50 Ω
- Torque de 34 Newtons · metro = ± 35 gramos · cm
- Frecuencia máxima 100Hz que equivale a un delay de 10 ms



Mover el motor sin librerías:

```
digitalWrite(9, HIGH);
digitalWrite(8, LOW);
delay(retardo);
digitalWrite(11, HIGH);
digitalWrite(10, HIGH);
digitalWrite(9, LOW);
digitalWrite(8, LOW);
delay(retardo);
digitalWrite(11, HIGH);
digitalWrite(10, LOW);
digitalWrite(9, LOW);
digitalWrite(8, HIGH);
delay(retardo);
```

Paso Doble			
1	1	0	0
0	1	1	0
0	0	1	1
1	0	0	1

```

//--- modo con array ---
#define IN1 8
#define IN2 9
#define IN3 10
#define IN4 11
int paso [4][4] =
{
  {1, 1, 0, 0},
  {0, 1, 1, 0},
  {0, 0, 1, 1},
  {1, 0, 0, 1}
};

void setup()
{
  // Todos los pines en modo salida
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
}
void loop()
{
  for (int i = 0; i < 4; i++)
  {
    digitalWrite(IN1, paso[i][0]);
    digitalWrite(IN2, paso[i][1]);
    digitalWrite(IN3, paso[i][2]);
    digitalWrite(IN4, paso[i][3]);
    delay(10);
  }
}
    
```

Mover el motor con librería:

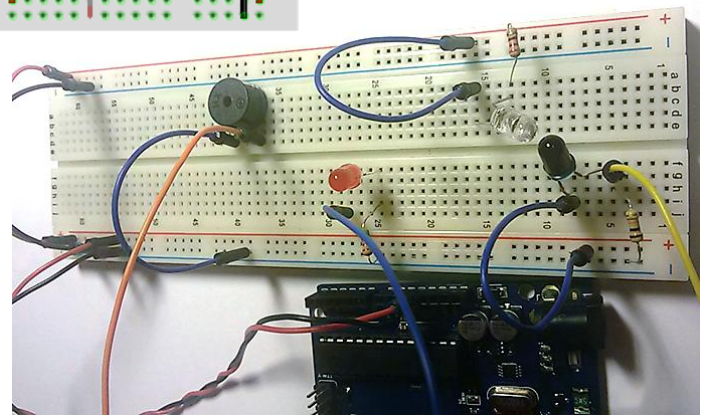
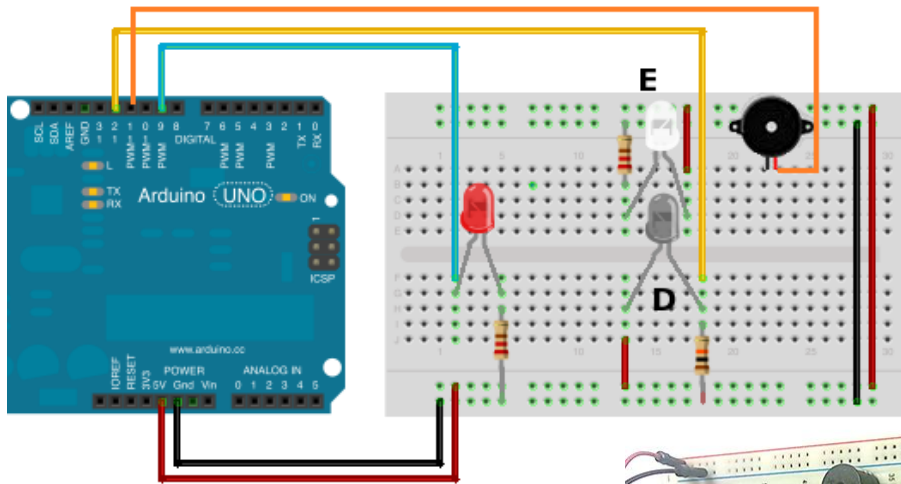
```

#include <Stepper.h>
#define STEPS 2048
Stepper stepper(STEPS, 8, 9, 10, 11);
void setup() {
  stepper.setSpeed(10);
}

void loop() {
  stepper.step(2048);
}
    
```

Lección13: SENSOR OPTOACOPLADOR POR INFRAROJOS CON BUZZER

Principio básico del opto acoplador basado en dos componentes: un LED emisor de luz infrarroja y un transistor foto receptor. Necesitaremos un opto interruptor infrarrojo ITR8102 - 2 resistencias de 220 Ω (rojo, rojo, café) 1 resistencia de 10 K Ω (café, negro, naranja) y un LED y un zumbador (buzzer).



Programa:

```
const int Sensor = 12;    // numero pin del sensor
const int Led     = 9;    // numero pin del led
const int audio   = 11;  // numero pin del zumbador
int estadoSensor = 0;    // variable para leer y guardar
el estado del sensor

void setup() {

    pinMode(Led, OUTPUT); // inicializa pin led como salida
    pinMode(Sensor, INPUT); // inicializa pin sensor como entrada
    pinMode(audio, OUTPUT); // inicializa pin buzzer led como salida
}

void loop(){
    estadoSensor = digitalRead(Sensor); // lee estado del sensor
    if (estadoSensor == HIGH) { // HIGH, sensor activado
        digitalWrite(Led, HIGH);
        sirena();
    }
    else {
        digitalWrite(Led, LOW); // LOW, sensor desactivado
    }
}

void sirena(){
    tone(audio, 2000, 100);
    delay(200);
    tone(audio, 1500, 100);
    delay(200);
}
```

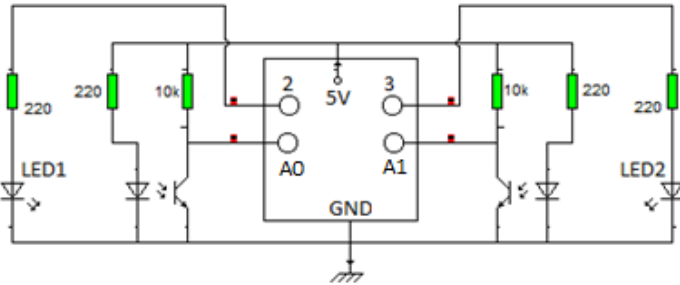
Lección12: SEGUIDOR DE LINEAS CON SERVOMOTORES y Paro por ultrasonido

Hemos adaptado el sensor opto acoplador del ejercicio anterior con el control de dos servomotores, que tienen más precisión que los motores de continua.

Primera parte: Sensores:

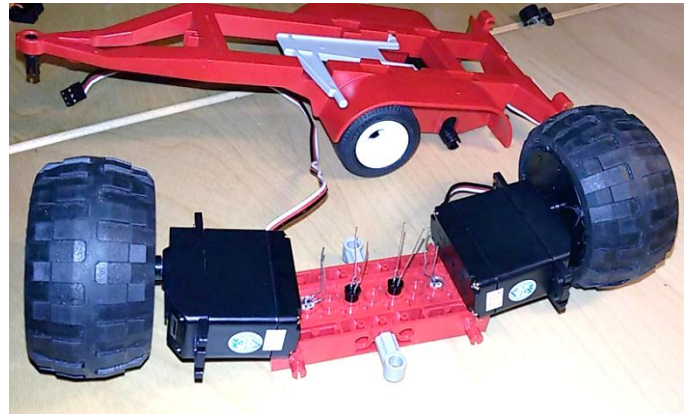
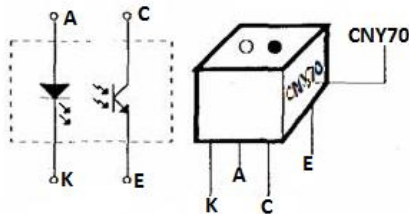
Circuito sensores infrarrojos

Corresponde a la parte de la detección de los sensores opto-acopladores (un par de LEDs emisores de luz infrarroja y dos transistores foto-receptores) y la activación de dos LEDs como testigos de su funcionamiento.



Conectamos el colector de los transistores optoacopladores a Vcc a través de una resistencia de 10KΩ y a las entradas analógicas A0 y A1 para recibir valores entre 0 y 1024 y los emisores a GND.

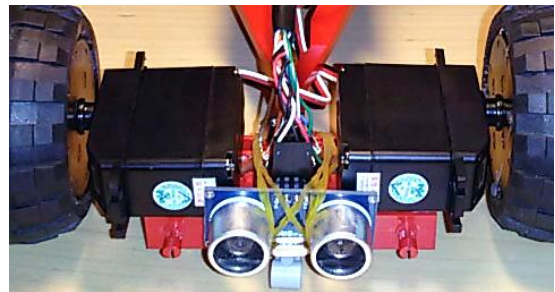
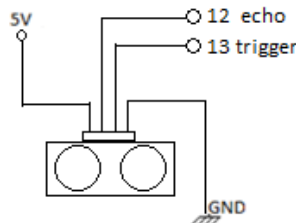
Si utilizas el sensor CNY70:



Circuito sensor ultrasonidos

HC-SR04 de 4 pines: los dos de los extremos, de alimentación, conectados a +5Vcc y GND y los dos centrales: el disparador, *trigger* conectado al pin digital 13 en modo output y el receptor, echo conectado al pin 12 en modo input.

En caso detección de obstáculo por el ultrasonido hemos asociado un buzzer de alarma conectad al pin 9 y a GND y activamos también la intermitencia de ambos LEDs de testigo anteriores.



Test1. Sólo con LEDS y optos

```
const int infra_iz=A0; //pin Infrarojo izquierdo
const int infra_der=A1; //pin Infrarojo derecho
const int led_iz=2; //pin LED izquierdo
const int led_der=3; //pin LED derecho
const int negro=700; //limite para el negro ajustar en monit serie
int medida_infra_iz = 0; // variable para estado del sensor
int medida_infra_der = 0; // variable para estado del sensor
int estado_infra_iz = 0; // variable para estado del sensor
int estado_infra_der = 0; // variable para estado del sensor
//-----
void setup() {
  pinMode(led_iz, OUTPUT); // inicializa el pin como salida
  pinMode(led_der, OUTPUT); // inicializa el pin como salida
  Serial.begin (9600); //inicia puerto serie para comprobaciones
}
//-----
void loop(){
  medida_infra_iz=analogRead(infra_iz); //lee y guarda valor infra_iz
  medida_infra_der=analogRead(infra_der); // lee y guarda infra_der
  if (medida_infra_iz>negro) {estado_infra_iz = 1;} //estado iz
  else {estado_infra_iz = 0;}
  if (medida_infra_der>negro){estado_infra_der = 1;} // estado der
  else {estado_infra_der = 0;}
  Serial.println("Izdo / dcho: " + String(medida_infra_iz)+ " / " +
  String(medida_infra_der));
}
```

```
if ((estado_infra_iz == 1) && (estado_infra_der == 1))
{ // ambos sensores activados
  digitalWrite(led_iz, 1); //enciende LED testigo izdo
  digitalWrite(led_der, 1); //enciende LED testigo dere
  //adelante();
}
else if (estado_infra_iz == 1)
{
  digitalWrite(led_iz, 1); //enciende el LED testigo izdo
  digitalWrite(led_der, 0); //apaga LED testigo dere
  //gira_iz();
}
else if (estado_infra_der == 1) // si infra derecho activado
{
  digitalWrite(led_iz, 0); //apaga el LED testigo izdo
  digitalWrite(led_der, 1); //enciende el LED testigo dere
  //gira_der(); //gira derecha
}
else //los dos apagados
{
  digitalWrite(led_iz, 0); //apaga el LED testigo dcho
  digitalWrite(led_der, 0); //apaga el LED testigo dcho
  //para();
}
}
```

Ejer completo con 2 optos, 1 infra y 2 servos

```
#include <Servo.h> //usa la librería para servos
Servo M_Iz; // Servo de la rueda izquierda (pin PWM 10)
Servo M_Der; // Servo de la rueda derecha (pin PWM 11)
const int tiempo_delay=90;
const int M_vel=40; //velocidad de los motores
const int buzzer=9; //pin zumbador buzzer
const int ultra_echo=12; //pin ultrasonidos Echo
const int ultra_trigger=13; //pin ultrasonidos Trigger
const int infra_iz=A0; //pin Infrarojo izquierdo
const int infra_der=A1; //pin Infrarojo derecho
const int test_iz=2; //pin LED izquierdo
const int test_der=3; //pin LED derecho
const int negro=450; //valor limite para el negro
const int dist_obstaculo=10; // distancia a un obstaculo
int medida_infra_iz = 0; // variable para estado del sensor
int medida_infra_der = 0; // variable para estado del sensor
int estado_infra_iz = 0; // variable para estado del sensor
int estado_infra_der = 0; // variable para estado del sensor
int estado_ultra = 0; // variable para estado del sensor
//-----
void setup() {
  M_Iz.attach(10); // Asocia servo derecho a salida PWM 10
  M_Der.attach(11); // Asocia servo izquierdo a salida PWM 11
  pinMode(test_iz, OUTPUT); // inicializa el pin como salida
  pinMode(test_der, OUTPUT); // inicializa el pin como salida
  pinMode(buzzer, OUTPUT); // inicializa el pin como salida
  pinMode(ultra_trigger, OUTPUT); // inicializa el pin como entrada
  pinMode(ultra_echo, INPUT); // inicializa el pin como entrada
  Serial.begin(9600); //inicilaiza puerto serie para comprobaciones
}
//-----
void loop(){
  medida_infra_iz=analogRead(infra_iz); // lee el valor infra_iz
  y lo guarda
  medida_infra_der=analogRead(infra_der); // lee el valor
  infra_der y lo guarda
  if (medida_infra_iz>negro) {estado_infra_iz = 1;}
  //activa el valor del estado iz
  else {estado_infra_iz = 0;}
  if (medida_infra_der>negro){estado_infra_der = 1;} // activa el
  valor del estado der
  else {estado_infra_der = 0;}
  Serial.println("Izdo / dcho: " + String(medida_infra_iz)+ " / "
+ String(medida_infra_der));
  comprueba_obstaculos();
  if (estado_ultra == 1) {para();}
  //para si sensor detecta obstaculo
  else
  {
  if (medida_infra_iz>medida_infra_der) {gira_der();}
  else {gira_iz();} //busca el lado oscuro
  if ((estado_infra_iz == 1) && (estado_infra_der == 1))
  { // ambos sensores activados
  digitalWrite(test_iz, 1); //enciende LED testigo izdo
  digitalWrite(test_der, 1); //enciende LED testigo dere
  adelante();
  }
  else
  {
  digitalWrite(test_iz, 0); //apaga LED testigo izdo
  digitalWrite(test_der, 0); //apaga LED testigo dere
  if (estado_infra_iz == 1)
  {
  digitalWrite(test_iz, 1); //enciende el LED testigo izdo
  gira_iz();
  }
  else if (estado_infra_der == 1)
  // si sensor IR derecho activado
  {digitalWrite(test_iz, 0); //apaga el LED testigo izdo
  digitalWrite(test_der, 1); //enciende el LED testigo dere
  gira_der(); //gira derecha
  }
  else
  {
  if (medida_infra_iz>medida_infra_der) {gira_der();}
  else {gira_iz();} //busca el lado oscuro
  digitalWrite(test_iz, 0); //apaga el LED testigo dcho
  digitalWrite(test_der, 0); //apaga el LED testigo dcho
  para();
  }
  }
}
//-----
void sirena(){
  tone(9, 2000, 100);
  digitalWrite(test_iz, 1); //enciende el LED testigo izdo
  digitalWrite(test_der, 1); //enciende el LED testigo dere
  delay(200);
  tone(9, 1500, 100);
  digitalWrite(test_iz, 0); //enciende el LED testigo izdo
  digitalWrite(test_der, 0); //enciende el LED testigo dere
  delay(200);
}
//-----
void comprueba_obstaculos()
{ long duracion, distancia ;
  digitalWrite(ultra_trigger, LOW);
  // Nos aseguramos de trigger está desactivado
  delayMicroseconds(2);
  // Para asegurarnos de que trigger esta LOW
  digitalWrite(ultra_trigger, HIGH);
  // Activamos un pulso de salida
  delayMicroseconds(10);
  // Esperamos 10micros con el pulso activo
  digitalWrite(ultra_trigger, LOW);
  // Cortamos el pulso y a esperar el echo
  duracion = pulseIn(ultra_echo, HIGH) ;
  distancia = duracion / 2 / 29.1 ;
  Serial.println("Distancia: " + String(distancia) + " cm.");
  if ( distancia < dist_obstaculo){
  estado_ultra = 1;
  sirena();
  delay (100);} // Para limitar número de mediciones
  else {estado_ultra = 0;}
}
//-----
void gira_iz() //gira motor derecho y para el izquierdo
{M_Der.write(90-M_vel);
  M_Iz.write(92); //90 es posición central?
  delay(tiempo_delay);
}
void gira_der() //gira motor izdo y para el derecho
{M_Der.write(90);
  M_Iz.write(92+M_vel);
  delay(tiempo_delay);
}
void para() //para los dos motores
{M_Der.write(90);
  M_Iz.write(92);
  delay(tiempo_delay);
}
void adelante() //ambos motores
{M_Der.write(90-M_vel);
  M_Iz.write(92+M_vel);
  delay(tiempo_delay);
}
```

Lección13: Comunicación por cristal líquido LCD

Demuestra el uso de una pantalla LCD de 16x2. La biblioteca LiquidCrystal funciona con todas las pantallas LCD que son compatibles con el controlador Hitachi HD44780 como el QC1602A

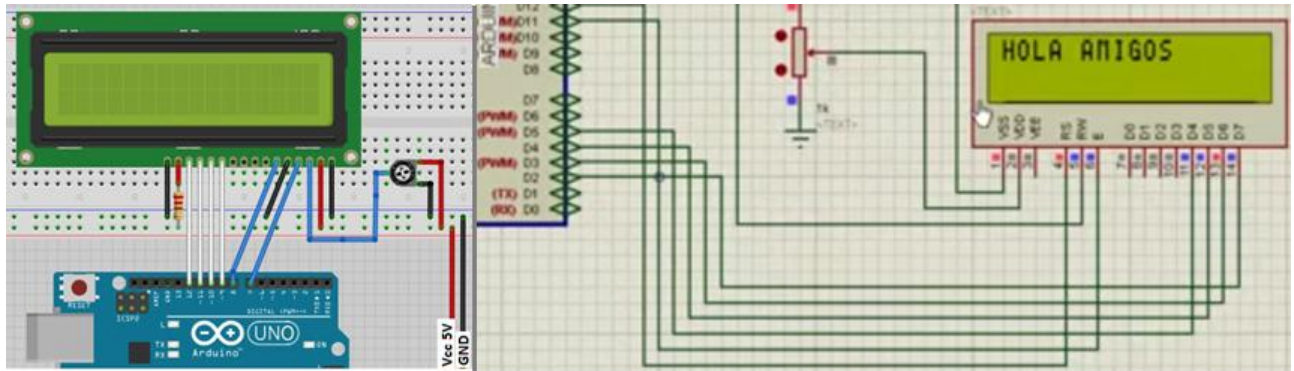


Tabla de conexiones:

- **Conexiones:** El potenciómetro 4K7Ω se utiliza para ajustar el contraste. Conectar un lado a GND y el lado opuesto a + 5V. A continuación, conecte el pasador central al pin 3 en la pantalla LCD.
- **Prueba del código:** Abre el Arduino IDE y selecciona: Archivo -> Ejemplos -> LiquidCrystal -> HelloWorld

En la librería: #include <LiquidCrystal.h>

Utilizamos:

```
lcd.begin(16, 2); //inicia en col 16 fila 2
lcd.setCursor(0, 1); //pon en col 0 fila 1
lcd.print("Ofimega academia");
```

LCD Pin	Símbolo Función	Arduino Pin
1	Vss: tierra (0 V) a GND	GND
2	Vdd: alimentación (4,5 - 5,5 V)	+ 5V
3	Vo: Ajuste de contraste a potenciómetro	Pot. Pin Medio
4	Rs: registro señal de selección	12
5	R / W señal de lectura / escritura	GND
6	E: enable - señal de habilitación	1
11	DB4 bus de datos - modo de 4 bits	5
12	DB5 bus de datos - modo de bits	4
13	DB6 bus de datos - modo de 4 bits	3
14	DB7 bus de datos- modo de 4 bits	2
15	A 'Back Light' ánodo – R220Ω o directo a +5V	+ 5V
16	K Cátodo 'Back Light' luz trasera	GND

Ejercicio propuesto: Medir la distancia por el ultrasonidos o la temperatura y mostrarla en el cristal líquido.



Lección 14a: Comunicación serie con PC mediante USB y el IDE Processing.

El programa enciende el LED 13 de Arduino pero es controlado por el PC a través del puerto serie. Existen dos métodos para controlar *Arduino* desde *Processing*:

1. Mediante la Librería Arduino para Processing
2. Directamente mediante la lectura/escritura de datos a través de un puerto (serie o Bluetooth).

En este caso utilizaremos el método de control, vía puerto serie, el encendido y apagado de un LED al pasar el mouse sobre un recuadro en la ventana del PC). Este ejercicio está adaptado de un ejemplo de Processing. Como podemos utilizar el LED 13 interno, no necesitamos cableado en Arduino.

1º Descargamos el IDE de processing desde <https://processing.org/download/> si aún no lo tenemos.

2º Configuramos Processing para serial: <http://processing.org/reference/libraries/serial/>

Programa para Processing (Véase introducción a Processing)

```
/******  
* Comprobar si el ratón está sobre un rectángulo y escribe el estado del puerto serie.  
* Este ejemplo comunica con Arduino por el cable serie USB.  
*****/  
  
import processing.serial.*;  
Serial elPuerto; // Crea el objeto de la clase Serial llamado elPuerto  
int val; // Variable entera  
  
void setup()  
{  
  size(200, 200);  
  String nombre = Serial.list()[0]; // Serial.list()[0] -> abre el primer puerto serie que  
                                     encuentra, empieza normalmente en el COM1  
  elPuerto = new Serial(this, nombre, 9600);  
}  
  
void draw() { //Dibuja la ventana de interacción en modo bucle loop  
  background(255); //fondo de la ventana en blanco  
  if (mouseOverRect() == true) { // si la función dice que el ratón está dentro...  
    fill(204); // cambia relleno a color gris y  
    elPuerto.write('H'); // manda el caracter H por el puerto serie  
  }  
  else { // Si el ratón no está dentro,  
    fill(0); // cambia a color negro  
    elPuerto.write('L'); // y manda el letra L por serie  
  }  
  rect(50, 50, 100, 100); // Dibuja el recuadro  
}  
  
boolean mouseOverRect() { // la función devuelve true si está dentro del area  
  if ((mouseX >= 50) && (mouseX <= 150) && (mouseY >= 50) && (mouseY <= 150)) return true;  
  else return false;}
```

Programa para Arduino

```
char val; // Data received from the serial port  
int ledPin = 13; // Set the pin to digital I/O 4  
void setup() {  
  pinMode(ledPin, OUTPUT); // Set pin as OUTPUT  
  Serial.begin(9600); // Start serial communication at 9600 bps  
}  
void loop() {  
  while (Serial.available()) { // If data is available to read,  
    val = Serial.read(); // read it and store it in val  
  }  
  if (val == 'H') { // If H was received  
    digitalWrite(ledPin, HIGH); // turn the LED on  
  } else {  
    digitalWrite(ledPin, LOW); // Otherwise turn it OFF  
  }  
  delay(100); // Wait 100 milliseconds for next reading  
}
```

Lección 14b: Comunicación serie teclado PC mediante USB y el IDE Processing.

Control de tres LEDs desde las teclas **1,2 y 3**

El programa que controla y monitoriza los datos de Arduino estará en el IDE Processing.

Existen dos métodos para controlar Arduino desde processing:

3. Mediante la Librería Arduino para Processing
4. Directamente mediante la lectura/escritura de datos a través de un puerto (serie o Bluetooth).

En este caso utilizaremos el método de control, vía puerto serie, el encendido y apagado de cualquiera de los tres LEDs conectados a las salidas digitales PIN13, PIN12, PIN11 mediante las teclas “1”, “2” y “3” respectivamente actuando estas en modo *biestable* (una pulsación enciende la siguiente pulsación apaga)



1º Descargamos el IDE de procesing desde <https://processing.org/download/> si aún no lo tenemos.

2º Configuramos Processing para serial: <http://processing.org/reference/libraries/serial/>

Programa para Processing

```
import processing.serial.*;    //Importamos la librería
Serial elPuerto;
void setup() {
  size(200, 200); //tamaño de la ventana de 200x200 pixels
  elPuerto = new Serial(this, Serial.list()[0], 9600);
}
void draw() //Dibuja la ventana de interacción en modo bucle loop
{
  background(0); //color de fondo negro en modo RGB
}
void keyReleased() { //Desencadena el evento tecla liberada (o keyPressed() la pulsada)
  elPuerto.write(key); //Manda al puerto el valor de la tecla
  background(255); //fondo color blanco en modo RGB
}
//-> la función keyReleased () se llama una vez cada vez que se suelta una tecla.
//-> La variable key contiene el valor de la tecla en código ASCII.
//-> Véase: https://processing.org/reference/
```

Programa para Arduino

```
int ledPin1= 13, ledPin2= 12, ledPin3= 11;
int status1 = HIGH, status2 = HIGH, status3 = HIGH;
int val;
void setup() {
  Serial.begin(9600);
  pinMode(ledPin1, OUTPUT);
  pinMode(ledPin2, OUTPUT);
  pinMode(ledPin3, OUTPUT);
}
void loop(){
  val= Serial.read(); // lee el valor del puerto
  if(val!= -1) // si es alguna tecla... (si no es ninguna, por defecto, vale -1)
  {
    switch(val) { //activa los posibles casos según el valor
      case'1': status1 = !status1; break; // si la tecla es 1 cambia la variable flag estado1
      case'2': status2 = !status2; break; // si la tecla es 2 cambia la variable flag estado2
      case'3': status3 = !status3; break; // si la tecla es 3 cambia la variable flag estado3
    }
  }
  digitalWrite(ledPin1, status1); //saca por el pin 13 el nivel del estado1
  digitalWrite(ledPin2, status2); //saca por el pin 12 el nivel del estado2
  digitalWrite(ledPin3, status3); //saca por el pin 11 el nivel del estado3
}
```

Programa para Arduino para control de servo por teclado

Utilizando el programa en processing para contrlar por teclado dos servos que se encargan de la dirección y el avance de un coche eléctrico:

```
// CONTROL VEHICULO 2 SERVOS POR TECLADO - PROGRAMA PARA ARDUINO
//===== TESTED OK =====
#include <Servo.h> // Incluir la librería Servo
//---- objetos -----
Servo servo_dir; // Creo objeto tipo Servo para servo dirección
Servo servo_mot; // Creo objeto tipo Servo para servo motriz o motor
//---- variables -----
int angulo ; //ángulo giro ruedas
int vel; //velocidad de avance, empieza en 0
int valor_tecla=0; //tecla de envío por teclado
// ---- Inicio puesta en marcha ----
void setup(){
Serial.begin(9600); //iniciamos comunicación serie con USB del PC
servo_dir.attach(9) ; // Conectar servo1 al pin 9
servo_mot.attach(10) ; // Conectar servo motriz al pin 10
angulo=90; //inicializa angulo
vel=0;
servo_dir.write(angulo);
servo_mot.write(90+vel);
}
void loop()
{
valor_tecla= Serial.read(); // lee el valor del puerto
if(valor_tecla!= -1) // si es alguna tecla... (si no es ninguna, por defecto, vale -1)
{
switch(valor_tecla) { //activa los posibles casos según el valor
case'1': angulo = angulo+10; break; // si la tecla es 1 cambia la variable flag estado1
case'2': angulo = 90; break; // si la t1111tecla es 2 cambia la variable flag estado2
case'3': angulo = angulo-10; break; // si la tecla es 3 cambia la variable flag estado3
case'q': vel = vel+2; break; // si la tecla es q aumenta vel
case'a': vel=0 ; break; // si la tecla es a para
case'z': if (vel>0) {vel=vel-2;}; break; // si la tecla es z y vel>0 disminuye velocidad
}
servo_dir.write(angulo);
servo_mot.write(map(vel, 0, 100, 90, 180)); //cambia valores rango 0 a 100 a angulo 90 a 180
delay(20);
}
}

//SKETCH PARA PROCESSING ENVIO TECLA
//----- tested ok -----
import processing.serial.*; //Importamos la librería
Serial elPuerto;
void setup()
{
size(200, 200); //tamaño de la ventana de 200x200 pixels
elPuerto = new Serial(this, Serial.list()[0], 9600);
}
void draw() //Dibuja la ventana de interacción en modo bucle loop
{
background(0); //color de fondo negro en modo RGB
}
void keyReleased()
{ //Desencadena el evento tecla liberada (o keyPressed() la pulsada)
elPuerto.write(key); //Manda al puerto el valor de la tecla
background(255); //fondo color blanco en modo RGB
delay(20);
}
//-> la función keyReleased () se llama una vez cada vez que se suelta una tecla.
//-> La variable key contiene el valor de la tecla en código ASCII o su carácter equivalente entre apóstrofes
//-> Véase: https://processing.org/reference/
```

Lección15: Comunicación serie con móvil mediante Bluetooth.

Bluetooth: Transmisión inalámbrica, con nombre de rey noruego, a 2,4 GHz. Puede ser Master o Slave como el módulo HC5 o solo slave como el HC6 (+ económico).

Modelos: HC6 esclavo sólo puede conectarse a un master.

HC-05 es un master BlueTooth y puede conectarse a varios Slave.

Módulo JY-MCU: Los dispositivos de este tipo tienen que “emparejarse” y tienen que compartir una contraseña para que los datos puedan intercambiarse. Por defecto, estos módulos tienen la contraseña 1234, aunque tanto esto como el nombre, pueden ser actualizados mediante unos comandos especiales, llamados AT.

Conexión al Arduino:

- Módulo HC6: tiene 4 patillas. 2 para la alimentación y 2 para la comunicación Rx Tx
- Módulo HC5: tiene 6 pines.

Transmisión: Patillas 1 y 2. La transmisión de datos (TX) del bluetooth conectada a la (RX) en el arduino, y viceversa.

- pin digital 0: RX <- (arduino recibe a través de este pin)
- pin digital 1: TX -> (arduino envía a través de este pin)

Alimentación: Patillas 3 y 4. Aunque el módulo funciona a 3.3v, normalmente las placas comerciales, llevan un regulador que permite los 5v de arduino Vcc y a masa GND el negativo.

Funcionamiento: Al conectar el módulo, este empezará a parpadear indicando que está buscando un master.

Configuración: Enviar por serie los códigos AT para configurar la velocidad, nombre del dispositivo o clave.

El módulo HC-05 dispone de un pulsador para entrar en este modo. (Se mantiene pulsado al iniciar alimentación)

Comandos básicos AT: (Escribirlos en mayúsculas)

AT+VERSION: Pide la versión del Firmware AT+ROLE: en HC-05 para ver el modo master o slave

AT+NAMEXXX o AT+NAME=XXX : Cambia el nombre de identificación del módulo: AT+NAME=OFIMEGA

AT+BAUDX para HC-06 ó AT+UART para HC-05: Fija la velocidad de comunicación según la tabla:

1 configura	1200bps	5 configura	19200bps
2 configura	2400bps	6 configura	38400bps
3 configura	4800bps	7 configura	57600bps
4 configura	9600bps (Default)	8 configura	115200bps

Ejemplo: AT+BAUD7 configura la comunicación a 57600 baudios

AT+PINXXXX, configura número de acceso para la vinculación en HC=6 ó AT+PSWD: ver el password en HC-05

Ejemplo: AT+PIN1234 ó AT+PSWD=1234 para HC-05, establece 1234 como PIN

AT+ORGL: Restablece los valores de fábrica

Ejemplo de envío de comandos AT: (siempre en mayúsculas):

```
Serial.begin(9600); //CONEXION SERIE USB CON ORDENADOR
Serial1.begin(9600); //CONEXION SERIE PARA EL MODULO BT
Serial.println("Terminal para configurar BT(JY-MCU)");
Serial.println("Comandos AT. USA MAYUSCULAS");
Serial.println("Comando, Respuesta, Parametros");
Serial.println("AT, OK, --Verifica la conexión--");
Serial.println("AT+VERSION, --Devuelve la version--");
Serial.println("AT+BAUDx, OKxxxx, Set x to: 1=1200 2=2400 3=4800 4=9600 5=19200 6=38400
7=57600 8=115200 -para cambiar la velocidad--");
Serial.println("AT+NAMEstring, nombrenuevo (20max)"); //NAMEnombre todojunto
Serial.println("AT+PINxxxx, Cambia el pin (1234 por defecto)");
Serial.println("AT+ROLEx,1=MASTER/0=SLAVE --SOLO MASTER");
```

Ejercicio 1.

Aunque es posible conectar los pines RX y Tx a los equivalentes de Arduino en los pines 0 y 1 digitales, sin más que cruzarlos (BT Tx a Arduino Rx y BT Rx a Aduino Tx), para evitar interferencias con la comunicación serie de Arduino con el PC a través del USB que usa los mismos pines, mejor destinar otro par de pines cualesquiera a la transmisión. Aunque para ello tenemos que importar una librería que habilite la comunicación serie con otros pines como es la librería **SoftwareSerial**.

Importamos la librería que viene de serie en el IDE y creamos un nuevo objeto serie llamado BT1 conectado a los pines 4 y 2:

```
#include <SoftwareSerial.h>
SoftwareSerial BT1(4,2); // RX, TX
```

Y después, podemos usar BT1 exactamente igual a como usamos Serial.



Ejercicio 0 test bluetooth

```
//*****
// PRUEBA CERO BLUETOOTH ARDUINO OFIMEGA
//*****
//ENCIENDE EL PIN 13 SI EL bt ESTA DISPONIBLE
#include <SoftwareSerial.h> //libreria de transmisión
SoftwareSerial BT(2,3); // crea objeto serie BT
//recordar que se cruzan pin 2 al TX pin 3 al RX
void setup() {
  Serial.begin(9600);//para monitor serie
  BT.begin(9600);
  pinMode(13,OUTPUT);
  Serial.println("BT ACTIVANDO...");
}
void loop() {
  delay(100);
  if (BT.available())
  {
    digitalWrite(13, HIGH); //led enciende
    Serial.println("BT ACTIVADO");
  }
  else
  {
    digitalWrite(13, HIGH); //led apaga
    Serial.println("BT DESACTIVADO");
  }
}
```

Ejercicio 1.A test de comandos

```
**PROBADO EN HC-05 Ofimega OK **
#include <SoftwareSerial.h> //libreria de transmisión
SoftwareSerial BT1(10,11); //prefer. pines altos
// RX, TX recordar que se cruzan RX a 11 y Tx a 10
void setup()
{
  Serial.begin(9600);
  Serial.println("Listo");
  BT1.begin(38400); //-> Cambiado de 9600
}
void loop()
{
  if(BT1.available()) Serial.write(BT1.read());
  //lee el BT y muestra en Arduino
  if(Serial.available()) BT1.write(Serial.read());
  //lee del arduino y manda al BT
}
//-> Enviar por monitor serial: AT
AT+DSWD? - AT+UART? -> devuelve 9600
```

Ejercicio 1.B

Como el modulo HC-06, a diferencia del HC-05, no espera un terminador de línea como \n, si no que salta por tiempo, lo que nos fuerza a escribir deprisa, lee una línea con GetLine() y finaliza con intro antes de enviar la línea completa al BT1. Listo para enviar comandos AT.

```
#include <SoftwareSerial.h>
SoftwareSerial BT1(4,2); // RX, TX recorder que se cruzan
void setup()
{
  Serial.begin(9600);
  Serial.println("Enter AT commands:");
  BT1.begin(9600);
}
void loop()
{
  if (BT1.available())
    Serial.write(BT1.read());

  if (Serial.available())
  { String S = GetLine();
    BT1.print(S);
    Serial.println("---> " + S);
  }
}
String GetLine()
{ String S = "" ;
  if (Serial.available())
  { char c = Serial.read(); ;
    while ( c != '\n') //Hasta que el caracter sea intro
    { S = S + c ;
      delay(25) ;
      c = Serial.read();
    }
    return( S + '\n') ;
  }
}
```


Control de giro servomotor por Bluetooth

```
// Servomotor regulando ángulo por bluetooth
// recursos: un LED con Resis 220 y un servomotor
//           + aplicación BT que envíe datos en forma de caracteres
// Tested by Ofimega
//-----

#include <Servo.h>           // Incluye la librería Servo
#include <SoftwareSerial.h> // Incluye la librería transmisión serie bluetooth
Servo servo1;              // Crear un objeto tipo Servo llamado servo1
SoftwareSerial BT(10,11);  // Puerto serie BT conectado RX, TX a pines 10 y 11
                          // recordar que van cruzados pin 10 al TX del BT

int angulo = 0 ;
int angulofin = 0 ;
int modoauto=0;
byte datoB; //dato recibido por BT o por USB del tipo byte (1 caracter)

void setup()
{
  servo1.attach(9) ; // Conectar servo1 (cable amarillo) al pin 9
  Serial.begin(9600); //activa transmisión serie USB PC a 9600 baudios
  BT.begin(9600);    //activa transmisión por BT
  pinMode(4, OUTPUT); //salida LED Conectar cátodo a pin4 y R220 a GND
}

void loop()
{
  delay(10);
  if (BT.available()) //si el puerto BT esta disponible
  {
    datoB = BT.read();
  }
  else //si no va por BT, controla por el serial USB
  {
    datoB= Serial.read();
    Serial.println(String("No disponible"));
  }
  if (datoB==65) // letra A
  {
    digitalWrite(4,HIGH); //salida digital LED
    modoauto=1;
  }
  else if (datoB==68) //letra D
  {
    digitalWrite(4,LOW); //salida digital LED
    modoauto=0;
  }
  else if (datoB>=48 && datoB<=57) // numeros
  {
    if (modoauto==1)
    {
      angulofin = map(datoB , 48,57, 0, 180); // escala numeros 0 al 9 en ascii
      while (angulo!=angulofin) //bucle de retardo progresivo
      {
        if (angulo>angulofin) angulo--;
        else if (angulo<angulofin) angulo++;
        servo1.write(angulo);
        delay(10);
      }
    }
  }
}
}
```

Control de giro mástil aerogenerador por potenciómetro y por Bluetooth

Un potenciómetro bajo la veleta indicará el ángulo de giro de un servomotor situado en la base del mástil del aerogenerador. Los datos enviados/recibidos por serial o BT tienen que ser del tipo carácter.

```
//Servo regulado ángulo por potenciómetro y salida mostrada
por otra señal de entrada analógica
//-----
#include <Servo.h> // Incluye la librería Servo
#include <SoftwareSerial.h> // Incluye la librería transmisión
serie bluetooth
Servo servo1; // Crear un objeto tipo Servo
llamado servo1
SoftwareSerial BT(10,11); // Puerto serie BT conectado RX, TX
a pines 10 y 11
//recordar que van cruzados pin 10
al TX del BT
int angulo = 0 ;
int PosInicial=0;
int PosFinal=0;
int valorLED=0;
int pin_pot=0;
float Vin=0;
int activalarma=0;
int modoauto=1;
int lecturaVin=0; //valor de lectura de la tensión
int vel=0; //valor de la velocidad
int limit=1000; //valor establecido límite máximo
byte datoB,datoU; //datos enviados por BT o por USB del tipo
byte (1 caracter)
//int controlservo(PosFinal);
void setup()
{
  pin_pot = 0; // Conectado el pin de entrada analógica
  A0 al central del potenciómetro
  servo1.attach(9) ; // Conectar servo1 (cable amarillo)
  al pin 9
  Vin=1; //conectar señal de entrada Volt. in a pin A1
  Serial.begin(9600); //activa transmisión serie USB PC a
  9600 baudios
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
}

void loop()
{
  if (activalarma==0) PosFinal=analogRead(A0); else
  PosFinal=0;
  if (modoauto=1) controlservo(); //si está en modo
  automático se controla por veleta
  //control de voltaje entrada por Vin (A1) y testigo LED
  por salida analógica A3
  lecturaVin=analogRead(Vin);
  valorLED=lecturaVin/200; //valorLED amplificado o
  reducido para max 250 segun valor de lectura serial
  vel=lecturaVin*100;
  Serial.println(String("Tensio Volts: ") +lecturaVin);
  //muestra valor por serial de la lectura
  Serial.println(String("\nVelocitat rpm:" ) +vel);
  analogWrite(3, valorLED) ; //salida analógica por A3
  if (lecturaVin>limit) //si la lectura es mayor al límite
  {
    activalarma=1; //activa el flag alarma
    alarma(); //salta al procedimiento alarma
  }
  else
  {
    activalarma=0; //desactiva el flag alarma
    controlBT(); //llama al procedimiento de control por BT
    digitalWrite(4,HIGH); //salida digital por A3 LED piloto
    en estado encendido.
    delay(50);
  }

  void controlservo() //control de giro servomotor por
  potenciómetro de la veleta...
  {
    //PosFinal=analogRead(A0);
    while (PosInicial!=PosFinal) //bucle de retardo
    progresivo para aumentar suavidad de giro
    {
      if (PosInicial>PosFinal) PosInicial--; else
      PosInicial++;
      angulo = map(PosInicial , 0,1000, 0, 180); // y
      escala
      servo1.write(angulo);
      delay(15);
    }
    PosInicial=PosFinal;
  }

  void alarma()
  {
    digitalWrite(4,LOW); //apaga LED4 para parpadeo.
    Serial.println(String("\n <-- Alarma velocitat limit-->
    ") +vel); // Informa por serial USB
    delay(100);
  }

  void controlBT()
  {
    if (BT.available()) //si está disponible el puerto BT
    {
      datoB= BT.read(); //leemos el dato y lo guarda en una
      variable de tipo carácter
      if (datoB >= '0' && datoB <= '9' )
      { // chequeamos si es un valor entre 0 y 9
        int preAngulo = datoB - '0'; // si lo es lo
        convertimos a un valor entero entre 0 y 9
        PosFinal = map(preAngulo, 0, 9, 0, 100); // y lo
        mapeamos a un valor entre 0 y 900
        controlservo(); //llama a la función de girar en
        servo
      }
      else if (datoB='A') modoauto=1;
      else if (datoB='D') modoauto=0;
    }
  }
}

```


USB para Arduino, con USB Host Shield.

El modulo USB para Arduino, conocido como USB Host Shield. Se le denomina host, porque hace las veces de un controlador maestro. Para control de un servo desde un mando PS3, PS4 o Xbox conectado a una USB o mediante un emisor Bluetooth por USB:

Descargar y añadir la librería desde: https://github.com/felis/USB_Host_Shield_2.0 a las librerías del software de programación de Arduino.

Guárdala en la carpeta: C:\Program Files (x86)\Arduino\libraries

Con el nombre: *USBHostShield*

Conexión por USB de un mouse, PS3 o PS4 o Xbox:

Abre el ejemplo:

Archivo – Ejemplos – USBHostShieldLibraryXX ->PS4Usb

```
#include <PS4USB.h>

// Satisfy the IDE, which needs to see the include statment
// in the ino too.
#ifdef dobogusinclude
#include <spi4teensy3.h>
#endif
#include <SPI.h>

USB Usb;
PS4USB PS4(&Usb);

bool printAngle, printTouch;
uint8_t oldL2Value, oldR2Value;

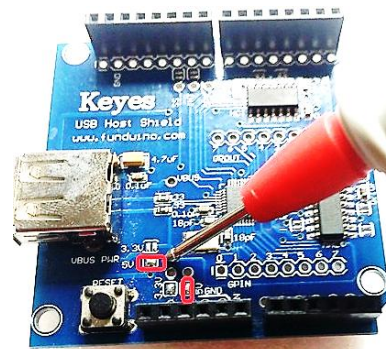
void setup() {
  Serial.begin(115200);
  if (Usb.Init() == -1) {
    Serial.print(F("\r\nOSC no inicia"));
    while (1); // Halt
  }

  Serial.print(F("\r\nPS4 USB Library Started"));
}

void loop() {
  Usb.Task();
  if (PS4.connected()) {
    Serial.print("Mando conectado");
    if (PS4.getButtonClick(UP)) {
      Serial.print(F("\r\nUp"));
      PS4.setLed(Red);
    }
    if (PS4.getButtonClick(RIGHT)) {
      Serial.print(F("\r\nRight"));
      PS4.setLed(Blue);
    }
    if (PS4.getButtonClick(DOWN)) {
      Serial.print(F("\r\nDown"));
      PS4.setLed(Yellow);
    }
    if (PS4.getButtonClick(LEFT)) {
      Serial.print(F("\r\nLeft"));
      PS4.setLed(Green);
    }
  }
}
```



Problemas: Si no responde al subir el sketch con el *Shield* conectado, puede que el puerto USB del PC no tenga suficiente intensidad. En ese caso, prueba a subirlo con el Arduino conectado a una pila externa. Ojo con los Shields chinos, a veces es necesario puentear las entradas de 5 y 3,3 V.



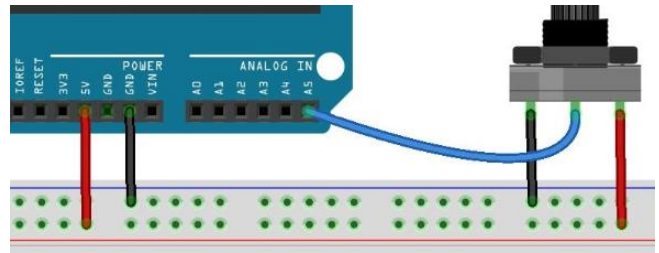
Prácticas básicas en Arduino

Práctica 1. Entradas analógicas - salida serie

Parte 1:

Montaje hardware:

- ▶ Montamos el esquema de la figura, donde damos tensión a los extremos de un potenciómetro y conectamos el pin central (el variable) a la entrada de la puerta A5 de Arduino.
- ▶ Los convertidores ADC leen valores de tensión y no resistencia, por lo tanto, lo que vamos a leer es la caída de tensión en el potenciómetro a medida que giramos. Esto lo mostraremos en el monitor serie.
- ▶ Los convertidores de Arduino UNO y Mega son de 10 bits de resolución por lo que nos devolverá valores entre 0 y $2^{10} = 1.024$ para tensiones entre 0 y 5V. En cambio, el Arduino DUE dispone de convertidores de 12 bits por lo que el valor de sus lecturas estará entre 0 y 1012 o sea 4.096, es decir tiene mejor resolución (pero sólo puede leer hasta 3,3V).
- ▶ Asegúrate de no usar sensores que puedan dar más de 5V máximo (con Arduino UNO y Mega), ya que dañarías el chip principal de Arduino.



Montaje software:

Sketch 1: Lectura serial

Lectura del potenciómetro y pasado a una variable entera llamada *Lectura*. Luego mostrar esta lectura en el monitor serie usamos la función:

```
Serial.print(valor, formato);  
Ej: Serial.println(Lectura, BIN);  
     Serial.println(Lectura, HEX);  
     Serial.println(1.555, 2); //2 decimales
```

Sketch 2: Umbral

Usa una constante *Umbral* para encender el led 13 al llegar la lectura a ese valor y apagarlo si es inferior.

Sketch 3: Contador de pulsosint conta =0;

```
int flag=0;  
void setup()  
{  
  pinMode(13,OUTPUT);  
  Serial.begin(9600);  
}  
void loop()  
{  
  int Lectura = analogRead(A5) ;  
  delay(Lectura) ;  
  if (flag==0)  
  {flag=1;  
   conta++;  
   Serial.println(conta);  
  }  
  else flag=0;  
  digitalWrite(13, flag);  }
```

```
void setup()  
{  
  Serial.begin(9600); // transm. serie  
}  
void loop()  
{  
  int Lectura = analogRead(A5) ;  
  Serial.println(Lectura);  
  delay(1000) ;  
}  
  
const int umbral = 500;  
void setup()  
{  
  pinMode(13,OUTPUT);  
  Serial.begin(9600);  
}  
void loop()  
{  
  int Lectura = analogRead(A5) ;  
  Serial.println(Lectura);  
  if (Lectura > umbral) digitalWrite(13, HIGH);  
  else digitalWrite(13, LOW);  
  delay(1000) ;  
}
```

Sketch 4: Indicador de 4 leds

```

const int umbral = 500;
void setup()
{
  pinMode(13,OUTPUT);
  pinMode(12,OUTPUT);
  pinMode(11,OUTPUT);
  pinMode(10,OUTPUT);
  Serial.begin(9600);
}
void loop()
{
  digitalWrite(13, LOW);
  digitalWrite(12, LOW);
  digitalWrite(11, LOW);
  digitalWrite(10, LOW);
  int Lectura = analogRead(A5) ;
  int nivel = round(Lectura/255);

  Serial.println(nivel);
  switch(nivel){
    case 0:
      digitalWrite(13, HIGH);
      break;
    case 1:
      digitalWrite(12, HIGH);
      break;
    case 2:
      digitalWrite(11, HIGH);
      break;
    default:
      digitalWrite(10, HIGH);
      break;
  }
  delay(1000) ;
}

```

Sketch avanzado:

La función millis() que nos indica en milisegundos el tiempo transcurrido desde que iniciamos Arduino y la podemos usar para contar lecturas/segundo.

Usamos una var T unsigned long para guardar millis porque es el tipo que Arduino usa internamente para su reloj. Sería un error manejar millis con un int porque su valor máximo es 32.767

```

void setup()
{ Serial.begin(9600); }
void loop()
{
  unsigned long T ;
  int n = 0 ;
  T = millis();
  while (millis() <= T + 1000) // Mientras no pase un Segundo = 1000 mS
  {
    analogRead( A5) ;
    n++ ; // Contamos cada vez que leemos
  }
  Serial.println(n);
}

```

Práctica 2. Entrada digital – sensor infrarojos pasivo PIR

Los PIR captan la radiación infrarroja que emiten determinadas fuentes de energía tales como el calor del cuerpo humano o animales. Es llamado pasivo debido a que no emite radiaciones, sino que las recibe.

Su componente principal es un sensor piroeléctrico. Se trata de un componente electrónico diseñado para detectar cambios en la radiación infrarroja recibida. La información infrarroja llega al sensor piroeléctrico a través de una lente de fresnell que divide el área protegida en sectores hasta un ángulo de barrido de 120 °.

Incorpora dos potenciómetros para ajuste de la sensibilidad y el tiempo.

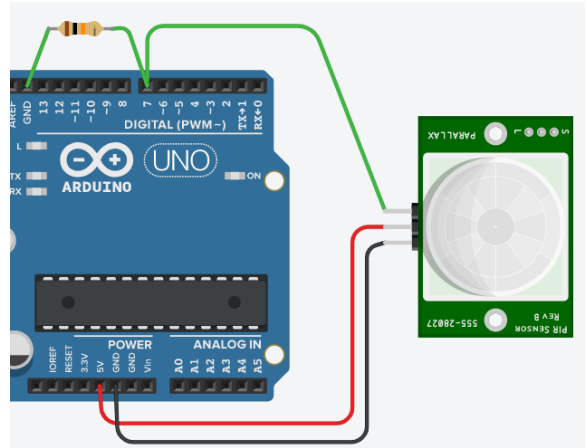
Se alimenta entre 5V y 12 V. Salida desde CMOS (3,3 V) a TTL (5V) con ajuste variable de tiempo desde 0,3 seg a 5 min.

Prueba 1 de sensor de movimiento PIR:

```
byte sensor=7;
byte led=13;
void setup() {
  pinMode(sensor, INPUT);
  pinMode(led,OUTPUT);
}

void loop() {
  if (digitalRead(sensor)==HIGH)
    digitalWrite(led,HIGH);
  else
    digitalWrite(led,LOW);

  delay(100);
}
```

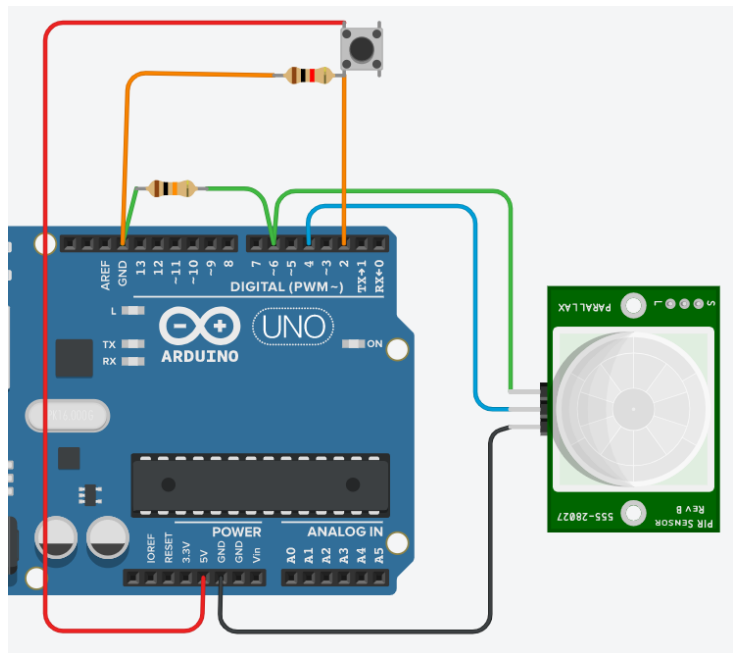


Prueba 2 de sensor de movimiento PIR:

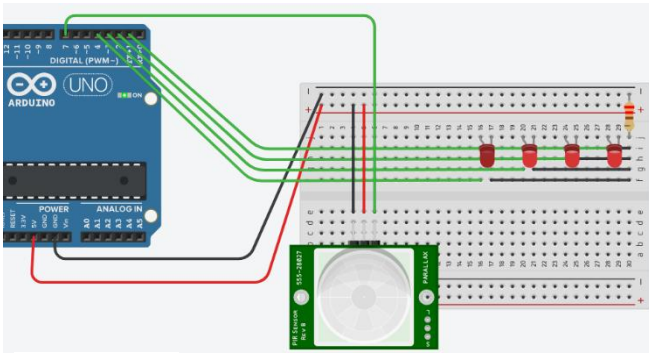
```
byte sensor=6;
byte activador=4;
byte led=13;
byte pulsador=2;
void setup() {
  pinMode(sensor, INPUT);
  pinMode(pulsador,INPUT);
  pinMode(activador,OUTPUT);
  pinMode(led,OUTPUT);
  digitalWrite(activador,LOW);
}

void loop() {
  delay(10);
  if (digitalRead(pulsador)==HIGH)
  {
    delay(1000);
    digitalWrite(activador,HIGH);
  }

  if (digitalRead(sensor)==HIGH)
    digitalWrite(led,HIGH);
  else
    digitalWrite(led,LOW);
}
```



Prueba 3 de sensor de movimiento PIR:



```
byte sensor=7;
byte led1=1;
byte led2=2;
byte led3=3;
byte led4=4;
void enciende(); //usaremos una función aparte
llamada enciende
```

```
void setup() {
  pinMode(sensor, INPUT);
  pinMode(led1,OUTPUT);
  pinMode(led2,OUTPUT);
  pinMode(led3,OUTPUT);
  pinMode(led3,OUTPUT);
}
```

Mejora 3.1:

```
byte sensor=7;
byte leds;
int duracion=1000; //valor predeterminado
void enciende(); // -> aquí declaramos la función
(termina en ; )

void setup()
{
  pinMode(sensor, INPUT);
  for(leds=1;leds<4;leds++)
  pinMode(leds, OUTPUT); // pines 6 a 13 como salida
}

void loop() {
  if (digitalRead(sensor)==HIGH) enciende();
  delay(100);
}

void enciende()
{
  for(leds=1;leds<4;leds++)
  {
    digitalWrite(leds, HIGH);
    delay(200);
  }
  delay(duracion);
  for(leds=4;leds>0;leds--)
  {
    digitalWrite(leds, LOW);
    delay(1000);
  }
}
```

```
void loop() {
  if (digitalRead(sensor)==HIGH) enciende();
  delay(100);
}

void enciende()
{
  digitalWrite(led1,HIGH);
  delay(100);
  digitalWrite(led2,HIGH);
  delay(100);
  digitalWrite(led3,HIGH);
  delay(100);
  digitalWrite(led4,HIGH);
  delay(2000);
  digitalWrite(led4,LOW);
  delay(1000);
  digitalWrite(led3,LOW);
  delay(1000);
  digitalWrite(led2,LOW);
  delay(1000);
  digitalWrite(led1,LOW);
  delay(1000);
}
```

Mejora 3.2:

```
Cambio a ultrasonidos
#define trig 7 //pin 7 al trigger
#define echo 6 //pin 6 al echo
byte leds;
int duracion=1000; //valor predeterminado
void enciende(); //
void setup()
{
  pinMode(trig, OUTPUT); //configuramos pines
  pinMode(echo, INPUT);
  for(leds=1;leds<4;leds++)
  pinMode(leds, OUTPUT); // pines 6 a 13 como salida
}

void loop() {
  long tiempo, distancia ; // variables grandes
  digitalWrite(trig, LOW); // desactivado
  delayMicroseconds(2);
  digitalWrite(trig, HIGH); // Activamos pulso
  delayMicroseconds(10); // mantenemos 10µs
  digitalWrite(trig, LOW); // Cortamos pulso
  tiempo = pulseIn(echo, HIGH) ;// Tiempo en recibir
  distancia = tiempo / 2 / 29.1;// Convertimos
  if (distancia<200) enciende();
}

void enciende()
{
  for(leds=1;leds<4;leds++) {
    digitalWrite(leds, HIGH);
    delay(200);
  }
  delay(duracion);
  for(leds=4;leds>0;leds--){
    digitalWrite(leds, LOW);
    delay(1000);
  }
}
```

Práctica 1. Puerto serie

En la placa Arduino UNO, los pines Tx y Rx estarán internamente conectados al puerto USB, por lo que podremos utilizar el puerto Serie a través del conector USB

Repasemos las funciones principales:

Salida:

- Serial.begin(velocidad): Donde velocidad serán los baudios por segundo. Frecuentemente 9600
- Serial.print(Mensaje): Envía texto (entre comillas dobles) o caracteres ASCII (entre apóstrofes).
- Serial.println(mensaje): Imprime el texto ASCII, agregándole nueva línea y retorno de carro (ASCII 13, o '\r')

Entrada:

- Serial.available(): Disponible para leer desde el puerto serie el búfer de recepción (que contiene 64 bytes).
- Serial.read(): Lee datos seriales entrantes.
- serialEvent(): Función que detecta si hay datos en el puerto serie como interrupción asíncrona.

Encender y Apagar LEDS desde PC por Puerto Serie

Ejercicio nivel 1

```
int ledPin1= 13;
int val=0;
void setup() {
  Serial.begin(9600);
  pinMode(ledPin1, OUTPUT);
}
void loop(){
  val= Serial.read(); // lee el valor del puerto
  if(val!= -1) // si es alguna tecla...
  {
    if (val=='1') digitalWrite(ledPin1, HIGH);
    else digitalWrite(ledPin1, LOW);
  }
}
```

Ejercicio nivel 2

```
int ledPin1= 13, ledPin2= 12, ledPin3= 11;
int status1 = HIGH, status2 = HIGH, status3 = HIGH;
int val;
void setup() {
  Serial.begin(9600);
  pinMode(ledPin1, OUTPUT);
  pinMode(ledPin2, OUTPUT);
  pinMode(ledPin3, OUTPUT);
}
void loop(){
  val= Serial.read(); // lee el valor del puerto
  if(val!= -1) // si es alguna tecla...
  {
    switch(val) { //activa los posibles casos
      case'1': status1 = !status1; break; // si tecla es 1
      case'2': status2 = !status2; break; // si tecla es 2
      case'3': status3 = !status3; break; // si tecla es 3
    }
  }
  digitalWrite(ledPin1, status1); //saca estado1
  digitalWrite(ledPin2, status2); //saca estado2
  digitalWrite(ledPin3, status3); //saca estado3
}
```

Ejercicio nivel 3

```
int pinLED1 = 13;
String entradaSerie = ""; // para almacenar entrada
bool entradaCompleta = false; // si está completo

void setup() {

  pinMode(pinLED1,OUTPUT);
  Serial.begin(9600); // Iniciar el puerto serie
}

void loop() {

  if(entradaCompleta) {
    if(entradaSerie == "encender\n"){
      digitalWrite(pinLED1,HIGH);
      Serial.print("LED encendido!\n");
    }
    else if(entradaSerie == "apagar\n"){
      digitalWrite(pinLED1,LOW);
      Serial.print("LED apagado :(\n");
    }
    else { // Cualquier otro dato recibido
      //Serial.println("Dato inválido!");
    }
    entradaSerie = "";
    entradaCompleta = false;
  }
}
```

```
// Función que se activa al recibir algo por
// el puerto serie, Interrupción del Puerto Serie.
```

```
void serialEvent() { interrupción por puerto serie

while (Serial.available()) { // Obtener bytes
  char inChar = (char)Serial.read();
  entradaSerie += inChar; // Agregar al String
  if (inChar == '\n') { //detecta caractre Enter
    entradaCompleta = true;
  }
}
}
```

introducción a Processing

Programas básicos de ejemplo:

Ejercicio 1

```
void setup() //solo una vez
{
  println("Bienvenido a mi programa cutre");
  size(500,400);
  background(200); //tono de gris
  text("Dibujo un punto",300,10);
  point(300, 300);
  text("Dibujo una linea",300,30);
  line(30, 30, 30,200);

  text("Dibujo una linea roja",300,40);
  stroke(250,0,0);
  line(30, 100, 200,100);
  text("Dibujar un triángulo",300,50);
  triangle(10,10,20,02,30,30);
  text("Dibujo una elipse negra",300,60);
  fill(0); //negro
  ellipse(200,200,100,100);

}

void draw() //repite en bucle
{
  fill(0,200,0);
  rect (mouseX,mouseY,40,40);
  if (mouseButton==LEFT) {
    fill(0,0,250);
    ellipse(mouseX,mouseY,100,100);
  }
}
```

Ejercicio 2

```
void setup() //solo una vez
{
int incremento=200;
size(500,400);
textSize(16);
fill(0,0,250); //azul
}

void draw() //repite en bucle
{
background(240,240,10);//amarillo
text("Pulsa una tecla",20,20);
String texto="has pulsado: ";
if (keyPressed==true) {
text(texto+key,200,200);
}
}
```

Control del módulo bluetooth HC-05

Práctica test de comandos

```

**PROBADO EN HC-05 Ofimega OK **

#include <SoftwareSerial.h> //librería de transmisión
SoftwareSerial BT1(10,11); //prefer. pines altos
// RX, TX recordar que se cruzan RX a 11 y Tx a 10

void setup()
{
  Serial.begin(9600);
  Serial.println("Listo");
  BT1.begin(38400); //-> Cambiado de 9600
}

void loop()
{
  if(BT1.available()) Serial.write(BT1.read());
  //lee el BT y muestra en Arduino
  if(Serial.available()) BT1.write(Serial.read());
  //lee del arduino y manda al BT
}

//-> Enviar por monitor serie en modo NL&CR:
AT ↵ devuelve OK - AT+UART? -> devuelve 9600
  
```

Práctica control por móvil 1

Conexionamos como en la figura →

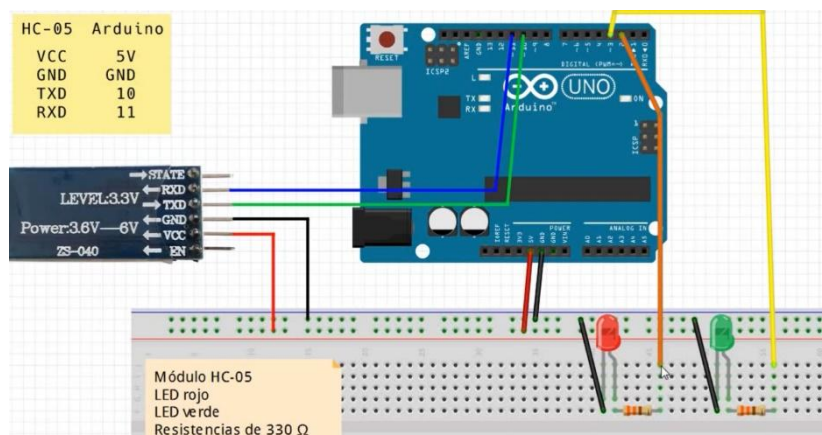
Instalar la aplicación para Android en el dispositivo móvil desde la PlayStore: Bluetooth Serial Controller

Donde aparecerán unos botones programables para enviar comandos específicos.

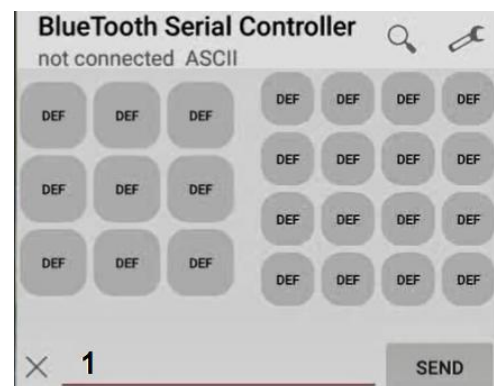
Sketch 1:

```

#include <SoftwareSerial.h> //librería de transmisión
SoftwareSerial BT1(10,11); //prefer. pines altos
// RX, TX recordar que se cruzan RX a 11 y Tx a 10
char DATO = 0;
int LED1=13;
void setup()
{
  BT1.begin(38400); //-> Cambiado de 9600
  pinMode(LED1,OUTPUT);
}
void loop()
{
  if(BT1.available()) //si hay datos disponibles...
  {
    DATO=BT1.read();
    if (DATO=='1') digitalWrite(LED1,HIGH); //enciende el LED
    if (DATO=='2') digitalWrite(LED1,LOW); //apaga el LED
  }
}
  
```



Ambos NL & CR 9600 baudio



Prueba: Ejecutar la app en el móvil. Poner en orientación horizontal. Pulsar en lupa para conectar con el HC-05 y enviar el número de prueba 1 para encender el LED del Arduino. Poner en *Preference* el modo 9 botones.

Variaciones: Encender y apagar con el mismo número: `if (DATO=='1') digitalWrite(LED1, !digitalRead(LED1));`

Variar el Brillo:

```
if (DATO=='2') {BRILLO=BRILLO+10; if (BRILLO>255) BRILLO=255; analogWrite(LED1,BRILLO);
```

```
if (DATO=='3') {BRILLO=BRILLO-10; if (BRILLO<0) BRILLO=0; analogWrite(LED1,BRILLO);
```

Sketch 2: Servo controlado por bluetooth

```
#include <Servo.h> // Incluimos la librería Servo
#include <SoftwareSerial.h> //librería de transmisión
SoftwareSerial BT1(10,11); //prefer. pines altos
char DATO = 0;
Servo servo_mot; // Creamos un objeto tipo Servo llamado servo_mot
int vel = 0; // variable entera para control de velocidad
void setup()
{
  servo_mot.attach(12); // Conectar cable servo de control al pin 12
  Serial.begin(9600); //activamos la comunicación serie con el PC
  Serial.println ("Q sube vel, Z baja vel, A para:");
  BT1.begin(38400); //-> Cambiado de 9600
}
void loop()
{
  if(BT1.available()) //si hay datos disponibles...
  {
    DATO=BT1.read();
    if(DATO != -1) // (si no es ninguna, por defecto, vale -1)
    {
      switch(DATO) { //activa los posibles casos según el valor
        case'q': vel = vel+2; break; // si la tecla es q aumenta vel
        case'a': vel=0 ; break; // si la tecla es a para
        case'z': vel=vel-2; break; // si la tecla es z y vel>0 disminuye velocidad
      }
      servo_mot.write(map(vel, 0, 100, 90, 180)); //cambia valores rango 0 a 100 a angulo 90 a 180
      delay(20);
    }
  }
}
```

En Bluetooth Serial Controller: Configurar los botones programables en Preferences:

Button 1: Name + Command: q,

Button 2: Name: - Command: a

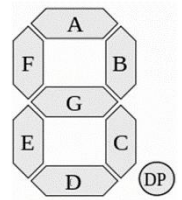
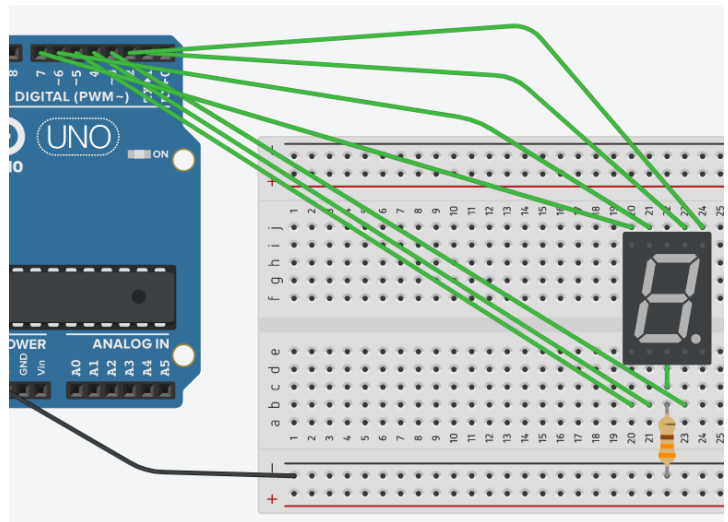
Button 3: Name Stop Command: z

Control display de 7 segmentos.

```
int salidas[] = {1,2,3,4,5,6,7,8};
bool leds[4][8] //display de 7 segmentos
{
  //{a,b,c,d,e,f,g,dp}
  {1,1,1,1,1,1,0,0},//0
  {0,1,1,0,0,0,0,0},//1
  {1,1,0,1,1,0,1,0},//2
  {1,1,1,1,0,0,1,0},//3
};

void setup() {
  for (int i = 1; i < 8; i++)
    pinMode(salidas[i], OUTPUT);
  for (int i = 1; i < 8; i++) { //Inicializa
  las salidas
  digitalWrite(salidas[i], LOW);}
}

void loop() {
  for (int num=0;num<4;num++){
    for (int i = 1; i < 9; i++)
      digitalWrite(salidas[i], leds[num][i]);
    delay(1000);
  }
}
```



Ejercicio 2

```

void displaya(int digit);
void apagaleds();
int a = 2; // segment "a"
int b = 3; // segment "b"
int c = 4; // segment "c"
int d = 5; // segment "d"
int e = 6; // segment "e"
int f = 7; // segment "f"
int g = 8; // segment "g"

int entrada=A0;
int valor = 0;

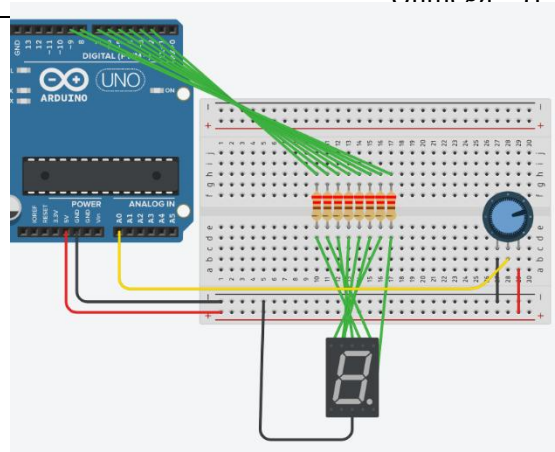
void setup() {
for (int i=2;i<9;i++)
pinMode(i, OUTPUT); //modo salida
}

void loop() {
apagaleds();
valor = analogRead(entrada);
float voltage= valor * (5.0 / 1023.0);
int a = round (voltage);
//int val=(int) ((a*5)/9);
displaya(a);
}

void displaya(int digit)
{
//segment a
if(digit==2 || digit == 3 || digit == 5||digit == 8||digit == 9 || digit == 0||digit == 7)
digitalWrite(a,0);
//segment b
if(digit==2 || digit == 3 || digit == 7||digit == 8||digit == 0|| digit == 1||digit == 9)
digitalWrite(b,0);
//segment c
if(digit==3 || digit == 5 || digit == 7||digit == 6||digit == 8|| digit == 1||digit == 0||digit == 9||digit == 4)
digitalWrite(c,0);
//segment d
if(digit==2 || digit == 3 || digit == 5||digit == 6 ||digit == 8||digit == 0)
digitalWrite(d,0);
//segment e
if(digit == 2 || digit ==6 || digit == 8 || digit==0)
digitalWrite(e,0);
//segment f
if(digit == 4 || digit ==5 || digit==6 || digit ==8|| digit==0||digit == 9)
digitalWrite(f,0);
//segment g
if (digit==2 || digit==3 ||digit==4 ||digit==5 ||digit==6 ||digit==8||digit == 9)
digitalWrite(g,0);
}

void apagaleds()
{
digitalWrite(a,0);
digitalWrite(b,0);
digitalWrite(c,0);
digitalWrite(d,0);
digitalWrite(e,0);
digitalWrite(f,0);
digitalWrite(g,0);
}

```



Variante 2 inclueyndo letras:

```
void displaya(int digit);
void displaya letra(char letra);
void apaga leds();
int a = 2; // segment "a"
int b = 3; // segment "b"
int c = 4; // segment "c"
int d = 5; // segment "d"
int e = 6; // segment "e"
int f = 7; // segment "f"
int g = 8; // segment "g"

int entrada=A0;
int valor = 0;

void setup() {
for (int i=2;i<9;i++)
pinMode(i, OUTPUT); //modo salida
}

void loop() {

// valor = analogRead(entrada);
// float voltage= valor * (5.0 / 1023.0);
// int a = round (voltage);
//int val=(int) ((a*5)/9);
apaga leds();
displaya letra('c');
delay(1000);
apaga leds();
displaya letra('a');
delay(1000);

}

void displaya(int digit)
{
//segment a
if(digit==2 || digit == 3 || digit == 5||digit == 8||digit ==
9 || digit == 0||digit == 7)
digitalWrite(a,0);
//segment b
if(digit==2 || digit == 3 || digit == 7||digit == 8||digit ==
0|| digit == 1||digit == 9)
digitalWrite(b,0);
//segment c
if(digit==3 || digit == 5 || digit == 7||digit == 6||digit ==
8|| digit == 1||digit == 0||digit == 9||digit == 9)
digitalWrite(c,0);
//segment d
if(digit==2 || digit == 3 || digit == 5||digit == 6 ||digit
== 8||digit == 0)
digitalWrite(d,0);
//segment e
if(digit == 2 || digit ==6 || digit == 8 || digit==0)
digitalWrite(e,0);
//segment f
if(digit == 4 || digit ==5 || digit==6 || digit ==8||
digit==0||digit == 9)
digitalWrite(f,0);
//segment g
if (digit==2 || digit==3 ||digit==4 ||digit==5 ||digit==6
||digit==8||digit == 9)
digitalWrite(g,0);
}

void displaya letra(char letra)
{
switch (letra)
{
case 'c':{
digitalWrite(a,0);
digitalWrite(d,0);
digitalWrite(e,0);
digitalWrite(f,0);
break;
}
}
```

```
case 'a':{
digitalWrite(a,0);
digitalWrite(b,0);
digitalWrite(c,0);
digitalWrite(e,0);
digitalWrite(f,0);
digitalWrite(g,0);
break;
}
}
```

```
void apaga leds()
{
digitalWrite(a,1);
digitalWrite(b,1);
digitalWrite(c,1);
digitalWrite(d,1);
digitalWrite(e,1);
digitalWrite(f,1);
digitalWrite(g,1);
}
```