Technical guide for BBC micro:bit

Project number: 2023-1-PL01-KA220-SCH-000154043

Project number: 2023-1-PL01-KA220-SCH-000154043

IoT4Schools
"Bringing the Internet of Things in school education as a tool to address 21st century challenges"

Technical Guide for BBC micro:bit

Authors: C. Papasarantou, R. Alimisi [EDUMOTIVA]

Contributors: A. Tefelska, D. Tefelski [WUT]

Reviewers: Heron-Mathisis, Atermon

Co-funded by
the European Union

About the technical guide for BBC micro:bit

The present document, produced in the framework of the IoT4Schools Erasmus+ project, serves as a technical guide for the BBC micro:bit. The objective of this document is to assist educators and students in becoming familiar with the BBC micro:bit microcontroller in terms of both operational and programming aspects, as well as through the lens of the Internet of Things (IoT). In order to achieve this objective, the guide includes a description of the board, its built-in sensors and actuators and a comprehensive overview of the official programming environment, namely the Microsoft Makecode software, with a particular focus on its block-based version. Furthermore, instructions are provided on how to utilize the integrated Bluetooth radio antenna, accompanied by indicative examples of its implementation, thus inspiring readers on how the micro:bit can be effectively employed in the development of IoT projects.

Table of contents

1. Introduction to BBC micro:bit

BBC micro:bit board (Figure 1) is a pocket size microcontroller, primarily designed for educational purposes to introduce young learners to the ICT field in a simple and intuitive way. Having several in built input and output devices, micro:bit offers a handy solution for the creation of a number of STEM related interesting projects, without the need of extra hardware. In addition, the micro:bit is compatible with many of the sensors and electronic components commonly used with other boards (such as Raspberry Pi Pico and Arduino), extending its capabilities even further. There are currently two different versions of the micro:bit board: V1 and V2. In the IoT4Schools project, the V2 version is used as it has some additional components.

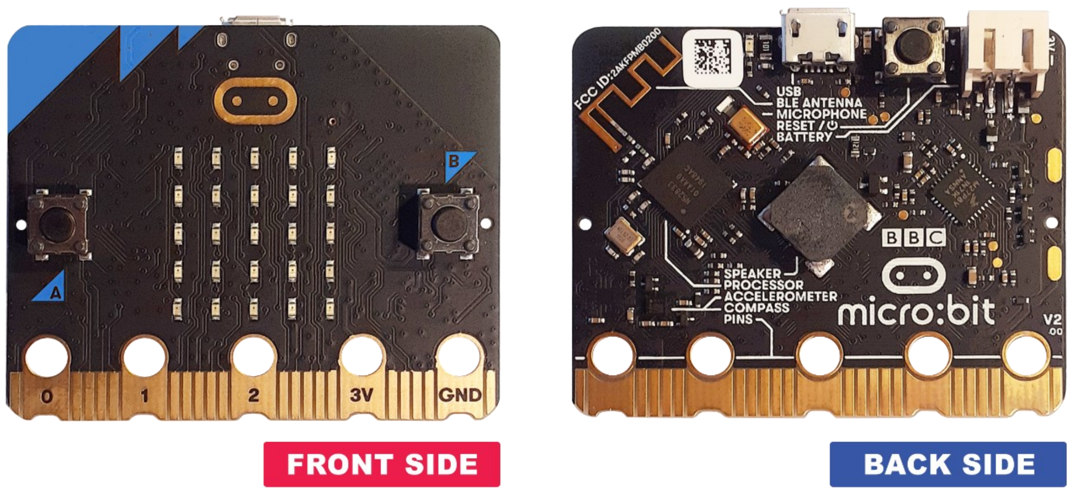


Figure 1: The two sides of the BBC micro:bit board

In some cases, the use of different edge connector breakout boards is recommended, as they allow the stable connection of more electronic components, making the circuit design process easier. Connecting the micro:bit board to an edge connector breakout board is normally a straightforward process, as it only involves clipping the micro:bit onto the edge connector (see an example in Figure 2).

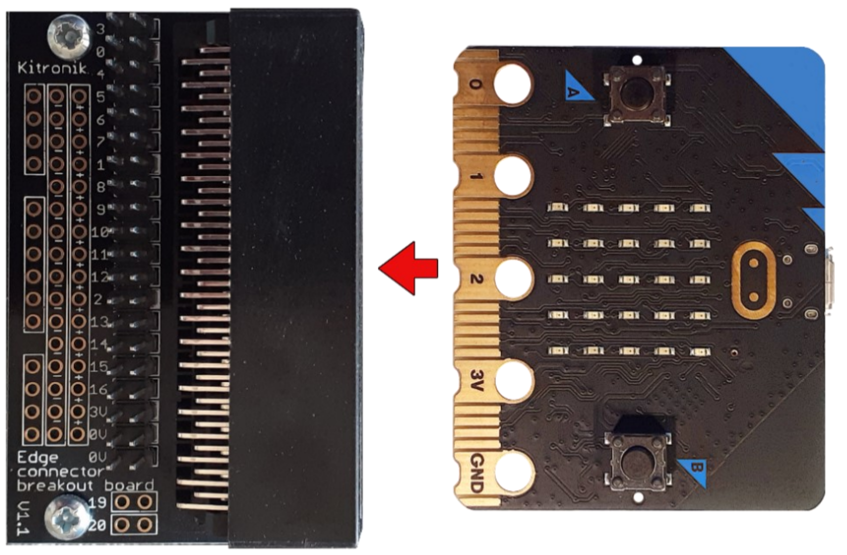


Figure 2: Clipping the micro:bit onto an indicative edge connector An indicative edge connector breakout board

2. The BBC micro:bit board

As mentioned, the BBC micro:bit board has a number of in-built input and output devices. Some of these are on the front of the micro:bit and some are on the back of the micro:bit board. The following diagrams show where each device is located.

Front side (Figure 3):

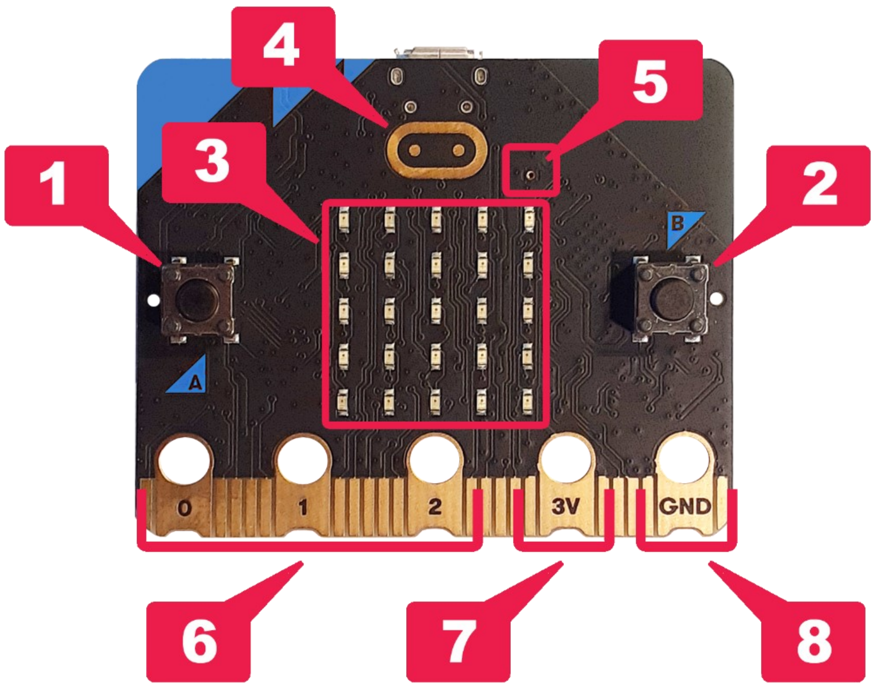


Figure 3: In-built input and output devices on the front side of the micro:bit

- Button A **(1)** and button B **(2)**; two input devices that can be programmed to trigger an action when pressed
- A 5x5 LED display **(3)**; functioning as an input (light sensor) and output (display screen) device
- A touch sensor **(4)** (available in V2); an input device that allows micro:bit to respond to touch
- A microphone **(5)** (available in V2); an input device that allows micro:bit to respond to external audio events
- Several GPIO pins **(6)**, including 3V power **(7)** and Ground **(8)** for connecting more electronic components through crocodile clips or through an Edge Connector breakout board.

Back side (Figure 4):

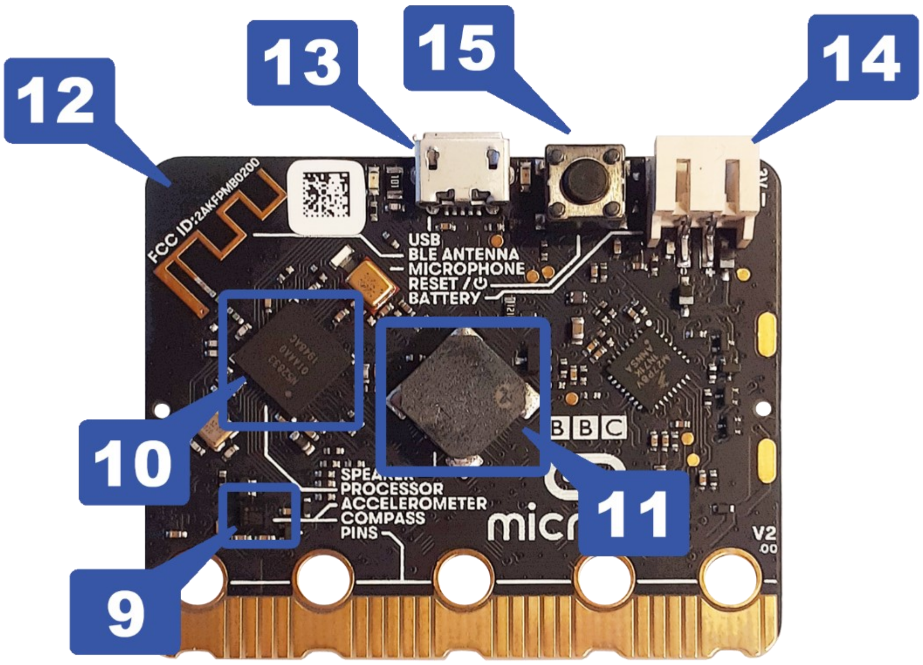


Figure 4: In-built input and output devices on the back side of the micro:bit

- An accelerometer **(9)**; an input motion sensor that detects if micro:bit is shaken, dropped or turned upside down
- A compass **(9)**; an input sensor that detects Earth's magnetic field allowing micro:bit to detect the direction that is facing
- A processor **(10)**; the "brain" of the micro:bit that processes all the information by receiving input, running programs and giving outputs. Inside the processor there is a temperature sensor, allowing micro:bit to measure its temperature in degrees of Celsius
- A speaker **(11)** (available in V2); an output device that allows the reproduction of sounds
- A Bluetooth Radio antenna **(12)**; that allows micro:bit to wirelessly communicate with other micro:bits or other devices. This in-built device is extremely valuable for the realization of IoT projects (see section 4).
- A USB socket **(13)**; through which microbit can be connected to a computer or to an external power source

- A JST battery connector **(14)**; where a power source with a JST connector can be connected
- A power switch **(15)** (available in V2); a button that can power on/off or reset micro:bit when hold pressed

3. Programming BBC micro:bit – The Microsoft Makecode Environment

The official software for programming the BBC micro:bit is Microsoft Makecode (https://makecode.microbit.org/); a web-based software that offers both block-based and text-based (Javascript or Python) coding solutions, making it a flexible and suitable for young students programming tool. To use Makecode software, internet connection is needed. This adds some limitations regarding accessibility. Makecode can be also used through smart mobile devices (phones or tablets) and through iOS and Android apps that are available for download[1].

The micro:bit board can be also programmed in various block-based environments, such as Scratch (via the Scratch Link App), pictoblox, microblocks.fun, Open Roberta, Art.bit, micro:bit Python editor (https://python.microbit.org/v/3) and more. Unlike all the others, Scratch is officially supported by the micro:bit Educational Foundation.

3.1 Microsoft Makecode environment: becoming familiar with the interface

The following diagram (Figure 5) shows an overview of the Microsoft Makecode environment.

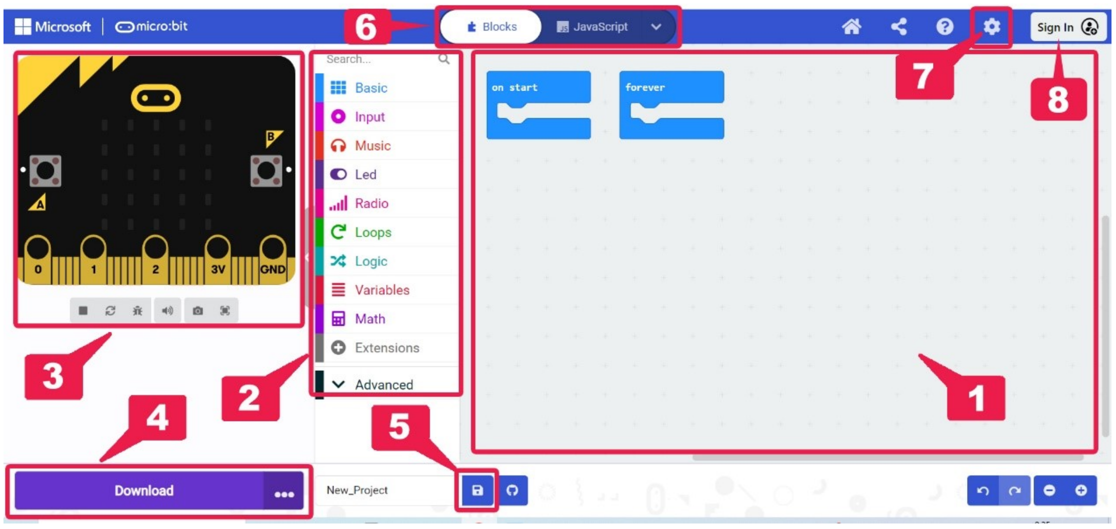


Figure 5: The Microsoft Makecode interface

[1] **Note:** To connect and exchange data with a smart device (smartphone or tablet) you must first pair your micro:bit through Bluetooth. To do this, simultaneously press buttons A (1) and B (2) (Figure 3). While holding them pressed, press the reset button (15) (Figure 4) for one second. Then release it and continue holding Buttons A and B, until the LED screen (3) (Figure 3) displays a pattern. Release the buttons. After a while the Bluetooth symbol or the word “Paired” will be displayed on the LED screen. To find micro:bit, open your smart device's Bluetooth settings and scan for available devices.

In particular, there you can find:

- an area **(1)** where you can assemble your code/script by dragging, dropping and snaping command blocks
- several command blocks **(2)** grouped in color-coded groups, to program your project
- a simulator **(3)** where you can test how your code/script works, before downloading it to your micro:bit board
- a download button **(4)**, to directly download the code/script to your micro:bit
- a save button **(5)**, to save the code/script locally to your computer as a .hex file
- a switcher **(6)**, to switch to a text-based coding environment (JavaScript or Python), and vice-versa
- a "more…" **(7)** button/menu to access various functions such as "project settings", "extensions", "language" and more.
- A sign in **(8)** button, to log into your personal account. It is highly recommended that you create an account to avoid losing your projects.

3.2 Microsoft Makecode environment: creating a new project

To create and start working on a new project enter to your browser Makecode's website address (https://makecode.microbit.org/), and at the homepage, click on the New Project tab (1). On the "Create a Project" pop-up menu (2), type a name of your choice for your project, and then click on the "Create" button (Figure 6).

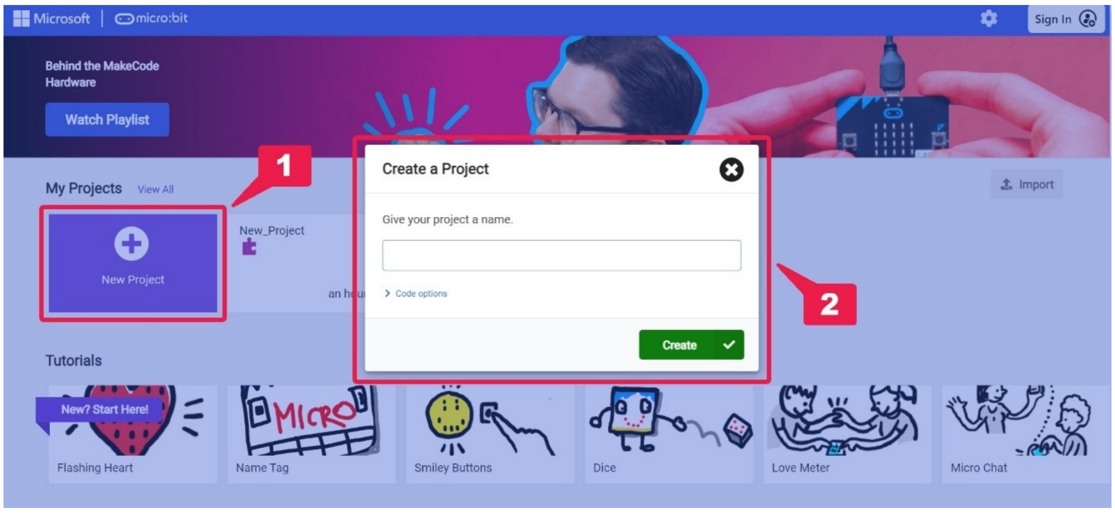


Figure 6: Creating a new project

You can also import an existing project by clicking the "import" button **(3)**, and selecting where you want to import from (i.e. from your computer, from the cloud or from GitHub), from the "Import" pop-up menu **(4)** (Figure 7).

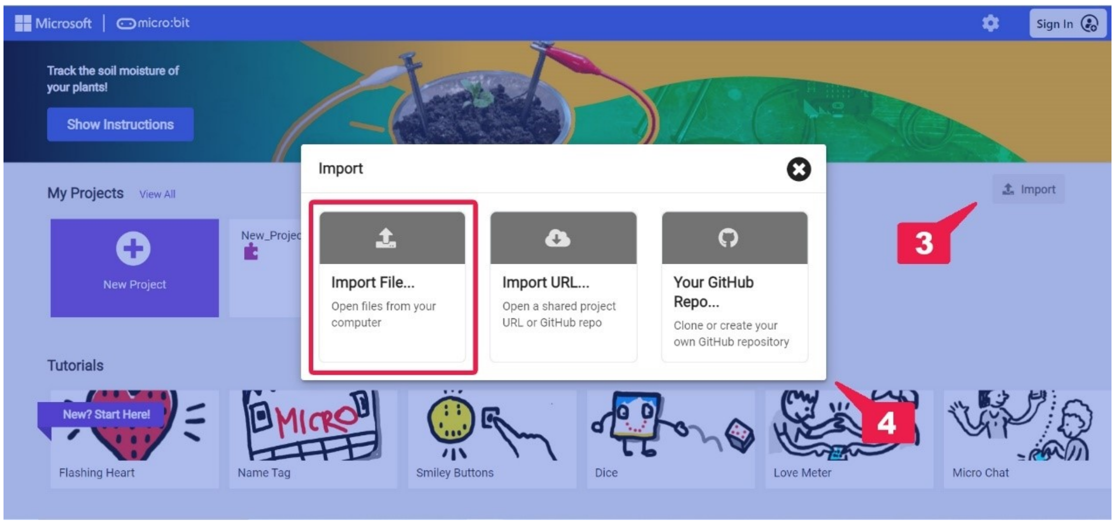


Figure 7: Selecting where to import from

3.3 Microsoft Makecode environment: command blocks and groups

There are many different command blocks that you can use in your project (Figure 8). These blocks are organized into different color-coded command groups **(1)**. Take some time to explore what all these groups can do, and what kind of blocks contained therein. For instance, "**Basic**" command group contains blocks such as "On start" and "Forever" event handler blocks, or "show number/string …" blocks **(2)**, that help you to program some basic actions and functions. "**Input**" command group, contains command blocks that help you programm several input devices such as buttons, light sensor and temperature, while "**Logic**" command group contains operators and constants, such as "if…else.." condition commands and boolean logic commands.

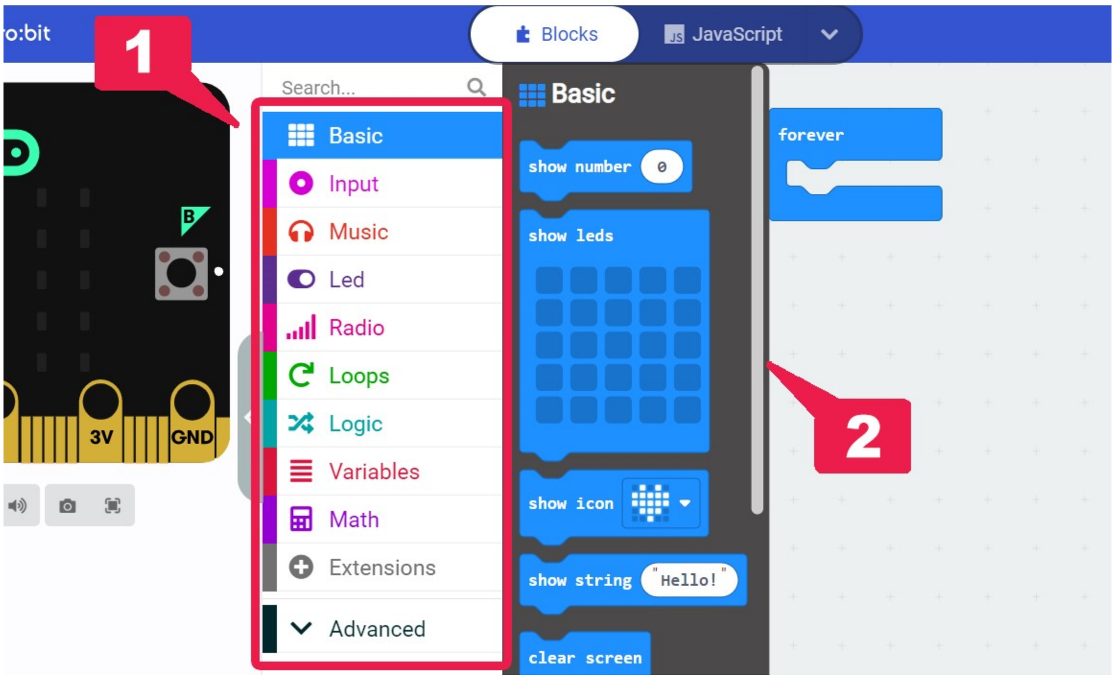


Figure 8: Groups and blocks of commands

When clicked, some command groups (such as "**Input**" or "**Radio**") will reveal a sub-menu "…more" (Figure 9). These groups contain a second tab with command blocks.

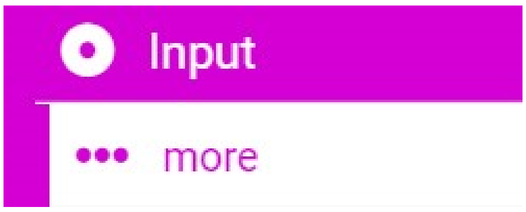


Figure 9: the "…more" sub-menu

There is a black command group called "**Advanced**" (Figure 10a.). Clicking on this will take you to other command groups (Figure 10b.) that can be used in advanced projects, such as "**Functions**" and "**Arrays**".

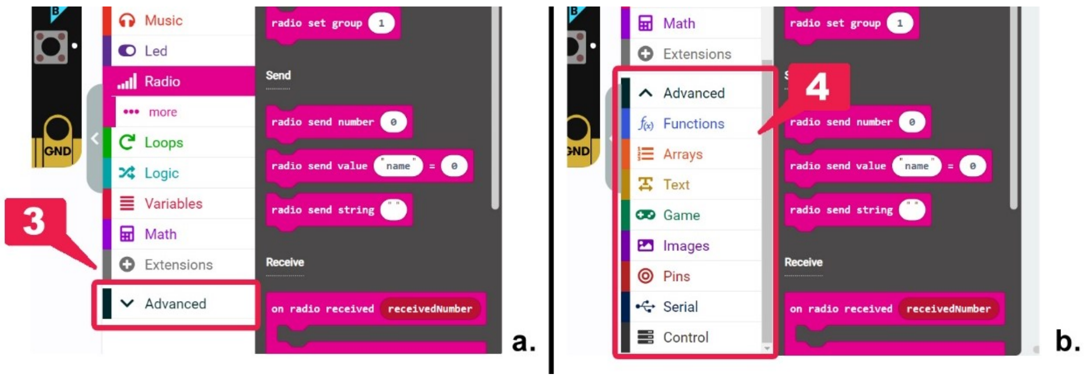


Figure 10: a. Advanced menu selection; b. command groups included in Advanced menu

In addition to the standard command groups, you can import other blocks designed and provided by other programmers in the micro:bit community. These blocks are located in the "**Extensions**" group **(5)** (Figure 11) and can help to program more complex electronics and sensors, thus extending the capabilities of the micro:bit.

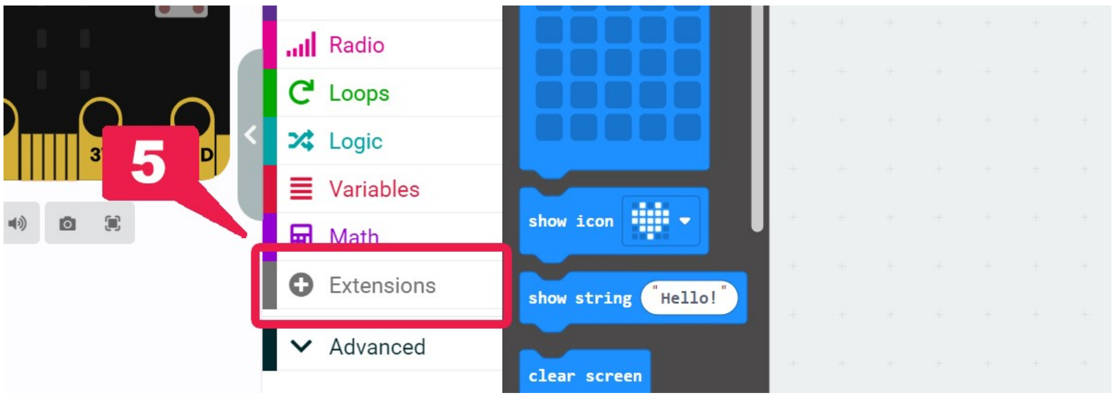


Figure 11: The "Extensions" command group

Clicking on the "**Extensions**" group will take you to the Extensions search menu (Figure 12). Here you can either search among the recommended extensions **(7)**, or type the name **(6)** of a component you want to program (e.g. ESP8266, Bluetooth etc.) and see the available options.

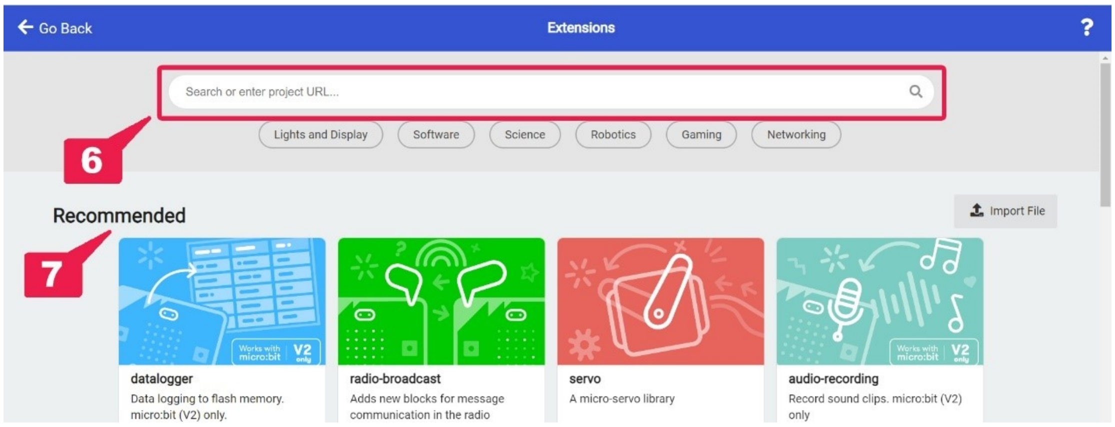

Go Back
Extensions
?
Search or enter project URL...
Lights and Display
Software
Science
Robotics
Gaming
Networking
6
Recommended
Import File
7
datalogger
Data logging to flash memory. micro:bit (V2) only.
radio-broadcast
Adds new blocks for message communication in the radio
servo
A micro-servo library
audio-recording
Record sound clips. micro:bit (V2) only

Figure 12: Extensions search menu

3.4 Microsoft Makecode environment: assembling a code/script

To create/assemble a script in the block-based version of Makecode, you have to snap together different command blocks in a way that makes sense from a coding/programming perspective. A script can be assembled by using hat and stack blocks (Figure 13). Hat blocks are mandatory, since they are the blocks that start a script. Stack blocks, which are usually the main commands of a code, are puzzle-like blocks that can fit inside a hat block. A script must contain at least one hat block and as many stack blocks as necessary for the purpose of a project.

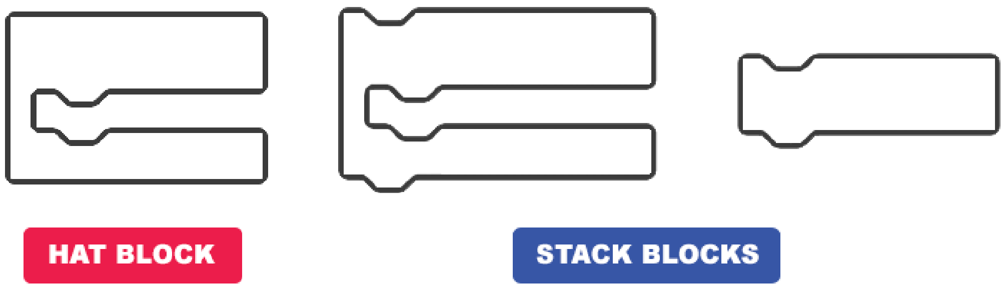


Figure 13: Examples of how Hat and Stack blocks can look like

Additionally, there are also some capsule-like and some hexagonal blocks (Figure 14), which cannot be used independently, but must be integrated into specific stack blocks. These blocks usually contain data such as strings, numeric data, variables, Boolean operators, and comparisons.

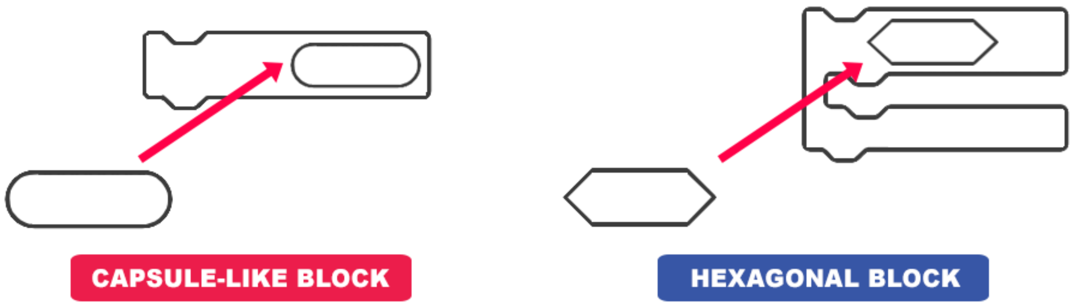


Figure 14: Capsule-like and hexagonal blocks

Note: If you are not sure whether a block will fit into another block, just try it. If the commands cannot be combined, you will either not be allowed to integrate the block or you will receive a warning (Figure 15).

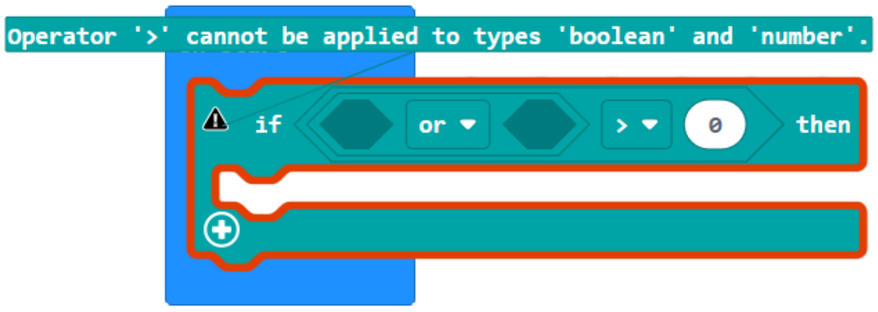

Operator '>' cannot be applied to types 'boolean' and 'number'.

Figure 15: Error in the code. The hexagonal operator cannot be integrated in this part of the script

Some commands may have a drop-down menu (Figure 16). Click the arrow and search the floating menu that appears, to discover additional programming options.

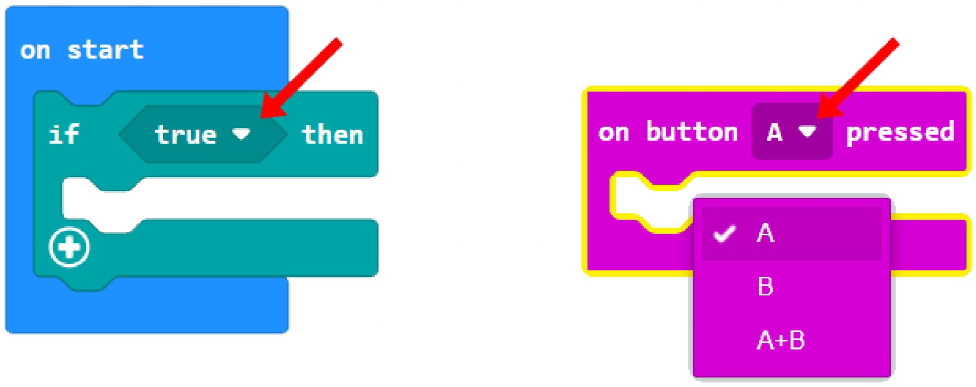


Figure 16: Commands with drop-down menu

Figure 17 presents some conceptual examples of script structures. Such scripts can be assembled by dragging and dropping command blocks into the assembly script area, and snapping them together. The length of the presented scripts is indicative.

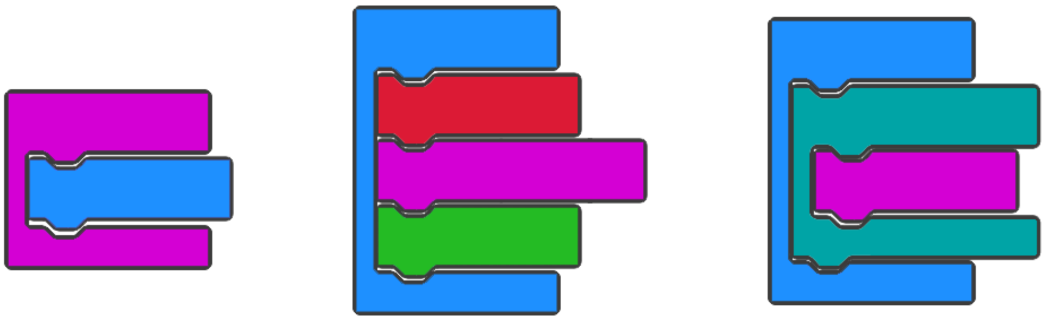


Figure 17: Examples of assembled scripts

If a block (or a stack of blocks) is grey, this means that this script is inactive and it will not be executed (Figure 18).

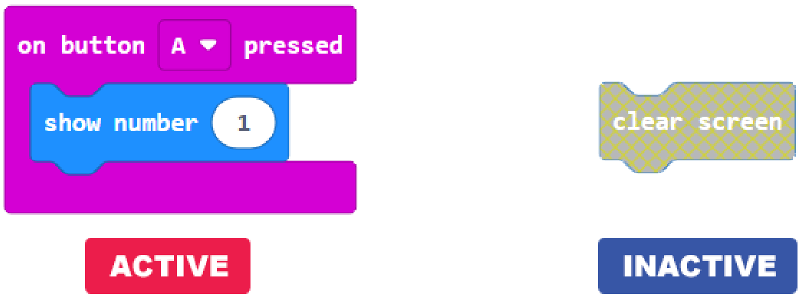


Figure 18: Examples of active and inactive scripts/codes. The clear screen command will not be executed

To delete a block or an entire script you can either right click on the block, and select “delete block(s)” from the floating menu (Figure 19 a.), or hold click on the block (or the script), and drag it on the left, until the purple trash bin appears (Figure 19b.)

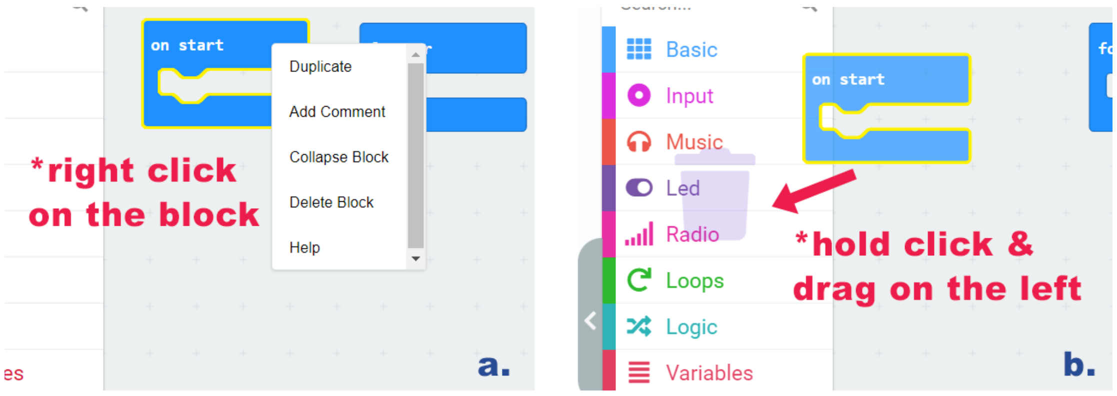


Figure 19: Deleting a command block

3.5 Microsoft Makecode environment: downloading and testing a project or a code/script

To download a project or a code/script to the micro:bit you need to connect the micro:bit board to your computer using a USB cable (Figure 20) and then click the “download” button [Figure 5, **(4)**] in the Makecode software.

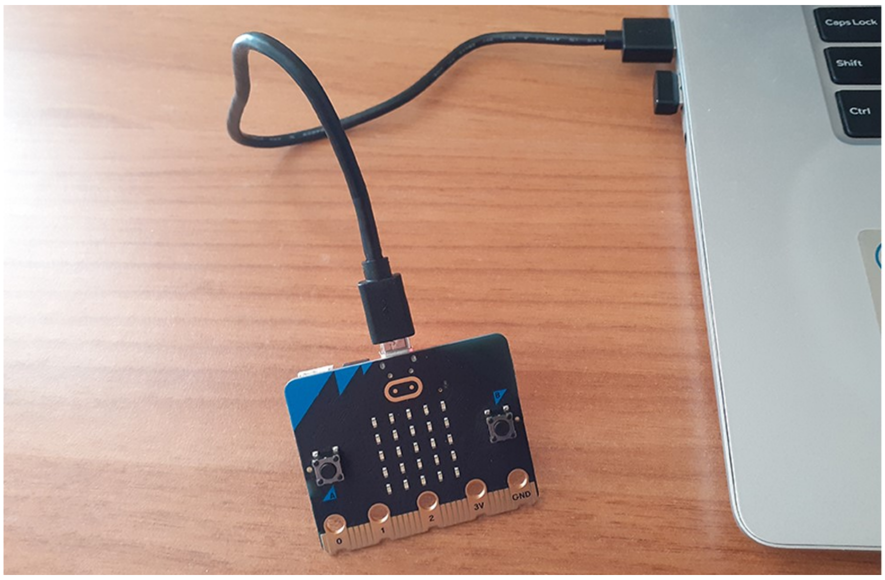
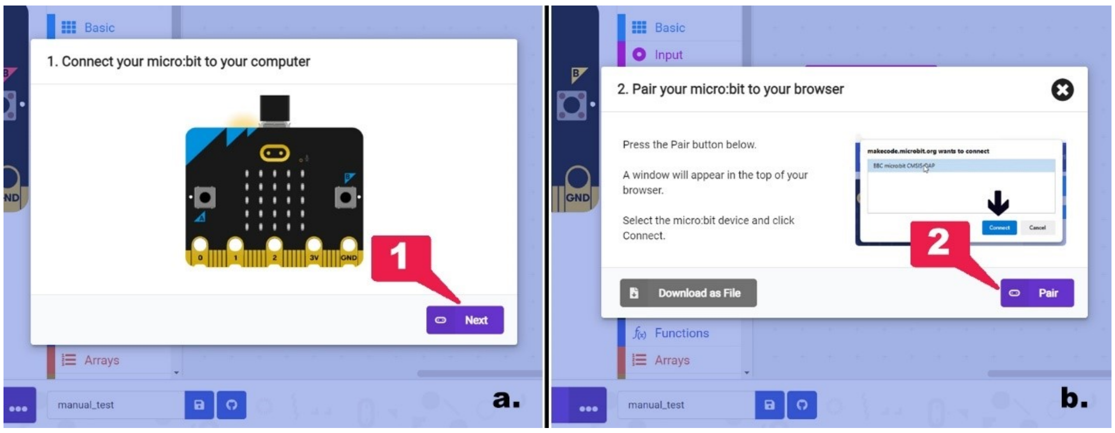


Figure 20: Connecting the micro:bit board to a computer using a USB cable

If you forget to connect your micro:bit or if the script does not download immediately, Makecode provides step-by-step instructions on how to pair the micro:bit with the Makecode environment (Figure 21a, 21b).


1. Connect your micro:bit to your computer

2. Pair your micro:bit to your browser

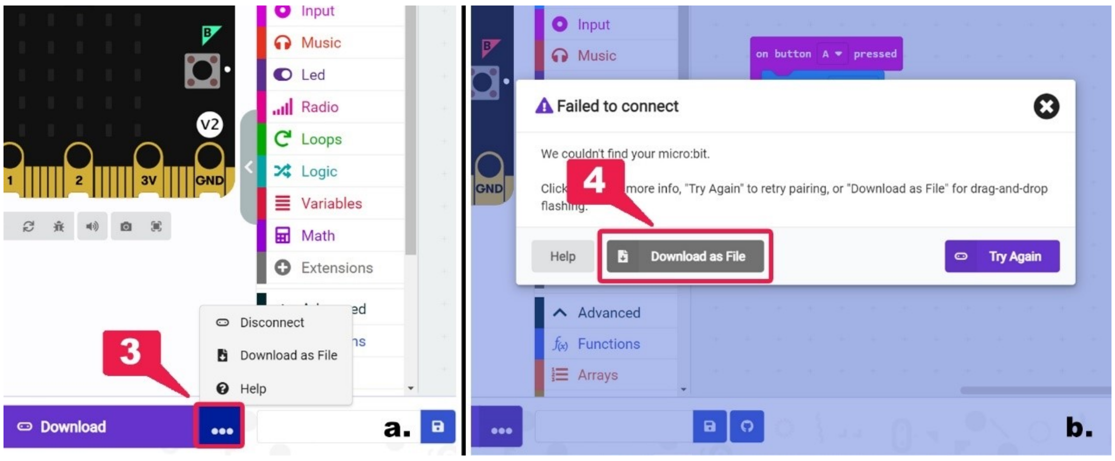
Press the Pair button below.

A window will appear in the top of your browser.

Select the micro:bit device and click Connect.

Figure 21: a. Step to connect your micro:bit to your computer; b. Instructions on how to pair your micro:bit to your browser.

Alternatively, if the micro:bit fails to connect, you can download the script locally to your computer as a .hex file by clicking on the '...' button (3) or clicking on the 'Download as File' button (4) (Figure 22).



We couldn't find your micro:bit.

Click … more info, "Try Again" to retry pairing, or "Download as File" for drag-and-drop flashing.

Figure 22: Downloading the project as a .hex file

To test your project or code/script, you can either use the micro:bit simulator (Figure 23a), or download it to your micro:bit board (Figure 23b).

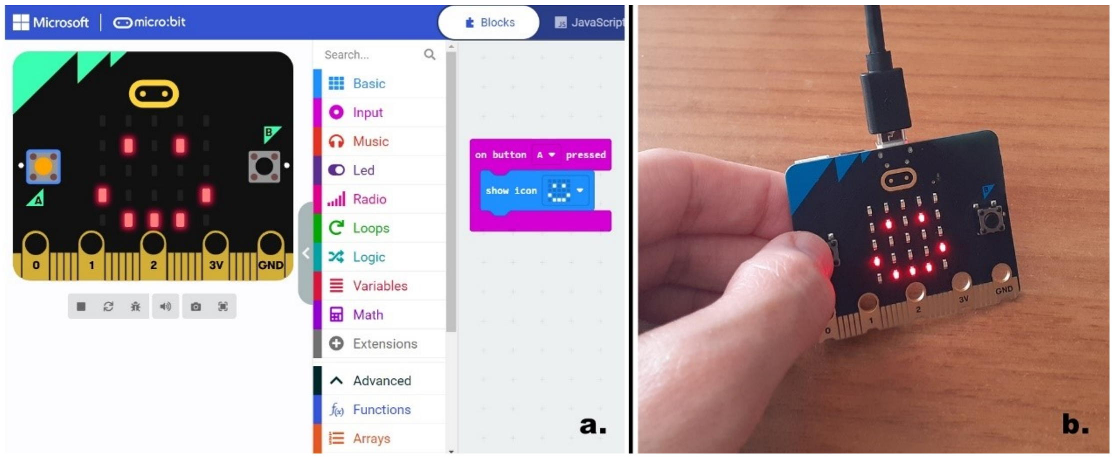


Figure 23: a. Testing the project to the simulator; b. Testing the project to the micro:bit

4. How to use micro:bit for introducing IoT

4.1 Bluetooth Radio communication

As mentioned, the BBC micro:bit board has a very low power Bluetooth Radio antenna [Figure 4, **(12)**] that allows the micro:bit to communicate wirelessly with other micro:bits or other devices. This Bluetooth is the same radio that a smartphone uses to communicate with other wireless devices (such as handsfrees or smartwatches). This feature is valuable not only for introducing IoT to your students, but also for creating a range of IoT projects.

Currently, micro:bit can only be programmed to send or receive text messages or/and numeric values to other devices. This can be done through two main command groups: **a.** the Radio command group and **b.** the Bluetooth command group.

Here are some of the key command blocks from both groups:

4.1.a. Radio command group

By using the blocks included in the Radio command group you can introduce your students to IoT and the idea of data communication between two or more devices. Click on the Radio command group to explore the available command blocks (Figure 24).

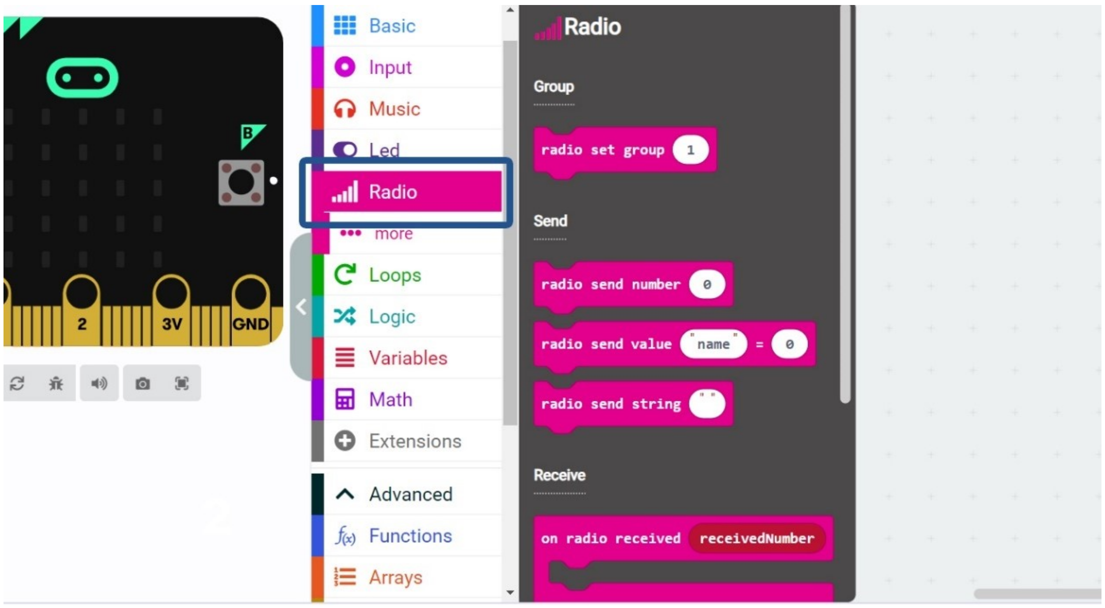


Figure 24: Clicking on the Radio command group

On the main tab of Radio group there are available 8 command blocks, while on the "…more" tab, there are 4 more command blocks.

The **hat blocks** that are mostly used in projects are the OnReceivedNumber and OnReceivedString (Figure 25)**.** These hat blocks will execute part of a script/code when the micro:bit **receives** a **number** or a **string** over radio.

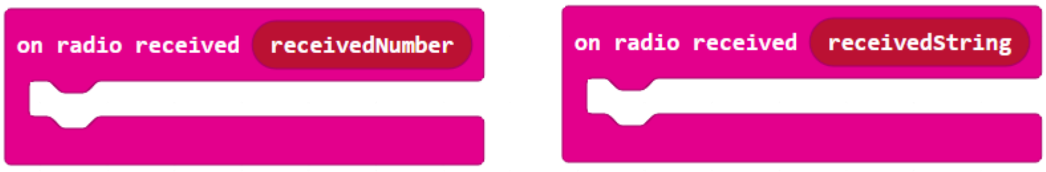

on radio received receivedNumber
on radio received receivedString

Figure 25: OnReceivedNumber and OnReceivedString hat blocks

The stuck block that should be used in all projects is the "**radio set group …**" command block (Figure 26). This stuck block is essential as it sets the channel (or ID) of the radio. Micro:bits on the same channel can "talk" to each other and exchange information. If you don't set an ID, the micro:bit will randomly choose a channel, which is likely to cause your project to fail. Channel/ID numbering ranges from 1 to 255.

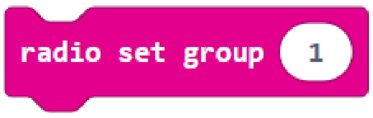


Figure 26: The radio set group command block

Other stuck blocks often utilized in projects are the "**radio send number …**" and the "**radio send string "…"**" command blocks (Figure 27), which are used to broadcast numbers and strings respectively to other micro:bits connected to the same id/channel.

Note: keep in mind that a string can be up to 19 characters long

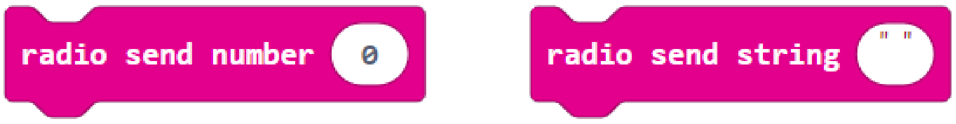


Figure 27: The radio send number and radio send string command blocks

Another interesting stuck block (located on the "…more" tab) is the "**radio set transmit power …**" command block (Figure 28). This block determines how weak or how strong is the radio signal of a micro:bit, on a scale from 0 to 7. You can use this block to raise discussion in the classroom about signal strength and how it relates to the distance over which two (or more) devices can communicate.

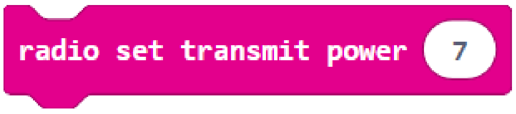


Figure 28: The radio set transmit power command block

4.1.b. Bluetooth command group

To create projects where micro:bit exchanges data with other Bluetooth devices, you need to use the blocks contained in the Bluetooth command group, offering additional services. This group is not included in the main command groups, and should be imported via the "extensions" group. Therefore, click on the "Extensions", and type "Bluetooth" in the search bar **(1)** (Figure 29). Then select the first result returned **(2)** (Figure 29).

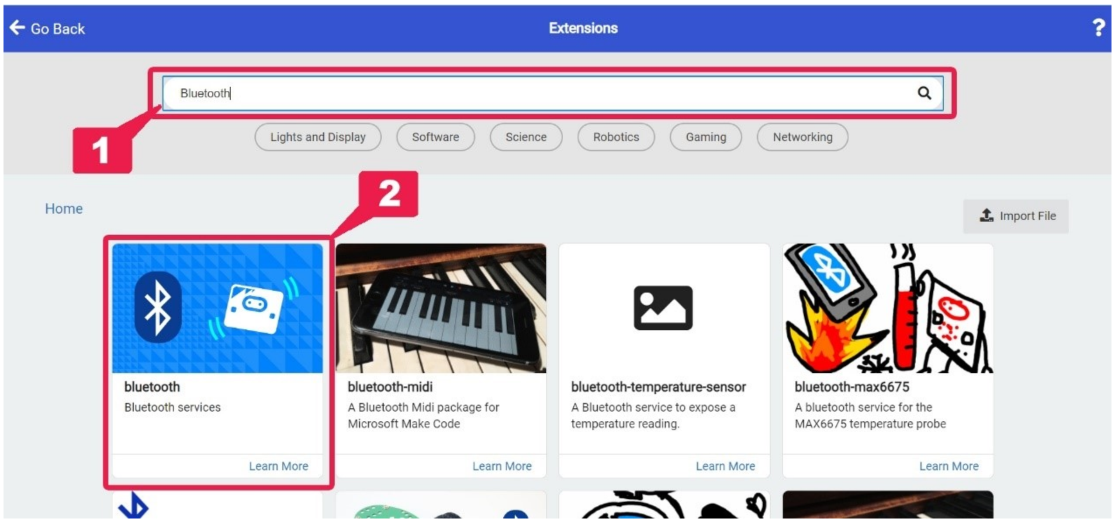
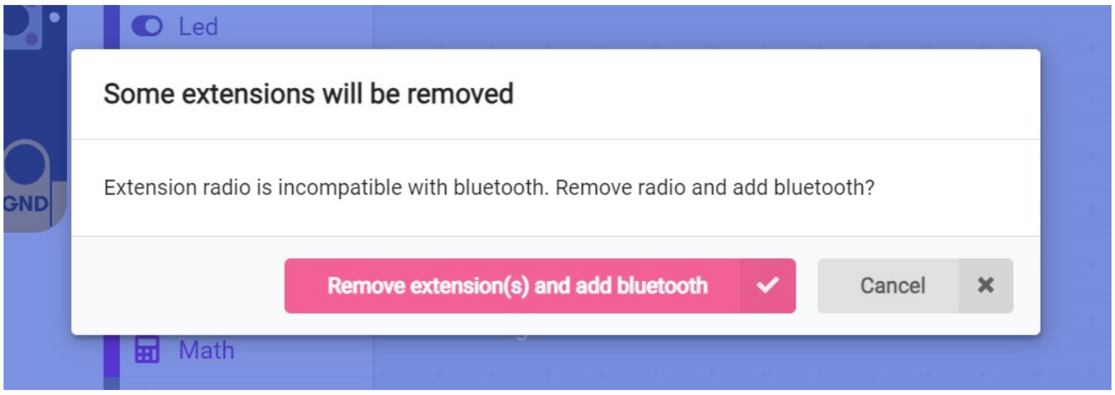


Figure 29: Finding Bluetooth extension

You will receive a message informing you that radio is incompatible with Bluetooth and should be removed. Click on the "Remove extension(s) and add Bluetooth" (Figure 30), to load the Bluetooth command group into your project.


Some extensions will be removed

Extension radio is incompatible with bluetooth. Remove radio and add bluetooth?

Figure 30: Warning for removing radio extension

Click on the Bluetooth command group to take a look on the available command blocks (Figure 31).

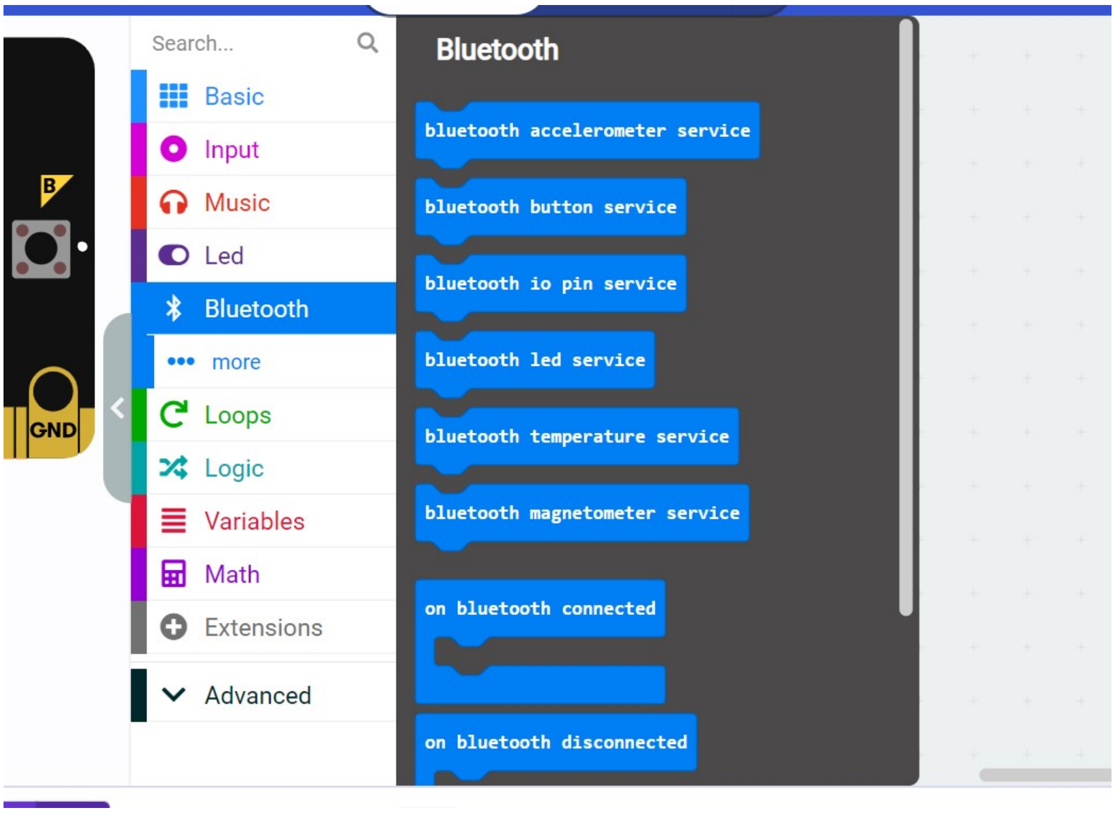


Figure 31: Exploring the available Bluetooth command blocks

Two of the most useful **hat blocks** are the **OnBluetoothConnected** and **OnBluetoothDisconnected** (Figure 32)**.** The former will execute part of a script/code when the micro:bit is connected to another device via Bluetooth. The latter will execute an event handler when a micro:bit connected to another device via Bluetooth is disconnected.

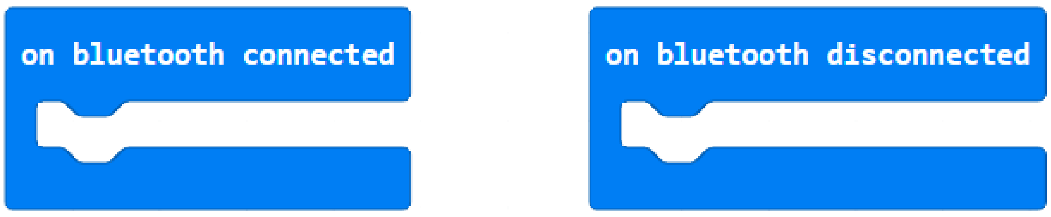


Figure 32: The "on Bluetooth connected" and "on Bluetooth disconnected" hat blocks

Another important hat block is "**Bluetooth on data received …**" (Figure 33). When a delimiter is matched in the received data, this hat block executes part of the code. Delimiters are characters in a received data string that split the string into smaller strings to form separate data items. Some delimiters are commas (,), hashtags (#), colons (:), and semicolons (;).

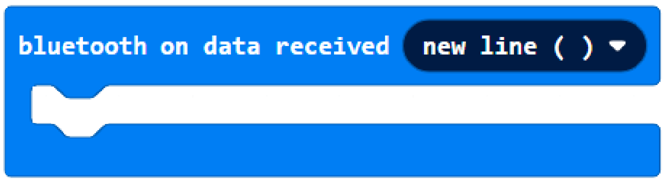

bluetooth on data received new line ( )

Figure 33: The Bluetooth on data received … block

The stuck block that should be used in all projects is the "**Bluetooth UART service**" command block (Figure 34). This stuck block is essential as it allows another device like a smartphone to exchange any data with the micro:bit. UART stands for Universal Asynchronous Receiver Transmitter and is one way to establish serial data communications.

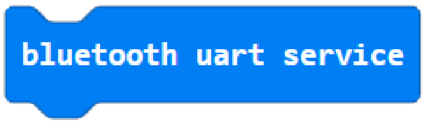
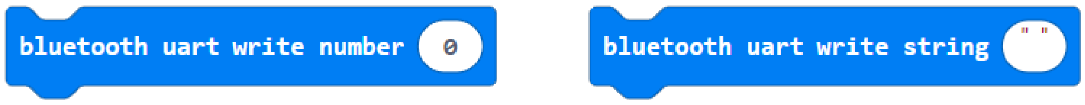

bluetooth uart service

Figure 34: The Bluetooth uart service block

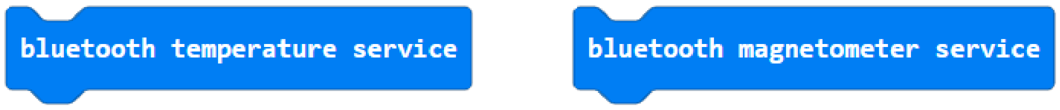
The "**Bluetooth uart write number**" and "**Bluetooth uart write string**" command blocks (Figure 35) allow another Bluetooth device to exchange numeric or string data with the micro:bit, only if the uart service is running.


bluetooth uart write number 0

bluetooth uart write string " "

Figure 35: The "Bluetooth uart write number" and "Bluetooth uart write string" command blocks

Various other services are available, such as temperature and magnetometer (Figure 36), which allow the micro:bit to share data on these specific parameters with other Bluetooth devices.


bluetooth temperature service

bluetooth magnetometer service

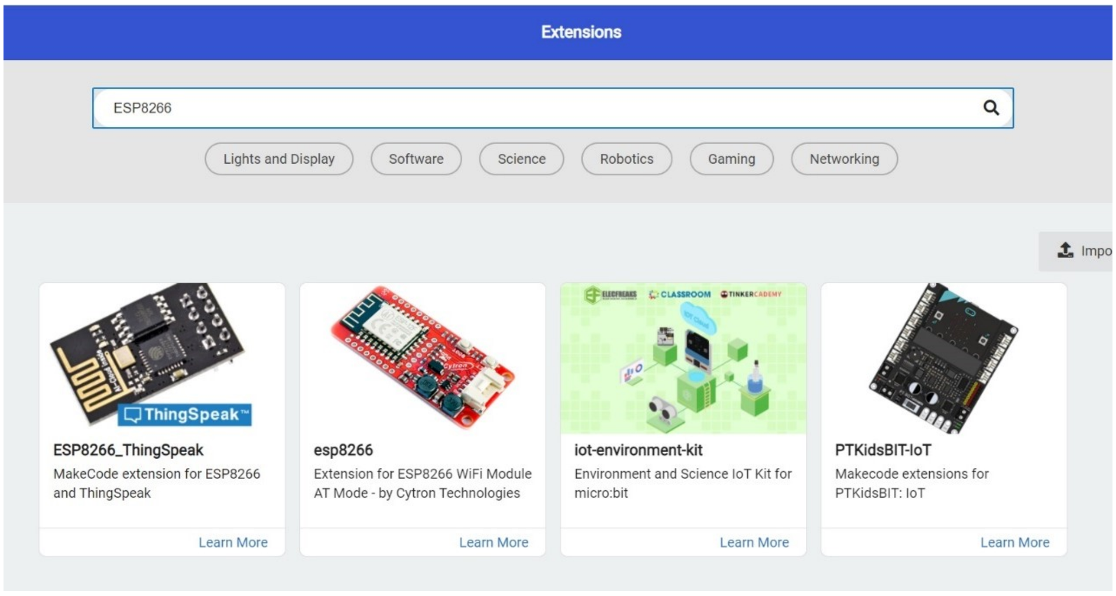
Figure 36: The "Bluetooth temperature service" and the "Bluetooth magnetometer service" command blocks

Note: Visit this link to (https://makecode.microbit.org/reference/bluetooth) to find out more about all the available Bluetooth command blocks

4.2 WiFi communication via ESP8266 WiFi module

Apart from the Bluetooth radio, micro:bit can exchange data with other devices over WiFi using an ESP8266 WiFi module. Through ESP8266, micro:bit can communicate with the Internet, send and receive data, and interact with web-based services and APIs.

ESP8266 can be found as a separate module or integrated into breakout edge connectors such as the IoT Bit. To program ESP8266, you need to search for the appropriate extension in the Extensions menu (Figure 37).



ESP8266_ThingSpeak
MakeCode extension for ESP8266 and ThingSpeak
Learn More

esp8266
Extension for ESP8266 WiFi Module AT Mode - by Cytron Technologies
Learn More

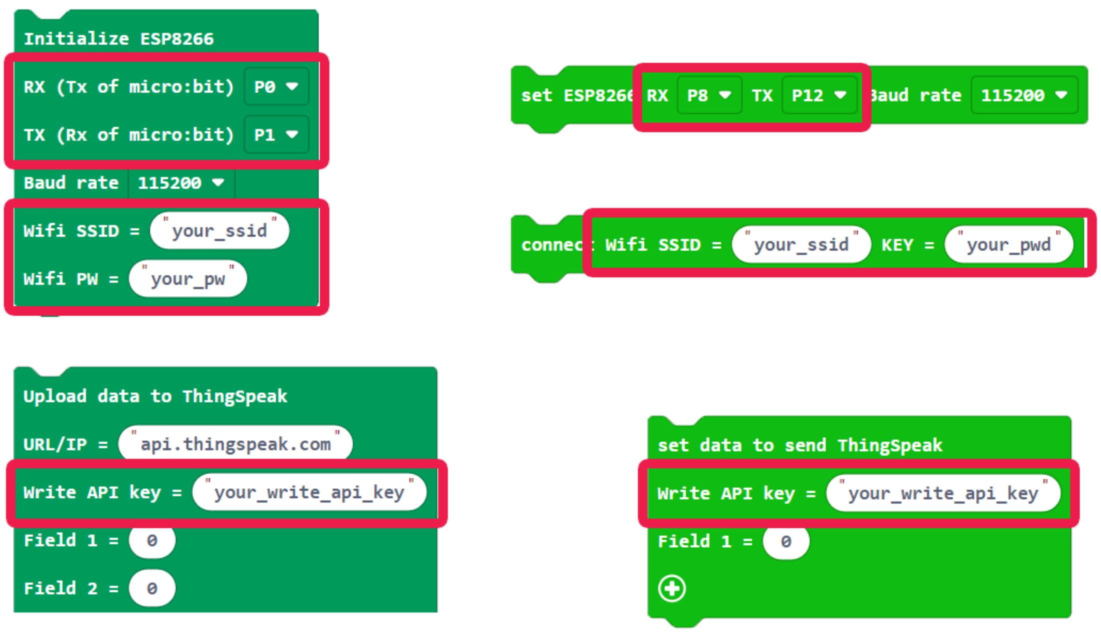
iot-environment-kit
Environment and Science IoT Kit for micro:bit
Learn More

PTKidsBIT-IoT
Makecode extensions for PTKidsBIT: IoT
Learn More

Figure 37: Different extensions for programming ESP8266

Depending on the extension selected, the available command blocks may differ. In all cases, however, there are some common fields that need to be filled in. These are the “Wifi SSID”, which is the name of your wifi connection, the “Wifi PW”, which is the password for your internet connection, and the “Write API key”, which can be found in the channel created in the ThingSpeak platform[2] (Figure 38). You will also need to set the RX and TX pins, to allow serial communication between the micro:bit and the WiFi module (Figure 38).

[2] Which will be further explained in the IoT projects



```
Initialize ESP8266
RX (Tx of micro:bit) P0
TX (Rx of micro:bit) P1
Baud rate 115200
Wifi SSID = "your_ssid"
Wifi PW = "your_pw"
```

```
set ESP8266 RX P8 TX P12 Baud rate 115200
```

```
connect Wifi SSID = "your_ssid" KEY = "your_pwd"
```

```
Upload data to ThingSpeak
URL/IP = "api.thingspeak.com"
Write API key = "your_write_api_key"
Field 1 = 0
Field 2 = 0
```

```
set data to send ThingSpeak
Write API key = "your_write_api_key"
Field 1 = 0
```

Figure 38: Indicative blocks located in different extensions, and the common fields required

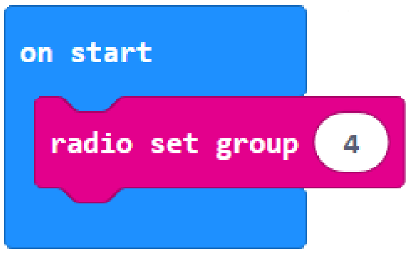
5. Indicative activities to become familiar with micro:bit and radio communication

This section includes a number of indicative activities to become familiar with micro:bit and radio communication.

5.1 Sending a smiley face

The purpose of this activity is to send a smiley face to another microbit, when button A is pressed.

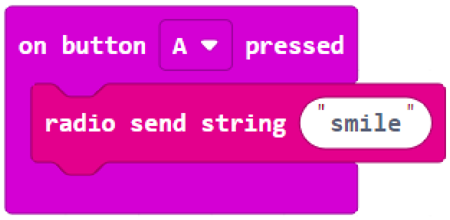
The first step is to set the channel (or the ID) on which your micro:bit will broadcast. From the "**Basic**" command group, drag the "**on start**" hat block into the code assembly area. Then, from the "**Radio**" command group, drag the "**radio set group…**" command, and place it inside the "**on start**" hat block. Set the channel to any number you like between 1 and 255 (4 in the example below).



```
on start
    radio set group 4
```

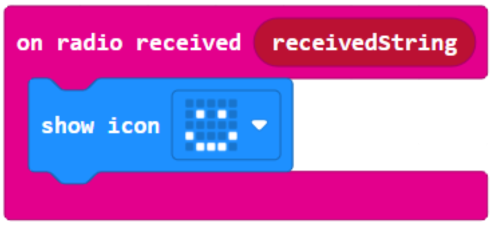
The next step is to program button A to send a message when it is pressed. From the "**Input**" command group drag the "**on button A pressed**" hat block into the code assembly area. Then, from the "**Radio**" command group, drag the "**radio send string…**" command, and place it inside the "**on button A pressed**" hat block. Then write something meaningful such as "smile".

This part of the code will allow you to send the message "smile", when the button A is pressed.



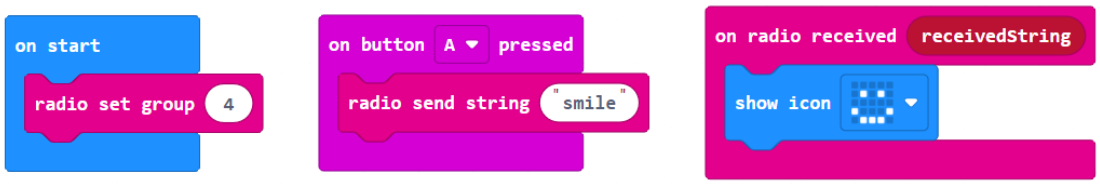
```
on button A ▾ pressed
    radio send string "smile"
```

The last step is to program what will happen when another micro:bit receives the message "smile". From the "**Radio**" command group, drag the "**on radio received receivedString**" hat block into the code assembly area. Then, from the "**Basic**" command group, drag the "**show icon…**" command, and place it inside the "**on radio received…**" hat block. Then from the floating menu choose the smiley face icon.



```
on radio received receivedString
    show icon
```

Download the entire code (Figure 39) to **two** micro:bits and observe the result.



```
on start
    radio set group 4

on button A pressed
    radio send string "smile"

on radio received receivedString
    show icon
```

Figure 39: The entire code of the 1st activity

To test the code you can either use two micro:bits (Figure 40) or the Makecode simulator (Figure 41).

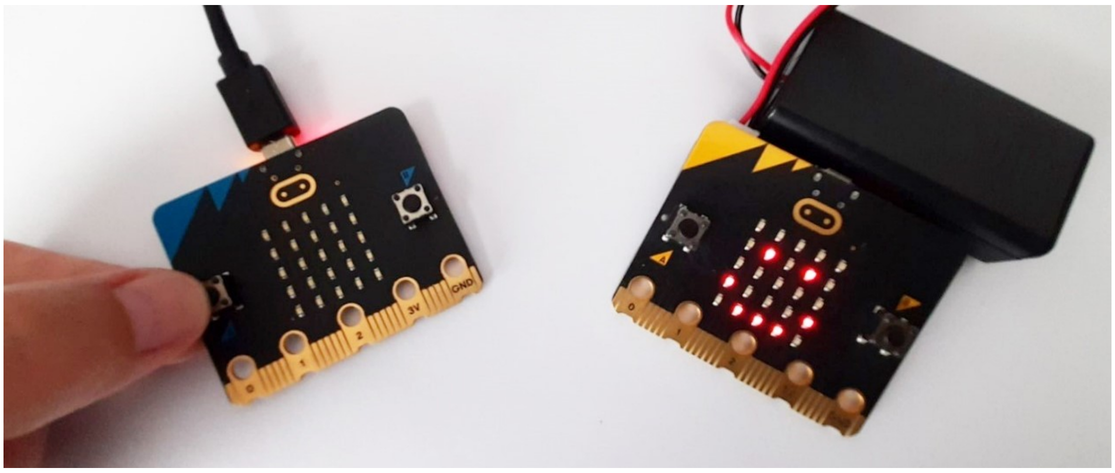
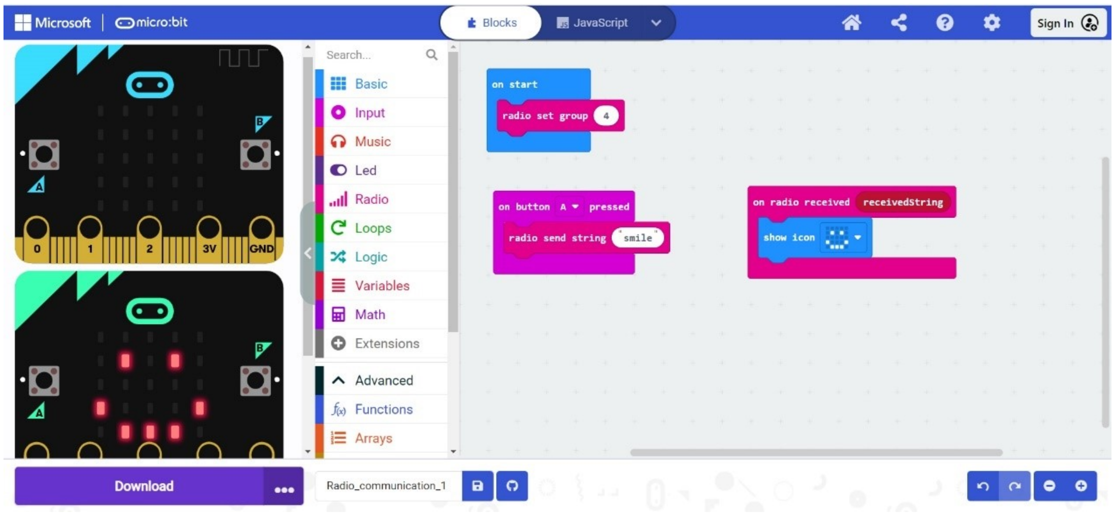


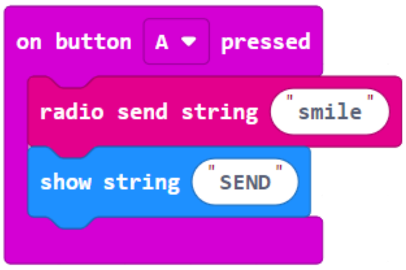
Figure 40: Sending a smiley face to the yellow microbit by pressing the button A on the blue microbit



```
on start
    radio set group 4

on button A ▾ pressed
    radio send string "smile"

on radio received receivedString
    show icon
```

Figure 41: Testing the code on the Makecode simulator

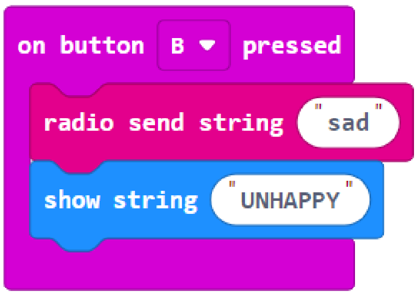
Tip: You can also add a "**show string...**" command inside the "on button A pressed" hat block to make the communication more lively/active. In this case the micro:bit sending the message will display the inserted string (SEND in the example), making the interaction more intuitive.



```
on button A ▾ pressed
    radio send string "smile"
    show string "SEND"
```

5.2 Sending a smiley and a sad face

This activity is an extended version of the 1st activity. The purpose of this second activity is to build on the previous code, and program button B to send a sad face when pressed. Add another "**on button A pressed**" hat block, and select "**B**" from its floating menu. Inside this new hat block add a "**radio send string…**" command, and write something meaningful, such as "sad". Also, add a "**show string…**" command, and type in "unhappy".



The next step is to program the micro:bit that receives the messages to display a smiley face when it receives the "smile" message and a sad face when it receives the "sad" message. To do this, we need to use two "**if..then**" commands from the "**Logic**" command group, and two string comparison operators (Figure 42), and place them inside the "**on radio received receivedString**" hat block (Figure 43).

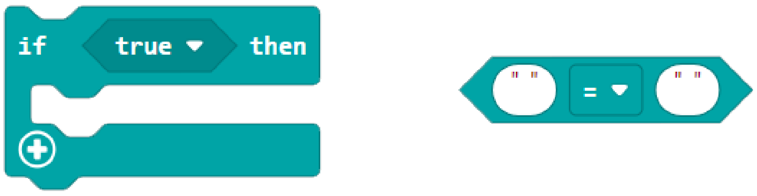


Figure 42: The "if …then" logic command and the string comparison operator

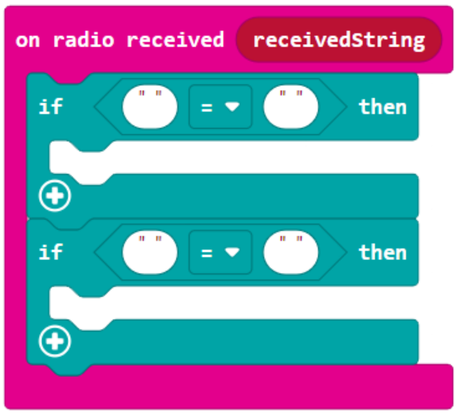
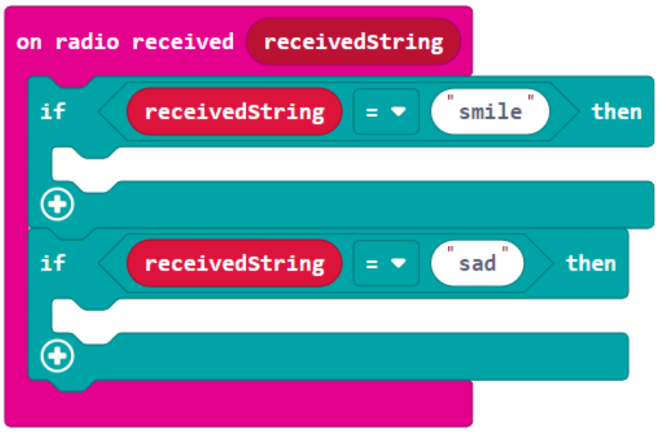


Figure 43: The "on radio received…" hat block, after adding the two "if…then" logic commands

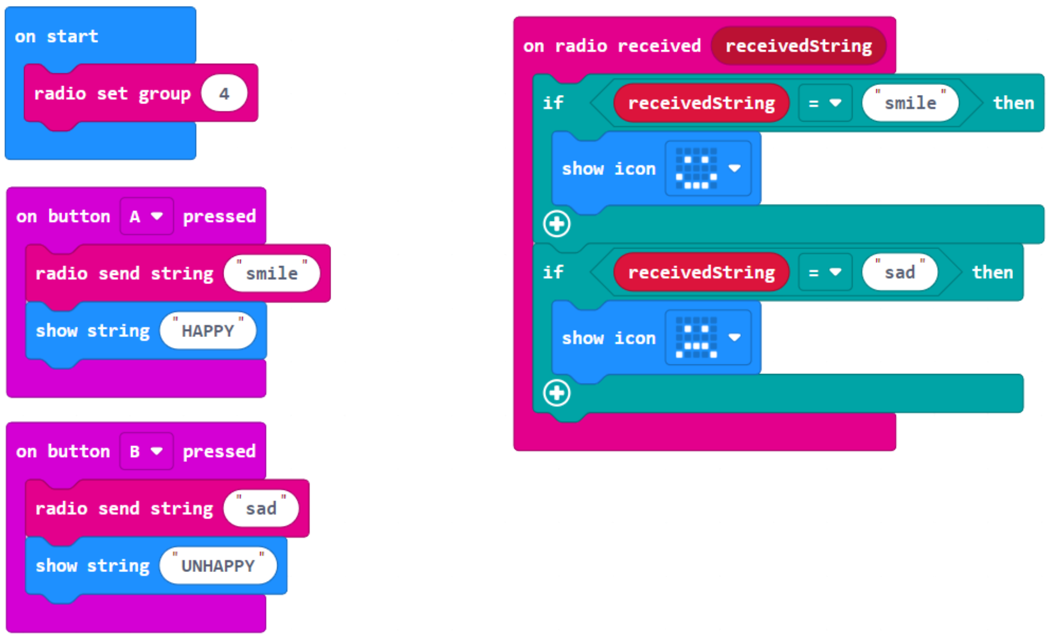
Based on the message received (smile or sad), you must program each of these "**if...then**" commands to display the corresponding icon (a smiley or a sad face). Inside each string comparison operator, you need to add a "**receivedString**" command to the left part of the equation, and type the string "smile" or "sad" to the right part of the equation.



```
on radio received receivedString
    if receivedString = "smile" then
    
    if receivedString = "sad" then
    
```

Finally, inside each "**if…then**" logic command you have to place the corresponding "show icon" command.

After this final step, the entire code should look like the one shown in Figure 44.



```
on start
    radio set group 4

on button A pressed
    radio send string "smile"
    show string "HAPPY"

on button B pressed
    radio send string "sad"
    show string "UNHAPPY"

on radio received receivedString
    if receivedString = "smile" then
        show icon
    if receivedString = "sad" then
        show icon
```

Figure 44:The entire code of the second activity

Download the code to two micro:bits, or use the Makecode simulator and observe the results.

5.3 Sending temperature data to another micro:bit

The purpose of this activity is to show how you can send temperature data to another micro:bit, when button A is pressed.

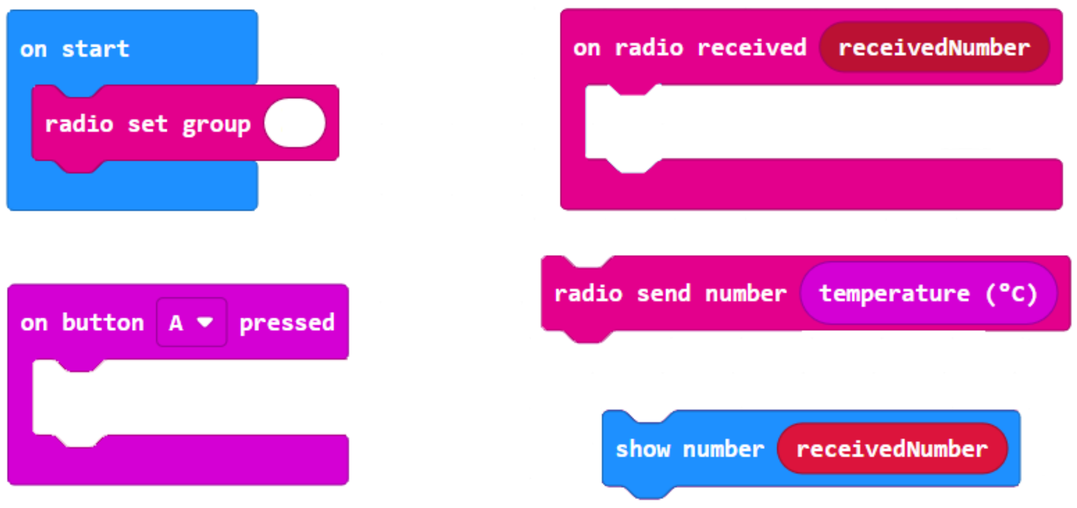
First you need to set the channel (or the ID) on which your micro:bit will broadcast.

You will then need to program button A to send temperature values using the radio.

Finally, another micro:bit must be programmed to display the temperature when it receives the corresponding value.

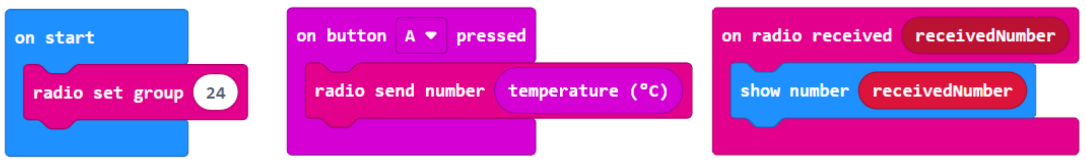
With this information in mind, try to assemble the following semi-structured code.

Tip: the “**temperature**” command block is located in the “**Input**” command group.



```
on start
    radio set group ( )

on radio received receivedNumber

on button A ▾ pressed

radio send number temperature (°C)

show number receivedNumber
```

Test the code (which should look like the one shown in Figure 45) by downloading it to two micro:bits, or by using the Makecode simulator.



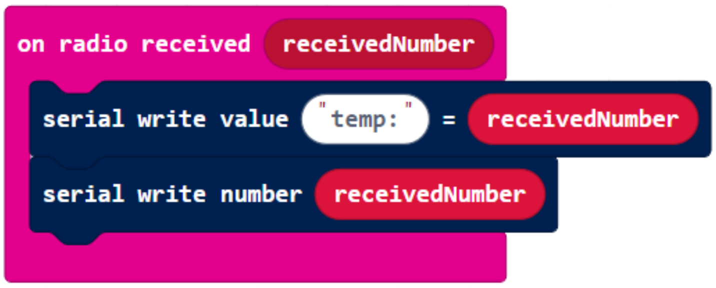
```
on start
    radio set group 24

on button A ▾ pressed
    radio send number temperature (°C)

on radio received receivedNumber
    show number receivedNumber
```

Figure 45: The entire code of the third activity

Displaying data on Makecode's serial port

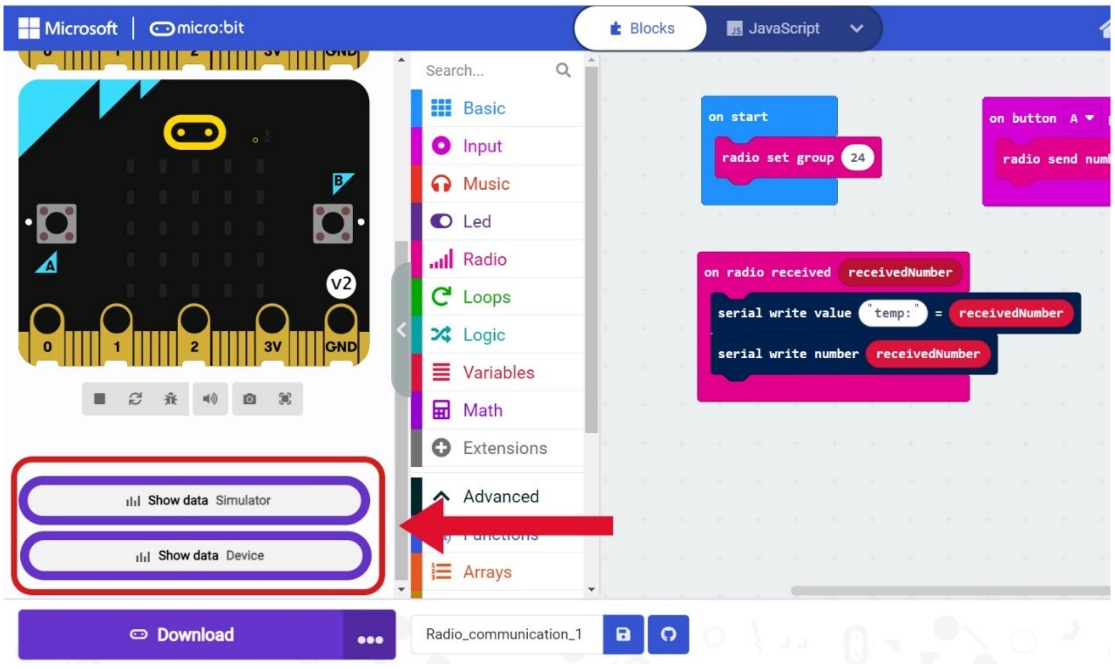
In addition, you can monitor the recorded data on Makecode’s serial port, by using some block commands from the “**Serial**” command group.

Inside the “**on radio received receivedNumber**” hat block, place the “**serial value x = …**” and the “**serial write numbers …**” command blocks. Then type "temp" in the x field and place “**receivedNumber**” in the other two fields.



```
on radio received receivedNumber
    serial write value "temp:" = receivedNumber
    serial write number receivedNumber
```

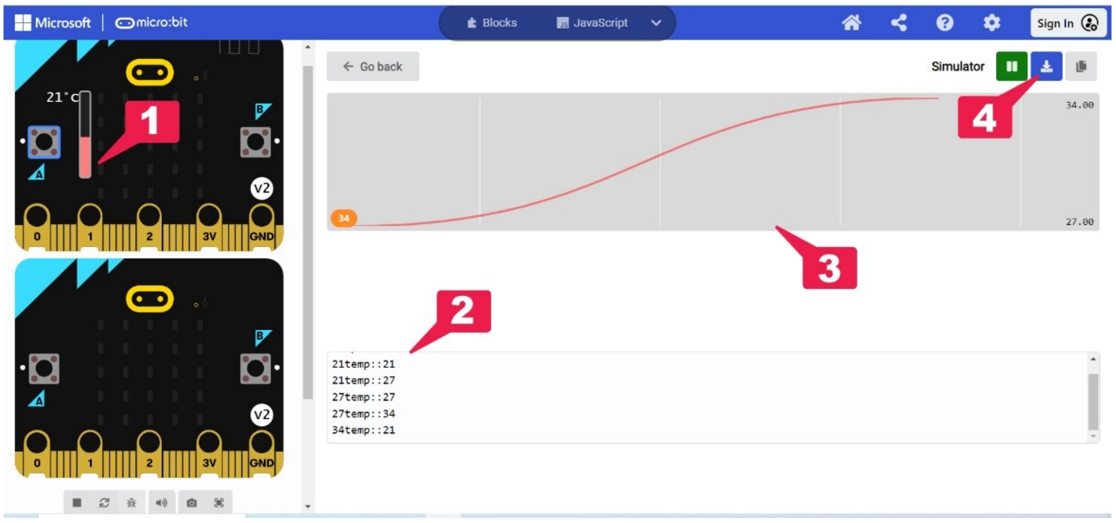
Test the new code by downloading it to two micro:bits, or by using the simulator. You can monitor the received data by selecting either “Show data device” or “Show data simulator” (Figure 46).



```
on start
    radio set group 24

on button A
    radio send num

on radio received receivedNumber
    serial write value "temp:" = receivedNumber
    serial write number receivedNumber
```

Figure 46: Selecting the data you want to monitor

Tip: Select “Show data simulator”, and change the temperature to the simulator’s thermometer (1) to observe how different data can be monitored (2) and displayed (3) over time in the serial monitor (Figure 47). You can also click on the download button (4), to download the recorded data as a spreadsheet.



```
21temp::21
21temp::27
27temp::27
27temp::34
34temp::21
```

Figure 47: Makecode's serial monitor