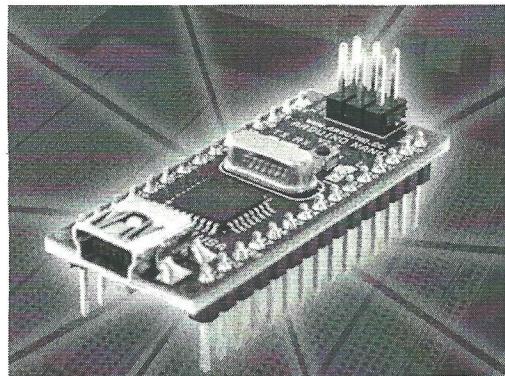


ARDUINO NANO

CURSO DE MICROCONTROLADORES

PLATAFORMA ARDUINO

ARDUINO NANO V3.0



ARDUINO

Arduino es una plataforma de prototipos electrónica de código abierto (open-source) basada en hardware y software flexibles y fáciles de usar.

Arduino puede monitorear el entorno mediante la recepción de entradas desde una variedad de sensores y puede afectar a su alrededor mediante el control de luces, motores y otros actuadores. El microcontrolador de la placa se programa usando el ***Arduino Programming Language*** (similar al lenguaje C). Los proyectos de Arduino pueden ser autónomos o se pueden comunicar con software en ejecución en un ordenador (por ejemplo con Flash, Processing, MaxMSP, etc.).

Las placas se pueden ensamblar a mano o encargarlas preensambladas; el software se descargan gratuitamente. Los diseños de referencia del hardware (archivos CAD) están disponibles bajo licencia open-source, por lo que eres libre de adaptarlas a tus necesidades.

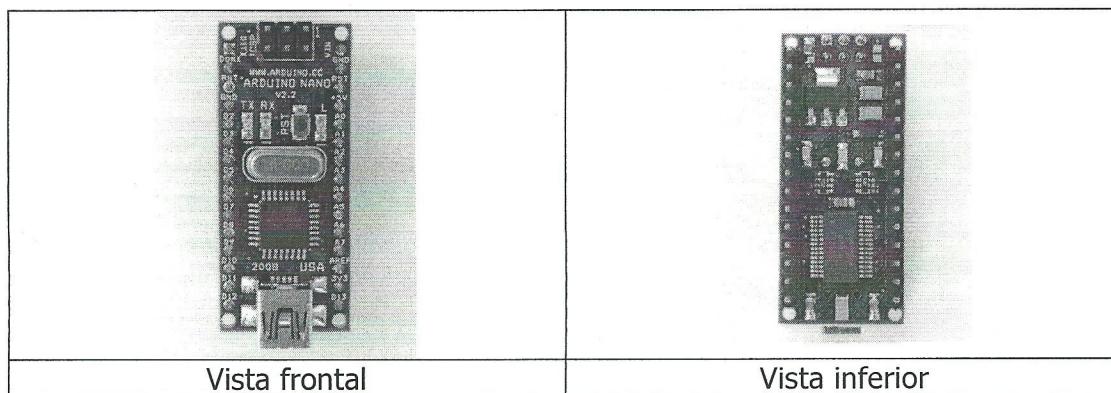
Hay muchos otros microcontroladores y plataformas microcontroladoras disponibles para computación física. Parallax Basic Stamp, Netmedia's BX-24, Phidgets, MIT's Handyboard, y muchas otras ofertas de funcionalidad similar. Todas estas herramientas toman los desordenados detalles de la programación de microcontrolador y la encierran en un paquete fácil de usar. Arduino también simplifica el proceso de trabajo con microcontroladores, pero ofrece algunas ventajas para profesores, estudiantes y aficionados interesados sobre otros sistemas:

ARDUINO NANO

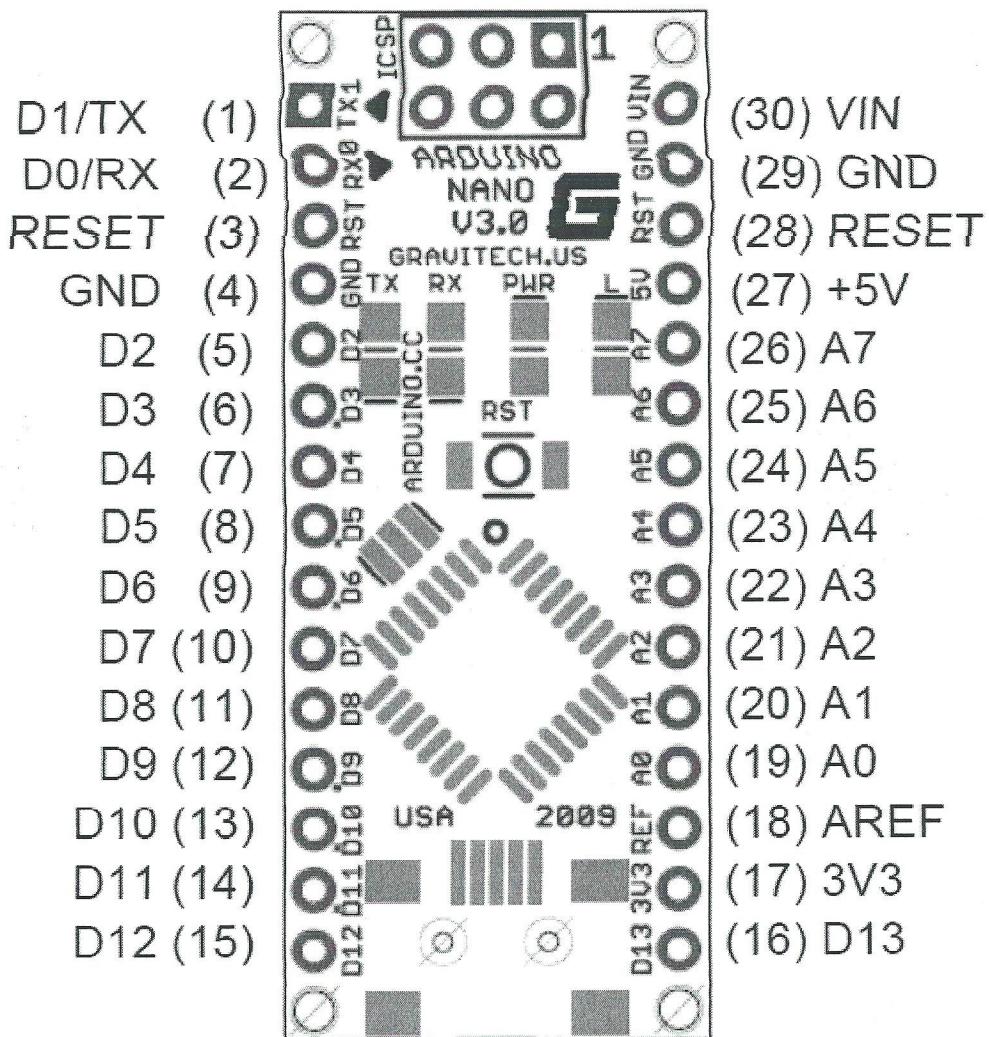
- Barato: Las placas Arduino son relativamente baratas comparadas con otras plataformas microcontroladoras.
- Multiplataforma: El software de Arduino se ejecuta en sistemas operativos Windows, Macintosh OSX y GNU/Linux. La mayoría de los sistemas microcontroladores están limitados a Windows.
- Entorno de programación simple y claro: El entorno de programación de Arduino es fácil de usar para principiantes, pero suficientemente flexible para que usuarios avanzados puedan aprovecharlo también. Para profesores, está convenientemente basado en el entorno de programación Processing, de manera que los estudiantes aprendiendo a programar en ese entorno, estarán familiarizados con el aspecto y la imagen de Arduino.
- Código abierto y software extensible: El software Arduino está publicado como herramientas de código abierto, disponible para extensión por programadores experimentados. El lenguaje puede ser expandido mediante librerías C++, y la gente que quiera entender los detalles técnicos pueden hacer el salto desde Arduino a la programación en lenguaje AVR C en el cual está basado. De forma similar, puedes añadir código AVR-C directamente en tus programas Arduino si quieres.
- Código abierto y hardware extensible: El Arduino Nano V3.0 está basado en microcontroladores ATMEGA328 de Atmel. Los planos para los módulos están publicados bajo licencia Creative Commons, por lo que diseñadores experimentados de circuitos pueden hacer su propia versión del módulo, extendiéndolo y mejorándolo. Incluso usuarios relativamente inexpertos pueden construir la versión de la placa del módulo para entender como funciona y ahorrar dinero.

Hardware

ARDUINO NANO

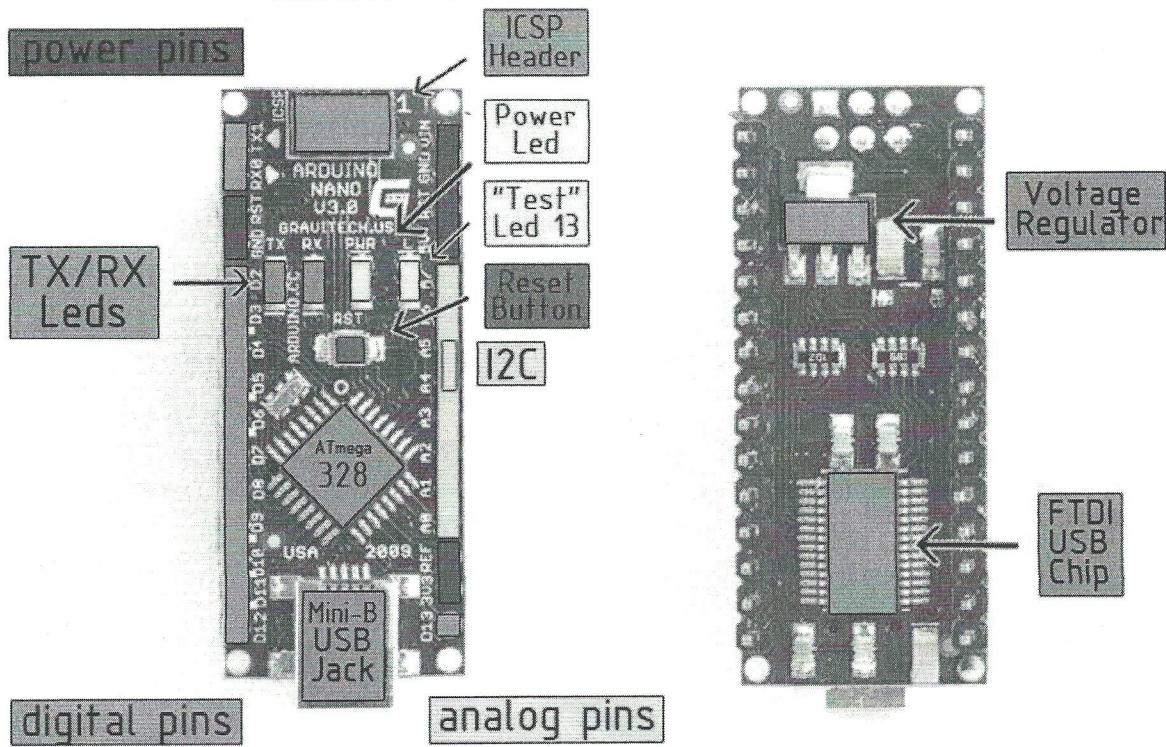


DIRECCIONAMIENTO DE LOS PINES



No de Pin	Asignación	Tipo	Descripción
1-2,5-16	D0-D13	I/O	Entrada/salida digital Puerto 0 a 13
3, 28	Reset	Input	Botón de reset
4, 29	GND	PWR	Conexión a tierra
17	3.3V	Output	Salida de 3.3V
18	AREF	Input	Voltaje de referencia ADC
19-26	A0-A7	Input	Canales de entradas analógicas
27	5V	Output or Input	Entrada o salida de voltaje 5V
30	Vin	PWR	Voltaje de alimentación

ESPECIFICACIONES TÉCNICAS



Microcontrolador	Atmel ATmega328
Voltaje de operación	5V
Voltaje de entrada	7V – 12V
Pines de E/S digitales	14 (6 pueden ser salidas PWM)
Pines de entrada analógica	8
Consumo de corriente por pin	40 mA
Capacidad de memoria para programa	32KB (2k usados por el bootloader)
SRAM	2KB
EEPROM	1KB
Velocidad de reloj	16 MHz
Dimensiones	0.7 in X 1.7 in

PROGRAMACIÓN

Estructura

La estructura básica del lenguaje de programación Arduino es bastante simple y se organiza en al menos dos partes o funciones que encierran bloques de declaraciones.

```
void setup()
{
statements;
}
void loop()
{
statements;
}
```

Ambas funciones son requeridas para que el programa funcione.

Setup. Es la primera función a ejecutar en el programa, es ejecutada una vez y es usada para asignar

pinMode o inicializar las comunicaciones serie.

```
void setup()
{
pinMode(pin, OUTPUT); //ajusta 'pin' como salida
}
loop()
```

La función loop se ejecuta a continuación e incluye el código que se ejecuta continuamente - leyendo entradas, activando salidas, etc. Esta función es el núcleo de todos los programas Arduino y hace la mayor parte del trabajo.

```
void loop()
{
digitalWrite(pin, HIGH); //Activa 'pin'
delay(1000); //espera un segundo
digitalWrite(pin, LOW); //Desactiva 'pin'
delay(1000); //espera un segundo
}
```

Funciones

Una función es un bloque de código que tiene un nombre y un grupo de declaraciones que se ejecutan cuando se llama a la función. Podemos hacer uso de funciones integradas como void setup() y void loop() o escribir nuevas.

Las funciones se escriben para ejecutar tareas repetitivas y reducir el desorden en un programa.

En primer lugar se declara el tipo de la función, que será el valor retornado por la función (int, void...). A continuación del tipo, se declara el nombre de la función y entre paréntesis, los parámetros que se pasan a la función.

```
type functionName(parameters)
{
statements;
}
```

La siguiente función int delayVal(), asigna un valor de retardo en un programa por lectura del valor de un potenciómetro.

```
int delayVal()
{
int v; //crea una variable temporal 'v'
v = analogRead(pot); //lee el valor del potenciómetro
v /= 4; //convierte 0-1023 a 0-255
return v; //devuelve el valor final de v
}
```

LLaves {}

Las llaves definen el comienzo y el final de bloques de función y bloques de declaraciones como void loop() y sentencias for e if. Las llaves deben estar balanceadas (a una llave de apertura { debe seguirle una llave de cierre }). Las llaves no balanceadas provocan errores de compilación.

```
void loop()
{
statements;
}
```

El entorno Arduino incluye una práctica característica para chequear el balance de llaves. Sólo selecciona una llave y su compañera lógica aparecerá resaltada.

Punto y coma ;

Un punto y coma debe usarse al final de cada declaración y separa los elementos del programa.

También se usa para separar los elementos en un bucle for. int x = 13; //declara la variable 'x' como el entero 13.

Nota: Olvidar un punto y coma al final de una declaración producirá un error de compilación.

Bloques de comentarios /*...*/

Los bloques de comentarios, o comentarios multilínea, son áreas de texto ignoradas por el programa y se usan para grandes descripciones de código o comentarios que ayudan a otras personas a entender partes del programa. Empiezan con /* y terminan con */ y pueden abarcar múltiples líneas. /* Este es un bloque de comentario encerrado no olvides cerrar el comentario tienen que estar balanceados! */. Como los comentarios son ignorados por el programa y no ocupan espacio en memoria.

Comentarios de línea //

Comentarios de una línea empiezan con // y terminan con la siguiente línea de código. Como el bloque de comentarios, son ignorados por el programa y no toman espacio en memoria. // Este es un comentario de una línea. Comentarios de una línea se usan a menudo después de declaraciones válidas para proporcionar más información sobre qué lleva la declaración o proporcionar un recordatorio en el futuro.

Variables

Una variable es una forma de llamar y almacenar un valor numérico para usarse después por el programa. Como su nombre indica, las variables son números que pueden cambiarse continuamente al contrario que las constantes, cuyo valor nunca cambia. Una variable necesita ser declarada y, opcionalmente, asignada al valor que necesita para ser almacenada.

```
int inputVariable = 0; //declara una variable y asigna el valor a 0
inputVariable = analogRead(2); //ajusta la variable al valor del pin
                           //analógico 2
```

Una vez que una variable ha sido asignada, o reasignada, puedes testear su valor para ver si cumple ciertas condiciones, o puedes usarlo directamente.

```
if(inputVariable < 100) //comprueba si la variable es menor que 100
{
  inputVariable = 100; //si es cierto asigna el valor 100
}
delay(inputVariable); //usa la variable como retardo
```

Declaración de variable

Todas las variables tienen que ser declaradas antes de que puedan ser usadas. Declarar una variable significa definir su tipo de valor, como int, long, float, etc., definir un nombre específico y opcionalmente, asignar un valor inicial. Esto sólo necesita hacerse una vez en un programa pero el valor puede cambiarse en cualquier momento usando aritmética y varias asignaciones. int inputVariable = 0; Una variable puede ser declarada en un número de posiciones en todo el programa y donde esta definición tiene lugar determina que partes del programa pueden usar la variable.

Ámbito de la variable

Una variable puede ser declarada al comienzo del programa antes del void setup(), localmente dentro de funciones, y algunas veces en un bloque de declaración, por ejemplo bucles for. Donde la variable es declarada determina el ámbito de la variable, o la habilidad de ciertas partes de un programa de hacer uso de la variable. Una variable global es una que puede ser vista y usada por cualquier función y declaración en un programa. Esta variable se declara al comienzo del programa, antes de la función setup().

Una variable local es una que se define dentro de una función o como parte de un bucle for. Sólo es visible y sólo puede ser usada dentro de la función en la cual fue declarada. Además, es posible tener dos o más variables del mismo nombre en diferentes partes del programa que contienen diferentes valores. int value; //'value' es visible por cualquier función.

```
void setup()
{
//no se necesita setup
}
void loop()
{
for(int i=0; i<20;) // 'i' es sólo visible dentro del bucle for
{
i++;
33
}
float f; // 'f' es sólo visible dentro de loop
}
```

ARDUINO NANO

EJEMPLOS

Ejem. 1. Control del encendido y apagado de un led con temporizadores.
El led del puerto D13 se prenderá y 0.5 segundos después se apagará, se prenderá nuevamente de manera automática después de 1 segundo.

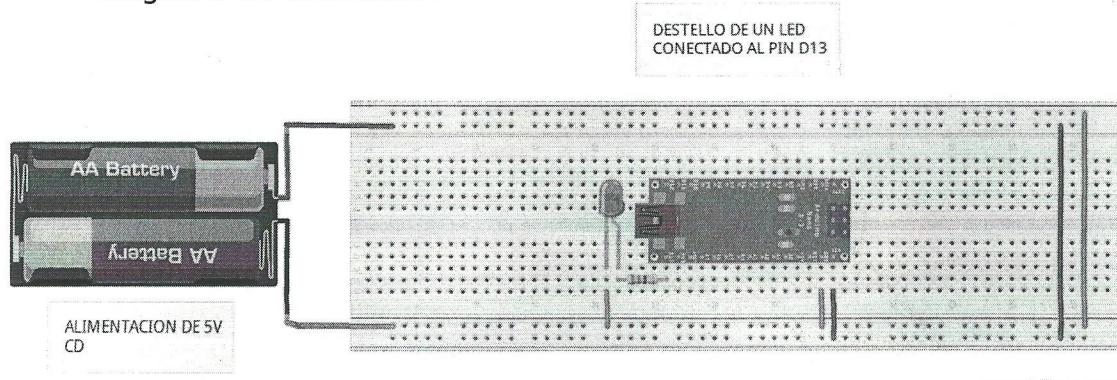
Programa:

```
int led = 13;

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

void loop() {
  digitalWrite(led, HIGH);
  delay(1000);
  digitalWrite(led, LOW);
  delay(1000);
}
```

Diagrama de conexiones:



Modificaciones:

- Debe prender y apagar el led en el puerto D10.
- Deben prender y apagar de manera secuencial los led conectados a los puertos D6 y D12.

ARDUINO NANO

Ejem. 2. Control de encendido y apagado de un led conectado al puerto D11, con un interruptor conectado al puerto D8.

Programa:

```
int boton = 8;
int led = 11;

int Valboton= 0;

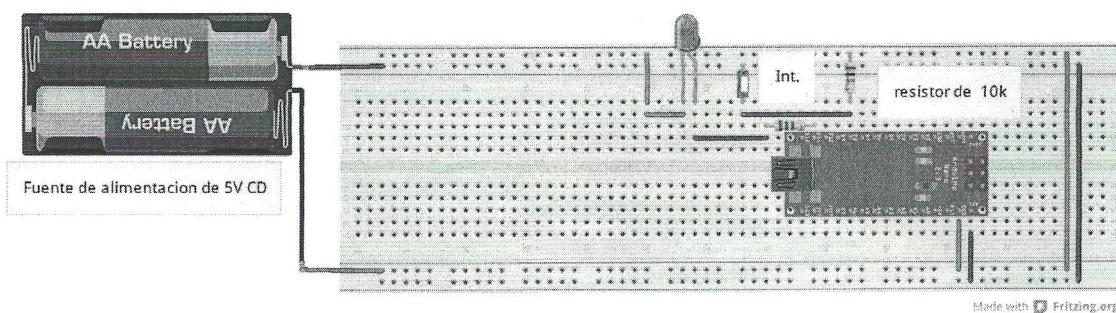
void setup() {
    pinMode(led, OUTPUT);
    pinMode(boton, INPUT);
}

void loop() {
    Valboton = digitalRead(boton);

    if (Valboton == HIGH) {
        digitalWrite(led, HIGH);
    }
    else {

        digitalWrite(led, LOW);
    }
}
```

Diagrama:



ARDUINO NANO

Ejem. 3. Control de encendido y apagado de dos led conectado a los puertos D12 y D13, con sus respectivos interruptores conectados a los puertos D2 y D3.

Programa:

```

int boton1 = 2;
int boton2 = 3;
int led1 = 12;
int led2 = 13;
int Var1= 0;
int Var2= 0;

void setup() {
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  pinMode(boton1, INPUT);
  pinMode(boton2, INPUT);
}

void loop() {
  Var1 = digitalRead(boton1);

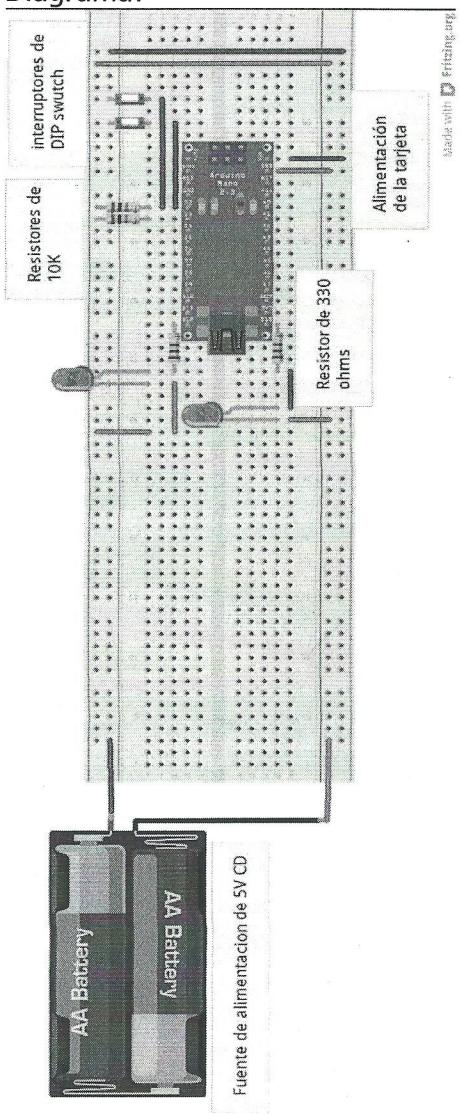
  if (Var1 == HIGH) {
    digitalWrite(led1, HIGH);
  }
  else {
    digitalWrite(led1, LOW);

    Var2 = digitalRead(boton2);

    if (Var2 == HIGH) {
      digitalWrite(led2, HIGH);
    }
    else {
      digitalWrite(led2, LOW);
    }
  }
}

```

Diagrama:



Modificaciones:

- Integrar dos leds más (D10 y D11) con sus interruptores (D4 y D5). Para realizar una secuencia una secuencia similar a la anterior.

Ejem. 4. Un led destella (D12) una cantidad definida de veces (10).

<p>Programa:</p> <pre>byte i; void setup() { pinMode(12, OUTPUT); } void loop() { for(i=0; i<10; i=i+1) { digitalWrite(12, HIGH); delay(250); digitalWrite(12, LOW); delay(500); } delay(5000); }</pre>	
---	--

Modificaciones:

```
byte i;
int var=0;

void setup() {
  pinMode(13, OUTPUT);
  pinMode(2, INPUT);

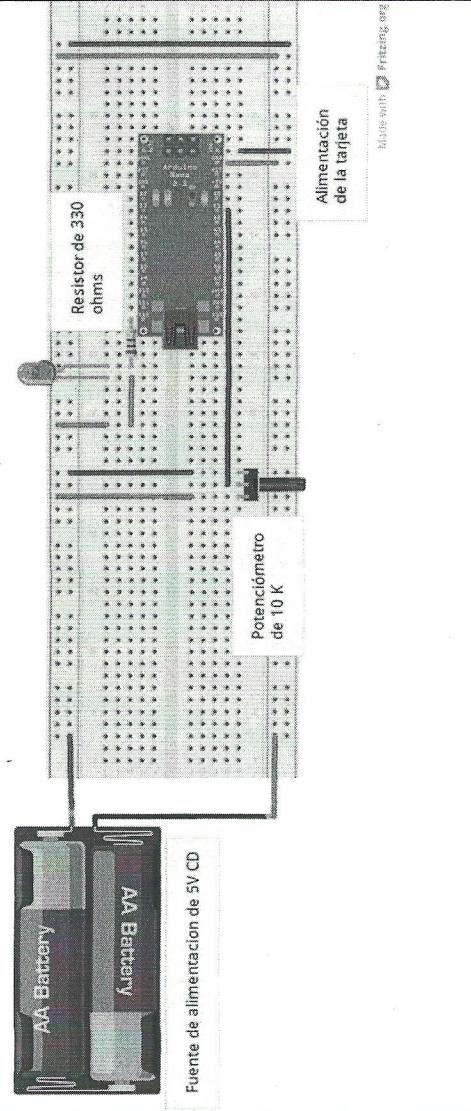
}

void loop() {
  var=digitalRead(2);
  if (var==HIGH){
    for( i=0; i<10; i=i+1) {
      digitalWrite(13, HIGH);
      delay(250);
      digitalWrite(13, LOW);
      delay(500);
    }
    delay(5000);
  }
}
```

- a) Considerando el siguiente programa integrar las conexiones.

Ejem. 5. Control del destello de un led (D12), con un potenciómetro (A1).

```
int led = 12;
int Pot = 1;
int Puls;
void setup(){
pinMode(led, OUTPUT);
}
void loop()
{
Puls = analogRead(Pot);
digitalWrite(led, HIGH);
delay(Puls);
digitalWrite(led, LOW);
delay(Puls);
}
```



Modificaciones:

- Integrar un led (D11) y dos interruptores (D3 y D4). Dependiendo del interruptor que se active le corresponderá el encendido de uno de los leds y su destello es controlado por el potenciómetro.