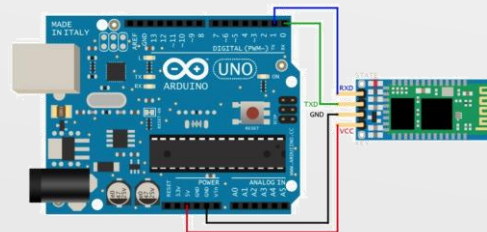


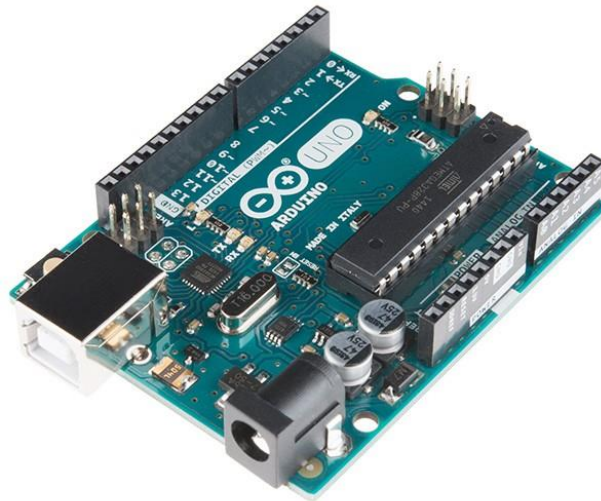
ACTUALIZACIÓN CIENTÍFICO DIDÁCTICA PARA PROFESORES DE TECNOLOGÍA, CONTROL Y ROBÓTICA (PARTE 1)



Miguel Salvador González
CURSO 2021-2022

¿Qué es Arduino?

- ❑ Se trata de un dispositivo programable utilizado en sistemas de control y robots.
- ❑ Cumple la función de regulador, es decir, recoge información de los sensores y controla los actuadores para que realicen la tarea deseada.
- ❑ Se trata de un hardware libre, cualquiera puede construirlo a partir de la documentación disponible para ello



Otras tarjetas controladoras



FLOW GO



TARJETAS CONTROLADORAS

FISCHERTECHNIK



LLWin 3.0



ARDUINO



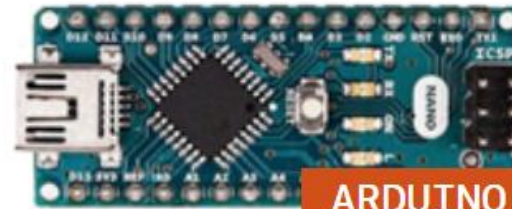
SCRATCH

C
LENGUAJE

Placas Arduino



ARDUINO UNO



ARDUINO NANO



ARDUINO LEONARDO



ARDUINO MEGA 2560

Arduino UNO

- ❑ Es la placa más utilizada y es la que suele venir en los kits de iniciación.
- ❑ Se alimenta a través de cable usb o adaptador de corriente tipo impresora.
- ❑ Tiene 14 pines digitales y 5 analógicos
- ❑ Utiliza el microcontrolador ATmega328P



Arduino Leonardo

- ❑ Se trata de una evolución de Arduino Uno
- ❑ Utiliza el microcontrolador ATmega32u4 y dispone de los mismos pines que Arduino Uno.
- ❑ Al conectarlo al ordenador, éste lo puede detectar como si fuera un ratón o teclado.
- ❑ Utiliza un cable micro USB



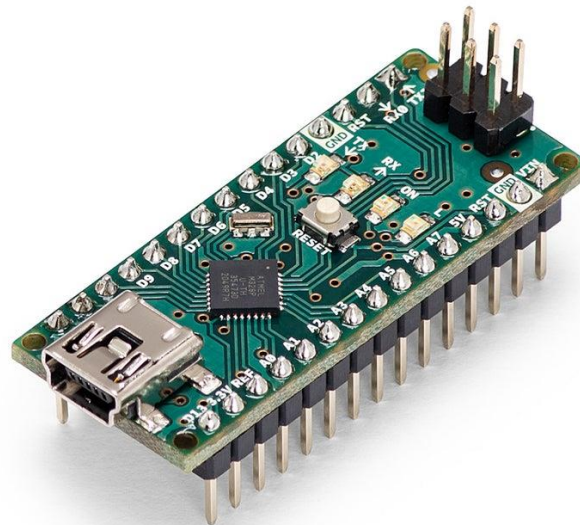
Arduino MEGA

- ❑ Dispone de un gran número de pines (70)
- ❑ Utiliza el microcontrolador ATmega2560, más potente que el de UNO y más memoria
- ❑ Se utiliza en aplicaciones complejas o que requieran un gran número de sensores o actuadores (impresoras 3D)













Arduino NANO

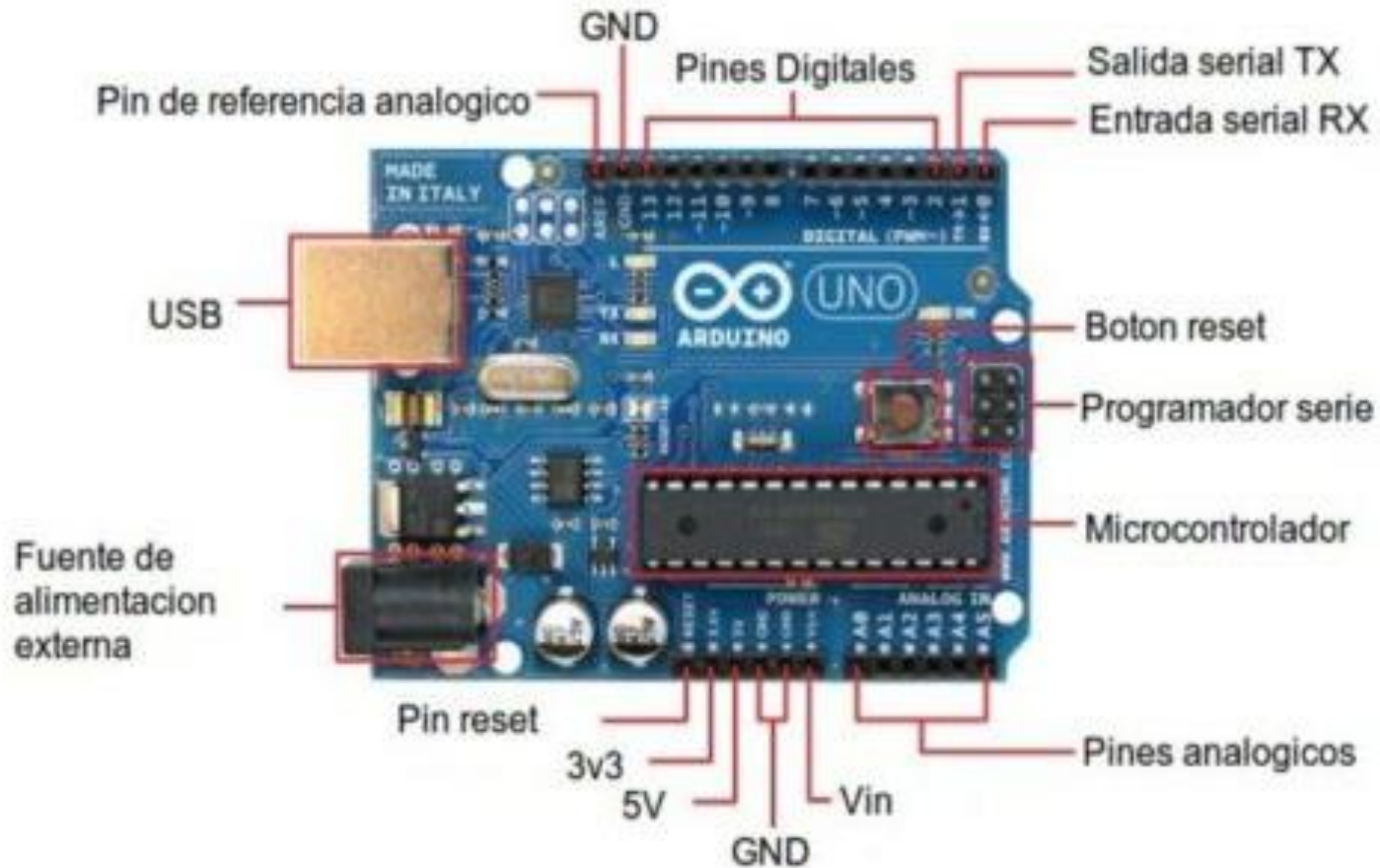
- ❑ Su tamaño es muy inferior a Arduino UNO.
- ❑ Utiliza el microcontrolador AT328
- ❑ Incorpora 14 puertos digitales y 8 analógicos
- ❑ Se utiliza en aplicaciones donde el tamaño o peso sean aspectos importantes (drones, bots)
- ❑ Se alimenta a través de micro USB



Arduino UNO: Especificaciones

										
Fabricante	Arduino	Arduino	Arduino	Arduino	Arduino	Arduino	Arduino	Netduino	Texas Instruments	Fundación Raspberry Pi
Modelo	Pro Mini	Nano	Uno	Mega / Mega 2560	Leonardo	Micro	Due	Netduino 2	Stellaris Launchpad LM4F120	Raspberry Pi Mod.B
Microcontrolador	AVR Atmega 168 ó 328 8bits	AVR ATmega 168 ó 328 8bits	AVR ATmega 328 8bits	AVR ATmega2560 8bits	AVR ATmega 32u4 8bits	AVR ATmega 32u4 8bits	ARM SAM3X8E Cortex-M3 32bits	ARM STM32F2 Cortex-M3 32bits	ARM LM4F120H5QR Cortex-M4 32bits	ARM Broadcom BCM2835
Frecuencia	16Mhz	16Mhz	16Mhz	16Mhz	16Mhz	16Mhz	84Mhz	120Mhz	80Mhz	700Mhz
Memoria RAM	2KIB	2KIB	2KIB	8KIB	2.5KIB	2.5KIB	56KIB (64+32KIB)	60KIB	32KIB	512MIB
Memoria EEPROM	1KIB	1KIB	1KIB	4KIB	1KIB	1KIB	0	0	-	-
Memoria FLASH	16 ó 32KIB	16 ó 32KIB	32KIB	128 ó 256KIB	32KIB	32KIB	512KIB	192KIB	256KIB	-
Pines digitales entradas/salidas	14/14	14/14	14/14	54/54	20/20	20/20	54/54	20/20	43/43	8/8
Tensión/corriente pines digitales	3.3v ó 5v 40mA	5v 40mA	5v 40mA	5v 40mA	5v 40mA	5v 40mA	3.3v 3*15mA (130mA entre todos)	3.3v~5v 25mA (125mA entre todos)	5v	-
Pines analógicos entradas/salidas	6/0	8/0	6/0	16/0	12/0	12/0	12/2	6/0	-	-
Tensión/resolución pines analógicos	3.3v ó 5v 10bits (1024 valores)	5v 10bits (1024 valores)	5v 10bits (1024 valores)	5v 10bits (1024 valores)	5v 10bits (1024 valores)	5v 10bits (1024 valores)	3.3v 12bits (4096 valores)	5v 12bits (4096 valores)	-	-
Pines con interrupción externa	2	2	2	6	2	2	-	-	-	-
Pines PWM	6	6	6	15	7	7	12	6	-	-
Conexiones Serial / UART	1	1	1	4	1	1	4	4	8	Si
Conexiones I2C / TWI	1	1	1	1	1	1	2	1	4	Si
Conexiones ISP / ICSP	1	1	1	1	1	1	1	1	-	Si
Conexión USB	No (necesita adaptador externo)	Si	Si, USB-B	Si, USB-B	Si, Nativa, MicroUSB	Si, Nativa, MicroUSB	Si, Nativa, MicroUSB	Si, Nativa, MicroUSB	Si, Nativa, MicroUSB	Si, MicroUSB
Conexión USB de depuración	No	No	No	No	No	No	Si, MicroUSB	Si, MicroUSB	Si, MicroUSB	-
Conexión Bluetooth	No	No	No	No	No	No	No	No	No	-
Conexión WiFi	No	No	No	No	No	No	No	No	No	-
Conexión Ethernet	No	No	No	No	No	No	No	No	No	Si
Conexión USB Host	No	No	No	No	No	No	Si	No	Si	Si
Almacenamiento por SD	No	No	No	No	No	No	No	No	No	Si
Corriente en el pin de 5v	-	500mA	500~800mA	500~800mA	500~800 mA	500mA	800mA	-	-	-
Corriente en el pin de 3.3v	-	50mA	50mA	50mA	50mA	50mA	800mA	-	-	-
Voltaje de alimentación por el USB	3.3v ó 5v (sin usb)	5v	5v	5v	5v	5v	5v	5v	5v	5v
Voltaje de alimentación recomendado por el Jack	3.35 -12 V (modelo 3.3v) ó 5 - 12 V (modelo 5v)	7~12v	7~12v	7~12v	7~12v	7~12v	7~12v	7.5~9v	-	-
Voltaje de alimentación límite por el Jack	-	6~20v	6~20v	6~20v	6~20v	6~20v	6~20v	-	-	-
Precio oficial	15+g	-	20€+g	40€+g	18€+g	18€+g	39€+g	~35\$+g	13\$+g	~43\$+g
Precio BBB	~4€	~9€	~10€	~12€	11€~	~16€	~38€	25~30€	~15€	~35€

Arduino UNO: Partes



Arduino: Pines digitales

- ❑ Son los pines numerados del 0 al 13.
- ❑ Sirve para enviar o recibir señales en forma digital.
- ❑ Solo pueden tomar valores de 0 o 5V.



Arduino: Pines analógicos

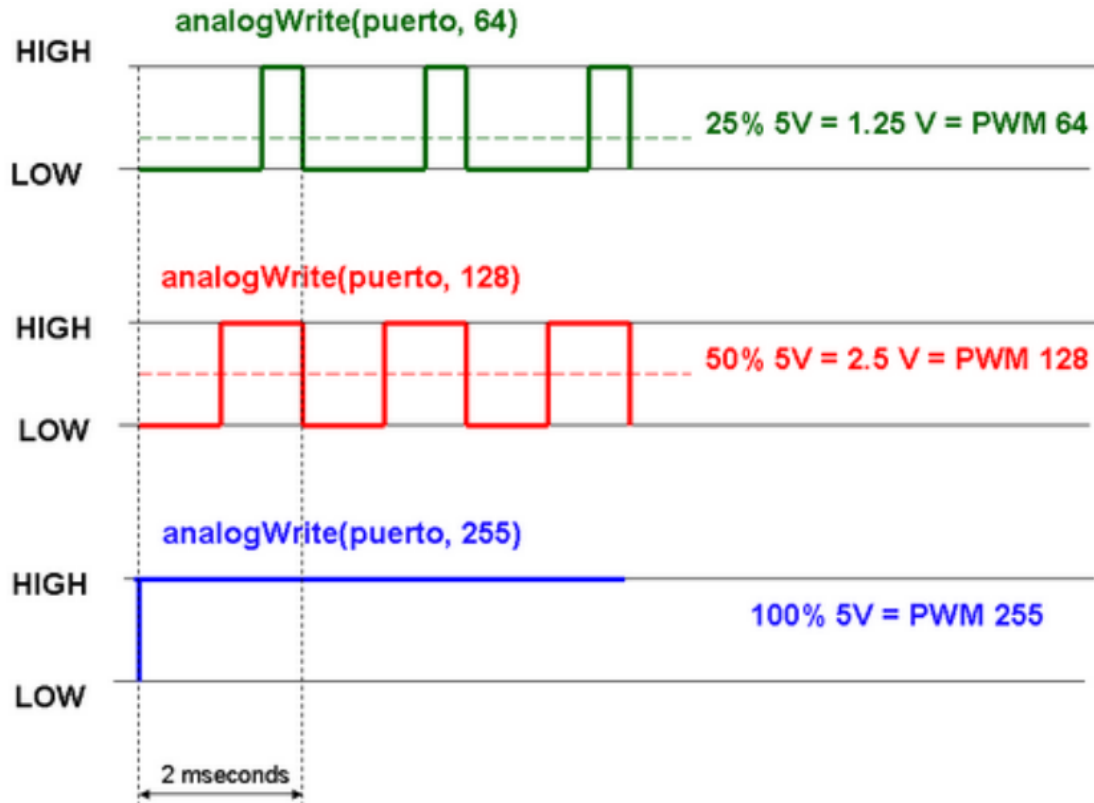
- ❑ Son los pines numerados del A0 al A5
- ❑ Sirve para recibir señales en forma analógica
- ❑ Pueden tomar cualquier valor entre de 0 y 5V.



Arduino UNO

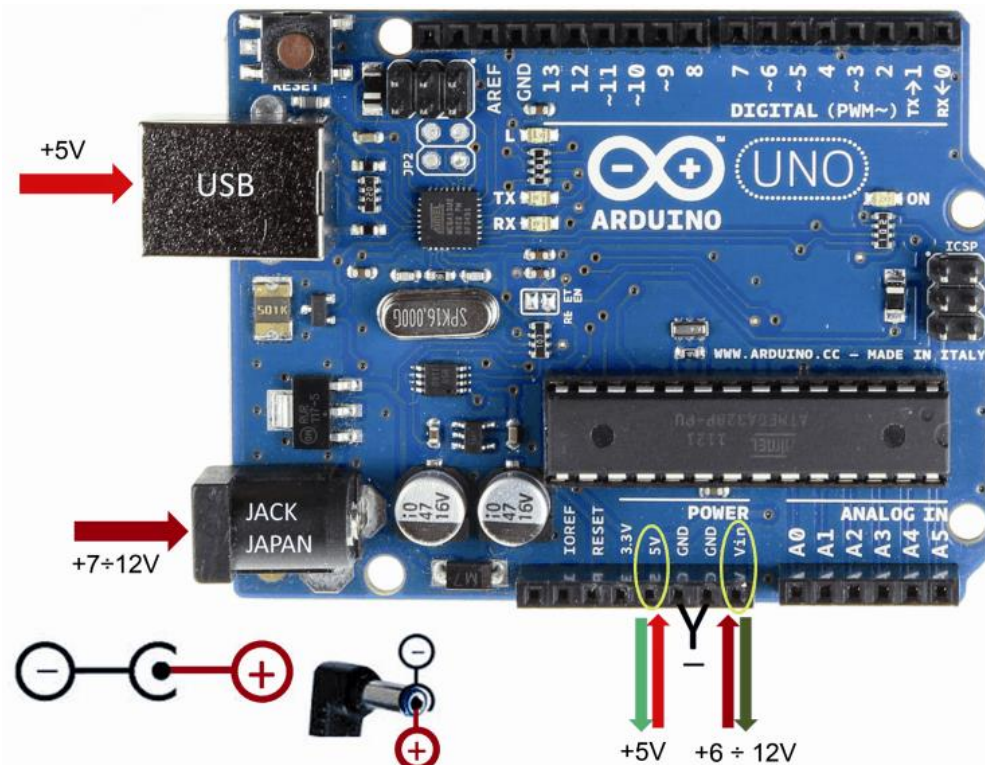
Entradas Analógicas

Arduino: Funcionamiento PWM



Arduino: Alimentación y conexión

- ❑ El conector Jack permite la alimentación entre 7 y 12 V
- ❑ El conector USB permite la alimentación a través de un cargador de móvil o el ordenador (5V)
- ❑ El conector USB también permitirá cargar programas en la placa



Arduino: Cómo se programa

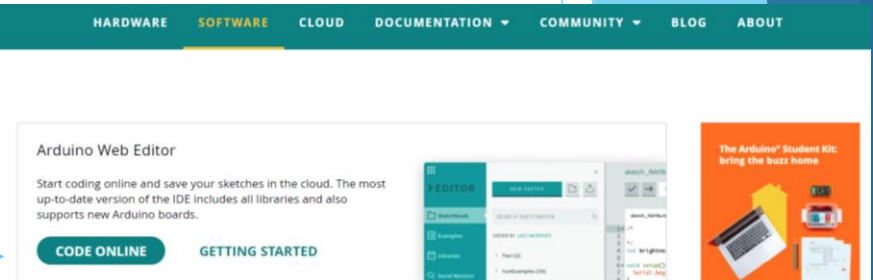
- Existen dos maneras de programar Arduino, mediante código o mediante bloques.

Programación por código	Programación con bloques
- Utiliza un lenguaje de alto nivel muy similar a C++	- Se basa en unir bloques
- Necesitas un IDE para programar aunque también se puede hacer vía web	- Necesitas un programa instalado o desde una aplicación web
- Más complejo de programar, pero más potente.	- Es más sencillo pero sus posibilidades son más limitadas. Es ideal para empezar en clase.
	- Tinkercad, Arduino Blocks, Mblock

Programación por código

- Se accede desde <https://www.arduino.cc/en/software>
- Existen dos opciones de programar:

- A través del editor web



Downloads

- A través del IDE de Arduino

Arduino IDE 1.8.19

The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. This software can be used with any Arduino board.

Refer to the [Getting Started](#) page for Installation instructions.

SOURCE CODE

Active development of the Arduino software is **hosted by GitHub**. See the instructions for **building the code**. Latest release source code archives are available [here](#). The archives are PGP-signed so they can be verified using [this](#) gpg key.

DOWNLOAD OPTIONS

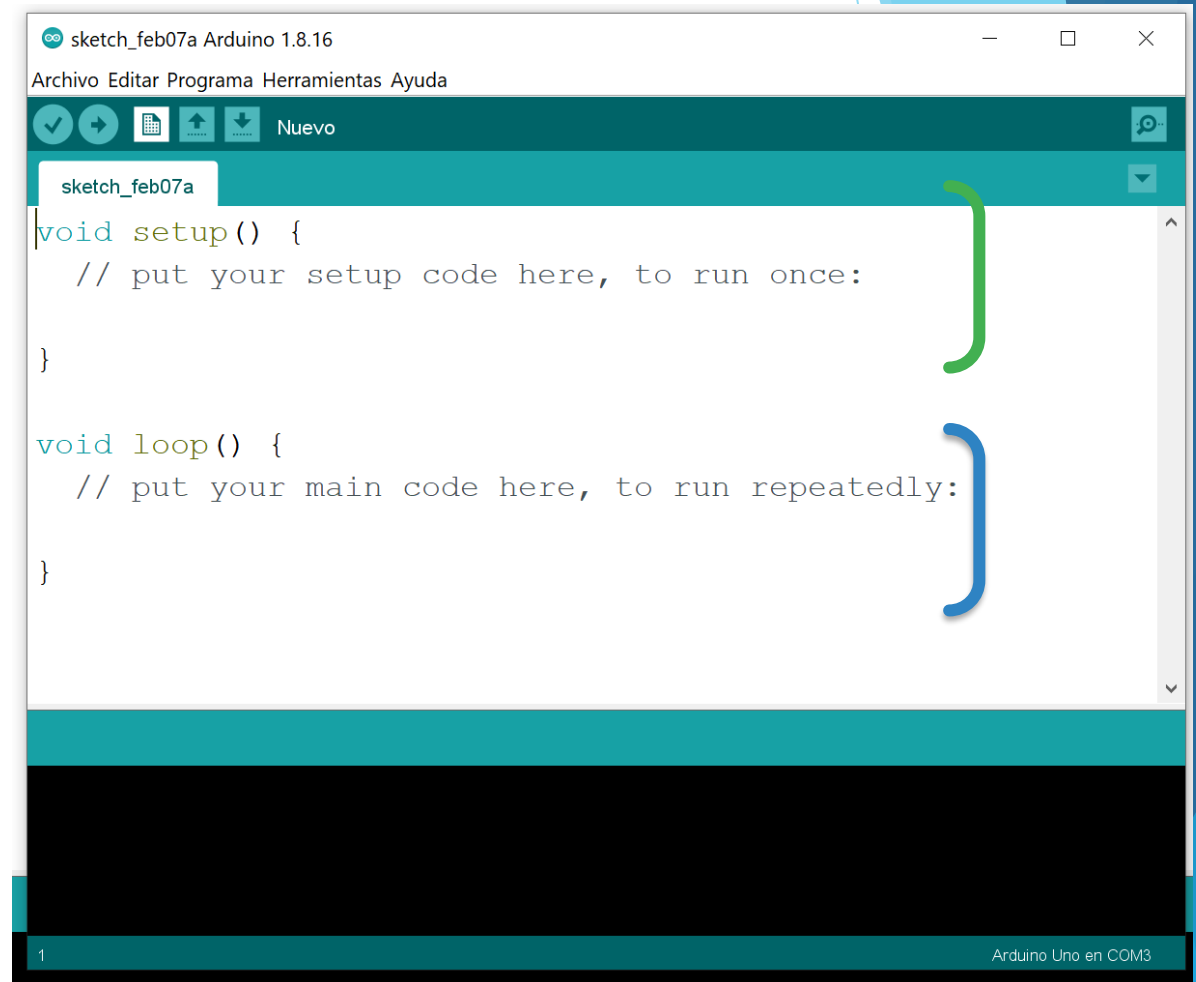
- Windows** Win 7 and newer
- Windows** ZIP file
- Windows app** Win 8.1 or 10 [Get](#)
- Linux** 32 bits
- Linux** 64 bits
- Linux** ARM 32 bits
- Linux** ARM 64 bits
- Mac OS X** 10.10 or newer

Release Notes

Checksums (sha512)

IDE de Arduino

- Estas instrucciones se ejecutan sólo al inicio del programa
- Estas instrucciones de ejecutan repetidamente

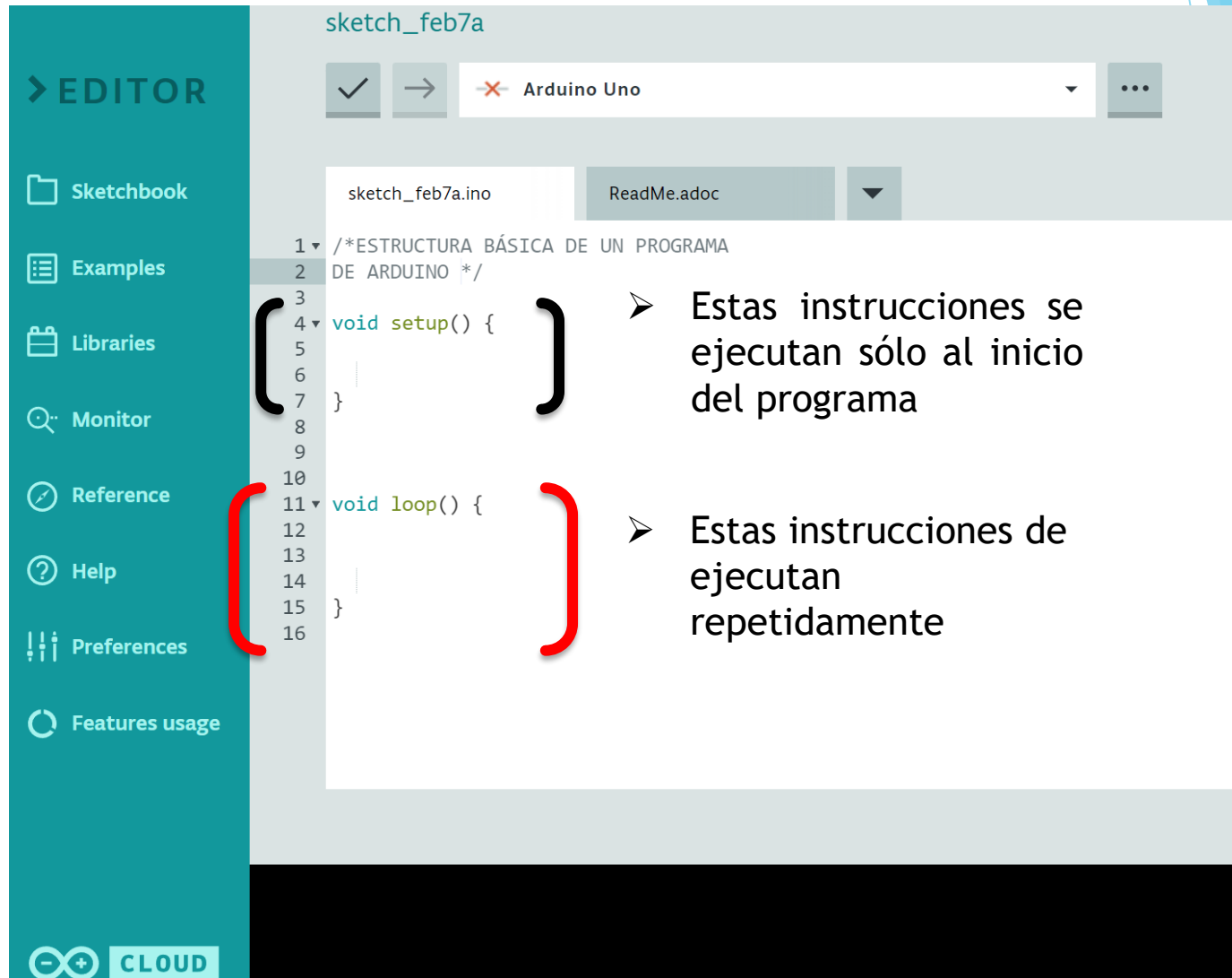


The screenshot shows the Arduino IDE interface. The title bar reads "sketch_feb07a Arduino 1.8.16". The menu bar includes "Archivo", "Editar", "Programa", "Herramientas", and "Ayuda". The toolbar contains icons for "Nuevo" and other functions. The main editor area shows the following code:

```
sketch_feb07a  
void setup() {  
    // put your setup code here, to run once:  
}  
  
void loop() {  
    // put your main code here, to run repeatedly:  
}
```

At the bottom of the IDE, the status bar shows "1" on the left and "Arduino Uno en COM3" on the right.

Editor Web



The screenshot shows the Arduino IDE web editor interface. On the left is a teal sidebar with navigation options: EDITOR, Sketchbook, Examples, Libraries, Monitor, Reference, Help, Preferences, and Features usage. At the bottom of the sidebar is a 'CLOUD' button. The main editor area displays a sketch named 'sketch_feb7a' for an 'Arduino Uno' board. The sketch file 'sketch_feb7a.ino' is open, showing the following code:

```
1 /*ESTRUCTURA BÁSICA DE UN PROGRAMA
2 DE ARDUINO */
3
4 void setup() {
5
6
7 }
8
9
10
11 void loop() {
12
13
14
15 }
16
```

Annotations on the code:

- A black bracket on the left side of lines 4-7 highlights the `void setup()` function.
- A red bracket on the left side of lines 11-15 highlights the `void loop()` function.
- Two callout boxes on the right side explain the functions:
 - A black arrow points from the `void setup()` function to the text: "Estas instrucciones se ejecutan sólo al inicio del programa".
 - A red arrow points from the `void loop()` function to the text: "Estas instrucciones de ejecutan repetidamente".

At the bottom left of the editor, there is a 'CLOUD' button with a refresh icon.

Programación por código

Definiciones importantes

- ❑ **Constantes:** datos cuyo valor no cambia durante la ejecución del programa.
- ❑ **Variables:** son datos que toman cierto valor y que puede ser modificado a lo largo del programa. En Arduino al definir las variables se le asigna un valor y un tipo.

Tipos de Datos	Memoria que ocupa	Rango de valores
boolean	1 byte	0 o 1 (True o False)
byte / unsigned char	1 byte	0 – 255
char	1 byte	-128 – 127
int	2 bytes	-32.768 – 32.767
word / unsigned int	2 bytes	0 – 65.535
long	2 bytes	-2.147.483.648 – 2.147.483.647
unsigned long	4 bytes	0 – 4.294.967.295
float / double	4 bytes	-3,4028235E+38 - 3,4028235E+38
string	1 byte + x	Array de caracteres
array	1 byte + x	Colección de variables

Programación por código

Operadores:

- ❑ **Operadores lógicos:** sirven para establecer comparaciones o condiciones, de tal forma que si se cumple devuelve TRUE y si no FALSE. Son:

	Operadores lógicos
==	Igual
!=	Distinto
<	Menor que
>	Mayor que
<=	Menor o igual que
>=	Mayor o igual que

	Booleanos
&&	AND
	OR

- ❑ **Operadores matemáticos:** se utilizan para manipular valores, son: SUMA (+), RESTA (-), MULTIPLICACIÓN(*), DIVISIÓN (/).

Programación por código

- ❑ **Comentarios:** sirve para introducir aclaraciones en el código escrito.
 - // → si el texto está en una misma línea
 - /* comentario */ → si el texto está en distintas líneas.
- ❑ **Funciones:** son códigos ya escritos que pueden ser utilizados escribiendo el nombre de la función. Por ejemplo: *digitalRead()* sirve para leer el estado de un pin digital. Las iremos viendo a lo largo de los programas.
- ❑ **Librerías:** son un conjunto de funciones agrupadas en un fichero. Se escriben al principio del programa (precedidas del símbolo #)

Programación por código

Del mismo modo conviene conocer ciertas estructuras de programación:

- ❑ **Estructuras de control selectivo:** denominadas condicionales, porque se utilizan para la toma de decisiones. Son *if*, *else*, *switch*.
- ❑ **Estructuras de control iterativas:** denominadas bucles, porque permiten su ejecución repetitiva de sentencias durante un número determinado de veces, controlado y definido por un algoritmo. Son *while*, *for* y *do while*.

Prácticas de Arduino

PRÁCTICA 1: ENCENDIDO DE UN LED Y PARPADEO

PRÁCTICA 2: SEMÁFORO

PRÁCTICA 3: ENCENDIDO SECUENCIAL DE LEDS

PRÁCTICA 4: PULSADORES

PRÁCTICA 5: CONMUTADA

PRÁCTICA 6: LED MULTICOLOR

PRÁCTICA 7: SIMULACIÓN DE LA LUZ DE UNA VELA

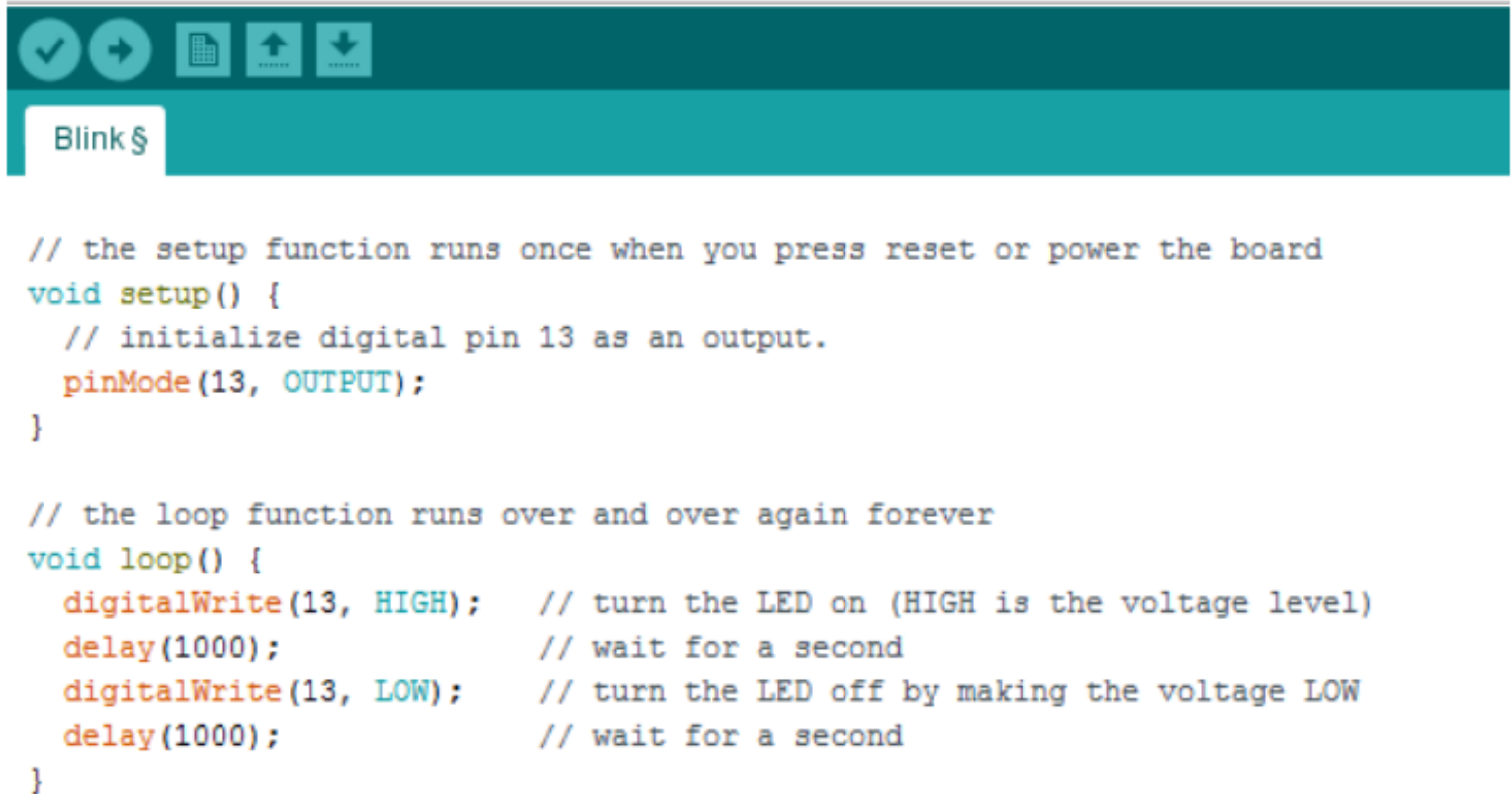
PRÁCTICA 8: ENCENDIDO DE LUZ AUTOMÁTICO

PRÁCTICA 9: DADO ELECTRÓNICO

Práctica 1: Encendido de led y parpadeo

- ❑ Para empezar a trabajar con Arduino vamos a simular un ejemplo de código llamado BLINK, en el que veremos cómo se enciende y apaga el diodo LED que incluye la placa en el pin 13.

Sketch:



```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(13, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(13, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000); // wait for a second
  digitalWrite(13, LOW); // turn the LED off by making the voltage LOW
  delay(1000); // wait for a second
}
```

Práctica 1: Encendido de led y parpadeo

- ❑ Vamos a modificar el programa anterior para que el tiempo cada vez sea mayor, dependiendo del número de veces que se ejecute el programa.

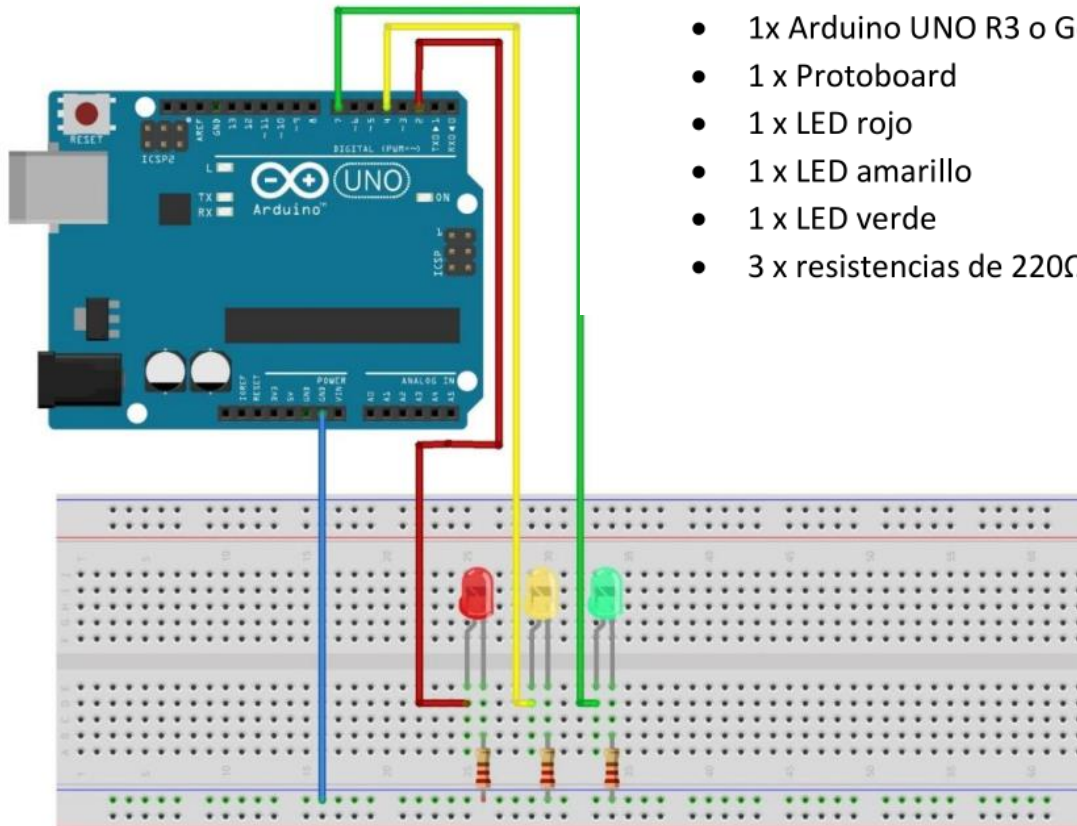
Sketch:

```
Blink-aumento_apagado $  
  
int ledPin=13;  
int n=0; // entero que contará el paso por la función loop  
void setup() {  
    // initialize digital pin 13 as an output.  
    pinMode(ledPin, OUTPUT);  
}  
  
// the loop function runs over and over again forever  
void loop() {  
    digitalWrite(ledPin, HIGH); //Enciendo el LED  
    delay(1000);                // Espero un segundo  
    digitalWrite(ledPin, LOW); //Apago el LED  
    n=n+100; // Incrementamos n  
    delay(n); // Pausa de un tiempo n  
}
```

Práctica 2: Semáforo

- ❑ Vamos a realizar un semáforo que tenga la siguiente secuencia de luces: luz roja durante 6 segundos y se apaga, luz verde durante 4 segundos y luz amarilla 2 segundos con dos destellos.

Esquema:



Material necesario

- 1x Arduino UNO R3 o Genuino y cable de conexión al PC
- 1 x Protoboard
- 1 x LED rojo
- 1 x LED amarillo
- 1 x LED verde
- 3 x resistencias de 220Ω

Práctica 2: Semáforo

```
p3-semaforo $
```

```
int rojo=7;//led rojo
int amarillo=4;//led amarillo
int verde=2;//led verde
void setup() {
  pinMode(7,OUTPUT);//configuro los pines de los LED como salidas
  pinMode(4,OUTPUT);//configuro los pines de los LED como salidas
  pinMode(2,OUTPUT);//configuro los pines de los LED como salidas
  // put your setup code here, to run once:

}

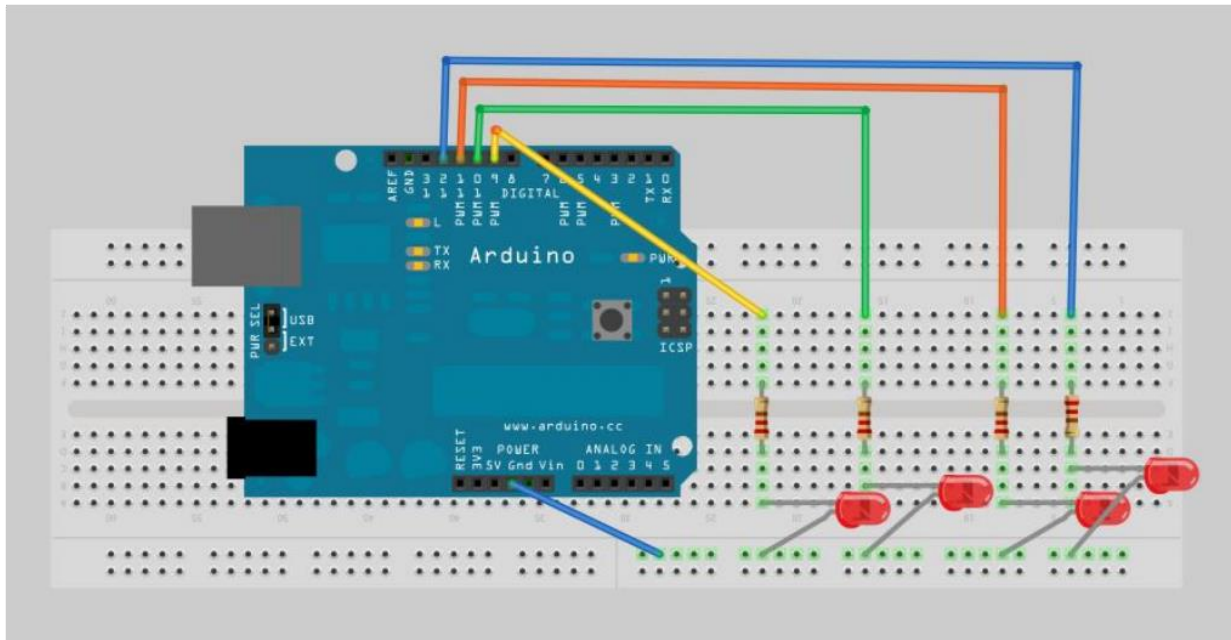
void loop() {
  digitalWrite(7,HIGH);
  delay(6000);//el tiempo está en ms, es decir, espero 6segundos
  digitalWrite(7,LOW);//Apago el LED rojo
  digitalWrite(2,HIGH);//Enciende verde
  delay(4000);
  digitalWrite(2,LOW);//Apaga verde
  digitalWrite(4,HIGH);//Enciende amarillo
  delay(2000);
  digitalWrite(4,LOW);//Apaga amarillo
  /*digitalWrite(4,HIGH);//Enciende amarillo para hacer un parpadeo a mayores
  delay(2000);
  digitalWrite(4,LOW);//Apaga amarillo
  delay(2000);*/
}
```


Práctica 3: Encendido secuencial de leds

- ❑ Realiza el encendido secuencial de cuatro leds empezando del pin 12 hasta el pin 9. Cuando se hayan encendido todos los LEDs se apagarán los 4 y se volverá a repetir el ciclo. Vamos a considerar un tiempo entre el encendido de cada LED y el apagado de los 4 LEDS de 1 segundo.

Material necesario

- 1x Arduino UNO R3 o Genuino y cable de conexión al PC
- 1 x Protoboard
- 4 x LED
- 4 x resistencias de 220Ω



Práctica 3: Encendido secuencial de Leds

```
P5-Encendido_secuencial-apagado_todo_a_la_vez$
```

```
int led1=9;
int led2=10;
int led3=11;
int led4=12;

void setup() {
  pinMode(led1,OUTPUT);
  pinMode(led2,OUTPUT);
  pinMode(led3,OUTPUT);
  pinMode(led4,OUTPUT);
}

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

Práctica 3: Encendido secuencial de Leds

Variación del Sketch: podemos simplificar el código utilizando la orden FOR, como se muestra en el siguiente sketch.

```
P5-Encendido_secuencial-apagado_secuencial$
```

```
int tiempo=1000;
int n;
int cont;

void setup() { //comienza la configuración de los pines
  for (n=9; n<13;n++){
    pinMode (n,OUTPUT); //desde el pin 9 hasta el 13, sin incluirlo
  }
}

void loop() {
  for (cont=0;cont<4;cont++){
    for (n=9;n<13;n++){ //para cada pin del 9 al 12
      digitalWrite (n,HIGH); //Encendemos
      delay (tiempo); //esperamos
    }
    for (n=9;n<13;n++){ //para cada pin del 9 al 12
      digitalWrite (n,LOW); //Apagamos
      digitalWrite (n,LOW); //apagamos
    }
    delay(tiempo); //esperamos
  }
}
```

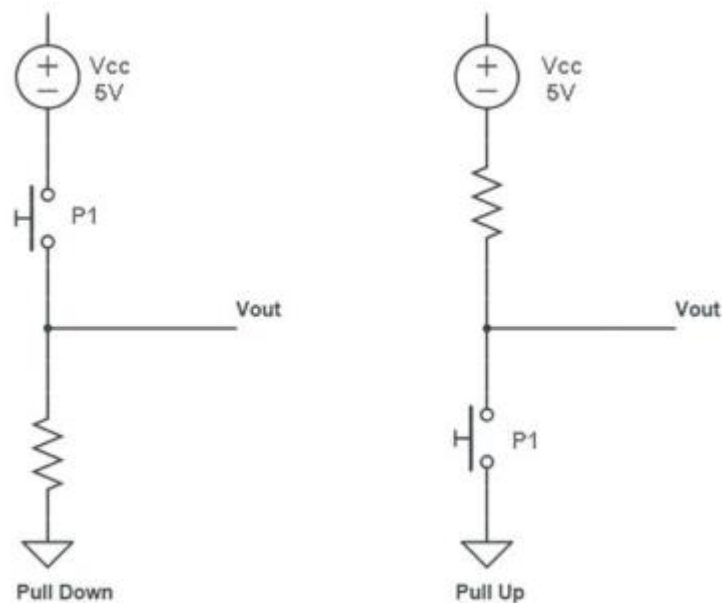
a) Modifica el programa anterior para que se apaguen todos los LED a la vez.

b) Modifica el sketch anterior para que los LED se apaguen en orden inverso: del pin 9 al 12.

Práctica 4: Pulsadores. Pull up y Pull down

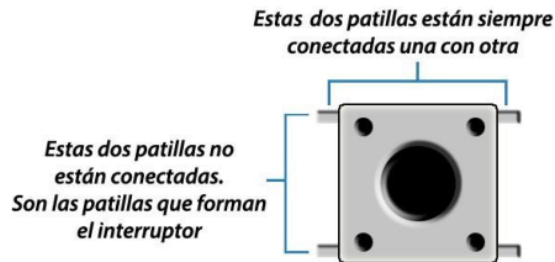
- ❑ Para conectar pulsadores en las entradas digitales de Arduino, se utiliza la conexión mediante resistencias pull-down o pull up (como veremos sólo cambia el circuito, el software que utilizamos para hacer funcionar el circuito no va a variar), de tal forma, que siempre haya un estado lógico a la entrada del microprocesador, es decir, 5 o 0 voltios.

Las configuraciones se muestran en la siguiente figura:



Práctica 4: Pulsadores. Pull up y Pull down

Un pulsador va a tener cuatro patillas. Podemos recurrir a un polímetro para comprobar las conexiones de continuidad o bien fijarnos en su forma, tal y como se muestra en la siguiente imagen:



Ahora vamos a realizar un montaje con resistencia Pull-up y pull-down para analizar su funcionamiento.

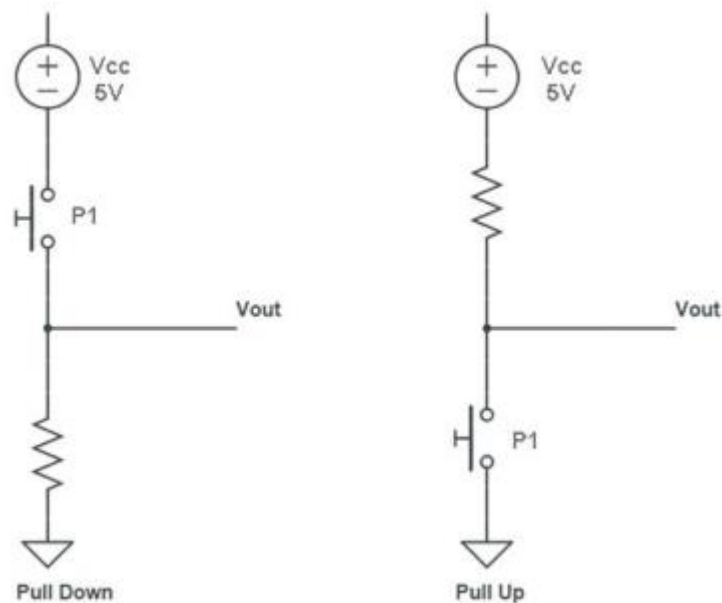
Material necesario:

- 1x Arduino UNO R3 o Genuino y cable de conexión al PC
- 1 x Protoboard
- 1 x LED rojo
- 1 x resistencias de 220Ω
- 1 x pulsador
- 1x resistencia $10k\Omega$
- Cables de conexión

Práctica 4: Pulsadores. Pull up y Pull down

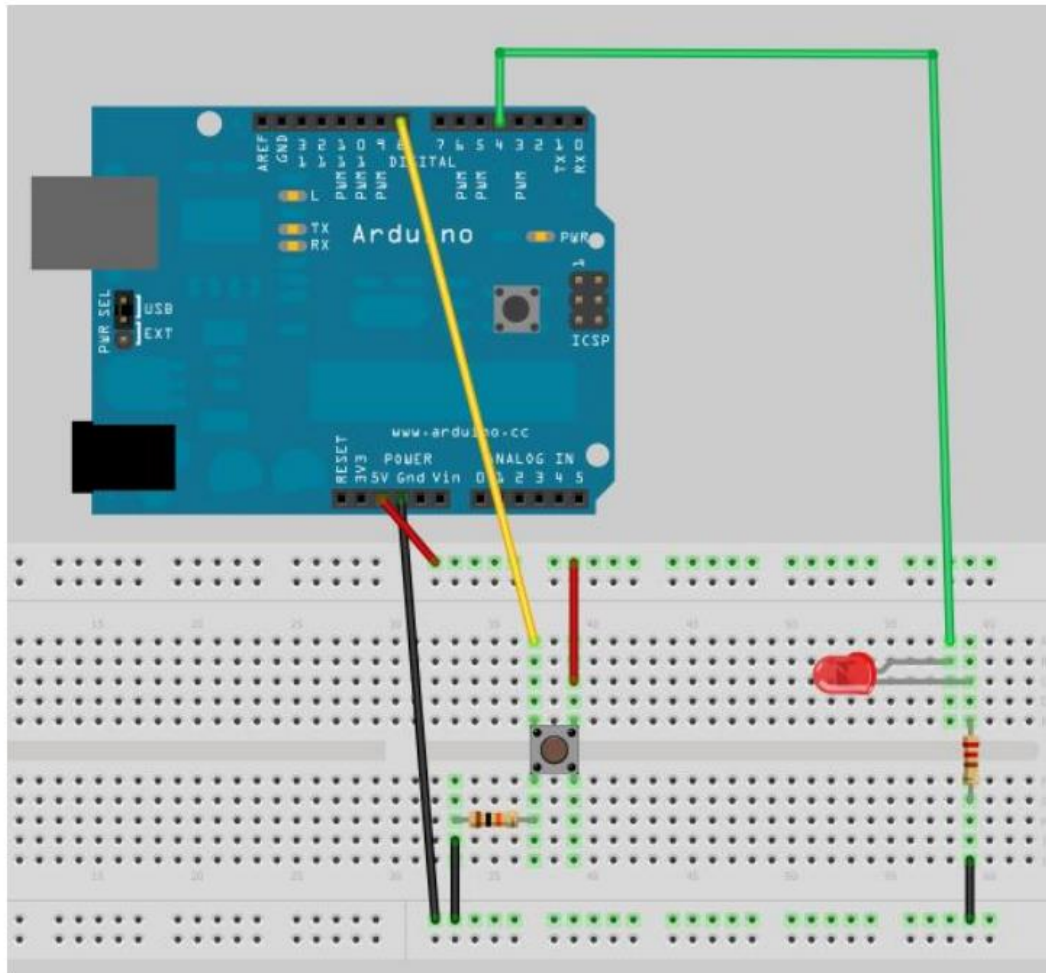
- ❑ Para conectar pulsadores en las entradas digitales de Arduino, se utiliza la conexión mediante resistencias pull-down o pull up (como veremos sólo cambia el circuito, el software que utilizamos para hacer funcionar el circuito no va a variar), de tal forma, que siempre haya un estado lógico a la entrada del microprocesador, es decir, 5 o 0 voltios.

Las configuraciones se muestran en la siguiente figura:



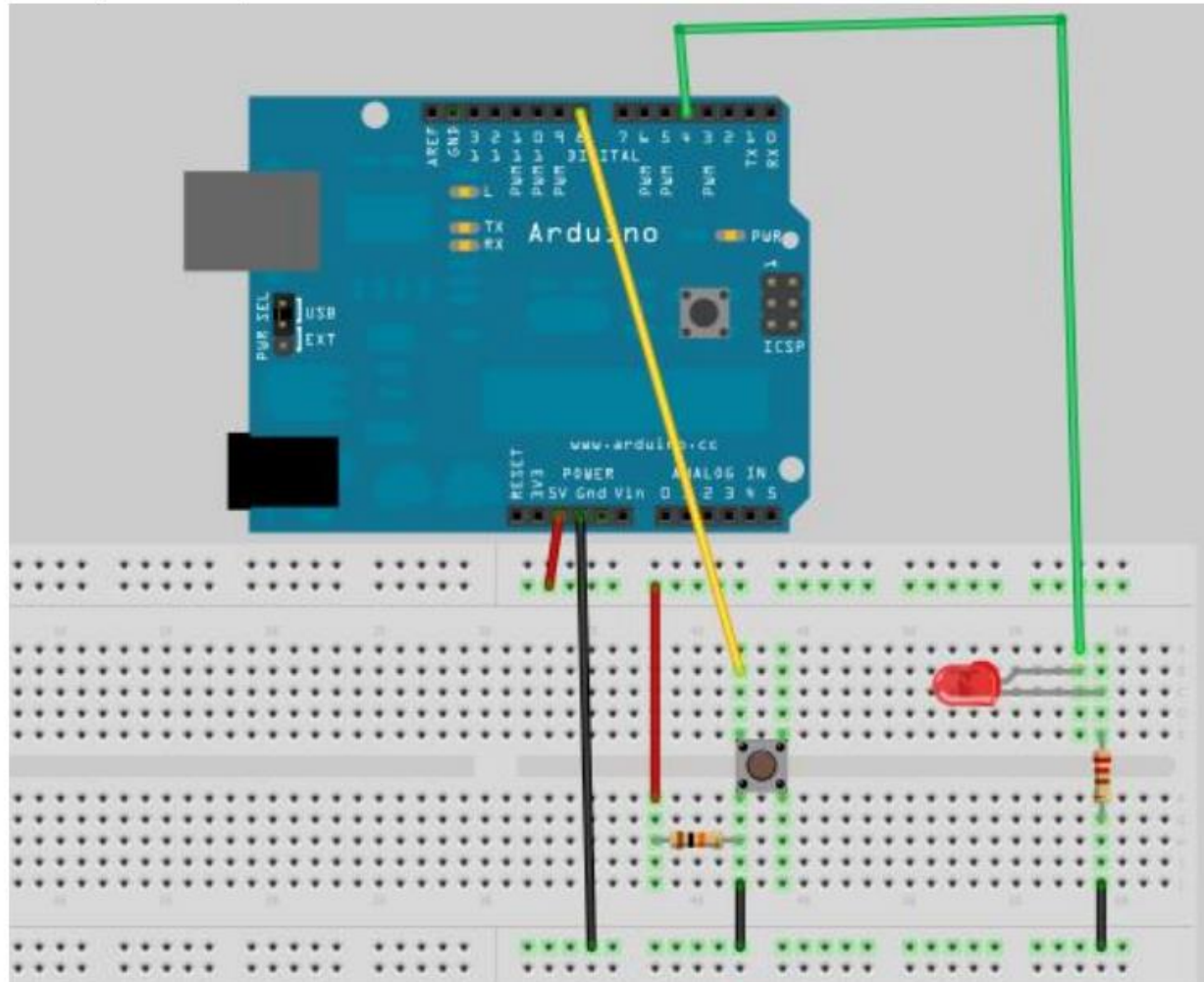
Práctica 4: Pulsadores. Pull up y Pull down

- ❑ Pull down: Detectamos que el pulsador se activa



Práctica 4: Pulsadores. Pull up y Pull down

- ❑ Pull up: Detectamos que el pulsador no está activado



Práctica 4: Pulsadores. Pull up y Pull down

- ❑ El siguiente programa encenderá un led al activar el pulsador.

```
p2-encendidoled_con_pulsador$
```

```
int estado=0; //defino el valor del botón, en principio no pulsado
void setup() {
  pinMode (8,INPUT); //Declaramos el botón como entrada
  pinMode (4,OUTPUT); //Declaramos el LED como salida
}

void loop() {
  estado=digitalRead(8); //leemos el valor de la entrada
  if(estado == 1){
    digitalWrite(4,HIGH); // si el estado es HIGH encendemos la salida
  }
  else{
    digitalWrite(4,LOW); //si no esta en HIGH apagamos el LED
  }
  // put your main code here, to run repeatedly:
}
```

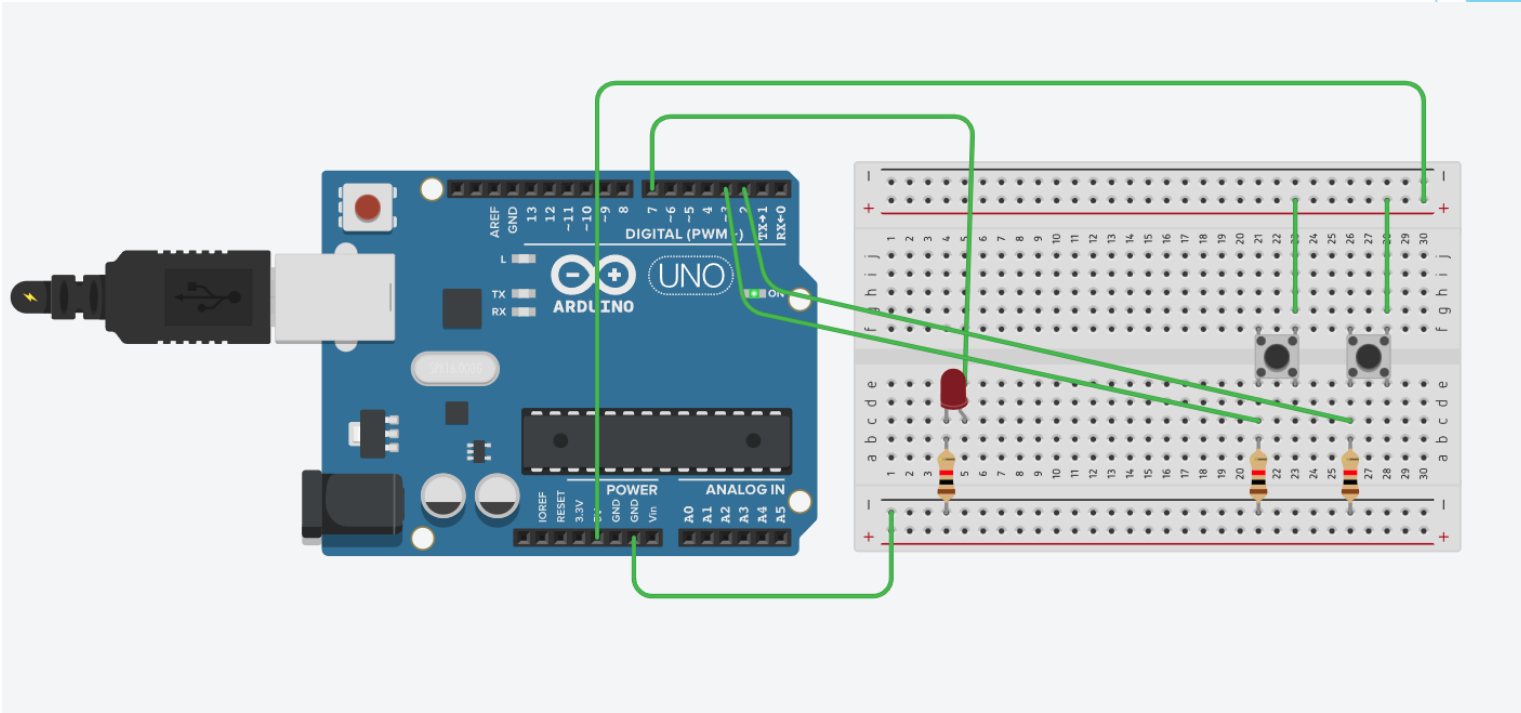
Práctica 5: Conmutada

- ❑ Se desea controlar el encendido de un LED conectado al pin 13 mediante dos pulsadores conectados en los pines 2 y 3. El funcionamiento va a ser similar a una conmutada instalada en una vivienda (control de un punto de luz desde dos puntos); es decir, al pulsar cualquiera de los dos pulsadores, si el LED está encendido, se apagará; y si está apagado, se encenderá. Así pues, con los dos pulsadores puedo controlar el encendido y apago del LED.

Material necesario

- 1x Arduino UNO R3 o Genuino y cable de conexión al PC
- 1 x Protoboard
- 1 x LED
- 1x resistencia de 220Ω
- 2 x pulsadores
- 2 x resistencias $10k\Omega$
- Cables de conexión

Práctica 5: Conmutada



Práctica 5: Conmutada

P5_conmutada_curso_arduino §

```
int estado = 0;
void setup()
{
  pinMode(7, OUTPUT);
  pinMode(2, INPUT);
  pinMode(3, INPUT);
}

void loop()

{

  if (digitalRead(2)==1 || digitalRead(3)==1) {
    if (estado == 0){
      estado = 1;
      digitalWrite(7, HIGH);

    }

    else {

      estado = 0;
      digitalWrite(7,LOW);
    }

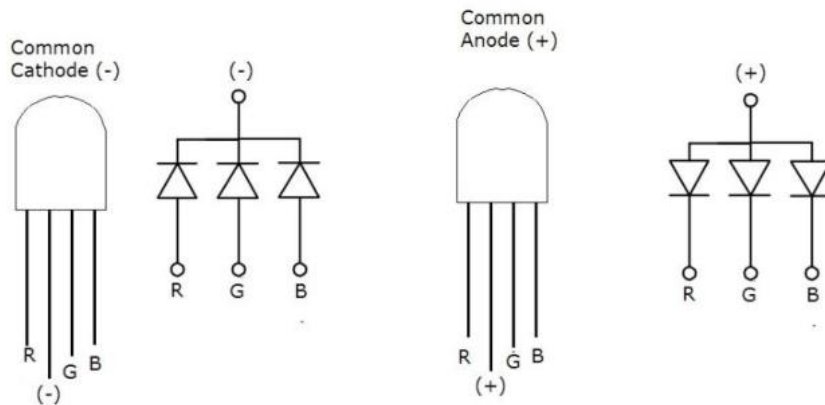
    while(digitalRead(2)==1 || digitalRead(3) ==1){}

  }

  delay(50);
}
```

Práctica 6: Led multicolor

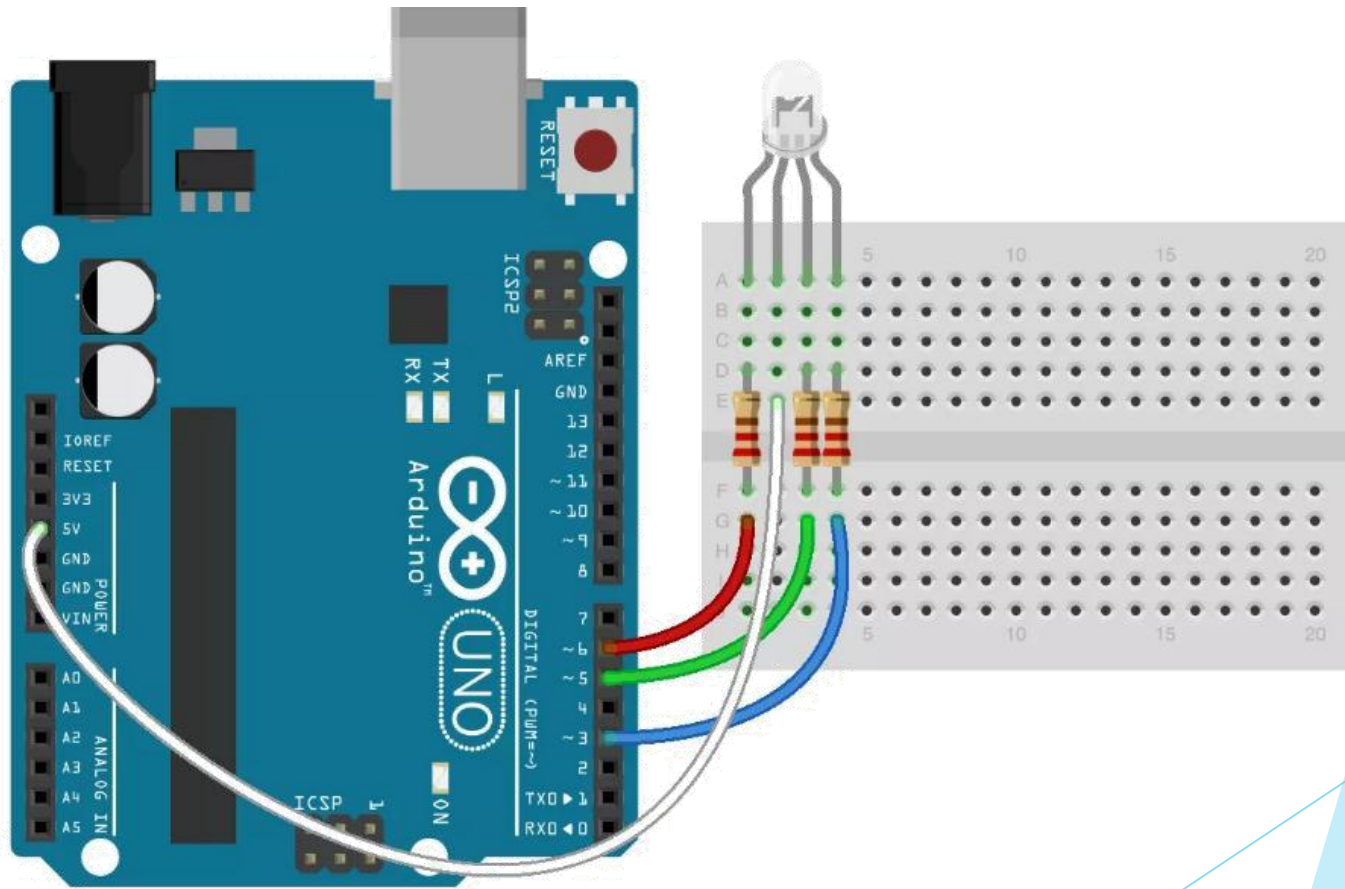
- ❑ En este caso vamos a realizar el montaje de un diodo RGB, LED multicolor. El RGB tiene 4 terminales y su conexión se muestra en la figura. Con dos pines se pueden obtener diversos colores, por ejemplo: Amarillo = Rojo + Verde, Magenta = Rojo + Azul. Así. podríamos sustituir los 3 LED del semáforo anterior por éste.



Material necesario

- 1x Arduino UNO R3 o Genuino y cable de conexión al PC
- 1 x Protoboard
- 1 x RGB multicolor
- 3x resistencia de 220Ω
- Cables de conexión

Práctica 6: Led multicolor

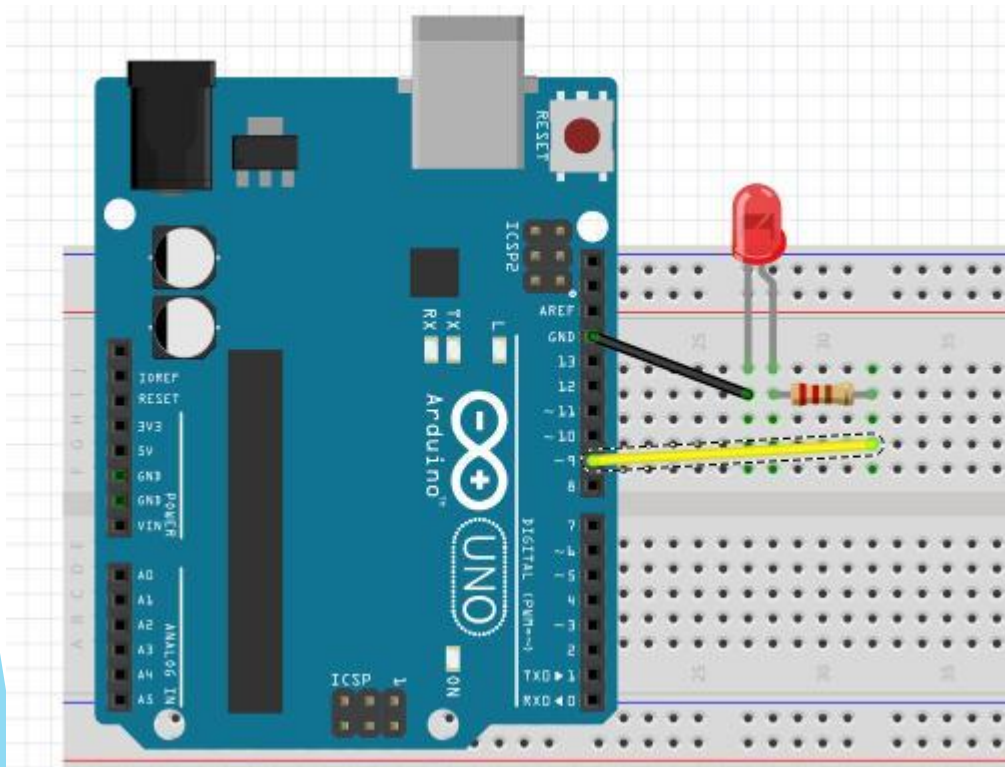


Práctica 6: Led multicolor

```
p7-led_rgb
/* led RGB anodo comun */
int rojo=4;//Aisgnamos las variables
int verde=3;
int azul=2;
int retardo=2000;
void setup() { //configuramos los pines
  pinMode(rojo,OUTPUT);
  pinMode(azul,OUTPUT);
  pinMode(verde,OUTPUT);
}
void loop() {
  digitalWrite(rojo,HIGH);//Enciendo rojo
  digitalWrite(verde,LOW);
  digitalWrite(azul,LOW);
  delay(retardo);
  digitalWrite(azul,HIGH);// al añadir azul se convierte en magenta
  delay(retardo);
  digitalWrite(rojo,LOW);// apago rojo, nos quedamos en AZUL
  delay(retardo);
  digitalWrite(verde,HIGH);//Añado verde, se convierte en cian
  delay(retardo);
  digitalWrite(azul,LOW);//Apago azul, me quedo en verde
  delay (retardo);
  digitalWrite(rojo,HIGH);//Añado rojo, obtengo amarillo
  delay(retardo);
  digitalWrite(azul,HIGH);//Añado azul, obtengo blanco
  delay(retardo);
}
```


Práctica 7: Simulación de luz de vela mediante PWM

- ❑ Se trata de simular el movimiento de la llama de una vela. Hacemos uso de la instrucción para generar un número aleatorio que lo asignamos a una salida analógica PWM y otro número que lo asociamos a la variable de temporización (tiempo que esperamos a para cambiar el valor de la salida).



Materiales:

- 1x tarjeta UNO R3
- 1x diodo LED
- 1x resistencia 220Ω
- Cables de conexión

Práctica 7: Simulación de luz de vela mediante PWM

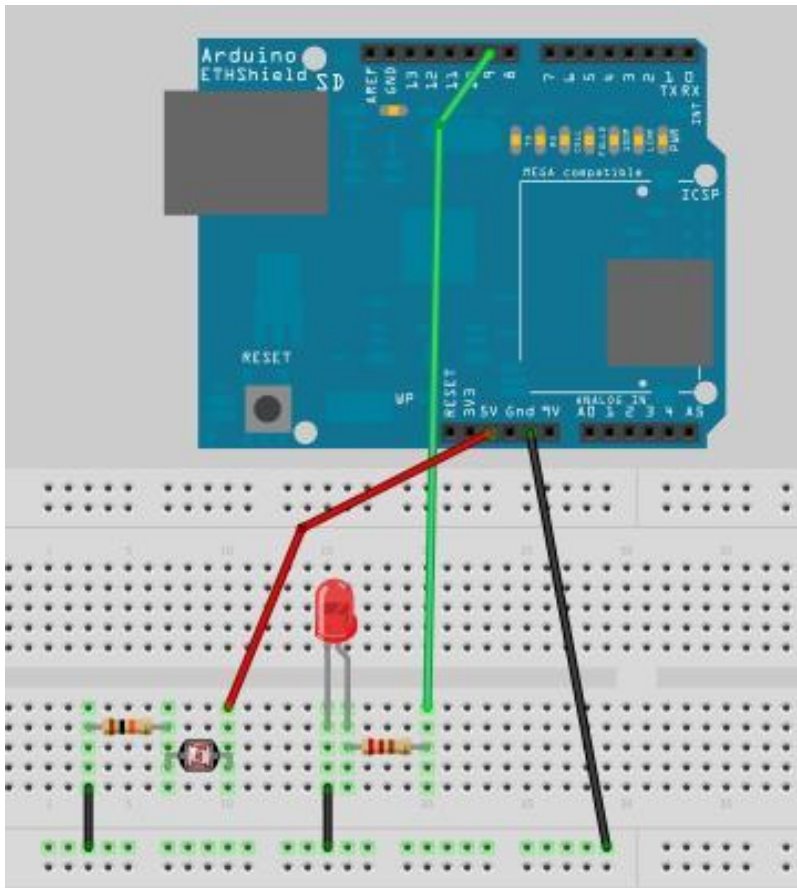
```
p9-simulador_vela $
```

```
int led=9;//selecciona el puerto PWM numero 9
int val=0;//define y pone a cero la variable "brillo"
int delayval=0;//define el intercambio de valor de salida
void setup() {
    pinMode(led, OUTPUT);
}

void loop() {
    val=random(100,255); //genera un número aleatorio entre 100 y 255
    analogWrite(led,val); //enviamos ese valor a la salida 9
    delayval=random(50,150);//generamos un numero aleatorio entre 50 y 150
    delay(delayval);//esperamos el tiempo delayval en milisegundos
}
```

Práctica 8: Encendido de luz automático

- ❑ En esta práctica vamos a implementar un programa de encendido automático de luz, donde se enciende o paga un LED en función de la cantidad de luz que recibe una LDR.



Material necesario

- 1x Arduino UNO R3
- 1 x Protoboard
- 1 x resistencia LDR
- 1 x diodo LED
- 1 x resistencia de 220Ω
- 1 x resistencia de 10KΩ
- Cables de conexión

Práctica 8: Encendido de luz automático

p8-Interruptor_crepuscular §

```
int LDR=0;//pin conectado a la LDR
int LED=9;//Pin conectado al LED
int luminosidad; // variable para almacenar la lectura de la luminosidad
int umbral=600;//valor umbral a partir del cual conmuta el LED
void setup() {
  Serial.begin(9600);//se inicia el puerto serie
  pinMode (LED, OUTPUT);
  digitalWrite(LED,LOW); //comienza el LED apagado
}
void loop() {
  luminosidad=analogRead(LDR);//Leemos el pin de entrada
  Serial.println(luminosidad);//escribimos por el puerto serie el valor de la lectura
  if (luminosidad>umbral){
    digitalWrite(LED,LOW);//Si la luminosidad es mayor, apagamos el LED
  }
  else{
    digitalWrite(LED,HIGH);}
  delay (200);
}
```

Práctica 8: Encendido de luz automático

- ❑ Vamos a modificar el interruptor crepuscular anterior, de tal forma que el diodo se encienda condiferente luminosidad, en función de la luz que reciba la LDR. Por ejemplo:
 - A más de 700 el LED estará apagado
 - Entre 550 y 700 el LED lucirá al 30%
 - A menos de 550 el LED lucirá al 100%

- ❑ Para realizar este ejercicio es necesario que consideremos el LED como una salida analógica, para poder variar su valor de salida. EL montaje es el mismo que en el caso anterior.

Práctica 8: Encendido de luz automático

p8-Interrupcion_crepuscular-2umbrales §

```
int LDR=0;//pin conectado a la LDR
int LED=9;//Pin conectado al LED
int luminosidad; // variable para almacenar la lectura de la luminosidad
int umbral1=700;
int umbral2=550;
void setup() {
  Serial.begin(9600);//se inicia el puerto serie
  pinMode (LED, OUTPUT);
  analogWrite(LED,0); //comienza el LED apagado
}
void loop() {
  luminosidad=analogRead(LDR);//Leemos el pin de entrada
  Serial.println(luminosidad);//escribimos por el puerto serie el valor de la lectura
  if (luminosidad>umbral1){
    analogWrite(LED,0);//Si la luminosidad es mayor, apagamos el LED
  }
  else{
    if (luminosidad<umbral2){
      analogWrite(LED,255);}
    else{
      analogWrite(LED,80);}
  }
  delay (200);
}
```

Práctica 9: Generación de números aleatorios

- ❑ Esta práctica nos va a servir para ver como Arduino genera números aleatorios mediante la función random()

```
p10_aleatorios
int numero =0;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
}

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

  randomSeed(analogRead(0));
  numero=random(0,100);
  Serial.println(numero);

  delay(1000);
}
```

```
p10_aleatorios §
int numero =0;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
}

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

  //randomSeed(analogRead(0));
  numero=random(0,100);
  Serial.println(numero);

  delay(1000);
}
```


Práctica 10: Dado electrónico

- ❑ En este caso vamos a implementar un dado a base de leds que nos puede ser útil para nuestros juegos favoritos de mesa.
- ❑ Nuestro dado electrónico está formado por 7 leds, dispuestos como se muestran en la imagen, de tal forma que se puedan representar los números del 1 al 6, según encendamos unos LEDs u otros.

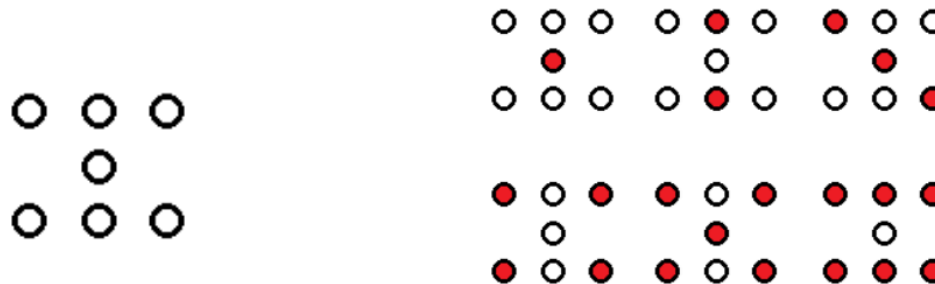


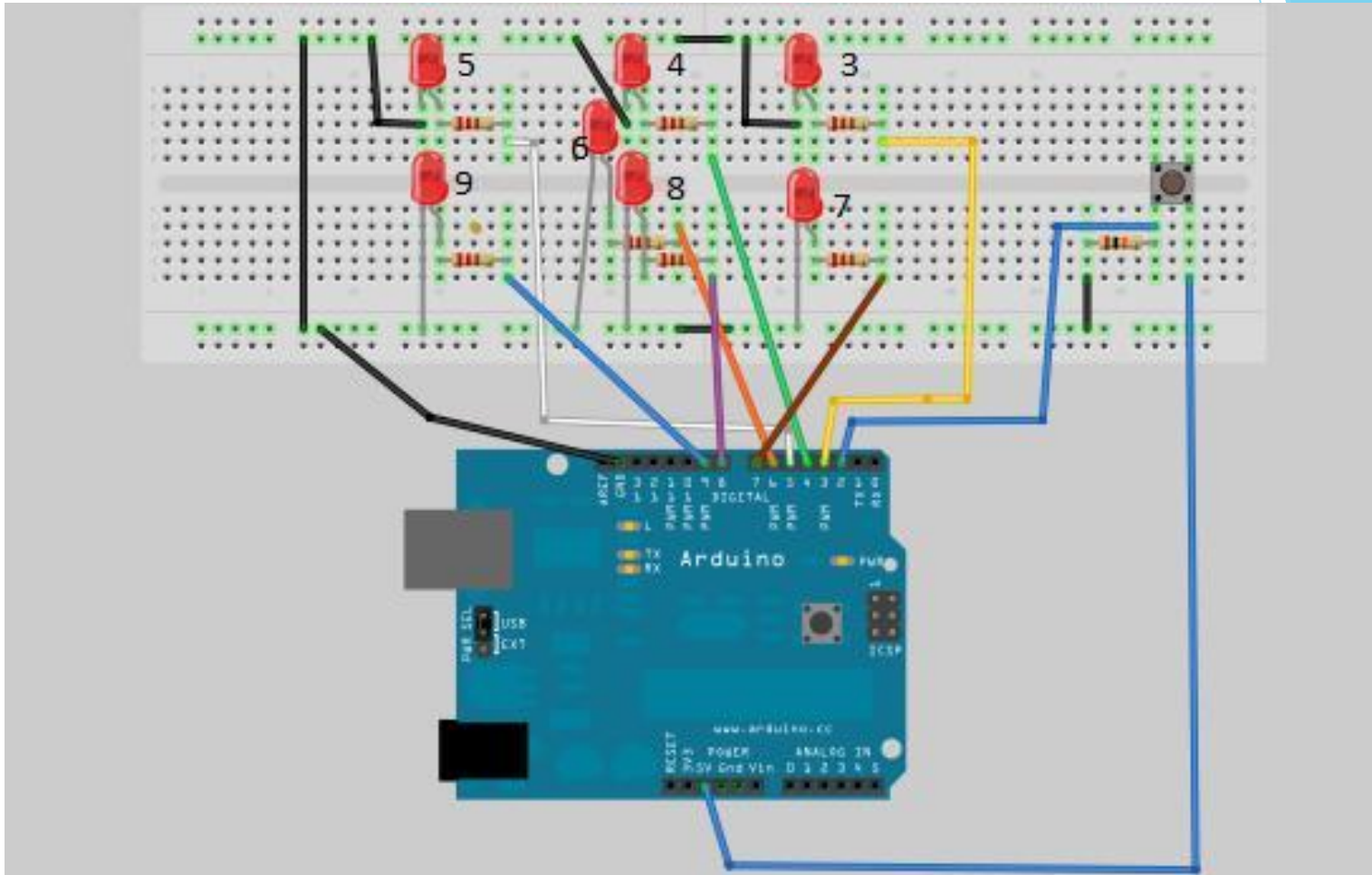
Ilustración 1-Disposición de los leds y números representados con LEDs

Material necesario

- 1x Arduino UNO R3 o Genuino y cable de conexión al PC
- 1 x Protoboard
- 7 x diodo LED
- 7 x resistencia de 220Ω
- 1 x pulsador
- 1 x resistencia de $10K\Omega$
- Cables de conexión

Práctica 10: Dado electrónico

□ Esquema del montaje



Práctica 10: Dado electrónico

❑ Código

p9-dado_electr_nico

```
/*dado electrónico*/
void setup() {
  //iniciamos los pines de los diodos LED
  pinMode(3,OUTPUT);
  pinMode(4,OUTPUT);
  pinMode(5,OUTPUT);
  pinMode(6,OUTPUT);
  pinMode(7,OUTPUT);
  pinMode(8,OUTPUT);
  pinMode(9,OUTPUT);

  pinMode(2,INPUT);//Declaramos el pin 2 como la entrada del pulsador
  randomSeed(analogRead(0));//Iniciamos el generador de números aleatorios a 0

  //apagamos todos los diodos leds
  digitalWrite(3,LOW);
  digitalWrite(4,LOW);
  digitalWrite(5,LOW);
  digitalWrite(6,LOW);
  digitalWrite(7,LOW);
  digitalWrite(8,LOW);
  digitalWrite(9,LOW);
}
```

Práctica 10: Dado electrónico

❑ Código

```
void loop () {  
  
  int numero;//definimos la variable donde se va a guardar el número aleatorio  
  
  while (!digitalRead (2));    //Espera a que aprieten pulsador  
  
  escribe_dado (0);           //Apaga todos los leds  
  
  while (digitalRead (2));    //Espera a que suelten pulsador  
  
  numero = random(1, 7);      //Genera un numero al azar entre 1 y 6  
  
  escribe_dado (numero);      //Mostrar el numero en el dado  
  
}
```

Práctica 10: Dado electrónico

□ Código

```
void escribe_dado (int num) {  
    //Escribe numero en el dado  
    switch (num) {  
  
        case 1: // el numero 1  
            digitalWrite (3, LOW);  
            digitalWrite (4, LOW);  
            digitalWrite (5, LOW);  
            digitalWrite (6, HIGH);  
            digitalWrite (7, LOW);  
            digitalWrite (8, LOW);  
            digitalWrite (9, LOW);  
            break;  
  
        case 2: // el numero 2  
            digitalWrite (3, LOW);  
            digitalWrite (4, HIGH);  
            digitalWrite (5, LOW);  
            digitalWrite (6, LOW);  
            digitalWrite (7, LOW);  
            digitalWrite (8, HIGH);  
            digitalWrite (9, LOW);  
            break;  
    }  
}
```

Práctica 10: Dado electrónico

❑ Código

```
case 3: //el numero 3
    digitalWrite (3, HIGH);
    digitalWrite (4, LOW);
    digitalWrite (5, LOW);
    digitalWrite (6, HIGH);
    digitalWrite (7, LOW);
    digitalWrite (8, LOW);
    digitalWrite (9, HIGH);
    break;
```

```
case 4: // el numero 4
    digitalWrite (3, HIGH);
    digitalWrite (4, LOW);
    digitalWrite (5, HIGH);
    digitalWrite (6, LOW);
    digitalWrite (7, HIGH);
    digitalWrite (8, LOW);
    digitalWrite (9, HIGH);
    break;
```

Práctica 10: Dado electrónico

❑ Código

```
case 5: //el numero 5
    digitalWrite (3, HIGH);
    digitalWrite (4, LOW);
    digitalWrite (5, HIGH);
    digitalWrite (6, HIGH);
    digitalWrite (7, HIGH);
    digitalWrite (8, LOW);
    digitalWrite (9, HIGH);
    break;
```

```
case 6: //el numero 6
    digitalWrite (3, HIGH);
    digitalWrite (4, HIGH);
    digitalWrite (5, HIGH);
    digitalWrite (6, LOW);
    digitalWrite (7, HIGH);
    digitalWrite (8, HIGH);
    digitalWrite (9, HIGH);
    break;
```


Práctica 10: Dado electrónico

❑ Código

```
default:
case 0: //Apagar todos los leds
    digitalWrite (3, LOW);
    digitalWrite (4, LOW);
    digitalWrite (5, LOW);
    digitalWrite (6, LOW);
    digitalWrite (7, LOW);
    digitalWrite (8, LOW);
    digitalWrite (9, LOW);
    break;
    }
}
```

Práctica 11: Programación de servo

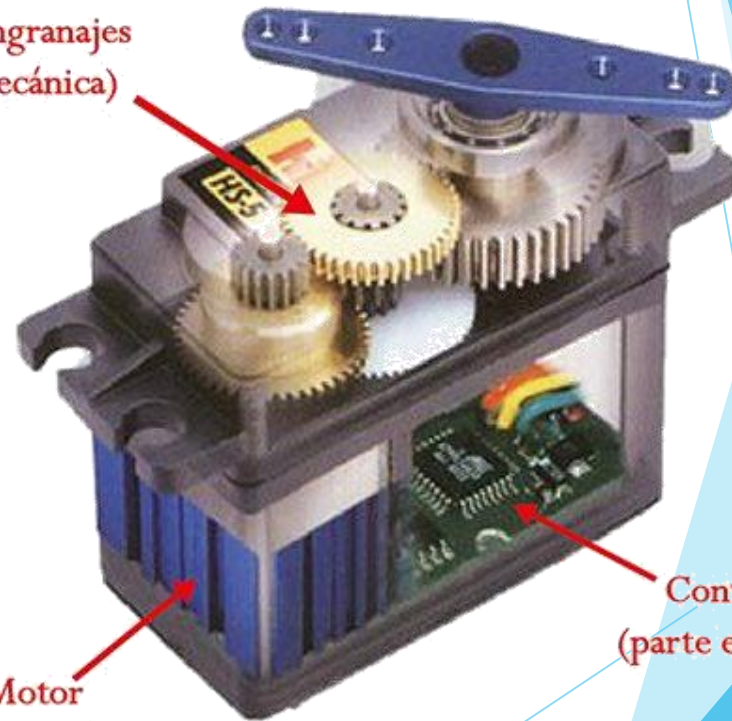
- ❑ Se trata de un motor de cc capaz de posicionarse en un ángulo determinado dentro de un rango que suele ser entre 0 °y 180°



Caja de engranajes
(parte mecánica)

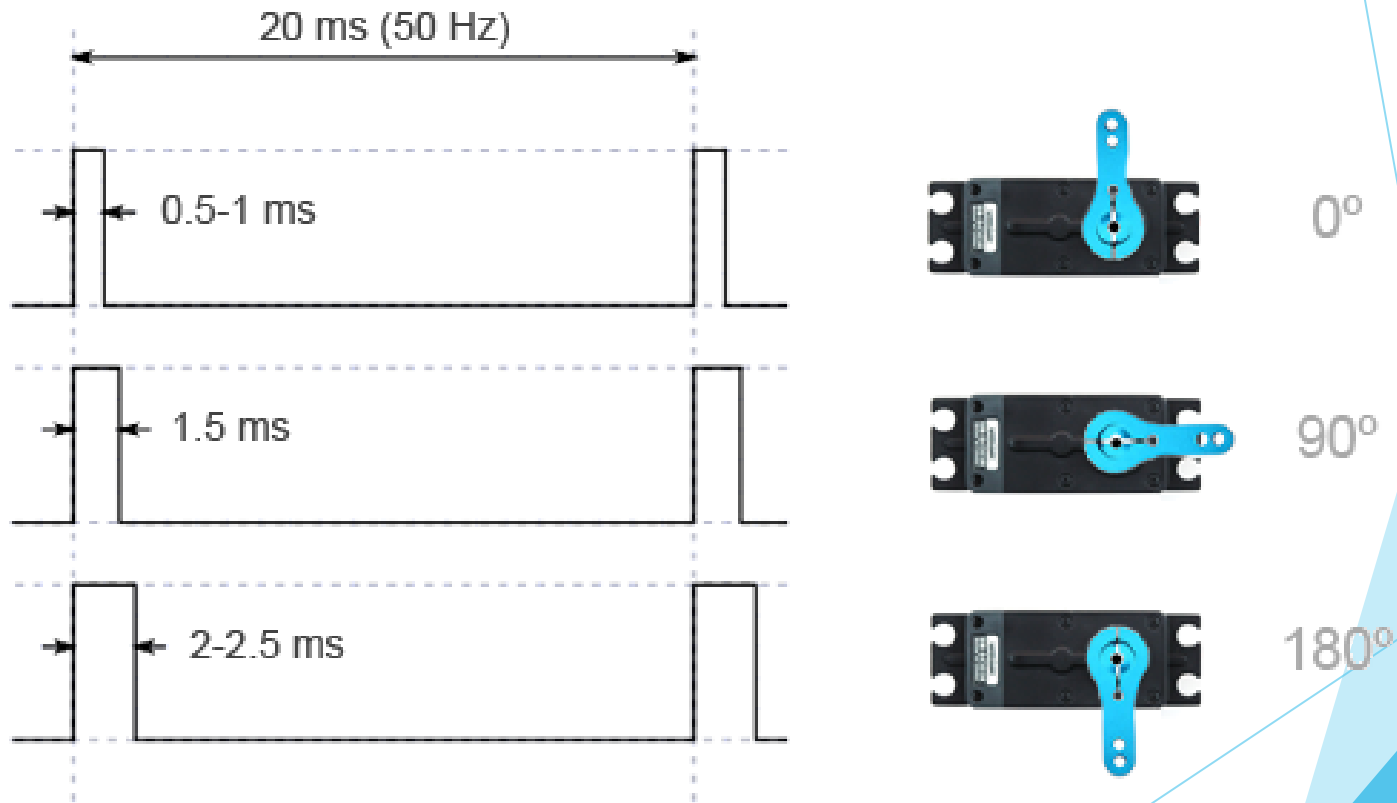
Motor
(parte eléctrica)

Controlador
(parte electrónica)



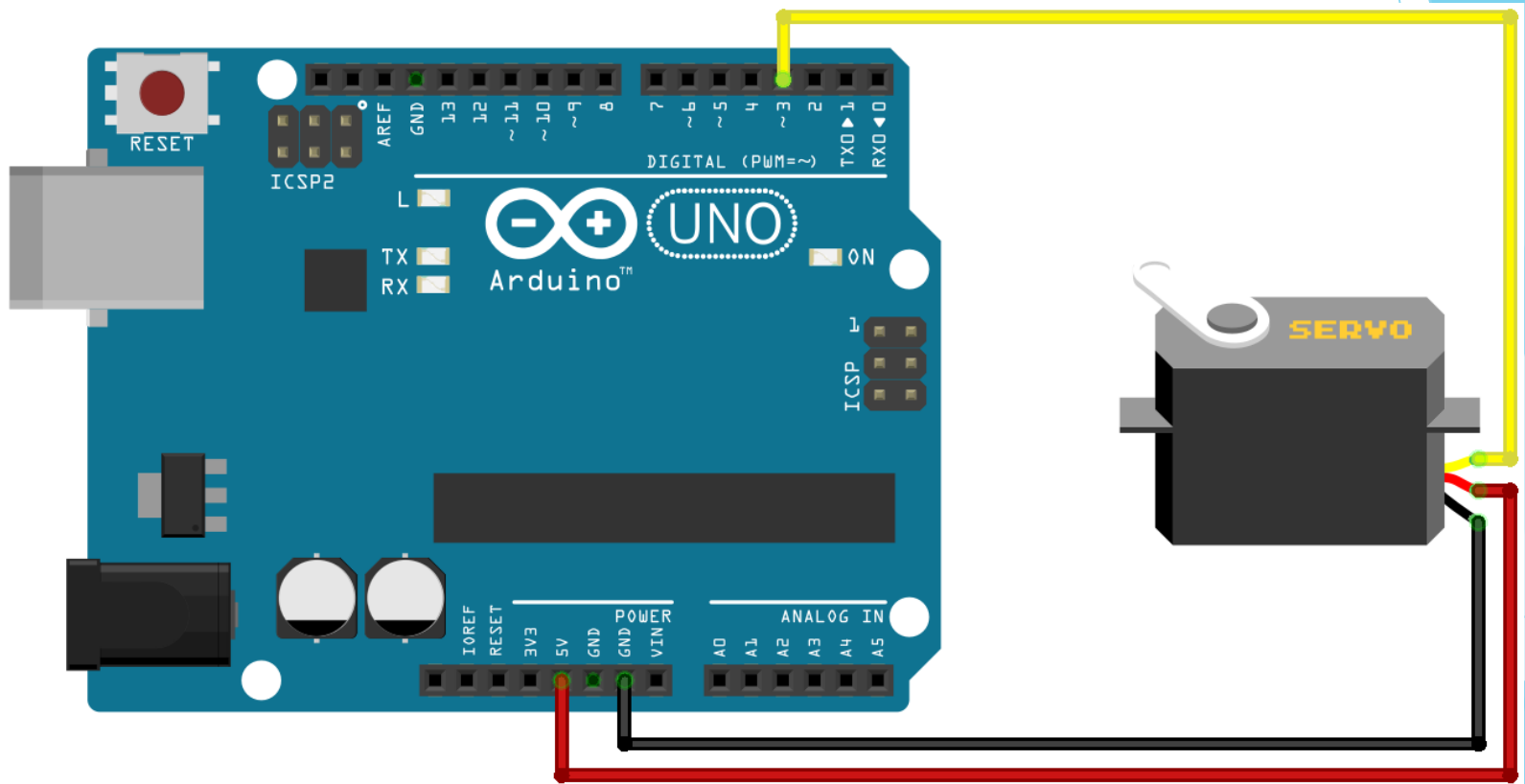
Práctica 11: Programación de servo

- ❑ Para comunicar el ángulo deseado utilizamos una señal pulsada con periodo de 20ms. El ancho del pulso determina la posición del servo.



Práctica 11: Programación de servo

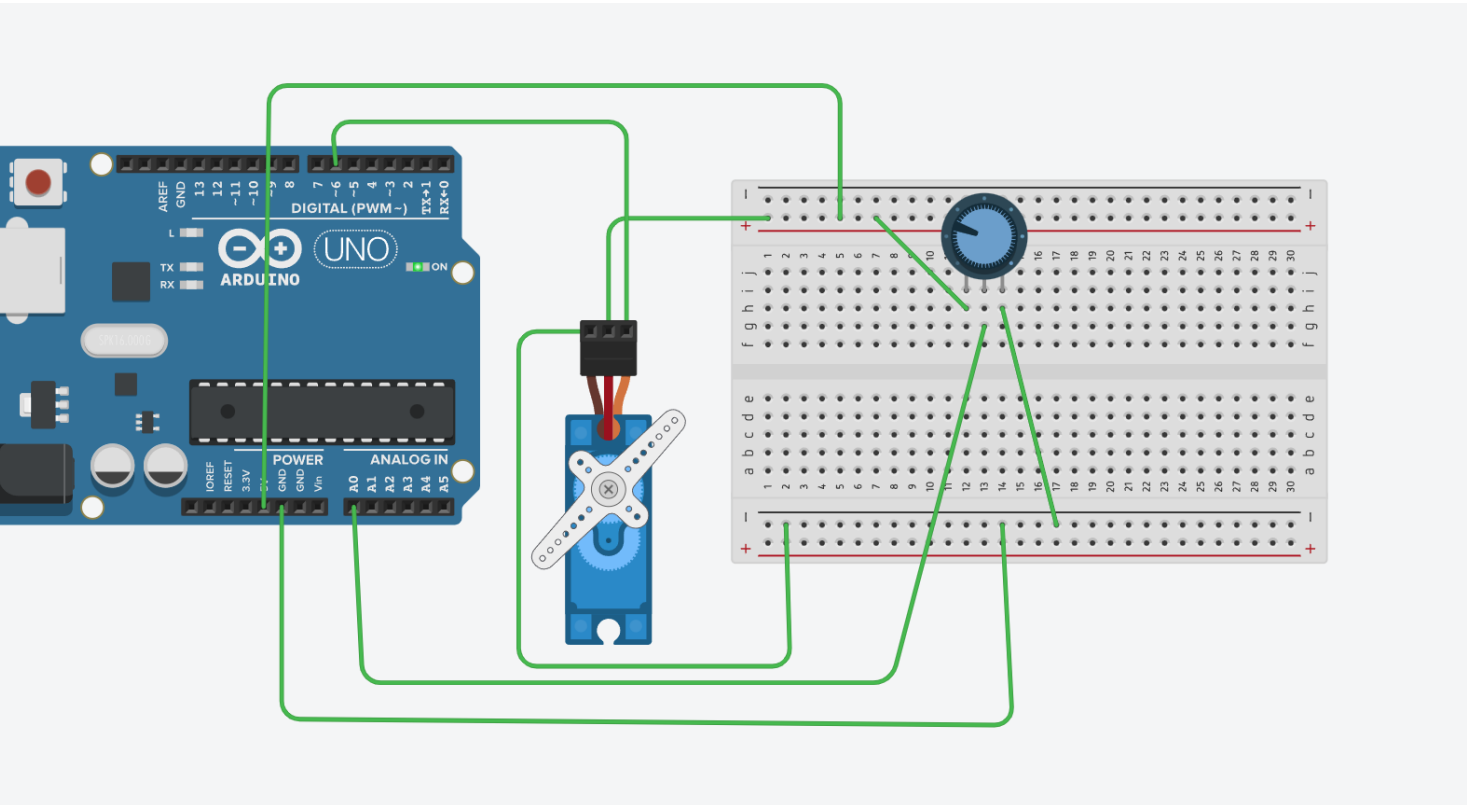
- Esquema de montaje



fritzing

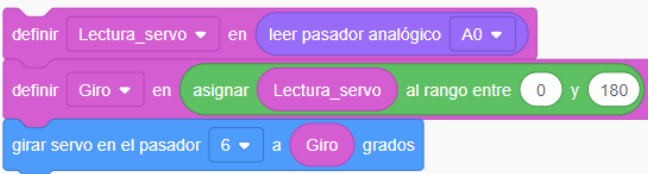
Práctica 11: Programación de servo

- Esquema de montaje: Control de servo mediante potenciómetro



Práctica 11: Programación de servo

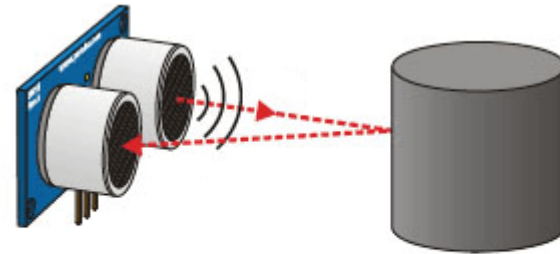
- ❑ Ejemplo de código: Control de servo mediante potenciómetro.



```
1 // C++ code
2 //
3 #include <Servo.h>
4
5 int Giro = 0;
6
7 int Lectura_servo = 0;
8
9 Servo servo_6;
10
11 void setup()
12 {
13     pinMode(A0, INPUT);
14     servo_6.attach(6, 500, 2500);
15 }
16
17
18 void loop()
19 {
20     Lectura_servo = analogRead(A0);
21     Giro = map(Lectura_servo, 0, 1023, 0, 180);
22     servo_6.write(Giro);
23     delay(10); // Delay a little bit to improve simulat
24 }
```

Práctica 12: Sensor de ultrasonidos

- Permite la medida de distancia mediante el envío y recepción de ondas sonoras.



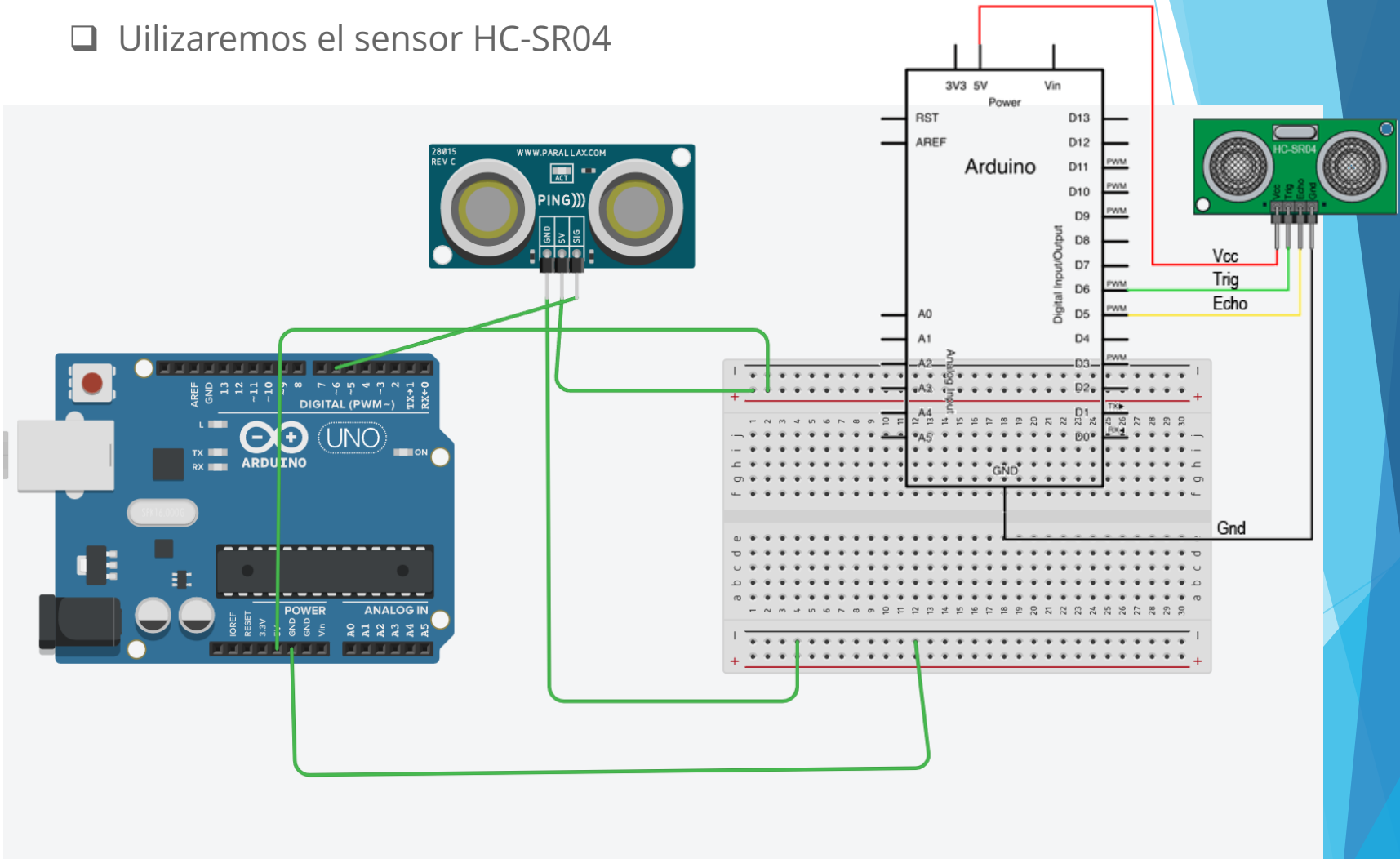
$$\begin{aligned} \text{Tiempo} &= 2 \cdot (\text{Distancia} / \text{Velocidad}) \\ \text{Distancia} &= \text{Tiempo} \cdot \text{Velocidad} / 2 \end{aligned}$$

$$343 \frac{m}{s} \cdot 100 \frac{cm}{m} \cdot \frac{1}{1000000} \frac{s}{\mu s} = \frac{1}{29.2} \frac{cm}{\mu s}$$

$$\text{Distancia}(cm) = \frac{\text{Tiempo}(\mu s)}{29.2 \cdot 2}$$

Práctica 12: Sensor de ultrasonidos

- Utilizaremos el sensor HC-SR04



Práctica 12: Sensor de ultrasonidos

- ❑ Ejemplo de código: Medida de distancia de un objeto.

definir Distancia en leer el sensor de distancia ultrasónico en el pasador del desencadenador 6 pasador de eco 6 en las unidades cm

imprimir en monitor en serie Distancia, nueva línea con

esperar 1 segundos

```
// C++ code
//
int Distancia = 0;

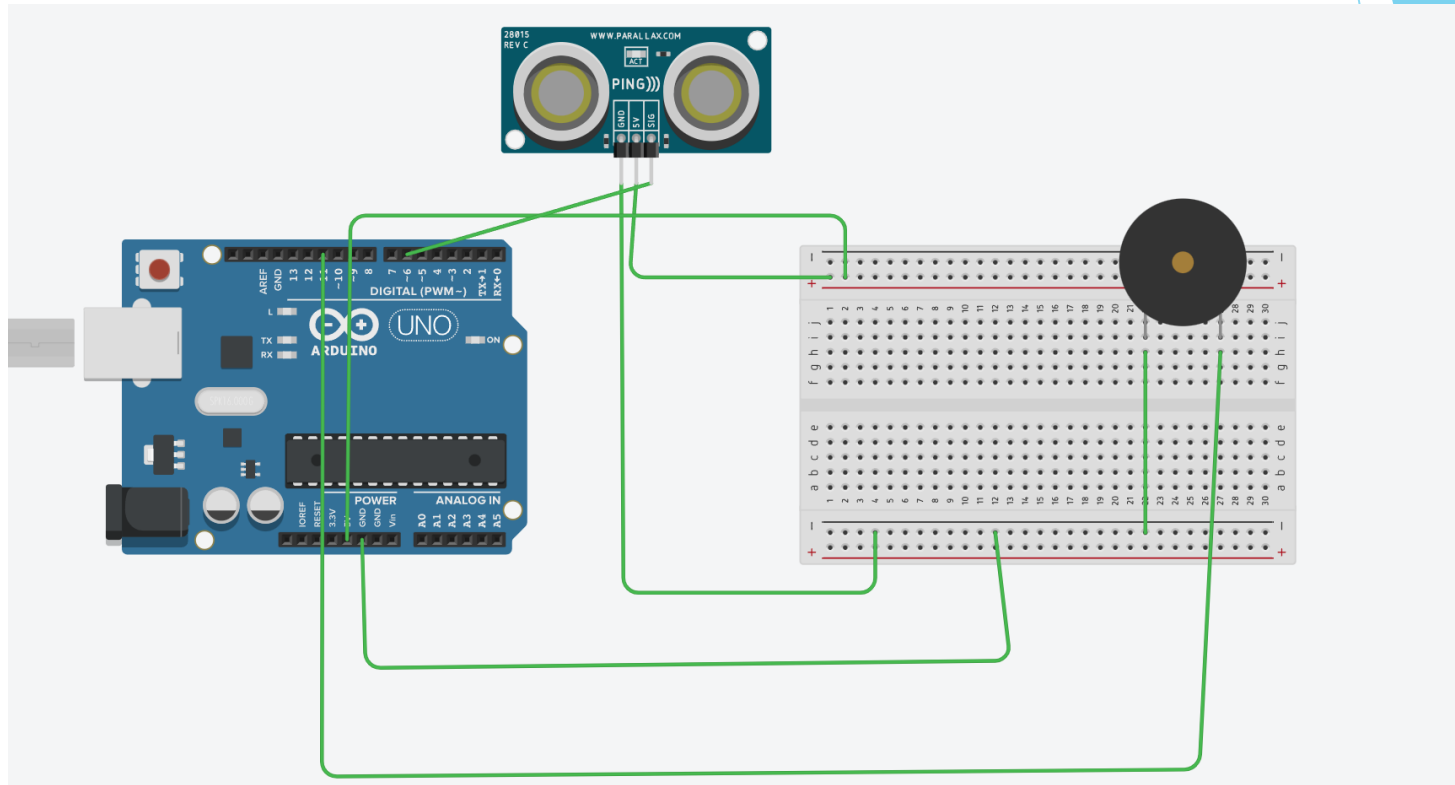
long readUltrasonicDistance(int triggerPin, int echoPin)
{
  pinMode(triggerPin, OUTPUT); // Clear the trigger
  digitalWrite(triggerPin, LOW);
  delayMicroseconds(2);
  // Sets the trigger pin to HIGH state for 10 microseconds
  digitalWrite(triggerPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(triggerPin, LOW);
  pinMode(echoPin, INPUT);
  // Reads the echo pin, and returns the sound wave travel time in microseconds
  return pulseIn(echoPin, HIGH);
}

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

void loop()
{
  Distancia = 0.01723 * readUltrasonicDistance(6, 6);
  Serial.println(Distancia);
  delay(1000); // Wait for 1000 millisecond(s)
}
```

Práctica 12: Sensor de ultrasonidos

- ❑ Ejemplo de montaje: Medida de distancia de un objeto y alarma al acercarse..



Práctica 12: Sensor de ultrasonidos

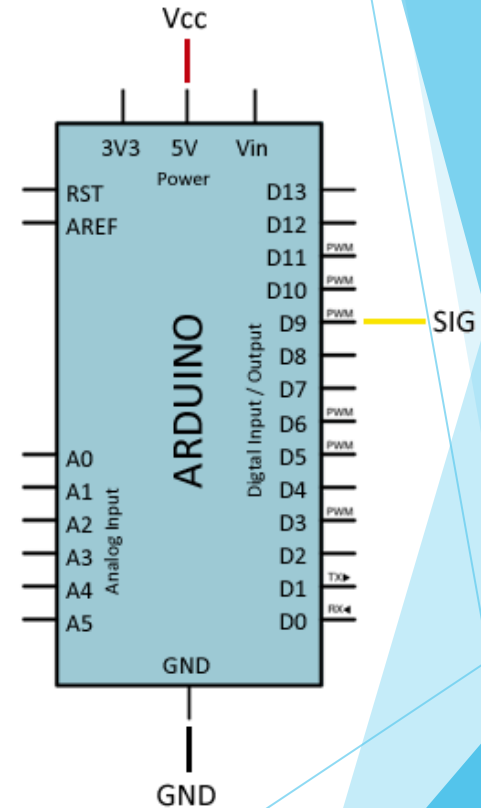
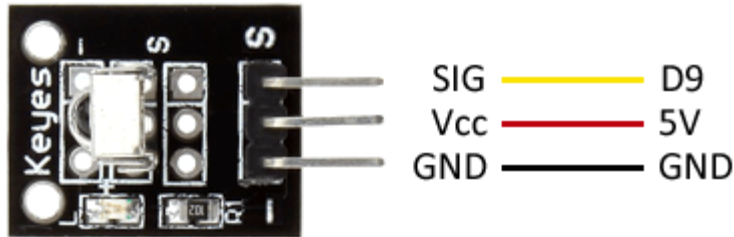
- ❑ Ejemplo de montaje: Medida de distancia de un objeto y alarma al acercarse..

```
definir Distancia en leer el sensor de distancia ultrasónico en el pasador del desencadenador 6 pasador de eco 6 en las unidades cm
imprimir en monitor en serie Distancia del objeto: , nueva línea con
imprimir en monitor en serie Distancia , nueva línea con
esperar 1 segundos
si Distancia < 25 entonces
  definir pasador 11 en ALTA
si no
  definir pasador 11 en BAJA
```

The image shows a Scratch script for an ultrasonic sensor. It starts with a 'definir' block that sets a variable named 'Distancia' to be read from an ultrasonic sensor on pin 6, with an echo pin of 6 and units in centimeters. This is followed by two 'imprimir en monitor en serie' blocks: the first prints 'Distancia del objeto:' followed by a comma and a new line, and the second prints the value of 'Distancia' followed by a comma and a new line. Then, there is an 'esperar' block for 1 second. A conditional 'si' block checks if 'Distancia' is less than 25. If true, it sets pin 11 to 'ALTA' (HIGH); if false, it sets pin 11 to 'BAJA' (LOW).

Práctica 13 : Sensor de infrarojos

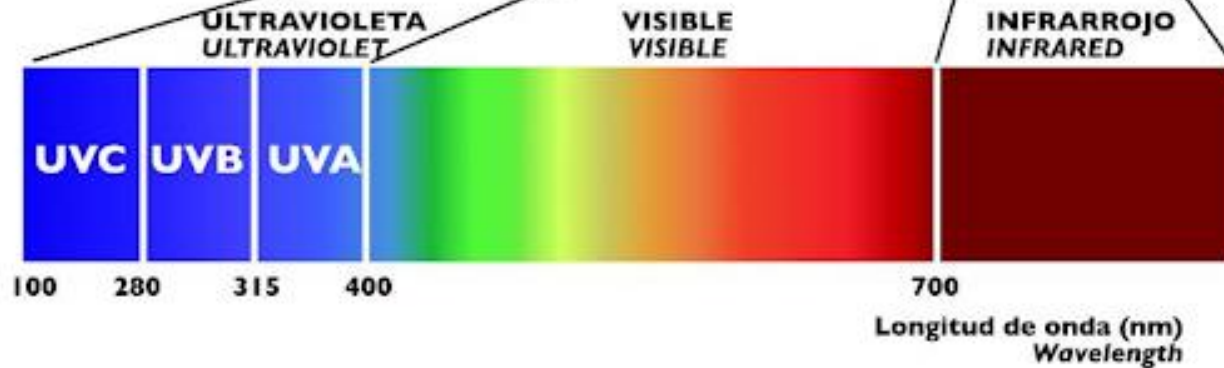
- Este sensor es capaz de recibir información en forma de radiación infrarroja. Utilizaremos para esta práctica el sensor HX1838 o similar



Práctica 13: Sensor de infrarojos

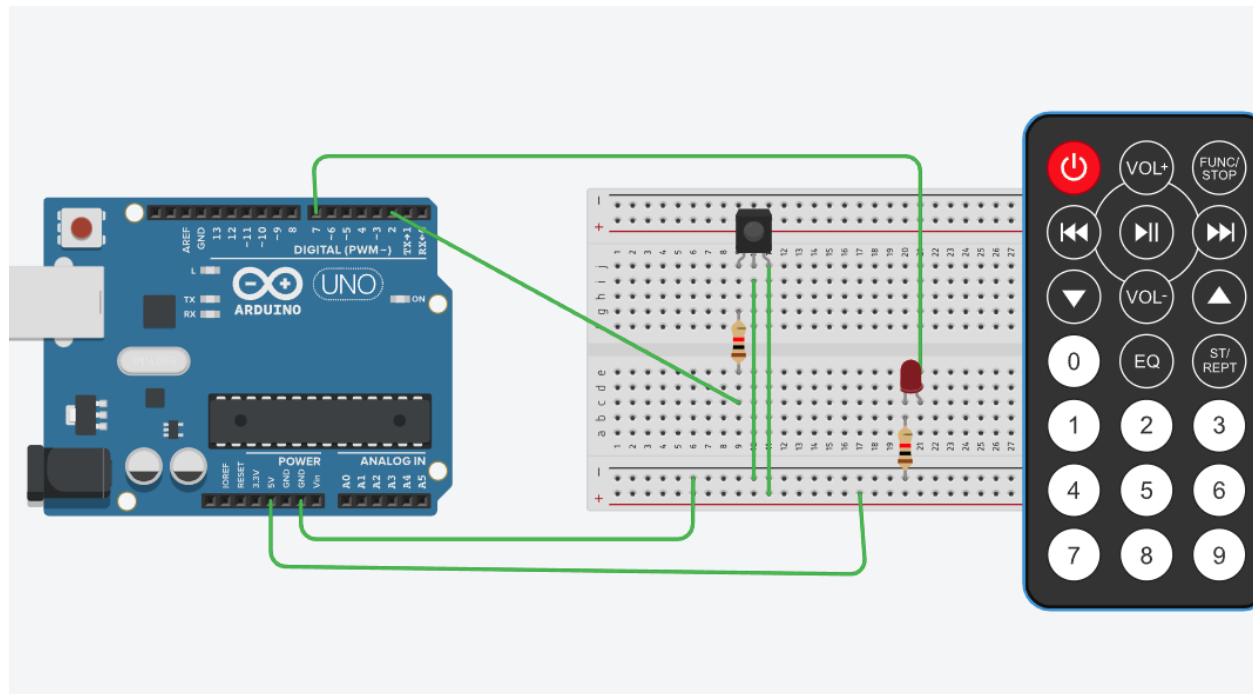
- ❑ La radiación infraroja tiene una longitud de onda superior a los 700nm y no es visible al ojo humano, pero puede ser percibido por cámaras de vídeo o fotográficas.

Espectro solar *Solar Spectrum*



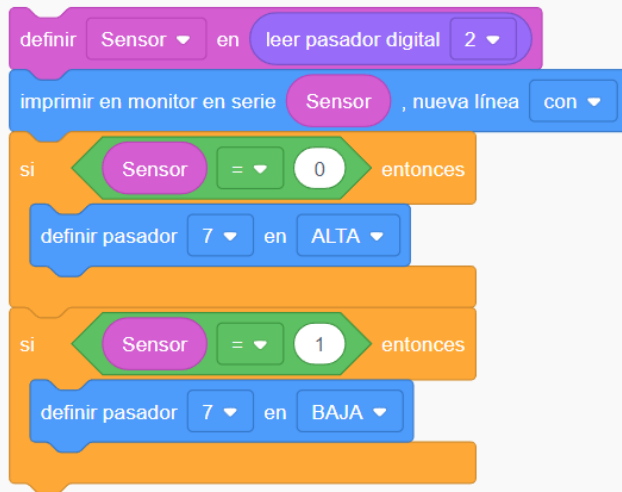
Práctica 13: Sensor de infrarojos

❑ Esquema de conexión



Práctica 13 : Sensor de infrarojos

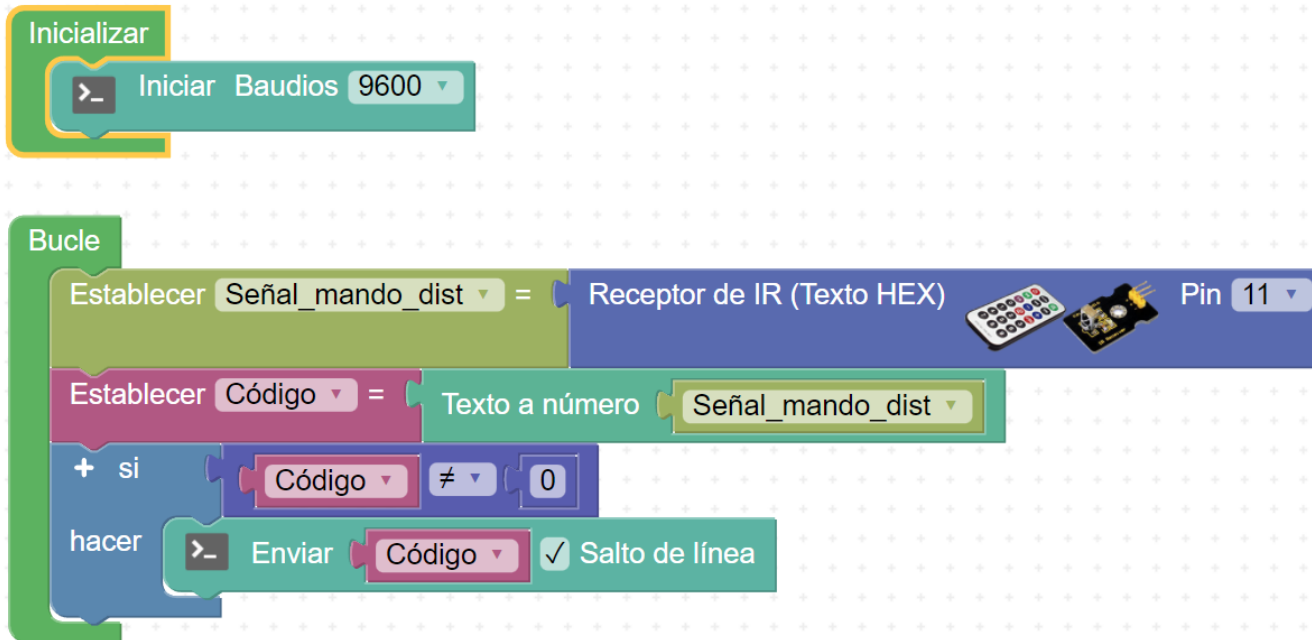
- ❑ Programa que enciende un led cuando el sensor infrarojo recibe alguna señal.



```
1 // C++ code
2 //
3 int Sensor = 0;
4
5 void setup()
6 {
7   pinMode(2, INPUT);
8   Serial.begin(9600);
9
10  pinMode(7, OUTPUT);
11 }
12
13 void loop()
14 {
15   Sensor = digitalRead(2);
16   Serial.println(Sensor);
17   if (Sensor == 0) {
18     digitalWrite(7, HIGH);
19   }
20   if (Sensor == 1) {
21     digitalWrite(7, LOW);
22   }
23   delay(10); // Delay a litt
24 }
```

Práctica 13 : Sensor de infrarojos con Arduino Blocks

- ❑ Programa para identificar el código de cada botón del mando a distancia



Práctica 13 : Sensor de infrarojos con Arduino Blocks

- ❑ Programa encender y apagar leds con los botones 1 al 4 del mando a distancia.

The image shows a screenshot of the Arduino Blocks programming environment. The code is organized into a 'Bucle' (Loop) block. It starts with an 'Establecer Señal_mando_dist = Receptor de IR (Texto HEX) Pin 11' block, which is connected to an IR receiver module. This is followed by an 'Establecer Código = Texto a número Señal_mando_dist' block. A 'si' (if) block checks if 'Código ≠ 0'. Inside this 'si' block is a 'hacer' (do) loop. The 'hacer' loop contains four 'si' blocks, each corresponding to a button code (805, 802, 803, 804). For each code, there is a 'hacer' block that performs digital writes to pins 5, 6, and 7. Codes 805, 802, and 803 result in pins 5, 6, and 7 being set to 'ON', respectively. Code 804 results in pins 5, 6, and 7 being set to 'OFF'.

```
graph TD
    subgraph Bucle
        A[Establecer Señal_mando_dist = Receptor de IR (Texto HEX) Pin 11]
        B[Establecer Código = Texto a número Señal_mando_dist]
        C[+ si Código ≠ 0]
        D[hacer]
        E[+ si Código = 805]
        F[hacer Escribir digital Pin 5 ON]
        G[+ si Código = 802]
        H[hacer Escribir digital Pin 6 ON]
        I[+ si Código = 803]
        J[hacer Escribir digital Pin 7 ON]
        K[+ si Código = 804]
        L[hacer Escribir digital Pin 5 OFF]
        M[hacer Escribir digital Pin 6 OFF]
        N[hacer Escribir digital Pin 7 OFF]
    end
    A --> B
    B --> C
    C --> D
    D --> E
    E --> F
    D --> G
    G --> H
    D --> I
    I --> J
    D --> K
    K --> L
    L --> M
    M --> N
```

Programación mediante bloques: Arduino Blocks

The screenshot shows the Arduino Blocks web interface. The top navigation bar includes the Arduino Blocks logo, search options, and user information. The main workspace is currently empty, with a project titled "Lectura LDR - Puerto Serie" selected. A left sidebar lists various block categories such as "Lógica", "Control", "Matemáticas", "Texto", "Variables", "Listas", "Funciones", "Tiempo", "Entrada/Salida", "Sensores", "Actuadores", "Motor", "Periféricos", and "Visualización". The right sidebar contains control buttons for "Subir", "Consola", and "COM8", along with a trash icon labeled "Papelera". Blue arrows point to these elements with labels: "Bloques de programación" points to the left sidebar, "Subida de archivo a placa" points to the "Subir" button, "Apertura de consola serie" points to the "Consola" button, "Selección de puerto" points to the "COM8" dropdown, and "Papelera" points to the trash icon.

Arduino Blocks

Buscar proyectos Proyectos

Recursos 🇪🇸 entregas.tecno@gmail.com Cerrar sesión

Bloques Información Archivos Lectura LDR - Puerto Serie

Subir Consola COM8

Lógica

Control

Matemáticas

Texto

Variables

Listas

Funciones

Tiempo

Entrada/Salida

Sensores

Actuadores

Motor

Periféricos

Visualización

Pantalla LCD (I2C)

Pantalla LCD (4-Bit)

Pantalla OLED

LedMatrix 8x8

NeoPixel

Bloques de programación

Subida de archivo a placa

Apertura de consola serie

Selección de puerto

Papelera

Práctica 14: Medida de humedad y temperatura con dht11

❑ Características básicas



DHT11

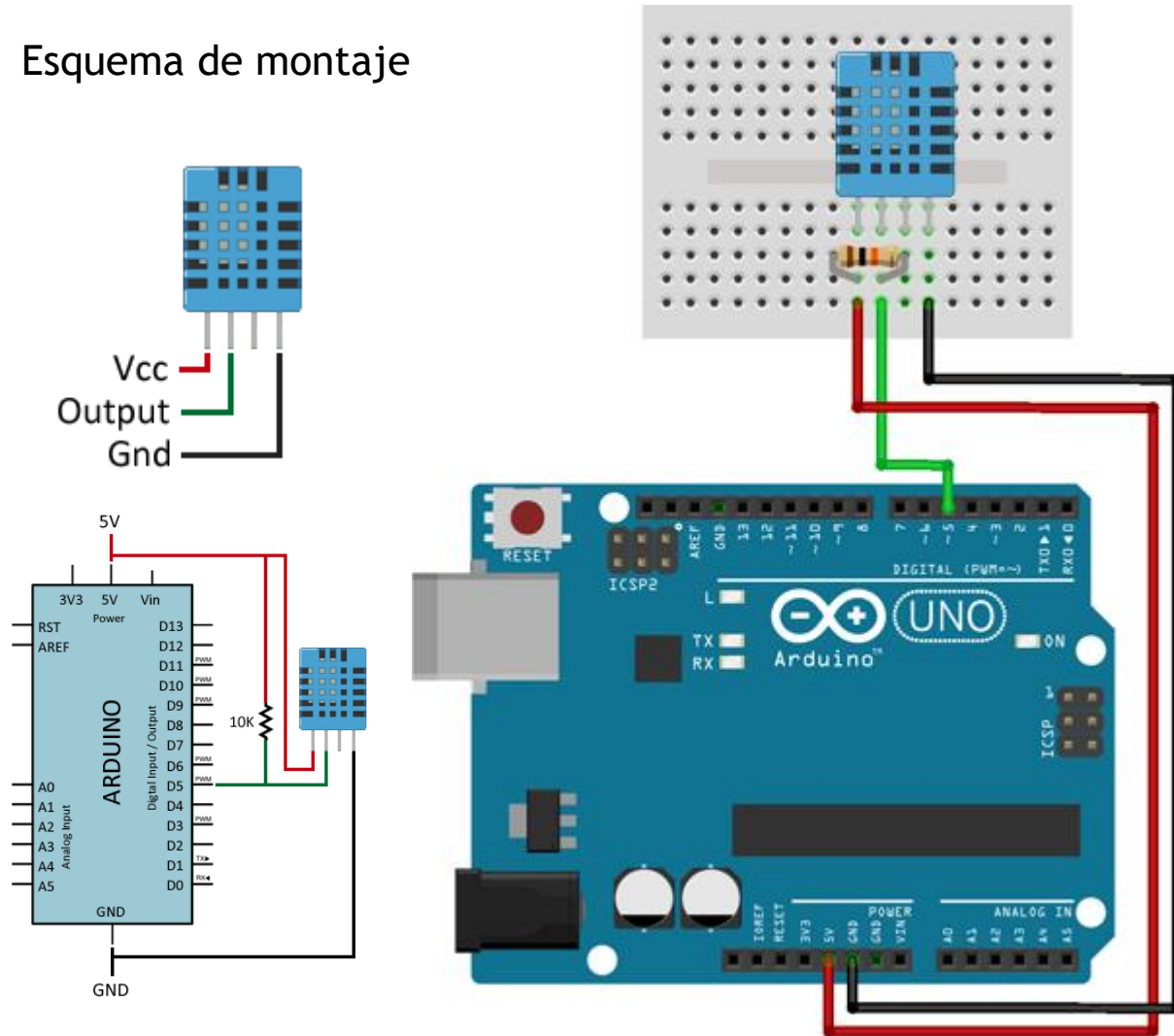


DHT22

Modelo	DHT11	DHT22
Rango de medición de humedad	20-90 % HR	0-100 % HR
Rango de medición de temperatura	0 hasta 50 °C	-40 hasta 80 °C
Precisión de temperatura	±2 °C	±0.5 °C
Precisión de humedad	±5 % HR	±2 % HR

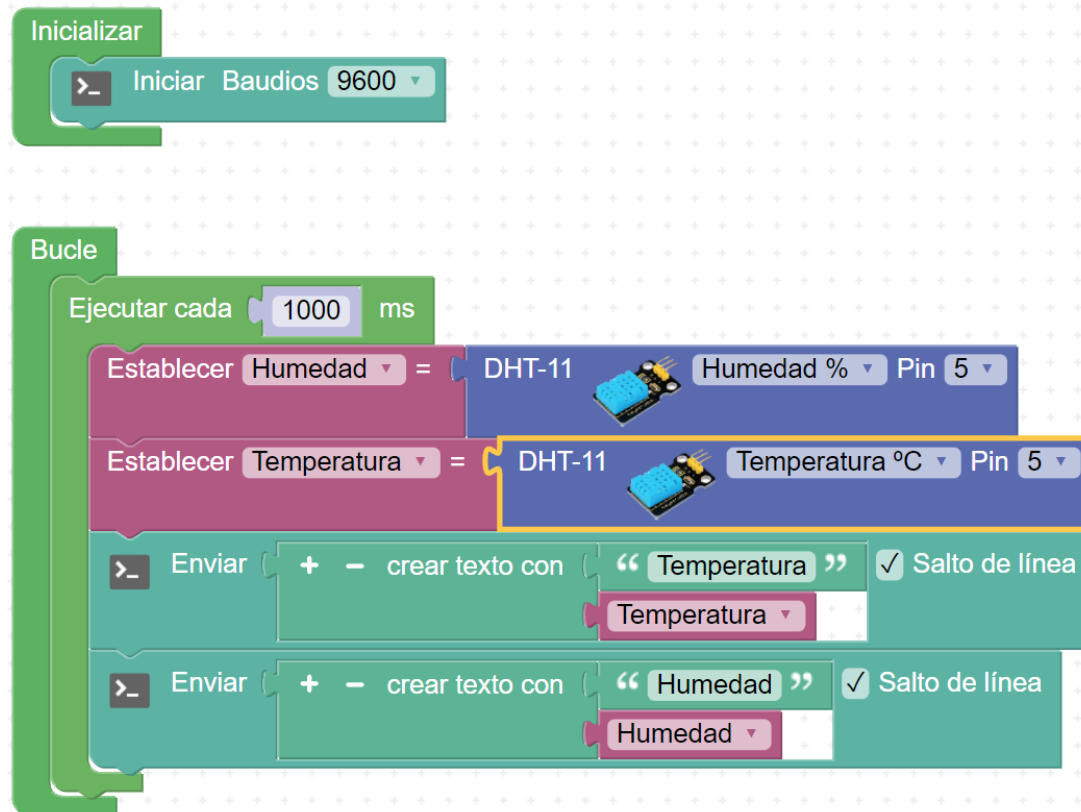
Práctica 14: Medida de humedad y temperatura con dht11

Esquema de montaje



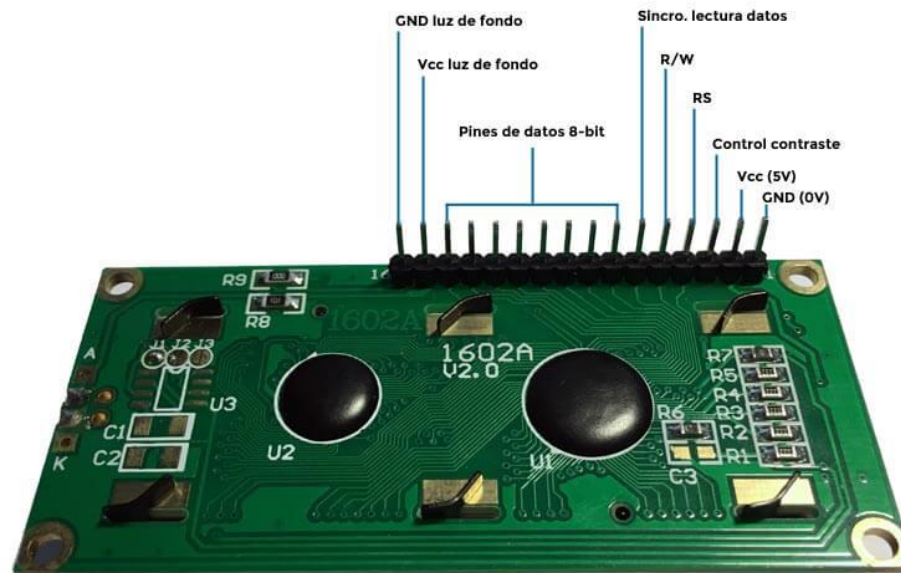
Práctica 14: Medida de humedad y temperatura con dht11

- ❑ Vamos a diseñar un programa que permita realizar lecturas de la temperatura y humedad ambiental cada segundo y envíe ambos datos al puerto serie para su visualización.



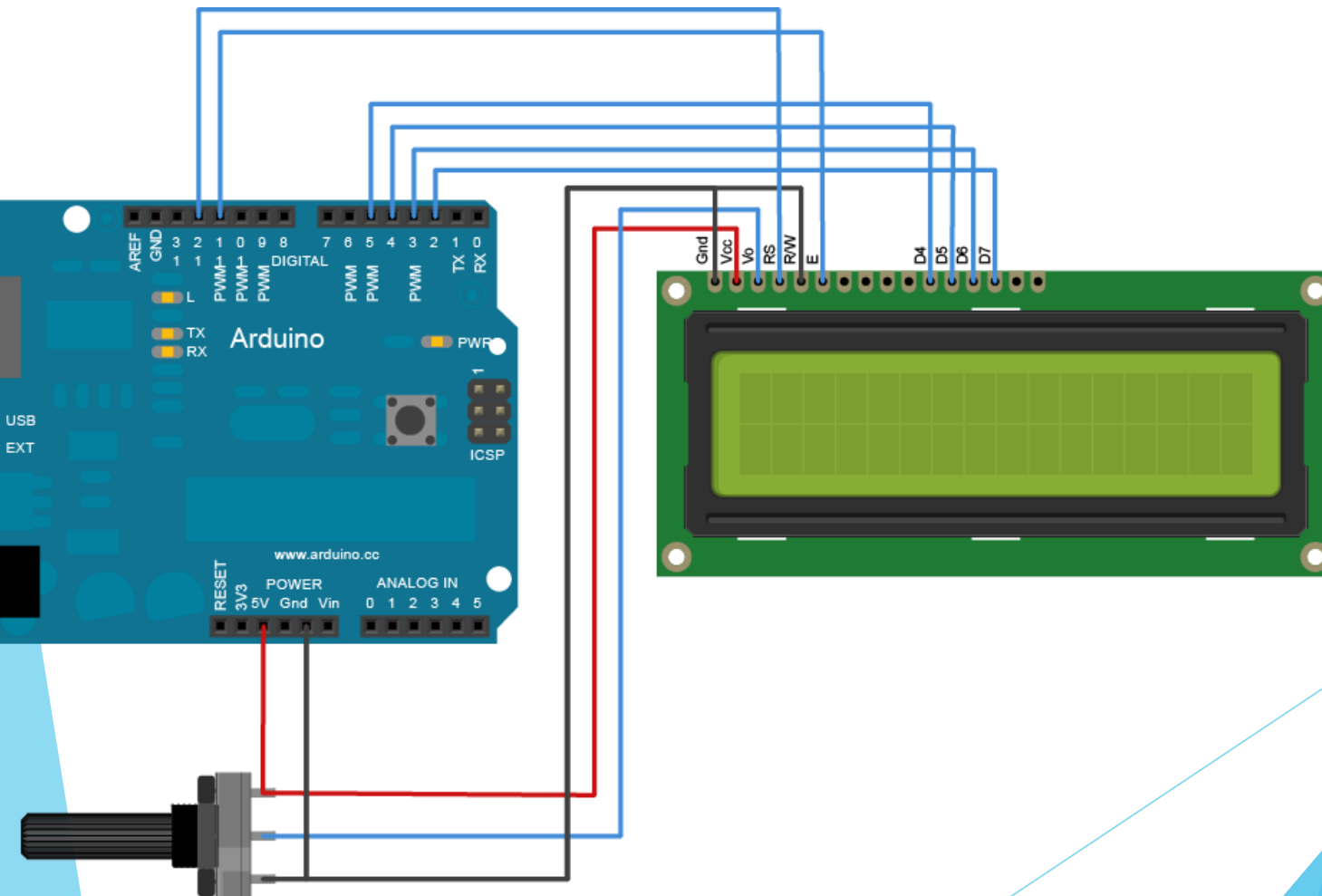
Práctica 15: Display LCD y sensor LDR

- ❑ Pantalla LCD 16 pines. Poseen 2 filas de 16 caracteres cada una



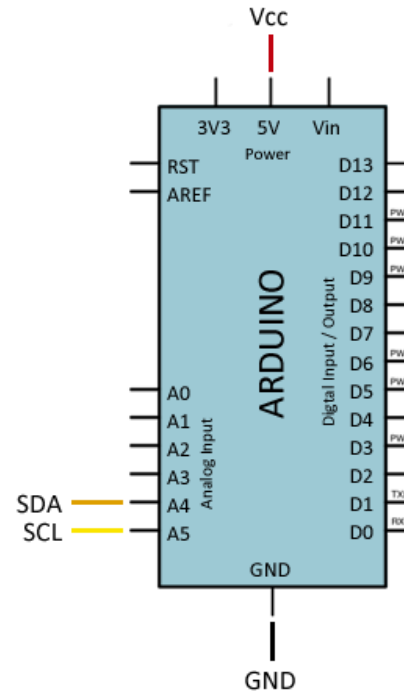
Práctica 15: Display LCD y sensor LDR

- ❑ Conexión pantalla LCD 16 pines.



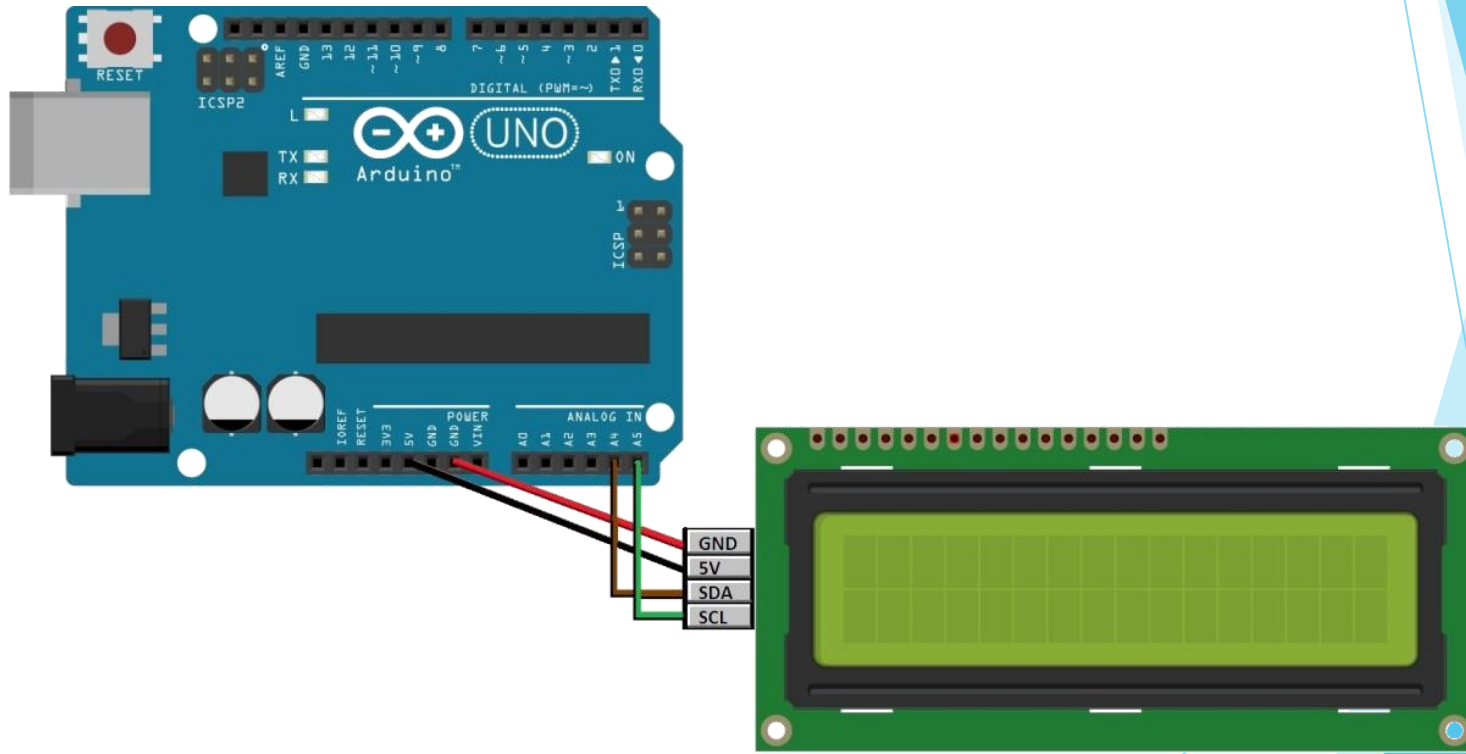
Práctica 15: Display LCD y sensor LDR

- Pantalla LCD con bus I2C. Poseen 2 filas de 16 caracteres cada una



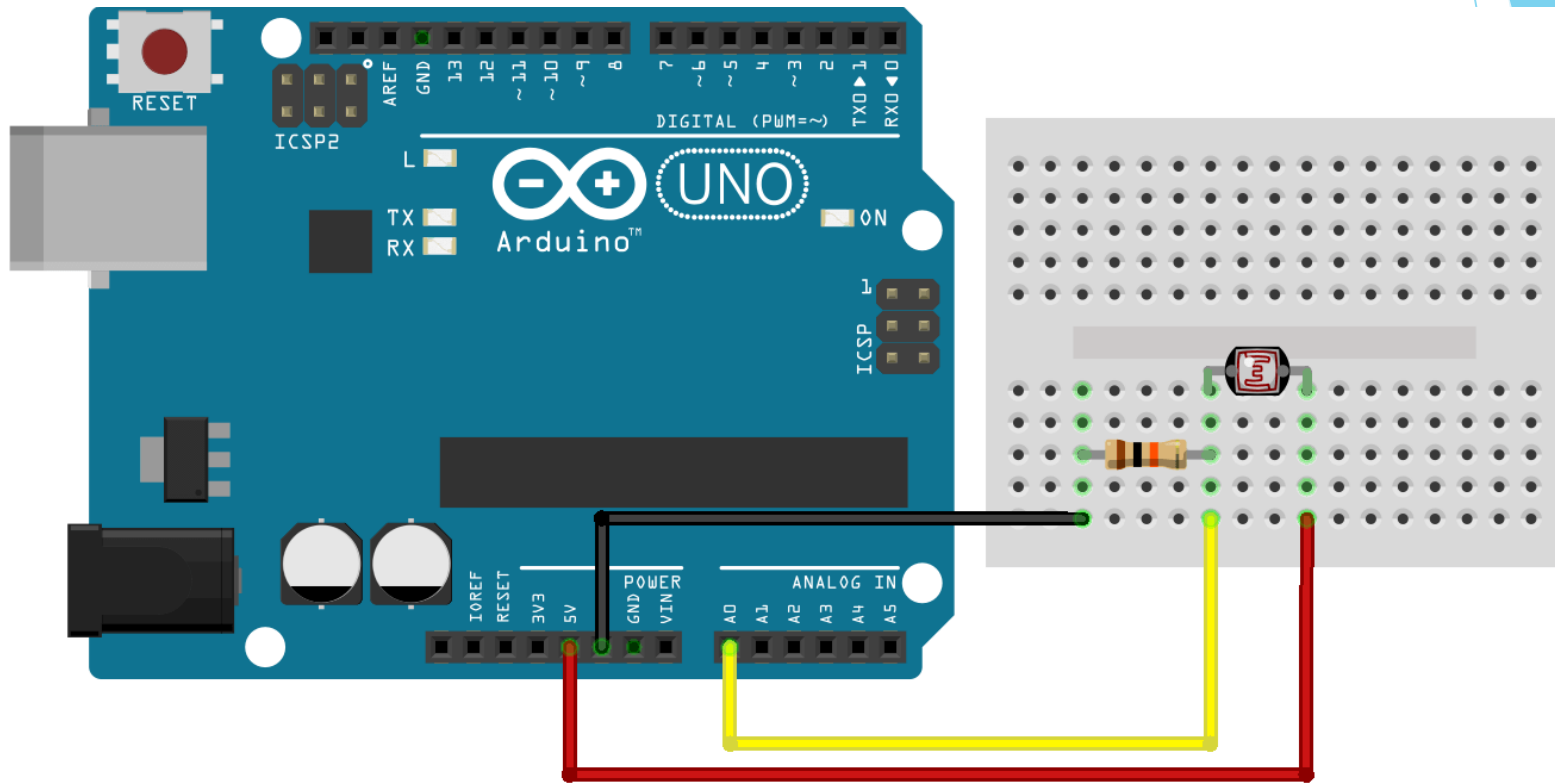
Práctica 15: Display LCD y sensor LDR

- ❑ Esquema de montaje pantalla LCD con bus I2C.



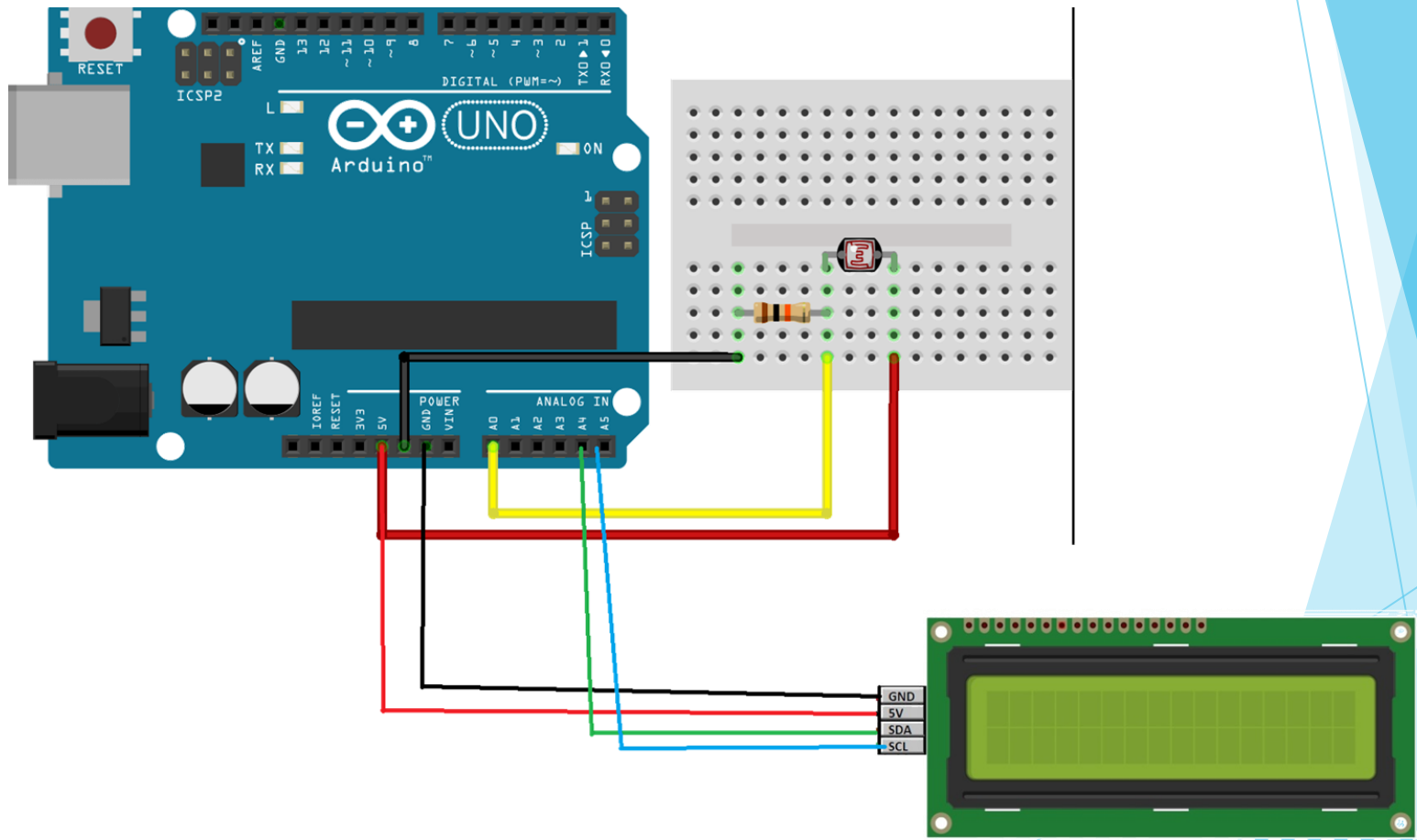
Práctica 15: Display LCD y sensor LDR

- ❑ Esquema de montaje de una resistencia LDR



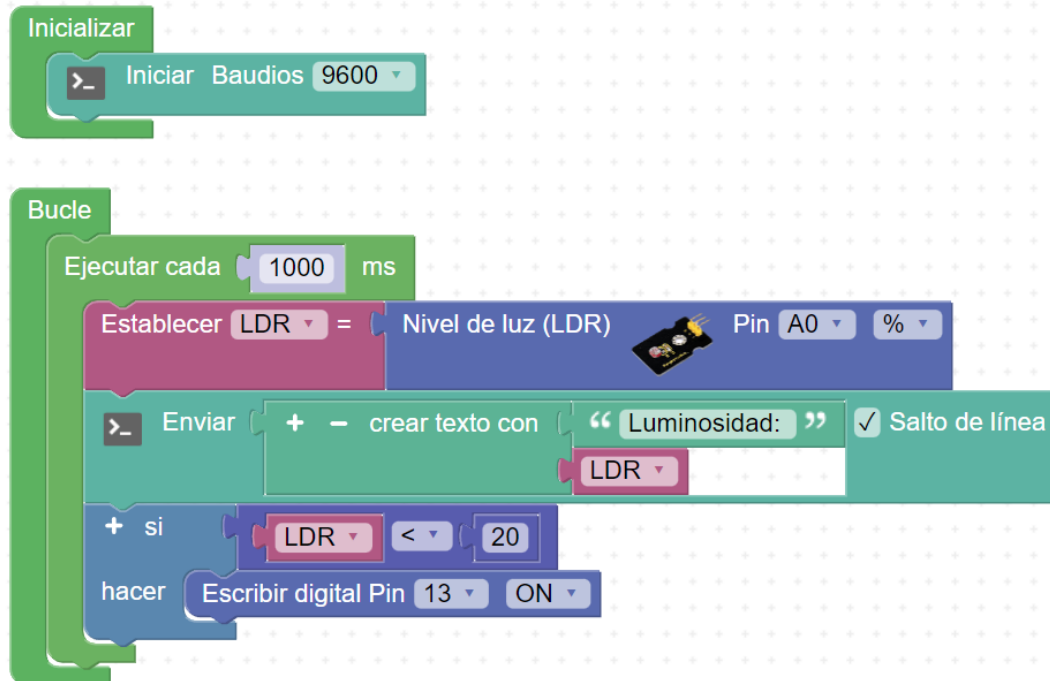
Práctica 15: Display LCD y sensor LDR

- ❑ Esquema de montaje para lectura de luminosidad en %



Práctica 15: Display LCD y sensor LDR

- ❑ Lectura de luminosidad en % y envío a puerto serie para su visualización.



Práctica 15: Display LCD y sensor LDR

- ❑ Lectura de luminosidad en % y visualización a través de pantalla LCD

The image shows a Scratch-style code editor with the following blocks:

- Inicializar**
 - Iniciar Baudios 9600
 - LCD # 1 Iniciar 2x16 I2C ADDR 0x27 *
 - LCD # 1 Limpiar
- Bucle**
 - Ejecutar cada 1000 ms
 - Establecer LDR = Nivel de luz (LDR) Pin A0 %
 - LCD # 1 Imprimir Columna 0 Fila 0 " Medidor de luminosidad "
 - LCD # 1 Imprimir Columna 0 Fila 1 + - crear texto con " Luminosidad " + " Luminosidad: " + LDR
 - Enviar + - crear texto con " Luminosidad: " Salto de línea + LDR
 - + si LDR < 20 hacer Escribir digital Pin 13 ON

Práctica 16: Control de motor paso a paso

- ❑ Se trata de un motor muy utilizado en robótica que convierte pulsos eléctricos en movimientos angulares.

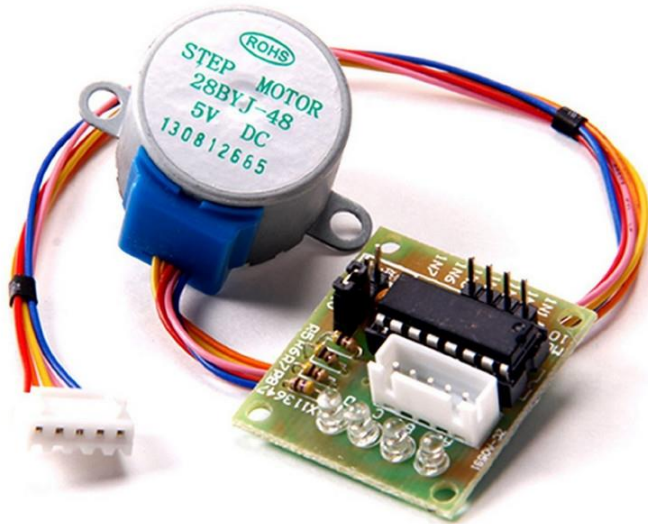


www.palolu.com



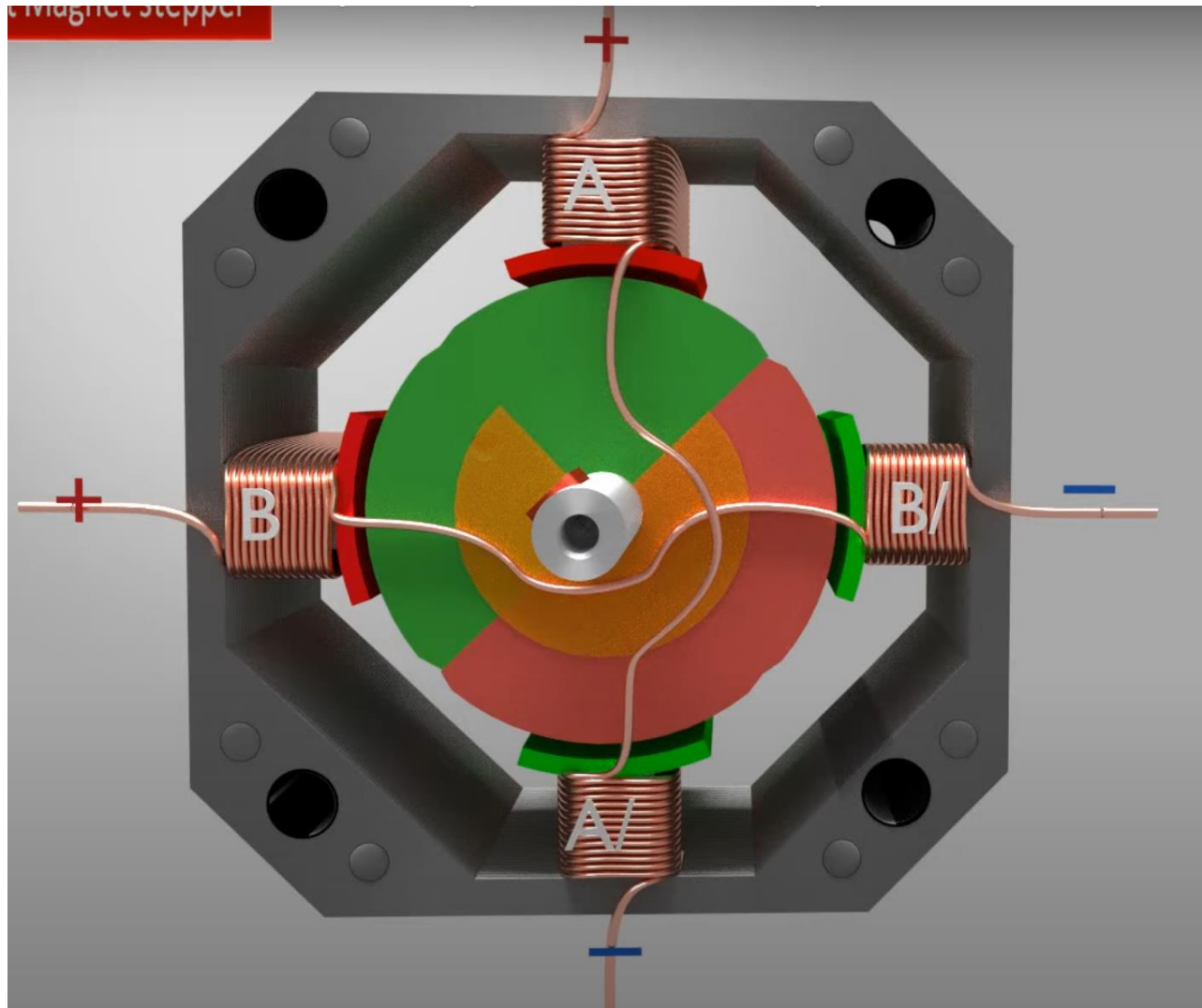
Práctica 16: Control de motor paso a paso.

□ En nuestro caso utilizaremos el motor 28BYJ-48

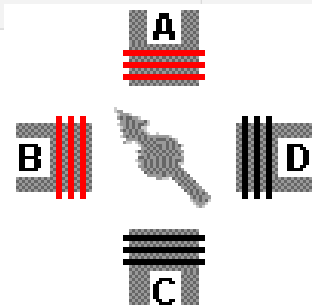


- Tensión nominal entre 5 y 12V
- 4 fases
- Par motor 0,34 kg*cm
- Consumo 55A
- Reductora 1/64
- 8 pasos por vuelta
- Precisión de 0.088°
- Precio aprox 3€

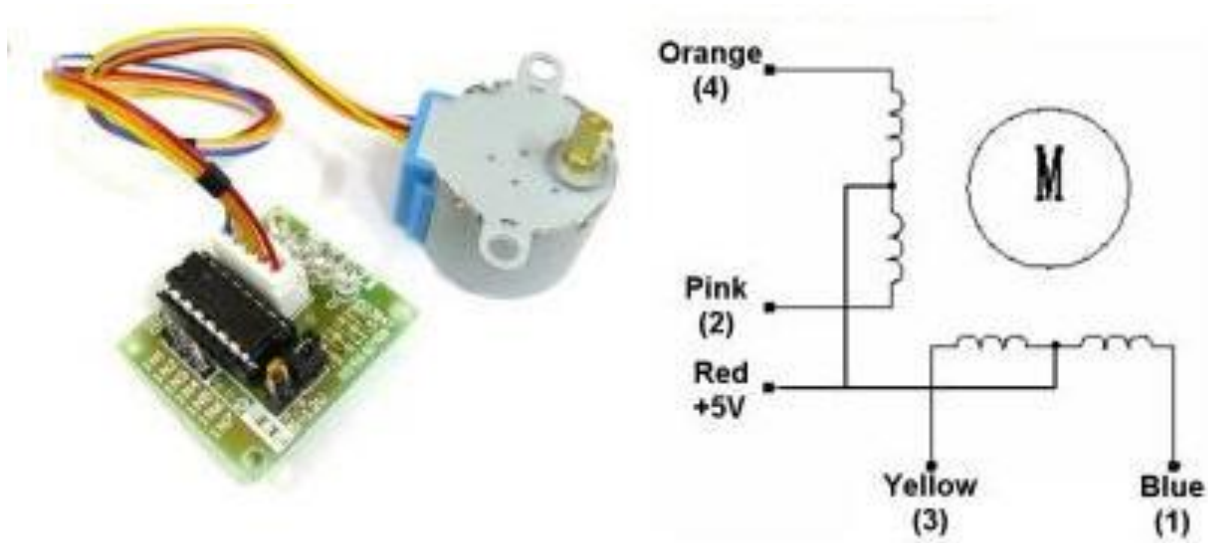
Práctica 16: Control de motor paso a paso. Funcionamiento



PASO	BOBINA A	BOBINA B	BOBINA C	BOBINA D	
1	ON	ON	OFF	OFF	
2	OFF	ON	ON	OFF	
3	OFF	OFF	ON	ON	
4	ON	OFF	OFF	ON	



Práctica 16: Control de motor paso a paso. Funcionamiento

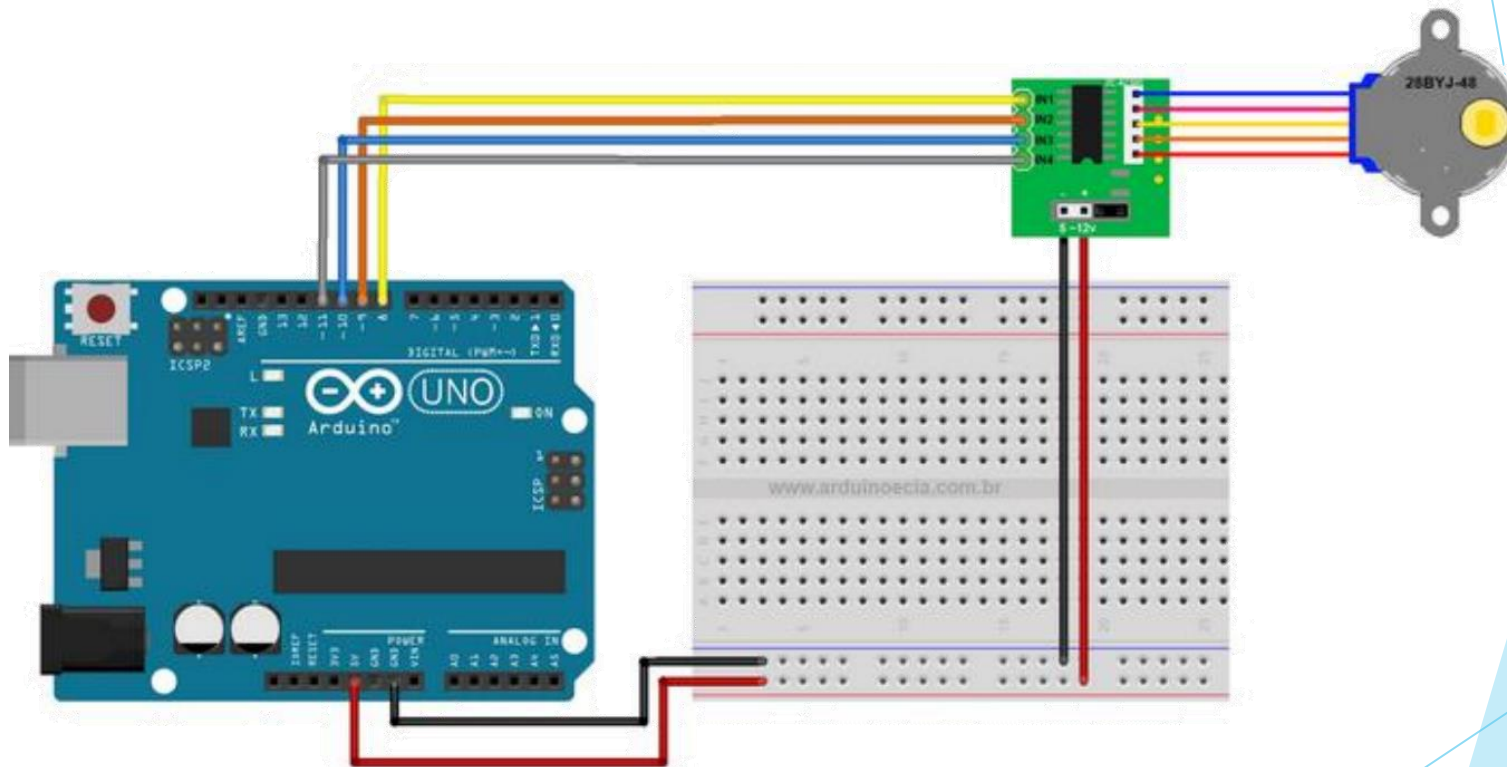


Half-Step Switching Sequence

Lead Wire Color	---> CW Direction (1-2 Phase)							
	1	2	3	4	5	6	7	8
4 Orange	-	-						-
3 Yellow		-	-	-				
2 Pink				-	-	-		
1 Blue						-	-	-

Práctica 16: Control de motor paso a paso

- ❑ Esquema de conexión



Práctica 16: Control de motor paso a paso

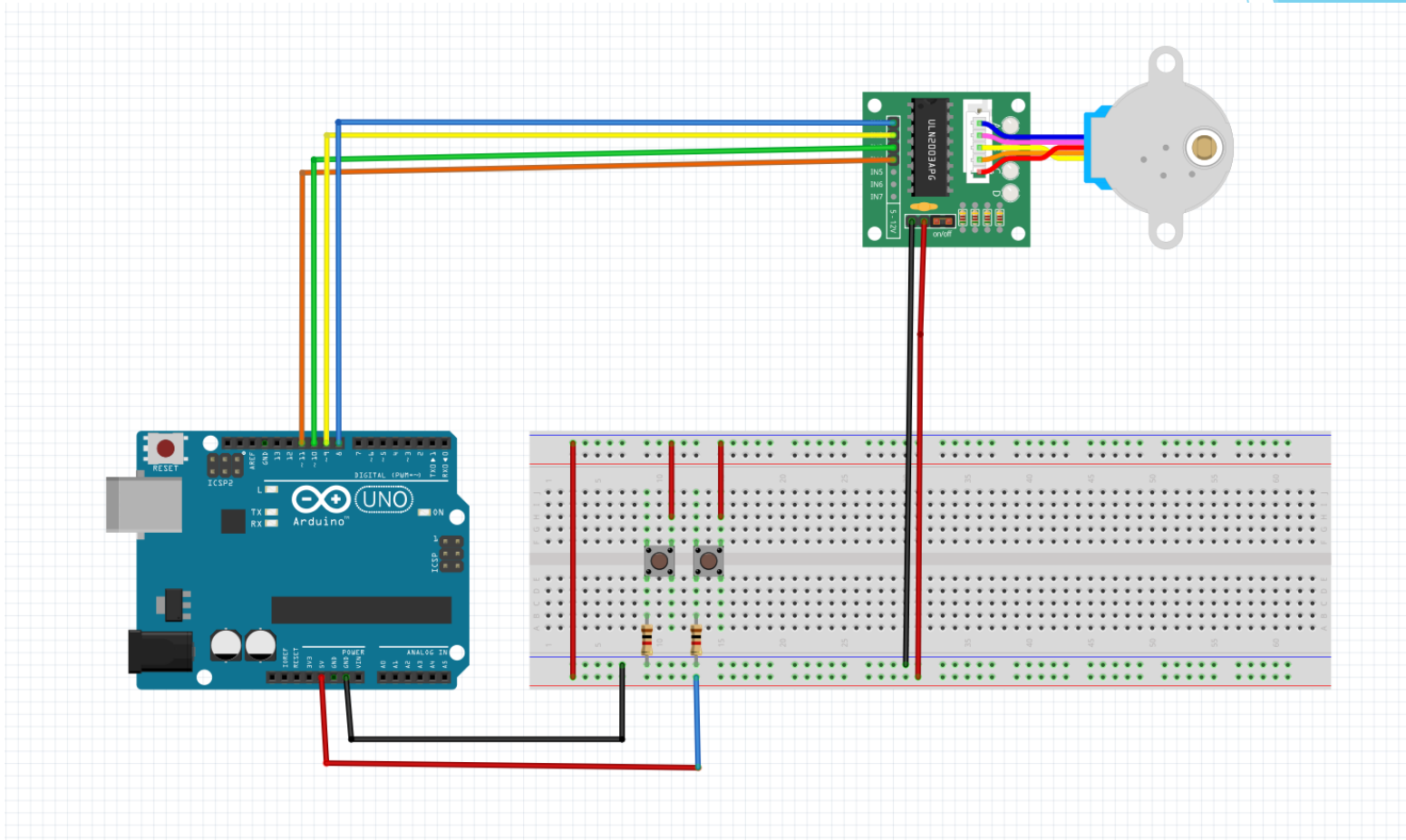
- ❑ Código de programa para girar en un sentido y luego en otro.

```
Inicializar
Paso a paso # 1 Pasos/vuelta 256 Pin-1 8 Pin-2 9 Pin-3 10 Pin-4 11
Paso a paso # 1 Velocidad (rpm) 30
```

```
Bucle
Paso a paso # 1 Pasos 2048
Esperar 1000 milisegundos
Paso a paso # 1 Pasos -2048
Esperar 1000 milisegundos
```

Práctica 16: Control de motor paso a paso

- Control del motor paso a paso mediante pulsadores



Práctica 16: Control de motor paso a paso

□ Control del motor paso a paso mediante pulsadores

Inicializar

Paso a paso # 1 Pasos/vuelta 256 Pin-1 8 Pin-2 9 Pin-3 10 Pin-4 11

Paso a paso # 1 Velocidad (rpm) 30

Bucle

+ si Leer digital Pin 4 = verdadero

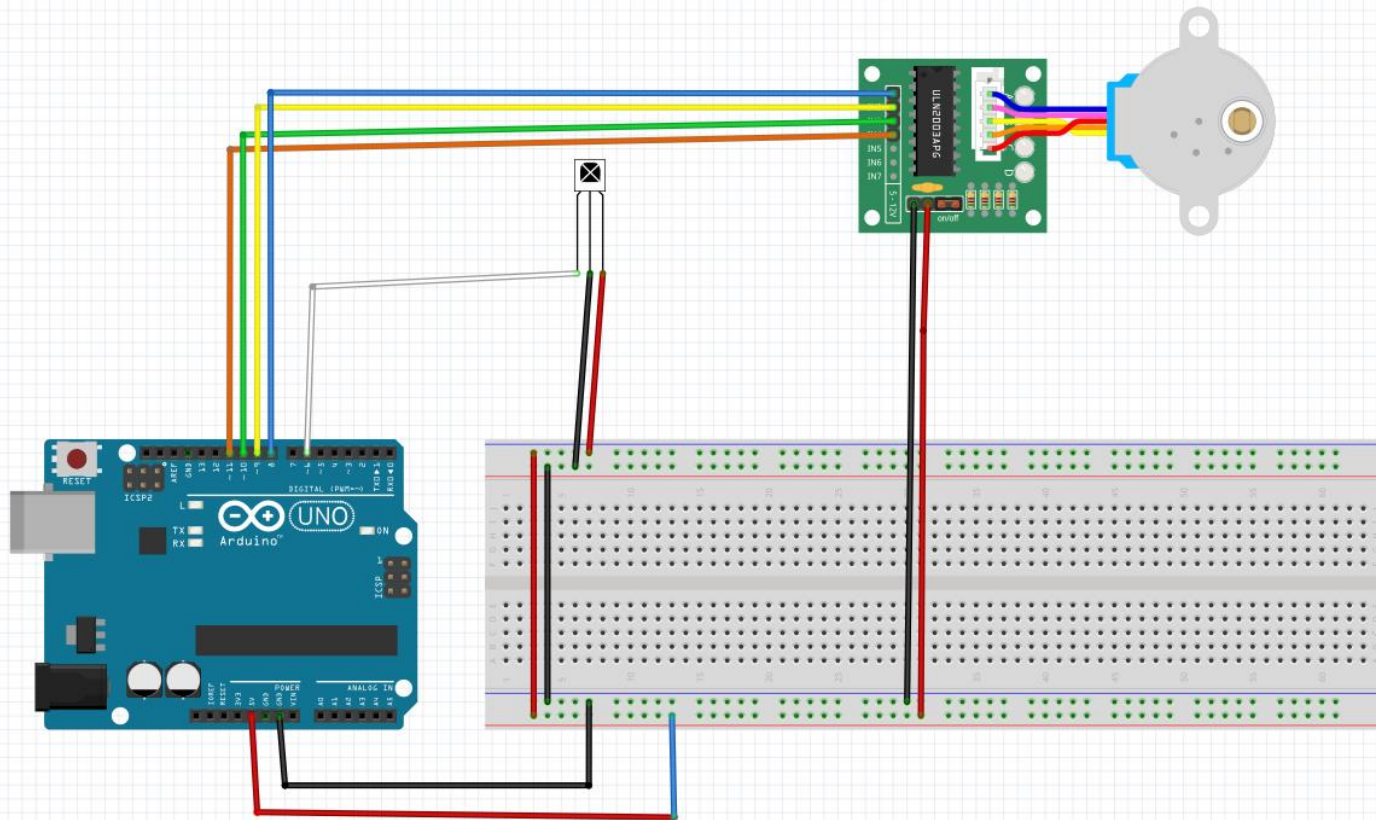
hacer Paso a paso # 1 Pasos 8

+ si Leer digital Pin 5 = verdadero

hacer Paso a paso # 1 Pasos -8

Práctica 16: Control de motor paso a paso

- Control del motor paso a paso mediante mando infrarojo



Práctica 16: Control de motor paso a paso

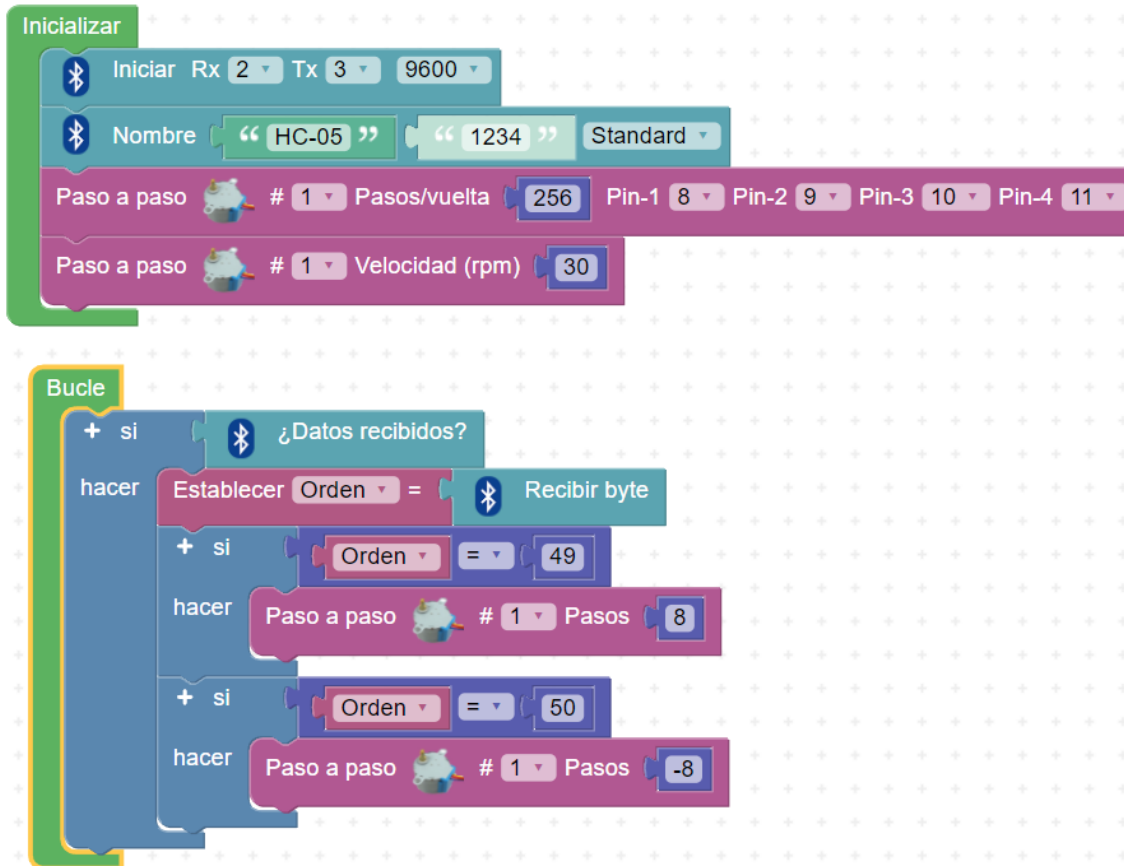
- Control del motor paso a paso mediante mando infrarojo

```
Inicializar
  Iniciar Baudios 9600
  Paso a paso # 1 Pasos/vuelta 512 Pin-1 8 Pin-2 9 Pin-3 10 Pin-4 11
  Paso a paso # 1 Velocidad (rpm) 30

Bucle
  Establecer Señal_mando_IR = Receptor de IR (Texto HEX) Pin 6
  Establecer Código_señal = Texto a número Señal_mando_IR
  + si Código_señal ≠ 0
  hacer
    >_ Enviar Código_señal Salto de línea
    + si Código_señal = 802
    hacer Paso a paso # 1 Pasos 8
    + si Código_señal = 803
    hacer Paso a paso # 1 Pasos -8
```

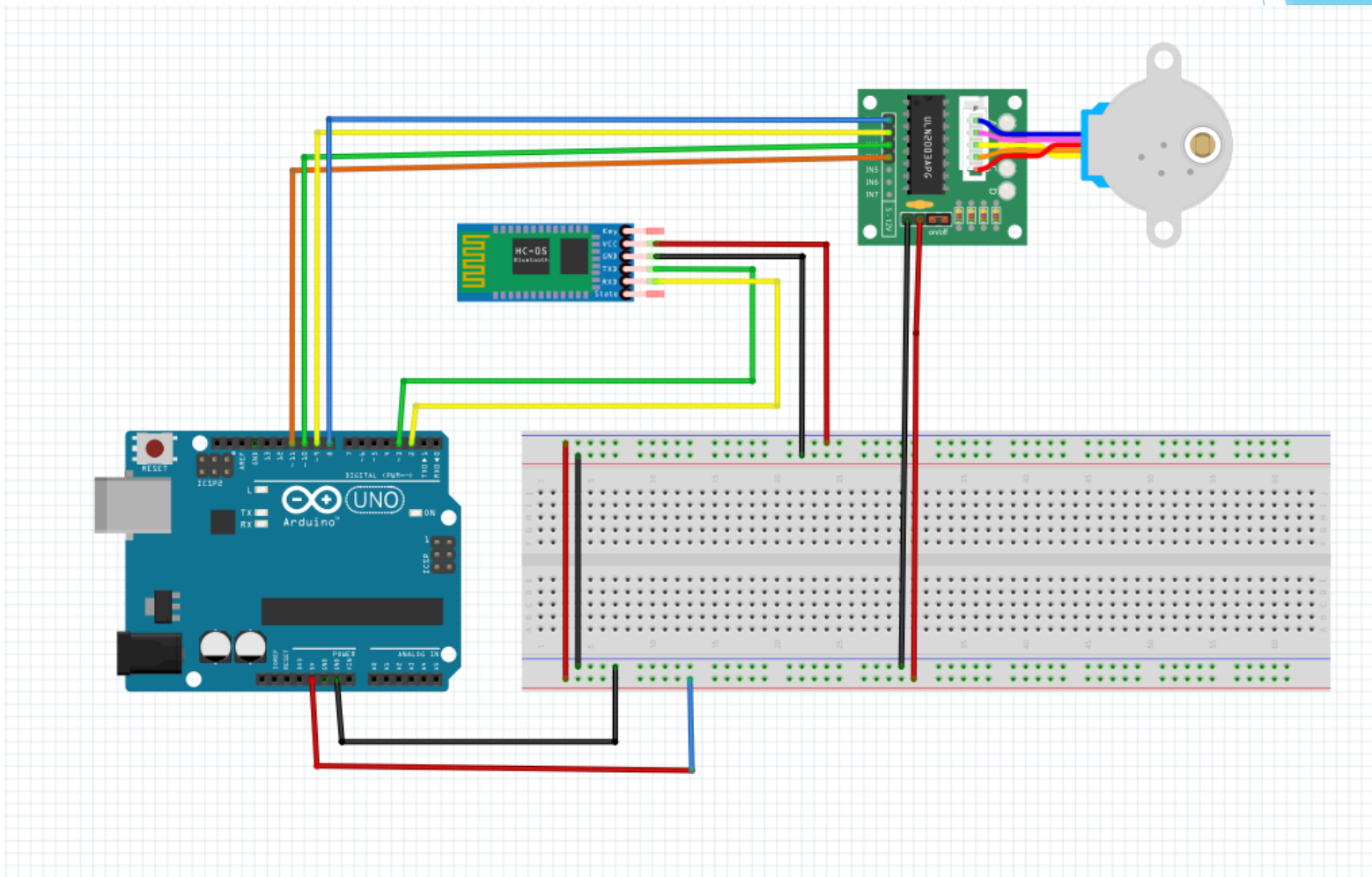

Práctica 16: Control de motor paso a paso

- ❑ Control del motor paso a paso mediante bluetooth



Práctica 16: Control de motor paso a paso

- ❑ Control del motor paso a paso mediante bluetooth



Práctica 16: Control de motor de corriente continua

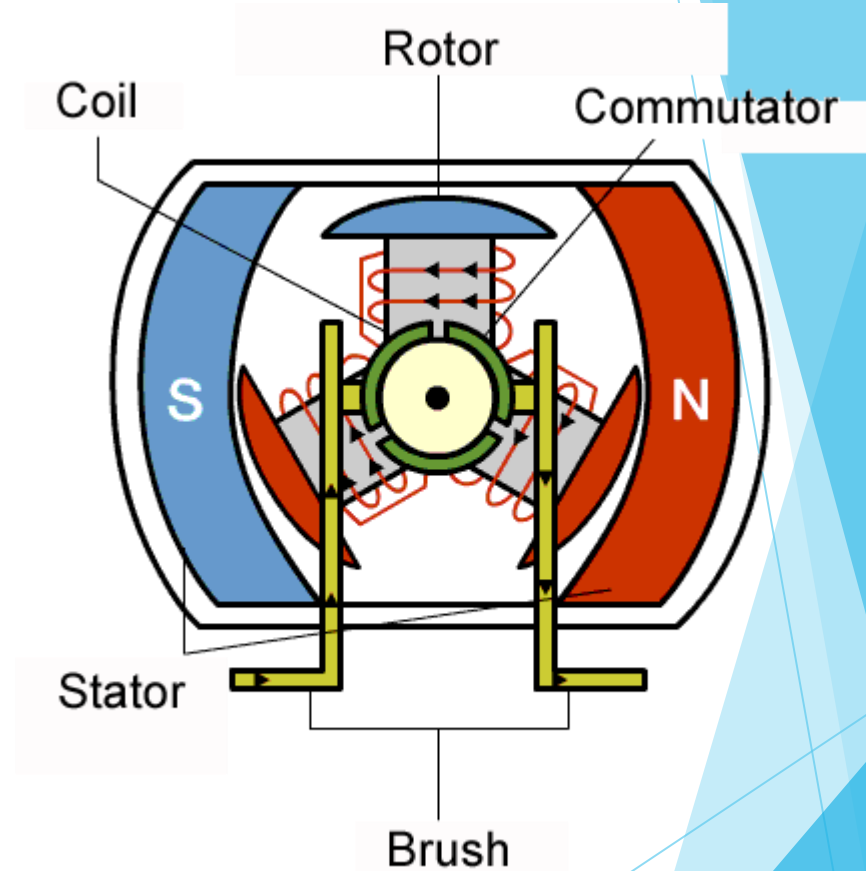


Silicon Steel Laminations



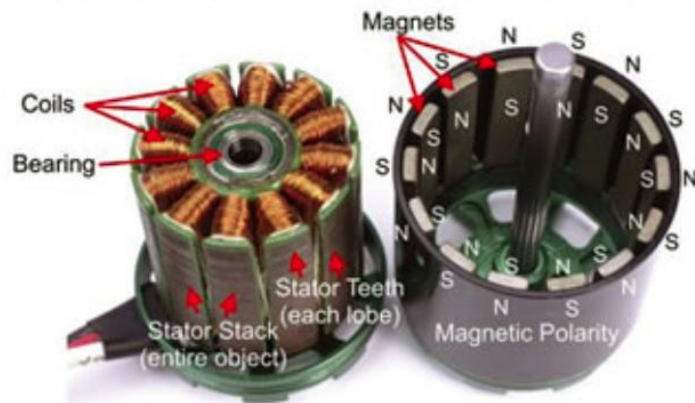
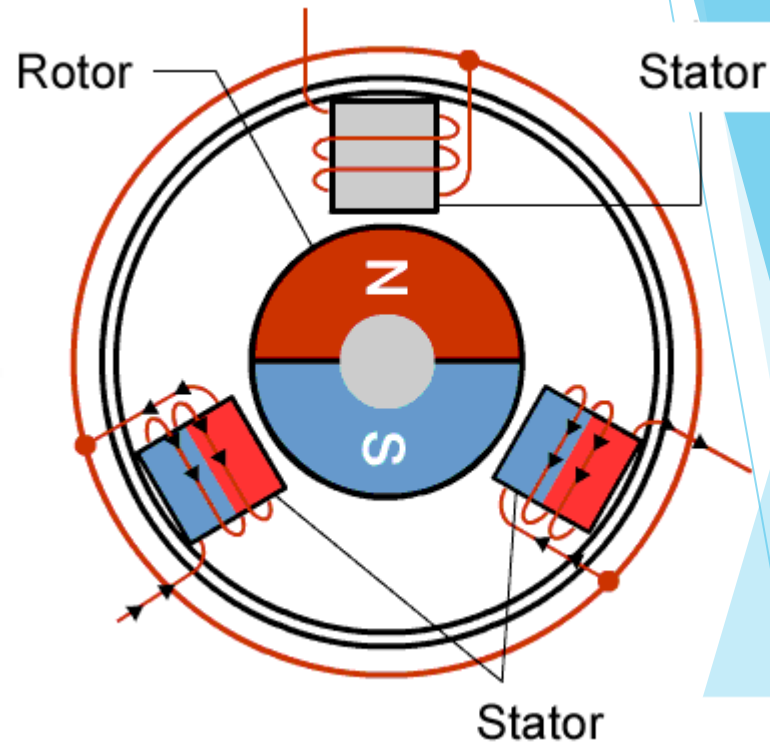
Rotor Windings

Commutator



MOTOR DE CORRIENTE CONTÍNUA CON ESCOBILLAS

Práctica 16: Control de motor de corriente continua

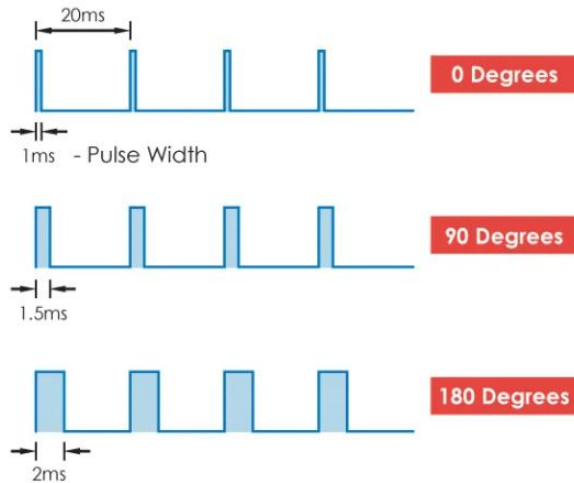


MOTOR DE CORRIENTE CONTÍNUA SIN ESCOBILLAS - BRUSHLESS

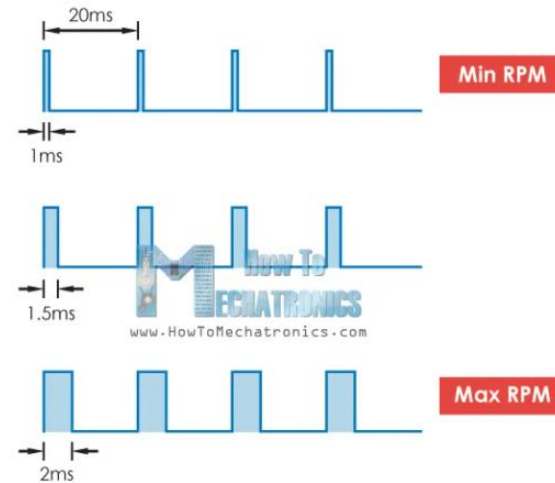
Práctica 16: Control de motor de corriente continua



SERVO MOTOR CONTROL



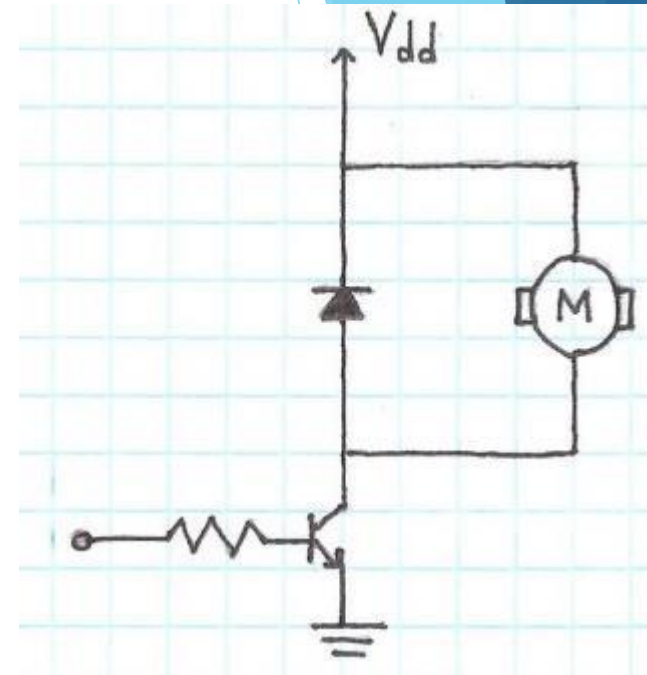
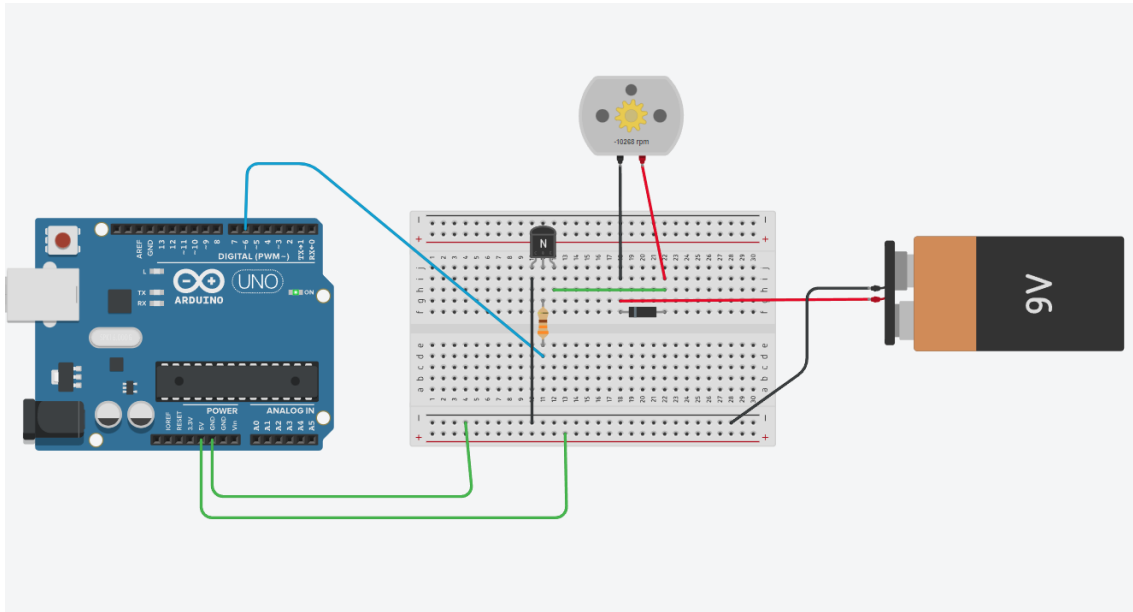
BLDC MOTOR CONTROL



MOTOR DE CORRIENTE CONTÍNUA SIN ESCOBILLAS - BRUSHLESS

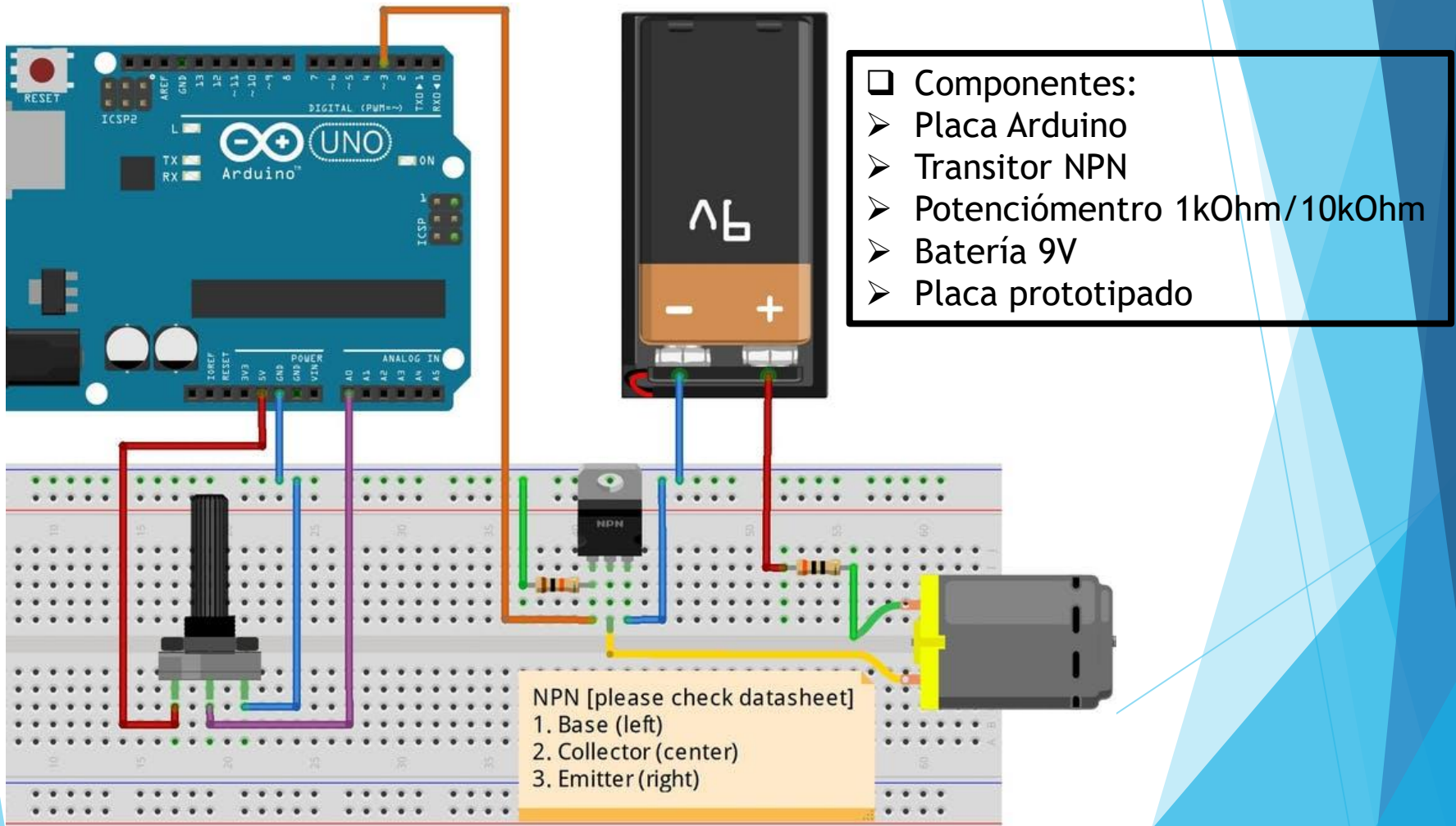
Práctica 16: Control de motor de corriente contónua

- ❑ Control de motor cc mediante transistor npn. Esquema de conexión.



Práctica 16: Control de motor de corriente contónua

❑ Control de motor cc mediante transistor BC 548. Esquema de conexión.



Práctica 16: Control de motor de corriente contónua

- Control de motor cc mediante transistor BC 548. Código.

Inicializar

Iniciar Baudios 9600

Bucle

Ejecutar cada 1000 ms

Enviar Velocidad Salto de línea

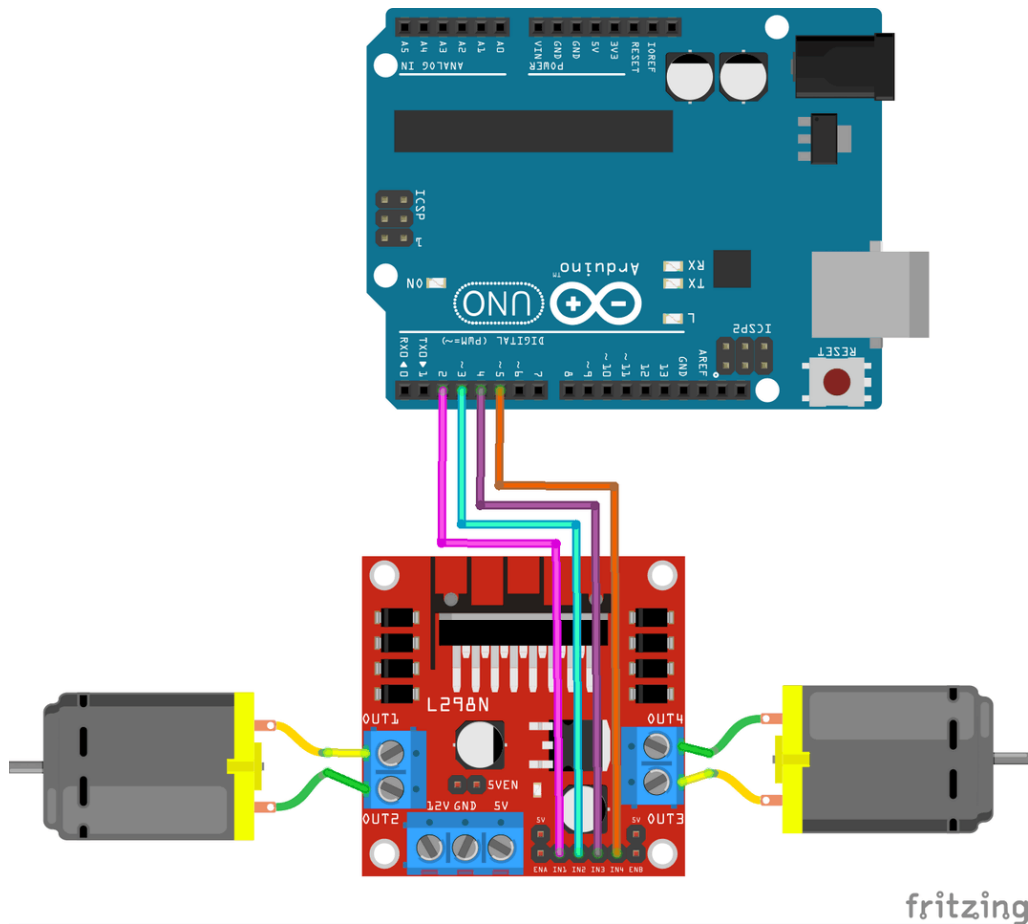
Establecer Lectura_pot = Leer analógica Pin A0

Establecer Velocidad = mapear Lectura_pot de 0 - 1023 a 0 - 255

Escribir analógica (PWM) Pin 6 Valor Velocidad

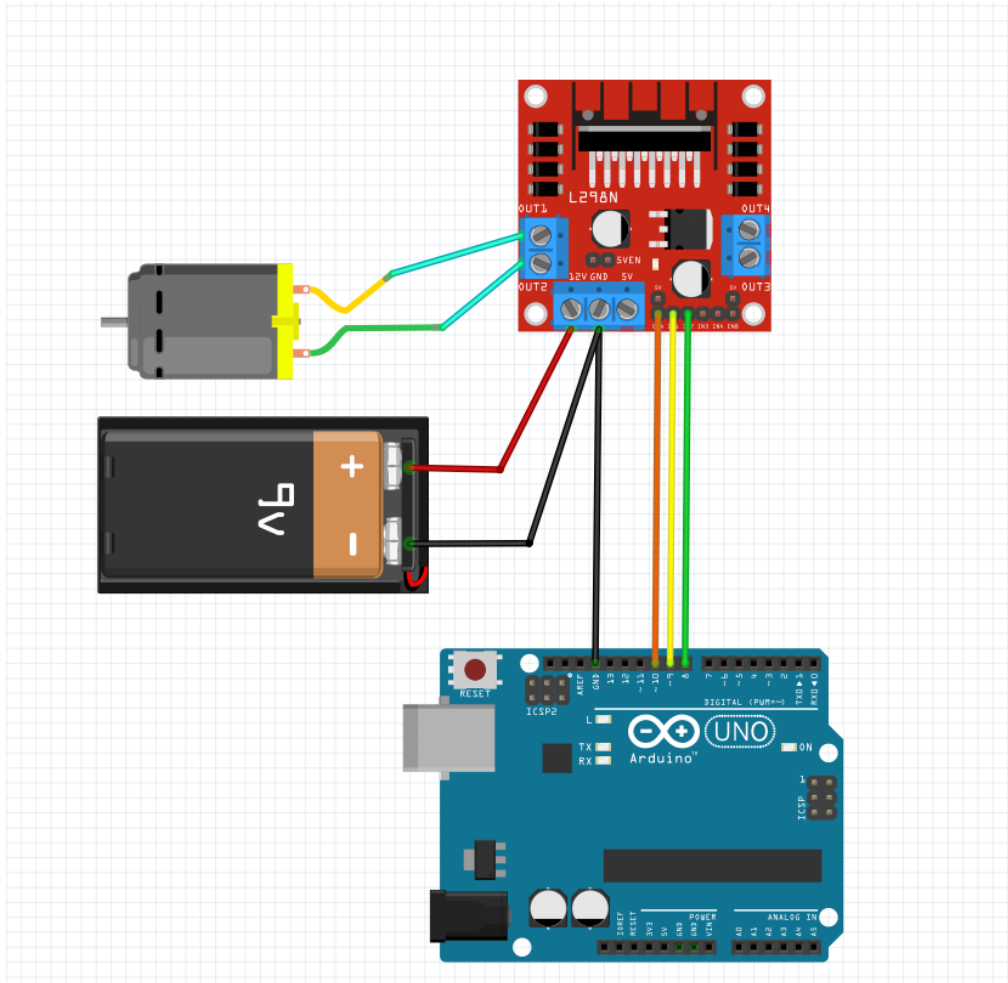
Práctica 16: Control de motor de corriente continua

- ❑ Control de motor cc mediante drive motor. L298N Esquema de conexión. 2 motores sin variación de velocidad.



Práctica 16: Control de motor de corriente continua

- ❑ Control de motor cc mediante drive motor. L298N Esquema de conexión. Un motor con variación de velocidad.



Práctica 16: Control de motor de corriente contónua

- ❑ Control de motor cc mediante drive motor. Variamos la velocidad de giro mediante un potenciómetro.

```
Inicializar
  Iniciar Baudios 9600

Bucle
  Establecer Lectura_Pot = Leer analógica Pin A0
  Establecer Velocidad = mapear Lectura_Pot de 0 - 1023 a 0 - 255
  Enviar Velocidad con Salto de línea
  Escribir analógica (PWM) Pin 10 Valor Velocidad
  Escribir digital Pin 8 OFF
  Escribir digital Pin 9 ON
```

Práctica 16: Control de motor de corriente continua.

- Control de motor cc mediante drive motor. Control con 3 pulsadores

The code is organized into three main sections: Initialization, a main loop, and three event-driven blocks for button presses.

- Inicializar:** A block to "Iniciar Baudios" (Start Baudios) set to 9600.
- Bucle (Loop):** A continuous loop containing:
 - "Establecer Lectura_Pot = Leer analógica Pin A0" (Set Pot Reading = Read Analog Pin A0)
 - "Establecer Velocidad = mapear Lectura_Pot de 0 - 1023 a 0 - 255" (Set Velocity = map Pot Reading from 0-1023 to 0-255)
 - "Enviar Velocidad" (Send Velocity) with "Salto de línea" (New line) checked.
 - "Escribir analógica (PWM) Pin 10 Valor Velocidad" (Write Analog (PWM) Pin 10 Value Velocity)
 - "Escribir digital Pin 8 OFF" (Write Digital Pin 8 OFF)
 - "Escribir digital Pin 9 ON" (Write Digital Pin 9 ON)
 - A conditional block: "si Pulsador Pin 2 Invertir" (if Button Pin 2 Inverted) followed by a "hacer Adelante" (do Forward) block.
 - A conditional block: "si Pulsador Pin 3 Invertir" (if Button Pin 3 Inverted) followed by a "hacer Atrás" (do Backward) block.
 - A conditional block: "si Pulsador Pin 4 Invertir" (if Button Pin 4 Inverted) followed by a "hacer Parada" (do Stop) block.
- para Parada (Stop):** A block triggered by a button press, containing:
 - "Escribir digital Pin 8 OFF" (Write Digital Pin 8 OFF)
 - "Escribir digital Pin 9 OFF" (Write Digital Pin 9 OFF)
- para Adelante (Forward):** A block triggered by a button press, containing:
 - "Escribir analógica (PWM) Pin 10 Valor 255" (Write Analog (PWM) Pin 10 Value 255)
 - "Escribir digital Pin 8 OFF" (Write Digital Pin 8 OFF)
 - "Escribir digital Pin 9 ON" (Write Digital Pin 9 ON)
- para Atrás (Backward):** A block triggered by a button press, containing:
 - "Escribir analógica (PWM) Pin 10 Valor 255" (Write Analog (PWM) Pin 10 Value 255)
 - "Escribir digital Pin 8 ON" (Write Digital Pin 8 ON)
 - "Escribir digital Pin 9 OFF" (Write Digital Pin 9 OFF)

Práctica 17: Medida de potencia de aerogenerador



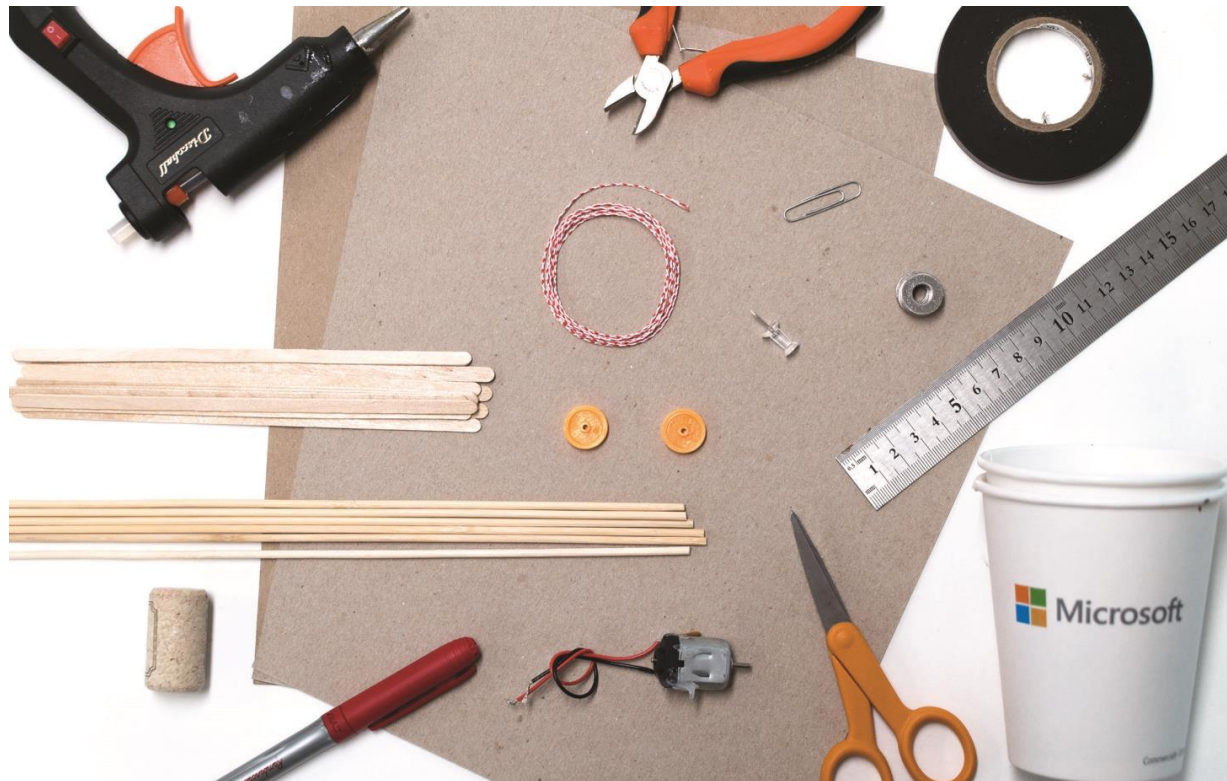
Práctica 17: Medida de potencia de aerogenerador



- ❑ En esta práctica diseñaremos un aerogenerador sencillo a partir de materiales caseros. Este aerogenerador moverá un pequeño motor de cc cuya potencia generada intentaremos medir con Arduino.

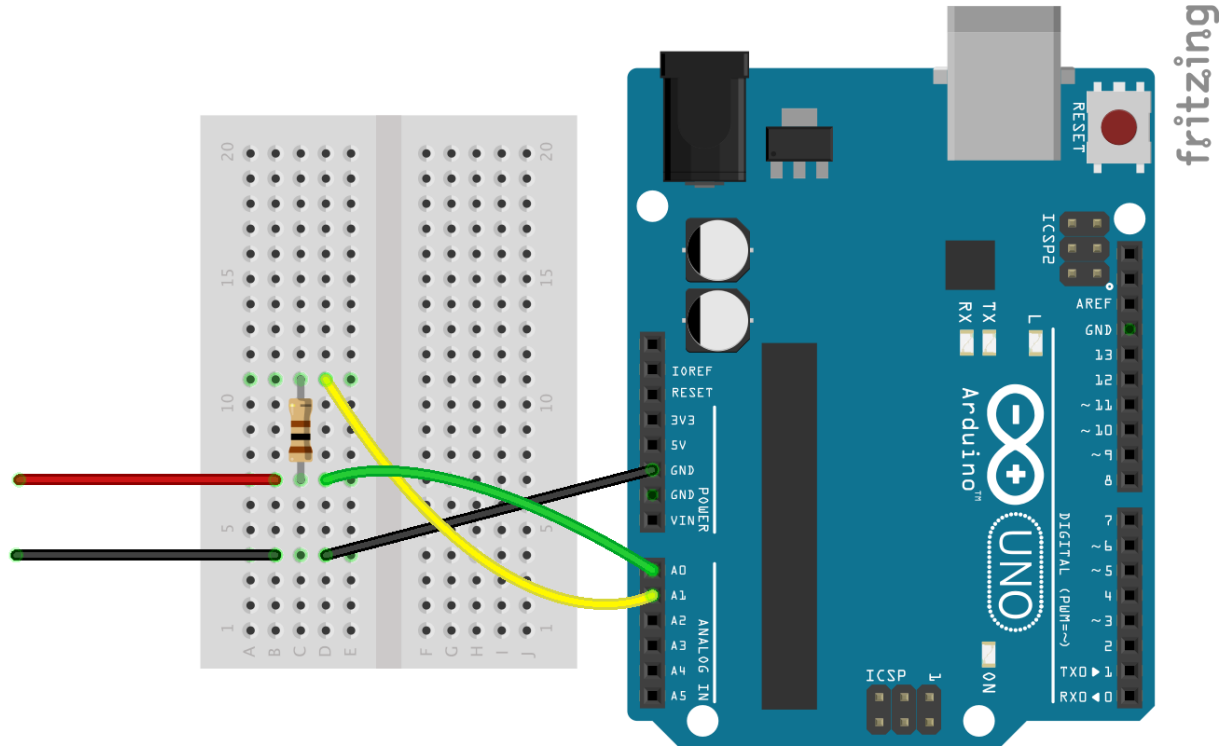
Práctica 17: Medida de potencia de aerogenerador

❑ Materiales



Práctica 17: Medida de potencia de aerogenerador

- ❑ Esquema de montaje:



Práctica 17: Medida de potencia de aerogenerador

❑ Código:

```
Inicializar
LCD # 1 Iniciar 2x16 I2C ADDR 0x27 *
Establecer Voltaje = 0
Establecer Intensidad = 0
Establecer Potencia = 0
Establecer Energía Kwh = 0
Establecer Energía (J) = 0
LCD # 1 Imprimir Columna 0 Fila 0 " Bienvenidos "
LCD # 1 Imprimir Columna 0 Fila 1 " Medidor de potencia "
Esperar 2000 milisegundos
```

```
Bucle
LCD # 1 Limpiar
Ejecutar cada 1000 ms
  repetir 100 veces
    hacer
      Establecer Lectura A0 = Leer analógica Pin A0
      + si Lectura A0 > Voltaje
        hacer
          Establecer Voltaje = Lectura A0
        Esperar 1 milisegundos
      Establecer Lectura A1 = Leer analógica Pin A1
      Establecer Intensidad = Lectura A1 ÷ 1024
      Establecer Intensidad = Intensidad × 100
      Establecer Intensidad = Intensidad × 5
      Establecer Potencia = Voltaje × Intensidad
      Establecer Energía (J) = Potencia + Energía (J)
    LCD # 1 Imprimir Columna 0 Fila 0 + - crear texto con + - crear texto con " Pot= "
    " W "
    LCD # 1 Imprimir Columna 0 Fila 1 + - crear texto con + - crear texto con " E= "
    " J "
```

Práctica 18: Escudo térmico

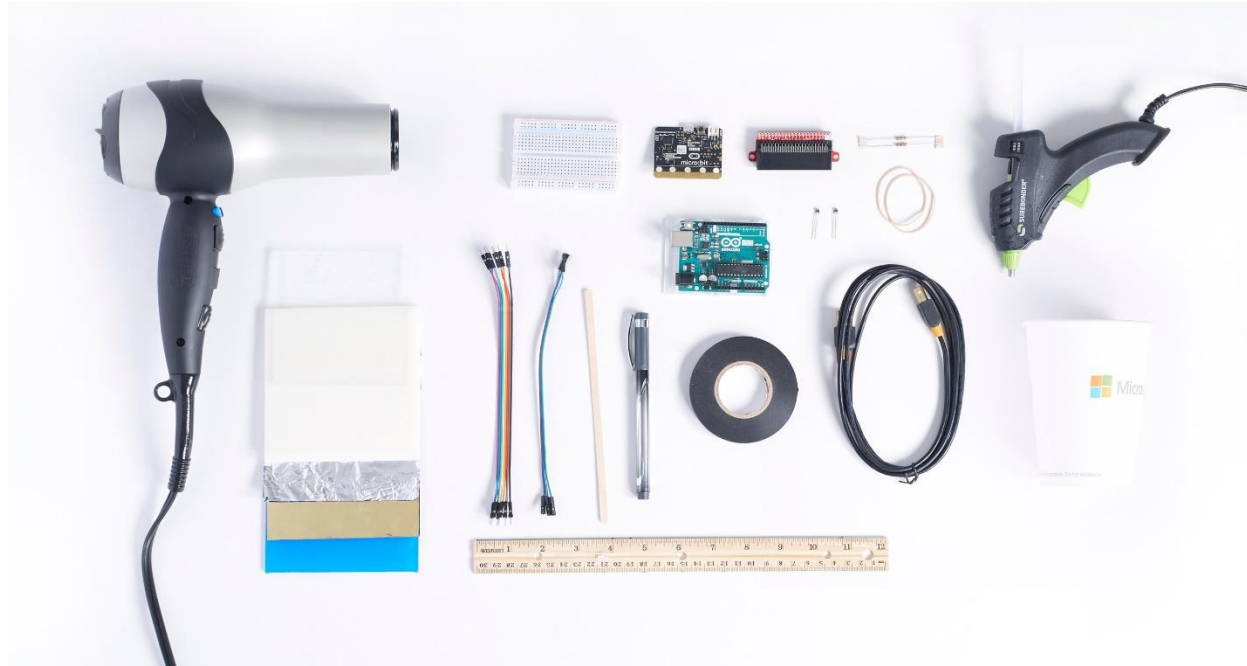
- ❑ En esta práctica vamos a utilizar Arduino para comprobar las diferentes conductividades térmicas que poseen los materiales y comprobar cuáles son los más adecuados para aplicaciones como los recubrimientos de naves espaciales.



Práctica 18: Escudo térmico



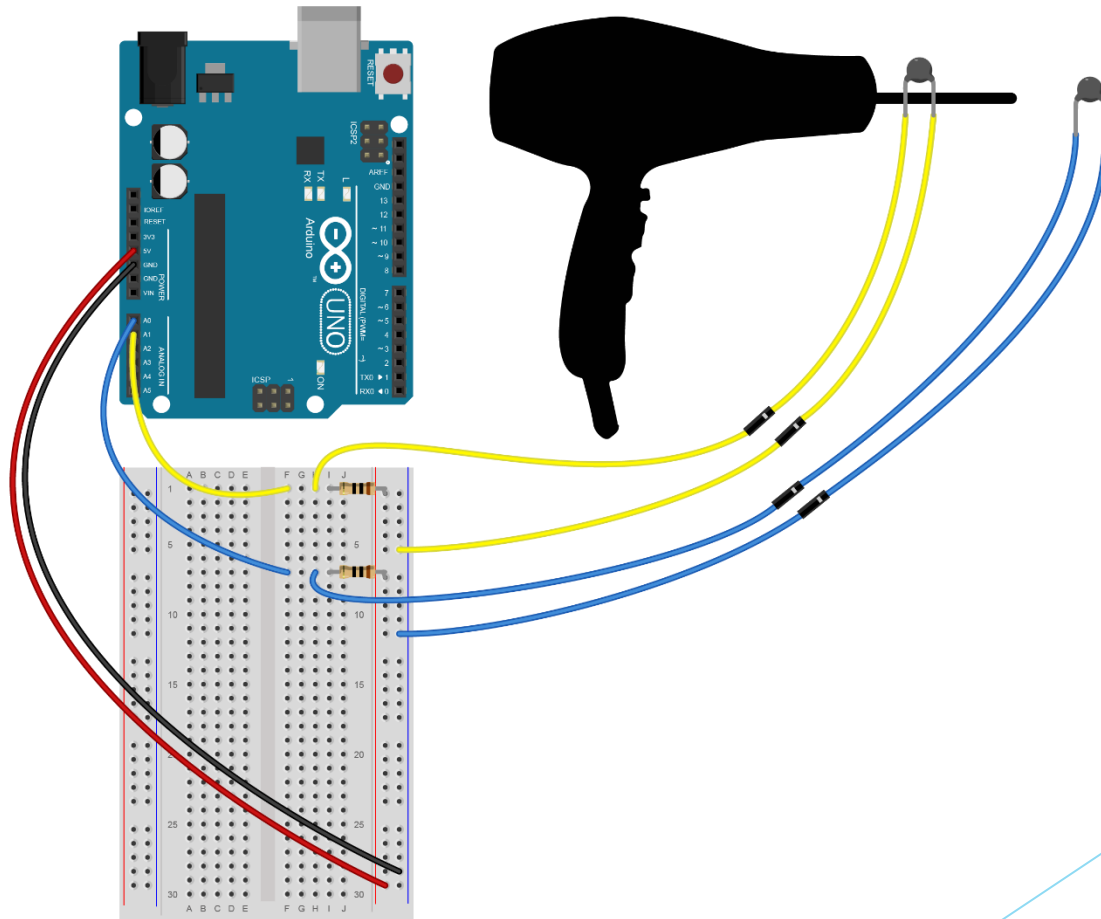
Práctica 18: Escudo térmico



- ❑ Para simular el calor generado en la reentrada utilizaremos un secador de pelo. Después utilizaremos dos termisoteres para medir la temperatura fuera y dentro de la cápsula.

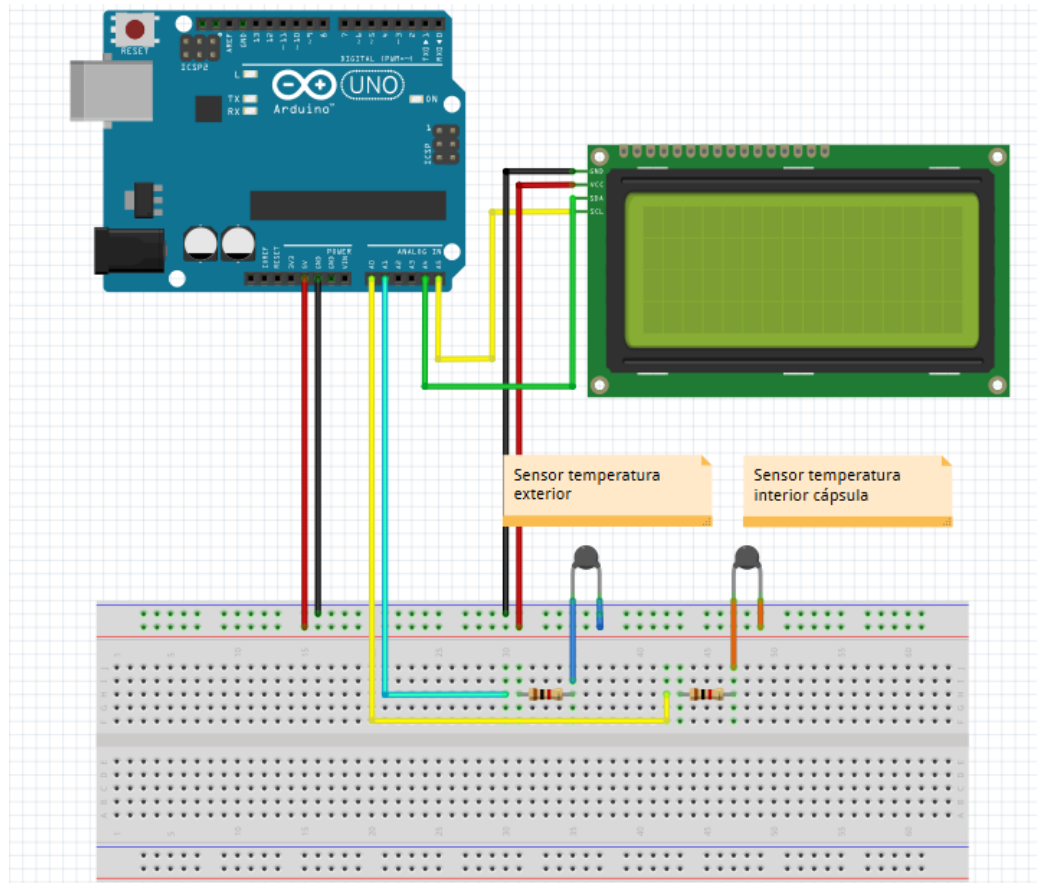
Práctica 18: Escudo térmico

- ❑ Esquema general de montaje.



Práctica 18: Escudo térmico

- ❑ Esquema de montaje para medida de temperaturas y visionado por LCD.



Práctica 18: Escudo térmico

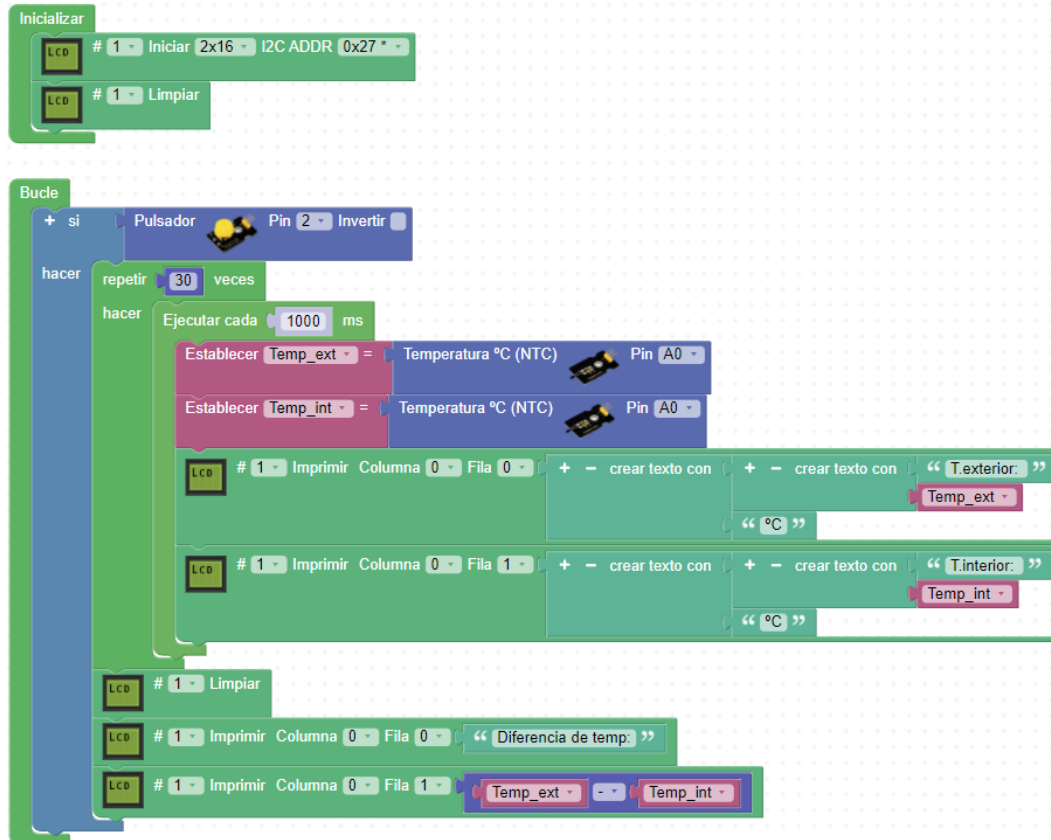
- ❑ Código de medida de temperaturas interior y exterior de la cápsula.

```
Inicializar
  LCD # 1 Iniciar 2x16 I2C ADDR 0x27 *
  LCD # 1 Limpiar

Bucle
  Ejecutar cada 1000 ms
    Establecer Temp_ext = Temperatura °C (NTC) Pin A0
    Establecer Temp_int = Temperatura °C (NTC) Pin A0
    LCD # 1 Imprimir Columna 0 Fila 0 + - crear texto con + - crear texto con " T.exterior: "
    Temp_ext
    " °C "
    LCD # 1 Imprimir Columna 0 Fila 1 + - crear texto con + - crear texto con " T.interior: "
    Temp_int
    " °C "
```

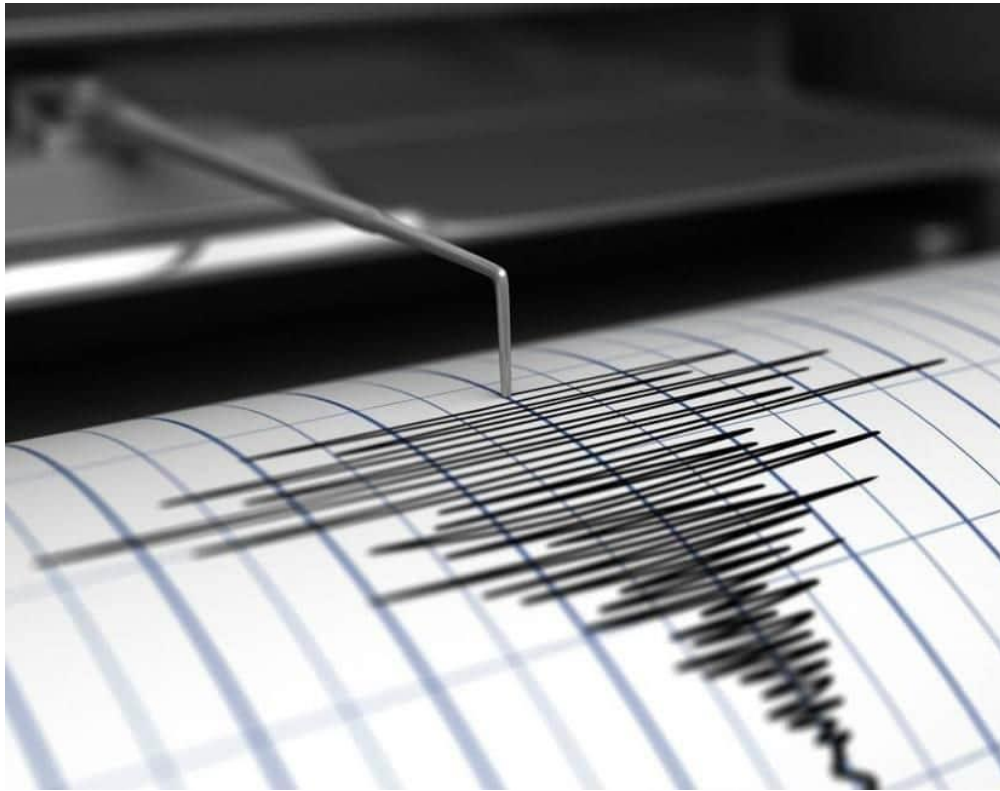
Práctica 18: Escudo térmico

- ❑ Medida de diferencia de temperaturas exterior e interior tras 30s.



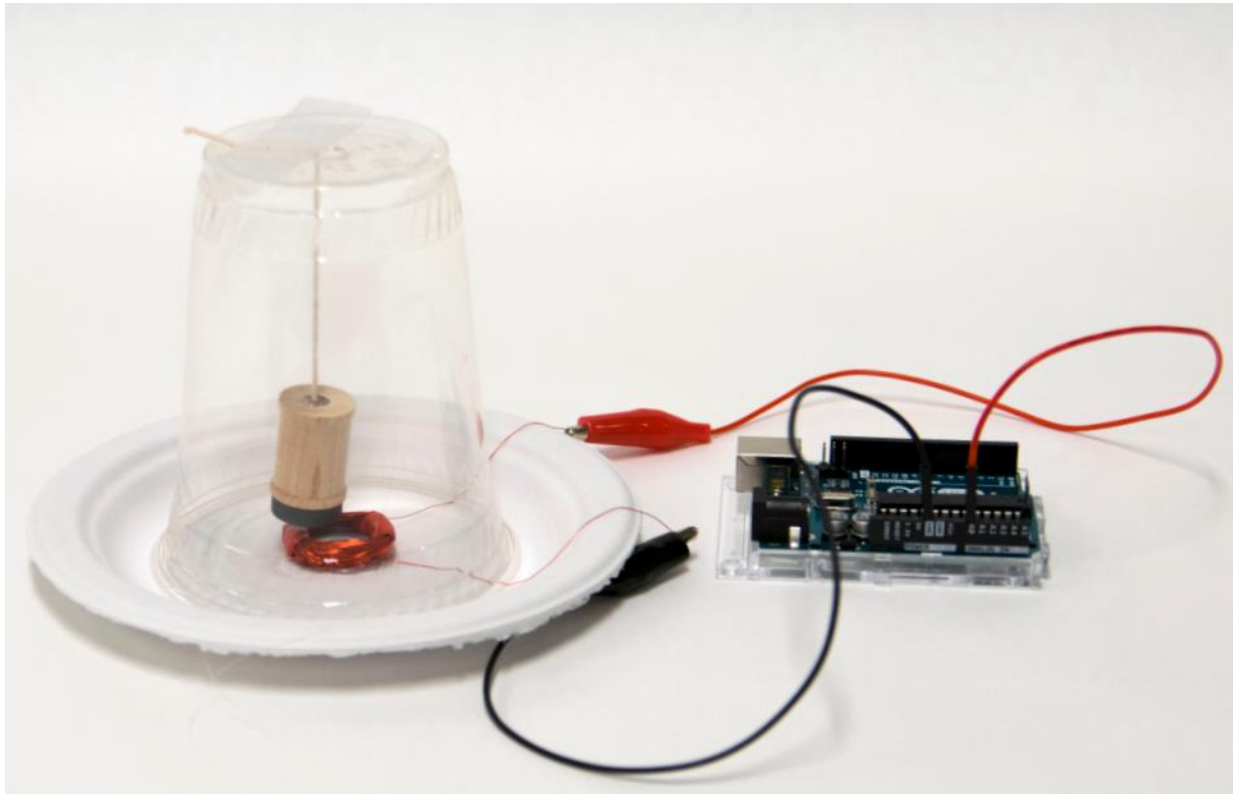
Práctica 19: Sismógrafo

- ❑ Con este programa vamos a simular el comportamiento de un sismógrafo, diseñando un dispositivo capaz de detectar vibraciones y temblores de tierra.



Práctica 19: Sismógrafo

- ❑ Al simular el terremoto el imán suspendido por el corcho se moverá, variando el campo magnético que atraviesa la espira e induciendo una corriente que podrá ser medida por Arduino



Práctica 19: Sismógrafo

❑ Materiales

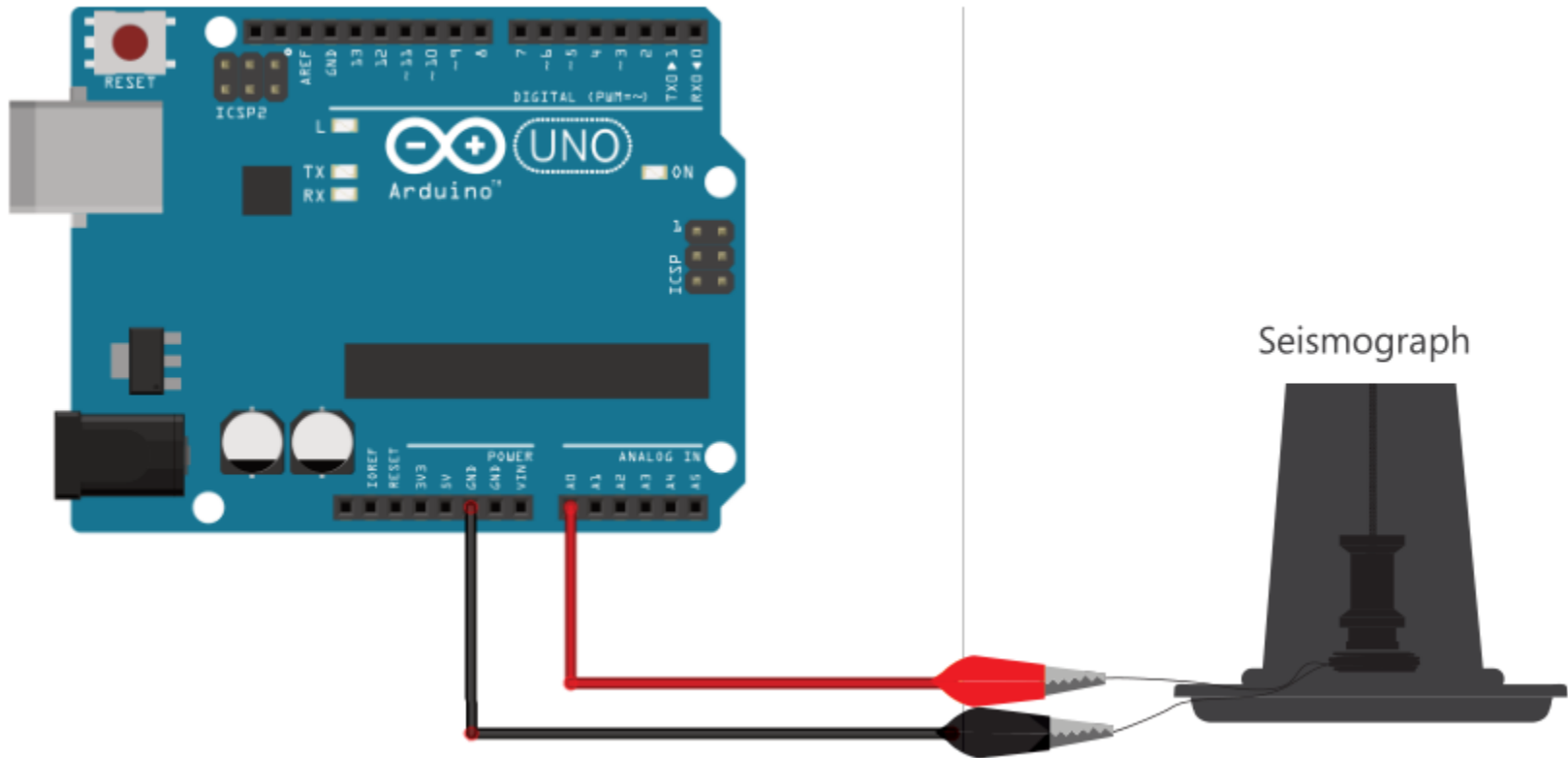


1 carrete de madera para manualidades
1 imán de disco (1.9 cm)
1 extensión de cordón, 15 cm
1 arandela de acero o de hierro

1 extensión de cable magnético de 32 awg, 3.60 m
2 pinzas de contacto
1 copa de 12 oz de plástico transparente
1 plato de papel
1 pieza de papel cartón

Práctica 19: Sismógrafo

- ❑ Esquema de conexión:



Práctica 19: Sismógrafo

❑ Código

```
Scratch Code for Sismógrafo Project:  
  
Inicializar  
- Establecer Intensidad = 0  
- LCD # 1 Iniciar 2x16 I2C ADDR 0x27 *  
- LCD # 1 Limpiar  
  
Bucle  
- Establecer Lectura A0 = Leer analógica Pin A0  
- Establecer Intensidad = mapear Lectura A0 de 0 - 1023 a 0 - 10  
- LCD # 1 Imprimir Columna 0 Fila 0 "Intensidad temblor:"  
- LCD # 1 Imprimir Columna 0 Fila 1 Intensidad
```

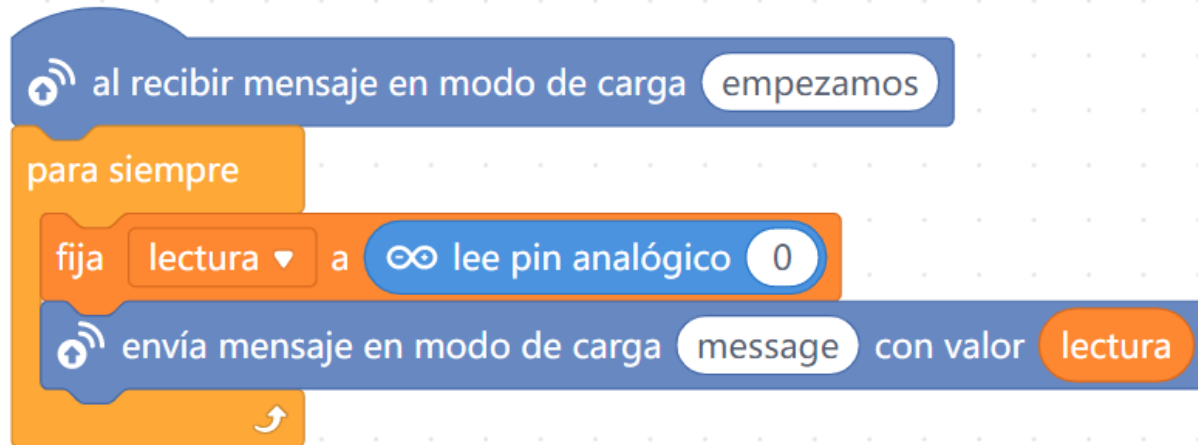
Práctica 19: Sismógrafo

- Ahora vamos a intentar simular el sismógrafo utilizando Scratch.



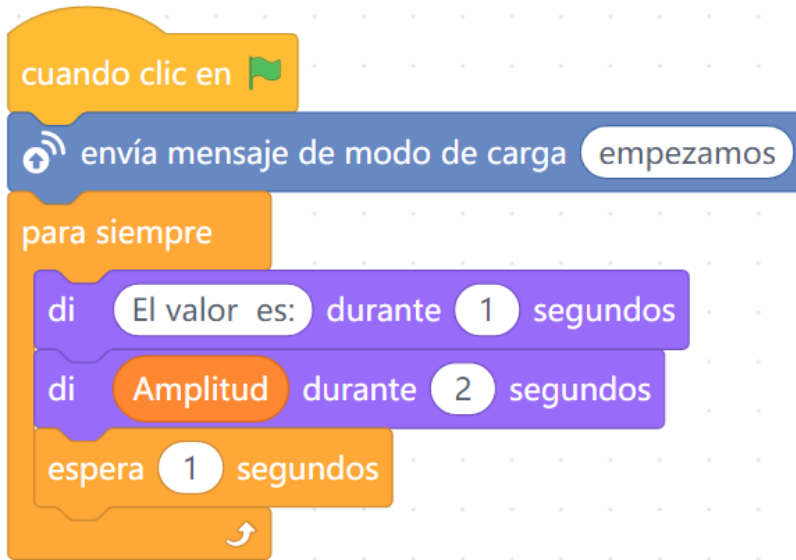
Práctica 19: Sismógrafo

- ❑ Arduino se encarga de leer el voltaje generado por el movimiento del imán y de enviarlo al Panda para su visionado.

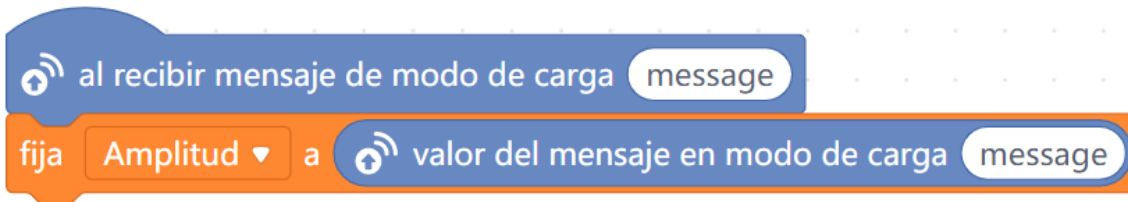


Práctica 19: Sismógrafo

- ❑ El Panda se encarga de visualizar el valor de la amplitud del temblor.



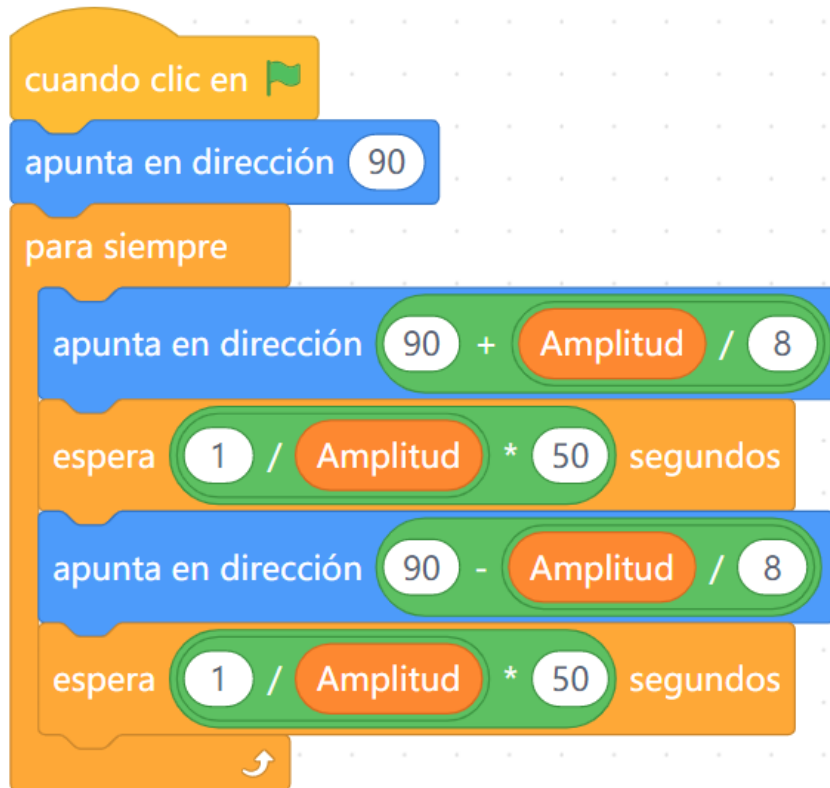
```
cuando clic en [bandera]
  envía mensaje de modo de carga [empezamos]
  para siempre
    di [El valor es: durante 1 segundos]
    di [Amplitud durante 2 segundos]
  espera 1 segundos
```



```
al recibir mensaje de modo de carga [message]
  fija [Amplitud] a [valor del mensaje en modo de carga message]
```


Práctica 19: Sismógrafo

- ❑ La aguja se moverá con un ángulo y velocidad directamente proporcionales a la amplitud del temblor.



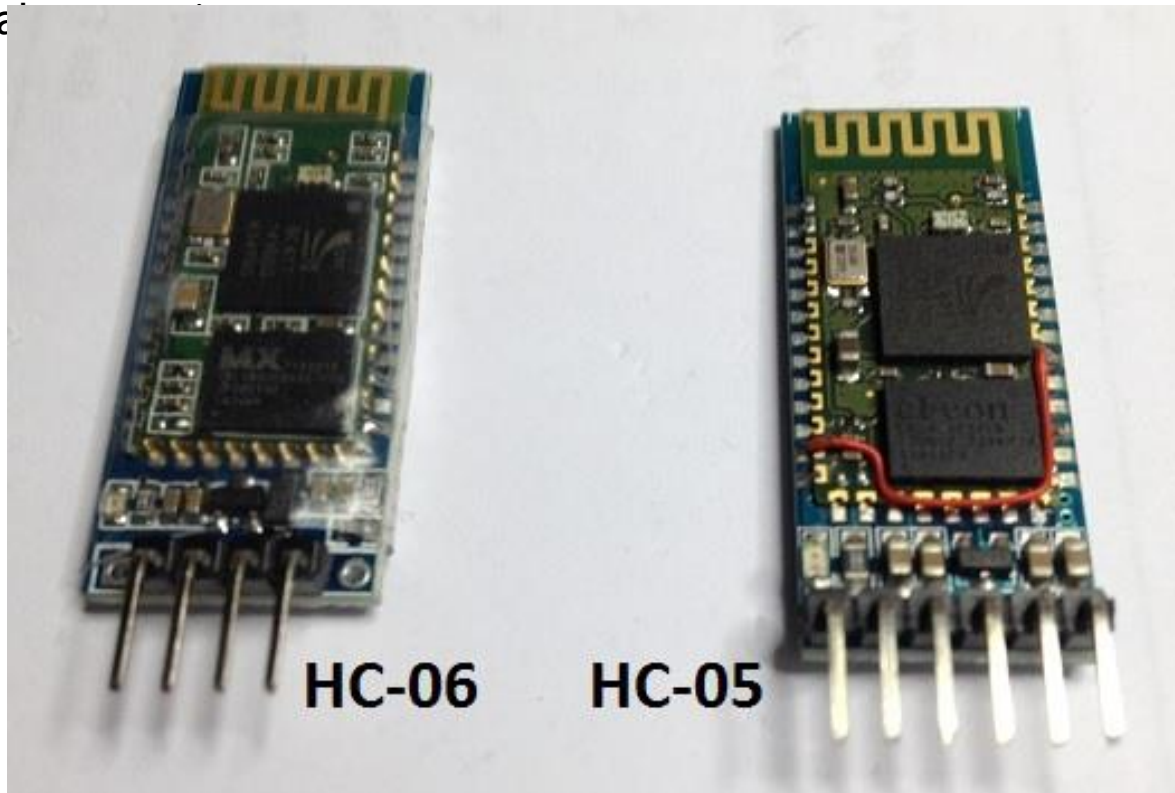
Práctica 20: Comunicación bluetooth

- ❑ Características básicas.
- La tecnología Bluetooth permite la comunicación inalámbrica entre dispositivos digitales.
- Está especializada en transferencias de datos en distancias cortas.
- Destaca por un bajo consumo y por la sencillez de conexión.
- Posee velocidades de transferencias de datos inferiores a otras tecnologías como WI-FI.
- No necesita que los dispositivos estén “encarados” para efectuar la comunicación.
- Para comunicar el Arduino con un dispositivo móvil necesitaremos una aplicación de Terminal bluetooth.



Práctica 20: Comunicación bluetooth

- ❑ Para comunicarnos con Arduino utilizando el protocolo bluetooth utilizaremos los módulos HC-05 o HC-06.
- ❑ El hardware de ambos chips es el mismo pero presentan algunas diferencias en cuanto a funcionalidades siendo en este aspecto el HC-05



Práctica 20: Comunicación bluetooth - Configuración (0)

❑ CÓDIGO PARA PODER CONFIGURAR:

conf_bluetoh

```
#include <SoftwareSerial.h> // Incluimos la librería SoftwareSerial
SoftwareSerial BT(10,11); // Definimos los pines RX y TX del Arduino conectados al Bluetooth

void setup()
{
  BT.begin(9600); // Inicializamos el puerto serie BT (Para Modo AT 2)
  Serial.begin(9600); // Inicializamos el puerto serie
}

void loop()
{
  if(BT.available()) // Si llega un dato por el puerto BT se envía al monitor serial
  {
    Serial.write(BT.read());
  }

  if(Serial.available()) // Si llega un dato por el monitor serial se envía al puerto BT
  {
    BT.write(Serial.read());
  }
}
```

Práctica 20: Comunicación bluetooth - Configuración (1)

El módulo de comunicación bluetooth puede presentar 4 modos de funcionamiento dependiendo del modelo.

HC-05

- Posee los 4 modos de funcionamiento.
- Nombre por defecto: HC-05
- Contraseña: 1234
- Velocidad por defecto (baud rate): :9600

HC-06

- Posee sólo los modos de funcionamiento desconectado y conectado.
- Nombre por defecto: HC-06
- Contraseña: 1234
- Velocidad por defecto (baud rate): :9600

Práctica 20: Comunicación bluetooth - Configuración (2)

El módulo de comunicación bluetooth posee 4 modos de funcionamiento

➤ MODO DESCONECTADO:

- Entra a este estado tan pronto alimentas el modulo, y cuando **no se ha establecido una conexión** bluetooth con ningún otro dispositivo
- EL LED del módulo en este estado **parpadea rápidamente**
- En este estado a diferencia del HC-06, el HC-05 no puede interpretar los comandos AT

➤ MODO CONECTADO

- Entra a este estado cuando **se establece una conexión** con otro dispositivo bluetooth.
- El LED hace un **doble parpadeo** en el caso del Hc_05 y permanece fijo en el HC -06
- Todos los datos que se ingresen al HC-05 por el Pin RX se transmiten por bluetooth al dispositivo conectado, y los datos recibidos se devuelven por el pin TX. La comunicación es transparente

Práctica 20: Comunicación bluetooth - Configuración (3)

El módulo de comunicación bluetooth posee 4 modos de funcionamiento

➤ MODO AT1

- Para entrar a este estado **después de conectar y alimentar el modulo es necesario presionar el botón del HC-05.**
- En este estado, **podemos enviar comandos AT**, pero a la misma velocidad con el que está configurado.
- EL LED del módulo en este estado **parpadea rápidamente igual que en el estado desconectado.**

➤ MODO AT2

- Para entrar a este estado es necesario tener **presionado el botón al momento de alimentar el modulo**, es decir el modulo debe encender con el botón presionado, después de haber encendido se puede soltar y permanecerá en este estado.
- En este estado, para enviar comandos AT es necesario hacerlo a la velocidad de 38400 baudios, esto es muy útil cuando nos olvidamos la velocidad con la que hemos dejado configurado nuestro modulo.
- EL LED del módulo en este estado **parpadea lentamente.**

Práctica 20: Comunicación bluetooth - Configuración (4)

❑ COMANDOS BÁSICOS DE CONFIGURACIÓN:

➤ **Test de comunicación.**

Enviar: AT
Recibe: OK

➤ **Cambio de nombre del dispositivo:**

Enviar: AT+NAMEnombre Ejemplo: AT+NAMEgrupo1
Recibe: Oksetname

➤ **Cambio de código de vinculación:**

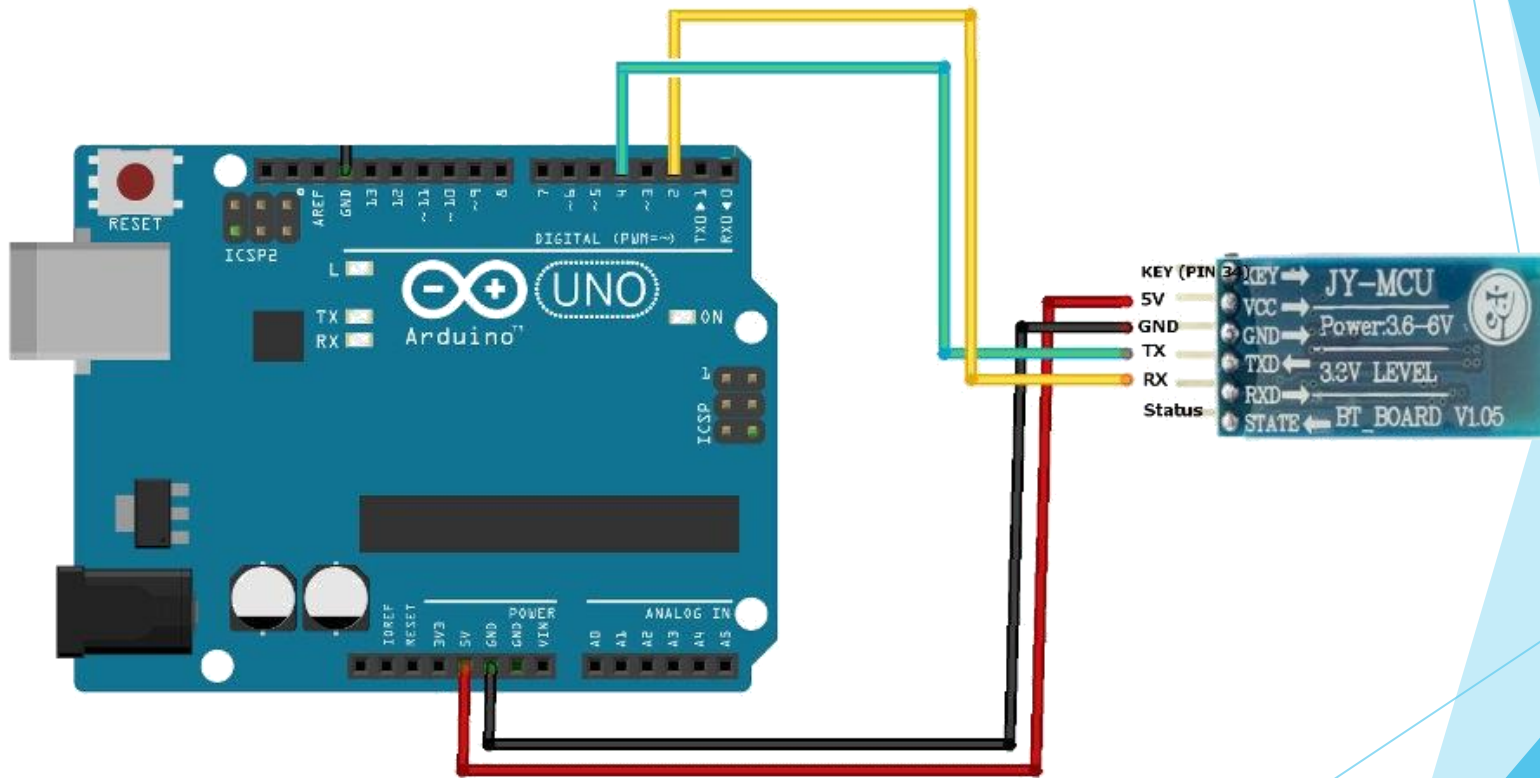
Enviar: AT+PINpin Ejemplo: AT+PIN584
Recibe: Oksetpin

➤ **Cambio de velocidad de comunicación**

Enviar: AT+BAUDvelocidad Ejemplo: AT+BAUD38400
Recibe: Okvelocidad

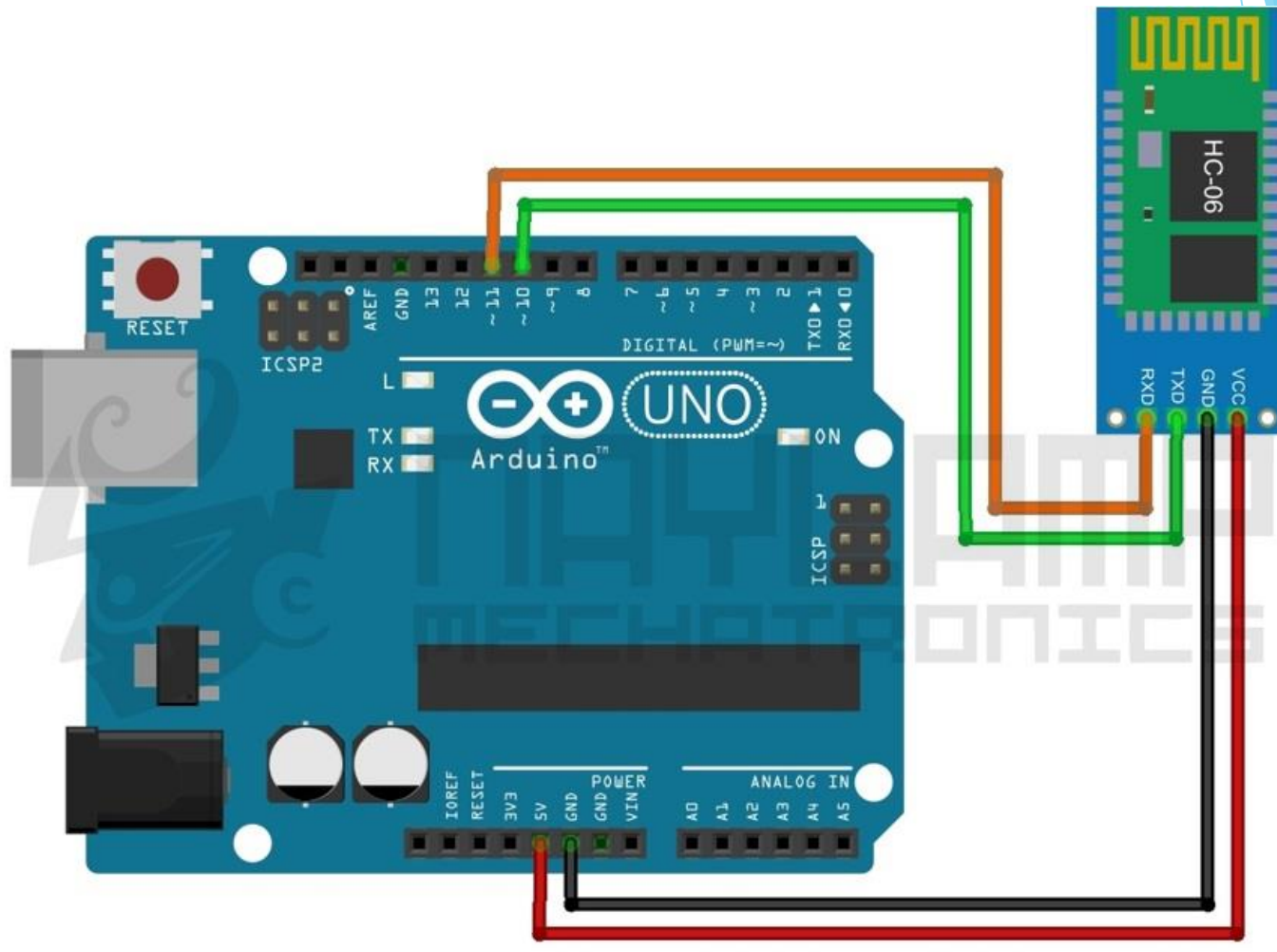
Práctica 20: Comunicación bluetooth

❑ Esquema de conexión HC-05



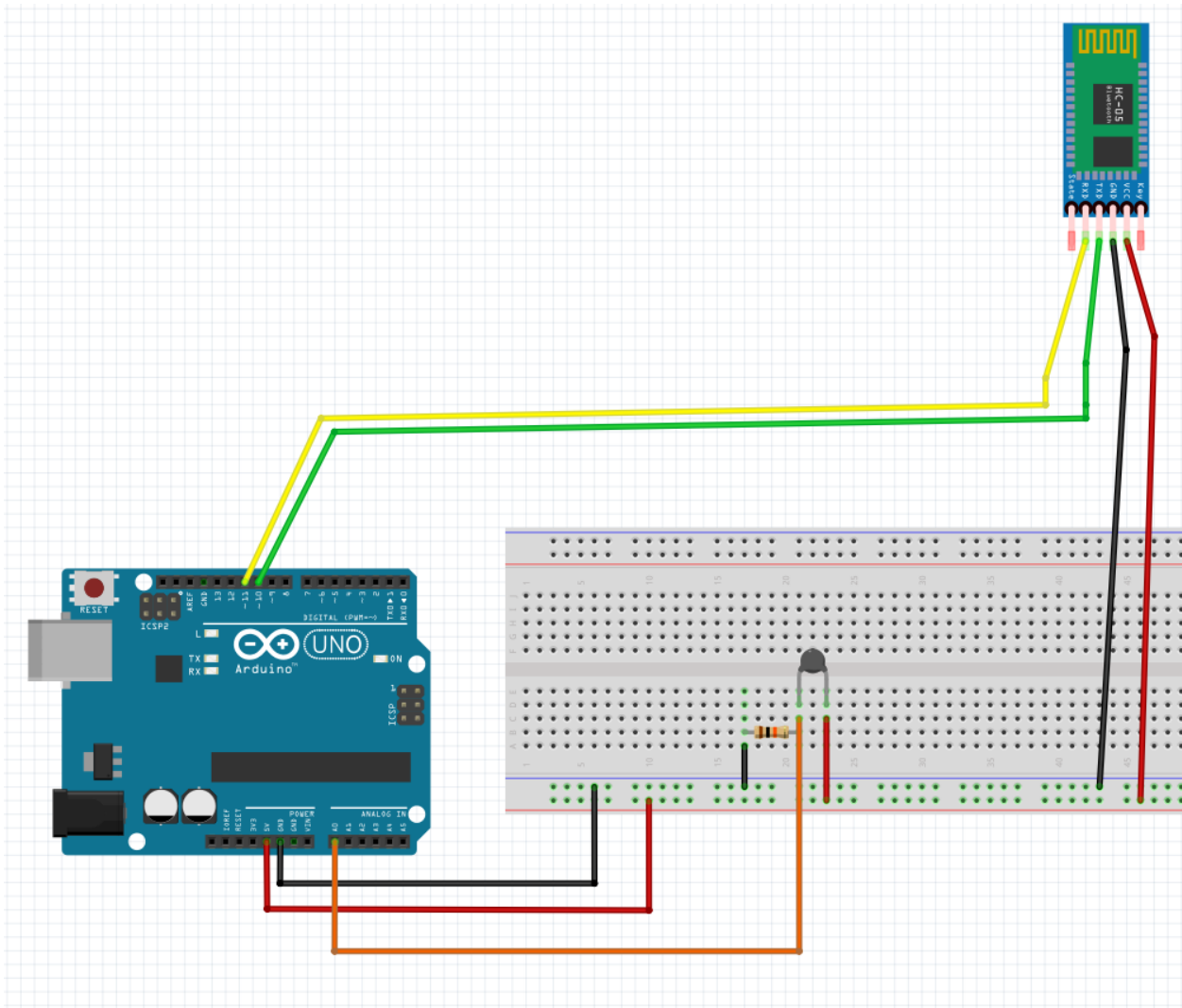
Práctica 20: Comunicación bluetooth

- ❑ Esquema de conexión HC-06



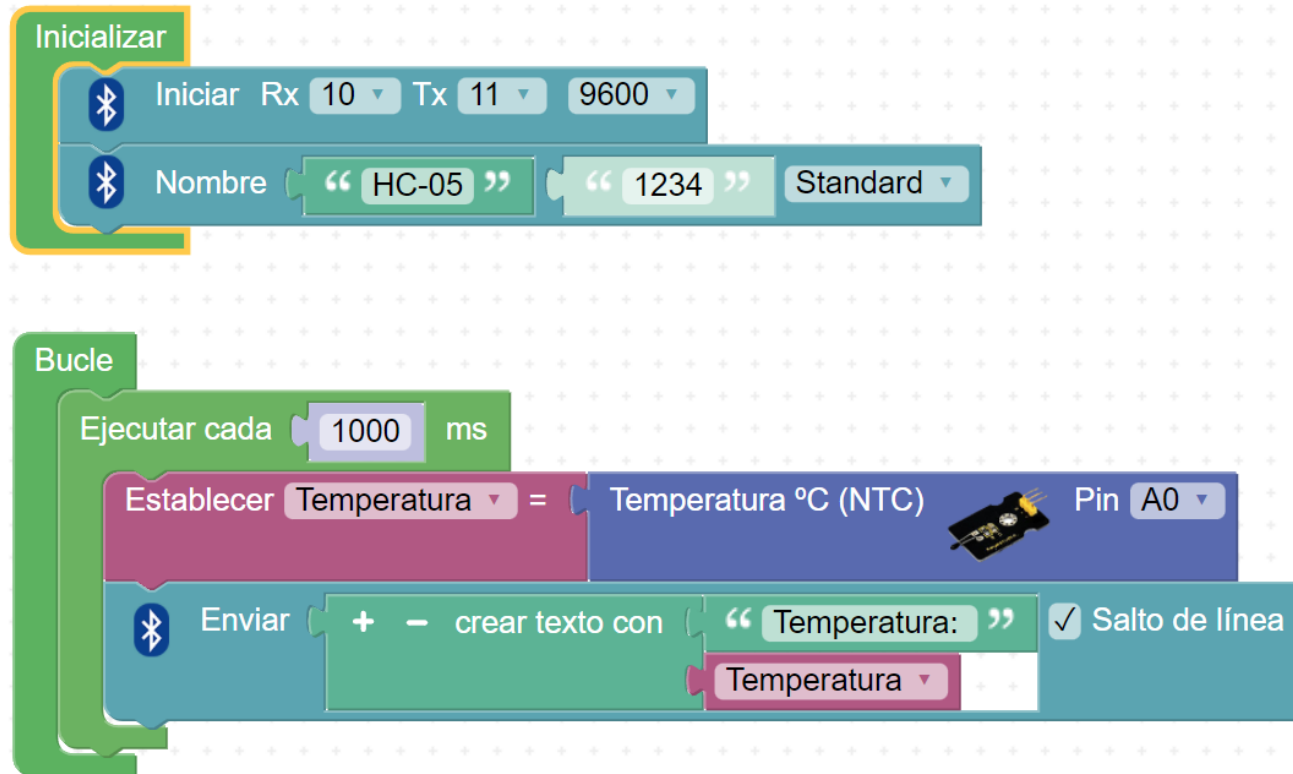
Práctica 20: Comunicación bluetooth

- ❑ Esquema de montaje para la medida de temperatura y envío a través de bluetooth mediante NTC



Práctica 20: Comunicación bluetooth

- ❑ Código de programa para la medida de temperatura y envío a través de bluetooth



The image shows a Scratch script for a temperature measurement project. It is divided into two main sections: 'Inicializar' (Initialize) and 'Bucle' (Loop).

Inicializar

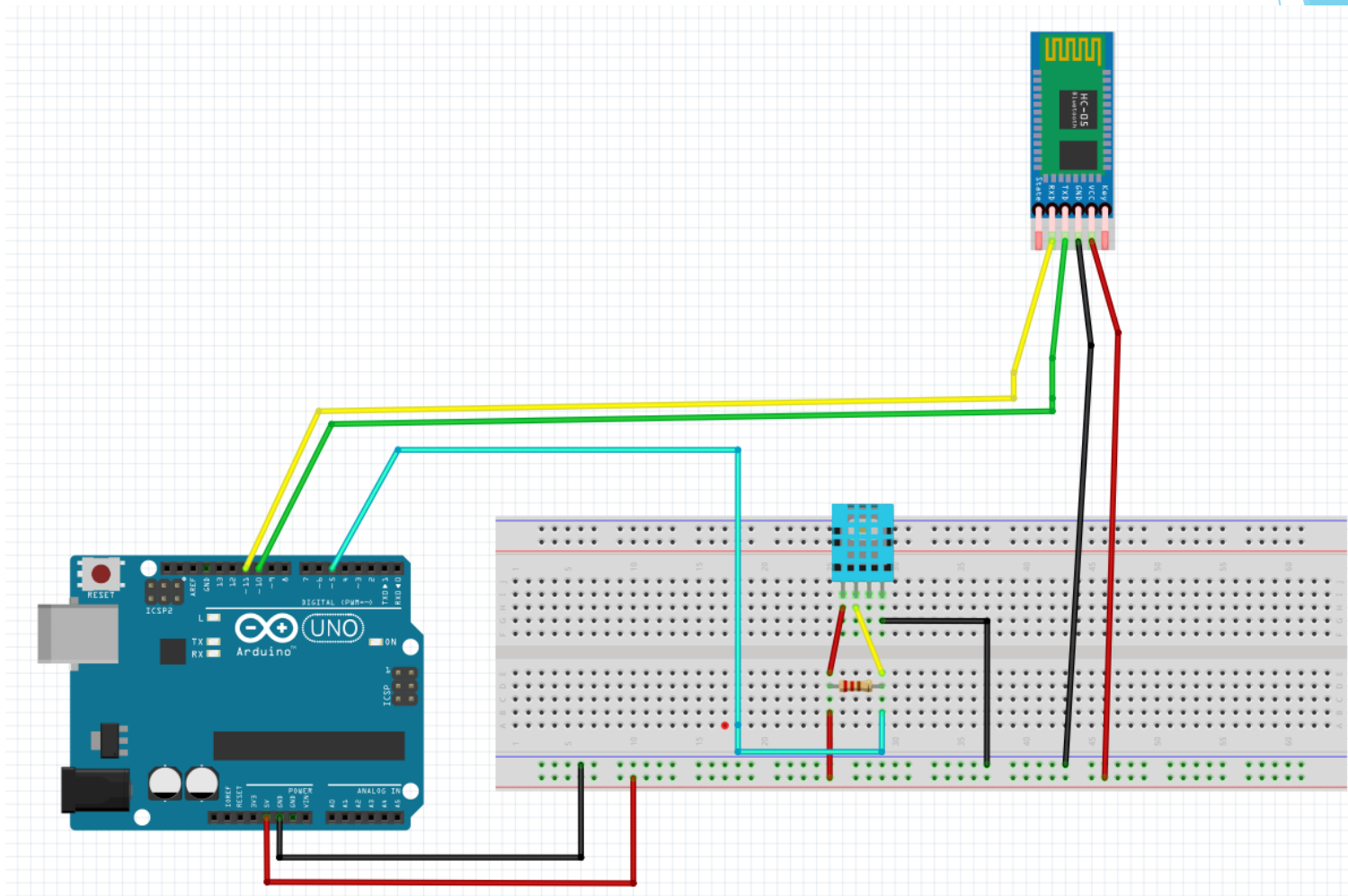
- Iniciar Rx**: Set to 10, Tx to 11, and Baud rate to 9600.
- Nombre**: Set to "HC-05", ID to "1234", and Mode to Standard.

Bucle

- Ejecutar cada**: Set to 1000 ms.
- Establecer Temperatura**: Set to "Temperatura °C (NTC)" sensor on Pin A0.
- Enviar**: Send a message "Temperatura:" followed by the temperature value, with a line break checked.

Práctica 20: Comunicación bluetooth

- ❑ Código de programa para la medida de temperatura y envío a través de bluetooth con dht11



Práctica 20: Comunicación bluetooth

- ❑ Código de programa para la medida de temperatura y envío a través de bluetooth

Práctica 20: Comunicación bluetooth

- ❑ Código de programa para el encendido de led a través del bluetooth. Se encenderán con el 1 y el 2. También lee la señal de un potenciómetro.

```
graph TD
    subgraph Inicializar
        B1[Bluetooth Iniciar Rx 2 Tx 3 9600]
        B2[Bluetooth Nombre "HC-05" "1234" Standard]
    end
    subgraph Bucle
        E[Ejecutar cada 2000 ms]
        P[Establecer Lectura A0 = Potenciómetro Pin A0 %]
        Env[Enviar + - crear texto con "Potenciómetro: " Salto de línea Lectura A0]
        S1[si ¿Datos recibidos?]
        subgraph hacer
            R[Recibir byte]
            S2[si Orden = 49]
            subgraph hacer
                E1[Escribir digital Pin 13 ON]
            end
            S3[si Orden = 50]
            subgraph hacer
                E2[Escribir digital Pin 13 OFF]
            end
        end
    end
```

The image shows a Scratch-style code editor with the following blocks:

- Inicializar**
 - Bluetooth Iniciar Rx 2 Tx 3 9600
 - Bluetooth Nombre "HC-05" "1234" Standard
- Bucle**
 - Ejecutar cada 2000 ms
 - Establecer Lectura A0 = Potenciómetro Pin A0 %
 - Enviar + - crear texto con "Potenciómetro: " Salto de línea Lectura A0
 - si ¿Datos recibidos?
 - hacer
 - Recibir byte
 - si Orden = 49
 - hacer Escribir digital Pin 13 ON
 - si Orden = 50
 - hacer Escribir digital Pin 13 OFF

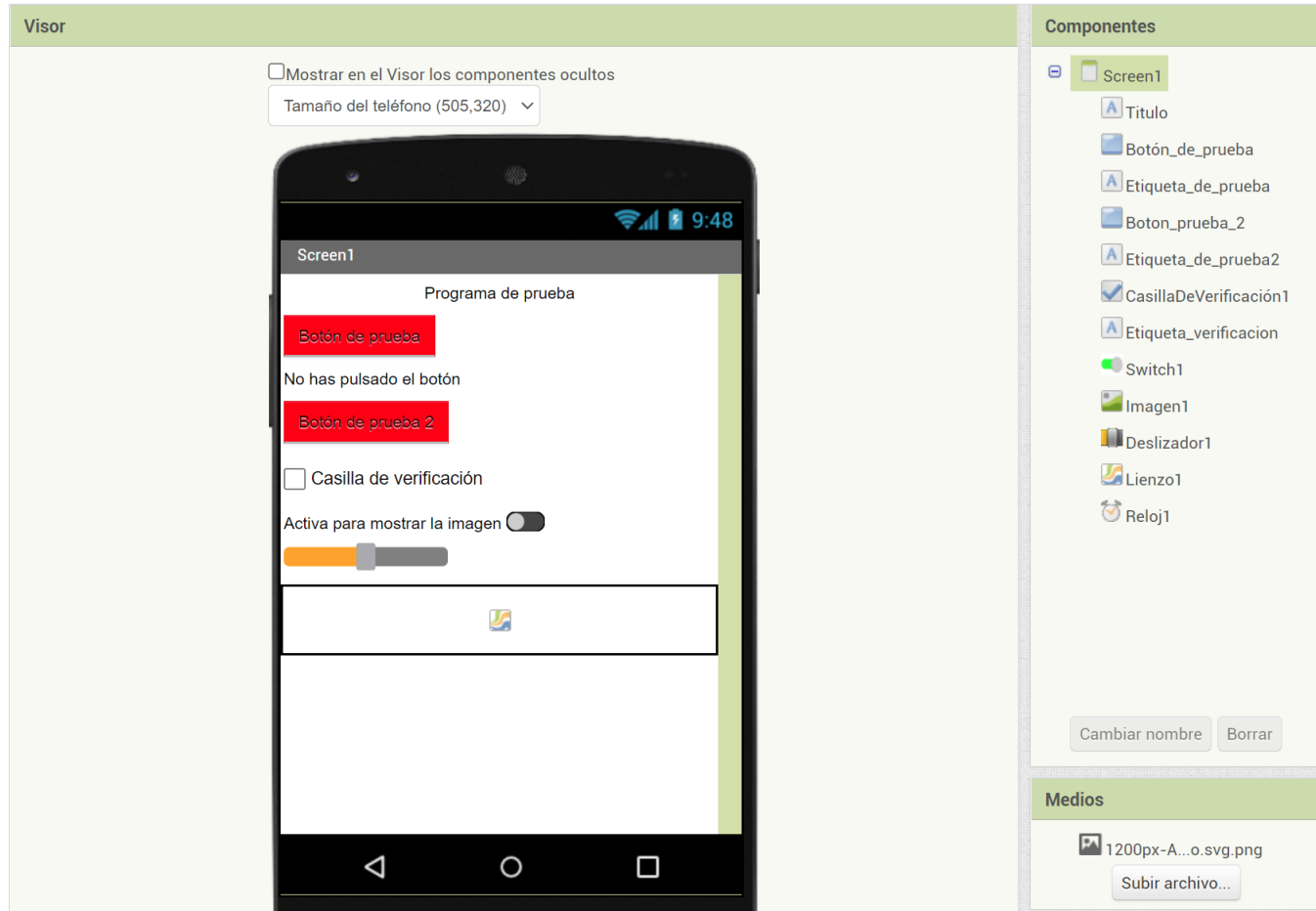
Práctica 20: Comunicación bluetooth

❑ Tabla Código ASCII

Caracteres ASCII de control			Caracteres ASCII imprimibles				ASCII extendido (Página de código 437)									
00	NULL	(carácter nulo)	32	espacio	64	@	96	`	128	Ç	160	á	192	Ł	224	Ó
01	SOH	(inicio encabezado)	33	!	65	A	97	a	129	ü	161	í	193	ł	225	õ
02	STX	(inicio texto)	34	"	66	B	98	b	130	é	162	ó	194	Ł	226	ö
03	ETX	(fin de texto)	35	#	67	C	99	c	131	â	163	ú	195	ł	227	õ
04	EOT	(fin transmisión)	36	\$	68	D	100	d	132	ä	164	ñ	196	—	228	ö
05	ENQ	(consulta)	37	%	69	E	101	e	133	à	165	Ñ	197	+	229	Õ
06	ACK	(reconocimiento)	38	&	70	F	102	f	134	ã	166	ª	198	ä	230	µ
07	BEL	(timbre)	39	'	71	G	103	g	135	ç	167	º	199	Ä	231	þ
08	BS	(retroceso)	40	(72	H	104	h	136	ê	168	¿	200	Ł	232	Ɔ
09	HT	(tab horizontal)	41)	73	I	105	i	137	ë	169	®	201	ł	233	Ù
10	LF	(nueva línea)	42	*	74	J	106	j	138	è	170	¬	202	Ł	234	Û
11	VT	(tab vertical)	43	+	75	K	107	k	139	ï	171	½	203	ł	235	Ü
12	FF	(nueva página)	44	,	76	L	108	l	140	î	172	¼	204	ł	236	ý
13	CR	(retorno de carro)	45	-	77	M	109	m	141	ï	173	ı	205	=	237	ÿ
14	SO	(desplaza afuera)	46	.	78	N	110	n	142	Ä	174	«	206	ł	238	—
15	SI	(desplaza adentro)	47	/	79	O	111	o	143	Å	175	»	207	ı	239	'
16	DLE	(esc.vínculo datos)	48	0	80	P	112	p	144	É	176	⋯	208	ò	240	≡
17	DC1	(control disp. 1)	49	1	81	Q	113	q	145	æ	177	⋮	209	Đ	241	±
18	DC2	(control disp. 2)	50	2	82	R	114	r	146	Æ	178	⋮	210	È	242	≡
19	DC3	(control disp. 3)	51	3	83	S	115	s	147	ò	179	⋮	211	Ë	243	¾
20	DC4	(control disp. 4)	52	4	84	T	116	t	148	ö	180	⋮	212	È	244	¶
21	NAK	(conf. negativa)	53	5	85	U	117	u	149	ò	181	À	213	ı	245	§
22	SYN	(inactividad sinc)	54	6	86	V	118	v	150	ù	182	Â	214	ı	246	÷
23	ETB	(fin bloque trans)	55	7	87	W	119	w	151	ù	183	À	215	ı	247	˙
24	CAN	(cancelar)	56	8	88	X	120	x	152	ÿ	184	©	216	ı	248	˚
25	EM	(fin del medio)	57	9	89	Y	121	y	153	Ö	185	⋮	217	ı	249	¨
26	SUB	(sustitución)	58	:	90	Z	122	z	154	Ü	186	⋮	218	ı	250	·
27	ESC	(escape)	59	;	91	[123	{	155	ø	187	⋮	219	ı	251	˘
28	FS	(sep. archivos)	60	<	92	\	124		156	£	188	⋮	220	ı	252	˚
29	GS	(sep. grupos)	61	=	93]	125	}	157	Ø	189	¢	221	ı	253	˚
30	RS	(sep. registros)	62	>	94	^	126	~	158	x	190	¥	222	ı	254	■
31	US	(sep. unidades)	63	?	95	_			159	f	191	ł	223	ı	255	nbsp

Práctica 22: App inventor - Aspectos básicos

- ❑ Esta primera práctica servirá para familiarizarnos con los elementos básicos que podemos utilizar en nuestras aplicaciones.



Práctica 21: APP INVENTOR - Aspectos básicos

❑ Código App Inventor

```
cuando Botón_de_prueba . Clic
ejecutar
  poner Botón_de_prueba . ColorDeFondo como 
  poner Etiqueta_de_prueba . Texto como "Has pulsado el botón"
```

```
cuando Boton_prueba_2 . Presionar
ejecutar
  poner Boton_prueba_2 . ColorDeFondo como 
  poner Etiqueta_de_prueba2 . Texto como "Estás pulsando el botón"
```

```
cuando Boton_prueba_2 . Soltar
ejecutar
  poner Boton_prueba_2 . ColorDeFondo como 
  poner Etiqueta_de_prueba2 . Texto como "No estás pulsando el botón"
```

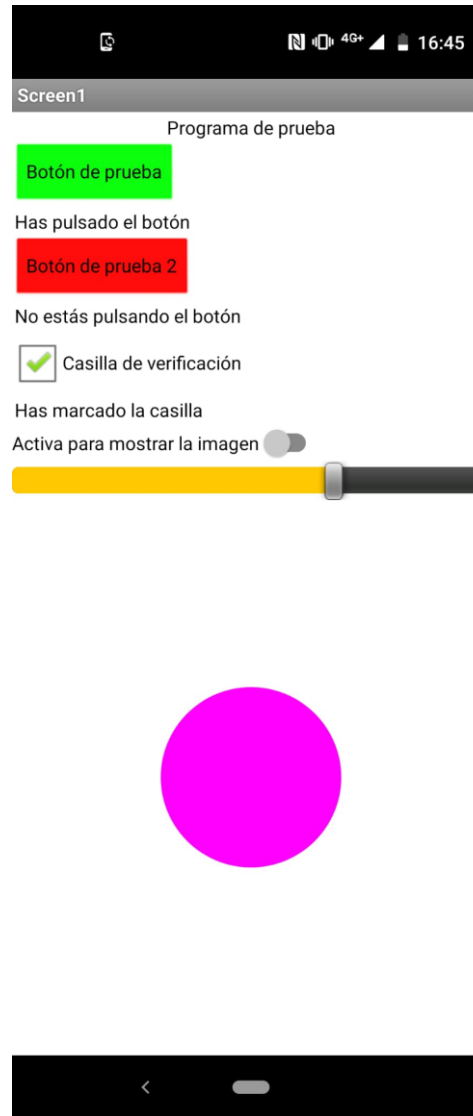
```
cuando CasillaDeVerificación1 . Cambiado
ejecutar
  si
    CasillaDeVerificación1 . Verificado = cierto
  entonces
    poner Etiqueta_verificacion . Texto como "Has marcado la casilla"
  sino
    poner Etiqueta_verificacion . Texto como "No has marcado la casilla"
```

```
cuando Reloj1 . Temporizador
ejecutar
  llamar Lienzo1 . Limpiar
  poner Lienzo1 . ColorDePintura como 
  llamar Lienzo1 . DibujarCirculo
    centerX Lienzo1 . Ancho / 2
    centerY Lienzo1 . Alto / 2
    radius Deslizador1 . PosiciónDelPulgar
    fill cierto
```

```
cuando Switch1 . Cambiado
ejecutar
  si
    Switch1 . On = cierto
  entonces
    poner Imagen1 . Visible como cierto
  sino
    poner Imagen1 . Visible como falso
```

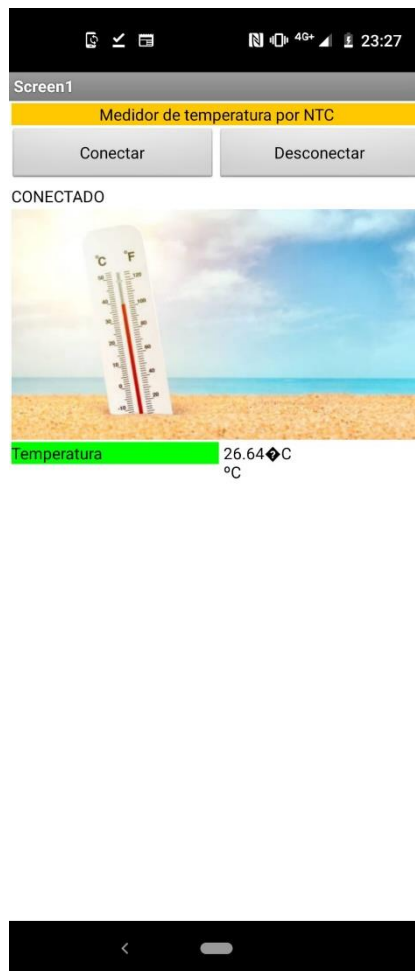
Práctica 21: APP INVENTOR - Aspectos básicos

❑ Resultado final



Práctica 23: App inventor - Lectura de temperatura y humedad

- ❑ Con esta práctica vamos programar una aplicación que nos permita visionar en tiempo real la temperatura y la humedad que está leyendo un sensor conectado al Arduino..



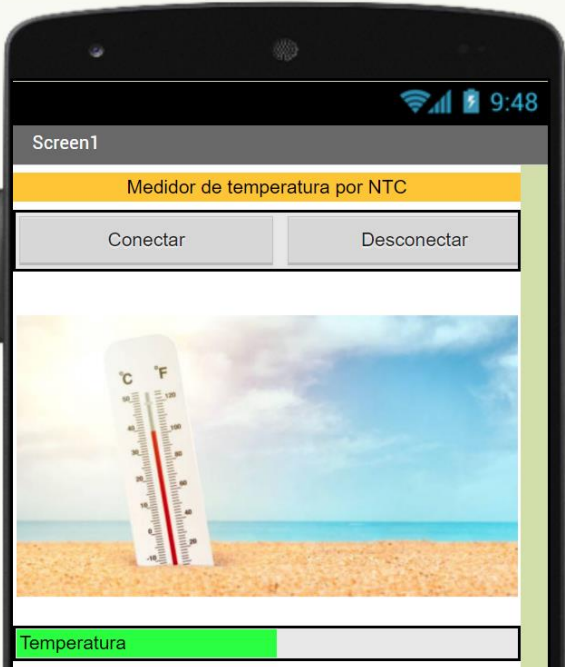
Práctica 23: App inventor - Lectura de temperatura y humedad

❑ Elementos de interfaz

Visor

Mostrar en el Visor los componentes ocultos

Tamaño del teléfono (505,320) ▾

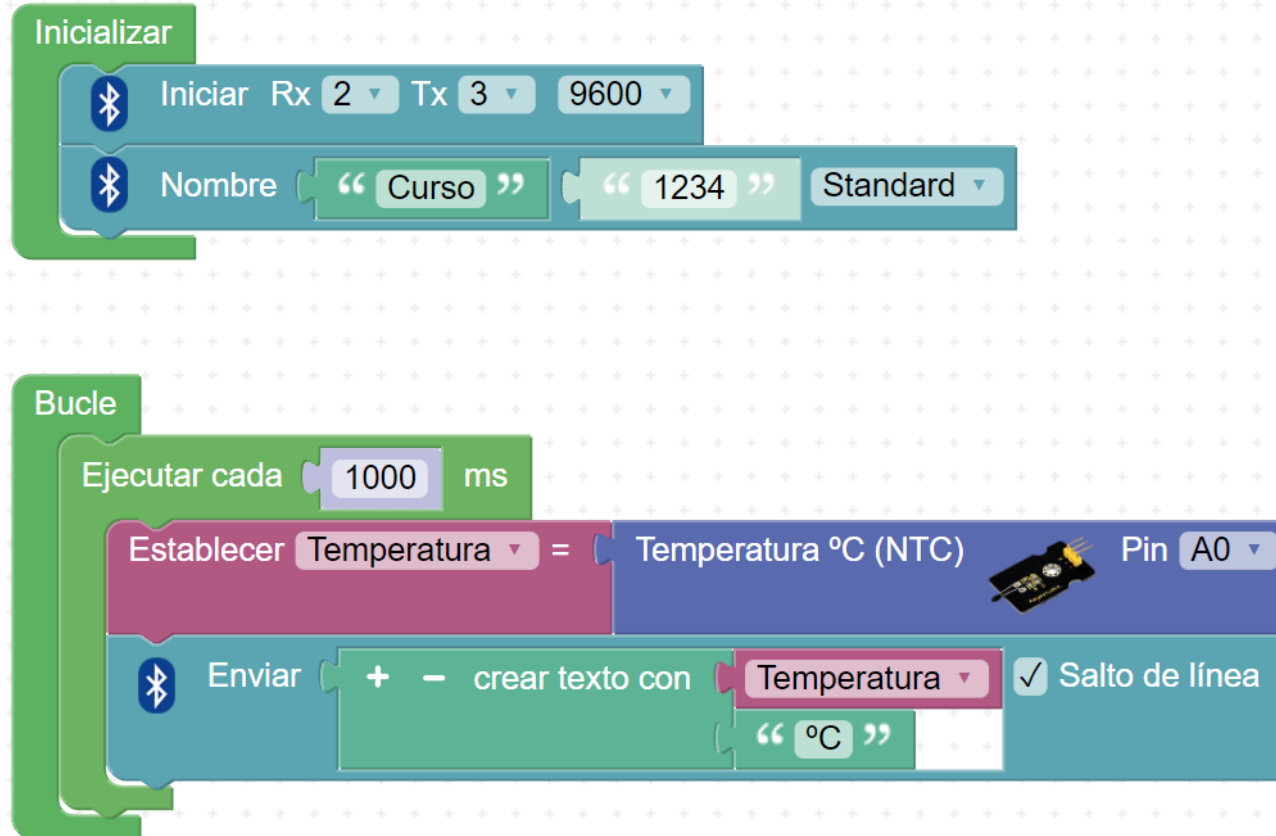


Componentes

- Screen1
 - TITULO
- DisposiciónHorizontal2
 - Selector_conexion
 - Botón1
- Info_conexion
- Imagen1
- DisposiciónHorizontal1
 - Etiqueta_color
 - Etiqueta_temperatura
- ClienteBluetooth1
- ServidorBluetooth1
- Reloj1

Práctica 23: App inventor - Lectura de temperatura y humedad

❑ Código Arduino Blocks



The image shows a screenshot of the Arduino Blocks programming environment. The code is organized into two main sections: 'Inicializar' (Initialize) and 'Bucle' (Loop).

Inicializar

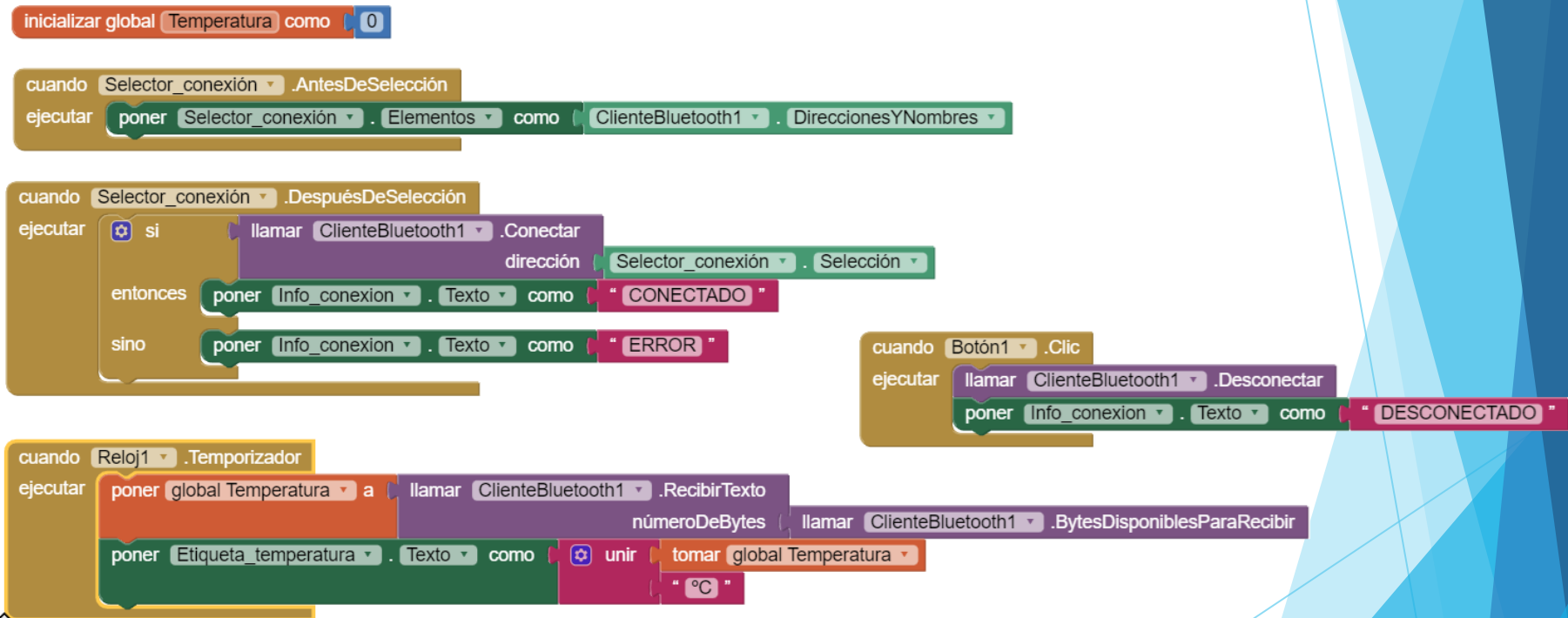
- Iniciar Rx**: 2, Tx 3, 9600
- Nombre**: "Curso", "1234", Standard

Bucle

- Ejecutar cada**: 1000 ms
- Establecer Temperatura**: = Temperatura °C (NTC) Pin A0
- Enviar**: + - crear texto con Temperatura Salto de línea " °C "

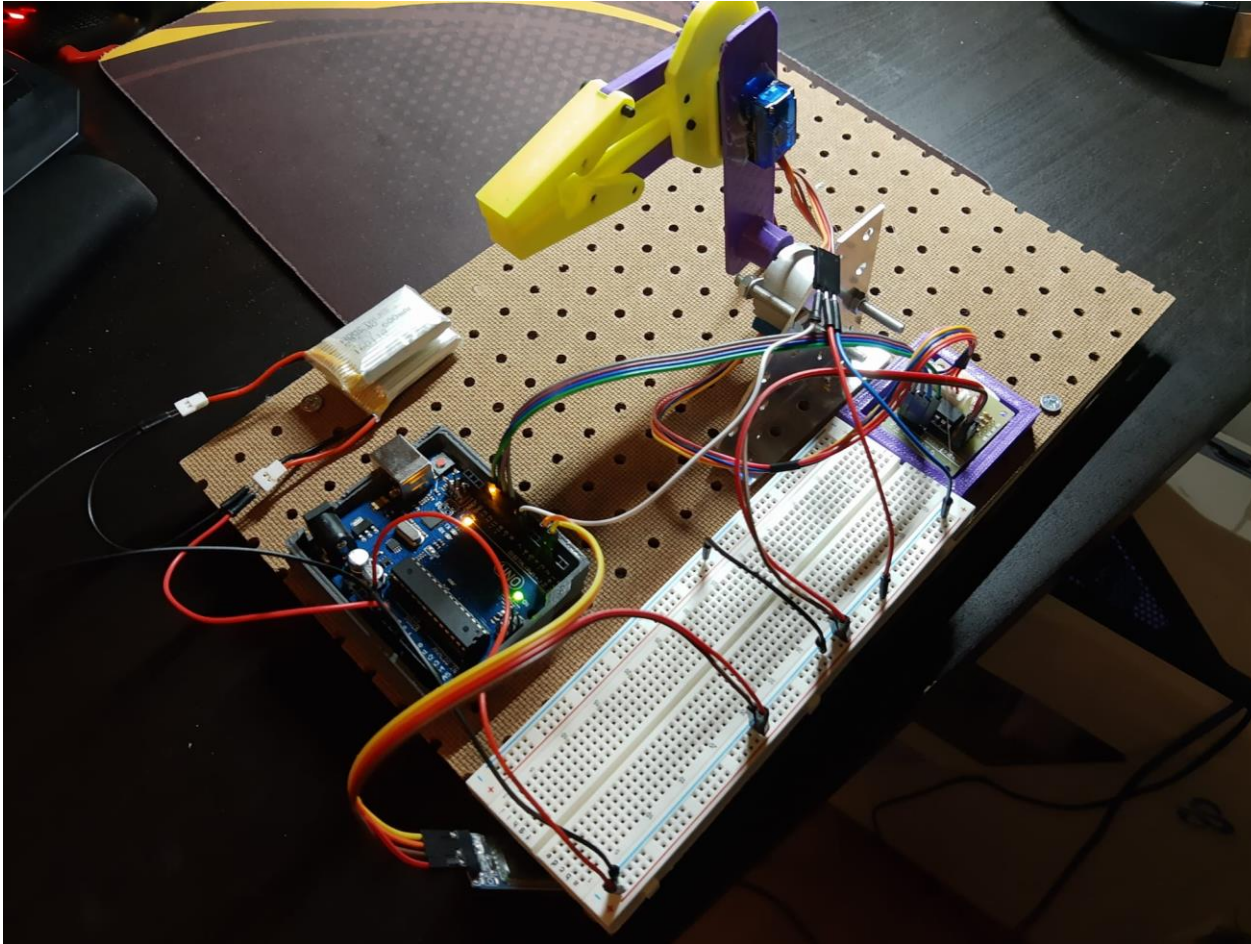
Práctica 23: App inventor - Lectura de temperatura y humedad

❑ Código App inventor



Práctica 24: App inventor - Control de brazo robótico

- ❑ Ahora vamos a diseñar una aplicación para controlar un brazo robótico compuesto por un motor paso a paso y un servo.



Práctica 24: App inventor - Control de brazo robótico

❑ Esquema de conexiones.

Mostrar en el Visor los componentes ocultos

Tamaño del teléfono (505,320) ▾

Screen1

CONTROL PINZA

conectar Desconectar

ABRIR PINZA CERRAR PINZA

Screen1

- Titulo
- DisposiciónHorizontal1
 - SelectorDeLista1
 - Botón1
- Etiqueta1
- Imagen1
- DisposiciónHorizontal2
 - Boton_Giro_horario
 - Boton_Giro_antihorari
- DisposiciónHorizontal3
 - Apertura_pinza
 - Cierre_pienza
- ClienteBluetooth1
- ServidorBluetooth1

Cambiar nombre Borrar

Medios

- aBRIR_PINZA.png
- cERRAR_PINZA.png
- giro_antihor.png

Práctica 24: App inventor - Control de brazo robótico

❑ Código en Arduino Blocks

```
cuando SelectorDeLista1 .AntesDeSelección
ejecutar poner SelectorDeLista1 . Elementos como ClienteBluetooth1 . DireccionesYNombres
```

```
cuando SelectorDeLista1 .DespuésDeSelección
ejecutar si
  llamar ClienteBluetooth1 .Conectar
  dirección SelectorDeLista1 . Selección
entonces poner Etiqueta1 . Texto como " CONECTADO "
sino poner Etiqueta1 . Texto como " ERROR "
```

```
cuando Botón1 .Clic
ejecutar llamar ClienteBluetooth1 .Desconectar
poner Etiqueta1 . Texto como " DESCONECTADO "
```

```
cuando Boton_Giro_horario .Clic
ejecutar llamar ClienteBluetooth1 .EnviarNúmero1Byte
número 49
```

```
cuando Boton_Giro_antihorario .Clic
ejecutar llamar ClienteBluetooth1 .EnviarNúmero1Byte
número 50
```

```
cuando Apertura_pinza .Clic
ejecutar llamar ClienteBluetooth1 .EnviarNúmero1Byte
número 51
```

```
cuando Cierre_pinza .Clic
ejecutar llamar ClienteBluetooth1 .EnviarNúmero1Byte
número 52
```

Práctica 24: App inventor - Control de brazo robótico

❑ Código en App Inventor

The image shows the code for an App Inventor application, divided into two main sections: 'Inicializar' (Initialize) and 'Bucle' (Loop).

Inicializar (Initialize):

- Bluetooth: Iniciar Rx 2, Tx 3, 9600.
- Bluetooth: Nombre "HC-05", "1234", Standard.
- Motor: Paso a paso # 1, Pasos/vuelta 512, Pin-1 8, Pin-2 9, Pin-3 10, Pin-4 11.
- Motor: Paso a paso # 1, Velocidad (rpm) 30.

Bucle (Loop):

- Condición: + si ¿Datos recibidos?
- Acción: hacer Establecer Código_señal = Recibir byte.
- Condición: + si Código_señal = 49.
- Acción: hacer Paso a paso # 1, Pasos 32.
- Condición: + si Código_señal = 50.
- Acción: hacer Paso a paso # 1, Pasos -32.
- Condición: + si Código_señal = 51.
- Acción: hacer Servo Pin 7, Grados Ángulo 0°, Retardo (ms) 0.
- Condición: + si Código_señal = 52.
- Acción: hacer Servo Pin 7, Grados Ángulo 315°, Retardo (ms) 0.

Práctica 24: App inventor - Control de brazo robótico

☐ Resultado final.

