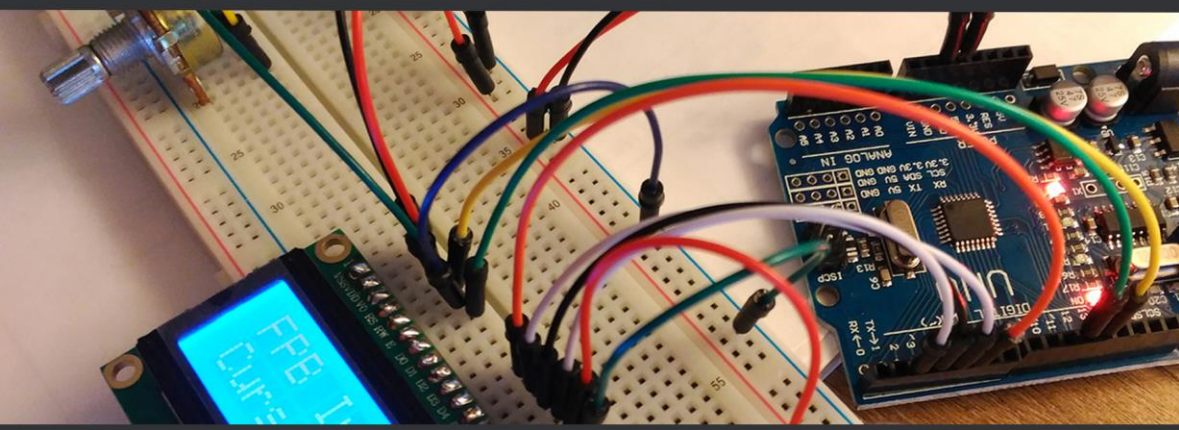
Arduino Blocks

PROGRAMACIÓN VISUAL CON BLOQUES PARA ARDUINO

2ª Edición

manual de referencia + 40 proyectos



lógica · control · matemáticas · texto · variables · listas · funciones
entrada/salida · temporización· puerto serie · bluetooth · sensores · actuadores
lcd · eeprom · servos · motores · keypad · reloj rtc · gps · tarjetas sd · IoT mqtt

JUAN JOSÉ LÓPEZ ALMENDROS

arduinoblocks.com

2ª EDICIÓN

JUAN JOSÉ LÓPEZ ALMENDROS

arduinoblocks.com

Ingeniero Técnico en Informática de Sistemas
Técnico Superior en Desarrollo de Productos Electrónicos
Profesor en Salesianos Juan XXIII – Alcoy

Colaborador y distribuidor oficial Keyestudio:

www.innovadidactic.com

ÍNDICE

1 INTRODUCCIÓN

La plataforma ArduinoBlocks nos permite iniciarnos en el mundo de la electrónica, robótica y la automatización de una forma sencilla e intuitiva.

La motivación de desarrollar la plataforma ArduinoBlocks y escribir este libro nace de mi trabajo como docente con alumnos de entre 12 y 18 años, sin previos conocimientos de programación, que quieren adentrarse en el mundo Arduino partiendo de una nociones básicas de electricidad y electrónica.

ArduinoBlocks es la herramienta perfecta para niños, jóvenes y adultos que quieren empezar a usar Arduino desde el primer momento sin necesidad de escribir ni una sola línea de código.

1.1 PLATAFORMA ARDUINO

Arduino es un proyecto de hardware libre que ideó una plataforma completa de hardware y software compuesta por placas de desarrollo que integran un microcontrolador y un entorno de desarrollo IDE. La idea surgió para facilitar el uso de la electrónica en proyectos multidisciplinarios.

El hardware consiste en una placa de circuito impreso con un microcontrolador, normalmente Atmel AVR, y puertos digitales/analógicos de entrada/salida donde conectar sensores y/o actuadores.

La primera placa Arduino fue introducida en 2005, ofreciendo un bajo costo y facilidad para uso de novatos y profesionales.

Arduino 2006

Arduino 2016

Existen múltiples placas Arduino con diferentes características y distintos microcontroladores.

El más utilizado y estándar es el Arduino UNO, sin embargo en algunos casos podemos necesitar otra placa Arduino para adaptarnos al tipo de proyecto a realizar.

https://www.arduino.cc/en/Main/Boards

Algunas de las placas Arduino más utilizadas:

Arduino UNO

Es el modelo más estándar y el más utilizado.

Arduino MEGA

Mayor potencia, más recursos hardware y más memoria

Arduino Nano

Similar potencia que el Arduino UNO pero de menor tamaño

Arduino LilyPad

Muy pequeño. Ideal para "wearables"

Dentro de nuestros proyectos, la placa Arduino será el "*cerebro*" que gestiona toda la información recogida desde los sensores, toma decisiones y actúa sobre los elementos de control conectados.

Según las necesidades del proyecto deberemos elegir la placa Arduino más apropiada. Actualmente ArduinoBlocks sólo soporta las placas Arduino UNO, Nano y MEGA.

Por lo general la mayoría de proyectos se realizan con el modelo *Arduino UNO* y es el modelo utilizado en los ejemplos de este libro.

La placa *Arduino NANO* nos puede ser útil en casos donde el tamaño del proyecto debe ser más ajustado.

El tamaño de memoria Flash para el programa es de 32KB en los dos modelos de Arduino soportados de los cuales debemos restar el tamaño del "*bootloader*" pregrabado (0.5KB en Arduino UNO y 2KB en Arduino NANO)

1.2 PLATAFORMA ARDUINOBLOCKS

ArduinoBlocks es una plataforma web online donde podemos programar nuestra placa Arduino de forma visual sin necesidad de conocer el lenguaje C++ que utiliza Arduino IDE.

La programación en ArduinoBlocks se realiza con bloques al estilo *AppInventor* o *Scratch*. No tenemos que escribir líneas de código y no nos permitirá unir bloques incompatibles evitando así posibles errores de sintaxis.

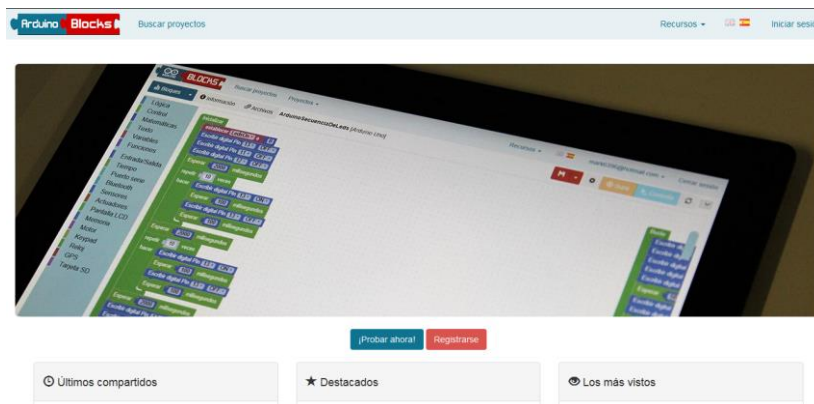
La plataforma ArduinoBlocks genera, compila y sube el programa a la placa Arduino por medio de la conexión USB. Una vez subido el programa, la placa Arduino no necesitará de la conexión al PC para funcionar pudiendo alimentarla con baterías o una fuente de alimentación para que funcione de forma autónoma.

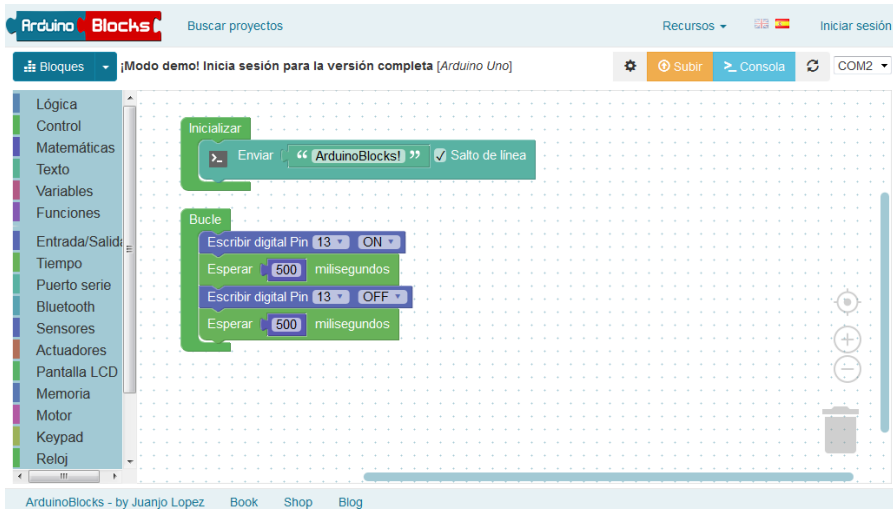
ArduinoBlocks actualmente funciona con todos los navegadores de última generación: Mozilla Firefox, Google Chrome, Opera, Safari, etc.

Registrándonos como usuarios de la plataforma ArduinoBlocks podemos aprovechar todas estas posibilidades:

- Guardar tus proyectos en la nube de ArduinoBlocks.
- Añadir información al proyecto: descripción, componentes utilizados, imágenes, etc.
- Añadir archivos adjuntos relacionados con el proyecto: esquemas, fotos, archivos para impresión 3D, aplicaciones, etc.
- Compartir proyectos con el resto del mundo.
- Importar proyectos compartidos por otros usuarios.
- Valorar y comentar proyectos
- Programar directamente Arduino desde el propio navegador (Con la aplicación: *ArduinoBlocks-Connector, ver apdo. 1.3*)
- Utilizar la consola serie desde el propio navegador

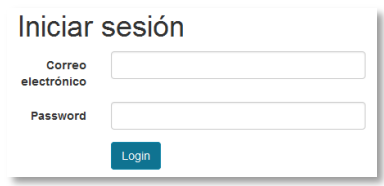
www.arduinoblocks.com





- **Inicio de sesión**

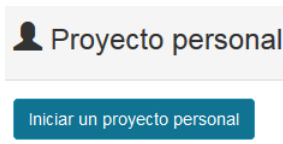
  Debemos iniciar sesión o crear previamente una nueva cuenta en caso de acceder por primera vez. Esto nos permitirá acceder a nuestros proyectos en la nube y a todas las ventajas de la comunidad ArduinoBlocks.



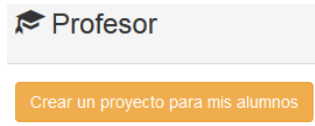
- **Creación de un nuevo proyecto**

  Para iniciar un nuevo proyecto debemos hacer clic en el menú:
  Proyectos → Nuevo proyecto

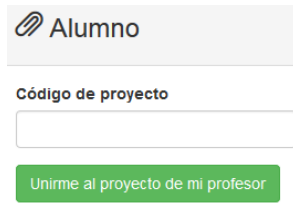
  Seleccionamos el tipo de proyecto:



Iniciar un nuevo proyecto que sólo será accesible para el usuario. Posteriormente se puede compartir al resto de la comunidad si se desea.

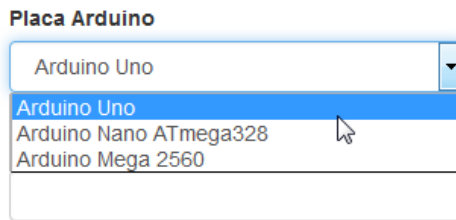


Iniciar un proyecto como profesor. De esta forma no se inicia un proyecto como tal, sino que se especifican los datos del proyecto y se genera un código para que los alumnos se puedan suscribir al proyecto. El profesor podrá supervisar y valorar los proyectos de sus alumnos.



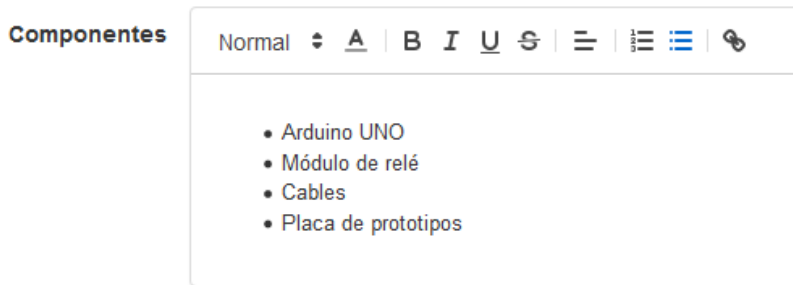
De esta forma nos unimos a un proyecto planteado por el profesor. Nosotros realizaremos el proyecto como si de un proyecto personal se tratara, pero el profesor podrá supervisar y valorar nuestro trabajo.

En caso de un proyecto personal o como profesor debemos seleccionar en el siguiente paso la placa Arduino a utilizar:



Se debe indicar un nombre descriptivo corto y una descripción más detallada.

En la sección componentes podemos indicar los componentes utilizados en el proyecto:



- **Área de programación del proyecto:**

Las principales secciones del área de programación son:

Herramientas

Área de programa

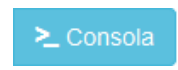
Bloque de inicialización

Bloque de bucle del programa principal

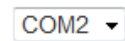
Opciones

Subir el programa a la placa Arduino conectada:

Mostrar la consola serie:



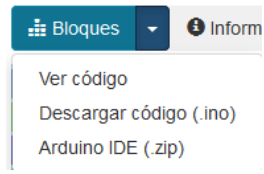
Puerto de conexión de la placa Arduino:



Para añadir bloques al programa arrastramos desde la barra de herramientas al área de programa, insertando dentro del bloque de inicialización o de bucle.

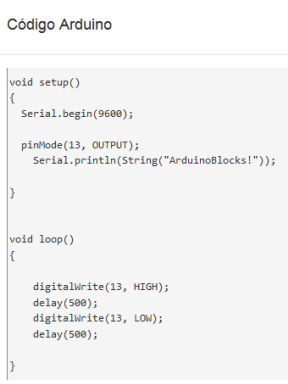
Los bloques que estén fuera del bloque de inicialización o del bloque del bucle del programa principal serán ignorados a la hora de generar el programa (excepto los bloques de funciones).

ArduinoBlocks genera el código de Arduino a partir de los bloques. El programa se puede compilar y subir directamente a la placa Arduino gracias a la aplicación *ArduinoBlocks-Connector* (disponible para descargar desde la web), sin embargo si deseamos ver o descargar el código podemos realizarlo desde el área de bloques.



Código generado por ArduinoBlocks

Copiar código en Arduino IDE o abrir el archivo descargado de código



Si descargamos o copiamos el código generado por ArduinoBlocks en Arduino IDE podemos necesitar algunas librerías no incluidas con Arduino IDE. Para ello debemos descargarla y añadirlas a la plataforma Arduino IDE para una correcta compilación del programa, o utilizar la opción descargar .zip para Arduino IDE lo que nos facilitará un archivo comprimido con el código de nuestro programa y todas las librerías necesarias incluidas en la misma carpeta.

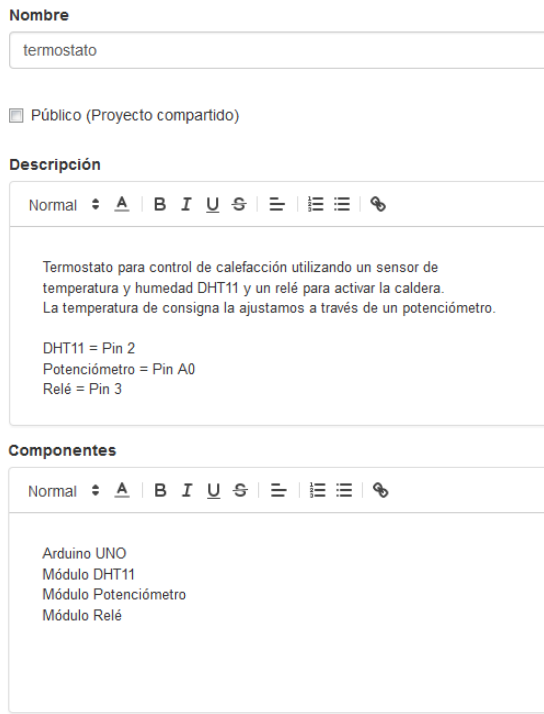
http://www.arduinoblocks.com/web/help/libraries

La opción más rápida y sencilla es la compilación y programación directa desde el propio navegador junto a la aplicación *ArduinoBlocks-Connector:*

http://www.arduinobocks.com/web/site/abconnector

- **Área de información del proyecto:**

  Un proyecto electrónico debe estar siempre correctamente documentado. En la sección información podemos añadir información o modificar la indicada durante la creación del proyecto.



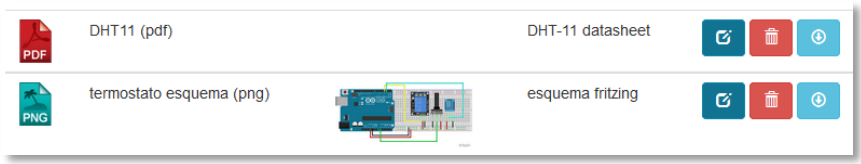
  Señalando la opción "*público*" podemos hacer que nuestro proyecto esté disponible públicamente para que otros usuarios busquen nuestro proyecto (sin poder editarlo) y pueden importar una copia a su propia cuenta.

  Ejemplo de enlace público para compartir nuestro proyecto:

  Al indicar nuestro proyecto como "*público"* aparecerá en la lista búsqueda de proyectos compartidos para todos los usuarios de ArduinoBlocks.

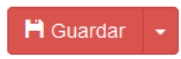
- **Área de archivos adjuntos del proyecto**

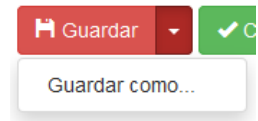
  De igual forma podemos adjuntar imágenes, hojas de datos o cualquier otro archivo relacionado con el proyecto.



- **Guardar**

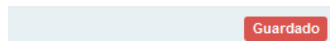
  ArduinoBlocks guarda automáticamente el proyecto cada cierto tiempo. En caso de querer asegurarnos el guardado podemos pulsar el botón "*Guardar*".
  También podemos crear un nuevo proyecto a partir del actual pulsando la opción "*Guardar como*". Automáticamente se abrirá el nuevo proyecto creado a partir del primero.





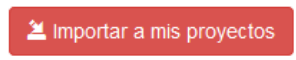
- **Barra de información**

  En la parte inferior derecha podemos obtener la información de guardado y algunos avisos que nos muestra la aplicación.



- **Importar un proyecto:**

  Si accedemos a visualizar un proyecto compartido por otro usuario aparecerá un botón "importar a mis proyectos", de esta forma podemos crear una copia del proyecto en mis proyectos personales para poder modificarlo a mi gusto.



- **"Me gusta"**

  De igual forma si accedemos a ver proyectos compartidos por otros usuarios aparecerá un botón "me gusta" para valorar positivamente el trabajo realizado por el usuario.

- **Estructura de un nuevo proyecto:** Un proyecto Arduino tiene siempre dos estructuras importantes en su interior, esto se ve reflejado claramente al crear un nuevo proyecto en ArduinoBlocks.

1. **Bloque "*inicializar*" o "*setup*":**

El contenido de este bloque sólo se una vez durante el inicio del microcontrolador de Arduino (o si pulsamos el reset y la placa Arduino se reinicia). Este bloque se utiliza para inicializar variables, configurar sensores, actuadores o periféricos, etc.

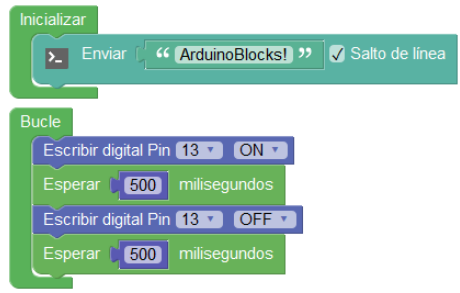
2. **Bloque "*bucle*" o "*loop*":**

El contenido de este bloque se repite indefinidamente. Dentro de este bloque añadiremos los bloques de nuestro programa con la funcionalidad deseada.

Cualquier bloque que no esté dentro del bloque de inicialización o de bucle y no forme parte de una función (ver apartado 3.2.6) será ignorado a la hora de generar el código.

Ejemplo: Al iniciar se establece la variable a 0
Se envía y se incrementa cada segundo indefinidamente:

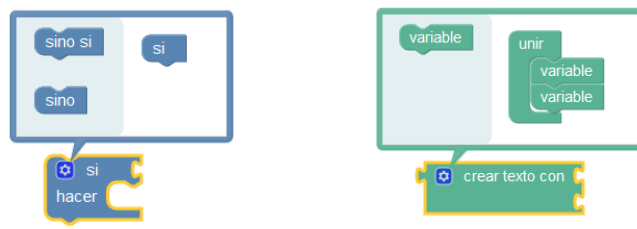
Ejemplo: Al iniciar (o reset) se envía un mensaje por el puerto serie El led conectado al pin 13 se ilumina, espera 500ms, se apaga y espera otros 500ms (este ciclo ser repetirá indefinidamente).



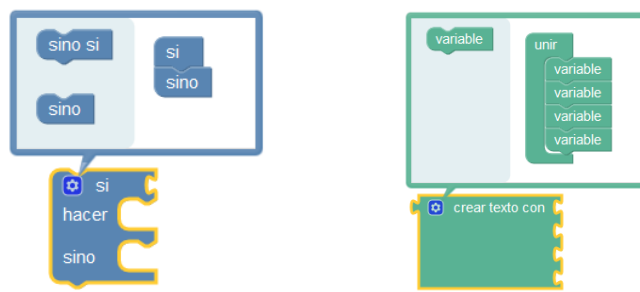
***Importante**: El "bootloader" de Arduino normalmente tiene configurada la opción de resetear el microcontrolador cuando se inicia una conexión serie, por tanto si conectamos con la consola serie del PC hay que tener en cuenta que se reiniciará el programa y se ejecutará el bloque "inicializar"*

- **Configuración de bloques:**

  Algunos bloques permiten configurar o alterar su funcionamiento. Para desplegar las opciones posibles pinchamos sobre el icono superior izquierda del bloque con apariencia de rueda dentada:

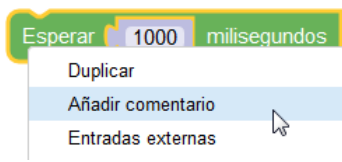


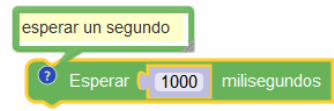
  La configuración se realiza arrastrando los modificadores del bloque de la parte izquierda a la parte derecha:

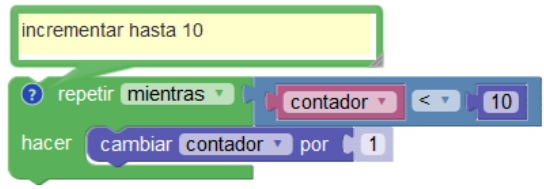


- **Comentarios**:

  Si necesitamos añadir un comentario a un bloque desplegamos las opciones del bloque pinchando con el botón derecho y añadimos un comentario pinchando en el icono del interrogante:



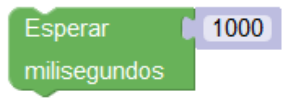




- **Otras opciones de bloque (botón derecho sobre el bloque):**

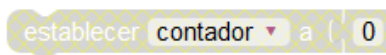
  - **Duplicar:**
    Crea una copia del bloque actual.

  - **Entradas en línea:**
    Modifica el aspecto del bloque de forma compacta o en línea.



  - **Contraer / expandir bloque:**
    Reduce el tamaño del bloque para ahorrar espacio mientras no necesitamos editarlo.

  - **Desactivar bloque:**
    El generador de código no tendrá en cuenta este bloque.



- **Eliminar:**
  Elimina el bloque.

- **Ayuda:**
  Abre un enlace con ayuda sobre la función del bloque.

- **Iconos del editor:**

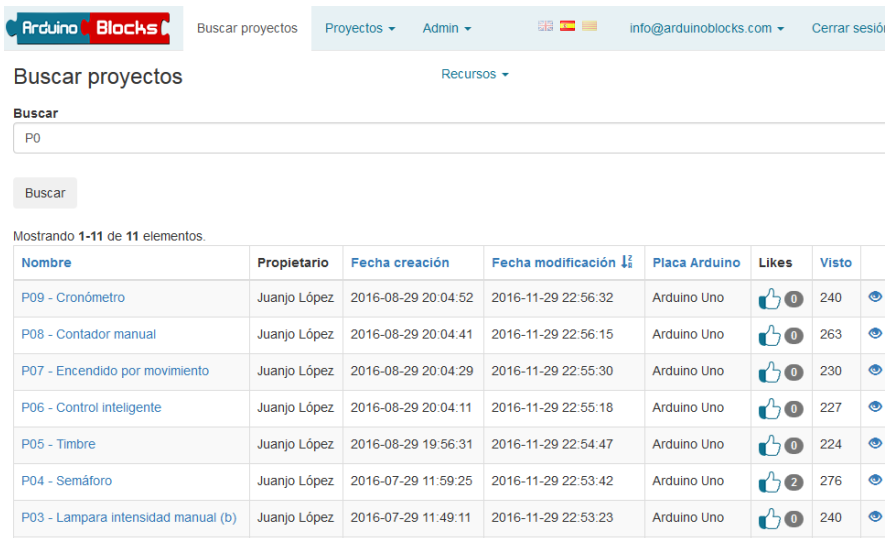
Restaurar escala y centrar.

Ampliar o reducir escala (zoom).

Arrastrando bloques sobre la papelera podemos eliminarlos fácilmente.

- **Búsqueda de proyectos compartidos por otros usuarios:**

Indicando un parámetro de búsqueda podemos buscar proyectos compartidos por otro usuario y accede a visualizarlos.



1.3 ARDUINOBLOCKS-CONNECTOR

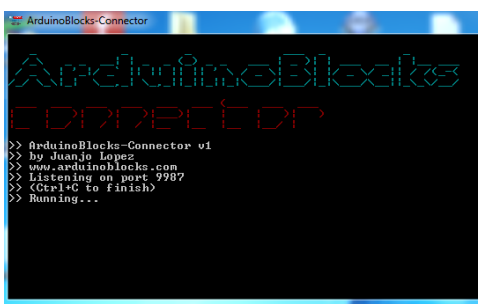
ArduinoBlocks-Connector es una aplicación nativa que hace de puente entre la plataforma on-line ArduinoBlocks y el hardware Arduino.

La aplicación *ArduinoBlocks-Connector* se encarga de recibir el código generado por ArduinoBlocks, compilarlo y subirlo a la placa Arduino, sin esta aplicación ArduinoBlocks funciona pero no puede subir el programa a la placa Arduino pues el navegador web no dispone de posibilidad de realizar estas funciones por sí sólo.

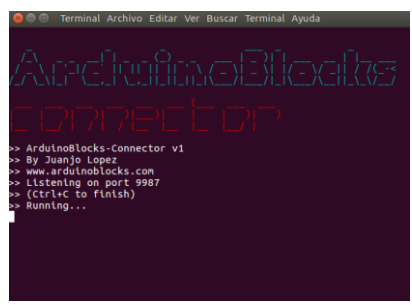
ArduinoBlocks-Connector está disponible para los principales sistemas operativos. Accede al área de descargas de arduinoblocks.com para obtener la última versión y más información sobre el proceso de instalación y configuración.

http://www.arduinoblocks.com/web/site/abconnector

ArduinoBlocks-Connector ejecutándose bajo Windows



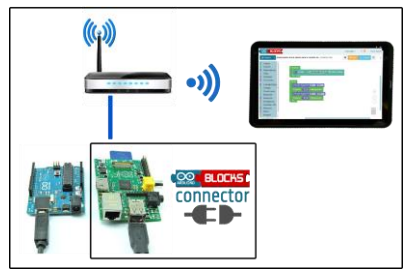
ArduinoBlocks-Connector ejecutándose bajo Ubuntu



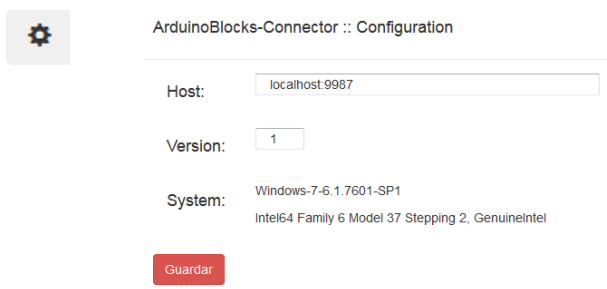
ArduinoBlocks-Connector debe ejecutarse en el equipo donde está conectado Arduino físicamente (por conexión USB). La configuración normal es instalar *ArduinoBlocks-Connector* en el mismo equipo donde se trabaja con ArduinoBlocks.

Podemos configurar la plataforma ArduinoBlocks para conectarse con la aplicación *ArduinoBlocks-Connector* en otro equipo.

Ejemplo: Arduino conectado por USB a una RaspberryPi con ArduinoBlocks-Connector instalado. La programación sería realizada desde una Tablet Android. El programa se compila y sube a la placa Arduino de forma remota a través de la red local.



Para programar remotamente un Arduino conectado a un ordenador en red con la aplicación *ArduinoBlocks-Connector*, debemos modificar el *Host* con la dirección IP del equipo en la red al que está conectado la placa Arduino (en lugar de *localhost*).

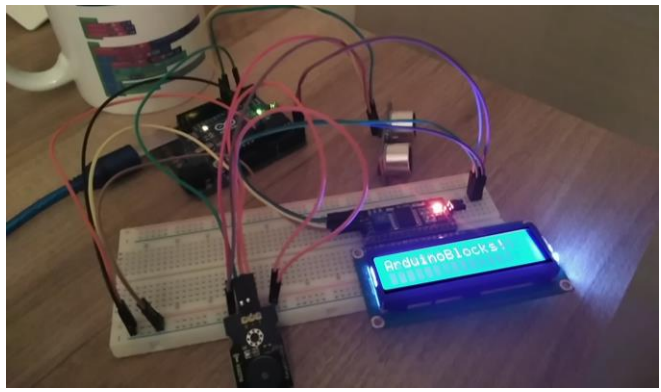


2 HARDWARE

La parte hardware del proyecto Arduino está formada por el conjunto de placas Arduino disponibles en el mercado o que tú mismo te puedes fabricar (Arduino es un proyecto totalmente abierto)

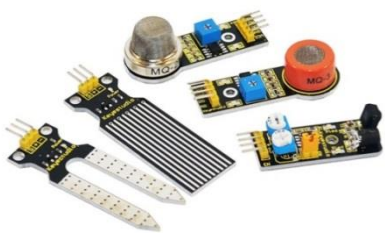
Además de la placa Arduino para cualquier proyecto robótico o de automatización debemos añadir un conjunto de sensores y actuadores para realizar las funciones necesarias.

Las conexiones entre sensores, actuadores y Arduino se pueden realizar mediante la ayuda de una placa de prototipos (*protoboard* o *breadboard*).



Podemos utilizar sensores y actuadores de forma modular. Existen múltiples soluciones de este tipo en el mercado.

Ejemplo: Sensores y actuadores modulares

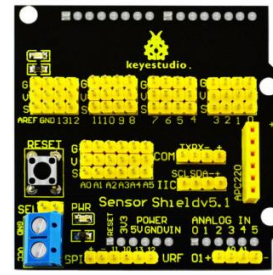


Kits modulares de iniciación:

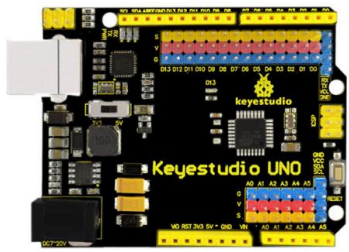
(keyestudio)

Kit de 36 sensores y actuadores

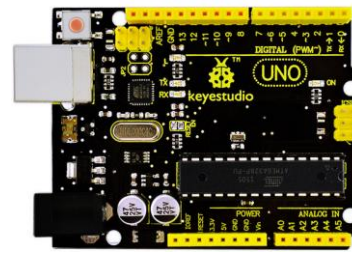
Sensor shield



Arduino UNO(upgrated version)



Arduino UNO R3



Keyestudio EASY-plug

Ejemplo de conexión modular:

2.1 CONCEPTOS BÁSICOS DE ELECTRÓNICA

A la hora de iniciar un proyecto de robótica debemos tener claras algunas nociones de electricidad y electrónica básicas. El propósito de este libro no es aprender estos conceptos sobre electrónica, por lo tanto se asumen unos conocimientos previos básicos de electricidad y electrónica.

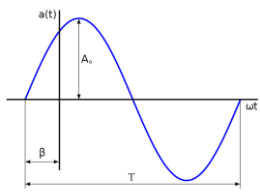
Vamos a hacer un breve repaso de los conceptos más importantes que debemos conocer:

La **corriente eléctrica** se define como el movimiento de los electrones a través de un conductor, según el tipo de desplazamiento se define como corriente continua o alterna.

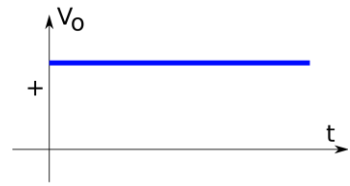
En la **corriente alterna** los electrones cambian de dirección de movimiento 50 veces por segundo (en redes eléctricas como la de España de 50Hz, en otros países puede ser 60Hz). El movimiento descrito por los electrones es sinusoidal.

En la **corriente continua** los electrones se desplazan siempre en la misma dirección. Arduino funciona con corriente continua.

C.A.



C.C.



Las principales magnitudes físicas que debemos conocer son:

Voltaje o tensión eléctrica: Energía acumulada por unidad de carga que hace que las cargas circulen por el circuito (genera una corriente). Se mide en voltios (V)

Intensidad: número de electrones que atraviesan la sección de un conductor por unidad de tiempo. Se mide en amperios (A)

Resistencia: mide la oposición que ofrece un material al paso de corriente eléctrica. Se mide en Ohmios (Ω)

Potencia: es la energía consumida o desprendida por un elemento en un momento determinado. Se mide en Watios (W)

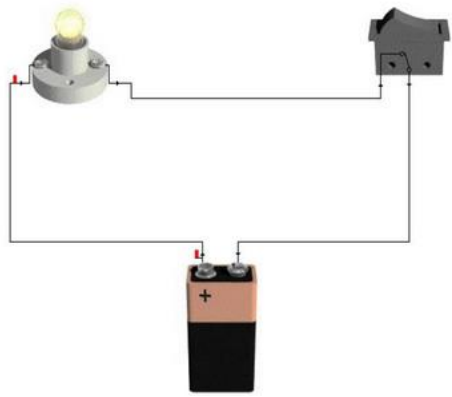
$$P = V \cdot I$$

Ley de Ohm: Es una sencilla fórmula matemática que relaciona estas tres

magnitudes básicas entre sí.

$$V = I \cdot R$$

Circuito eléctrico: Conjunto de elementos unidos de tal forma que permiten el paso de corriente eléctrica para conseguir algún efecto (luz, calor, movimiento, etc.)



2.2 LA FUENTE DE ALIMENTACIÓN

La placa Arduino necesita energía para funcionar, existen varias formas de alimentar la placa Arduino:

-A través del conector USB: cuando conectamos al ordenador para programarlo o utilizando un "*power bank*" con conexión USB por ejemplo.

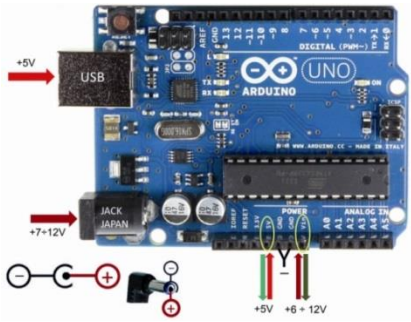
-A través del conector de alimentación externa. La fuente de alimentación conectada debe ofrecer un voltaje DC de 7 a 12v. Internamente la placa Arduino UNO regula la tensión a 5v.

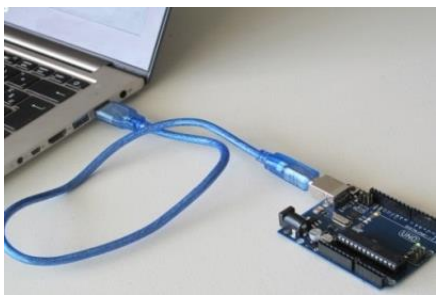
A través de los pines 3.3v , 5v, GND y Vin obtenemos la alimentación para circuitos auxiliares, sensores, shields, etc.:

- 3.3v proporciona una tensión de 3.3v y una corriente máxima de 50mA
- 5v proporciona una tensión de 5v y una corriente máxima de 300mA
- GND es el nivel 0v de referencia
- Vin proporciona la tensión de alimentación conectada al conector de alimentación (sin regular, igual a la tensión de la fuente de alimentación conectada)

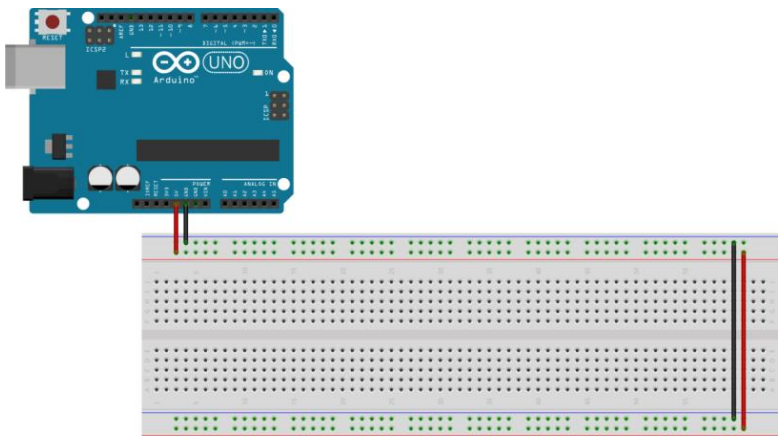
Normalmente alimentaremos la placa Arduino a través del USB durante su

programación desde el PC. Si la placa Arduino va a funcionar de forma autónoma sin interactuar con el PC podemos alimentarla desde una fuente de alimentación o con una batería a través del conector Jack (aplicar de 7 a 12v).





Conexión recomendada de la tensión de alimentación a la placa de prototipos.



Placa de prototipos: https://es.wikipedia.org/wiki/Placa_de_pruebas

2.3 LA PLACA ARDUINO UNO

Arduino UNO es la placa Arduino más utilizada de todas las versiones existentes, y es la que vamos a utilizar en este libro,

Especificaciones técnicas:

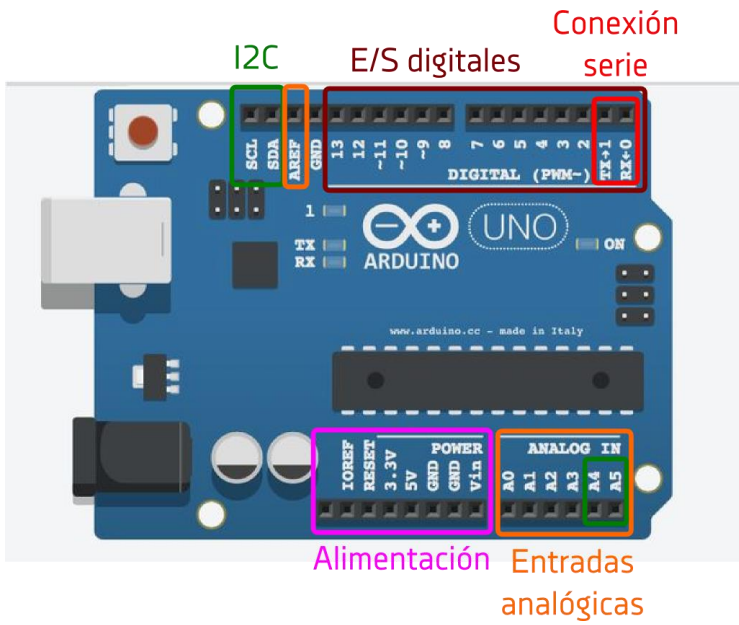
Microcontrolador	ATmega328P
Alimentación	5V
Alimentación (recomendada)	7-12V
Alimentación (límite)	6-20V
Número de pines E/S	14 (6 con salida PWM)
Número de pines PWM	6
Número de pines analógicos	6
Corriente pines E/S	20 mA
Corriente pin de 3.3V	50 mA
Memoria Flash	32 KB (ATmega328P) (0.5 KB para el bootloader)
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Velocidad de reloj	16 MHz
Largo	68.6 mm
Ancho	53.4 mm
Peso	25 g

https://www.arduino.cc/en/Main/ArduinoBoardUno

El tamaño de la memoria para programa de la placa Arduino UNO es de 32 KBytes, si se supera este tamaño al generar y compilar el código se producirá un mensaje de error al intentar subir el programa a la placa Arduino.

La placa Arduino UNO dispone de múltiples pines de conexión en formato de conector hembra:

Los pines están agrupados por función o tipo:



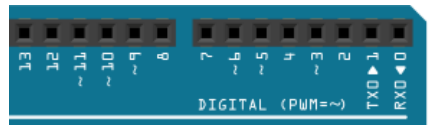
PINES DE ALIMENTACIÓN:

Permiten obtener la tensión necesaria para alimentar sensores, actuadores u otros periféricos conectados a la placa Arduino

IOREF	Indica la tensión de trabajo de las E/S de este modelo de placa. (Arduino UNO IOREF = 5V)
RESET	Permite reiniciar la placa a través de este pin
3.3V	Suministra 3.3v
5V	Suministra 5v
GND	Tierra o 0V (negativo)
Vin	Obtiene el voltaje aplicado por la fuente de alimentación con la que se está alimentando. También permite alimentar la placa por este pin, aplicando tensión de entrada (7-12v)

PINES DE ENTRADAS/SALIDAS DIGITALES

Los pines digitales permiten trabajar con dos estados (ON/OFF, Activado/Desactivado, 1/0). Los pines se pueden configurar como entrada o como salida según se necesite conectar un sensor o un actuador.

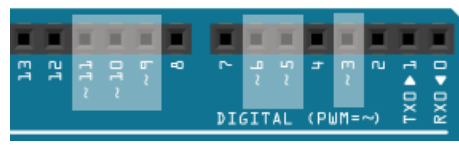


Pin digital configurado como entrada:	Tensión aplicada externamente al pin: 0v...1,5v = OFF / 0 / desactivado 3v...5v = ON / 1 / activado
Pin digital configurado como salida:	Tensión suministrada por el pin: OFF / 0 / desactivado = 0v ON / 1 / activado = 5v

Pines: 0, 1	Estos dos pines se pueden utilizar como entradas / salidas digitales. ArduinoBlocks no utiliza los pines 0,1 como pines de E/S. Los reserva para la conexión serie y la programación desde el PC.
Pines: 2...13	Pines digitales de uso general. Podemos utilizarlos como entrada o salida. Según utilicemos un actuador o un sensor ArduinoBlocks configurará automáticamente el pin como entrada o salida según sea necesario.
Pines A0...5	Los pines de entrada analógicos también se pueden usar como pines de E/S digitales convencionales.

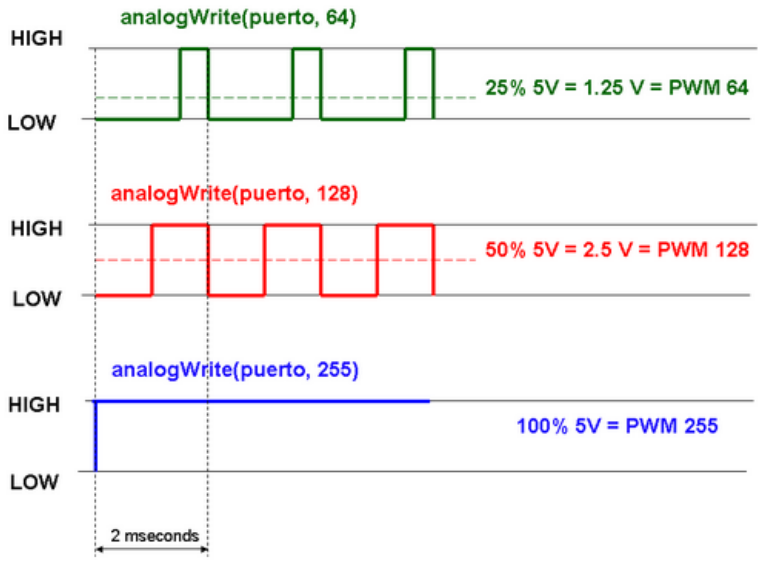
PINES DE SALIDAS ANALÓGICAS

Arduino no tiene salidas puramente analógicas, pero podemos imitar a una salida analógica mediante la técnica PWM (Pulse Width Modulation = Modulación en Anchura de Pulso).



Pines: 3,5,6,9,10,11	Dentro de los pines digitales estos pines permiten utilizar como salida digital PWM (modulación en ancho de pulso) para simular una salida pseudo-analógica.

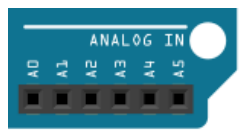
Funcionamiento del PWM:



PINES DE ENTRADAS ANALÓGICAS

Estos pines sólo se pueden utilizar como entradas. Las entradas analógicas leen un voltaje entre 0 y 5V y a través de un ADC (Analog to Digital Converter) obtienen un valor de 10 bits proporcional a la señal de entrada.

10 bits = 1024 valores (0 ... 1023)



Pines: A0...A5	6 Entradas analógicas (resolución 10 bits: 0...1023)

PINES DE COMUNICACIÓN SERIE

Estos pines conectan con la unidad serie (UART) interna del microprocesador de Arduino. Una conexión serie utiliza un pin para la señal de envío de datos (TX) otro para la recepción de datos (RX) y la señal GND.

Pin 0	RX: a través de este pin se reciben datos hacia Arduino
Pin 1	TX: a través de este pin se envían datos desde Arduino

Los pines 0,1 conectan con el puerto serie implementado en el hardware Arduino. En caso de necesidad se podrán implementar otras conexiones serie a través de otros pines digitales emulando el puerto serie con librerías software (por ejemplo para la conexión con el módulo Bluetooth HC-06 explicado más adelante)

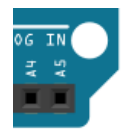
https://www.arduino.cc/en/Reference/SoftwareSerial

(más información sobre la conexión serie: Apartado 2.6.1)

PINES DE COMUNICACIÓN I2C

El bus de comunicación I2C permite conectar redes de periféricos con una comunicación bidireccional entre Arduino y el periférico.

Pin A4 / SDA	Línea de datos del bus I2C
Pin A5 / SCL	Línea de reloj del bus I2C

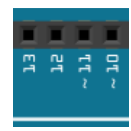


(más información sobre el bus I2C: Apartado 2.6.2)

PINES DE COMUNICACIÓN SPI

El bus de comunicación SPI permite conectar redes de periféricos con una comunicación bidireccional entre Arduino y el periférico.

Pin 12 / MISO	Master In Slave Out
Pin 11 / MOSI	Master Out Slave In
Pin 13 / SCK	Serial Clock
Pin 10 / SS	Slave Select



(más información sobre bus SPI: Apartado 2.6.3)

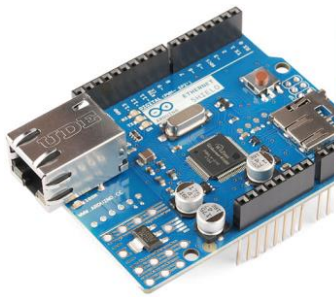
Una de las ventajas de la placa Arduino es que incorpora un programa pregrabado en el microcontrolador. Este programa conocido como "*bootloader*" o cargador de arranque permite desde el principio reprogramar el microcontrolador de Arduino a través de su puerto USB sin necesidad de un programador externo ni el uso del sistema ICSP (In Circuit Serial Programming) utilizado en otros sistemas.

Otra ventaja evidente del sistema Arduino es el entorno de programación "*Arduino IDE*" sencillo ofrecido de forma totalmente libre y que facilita enormemente la programación de este tipo de microcontroladores para inexpertos.

La clave del éxito de la plataforma Arduino es que es una plataforma totalmente abierta y existe una gran comunidad de colaboradores y desarrolladores. Un ejemplo de las aportaciones de la comunidad Arduino son las conocidas como "*shields*", que son módulos de extensión apilables para Arduino con las que podemos añadir rápidamente funcionalidades a la placa Arduino.

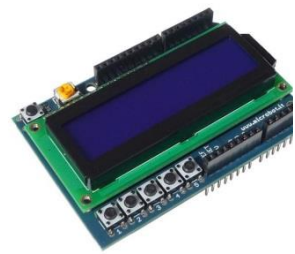
Ejemplos de "Shields" para Arduino UNO:

Ethernet

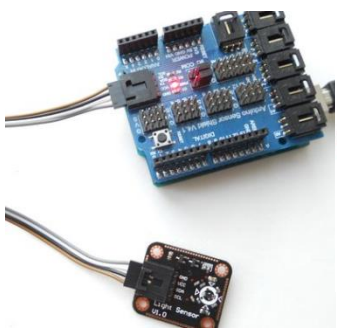


Relés

LCD y botones



Sensores



GPS / GSM

WiFi

+información:
https://www.arduino.cc/en/Main/ArduinoShields

2.4 SENSORES

Un sensor es un objeto capaz de detectar magnitudes físicas o químicas y transformarlas en variables eléctricas.

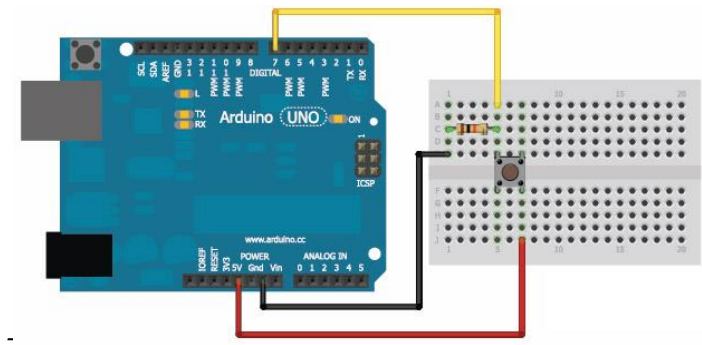
Los sensores o periféricos de entrada nos permiten obtener información del mundo real para utilizarla desde el programa de Arduino.

La interfaz de conexión de un sensor con Arduino lo podemos clasificar en tres tipos: digital, analógico o datos.

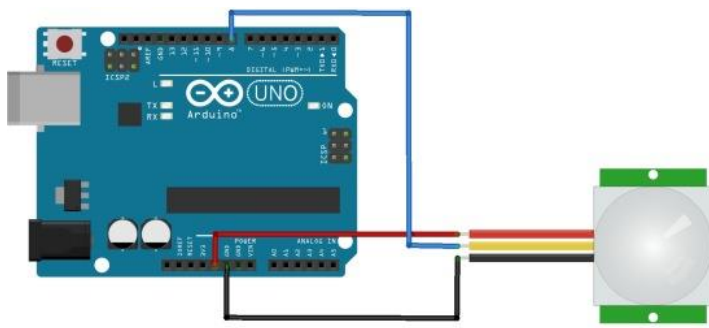
-**Digital**: un sensor digital sólo tiene dos estados: activado/desactivado, ON/OFF, 1/0, Alto/Bajo, … En este caso conectaremos el sensor a una de las entradas digitales de Arduino para leer el estado.

Ejemplo: un pulsador es un tipo de sensor sencillo que sólo nos da dos estados, "*pulsado o no pulsado*". Conectado a la placa Arduino debe generar 0v en reposo y 5v al pulsarlo. De esta forma desde el programa de Arduino podremos leer el estado del botón.

Ejemplo de conexión de un sensor digital (pulsador):

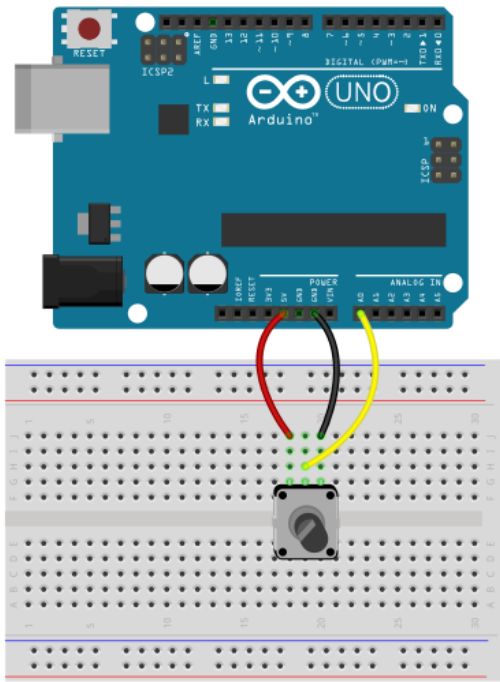


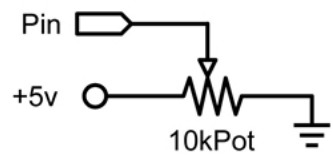
Conexión de un sensor digital de movimiento (PIR):



-**Analógico**: el sensor nos puede dar un rango de valores, normalmente se traduce en un valor de tensión o de corriente variable en función de la señal captada al sensor. En este caso conectaremos el sensor a una de las entradas analógicas de Arduino. El rango de entrada será una tensión entre 0v (GND) y 5v.

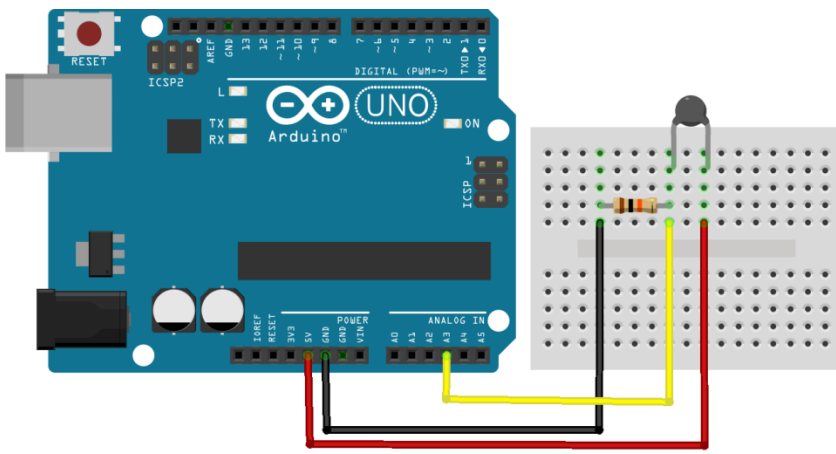
Conexión de un sensor potenciómetro al pin de entrada analógico A0:





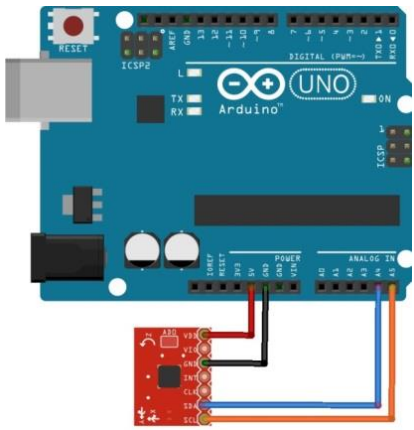
Cuando el potenciómetro está en un extremo el voltaje aplicado al pin de Arduino es 5v, en el otro extremo es de 0v. Durante el recorrido, gracias a la variación de resistencia del potenciómetro, se aplica el valor de voltaje proporcional a la posición del potenciómetro entre 0 y 5 voltios.

Conexión de una resistencia NTC (variable según la temperatura):

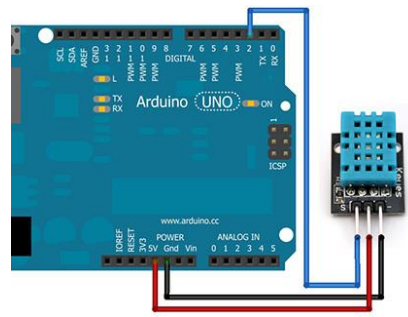


-**Datos**: el sensor ofrece su información a través de una interfaz de comunicación. La forma de comunicación puede ser por sistemas estándar como I2C o SPI (ver apartado 2.6 sobre buses de comunicación) o algunos sensores usan su propio protocolo para codificar la información y debemos realizar desde el software la decodificación correcta para interpretar los datos del sensor (normalmente los desarrolladores de este tipo de sensores ofrecen una librería software para Arduino que hace todo el trabajo)

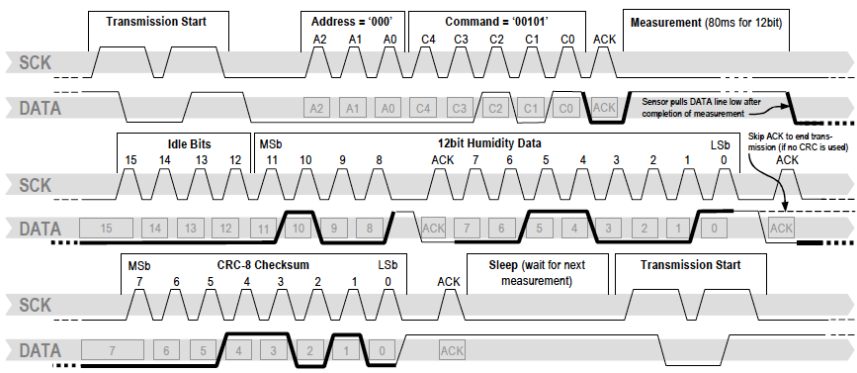
Sensor de acelerómetros con conexión I2C



Sensor DHT11 de temperatura y humedad con protocolo de comunicación propio



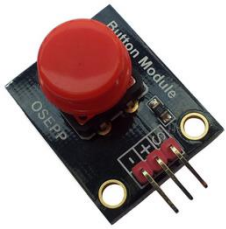
Trama de datos recibida desde el sensor DHT11



¡No te asustes, ArduinoBlocks hará el trabajo de decodificar estos datos!

Algunos módulos de sensores utilizados con Arduino:

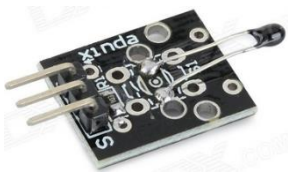
Pulsador



Potenciómetro

Sensor de distancia

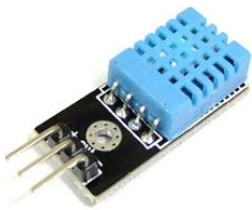
Sensor de temperatura



Sensor de obstáculos

Codificador (encoder) rotativo

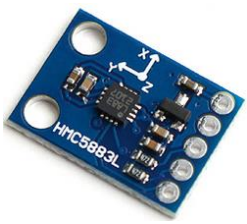
Sensor DHT-11 de temperatura y humedad



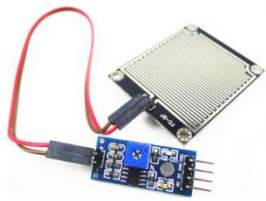
Joystick

Sensor de efecto Hall

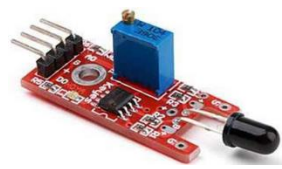
Sensor de orientación



Sensor de lluvia



Sensor de llama



2.5 ACTUADORES

Un actuador es un dispositivo capaz de transformar la energía eléctrica en la activación de un proceso con la finalidad de generar un efecto sobre elementos externos.

Un actuador o periférico de salida permite actuar sobre el mundo real desde el programa de Arduino.

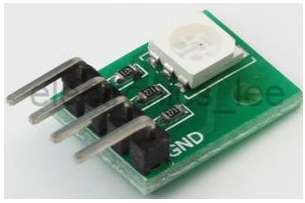
Algunos módulos de actuadores utilizados con Arduino:

Módulo relé:

Servomotor:

Módulo led:

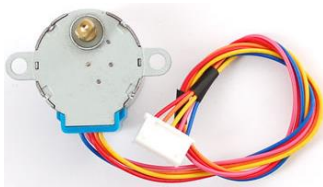
Módulo led RGB:



Módulo zumbador:

Pantalla LCD:

Motor paso a paso:



Motor DC:

Módulo emisor IR:

2.6 COMUNICACIONES

La placa Arduino permite múltiples vías de comunicación con el exterior, por un lado disponemos del bus I2C o del SPI pensado para periféricos externos o sensores mientras que como vía de comunicación principal para la programación o monitorización tenemos la conocida como conexión serie (puerto serie) a través del conector USB.

2.6.1 COMUNICACIÓN SERIE

El microcontrolador Atmel de Arduino dispone de un controlador de comunicación serie (UART) integrado. La comunicación se realiza de forma bidireccional, utilizando un pin para transmitir los datos y otro para recibir.

Es muy importante tener en cuenta que este puerto serie es el que se utiliza para "subir" el firmware y reprogramar la placa Arduino desde un ordenador (*bootloader*).

Las primeras placas Arduino disponían de un conector de puerto serie tipo DB9 de 9 pines utilizado antiguamente para este tipo de conexiones. Hoy en día se utiliza un chip de conversión serie a USB que permite emular en el equipo un puerto serie estándar.

Durante el uso normal podemos utilizarlo como vía de comunicación sencilla entre el microcontrolador y el un PC.

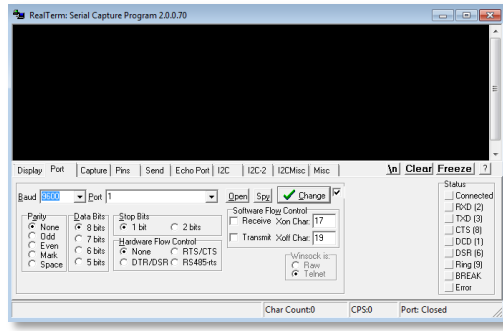
Arduino UNO sólo dispone de un puerto serie hardware aunque podemos emular más puertos serie vía software. La conexión serie es utilizada por algunos periféricos o sensores para interactuar con Arduino:

Módulo GPS serie:

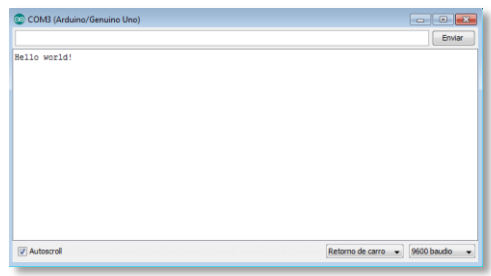
Módulo Bluetooth HC-06 con conexión serie

Para poder desde un ordenador visualizar los datos recibidos vía puerto serie debemos utilizar una aplicación de tipo “terminal” o “consola” serie:

Realterm - consola serie para Windows



Arduino IDE - serial monitor



ArduinoBlocks – consola serie
(*Se necesita instalar la aplicación ArduinoBlocks-Connector*)

A la hora de establecer una conexión serie los dos extremos que intervienen en la conexión (en este caso Arduino y el PC) deben establecer el mismo valor en la velocidad de la conexión.

Velocidad en baudios estándar: 9600 bits por segundo

Otras velocidades utilizadas: 4800, 19200, 38400, 57600, 115200, ...

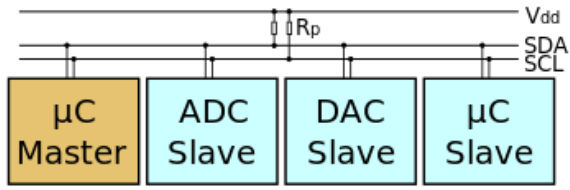
2.6.2 COMUNICACIÓN I2C/TWI

El bus I2C (I²C o TWI) es un bus de datos seire desarrollado por Philips.

Se utiliza principalmente internamente para la comunicación entre diferentes partes de un circuito, por ejemplo, entre un controlador y circuitos periféricos integrados.

Atmel introdujo por motivos de licencia la designación TWI (Two-Wired-Interface) actualmente utilizada por algunos otros fabricantes. Desde el punto de vista técnico, TWI e I2C son idénticos.

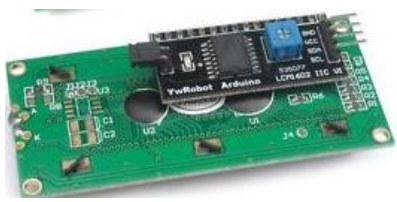
El I2C está diseñado como un bus maestro-esclavo. La transferencia de datos es siempre inicializada por un maestro; el esclavo reacciona. Es posible tener varios maestros (Multimaster-Mode). En el modo multimaestro pueden comunicar dos maestros entre ellos mismos, de modo que uno de ellos trabaja como esclavo. El arbitraje (control de acceso en el bus) se rige por las especificaciones, de este modo los maestros pueden ir turnándose.

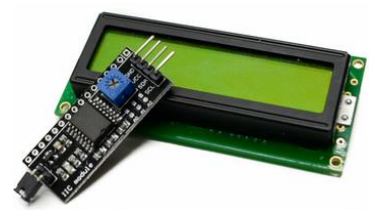


La dirección de I2C estándar es el primer byte enviado por el maestro, aunque los primeros 7 bits representan la dirección y el octavo bit (R/W-Bit) es el que comunica al esclavo si debe recibir datos del maestro (low/bajo) o enviar datos al maestro (high/alto). Por lo tanto, I2C utiliza un espacio de direccionamiento de 7 bits, lo cual permite hasta 112 nodos en un bus (16 de las 128 direcciones posibles están reservadas para fines especiales).

Cada uno de los circuitos integrados con capacidad de soportar un I2C tiene una dirección predeterminada por el fabricante, de la cual los últimos tres bits (subdirección) pueden ser fijados por tres pines de control. En este caso, pueden funcionar en un I2C hasta 8 circuitos integrados. Si no es así, los circuitos integrales (que precisan ser idénticos) deben ser controlados por varios buses I2C separados.

Pantalla LCD con módulo de conexión I2C

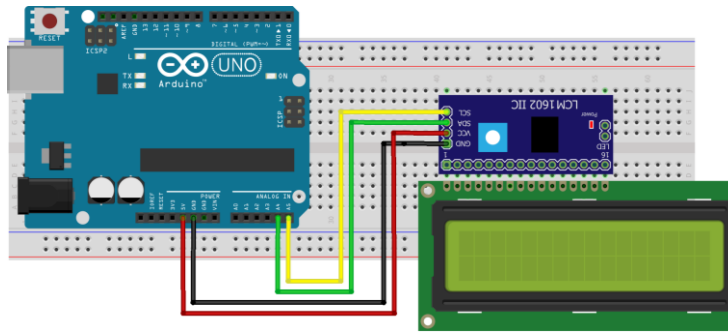




La conexión I2C en Arduino UNO se realiza en los pines:

SDA: Pin A4
SCL: Pin A5

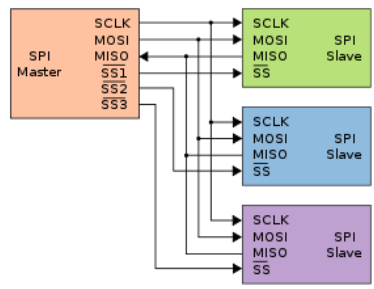
Ejemplo de conexión de módulo I2C para control de pantalla LCD:



2.6.3 COMUNICACIÓN SPI

El Bus SPI (del inglés *Serial Peripheral Interface*) es un estándar de comunicaciones, usado principalmente para la transferencia de información entre circuitos integrados en equipos electrónicos. El bus de interfaz de periféricos serie o bus SPI es un estándar para controlar casi cualquier dispositivo electrónico digital que acepte un flujo de bits serie regulado por un reloj (comunicación sincrónica).

Incluye una línea de reloj, dato entrante, dato saliente y un pin de *Chip Select*, que conecta o desconecta la operación del dispositivo con el que uno desea comunicarse. De esta forma, este estándar permite multiplexar las líneas de reloj.



La sincronización y la transmisión de datos se realiza por medio de 4 señales:

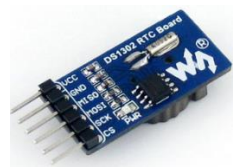
- **SCLK** *(Clock)*: Es el pulso que marca la sincronización. Con cada pulso de este reloj, se lee o se envía un bit. También llamado TAKT (en Alemán).
- **MOSI** *(Master Output Slave Input)*: Salida de datos del Master y entrada de datos al Slave. También llamada SIMO.
- **MISO** *(Master Input Slave Output)*: Salida de datos del Slave y entrada al Master. También conocida por SOMI.
- **SS/CS/Select**: Para seleccionar un Slave, o para que el Master le diga al Slave que se active. También llamada SSTE.

Algunos periféricos SPI:

Tarjeta micro SD

Pantalla OLED

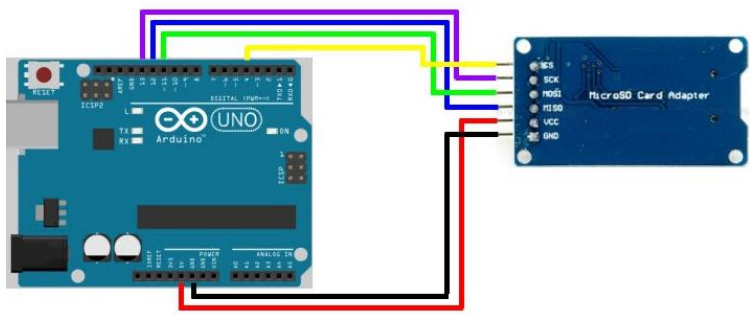
Reloj de tiempo real (RTC)



La conexión SPI en Arduino UNO se realiza en los pines:

MOSI: Pin 11
MISO: Pin 12
SCLK: Pin 13
SS/CS: Depende de la programación, puede usarse cualquier pin.

Ejemplo de conexión de módulo para tarjetas SD:
(el pin SS está conectado al pin 4)



3 SOFTWARE

Una vez tenemos definido el hardware necesario para un proyecto el siguiente paso es programar el microcontrolador de la placa Arduino para que realice las tares necesarias para el funcionamiento deseado.

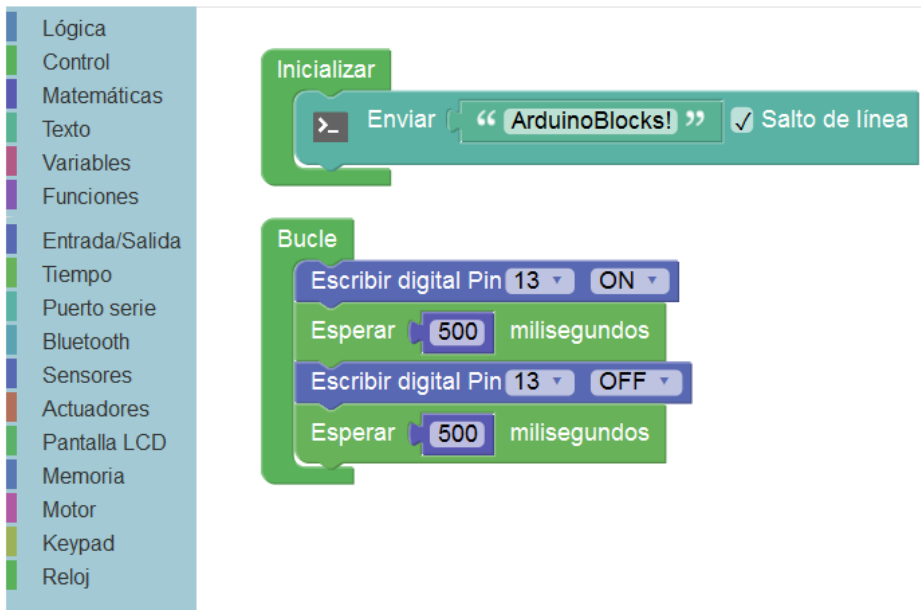
La programación de la placa Arduino se realiza normalmente en lenguaje C++ desde el entorno Arduino IDE. Para programar debemos conocer primero este lenguaje, lo cual supone mucho tiempo del que muchas veces no disponemos.

En los últimos años han aparecido entornos mucho más sencillos e intuitivos para desarrollar aplicaciones que nos permiten introducirnos de forma práctica y sencilla en el mundo de la programación. Es el caso de *Scratch*, un entorno de desarrollo de videojuegos multiplataforma, y *AppInventor*, un entorno de desarrollo de aplicaciones para dispositivos móviles *Android*.

ArduinoBlocks, al igual, es un entorno online que nos permite programar Arduino (sin necesidad de conocer el lenguaje de programación C++) de forma visual al estilo de programación de bloques.

ArduinoBlocks implementa bloques generales comunes a cualquier entorno de programación y por otro lado bloques específicos para Arduino donde podemos acceder a leer/escribir datos de los pines de entrada/salida, acceder a información de sensores conectados, manejar actuadores, periféricos como la pantalla LCD y muchas funcionalidades más.

Programa de ejemplo generado automáticamente en modo "prueba":



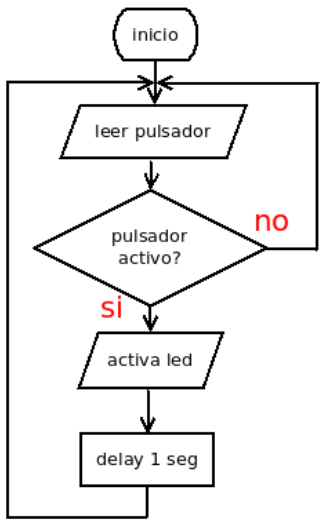
3.1 ALGORITMOS

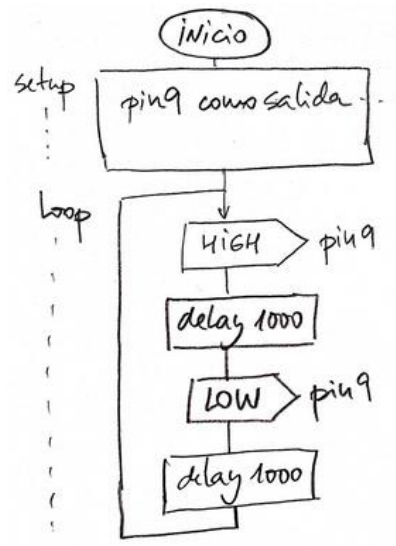
Un algoritmo es un conjunto prescrito de instrucciones o reglas bien definidas, ordenadas y finitas que permite realizar una actividad mediante pasos sucesivos que no generen dudas a quien deba realizar dicha actividad.

A la hora de programar en cualquier lenguaje de programación lo primero que tenemos que hacer es plantear el algoritmo que queremos desarrollar y posteriormente implementarlo en el lenguaje de programación elegido.

A pesar de que la programación por bloques es muy intuitiva y visual, siempre es recomendable plantear el algoritmo antes de empezar un proyecto.

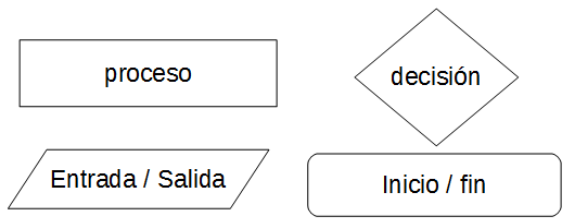
Ejemplos de diagramas de flujo para definir un algoritmo:





La definición previa del algoritmo nos permitirá agilizar el proceso de creación del programa.

Simbología básica para la definición gráfica de un algoritmo:



3.2 BLOQUES DE USO GENERAL

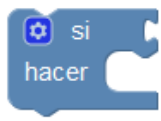
Los bloques de uso general nos permiten implementar funciones comunes en cualquier entorno o sistema programable. Esto incluye funciones lógicas, matemáticas, condiciones, bucles, funciones de texto, etc.

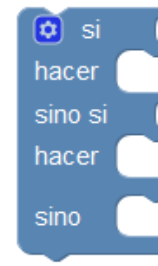
3.2.1 LÓGICA

Con estos bloques tenemos acceso a las funciones lógicas necesarias para implementar en nuestro programa de Arduino.

Las funciones lógicas trabajan con valores o expresiones de "verdadero" o "falso"

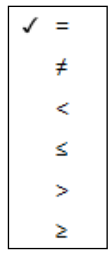
- **Condición / decisión**: Evalúa una condición lógica, si se cumple realiza el bloque "hacer" si no se cumple realiza el bloque "sino" (opcional)





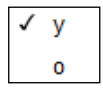
Ejemplo:

- **Evaluar condición:** Devuelve verdadero o falso según si la condición indicada se cumple entre los dos operandos.



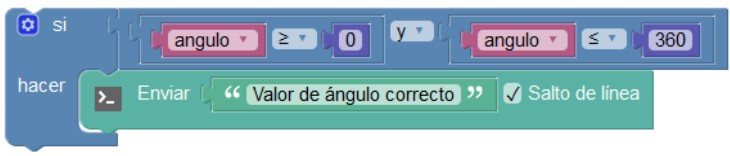
=	Igual
≠	Distintos
<	Menor que
≤	Menor o igual que
>	Mayor que
≥	Mayor o igual que

- **Conjunción/Disyunción**: Evalúa dos expresiones lógicas y devuelve verdadero o falso según la función lógica seleccionada.



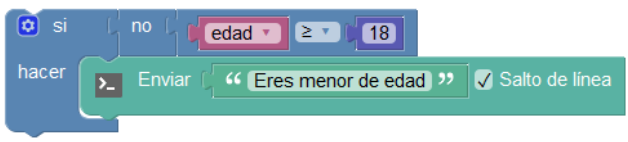
y “and”	Se cumple si las dos operandos son verdaderos
o “or”	Se cumple si alguno de los dos operandos es verdadero.

Ejemplo:



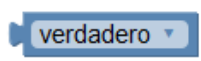
- **Negación**: Permite negar (invertir) un valor lógico de verdadero o falso.

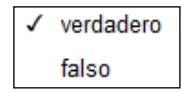
Ejemplo:

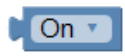


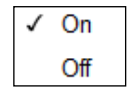
- **Constantes lógicas:** son valores booleanos indicando uno de los dos

estados posibles









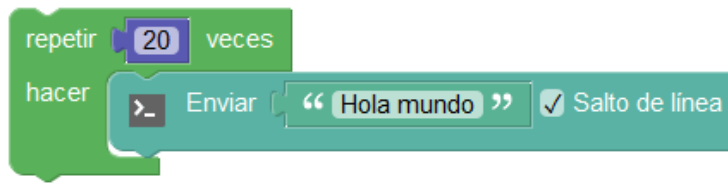
On	= Verdadero
Off	= Falso

3.2.2 CONTROL

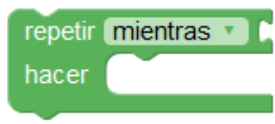
Las estructuras de control nos permiten realizar bucles e iteraciones.

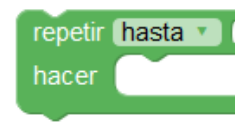
- **Repetir**: Repite (n) veces los bloques de su interior.

Ejemplo:



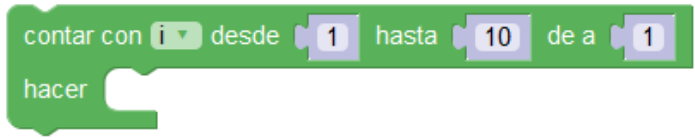
- **Repetir según condición**: Repite mientras o hasta que se cumpla una condición.



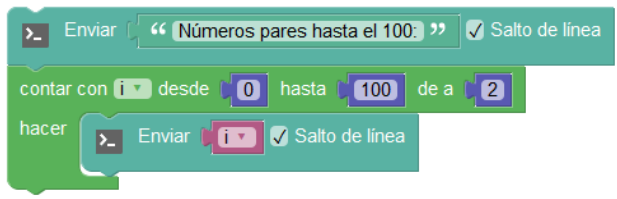


Ejemplo:

- **Contar**: Realiza un bucle contando con un variable *índice.* Se define un valor de inicio, una valor de fin y los incrementos que se realizarán en cada iteración del bucle. Dentro del bucle podremos usar esta variable.



Ejemplo:



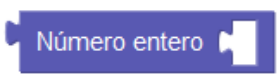
3.2.3 MATEMÁTICAS

- **Constante numérica**: Permite especificar un valor numérico entero o decimal

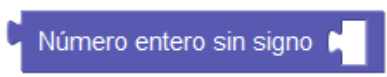
Ejemplo:

- **Número entero / sin signo:** Trata el valor como un entero. Si especificamos sin signo, trata el valor como una variable sin signo internamente.

  Para las variables ArduinoBlocks utiliza el tipo de dato "*double*" cuando traduce el programa a lenguaje C++. En caso de hacer la conversión se trata como un "*cast*" a un tipo de datos "*long*" o "*unsigned long*"

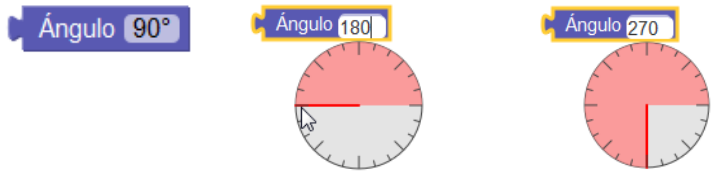


Trata el valor como tipo entero

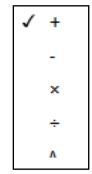


Trata el valor como tipo entero sin signo

- **Ángulo**: Permite definir un valor de ángulo en grados. Es un valor numérico tal cual, pero con la ventaja que permite definir el valor de una forma visual viendo el ángulo gráficamente.



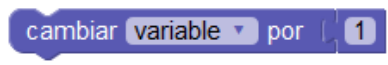
- **Operaciones básicas:**



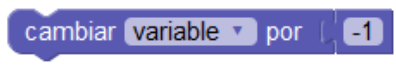
+	Suma
-	Resta
x	Multiplicación
÷	División
^	Potencia

Ejemplo:

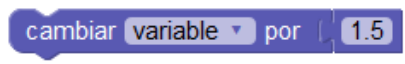
- **Cambiar variable**: Aumenta o disminuye el valor de una variable por el valor indicado (si es un valor positivo aumenta si es negativo disminuye)



Aumenta la variable en +1
variable = variable + 1

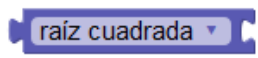


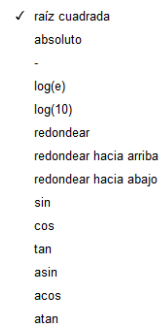
Disminuye la variable en -1
variable = variable – 1



Aumenta la variable en +1.5
variable = variable + 1.5

- **Funciones matemáticas:**



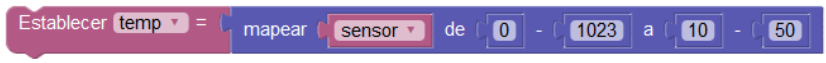


- **Atan2**: Calcula la arco-tangente de *y/x*, siendo *y* el primer parámetro y *x* el segundo.

- **Mapear**: Permite modificar el rango de un valor o variable desde un rango origen a un rango destino. Esta función es especialmente útil para adaptar los valores leídos de sensores o para adaptar valores a aplicar en un actuador.

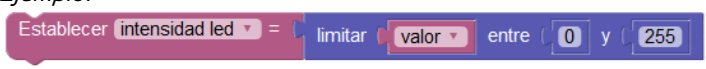
Ejemplo:
-Sensor de temperatura: 10ºC ... 50ºC
-Arduino lectura analógica: 0 ... 1023

Necesitamos convertir del rango 0-1023 leído al rango 10ºC-50ºC:



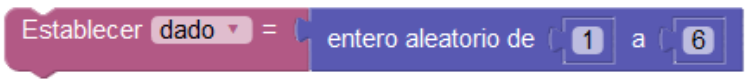
- **Limitar**: Permite acotar el valor mínimo y máximo.

Ejemplo:



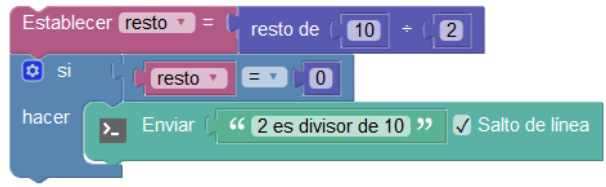
- **Número aleatorio**: Genera un valor aleatorio entre los valores especificados.

Ejemplo:



- **Resto**: Obtiene el resto de la división de los dos operandos.

Ejemplo:

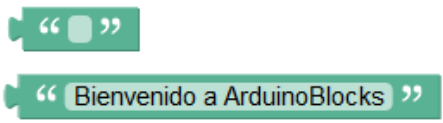


Ejemplo: uso de varios bloques de operaciones matemáticas:

3.2.4 TEXTO

Las funciones de texto son especialmente útiles con la utilización en el puerto serie (consola), y otros periféricos como pantallas LCD. Permiten trabajar con variables de tipo texto o con textos prefijados.

- **Constante de texto**: Define un texto de forma estática.

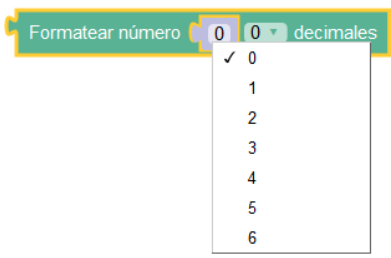


- **Formatear número**: Obtiene en forma de texto el valor de una variable o constante numérica en el formato especificado.

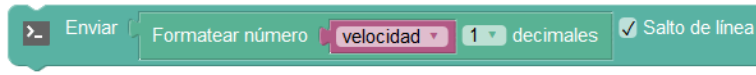
HEX	Genera el texto con la representación hexadecimal del valor.
DEC	Genera el texto con la representación decimal del valor.
BIN	Genera el texto con la representación binaria del valor.

Ejemplo:

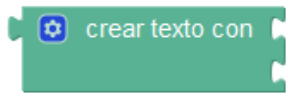
- **Formatear número con decimales:** Realiza la conversión de una variable o constante numérica a texto igual que el bloque anterior pero pudiendo indicar el número de decimales a mostrar.



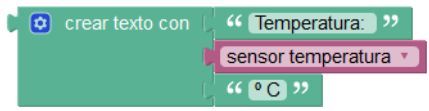
Ejemplo: enviar por la consola el valor de una variable como texto formateado con 6 y 1 decimales respectivamente:



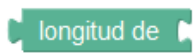
- **Crear texto con**: Crea un texto a partir de la unión de otros textos o variables. Las variables especificadas se convertirán a texto con formato decimal.



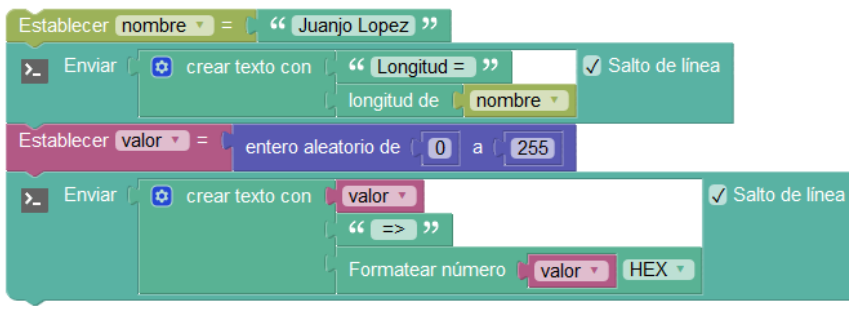
Ejemplo:



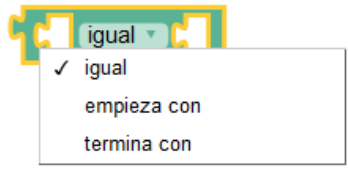
- **Longitud** : Obtiene el número de caracteres del texto.



Ejemplo de uso de bloques de texto:

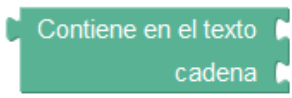


- **Comparación de textos** : Permite comparar dos cadenas de texto. El resultado es un valor lógico de verdadero o falso.

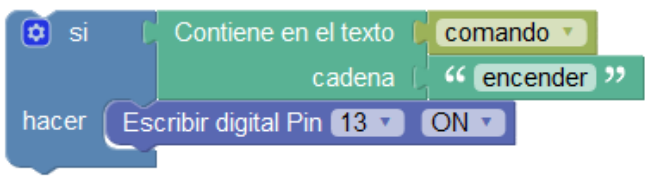


Ejemplo: comparación de texto y variables de tipo texto

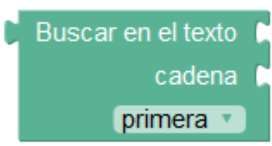
- **Contiene el texto:** Comprueba si existe un texto dentro del texto indicado. Devuelve verdadero si existe y falso en caso contrario.



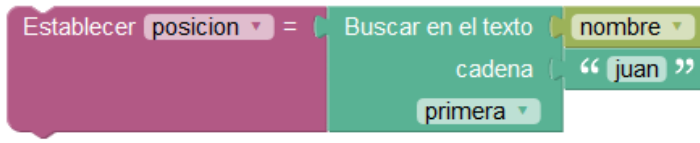
Ejemplo:



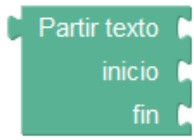
- **Buscar en el texto:** Busca la posición de un texto dentro de otro texto. Si el texto buscado no se encuentra devuelve el valor 0, en otro caso devuelve la posición donde empieza el texto.



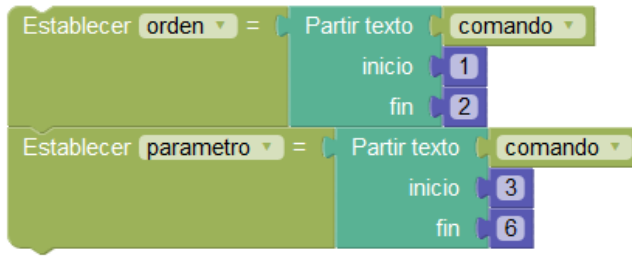
Ejemplo:



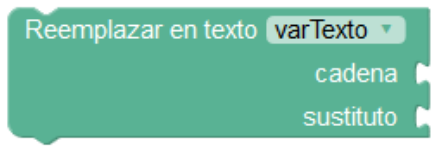
- **Partir texto:** Obtiene una parte del texto, indicando la posición de inicio y fin dentro del texto para crear la subcadena.



Ejemplo:



- **Reemplazar en texto:** Reemplaza todas las ocurrencias del texto indicado por el nuevo dentro de la variable de texto seleccionada.



Ejemplo:

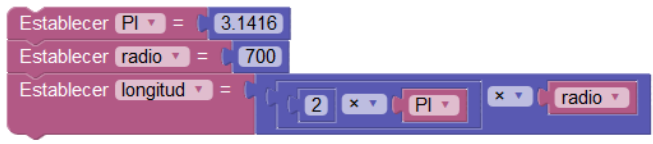
3.2.5 VARIABLES

Una variable es un hueco en la memoria donde el programa puede almacenar valores numéricos. El sistema nos permiten asignarles un nombre simbólico como por ejemplo "temperatura exterior", "velocidad", "posición servo 1","estado",... para facilitar su uso.

Hay tres tipos de variables en ArduinoBlocks: numéricas, booleanas y de texto.

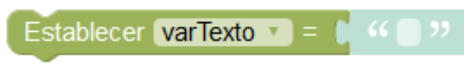
- **Variables numéricas**: permite valores numéricos enteros o con decimales, internamente se representan con el tipo de datos "doublé" a la hora de generar el código para Arduino. Este tipo utiliza 4 bytes y permite almacenar valores en el rango: -3.4028235E+38 a 3.4028235E+38

Ejemplo:



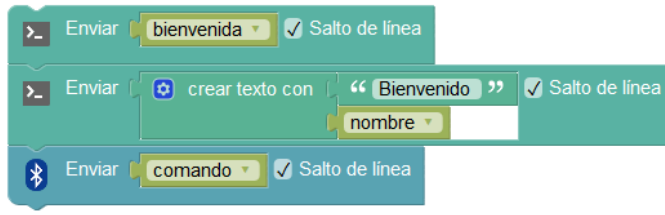
Ejemplo:

- **Variables de texto**: permite almacenar valores de texto. Internamente utiliza el tipo de dato "String" a la hora de generar el código para Arduino.

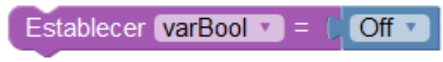


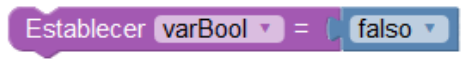
Ejemplo:

Ejemplo:



- **Variables booleanas**: permite almacenar valores lógicos booleanos de dos estados (verdadero/falso, ON/OFF, HIGH/LOW, ...)





Ejemplo:

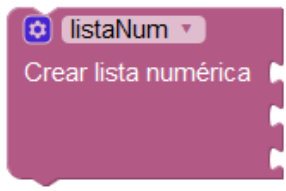
Ejemplo:

3.2.6 LISTAS

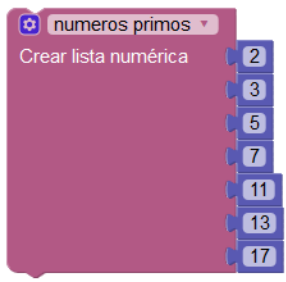
Las listas de datos nos permiten almacenar un listado de valores y acceder a ellos por su posición en la lista. Las listas pueden ser de tipo numéricas o de texto.

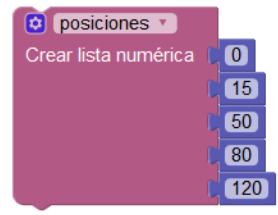
- **Listas numéricas:**

  Podemos crear una lista asignándole un nombre a la lista y asignándole valores iniciales.



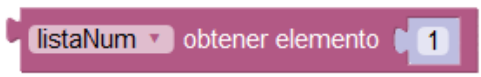
Ejemplo:



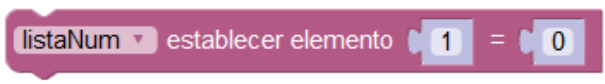


Para saber el número de elementos que tenemos en una lista podemos usar el bloque:

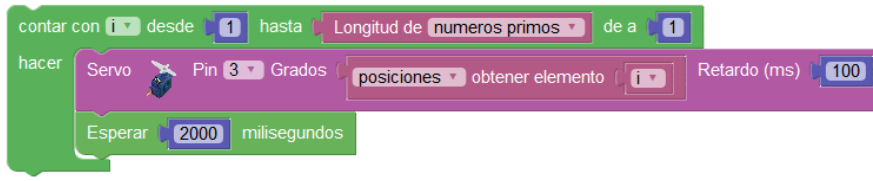
En una lista podemos obtener el valor de una posición (desde la 1 hasta el número de elementos en la lista) con el bloque:



O cambiar el valor de un elemento indicando su posición y el nuevo valor:

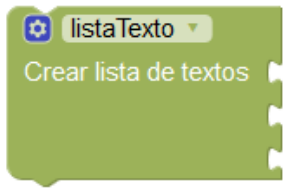


Ejemplo:

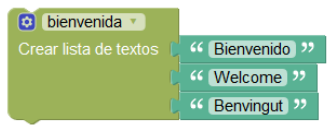


- **Listas de textos:**

  Podemos crear una lista asignándole un nombre a la lista y asignándole valores iniciales.



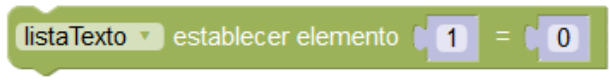
Ejemplo:



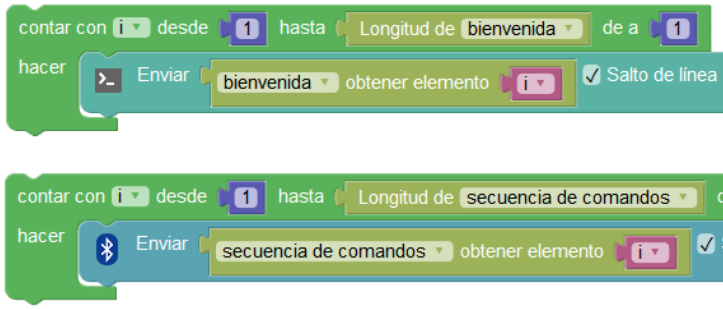
Para saber el número de elementos que tenemos en una lista podemos usar el bloque:

En una lista podemos obtener el valor de una posición (desde la 1 hasta el número de elementos en la lista) con el bloque:

O cambiar el valor de un elemento indicando su posición y el nuevo valor:



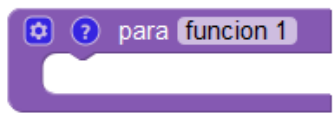
Ejemplo:



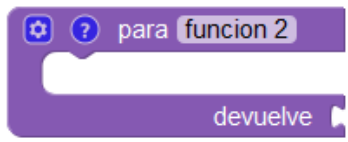
3.2.7 FUNCIONES

Las funciones permiten agrupar bloques de código. Esto es útil cuando un bloque de código se repite en varias partes del programa y así evitamos escribirlo varias veces o cuando queremos dividir el código de nuestro programa en bloques funcionales para realizar un programa más entendible.

- **Definición de una función**: La definición consiste en crear el grupo donde podremos insertar el código de bloques que forma la función. Debemos darle un nombre representativo que utilizaremos para llamar a esa función y ejecutarla.

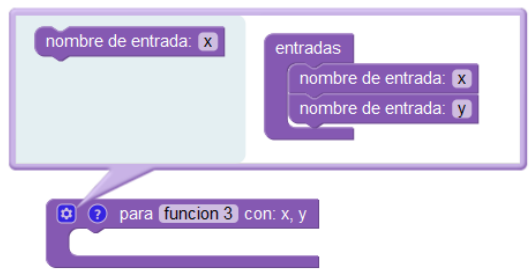


Función sin valor de retorno. La función ejecuta los bloques de su interior y vuelve al punto de llamada.

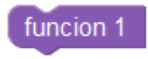


Función con valor de retorno. La función ejecuta los bloques de su interior y devuelve un resultado.

- **Parámetros:** A las funciones se les pueden añadir parámetros para especificar en la llamada.

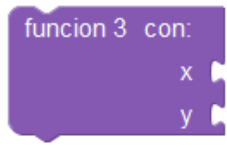


- **Llamada a una función:** Permite llamar a la ejecución de la función, se ejecutarán los bloques internos de la función y al terminar se seguirá la ejecución por donde se había realizado la llamada a la función.



Llamada a una función sin valor de retorno.

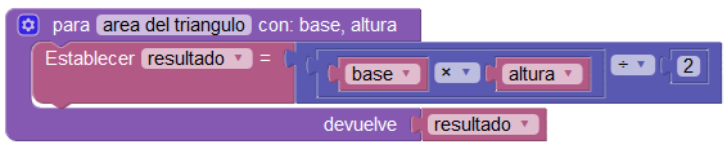
Llamada a una función con valor de retorno.



Llamada a una función sin valor de retorno y con 2 parámetros

Ejemplo: Función para calcular el área de un triángulo

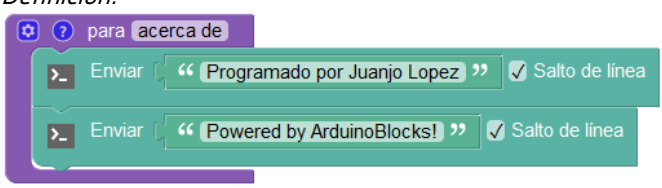
Definición:



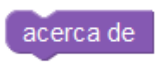
Llamada:

Ejemplo: Función para enviar información por la consola

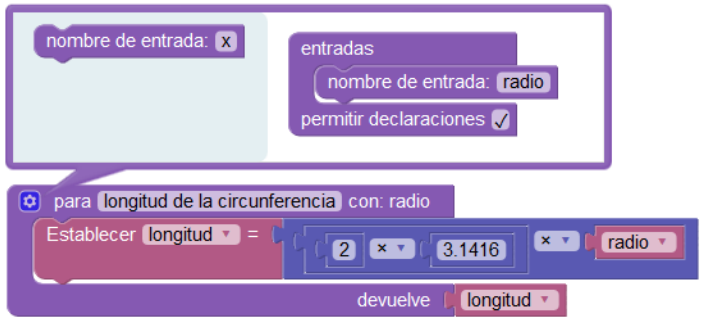
Definición:



Llamada:

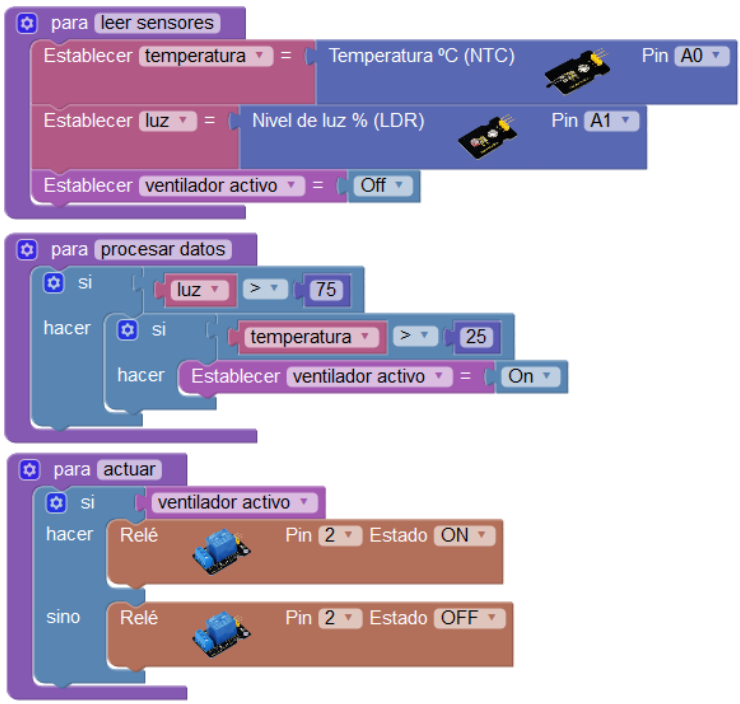


Ejemplo: Función para calcular la longitud de una circunferencia

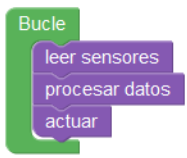


Ejemplo: División en partes funcionales de un programa real.

Definición:



Llamada desde el bucle principal del programa:



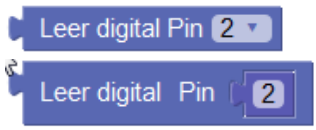
3.3 BLOQUES ARDUINO

En el siguiente apartado veremos los bloques relacionados con funciones propias de la placa Arduino. Estos bloques nos permitirán acceder a funcionalidades del propio microcontrolador y otros estarán orientados a sensores, actuadores o periféricos que podemos conectar a la placa Arduino para desarrollar nuestros proyectos.

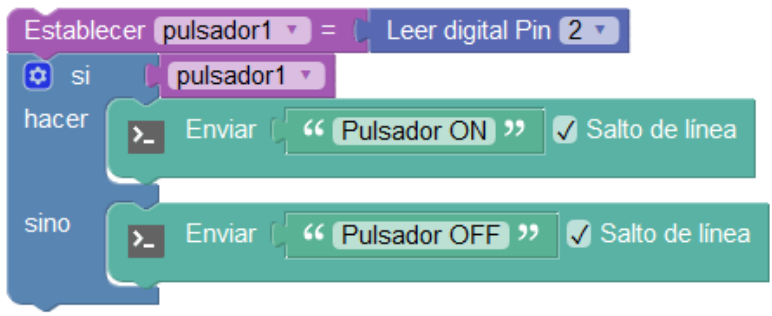
3.3.1 ENTRADA/SALIDA

Las funciones de entrada/salida genéricas nos permiten leer o escribir en los pines digitales y analógicos de la placa Arduino descritos en el apartado 2.3.

- **Leer pin digital**: Obtiene el valor digital del pin (0/1, ON/OFF, verdadero/falso). *(Recuerda para leer un ON/1 debemos aplicar 5v en la entrada digital y 0v para leer un OFF/0)*

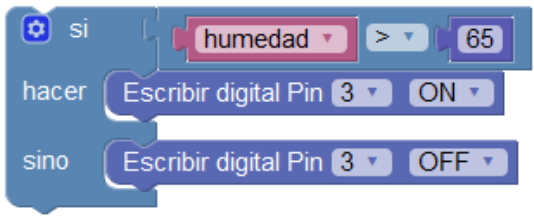


Ejemplo:



- **Escribir pin digital**: Escribe el valor en un pin digital pin (0/1, ON/OFF, verdadero/falso).
  (Si se activa, la salida suministrará 5v en caso contrario 0v)

Ejemplo:

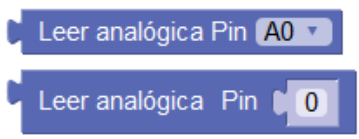


Versión equivalente:

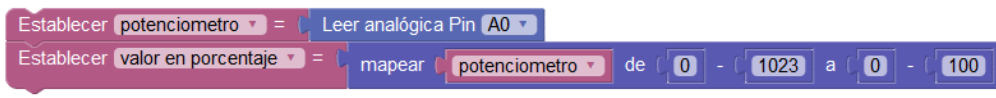
- **Leer pin analógico**: Lee el valor de una entrada analógica.
  El conversor interno DAC (Digital Analog Converter) es de 10 bits por lo que los valores leídos de una entrada analógica van de 0 a 1023

  10 bits = 2 ^ 10 = 1024 posibles valores

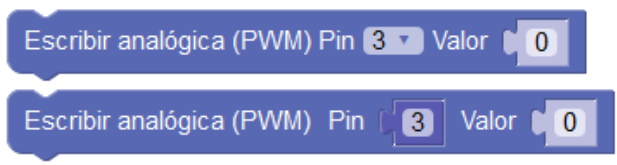
Voltaje en la entrada analógica	*Valor leído*
0 voltios	0
2.5 voltios	512
5 voltios	1023



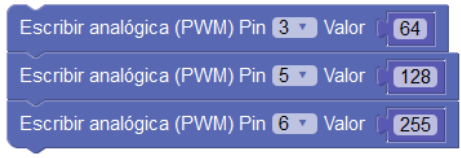
Ejemplo:



- **Escribir pin analógico**: Establece el valor del ciclo de pulsos activo/inactivo de una salida digital PWM. El valor debe estar en el rango entre 0 y 255.



Ejemplo: pin 3 al 25%, pin 5 al 50% , pin 6 al 100%



- **Leer pulso:** Lee un pulso en un pin hasta que el valor de la entrada cambie a estado alto (ON) o bajo (OFF). Mide la duración del pulso en microsegundos. Si se supera el tiempo de espera indicado sin cambiar de estado devolverá el valor 0.

3.3.2 TIEMPO

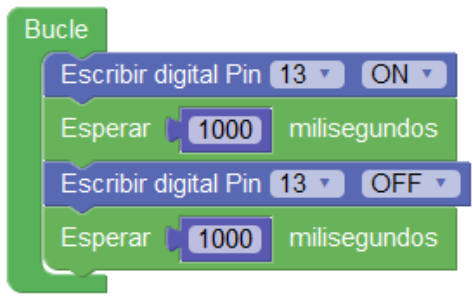
Las funciones de tiempo o retardo nos permiten realizar pausas y obtener información sobre el tiempo transcurrido dentro del microcontrolador.

- **Esperar**: Realiza una pausa (**bloquea** la ejecución del programa) hasta seguir con la ejecución del siguiente bloque.

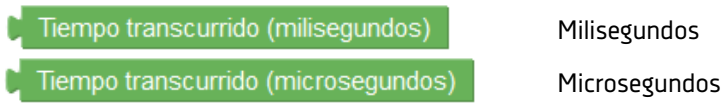
Milisegundos

Microsegundos

Ejemplo: Led 1 segundo encendido, 1 segundo apagado...

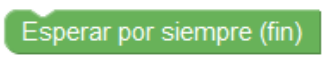


- **Tiempo transcurrido**: Obtiene un valor con el tiempo transcurrido desde el inicio o reset del microcontrolador de la placa Arduino. El valor puede ser en milisegundos o microsegundos.

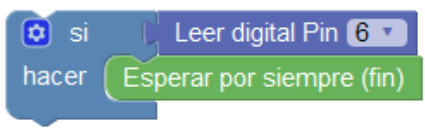


Ejemplo : Ejecutar la Tarea1 cada 3 segundos y la Tarea2 cada 7 segundos sin bloquear la ejecución del programa:

- **Esperar por siempre:** Bloquea indefinidamente la ejecución finalizando por tanto el programa.



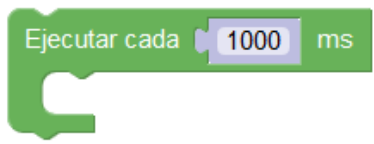
Ejemplo: al activar la entrada del pin 6 se finaliza la ejecución.



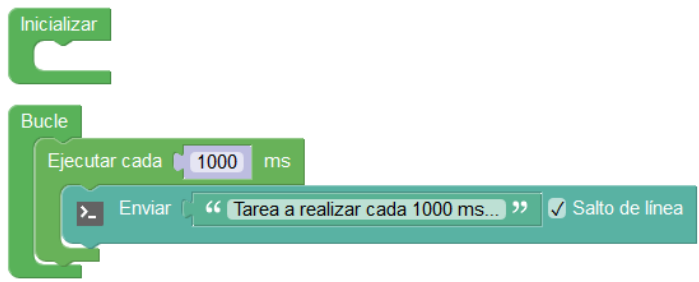
Ejemplo: Funcionamiento equivalente (esperar por siempre):

- **Ejecutar cada:** Bloque que implementa automáticamente la función de tareas explicada anteriormente.

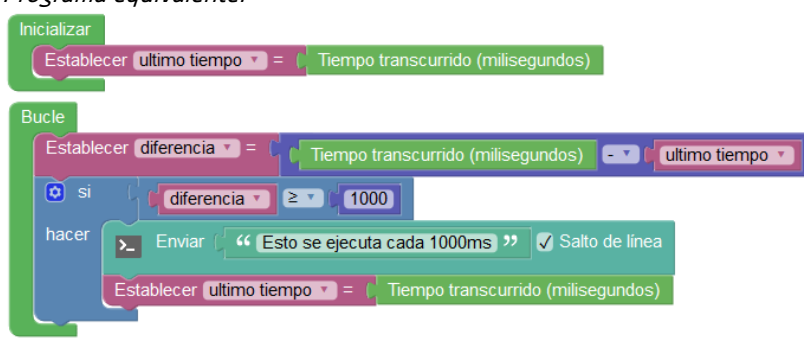
  IMPORTANTE: Este bloque **no bloquea la ejecución del programa**



Ejemplo: Ejecuta los bloques en su interior si el tiempo transcurrido desde la última ejecución es mayor o igual a 1000 ms



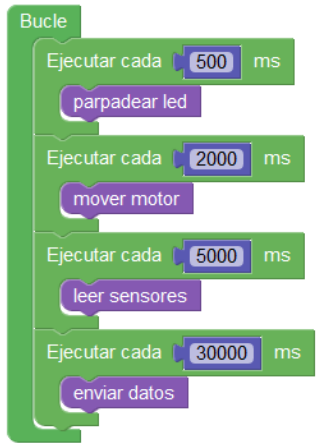
Programa equivalente:



Cuando necesitemos realizar distintas tareas periódicas y que parezca que se ejecuten paralelamente sin bloquearse unas a otras utilizaremos este tipo de bloque “ejecutar cada”.

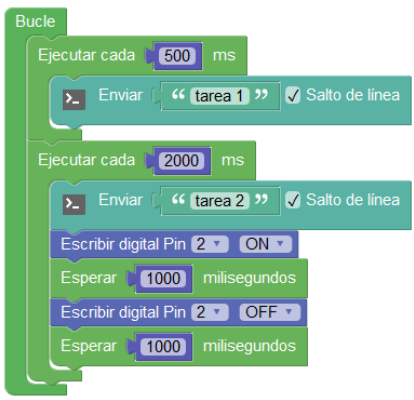
Si en el programa utilizamos bloques como por ejemplo el GPS (Apdo. 3.3.12) obligatoriamente debemos evitar los bloques de "*esperar*" si queremos que el programa funcione correctamente. **Consultar Anexo I** para ver los bloques incompatibles con bloques tipo "*esperar*"

Ejemplo: tareas simultáneas con distintos periodos de ejecución utilizando bloques "ejecutar cada"



La precisión de la ejecución de tareas de esta forma depende del tiempo que emplea cada tarea, si una tarea "*tarda*" mucho bloqueará y "*retrasará*" al resto. Para un funcionamiento correcto cada tarea debe ejecutarse en el menor tiempo posible y no usar nunca bloques de tipo esperar o realizar bucles de indeterminada duración que puedan quedarse en ejecución por tiempo indefinido.

*Ejemplos de lo que **NO** se debería hacer dentro de las tareas:*

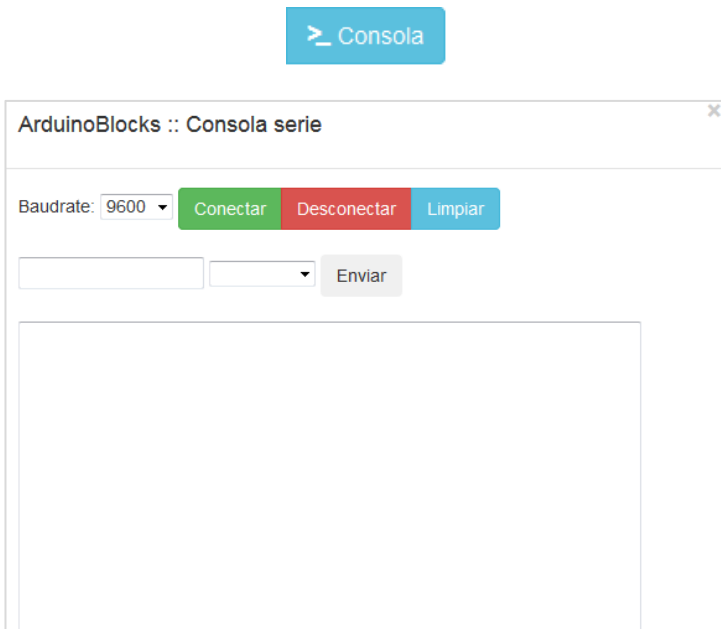


3.3.3 PUERTO SERIE

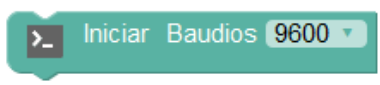
La comunicación vía puerto serie es muy utilizada. Es una vía de comunicación bidireccional sencilla que nos permite enviar información desde Arduino que visualizaremos en la consola o al contrario, enviar información desde la consola que recibiremos en el Arduino.

En muchas ocasiones simplemente se utiliza como una forma de depurar o mostrar información para saber si nuestro programa dentro del microcontrolador de Arduino está funcionando bien, en otros casos se puede utilizar de una forma más compleja sirviendo de vía de comunicación con aplicaciones en un PC, con periféricos como un GPS o comunicando con otros sistemas o por qué no, con otra placa Arduino.

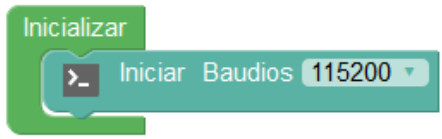
En ArduinoBlocks tenemos acceso a la consola via web (*con ArduinoBlocks-Connector* instalado) aunque podemos utilizar si lo preferimos cualquier aplicación de consola o terminal serie compatible con nuestro sistema.



- **Iniciar**: Configura la velocidad de la comunicación serie. Este valor debe ser igual en la consola y en el programa Arduino para establecer una comunicación correcta. Por defecto, y si no se pone nada, la velocidad es 9600bps.

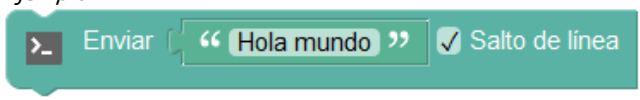


Ejemplo:



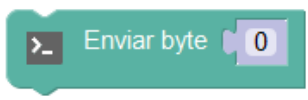
- **Enviar**: Escribe un valor de texto o el valor de una variable en el puerto serie. La opción "Salto de línea" permite añadir o no un retorno de carro al final del envío para bajar de línea.

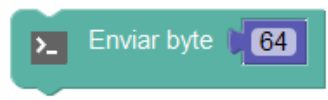
Ejemplo:



- **Enviar byte**: Envía un valor numérico como un byte (8 bits). Por tanto el valor debe estar comprendido entre 0 y 255.

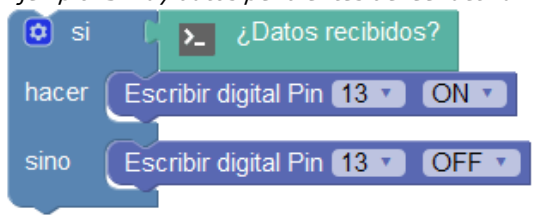
Ejemplo: Enviar byte con valor 64





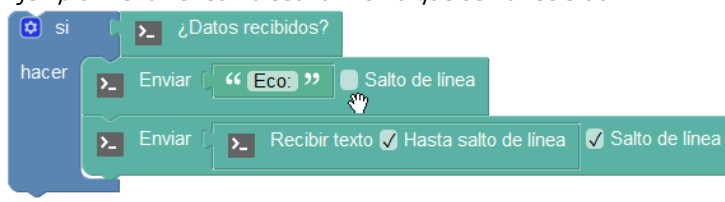
- **¿Datos recibidos?**: Obtiene un valor de verdadero si hay datos recibidos pendientes de procesar o falso si no se ha recibido nada por la conexión serie.

Ejemplo: Si hay datos pendientes de leer activar el pin 13



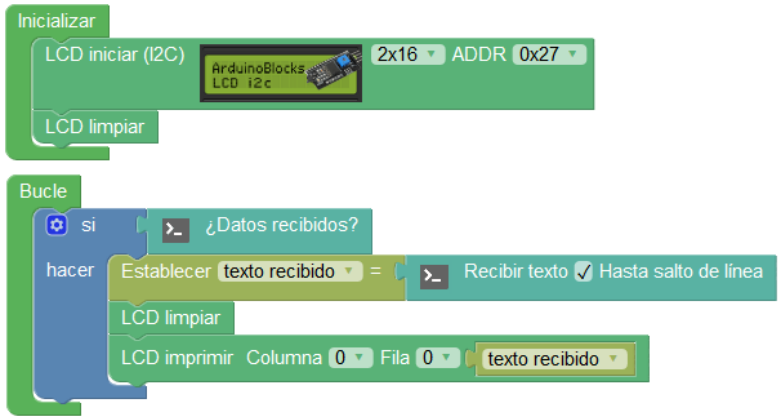
- **Recibir texto**: Lee una cadena de texto recibida por el puerto serie. Si se indica la opción "hasta salto de línea" en cuanto se encuentra un salto de línea devuelve el texto recibido. Si no, hasta que se dejen de recibir datos.

Ejemplo: Devolver como eco lo mismo que se ha recibido



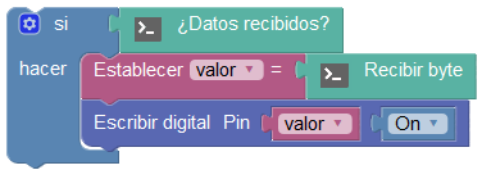
Ejemplo:

Ejemplo: Mostrar texto recibido por serie en una pantalla LCD

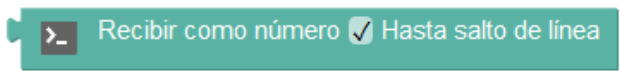


- **Recibir byte**: Leer un byte (8 bits) del puerto serie

Ejemplo: Activar el pin correspondiente al byte recibido



- **Recibir como número**: Leer una cadena de texto recibida por el puerto serie e intenta interpretarla como un número (analiza la cadena de texto buscando un formato numérico válido)



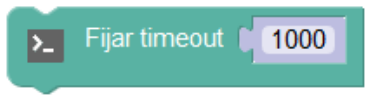
La opción "Hasta salto de línea" permite definir hasta donde se intentará interpretar los datos recibidos como un número. Normalmente las aplicaciones de terminal serie permiten enviar texto añadiendo automáticamente el salto de línea al final. Si no marcamos esta opción Arduino intentará interpretar también el salto de línea como un número dando error y obteniendo el valor 0.

Al pulsar "Enviar" en la consola serie, se envía a Arduino:
"1" + código de salto de línea (\n)

En este caso si no está activada la casilla "hasta salto de línea" leeremos el valor enviado "1" y luego un "0" (por error de intentar interpretar el salto de línea como un número)

Ejemplo: Recibe un número enviado como texto desde la consola. Interpreta el número. Si es "1" activa el pin 13 , si es "2" apaga el pin 13:

- **Fijar timeout:** Establece el tiempo máximo de espera en la recepción de datos (valor en milisegundos).



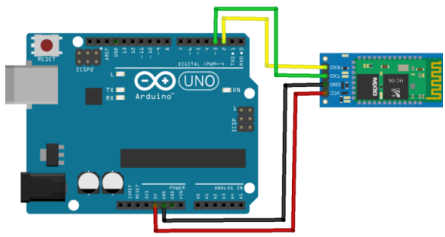
3.3.4 BLUETOOTH

La comunicación con el módulo Bluetooth HC-06 es exactamente igual que la del puerto serie, de hecho lo que hace el módulo Bluetooth es encapsular toda la información serie a través de una conexión serie virtual a través de un perfil Bluetooth de emulación de puerto serie.

Podemos simular una conexión serie con un dispositivo móvil (con Bluetooth compatible con el perfil de puerto serie), un PC u otro módulo Bluetooth similar en otro dispositivo.

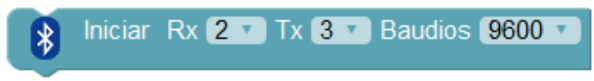
Arduino UNO sólo posee un puerto serie implementado en su hardware, para no utilizar el módulo Bluetooth en los pines 0 y 1 (correspondientes al puerto serie hardware) e interferir con la comunicación serie o la programación del dispositivo (como hacen otros entornos) los bloques de Bluetooth implementan un puerto serie software que funciona exactamente igual pudiendo configurarse en cualquier otro pin digital tanto para RX (recibir) como para TX (transmitir).

Ejemplo de conexión del módulo Bluetooth HC-06



BlueTerm Android

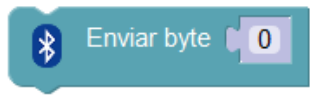
- **Iniciar**: Permite configurar los pines donde está conectado el módulo Bluetooth y la velocidad a la que vamos a trabajar.



- **Nombre**: El módulo Bluetooth HC-06 permite configurar el nombre y el código PIN a través de comandos. Con este bloque podemos hacerlo fácilmente, el único requisito para que funcione es que ningún dispositivo Bluetooth esté conectado en ese momento al módulo HC-06. Por otro lado normalmente es necesario reiniciar el módulo para que aparezca la nueva configuración (y desemparejar el dispositivo móvil si ya lo estaba).

- **Enviar**: Escribe un valor de texto o el valor de una variable en el puerto serie. La opción "Salto de línea" permite añadir o no un retorno de carro al final del envío para bajar de línea.

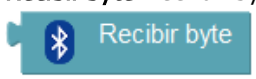
- **Enviar byte**: Envía un valor numérico como un byte (8 bits). Por tanto el valor debe estar comprendido entre 0 y 255.



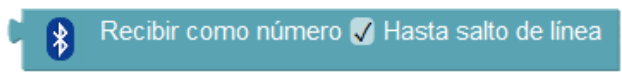
- **¿Datos recibidos?**: Obtiene un valor de verdadero si hay datos recibidos pendientes de procesar o falso si no se ha recibido nada por la conexión serie.

- **Recibir texto**: Lee una cadena de texto recibida por el puerto serie. Si se indica la opción "hasta salto de línea" en cuanto se encuentra un salto de línea devuelve el texto recibido. Si no, hasta que se dejen de recibir datos.

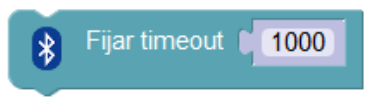
- **Recibir byte**: Leer un byte (8 bits) del puerto serie.



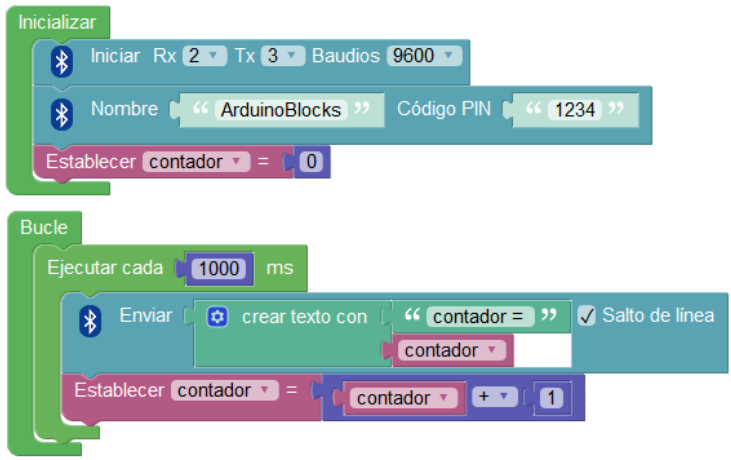
- **Recibir como número**: Leer una cadena de texto recibida por el puerto serie e intenta interpretarla como un número. Funciona igual que el boque del puerto serie (ver detalles de funcionamiento en el puerto serie)



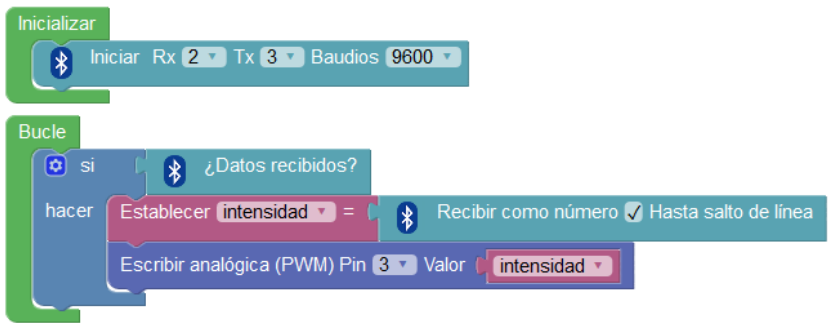
- **Fijar timeout:** Establece el tiempo máximo de espera en la recepción de datos por la conexión serie Bluetooth (valor en milisegundos)



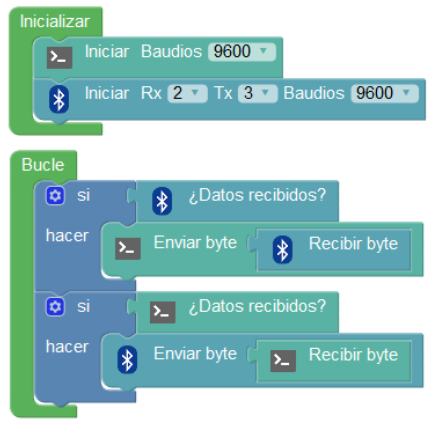
Ejemplo: Envío de una variable contador a través de Bluetooth



Ejemplo: Recepción de un valor por para establecer la intensidad de un led



Ejemplo: pasarela serie <-> Bluetooth



3.3.5 SENSORES

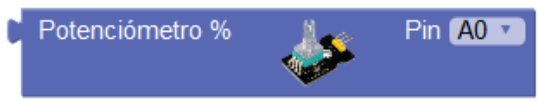
En el mercado existen infinidad de sensores y módulos para Arduino, aunque con los bloques genéricos descritos en el apartado 3.3.1 (entrada/salida) podemos leer la información de la mayoría de sensores digitales y analógicos ArduinoBlocks implementa bloques específicos para los sensores más comunes del mercado. Estos bloques a veces se limitan a leer la información digital o analógica, según el tipo de sensor, y en otros casos realizan una adaptación de los datos leídos para ajustarlos a la realidad (por ejemplo al leer un sensor de temperatura adapta la lectura a grados centígrados con un cálculo interno).

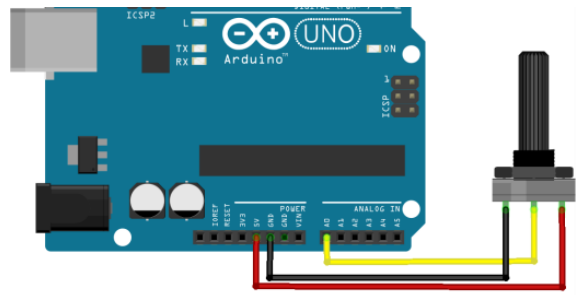
ArduinoBlocks incorpora bloques para la mayoría de sensores modulares que podemos encontrar en el mercado, algunos muy populares como los sensores de Keyestudio y similares.

ArduinoBlocks es una plataforma online en continua evolución por lo que seguramente desde la edición de este libro ya incorporará nuevos sensores con nuevas funcionalidades.

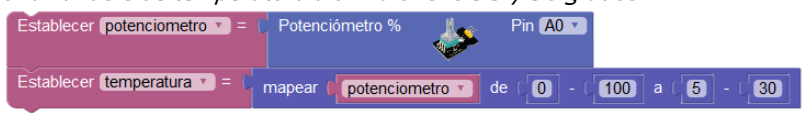
- **Sensor potenciómetro:** Nos permite obtener la posición del mando rotativo. Ángulo de operación de unos 270º. Varía el valor de voltaje aplicado a la entrada en función de la posición de su resistencia variable interna.

  Tipo: Analógico *Pin: A0-A5* *Valor: 0-100 (%)*



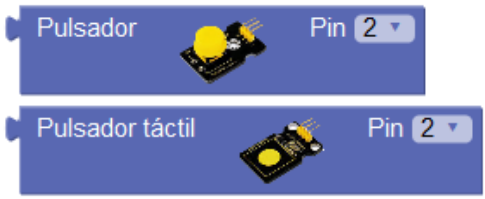


Ejemplo: Sensor potenciómetro conectado al pin analógico A0 para ajustar una variable de temperatura a un valor entre 5 y 30 grados.



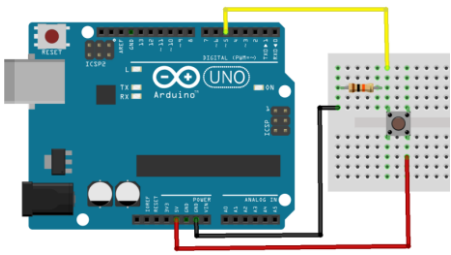
- **Sensor pulsador/pulsador táctil:** Botón para interactuar de forma táctil.

  Tipo: Digital *Pin: 2-13/A0-A5* *Valor: 0/1 (F/V, Off/On)*

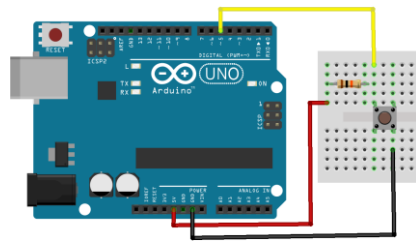


Dependiendo de la conexión que hagamos del pulsador, o en caso de utilizar módulos de pulsador de diferentes fabricantes, la lógica de funcionamiento del pulsador puede ser diferente:

Conexión:
sin presionar "off" / presionado: "on"



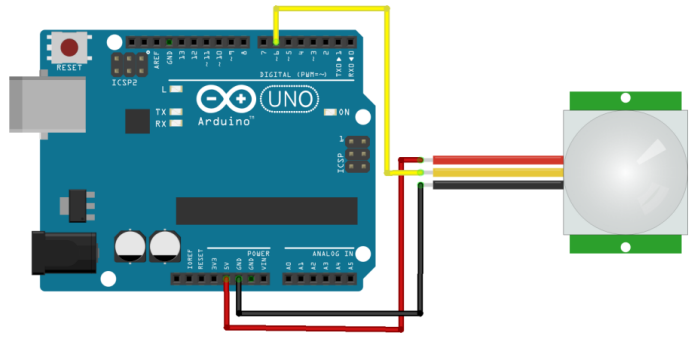
Conexión:
sin presionar "on" / presionado: "off"



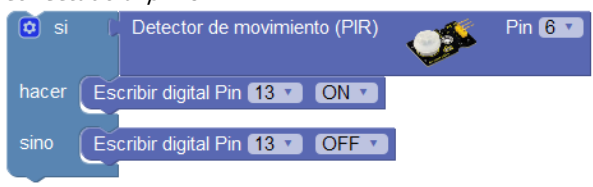
Alguno módulos de pulsador internamente trabajan de forma inversa por su conexión interna. En ese caso el pulsador siempre está dado una señal "On" y cuando lo pulsamos genera la señal "Off". En ese caso podemos invertir la condición para detectar cuando está pulsado:

- **Sensor de movimiento (PIR):** Se activa cuando detecta movimiento a su alrededor, a partir de un tiempo sin detección el sensor vuelve a desactivarse.

  Tipo: Digital *Pin: 2-13/A0-A5* *Valor: 0/1 (F/V, Off/On)*

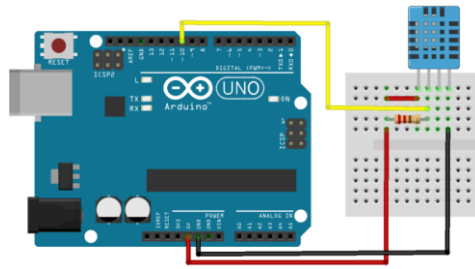


Ejemplo: Encendido del led del pin 13 al detectar movimiento. Sensor PIR conectado al pin 6:

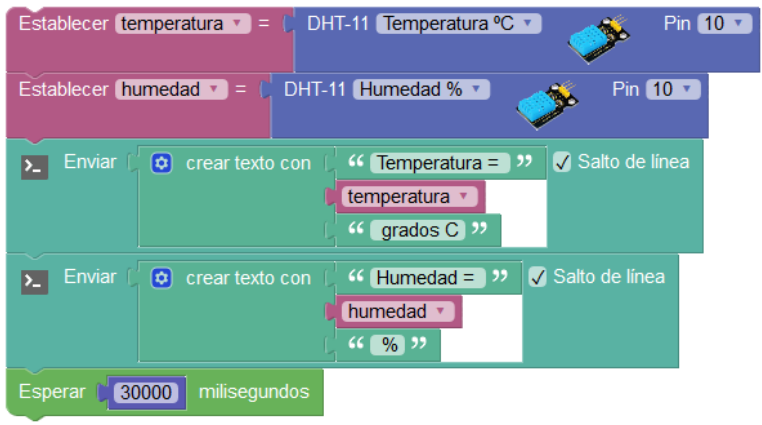


- **Sensor de temperatura y humedad (DHT-11):** El sensor DHT-11 es un sensor que utiliza un protocolo de comunicación propio para facilitarnos el valor de temperatura y humedad ambiente. ArduinoBlocks internamente utiliza una librería para obtener la información decodificada del sensor.
  Es un sensor de baja precisión pero muy económico y versátil. En un único pin nos permite obtener dos valores con una precisión suficiente para muchas aplicaciones sencillas.

  Tipo: Datos
  Pin: 2-13/A0-A5
  Valor: Temperatura: 0-50ºC ±2ºC / Humedad: 20-90% ±5%



Ejemplo: Mostrar por la consola cada 30 segundos el valor de temperatura y humedad. Sensor conectado al pin 10.

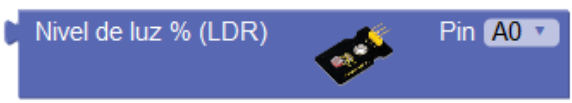


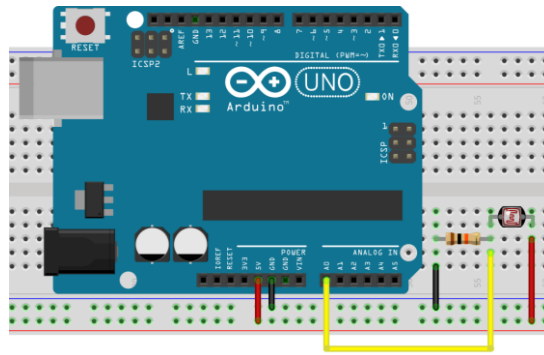
- **Sensor de temperatura y humedad (DHT-22):** El sensor DHT-22 es una versión mejorada del sensor DHT-11 con mayor rango de medida y precisión.

  Tipo: Datos
  Pin: 2-13/A0-A5
  Valor: Temperatura: -40º - 125ºC ±0.5ºC / Humedad: 0-100% ±2%

- **Sensor de luz (LDR):** Obtiene el nivel de luz ambiente mediante la resistencia LDR que varía en función de la luz ambiente aplicada.

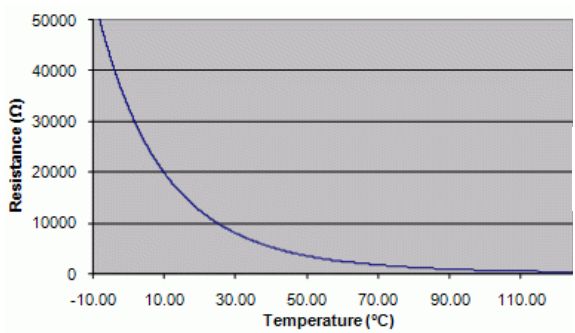
  Tipo: Analógico *Pin: A0-A5* *Valor: 0-100 (%)*





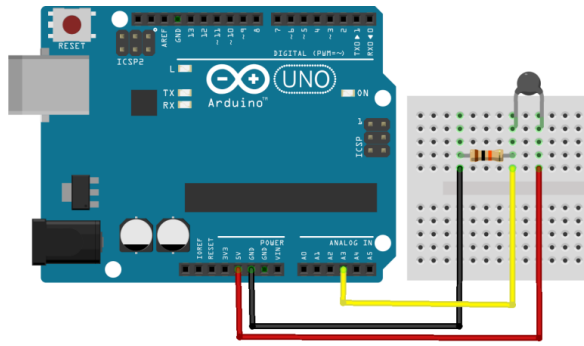
Ejemplo: Encendido de un led cuando el nivel de luz es inferior al 25%

- **Sensor de temperatura (NTC):** Obtiene el valor de la temperatura ambiente. Utiliza una resistencia variable NTC que varía su valor en función de la temperatura ambiente. La relación resistencia/temperatura no es lineal, pero internamente se calcula el valor en grados aplicando la siguiente fórmula para obtener el valor corregido en ºC:



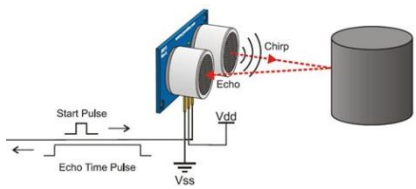
$$T_C = \frac{1}{\frac{1}{B}\ln(\frac{R}{R_N}) + \frac{1}{T_N + 273}} - 273$$

Tipo: Analógico *Pin: A0-A5* *Valor:* -40...125ºC



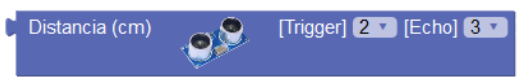
Ejemplo: Mostrar temperatura por la consola serie cada 5 segundos. Sensor conectado al pin A3:

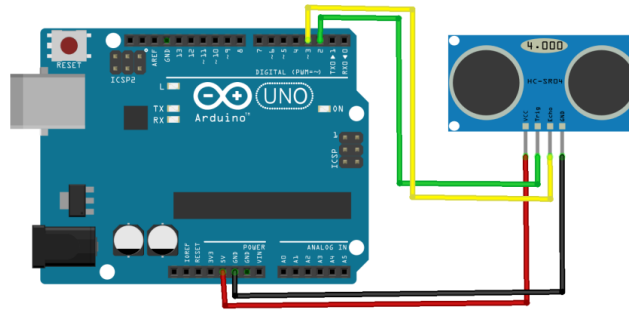
- **Sensor de distancia (HC-SR04):** El sensor genera una serie de tonos de ultrasonidos (no audibles), estos tonos si rebotan en una superficie vuelven y son captados por un receptor de ultrasonidos que incorpora el propio sensor. Midiendo el tiempo que tardan en volver los ultrasonidos podemos calcular la distancia a la que se encuentra el objeto sobre el que han rebotado.



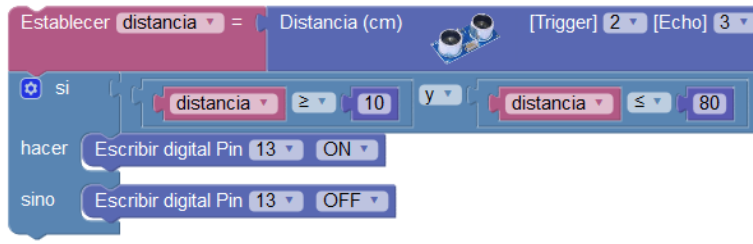
$$d_{sensor} = \frac{\Delta t \cdot 340\frac{m}{s}}{2}$$

Tipo: Datos
Pin Trigger (emisión): 2-13/A0-A5
Pin Echo (recepción): 2-13/A0-A5
Valor: 2 – 400 cm





Ejemplo: Activación del led en el pin 13 cuando se detecta un objeto entre 10 y 80 cm de distancia

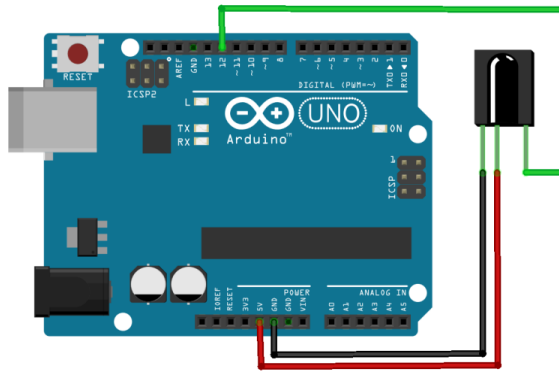


- **Sensor receptor de infrarrojos:** Permite decodificar los protocolos de señales de pulsos infrarrojos utilizados por los mandos a distancia.

  Protocolos detectados: RC5, RC6, NEC, SONY, PANASONIC,JVC, SAMSUNG, WHYNTER, AIWA, LG, SANYO, MITSUBISHI, DENON.

  Tipo: Datos
  Pin: 11

  Valor: código recibido / 0 = ningún código detectado.



Dependiendo el tipo de mando recibiremos unos códigos con valores de un tamaño u otro. Algunos mandos utilizan códigos de 32 bits, al almacenar el valor del código recibido en una variable de ArduinoBlocks se convierte a un valor decimal de 32 bits con signo y eso puede producir una alteración en el valor mostrado (número decimales extraños).

Para evitar este problema podemos tratar el valor como un valor entero sin signo de 32 bits añadiendo el bloque "Número entero sin signo" visto en el apartado de bloques matemáticos (3.2.3).

Ejemplo: Mostrar por consola el código recibido:

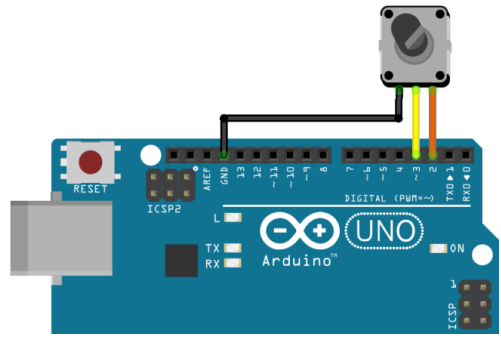
- **Sensor encoder (codificador) rotativo:** Un encoder rotativo es un elemento que indica su posición mediante posiciones codificadas. Cuando pasamos por cada paso se nota un pequeño salto que indica que se ha llegado a la nueva posición. Estos codificadores constan de dos pines de señal para el codificador y un pin para un pulsador que lleva integrado. Los dos pines del codificador nos dan la información en forma digital con un total de 4 combinaciones: 00, 01, 10, 11.

  El encoder no tiene ninguna posición predefinida y no tiene límite de giro en ningún sentido. Automáticamente mantiene un valor interno con la posición virtual según los pasos en un sentido u otro, empezando siempre en 0.

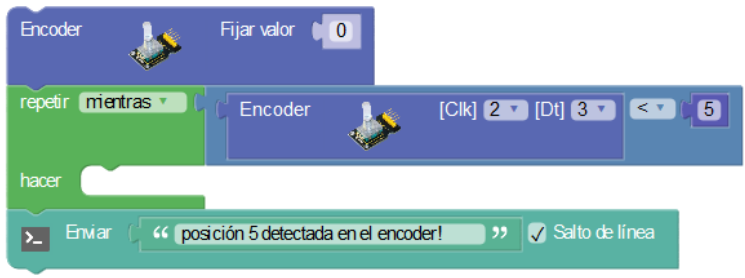
Tipo: Datos
Pin: Clk (A): 2 / Dt (A): 3
Valor: posición virtual del encoder (variable interna)

Obtener la posición "virtual" actual del encoder

Fijar el valor de posición "virtual" a un valor.

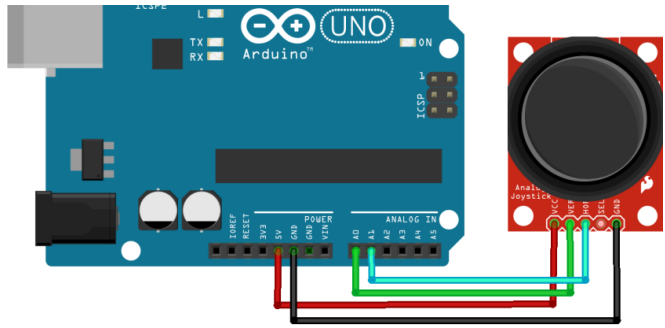


Ejemplo: Esperar 5 saltos hacia la derecha del encoder

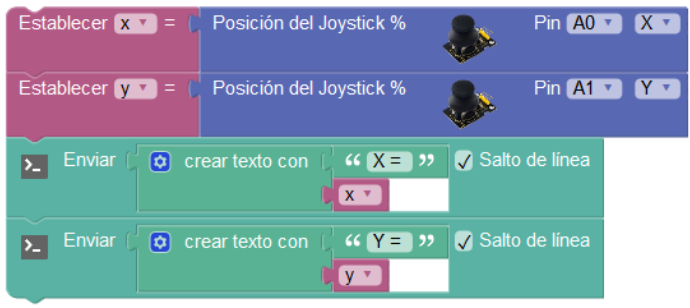


- **Sensor de joystick:** Este tipo de sensor de palanca se basa en dos potenciómetros que detectan la posición en cada uno de los ejes X e Y.

Tipo: Analógico
Pin: A0-A5
Valor: 0-100 % posición X o Y

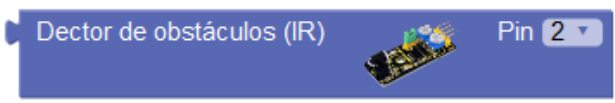


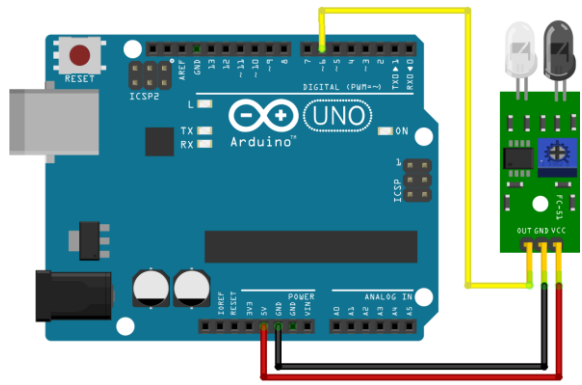
Ejemplo: Mostrar en la consola la posición X e Y:



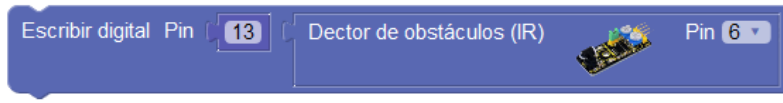
- **Sensor detector de obstáculos (IR):** Mediante el uso de un diodo emisor de IR y un fototransistor receptor de IR permite detectar cuando hay un obstáculo cerca por el reflejo de la luz IR. Este tipo de sensor permite normalmente un ajuste para definir la distancia a la que se activa el sensor.

Tipo: Digital *Pin: 2-13/A0-A5* *Valor: 0/1 (F/V, Off/On)*



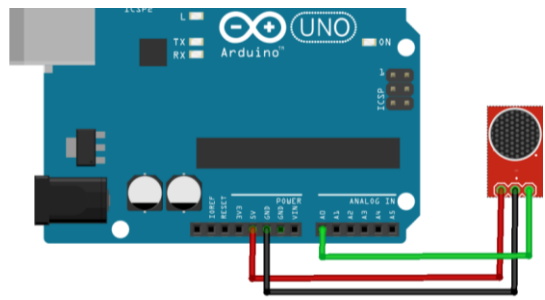


Ejemplo: Encendido del led en el pin 13 cuando se detecta un objeto cerca. Sensor conectado al pin 6

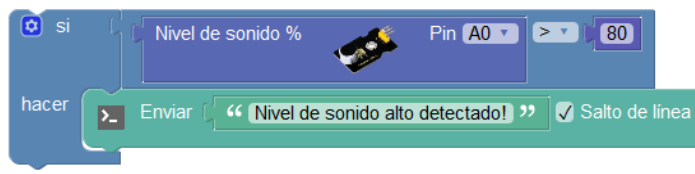


- **Sensor de nivel de sonido:** Detecta el nivel de sonido ambiente.

Tipo: Analógico *Pin: A0-A5/A0-A5* *Valor: 0-100 (%)*



Ejemplo - detector de nivel de sonido alto. Sensor en pin A0



- **Sensor sigue líneas:** Su funcionamiento es idéntico al detector de obstáculos por IR. Se utiliza para detectar superficies blancas/negras.

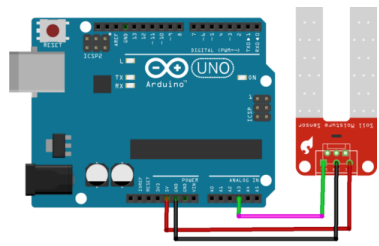
  Tipo: Digital *Pin: 2-13/A0-A5* *Valor: 0/1 (F/V, Off/On)*

- **Sensor foto-interruptor:** Detecta cuando un objeto interrumpe un haz de luz entre un emisor y receptor.

  Tipo: Digital *Pin: 2-13/A0-A5* *Valor: 0/1 (F/V, Off/On)*

- **Sensor sonda de humedad:** Mide la humedad con la ayuda de una sonda que se introduce en la tierra.

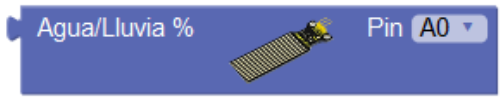
  Tipo: Analógico *Pin: A0-A5/A0-A5* *Valor: 0-100%*



Ejemplo: Activación del led conectado al pin 13 en caso de detectar un nivel de humedad inferior al 20%. Sensor conectado en el pin A3

- **Sensor de lluvia/agua:** Mide el nivel de agua o lluvia.

  Tipo: Analógico *Pin: A0-A5* *Valor: 0-100%*

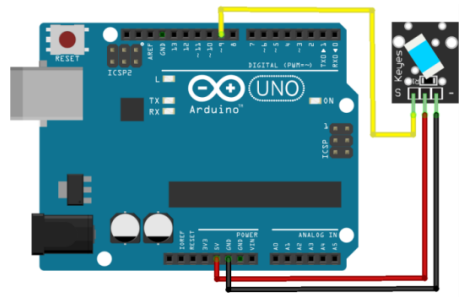


- **Sensor de golpe:** Detecta un impacto o golpe.

  Tipo: Digital *Pin: 2-13/A0-A5* *Valor: 0/1 (F/V, Off/On)*

- **Sensor de orientación:** Detecta si la orientación es vertical / horizontal.

  Tipo: Digital *Pin: 2-13/A0-A5* *Valor: 0/1 (F/V, Off/On)*



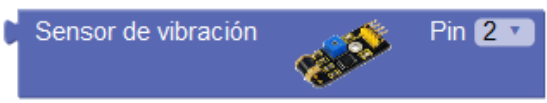
Ejemplo: Encender el led conectado al pin 13 cuando el sensor está inclinado. Sensor conectado al pin 9.

- **Sensor de campo magnético:** El sensor se activa con la presencia de un campo magnético cerca.

  Tipo: Digital *Pin: 2-13/A0-A5* *Valor: 0/1 (F/V, Off/On)*

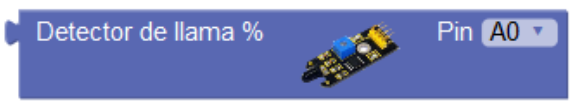
- **Sensor de vibración:** Se activa cuando detecta una vibración.

  Tipo: Digital *Pin: 2-13/A0-A5* *Valor: 0/1 (F/V, Off/On)*



- **Sensor detector de llama:** Detecta el nivel de fuego o una llama detectando la frecuencia de luz del fuego.

  Tipo: Analógico *Pin: A0-A5/A0-A5* *Valor: 0-100%*



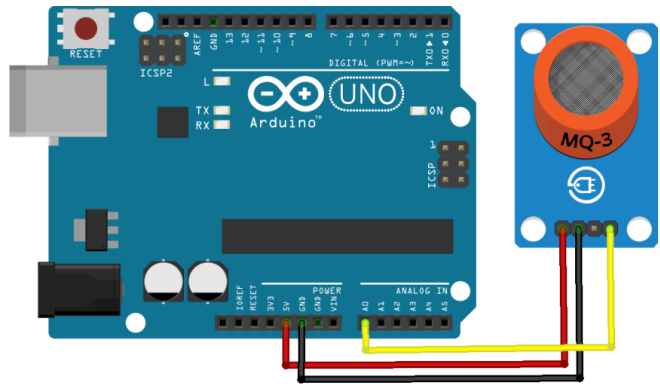
- **Sensor de nivel de gas:** Detecta el nivel de gas en el ambiente.

  Tipo: Analógico *Pin: A0-A5/A0-A5* *Valor: 0-100%*

Existen varias versiones del sensor que detectan diferentes gases:
MQ-2: Gases combustibles
MQ-4: Gas natural y metano
MQ-8: Gas hidrógeno
MQ-7: Gas monóxido de carbono
MQ-135: Sensor de calidad del aire

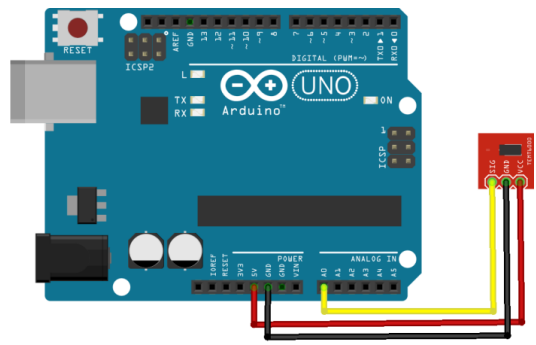
- **Sensor de nivel de alcohol:** Es una variación del sensor de gas (MQ-3) que mide el nivel de alcohol.

Tipo: Analógico *Pin: A0-A5* *Valor: 0-100%*



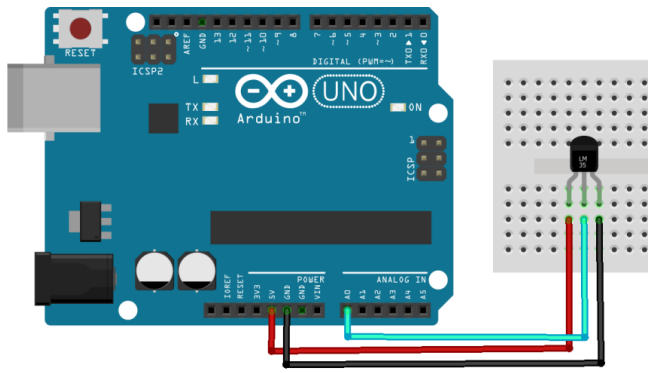
- **Sensor de nivel de luz (TEMT6000):** Permite medir la luz ambiente con alta precisión además de ser un sensor que mide sólo la luz ambiente que percibe el ojo humano, filtrando el espectro de luz no visible y realizando así una medición más real para ciertas aplicaciones.

Tipo: Analógico *Pin: A0-A5* *Valor: 0-100%*



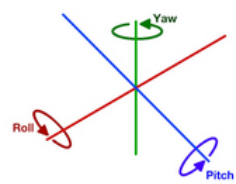
- **Sensor de temperatura (LM35):** Sensor de temperatura calibrado con precisión de 1ºC. La salida es lineal y cada ºC equivale a 10mV. El rango de medida es de -55ºC hasta 150ºC.

  Tipo: Analógico *Pin: A0-A5* *Valor: -55 ... 150 ºC*

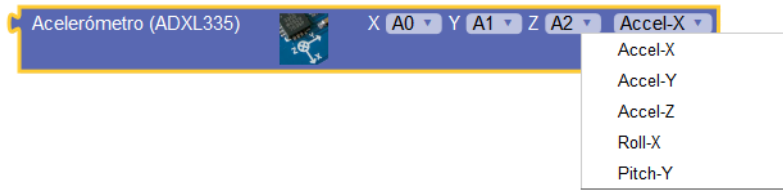


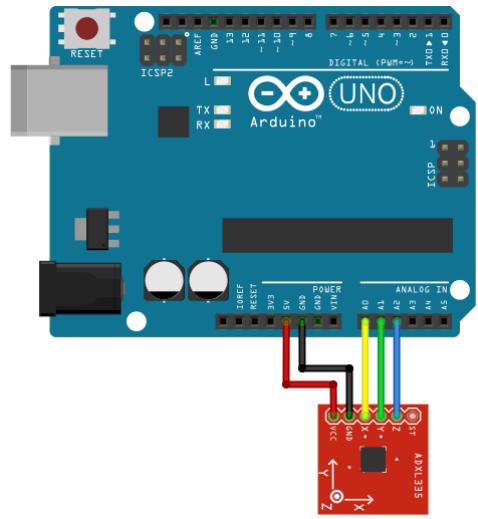
- **Sensor acelerómetro (ADXL335):** Este tipo de sensores permite medir la aceleración en los tres ejes espaciales X,Y y Z. En reposo los ejes X e Y deben tener una valor aproximo de 0G y el eje Z debe tener un valor aproximadamente de 1G (la aceleración de la gravedad). Ante cualquier sacudida o movimiento el sensor nos indicará la aceleración en cada eje en unidades G. Este sensor permite medir desde -3G hasta 3G.

Internamente con cálculos trigonométricos se pueden obtener los ángulos de rotación en X (Roll) y en Y (Pitch). Sin embargo para calcular la rotación en Z (Yaw) debemos utilizar otro tipo de acelerómetros que se complementan con un giroscopio.

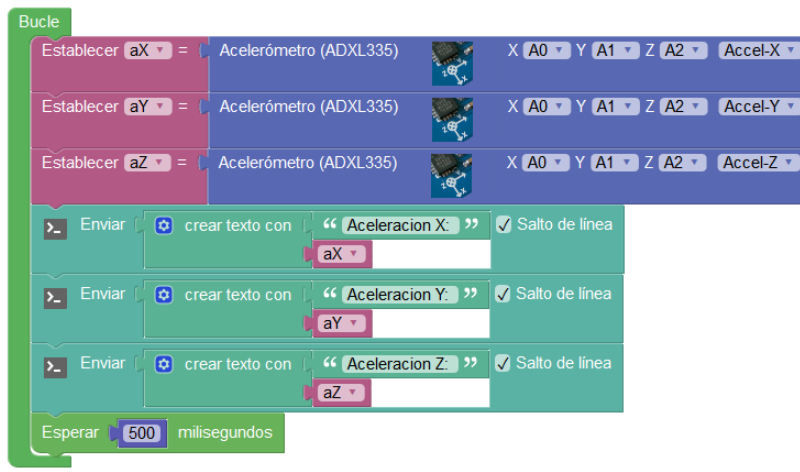


Tipo: Analógico *Pin: A0-A5* *Valor: -3G ... 3G*





Ejemplo: Mostrar los valores de aceleración por consola serie



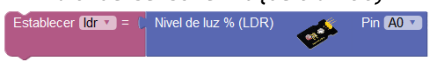
USO DE OTROS SENSORES

ArduinoBlocks implementa los sensores vistos anteriormente para simplificar el uso. En algunos casos procesa los datos leídos para obtener un valor (por ejemplo el sensor de temperatura NTC o LM35) y en otros casos "normaliza" el valor a un rango de % (0 a 100) para simplificar su uso.

En algunos casos puede que necesitemos leer el valor del sensor directamente para obtener una mayor precisión o simplemente porque el sensor no está implementado en ArduinoBlocks y necesitamos usar los bloques de entrada/salida genéricos para obtener datos del sensor digital o analógico.

Ejemplo 1: Lectura del valor de luz con LDR para obtener más precisión.

Sensor de luz:
valor del sensor en % (de 0 a 100)

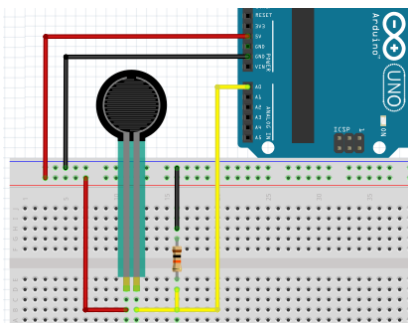


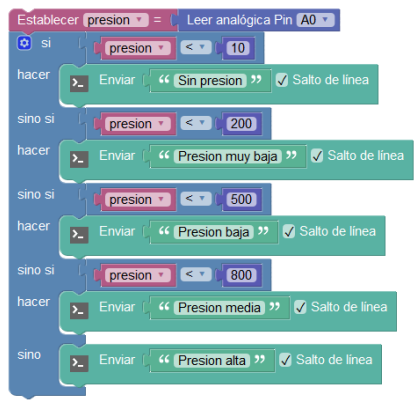
Valor del sensor directamente de la entrada analógica (valor de 0 a 1023)

Ejemplo 2: sensor de presión analógico.

Este sensor modificar su resistencia en función de la presión que se ejerce sobre él y por tanto conectado a una entrada analógica variará el voltaje leído en ella. La fuerza aplicada se traduce en un valor analógico leído de 0 a 1023. En estos casos debemos consultar las especificaciones del fabricante para interpretar el dato obtenido.

Force (lb)	Force (N)	FSR Resistance	(FSR + R) ohm	Current thru FSR+R	Voltage across R
None	None	Infinite	Infinite!	0 mA	0V
0.04 lb	0.2 N	30 Kohm	40 Kohm	0.13 mA	1.3 V
0.22 lb	1 N	6 Kohm	16 Kohm	0.31 mA	3.1 V
2.2 lb	10 N	1 Kohm	11 Kohm	0.45 mA	4.5 V
22 lb	100 N	250 ohm	10.25 Kohm	0.49 mA	4.9 V

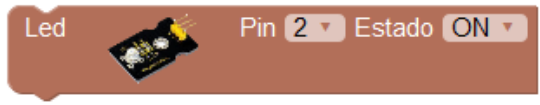




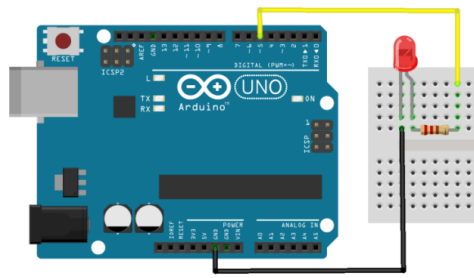
3.3.6 ACTUADORES

- **Led:** Permite controlar el encendido/apagado de un led (diodo emisor de luz).

  Tipo: Digital *Pin: 2-13* *Valor: 0/1 (F/V, Off/On)*



  Ejemplo: Parpadeo de un led cada segundo. Led conectado al pin 5:



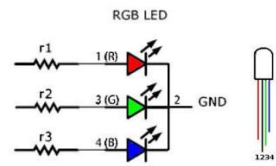
- **Led intensidad (PWM):** Permite controlar la intensidad de iluminación de un led conectado a una salida PWM.

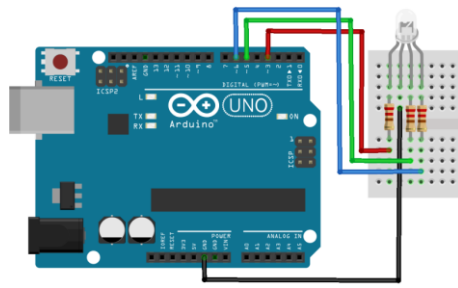
  Tipo: PWM *Pin: 3,5,6,9,10,11* *Valor: 0-255*

Ejemplo: Aumento progresivo de la intensidad del led:

- **Led RGB:** Controla un led RGB. Define un color calculando automáticamente los valores de cada componente R,G y B para definir el color.

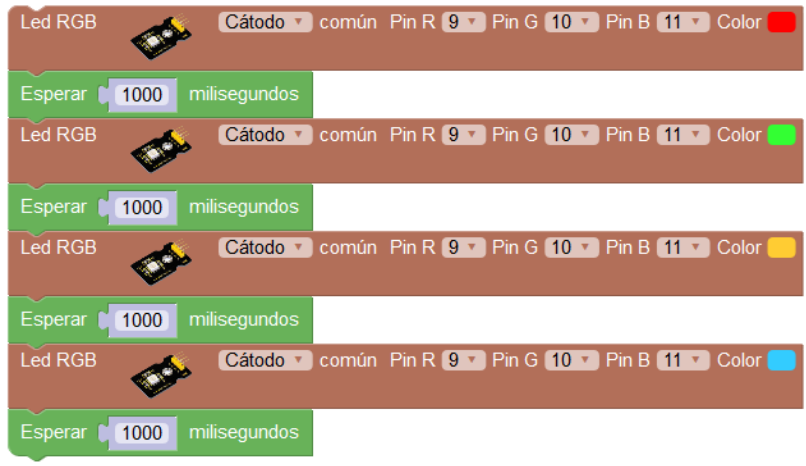
Tipo: PWM
Pin R/G/B: 3,5,6,9,10,11
Valor: Color





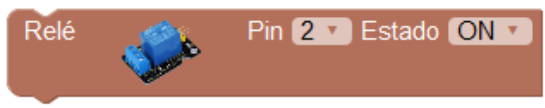
Si utilizamos un led RGB de ánodo común (+), el funcionamiento es a la inversa. Indicándolo en el bloque automáticamente los valores se invierten para lograr el color seleccionado.

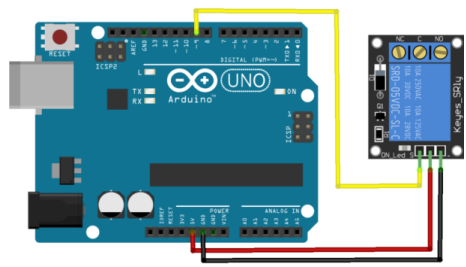
Ejemplo – Cambio de colores



- **Relé:** Controla la activación de un relé.

Tipo: Digital *Pin: 2-13* *Valor: 0/1 (F/V, Off/On)*

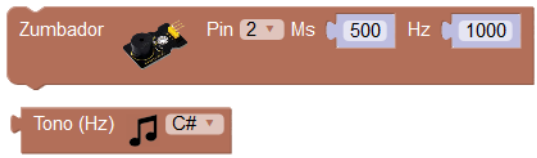




Ejemplo: Activación/desactivación de un relé cada 5 segundos:

- **Zumbador (pasivo):** Un zumbador pasivo es un dispositivo piezoeléctrico que permite generar sonidos. Podemos generar tonos de la frecuencia deseada. (Si utilizamos un zumbador activo el propio zumbador genera su frecuencia y lo podremos activar o desactivar con una simple salida digital).

Tipo: PWM
Pin R/G/B: 3,5,6,9,10,11
Ms: Duración del tono en milisegundos
Hz: Frecuencia del tono



Ejemplo: Tonos de medio segundo. Zumbador conectado al pin 3:

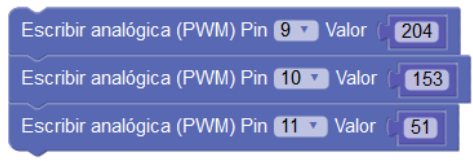
USO DE OTROS ACTUADORES

Al igual que con los sensores, puede que tengamos actuadores digitales o analógicos no implementados en ArduinoBlocks o que queramos controlar de una forma diferente a como lo hacen los bloques. Para ello podemos utilizar directamente los bloques de entrada/salida genéricos.

Ejemplo 1: Control de un led RGB con salidas analógicas PWM:

Con el bloque específico:

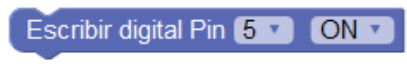
Con los bloques genéricos (Naranja => Rojo = 204, Verde=153, Azul=51)



Ejemplo 2: Control de un relé con salida digital

Con bloque específico:

Con bloque genérico



Ejemplo 3: Control de un led con salida digital

Con bloque específico:

Con bloque genérico

3.3.7 PANTALLA LCD

Uno de los periféricos más utilizados que podemos conectar a Arduino es una pantalla LCD (display) para mostrar información e interactuar con el usuario.

La pantalla LCD más sencilla y utilizada es el de tipo alfanumérico de 2 líneas y 16 caracteres por línea, o de 4 líneas y 20 caracteres por línea

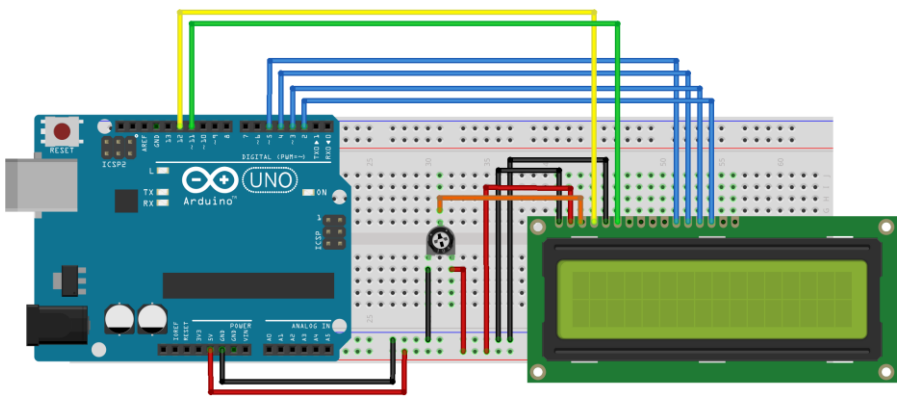
LCD 2x16

LCD 4x20

ArduinoBlocks nos permite conectar una pantalla de dos forma diferentes:

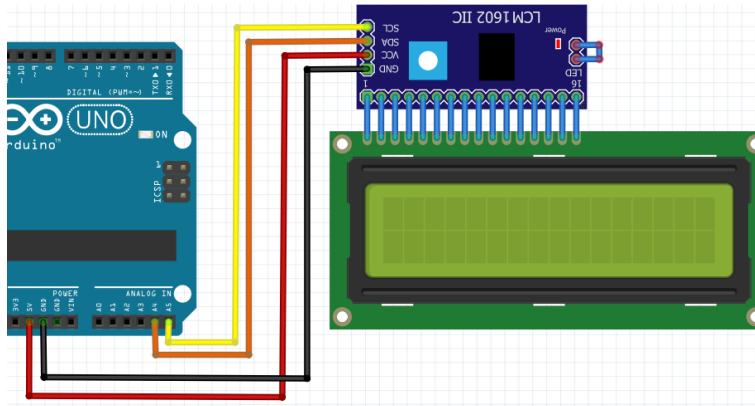
- **Conexión con bus de 4 bits + control EN / RS:**

Necesitamos 4 bits para datos y dos señales de control En (Enable) y Rs (Register select). La conexión RW la conectamos fija a GND. Además se debe añadir una resistencia ajustable o un potenciómetro para regular el contraste de la pantalla.



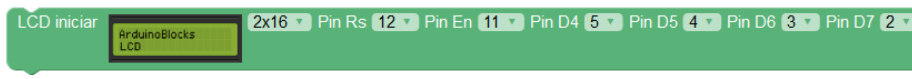
- **Conexión por bus de comunicaciones I2C:**

Es la forma más sencilla, necesitamos una pantalla con interfaz I2C o un módulo adaptador que realiza todo el trabajo.

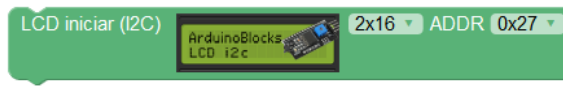


- **LCD iniciar:** Permite configurar la forma de conexión de la pantalla LCD a la placa Arduino. Recomendable en el bloque "inicializar".

Iniciar con conexión de 4 bits:



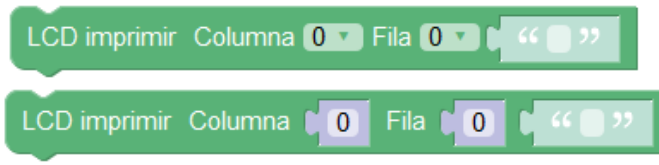
Iniciar con conexión I2C:



(La dirección I2C depende del módulo o pantalla LCD, 0x27 es la más común)

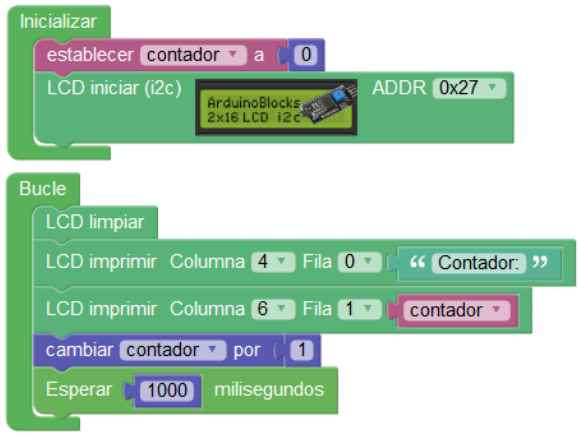
- **LCD limpiar:** Borra el contenido de toda la pantalla LCD

- **LCD imprimir:** Imprime un texto o variable en la fila y columna seleccionada dentro de la pantalla.



(Fila 0: superior, Fila 1: Inferior, Columna: 0...15)

Ejemplo: Contador en pantalla LCD con conexión I2C



3.3.8 MEMORIA EEPROM

La memoria EEPROM es un memoria interna del microcontrolador de Arduino que nos permite guardar información. Tiene la propiedad de no ser volátil, por lo que la información permanece guardada en ella aunque quitemos la alimentación eléctrica.

Esta memoria es perfecta para almacenar información de configuración de la aplicación o valores de estado que se necesiten recuperar después de un corte de la alimentación eléctrica.

El microcontrolador de la placa Arduino UNO tiene 1024 bytes de memoria EEPROM, sin embargo en ArduinoBlocks cada variable usada internamente utiliza 4 bytes por lo que a la hora de almacenar o recuperar una variable de la memoria EEPROM sólo podemos almacenar en 256 posiciones (256 x 4 = 1024 bytes).

Direccionamiento Arduino: 0-1023

0
1
2
3
4
5
6
7
8
9
10

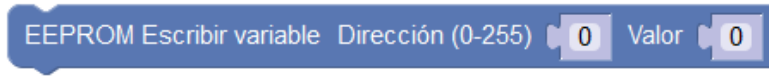
Direccionamiento ArduinoBlocks: 0-255

0
1
2

11
...

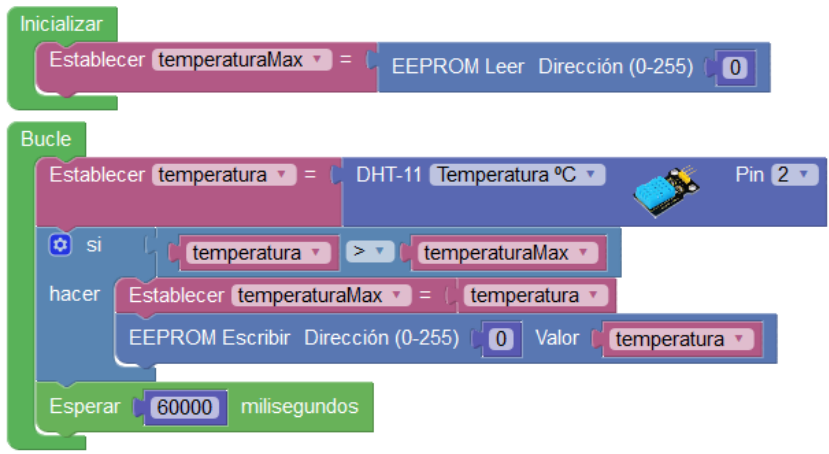
...

- **EEPROM escribir variable:** Guarda un valor o variable en una posición de memoria de la memoria.



- **EEPROM leer variable:** Leer un valor de una posición de la memoria.

Ejemplo: Lee la temperatura cada minuto y guarda la temperatura máxima en la memoria EEPROM para reservarla aunque cortemos la alimentación:



IMPORTANTE: La memoria EEPROM suele venir inicializada a 0xFF por lo que para un uso correcto deberíamos ponerla a 0 en algunos casos:

3.3.9 MOTORES

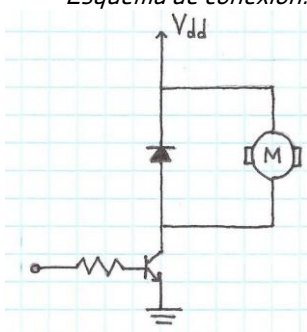
Desde la placa Arduino es fácil controlar varios tipos de motores.

Motores de corriente continua

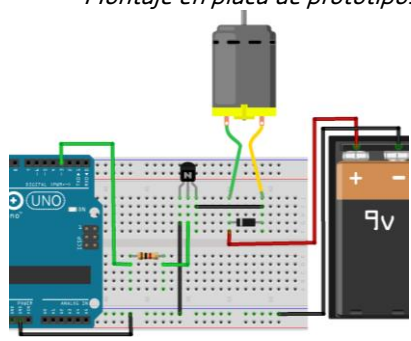
Las salidas de la placa Arduino no proporcionan suficiente corriente para controlar un motor de corriente continua (Arduino proporciona unos 50mA y un motor puede consumir unos 1000mA) por lo que necesitaremos realizar un pequeño circuito con un transistor para controlar una corriente mucho mayor.

Utilizaremos un transistor NPN en modo corte/saturación que permitirá, como un interruptor, el paso de una intensidad de corriente mucho más alta desde un fuente de alimentación auxiliar.

Esquema de conexión:



Montaje en placa de prototipos

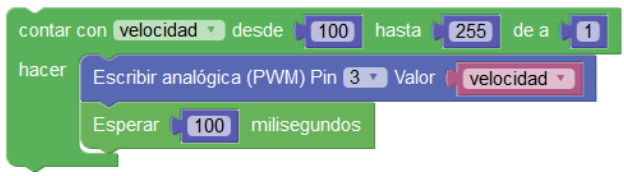


La pila de 9v genera la corriente necesaria para mover el motor. A través del pin 3 generamos la señal que activa el transistor y permite el paso de corriente de la pila. Si utilizamos la salida como PWM podremos controlar la velocidad del motor (si se escribe un valor bajo, menos de 100 aproximadamente, el motor no girará por no aplicarle la suficiente energía)

Activar giro del motor:

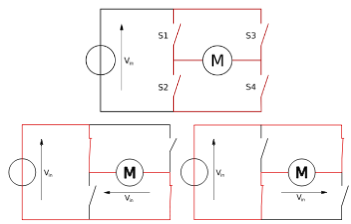
Activar giro controlando velocidad

Ejemplo: Aumento progresivo de la velocidad

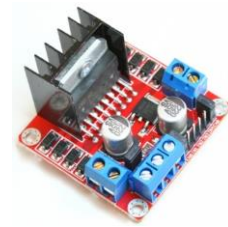


Si necesitamos controlar además el sentido de giro de motor debemos utilizar un "*puente en H*" que nos permite invertir la polaridad en el motor. Lo más fácil es utilizar un driver integrado como el chip L293D o un módulo para Arduino que integre todos los componentes.

Esquema de un puente en H para controlar la dirección de giro de un motor



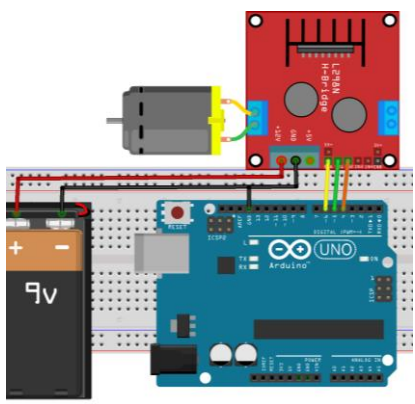
Módulo típico con configuración en puente H para control de motores de C.C.



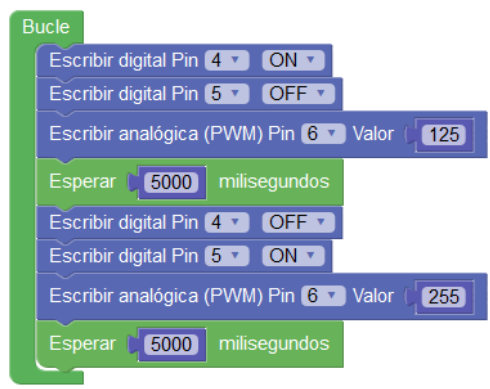
Estos módulos suelen integrar el control para dos motores. Los pines de un módulo de control de motores en puente en H suele tener estas conexiones:

IN1, IN2	Controla el sentido de giro del motor 1	
	IN1 = ON / IN2 = OFF	Giro en un sentido
	IN1 = OFF / IN2 = ON	Giro en sentido contrario
	IN1 = OFF / IN2 = OFF	Parado
IN3, IN4	Controla el sentido de giro del motor 2	
	IN3 = ON / IN4 = OFF	Giro en un sentido
	IN3 = OFF / IN4 = ON	Giro en sentido contrario
	IN3 = OFF / IN4 = OFF	Parado
EN1	Habilita el motor 1 (control de velocidad del motor 1 con PWM)	
EN2	Habilita el motor 2 (control de velocidad del motor 2 con PWM)	

Pines 4,5 (IN1, IN2): control de giro
Pin 6 (EN1): control velocidad PWM



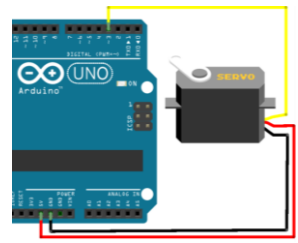
Giro cada 5s en un sentido con distinta velocidad



Servomotor

Los servomotores son motores DC a los que se les ha añadido una reductora y una electrónica de control PID que permite controlar el motor situándolo en una posición muy precisa. El servomotor está intentando siempre situarse en la posición indicada, de forma que si se le fuerza o impide ir hasta la posición indicada intentará moverse a la posición indicada continuamente.

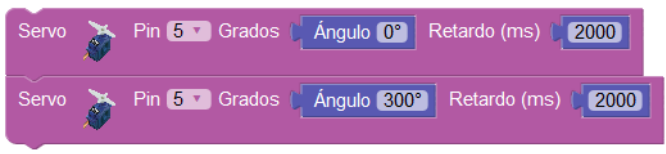
Los servomotores pueden situarse en una posición entre 0º y 300º aproximadamente según el modelo. Un servomotor no permite el giro libre a no ser que se modifique con ese propósito.



El control de la posición de un servomotor se realiza mediante PWM por lo que necesitamos conectarlo a una salida digital de tipo PWM.

En ArduinoBlocks tenemos un bloque que nos permite controlar fácilmente un servomotor indicándole la posición en grados donde queremos que se sitúe y el retardo en milisegundos para darle tiempo a que se mueva hasta la posición indicada.

Ejemplo: movimiento de un servomotor conectado al pin 5:

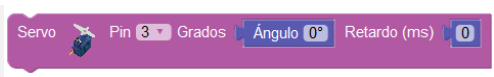
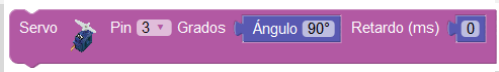


Ejemplo: Mover el servo de 0 a 300 grados de 10 en 10 grados

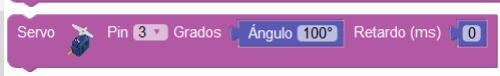
Existe un tipo especial de servomotor que permite la rotación continua. En algunos casos se trata de servomotores "trucados" de forma que se modifican para permitir la rotación continua quitando los topes mecánicos y se sustituye el potenciómetro por un divisor de tensión con dos resistencia iguales (en algunos casos no se ponen resistencias y se bloquea el potenciómetro para que no gire dejándolo justo en su punto central).

En cualquier caso también podemos comprar un servomotor de rotación continua listo para funcionar sin tener que hacer bricolaje.

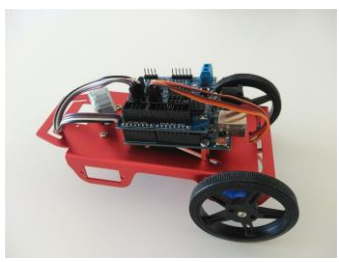
El control de un servomotor de rotación continua se realiza de igual manera, pero su reacción es diferente.

0º		Giro en un sentido (máxima velocidad)
90º		Parado
180º		Giro en sentido contrario (máxima velocidad)

Si utilizamos valores cercanos a 90º el motor girará a una velocidad más lenta en cada uno de los sentidos.

80º		Giro en un sentido (velocidad lenta)
100º		Giro en sentido contrario (velocidad lenta)

Ejemplo: robot propulsado por dos servos de rotación continua



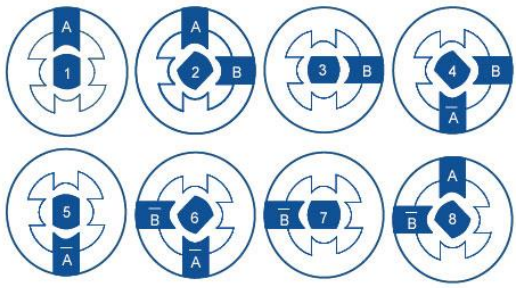
Motor paso a paso

Este tipo de motor es capaz de avanzar una serie de grados (paso) dependiendo de su entrada de control. Son ideales para los mecanismos donde se requiera mucha precisión, por ejemplo son utilizados para los mecanismos de movimiento de las impresoras 3D.

Estos motores están formados por un rotor sobre el que van aplicados varios imanes permanentes y por un cierto número de bobinas excitadoras en su estator.

La excitación de las bobinas se controla externamente y determina el giro del rotor.

Secuencia de activación de las bobinas para giro del motor en una dirección



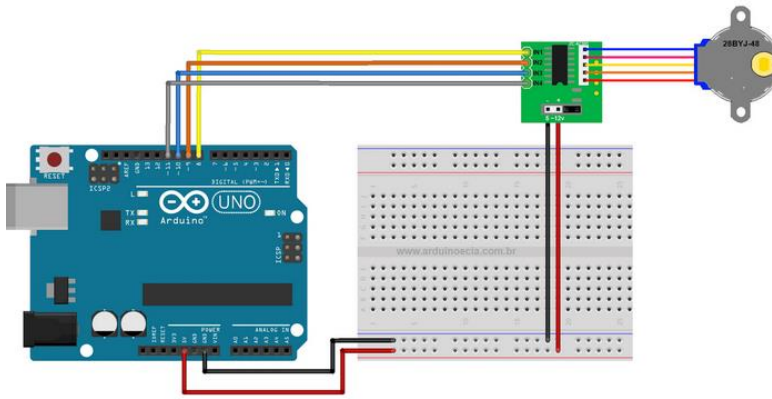
Motor paso a paso y módulo de control

Para controlar un motor paso a paso utilizamos un módulo capaz de controlar cada una de las 4 señales de control que activan cada bobina. Realizando la secuencia correcta movemos el motor, según el número de pasos y velocidad a la que avanzamos en la secuencia. Todo esto se realiza internamente automáticamente.

- **Pasos/vuelta:** Configura la conexión de las bobinas del motor paso a paso así como el parámetro de pasos por vuelta para controlar el motor correctamente.

- **Velocidad:** Establece la velocidad de giro del motor en rpm (revoluciones por minuto).

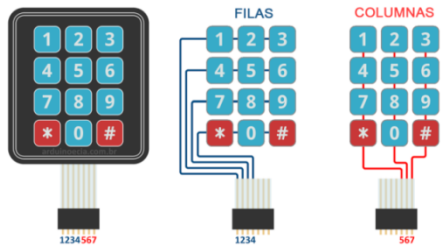
- **Pasos:** Avanza un número de pasos el motor.



Ejemplo - Control de dos motores paso a paso

3.3.10 KEYPAD

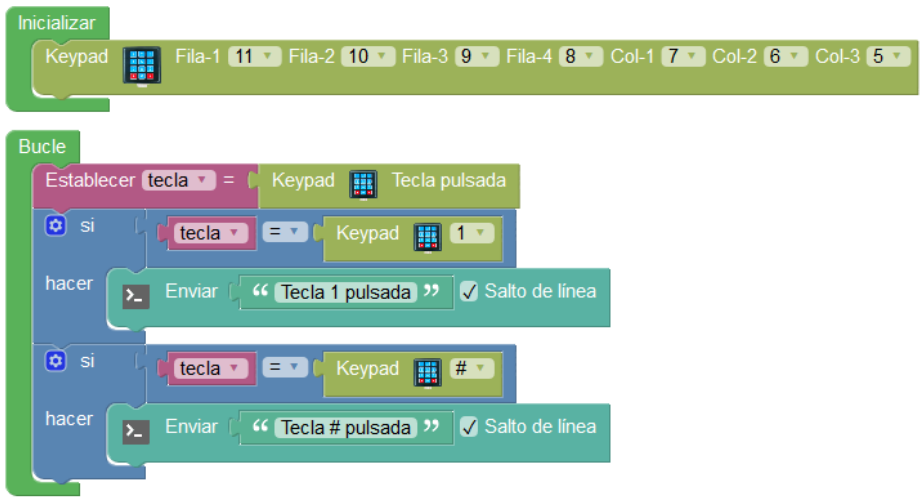
El teclado o "keypad" nos permite de una forma sencilla añadir un pequeño teclado numérico a nuestro proyecto. Se basa en una botonera conectada de forma matricial por filas y columnas. ArduinoBlocks gestiona automáticamente la detección de filas y columnas activadas para detectar la tecla pulsada.



- **Configuración del keypad:** define los pines de conexión para las filas y columnas del keypad.

- **Tecla pulsada:** obtiene la tecla pulsada actualmente en el keypad.

Ejemplo: Detección de las teclas '1' y '#'

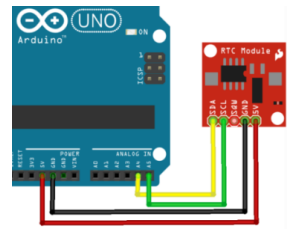


3.3.11 RELOJ DE TIEMPO REAL (RTC)

El reloj de tiempo real DS3231 es un reloj de alta precisión. El reloj incorpora una batería para guardar la fecha y la hora cuando la placa Arduino pierde la alimentación.

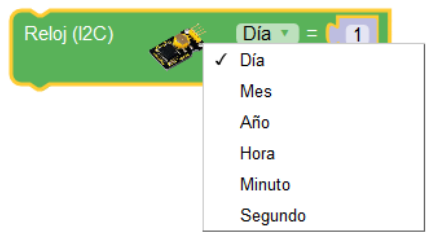
Se comunica con el microcontrolador de Arduino por comunicación I2C.

Usaremos las conexiones: VCC, GND, SCL, SDA



- **Reloj fijar fecha/hora:** permite actualizar los valores de fecha y hora.

- **Fijar campo de fecha/hora:**

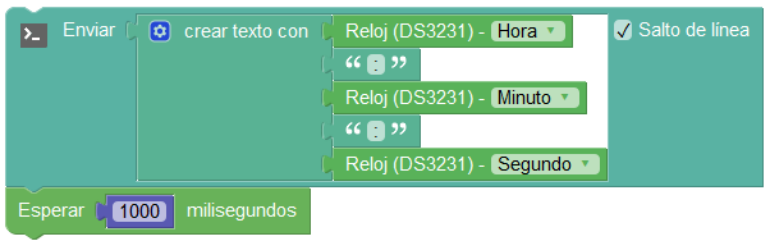


- **Obtener campo de fecha/hora:** permite obtener los campos de fecha y hora de forma independiente.

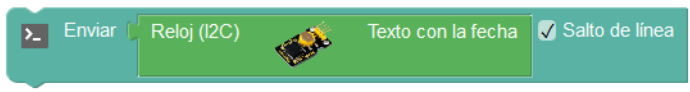
- **Obtener texto con la fecha:** permite obtener un valor de tipo texto con la fecha formateada como DD/MM/YYYY

- **Obtener texto con la hora:** permite obtener un valor de tipo texto con la fecha formateada como hh:mm:ss

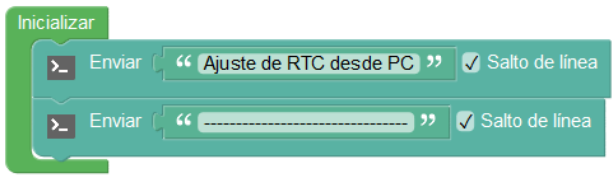
Ejemplo: enviar la hora completa cada segundo por la conexión serie

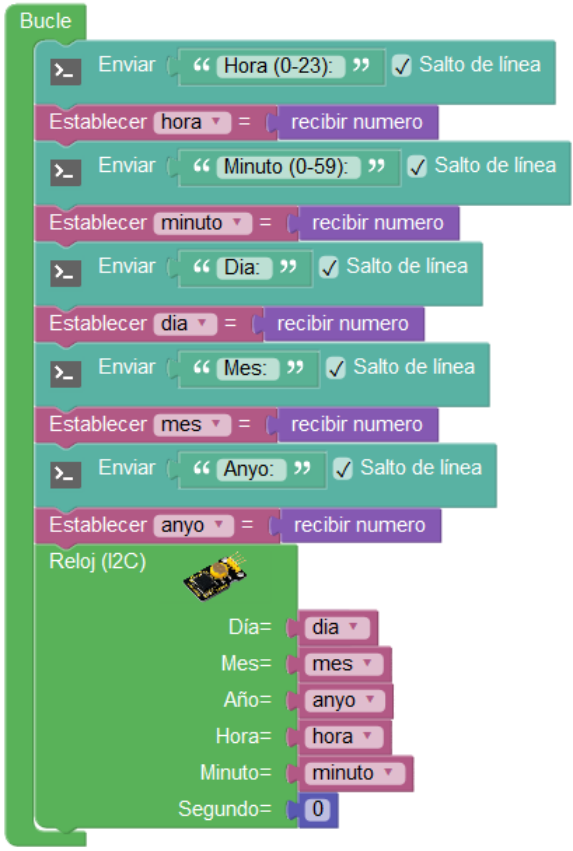


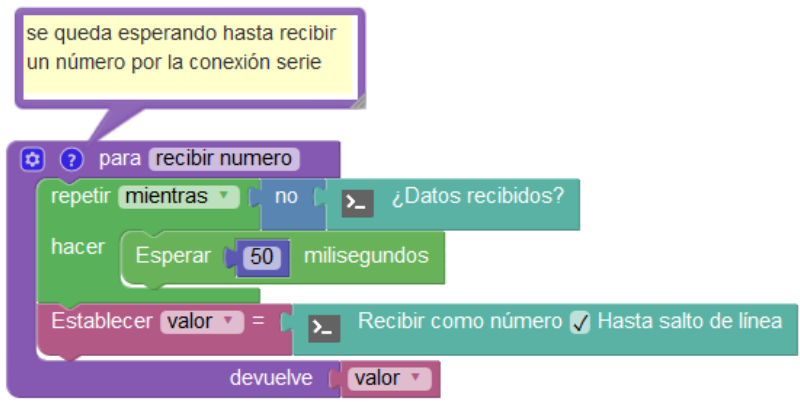
Ejemplo: enviar la fecha por la consola serie



Ejemplo: Ajuste de fecha y hora del reloj RTC desde PC (vía consola serie)

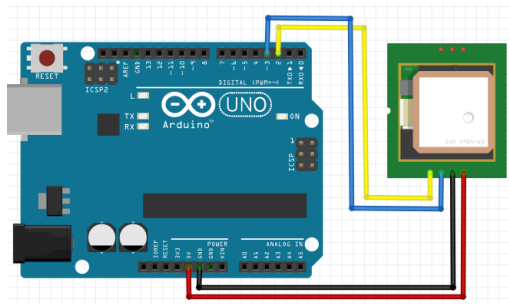



Bucle
Enviar “ Hora (0-23): ” Salto de línea
Establecer hora = recibir numero
Enviar “ Minuto (0-59): ” Salto de línea
Establecer minuto = recibir numero
Enviar “ Dia: ” Salto de línea
Establecer dia = recibir numero
Enviar “ Mes: ” Salto de línea
Establecer mes = recibir numero
Enviar “ Anyo: ” Salto de línea
Establecer anyo = recibir numero
Reloj (I2C)
Día= dia
Mes= mes
Año= anyo
Hora= hora
Minuto= minuto
Segundo= 0


se queda esperando hasta recibir un número por la conexión serie
para recibir numero
repetir mientras no ¿Datos recibidos?
hacer Esperar 50 milisegundos
Establecer valor = Recibir como número Hasta salto de línea
devuelve valor

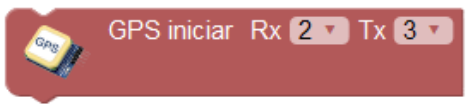
3.3.12 GPS

Los módulos GPS nos permiten de forma sencilla obtener los datos de posición global (latitud/longitud), velocidad, orientación, altitud, … facilitados por el sistema de posicionamiento global. El módulo GPS conectado debe ser un módulo de conexión serie que proporcione los datos según el protocolo NMEA. Uno de los módulos más utilizados de este tipo son los GPS NEO-6.



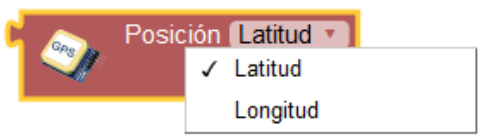
Ejemplo de conexión del módulo GPS

- **GPS Iniciar:** Inicia el módulo GPS indicando los pines utilizados para la comunicación serie con el módulo.



- **¿Datos válidos?:** Indica verdadero en caso de que el módulo GPS reciba señal desde los satélites GPS de forma correcta y los datos obtenidos sean válidos, si no obtendremos valor falso.

- **Posición:** Obtiene la latitud y longitud para así obtener la información de la posición actual. Los valores de latitud y longitud son valores decimales que determinan nuestra posición sobre la Tierra.

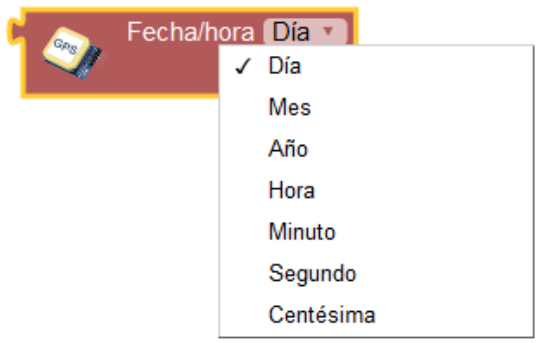


- **Velocidad:** Obtiene el valor de la velocidad a la que nos movemos, puede ser en Km/h o Millas/h.

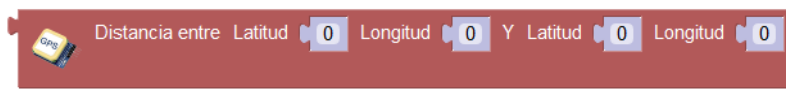
- **Altitud:** Obtiene la altitud en metros sobre el nivel del mar.

- **Rumbo:** Indica el valor en grados de la dirección a las que nos dirigimos.

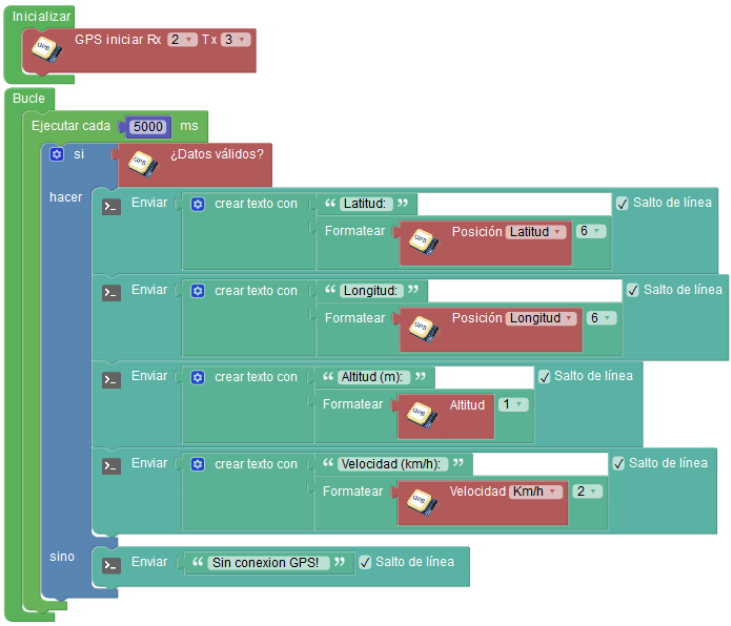
- **Fecha/hora:** Obtiene los valores de fecha y hora recibidos desde el satélite GPS.



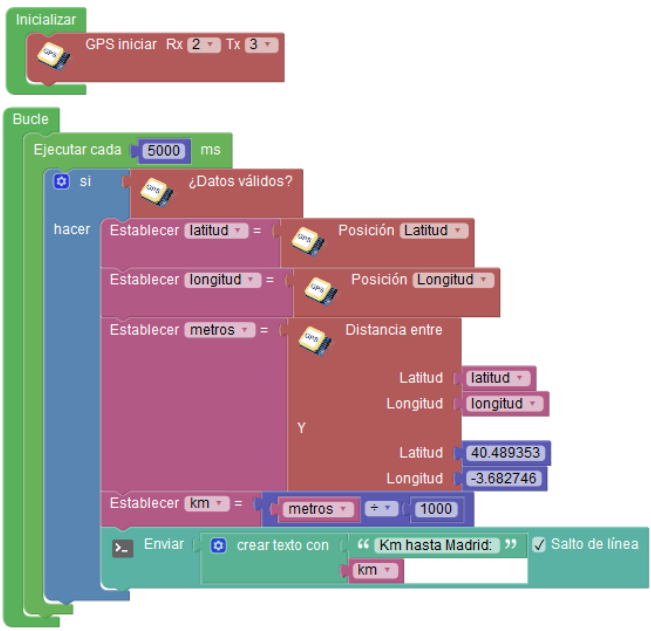
- **Distancia entre:** Calcula los metros de distancia en línea recta entre dos puntos indicando la latitud y longitud del punto inicial y final.



Ejemplo: Mostrar la información de la posición GPS por la conexión serie cada 5s



Ejemplo: Mostrar la distancia en Km hasta Madrid desde nuestra posición actual cada 5s

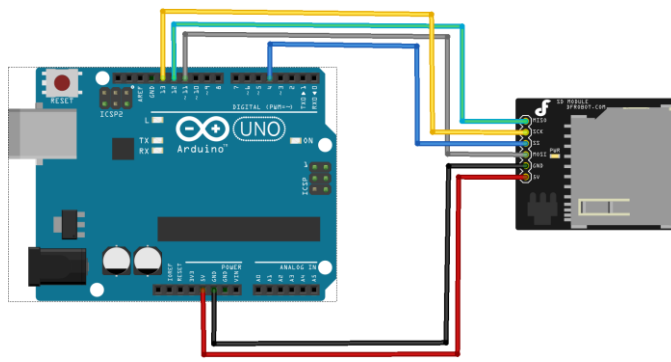


3.3.13 TARJETA SD

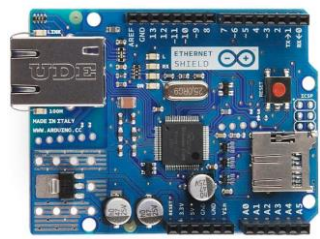
Los bloques de tarjeta SD nos permiten trabajar con archivos almacenados en una tarjeta SD o microSD conectada a Arduino. Los módulos para tarjetas SD utilizan la conexión SPI para comunicarse con la placa Arduino.

Este tipo de almacenamiento nos permite realizar aplicaciones de registro de datos (datalogger), guardar configuración, etc.

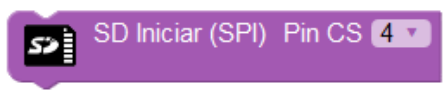
Los módulos o shields SD se conectan con la interfaz SPI utilizando los pines 11,12 y 13 y otro pin para CS (normalmente las shields utilizan el pin 4).



Algunas shields como la "*Ethernet*" incorporan también un módulo para tarjetas SD. Debemos comprobar su documentación para asegurarnos los pines que utilizan (en el caso de la shield "*Ethernet*" utiliza el pin 4 para CS)



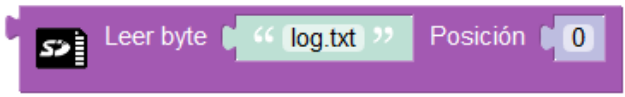
- **Iniciar SD:** Inicia el uso del módulo de tarjetas SD indicando los pines donde está conectado. (los pines SPI son fijos, sólo indicamos el pin CS)



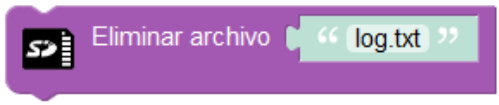
- **Imprimir:** Escribe un texto dentro de un archivo de texto en la tarjeta SD. El texto se añade al final del contenido actual del archivo.

- **Escribir byte:** Escribe un byte al final del archivo indicado.

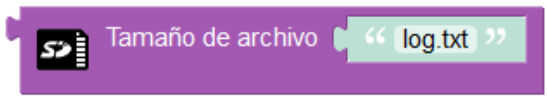
- **Leer byte:** Lee un byte del archivo indicado de la posición seleccionada.



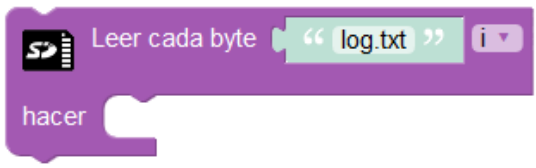
- **Eliminar archivo:** Elimina un archivo de la tarjeta SD.



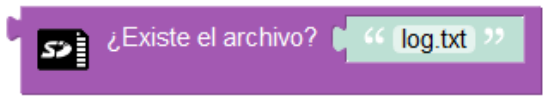
- **Tamaño de archivo:** Obtiene el tamaño en bytes del archivo indicado.



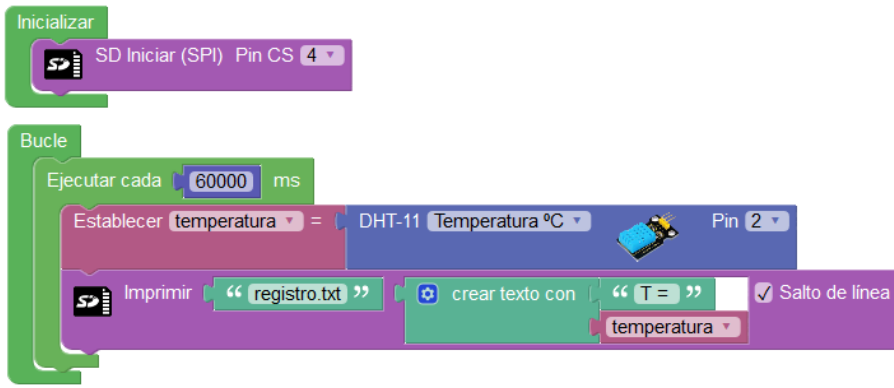
- **Leer cada byte:** Permite leer byte a byte todos los datos de un archivo.



- **Existe el archivo:** Obtiene el valor verdadero si el archivo existe o falso en caso contrario.

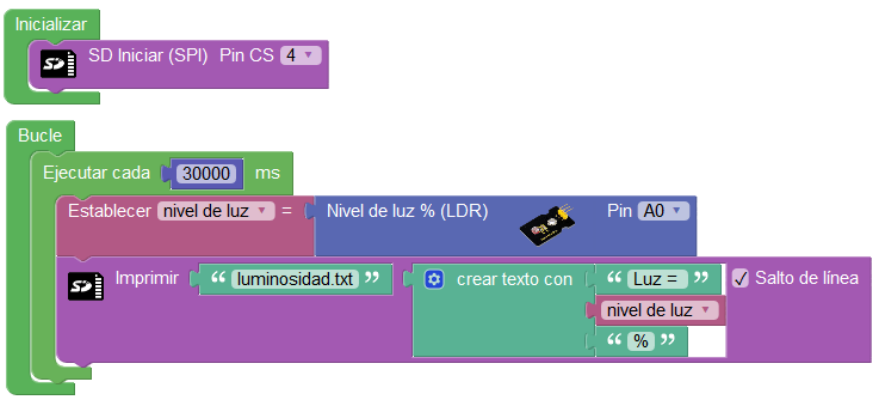


Ejemplo: Registrar la temperatura en un archivo de texto cada minuto



Ejemplo: Volcar todo el contenido de un archivo por la consola serie

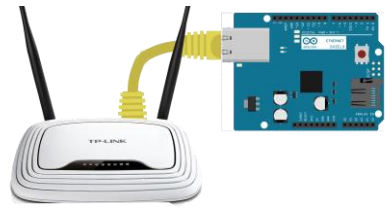
Ejemplo: Registrar el nivel de luminosidad medido con una LDR cada 30s



3.3.13 MQTT

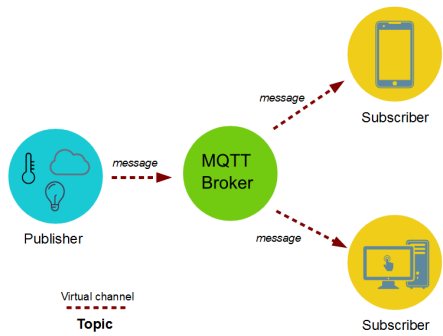
El protocolo MQTT permite conectar nuestro Arduino al IoT (internet de las cosas) a través de la shield Ethernet. La shield Ethernet utiliza los pines 10,11,12 y 13 (SPI) . Además utiliza el pin 4 si utilizamos el módulo de tarjetas SD que incorpora.

La shield Ethernet incorpora un conector RJ45 para cable Ethernet que debemos conectar a nuestro router o switch con conexión a internet.



MQTT es un protocolo de comunicación para redes TCP/IP muy sencillo y ligero en el que todos los dispositivos se conectan a un servidor (llamado "*broker*"). Los dispositivos pueden enviar (*publicar*) o recibir (*suscribirse*) mensajes asociándoles un "*topic*" (*tema*).

El "*broker*" se encarga de gestionar los mensajes y distribuirlos entre todos los dispositivos conectados.



Podemos implementar nuestro propio servidor/broker. Existen *brokers* MQTT de código libre como "Mosquitto" que podemos instalar en diferentes sistemas operativos de forma sencilla. Un ejemplo típico es configurar una *Raspberry Pi* como servidor MQTT en casa. Si queremos que el sistema esté abierto a internet deberemos configurar nuestra conexión adecuadamente al igual que obtener nuestra IP pública actual o contratar una IP pública fija.

Por otro lado podemos utilizar *brokers* MQTT públicos disponibles en internet con fines experimentales o docentes y en cualquier otro caso podemos contratar servicios de *brokers* de pago con diferentes limitaciones de ancho de banda o número de conexiones según nuestras necesidades.

Algunos brokers MQTT públicos:

iot.eclipse.org
broker.hivemq.com

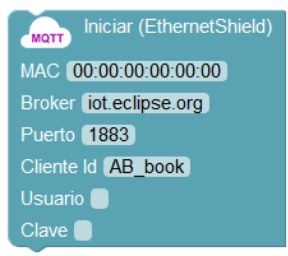
www.cloudmqtt.com
(con usuario y clave, opción gratuita limitada en velocidad y conexiones)

La comunicación entre los nodos de un sistema MQTT se realizan enviando mensajes. Los nodos envían los mensajes al broker y éste se encarga de distribuirlos entre el resto de nodos. Cada mensaje consta de un "*topic*" o tema y el cuerpo del mensaje en sí. Un nodo se puede suscribir a un "*topic*" de forma que recibirá todos los mensajes que tengan ese "*topic*". Cada nodo puede publicar mensajes con el "*topic*" deseado.

Por ejemplo podemos utilizar el topic: "*temp/comedor*" para que un nodo envíe la temperatura del comedor, por otro lado todos los nodos que deseen conocer la temperatura del comedor se suscribirán al topic : "*temp/comedor*" y recibirán automáticamente los mensajes de este tipo.

Bloques para la programación MQTT:

- **Iniciar MQTT:** Inicia la conexión MQTT a través de la shield Ethernet. La dirección MAC generada es aleatoria, si nuestra shield incluye una etiqueta con la dirección MAC debemos ponerla. Por otro lado indicaremos el *broker* y puerto a utilizar, el usuario/clave si es necesario y el identificador del cliente MQTT. La tarjeta de red Ethernet intentará obtener la configuración IP de forma automáticamente por lo que nuestra red deberá tener un servidor DHCP activo que proporcione esta información (cualquier router doméstico lleva esta opción activa por defecto).



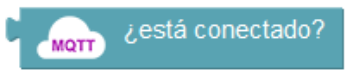
- **Publicar MQTT:** Permite enviar un mensaje al *broker* para que los nodos suscritos a este *topic* reciban el valor. El tema es el "*topic*" a publicar y el valor puede ser un valor fijo (texto o numérico) o el valor de una variable.

- **Suscribir MQTT:** Este bloque realiza la suscripción a un "*topic*" o tema. ArduinoBlocks mapea el valor recibido en el mensaje a una variable de forma que cuando se recibe un mensaje del "*topic*" automáticamente el valor de la variable se actualizará.

  Para variables numéricas (el valor recibido debe ser un número válido y se guardara en la variable indicada)

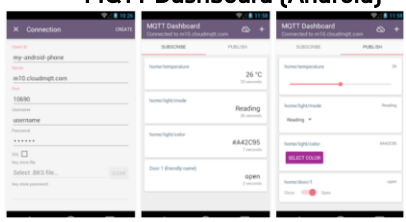
  En este caso almacenaremos el mensaje recibido en una variable de tipo texto:

- **¿Está conectado?:** Obtiene el estado de la conexión, indicando verdadero si se ha podido establecer la conexión con el *broker* o falso en caso contrario.

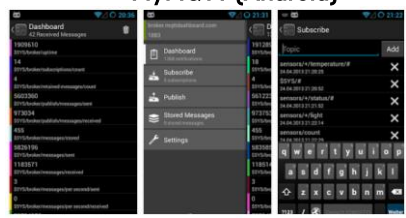


Existen multitud de aplicaciones, especialmente para dispositivos móviles, para conectarse a un *broker* MQTT y publicar o suscribirse a *topics*. Algunas de ellas además permiten crear paneles de control y monitorización muy llamativos.

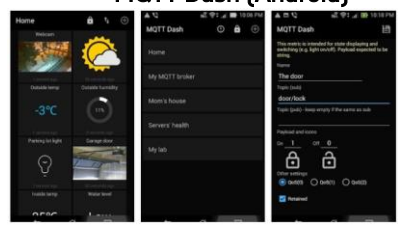
MQTT Dashboard (Android)



MyMQTT (Android)

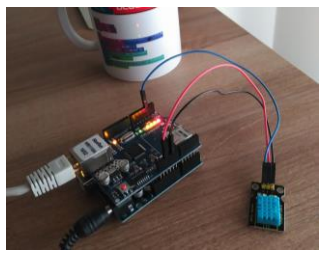


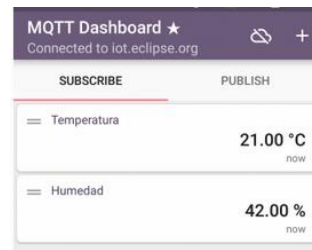
MQTT Dash (Android)

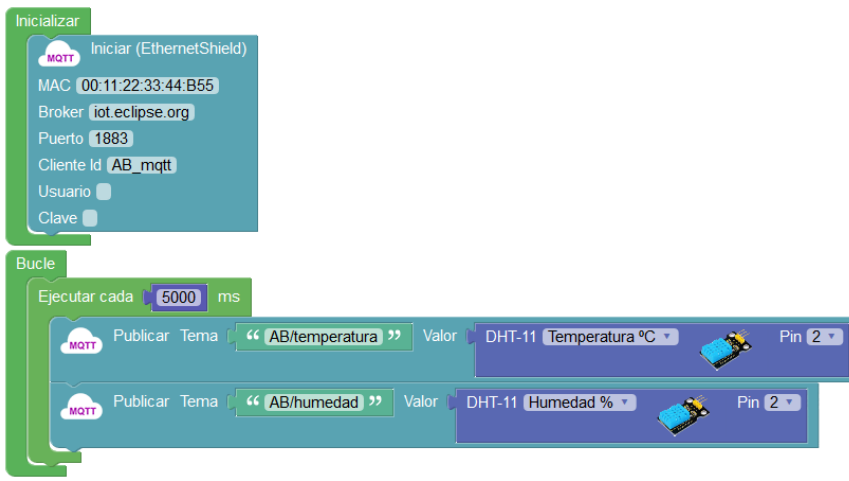


Ejemplo: Enviar la temperatura y humedad medido cada 5s

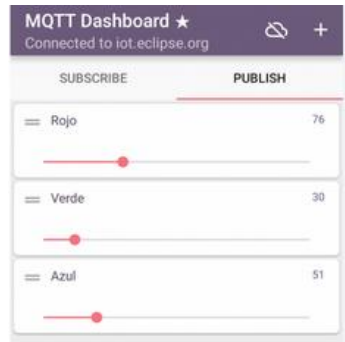
(recuerda que no debes utilizar bloqueos de tiempo para que el sistema MQTT funcione bien)

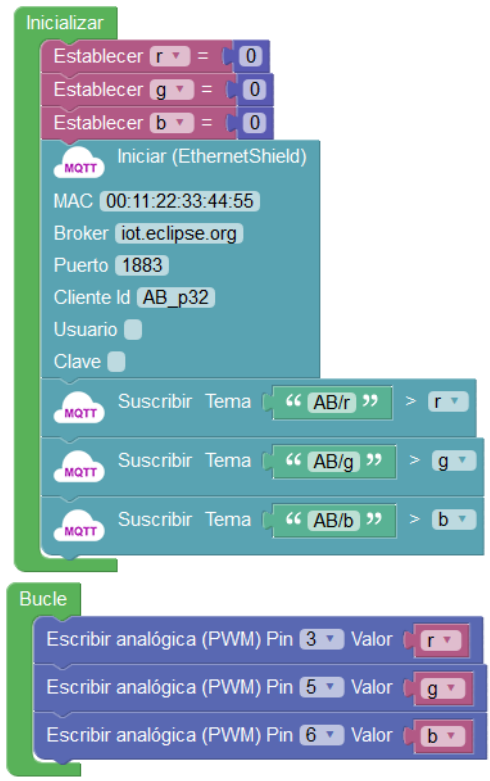




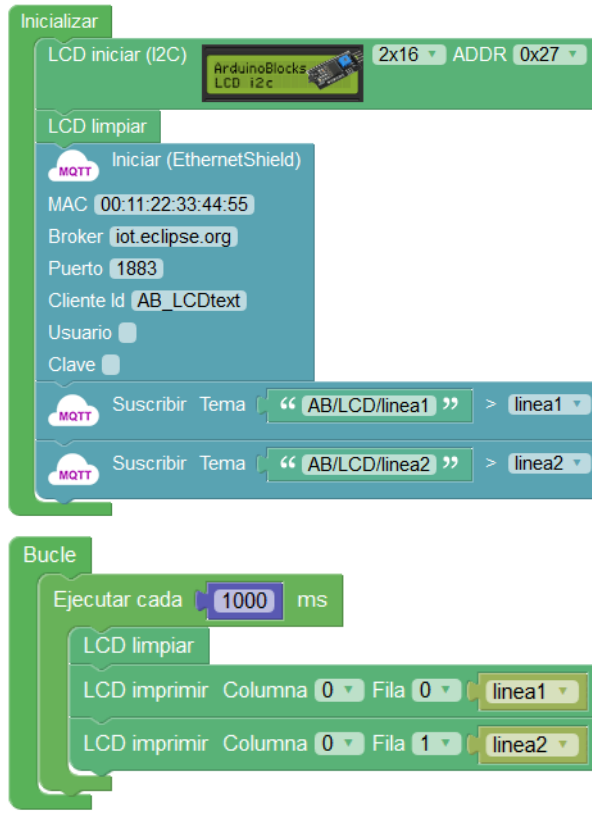


Ejemplo: Suscribirse a tres "topics" para controlar un led RGB.

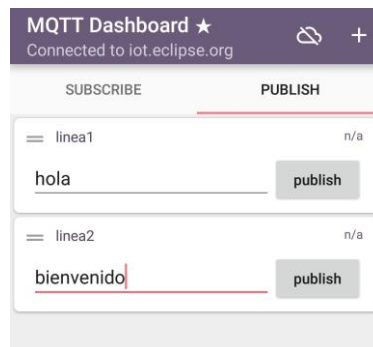




Ejemplo: Recibir un texto via MQTT para mostrar en un display LCD



Desde la aplicación MQTT Dashboard (Android) podemos modificar los valores de los textos que se visualizan en cada línea del LCD:



4 PROYECTOS

A continuación se detallan 40 proyectos desarrollados en ArduinoBlocks con esquemas y programas de bloques.

La funcionalidad de cada proyecto está simplificada, el objetivo es mostrar las posibilidades de la plataforma con aplicaciones reales sencillas.

En la web de ArduinoBlocks podemos encontrar proyectos realizados por otros usuarios (proyectos compartidos) que nos sirvan también como inspiración o punto de partida para nuestros propios proyectos.

Los 40 proyectos de este libro se encuentran compartidos y accesibles en la web de ArduinoBlocks.

http://www.arduinoblocks.com/web/site/search

Listado de proyectos resueltos:

P01.-Secuenciador de luces
P02.-Simulación amanecer y anochecer
P03.-Lámpara con regulación manual
P04.-Semáforo
P05.-Timbre
P06.-Control inteligente de iluminación
P07.-Encendido automático por movimiento
P08.-Contador manual
P09.-Cronómetro
P10.-Fotómetro
P11.-Iluminación crepuscular
P12.-Encendido y apagado con palmada
P13.-Termómetro
P14.-Termostato
P15.-Medidor de distancia
P16.-Riego automático
P17.-Lámpara multicolor con control IR
P18.-Piano con teclado
P19.-Sensor de aparcamiento
P20.-Control pan/tilt con joystick

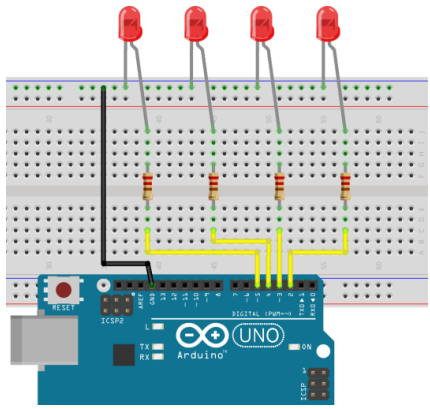
P21.-Control de un led desde PC
P22.-Control de relés por Bluetooth
P23.-Estación meteorológica
P24.-Control de led por voz
P25.-Control domótico
P26.-Visualización GPS en LCD
P27.-Aviso por exceso de velocidad
P28.-Alarma por alejamiento
P29.-Registrador GPS en tarjeta SD
P30.-Registro de temperatura y humedad en tarjeta SD
P31.-Control de servo con acelerómetro
P32.-Sensor de caída con aviso a emergencias vía Bluetooth
P33.-MQTT (IoT): Control de led RGB
P34.-MQTT (IoT): Estación meteorológica
P35.-MQTT (IoT): Control domótico
P36.-Robot con servos controlador por Bluetooth
P37.-Robot con motores DC - Control Bluetooth
P38.-Robot con motores DC – Evita obstáculos
P39.-Robot con motores DC – Sigue líneas
P40.-Brazo robótico con servos – Control PC

P01 - SECUENCIADOR DE LUCES

Un secuenciador es capaz de repetir ciclos de encendido y apagado de leds siguiendo un orden para lograr un efecto visual llamativo y divertido. Podemos utilizar nuestro secuenciador de luces para decorar el árbol de Navidad, realizar carteles luminosos llamativos o simplemente para animar una fiesta con los amigos.

Material necesario:

- 4 x leds de los colores deseados.
- 4 x resistencias de 220 Ω.
- Placa de prototipos, cables de interconexión.



Conexiones:
Led 1 = Pin 2
Led 2 = Pin 3
Led 3 = Pin 4
Led 4 = Pin 5

Programa ArduinoBlocks:

Secuencia 1:

Secuencia 2:

P02 - SIMULACIÓN DE AMANECER Y ANOCHECER

Con este proyecto vamos a simular el ciclo solar de anochecer y amanecer donde la luz disminuye o aumenta progresivamente de forma suave.

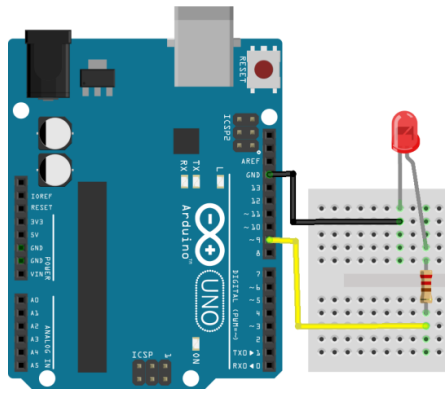
Aplicaciones de ejemplo:

-Belén Navideño con simulación de día/noche

-Aviario para cría de aves

Material necesario:

- 1 x led
- 1 x resistencia de 220 Ω.
- Placa de prototipos, cables de interconexión.



Conexiones:
Led = Pin ~9

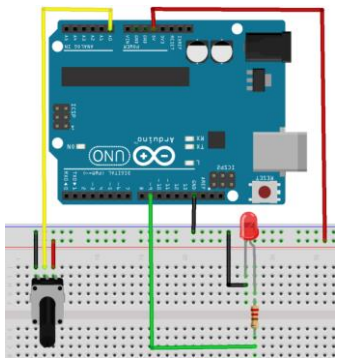
Programa ArduinoBlocks:

P03 - LÁMPARA CON REGULACIÓN DE INTENSIDAD MANUAL

Mediante el uso de un potenciómetro rotativo vamos a controlar la intensidad de iluminación de un led.

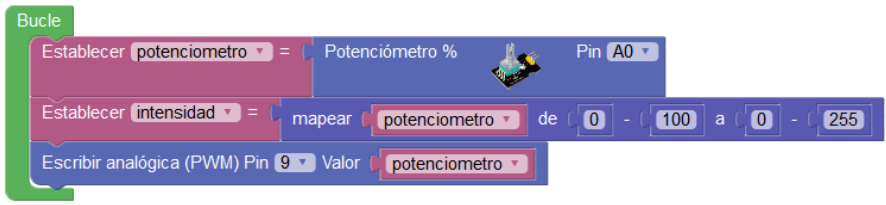
Material necesario:

- 1 x led
- 1 x resistencia de 220 Ω.
- 1 x potenciómetro 10k
- Placa de prototipos, cables de interconexión.

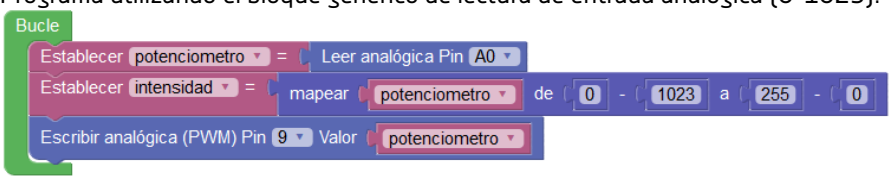


Conexiones:
Led = Pin ~9
Potenciómetro = Pin A0

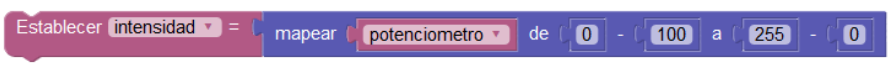
Programa utilizando el bloque de potenciómetro (0-100%):



Programa utilizando el bloque genérico de lectura de entrada analógica (0-1023):



Modificando el mapeo del valor leído al valor enviado al led podemos invertir el sentido de giro para aumentar o disminuir la intensidad del led en sentido contrario:

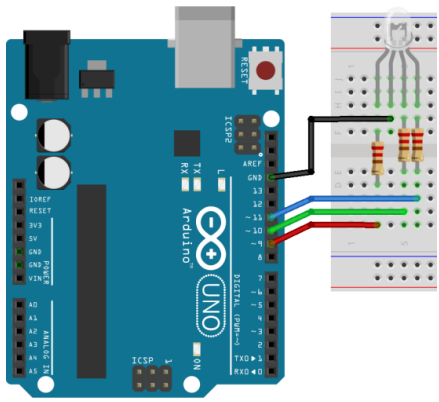


P04 - SEMÁFORO

Con la ayuda de un led RGB vamos a realizar un proyecto que simule el funcionamiento de un semáforo. El semáforo tendrá un tiempo en verde para permitir el paso, un tiempo pequeño en naranja parpadeando marcando peligro y un tiempo en rojo prohibiendo el paso.

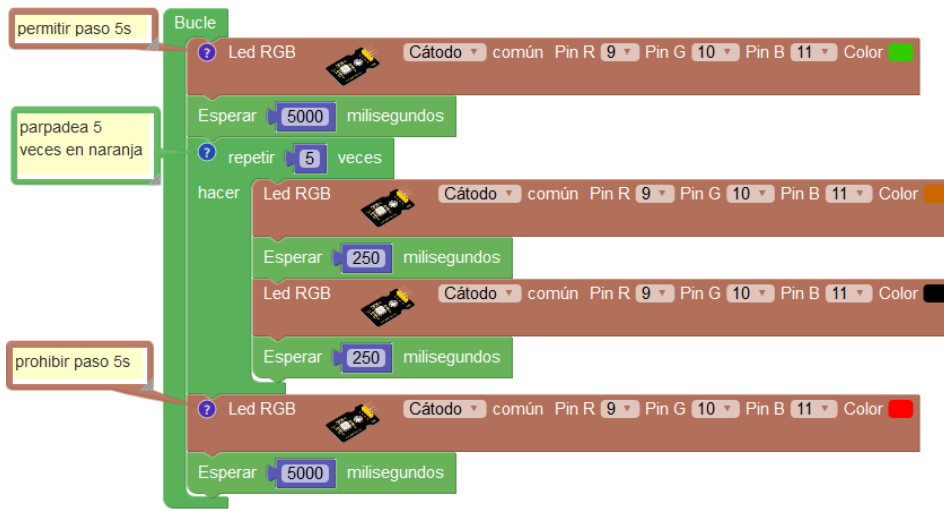
Material necesario:

- 1 x led RGB de cátodo común
- 3 x resistencia de 220 Ω.
- Placa de prototipos, cables de interconexión.



Conexiones:
Led R = Pin ~9
Led G = Pin ~10
Led B = Pin ~11

Programa ArduinoBlocks:

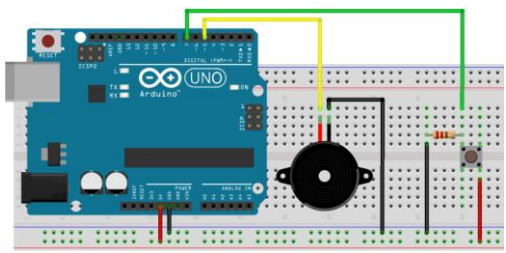


P05 - TIMBRE

Con este sencillo proyecto vamos a realizar un timbre para casa, al detectar el pulsador presionado reproduciremos una melodía con el zumbador.

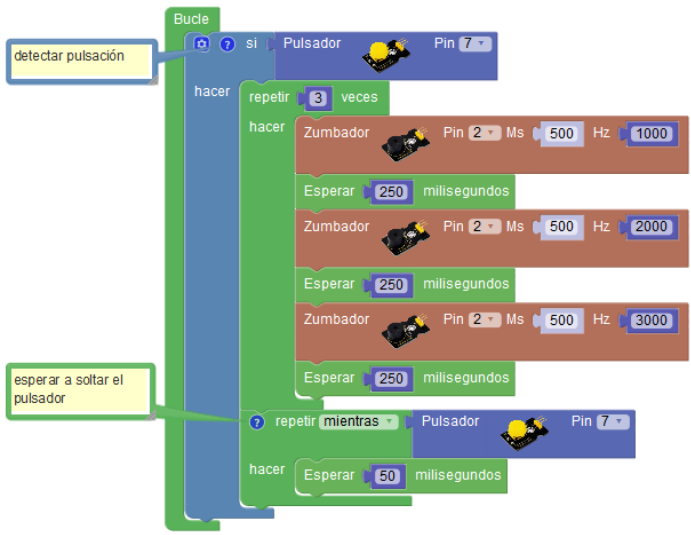
Material necesario:

- 1 x zumbador
- 1 x pulsador
- 1 x resistencia 10 kΩ
- Placa de prototipos, cables de interconexión.

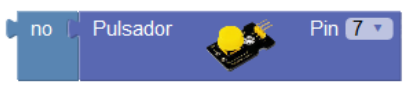


Conexiones:
Zumbador = Pin 5
Pulsador = Pin 7

Programa ArduinoBlocks:



Si el pulsador funciona de forma lógica inversa (normal = “ON” / pulsado = “OFF”) sólo haría falta negar el estado del pulsador:

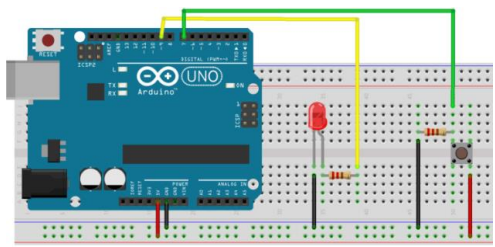


P06 - CONTROL INTELIGENTE DE ILUMINACIÓN

Vamos a realizar un proyecto donde con un único pulsador podemos encender, apagar o regular la intensidad de un led. Con una pulsación corta encenderemos o apagaremos el led. Con una pulsación larga aumentaremos en pequeños incrementos la intensidad de iluminación del led.

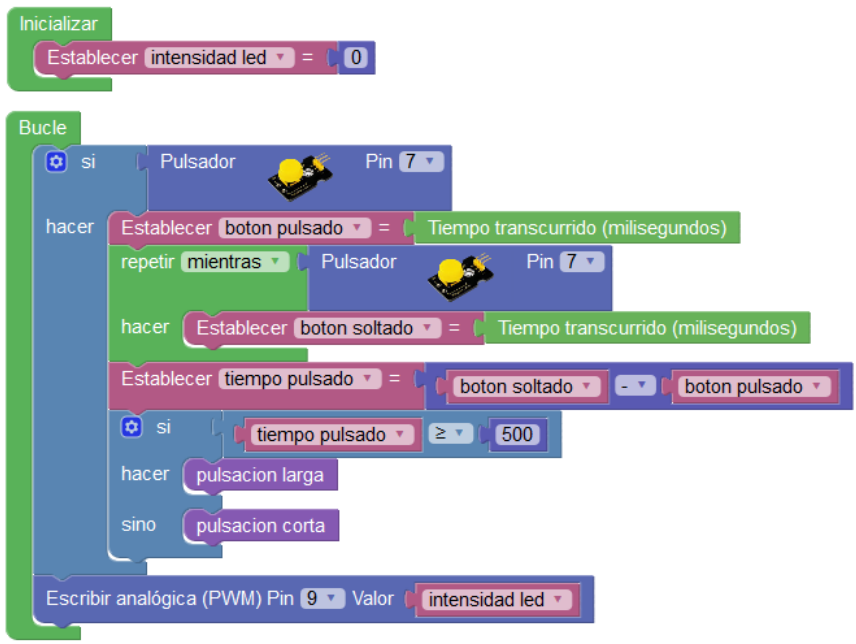
Material necesario:

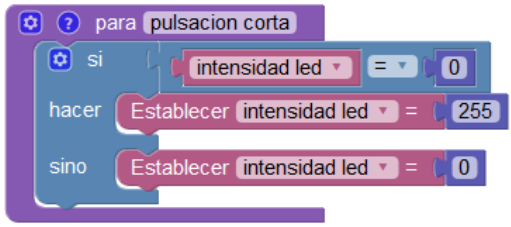
- 1 x led
- 1 x pulsador
- 1 x resistencia de 220Ω
- 1 x resistencia 10 kΩ
- Placa de prototipos, cables de interconexión.

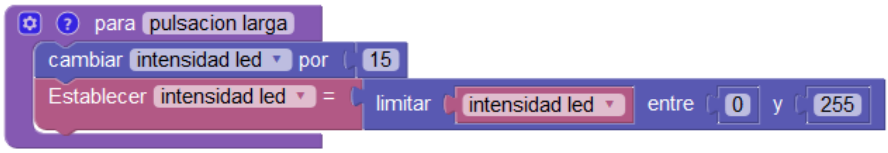


Conexiones:
Led = Pin ~9
Pulsador = Pin 7

Programa ArduinoBlocks:

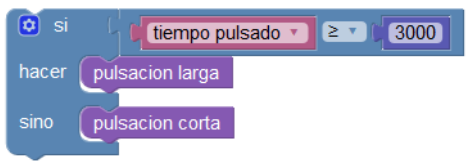




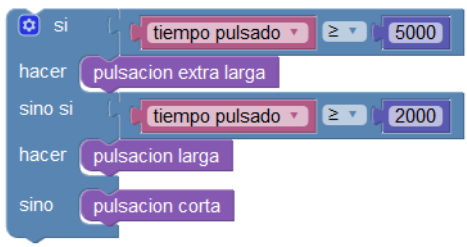


Si el pulsador funciona de forma lógica inversa (normal = “ON” / pulsado = “OFF”) sólo haría falta negar el estado del pulsador:

Si queremos ajustar el tiempo para la pulsación larga podemos modificar el valor fijo de 500 por otro valor a nuestro gusto. Por ejemplo para detectar pulsaciones más largas, por ejemplo de 3 o más segundos, realizaríamos el siguiente cambio:



Encadenando varias condiciones podríamos detectar pulsaciones de distintos tiempos:

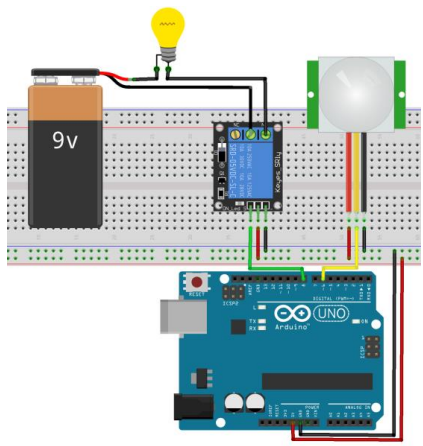


P07 - ENCENDIDO AUTOMÁTICO POR MOVIMIENTO

El ahorro energético es cada vez más importante. Por eso con este sistema además de comodidad evitamos dejarnos la luz encendida. El sistema automáticamente activará la luz cuando detecte la presencia de una persona y transcurrido un tiempo a partir de dejar de detectar la presencia la luz se apagará.

Material necesario:

- 1 x módulo de relé
- 1 x módulo de detección de movimiento PIR
- Placa de prototipos, cables de interconexión.



Conexiones:
Relé = Pin 8
Sensor PIR = Pin 6

Programa ArduinoBlocks:

IMPORTANTE: Los detectores PIR en muchos casos incorporan unos potenciómetros para ajustar el retardo y la sensibilidad. Un mal ajuste puede hacer que nuestro montaje no funcione de la forma deseada.

P08 - CONTADOR MANUAL

Mediante un pulsador iremos incrementando un contador que se visualizará en una pantalla LCD. Si hacemos una pulsación larga (5s o más) se reiniciará el contador para empezar una nueva cuenta.

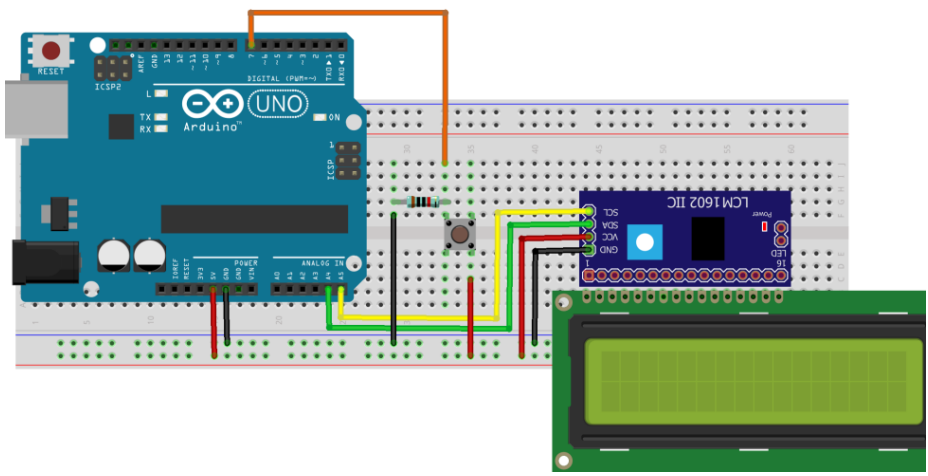
Material necesario:

- 1 x pantalla LCD 2x16 (con módulo de conexión i2c)
- 1 x pulsador
- 1 x resistencia 10kΩ
- Placa de prototipos, cables de interconexión.

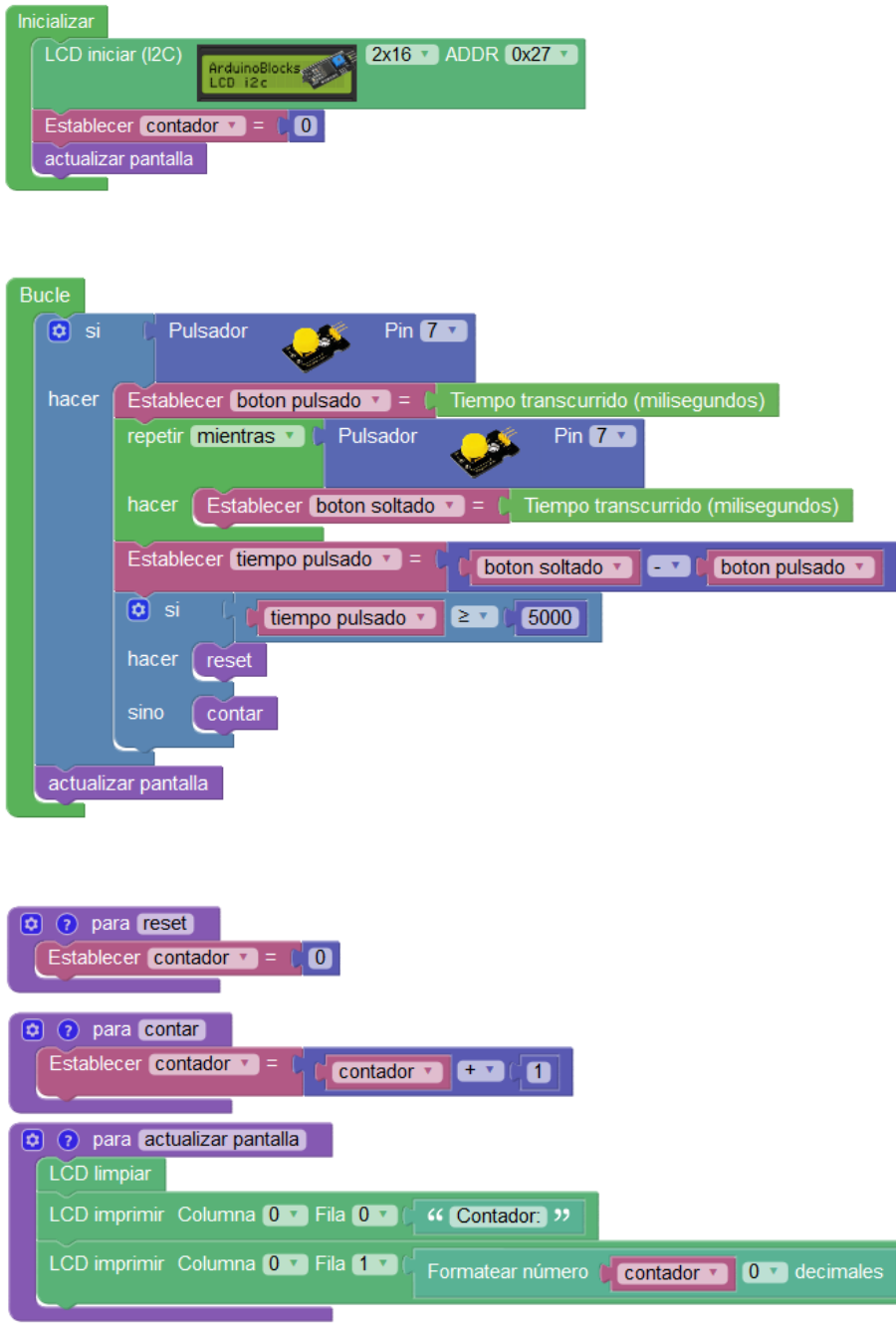
Conexiones

Pulsador = Pin 7

LCD (i2c) = Pin SDA (A4) , Pin SCL (A5)



Programa ArduinoBlocks:

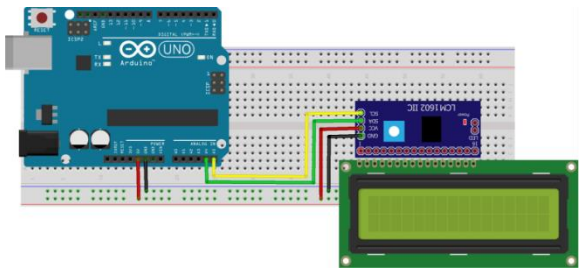


P09 – CRONÓMETRO / CUENTRA ATRÁS

Con sólo una pantalla LCD crearemos un cronómetro capaz de contar horas, minutos y segundos. Con el mismo montaje se implementan los programas de cuenta hacia adelante y de cuenta hacia atrás.

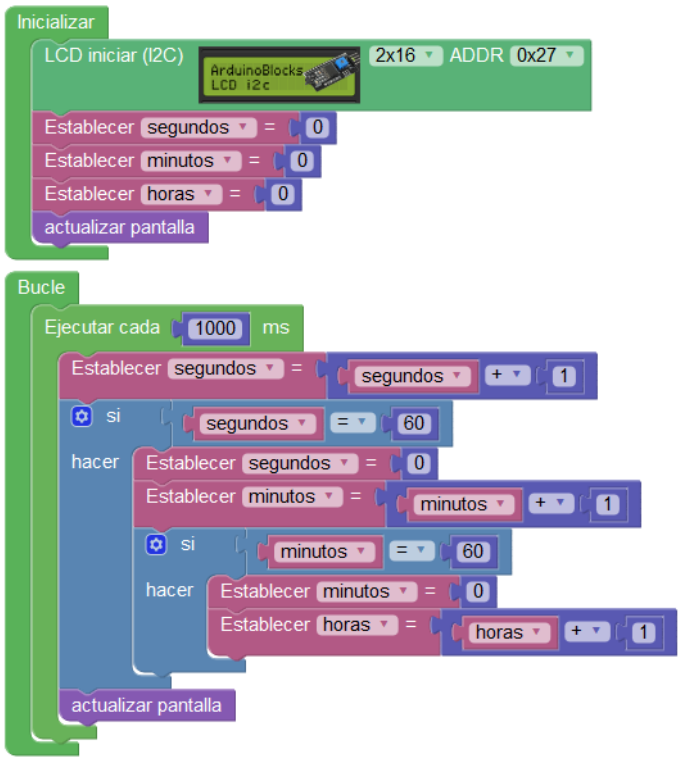
Material necesario:

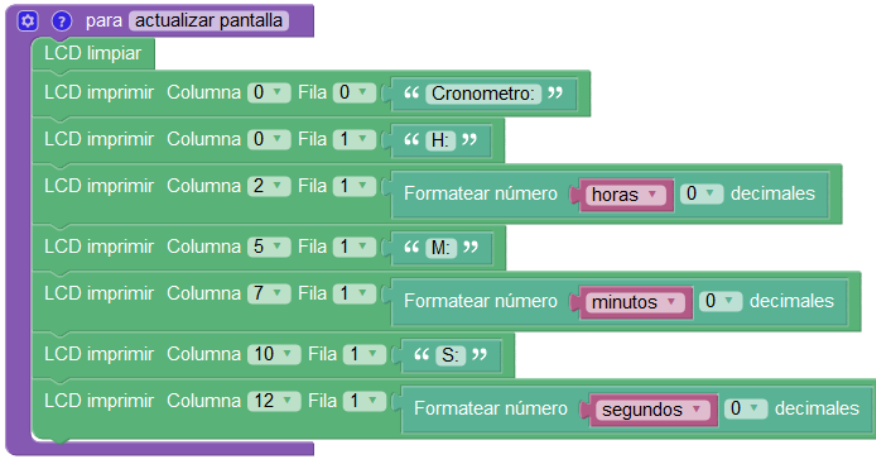
- 1 x pantalla LCD 2x16 (con módulo de conexión i2c)
- Placa de prototipos, cables de interconexión.



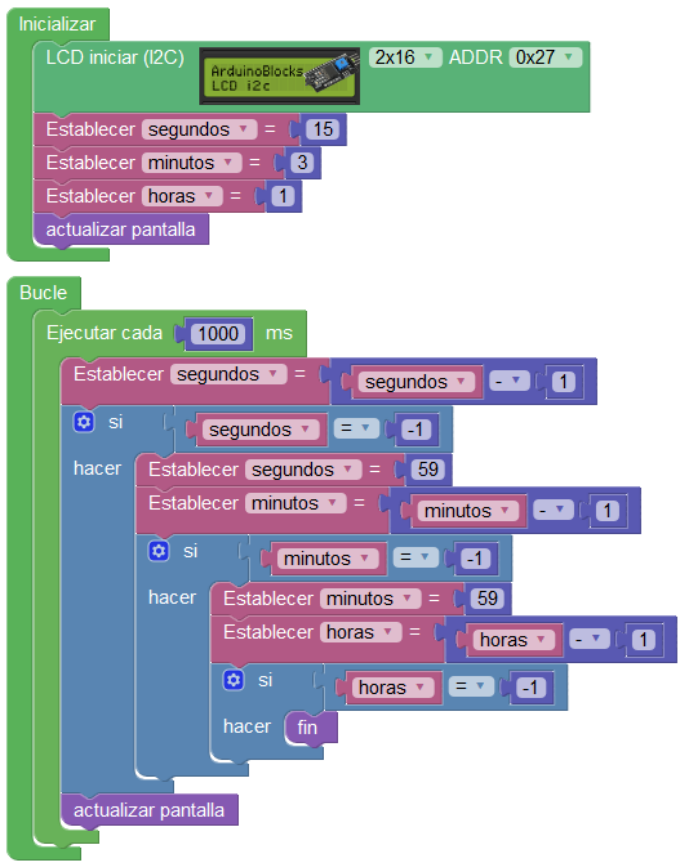
Conexiones LCD:
Pin SDA (A4)
Pin SCL (A5)

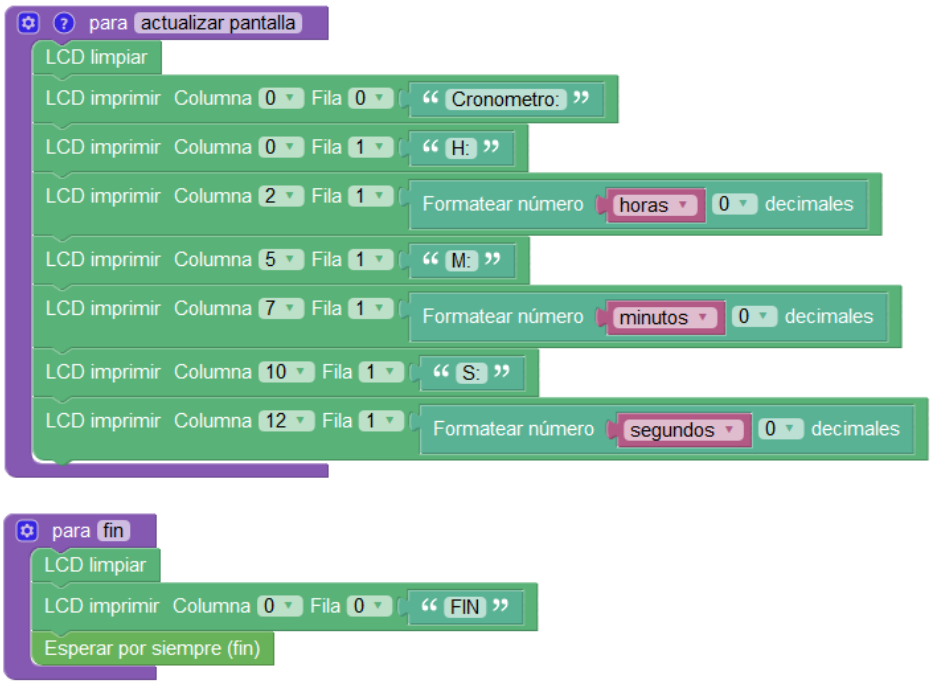
Programa ArduinoBlocks:



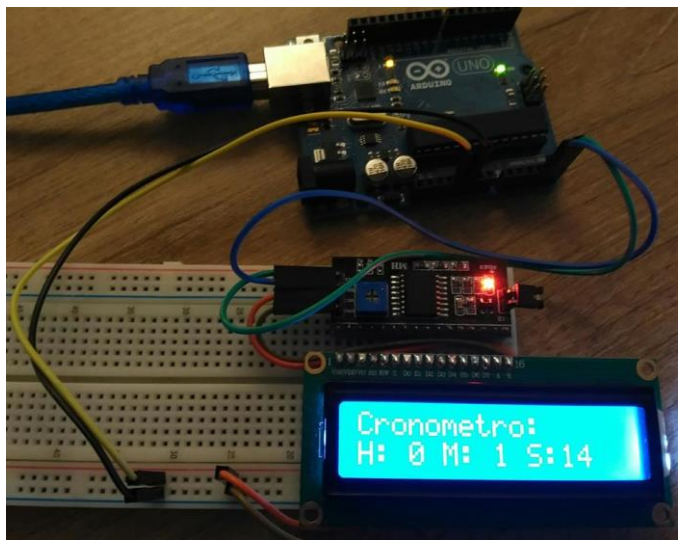


La versión para la cuenta atrás:
(iniciamos las variables horas, minutos y segundos al valor inicial deseado)





Vista real del montaje en funcionamiento:



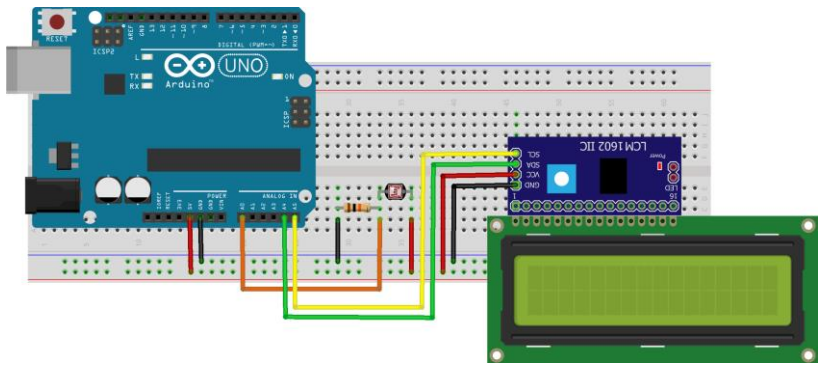
P10 - FOTÓMETRO

Un fotómetro es un dispositivo que nos permite medir la cantidad de luz ambiente. El valor se mostrará en una pantalla LCD.

Material necesario:

- 1 x pantalla LCD 2x16 (con módulo de conexión i2c)
- 1 x resistencia LDR 10kΩ
- 1 x resistencia 10kΩ
- Placa de prototipos, cables de interconexión

Conexiones:
LDR = Pin A0
LCD (i2c) = Pin SDA (A4) , Pin SCL (A5)



Programa ArduinoBlocks:

P11 - ILUMINACIÓN CREPUSCULAR

Un sistema de iluminación crepuscular permite el encendido automático de un sistema de iluminación cuando no hay suficiente luz ambiente natural (atardecer/anochecer) y de igual forma su apagado al tener la suficiente luz natural (amanecer).

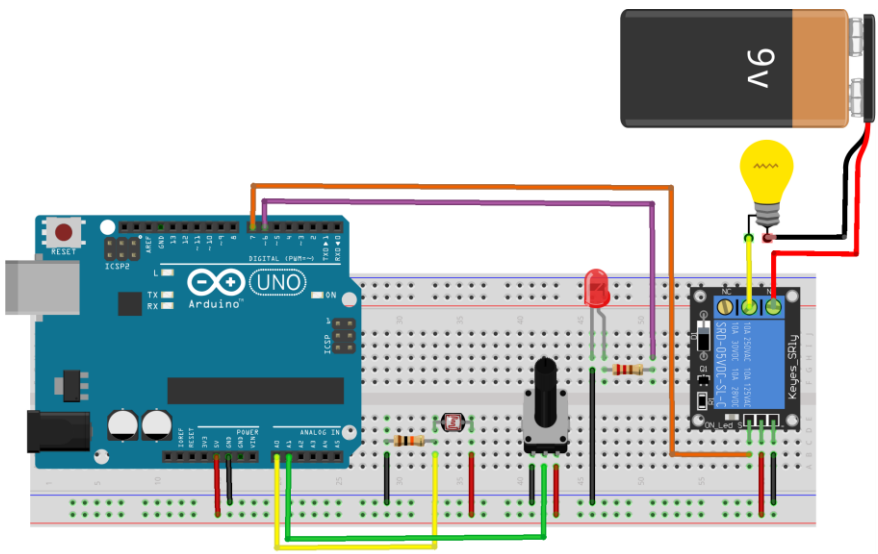
Mediante un potenciómetro podremos ajustar el nivel de luz ambiente (umbral) al que queremos que se encienda o apague el sistema de iluminación.

Material necesario:

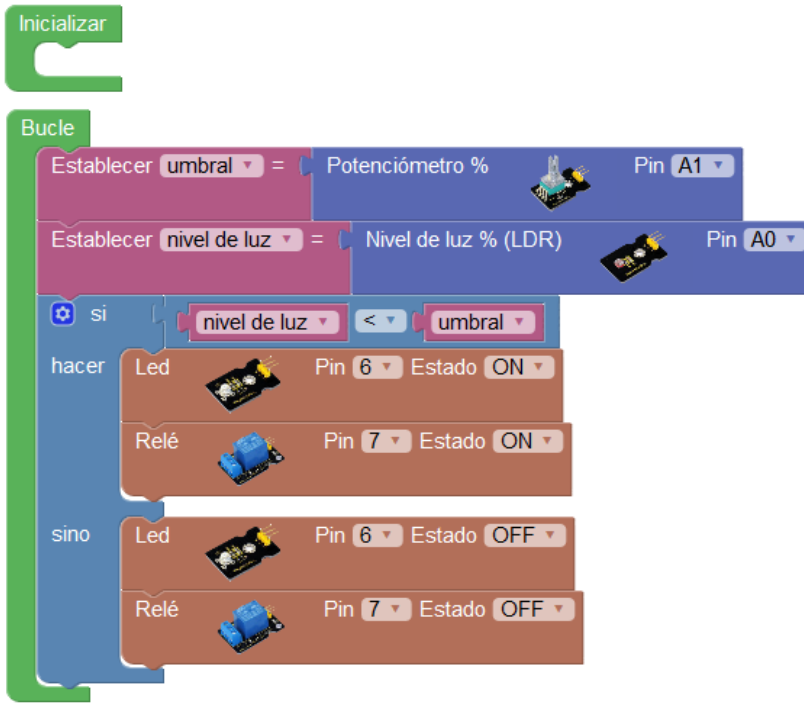
- 1 x resistencia LDR kΩ
- 1 x resistencia 10kΩ
- 1 x potenciómetro rotativo 10kΩ
- 1 x led
- 1 x resistencia 220Ω
- 1 x módulo de relé
- Placa de prototipos, cables de interconexión

Conexiones:

Potenciómetro = Pin A1
LDR = Pin A0
Led = Pin 6
Relé = Pin 7



Programa ArduinoBlocks:



En algún caso, si conectamos la LDR de forma distinta , el valor numérico obtenido será inverso al nivel de luz ambiente (a más luz menor número) por lo que el ajuste sería al contrario:

P12 - ENCENDIDO / APAGADO CON PALMADA

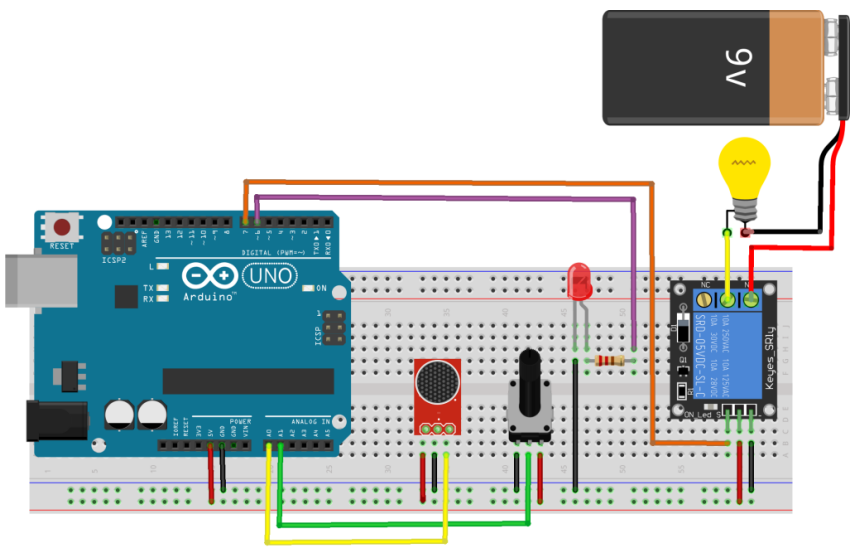
Este montaje nos permitirá sorprender a nuestros invitados en casa. Con un sonido intenso como el de una palmada podemos encender y apagar la luz de nuestra habitación.

Este sencillo sistema nos permite controlar la luz sin movernos del sofá. Como habrás comprobado no sólo sirve una palmada, cualquier sonido que supere el umbral configurado activará el sistema (la palmada nunca falla).

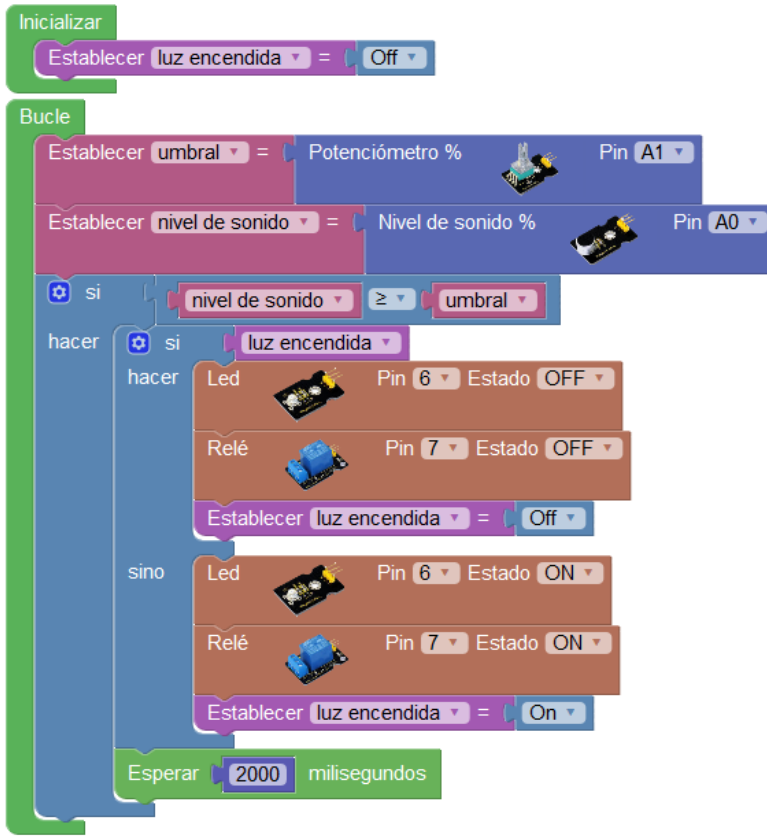
Material necesario:

- 1 x módulo de sensor de sonido
- 1 x potenciómetro rotativo 10kΩ
- 1 x led
- 1 x resistencia 220Ω
- 1 x módulo de relé
- Placa de prototipos, cables de interconexión

Conexiones:
Sensor de sonido = Pin A0
Potenciómetro = Pin A1
Led = Pin 6
Relé = Pin 7

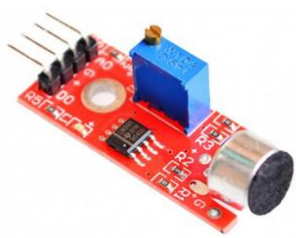


Programa ArduinoBlocks:



La espera de 2000 ms es para evitar encendidos y apagados muy consecutivos. En caso de detectar una palmada (o sonido fuerte) se esperará 2000ms hasta volver a poder detectar otro nuevo sonido. Este valor se puede ajustar.

Los módulos de sensor de sonido con salida analógica normalmente tienen un potenciómetro para ajustar la sensibilidad.



P13 - TERMÓMETRO

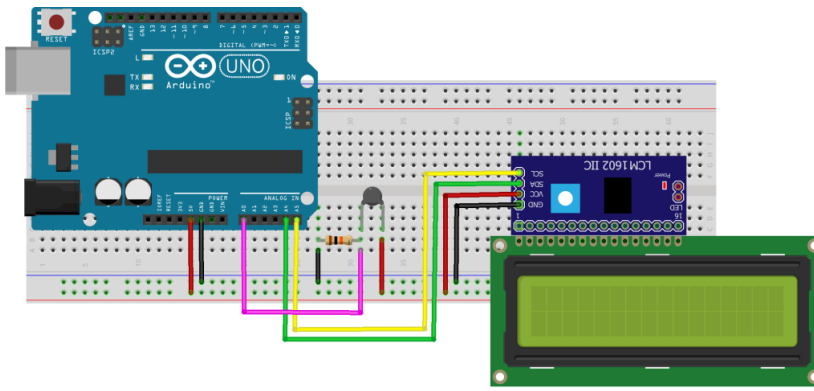
Construye tu propio termómetro digital con este sencillo montaje. La temperatura se visualiza cómodamente en la pantalla LCD.

El sensor de temperatura utilizado es una resistencia NTC.

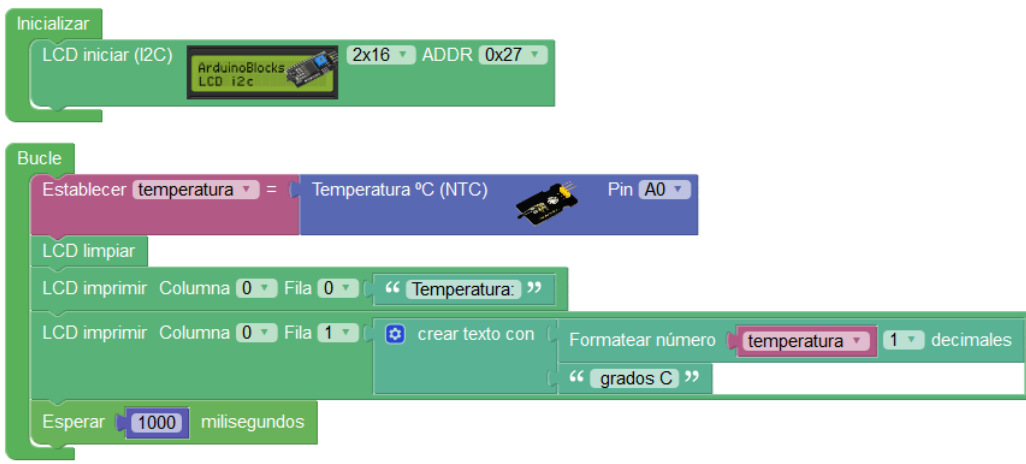
Material necesario:

- 1 x pantalla LCD 2x16 (con módulo de conexión i2c)
- 1 x resistencia NTC 10kΩ
- 1 x resistencia 10kΩ
- Placa de prototipos, cables de interconexión

Conexiones:
Resistencia NTC = Pin A0
LCD (i2c) = Pin SDA (A4) , Pin SCL (A5)



Programa ArduinoBlocks:



P14 - TERMOSTATO

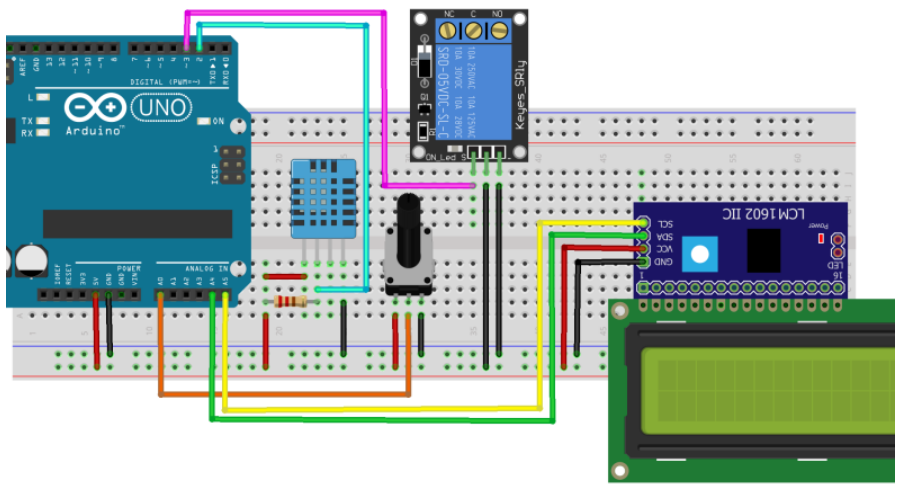
Un termostato permite controlar un sistema de calefacción (o de refrigeración) para actuar y conseguir la temperatura deseada en función de la temperatura ambiente.

El termostato de este montaje permite controlar un sistema de calefacción, activando la caldera (o cualquier otra fuente de calor como un radiador eléctrico) para conseguir la temperatura deseada cuando la temperatura ambiente sea inferior a la temperatura deseada (en el caso de un sistema de refrigeración sería al revés).

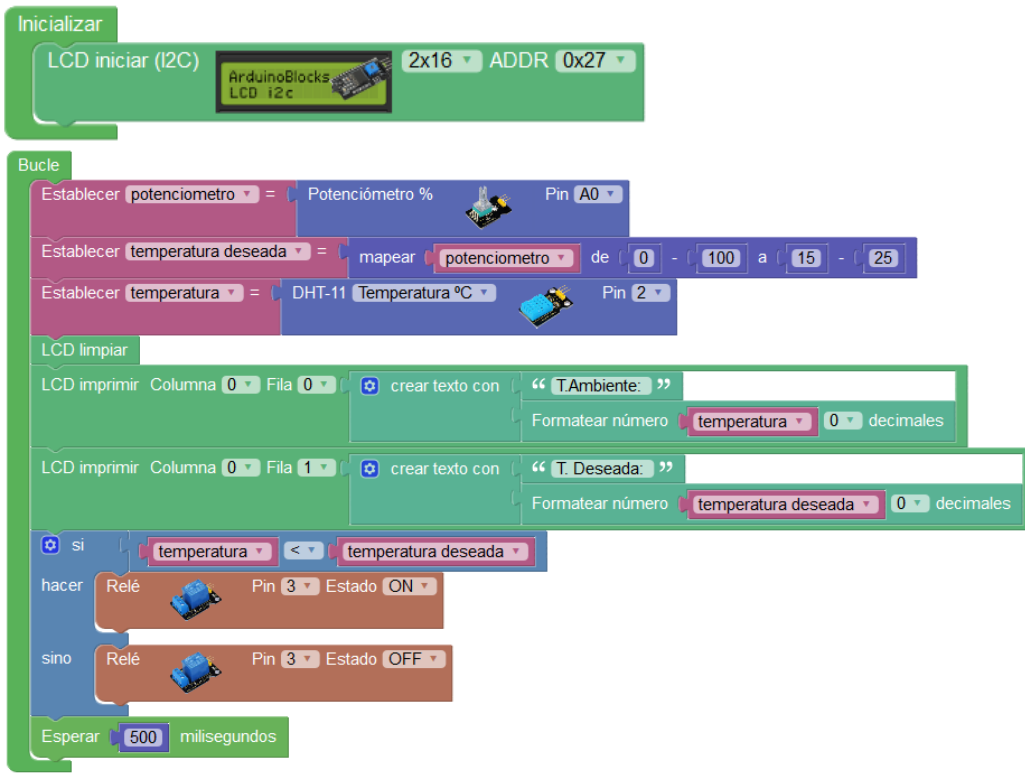
Material necesario:

- 1 x pantalla LCD 2x16 (con módulo de conexión i2c)
- 1 x sensor DHT-11
- 1 x potenciómetro 10kΩ
- 1 x módulo de relé
- Placa de prototipos, cables de interconexión

Conexiones:
Potenciómetro = Pin A0
Sensor DHT-11 = Pin 2
Relé = Pin 3
LCD (i2c): Pin SDA (A4) / Pin SCL (A5)

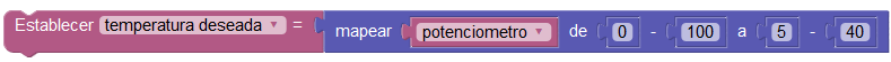


Programa ArduinoBlocks:



Con el potenciómetro podemos ajustar un valor entre 15 y 25º C

Si deseamos cambiar este rango debemos modificar el mapeo al rango deseado. Por ejemplo si queremos poder ajustar entre 5 y 40 º C:



Si quisiéramos realizar un termostato para enfriar (activando un ventilador o aire acondicionado), el funcionamiento sería inverso:

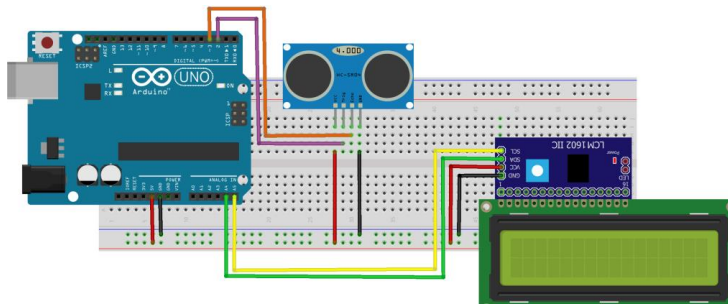
P15 - MEDIDOR DE DISTANCIA

Mediante el sensor de ultrasonidos y la pantalla LCD podemos realizar un dispositivo capaz de medir y visualizar la distancia hasta el objeto más cercano mostrando la distancia en cm.

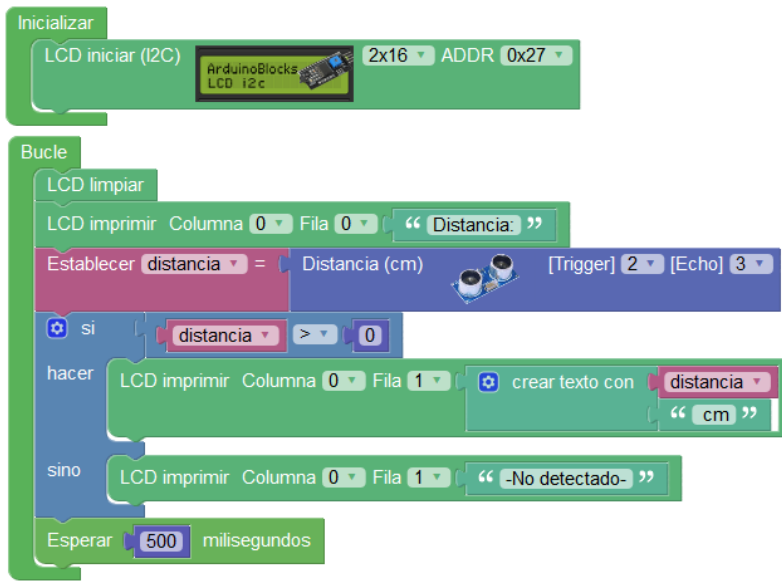
Material necesario:

- 1 x pantalla LCD 2x16 (con módulo de conexión i2c)
- 1 x sensor de ultrasonidos HC-SR04
- Placa de prototipos, cables de interconexión

Conexiones:
Sensor HC-SR04: Trigger = Pin 2 , Echo = Pin 3
LCD (i2c) = Pin SDA (A4) , Pin SCL (A5)



Programa ArduinoBlocks:

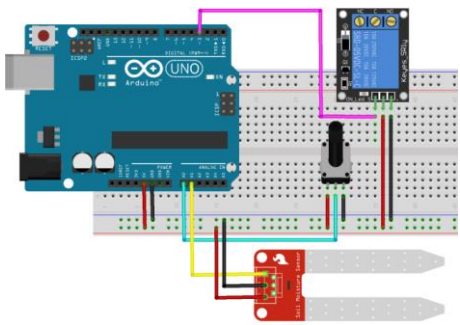


P16 - RIEGO AUTOMÁTICO

Mediante el sensor de humedad detectaremos el nivel de humedad de la tierra. Si el nivel de humedad es inferior al ajustado mediante un potenciómetro se activará una electroválvula (para regar) a través de un relé. El sistema comprueba la humedad una vez por minuto.

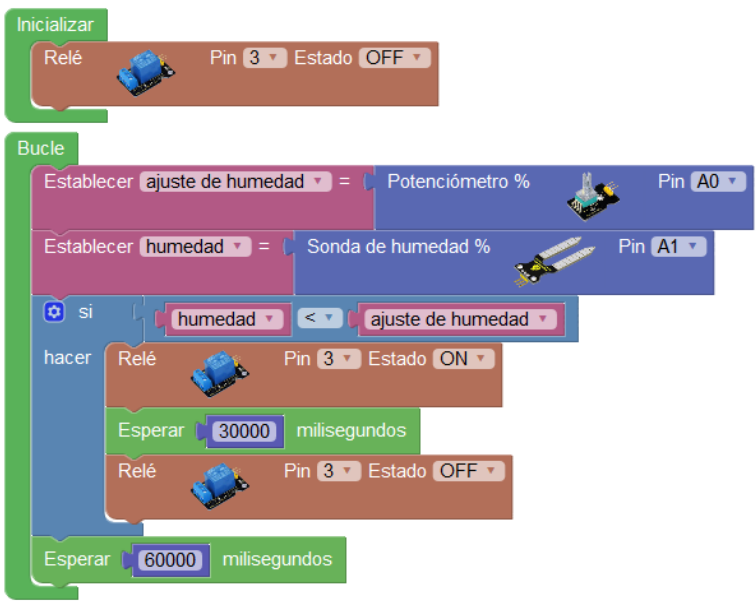
Material necesario:

- 1 x sensor de humedad
- 1 x potenciómetro rotativo
- 1 x módulo de relé
- Placa de prototipos, cables de interconexión



Conexiones:
Sensor humedad = Pin A1
Potenciómetro = Pin A0
Relé = Pin 3

Programa ArduinoBlocks:

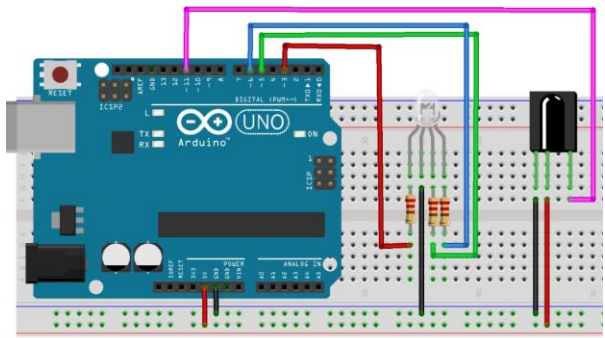


P17 - LÁMPARA MULTICOLOR CONTROLADA CON MANDO IR

Esta lámpara permitirá ajustar el ambiente a cada momento. Con la ayuda de un mando a distancia IR podremos cambiar el color y la intensidad a nuestro gusto.

Material necesario:

- 1 x led RGB (cátodo común)
- 1 x sensor IR
- 1 x mando IR
- Placa de prototipos, cables de interconexión

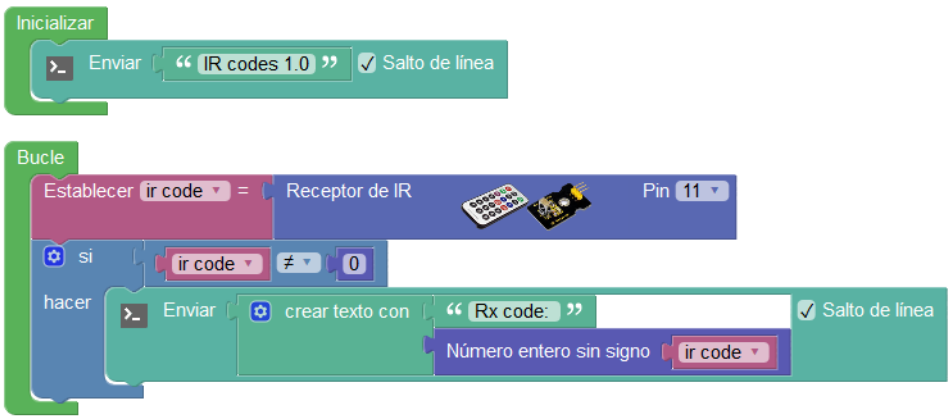


Conexiones:
Led R = Pin ~3
Led G = Pin ~5
Led B = Pin ~6
Sensor IR = Pin 11

Tecla "1"	16724175
Tecla "2"	16718055
Tecla "3"	16743045
Tecla "4"	16716015

Programa ArduinoBlocks:

Antes de realizar este montaje es recomendable obtener los códigos para cada botón del mando a distancia utilizado. Esto se puede realizar fácilmente con este programa para obtener los códigos de cada botón por la consola serie:



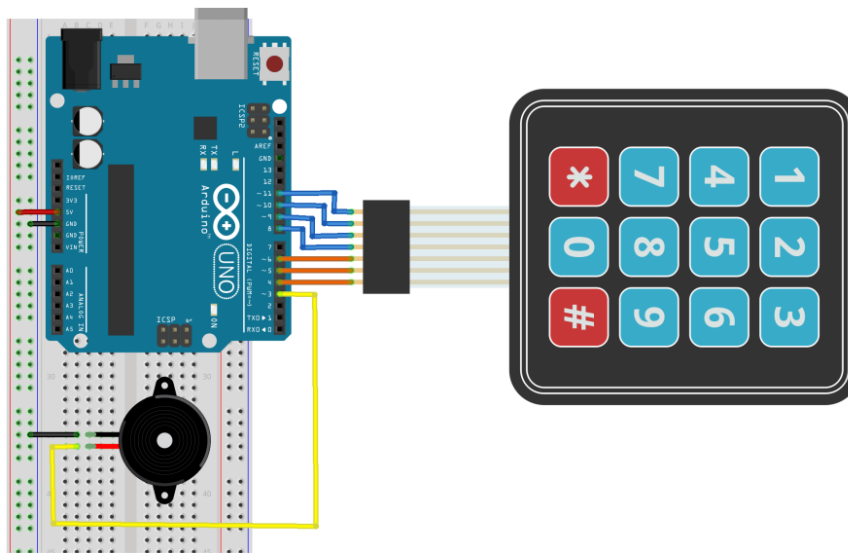
P18 - PIANO CON TECLADO

Un sencillo piano digital para poder tocar y componer nuestras propias melodías. Cada tecla del keypad reproducirá un tono en el zumbador conectado.

Material necesario:

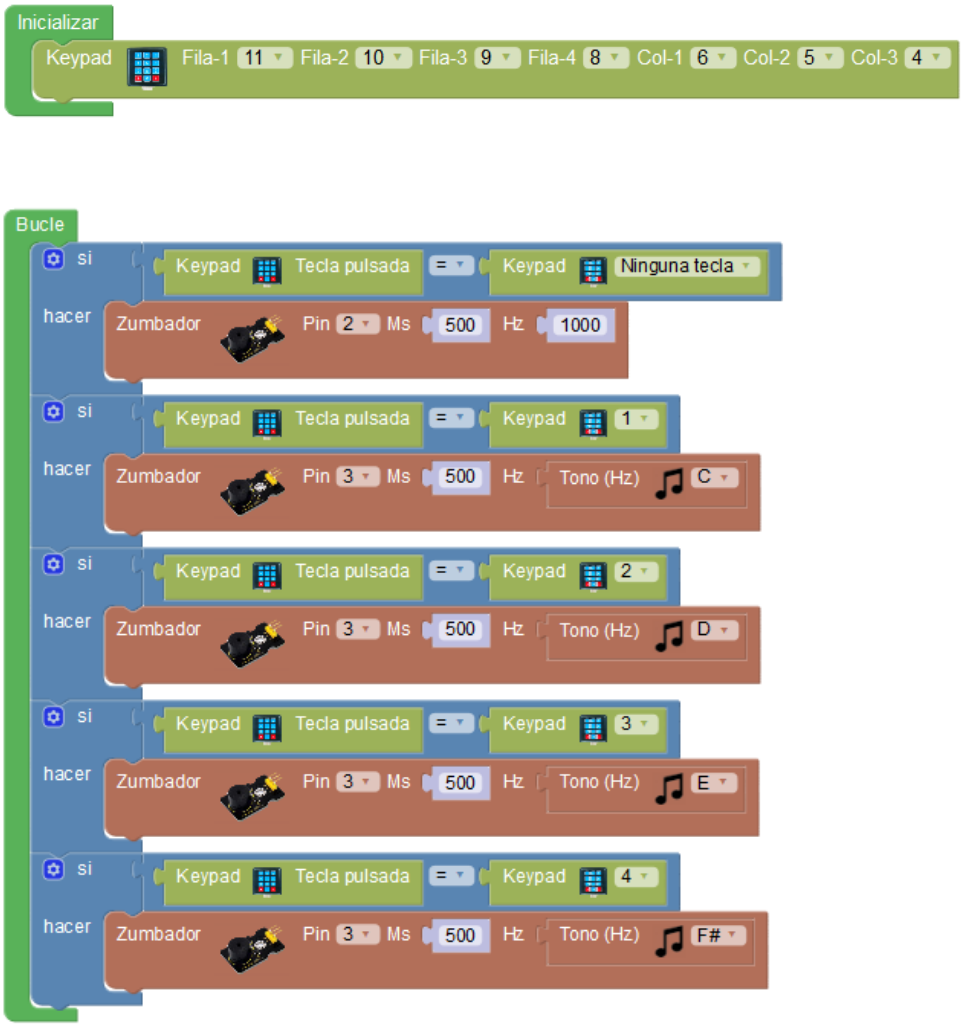
- 1 x zumbador pasivo
- 1 x keypad 3x4
- Placa de prototipos, cables de interconexión

Conexiones:
Keypad: Fila 1 = Pin 11, Fila 2 = Pin 10, Fila 3 = Pin 9, Fila 4 = Pin 8
Keypad: Col-1 = Pin 6, Col-2 = Pin 5, Col-3 = Pin 4
Zumbador = Pin ~3



Ejemplos de zumbadores pasivo para utilizar en este proyecto.

Programa ArduinoBlocks:

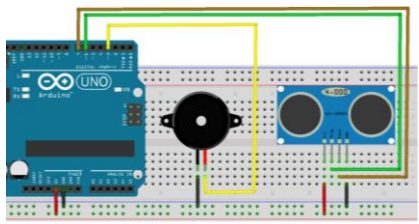


P19 - SENSOR DE APARCAMIENTO

Hoy en día todos los coches modernos incorporan un sensor de aparcamiento. Este sensor nos permite detectar los objetos que se encuentran delante y detrás del vehículo para evitar colisionar. De una forma sencilla podemos crear nuestro propio sensor de aparcamiento.

Material necesario:

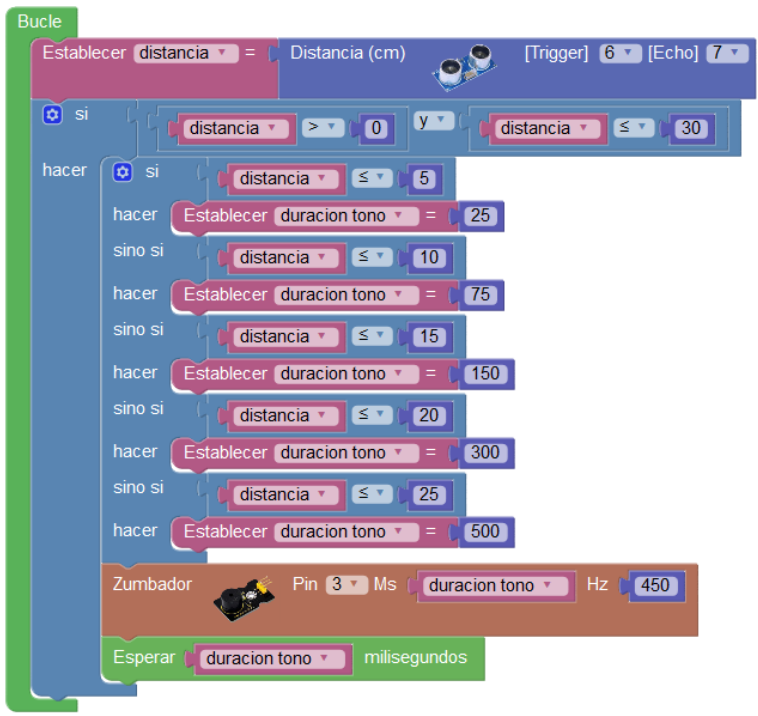
- 1 x zumbador
- 1 x sensor de ultrasonidos HC-SR04
- Placa de prototipos, cables de interconexión



Conexiones:
Sensor HC-SR04:
Trigger = Pin 6
Echo = Pin 7

Zumbador = Pin ~3

Programa ArduinoBlocks:



P20 - CONTROL PAN CON JOYSTICK

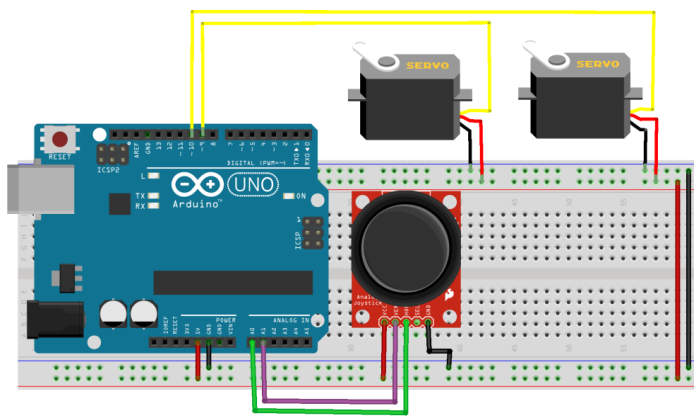
El movimiento conocido como pan/tilt (horizontal/vertical) permite controlar la posición en dos ejes. Este tipo de controles se utiliza comúnmente para mover cámaras de seguridad, detectores de obstáculos, etc.

Existen pequeños mecanismo que implementan la función de pan/tilt gracias a la integración de dos servos.

Material necesario:

- 2 x micro-servos + mecanismo pan/tilt
- 1 x módulo joystick
- Placa de prototipos, cables de interconexión

Conexiones:
Servo pan = Pin ~9
Servo tilt = Pin ~10
Joystick X = Pin A0
Joystick Y = Pin A1



Programa ArduinoBlocks:

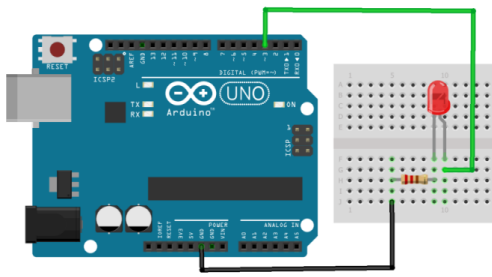
P21 - CONTROL DE UN LED DESDE PC

Vamos a realizar el control de la iluminación de un led controlando su intensidad desde un PC (desde el monitor serie del PC)

La idea es sencilla, recibimos un valor a través del terminal serie entre Arduino y el PC y cambiamos la intensidad del led al valor recibido (recuerda que como es un valor para la salida PWM debe ser un valor entre 0 a 255)

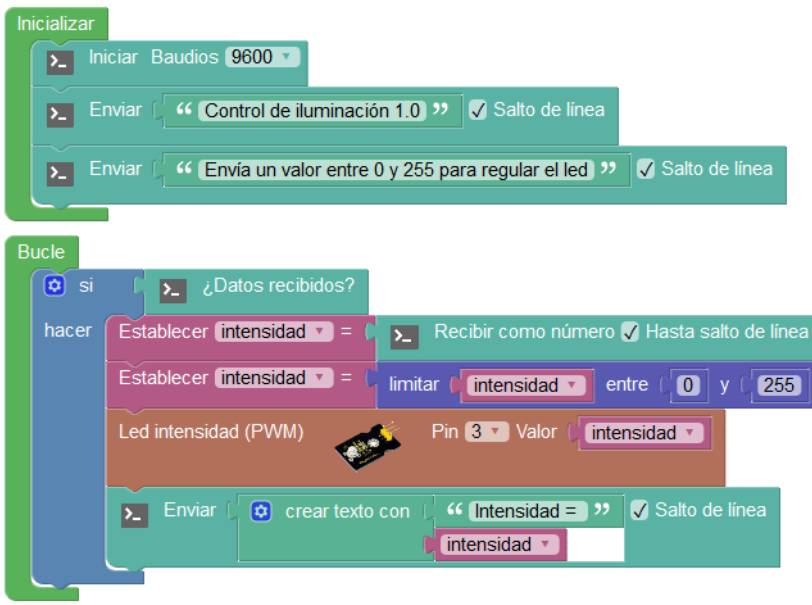
Material necesario:

- 1 x led
- 1 x resistencia 220Ω
- Placa de prototipos, cables de interconexión



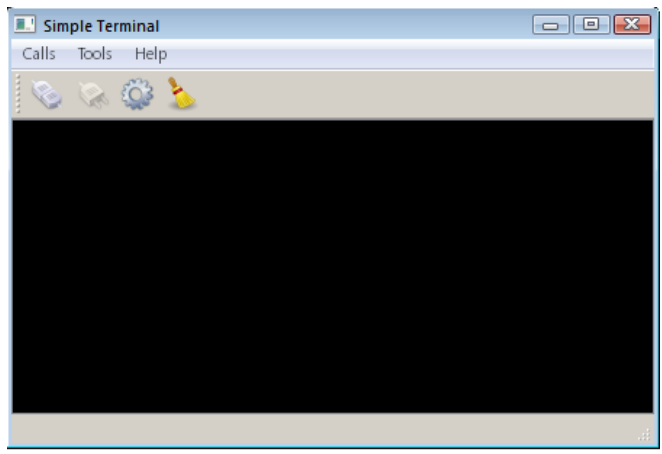
Conexiones:
Led = Pin ~3

Programa ArduinoBlocks:



Ejemplo de control desde la consola de ArduinoBlocks:

Podemos utilizar otras aplicaciones de consola serie:



P22 - CONTROL DE RELÉS POR BLUETOOTH

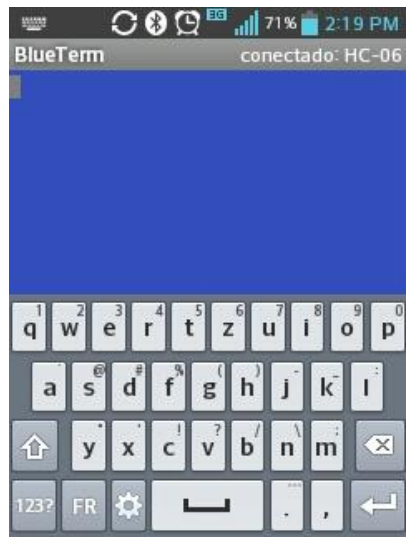
La comunicación inalámbrica Bluetooth apareció en los dispositivos móviles hace varios años y nos permite de una forma sencilla y rápida transferir información entre dispositivos a una distancia de hasta 100m.

Existen diferentes módulos para Arduino que nos permiten la utilización de la conexión Bluetooth, ArduinoBlocks es compatible con el módulo HC-06.

(repasa la utilización del módulo Bluetooth en el apartado 3.3.4)

Para el envío y recepción de datos utilizaremos una consola serie bluetooth desde algún dispositivo móvil como un Smartphone o Tablet.

Ejemplo: Aplicación "BlueTerm" para dispositivos Android

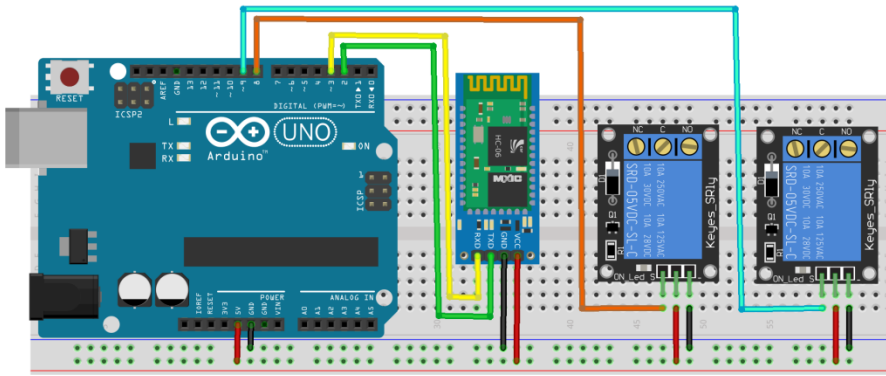


https://play.google.com/store/apps/details?id=es.pymasde.blueterm&hl=es

Material necesario:

- 1 x módulo Bluetooth HC-06
- 2 x módulo relé
- Dispositivo móvil con conexión Bluetooth
- Placa de prototipos, cables de interconexión

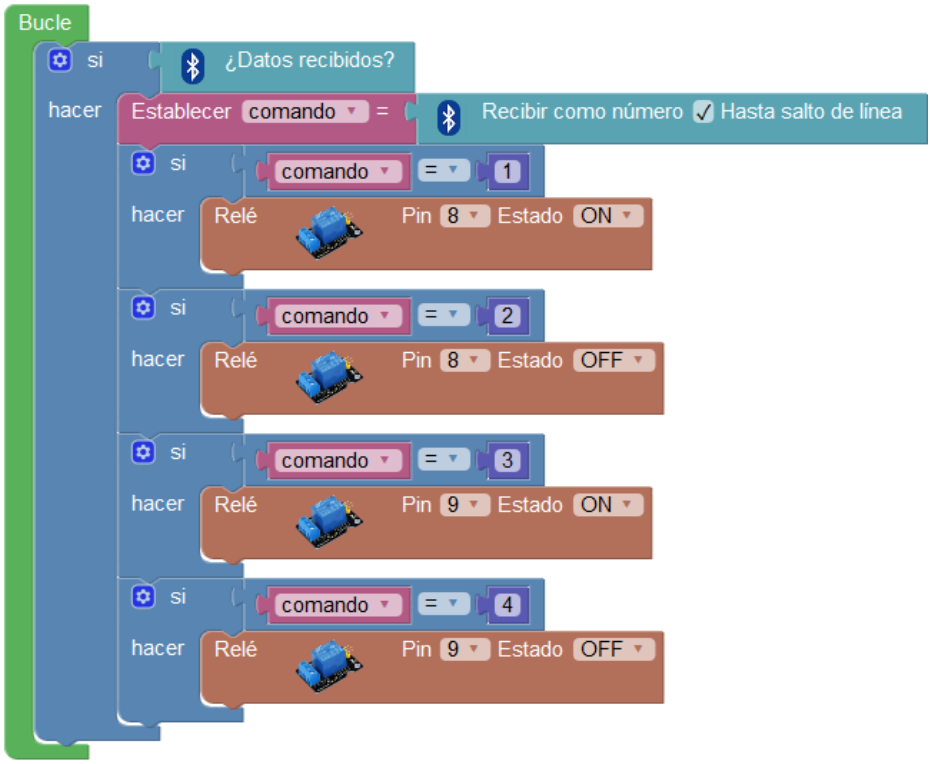
Conexiones:
Bluetooth RX = Pin 2, TX = Pin 3
Relé 1 = Pin 8
Relé 1 = Pin 9



Para el control por Bluetooth vamos a implementar un protocolo de comunicación muy sencillo donde cada comando es un número que realizará una función:

Comando (número)	*Función*
1	*Relé 1 = ON*
2	*Relé 1 = OFF*
3	*Relé 2 = ON*
4	*Relé 2 = OFF*

Programa ArduinoBlocks:



Módulo de relés utilizado en el proyecto:

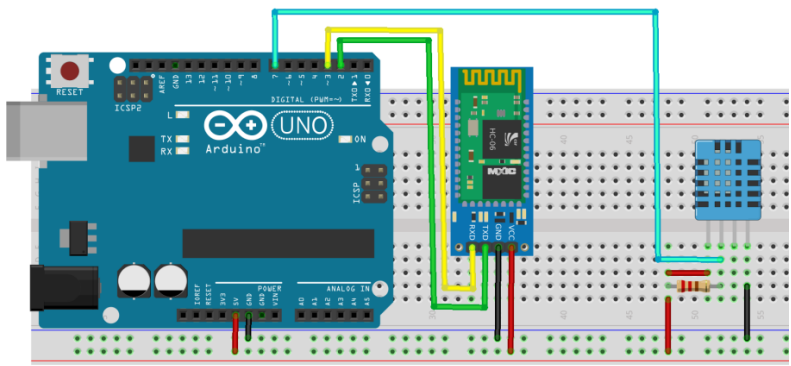
P23 - ESTACIÓN METEOROLÓGICA BLUETOOTH

En proyectos anteriores hemos visto como obtener la temperatura y la humedad fácilmente con el sensor DHT-11. Con este proyecto vamos a aplicar esta idea pero pudiendo monitorizar los datos de temperatura y humedad remotamente, así podemos tener la central meteorológica en un lugar alejado, como por ejemplo en la terraza de casa, y los datos los podemos visualizar cómodamente en el interior en un dispositivo móvil con conexión Bluetooth.

Material necesario:

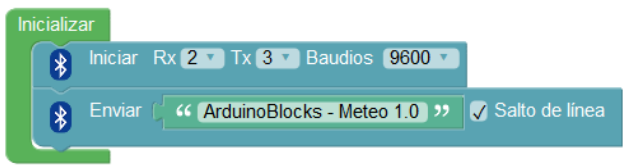
- 1 x sensor DHT-11
- 1 x módulo Bluetooth HC-06
- 1 x dispositivo móvil con conexión Bluetooth
- Placa de prototipos, cables de interconexión

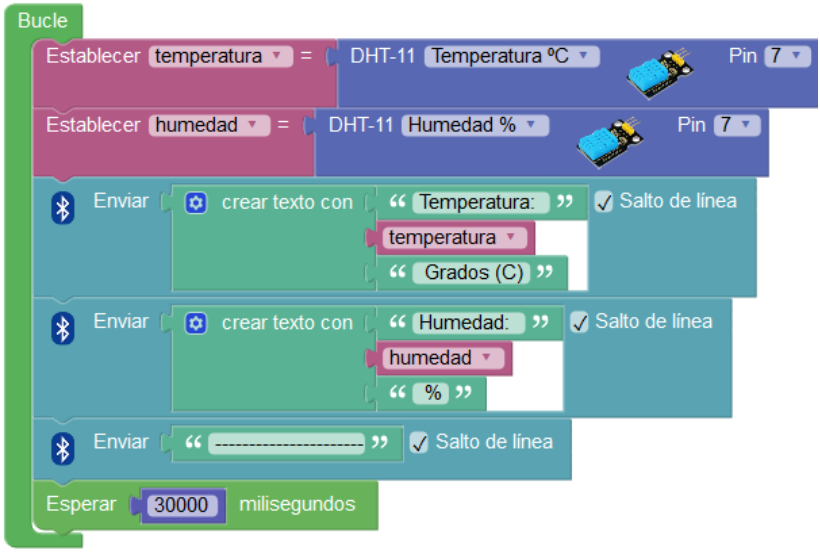
Conexiones:
Bluetooth RX = Pin 2 / TX = Pin 3
Sensor DHT-11 = Pin 7



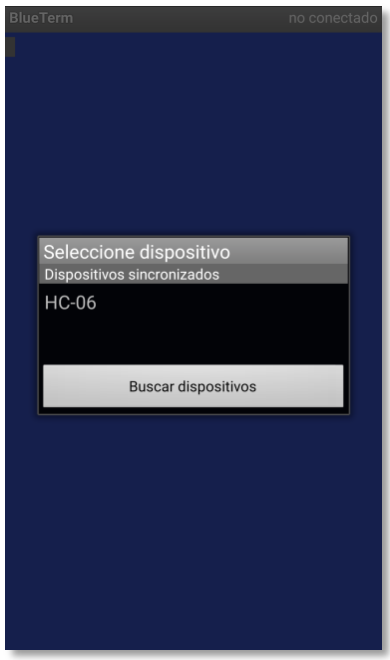
El programa es sencillo, cada 30 segundos enviamos a través de la conexión Bluetooth los datos de temperatura y humedad. En una aplicación tipo “BlueTerm” de Android podemos recibir y visualizar los datos en tiempo real.

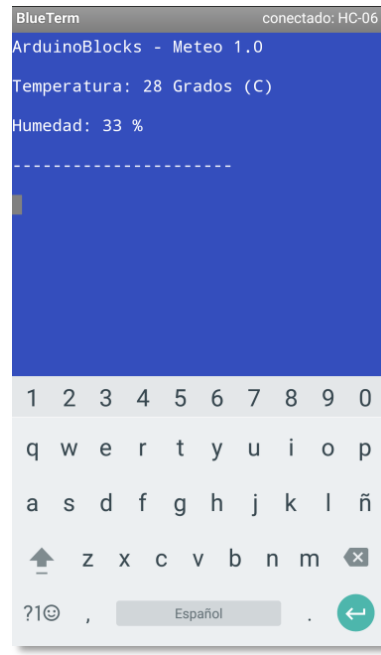
Programa ArduinoBlocks:





Visualización desde la aplicación BlueTerm en Android:





P24 - CONTROL DE LED POR VOZ (ANDROID+BLUETOOTH)

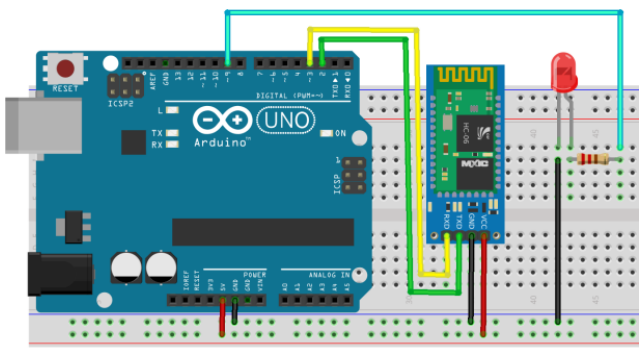
El terminal Bluetooth es muy útil y fácil de utilizar pero en algunas ocasiones necesitamos crear nuestra propia interfaz de control en el dispositivo. Para crear aplicaciones fácilmente en Android disponemos de la extraordinaria herramienta *AppInventor*. Esta plataforma nos permite crear aplicaciones Android de forma visual y programarla con lenguaje de bloques. Lo único que necesitamos es una cuenta de Google para poder utilizarla.

El siguiente proyecto utiliza una aplicación muy sencilla creada en *AppInventor* que reconoce la voz y enviará un comando u otro a través de Bluetooth para encender o apagar un led.

Comando de voz	*Comando enviado*	*Acción de Arduino*
"encender"	1	Encender el led
"apagar"	2	Apagar el led
"parpadear"	3	Parpadea 3 segundos

Material necesario:

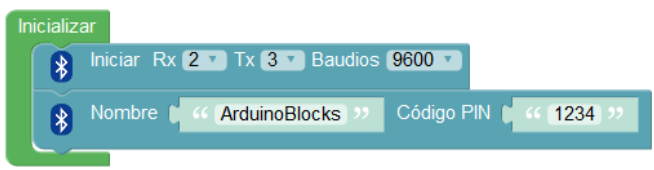
- 1 x led
- 1 x resistencia 220Ω
- 1 x módulo Bluetooth HC-06
- 1 x dispositivo móvil con sistema Android
- Placa de prototipos, cables de interconexión

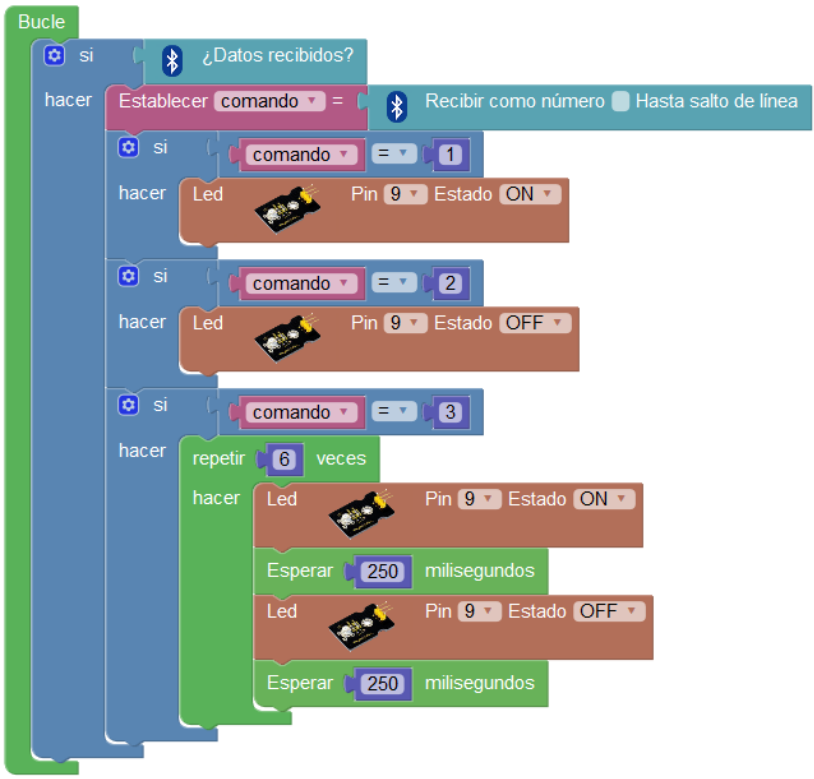


Conexiones:
Bluetooth:
RX = Pin 2
TX = Pin 3

Led = Pin ~9

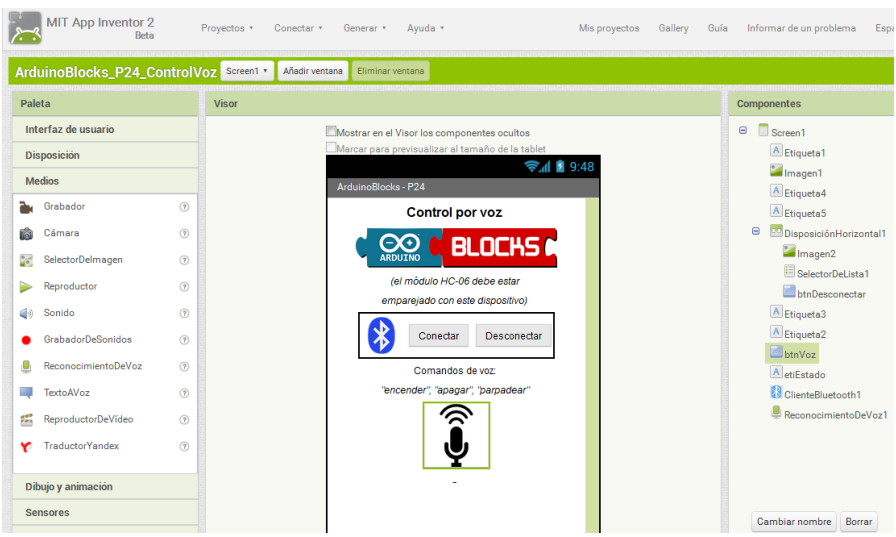
Programa ArduinoBlocks:



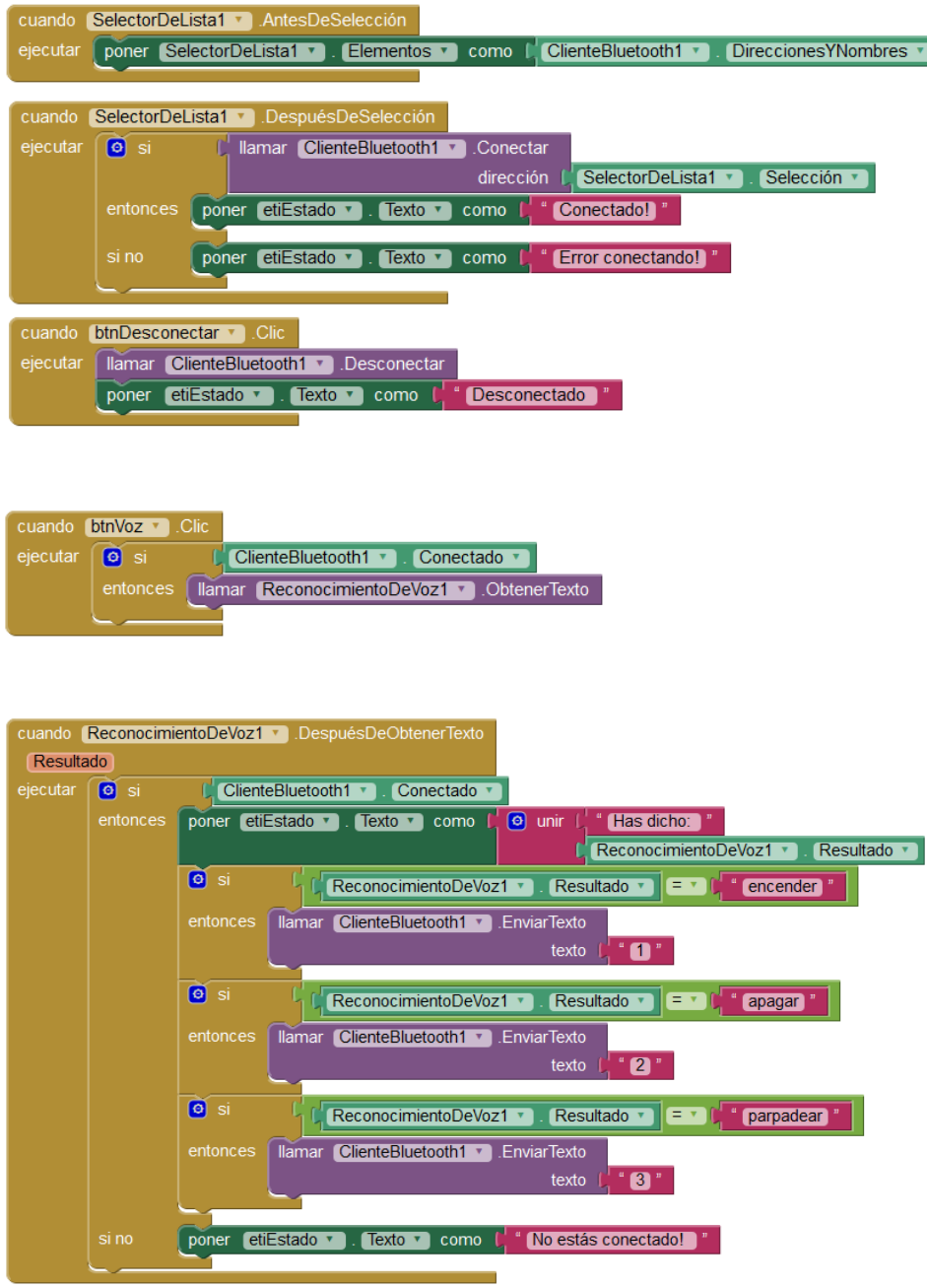


(no se ha marcado la opción "Hasta salto de línea" en la recepción Bluetooth porque desde la aplicación Android no enviamos salto de línea)

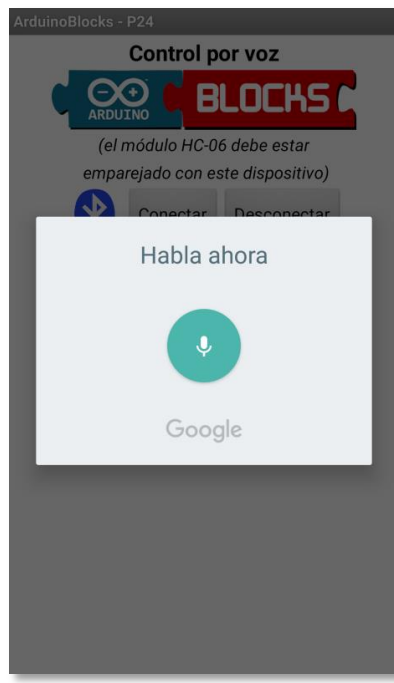
Diseño de la interfaz de la aplicación Android con *AppInventor*.



Código de la aplicación Android con *AppInventor*.



Aplicación AppInventor funcionando:



P25 - CONTROL DOMÓTICO (ANDROID+BLUETOOTH)

Con todo lo aprendido en los proyectos anteriores vamos a implementar un proyecto un poco más complejo implementando un control domótico para casa.

La domótica es la aplicación de la automatización y la robótica en el hogar. La domótica debe cumplir funciones de confort, seguridad y ahorro energético.

Cada día aparecen más sistemas domóticos pero gracias a Arduino podemos crearnos nuestros sistemas propios de automatización y programar el funcionamiento de la aplicación según nuestras necesidades.

Este proyecto implementa un sistema muy sencillo de automatización, pero nos sirve como una aproximación para entender cómo funcionan estos sistemas.

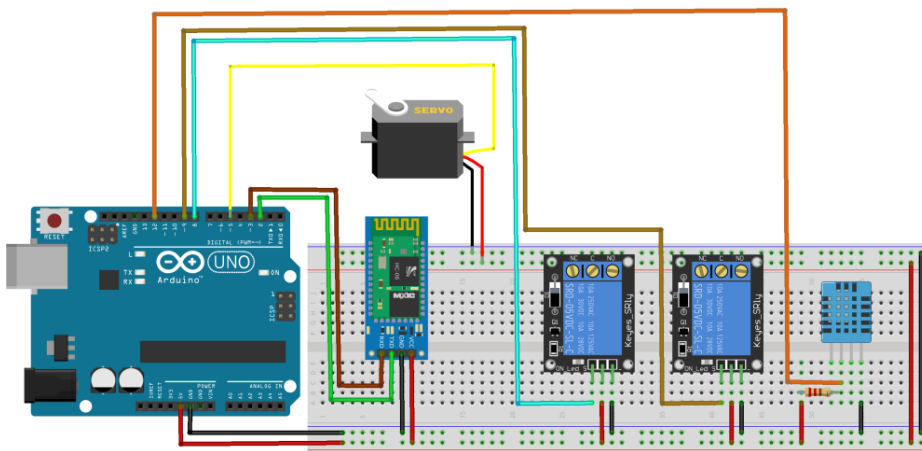
El control domótico con Arduino y Android incluirá:

-Control de dos relés para iluminación ON/OFF
-Monitorización de temperatura desde dispositivo móvil
-Control de una persiana (simulada con un servo)
-Escenas (confort, apagar todo, simular presencia)
-Control desde dispositivo móvil Android vía Bluetooth

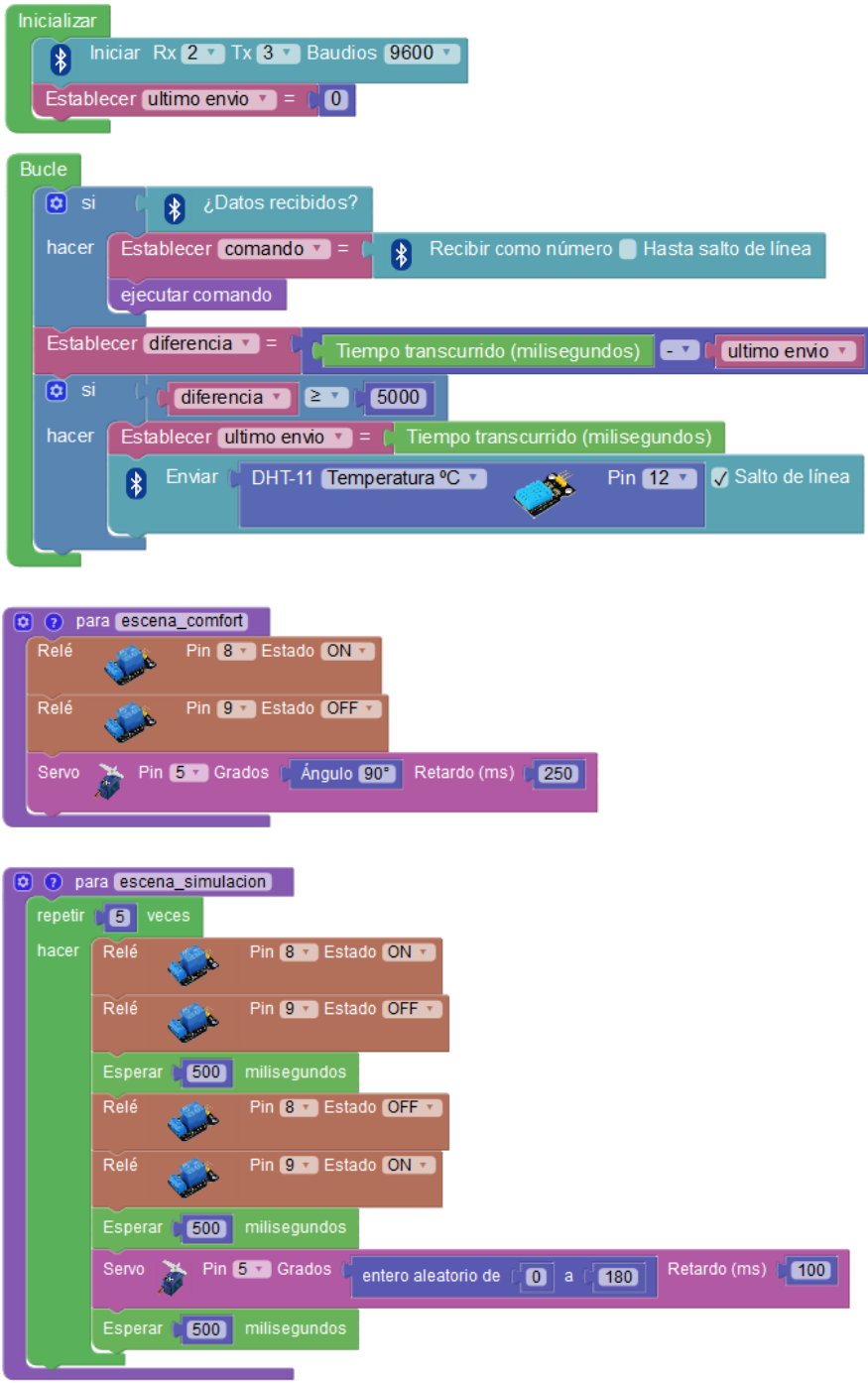
Material necesario:

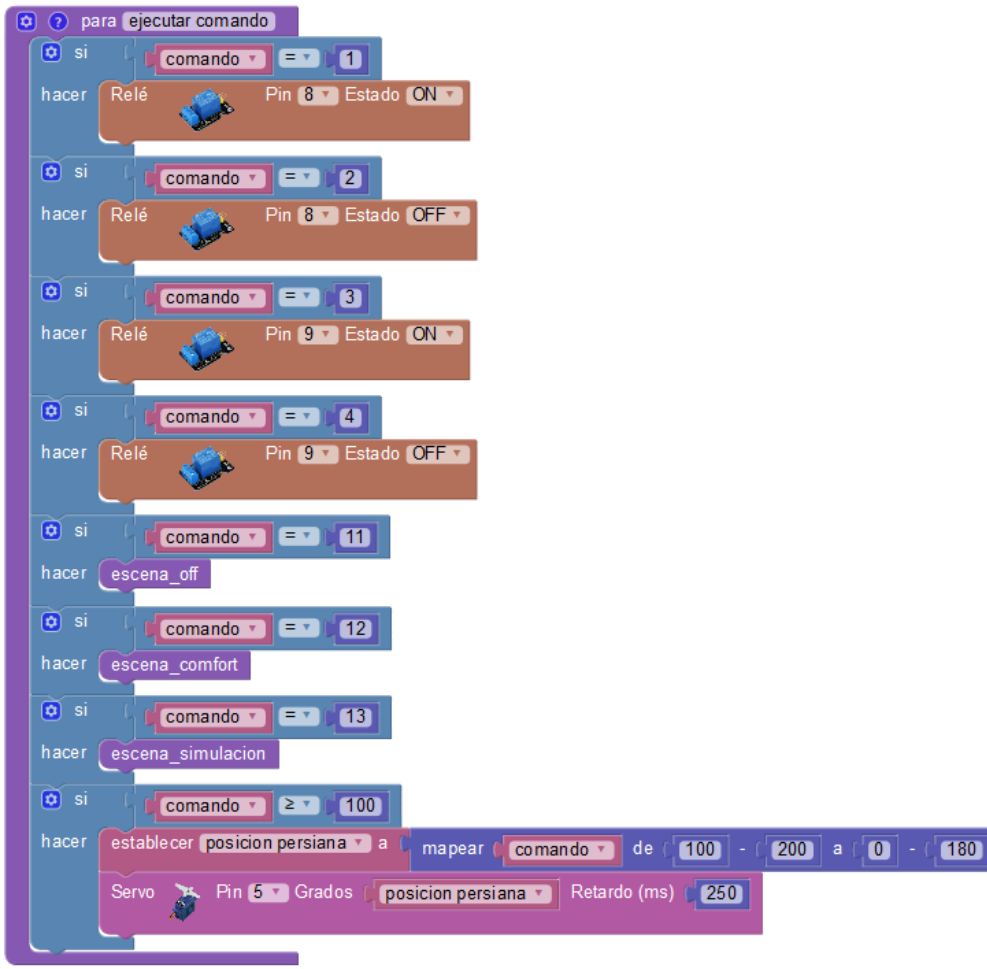
- 1 x módulo Bluetooth HC-06
- 1 x sensor DHT-11
- 1 x servomotor
- 2 x módulo relé
- Placa de prototipos
- Cables de interconexión

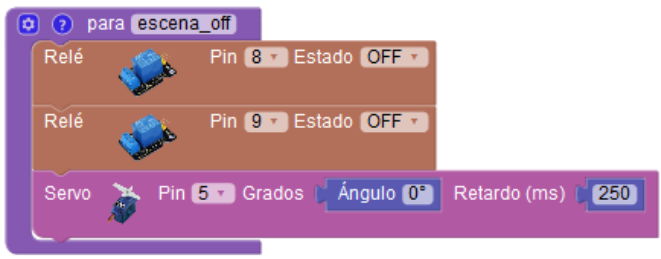
Conexiones:
Bluetooth RX = Pin 2
Bluetooth TX = Pin 3
Sensor DHT-11 = Pin 12
Relé 1 = Pin 8
Relé 2 = Pin 9
Servo = Pin ~5



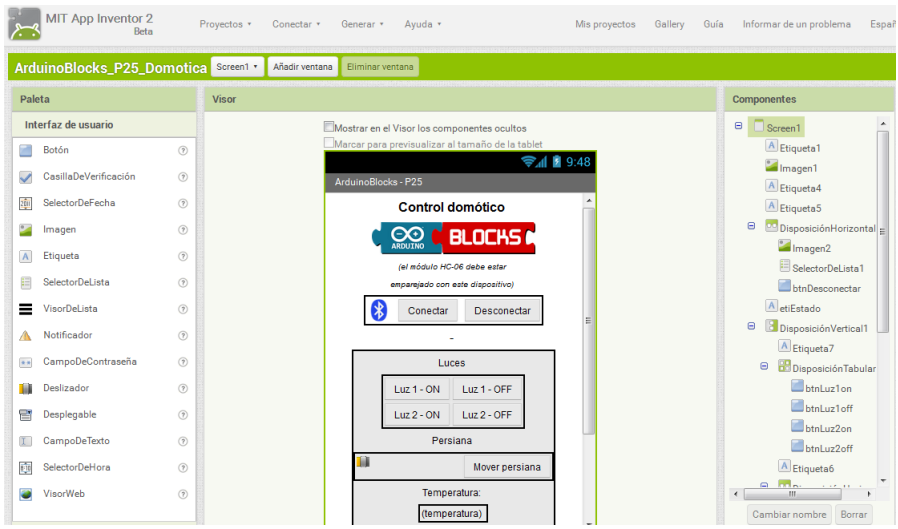
Programa ArduinoBlocks:



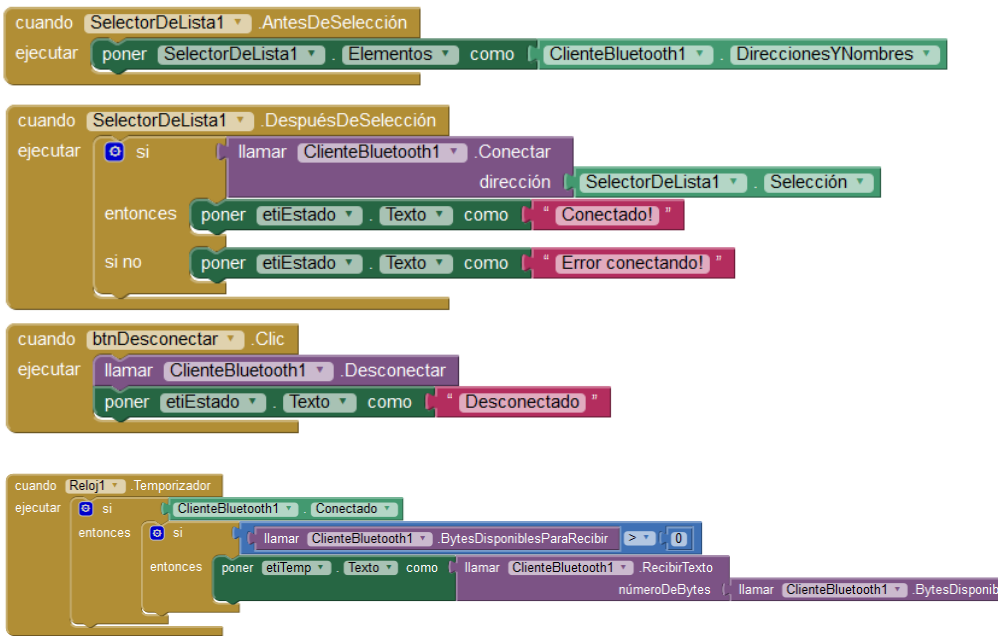


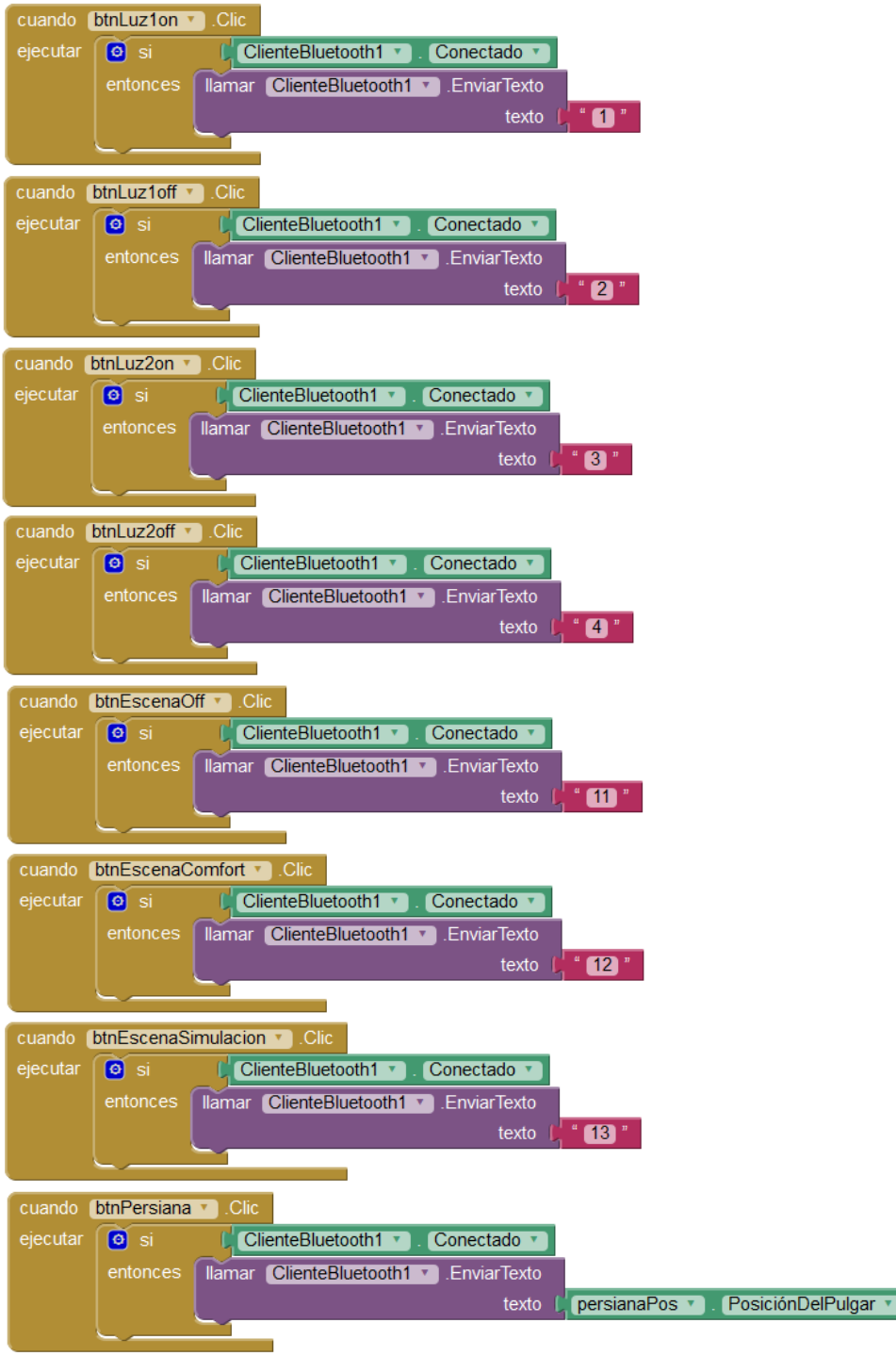


Diseño de la interfaz de la aplicación *Android* con *AppInventor*.

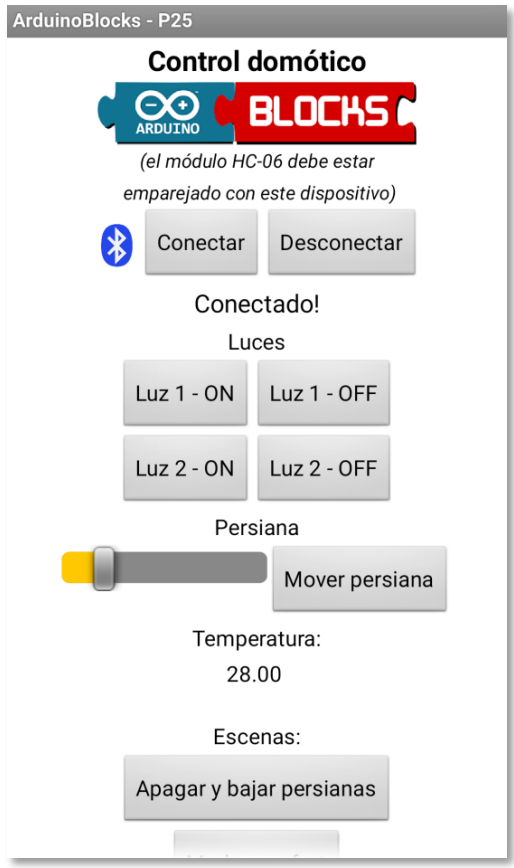


Código de la aplicación Android con AppInventor:





Aspecto final de la aplicación de control y monitorización:



P26 - GPS CON VISUALIZACIÓN LCD

El siguiente proyecto nos permite visualizar la información de obtenida desde el módulo GPS en una pantalla LCD. Este proyecto nos permitiría por ejemplo añadir un indicador de velocidad, altitud, posición, etc. a nuestra bicicleta o motocicleta.

Cada 5s se mostrarán unos datos diferentes en la pantalla:

-Pantalla 1: Información de latitud y longitud
-Pantalla 2: Velocidad y altitud
-Pantalla 3: Fecha y hora recibida del satélite GPS

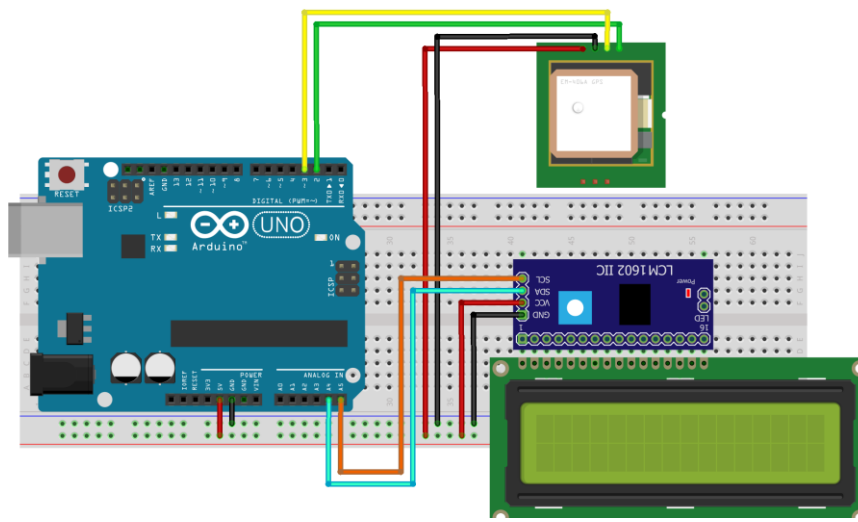
Para un correcto funcionamiento el módulo GPS debe estar preferiblemente en un espacio a cielo abierto y puede tardar unos minutos en obtener información válida.

Material necesario:

- 1 x módulo GPS
- 1 x LCD con conexión I2C
- 1 x Arduino UNO
- Placa de prototipos
- Cables de interconexión

Conexiones:
GPS RX = Pin 2
GPS TX = Pin 3
LCD = I2C

Esquema de conexión:



Programa ArduinoBlocks:

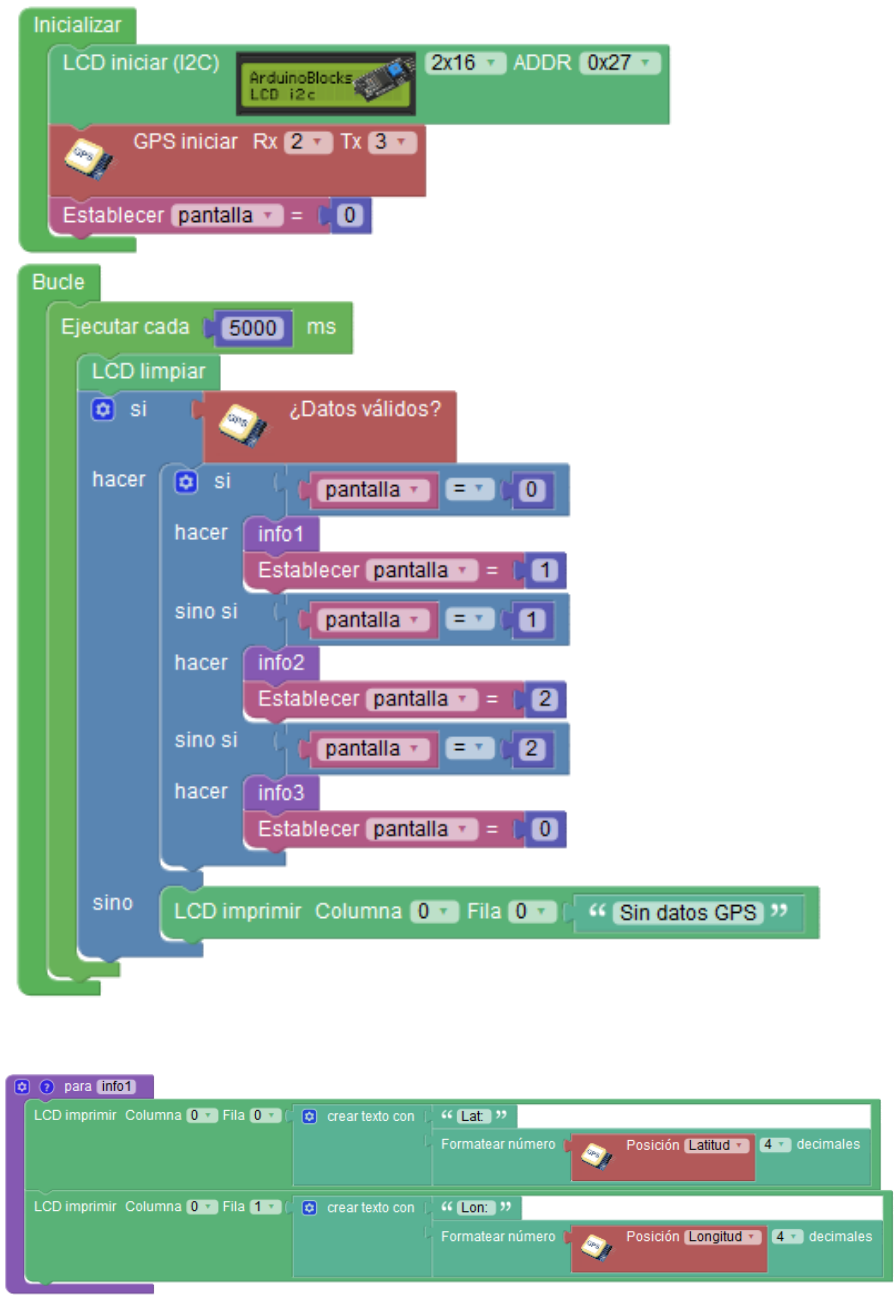
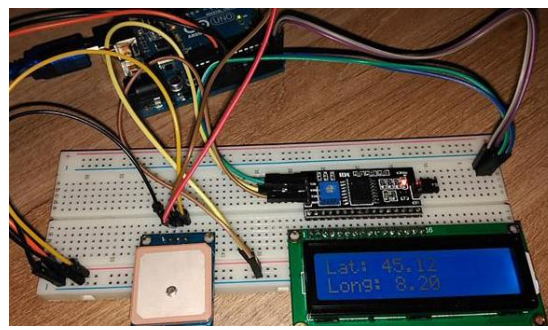


Imagen real del montaje:



P27 - AVISO POR EXCESO DE VELOCIDAD

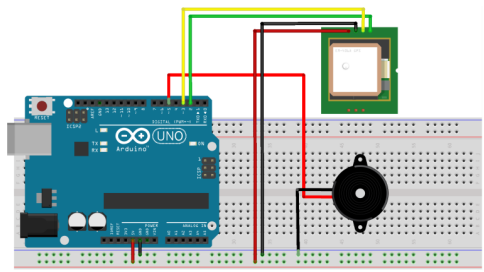
El siguiente montaje nos avisará cuando superemos una velocidad indicada. La velocidad máxima podemos ajustarla cambiando el valor de una variable en el programa. En caso de superar la velocidad indicada sonará un pitido producido por un zumbador. El valor de velocidad se obtendrá desde un módulo GPS.

Material necesario:

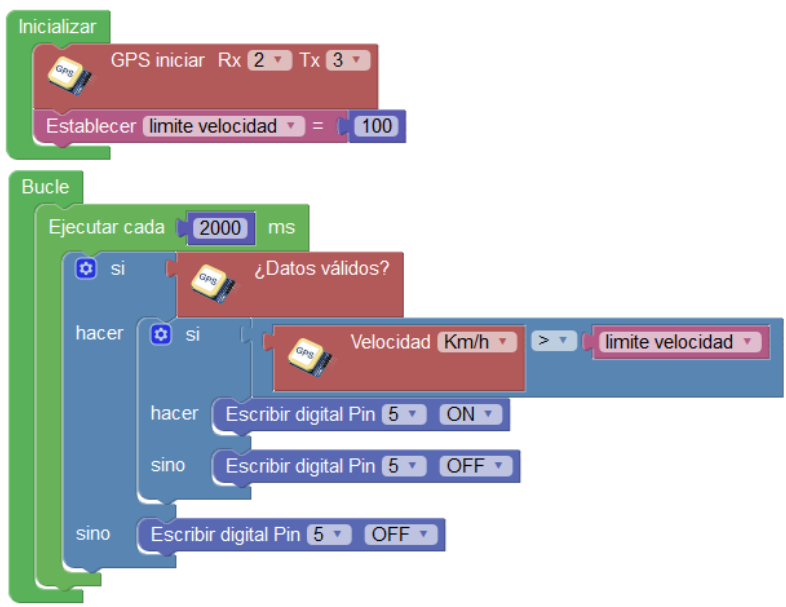
- 1 x módulo GPS
- 1 x Zumbador
- 1 x Arduino UNO
- Placa de prototipos y cables

Conexiones:
GPS RX = Pin 2
GPS TX = Pin 3
Zumbador = Pin ~5

Esquema de conexión:



Programa ArduinoBlocks:



P28 - ALARMA POR ALEJAMIENTO

Mediante el GPS podemos conocer en todo momento la posición exacta en la que nos encontramos, de igual forma podemos calcular la distancia (en línea recta) respecto a una posición preestablecida de forma que sabemos si estamos lejos o cerca de ese punto.

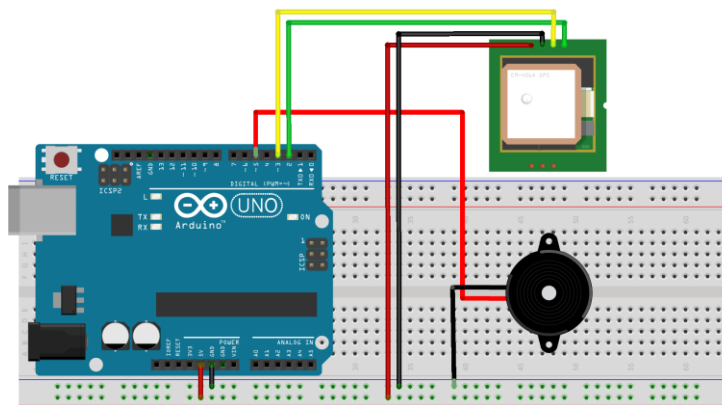
Este montaje detecta la distancia respecto a un punto prefijado y activará un aviso sonoro (zumbador) en caso de alejarnos más de 500m de ese lugar. Este proyecto puede ser útil por ejemplo para evitar que niños o personas mayores se desorienten y se pierdan alejándose de una zona establecida.

Material necesario:

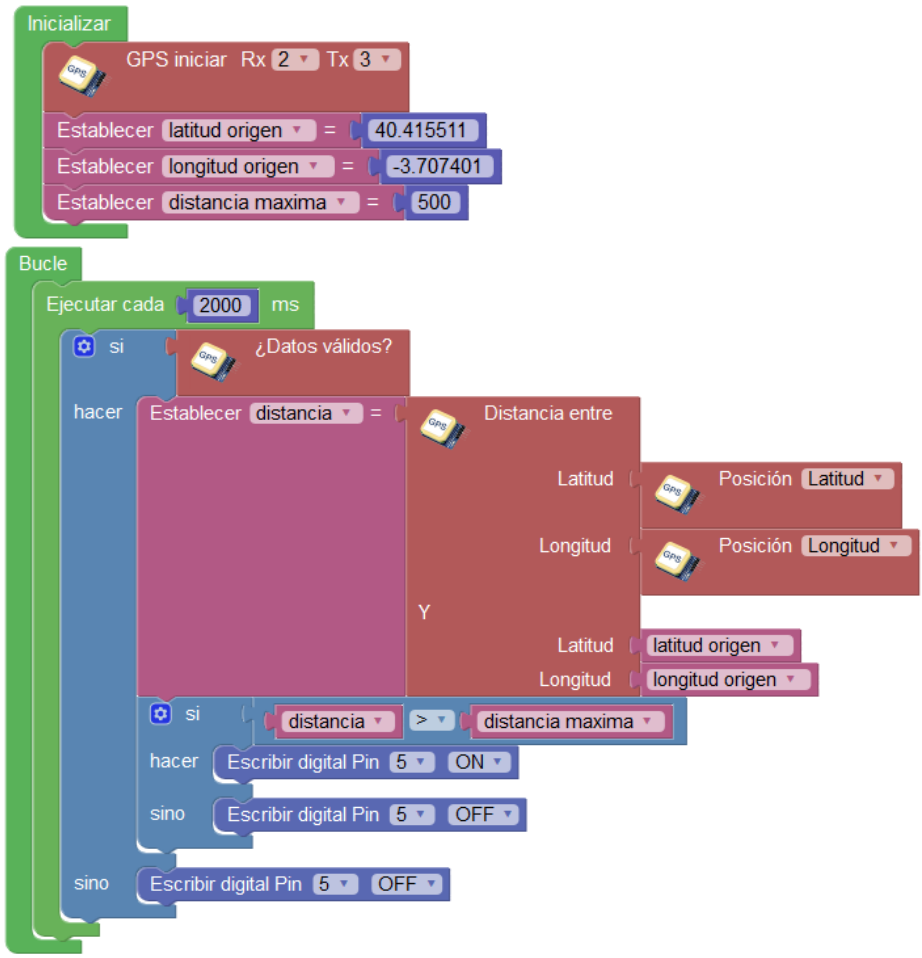
- 1 x módulo GPS
- 1 x Zumbador
- 1 x Arduino UNO
- Placa de prototipos
- Cables de interconexión

Conexiones:
GPS RX = Pin 2
GPS TX = Pin 3
Zumbador = Pin ~5

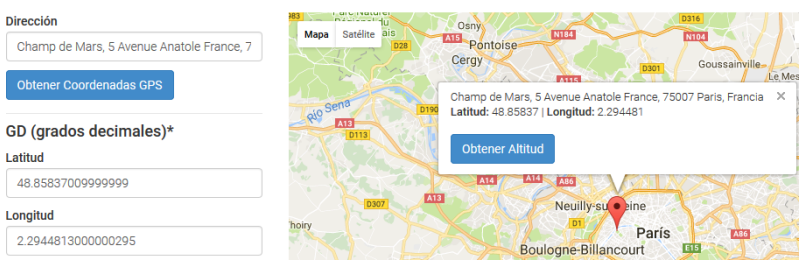
Esquema de conexión:



Programa ArduinoBlocks:



Para obtener las coordenadas GPS del punto origen podemos utilizar sitios webs como por ejemplo: http://www.coordenadas-gps.com/ donde indicando una dirección o marcando sobre el mapa podemos obtener fácilmente la latitud y longitud del punto:



P29 - REGISTRADOR GPS EN TARJETA SD

Un registrador GPS puede ser de gran utilidad para registrar nuestras rutas y su posterior procesamiento o visualización. Podemos utilizarlo por ejemplo para grabar nuestras rutas en bicicleta. El archivo generado en formato CSV se puede abrir fácilmente en aplicaciones de hoja de cálculo para su posterior procesamiento o con herramientas más potentes como por ejemplo la aplicación web GpsVisualizer o Google Maps.

Para poder visualizar el mapa con el recorrido grabado correctamente en la web GpsVisualizer (http://www.gpsvisualizer.com/) debemos generar el archivo CSV con un formato específico según nos indican en su sitio web:

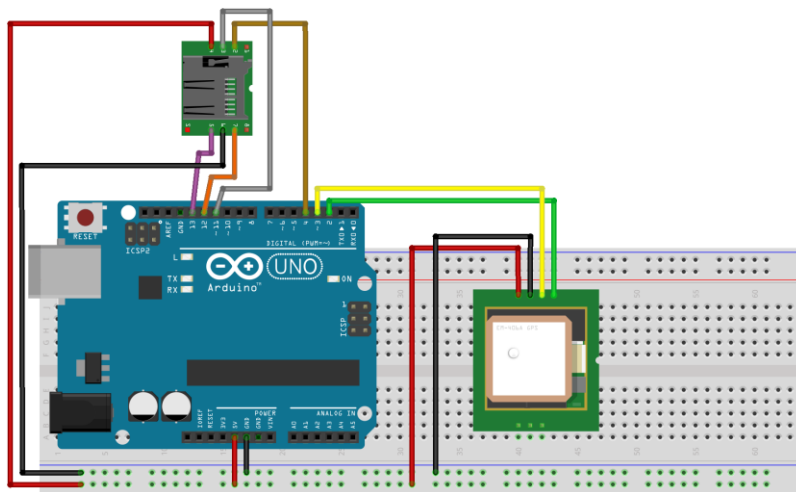
```
type,latitude,longitude,alt
T,45.9874167,-76.8752333,79.8
T,45.9860000,-76.8737833,111.4
T,45.9850500,-76.8724833,107.9
T,45.9844000,-76.8716333,120.0
T,45.9839500,-76.8710000,117.5
```

Material necesario:

- 1 x módulo GPS
- 1 x módulo tarjeta SD
- 1 x Arduino UNO
- Placa de prototipos
- Cables de interconexión

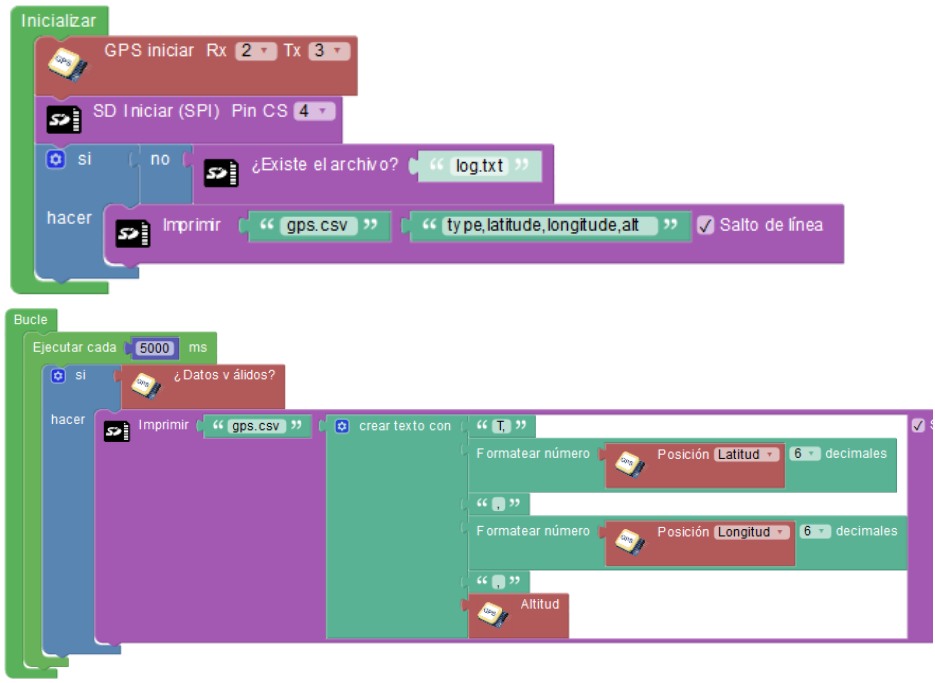
Conexiones:
GPS RX = Pin 2
GPS TX = Pin 3
SD = SPI

Esquema de conexión:



Programa ArduinoBlocks:

Si el archivo "gps.csv" no existe en la tarjeta inicializa la primera línea con el texto "type,latitude,longitude,alt". Cada 5s si tenemos datos válidos desde el GPS se registra una nueva línea con la información de posición y altitud.



Una vez finalizado el registro de datos podemos copiar el archivo gps.csv de la tarjeta de memoria a un PC y con la ayuda de la web GpsVisualizer obtendremos la ruta dibujada sobre el mapa:

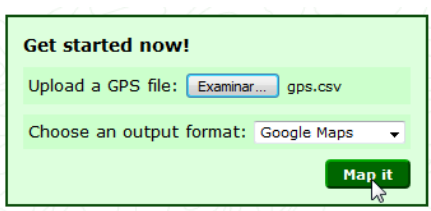
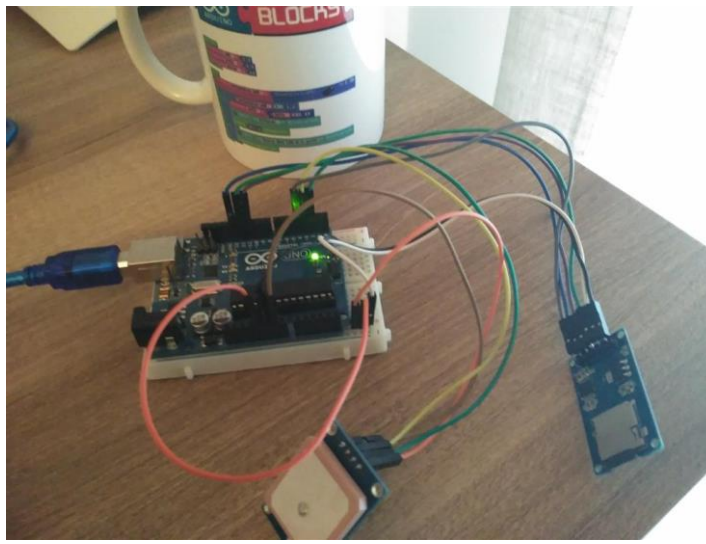
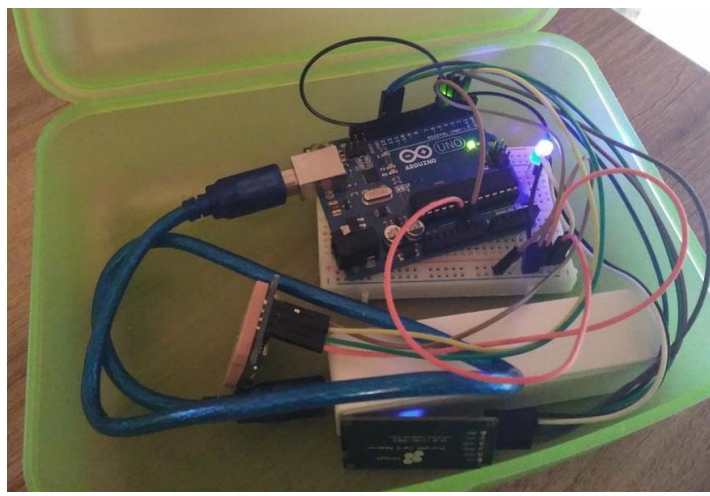


Imagen del montaje real con módulo GPS y tarjeta micro SD



Versión “empaquetada” y conectada a un power-bank USB como fuente de alimentación para poder transportarlo fácilmente.



P30 - REGISTRO DE TEMPERATURA/HUMEDAD EN SD

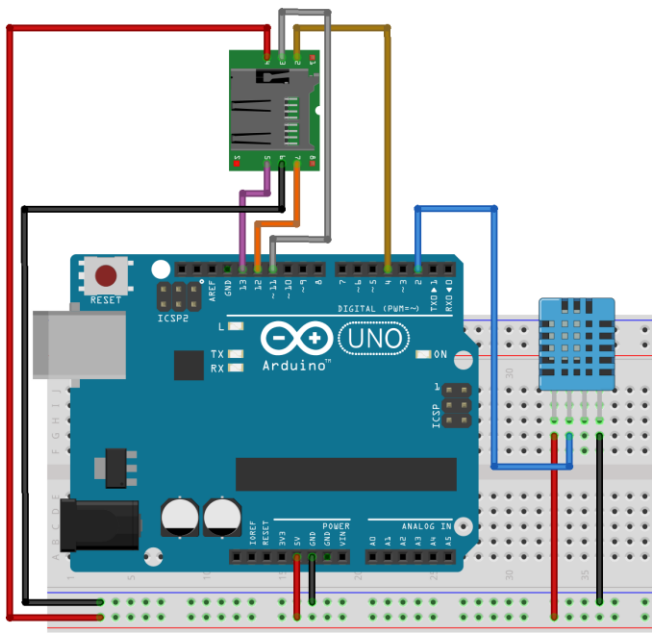
Controlar la temperatura y humedad de un lugar puede ser muy útil para comprobar si nuestro sistema de calefacción o refrigeración funciona bien. Normalmente no podemos estar visualizando los valores de temperatura y humedad en todo momento por lo que puede ser una gran idea registrar estos valores en un archivo para poder posteriormente visualizar los datos.

Material necesario:

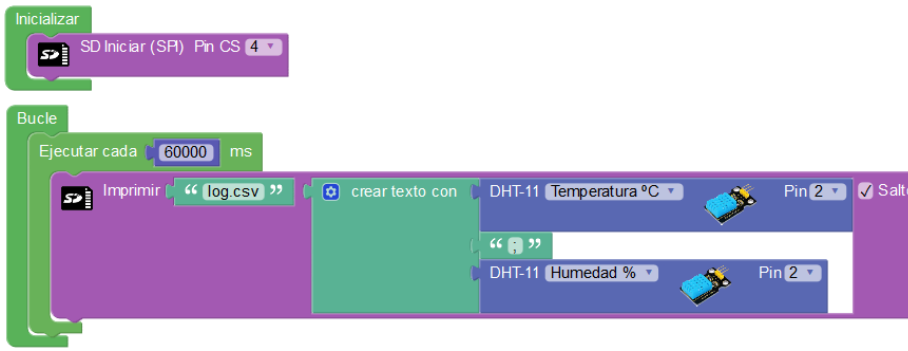
- 1 x sensor DHT11
- 1 x módulo tarjeta SD
- 1 x Arduino UNO
- Placa de prototipos
- Cables de interconexión

Conexiones:
SD = SPI
DHT11 = Pin 2

Esquema de conexión:



Programa ArduinoBlocks:



El archivo generado en la memora SD se llama “log.csv”.

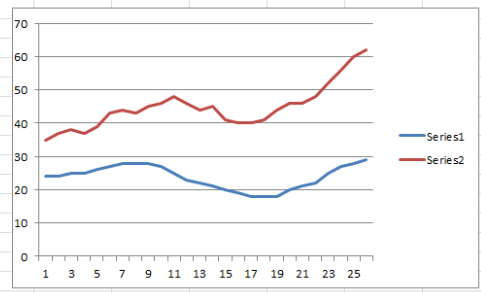
Un ejemplo de los datos almacenados visualizados en un editor de texto plano y procesado en una aplicación de hoja de cálculo:

Editor de texto:

```
24.00;35.00
24.00;37.00
25.00;38.00
25.00;37.00
26.00;39.00
27.00;43.00
28.00;44.00
28.00;43.00
28.00;45.00
27.00;46.00
25.00;48.00
23.00;46.00
22.00;44.00
21.00;45.00
20.00;41.00
19.00;40.00
18.00;40.00
18.00;41.00
18.00;44.00
20.00;46.00
21.00;46.00
```

Importación a LibreOffice Calc o Excel y creación de gráficas con los valores:

	A	B
1	24	35
2	24	37
3	25	38
4	25	37
5	26	39
6	27	43
7	28	44
8	28	43
9	28	45
10	27	46
11	25	48
12	23	46
13	22	44
14	21	45
15	20	41
16	19	40
17	18	40
18	18	41
19	18	44
20	20	46



Módulo de tarjeta micro SD (conexión SPI) utilizado en el proyecto:

P31 - CONTROL DE SERVO CON ACELERÓMETRO

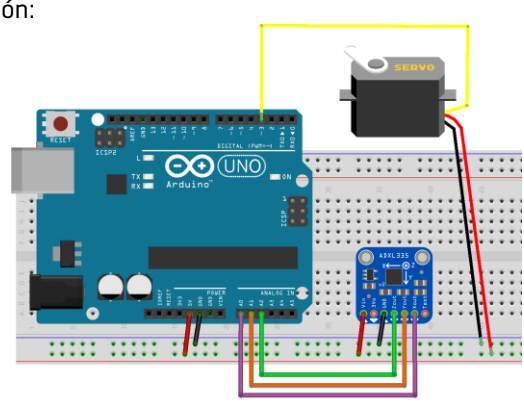
Este proyecto permite controlar un servo a partir de los movimientos detectados por un acelerómetro. Por ejemplo el acelerómetro podría estar fijado al dedo de una persona y con sus movimientos controlar el movimiento de un dedo robótico movido por un servo.

Material necesario:

- 1 x Módulo acelerómetro ADXL335
- 1 x Servomotor
- 1 x Arduino UNO
- Placa de prototipos
- Cables de interconexión

Conexiones:
Servo = Pin ~3
ADXL335 = A0,A1,A2

Esquema de conexión:



Programa ArduinoBlocks:

Vídeo del proyecto funcionando:
https://youtu.be/aeScceN1D7w

P32 - SENSOR DE CAÍDAS CON AVISO A EMERGENCIAS (VÍA BLUETOOTH + APP ANDROID)

Cuando se produce un impacto, se produce una desaceleración fuerte al para bruscamente un cuerpo. Esta desaceleración o "frenazo brusco" lo podemos detectar con un acelerómetro.

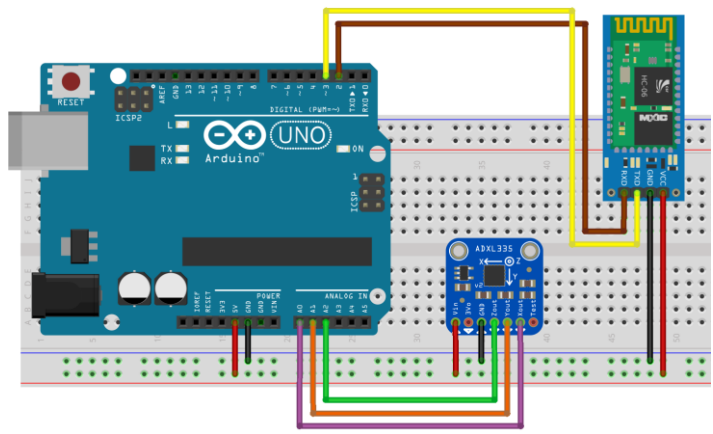
Siguiendo esta teoría vamos a realizar un proyecto en el que ante una caída se envíe una señal vía Bluetooth a una aplicación móvil Android que automáticamente llamará a un teléfono de emergencias.

Material necesario:

- 1 x Módulo acelerómetro ADXL335
- 1 x Módulo Bluetooth HC-06
- 1 x Arduino UNO
- Placa de prototipos
- Cables de interconexión

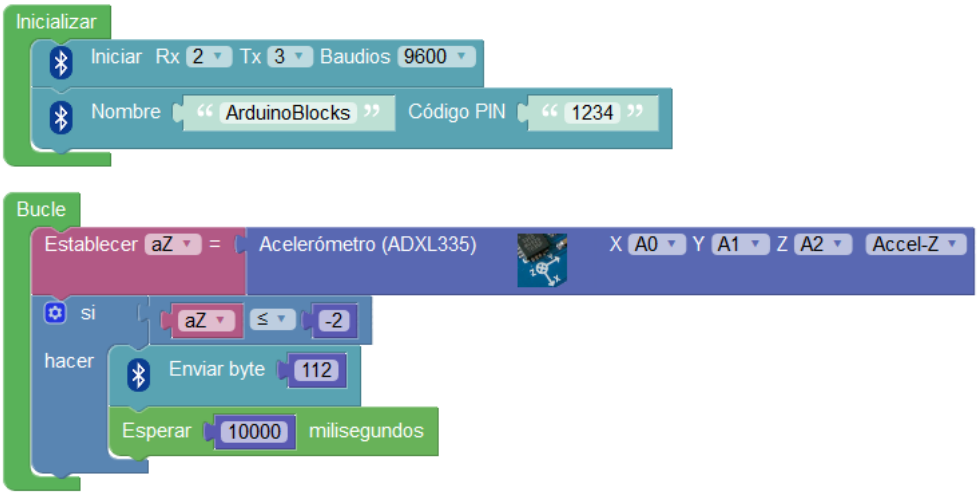
Conexiones:
Bluetooth RX = Pin 2
Bluetooth TX = Pin 3
ADXL335 = A0,A1,A2

Esquema de conexión:

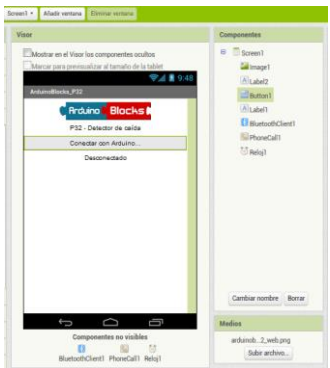


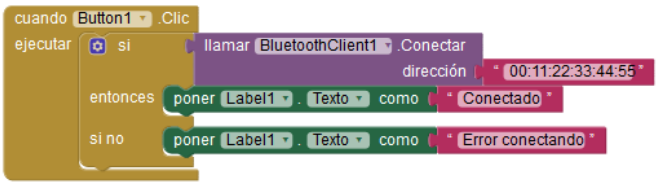
Programa ArduinoBlocks:

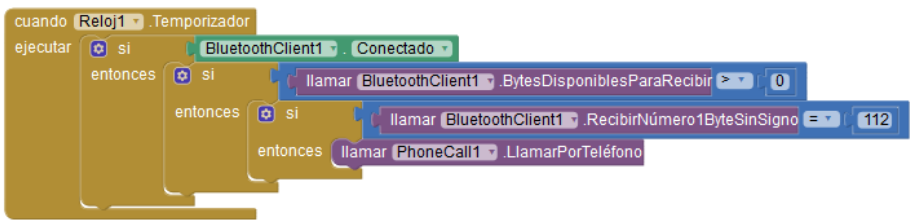
El programa comprueba continuamente la aceleración en el eje Z, en caso de detectar un valor menor que -2 (una desaceleración fuerte) envía el valor 112 a través de la conexión Bluetooth.



Aplicación Android con AppInventor:







P33 - MQTT (IOT): CONTROL DE LED RGB

Como ya hemos visto un led RGB nos permite obtener multitud de tonos de luz de diferentes colores simplemente combinando los valores de rojo, verde y azul de los que se compone. Este proyecto nos va a permitir controlar un led RGB desde el móvil y además vía internet, es decir, desde cualquier lugar del mundo con conexión a internet podremos ajustar nuestro led al color e intensidad deseado.

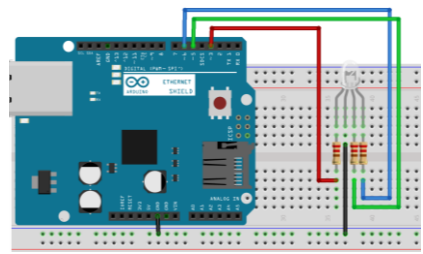
Para ello utilizaremos una Shield Ethernet que debemos conectar a internet mediante un router.

Material necesario:

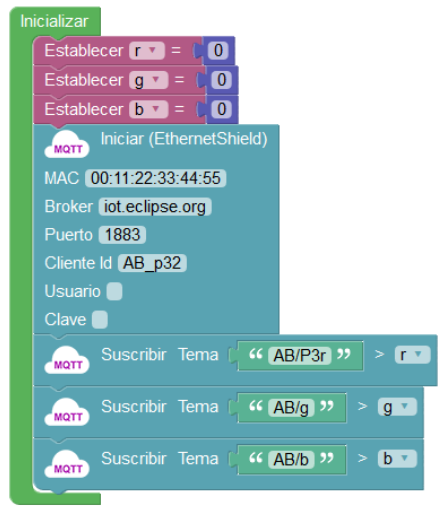
- 1 x Led RGB (cátodo común)
- 1 x Arduino Ethernet Shield
- 1 x Arduino UNO
- Placa de prototipos y cables

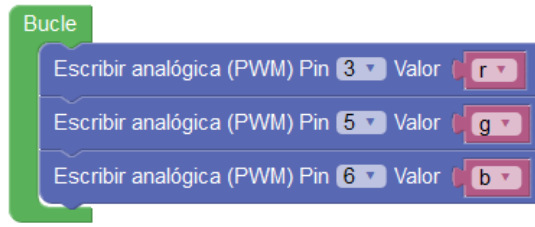
Conexiones:
RGB R = Pin ~3
RGB G = Pin ~5
RGB B = Pin ~6

Esquema de conexión:



Programa ArduinoBlocks:

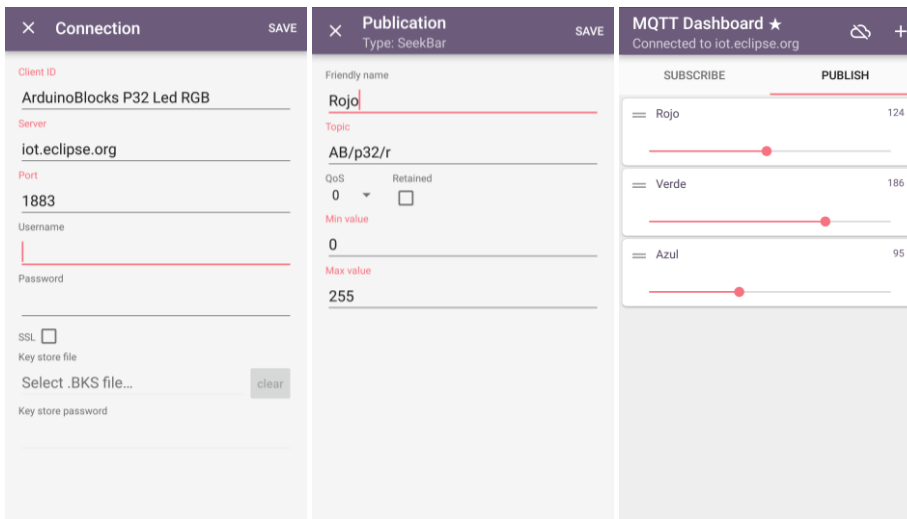




Para el control utilizaremos una aplicación como MQTT Dashboard (Android) o similar.

Configuración de la conexión con el broker MQTT

Creación de las barras de desplazamiento asociadas a cada tema "AB/p32/r", "AB/p32/g", "AB/p32/b"



<u>IMPORTANTE</u>: En este proyecto hemos utilizado un broker MQTT público y gratuito con fines experimentales, cualquier cliente que conecte a este broker y utilice los mismos temas (topics) podrá controlar o monitorizar nuestro proyecto. En un proyecto real debemos configurar nuestro propio broker MQTT seguro o utilizar uno público con seguridad (ver apdo. MQTT en el capítulo 3.3.13)

P34.-MQTT (IOT): ESTACIÓN METEOROLÓGICA

Gracias al protocolo MQTT vamos a implementar una sencilla estación meteorológica cuyos datos serán publicados por internet y visualizados con una aplicación cliente MQTT en un dispositivo móvil desde cualquier parte del mundo.

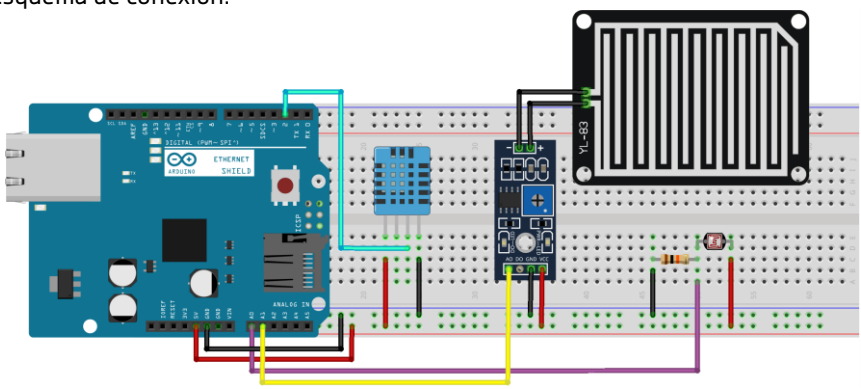
Los datos son actualizados cada 30s.

Material necesario:

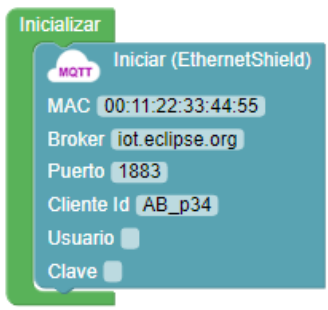
- 1 x Arduino Ethernet Shield
- 1 x Arduino UNO
- 1 x DHT-11
- 1 x LDR
- 1 x Sensor de lluvia
- Placa de prototipos y cables

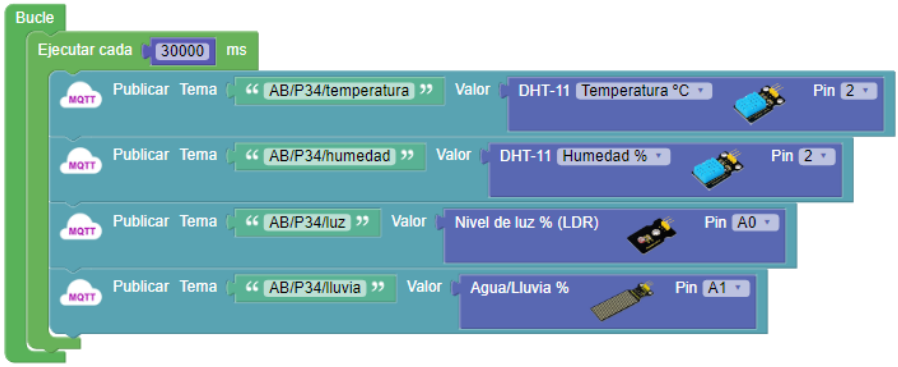
Conexiones:
DHT-11 = Pin 2
LDR = Pin A0
Sensor lluvia = Pin A1

Esquema de conexión:

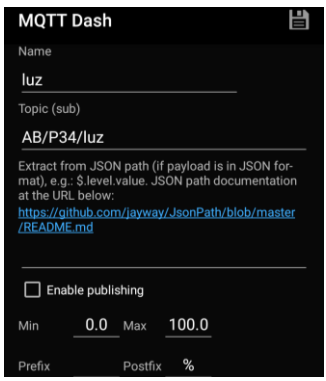


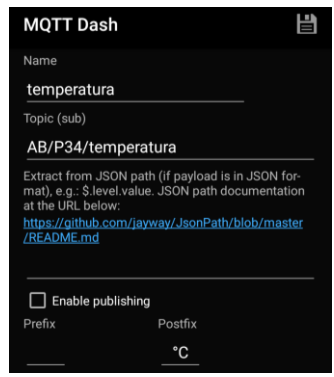
Programa ArduinoBlocks:





Ejemplo de monitorización desde aplicación MQTT Dash (Android):





https://play.google.com/store/apps/details?id=net.routix.mqttdash&hl=es

P35.-MQTT (IOT): CONTROL DOMÓTICO

En proyectos anteriores hemos realizado sencillas simulaciones de una instalación domótica controlada por Bluetooth. En este proyecto implementamos una funcionalidad similar con la ventaja del protocolo MQTT vía internet y del sencillo y cómodo control desde un terminal móvil.

Vamos a controlar dos puntos de luz accionados por relé y la posición de una persiana simulada con un servo.

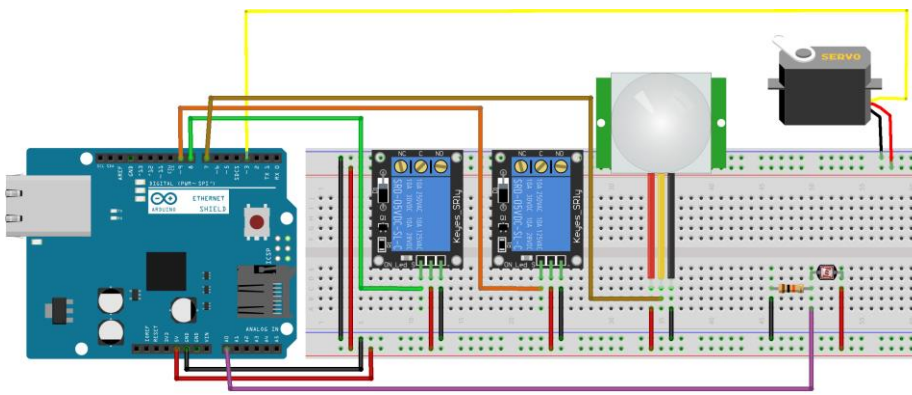
Por otro lado detectaremos la presencia con un sensor PIR y el nivel de luz ambiente con una LDR.

Material necesario:

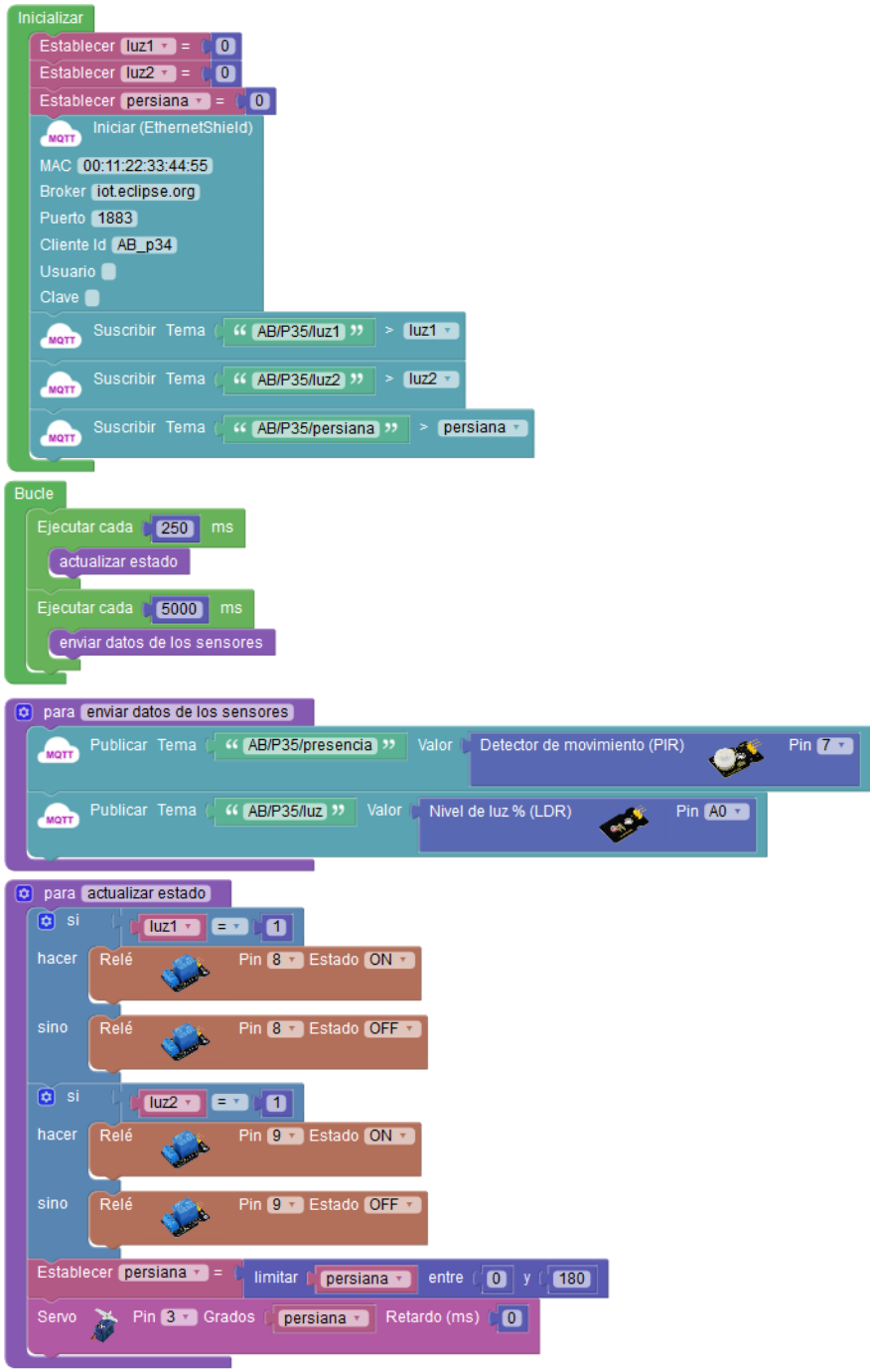
- 2 x Módulo relé
- 1 x Arduino Ethernet Shield
- 1 x Arduino UNO
- 1 x Módulo LDR
- 1 x Sensor PIR
- 1 x Servo
- Placa de prototipos y cables

Conexiones:
Servo = Pin 3
Relé 1 = Pin 8
Relé 2 = Pin 9
Sensor PIR = Pin 7
LDR = Pin A0

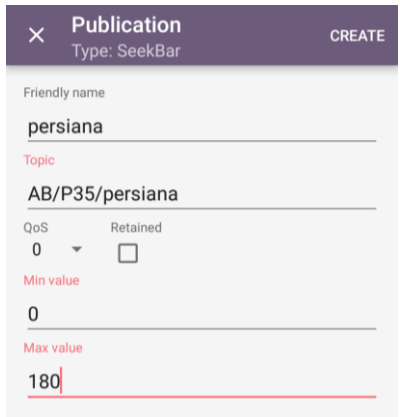
Esquema de conexión:

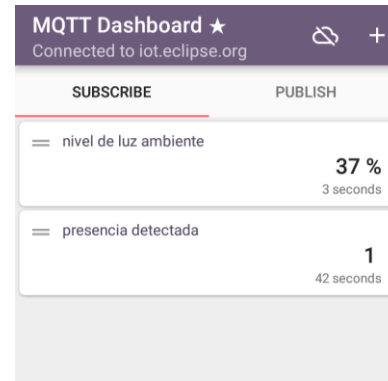


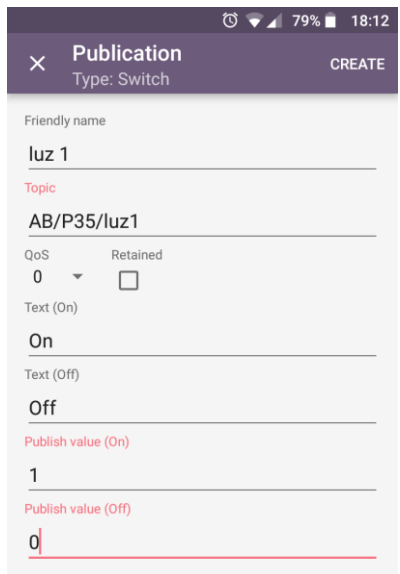
Programa ArduinoBlocks:

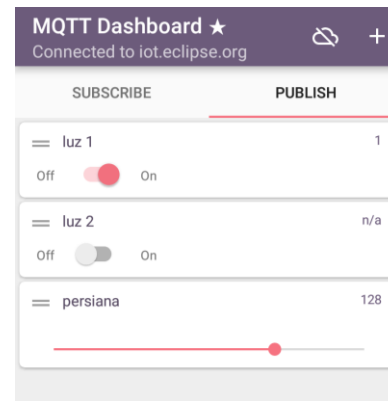


Ejemplo de control y monitorización desde aplicación MQTT Dashboard (Android):









https://play.google.com/store/apps/details?id=com.thn.iotmqttdashboard&hl=es

P36.-ROBOT CON SERVOS – CONTROL BLUETOOTH

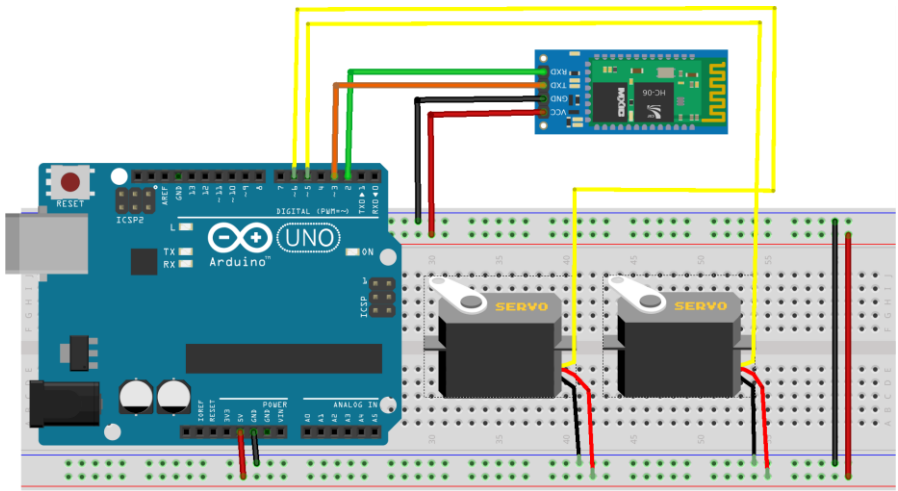
Vamos a realizar un pequeño vehículo que utiliza dos servos de rotación continua para el movimiento y un módulo Bluetooth HC-06 para comunicarse con una aplicación móvil y ser controlado de forma sencilla.

Material necesario:

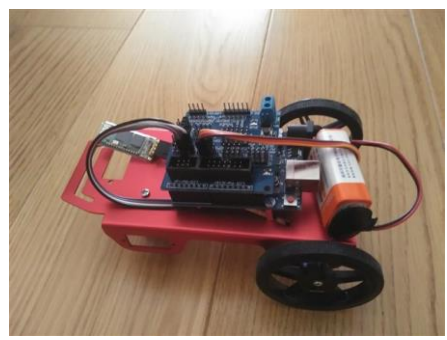
- 1 x Arduino UNO
- 1 x Sensor shield
- 1 x Módulo Bluetooth HC-06
- 2 x Servo rotación continua
- 1 x Batería 9v 1000mAh
- 1 x Cables

Conexiones:
Servo 1 = Pin 5
Servo 2 = Pin 6
HC-06 Rx=2
HC-06 Tx=3

Esquema de conexión:



Plataforma utilizada y montaje final:



Para el control se ha utilizado la aplicación gratuita “Bluetooth RC Controller” (Android):

https://play.google.com/store/apps/details?id=braulio.calle.bluetoothRCcontroller

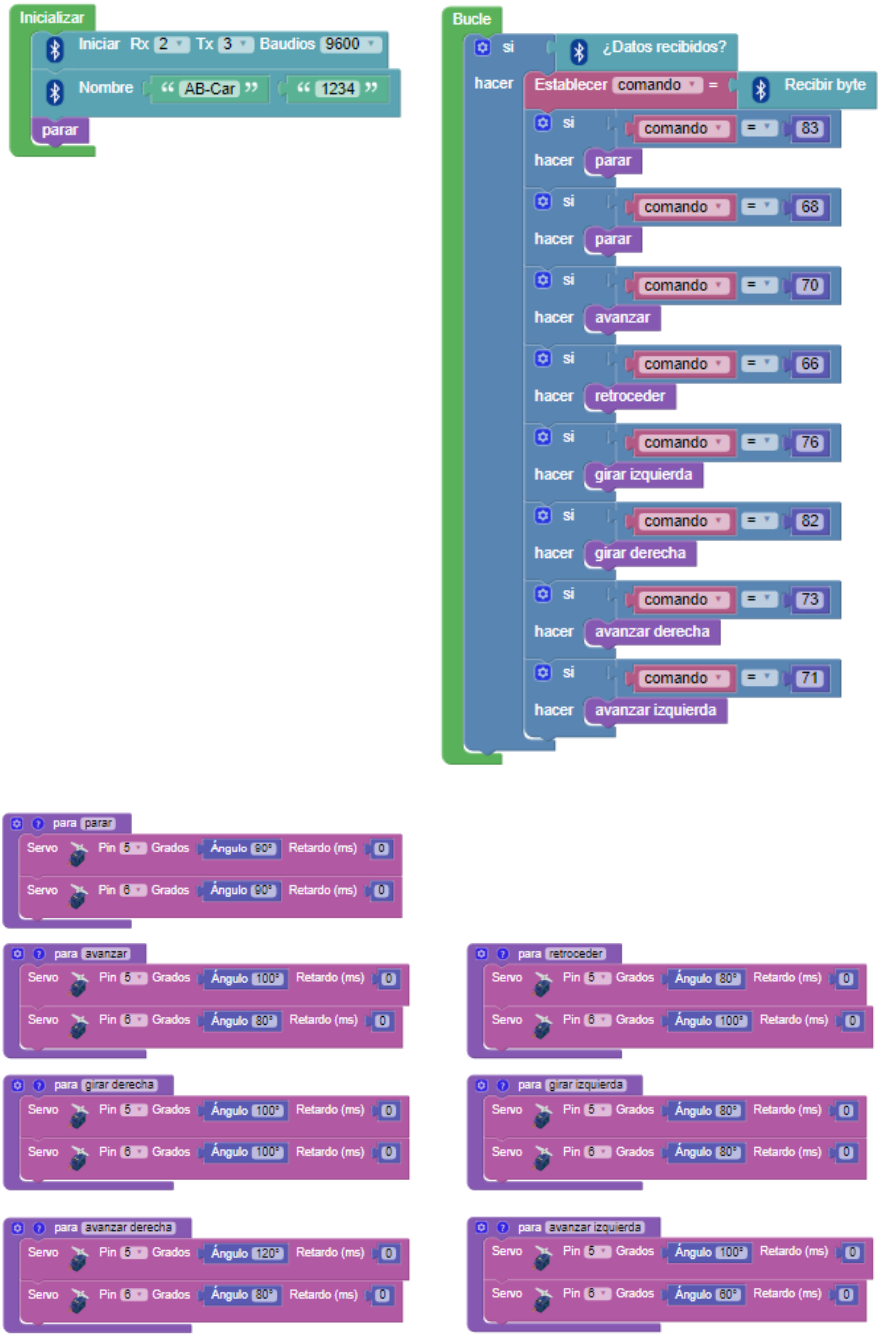
La aplicación enviará unos códigos a través de la conexión para cada acción:

Avanzar	F (ASCII: 70)
Retroceder	B (ASCII: 66)
Izquierda	L (ASCII: 76)
Derecha	R (ASCII: 82)
Avanzar derecha	G (ASCII: 71)
Avanzar izquierda	I (ASCII: 73)
Parar	S (ASCII: 83)

Control de los servos de rotación continua:

90º	parado
0º	gira en un sentido a máxima velocidad
180º	gira en sentido contrario a máxima velocidad

Programa ArduinoBlocks:



P37.-ROBOT CON MOTORES DC – CONTROL BLUETOOTH

Los robots móviles con motores DC son los más habituales, el proyecto anterior con servos de rotación continua es muy sencillo pero es más habitual y fácil de conseguir robots con motores de corriente continua.

Para controlar este tipo de motores como se ha descrito anteriormente necesitamos un driver o controlador de puente en H que nos permite controlar la dirección de giro y la velocidad.

De igual forma que en el proyecto anterior se utiliza un modulo HC-06 para control vía Bluetooth a través de la aplicación "Bluetooth RC Controller" (Android).

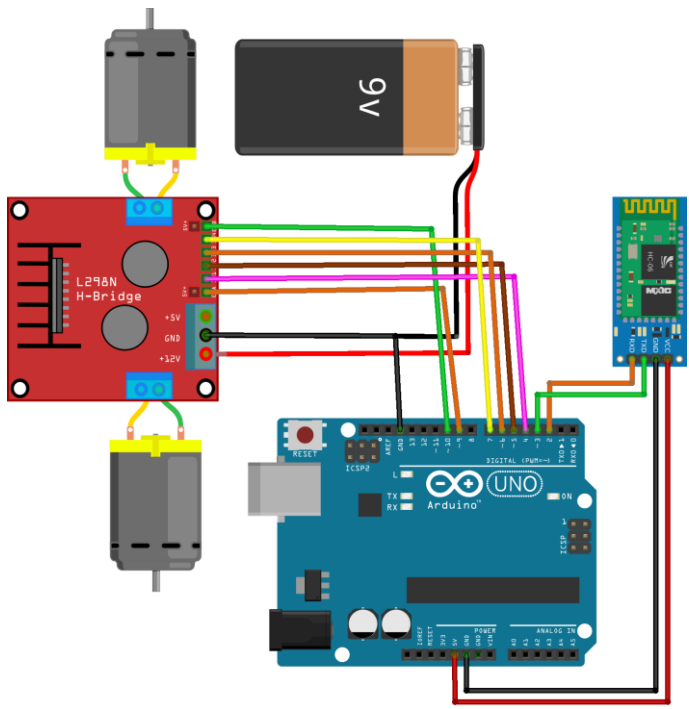
Material necesario:

- 1 x Arduino UNO
- 1 x Módulo puente en H
- 1 x Módulo Bluetooth HC-06
- 1 x Cables
- 2 x Motores DC + ruedas
- 1 x Estructura vehículo

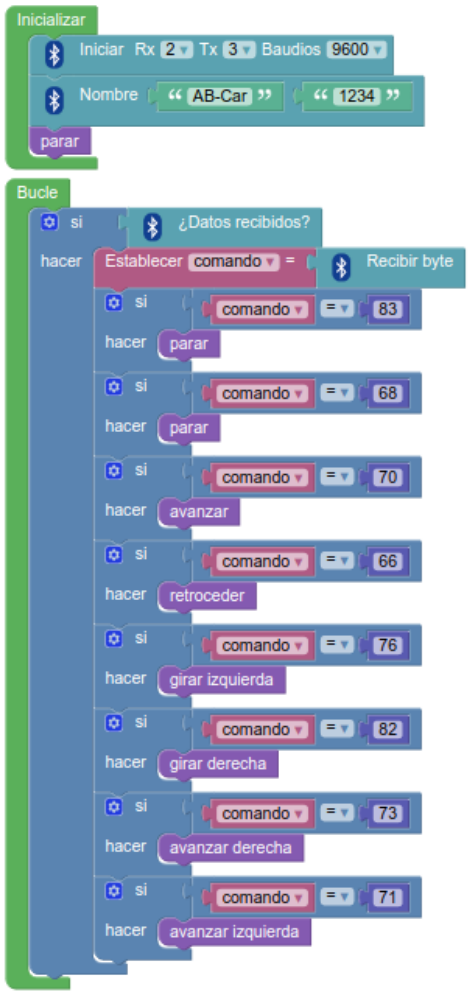
Conexiones:

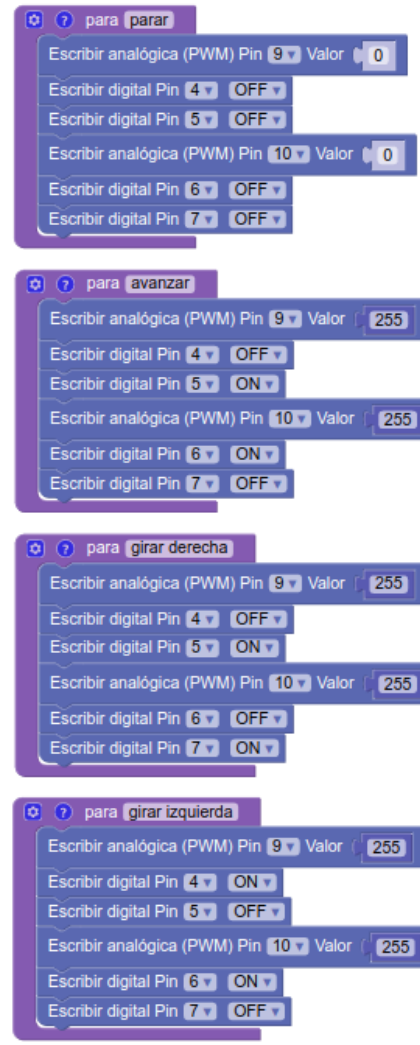
Puente-H EN1 = Pin 9
Puente-H M1 = Pines 4,5
Puente-H EN2 = Pin 10
Puente-H M2 = Pines 6,7
Bluetooth RX = Pin 2
Bluetooth TX = Pin 3

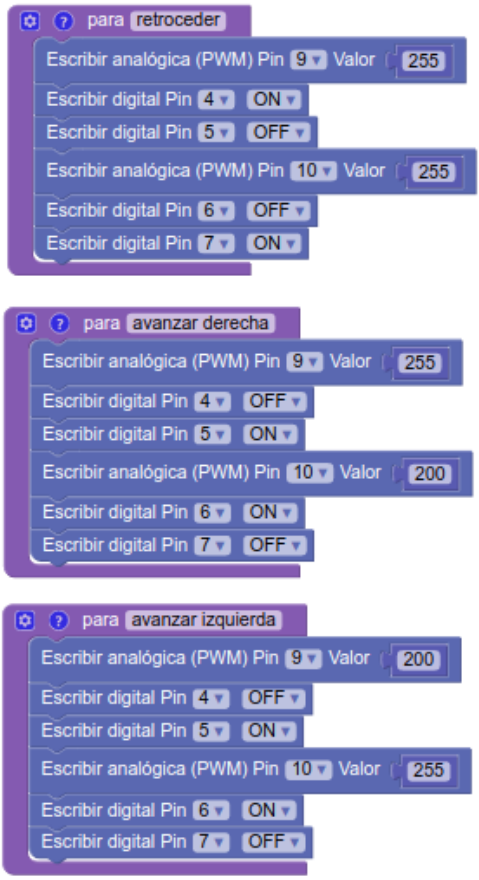
Esquema de conexión:



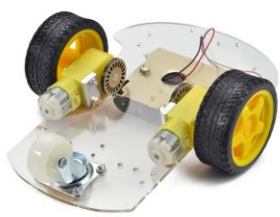
Programa ArduinoBlocks:

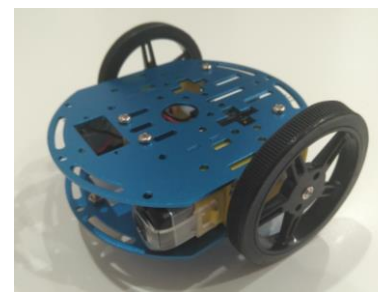






Ejemplo de estructuras para robots de 2 ruedas:





P38.-ROBOT CON MOTORES DC – EVITA OBSTÁCULOS

Un robot evita obstáculos es un tipo de robot autónomo que automáticamente detecta obstáculos delante de él y los intenta esquivar. El robot se mueve continuamente girando al detectar un obstáculo.

Para la detección de obstáculos se utiliza un sensor HC-SR04. Al detector un obstáculo el robot gira de forma aleatoria a la izquierda o a la derecha para cambiar de dirección.

El control de los motores DC se realizará con un modulo de puente en H como en los proyectos anteriores.

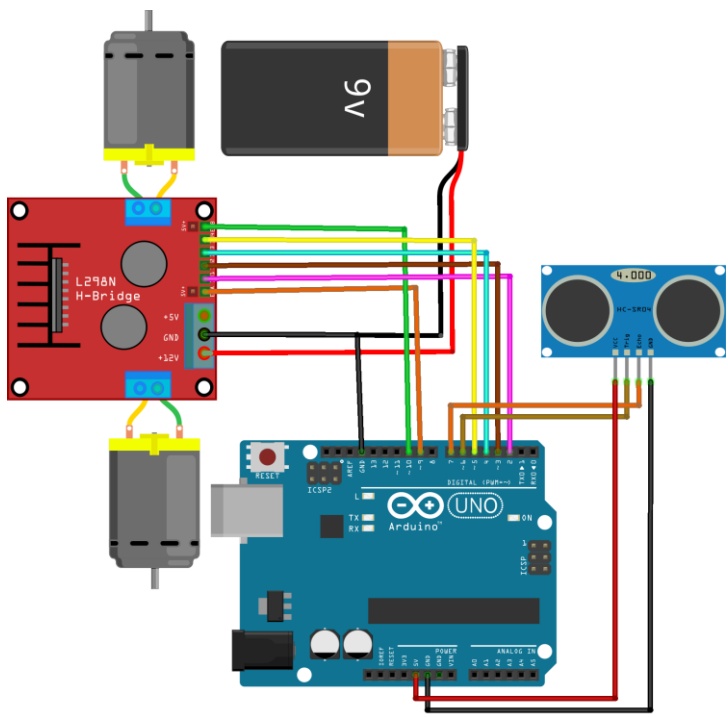
Material necesario:

- 1 x Arduino UNO
- 1 x Módulo puente en H
- 1 x Módulo HC-SR04
- 1 x Cables
- 2 x Motores DC + ruedas
- 1 x Estructura vehículo

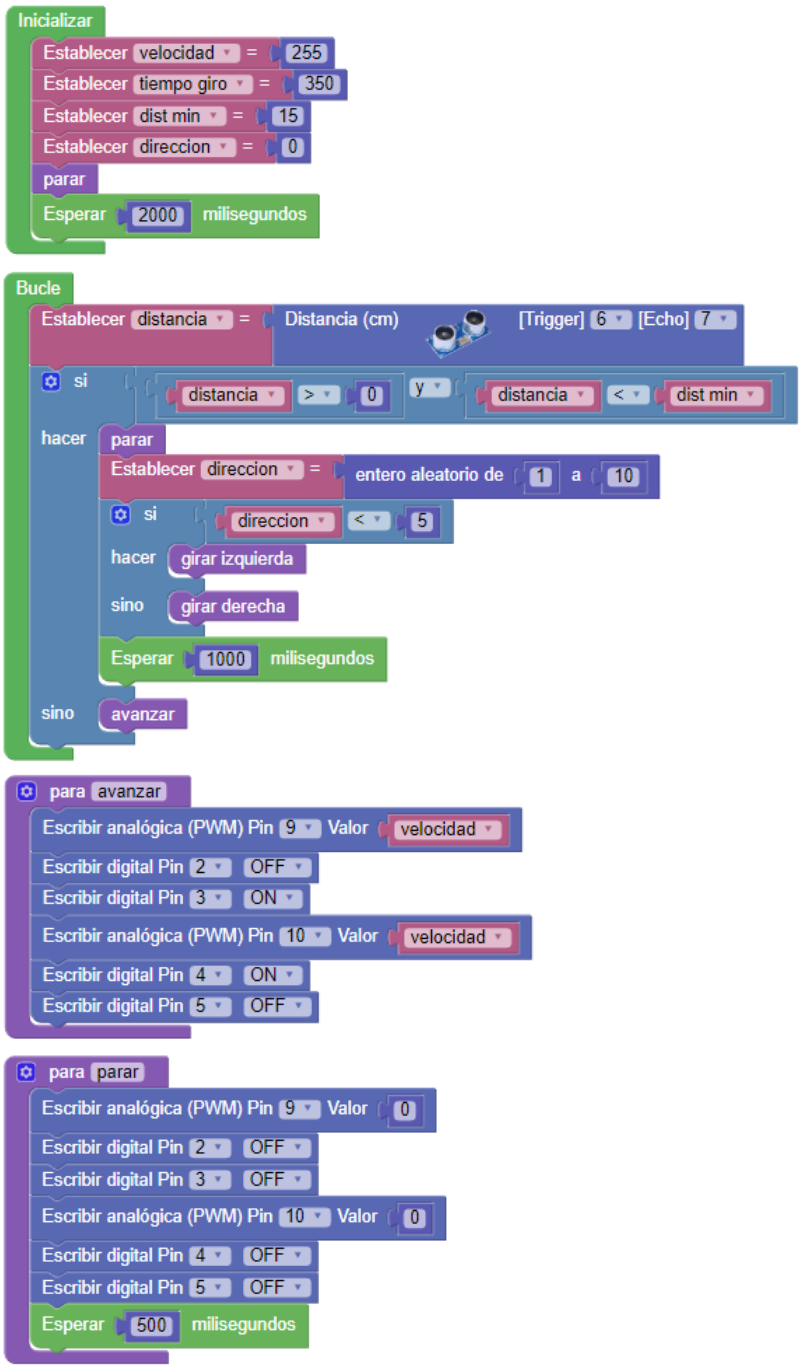
Conexiones:

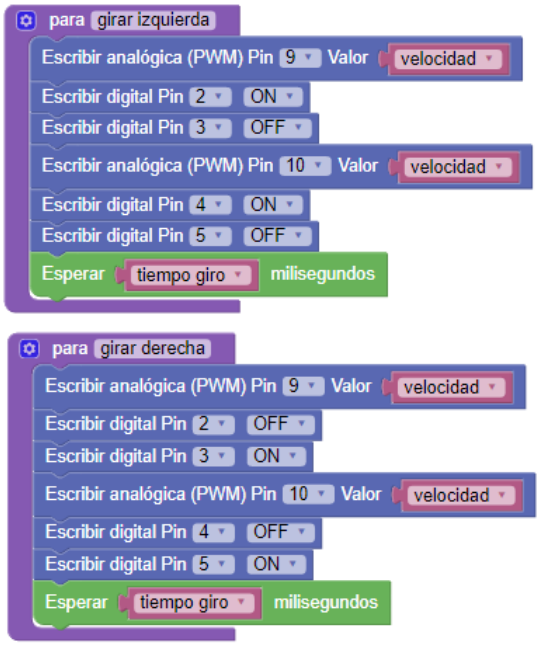
Puente-H EN1 = Pin 9
Puente-H M1 = Pines 2,3
Puente-H EN2 = Pin 10
Puente-H M2 = Pines 4,5
HC-SR04 Trigger = Pin 6
HC-SR04 Echo = Pin 7

Esquema de conexión:

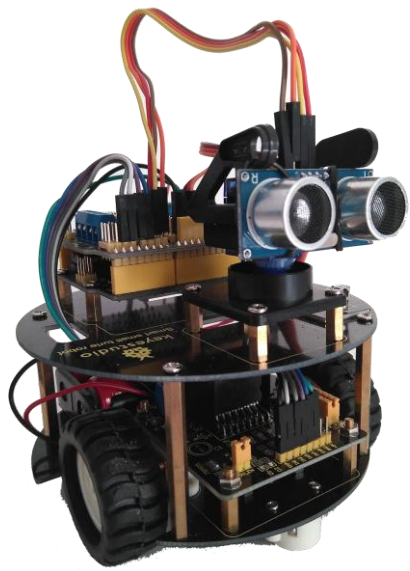


Programa ArduinoBlocks:





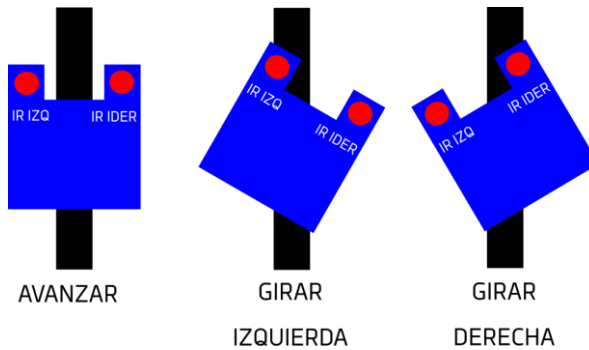
Ejemplo de robot evita obstáculos:



P39.-ROBOT CON MOTORES DC - SIGUE LÍNEAS

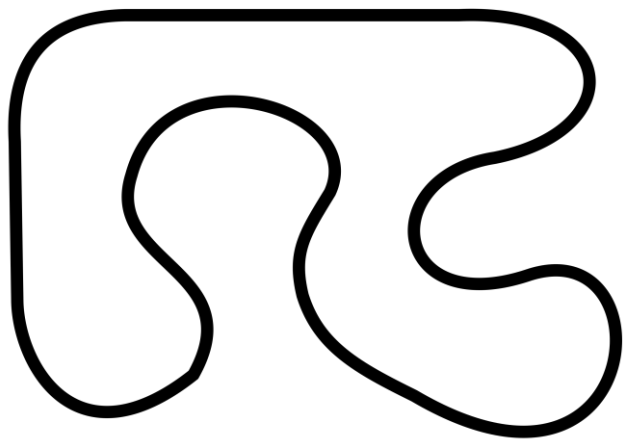
Un robot sigue líneas consiste en un robot motorizado que seguirá el recorrido marcado por una línea negra sobre una superficie blanca.

El seguimiento de la línea se realiza gracias a unos sensor IR que detectan si la superficie sobre la que están es blanca o negra (por la reflexión de la luz IR).



El tiempo de giro y la velocidad se debe ajustar en función de los motores utilizados, la distancia entre los sensores, el ancho de la líneas, etc.

Ejemplo de circuito para robot sigue líneas:



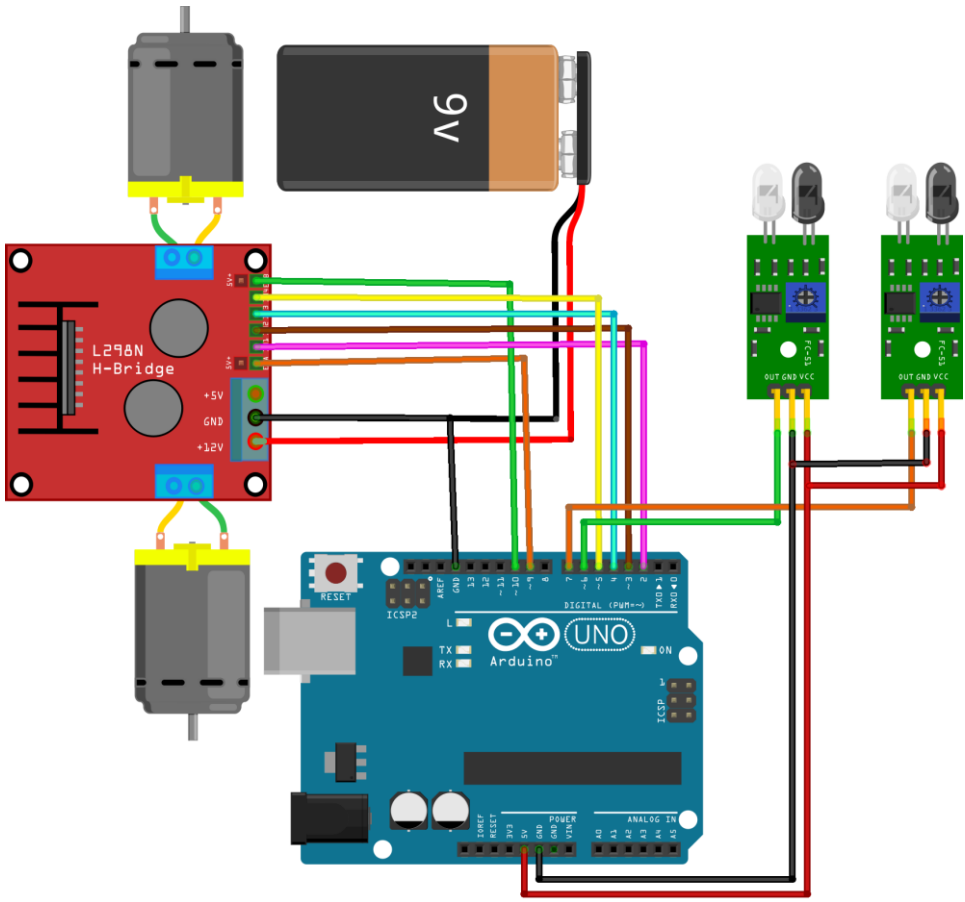
Material necesario:

- 1 x Arduino UNO
- 1 x Módulo puente en H
- 2 x Sensor IR
- 1 x Cables
- 2 x Motores DC + ruedas
- 1 x Estructura vehículo

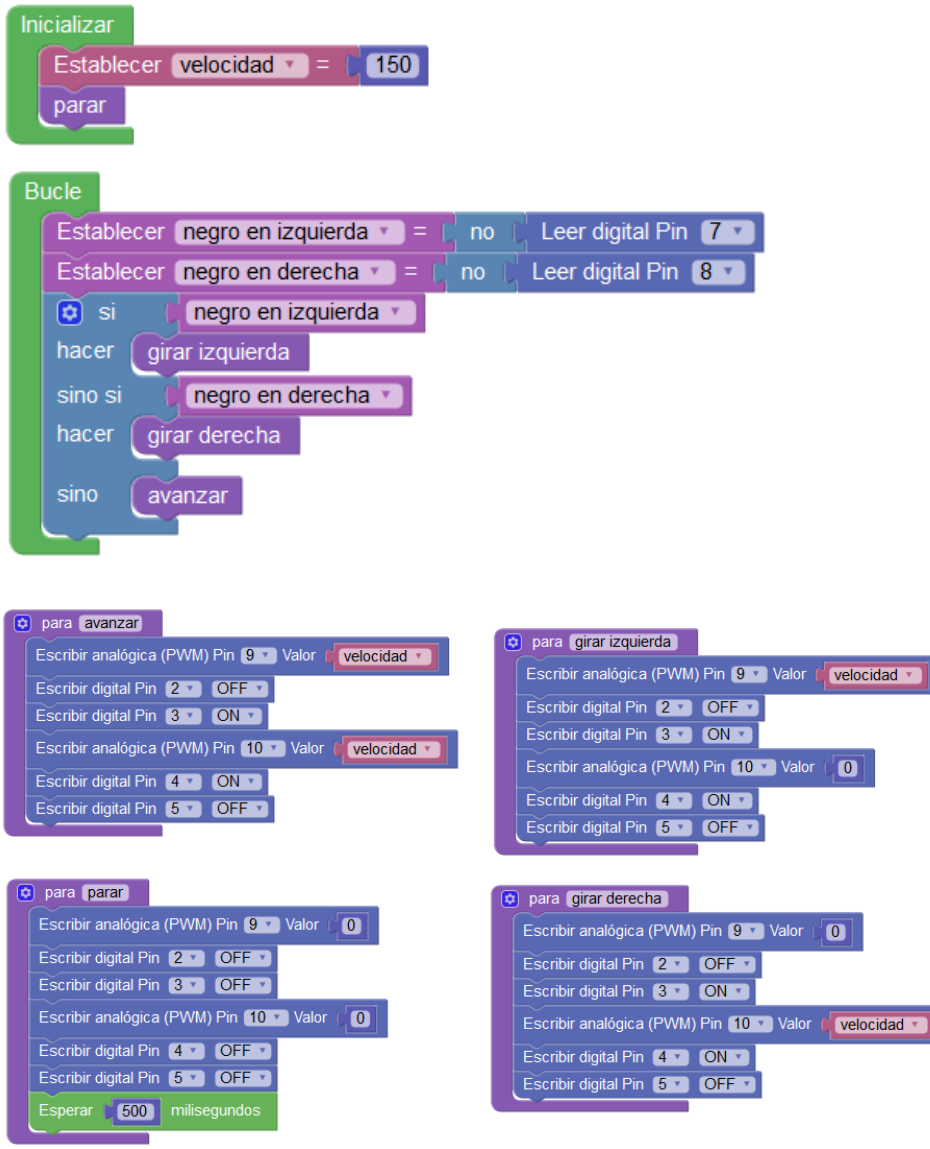
Conexiones:

Puente-H EN1 = Pin 9
Puente-H M1 = Pines 2,3
Puente-H EN2 = Pin 10
Puente-H M2 = Pines 4,5
Sensor IR Izq. = Pin 6
Sensor IR der. = Pin 7

Esquema de conexión:



Programa ArduinoBlocks:



P40.-BRAZO ROBÓTICO CONTROLADO DESDE PC (CONSOLA)

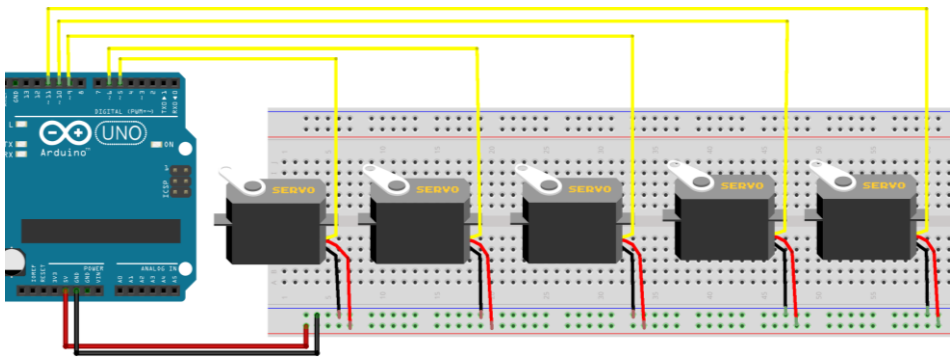
Un brazo robótico consiste en una serie de articulaciones mecánicas accionadas por motores que controlan el movimiento. Para la realización del proyecto se han utilizado 4 servos para el control de los movimientos de las articulaciones y 1 servo en el extremo del brazo para mover una pinza que abre y cierra cone el objetivo de poder coger y soltar objetos.

Material necesario:

- 1 x Arduino UNO
- 1 x Sensor shield
- 5 x Servos
- 1 x Cables

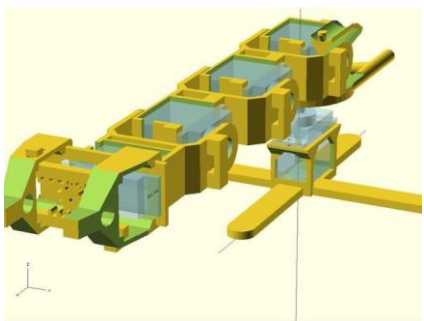
Conexiones:
Servo-1 = Pin 5
Servo-2 = Pin 6
Servo-3 = Pin 9
Servo-4 = Pin 10
Servo-Pinza= Pin 11

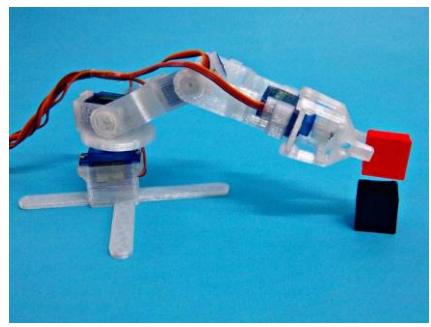
Esquema de conexiones:



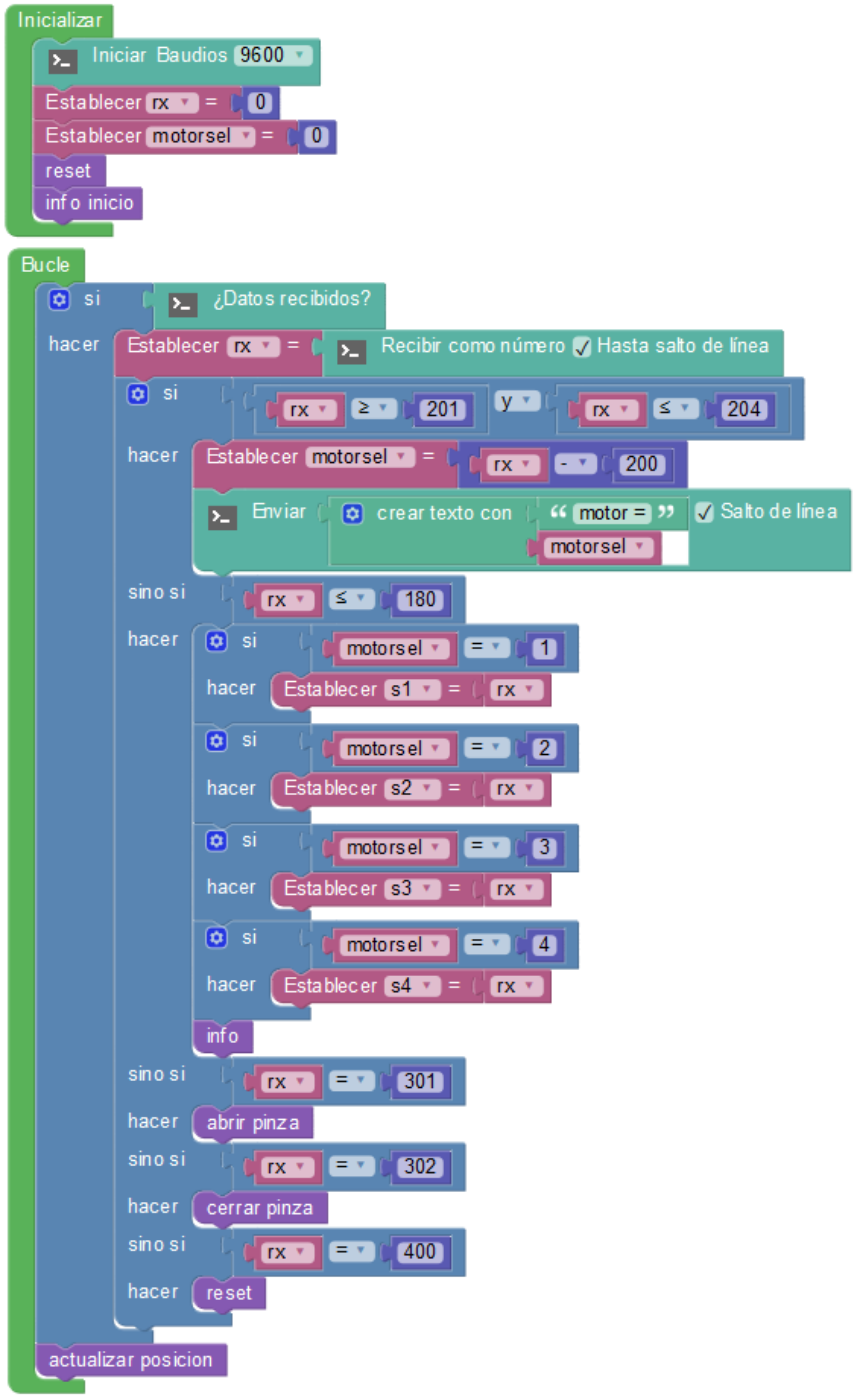
El brazo robótico para probar este proyecto ha sido impreso en 3D. El modelo puedes encontrarlo compartido en el enlace:

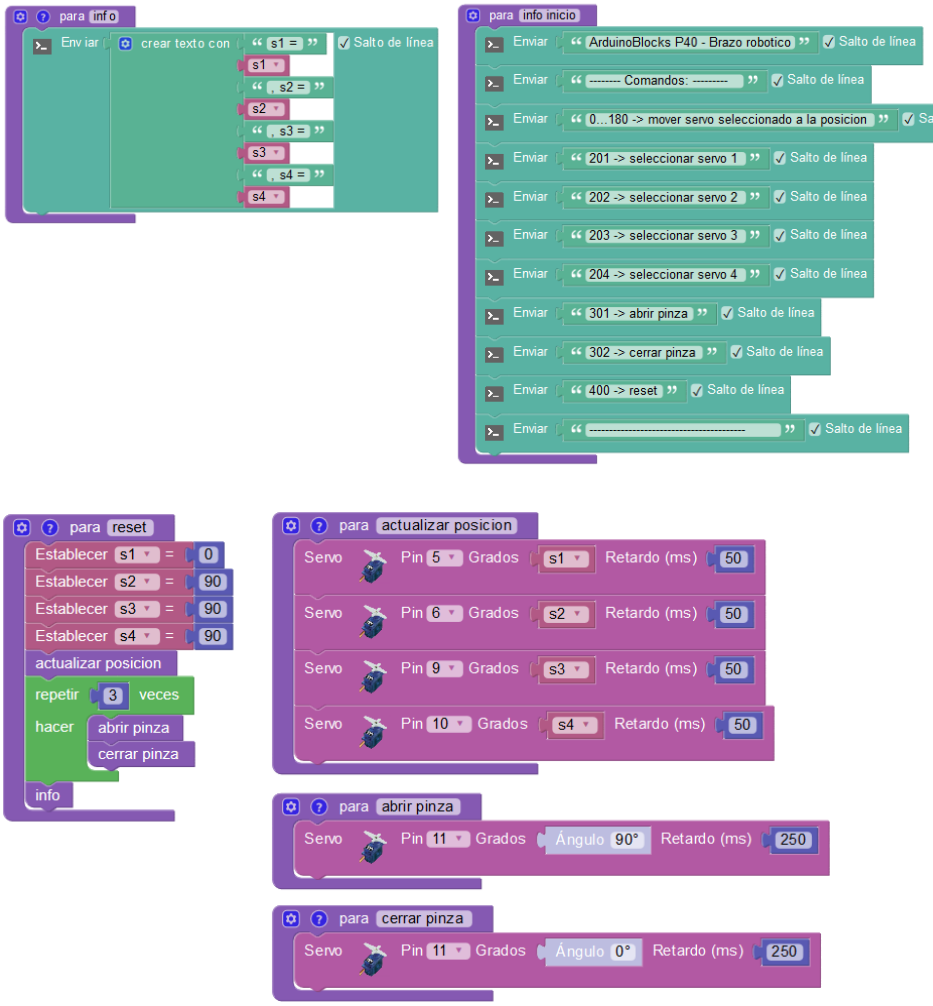
https://www.thingiverse.com/thing:65081



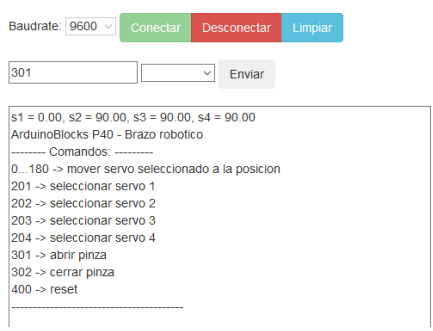


Programa ArduinoBlocks:





El control se realizará desde la consola serie desde un PC. Se puede utilizar directamente la consola de ArduinoBlocks.



ANEXO I: Bloques incompatibles con bloqueos de tiempo.

Los siguientes bloques deben realizar tareas periódicas en segundo plano y por lo tanto debemos evitar situaciones en nuestro programa donde se bloquee la ejecución. Si un bloque o bloques consumen mucho tiempo de ejecución del procesador no dejarán realizar estas tareas en segundo plano y por lo tanto el funcionamiento no será correcto.

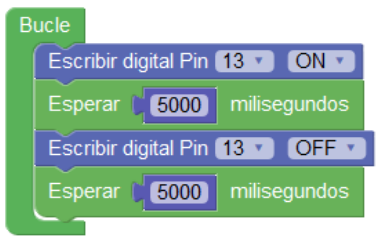
Los bloques GPS necesitan leer periódicamente los datos desde el módulo para obtener la información actualizada. Si utilizamos bloqueos en nuestro programa los datos GPS no serán válidos.

Los bloques MQTT gestionan la comunicación a través de la red Ethernet (TCP/IP) de forma continua en segundo plano, si bloqueamos la ejecución del programa no se realizará correctamente la comunicación.

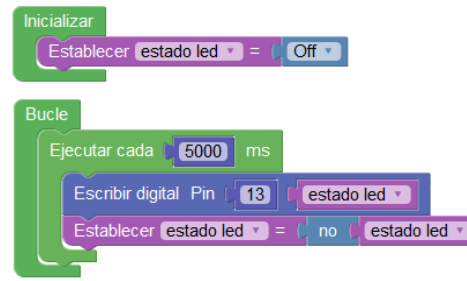
Por lo tanto en estos casos es recomendable seguir siempre un método de programación por tareas utilizando bloques del tipo "*ejecutar cada*" (ver apdo. 3.3.2)

Ejemplo: Parpadeo de led cada 5s con bloqueo y sin bloqueo

Utilizando bloqueo de tiempo:

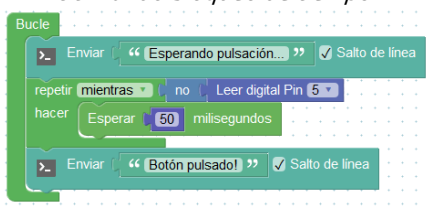


Solución sin bloqueos:



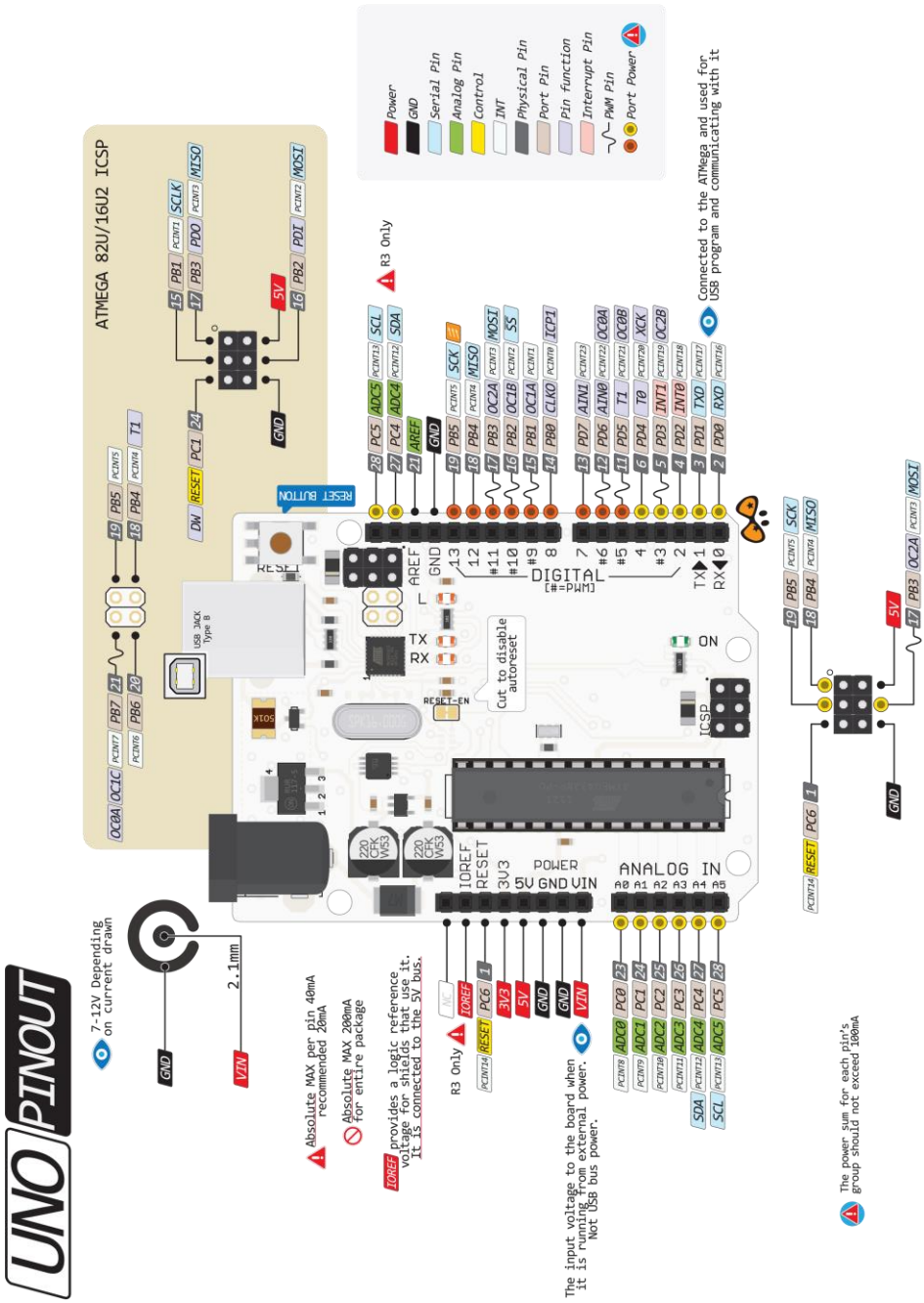
Ejemplo: Comprobar pulsación de un botón en el pin 5

Utilizando bloqueo de tiempo:



Solución sin bloqueos:

ANEXO II: Arduino UNO R3 pinout



BIBLIOGRAFÍA Y ENLACES DE INTERÉS:

http://www.arduinoblocks.com
https://www.arduino.cc/
http://www.keyestudio.com/
http://aulatecnologia.com/ESO/TERCERO/teoria/electricidad/electricidad.htm
https://es.wikipedia.org/wiki/Corriente_el%C3%A9ctrica
https://es.wikipedia.org/wiki/Potencia_el%C3%A9ctrica
http://embedded-lab.com/blog/humidity-and-temperature-measurements-with-sensirions-sht1xsht7x-sensors-part-1/
http://realterm.sourceforge.net/
https://es.wikipedia.org/wiki/I%C2%B2C
https://es.wikipedia.org/wiki/Serial_Peripheral_Interface
https://es.wikipedia.org/wiki/Algoritmo
http://ai2.appinventor.mit.edu
https://en.wikipedia.org/wiki/Bluetooth
http://www.c-sharpcorner.com/uploadfile/167ad2/how-to-use-ultrasonic-sensor-hc-sr04-in-arduino/
https://es.wikipedia.org/wiki/Codificador_rotatorio
http://www.pighixxx.com
https://learn.sparkfun.com/tutorials/what-is-an-arduino
https://www.arduino.cc/en/Tutorial/HomePage
https://scratch.mit.edu/
http://fritzing.org/home/
https://learn.adafruit.com
http://www.gpsvisualizer.com
http://www.coordenadas-gps.com/
https://www.thingiverse.com/
https://www.tinkercad.com/
https://circuits.io/

ARDUINOBLOCKS EN LAS REDES SOCIALES:

https://www.facebook.com/ArduinoBlocks
https://www.youtube.com/channel/UCoJwWGyd8a2pxzJHFdftXYw
https://twitter.com/arduinoblocks
https://es.linkedin.com/in/arduinoblocks-programaci%C3%B3n-visual-5169a9133

NOVEDADES, PROYECTOS Y NUEVOS BLOQUES:

http://www.arduinoblocks.com/blog/

COLABORADOR Y DISTRIBUIDOR OFICIAL KEYESTUDIO:

http://www.innovadidactic.com

CONTACTO:

Juanjo López
info@arduinoblocks.com

ANOTACIONES

La plataforma ArduinoBlocks nos permite iniciarnos en el mundo de la electrónica y robótica de una forma sencilla, rápida e intuitiva. Es la herramienta perfecta para que niños, jóvenes y no tan jóvenes se inicien en el fascinante mundo Arduino.

Empieza a crear proyectos reales desde el primer momento sin escribir ni una línea de código, comparte tus proyectos con la comunidad y ten todo tu trabajo siempre a salvo y accesible en la nube.

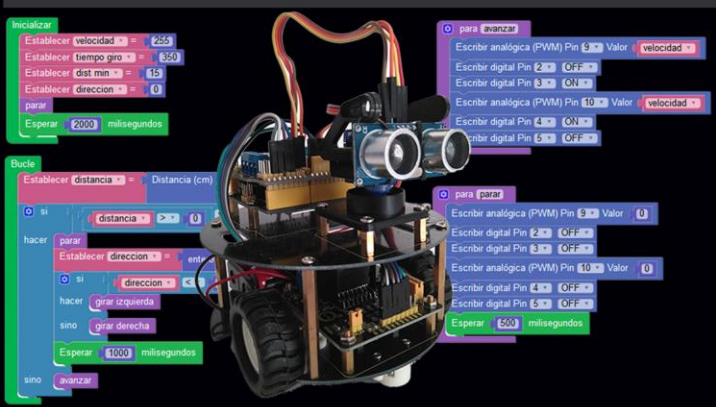
¡Deja volar tu imaginación y únete al movimiento maker de la mano de ArduinoBlocks!

La 2ª Edición incluye:

· Introducción a la plataforma ArduinoBlocks
· Referencia a los bloques de uso general
· Referencia a los bloques de Arduino: entrada/salida, tiempo, serie, bluetooth, sensores, actuadores, lcd, eeprom, servos, motores, keypad, reloj rtc, gps, tarjeta sd, IoT mqtt

· 40 Proyectos resueltos:

- · Secuencias de luces
- · Simulación amanecer/anochecer
- · Lámpara con regulación manual
- · Semáforo
- · Timbre
- · Control inteligente de iluminación
- · Encendido automático por movimiento
- · Contador manual
- · Cronómetro
- · Fotómetro
- · Iluminación crepuscular
- · Encendido y apagado con palmada
- · Termómetro
- · Termostato
- · Medidor de distancia
- · Riego automático
- · Lámpara multicolor con control IR
- · Piano con teclado
- · Sensor de aparcamiento
- · Control pan/tilt con joystick
- · Control de un led desde PC
- · Control de relés por Bluetooth
- · Estación meteorológica
- · Control de led por voz
- · Control domótico
- · Visualización GPS en LCD
- · Aviso por exceso de velocidad
- · Alarma por alejamiento
- · Registrador GPS en tarjeta SD
- · Registro de temperatura y humedad en tarjeta SD
- · Control de servo con acelerómetro
- · Sensor de caída con aviso a emergencias Bluetooth
- · MQTT (IoT): Control de led RGB
- · MQTT (IoT): Estación meteorológica
- · MQTT (IoT): Control domótico
- · Robot con servos controlador por Bluetooth
- · Robot con motores DC - Control Bluetooth
- · Robot con motores DC – Evita obstáculos
- · Robot con motores DC – Sigue líneas
- · Brazo robótico con servos – Control PC



Juanjo López
info@arduinoblocks.com