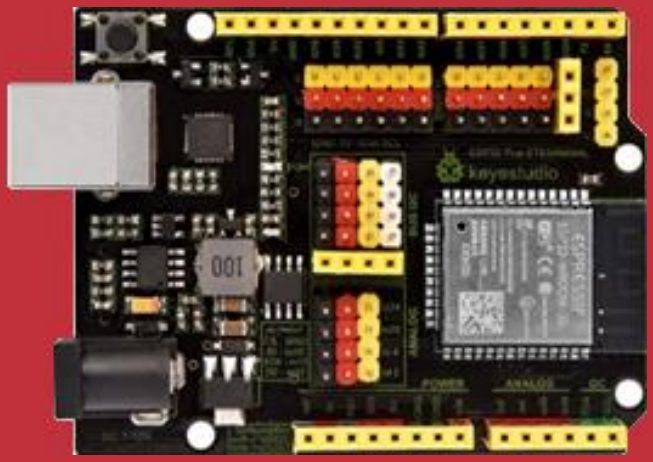
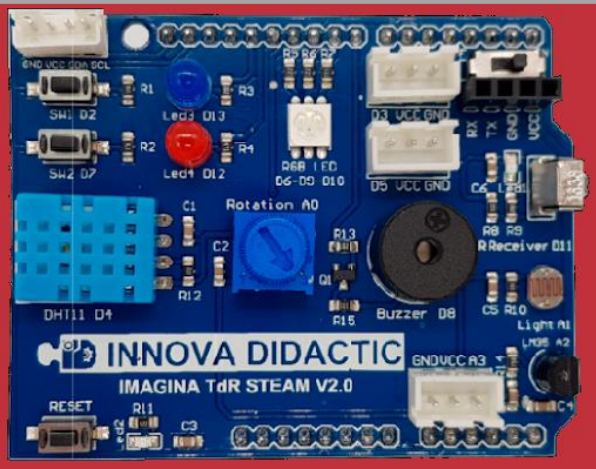
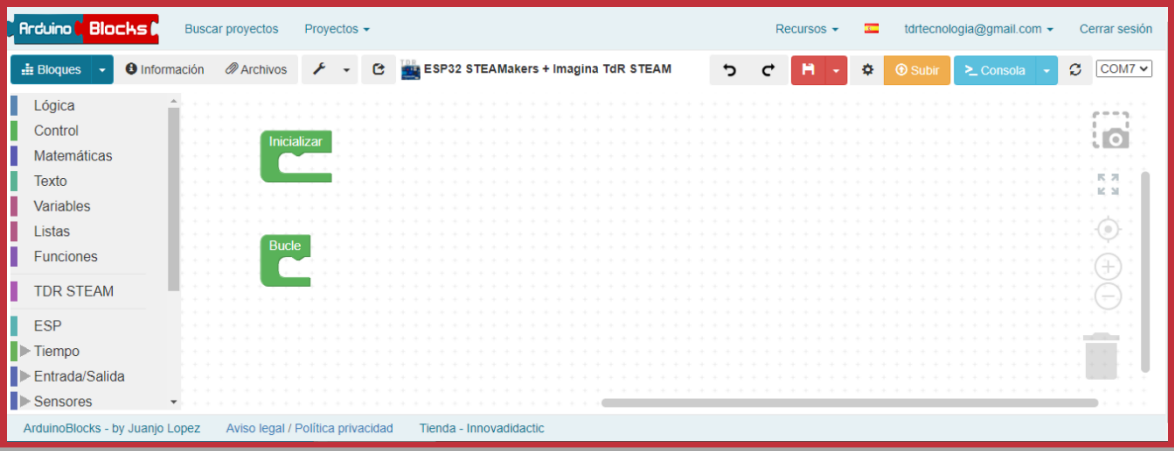
Physical Computing

ESP32 Plus STEAMakers Imagina TDR STEAM ArduinoBlocks







Versión 4.0

(Junio 2022)

INNOVA DIDACTIC

Este manual ha sido elaborado por Fernando Hernández y el equipo de desarrollo y formación de Innova Didactic.

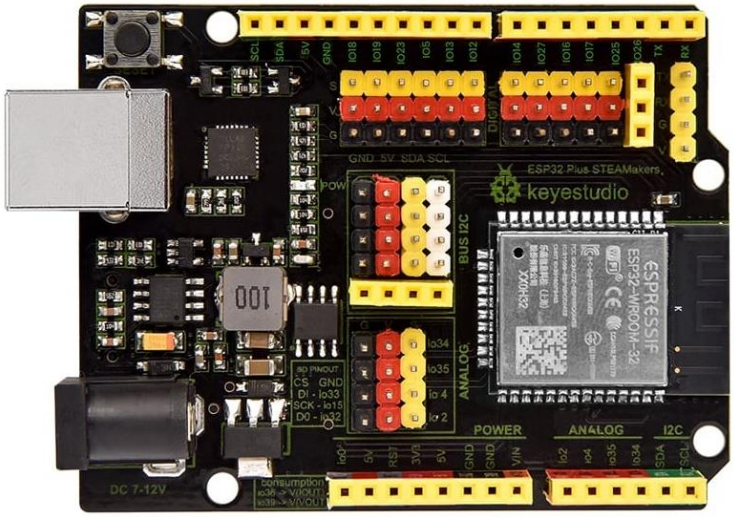
El documento ha sido revisado por Juanjo López, Eduard Casadevall y Toni Moreno. Y se ha desarrollado con el soporte de ArduinoBlocks, Robolot Team e Innova Didàctic S.L.

Cualquier sugerencia a: tdrtecnologia@gmail.com

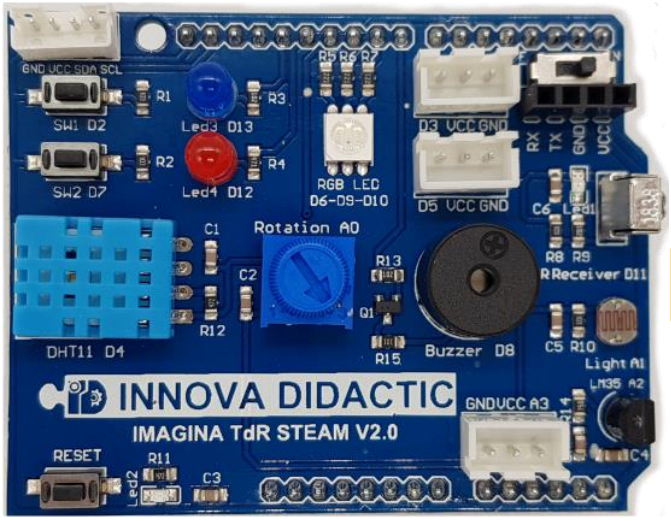
Índice

1 Introducción.

El presente manual pretende ser una herramienta base para iniciarse en el mundo de la programación, la electrónica, las comunicaciones y la robótica utilizando para ello las placas: ***ESP32 Plus STEAMakers***, la placa ***Imagina TDR STEAM*** y el entorno de programación **ArduinoBlocks**. Con estas tres herramientas podremos desarrollar infinidad de proyectos desde un nivel muy básico hasta proyectos de alta complejidad utilizando todo el potencial que nos ofrecen. Además, tiene compatibilidad con la mayoría de funcionalidades de Arduino UNO, pero con mayor potencia y versatilidad.



ESP32 Plus STEAMakers



Imagina TdR STEAM v2.0

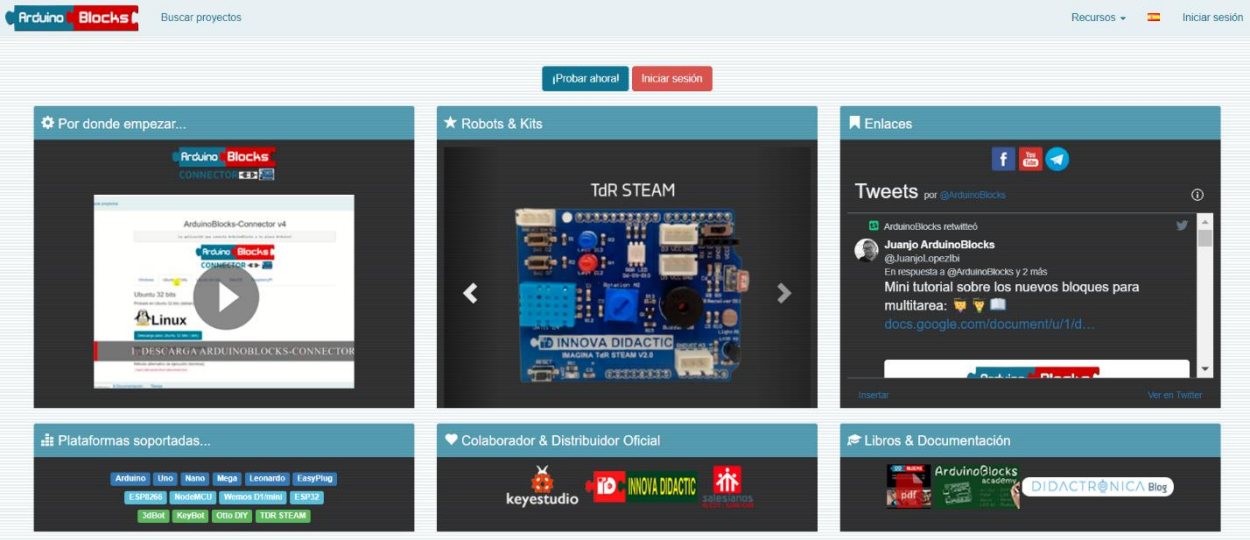
La placa ***Imagina TDR STEAM*** es una ***Shield*** (placa/escudo). Significa que es una placa que tiene que ir colocada sobre otra que contiene el sistema de control. La placa ***Imagina TDR STEAM*** tiene numerosos sensores y actuadores que permitirán hacer infinidad de proyectos. Además, la nueva placa ***ESP32 Plus STEAMaker*** nos ofrece una cantidad ilimitada de prestaciones al estar basada en un microcontrolador de 32 bits con conectividad Wifi y Bluetooth integrada en la propia placa y zócalo para tarjetas µSD para el almacenamiento de datos. También dispone de conexiones para todas las entradas y salidas con posibilidad de tener la alimentación adjunta y puertos de expansión I2C para poder conectar diferentes dispositivos directamente en la placa. La placa está basada en el microcontrolador **ESP32-WROOM-32**.

Las principales especificaciones técnicas:

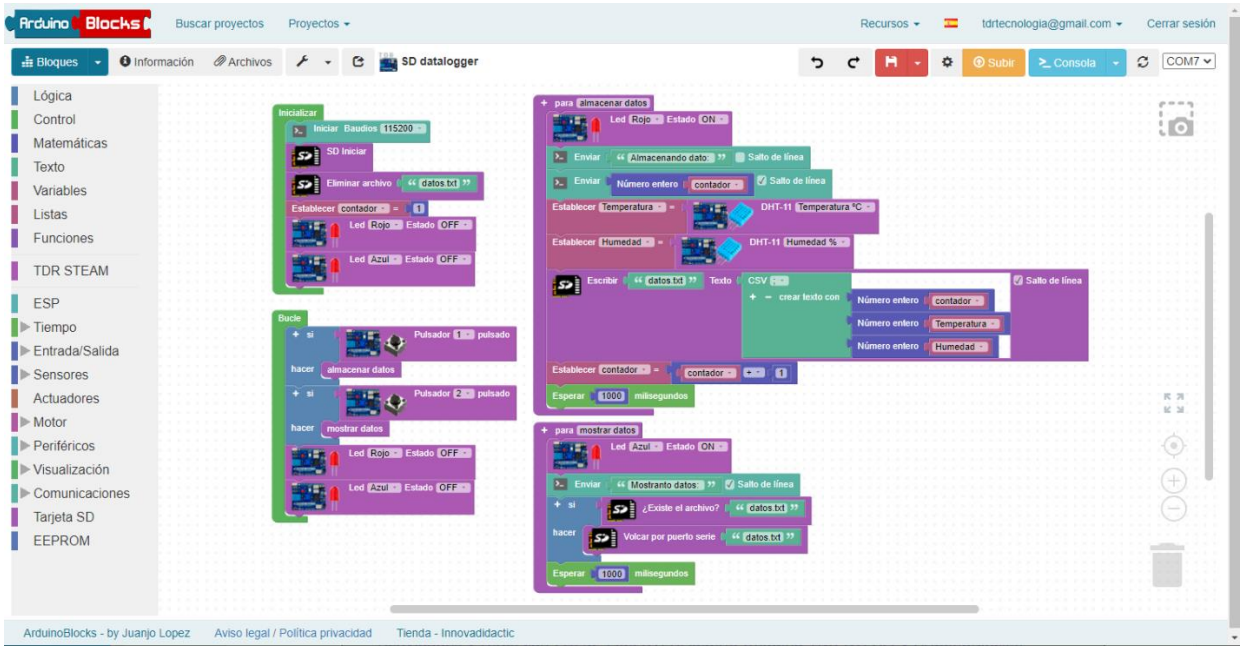
- CPU Xtensa LX6
- CPU de 2 núcleos
- Arquitectura de 32 bits
- Frecuencia de trabajo de 160 MHz
- Comunicaciones Wifi y Bluetooth
- Memoria RAM
- Memoria Flash
- 11 conversores Analógico-Digital de 12 bits de resolución
- 2 conversores Digital-Analógico de 8 bits
- Conectividad I2C

El manual presenta una serie de actividades guiadas y retos para aprender a programar de una manera entretenida y divertida mientras aprendemos conceptos relacionados con las **S.T.E.A.M.** (Science, Technology, Engineering, Arts, Math). También permite adentrarnos en el mundo de las comunicaciones Bluetooth y Wifi y, por tanto, en el mundo del **IoT**.

Para realizar la programación con las placas ***ESP32 Plus STEAMakers*** e ***Imagina TDR STEAM*** utilizaremos un lenguaje de programación visual basado en bloques llamado ***ArduinoBlocks***.



Hay quien podría pensar que un lenguaje de este estilo es muy básico y limitado, pero ya veremos a lo largo del manual la gran potencialidad y versatilidad que tiene este programa desarrollado por Juanjo López, con el soporte del equipo de desarrollo y formación de Robolot Team, e Innova Didàctic S.L.



Con la placa ***Imagina TDR STEAM*** tenemos la mayoría de sensores y actuadores que necesitamos para poder introducirnos en el mundo de la programación de entornos físicos (*Physical Computing*). También tenemos conexiones de expansión para poner conectar más elementos externos.

Página del entorno de programación: ArduinoBlocks.

www.arduinoblocks.com

Página para comprar material: Innova Didactic S.L.

shop.innovadidactic.com

Páginas de información:

- **Innova Didactic:**

  https://content.innovadidactic.com/category/tutoriales

- **Didactrónica:**

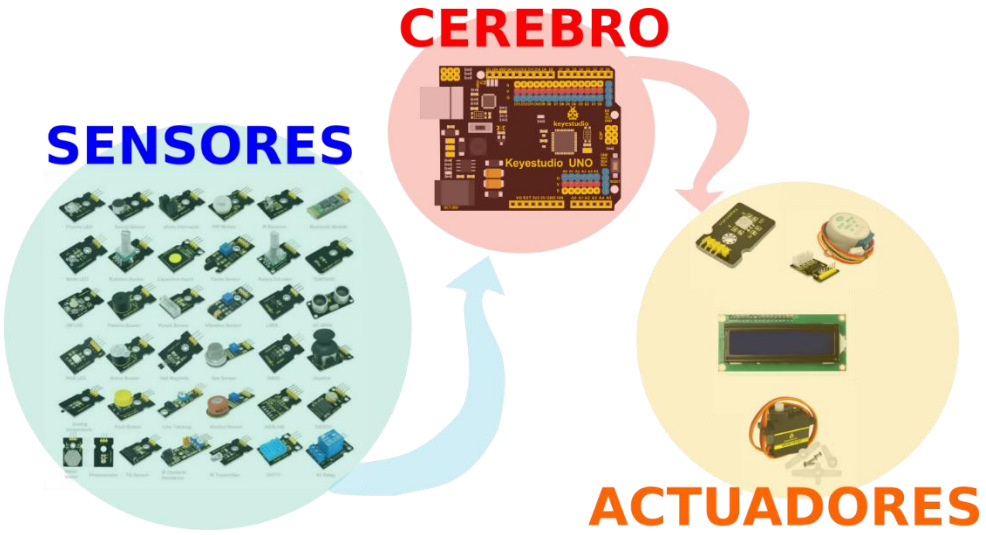
  www.didactronica.com

E-mail documentación:

tdrtecnologia@gmail.com

2 Funcionamiento de un sistema de control programado.

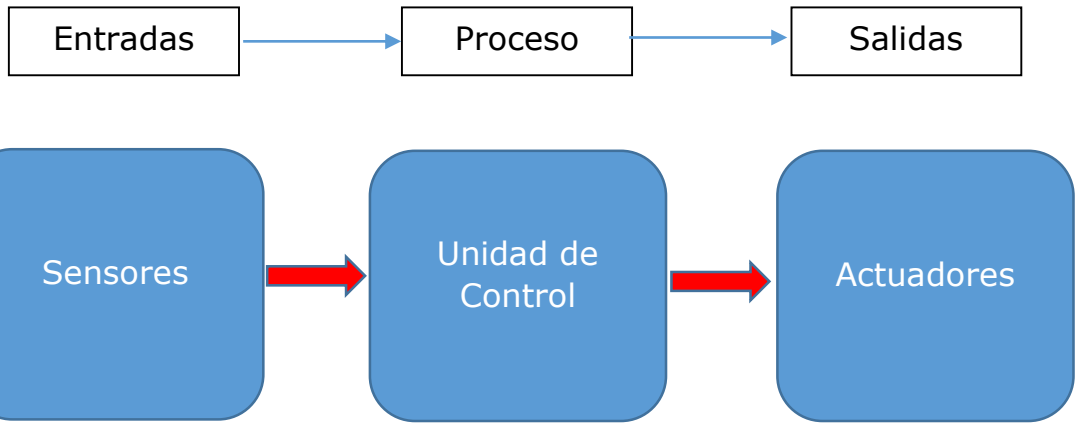
Un sistema de control programado funciona de manera similar a la de un ser humano. Cuando nuestro cerebro recibe información de los sentidos; oído, olfato, gusto, vista y tacto; analiza esta información, la procesa y da órdenes a nuestros músculos para realizar movimientos, dar estímulos a las cuerdas vocales para emitir sonidos, etc. Los 5 sentidos equivalen a entradas de información y la voz los músculos serían las salidas.

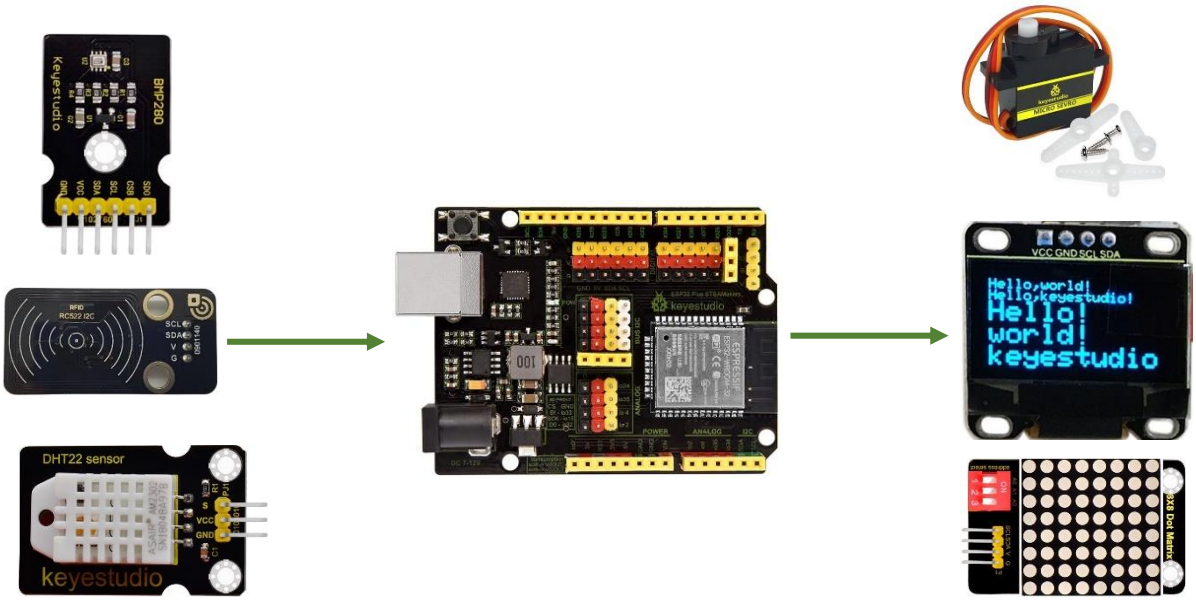


En el caso de un sistema de control programado, un microcontrolador hace la función de cerebro. Este componente electrónico unas entradas de información donde se conectan los sensores de luz (LDR), temperatura (NTC), sonido… y también tiene salidas, donde se conectan los motores, leds, zumbadores, etc.

La diferencia principal es que, así como nuestro cerebro ha ido aprendido lo que tiene que hacer a lo largo de nuestra vida a base de estímulos, el sistema programado tiene su memoria vacía, es decir, no sabe lo que debe hacer. Entonces nosotros tenemos que decirle como tiene que actuar en función de las señales que recibe de los sensores. Estas órdenes se generan mediante el código de programa que introducimos en nuestro sistema de control.

En todo sistema de control automático, los sensores (entradas) convierten las magnitudes físicas, químicas, etc. en eléctricas, creando así la información. Esta información se envía a la unidad de control (proceso) que procesa dicha información y genera las órdenes mediante señales eléctricas que serán convertidas en los actuadores (salidas) en otro tipo de magnitudes.





3 Componentes de la placa Imagina TDR STEAM.

La placa **Imagina TDR STEAM** es una placa didáctica desarrollada por el equipo ***ROBOLOT TEAM*** que presenta la gran ventaja de tener una gran cantidad de sensores, actuadores y conexiones de expansión incorporados directamente en ella. Únicamente hay que conectar esta placa (*shield*) a una placa Arduino UNO (en nuestro caso, a una placa compatible llamada ***ESP32 Plus STEAMaker***) y ya está todo listo para empezar a programar.

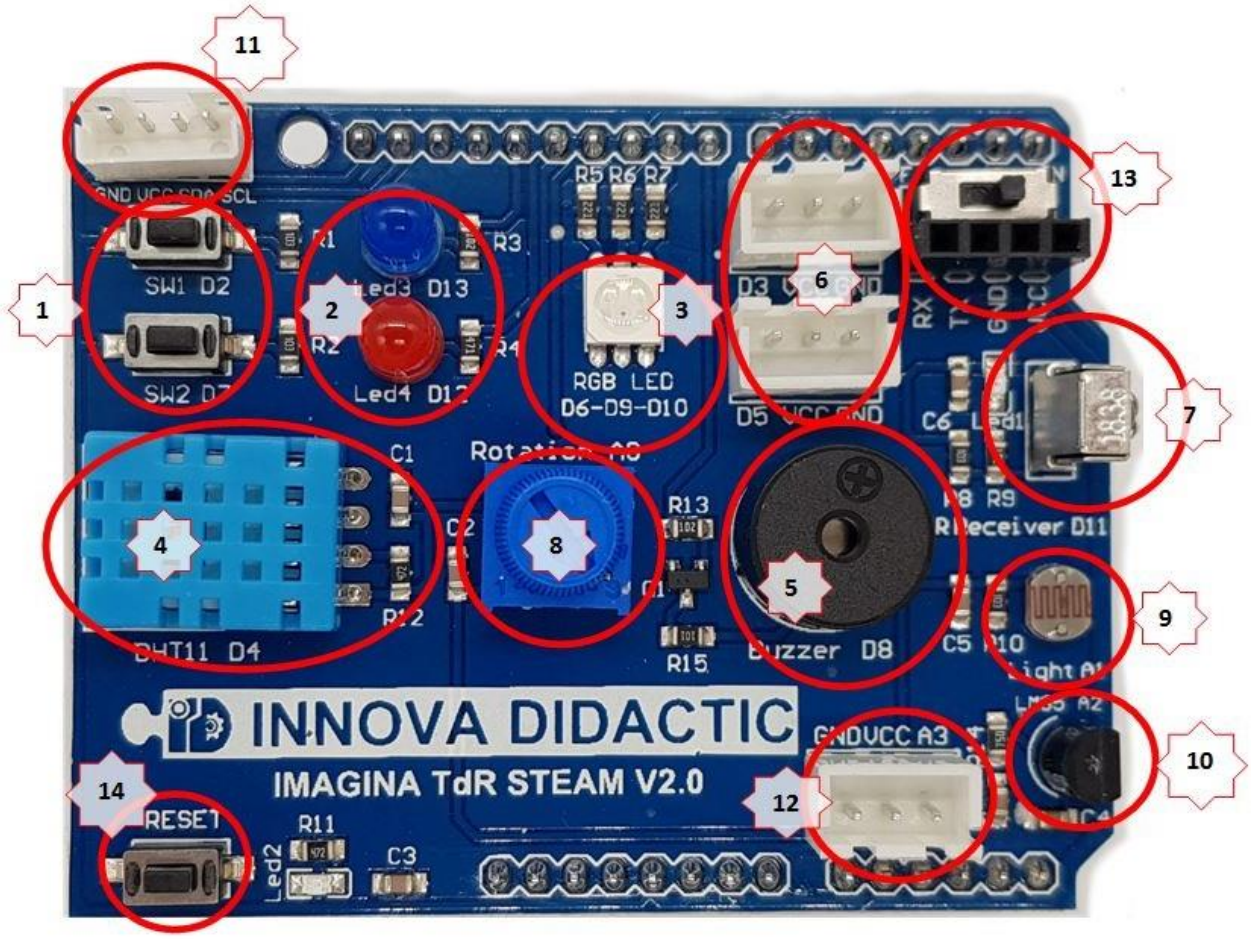
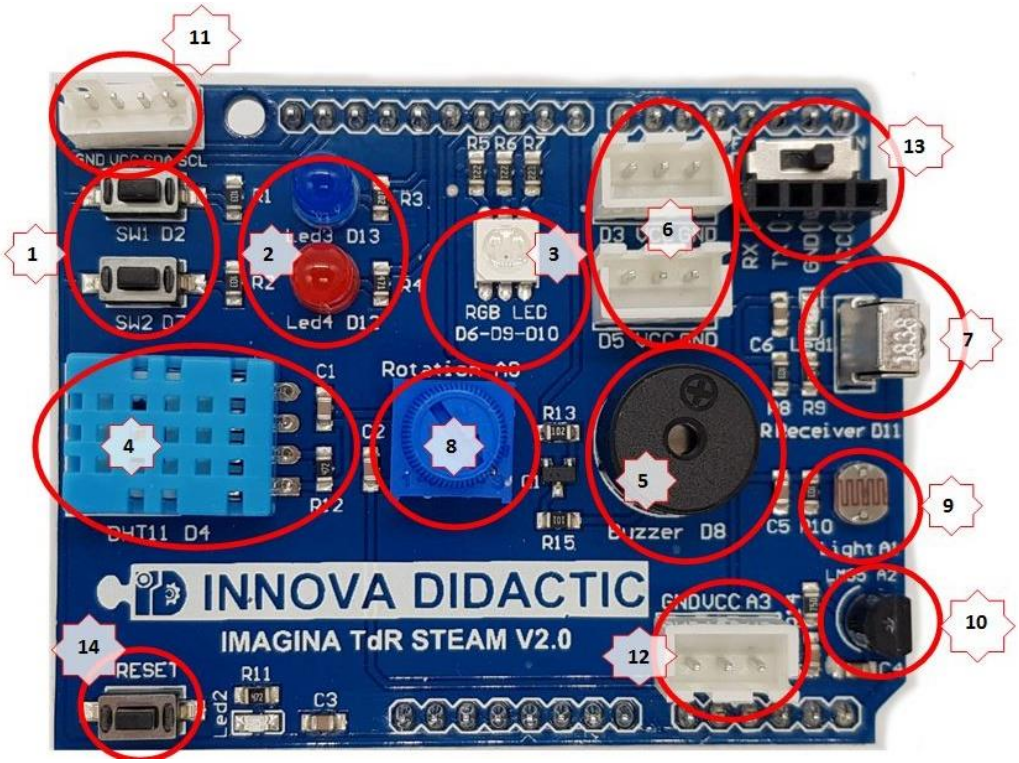


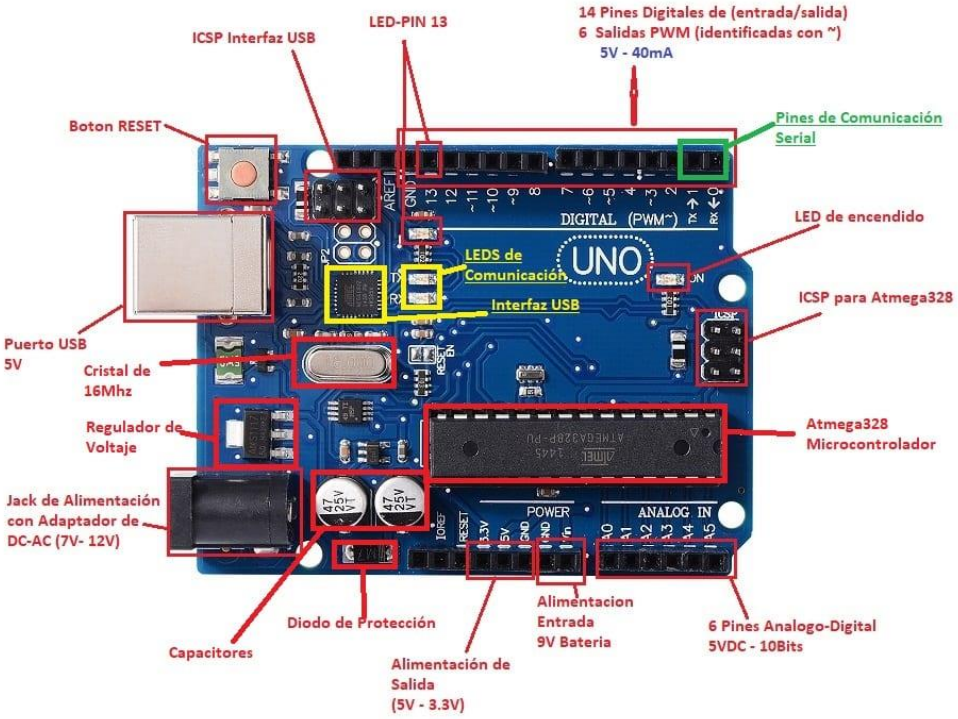
Tabla con la relación de elementos que hay en la placa Imagina TDR STEAM y sus conexiones:

	Sensor/ Actuador/ Módulo	Pin de conexión
1	Dos pulsadores (SW1, SW2)	D2 y D7
2	Dos leds (Azul Led3 y Rojo Led4)	D13 y D12
3	Led RGB	D6-D9-D10
4	Módulo DHT11 Sensor de Temperatura y Humedad	D4
5	Zumbador o Piezoeléctrico	D8
6	Dos puertos (Entradas/Salidas) digitales	D3 y D5
7	Módulo receptor de infrarrojos (IR)	D11
8	Módulo potenciómetro giratorio	A0
9	Sensor de luminosidad (LDR)	A1
10	Sensor de temperatura (LM35)	A2
11	Interface I2C compatible con sensores y módulos Keyestudio	SDA-A4 SCL-A5
12	Puerto Entrada Analógico	A3
13	Conexión de comunicaciones Bluetooth y Wifi (Switch On/Off)	Rx - Tx
14	Botón de reinicio.	-

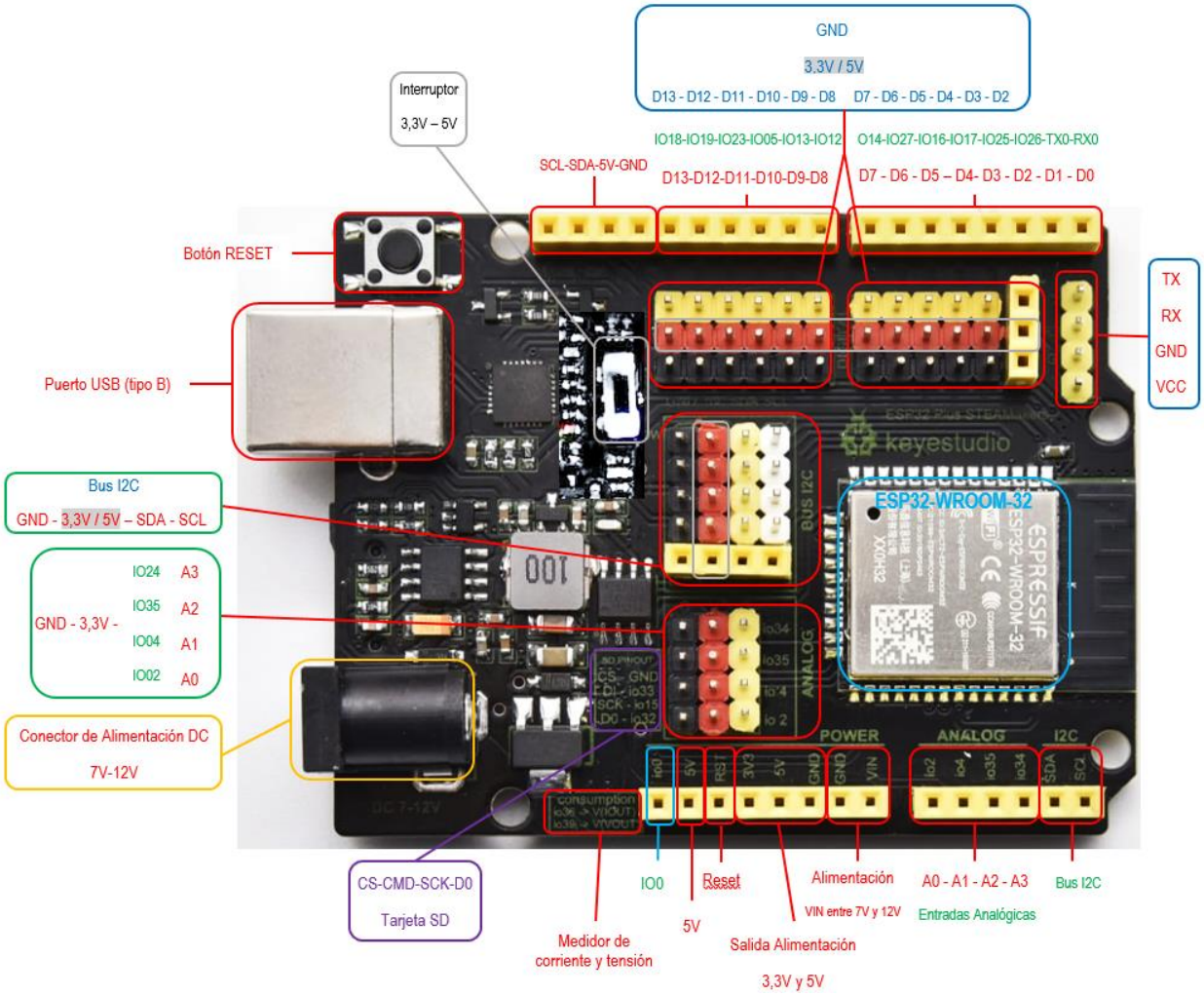


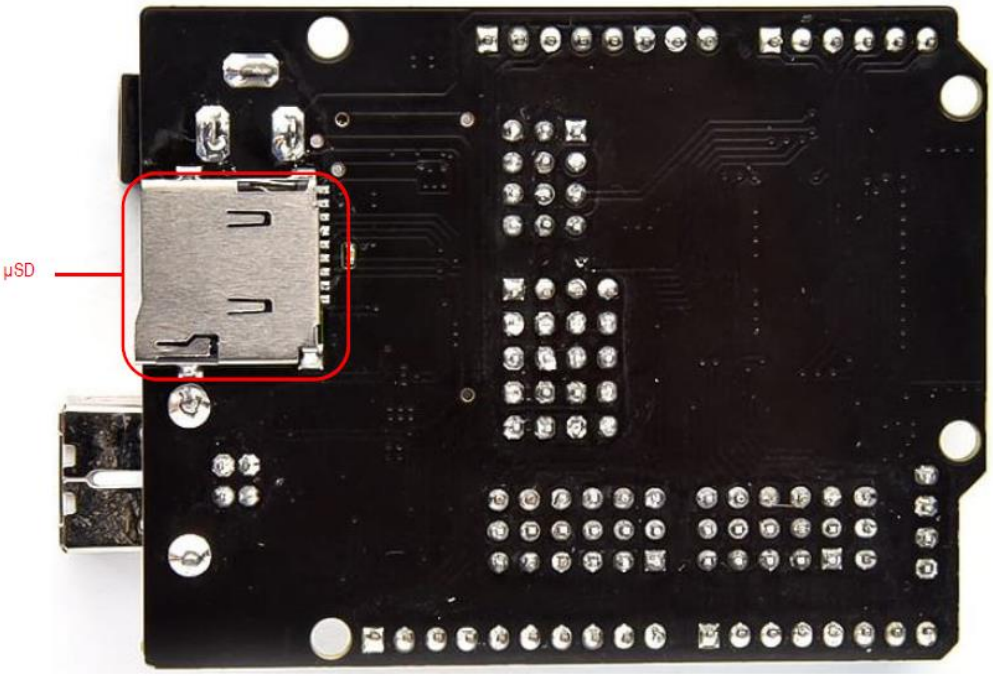
4 Placa de control ESP32 Plus STEAMakers.

La placa de control que gestiona la placa ***Imagina TDR STEAM*** está basada en una placa ***Arduino UNO***. Arduino es una plataforma de prototipos electrónicos de código abierto (Open-Source) basada en hardware y software libre, flexible y fácil de usar, con una comunidad muy grande que genera muchísima información. Esta plataforma permite crear diferentes tipos de sistema para diferentes usos. Puede ser empleada por *artistas, diseñadores*, ingenieros, profesores, alumnos, etc., y en general, cualquier persona que esté interesada en crear objetos o entornos interactivos, prototipos, sistemas robóticos, etc.



La placa ***ESP32 Plus STEAMakers*** tiene los siguientes elementos:





Al ser Arduino un hardware libre existen multitud de fabricantes que han desarrollado versiones basadas en Arduino. Uno de esos fabricantes es ***Keyestudio***, que ha desarrollado juntamente con el equipo de Innova Didactic una placa compatible con Arduino, pero basada en un ESP32. La nueva placa es la ***ESP32 Plus STEAMakers***.

Las características más importantes de esta placa son:

- Microcontrolador Tensilica Xtensa 32-bit LX6 a 160MHz.
- Conectividad Wifi 802.11 b/g/n/e/i.
- Conectividad Bluetooth 4.2 y modo BLE.
- Zócalo para tarjetas µSD.
- 14 entradas y salidas digitales con alimentación.
- Conector serie hembra con alimentación.
- Conector I2C para conectar hasta 5 dispositivos a la vez sobre la misma placa.
- Conector hembra I2C para conexión de una pantalla OLED.
- Conector de Reset.
- Conector de 5V
- Conector de 3.3V
- Interruptor 3.3-5V seleccionable para cambiar entre estas dos tensiones en algunos pines de alimentación.
- Entradas y salidas analógicas.
- Sensor Hall y de temperatura integrado.
- 2 convertidores Digital-Analógico (DAC) de 8 bits.
- 16 convertidores Analógico-Digital (ADC) de 12 bits.
- 16 canales PWM.
- 2 UART.
- 2 canales I2C.
- 4 canales SPI.
- 448Kb ROM.
- 520 KB SRAM.
- 8KB+8KB SRAM en RTC.
- 1kbit eFUSE.
- 512 bytes Memoria Flash (EEPROM).
- 10 sensores táctiles.
- 4 temporizadores internos de 64 bits.

No están disponibles todas las características del controlador ESP-WROOM-32, ya que algunos pines tienen funciones dobles y se utilizan en la placa de forma específica (como, por ejemplo, para controlar la tarjeta SD). Pero la mayoría de funciones se pueden utilizar, además de disponer la placa ***ESP32 Plus STEAMakers*** de una mejor conexión de elementos debido a los pines

para conectores tipo Dupont de entrada y salida, de I2C y de alimentación. Además, algunos pines de alimentación pueden cambiar su valor (3,3V o 5V) mediante un interruptor en función de nuestras necesidades.

Las conexiones de la placa ***Imagina TDR STEAM*** con la placa ***ESP32 Plus STEAMakers*** son las siguientes (igual que si utilizamos cualquier placa compatible con Arduino UNO):

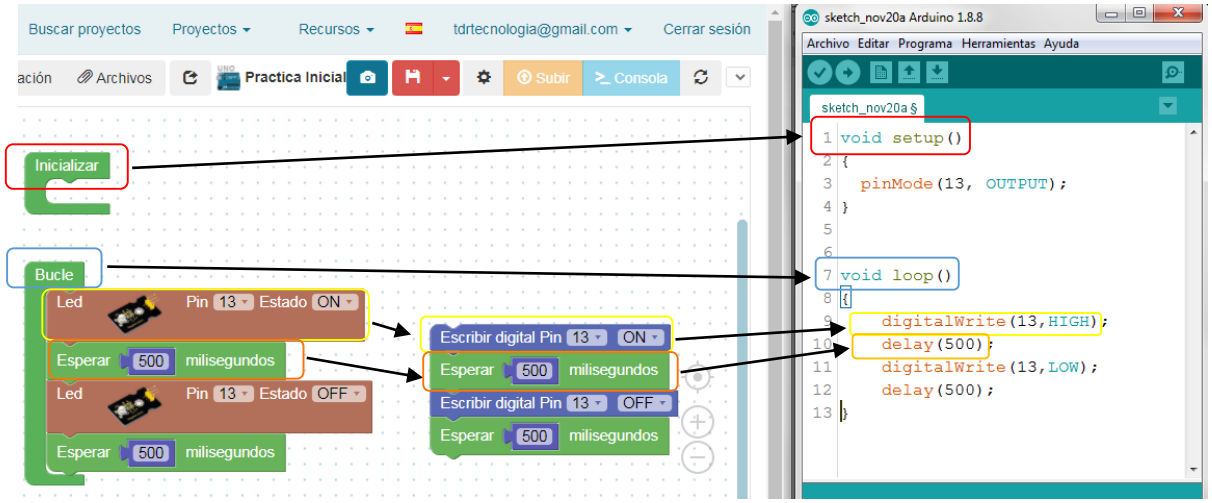
Conexión	Descripción	Tipo
D0	Rx	Comunicación PC/Bluetooth/Wifi
D1	Tx	
D2	Pulsador SW1	Entrada digital
D3	Libre	E/S digital
D4	DHT11 (sensor de humedad + temperatura)	Entrada digital
D5	Libre	E/S digital
D6	RGB (rojo)	Salida digital
D7	Pulsador SW2	Entrada digital
D8	Zumbador	Salida digital
D9	RGB (verde)	Salida digital
D10	RGB (azul)	Salida digital
D11	Sensor infrarrojos	Entrada digital
D12	Led rojo	Salida digital
D13	Led azul	Salida digital
A0	Potenciómetro	Entrada analógica
A1	LDR (sensor de luz)	Entrada analógica
A2	LM35 (sensor de temperatura)	Entrada analógica
A3	Libre	Entrada analógica
A4	SDA	I2C
A5	SCL	

Compatibilidad y descripción de pines:

Importante*: *Todos los pines IOxx son entradas y salidas digitales, algunas con más funciones. Utilizando la comunicación Wifi no funciona el ADC2.

Arduino UNO	Imagina TDR STEAM	ESP32 Plus STEAMakers		
PIN	Función	PIN	Función	Función ampliada
0	Rx	IO03	Rx	UART 0 RX
1	Tx	IO01	Tx	UART 0 TX
2	Pulsador	IO26	ADC2 CH9	DAC2
3	Libre	IO25	ADC2 CH8	DAC1
4	DHT11	IO17		UART 2 TX
5	Libre	IO16		UART 2 RX
6	Led RGB – R	IO27	ADC2 CH7	ADC2-7 / TOUCH7
7	Pulsador	IO14	ADC2 CH6	ADC2-6 / TOUCH6
8	Zumbador	IO12	ADC2 CH5	ADC2-5 / TOUCH5
9	Led RGB – G	IO13	ADC2 CH4	ADC2-4 / TOUCH4
10	Led RGB – B	IO05		VSPI CSO
11	Infrarojos	IO23		VSPI MOSI
12	Led Rojo	IO19		VSPI MISO
13	Led Azul	IO18		VSPI CLK
GND		GND		
AREF		RESET		
SDA	I²C	IO21		
SCL		IO22		
A0	Potenciómetro	IO02	ADC2 CH2	
A1	LDR	IO04	ADC2 CH0	
A2	LM35 (temp.)	IO36	ADC1 CH0	
A3	Libre	IO34	ADC1 CH6	
A4	I²C	IO38		
A5	I²C	IO39	ADC1 CH3	
VIN		VIN		
GND		GND		
GND		GND		
5V		5V		
3V3		3V3		
RST		RESET		
5V		5V		
		IO00	No conectar!	
-		IO32	D0 – Tarjeta SD	
-		IO15	CLK – Tarjeta	
-		IO33	CMD – Tarjeta	
-		IO35	IOUT	Medidor de corriente y tensión
-		IO37	VOUT	

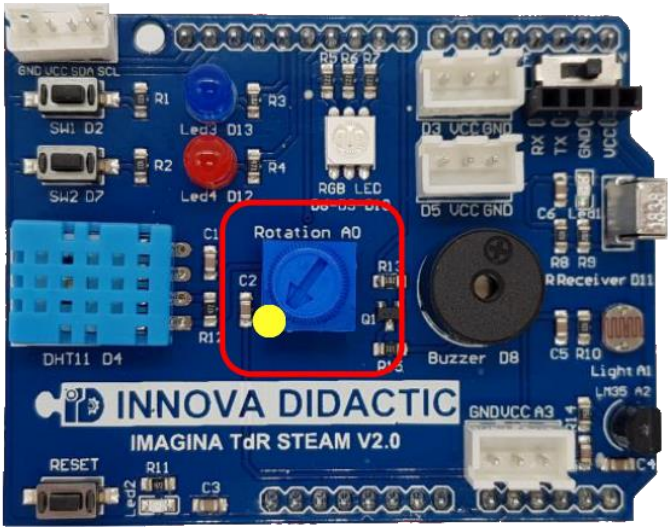
Para realizar la programación la podemos hacer mediante la IDE de Arduino o mediante ArduinoBlocks. Como podemos ver son dos sistemas diferentes. En la IDE de Arduino la programación se realiza mediante instrucciones (derecha de la imagen) y en ArduinoBlocks se realiza mediante bloques (izquierda de la imagen). En la siguiente imagen se hace una comparación de código entre ArduinoBlocks y Arduino IDE.



Utilizar ArduinoBlocks simplifica y hace más inteligible el código, lo que permite iniciarse en el mundo de la programación de un modo más amigable. ArduinoBlocks también permite programar de diversas formas, ya que tiene bloques específicos que realizan las mismas funciones pero que se pueden entender de forma más sencilla.

4.1 Información importante.

Para poder aprovechar los recursos de la placa Imagina TDR STEAM conjuntamente con la placa ESP32 Plus STEAMakers se utilizan una serie de conexiones en común. Es **muy importante** que el **potenciómetro esté en la posición izquierda** (punto de color amarillo), ya que comparte la conexión A0 (GPIO02) con el sistema de **grabación del programa**. Si no está en esa posición se producirá un error en el envío del programa.



También es muy importante que el ***Connector*** esté abierto siempre. La secuencia correcta de operación es la siguiente:

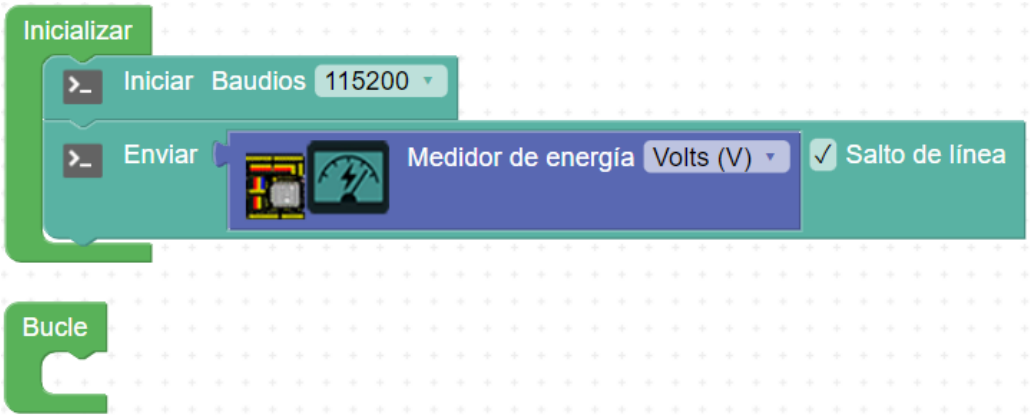
1. Conectar la placa ***ESP32 Plus STEAMakers*** al puerto USB con su correspondiente cable.
2. Abrir el ***Connector***.
3. Abrir ***ArduinoBlocks***. Aparecerá en puerto COM que ha asignado.

En el caso de no realizarlo en este orden puede suceder que no quede asignado el puerto COM de forma correcta. Si se produce un problema en el envío del programa puede ser que se haya roto la vinculación entre la placa y el ordenador, así que procederemos de la siguiente forma:

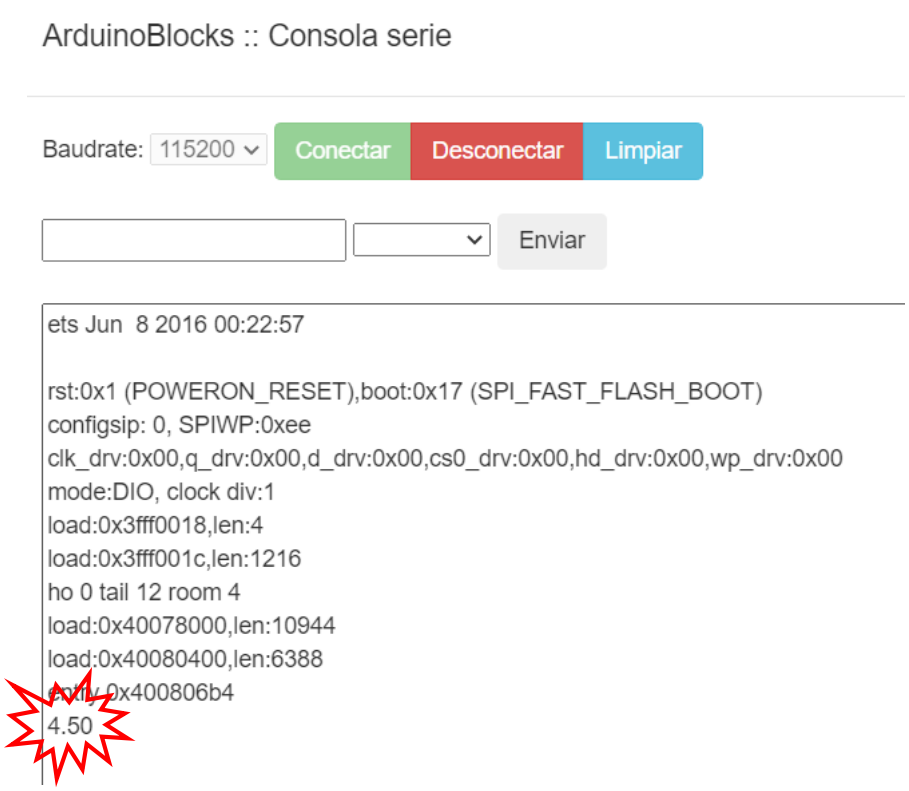
1. Desconectar la placa ***ESP32 Plus STEAMakers***.
2. Cerrar el ***Connector***.
3. Conectar la placa ***ESP32 Plus STEAMakers***.
4. Abrir el ***Connector***.

La placa de control ***ESP32 Plus STEAMakers***, como cualquier placa basada en ESP32 debe tener unos valores de alimentación mínimos. Gracias a que lleva implementado un medidor de tensión e intensidad podemos saber la tensión de alimentación en todo momento. Si la tensión de alimentación baja de aproximadamente 4,8V la placa no funcionará correctamente (sobre todo la transmisión wifi). Es recomendable realizar una verificación de la tensión que está entregando el puerto COM del ordenador. Si la tensión no es próxima a 5V deberemos cambiar el cable USB, alimentar el puerto USB de forma externa o alimentar la placa con una fuente de alimentación.

Podemos realizar un pequeño programa para verificar la alimentación de la placa ***ESP32 Plus STEAMakers***.



Aquí tenemos dos ejemplos, uno con la alimentación correcta y otro con una alimentación defectuosa (muy baja).

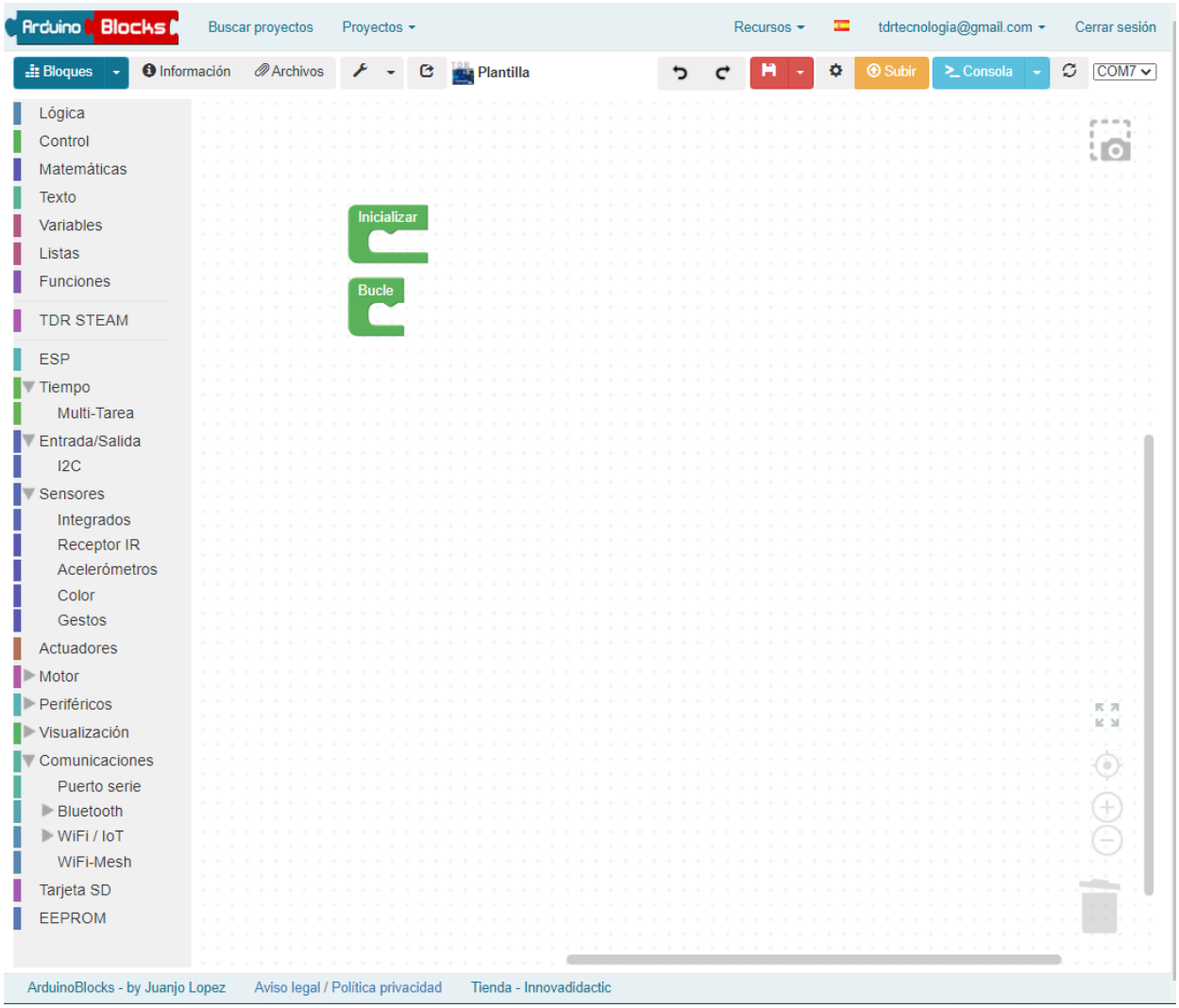


Sería un buen hábito colocar estos bloques al inicio de cualquier programa para asegurarnos de una buena alimentación, sobre todo si trabajamos directamente con la placa conectada al USB del ordenador y empezamos a conectar elementos externos a la placa (pantallas, leds, etc.).

También hay que tener en cuenta que cuando utilizamos la comunicación mediante Wifi deja de funcionar el ADC2, por lo que en la placa Imagina TDR STEAM no funcionarán ni el potenciómetro (A0) ni la LDR (A1).

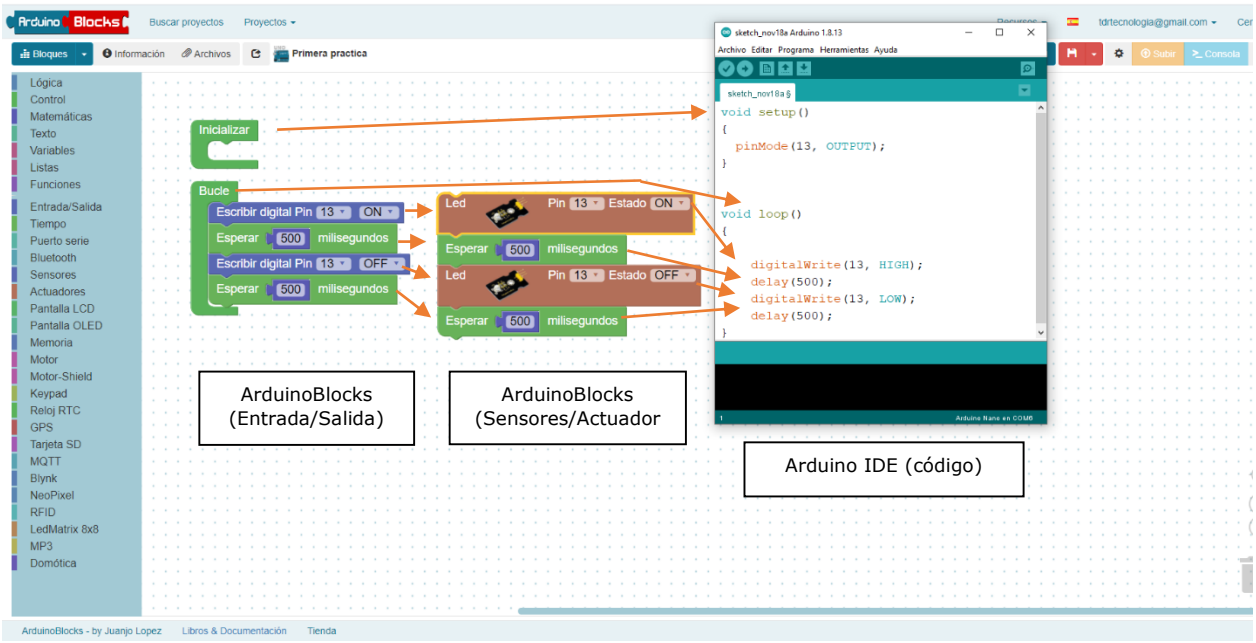
5 Programación con ArduinoBlocks.

ArduinoBlocks es un lenguaje de programación gráfico por "Bloques" creado por el profesor Juanjo López. Está pensado para que niñas y niños aprendan a programar con placas Arduino a partir de unos 8-9 años. Los distintos bloques sirven para leer y escribir las distintas entradas y salidas de la placa, así como programar funciones lógicas, de control, etc.



En este manual usaremos ArduinoBlocks dedicado al uso de la placa Imagina TDR STEAM, con estos bloques podremos programar las entradas y salidas de nuestra placa para que realice las tareas que queramos.

Podemos programar ArduinoBlocks de diferentes maneras ya que tiene múltiples bloques. También permite exportar el código para la IDE de Arduino. Para este manual utilizaremos, principalmente, la programación que se muestra en medio (bloques específicos para ***Imagina TDR STEAM***), ya que es más fácil de entender.

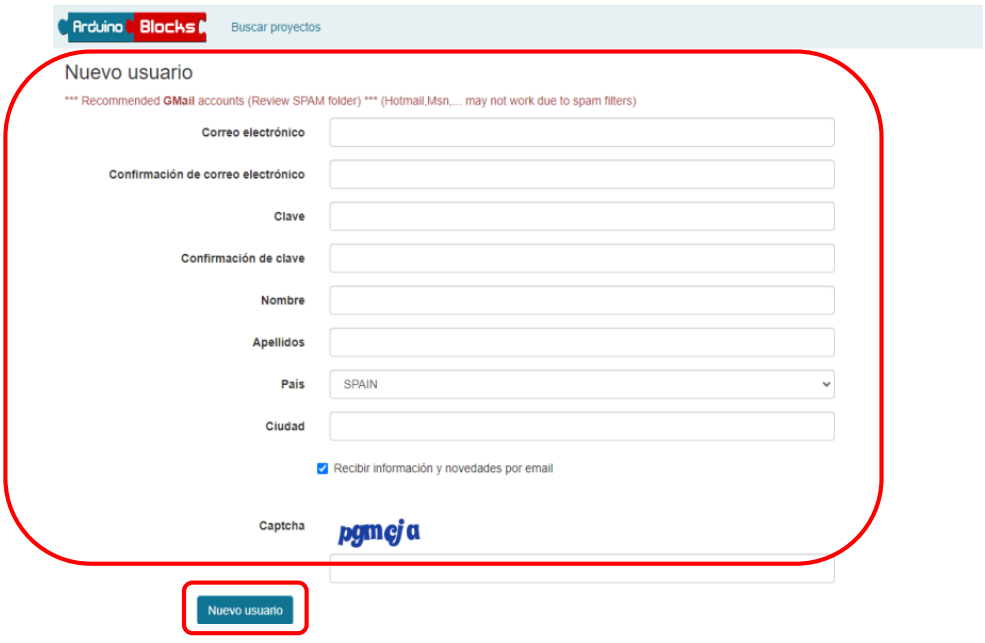


6 Instalación de ArduinoBlocks.

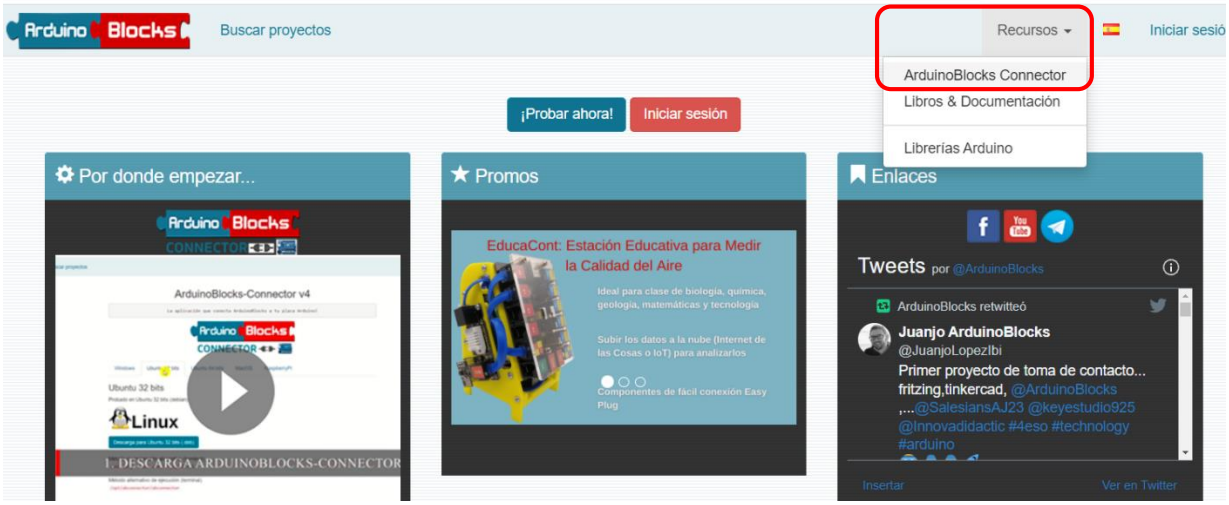
ArduinoBlocks trabaja on-line, pero tenemos que instalar un pequeño programa que será el encargado de conectar nuestro programa con la placa ***ESP32 Plus STEAMakers***. Este programa basado en Python se llama *Connector*. Deberemos tener siempre abierto este programa cuando enviemos el programa a la placa o cuando queramos hacer comunicaciones por la *Consola* o el *Serial Plotter* de ArduinoBlocks.

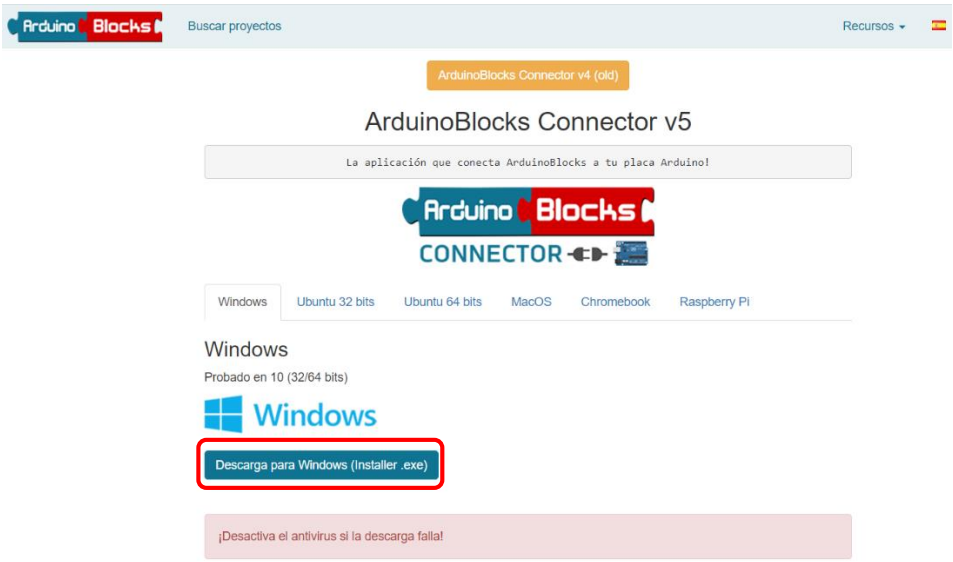
Primero deberemos crear una cuenta en ArduinoBlocks y después instalar el software ***Connector***.

- Para crear una ***cuenta*** en ***ArduinoBlocks***:

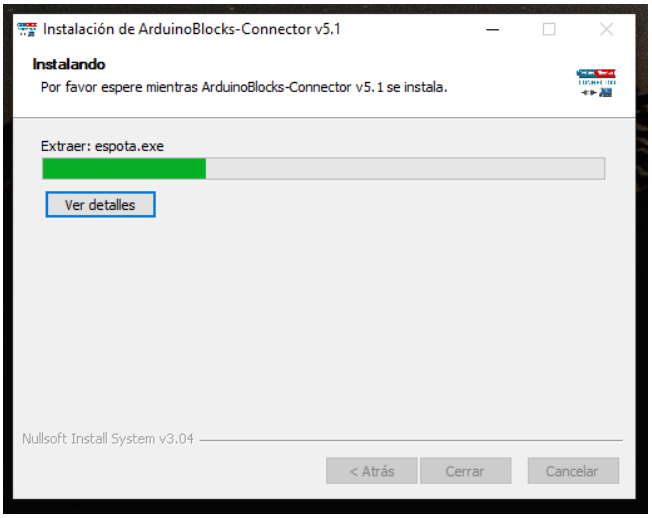


- Para instalar ***Connector***:





Aparecerá una ventana donde muestra el progreso de la instalación:

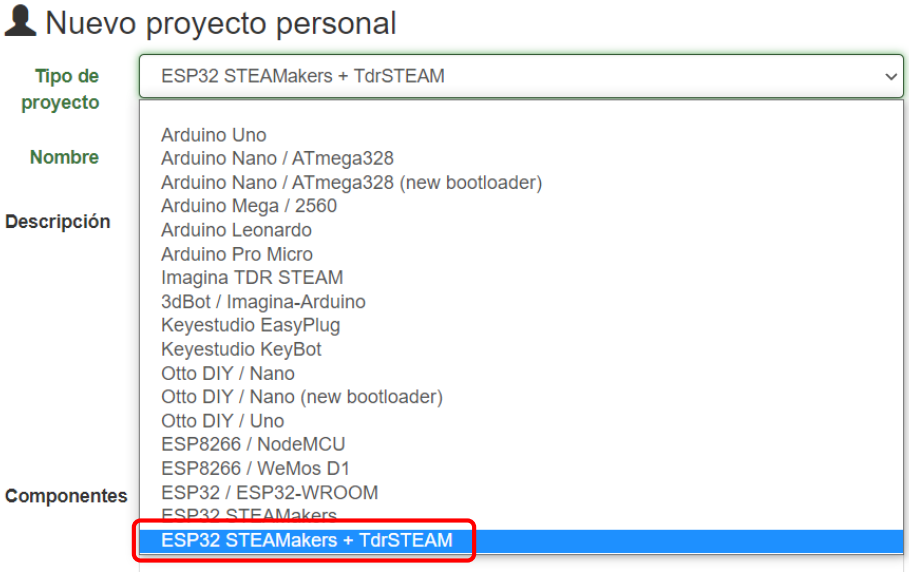


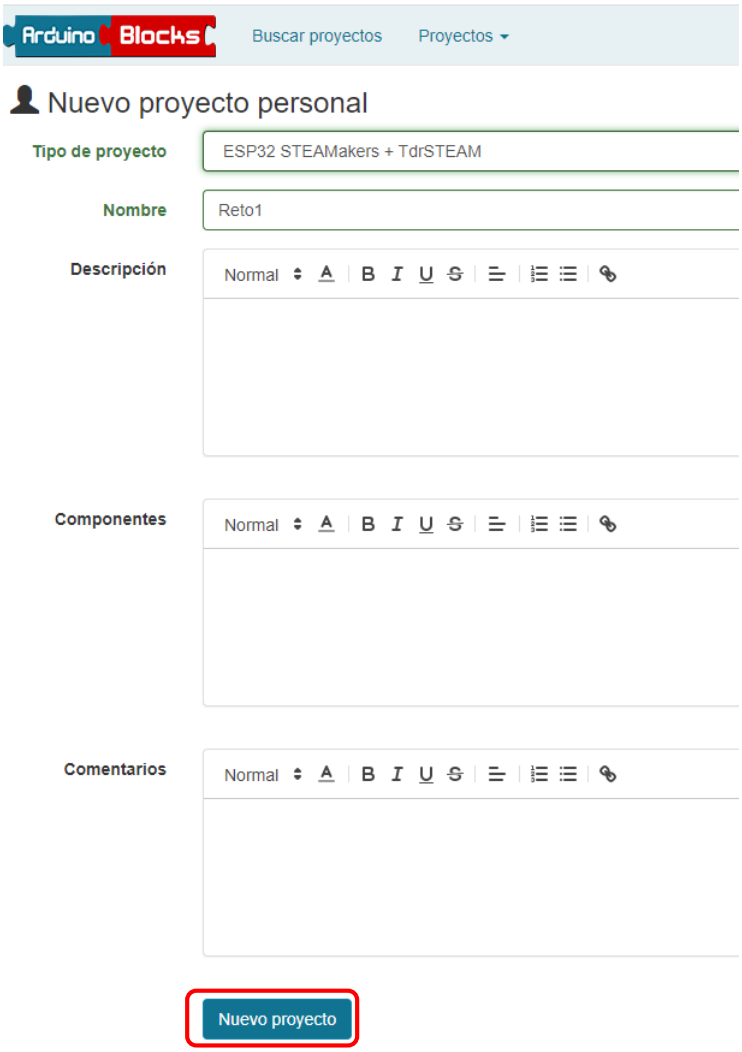
El icono que aparece después de la instalación es el siguiente:

Una vez instalado el programa, podemos subir un programa vacío para comprobar que funciona la transferencia a la placa.

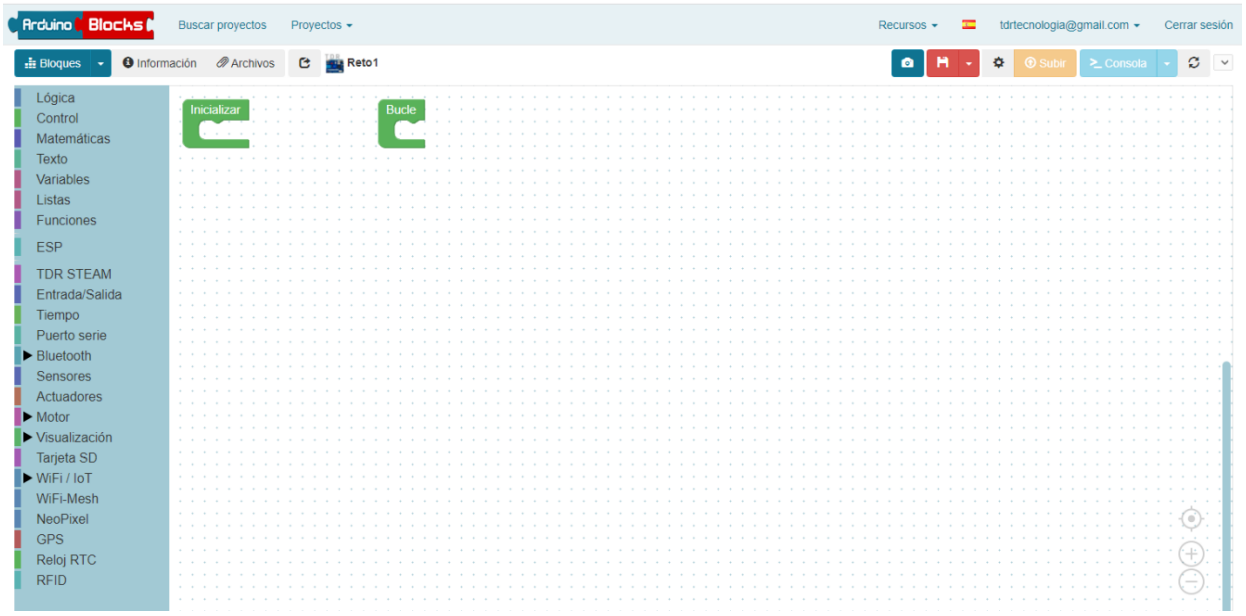
Entramos en la sesión de ArduinoBlocks y elegimos Iniciar un proyecto personal.

Y elegimos el *Tipo de proyecto*: ESP32 STEAMakers + TdrSTEAM.



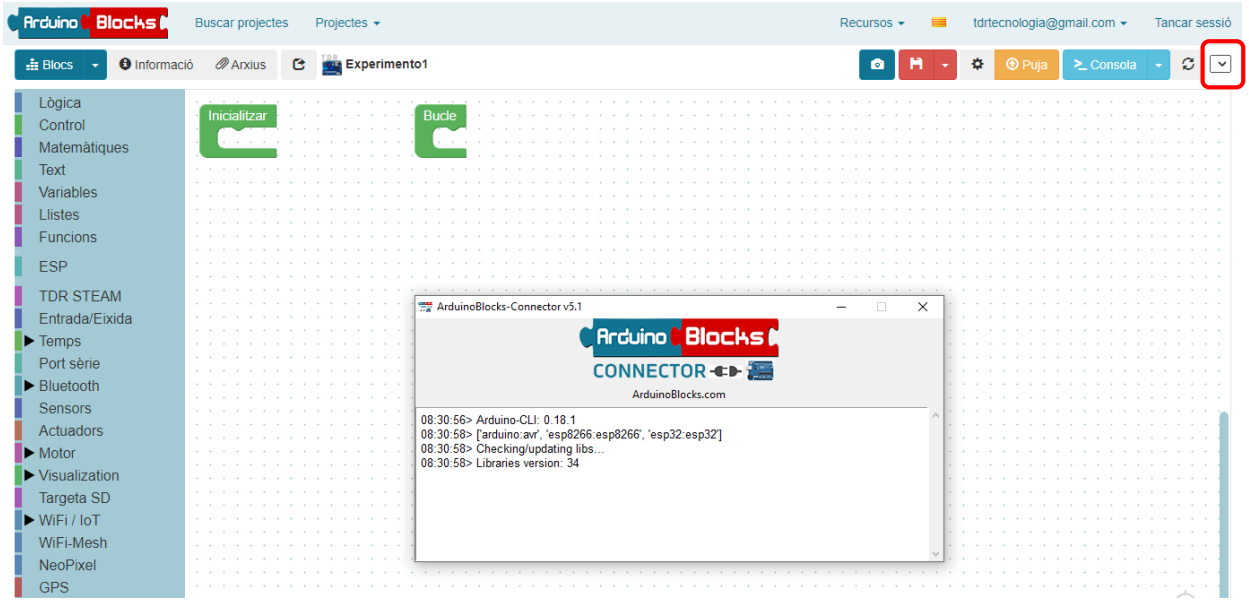


La pantalla del entorno de programación que aparece es la siguiente:

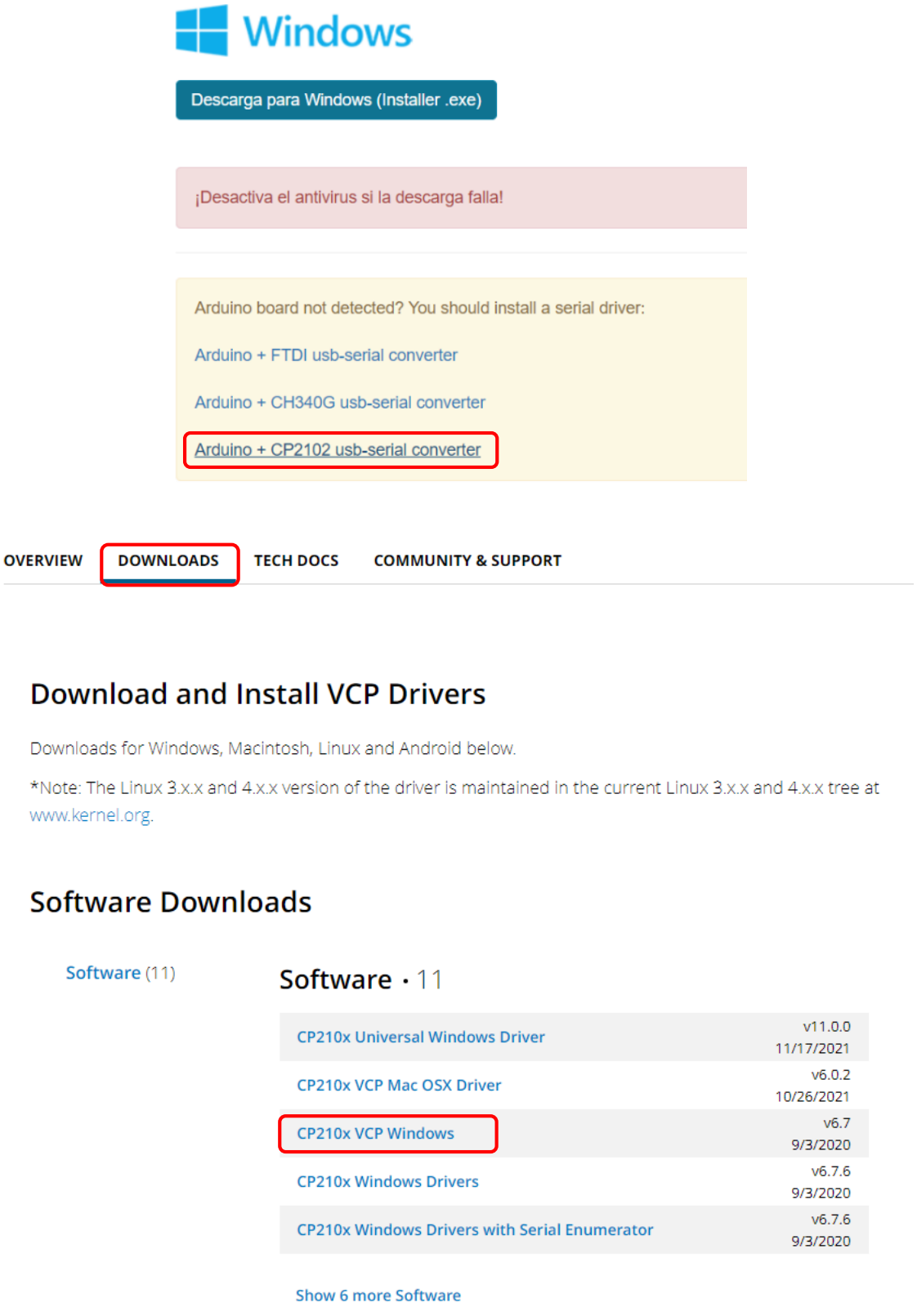


Abrimos el *Connector* (siempre con la placa conectada por USB al ordenador) y aparece la siguiente pantalla (que deberemos dejar siempre abierta):

Puede ser que sea necesario instalar los drivers para el puerto USB de la placa ***ESP32 Plus STEAMakers***, si cuando tenemos abierto el *Connector* y abrimos *ArduinoBlocks* y no aparece asignado ningún puerto COM. Si, en cambio, aparece ya el puerto COM asignado, podemos saltar la instalación del Driver para placas ESP32: ***Arduino + CP2102 usb-serial converter***.

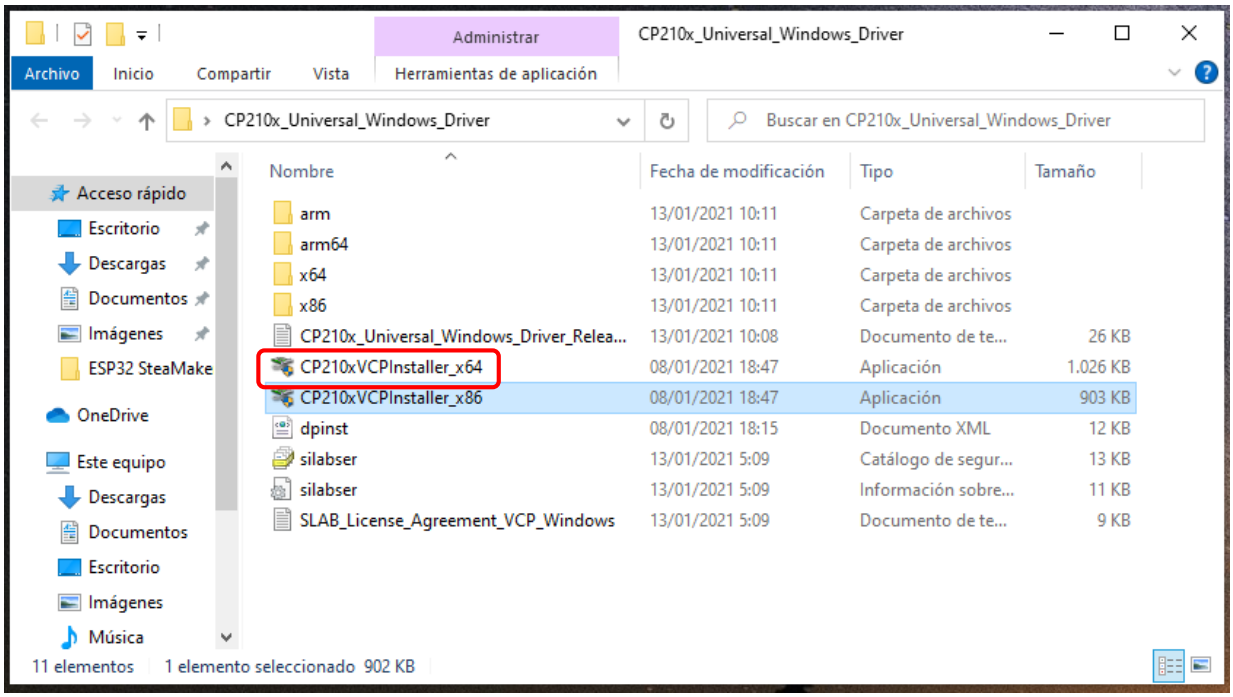


Debemos abrir otra vez la pestaña de *Recursos* → *ArduinoBlocks Connector* e instalar el Driver para las placas ESP32: *Arduino + CP2102 usb-serial converter*

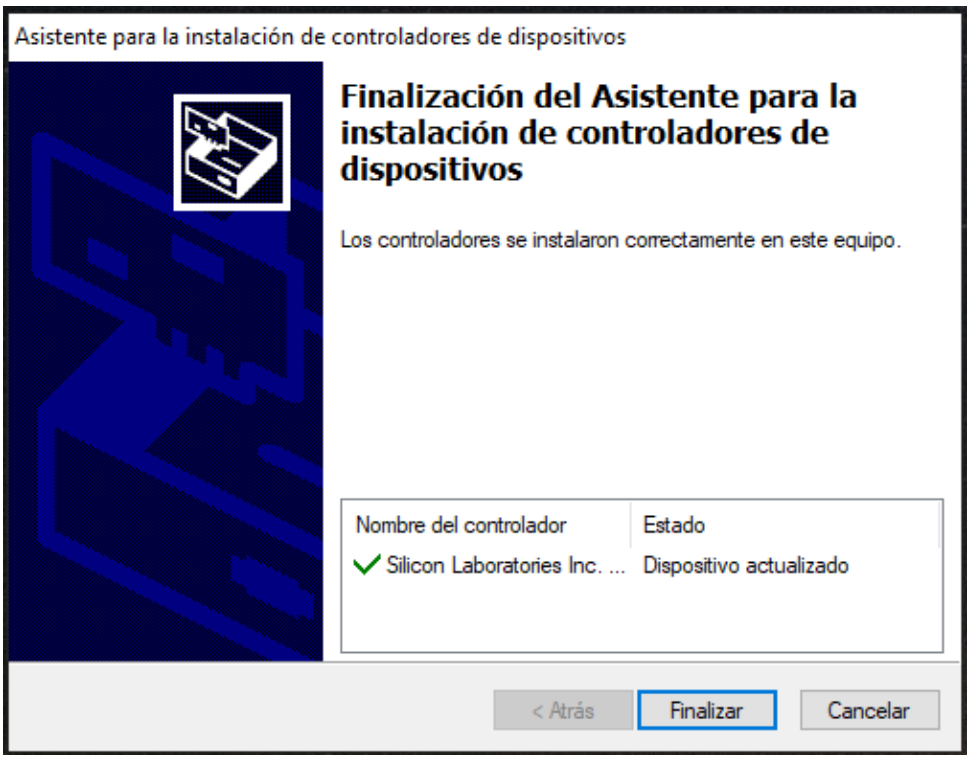


Descargará una carpeta comprimida en Zip: CP210x_VCP_Windows

Ejecutaremos el instalador para 32 o 64 bits (64 bits en nuestro caso).

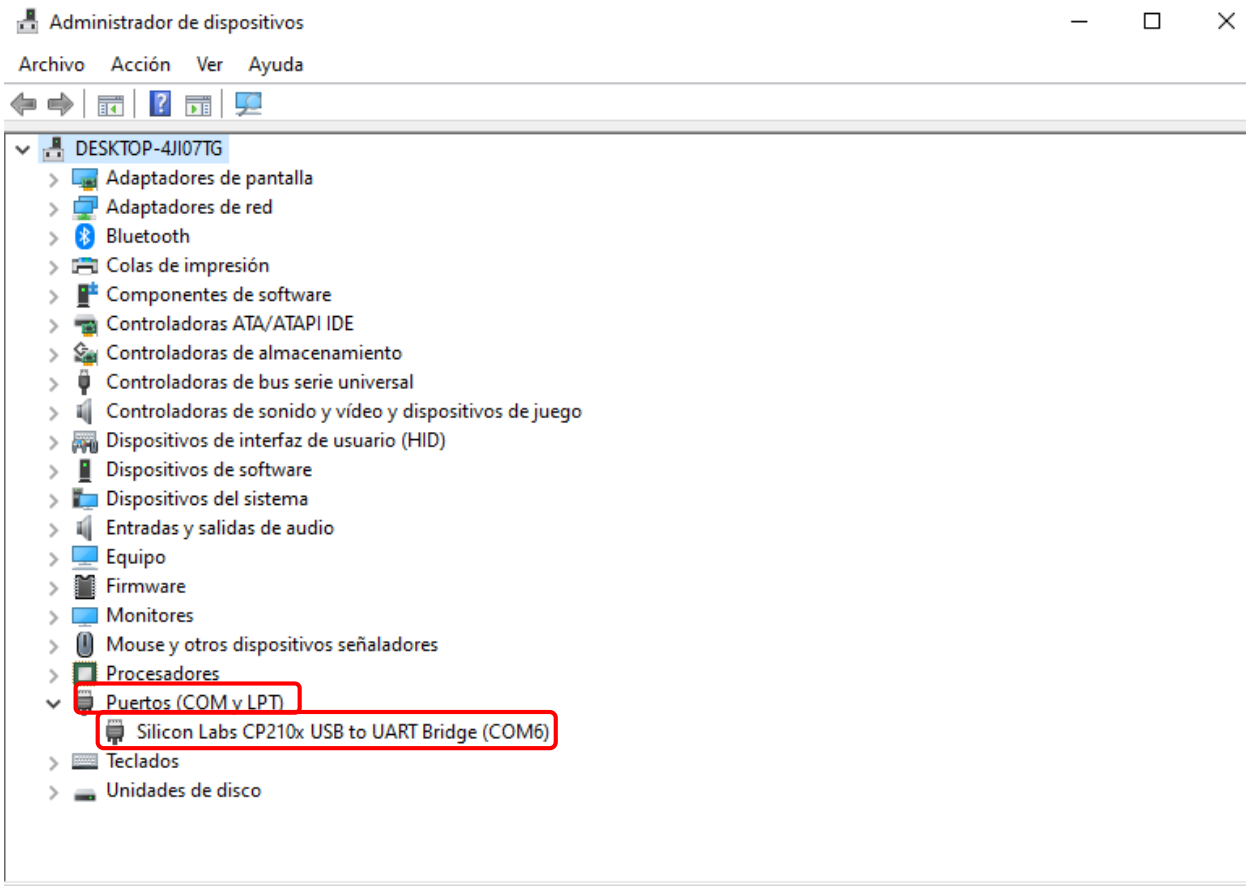


Una vez instalado aparecerá la siguiente ventana:

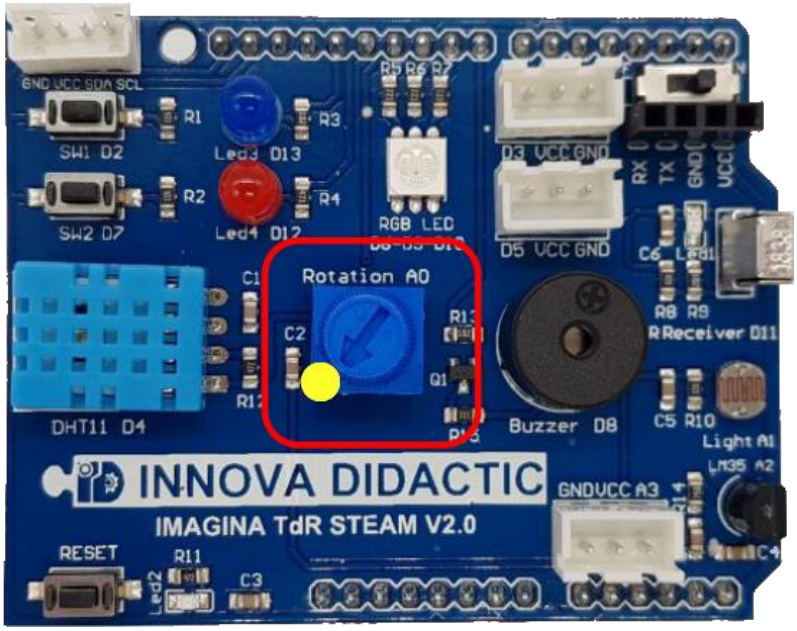


En ArduinoBlocks nos aparecerá el puerto COM que le ha asignado el sistema:

Si abrimos el *Controlador de Dispositivos* de Windows podemos ver el puerto asignado.

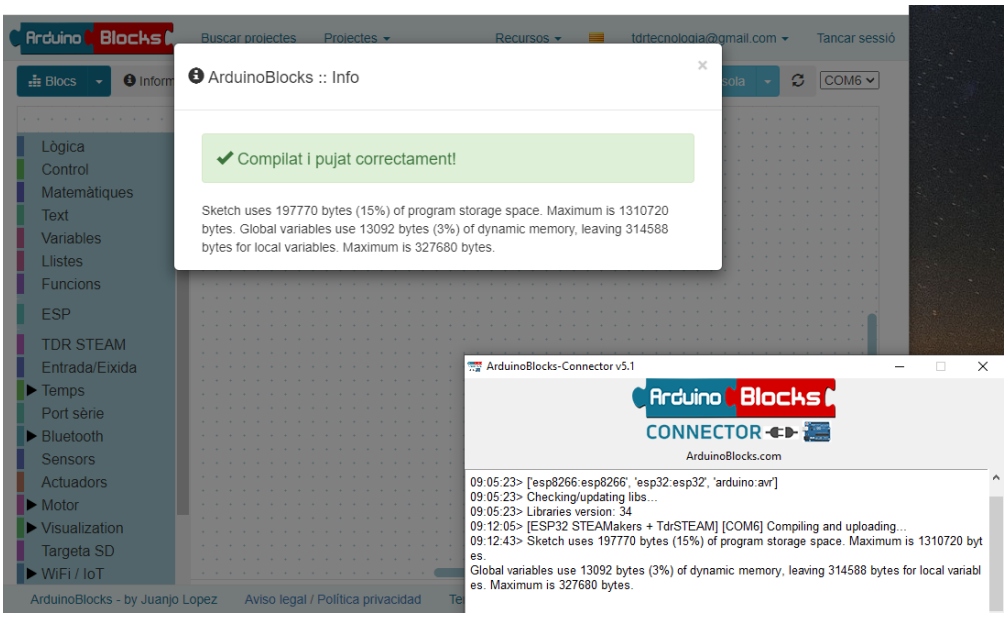


Para probar que el puerto de comunicaciones funciona correctamente, probaremos de enviar un programa vacío. Es **muy importante** que el **potenciómetro esté en la posición izquierda** (punto de color amarillo), ya que comparte una señal con el sistema de **grabación del programa**.



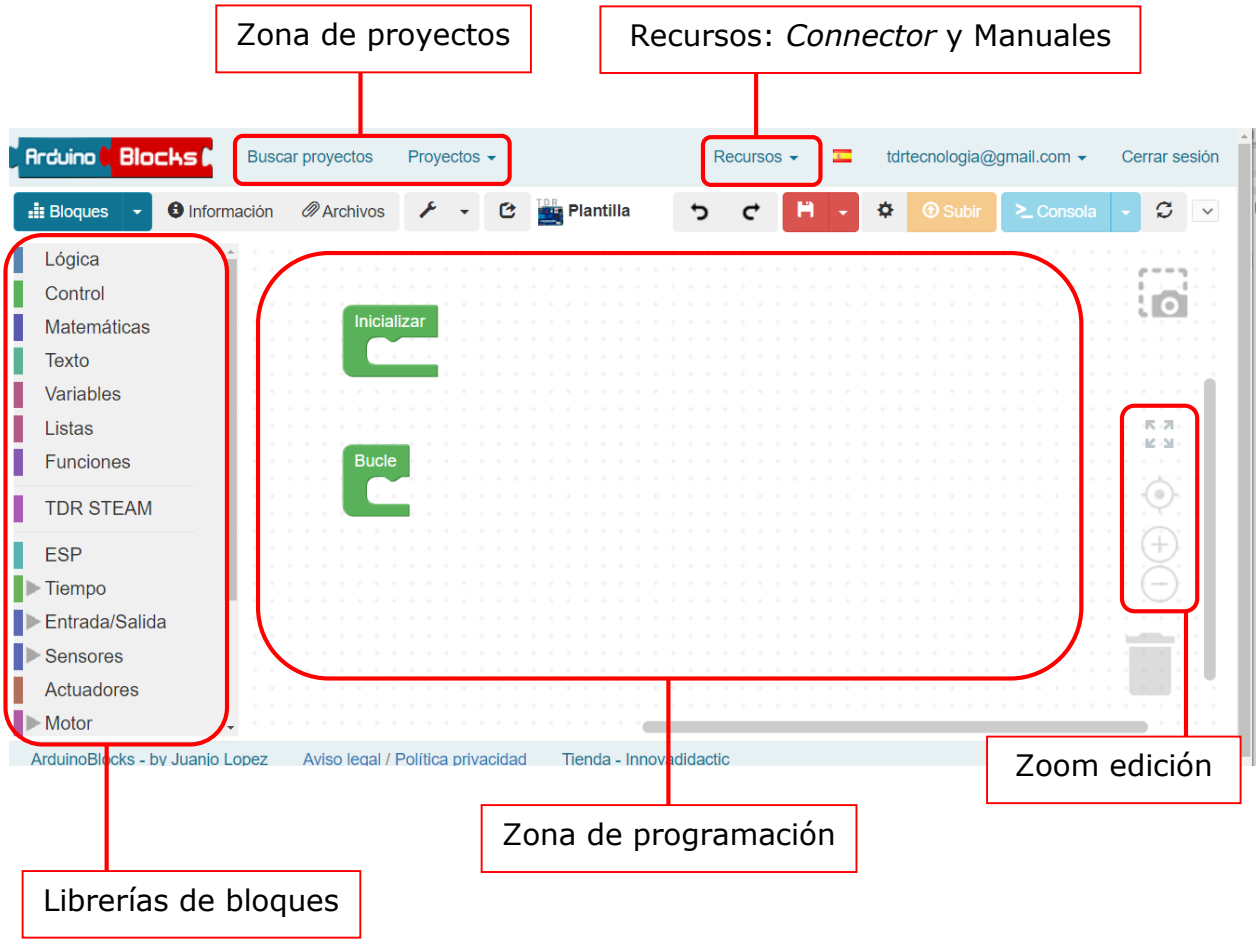
Si no colocamos el potenciómetro en esta posición, cuando enviemos el programa, encenderá el led RGB de color blanco y mostrará un mensaje de error.

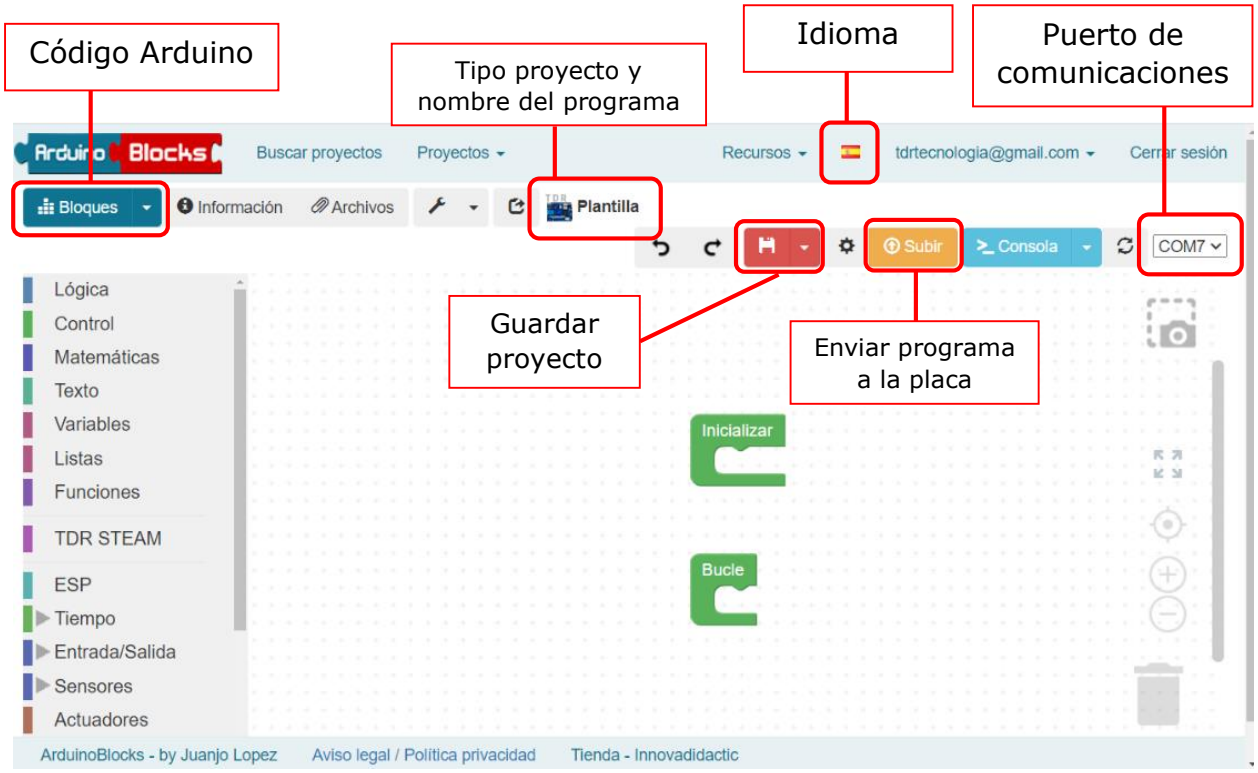
Una vez enviado el programa (tarda unos segundos, más que para una placa Arduino normal) aparece esta ventana emergente:

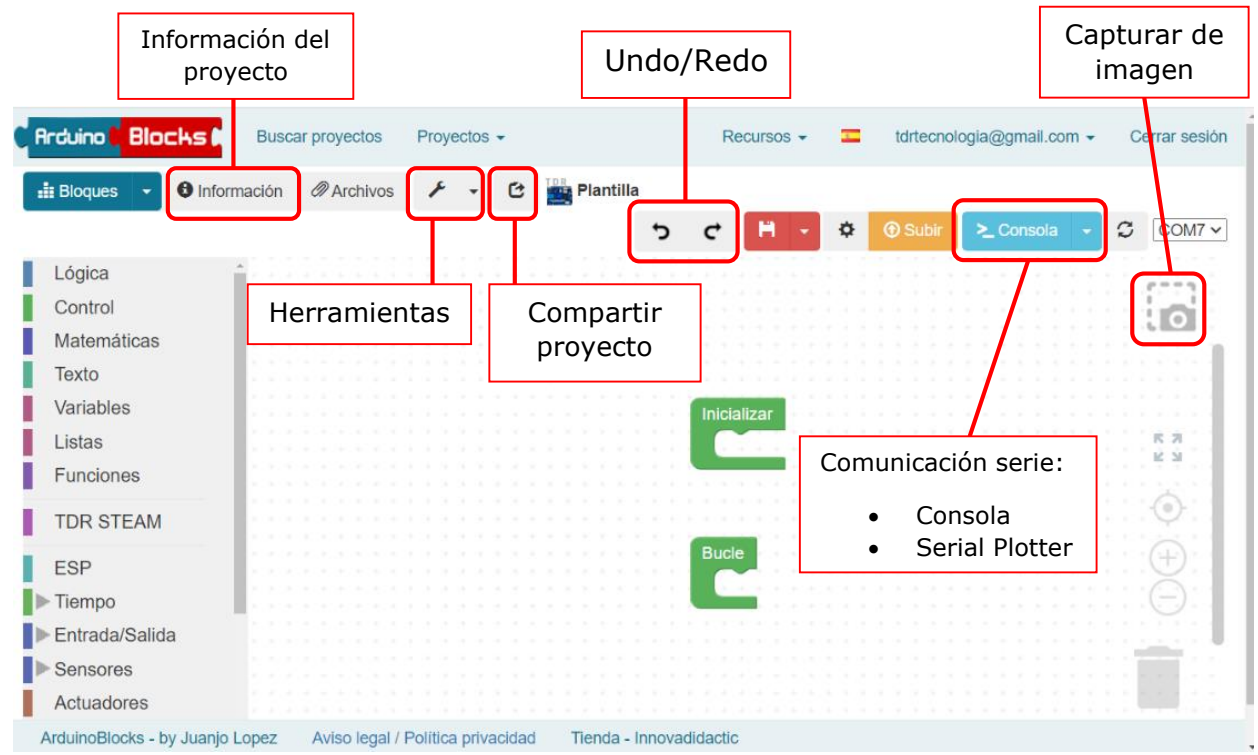


6.1 Características básicas de ArduinoBlocks.

Vamos a ver las características básicas del entorno de programación. Para poder enseñar de forma visual las diferentes zonas del entorno de programación, utilizaremos diferentes imágenes para indicar dónde están los diferentes elementos.

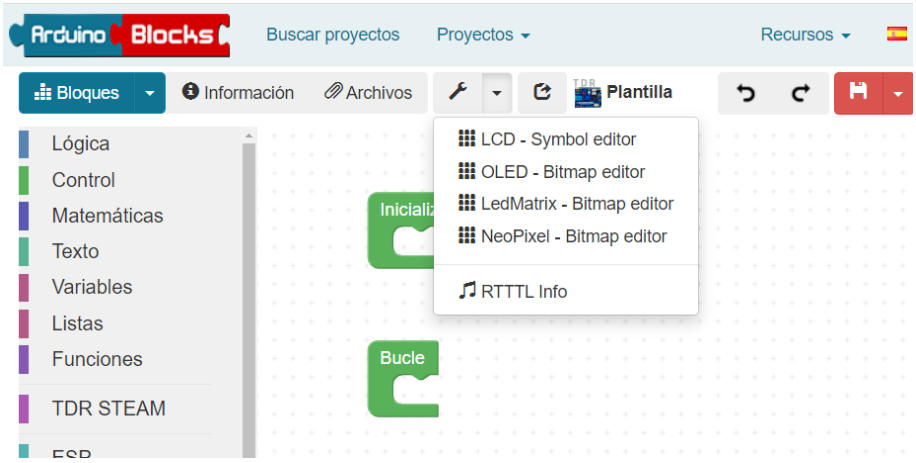



Código Arduino
Tipo proyecto y nombre del programa
Idioma
Puerto de comunicaciones
Guardar proyecto
Enviar programa a la placa
Buscar proyectos
Proyectos
Recursos
tdrtecnologia@gmail.com
Cerrar sesión
Bloques
Información
Archivos
Plantilla
Subir
Consola
COM7
Lógica
Control
Matemáticas
Texto
Variables
Listas
Funciones
TDR STEAM
ESP
Tiempo
Entrada/Salida
Sensores
Actuadores
Inicializar
Bucle
ArduinoBlocks - by Juanjo Lopez
Aviso legal / Política privacidad
Tienda - Innovadidactic

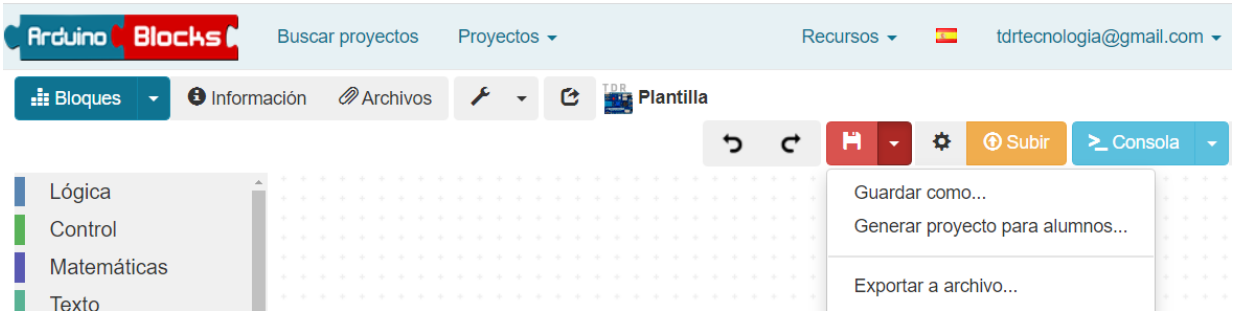

Información del proyecto
Undo/Redo
Capturar de imagen
Herramientas
Compartir proyecto
Comunicación serie:
Consola
Serial Plotter
Buscar proyectos
Proyectos
Recursos
tdrtecnologia@gmail.com
Cerrar sesión
Bloques
Información
Archivos
Plantilla
Subir
Consola
COM7
Lógica
Control
Matemáticas
Texto
Variables
Listas
Funciones
TDR STEAM
ESP
Tiempo
Entrada/Salida
Sensores
Actuadores
Inicializar
Bucle
ArduinoBlocks - by Juanjo Lopez
Aviso legal / Política privacidad
Tienda - Innovadidactic

Código Arduino:

Herramientas:

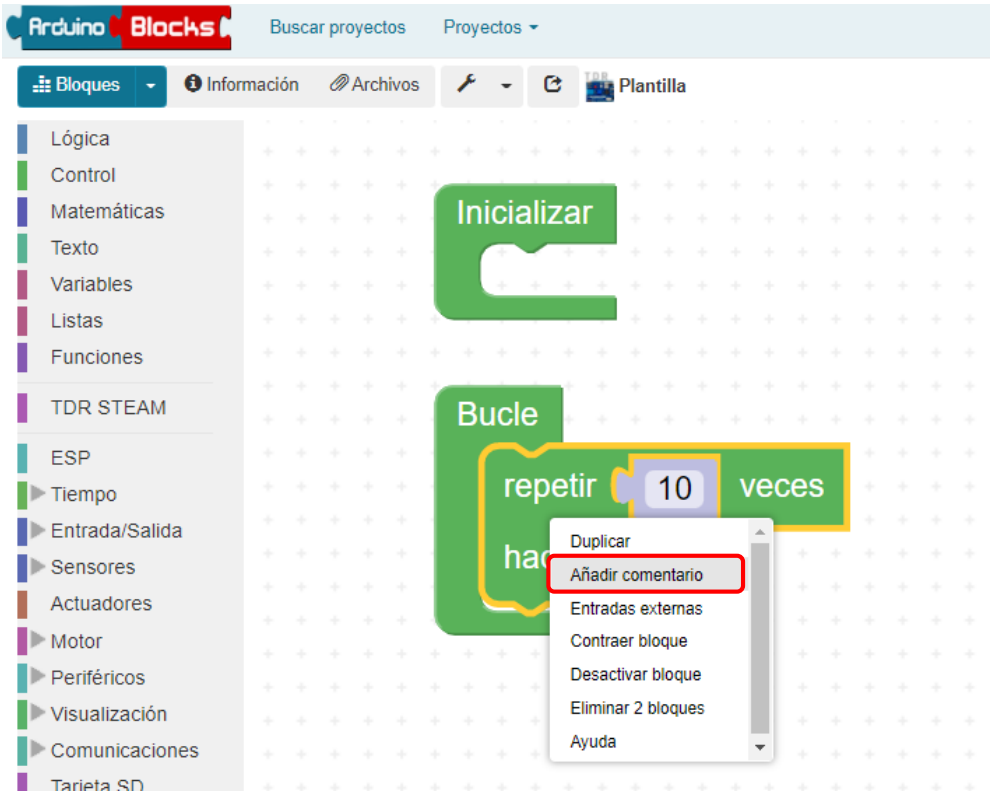


Guardar proyecto:

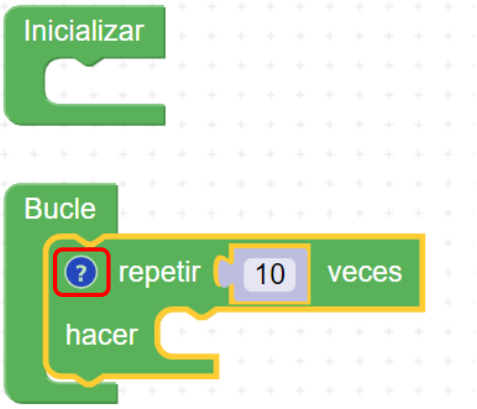


ArduinoBlocks permite añadir comentarios en todos los bloques. Estos comentarios aparecen en forma de nube junto al bloque que se puede mover y colocar donde deseemos.

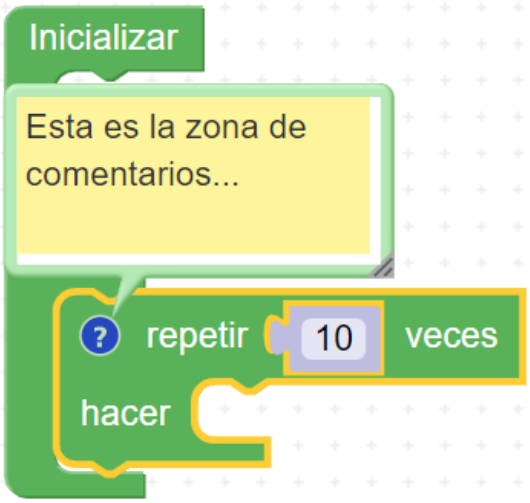
Apretando con el botón derecho del ratón en el bloque, aparece un menú. Seleccionaremos la opción *Añadir comentario*.



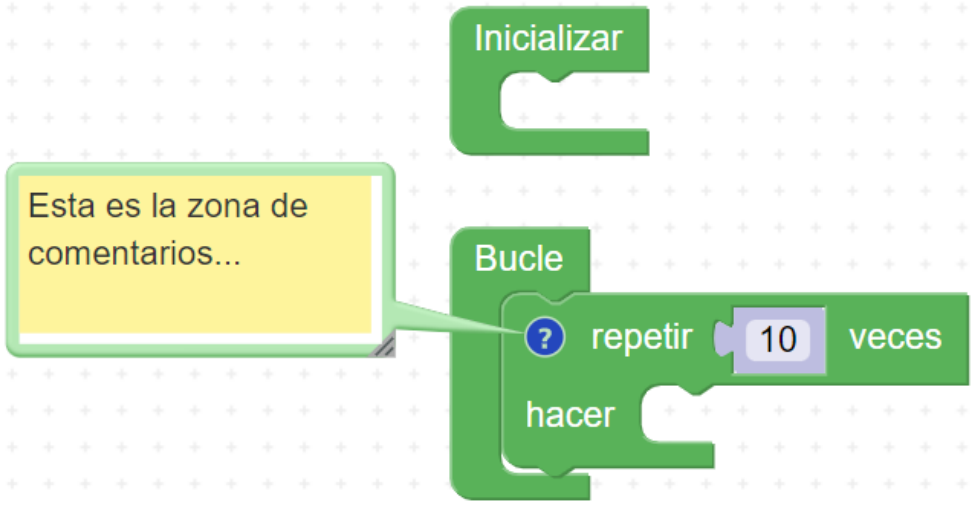
Aparecerá entonces un signo de interrogación en la esquina superior izquierda.



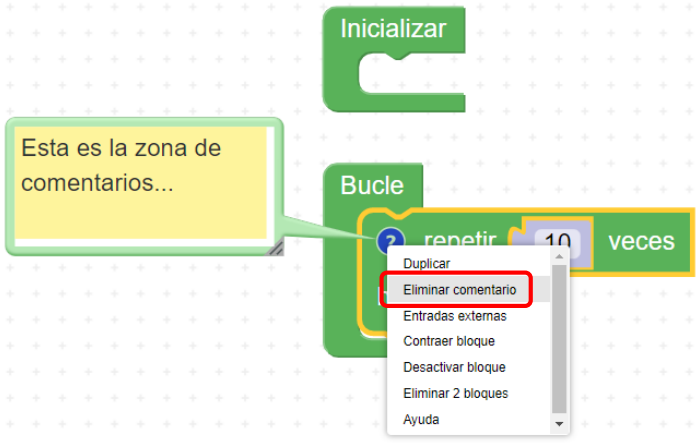
Pulsando sobre el interrogante aparece una zona para poner el texto de información.



Podemos mover el cuadro de comentarios a la zona que deseemos.



Los comentarios también se pueden eliminar.



7 Actividades con ESP32 Plus STEAMakers e Imagina TDR STEAM.

A continuación, os proponemos una serie de actividades y retos para aprender a programar la placa ***ESP32 Plus STEAMakers*** y la placa ***Imagina TDR STEAM*** con ArduinoBlocks paso a paso.

- A01. El led.
- A02. El led RGB.
- A03. El zumbador.
- A04. El pulsador.
- A05. El potenciómetro.
- A06. La fotocélula (LDR o sensor de luz).
- A07. Sensor de temperatura LM35D.
- A08. Sensor de temperatura y humedad DHT-11.
- A09. Sensor de infrarrojos (IR).
- A10. El micrófono.
- A11. El servomotor.
- A12. Puertos de expansión I2C.
    - Pantalla LCD 16x2.
    - Pantalla OLED.
    - Matriz 8x8.
    - Sensor de color.
    - Sensor de reconocimiento de gestos.
    - Sensor RFID.
    - Controlador de servos PCA9685.
    - Sensor de CO2 equivalente CCS811.
    - Sensor acelerómetro de tres ejes ADXL345.
    - Sensor de presión barométrica BMP280.
    - Sensor giroscopio y acelerómetro MPU6050.
- A13. Tarjeta SD.
- A14. Serial Plotter
- A15. Consumo de energía.
- A16. Sensores internos.
- A17. Multitarea.
- A18. Interrupciones.
- A19. Memoria FLASH/EEPROM.
- A20. Sistemas de comunicaciones: Bluetooth y Wifi.
- A21. Comunicación Bluetooth.
- A22. Comunicación Wifi.
    - Conexión a red Wifi.
    - Servidor HTTP.

- MQTT.
- Wifi-Mesh.

También hay un apartado (8) dedicado a trabajar únicamente con la placa ***ESP32 Plus STEAMakers*** y, en el apartado 9, veremos resueltos una serie de proyectos de ejemplo resueltos.

En las actividades y retos, primero hay una explicación del componente electrónico si no se ha utilizado antes, de los elementos de programación o de algún programa adicional que se necesite. A continuación, una descripción de la actividad a realizar (puede ser más de una práctica/reto). Por último, se propone una actividad de ampliación en algunas ocasiones.

También se adjunta el enlace a algunos de los ejercicios realizados con ArduinoBlocks que se pueden importar directamente desde el entorno de programación.

7.1 Reto A01. El led.

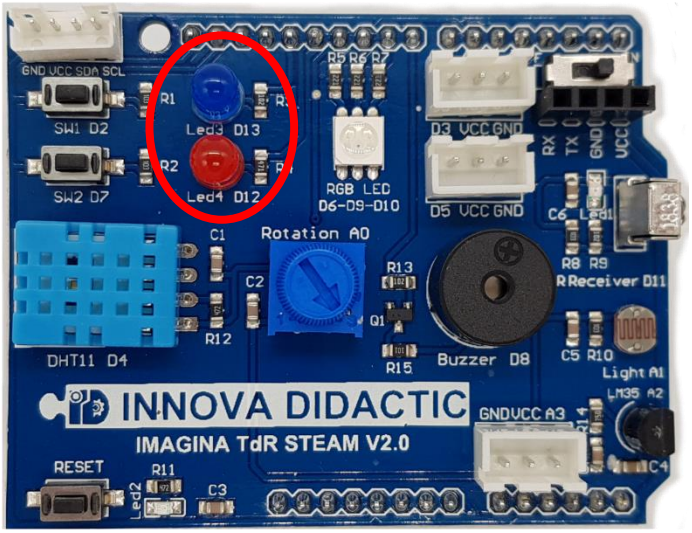
Reto A01. El led.

Vamos a empezar con nuestro primer reto. Vamos a realizar un programa que va a encender y apagar el led rojo correspondiente al pin D12.

Un LED (Diodo Emisor de Luz) es un diodo semiconductor que emite luz. Se usan como indicadores en muchos dispositivos, y cada vez con mucha más frecuencia en iluminación. Los leds presentan muchas ventajas sobre las fuentes de luz incandescente como un consumo de energía mucho menor, mayor tiempo de vida, menor tamaño, gran durabilidad y fiabilidad.

El led tiene una polaridad, un orden de conexión, y al conectarlo al revés no funciona. Para evitar que se queme siempre debe llevar una resistencia en serie para limitar la corriente eléctrica que le atraviesa.

La placa Imagina TDR STEAM dispone de dos leds (uno azul y otro rojo) conectados en los pines D13 (azul) y D12 (rojo).



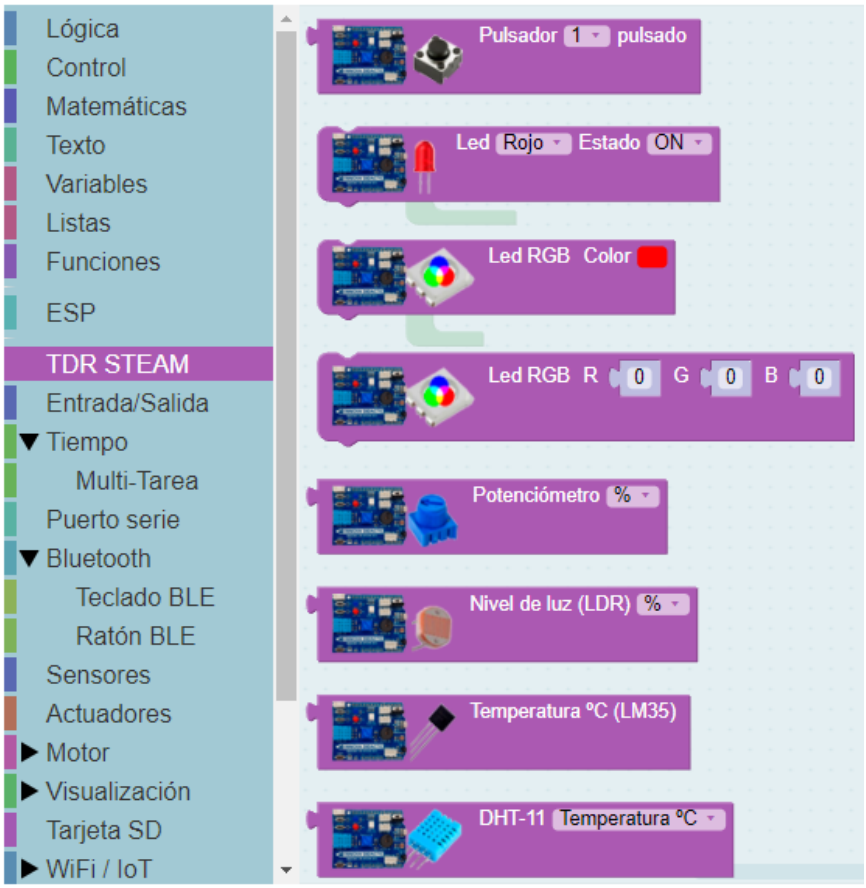
7.1.1 Reto A01.1. ON/OFF led rojo.

Reto A01.1. ON/OFF led rojo.

Entramos en ArduinoBlocks en el tipo de proyecto para la placa ***Imagina TDR STEAM***. En la columna de la izquierda tenemos las agrupaciones de bloques clasificados en función de su tipología.

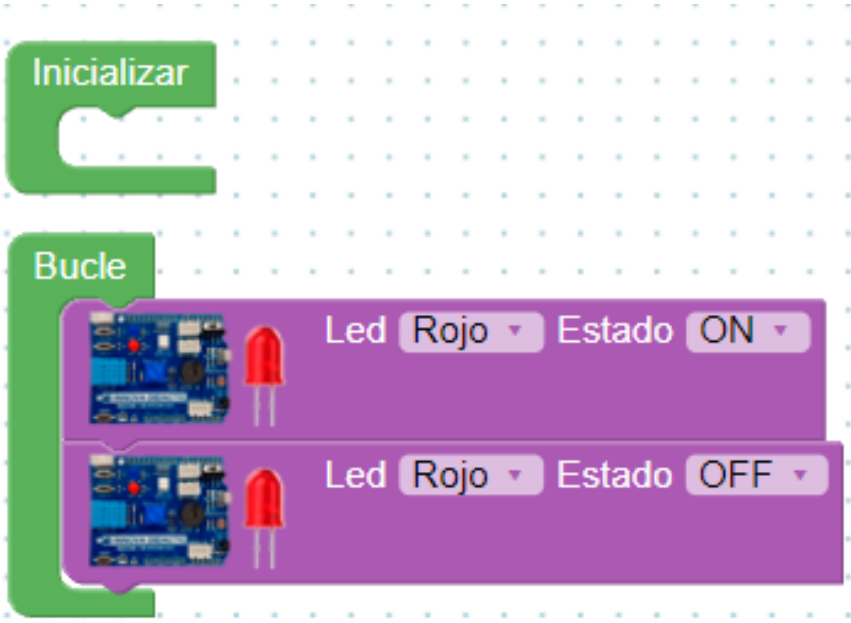
En el área de programación siempre hay dos bloques verdes (*Inicializar* y *Bucle*). Estos bloques siempre aparecen al iniciar un nuevo programa. Pues bien, todo lo que se meta dentro del bloque de *Inicializar* sólo se ejecutará la primera vez que se inicie el programa, mientras que si se colocan dentro del Bucle se ejecutarán una y otra vez hasta que apaguemos la placa.

Utilizaremos los bloques propios de la placa ***Imagina TDR STEAM***.



Seleccionaremos el bloque de *Actuadores*. Vamos a introducir nuestro bloque de led en el Bucle y dejamos el color Rojo. El led puede tener dos estados: ON/OFF, que podemos cambiar en el menú despegable.

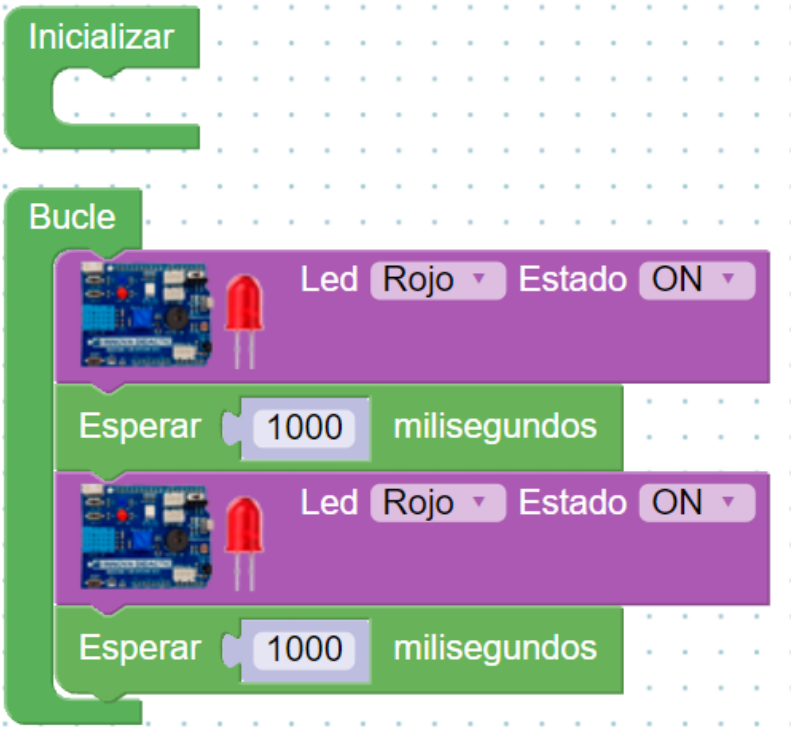
Si sólo dejamos este bloque con el led en estado ON, este quedaría encendido para siempre, para que se apague deberemos cambiar el estado a OFF.



Pero este programa no es correcto del todo. No hay tiempos que indiquen cuanto tiempo tiene que estar encendido o apagado el led. Necesitamos ir al bloque de *Tiempo* y seleccionar *Esperar (valor) milisegundos* (recuerda: 1.000 milisegundos es 1 segundo).

Agregaremos el bloque de espera de tiempo:

Introduciremos una espera de 1000 milisegundos (1 segundo) entre cada estado del led.



Ahora tenemos el led encendido durante 1 segundo y apagado otro. Esto ser repetirá por tiempo infinito hasta que quitemos la alimentación a la placa.

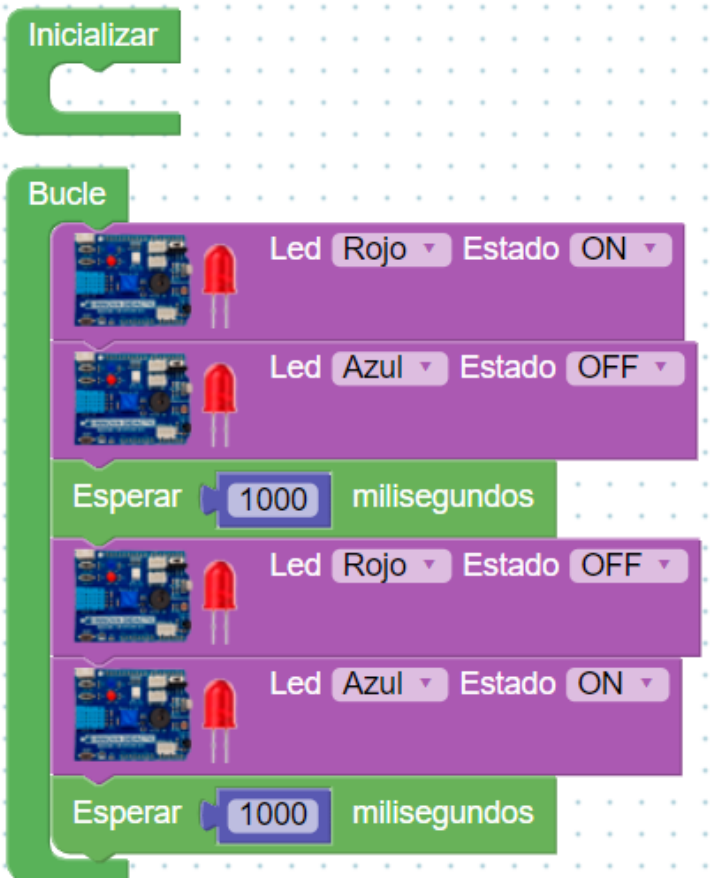
El programa se quedará guardado en la memoria del microcontrolador de forma permanente, así que, si lo alimentamos con una fuente de alimentación externa, seguirá funcionando.

Actividad de ampliación: prueba ahora de hacer una intermitencia más rápida (500ms ON y 250ms OFF).

7.1.2 Reto A01.2. ON/OFF led rojo y azul.

Reto A01.2. ON/OFF led rojo y azul.

Como hemos comentado anteriormente la placa dispone de dos leds (rojo y azul). Ahora vamos a realizar un programa para que se vayan alternando su encendido y apagado.

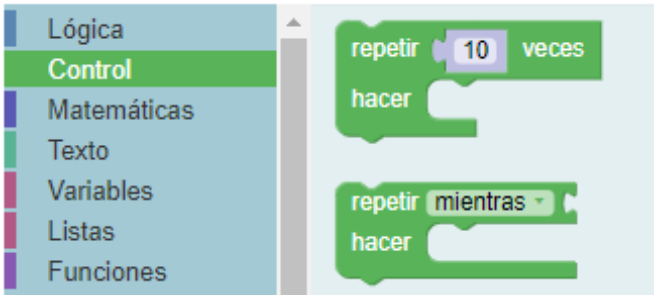


Actividad de ampliación: prueba ahora de hacer que los leds rojo y azul se enciendan a la vez y con los siguientes tiempos: 500ms ON y 250ms OFF.

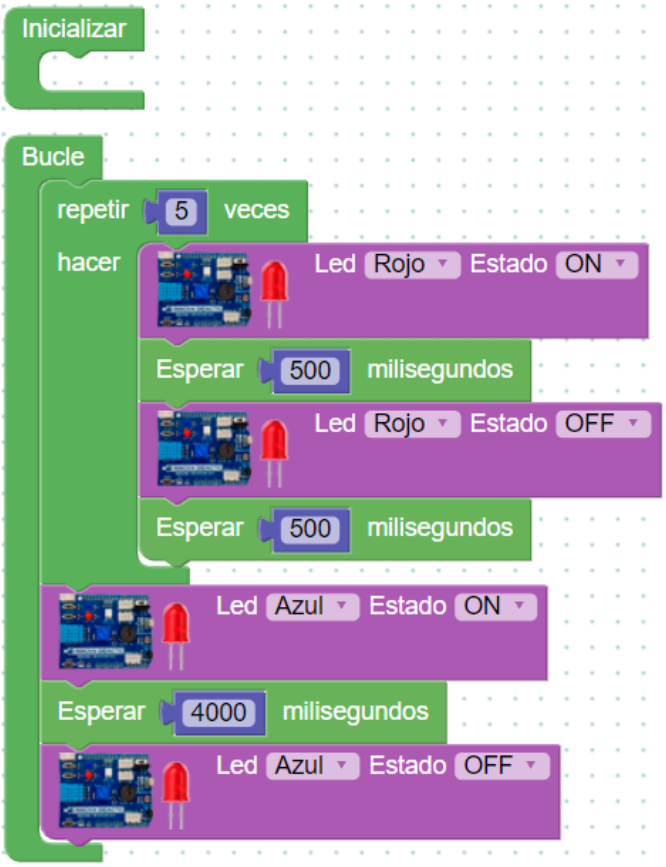
7.1.3 Reto A01.3. ON/OFF led rojo y azul con repeticiones.

Reto A01.3. ON/OFF led rojo y azul con repeticiones.

Imagina que queremos hacer un ciclo de repetición. Queremos repetir 5 veces el encendido y apagado del led rojo antes de que se encienda el azul. Para realizar esta acción lo podemos hacer de la siguiente forma: en el menú de *Control* existe el bloque *Repetir (valor) veces hacer…*



En el siguiente programa fíjate como el led rojo se enciende y apaga cada medio segundo (500ms) 5 veces y después se queda el led azul encendido durante 4 segundos (4000ms).



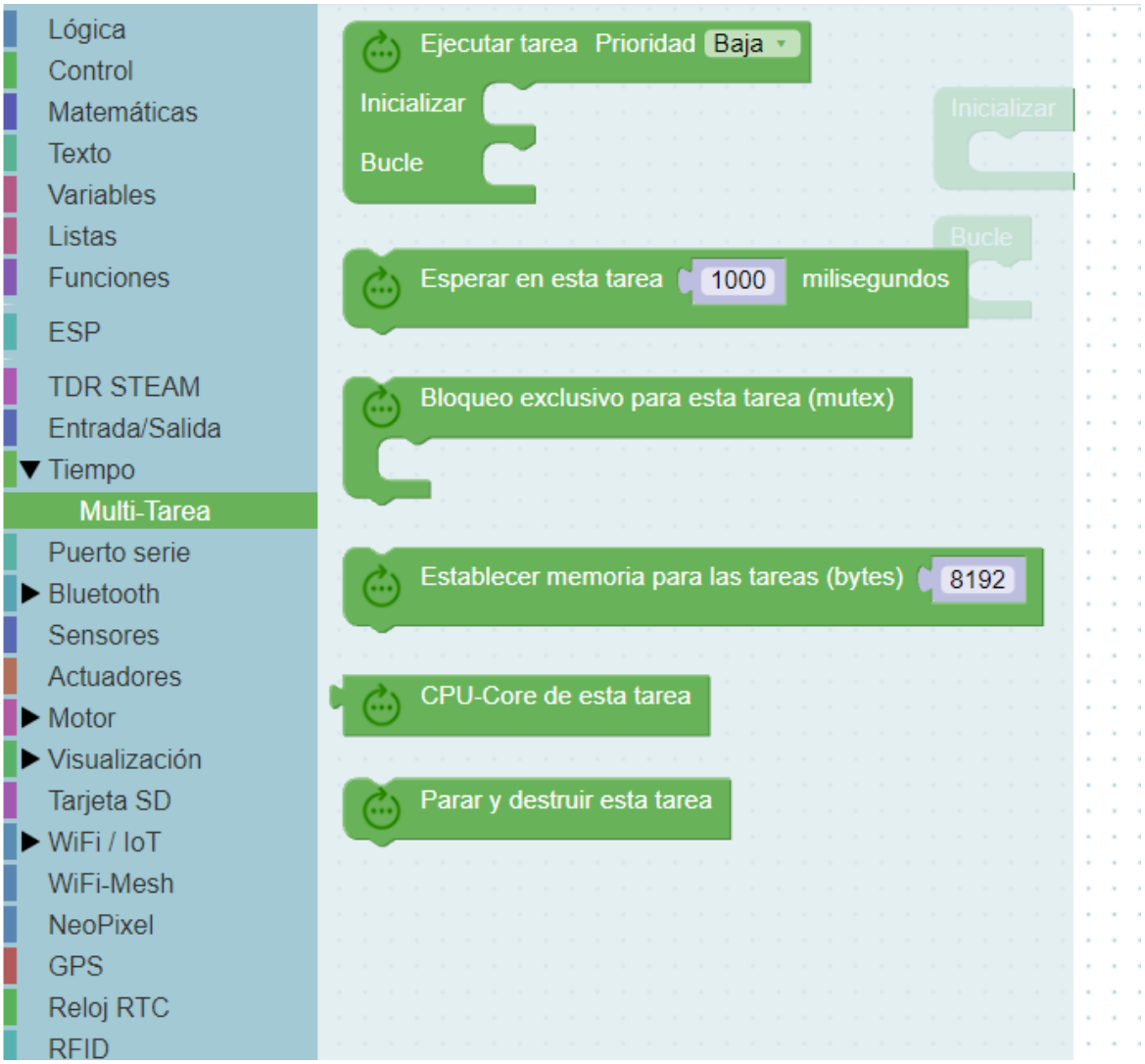
Actividad de ampliación: prueba ahora de hacer que el led rojo se encienda 10 veces cada vez que el led azul se enciende 3 veces (tiempo de 500ms).

7.1.4 Reto A01.4. Multitarea: parpadeos independientes.

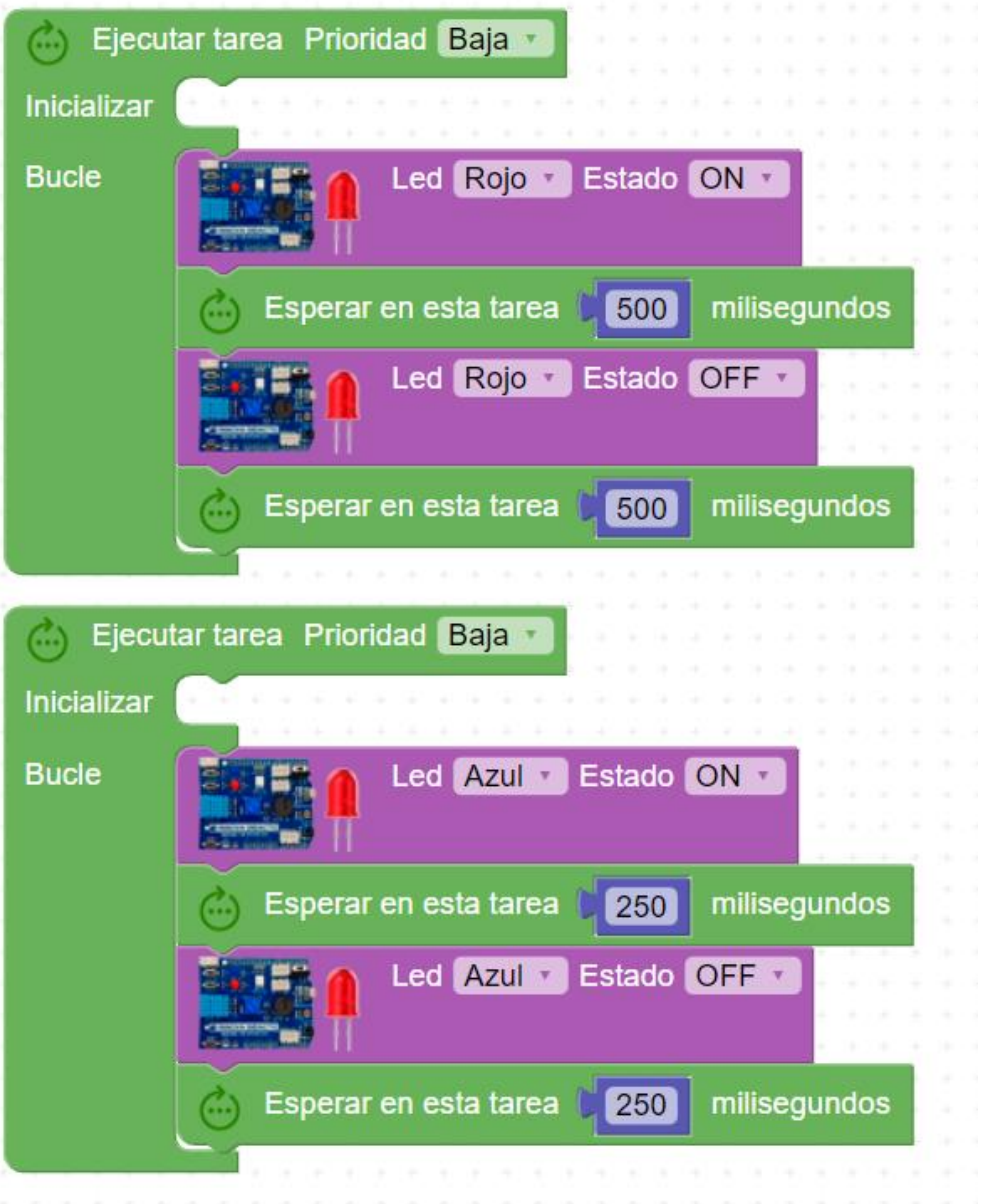
Reto A01.4. Multitarea: parpadeos independientes.

Una ventaja que posee el ***ESP32 Plus STEAMakers*** es que, al estar basado en un ESP32 con dos microcontroladores internos, podemos hacer que cada microcontrolador trabaje en una tarea o varias tareas. Esto también se puede hacer en las placas Arduino UNO, pero con el ESP32 tenemos más potencia para realizar estas tareas. Esto lo desarrollaremos en capítulos posteriores.

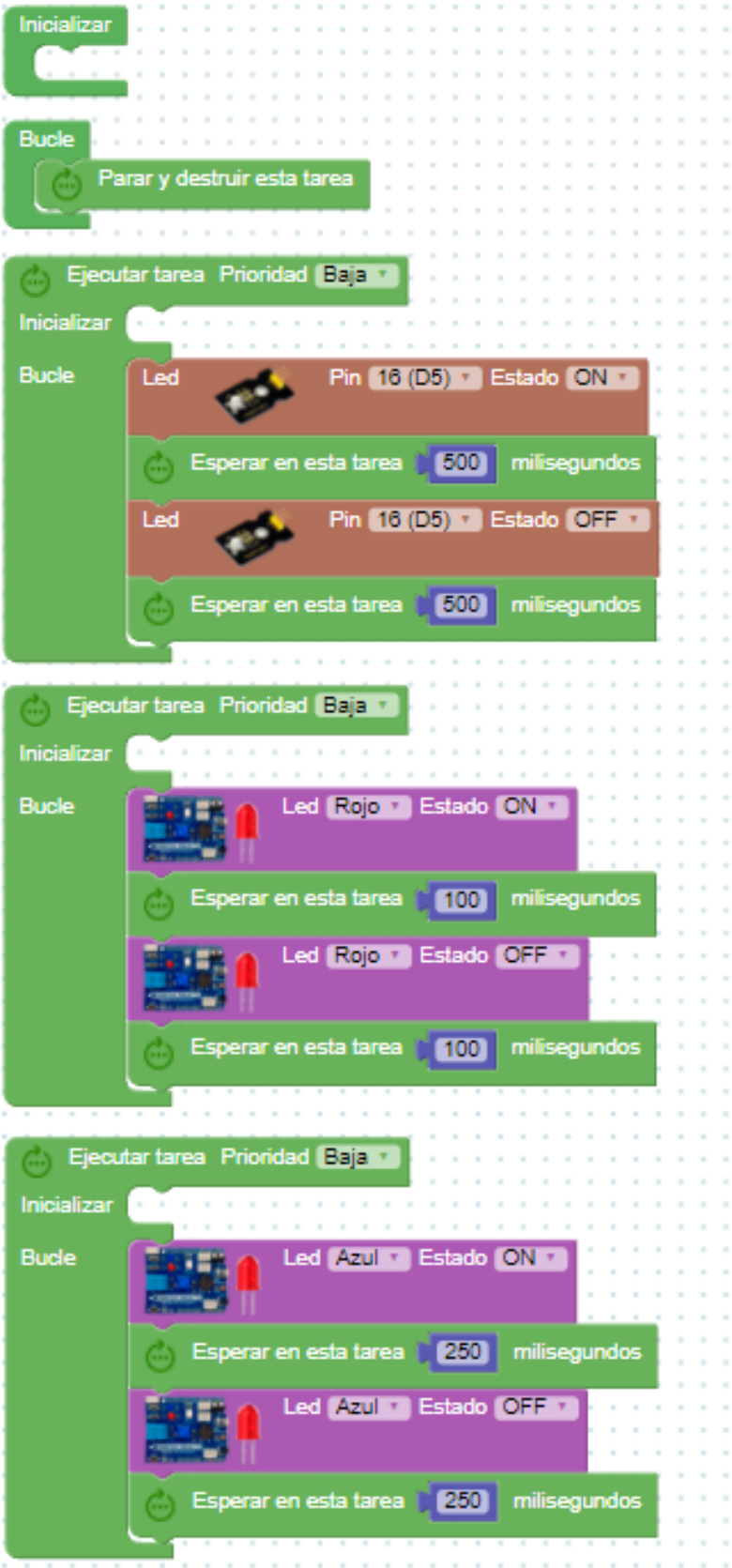
Ahora, vamos a crear un programa que haga que los dos leds parpadeen de forma independiente. En el grupo de funciones de Tiempo están las funciones para trabajar con el modo multitarea.



Diseñaremos un programa muy sencillo, sin entrar en detalles de este modo de funcionamiento, que consistirá en hacer dos parpadeos con dos leds a velocidades diferentes:



Incluso podemos realizar más tareas simultáneas. Aquí os mostramos como parpadean los dos leds de la placa y otro led añadido a la salida D5, los tres con intermitencias diferentes:



Enlace al programa: ArduinoBlocks Projects\multitarea_3leds.abp

7.2 Reto A02. El led RGB.

Reto A02. El led RGB.

Un **led RGB** es un led que incorpora en su mismo encapsulado tres leds. Las siglas RGB corresponden a: R (Red=rojo), G (Green=verde) y B (Blue=azul). Con estos tres colores, en óptica, se puede formar cualquier color, ajustando de manera individual la intensidad de cada color. Los tres leds están unidos por el negativo o cátodo (RGB de cátodo común).

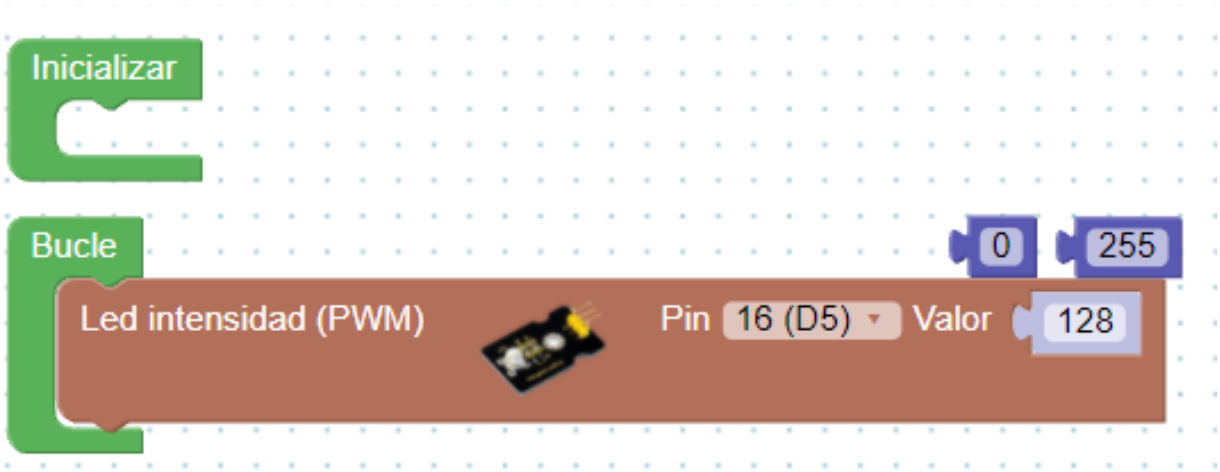
Símbolo

Componente

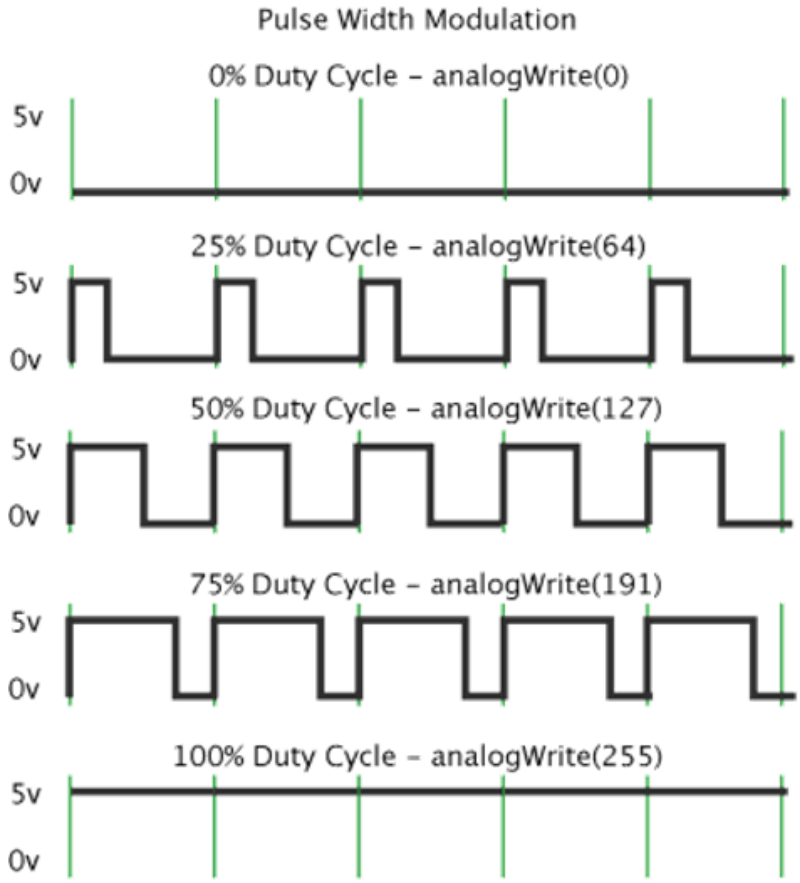
En Arduino, cada uno de esos leds podría tomar 256 colores diferentes, es decir, el Rojo podría ir desde 0 hasta 255, el Verde de 0 a 255 y el Azul de 0 a 255, en total un led RGB podría dar más de 16,5 millones de colores diferentes.

La placa ***Imagina TDR STEAM*** dispone de un led RGB conectado a los pines (D6-Red, D9-Green y D10-Blue). Estos tres pines son PWM para permitir regular su intensidad.

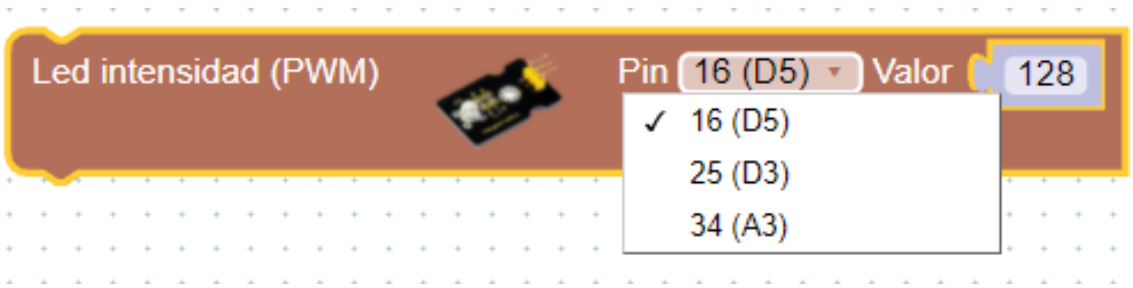
La modulación PWM permite generar una señal analógica mediante una salida digital. Utiliza un sistema de codificación de 8 bits (en el sistema binario: 2^8=256, per tanto, del 0 al 255).



La modulación PWM es la abreviatura de ***Pulse-Width Modulation*** (modulación de ancho de pulso). Algunas de las salidas digitales sólo tienen dos estados: **ALTO/BAJO**, **ON/OFF**, **ENCENDIDO/APAGADO**. Es decir, corresponden a una salida de 5 V (ON) y de 0 V (OFF). Con esto sólo podemos hacer actividades de encender y apagar un led, no podríamos controlar su brillo de menos a más o viceversa. Para poder simular una señal analógica lo realiza por la proporción entre el estado alto (ON) y bajo (OFF) de la señal. El control digital se utiliza para crear una onda cuadrada de ciclo de trabajo diferente, una señal conmutada entre encendido y apagado. Este patrón de encendido y apagado puede simular voltajes entre encendido total (5 voltios) y apagado (0 voltios) al cambiar la parte del tiempo que la señal pasa en comparación con el tiempo que la señal pasa. La modulación de ancho de pulso, o PWM, es una técnica para obtener resultados analógicos con medios digitales. Se utiliza mucho para controlar leds, velocidades de motores, producción de sonidos, etc.



La placa ***ESP32 Pus STEAMakers*** tiene muchas salidas PWM, pero en la placa ***Imagina TDR STEAM*** sólo se puede controlar por PWM el led RGB (pines 6, 9 y 10) y los tres pines que quedan libres para conectar elementos externos.

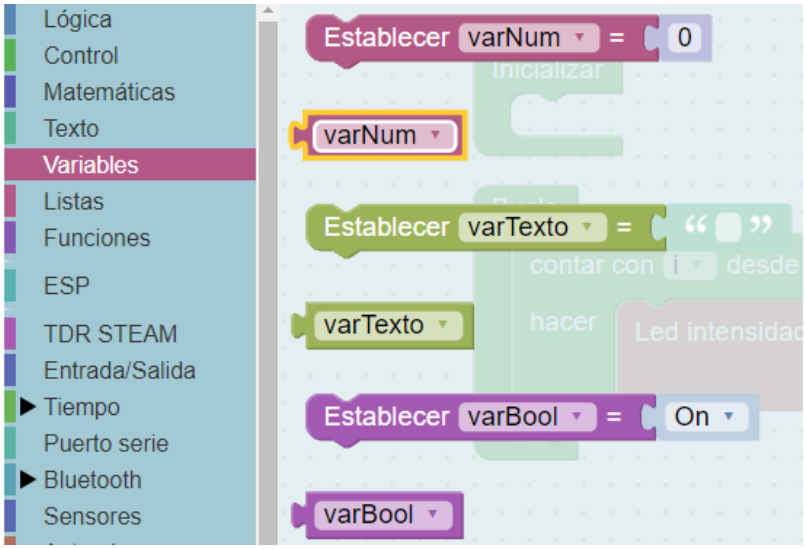


Podemos conectar un led externo a cualquiera de las tres salidas libres y comprobar el funcionamiento de un programa para ver cómo se produce la regulación.

Para hacer el programa vamos a utilizar un bloque llamado contador, que nos permite hacer un número determinado de veces una acción y sabemos en todo momento el valor del contador. La variable que utiliza por defecto es ***i***, pero se puede cambiar por cualquier otra. A grandes rasgos, una variable es

un hueco en una estantería en la cual se van introduciendo diferentes valores que el programador utilizará según los necesite. Cada hueco en la estantería es una variable. Dentro de cada hueco podemos colocar diferentes cosas, es decir, diferentes tipos de variables.

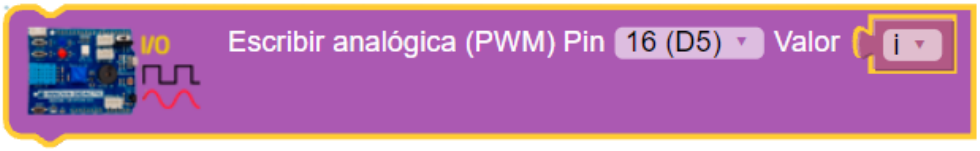
En el programa inicialmente el valor de la *Variable i* vale 1. Cada vez que se repite el bucle su valor va aumentado de 1 en 1. Para introducir el Valor del PWM deberemos ir a *Variables* y seleccionar el tipo *varNum*.



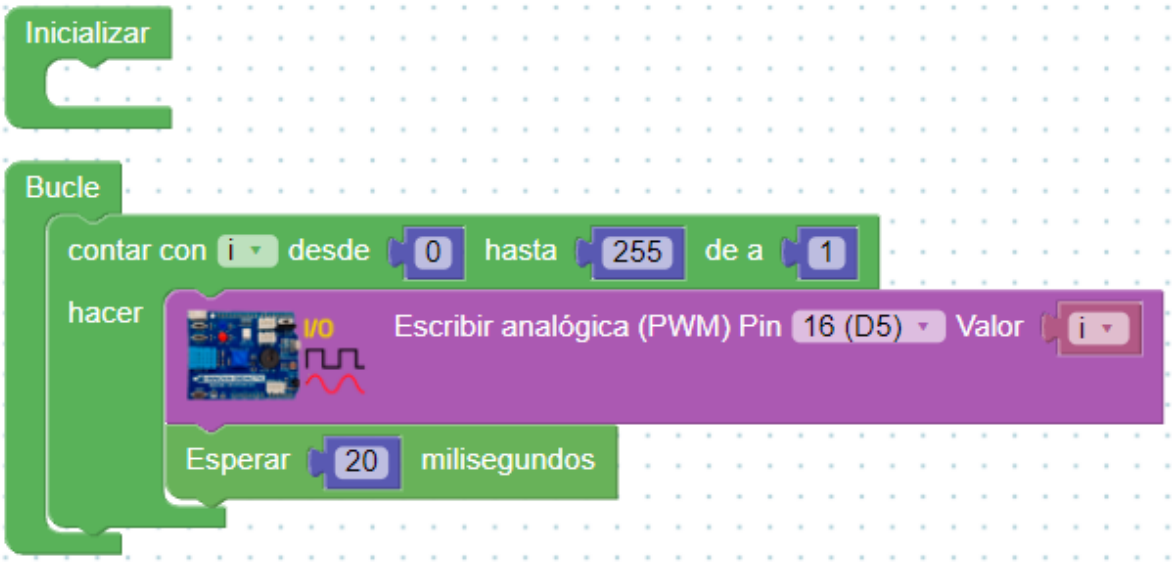
En el menú desplegable de *varNum* seleccionar *i*.

Y el programa quedará de la siguiente forma:

También lo podemos hacer con el bloque específico para la ***Imagina TDR STEAM***.



El programa quedaría de la siguiente forma:

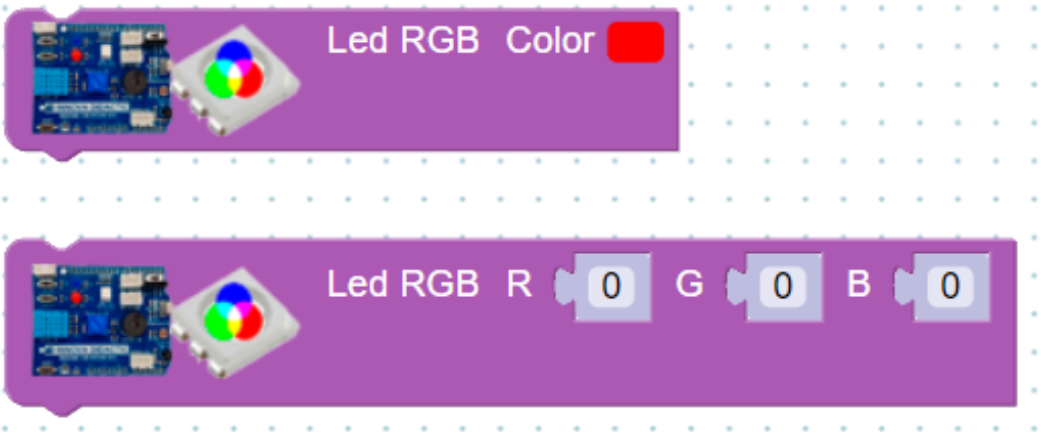


Actividad de ampliación: intenta completar el programa haciendo que el brillo del LED ascienda de 0 a 255 y luego descienda de 255 a 0 (sólo tendrás que intercambiar los valores en cada ciclo).

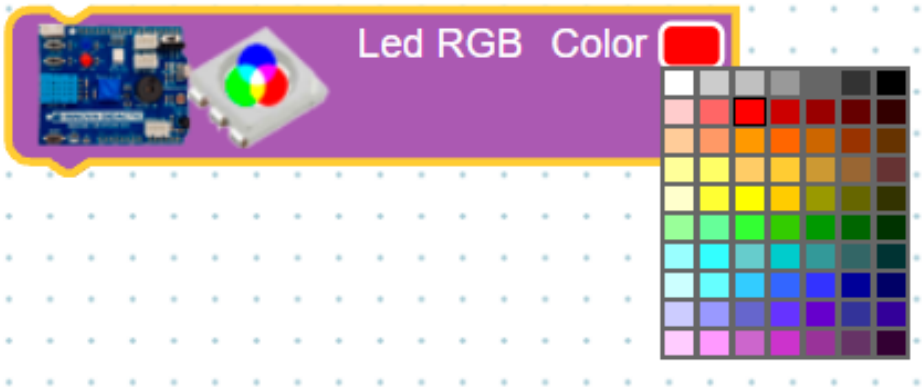
7.2.1 Reto A02.1. Creación de colores RGB.

Reto A02.1. Creación de colores RBG.

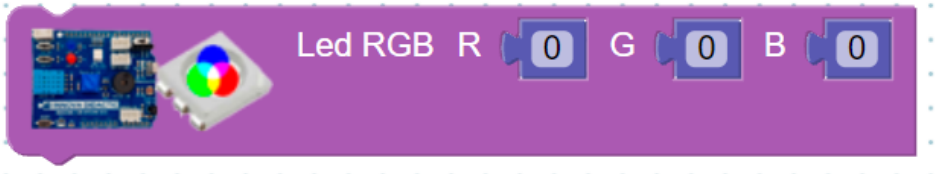
Con este sencillo programa vamos a crear colores con el led RGB. ArduinoBlocks dispone de dos bloques específicos para poder trabajar con el led RGB.



En el primer bloque podemos elegir un color de la paleta de colores y encenderá el led de ese color.

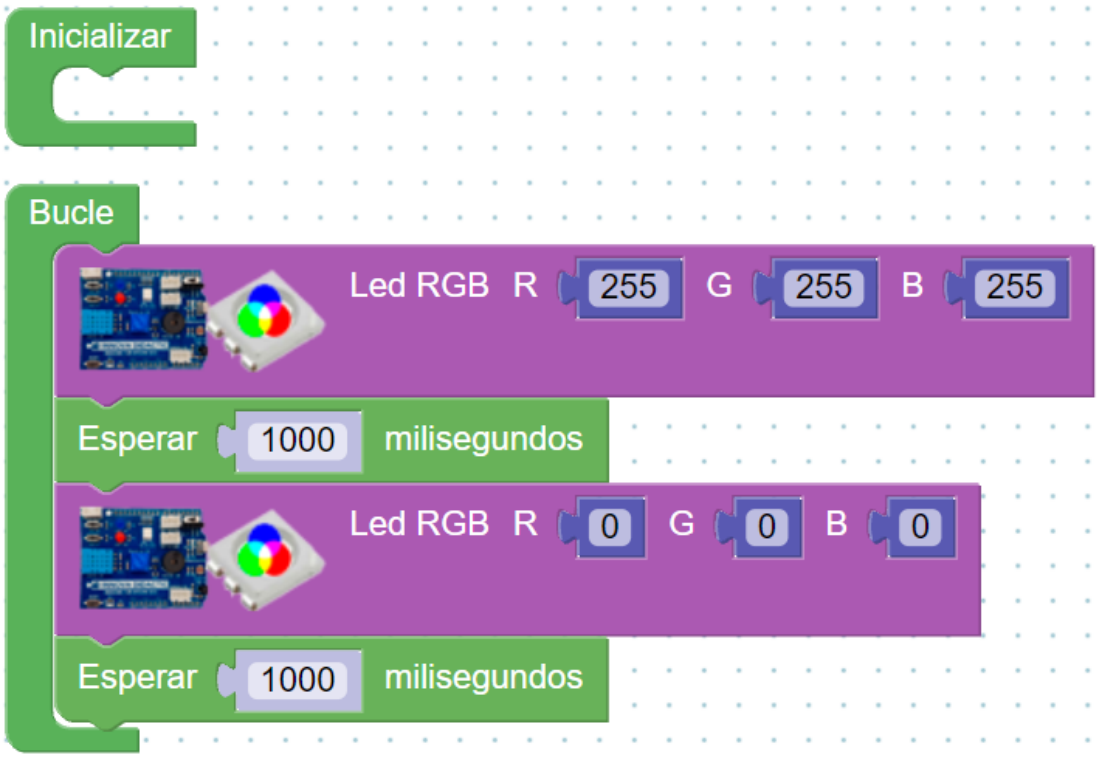


Y en el segundo bloque podemos especificar la cantidad (en forma de número) de cada color. La cantidad numérica de cada color va de 0 a 255.



Si todos los valores están a 0 equivale a que el led RGB esté apagado (negro) y si todos los valores valen 255 se verá de color blanco.

Haremos un programa que haga una intermitencia entre negro (apagado) y blanco (encendido).

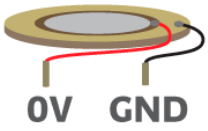


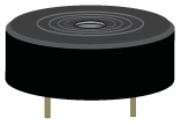
Actividad de ampliación: prueba ahora de hacer la combinación de los colores: R+B – R+G – G+B (puedes guiarte con la imagen siguiente).

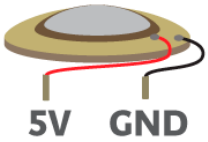
7.3 Reto A03. El zumbador.

Reto A03. El zumbador.

El zumbador (Buzzer en inglés) es un transductor electroacústico que produce un sonido o zumbido continuo o intermitente. En función de si se trata de un zumbador *Activo* o *Pasivo*, este zumbido será del mismo tono o lo podremos variar. Sirve como mecanismo de señalización o aviso y se utiliza en múltiples sistemas, como en automóviles o en electrodomésticos.



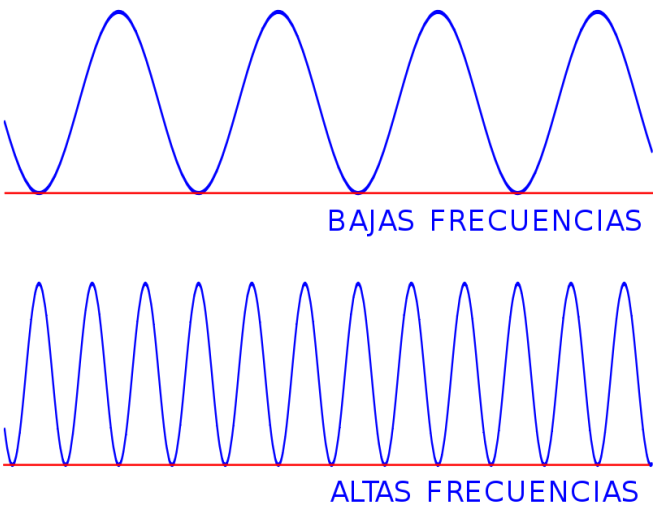




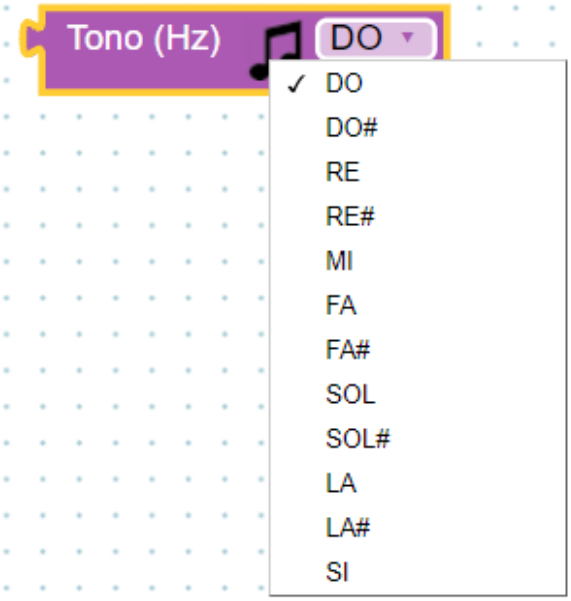
La placa ***Imagina TDR STEAM*** tiene un zumbador pasivo que está conectado en el pin D8.

ArduinoBlocks tiene 4 bloques específicos para programar el zumbador están en el menú de *TDR STEAM*.

El sonido que emite el zumbador depende de la frecuencia de emisión del sonido. La frecuencia es el número de repeticiones por unidad de tiempo de cualquier evento periódico. Sabemos que el sonido se transmite en forma de onda y la frecuencia de un sonido es el número de oscilaciones o variaciones de la presión por segundo, nos indica cuantos ciclos por segundo tiene una onda.



Cada nota musical es una frecuencia. En ArduinBlocks ya está definida una octava entera con sus 12 notas correspondientes.



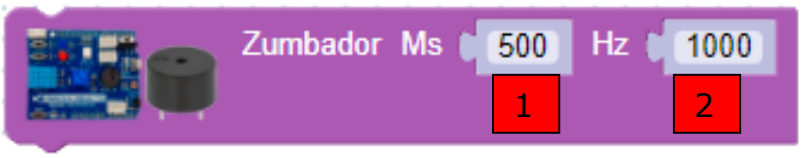
En la siguiente tabla están las frecuencias del sonido de las notas musicales:

Nota	Frecuencia (Hz)
do (control)	261.6
do#	277.2
re#	293.7
mi	329.6
fa	349.2
fa#	370
sol	392
sol#	415.3
la	440
la#	466.2
si	493.2
do	523.3

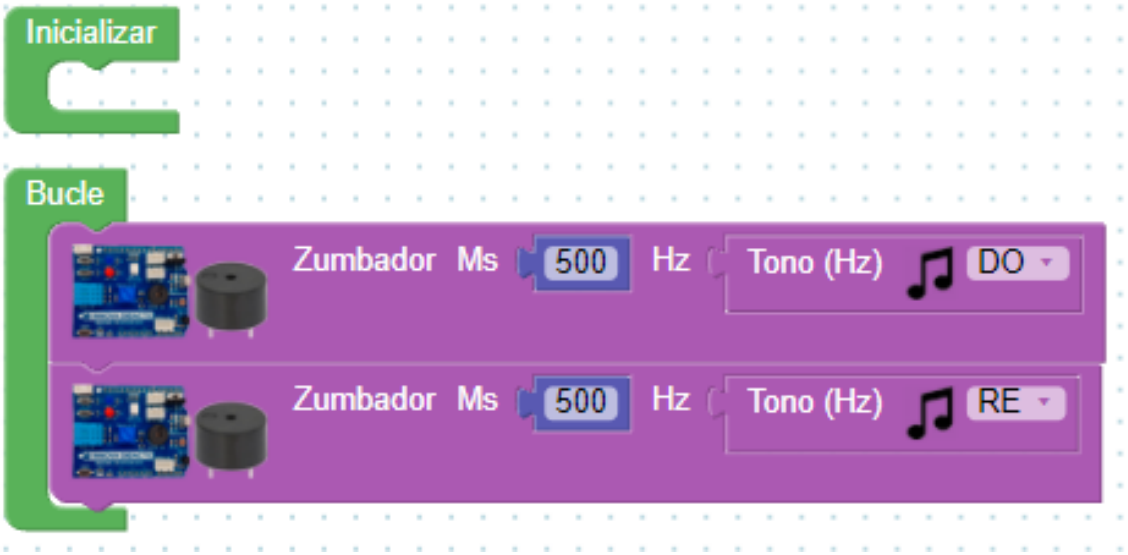
7.3.1 Reto A03.1. Primeros sonidos con el zumbador.

Reto A03.1. Primeros sonidos con el zumbador.

En el bloque Zumbador podemos variar dos parámetros: Ms (1) es el tiempo que dura cada sonido en milisegundos y Hz (2) es la frecuencia a la que vibra la membrana del zumbador para emitir el sonido.



Prueba con este sencillo programa cómo suena el zumbador.



Actividad de ampliación: cambia ligeramente el programa introduciendo unas esperas entre un sonido y otro para ver las diferencias que aparecen en la ejecución del programa.

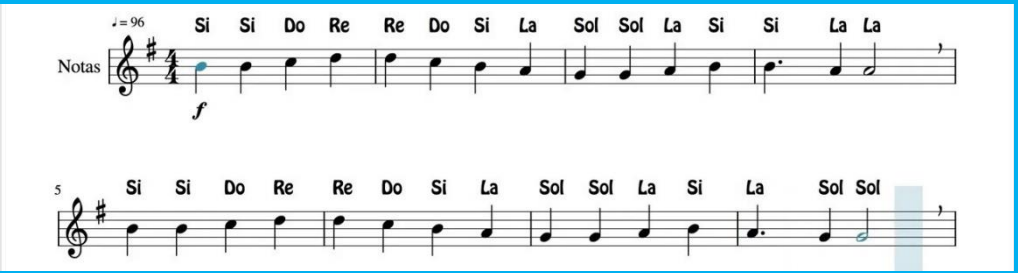
7.3.2 Reto A03.2. Escalas musicales con el zumbador.

Reto A03.2. Escalas musicales con el zumbador.

Ahora realizaremos un programa que emita las notas de la escala diatónica (teclas bemoles).

Se puede introducir una pequeña espera entre nota y nota si no queremos que suenen juntas (ligadas).

Actividad de ampliación: intenta tocar esta melodía utilizando el zumbador. Las notas negras deben tener una duración de 500ms, la negra con puntillo 750ms y la blanca 1000ms.

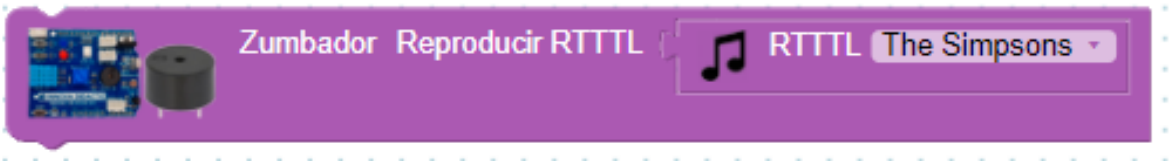


7.3.3 Reto A03.3. Melodías con RTTTL.

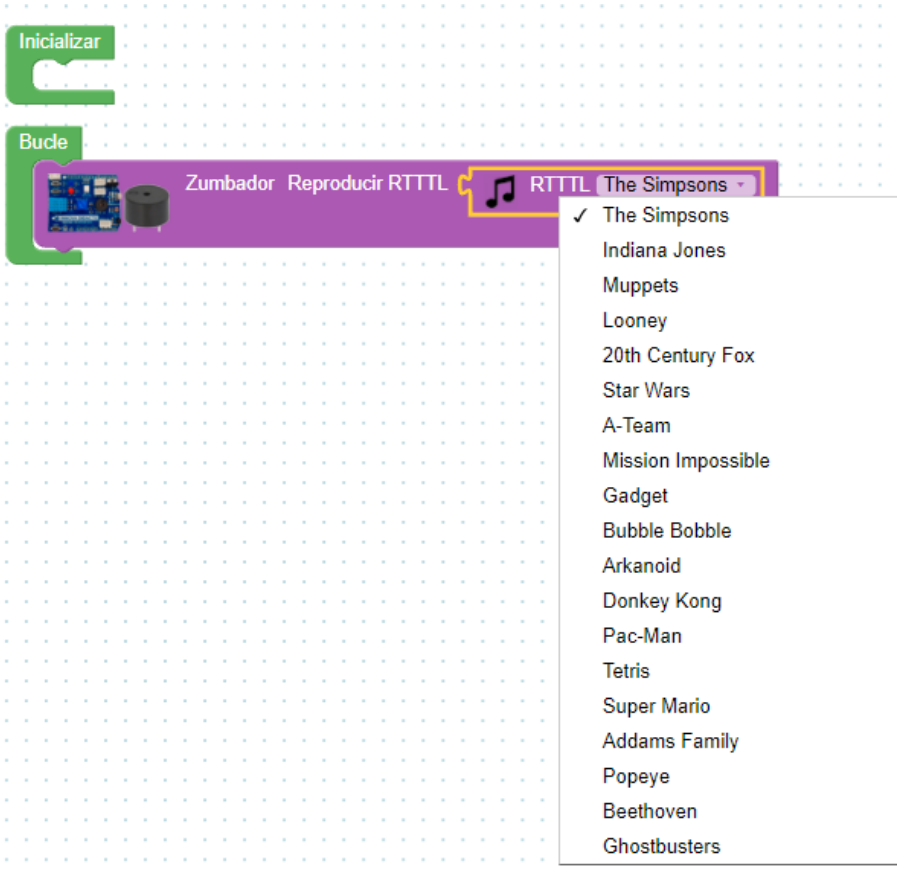
Reto A03.3. Melodías con RTTTL.

Las melodías en formato RTTTL (Ring Tone Text Transfer Language) es un lenguaje muy simple, creado por Nokia, con el objetivo inicial de definir de forma sencilla partituras musicales en formato texto para móviles.

Estas melodías RTTTL se pueden introducir de forma sencilla desde ArduinoBlocks y sólo se necesitan dos bloques.



Realiza este programa y elige una de las melodías que hay disponibles en el menú desplegable del bloque RTTTL.



Actividad de ampliación: prueba con diferentes melodías RTTL.

7.4 Reto A04. El pulsador.

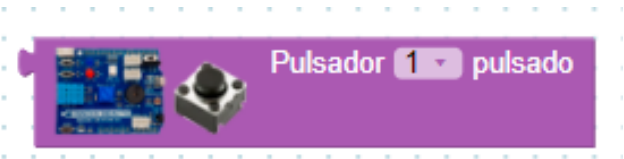
Reto A04. El pulsador.

En el siguiente reto vamos a utilizar el pulsador. Previamente debemos recordar que diferencia hay entre un pulsador y un interruptor. Un interruptor es un dispositivo que abre o cierra en paso de la corriente eléctrica, por ejemplo, los interruptores de la luz de nuestras casas, cada vez que los pulsamos cambian de estado y permanecen en él hasta ser pulsados de nuevo. Sin embargo, un pulsador sólo se activa mientras dure la pulsación volviendo a su estado inicial en el momento en el que se deje de pulsar.

Hay dos tipos de pulsadores; los NA (normalmente abierto) o los NC (normalmente cerrado), con lo que al pulsarlo se activará la función inversa de la que en ese momento este realizando.

La placa Imagina TDR STEAM tiene dos pulsadores de nominados SW1 y SW2 que van asociados a los pines D2 y D7 respectivamente.

Ahora vamos a realizar un programa en el cual al pulsar sobre el pulsador se encienda el led y se apague cuando lo dejemos de pulsar. En el menú de *TDR STEAM* encontramos el bloque del pulsador.



7.4.1 Reto A04.1. Control ON/OFF de un led con un pulsador.

Reto A04.1. Control ON/OFF de un led con un pulsador.

Para realizar este programa necesitamos conocer unas de las funciones más utilizadas en programación. Las funciones del menú *Lógica* con las funciones de condición (condicionales).

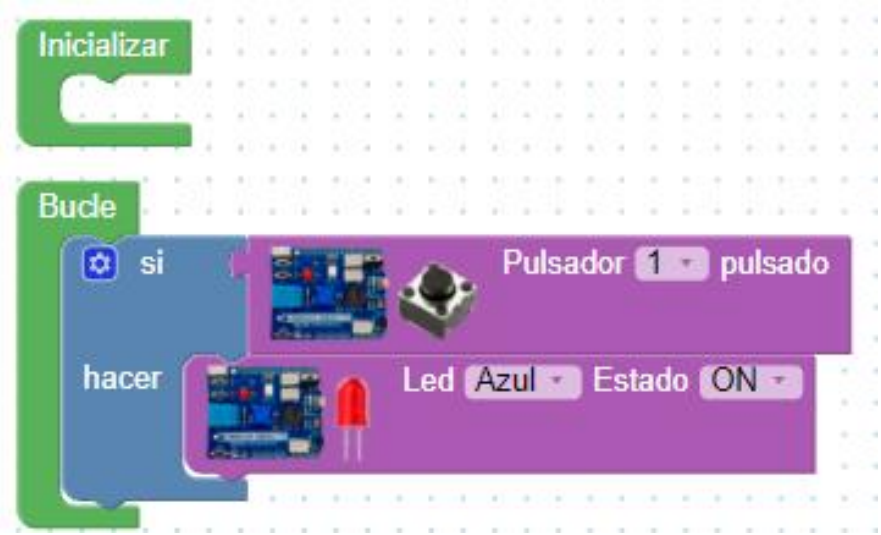
Condicionales:

si (condición) hacer (acciones)

Se trata del famoso bucle *Si* (*if* en inglés) que es uno de los pilares de la programación, ya que permite evaluar estados y tomar decisiones en consecuencia.

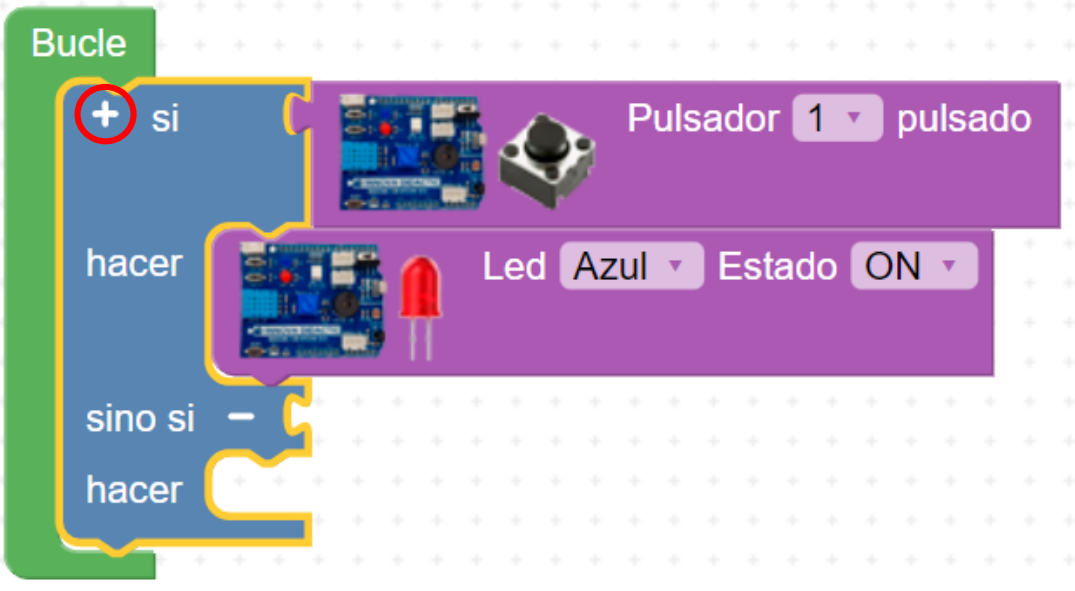
El funcionamiento es el siguiente: si se cumple la condición incluida en su primer apartado, entonces se realiza la acción incluida en su segundo aparado. En caso contrario, no se hace nada.

En el apartado de condiciones se pueden introducir multitud de factores: estado de sensores (analógicos o digitales), comparaciones, igualdades, operaciones matemáticas, etc. Debemos tener en cuenta que siempre estamos trabajando con condiciones cierto/falso. Usando el bloque lógico *condicional de Si…. hacer…* el programa quedaría como la imagen.

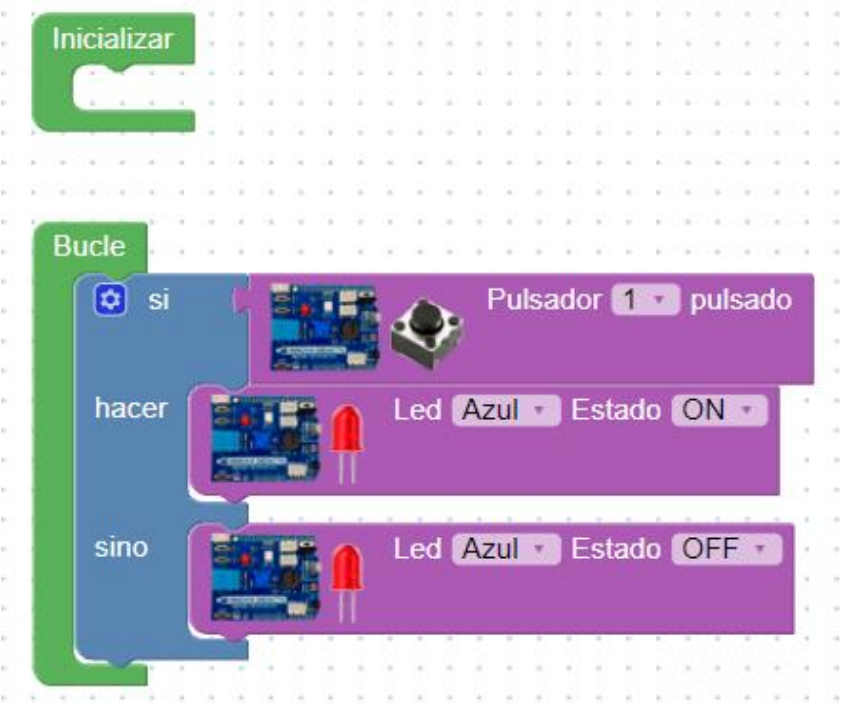


No se apaga el led azul nunca, esto no es lógico, en ningún momento del programa decimos que el led tenga que estar en la posición OFF. Con este otro programa al pulsar el SW1 se enciende el led azul y dejar de pulsar SW1 se apaga.

Modificaremos el bloque condicional para añadir una nueva condición pulsando sobre el +.

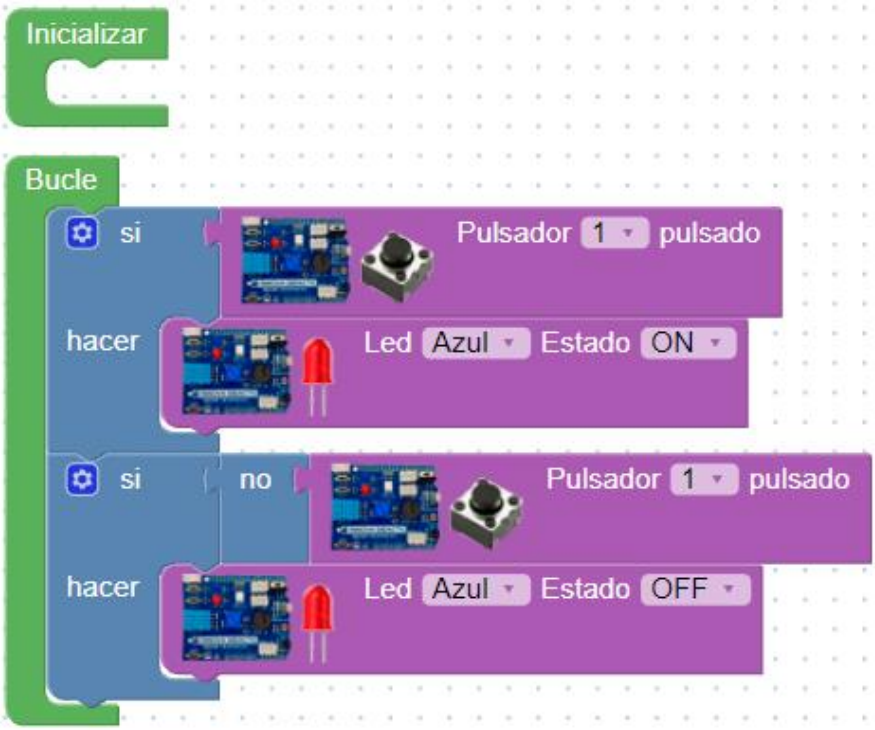


El programa resultante será:



El led se encenderá mientras apretemos el pulsador y se apagará al soltarlo.

Otra forma diferente de hacer el programa sería:



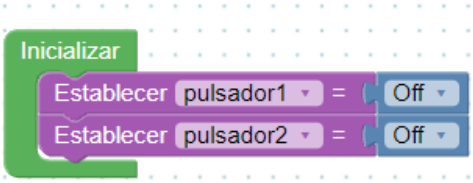
Actividad de ampliación: prueba ahora de que el funcionamiento del led sea a la inversa.

7.4.2 Reto A04.2. Control ON/OFF de un led con dos pulsadores.

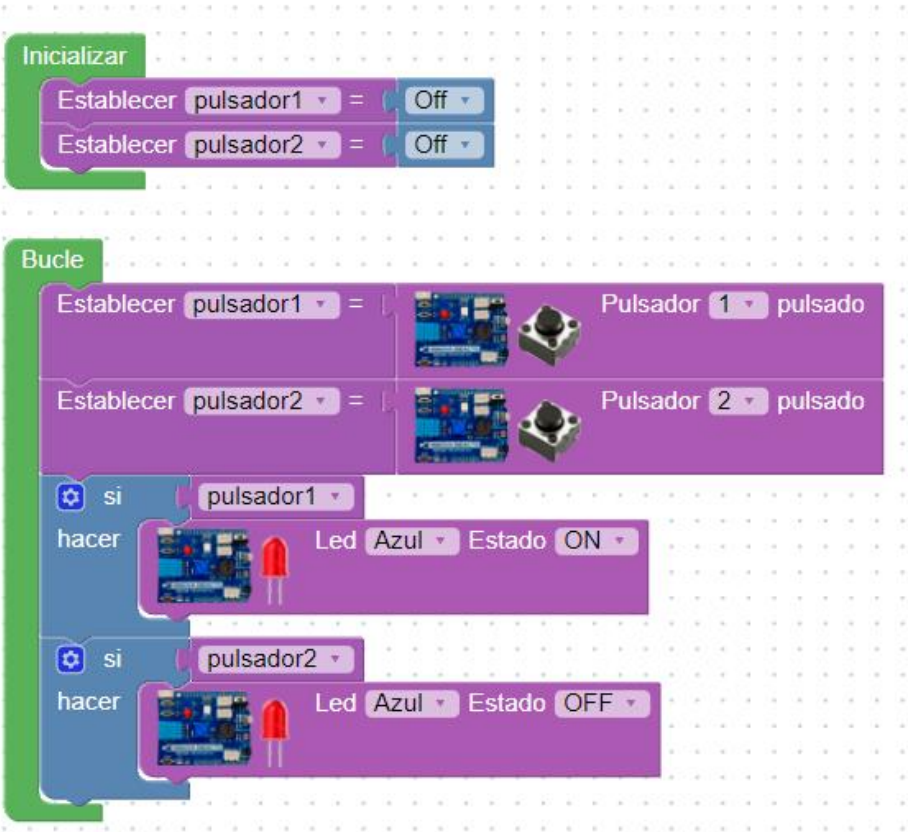
Reto A04.2. Control ON/OFF de un led con dos pulsadores.

También podemos realizar el programa anterior definiendo una variable para cada sensor. El programa es un poco más largo, pero para programas más complejos es muy útil.

Primero creamos dos variables booleanas para almacenar el estado de los pulsadores.



En el Bucle, leeremos las dos entradas de los pulsadores y en función del estado de cada pulsador, encenderemos o apagaremos el led. El programa resultante es el siguiente:

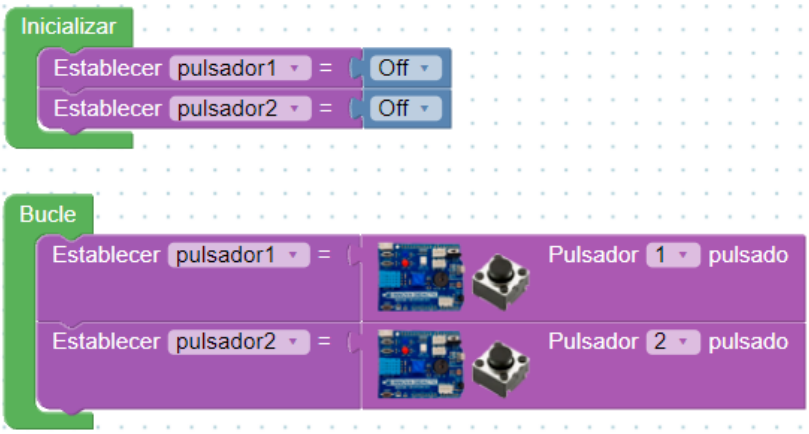


Actividad de ampliación: prueba ahora de que el funcionamiento de los pulsadores sea a la inversa.

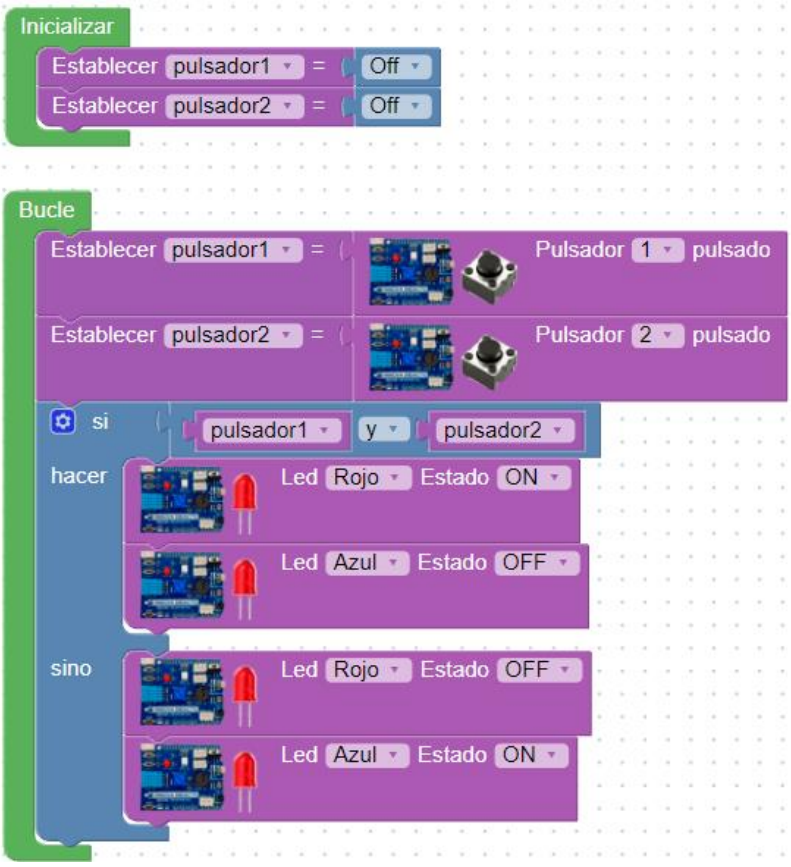
7.4.3 Reto A04.3. Control ON/OFF de dos leds y dos pulsadores.

Reto A04.3. Control ON/OFF de dos leds y dos pulsadores.

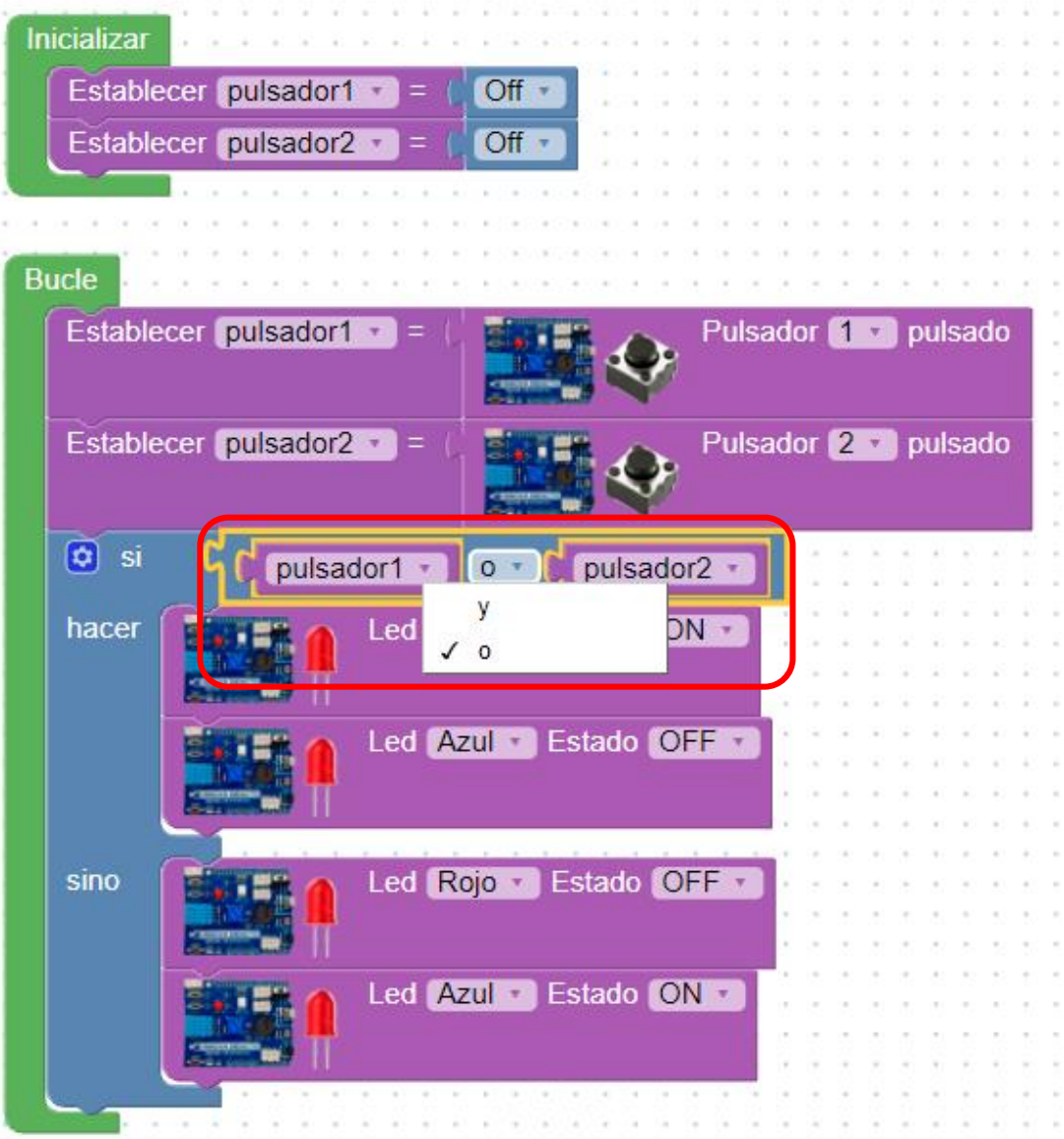
Vamos a realizar diferentes programas para el control de dos leds mediante dos pulsadores. Crearemos dos variables para almacenar el estado de los pulsadores y la lectura del estado de los pulsadores. Esta parte será común para todos los programas y después, haremos diferentes condiciones para el encendido y apagado de los leds.



Vamos a encender los leds de manera alternativa si pulsamos los dos pulsadores a la vez.



Ahora podemos probar de cambiar la condición a que si apretamos cualquiera de los dos pulsadores encenderá el led rojo y apagará el led azul y, si no se cumple esta condición, el led rojo se apagará y se encenderá el azul. Sólo tenemos que cambiar la condición en el condicional.

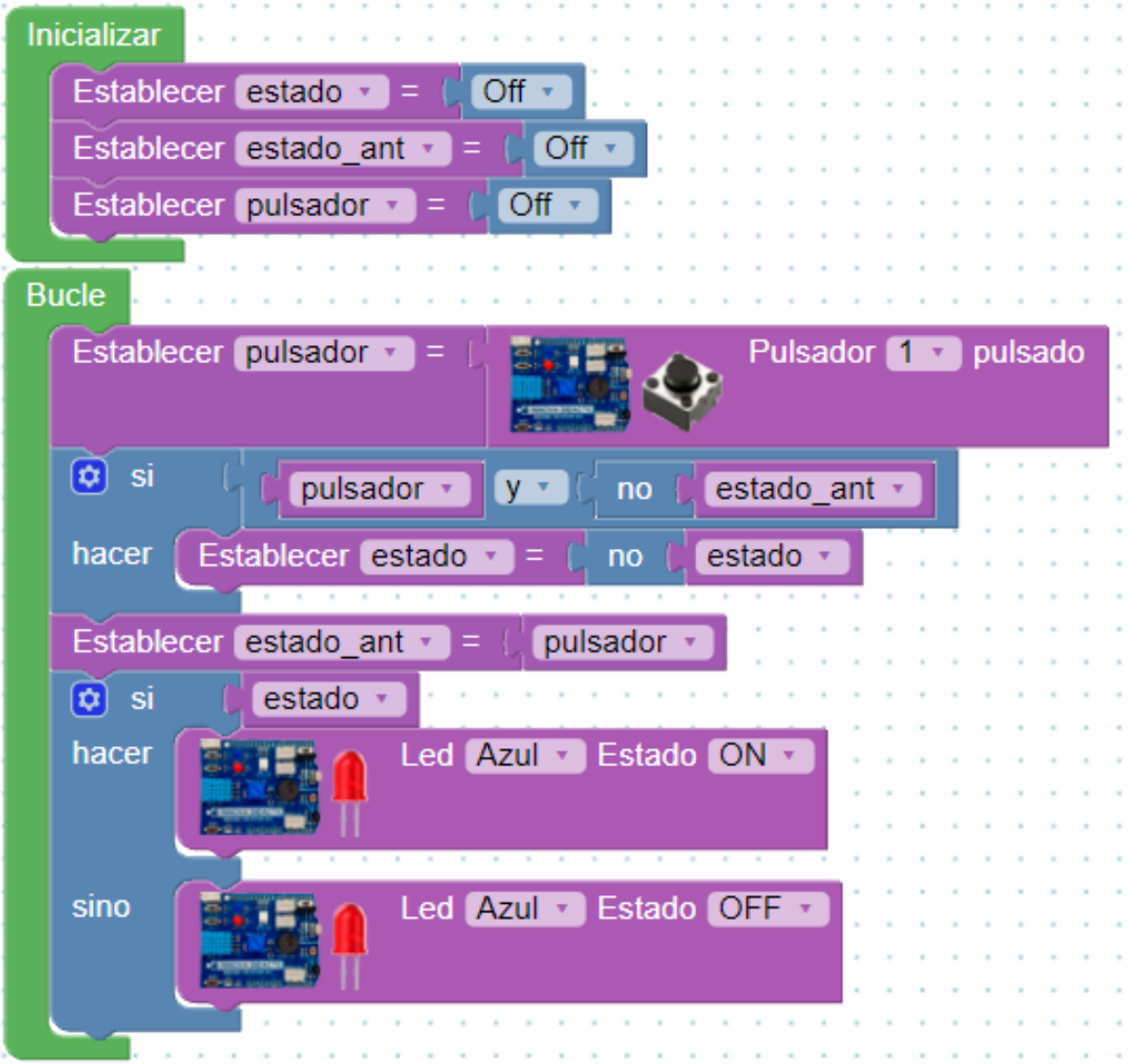


Actividad de ampliación: realiza los dos programas anteriores y comprueba su funcionamiento.

7.4.4 Reto A04.4. Control ON/OFF de un led por pulsación.

Reto A04.4. Control ON/OFF de un led por pulsación.

Ahora vamos a realizar un programa que encienda un led cuando apretemos el pulsador y se mantendrá encendido hasta que volvamos a apretarlo, que lo dejará apagado.



Enlace al programa: ArduinoBlocks Projects\pulsador_4.abp

Actividad de ampliación: realiza el programa, comprueba su funcionamiento e intenta entender cómo funciona.

7.5 Reto A05. El potenciómetro.

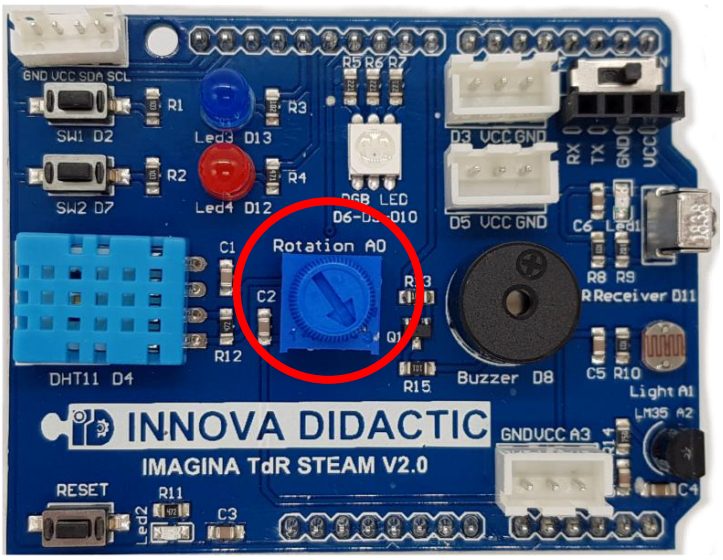
Reto A05. El potenciómetro.

Un potenciómetro es una resistencia cuyo valor es variable ya que son un tipo de resistencias especiales que tienen la capacidad de variar su valor cambiando de forma mecánica su posición. Con ellos indirectamente, se puede controlar la intensidad de corriente que fluye por un circuito si se conecta en paralelo, o controlar el voltaje al conectarlo en serie. Son adecuados para su uso como elemento de control en los aparatos electrónicos como el control de volumen, brillo, etc.

Símbolo

Componente

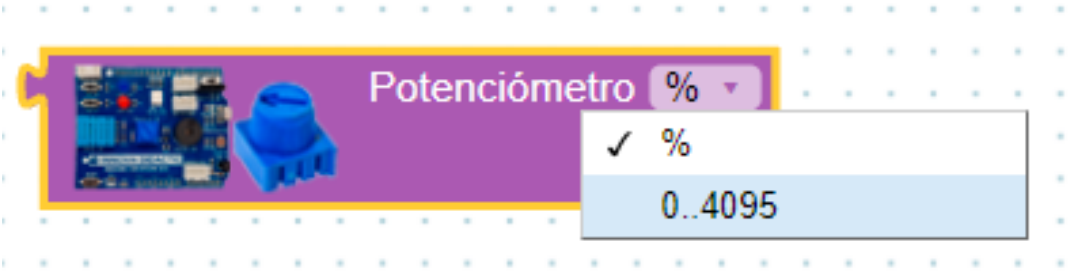
La placa ***Imagina TDR STEAM*** tiene un potenciómetro denominado *Rotation* que van asociado al pin A0. Las entradas *A_{número}* son entradas analógicas, así que empezamos con el uso de este tipo de entradas. Este potenciómetro permite realizar un giro de unos 270º entre topes (3/4 de vuelta).



La diferencia entre un sensor **analógico** y **digital** es que mientras este último, el digital, sólo permite dos tipos de entradas, 0-1, *alto-bajo*, *high-low*, *on-off*, un sensor analógico puede tener infinidad de valores. En la ***ESP32 Plus STEAMakers***, las entradas analógicas pueden tener 2^{12} valores (12 bits de resolución=4.096 valores), es decir, valores comprendidos entre 0 y 4.095.

En el menú de *TDR STEAM* de ArduinoBlocks, disponemos de un bloque específico para realizar programas utilizando el potenciómetro de nuestra placa.

En el desplegable del bloque del sensor, podemos elegir su lectura en porcentaje (%) o en valor (de 0 a 4095).

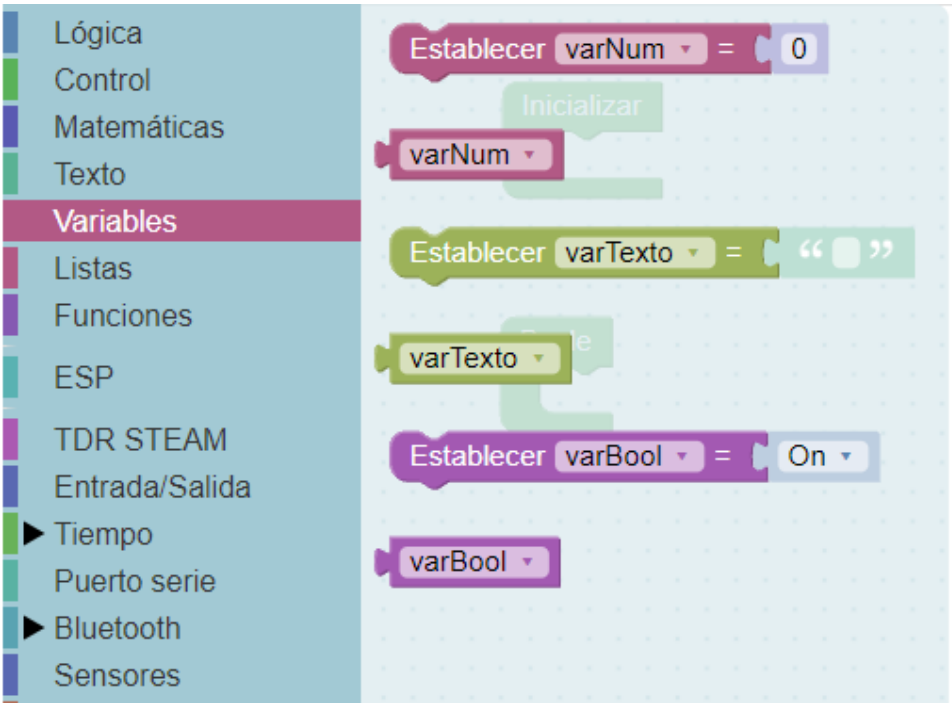


7.5.1 Reto A05.1. Lectura de valores con el puerto serie.

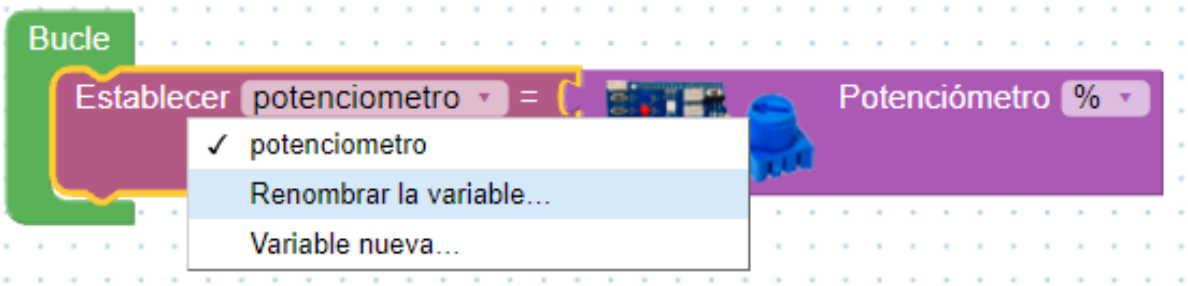
Reto A05.1. Lectura de valores con el puerto serie.

Para realizar una lectura de los valores del sensor es necesario utilizar la *Consola* (lector de datos por el puerto serie) que nos ofrece ArduinoBlocks, vamos a ver como se hace.

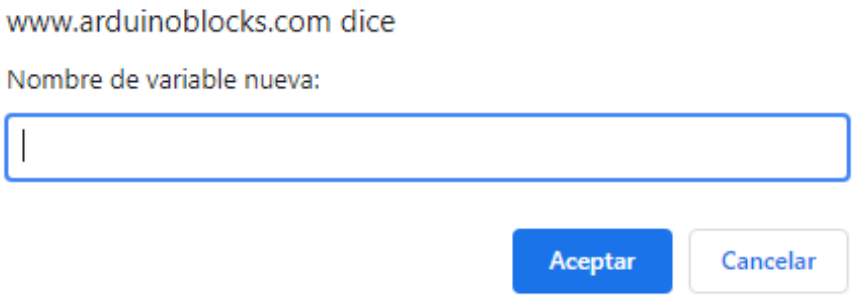
En primer lugar, generamos una variable a la que llamaremos *potenciometro* (en programación no se ponen acentos).



Ahora fijaremos el valor de la variable al valor del potenciómetro, tal y como está en la imagen.



Para cambiar el nombre de la variable pulsaremos sobre el menú desplegable del bloque de la variable y elegiremos *Variable nueva…* nos aparecerá una ventana en la que escribiremos el nuevo nombre y daremos a *Aceptar*.

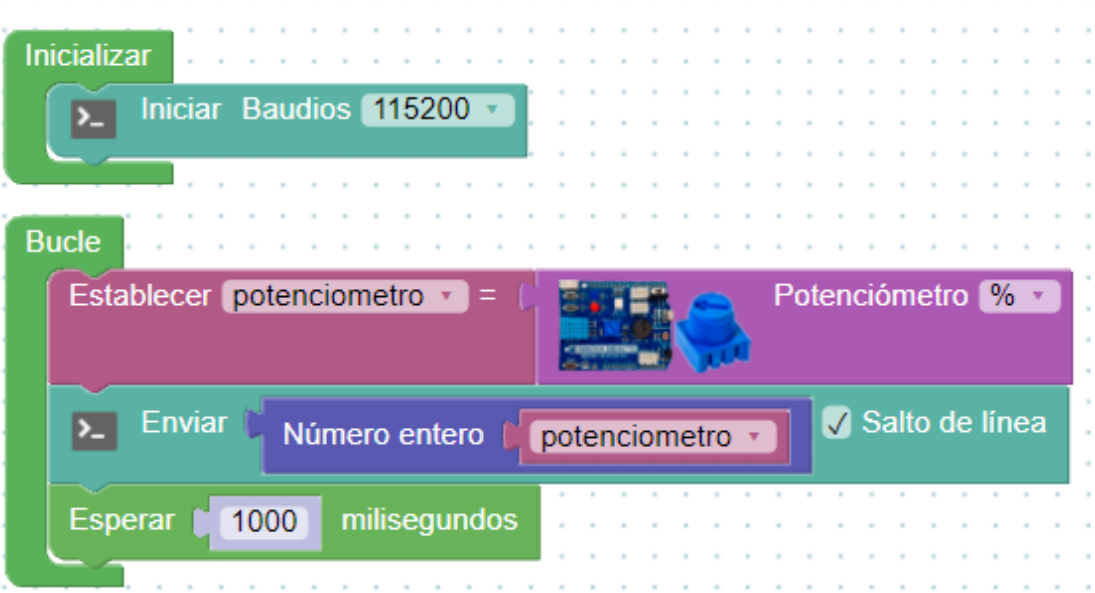


Es importante establecer la variable con el valor del potenciómetro dentro de *Bucle*, ya que si sólo se hace en *Inicializar* el valor siempre será el mismo a lo largo de todo el programa. En otras ocasiones interesa establecer las variables en el inicio, pero no es este el caso.

Continuando con el programa, ahora nos faltan los bloques del *Puerto Serie*. El primero que debemos utilizar es el *Iniciar Baudios 115200* que siempre lo colocaremos en el Inicio y después el bloque *Enviar.*

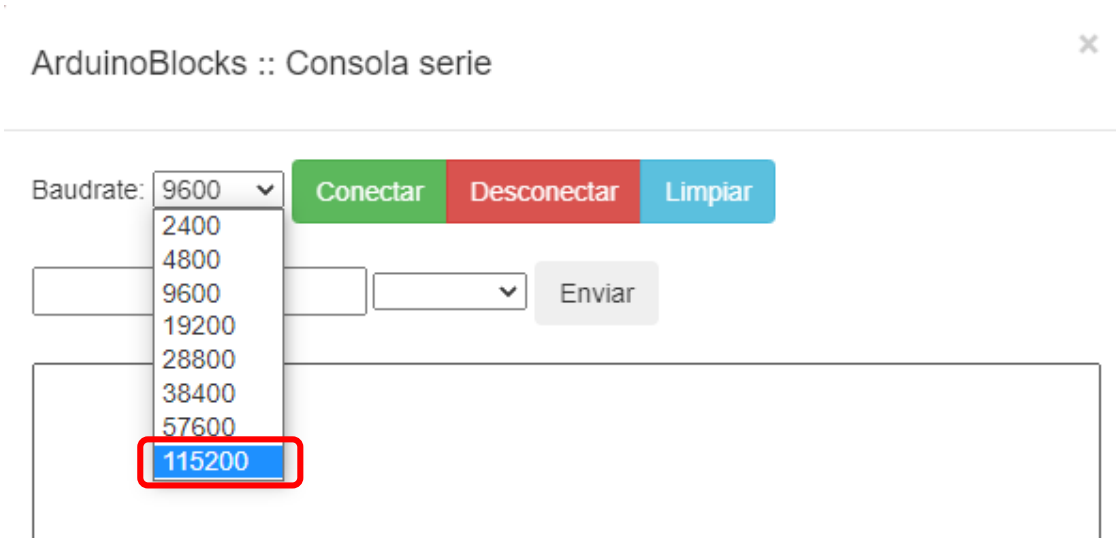
El programa quedará de la siguiente forma:

Para que no salga en la pantalla de la Consola los números decimales añadiremos el bloque de *Número entero* que está en el menú *Matemáticas* cuando enviemos el valor de la variable. El programa definitivo será:



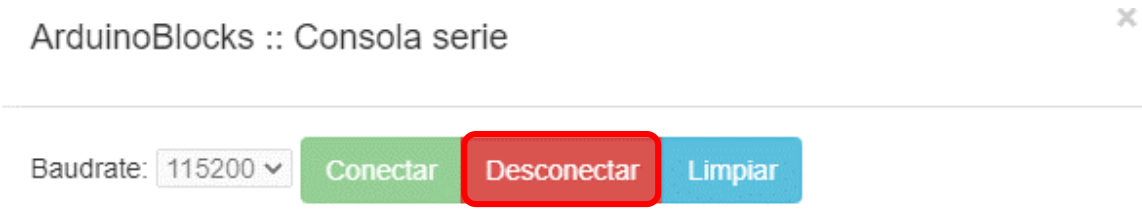
Ahora ya podemos enviar el programa a la placa y después abrimos la *Consola*.

Se abrirá la siguiente ventana donde podremos elegir la velocidad de comunicación (Baudrate). Pulsaremos sobre el botón Conectar para poder comenzar a visualizar valores. De esta manera podremos ver cada segundo el valor de nuestro potenciómetro. Gira el potenciómetro y observa cómo van cambiando los valores.

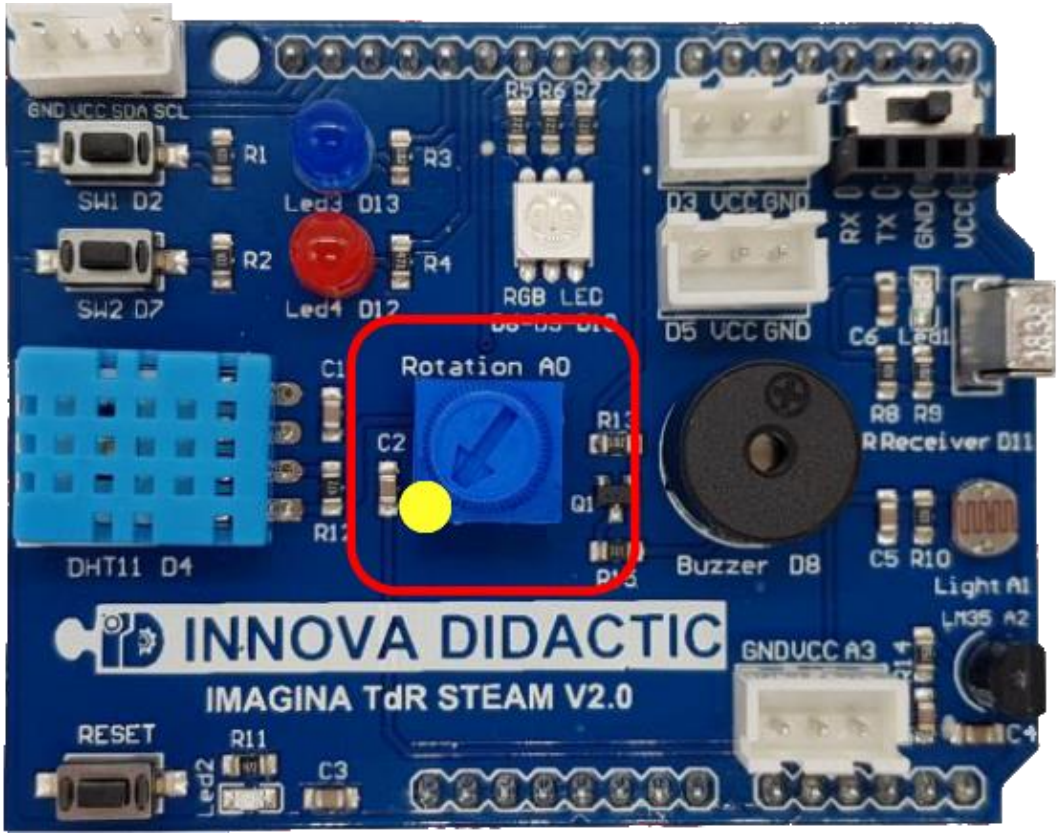


Al variar el potenciómetro podremos ver datos entre 0 y 100 (en %).

Pulsa sobre Desconectar para cerrar la comunicación.



Recuerda que para volver a enviar el programa a la placa el potenciómetro deberá estar en la posición 0 (potenciómetro a la izquierda, donde indica el punto amarillo).



Actividad de ampliación: prueba ahora quitando el tic de *Salto de línea* a ver qué sucede.

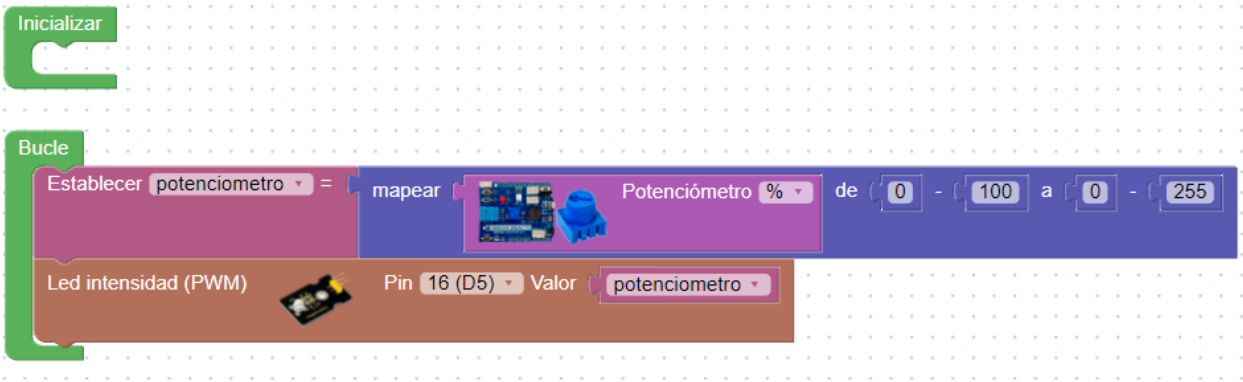
7.5.2 Reto A05.2. Ajuste de valores de entrada y salida: mapear.

Reto A05.2. Ajuste de valores de entrada y salida: mapear.

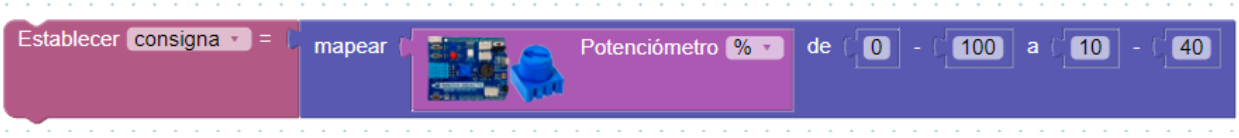
Existe un pequeño "problema" entre las entradas y las salidas en Arduino. Las entradas trabajan con 12 bits (2^{12} valores = 0 a 4095) y las salidas trabajan a 8 bits (2^8 valores = 0 a 255). Debido a esto, debemos realizar un cambio de escala. A este cambio de escala se le llama "mapear".

En el menú *Matemáticas* existe un bloque llamado *mapear*. Este bloque 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.

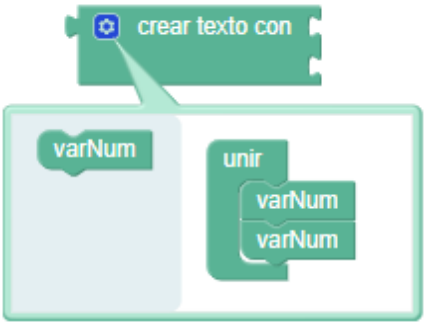
Haremos una pequeña prueba conectando un led a la salida D5 (pines libres) para poder hacer la regulación del led mediante el potenciómetro.



Vamos a realizar una actividad en la que queremos definir una consigna de temperatura y sus límites están en un rango entre 10ºC y 40ºC. Para ello definiremos una variable, llamada Consigna, que será el valor *mapeado* del potenciómetro.

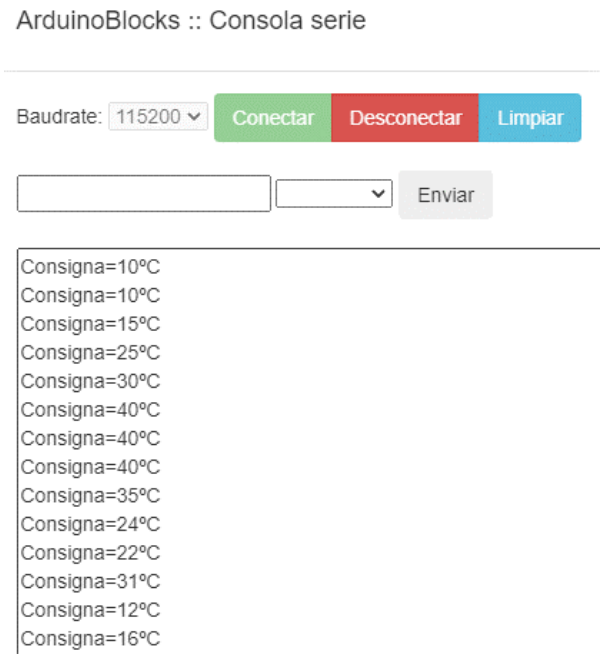


Ampliaremos el programa anterior para realizar lecturas por el puerto serie utilizando el nuevo bloque de *crear texto con...*. Fíjate como al pulsar sobre el símbolo del engranaje podemos ampliar las líneas añadiendo *varNum* a la parte derecha.



Después añade el bloque y termina el programa como queda en la imagen.

Por último, carga el programa, abre la *Consola* y comprueba las lecturas moviendo el potenciómetro.

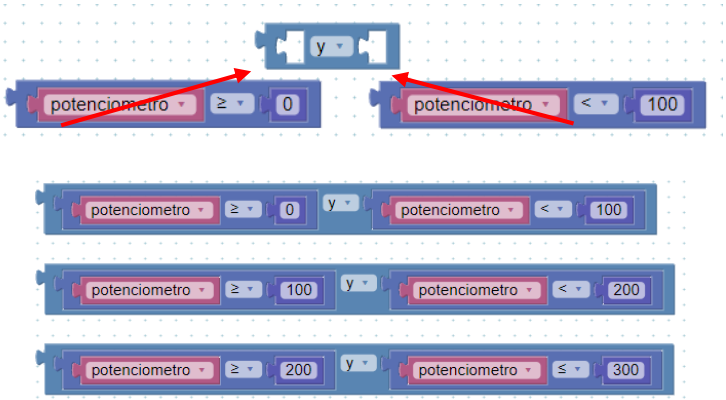


Actividad de ampliación: cambia ahora el rango de salida y el texto que envía por el puerto serie.

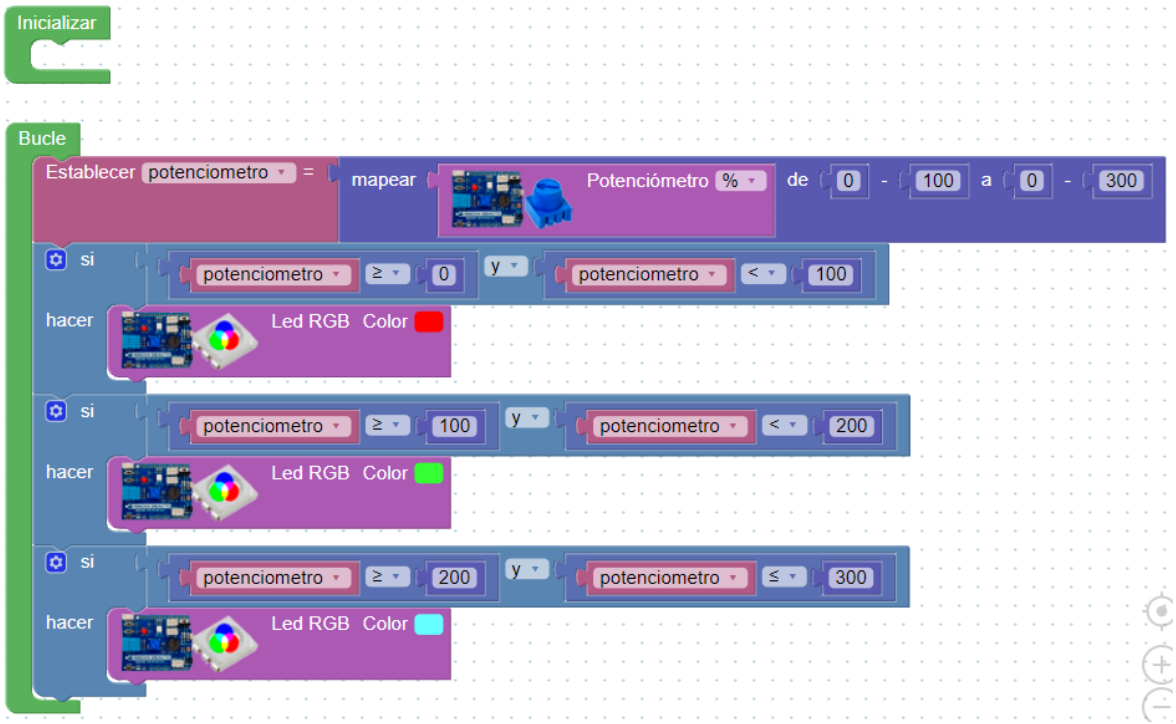
7.5.3 Reto A05.3. Control del led RGB con el potenciómetro.

Reto A05.3. Control del led RGB con el potenciómetro.

En la siguiente actividad vamos a controlar los colores del led RGB utilizando el potenciómetro. Vamos a hacer que cambie de color según varíe el valor del potenciómetro. Es decir, cuando el valor del potenciómetro se encuentre entre 0 y 100 que el color del led sea rojo, cuando se encuentre entre 101 y 200 que sea verde y cuando esté entre 201 y 300 que sea azul. Del menú *Lógica* vamos a necesitar los bloques para realizar las condiciones. Las condiciones las crearemos poco a poco, como se ve en la siguiente imagen.



El programa quedaría como muestra la imagen:



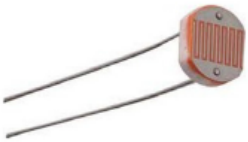
Actividad de ampliación: completa el programa con más condiciones y más colores.

7.6 Reto A06. La fotocélula (LDR o sensor de luz).

Reto A06. La fotocélula (LDR o sensor de luz).

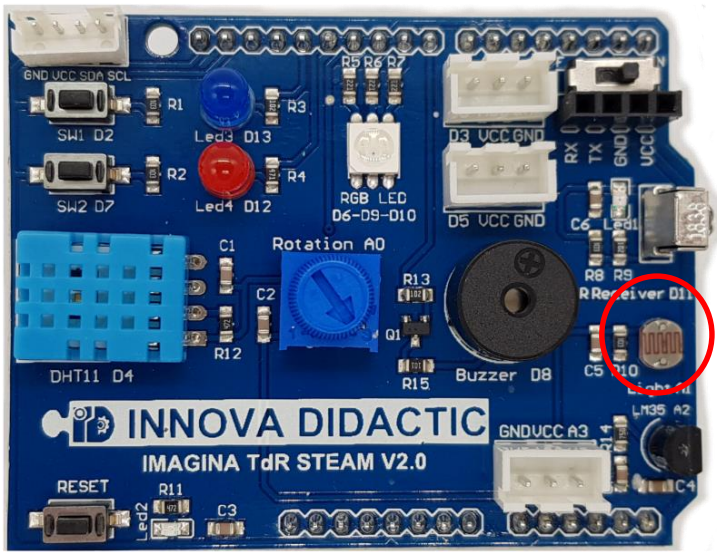
Ahora que ya sabemos usar el Puerto Serie para leer los valores de los sensores, vamos a utilizarlo para ver el valor de una fotocélula (LDR). Una LDR (Light Dependent Resistor) es un resistor que varía su valor de resistencia eléctrica dependiendo de la cantidad de luz que incide sobre él. El valor de la resistencia disminuye con el aumento de intensidad de luz que incide sobre ella.

Símbolo

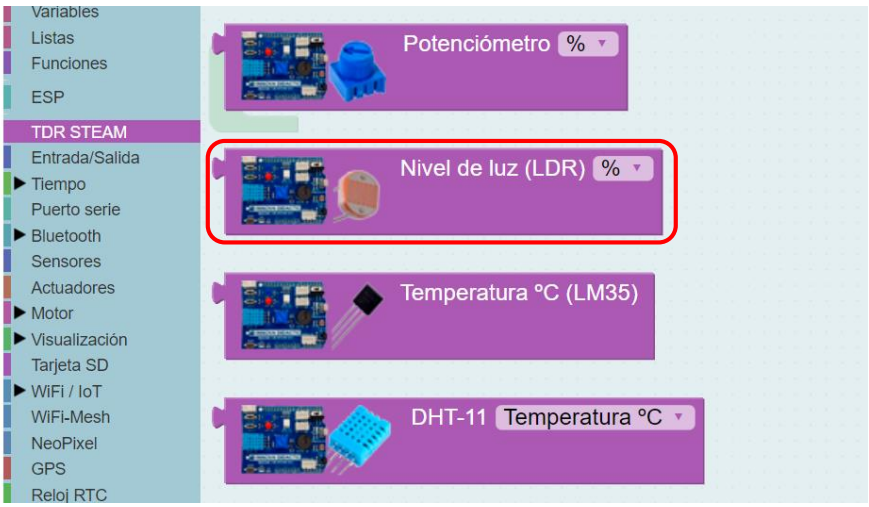


Componente

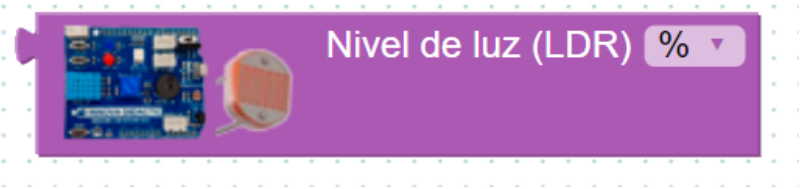
En la placa ***Imagina TDR STEAM*** la fotorresistencia está denominada como "*Light*" y viene conectada en el Pin analógico A1.



En el menú *TDR STEAM* de ArduinoBlocks hay un bloque específico para el uso de este sensor.

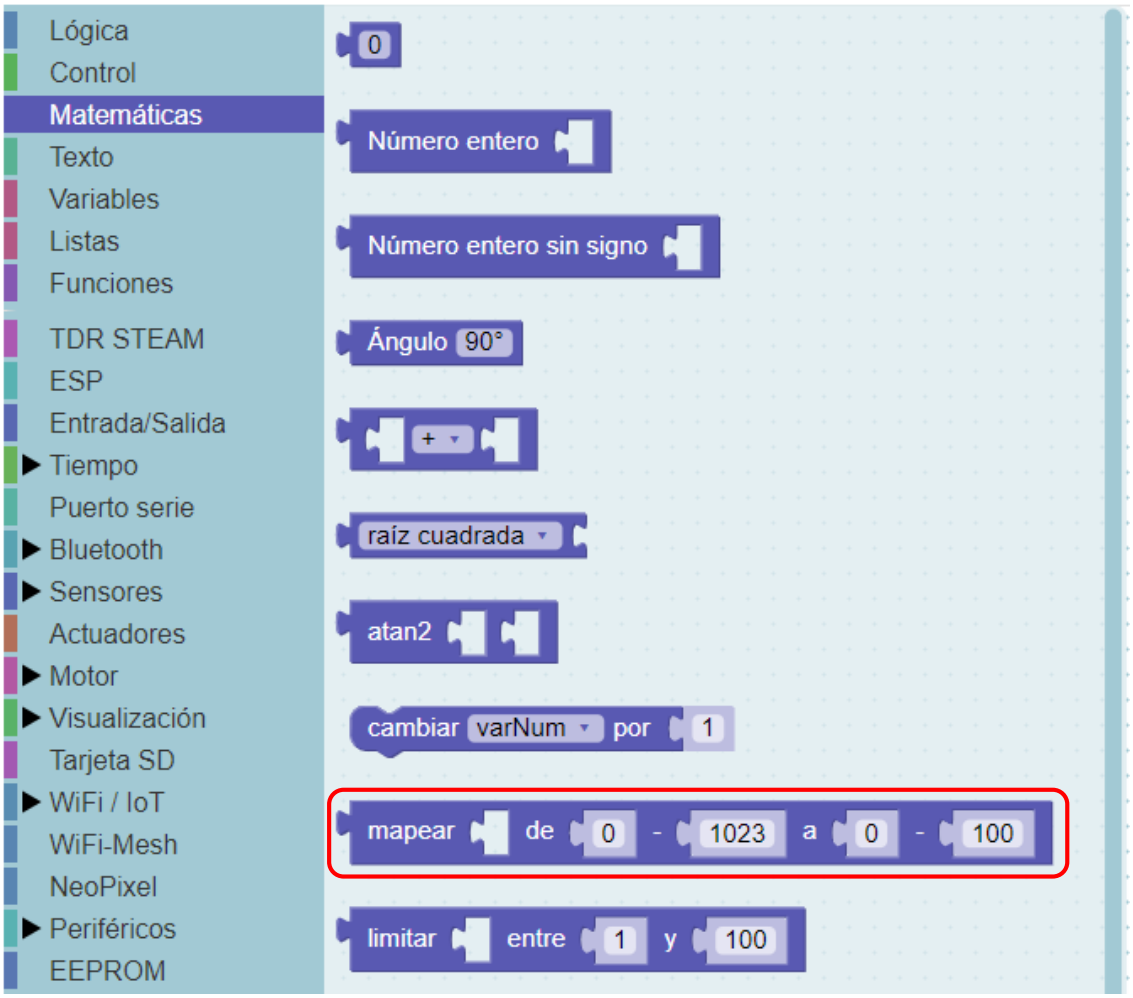


En este bloque, también se puede seleccionar el tipo de lectura del valor del sensor en % o en unidades de 0 a 4095.



El nivel mínimo que detecta la LDR con el ESP32 Plus STEAMakers es de aproximadamente 27, así que podemos hacer un escalado para que funcione mejor. Esto lo realizaremos en la actividad 2.

Disponemos de un bloque para poder realizar cambios de escala (maperar).

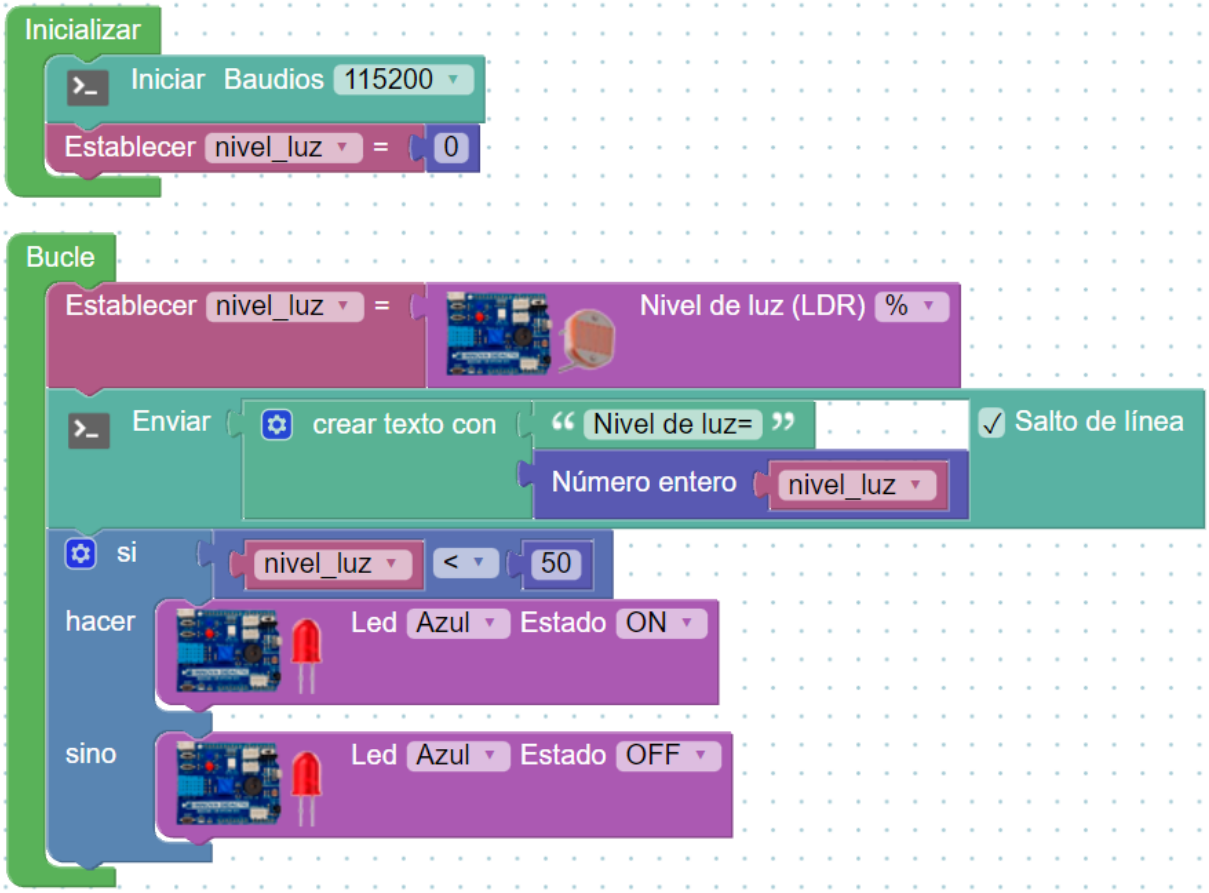


7.6.1 Reto A06.1. Encender y apagar un led según el nivel de luz.

Reto A06.1. Encender y apagar un led según el nivel de luz.

En esta actividad vamos a simular en el encendido automático de una farola cuando se hace de noche. Utilizando la LDR y el led azul vamos a hacer que cuando la LDR esté a oscuras se ilumine el led azul.

El programa es muy sencillo. Hay que generar una variable que la llamaremos *nivel_luz* y la asignaremos al sensor LDR en %. También mostraremos los datos por el puerto serie (*Consola*). Incluiremos un condicional para que, si el valor es menor de 50, se encienda el led azul y, sino, permanezca apagado.



Actividad de ampliación: realiza un programa basándote en el anterior que encienda también el led rojo si el valor de luz es superior al 80%.

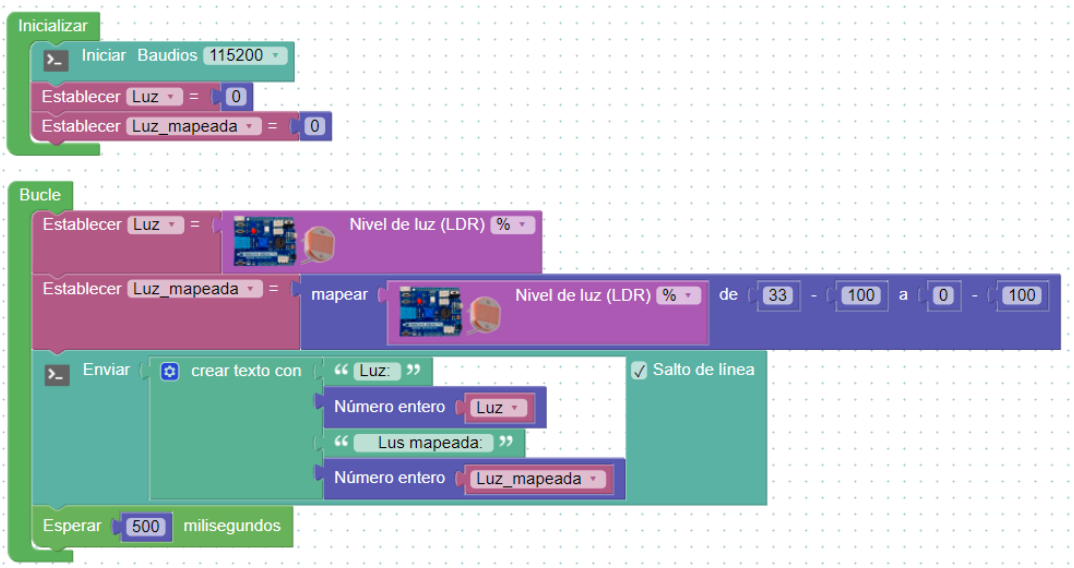
7.6.2 Reto A06.2. Cambio de escala.

Reto A06.2. Cambio de escala.

En esta actividad vamos a realizar un cambio de escala ya que el valor inferior de medición de la LDR es de aproximadamente 30%. Miraremos el valor mínimo real de medición y, a continuación, realizaremos el cambio de escala. Para poder realizar este proceso tenemos que ver los datos obtenidos por la *Consola*.

Enlace al programa: ArduinoBlocks Projects\ldr_cambio_escala.abp

El programa realizado es el siguiente:

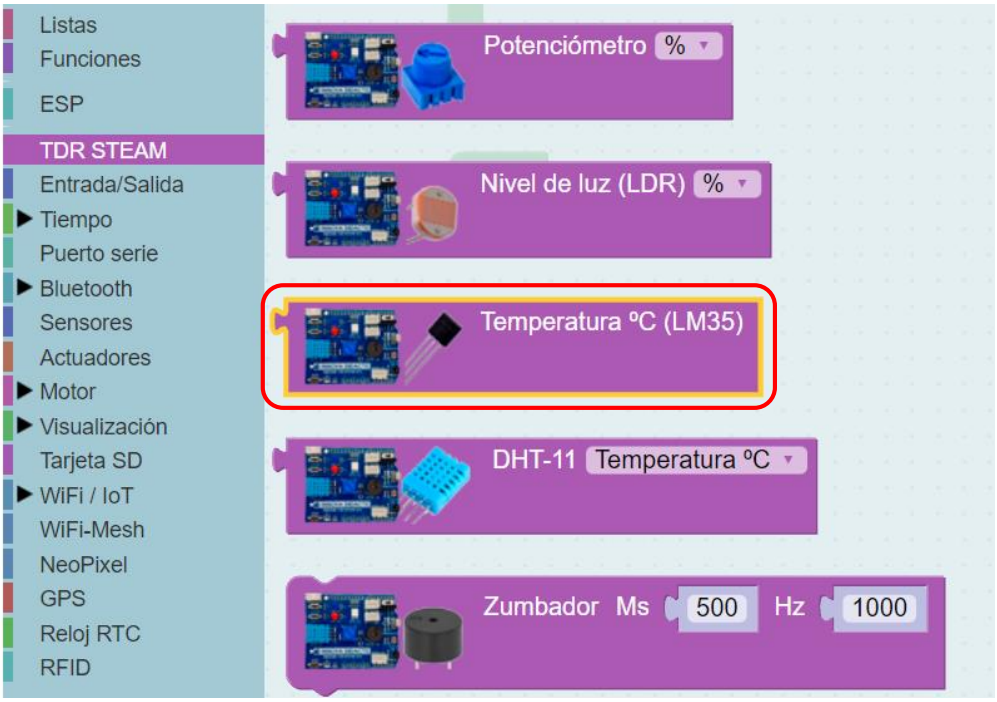


7.7 Reto A07. Sensor de temperatura LM35D.

Reto A07. Sensor de temperatura LM35D.

En el siguiente reto vamos a medir la temperatura de una habitación utilizando el sensor de temperatura LM35D. El LM35D es un sensor de temperatura que tiene un rango de temperatura de 0º a 100º ℃ y una sensibilidad de 10mV/ºC.

La placa Imagina TDR STEAM dispone de este sensor LM35D y está conectado en el Pin analógico A2. En el menú *TDR STEAM* de ArduinoBlocks hay un bloque específico para el uso de este sensor.



7.7.1 Reto A07.1. Lectura del valor de la temperatura.

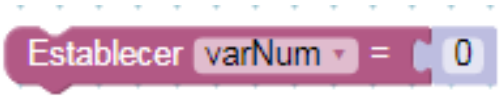
Reto A07.1. Lectura del valor de la temperatura.

Para ello vamos a repasar el concepto de variables. Lo vimos inicialmente con la práctica del LED PWM en la que incrementábamos y disminuíamos su brillo utilizando una variable *i*. En esta ocasión vamos a profundizar un poco más.

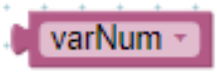
Las variables son elementos muy comunes en programación. Básicamente, crear una variable es darle un nombre a un dato o a una lectura. Por ejemplo, las mediciones de valores de temperatura las podemos guardar en una variable que se llame *Temperatura* o las del sensor de ultrasonidos en una llamada *Distancia*, etc. No es obligatorio su uso, pero nos permiten trabajar más cómodamente, además, como podemos personalizar su nombre, ayudan a clarificar el código y utilizar un lenguaje más natural.

Al trabajar con variables vamos a tener dos tipos de bloques principales:

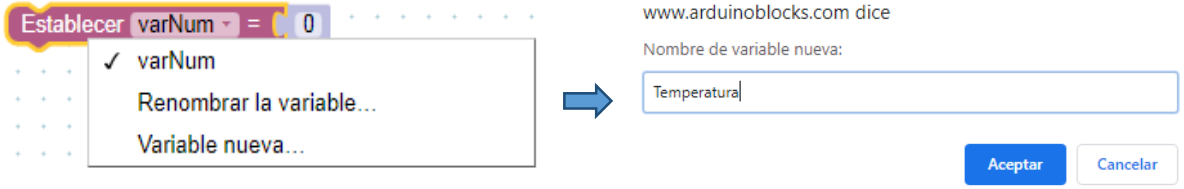
- El bloque en el que le damos valor a la variable:



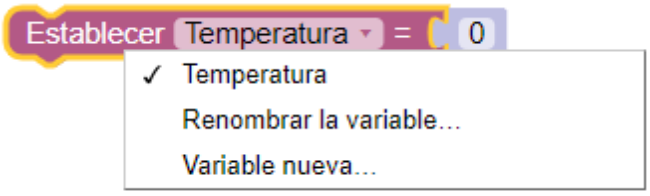
- Y el bloque de la propia variable creada, para poder insertarla y combinarla con otros bloques:



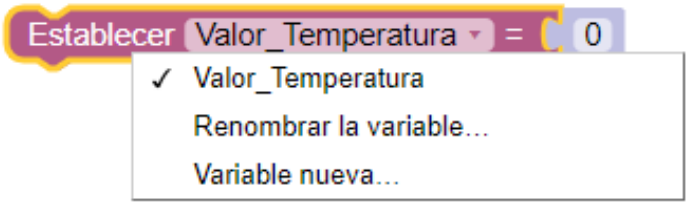
También podemos personalizar el nombre de la variable, de la siguiente forma:



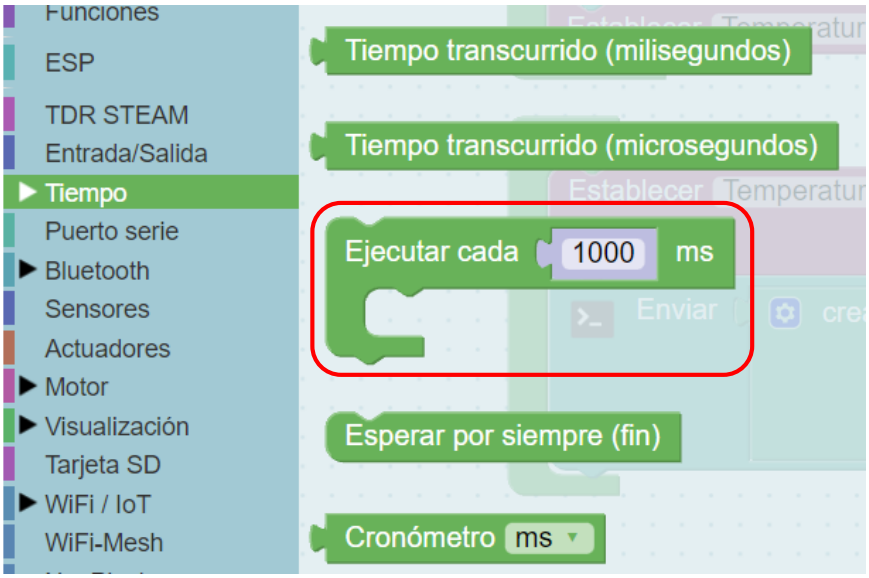
Una vez creada la nueva variable, podemos seleccionarla pulsando sobre el desplegable:



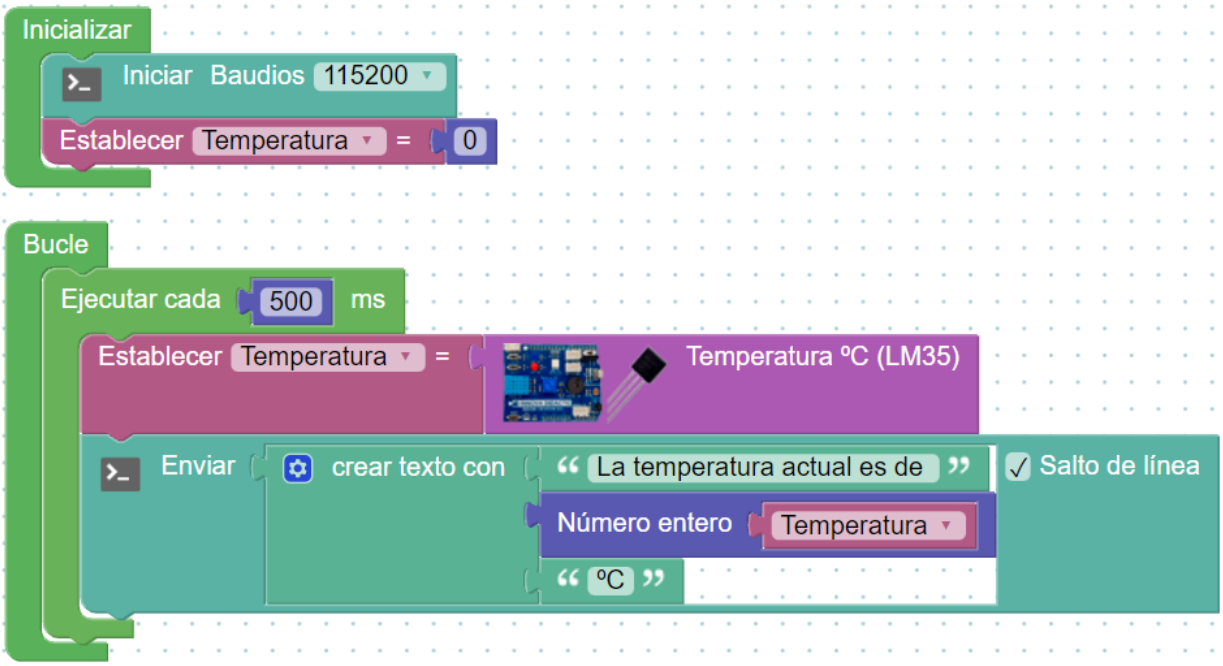
Hay que tener en cuenta que las variables solo pueden estar formadas por una palabra. Si quieres incluir varias palabras, puedes usar el truco de separarlas con una barra baja "_", como en el ejemplo, *Valor_Temperatura*.



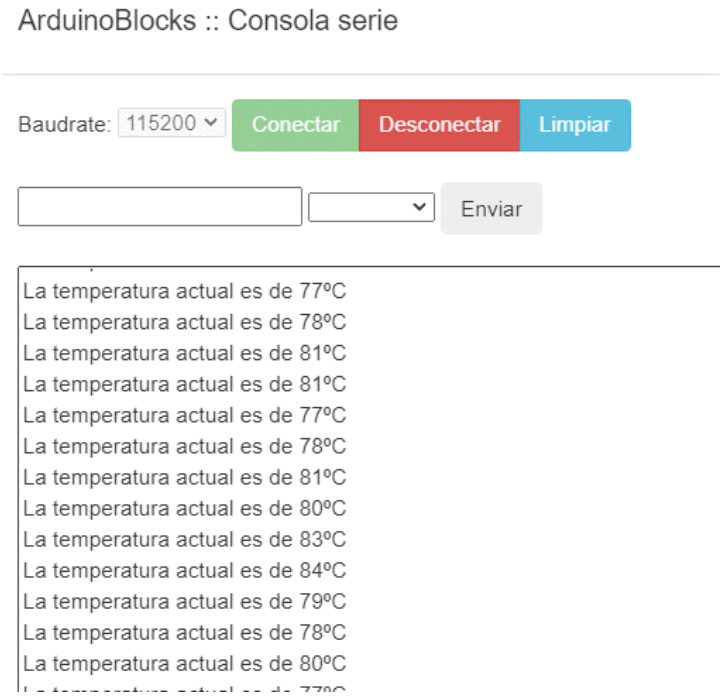
El programa que vamos a realizar mostrará valor del sensor de temperatura a través de la Consola. Haremos que se ejecute cada 500 milisegundos, pero en lugar de utilizar el bloque de *Esperar* utilizaremos el bloque *Ejecutar cada xxx ms*.



El programa resultante será:



Envía el programa a la placa, conecta la *Consola* y comprueba la información que aparece en el terminal.



Actividad de ampliación: realiza un programa que muestre datos más rápido y que muestre otro texto (Temp.= xx ºC).

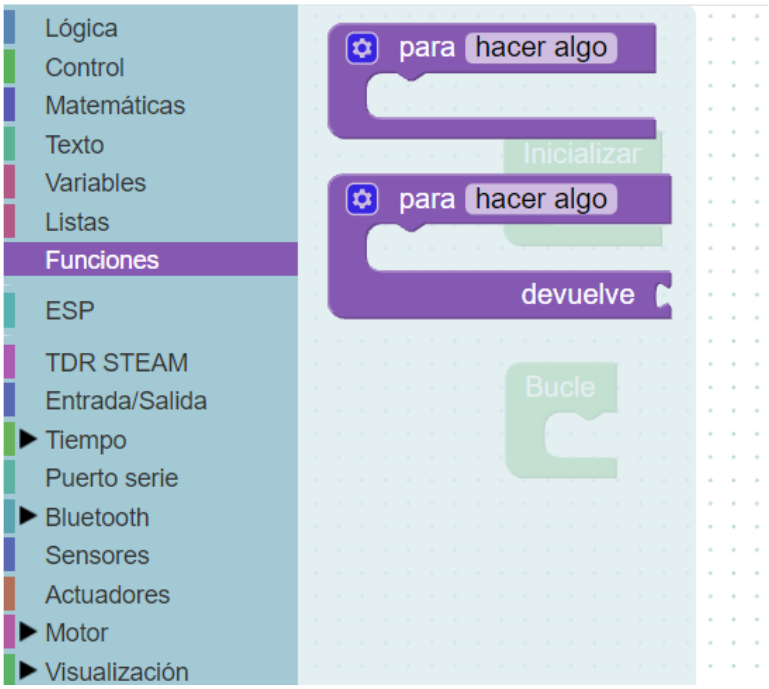
7.7.2 Reto A07.2. Alarma por exceso de temperatura.

Reto A07.1. Alarma por exceso de temperatura.

Ahora vamos a hacer una alarma sonora y acústica. El programa consistirá en hacer que el led rojo y el zumbador se accionen a la vez cuando el sensor de temperatura detecte una temperatura superior o igual a 23ºC.

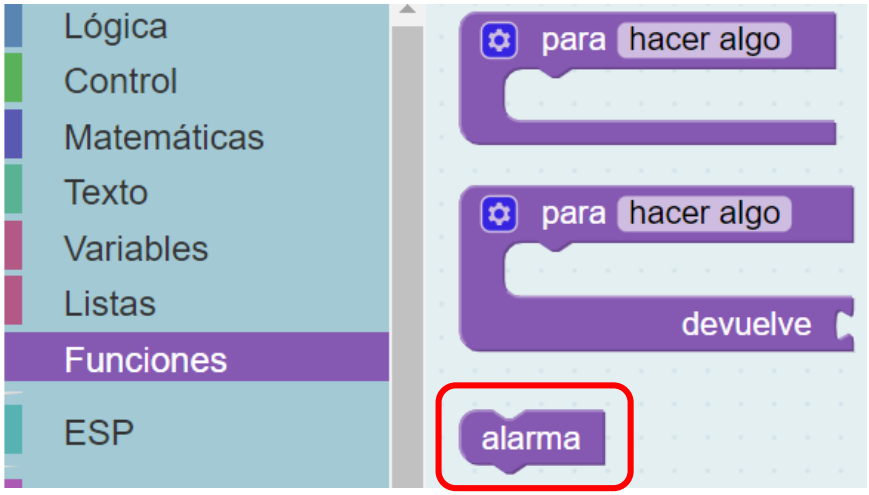
Para ello vamos a utilizar las *Funciones*. Las funciones permiten agrupar bloques de código. Esto es útil cuando un bloque de código se repite en varias partes del programa y así evitamos escribirlo varias veces o cuando queremos dividir el código de nuestro programa en bloques funcionales para realizar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

Las funciones nos permiten realizar tareas que se repiten a lo largo del programa. En el menú "Funciones" tenemos el bloque "*para (hacer algo)*". Lo utilizaremos para crear nuestra función como la siguiente imagen.



Debemos escribir el nombre de *alarma* dentro de la función. Al hacer esto automáticamente en el menú *Función* aparecerá un nuevo bloque llamado *alarma*. Cada vez que llamemos a esta función ejecutará el código que hay dentro de la función que hemos diseñado.

Podemos observar que ha aparecido un nuevo bloque dentro del apartado *Funciones*. El nuevo bloque se llama *alarma*, como podemos observar en la siguiente imagen:

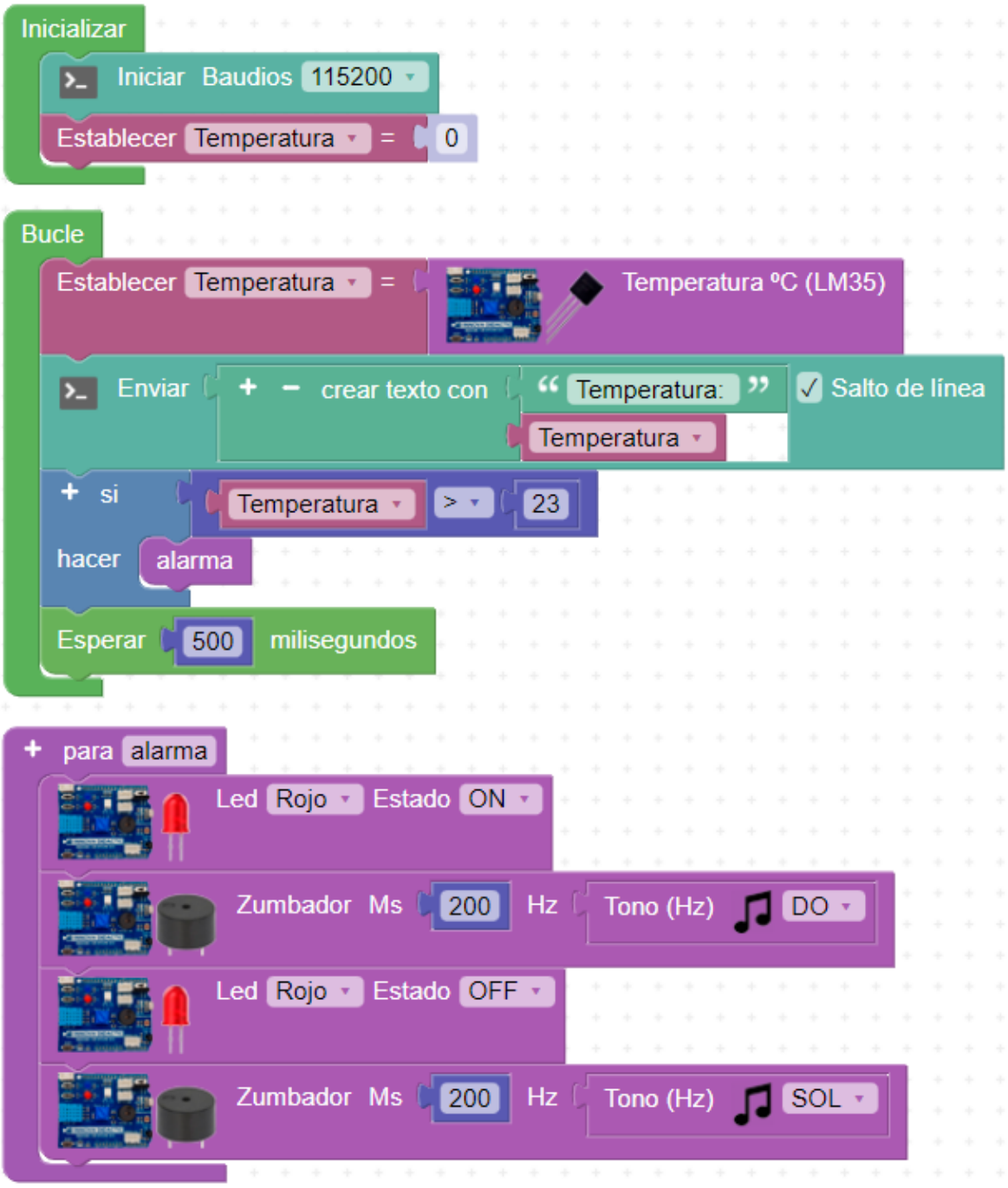


Añadiremos también la opción de poder ver los datos por la Consola. Para simular el programa también se puede utilizar el potenciómetro en % para simular que introducimos el parámetro de la temperatura.

Recordar que si variamos el potenciómetro deberemos volver a ponerlo a la izquierda para enviar el programa.

Enlace al programa: ArduinoBlocks Projects\alarma_exceso_temp.abp

El programa final sería el siguiente:



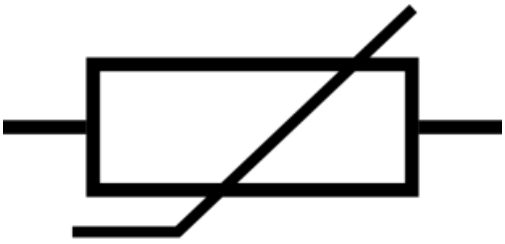
Actividad de ampliación: realiza un programa que varíe el programa anterior para que encienda de color verde el led RGB si la temperatura inferior a los 28ºC y rojo si la temperatura es superior a los 35ºC. En el rango intermedio se indicará de color azul.

7.8 Reto A08. Sensor de temperatura y humedad DHT11.

Reto A08. Sensor de temperatura y humedad DHT11.

En esta actividad vamos a leer los valores de temperatura y humedad utilizando el sensor DHT11. Este sensor mide temperaturas en un rango de acción de 0ºC a +50ºC con un error de +/- 2ºC y la humedad relativa entre 20% y 90% con un error de +/-5%. No es un sensor con una gran sensibilidad, pero cumple nuestros objetivos sobradamente.

El sensor de temperatura es un termistor tipo NTC. Un termistor es un tipo de resistencia (componente electrónico) cuyo valor varía en función de la temperatura de una forma más acusada que una resistencia común. Su funcionamiento se basa en la variación de la resistividad que presenta un semiconductor con la temperatura.

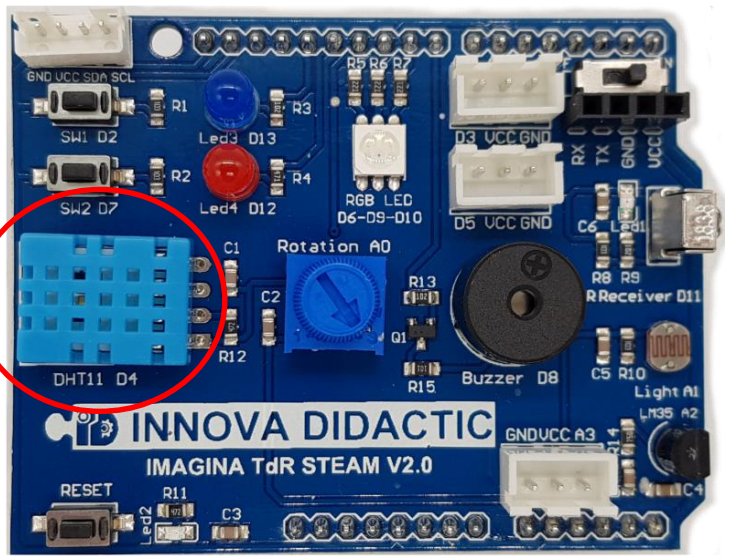


El término proviene del inglés "*thermistor*", el cual es un acrónimo de las palabras *Thermally Sensitive Resistor* (resistencia sensible a la temperatura). Existen dos tipos fundamentales de termistores:

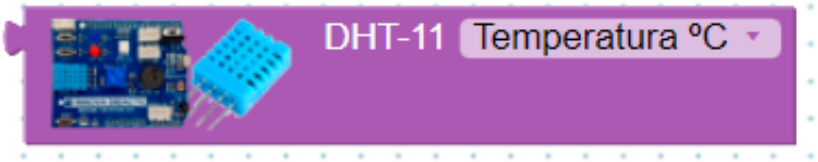
- Los que tienen un coeficiente de temperatura negativo (en inglés *Negative Temperature Coefficient* o NTC), los cuales decrementan su resistencia a medida que aumenta la temperatura.

- Los que tienen un coeficiente de temperatura positivo (en inglés *Positive Temperature Coefficient* o PTC), los cuales incrementan su resistencia a medida que aumenta la temperatura.

La placa Imagina TDR STEAM dispone de un sensor DHT11 conectado al pin D4. En un principio podríamos pensar que se trataría de un sensor analógico o que tuviera dos entradas, una para la temperatura y otra para la humedad. Pero las propias características de diseño del sensor, hace que se puedan realizar todas las lecturas por un solo puerto digital.



En ArduinoBlocks en el menú de *TDR STEAM* tenemos el bloque específico para programar este sensor:

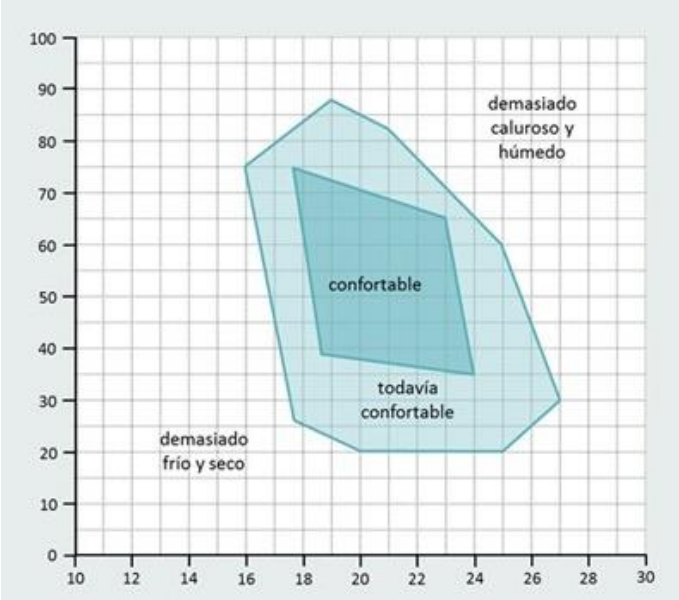


7.8.1 A08.1. Zona de confort con DHT11.

Reto A08.1. Zona de confort con DHT11.

Puede definirse *confort térmico*, o más propiamente *comodidad higrotérmica*, como la ausencia de malestar térmico. En fisiología, se dice que hay *confort higrotérmico* cuando no tienen que intervenir los mecanismos termorreguladores del cuerpo para una actividad sedentaria y con una indumentaria ligera. Esta situación puede registrarse mediante índices que no deben ser sobrepasados para que no se pongan en funcionamiento los sistemas termorreguladores (metabolismo, sudoración y otros).

Según la imagen adjunta vamos a marcar unos puntos de temperatura y humedad en los que estaremos dentro de la zona de confort térmico, dentro de una zona de medio confort y fuera de la zona de confort.



Usando el Led RGB vamos a indicar esas zonas:

- Led RGB en ROJO; fuera de la zona de confort.
- Led RGB en VERDE; en la zona de confort.

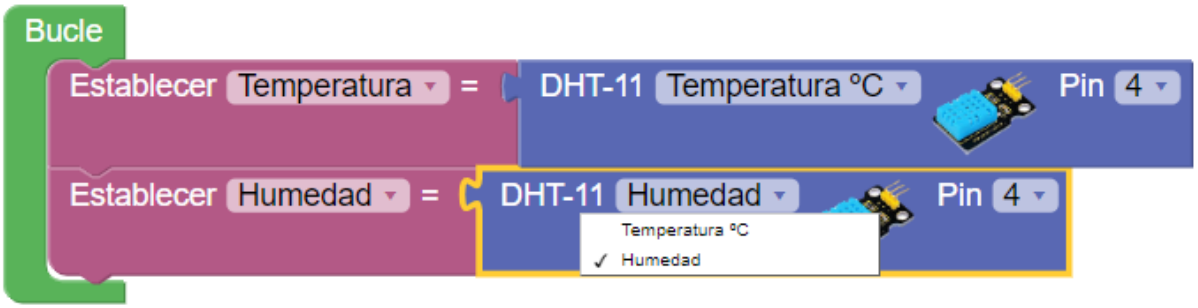
- *Zona VERDE:*
  - Humedad entre 40% y el 65%.
  - Temperatura entre 18ºC y 24ºC.

El led brillará en VERDE dentro de los parámetros de la Zona VERDE, para el resto el led estará parpadeando en color ROJO.

Para realizar este programa necesitaremos varios de los bloques del menú de Lógica. Necesitaremos evaluar una *condición Lógica* y utilizar *conjunciones y disyunciones.*

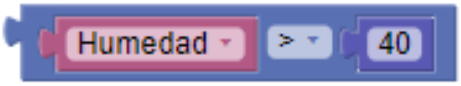
- *Y*: se cumple si los dos operandos son verdaderos.
- *O*: se cumple si alguno de los dos operandos es verdadero.

Antes de evaluar las condiciones debemos establecer las dos variables; la variable *Temperatura* y la variable *Humedad*. Recuerda que en el menú desplegable del sensor DHT-11 debes elegir la Temperatura o la Humedad y que está conectado al pin D4.



Usando 3 bloques de conjunciones debes crear el siguiente bloque:

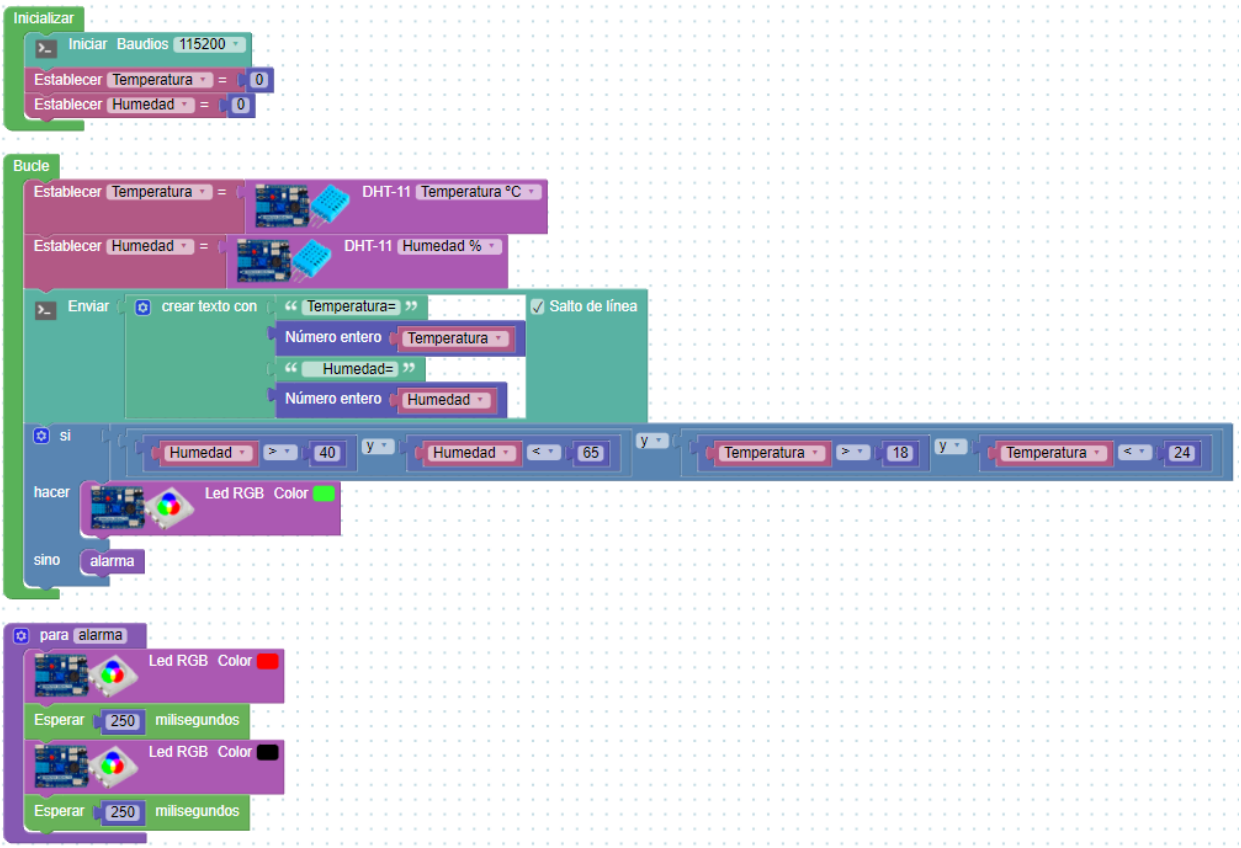
Después ir metiendo las condiciones en cada uno de ellos:



Y une todo hasta conseguir esta condición final:

Por último, crea una función, que la puedes llamar *alarma*. Observa cómo para poder dar el color negro al led RGB es apagarlo. También mostraremos los valores por la *Consola*, añadiendo los bloques necesarios.

Y con todo esto, el programa final quedaría así:



Enlace al programa: ArduinoBlocks Projects\zona_confort.abp

Actividad de ampliación: realiza un programa con las tres zonas por colores. Ten cuidado con las zonas donde se unen los rangos, ya que si el valor es justo (por ejemplo 40) deberás poner el símbolo >= o <=. También puedes mostrar los valores de humedad y temperatura por el puerto serie.

7.9 Reto A09. Receptor de infrarrojos (IR).

Reto A08. Receptor de infrarrojos (IR).

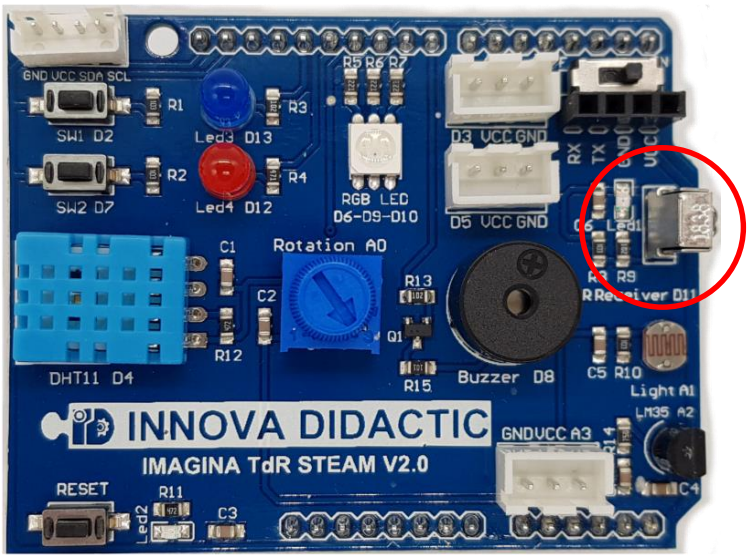
Una gran parte de los electrodomésticos utilizan mandos a distancia de infrarrojos, como los televisores o equipos musicales. El sensor infrarrojo es un dispositivo optoelectrónico capaz de medir la radiación electromagnética infrarroja de los cuerpos en su campo de visión. Todos los cuerpos emiten una cierta cantidad de radiación, esta resulta invisible para nuestros ojos, pero no para estos aparatos electrónicos, ya que se encuentran en el rango del espectro justo por debajo de la luz visible.

En el caso del receptor de infrarrojos (IR) de la placa IMAGINA TDR STEAM permite codificar los protocolos de señales de pulsos infrarrojos utilizados por los mandos a distancia. Los protocolos detectados son los siguientes: RC5, RC6, NEC, SONY, PANASONIC, JVC, SAMSUNG, WHYNTER, AIWA, LG, SANYO, MITSUBISHI y DENON. Es decir, detectaría cualquier señal emitida por uno de esos mandos.

El mando a distancia contiene un circuito interno, un procesador y un led (*Light Emitting Diode*) que emite la señal infrarroja.

La señal infrarroja transmite el código correspondiente al botón del mando a distancia pulsado y lo transmite al dispositivo en forma de una serie de impulsos de luz infrarroja. Pensemos en el código Morse, que ya vimos en una de las prácticas anteriores, y sus tonos cortos y largos. De forma análoga, los pulsos de luz infrarroja transmitidos son de dos tipos, los llamados 0 y 1. Los 0 podrían verse como los tonos cortos y los 1 como los tonos largos. El receptor IR recibe la serie de impulsos de infrarrojos y los pasa a un procesador que descodifica la serie de 0 y 1 en los bits digitales para después realizar la función que programemos.

En la placa *IMAGINA TDR STEAM* el sensor receptor IR se encuentra conectado al pin digital D11.



En el menú de *TDR STEAM* de ArduinoBlocks tenemos dos bloques para programar nuestro receptor IR, uno propio del sensor y otro para el mando.

7.9.1 Reto A09.1. Recepción de comandos por infrarrojos.

Reto A09.1. Recepción de comandos por infrarrojos.

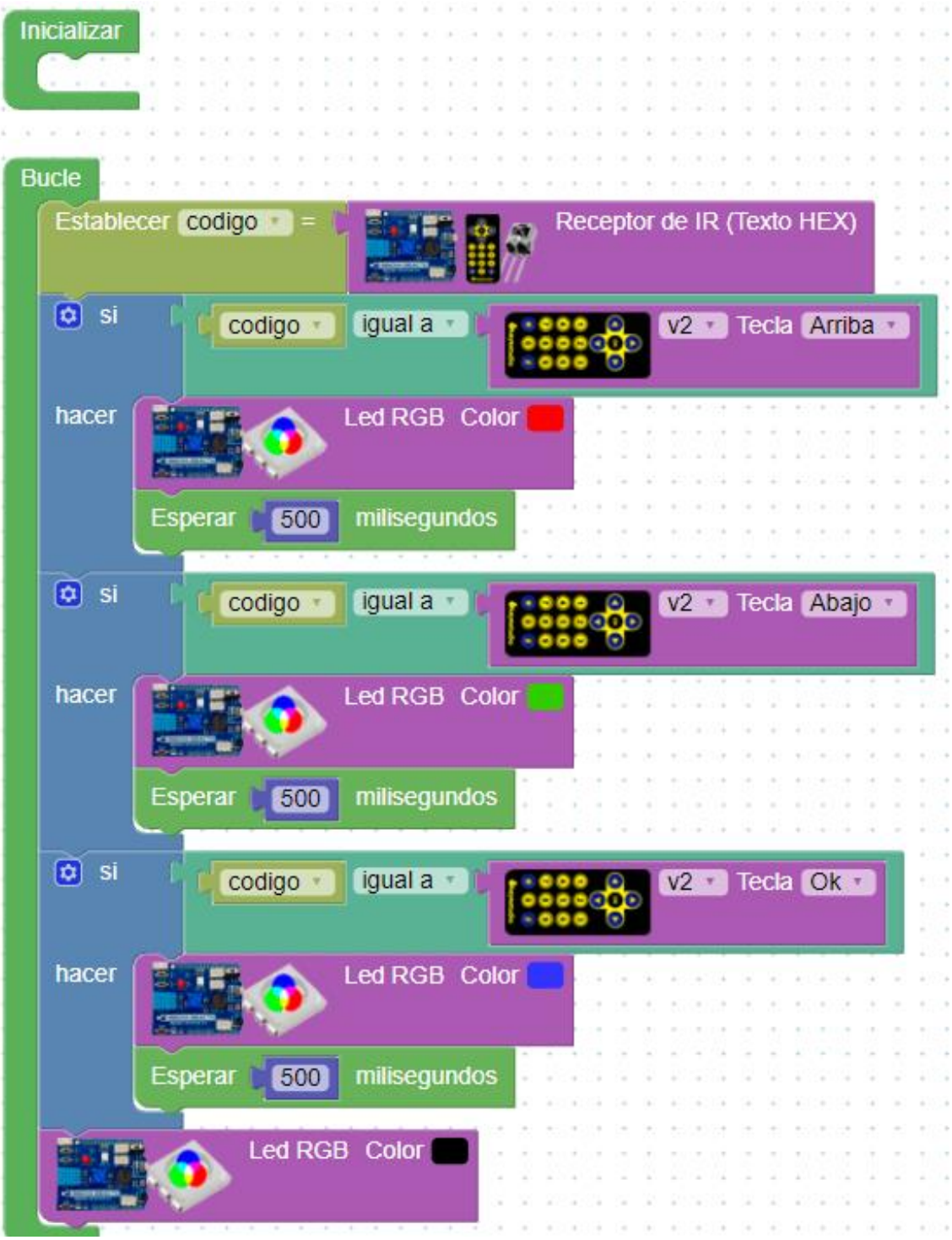
Vamos a realizar una pequeña actividad en la que utilicemos el receptor IR y el mando emisor de Keyestudio. Primero crearemos una variable de tipo texto a la que llamaremos *codigo* (*sin acento, en programación se debe evitar poner acentos y símbolos*) y la estableceremos con el bloque del Receptor IR:

A continuación, estableceremos la siguiente igualdad y lo haremos como condición para encender el led RGB de color rojo. La condición la realizaremos con la igualdad que hay dentro de los bloques de *Texto*.

Podemos completar el programa dando distintas acciones a distintos botones.

- La flecha hacia arriba se encenderá el color rojo.
- La flecha hacia abajo se encenderá el color verde.
- La tecla OK se encenderá el color azul.
- Si no se pulsa ninguna tecla estará apagado el led.

El programa resultante será el siguiente:



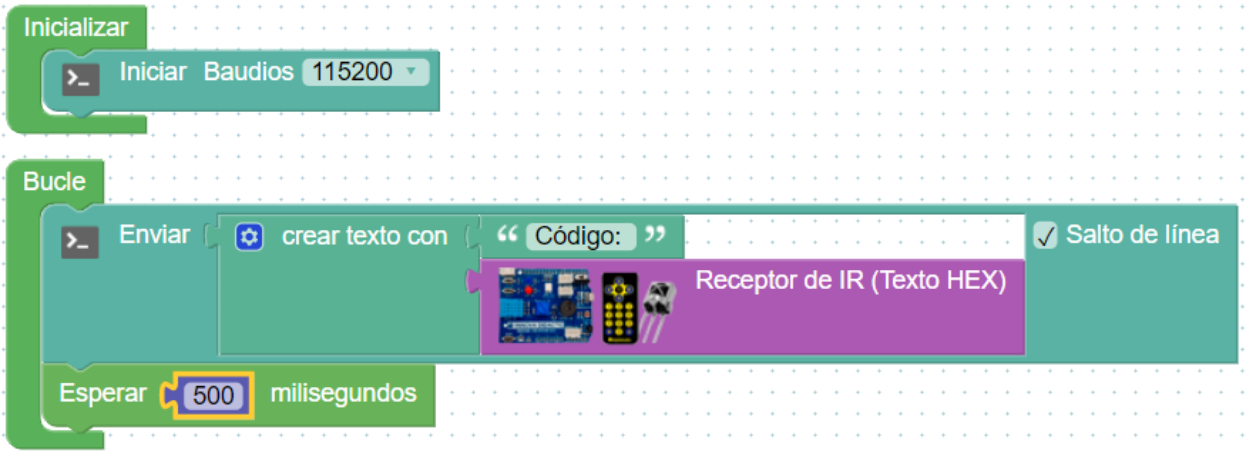
Enlace al programa: ArduinoBlocks Projects\mando_1.abp

Actividad de ampliación: realiza un programa con el que puedas ver qué códigos envía el mando y muéstralos por el puerto serie.

7.9.2 Reto A09.2. Mando universal.

Reto A09.2. Mando universal.

Vamos a realizar un programa para poder utilizar cualquier emisor de infrarrojos (mando o móvil) con nuestra placa ***Imagina TRD STEAM***. Primero realizaremos un sencillo programa para leer los códigos que nos envía en dispositivo a la placa. Leeremos estos códigos en la *Consola*.

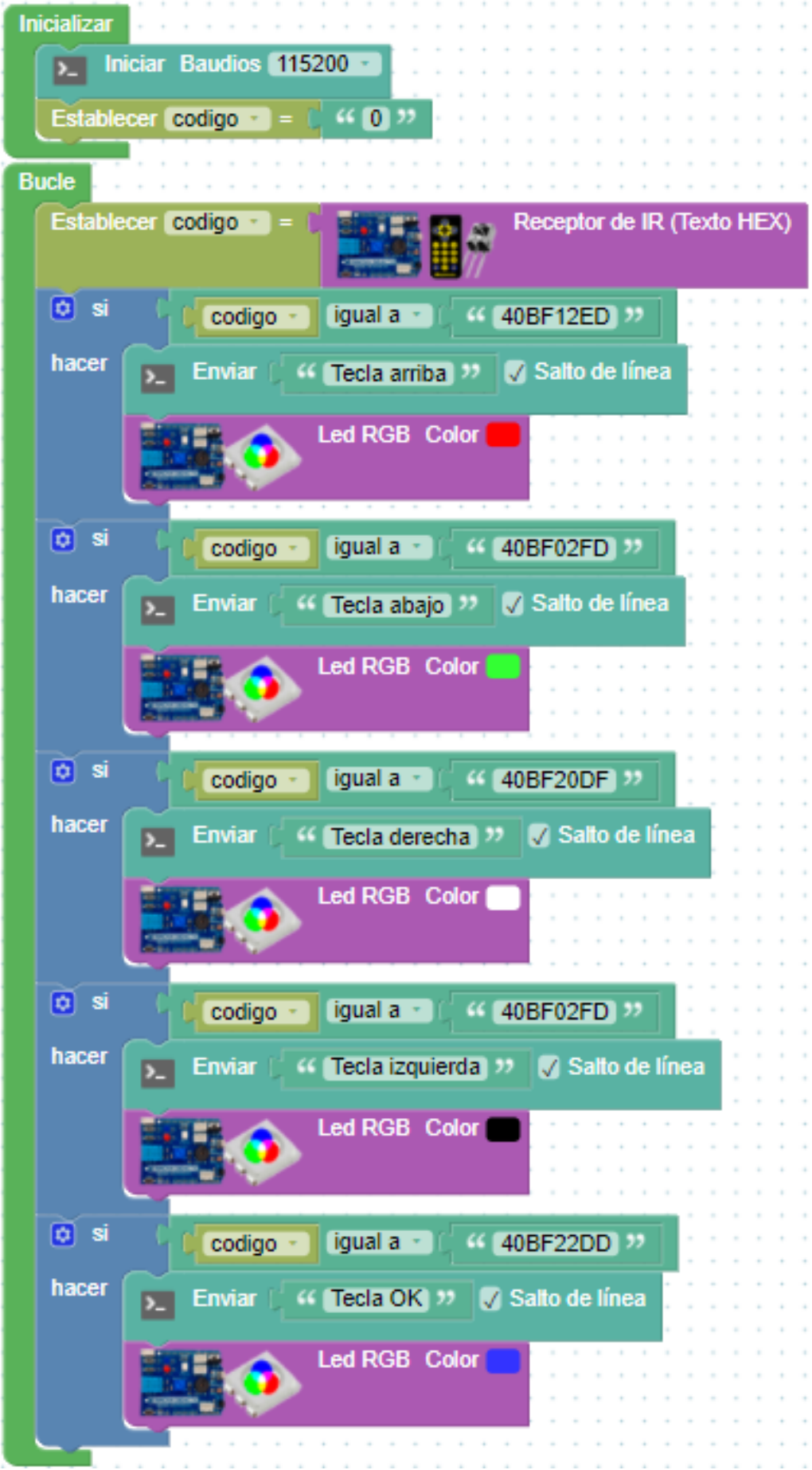


Realizaremos una tabla con los valores que devuelve el mando al apretar las diferentes teclas. En nuestro caso de ejemplo son estos:

Tecla	Código
Arriba	40BF12ED
Abajo	40BF02FD
Derecha	40BF20DF
Izquierda	40BF02FD
OK	40BF22DD

Ahora ya podemos realizar un nuevo programa que identifique estos códigos para poder realizar las funciones que queramos. Identificaremos la tecla pulsada y encenderemos el led RGB de colores diferentes.

Enlace al programa: ArduinoBlocks Projects\mando_2.abp

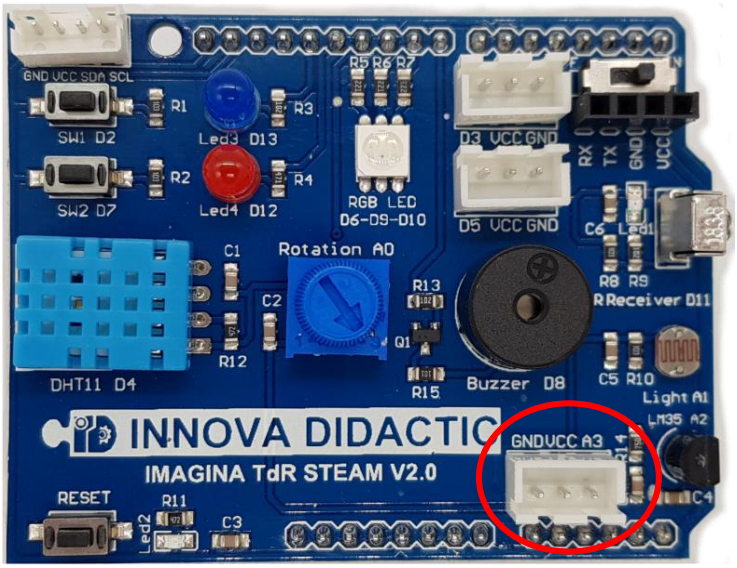

Inicializar
Iniciar Baudios 115200
Establecer codigo = " 0 "
Bucle
Establecer codigo =
Receptor de IR (Texto HEX)
si
codigo igual a " 40BF12ED "
hacer
Enviar " Tecla arriba " Salto de línea
Led RGB Color
si
codigo igual a " 40BF02FD "
hacer
Enviar " Tecla abajo " Salto de línea
Led RGB Color
si
codigo igual a " 40BF20DF "
hacer
Enviar " Tecla derecha " Salto de línea
Led RGB Color
si
codigo igual a " 40BF02FD "
hacer
Enviar " Tecla izquierda " Salto de línea
Led RGB Color
si
codigo igual a " 40BF22DD "
hacer
Enviar " Tecla OK " Salto de línea
Led RGB Color

7.10 Reto A10. El micrófono.

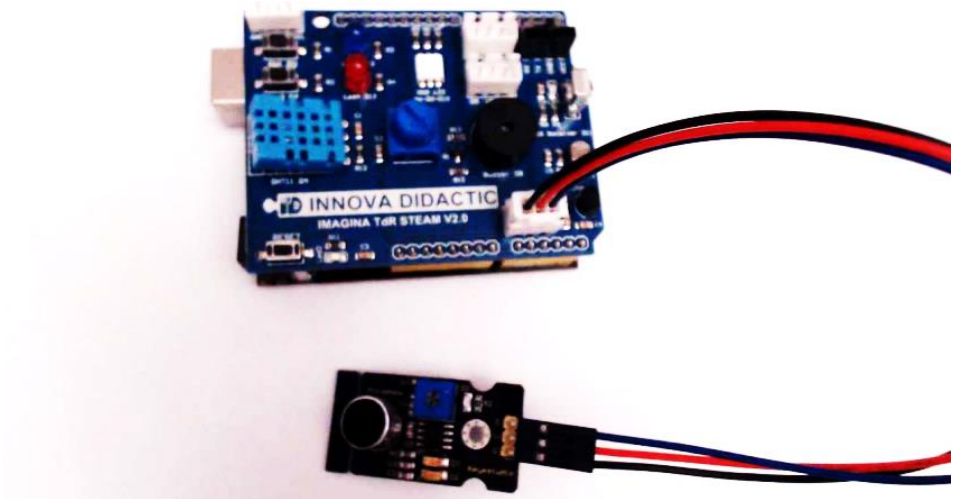
Reto A10. El micrófono.

El micrófono es un dispositivo que convierte las variaciones de presión en una señal eléctrica. El micrófono que vamos a utilizar debemos conectarlo a una entrada analógica. Como tenemos libre la entrada A3, utilizaremos esta entrada.

Conectaremos los cables respetando los colores como se muestra en la tabla siguiente:



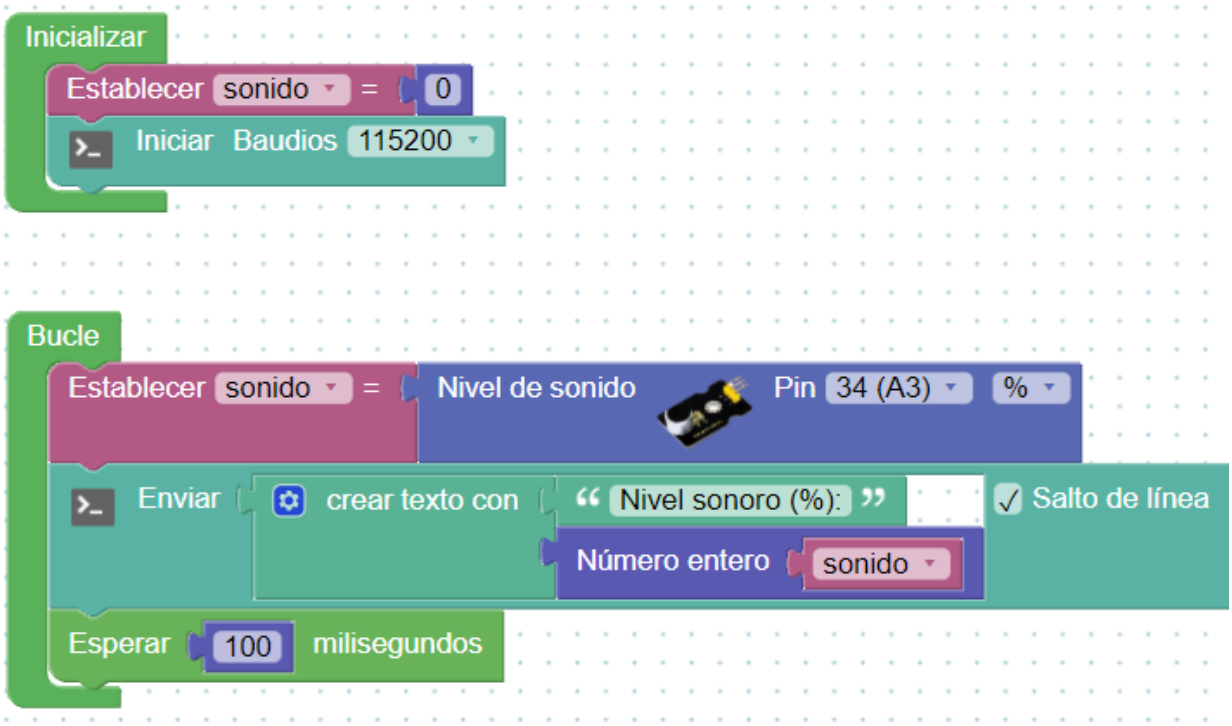
Imagina TDR STEAM	*Color Cable*	*Micrófono*
GND	negro	G (GND)
VCC	rojo	V (VCC)
A3	azul	S (Señal)



7.10.1 Reto A10.1. Nivel de sonido con el micrófono.

Reto A10.1. Nivel de sonido con el micrófono.

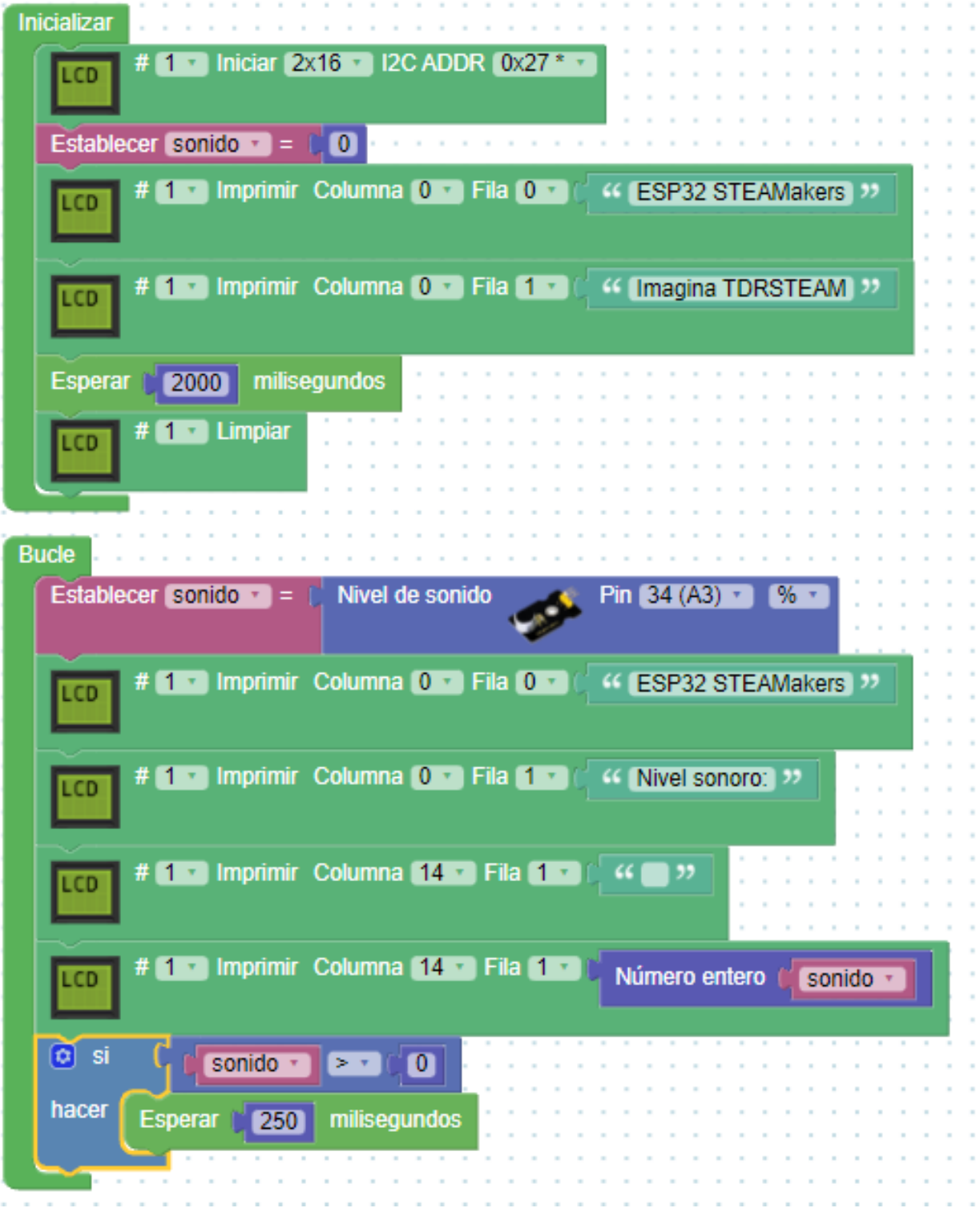
Vamos a realizar un sencillo programa en el que veamos por el puerto serie el nivel de sonido en porcentaje.



Vamos a hacer un nuevo programa, mostrando los datos por la pantalla LCD. Primero configuraremos la pantalla, crearemos una variable para almacenar el nivel de sonido y mostraremos un mensaje. En el programa principal leeremos el nivel de sonido almacenándolo en una variable y mostraremos los datos por la pantalla. Para borrar el valor del nivel sonoro en la pantalla LCD introduciremos un texto formado por dos espacios. Por último, si el valor es mayor de 0 realizará una pequeña espera de tiempo para poder visualizar mejor el valor por la pantalla.

La explicación de cómo conectar la pantalla LCD está explicada en el apartado de comunicaciones I2C.

El programa resultante será el siguiente:

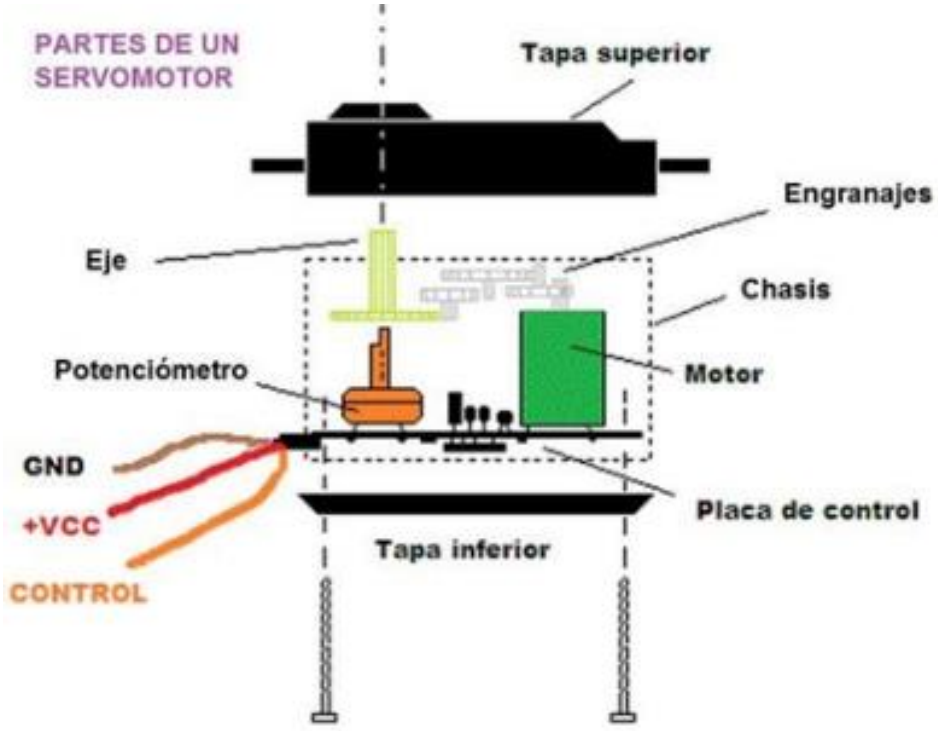


Actividad de ampliación: modifica el programa para que muestre los datos tanto por la pantalla LCD como por el puerto serie.

7.11 Reto A11. El servomotor.

Reto A11. El servomotor.

El servomotor es un tipo de motor muy utilizado en robótica. Es un tipo de motor que puede posicionar su eje en un ángulo determinado. Para realizar esto, consta de un motor acoplado a un grupo de engranajes y una pequeña placa de control con un potenciómetro para saber su posición. Para funcionar necesita alimentación (VCC y GND) y una señal de control. Generalmente, los servomotores giran unos 270º pero hay algunos tipos que permiten ángulos diferentes e, incluso, de rotación continua.



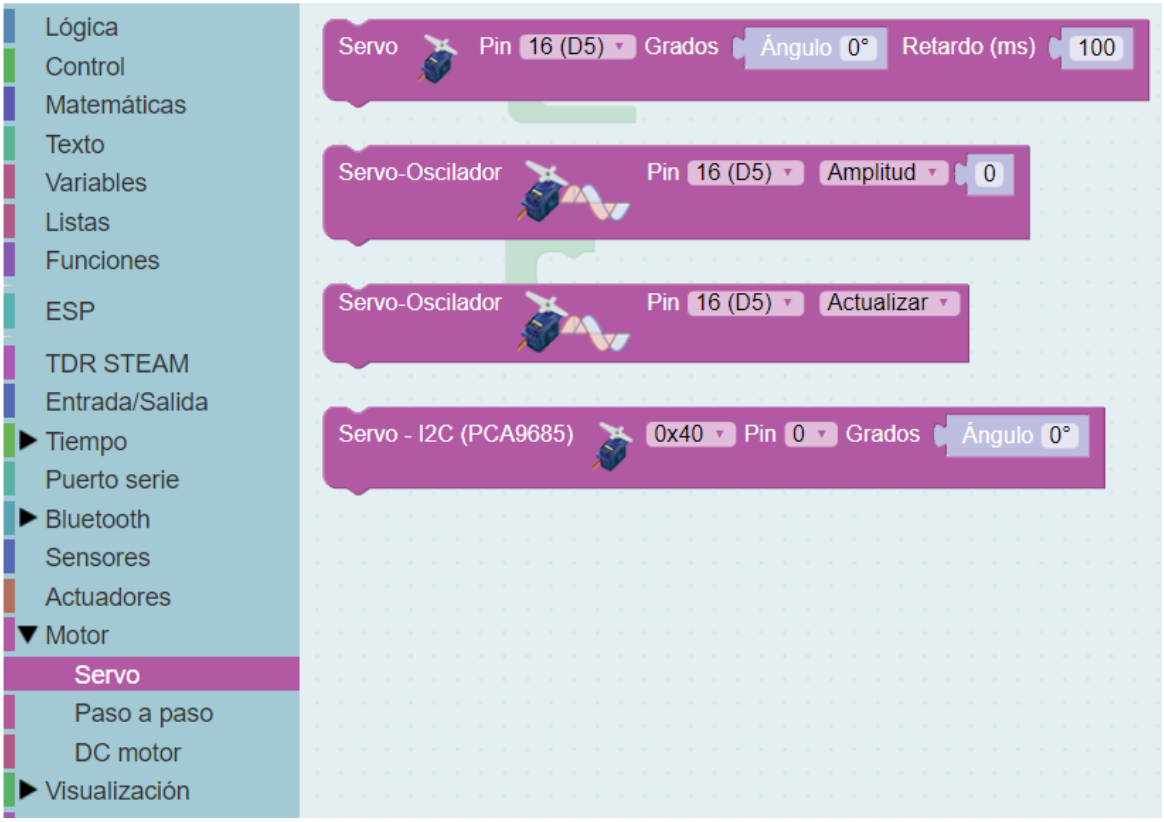
Para utilizar el servomotor utilizaremos una de las salidas que nos quedan libres en la placa ***Imagina TDR STEAM***. Los cables del motor los conectaremos respetando las indicaciones marcadas con los colores:

- Marrón – GND
- Rojo – VCC
- Naranja – Señal (D3)

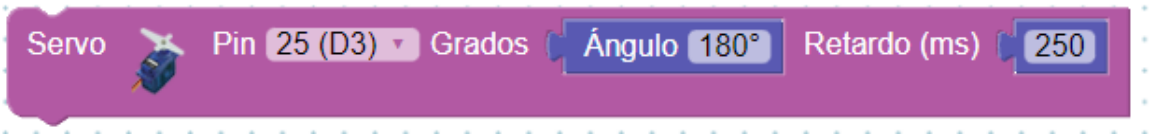
7.11.1 Reto A11.1. Indicador de posición.

Reto A11.1. Indicador de posición.

Vamos a realizar un programa que haga mover el motor para que indique la posición del eje del motor. En ArduinoBlocks tenemos boques específicos para controlar este tipo de motores.

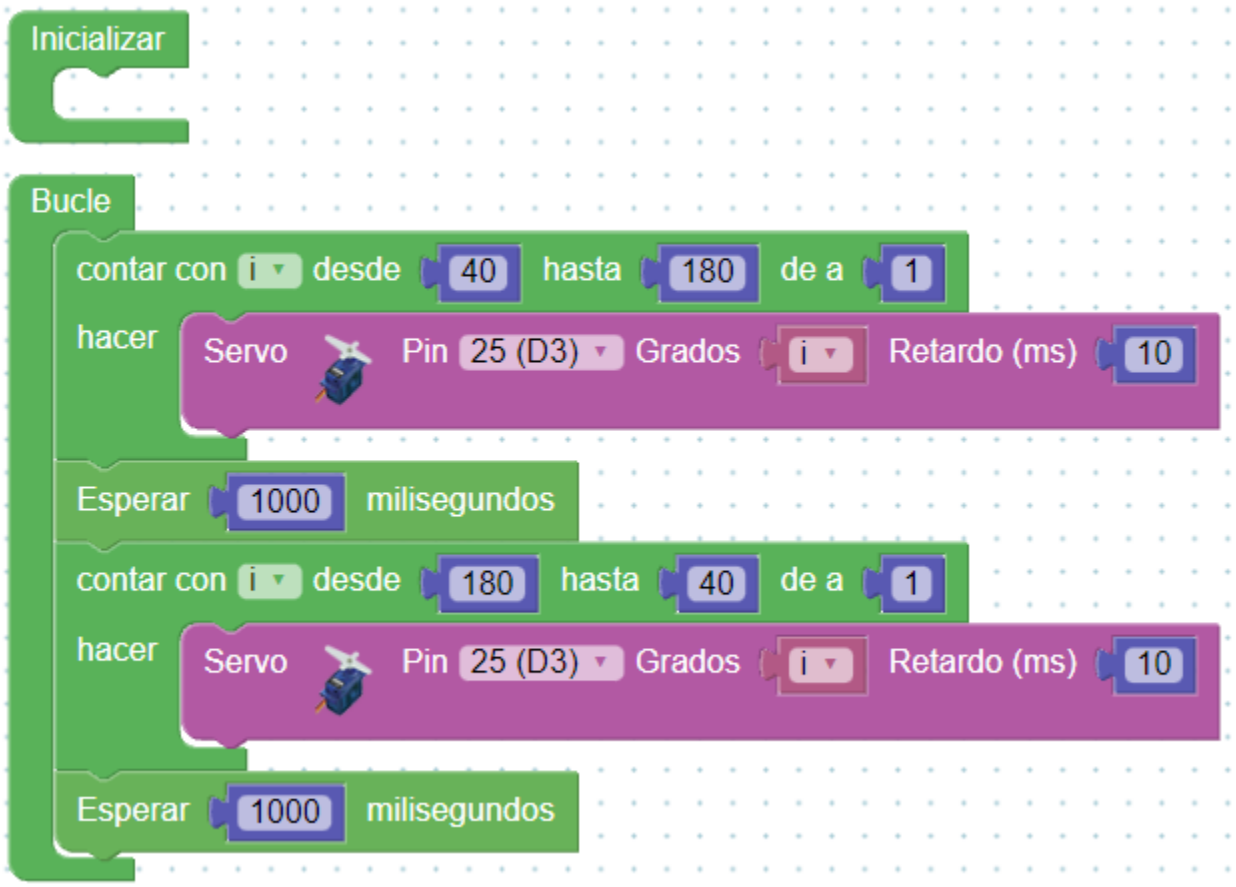


Utilizaremos en primer tipo de bloque. Aquí podemos configurar donde vamos a conectar el motor, el ángulo de giro y la velocidad del movimiento.



Vamos a crear el programa para que gire el motor en un sentido y después en el sentido contrario.

El programa que vamos a hacer es el siguiente:



Enlace al programa: ArduinoBlocks Projects\servomotor.abp

Actividad de ampliación: modifica el programa para que gire entre 30 y 200 grados con una velocidad de pasos de 2ms por paso.

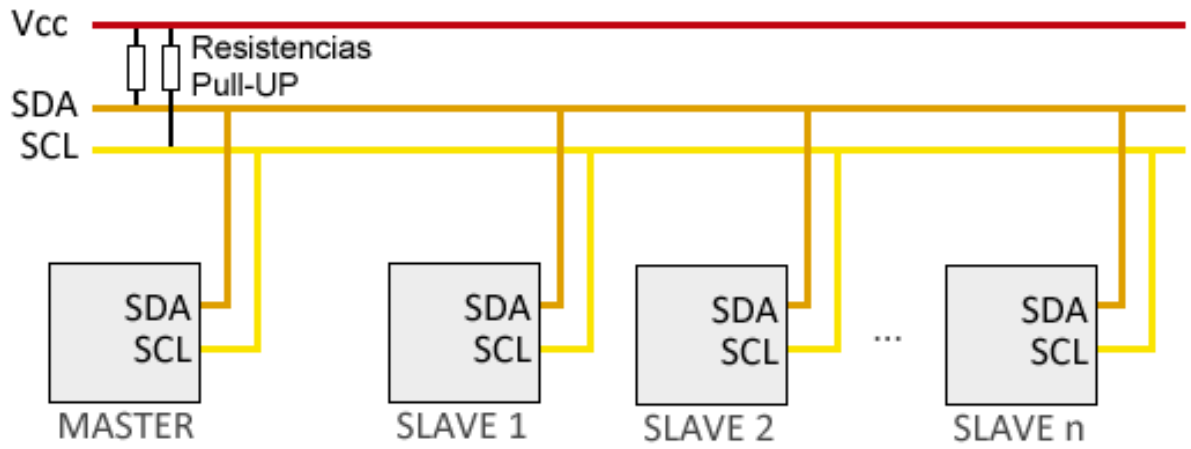
7.12 Reto A12. Puertos de expansión I2C.

Reto A12. Puertos de expansión I2C.

El estándar I2C (Inter-Integrated Circuit) fue desarrollado por Philips en 1982 para la comunicación interna de dispositivos electrónicos en sus artículos. Posteriormente fue adoptado progresivamente por otros fabricantes hasta convertirse en un estándar del mercado.

I2C también se denomina TWI (Two Wired Interface) únicamente por motivos de licencia. No obstante, la patente caducó en 2006, por lo que actualmente no hay restricción sobre el uso del término I2C. Así que es exactamente lo mismo, pero se utiliza más el término I2C (I^2C).

El bus I2C requiere únicamente dos cables para su funcionamiento, uno para la señal de reloj (SCL) y otro para el envío de datos (SDA), lo cual es una ventaja frente al bus SPI. Por contra, su funcionamiento es un poco más complejo, así como la electrónica necesaria para implementarla.

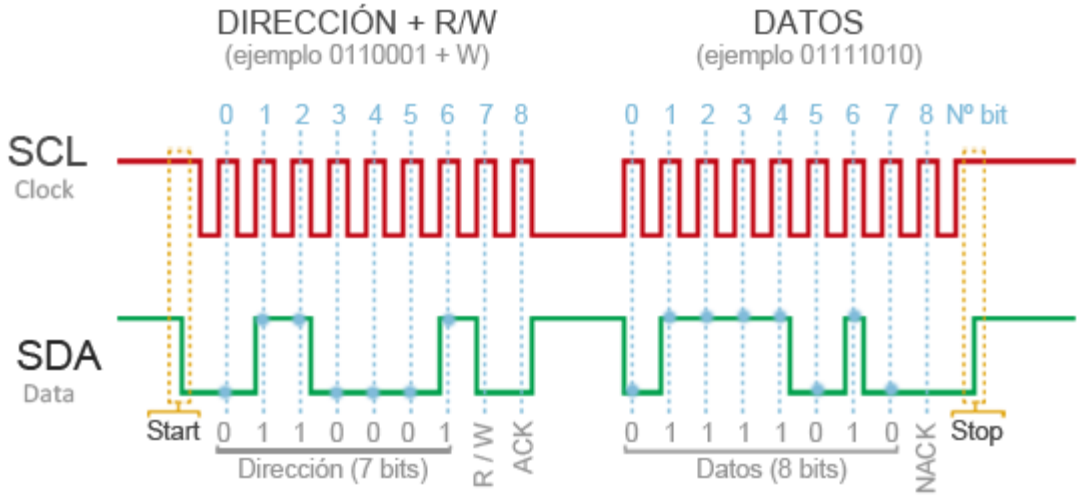


En el bus, cada dispositivo dispone de una dirección, que se emplea para acceder a los dispositivos de forma individual. Esta dirección puede ser fijada por hardware (en cuyo caso, frecuentemente, se pueden modificar los últimos 3 bits mediante *jumpers* o interruptores, o por software.

En general, cada dispositivo conectado al bus debe tener una dirección única. Si tenemos varios dispositivos similares tendremos que cambiar la dirección o, en caso de no ser posible, implementar un bus secundario.

El bus I2C tiene una arquitectura de tipo maestro-esclavo. El dispositivo maestro inicia la comunicación con los esclavos, y puede mandar o recibir datos de los esclavos. Los esclavos no pueden iniciar la comunicación (el maestro tiene que preguntarles), ni hablar entre si directamente.

El bus I2C es síncrono. El maestro proporciona una señal de reloj, que mantiene sincronizados a todos los dispositivos del bus. De esta forma, se elimina la necesidad de que cada dispositivo tenga su propio reloj, de tener que acordar una velocidad de transmisión y mecanismos para mantener la transmisión sincronizada (como en UART)



A este bus de comunicaciones se le pueden conectar múltiples dispositivos:

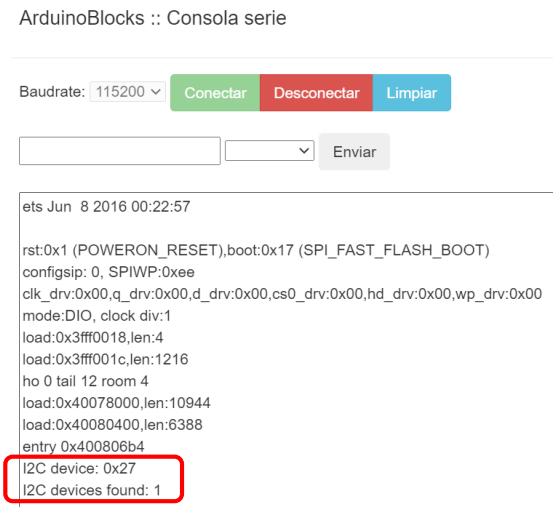
- Pantalla LCD.
- Pantalla OLED.
- Matriz de leds.
- Acelerómetros.
- Giroscopios.
- Sensores de temperatura.
- Controladores de servomotores.
- Sensor de color.
- RFID.
- Sensor de movimientos.
- Etc.

Podemos identificar de forma muy sencilla la dirección I2C de los dispositivos que tengamos conectados a la placa ***ESP Plus STEAMakers***.

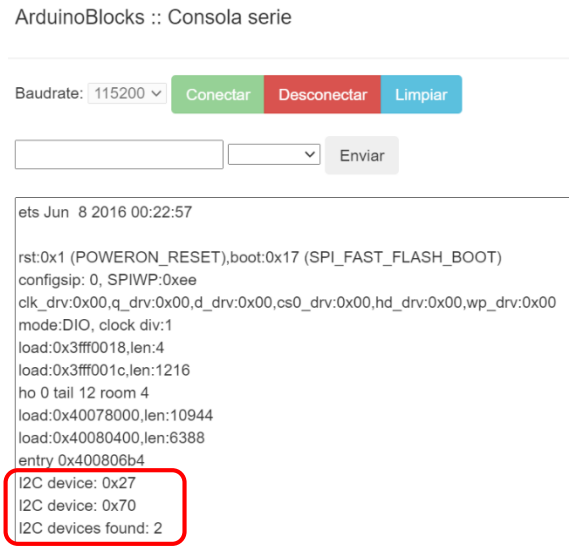
Enlace al programa: ArduinoBlocks Projects\i2c.abp

Vamos a realizar un programa para leer la dirección I2C de una pantalla LCD.

Ahora podemos abrir la *Consola* para ver la dirección del dispositivo.



Conectaremos también una matriz de leds 8x8. Veremos dos dispositivos con las direcciones 0x27 (pantalla LCD) y 0x70 (matrix de leds 8x8).



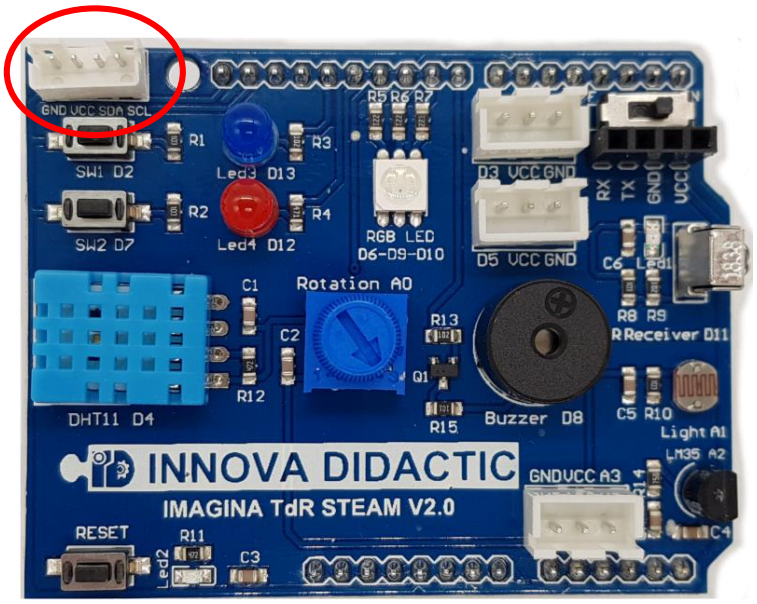
A continuación, vamos a trabajar con algunos de estos dispositivos I2C.

7.12.1 Reto A12.1. Pantalla LCD 16x2.

Reto A12.1. Pantalla LCD 16x2.

Vamos a realizar la conexión de una pantalla LCD (16x2). La pantalla LCD utilizada es una pantalla de 16 caracteres (por fila) y dos columnas. Esta pantalla tiene 4 conexiones, dos cables (SDA y SCL para el bus de comunicaciones I2C) y los dos cables de alimentación (VCC y GND).

Conectaremos la pantalla LCD a la placa *Imagina TDR STEAM* en el conector indicado mediante un cable con el conector pertinente.

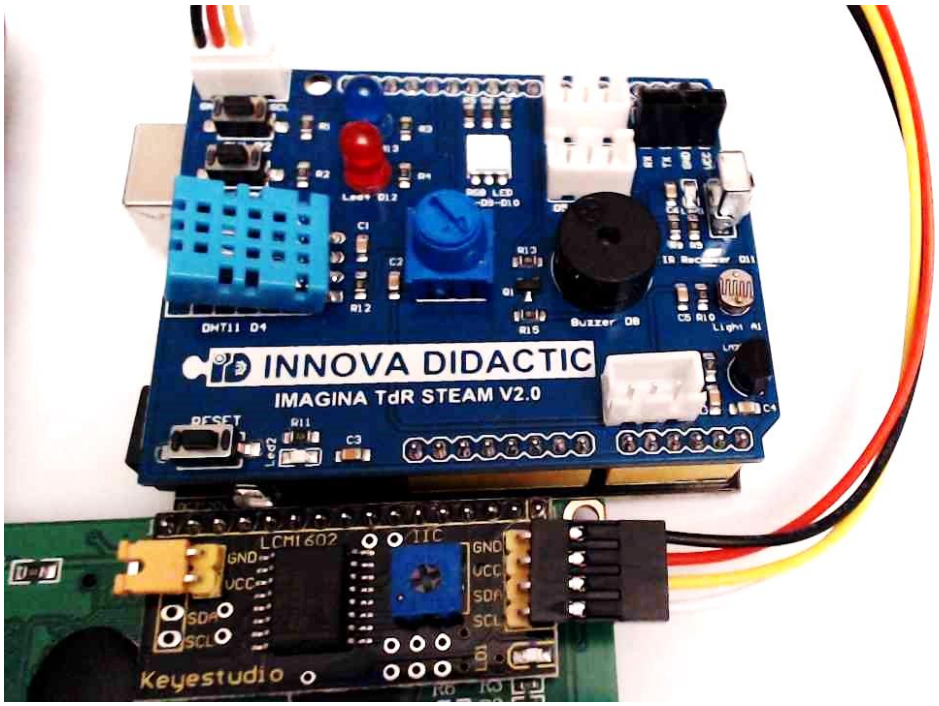


Hemos de tener cuidado y respetar las conexiones, tal y como se indica en la siguiente tabla (el color del cable es una sugerencia):

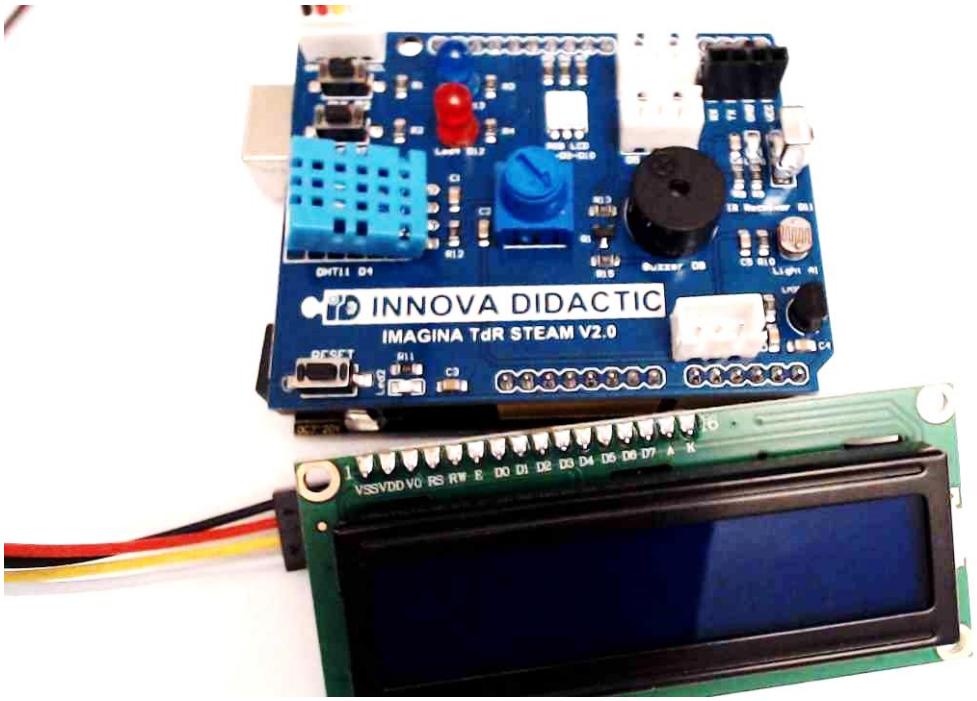
Imagina TdR STEAM	***Color Cable***	***Pantalla LCD***
GND	Negro	GND
VCC	Rojo	VCC
SDA	Amarillo	SDA
SCL	Blanco	SCL

Mediante I2C podremos conectar hasta 8 pantallas LCD simultáneamente y programarlas con ArduinoBlocks.

Conexión de los cables a la pantalla y a la placa Imagina TDR STEAM.

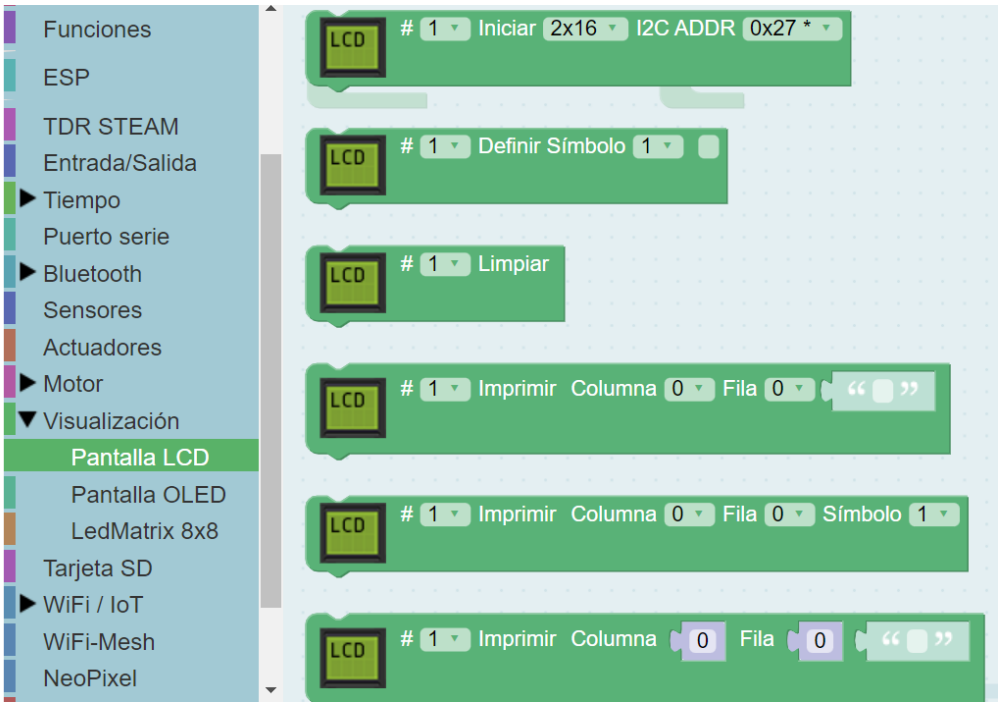


En la pantalla aparecerán correctamente los datos cuando la coloquemos es esta posición, sino se verán al revés:

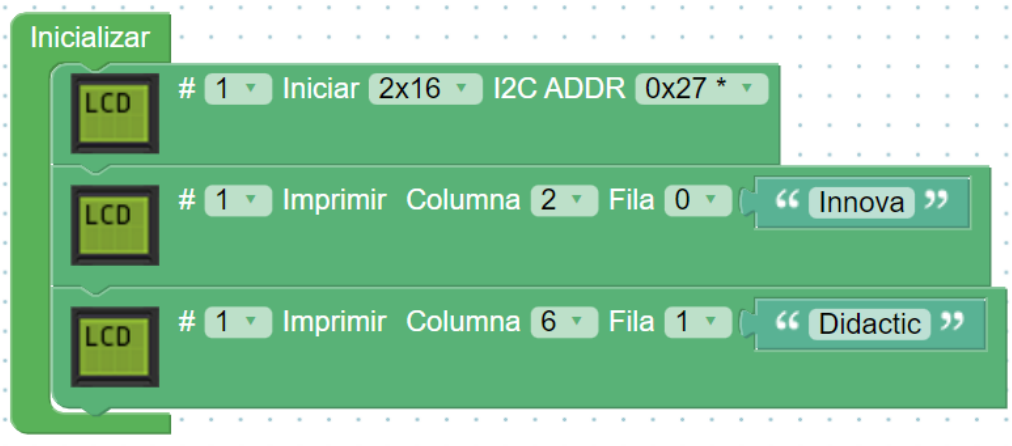


En la configuración de la pantalla hemos de seleccionar la dirección de configuración 0x27.

En ArduinoBlocks hay una serie de funciones específicas para el control de este tipo de pantallas. Las funciones están dentro del menú *Visualización*.



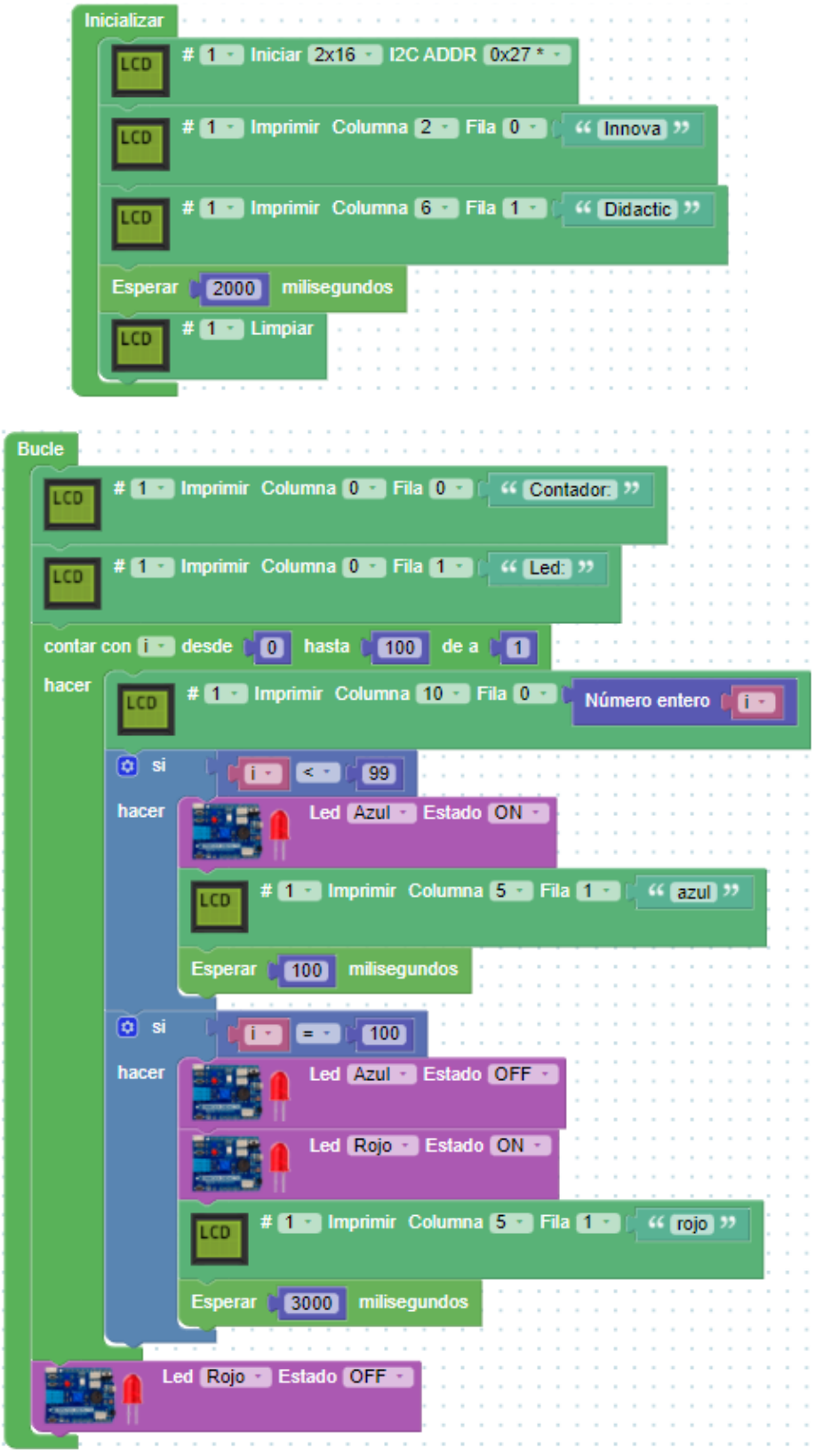
Vamos a realizar un programa que muestre una información inicial por la pantalla LCD.



Después realizaremos un control con el led rojo y azul. Haremos un contador de 0 a 99 y el led azul estará encendido y cuando este valor sea 100 encienda el led rojo durante 3 segundos y vuelva a reiniciar el ciclo. Mostrará el valor de contaje por la pantalla LCD y que led está encendido.

Enlace al programa: ArduinoBlocks Projects\lcd.abp

El programa resultante será el siguiente:

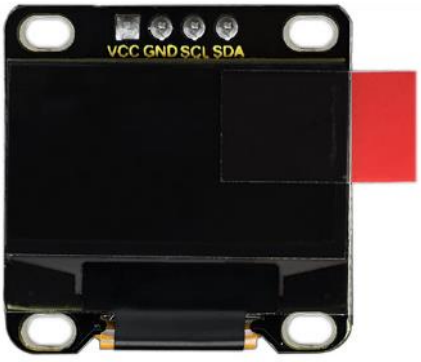


Actividad de ampliación: modifica el programa para que muestre otra información al inicializar y que el contador cuente hasta 200 y los incrementos sean de dos en dos.

7.12.2 Reto A12.2. Pantalla OLED.

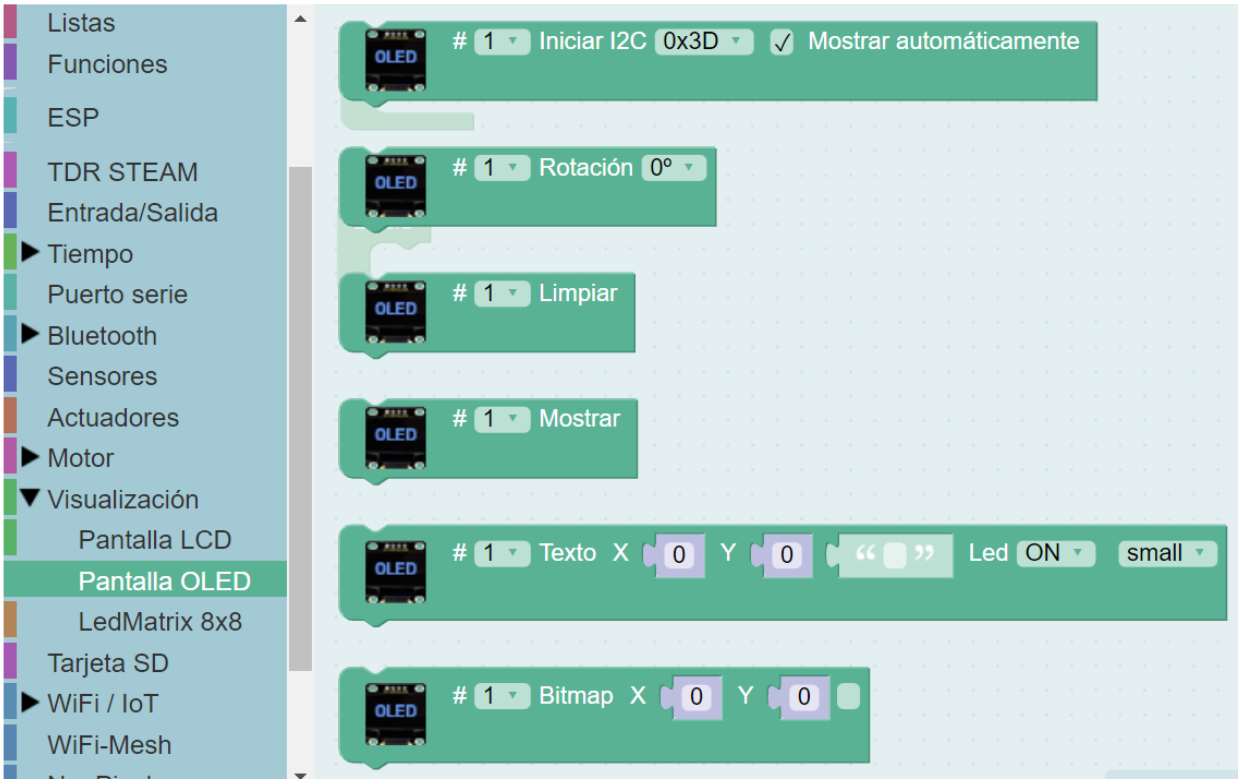
Reto A12.2. Pantalla OLED.

La pantalla OLED es una pantalla de 0,96” (128x64) que funciona mediante bus de comunicaciones I2C. Como solamente está encendido el píxel elegido es un tipo de pantalla muy eficiente. Es una pantalla muy útil para mostrar información o imágenes.



Conectaremos la pantalla al conector I2C de la placa **Imagina TDR STEAM**. También se puede conectar directamente a la placa ***ESP32 Plus STEAMakers*** en el conector I2C hembra que tiene.

ArduinoBlocks dispone de bloques específicos para poder controlar esta pantalla.

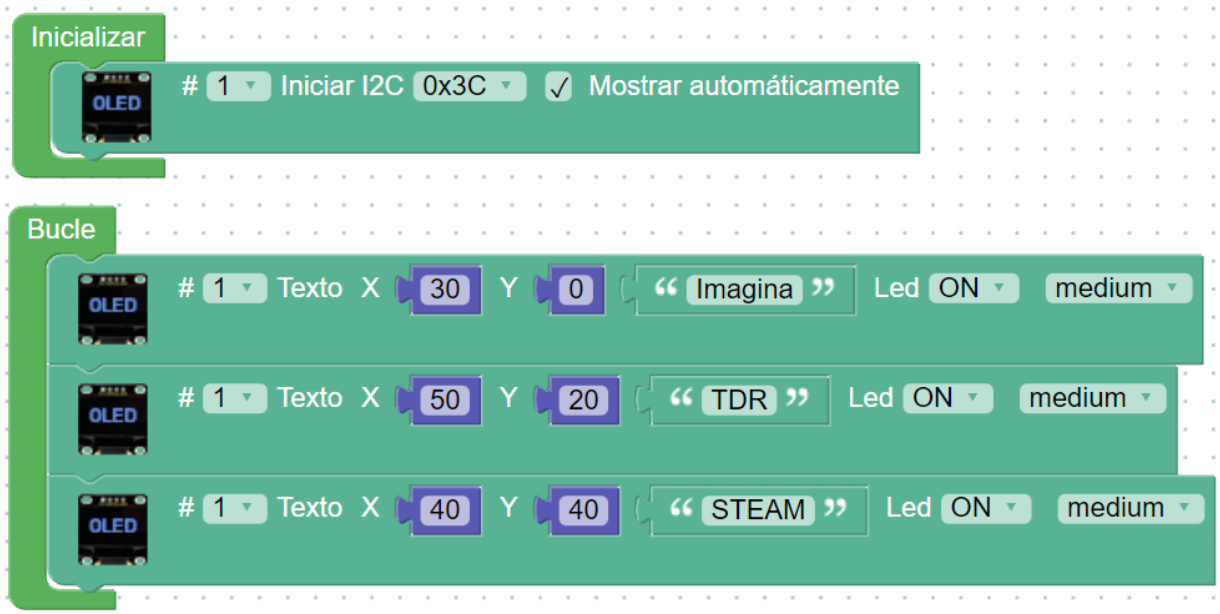


En esta actividad sólo vamos a trabajar con algunos de los bloques.

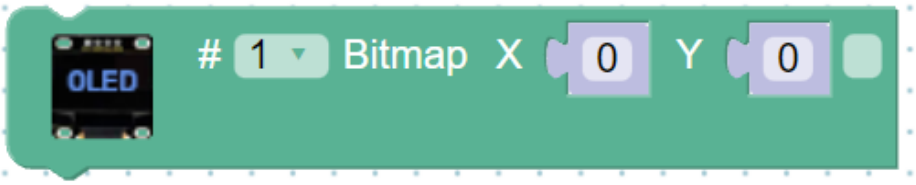
Realizaremos un programa que muestre un mensaje por la pantalla OLED. Primero configuraremos la dirección I2C al valor 0x3C. A continuación, mostraremos el texto por pantalla.

Podemos elegir la colocación del texto (x,y), el texto que es mostrará y entre tres tamaños de letra (*small*, *medium*, *big*).

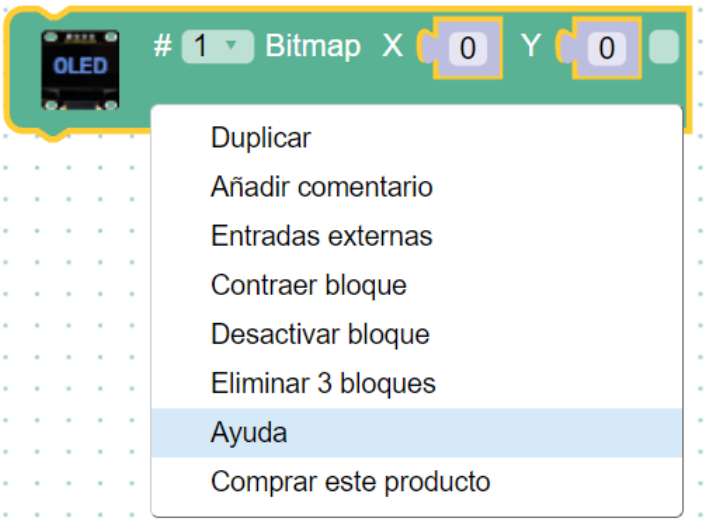
El programa resultante será:



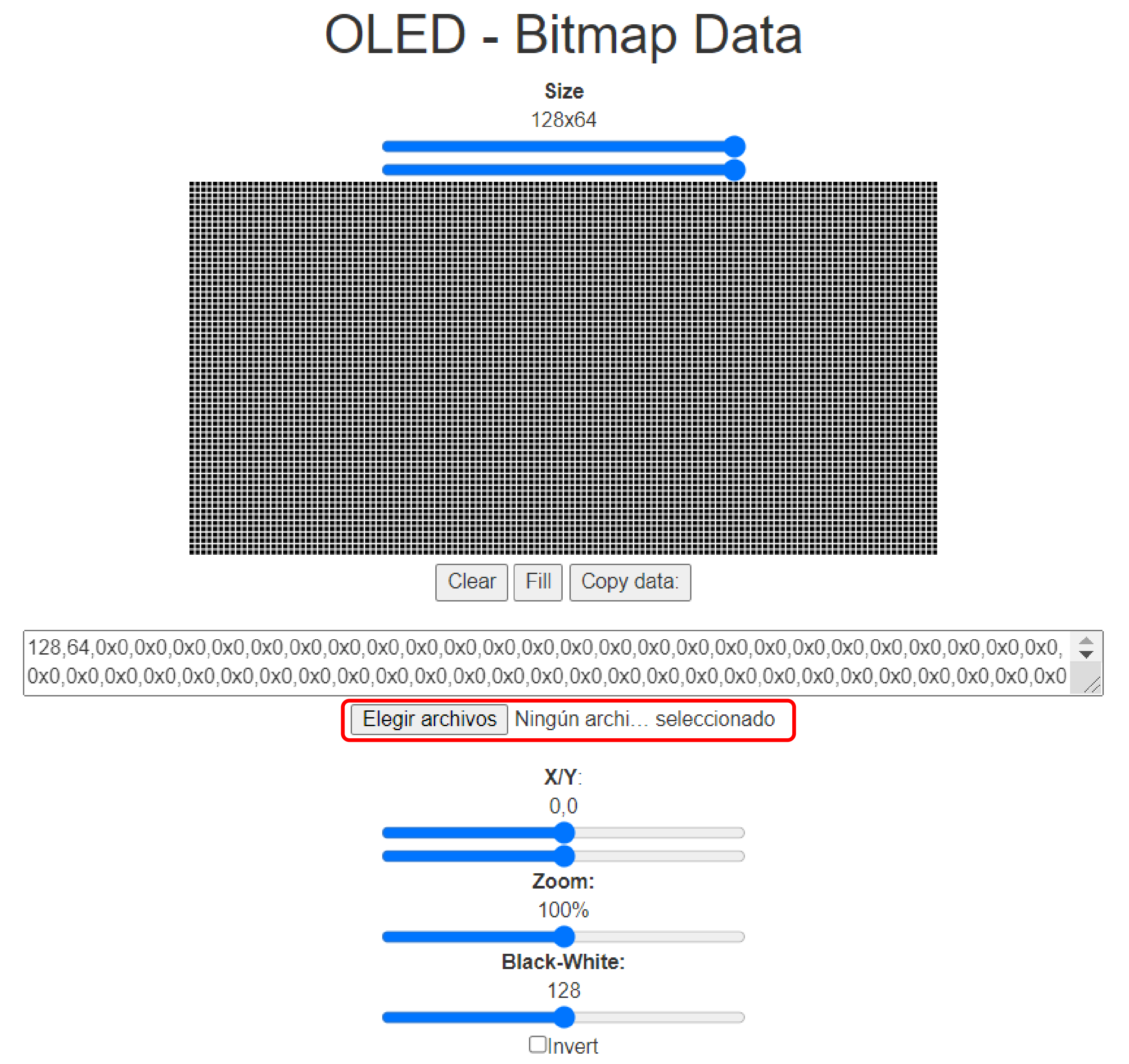
También podemos mostrar imágenes por la pantalla OLED. Vamos a ver cómo se realiza este proceso. Primero elegiremos el bloque para introducir un *Bitmap*.



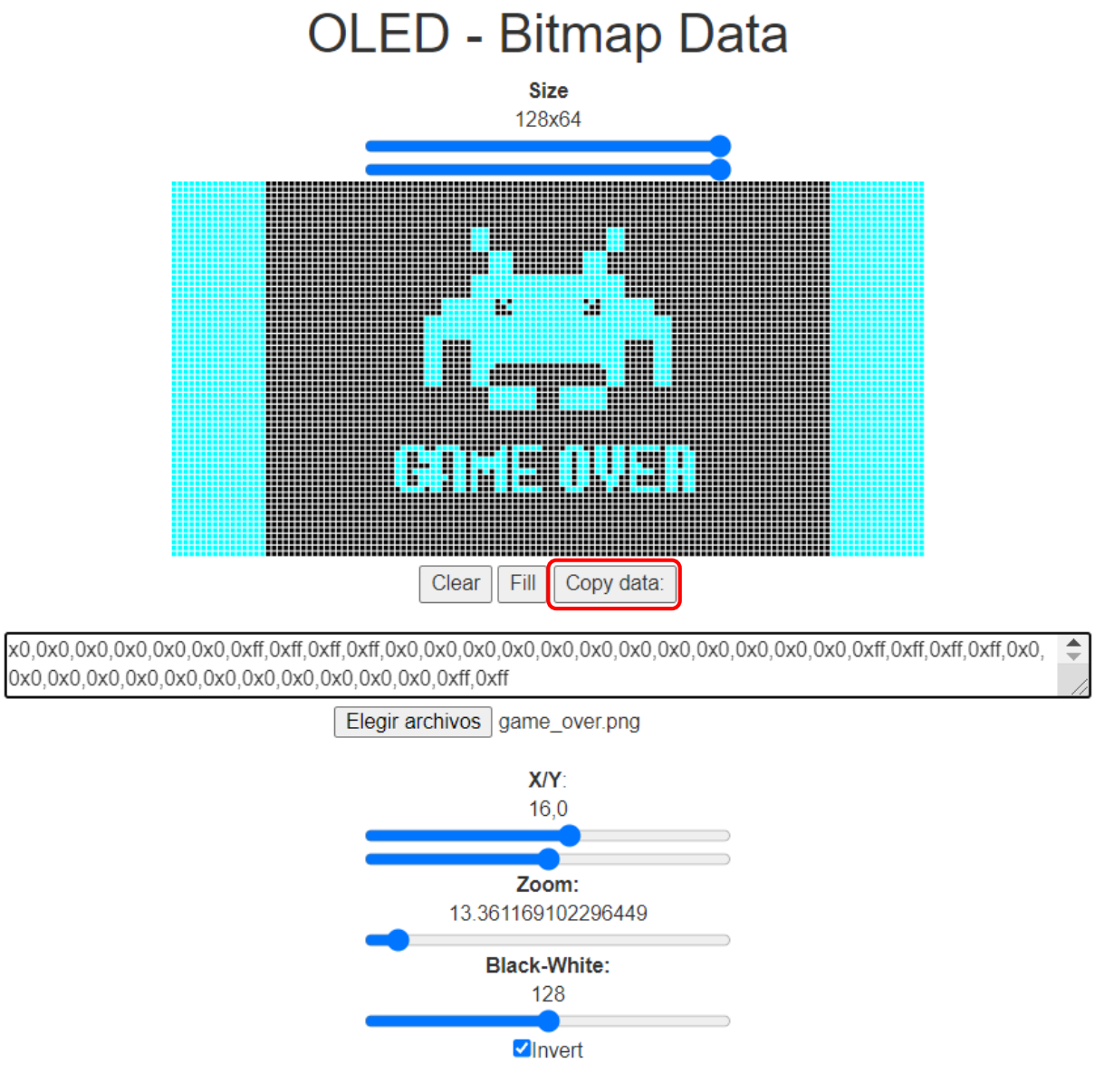
A continuación, pulsamos con el botón derecho del ratón y aparece un desplegable con diferentes opciones.



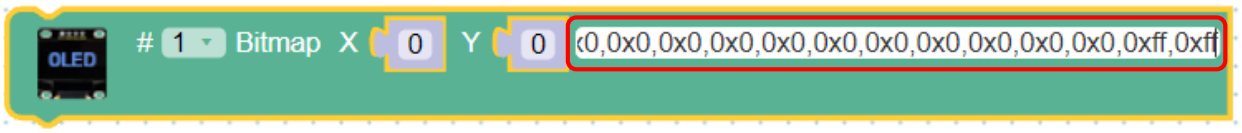
Elegiremos la opción *Ayuda* y nos mostrará una pantalla en el explorador (*OLED – Bitmap Data*).



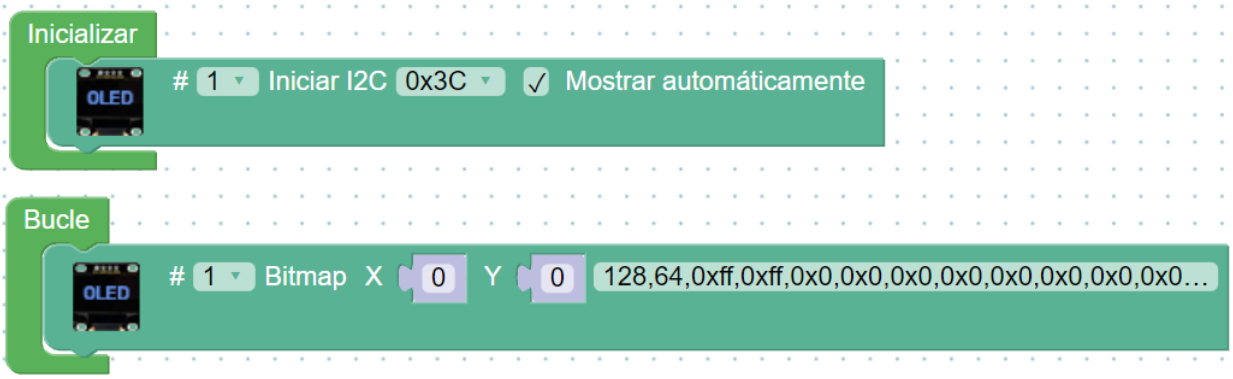
En el botón *Elegir archivos* podemos cargar una imagen. Una vez cargada la imagen la podemos invertir, desplazar, etc.



A continuación, copiamos los datos pulsando sobre *Copy data*. Y los pegamos dentro del bloque.



Ahora ya podemos visualizar la imagen en la pantalla OLED con este sencillo programa.



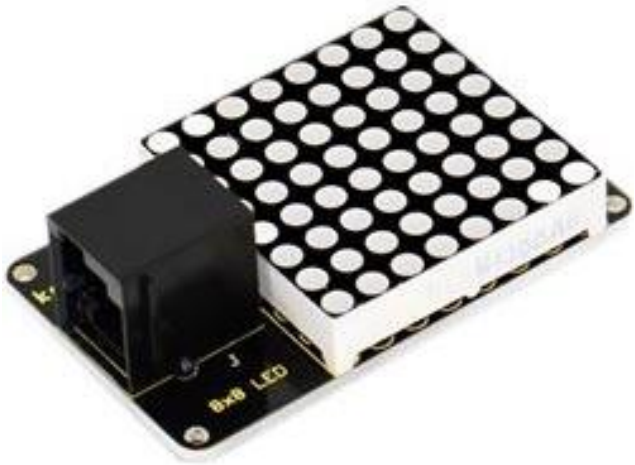
Enlace al programa: ArduinoBlocks Projects\oled.abp

Actividad de ampliación: realiza un programa que muestre una pantalla inicial con una imagen y, a continuación, que en la primera ponga la palabra "Contador" y en la segunda línea el valor de un contador de 0 a 255.

7.12.3 Reto A12.3. Matriz 8x8.

Reto A12.3. Matriz 8x8.

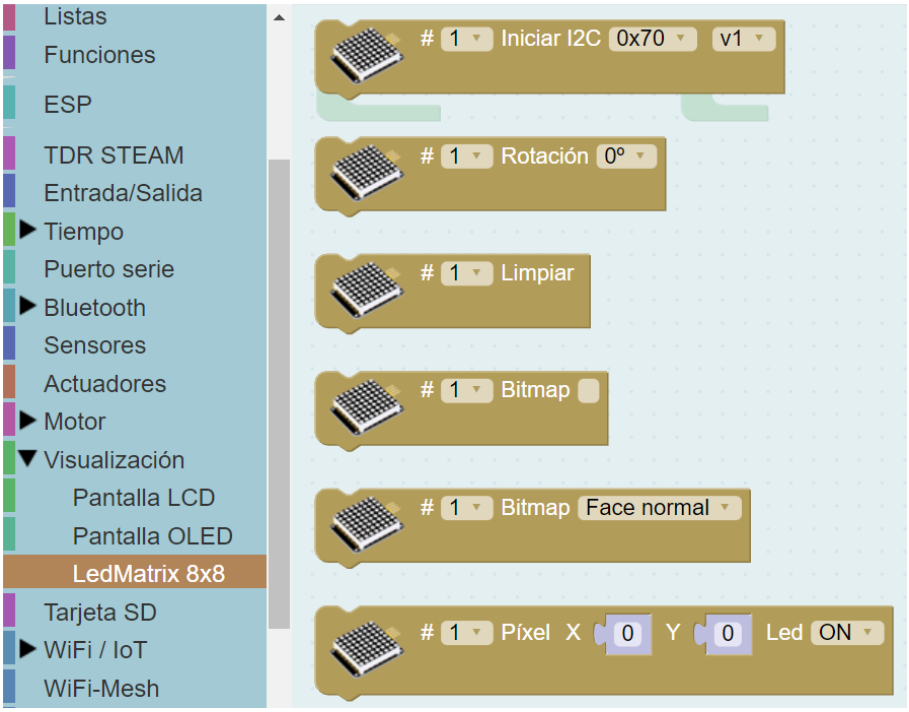
Este dispositivo está compuesto por 64 leds que forman una matriz de 8x8 con comunicación I2C. Este módulo utiliza el chip HT16K33 para controlar la matriz de puntos. Hay modelos de matrices 8x8 que van con una dirección I2C fija y otros modelos que tienen 3 interruptores DIP para poder cambiar la dirección I2C (la parte final de la dirección). Esto nos permite poder direccionar y controlar simultáneamente hasta 8 dispositivos. Este ser el modelo que utilizaremos.



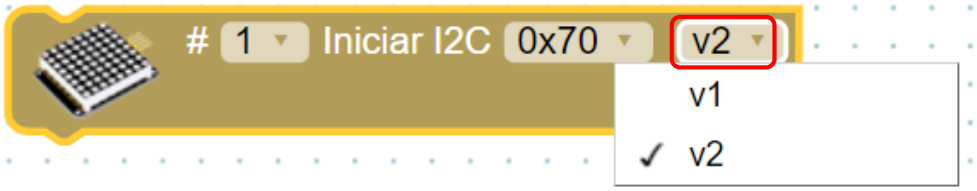
En la configuración del direccionamiento I2C podemos elegir entre 8 combinaciones (2^3=8) posibles. Si dejamos a OFF los tres interruptores (en la parte inferior) la dirección será 0x70 y si los ponemos a ON será 0x77.
El bit de menor peso (LSB) es el que está en el interruptor 1 y el bit de mayor peso (MSB) es el del interruptor 3.

Dirección	***1***	***2***	***3***
0x70	OFF	OFF	OFF
0x71	ON	OFF	OFF
0x72	OFF	ON	OFF
0x73	ON	ON	OFF
0x74	OFF	OFF	ON
0x75	ON	OFF	ON
0x76	OFF	ON	ON
0x77	ON	ON	ON

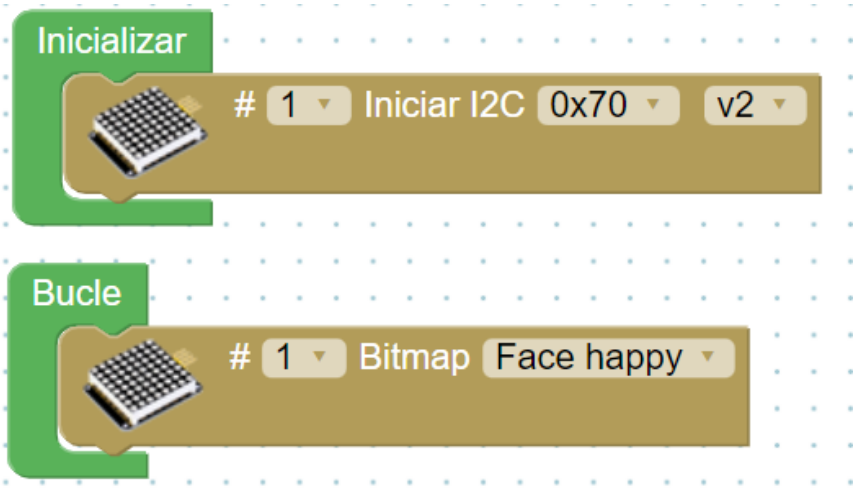
Tenemos una serie de funciones definidas para poder trabajar con este dispositivo. Estos bloques los tenemos dentro de *Visualización*.



También tenemos que elegir la versión de nuestra matriz, en este caso es la versión V2.



Vamos a realizar un sencillo programa que muestre una imagen en la matriz.

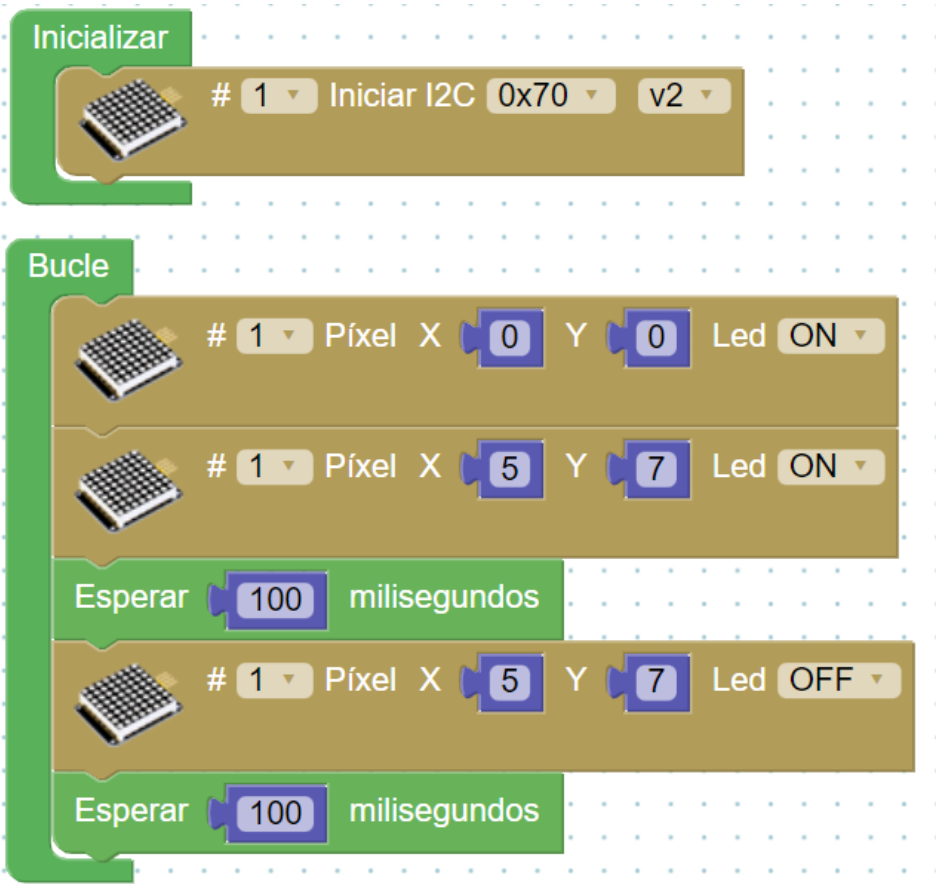


Ahora realizaremos otro programa que haga parpadear un led y otro esté fijo.

La posición (0,0) de la matriz está en la zona inferior derecha.

Posición (x,y) → (0,0)

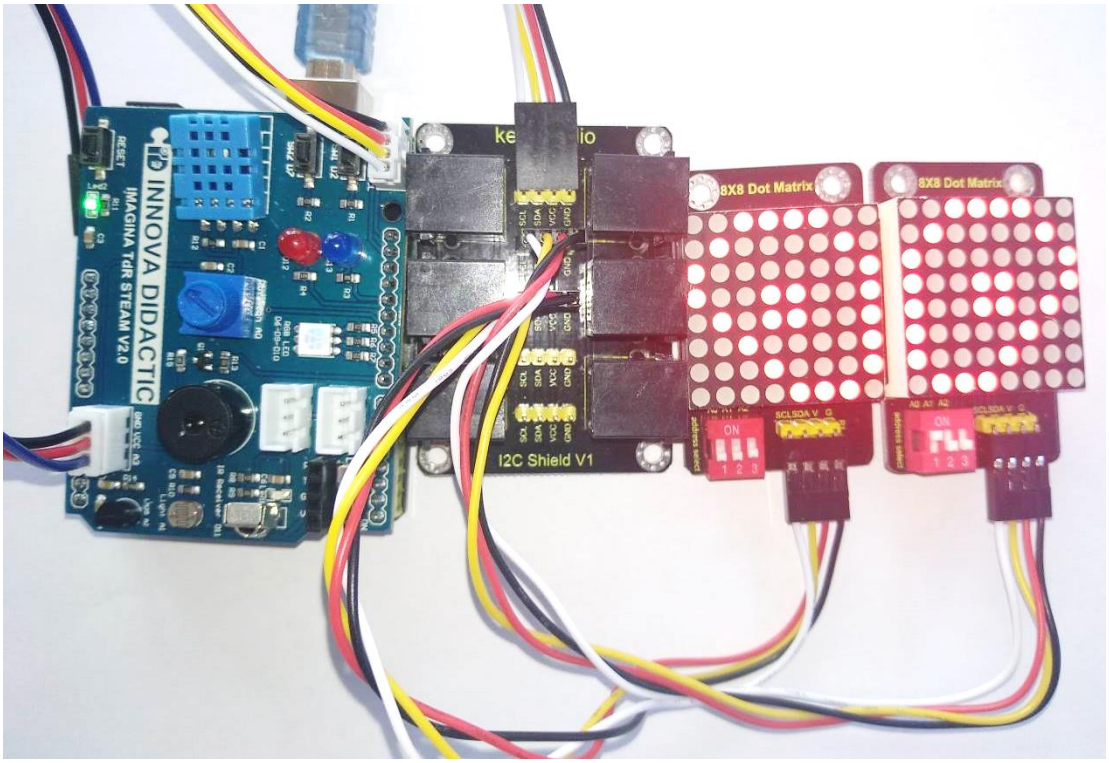
El programa resultante será:



Enlace al programa: ArduinoBlocks Projects\matriz_8x8_1.abp

Vamos a realizar ahora un programa con dos matrices 8x8 en el que mostraremos la imagen de dos ojos. Deberemos elegir el número y la dirección I2C de cada matriz. Para conectar las dos matrices podemos utilizar un *Hub I2C* para realizar las conexiones.

Enlace al programa: ArduinoBlocks Projects\matriz_8x8_2.abp



Actividad de ampliación: realiza un programa que una matriz 8x8 simule que una flecha está rotando en sentido horario.

7.12.4 Reto A12.4. Sensor de color.

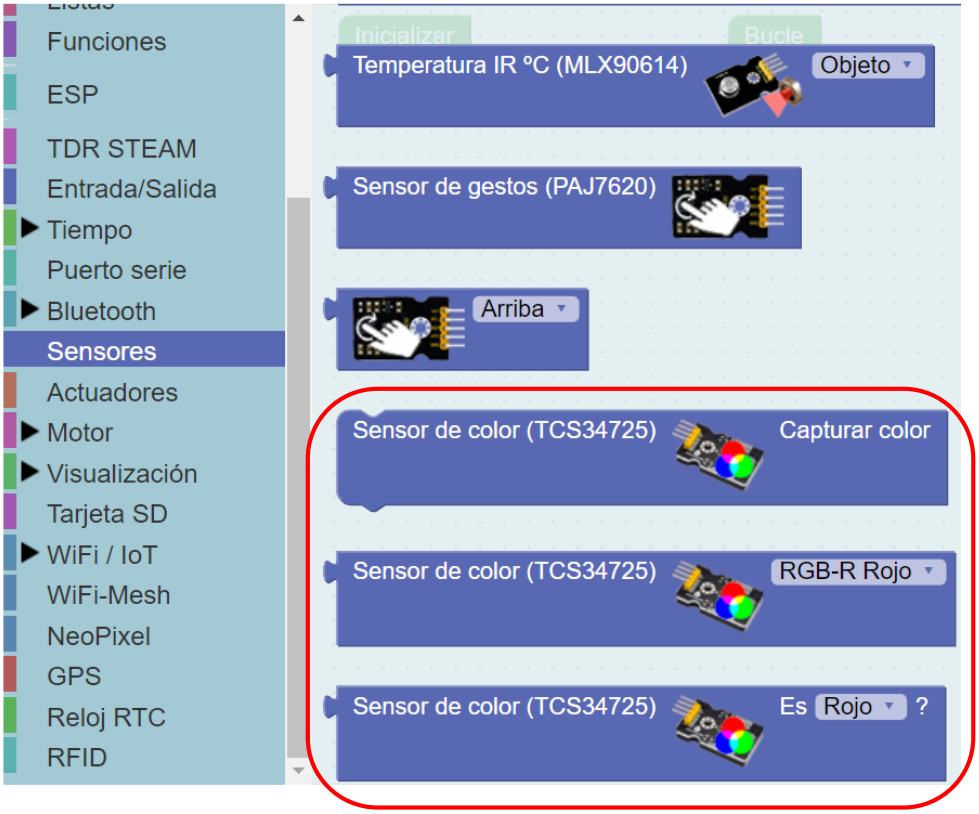
Reto A12.4. Sensor de color.

El sensor de color TCS 34725 es un sensor que puede reconocer el color de una superficie por detección óptica. El sensor se ilumina con una luz brillante y emite los tres colores RGB para ayudar a restaurar el color. Además, para evitar la interferencia del entorno y aumentar la precisión, dispone de un sistema de luz infrarroja en la parte inferior del sensor, de manera que el elemento del espectro infrarrojo incidente se minimiza para que la gestión del color sea más precisa.

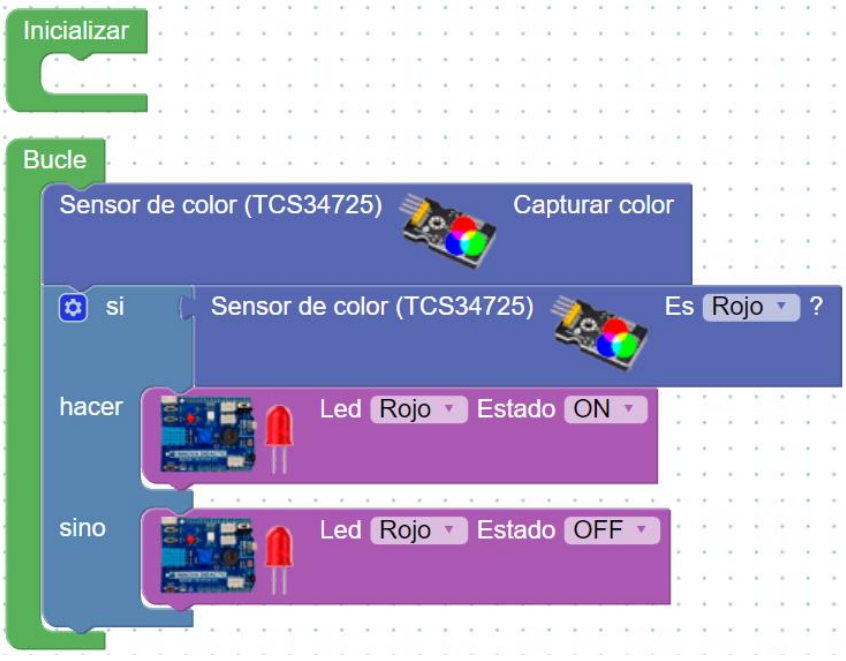
En la parte inferior del sensor se pueden ver los cuatro leds de color amarillo que realizan la iluminación y el sensor que está en la zona central. El sensor tiene un filtro de alta sensibilidad y un amplio rango dinámico.

El sensor se comunica con la placa ***ESP32 Plus STEAMakers*** a través del bus de comunicaciones I2C. No necesitaremos asignarle direccionamiento porque ya vienen predefinido dentro del propio bloque.

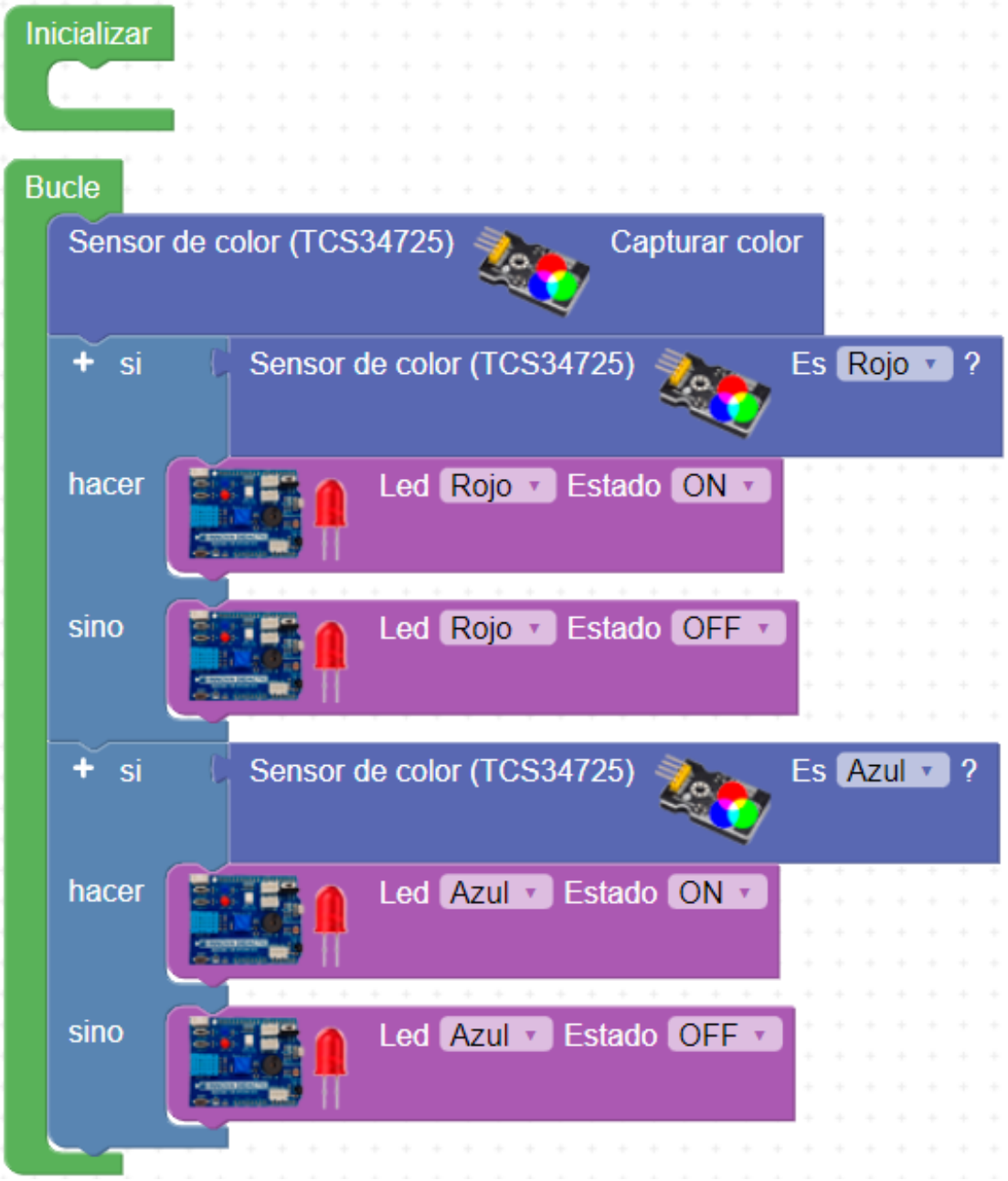
ArduinoBlocks dispone de tres bloques específicos para controlar este sensor y los encontramos dentro del apartado de *Sensores*.



Siempre tenemos que hacer una captura de color para poder identificar el color y, a continuación, determinar el parámetro que queremos identificar. Vamos a realizar un programa para identificar una superficie de color roja. Si es correcto el color se encenderá el led rojo y sino permanecerá apagado.

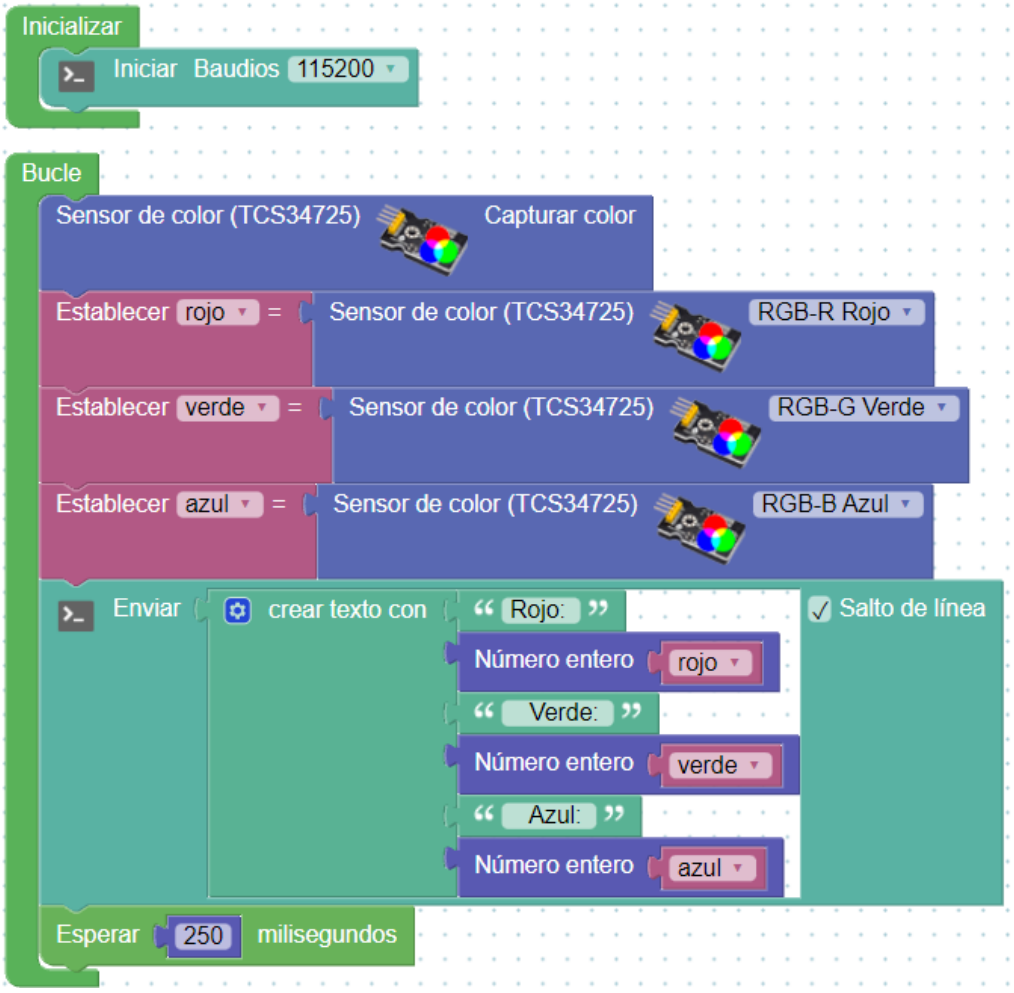


Podemos ampliar el programa anterior añadiendo la detección del color azul y encendiendo el led azul. El programa resultante quedará de la siguiente forma:



Enlace al programa: ArduinoBlocks Projects\sensor_color.abp

También podemos identificar la cantidad de cada componente RGB que hay en un color de una superficie. Para ello, vamos a realizar un programa para identificar las componentes de cada color y muestre los datos por la *Consola*.



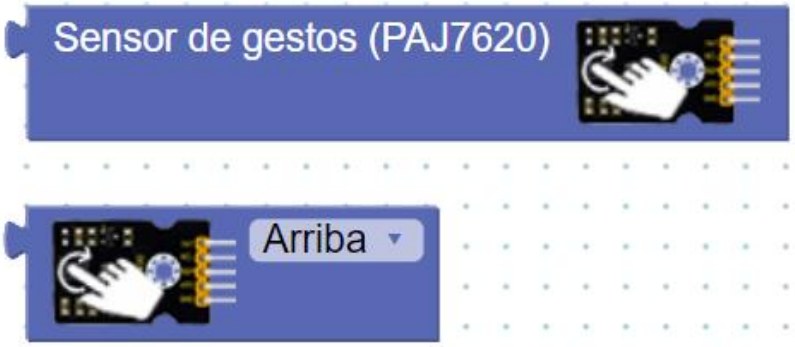
Actividad de ampliación: realiza un programa que en función de los colores toque una nota musical con el zumbador.

7.12.5 Reto A12.5. Sensor de reconocimiento de gestos.

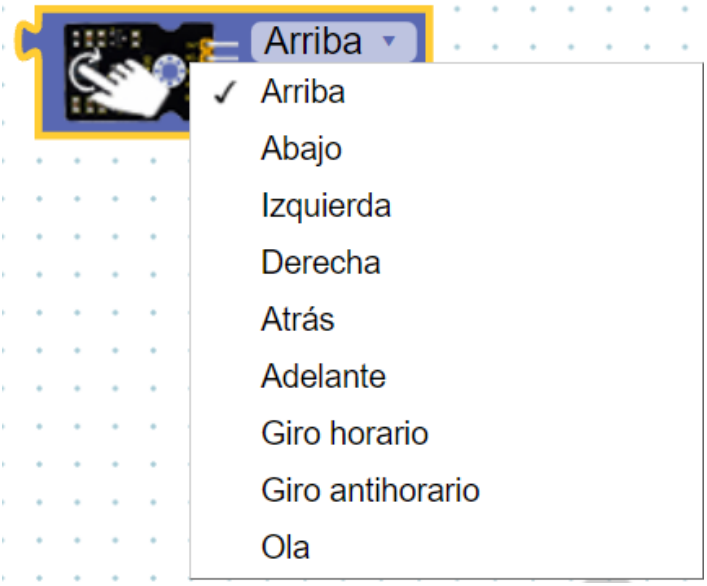
Reto A12.5. Sensor de reconocimiento de gestos.

El sensor de reconocimiento de gestos integra el chip PAJ7620 para el reconocimiento de gestos e interfaz I2C. Este sensor de gestos puede reconocer nueve gestos, incluidos moverse hacia arriba, hacia abajo, hacia la izquierda, hacia la derecha, hacia adelante, hacia atrás, en el sentido de las agujas del reloj, en sentido contrario a las agujas del reloj y de tipo onda.

En el apartado de *Sensores* de ArduinoBlocks tenemos dos bloques para trabajar con este sensor.

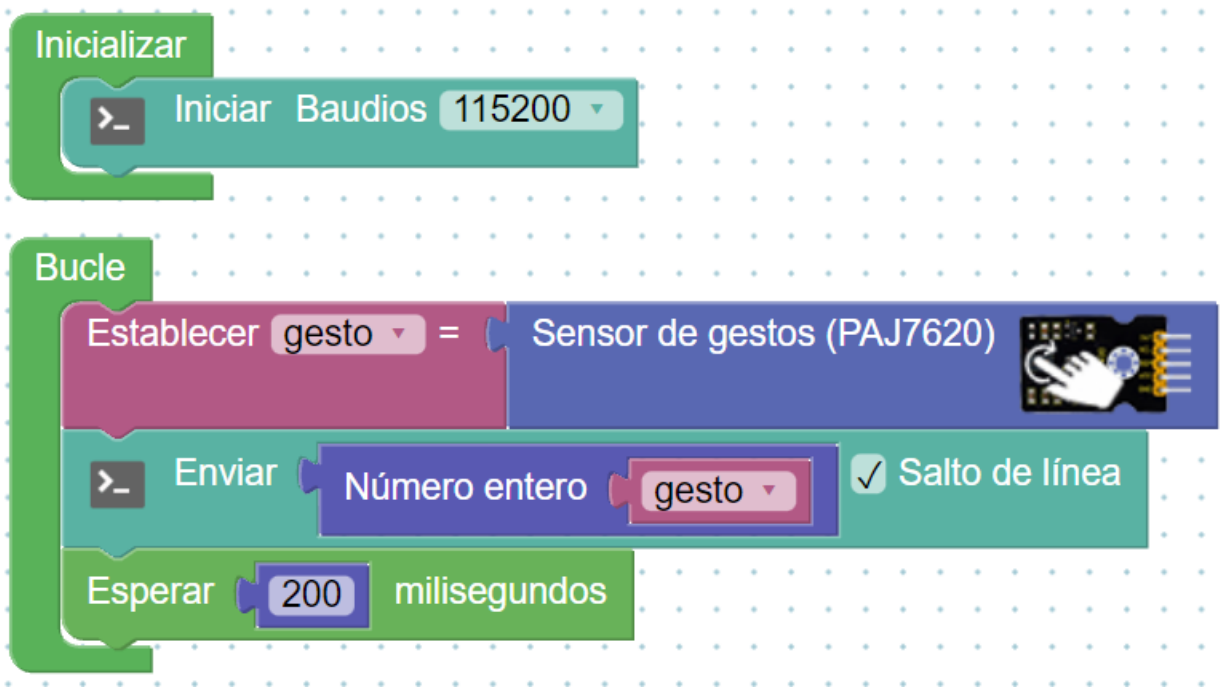


Podemos elegir entre 9 tipos de gestos diferentes:



Primero deberemos leer el gesto y luego determinar que gesto estamos haciendo. Mostraremos los valores de los gestos por la *Consola*.

Con este sencillo programa podemos determinar qué valor numérico equivale con cada movimiento.



Enlace al programa: ArduinoBlocks Projects\sensor_gestos_1.abp

Partiendo del programa anterior, ahora vamos a realizar un programa que encienda o apague el led rojo y azul en función de los movimientos de la mano:

- Arriba: enciende led rojo.
- Abajo: apaga led rojo.
- Izquierda: enciende led azul.
- Derecha: apaga led azul.

El programa resultante es el siguiente:

Enlace al programa: ArduinoBlocks Projects\sensor_gestos_2.abp

Actividad de ampliación: realiza un programa que en función de cada uno de los 9 movimientos encienda el led RGB con un color diferente.

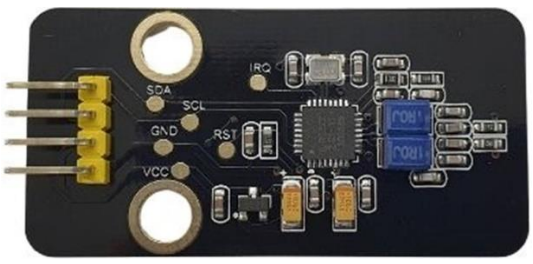
7.12.6 Reto A12.6. Sensor RFID.

Reto A12.6. Sensor RFID.

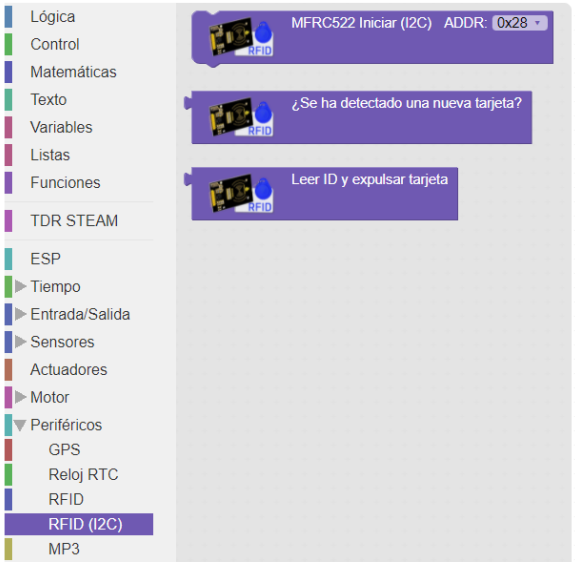
El sensor de RFID (sensor de identificación por radiofrecuencia: *Radio Frequency IDentification*) es un sistema de identificación de productos que puede parecer similar al código de barras tradicional, pero que tiene grandes ventajas con respecto a este. A diferencia del código de barras que utiliza la imagen para identificar una etiqueta, el sistema RFID utiliza las ondas de radio para comunicarse con un circuito electrónico. Puede estar montado sobre gran cantidad de soportes, como por ejemplo un *tag* o etiqueta RFID, una tarjeta o un transpondedor.

Un circuito RFID tiene una gran capacidad de almacenamiento de datos, por lo que permite guardar mucha más información que las etiquetas de código de barras tradicional. Su tecnología hace que sean muy difíciles de duplicar lo que aumenta su seguridad y permiten realizar la lectura de forma prácticamente instantánea, a distancia y sin necesidad de línea de visión.

El sensor RFID utilizado está basado en el módulo MF522-AN. Es fácil de utilizar, de bajo costo y, adecuado para el desarrollo de equipos y el desarrollo de aplicaciones avanzadas para usuarios de lectores. Este módulo se puede conectar de forma sencilla a través de sus cuatro terminales, ya que utiliza un sistema de comunicación I2C para conectarse a un microcontrolador.



ArduinoBlocks dispone de una serie de bloques específicos para realizar la programación de este sensor dentro del apartado *Periféricos-RFID(I2)*.



Disponemos de tres bloques para trabajar con este sensor. Vamos a realizar un sencillo programa para leer una tarjeta RFID.

El programa que vamos realizar nos permitirá leer el código de la tarjeta RFID y lo mostrará por la *Consola*.

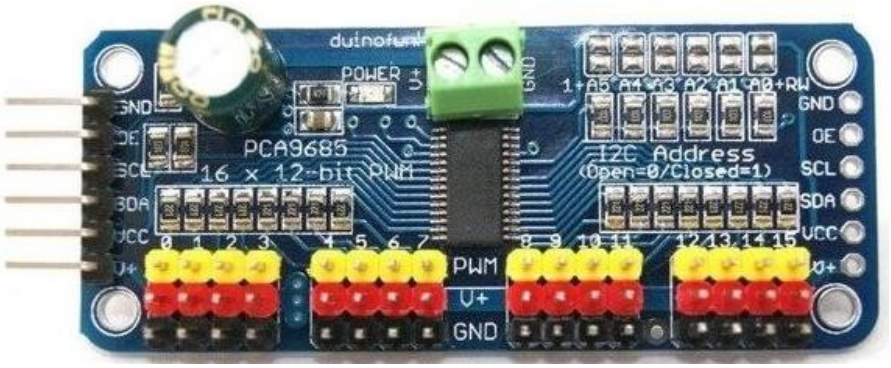
Podemos observar que el dispositivo I2C de lectura de RFID tiene la dirección 0x28, hay un dispositivo I2C conectado a la placa ***ESP32 Plus STEAMakers*** y el código del elemento RFID que leemos es: 046DDBBA114A80.

Actividad de ampliación: realiza el programa anterior y comprueba su funcionamiento.

7.12.7 Reto A12.7. Controlador de servos PCA9685.

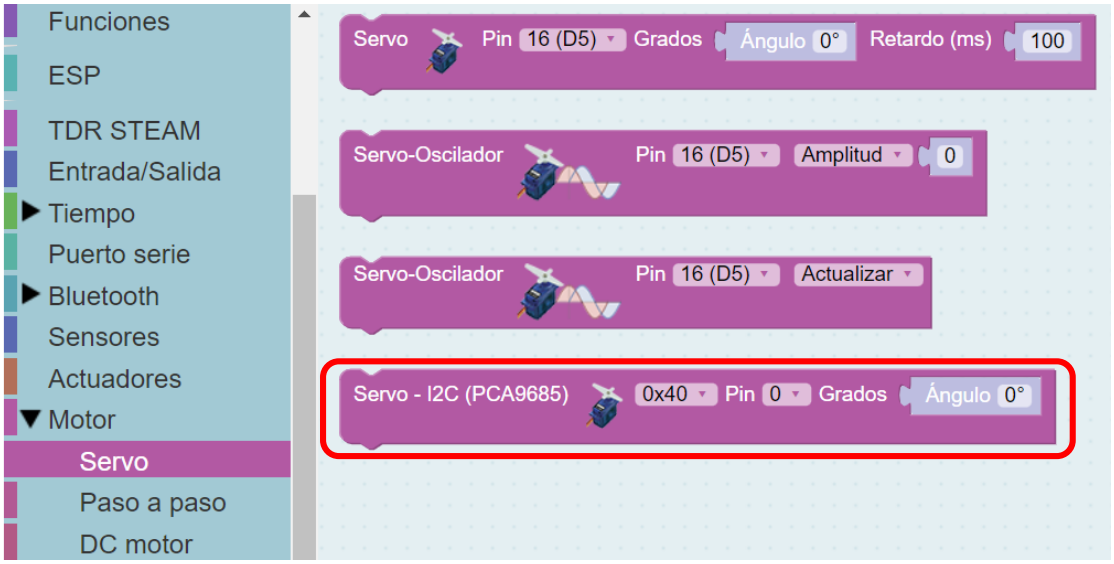
Reto A12.7. Controlador de servos PCA9685.

La placa PCA9685 es una placa que permite controlar hasta 16 servomotores mediante comunicación I2C. Dispone de una alimentación independiente para los motores, así que no utilizaremos la alimentación de la placa ***ESP32 Plus STEAMakers***.

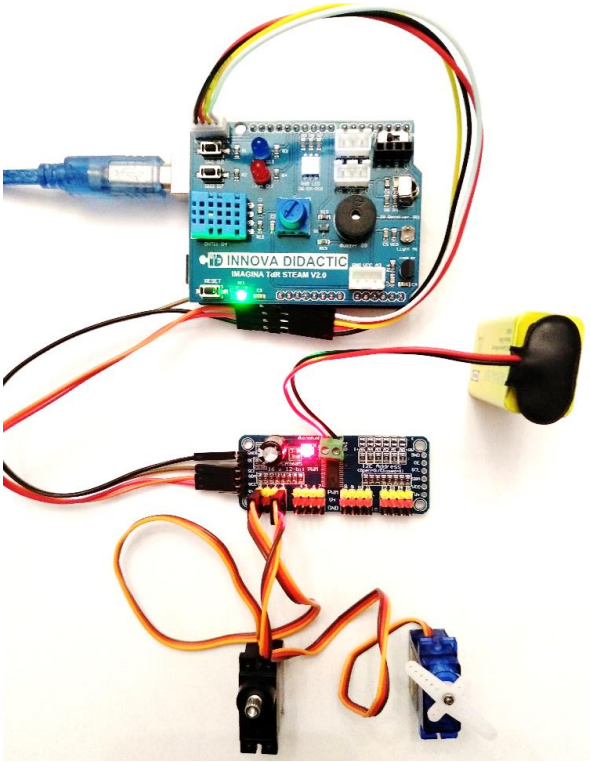


Para conectarla utilizaremos cable Dupont respetando las conexiones VCC – GND – SDA – SCL (el resto de pines los dejaremos libres). Conectaremos directamente sobre la tira de pines acodados hacia el conector I2C de la ***Imagina TDR STEAM***. Para la alimentación de los motores podemos utilizar una fuente de alimentación, pilas o baterías recargables en el conector GND – V+. La tensión máxima para los motores (V+) será de 6V.

En ArduinoBlocks, disponemos de un bloque específico para poder controlar varios servomotores a la vez.



Realizaremos un programa para controlar dos servomotores conectados en las posiciones 0 y 3.



Actividad de ampliación: realiza el programa anterior y comprueba su funcionamiento.

7.12.8 Reto A12.8. Sensor de CO2 equivalente CCS811.

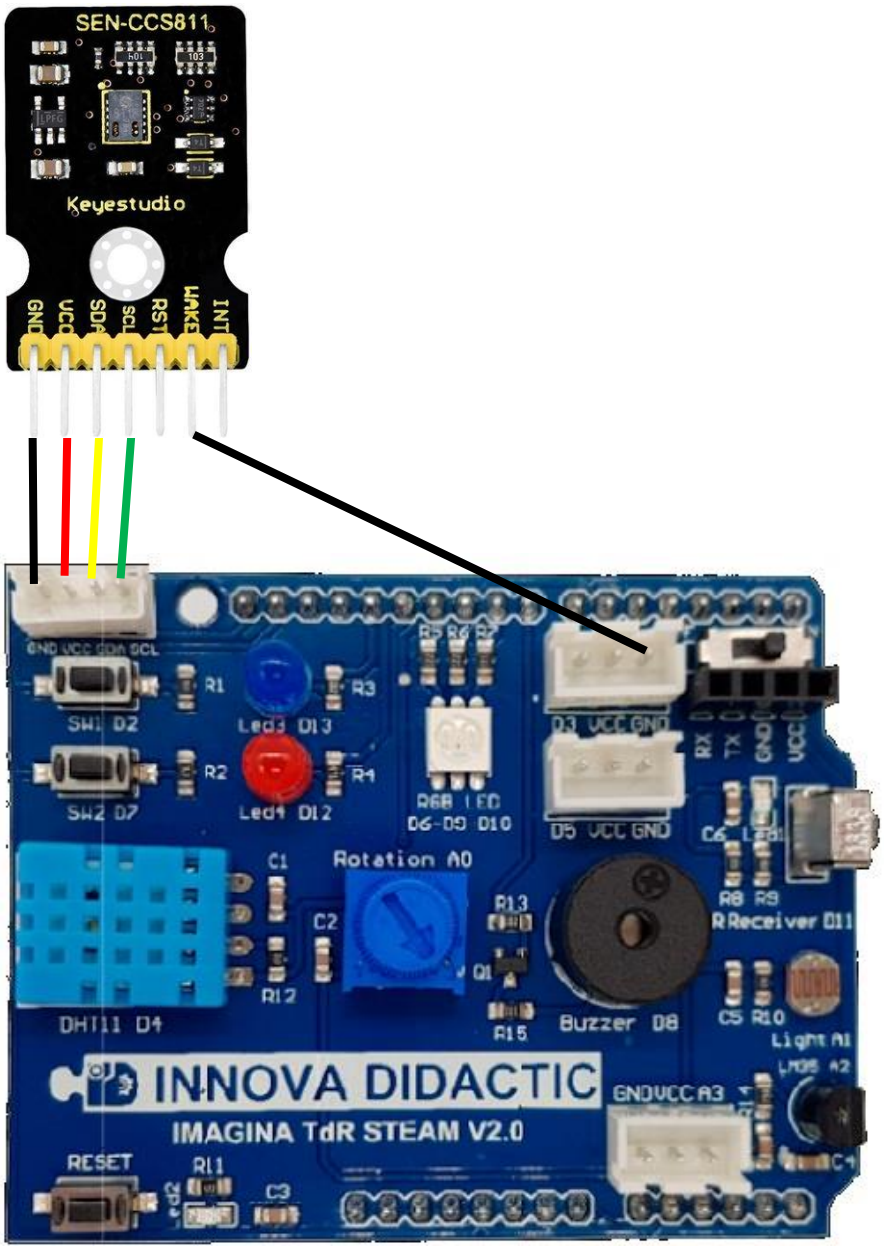
Reto A12.8. Sensor de CO2 equivalente CCS811.

El sensor CCS811 es un sensor para la medición calidad del aire interior, que podemos usar fácilmente. Para determinar la calidad del aire interior, el CCS811 es un sensor multigas MOX(Metal-Oxide) que incluye medición de monóxido de carbono (CO) y compuestos volátiles (VOCs) como etanol, aminas, o hidrocarburos aromáticos. Con ellos, el sensor CCS811 puede determinar la cantidad de dióxido de carbono equivalente (eCO2). El rango de medición es entre 400 a 8192 ppm en eCO2 y de 0 a 1187 ppb en TVOC.

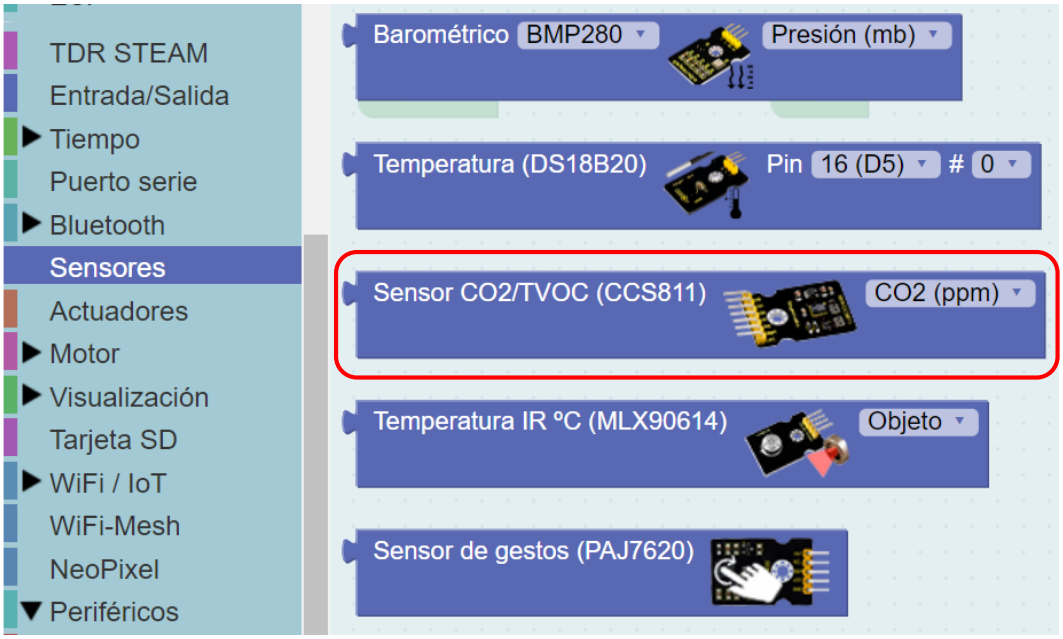
Al igual que la mayoría de sensores químicos, el sensor CCS811 necesita un precalentamiento. El fabricante recomienda que el sensor funcione durante 20 minutos para que las mediciones se estabilicen, y durante 48 horas si se cambia de ubicación.

El sensor CCS811 incluye un conversor ADC y un procesador interno para realizar los cálculos y comunicación a través de bus I2C.

Para la conexión utilizaremos los 4 pines que se encuentran a la izquierda y conectaremos a la placa ***Imagina TDR STEAM*** mediante cables Dupont. Es muy importante conectar el pin WAKE a GND porque, sino, no enviará datos.

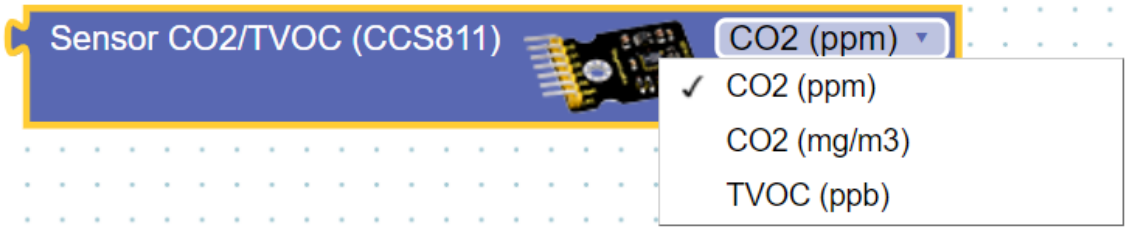


ArduinoBlocks dispone de un bloque para poder trabajar con este sensor.

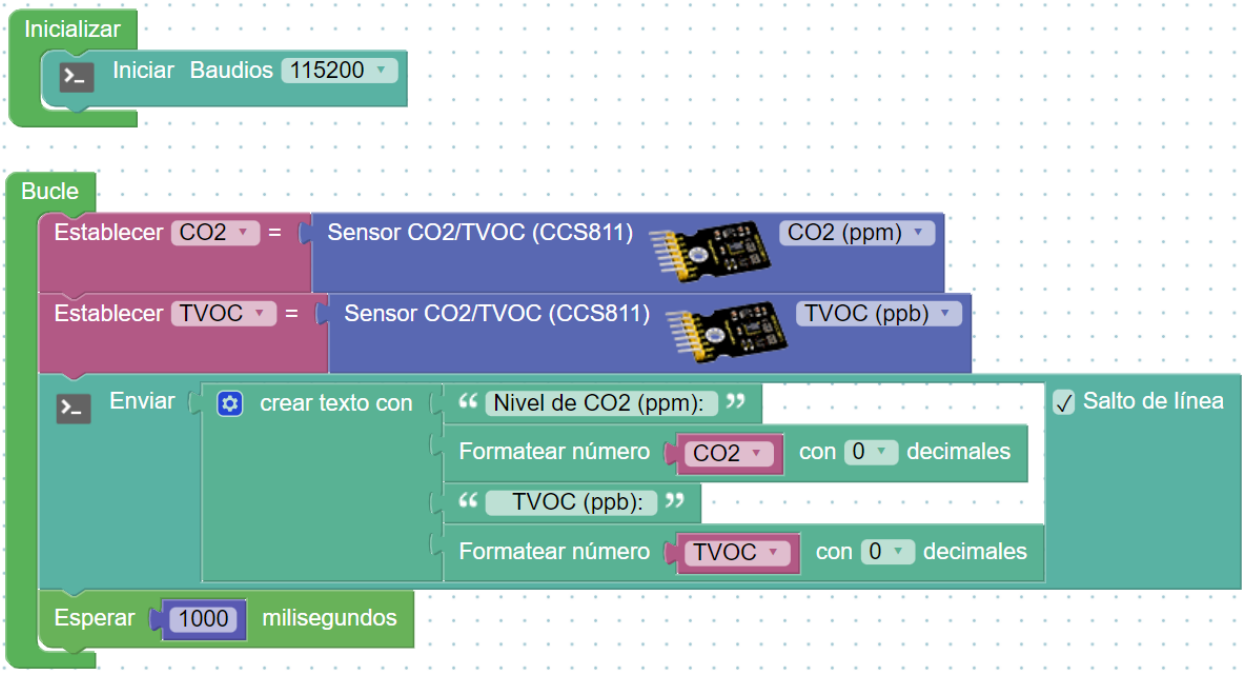


El sensor CCS811 dispone de tres opciones para mostrar la información:

- CO2 (ppm):
- CO2 (mg/m^3);
- TVOC (ppb):



Vamos a realizar un programa para poder ver los datos de CO2 en ppm por la *Consola*. El programa necesitará unos segundos para empezar a mostrar valores adecuados.



Enlace al programa: ArduinoBlocks Projects\ccs811.abp

A continuación, se muestran una serie de datos obtenidos. Aumenta el valor cuando encendemos cerca un mechero (por ejemplo).

```
Nivel de CO2 (ppm): 400   TVOC (ppb):  0
Nivel de CO2 (ppm): 2537   TVOC (ppb): 325
Nivel de CO2 (ppm): 1014   TVOC (ppb): 93
Nivel de CO2 (ppm): 539   TVOC (ppb): 21
Nivel de CO2 (ppm): 448   TVOC (ppb):  7
Nivel de CO2 (ppm): 439   TVOC (ppb):  5
Nivel de CO2 (ppm): 420   TVOC (ppb):  3
Nivel de CO2 (ppm): 400   TVOC (ppb):  0
Nivel de CO2 (ppm): 400   TVOC (ppb):  0
Nivel de CO2 (ppm): 400   TVOC (ppb):  0
```

Actividad de ampliación: realiza el programa anterior y comprueba su funcionamiento.

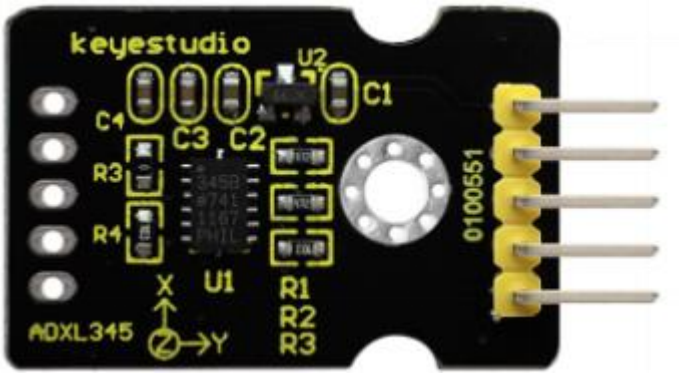
7.12.9 Reto A12.9. Sensor acelerómetro de tres ejes ADXL345.

Reto A12.9. Sensor acelerómetro de tres ejes ADXL345.

El sensor ADXL345 es un acelerómetro de 3 ejes independientes (3DOF) de tipo micromecanizado (MEMS) con medición de alta resolución (13 bits) de hasta +/-16 g. Los datos de salida digital se formatean como un complemento de 16 bits a dos y se puede acceder a ellos a través de una interfaz digital SPI (3 o 4 cables) o I2C. En nuestro caso utilizaremos el sistema de comunicación mediante I2C.

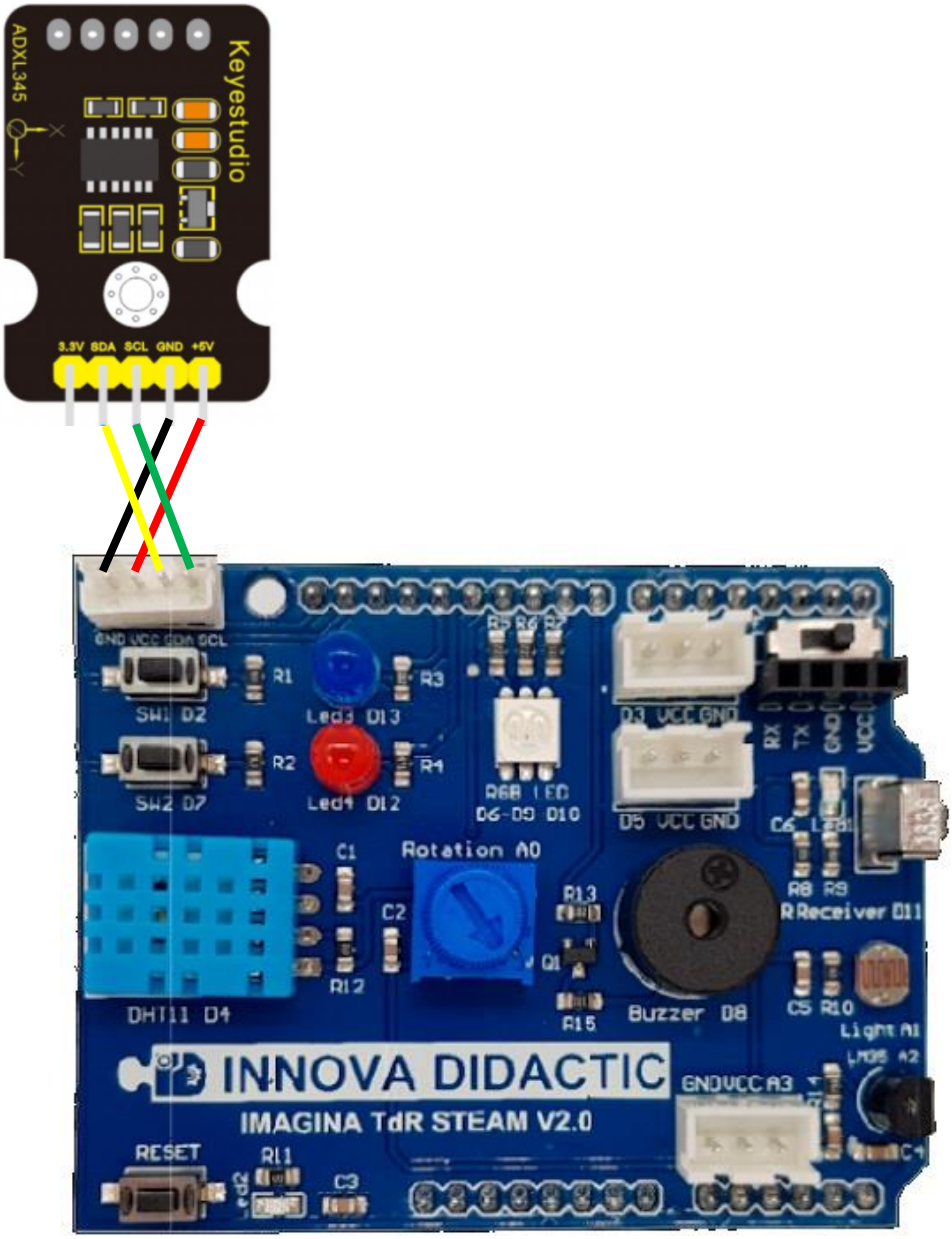
El ADXL345 tiene un rango de medición ajustable entre ±2g, ±4g, ±8g, ±16g, y una alta resolución de hasta 13 bits, con una sensibilidad de 40mg/LSB en todos los rangos, lo que equivale a una precisión superior a un 1º.

Los acelerómetros son dispositivos muy utilizados en todo tipo de dispositivos. Se utilizan para determinar la orientación en móviles o tablets, la detección de caída libre, la medición de pasos en podómetros, la estabilización de cámaras, alarmas y sensores de vibración, etc.



El ADXL345 dispone de dos pines de interrupciones que podemos configurar para responder a ciertos eventos. Estos eventos incluyen la detección de movimientos rápidos producidos por golpes y vibraciones, en uno o dos pulsos, y detección de condiciones de caída libre 0g. Pero en la configuración de Keyestudio no tenemos acceso a esas conexiones.

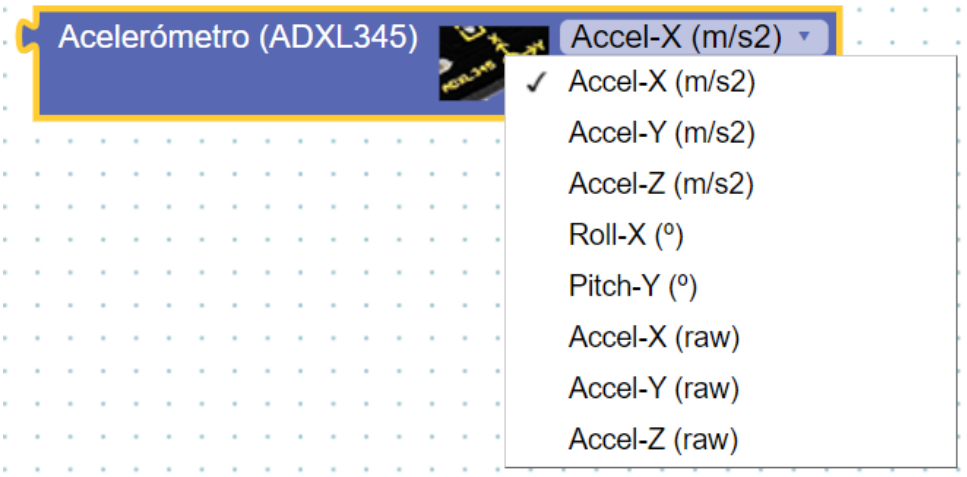
La conexión la realizaremos utilizando cable Dupont y respetando el nombre de las conexiones, tal y como se observa en la siguiente imagen.



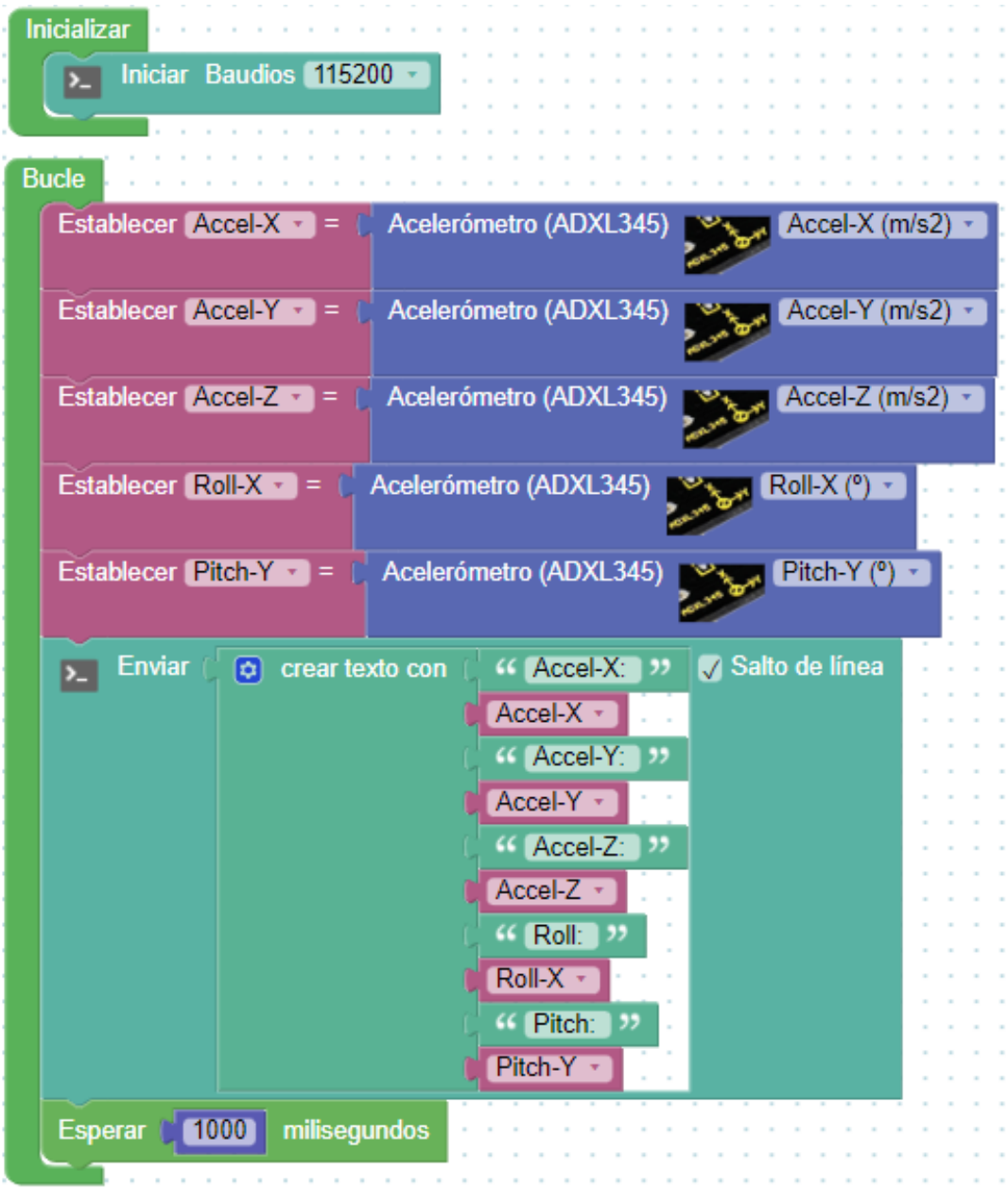
ArduinoBlocks dispone de un bloque para poder trabajar con este sensor.

Dentro del bloque podemos elegir entre los diferentes datos que nos puede entregar el sensor:

- Accel-X (m/s^2): aceleración en el eje X en m/s^2.
- Accel-Y (m/s^2): aceleración en el eje Y en m/s^2.
- Accel-Z (m/s^2): aceleración en el eje Z en m/s^2.
- Roll-X (º): rotación (eje X) en grados.
- Pitch-Y (º): cabeceo (eje Y) en grados.
- Accel-X (raw):
- Accel-Y (raw):
- Accel-Z (raw):

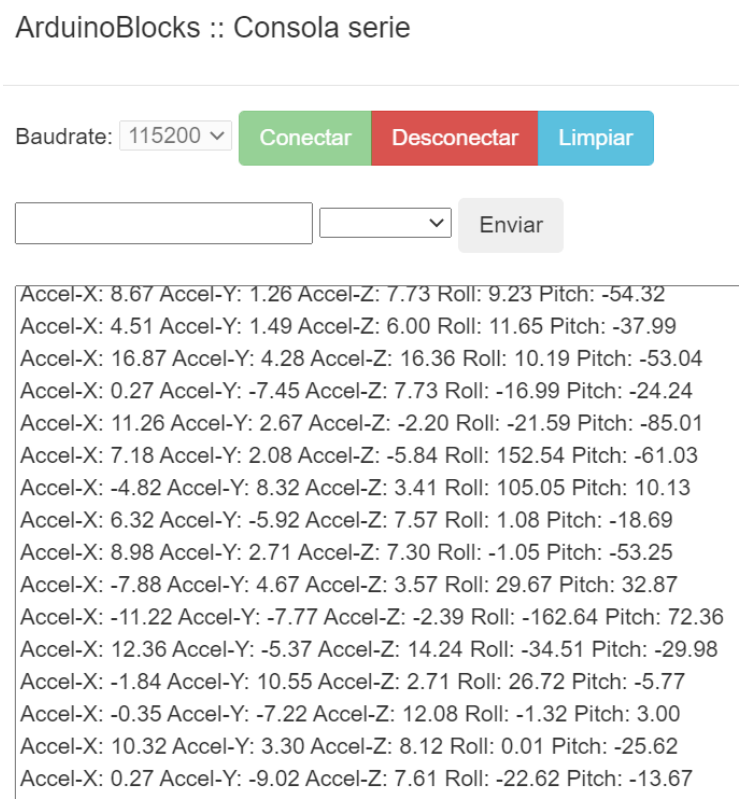


Vamos a realizar un programa para ver los datos de aceleración en los tres ejes X-Y-Z, y el Roll (X) y el Pitch (Y).



Enlace al programa: ArduinoBlocks Projects\adxl345.abp

Los datos obtenidos serán del tipo:



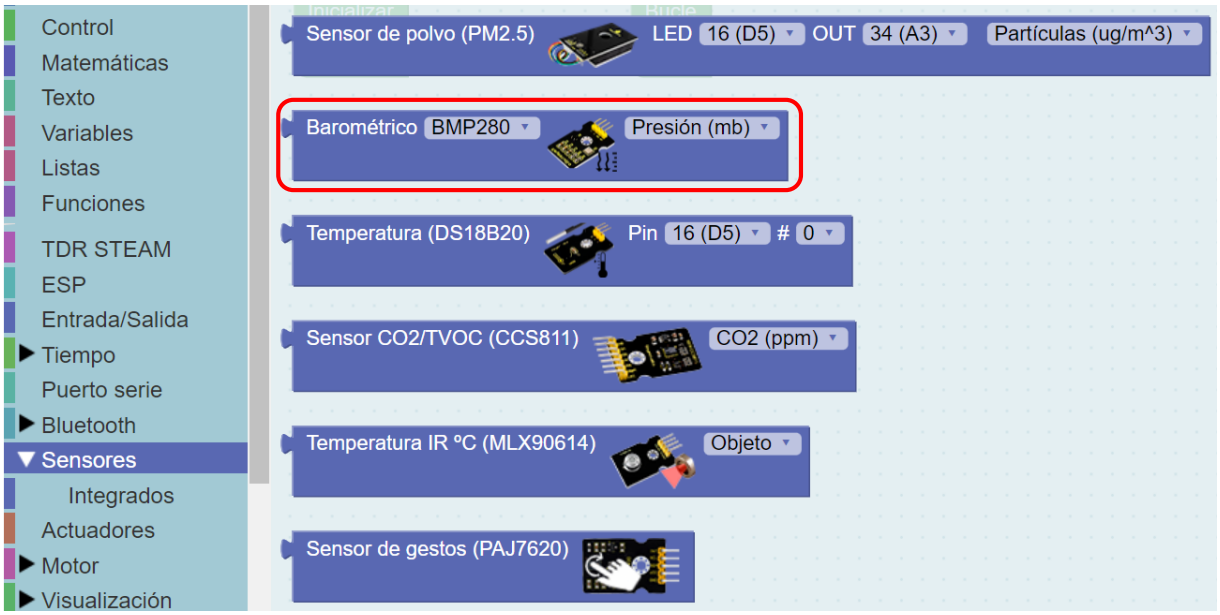
Actividad de ampliación: realiza el programa anterior y comprueba su funcionamiento.

7.12.10 Reto A12.10. Sensor de presión barométrica BMP280.

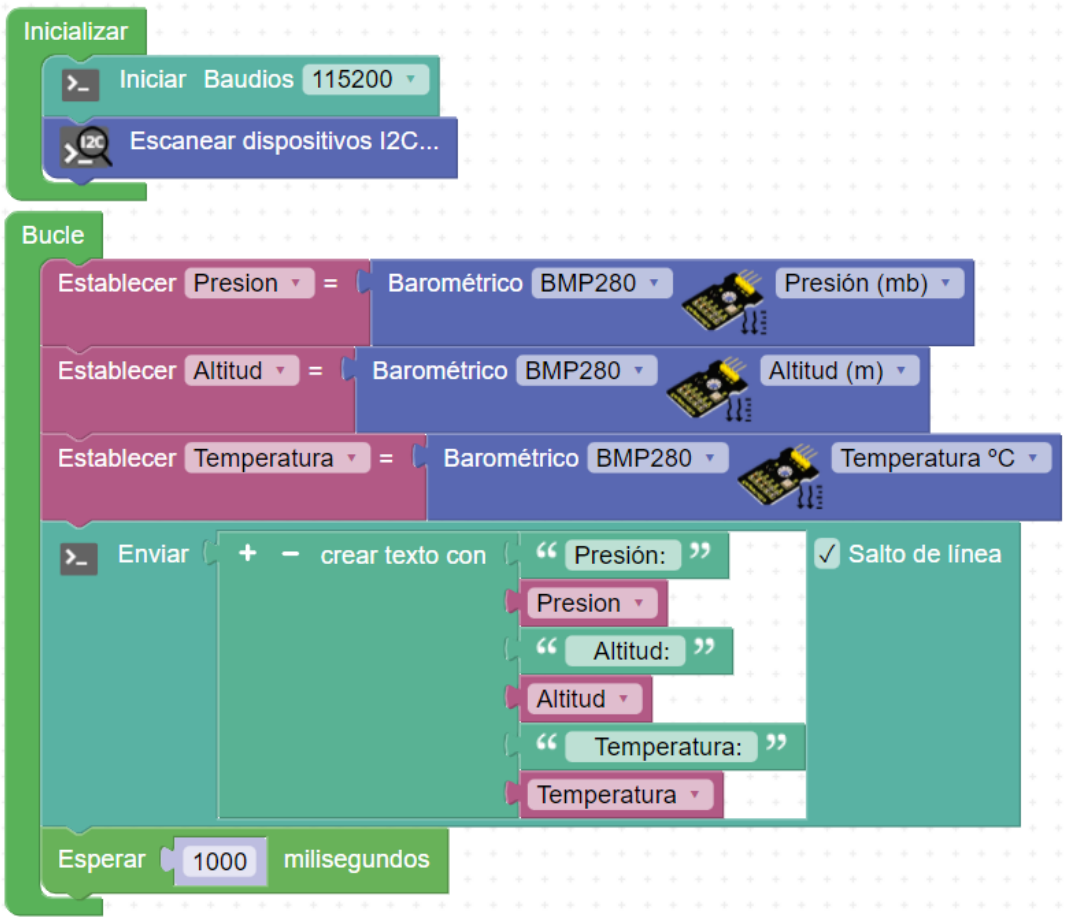
Reto A12.10. Sensor de presión barométrica BMP280.

El sensor BMP280 es un sensor de presión barométrica, temperatura y altitud. Dispone de un sistema para medir la presión de tipo piezoresistivos con alta precisión y linealidad, así como estabilidad a largo plazo y alta robustez EMC. Las múltiples opciones de trabajo del dispositivo brindan la máxima flexibilidad. El módulo tiene dos modos: comunicación I2C y modo de comunicación SPI. Utilizaremos el sistema de comunicación mediante bus I2C para el envío de datos.

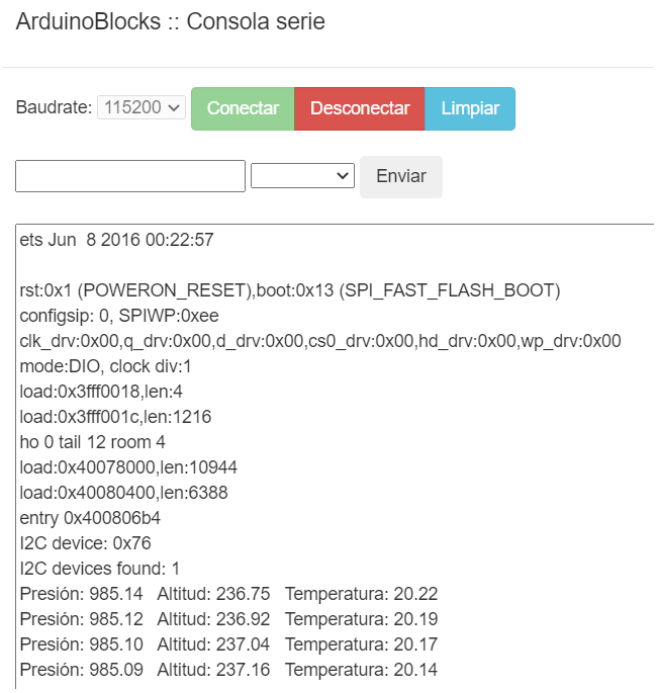
ArduinoBlocks dispone de un bloque específico para poder trabajar con este sensor.



Realizaremos un programa para medir la presión atmosférica, la temperatura y la altitud, y veremos los datos por la *Consola*.



Enlace al programa: ArduinoBlocks Projects\bmp280.abp



Actividad de ampliación: realiza el programa anterior y comprueba su funcionamiento.

7.12.11 Reto A12.11. Sensor giroscopio y acelerómetro MPU6050.

Reto A12.11. Sensor giroscopio y acelerómetro MPU6050.

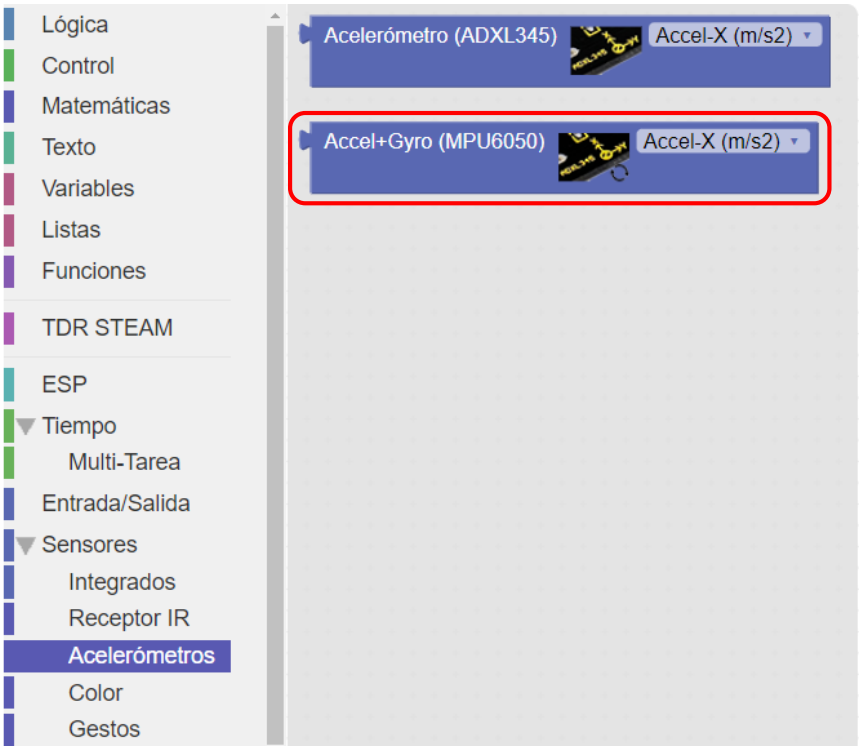
El sensor MPU6050 es un sensor de 9 ejes Motion Tracking que combina un giroscopio MEMS de 3 ejes, un acelerómetro MEMES de 3 ejes y un procesador de movimiento digital (DMP). Acepta directamente las entradas de una brújula externa de 3 ejes para proporcionar una salida completa de 9 ejes.

El MPU-6050 dispone tres convertidores analógicos/digitales (ADC) de 16 bits para digitalizar las salidas del giroscopio y tres ADC de 16 bits para digitalizar las salidas del acelerómetro.

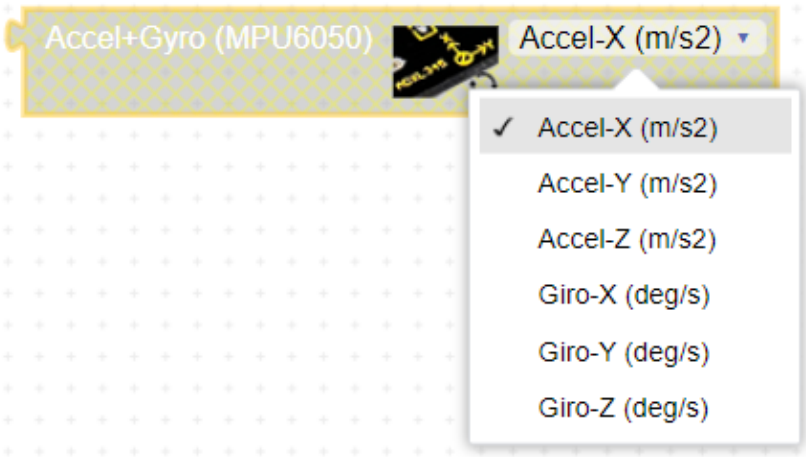
Para un seguimiento de precisión de movimientos rápidos y lentos, las piezas cuentan con un rango de escala de giroscopio programable por el usuario de ± 250, ± 500, ± 1000 y ± 2000 °/seg (dps) y un acelerómetro programable por el usuario de escala completa de rango de ± 2g, ± 4g, ± 8g y ± 16g.

Se comunica directamente mediante bus I2C a cualquier sistema de control. Se alimenta mediante tensión comprendida entre 2.4V y 3.6V, aunque la mayoría de los módulos disponen de un sistema de regulación que permite conectarlos directamente a 5V, como sucede en este caso.

ArduinoBlocks dispone de un bloque específico para poder trabajar con este sensor.



Se puede acceder a 6 datos diferentes dentro de este bloque, la aceleración en los 3 ejes y el giroscopio en los 3 ejes.



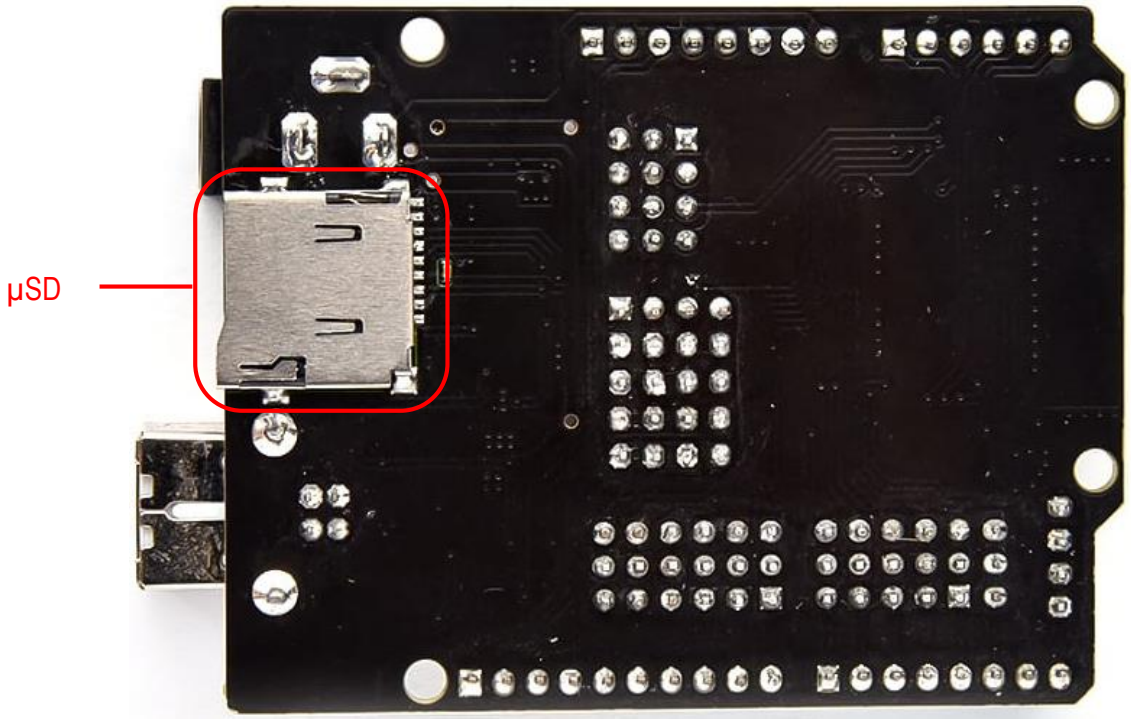
7.13 Reto A13. Tarjeta SD.

Reto A13. Tarjeta SD.

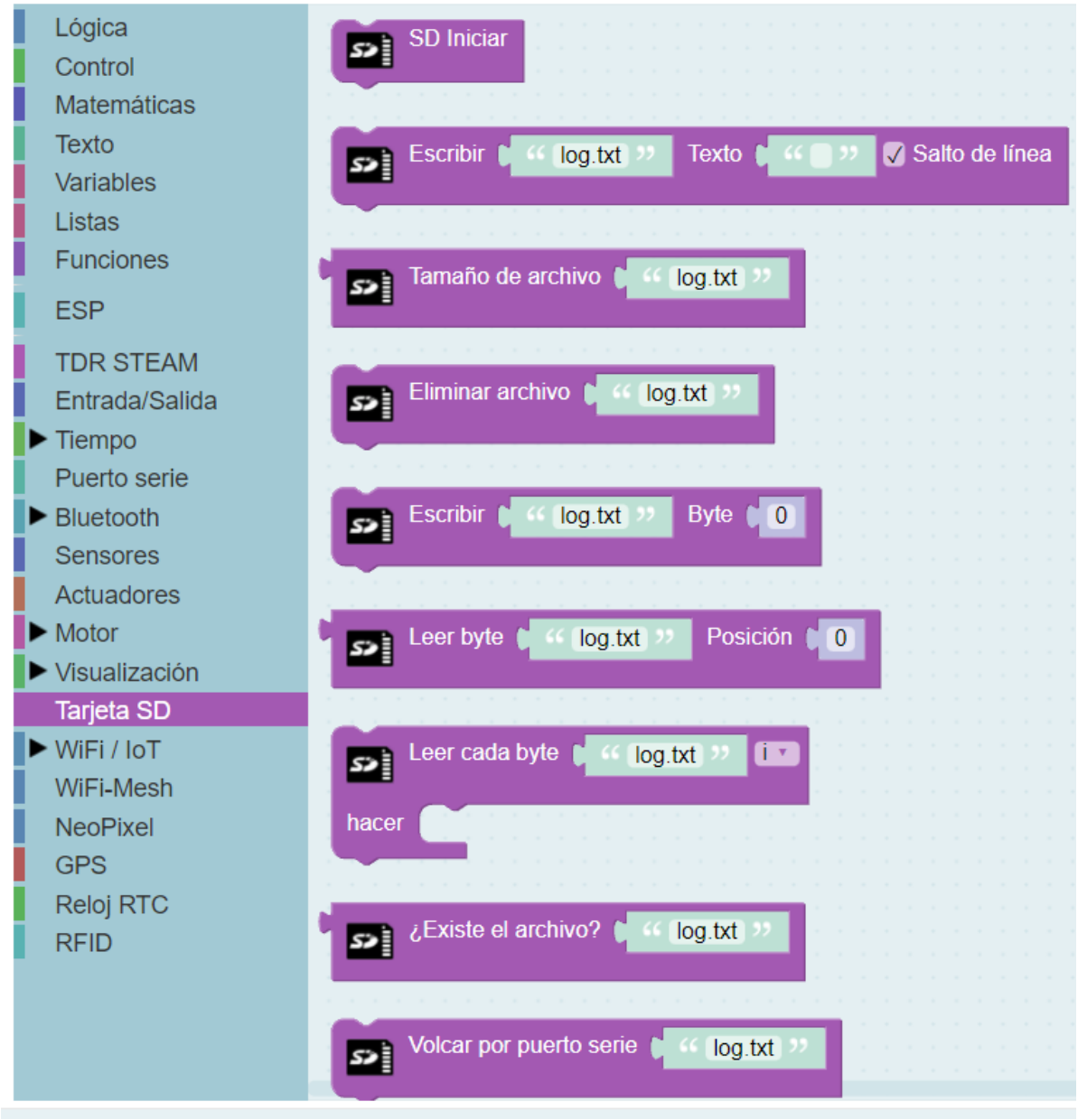
La placa ***ESP32 Plus STEAMakers*** dispone de un zócalo para poder insertar tarjetas micro-SD. Para que funcione tiene que estar formateada en formato FAT32 permitiendo tamaños de hasta 2TB y archivos de un tamaño máximo de 2GB. La tarjeta le da gran flexibilidad a la placa ya que permite poder escribir y leer datos. Una de las funcionalidades más útiles es poder almacenar los datos de los sensores a modo de *Datalogger*. También nos permitirá leer datos de la tarjeta (por ejemplo, configuraciones de conexiones Wifi) para poder utilizarlas directamente en un programa.

El zócalo para introducir la tarjeta micro SD se encuentra en la parte inferior de la placa, como podemos ver en la imagen siguiente.

Nota importante: *el tipo de tarjeta de memoria SD que podemos conectar es de* ***máximo 32GB y formateada en FAT32****, sin son de una capacidad mayor pueden no funcionar.*



Existen una serie de bloques específicos en ArduinoBlocks para controlar la funcionalidad de la tarjeta SD.



Realizaremos una serie de actividades para familiarizarnos con el manejo de la tarjeta SD y la placa ***ESP32 Plus STEAMakers***. Utilizaremos los bloques básicos que nos permitirán almacenar y leer datos de la SD.

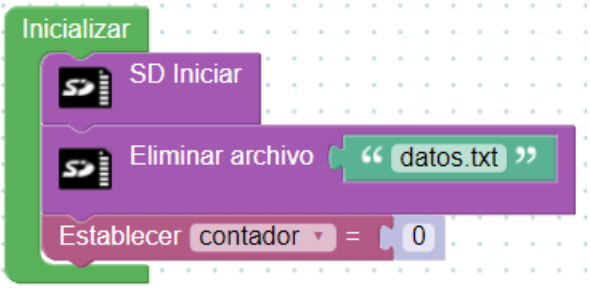
7.13.1 Reto A13.1. Almacenar datos: Datalogger.

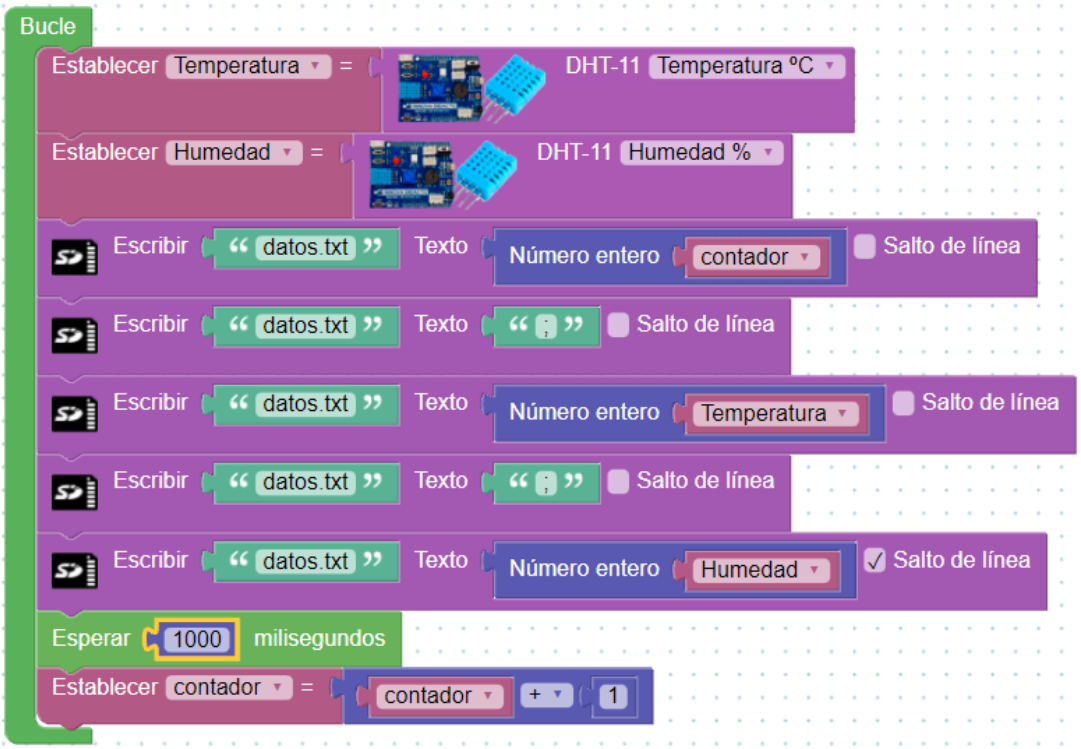
Reto A13.1. Almacenar datos: Datalogger.

Vamos a realizar un programa que nos permita almacenar los valores de temperatura y humedad en un fichero.

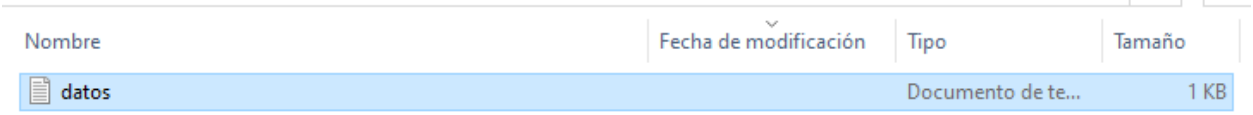
Primero inicializaremos la tarjeta SD y borraremos el archivo que pueda haber anteriormente. A continuación, leeremos la temperatura y la humedad y la almacenaremos en el fichero de texto *datos.txt*. Para separar los datos añadiremos el símbolo ";" para poder extraer luego los datos de forma más sencilla. Tomaremos las muestras en intervalos de 1 segundo.

El programa que realizaremos es el siguiente:

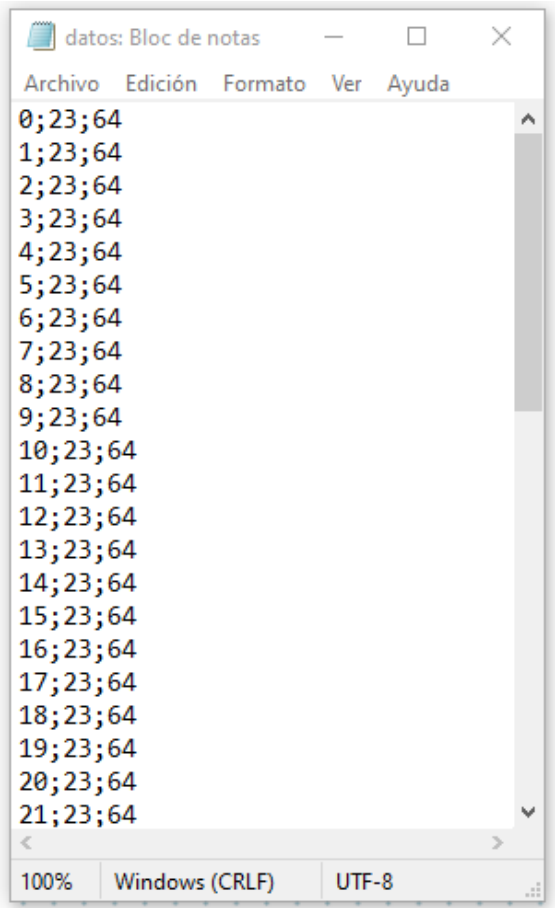




Una vez abramos la tarjeta en nuestro ordenador observaremos que se ha creado el fichero con el nombre que le hemos asignado.



Y si abrimos el fichero veremos la secuencia de muestras que hemos tomado y el formato de números obtenido.



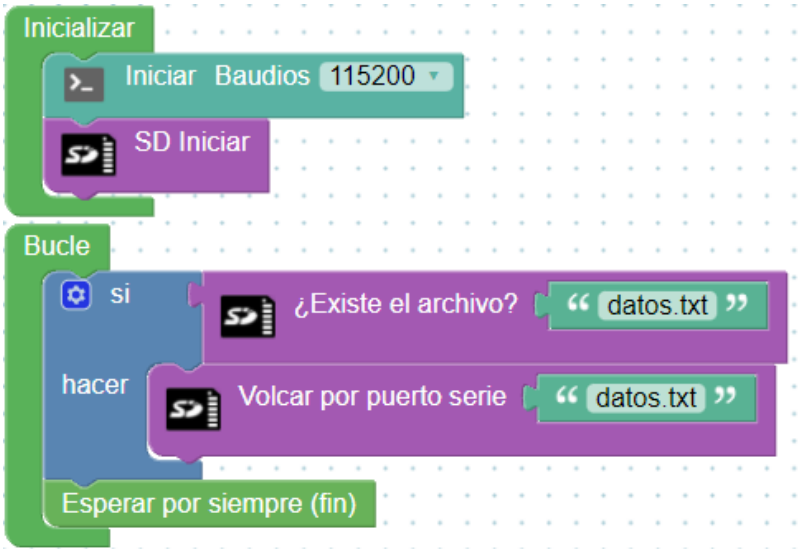
Actividad de ampliación: realiza el programa anterior y comprueba su funcionamiento.

7.13.2 Reto A13.2. Leer datos almacenados.

Reto A13.2. Leer datos almacenados.

Ahora vamos a leer los datos que hemos almacenado en la actividad anterior. Vamos a realizar un programa que nos permita volcar todos los datos por el puerto serie.

El programa resultante es el siguiente:



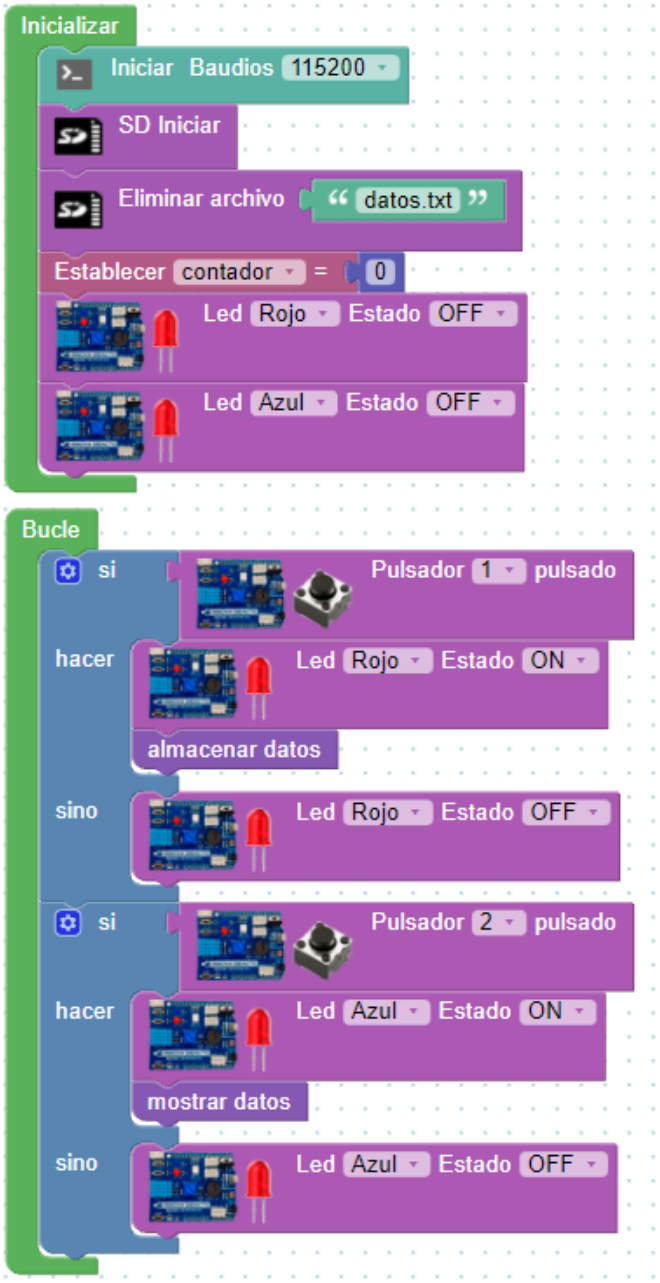
Los datos obtenidos por la consola son los que hay almacenados en el fichero de la tarjeta SD.

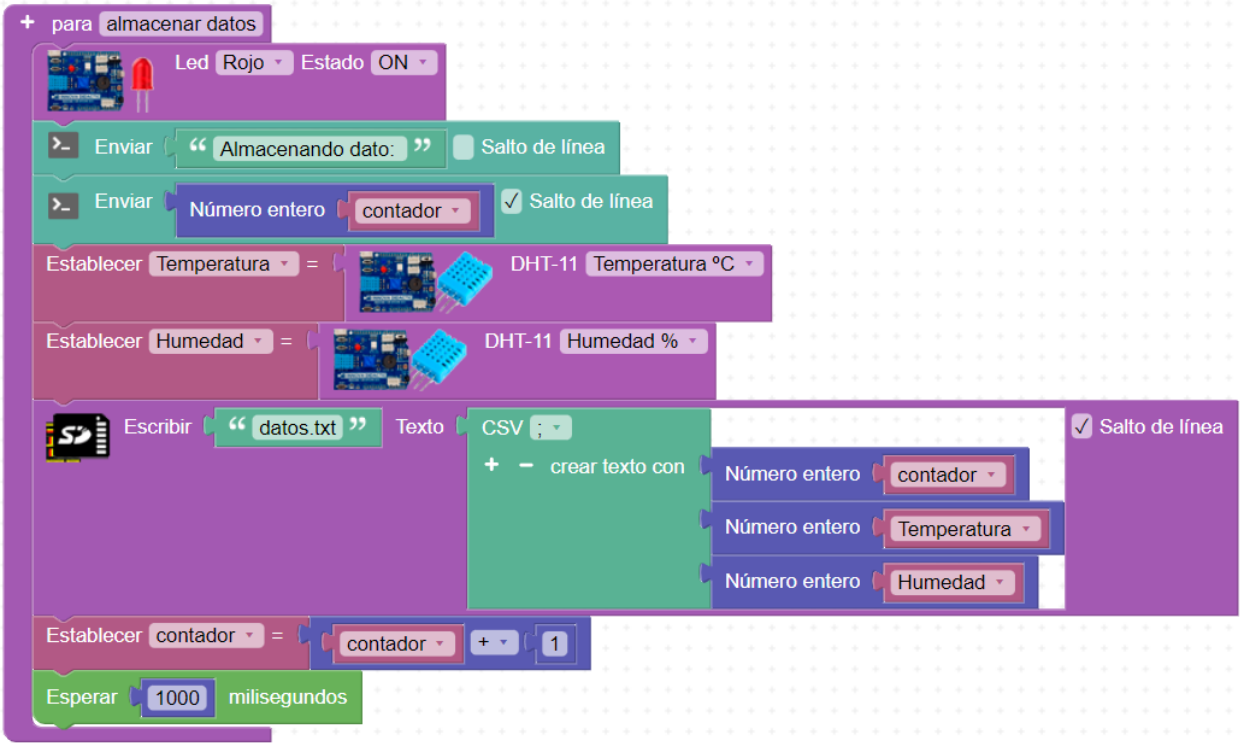
Actividad de ampliación: realiza el programa anterior y comprueba su funcionamiento.

7.13.3 Reto A13.3. Escribir y leer datos.

Reto A13.3. Escribir y leer datos.

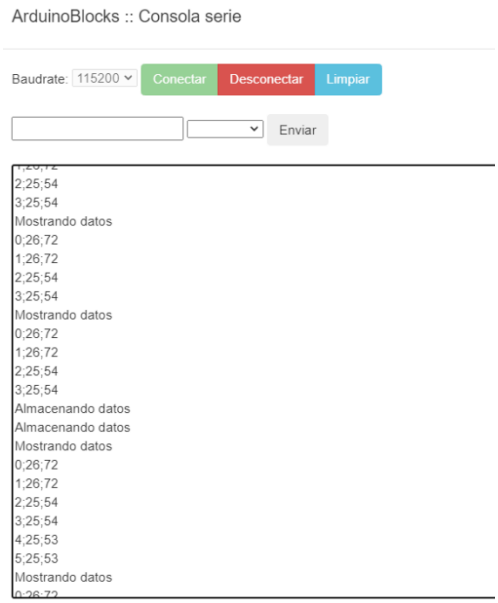
A continuación, vamos a realizar un programa que reúna las funcionalidades vistas anteriormente. Utilizaremos un pulsador para iniciar la grabación de los datos y otro pulsador para parar la grabación. En el momento que se pare la grabación de datos, los mostrará por la *Consola*. Mientras que esté almacenando datos se encenderá el led rojo y cuando los muestre por la *Consola* se encenderá el led azul. Cada vez que apretemos el pulsador de almacenar datos se irán añadiendo a las muestras que ya tenemos registradas.



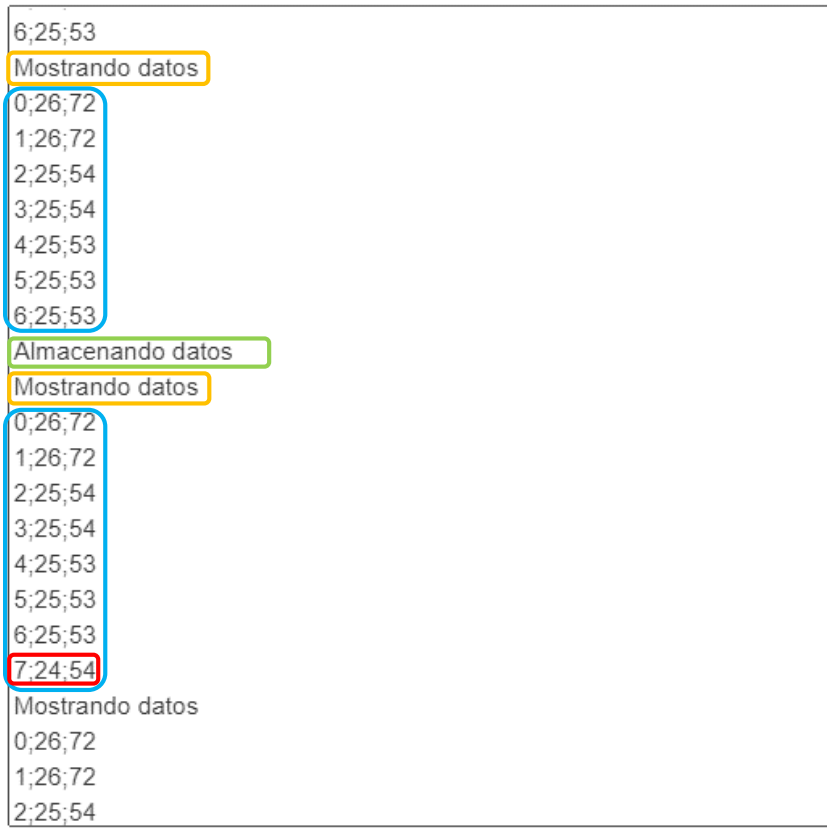


Enlace al programa: ArduinoBlocks Projects\sd_1.abp

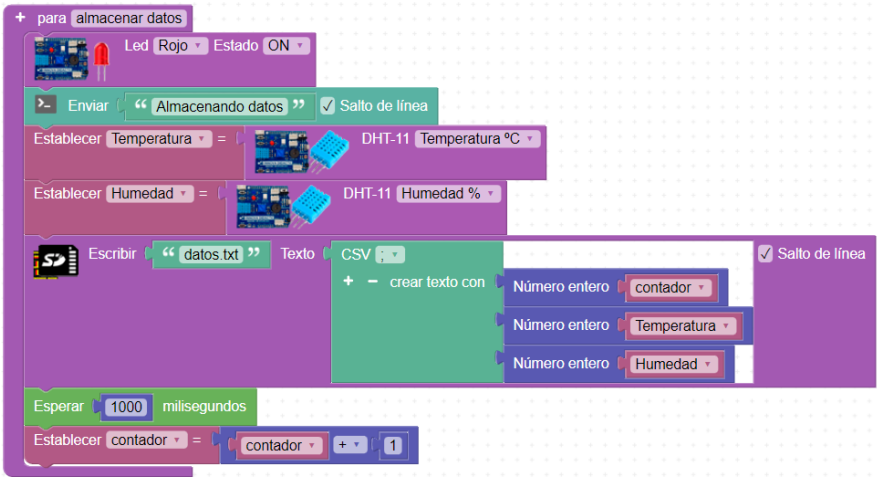
La información que veremos por la consola será la siguiente (cada vez que abrimos la Consola se reinicia el programa y empieza a coger muestras desde 0):



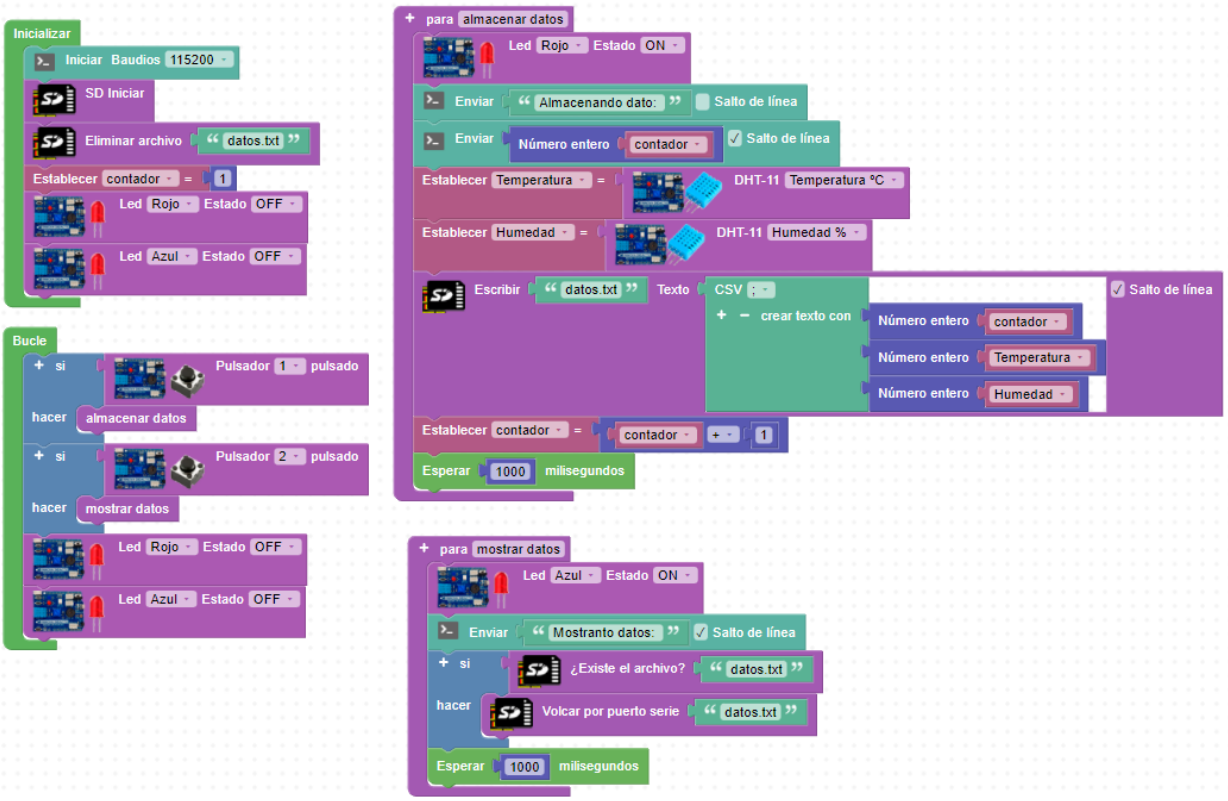
Como podemos observar en la imagen siguiente, cada vez que almacena una muestra (Almacenando datos) se añade una nueva muestra al fichero de datos. En el ejemplo tenemos 6 muestras y, cuando pulsamos el botón de almacenar, almacena 1 nueva muestra. Al pulsar el botón de mostrar, ahora aparecen 7 muestras.



Podemos mejorar el programa si en lugar de escribir cada dato en la SD (que abre y cierra el fichero datos.txt) lo hacemos de una vez. La función de almacenar datos quedaría de la siguiente forma:



El programa definitivo quedaría de la siguiente forma:



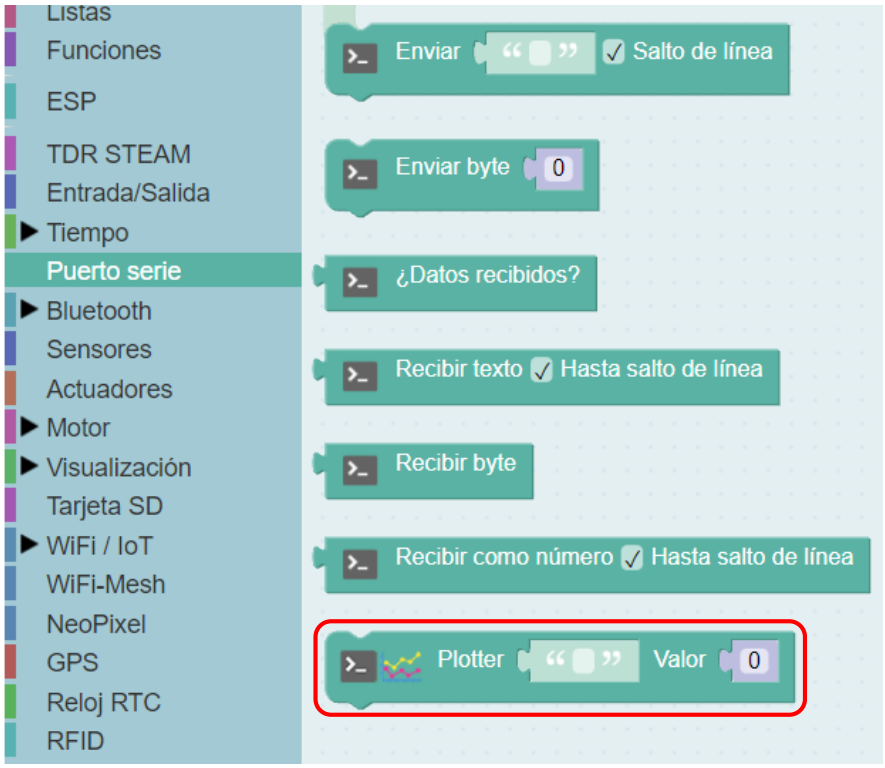
Actividad de ampliación: realiza el programa anterior y comprueba su funcionamiento.

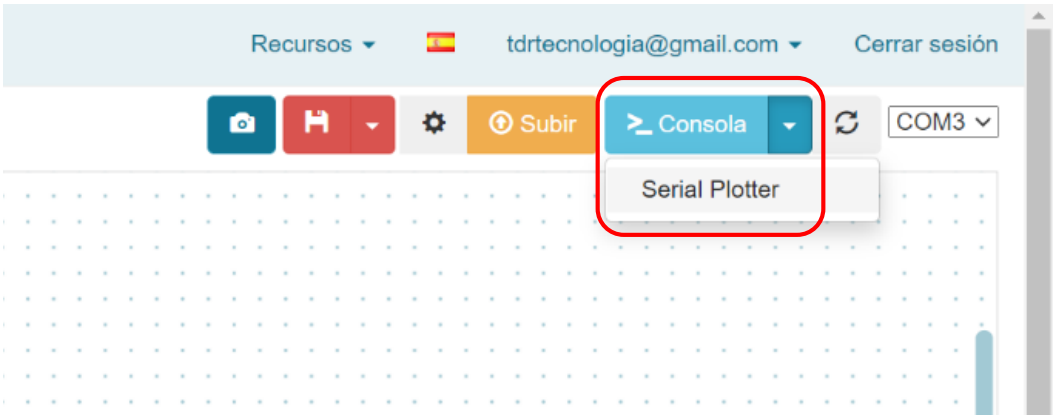
7.14 Reto A14. Serial Plotter.

Reto A14. Serial Plotter.

ArduinoBlocks dispone de una funcionalidad para poder ver representada de forma gráfica los valores de sensores, variables numéricas, etc. Esta funcionalidad es el Serial Plotter. También permite descargar en formato CSV el primero de los sensores de la gráfica.

En el apartado de *Puerto serie* disponemos de un bloque para poder visualizar datos.





7.14.1 Reto A14.1. Serial plotter con un sensor.

Reto A14.1. Serial plotter con un sensor.

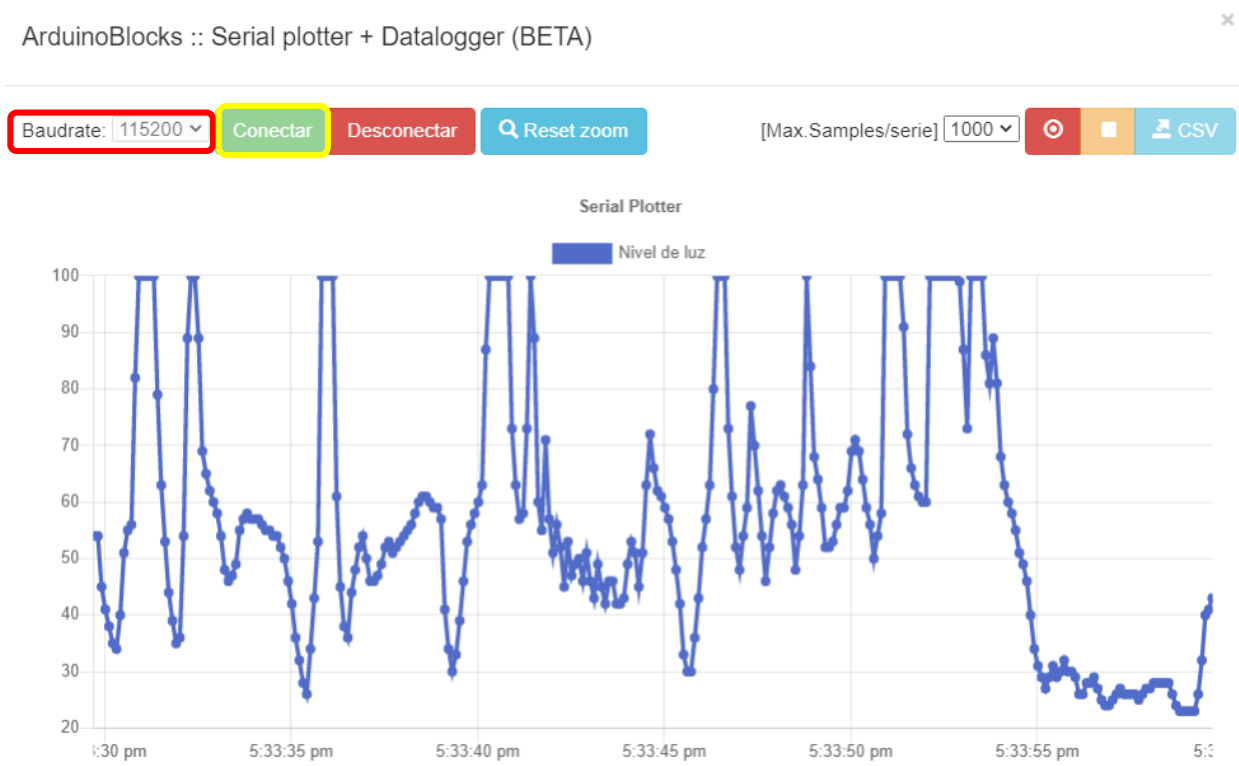
Vamos a realizar un programa muy sencillo e interesante que nos permitirá ver los datos del potenciómetro en forma de gráfica y los podremos exportar en formato CSV para poder tratarlos posteriormente.

Con este programa conseguiremos realizar un sistema de adquisición de datos de la LDR para poder ver los datos de forma gráfica. Este es el programa en ArduinoBlocks que hemos confeccionado.

Enlace al programa: ArduinoBlocks Projects\plotter_1.abp

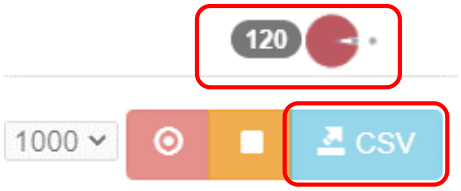
Enviamos el programa a la placa y activamos el *Serial Plotter* a través de la pestaña que hay junto a *Consola*.

Pondremos la velocidad de comunicación (*baudrate*) a 115200 y después pulsaremos ***Conectar*** para empezar a ver los datos.

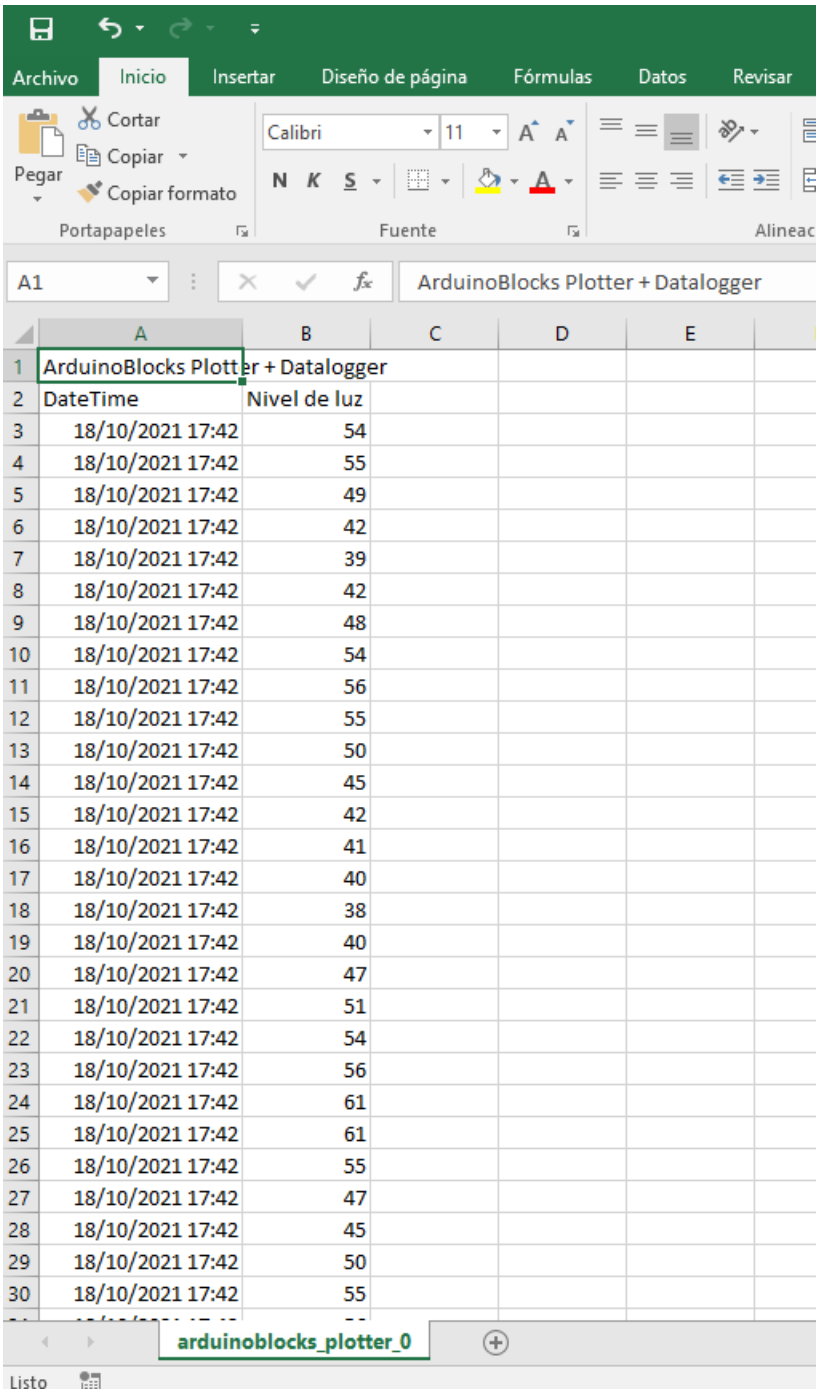


Para poder guardar los datos en CSV hemos de apretar el botón de grabación, adquirir los datos que queremos y apretar el botón de parar grabación.

Podremos ver la cantidad de muestras recogidas.



Ahora pulsamos en el botón CSV para guardar los datos en nuestro ordenador y poder trabajar con el fichero de datos creado.



	A	B
1	ArduinoBlocks Plotter + Datalogger	
2	DateTime	Nivel de luz
3	18/10/2021 17:42	54
4	18/10/2021 17:42	55
5	18/10/2021 17:42	49
6	18/10/2021 17:42	42
7	18/10/2021 17:42	39
8	18/10/2021 17:42	42
9	18/10/2021 17:42	48
10	18/10/2021 17:42	54
11	18/10/2021 17:42	56
12	18/10/2021 17:42	55
13	18/10/2021 17:42	50
14	18/10/2021 17:42	45
15	18/10/2021 17:42	42
16	18/10/2021 17:42	41
17	18/10/2021 17:42	40
18	18/10/2021 17:42	38
19	18/10/2021 17:42	40
20	18/10/2021 17:42	47
21	18/10/2021 17:42	51
22	18/10/2021 17:42	54
23	18/10/2021 17:42	56
24	18/10/2021 17:42	61
25	18/10/2021 17:42	61
26	18/10/2021 17:42	55
27	18/10/2021 17:42	47
28	18/10/2021 17:42	45
29	18/10/2021 17:42	50
30	18/10/2021 17:42	55

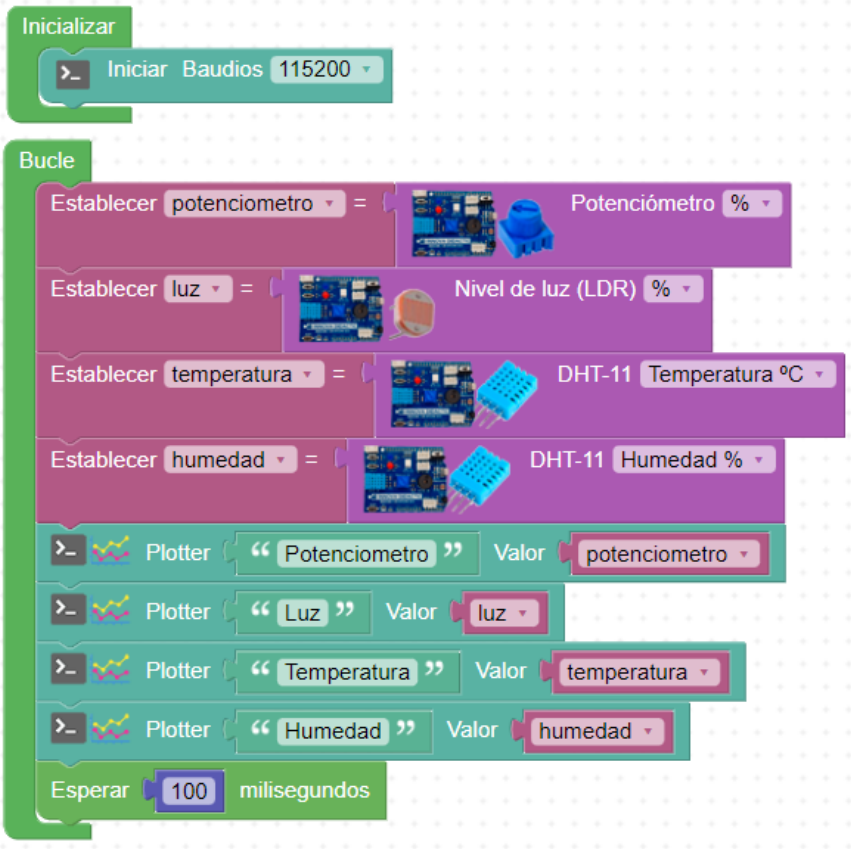
Actividad de ampliación: modifica el programa para que muestre los datos ahora de otro sensor.

7.14.2 Reto A14.2. Serial plotter con varios sensores.

Reto A14.2. Serial plotter con varios sensores.

Vamos a realizar un programa que permita ver 4 sensores a la vez. Para ello deberemos tener en cuenta las escalas en las que trabaja cada sensor.

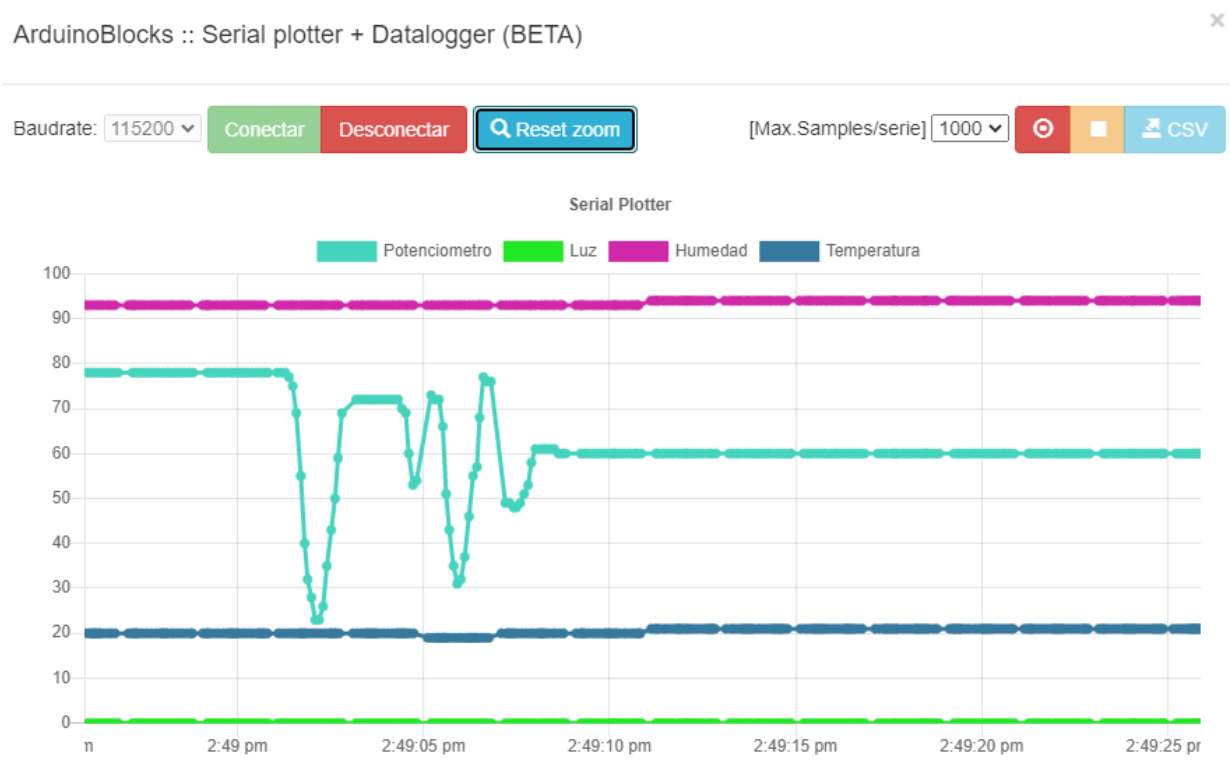
Adquiriremos los valores del potenciómetro, el nivel de luz de la LDR y los valores de humedad y temperatura del sensor DHT-11. Para estar en escalas aproximadas los valores del potenciómetro y la luz las trataremos en % (valores entre 0 y 100) y la humedad y temperatura en sus valores normales (humedad entre 0 y 100, y temperatura entre 0 y 40 aprox.).



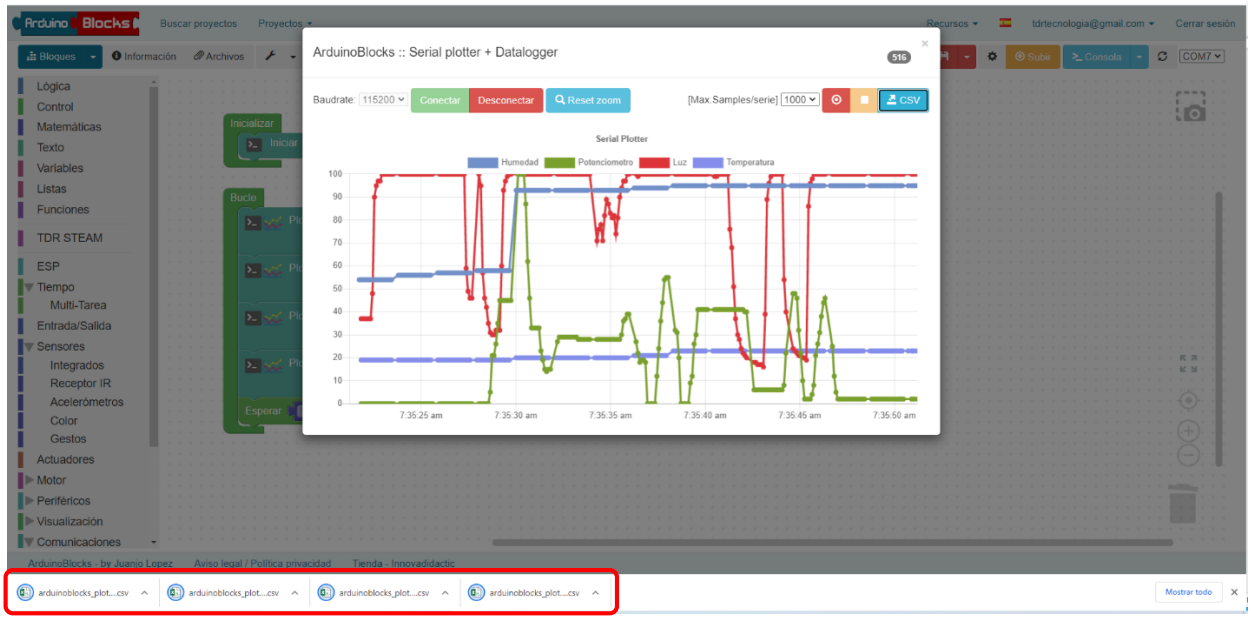
Enlace al programa: ArduinoBlocks Projects\plotter_4var.abp

Recordatorio: Cuando se realicen las pruebas deberemos poner a 0 (posición izquierda) el potenciómetro para volver a enviar el programa a la placa, sino dará un mensaje de error y no subirá el programa.

La gráfica obtenida es la siguiente:



También podemos almacenar los datos de los cuatro sensores para realizar una exportación a CSV. Creará tantos ficheros CSV como sensores tengamos.

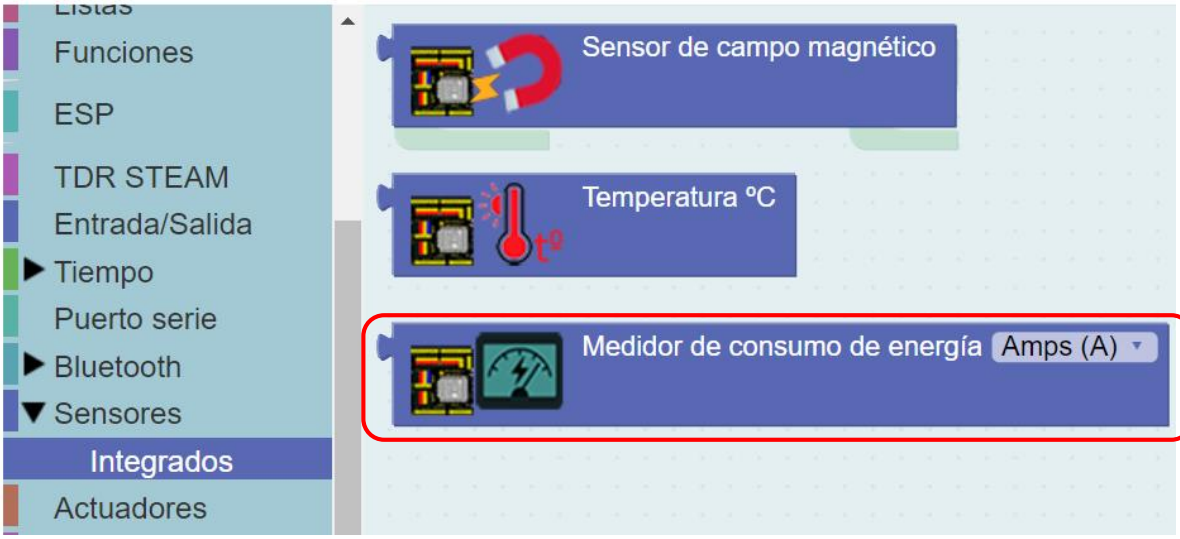


Actividad de ampliación: realiza un programa similar al anterior donde se puedan ver los valores de otros sensores (puedes ajustar las escalas de los sensores para que la gráfica sea legible).

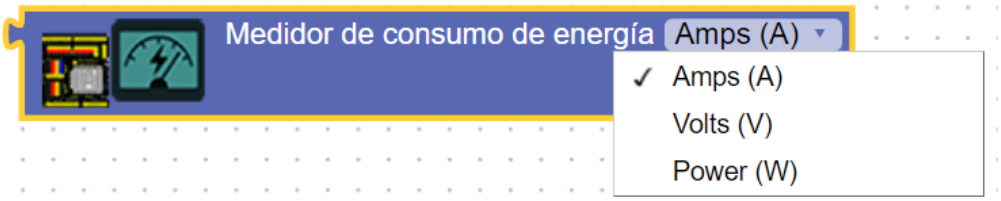
7.15 Reto A15. Consumo de energía.

Reto A15. Consumo de energía.

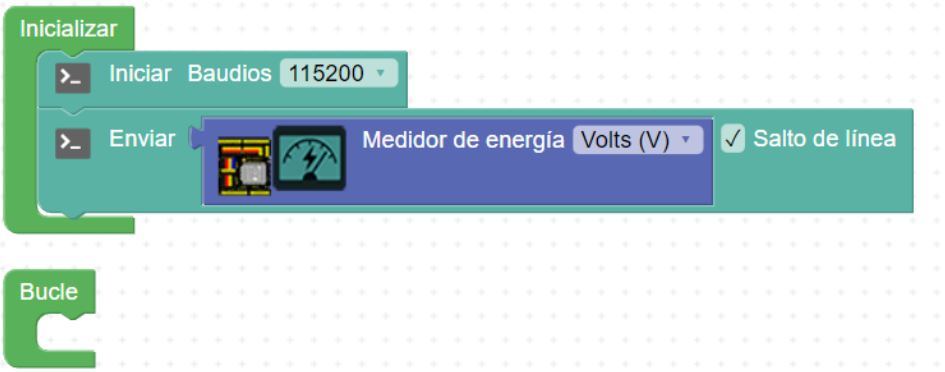
La placa ***ESP32 Plus STEAMakers*** dispone de un sistema para poder medir el consumo de energía. El medidor de consumo de energía está dentro de la función de *Sensores Integrados*.



Este bloque permite medir tanto la intensidad (Amps) como la tensión (Volts) y, también, el cálculo de la potencia (Power).



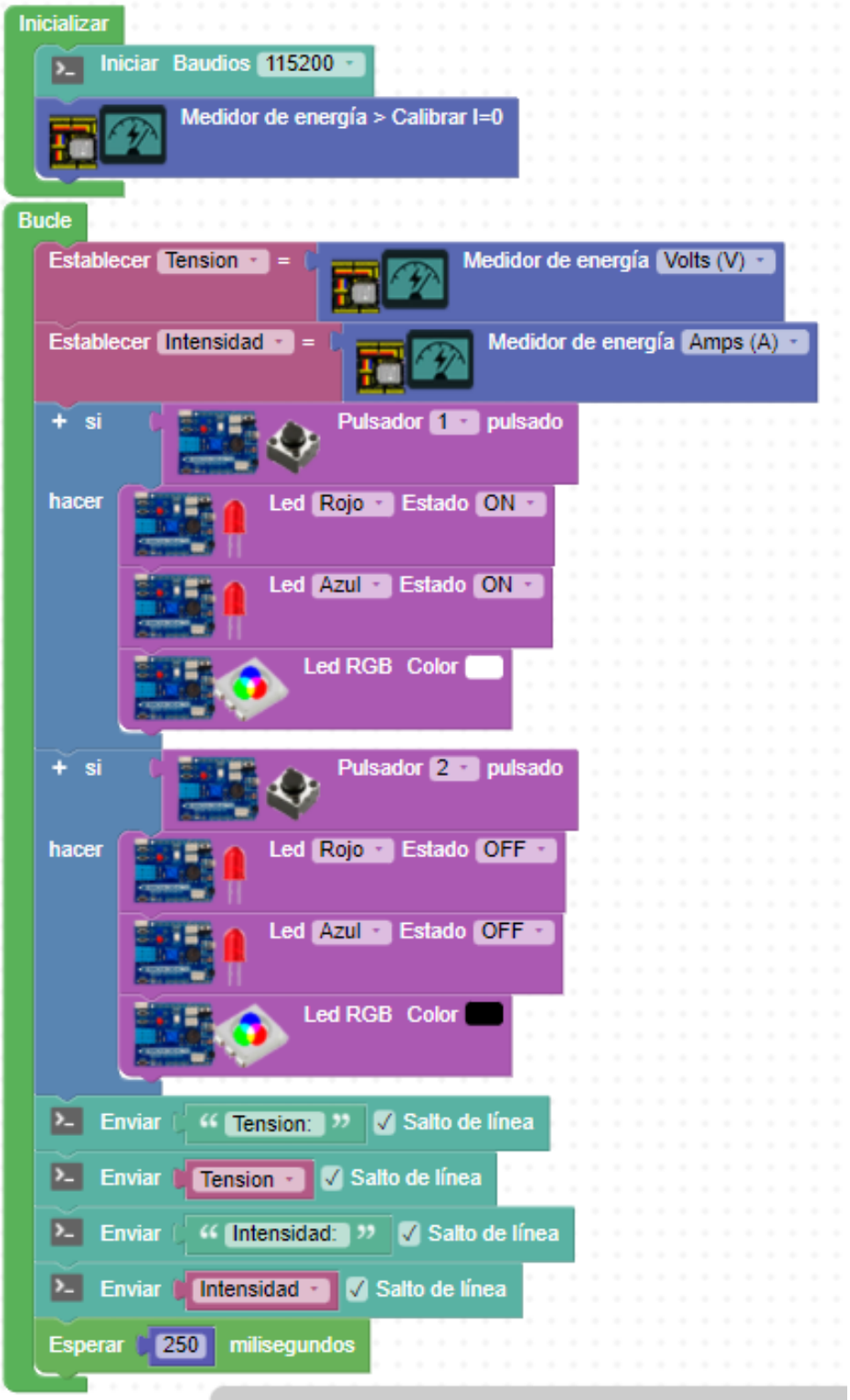
Enlace al programa: ArduinoBlocks Projects\alimentacion.abp



7.15.1 Reto A15.1. Monitorización del consumo de energía.

Reto A15.1. Monitorización del consumo de energía.

Realizaremos un sencillo programa para poder saber los tres parámetros.



Enlace al programa: ArduinoBlocks Projects\consumo_energia.abp

En la *Consola* podemos ver los valores de estos tres parámetros.

- Si no tenemos nada conectado, el valor de intensidad será 0.

- En el momento de encender los dos leds y el led RGB se producirá una pequeña variación en la intensidad.

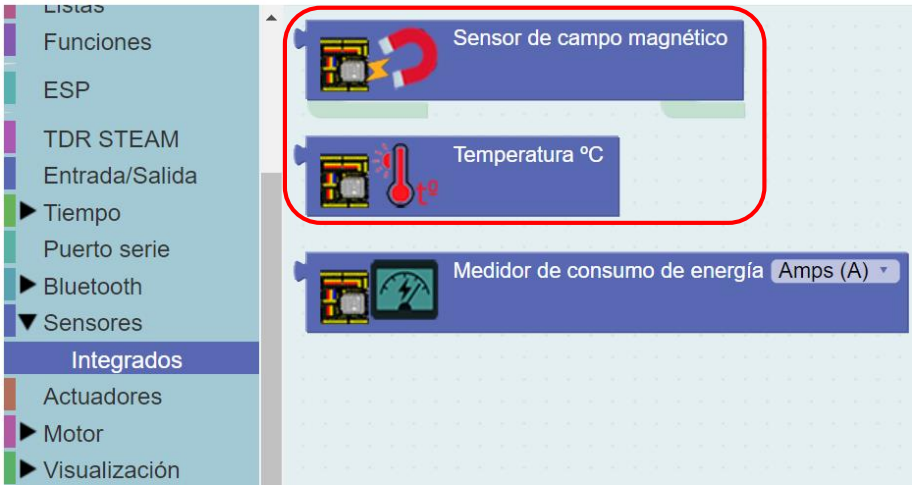
Actividad de ampliación: realiza el programa anterior y comprueba su funcionamiento.

7.16 Reto A16. Sensores internos.

Reto A16. Sensores internos.

La placa ***ESP32 Plus STEAMakers*** al estar basada en el **ESP32-WROOM-32** dispone de dos sensores integrados. Se trata de un sensor de efecto hall y un sensor de temperatura. Los bloques para utilizarlos están en *Sensores Integrados*.

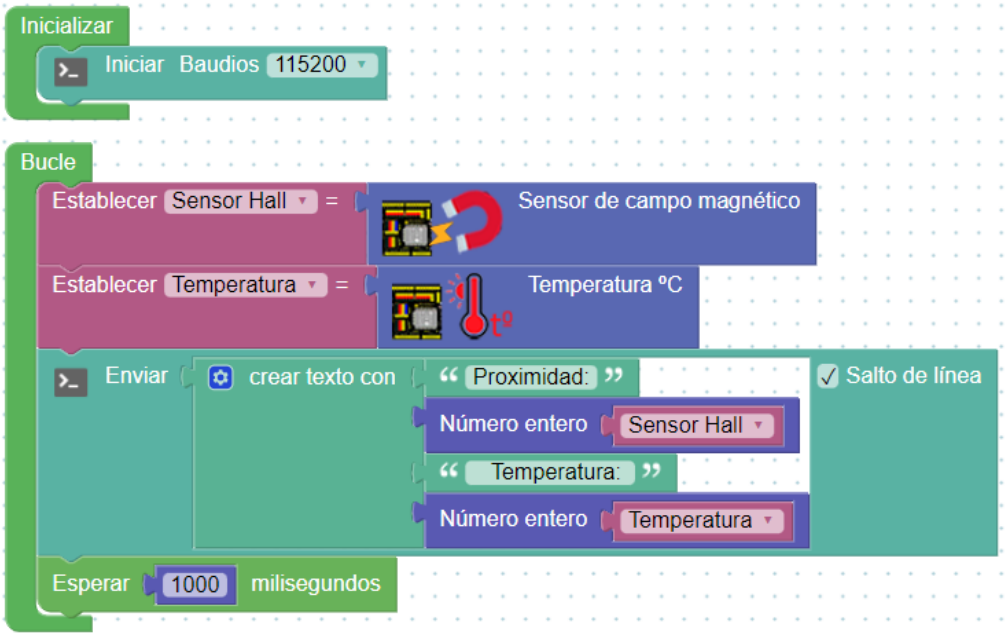
Con el sensor de efecto hall podremos detectar variaciones de campo magnético en la proximidad de la placa. Con el sensor de temperatura podremos controlar la temperatura a la que se encuentra el procesador de la placa.



7.16.1 Reto A16.1. Lectura de los sensores internos del ESP32.

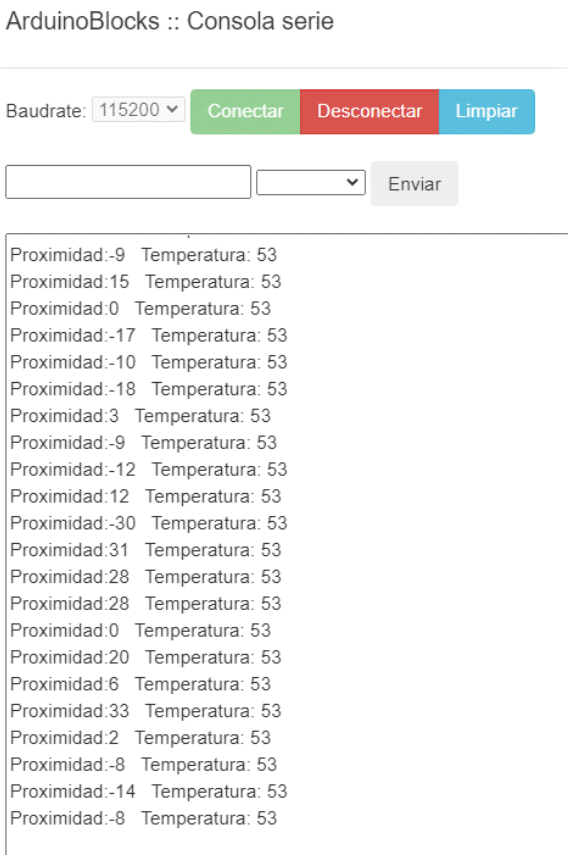
Reto A16.1. Lectura de los sensores internos del ESP32.

Realizaremos un programa para observar los valores de estos dos sensores.



Enlace al programa: ArduinoBlocks Projects\sensores_internos.abp

En la Consola podemos ver los valores de la temperatura de la placa y si tenemos próximo un objeto metálico.



Podemos observar que al colocar un objeto metálico sobre el ESP32 el valor de proximidad aumenta y al quitarlo disminuye.

Actividad de ampliación: realiza el programa anterior y comprueba su funcionamiento.

7.17 Reto A17. Multitarea.

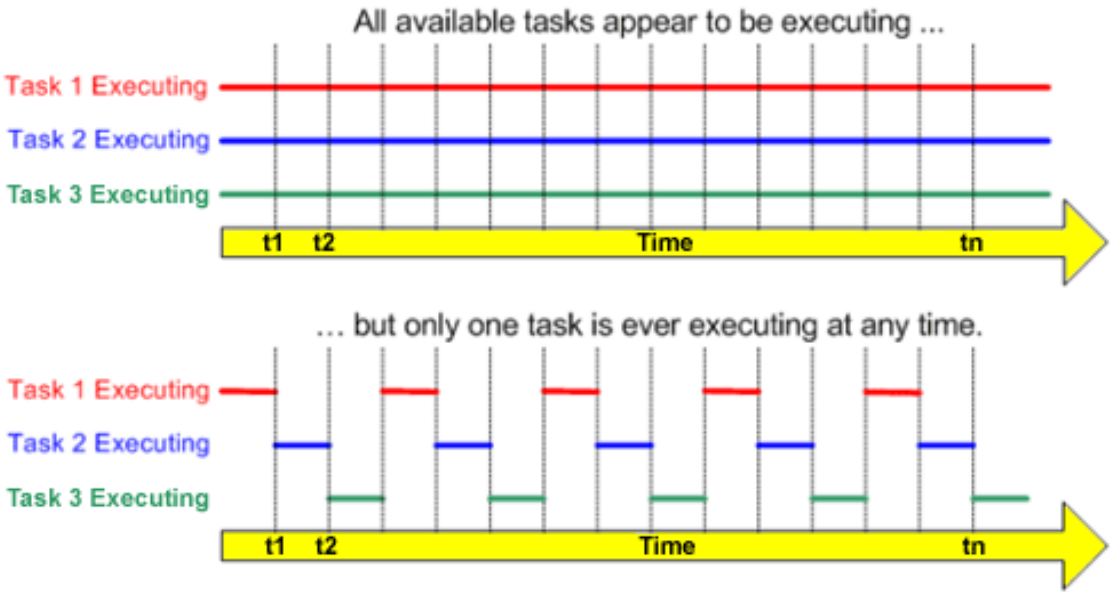
Reto A17. Multitarea.

Vamos a explicar de forma más profunda cómo funciona el sistema multitarea. La opción que usan los sistemas multitarea con un sólo procesador (por ejemplo, los ordenadores que hasta no hace muchos años sólo tenían un núcleo) es implementar una capa por software que se encargue de gestionar el tiempo del microprocesador y dar a cada tarea el tiempo que crea que es necesario para que se ejecute correctamente. Esta tarea la realiza el Sistema Operativo. Algunas placas como Arduino UNO o el ESP32 no tienen un sistema operativo, pero sí podemos añadir una librería que nos haga las funciones de un sistema operativo, por lo menos en cuanto a la gestión del tiempo se refiere.

Existen diferentes implementaciones para gestionar una multitarea más o menos real, pero siempre tenemos que tener en cuenta la capacidad limitada del microcontrolador que estemos utilizando. Si le pedimos más de lo que puede procesar, el microcontrolador no dará a basto y no podrá ejecutar todas las tareas en tiempo real e incluso podrá llegar a reiniciarse.

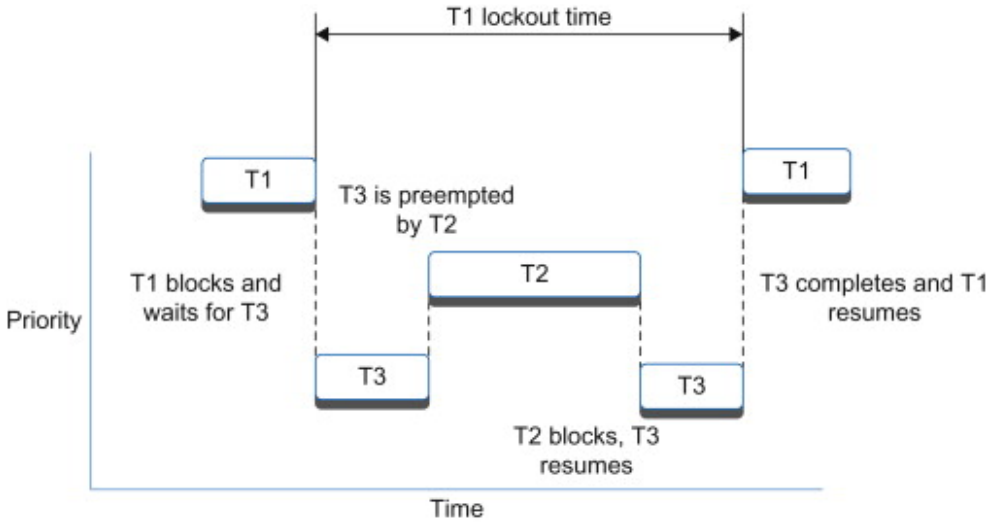
Los sistemas software de multitarea (a falta de tener varios procesadores como en el caso del ESP32-WROOM-32 en el que se basa la placa ***ESP32 Plus STEAMakers***, que dispone de dos núcleos) utilizan un planificador (*scheduler*) que se encarga de repartir el tiempo de procesamiento entre las distintas tareas, de forma que a cada una le toca un fragmento del tiempo de ejecución de esa parte de programa. El planificador, en caso de disponer de varios núcleos, también se encarga de repartir las tareas entre ellos.

A continuación, podemos ver en la siguiente imagen un esquema con 3 tareas que se van alternando, y en apariencia parece que las 3 se ejecutan a la vez. En este caso, el tiempo de procesador se divide igual para cada tarea.



Los planificadores de multitarea permiten además asignar a cada tarea una prioridad, para así darle preferencia a las tareas más críticas o que necesitan más tiempo de procesamiento. Si creamos muchas tareas con alta prioridad puede que afectemos a las demás dejando poco tiempo de procesamiento para ellas.

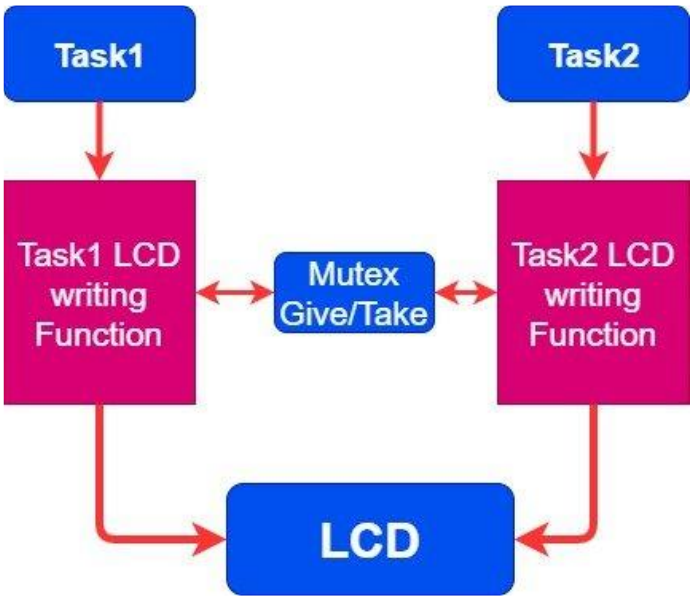
Podemos observar en la siguiente imagen un esquema de varias tareas con distintas prioridades, variando así su tiempo de microprocesador asignado.



Cada tarea tiene su propio espacio de memoria, por lo que crear demasiadas tareas también puede dejarnos el procesador sin memoria. Si la memoria asignada a las tareas tampoco es suficiente para almacenar los datos se podría reiniciar de forma inesperada la placa o funcionar incorrectamente. Siempre hay que ser consciente de los limitados recursos de los que disponemos.

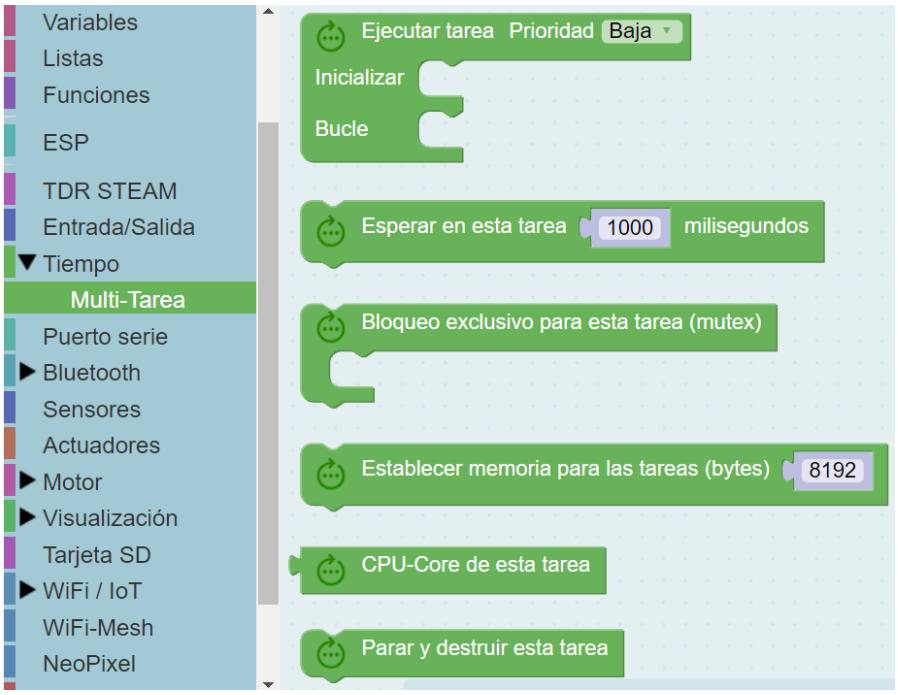
Existen unos semáforos para dar prioridad a unas tareas respecto a otras. El que más nos interesa es el semáforo "*mutex*" o de exclusión mutua, que permite que bloqueemos el sistema multitarea, realizaremos la tarea crítica y, a continuación, liberemos otra vez el control. Por supuesto, estas tareas críticas deben ser lo más cortas posibles: una escritura crítica en una variable, el envío de un dato, una actualización de una pantalla LCD, etc., siempre cosas simples. Los semáforos debemos usarlos en casos que tengamos claro que se pueden crear conflictos, pues su abuso puede hacer que el sistema multitarea empiece a fallar.

Esquema de acceso a un mismo recurso por parte de 2 tareas diferentes:

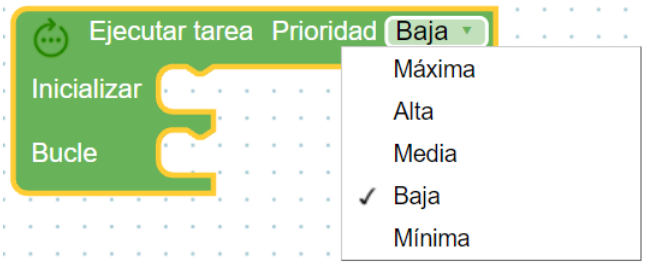


Siempre, en programación, hay que evitar los bloques de tipo *espera*, ya que hace que el microcontrolador no está realizando ninguna tarea, pero el sistema multitarea le asignará un tiempo de procesamiento a esa tarea que no hace nada. Para solucionar este problema existe una tarea que le dice al controlador que va a estar un rato sin hacer nada, pero que luego tiene que volver a esa tarea.

ArduinoBlocks dispone de una serie de bloques para poder trabajar con el sistema multitarea:

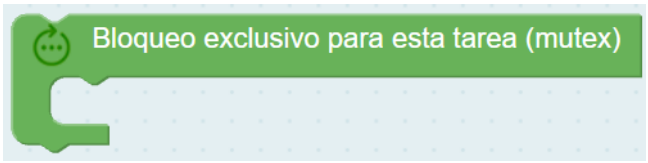


- Permite crear una nueva tarea con su bloque de *Inicializar* y de *Bucle* al igual que cuando utilizamos un programa normal. Debemos asignar una prioridad a cada tarea, por defecto es mejor dejarlas en *Baja* y luego se pueden ir ajustando si es necesario. Para gestionar mejor las prioridades, es recomendable en algunos casos no utilizar el *Inicializar* y *Bucle* principales, ya que suele tener preferencia sobre todas estas tareas y es más difícil de equilibrar las prioridades.

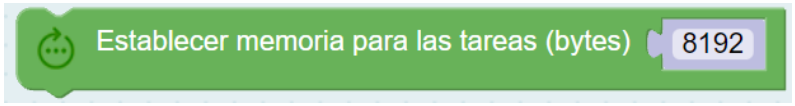


- Es el bloque de esperar óptimo para tareas, pues deja funcionar al resto de tareas de forma más eficiente mientras se espera en ésta. Este bloque tiene menos precisión que el bloque *Esperar* original. Si necesitamos hacer esperas muy precisas (o de menos de 20 ms) debemos usar el *Esperar* tradicional. Pero nos servirá en la mayoría de casos.

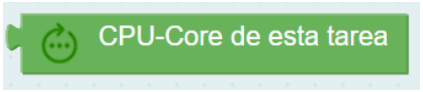
- Si tenemos que hacer alguna acción crítica que no queremos que sea interrumpida internamente por el planificador del sistema multitarea podemos poner este bloque y dentro los bloques críticos (no utilizar si no es estrictamente necesario).



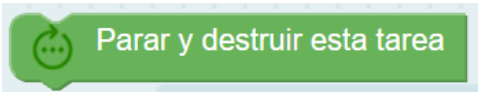
- Cada tarea tiene su propio espacio de memoria reservado, esta es la cantidad por defecto para las tareas (192 bytes), si necesitamos ajustarla podemos utilizar este bloque en el *Inicializar* principal y se ajustará para todas las tareas. Un mal ajuste puede provocar reinicios del microcontrolador o mal funcionamiento.



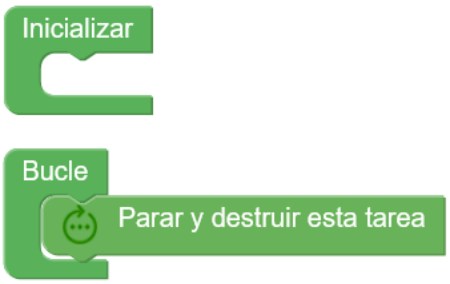
- Permite determinar que *CPU-Core* está realizando una tarea.



- Las tareas en principio, igual que el *Bucle* principal, están pensadas para ejecutarse de forma indefinida. Si una tarea deja de ser necesaria, la forma de terminarla es con este bloque, que parará la ejecución y liberará la memoria de la tarea en la que se ejecuta.



  Por ejemplo, si no queremos usar el bucle principal, podemos implementar todo nuestro programa en tareas y no usaremos la tarea principal (*Bucle*), eliminando esa tarea que se crea de forma automática al inicio.



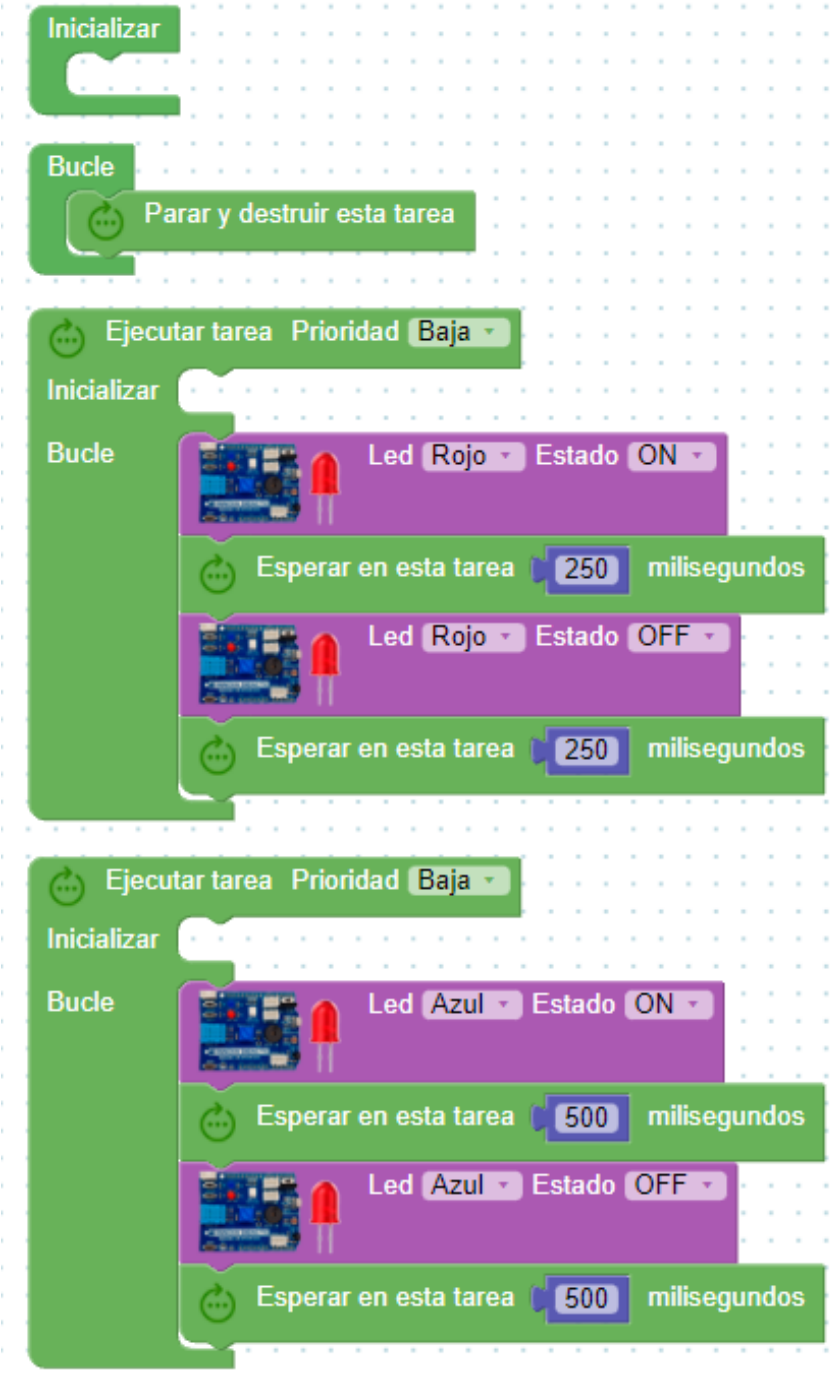
Atención: *El sensor el sensor de temperatura y humedad DHT11 necesita trabajar con una prioridad como mínimo Media para que realice correctamente las mediciones.*

Nota importante: *es necesario que cada multitarea tenga un bloque de* ***Esperar en esta tarea (tiempo)*** *para que se puedan ejecutar el resto de tareas. No es necesario destruir la tarea del* ***Bucle*** *principal.*

7.17.1 Reto A17.1. Multitarea I.

Reto A17.1. Multitarea I.

Vamos a realizar intermitencias de los dos leds que tiene la placa ***Imagina TDR STEAM***. Pararemos y destruiremos el *Bucle* principal y crearemos dos bloques de *Ejecutar tareas* con *Prioridad Baja*.

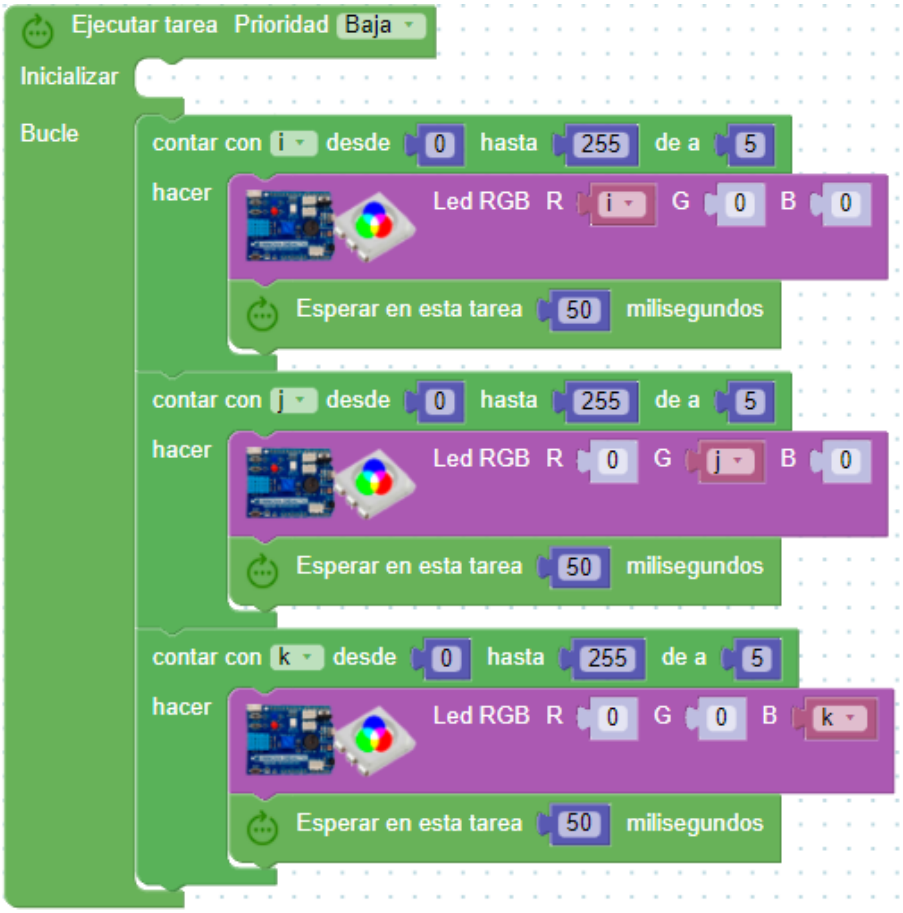


Actividad de ampliación: realiza el programa anterior y comprueba su funcionamiento.

7.17.2 Reto A17.2. Multitarea II.

Reto A17.2. Multitarea II.

Ahora vamos a añadir más elementos multitarea. Al programa anterior, vamos a añadirle el control del led RGB y enviar por el puerto serie a la *Consola* el valor de la LDR.

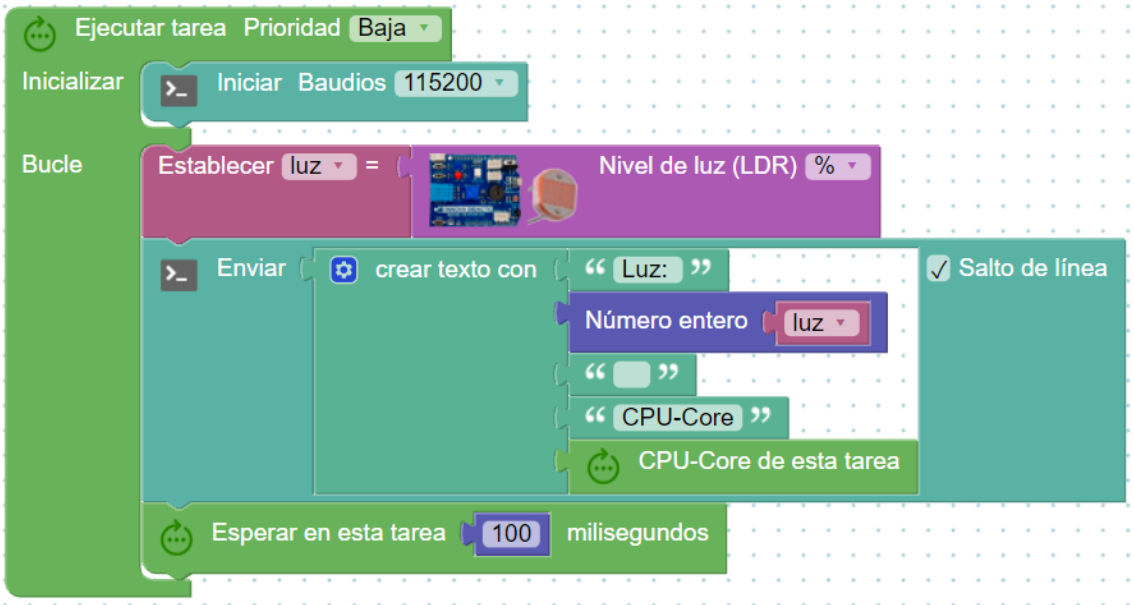


Actividad de ampliación: realiza el programa anterior y comprueba su funcionamiento.

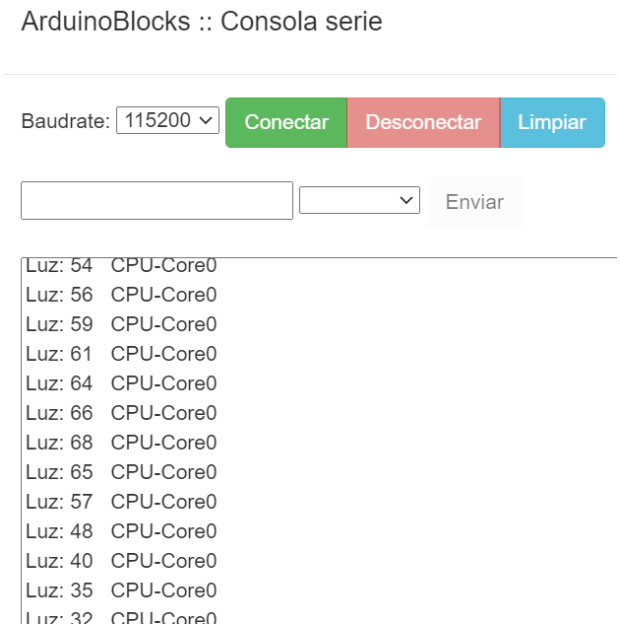
7.17.3 Reto A17.3. Multitarea III.

Reto A17.3. Multitarea III.

Modificaremos el último bloque para que nos muestre que *CPU-Core* está realizando esa tarea.



Los valores obtenidos en la *Consola serie* se muestran a continuación:



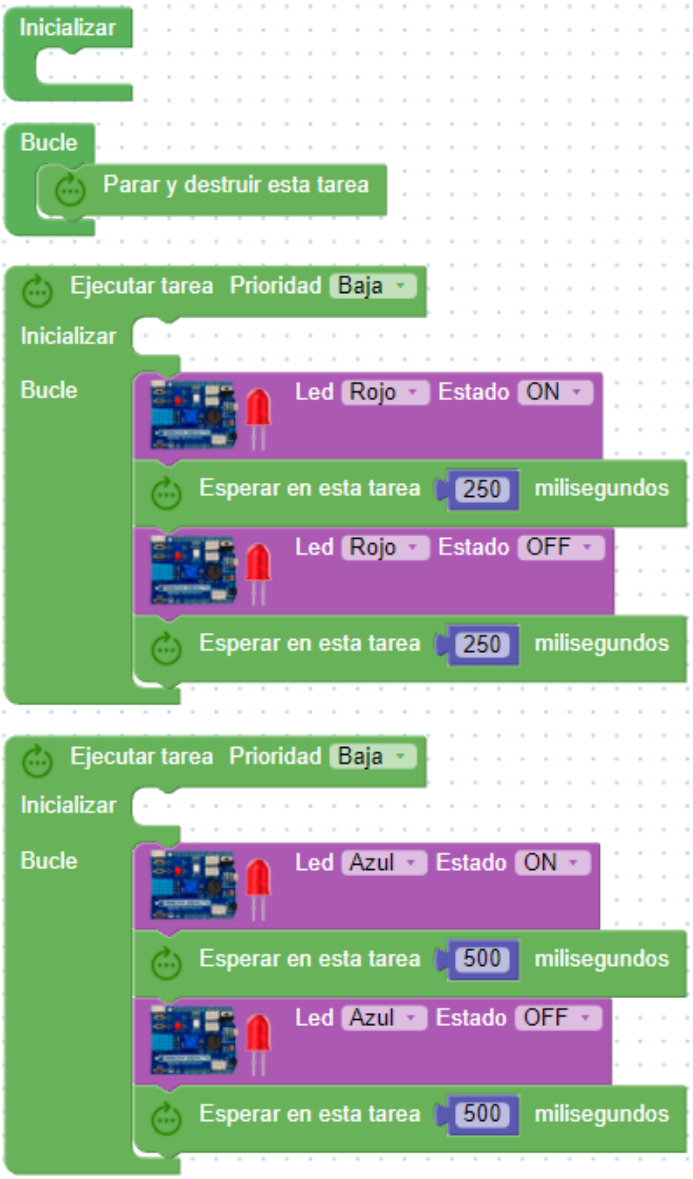
Actividad de ampliación: realiza el programa anterior y comprueba su funcionamiento.

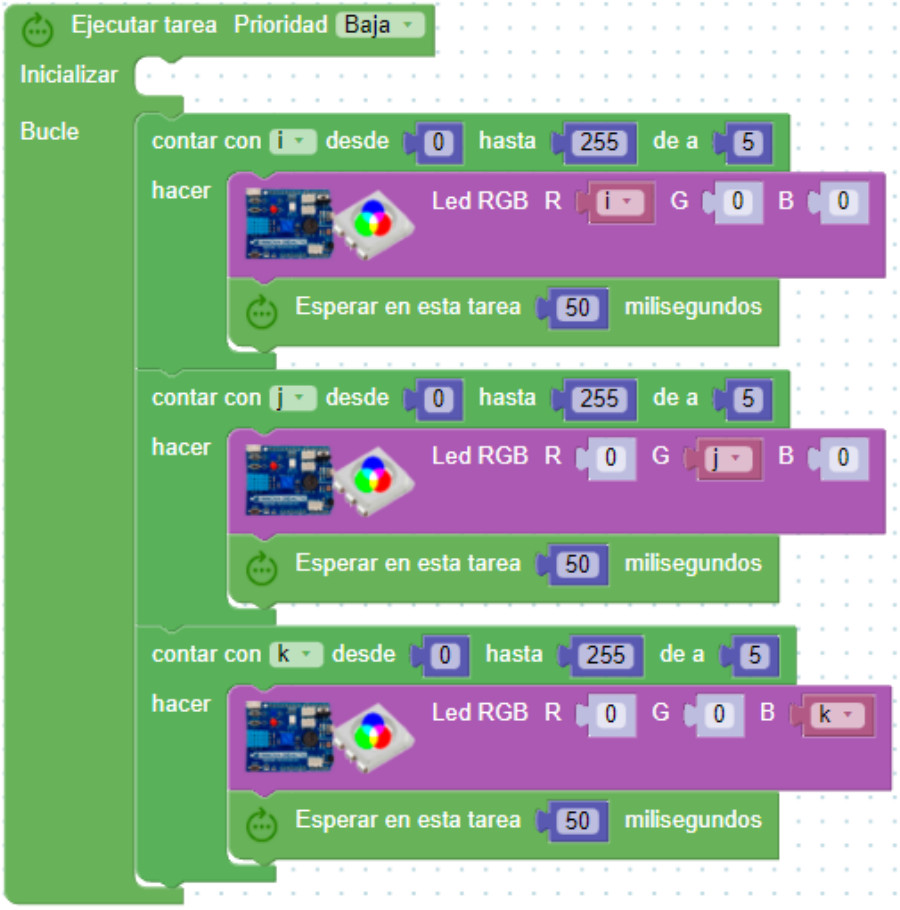
7.17.4 Reto A17.4. Multitarea IV.

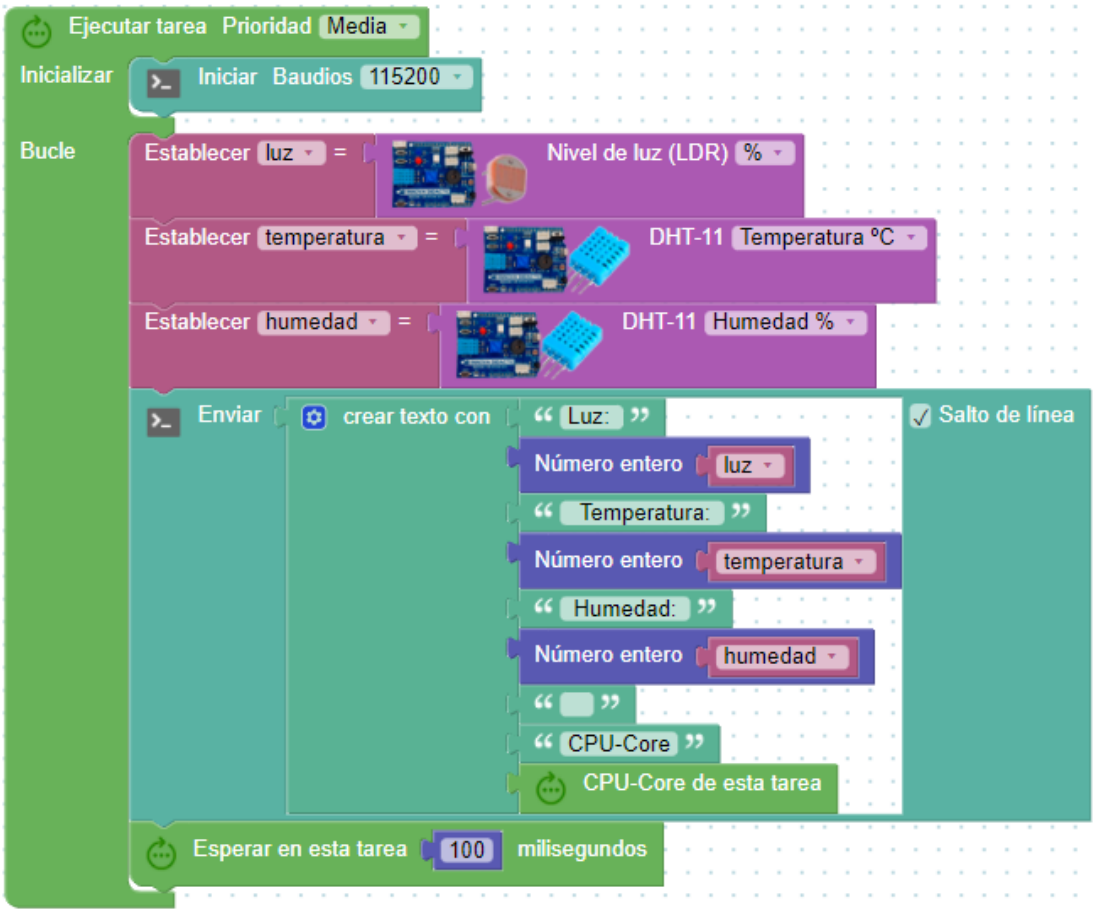
Reto A17.4. Multitarea IV.

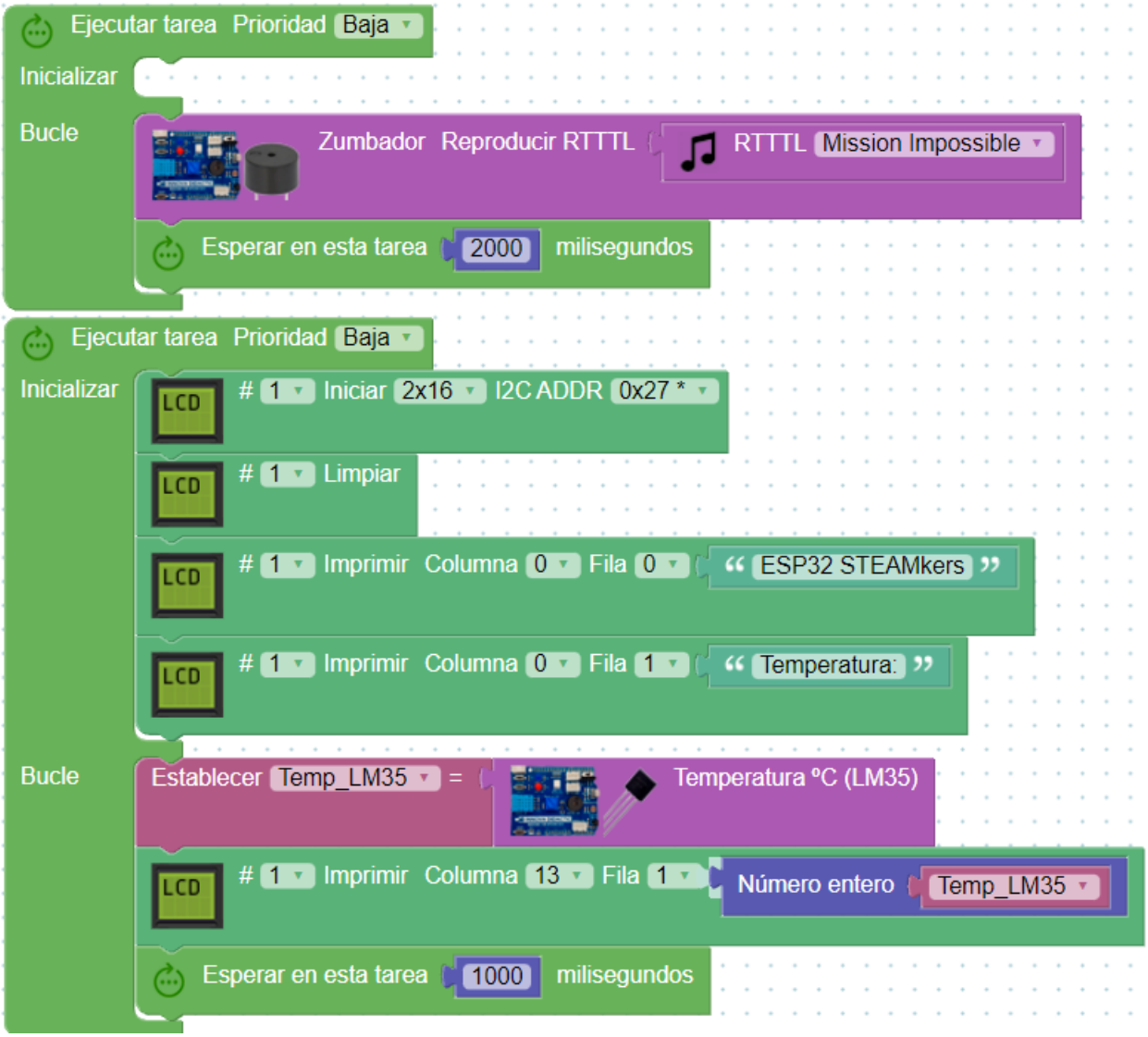
Y, la prueba definitiva del sistema multitarea es tener los siguientes elementos trabajando a la vez:

- Led rojo con intermitencia de 250ms.
- Led azul con intermitencia de 500ms.
- Led RGB cambiando de colores cada 50ms.
- Sensores de humedad y temperatura (DHT11) y LDR enviando datos por el puerto serie a la *Consola*. Mostrará también la *CPU-Core* que está trabajando en esa tarea. Tiene que estar en prioridad Media.
- Altavoz reproduciendo una melodía RTTTL.
- Pantalla LCD mostrando la temperatura del sensor LM35 cada 1000ms.




Ejecutar tarea Prioridad Baja
Inicializar
Bucle
contar con i desde 0 hasta 255 de a 5
hacer
Led RGB R i G 0 B 0
Esperar en esta tarea 50 milisegundos
contar con j desde 0 hasta 255 de a 5
hacer
Led RGB R 0 G j B 0
Esperar en esta tarea 50 milisegundos
contar con k desde 0 hasta 255 de a 5
hacer
Led RGB R 0 G 0 B k
Esperar en esta tarea 50 milisegundos


Ejecutar tarea Prioridad Media
Inicializar
Iniciar Baudios 115200
Bucle
Establecer luz = Nivel de luz (LDR) %
Establecer temperatura = DHT-11 Temperatura °C
Establecer humedad = DHT-11 Humedad %
Enviar crear texto con
Luz:
Número entero luz
Temperatura:
Número entero temperatura
Humedad:
Número entero humedad
CPU-Core
CPU-Core de esta tarea
Salto de línea
Esperar en esta tarea 100 milisegundos



Enlace al programa: ArduinoBlocks Projects\multitarea_4.abp

Actividad de ampliación: realiza el programa anterior y comprueba su funcionamiento.

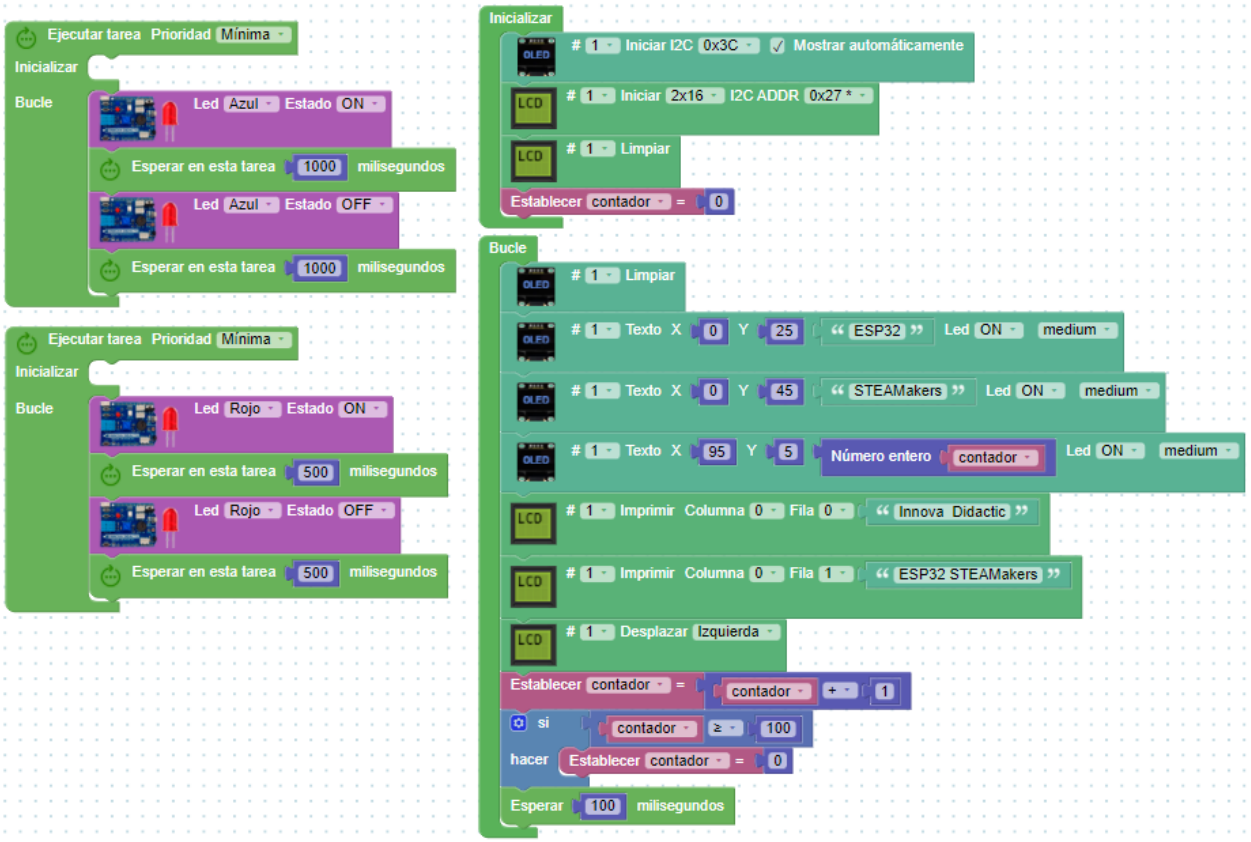
7.17.5 Reto A17.5. Multitarea V.

Reto A17.5. Multitarea V.

Vamos a trabajar con varios elementos I2C a la vez como una pantalla LCD y una pantalla OLED. También tendremos dos leds (rojo y azul) parpadeando a diferentes frecuencias. Para que todo funcione correctamente deberemos tener toda la programación correspondiente al I2C en el *bucle* principal.

El led azul parpadeará cada segundo y el led rojo cada medio segundo ejecutándose de forma paralela las dos. En el programa principal se mostrará un mensaje en la pantalla LCD que se irá desplazando a la izquierda y un contador con un mensaje en la pantalla OLED.

Enlace al programa: ArduinoBlocks Projects\multitarea_5.abp



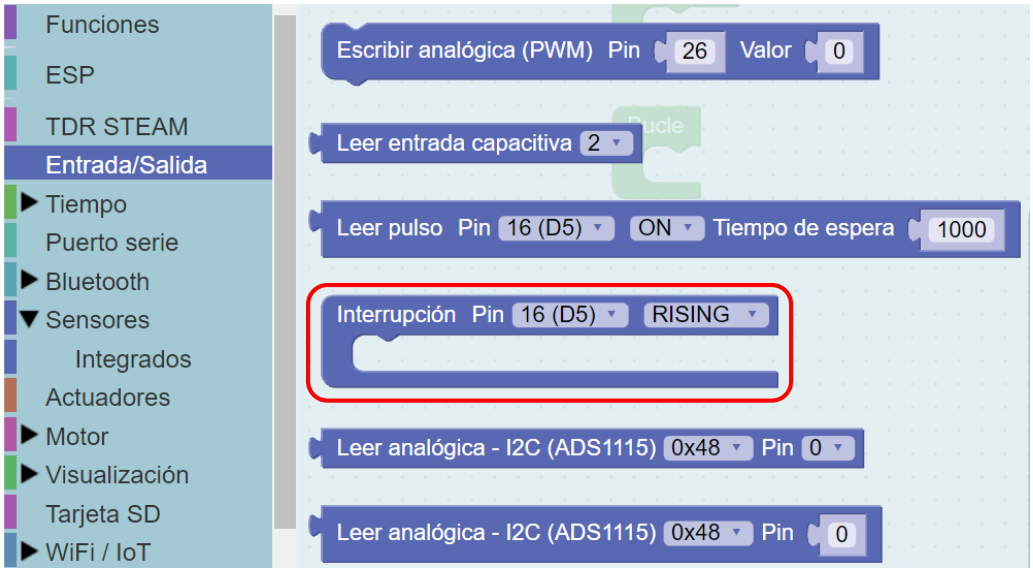
Actividad de ampliación: realiza el programa anterior y comprueba su funcionamiento.

7.18 Reto A18. Interrupciones.

Reto A18. Interrupciones.

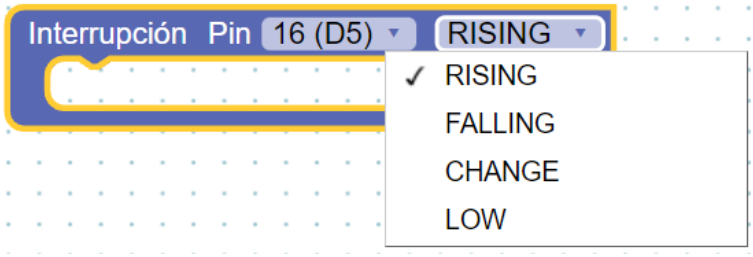
En la placa Imagina TDR STEAM disponemos de dos entradas de interrupciones en D3 y D5. Una interrupción es un evento que hace que el microcontrolador deje de realizar la tarea actual y pase a efectuar otra actividad prioritaria. Esta priorización se realiza mediante hardware, normalmente mediante un pulsador.

ArduinoBocks dispone de un bloque de interrupción dentro del apartado de Entrada/Salida.



Permite realizar la interrupción por 4 eventos diferentes:

- Rising: apretamos el pulsador.
- Falling: soltamos el pulsador.
- Change: cambio de estado del pulsador.
- Low: estado bajo (0) del pulsador.

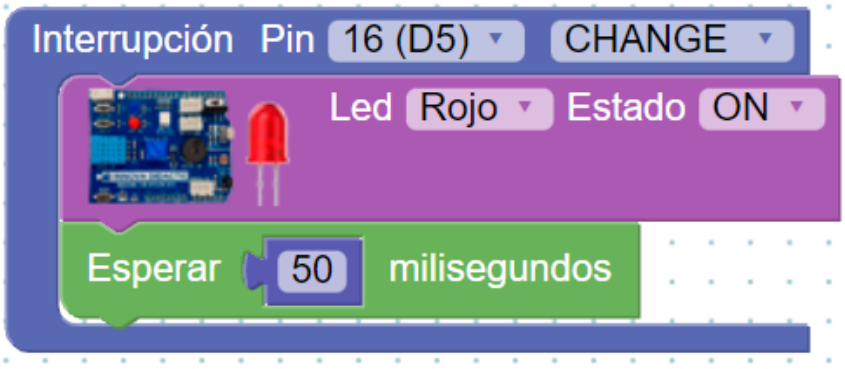


7.18.1 Reto A18.1. Control de interrupciones.

Reto A18.1. Control de interrupciones.

Vamos a realizar un sencillo programa para comprobar el funcionamiento de esta función de interrupción. El led azul estará realizando una intermitencia cada 200 milisegundos y el led rojo permanecerá apagado. Cuando apretemos el pulsador conectado en D5 el led rojo se encenderá durante 50 milisegundos.

Ahora cambiaremos solamente el tipo de interrupción a *CHANGE* y observaremos que el led rojo se enciende al apretar y al soltar el pulsador.



Enlace al programa: ArduinoBlocks Projects\interrupciones.abp

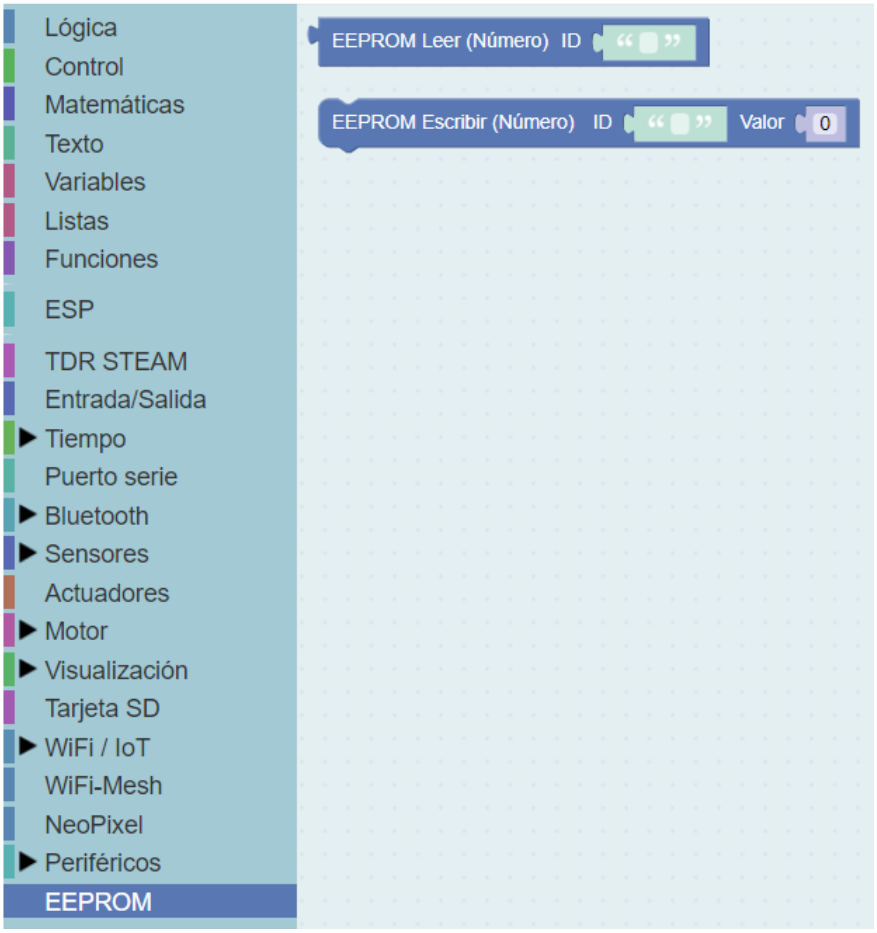
Actividad de ampliación: realiza el programa anterior y comprueba su funcionamiento.

7.19 Reto A19. Memoria FLASH/EEPROM.

Reto A19. Memoria FLASH/EEPROM.

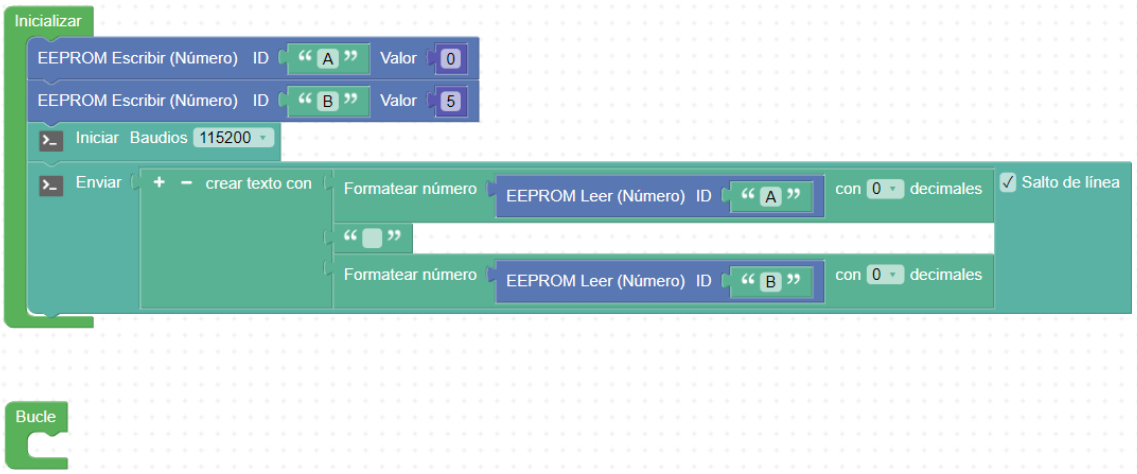
El microcontrolador de la placa ***ESP32 Plus STEAMakers*** está basado en un ESP32-WROOM-32 que tiene una zona de memoria Flash con la que se puede interactuar de forma similar a la EEPROM de una placa Arduino. Puede mantener los datos en la memoria incluso después de apagar la placa. Una cosa importante a tener en cuenta es que la EEPROM tiene un tamaño y una vida útil limitados. Las celdas de memoria pueden leerse tantas veces como sea necesario, pero el número de ciclos de escritura está limitado a 100.000. Es aconsejable prestar atención al tamaño de los datos almacenados y a la frecuencia con la que se desea actualizarlos. La memoria EEPROM puede almacenar 512 valores de 0 a 255.

En ArduinoBlocks disponemos de dos bloques para poder trabajar con la memoria EEPROM, uno de lectura y otro de escritura.



Realizaremos un programa para comprobar que se pueden almacenar datos en la memoria EEPROM. Almacenaremos en la posición A el valor 0 y en la posición B el valor 5. Una vez cargado el programa nos mostrará por el puerto serie (*Consola*) los valores almacenados.

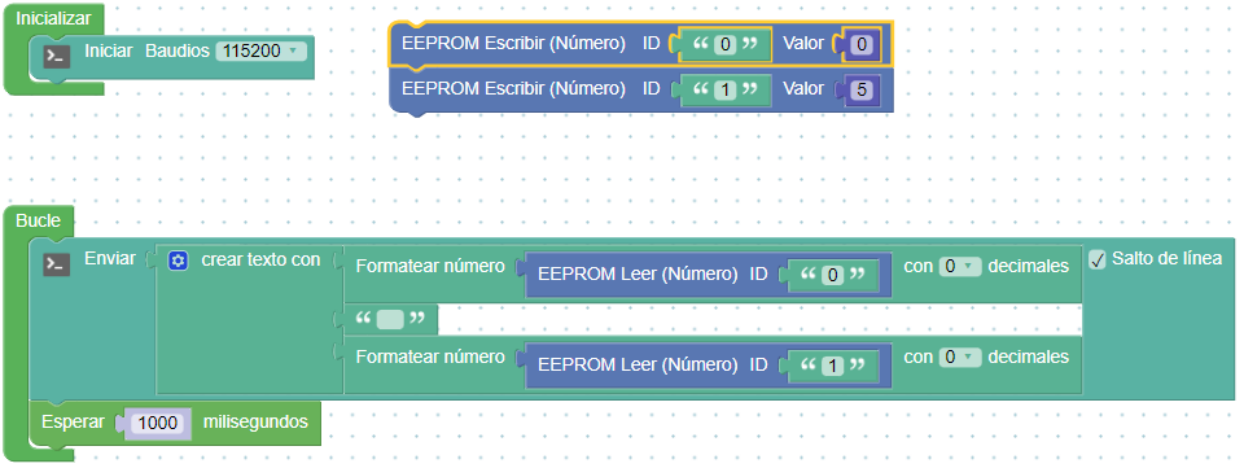
El programa resultante será:



Enlace al programa: ArduinoBlocks Projects\eeprom_0.abp

Los datos visualizados por la *Consola* serán:

Volvemos a enviar el programa, pero eliminado la parte de escribir en la EEPROM para comprobar que realmente se quedan almacenados.



Observamos que, efectivamente, los datos se quedan almacenados.

7.19.1 A19.1. Lectura/escritura en la memoria EEPROM I.

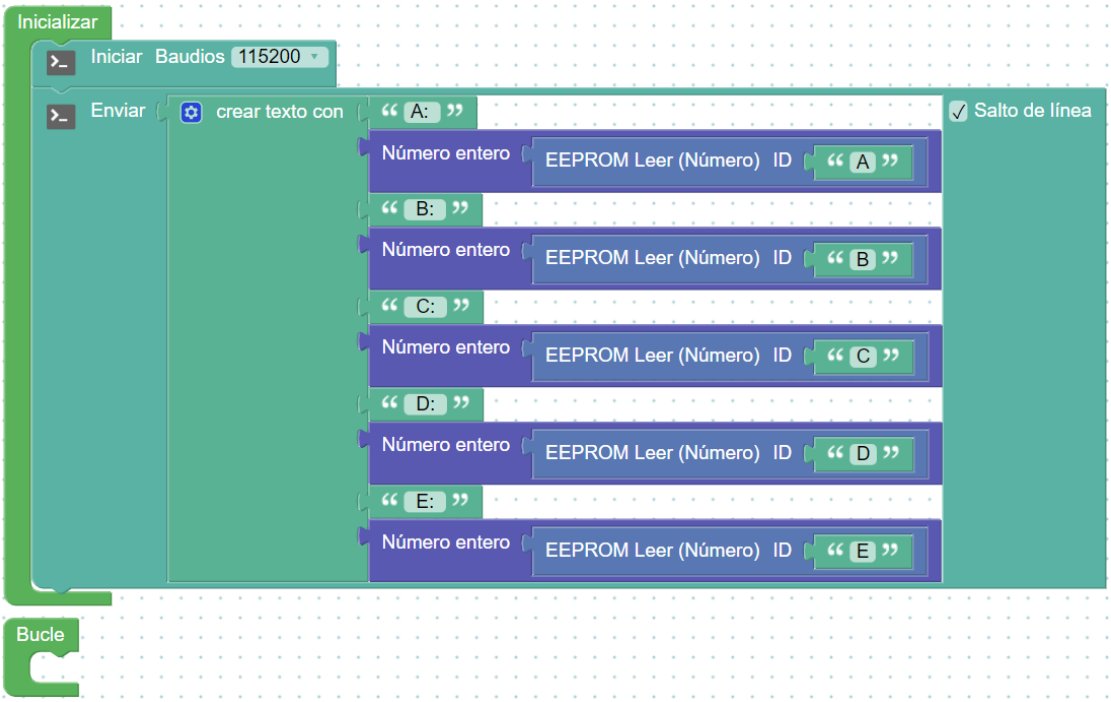
Reto A19.1. Lectura/escritura en la memoria EEPROM I.

Vamos a realizar dos programas, uno para almacenar datos en la EEPROM y otro para poder leerlos. En el primer programa crearemos 5 zonas de memoria (A-B-C-D-E) donde almacenaremos los números del 0 al 4 respectivamente.

El programa para almacenar los datos en la EEPROM será el siguiente:

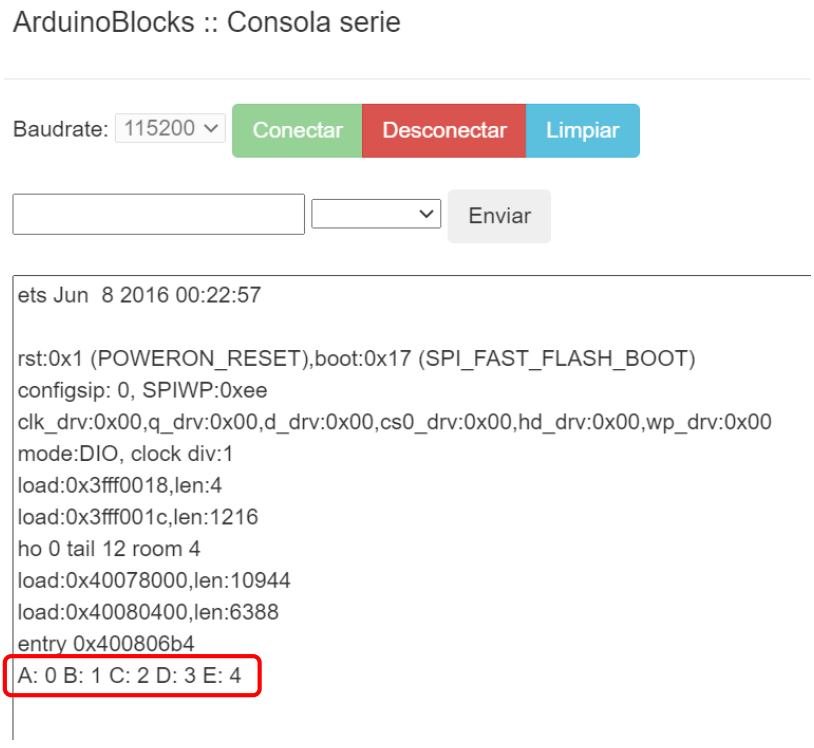
Enlace al programa: ArduinoBlocks Projects\eeprom_1.abp

El programa para leer los datos almacenados en la EEPROM será el siguiente (podemos desconectar la placa antes para comprobar que, aunque se quede sin alimentación, los datos se quedan almacenados).



Enlace al programa: ArduinoBlocks Projects\eeprom2.abp

Podemos comprobar por la *Consola* que los datos se han almacenado correctamente:



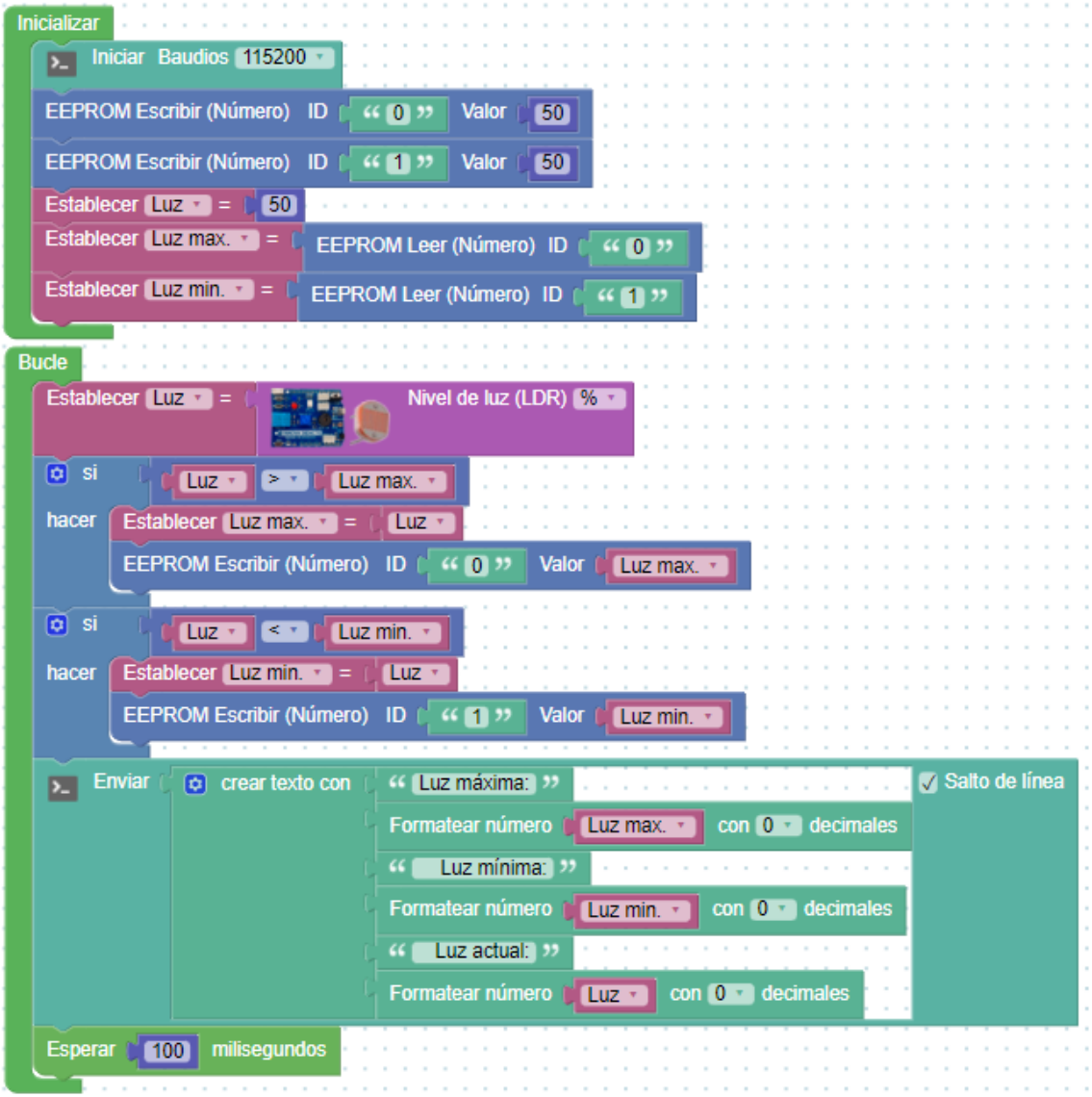
Actividad de ampliación: realiza el programa anterior y comprueba su funcionamiento.

7.19.2 A19.2. Lectura/escritura en la memoria EEPROM II.

Reto A19.2. Lectura/escritura en la memoria EEPROM II.

Vamos a realizar un programa para almacenar datos en la EEPROM. Los datos que almacenaremos será el valor máximo de luz en la posición 0 y el valor mínimo de luz en la posición 1. Podremos consular los valores por la *Consola*.

Primero estableceremos los dos valores a 50 con un sencillo programa. El programa mantendrá con los datos almacenados en la EEPROM, aunque quitemos la alimentación.



Actividad de ampliación: realiza el programa anterior y comprueba su funcionamiento.

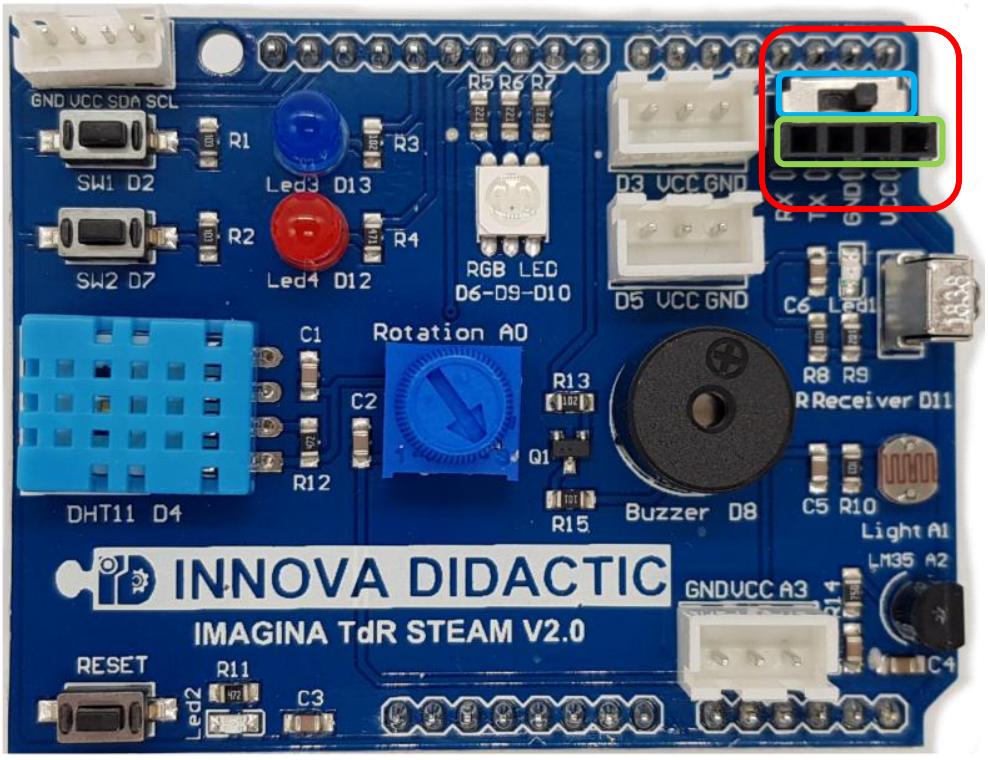
7.20 Reto A20. Sistemas de comunicaciones: Bluetooth y Wifi.

La placa ***ESP32 Plus STEAMakers*** dispone de conectividad Wifi y Bluetooth integrada en la propia placa, sin necesidad de ningún elemento externo. En la placa ***Imagina TDR STEAM*** disponemos de un puerto de comunicaciones serie externo que nos permite conectar otros módulos Bluetooth o Wifi. Además, dispone de un interruptor para poder conectarlos o desconectarlos ya que utiliza los mismos pines Rx/Tx que para comunicarse con el ordenador).

En nuestro caso utilizaremos el sistema de comunicación Bluetooth y Wifi de la propia placa, por lo que dejaremos en posición OFF el interruptor (si está en posición ON y hay un dispositivo conectado, no podremos enviar el programa a la placa).

Estado del interruptor:

- OFF: comunicación con el ordenador.
- ON: comunicación con los diferentes módulos Bluetooth o Wifi conectador en el conector hembra.



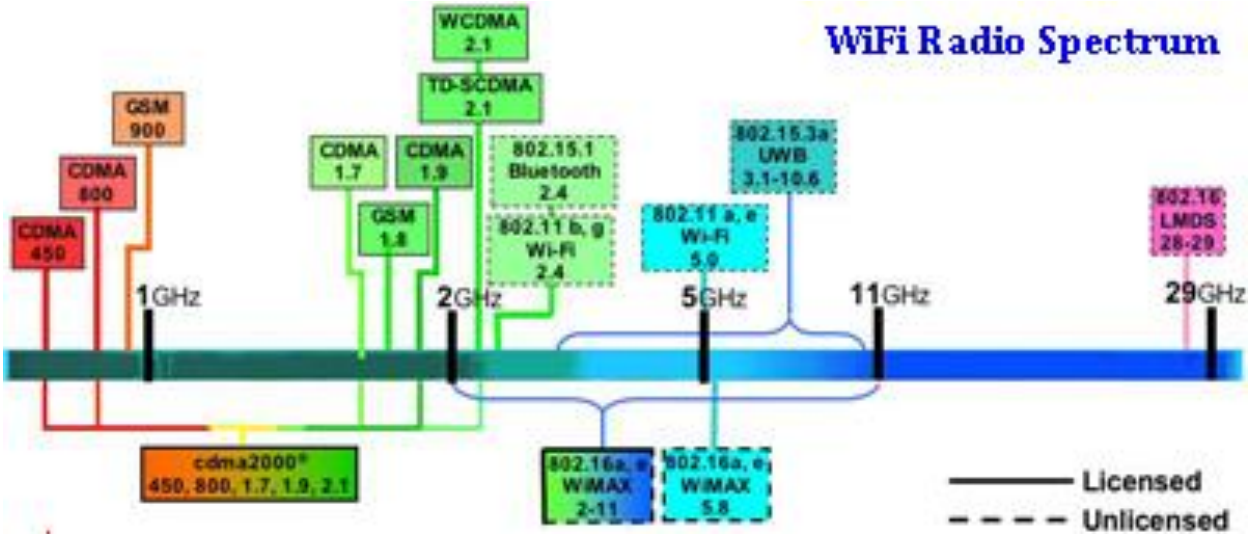
Vamos a utilizar la conectividad Wifi y Bluetooth que viene ya integrada en la placa, sin la necesidad de añadir elementos externos. Esto simplificará el montaje de elementos externo y abarata costes a la hora de montar proyecto.

Estas dos tecnologías las podemos resumir de la siguiente forma:

- La comunicación Bluetooth consiste en: es una tecnología (protocolo) de comunicación que permite la comunicación inalámbrica de corto alcance entre dispositivos digitales. El rango aproximado de alcance es de unos 10-20 metros.

- La comunicación Wifi consiste en: es una tecnología (protocolo) de comunicación que permite la comunicación inalámbrica utiliza las ondas de radiofrecuencia. Su alcance es de unos 100-150 metros. Nos permitirá realizar proyectos de IoT.

La comunicación Wifi se utiliza más para la conexión de dispositivos a Internet (IoT), mientras que la comunicación Bluetooth se utiliza para conectar dispositivos entre sí. La comunicación Wifi también permite el intercambio de datos entre dispositivos que están conectado en una misma red.



A continuación, haremos una breve introducción a IoT (Internet of Things). IoT en la interconexión de dispositivos y objetos a través de una red (privada o Internet) donde pueden interaccionar entre ellos. Cualquier elemento que se pueda imaginar podría ser conectado a Internet e interactuar, con o sin la necesidad de la intervención humana. El objetivo, por tanto, es una

interacción de máquina a máquina, más conocido como interacción M2M (Machine to Machine) o dispositivos M2M.

Las aplicaciones de esta tecnología permiten mejorar la vida cotidiana de las personas, así como los entornos industriales donde esté implantado. Algunos ejemplos de aplicaciones son:

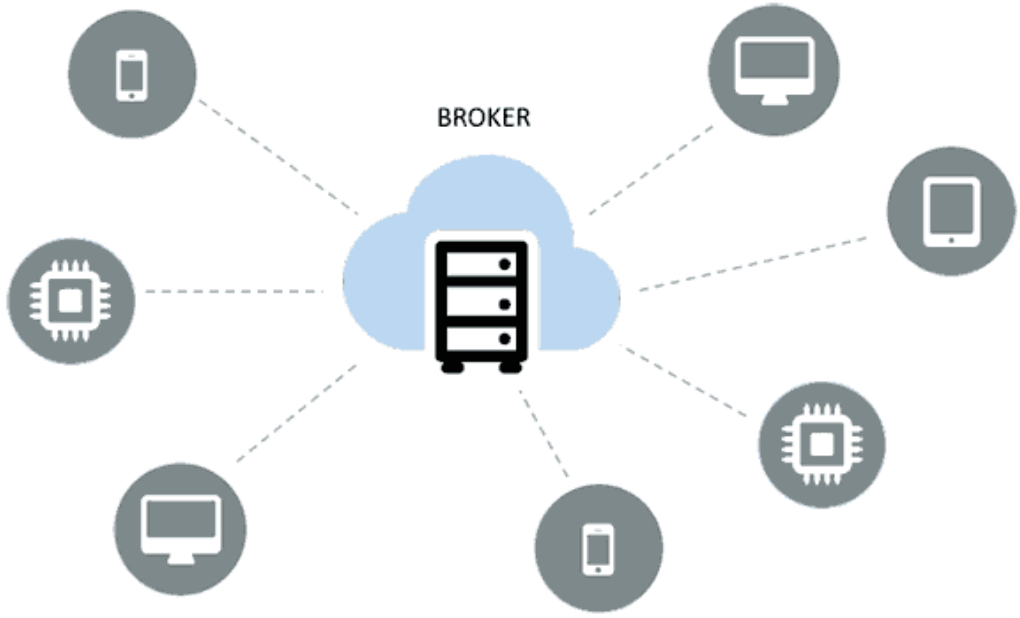
- Sistemas domóticos para hogares.
- Conexión de electrodomésticos para su control remoto.
- Conexión de objetos cotidianos.
- Sistemas de alarma en aplicaciones industriales.
- Control de elementos industriales.
- Aplicación en Smart Cities i Smart Buildings.
- Etc.

Para los sistemas IoT se utilizan diferentes tecnologías y protocolos de comunicación. Últimamente han aparecido en el mercado gran cantidad de dispositivos con conectividad que son de pequeño tamaño, baratos y con un consumo energético bajo, como el ESP8266 y el ESP32. Para la comunicación entre los diferentes dispositivos IoT se utilizan redes Wifi. Existen diferentes protocolos (normas definidas para que los dispositivos puedan comunicarse) para realizar esta comunicación Wifi entre dispositivos. Algunos de los protocolos M2M que existen son:

- **AMQP** (*Advanced Message Queuing Protocol*): es un protocolo PubSub de Message Queue. Está diseñado para asegurar la confiabilidad e interoperabilidad. Está pensado para aplicaciones corporativas, con mayor rendimiento y redes de baja latencia. No resulta tan adecuado para aplicaciones de IoT con dispositivos de bajos recursos.
- **CoAP** (*Constrained Application Protocol*): es un protocolo pensado para emplearse en dispositivos de IoT de baja capacidad. Emplea el modelo REST de HTTP con cabeceras reducidas, añadiendo soporte UDP, multicast, y mecanismos de seguridad adicionales.
- **MQTT** (*MQ Telemetry Transport*): es un protocolo PubSub de Message Service que actúa sobre TCP. Destaca por ser ligero, sencillo de implementar. Resulta apropiado para dispositivos de baja potencia como los que frecuentemente tenemos en IoT. Está optimizado para el routing activo de un gran número de clientes conectados de forma simultánea.
- **STOMP** (*Streaming Text Oriented Messaging Protocol*): es un protocolo sencillo que emplea HTTP y mensajes de texto para buscar el máximo de interoperabilidad.

- **WAMP** (*Web Application Messaging Protocol*): es un protocolo abierto que se ejecuta sobre WebSockets, y provee tanto aplicaciones de PubSub como rRPC.
- **WMQ** (*Websphere MQ*): es un protocolo de Message Queue desarrolado por IMB.
- **XMPP** (*eXtensible Messaging and Presence Protocol*) es un protocolo abierto basado en XML diseñado para aplicaciones de mensajería instantánea.

Para comunicar un número de dispositivos que están distribuido en ubicaciones y redes desconocidas y que se comuniquen de forma fiable es externalizar la comunicación a un servicio de notificaciones centralizado. Podemos disponer de un servidor central que se encarga de recibir los mensajes de todos los dispositivos emisores y distribuirlos a los receptores. De forma genérica llamaremos a este servidor 'Router' o 'Broker'.



Este servidor, que tiene una dirección fija (o equivalente a un dominio), de forma que es accesible por todos los dispositivos. Así resolvemos el problema de tener que encontrar al otro dispositivo. El servidor mantiene un registro de los dispositivos conectados, recibe los mensajes y los distribuye al resto dispositivos, filtrando los destinatarios según algún criterio. Los dispositivos en ningún momento 'ven' o dependen del resto de dispositivos. Por tanto, esta infraestructura nos proporciona la escalabilidad.

7.21 Reto A21. Comunicación Bluetooth.

Reto A21. Comunicación Bluetooth.

Para poder hacer la comunicación por Bluetooth con nuestra placa ***Imagina TDR STEAM*** disponemos de una serie de bloques específicos para la comunicación Bluetooth.

Tenemos bloques para poder trabajar con comunicaciones Bluetooth, pero también disponemos de bloques para poder conectar un ratón o un teclado por Bluetooth (no se pueden utilizar los dos a la vez).

Para realizar la comunicación Bluetooth necesitaremos dos programas para poder trabajar:

- *ArduinoBlocks*: programa que funcionar en la placa ***ESP32 Plus STEAMakers***.
- *AppInventor2*: aplicación que funcionará en el teléfono móvil.

Realizaremos dos actividades compuestas cada una de ellas por dos programas (*AppInventor2* y *ArduinoBlocks*).

7.21.1 Reto A21.1. Programación básica con Bluetooth.

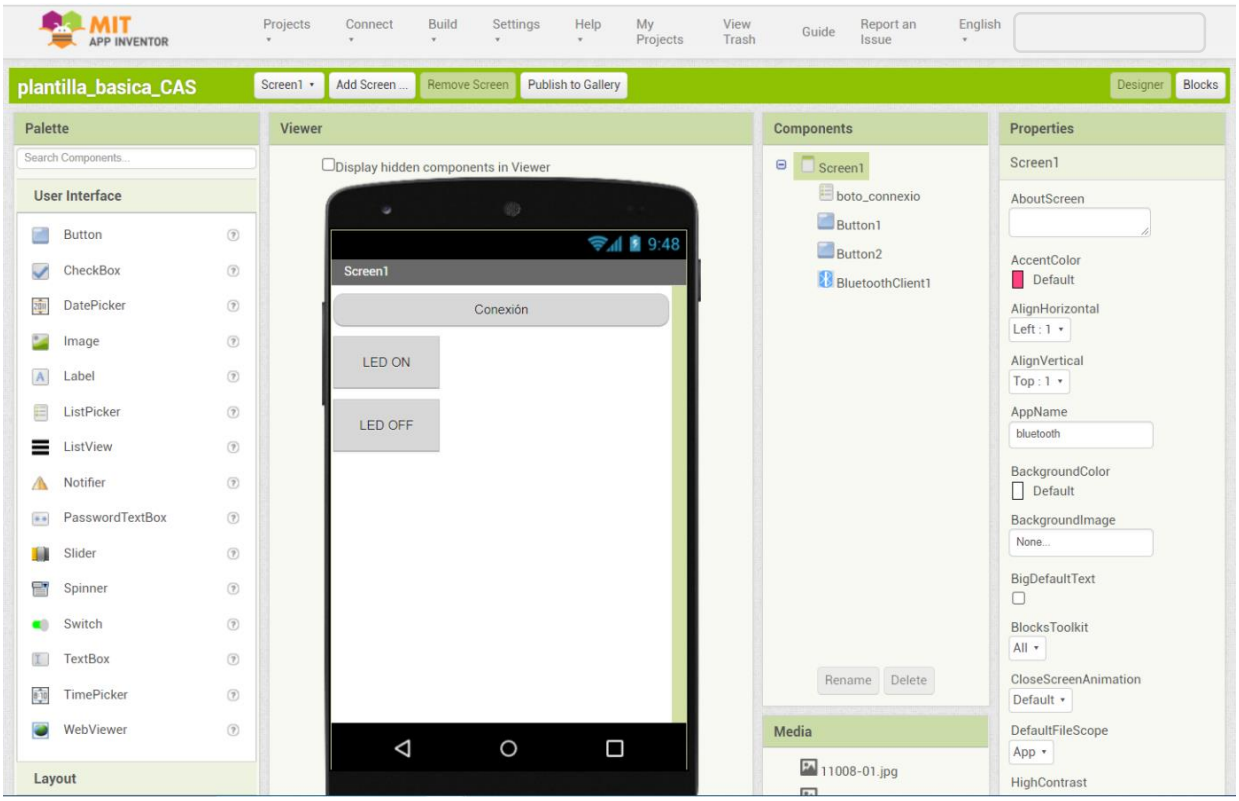
Reto A21.1. Programación básica con Bluetooth.

Primero procederemos a crear la aplicación con *AppInventor2*. Para ello, deberemos crear una cuenta para *AppInventor2* a través del siguiente enlace:

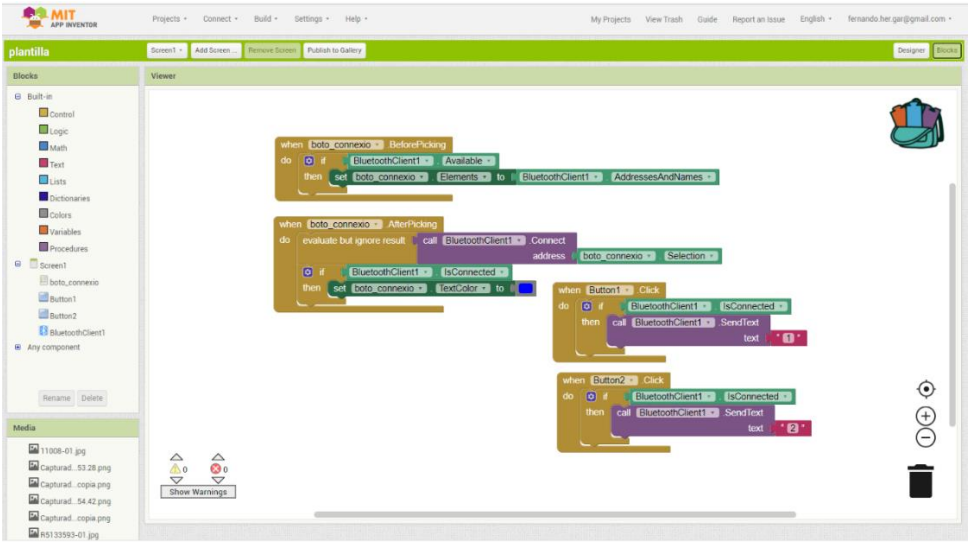
http://ai2.appinventor.mit.edu/

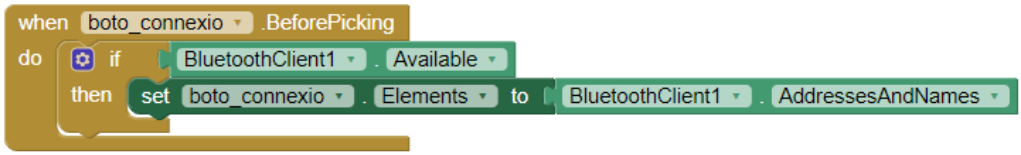
Una vez creada la cuenta procederemos a realizar la programación, tanto en *Designer* como en *Blocks*. A continuación, se detallan las dos partes del programa.

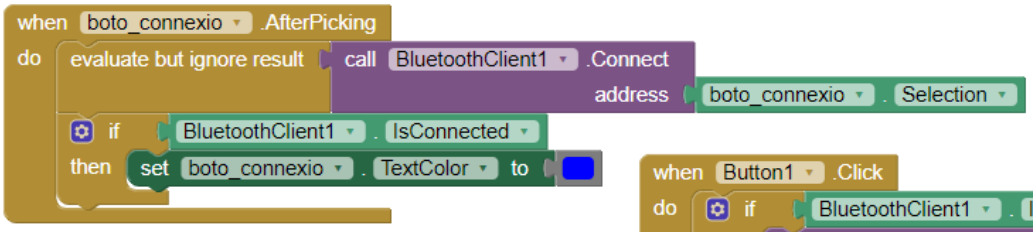
- Designer:

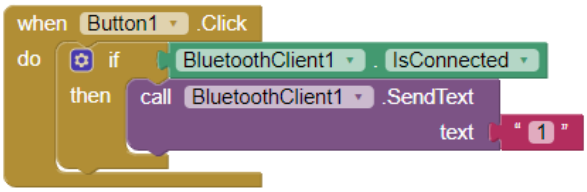


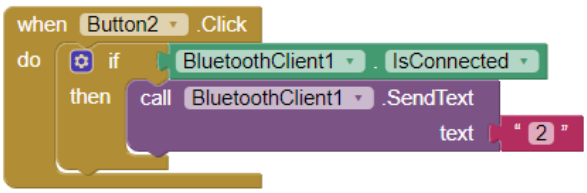
- Blocks:







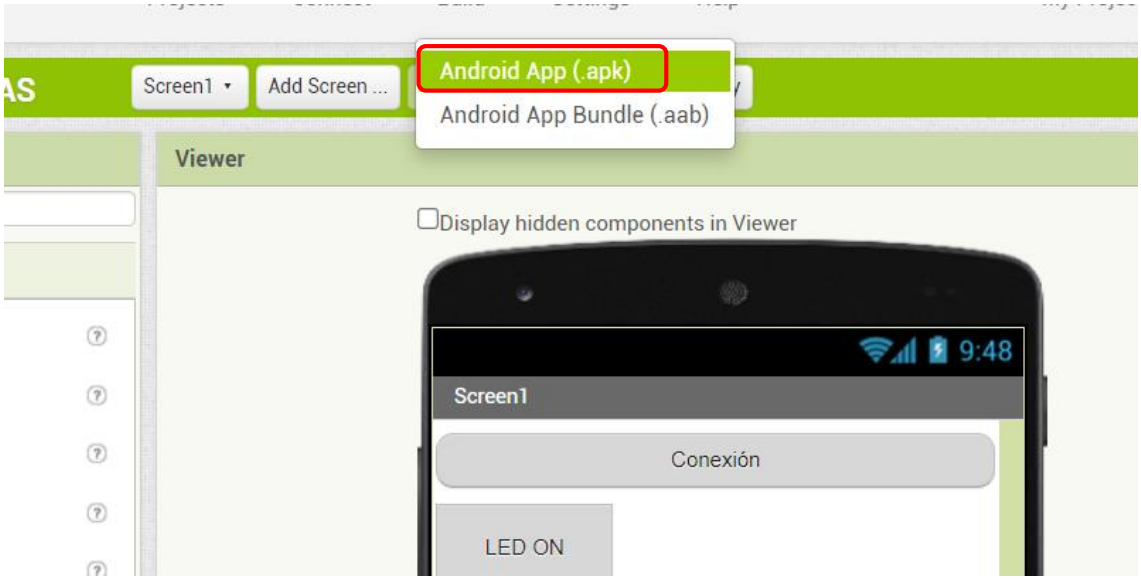




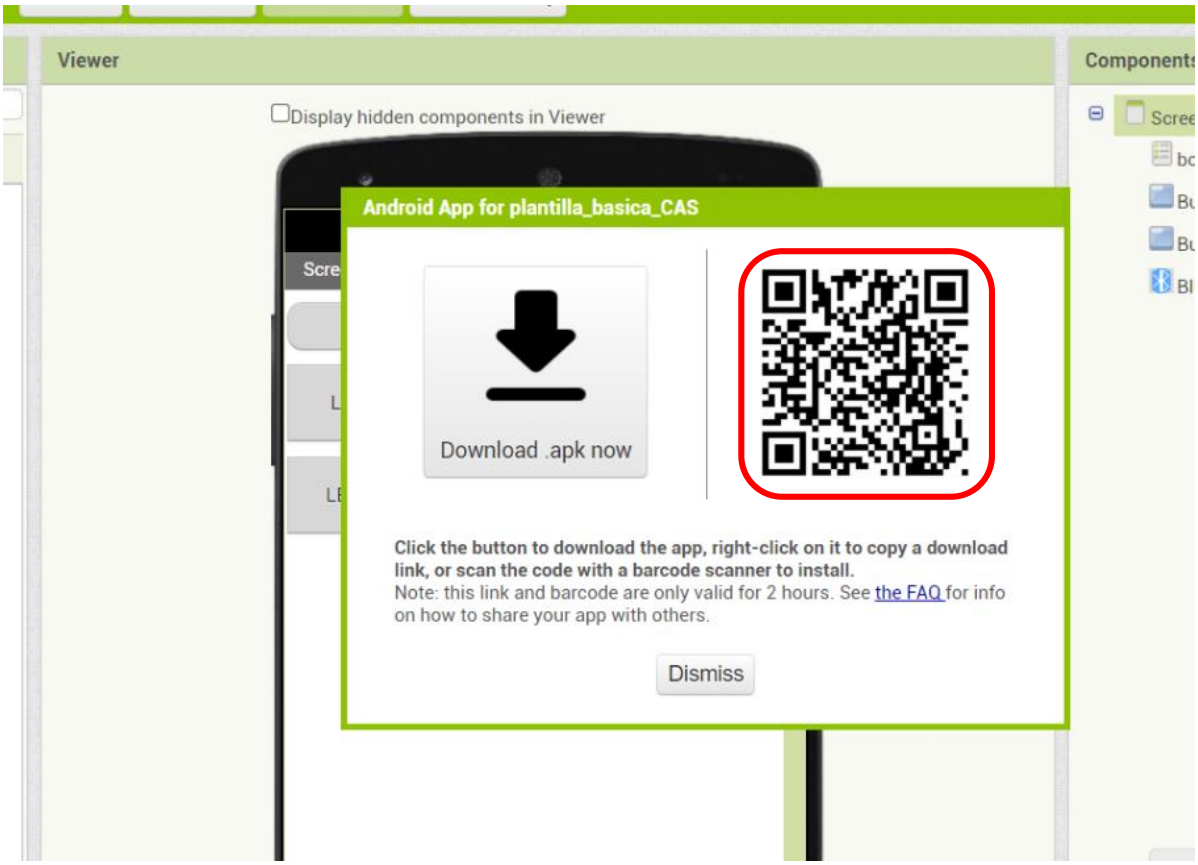
Enlace al programa: [AppInventor2\plantilla_basica_CAS.aia]

Una vez realizamos el programa, instalamos la aplicación creada en el teléfono móvil.

En el menú *Build*, creamos el fichero que deberemos instalar en el teléfono.

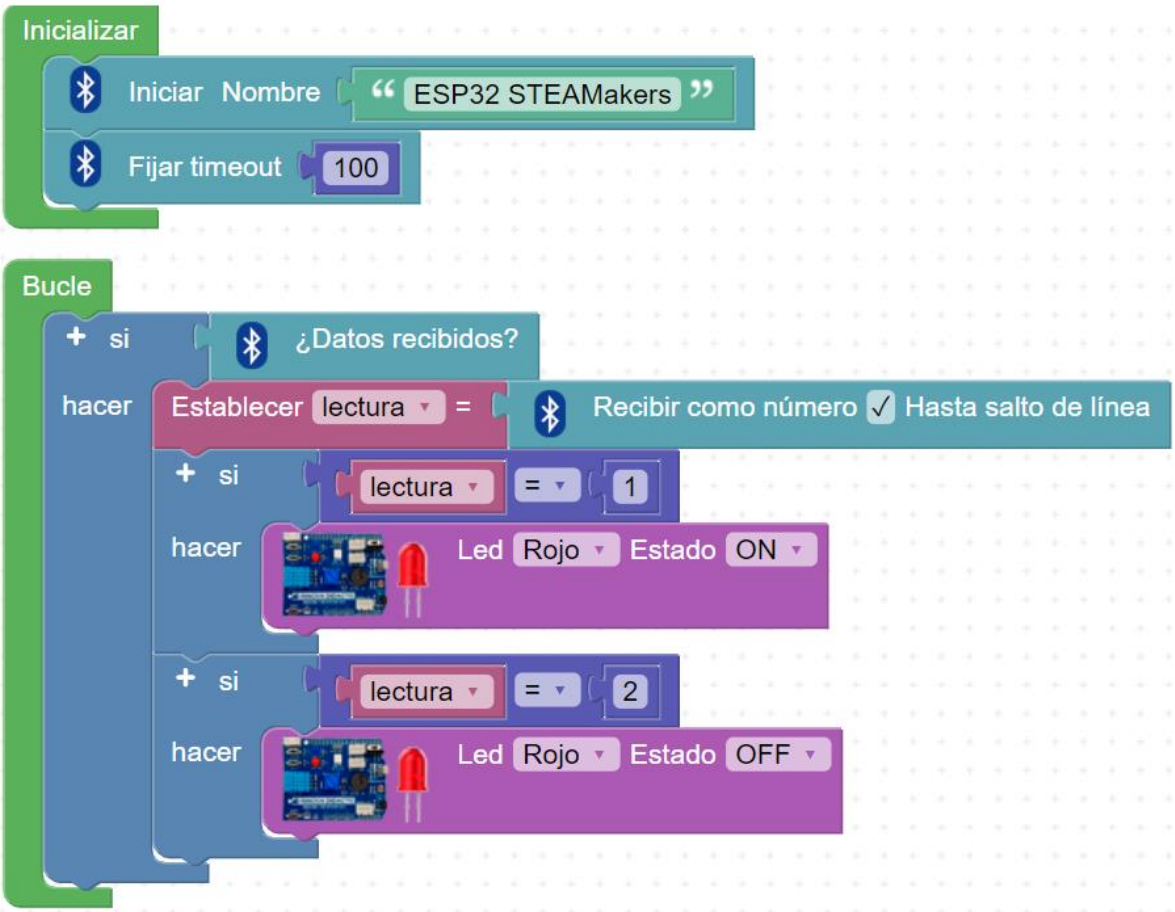


AppInventor2 crea un código QR para poder descargar e instalar la aplicación.



A continuación, procederemos a realizar el programa en ArduinoBlocks y que irá en la placa ***Imagina TDR STEAM***.

Primero inicializaremos el nombre de nuestro dispositivo para poder identificarlo. A continuación, si recibimos datos por el puerto Bluetooth identificaremos el valor recibido y, en función de ese valor, encenderemos o apagaremos el led rojo.



Mediante el bloque *Fijar timeout* podemos controlar el tiempo en la transmisión de datos por bluetooth. Por defecto está fijado en 1000 ms pero podemos reducirlo a 100 en este programa y funciona de forma más fluida.

Enlace al programa: ArduinoBlocks Projects\bluetooth_1.abp

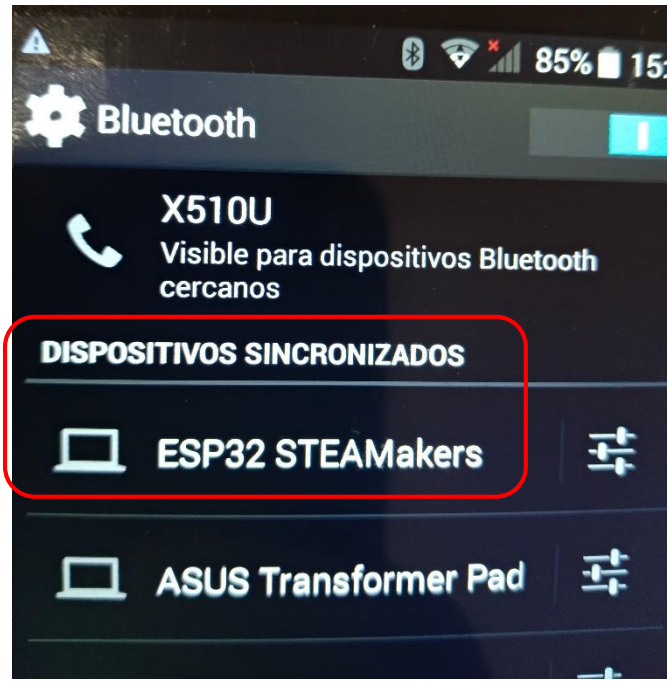
Una vez realizados los dos programas, seguiremos los siguientes pasos:

- Enviar el programa de ArduinoBlocks a la placa.
- Instalar la aplicación en el teléfono móvil.
- Sincronizar el Bluetooth del móvil con el módulo Bluetooth de la placa.
- Probar el programa.

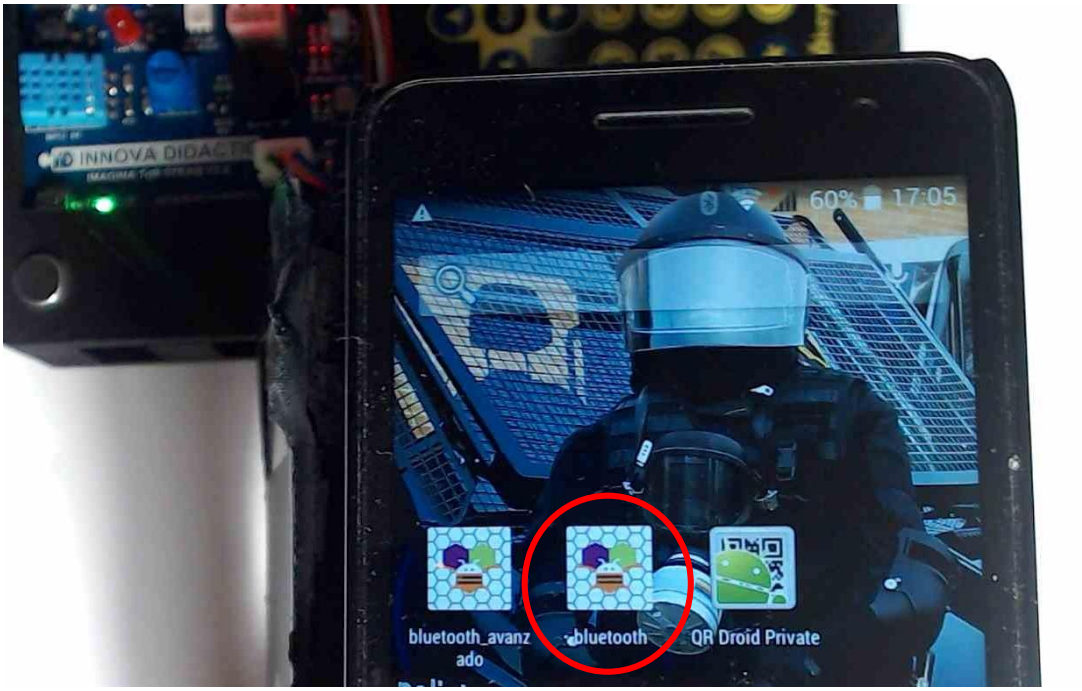
Vamos a vincular el sistema de comunicación Bluetooth del ***ESP32 Plus STEAMakers*** con el teléfono móvil. Activamos el módulo Bluetooth del teléfono móvil.

Abrimos el servicio Bluetooth del móvil y buscamos el nuevo dispositivo. En *Dispositivos Disponibles* aparecerá un dispositivo llamado **ESP32 STEAMakers**, que es el nombre que le hemos asignado.

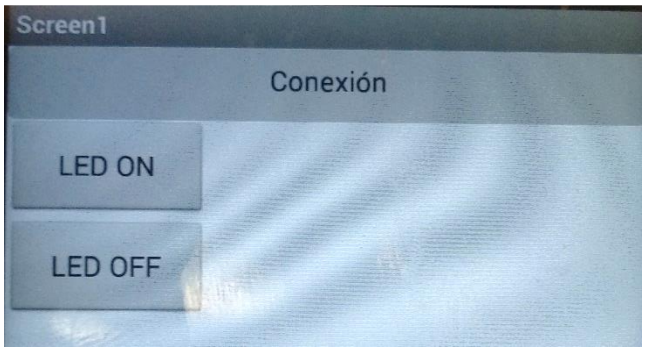
Aparecerá en *Dispositivos Sincronizados* el nombre que le hemos asignado al bluetooth de la placa *ESP32 STEAMakers*. Si nos pide contraseña deberemos introducir: 1234 (opcionalmente puede ser 0000). Ahora ya tenemos sincronizados los dos dispositivos Bluetooth.



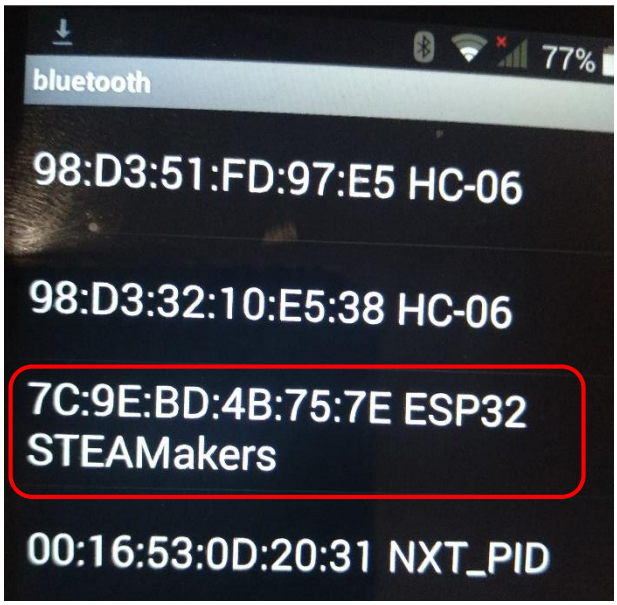
A continuación, abrimos la aplicación (en este caso se llama "Bluetooth).



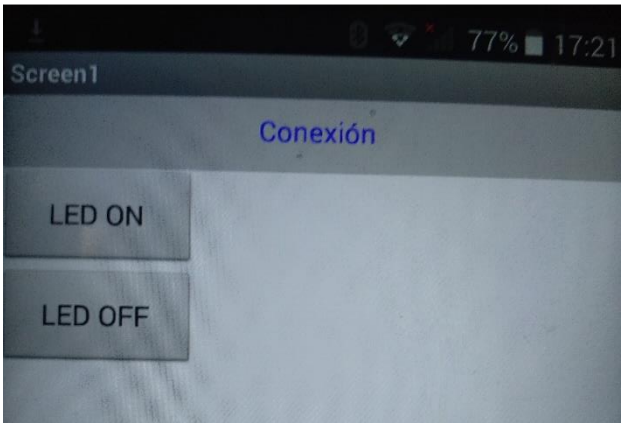
Apretamos el botón *Conexión* para poder hacer la conexión (está en letras de color negro).



Seleccionamos nuestro módulo Bluetooth (mediante la dirección MAC):

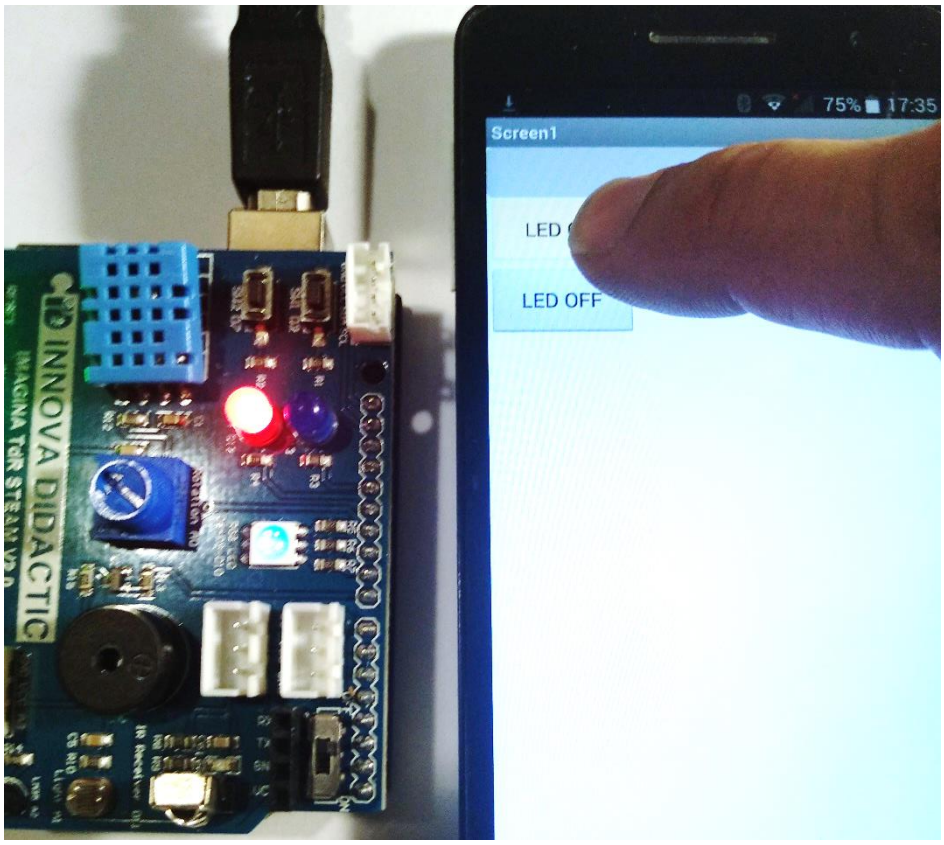


Ahora el botón *Conexión* aparecerá con las letras de color azul.



Tenemos dos opciones:

- Si apretamos el botón LED ON se encenderá el led azul.
- Si apretamos el botón LED OFF se apagará el led azul.



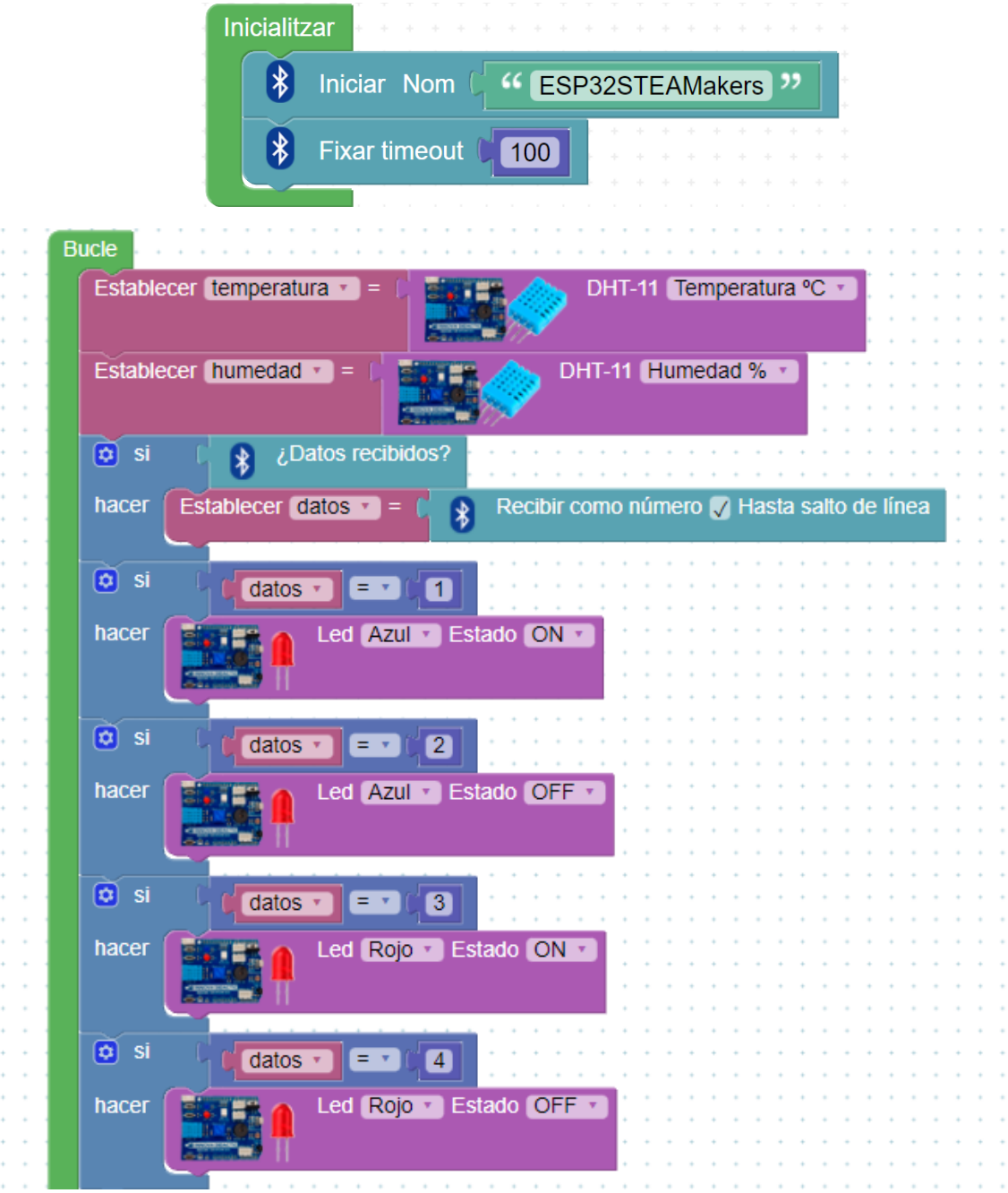
Actividad de ampliación: realiza los programas anteriores y comprueba su funcionamiento.

7.21.2 Reto A21.2. Programación avanzada con Bluetooth.

Reto A21.2. Programación avanzada con Bluetooth.

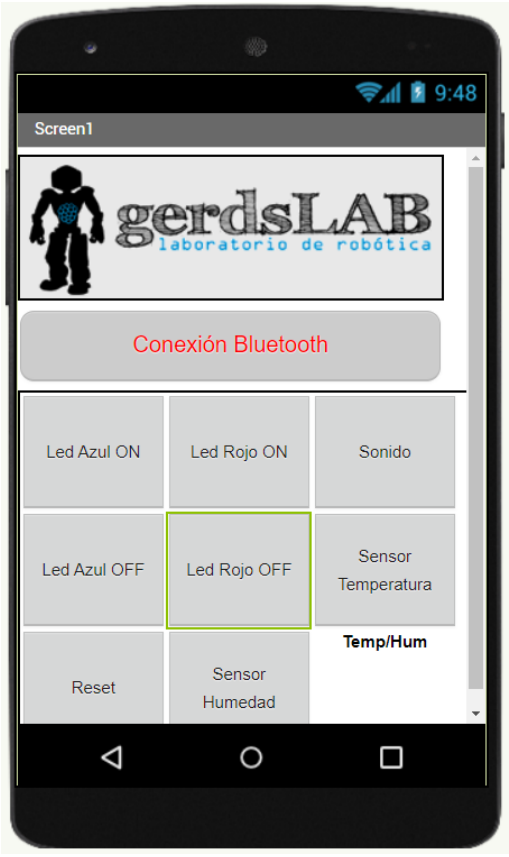
Vamos a realizar una programación en la que podamos controlar más elementos de la placa ***Imagina TDR STEAM***. Controlaremos los dos leds y podremos ver la temperatura y humedad. Primero crearemos el programa con ArduinoBlocks y, a continuación, la aplicación en *AppInventor2*.

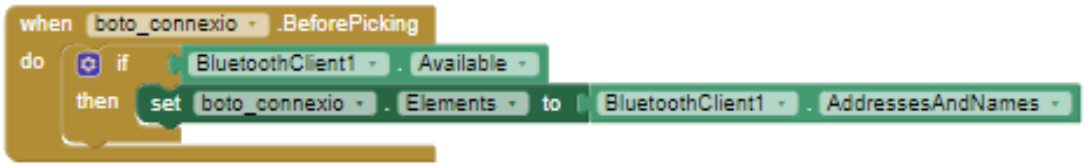
- ArduinoBlocks:

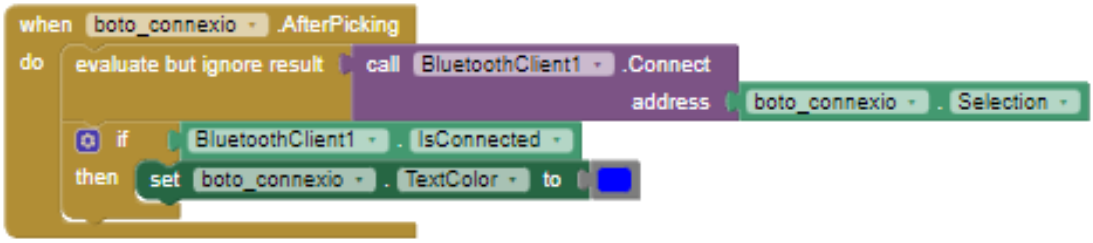


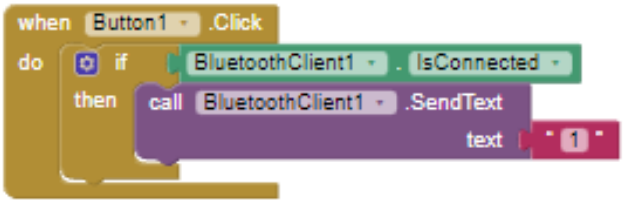
Enlace al programa: ArduinoBlocks Projects\bluetooth_2.abp

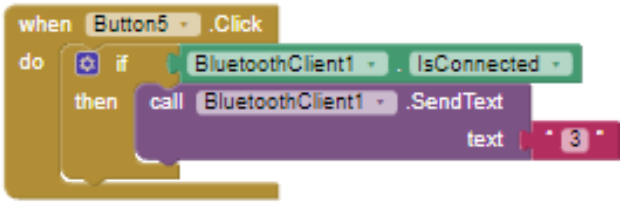
- AppInventor2:

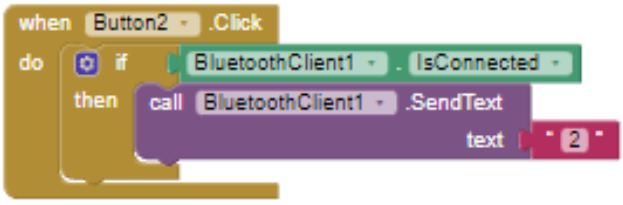


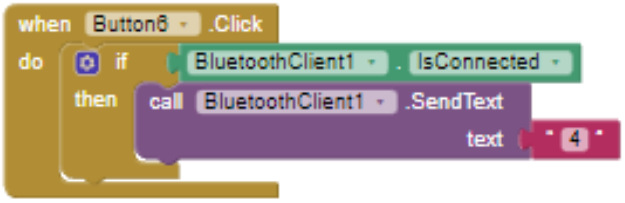

when boto_connexio .BeforePicking
do if BluetoothClient1 . Available
then set boto_connexio . Elements to BluetoothClient1 . AddressesAndNames

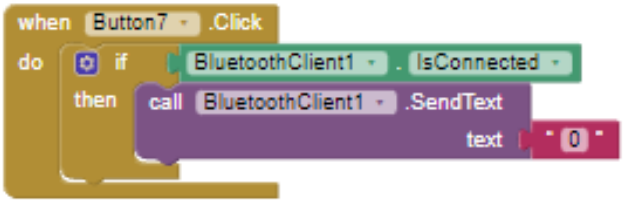

when boto_connexio .AfterPicking
do evaluate but ignore result call BluetoothClient1 .Connect
address boto_connexio . Selection
if BluetoothClient1 . IsConnected
then set boto_connexio . TextColor to

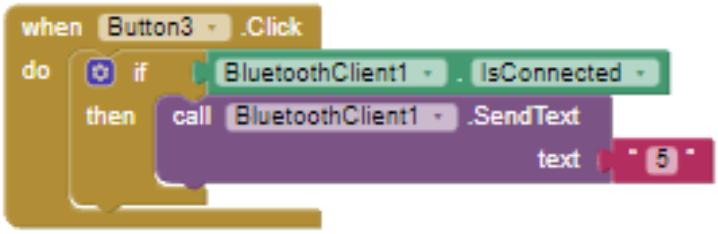

when Button1 .Click
do if BluetoothClient1 . IsConnected
then call BluetoothClient1 .SendText
text 1

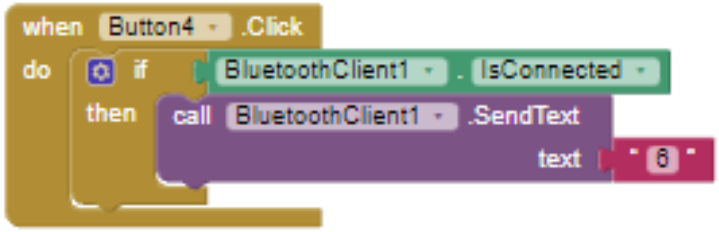

when Button5 .Click
do if BluetoothClient1 . IsConnected
then call BluetoothClient1 .SendText
text 3

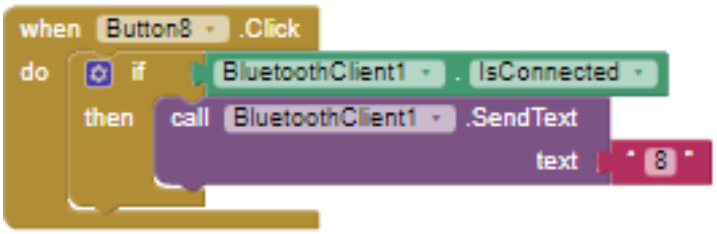

when Button2 .Click
do if BluetoothClient1 . IsConnected
then call BluetoothClient1 .SendText
text 2

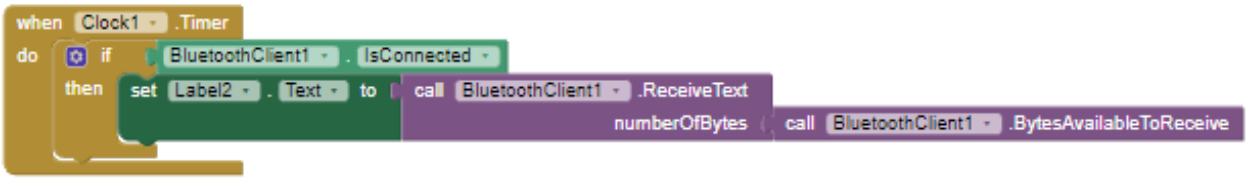

when Button6 .Click
do if BluetoothClient1 . IsConnected
then call BluetoothClient1 .SendText
text 4


when Button7 .Click
do if BluetoothClient1 . IsConnected
then call BluetoothClient1 .SendText
text 0









Enlace al programa: AppInventor2\plantilla_avanzado_CAS.aia

Actividad de ampliación: realiza los programas anteriores y comprueba su funcionamiento.

7.22 Reto A22. Comunicación Wifi.

Reto A22. Comunicación Wifi.

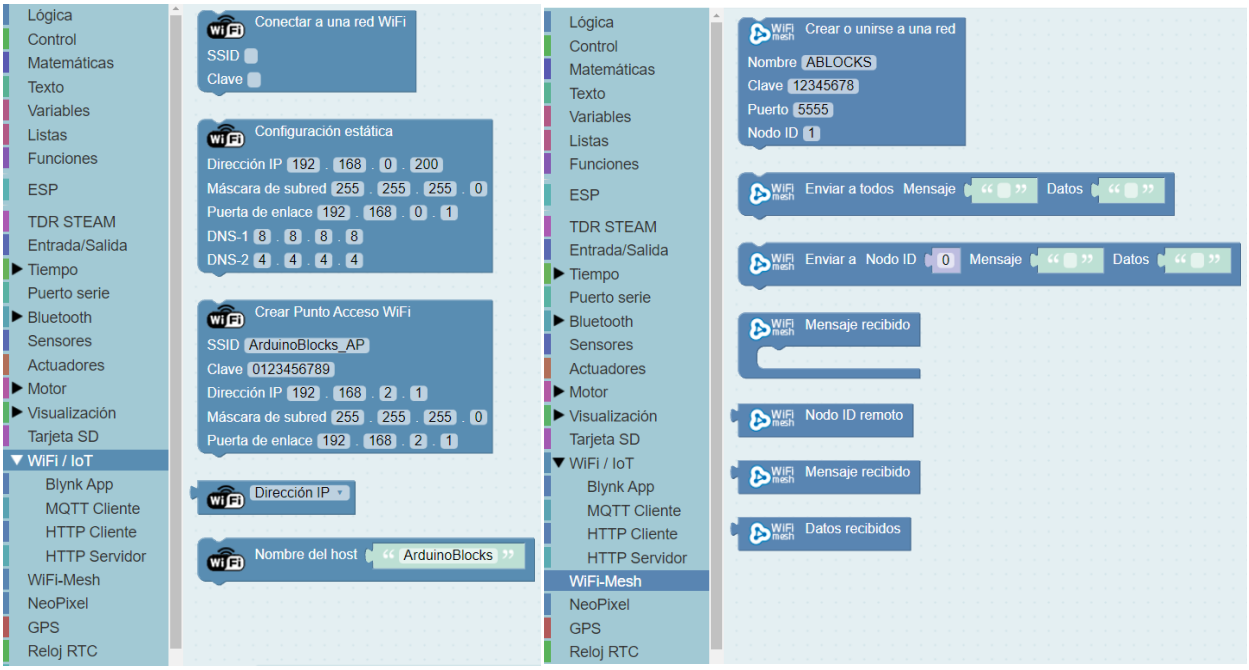
ArduinoBlocks dispone de un gran número de bloques dedicados a la comunicación Wifi que nos permiten desarrollar un gran número de aplicaciones.

El propio microcontrolador ESP32-WROOM-32 implementa el hardware y software compatible con redes Wifi 802.11 b/g/n 2.4GHz (soporta WFA/WPA/WPA2/WAPI).

Nota importante: cuando utilizamos la comunicación Wifi la placa ***ESP32Plus STEAMakers*** deshabilita internamente el ADC2, por lo que no funcionan las entradas A0 (potenciómetro) y A1 (LDR) como entradas analógicas. Si queremos conectar en la ***Imagina TDR STEAM*** un sensor analógico más lo deberemos conectar en A3.

Hay dos categorías de bloques:

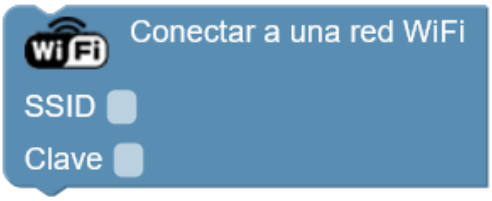
- WiFi/IoT.
  - HTTP Cliente.
  - HTTP Servidor.
  - MQTT Cliente.
  - Blynk App.
- WiFi-Mesh



Con nuestra tarjeta ***ESP32 Plus STEAMakers*** solamente podemos conectarnos a redes de 2.4GHz y no nos podemos conectar a las nuevas redes de 5GHZ.

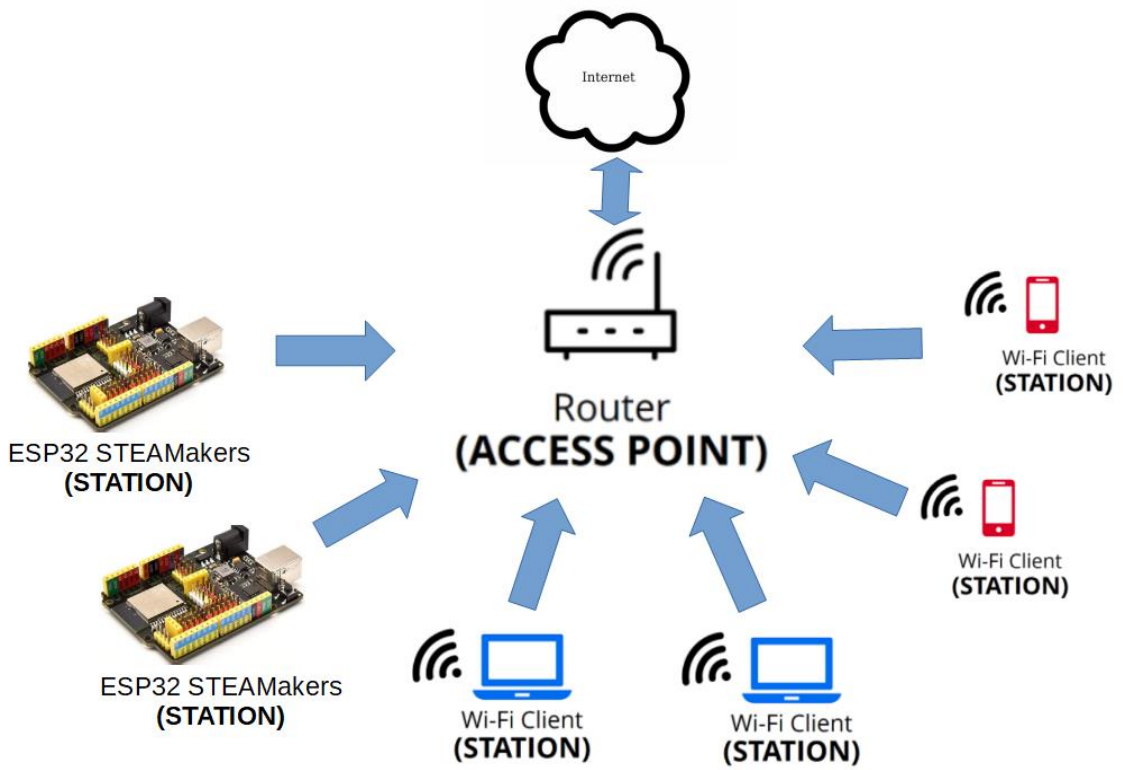
Tenemos diferentes bloques para poder de trabajar con redes Wifi:

- Conectarse a una red Wifi como cliente (modo cliente/estación): esta opción es la más habitual. Con este bloque conectamos la placa ***ESP32 Plus STEAMakers*** a una red Wifi existente. Para ello necesitamos el nombre de la red Wifi (SSID) y la clave (WEP, WPA, etc.).



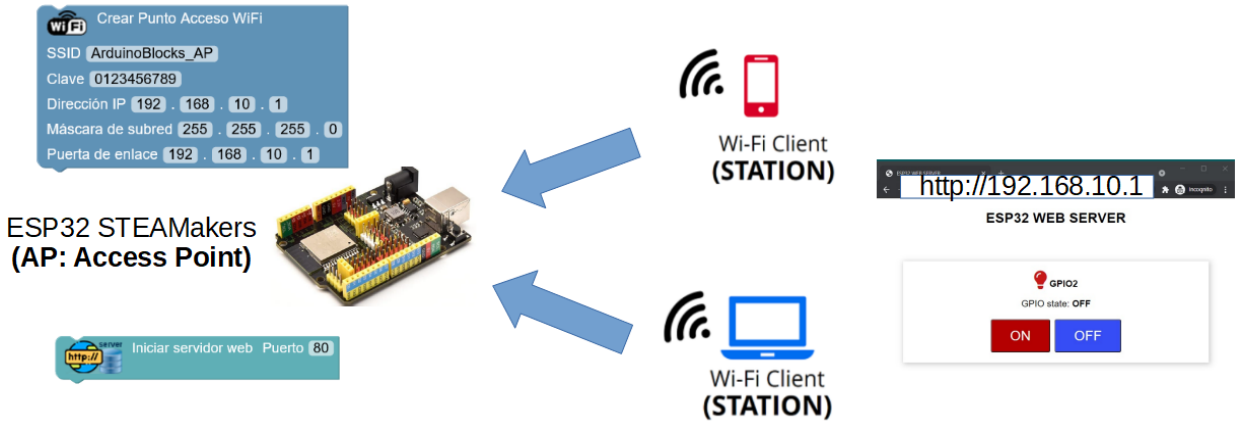
Por defecto, una vez conectados a la red obtendremos la configuración de red (IP, puerta de enlace, etc.) de forma automática (suponiendo que hay un router que lo hace de forma correcta en nuestra red). Si todo es correcto, tendremos acceso a nuestra red WLAN y conexión a internet a través del router.

Este es el caso más simple (y el más habitual). Después de este bloque ya podremos usar cualquiera de los servicios explicados más adelante (Blynk, MQTT, etc.).



- Crear una red Wifi propia (modo punto de acceso): en algún caso, si no tenemos una red Wifi donde conectarnos, el ***ESP32 Plus STEAMakers*** es capaz de crear su propia red Wifi. Significa que crea un punto de acceso donde nos podemos conectar con otros dispositivos de red (portátil, Tablet, móvil, otros ***ESP32 Plus STEAMakers***, etc.).

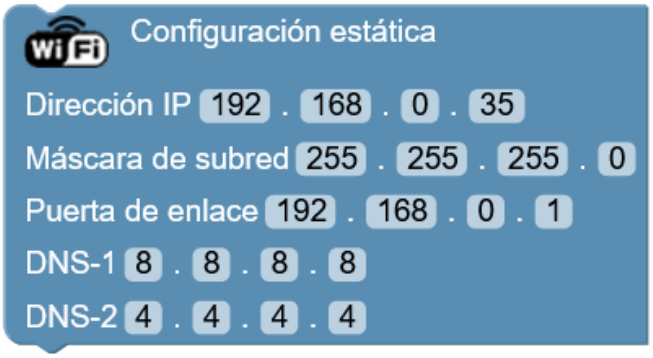
  Hay que tener en cuenta que al crear un punto de acceso Wifi de esta forma, podemos conectar, por ejemplo, nuestro móvil al nuevo Wifi creado, pero no tendrá conexión a internet con esta conexión, simplemente es una forma de enlazar de forma inalámbrica a dos (o más) dispositivos para intercambiar información.



- Cambiar el nombre del dispositivo en la red (Nombre del host - hostname): tenemos un bloque que nos permite cambiar el *hostname* del dispositivo para la red:

- Dirección IP estática (en vez de usar la configuración automática por defecto): en algunos casos nos puede interesar que nuestro dispositivo tenga una IP fija dentro de la red y no depender del router (servidor DHCP) para que nos asigne cada vez una IP distinta en cada conexión a la red.

  Por ejemplo, en casos en los que nuestro dispositivo vaya a ser un servidor web y necesitemos conectarnos a él desde otros dispositivos, o para redes donde el servicio DHCP de IP automática no está activado. Para ello usamos el bloque de configuración de IP estática que permite asignarle al dispositivo una IP y configuración básica de red de forma manual:



  - *Dirección IP:* La dirección que queremos dentro de la red a la que nos vamos a conectar. En un router doméstico uno de los rangos de IP más habituales es 192.168.0.X o 192.168.1.X, siendo la dirección 192.168.0.1 la del router por defecto, y el rango DHCP (las IPs de los dispositivos a los que se les asigna automático) suele empezar en la 192.168.0.100 en adelante (101, 102, …). Por eso, es recomendable en primer lugar saber si hay otros dispositivos con IP fija en la red y hacer un listado de dispositivos e IPs para evitar duplicados de IPs (error en la red). Y, por tanto, las IPs fijas por seguridad deberían estar fuera del rango del DHCP (menor que la 100). Recomendación en este ejemplo: poner una IP entre la 2 y las 99 para no tener conflicto con la IP del router ni con el rango de IPs asignadas automáticamente a otros dispositivos con el DHCP.

- *Máscara de subred:* Es la máscara que nos separa la parte de la red y los dispositivos dentro de la dirección IP. En el ejemplo he dejado la máscara de 24 bits de red y 8 bits de dispositivos por defecto en la mayoría de redes LAN domésticas.

- *Puerta de enlace:* Es la IP a través la cual podemos acceder a otras redes (internet), en nuestro caso debemos poner la IP del router.

- *DNS:* Son dos direcciones IP de servidores DNS en internet para resolver nombres de dominios. Podríamos usar la dirección IP del router, aunque en el ejemplo he puesto las IPs públicas de los servidores DNS de Google.

Realizaremos una breve explicación de qué es una dirección IP y sus características principales.

Una dirección IP son una serie de números que se asignan a un determinado dispositivo tipo ordenador, impresora, teléfono móvil, Tablet o router, etc. que se conecta a una red. Esta dirección sirve para identificar la identidad y ubicación de cualquier dispositivo que sea capaz de utilizar una red en concreto, es decir, una especie de dirección virtual donde enviar y recibir información. Existen dos tipos de direcciones IP como son IPv4 y IPv6. Las IPv4 constan de una serie de cuatro números que van de 0 al 255, separados por puntos, mientras que las IPV6 son de ocho grupos de cuatro dígitos hexadecimales y se representan separados por los dos puntos. En una dirección IP vemos dos partes, los tres primeros números que son el identificador (ID) de red y el último que es el identificador de host.

Existen dos tipos de IP: públicas y privadas.

- IP Pública: aquella que un dispositivo tiene al conectarse a un router y es la misma para todos los que estén en esa misma red.
- IP Privada: la que tiene cada dispositivo dentro de una red local y con la que se le puede identificar para saber quién es.

También existen las direcciones IP estáticas y dinámicas:

- IP dinámicas: aquellas que se asignan de manera aleatoria a nuestro equipo cada vez que se conecta. Es decir, no siempre vamos a tener la misma dirección IP, siendo diferentes cada vez que conectemos. Se suelen utilizar en redes domésticas. El inconveniente es que no es posible saber a quién pertenece realmente una IP y es difícil encontrar un determinado equipo desde fuera de la red local.
- IP estáticas: suelen ser las que se instalan en empresas y se le asigna la misma IP para cada puesto, sabiendo que equipo está utilizando esa dirección y teniendo siempre la misma dirección.

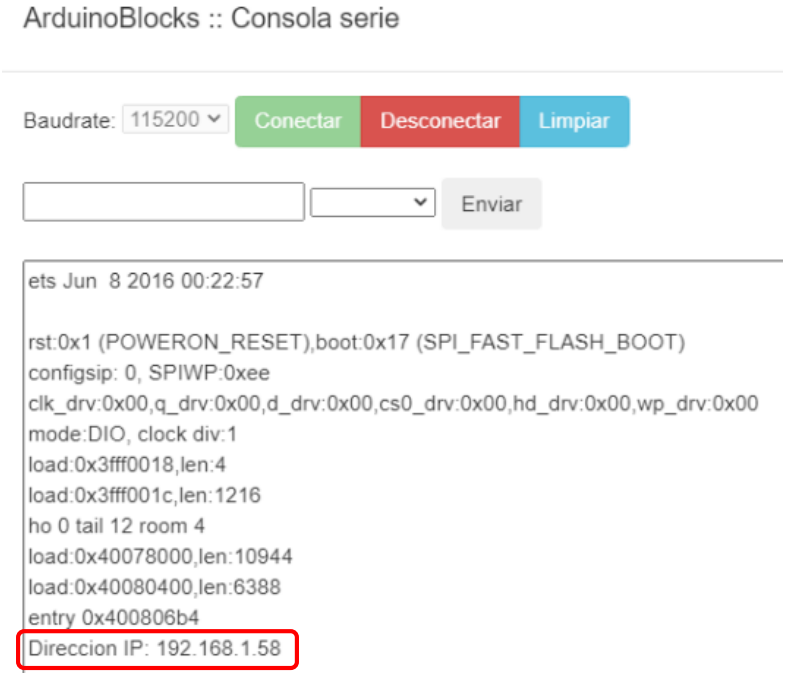
Con todo lo visto hasta ahora podemos conectarnos a una red Wifi o crear nuestra propia red Wifi. Pero también podemos conectarnos a Blynk, MQTT, un servidor HTTP (web), etc. A continuación, vamos a trabajar con algunos de estos recursos.

7.22.1 Reto A22.1. Conexión a una red Wifi.

Reto A22.1. Conexión a una red Wifi.

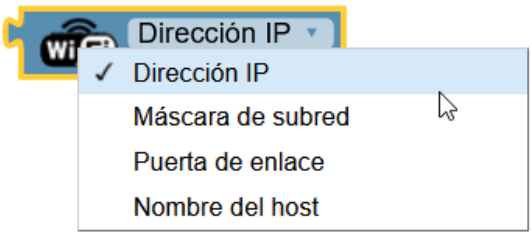
El primer programa que vamos a realizar es un sencillo programa que nos permitirá conectarnos a una red Wifi y comprobar si se ha realizado la conexión enviando por el puerto serie (*Consola*) la dirección IP asignada.

Podemos observar la dirección IP y, por lo tanto, que se ha establecido la comunicación.



También podemos otra serie de datos de nuestra conexión. Como podemos ver en el menú desplegable del bloque, podemos obtener los siguientes datos:

- Dirección IP.
- Máscara de subred.
- Puerta de enlace.
- Nombre del host.

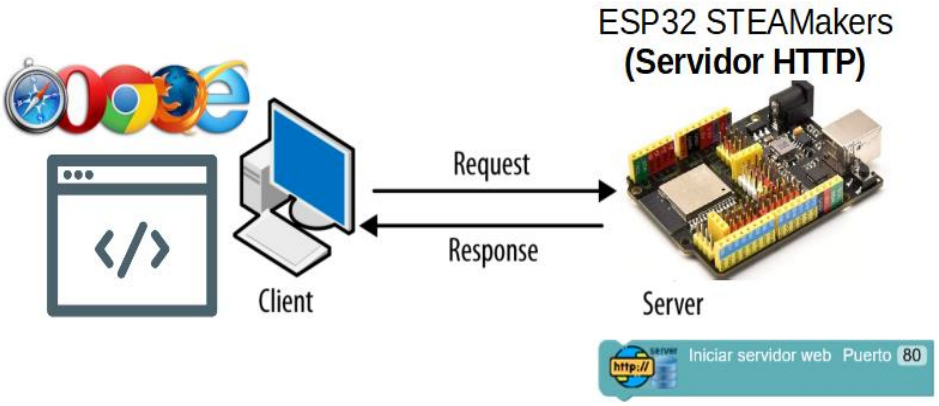


Actividad de ampliación: realiza el programa anterior y comprueba su funcionamiento.

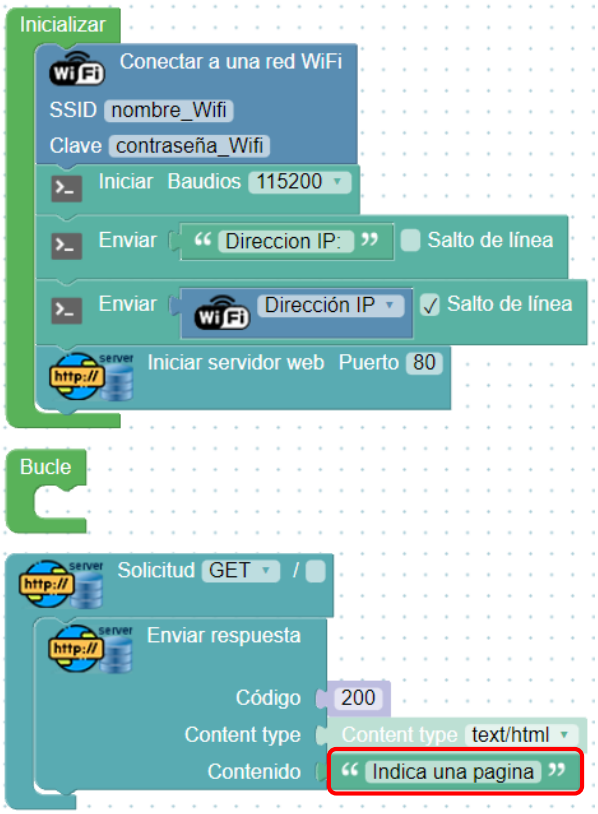
7.22.2 Reto A22.2. Servidor HTTP (Web) I.

Reto A22.2. Servidor HTTP (Web) I.

La placa ***ESP32 Plus STEAMakers*** permite implementar un servidor web de forma sencilla y potente. Un servidor web es un servicio de red que está esperando conexiones. Cuando un dispositivo se conecta le hará una petición en formato HTTP (Hiper Text Transfer Protocol) que normalmente será una página web (documento HTML) o un archivo de otro tipo: CSV, texto, imagen, etc. No siempre el servidor debe retornar datos útiles, el servidor web puede utilizarse como una interfaz para control remoto, permitiendo intercambiar datos y ejecutar acciones remotas.

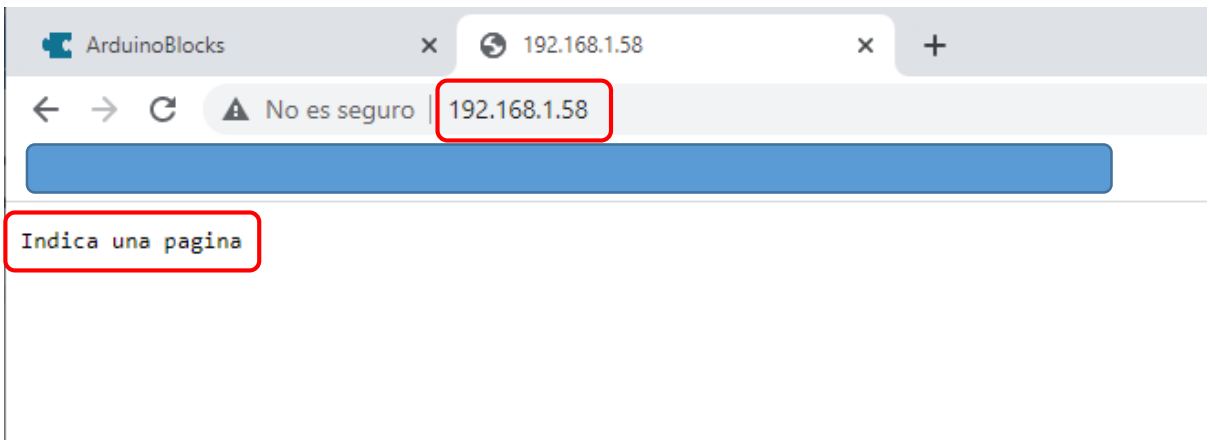


Vamos a crear un servidor web como interfaz de control remoto.

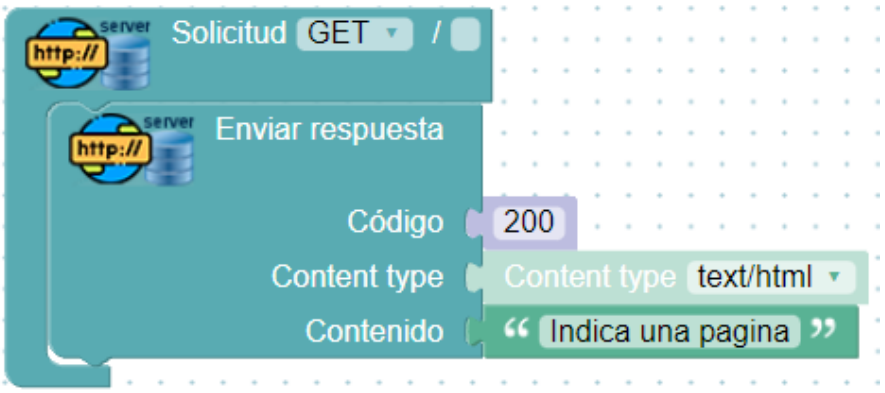


Consultamos la IP asignada desde la *Consola*, en este caso 192.168.1.58:

En el explorador, indicamos la dirección IP y nos mostrará el mensaje.



En este ejemplo, la respuesta se compone de 3 partes importantes:



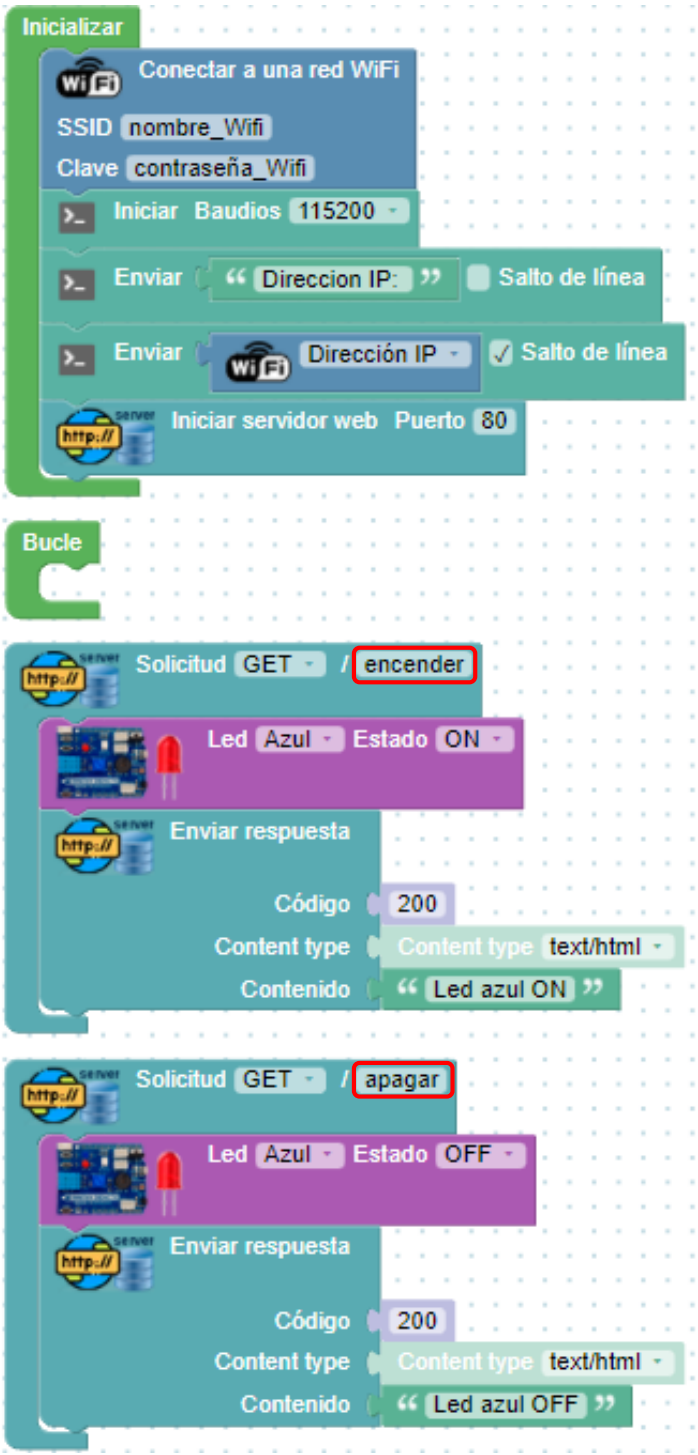
- Código: 200 - Código de respuesta HTTP OK.
- Content-Type: tipo de contenido que se va a enviar, normalmente text/html, pero podría ser text/CSV, text/xml, etc.
- Contenido: el contenido tal cual que se envía, para hacerlo correcto debería ser texto formateado en HTML de forma correcta. Abusando de la capacidad de los navegadores actuales para procesar casi cualquier tipo de contenido "*mal formateado*" le enviamos un sencillo texto de saludo.

Actividad de ampliación: realiza el programa anterior y comprueba su funcionamiento.

7.22.3 Reto A22.3. Servidor HTTP (Web) II.

Reto A22.3. Servidor HTTP (Web) II.

Vamos a realizar un programa para poder encender y apagar el led azul desde el navegador. Solamente cambiaremos el contenido de la *Solicitud*, pero, además, será crearemos dos solicitudes para poder realizar las dos acciones sobre el led.



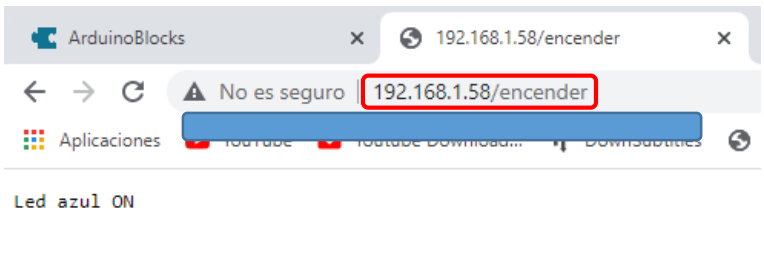
Las peticiones HTTP las haremos normalmente desde un navegador, poniendo la IP de nuestro dispositivo en la barra de direcciones con el protocolo http. Por ejemplo, si nuestro dispositivo tiene la dirección IP 192.168.1.58:

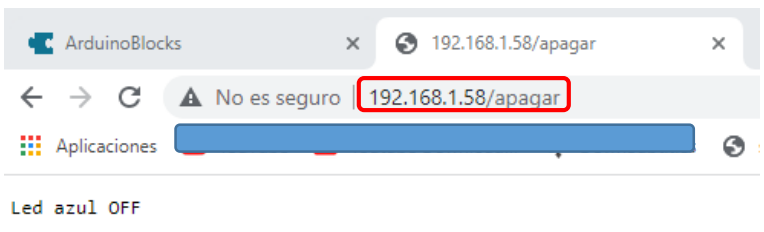
El formato de la URL que podemos solicitar a nuestra ***ESP32 Plus STEAMakers*** es:

http://direccionip/accion?parametros

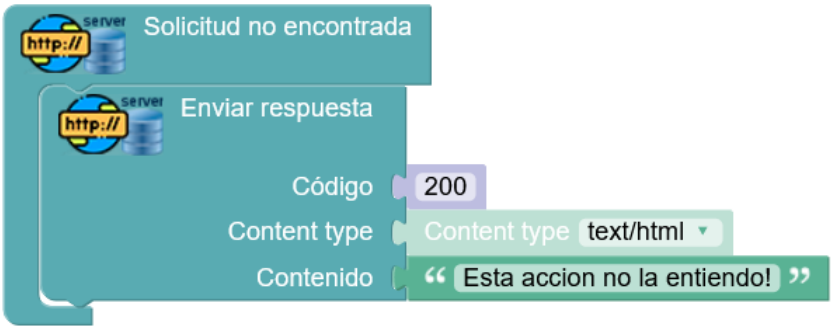
- La dirección IP es la que tenga el dispositivo, como hemos visto anteriormente en este ejemplo en mi red doméstica la IP es 192.168.0.155 (variará en cada red, y según el DHCP asigne en cada momento o si la ponemos de forma fija)
- La acción (opcional) será la web o comando que queremos pedirle al servidor. En un servidor web real (este es real, me refiero a uno tradicional tipo Apache, etc…) sería el documento del servidor que estamos solicitando (ejemplo: index.html, index.php, guardardatos.jsp, etc.).
- Los parámetros (en este caso parámetros tipo GET) son datos adicionales que le damos al servidor normalmente como información extra a la acción solicitada. Los parámetros GET son opcionales, siempre van después del símbolo "*?*" y se forman con pares "*clave=valor*" separados entre ellos por "&".

Desde el navegador encenderemos y apagaremos el led azul, utilizando los comandos que hemos creado en la *Solicitud*.

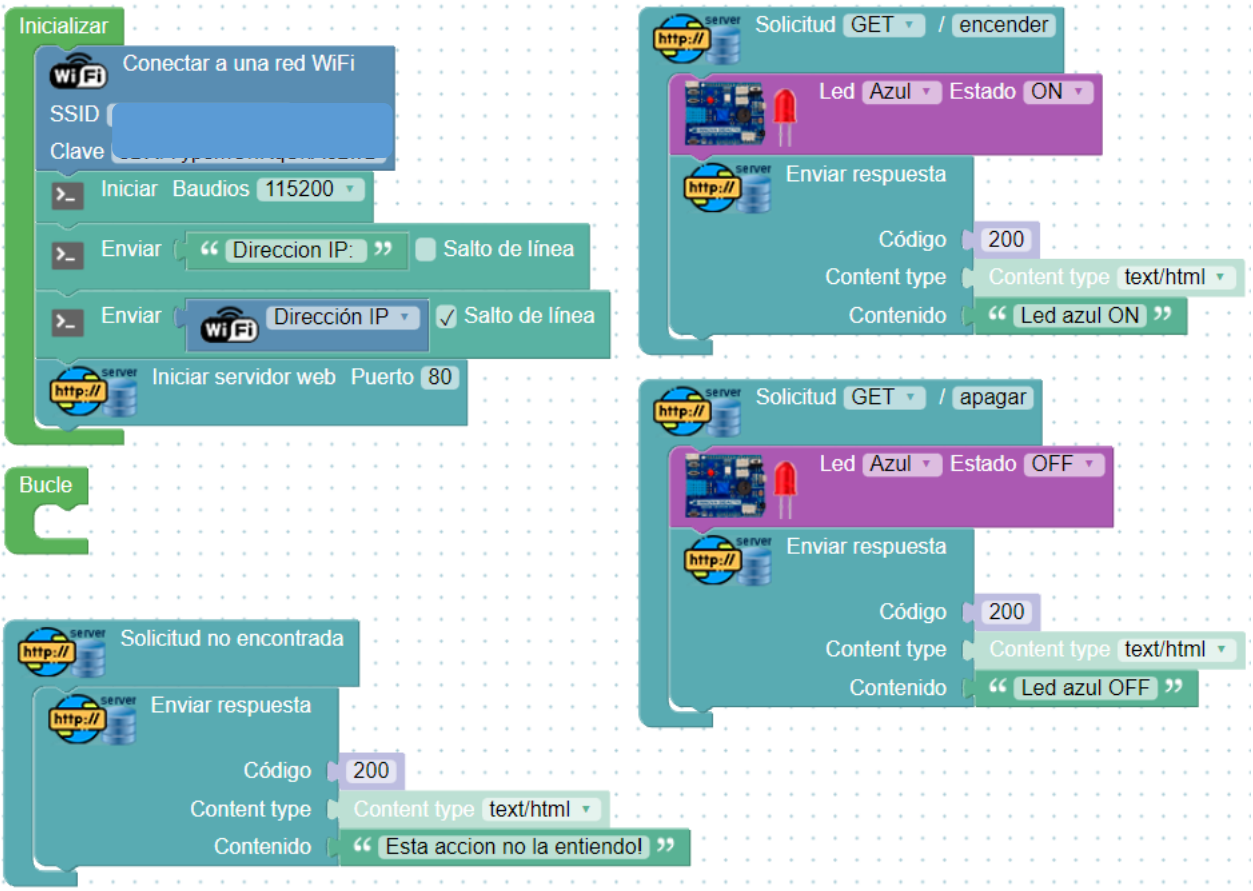




Por otro lado, deberíamos añadir otra respuesta para cuando se solicite una acción no válida. Por ejemplo, si pido la acción *parpadear* cuando sólo tengo implementadas la acción de *encender* y *apagar* debería indicar un error.



El programa definitivo será:



Enlace al programa: ArduinoBlocks Projects\http_2.abp

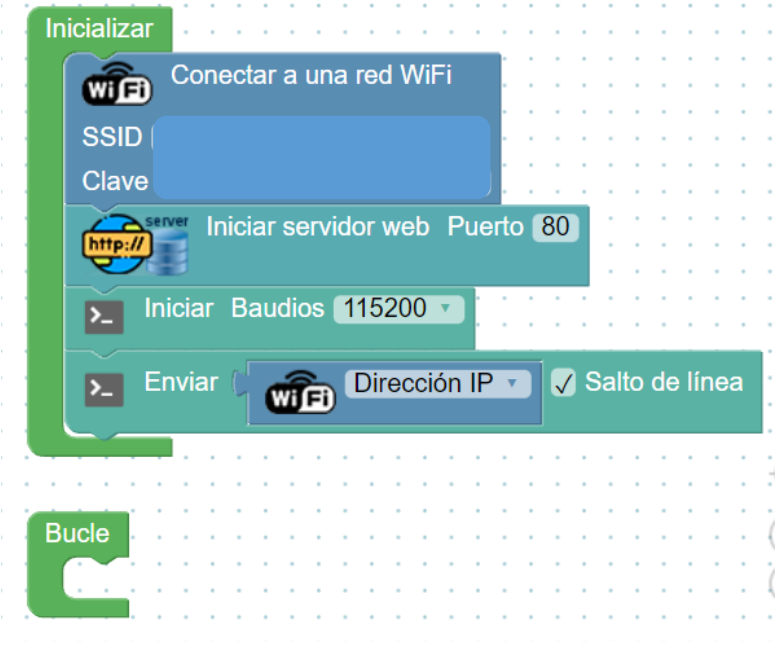
Actividad de ampliación: realiza el programa anterior y comprueba su funcionamiento.

7.22.4 Reto A22.4. Servidor HTTP (Web) III.

Reto A22.4. Servidor HTTP (Web) III.

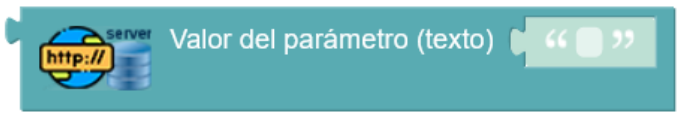
Basándonos en la idea del ejemplo anterior, vamos a hacer uso de los parámetros *GET* para permitir que nuestro servidor web sea capaz de encender o apagar cualquier led conectado a cualquiera de los pines disponibles.

Iniciar servidor web (80) y detectaremos que dirección IP tenemos asignada, igual que en la actividad anterior.

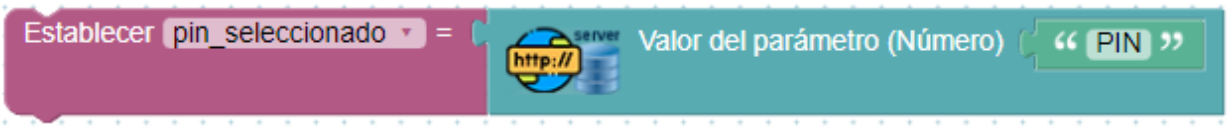


Realizaremos las funciones para encender y apagar cualquier led (probaremos con un led conectado en D3 – pin IO25).

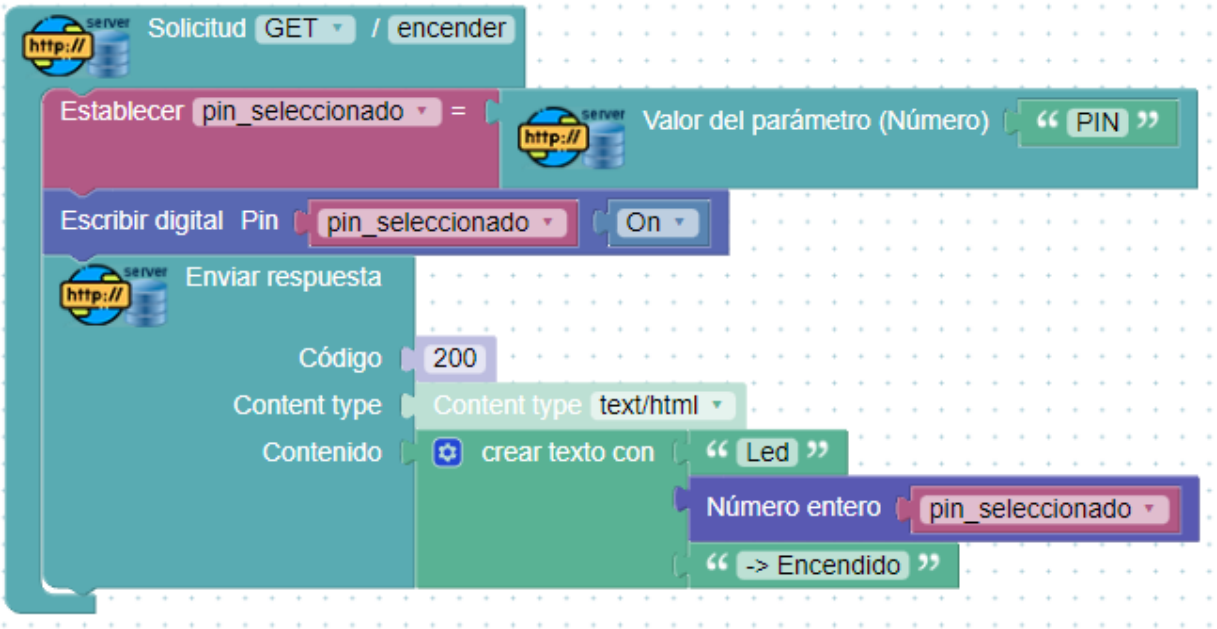
Con los bloques "*valor del parámetro*" obtenemos parámetros de la URL.



En el ejemplo usaremos la versión del bloque que obtiene el parámetro y lo procesa automáticamente como un valor numérico.

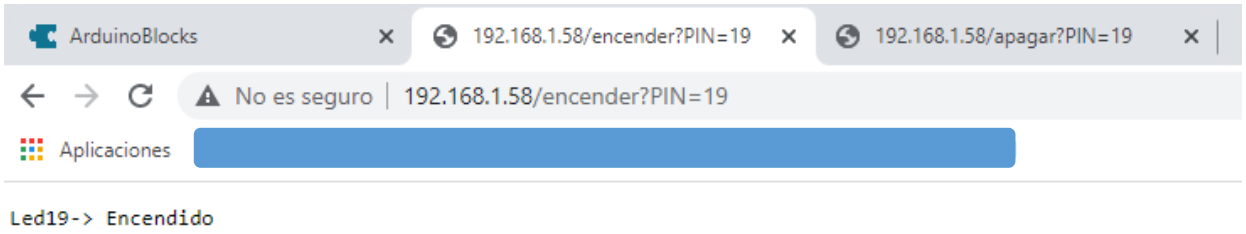


Vamos a realizar las dos funciones que nos permitirán encender y apagar cualquier led. La función para **encender** el led sería:



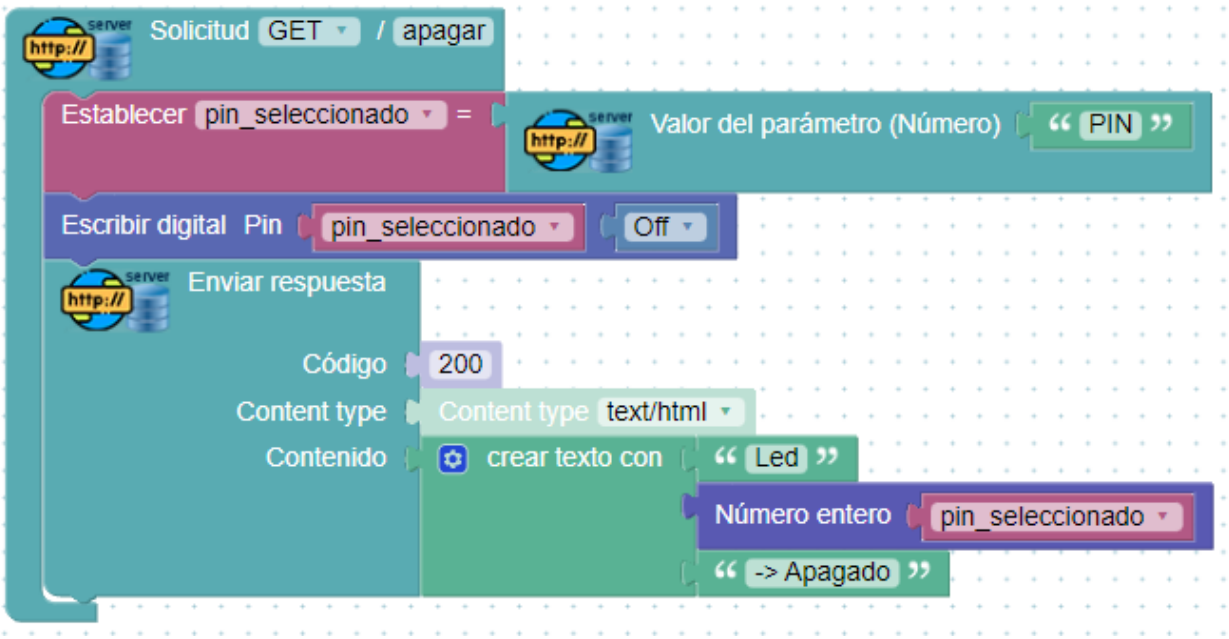
La petición correcta para encender el led anterior, conectado al pin 19 (GPIO 19 o D12-Led Rojo) sería:

http://192.168.1.58/encender?PIN=19



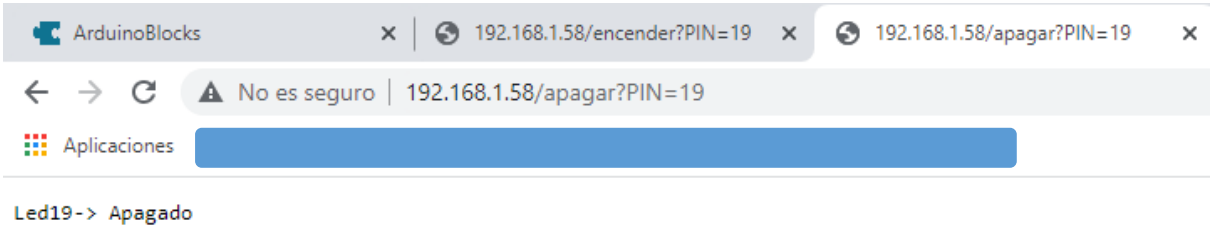
Y así podríamos indicar cualquiera de los pines GPIO disponibles para activarlo a *ON*.

La función para **apagar** el led sería:



La petición correcta para apagar el led anterior, conectado al pin 19 (GPIO 19 o D12-Led Rojo) sería:

http://192.168.1.58/apagar?PIN=19



Y así podríamos indicar cualquiera de los pines GPIO disponibles para activarlo a *OFF*.

Enlace al programa: ArduinoBlocks Projects\http_3.abp

Actividad de ampliación: realiza el programa anterior y comprueba su funcionamiento.

7.22.5 Reto A22.5. Servidor HTTP (Web) IV.

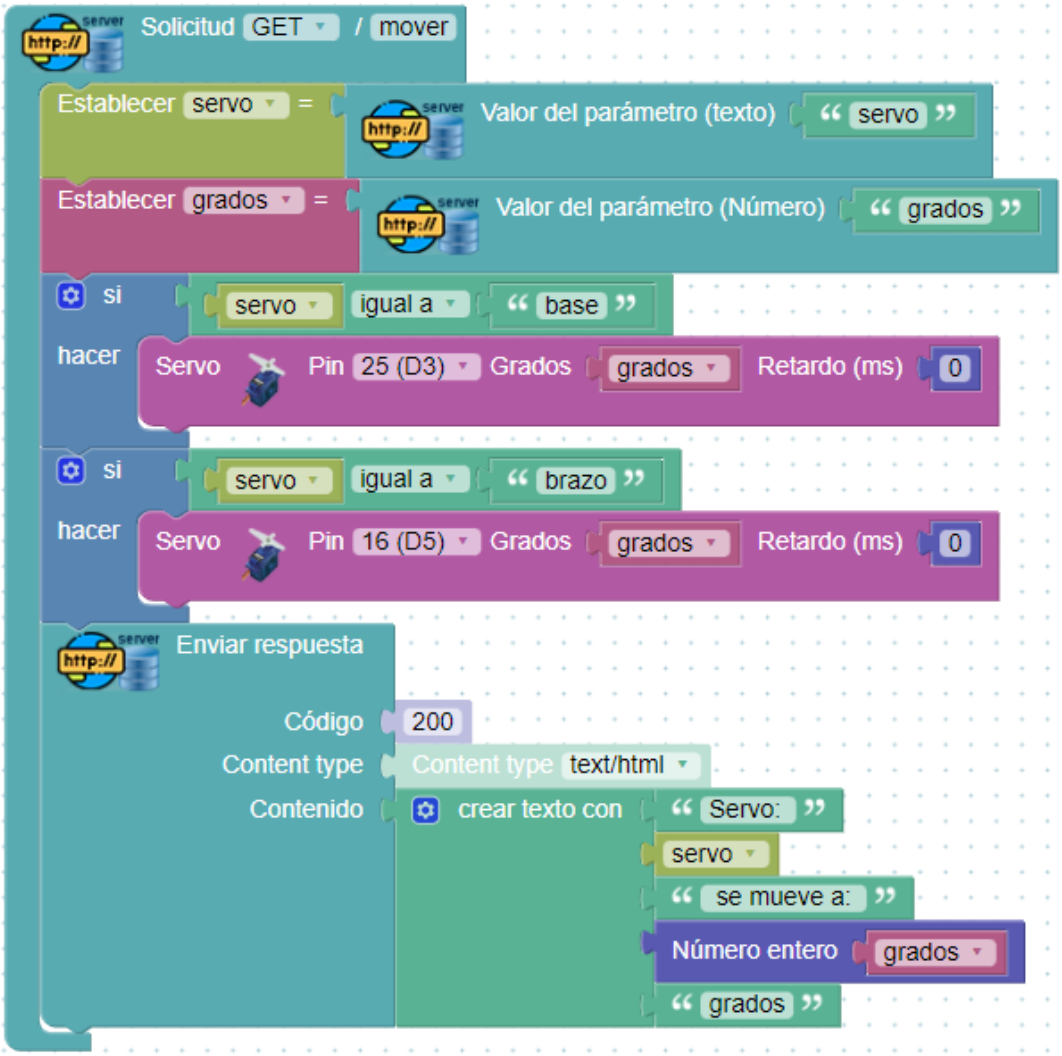
Reto A22.5. Servidor HTTP (Web) IV.

Con lo aprendido hasta ahora realizar el control de varios servos vía peticiones HTTP desde un navegador.

El bucle Inicializar seguirá igual que en las actividades anteriores.

Vamos a conectar 2 servos simulando un brazo robótico con una base, un brazo y una pinza. En **D3** conectaremos la **base** y en **D5** el **brazo** (no lo podemos conectar en A3 porque es un pin sólo de entradas). Definiremos la acción *mover* en la que introduciremos 2 parámetros:

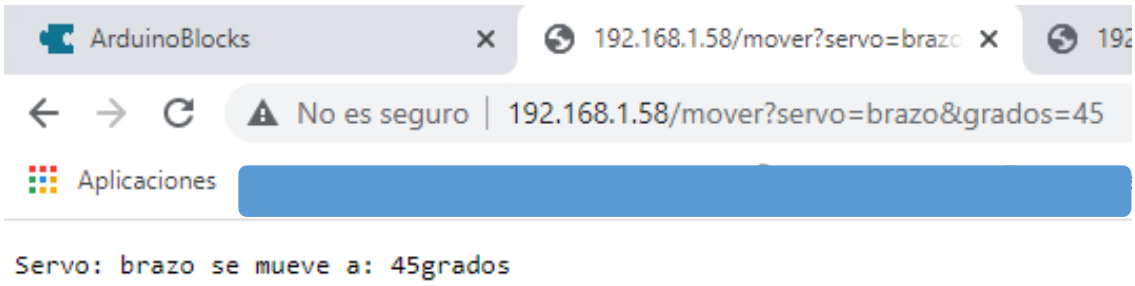
- ***servo****:* para indicar el servo (a cada uno le hemos puesto un nombre)
- ***grados****:* para indicar la posición a donde mover ese servo indicado.



Enlace al programa: ArduinoBlocks Projects\http_4.abp

Para mover los servos utilizaremos comando del tipo:

http://192.168.1.58/mover?servo=brazo&grados=45



Otros ejemplos válidos:

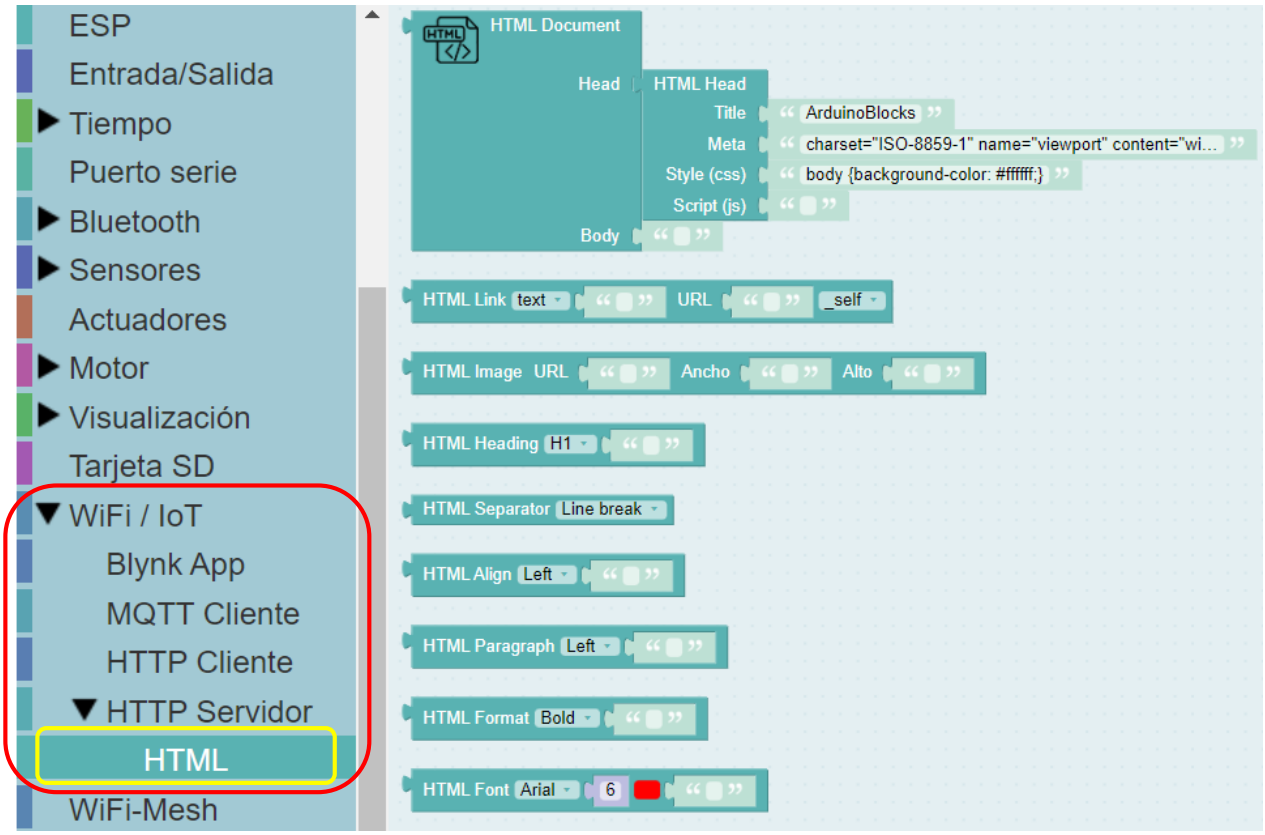
http://192.168.1.58/mover?servo=base&grados=0

Actividad de ampliación: realiza el programa anterior y comprueba su funcionamiento.

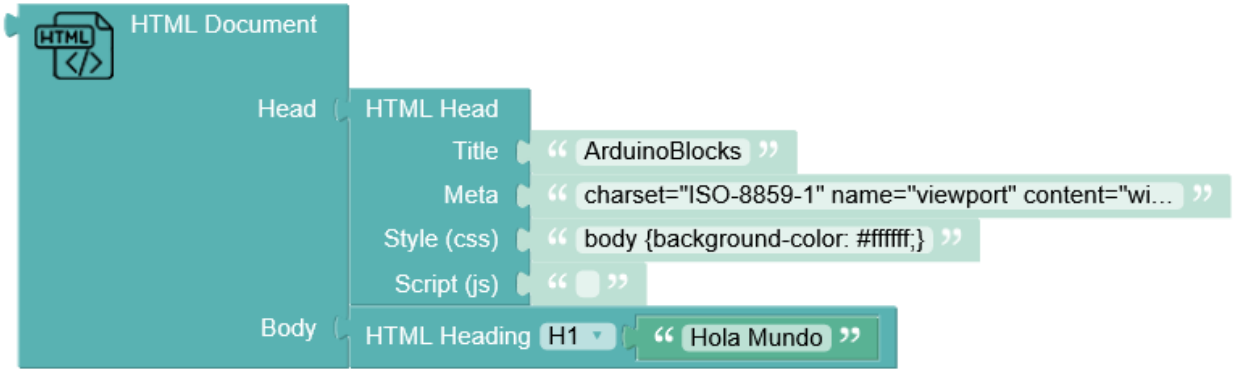
7.22.6 Reto A22.6. Contenido HTML.

Reto A22.6. Contenido HTML.

Podemos generar contenido HTML con ArduinoBlocks. Disponemos de una serie de bloques específicos que nos permiten generar texto con formato HTML especialmente pensado para devolver contenido en formato web desde el servidor HTTP de la placa ***ESP32 Plus STEAMakers***.



Cada bloque implementa una (o varias) etiquetas básicas del lenguaje de marcas HTML. Un ejemplo de una web básica de "hola mundo".



El contenido es simplemente texto formateado según el estándar HTML. Podemos devolver respuestas en formato HTML en las peticiones del servidor HTTP, por ejemplo, esto generaría un documento HTML en caso de solicitar una web no encontrada, con un mensaje de ERROR en grande y color rojo.

Primero realizaremos la programación de *Inicializar* con los siguientes elementos:

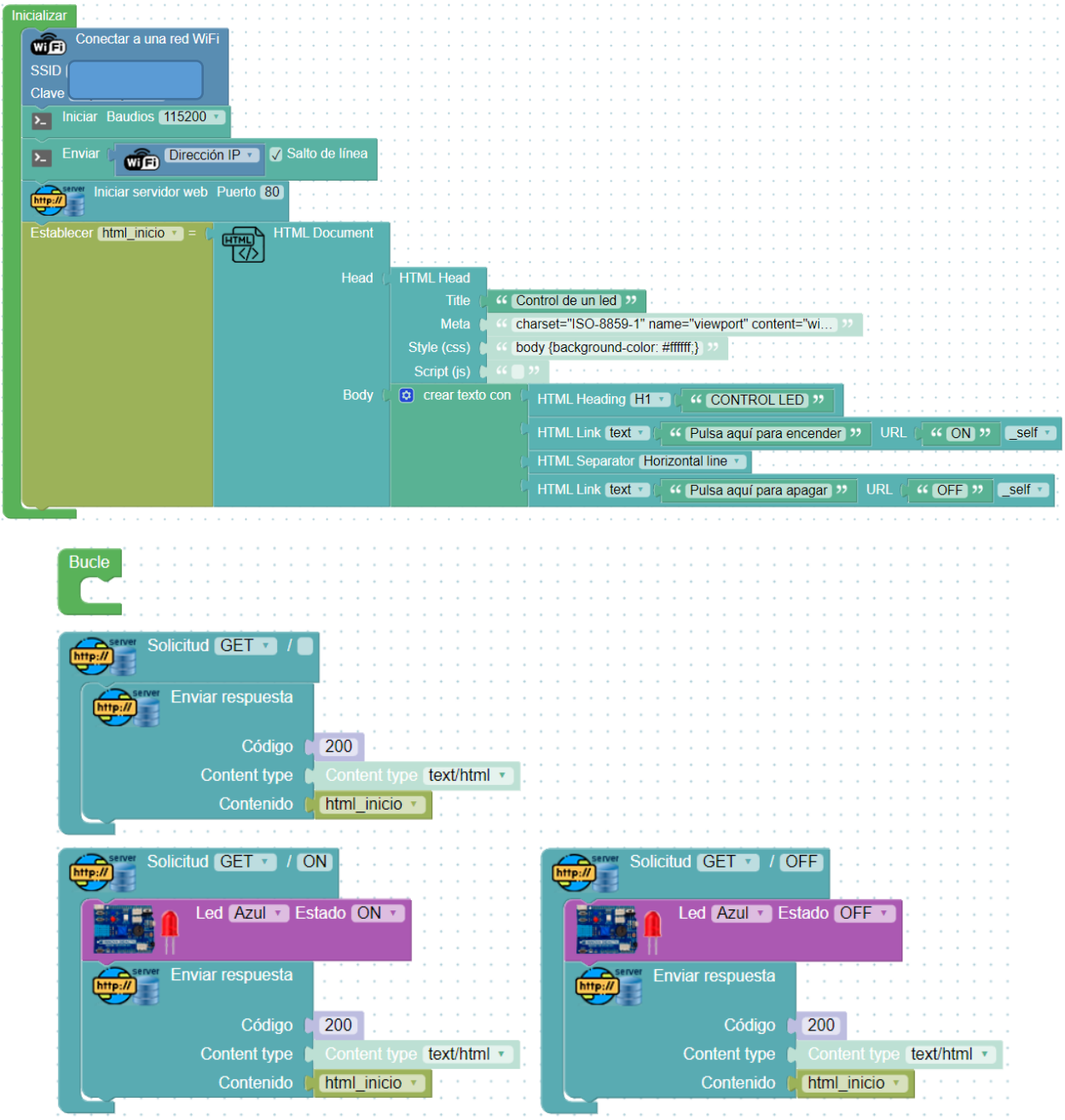
- *Conectar a una red Wifi*.
- Configuraremos el Puerto serie a 115200 baudios.
- Enviaremos la IP por el puerto serie para poder verla por la *Consola*.
- Iniciaremos el servidor web.
- Crearemos una variable de tipo texto donde introduciremos el documento HTML (*HTML Document*).

Después crearemos las tres funciones para encender y apagar el led azul, y la página de inicio.

Subimos el programa a la placa y abrimos la *Consola* para ver la IP asignada:

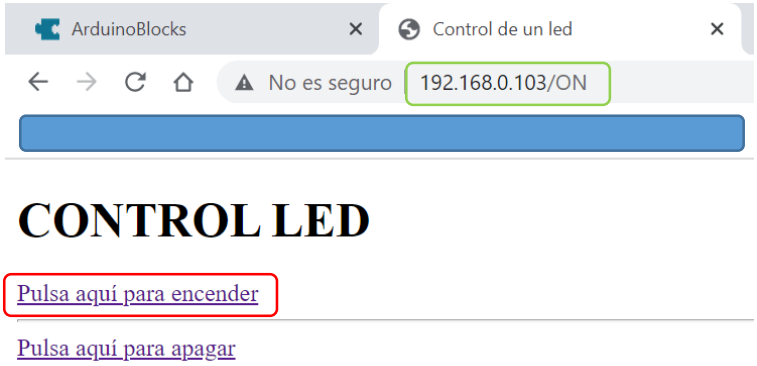
Enlace al programa: [ArduinoBlocks Projects\html.abp]

El programa resultante será:

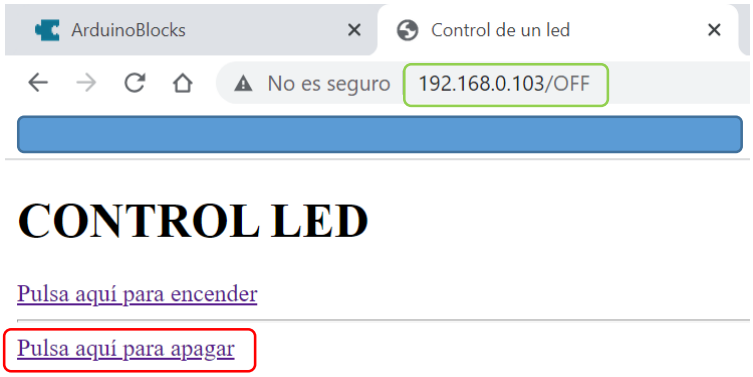


En el navegador introducimos nuestra IP y aparecerá la página web de inicio.

Cuando pulsemos en *Pulsa aquí para encender* se mostrará la siguiente imagen y se encenderá el led azul:



Cuando pulsemos en *Pulsa aquí para apagar* se mostrará la siguiente imagen y se apagará el led azul:



Actividad de ampliación: realiza el programa anterior y comprueba su funcionamiento.

7.22.7 Reto A22.7. Servidor HTTP (Web) V.

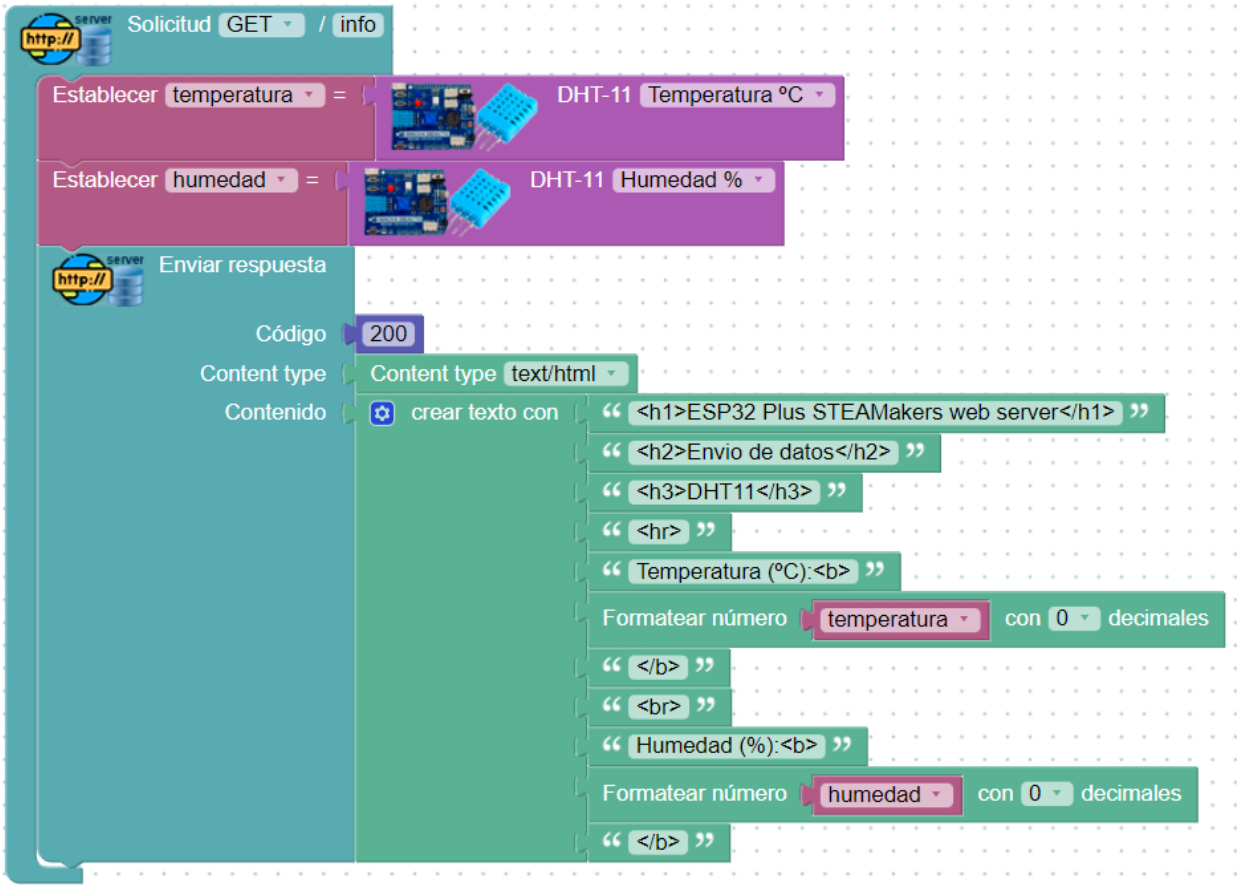
Reto A22.7. Servidor HTTP (Web) V.

Obtener una web con la información de un sensor DHT11 que mide temperatura y humedad:

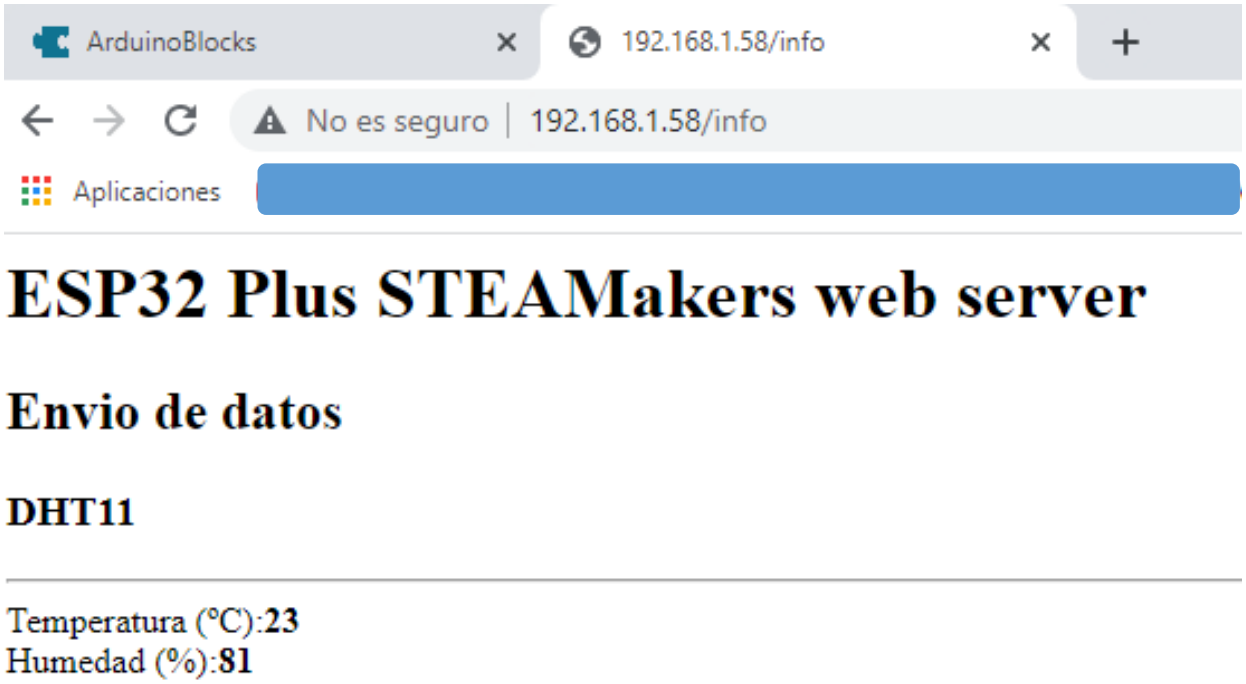
El bucle Inicializar seguirá igual que en las actividades anteriores.

La acción *info* nos enviará la información de temperatura y humedad en una página web. Utilizaremos para formatear la web de una forma más vistosa se utilizan códigos HTML básicos como (h2, h3, hr, b), para información sobre códigos HTML básicos podemos consultar en sitios online personalizados (https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/aprende-html-tutorial-para-principiantes/):

Veamos el bloque resultante:



El resultado de la web generada por nuestra placa ***ESP32 Plus STEAMakers*** con el sensor DHT11 es el siguiente (cada vez que actualizamos la página se regenera con los datos actualizados).



Para dejarlo más vistoso (asumiendo que nuestra red tiene internet) podemos insertar en el código HTML una imagen con la URL completa de internet.

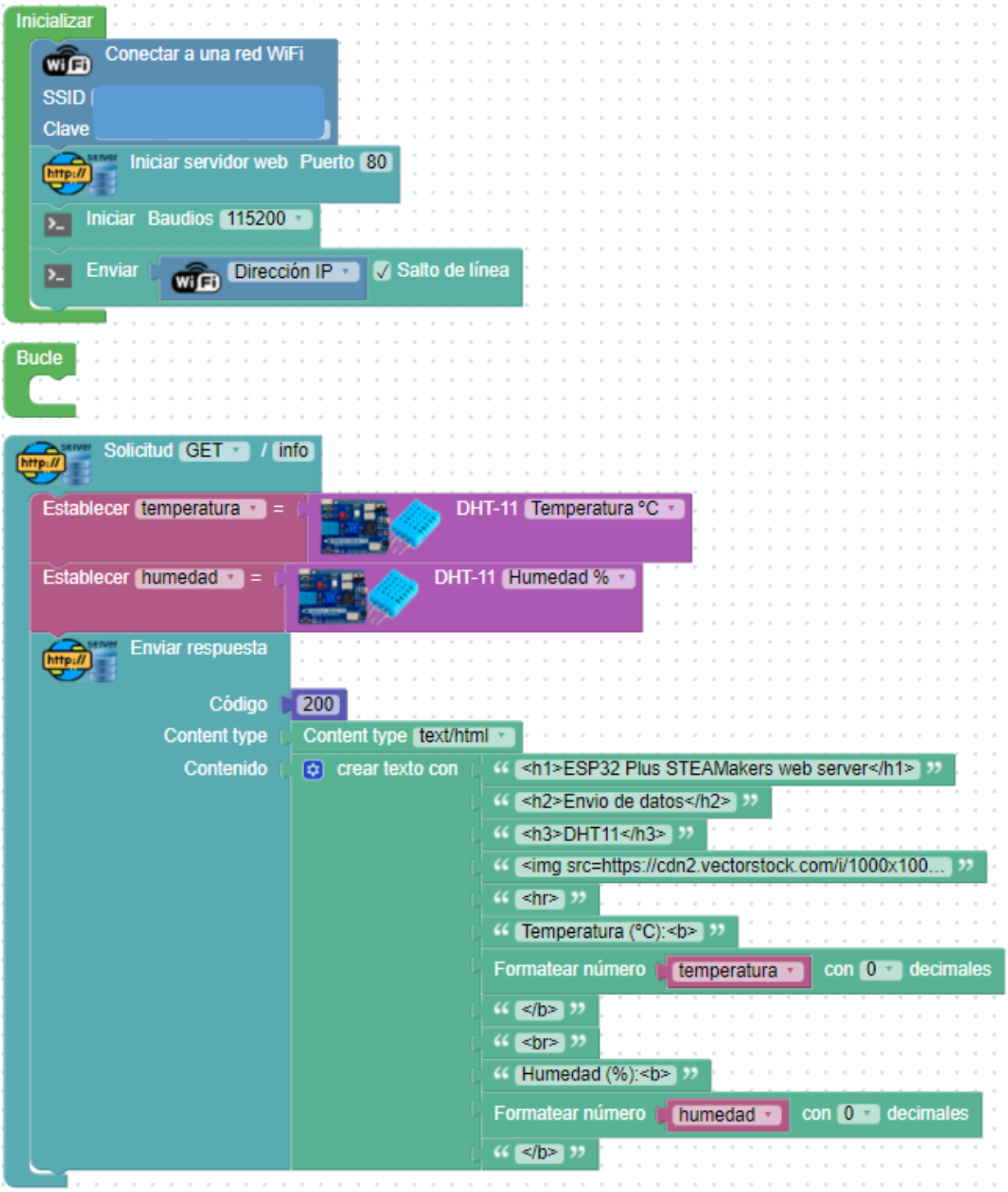
Buscamos un icono de temperatura y humedad y obtenemos su url:

https://cdn2.vectorstock.com/i/1000x1000/75/91/humidity-icon-vector-24247591.jpg

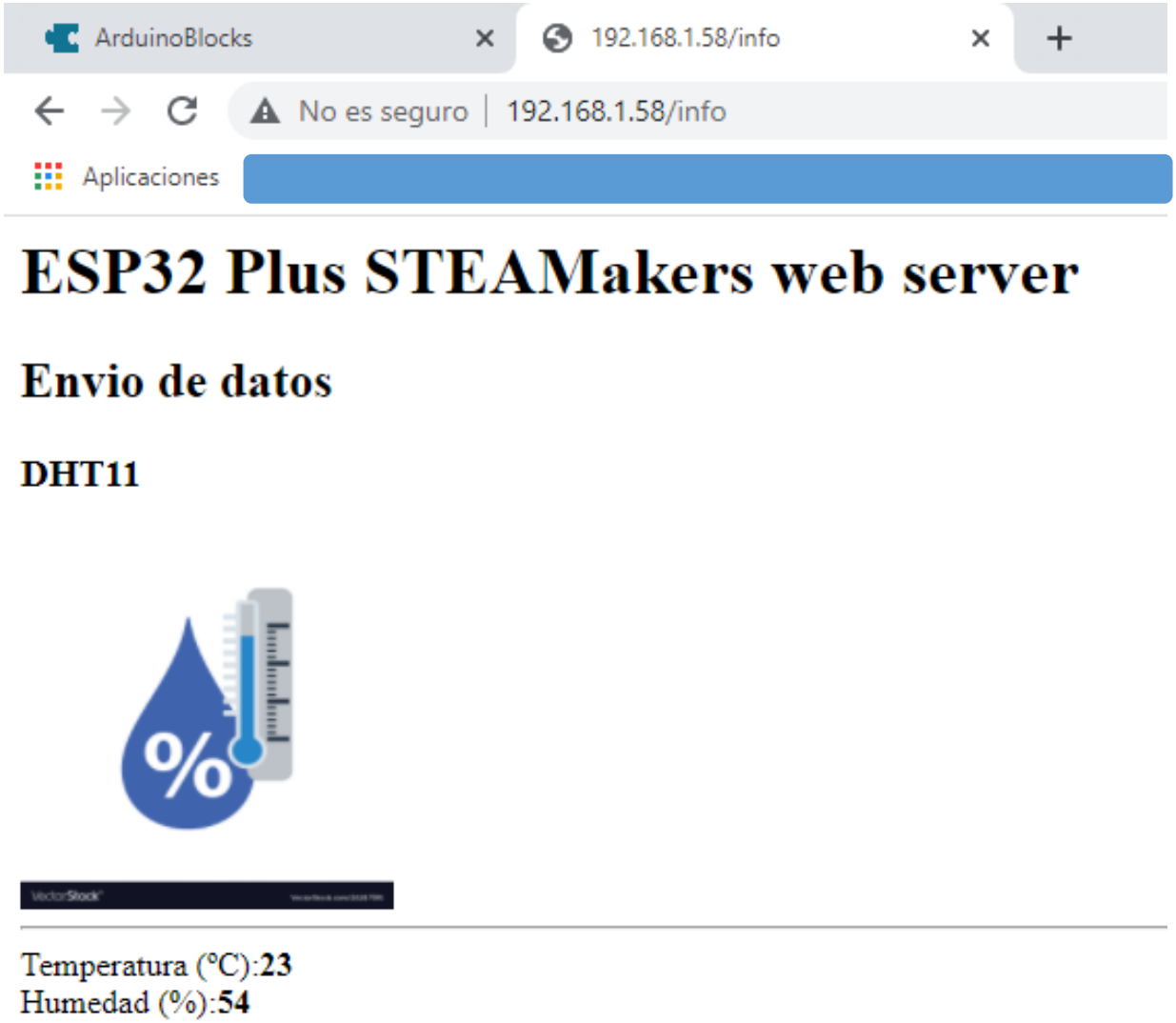
Y añadimos el código en el servidor web de la ***ESP32 Plus STEAMakers***.

Enlace al programa: ArduinoBlocks Projects\http_5.abp

El programa resultante final será:



Y los datos mostrados:



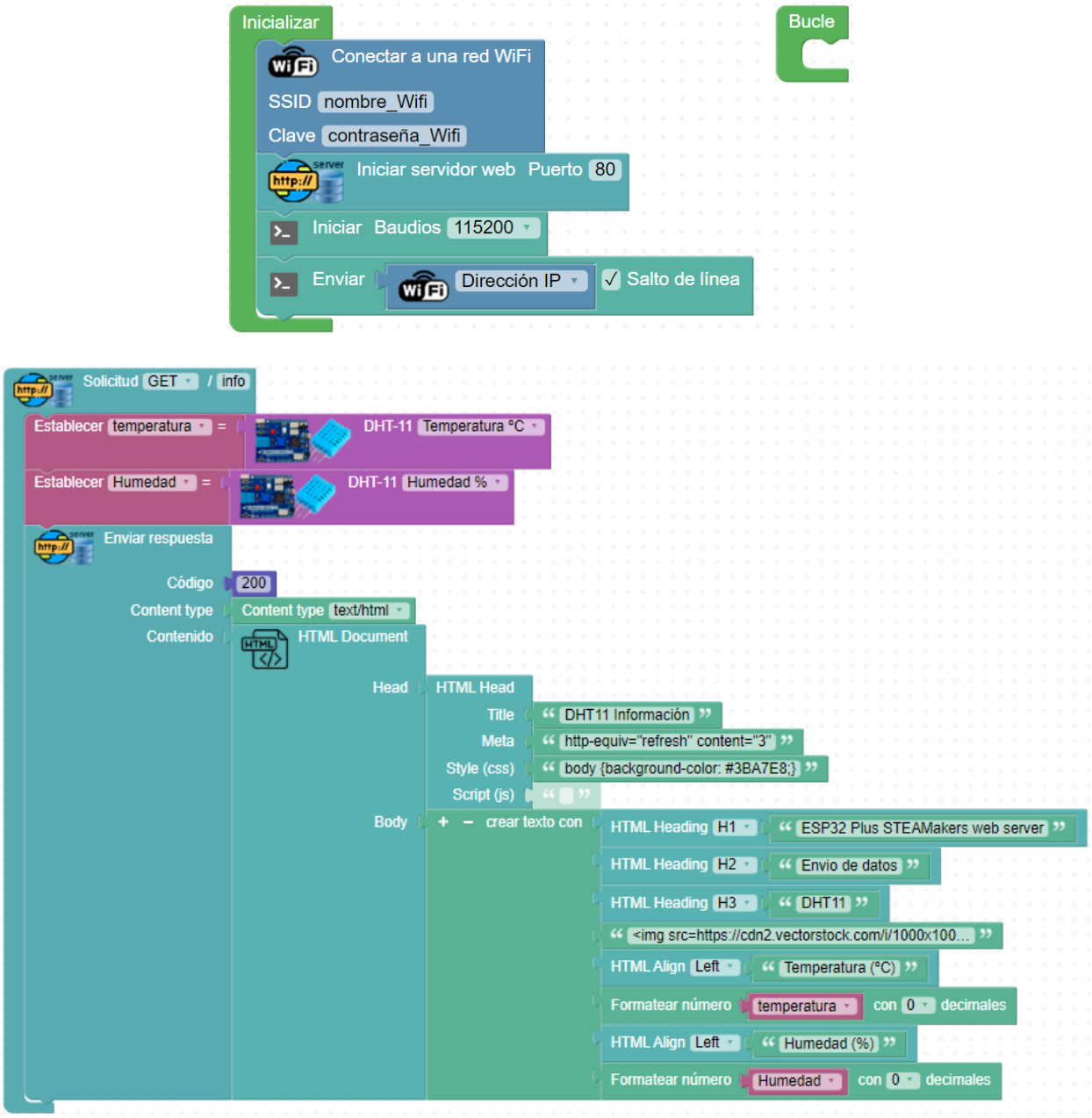
Actividad de ampliación: realiza el programa anterior y comprueba su funcionamiento.

7.22.8 Reto A22.8. Servidor HTTP (Web) VI.

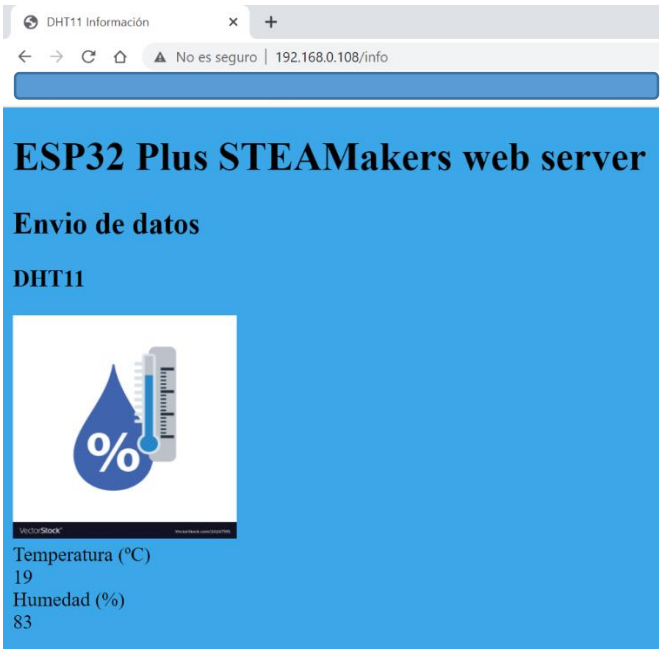
Reto A22.8. Servidor HTTP (Web) VI.

Podemos mejorar el programa anterior con nuevos bloques existentes en ArduinoBlocks que nos permiten modificar directamente la programación HTML de nuestra web.

Cambiaremos el color de fondo y haremos que se actualice automáticamente cada 3 segundos.



La página web que aparecerá y se recargará automáticamente cada 3 segundos será la siguiente:

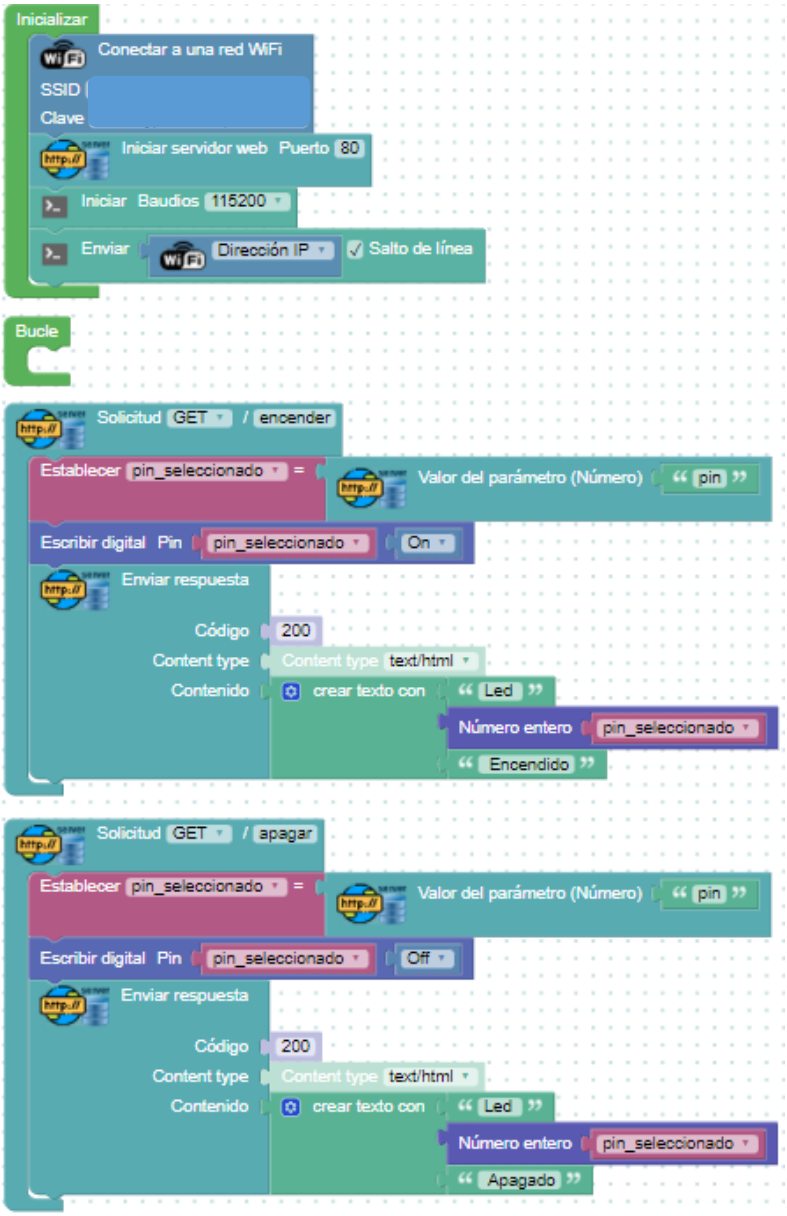


7.22.9 Reto A22.9. Servidor HTTP (Web) – AppInventor2.

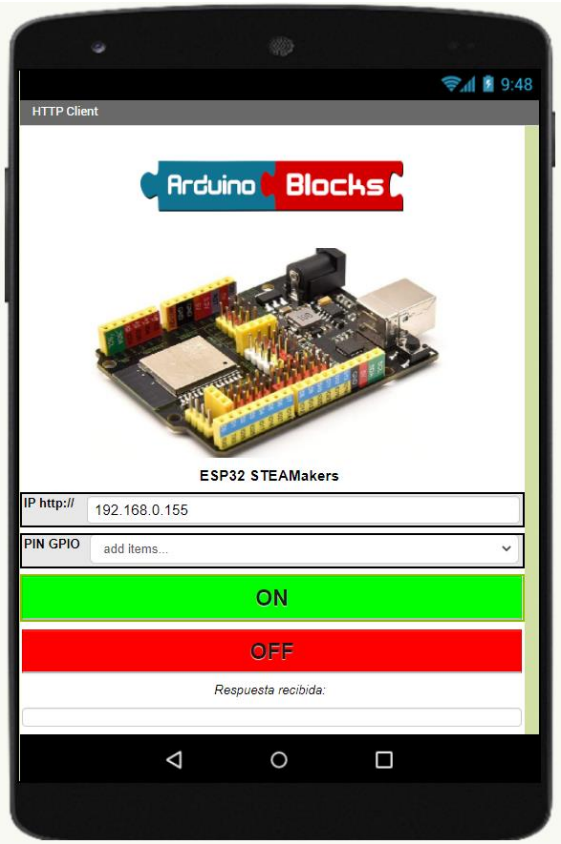
Reto A22.9. Servidor HTTP (Web) – AppInventor2.

Las peticiones HTTP con parámetros GET son muy fáciles de probar y usar con el navegador. Pero no siempre usamos un navegador, si no que implementamos esta opción para comunicar y controlar desde otros sistemas más personalizados. Por ejemplo, podríamos hacer una aplicación móvil con *AppInventor2* que envía peticiones HTPP (de forma transparente) desde la aplicación.

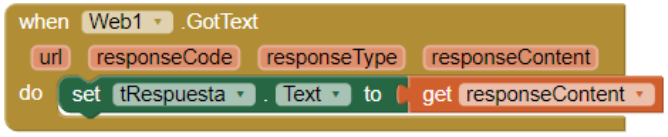
Haremos el programa en ArduinoBlocks para encender y apagar el led. Podremos elegir el led que queremos encender.

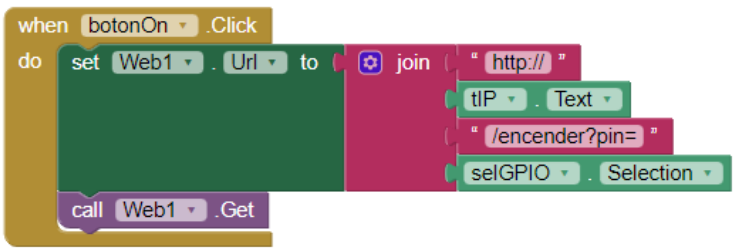


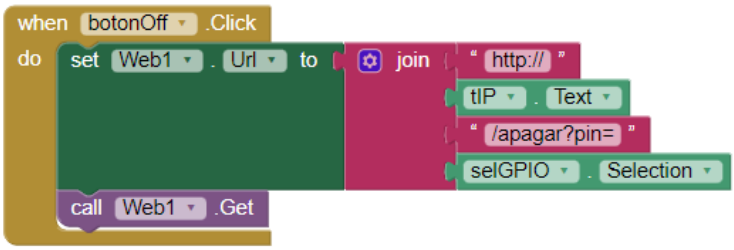
Podemos diseñar una sencilla aplicación en *Appinventor2* para probar el ejemplo anterior. Crearemos dos botones y un componente *web* que es lo único que necesitamos.



En los bloques de código de *Appinventor2* definimos las peticiones HTTP al servidor de nuestra ***ESP32 Plus STEAMakers*** con los parámetros para encender o apagar.

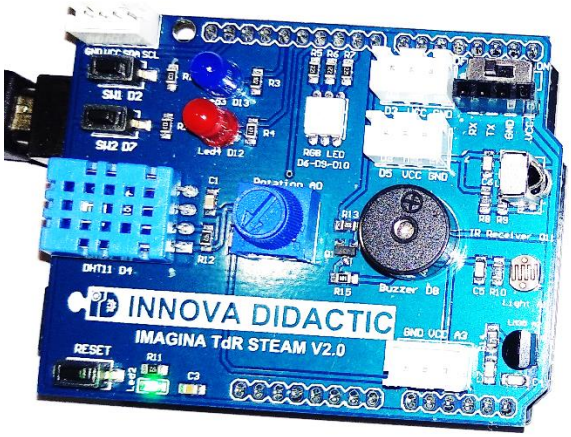






Descargamos la aplicación en el teléfono móvil, la instalamos y ya la podemos probar.

Entramos en la aplicación nuestra dirección IP y el pin de conexión (en el ejemplo el GPIO19 que es el D12 – Led Rojo).

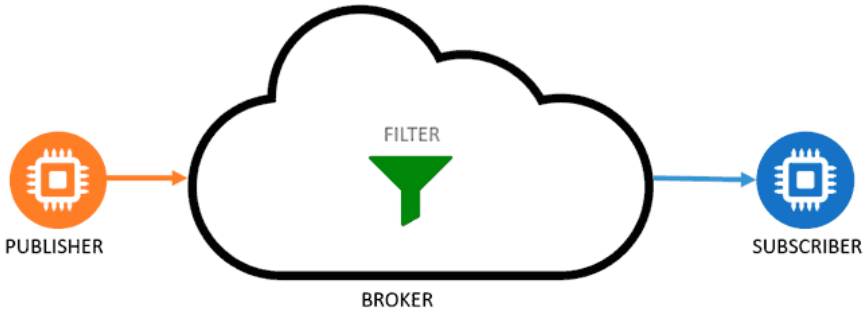


Actividad de ampliación: realiza los programas anteriores y comprueba su funcionamiento.

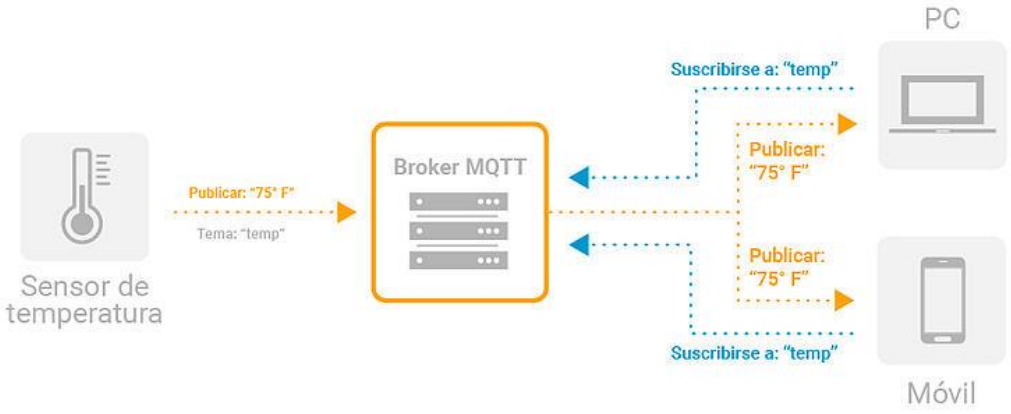
7.22.10 Reto A22.10. MQTT: Enviar datos a ThingSpeak.

Reto A22.10. MQTT: Enviar datos a ThingSpeak.

MQTT (*Message Queing Telemetry* Transport) es un protocolo de comunicación *Machine to Machine (M2M)* que permite la comunicación entre dispositivos de IoT. Utiliza una conexión que se mantiene abierta y se va reutilizando en cada comunicación. El funcionamiento del MQTT es un servicio de mensajería *push* con patrón publicador/suscriptor (*pub/sub*). Los clientes se conectan a un servidor central denominado *bróker*. Para filtrar los mensajes que se envían a cada cliente, estos mensajes se disponen en *topics* organizados jerárquicamente.

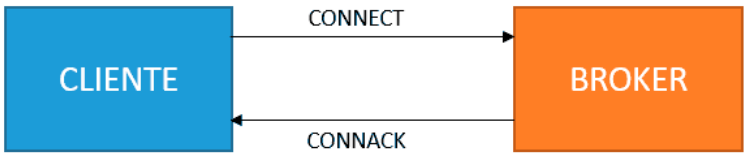


Este sistema, al mantener una conexión constante, hace posible que el servidor ya no espere la petición de los dispositivos ni tenga que preguntar constantemente si hay datos nuevos, ni realizar nuevas conexiones. Solamente quién necesita la información realiza la consulta y envía los mensajes.



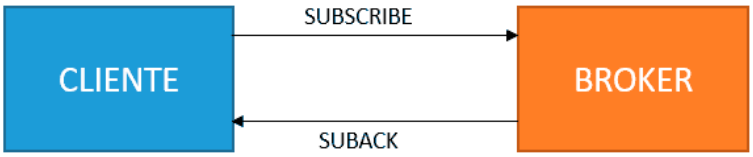
Los clientes inician una conexión TCP/IP con el *broker*, el cual mantiene un registro de los clientes conectados. Esta conexión se mantiene abierta hasta que el cliente la finaliza. Por defecto, MQTT emplea el puerto 1883 y el 8883 cuando funciona sobre TLS.

Para ello el cliente envía un mensaje CONNECT que contiene información necesaria (nombre de usuario, contraseña, client-id…). El *broker* responde con un mensaje CONNACK, que contiene el resultado de la conexión (aceptada, rechazada, etc.).



Para enviar los mensajes el cliente emplea mensajes PUBLISH, que contienen el *topic* y el *payload*.

Para suscribirse y desuscribirse se emplean mensajes SUBSCRIBE y UNSUSCRIBE, que el servidor responde con SUBACK y UNSUBACK.



Por otro lado, para asegurar que la conexión está activa los clientes mandan periódicamente un mensaje PINGREQ que es respondido por el servidor con un PINGRESP. Finalmente, el cliente se desconecta enviando un mensaje de DISCONNECT.

De igual forma si usamos un servidor/broker MQTT en internet debemos asegurarnos de conectar nuestro dispositivo a internet, mientras que de igual forma podríamos usar la opción de crear un punto de acceso propio siempre que uno de los dispositivos conectados a mi red tuviera el servidor MQTT activo.

Vamos a realizar la programación utilizando MQTT aplicado a ThingSpeak. Para poder visualizar los datos enviados desde la placa ***Imagina TDR STEAM*** utilizaremos el programa ThingSpeak y la aplicación ThingView (opcional).

Deberemos realizar los siguientes programas:

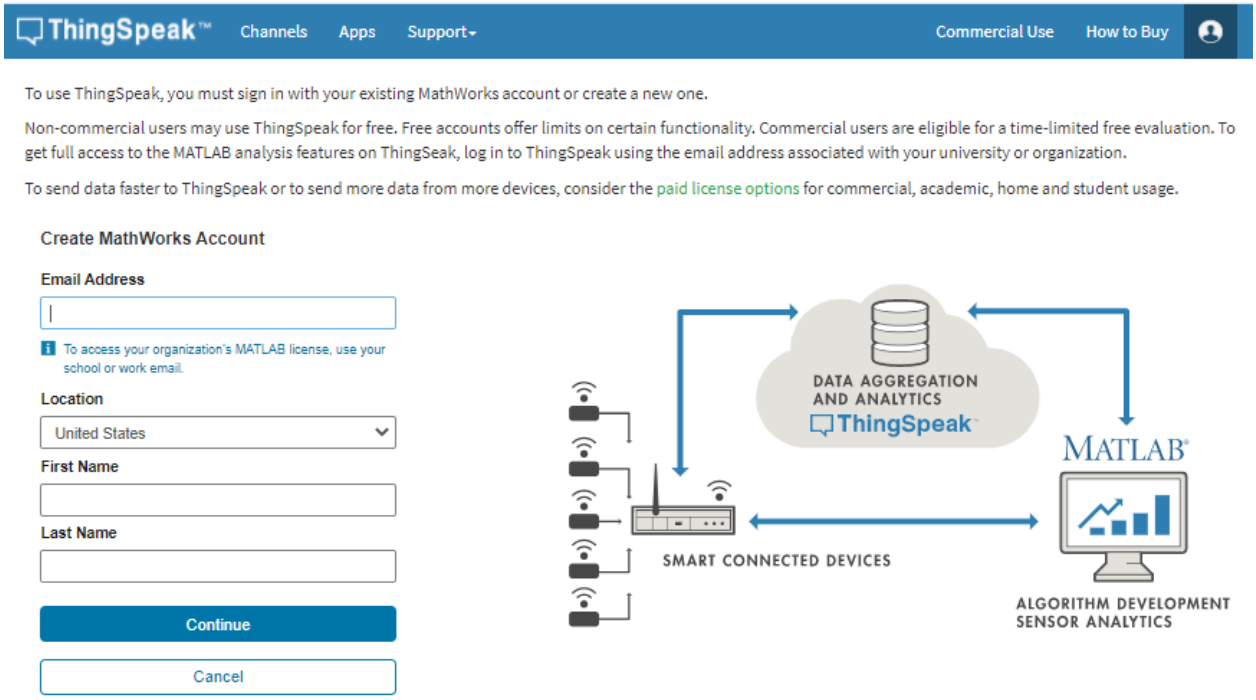
- ***ArduinoBlocks***: programa de recogida y envío de datos.
- ***ThingSpeak***: programa per ver los datos en el ordenador a través de Internet.
- ***ThingView***: aplicación para ver los datos en el teléfono móvil.

Para configurar los tres elementos hemos de seguir los pasos descritos a continuación, aunque solamente con ArduinoBlocks y ThingSpeak ya podemos hacer nuestra primera práctica de IoT.

7.22.10.1 Reto A22.10.1. ThingSpeak.

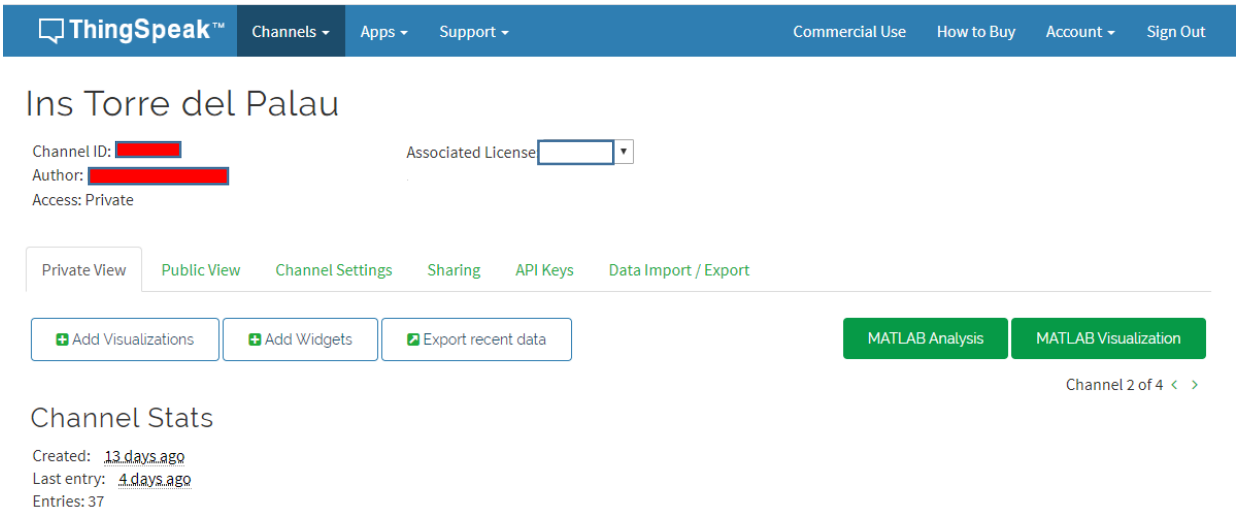
Reto A22.10.1. ThingSpeak.

Crear una cuenta en ThingSpeak: https://thingspeak.com/login



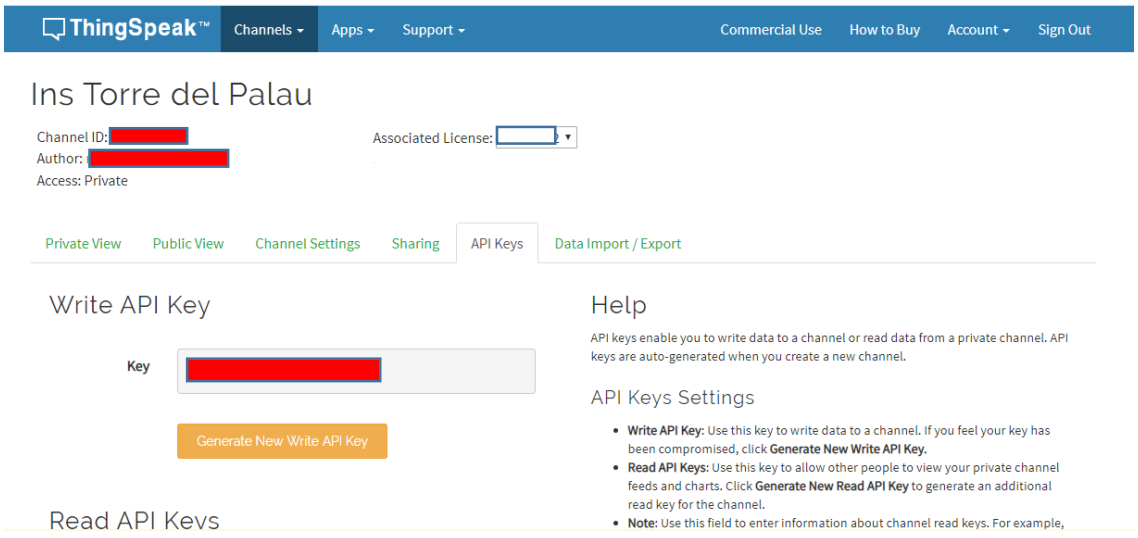
Una vez que hemos creado la cuenta ThingSpeak hemos de apuntar los siguientes datos:

- Channel ID: referencia de nuestro dispositivo.
- Author: referencia del autor del dispositivo.

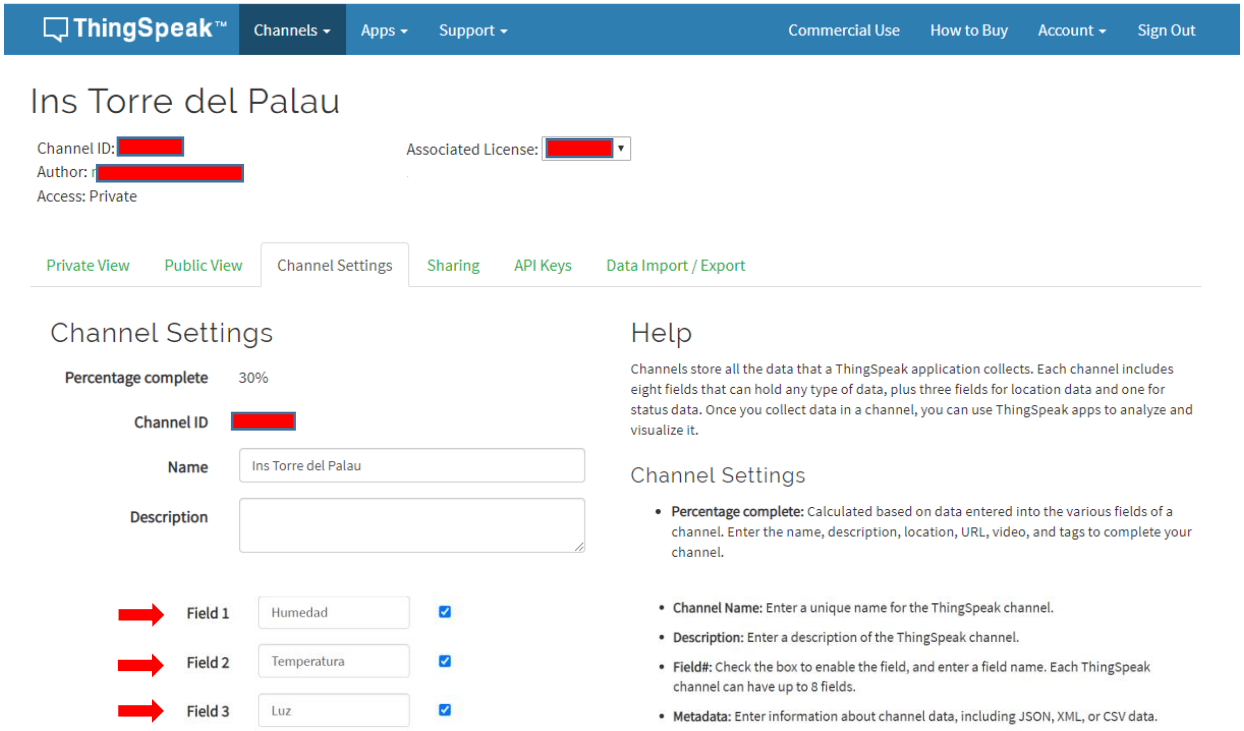


Otros datos importantes están en la pestaña "API Keys".

- Write API Key: código identificativo para enviar los datos.



Por último, hemos de hacer la configuración de los canales dónde recibiremos los datos en *Channel Settings*, en nuestro caso vamos a tener 3 canales/campos (Fields).



7.22.10.2 Reto A22.10.2. ArduinoBlocks.

Reto A22.10.2. ArduinoBlocks.

Una vez configurado ThingSpeak, prepararemos el programa de ArduinoBlocks para leer los datos de los sensores y enviarlos vía Wifi.

Enviaremos los valores de humedad y temperatura con el sensor DHT11 y el valor de la luz con el sensor LDR. Cada uno de estos valores estará asignado a un campo de los canales de Thingspeak (están en orden).

En el bloque "Conectar a una red WiFi:

- SSID: nombre de la red Wifi.
- Clave: contraseña de nuestra red Wifi.

En el bloque "Inicializar", configuraremos:

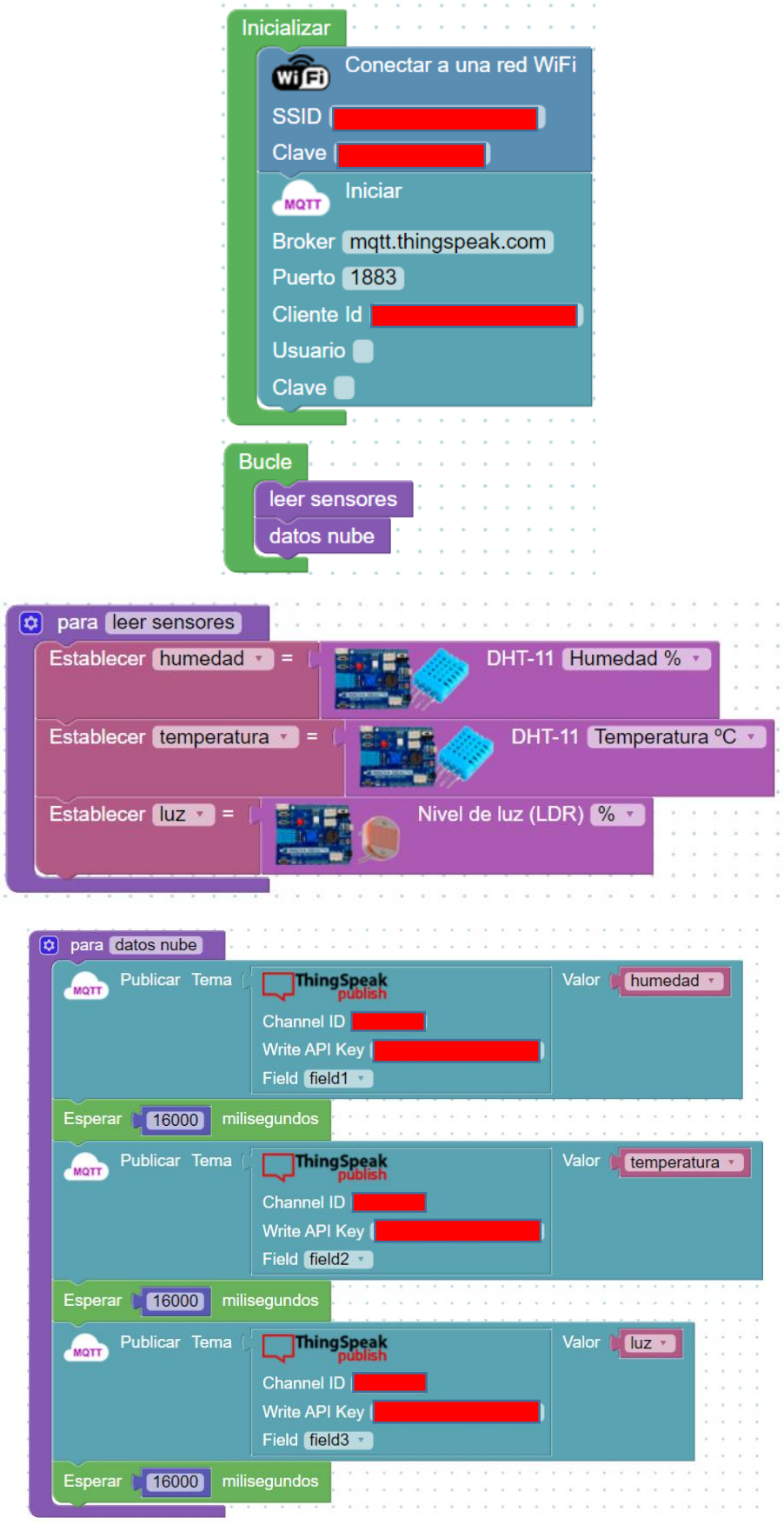
- Broker: mqtt.thingspeak.com.
- Puerto: por defecto (1883).
- Cliente Id: nombre de usuario que tenemos en ThingSpeak (mwa0xxx), aunque si no se pone también funciona.
- Usuario: no hace falta introducir datos.
- Clave: no hace falta introducir datos.

A continuación, configuraremos la publicación de los datos con la función *para: subir datos a la nube*:

- Channel ID: identificador de nuestro canal en ThingSpeak.
- Write API Key: código identificativo para enviar los datos a ThingSpeak.

Si utilizamos una cuenta gratuita de ThingSpeak el envio de cada dato no puede ser en un tiempo inferior a 16 segundos (16000 milisegundos).

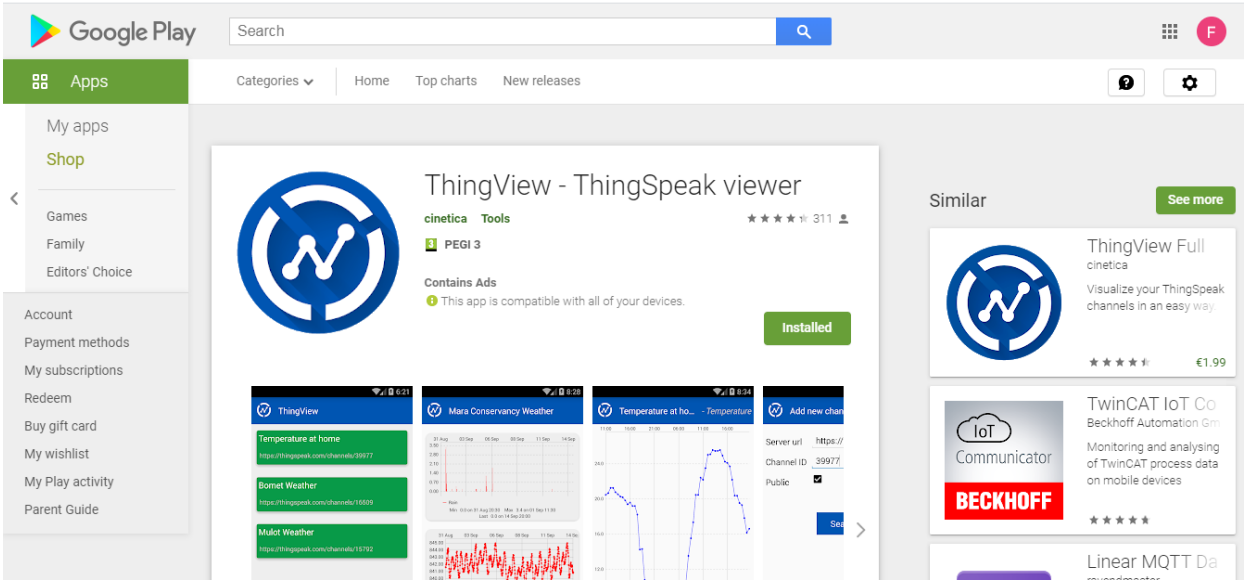
El programa creado es el siguiente:



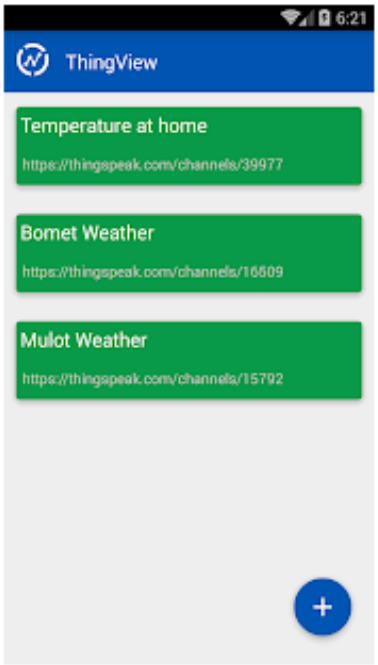
7.22.10.3 Reto A22.10.3. ThingView.

Reto A22.10.3. ThingView.

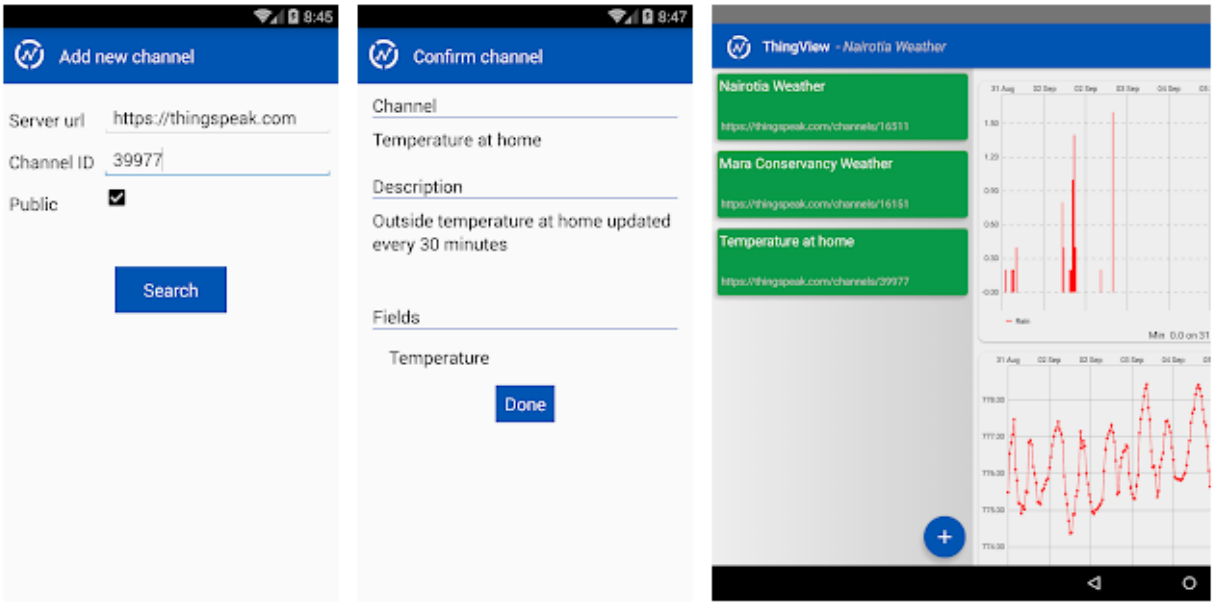
Si queremos ver los datos en el móvil, podemos instalar la aplicación ThingView. Para hacer la instalación hemos de seguir los siguientes pasos:



Add channel → *Channel ID*: ponemos el código de nuestro canal de ThingSpeak (Channel ID).



Visualización de datos en ThingView.



Actividad de ampliación: realiza los programas anteriores y comprueba su funcionamiento.

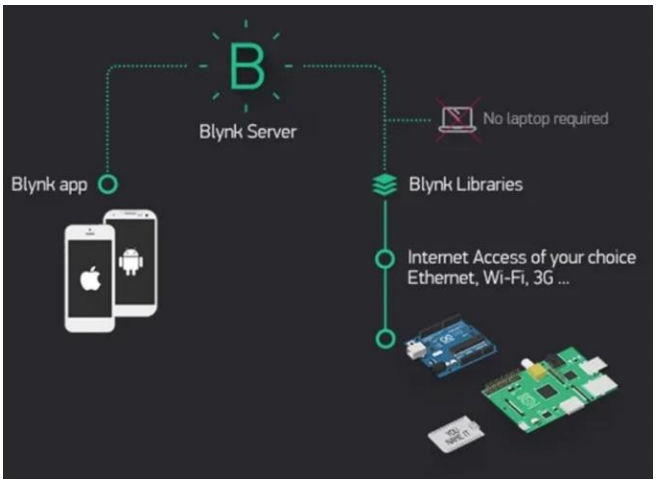
7.22.11 Reto A22.11. Blynk.

Reto A22.11. Blynk.

Blynk es una plataforma para IoT compatible con ESP32, Arduino, ESP8266 y muchísimos sistemas más. Blynk permite que cualquiera pueda controlar fácilmente su proyecto mediante con una Tablet o móvil con sistema iOS o Android.

Para utilizar el servicio Blynk tenemos dos opciones:

- *Usar el servidor Blynk de Internet* (con limitaciones si no pagamos): debemos registrarnos y crear una nueva cuenta desde la app. Esta es la opción más sencilla y rápida. Al usar el servidor de Blynk tendremos limitaciones (si no queremos pagar), pero puede ser la opción más fácil para iniciarse.
- *Montar un servidor propio de Blynk*: Esta opción es un poco más complicada pues supone ejecutar el servidor en un PC en la red y gestionar los usuarios nosotros mismos. Es una opción muy potente pero más compleja.



Si utilizamos el servicio online limitado de Blynk debemos asegurarnos que la conexión a la red Wifi de nuestro dispositivo ***ESP32 Plus STEAMakers*** nos proporciona conexión a internet (por tanto, se desaconseja la opción de crear un punto de acceso). Si utilizamos la opción de punto de acceso para que otros dispositivos se conecten a nuestro dispositivo, el servidor Blynk debe estar instalado en unos de los dispositivos que se conectan a mi red.

El sistema Blynk consta de tres partes:

- Aplicación móvil (app) para Android o iOS.
- Servidor Blynk (instalable en Windows, Linux, MacOs, etc.) o utilizando el servidor propio en internet (con limitaciones si no pagas).

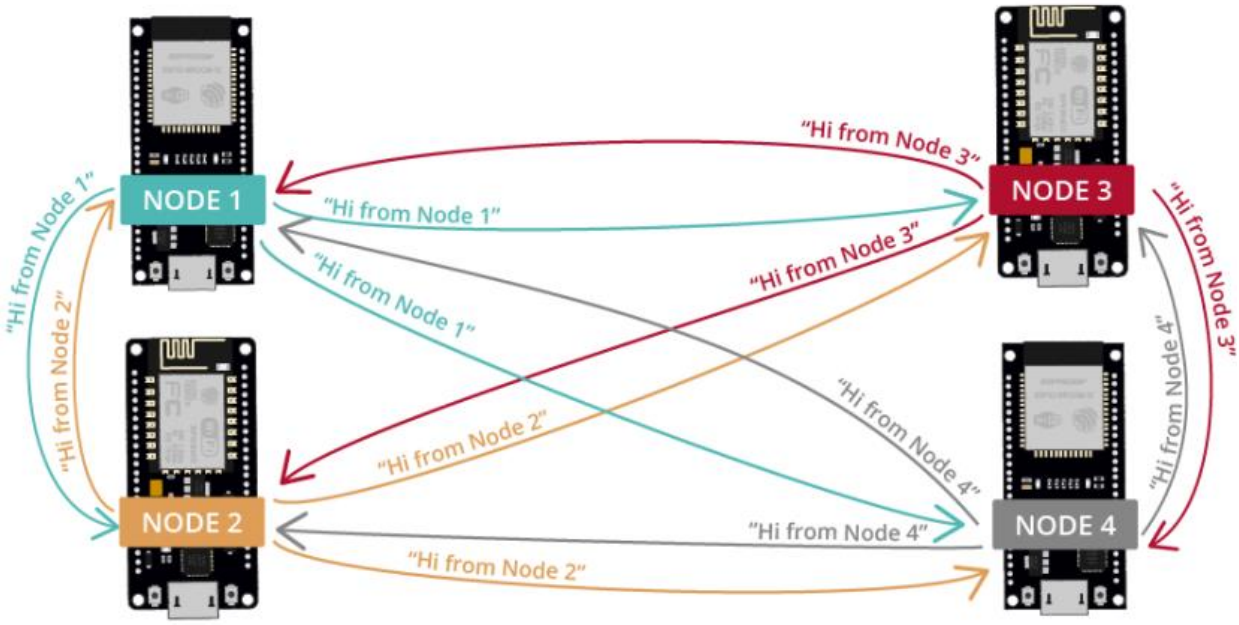
- Firmware en el dispositivo ESP32, Arduino, ESP8266, etc. (con ArduinoBlocks).

En este manual de actividades no vamos a trabajar con Blynk, pero están los bloques en ArduinoBlocks para poder hacerlo.

7.22.12 Reto A22.12. Wifi-Mesh.

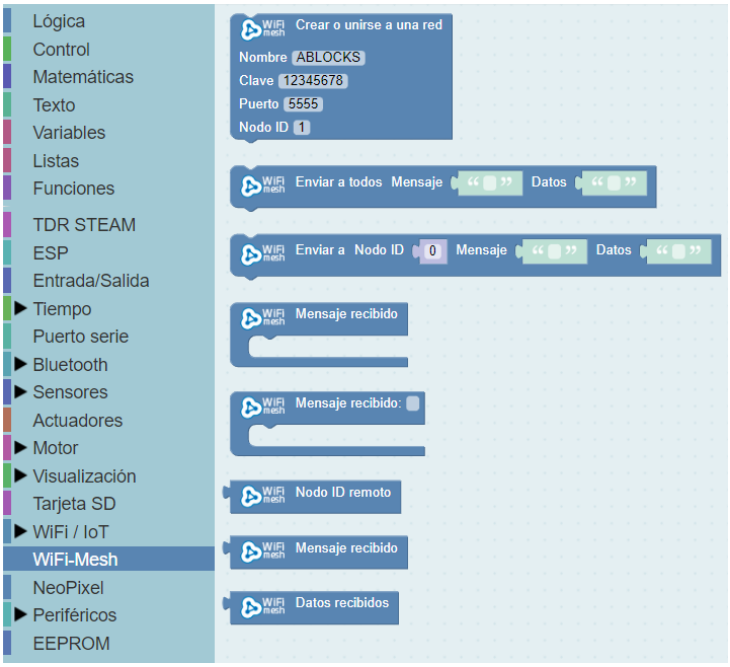
Reto A22.12. Wifi-Mesh.

Wifi-Mesh es una malla de dispositivos implementando una red wifi descentralizada, de forma que todas las placas ***ESP32 Plus STEAMakers*** estarán interconectadas de forma sencilla y sin necesidad de una red Wifi previa o punto de acceso (la crean ellas mismas automáticamente). Cada dispositivo puede enviar mensajes al resto y recibir de cualquiera, se produce un intercambio de información bidireccional entre ellas.

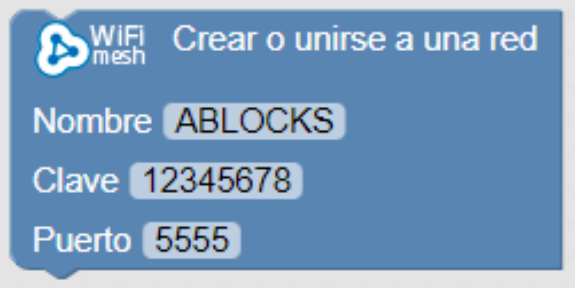


Una red Wifi de tipo *mesh* o de malla es una red compuesta una única red Wifi con el mismo SSID y contraseña.

En ArduinoBlocks el proceso de intercambio de mensajes entre placas ***ESP32 Plus STEAMakers*** es muy fácil de utilizar. Todos los dispositivos (nodos) que vayan a formar parte de la red Mesh (malla) deben compartir la misma configuración de nombre y clave de la red. Debemos ***esperar unos 30 segundos*** para asegurarnos de que se establezca la ***comunicación entre las placas***. Disponemos de unos bloques específicos para poder trabajar con este sistema.



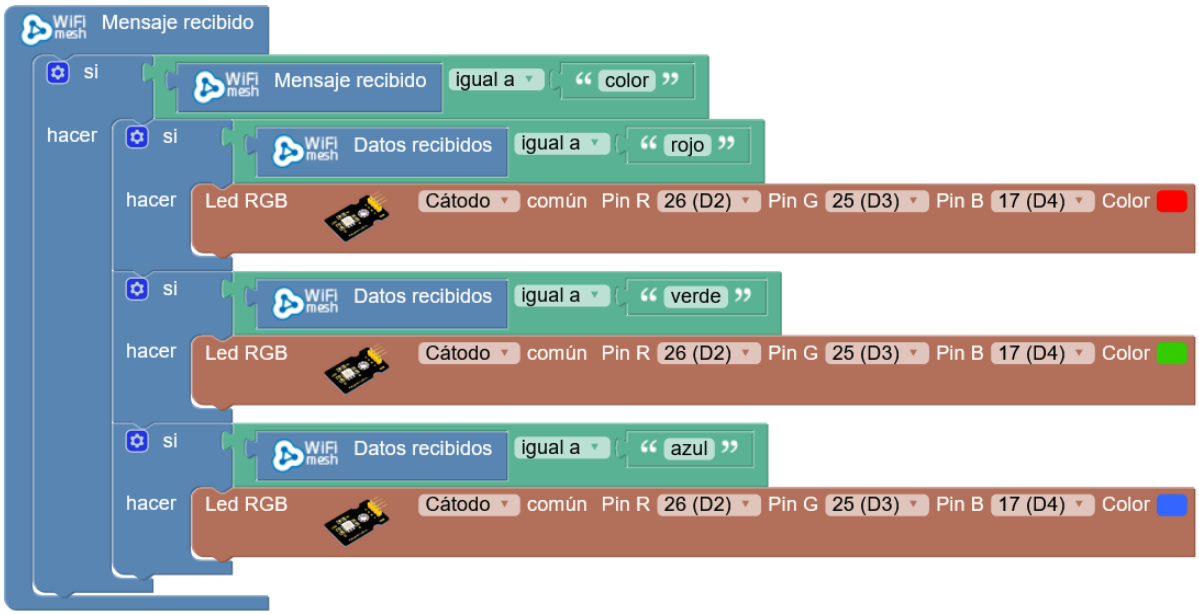
Todos los dispositivos (nodos) que vayan a formar parte de la red Mesh (malla) deben compartir la misma configuración de nombre, clave de la red y puerto.



Enviar un mensaje a todos los nodos de la red: envía un mensaje y datos (opcionales) a todos los nodos conectados a nuestra red *mesh* (malla).

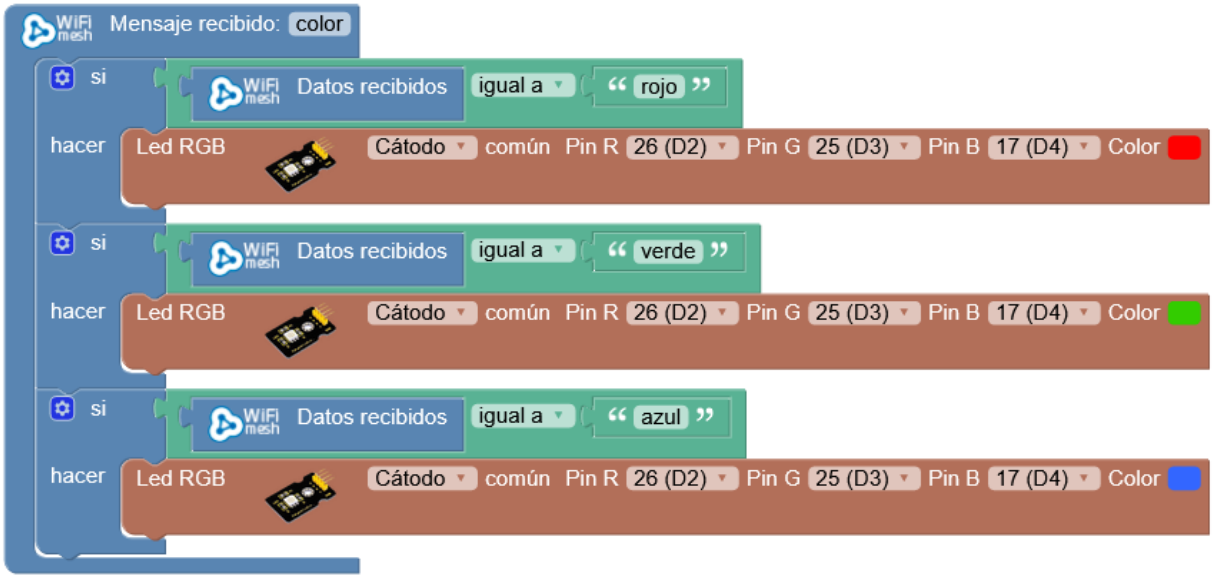
El bloque *Mensaje recibido* recibe todos los mensajes y dentro podemos determinar el mensaje y los datos recibidos:

Y si añadimos datos extras al mensaje, podemos recibirlos de igual forma:



Para facilitar, podemos usar el bloque de recibir mensajes en concreto:

O siguiendo con el ejemplo anterior:



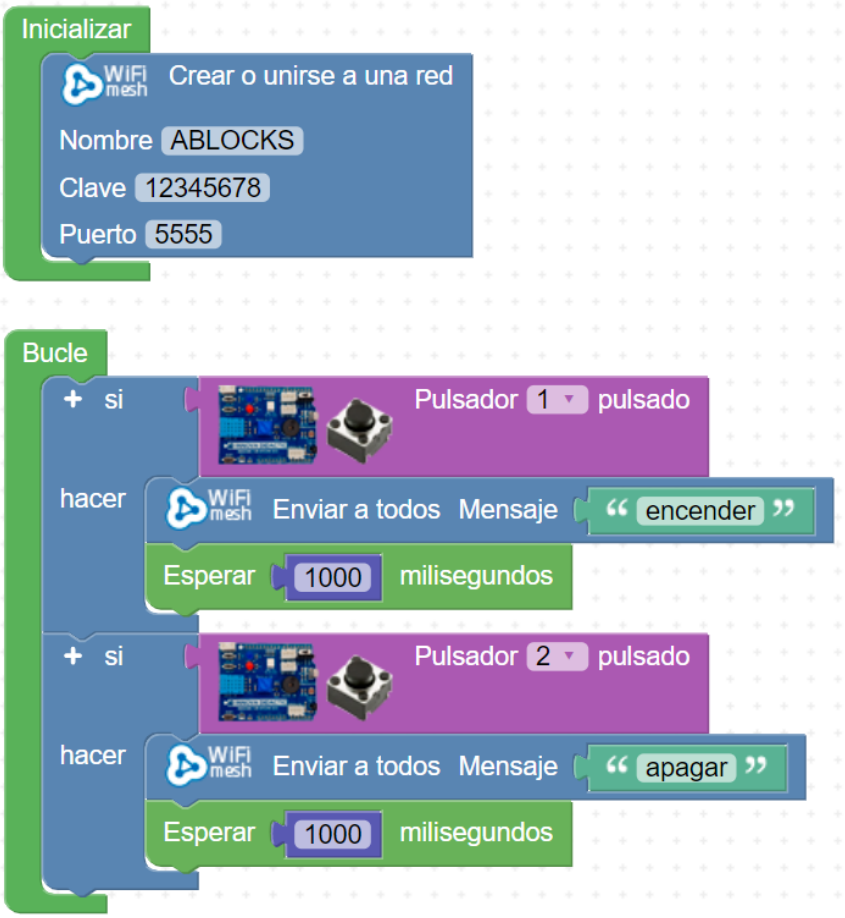
7.22.12.1 Reto A22.12.1 Wifi-Mesh I.

Reto A22.12.1. Wifi-Mesh I.

Vamos a realizar la comunicación entre dos placas ***ESP32 Plus STEAMakers*** mediante Wifi-Mesh. Para realizar dicha comunicación, vamos a realizar dos programas, uno para cada placa. Vamos a encender y apagar el led rojo de la placa MESH_2 mediante los pulsadores SW1 y SW2 de la placa MESH_1.

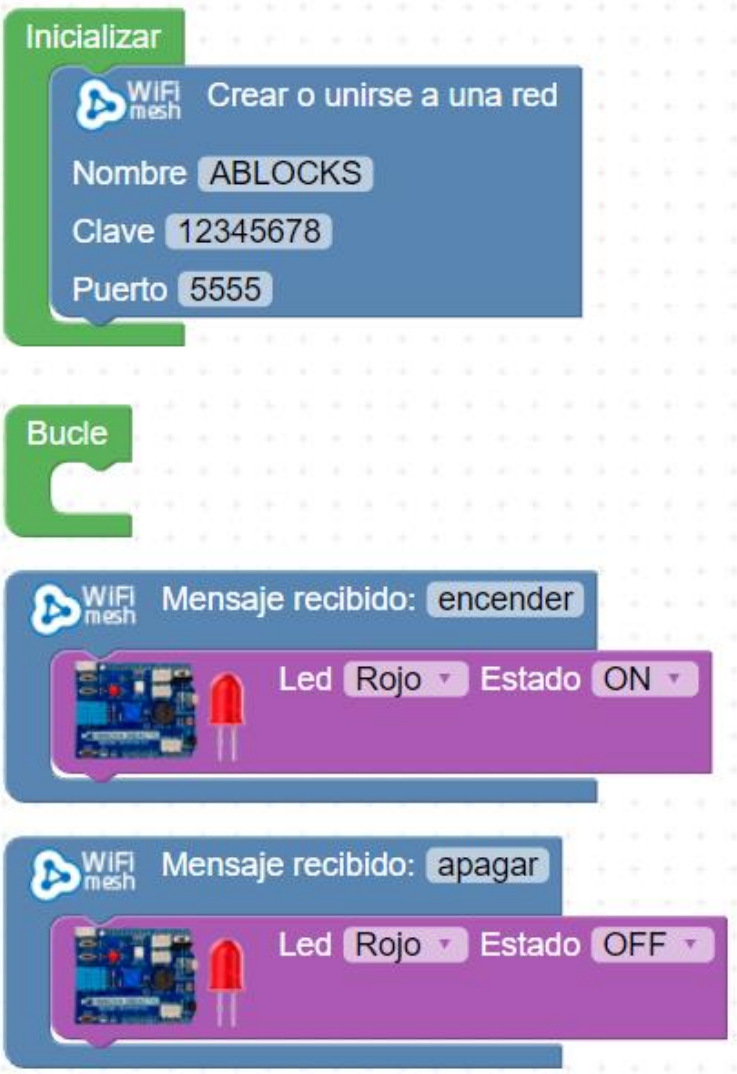
Una vez cargados los programas en las placas deberemos esperar ***unos 30 segundos*** para que se inicie la comunicación Wifi entre las placas. La placa MESH-1 será la que envíe los datos y la placa MESH-2 será la que reciba los datos.

El programa de MESH_1 enviará *encender* o *apagar* si apretamos SW1 o SW2 respectivamente. El código del programa de MESH_1 será el siguiente:



Enlace al programa: ArduinoBlocks Projects\wifi_mesh_1a.abp

El programa de MESH_2 recibirá *encender* o *apagar*. Si recibe encender encenderá el led rojo y si recibe apagar apagará el led rojo. El código del programa de MESH_2 será el siguiente:



Enlace al programa: ArduinoBlocks Projects\wifi_mesh_1b.abp

Actividad de ampliación: realiza el programa anterior y comprueba su funcionamiento.

7.22.12.2 Reto A22.12.2. Wifi-Mesh II.

Reto A22.12.2. Wifi-Mesh II.

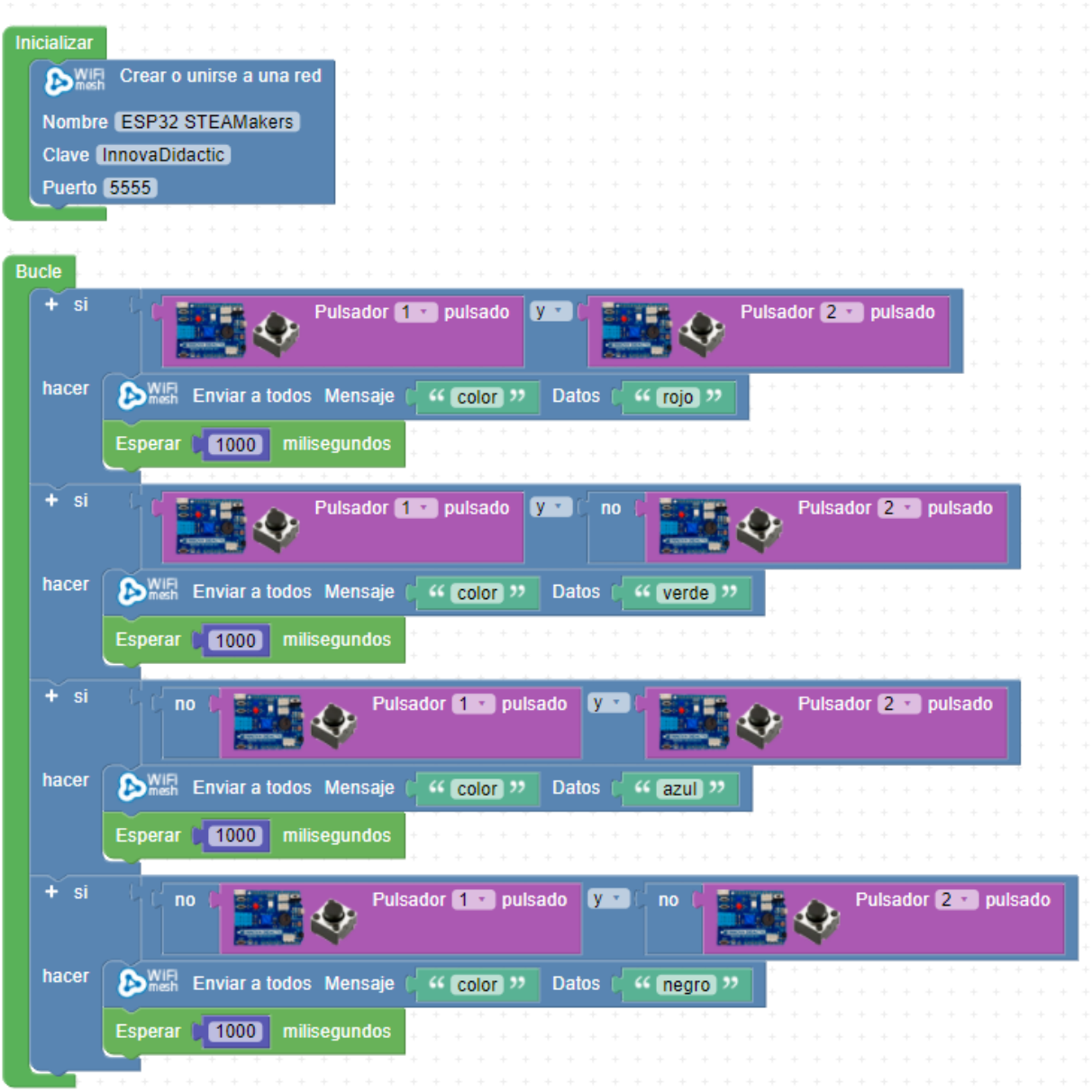
En esta actividad vamos a cambiar el *Nombre* y la *Clave* de nuestra nueva red. Vamos a programar nuestras placas para controlar un led RGB en la placa MESH_2 mediante los pulsadores que tenemos en la placa MESH_1.

La configuración será:

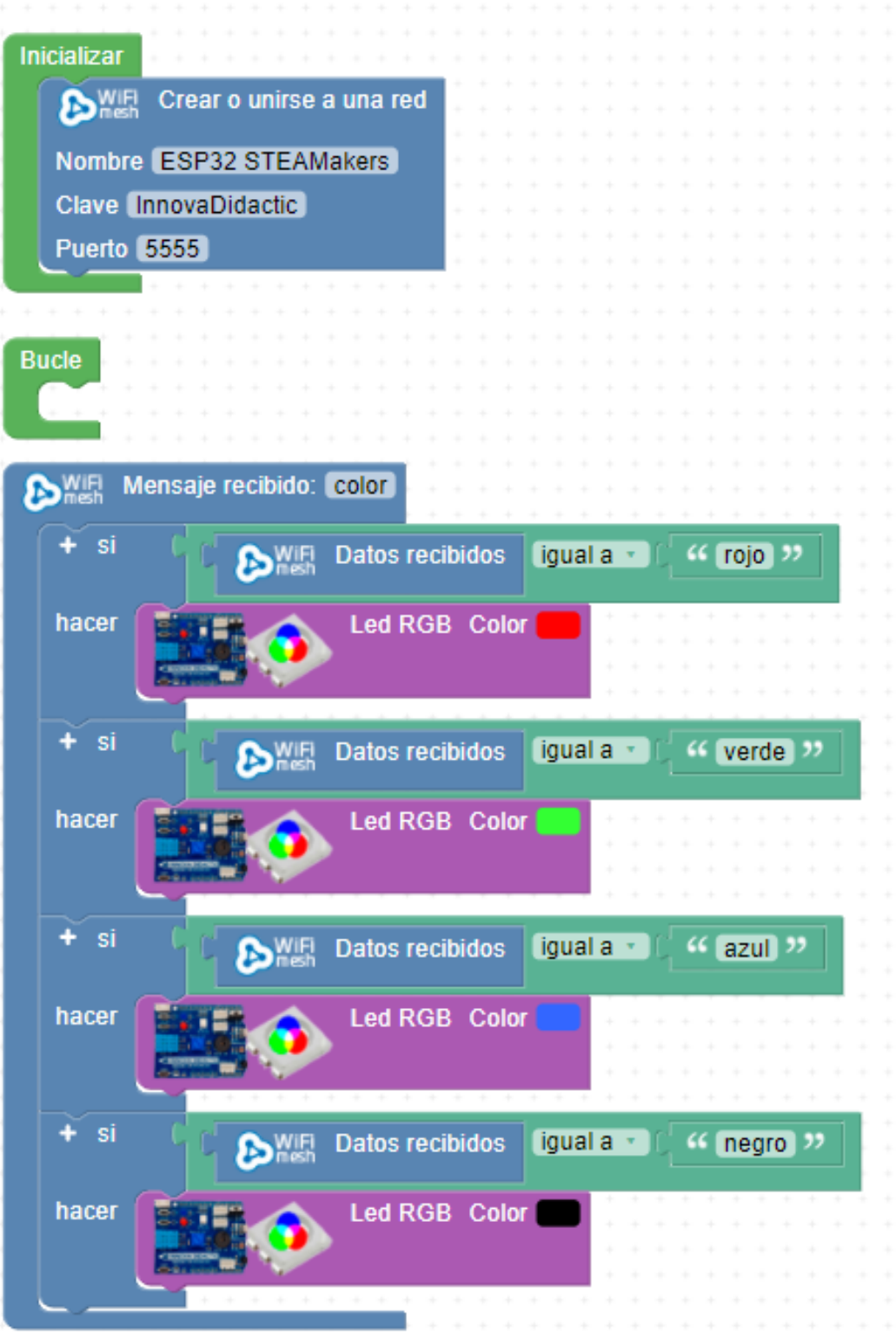
- SW1 y SW2 (los dos pulsadores a la vez): encenderá el led RGB de color rojo.
- SW1 y NO SW2 (pulsamos sólo SW1): encenderá el led RGB de color verde.
- NO SW1 y SW2 (pulsamos sólo SW2): encenderá el led RGB de color azul.
- NO SW1 y NO SW2 (no pulsamos ningún pulsador): encenderá el led RGB de color negro.

La placa MESH-1 será el nodo 1 y la placa MESH-2 será el nodo 2.

El programa de MESH-1 será:



El programa de MESH-2 será:



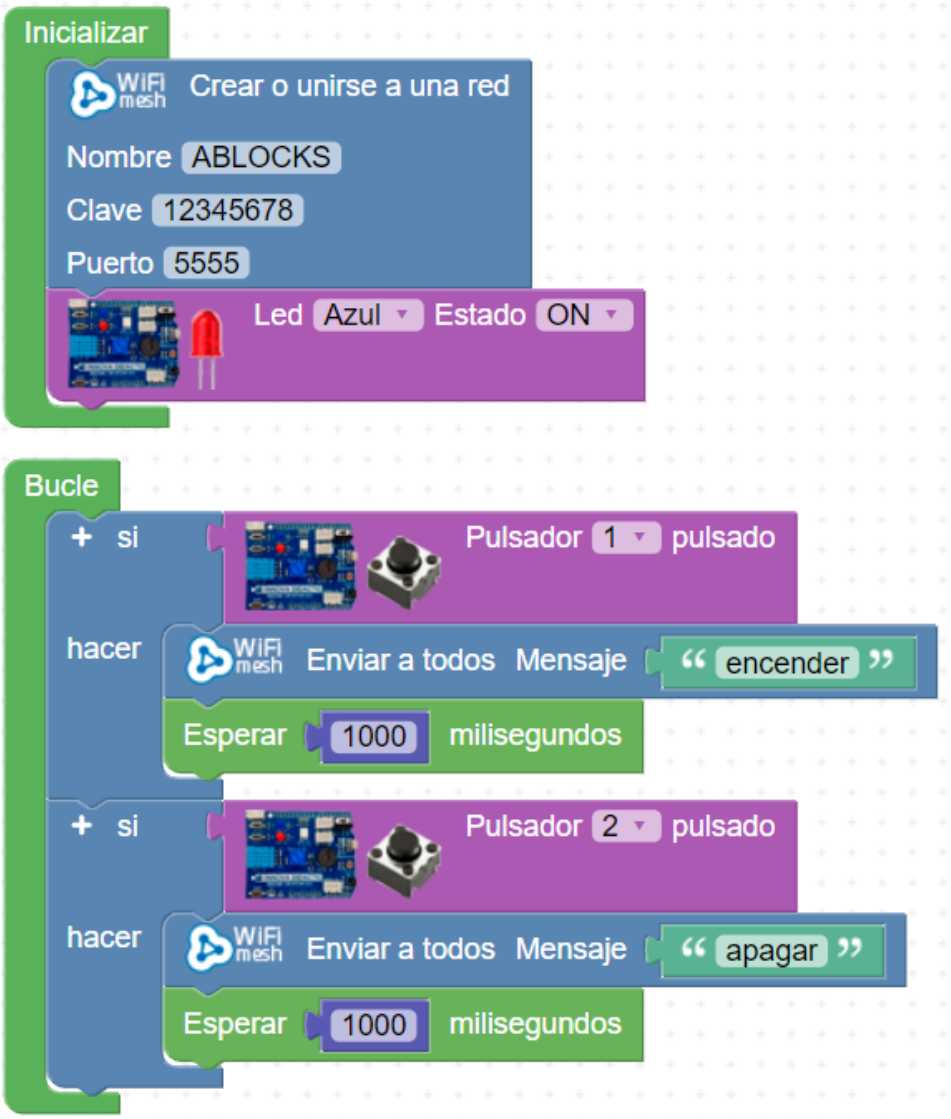
Actividad de ampliación: realiza el programa anterior y comprueba su funcionamiento.

7.22.12.3 Reto A22.12.3. Wifi-Mesh III.

Reto A22.12.3. Wifi-Mesh III.

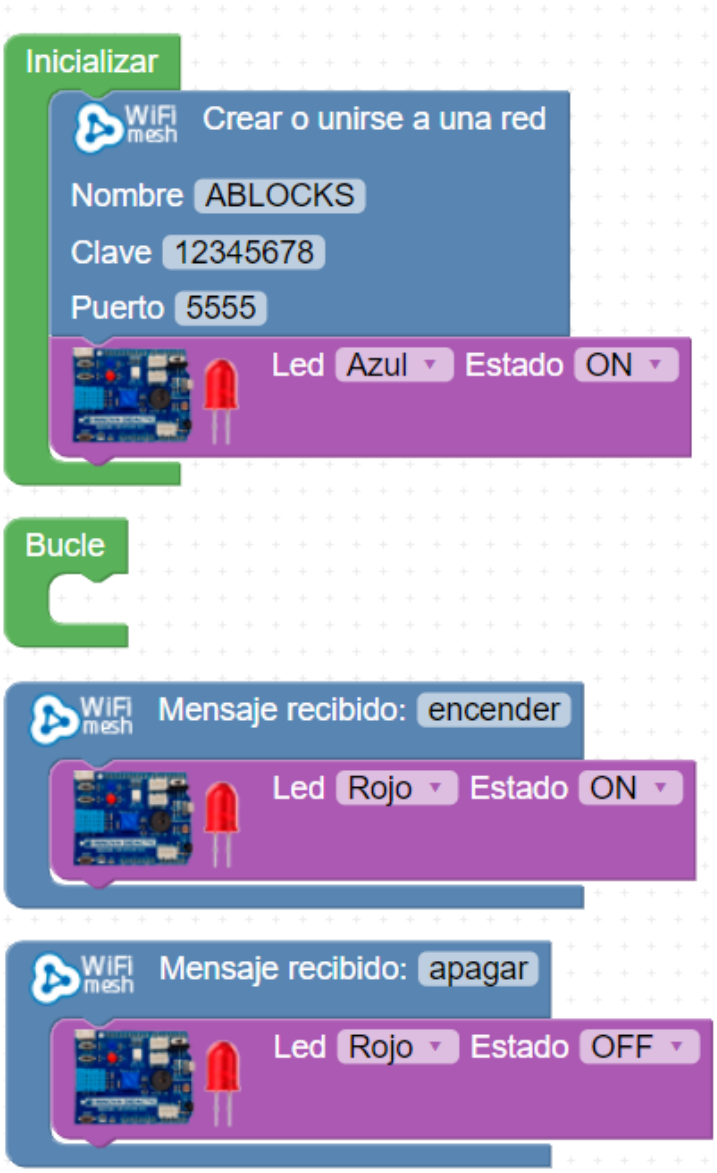
En esta actividad interconectaremos mediante Wifi 4 placas ***ESP32 Plus STEAMakers***. Probaremos como la placa 1 envía el mensaje de encender o apagar el led rojo al resto de placas (2-3-4). Dejaremos encendido el led azul para indicar que han empezado a funcionar.

Programa de la placa 1:



Enlace al programa:

Programa de las placas 2-3-4:



Enlace al programa:

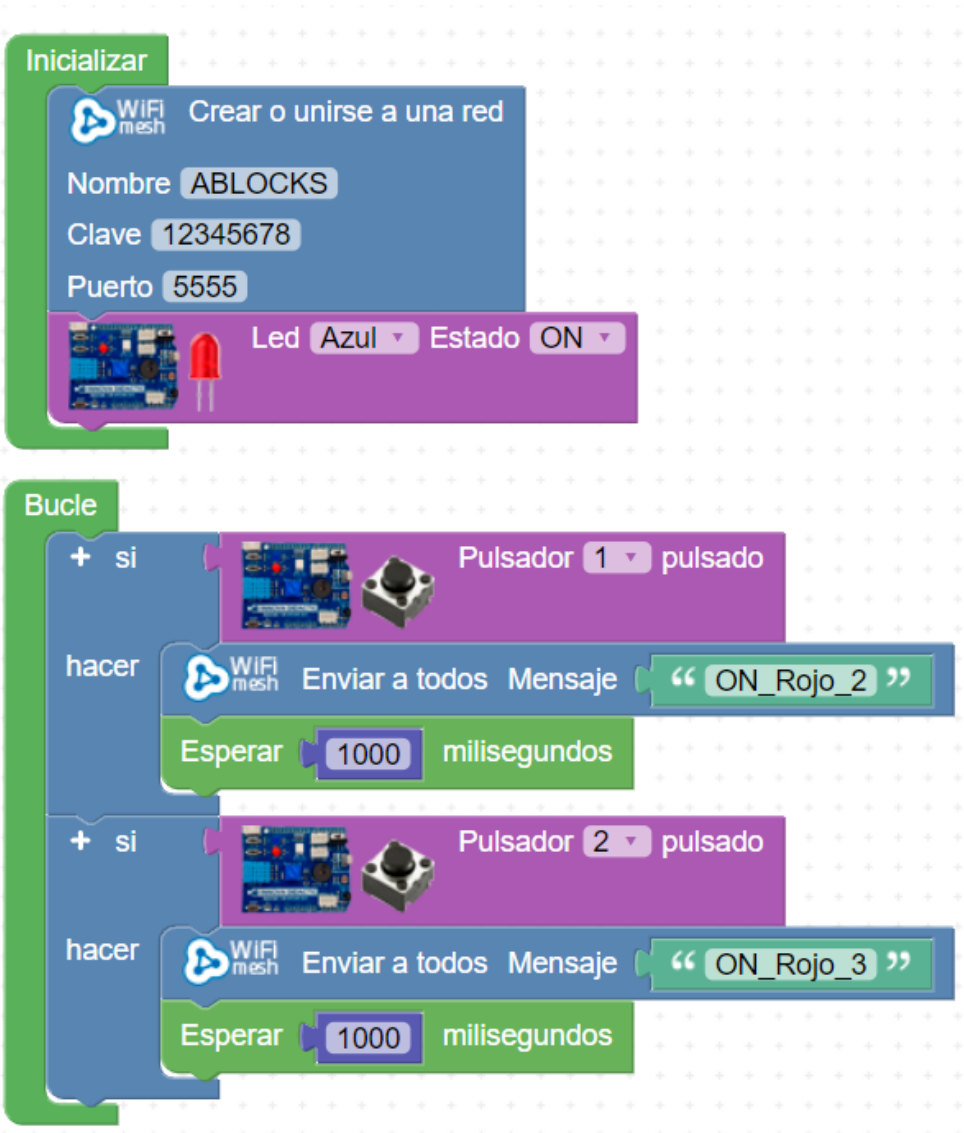
Actividad de ampliación: realiza los programas anteriores y comprueba su funcionamiento.

7.22.12.4 Reto A22.12.4. Wifi-Mesh IV.

Reto A22.12.4. Wifi-Mesh IV.

En esta actividad vamos a crear una malla wifi con 3 placas. La placa 1 será la master y las placas 2 y 3 serán sus esclavas. Mediante los pulsadores de la placa 1 controlaremos el encendido del led rojo de las otras dos placas. Los leds harán una intermitencia 5 veces y quedarán apagados.

El programa de la placa 1 será:

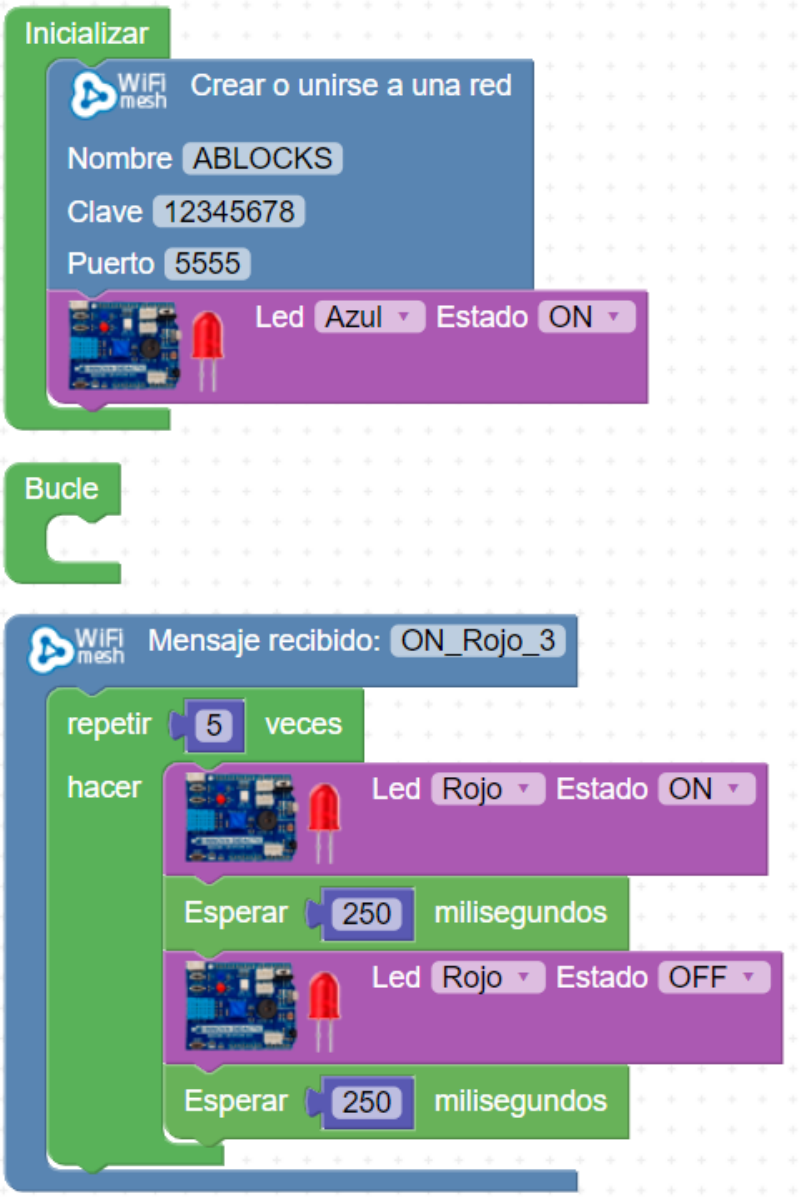


Enlace al programa: ArduinoBlocks Projects\wifi_mesh_4a.abp

El programa de la placa 2 será:

Enlace al programa: ArduinoBlocks Projects\wifi_mesh_4b.abp

Y el de la placa 3:

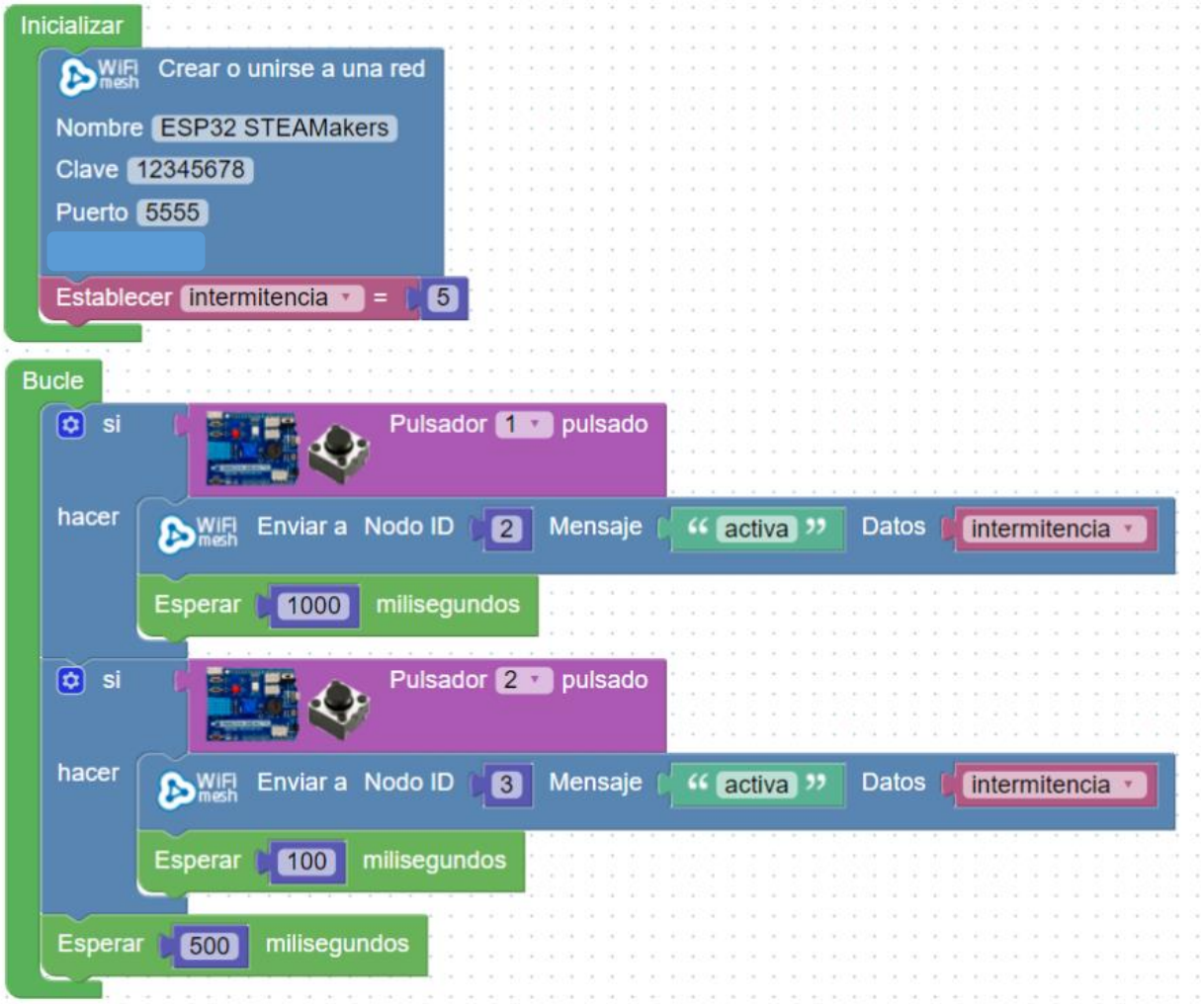


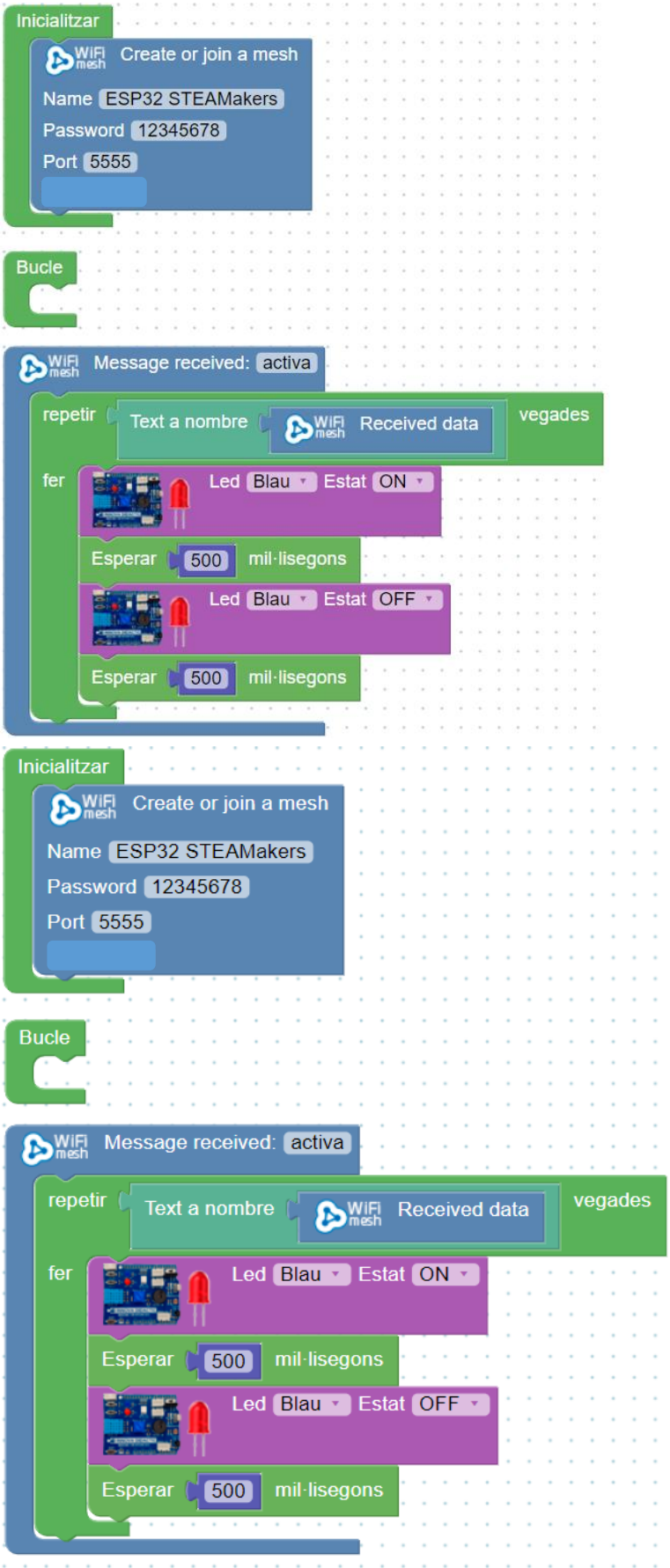
Enlace al programa: ArduinoBlocks Projects\wifi_mesh_4c.abp

7.22.12.5 Reto A22.12.5. Wifi-Mesh V.

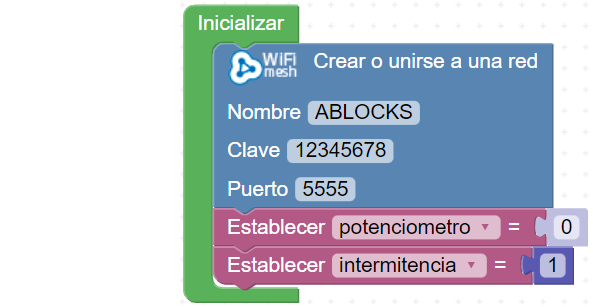
Reto A22.12.5. Wifi-Mesh V.

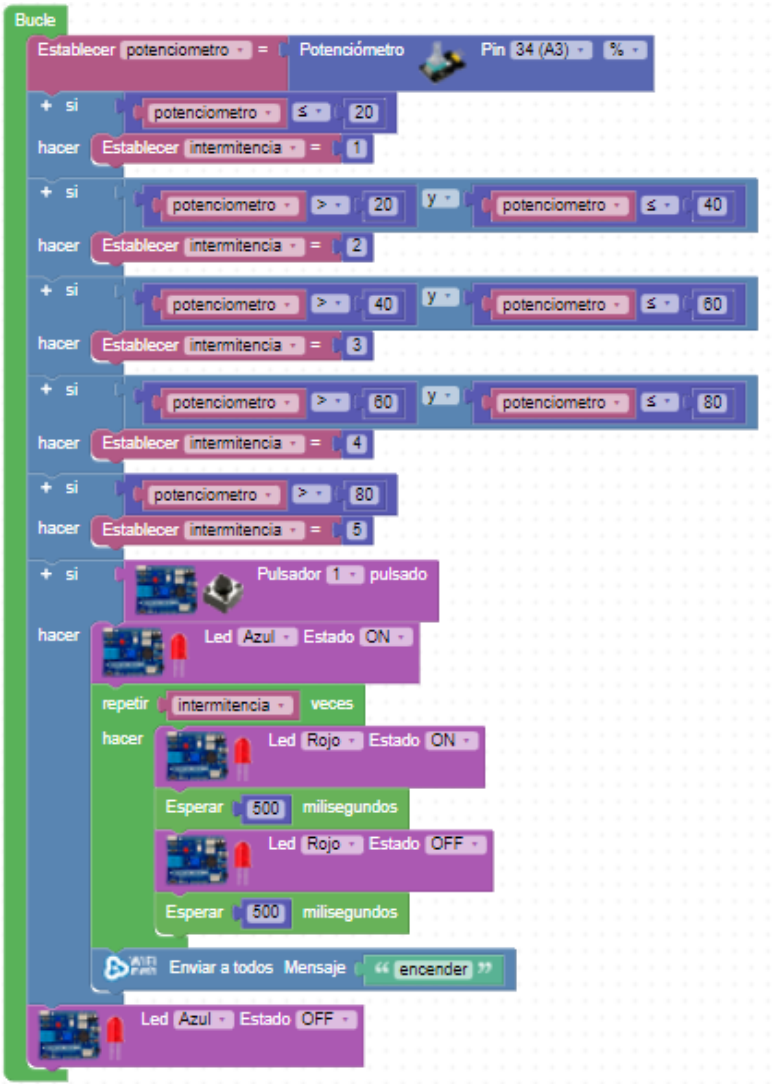
En esta actividad vamos a interconectar 2 placas. La placa 1 enviará una orden al pulsar el pulsador 1 a la placa 2 para que una cantidad de intermitencias (por ej. 5). Con el pulsador 2 de la placa 1 enviaremos la cantidad de intermitencias a la placa 3 (por ej. 5).




Inicialitzar
WiFi mesh Create or join a mesh
Name ESP32 STEAMakers
Password 12345678
Port 5555
Bucle
WiFi mesh Message received: activa
repetir Text a nombre WiFi mesh Received data vegades
fer
Led Blau Estat ON
Esperar 500 mil·lisegons
Led Blau Estat OFF
Esperar 500 mil·lisegons
Inicialitzar
WiFi mesh Create or join a mesh
Name ESP32 STEAMakers
Password 12345678
Port 5555
Bucle
WiFi mesh Message received: activa
repetir Text a nombre WiFi mesh Received data vegades
fer
Led Blau Estat ON
Esperar 500 mil·lisegons
Led Blau Estat OFF
Esperar 500 mil·lisegons

También podemos modificar el programa de la placa 1 para poder regular la intermitencia. Para realizarlo, conectaremos un potenciómetro externo en A3 (el de la placa ***Imagina TDR STEAM*** no funciona cuando utilizamos la Wifi). El programa de la placa 1 quedaría así:





Enlace al programa: ArduinoBlocks Projects\wifi_mesh_5c.abp

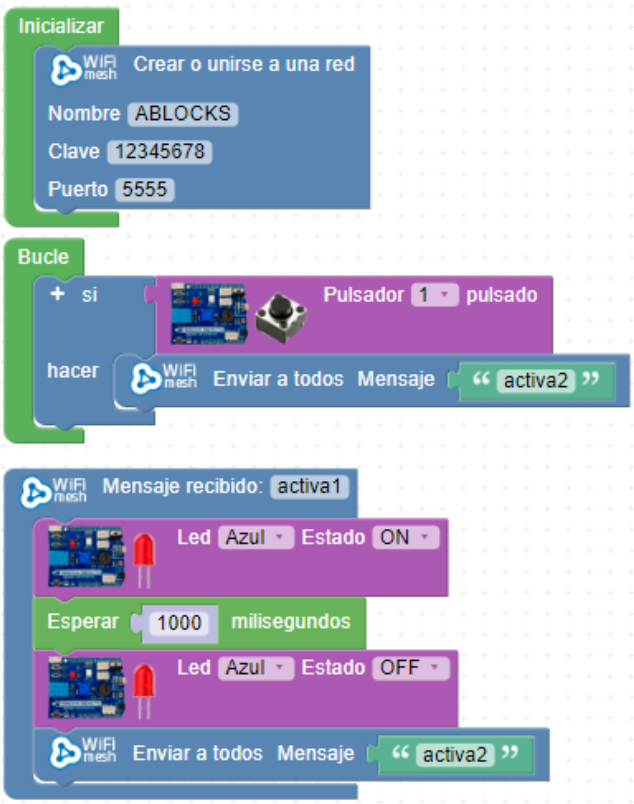
Actividad de ampliación: realiza el programa anterior y comprueba su funcionamiento.

7.22.12.6 Reto A22.12.6. Wifi-Mesh VI.

Reto A22.12.6. Wifi-Mesh VI.

En esta actividad, vamos a crear una comunicación cíclica entre tres placas. La primera placa activará la segunda, la segunda a la tercera y la tercera a la primera. Programaremos un pulsador en la placa 1 para que inicie la secuencia.

Programa de la placa 1:

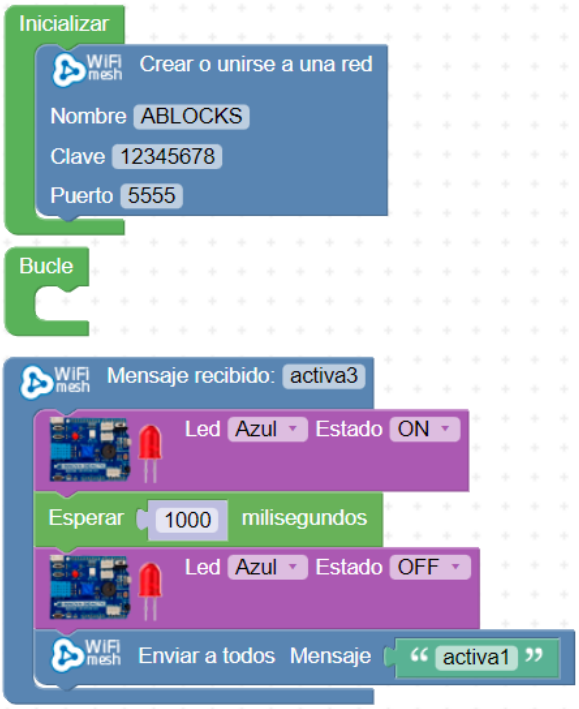


Enlace al programa: ArduinoBlocks Projects\wifi_mesh_6a.abp

Programa de la placa 2:

Enlace al programa: ArduinoBlocks Projects\wifi_mesh_6b.abp

Programa de la placa 3:



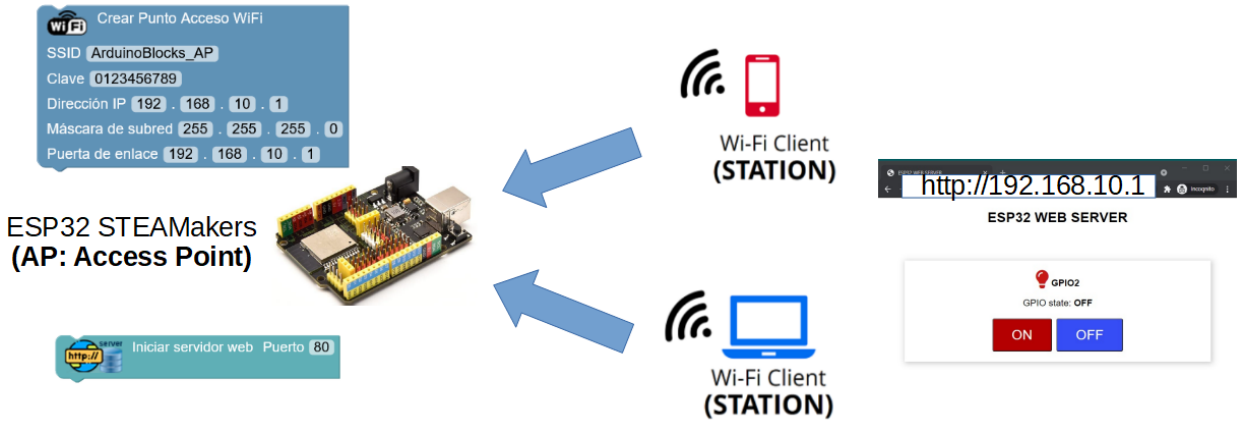
Enlace al programa: ArduinoBlocks Projects\wifi_mesh_6c.abp

7.22.13 Reto A22.13. Creación de un punto de acceso.

Reto A22.13. Creación de un punto de acceso.

Vamos a realizar un programa para poder crear un punto de acceso. Realmente lo que estamos creando es una red Wifi para poder conectar a ella diferentes dispositivos. Para realizarlo no necesitamos ninguna conexión a Internet. Lo que se creará es una red Wifi local a la que se pueden conectar diferentes dispositivos Wifi para compartir información.

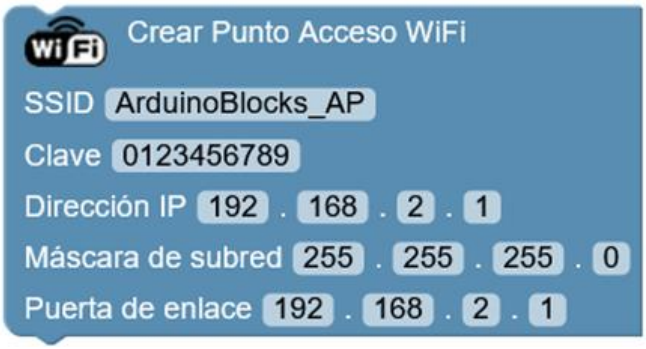
Crear un punto de acceso puede ser útil, por ejemplo, si queremos montar una estación meteorológica en un lugar apartado donde no hay conectividad Wifi. Podemos conectar todos los sensores de la estación a la placa ***ESP32 Plus STEAMakers*** y programarla como punto de acceso Wifi, de forma que cuando estemos cerca con nuestro móvil o Tablet nos conectemos a esa red y podemos acceder a un servidor web del propio dispositivo, donde nos dé la información de los sensores. Otro ejemplo podría ser la realización de un coche con la placa ***ESP32 Plus STEAMakers*** al que nos conectemos a su red Wifi propia y una vez conectados con una aplicación desde el móvil podamos controlar remotamente el coche.



Para crear una red Wifi usaremos el siguiente bloque donde configuraremos:

- SSID: nombre de la nueva red que vamos a crear.
- Clave: clave para conectarse a la nueva red.
- Dirección IP: dirección de este nuevo punto de acceso (del propio dispositivo).
- Máscara de red: por defecto 255.255.255.0

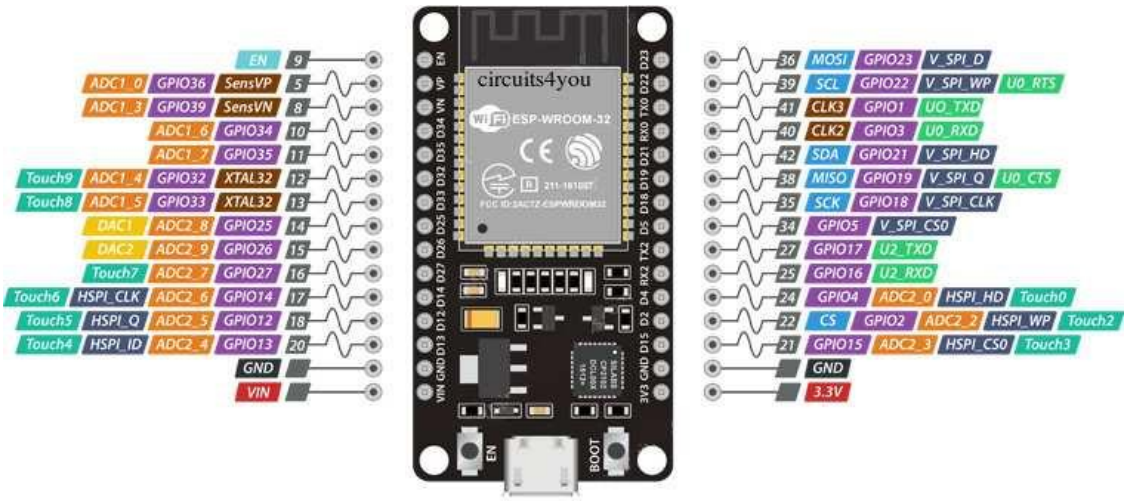
- Puerta de enlace: si tenemos conexión con otro dispositivo en la red que nos dé acceso a Internet o a otra red podemos poner la IP como puerta de enlace, si no dejamos la misma del dispositivo.

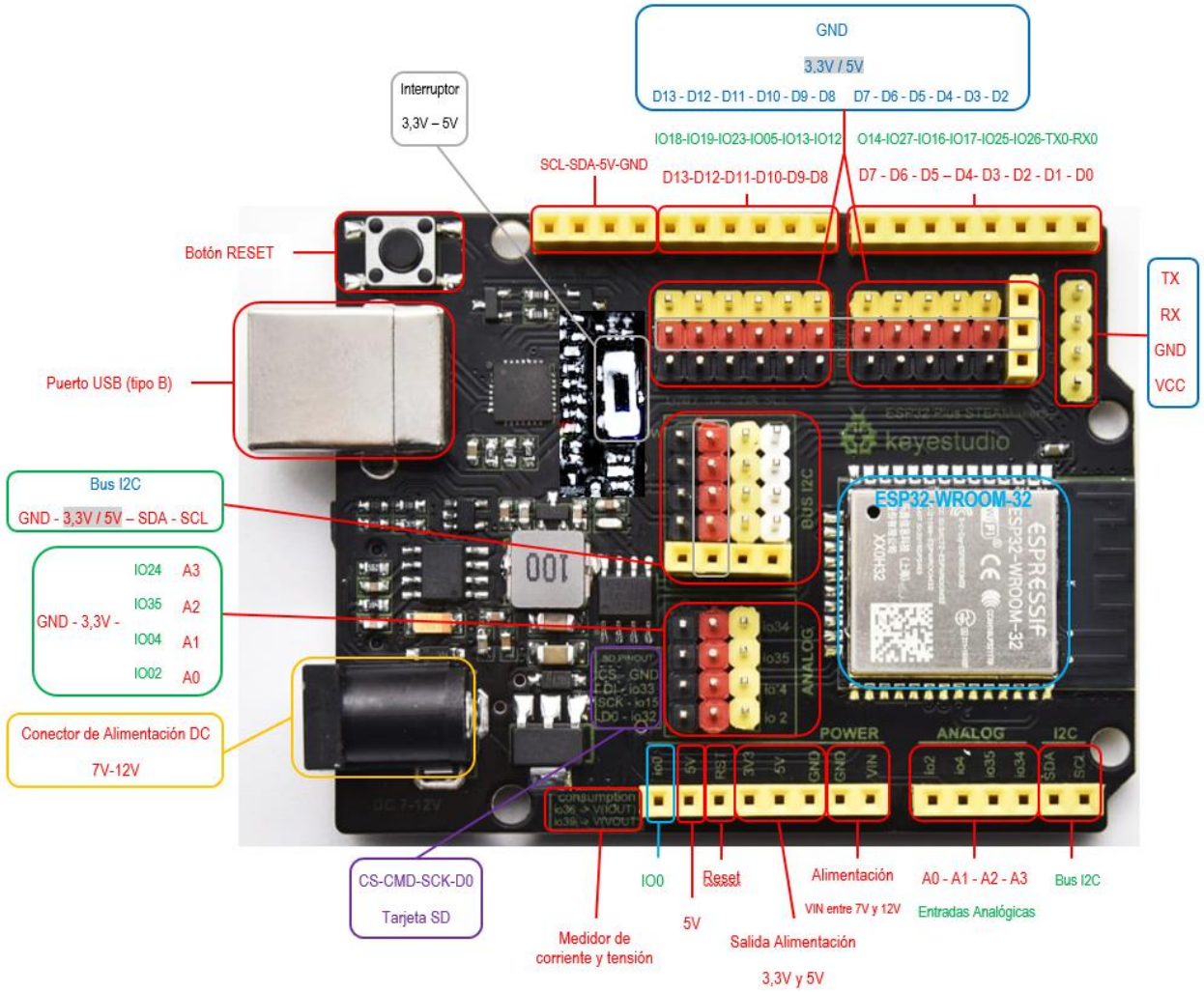


Podemos a realizar un programa para crear un punto de acceso Wifi y que podamos conectar varias placas ***ESP32 Plus STEAMakers*** a esa red Wifi

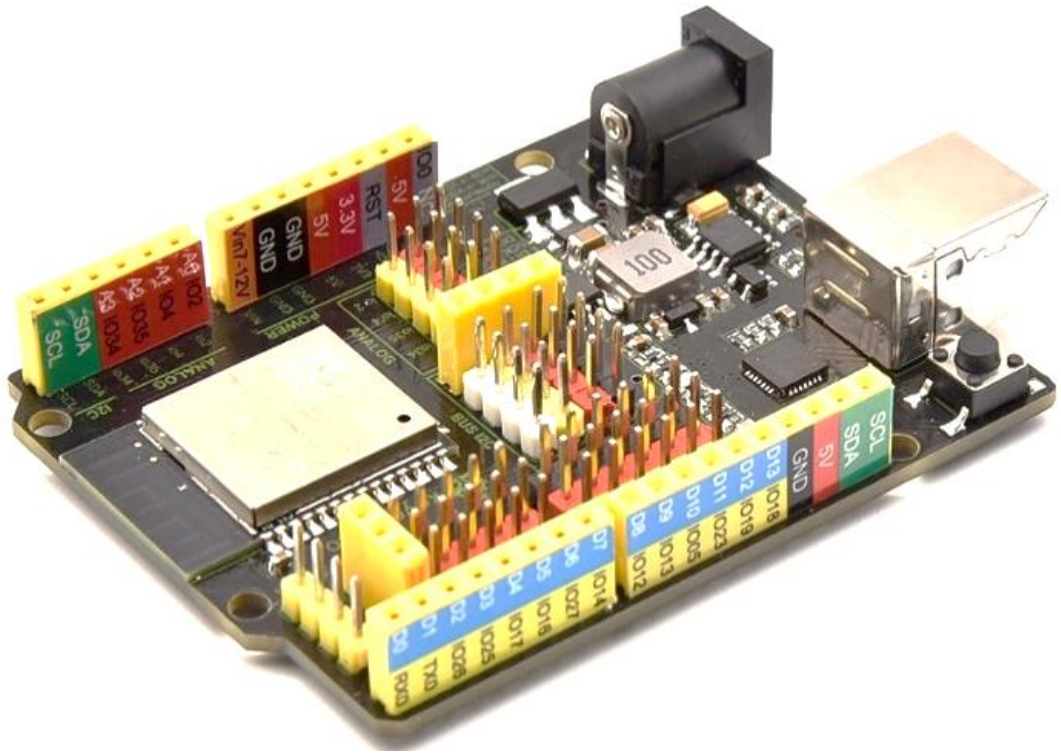
8 ESP32 Plus STEAMakers, todas sus posibilidades.

A continuación, os vamos a presentar todas las posibilidades que ofrece la placa ***ESP32 Plus STEAMakers***. Esta placa está basada en el microcontrolador ESP32-WROOM-32 de Espresiff.





Como vimos al principio del documento, existen una serie de pines que tienen diferentes funciones. En la siguiente tabla tenemos las funciones de cada pin. Para que sea más sencillo de encontrar los pines en la placa, están ordenados según la disposición de conexionado de la placa (las casillas tienen los mismos colores que las pegatinas de la placa).



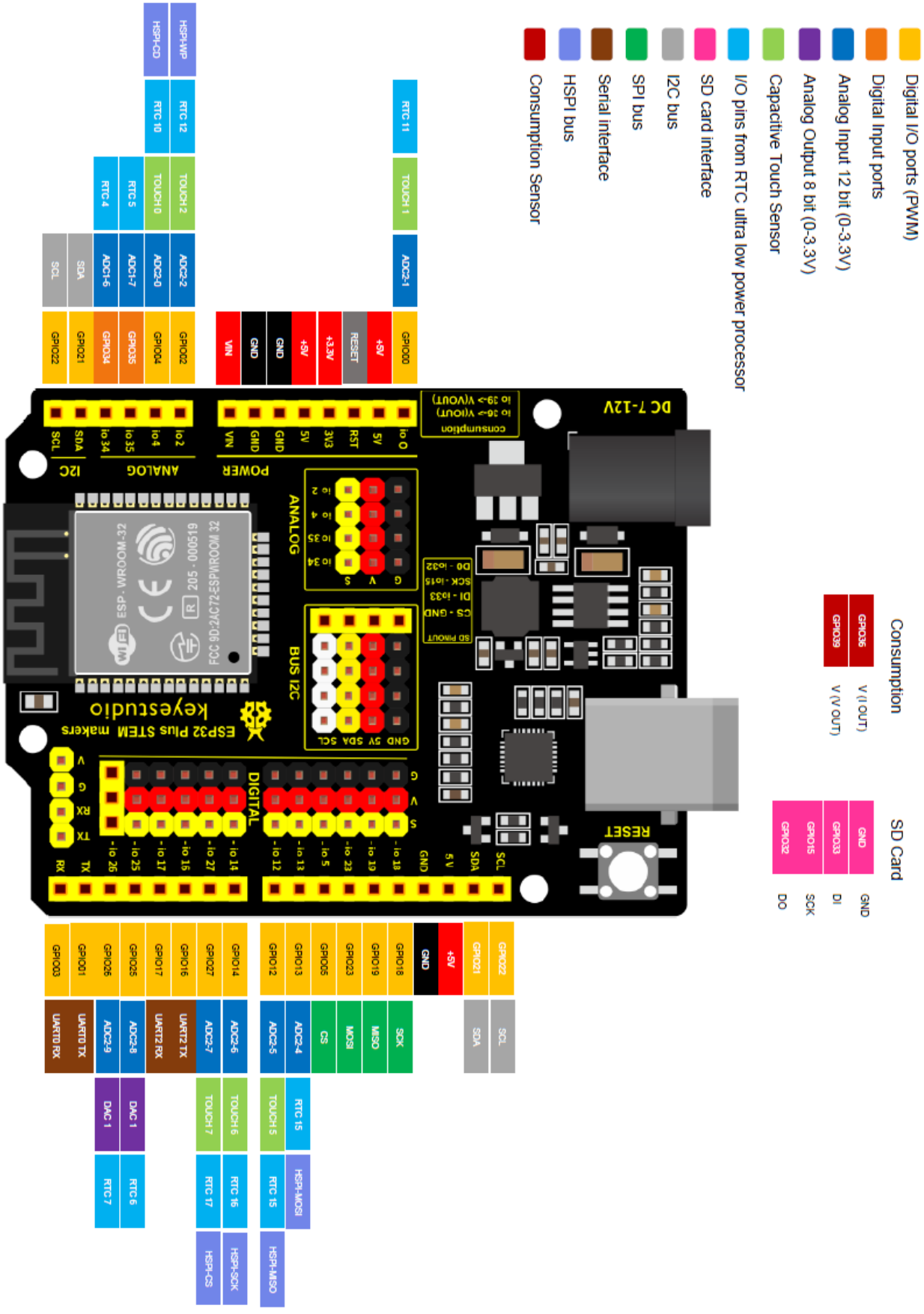
Todos los IOxx son también salidas PWM (menos IO00, no utilizable).

ESP32	PIN	Función 1	Función 2	Función 3
GPIO03	RX0	UART 0 RX	CLK2	
GPIO01	TX0	UART 0 TX	CLK3	
GPIO26	IO26	ADC2 CH9	DAC2	
GPIO25	IO25	ADC2 CH8	DAC1	
GPIO17	IO17	UART 2 TX		
GPIO16	IO16	UART 2 RX		
GPIO27	IO27	ADC2 CH7	TOUCH7	HSPI-CS
GPIO14	IO14	ADC2 CH6	TOUCH6	HSPI-SCK
GPIO12	IO12	ADC2 CH5	TOUCH5	HSPI-MISO
GPIO13	IO13	ADC2 CH4	TOUCH4	HSPI-MOSI
GPIO05	IO05	CS		
GPIO23	IO23	MOSI		
GPIO19	IO19	MISO	U0-CTS	
GPIO18	IO18	SCK		
GND	GND	GND		
	5V			
GPIO21	IO21	SDA	V-SPI-HD	
GPIO22	IO22	SCL	V-SPI-WP	

ESP32	PIN	Función 1	Función 2	Función 3
GPIO22	SCL			
GPIO21	SDA			
GPIO34	IO34	ADC1 CH6		
GPIO35	IO35	ADC1 CH7		
GPIO04	IO04	ADC2 CH0	TOUCH0	HSPI-CD
GPIO02	IO02	ADC2 CH2	TOUCH2	HSPI-WP
VIN	VIN7-12V	VIN7-12V		
GND GND	GND GND	GND GND		
	5V	5V		
3.3V	3V3	3V3		
EN	RST	RESET		
	5V	5V		
GPIO00	IO00	NC (no conectar)	TOUCH1	ADC2 CH1

ESP32	PIN	Función 1	Función 2	Función 3
GPIO32	IO32	D0 – Tarjeta	No son accesibles	
GPIO15	IO15	SLK – Tarjeta		
GPIO33	IO33	DI – Tarjeta		
GPIO36	IO36	IOUT	Medidor de corriente y tensión	
GPIO39	IO39	VOUT		

ESP32 Plus STEAMakers pinout:



Dispone de un interruptor para poder cambiar la alimentación entre 3,3V y 5V del bloque izquierdo de la placa, de IO26 a IO08 y para las 5 conexiones I2C (4 macho y 1 hembra).

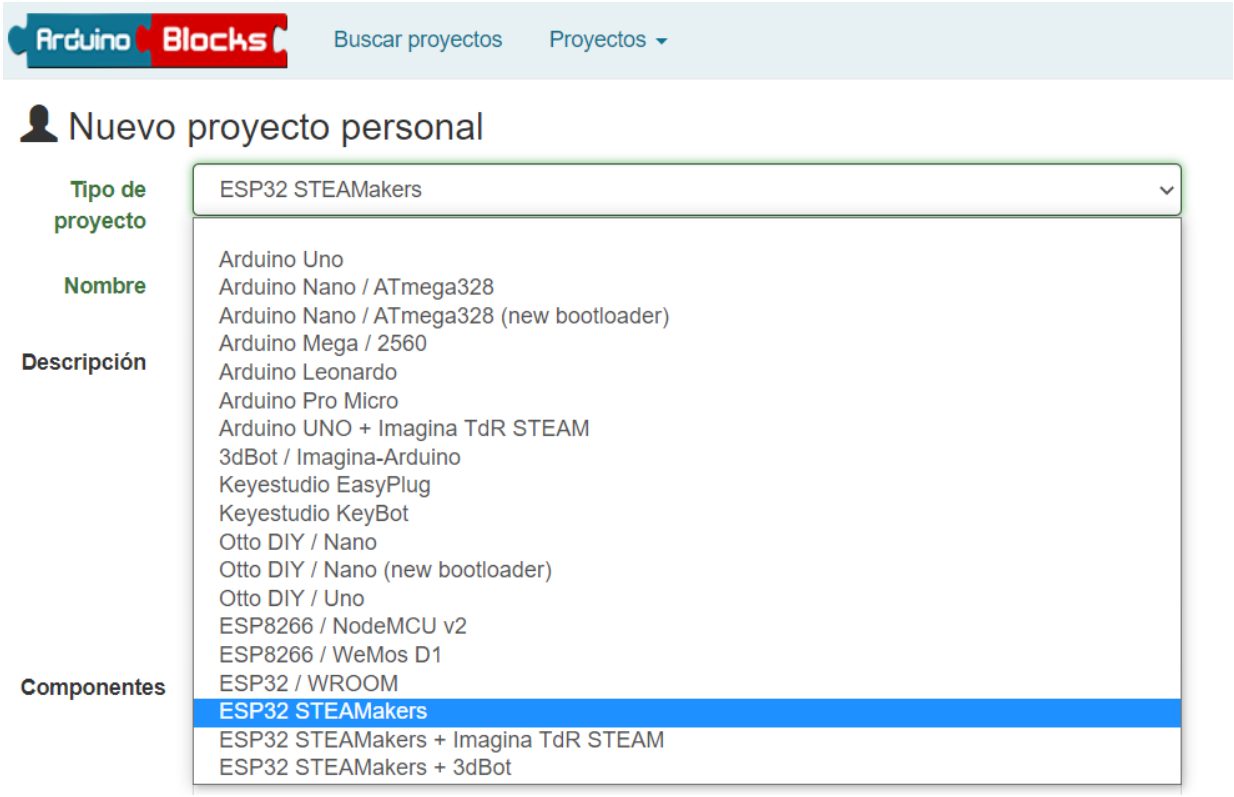
Tabla comparativa entre las diferentes placas educativas más utilizadas:

Specs/Board	STM32F103C8T6	ESP32	ESP8266	Arduino Nano
CPU	Arm Cortex-M3	Xtensa LX6	Xtensa L106	ATmega328P
CPU Core	1	2*	1	1
Architecture	32 bits	32 bits	32 bits	8 bits
CPU Frequency	72 MHz	160 MHz	80 MHz	16 MHz
Wireless	No	WiFi, Bluetooth	WiFi	No
RAM	20 KB	512 KB	160 KB	2 KB
Flash	64 KB/128kB	4 MB	4 MB	32 KB
GPIO Pins	37	36	17	14
SPI/I2C/I2S/UART	2/2/0/3	4/2/2/3	2/1/2/2	1/1/0/1
ADC Res/Ch	10 * 12-bit	6 * 12-bit	1 * 10-bit	6 * 10-bit
DAC Pins	0	2	0	0

Como podemos observar, el ESP32 nos ofrece unas prestaciones globales superior al resto de placas comparadas.

Todas las actividades que hemos realizado con la *shield* ***Imagina TDR STEAM*** se pueden realizar de forma similar, cambiando les sensores y actuadores utilizados en esta placa por unos externos. Realizaremos algunas actividades en forma de ejemplo y, posteriormente, funcionalidades que no se pueden utilizar si tenemos conectada la *shield*.

Para crear un nuevo proyecto utilizaremos el Tipo de proyecto: ***ESP32 Plus STEAMakers***.



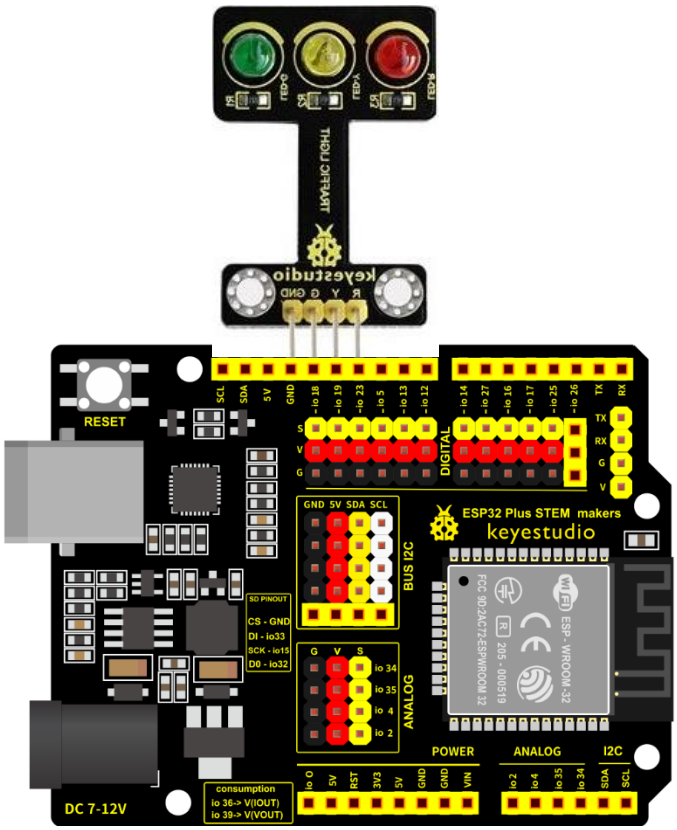
8.1 Reto A23. El semáforo.

Reto A23. El semáforo.

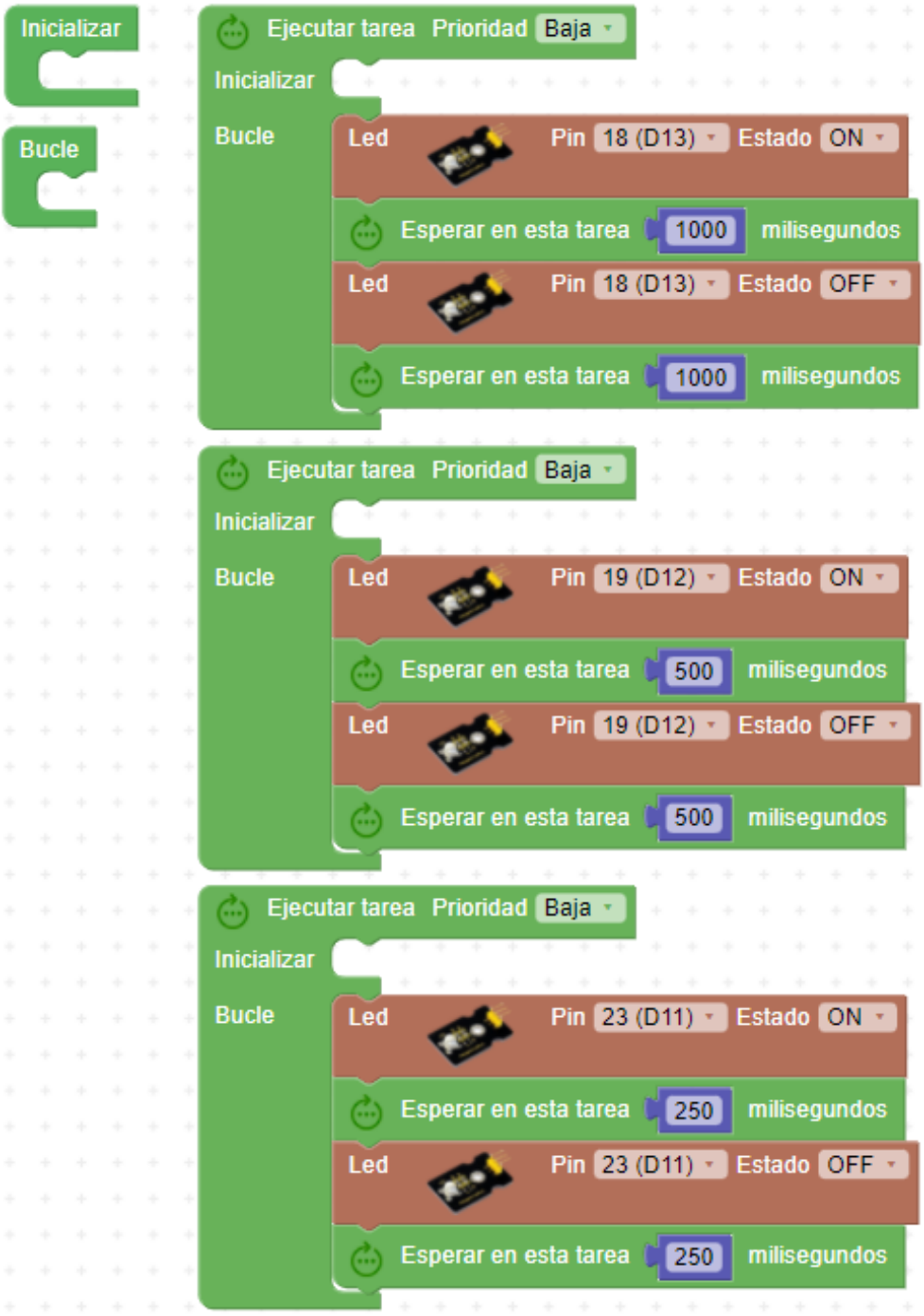
Vamos a empezar por programar un semáforo conectado directamente sobre la placa ***ESP32 Plus STEAMakers***. Existe un módulo de Keyestudio (KS0310) que se puede conectar directamente en los pines de salida (tal y como se muestra en la siguiente imagen).

Las conexiones son:

ESP32 STEAMakers	Semáforo
GND	GND
IO18	G (Green)
IO19	Y (Yellow)
IO23	R (Red)



Realizaremos un programa que ejecute tres intermitencias con intervalos de tiempo diferente mediante el método de multitarea.



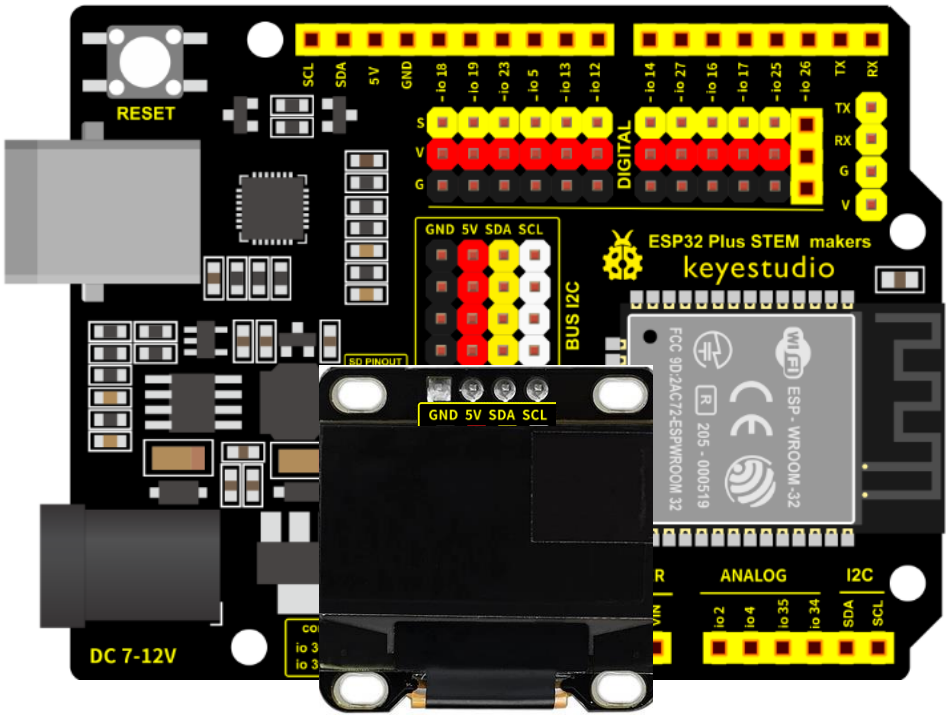
8.2 Reto A24. Pantalla OLED.

Reto A24. Pantalla OLED.

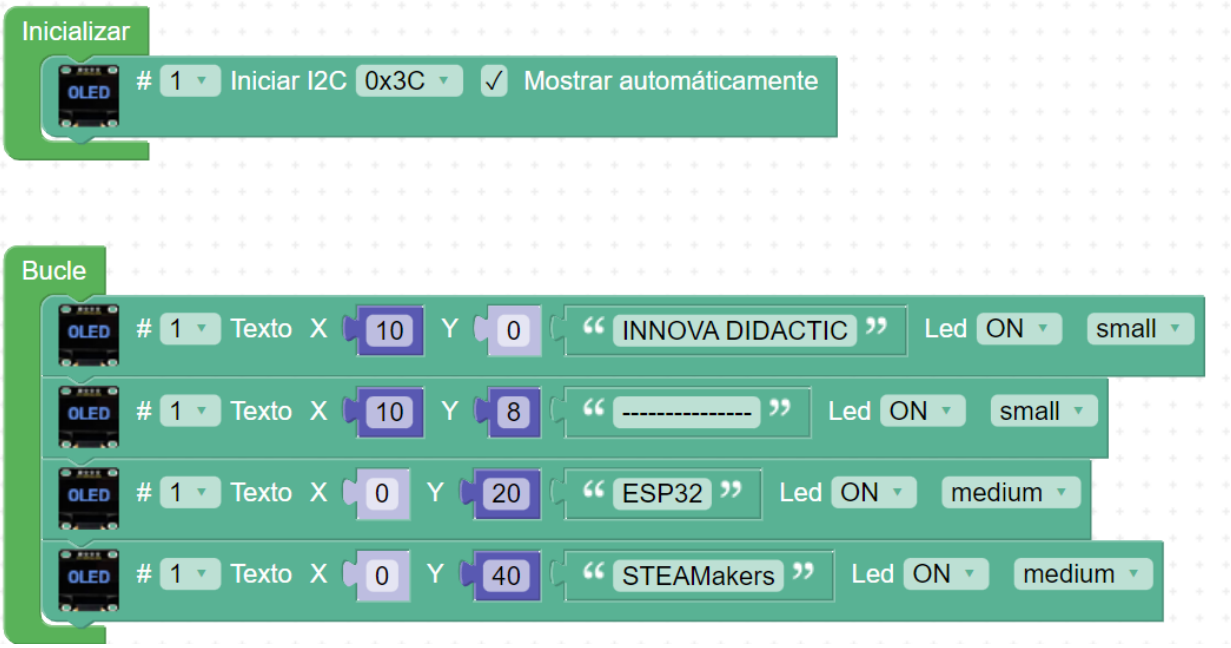
Vamos a realizar un programa para controlar la pantalla OLED. Existe un modelo de pantalla OLED de Keyestudio que se puede conectar directamente sobre la placa ***ESP32 Plus STEAMakers***.

Debemos tener muy en cuenta que existen muchos modelos de pantallas OLED que únicamente cambia su conexionado (pines). Hay que asegurarse que coinciden los pines de conexión de la pantalla OLED con los de la placa ***ESP32 Plus STEAMakers***.

En la imagen que hay a continuación está indicada la posición de la pantalla y los pines de conexión.



A continuación, realizaremos un programa para que mande unos mensajes a la pantalla.



8.3 Reto A25. Sensor TOUCH.

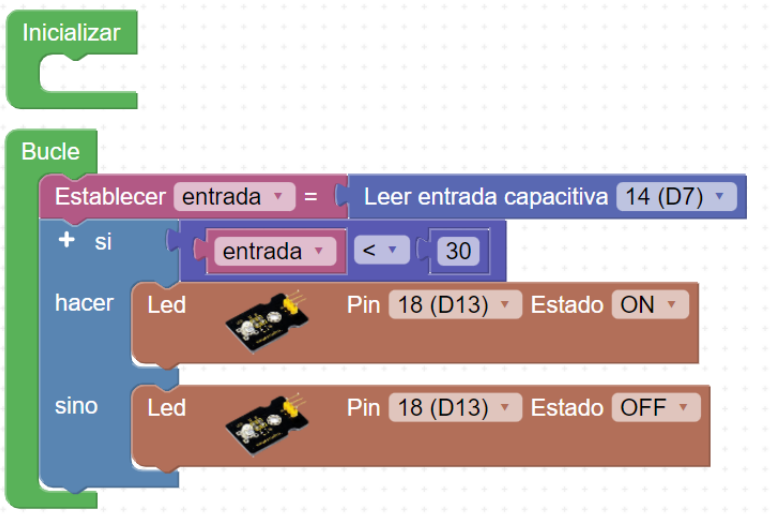
Reto A25. Sensor TOUCH.

La placa ***ESP32 Plus STEAMakers*** dispone de una serie de entradas marcadas como TOUCH. Este tipo de entrada, es una entrada de tipo capacitiva que solo necesita un cable para funcionar como si fuera un pulsador táctil.

Las entradas tipos TOUCH que dispone el ESP32 son las siguientes:

- GPIO00: TOUCH1 (no disponible en ArduinoBlocks – no es recomendable la utilización de este pin)
- GPIO02: TOUCH2
- GPIO04: TOUCH0
- GPIO12: TOUCH5
- GPIO13: TOUCH4
- GPIO14: TOUCH6
- GPIO15: TOUCH3 (no disponible – SD)
- GPIO27: TOUCH7
- GPIO32: TOUCH9 (no disponible – SD)
- GPIO33: TOUCH8 (no disponible – SD)

Vamos a realizar un programa que encenderá un led conectado en D13 (IO18) y que se encenderá cuando toquemos un cable conectado en D7 (IO14).



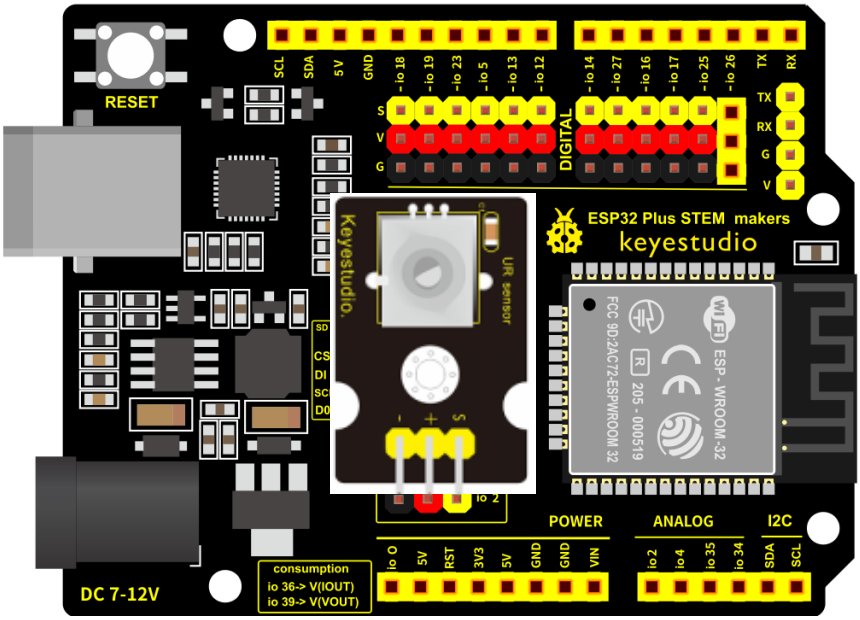
Enlace al programa: ArduinoBlocks Projects\touch.abp

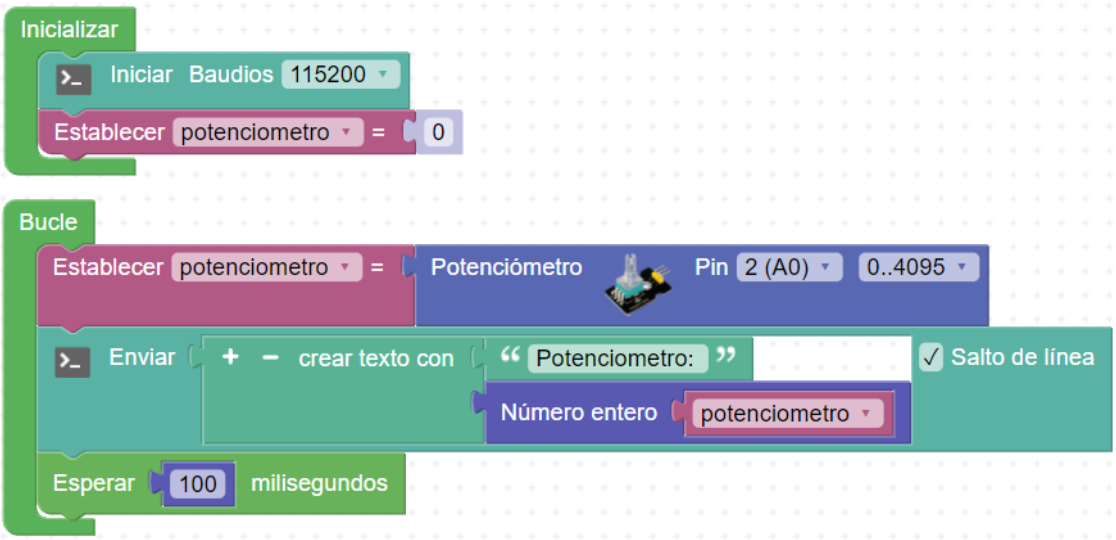
8.4 Reto A26. Potenciómetro.

Reto A26. Potenciómetro.

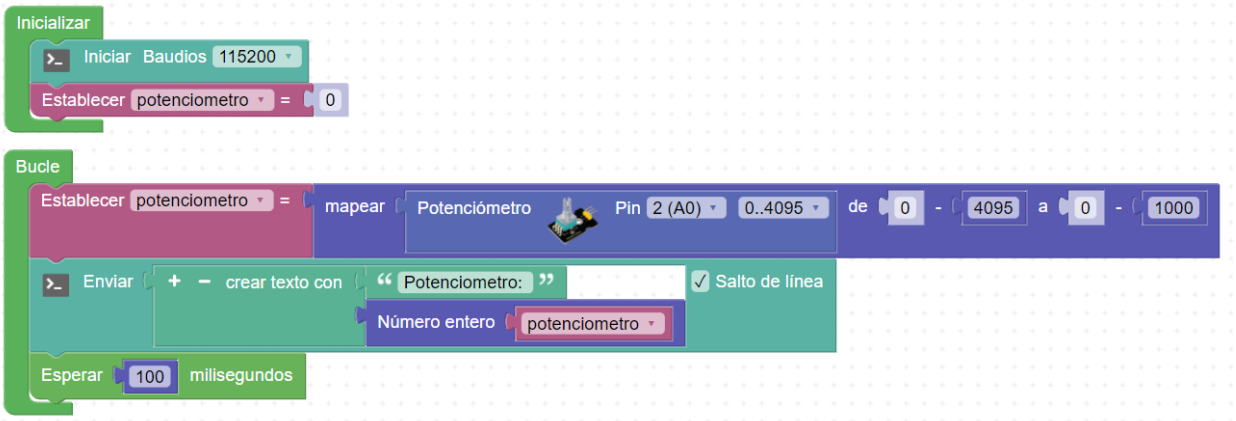
La placa ***ESP32 Plus STEAMakers*** dispone de cuatro entradas analógicas con alimentación incorporada que nos permite conectar un potenciómetro de forma directa.

Lo podemos conectar de la forma siguiente. El rango de medición lo podemos seleccionar en porcentaje (% de 0 a 100) o en valor de 0 a 4095 (que corresponde a una lectura de 12 bits). Debemos recordar que el ESP32 funciona a 3,3V y, las entradas analógicas solamente podrán leer valores hasta 3,3V. La tener la alimentación del potenciómetro en la placa, este lo podemos alimentar directamente a 3,3V por lo que estará midiendo en todo su rango.





Podemos acondicionar este rango de medición a cualquier rango de trabajo mediante el bloque de *mapear*. Este bloque realiza un cambio de escala entre la que tenemos y la que queremos obtener. Por ejemplo, si queremos cambiar el rango anterior y tener un rango entre 0 y 1000 unidades, procederemos de la siguiente forma:

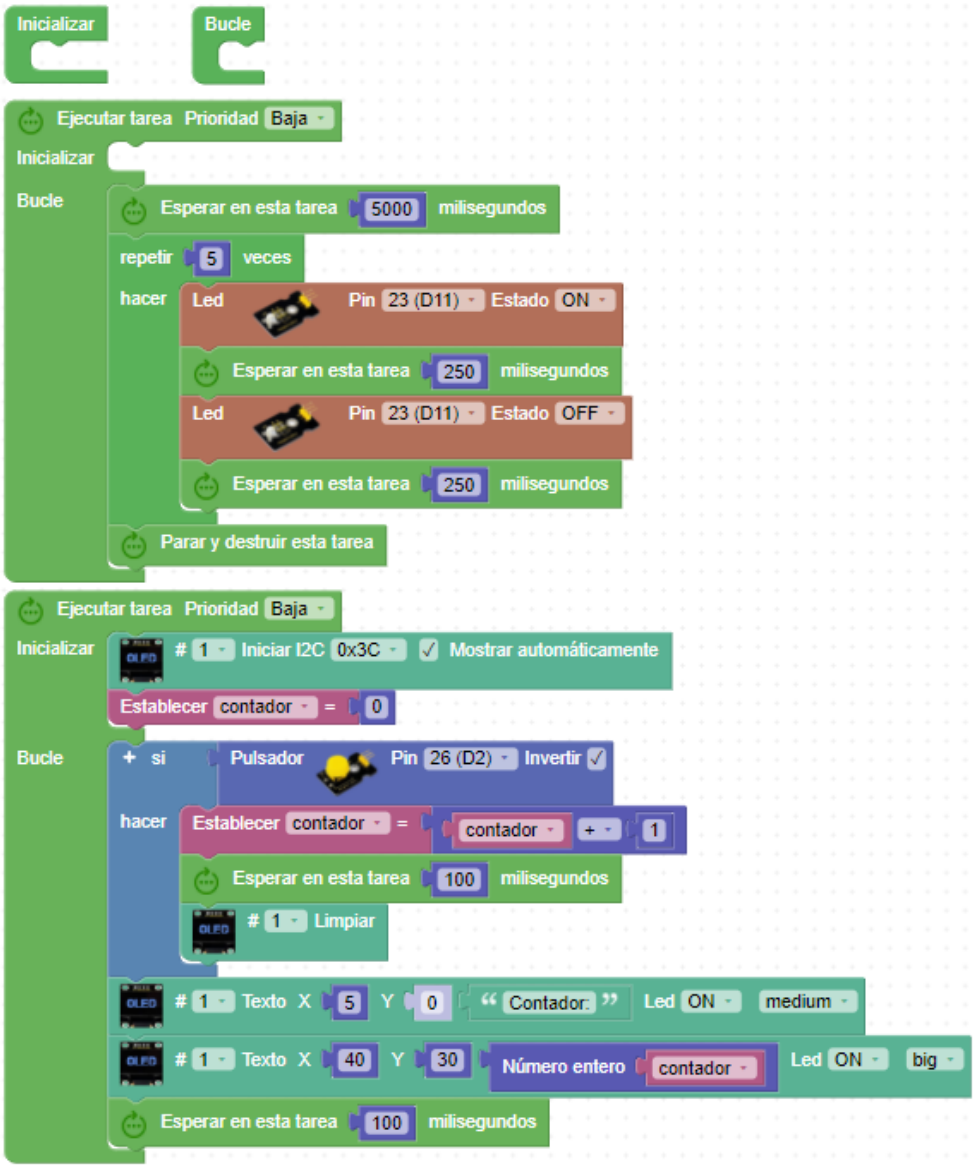


8.5 Reto A27. Multitarea avanzada 1.

Reto A27. Multitarea avanzada 1.

En ArduinoBlocks podemos eliminar una tarea que solo queramos que se ejecute un número determinado de veces y después deje de ejecutarse.

Vamos a realizar un programa que en una de las tareas encenderá parpadeando la luz roja del semáforo (IO23 – D11) al cabo de 5 segundos de empezar el programa. Parpadeará 5 veces y después se destruirá esta tarea. En la pantalla OLED mostraremos un contador de pulsaciones de un pulsador conectado en IO26 (D2). Como tenemos un pulsador NC (funciona al revés) introduciremos una negación en la lectura del pin donde se encuentra el pulsador (activando el *tic* de Invertir).

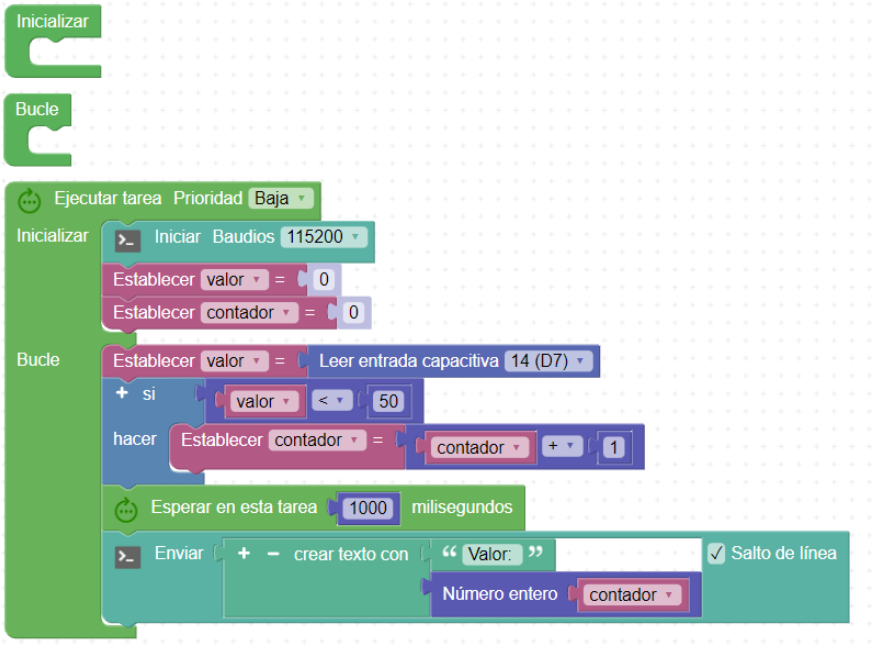


Enlace al programa: ArduinoBlocks Projects\multitarea_avanzada_1.abp

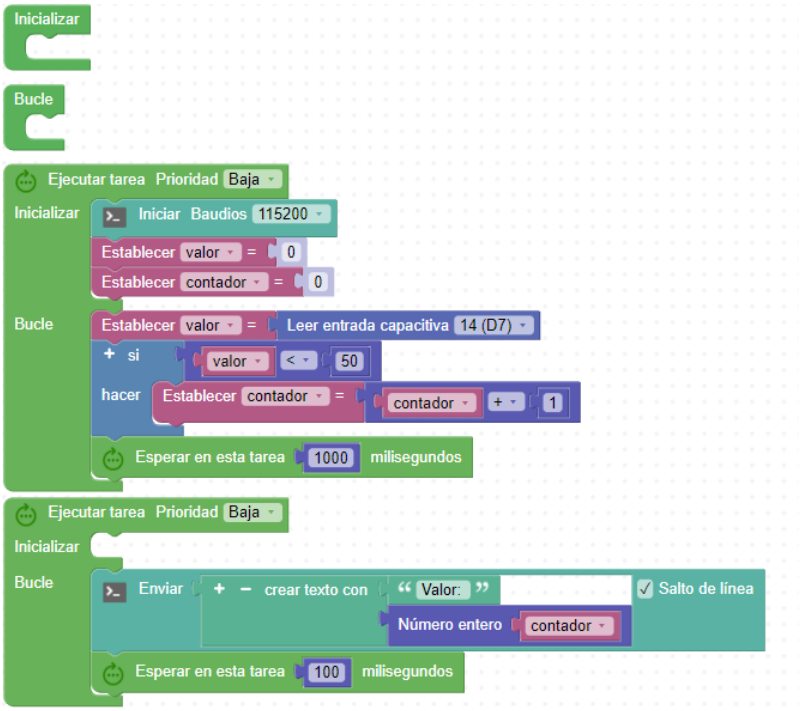
8.6 Reto A28. Multitarea avanzada 2.

Reto A28. Multitarea avanzada 2.

La placa ***ESP32 Plus STEAMakers*** permite realizar multitarea. Podemos tener tareas que compartan variables.

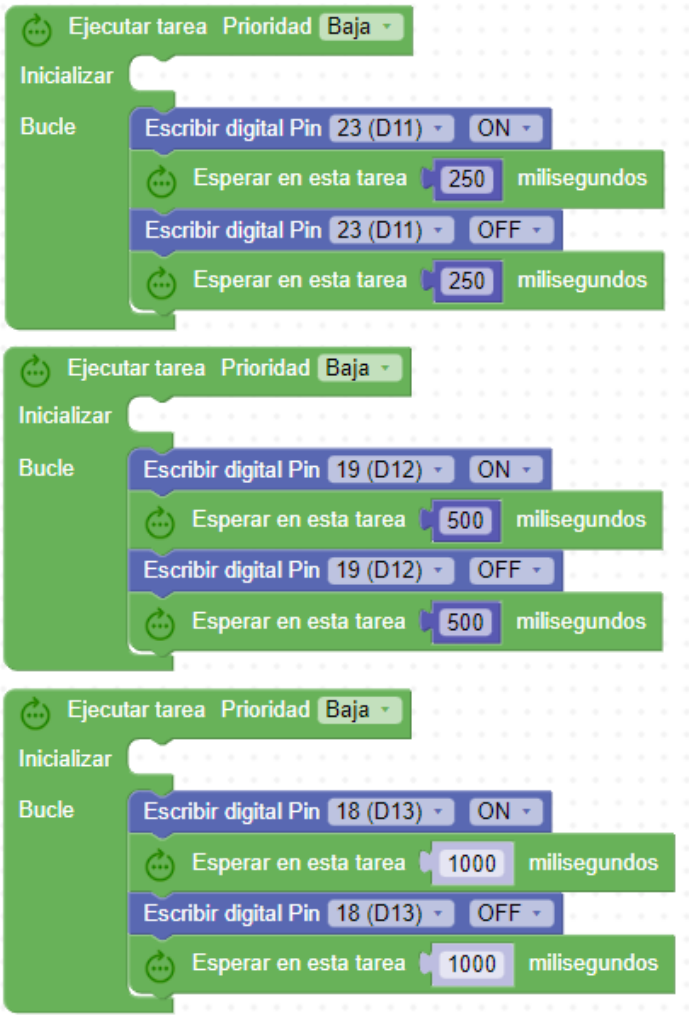


Este programa permitirá mostrar los datos de forma más rápida.



Enlace al programa: ArduinoBlocks Projects\multitarea_avanzada_2.abp

También podemos añadir la intermitencia de varios leds (como los de un semáforo conectado en D11-D12-D13). Las tareas a añadir son las siguientes:

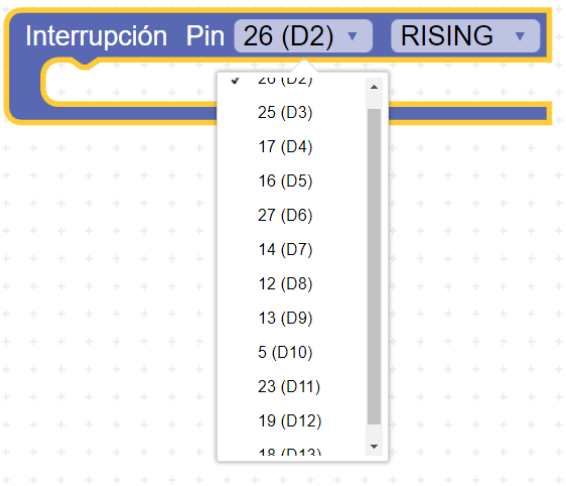


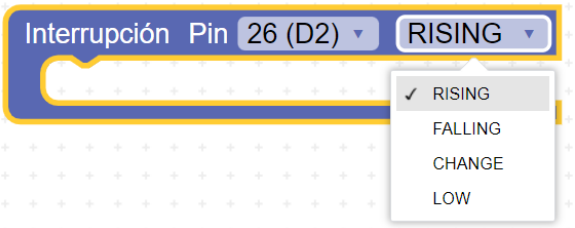
Enlace al programa: ArduinoBlocks Projects\multitarea_avanzada_3.abp

8.7 Reto A29. Interrupciones.

Reto A29. Interrupciones.

La placa ***ESP32 Plus STEAMakers*** permite realizar interrupciones. Tenemos pines que permiten definir o crear interrupciones en el microcontrolador de la placa para que deje de hacer una tarea, atienda a la interrupción (ejecutando el código incluido en ella) y volver al punto donde se había creado. El código que hay dentro de la interrupción debe durar un corto espacio de tiempo, ya que, sino, dejará de ejecutar la interrupción. Disponemos de los pines de D2 a D13 para poder introducir una interrupción. Existen cuatro modos de interrupción en función del estado del pulsador.

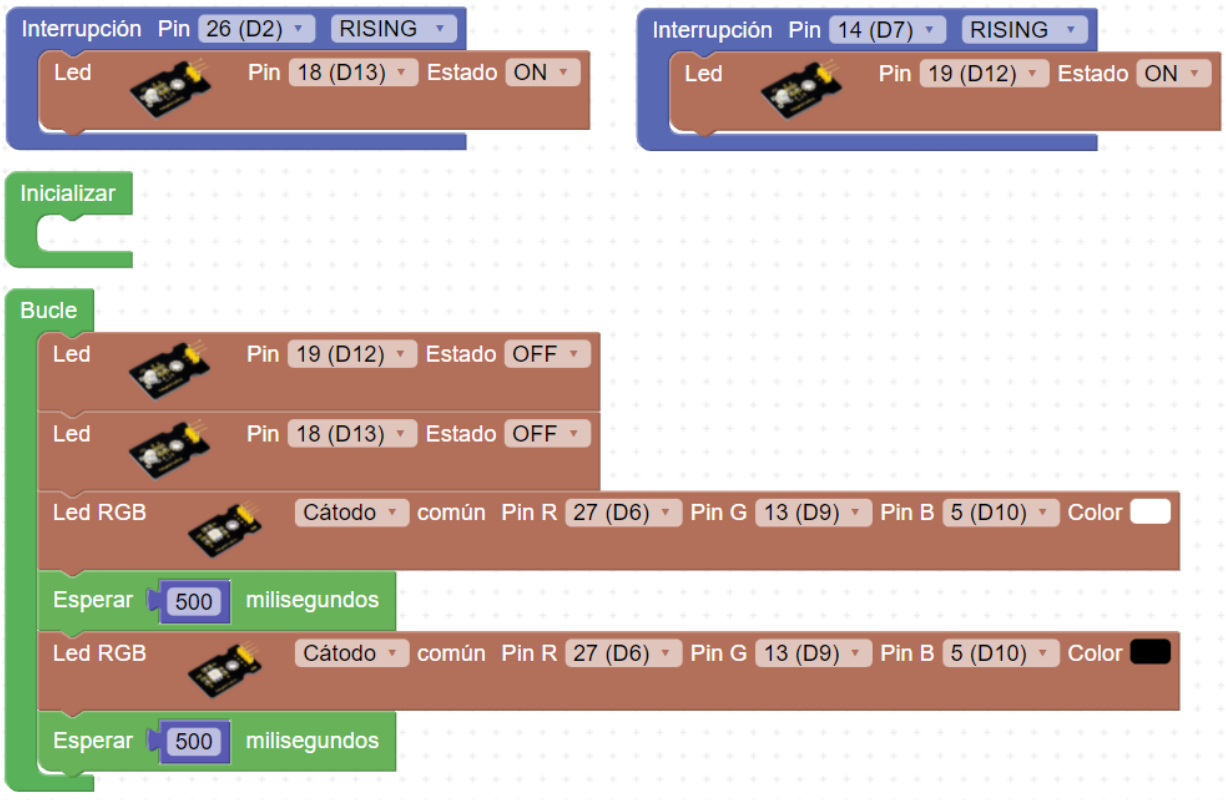




Vamos a realizar un programa que ejecute dos interrupciones mediante dos pulsadores conectados en IO26 (D2) y IO14 (D7) que controlarán el encendido de dos leds (cada led está asociado a un pulsador).

En el programa principal, el led RGB está haciendo una intermitencia entre blanco y negro cada 500ms.

El programa propuesto es el siguiente:



Podemos comprobar que pueden llegar a ejecutarse las dos interrupciones de forma simultánea (al apretar los dos pulsadores a la vez se encienden los dos leds).

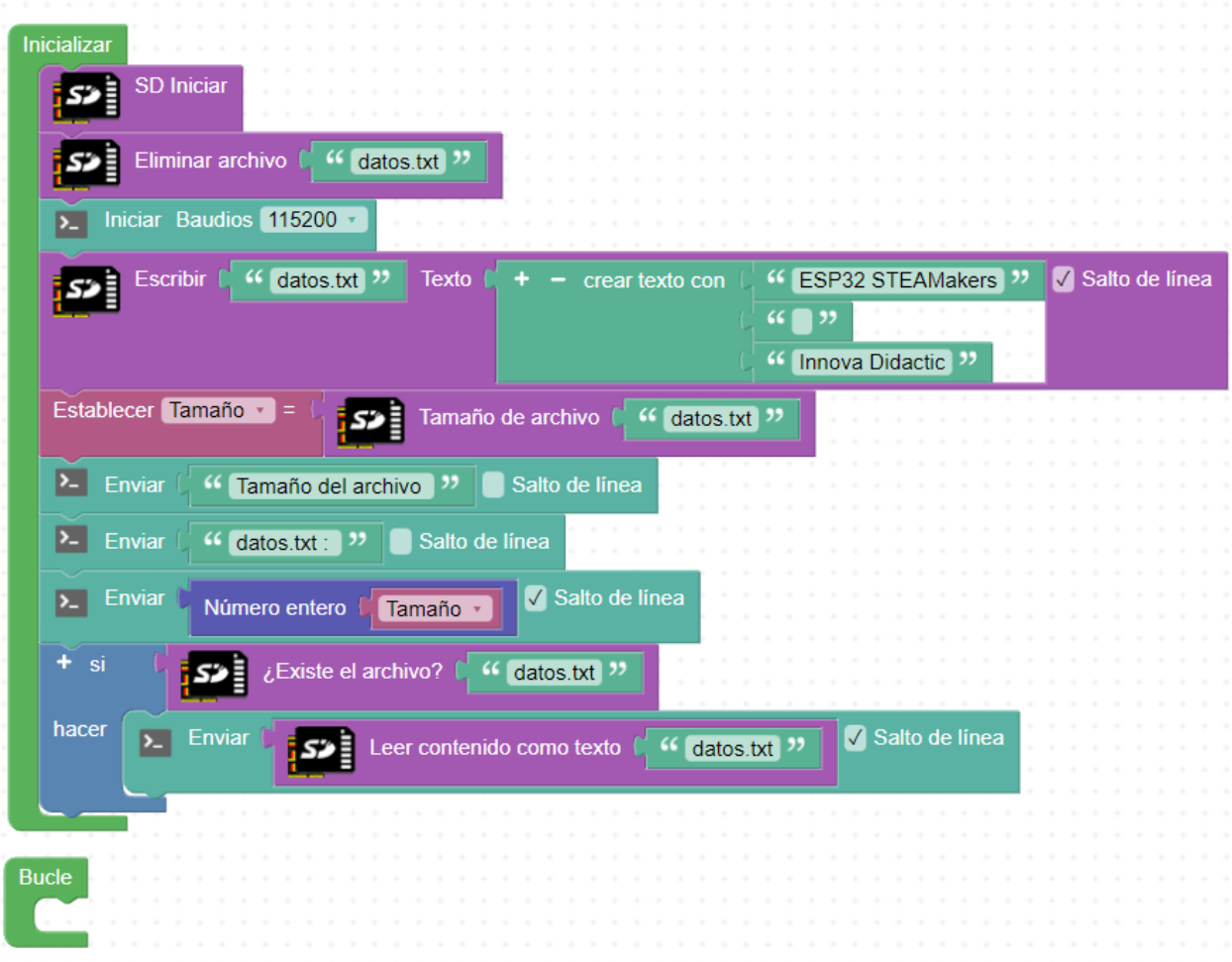
Enlace al programa: ArduinoBlocks Projects\interrupciones_dobles.abp

8.8 Reto A30. Profundizando en la SD 1.

Reto A30. Profundizando en la SD 1.

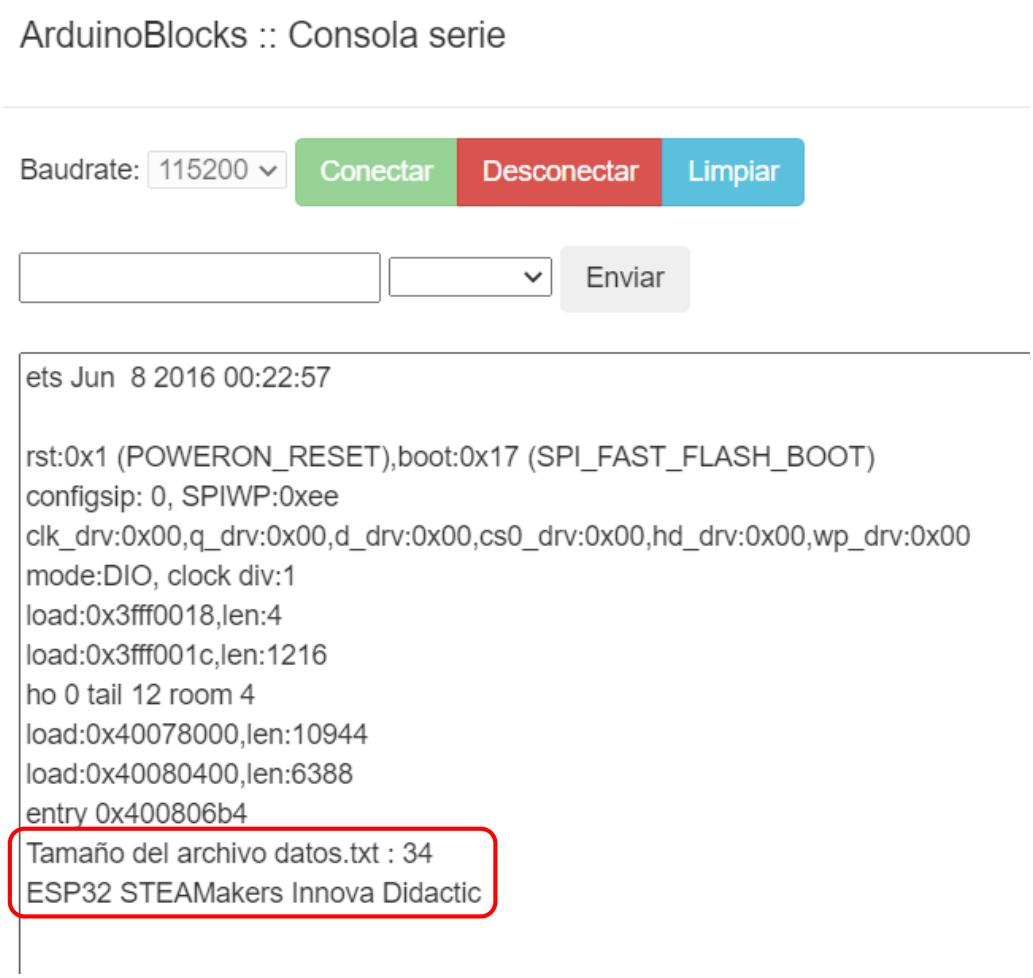
La placa ***ESP32 Plus STEAMakers*** dispone de una ranura para una tarjeta SD. Esto nos permite poder almacenar información en una tarjeta de memoria.

Vamos a realizar un programa que guarde un texto en un fichero (*datos.txt*) para, posteriormente, leer el tamaño del fichero con los datos y visualizar el contenido del fichero a través de la *Consola*.



Enlace al programa: ArduinoBlocks Projects\sd_profundidad_1.abp

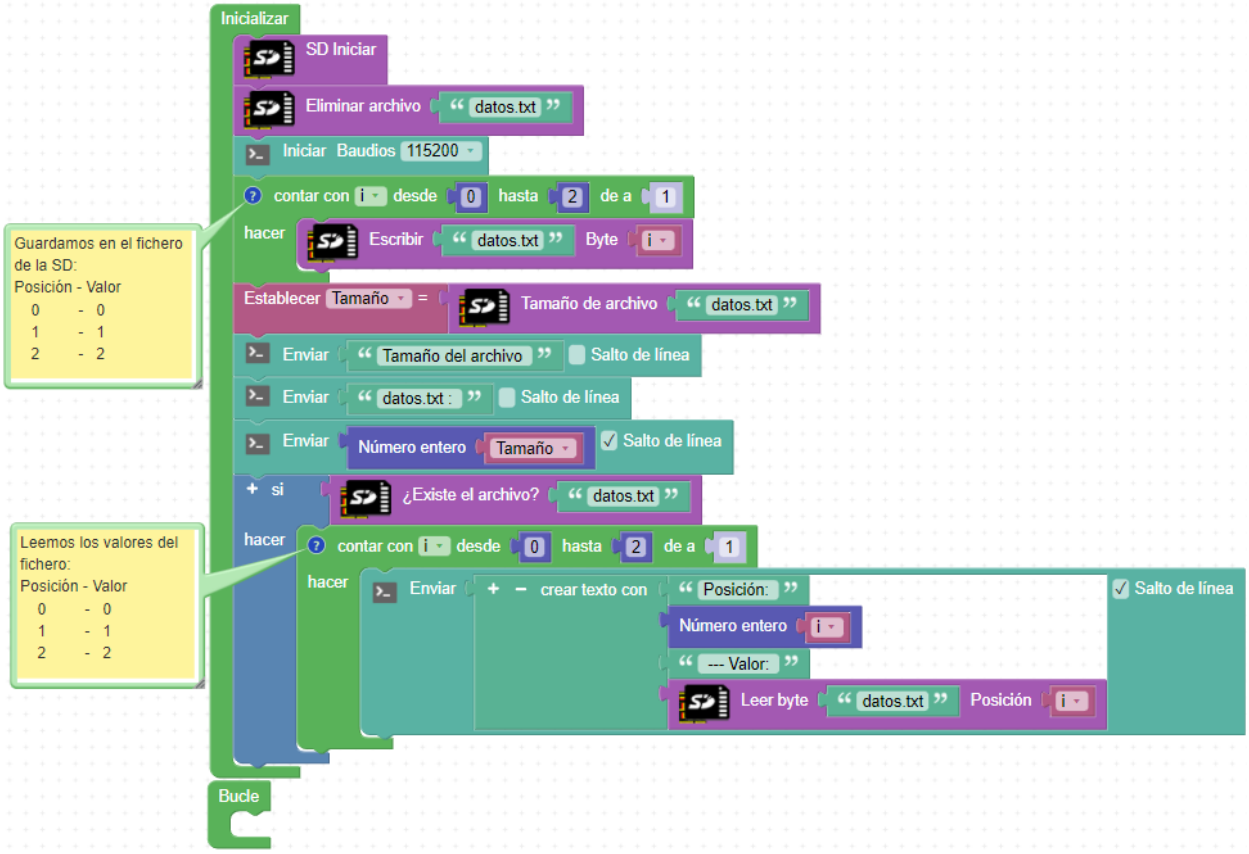
Los datos obtenidos en la *Consola* son los siguientes:



8.9 Reto A31. Profundizando en la SD 2.

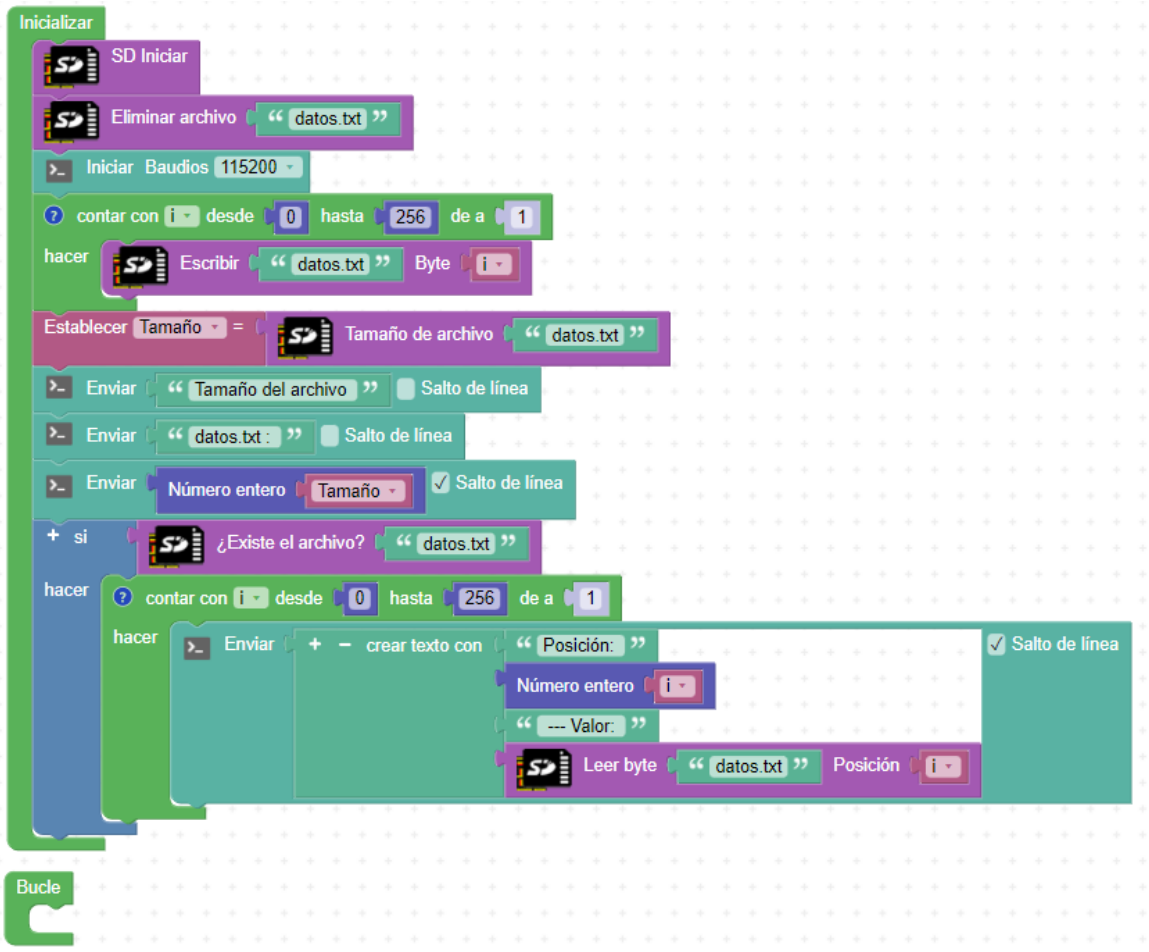
Reto A31. Profundizando en la SD 2.

En la actividad anterior hemos trabajado con datos de texto, ahora vamos a trabajar con bytes. Podemos almacenar valores (bytes) con un valor entre 0 y 255. Vamos a realizar un programa que guarde el valor de la posición (3 bytes).



Enlace al programa: ArduinoBlocks Projects\sd_profundidad_2.abp

Si guardamos un valor mayor de 256, el valor almacenado será 0. Vamos a realizar un programa para comprobarlo.



Enlace al programa: ArduinoBlocks Projects\sd_profundidad_3.abp

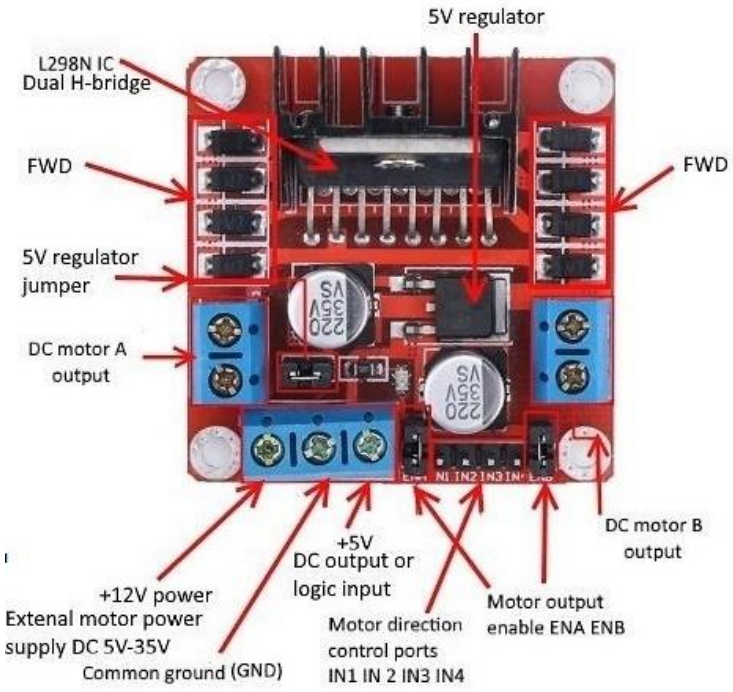

ArduinoBlocks :: Consola serie
Baudrate: 115200
Conectar
Desconectar
Limpiar
Enviar
Posición: 238 --- Valor: 238
Posición: 239 --- Valor: 239
Posición: 240 --- Valor: 240
Posición: 241 --- Valor: 241
Posición: 242 --- Valor: 242
Posición: 243 --- Valor: 243
Posición: 244 --- Valor: 244
Posición: 245 --- Valor: 245
Posición: 246 --- Valor: 246
Posición: 247 --- Valor: 247
Posición: 248 --- Valor: 248
Posición: 249 --- Valor: 249
Posición: 250 --- Valor: 250
Posición: 251 --- Valor: 251
Posición: 252 --- Valor: 252
Posición: 253 --- Valor: 253
Posición: 254 --- Valor: 254
Posición: 255 --- Valor: 255
Posición: 256 --- Valor: 0

8.10 Reto A32. Control de motores I.

Reto A32. Control de motores I.

Vamos a ver cómo funciona el sistema de control de motores de corriente continua. Los motores de corriente continua se controlan mediante la variación de la tensión aplicada. Para realizarlo es necesario colocar una etapa de potencia entre la placa de control y los motores.

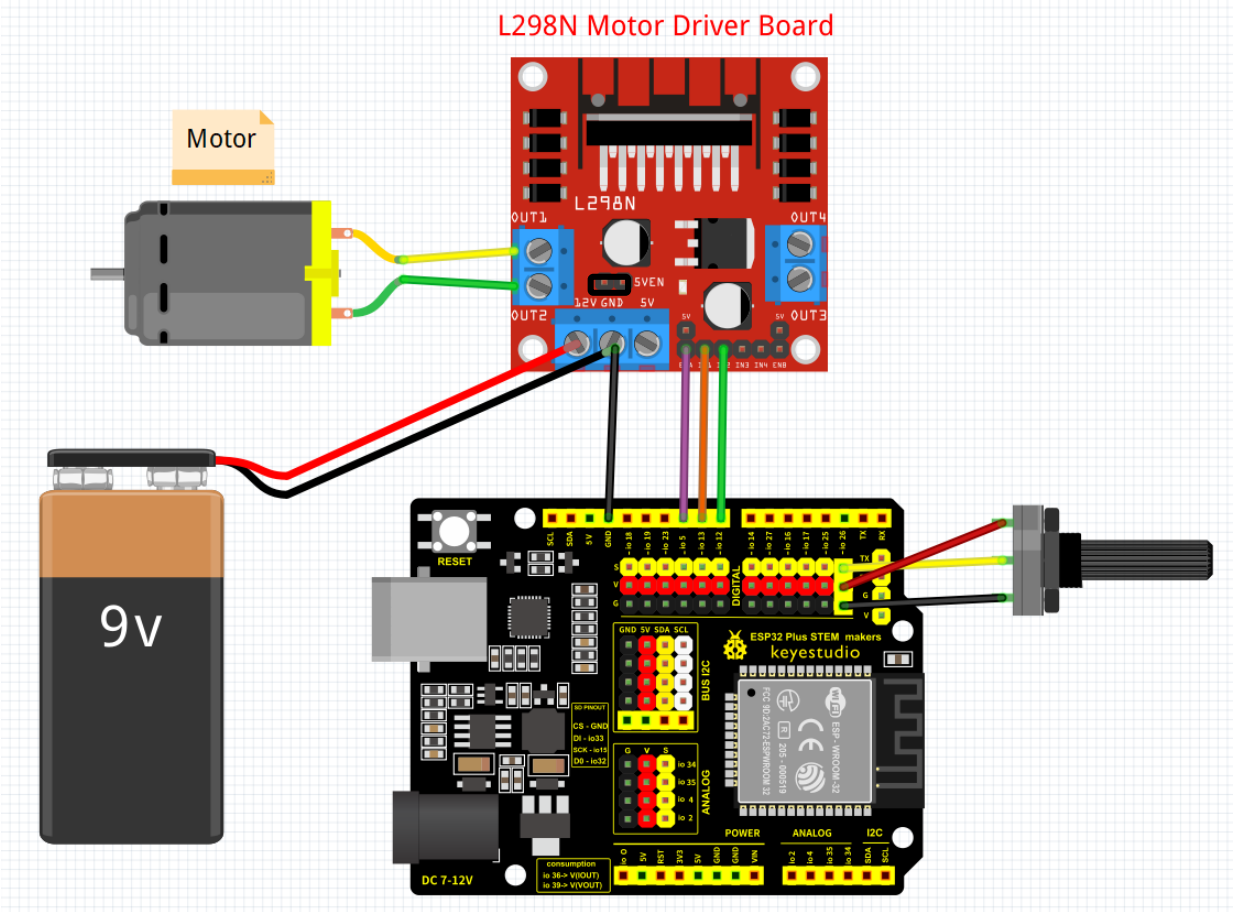
Vamos a utilizar la etapa de potencia L298N. Esta etapa permite controlar la velocidad y el sentido de giro de dos motores de corriente continua.



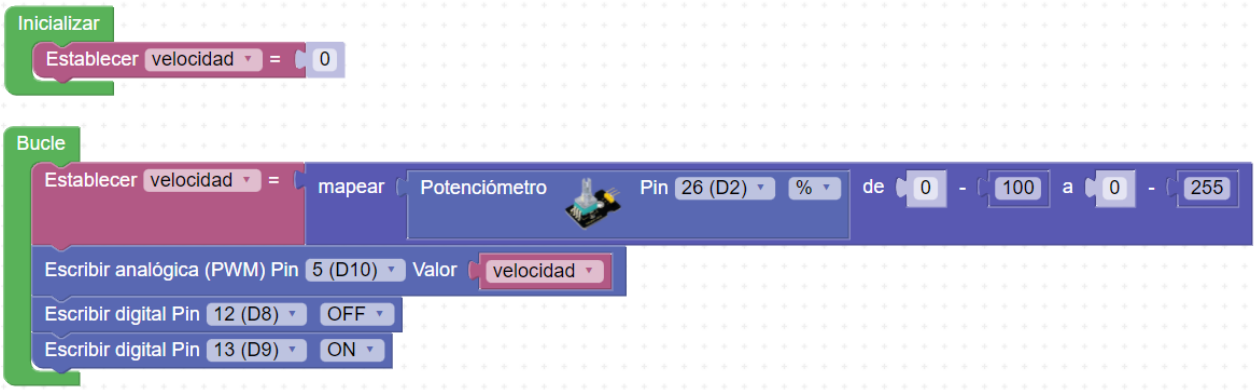
Para empezar, vamos a realizar el control de un motor de corriente continua mediante un potenciómetro. Conectaremos el potenciómetro en D2, ENA con D10, IN1 con D9 y IN2 con D8.

El control de velocidad se realiza mediante modulación PWM por la entrada ENA de la etapa de potencia L298N. Por las entradas IN1 e IN2 controlamos el sentido de giro del motor.

Vamos a conectar todos los elementos como se muestran en la imagen (debemos poner el puente de la alimentación):

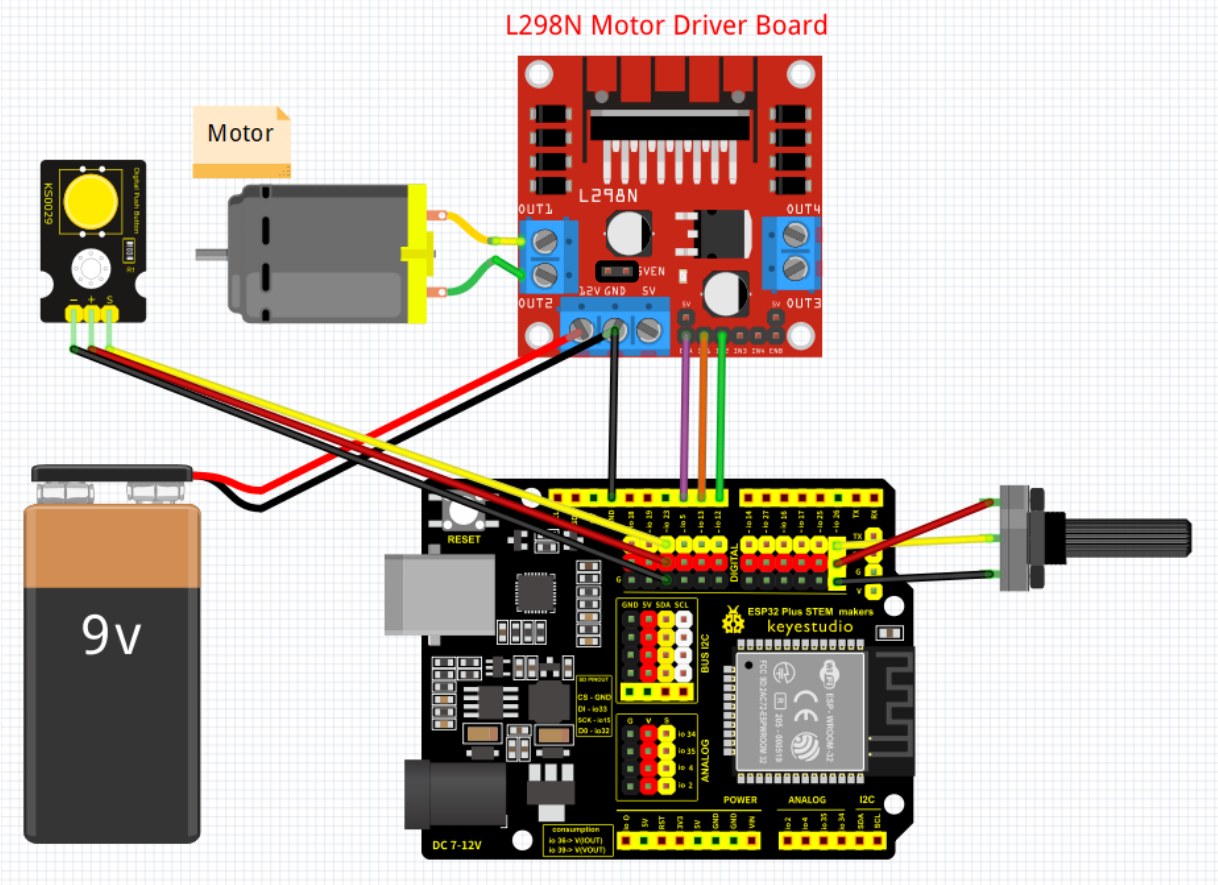


Y el programa para control el motor con el potenciómetro es el siguiente:



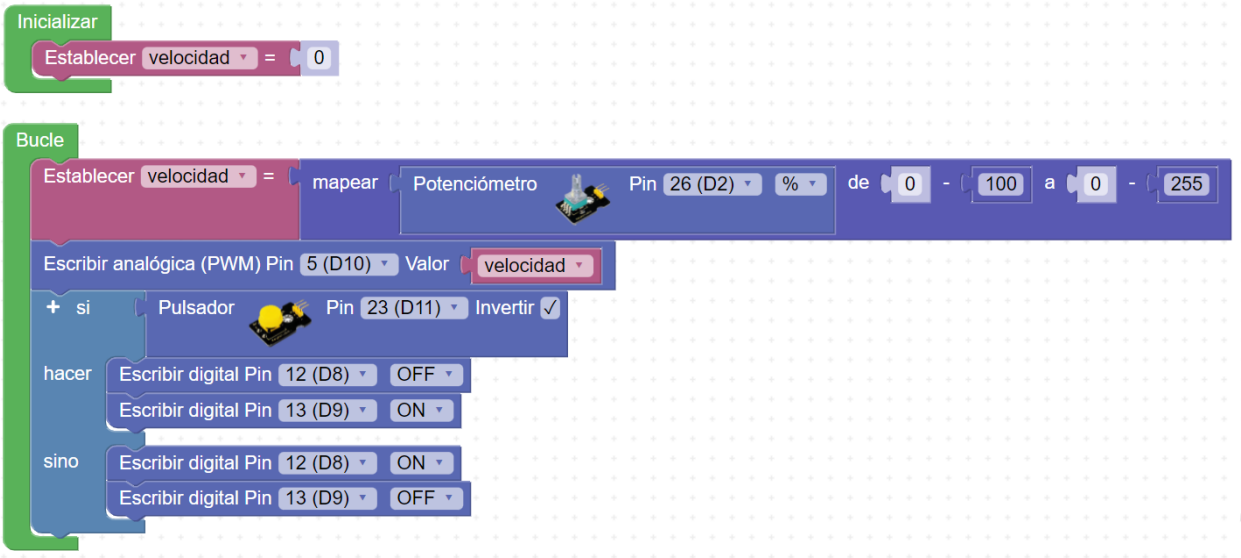
Enlace al programa: ArduinoBlocks Projects\control_motores_1.abp

Vamos a mejorar el programa añadiendo un pulsador para seleccionar el sentido de giro. Si apretamos el pulsador, el motor girará en un sentido y sino girará en el otro sentido.



Enlace al esquema: Esquemas Fritzing\1_motor.fzz

El programa será:

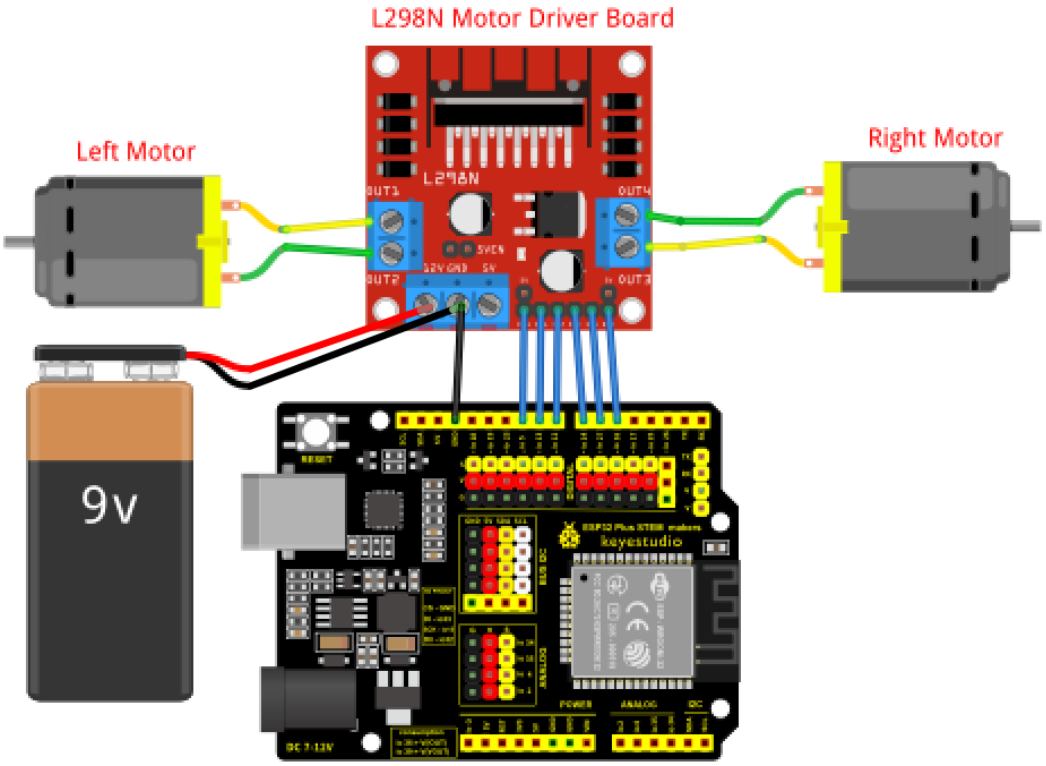


Enlace al programa: ArduinoBlocks Projects\control_motores_2.abp

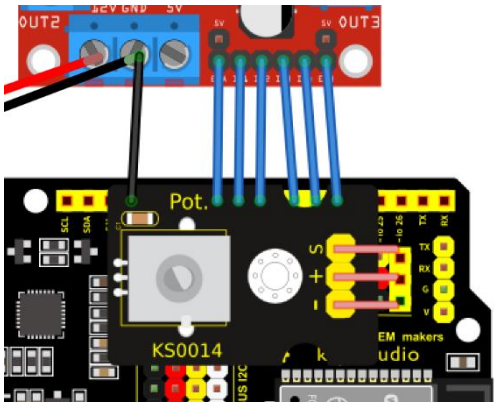
8.11 Reto A33. Control de motores II.

Reto A33. Control de motores II.

Ahora vamos a realizar el control de dos motores simultáneamente. El control de la velocidad mediante PWM se hace a través de ENA (motor A) y ENB (motor B). Las entradas IN1-IN2 sirven para controlar el sentido de giro del motor A y IN3-IN4 sirven para controlar el sentido de giro de B. El esquema de conexionado será el siguiente:

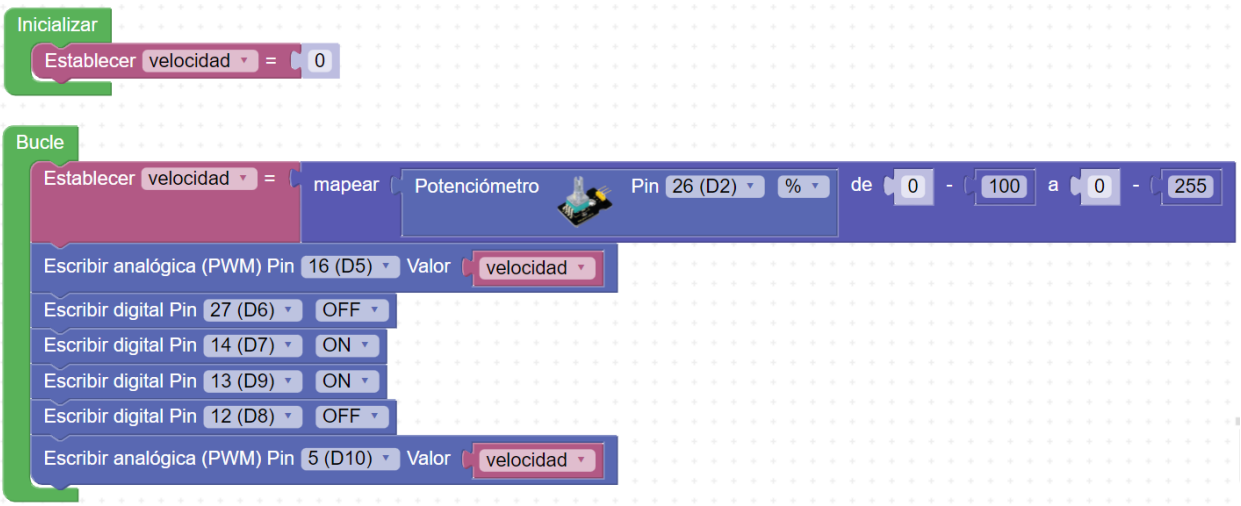


Primero vamos a controlar la velocidad dos motores mediante un potenciómetro conectado en GPIO26 (D2). Conectaremos el potenciómetro directamente en el conector hembra que hay en GPIO26 (D2).



Enlace al esquema: Esquemas Fritzing\2_motores.fzz

Con este sencillo programa podemos controlar la velocidad de los dos motores a la vez.



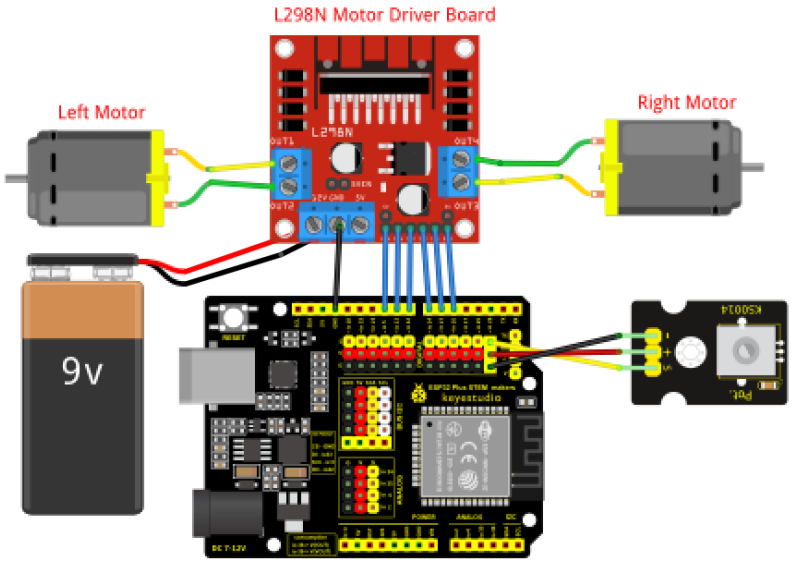
Enlace al programa: ArduinoBlocks Projects\control_2_motores.abp

En las siguientes actividades mejoraremos el control de los motores permitiendo modificar el sentido de giro, velocidad, etc.

8.12 Reto A34. Control de motores III.

Reto A34. Control de motores III.

Ahora vamos a controlar el sentido de giro mediante el envío de comando a través del puerto serie (Consola). El esquema de conexionado es el mismo que en la actividad anterior. Conectaremos un potenciómetro en D2 para controlar la velocidad.



Enlace al esquema: Esquemas Fritzing\2_motores.fzz

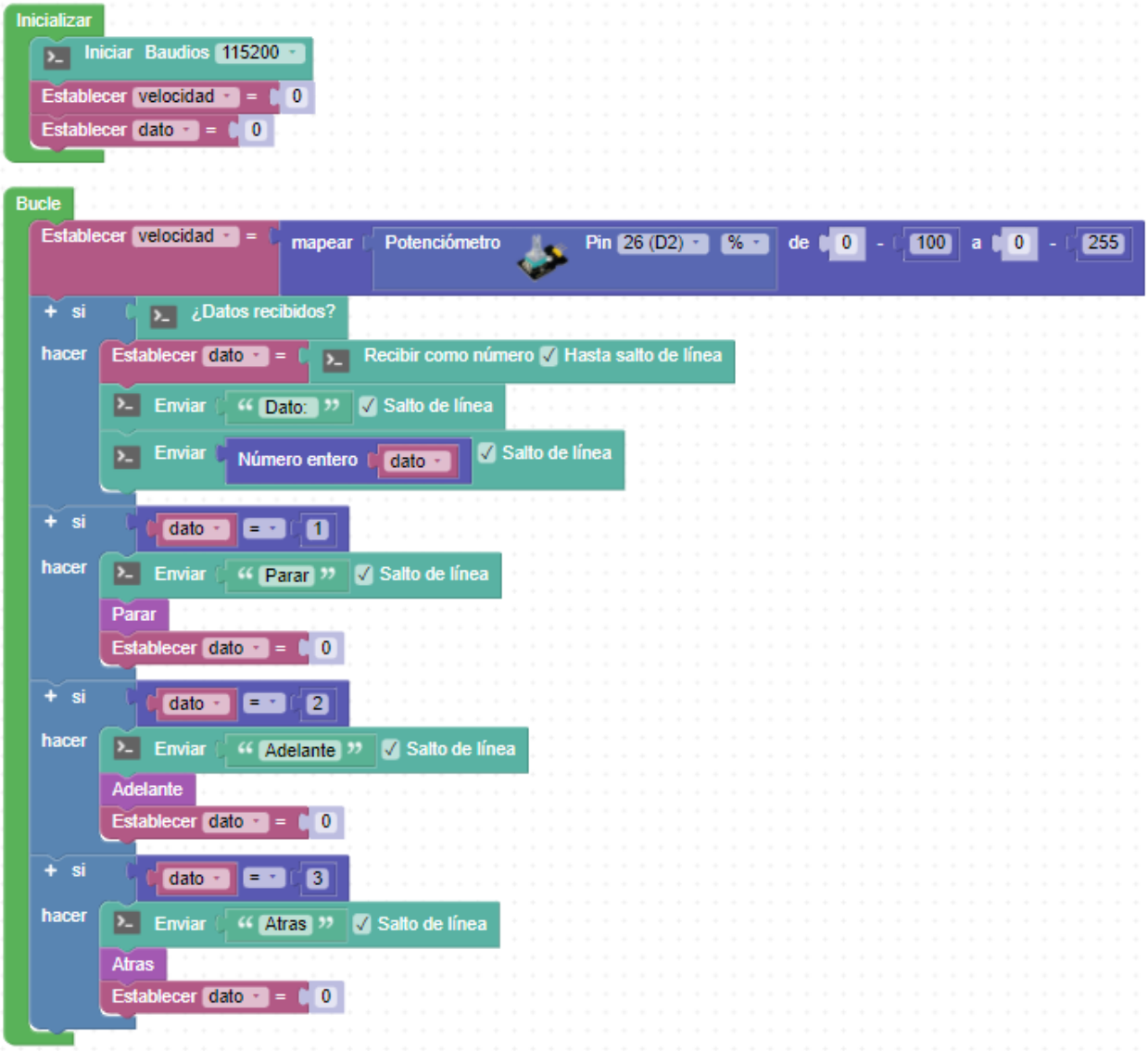
Los comandos de control serán los siguientes:

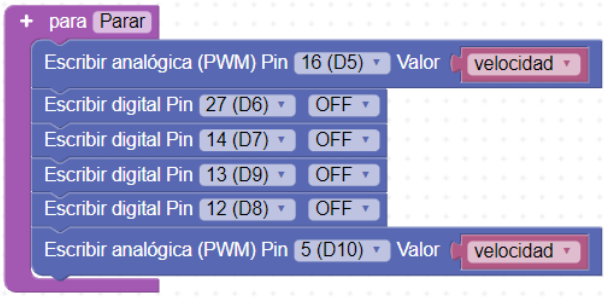
- 0: no hacer nada
- 1: Parar
- 2: Adelante
- 3: Atrás

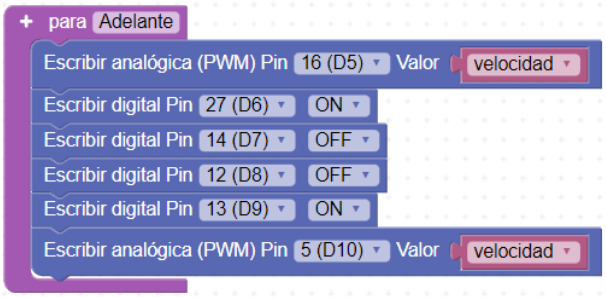
Lo realizaremos mediante funciones. Haremos las funciones básicas, pero se puede completar con más funciones. El cambio de velocidad lo realizará cada vez que carguemos una función.

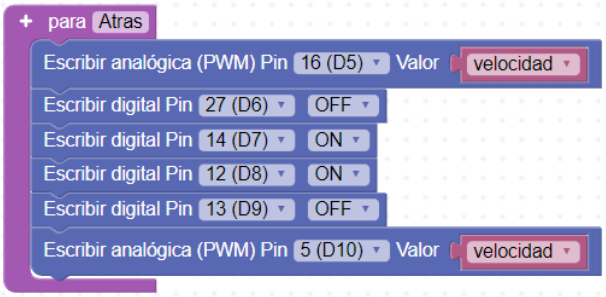
Enlace al programa: ArduinoBlocks Projects\control_motores_consola.abp

El programa resultante será:




para Parar
Escribir analógica (PWM) Pin 16 (D5) Valor velocidad
Escribir digital Pin 27 (D6) OFF
Escribir digital Pin 14 (D7) OFF
Escribir digital Pin 13 (D9) OFF
Escribir digital Pin 12 (D8) OFF
Escribir analógica (PWM) Pin 5 (D10) Valor velocidad


para Adelante
Escribir analógica (PWM) Pin 16 (D5) Valor velocidad
Escribir digital Pin 27 (D6) ON
Escribir digital Pin 14 (D7) OFF
Escribir digital Pin 12 (D8) OFF
Escribir digital Pin 13 (D9) ON
Escribir analógica (PWM) Pin 5 (D10) Valor velocidad


para Atras
Escribir analógica (PWM) Pin 16 (D5) Valor velocidad
Escribir digital Pin 27 (D6) OFF
Escribir digital Pin 14 (D7) ON
Escribir digital Pin 12 (D8) ON
Escribir digital Pin 13 (D9) OFF
Escribir analógica (PWM) Pin 5 (D10) Valor velocidad

8.13 Reto A35. Nivel sonoro con Neopixel.

Reto A35. Nivel sonoro con Neopixel.

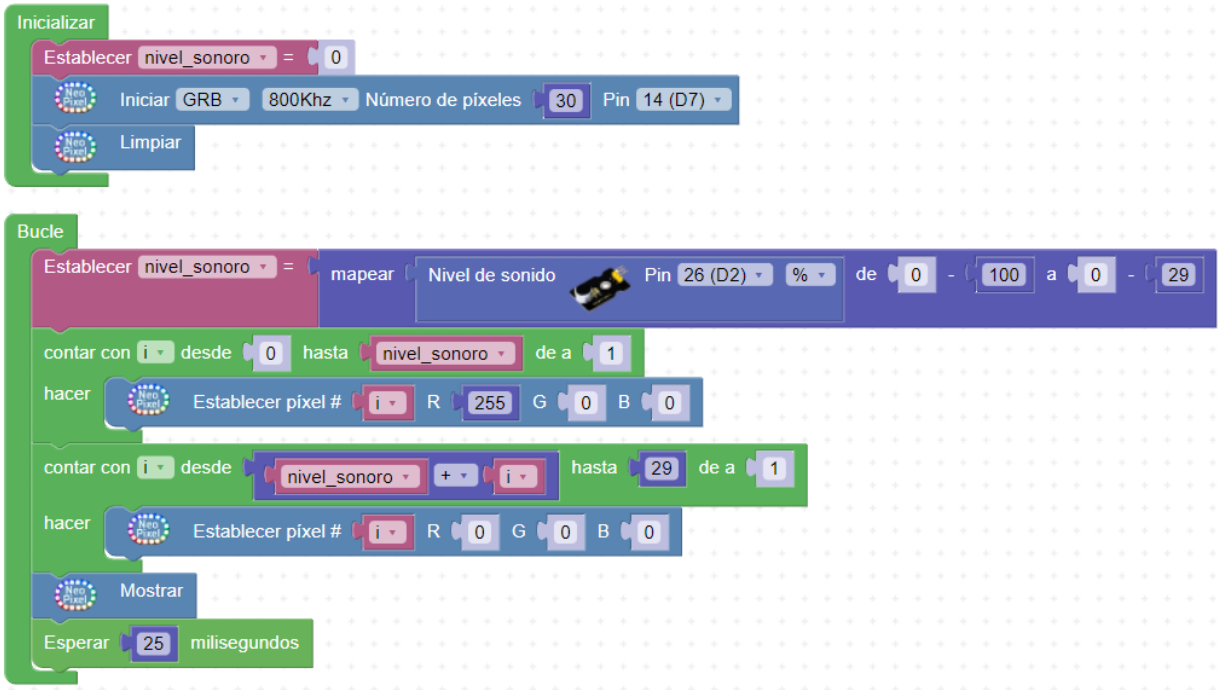
Neopixel es una marca creada por Adafruit Industries para referirse a algunos leds RGB direccionables individualmente, es decir LEDs que cuentan con un circuito lógico integrado dentro de sí mismos, circuito que hace posible controlar con un solo pin digital el color de cada LED en una secuencia de leds encadenados.

En Arduinoblocks, disponemos de bloques específicos para poder programar este tipo de leds.

Enlace al programa: [ArduinoBlocks Projects\nivel_sonoro_neopixel.abp]

Primero tenemos que definir el tipo de Neopixel que vamos a utilizar y donde lo tenemos conectado. Esto lo definiremos en el bloque *Inicializar*. A continuación, en Bucle, podremos comenzar a encender un led del color que queremos.

Vamos a realizar un programa que encienda de color rojo el nivel sonoro y mantenga el resto apagados. Conectaremos la tira de 30 leds a la salida D7 y el micrófono a D2.



8.14 Reto A36. Control de relés.

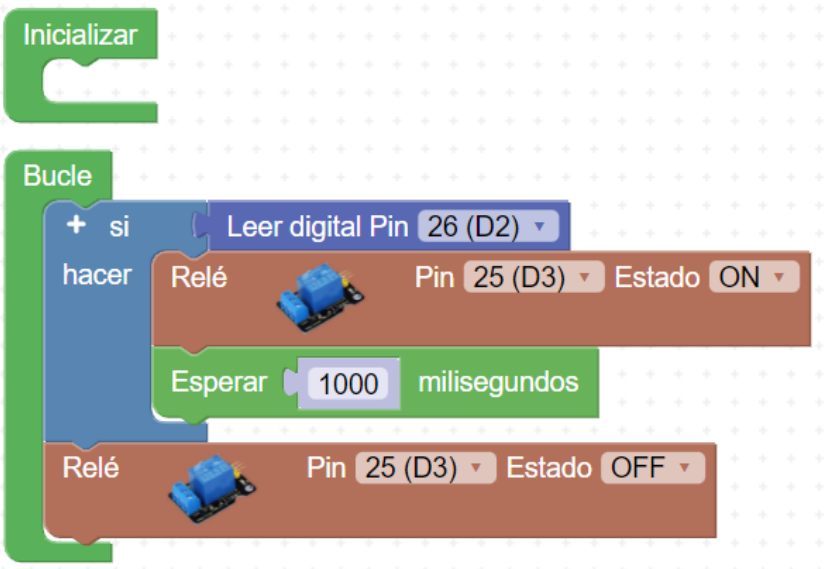
Reto A36. Control de relés.

El relé es un dispositivo electrónico formado por una bobina y unos contactos eléctricos asociados a ella. Nos permite controlar cargas que pueden funcionar a una tensión diferente de nuestra placa y/o para corriente grandes.

Puede ser activado directamente desde una salida. Para realizar la prueba de funcionamiento, vamos a conectar un pulsador en GPIO26 (D2) y un relé en GPIO25 (D3). Cuando apretemos el pulsador el relé se activará y al cabo de un segundo se desactivará.

Enlace al programa: ArduinoBlocks Projects\rele.abp

El programa que vamos a realizar es el siguiente:



8.15 Reto A37. Panel táctil y Neopixel.

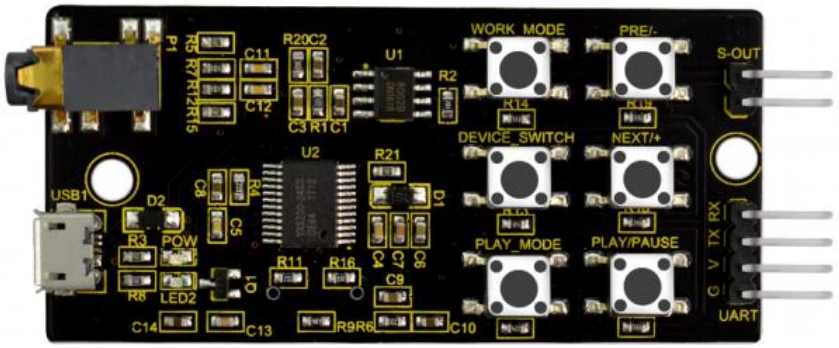
Reto A37. Panel táctil y Neopixel.

Existe

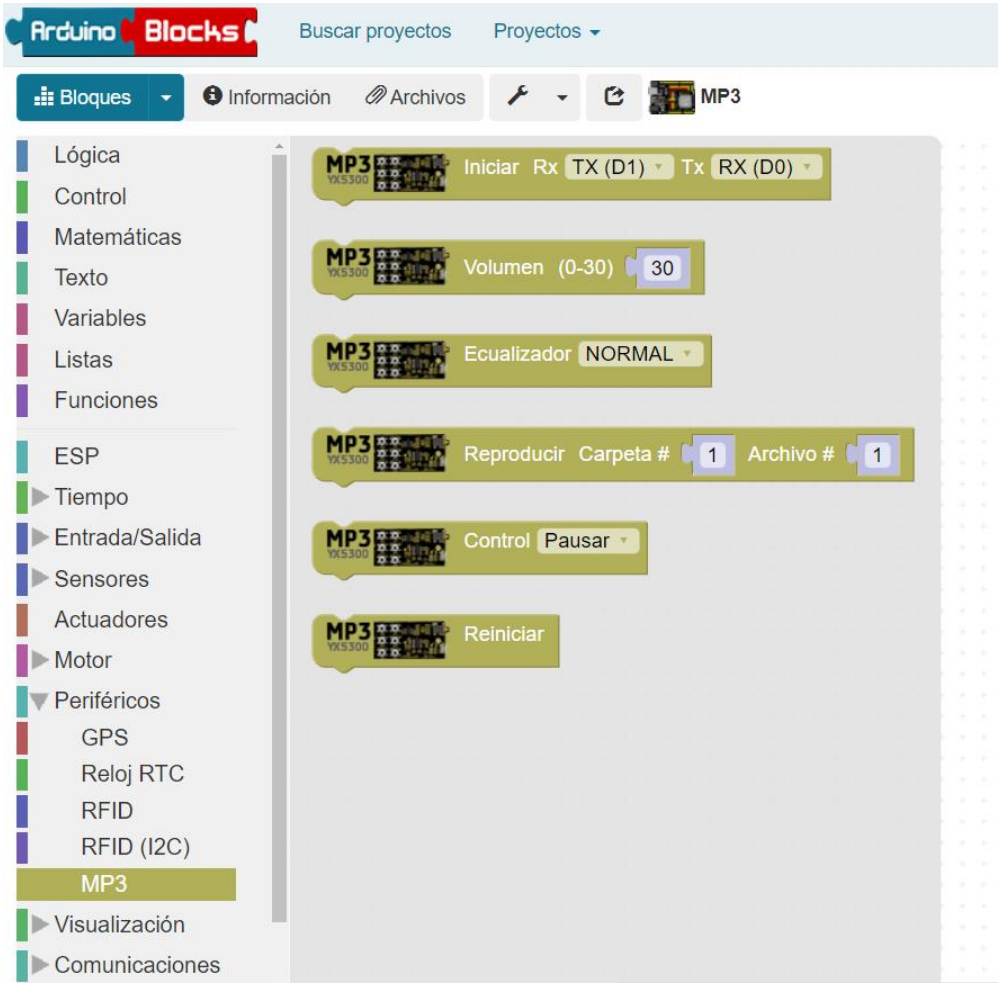
8.16 Reto A40. Reproducir música con MP3 (YX5200-24SS).

Reto A40. Reproducir música con MP3 (YX5200-24SS).

Existe en el mercado un reproductor de MP3 de Keyestudio que podemos controlar desde la placa ***ESP32 Plus STEAMakers*** y lo podemos programar con ArduinoBlocks.



Los bloques para programar el módulo MP3 están dentro de la categoría de *Periféricos*.



9 Proyectos KIT: Imagina TDR STEAM + extras.

A continuación, os detallamos una serie de proyectos que se pueden realizar con el kit formado por:

- ESP32 Plus STEAMakers.
- Imagina TDR STEAM.
- Micrófono.
- Pantalla LCD.
- Cable de tres hilos para conexión del micrófono.
- Cables de cuatro hilos para la conexión de la pantalla LCD.
- Cable USB.
- RFID I2C.

9.1 Proyecto propuesto 1: Alarma doméstica.

En este proyecto, vamos a realizar un sistema de alarma formado por la placa ***ESP32 PLUS STEAMakers*** y la placa ***Imagina TDR STEAM*** con el kit de ampliación formado por un sensor de sonido, una pantalla LCD y un mando a distancia.

A continuación, detallaremos las funcionalidades de la alarma.

1. Activaremos (armar) la alarma con la tecla * del mando a distancia para que empiece a funcionar.
2. Con la tecla # desactivaremos (desarmaremos) la alarma.
3. Con la tecla OK pararemos la alarma en el caso que salte por la detección de algún sensor.
4. Cuando se arme la alarma se encenderá el led azul y si está desarmada se apagará el led azul.
5. Cuando suene la alarma se encenderá el led rojo y sonará el zumbador.
6. Simularemos la detección de personas (sensores PIR) con los pulsadores SW1 i SW2.
7. Detectaremos si hay un cambio brusco de la iluminación (+/-10%) mediante la LDR.
8. Detectaremos si se produce un ruido fuerte (+30%) con el micrófono.
9. Los colores del led RGB indicarán el tipo de alarma:
    a. Rojo: Pulsador 1
    b. Verde: Pulsador 2
    c. Azul: Sonido
    d. Blanco: Luz
10. En la pantalla LCD se mostrará el estado de la alarma:
    a. Línea 0: Activada/Desactivada.
    b. Línea 1: si ha saltado o no.

Enlace al programa: ArduinoBlocks Projects\alarma_domestica.abp

Explicación del programa:

En este apartado, vamos a explicar cómo está programado nuestro sistema de alarma. Definiremos los bloques de Inicializar y bucle. A continuación, definiremos todas las funciones que se deben crear.

Inicializar

- Configuración pantalla LCD.
- Llamada a la función ***variables.***
- Llamada a la función ***pantalla_inicio.***
- Llamada a la función ***alarma_desactivada.***
- Apagar led RGB (color negro).

Bucle

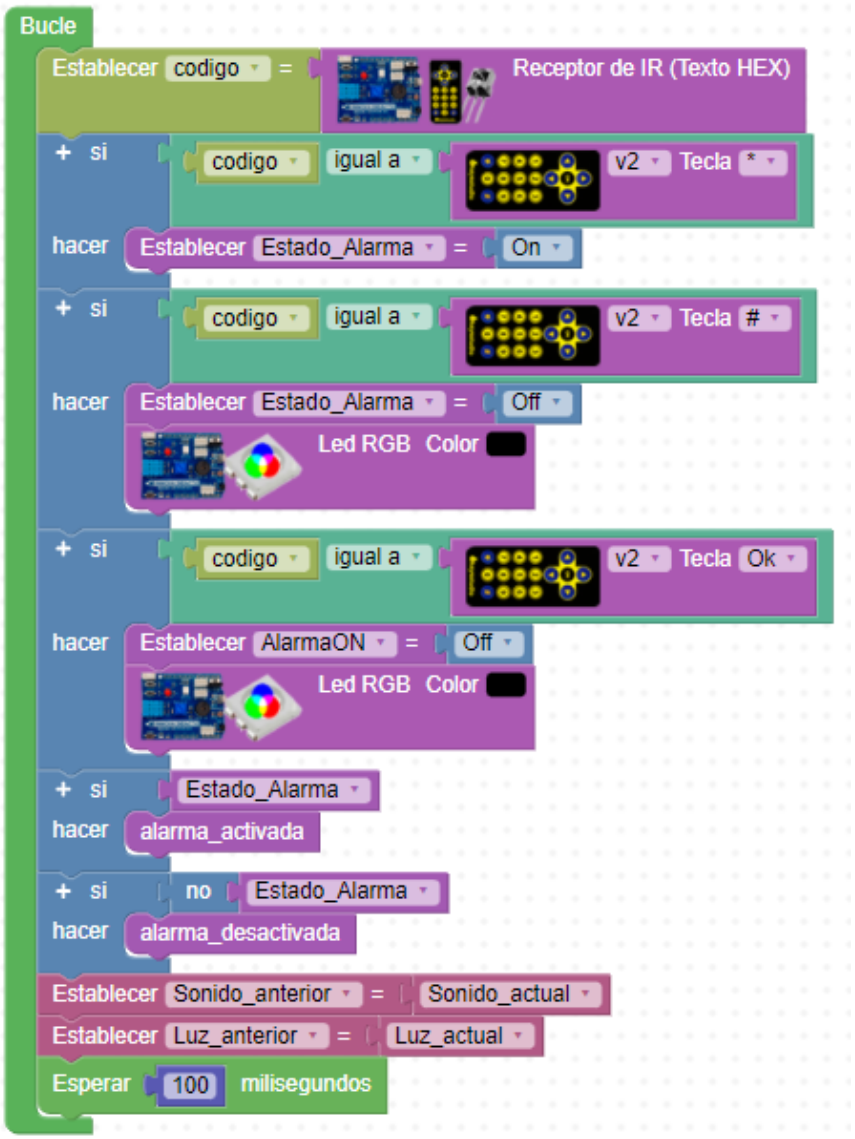
- Leer código del mando a distancia: *Codigo*
- Condiciones de las teclas pulsadas del mando a distancia:
  - *****: alarma encendida (Alarma ON) → *Estado_Alarma* a ON
  - **#**: alarma apagada (Alarma OFF) → *Estado_Alarma* a OFF
  - **OK**: para alarma → *AlarmaON* OFF
- Si *Estado_Alarma* ON → ***alarma_activada***
- Si *Estado_Alarma* OFF → ***alarma_desactivada***
- Actualizar los valores de sonido anterior con el valor actual:

  Sonido_anterior = *Sonido_actual*

- Actualizar los valores de luz anterior con el valor actual:

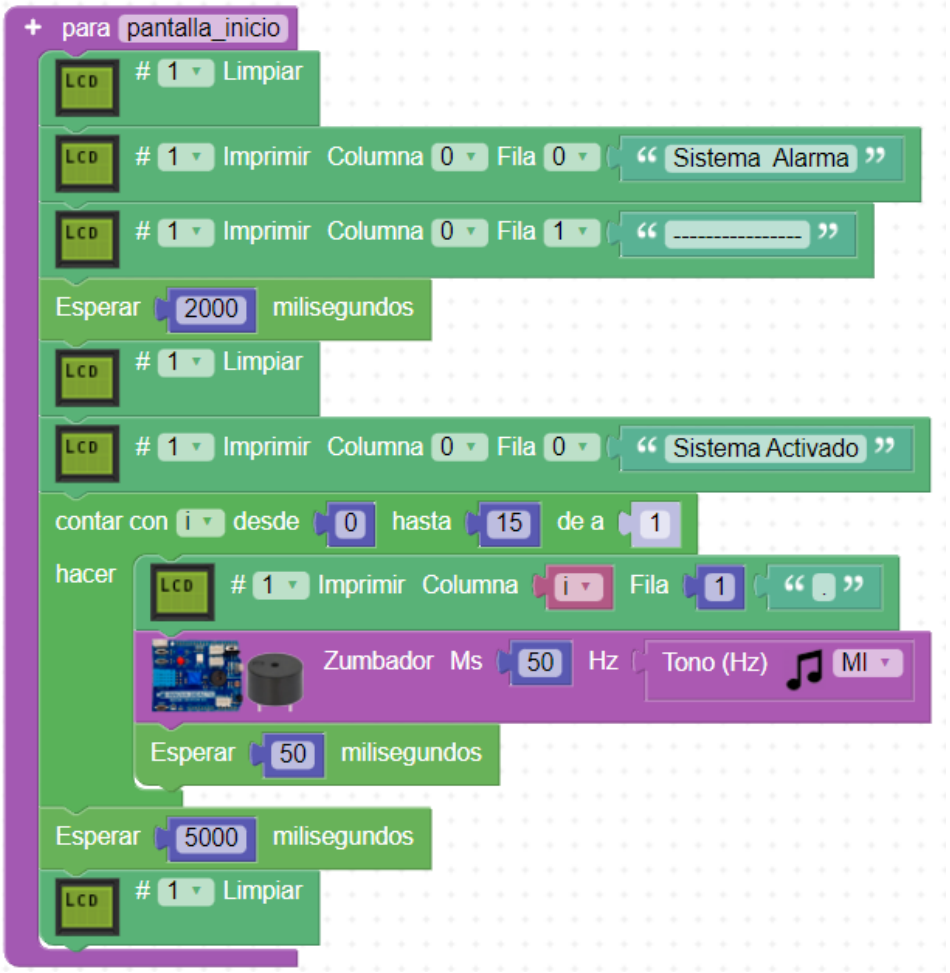
  Luz anterior = *Luz_actual*

- Espera de 100ms

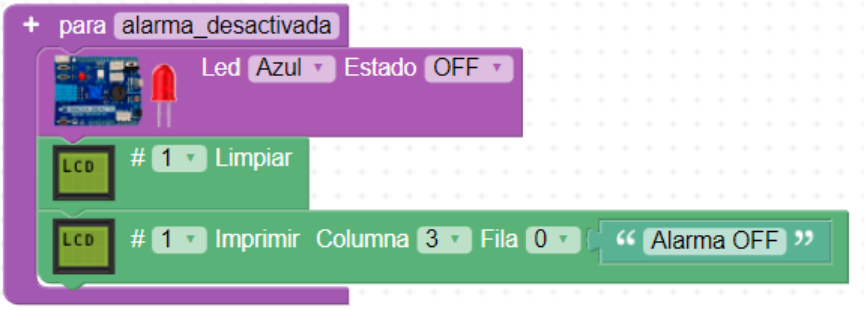


- Función: ***variables***
  - Numéricas:
    - *Luz_actual*: valor actual de la LDR.
    - *Luz anterior*: valor anterior de la LDR.
    - *Sonido_actual*: valor actual del micrófono.
    - *Sonido_anterior*: valor anterior del micrófono.
  - Texto:
    - *Codigo*: valor que envía el mando a distancia de infrarrojos.
  - Booleanas:
    - *Estado_Alarma*: si la alarma está activada (ON) o desactivada (OFF).
    - *AlarmaON*: enclavamiento de la alarma para poder pararla cuando salte pero que se mantenga activada.
    - *Pulsador_1*: estado del pulsador SW1.
    - *Pulsador_2*: estado del pulsador SW2.

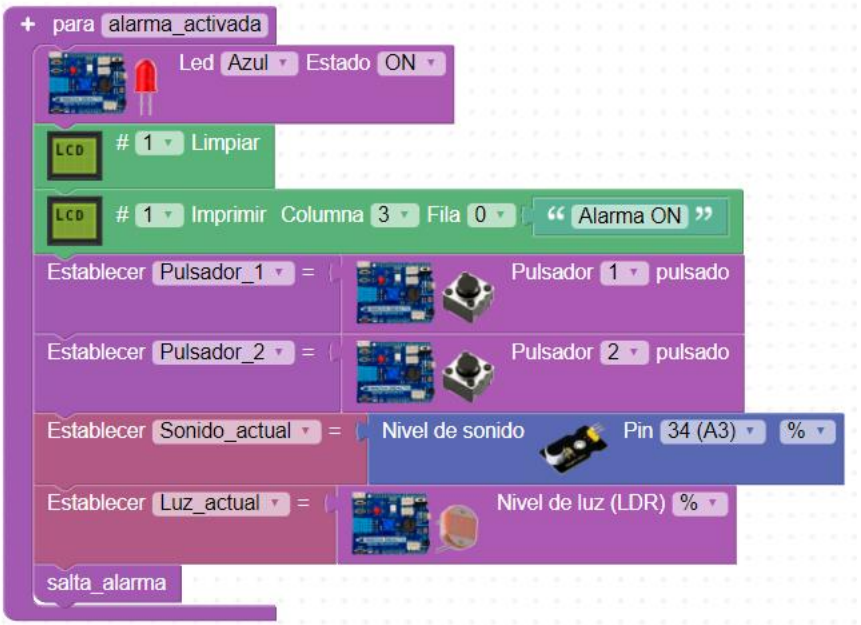
- Función: ***pantalla_inicio***
  - Limpiar LCD.
  - Texto: Sistema Alarma
  - Texto: ----------
  - Esperar 2 segundo.
  - Limpiar LCD.
  - Texto: Sistema en marcha.
  - Texto: Animación de texto: van saliendo los puntos (cada 50ms) y cada vez que aparece uno suena la nota *Mi* durante 50ms por el zumbador.
  - Espera 1 segundo.



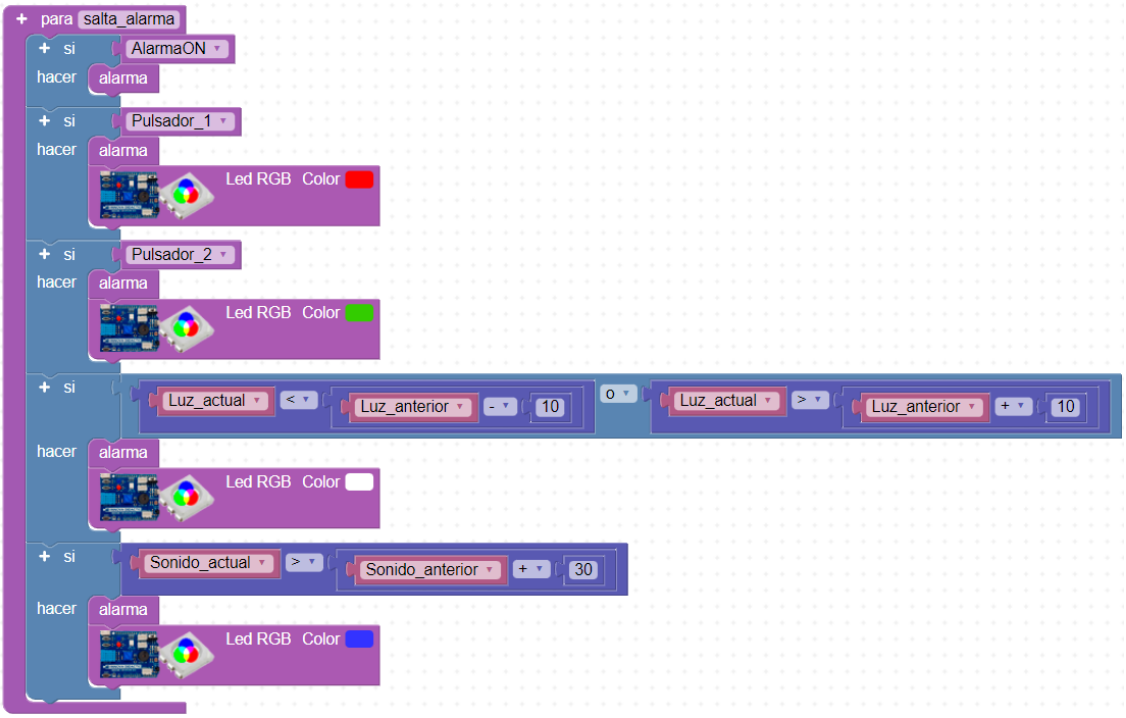
- Función: ***alarma_desactivada***
  - Apagar led Azul.
  - Limpiar LCD.
  - Texto: Alarma OFF



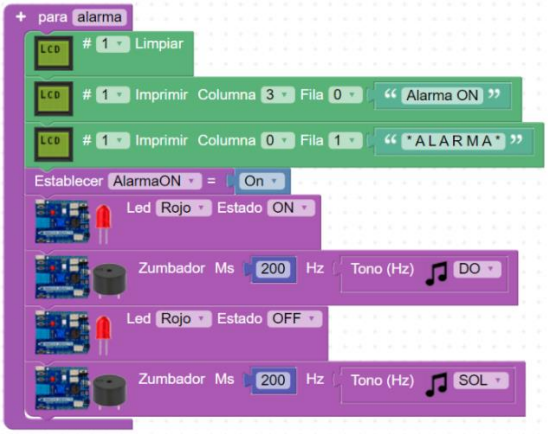
- Función: ***alarma_activada***
  - Encender led Azul.
  - Limpiar LCD.
  - Texto: Alarma OFF
  - Leer sensores:
    - Pulsador 1: SW1
    - Pulsador 2: SW2
    - Micrófono
    - Sensor de luz LDR
  - Llamada a la función ***salta_alarma***



- Función: ***salta_alarma***
  - Definimos las condiciones para que salte la alarma:
    - *AlarmaON* a ON (enclavamiento).
    - *Pulsador_1*: apretado. Enciende el led RGB a rojo.
    - *Pulsador_2*: apretado. Enciende el led RGB a verde.
    - *Luz_anterior*-10<*Luz_actual*<*Luz_anterior*+10. Enciende el led RGB a blanco.
    - *Sonido_actual*<*Sonido_anterior*+30. Enciende el led RGB a azul.



- Función: ***alarma***
  - Limpiar LCD.
  - Texto: Alarma ON
  - Texto A L A R M A
  - Poner la variable de enclavamiento *AlarmaON* a ON.
  - Parpadeo del led Rojo y sonidos Do-Sol por el zumbador.



9.2 Proyecto propuesto 2: Control de un parking.

En este proyecto, vamos a realizar un sistema de control de un parking formado por la placa ***ESP32 Plus STEAMakers*** y la placa ***Imagina TDR STEAM*** con el kit de ampliación formado por un sensor de sonido, una pantalla LCD y un mando a distancia.

A continuación, detallaremos las funcionalidades de la alarma.

1. Simularemos la detección de entrada de un coche con el pulsador SW1.
2. Simularemos la detección de salida de un coche con el pulsador SW2.
3. El led azul indicará que el parking no está completo.
4. El led rojo indicará que el parking está completo.
5. En la pantalla LCD indicará si entra o sale un coche y el número de plazas libres que hay en el parking.
6. Para indicar que sube la barrera utilizaremos el zumbador y el led RGB en color azul.
7. Para indicar que baja la barrera utilizaremos el zumbador y el led RGB en color rojo.

Enlace al programa: ArduinoBlocks Projects\proyecto_parking.abp

Explicación del programa:

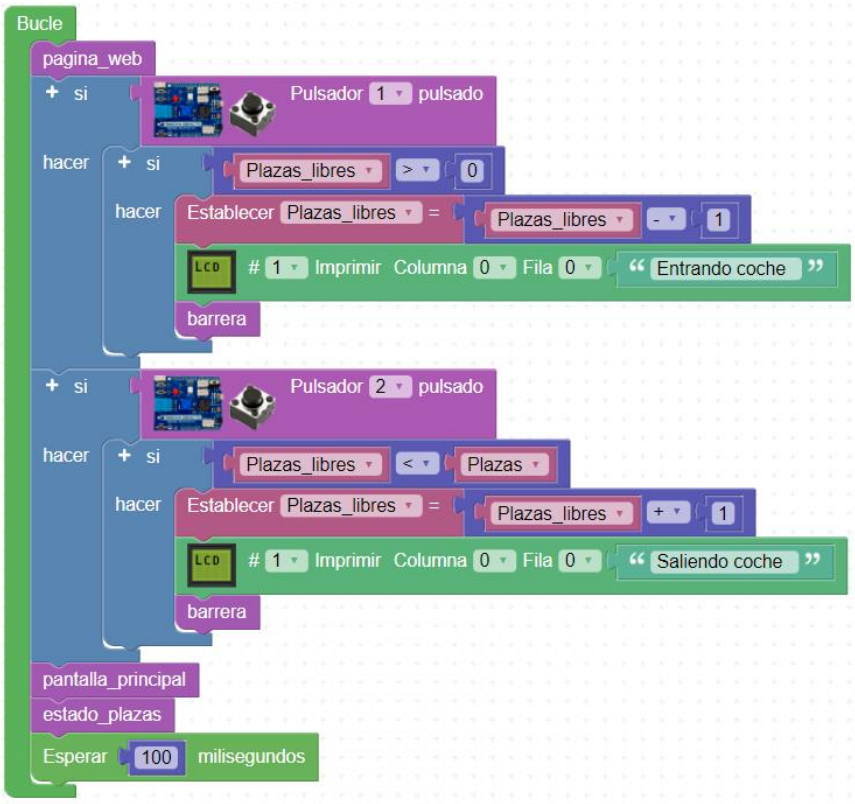
En este apartado, vamos a explicar cómo está programado nuestro sistema de control del parking. Definiremos los bloques de Inicializar y bucle. A continuación, definiremos todas las funciones que se deben crear.

Inicializar

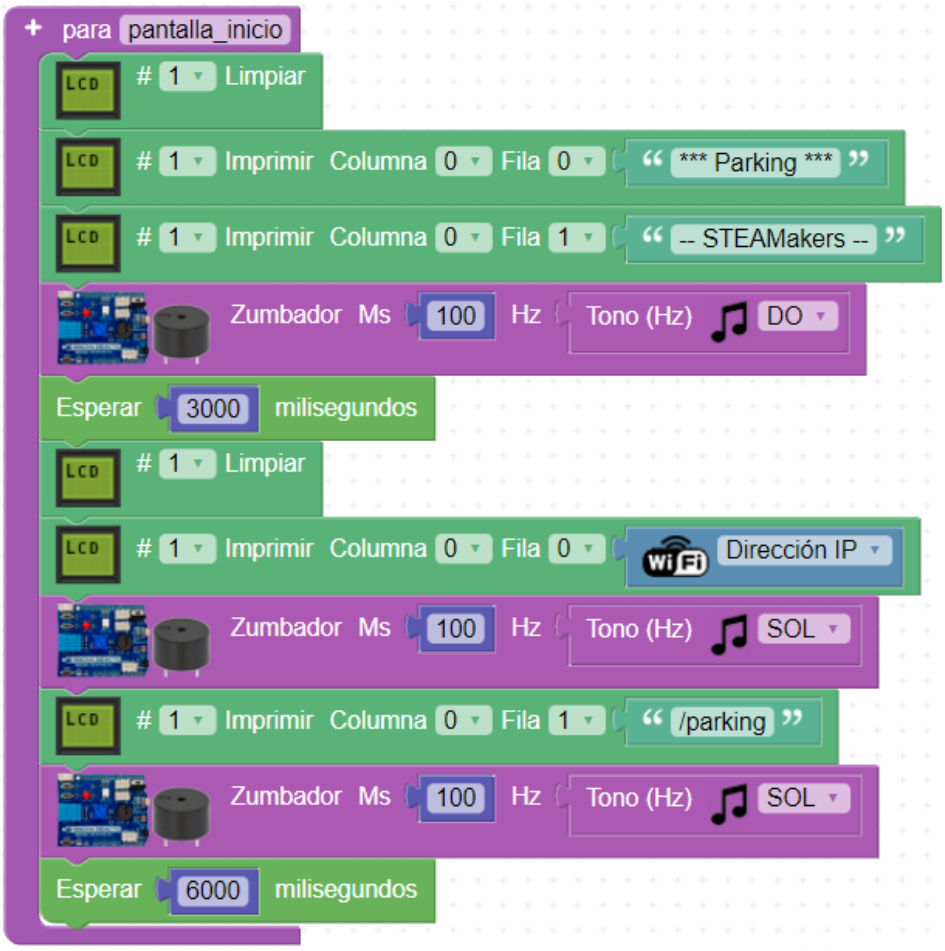
- Configuración de la red Wifi.
- Configuración del servidor Web.
- Configuración pantalla LCD.
- Apagar led RGB (color negro).
- Creamos las dos variables para el total de plazas y las plazas libres que hay en el parking *Plazas* y *Plazas_libres*.
- Llamada a la función ***pantalla_inicio.***

Bucle

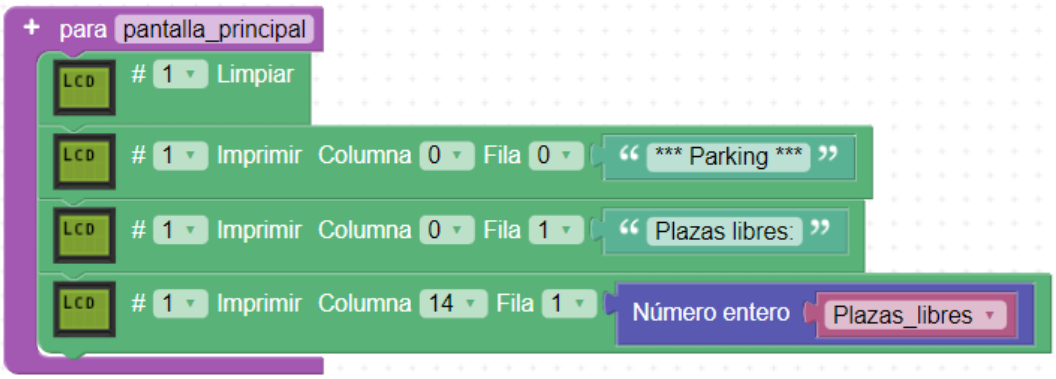
- Generar la página web con la llamada a la función ***pagina_web***.
- Condiciones de los pulsadores que indican si entra o sale un coche:
  - **Pulsador 1**: Entra un coche
    - Incrementamos la variable *Contador* en una unidad.
    - Si la cantidad de *Contador* es menor o igual al número de *Plazas*:
      - Mostramos el texto: Entrando coche
      - Calculamos el número de plazas libres *Plazas_libres*.
      - Simulamos el funcionamiento de la barrera con la función ***barrera***.
  - **Pulsador 2**: Sale un coche
    - Decrementamos la variable *Contador* en una unidad.
    - Si la cantidad de *Contador* es mayor de 0:
      - Mostramos el texto: Saliendo coche
      - Calculamos el número de plazas libres *Plazas_libres*.
      - Simulamos el funcionamiento de la barrera con la función ***barrera***.
- Llamamos a la función ***pantalla_principal*** para mostrar datos por la pantalla LCD.
- Llamamos a la función ***estado_plazas*** para indicar si el parking está lleno o vacío y la ocupación de plazas con el led RGB (código de colores).
- Espera de 100ms.



- Función: ***pantalla_inicio***
  - Limpiar la pantalla LCD.
  - Mostrar el texto: *** Parking ***
  - Mostrar el texto: -- STEAMakers --
  - Hacer una nota (DO).
  - Esperar 3 segundos.
  - Limpiar la pantalla LCD.
  - Mostrar en la pantalla LCD la dirección IP asignada.
  - Hacer una nota (SOL).
  - Mostrar el texto: /parking (que será el texto que hay en la solicitud GET).
  - Hacer una nota (SOL).
  - Esperar 6 segundos.

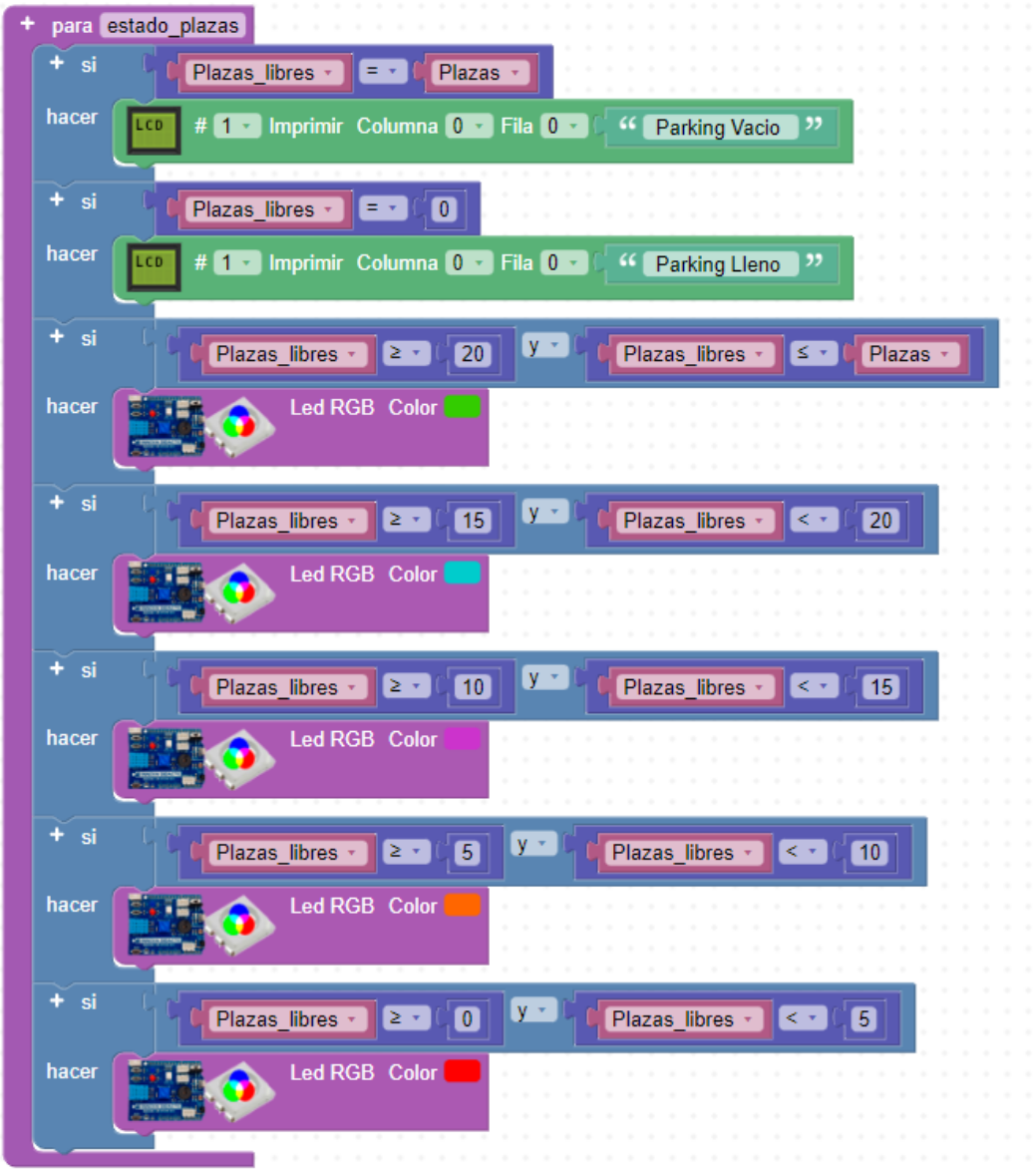


- Función: ***pantalla_principal***
  - Limpiar la pantalla LCD.
  - Mostrar el texto: *** Parking ***
  - Mostrar el texto: Plazas libres:
  - Mostrar el valor de la variable *Plazas_libres*.

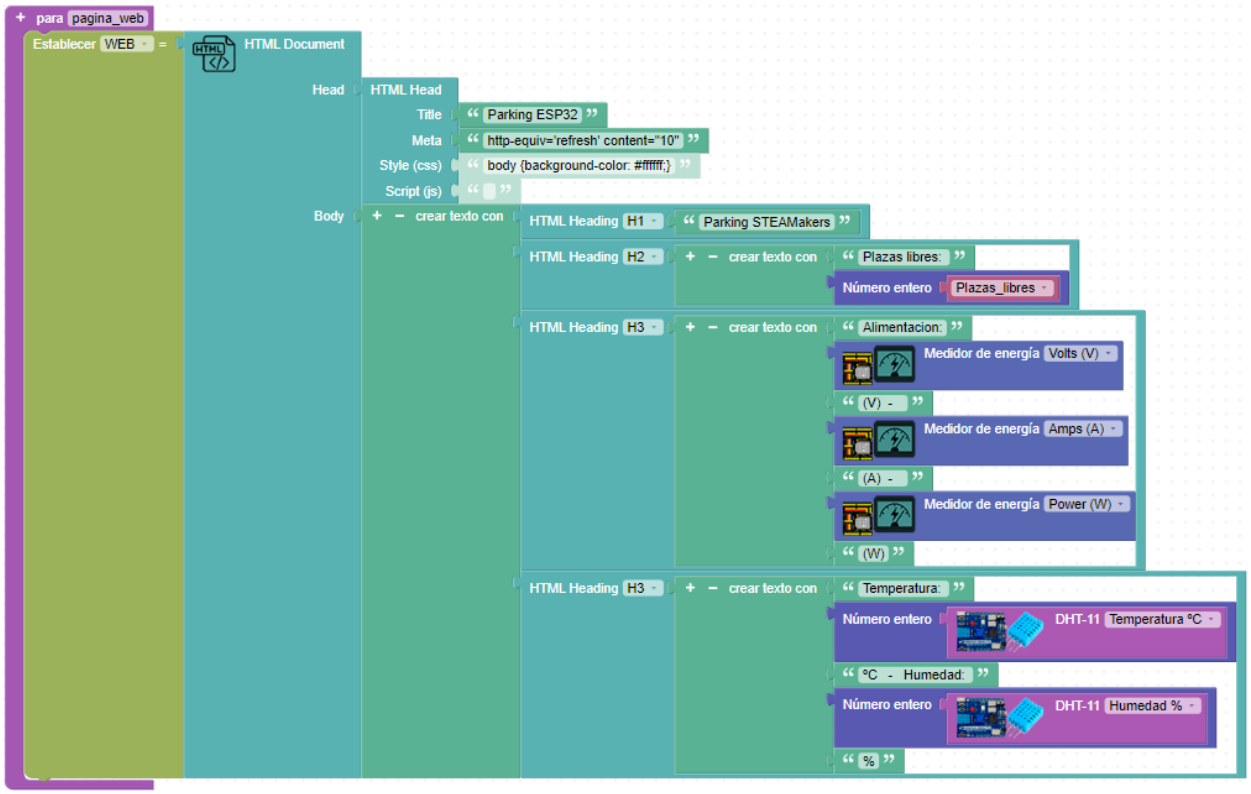


- Función: ***barrera***
  - Encender el led azul.
  - Repetir 15 veces la nota LA con una espera de 50ms entre notas.
  - Apagar el led azul.

- Función: ***estado_plazas***
  - Si todas las plazas están libres mostrará el texto: Parking vacío
  - Si todas las plazas están ocupadas mostrará el texto: Parking lleno
  - En función del número de plazas libres el led RGB mostrará un color diferente.



- Función: ***pagina_web***
  - Crearemos una variable de tipo texto a la que llamaremos *WEB* y donde estará almacenada nuestra página web.
  - Definimos la estructura de la página web y mostraremos los siguientes datos:
    - El número de plazas libres.
    - La tensión, intensidad y potencia del sistema de control.
    - La temperatura y humedad del parking.



- Solicitud: ***get/parking***

9.3 Proyecto de ejemplo 1: Control web.

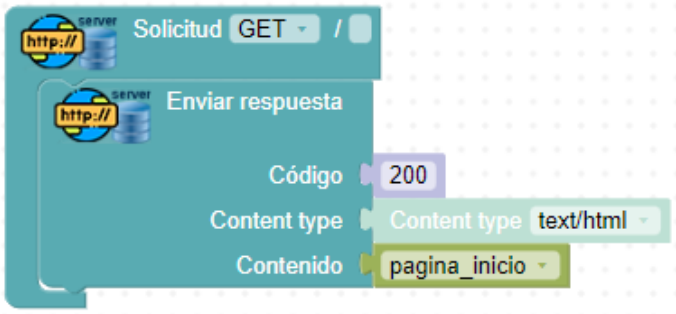
Este proyecto basado en el proyecto de ***Luis Notario*** vamos a realizar el control de la placa ***Imagina TDR STEAM*** a través de una página web. Controlaremos los siguientes elementos:

- Led rojo (ON/OFF).
- Led azul (ON/OFF).
- Led RGB.
- Sensor de temperatura y humedad DHT 11.
- Consumo de energía.

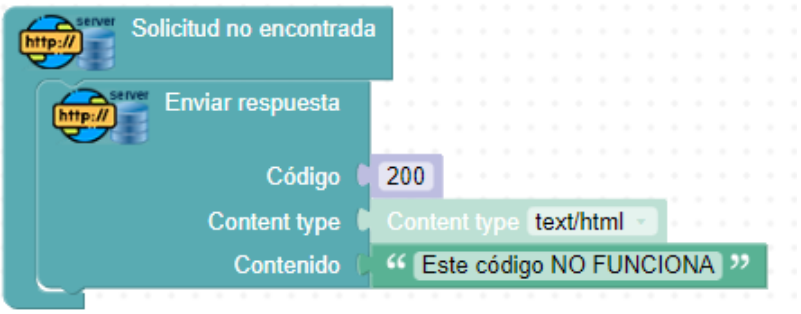
También mostrará la página de inicio y un mensaje de error cuando no se realiza correctamente la petición de página web.

Enlace al programa: ArduinoBlocks Projects\control_web.abp

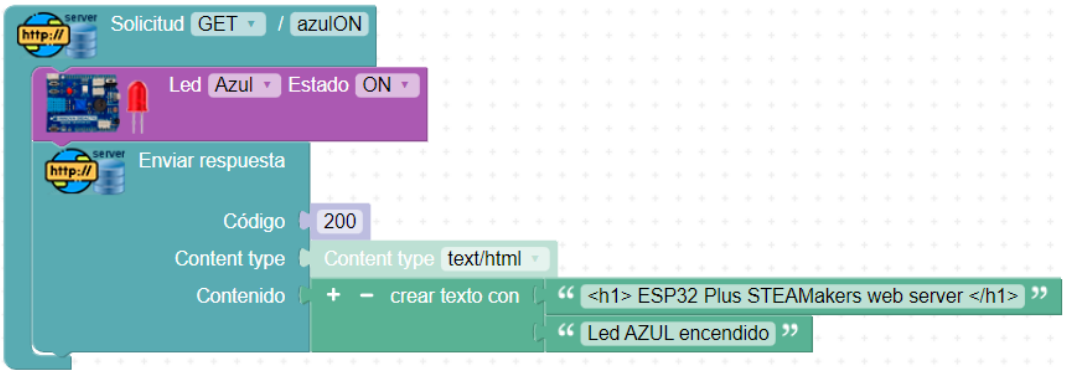
- Petición de página de inicio.

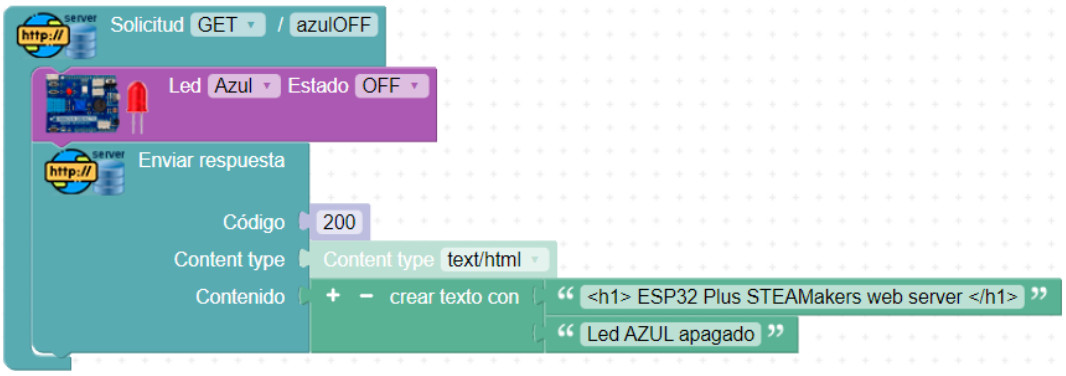


- Petición de página no encontrada (error en la petición).

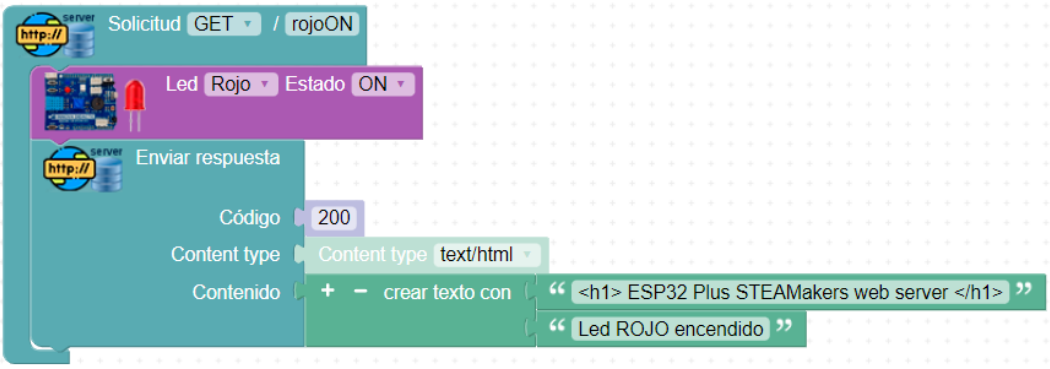


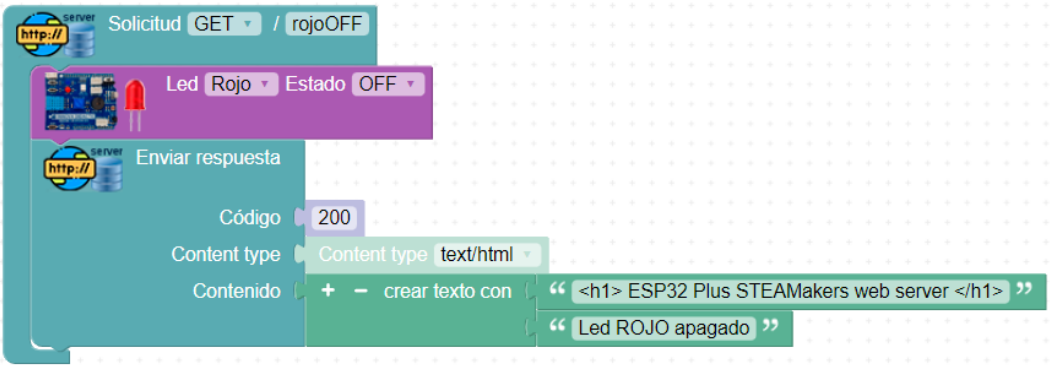
- Petición para encender y apagar el led azul.



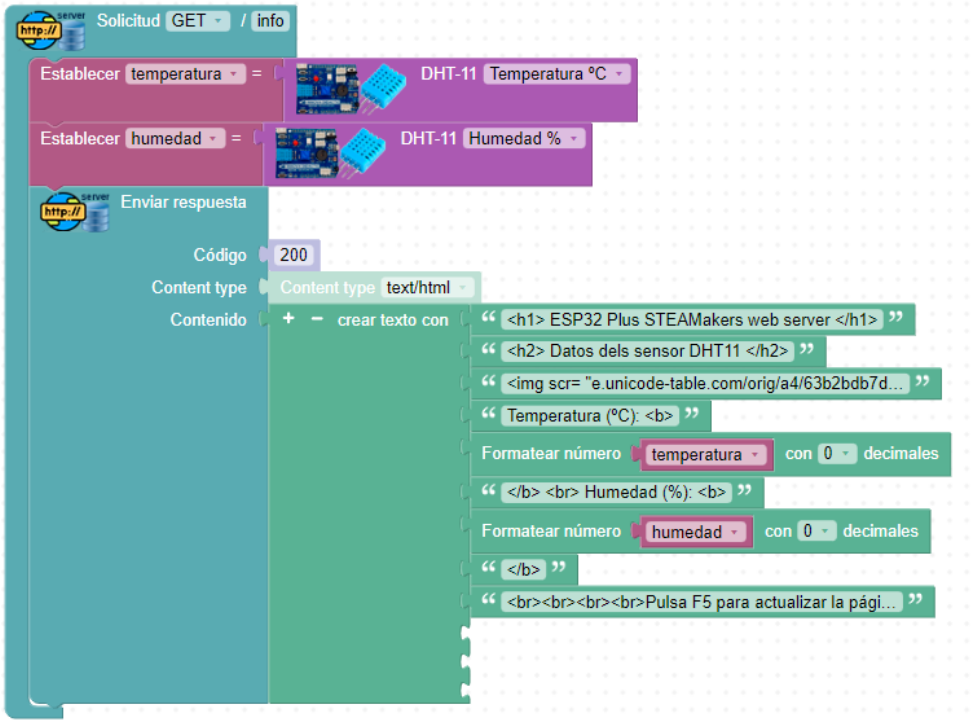


- Petición para encender y apagar el led azul.

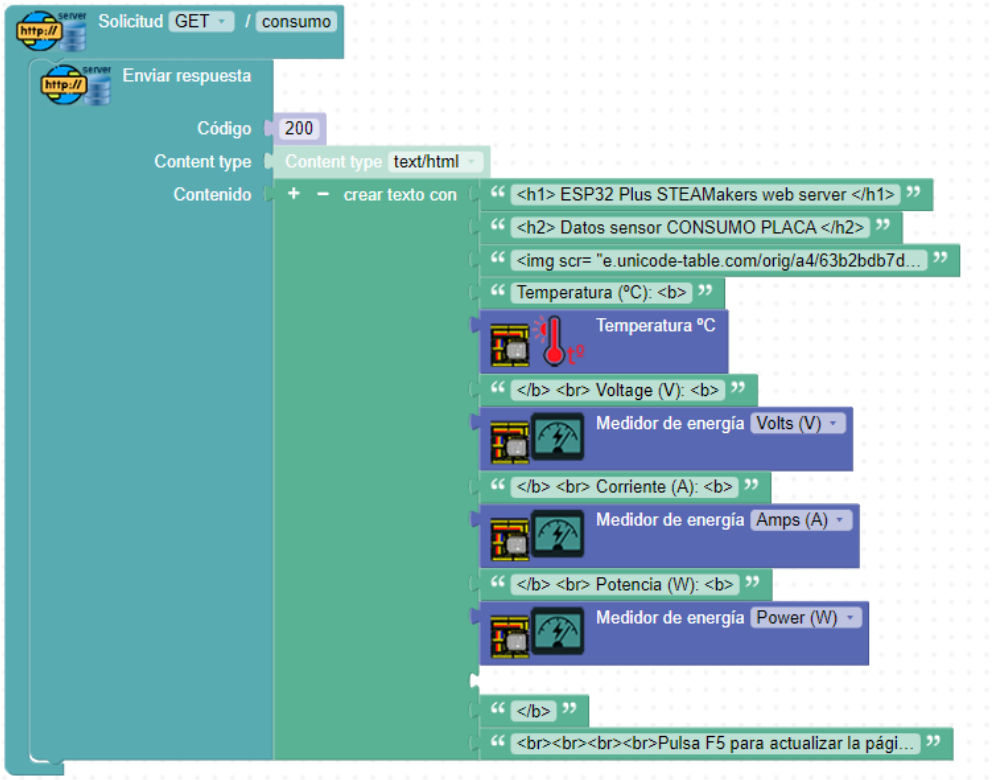




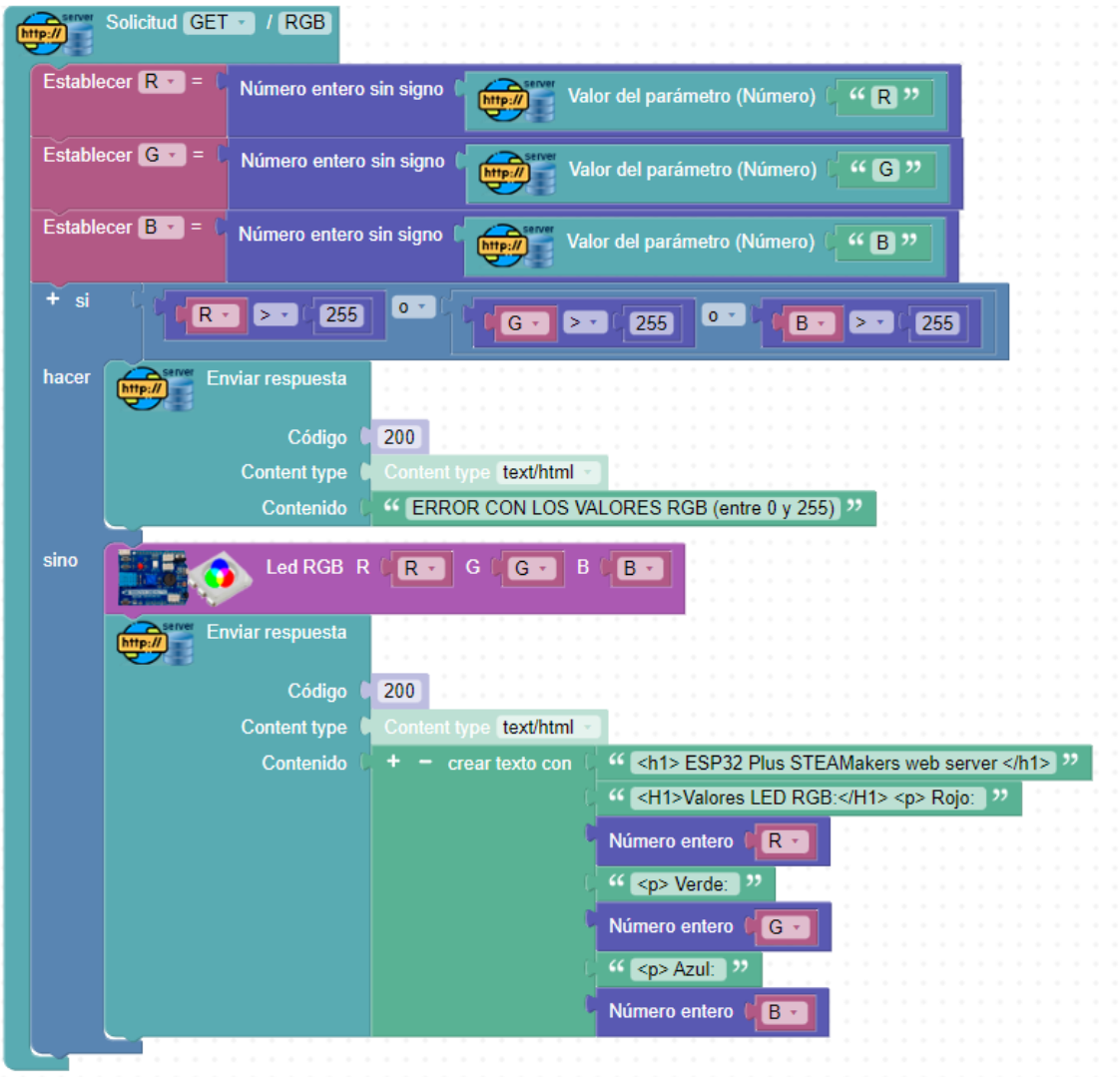
- Petición para obtener información de la temperatura y humedad mediante el sensor DHT11.



- Petición del consumo.

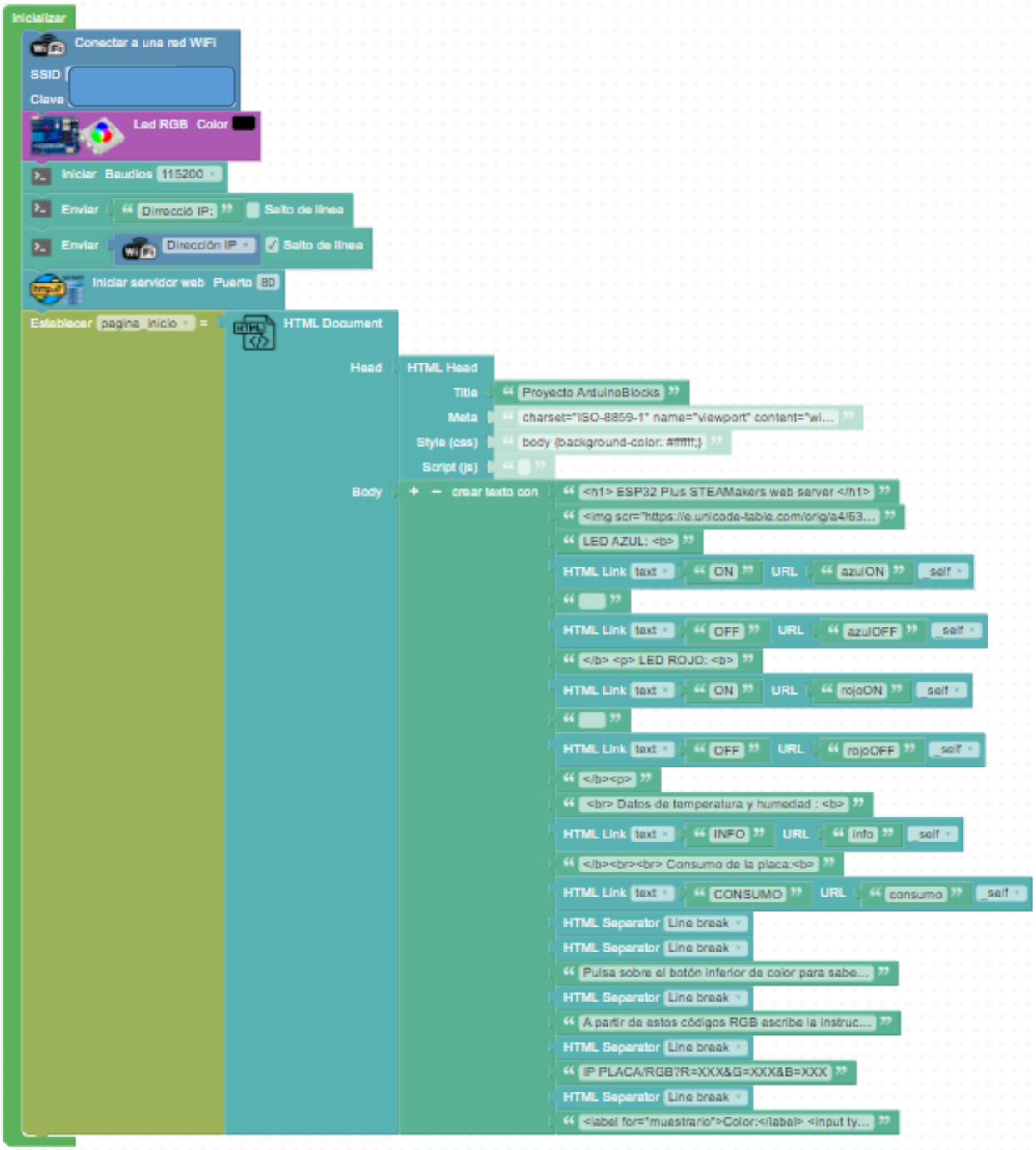


- Petición para ajustar el color del led RGB.



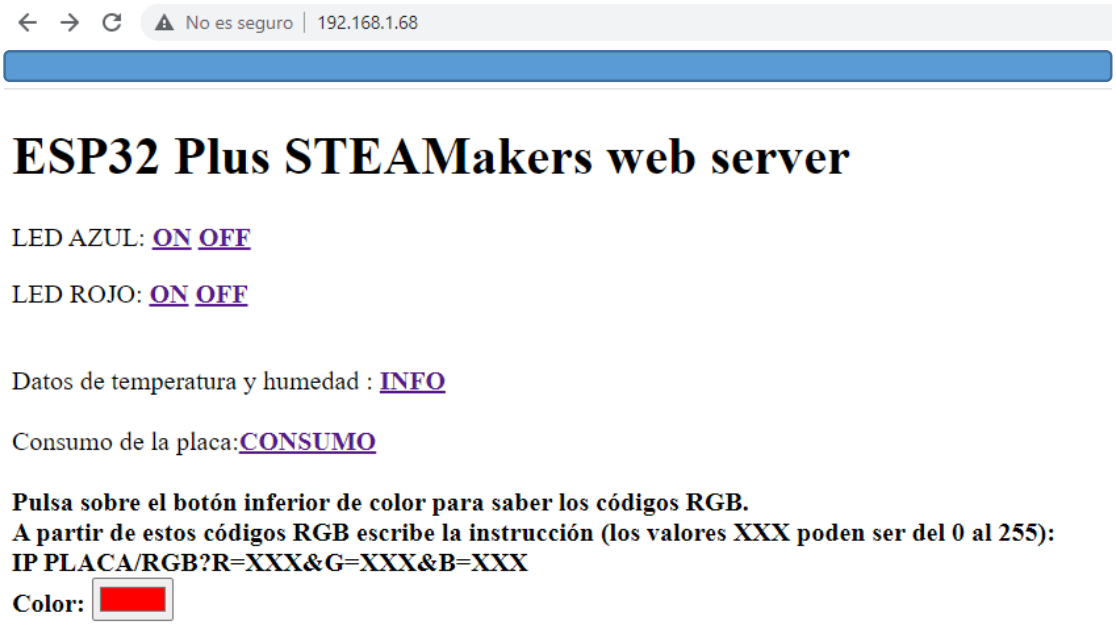
- Programa principal.

- Programa de inicialización.



A continuación, podemos observar las diferentes páginas que van apareciendo en el navegador.

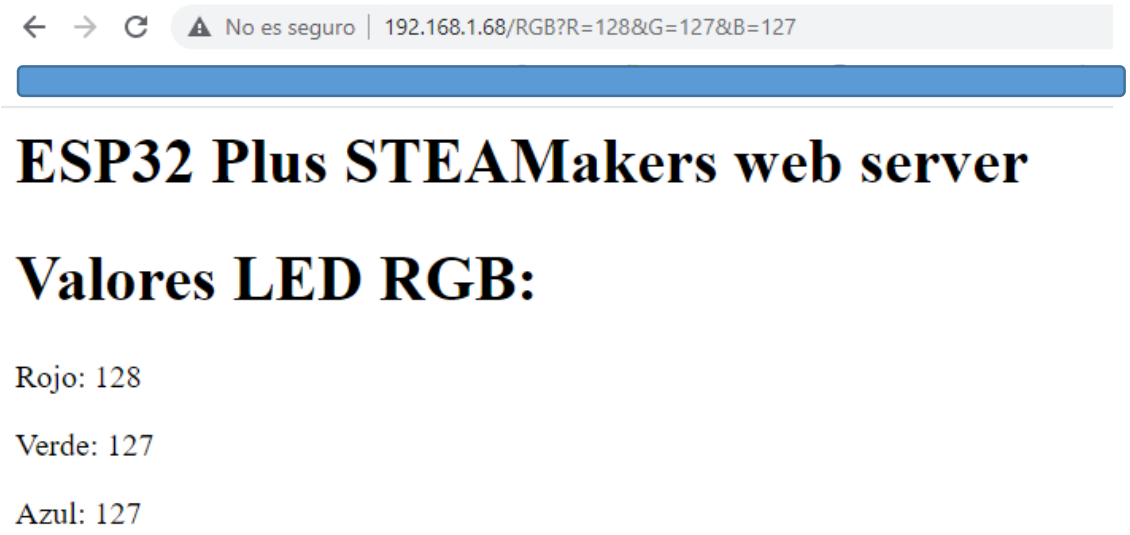
- Pantalla de inicio:



- Pantalla donde se muestra el estado de los leds:

- Pantalla:

- Pantalla para establecer los valores del led RGB:



9.4 Proyecto de ejemplo 2: Control de asistencia con RFID.

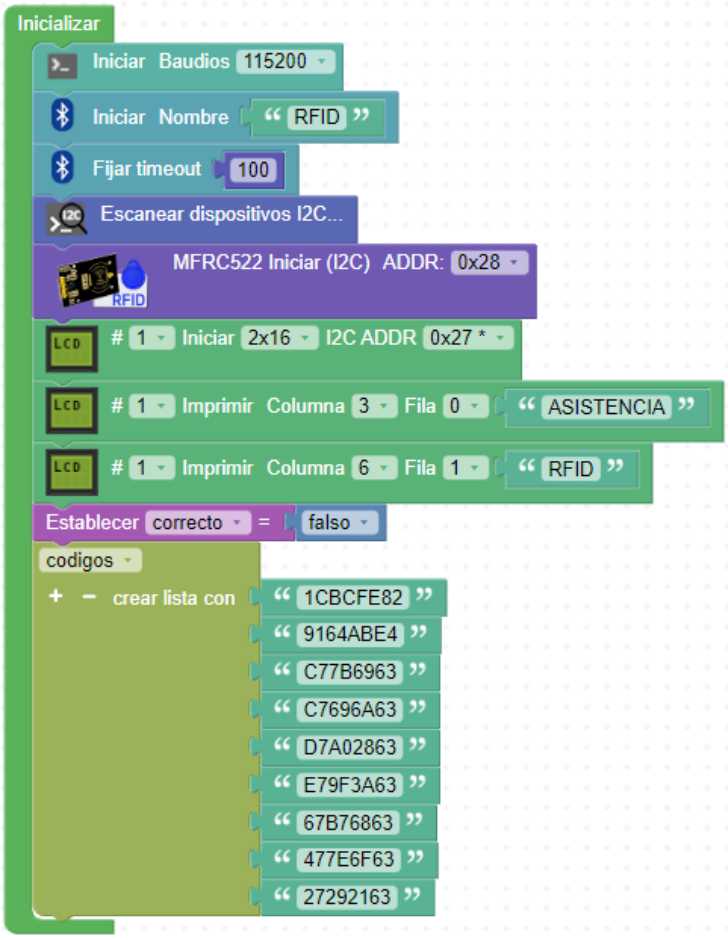
Este proyecto basado en el proyecto de ***Toni Hortal*** vamos a realizar el control de asistencia mediante tarjetas RFID. Iremos identificando los alumnos/as que van accediendo al centro y nos lo irá mostrando en la aplicación de teléfono móvil a través de bluetooth.

Archivos:

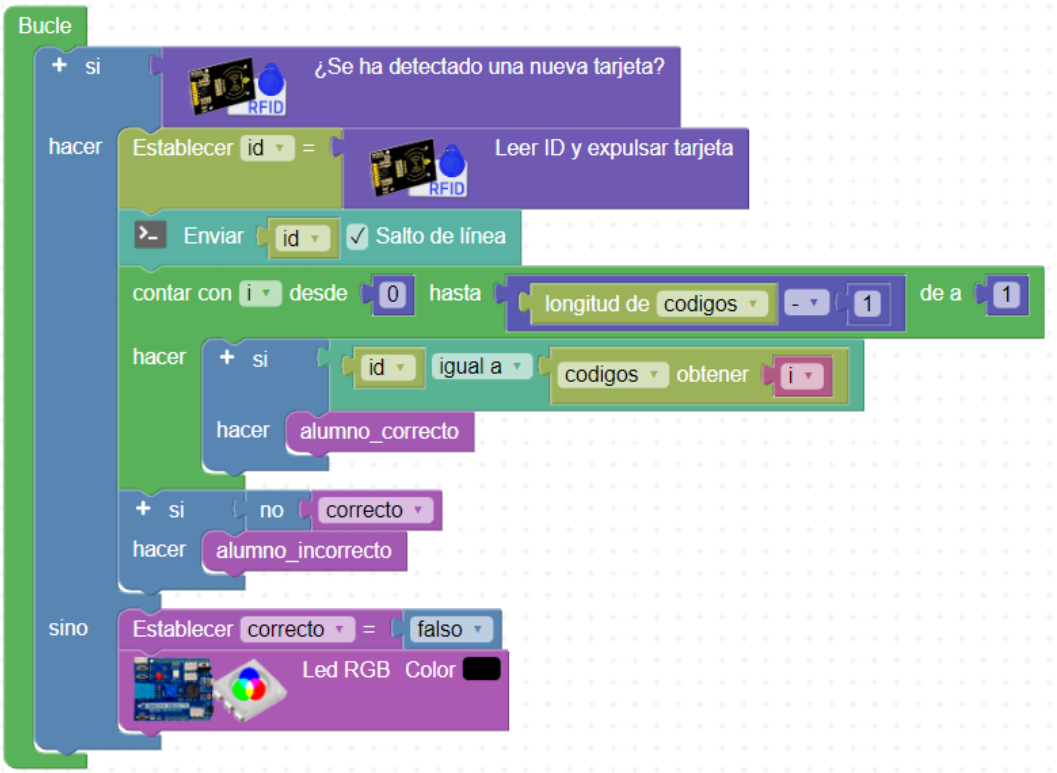
- Arduino_Blocks: ArduinoBlocks Projects\control_asistencia_rfid.abp
- AppInventor2: AppInventor2\RFID.aia

A continuación, se muestra el código de los diferentes programas del proyecto.

- Inicialización:

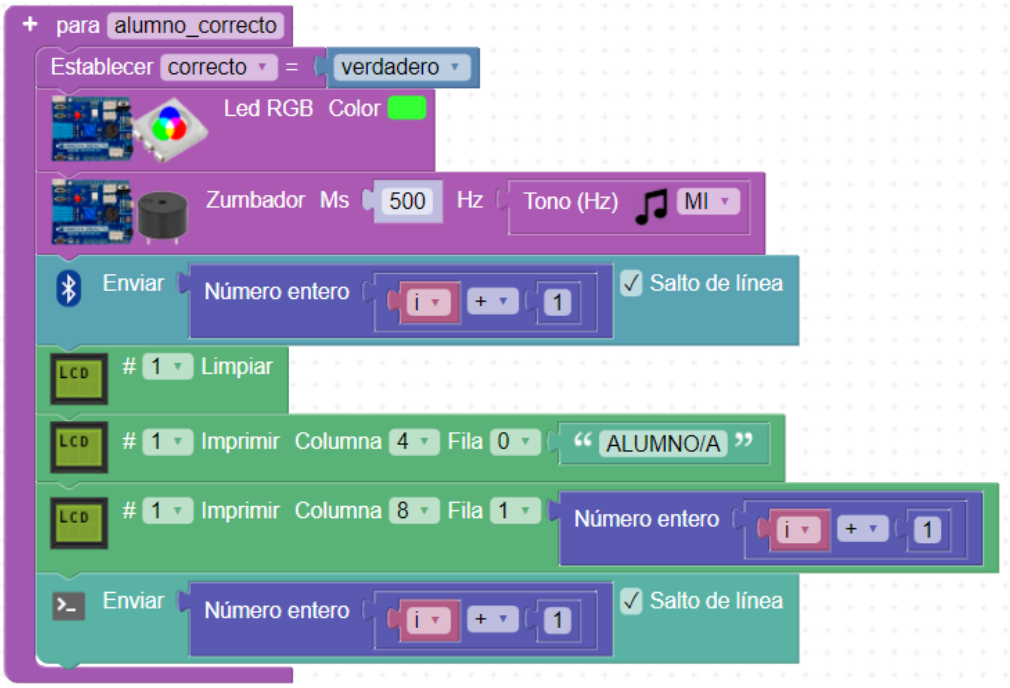


- Programa principal:



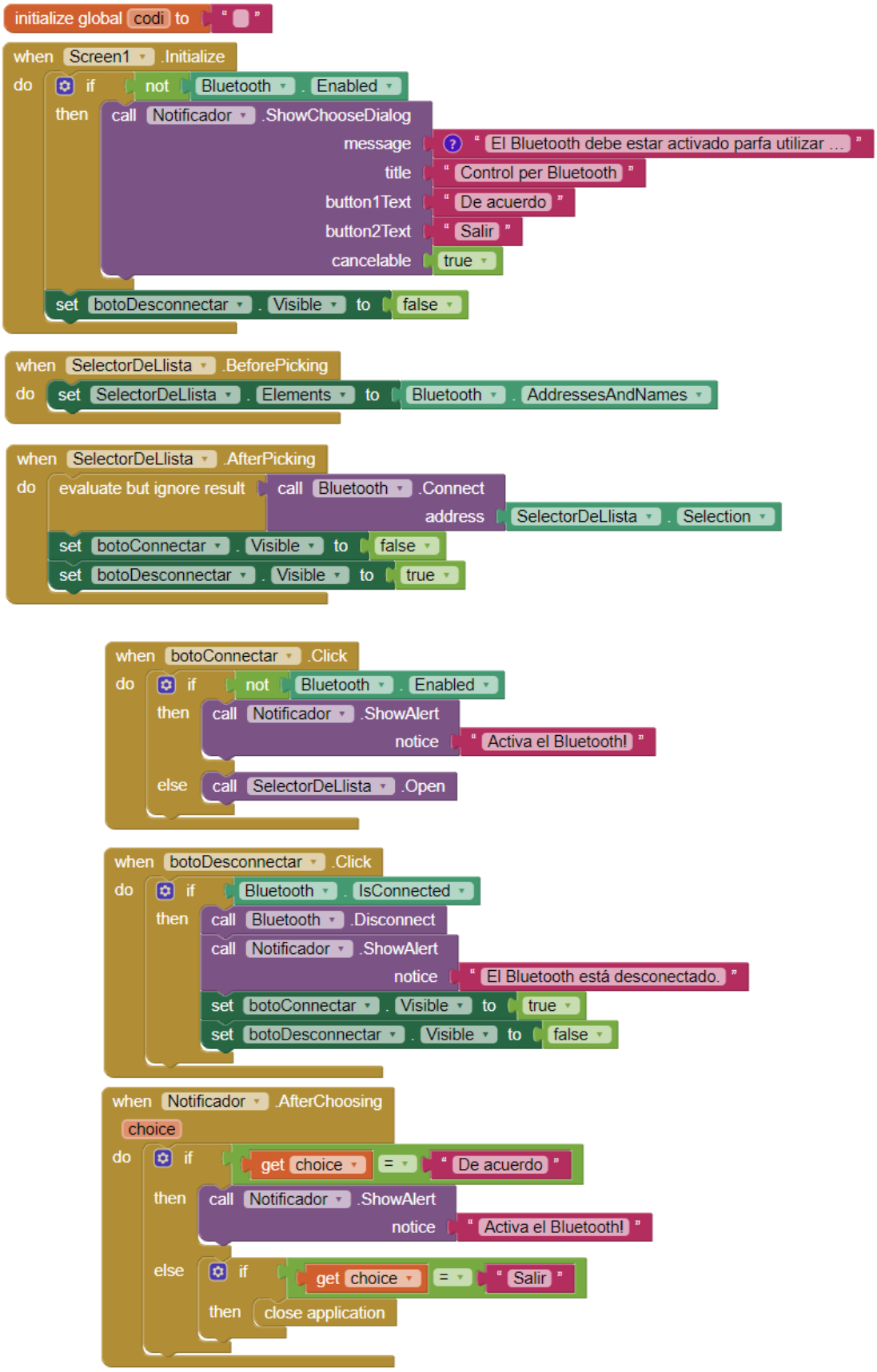
- Función *alumno_incorrecto*:

- Función *alumno_correcto*:

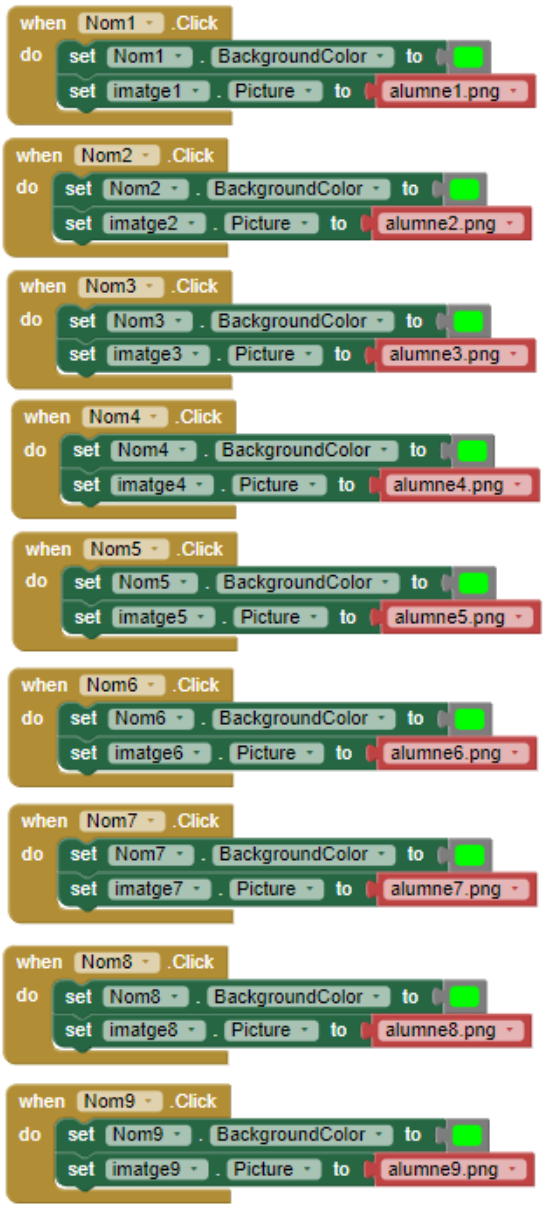


- AppInventor2 Designer:

- AppInventor2 Blocks:




when Rellotge .Timer
do if Bluetooth . IsConnected
then set global codi to call Bluetooth .ReceiveText
numberOfBytes call Bluetooth .BytesAvailableToReceive
set Codi . Text to get global codi
if get global codi = " 1 "
then set Nom1 . BackgroundColor to
set imatge1 . Picture to alumne1.png
if get global codi = " 2 "
then set Nom2 . BackgroundColor to
set imatge2 . Picture to alumne2.png
if get global codi = " 3 "
then set Nom3 . BackgroundColor to
set imatge3 . Picture to alumne3.png
if get global codi = " 4 "
then set Nom4 . BackgroundColor to
set imatge4 . Picture to alumne4.png
if get global codi = " 5 "
then set Nom5 . BackgroundColor to
set imatge5 . Picture to alumne5.png
if get global codi = " 6 "
then set Nom6 . BackgroundColor to
set imatge6 . Picture to alumne6.png
if get global codi = " 7 "
then set Nom7 . BackgroundColor to
set imatge7 . Picture to alumne7.png
if get global codi = " 8 "
then set Nom8 . BackgroundColor to
set imatge8 . Picture to alumne8.png
if get global codi = " 9 "
then set Nom9 . BackgroundColor to
set imatge9 . Picture to alumne9.png



- Captura de pantalla de la aplicación en marcha:

ESP32 Plus STEAMakers