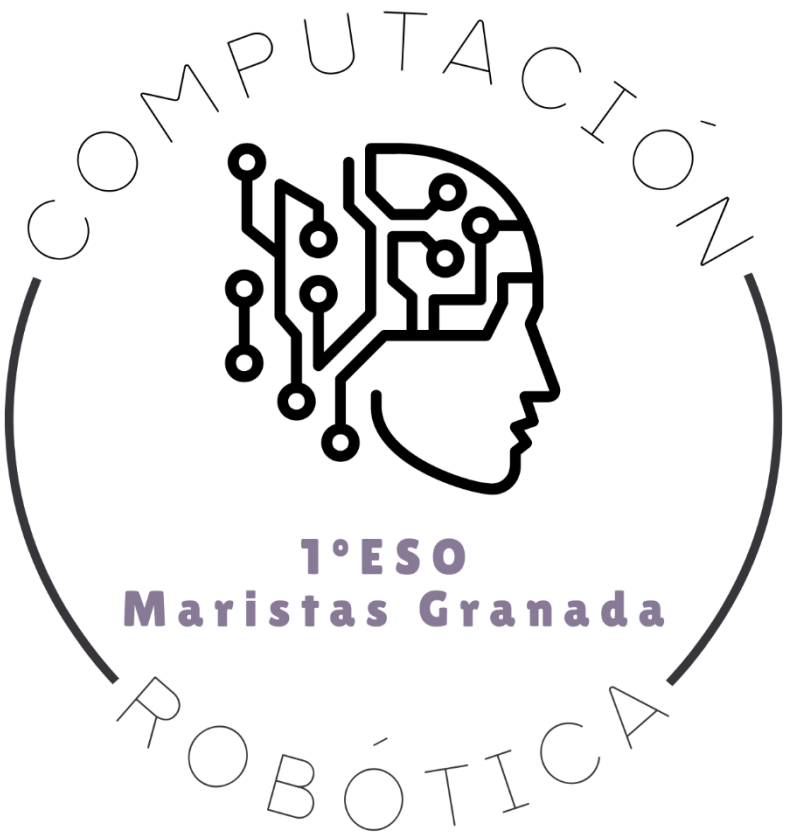
CURSO 2024-2025



DOCUMENTO DE TRABAJO: SESIONES 9-16

COMPUTACIÓN Y ROBÓTICA 1ºESO

COLEGIO MARISTA LA INMACULADA
CALLE SÓCRATES, 8
18002 - GRANADA

Índice

“Knowledge isn't free. You have to pay attention.”

— Richard P. Feynman

Teoría

9. Código César como ejemplo de codificación de textos

La actividad de cifrado César con Scratch, que vamos a desarrollar en esta sesión, está inspirada en la información disponible en la web de la Universidad de Vigo:

https://telocuentanlasmatematicas.webs.uvigo.es

El senador romano Julio César (s. I a.C.) fue uno de los grandes estrategas del imperio romano. Enviaba gran cantidad de mensajes escritos para organizar a sus tropas, extendidas a lo largo de cientos de kilómetros. O para comunicarse con el senado de Roma o con su propia familia. Si un mensaje era interceptado por el enemigo, existía el riesgo de que fuese leído por sus rivales. Para lo cual ideó uno de los primeros sistemas de cifrado de mensajes que se conocen.

Vamos a considerar el siguiente alfabeto de 27 letras (usamos la letra Ñ, aunque es cierto que esa letra no existía en tiempos del imperio romano): ABCDEFGHIJKLMNÑOPQRSTUVWXYZ

La letra A ocupa la posición 1. La letra B ocupa la posición 2. La letra C ocupa la posición 3. Y así hasta la letra Z, que ocupa la posición 27. Cuando Julio César escribía un mensaje cifrado, sustituía cada letra del mensaje original por la letra que estaba situada 3 posiciones a la derecha. De esta forma la A pasaba a D, la B pasaba a E, la C pasaba a F, y así sucesivamente. ¡Ojo! Siguiendo este razonamiento, la X pasaba a A, la Y pasaba a B, y la Z pasaba a C (al terminar el alfabeto, se comenzaba de nuevo por el principio).

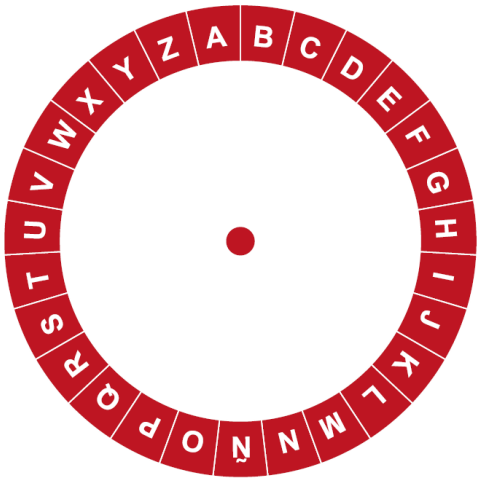
Original	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z
Cifrado	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

Si el receptor conocía la regla de las 3 posiciones, podía descifrar rápidamente el mensaje. Y si no conocía la regla… pues tenía serios problemas para entender qué ponía.

Esta regla de cifrado no es demasiado complicada. Hoy en día es posible que la veamos bastante sencilla de desvelar. Pero piensa que, en la época de Julio César, existía una variedad de lenguas diferentes, cada pueblo/tribu no solía dominar bien el idioma de otro pueblo/tribu y muchas personas (desgraciadamente) no sabían leer ni escribir. Esto hacía poco probable que alguien pudiese comprender la regla de cifrado del mensaje.

Con esta técnica de cifrado, la palabra original SALUDO se escribiría como VDÑXGR. Es decir, algo bastante ininteligible.

Los siguientes discos cumplen la función de la tabla anterior: el disco azul (imagen izquierda) representa las letras originales y el disco rojo (imagen derecha) muestra las letras cifradas. Pero tienen una ventaja respecto a la tabla. Puedes hacer girar los discos para que la clave no sea solo de 3 posiciones a la derecha, sino del número de posiciones que se desee.



Podemos colocar el disco azul dentro del disco rojo. Y rotar el disco exterior rojo para que una letra original del disco azul coincida con una letra cifrada del disco rojo con el salto que queramos: 1 posición a la derecha, 2 posiciones a la derecha, 3 posiciones a la derecha, etc.

Por ejemplo, si la letra A del disco azul coincide con la letra L del disco exterior rojo, significa que estamos llevando la letra A original 11 posiciones a la derecha, hasta conseguir la letra cifrada L. Y los discos nos darán directamente el cifrado de todas las letras con 11 posiciones a la derecha.

'Te lo cuenta una espía' Diseño de un algoritmo de cifrado César con Scratch

Puedes aprender un poco más sobre este tipo de codificación con el siguiente vídeo (está en gallego, pero se comprende perfectamente). Solo tienes que ver el vídeo hasta el minuto 2:33:

https://tv.uvigo.es/video/64c125f9b5099d3472339b35

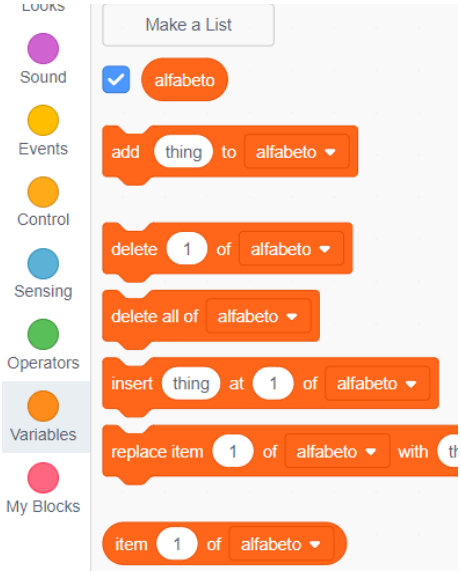
9.1. Automatizar el cifrado con Scratch: creación de listas e interfaz de comunicación máquina-usuario

Tanto si usamos la tabla del apartado anterior, como si usamos los discos, estamos realizando una tarea que sigue un patrón muy concreto: Desplazar una letra un número concreto de posiciones hacia la derecha. Ya sabemos que los ordenadores son especialmente eficaces a la hora de realizar tareas que siguen un patrón, porque nos permiten ahorrar una gran cantidad de tiempo y de esfuerzos.

Por eso, vamos a diseñar con Scratch un programa que pida la letra a cifrar, el número de desplazamientos hacia la derecha, y devuelva la letra cifrada. Los pasos a cumplir por nuestro algoritmo son:

1. Crear una lista de elementos, donde cada elemento tenga asociado una letra del alfabeto.
2. Detectar la posición numérica de cada letra dentro de la lista.
3. Desplazarse a la derecha según la cantidad indicada por el usuario.
4. Relacionar la posición final con el contenido almacenado por el elemento de la lista situado en esa posición final.

Dentro de la categoría “Variables” encontramos la posibilidad de crear una lista. Al pulsar sobre “Crear una lista” podemos indicar el nombre de la lista. Acto seguido, sobre el escenario, aparece un desplegable con la lista sin ítems.



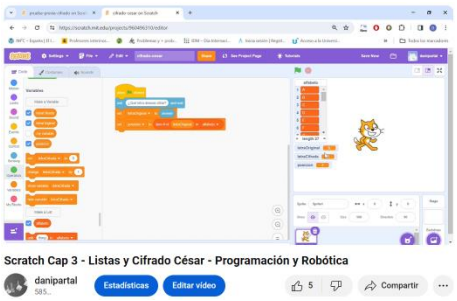
Pulsando en el icono “+” podemos añadir, uno a uno, todos los términos de la lista. Como muestra la imagen de la derecha, podemos llamar nuestra lista con el nombre “alfabeto”. Cada ítem de la lista albergará una letra: el ítem 1 contendrá la letra A; el ítem 2 contendrá la letra B; … ; el ítem 27 contendrá la letra 27.

Los requisitos que debe cumplir el programa son los siguientes:

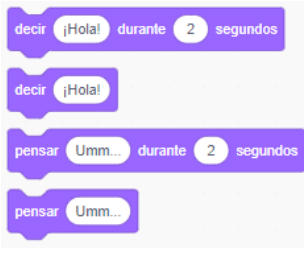
1. Preguntar al usuario la letra a codificar.
2. Almacenar la respuesta del usuario en una variable, y localizar la posición de esa letra dentro de la lista que contiene a todo el alfabeto.
3. Preguntar al usuario cuántas posiciones desea desplazarse hacia la derecha, para realizar la codificación.
4. Añadir ese desplazamiento a la posición asignada para la letra original del usuario.
5. Detectar la letra vinculada a la posición que se obtiene tras el desplazamiento
6. Devolver la letra cifrada en un mensaje por pantalla.

La explicación detallada de todo el programa la tienes disponible en el vídeo del siguiente enlace:

https://youtu.be/2_j16YgsCps



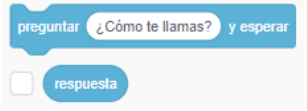
El programa necesita interaccionar con el usuario para preguntar la letra original, para preguntar el número de posiciones a desplazar, y para devolver la letra codificada. Es decir, necesitamos una **interfaz**: una forma de comunicación entre la máquina y el ser humano. Los siguientes bloques de Scratch son muy útiles para gestionar esta intercomunicación:



Los bloques de color morado pertenecen a la categoría “Apariencia”. La imagen de la derecha muestra cuatro bloques de esta categoría.

El primero y el segundo muestran un “bocadillo” de diálogo con el texto que se le indique, con la diferencia de que en el primer caso el texto está visible solo el tiempo en segundos que indiquemos. El tercero y el cuarto son idénticos a los dos primeros, con la salvedad de que aparece una “nube” a modo de pensamiento.

Estos bloques morados son especialmente útil para informar al usuario con un mensaje de texto. (proceso de salida).



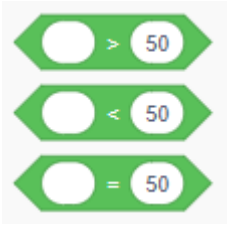
Con frecuencia no solo debemos informar al usuario, sino también preguntar y esperar una respuesta. Para esto podemos elegir, dentro de la categoría “Sensores”, el bloque “preguntar”. El programa mostrará el texto

que indiquemos en este bloque, emergiendo en el escenario un campo de texto donde teclear la respuesta. El código de programación queda a la espera de que el usuario pulse la tecla "Enter" para confirmar su respuesta.

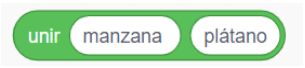
La respuesta quedará automáticamente almacenada en una variable, creada por defecto en Scratch, llamada *respuesta*. Si activamos la casilla situada junto al bloque "respuesta" veremos impresa en el escenario el contenido de esa variable. Esta variable funciona de la misma forma que cualquier variable que nosotros creemos en la categoría "Variables".

¡Ojo! Si hacemos dos preguntas consecutivas, el programa funcionará así:

1. Se muestra en pantalla la pregunta 1.
2. La respuesta del usuario se almacena en la variable *respuesta*.
3. Se muestra en pantalla la pregunta 2.
4. La nueva respuesta del usuario se almacena otra vez en la variable *respuesta*, por lo que se pierde el contenido guardado tras la primera pregunta.

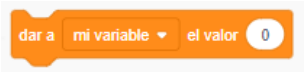


En sesiones anteriores ya hemos utilizado la categoría "Operadores", de color verde, que contiene las operaciones matemáticas básicas de sumar, restar, multiplicar y dividir. Además, podemos encontrar operaciones de comparación (ver imagen de la derecha): "mayor que", "menor que" e "igual que". El operador "igual que" funciona como el comando que vimos en PSeInt representado por dos signos iguales ==.

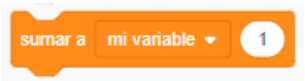


Otro bloque especialmente útil de la categoría "Operadores" es el bloque unir. Permite unir el contenido de dos textos o dos variables. Este bloque es de mucha utilidad cuando en el mensaje de salida hacia el usuario deseamos escribir una frase (primera parte) y a continuación el contenido de una variable (segunda parte).

Ya hemos la categoría "Variables" en Scratch. Vamos a explicar algo más sobre sus bloques asociados.



El bloque "dar a mi variable" permite, en primer lugar, elegir de un desplegable la variable con la que trabajar. Por defecto, Scratch siempre tiene creada una variable llamada "mi variable". En segundo lugar, guarda como contenido de la variable el valor que indiquemos en el bloque. Puede ser un valor numérico, una letra, un texto o un valor lógico Verdadero/Falso.



El bloque "sumar a mi variable" añade al contenido de la variable el valor que indiquemos en este bloque. Por ejemplo, en la imagen de la izquierda, sumamos 1 al valor ya almacenado en la variable "mi variable". El contenido de la variable queda actualizado a este nuevo valor. Este bloque es especialmente útil cuando queramos aumentar, de uno en uno, el contenido de una variable según se cumpla o no una condición del algoritmo. Verás, en programas más complejos, que usaremos con frecuencia variables cuya función es trabajar como un contador (cuentan de uno en uno).

Por último, para trabajar con los elementos de la lista que hemos creado en el programa de cifrado César y para mostrar la información de salida hacia el usuario, es muy útil conocer el funcionamiento de los siguientes bloques:

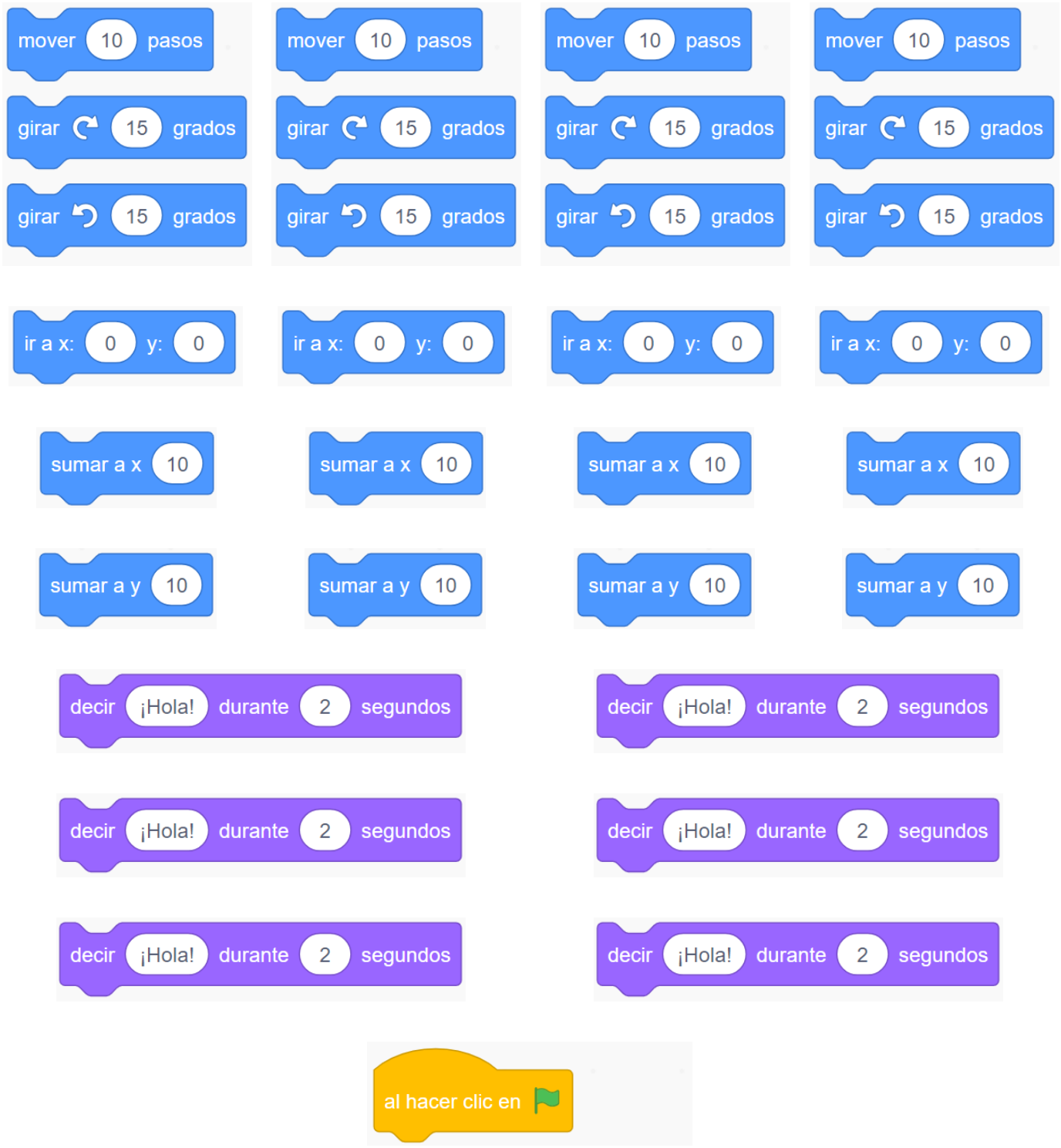
El bloque "elemento" devuelve el contenido del ítem de la lista que estamos indicando con el número. Por ejemplo (ver imagen de la izquierda), "elemento 1 de alfabeto" devuelve el contenido del ítem 1 de la lista alfabeto. El bloque "# de elemento" devuelve la posición del ítem de la lista que contiene al texto indicado. Por ejemplo (ver imagen), "# de elemento de cosa en alfabeto" devuelve la posición del ítem de la lista alfabeto cuyo contenido coincida con la expresión cosa.

Y ya tendrías toda la información necesaria para escribir, en Scratch, el algoritmo de cifrado César. La siguiente imagen te muestra el código de programación por bloques completo**. ¡Ojo! Este programa tiene sus limitaciones. Hay letras que no se codifican bien. ¿Qué pasa si, al sumar el desplazamiento, superamos la cantidad 27 (posición de la letra Z) ¿Se te ocurre como controlar, con un condicional, este supuesto y cómo mejorar el código del programa?**

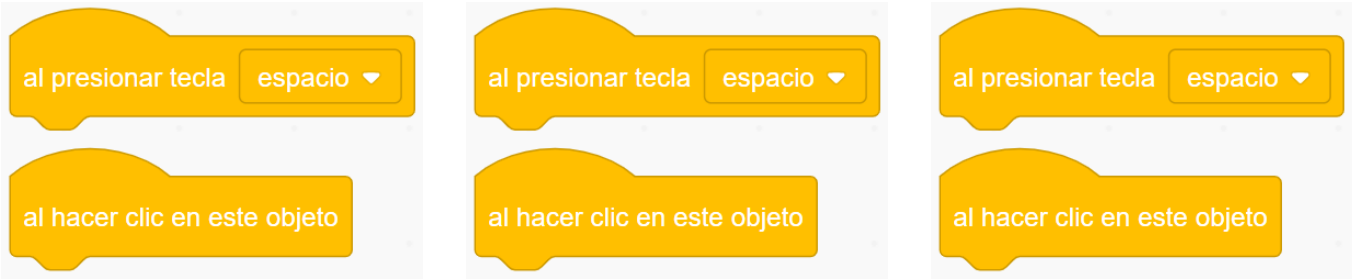
10. Gincana de retos con bloques recortables de Scratch

En esta sesión vamos a poner en práctica todo lo aprendido sobre Scratch. Pero en vez de trabajar con el ordenador, vamos a trabajar en el patio, en grupos de 4-5 personas, con bloques recortables con los que montar el código de programación. El profesor ofrecerá a cada grupo pliegos a color en tamaño A3 con los bloques. Una vez estén todos los bloques recortados, iremos al patio para poder movernos libremente y organizar sobre el suelo la unión de los distintos bloques que ejecuten los programas que se plantean en la sección de retos del final de este documento.

Solo puedes hacer los programas con los bloques que te entregue el profesor. Puedes consultar tu cuaderno de clase, pero no el ordenador ni los tutoriales online para resolver los retos de esta sesión. En los bloques puedes escribir el nombre o el texto (número de pasos, ángulo de giro, nombre de variable, etc.) que necesites para diseñar correctamente tus programas.

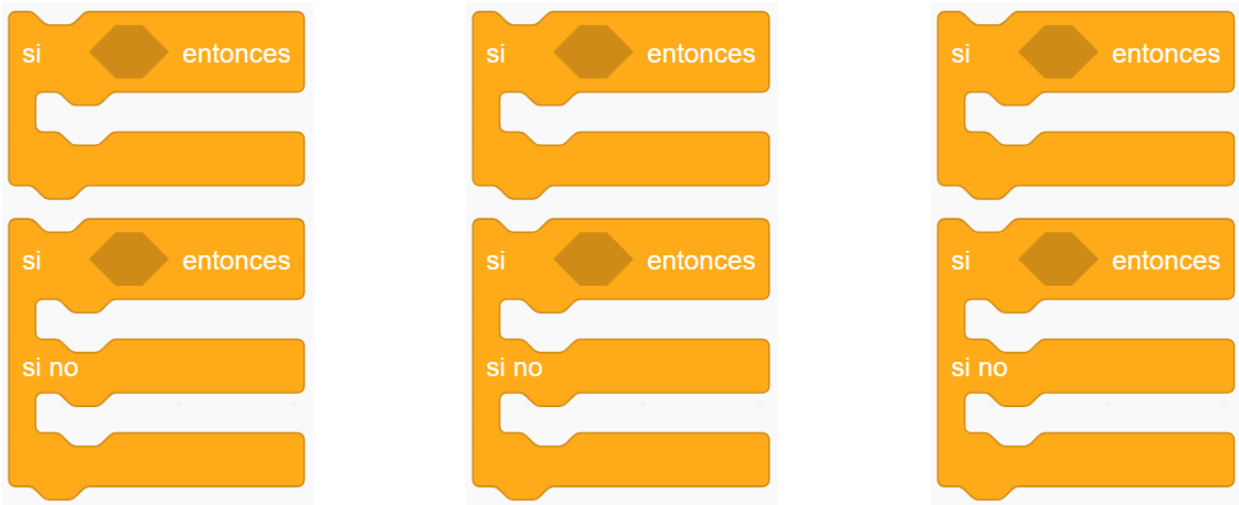



al presionar tecla espacio
al hacer clic en este objeto
al presionar tecla espacio
al hacer clic en este objeto
al presionar tecla espacio
al hacer clic en este objeto


al presionar tecla espacio
al hacer clic en este objeto
al presionar tecla espacio
al hacer clic en este objeto
al presionar tecla espacio
al hacer clic en este objeto

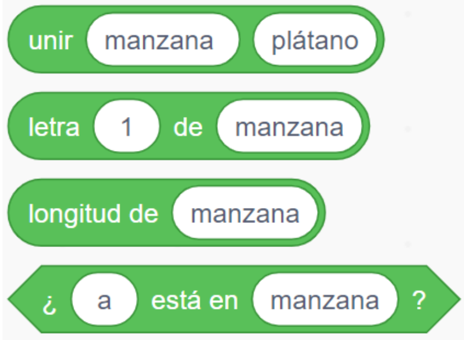

esperar 1 segundos
esperar 1 segundos
esperar 1 segundos


repetir 10
por siempre
repetir 10
por siempre
repetir 10
por siempre

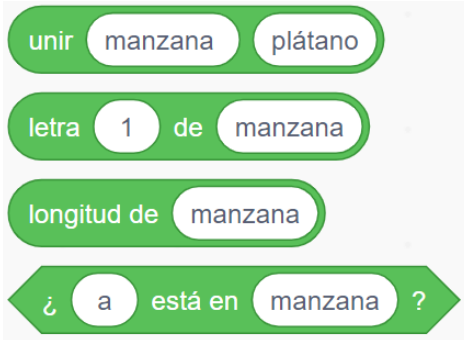

si entonces
si entonces
si no
si entonces
si entonces
si no
si entonces
si entonces
si no

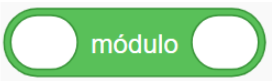

repetir hasta que
repetir hasta que
repetir hasta que
¿tocando el color ?
¿tocando el color ?
¿tocando el color ?
preguntar ¿Cómo te llamas? y esperar
respuesta
preguntar ¿Cómo te llamas? y esperar
respuesta
preguntar ¿Cómo te llamas? y esperar
respuesta
preguntar ¿Cómo te llamas? y esperar
respuesta
¿tecla espacio presionada?
¿tecla espacio presionada?
¿tecla espacio presionada?
¿tecla espacio presionada?
+
-
*
/
+
-
*
/
+
-
*
/
+
-
*
/
+
-
*
/
número aleatorio entre 1 y 10
número aleatorio entre 1 y 10
> 50
< 50
= 50
> 50
< 50
= 50
> 50
< 50
= 50
> 50
< 50
= 50

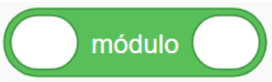

unir manzana plátano
letra 1 de manzana
longitud de manzana
¿ a está en manzana ?

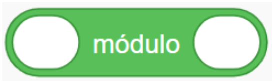

unir manzana plátano
letra 1 de manzana
longitud de manzana
¿ a está en manzana ?


unir manzana plátano
letra 1 de manzana
longitud de manzana
¿ a está en manzana ?


unir manzana plátano
letra 1 de manzana
longitud de manzana
¿ a está en manzana ?


módulo


módulo


módulo


mi variable
dar a mi variable el valor 0
sumar a mi variable 1

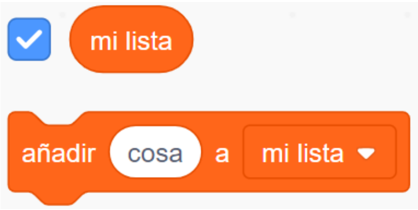

mi variable
dar a mi variable el valor 0
sumar a mi variable 1

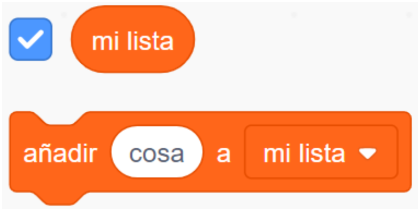

mi variable
dar a mi variable el valor 0
sumar a mi variable 1

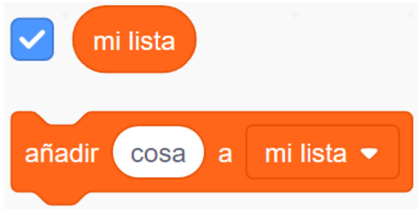

mi variable
dar a mi variable el valor 0
sumar a mi variable 1


mi variable
dar a mi variable el valor 0
sumar a mi variable 1


mi variable
dar a mi variable el valor 0
sumar a mi variable 1


mi lista
añadir cosa a mi lista


mi lista
añadir cosa a mi lista

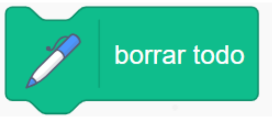

mi lista
añadir cosa a mi lista

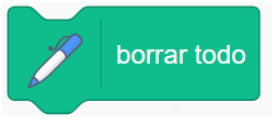

elemento 1 de mi lista
de elemento de cosa en mi lista
longitud de mi lista
¿ cosa está en mi lista ?

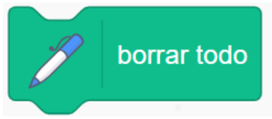

elemento 1 de mi lista
de elemento de cosa en mi lista
longitud de mi lista
¿ cosa está en mi lista ?

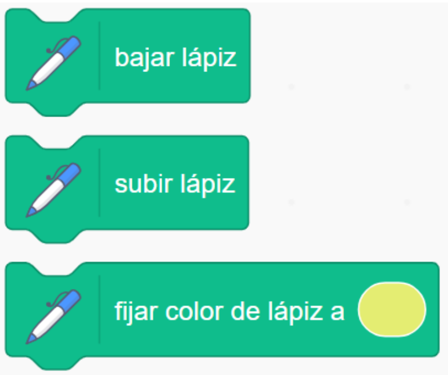

elemento 1 de mi lista
de elemento de cosa en mi lista
longitud de mi lista
¿ cosa está en mi lista ?

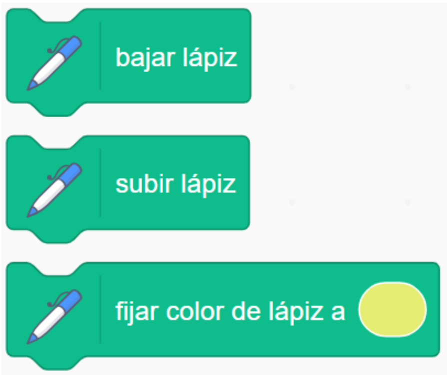

elemento 1 de mi lista
de elemento de cosa en mi lista
longitud de mi lista
¿ cosa está en mi lista ?

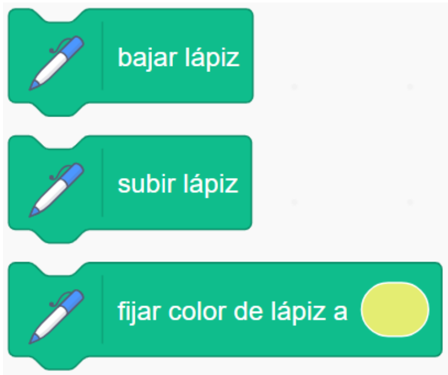

borrar todo

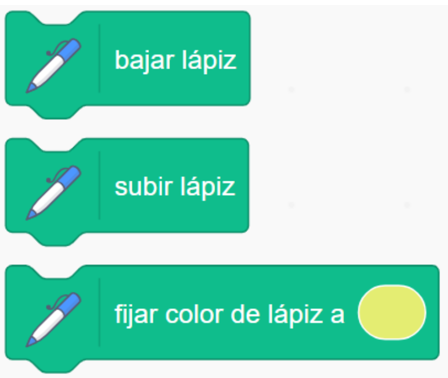

borrar todo

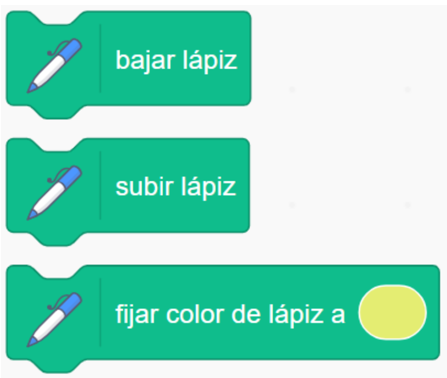

borrar todo

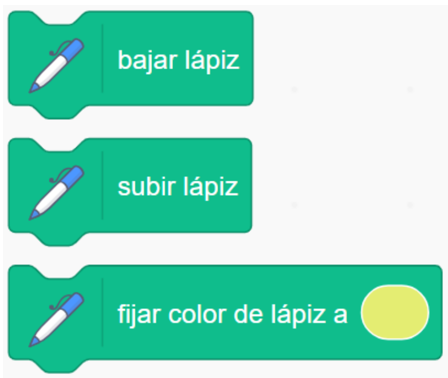

bajar lápiz
subir lápiz
fijar color de lápiz a


bajar lápiz
subir lápiz
fijar color de lápiz a


bajar lápiz
subir lápiz
fijar color de lápiz a


bajar lápiz
subir lápiz
fijar color de lápiz a


bajar lápiz
subir lápiz
fijar color de lápiz a


bajar lápiz
subir lápiz
fijar color de lápiz a

11. Continuación de la gincana de retos con Scratch de la sesión anterior

Recortar los bloques y pensar los códigos de programación de los retos de la sesión 10 conlleva bastante tiempo. Por lo tanto, en la sesión 11 continuamos con la misma dinámica de trabajo de la sesión 10. Estos retos generan una única actividad de calificación en la evaluación de la asignatura.

12. Sensores. Aplicaciones de la placa micro:bit

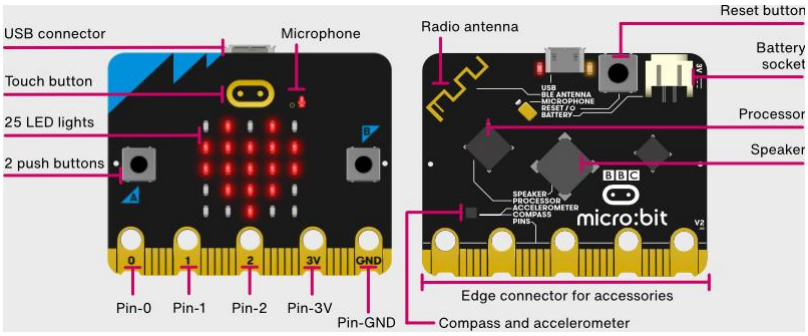
Un robot recibe señales del exterior, las procesa y actúa. Recibe la señal por los **sensores** (de temperatura, de luz, de distancia, etc.). Procesa la información con el **ordenador instalado en la microcontroladora**. Y se pone en movimiento o actúa con los **actuadores** (motores, luces, altavoces, etc.).

Los sensores perciben las señales del entorno y son capaces de detectar magnitudes físicas o químicas como temperatura, velocidad, inclinación, concentración de materia, etc. y convertirlas en señales eléctricas. Gracias a esta transformación a señal eléctrica, y a los códigos de programación, el ordenador de la microcontroladora puede procesar la información.

La microcontroladora micro:bit lleva ya incorporada varios sensores en la propia placa de la microcontroladora:

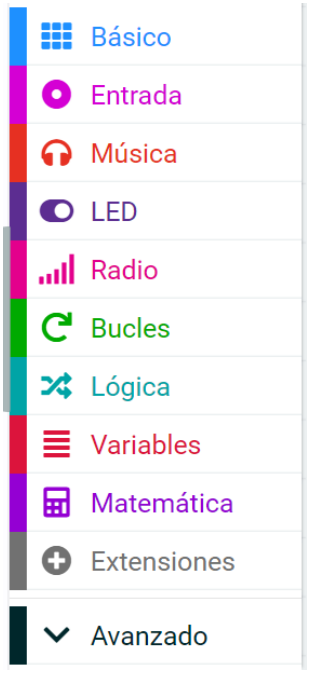
- Dos botones de contacto, simbolizados con las letras A y B.
- Un botón capacitivo en el logotipo (funciona al estilo de una pantalla táctil).
- 25 diodos LEDs sensibles a la cantidad de luz.
- Un micrófono.
- Un sensor de temperatura.
- Un sensor de movimiento.
- Un sensor de cambio de velocidad (acelerómetro).
- Una brújula.
- Una antena que recibe (y emite) señales de radio.
- Una antena de comunicación por Bluetooth.

Además, la placa micro:bit ofrece un conector micro-USB para conectar el ordenador y poder descargar el archivo con el código de programación.



Podemos programar la placa micro:bit con una extensión de Scratch. No obstante, para acostumbrarnos a trabajar en distintos entornos de programación, vamos a trabajar con la zona de edición online MakeCode especialmente diseñado para micro:bit.

En las primeras sesiones de la asignatura ya usamos, a modo introductorio, MakeCode. Ahora vamos a diseñar programas más complejos con micro:bit. Las categorías de bloques de MakeCode, como puedes ver en la imagen de la derecha, son muy parecidas a las categorías de bloques de Scratch.



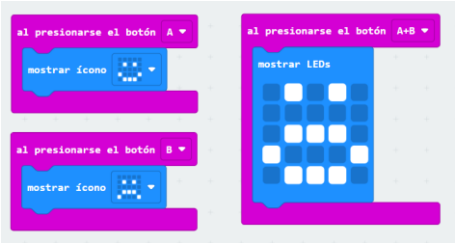
12.1. Botones de contacto

La web de micro:bit está lleno de tutoriales online. Aprendamos a interactuar con la placa pulsando los botones A y B, con el siguiente programa:

https://microbit.org/projects/make-it-code-it/emotion-badge/

Al iniciar el programa, el display no muestra nada. Si pulsamos A, aparece una cara feliz. Si pulsamos B, aparece una cara triste. Si pulsamos a la vez A + B aparece una cara de asombro. El bloque “al presionar el botón …” está disponible en la categoría “Entrada”.

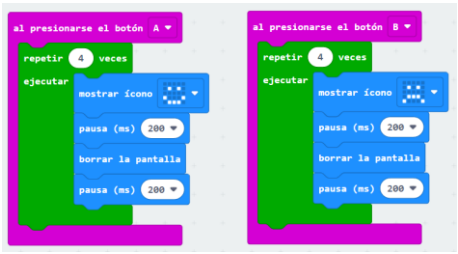
Fíjate que en MakeCode, si sobre un bloque pulsas con el botón derecho del ratón, puedes duplicar todo el contenido del bloque. Lo cual facilita la tarea de crear bloques con estructura similar. Esta opción de duplicar también la vimos en la zona de edición de Scratch.



12.2. Bucle "repetir" en MakeCode y pausas temporales en la ejecución del código

Cuando deseamos que una acción se repita varias veces, de manera consecutiva, podemos crear un loop con el bloque "repetir … veces". En el siguiente programa, si pulsamos A la cara sonriente aparece y desaparece 4 veces. Mientras que, si pulsamos B, hace lo propio la cara triste. El bloque "repetir" está disponible en la categoría "Bucles".

https://microbit.org/projects/make-it-code-it/flashing-emotions/?editor=makecode



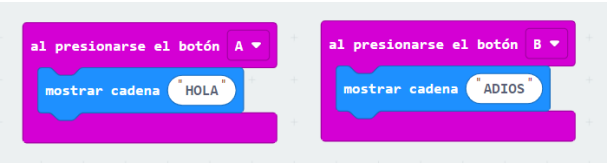
Cada bloque "al presionar el botón A" y "al presionarse el botón B" funcionan como un "Evento" en Scratch: Solo si pulsamos algunos de los botones, se ejecutan los bloques contenidos dentro de ellos.

El bloque "pausa (ms)" acepta un valor numérico para indicar los milisegundos que la ejecución del código permanecerá en espera. Este bloque está disponible en la categoría "Básico".

12.3. Mostrar una cadena de texto en la pantalla LED

Si deseamos mostrar una palabra en el display, ¿lo hacemos letra a letra? Es más eficiente mostrar la palabra con desplazamiento horizontal, al estilo de las pantallas informativas del metro, donde las frases largas aparecen por la derecha y se esconden por el lado izquierdo.

El siguiente programa muestra la cadena de texto "HOLA" cuando pulsamos A. Y la cadena de texto "ADIOS" cuando pulsamos B. El bloque que nos permite mostrar en movimiento una cadena de texto es el bloque "mostrar cadena", contenido en la categoría "Básico".



12.4. Bloque condicional "si" en MakeCode

La placa cuenta con un acelerómetro integrado, sensible a impactos bruscos sobre la placa o agitaciones:

https://microbit.org/projects/make-it-code-it/get-silly/

Modificamos uno de los códigos de los apartados anteriores de esta sesión. Mostramos cara feliz al pulsar A. Cara triste al pulsar B. Y una cruz al agitar, usando el bloque condicional "si agitado" (disponible en la categoría "Entrada").

El bloque "si agitado" puede reaccionar ante diversas acciones: agitar, movimiento horizontal, caída libre, giro, grado de aceleración que sufre la placa (en unidades del valor g de la gravedad terrestre: 3g, 6g y 8g), etc. Estas opciones se pueden seleccionar en el desplegable del bloque "si agitado".

12.5. Número aleatorio en MakeCode

El bloque "mostrar número" (categoría "Básico") y el bloque "escoger al azar de … a …." (categoría "Matemática") permiten, de forma muy sencilla, crear un dado que fluctúa de puntuación cada vez que agitemos la placa:

https://microbit.org/projects/make-it-code-it/dice/

12.6. Diseñar un dado de seis opciones con micro:bit

Vamos a modificar el código anterior para que nuestro dado muestre en la pantalla LED la puntuación usando puntos en vez de números. De paso, manejaremos nuevos bloques condicionales en MakeCode. El código de ejemplo está disponible en:

https://microbit.org/projects/make-it-code-it/graphical-dice/

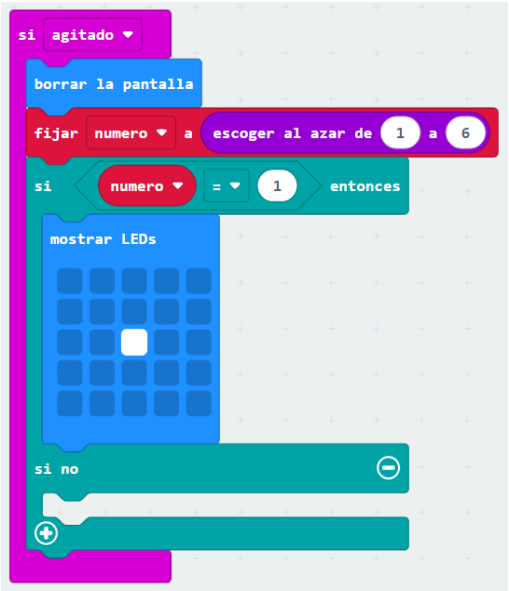
Crearemos una variable llamada *numero* con la opción "Variables" (recuerda que no ponemos tildes a los nombres de las variables). Esta variable *numero* va a almacenar el valor del número aleatorio que se produce al agitar la placa. Y según el valor almacenado en *numero* mostraremos en el display un punto, dos puntos, tres puntos, cuatro puntos, cinco puntos o seis puntos.

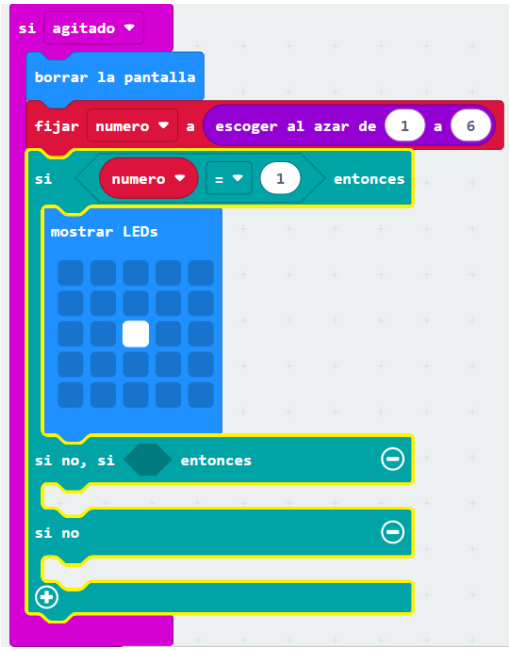
Dentro de la categoría “Lógica” encontramos el bloque “si … entonces … si no …”. Este bloque condicional permite ejecutar una acción si se cumple una condición o ejecutar otra acción si no se cumple dicha condición. Y podemos encadenar tantos bloques “si … entonces … si no …” como deseemos.

En la categoría “Lógica” también encontramos los bloques que permiten comparar números: mayor que, menor que e igual que. ¡Ojo! Hay otro bloque que permite comparar cadenas de texto, y se distingue por las dobles comillas que aparecen en el espacio para escribir los valores a comparar. En la imagen de la derecha, los dos primeros bloques comparar números, mientras que el tercero (verás las dobles comillas indicadas) comparan cadenas de texto. La confusión al usar ambos bloques generan típicos errores a la hora de programar.

- Si el valor de la variable *numero* es igual a 1, dibujamos en el display un punto justo en el centro de la matriz de Leds. Pulsando en el icono + de la parte inferior del bloque “si … entonces … si no …” podemos añadir más condicionales encadenados.
- Si la variable *numero* vale 2, la pantalla muestra 2 puntos.
- Si la variable *numero* vale 3, dibujamos 3 puntos.
- Si la variable *numero* vale 4, mostramos 4 puntos.
- Si la variable *numero* vale 5, se encienden 5 puntos.
- Si la variable *numero* vale 6, se iluminan 6 puntos.

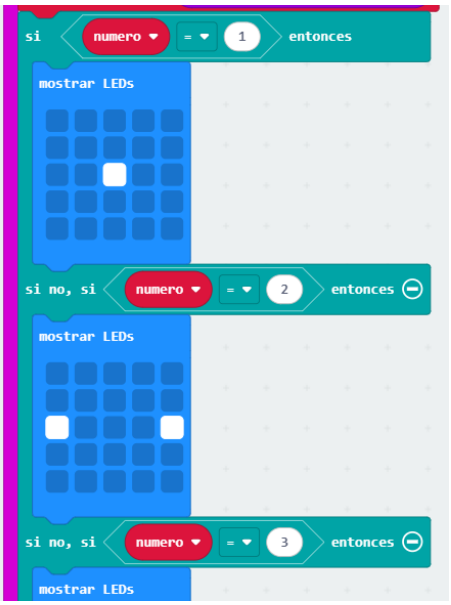
Recuerda usar la opción duplicar, pulsando con el botón derecho del ratón, para copiar y pegar bloques que ya tienes definidos. Esto agilizar mucho la tarea de creación de bloques con funcionalidades parecidas. Verás que el código se alarga mucho, verticalmente, en la pantalla de edición. Por eso es importante ser ordenado a la hora de programar.





Es importante borrar la pantalla tras agitar, para dar sensación visual al usuario que el antiguo valor desaparece y se muestra el valor nuevo del dado. El bloque “si agitado” es un evento que se ejecuta siempre que la placa se agite.

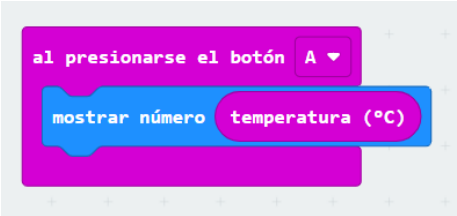
Fíjate en la imagen de la derecha como van enganchando, uno a continuación de otro, un concicional “si … entonces … si no …”. Usaremos seis veces el bloque “mostrar LEDs” para señalar, manualmente, los puntos que deseemos que se enciendan en cada tirada del dado.



13. Más aplicaciones de micro:bit con sensores. Usar funciones para ahorrar código de programación

13.1. Sensor de temperatura

La categoría "Entrada" posee un bloque "temperatura" que almacena el valor que lee el sensor de temperatura de la placa. Podemos mostrar el valor de la temperatura en la pantalla LED con el siguiente ejemplo (en la imagen de la derecha tienes la programación por bloques en MakeCode):



https://microbit.org/es-es/projects/make-it-code-it/thermometer

El valor de la temperatura aparece como un texto, de derecha a izquierda en el display.

Puedes introducir la placa en un congelador y, pasado un tiempo, sacarla. E ir pulsando el botón A para apreciar el aumento progresivo de la temperatura, hasta llegar al equilibrio con la temperatura ambiente del exterior.

13.2. Sensor de luz

Los diodos LED de la pantalla emiten luz y también son sensibles a la luz. Es decir, pueden funcionar para introducir información dentro la placa y para mostrar información hacia el exterior. El sensor de luz detecta variaciones lumínicas en una escala de referencia que oscila entre 0 y 255. Siendo 0 oscuridad total y siendo 255 el valor máximo de luz que puede captar el sensor. ¡Ojo! El sensor puede saturarse: Pasado un nivel máximo de luz, la señal que recoge el sensor no seguirá aumentando aunque nosotros aumentemos la intensidad de la luz externa.

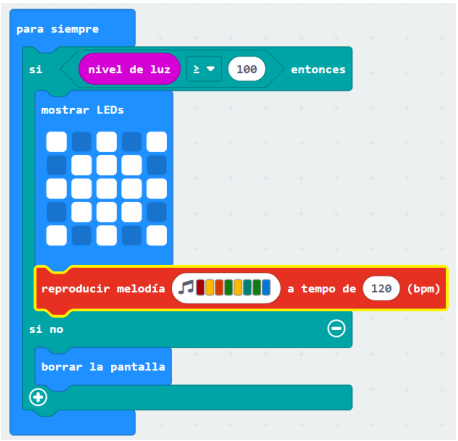
Con el siguiente código diseñamos un programa para que la placa muestre en la pantalla LED el dibujo de un Sol si la intensidad lumínica es mayor o igual que el nivel de referencia 100:

https://microbit.org/es-es/projects/make-it-code-it/sunlight-sensor/

En clase podemos usar una linterna para variar la cantidad de luz que llega al sensor de luz. Un cloque condicional "si … entonces … si no" controla que aparezca el sol en la pantalla LED si el nivel de luz es mayor o igual que 100, y que no aparezca nada (bloque "borrar pantalla") en caso contrario.

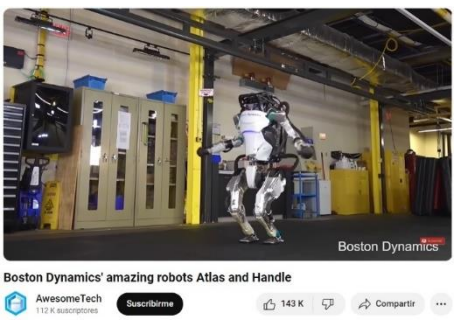
Mejoremos el código anterior, añadiendo una señal sonora cuando el nivel de luz excede de un valor determinado. Más adelante vamos a estudiar en más profundidad los bloques de la categoría "Música". Por ahora, si miras la imagen inferior, es suficiente con saber que el bloque "reproducir melodía" permite elegir ocho notas dentro de una octava (Desde DO hasta el DO de la siguiente octava) y reproducir cada nota al tempo marcado por el valor indicado en **bmp (beats per minute)**. Si dentro de esas ocho notas una no es seleccionada, se genera un silencio.

¡IMPORTANTE! Cuando el profesor lo indique, todos podremos probar en clase con los bloques de la categoría "Música". Pasado un tiempo prudencial, dejaremos de utilizarlo porque es muy molesta estar en un aula con 30 placas micro:bit sonando cada una con una melodía y tempo distintos.



13.3. Medir rango de temperaturas durante un periodo de tiempo. Las fuentes de alimentación

Si conectamos la placa a una batería externa (dos pilas AAA de 1,5 V), podemos emplear micro:bit durante un periodo de tiempo largo sin necesidad de estar conectado al ordenador mediante USB. La alimentación eléctrica y el consumo de energía son aspectos vitales en informática, electrónica y robótica. En el siguiente vídeo del canal de YouTube AwesomeTech, verás un robot de la empresa Boston Dynamics' haciendo piruetas increíbles. La mochila negra que lleva el robot a la espalda son las baterías, cruciales para garantizar el consumo energético de cada circuito del robot: https://www.youtube.com/watch?v=uhND7Mvp3f4



Una **fuente de alimentación** es un dispositivo que convierte la corriente eléctrica de los enchufes que tenemos en las paredes de casa o del colegio, en corriente continua útil para los pequeños circuitos electrónicos que encontramos en una placa como micro:bit. La corriente continua es el tipo de corriente que ofrece una batería o una pila.

Las fuentes de alimentación pueden dar uno o varios voltajes de salida en corriente continua. Donde la palabra "voltaje" (explicado a un nivel que podamos entender en 1ºESO) es una de las magnitudes físicas (junto a "intensidad") que permiten calcular la energía que consume un circuito eléctrico.

La parte fundamental que convierte la corriente alterna en corriente continua es el **transformador**. Por ejemplo, en el cable de alimentación de un ordenador portátil, el transformador (y otros componentes) lo encontramos en el interior de la cajita sólida que aparece en algún punto de la longitud del cable. El transformador, como su nombre indica, "transforma" la señal de la corriente alterna (que posee un voltaje del orden de las centenas, con grandes oscilaciones de la intensidad eléctrica) en corriente continua (del orden de unos pocos voltios que se suministra de forma constante, sin oscilaciones).

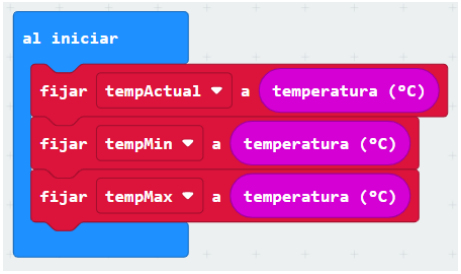
Las fuentes de alimentación y los circuitos eléctricos en funcionamiento generan calor. Eso es una pérdida de energía irrecuperable para el circuito, además de un obstáculo que ralentiza el funcionamiento de los ordenadores. Por eso, en todo circuito eléctrico, es de vital importancia la refrigeración de los dispositivos. **A menor temperatura, mejor funcionamiento de los equipos electrónicos**.

En el siguiente programa vamos a guardar información sobre la temperatura mínima registrada por la placa micro:bit, la temperatura máxima y la temperatura actual dentro de un periodo de tiempo: https://microbit.org/es-es/projects/make-it-code-it/max-min-thermometer/

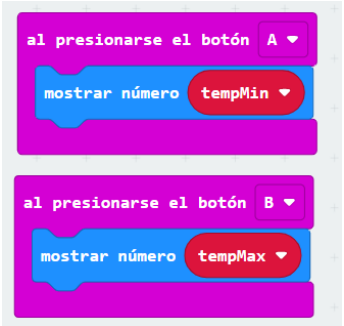
Creamos tres variables (categoría "Variables"):

- *tempActual*: almacena la temperatura actual.
- *tempMax*: almacena la temperatura máxima.
- *tempMin*: almacena la temperatura mínima.

Al inicio del programa en MakeCode fijamos estas tres variables al valor de la temperatura recogida por el sensor de temperatura. Este sensor está integrado en la parte posterior de la placa, en el microprocesador. Por lo tanto (ver imagen inferior), cuando el programa se inicia (bloque "al iniciar", situado en la categoría "Básico") las tras variables toman exactamente el mismo valor (que será el valor recogido por el sensor de temperatura de la placa).



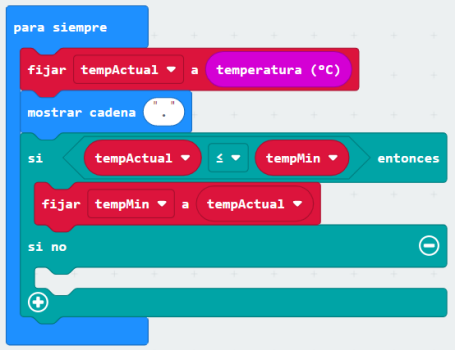
Cuando pulsemos el botón A, mostraremos en pantalla el valor de la temperatura mínima. Cuando pulsemos el botón B, mostraremos el valor de la temperatura máxima.



Una vez iniciado el programa, damos paso al código que controla el bloque "para siempre" (categoría "Básico"). Este bloque siempre está ejecutándose, para poder medir las variaciones de temperatura a lo largo del tiempo.

En el bloque "para siempre" informaremos, con un punto parpadeante, que la placa está recogiendo datos (aunque estos datos no se muestren en pantalla). Es una manera de informar al usuario que la placa está en funcionamiento y no debe desconectarla de la alimentación.

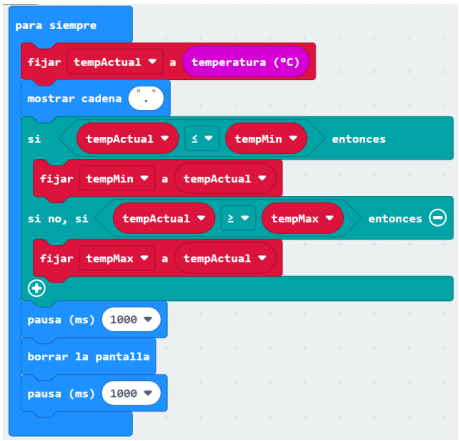
Fijamos el valor de la temperatura actual con el valor recogido por el sensor de temperatura (bloque "fijar" de la categoría "Variables"). Dibujamos un punto con el bloque "mostrar cadena" y comparamos el valor de la variable *tempMin* con el valor de *tempActual*.



Si *tempActual* es inferior al valor de *tempMin*, actualizamos el valor de *tempMin* al nuevo valor mínimo. En la imagen de la derecha verás que la comparación la hacemos con un bloque condicional "si … entonces …. si no …". En la siguiente tabla verás un ejemplo aclaratorio de cómo opera este condicional en nuestro programa de temperaturas.

A modo de ejemplo, para comprender mejor el código, supondremos que el sensor de temperatura toma un valor cada segundo			
Tiempo	**valor de sensor de temperatura**	**valor de *tempActual***	**valor de *tempMin***
0 s	20 ºC	20 ºC	20 ºC
1 s	20 ºC	20 ºC	20 ºC
2 s	21 ºC	21 ºC	20 ºC
3 s	22 ºC	22 ºC	20 ºC
4 s	20 ºC	20 ºC	20 ºC
5 s	18 ºC	18 ºC	18 ºC
6 s	17 ºC	17 ºC	17 ºC
7 s	20 ºC	20 ºC	17 ºC

Razonamos de forma parecida con *tempMax*, añadiendo un nuevo condicional "si … entonces … " enganchando con el condicional "si … entonces … si no …" (ver imagen de la derecha). Si la temperatura actual supera a la almacenada previamente como temperatura máxima, actualizamos el valor de *tempMax* al valor contenido en *tempActual*.



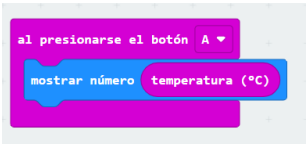
Añadimos una pausa de un 1 segundo y borramos la pantalla. Y el bucle infinito se repite (bloque "para siempre"). El punto parpadeante informa al usuario que el programa está midiendo, a la espera de que el usuario pulse el botón A o el botón B para recoger los datos de salida. Esto ahorra batería y solo muestra la información en el display cuando se le requiere.

¡Y ya está! Prueba a situar la placa, conectada a la batería, en dos lugares con temperatura claramente diferenciadas. Deja pasar algunos minutos y pulsa los botones A y B para conocer los valores mínimos y máximos en ese intervalo.

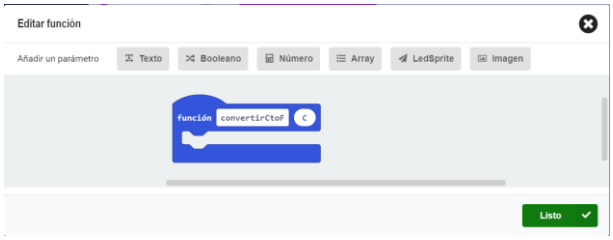
13.4. Uso de funciones en micro:bit

Vamos a crear nuestra primera función con micro:bit. **Una función es un trozo de código que, cada vez que lo uso en mi algoritmo, debo indicar un valor de inicio para que se ejecuten una serie comandos previamente diseñados y devuelva un valor concreto**.

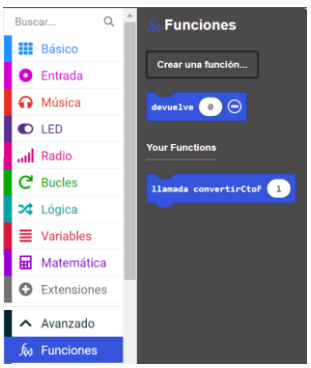
Por ejemplo, vamos a crear una función que pase de grado centígrados a grados Fahrenheit. En primer lugar, al pulsar el botón A mostramos en pantalla la temperatura en grados centígrados (ver imagen de la derecha).



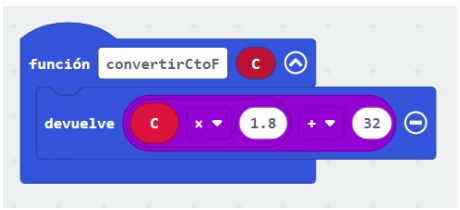
En segundo lugar, creamos la función. Accedemos a la categoría "Funciones". Le damos como nombre "convertirCtoF" y añadimos un parámetro tipo número como valor de entrada, que llamaremos *C* (por ser la inicial de la palabra centígrado, aunque podemos darle cualquier nombre). Esta letra *C* representa el valor de inicio que daremos a la función cada vez que la invoquemos en nuestro algoritmo.



La función devuelve un valor número, fruto de la operación de conversión siguiente: para pasar *C* grados centígrados a grados Fahrenheit debemos multiplicar C por 1.8 y sumar 32 unidades. Es decir, la operación matemática sería $C \times 1.8 + 32$. En el menú "Funciones" encontramos el bloque "devuelve" (ver imagen de la derecha).

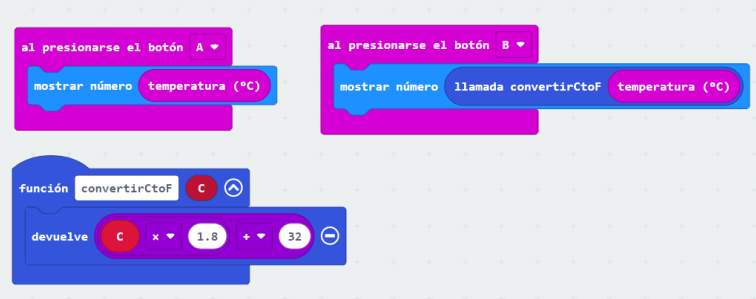


En el bloque "devuelve" debemos introducir dos bloques de matemáticas. Ojo con la jerarquía de operaciones: primero se realiza el producto (C x 1.8) y luego debemos sumar 32 unidades (ver imagen inferior). Como si usáramos paréntesis y jerarquía de operaciones en la clase de Matemáticas.



Finalmente, al pulsar el botón "B" se muestra en pantalla la conversión a grados Fahrenheit. Para ello, tenemos que usar el bloque "llamada" situado en la categoría "Funciones" para que se ejecute la función que hemos diseñado. Al llamar a la función, indicamos que el valor del sensor

de temperatura será el valor de entrada con el que operar en su interior. Es decir, el valor del sensor de temperatura hace las veces de *C* en la fórmula anterior.



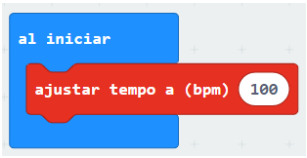
Las funciones son especialmente útiles porque podemos llamarlas tantas veces como queramos, introducir valores de entrada distintos en cada llamada y conseguir la ejecución de las mismas operaciones una y otra vez. Las funciones ahorran mucho código de programación.

14. Hacer música con micro:bit, usar pines digitales y variables lógicas booleanas

14.1. Crear un metrónomo para ajustar tempo musical

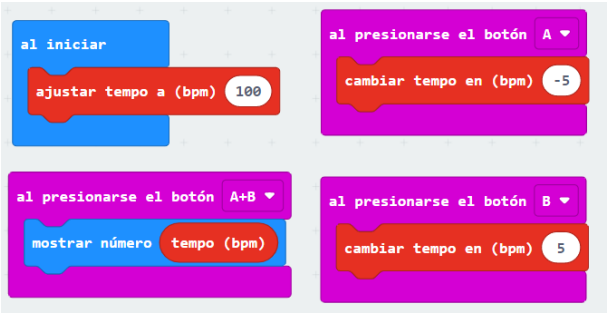
El primer programa de esta nueva sesión lo tienes descrito en:

https://microbit.org/projects/make-it-code-it/metronome/



Las siglas bpm quieren decir “beats per minute”. En español, golpes por minuto. Si ajustamos el tempo a 100 bpm al inicio del programa y elegimos una melodía que solo repita la nota DO, y decimos que esa nota tiene un pulso de duración, escucharemos 100 veces la nota DO en un minuto de tiempo.

Vamos a diseñar que, al pulsar el botón A, reduzcamos en 5 unidades los bpm. Mientras que, al pulsar el botón B, aumentaremos en 5 unidades el tempo. Al pulsar de manera conjunta A+B mostraremos en pantalla el número de bpm actual. Fíjate que el bloque “cambiar tempo en (bmp) …” suma la cantidad que indicamos al valor ya almacenado en tempo.



En un bucle infinito “para siempre” generamos un tono de duración 1/16 pulso, seguido de un descanso de 1 pulso (ver imagen de la derecha). De esta forma, el siguiente tono de paso del metrónomo se reproduce después del pulso de reposo. Así evitamos el solapamiento de las ondas sonoras.

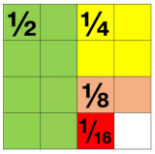
¿Cuánto tiempo, en segundos, dura un pulso? Depende del tempo. Si por ejemplo tengo 120 bpm significa que hay 120 beats en un minuto. Es decir, hay 120/60 pulsos en un segundo. O lo que es lo mismo, 2 pulsos por segundo. O bien, un pulso dura 0,5 segundos.

Ahora mismo, nuestro metrónomo funcionaría así:

- Emite un sonido durante 1/16 fracciones de pulso.
- Descansa 1 pulso antes de emitir el siguiente sonido.

Por lo tanto, en un ciclo del bucle el programa consume 1/16 + 1 pulso. Es decir, 17/16 fracciones de pulso. Estamos generando un desfase de 1/16. Por lo que nuestro metrónomo va 1/16 fracción de pulso más lento de lo que debería ser en realidad.

¿Cómo corregirlo? Haciendo que el bloque “rest” no consuma un pulso completo sino 15/16 fracciones de pulso. Fíjate en la siguiente imagen:



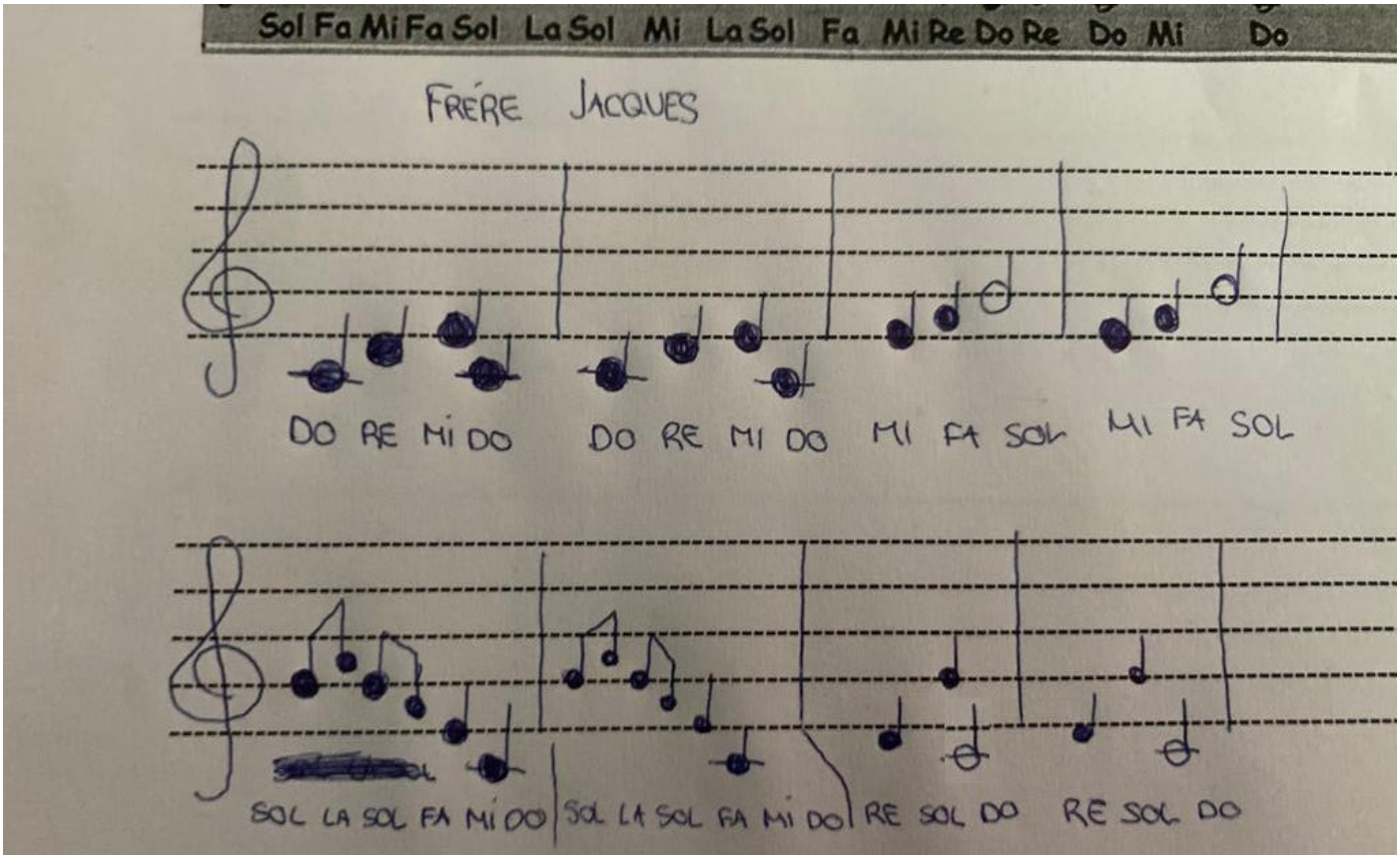
Añadimos cuatro bloques ”rest” que sumen 1/2 + 1/4 + 1/8 + 1/16 = 15/16. De esta forma, un bucle consume 1/16 por la duración del sonido más 15/16 del reposo. Y eso ya suma 16/16 = 1 pulso.

14.2. Melodía “Frere Jaques” con categoría “Música” de micro:bit

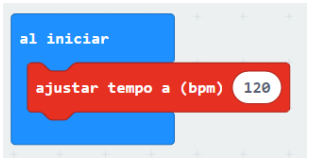
Tienes la descripción del código en: https://microbit.org/projects/make-it-code-it/frere-jacques-tune/

La siguiente imagen muestra el pentagrama de la melodía “Frere Jaques”. Una nota negra ocupa un pulso de duración. Mientras que una nota blanca ocupa dos pulsos. Tenemos un ritmo 4x4, por lo que encontramos cuatro notas negra en un mismo compás. Por ejemplo:

- En el primer compás del pentagrama encontramos la secuencia DO-RE-MI-DO. Cada nota es una negra y cada nota dura un pulso.
- En el tercer compás leemos RE-SOL-DO porque la nota DO es una blanca y dura dos pulsos. Una blanca dura el doble de una negra.

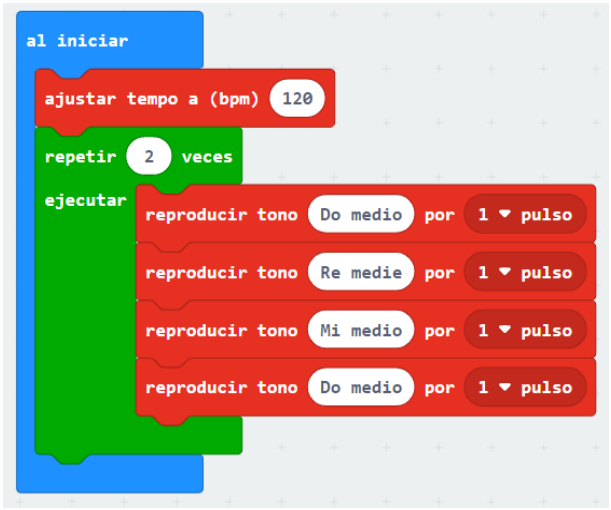


Ajustamos el tempo a 120 bpm al inicio del programa (ver imagen inferior).

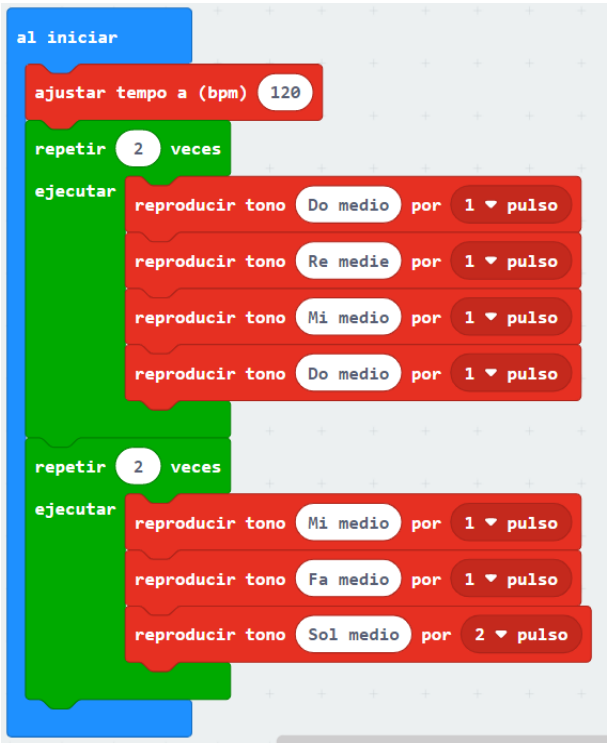


Añadimos tono “Do medio” con duración 1 pulso. A continuación elegimos un “Re medio”, luego “Mi medio” y terminamos con otro “Do medio”. Todas las notas con duración 1 pulso por ser negras.

Repetimos dos veces esta melodía de cuatro pulsos con un bloque “repetir”. Ajustamos la repetición a 2 veces y ya tendríamos los dos primeros compases (ver imágenes inferiores).

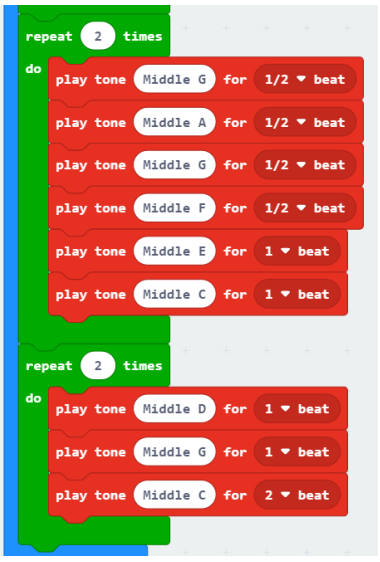


El tercer y el cuarto compás son idénticos. Cada compas contiene tres notas MI-FA-SOL. La nota final SOL dura 2 pulsos. Con ayuda, nuevamente, de un bloque “repetir” ahorramos bloques de código en nuestro programa (ver imagen siguiente).



En el quinto y sexto compás aparecen corcheas. Una corchea dura la mitad de tiempo de una negra. Por lo tanto, cada nota de la corchea dura 1/2 pulso, para que la suma de dos corcheas de como resultado 1/2 + 1/2 = 1 pulso. La imagen de la derecha muestra las notas en las escala anglosajona. Recuerda que:

- A → LA
- B → SI
- C → DO
- D → RE
- E → MI
- F → FA
- G → SOL



Finalmente, el séptimo y el octavo compás repiten la secuencia RE-SOL-DO, siendo la nota DO una nota blanca, por lo que tendrá dos pulsos de duración.

¡Y ya tendríamos la melodía completa!

14.3. Juego de reacción con placa micro:bit. Álgebra de Boole

Vamos a diseñar el siguiente juego. Dos jugadores (A y B) se retan a ser los primeros en pulsar un botón cuando la pantalla LED de micro:bit muestre el logotipo de un corazón. Para evitar dar un golpe demasiado fuerte sobre los interruptores de contacto de la placa, vamos a utilizar pinzas de cocodrilo

y papel aluminio para crear sobre un cartón los puntos de contacto, y llevar la señal eléctrica a los pines digitales de la placa. Nuestro propio cuerpo será el encargado de cerrar y abrir el circuito por donde viaje la señal eléctrica.

En primer lugar, debemos comprender cómo funcionan los pines situados en la parte inferior de la placa. Los **pines 0, 1 y 2 son pines de entrada y salida de señal digital**. Y pueden conectarse a otro dispositivo físico mediante pinzas de cocodrilo. Los pines simbolizados por **3V y GND** ("ground" en inglés, es decir, señal de tierra) **controlan la potencia eléctrica suministrada a la placa**. Es importante nunca unir con un cable el pin 3V con el pin GND, porque dañaríamos la placa para siempre. El siguiente vídeo resume las principales aplicaciones de estos pines (activar subtítulos en inglés): https://www.youtube.com/watch?v=EDgdHb0R96I

El funcionamiento del juego está basado en la propuesta siguiente:

https://microbit.org/es-es/projects/make-it-code-it/reaction-game/

Creamos una variable llamada *inicioJuego* (ver imagen derecha). Si su valor es Verdadero significa que el juego ha comenzado y los jugadores pueden pulsar los botones. Si su valor es Falso significa que el juego no ha comenzado y la pulsación de los jugadores que se adelanten al inicio del juego no será tenida en cuenta (así evitamos trampas de los jugadores).

En la categoría "Lógica" encuentras los bloques de variables booleanas. El **álgebra de Boole** es una forma de operar con variables que solo pueden tomar dos valores: Verdadero o Falso. Además, estas variables se relacionan entre sí con la intersección (conjunción y), con la unión (conjunción o) y con la negación (operador no). El álgebra de Boole está a la base del diseño de los ordenadores porque, si recuerdas, los bits solo pueden almacenar dos valores posibles (0 o 1). Por lo tanto, **podemos operar con los bits con el álgebra de Boole, asumiendo que 0 es Falso (F) y que 1 es Verdadero (V)**.

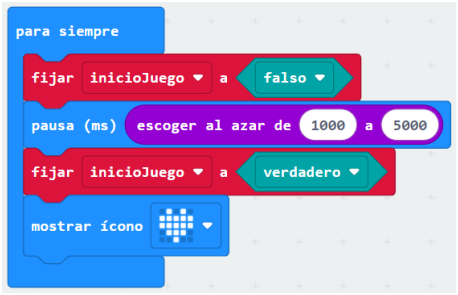
Las siguientes relaciones son útiles para comprender cómo operar con las variables booleanas:

- Negar algo Verdadero genera algo Falso.
- Negar algo Falso lo vuelve Verdadero.
- Verdadero y Verdadero es Verdadero. Intersección $V \cap V = V$.
- Verdadero y Falso es Falso. La intersección $V \cap F = F$ porque una de las dos variables de partida es falsa.
- Verdadero o Verdadero es Verdadero. Unión $V \cup V = V$.
- Verdadero o Falso es Verdadero. Unión $V \cup F = V$ porque al menos una de las dos variables de partida es verdadera.

Fijamos la variable *inicioJuego* a Falso porque el juego aún no ha comenzado. Fíjate que la forma hexagonal del bloque "falso" se puede introducir dentro de la forma ovalada del espacio contenido en el bloque "fijar". En esta ocasión, la forma de los bloques puede llevar a confusión.

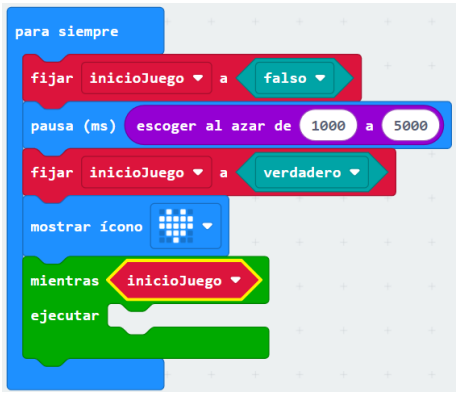
Generamos un número aleatorio entre 1.000 y 5.000, para saber los milisegundos que tardará el programa en mostrar en la pantalla Led el símbolo del corazón.

Una vez que ha pasado la pausa marcada por el bloque “pausa (ms) …” podemos jugar. Por lo que la variable *inicioJuego* pasa al valor lógico Verdadero y mostramos en pantalla la imagen del corazón.



En ese momento los jugadores recibirán el estímulo visual de la imagen de la pantalla y sabrán que el juego ha comenzado. Por lo que se lanzarán a ser los primeros en pulsar los botones de contacto.

El programa recogerá las pulsaciones de los jugadores siempre y cuando la variable *inicioJuego* sea Verdadera. Así evitamos la trampa de los jugadores que quieran adelantarse (como los atletas que salen ante del disparo de salida, en una carrea de velocidad) Por lo que **necesitamos un bloque “mientras” que se ejecutará solo si *inicioJuego* es Verdadero** (ver siguiente imagen).

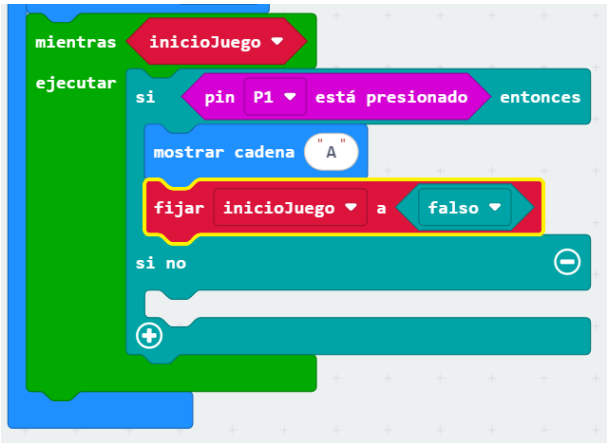


El jugador A controlará la señal eléctrica del Pin1. Mientras que **el jugador B actuará sobre el Pin2**. La siguiente imagen muestra cómo conectar los pines a los interruptores de papel aluminio sobre el cartón. El cartón de cada jugador necesita de dos trozos de papel de aluminio. El jugador debe tocar ambos trozos para cerrar el circuito eléctrico con su propio cuerpo. Uno de los trozos va conectado al pin GND (pinzas de cocodrilo de color blanco en la imagen inferior). Mientras que el otro trozo va conectado al Pin1 (jugador A) o al Pin2 (jugador B), como muestran las pinzas de color rojo y amarillo de la imagen.

Usamos un condicional “si … entonces … si no …” para comprobar, en primer lugar, si el jugador A ha cerrado el circuito del Pin1. Los bloques de pines están dentro de la categoría “Entrada”. La condición a cumplir es que Pin1 esté presionado. Eso significa que el jugador A debe estar tocando (de forma continua) los dos trozos de papel aluminio de su cartón.

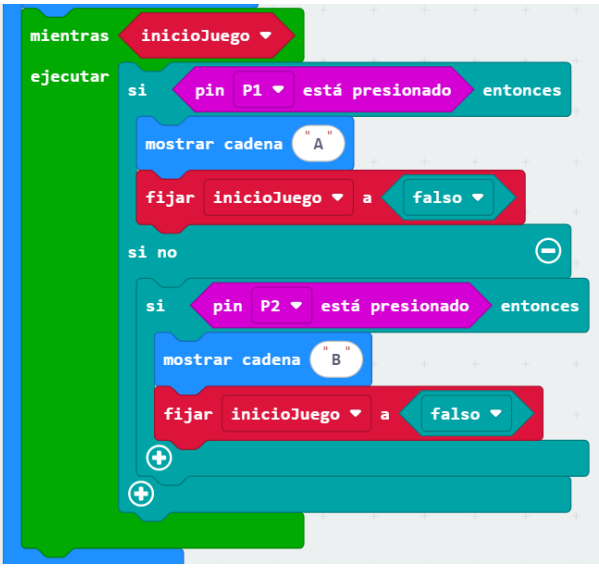
La expresión “está presionado” pide que los dos trozos de papel aluminio estén presionados coincidiendo en el tiempo, para que el juego funcione correctamente.

Si se confirma que Pin1 está presionado, la pantalla LED informa que ha ganado el jugador A y pone la variable *inicioJuego* a Falso para reiniciar el juego (ver imagen inferior).



Si Pin1 no está presionado, nos preguntamos si Pin2 está presionado. Usamos un condicional "si … entonces" dentro del condicional anterior. En esta ocasión, el mensaje a mostrar en pantalla sería que ha ganado el jugador B. Y nuevamente fijaríamos la variable la variable *inicioJuego* a Falso.

Un detalle. Los programas con condicionales se pueden resolver de diversas formas. Encadenando condicionales "si … entonces … si no …." unos dentro de otros, o bien usando nuevos bloques condiciones "si … entonces …". La preferencia de cada programador hará que le resulte más cómodo una opción u otra.



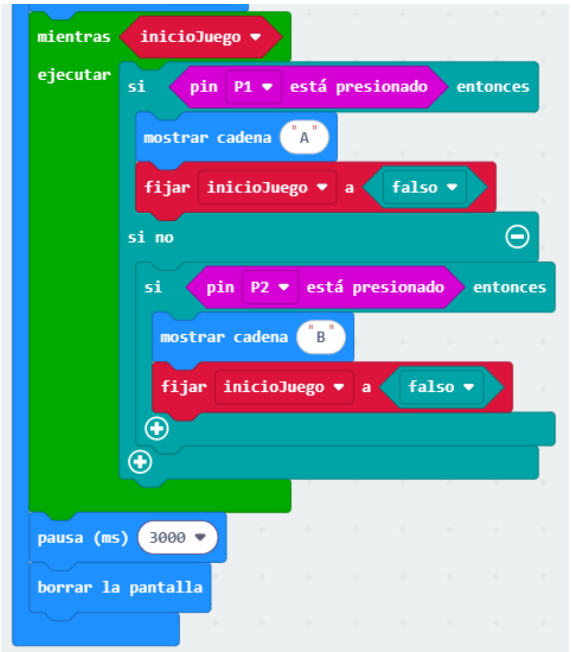
En el momento que uno de los dos jugadores ha ganado, y la variable *inicioJuego* ha pasado a Falso, salimos del bucle "mientras". Hacemos una pausa de unos segundos para que el mensaje del jugador ganador sea vea bien en la pantalla y borramos pantalla antes de reiniciar el juego (ver imagen de la derecha).

Este juego admite múltiples mejoras. Por ejemplo:

- Usar variables para llevar un contador de partidas ganadas por cada jugador.
- Añadir un cronómetro para informar del tiempo de reacción del jugador ganador en cada partida.
- Guardar un histórico con el tiempo de reacción más rápido.
- Añadir una señal sonora para informar acústicamente de que un jugador ha ganado.
- Añadir una señal sonora de error si algún jugador pulsa antes del inicio del juego.
- Añadir un tercer jugador haciendo uso del Pin0.

¿Te animas a ampliar tu código para cumplir algunos de estos requisitos?

En programación, la mejor manera de aprender es probar nuevas opciones sobre un programa base que ya dominamos. Esta dinámica de ampliar los programas, poco a poco, te acompañará en cualquier lenguaje de programación que aprendas en el futuro.



15. Actuadores. Controlar robot maqueen con la microcontroladora de la placa micro:bit

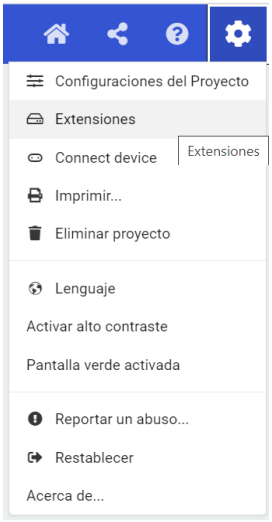
Un actuador son los elementos del robot que ejecutan las órdenes que reciben desde el ordenador. Estos dispositivos convierten la energía eléctrica en energía mecánica, luminosa o sonora. Ejemplos de actuadores son:

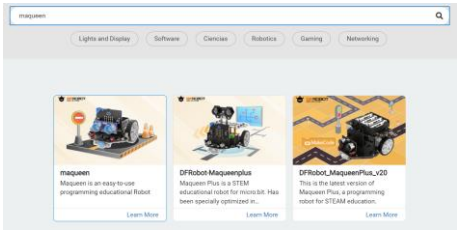
- Un motor: La energía eléctrica se convierte en energía mecánica que provoca el giro de un eje.
- Un servomotor: Es un motor de gran precisión que permite giros muy precisos en el intervalo desde los 0 grados hasta los 180 grados.
- Las luces LED (light emisor diode, en inglés): Emiten luces de color. Tienen polaridad, por lo que al conectarse al circuito eléctrico debe respetarse su parte positiva y su parte negativa.
- Un zumbador: Emite un sonido continuo de un mismo tono. También tiene polaridad.

Con la microcontroladora de la placa micro:bit podemos controlar el funcionamiento del robot maqueen. Además, este robot incorpora sensores nuevos como son el sensor de distancias por ultrasonido o el sensor infrarrojo.

15.1. Instalar extensión maqueen en MakeCode

Crea un nuevo proyecto en MakeCode y en la esquina superior derecha, donde está la rueda de configuración, pulsa para acceder a la opción “Extensiones” (ver imagen derecha). En la web que se abre tecleamos en el buscador “maqueen” y nos aparece la extensión a instalar para controlar el robot (ver imagen inferior).

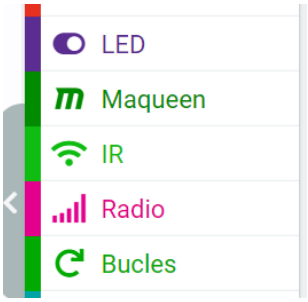




El robot se alimenta con tres pilas AAA, que se colocan en el portapilas. Este portapilas va adherido entre las dos ruedas del chasis. En la parte posterior del chasis encontramos una palanca grande de encendido/apagado. La placa se alimentará también desde el portapilas del robot, por lo que cuando conectemos la placa al chasis y activemos la palanca de encendido, comenzará a ejecutarse el programa que hayamos instalado en la placa.

La ranura de conexión por USB de la placa con el ordenador queda en la parte superior, por lo que es sencillo actualizar el programa en ejecución sin necesidad de desconectar la placa del chasis del robot.

Tras instalar la extensión maqueen, en el menú de categorías aparece una nueva categoría “Maqueen”. Contiene los bloques para controlar al robot con la placa micro:bit (ver imagen derecha).



Comencemos con un programa sencillo. Vamos a encender, de manera alterna, los dos led de la parte frontal del chasis. En la categoría “Maqueen” buscamos el bloque “LED” que permite ajustar el encendido y el apagado tanto del LED izquierda como del derecho.

Dentro de un bloque “para siempre” incluimos el bloque “LED” (ver imagen izquierda). Seleccionamos “derecho” y “encender”. E introducimos una pausa de 1.000 ms.

Acto seguido, apagamos el LED derecho y encendemos el izquierdo. Cada instrucción va en bloques separados. Esperamos nuevamente 1.000 ms para terminar apagando el LED izquierdo.

Este bucle se repetiría de manera indefinida por estar dentro del bloque “para siempre”. Si volcamos el programa en la placa micro:bit, la conectamos al robot y lo encendemos, veremos que ejecuta el programa de encendido y apagado que hemos diseñado.

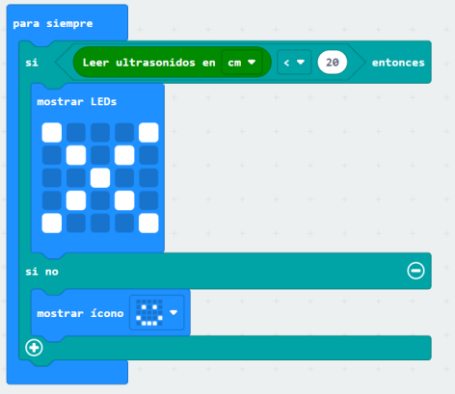
¡IMPORTANTE! Sé cuidadoso al manipular el robot. Sobre todo cuando comencemos a activar los motores, para evitar caídas desde la mesa de trabajo. El robot es muy frágil.

15.2. Sensor de distancias por ultrasonido

Vamos a poner en uso el sensor de distancias por ultrasonido que maqueen lleva incorporado en su parte delantera (en los cilindros que tienen la apariencia de los faros de un coche).

En la categoría “Maqueen” encontramos un bloque “Leer ultrasonidos en …”. Podemos introducir un número, que representará la distancia a partir del cual se activará el sensor de distancias.

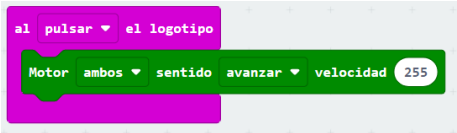
El código siguiente muestra en la pantalla LED de micro:bit el símbolo de una X si hay un objeto a una distancia menor o igual de 20 centímetros de los sensores. Y muestra una cara sonriente en caso contrario (ver imagen inferior). Puedes pasar tu propia mano por delante del robot para activar y desactivar el funcionamiento del sensor.



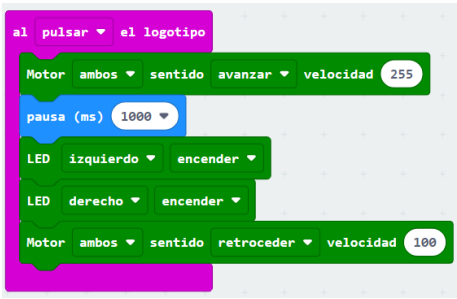
15.3. Activar motores del robot maqueen

Maqueen posee dos motores. Uno para cada rueda trasera. El bloque “Motor” permite elegir el motor a activar (izquierdo, derecho o ambos), permite avanzar o retroceder, y permite seleccionar la velocidad entre 256 valores posibles (el valor 0 significa parado y el valor 255 velocidad máxima). Estos valores de velocidad son valores relativos; sirven para hacernos una idea de cuándo va más rápido y de cuándo va más lento. Lógicamente, la velocidad real en m/s depende del nivel de las baterías.

Con el código de la imagen de la derecha, al pulsar sobre el logotipo (ya vimos que el logotipo funciona como un interruptor capacitivo) los dos motores se activan hacia delante, a máxima velocidad. Idealmente, el robot debería avanzar el línea recta. Pero los desequilibrios al montar los neumáticos, el polvo del suelo que suele acumular el eje de giro y el desnivel del propio suelo hacen prácticamente inviable que Maqueen, con este sencillo programa, vaya exactamente en línea recta.



Ampliamos este programa incluyendo lo siguiente: tras pasar 1 segundo (1.000 ms) en avance recto, vamos a encender ambos LED y retroceder por espacio de una décima de segundo (100 ms). Muy importante no confundir “avanzar” con “retroceder” y tener muy claro si activamos el motor derecho, el izquierdo o los dos a la vez.



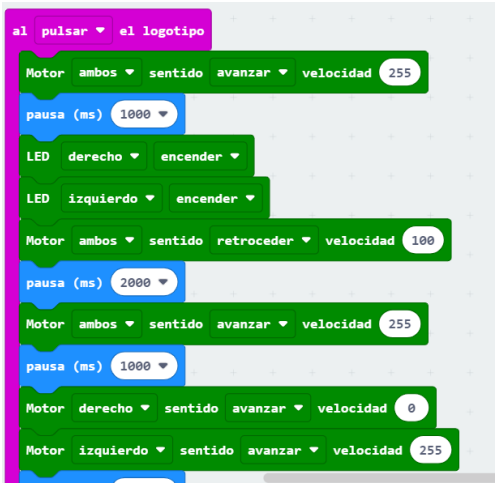
Podemos jugar con las distintas opciones del bloque “Motor” para conseguir giros. Si fijamos a 0 la velocidad de un motor y seguimos avanzando con el otro motor, maqueen gira. Es decir:

- Si queremos girar a la derecha, dejamos a 0 el motor derecho y avanzamos con el izquierdo.
- Si queremos girar a la izquierda, dejamos a 0 el motor izquierdo y avanzamos con el derecho.

A modo de ejemplo ejecutemos el siguiente pseudocódigo en MakeCode:

1. Avanzar recto por 1 segundo a velocidad 255.
2. Retroceder por 2 segundos a velocidad 100, con LEDs frontales encendidos.
3. Avanzar recto por 1 segundo a velocidad 255.
4. Girar a la derecha durante 1 segundo.
5. Activar de nuevo el motor derecho para que el robot avance recto 1 segundo.
6. Girar a la izquierda durante 1 segundo.
7. Activar el motor izquierdo para que el robot mantenga avance recto de manera indefinida.

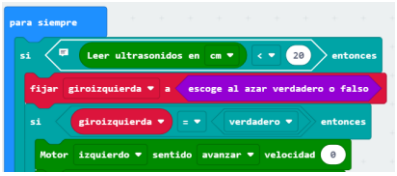
Las imágenes siguientes muestran la programación por bloques de este pseudocódigo.

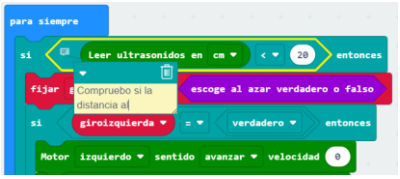


15.4. Introducir comentarios en el código de bloques de MakeCode

Ya reflexionamos en Scratch que, al crear un programa largo, es probable que con el paso de los días no nos acordemos bien de la lógica del programa. Para ello es muy recomendable introducir comentarios: texto escrito que explique qué estamos haciendo en un punto concreto del código.

Cuando introduces un bloque en MakeCode puedes pulsar sobre él con el botón derecho, elegir “Añadir comentario” y teclear en un cuadro de texto (ver ejemplos de las imágenes inferiores).



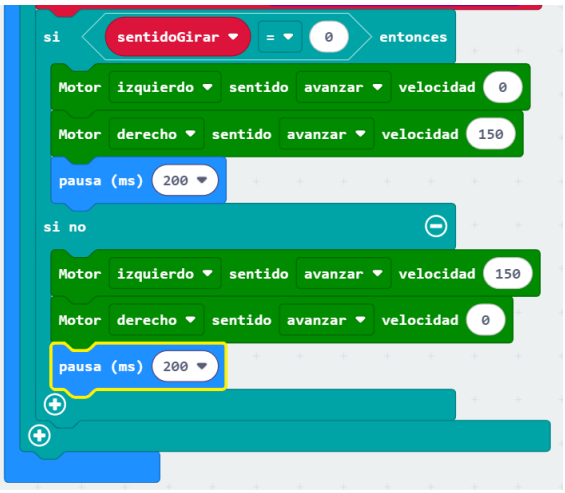


16. Evitar obstáculos con robot maqueen

Usando el sensor de ultrasonidos de la parte delantera, podemos introducir en el bloque “para siempre” un condicional que continuamente decida si la distancia detectada es inferior o no a 20 centímetros. En caso afirmativo, de manera aleatoria, se gira a izquierda o a derecha y se avanza en línea recta durante un tiempo determinado. En caso negativo, el vehículo prosigue su movimiento en línea recta. Como el bloque “para siempre” se ejecuta continuamente, el condicional estará siempre operativo para detectar posibles obstáculos.

La imagen superior muestra un condicional “si … entonces …” que activa su condición si el sensor de ultrasonidos detectar un obstáculos a una distancia inferior a 20 cm. Con una variable *sentidoGirar* (creada en la categoría “Variables”) y con un número aleatorio, vamos a decidir si giramos a la izquierda (valor 0) o giramos a la derecha (valor 1).

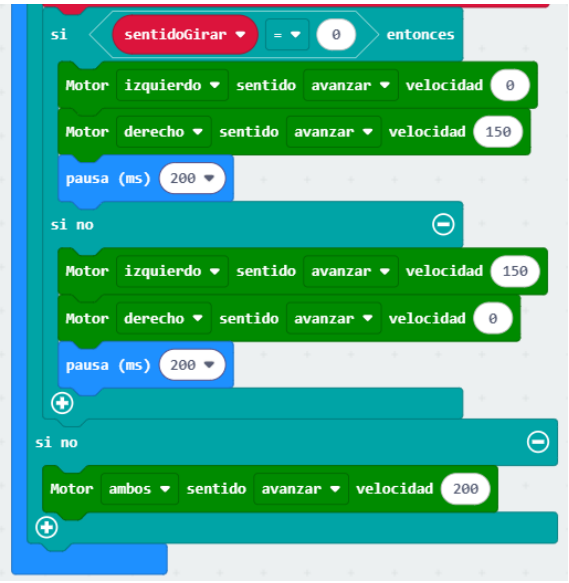
Con un nuevo condicional decidimos si giramos a la izquierda o a la derecha, según el valor que haya tomado de forma aleatoria la variable *sentidoGirar*. La duración temporal del giro será de 200 ms (ver imagen derecha). Fíjate que este nuevo condicional solo pregunta si la variable es igual a 0. Si la condición es cierta, significa que el giro debe ser a la izquierda. Y en caso contrario, la variable solo puede tomar el valor 1 porque el número aleatorio creado solo toma valores 0 o 1. Y si la variable vale 1 por descarte, la única opción posible es que el giro sea hacia la derecha.



Para terminar el programa, nos damos cuenta de que entramos en el primer condicional del bloque “para siempre” si la distancia detectada por el sensor es inferior a 20 cm. En caso contrario, significa que no hay obstáculo y el robot debe avanzar el línea recta. Por lo tanto, si no se cumple el primer condicional, deberemos activar hacia delante ambos motores en línea recta (ver imagen inferior).

¡Ojo! Cuando el programa de bloques se alarga demasiado verticalmente, puede ser difícil detectar visualmente qué parte o partes engloba cada condicional. Es una causa de error bastante frecuente pulsar en el botón + de extensión de condicionales en los bloques que no son los correctos.

Como siempre, si algo no funciona en tu programa, no desesperes. El ordenador no se equivoca. Somos nosotros los que, con paciencia y práctica, debemos depurar poco a poco nuestros errores.



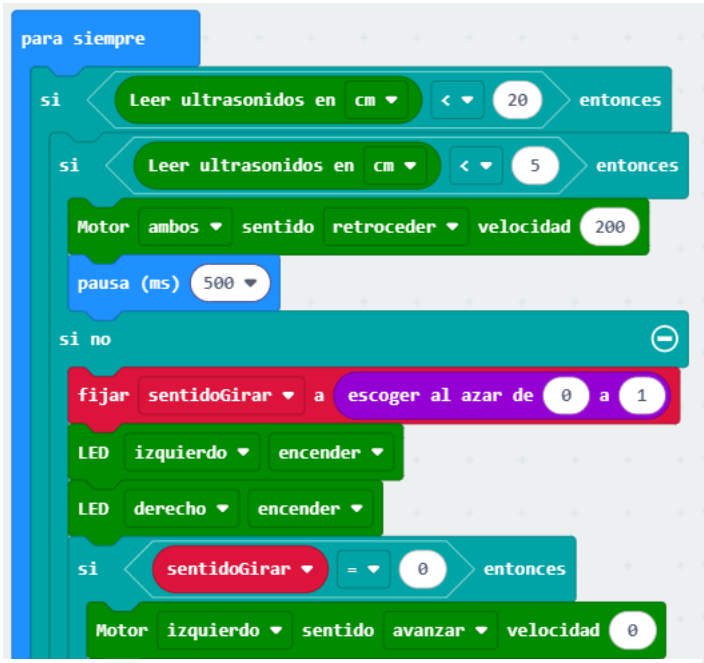
16.1. Mejorar el programa de evitar obstáculos

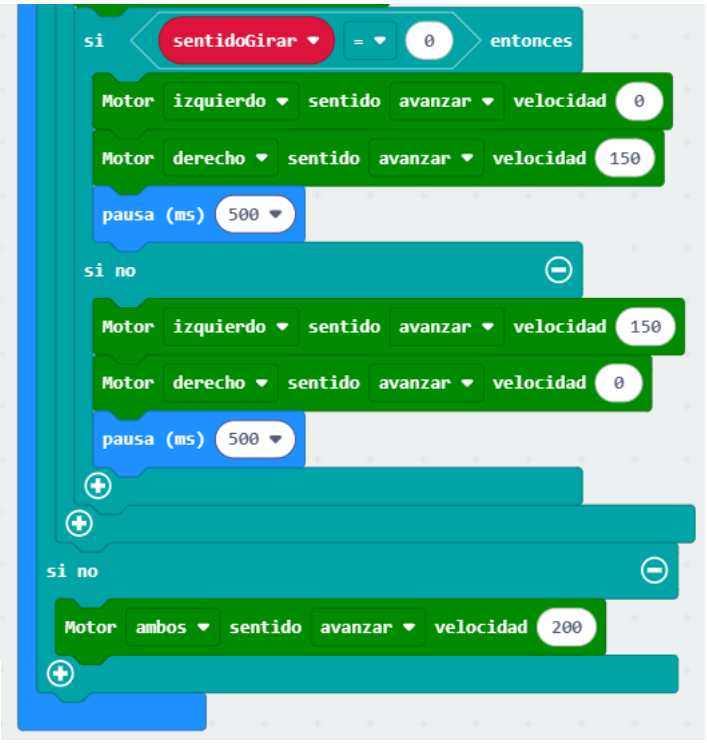
El código anterior tiene un serio problema: si el obstáculo es muy ancho (por ejemplo, una pared), maqueen gira al acercarse a la pared, pero tras girar apenas 200 ms, si el obstáculo no queda justo frente al sensor de distancias, el robot vuelve a avanzar en línea recta y puede terminar chocando lateralmente con el obstáculo y atascándose.

Es decir, con el programa anterior conseguimos evitar objetos pequeños y muy separados entre sí (por ejemplo, lastas o botellas situadas verticalmente sobre el suelo y con espacio amplio de separación entre ellas). Pero con una caja de zapatos o una mochila, lo normal es que maqueen termine impactando con ellos si no mejoramos nuestro código.

¿Qué podemos hacer? Si el coche no encuentra obstáculos, mantendremos alegremente su camino en línea recta. Pero si detecta un obstáculo a una distancia inferior a 20 cm, **añadimos un segundo condicional más restrictivo, por si el obstáculo se encuentra a menos de 5 cm para que el coche retroceda durante 500 ms**.

Además, podemos encender los leds delanteros para indicar visualmente cuándo el coche ha detectado un obstáculo a menos de 20 cm y aumentar el tiempo de giro de 200 ms a 500 ms, para así superar los obstáculos más anchos ya con el primer condicional. Las dos imágenes inferiores muestran el código completo de programación.





16.2. Sensor de luz para decidir el movimiento del robot maqueen

Imagina la siguiente situación: En una habitación a oscuras, o con poca luz, aplicamos un foco de luz (una linterna, por ejemplo). Y deseamos que maqueen, con ayuda del sensor de luz insertado en la placa micro:bit, se dirija automáticamente hacia el origen del foco de luz.

Instalamos, como otras veces, la extensión maqueen en MakeCode. Creamos dos variables. Una para guardar el valor actualizado del sensor de luz (*cantidadLuz*) y otra para determinar el sentido de giro del coche (*girar*), en búsqueda de la máxima iluminación posible. Ambas variables las iniciamos al valor 0 (ver imagen de la derecha), dentro del bloque "al iniciar".

Dentro del bloque "para siempre", ajustamos el brillo de los LED de micro:bit al nivel recibido por el sensor de luz (ver imagen de la izquierda). De esta forma, la intensidad de los LED será proporcional, en una escala de 0 a 255, al nivel de luz recogido por el sensor.

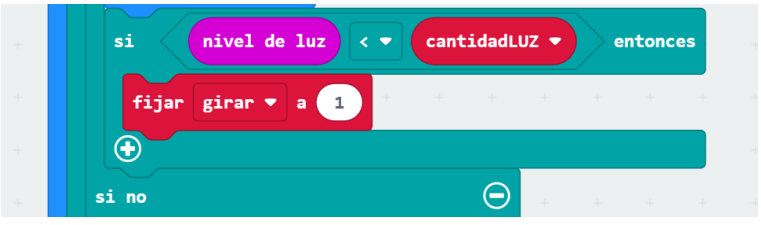
Ahora toca pensar. Nuestra variable *cantidadLUZ* toma el valor 0 al iniciarse el programa. Por lo tanto, a poca cantidad de luz que haya en la habitación, el valor del sensor de luz será mayor que este valor 0. En este caso, moveremos en línea recta a maqueen para que se acerque al probable foco emisor.

Actualizaremos el valor de la variable *cantidadLUZ* y esperaremos una cantidad de tiempo (una décima de segundo) antes de analizar otra vez el valor del sensor de luz. Como muestra la siguiente imagen, esta lógica de programación está contenida en un bloque condicional.

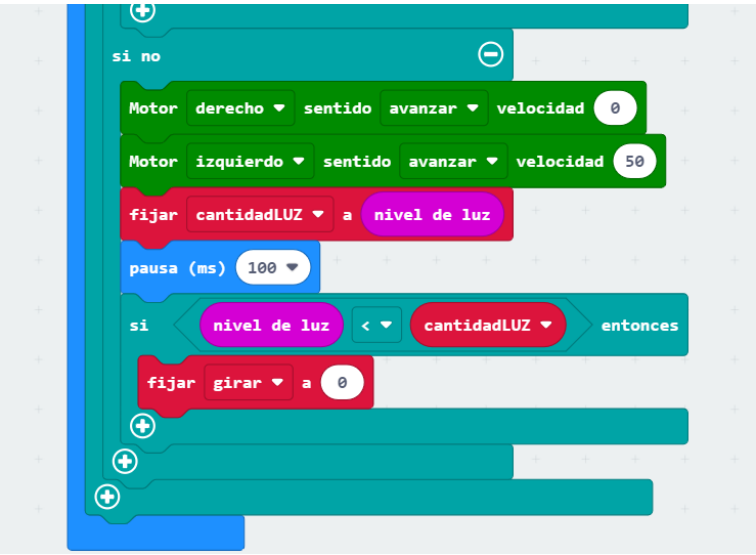
Si la condición no se cumple, es decir, si el valor del sensor de luz es menor que el valor almacenado en *cantidadLUZ* significa que nos estamos alejando del foco emisor. Por lo que giraremos el robot maqueen para reorientar su dirección.

Si la variable *girar* vale 0, giraremos a la izquierda. Como al iniciar el programa fijamos el valor de la variable a 0, significa que el primer giro siempre será hacia la izquierda. Actualizaremos nuevamente el valor de la variable "cantidadLUZ" y esperaremos nuevamente otra décima de segundo antes de seguir con el programa.

Pasada la décima de segundo, tendremos que preguntarnos si el valor del sensor de luz sigue siendo inferior al valor actualizado anteriormente en *cantidadLUZ*. En caso afirmativo, tendremos que girar en sentido contrario (hacia la derecha). Para ello, actualizamos la variable *girar* al valor 1.



La parte final del código es idéntica a los bloques contenidos en el condicional “si … entonces …”, con la salvedad de que giraremos a la derecha y de que la variable *girar* pasaría a tomar el valor 0, para girar en sentido contrario cuando se repta el bucle “para siempre”. Puedes probar el funcionamiento con una linterna que apunte al sensor de luz de la placa de micro:bit, situado en los LEDs de la pantalla.



Retos para resolver

Siempre, siempre, siempre, siempre debes anotar en tu cuaderno las explicaciones de clase y los esquemas y ejemplos que el profesor escriba o proyecte en la pizarra. Si no tienes el cuaderno limpio, completo y ordenado con las explicaciones de clase, los siguientes retos no serán calificados.

Todos los retos deben estar resueltos en el cuaderno, salvo los archivos informáticos con código de programación que el profesor te indique que el enseñes en clase o que le envíes por chat privado de Teams, para que pueda comprobar el correcto funcionamiento de los programas.

Retos de la Sesión 9

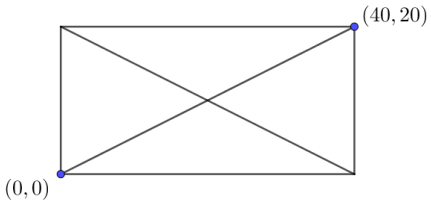
9.1. El profesor te entregará los dos discos del cifrado César. Recorta los discos y pégalos en tu cuaderno de tal forma que la letra A del disco interior (letras originales) coincida con la letra L del disco exterior (letras cifradas). Así conseguirás un cifrado de 11 posiciones hacia la derecha. Usando los discos, escribe de manera cifrada las siguientes palabras: TELESCOPIO, UNIVERSO, ZAPATO, NIÑO.

9.2. Realiza en Scratch el programa de codificación César que hemos explicado en clase, y cuyo código por bloques aparece al final de la sesión. ¡Ojo! Este programa tiene sus limitaciones. Hay letras que no se codifican bien. ¿Qué pasa si, al sumar el desplazamiento, superamos la cantidad 27 (posición de la letra Z) ¿Se te ocurre como controlar, con un condicional, este supuesto y cómo mejorar el código del programa? Responde en tu cuaderno con tu respuesta a estas dos cuestiones.

Retos de la Sesión 10

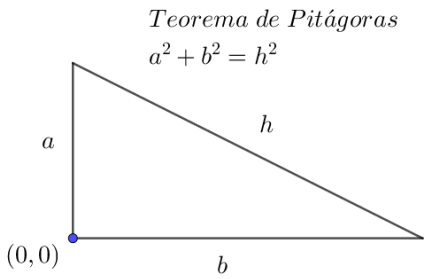
10.1. Con ayuda de los bloques a color recortados, crea los códigos de programación Scratch para diseñar los siguientes programas. Puedes consultar tu cuaderno personal, pero no usar el ordenador ni los tutoriales online para resolver los retos. Se valorará en la calificación los códigos que resuelvan los retos empleando la menor cantidad de bloques posibles. Además, en tu cuaderno es obligatorio que dibujes el código de programación por bloques que resuelve cada reto. Recuerda que el profesor puede preguntarte para que le expliques cualquier parte del código que presentes.

1. El campo de fútbol sala tiene dimensiones 40 pasos de largo por 20 pasos de ancho. Tomamos la cantidad "paso" de Scratch como unidad de referencia. Uno de los córner representa la posición (0,0) del escenario de Scratch. Colocamos en ese córner al objeto del gatito. Deseamos pintar en el escenario de Scratch las siguientes líneas del campo de fútbol, con la condición de que cada línea solo se puede pintar una única vez y que el color de las líneas diagonales debe ser diferente al color de las líneas de los lados del rectángulo.



2. Un tablero de ajedrez está formado por 8 filas y 8 columnas. En total, hay 64 casillas. Imagina que colocamos un grano de arroz en una de las casillas de la esquina. En la siguiente casilla, colocamos 2 granos de arroz. Y en la siguiente casilla, 4 granos de arroz. Y en la siguiente, 8 granos. Y en la siguiente 16. Así, hasta llegar a la última casilla. El programa de Scratch debe pedir al usuario el número de casilla y, tras realizar los cálculos necesarios, devolver por un mensaje de pantalla la cantidad de granos de arroz acumulados en la casilla elegida por el usuario.

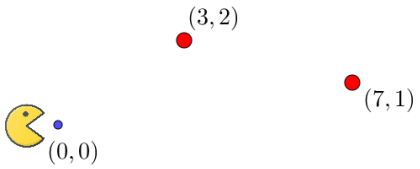
3. El un triángulo rectángulo, los dos lados que son perpendiculares entre sí se llaman catetos. Y el tercer lado, que es el más largo de los tres, se llama hipotenusa. Si un cateto se llama a, el otro cateto b y la hipotenusa h, el Teorema de Pitágoras afirma que se cumple la siguiente relación matemática: $a^2 + b^2 = h^2$. El programa debe pedir al usuario que introduzca los valores de los catetos a y b. El programa debe devolver, con un mensaje de texto, el valor de h^2. Además, el programa debe dibujar sobre el escenario de Scratch el triángulo con las medidas introducidas por el usuario. El vértice que une los dos catetos debe estar situado en la posición (0,0) del escenario. Y las líneas del triángulo solo se pueden dibujar una sola vez.



4. Colocamos el gatito de Scratch sobre la casilla sombreada con la letra K. Esa casilla representa en el escenario la posición (0,0). Cada desplazamiento a una casilla vecina, ya sea a derecha o a izquierda, es una longitud de un "paso" de Scratch. Generamos un número aleatorio entero entre 1 y 10. Si sale par, nos movemos un paso hacia la derecha. SI sale impar, nos movemos un paso hacia la izquierda. Repetimos este procedimiento del número aleatorio y del desplazamiento un total de cuatro veces. Haz un código de Scratch que realice esta secuencia y que muestre en pantalla la letra de las casillas por dónde va pasando el objeto en cada repetición. Además, responde en tu cuaderno a las siguientes preguntas: ¿Es igual de probable que el gatito acabe en cualquier casilla de la tabla? ¿Hay casillas más probables que otras? Explica de manera razonada tu argumentación.

G	H	I	J	K	L	M	N	Ñ

5. La siguiente imagen muestra el escenario de Scratch, con un objeto simbolizado por un comecocos. Este objeto está situado en la posición (0,0) del escenario. Diseña un programa donde, con las teclas de las flechas derecha, izquierda, arriba y abajo el comecocos pueda moverse por el escenario y comerse los puntos rojos. Cada pulsación de la flecha implica un avance de un "paso" de longitud en Scratch. El programa debe mostrar el mensaje "OK" cuando el comecocos llegue a un punto rojo. Además, responde a las siguientes preguntas: ¿Cuántas veces, como mínimo, deberemos pulsar las flechas del teclado para que el comecocos vaya desde la posición (0,0) a la posición (3,2)? ¿Y para ir de la posición (3,2) a la posición (7,1)? Razona de manera argumentativa tus respuestas.



Retos de la Sesión 11

11.1. Continuamos con los retos de la sesión anterior. Estos retos forman una única actividad de calificación en la evaluación de la asignatura.

Retos de la Sesión 12

12.1. Programa en MakeCode el dado con puntos iluminados en la pantalla LEDs que se explica al final de la sesión. Enséñalo al profesor en tiempo de clase. Ya sabes que el profesor puede preguntarte para que le expliques cualquier parte del código. Además, dibuja el diagrama de flujo de este programa del dado. Este diagrama debe recoger la ejecución del código tras agitar el usuario la placa. ¿Cómo puedes dibujar, en el diagrama de flujo, la condición "si agitado"? ¿Cómo puedes dibujar el mensaje de salida que representa la pantalla LED encendida?

Retos de la Sesión 13

13.1. Programa en MakeCode el ejemplo descrito en la sesión con el sensor de luz, mostrando en la pantalla LED el dibujo de un Sol si la intensidad de luz es mayor o igual que el nivel de referencia 100. Enséñalo al profesor en tiempo de clase . Además, copia en tu cuaderno el código de bloques de MakeCode de este programa.

13.2. Diseña en MakeCode el programa que recoge las variaciones de temperatura máxima y mínima, y que hemos explicado en clase. Enséñalo al profesor en tiempo de clase. Recuerda que puede preguntarte por cualquier aspecto del código. Además, copia en tu cuaderno la tabla que hemos explicado en clase que explica cómo funciona el condicional para actualizar el valor de la temperatura mínima.

13.3. Crea en MakeCode el programa que utiliza una función para convertir grados Centígrados en grados Fahrenheit que hemos trabajado en la sesión. Enséñalo al profesor en tiempo de clase. Recuerda que puede preguntarte por cualquier aspecto del código.

Retos de la Sesión 14

14.1. Enseña al profesor, en tiempo de clase, tu programa MakeCode con la meolodía "Frere Jaque" sonando en la placa micro:bit, tal y como lo hemos explicado en la sesión. Además, debes copiar en tu cuaderno la partitura de la melodía.

14.2. Crea con MakeCode el programa del juego de reacción que explicamos en la sesión. Enséñalo al profesor en tiempo de clase. El profesor entregará pinzas de cocodrilo, cartón, papel aluminio y pegamento para montar el circuito con la placa micro:bit. Además, debes copiar en tu cuaderno el código de bloques empleado. Cuida la estética, presentación, orden y limpieza al dibujar los bloques.

Retos de la Sesión 15

15.1. Programa en MakeCode el programa con el sensor de distancias por ultrasonido que hemos trabajado en la sesión. Además, copia en tu cuaderno el código de bloques empleado. Cuida la estética, presentación, orden y limpieza al dibujar los bloques. Recuerda que debes enseñarlo al profesor en tiempo de clase.

15.2. Muestra el robot maqueen funcionando con el programa de activación de motores que hemos explicado en la sesión. Además, en la zona de edición de MakeCode, debes introducir al menos dos comentarios aclaratorios sobre el funcionamiento del código .Recuerda que debes enseñarlo al profesor en tiempo de clase.

15.3. Con lo que has aprendido en esta sesión, debes crear por ti mismo el siguiente programa: el robot maqueen debe avanzar en línea recta y parar por completo cuando detecte un obstáculo a una distancia inferior de 5 centímetros. Recomendación: no pongas el robot a velocidad máxima, para evitar tropiezos o golpes en clase. Enseña al profesor el programa funcionando en tiempo de clase.

Retos de la Sesión 16

16.1. Crea un diagrama de flujo, lo más completo posible, que ilustre el primer programa de la sesión sobre evitar obstáculos con condicionales y con la variable *sentidoGirar* que toma, de manera aleatoria, el valor 0 o 1 para decidir si girar a la izquierda o a la derecha. Además, debes enseñar al profesor el programa funcionando correctamente en tiempo de clase. El profesor situará en un espacio diáfano un conjunto de latas y botellas de plástico, bastante separadas entre sí, para que maqueen pueda detectarlas y esquivarlas.

16.2. Introduce las mejores que hemos visto en la sesión sobre el programa de evitar obstáculos de mayor tamaño. Enseña al profesor en tiempo de clase el robot funcionando. Recuerda que el profesor puede preguntarte en cualquier momento para que el expliques cualquier parte del código de programación.

16.3. Enseña al profesor funcionando, en tiempo de clase, el programa seguidor de luz con micro:bit y maqueen.