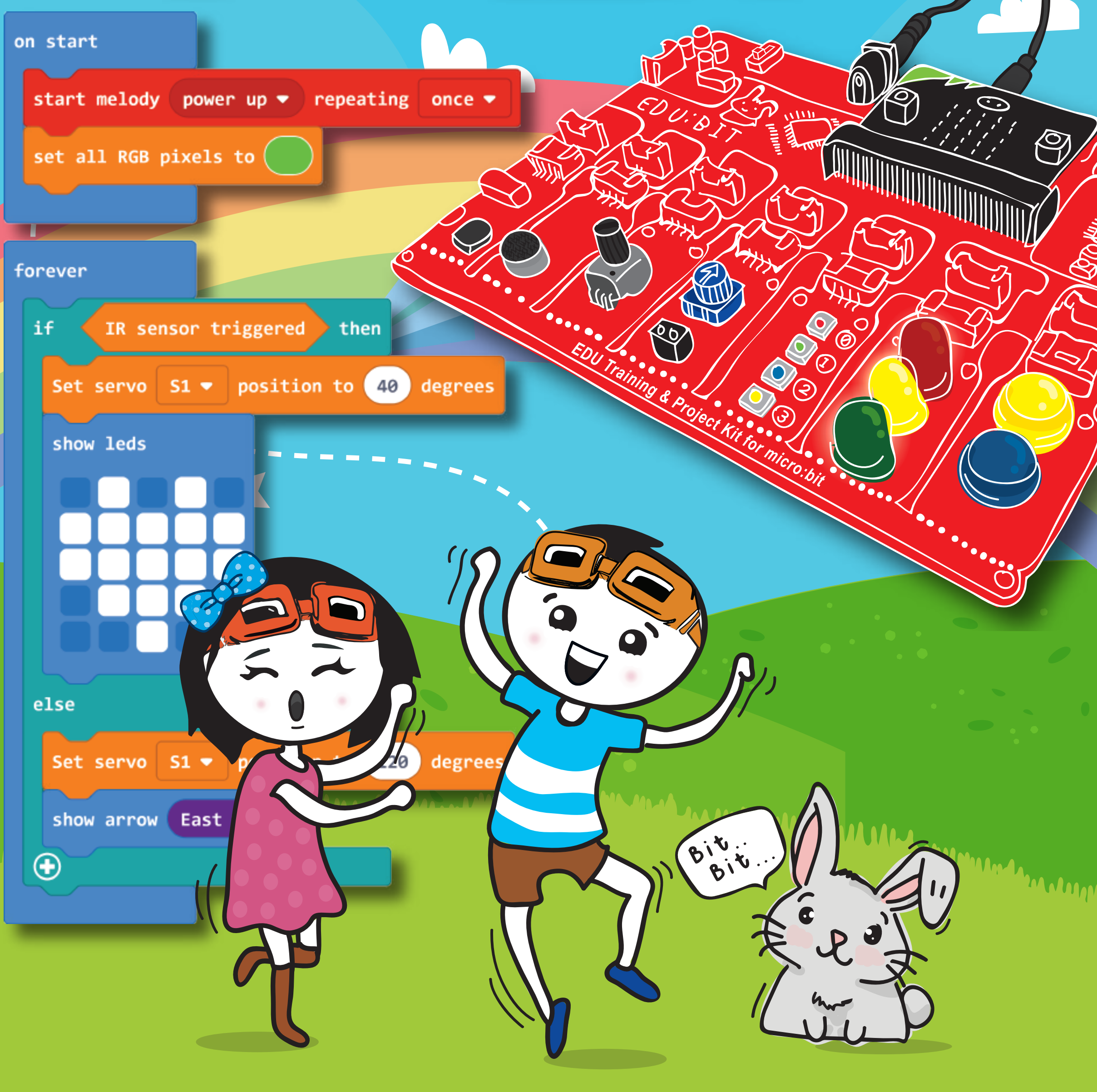
Cytron Technologies
CLASSROOM STEAM EDUCATION

Explore STEM & Coding with EDU:BIT



```
on start
    start melody  power up ▾  repeating  once ▾
    set all RGB pixels to ●

forever
    if  IR sensor triggered  then
        Set servo  S1 ▾  position to  40  degrees
        show leds
    else
        Set servo  S1 ▾  p  20  degrees
        show arrow  East
    ⊕
```

NOTE FROM RERO EDUTEAM @ CYTRON

Dear ____________ ,
(Child's Name)

Have you heard of micro:bit? It's a tiny programmable board designed in the UK and distributed widely around the world to encourage kids to learn coding in a fun and easy manner.

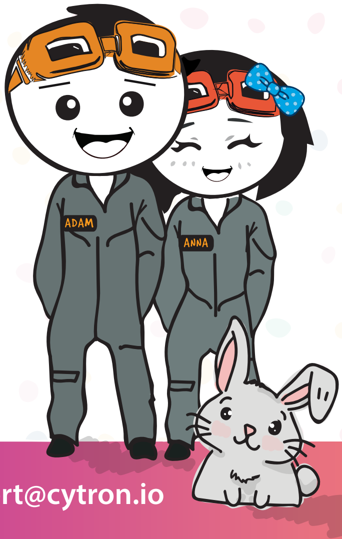
Engineers at Cytron took it up a notch by building this EDU:BIT board so that you can learn coding bit by bit. You have **Music Bit** with piezo buzzer and audio jack for you to play music, **Sound Bit** to detect noise, **Potentio Bit** for analog control, **IR Bit** to detect object, **RGB Bit** for colourful light display, **Traffic Light Bit** with red, yellow and green LEDs, and finally **Button Bit**, the up-sized version of the push-buttons on micro:bit board. With this kit, you also get a **DC Motor** and a **Servo Motor** to play with. How cool is that?!

Let's start, shall we? In the following pages, we are going to explore and recreate some classic childhood games such as *Rock Paper Scissors, Snakes and Ladders Game, Tag, You're It!, Talent Time Show, Twister, Simon Says* and other fun games. Follow the step-by-step guide to build the games and then have fun playing with your friends! Feel free to modify the code to create your own upgraded version of the games.

At the end of each chapter, there will be a challenge which requires you to apply what you've learned to build an application for your classroom. Give it a go and if you're stuck, we're always here to help you.

Are you ready? Let's embark on this exciting journey
- have fun learning and exploring!

Cheers,
Adam & Anna



Explore STEM and Coding with EDU:BIT Training & Project Kit

Written by
Cheryl Ng, SC Lim & Adrian Teo

Content tested & approved by
Joshayne D. Lim (7 years old)

Illustrated by
Suhana Oazmi

Fourth Printing, May 2021

Published by Cytron Technologies

To request permissions, contact the publisher at support@cytron.io.

ISBN 978-967-19475-0-0

First printing Aug 2020
Second printing Sept 2020
Third printing Oct 2020
Fourth printing May 2021

Published by:

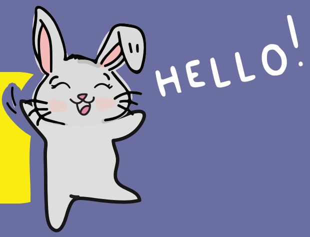
Cytron Technologies Sdn. Bhd.
1, Lorong Industri Impian 1,
Taman Industri Impian,
14000 Bukit Mertajam,
Penang, Malaysia.

Tel: +604-5480668
Fax: +604-5480669

www.cytron.io

Printed in Malaysia.

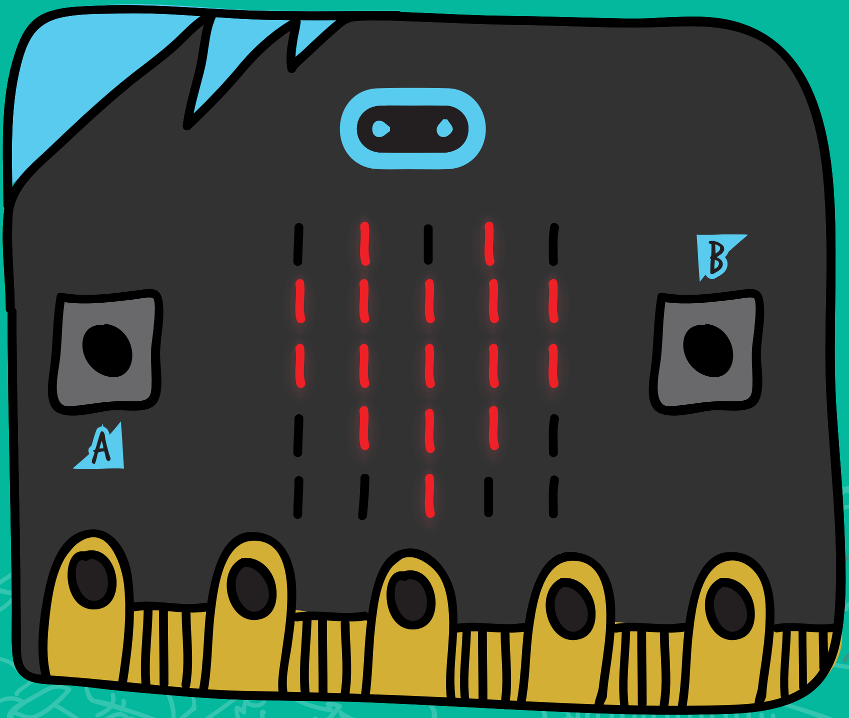
Content

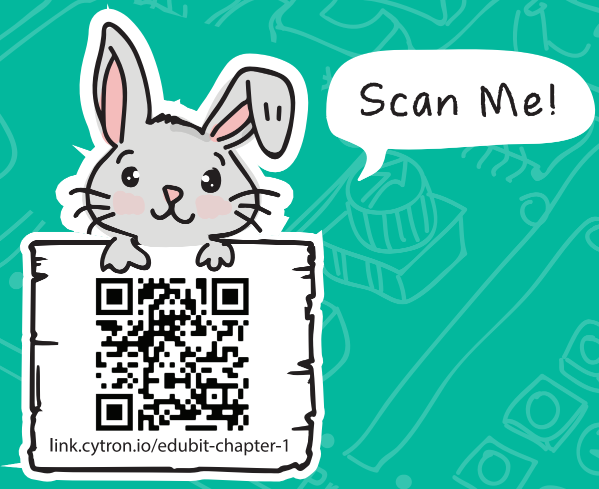


CHAPTER 1

> Hello, world!_

LED matrix on micro:bit

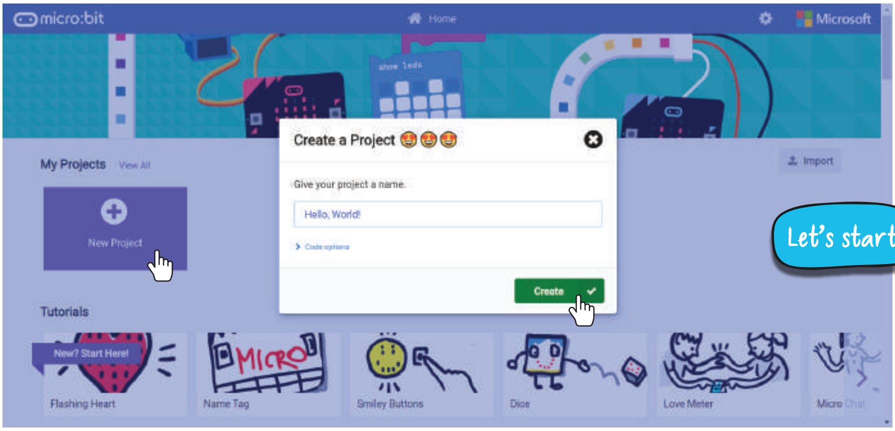




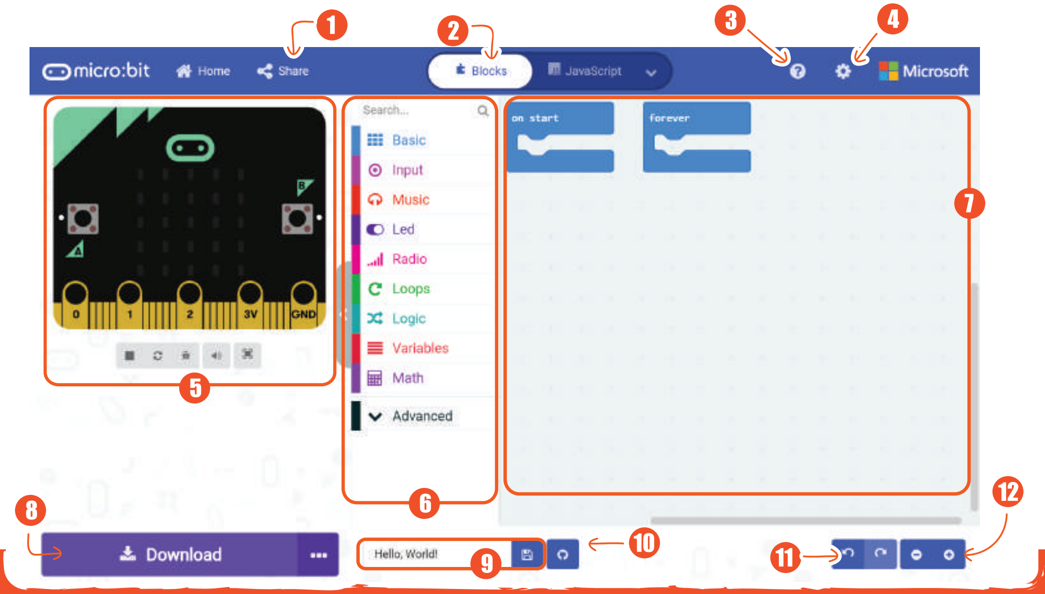
link.cytron.io/edubit-chapter-1

LET'S CODE!

Step 1 Open your browser and go to https://makecode.microbit.org/
Click '**New Project**'. Type in your project name and then click **'Create'.**

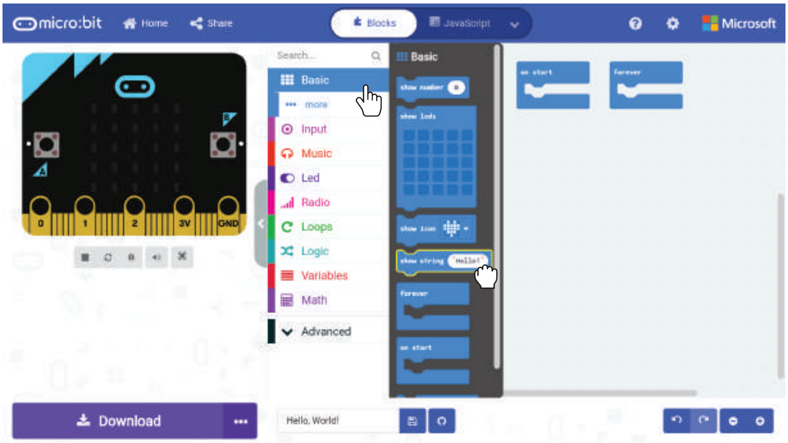


You will see this Microsoft MakeCode Editor page which allows you to easily program your EDU:BIT using drag-and-drop method.

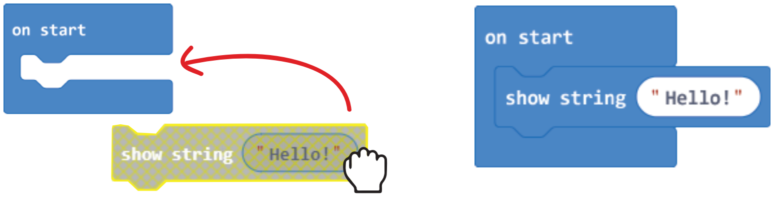


1) Publish and share your project.
2) Choose to program in Blocks, JavaScript or Python.
3) Open Help menu.
4) Change settings, add extensions and pair device.
5) **SIMULATOR** - Show you what your program will look like running on a micro:bit.
6) **TOOLBOX / CATEGORY DRAWER** - Click to see available coding blocks in each category. All blocks from the same category have the same colour.
7) **PROGRAMMING WORKSPACE** - Programs are constructed by snapping blocks together in this area.
8) Download your code to the micro:bit.
9) Name and save current project to your computer.
10) Create GitHub repository.
11) Undo/Redo.
12) Zoom in/out

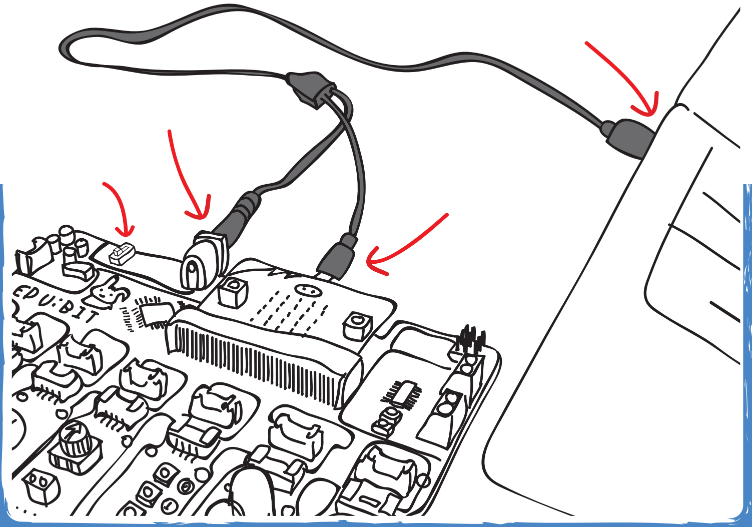
Step 2 Click [Basic] and then select [show string] block.



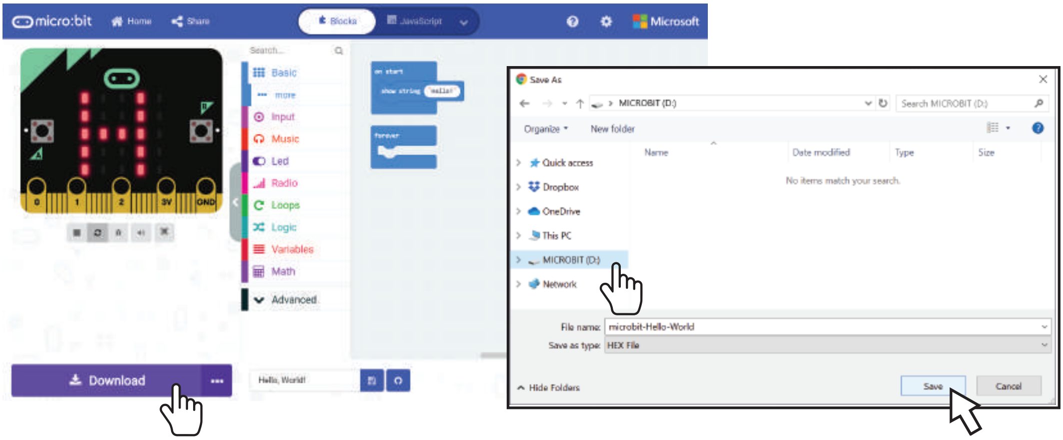
Step 3 Click and snap the [show string] block to the [on start] slot.



Step 4 Connect the USB cable to your computer and EDU:BIT as shown. Remember to power up EDU:BIT by sliding the switch to ON.



Step 5 Click **[Download]** button. In the pop up window, choose to download your project to the MICROBIT drive. Close the window when it says "Download completed".



Hello, World!

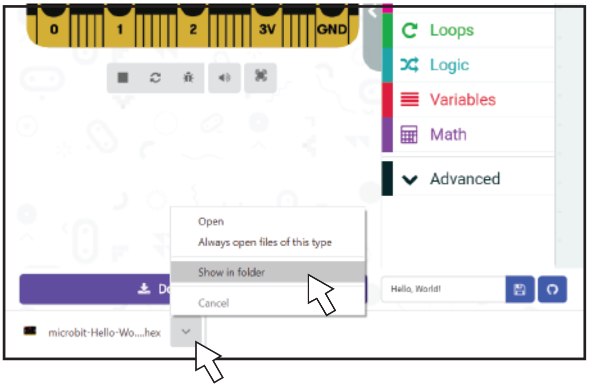
File name: microbit-Hello-World

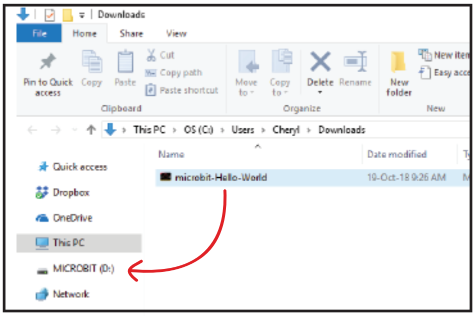
Save as type: HEX File

This process of transferring code is called **Flashing**. The orange LED on the back of the micro:bit flashes during the transfer and once completed, the code will run automatically.

NOTE!

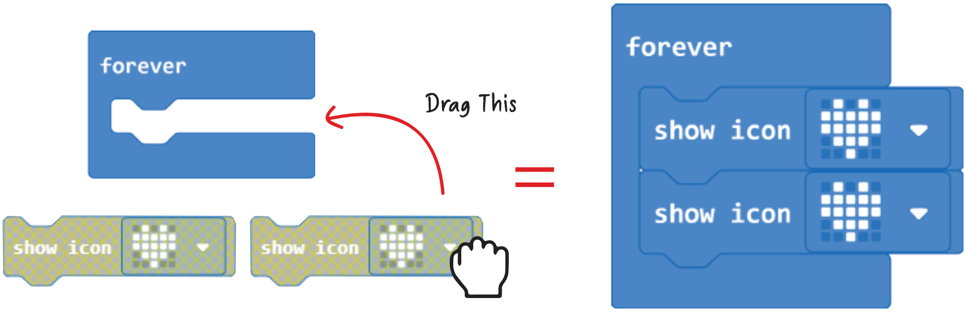
If the pop-up window does not appear, it means that the file has been automatically downloaded to the location where your browser is set to save downloads. Right-click on the downloaded .hex file which will appear at the bottom of the window and select 'Show in folder'. Click and drag the downloaded "microbit-xxxx.hex" file to the MICROBIT drive, as if you were copying a file to a flash drive.



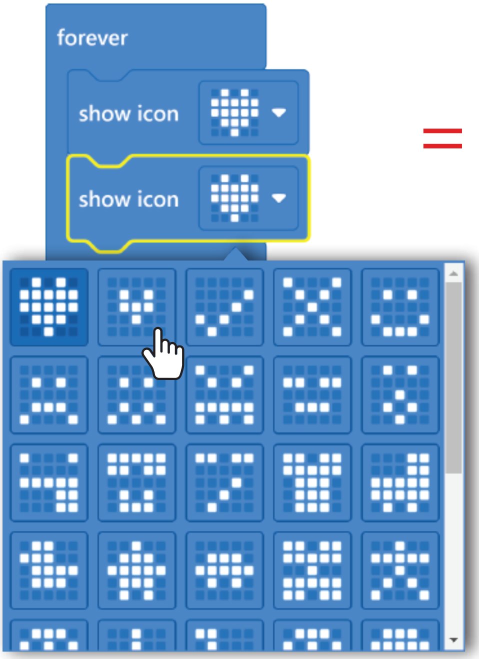


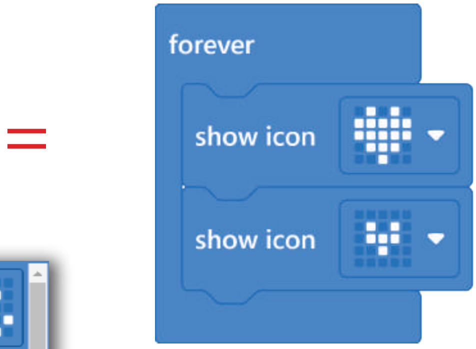
microbit-Hello-World 19-Oct-18 9:26 AM

Step 6 Click [Basic] and then click [show icon] block. Repeat to add another [show icon] block. Click and snap the [show icon] blocks to the [forever] slot.



Step 7 Left-click on the icon of the second [show icon] block and select the 'small heart' design from the pop-up window. Flash the code to your EDU:BIT.

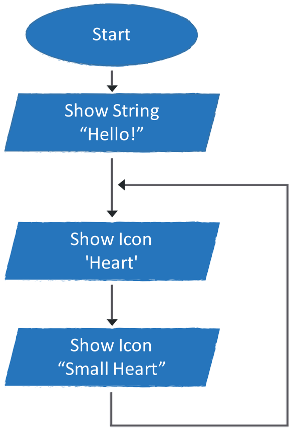




ANNA
It's love at first sight. Do you see a beating heart animation?

BREAK THE CODE

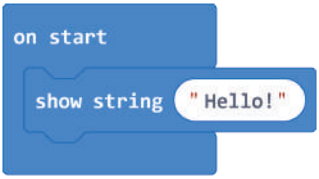
Did you notice that the "Hello!" text only scrolls across the display one time but the heart icons keep looping? Why?

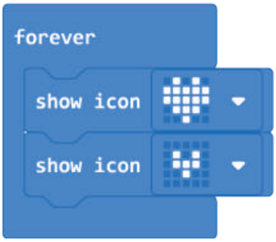

Start

Show String "Hello!"

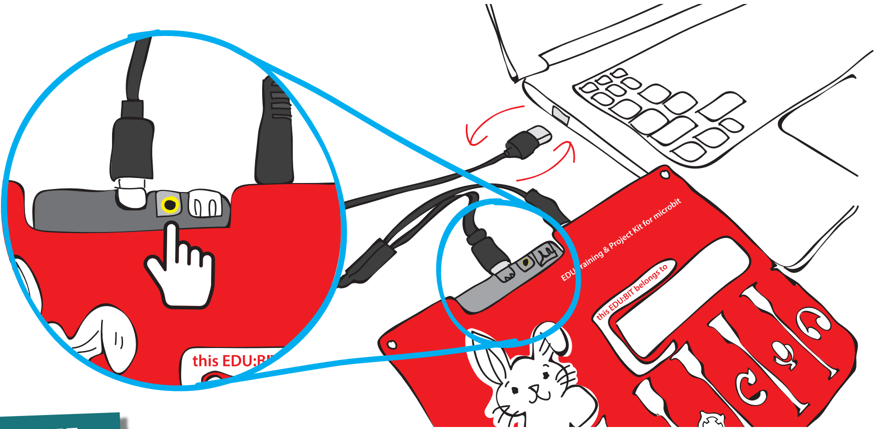
Show Icon 'Heart'

Show Icon "Small Heart"


```
on start
  show string "Hello!"
```


```
forever
  show icon ▾
  show icon ▾
```

The [on start] block runs the code on start (once). The [forever] block runs the code over and over again (forever).

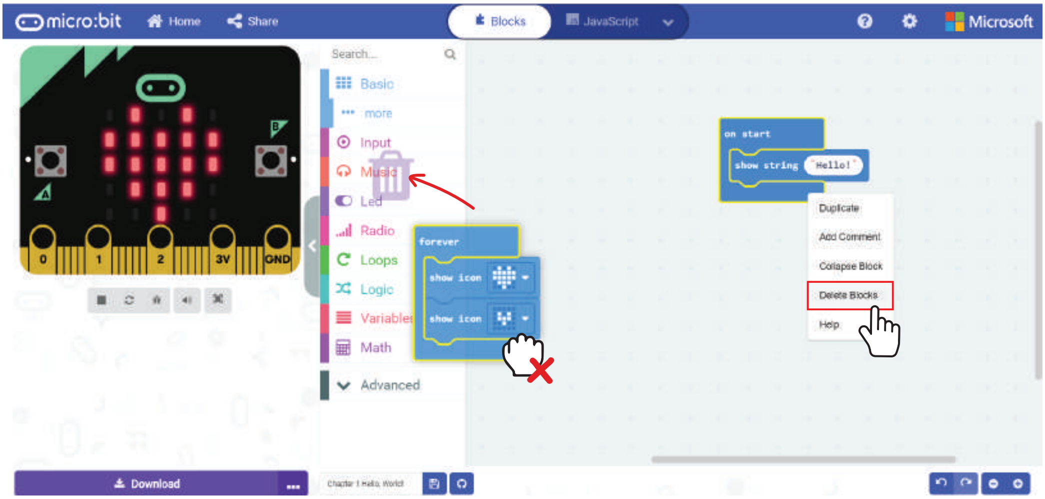


NOTE!

If you wish to start the program all over again, simply press the RESET button or unplug the USB cable, and then plug it back in again to reset the board.

Quick Tip #1!

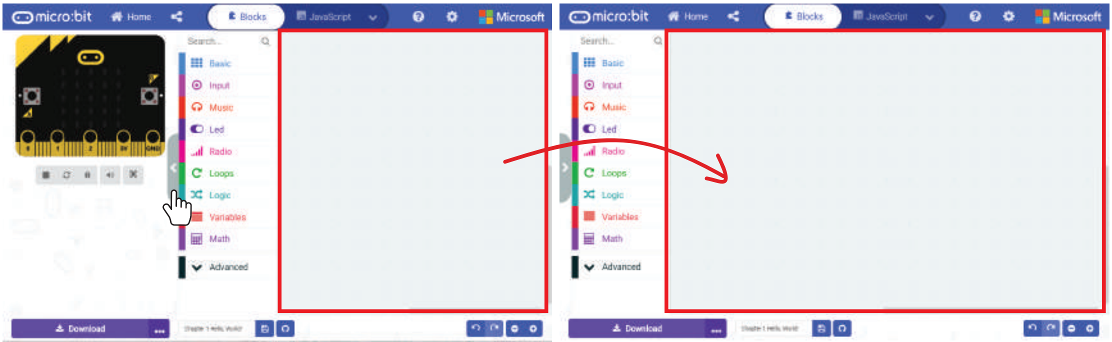
You can delete block(s) by clicking and dragging the unwanted block(s) to the Toolbox area. Release to delete the block(s) when the "bin" icon appears. Alternatively, you can right click the block and select “Delete Block”.

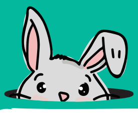


```
on start
    show string "Hello!"

forever
    show icon
    show icon
```

Quick Tip #2!

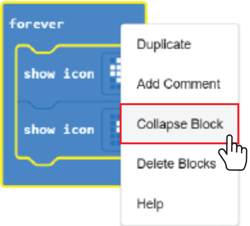
If you are not using the simulator window, you can click the tab to hide it so that you can have more space for your coding blocks.

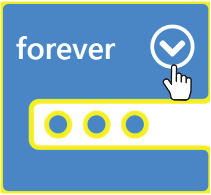




Quick Tip #3!

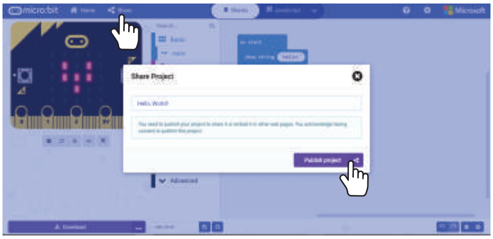
You can collapse a cluster of blocks by right-clicking on the block cluster and then select 'Collapse Block'. To expand a collapsed cluster of blocks, simply click on the ⌄ icon.

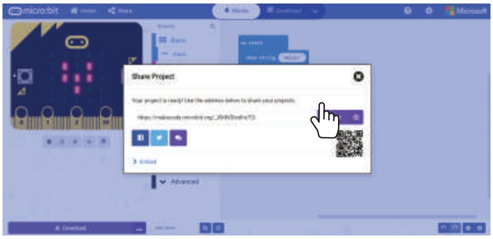




Quick Tip #4!

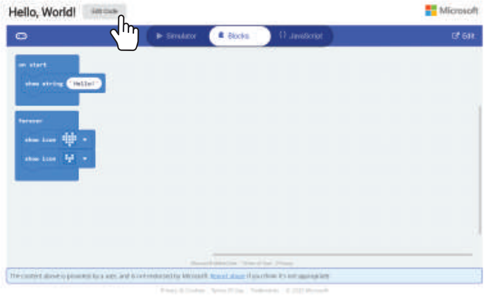
You can share your code with your teacher or friends by publishing your project and then sending them your project URL. To do that, click [**Share**], and then click [**Publish project**] in the pop-up window. You will then see a new pop-up window with your project URL. Click the [Copy] button to copy the project URL.





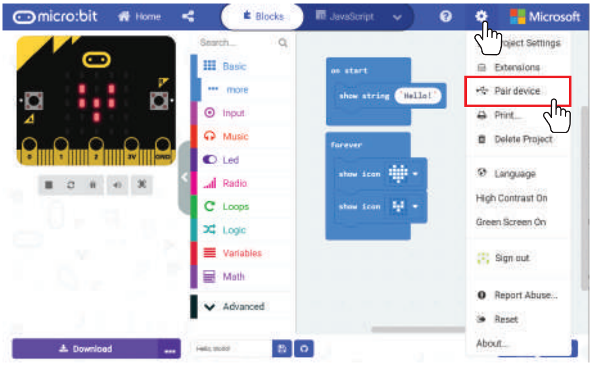
Quick Tip #5!

Your teacher or friends will see the following page when they open your project URL in a browser. They can view your code and also edit it by clicking the [**Edit Code**] button.



Quick Tip #6!

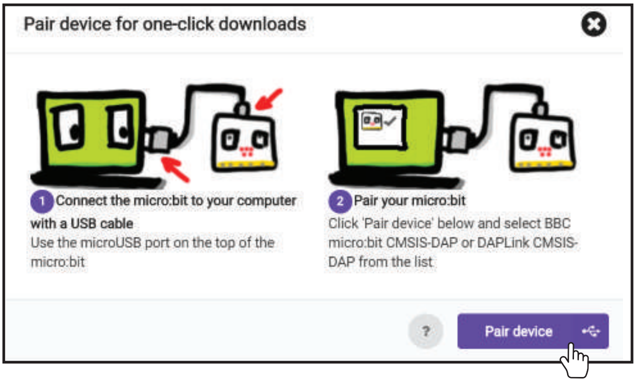
Do you know that you can pair device for one-click downloads? To do that, click the cogwheel icon and then select 'Pair device'.



NOTE!

You need to have the latest firmware on your micro:bit device and use either the new Edge or Chrome browser. Follow the instructions here if you need to update the firmware - https://microbit.org/get-started/user-guide/firmware/

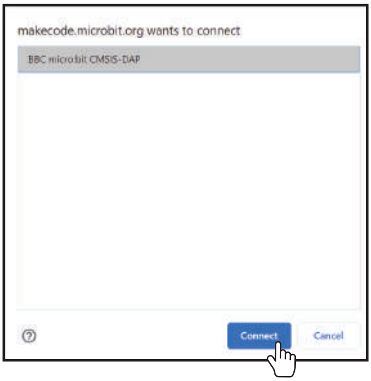
Make sure that your EDU:BIT is connected to your PC and then click [**Pair device**] button in the pop-up window. Next, select BBC micro:bit CMSIS-DAP or DAPLink CMSIS-DAP from the list and then click [**Connect**].



Pair device for one-click downloads

1 Connect the micro:bit to your computer with a USB cable
Use the microUSB port on the top of the micro:bit

2 Pair your micro:bit
Click 'Pair device' below and select BBC micro:bit CMSIS-DAP or DAPLink CMSIS-DAP from the list



makecode.microbit.org wants to connect

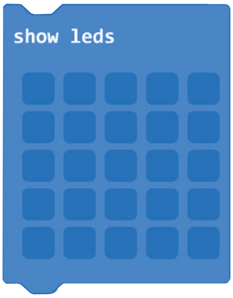
BBC micro:bit CMSIS-DAP

After you've paired your device, you can directly flash your code to your EDU:BIT when you click the [**Download**] button. Give it a try!

If you have problems pairing your device, you can refer to https://makecode.microbit.org/device/usb/webusb/troubleshoot for more info.

EXPLORE MORE BLOCKS

#1 Use the [show leds] block to design your own icons and [show number] block to display numbers.



```
show leds
```

```
show number 0
```

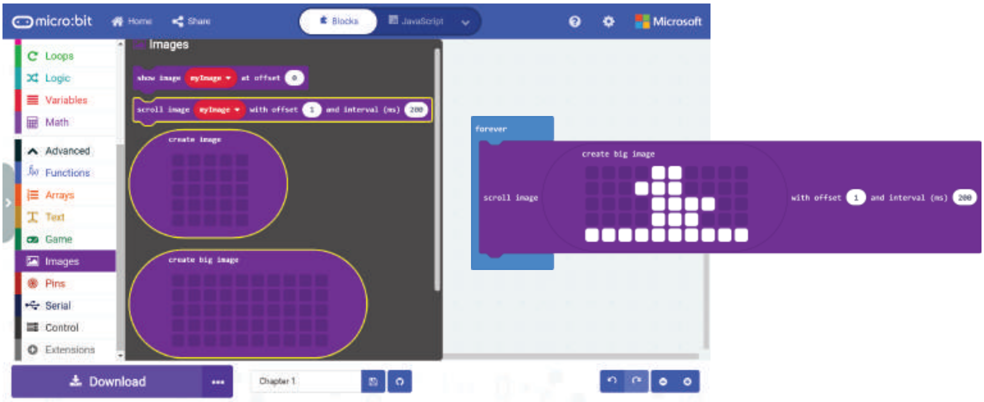
Do you know?

1000 milliseconds (ms) = 1 second

#2 Add a [pause] block to slow down the program. This function pauses the program for the number of milliseconds (ms) that you set.

```
pause (ms) 100
```

#3 To scroll an image across the LED matrix display, you can use [scroll image _ with offset _ and interval (ms) _] block, together with either [create image] block or [create big image] block, from the [Images] category drawer (under **Advanced** category).

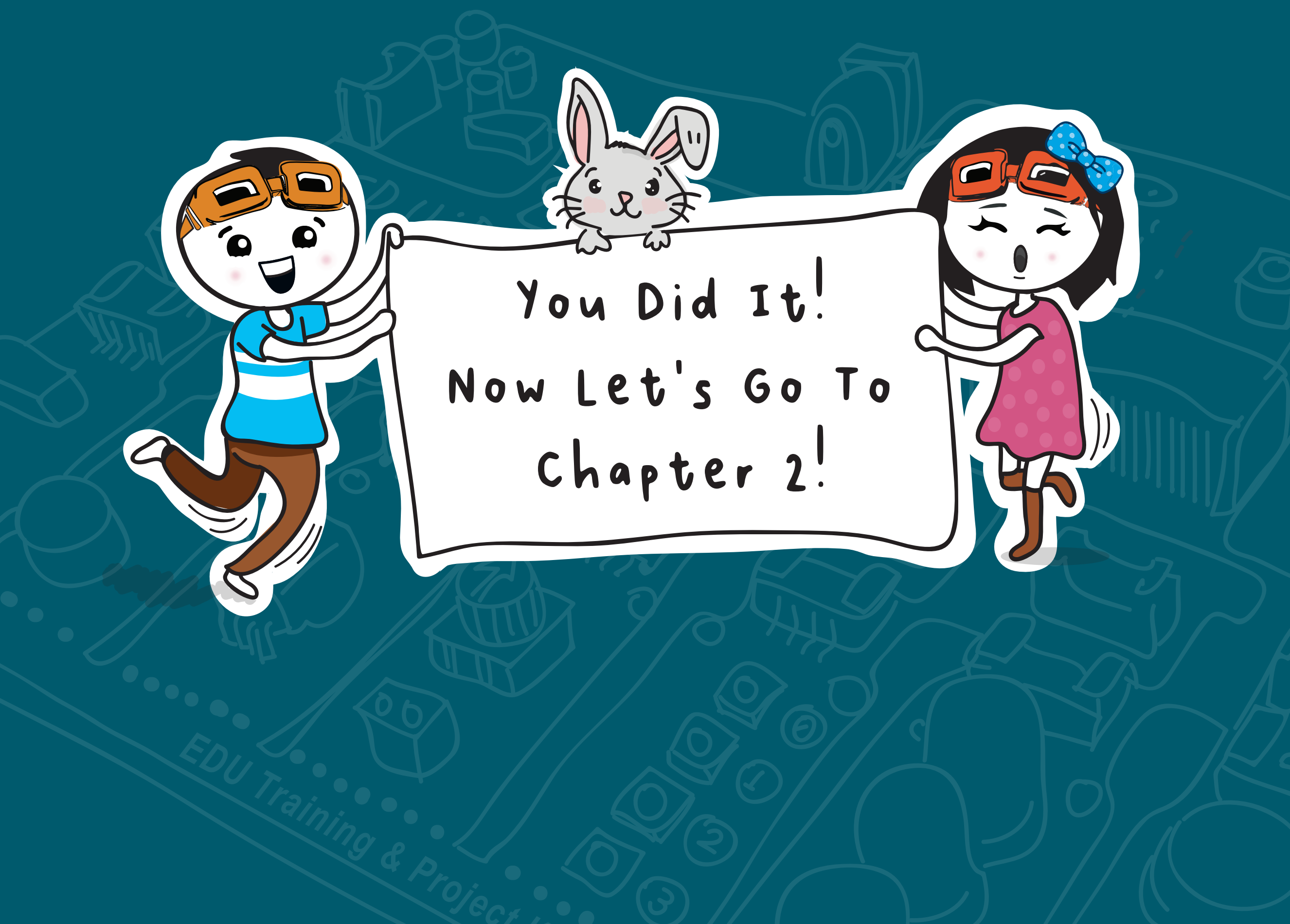


```
show image myImage at offset 0
scroll image myImage with offset 1 and interval (ms) 200
create image
create big image

forever
    scroll image create big image with offset 1 and interval (ms) 200
```

When you run the program, you will see little ducklings moving across the LED matrix display, one after another.

APPLICATION CHALLENGE

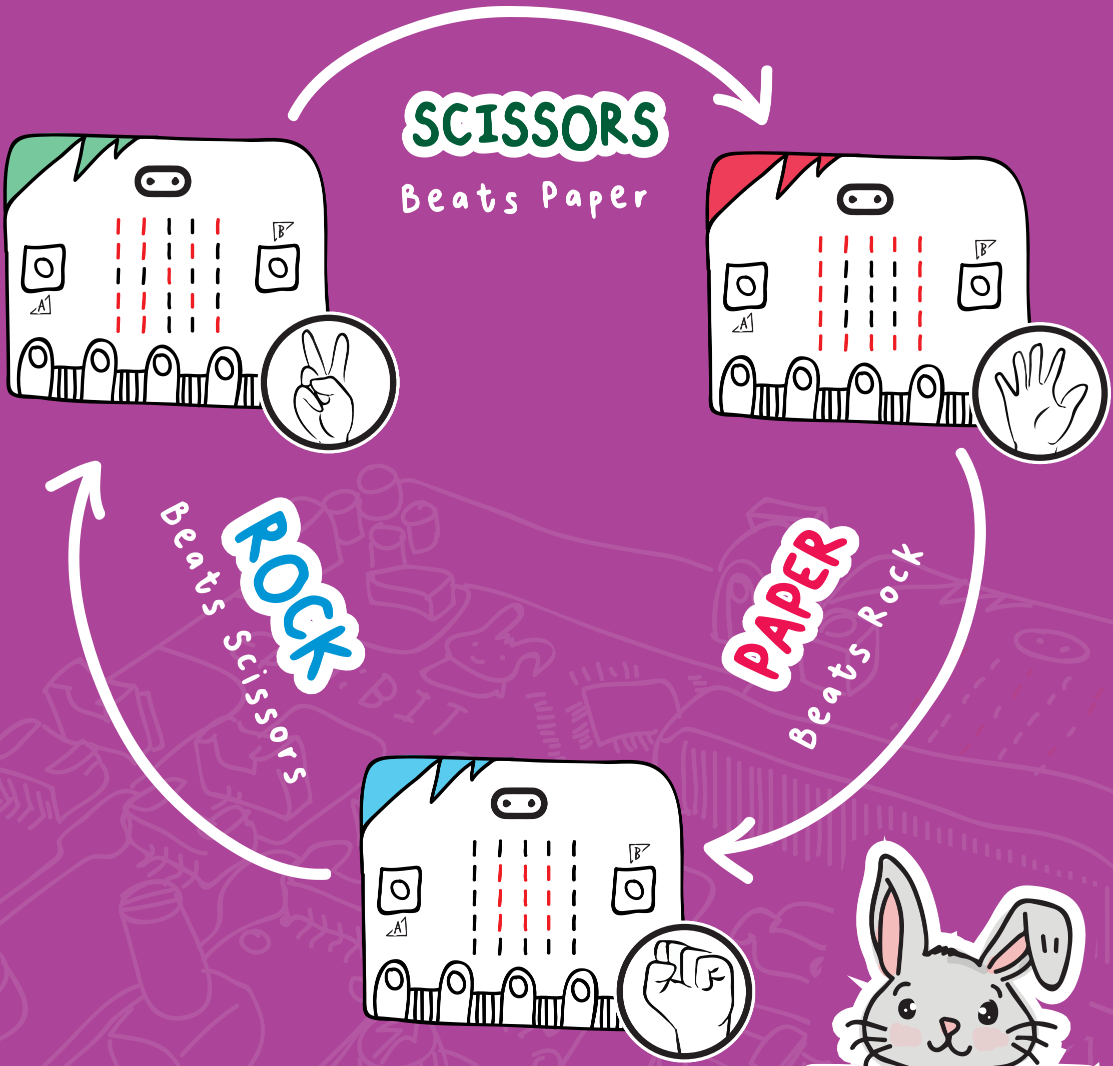
Program EDU:BIT to function as a digital announcement board.	
On start	Display a simple animation to attract attention and then scroll your class name.
Forever	Scroll today’s date and other important information for the class.



CHAPTER 2

Let's Play Rock Paper Scissors!

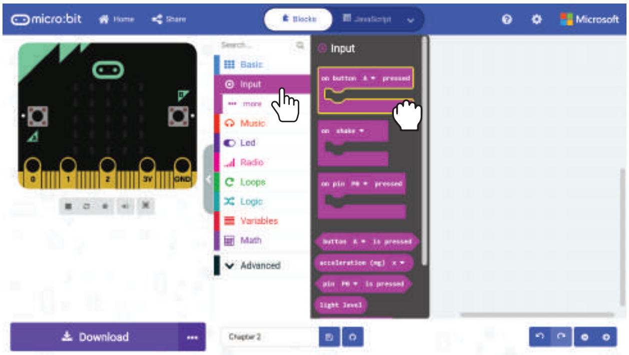
Push Buttons on micro:bit and Button Bit

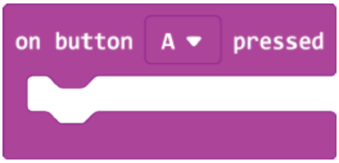


link.cytron.io/edubit-chapter-2

LET'S CODE!

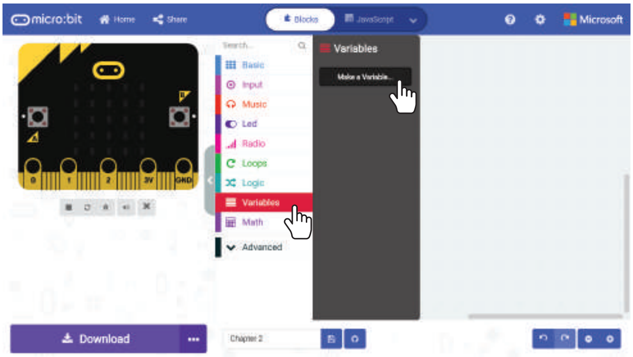
Step 1 Go to https://makecode.microbit.org/ (or simply click the **Home** icon if you're already in MakeCode Editor) and create a new project. Click **[Input]** category and then select **[on button _ pressed]** block.

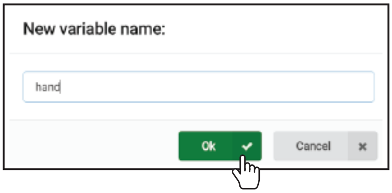




```
on button A ▾ pressed
```

Step 2 Click **[Variables]** category and then select **[Make a Variable]**. Type **'hand'** in the pop up window and then click OK.

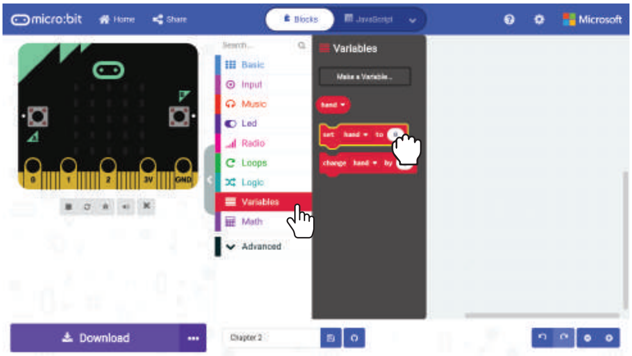


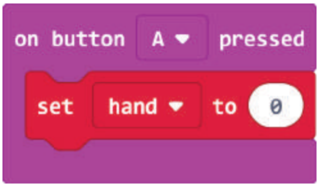


New variable name:

hand

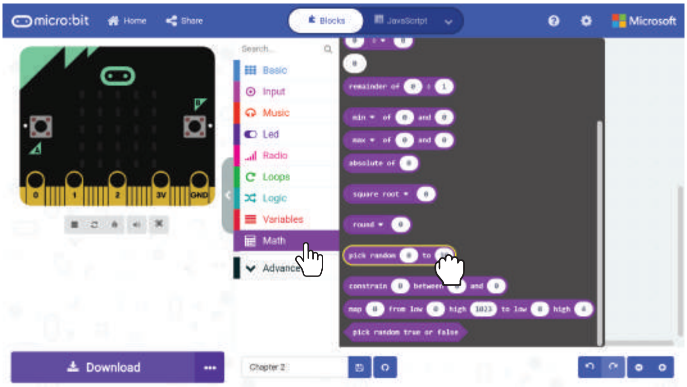
Step 3 Click **[Variables]** category and then select **[set _ to _]** block. Snap the block to the **[on button A pressed]** block.

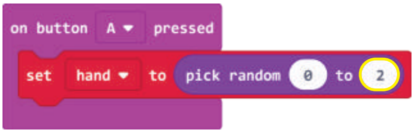




```
on button A ▾ pressed
    set hand ▾ to 0
```

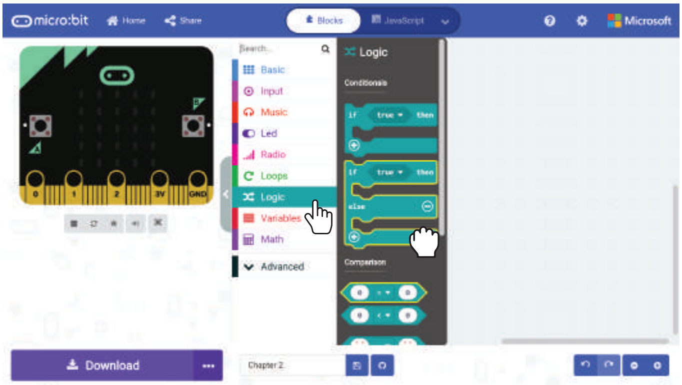
Step 4 Click [Math] category and select [pick random _ to _] block. Change the number 10 to **2.**

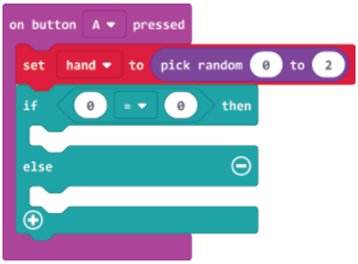




```
on button A ▾ pressed
    set hand ▾ to pick random 0 to 2
```

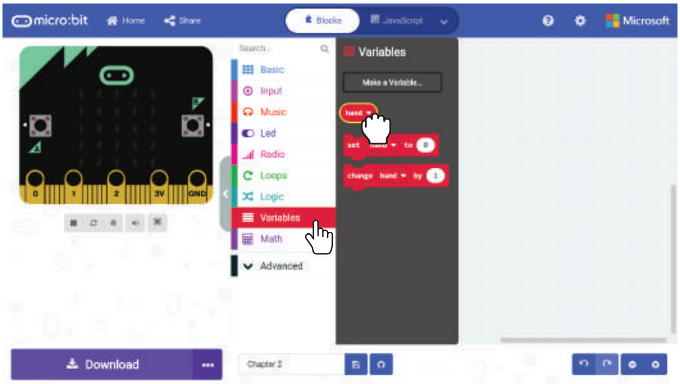
Step 5 Click [Logic] category and select [if-then-else] block and [_ = _] comparison block. Place the comparison block in the 'if' slot.

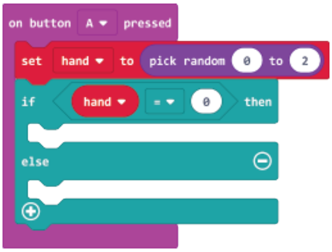




```
on button A ▾ pressed
    set hand ▾ to pick random 0 to 2
    if 0 = ▾ 0 then

    else

```

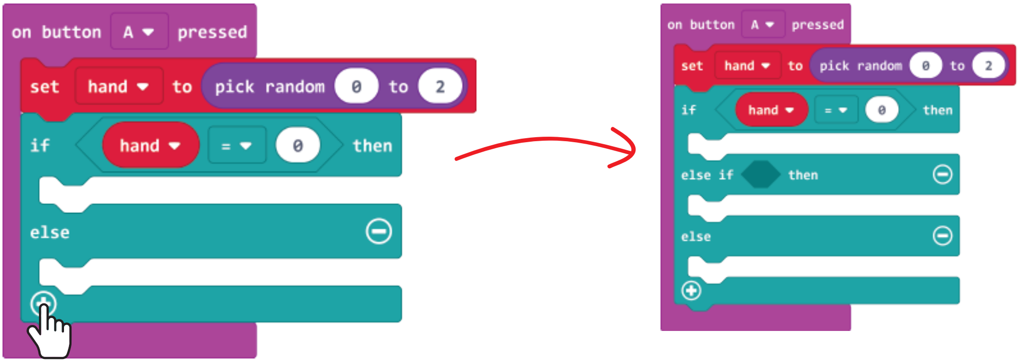
Step 6 Click [Variables] and then select [hand] block. Snap the block to the comparison block.





```
on button A ▾ pressed
    set hand ▾ to pick random 0 to 2
    if hand ▾ = ▾ 0 then

    else

```

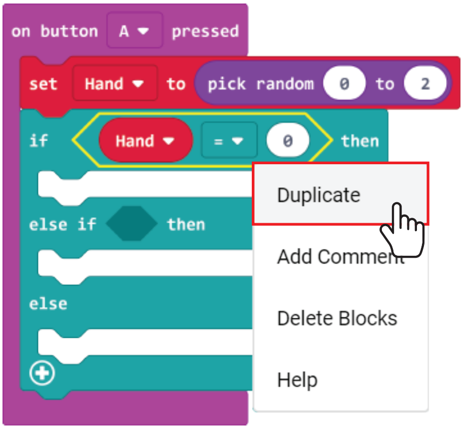
Step 7 Click on the ⊕ icon to add **[else if]** condition to the 'if' block.



```
on button A ▾ pressed
    set hand ▾ to pick random 0 to 2
    if hand ▾ = ▾ 0 then

    else ⊖

    ⊕
```

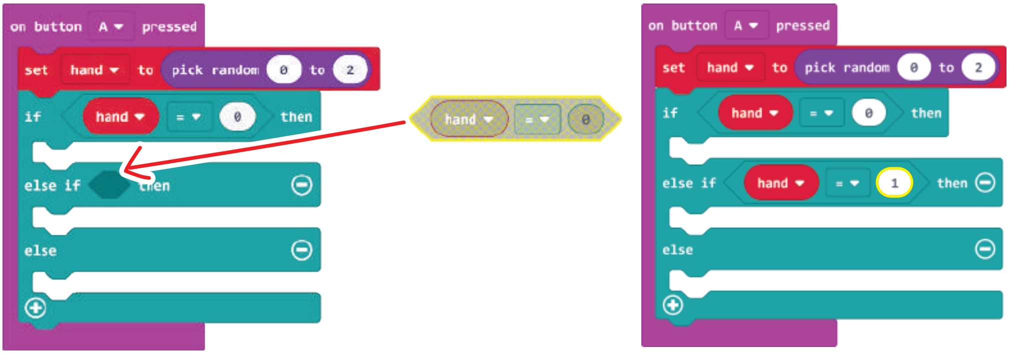
```
on button A ▾ pressed
    set hand ▾ to pick random 0 to 2
    if hand ▾ = ▾ 0 then

    else if ⬡ then ⊖

    else ⊖

    ⊕
```

Step 8 Right click on the comparison block and then select **"Duplicate"**.



```
on button A ▾ pressed
    set Hand ▾ to pick random 0 to 2
    if Hand ▾ = ▾ 0 then

    else if ⬡ then

    else

    ⊕
```

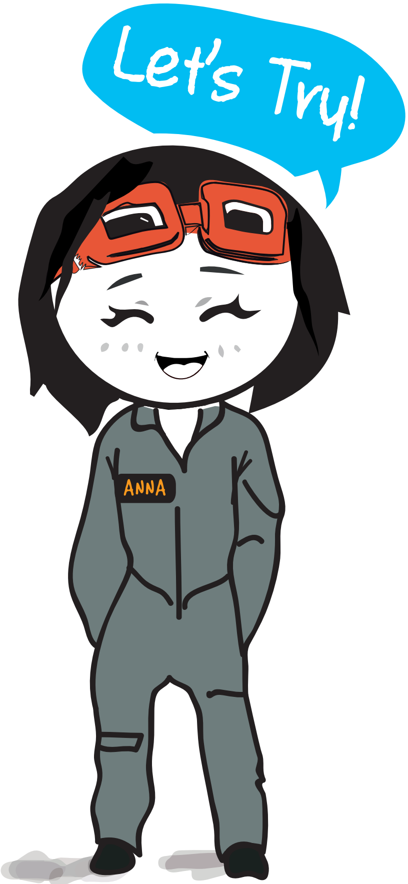
Step 9 Snap the duplicated block to the 'else if' slot and change the number 0 to **1.**

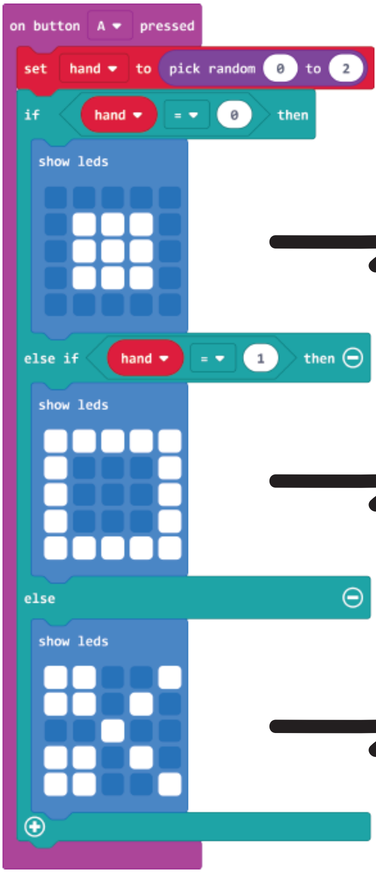


```
on button A ▾ pressed
    set hand ▾ to pick random 0 to 2
    if hand ▾ = ▾ 0 then

    else if ⬡ then ⊖

    else ⊖

    ⊕

hand ▾ = ▾ 0
```

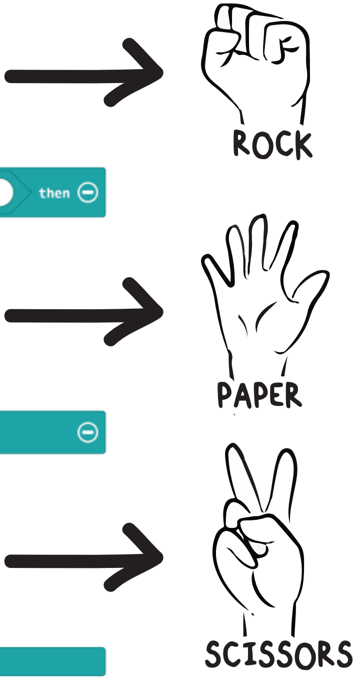
```
on button A ▾ pressed
    set hand ▾ to pick random 0 to 2
    if hand ▾ = ▾ 0 then

    else if hand ▾ = ▾ 1 then ⊖

    else ⊖

    ⊕
```

Step 10 Add **[Basic] : [show leds]** blocks to 'if' , 'else if' and 'else' slots. Click the boxes in the **[show leds]** blocks to create images according to the example shown below.





```
on button A ▾ pressed
  set hand ▾ to pick random 0 to 2
  if hand ▾ = ▾ 0 then
    show leds
      □ □ □ □ □
      □ ■ ■ ■ □
      □ ■ ■ ■ □
      □ ■ ■ ■ □
      □ □ □ □ □
  else if hand ▾ = ▾ 1 then ⊖
    show leds
      ■ ■ ■ ■ ■
      ■ □ □ □ ■
      ■ □ □ □ ■
      ■ □ □ □ ■
      ■ ■ ■ ■ ■
  else ⊖
    show leds
      ■ ■ □ □ ■
      ■ ■ □ ■ □
      □ □ ■ □ □
      ■ ■ □ ■ □
      ■ ■ □ □ ■
  ⊕
```

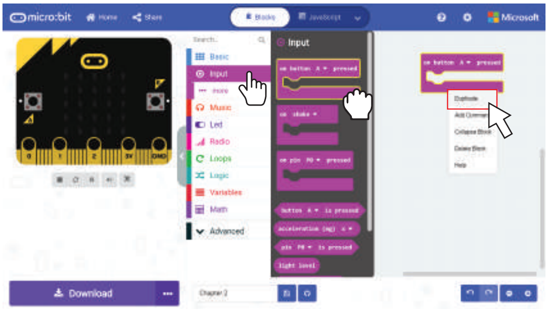


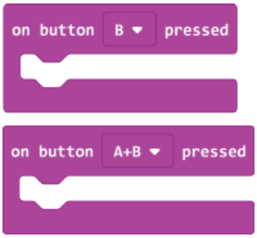
Flash the code to your EDU:BIT and now you can play Rock, Paper, Scissors with your friends. Whenever you press button A on micro:bit or the yellow button, the LED matrix will randomly display "rock" , "paper" or "scissors".

Reminder: If you want to keep the project, remember to save it to a folder on your computer by clicking the save button.

Let's bring this game up a notch by assigning 3 lives to each player. To do that you need to create a new variable named 'lives' and add the following coding blocks.

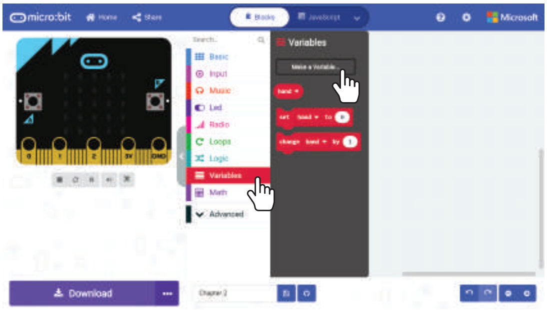
Step 11 Click [Input] category and select [on button _ pressed] block. Duplicate the block and change the setting to **'button B'** and **'button A+B'** respectively.

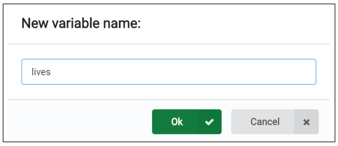




```
on button B ▾ pressed

on button A+B ▾ pressed
```

Step 12 Click [Variables] category and select [**Make a Variable**]. Type **'lives'** in the pop up window and then click OK.



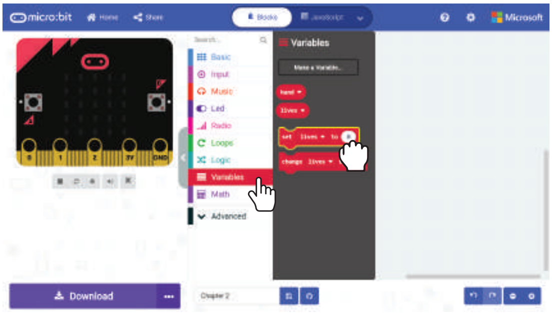


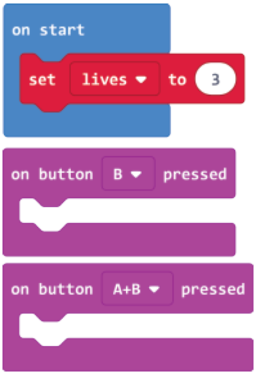
New variable name:

lives

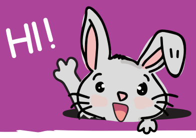
Ok ✓ Cancel ×

Step 13 Click [Variables] category and then select [set _ to _] block. Snap the block to [Basic] : [on start] block. Set the variable to **'lives'** and change the value to **3**.

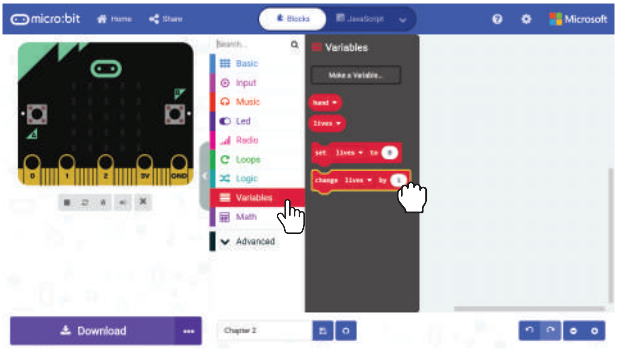


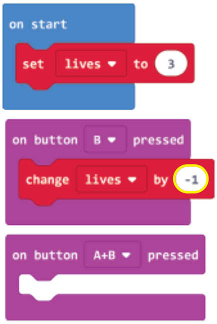


```
on start
    set lives ▾ to 3

on button B ▾ pressed

on button A+B ▾ pressed
```



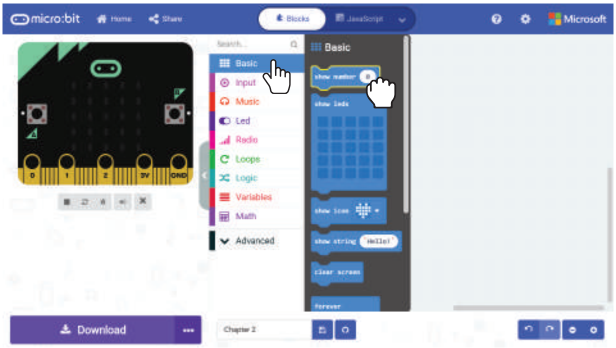
Step 14 Click **[Variables]** category again and then select **[change _ by _]** block. Snap the block to **[on button B pressed]** block. Set the variable to '**lives**' and change the value to **-1**.

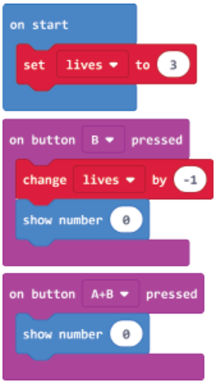




```
on start
  set lives to 3

on button B pressed
  change lives by -1

on button A+B pressed
```

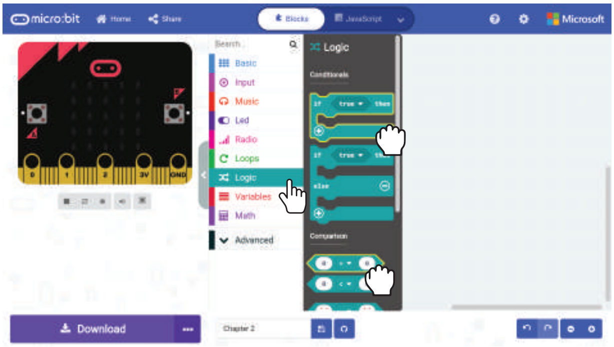
Step 15 Click **[Basic]** category and select **[show number]** block. Duplicate it and snap the blocks to both the **[on button A+B pressed]** blocks.

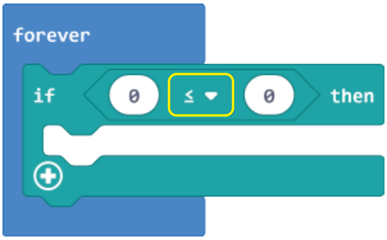




```
on start
  set lives to 3

on button B pressed
  change lives by -1
  show number 0

on button A+B pressed
  show number 0
```

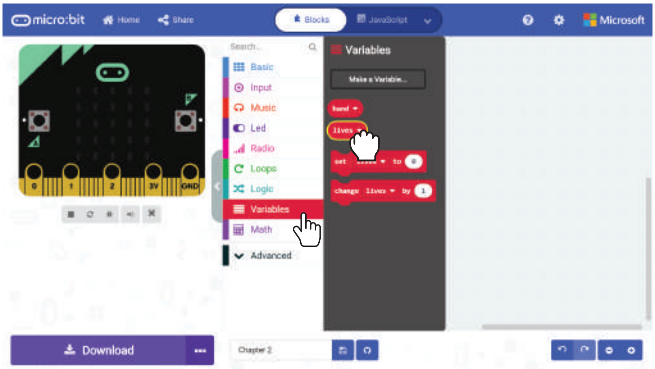
Step 16 Click **[Logic]** category and add **[if-then]** block and **[_ = _]** comparison block to your code. Snap them to the **[Basic] : [forever]** block and change the sign to ' ≤ '.

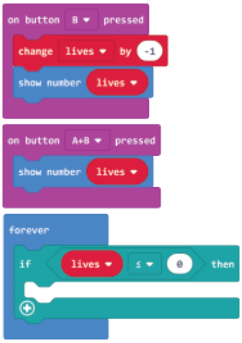




```
forever
  if 0 ≤ 0 then
```

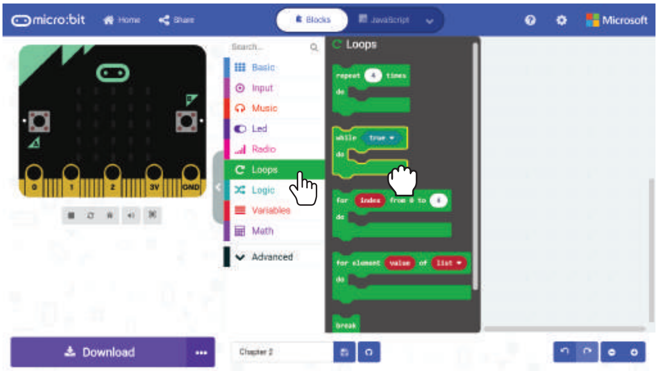
Step 17 Click **[Variables]** category and then select **[lives]** block. Duplicate and snap to both the **[show number]** blocks and also the left slot of the **[_ = _]** comparison block.

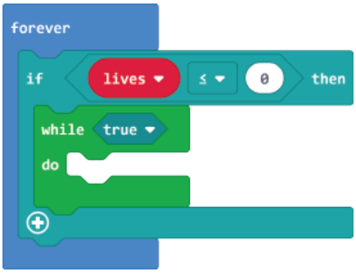




```
on button B pressed
    change lives by -1
    show number lives

on button A+B pressed
    show number lives

forever
    if lives ≤ 0 then
```

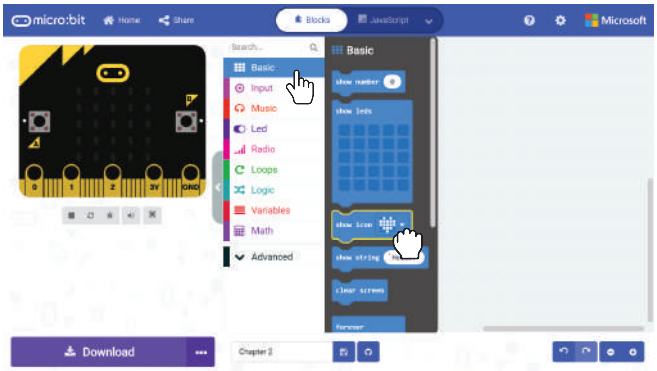
Step 18 Click **[Loops]** category and select **[while _ do]** block. Snap it to the slot of the **[if-then]** block.

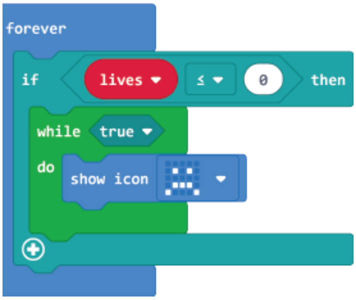




```
forever
    if lives ≤ 0 then
        while true
        do
```

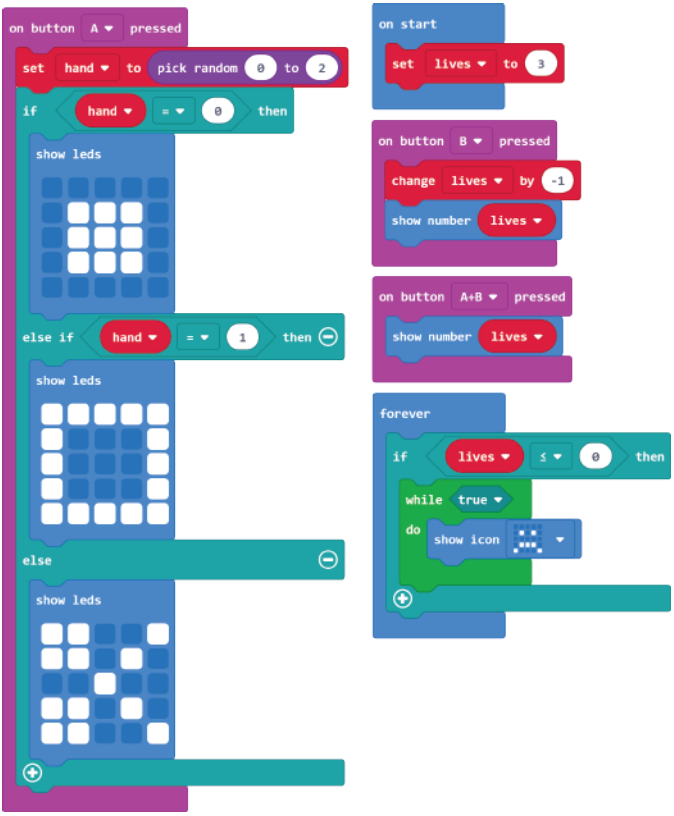
Step 19 Click **[Basic]** category and select **[show icon]** block. Snap it for the slot of the **[while _ do]** block and change the icon to a **'sad face'.**



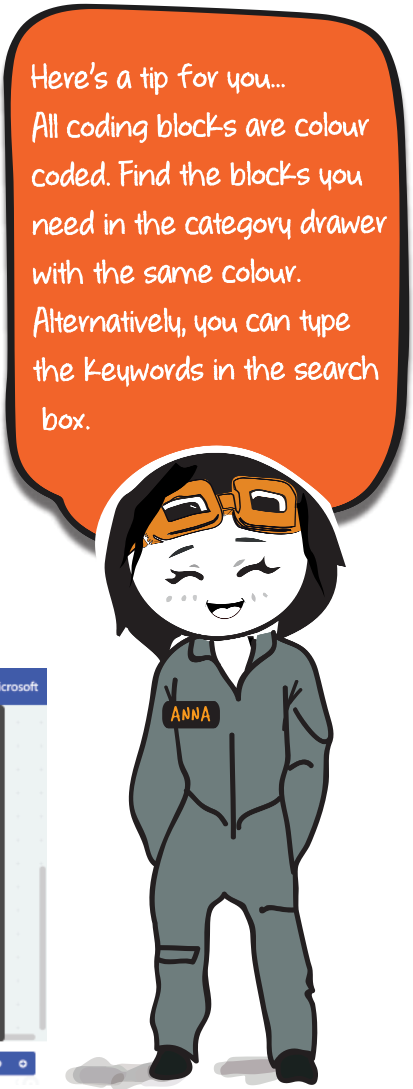


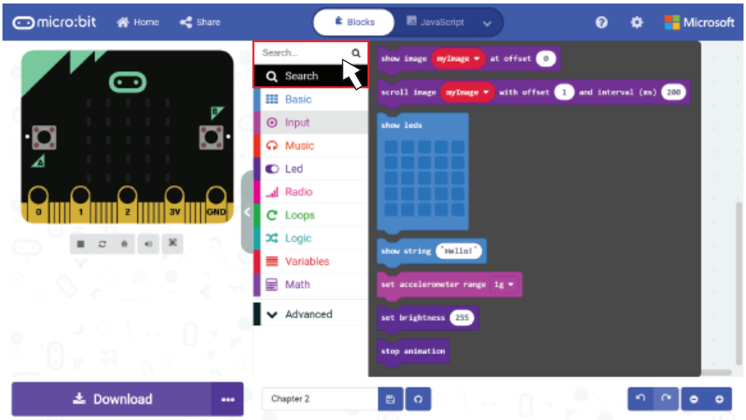
```
forever
    if lives ≤ 0 then
        while true
        do
            show icon
```

Step 20 Here's the complete code. Flash it to your EDU:BIT and have fun playing with your friends to determine the King or Queen of Rock, Paper and Scissors game.



```
on button A pressed
    set hand to pick random 0 to 2
    if hand = 0 then
        show leds
            . . . . .
            . # # # .
            . # # # .
            . # # # .
            . . . . .
    else if hand = 1 then
        show leds
            # # # # #
            # . . . #
            # . . . #
            # . . . #
            # # # # #
    else
        show leds
            # # . . #
            # # . # .
            . . # . .
            # # . # .
            # # . . #

on start
    set lives to 3

on button B pressed
    change lives by -1
    show number lives

on button A+B pressed
    show number lives

forever
    if lives ≤ 0 then
        while true
        do
            show icon
```

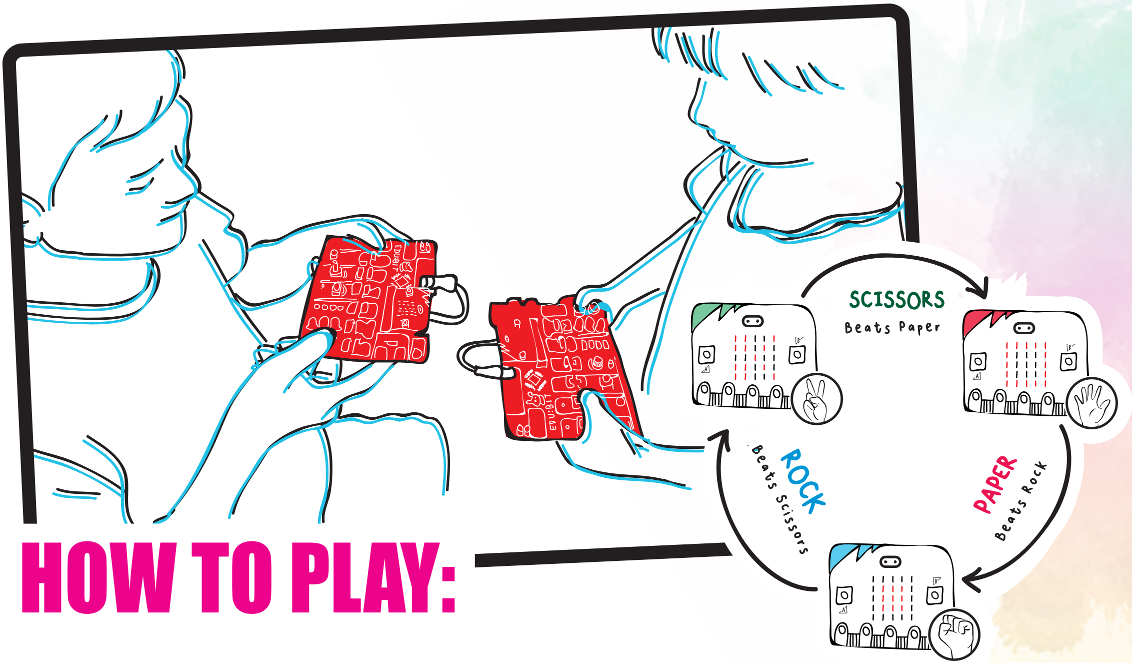

Here's a tip for you...
All coding blocks are colour coded. Find the blocks you need in the category drawer with the same colour.
Alternatively, you can type the keywords in the search box.
ANNA



```
show image myImage at offset 0
scroll image myImage with offset 1 and interval (ms) 200
show leds
show string "Hello!"
set accelerometer range 1g
set brightness 255
stop animation
```

Let's Play

Rock, Paper, Scissors - Advanced Version



HOW TO PLAY:

- Stand facing your opponent. When both players are ready, press the yellow button (Button A) to randomly display rock, paper or scissors.
- Compare and decide who wins.
- If you lose, you need to press the blue button (Button B) on your EDU:BIT once to minus one life.
- Press both yellow and blue buttons at the same time to check and display the number of remaining lives.
- If a player loses 3 times, it is Game Over and his/her EDU:BIT will display a sad face.

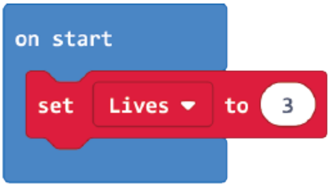
NOTE!

- To play another round, you need to RESET your board to start over again.
- If you do not have a friend to play with, you can always play against the simulator on your MakeCode Editor.

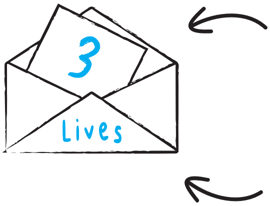
BREAK THE CODE

In computer programming, we use **variables** to store information or value that can be changed at runtime (when your program is running). You can think of a variable as a labelled envelope that contains a piece of paper with information written on it. The piece of paper can be taken out and replaced with another piece of paper with new information.

In our code earlier we created a variable called "lives" and assigned an initial value of 3 to it.



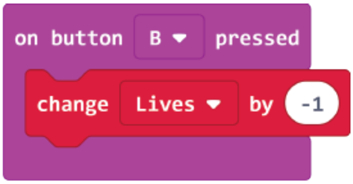
```
on start
    set Lives ▾ to 3
```



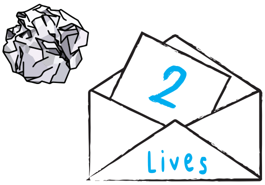
Number 3 written on a piece of paper is the information.

Envelope with 'Lives" written on it is known as the variable

Then, whenever button B is pressed, the value is changed by -1.

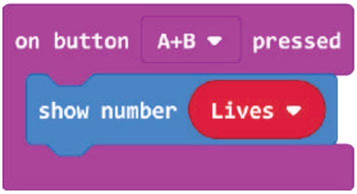


```
on button B ▾ pressed
    change Lives ▾ by -1
```

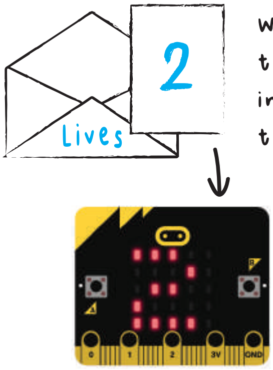


When Button B is pressed, the piece of paper with "3" written on it is removed from the envelope and replaced with another paper with "2" (i.e. 3 - 1 = 2) written on it.

Pressing buttons A+B simultaneously will result in the LED matrix displaying the current value for the variable lives.



```
on button A+B ▾ pressed
    show number Lives ▾
```

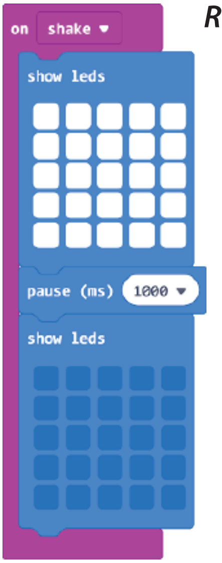


When Buttons A+B are pressed, take out the piece of paper inside the envelope and "read" the information on it.

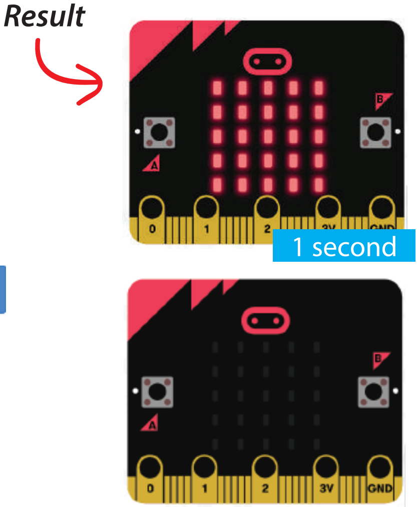
EXPLORE MORE BLOCKS

Besides [on button __ pressed] block, you can also use other blocks from [Input] category drawer for event-based programming. Actions performed by a user such as pressing a button or shaking the board is known as an 'event'.

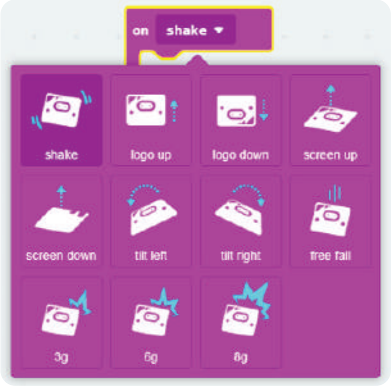
The following code, for example, will result in the LED matrix lighting up for 1 second whenever the board is shaken. Give it a try~



```
on shake ▾
  show leds
  pause (ms) 1000 ▾
  show leds
```

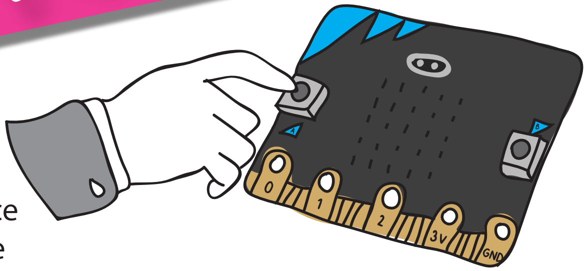


If you click on the [shake] button on the block, a pop up menu will appear showing a selection of other triggers. Try to program EDU:BIT to show a different icon for each of these conditions. Have fun exploring!

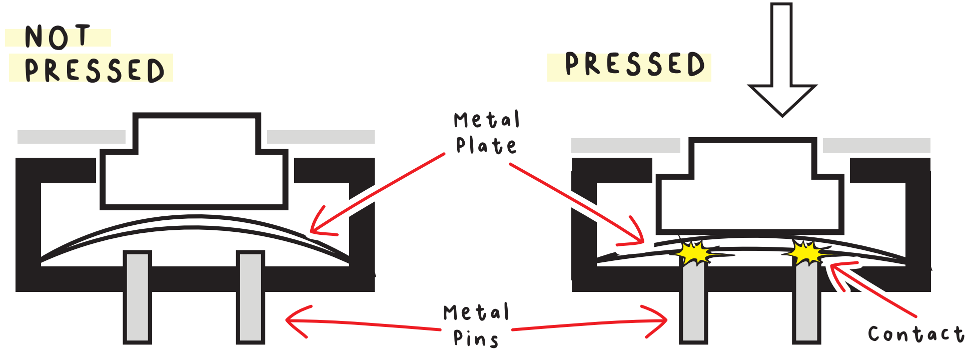


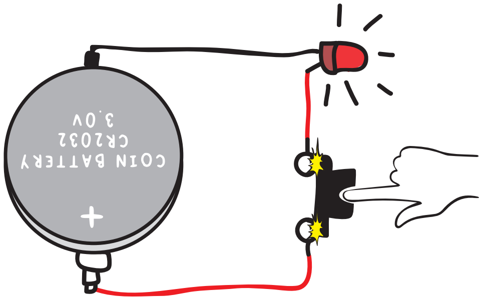
EDU:BIT can detect shaking events and know its own orientation because there's an inbuilt motion sensor on micro:bit.

FUN FACT!

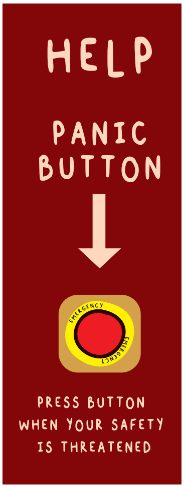


A **push button** is an input device or switch with only two possible states- pressed or not pressed.





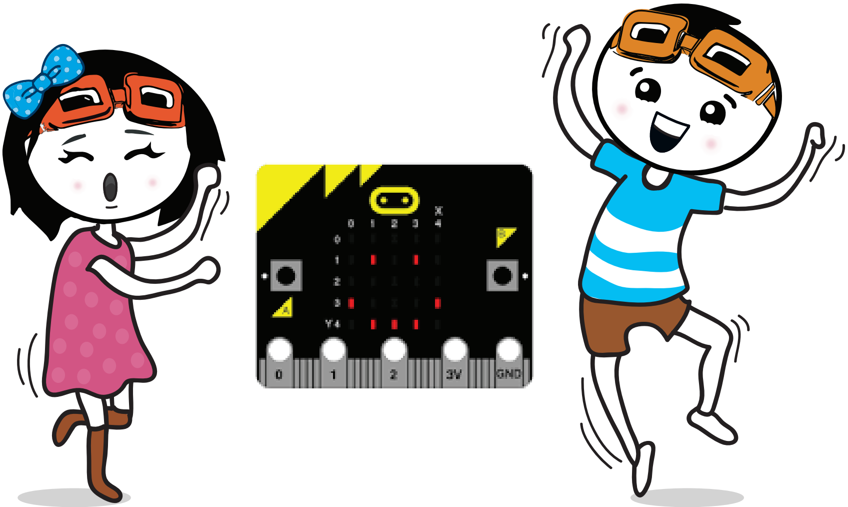
When the push button is pressed, it completes the electrical circuit and the LED is lighted up! Guess what happens when you release the push button?



Black, grey, green and white buttons are normally used for ON/OFF function and RED is used for panic button or emergency stop for machinery.

youtu.be/t_Qujjd_38o

APPLICATION CHALLENGE



Program EDU:BIT to function as a counter to record student attendance. Girls are required to press Button A when they enter the class; and boys are required to press Button B.

On start	Show a smiley face. Set variables Girl = 0 and Boy = 0
On Button A pressed (Yellow Button)	Change variable Girl by 1
On Button B pressed (Blue Button)	Change variable Boy by 1

On Buttons A+B pressed	Scroll the following info across the LED display:
	Total = (Girl + Boy) ; Girl = (Girl) ;
	Boy = (Boy)

CHAPTER 3

Let's Have Some Music~

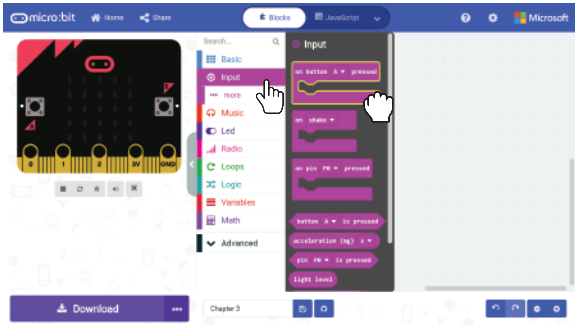
Music Bit (Piezo Buzzer + Audio Jack)

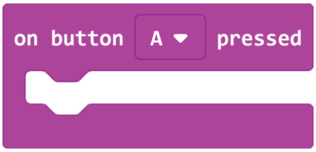
Scan Me!

link.cytron.io/edubit-chapter-3

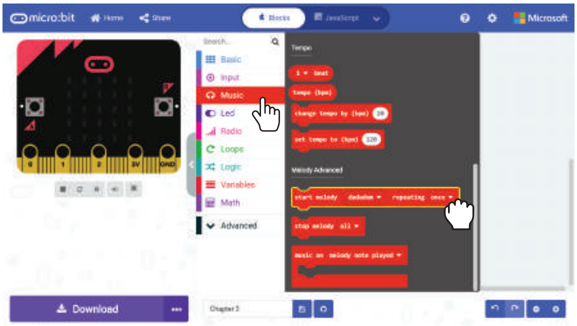
LET'S CODE!

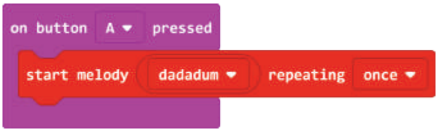
Step 1 Create a new project in your MakeCode Editor. Click **[Input]** category and then select **[on button _ pressed]** block.



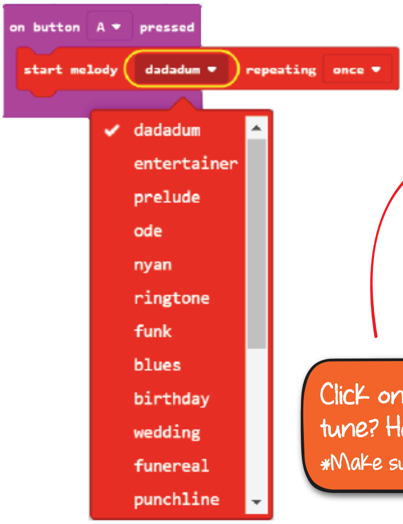


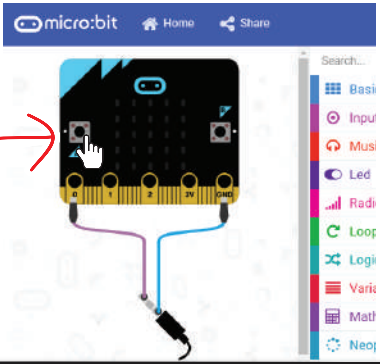
Step 2 Click **[Music]** category and then select **[start melody _ repeating _]** block.





Step 3 Click on **[dadadum]** and select **'birthday'** melody from the drop down list.



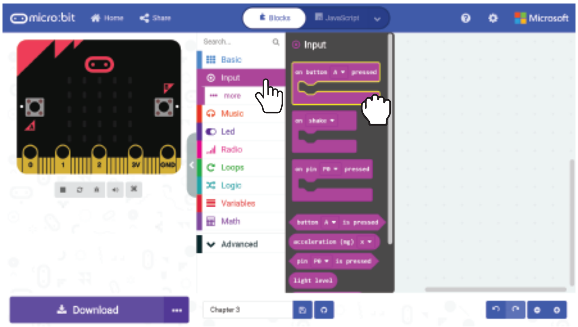


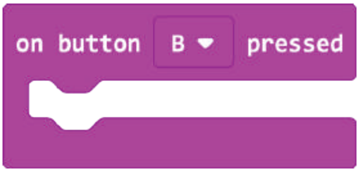
Click on Button A of your on-screen simulator. Do you hear a familiar tune? Have fun checking out the other melodies too~

*Make sure your computer speakers are turned on.

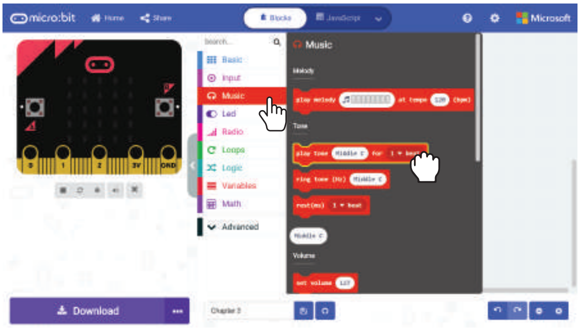
Besides the list of preset melodies, you can also program EDU:BIT to play any song you like. Let's try to play this catchy tune~

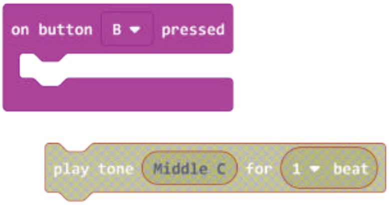
Step 4 Click [**Input**] category and then select [**on button _ pressed**] block. Select button **"B"**.



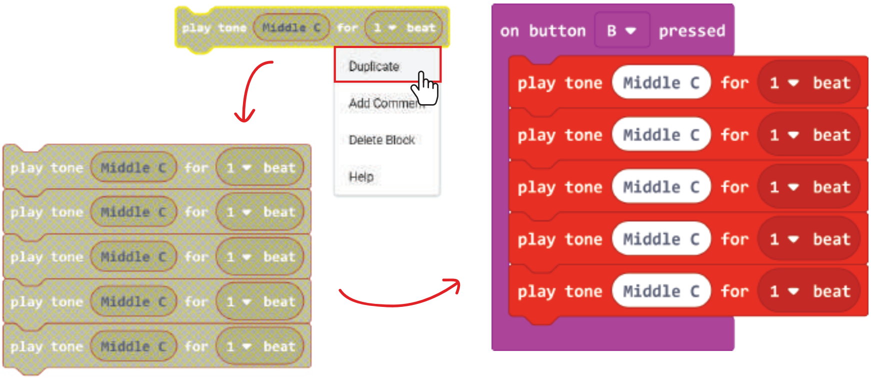


Step 5 Click [**Music**] category and then click [**play tone _ for _ beat**] block.

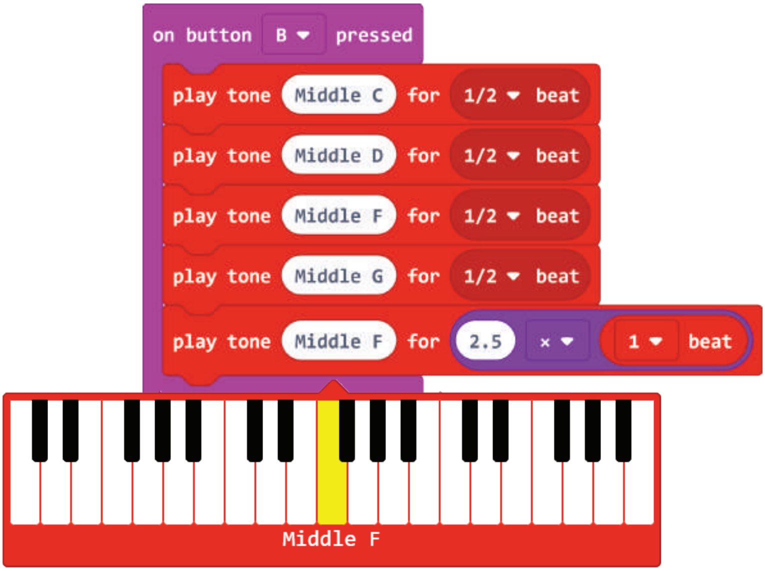




Step 6 In the Workspace, right-click on the [**play tone _ for _ beat**] block and then click 'Duplicate'. Repeat until you have a total of five [**play tone _ for _ beat**] blocks. Place the blocks in the slot of [**on button B pressed**] block.


play tone Middle C for 1 beat
Duplicate
Add Comment
Delete Block
Help
play tone Middle C for 1 beat
play tone Middle C for 1 beat
play tone Middle C for 1 beat
play tone Middle C for 1 beat
play tone Middle C for 1 beat
on button B pressed
play tone Middle C for 1 beat
play tone Middle C for 1 beat
play tone Middle C for 1 beat
play tone Middle C for 1 beat
play tone Middle C for 1 beat

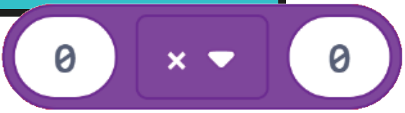
Step 7 Select the **'tone'** and **'beat'** of these **[play tone _ for _ beat]** blocks according to the sample code below:



```
on button B ▾ pressed
    play tone Middle C for 1/2 ▾ beat
    play tone Middle D for 1/2 ▾ beat
    play tone Middle F for 1/2 ▾ beat
    play tone Middle G for 1/2 ▾ beat
    play tone Middle F for 2.5 × ▾ 1 ▾ beat
```

NOTE!

The purple block is taken from [Math] category drawer.

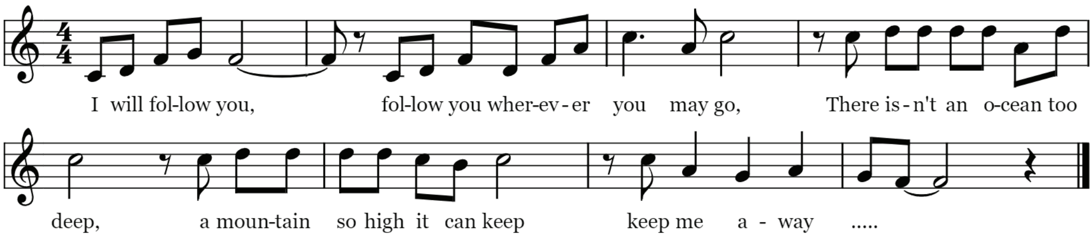


Click on Button B of your on-screen simulator.
Can you guess the song?

When you are programming, it is advisable to periodically check your code to make sure that you're on the right track. You can use the simulator for that purpose.

Step 8 Continue coding the rest of the song by adding more **[play tone _ for _ beat]** blocks and changing the **'tone'** and **'beat'** accordingly. You can refer to the next page for the tones and beats.

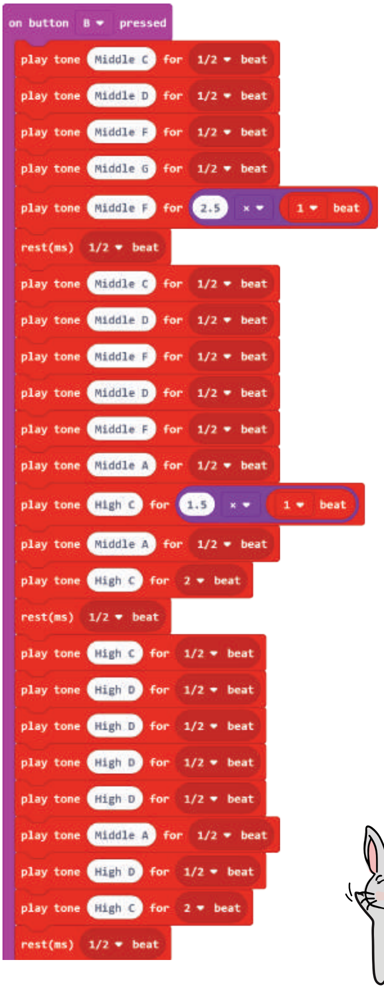
I Will Follow You


I will fol-low you, fol-low you wher-ev-er you may go, There is-n't an o-cean too
deep, a moun-tain so high it can keep keep me a - way .....

I
will
fol-
low
you,

Fol-
low
you
wher-
ev-
er
you
may
go,

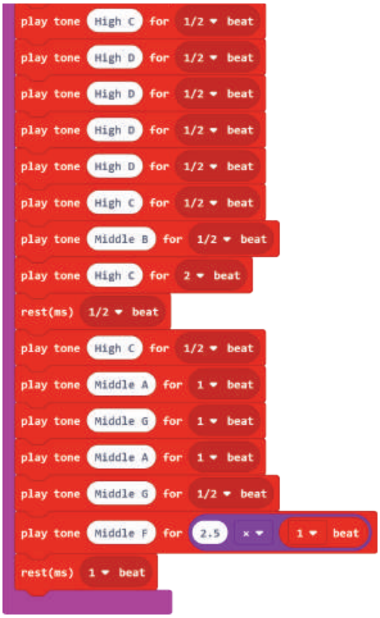
There
is
n't
an
o-
cean
too
deep



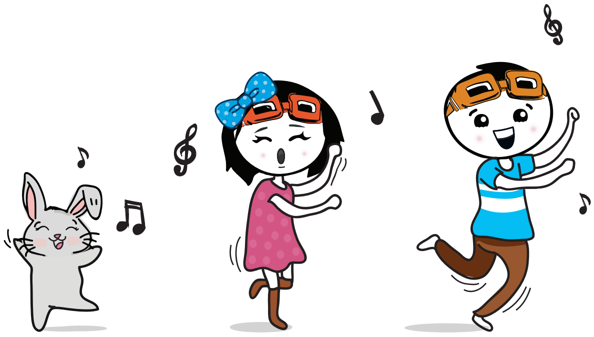
```
on button B pressed
    play tone Middle C for 1/2 beat
    play tone Middle D for 1/2 beat
    play tone Middle F for 1/2 beat
    play tone Middle G for 1/2 beat
    play tone Middle F for 2.5 × 1 beat
    rest(ms) 1/2 beat
    play tone Middle C for 1/2 beat
    play tone Middle D for 1/2 beat
    play tone Middle F for 1/2 beat
    play tone Middle D for 1/2 beat
    play tone Middle F for 1/2 beat
    play tone Middle A for 1/2 beat
    play tone High C for 1.5 × 1 beat
    play tone Middle A for 1/2 beat
    play tone High C for 2 beat
    rest(ms) 1/2 beat
    play tone High C for 1/2 beat
    play tone High D for 1/2 beat
    play tone High D for 1/2 beat
    play tone High D for 1/2 beat
    play tone High D for 1/2 beat
    play tone Middle A for 1/2 beat
    play tone High D for 1/2 beat
    play tone High C for 2 beat
    rest(ms) 1/2 beat
```

A
moun-
tain
so
high
it
can
keep,

Keep
me
a-
way
…
...

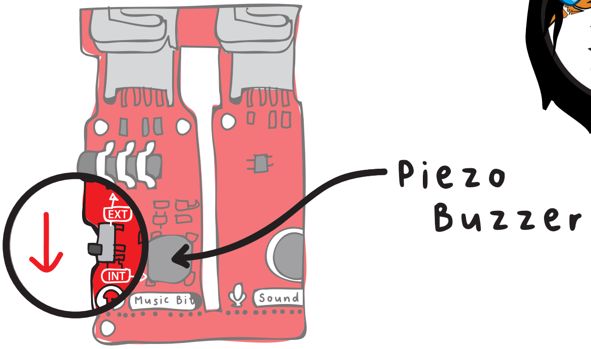


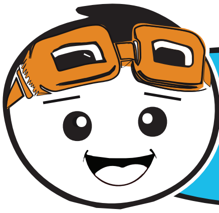
```
    play tone High C for 1/2 beat
    play tone High D for 1/2 beat
    play tone High D for 1/2 beat
    play tone High D for 1/2 beat
    play tone High D for 1/2 beat
    play tone High C for 1/2 beat
    play tone Middle B for 1/2 beat
    play tone High C for 2 beat
    rest(ms) 1/2 beat
    play tone High C for 1/2 beat
    play tone Middle A for 1 beat
    play tone Middle G for 1 beat
    play tone Middle A for 1 beat
    play tone Middle G for 1/2 beat
    play tone Middle F for 2.5 × 1 beat
    rest(ms) 1 beat
```



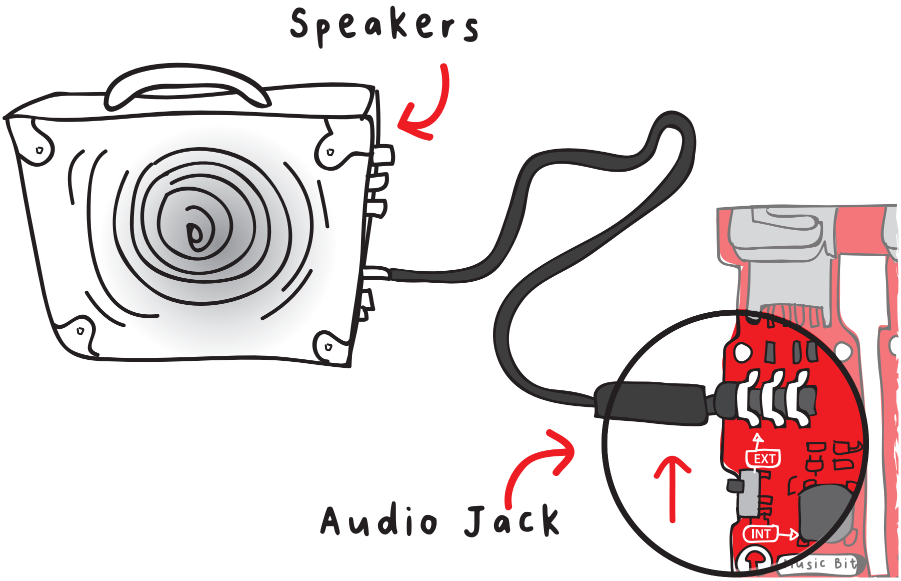
Step 9 Flash the completed code to your EDU:BIT.

EDU:BIT will play the song "I Will Follow You" whenever you press the blue button (Button B) on your EDU:BIT. Remember to power up your EDU:BIT and turn on the piezo buzzer by sliding the switch to INT (internal).



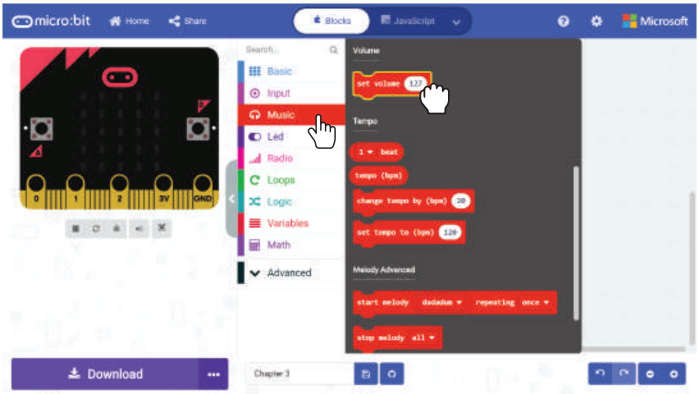


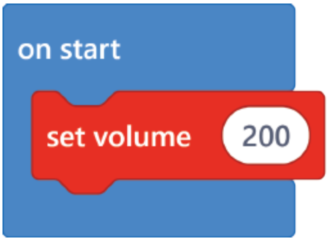
Alternatively you can connect external speakers or headphones to the audio jack on EDU:BIT. You'll need to slide the switch to EXT (external).



Is the music too soft? Or too loud? You can add a [set volume _] block to change the volume, which ranges from 0 to 255 (maximum loudness).

Step 10 Click [**Music**] category and select [**set volume _**] block. Snap the block to [**on start**] block and change the volume value to **200.**

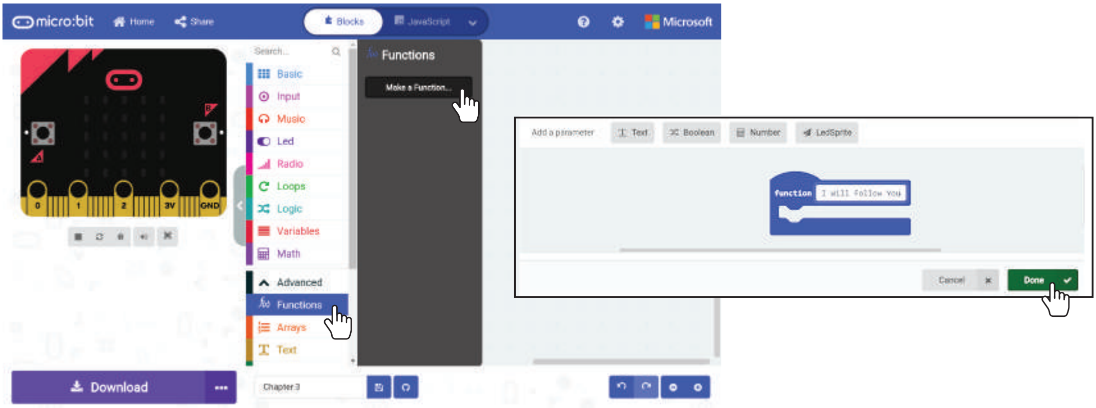




```
on start
    set volume 200
```

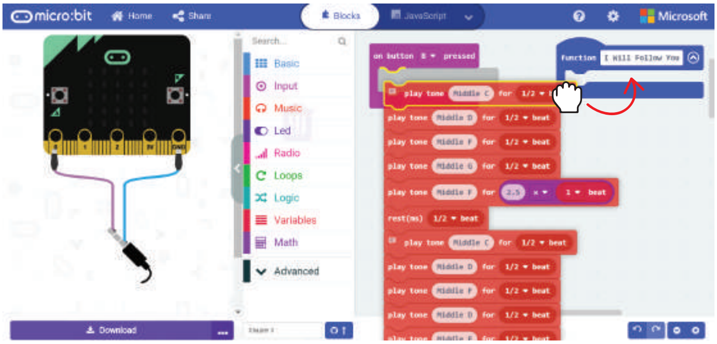
NOTE! We can make blocks of code that perform a specific task into a **function**, for example your code to play the song "I Will Follow You". In programming, a function refers to a routine or a set of procedures. Once a function is defined, it can be used in multiple places in your program without you having to rewrite the same blocks of code over and over again.

Step 11 Click [**Advanced**] category and then select [**Functions**] category. Click [**Make a Function**], and rename doSomething to **'I Will Follow You'** in the pop up window. Then click **'Done'.**



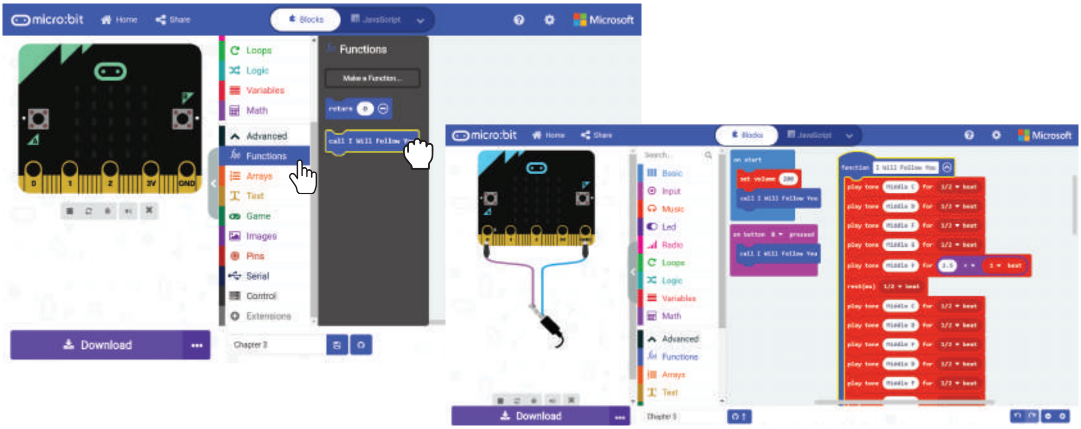
```
function I will Follow You
```

Step 12 A [function I Will Follow You] block will appear on your Editor. Click the topmost block in the [on Button B pressed] block, hold and drag all the blocks to the [function I Will Follow You] slot.



```
on button B ▾ pressed
    play tone Middle C for 1/2 ▾ beat
    play tone Middle D for 1/2 ▾ beat
    play tone Middle F for 1/2 ▾ beat
    play tone Middle G for 1/2 ▾ beat
    play tone Middle F for 2.5 × ▾ 1 ▾ beat
    rest(ms) 1/2 ▾ beat
    play tone Middle C for 1/2 ▾ beat
    play tone Middle D for 1/2 ▾ beat
    play tone Middle F for 1/2 ▾ beat
    play tone Middle D for 1/2 ▾ beat
    play tone Middle F for 1/2 ▾ beat

function I Will Follow You
```

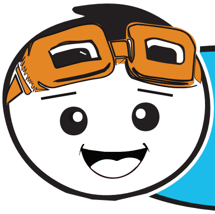
Step 13 Click [Functions] category and then select [call I Will Follow You] block. Duplicate the block. Snap the [call I Will Follow You] blocks to the [on start] block and [on button B pressed] block. Here's the sample code:



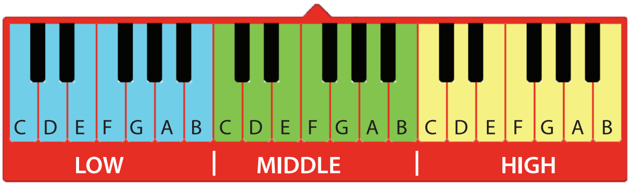
```
on start
    set volume 200
    call I Will Follow You

on button B ▾ pressed
    call I Will Follow You

function I Will Follow You
    play tone Middle C for 1/2 ▾ beat
    play tone Middle D for 1/2 ▾ beat
    play tone Middle F for 1/2 ▾ beat
    play tone Middle G for 1/2 ▾ beat
    play tone Middle F for 2.5 × ▾ 1 ▾ beat
    rest(ms) 1/2 ▾ beat
    play tone Middle C for 1/2 ▾ beat
    play tone Middle D for 1/2 ▾ beat
    play tone Middle F for 1/2 ▾ beat
    play tone Middle D for 1/2 ▾ beat
    play tone Middle F for 1/2 ▾ beat
```

Step 14 Flash the completed code to your EDU:BIT. Enjoy the music~

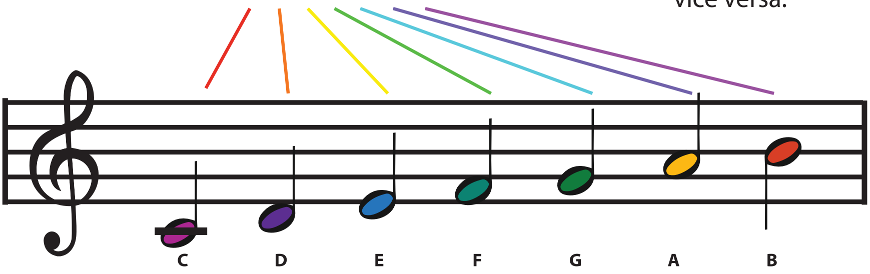
BREAK THE CODE



You can program EDU:BIT to play other songs if you know how to read music. Here's a simple guide to help you to "decode" a music score.



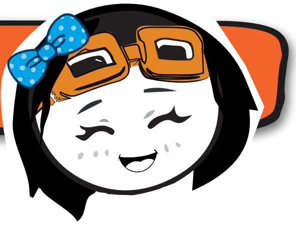
The position of a music note on the staff (i.e. the five horizontal lines) tells us which tone to play. The higher the note sits on the staff, the higher the pitch or frequency of the sound, and vice versa.



Sign	Rest	Relative Length	Duration
𝅝	𝄻	Whole Note	4 beats
𝅗𝅥	𝄼	Half Note	2 beats
♩	𝄾 or 𝄽	Quarter Note	1 beat
♪	𝄾	Eighth Note	1/2 beat
𝅘𝅥𝅯	𝄿	Sixteenth Note	1/4 beat

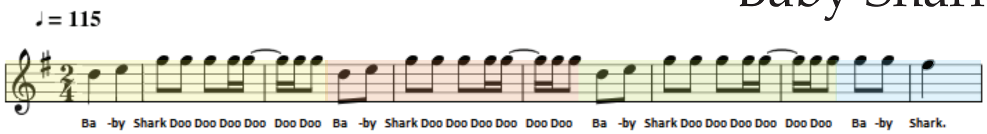
Different musical notations are used to tell us the duration (i.e. how long) a note is to be played.

BREAK THE CODE



Using the clues given, can you "decode" the following tune?

Baby Shark


♩ = 115
Ba -by Shark Doo Doo Doo Doo Doo Doo Ba -by Shark Doo Doo Doo Doo Doo Doo Ba -by Shark Doo Doo Doo Doo Doo Doo Ba -by Shark.

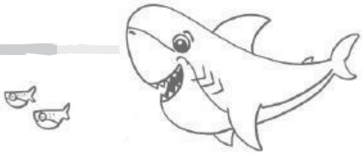
Line 1	Ba	-by	Shark	doo	doo,	doo	doo	doo	doo
Note	High D	High E	High G		High G	High G	High G		High G
Beat	1		1/2	1/2	1/2		1/2	1/4	1/2

Line 2	Ba	-by	Shark	doo	doo,	doo	doo	doo	doo
Note	High D		High G	High G	High G		High G	High G	High G
Beat	1/2	1/2	1/2		1/2	1/4	1/2		1/2

Repeat the same for Line 3

Line 4	Ba	-by	Shark
Note		High G	High F#
Beat	1/2	1/2	

Program EDU:BIT to play Baby Shark tune when the yellow button (Button A) and blue button (Button B) are pressed at the same time.



NOTE!

Use [set volume _] block to adjust the loudness of the tune.

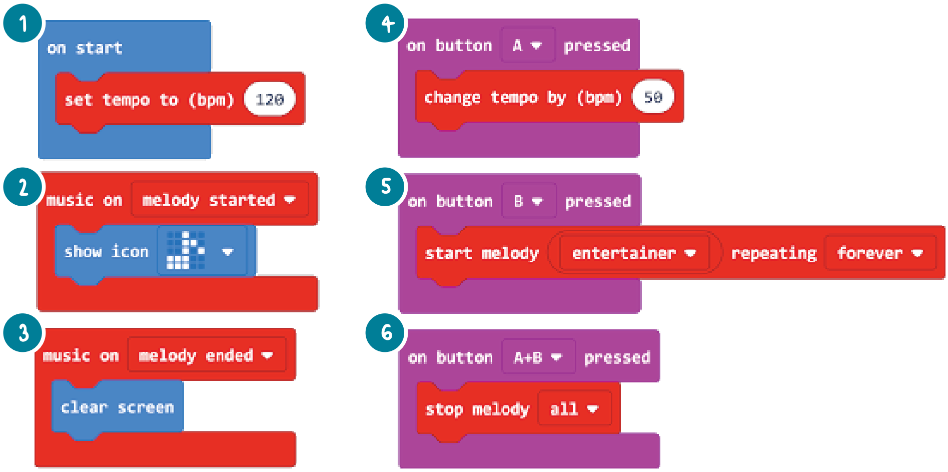
EXPLORE MORE BLOCKS

#1 You can set the "tempo" (i.e. the pace of your song) using the **[set tempo to (bpm) __]** block. The higher the bpm (beats per minute), the faster or livelier your tune will be. Use the **[change tempo by (bpm) __]** block to change the tempo.

#2 Use **[stop melody _]** block to stop a melody that is currently playing.

#3 You can also use **[music on _]** and its selection of conditions, such as melody started and melody ended, as event triggers in your code.

Here's a sample code:



```
1  on start
     set tempo to (bpm) 120

2  music on melody started ▾
     show icon ▾

3  music on melody ended ▾
     clear screen

4  on button A ▾ pressed
     change tempo by (bpm) 50

5  on button B ▾ pressed
     start melody entertainer ▾ repeating forever ▾

6  on button A+B ▾ pressed
     stop melody all ▾
```

1 In this program, the initial tempo is set at 120bpm.

2 On melody started, the LED matrix will display the musical note icon.

3 On melody ended, all LEDs on the matrix display will be turned off.

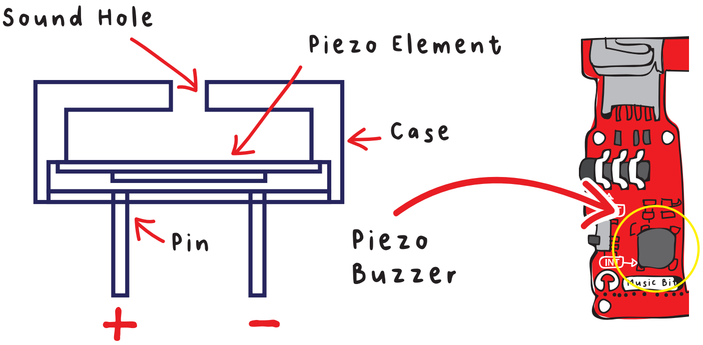
4 Each time Button A is pressed the tempo is increased by 50bpm.

5 The melody 'entertainer' will be played whenever Button B is pressed.

6 The melody stops when Buttons A+B are pressed simultaneously.

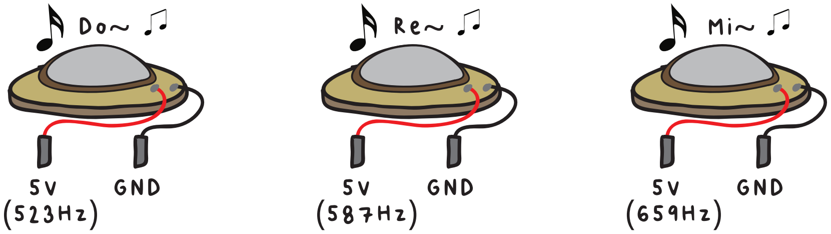
FUN FACT!

A **piezo buzzer** is commonly used to produce sound by vibrating a piece of piezo element when electric signal passes through it.



By changing the frequency of the electric signal, the speed of the vibrations changes; and hence, the piezo buzzer produces sound at a different tone.

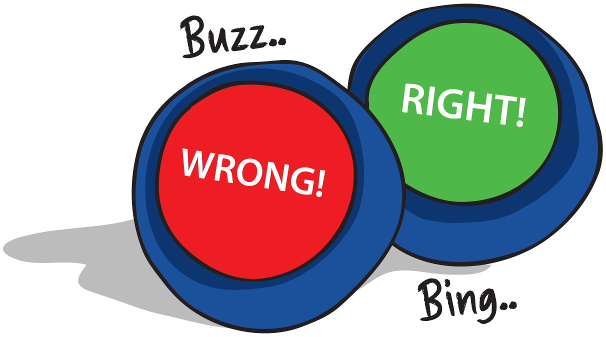
Piezo element



The human ear can hear in the range of 20Hz to 20,000Hz. Any sound below 20Hz is called infrasonic and everything above 20,000Hz is considered ultrasonic.

youtu.be/cxfPNc4Wefo

APPLICATION CHALLENGE



Program EDU:BIT to function as a Game Show Buzzer to signal correct/wrong answers.	
On start	Display a smiley face.
On Button A pressed (Yellow Button)	Display the ✓ icon and play "power up" melody once.
On Button B pressed (Blue Button)	Display the ✗ icon and play "wawawawaa" melody once.
On Buttons A+B pressed	Clear the screen.

CHAPTER 4

Win, Lose or Draw~

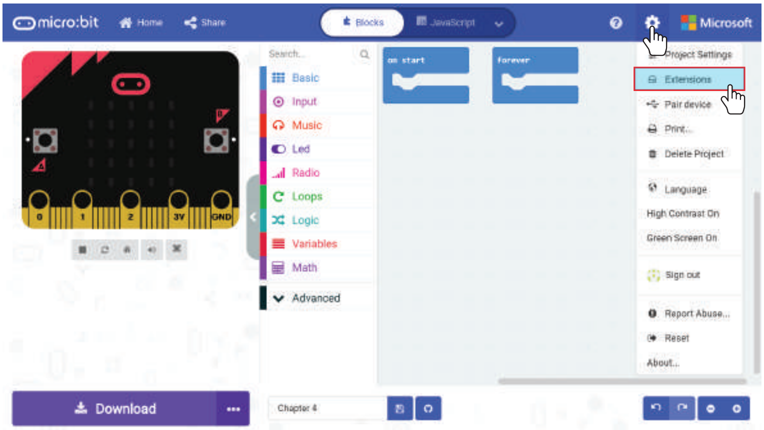
Traffic Light Bit

link.cytron.io/edubit-chapter-4

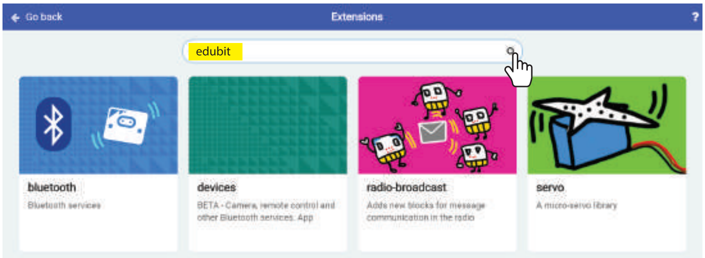
Do you notice a set of red. yellow and green LEDs on your EDU:BIT? That's Traffic Light Bit. To program it, you need to add EDU:BIT extension to your MakeCode Editor. Extensions are sets of custom blocks that we add to the editor to enable us to easily program micro:bit accessories, such as our EDU:BIT board.

LET'S CODE!

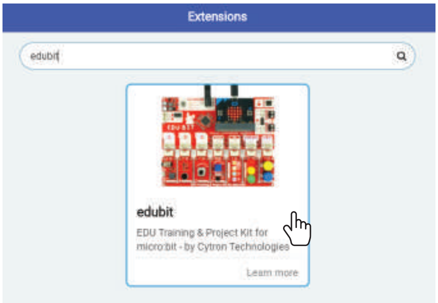
Step 1 Create a new project in your MakeCode Editor. Click the cogwheel icon ⚙ and then select **'Extensions'**. *You need Internet connection to add extensions.

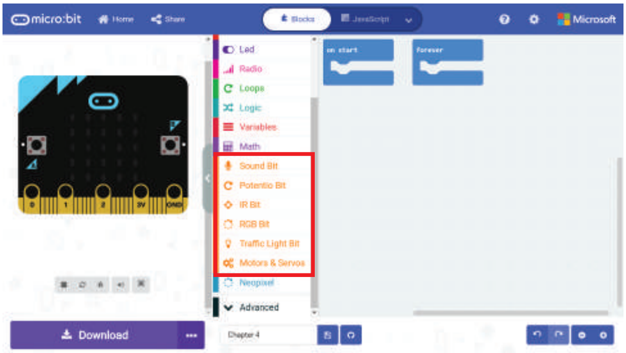


Step 2 Type **"edubit"** into the search box and click Enter.

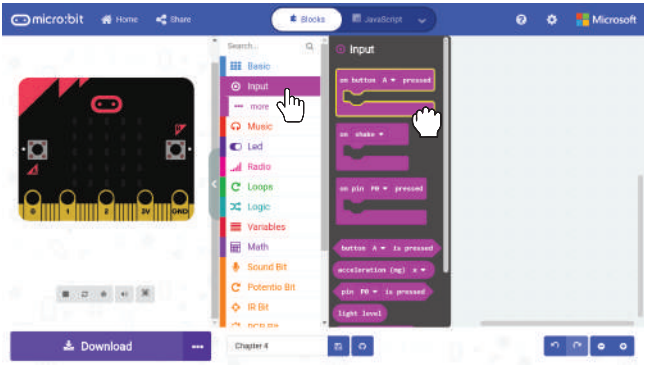

Go back
Extensions
edubit
bluetooth
Bluetooth services
devices
BETA - Camera, remote control and other Bluetooth services. App
radio-broadcast
Adds new blocks for message communication in the radio
servo
A micro-servo library

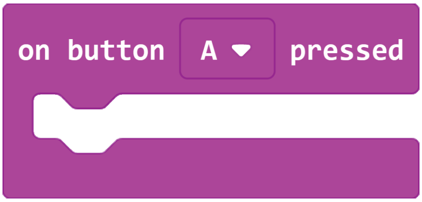
Step 3 Click **'edubit'** extension. Wait for it to load and you'll notice the following new category drawers in your MakeCode Editor.


Extensions
edubit
edubit
EDU Training & Project Kit for micro:bit - by Cytron Technologies
Learn more

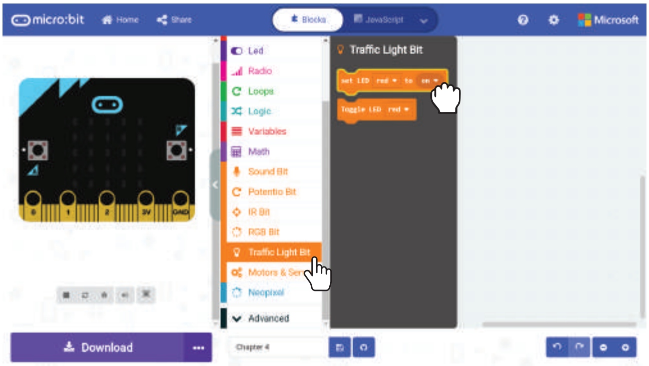


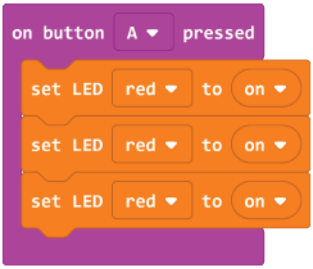
Step 4 Click **[Input]** category and then select **[on button _ pressed]** block.



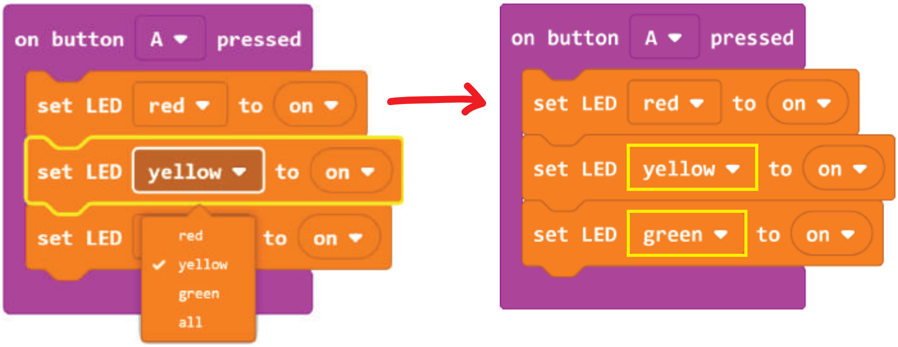


Step 5 Click **[Traffic Light Bit]** category and then select **[set LED _ to _]** block. In the Workspace, right-click on the **[set LED _ to _]** block and then click 'Duplicate'. Repeat until you have three **[set LED _ to _]** blocks. Snap the blocks to the **[on button A pressed]** slot.

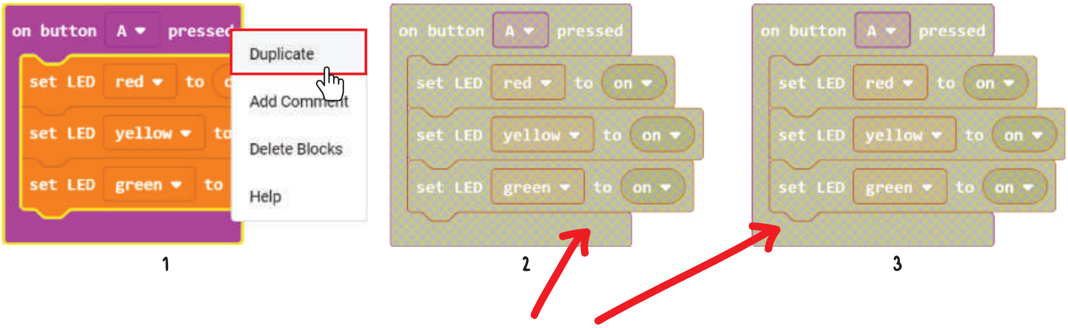




Step 6 Click on the colour selection and change the second and third blocks to **'yellow'** and **'green'** accordingly.



Step 7 Right-click on the **[on button _ pressed]** block and then select “Duplicate’. Repeat to get three sets of the same blocks.



```
on button A pressed
  set LED red to o
  set LED yellow to
  set LED green to
```

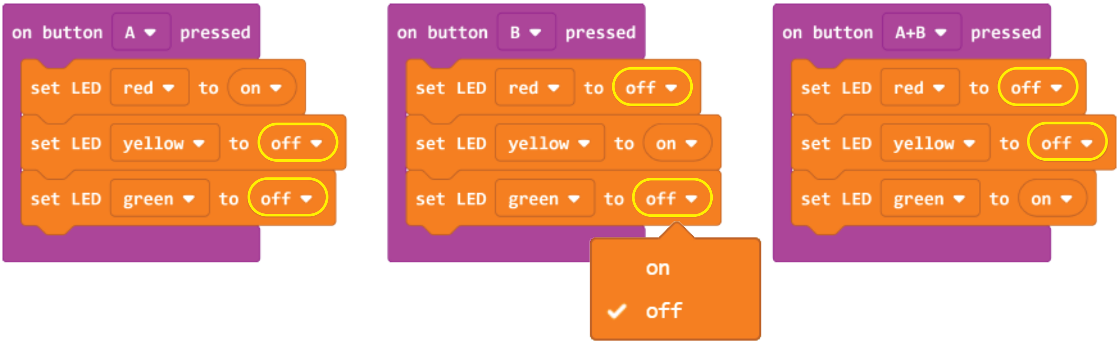
```
on button A pressed
  set LED red to on
  set LED yellow to on
  set LED green to on
```

```
on button A pressed
  set LED red to on
  set LED yellow to on
  set LED green to on
```

*These blocks are disabled and will not run because there are multiple **[on button A pressed]** blocks.

Step 8 Change “A” on the second and third **[on button _ pressed]** blocks to **“B”** and **“A+B”** respectively.

Step 9 Change the state of the LEDs from on to off, as follows.

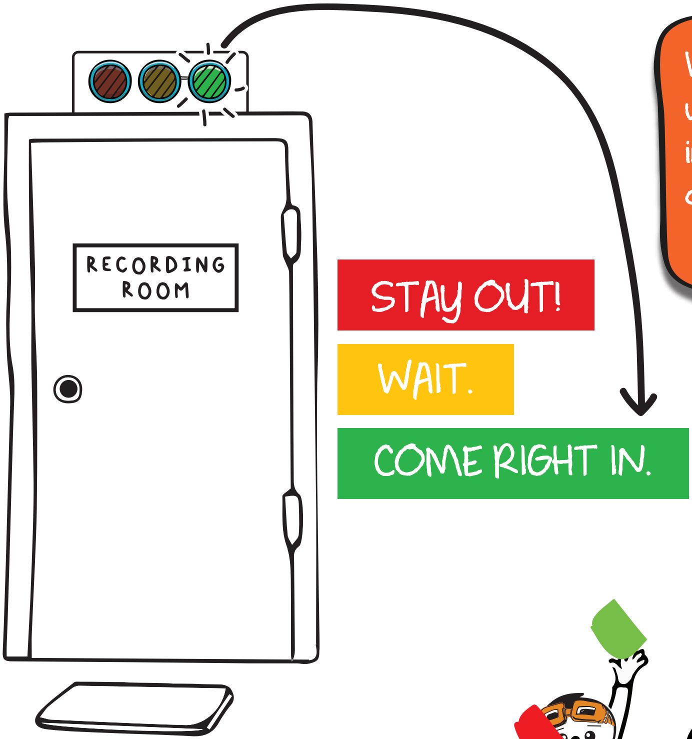


```
on button A pressed
  set LED red to on
  set LED yellow to off
  set LED green to off
```

```
on button B pressed
  set LED red to off
  set LED yellow to on
  set LED green to off
```

```
on button A+B pressed
  set LED red to off
  set LED yellow to off
  set LED green to on
```

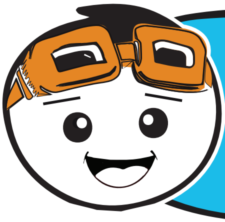
Step 10 Flash the code to your EDU:BIT and observe what happens when you press button A, button B and then both buttons A+B at the same time.



Woohoo...you can now use your EDU:BIT as your personal indicator. Can you think of other uses?

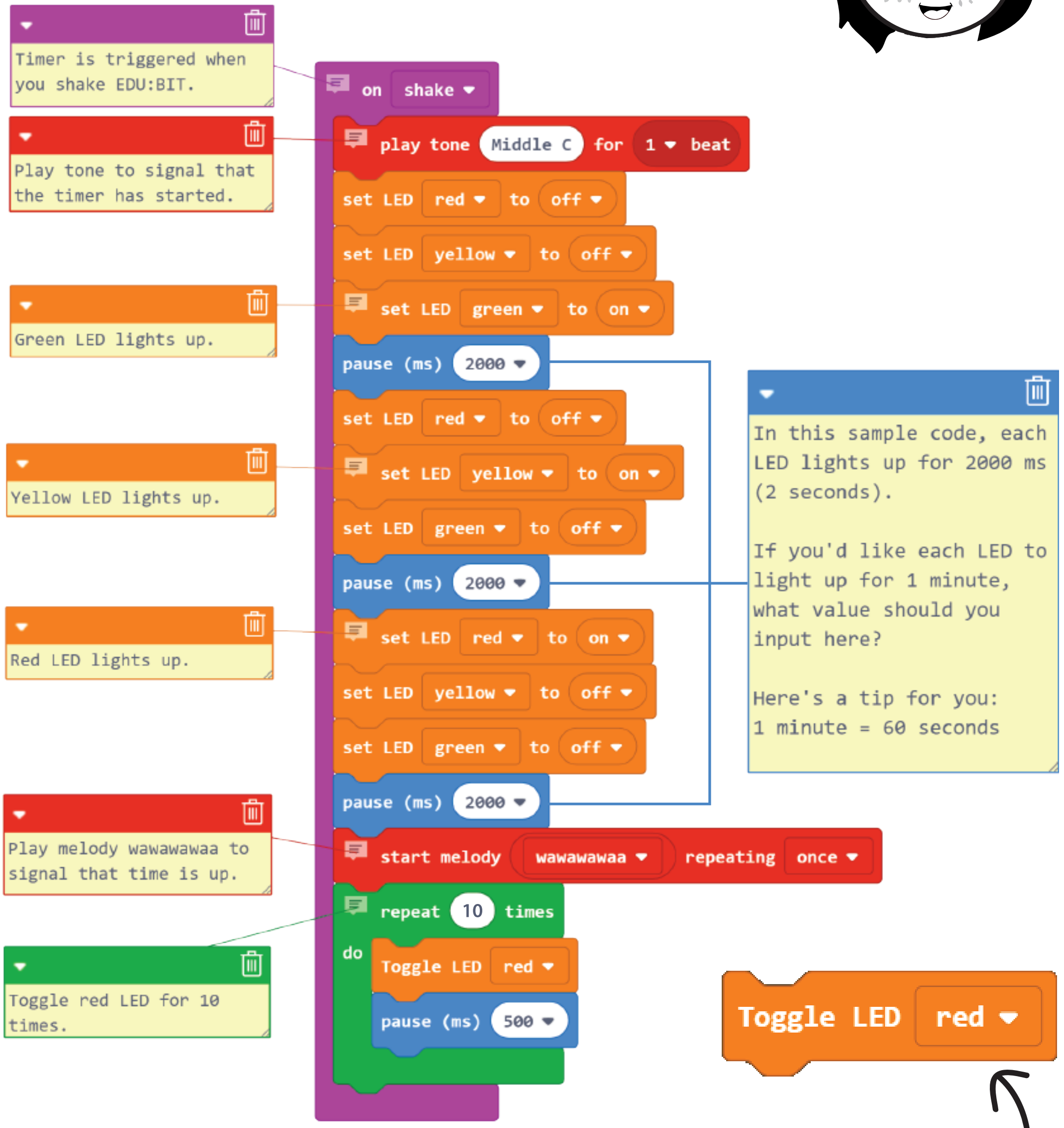
TRAFFIC LIGHT FEEDBACK

- 🔴 "I'm Stuck! Help."
- 🟡 "Still Working/trying"
- 🟢 "Got It!"



LED, or light emitting diode, is an example of a digital output device. It has only two possible states - ON or OFF; whereby ON is commonly represented by 1 (one) and OFF by 0 (zero).

You can also program your EDU:BIT to function as a timer indicator. Here's the sample code.



```
on shake
    play tone Middle C for 1 beat
    set LED red to off
    set LED yellow to off
    set LED green to on
    pause (ms) 2000
    set LED red to off
    set LED yellow to on
    set LED green to off
    pause (ms) 2000
    set LED red to on
    set LED yellow to off
    set LED green to off
    pause (ms) 2000
    start melody wawawawaa repeating once
    repeat 10 times
    do
        Toggle LED red
        pause (ms) 500
```

Timer is triggered when you shake EDU:BIT.

Play tone to signal that the timer has started.

Green LED lights up.

Yellow LED lights up.

Red LED lights up.

Play melody wawawawaa to signal that time is up.

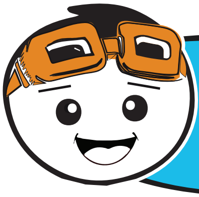
Toggle red LED for 10 times.

In this sample code, each LED lights up for 2000 ms (2 seconds).

If you'd like each LED to light up for 1 minute, what value should you input here?

Here's a tip for you:
1 minute = 60 seconds

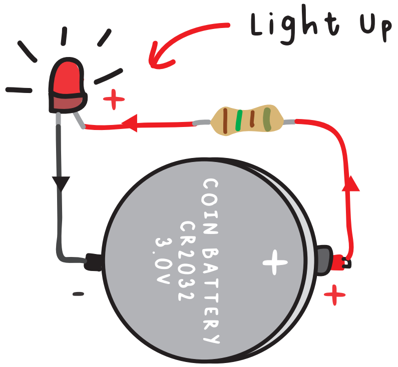
```
Toggle LED red
```

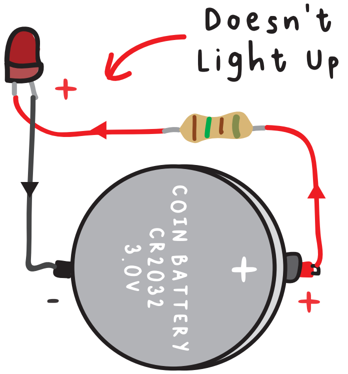


"Toggle" means to switch from one state to another. If the current state is ON, then it will switch to OFF; and vice versa. Thus when we toggle an LED repetitively, the LED will appear to be blinking.

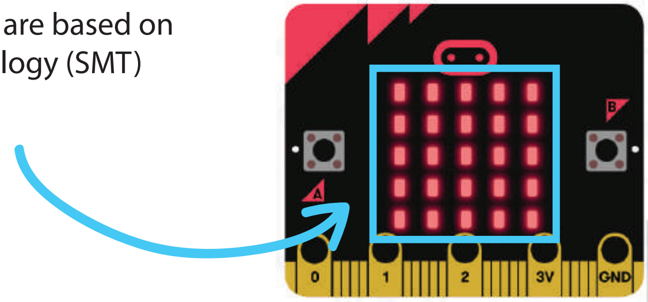
FUN FACT!

A **light-emitting diode (LED)** is a semiconductor device that produces light from electricity. It has 2 terminals, i.e. positive terminal and negative terminal. When an LED is connected in the correct polarity and current flows through it, the LED will emit light.





LEDs used in micro:bit are based on surface-mount technology (SMT) and can be very small.



Besides those on micro:bit, there are another 41 SMT LEDs on EDU:BIT board. Can you spot them all?

Learn more!

youtu.be/qqBmvHD5bCw

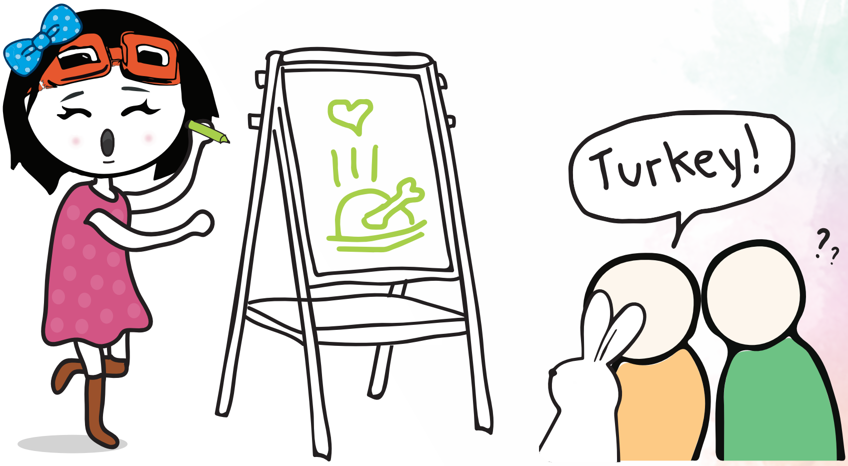
APPLICATION CHALLENGE

Program EDU:BIT to function as a score counter as well as a timer for a game such as "Win, Lose or Draw" and "Charade".	
On start	Set variable Team A = 0 Set variable Team B = 0
On Button A pressed (Yellow Button)	Change Team A by 1 Show Team A's current score
On Button B pressed (Blue Button)	Change Team B by 1 Show Team B's current score
On Buttons A+B pressed (Yellow + Blue Buttons)	Scroll Team A and Team B's score
On shake	Start timer for 1 minute by lighting up **green LED** (for 30 seconds), then **yellow LED** (for 20 seconds) and finally **red LED** (for 10 seconds). Play melody "wawawawaa" when time is up. Toggle the **red LED** for 10 times.

Here's a tip for you. You'll need to make two variables and name them Team A and Team B respectively.

Let's Play

Win, Lose or Draw~



HOW TO PLAY:

- Divide the class into 2 teams - Team A and Team B.
- One member from Team A will start by randomly picking a card. After reading silently the word on the card, shake the EDU:BIT to start the timer (1 minute).
- S/he can then start to draw pictures on the board for team members to guess. No talking or gesturing is allowed!
- One point is awarded to Team A (press Button A or the yellow button) if any of the team members guesses the word or phrase correctly before time is up.
- If Team A fails, Team B can attempt to "steal" a point by giving their best guess.
- Both teams take turns to draw and guess until the end of the game.
- Team with the most points wins!

Scan Here

NOTE!

Scan here to download printable cards with suggested challenge words.
If you are not into drawing, you can try Charade. Same rules apply but instead of drawing, you use gestures to act out the clues for your members to guess. Have fun!

CHAPTER 5

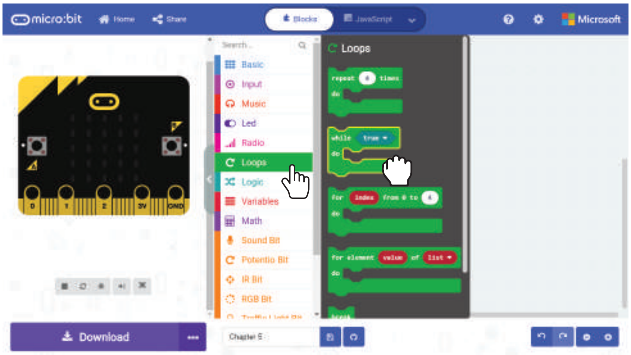
IR Digital Dice.. Let's Roll~

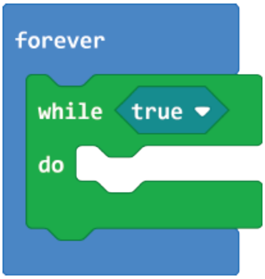
IR Bit

link.cytron.io/edubit-chapter-5

LET'S CODE!

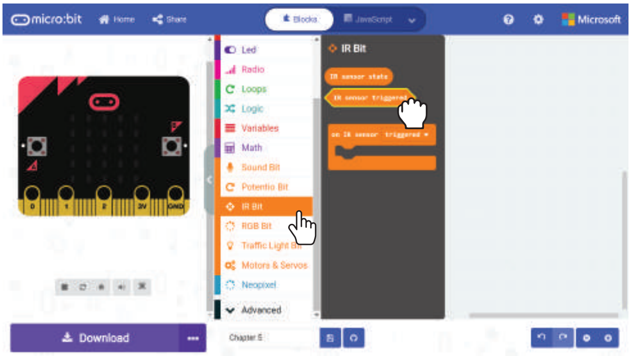
Step 1 Create a new project in your MakeCode Editor and add EDU:BIT extension (you can refer to page 40). Click [Loops] category and then select [while _ do] block. Snap the block to the [forever] slot.

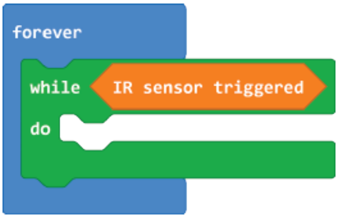




```
forever
  while true ▾
  do
```

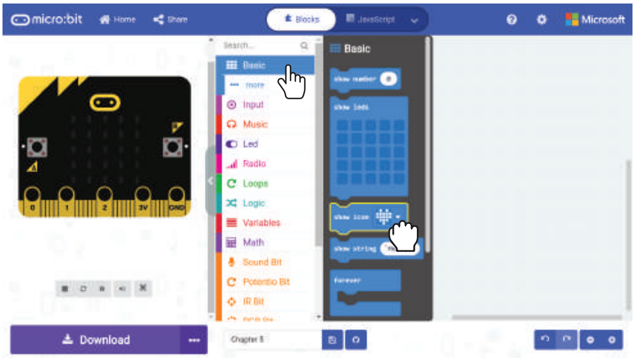
Step 2 Click [IR Bit] category and then select [IR sensor triggered] block. Snap the block to the condition slot on [while _ do] block.

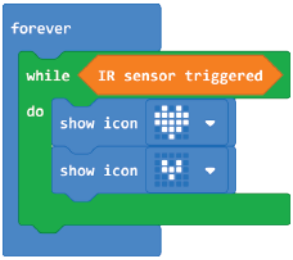




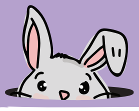
```
forever
  while IR sensor triggered
  do
```

Step 3 Click [Basic] category and add two [show icon] blocks. Change one of the icons to a "small heart". Snap both blocks to the [while _ do] block.

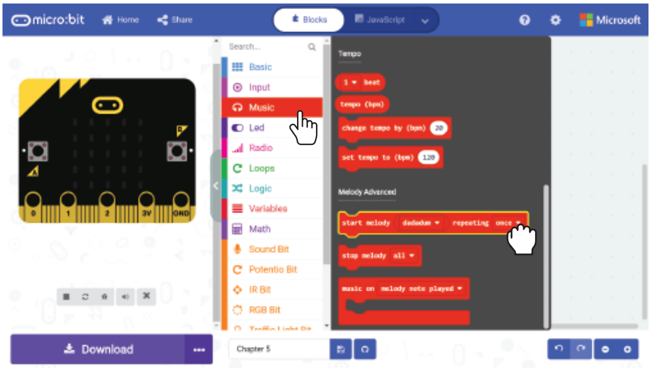


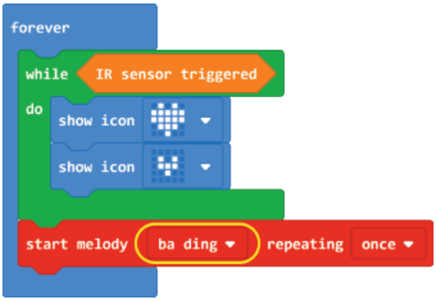


```
forever
  while IR sensor triggered
  do
    show icon ▾
    show icon ▾
```



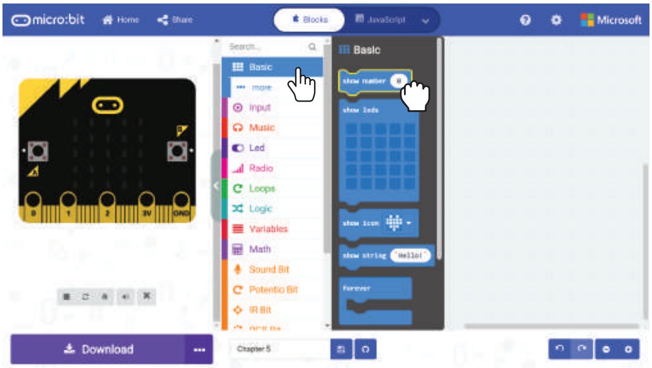
Step 4 Click on [Music] category and then select [start melody _ repeating _] block. Change melody "dadadum" to **"ba ding".**

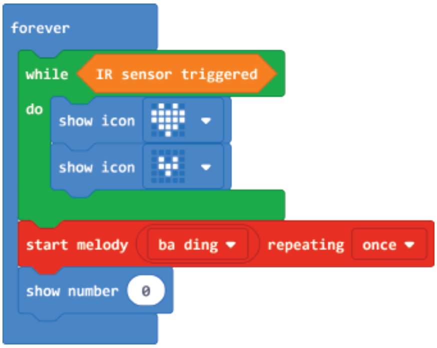




```
forever
  while IR sensor triggered
  do
    show icon
    show icon
  start melody ba ding ▾ repeating once ▾
```

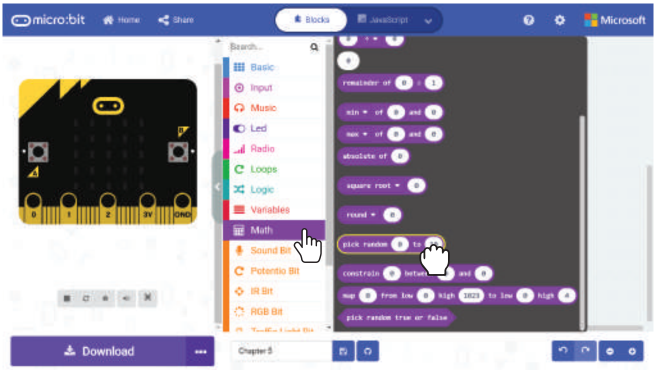
Step 5 Click [Basic] category and then select [show number] block.

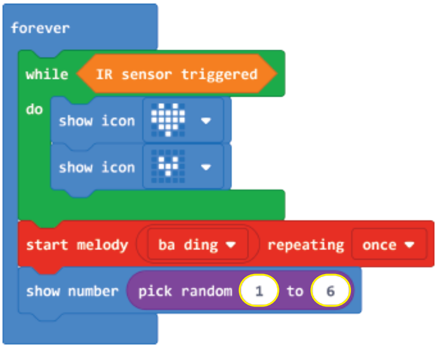




```
forever
  while IR sensor triggered
  do
    show icon
    show icon
  start melody ba ding ▾ repeating once ▾
  show number 0
```

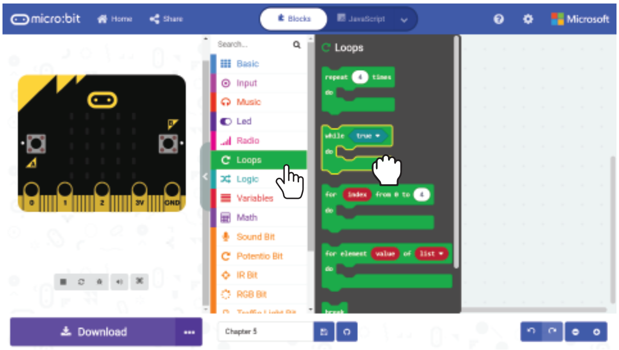
Step 6 Click [Math] category and then select [pick random _ to _] block. Set the numbers to **1** and **6.**

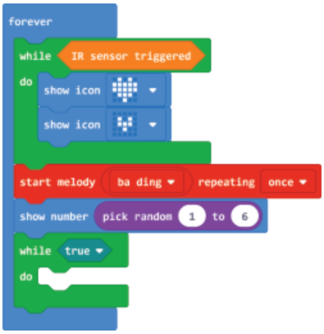




```
forever
  while IR sensor triggered
  do
    show icon
    show icon
  start melody ba ding ▾ repeating once ▾
  show number pick random 1 to 6
```

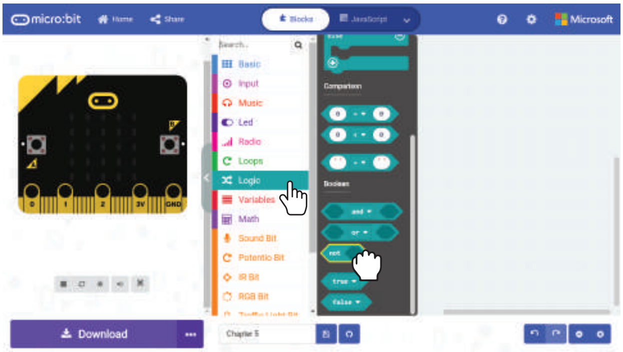
Step 7 Click [Loops] category and then select [while _ do] block.

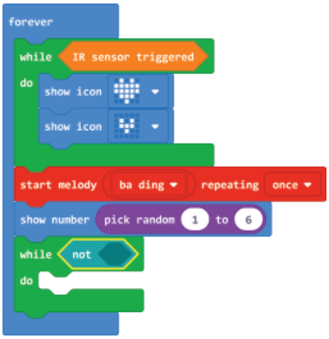




```
forever
  while IR sensor triggered
  do
    show icon
    show icon
  start melody ba ding repeating once
  show number pick random 1 to 6
  while true
  do
```

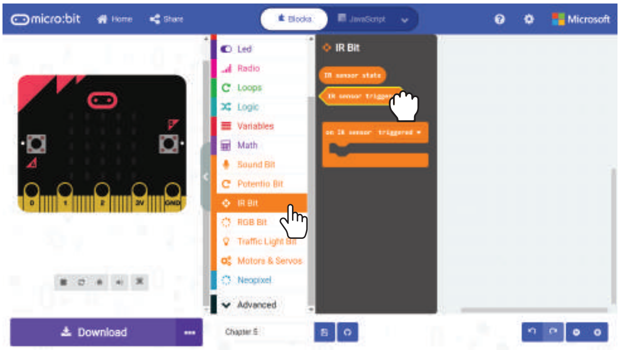
Step 8 Click [Logic] category and then select Boolean [not] block. Snap the block to the condition slot on [while _ do] block.

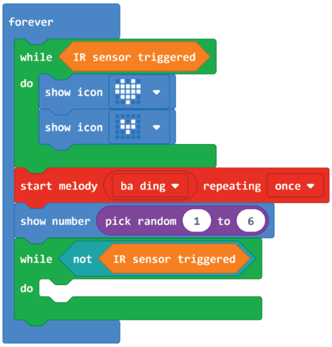




```
forever
  while IR sensor triggered
  do
    show icon
    show icon
  start melody ba ding repeating once
  show number pick random 1 to 6
  while not
  do
```

Step 9 Click [IR Bit] category and then select [IR sensor triggered] block. Snap the block to the empty slot of [not] block.



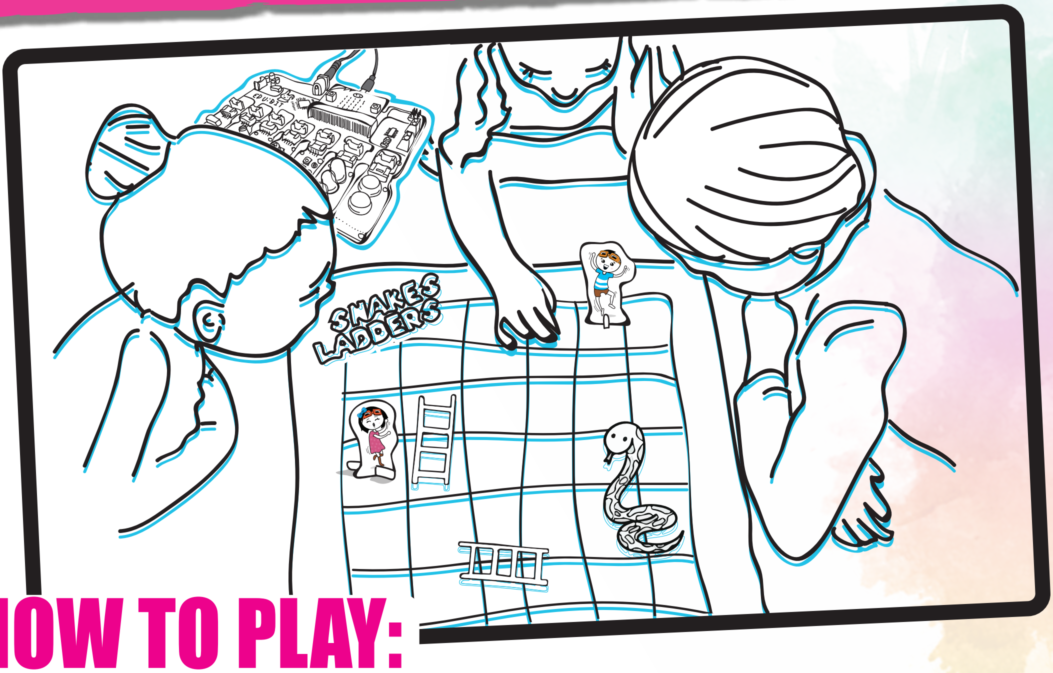


```
forever
  while IR sensor triggered
  do
    show icon
    show icon
  start melody ba ding repeating once
  show number pick random 1 to 6
  while not IR sensor triggered
  do
```

Step 10 Flash the code to your EDU:BIT.

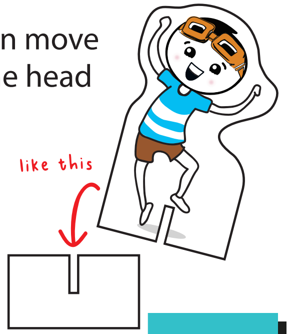
Let's Play

Snakes and Ladders Game



HOW TO PLAY:

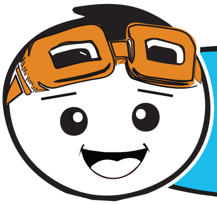
- Each player chooses one character piece and places it on the space that says 'Start Here'.
- Players then take turns to "roll the dice" - place your palm above the IR Bit. When you see a beating heart animation, remove your palm.
- Move your character piece forward the number of spaces displayed on the LED matrix (between 1 to 6).
- If your character piece lands at the bottom of a ladder, you can move up to the top of the ladder. If your character piece lands on the head of a snake, you must slide down to the tip of the snake's tail.
- The first player to reach 100 is the winner. Have fun!



NOTE!

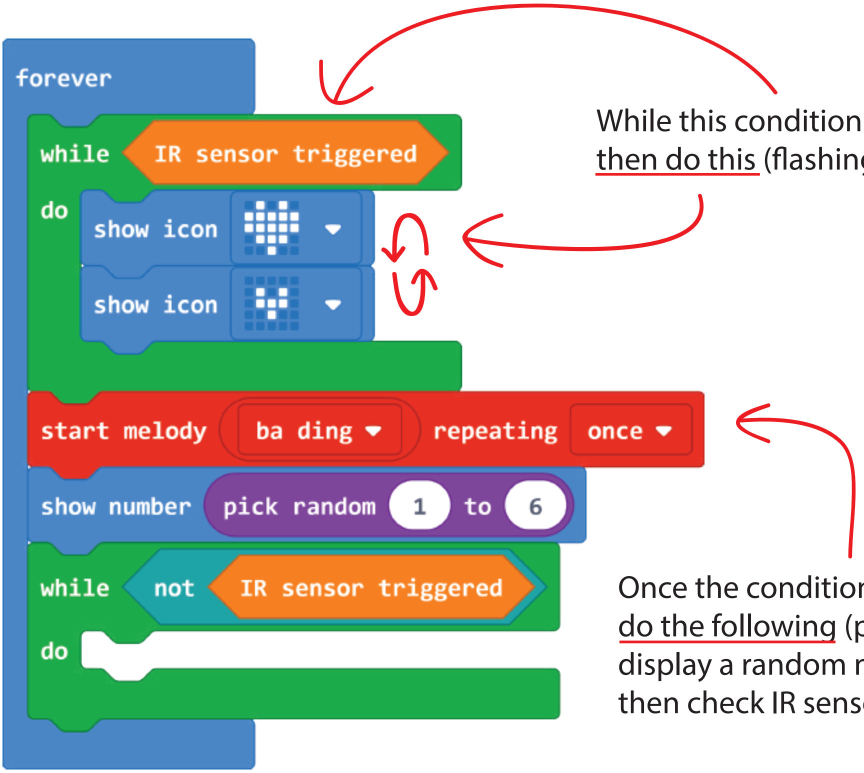
The Snakes and Ladders game board and character pieces are provided in the box Pop out the characters and their bottom stands, and then slot them together to form the character pieces.

BREAK THE CODE



We used [**while _ do**] block from [**Loops**] category in our code earlier. Do you know how a while loop works?

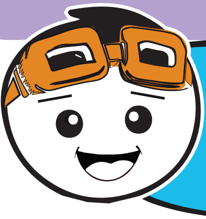
When the program comes to a [**while _ do**] block, it checks the condition. While the condition is met (or is TRUE), the program will execute the block(s) of code in the [**while _ do**] block. It will keep on looping , but once the condition is NOT met (or becomes FALSE), the program will exit the while loop and run the next blocks of code.



```
forever
    while  IR sensor triggered
    do
        show icon  ▾
        show icon  ▾
    start melody  ba ding ▾  repeating  once ▾
    show number  pick random  1  to  6
    while  not  IR sensor triggered
    do
```

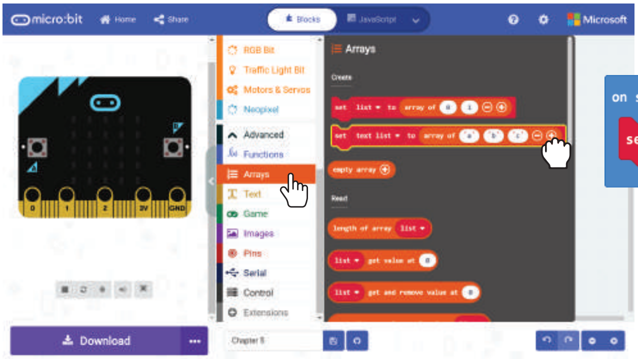
While this condition is met, then do this (flashing heart).

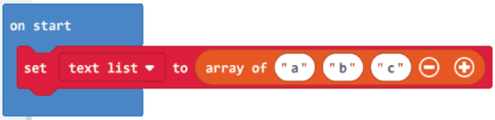
Once the condition is NOT met, do the following (play melody, display a random number and then check IR sensor)



Besides using EDU:BIT as a digital dice, you can also modify the earlier code to get EDU:BIT to help you out when you have several equally tempting options and you cannot decide which to pick, for example - what to eat for lunch? Yum yum~

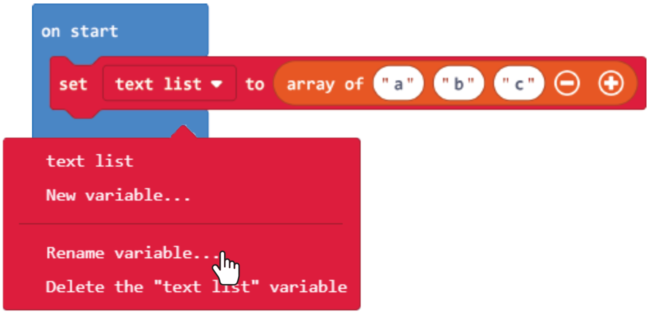
Step 11 Click [**Advanced**] : [**Arrays**] category and select **[set text list to array of _ _ _]** block. Snap the block to the **[Basic] : [on start]** slot.



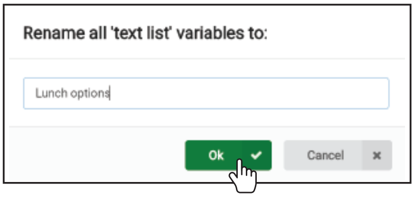


```
on start
  set text list ▾ to array of "a" "b" "c" ⊖ ⊕
```

Step 12 Click **[text list]** block and select **"Rename variable".** Type '**Lunch options**' in the pop up window and then click OK.



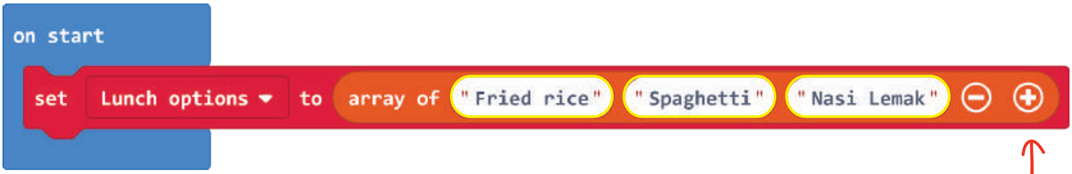
```
on start
  set text list ▾ to array of "a" "b" "c" ⊖ ⊕
```



Rename all 'text list' variables to:

Lunch options

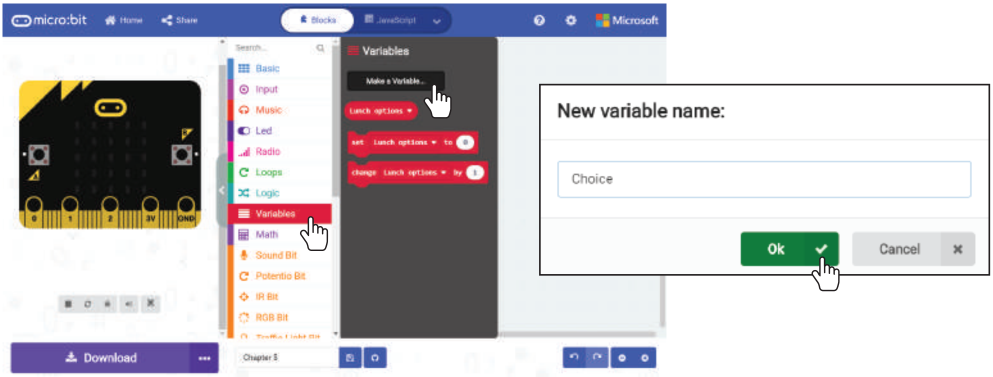
Step 13 Click on the array block one by one and type in one lunch option in each block.



```
on start
  set Lunch options ▾ to array of "Fried rice" "Spaghetti" "Nasi Lemak" ⊖ ⊕
```

Click this button to add more options.

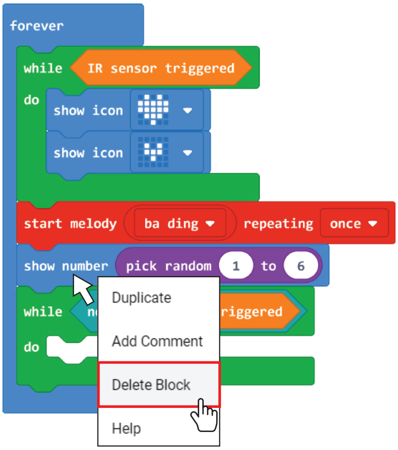
Step 14 Click **[Variable]** category and make a new variable named **'Choice'.**



New variable name:

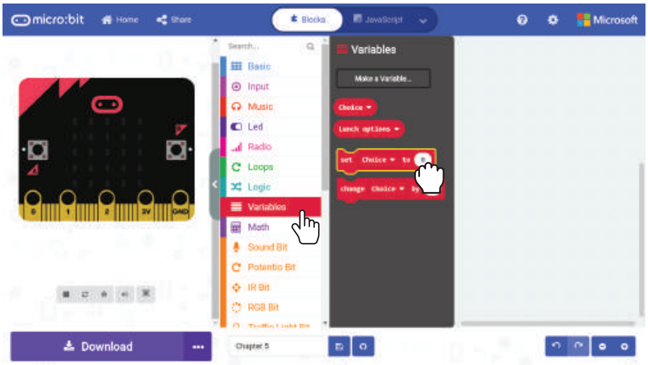
Choice

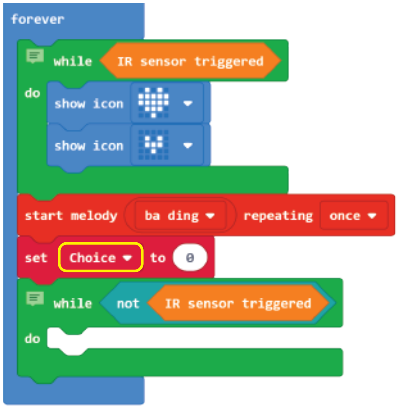
Step 15 Right click on **[show number [pick random _ to _]]** block and then select **'Delete Blocks'.**



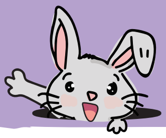
```
forever
    while IR sensor triggered
    do  show icon
        show icon
    start melody ba ding repeating once
    show number pick random 1 to 6
    while not IR sensor triggered
    do
```

Step 16 Click **[Variable]** category and select **[set _ to _]** block. Place the block between the **[start melody _ repeating _]** block and **[while _ do]** block. Change the variable to **'Choice'**.

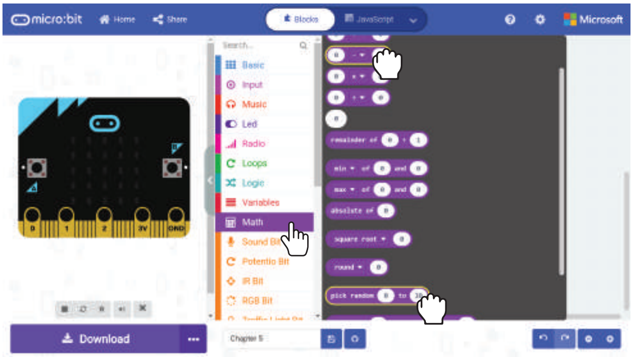


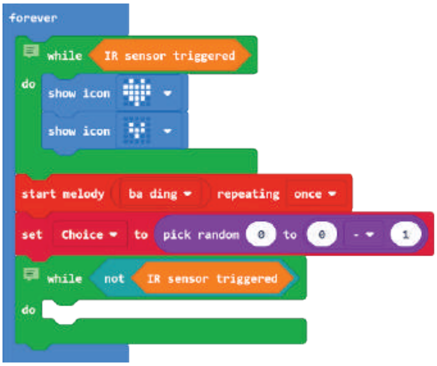


```
forever
    while IR sensor triggered
    do  show icon
        show icon
    start melody ba ding repeating once
    set Choice to 0
    while not IR sensor triggered
    do
```



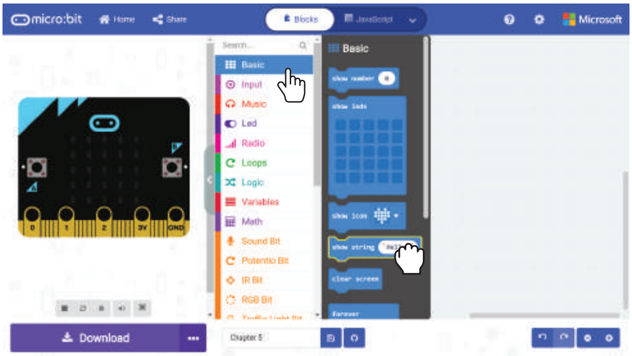
Step 17 Click [Math] category and add [pick random _ to _] and [_ - _] blocks to your code. Change the last value to 1.

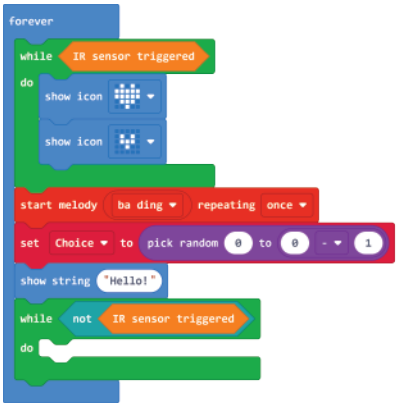




```
forever
  while IR sensor triggered
  do
    show icon
    show icon
  start melody ba ding ▾ repeating once ▾
  set Choice ▾ to pick random 0 to 0 - ▾ 1
  while not IR sensor triggered
  do
```

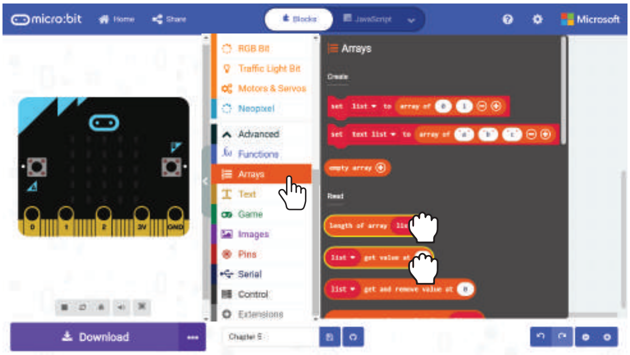
Step 18 Click [Basic] category and select [show string] block.

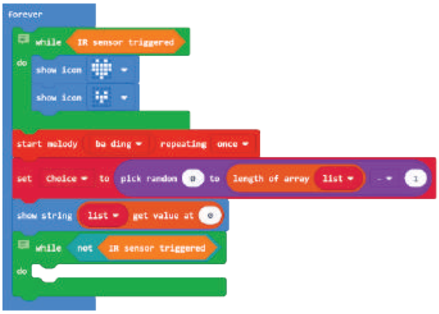




```
forever
  while IR sensor triggered
  do
    show icon
    show icon
  start melody ba ding ▾ repeating once ▾
  set Choice ▾ to pick random 0 to 0 - ▾ 1
  show string "Hello!"
  while not IR sensor triggered
  do
```

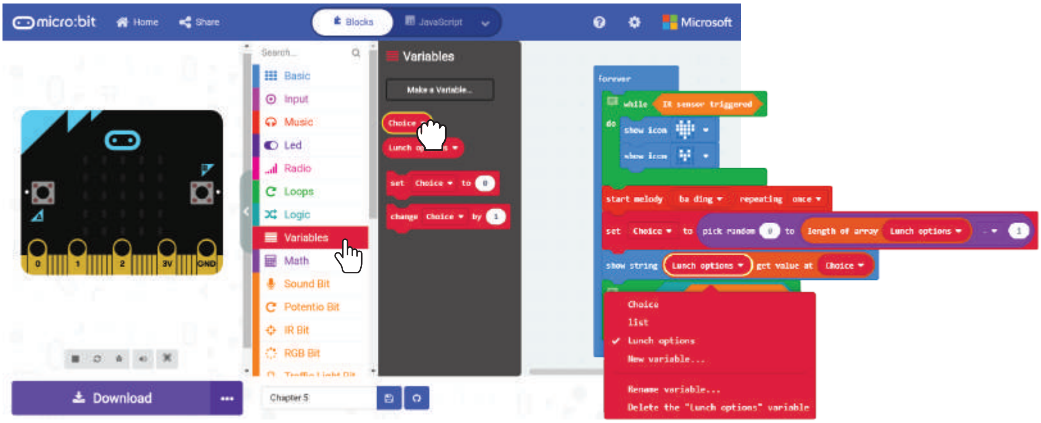
Step 19 Click [Advanced] : [Arrays] category and add [length of array _] and [_ get value at _] blocks to your code.





```
forever
  while IR sensor triggered
  do
    show icon
    show icon
  start melody ba ding ▾ repeating once ▾
  set Choice ▾ to pick random 0 to length of array list ▾ - ▾ 1
  show string list ▾ get value at 0
  while not IR sensor triggered
  do
```

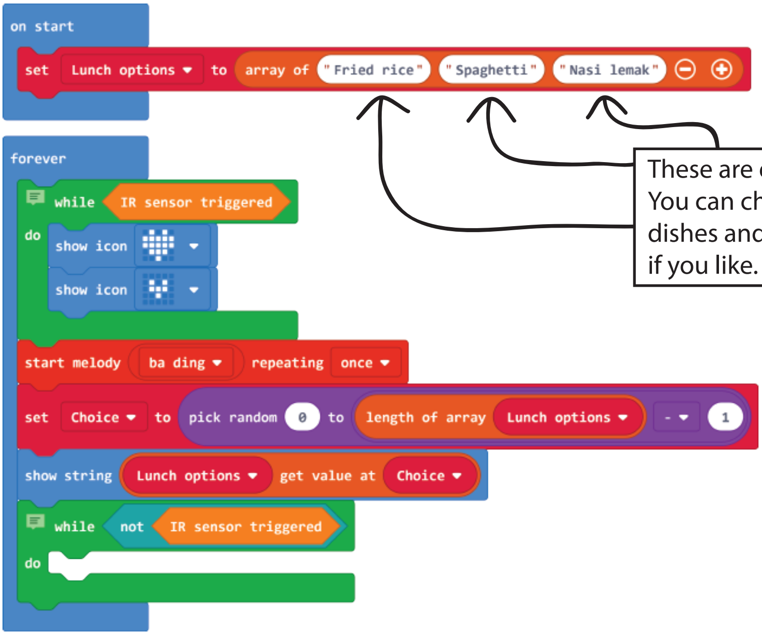
Step 20 Click on **[list]** and change the variable to **'Lunch options'** for both blocks. Finally, click **[Variable]** category and select **[Choice]** block. Snap it to the empty slot of **[_ get value at _]** block.



```
set Choice to 0
change Choice by 1
```

```
forever
    while IR sensor triggered
    do  show icon
        show icon
    start melody ba ding repeating once
    set Choice to pick random 0 to length of array Lunch options - 1
    show string Lunch options get value at Choice
```

Here's your complete "What's for lunch?" code:



```
on start
    set Lunch options to array of "Fried rice" "Spaghetti" "Nasi lemak"

forever
    while IR sensor triggered
    do  show icon
        show icon
    start melody ba ding repeating once
    set Choice to pick random 0 to length of array Lunch options - 1
    show string Lunch options get value at Choice
    while not IR sensor triggered
    do
```

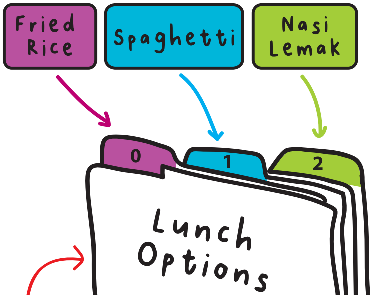
These are examples. You can change to other dishes and add more if you like.

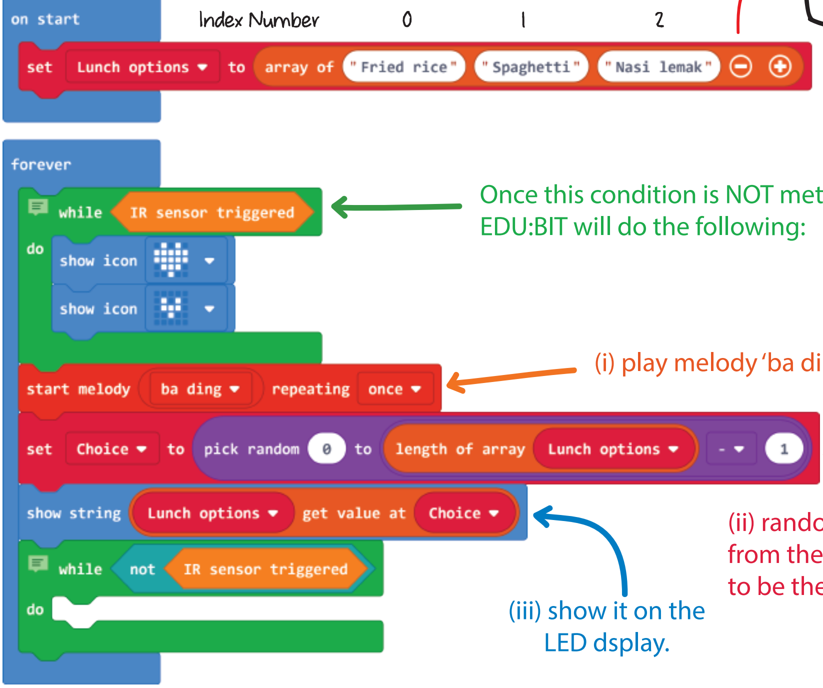
The next time you're unsure what to eat, you can let EDU:BIT decide for you by triggering the IR Bit and then moving your hand away from it. You can also modify this code to help you decide which game to play with your friends. Do you know what you need to change to do that?

BREAK THE CODE

An **array** is a list or collection of related variables. You can think of it as a folder with multiple sections, and each section is used to store a piece of information. We use an array so that we can easily modify our code when we need to add or remove elements from a list.

In this code, for example, we created an array with three elements and named it "Lunch options". We can then easily edit the food item represented by each element. We can also add option(s) or cut down the number of elements in the list by simply clicking the ⊕ or ⊖ buttons.





```
on start
  set Lunch options ▾ to array of "Fried rice" "Spaghetti" "Nasi lemak" ⊖ ⊕

forever
  while IR sensor triggered
  do  show icon ▾
      show icon ▾
  start melody ba ding ▾ repeating once ▾
  set Choice ▾ to pick random 0 to length of array Lunch options ▾ - ▾ 1
  show string Lunch options ▾ get value at Choice ▾
  while not IR sensor triggered
  do
```

Once this condition is NOT met, EDU:BIT will do the following:

(i) play melody 'ba ding' once,

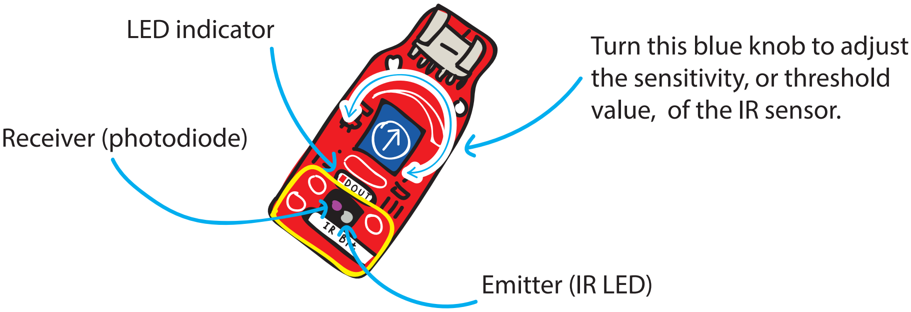
(ii) randomly pick an item from the lunch options list to be the 'choice', and

(iii) show it on the LED dsplay.

In programming, we start counting from 0, instead of 1. Hence, "Fried rice" is at index number 0 in the list and the last element "Nasi lemak" is at index number 2, even though it is the third item.

FUN FACT!

An **infrared (IR) sensor** is an electronic instrument that is commonly used to detect obstacles. The IR sensor consists of two parts - an emitter (IR LED) and a receiver (photodiode).

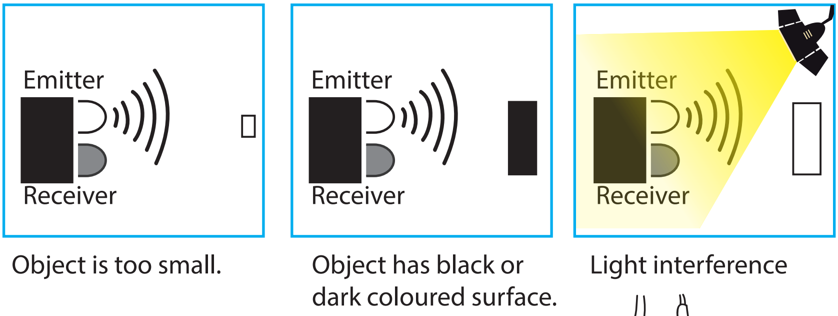

LED indicator
Receiver (photodiode)
Turn this blue knob to adjust the sensitivity, or threshold value, of the IR sensor.
Emitter (IR LED)
DOUT
IR Bit

How does it work?

The IR LED emits IR light which will be reflected to the receiver if an object is placed in front of the sensor. The IR Bit will be "triggered" if the amount of reflected light is greater than the threshold value. When triggered, the LED indicator on IR Bit will light up

If there is no object or the object is too far away, then very little or no IR light will be reflected to the receiver. Hence the IR Bit will not be triggered.

However, the IR sensor may not work as expected in the following conditions:



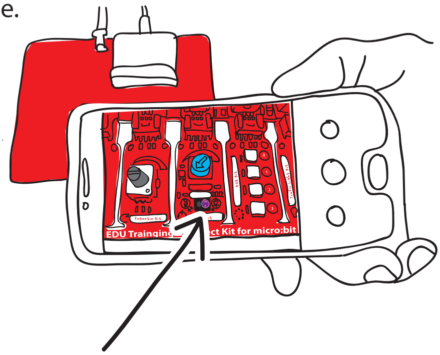
Object is too small.

Object has black or dark coloured surface.

Light interference

Do you know?

Infrared light is invisible to the naked eye, however, you can view infrared light by simply looking at the infrared LED through a phone camera.

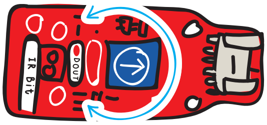


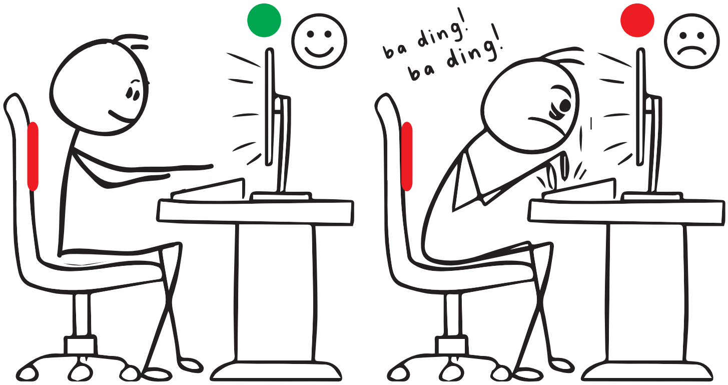
APPLICATION CHALLENGE

Program EDU:BIT to function as an anti-slouch detector.	
On start	Scroll reminder - "Mind your posture."
On IR triggered	Show a smiley face on the LED matrix display and light up green LED.
On IR NOT triggered	Start melody 'ba ding' repeating once. Show a sad face on the LED matrix display and blink the red LED.

How it works:

Attach EDU:BIT to the back of your chair as shown. Sit comfortably with a good posture. Adjust the blue knob on the IR Bit until the LED indicator lights up (IR Bit detects your back). This process is called calibration.



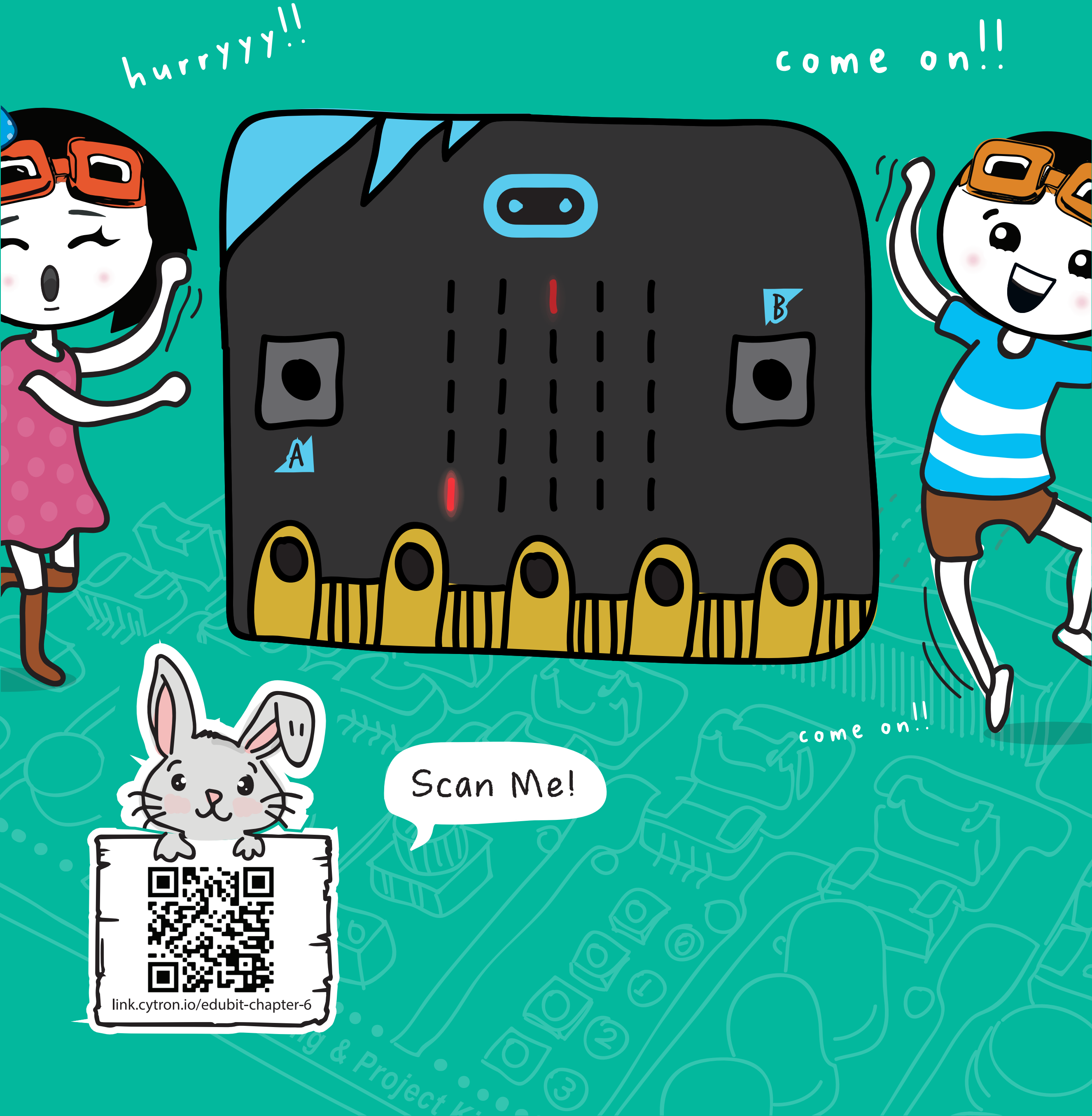


You will need to re-calibrate the IR Bit if you are wearing a different coloured shirt. Do you know why?

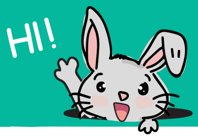
CHAPTER 6

Tag, You're It!

Potentio Bit

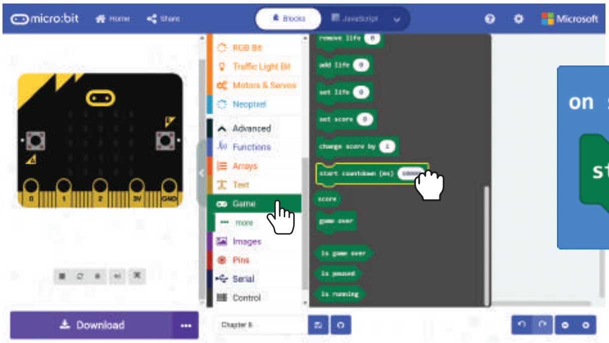


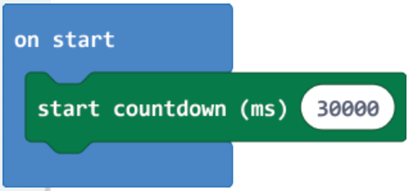
link.cytron.io/edubit-chapter-6



LET'S CODE!

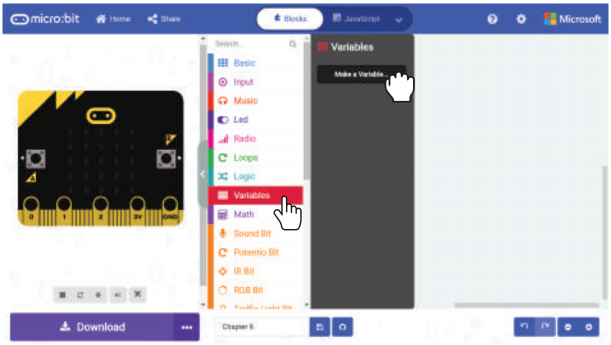
Step 1 In your MakeCode Editor, create a new project and add EDU:BIT extension (you can refer to page 40). Click [**Advanced**] and then [**Game**] category. Select [**start countdown (ms) _**] block, snap it to [**on start**] block and change the value to **30000** .

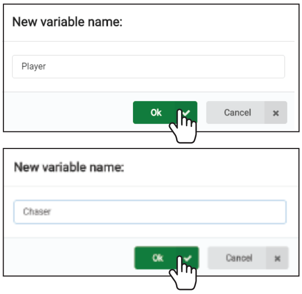




```
on start
    start countdown (ms) 30000
```

Step 2 Click [**Variables**] category and click [**Make a Variable**]. Type **'Player'** in the pop up window and then click OK. Create another variable named **'Chaser'**.





New variable name:

Player

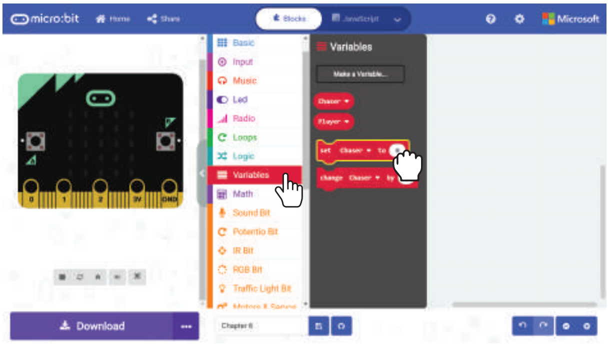
Ok ✓ Cancel ×

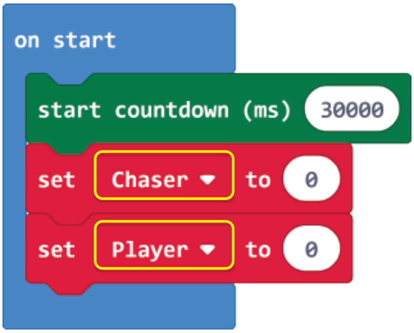
New variable name:

Chaser

Ok ✓ Cancel ×

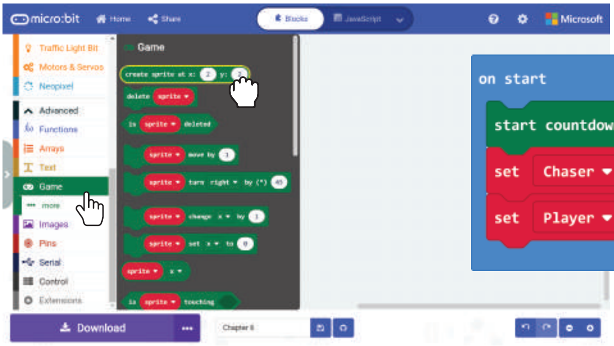
Step 3 Click [**Variables**] category and then select [**set _ to _**] block. Duplicate the [**set _ to _**] block and snap both blocks to the [**on start**] block. Set the variables as **'Chaser'** and **'Player'** respectively.





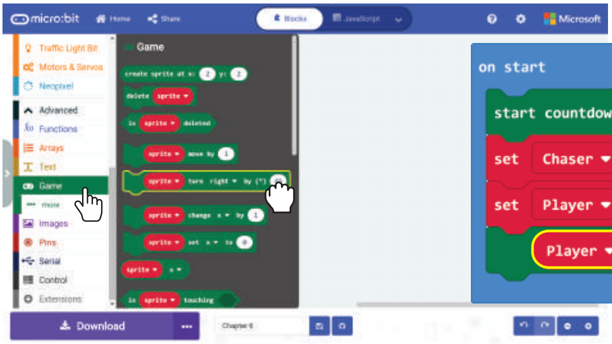
```
on start
    start countdown (ms) 30000
    set Chaser ▾ to 0
    set Player ▾ to 0
```

Step 4 Click [**Advanced**] and then [**Game**] category. Select [**create sprite at x: _ y: _**] block. Duplicate and snap the blocks to the [**set _ to _**] blocks. Change the values to **x: 0 y: 5** for '**Chaser**' and **x: 2 y: 0** for '**Player**'.



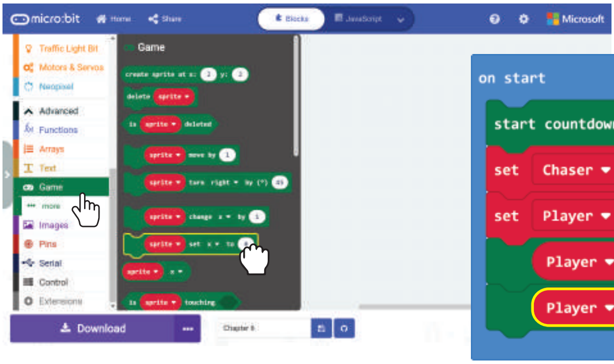
```
on start
    start countdown (ms) 30000
    set Chaser ▾ to create sprite at x: 0 y: 5
    set Player ▾ to create sprite at x: 2 y: 0
```

Step 5 Click [**Advanced**] : [**Game**] category and select [**_ turn _ by (°) _**] block. Place the block in the [**on start**] slot. Select the variable '**Player**' and set the degree to **90**.



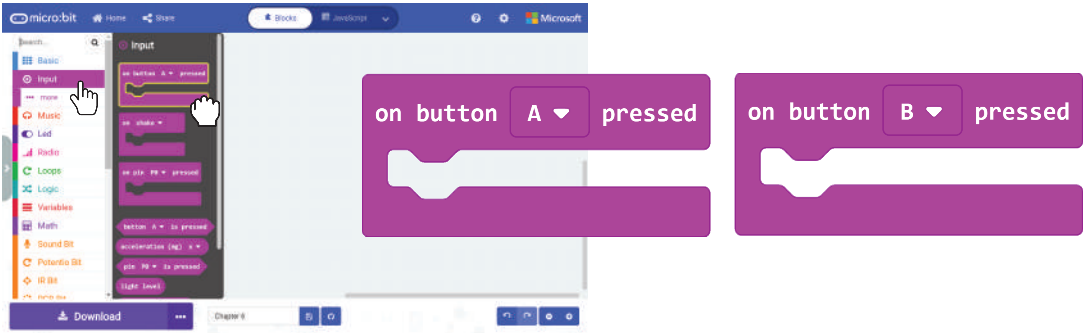
```
on start
    start countdown (ms) 30000
    set Chaser ▾ to create sprite at x: 0 y: 5
    set Player ▾ to create sprite at x: 2 y: 0
    Player ▾ turn right ▾ by (°) 90
```

Step 6 Click [**Advanced**] : [**Game**] category and select [**__ set __ to __**] block. Select the variable '**Player**', change '**x**' to '**brightness**' and set brightness to **50**.



```
on start
    start countdown (ms) 30000
    set Chaser ▾ to create sprite at x: 0 y: 5
    set Player ▾ to create sprite at x: 2 y: 0
    Player ▾ turn right ▾ by (°) 90
    Player ▾ set brightness ▾ to 50
```

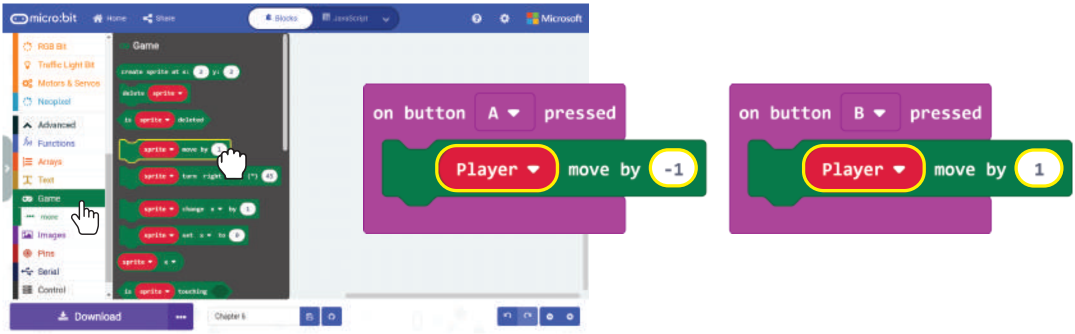
Step 7 Click **[Input]** category and select **[on button _ pressed]** block. Duplicate the block and select button **'B'** on the second **[on button _ pressed]** block.



```
on button A ▾ pressed
```

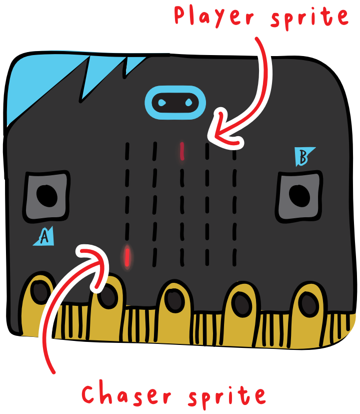
```
on button B ▾ pressed
```

Step 8 Click **[Game]** category and select **[_ move by _]** block. Duplicate and place the blocks in the **[on button _ pressed]** slots. Select variable **'Player'** for both blocks and change the values to **-1**" (on button A pressed) and **1** (on button B pressed).



```
on button A ▾ pressed
    Player ▾ move by -1
```

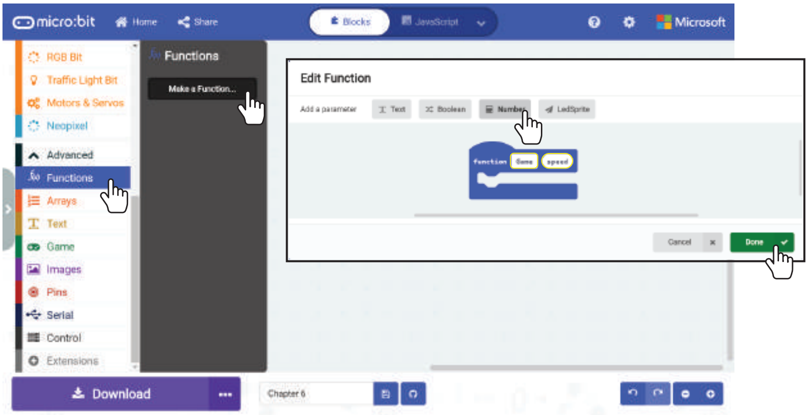
```
on button B ▾ pressed
    Player ▾ move by 1
```



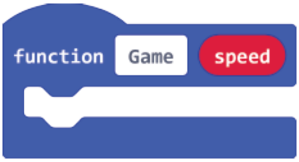
Flash the code to your EDU:BIT. When you press the blue button (Button B), do you notice the dimmer LED light moving downward? That's the Player sprite! A sprite is like "a little LED creature" you can control.

What will happen if you press the yellow button (Button A)?

Step 9 Click [**Advanced**] category and select [**Functions**] category, then click [**Make a Function...**]. In Edit Function window, rename 'doSomething' to **'Game'**. Next, click [**Number**] to add a parameter and rename 'num' to **'speed'** in the function block. Then click **'Done'**.

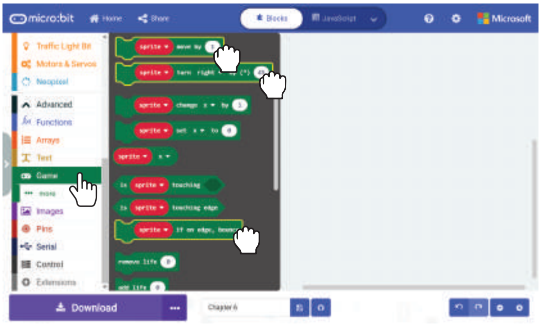


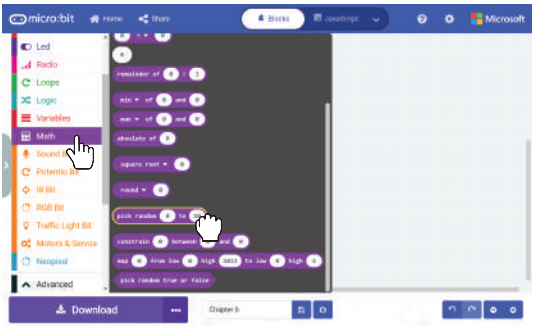
This function block will appear on your Workspace.

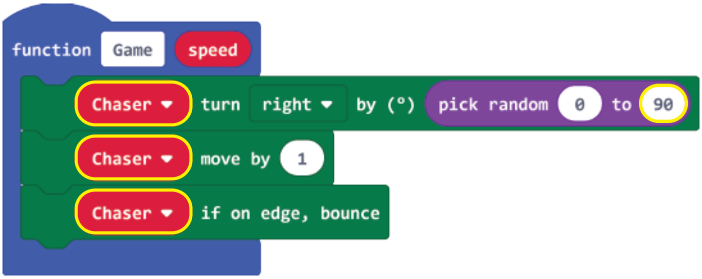


```
function Game speed
```

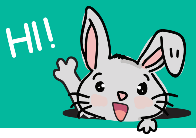
Step 10 Continue building the code by adding blocks from [**Advanced**] : [**Game**] and [**Math**] categories as shown below. Remember to change the variable to '**Chaser**' and the value to **90**.



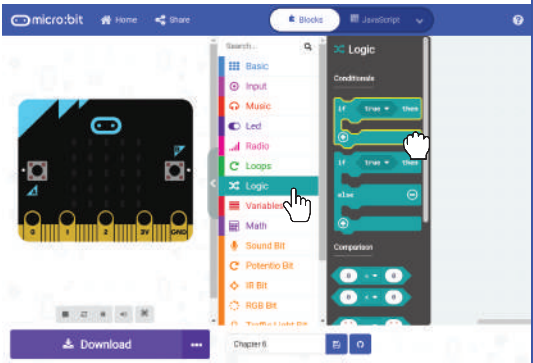


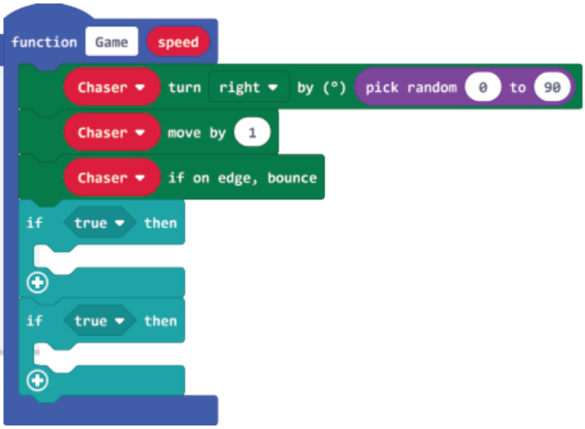


```
function Game speed
    Chaser ▾ turn right ▾ by (°) pick random 0 to 90
    Chaser ▾ move by 1
    Chaser ▾ if on edge, bounce
```



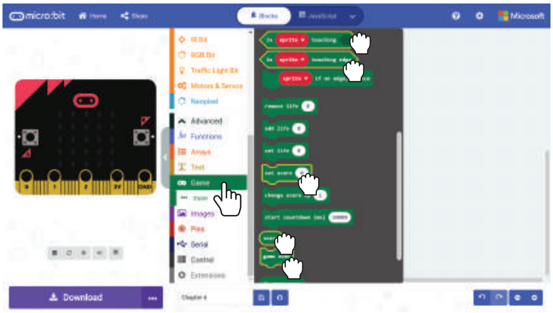
Step 11 Add two **[if-then]** blocks from **[Logic]** category to your code.

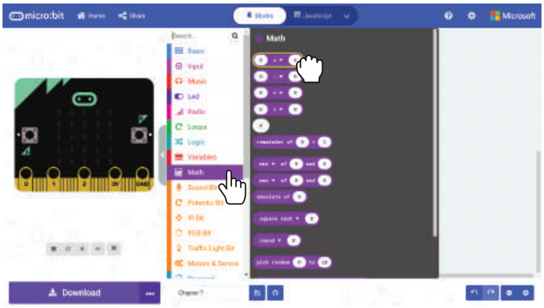


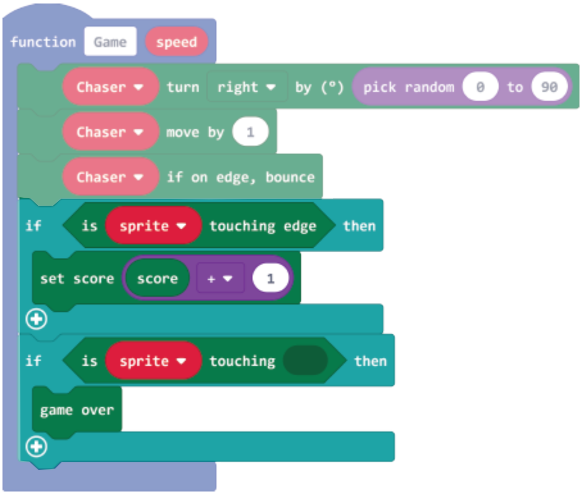


```
function Game speed
    Chaser ▾ turn right ▾ by (°) pick random 0 to 90
    Chaser ▾ move by 1
    Chaser ▾ if on edge, bounce
    if true ▾ then
    ⊕
    if true ▾ then
    ⊕
```

Step 12 Continue building the code by adding blocks from **[Advanced] : [Game]** and **[Math]** categories as shown below.

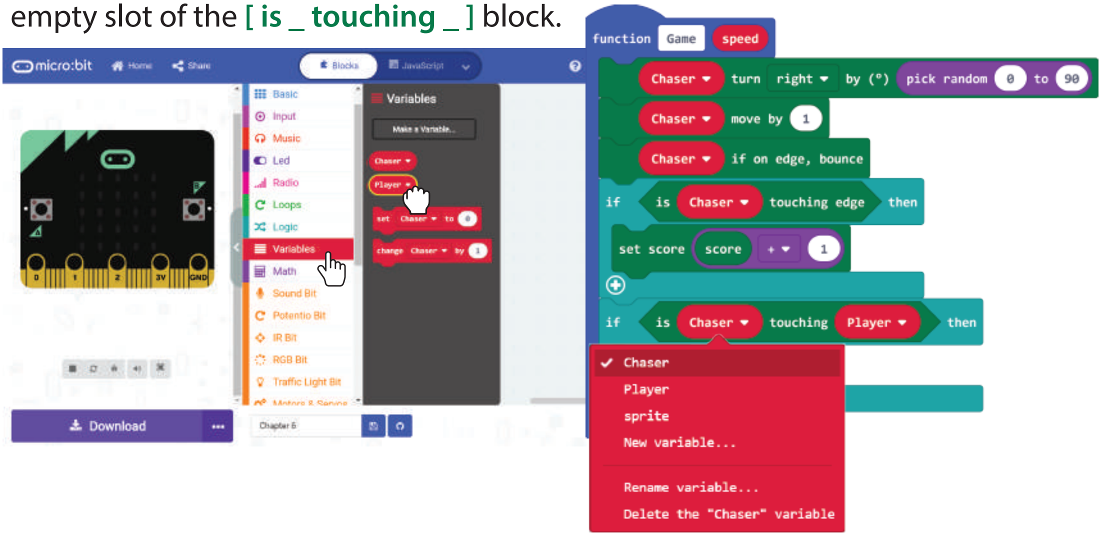






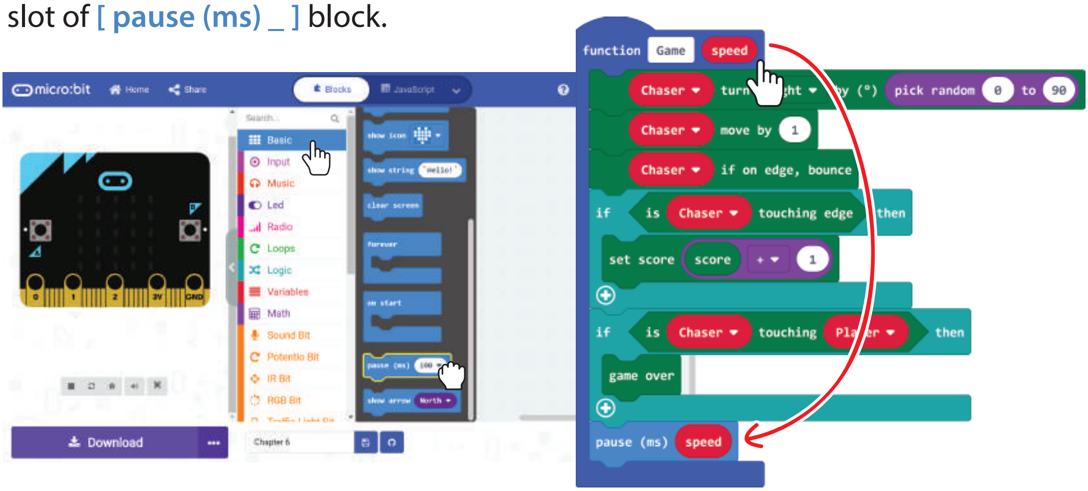
```
function Game speed
    Chaser ▾ turn right ▾ by (°) pick random 0 to 90
    Chaser ▾ move by 1
    Chaser ▾ if on edge, bounce
    if is sprite ▾ touching edge then
        set score score + ▾ 1
    ⊕
    if is sprite ▾ touching then
        game over
    ⊕
```

Step 13 Change both **[sprite]** blocks to **'Chaser'** by clicking it and then selecting **'Chaser'**. Click **[Variables]** category and select **[Player]** block. Snap it to the empty slot of the **[is _ touching _]** block.



```
function Game speed
    Chaser turn right by (°) pick random 0 to 90
    Chaser move by 1
    Chaser if on edge, bounce
    if is Chaser touching edge then
        set score score + 1
    if is Chaser touching Player then
```

Step 14 Click **[Basic]** category and select **[pause (ms) _]** block. Add to your code. Click on **[speed]** on the function block, hold and drag it to the empty slot of **[pause (ms) _]** block.

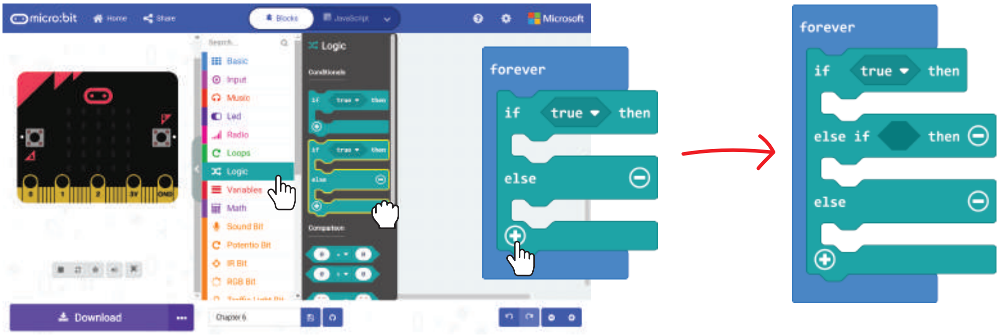


```
function Game speed
    Chaser turn right by (°) pick random 0 to 90
    Chaser move by 1
    Chaser if on edge, bounce
    if is Chaser touching edge then
        set score score + 1
    if is Chaser touching Player then
        game over
    pause (ms) speed
```

You can add excitement to the game by playing a melody when the Chaser touches the Player. Can you figure out which block to add and where to place the block?

Now, let's add different difficulty levels to the game!

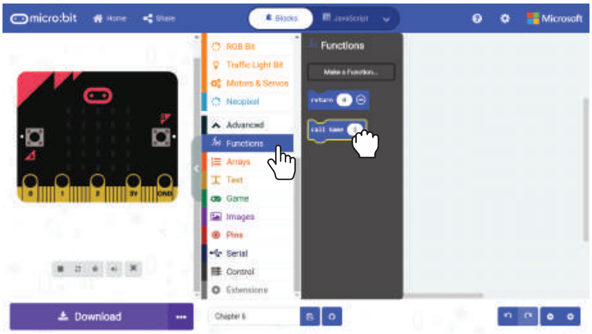
Step 15 Click **[Logic]** category and select **[if-then-else]** block. Place the block in **[forever]** slot. Click the (+) button to add another condition.

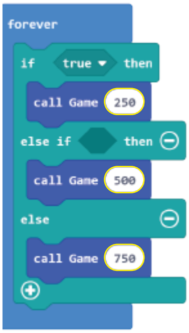


```
forever
  if true then
  else
```

```
forever
  if true then
  else if ⬡ then
  else
```

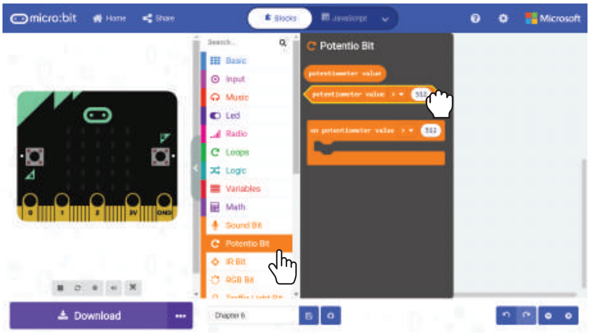
Step 16 Click **[Functions]** category and select **[call Game _]** block. Duplicate the block and snap the blocks to each of the **[if-then-else]** slots. Change the value of the **[call Game _]** blocks to **250, 500** and **750** accordingly.

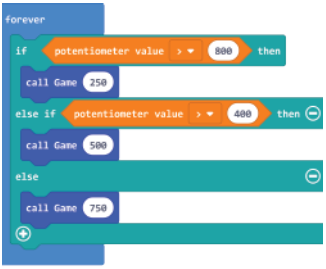




```
forever
  if true then
    call Game 250
  else if ⬡ then
    call Game 500
  else
    call Game 750
```

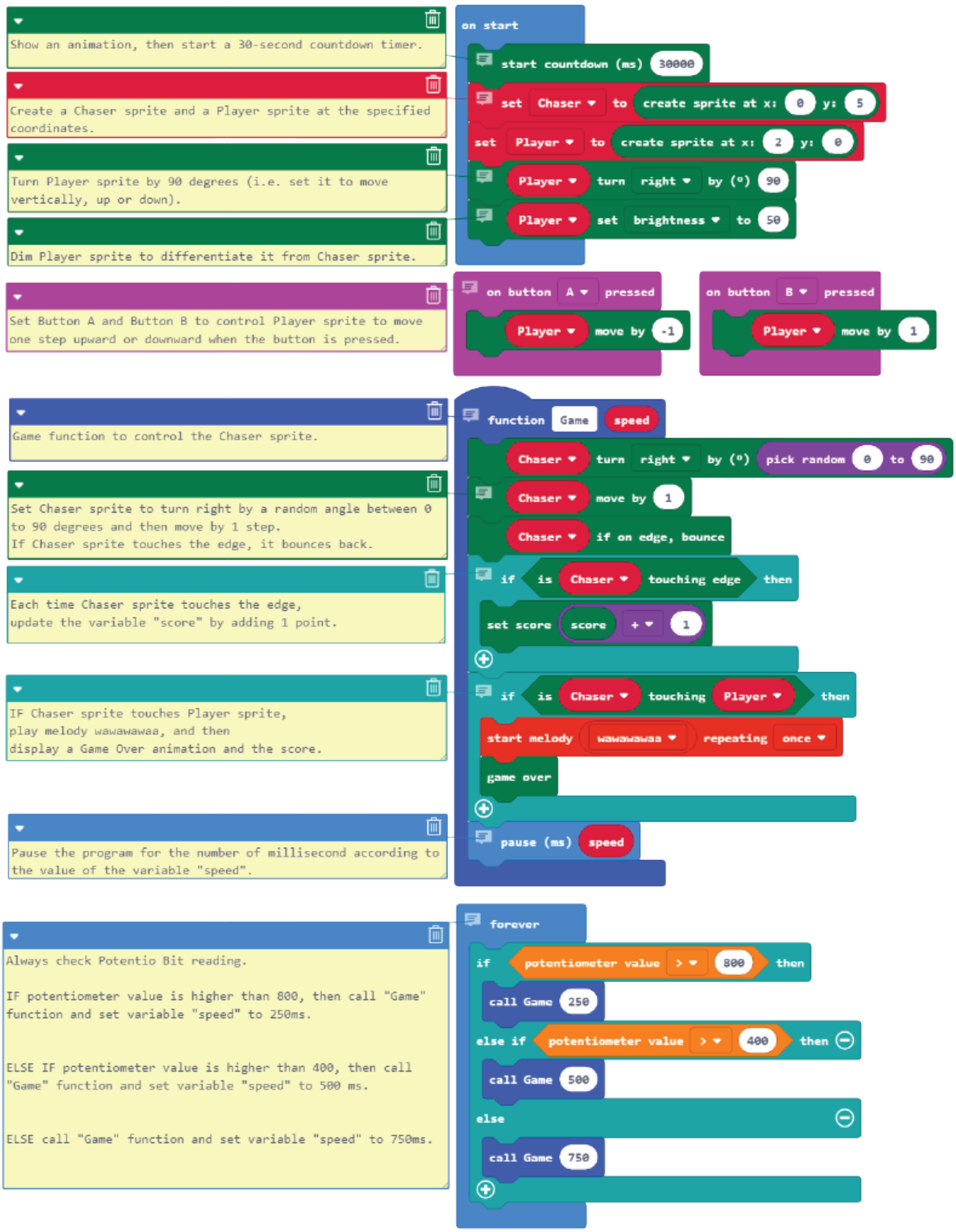
Step 17 Click **[Potentio Bit]** category and select **[potentiometer value >_]** block. Duplicate and place the blocks in the condition slots of the **[if-then-else]** block. Set the value to **800** for the first block and **400** for the second block.





```
forever
  if potentiometer value > 800 then
    call Game 250
  else if potentiometer value > 400 then
    call Game 500
  else
    call Game 750
```

Here's the complete code:



Show an animation, then start a 30-second countdown timer.

Create a Chaser sprite and a Player sprite at the specified coordinates.

Turn Player sprite by 90 degrees (i.e. set it to move vertically, up or down).

Dim Player sprite to differentiate it from Chaser sprite.

```
on start
    start countdown (ms) 30000
    set Chaser to create sprite at x: 0 y: 5
    set Player to create sprite at x: 2 y: 0
    Player turn right by (°) 90
    Player set brightness to 50
```

Set Button A and Button B to control Player sprite to move one step upward or downward when the button is pressed.

```
on button A pressed
    Player move by -1

on button B pressed
    Player move by 1
```

Game function to control the Chaser sprite.

Set Chaser sprite to turn right by a random angle between 0 to 90 degrees and then move by 1 step.
If Chaser sprite touches the edge, it bounces back.

Each time Chaser sprite touches the edge,
update the variable "score" by adding 1 point.

IF Chaser sprite touches Player sprite,
play melody wawawawaa, and then
display a Game Over animation and the score.

Pause the program for the number of millisecond according to the value of the variable "speed".

```
function Game speed
    Chaser turn right by (°) pick random 0 to 90
    Chaser move by 1
    Chaser if on edge, bounce
    if is Chaser touching edge then
        set score score + 1
    if is Chaser touching Player then
        start melody wawawawaa repeating once
        game over
    pause (ms) speed
```

Always check Potentio Bit reading.

IF potentiometer value is higher than 800, then call "Game" function and set variable "speed" to 250ms.

ELSE IF potentiometer value is higher than 400, then call "Game" function and set variable "speed" to 500 ms.

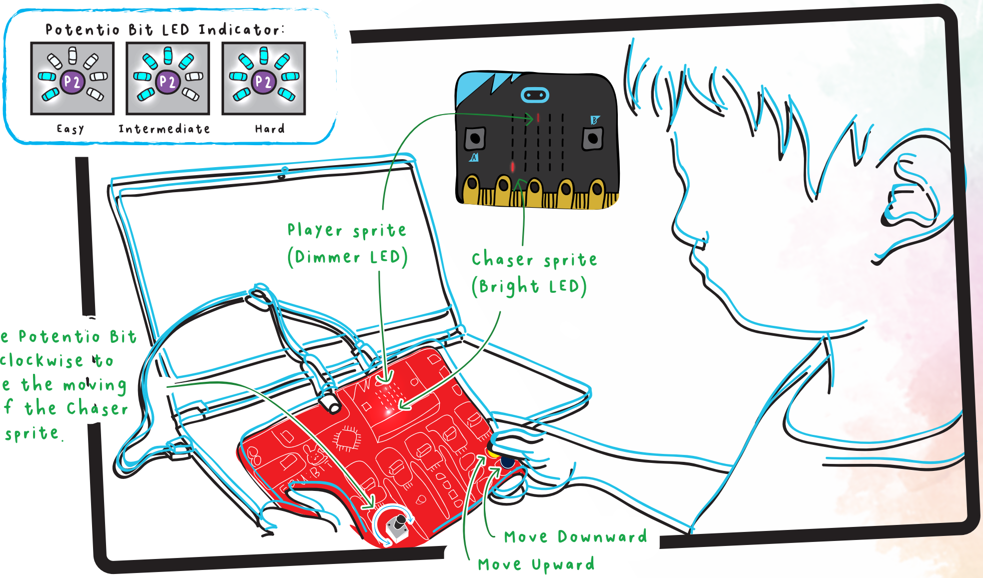
ELSE call "Game" function and set variable "speed" to 750ms.

```
forever
    if potentiometer value > 800 then
        call Game 250
    else if potentiometer value > 400 then
        call Game 500
    else
        call Game 750
```

Step 18 Upload the completed code to your EDU:BIT and have fun playing Tag, You're It! with your friends.

Let's Play

Tag, You're It!


Potentio Bit LED Indicator:
P2
P2
P2
Easy
Intermediate
Hard
Player sprite
(Dimmer LED)
Chaser sprite
(Bright LED)
Turn the Potentio Bit knob clockwise to increase the moving speed of the Chaser sprite.
Move Downward
Move Upward

HOW TO PLAY:

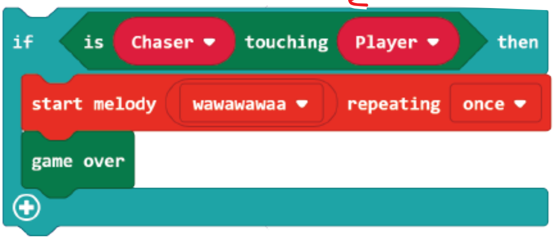
- When powered up, the Chaser sprite will keep moving in random direction.
- Move the Player sprite up or down to avoid the Chaser sprite.
  Press the yellow button (Button A) to move upward and the blue button (Button B) to move downward.
- Game Over if the Player is "touched" by the Chaser, or after the 30-second time limit is up.
- Each time the Chaser sprite "touches" the edge, you will score 1 point. Player with the highest score is the Winner! Have fun~

TIPS!

#1 To get a higher score within the 30-second time limit, you can increase the speed of the Chaser sprite so that it will "touch" the edge more often.
#2 After game over, you can press Buttons A+B simultaneously to start a new game. This is a built-in function of [Game] blocks.

BREAK THE CODE

In programming, we use **conditional if statements** to make decisions. In MakeCode, we use **[if-then]** or **[if-then-else]** conditional blocks from **[Logic]** category to form the condition. The program checks the condition statement and if it is TRUE, it executes the code in the conditional block. Else, if FALSE, it moves on to the next block of code.

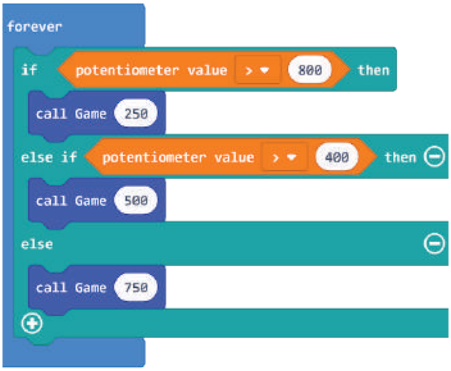


```
if  is Chaser ▾ touching Player ▾  then
    start melody  wawawawaa ▾  repeating once ▾
    game over
⊕
```

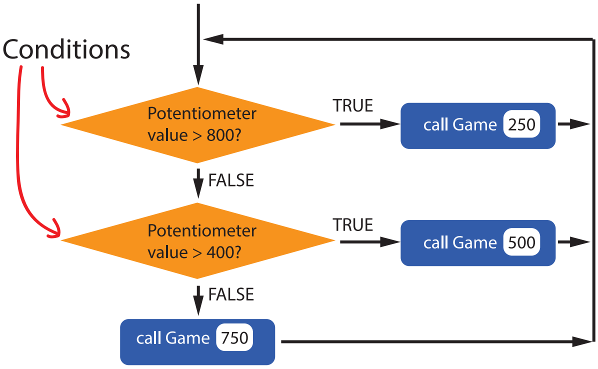
if this condition is met
(i.e. Chaser sprite touches Player sprite),
then do this
(play melody wawawawaa, and then display 'game over' animation).

When we have multiple conditions, the program will evaluate the conditions in sequence, from top to bottom, and execute the corresponding code of the first condition that returns TRUE. Thus, the higher a condition sits, the higher priority it holds compared to those that are below.

For example, this code in the game determines the moving speed of the Chaser sprite by comparing the value of the potentiometer against preset thresholds.



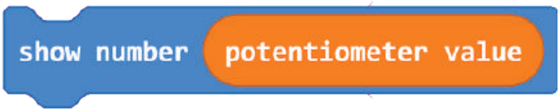
```
forever
    if  potentiometer value > ▾ 800  then
        call Game 250
    else if  potentiometer value > ▾ 400  then ⊖
        call Game 500
    else ⊖
        call Game 750
    ⊕
```



if potentiometer value > 800, call function Game (with variable speed set to 250 ms),
else if potentiometer value > 400, then call function Game (speed = 500 ms),
else call function Game (speed = 750 ms).

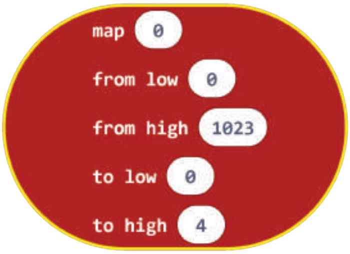
EXPLORE MORE BLOCKS

#1 Use a [Basic] : [show number] block with [Potentio Bit] : [potentiometer value] block to read and display the current potentiometer value.



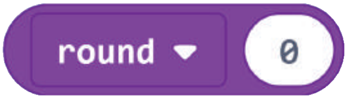
```
show number  potentiometer value
```

#2 The potentiometer returns a value between 0 to 1023. You can use the [map _ from low _ from high _ to low _ to high _] block from the **[Advanced]: [Pins]** category to map the reading to another range that is more meaningful.



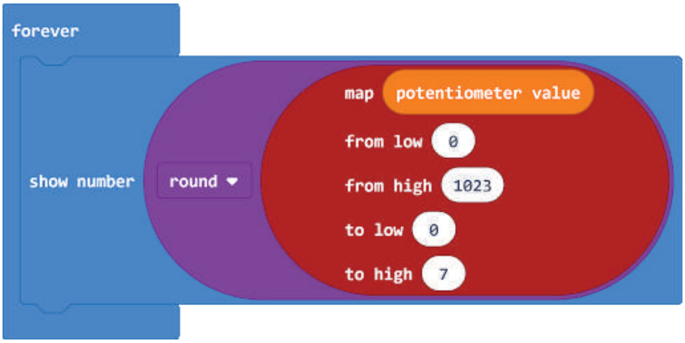
```
map 0
from low 0
from high 1023
to low 0
to high 4
```

#3 The mapping block will return a number with decimal points (e.g. 1.68, 3.998). To round up the number, use the [round _] block from [Math] category.

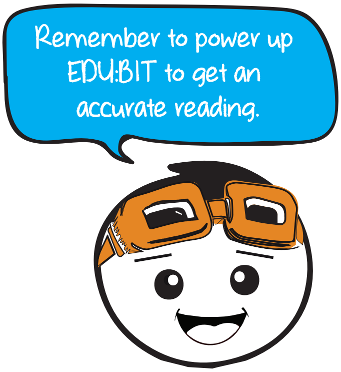


```
round ▾  0
```

Here's a sample code to map the Potentio Bit reading to a range of 0 to 7. The value is rounded up and displayed on the LED matrix.



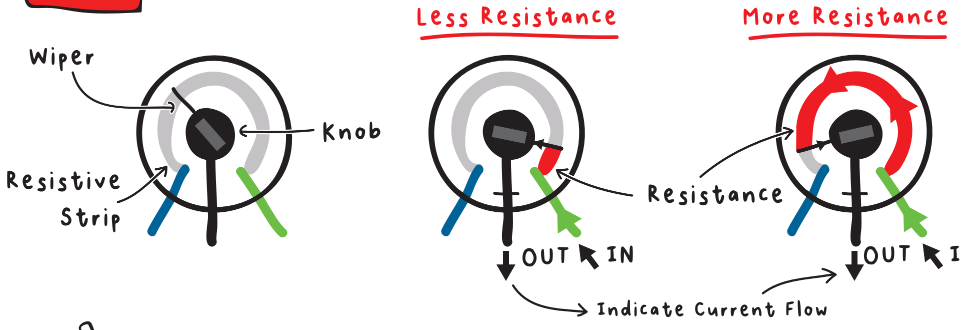
```
forever
    show number  round ▾  map potentiometer value
                          from low 0
                          from high 1023
                          to low 0
                          to high 7
```

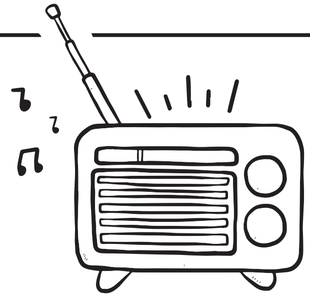


FUN FACT!

Potentiometers, also referred to as pots, are variable resistors with resistance that can be easily adjusted using a knob or slider.

If you have a 10,000Ω potentiometer, you can get a resistance value between 0Ω to 10,000Ω by changing the wiper position.

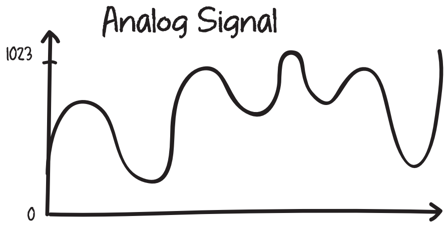


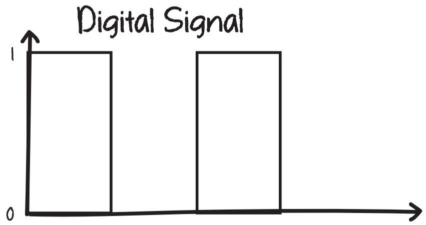


Here are some common applications:

- Audio volume of a speaker
- Frequency control of a radio
- Water heater temperature control

The potentiometer on EDU:BIT is a type of analog input device. It measures electric potential and converts the voltage measured (between 0V to 3.3V) into an integer value between 0 and 1023.





APPLICATION CHALLENGE

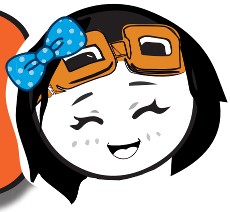
<table>
<tr><td colspan="2">Program EDU:BIT to be a timer. Use Potentio Bit to adjust the duration (between 0 to 60 seconds), Button A to activate the timer and Button B to reset.</td></tr>
<tr><td>On Start</td><td>Set Mode to 0</td></tr>
<tr><td>On Button A pressed (Yellow button)</td><td>Set Mode to 1
Set Start Time to running time
Show a smiley face icon</td></tr>
<tr><td>On Button B pressed (Blue button)</td><td>Set Mode to 0</td></tr>
<tr><td>Forever</td><td>Always check Mode
• IF Mode=0, then set Duration to rounded value of potentiometer reading mapped to low 0 and high 60 and display Duration on the LED matrix.
• ELSE IF Mode=1, check whether (running time - Start Time) > (Duration x 1000). If TRUE, play melody wawawawaa and then set Mode to 2.
• ELSE, show sad face icon.</td></tr>
</table>

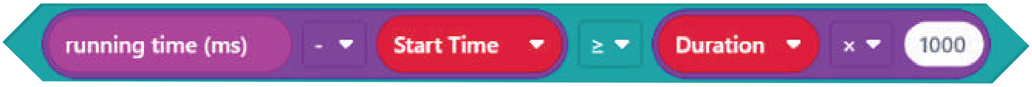
Here are some tips for you:

Tip #1 You need to create three variables: Mode, Start Time & Duration.

Tip#2 Running time (ms) block is from [Input] category.

Tip#3 Use this conditional statement to check whether time's up.





running time (ms) - Start Time ≥ Duration x 1000

CHAPTER 7

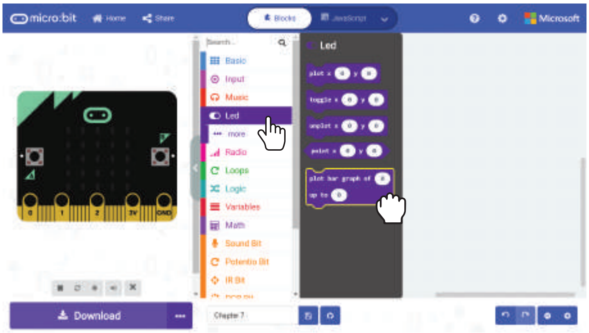
Let's Hear the Applause!

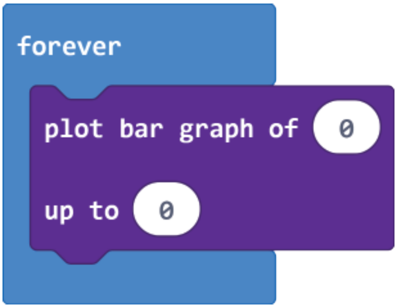
Sound Bit

link.cytron.io/edubit-chapter-7

LET'S CODE!

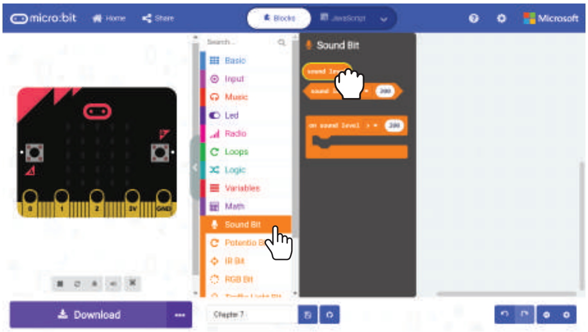
Step 1 Create a new project in your MakeCode Editor and add EDU:BIT extension (you can refer to page 40). Click **[Led]** category and select **[plot bar graph of _ up to _]** block. Snap the block to the **[forever]** slot.

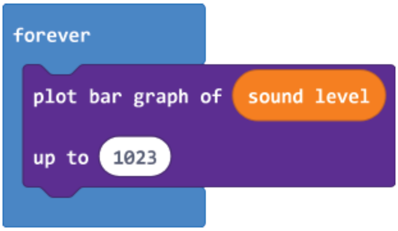




```
forever
    plot bar graph of 0
    up to 0
```

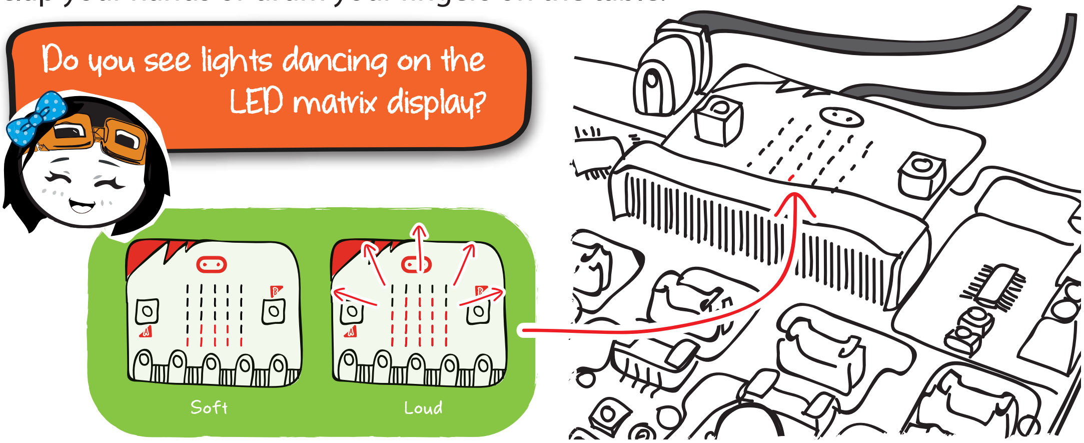
Step 2 Click **[Sound Bit]** category and select **[sound level]** block. Snap the block to the **[plot bar graph of _ up to _]** block and change the second 0 value to **1023.**





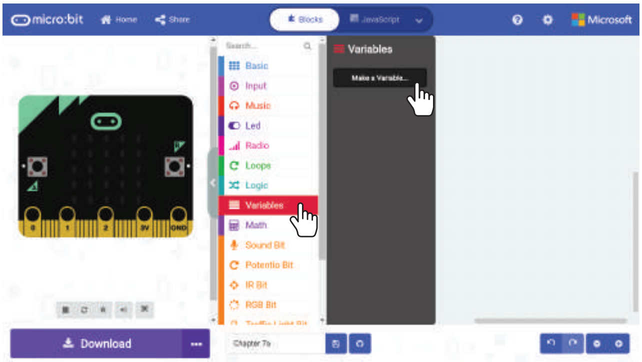
```
forever
    plot bar graph of sound level
    up to 1023
```

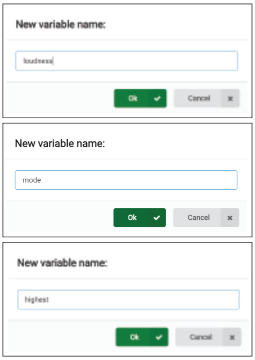
Step 3 Flash the code to your EDU:BIT. Observe the LED matrix display as you clap your hands or drum your fingers on the table.



Now let's turn EDU:BIT into an applause-o-meter.

Step 4 Start a new project and add EDU:BIT extension. Click **[Variables]** category and then click [**Make a Variable**]. Type '**mode**' in the pop up window and then click OK. Create two more variables named '**loudness**' and '**highest**'.





New variable name:

loudness

Ok ✓ Cancel ×

New variable name:

mode

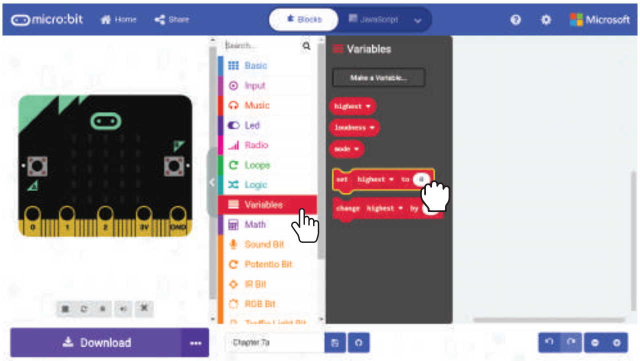
Ok ✓ Cancel ×

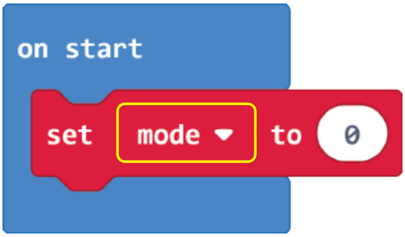
New variable name:

highest

Ok ✓ Cancel ×

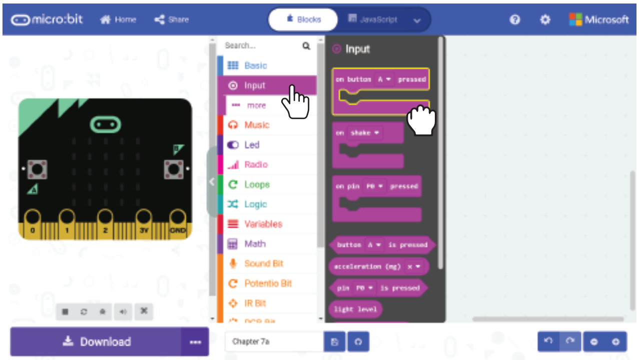
Step 5 Get **[set _ to_]** block from **[Variables]** category. Snap the blocks into the **[on start]** slot. Set the variable to 'mode'.

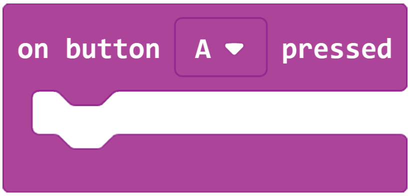




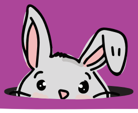
```
on start
    set mode ▾ to 0
```

Step 6 Click **[Input]** category and then select **[on button _ pressed]** block.

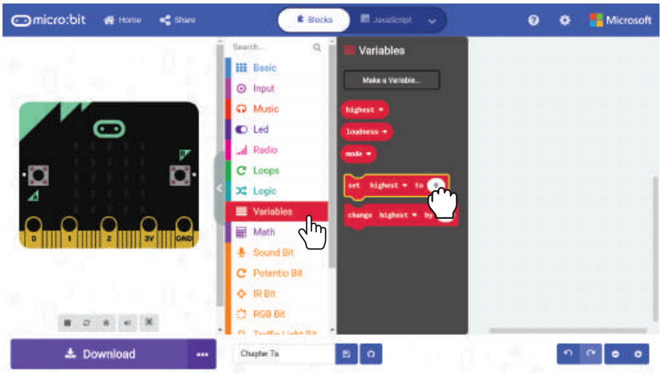


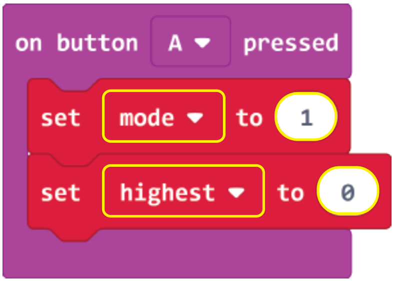


```
on button A ▾ pressed
```



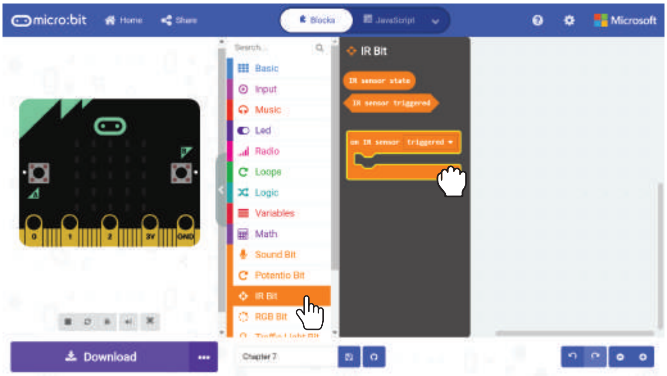
Step 7 Get two **[set _ to_]** blocks from **[Variables]** category and snap the blocks to the **[on button A pressed]** slot. Set the first variable to '**mode**' and the value to **1**, and the second variable to '**highest**' with the default value 0.

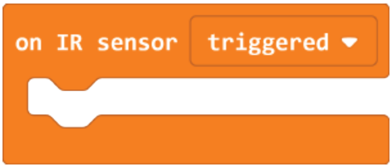




```
on button A ▾ pressed
    set mode ▾ to 1
    set highest ▾ to 0
```

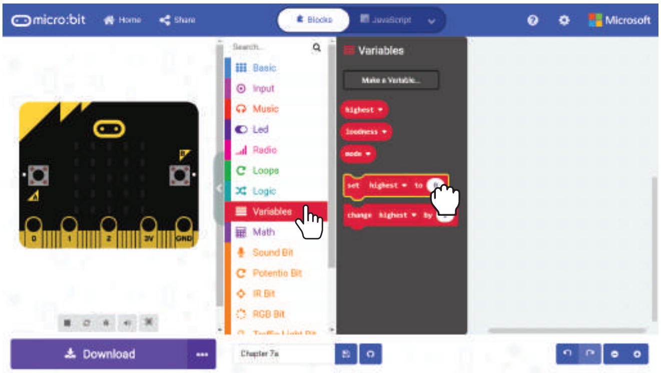
Step 8 Click **[IR Bit]** category and then select **[on IR sensor triggered]** block.

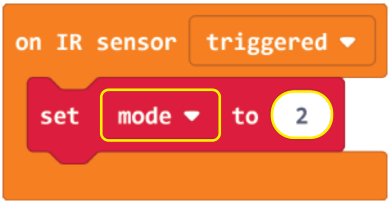




```
on IR sensor triggered ▾
```

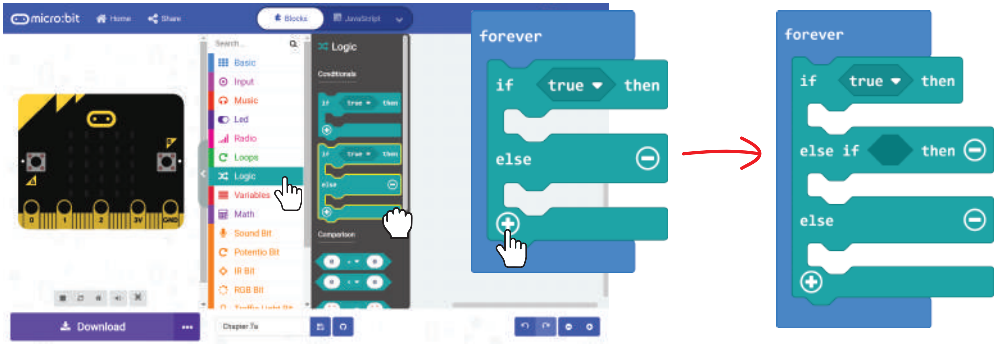
Step 9 Click **[Variables]** category and then select **[set _ to_]** block. Snap the block to the **[on IR sensor triggered]** slot and change the variable to **'mode'** and value to **2**.





```
on IR sensor triggered ▾
    set mode ▾ to 2
```

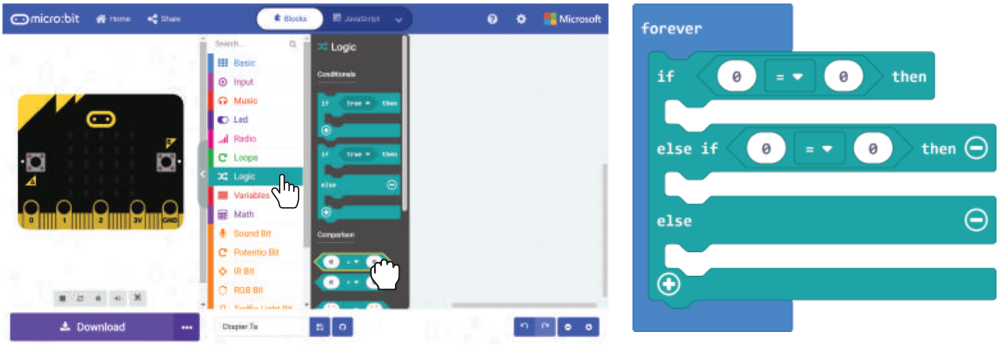
Step 10 Click **[Logic]** category and then select **[if-then-else]** block. Snap the block to the **[forever]** slot. Click on the plus icon to add another else-if condition to the block.



```
forever
    if true then
    else
```

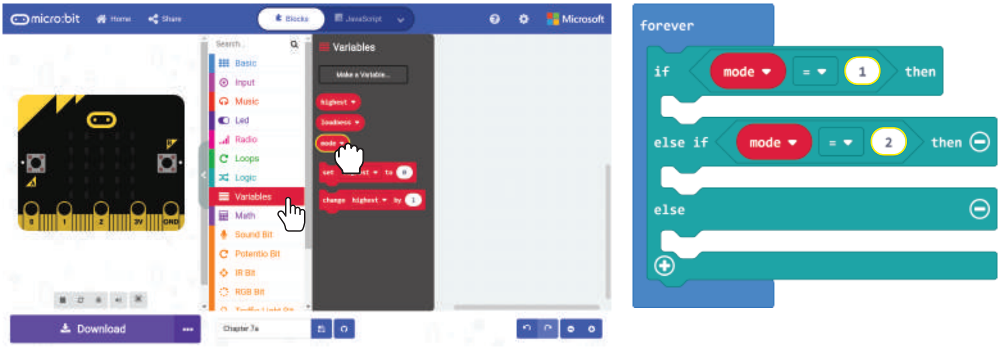
```
forever
    if true then
    else if then
    else
```

Step 11 Click **[Logic]** category and then select **[_ = _]** comparison block. Duplicate the block and snap the blocks to the condition slots of the **[if-then -else]** block.



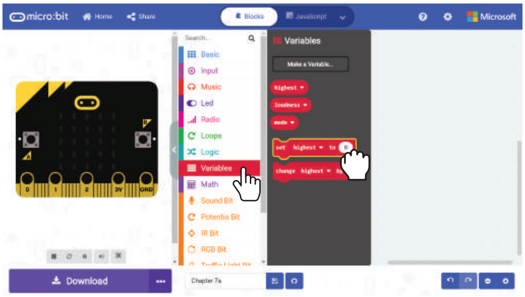
```
forever
    if 0 = 0 then
    else if 0 = 0 then
    else
```

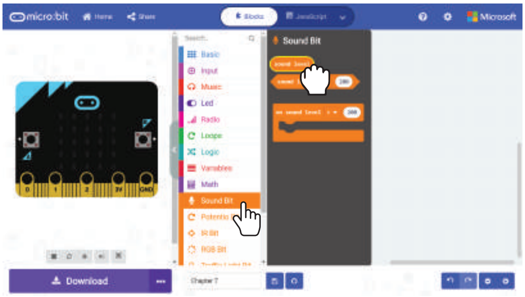
Step 12 Get **[mode]** from the **[Variable]** category and snap it to the left slot of the comparison blocks. Set the other slots to **1** and **2** respectively.

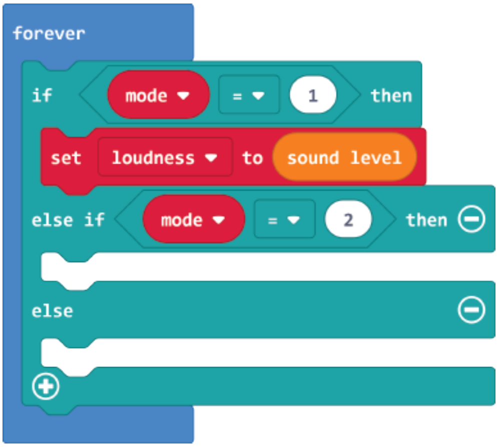


```
forever
    if mode = 1 then
    else if mode = 2 then
    else
```

Step 13 Get [set _ to_] block from [Variables] category and snap the block to the first [if-then-else] slot. Set the variable to '**loudness**' and snap [sound level] block from [Sound Bit] category to the value slot.

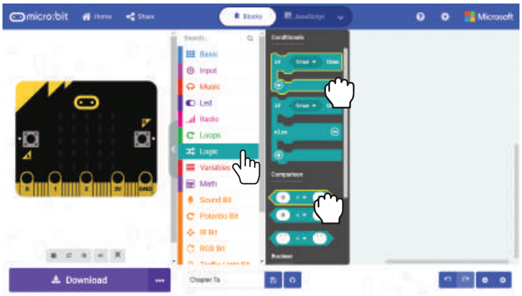


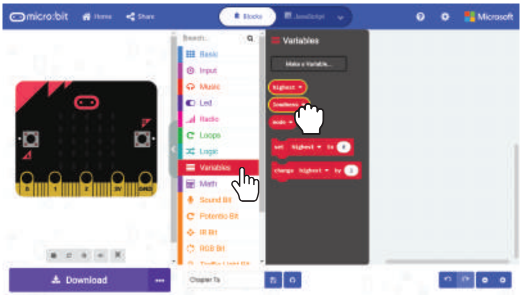


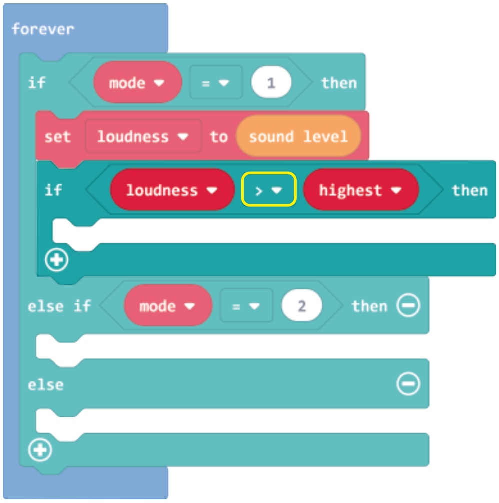


```
forever
    if mode = 1 then
        set loudness to sound level
    else if mode = 2 then
    else
```

Step 14 Click [Logic] category and then select [if-then] block and [_ = _] comparison block. Change the symbol = to **>**. Snap [loudness] and [highest] from the [Variables] category to the slots of the comparison block.

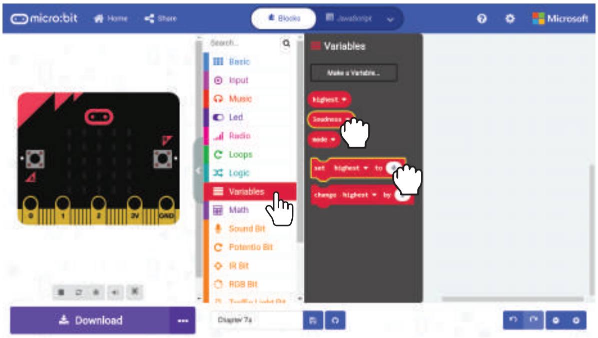


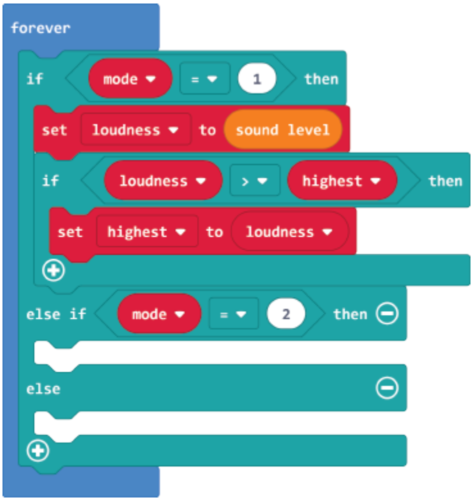




```
forever
    if mode = 1 then
        set loudness to sound level
        if loudness > highest then
    else if mode = 2 then
    else
```

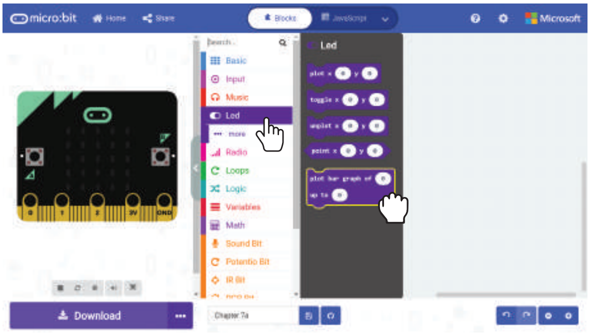
Step 15 Click **[Variables]** category and then select **[set _ to_]** block. Snap the block to the **[if-then]** slot and **[loudness]** from **[Variables]** category to the value slot.

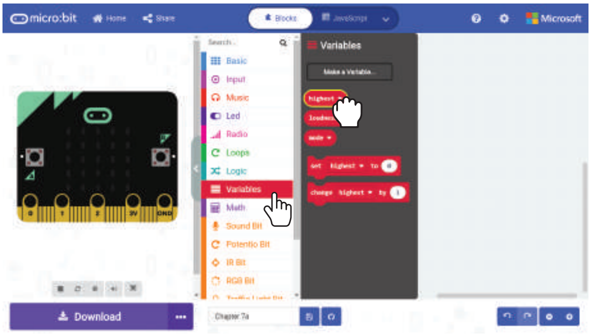


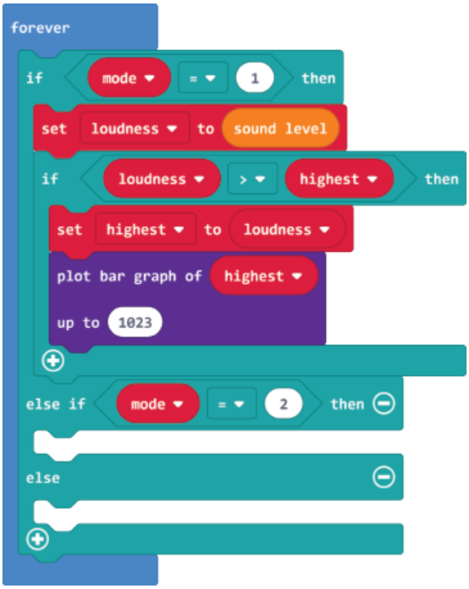


```
forever
  if mode = 1 then
    set loudness to sound level
    if loudness > highest then
      set highest to loudness
    ⊕
  else if mode = 2 then ⊖

  else ⊖

  ⊕
```

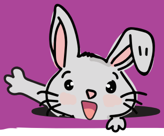
Step 16 Click **[Led]** category and select **[plot bar graph of _ up to _]** block. Click **[Variables]** category and select **[highest]** block. Snap the block to the **[plot bar graph of _ up to _]** block and change the value to **1023.**



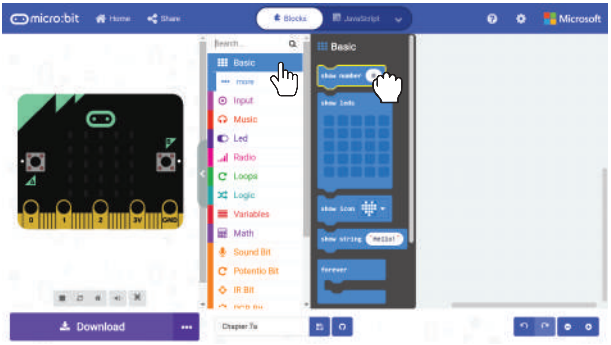


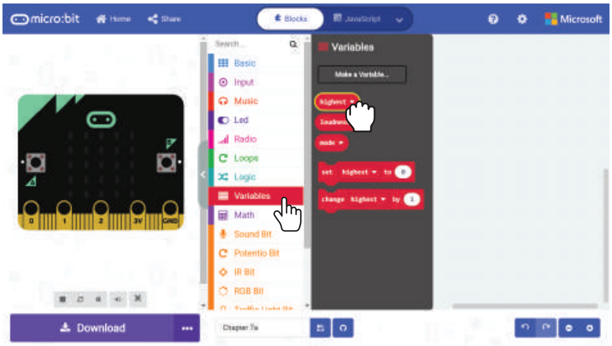


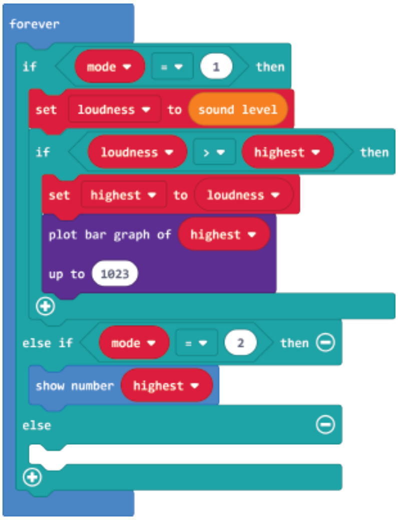
```
forever
  if mode = 1 then
    set loudness to sound level
    if loudness > highest then
      set highest to loudness
      plot bar graph of highest
      up to 1023
    ⊕
  else if mode = 2 then ⊖

  else ⊖

  ⊕
```



Step 17 Click [Basic] category and select [show number] block. Snap it to the second slot of the [if-then-else] block. Get [highest] block from [Variable] category and snap it to the value slot of [show number] block.

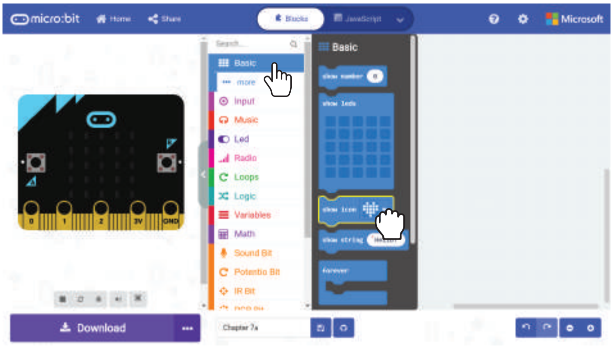


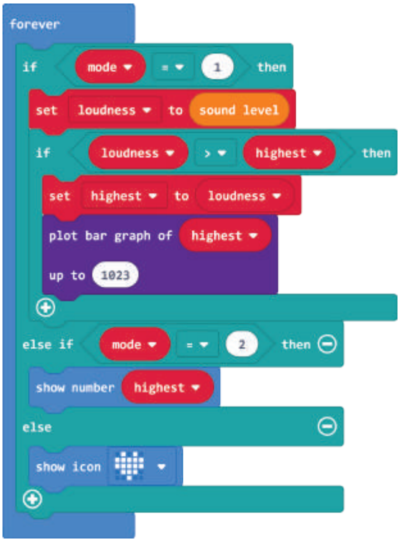




```
forever
    if mode = 1 then
        set loudness to sound level
        if loudness > highest then
            set highest to loudness
            plot bar graph of highest
            up to 1023
    else if mode = 2 then
        show number highest
    else
```

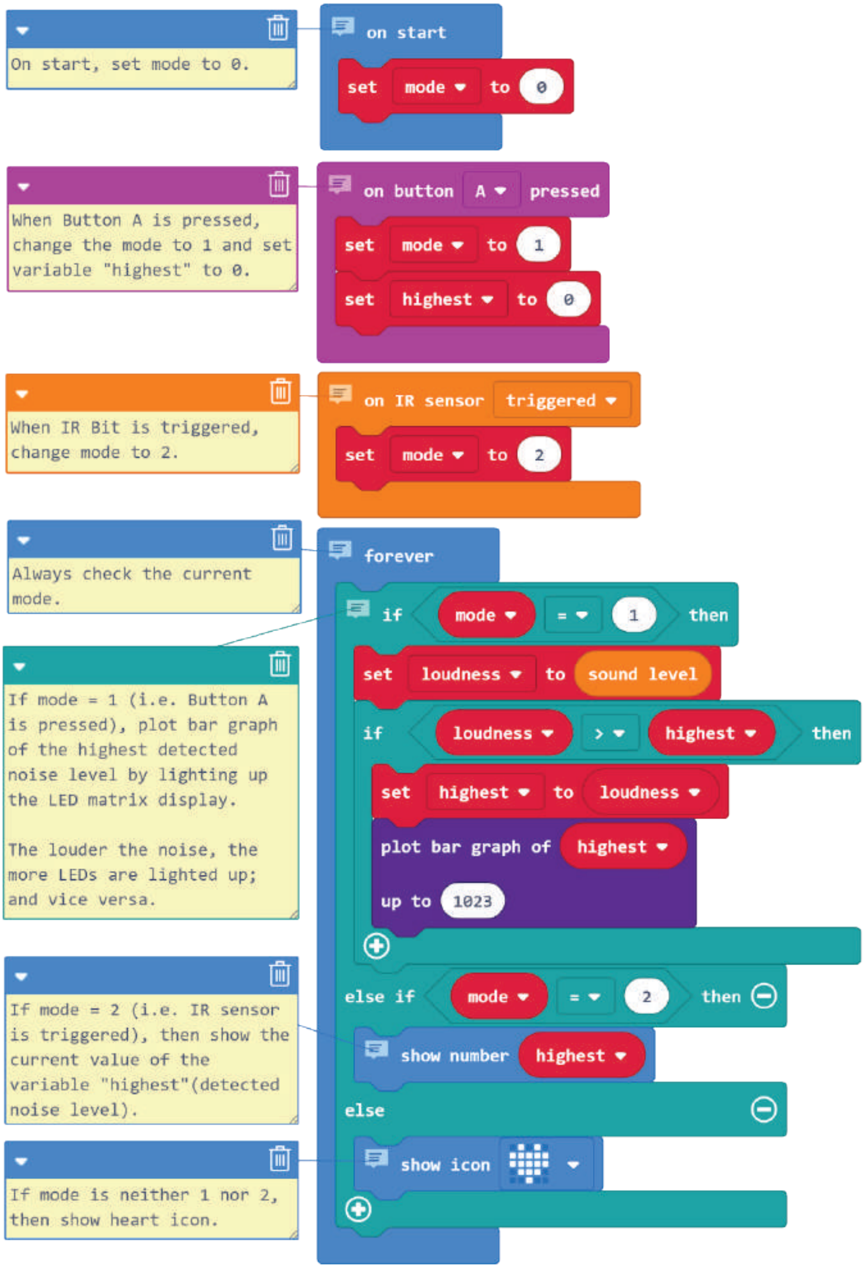
Step 18 Click [Basic] category and select [show icon] block. Snap it to the final slot of the [if-then-else] block.





```
forever
    if mode = 1 then
        set loudness to sound level
        if loudness > highest then
            set highest to loudness
            plot bar graph of highest
            up to 1023
    else if mode = 2 then
        show number highest
    else
        show icon
```

Here's the full code:



On start, set mode to 0.

```
on start
    set mode to 0
```

When Button A is pressed, change the mode to 1 and set variable "highest" to 0.

```
on button A pressed
    set mode to 1
    set highest to 0
```

When IR Bit is triggered, change mode to 2.

```
on IR sensor triggered
    set mode to 2
```

Always check the current mode.

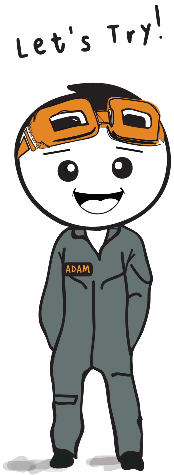
If mode = 1 (i.e. Button A is pressed), plot bar graph of the highest detected noise level by lighting up the LED matrix display.

The louder the noise, the more LEDs are lighted up; and vice versa.

If mode = 2 (i.e. IR sensor is triggered), then show the current value of the variable "highest"(detected noise level).

If mode is neither 1 nor 2, then show heart icon.

```
forever
    if mode = 1 then
        set loudness to sound level
        if loudness > highest then
            set highest to loudness
        plot bar graph of highest
        up to 1023
    else if mode = 2 then
        show number highest
    else
        show icon
```



Step 19 Flash the code to your EDU:BIT and you have an applause-o-meter ready for your Talent Time Show.

Let's Play

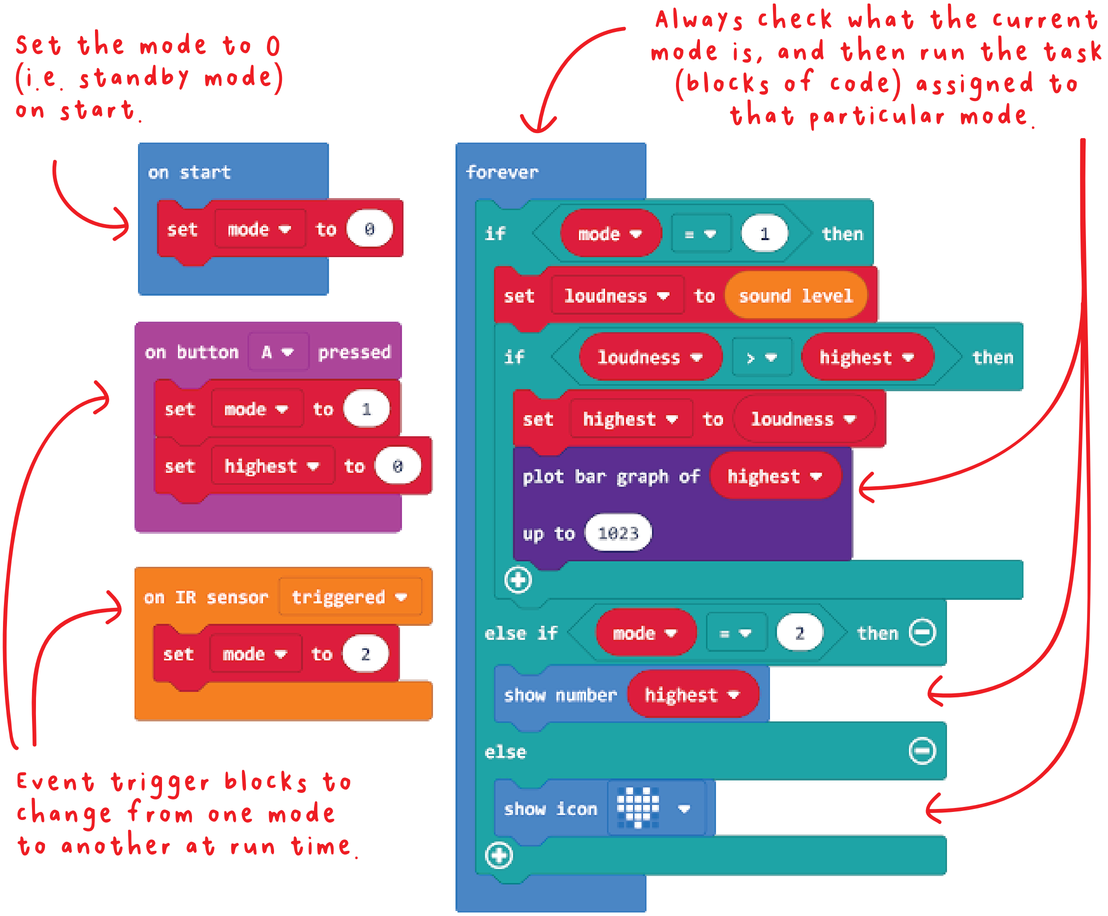
Let's Hear the Applause!

HOW TO PLAY:

- "Contestants" are given time to prepare a short performance, either individually, in pairs or as a team. You can choose to sing, or dance, or even tell a joke.
- When everyone's ready, take turns to showcase your talent. After each performance, the "audience" response by clapping their hands - the more they enjoy the show, the louder they clap.
- Once the applause dies down, trigger IR Bit to scroll the score (highest sound level recorded).
- Remember to press Button A to reset the score before the next performance.
- The Winner is the individual, pair or team that receives the loudest applause. Have fun!

BREAK THE CODE

When we have multiple tasks in one program, we can use event triggers to switch from one task to another. For the program to run smoothly, we use a forever loop to constantly check the current mode and then execute the corresponding block(s) of code.

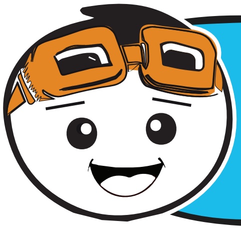


Set the mode to 0 (i.e. standby mode) on start.

Always check what the current mode is, and then run the task (blocks of code) assigned to that particular mode.

```
on start
    set mode to 0

on button A pressed
    set mode to 1
    set highest to 0

on IR sensor triggered
    set mode to 2

forever
    if mode = 1 then
        set loudness to sound level
        if loudness > highest then
            set highest to loudness
        plot bar graph of highest
        up to 1023
    else if mode = 2 then
        show number highest
    else
        show icon
```

Event trigger blocks to change from one mode to another at run time.

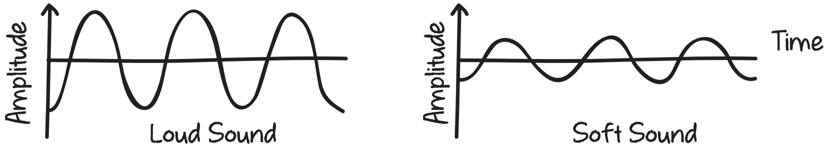


If you have additional modes/tasks to include, you can always add more event trigger blocks, such as [on shake] and [on sound level > _], to your code and click the (+) button on the [if-then-else] block to add more conditions.

FUN FACT!

Sound is produced when an object vibrates, for example a drum when it is struck. The vibration causes air molecules (medium) around it to vibrate and create sound waves.

A **sound sensor** is a module that detects the intensity of sound waves (i.e. the loudness) and converts it into an electrical signal.



In a way, a sound sensor works similarly to our ears which convert vibrations in the air into electrochemical signals which the brain translates into sounds that we know.

Here are some common applications:

- Burglar alarm noise detector
- Sound activated lighting
- Auditory baby monitor

Do you think our sound sensor can detect any noise in the outer space? Why or why not?

APPLICATION CHALLENGE

Program EDU:BIT to function as a classroom noise monitor. Light up the LEDs on Traffic Light Bit to indicate the noise level.

Noise Level	Sound Level Range	Light up Traffic Light Bit
Too noisy; please tune down your volume.	( ) to 1023	Red LED
Slightly noisy; please mind your volume.	( ) to ( )	Yellow LED
Acceptable noise level. Good!	0 to ( )	Green LED

Here are some tips for you...

Tip #1: You will need to determine the threshold values for each noise level beforehand.

Tip #2: For a more stable monitoring, get an average value of the noise level reading at regular intervals.

Too easy? Try this level-up challenge. Modify your code so that the threshold values are relative to the potentiometer value.

CHAPTER 8

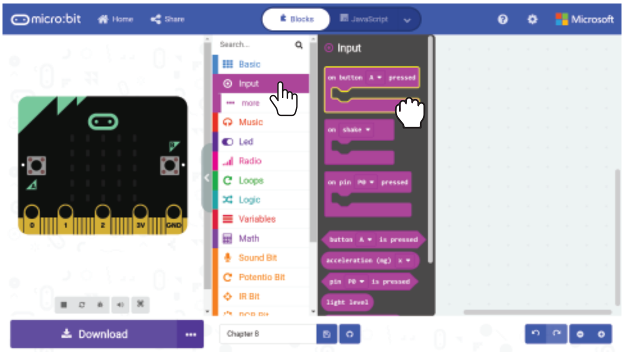
Let's Take a Spin
Round & Round We Go

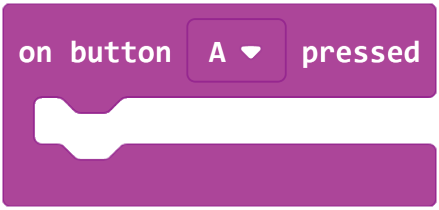
DC Motor

link.cytron.io/edubit-chapter-8

LET'S CODE!

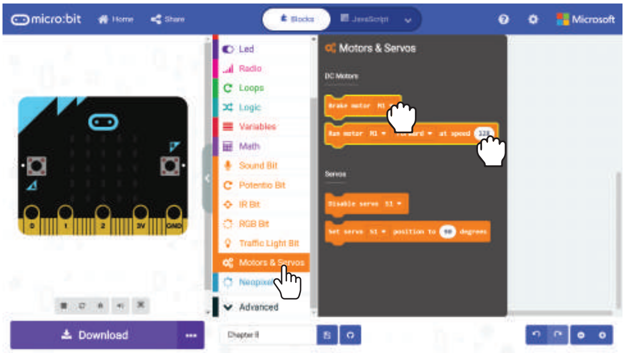
Step 1 In your MakeCode Editor, start a new project and add EDU:BIT extension (you can refer to page 40). Click [Input] category and then select [on button _ pressed] block.

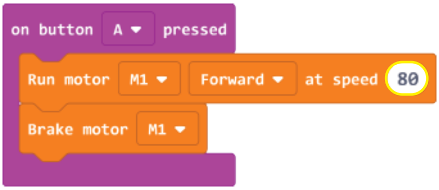




```
on button A ▾ pressed
```

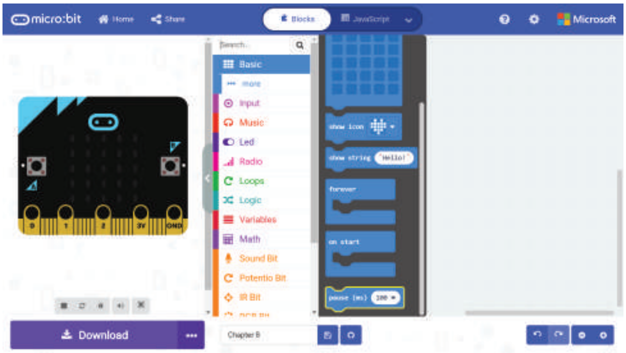
Step 2 Click [Motors & Servos] category and add [Run motor _ _ at speed _] block and [Brake motor _] block to your code. Change the speed value to **80.**

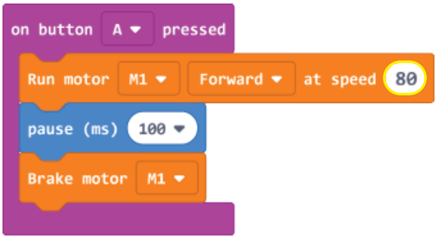




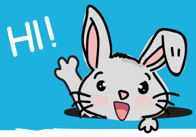
```
on button A ▾ pressed
    Run motor M1 ▾ Forward ▾ at speed 80
    Brake motor M1 ▾
```

Step 3 Click [Basic] category and select [pause _] block. Snap the block in between [Run motor _ _ at speed _] block and [Brake motor _] block.

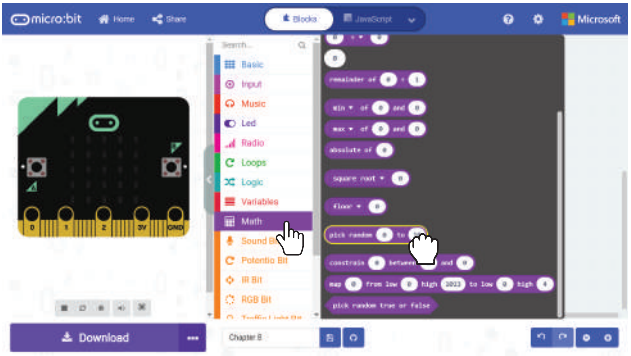


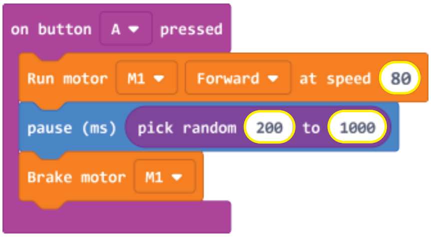


```
on button A ▾ pressed
    Run motor M1 ▾ Forward ▾ at speed 80
    pause (ms) 100 ▾
    Brake motor M1 ▾
```



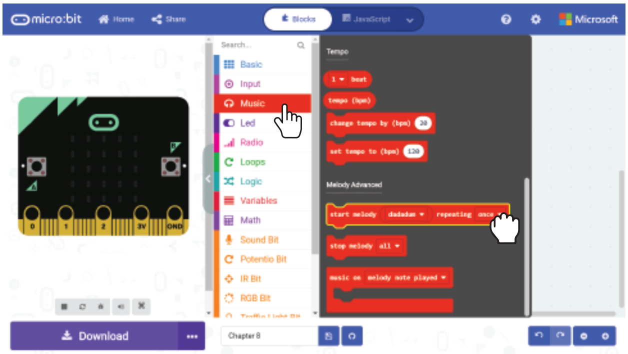
Step 4 Click [**Math**] category and select [**pick random _ to _**] block. Place the block in [**pause _**] block and change the values to **200** and **1000** respectively.

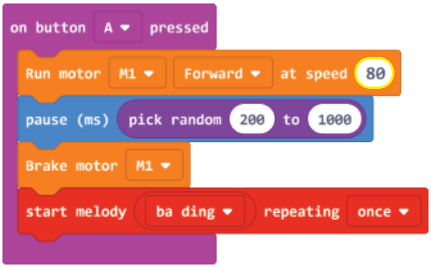




```
on button A ▾ pressed
    Run motor M1 ▾ Forward ▾ at speed 80
    pause (ms) pick random 200 to 1000
    Brake motor M1 ▾
```

Step 5 Click [**Music**] and select [**start melody _ repeating _**] block. Change the melody to "**ba ding**" (or choose any melody that you wish).



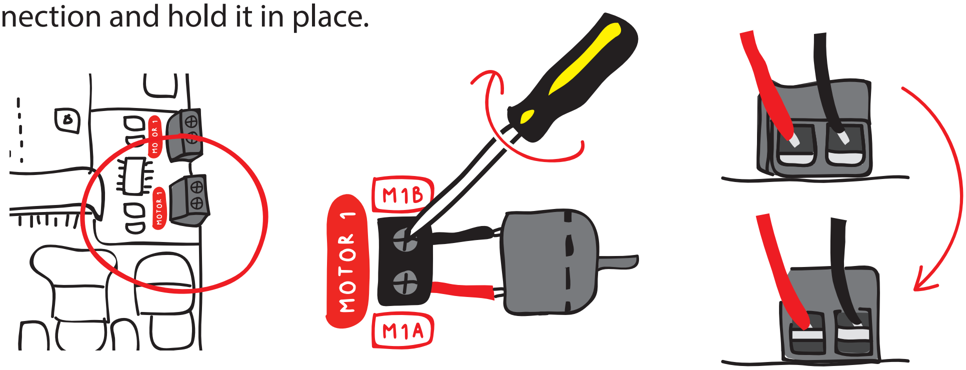


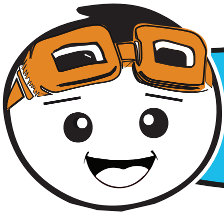
```
on button A ▾ pressed
    Run motor M1 ▾ Forward ▾ at speed 80
    pause (ms) pick random 200 to 1000
    Brake motor M1 ▾
    start melody ba ding ▾ repeating once ▾
```

Step 6 Flash the completed code to your EDU:BIT.

We can use this code for any game that requires a random spinner. Basically, the motor starts spinning when it is triggered, and then stops after a random period.

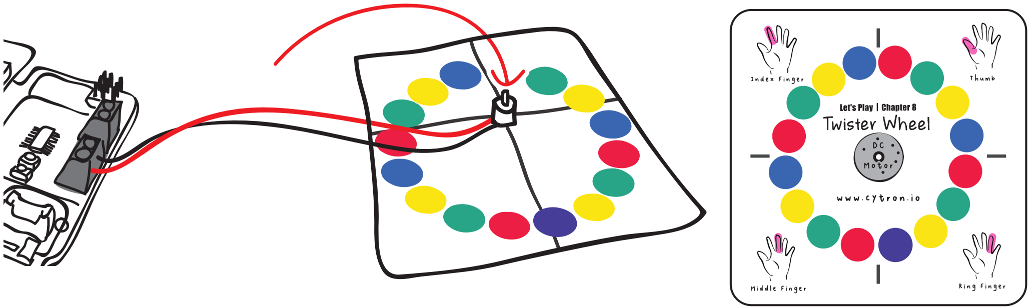
Step 7 Connect the DC motor to MOTOR 1 terminal - (i) insert the exposed wire, and then (ii) tighten the screw using the screwdriver provided to secure the connection and hold it in place.



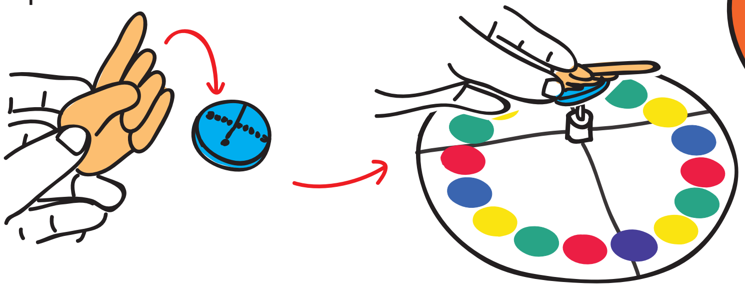


Press the yellow button (Button A) to test. If the motor is not spinning, check that the wire connection at the terminal is secure and EDU:BIT is powered ON.

Step 8 Use extra adhesive, such as double-sided tape or hot glue, to attach the DC motor to the center of the Twister wheel (as indicated).



Step 9 Pop out the pointer and attach it to the plastic disc with some adhesive. Then fix the disc in place on the motor shaft.



Twister wheel, pointer and game map are provided in the box.

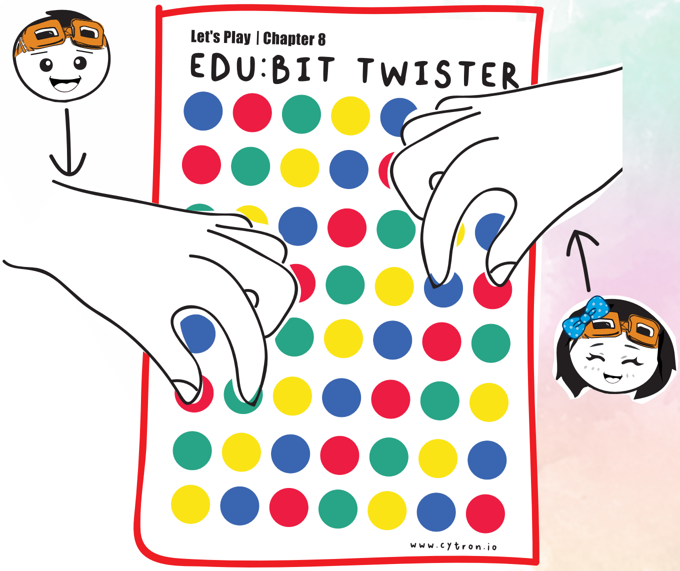
Let's Play

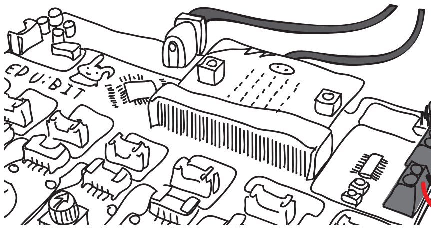
Let's Take a Spin!

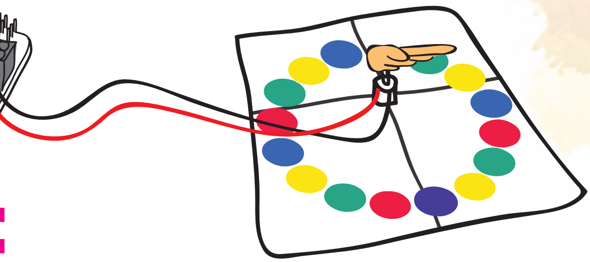
Setting up the Game

- Lay out the game map on the table. Two players sit facing each other. If there are additional players, they can take up the remaining sides (max 4 players)

- The referee sits nearby, with EDU:BIT twister wheel within easy reach.







HOW TO PLAY:

In this game, players take turns to place their fingers on coloured circles on the game map as instructed by the referee.

The referee's role is to press the Yellow Button (or Button A) to spin the pointer and then calls out the finger and colour that the pointer is pointing to, for example: "Index finger; Red".

When it is your turn, you must listen to the referee's instruction, and place the called-out finger on a circle with the correct colour. If that finger is already on a circle of the called-out colour, you must try to move it to another circle of the same colour.

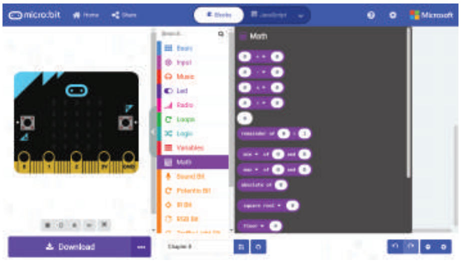
If you fail to complete a move successfully, you are out from the game.

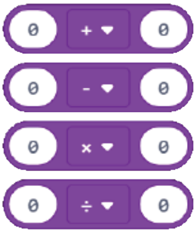
The last player left in the game is the WINNER!

EXPLORE MORE BLOCKS

You can use blocks in [Math] category to perform arithmetic operations on your variables.

#1 Use the following blocks to add, subtract, multiply or divide.

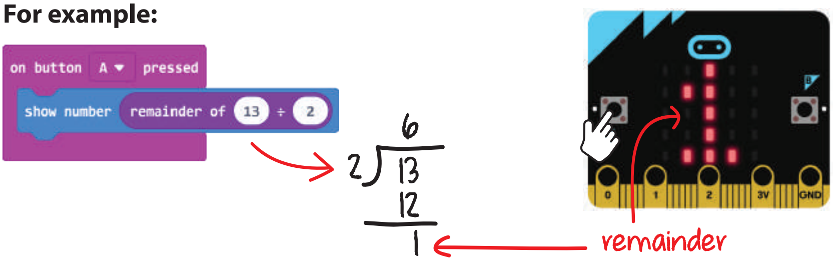




```
0 + ▾ 0
0 - ▾ 0
0 × ▾ 0
0 ÷ ▾ 0
```

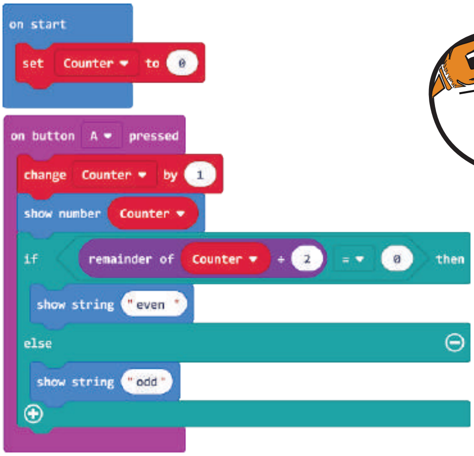
#2 You can use the [remainder of __ ÷ __] block to find out how much is left over if one number does not divide into the other number evenly.

For example:

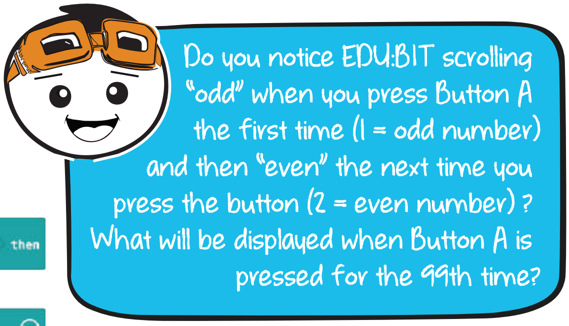


```
on button A ▾ pressed
    show number remainder of 13 ÷ 2
```

#3 You can also use the [remainder of __ ÷ __] block to decide whether a number is odd or even. Simply divide by 2. If the remainder is "1" then it is "odd"; if the remainder is "0" then it is "even". Let's try!



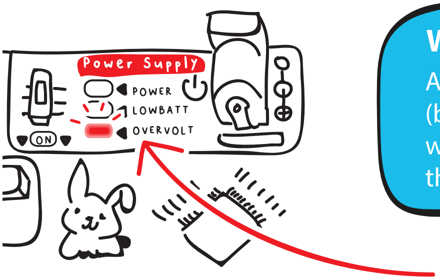
```
on start
    set Counter ▾ to 0

on button A ▾ pressed
    change Counter ▾ by 1
    show number Counter ▾
    if remainder of Counter ▾ ÷ 2 = ▾ 0 then
        show string "even"
    else
        show string "odd"
```


Do you notice EDU:BIT scrolling "odd" when you press Button A the first time (1 = odd number) and then "even" the next time you press the button (2 = even number)? What will be displayed when Button A is pressed for the 99th time?

FUN FACT!

A direct current motor, or more commonly known as **DC motor**, is a rotating electrical device that converts electrical energy into mechanical energy.

We need to apply input voltage to make a DC motor spin. We can control the spinning speed by adjusting the input voltage. The higher the input voltage, the faster the motor spins. The recommended voltage for the DC motor in EDU:BIT kit is 3.6V - 6V.

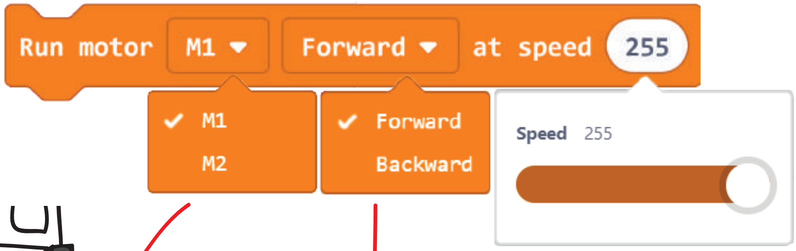


Overvoltage indicator

WARNING!

Applying high voltage to a motor (beyond the recommended voltage) will shorten a motor's life cycle in the long run.

You can easily control the spinning direction and speed of the DC motor using the following programing block.

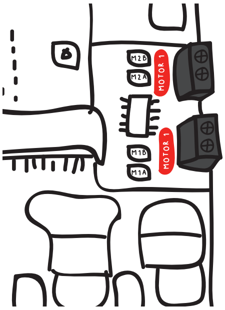


Run motor M1 ▾ Forward ▾ at speed 255

✓ M1
M2

✓ Forward
Backward

Speed 255



Spinning direction.

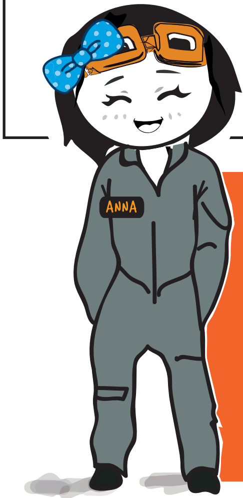
There are two DC motor terminals on EDU:BIT board. Select the correct terminal.

This is a relative value ranging from 0 to 255. The higher the value, the faster the motor spins.

Do you know that EDU:BIT has a built-in motor test circuit? Press the white buttons (labelled as M1A, M1B, M2A and M2B) to check whether your connection is secure and the motor is working fine.

APPLICATION CHALLENGE

Program EDU:BIT to function as a sound-activated fan whose speed is controlled by Potentio Bit.	
On Start	Show a heart icon (or any icon of your choice) Set variable Mode to 0
On sound level > __ (threshold value)	Change variable Mode by 1
Forever	Set variable 'Speed' to map potentiometer value from low 0 high 1023 to low 0 high 255. Always check Mode • IF Mode is an even number, then brake motor M1. • ELSE IF Mode is an odd number, then run motor M1 according to the value of the variable 'Speed' (as mapped from potentiometer reading).



Here are some tips for you…

Tip #1: You will need to determine the sound level trigger (i.e. threshold value) to activate and stop the motor.

Tip #2: You need to create two variables: Mode and Speed.

Tip #3: Attach the fan blades to the motor shaft and then run the program. If you do not feel air blowing when the motor is spinning, you need to change the spinning direction in your code.

CHAPTER 9

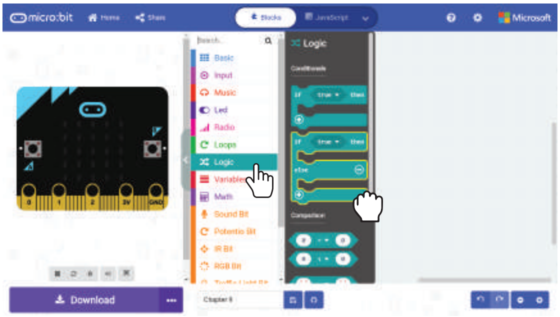
Penalty Shoot-Out... Goal!!!

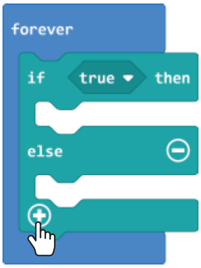
Servo Motor

link.cytron.io/edubit-chapter-9

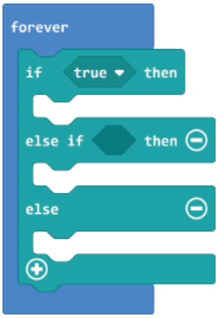
LET'S CODE!

Step 1 In your MakeCode Editor, create a new project and add EDU:BIT extension (you can refer to page 40). Click [**Logic**] category and then select [**if-then-else**] block. Snap the block to the [**forever**] slot. Click on the plus icon to add another else-if condition to the block.



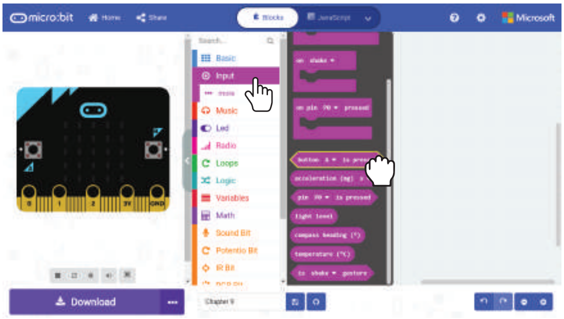


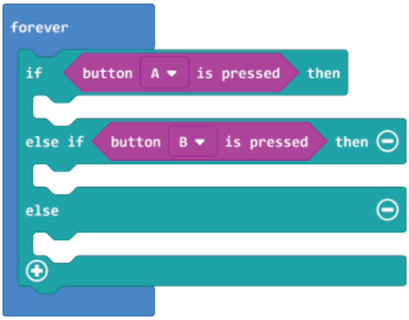
```
forever
  if true then

  else

  ⊕
```



```
forever
  if true then

  else if ⬡ then ⊖

  else ⊖

  ⊕
```

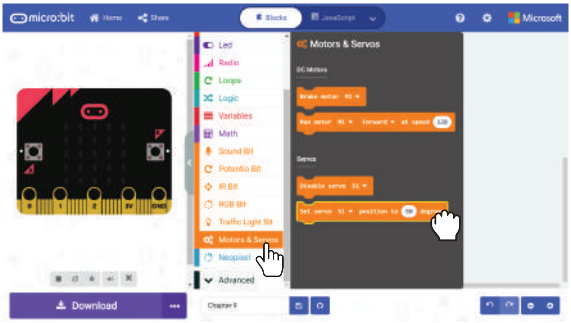
Step 2 Click [**Input**] category and select [**button _ is pressed**] block. Duplicate the block and snap them to the condition slots of the [**if-then-else**] block. Change the second block to '**button B**'.

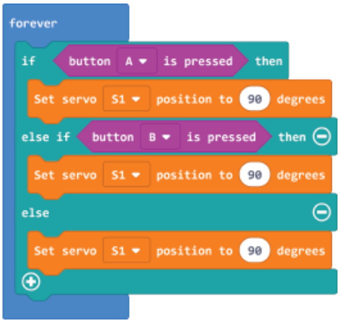




```
forever
  if button A is pressed then

  else if button B is pressed then ⊖

  else ⊖

  ⊕
```

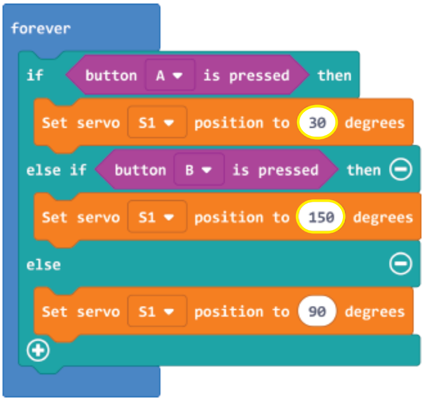
Step 3 Click [**Motors & Servos**] category and select [**Set servo _ position to _ degrees**] block. Duplicate the block and attach to each slot of the [**if-then-else**] block.





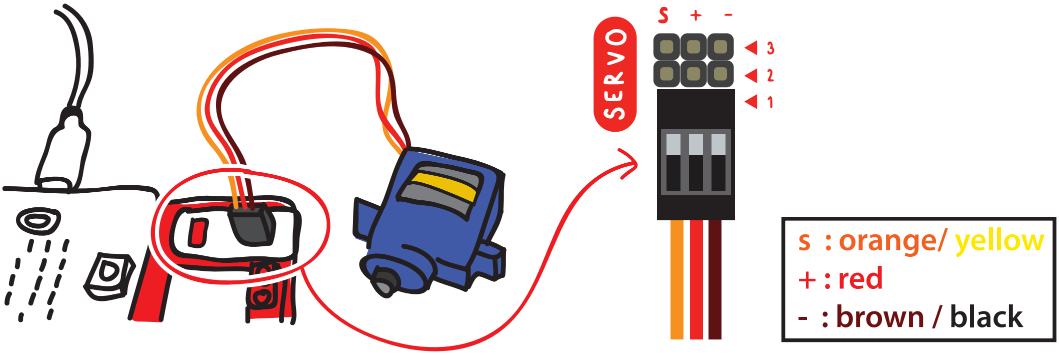
```
forever
  if button A is pressed then
    Set servo S1 position to 90 degrees
  else if button B is pressed then ⊖
    Set servo S1 position to 90 degrees
  else ⊖
    Set servo S1 position to 90 degrees
  ⊕
```

Step 4 Change the values of the first and second blocks to **30** and **150** respectively. Flash the code to your EDU:BIT.

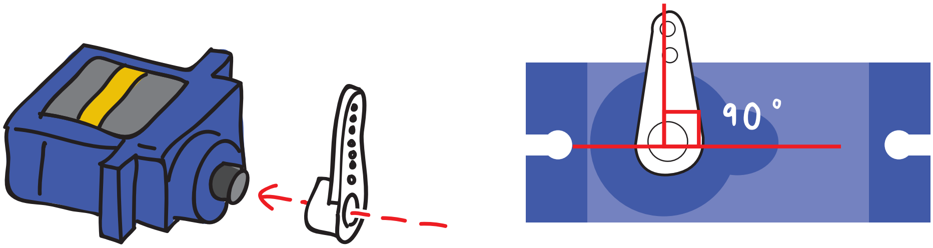


```
forever
  if  button A ▾ is pressed  then
    Set servo S1 ▾ position to 30 degrees
  else if  button B ▾ is pressed  then ⊖
    Set servo S1 ▾ position to 150 degrees
  else ⊖
    Set servo S1 ▾ position to 90 degrees
  ⊕
```

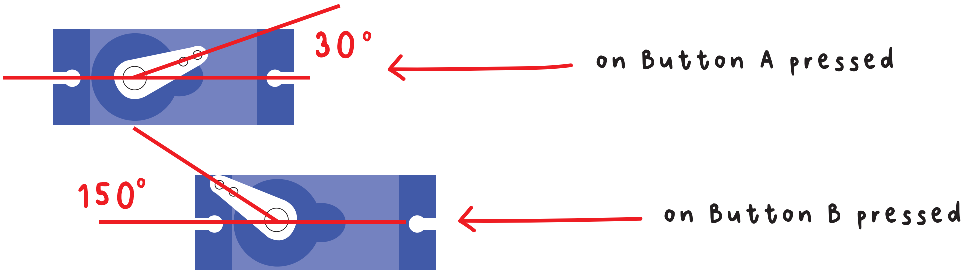
Step 5 Plug in the servo motor cable to Servo Port 1 on EDU:BIT as shown below.



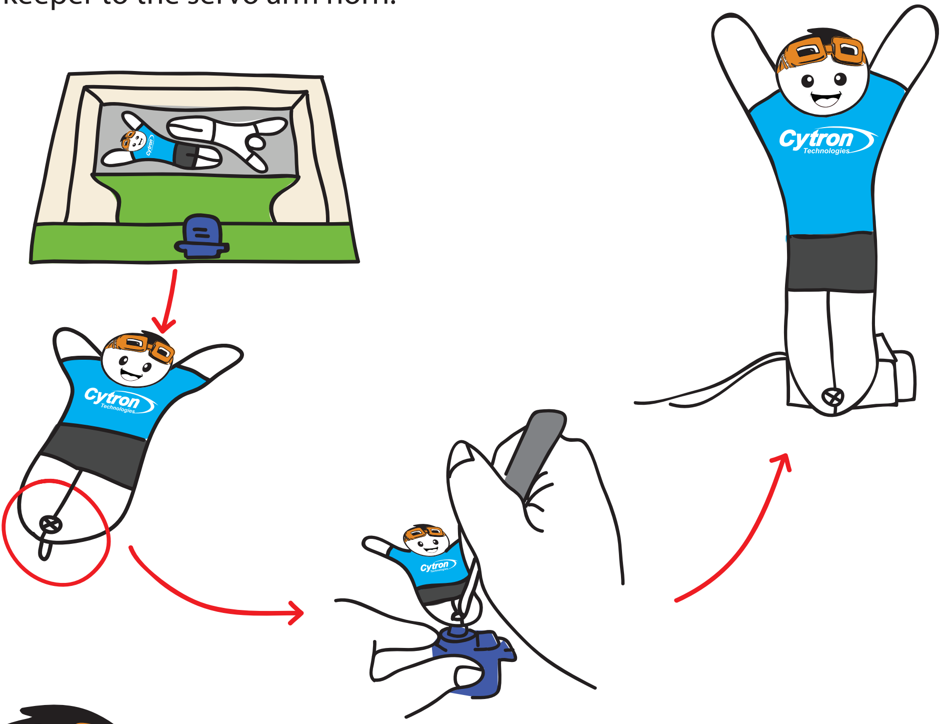
Step 6 Power up EDU:BIT and then attach a servo arm horn to the shaft of the motor servo at 90 degrees, as shown below.



Step 7 Press Button A, and then Button B, to test.

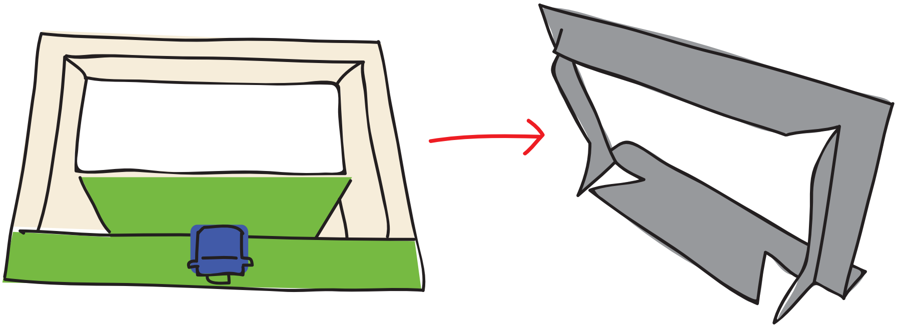


Step 8 Pop out the goalkeeper from the resource card provided. Using the screwdriver and screw provided, fasten the goalkeeper to the servo arm horn. Use extra adhesive, such as double-sided tape or hot glue, to firmly secure the goalkeeper to the servo arm horn.

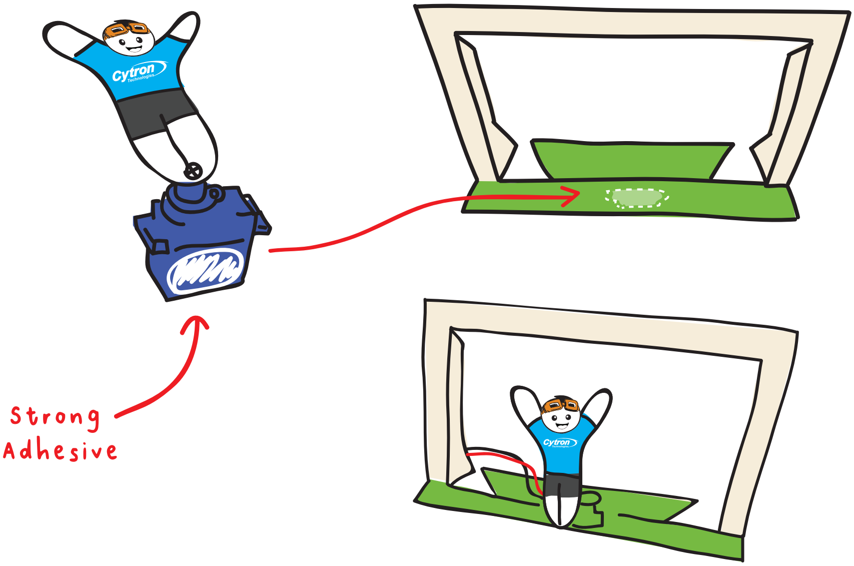


If you like, you can use the other pop-out figure provided and design your own jersey for the goalkeeper.

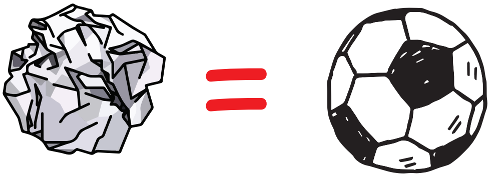
Step 9 Pop out the goal post and set it up as shown below.



Step 10 Use strong adhesive, such as double-sided tape or hot glue, to firmly secure the servo motor in place as indicated.

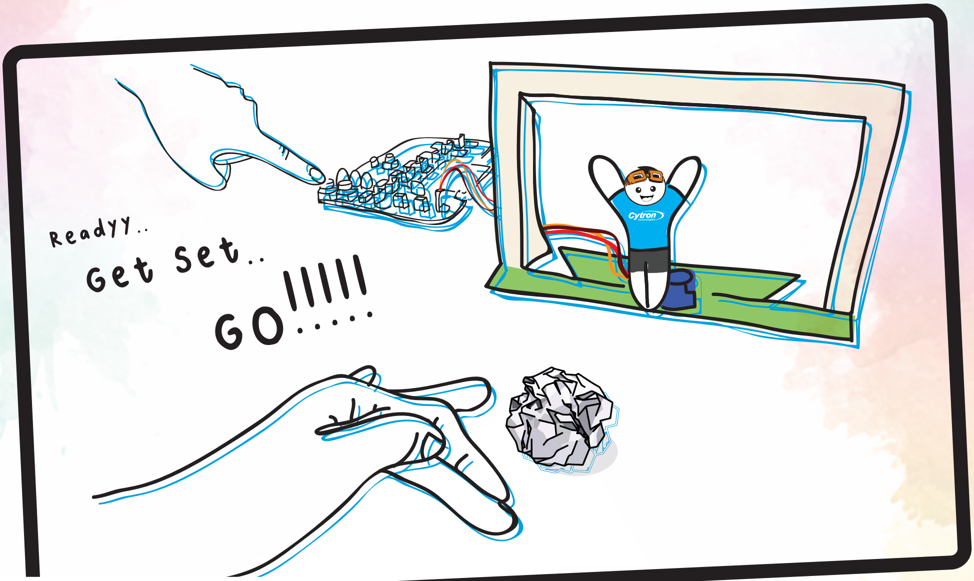


Step 11 Crumple a small piece of paper to become a substitute "football" and we're all set for a fun game of penalty shoot-out. Are you ready?



Let's Play

Penalty Shoot-Out... Goal!!!



HOW TO PLAY:

Set up the goalpost and mark the penalty spot (about 1 metre from the goal, adjust the distance for younger players).

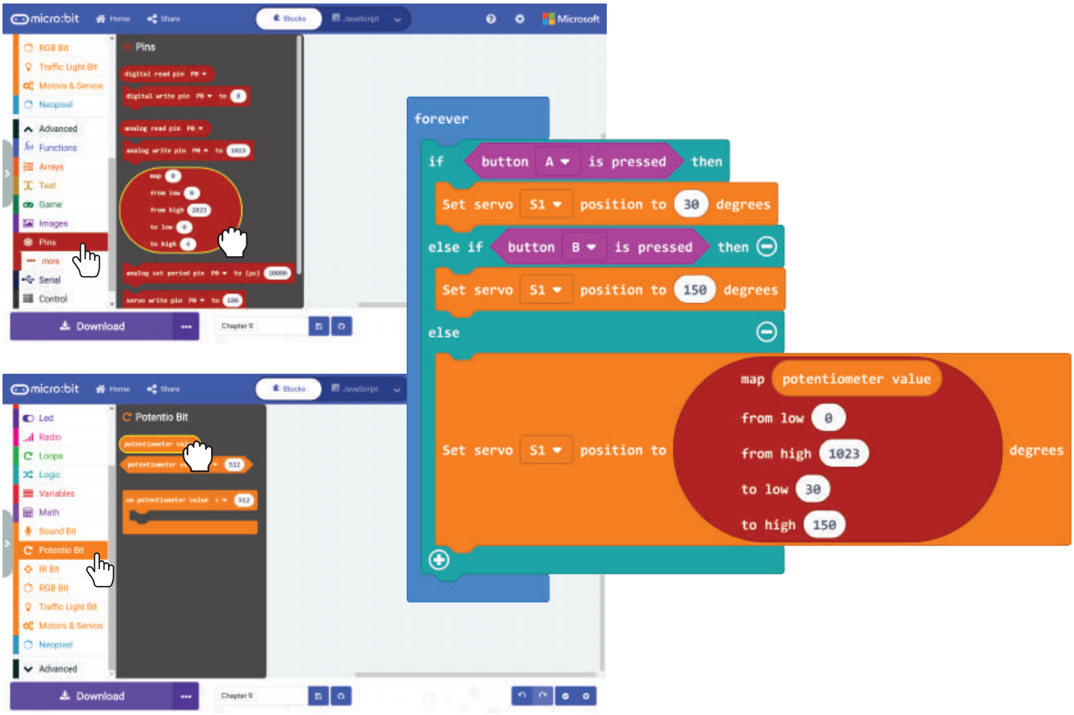
- Players take turns to be the Kicker and the Goalkeeper.
- The Kicker flicks the ball towards the goal.
- The Goalkeeper tries to block the ball by moving to the left by pressing the yellow button (Button A) or to the right by pressing the blue button (Button B).
- In one round, each player gets 5 chances. The player who scores the highest number of goals is the winner.

Do you Know? A penalty shoot-out is played to determine the winning team in a football match when the score is tied at the end of a regular game, and remains a draw even after the extra time. In a penalty shoot-out, each team is given five shots. The player shoots from the penalty mark towards the goal which is guarded by only the opposing team's goalkeeper. Victory goes to the team that scores more goals.

In our code earlier, we make the Goalkeeper swing to the left or right by pressing the push buttons. We can modify the code to control the Goalkeeper's position using Potentio Bit.

Step 12 Click [**Advanced**] category and select **[Pins]** category. Add **[map _ from low _ from high _ to low _ to high _]** block to your code.

Step 13 Click **[Potentio Bit]** category and select **[potentiometer value]** block. Snap the block to the **[map _ from low _ from high _ to low _ to high _]** block and change the last two values to 30 and 150 respectively.



```
forever
    if  button A ▾  is pressed  then
        Set servo  S1 ▾  position to  30  degrees
    else if  button B ▾  is pressed  then
        Set servo  S1 ▾  position to  150  degrees
    else
        Set servo  S1 ▾  position to  map  potentiometer value
                                          from low  0
                                          from high  1023
                                          to low  30
                                          to high  150      degrees
```

Step 14 Flash the code to your EDU:BIT. You can now control the Goalkeeper using Potentio Bit. Have fun!

If you'd like to practise making penalty kicks on your own, you can modify the code to "practice mode" by making the Goalkeeper swing to the left and to the right continually. Give it a try!

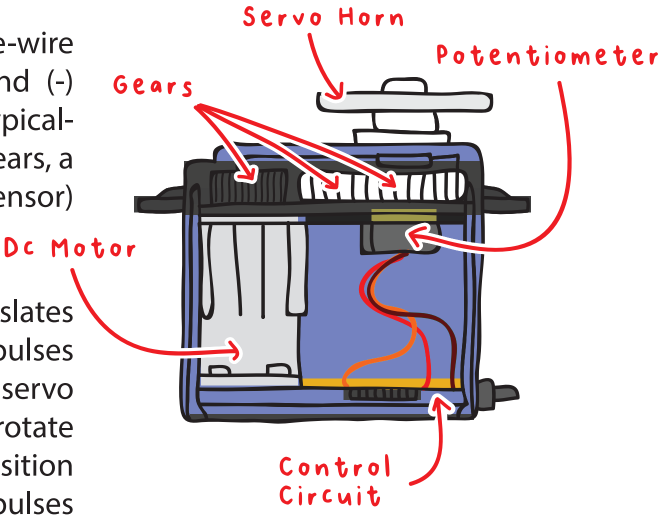
FUN FACT!

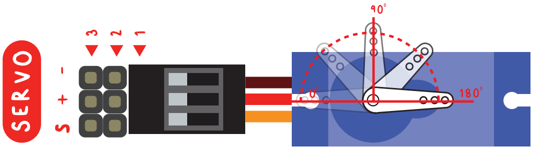
The **servo motor** in EDU:BIT kit is also known as a RC (radio control) servo. It is widely used in RC toy vehicles and small robots to control their movement.

A servo motor uses a three-wire system for power (+) , ground (-) and control or signal (s). It typically consists of a DC motor, gears, a potentiometer (position sensor) and a control circuit.

The built-in controller translates commands in the form of pulses to position in degree. The servo motor will constantly rotate towards and stay at the position which corresponds to the pulses received.

Unlike a DC motor that rotates continuously, we can control a servo motor rotation to our desired angle between the range of 0 to 180 degrees.



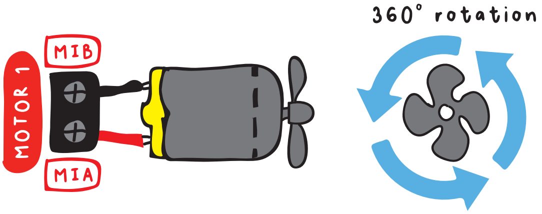


0-180° rotation

Servo Motor

VS

DC Motor



Learn more!

youtu.be/okxooamdAP4

APPLICATION CHALLENGE

Program EDU:BIT to function as a metronome. When powered on, swing the pointer (attached to the servo motor horn) to the left and then to the right repetitively at a constant tempo. Set the tempo to be controlled by Potentio Bit. When the yellow button (Button A) is pressed, show the current tempo (e.g. 120 bpm).

Here are some tips for you...

Tip #1: You need to create two variables: Tempo and Delay.

Tip #2: A typical metronome range is from 40 to 200 bpm.

Tip #3: A tempo of 60 bpm (or beat per minute) means the pointer swings from one end to another a total 60 times in a minute, i.e. once every one second.

Tip #4: The faster the tempo, the lesser the delay.

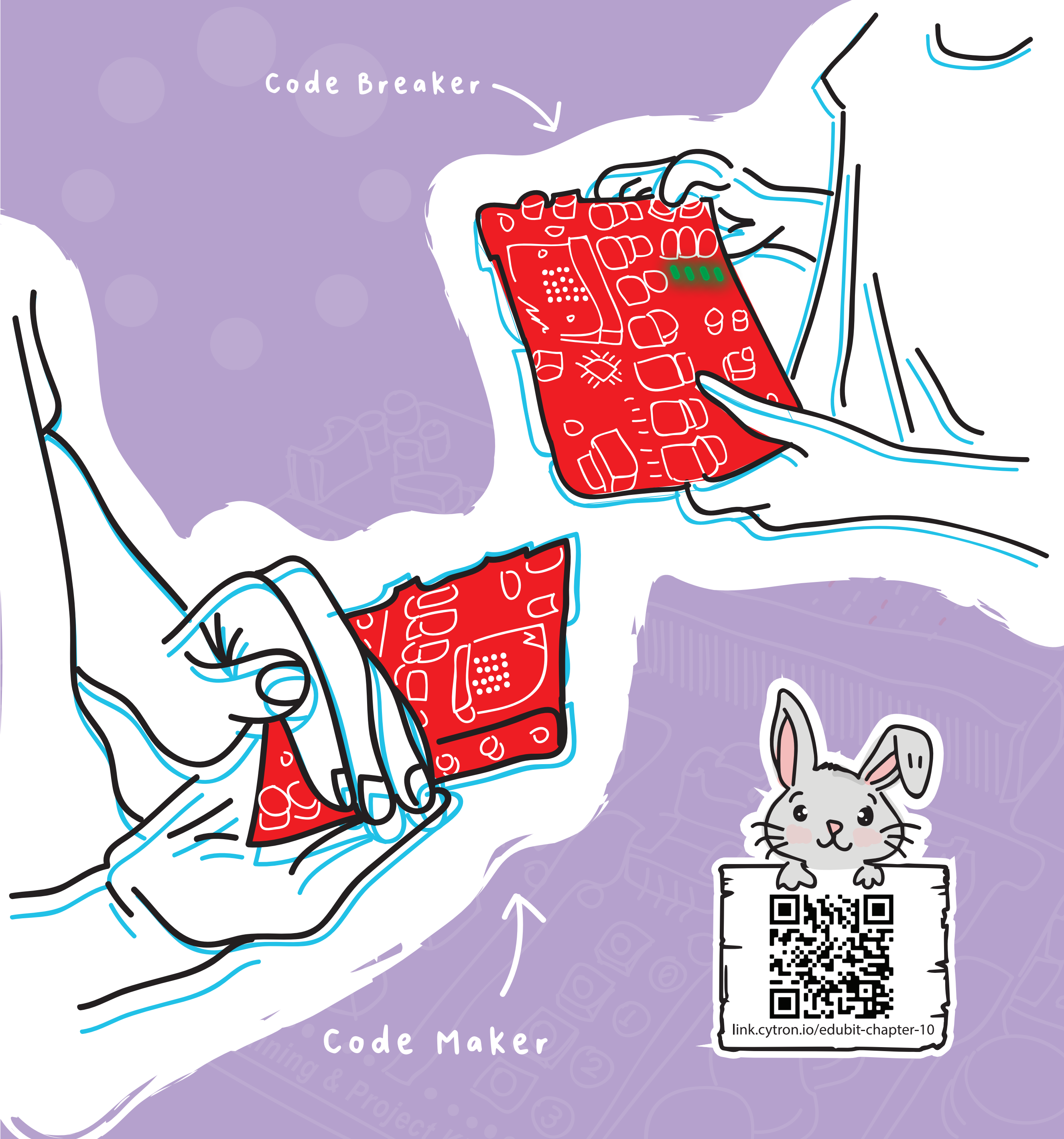
"A metronome is a device that produces an audible click or other sound at a regular interval that can be set by the user, typically in beats per minute (bpm). Musicians use the device to practice playing to a regular pulse. Metronomes typically include synchronized visual motion."

- wikipedia -

CHAPTER 10

Mastermind, Can You Break the Secret Code?

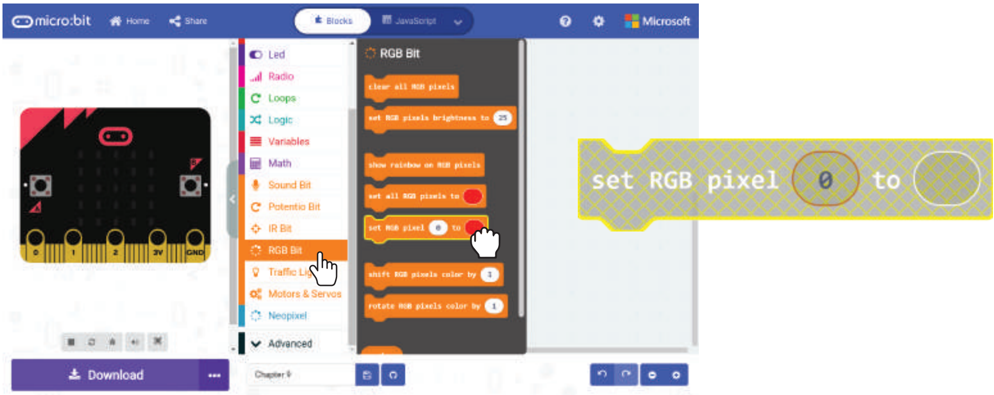
RGB Bit



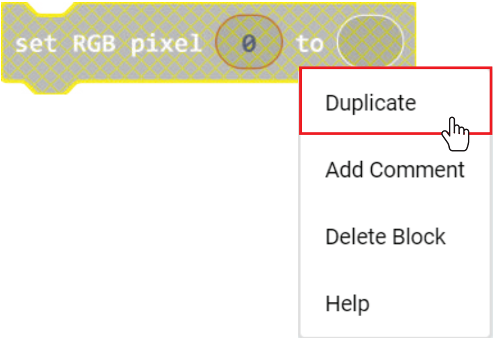
link.cytron.io/edubit-chapter-10

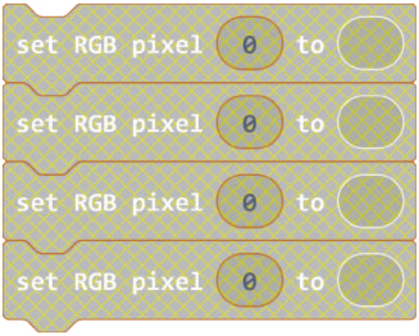
LET'S CODE!

Step 1 In your MakeCode Editor, create a new project and add EDU:BIT extension. Click [RGB Bit] category and then select [set RGB pixel _ to _] block.

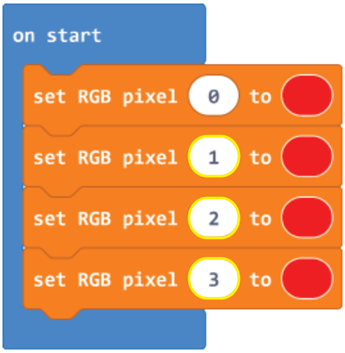


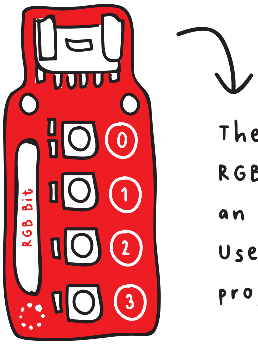
Step 2 In the Workspace, right-click on the [set RGB pixel _ to _] block and then click 'Duplicate'. Repeat until you have four [set RGB pixel _ to _] blocks.





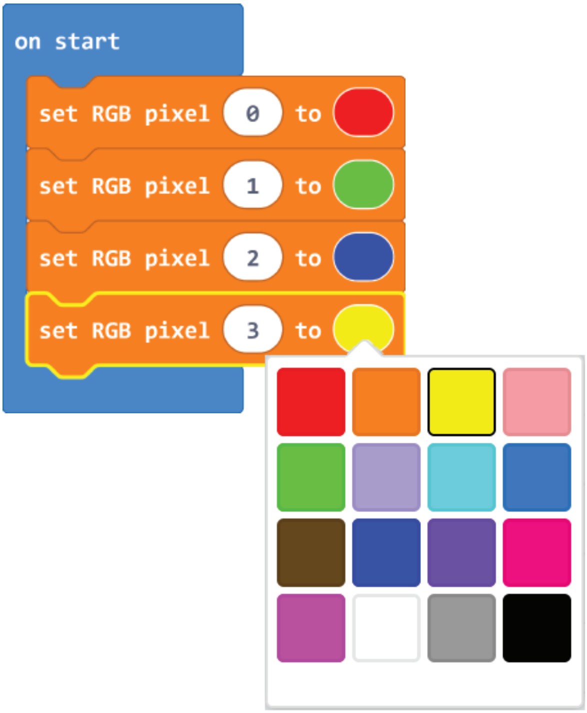
Step 3 Place the blocks in the slot of the [on start] block. Change the RGB pixel number from 0 to **1, 2,** and **3** for the second, third and fourth blocks.


```
on start
    set RGB pixel 0 to
    set RGB pixel 1 to
    set RGB pixel 2 to
    set RGB pixel 3 to
```



There are 4 RGB LEDs on RGB Bit and each is assigned an identification number (0-3). Use this number to program each LED individually.

Step 4 Flash the code to your EDU:BIT.



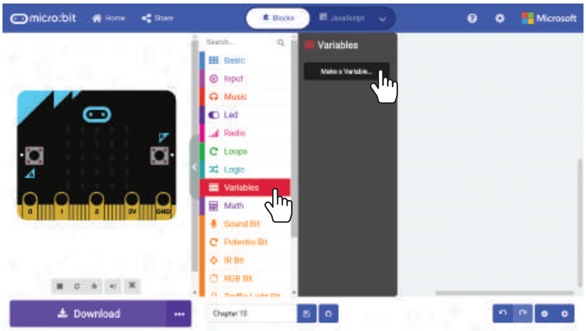
```
on start
    set RGB pixel 0 to 🟥
    set RGB pixel 1 to 🟩
    set RGB pixel 2 to 🟦
    set RGB pixel 3 to 🟨
```

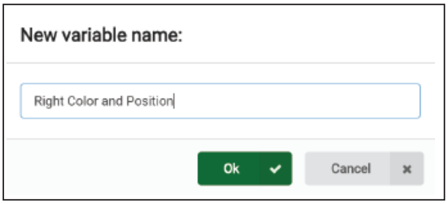
When you power up the board, the four LEDs on RGB Bit will light up according to the colours set.

You can easily change the colour by clicking the block and selecting another available colour from the palette.

Give it a try!

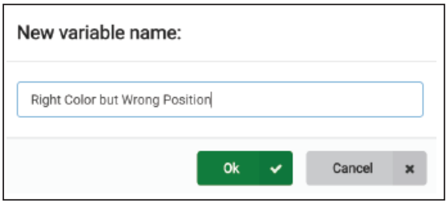
Step 5 Add two new variables and name them - **"Right Color and Position"** and **"Right Color but Wrong Position".**





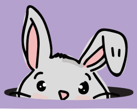
New variable name:

Right Color and Position

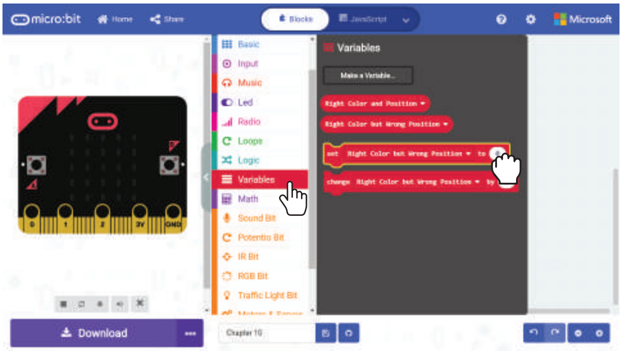


New variable name:

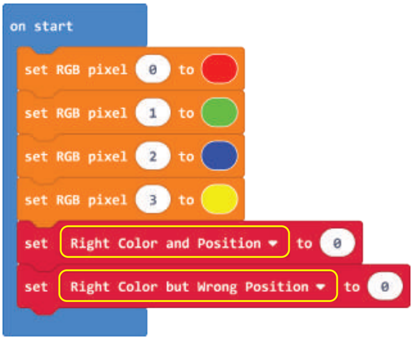
Right Color but Wrong Position



Step 6 Click **[Variables]** category and select **[set _ to _]** block. Duplicate the block and snap them to **[on start]** block. Set one variable to **"Right Color and Position"**, while the other one to **"Right Color but Wrong Position"**.

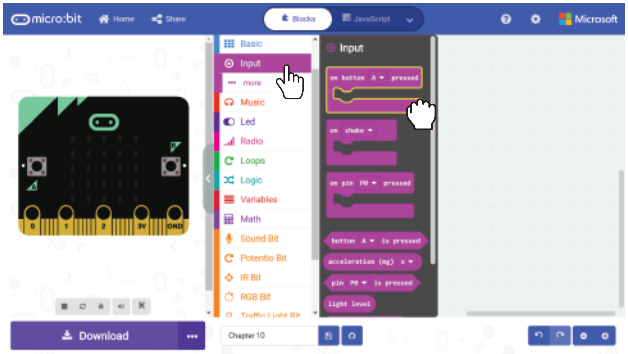


Right Color and Position
Right Color but Wrong Position
set Right Color but Wrong Position to 0
change Right Color but Wrong Position by 1

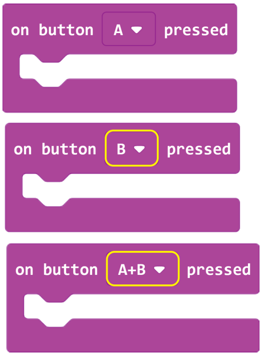


```
on start
    set RGB pixel 0 to
    set RGB pixel 1 to
    set RGB pixel 2 to
    set RGB pixel 3 to
    set Right Color and Position ▾ to 0
    set Right Color but Wrong Position ▾ to 0
```

Step 7 Click **[Input]** category and select **[on button _ pressed]** block. Duplicate the block. Change the second block to **B** and the third block to **A+B.**

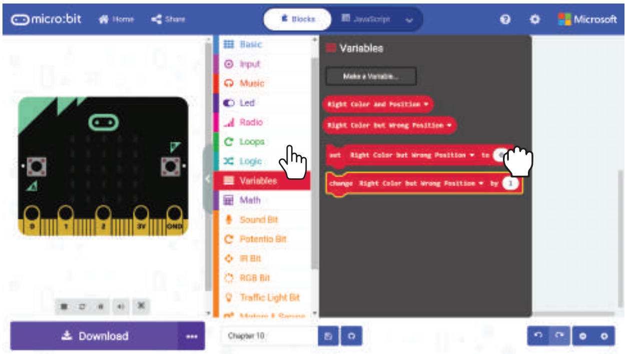


on button A ▾ pressed
on shake ▾
on pin P0 ▾ pressed
button A ▾ is pressed
acceleration (mg) x ▾
pin P0 ▾ is pressed
light level

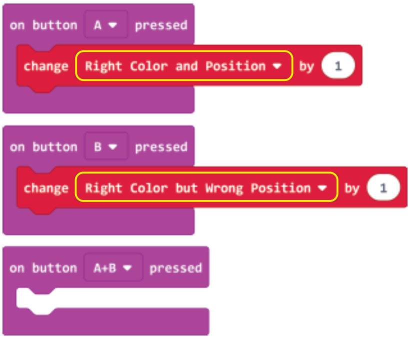


```
on button A ▾ pressed

on button B ▾ pressed

on button A+B ▾ pressed
```

Step 8 Click **[Variables]** category again and select **[change _ by _]** block. Duplicate and attach the blocks to **[on button A pressed]** and **[on button B pressed]** blocks. Set one variable to **"Right Color and Position"**, while the other one to **"Right Color but Wrong Position"**.

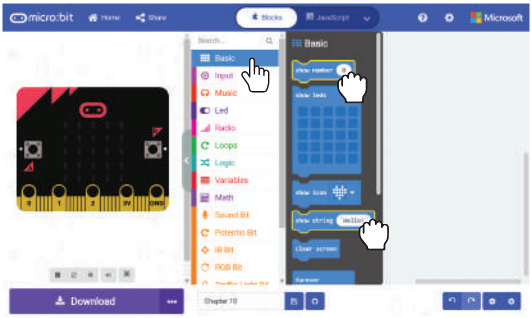


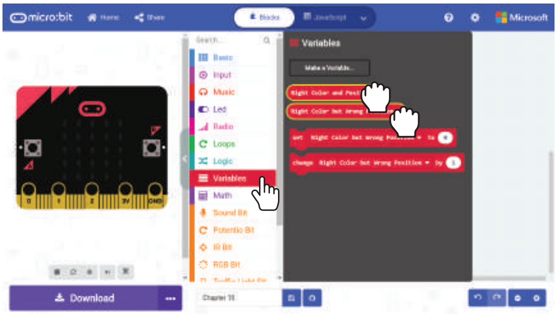
Right Color and Position
Right Color but Wrong Position
set Right Color but Wrong Position to 0
change Right Color but Wrong Position by 1



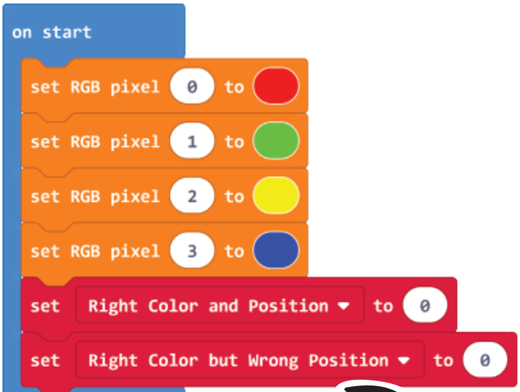
```
on button A ▾ pressed
    change Right Color and Position ▾ by 1

on button B ▾ pressed
    change Right Color but Wrong Position ▾ by 1

on button A+B ▾ pressed
```

Step 9 Add [show number] and [show string] blocks from [Basic] category, as well as [Right Color and Position] and [Right Color but Wrong Position] blocks from [Variables] category.

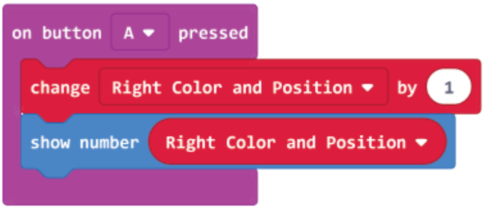




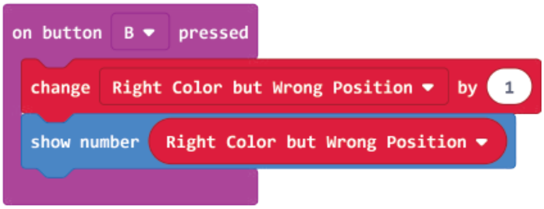
Step 10 Change "Hello!" in the [show string] blocks to **"A ="** and **"B ="** respectively. Here's the completed code.



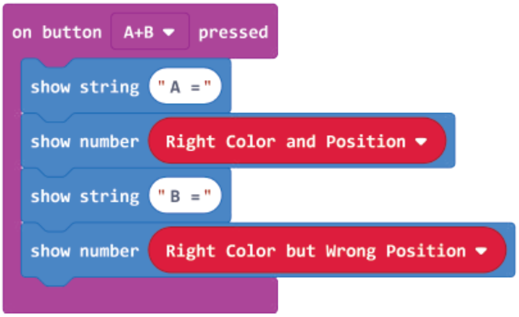
```
on start
    set RGB pixel 0 to 🟥
    set RGB pixel 1 to 🟩
    set RGB pixel 2 to 🟨
    set RGB pixel 3 to 🟦
    set Right Color and Position ▾ to 0
    set Right Color but Wrong Position ▾ to 0
```



```
on button A ▾ pressed
    change Right Color and Position ▾ by 1
    show number Right Color and Position ▾
```



```
on button B ▾ pressed
    change Right Color but Wrong Position ▾ by 1
    show number Right Color but Wrong Position ▾
```

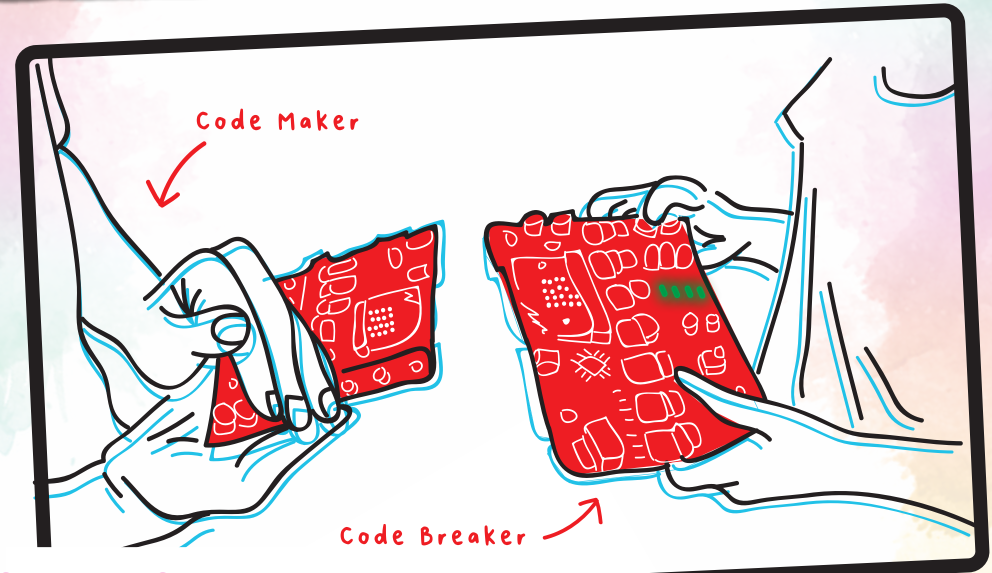


```
on button A+B ▾ pressed
    show string "A ="
    show number Right Color and Position ▾
    show string "B ="
    show number Right Color but Wrong Position ▾
```

Step 11 Flash the code to your EDU:BIT. You're now ready to play Mastermind Game.

Let’s Play

Mastermind, Can You Break The Secret Code?



HOW TO PLAY:

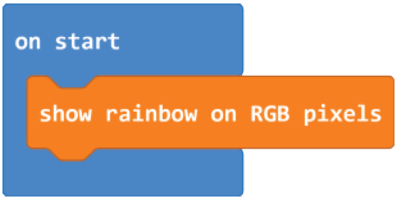
- Player 1, the Code Maker, sets the secret code by lighting up the four LEDs in a random sequence. Limit the colour option to 🔴 and 🟢 only for a start. Remember to cover your board so that the other player cannot see the colour pattern.
- Player 2, the Code Breaker, tries to guess the secret code. Player 2 lights up the RGB LEDs on his/her EDU:BIT, and then shows it to the Code Maker.
- The Code Maker checks and then presses the yellow button (Button A) and/or blue button (Button B) on the Code Breaker’s EDU:BIT to indicate how many LED(s) are lighted up with the “right color and position” and “right color but wrong position”.
- The Code Breaker can then press both the yellow and blue buttons at the same time to “read” the feedback given.
- Repeat Steps 2 and 3 until the Code Breaker correctly guesses the color sequence (maximum 10 attempts per round).
- Exchange roles to play another round.

How to Win:

You win the game if you successfully break the code (guess the colour sequence). Failing to do that, the victory goes to the Code Maker.

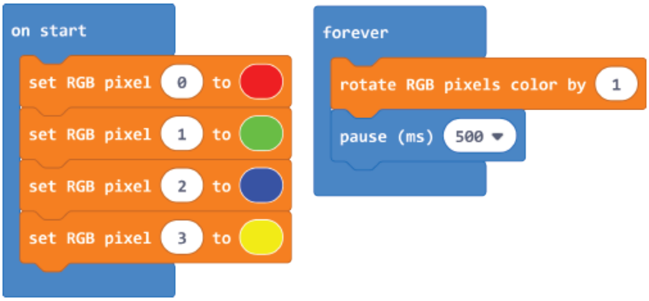
EXPLORE MORE BLOCKS

#1 You can set the LEDs on RGB Bit to display the colours of the rainbow by replacing [set RGB pixel _ to _] block with [show rainbow on RGB pixels] block.



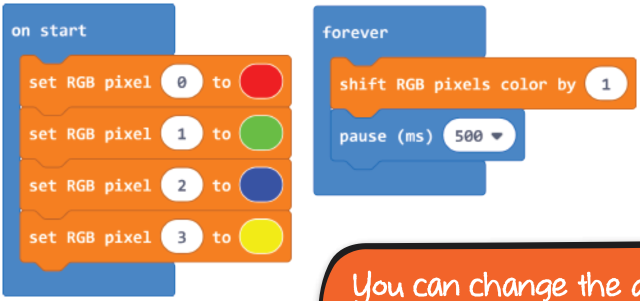
```
on start
    show rainbow on RGB pixels
```

#2 You can create a running light effect by placing a [rotate RGB pixels color by _] block in a [forever] block. Remember to add a [pause] block to slow down the program so that you can see the effect. Here's a sample code:



```
on start
    set RGB pixel 0 to 🟥
    set RGB pixel 1 to 🟩
    set RGB pixel 2 to 🟦
    set RGB pixel 3 to 🟨

forever
    rotate RGB pixels color by 1
    pause (ms) 500
```

#3 You can also shift the pixels one by one using the [shift RGB pixels color by _] block in a [forever] block. You will need to add a [pause] block to slow down the program so that you can see the effect, whereby the pixels will be turned off one by one. Here's a sample code:

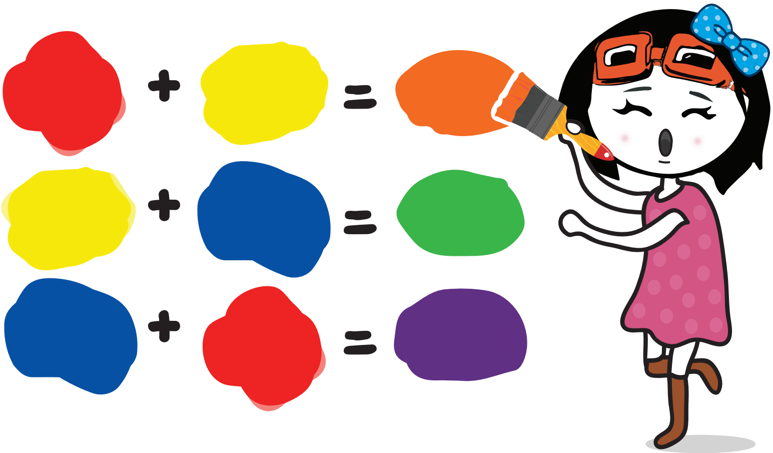


```
on start
    set RGB pixel 0 to 🟥
    set RGB pixel 1 to 🟩
    set RGB pixel 2 to 🟦
    set RGB pixel 3 to 🟨

forever
    shift RGB pixels color by 1
    pause (ms) 500
```

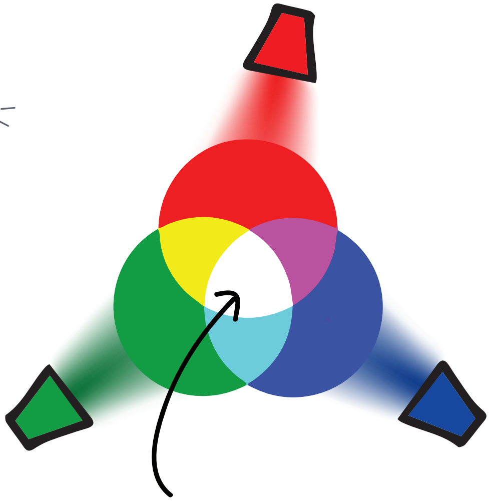
You can change the direction of the effect for #2 and #3 above by changing the setting from a positive value to a negative value.

FUN FACT!

In Art classes, you probably learn that the 3 primary colours are Red, Yellow and Blue. And when you mix them, you get this result:



However, for devices that use lights to display colours, e.g. your tv and computer screen, the RGB colour model is used.



In this model, the three light primaries are Red (R), Green (G) and Blue (B) instead; and when combined, they create White light!

APPLICATION CHALLENGE

Program EDU:BIT to be a memory game training tool.

How it works?

- To start, tilt EDU:BIT to the left to light up the LEDs on RGB Bit in a random colour pattern for a few seconds.
- You will have to observe and then say out the color sequence after the LEDs have turned off.
- To check your answer, press the blue button (Button B) to light up the RGB LEDs in the same pattern again.
- If you answer correctly, press the yellow button (Button A) to update and display your score. Game Over if you answer wrongly.
- You can adjust the difficulty level of the game by turning the knob of Potentio Bit to increase/decrease the duration that the LEDs stay lighted up.
- Player with the highest score wins!

Here are some tips for you:
Tip #1 : You need to create two variables - Score and Pattern.
Tip #2 : You need to pre-set the colour sequence (colour for each RGB LED) for each pattern. Use more colours or pre-set more patterns to make the game more challenging. And vice versa, you might want to limit the colours/patterns for younger players.

BONUS CHAPTER

Simon Says with LEDs

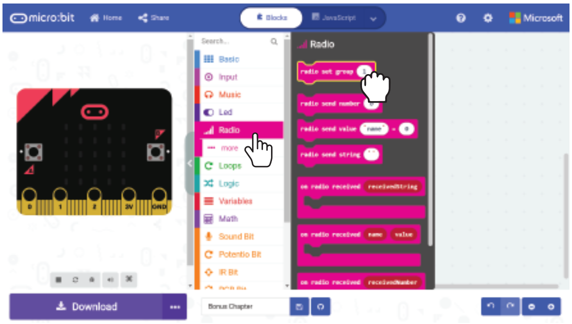
Radio Communication

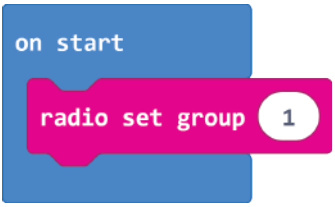
link.cytron.io/edubit-bonus-chapter

For any form of communication to take place, we need at least two parties - a sender and a receiver. In this game, we need two EDU:BITs to communicate with each other by sending and receiving radio broadcast signals.

LET'S CODE!

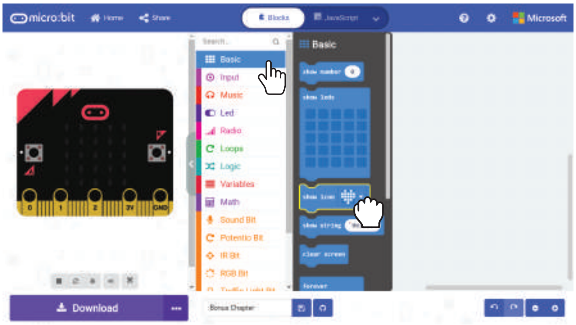
Step 1 In your MakeCode Editor, create a new project and add EDU:BIT extension. Click [Radio] category and select [radio set group _] block. Snap the block to the [on start] block.

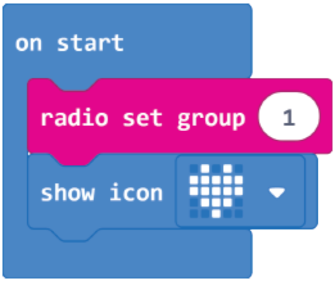




```
on start
    radio set group 1
```

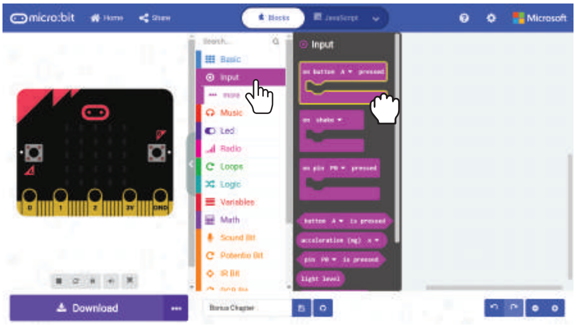
Step 2 In your MakeCode Editor, click [Basic] category and then add [show icon] block to your program.

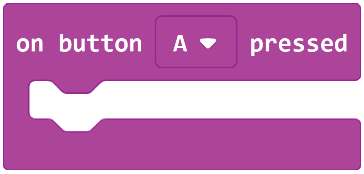




```
on start
    radio set group 1
    show icon
```

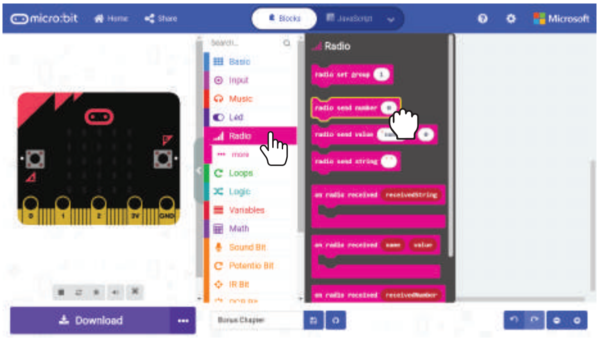
Step 3 Click [Input] category and select [on button _ pressed] block.

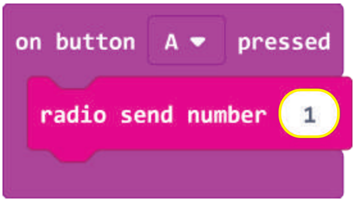




```
on button A pressed
```

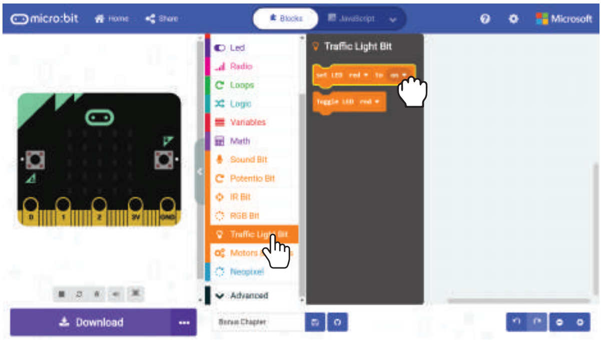
Step 4 Click [Radio] category and select [radio send number _] block. Change the value to **1.**

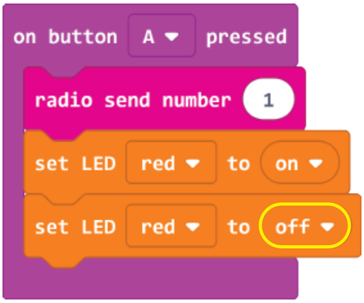




```
on button A ▾ pressed
    radio send number 1
```

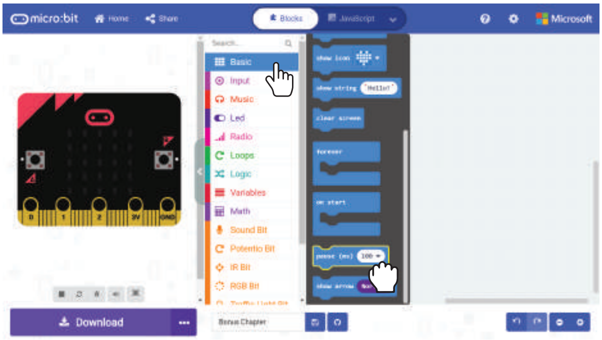
Step 5 Click [Traffic Light Bit] category and select [set LED _ to _] block. Duplicate it and snap both blocks to the [on button A pressed] block. Change the second block setting to **'off'.**

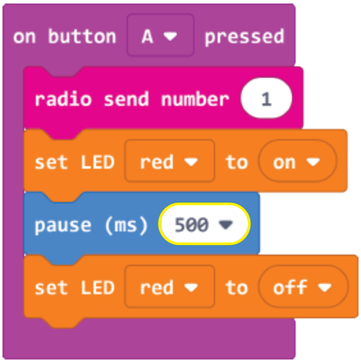




```
on button A ▾ pressed
    radio send number 1
    set LED red ▾ to on ▾
    set LED red ▾ to off ▾
```

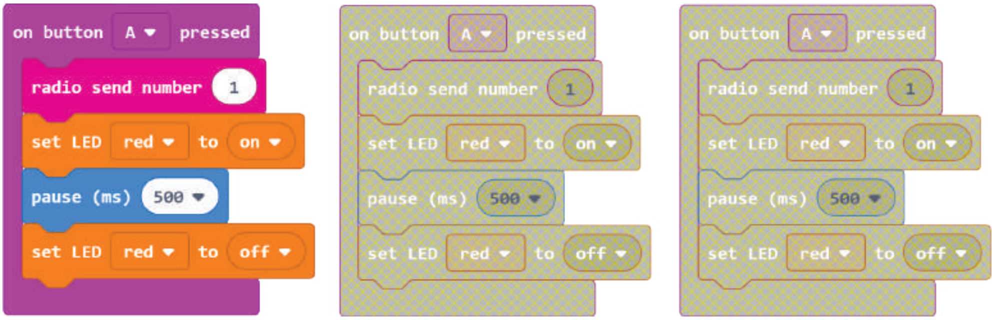
Step 6 Click [Basic] category and select [pause (ms) _]. Snap the block in between the [set LED _ to _] blocks and change the value to **500.**





```
on button A ▾ pressed
    radio send number 1
    set LED red ▾ to on ▾
    pause (ms) 500 ▾
    set LED red ▾ to off ▾
```

Step 7 Right-click on the **[on button A pressed]** block and then select "Duplicate'. Repeat to get three sets of the same blocks.

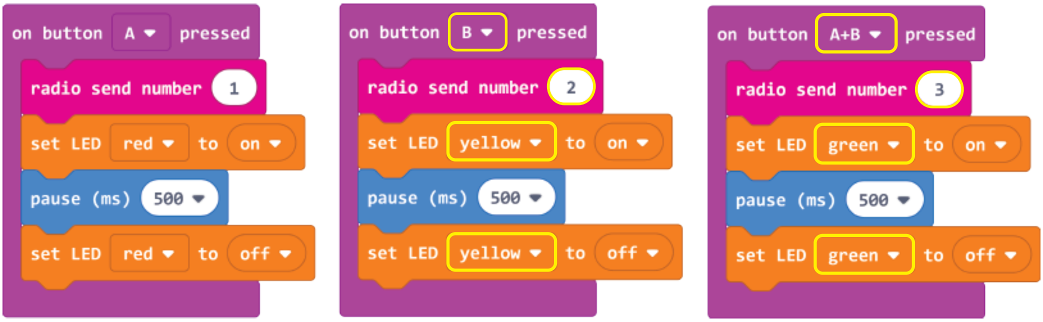


```
on button A ▾ pressed
    radio send number 1
    set LED red ▾ to on ▾
    pause (ms) 500 ▾
    set LED red ▾ to off ▾
```

```
on button A ▾ pressed
    radio send number 1
    set LED red ▾ to on ▾
    pause (ms) 500 ▾
    set LED red ▾ to off ▾
```

```
on button A ▾ pressed
    radio send number 1
    set LED red ▾ to on ▾
    pause (ms) 500 ▾
    set LED red ▾ to off ▾
```

Step 8 Change the settings for the button, number value and LED colour of the duplicated blocks of code to the following:

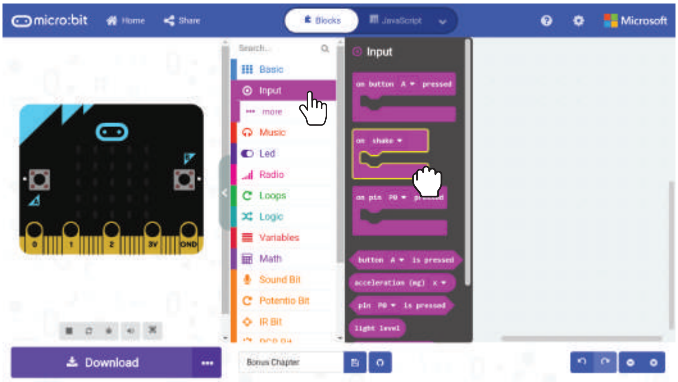


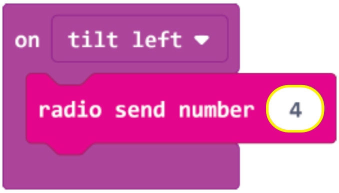
```
on button A ▾ pressed
    radio send number 1
    set LED red ▾ to on ▾
    pause (ms) 500 ▾
    set LED red ▾ to off ▾
```

```
on button B ▾ pressed
    radio send number 2
    set LED yellow ▾ to on ▾
    pause (ms) 500 ▾
    set LED yellow ▾ to off ▾
```

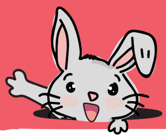
```
on button A+B ▾ pressed
    radio send number 3
    set LED green ▾ to on ▾
    pause (ms) 500 ▾
    set LED green ▾ to off ▾
```

Step 9 Click **[Input]** category and select **[on _]** block. Change the setting to **"tilt left"**. Click **[Radio]** category and select **[radio send number _]** block. Change the value to **4**.

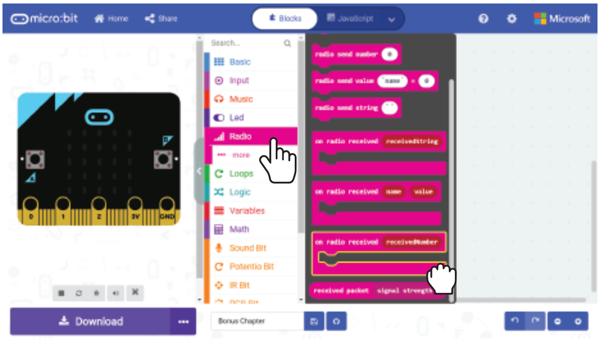


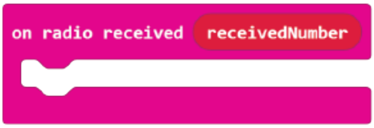


```
on tilt left ▾
    radio send number 4
```



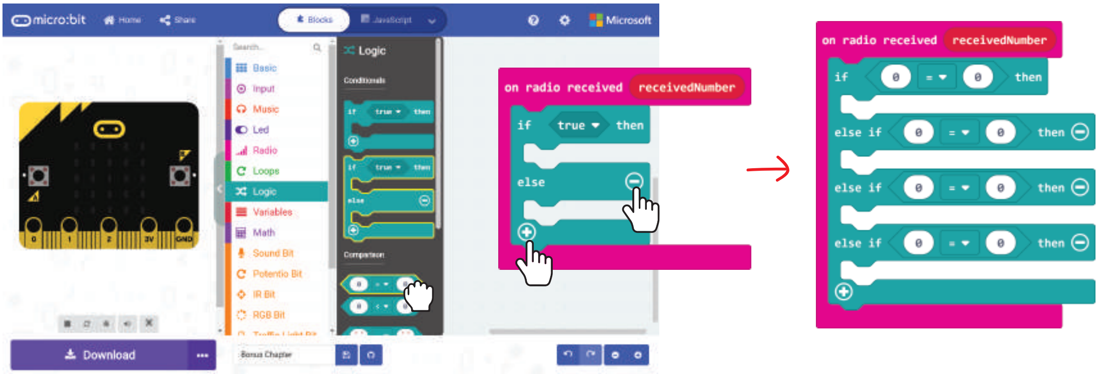
Step 10 Click **[Radio]** category and select **[on radio received receivedNumber]** block.





```
on radio received receivedNumber
```

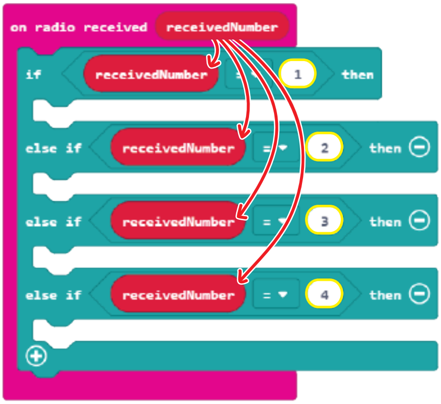
Step 11 Click **[Logic]** category and select **[if-then-else]** block. Click the ⊕ button to add three **[else if]** conditions and click the ⊖ button to remove the **[else]** condition. Attach **[Logic] : [_ = _]** comparison block to each of the "if" and "else if" slots.



```
on radio received receivedNumber
    if true then
    else
```

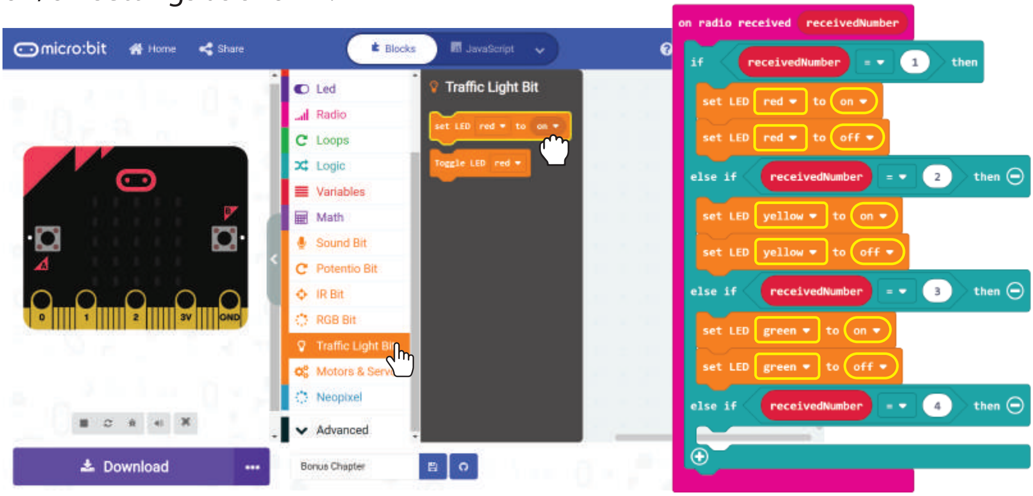
```
on radio received receivedNumber
    if 0 = 0 then
    else if 0 = 0 then
    else if 0 = 0 then
    else if 0 = 0 then
```

Step 12 Click and drag **'receivedNumber'** variable into the comparison blocks as shown and change the values to **1, 2, 3** and **4** respectively.



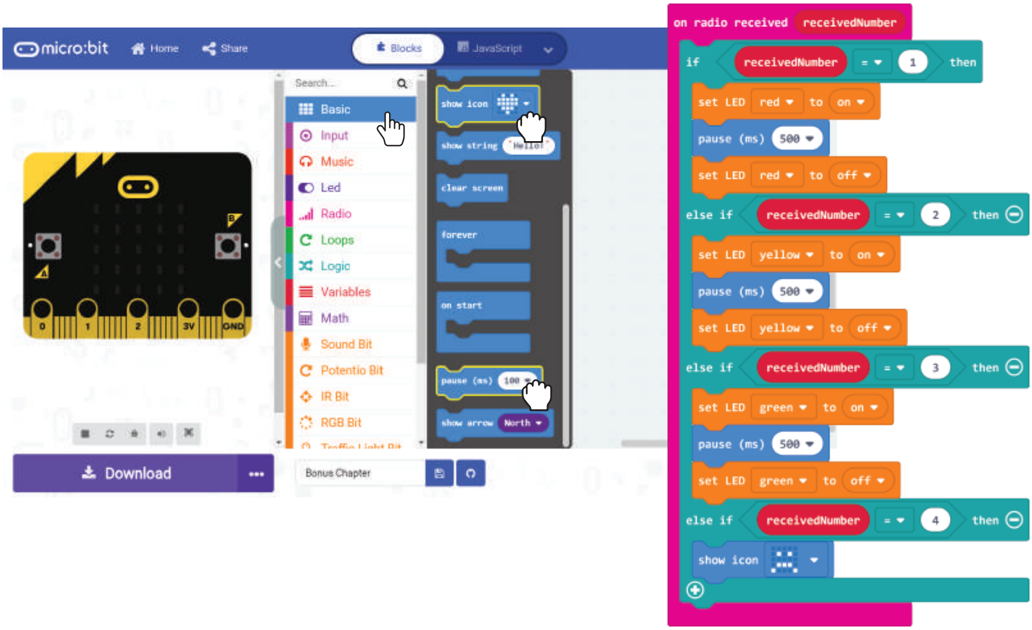
```
on radio received receivedNumber
    if receivedNumber = 1 then
    else if receivedNumber = 2 then
    else if receivedNumber = 3 then
    else if receivedNumber = 4 then
```

Step 13 Click **[Traffic Light Bit]** category and select **[set LED _ to _]** block. Duplicate the block and snap them to the first three slots. Change the colour and on/off settings as shown.



```
on radio received receivedNumber
    if receivedNumber = 1 then
        set LED red to on
        set LED red to off
    else if receivedNumber = 2 then
        set LED yellow to on
        set LED yellow to off
    else if receivedNumber = 3 then
        set LED green to on
        set LED green to off
    else if receivedNumber = 4 then
```

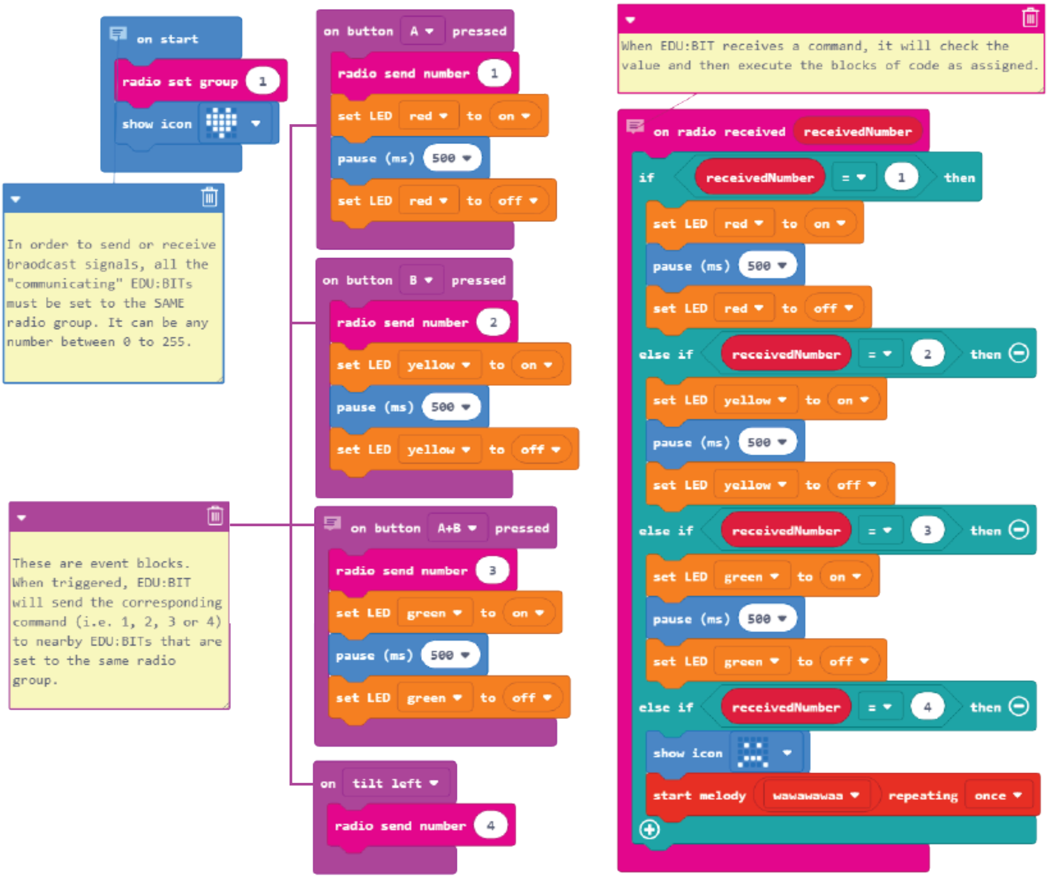
Step 14 Click **[Basic]** category and select **[pause (ms) _]**. Duplicate and snap the blocks in between the **[set LED _ to _]** blocks and change the value to **500**. Add **[Basic] : [show icon]** block to the final "else if" slot and change the icon to "sad face".



```
on radio received receivedNumber
    if receivedNumber = 1 then
        set LED red to on
        pause (ms) 500
        set LED red to off
    else if receivedNumber = 2 then
        set LED yellow to on
        pause (ms) 500
        set LED yellow to off
    else if receivedNumber = 3 then
        set LED green to on
        pause (ms) 500
        set LED green to off
    else if receivedNumber = 4 then
        show icon
```

Step 15 Click **[Music]** category and add **[start melody _ repeating _]** to complete your code. Change the melody to **"wawawawaa"** (or any melody of your choice).

Here's the complete code:



```
on start
    radio set group 1
    show icon
```

In order to send or receive braodcast signals, all the "communicating" EDU:BITs must be set to the SAME radio group. It can be any number between 0 to 255.

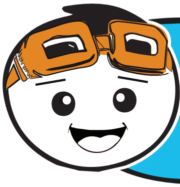
```
on button A pressed
    radio send number 1
    set LED red to on
    pause (ms) 500
    set LED red to off

on button B pressed
    radio send number 2
    set LED yellow to on
    pause (ms) 500
    set LED yellow to off

on button A+B pressed
    radio send number 3
    set LED green to on
    pause (ms) 500
    set LED green to off

on tilt left
    radio send number 4
```

These are event blocks. When triggered, EDU:BIT will send the corresponding command (i.e. 1, 2, 3 or 4) to nearby EDU:BITs that are set to the same radio group.

When EDU:BIT receives a command, it will check the value and then execute the blocks of code as assigned.

```
on radio received receivedNumber
    if receivedNumber = 1 then
        set LED red to on
        pause (ms) 500
        set LED red to off
    else if receivedNumber = 2 then
        set LED yellow to on
        pause (ms) 500
        set LED yellow to off
    else if receivedNumber = 3 then
        set LED green to on
        pause (ms) 500
        set LED green to off
    else if receivedNumber = 4 then
        show icon
        start melody wawawawaa repeating once
```

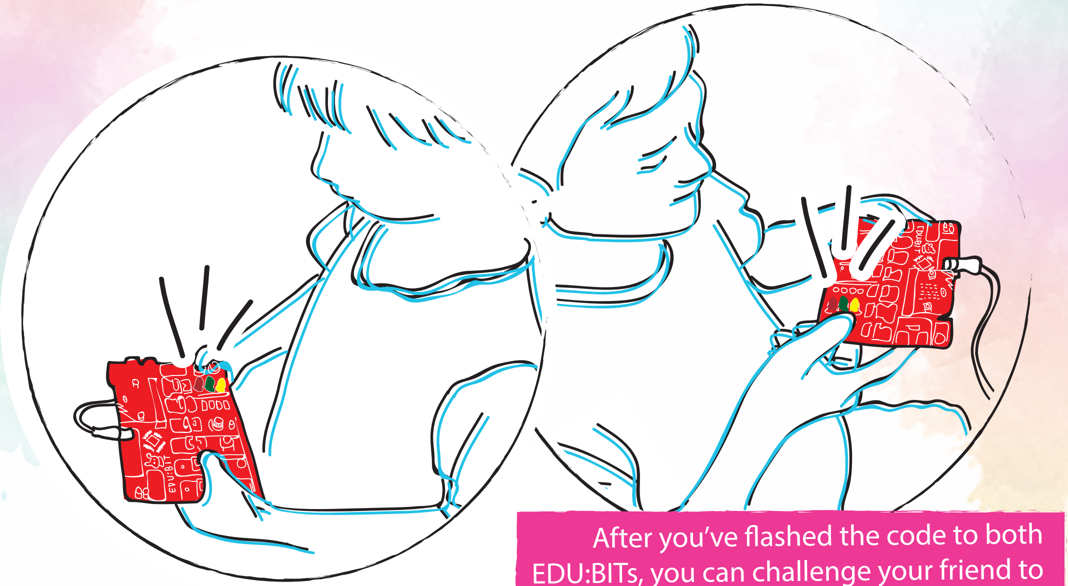
Step 16 Flash the completed code to your EDU:BIT as well as your friend's.



When both EDU:BITs are powered on, you can send radio signals to light up the LEDs on your friend's EDU:BIT; and vice versa, your friend can also press the buttons on his EDU:BIT to light up the LEDs on your board.

Let's Play

Simon Says with LEDs



After you've flashed the code to both EDU:BITs, you can challenge your friend to an interactive version of Simon Says game.

HOW TO PLAY:

Both players take turns to be "Simon". Press the buttons to light up the red, yellow and green LEDs when it is your turn.

- To start the game, Player 1 lights up ONE of the LEDs on Traffic Light Bit.
- Player 2 observes and then lights up the same LED, followed by another LED.
- The game continues with each player taking turns to repeat the latest sequence, and then light up another LED to add to the sequence.
- If any player lights up the wrong LED (or in wrong sequence), then the opponent will tilt his/her EDU:BIT to end the game.
- The loser will then need to reset his or her EDU:BIT to start a new game.

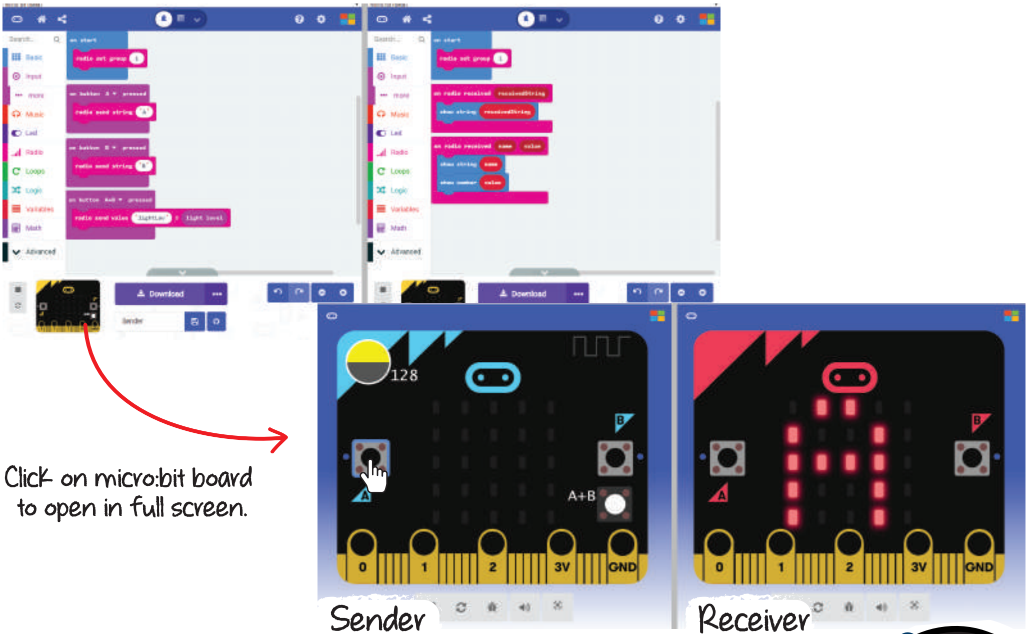
This game starts off easy but it gets progressively longer and more complex after each turn. In order to win, you'll need to observe the sequence carefully. It's a fun game to train your memory power.

EXPLORE MORE BLOCKS

#1 Besides sending numbers, you can also send text messages using the [radio send string " _ "] block. You will need to use [on radio received receivedString] block to receive a string broadcast signal. Maximum string length is 19 characters.

#2 Use [radio send value " _ " = _] block and [on radio received name value] block if you need to send and/or receive text and number together. Maximum string length is 8 characters.

If you do not have access to multiple EDU:BITs, you can still test out the radio communication function. Just go to makecode.com/multi to write your code for the "sender" and "receiver". You can view the result on the simulator windows.



```
on start
    radio set group 1
on button A ▾ pressed
    radio send string "A"
on button B ▾ pressed
    radio send string "B"
on button A+B ▾ pressed
    radio send value "lightLev" = light level
```

```
on start
    radio set group 1
on radio received receivedString
    show string receivedString
on radio received name value
    show string name
    show number value
```

Click on micro:bit board to open in full screen.

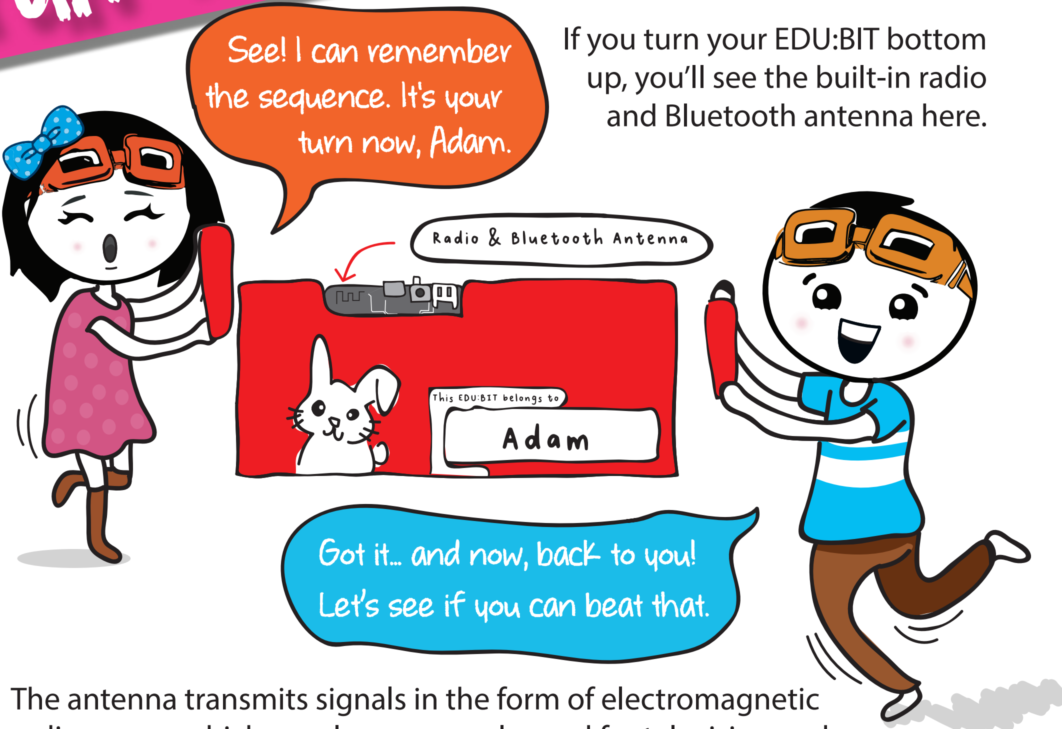
Sender

Receiver

When you press Button A on the "sender" micro:bit, the receiver micro:bit will receive the radio signal and display the received string, i.e. A. What happens if you press buttons A+B?

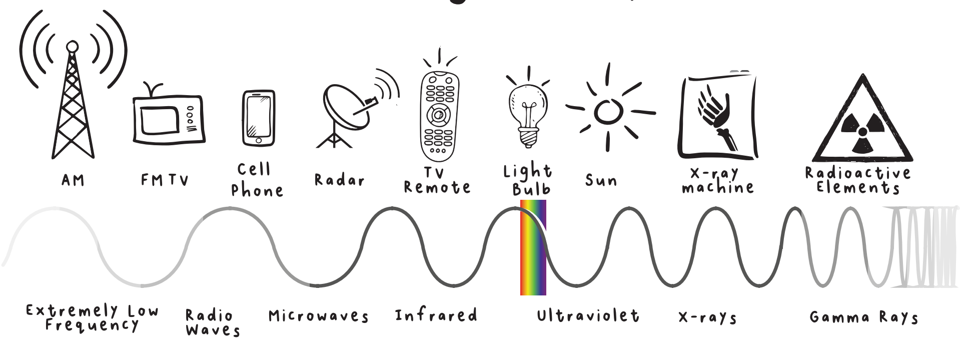
FUN FACT!

If you turn your EDU:BIT bottom up, you'll see the built-in radio and Bluetooth antenna here.


See! I can remember the sequence. It's your turn now, Adam.
Radio & Bluetooth Antenna
This EDU:BIT belongs to
Adam
Got it... and now, back to you! Let's see if you can beat that.

The antenna transmits signals in the form of electromagnetic radio waves, which are also commonly used for television and radio broadcasting, as well as satellite transmissions.

Electromagnetic Spectrum



NOTE!

In order for your EDU:BIT to send and receive radio broadcast signals from other EDU:BITs, you'll need to set all of them to the same radio group.

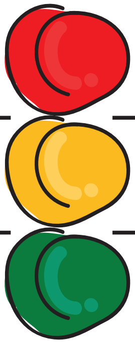
APPLICATION CHALLENGE

Set up a feedback network communication system for your class.

How it works?

Set every EDU:BIT in the class to the same radio group.

The teacher's EDU:BIT is set to scroll text when it receives a "string" signal and light up the LEDs on Traffic Light Bit when it receives a "number" radio signal, whereby ….

Number Received	Light up LED	What it means?
1	RED 	A / No / False
2	YELLOW	B / Maybe / Not Sure
3	GREEN	C / Yes / True

Students' EDU:BITs are set to send a string with the student's name and then send a number (1 or 2 or 3) to light up the LEDs on the teacher's EDU:BIT when triggered.

Press Button A to send number 1.
Press Button B to send number 2.
Press Buttons A+B to send number 3.

Here's a tip for you. Assign short nicknames (with just two or three characters) for each student in order to cut down the text scrolling time.

I have learnt to...

□ program EDU:BIT to display text and animation on the LED matrix.
□ download, save, publish and edit MakeCode .hex files.

□ use input blocks for event-based programming.
□ create and use variables

□ use piezo buzzer on Music Bit to play simple melodies.
□ create and use functions.
□ read music scores.

□ program EDU:BIT to control LEDs on Traffic Light Bit - on, off & toggle.
□ add extensions to MakeCode Editor.

□ program EDU:BIT to detect objects using IR Bit.
□ use while loops.
□ create and use arrays.

□ program EDU:BIT to read analog values from Potentio Bit.
□ map analog input readings.
□ use logic blocks for conditional programming.

□ program EDU:BIT to detect noise with Sound Bit.
□ use event triggers to switch between several modes.

□ program EDU:BIT to control a DC motor - spinning direction and speed control.
□ use math blocks to perform arithmetic operations.

□ program EDU:BIT to control a servo motor - angular position.

□ program EDU:BIT to light up RGB LEDs on RGB Bit in different colours/patterns.

□ program EDU:BIT to send and receive radio signals.

*Tick the check box if you've mastered the skill; otherwise go back to the relevant chapter to revise.

NOTE FROM RERO EDUTEAM @ CYTRON

CONGRATULATIONS!!!

You've made it through all the chapters and learned to code with MakeCode Editor. We hope you've had fun jazzing up and playing our selection of popular childhood games. And thumbs up for having developed some handy applications for your classroom.

By now, you should have a good grasp of what you can do with micro:bit and all the extra Bits that come with your EDU:BIT board. Psst… do you know that you can break off each of the Bits?

Go ahead and "break" them if you like. Once broken from the main board, you can build new projects with the various Bits; you'll need to connect them using the plug-and-play cables provided.

And now it's time for you to put on your thinking cap and brainstorm for new games and applications. We can't wait to see what amazing projects you are going to come up with.

Remember to share your creations with us. Email or leave us a message on our FB page. We'd love to hear from you.

Cheers,
Adam & Anna

link.cytron.io/
edubit-resource-hub

www.cytron.io www.fb.com/cytrontech support@cytron.io

ISBN 978-967-19475-0-0