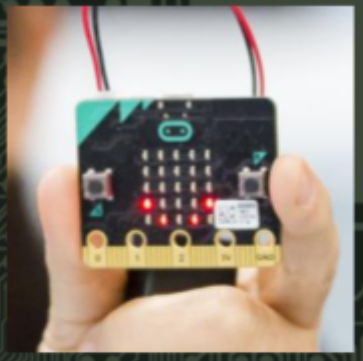
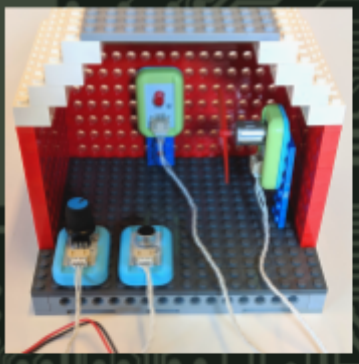
Physical tech from GO to WHOA!

BBC micro:bit & Boson

for the Australian
Digital Technologies Curriculum

VISUAL CODE EDITION

written by
Nathan Alison
Sanjin Dedić

Digital Learning and Teaching Victoria

DFROBOT
DRIVE THE FUTURE

Introduction

This series of activities introduces Physical Tech by building on coding skills from the Australian Digital Technologies curriculum at Year Levels 5-10.

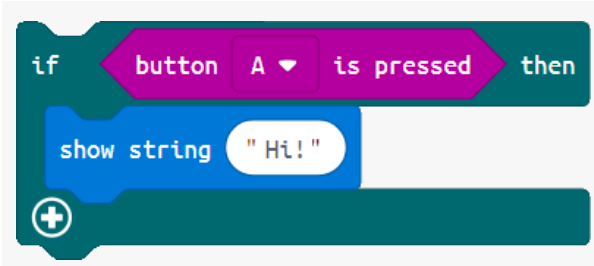
Start with the **BBC micro:bit** only to revise skills in algorithms and coding. Then, use modules from the **Boson** range to build physical tech designs for real-world interactions and data gathering.

Each activity includes TINKER extension tasks and JUMP OFF ideas for freeform projects to allow opportunities for design thinking.

Visual Code edition

This edition presents all code in **visual** (blocks).

See the links below for parallel documents.




```
if (input.buttonIsPressed(Button.A)) {
    basic.showString("Hi!");
}
```

```
if button_a.is_pressed():
    display.scroll('Hi!')
```

	Year levels		
	5 - 6	7 - 8	9 - 10
visual (blocks) this document.	✓	✓	
JavaScript parallel document.		✓	✓
Python parallel document.		✓	✓

Contents

IMPORTANT: Prior knowledge and skills

Although no prior experience with BBC micro:bit or Boson modules is required, the activities in this resource do not constitute an introduction to coding. These activities will revise coding skills already introduced and practiced, then make algorithms to apply to Physical Tech solutions.

Students should have already seen these concepts and skills:

- Branching (also called Conditionals): These are **if**/**else** structures used for decision making.
- Iteration (also called Loops): These are **while**, **for** or **forever** structures used to repeat code.
- Variables: Used to store values (eg. **name = 'Bob'**) and use later (eg. **print(name)**).

Recommended courses

These courses are recommended for introducing coding in a classroom setting, with the active involvement of a teacher. Having students copy out full, completed programs is not recommended as an effective strategy for teaching coding.

For **visual (blocks) code**:

- ACA Digital Technologies Challenges at Grok Learning (free for Years 3-8):
    - DT Challenge Blockly - Chatbot
    - DT Challenge Blockly - Turtle
    - DT Challenge Blockly - Space Invaders
- code.org Computer Science Fundamentals Course E and Course F (free).

For **JavaScript**:

- code.org Computer Science Discoveries Unit 3 (free)
- Hour of Code JavaScript app with CodeHS (free)
- JavaScript Hour of Code or complete course at CodeCraft (free)
- Khan Academy Computer Programming course (free)
- Codecademy Introduction to JavaScript (free)

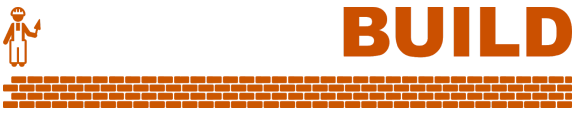
For **Python**:

- ACA Digital Technologies Challenges at Grok Learning (free for Years 3-8):
    - DT Challenge Python - Chatbot
    - DT Challenge Python - Turtle
- Python Fundamentals by Sanjin Dedic
- Python Hour of Code or complete course at CodeCraft (free)

Using this document

Activity structure

Each activity incorporates 3 sections:



A structured activity to build a skill, with accompanying theory and examples.

Challenges for students to practice their skills.

Solutions in APPENDIX A.

Ideas for students to pursue their own design projects with the skills acquired so far.

Warnings and notes

	Look out for **important notices** or **safety precautions** like this.
	This indicates an **enquiry** opportunity for students to discuss and figure out a solution.
	This gives **key knowledge** and points to **further resources**.

Coding languages

This document presents code in **visual (blocks) code** only, from ACTIVITY 1.2 onwards.

All code has been carefully designed so that you can also choose to teach entirely in one of the other languages with a parallel document (JavaScript or Python). A single-language approach is recommended. Flipping back and forth from visual mode will produce less readable code.

Note, the visual code environment has some limitations, particularly in the area of *functions*. Occasionally, an advanced activity will not be available as visual code.

See ACTIVITY 1.1 for advice on choosing a language.

Curriculum alignment

Digital Technologies (Australian Curriculum)

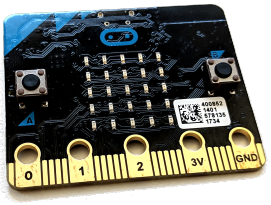
Processes and Production Skills

	Year levels 5-6	Year levels 7-8	Year levels 9-10
Working with data	Acquire, store and validate different types of data, and use a range of software to interpret and visualise data to create information (ACTDIP016)	Acquire data from a range of sources and evaluate authenticity, accuracy and timeliness (ACTDIP025)	Develop techniques for acquiring, storing and validating quantitative and qualitative data from a range of sources, considering privacy and security requirements (ACTDIP036)
		Analyse and visualise data using a range of software to create information, and use structured data to model objects or events (ACTDIP026)	Analyse and visualise data to create information and address complex problems, and model processes, entities and their relationships using structured data (ACTDIP037)
Defining the problem	Define problems in terms of data and functional requirements drawing on previously solved problems (ACTDIP017)	Define and decompose real-world problems taking into account functional requirements and economic, environmental, social, technical and usability constraints (ACTDIP027)	Define and decompose real-world problems precisely, taking into account functional and non-functional requirements and including interviewing stakeholders to identify needs (ACTDIP038)
Designing the solution	Design a user interface for a digital system (ACTDIP018)	Design the user experience of a digital system, generating, evaluating and communicating alternative designs (ACTDIP028)	Design the user experience of a digital system by evaluating alternative designs against criteria including functionality, accessibility, usability, and aesthetics (ACTDIP039)
	Design, modify and follow simple algorithms involving sequences of steps, branching, and iteration (repetition) (ACTDIP019)	Design algorithms represented diagrammatically and in English, and trace algorithms to predict output for a given input and to identify errors (ACTDIP029)	Design algorithms represented diagrammatically and in structured English and validate algorithms and programs through tracing and test cases (ACTDIP040)
Implementation	Implement digital solutions as simple visual programs involving branching, iteration (repetition), and user input (ACTDIP020)	Implement and modify programs with user interfaces involving branching, iteration and functions in a general-purpose programming language (ACTDIP030)	Implement modular programs, applying selected algorithms and data structures including using an object-oriented programming language (ACTDIP041)
Evaluation and impact	Explain how student solutions and existing information systems are sustainable and meet current and future local community needs (ACTDIP021)	Evaluate how student solutions and existing information systems meet needs, are innovative, and take account of future risks and sustainability (ACTDIP031)	Evaluate critically how student solutions and existing information systems and policies, take account of future risks and sustainability and provide opportunities for innovation and enterprise (ACTDIP042)
Communication, planning and collaboration	Plan, create and communicate ideas and information, including collaboratively online, applying agreed ethical, social and technical protocols (ACTDIP022)	Plan and manage projects that create and communicate ideas and information collaboratively online, taking safety and social contexts into account (ACTDIP032)	Create interactive solutions for sharing ideas and information online, taking into account safety, social contexts & legal responsibilities (ACTDIP043)
			Plan and manage projects using an iterative and collaborative approach, identifying risks and considering safety and sustainability (ACTDIP044)

What you need

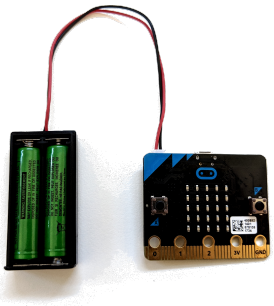
1 A class set (minimum 1 between 2 students) of BBC **micro:bit** devices.

Year 7-8 students may be familiar with this device from Primary Years. It might be used for other strands of the Digital Technologies curriculum in Years 7-8.



2 **Battery power** for the micro:bit, so that it can used away from the USB power provided by a computer.

The BBC micro:bit Go kit comes with a battery pack for two AAA batteries to be attached.



3 **Maker materials**.

For building physical tech solutions, we recommend:

- cardboard (with scissors and glue guns) *and/or*
- Lego or Lego-compatible parts

4 **Computers** or **laptops** with Internet access, to write code and transfer programs onto devices.

Note: micro:bit can also be coded from a mobile device or tablet, but this approach is not recommended at this time due to slow

5 **Boson Starter Kit for micro:bit** (for Modules 2 and 3).

To continue to Modules 2 and 3 of this course, this kit contains a selection of Boson electronics modules, as well as the micro:bit expansion board for Boson.

Boson Science Kit (for Module 3).

To continue to Module 3 of this course, this kit contains a selection of Boson electronics modules for scientific measurement and calibration.

MODULE 1

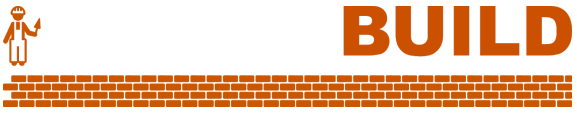
Coding skills with the BBC micro:bit.

ACTIVITY 1.1 - Hello... is it me you're looking for?

Goals

- Create and download your first program to the micro:bit.
- Use the scrolling display.
- Practice **iteration** (loops).

First program



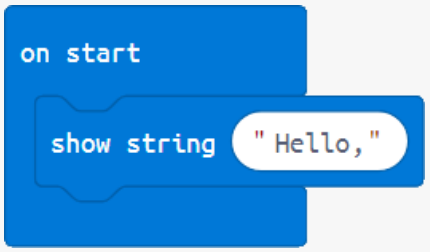
1. For **visual (block)** code or **JavaScript**, go to makecode.microbit.org in your web browser. For **Python**, go to python.microbit.org.

Which language should you choose? See the note Coding languages under Using this document.

Remember:

- Year Levels 5 - 6 are only required to do **visual (blocks)** code.
- **JavaScript** is a commonplace language on the Internet. Its syntax is similar to popular industry languages such as Java, C++, C#, as well as Arduino C. It qualifies as a General Purpose Language for Year 7-10 and beyond.
- **Python**'s readability and conciseness makes it an increasingly-popular language for learning coding. It is used in industry for quick prototyping and data analysis. It qualifies as a General Purpose Language for Year 7-10 and beyond.

2. Our first program will say hello to you. Replace *Name* with your own name.



Visual (blocks)

```
basic.showString("Hello, Name.");
```

JavaScript

```
from microbit import *
display.scroll('Hello, Name.')
```

Python

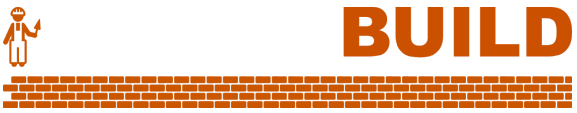
3. Connect the micro:bit via USB. It should appear to the computer like a USB disk / USB stick.
4. Click the **Download** button to get your .hex program file. Save or copy this file to the micro:bit, as if you were saving or copying to a USB stick.

   To show that the file is being sent to the micro:bit, an orange LED will blink quickly on the back.

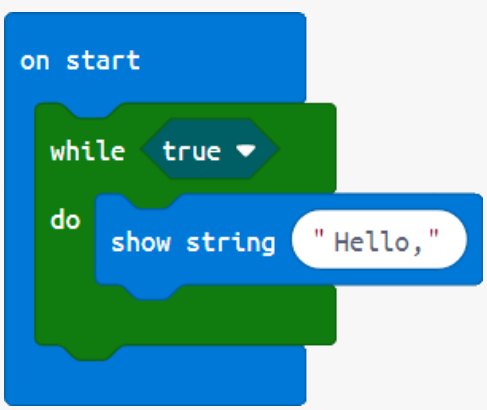
Most web browsers will automatically save files in a Downloads folder on your computer. To save time in the long run, you can instruct your browser to ask for a location each time you download a file.

- In *Chrome*, access Settings from the ⋮ menu, scroll down and click Advanced, then activate “Ask where to save each file before downloading”.
- In *Edge*, access Settings from the ··· menu, click “View advanced settings”, then activate “Ask me what to do with each download”.
- In *Firefox*, access Options from the ☰ menu, the choose “Always ask you where to save files”.
- In *Safari*, access Safari menu → Preferences, then under File Download Location (in the General tab), choose “Ask me where to save files”.

Say it again



1. Repeat the scrolling text forever using a simple while loop (iteration).



Visual (blocks)

```
while (true) {
    basic.showString("Hello,");
}
```

JavaScript

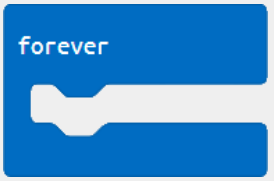
```
from microbit import *
while True:
  display.scroll('Hello,')
```

Python

2. Each time you modify your program, you must **Download** the new .hex program file and save it to the micro:bit.

We avoided using the **forever** block for our loop.

This resource avoids *event trigger* blocks, where possible. See APPENDIX C for more information.



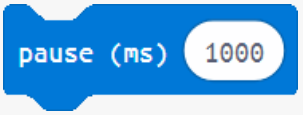
Looping with the micro:bit

TINKER

Try these extension tasks to practice iteration (see Solutions for complete programs):

A. Make a 4 second pause between each “Hello”.

Here’s the micro:bit command to pause for 1 second (1000 milliseconds):



Visual

```
basic.pause(1000);
```

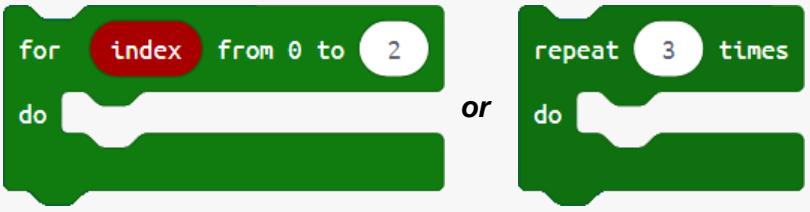
JavaScript

```
sleep(1000)
```

Python

B. Instead of looping forever, limit the loop to exactly 5 times.

Here’s the structure for looping 3 times:



Visual

```
for (let i = 0; i < 3; i++) {
    // Code to be repeated here.
}
```

JavaScript

```
for i in range(3):
  # Code to be repeated here.
```

Python

C. Now change the loop to happen 8 times.

D. Have the program show the index as it loops: “Hello 0”, “Hello 1”, “Hello 2”, etc.

Here’s the command to join two different variable types for display:

Visual

```
"text" + number
```

JavaScript

```
'text' + str(number)
```

Python

Fun with loops

Now that you can use the display and make code repeat, what other things could you make the micro:bit do? Here’s some ideas to get started:

- **Doomsday tag** - The micro:bit is a badge that counts down to 0, then displays a picture on the screen. Could you find a way to make it speed up as it approaches 0?
- **Pixel animation** - Use your loop index to move a pixel across or down the micro:bit display. How could you plot out a complete square?

Here’s the code for plotting and removing a pixel at position (2,2) on the display:


```
led.plot(2, 2);
```

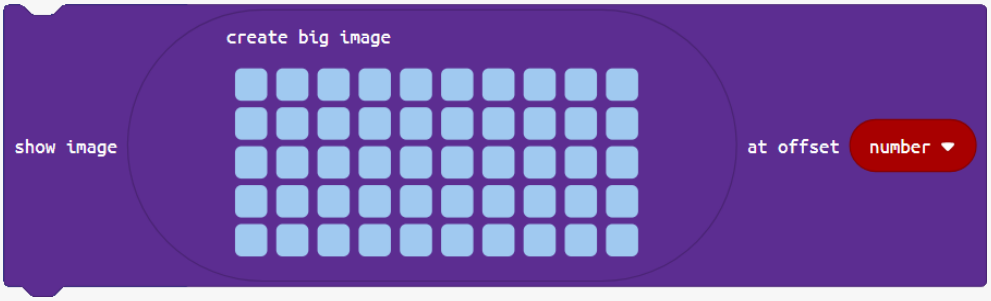
```
led.unplot(2, 2);
```

```
display.set_pixel(2, 2, 9)
```

```
display.set_pixel(2, 2, 0)
```

- **Big picture display** - Use your loop index to scroll a big picture across the micro:bit screen. Could you make it scroll back and forth?

Here's the code for showing a cityscape picture with an offset position:




```
images.createBigImage(`
    . . . . . # . . . .
    # # . . . # # . . .
    # # . # # # # . . #
    # # # # # # # # # #
    # # # # # # # # # #
    `).showImage(number);
```

```
display.show(Image('0000090000:'
                   '9900099000:'
                   '9909999009:'
                   '9999999999:'
                   '9999999999').shift_left(number))
```

ACTIVITY 1.2 - Magic 8 ball

Goals

- Make a Magic 8 ball to give random answers to your spoken questions, such as “Will I ever become Prime Minister?”
- Practice **branching** (decisions).
- Practice assigning and using **variables**, including **arrays** (lists).
- Practice generating random numbers.

JavaScript
parallel document

Python
parallel document

Choices, choices

BUILD

1. Here’s our basic plan for the Magic 8 Ball program (Use the reset button on the back of the micro:bit to run again.):

   Scroll “Ask your question then press button A.”

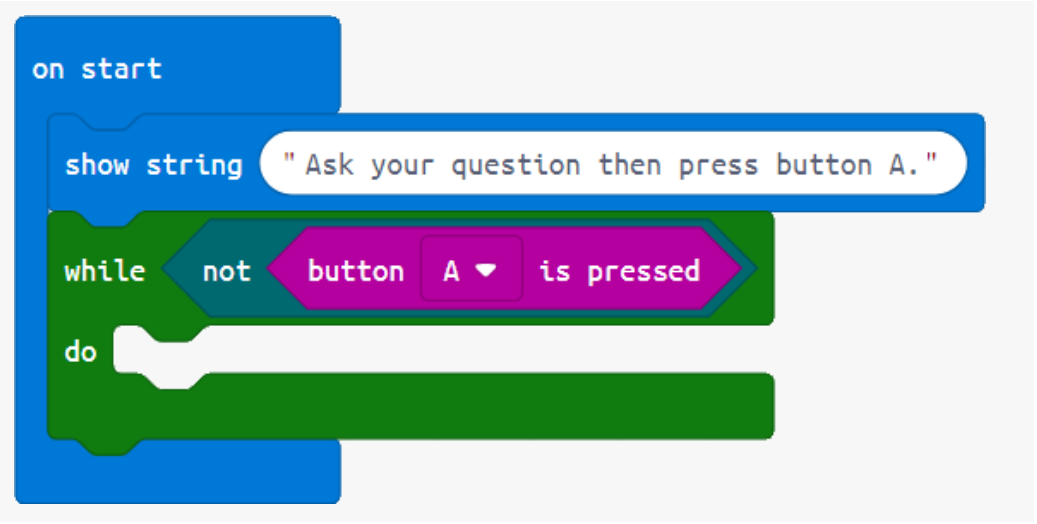
   Wait for Button A to be pressed

   Choose a random answer

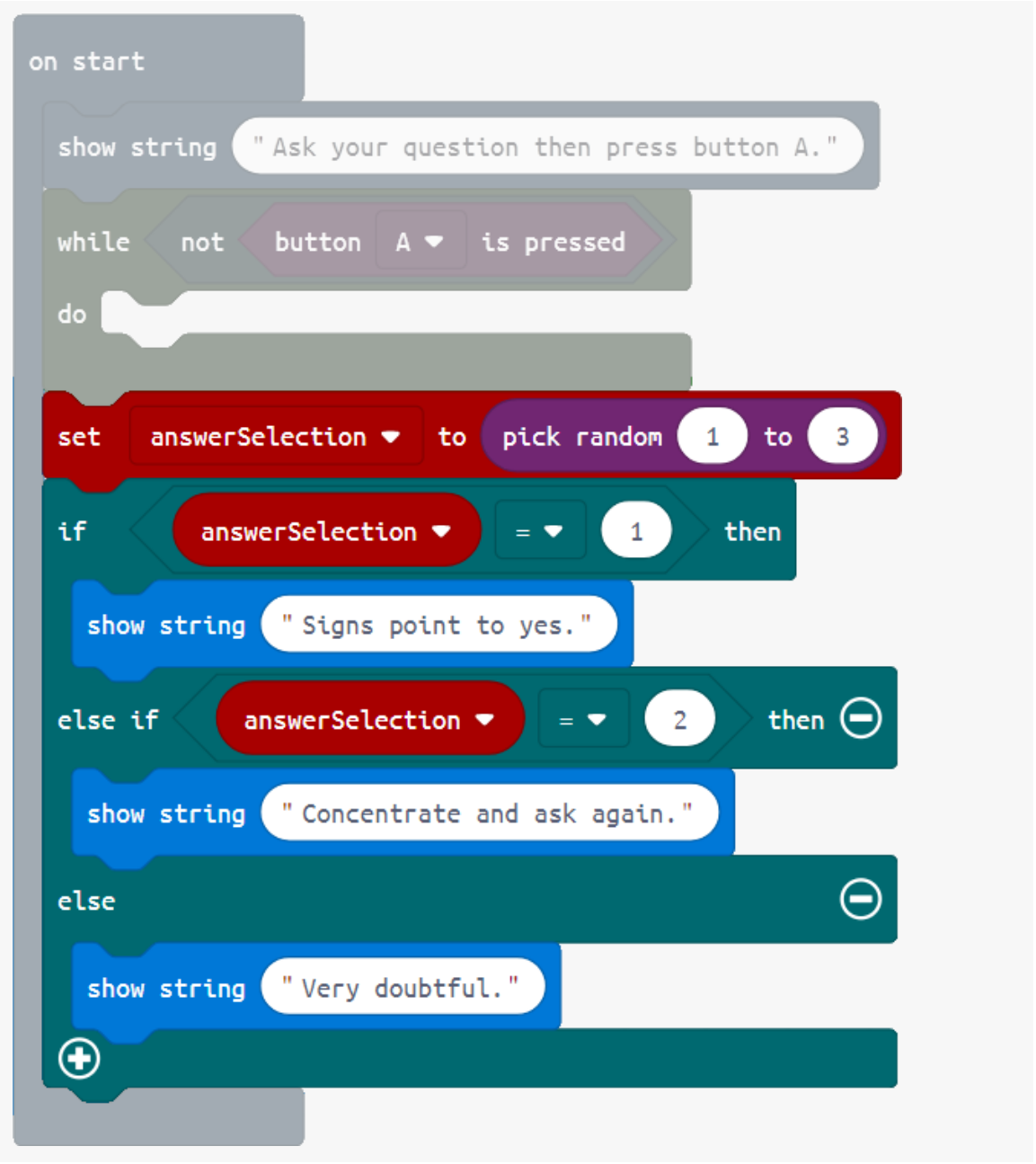
   Scroll the answer

2. To wait for button A to be pressed, we’ll use an empty loop. It will keep doing *nothing* as long as button A is not pressed.

Visual

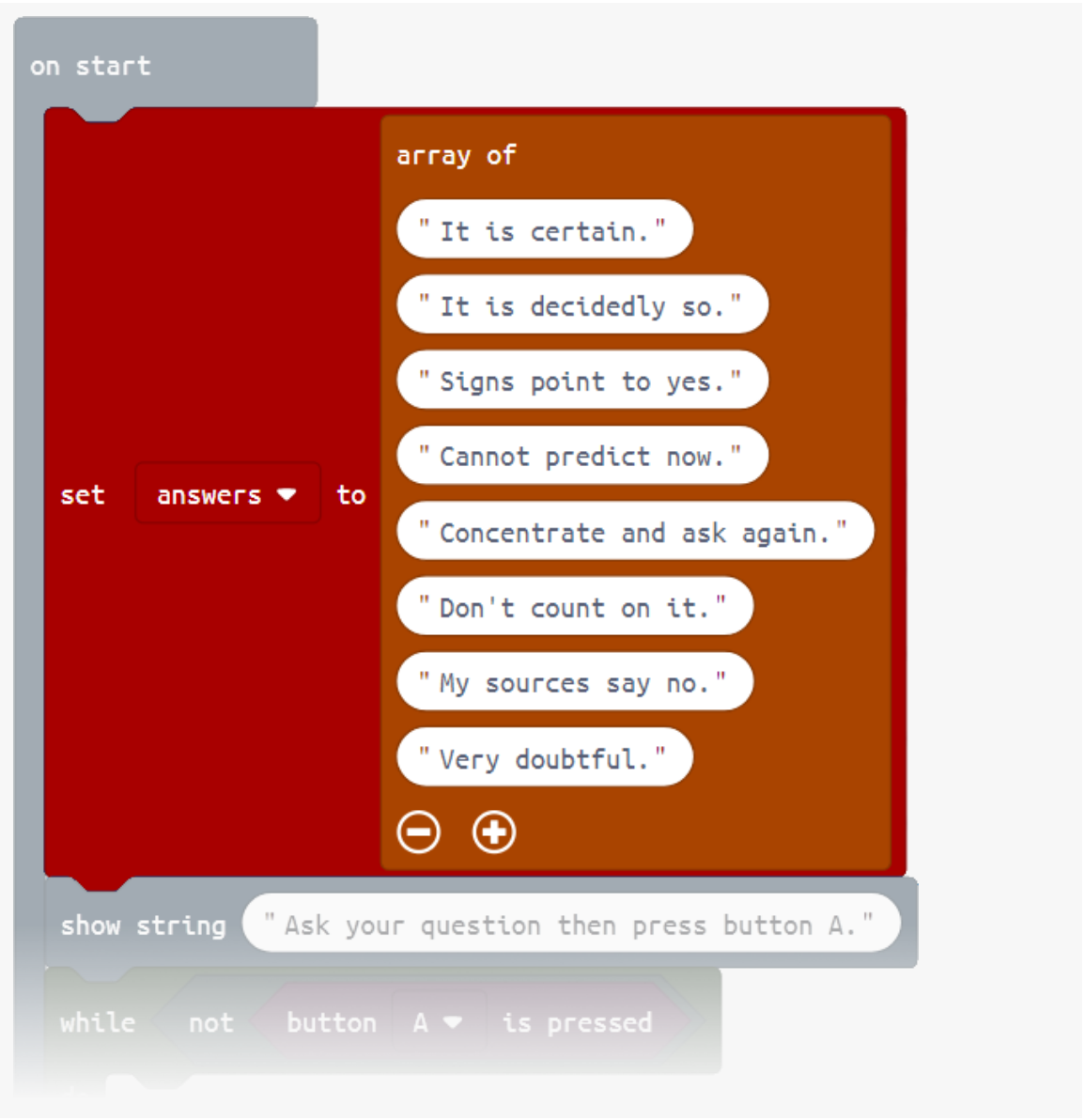


3. To give a random answer, we'll need to get a random number, like rolling a die. Store that number in a variable, then use it to choose different answers.

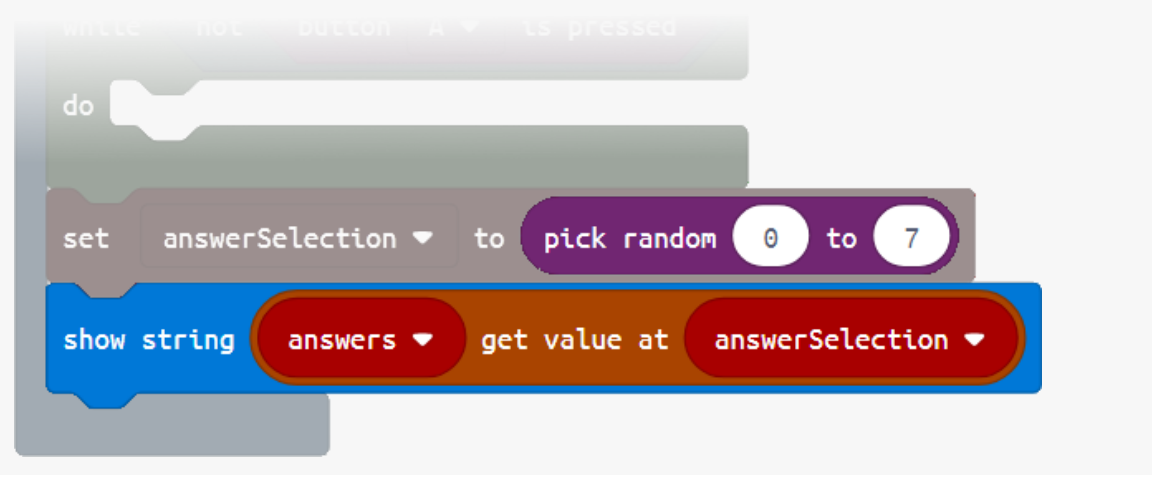


More answers

4. Our **if** and **else** blocks will become very large if we try to add lots more answers. A better way is to make a list (array) of all the answers before we start.



5. Now we can use our random number to access an answer from the list.



Why did we need to pick a random number between 0 and 7 (inclusive)?

- Our list contains 8 answers, with “It is certain” at position 0.
- This means the last answer “Very doubtful” is at position 7.

Playing with the Magic 8 Ball

TINKER

A. Change just the last line of the program so that the answer is always “Don’t count on it.”

B. Add some more answers of your own. Remember to increase the limit of the random number.

C. Put in a loop so that the program starts again, without you having to press reset.

D. Modify the program to have a cheat. Pressing button B always gives “It is certain”.

The sky's the limit

JUMP OFF

With the power of variables and random numbers, you can make the micro:bit do all sorts of unexpected things:

- Magic 8 Ball enhancements:
    - **Moody Magic 8 Ball** - Refine your Magic 8 Ball to start by showing a happy, sad or meh face. If happy, the Magic 8 Ball will be optimistic with its answers. If sad, it will be pessimistic. If meh, it may give any kind of answer.
    - **Warm up Magic 8 Ball** - If the temperature is below a certain threshold the Magic 8 Ball tells the user, "I am too cold to give predictions. Warm me up first!" If there is no fridge or ice pack available then the Magic 8 Ball can be too hot.

Here's the code for checking temperature:

    - **Things are about to go south** - Experiment with the compass heading block and create a very pessimistic message when the micro:bit is facing due South. This message should override all other random answers.

- **Greedy pig** - Program the micro:bit to play the game "Greedy pig". A random number is chosen between 0 and 9. Rolling a 2 means game over, but any other number gets added to the player's score. Press A to roll again, or B to take your money and run. You could make the game harder by adding more "deadly" numbers as the game goes on.

- **Dice simulation** - Use the micro:bit to help you run a dice-rolling simulation with a pair of six-sided dice. After 100 rolls, what was the most common total to get?

Hint: Store the total of each roll as a number in an array. The array will have 100 numbers in it.

ACTIVITY 1.3 - Reaction time challenge

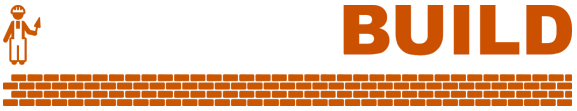
Goals

- Make a challenge to test your reaction time - how soon can you push the button?
- Record elapsed time with a **variable**.

JavaScript
parallel document

Python
parallel document

Random starting gun



1. Here's the complete program in Structured English (Pseudocode).

   Scroll "Get ready..."

   Wait for unpredictable time

   Show a symbol to start the challenge

   reactionTime ← *0*

   REPEAT until button B is pressed

   Increase ***reactionTime***

   END REPEAT

   Scroll "Your time: ", ***reactionTime***

2. We need a **random** pause time between 2 and 4 seconds before the challenge begins.

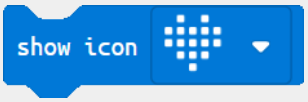
Visual

3. We'll need something to indicate the challenge has started. Light up the middle display pixel.

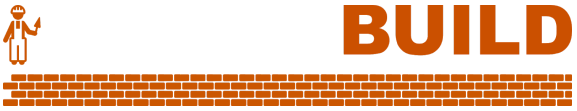
Why not use an icon?

The show icon command block causes a 600ms time delay.

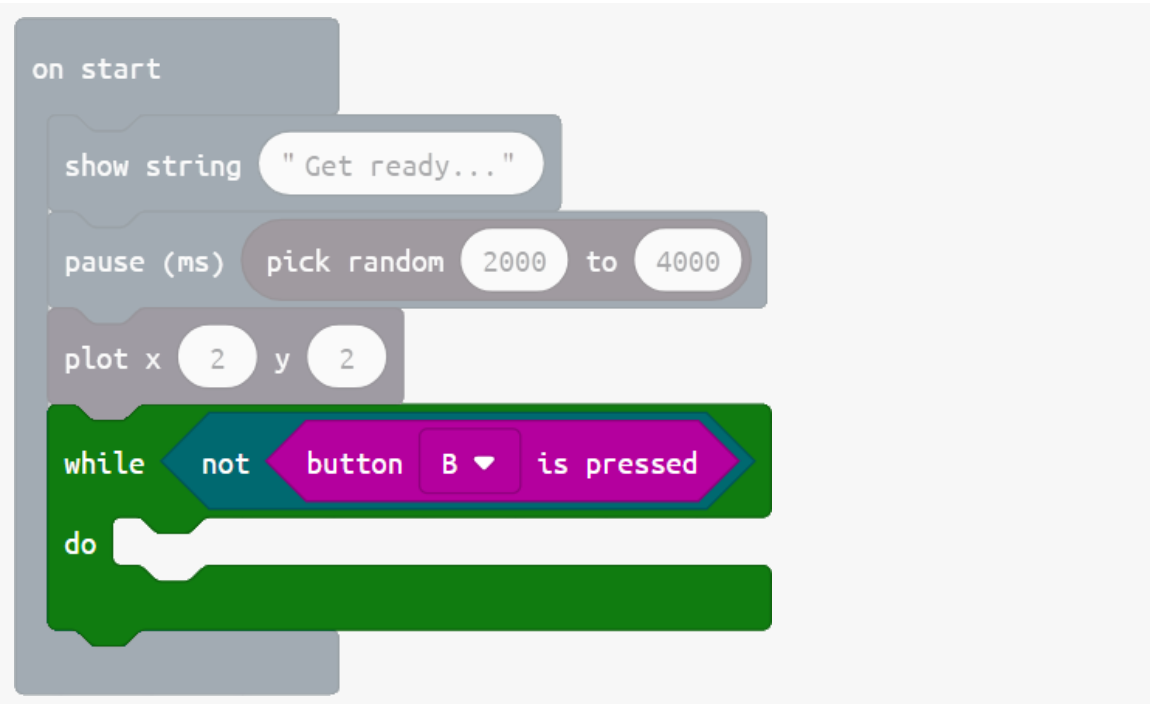
Normally that's not a problem, but it will not do for time-sensitive challenges like this.



Keeping track of time



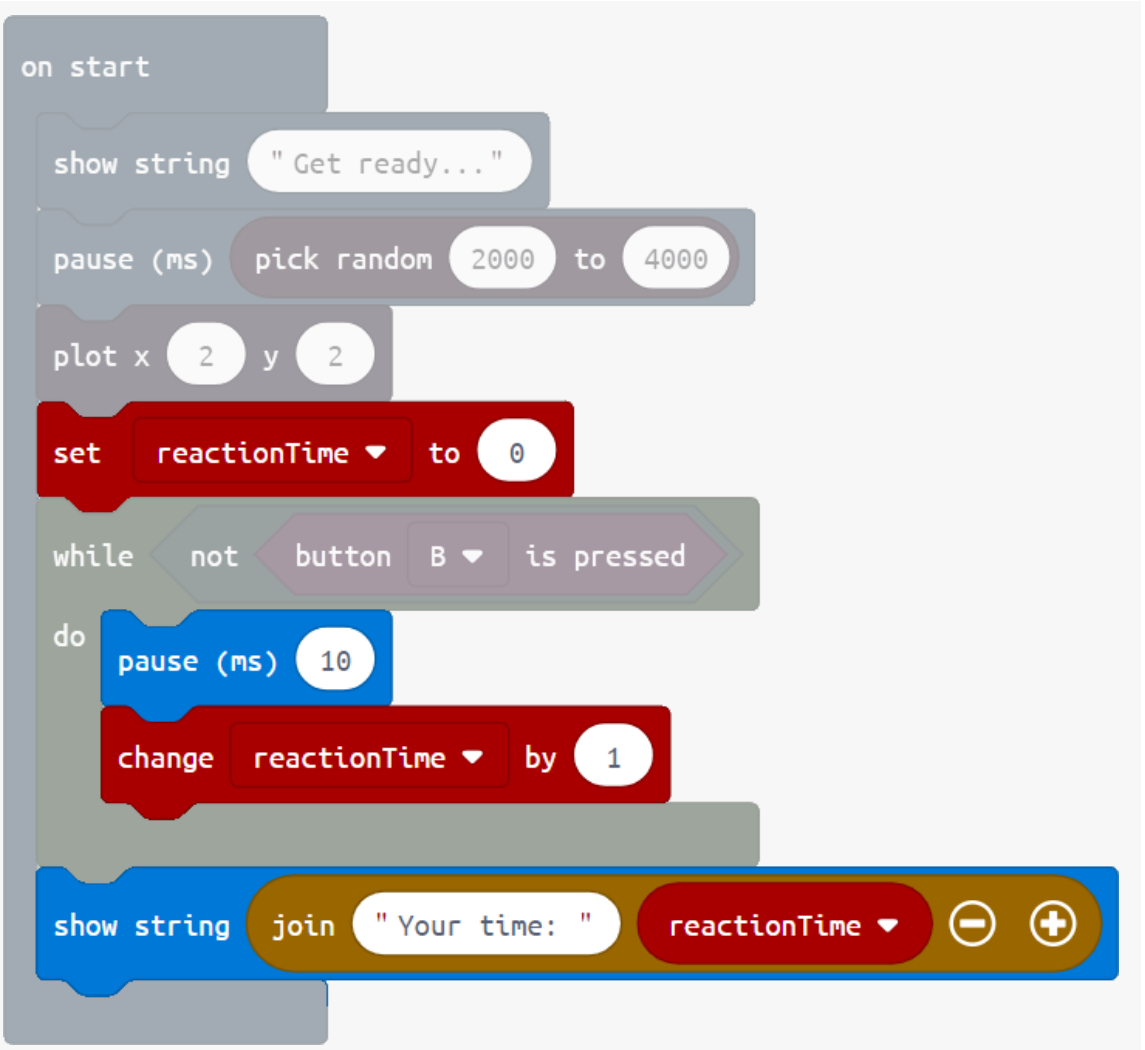
4. The challenge goes on *until* Button B is pressed. That sounds like a loop!



Reset button

Use the reset button on the back of the micro:bit to re-run your code without downloading it again.

5. Now comes the important part. We will count the reaction time in hundredths of a second, which means we need a variable to increase every 10 milliseconds.



Why measure in hundredths of a second?

Our program increases reactionTime in increments of 10 ms, rather than every 1 ms. This is due to limits with microprocessor hardware.

Refinements

A. Between 2 and 4 seconds is too long to wait at the start. Change it to between 1 and 3 seconds.

B. Change the program so that it shows a smiley face every time the user achieves a time lower than half a second, a meh face if lower than a second, and a sad face otherwise.

C. Put in a loop so that the program starts again, without you having to press reset.

D. Now limit the game to exactly 3 rounds, then say “Game over”.

E. For fun, have the pixel light up in a different random position each time you play.

Buttons for interactivity

A surprising number of games and tools can be made for just one or two buttons. Here’s some ideas to get started on your own project:

- **Improved challenge** - Improve the Reaction Time Challenge by storing the score for each round, then providing the player with an average score when all the rounds are over. You might even make it a “hotseat” challenge for 2 players.
- **Unlock a secret** - To unlock a secret message, you must enter the right sequence using buttons A and B. eg. ABAAB. There are always 5 buttons to press. If you get it wrong, you have to start again.

Do you think this is a very secure way to store a secret?

- How many possible combinations exist?

- **“The claw!”** - Can you stop the claw in the right place to grab the toy?
  A toy is represented by a pixel somewhere on the bottom row of the display. A claw at the top of the display moves left and right. Press a button to stop the claw in the right place, then watch it drop to grab the toy.

ACTIVITY 1.4 - Button masher

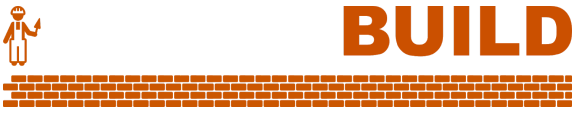
Goals

- Make a button masher game - how many times can you push the button during a limited time?
- Practice **Maths** operations.

JavaScript
parallel document

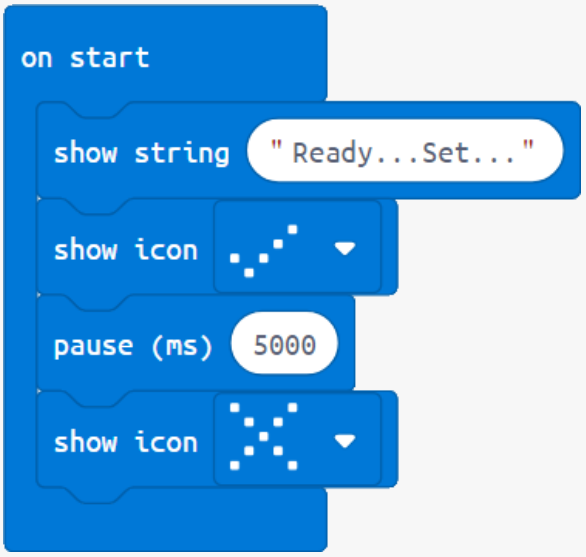
Python
parallel document

5 second timer



1. First, we need code to start the game, then end after 5 seconds. (Use the reset button on the back of the micro:bit to run again.)

Visual

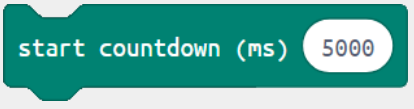


2. Wait, we have a problem! Using **pause** means that nothing can happen during the 5 seconds.

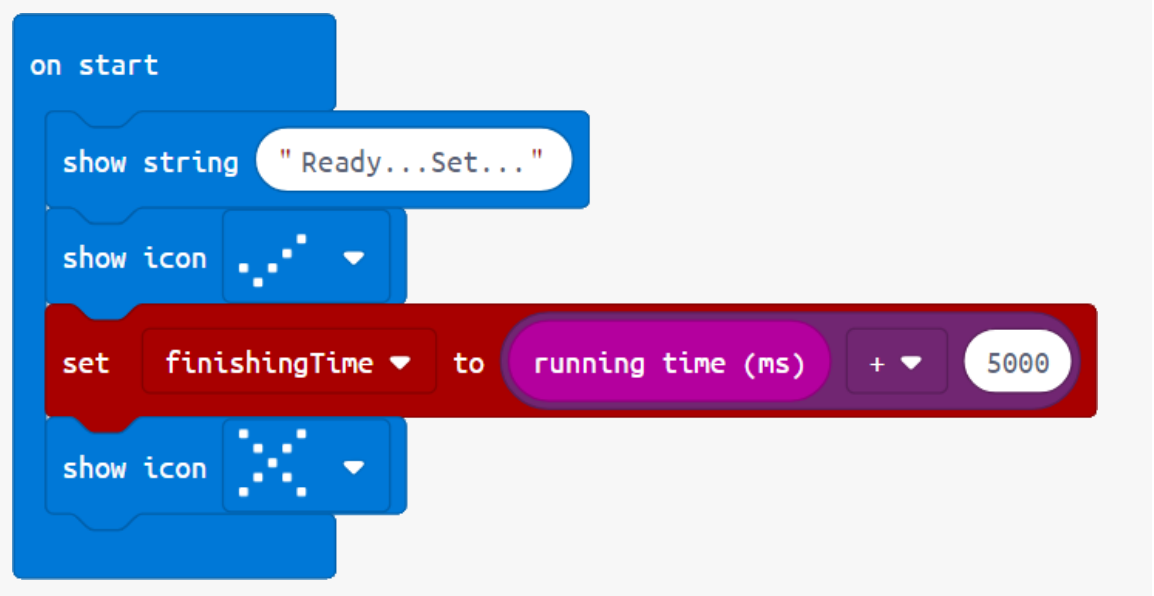
How can we get the micro:bit to set a timer for 5 seconds, but allow other things to happen in-between?

- We could use separate event blocks like the one at right, *but we avoid using these in this course (see APPENDIX C).*

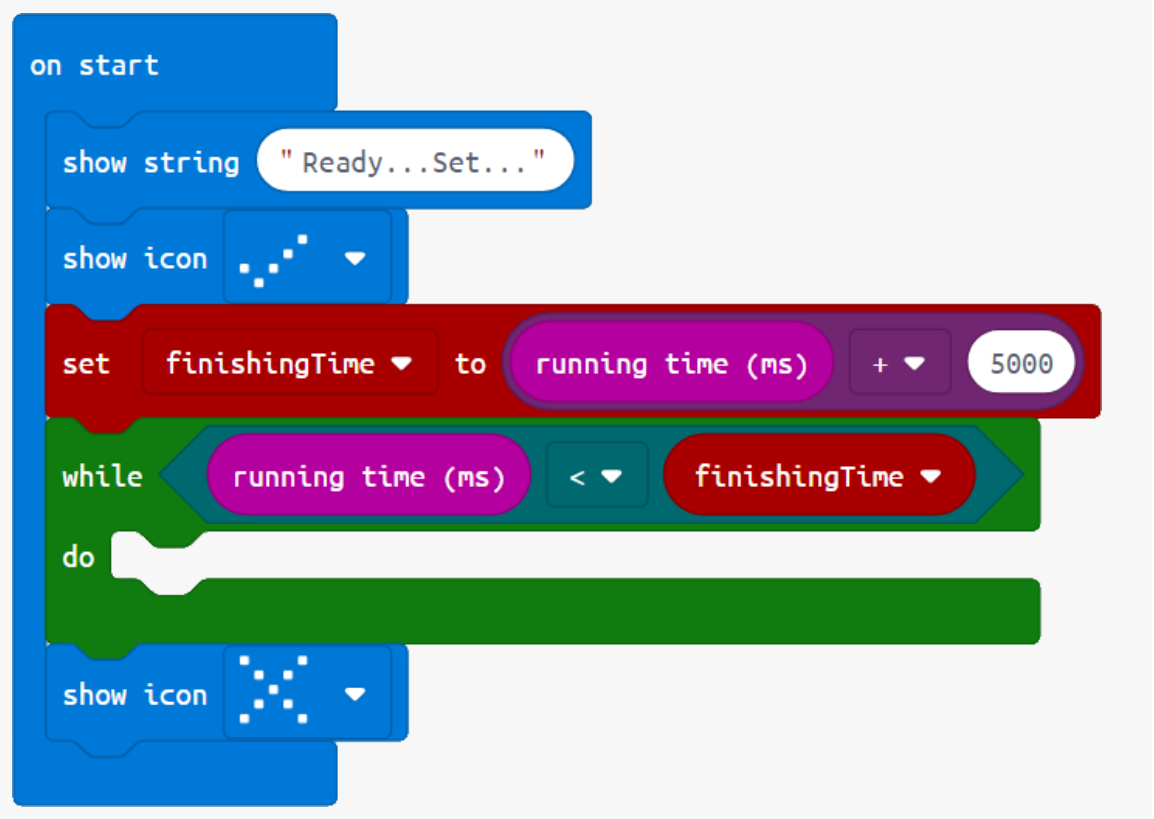
- Game blocks are available, *but these are special shortcuts unique to micro:bit. Also, they are not available in Python.*



Our solution: Make a variable for the finishing time. Set it to 5 seconds more than the current system running time.



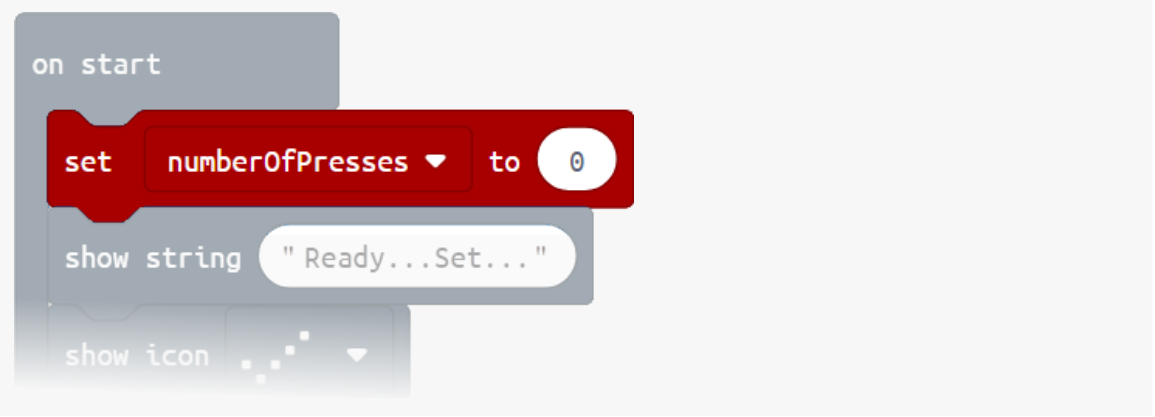
3. Now a loop just needs to keep checking that the current time is still below **finishingTime**. This solution means we can do other things inside the loop.



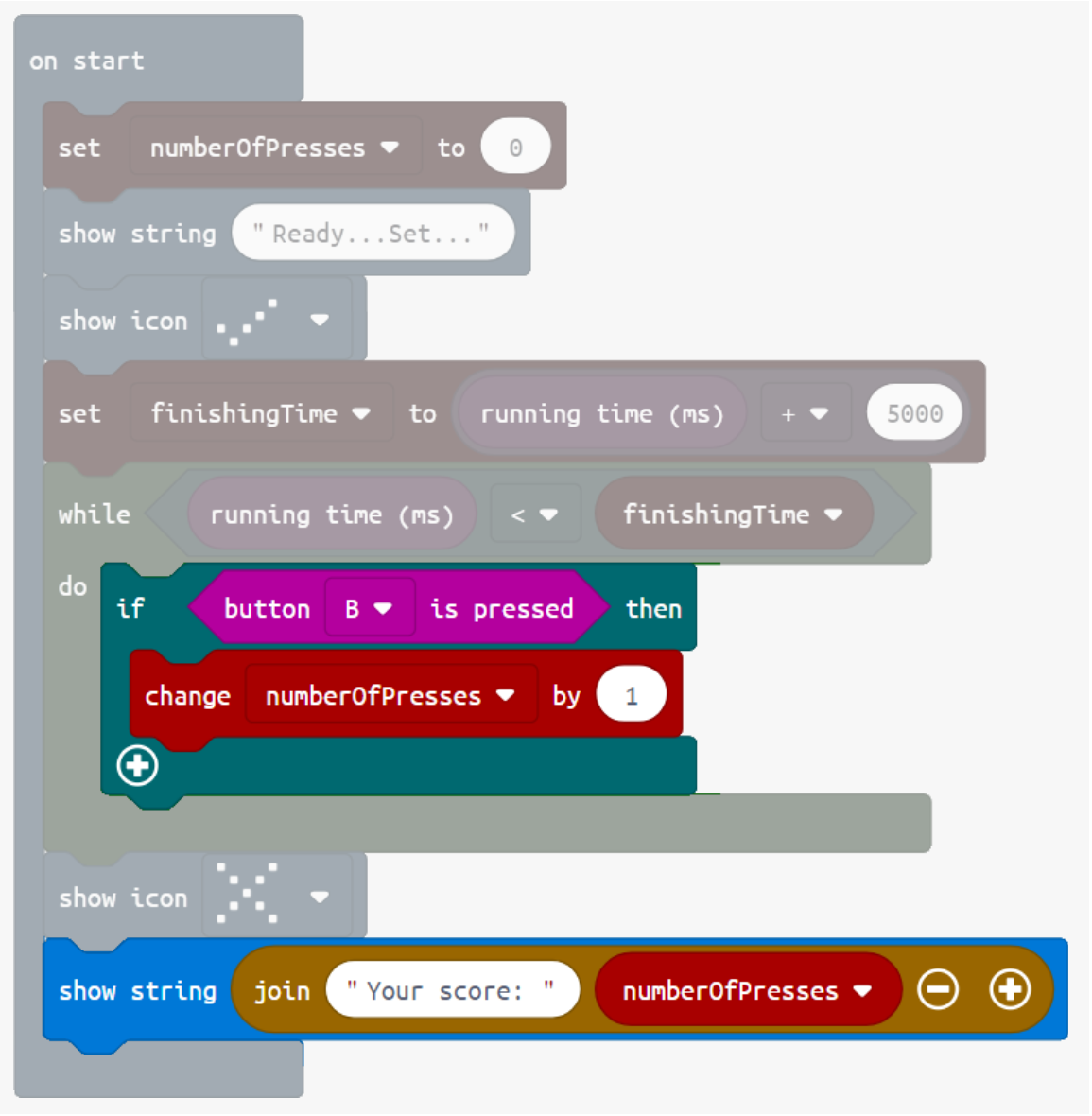
4. Here's our plan for the full program in Structured English (Pseudocode):

```
Scroll "Ready...Set..."
Show tick symbol
finishingTime ← 5 seconds from now
REPEAT while now is not yet finishingTime
    IF button B is pressed
        Add 1 to numberOfPresses
    END IF
END REPEAT
Show cross symbol
Scroll "Your score: ", numberOfPresses
```

5. To hold the number of presses we'll need a variable. We'll set it to 0 at the start.



6. Increase **numberOfPresses** when button B is pressed during the 5 second loop. Finally, display it at the end.

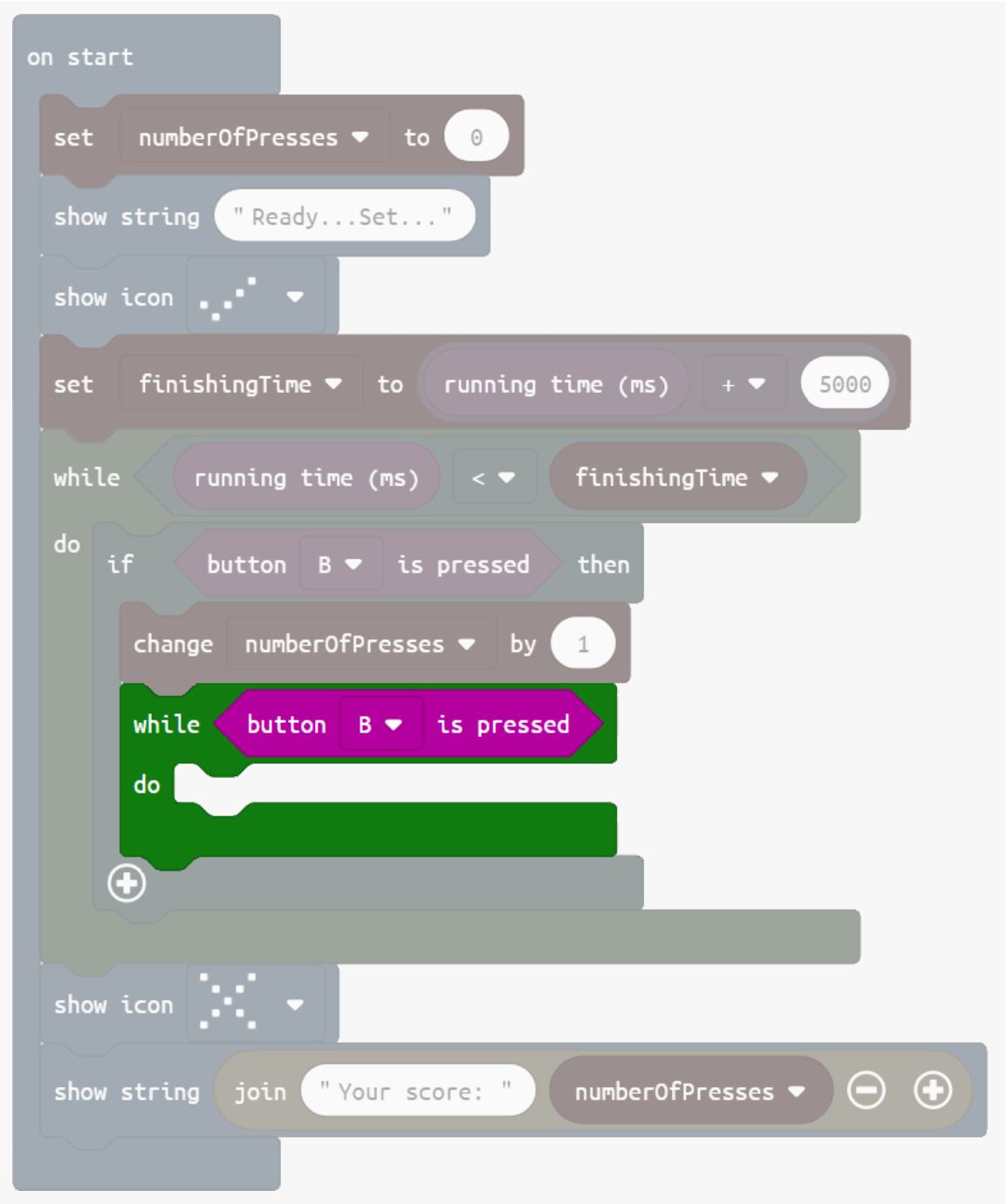


7. Did you test the program so far? You should have noticed a bug.

What is the bug, and what is causing it?

- The recorded number of presses is way too high, and it's not even consistent!
- The problem is with the **if** block:
  - It only checks if button B is *currently* being pushed down, but it doesn't wait for the button to be released.
  - The loop gets repeated thousands times during the 5 seconds, and **numberOfPresses** increases whenever it sees that button B is down.

8. We're going to have to wait until the button is released before allowing the loop to repeat. Add an empty extra loop inside the **if** block to do this waiting. All done!

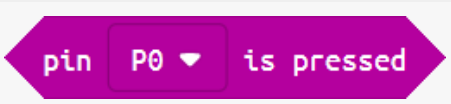


Alternate designs

Important: *Go back to the original completed game code before each of these tasks.*

A. Change the game to count touches on one of the micro:bit's pins, instead of the button.

Here's the micro:bit input check for Pin 0.

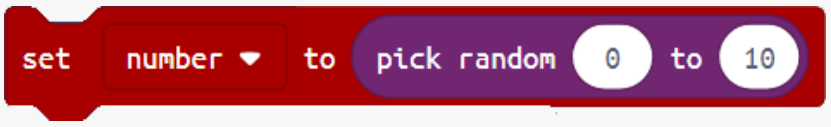


Note:

- You have to be touching the GND pin too.
- This works best if the micro:bit is running on battery power, not USB.

B. Make it a random time limit. Don't forget to tell the player the time limit before the game starts.

We discovered how to pick random numbers in ACTIVITY 1.2. Here's the code to store a random number between 0 and 10:



C. Give the player a target to win the game. eg. 15 presses in 5 seconds.

D. Run the original game three times. When finished all three games, display the average score.

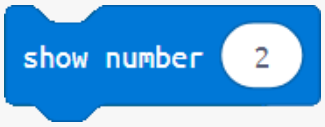
Hint: You'll need a loop around the whole program, and a way to remember the score for each game.

Maths means games

Most video games incorporate Maths, eg. character stats, points scored. Now that you can have a counter increasing, what else could you make?

- **Whack a mole** - When the left side of the screen lights up, quickly press button A. When it's the right, press button B. How many moles can you whack before the time runs out?
- **Maths quiz** - The micro:bit displays a Maths question with a single-digit answer. To select the answer, use button A to cycle through the numbers 0 to 9 on the screen, then button B to select one. Did the player get it right? How about a time limit to give an answer?

This code immediately displays the number 2 on the screen (no scrolling):



- **RPG fighter** - Your enemy attacks with a random damage value. Will you attack or block with your turn?

ACTIVITY 1.5 - Raining bricks

Goals

- Make a game where the player collects or avoids falling bricks.
- Generate and animate a falling brick from random positions.
- Detect a collision between the player and the brick.
- Use simple **functions** to help organise the program.

JavaScript
parallel document

Python
parallel document

The brick's behaviour

BUILD

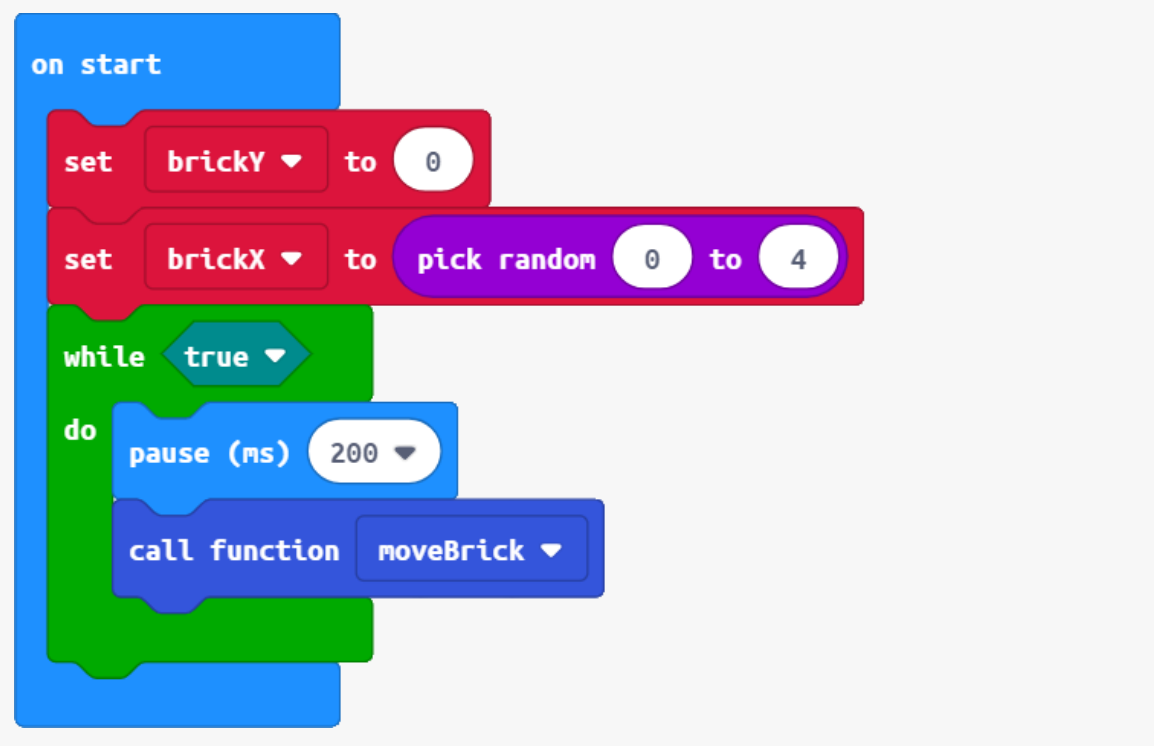
1. The brick will be represented by one lit pixel. It will start from a random *x*-position (0 to 4) at the top of the display, then move down one pixel down every 200 ms.

The 25 LEDs have coordinates as in the diagram.

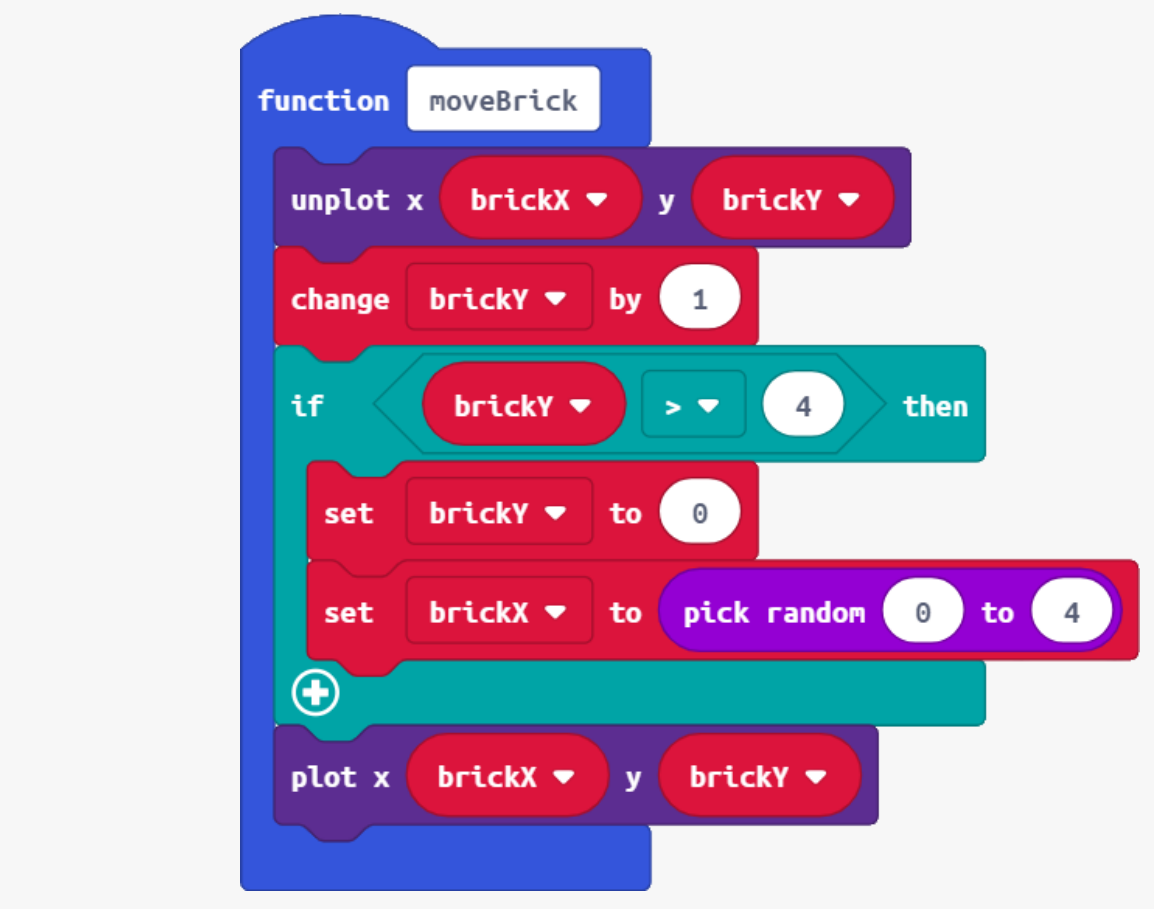
We'll use two variables **brickX** and **brickY** to keep track of the brick's position on the screen.

```
brickY ← 0                          Start at top of display.
brickX ← random number from 0 to 4
REPEAT forever
    Wait for 200 ms                 A short pause every time we loop.
    Unplot pixel at (brickX, brickY)  Before moving the brick, turn off the
                                      LED at its previous position.
    Add 1 to brickY                 Move down 1 pixel.
    IF brickY > 4                   If brick has gone past the bottom...
        brickY ← 0                    ...go back to the top,
        brickX ← random number from 0 to 4   and find a new position to start.
    END IF
    Plot pixel at (brickX, brickY)  Turn on the LED at the new position.
END REPEAT
```

2. First, let's set up the starting commands and the main loop. This program will get very big, so we'll create a simple function **moveBrick** to separate off all the code for moving the brick.



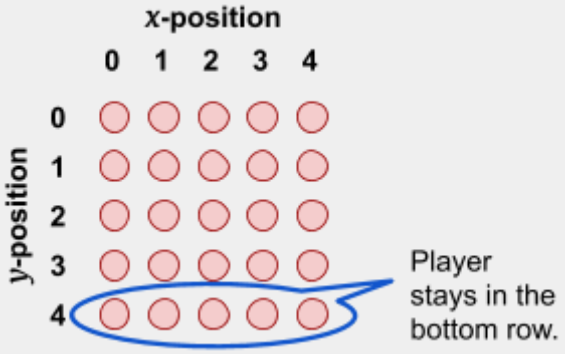
Here's the code for the function **moveBrick**:



The player's behaviour

3. Now let's plan the program for the player.

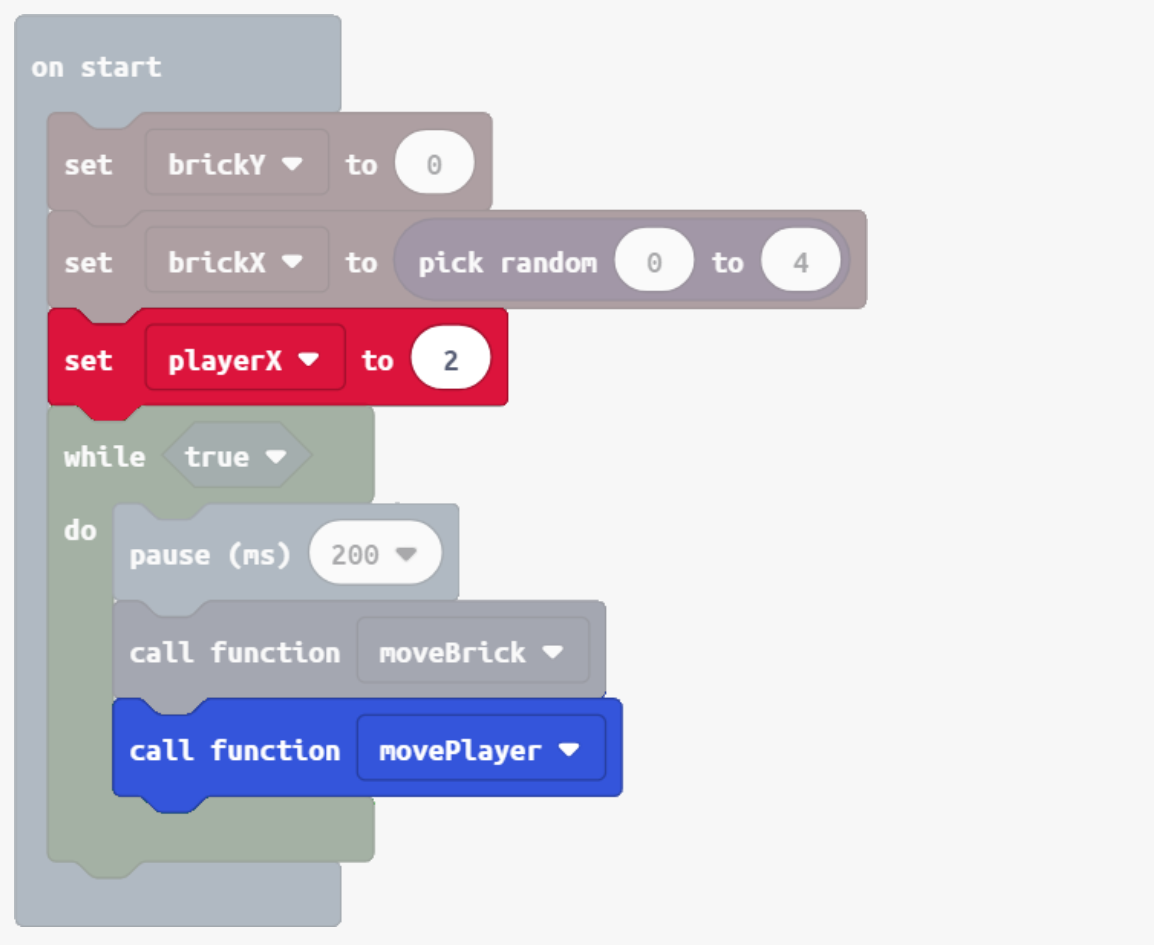
The player is always in the bottom row, but button A and button B will move them left and right.



We'll use one variable **playerX** to keep track of the player's position in the bottom row.

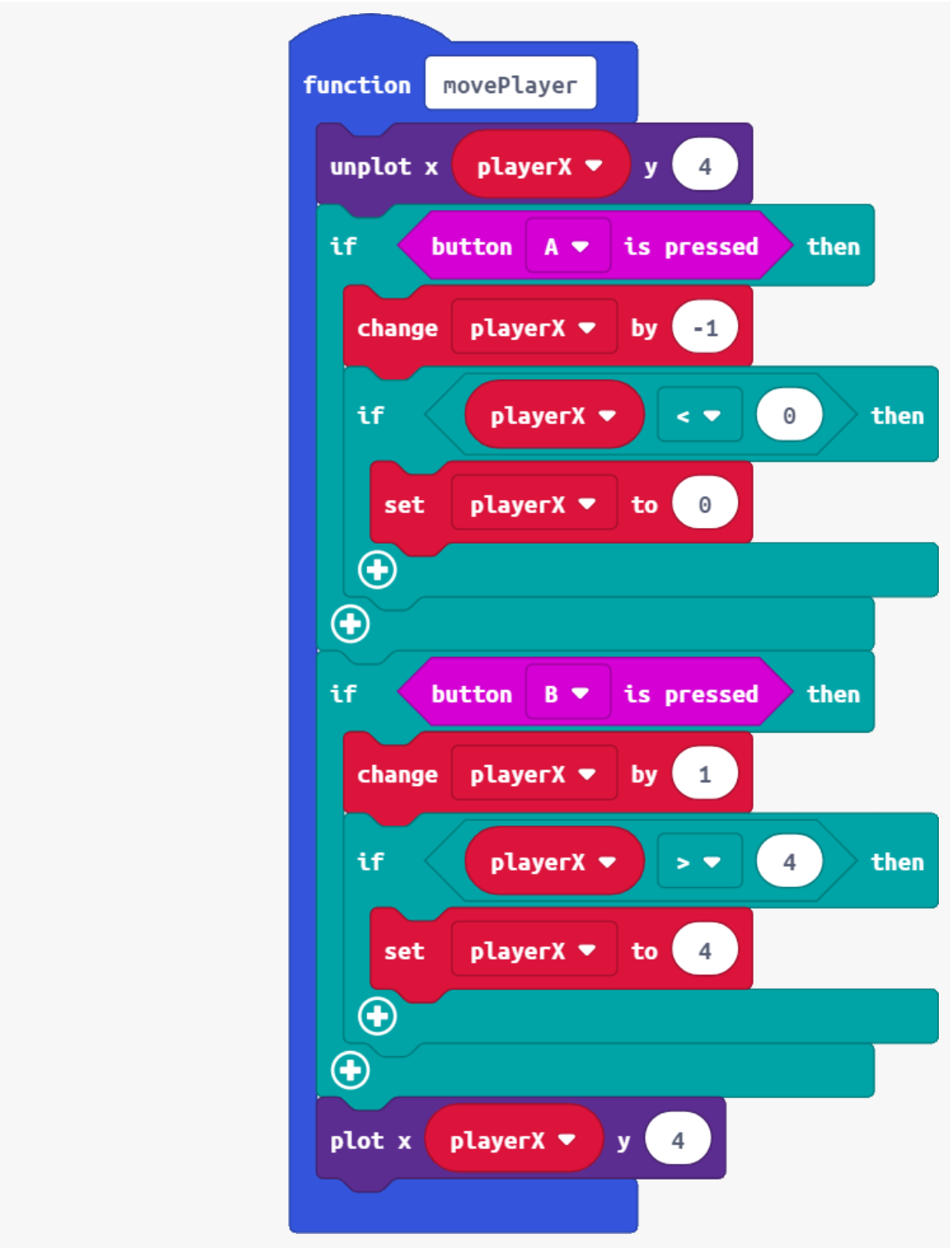
```
playerX ← 2                          Start in the middle at the bottom.
REPEAT forever                       (The same loop from the brick code.)
    Wait for 200 ms                  (The same wait from the brick code.)
    Unplot pixel at (playerX, 4)     Before moving the player, turn off the
                                     LED at their previous position.
    IF button A is pressed
        Minus 1 from playerX
        IF playerX < 0               If they've moved off the left side...
            playerX ← 0                  ...return to left side.
        END IF
    END IF
    IF button B is pressed
        Add 1 to playerX
        IF playerX > 4               If they've moved off the right side...
            playerX ← 4                  ...return to right side.
        END IF
    END IF
    Plot pixel at (playerX, 4)       Turn on the LED at the new position.
END REPEAT
```

4. Here's what to add to the main routine:

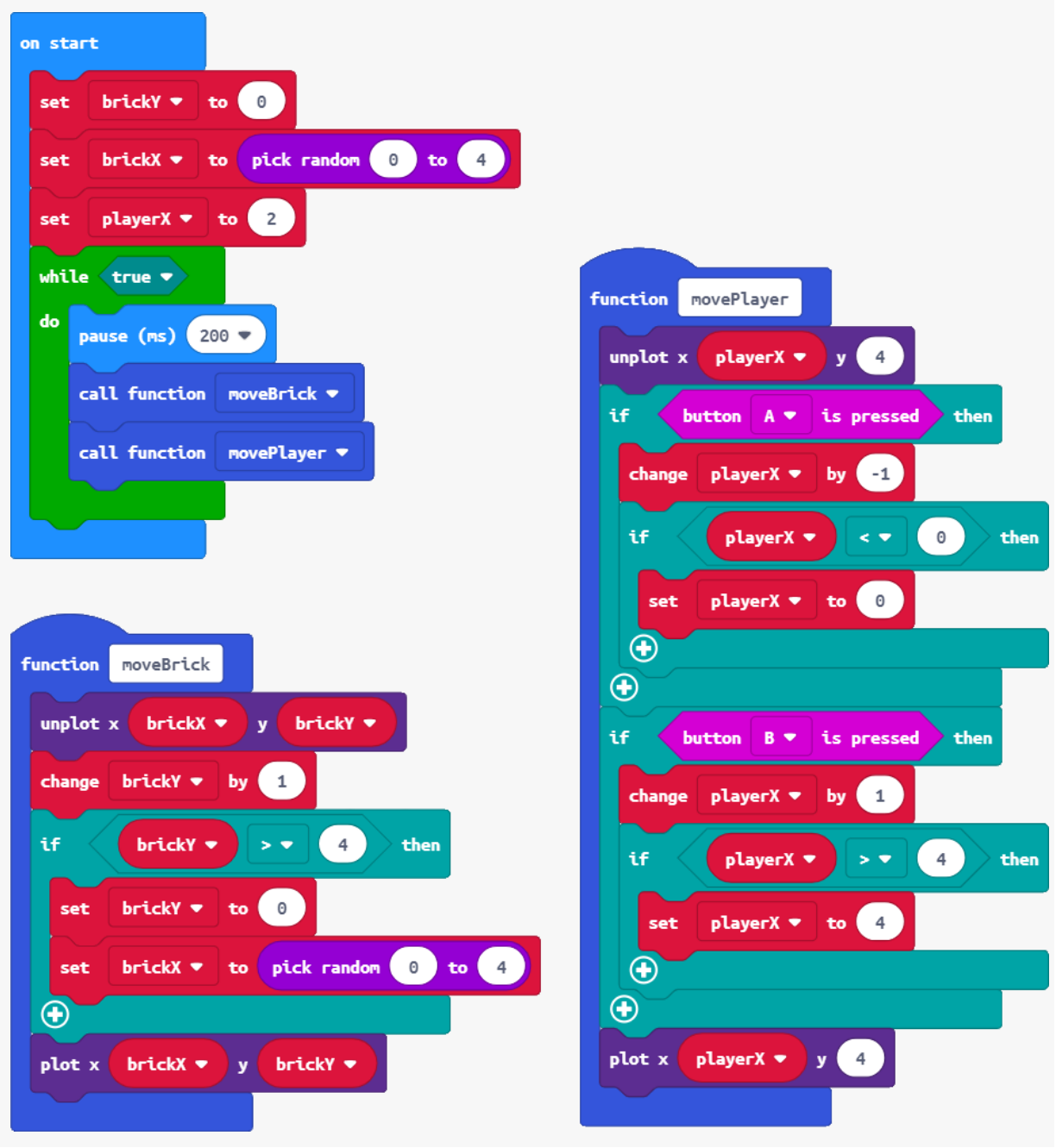


*(Code for function **movePlayer** is on next page...)*

Here's the code for function **movePlayer**:



5. Here's the complete program so far.



Making the game playable

A. Detect a collision. A collision is when the player and the brick are in exactly the same position. When this happens, blink the overlapping LED two times, then resume the game.

B. Create a variable to count collisions. End the game at exactly 3 collisions.

C. Create a score variable and increase it by 1 every time the player successfully dodges the brick. Print the score when the game is over.

D. When the game is over, congratulate the player if the score is greater than 20.

Animated games

Despite its small display, the micro:bit can be coded for many types of animated games.

- More improvements to Raining Bricks:
    - **Wet Wet Wet** - Introduce a second brick, which is generated when the first brick gets halfway down the screen.
    - **Tilt control** - Use the accelerometer reading to control the position of the player. Tilting the micro:bit left and right moves the player.
    - **Collect the bricks** - Change the rules so that the player now collects bricks, and the score goes up by 1 when a brick is collected. The game is over when 10 bricks have been missed (not collected). Select the original or new game from a menu by pressing button A or B.
- **Ski between the flags** - Pairs of pixels scroll up the screen, and the player pixel must pass between them.
- **Flappy pixel** - Press a button to keep the pixel in the air as it moves sideways and dodges mountains.

ACTIVITY 1.6 - Multiplayer dice game

Goals

- Make a simple dice game between two micro:bits.
- Work in pairs to use the micro:bit's radio functionality.

JavaScript
parallel document

Python
parallel document

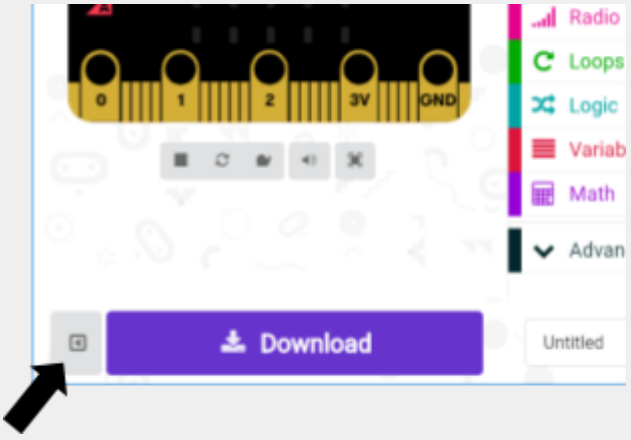
Sending and receiving

BUILD

Chrome and radio

At time of testing (January 2019), Chrome browser may **slow down** or **hang** when using radio commands along with loops in the online editor.

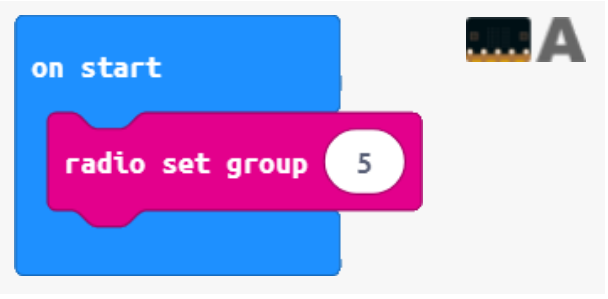
If this occurs, try to return to the micro:bit home page (top left), then restart the browser.

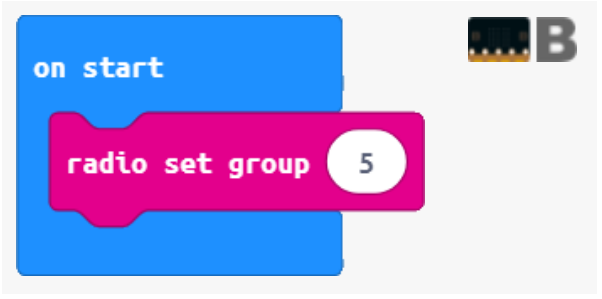


The problem may be avoided by hiding the micro:bit emulator *before* starting to code. This is done by pressing the button at the bottom left of the page.

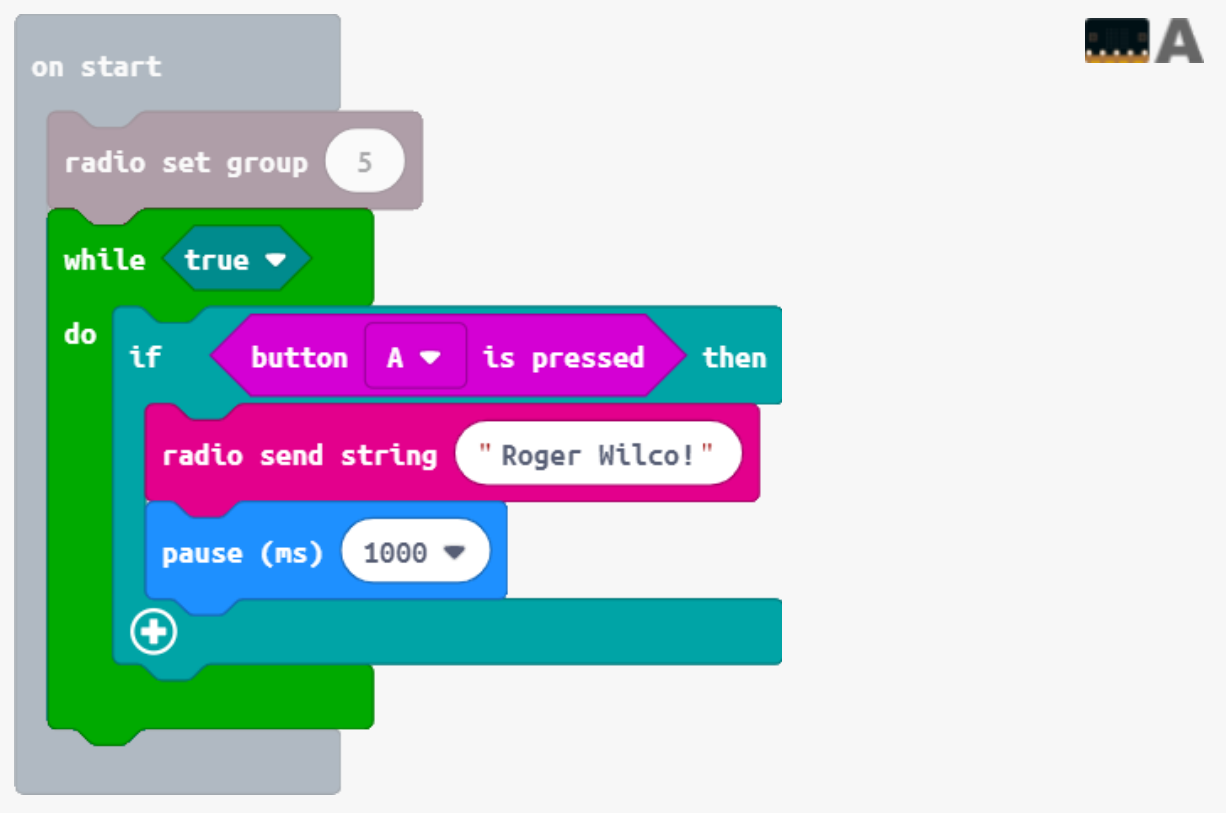
Firefox and Edge browsers tested on Windows 10 did not have the same issue.

1. To test the radio, each pair of micro:bits should use a different group number. We'll use 5 in this example.

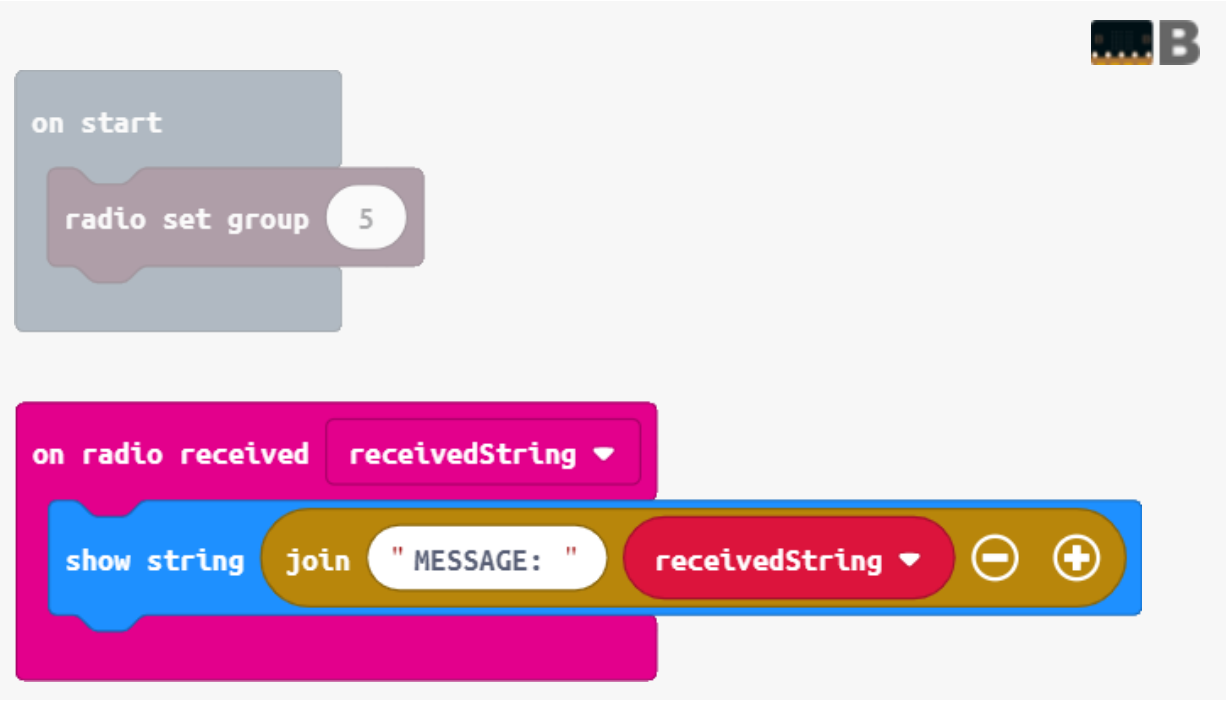




2. Code micro:bit **A** to send message whenever a button is pressed. Make the message unique.



Now code micro:bit **B** to display any message it receives. Try it out!

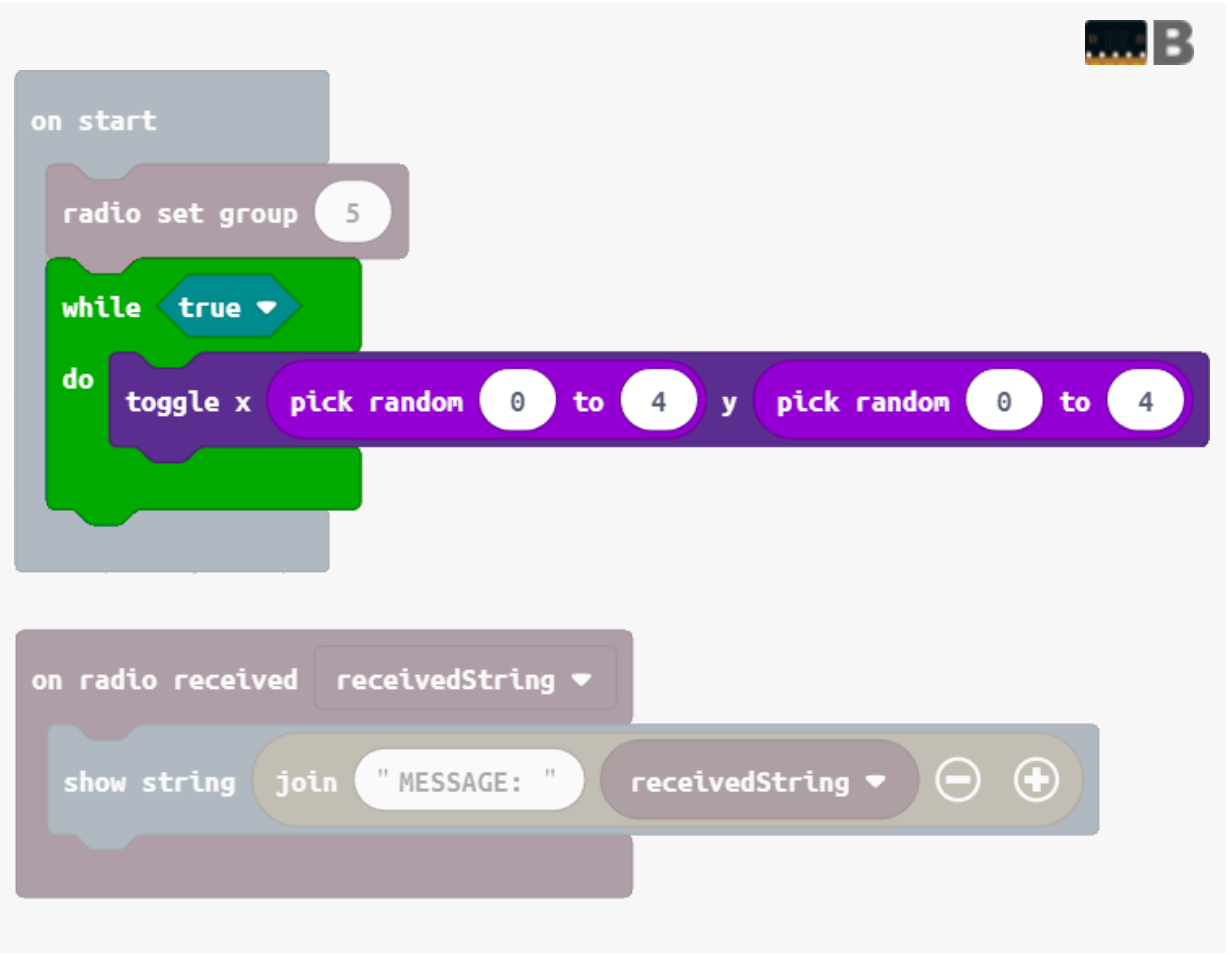


Using an event block

We have avoided using event blocks until now (see APPENDIX C).

Unfortunately, it is unavoidable for radio, since there is no standard command to check for new radio messages.

3. Let’s make micro:bit **B** flash LEDs while it’s waiting for a message to come in.



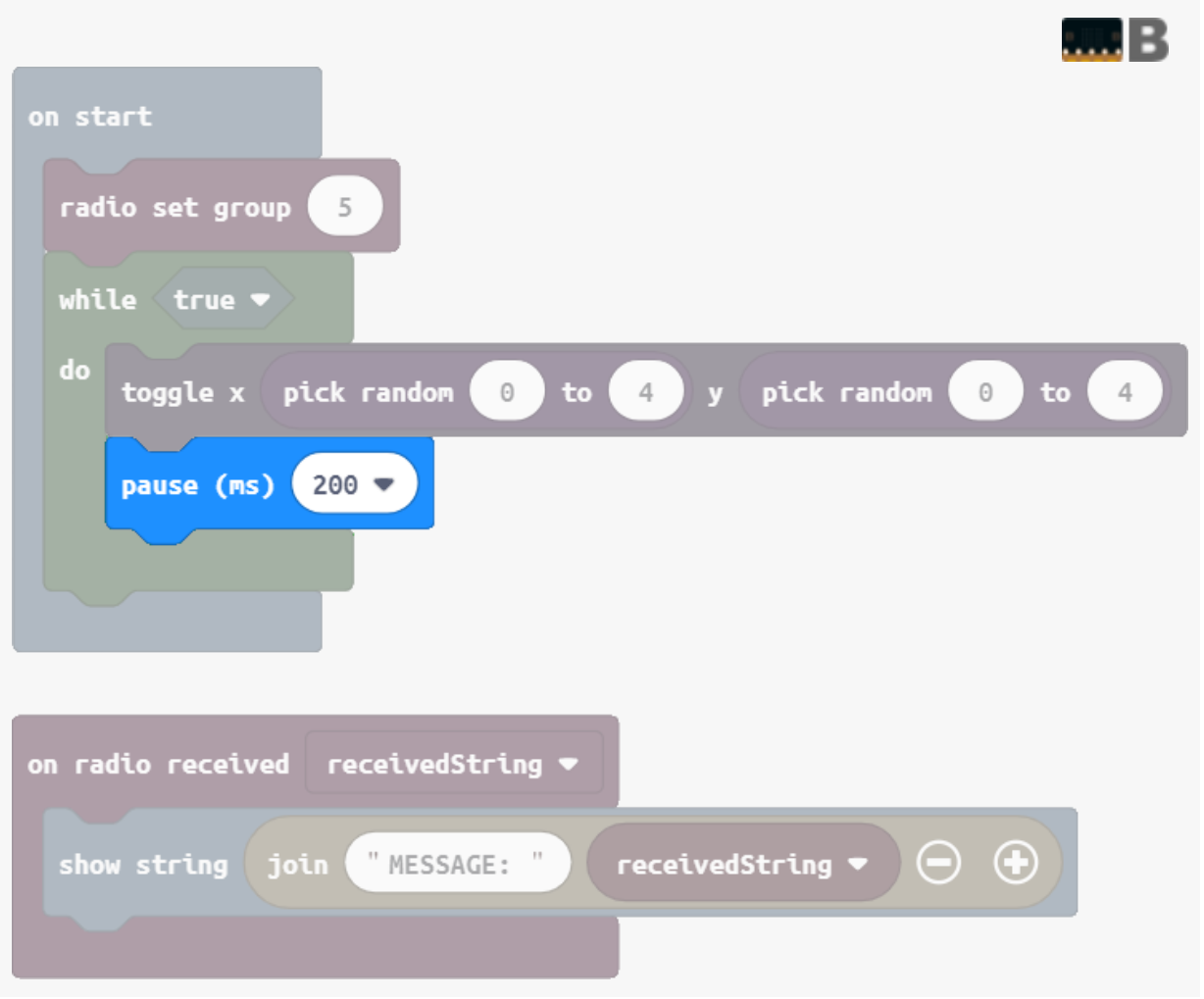
Something went wrong! No messages?

Micro:bit **B** is not receiving any messages. Why?

The event trigger **on radio received** will not happen unless there is some *idle time* in the main routine.

In other words, micro:bit **B** is too busy flashing to check for new messages. There must always be some kind of **pause** in the loop.

4. Let’s add a pause and see if that helps.

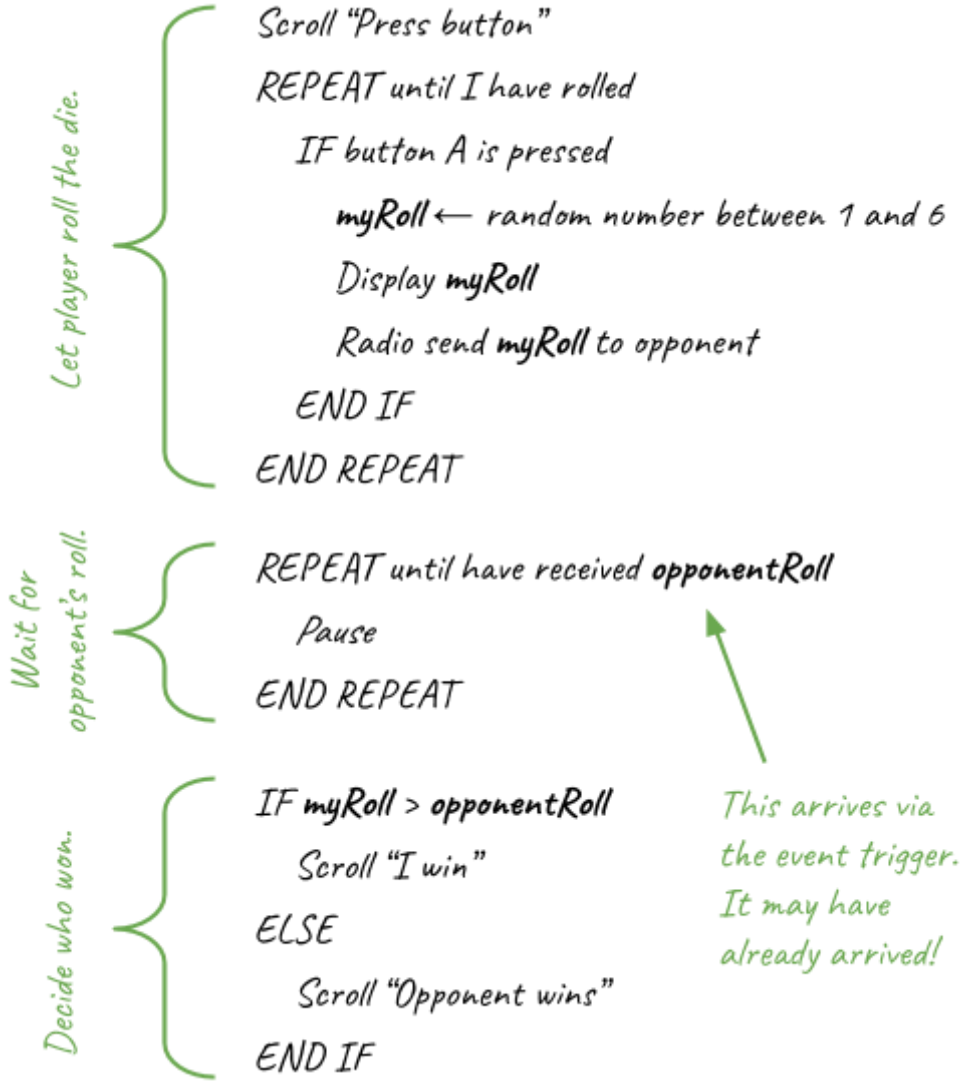


It still looks bad! How come?

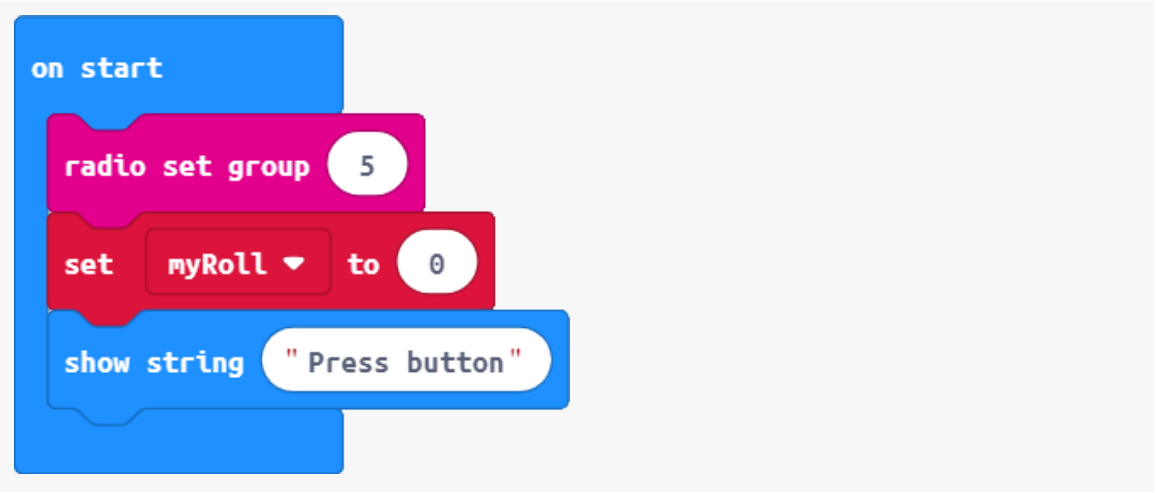
Radio messages are getting through now, but the main loop is going back to toggling those LEDs at the same time the message is being scrolled.

We’re going to have to smart when using event triggers.

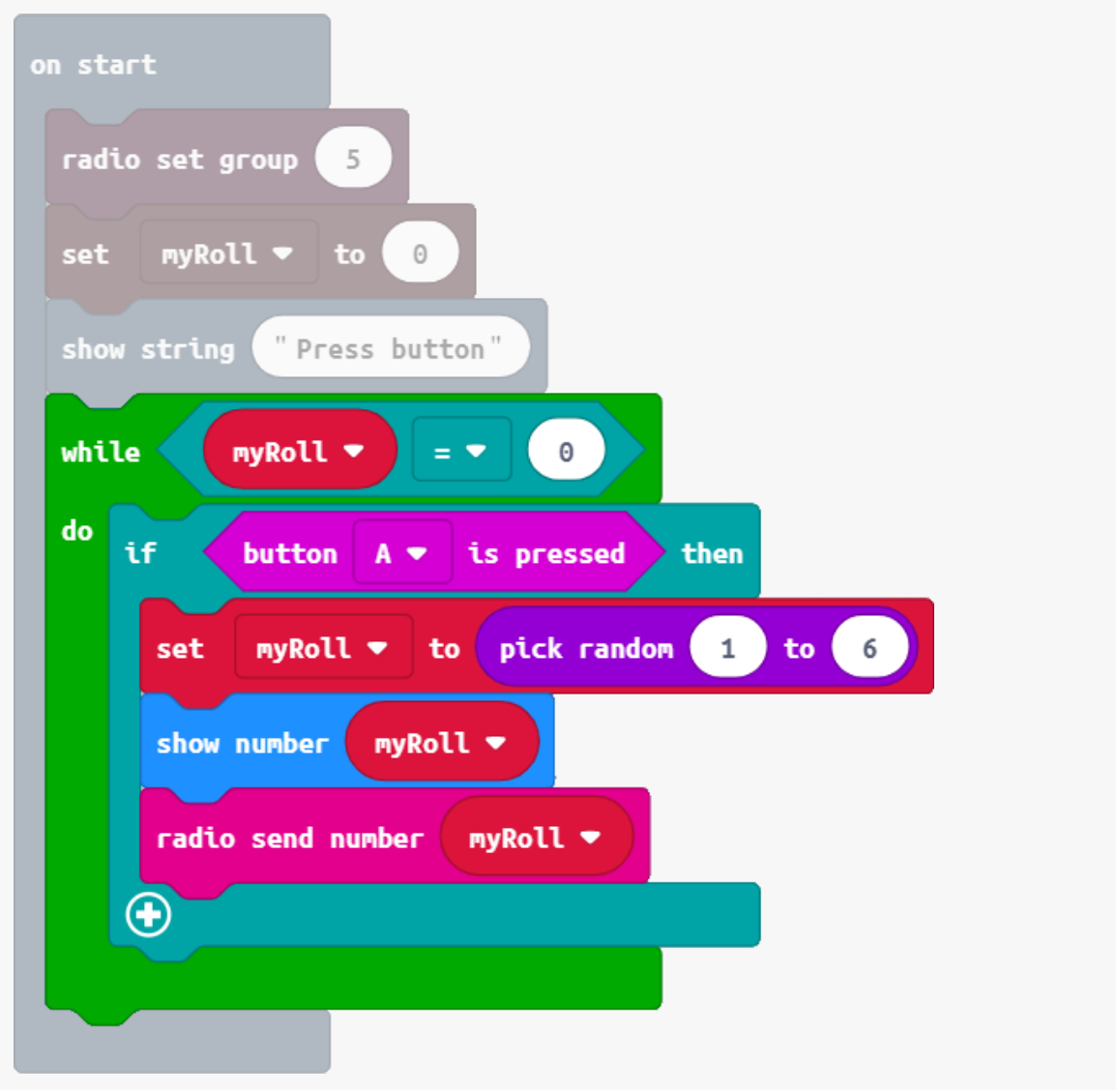
5. Here's our plan for the full program. The same program can go on **both** micro:bits.



6. Let's start with the player's roll. 0 is a good starting value for **myRoll**, since we can't roll a 0.



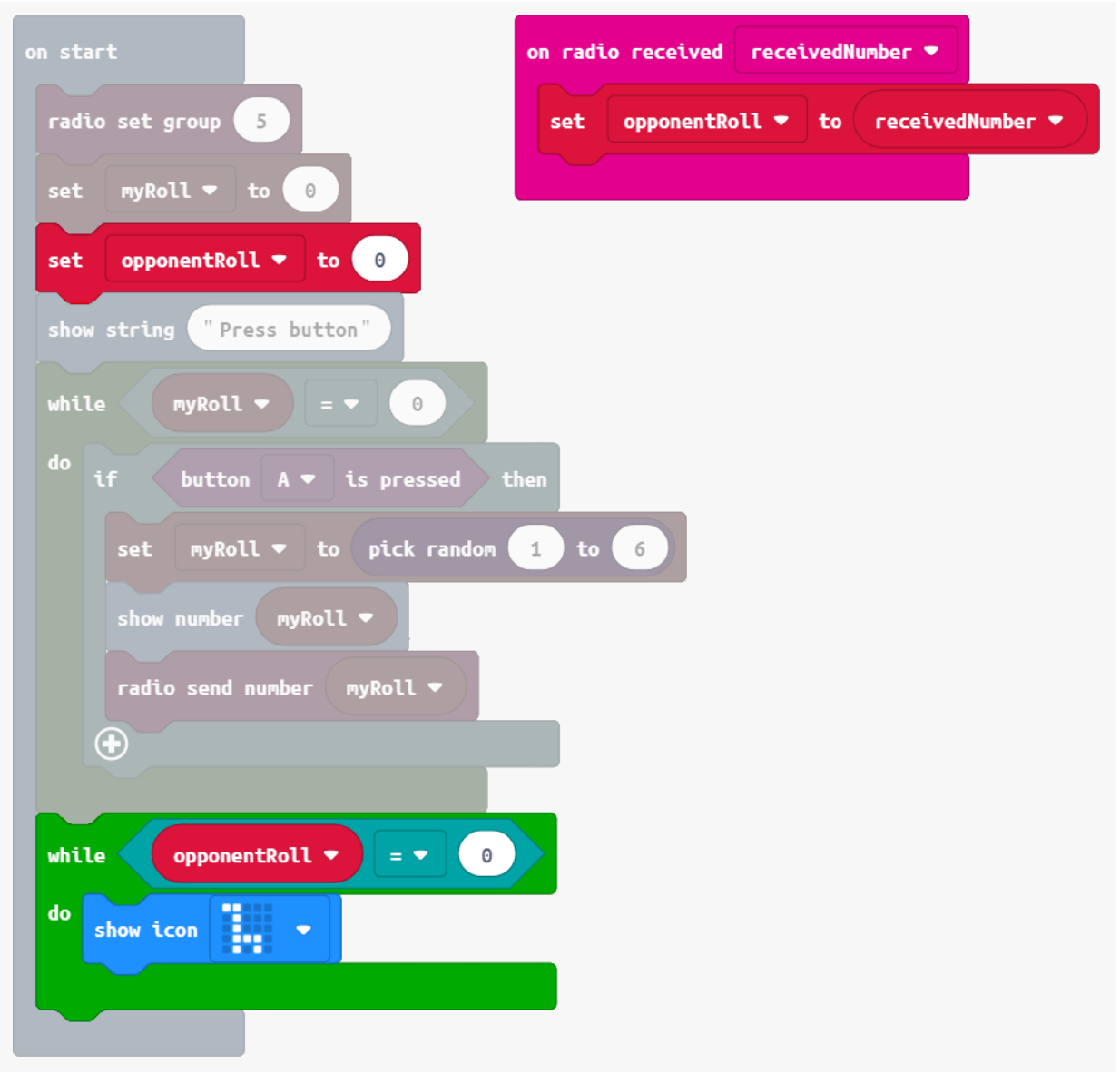
7. We'll wait for the button press until **myRoll** is *not* 0.



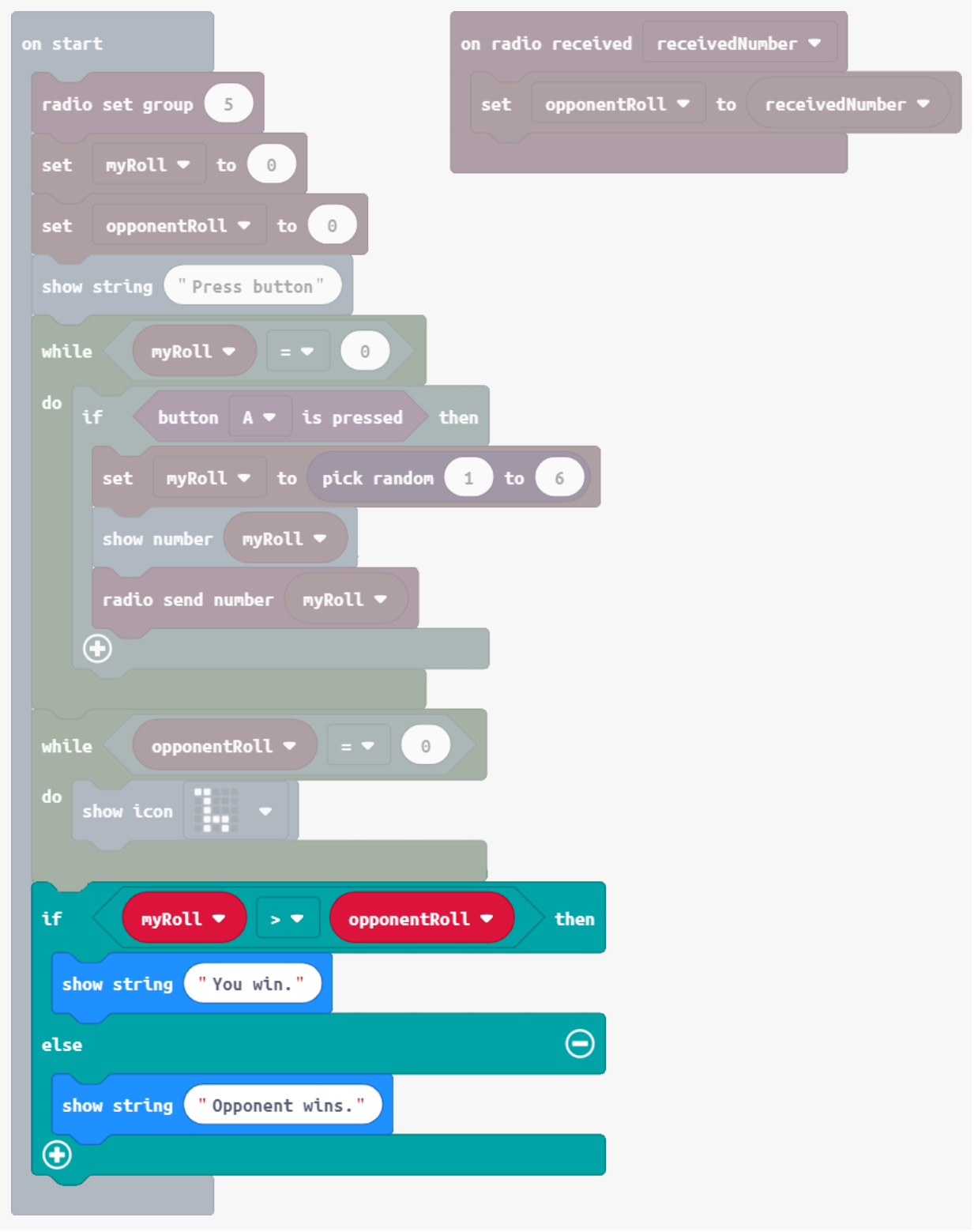
Opponent's roll

8. Now to receive the opponent's roll.

   The **show icon** is very important - it provides a pause so that incoming messages can be checked.



9. Finally, decide the winner.



Enhancing the game

A. Instead of showing your roll as a number, use the pixels to show dots just like a real die.

B. Write a loop so that the game starts again after the winner is shown.

C. Keep a score and make a “best out of 3” challenge.

D. Animate a ticking clock while waiting for the opponent.

Radio star

The simple radio functionality makes the micro:bit very powerful.

- More two-player games:
    - **Rock Paper Scissors** - Very similar to the dice game, but with simple logic: paper beats rock, scissors beat paper, rock beats scissors.
    - **Battleship** - This is more complicated. Start simple with one ship each, always in the same place. Then, add the more complex parts.
- **Swarm effects** - Combine radio commands with random numbers to make interesting animations with a class set of micro:bits, or conduct a simulation of a virus spreading.
- **Radio log** - If you’re recording data in a remote place (like temperature in the fridge), use radio messages to send the data to a second micro:bit in your hand.
- **Radio jammer** - It’s possible to create a program that cycles through all the radio groups and spams them with messages. Can you find a way for two micro:bits to still communicate if there is a jammer in the area?
- **Sing in harmony** - If you have audio outputs for your micro:bits, use radio to start them all singing different parts to a simple song. You can even generate harmonies if the melody micro:bit sends the major notes.

PROJECT - A digital solution

Steps

- **Investigate** and **define** a problem or need to address with a digital solution.
- **Design** the solution, including user interface and main algorithm.
- **Code**, **build** and **test** the solution, or a solution prototype.
- **Evaluate** the success of the solution or prototype, and examine possible impacts.

Investigate and define

We start by finding a problem or need we want to address through a physical tech solution.

PROMPTS FOR BRAINSTORMING / IDEATION

What bothers you or could be improved...

- in your classroom,
- in your hallways, lockers, common spaces,
- in your playground,
- in your house,
- in your local community?

Can you identify a need for...

- order and efficiency,
- safety,
- tuning into a social or environmental issue,
- fun and entertainment,
- something totally crazy?

Can you think of an existing solution that is...

- too slow,
- too big or too small,
- not connected enough?

EXAMPLE

After brainstorming, Abby, Ben and Kim identify a problem.

Their locker hallway is crowded because some students loiter too long when getting their things from their lockers.

Project Planning

Your team may formalise the planning of their project by:

- assigning roles to team members,
- fixing a “budget”,
- scheduling key dates such as meetings with the teacher or client,
- creating and maintaining a Gantt chart for the project.

DATA WORK

How can you get data to help you understand the problem?

- survey (paper or online)
- observation
- existing records
- Internet research

How will you...

- *validate* the data to know that it is useful and not full of errors,
- *store* the data securely,
- *analyse* the data to learn what it has to say,
- *visualise and present* the data / information learned.

EXAMPLE

The team prepares an online survey for students using the locker hallway.

Kim suggests using a spreadsheet for the data they get back. They remove any personal identity data, filter out error values and password-protect the spreadsheet file.

Next, they use some formulas to help prepare charts for presenting.

40% of surveyed students take more than 1 minute at their lockers. This may be the cause of the locker crowding.

SOLUTION REQUIREMENTS

What are the *functional requirements*?

- Break down the problem.
- Be specific - what must your solution primarily achieve?

What are the *non-functional requirements*?

- user friendliness,
- speed,
- accessibility?

Answer a few more questions:

- Which group of people will use your solution?
- What constraints are there (time, equipment, money, existing systems)?
- How will you judge if the solution works?

EXAMPLE

The team decides on 3 functional requirements:

1. The solution will be a timer that can be started by anyone next to a locker.
2. The solution will have a clear visual countdown to indicate time remaining.
3. The solution will alert students when 1 minute elapses.

Design

Design work happens before jumping right into coding and building.

USER INTERFACE (UI)

The user can interact with the solution via...

- buttons,
- pins,
- light sensor,
- temperature sensor,
- compass,
- accelerometer (movement, shaking),
- radio message from another micro:bit.

The user can be presented with output via...

- pixels,
- icons,
- scrolling text,
- radio message to another micro:bit,
- sound/music (with appropriate connection).

Draw two possible design mockups with labels (annotated).

Choose one design to proceed with.

EXAMPLE

Abby prepares the detailed mock-up for the team's chosen UI design.

The design allows either of the two buttons to start the timer. A button is pressed by the person at the locker, or by a person waiting.

A cardboard frame holds the micro:bit onto the wall near the locker, and shows how to use the timer.

Pixels on the display will show the countdown. A flashing icon will indicate that time is up.

MAIN ALGORITHM

Figure out your solution's main algorithm - the part that is most important for the functional requirements.

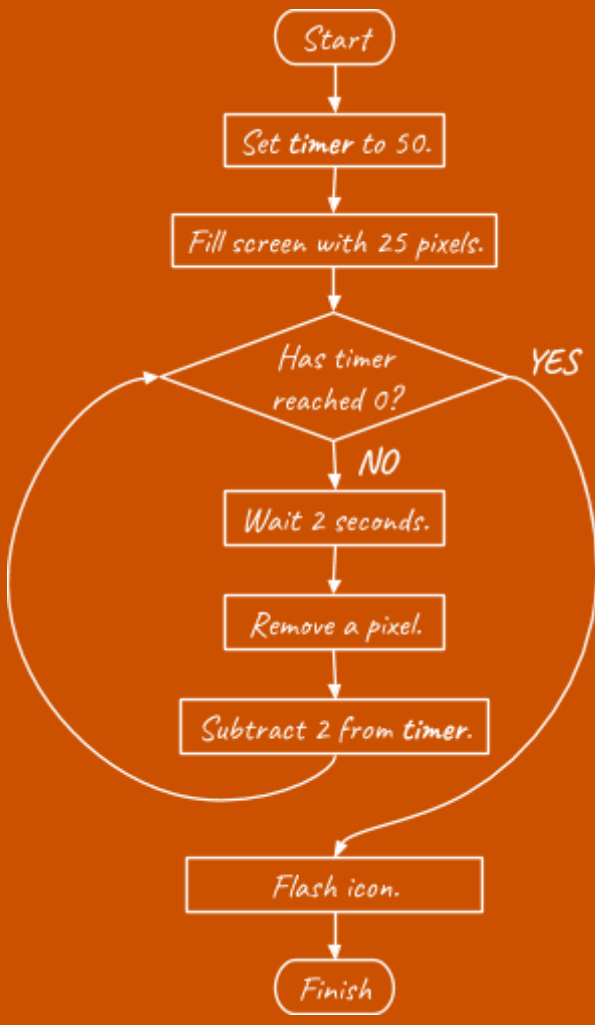
- Create a *flowchart* and/or *structured English (Pseudocode)*. Try to do this without jumping right into the real coding environment.
- Trace through your algorithm by keeping track of variable values. This can be formalised with a *trace table*.
- If your algorithm expects input data, experiment with different *test cases*.

Here's a list of code techniques and structures we've covered in MODULE 1:

- iteration (loops)
- branching (if / then)
- variables and arrays
- random numbers
- Maths operations
- functions (Python and JavaScript only)

EXAMPLE

The team prepares a flowchart. When they realise the display has 25 pixels, they decide to make the timer 50 seconds instead of 60.



Code, build and test

Now is the time to code and test the program, as well as building any physical parts.

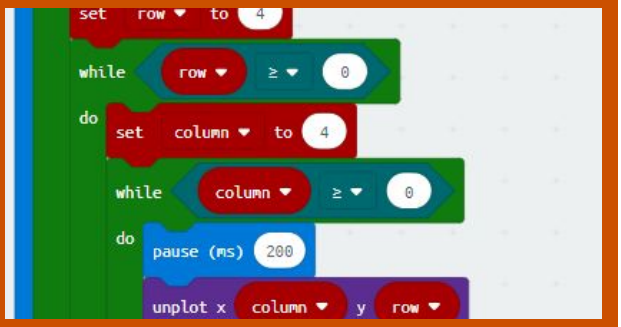
MODIFICATIONS

You may find that you must modify your design as you discover new constraints or opportunities while coding.

Just remember:

- Stick to the functional requirements.
- If you find you must alter the requirements, make note of any major changes to the design.

EXAMPLE



Ben realises that the code may need to be a little different from his team's flowchart, to keep track of the pixels while counting down.

TESTING

Try to record any informal testing you do as you develop your code.

Once the code is ready to be formally tested, prepare a *testing table* with inputs, expected output and actual output.

Include in your tests:

- unexpected user input (eg. a user presses a button at the wrong time, or for too long),
- input data values (if applicable)
    - expected values,
    - values that are out of expected range,
    - *boundary* values (on the boundary between in and out of range).

EXAMPLE

Input	*Expected output*	*Actual output*
Press button A	*Restart timer*	*✓ but only when icon is flashing*
Press button B	*Restart timer*	*✗ Does nothing.*

- Currently, only button A resets the timer. Button B can be added easily in code.
- The timer cannot be restarted until it runs out. This could limit its usefulness in a busy locker bay. This bug is a little trickier to fix in code.

Evaluate

This is the final stage. How did the solution do, and what are potential impacts to consider?

HOW DID WE GO?

Some useful questions to ask:

- Did the solution meet the functional requirements set out in the design stage?
- Was there any “creep”? Did we end up with something rather different to the design, even if it was good?
- Are there unresolved or unexpected problems with the solution?
- What data could be gathered to determine how well the solution works?

EXAMPLE

The team decides that the functional requirements have been met, but there are some unexpected problems:

- The micro:bit runs down its batteries while timing and while waiting to time.
- There is a risk of theft or damage of the device in the locker bay.
- It would be nice to have a sound as well as a visual alert, but that would require separate hardware.

IMPACT!

Lastly, consider the impacts of your solution:

- social (locally or wider society),
- environmental (this area is often ignored when dealing with high tech solutions),
- economic (jobs and livelihoods).

How does your solution compare with existing solutions?

Is it innovative, and how is that defined?

EXAMPLE

Kim suggests that, if some of the limitations of the solution could be resolved, the locker timer could have a positive social impact in the school.

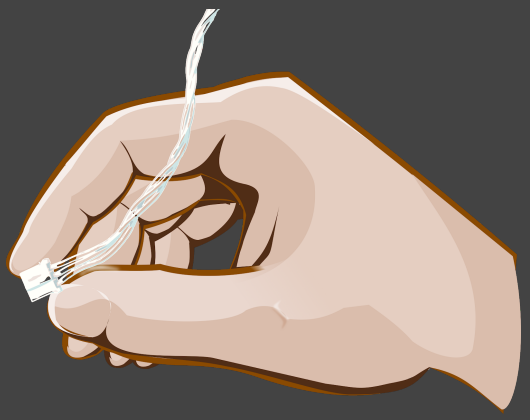
However, Abby reflects that the environmental impact is high - one micro:bit per locker, with batteries.

MODULE 2

Real-world projects with BBC micro:bit and **BOSON Starter Kit**.

THREE QUICK TIPS for Boson

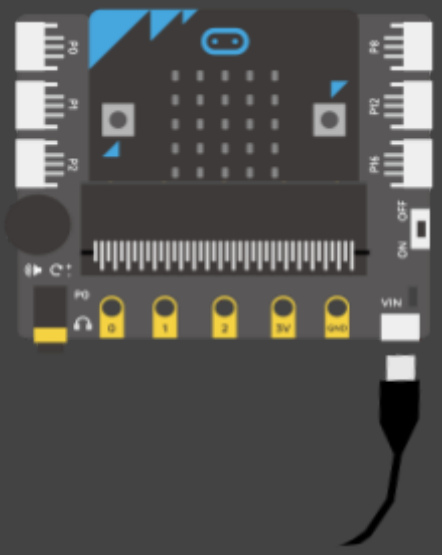
1 Never pull leads by the cord.
Always **grip the plug**.



2 Boson modules work best when USB power is supplied to the **VIN port** on the Boson Expansion Board.

USB power could come from a computer, an AC adaptor or a battery source.

(The micro:bit must still be programmed via its own USB port.)



3 **Use Alkaline** batteries.

Avoid Carbon Zinc batteries, which are usually slightly cheaper and may be marked as “Super Heavy Duty”.

(Battery power can produce different results to USB power.)

ACTIVITY 2.1 - Clap switch

Goals

- Create a hand clap-activated light and a fan for a home model.
- Test the Boson Sound Sensor and LED module.
- Test the Boson Rotation Sensor and Fan module.
- Use a **Boolean variable**.

JavaScript
parallel document

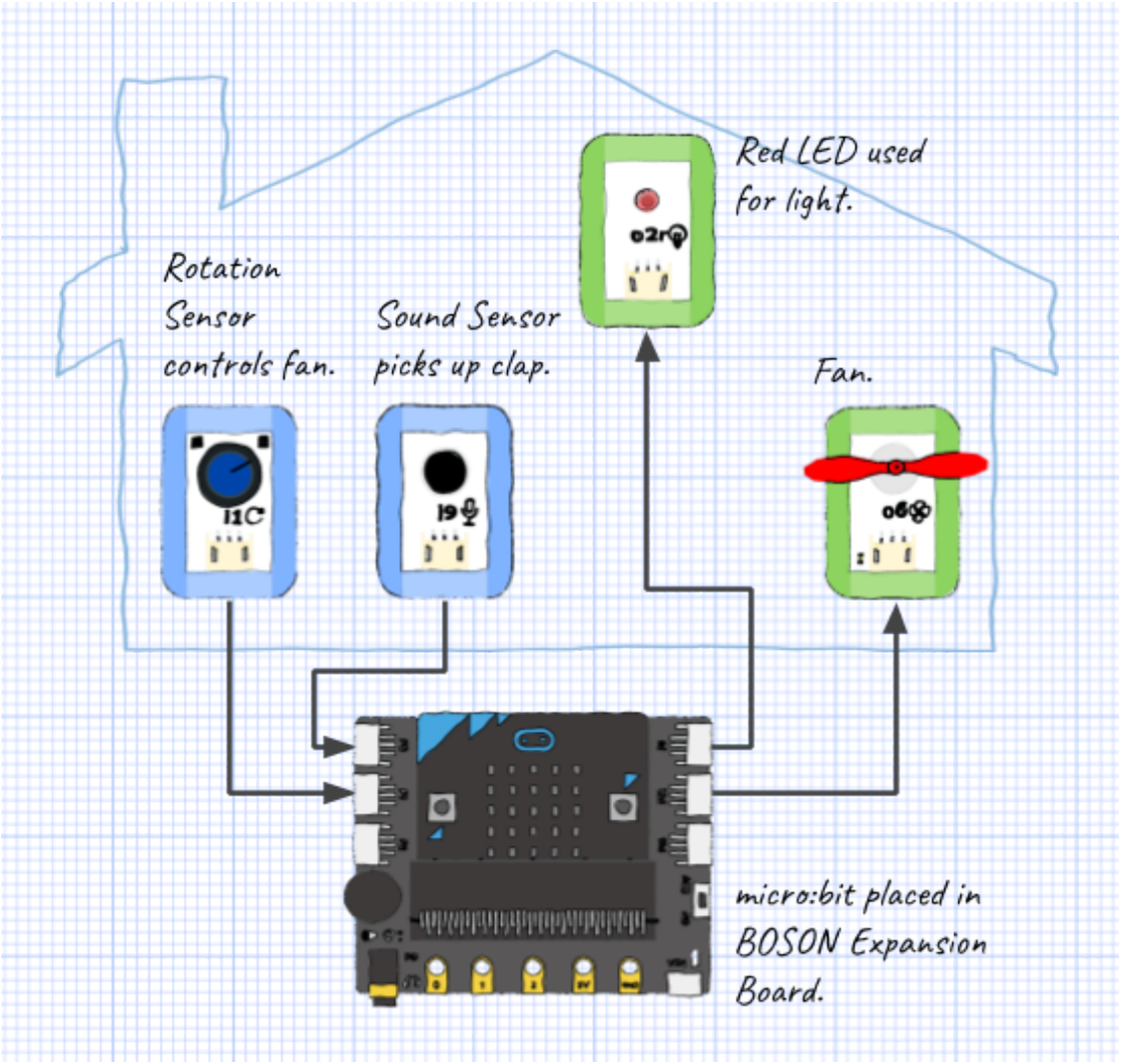
Python
parallel document

BUILD

Design overview

We will build a model for a home with:

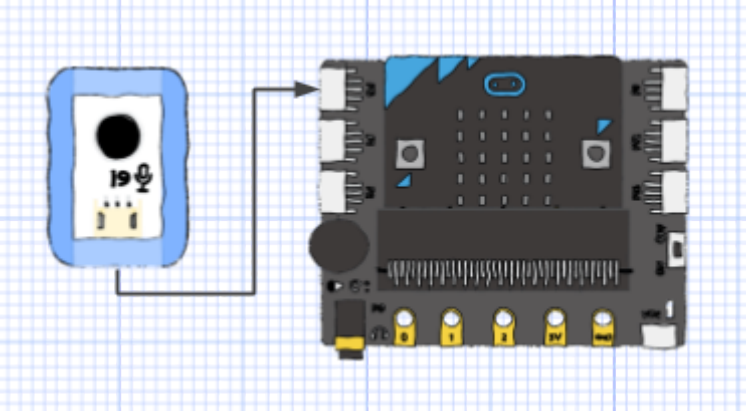
- a light that can be switched on or off by clapping hands,
- a fan controlled by a knob (rotation sensor).



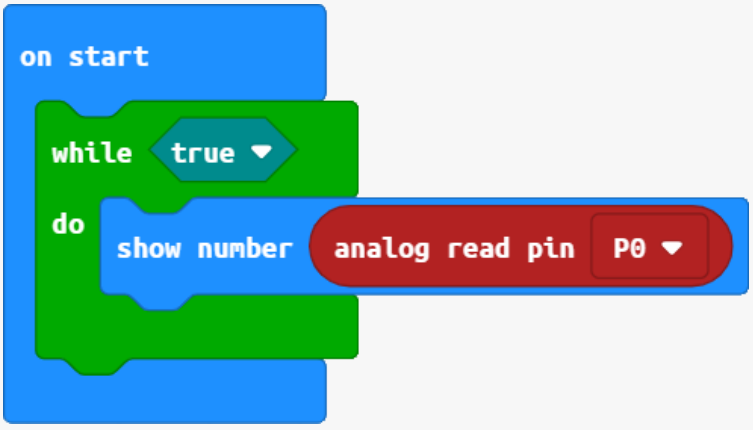
Testing the Sound Sensor

1. We need to find out the kind of values we will get from the Sound Sensor.

   Attach it to **P0** on the Boson Expansion Board.



2. Make a loop to keep getting the Sound Sensor reading on **P0** and scroll it on screen. (Scrolling takes time, so short sounds may not register if poorly timed.)



Visual

Analog input

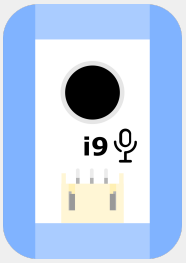
The Sound Sensor makes a voltage level that varies from 0V up to a maximum, not just on/off.

The micro:bit can read a number from 0 to 1023, but the real maximum value depends on the sensor and the power source.

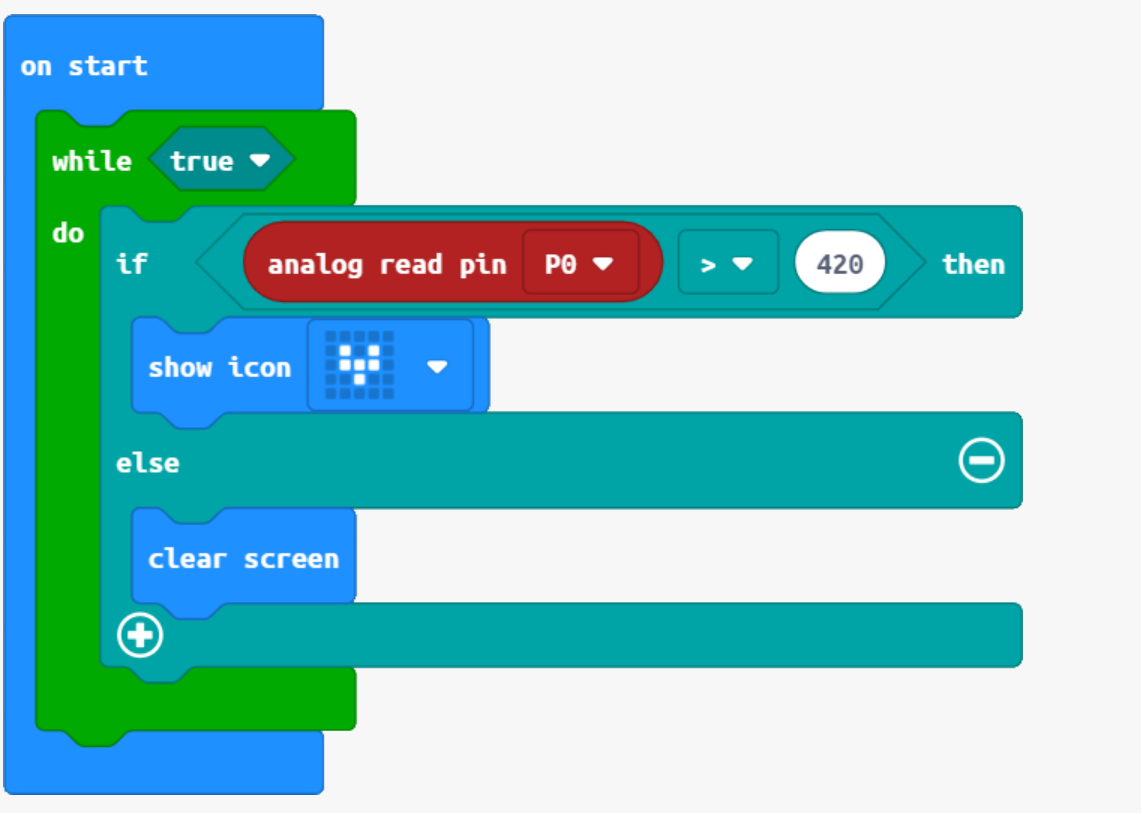
3. Note down the typical number you get without clapping (eg. 250), and with clapping (eg. 420).

The **Boson Sound Sensor** picks up noise in the environment, similar to a microphone.

In a noisy classroom, try to find the number value to distinguish background noise from a clap.



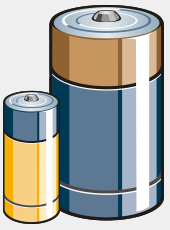
4. Write a loop to detect a clap. If the reading is higher than the number you determined, show a heart, otherwise clear the screen.



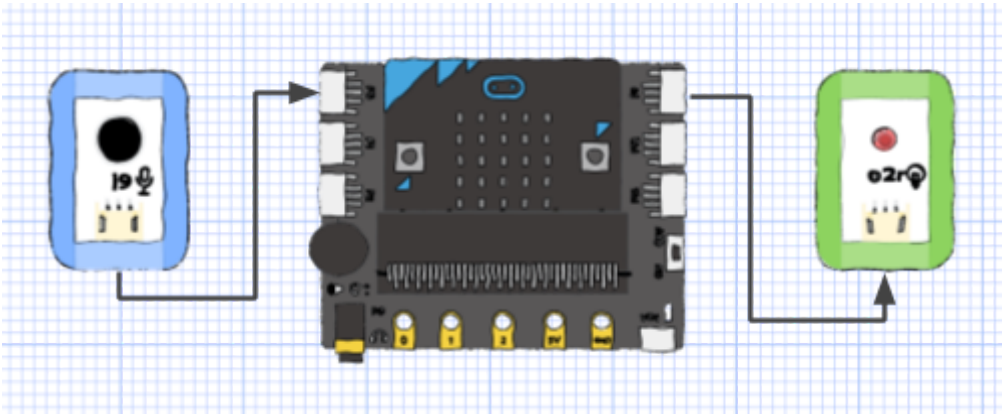
Boson power

The most reliable solution is to power everything via the **VIN** USB socket on the Boson Expansion Board. See THREE QUICK TIPS.

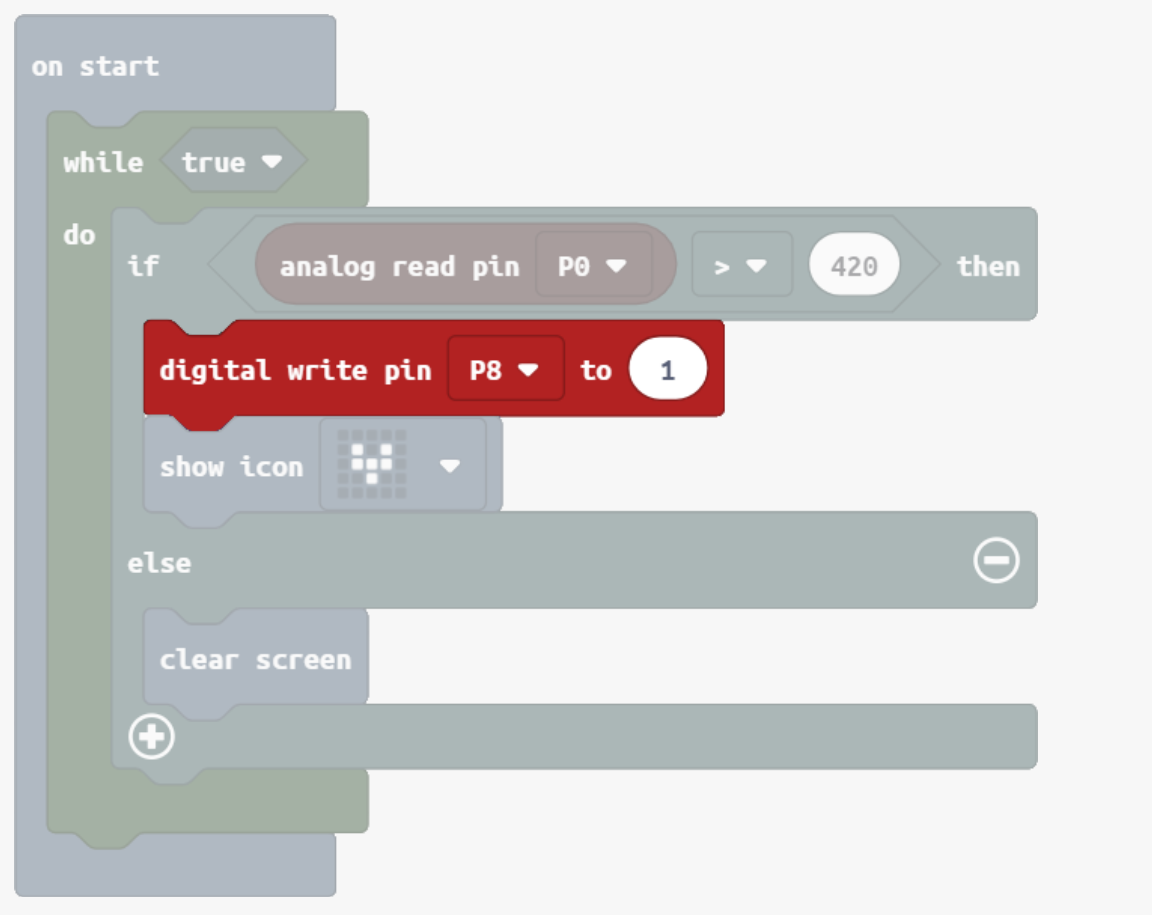
Results may vary if the micro:bit's power is used.



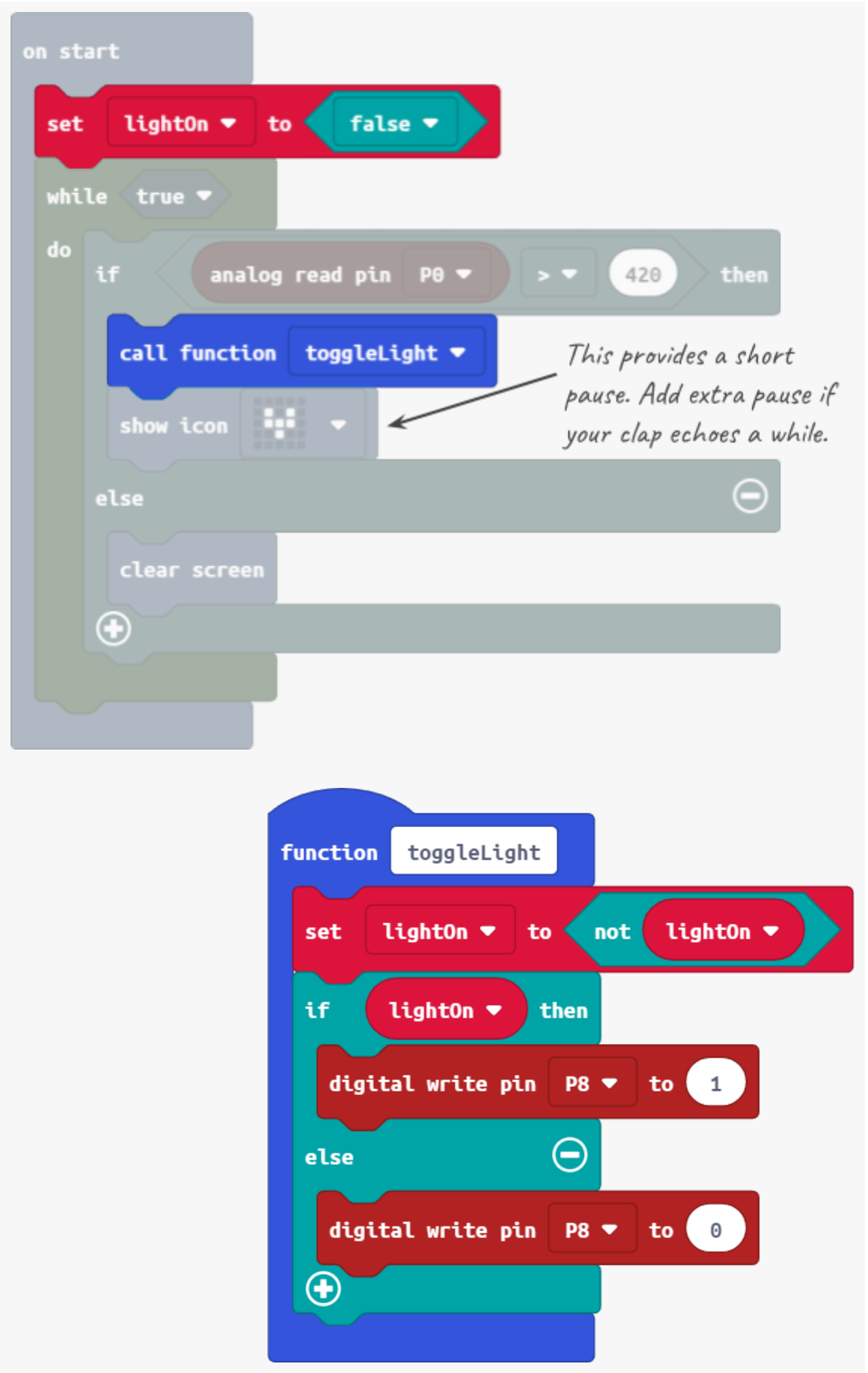
5. Now attach the Red LED module on **P8**.



6. Turn the light on when a clap is detected.



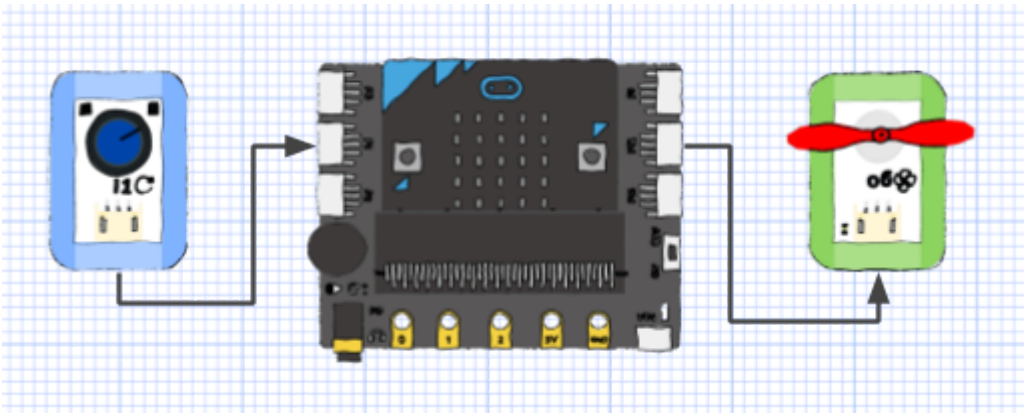
7. We want to *toggle* the light with a clap. Make a variable **lightOn** to remember whether the light is on or off, and a function **toggleLight** to flip it on or off.



Controlling the fan

8. Connect the Rotation Sensor to **P1**.

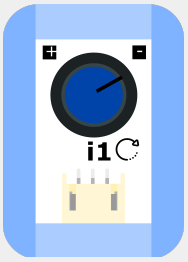
   Connect the Mini Fan Module to **P12**.



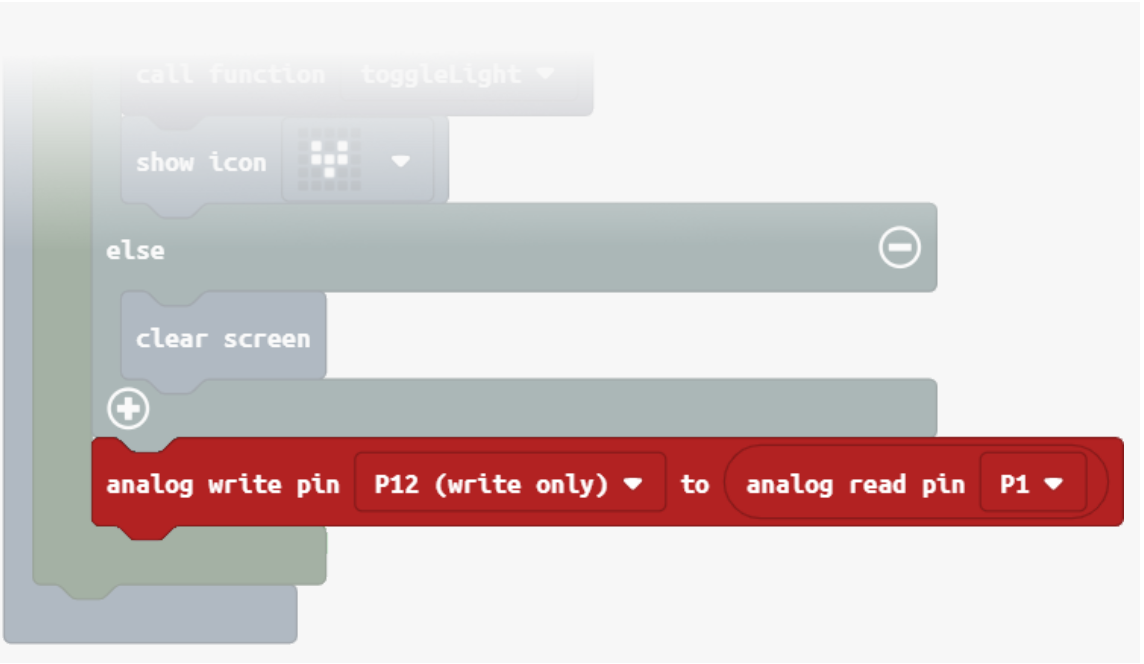
The **Boson Rotation Sensor** gives an analog voltage, like the sound sensor. But more reliable!

Turn it clockwise to increase the voltage.

The micro:bit should get a reading from 0 to 1023. How could you confirm the numbers?



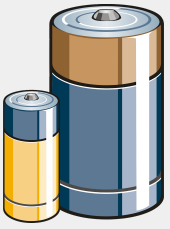
9. Inside the loop, use the reading from the sensor as the power value for the fan.



Driving the fan

A small voltage from the micro:bit is just enough to drive the Boson Fan motor.

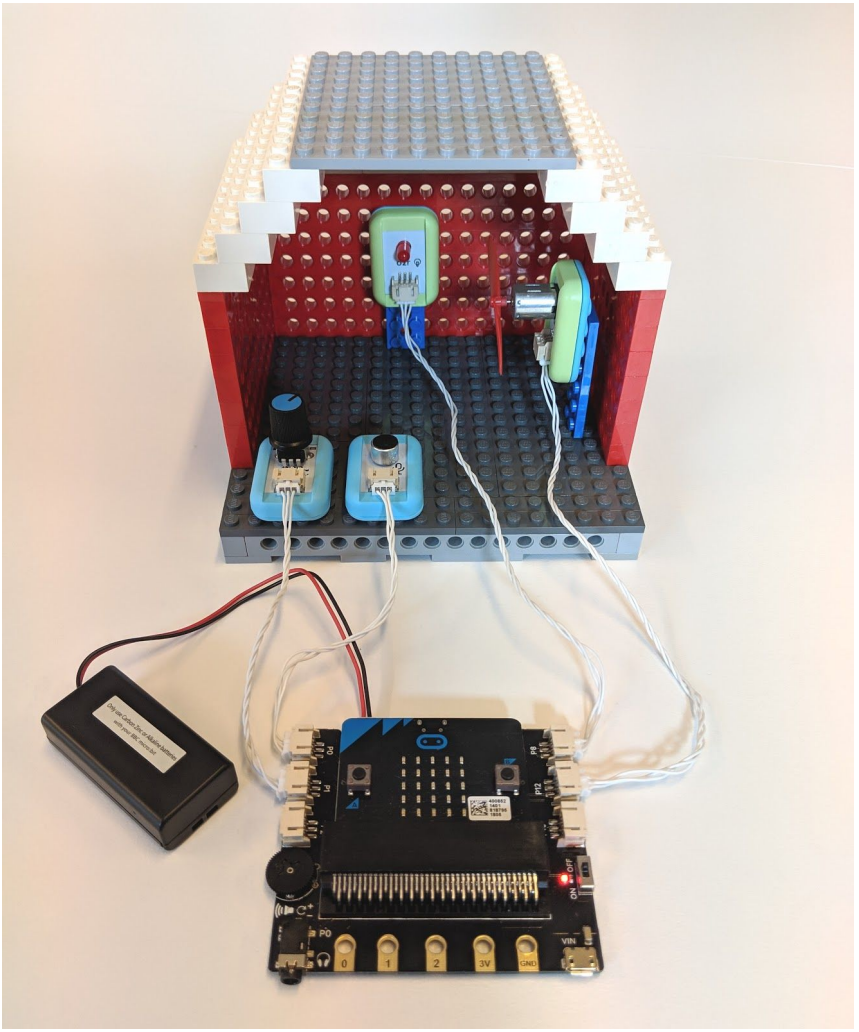
If running on batteries, you may need to give the fan a little tap to get momentum going.



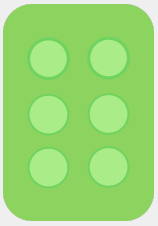
Make the house!

10. Use materials of your choice to build a house with the sensors and modules installed inside.

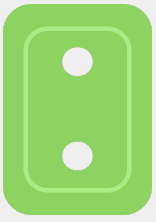
    Here's a LEGO construction for this model:



Boson modules come with 2 magnetic backpieces to use when building:



LEGO-compatible backpieces for attaching to LEGO bricks.



Screw-hole backpieces contain holes for securing with a screw and nut.

Tweaking the system

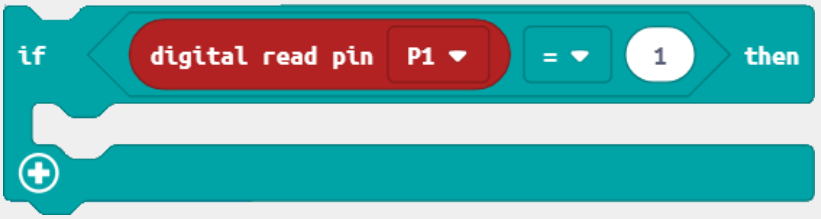
TINKER

A. Try swapping the fan module and the LED module. When done, swap them back again.

B. When no clap is happening, use the micro:bit's 25 pixels to display a chart of the current sound level.

C. Remove the rotation sensor and replace it with the push button for activating and deactivating the fan. You'll need to edit the code so that one push means on and another push means off.

The **Boson Push Button** is a digital sensor (pressed or not pressed). Use the digital read command and check for 1 or 0.



HINT: Once you detect a press, you'll need to wait for a release.

D. Add the rotation sensor to **P2**. Edit your program so that the rotation sensor is used to adjust the sensitivity for detecting claps.

Home automation

JUMP OFF

Modern home automation is little more than combining inputs and outputs to a system.

- More improvements to our house:
    - **Remote controlled** - Use the radio function from a second micro:bit to control the light or fan in the house.
    - **Double-clap** - A single clap toggles the light, but a double-clap toggles the fan. This is trickier to code than it sounds. *(HINT: First count the claps, then take action after).*
    - **Night light** - Use the micro:bit's light sensing to switch the house light on after dark, and turn it off in the presence of bright light.
- **Eco-aware apartment** - Use a timer on the micro:bit to turn *all* appliances off when no one is detected in the room for 60 seconds. Connect the Boson Motion Sensor to make this work.
- **Earthquake detector** - Combine the inputs from the sound sensor with the micro:bit's accelerometer to detect vibrations. Use the rotation sensor for calibration.

ACTIVITY 2.2 - Smart scarecrow

Goals

- Create a mechanical scarecrow that flaps its arms when movement is detected.
- Test the Boson Motion Sensor.
- Test the Mini Servo.

JavaScript
parallel document

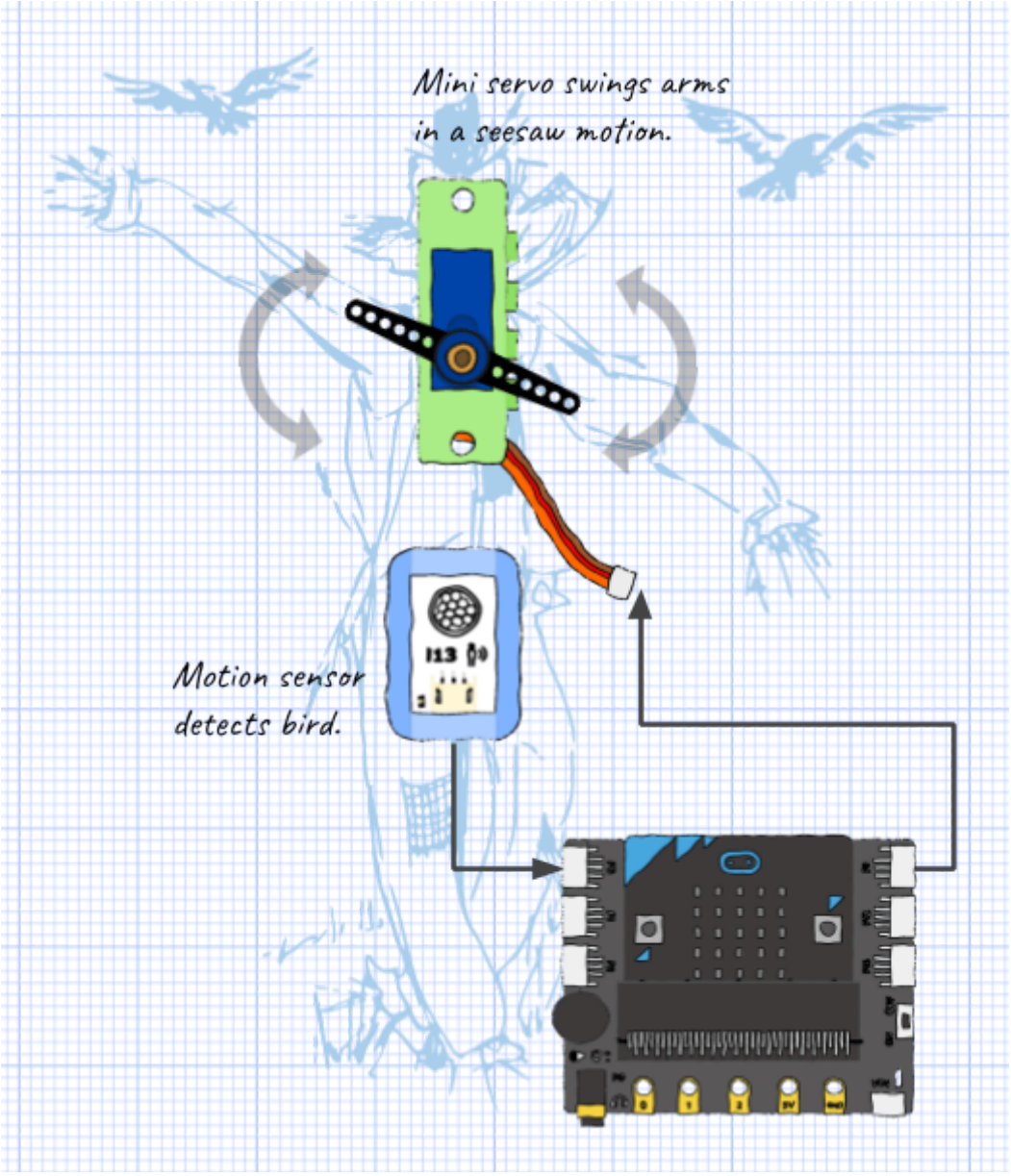
Python
parallel document

BUILD

Design overview

We will build a model for our scarecrow with:

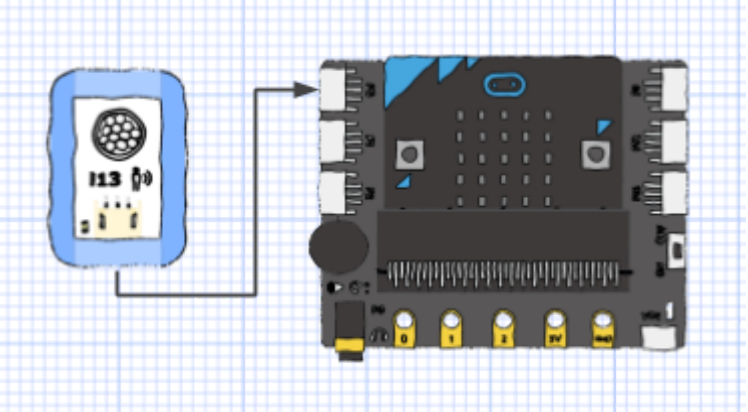
- a motion sensor to determine if a bird is present,
- moving arms that seesaw to scare away the bird.



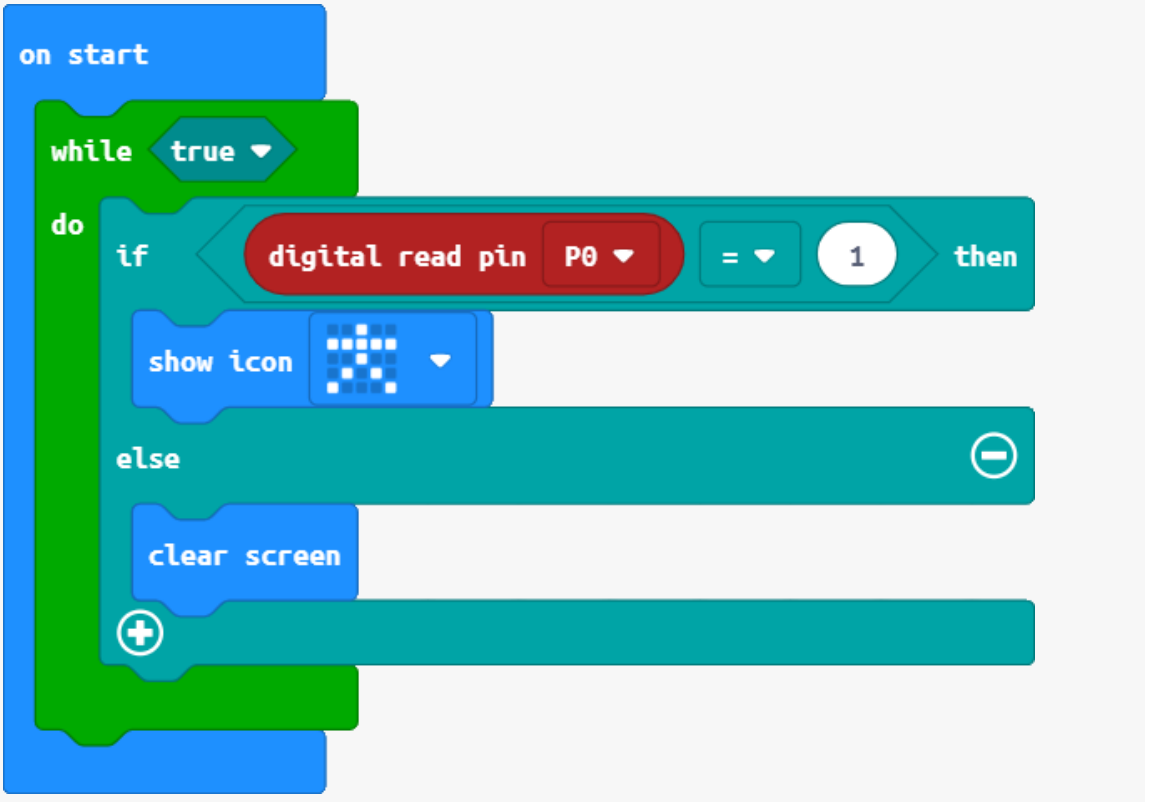
Investigating the Motion Sensor

1. We need to find out how sensitive the Motion Sensor is.

   Attach it to **P0** on the Boson Expansion Board.



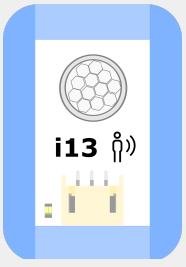
2. Make a loop to show an icon when the sensor fires. Test how sensitive it is. How far away before it doesn't detect motion?

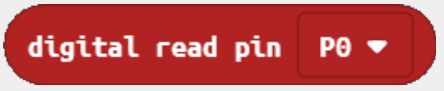


The **Boson Motion Sensor** is a Passive Infrared (PIR) sensor.

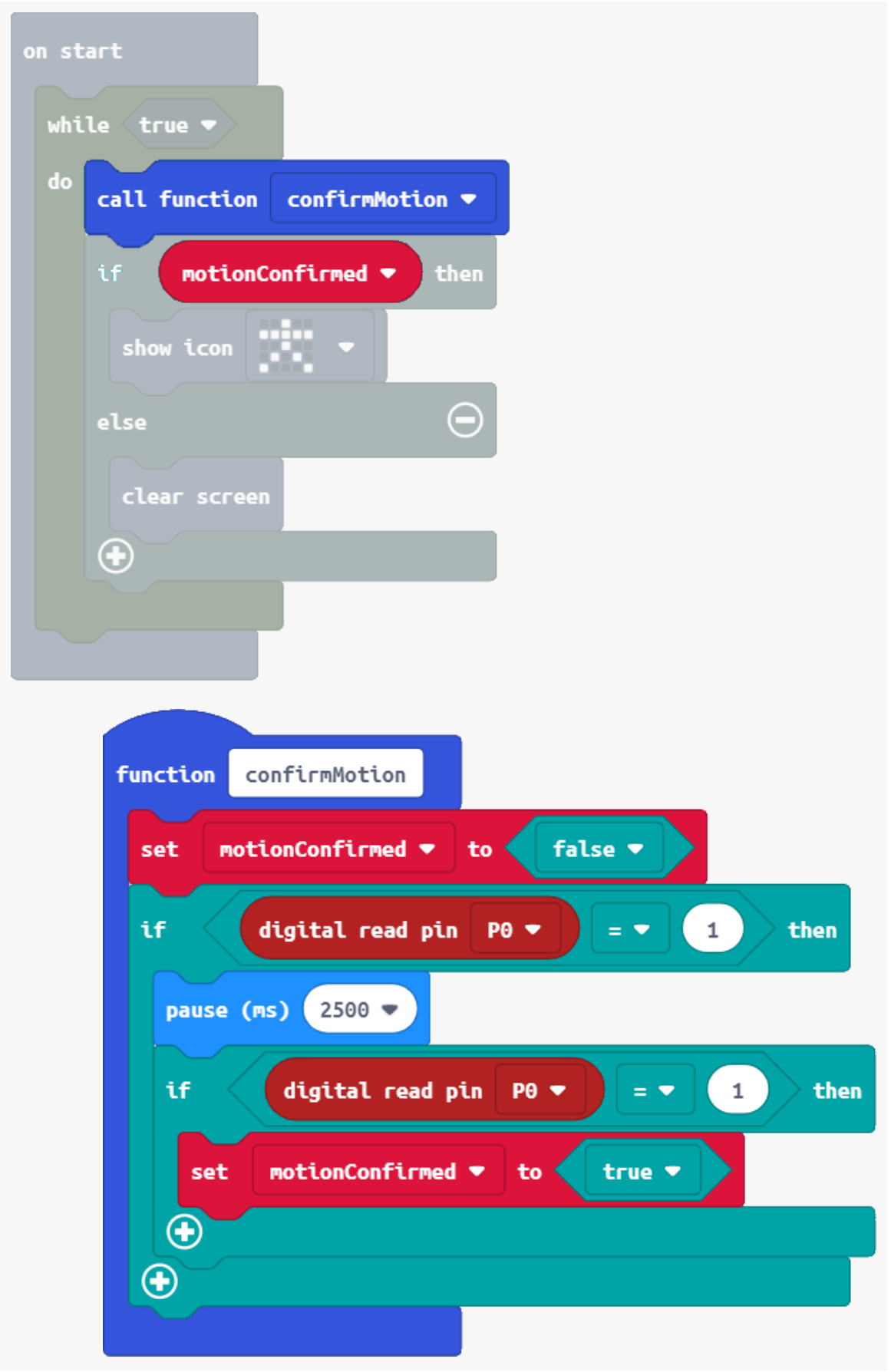
It detects movement by measuring a change in infrared light, such as from body heat.

The Motion Sensor can be ON (full voltage), or OFF (no voltage). It gives a **digital** value (1 or 0).



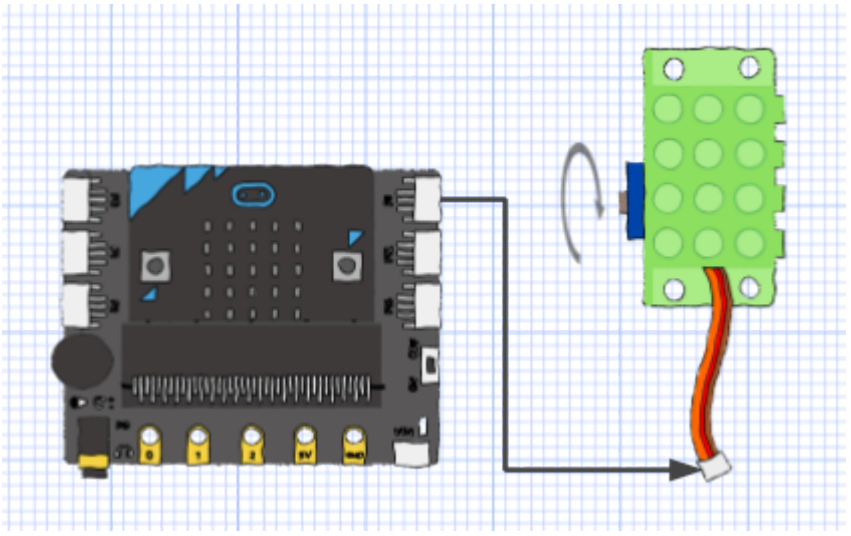


3. Write a function to screen out isolated readings (less than 2.5 seconds long). These readings may be erroneous.



Testing the Mini Servo

4. Attach the Mini Servo to **P8** on the Boson Expansion Board.

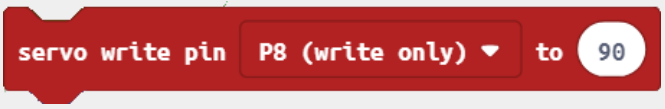


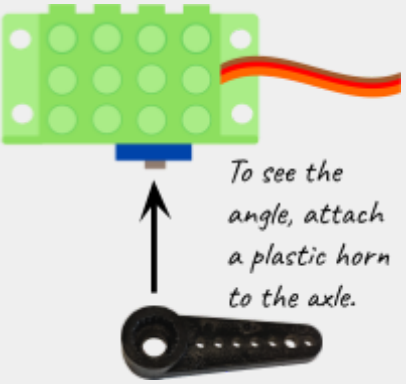
The **servo motor** is different from a regular motor.

Instead of setting a speed, you set an angle (from 0° to 180°). The axle rotates once to that position.

The servo motor expects a repeating voltage pulse to set its position (Pulse Width Modulation).

This special command activates a pulse on **P8**.
It drives the servo to the 90° position:



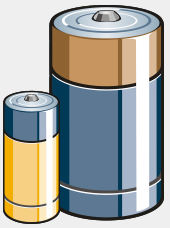


5. Write a program to test swinging the axle back and forth from 0° to 120°.

Boson power

The most reliable solution is to power everything via the **VIN** USB socket on the Boson Expansion Board. See THREE QUICK TIPS.

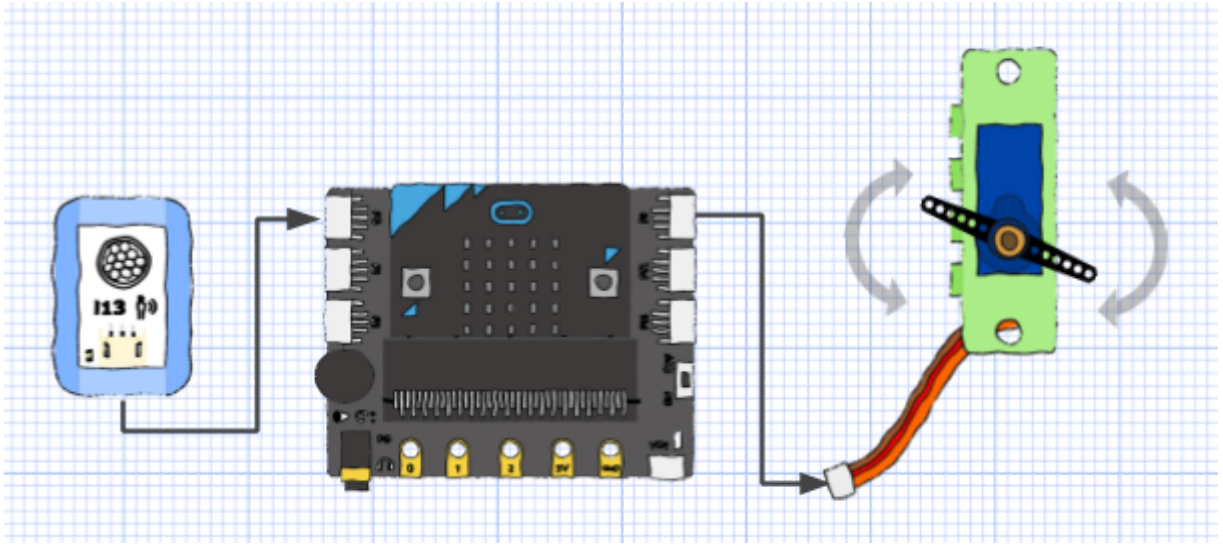
Results may vary if the micro:bit's power is used.



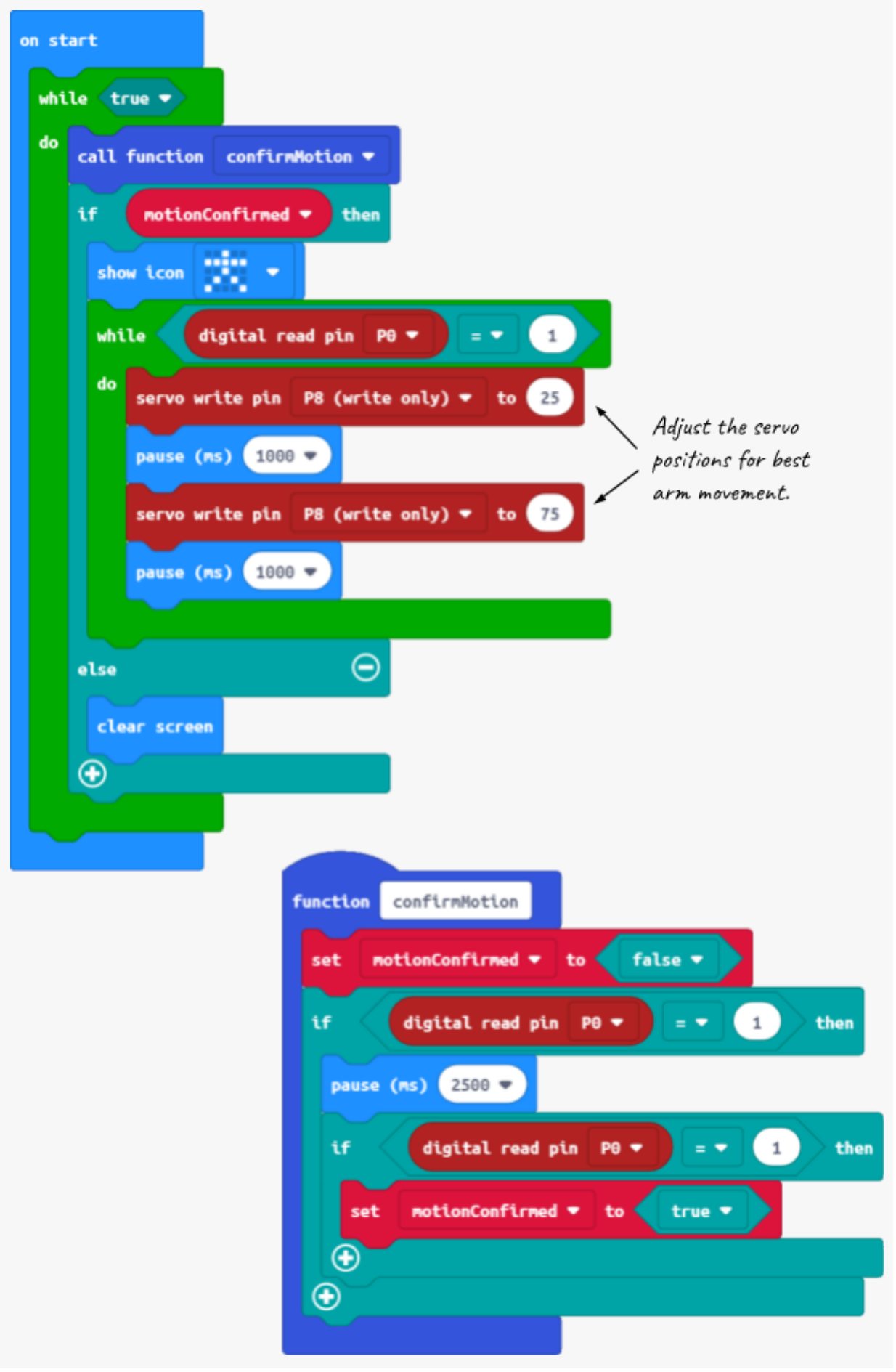
If I only had a brain

BUILD

6. It's time to bring the parts together. Attach both modules as below:

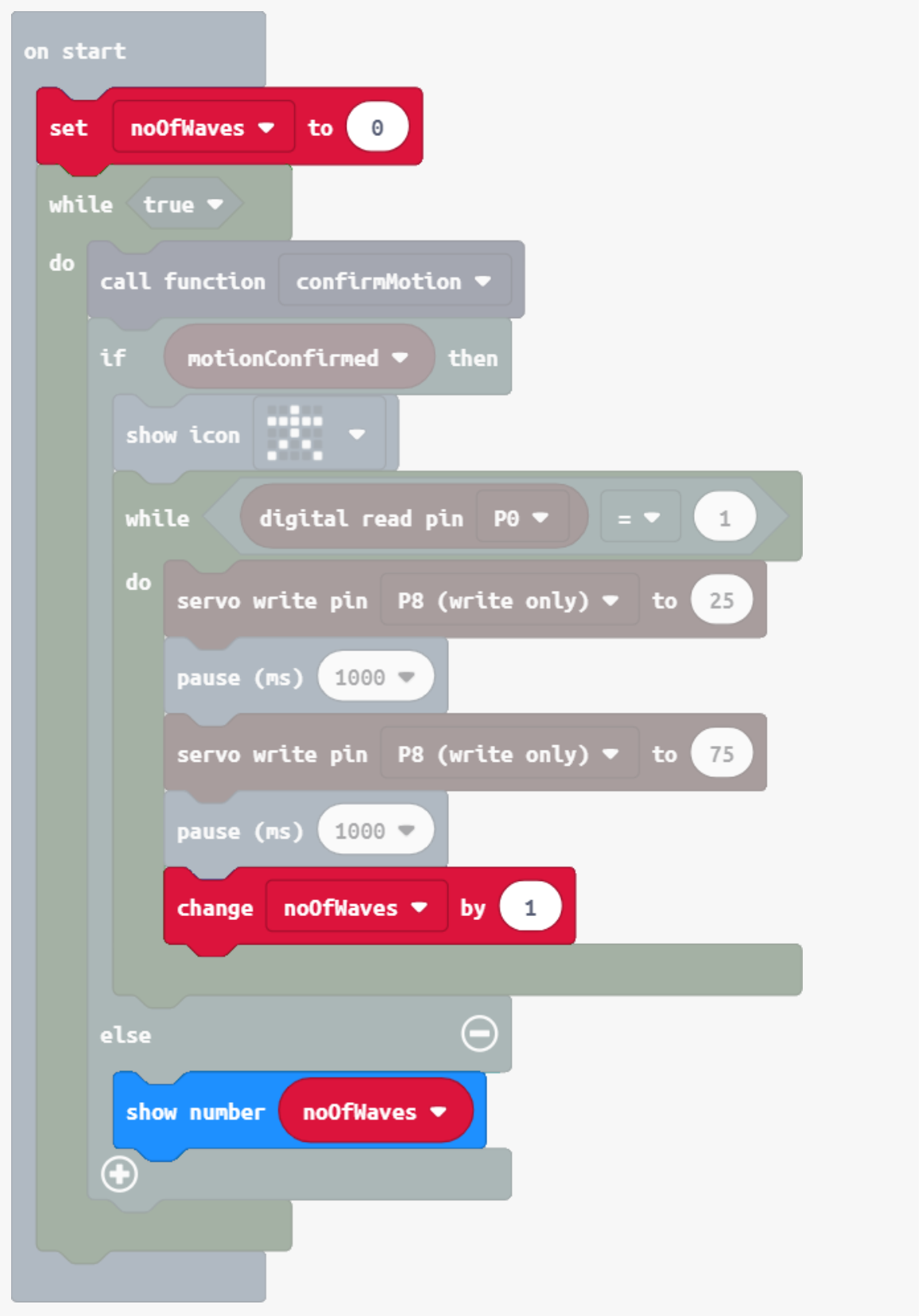


7. Combine the previous code to swing the arms as long as a confirmed motion is ongoing.



8. Collecting data is one way to see if the scarecrow is effective.

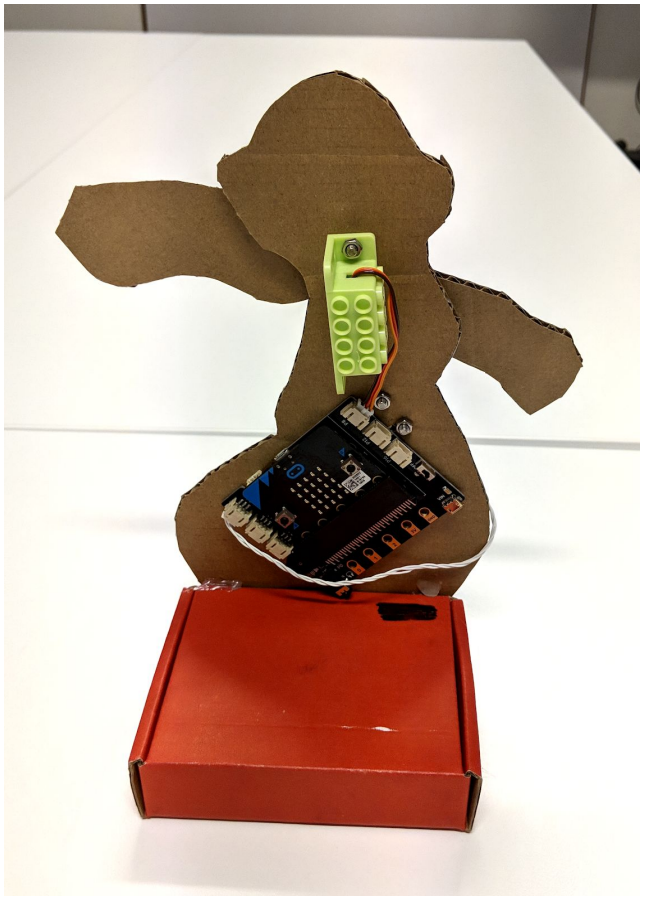
   In the main routine, add a counter for the number of times the scarecrow waves its arms.



Make the scarecrow

11. Use materials of your choice to build the scarecrow with modules attached.

    Here's a cardboard construction for this model.



Brainier

A. The arm waving is too slow and predictable. Can you make it more random?

B. Some scarecrows are designed with laser lights. Add the Boson LED Light to your scarecrow's head, and have it flash when the arms are waving.

C. A farmer wants the scarecrow to deactivate at night so that owls will stay. The micro:bit has a sensor on-board that is perfect for this.

D. Modify the code so that the counter increases only once for each new confirmation of motion.

Serving up movement

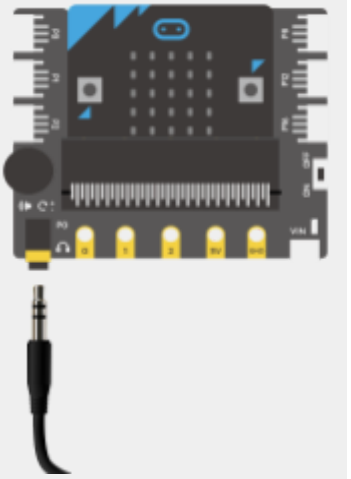
Servo motors make projects more exciting!

- More improvements to our scarecrow:
    - **Alarm bells** - Some research suggests that sounds can also be effective in scaring off birds. Use music commands to play tunes or tones when the scarecrow is moving.

Audio on P0

The Boson Expansion Board has an audio port for headphones or speakers, but it always uses **P0**.

You will need to move other sensors to **P1** or **P2**.



    - **Crowd scaring** - Other research suggests synchronised movement can help scare off birds. Use radio commands to fire off multiple micro:bit scarecrows at once.

- **Oscillating fan** - Use the Mini Servo to create a rotating platform for the Boson Mini Fan to sit upon. Can you figure out how to make it turn slowly?

- **Sunrise-to-sunset clock** - Create a fanciful circle artwork that slowly rotates 180° from sunrise to sunset over the course of the day. Could you enhance it with a button for resetting at dawn? Or even use the micro:bit's light or temperature sensing?

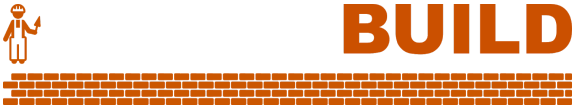
ACTIVITY 2.3 - Active music display

Goals

- Create a bright vertical volume display.
- Test the RGB LED strip.
- Work with a **Neopixel library**.
- Use Maths to adjust values from the Boson Sound Sensor.

JavaScript
parallel document

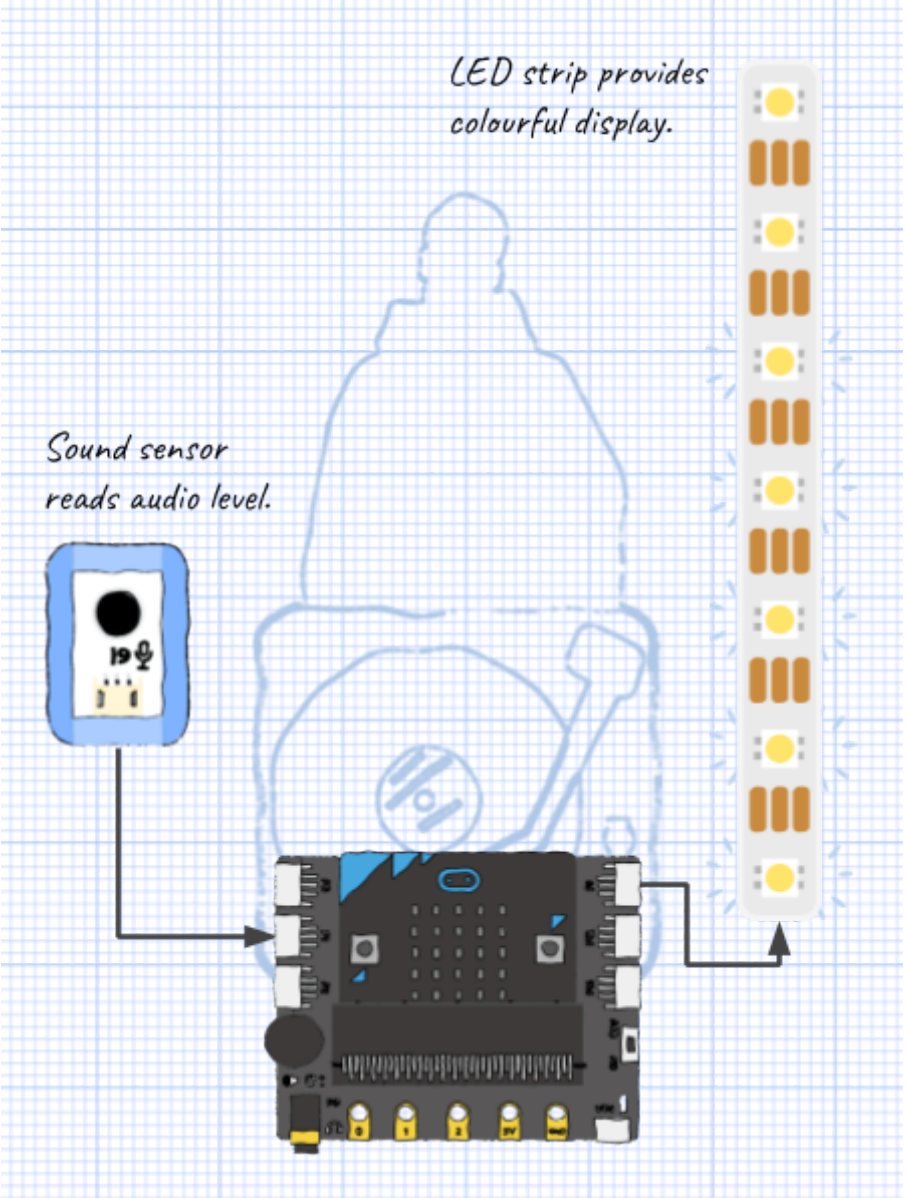
Python
parallel document



Design overview

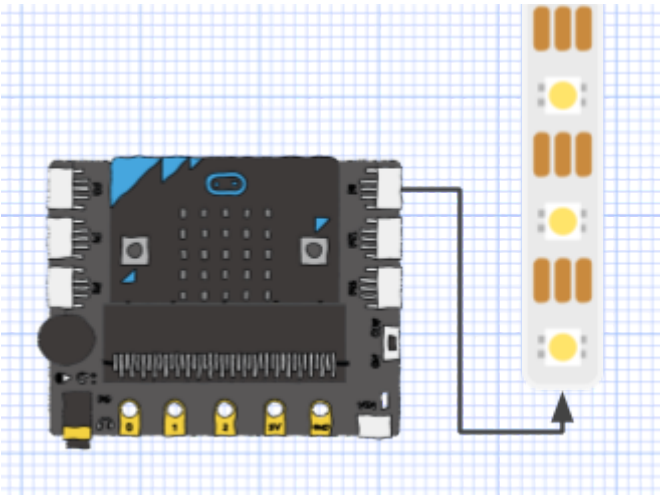
We will build a model DJ desk with:

- a sound sensor to read music volume,
- an RGB LED strip to display the sound actively.

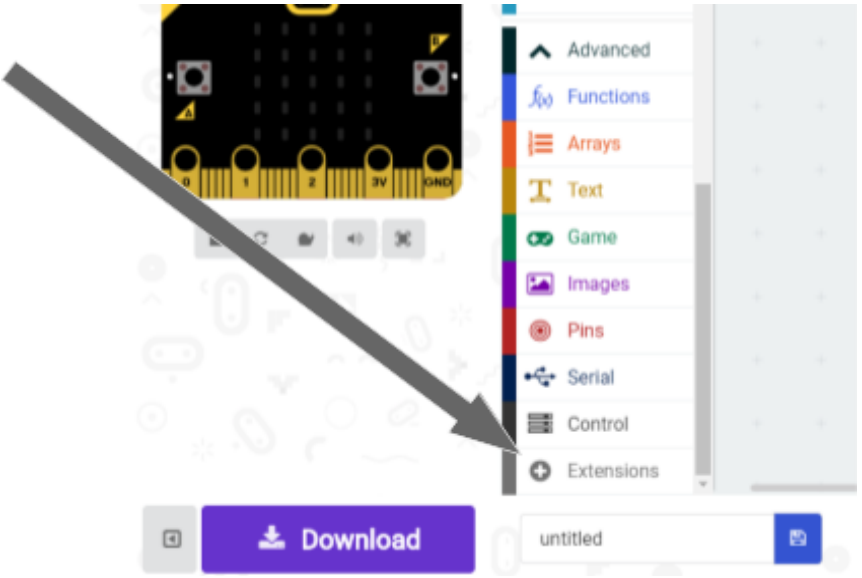


Exploring the LEDs

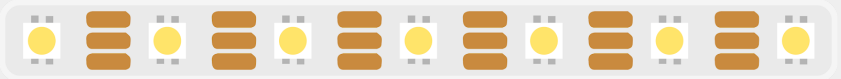
1. Connect the LED RGB Strip to **P8** on the Boson Expansion Board.



2. In the online editor, click **Extensions**, then choose **neopixel**.



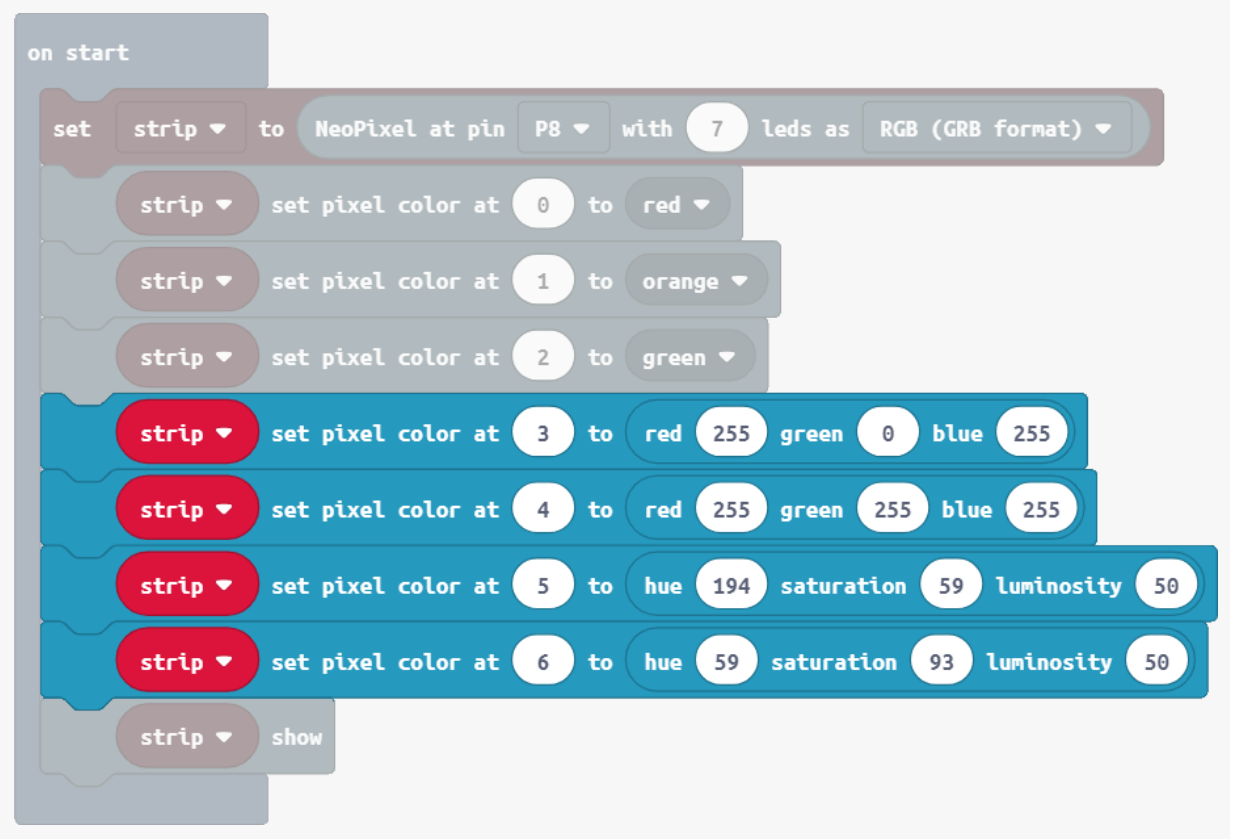
The **RGB LED Strip** uses the NeoPixel control protocol so that many lights can be controlled with just one lead.



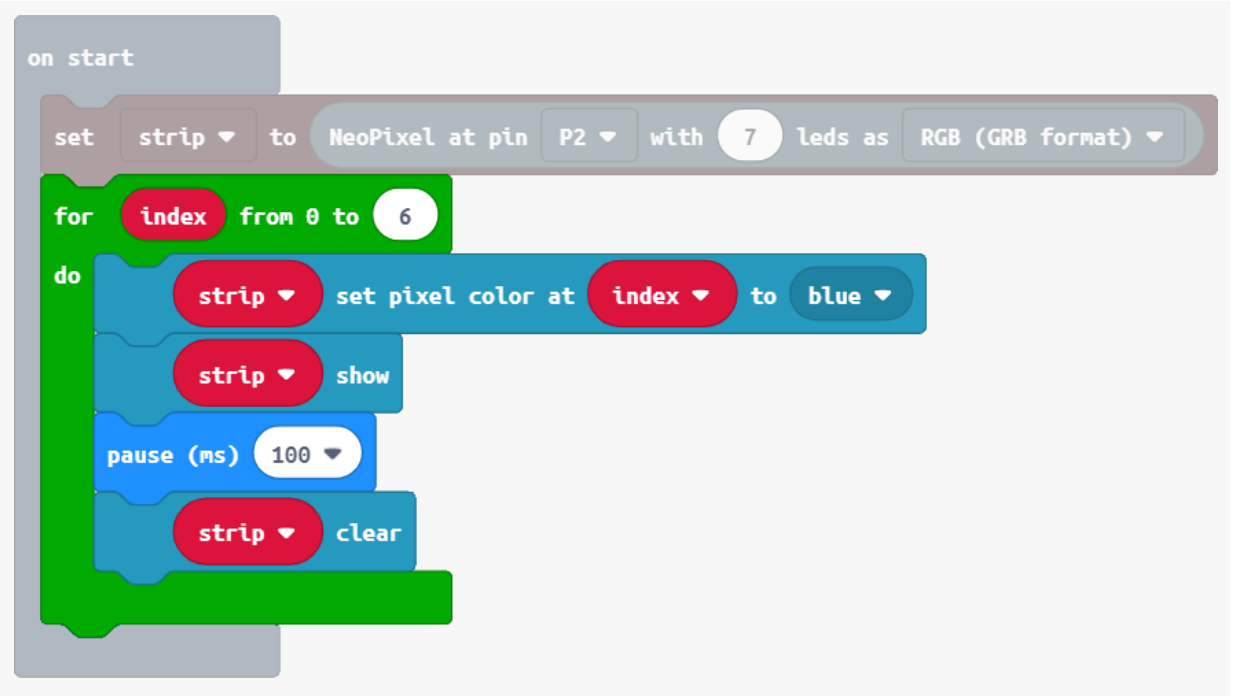
3. Our strip has 7 LEDs. Set up the neopixel control on **P8**.

4. To light up the first three LEDs, set the colours, then use the **show** command.

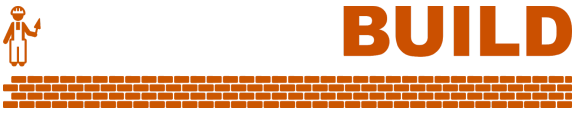
5. You can also make your own colours by combining Red, Green and Blue, or by combining Hue, Saturation and Luminosity.



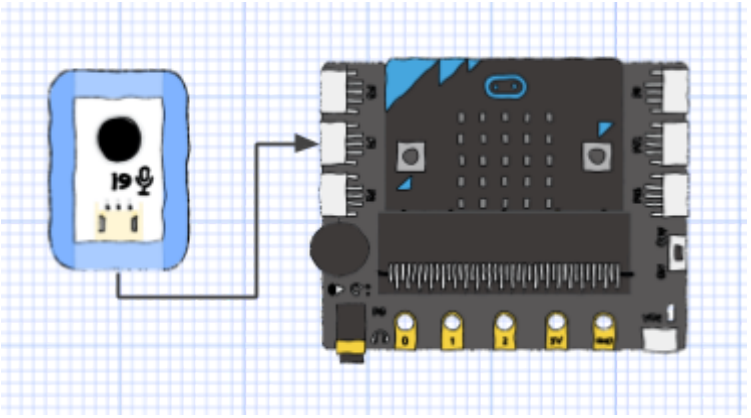
6. Here's a simple animation using a loop index to set each LED. Can you make it:
    - faster or slower?
    - repeat?
    - go back and forth?
    - do random colours?



Finding the sensor limit



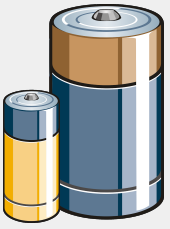
7. Now attach the Sound Sensor to **P1**. (We'll keep P0 free in case we want to make sound.)



Boson power

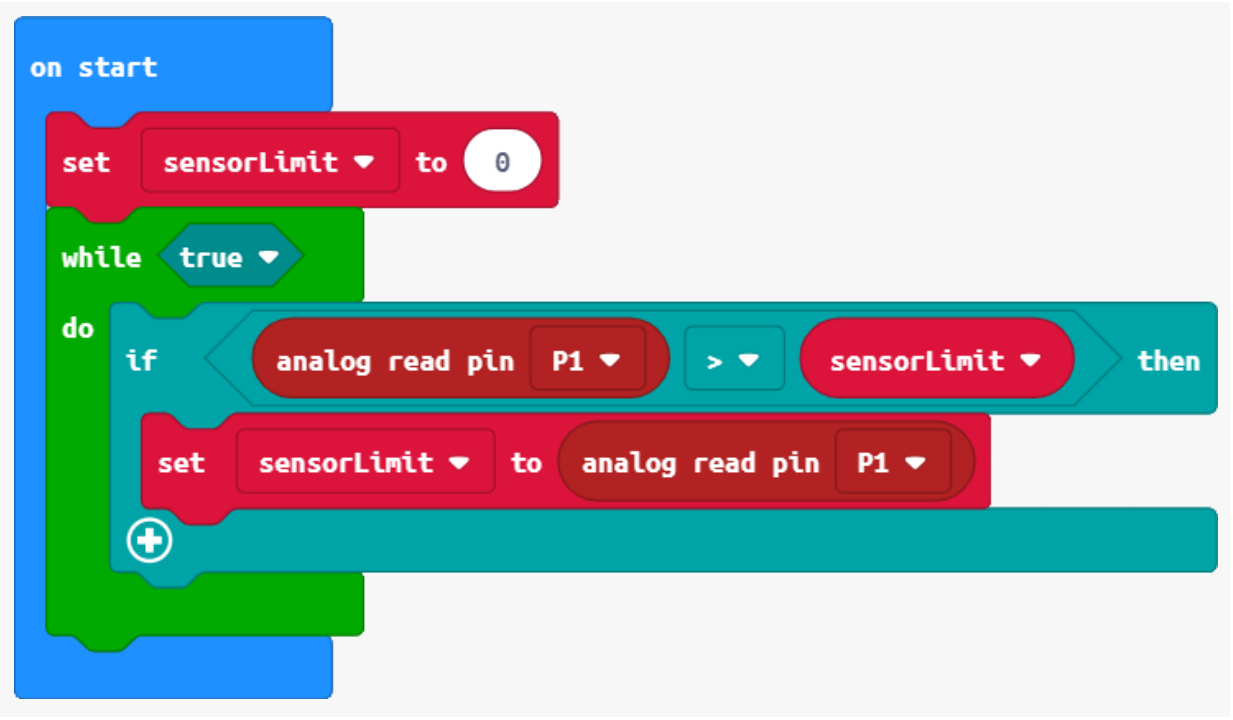
The most reliable solution is to power everything via the **VIN** USB socket on the Boson Expansion Board. See THREE QUICK TIPS.

Results may vary if the micro:bit's power is used.

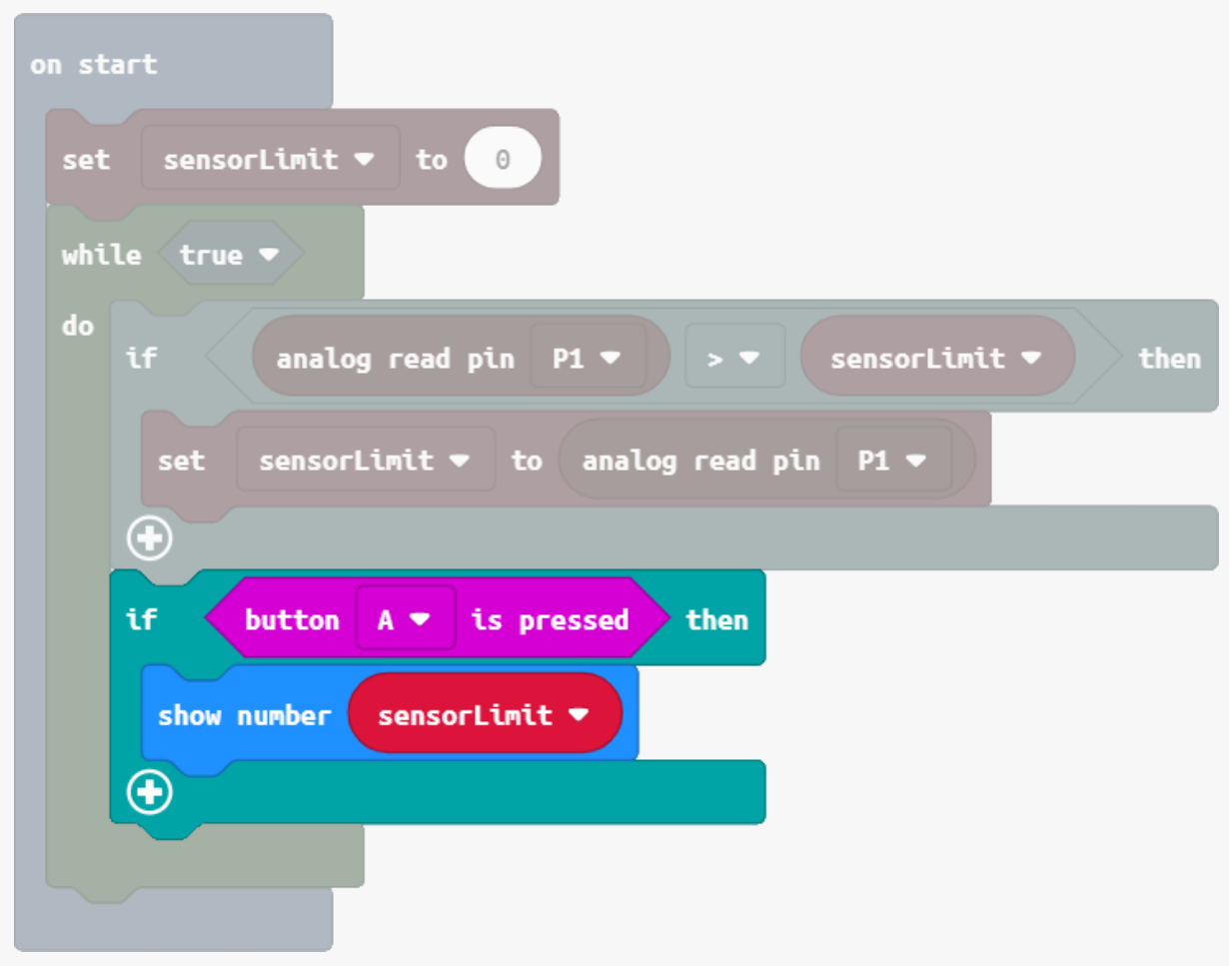


8. We know that the micro:bit reads between 0 and 1023, but we need to know the highest possible value the sound sensor can give.

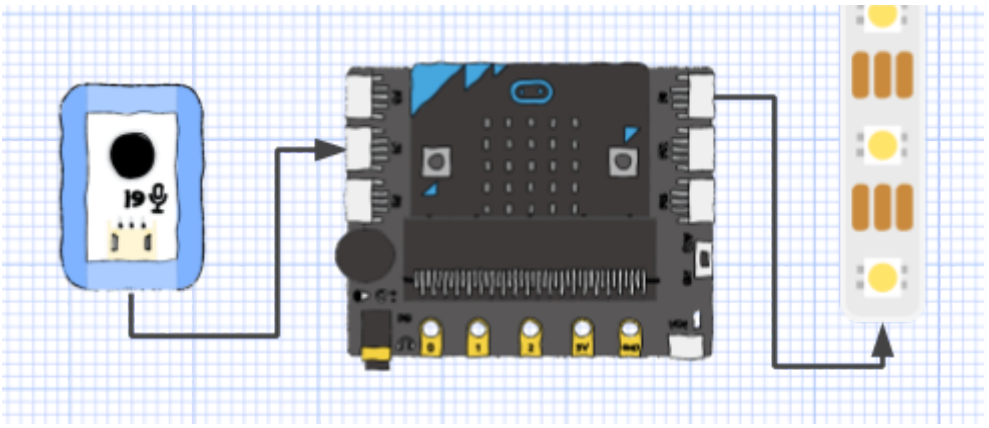
   Make a loop to learn and hold onto this upper limit. (A clap should be enough to reach it.)



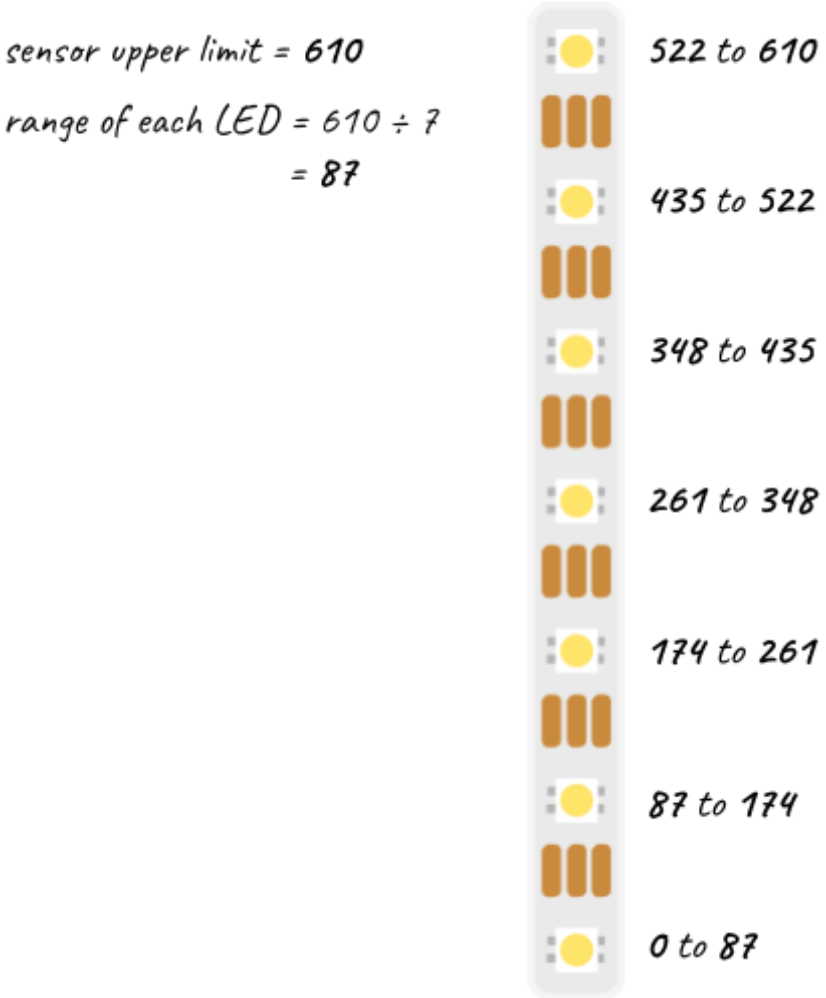
9. Use button A to display the upper limit. Write it down. (It may be around 600.)



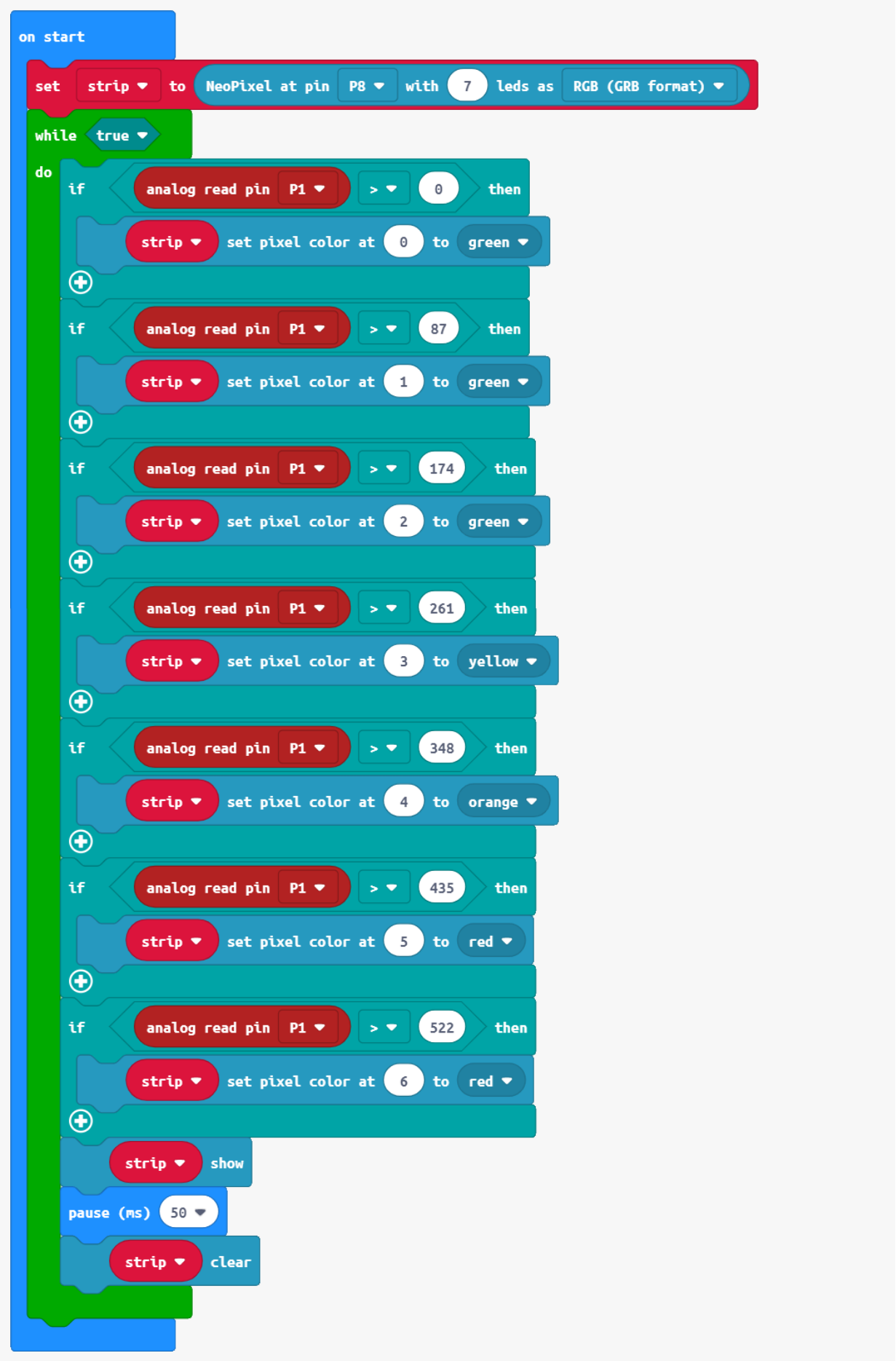
10. Here's the setup to code our vertical music display.



11. The strip has 7 LEDs, so divide your sound sensor's upper limit by 7 and find the range for each LED. eg:



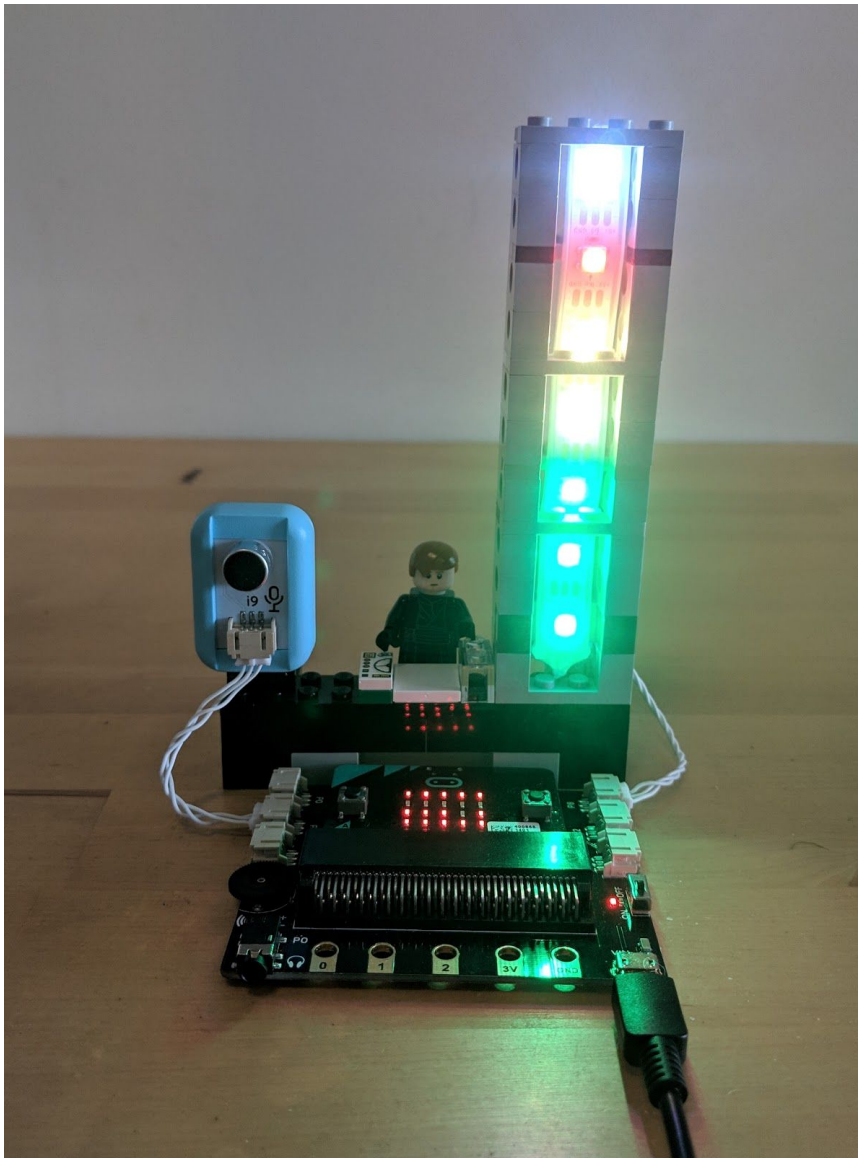
12. Here's the complete program. The LEDs go from green to red as they near the top.



Make a structure

12. Use materials of your choice to build a firm structure for your LED strip.

    Here's a LEGO construction with a DJ.



Cooler colours

A. Change the code so that all the lights are off when the environment is reasonably quiet.

B. Make the brightness of the LEDs change with volume too, giving a pulsing effect.

C. Use button A to toggle the new brightness effect on and off.

D. Mirror the volume display using the built-in bar graph command on the micro:bit LEDs.

Lights for work and play

The LED strip is a fun way to display measurements, or just pretty colours!

- More improvements to our music display:
    - **Colour alternative** - Change the way the colours work so that the green pixels turn orange when the orange levels are reached, and similarly for the red levels.
    - **Lingering rider** - Professional volume displays often include a lingering maximum - a single LED that stays for a couple of seconds on the most recent maximum value.
    - **Phat display** - DJ Bosonalot likes to change the display at times during the show. Write code so that Button B toggles to a rainbow cycle mode, which ignores volume.
- **Pretty thermometer** - Use the LED strip to display the temperature reading from the micro:bit.
- **Bouncing LED** - Use the Boson Push Button to bounce a pixel up the strip. It falls back down on its own. Perhaps you could use the micro:bit's accelerometer to make a paddle-ball toy.
- **Health bar** - Use the LED strip as a health bar for a game. All pixels are green when healthy, but they change towards red as the bar drops.

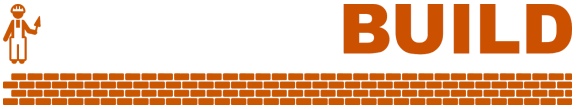
ACTIVITY 2.4 - Sound effects board

Goals

- Build an old-school sound effects board.
- Learn how to generate tones with square waveforms.
- Use multiple inputs to control sound effects.
- Implement a program with multiple modes to cycle through.

JavaScript
parallel document

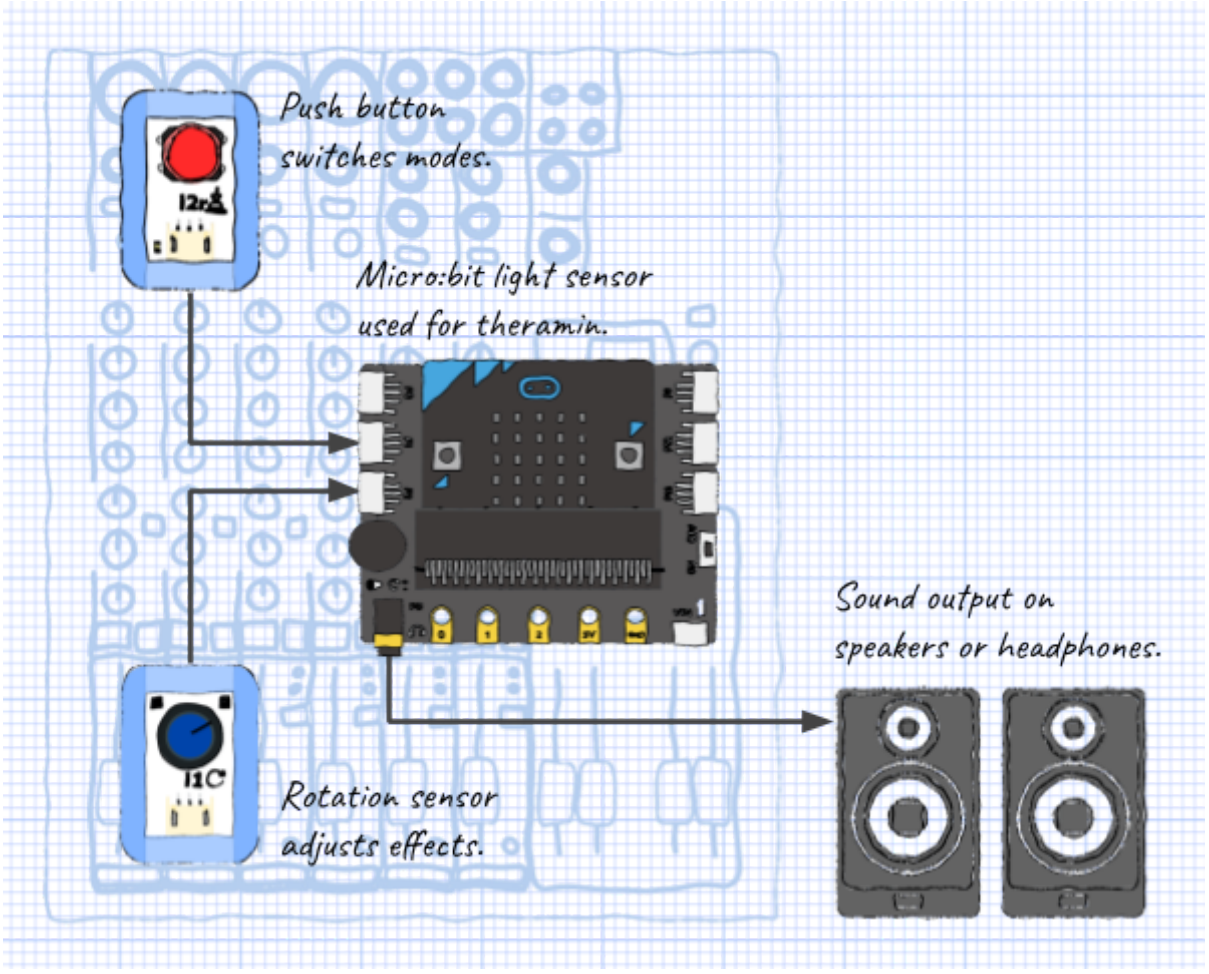
Python
parallel document



Design overview

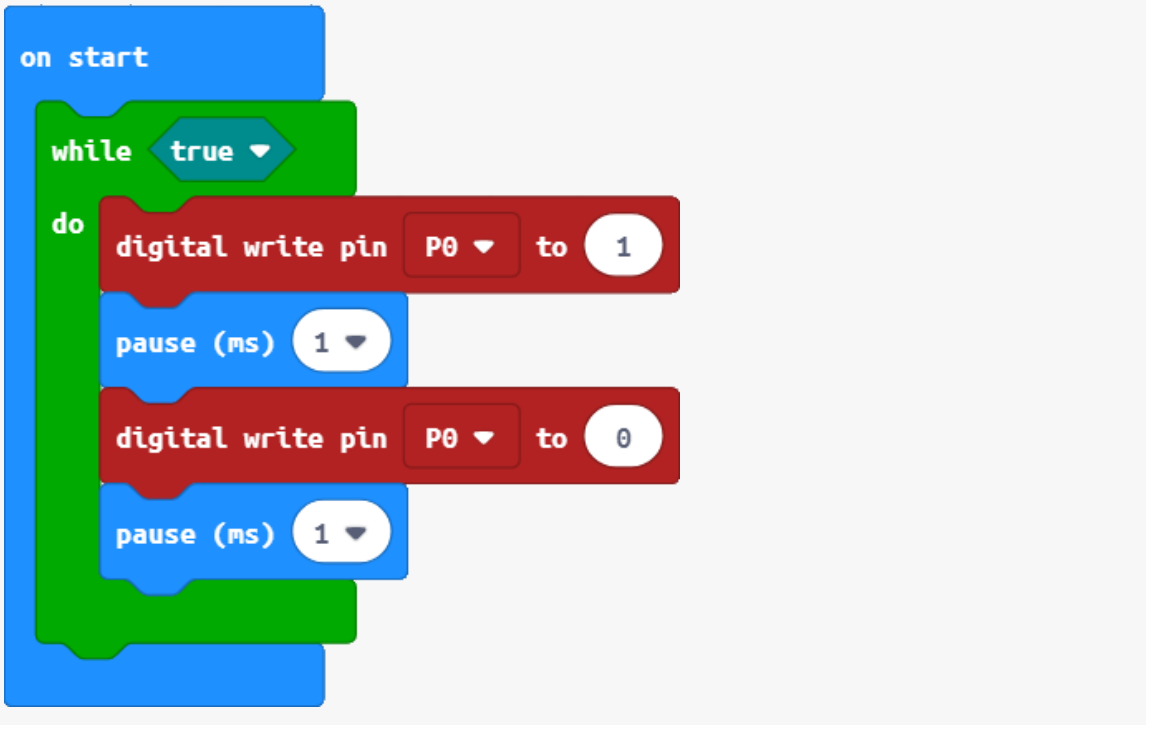
We will build a sound effects board with:

- a push button,
- a rotation sensor,
- speakers or headphones.



Generating tones

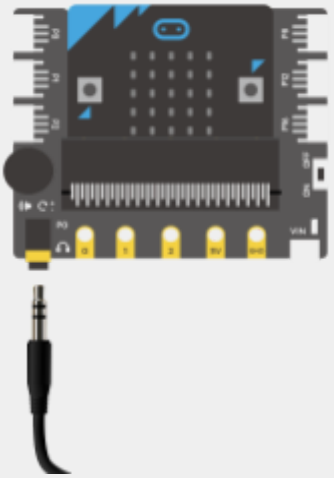
1. We can make the membrane inside a speaker vibrate to generate a sound. Set **P0** on to stretch the membrane and off to relax the membrane.



Audio on P0

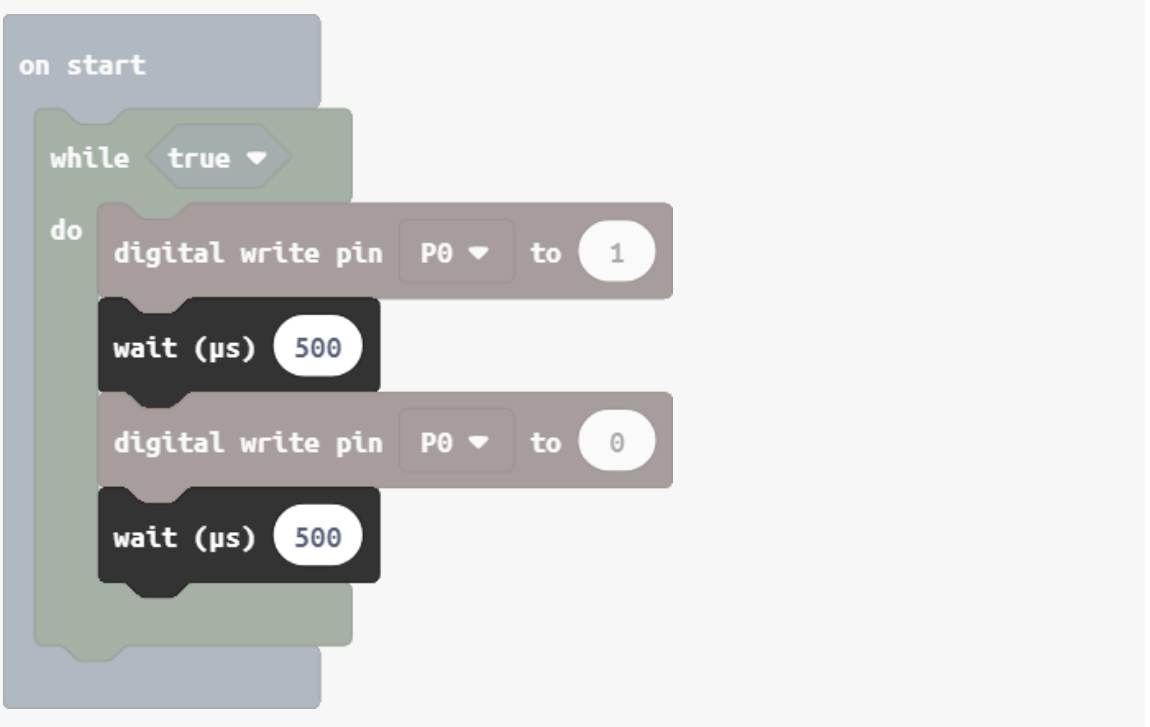
The Boson Expansion Board has an audio port for headphones or speakers.

It always uses **P0**, so other sensors will need to use **P1** or **P2**.



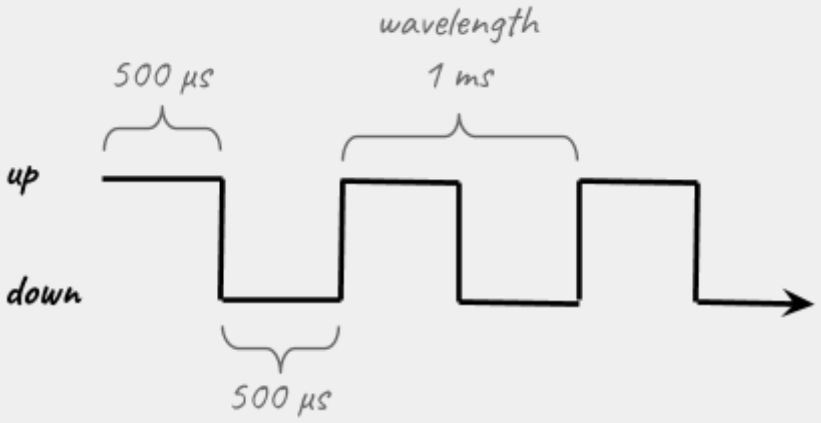
2. A shorter delay of 500 *micro*seconds makes the membrane vibrate faster.

   *(Find the **wait (µs)** command in the **Advanced** → **Control** menu)*



Wavelength

We've just generated a square waveform that is **up** for 500 microseconds and **down** for 500 microseconds.



This gives a full wavelength of **1 ms**.

Playing with wavelength

TINKER

A. Play with the wait time and observe what happens to the sound.

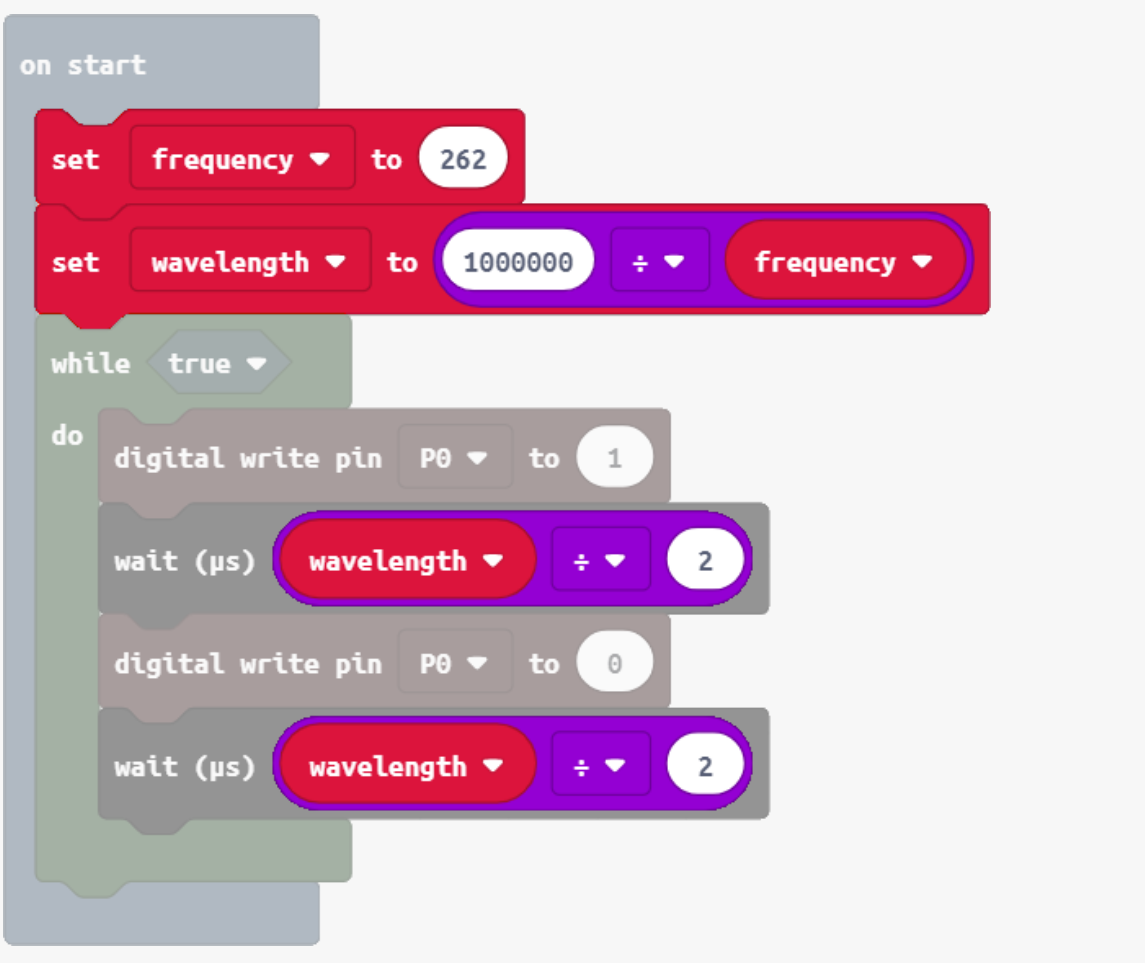
B. Replace the **while** loop with a **for** loop. Make the wait time change from 0 µs up to 2000 µs.

Using frequency

3. We measure frequency in waves per second. eg. Middle C is 262 Hz (waves per second).

   wavelength in µs = 1 000 000 ÷ frequency

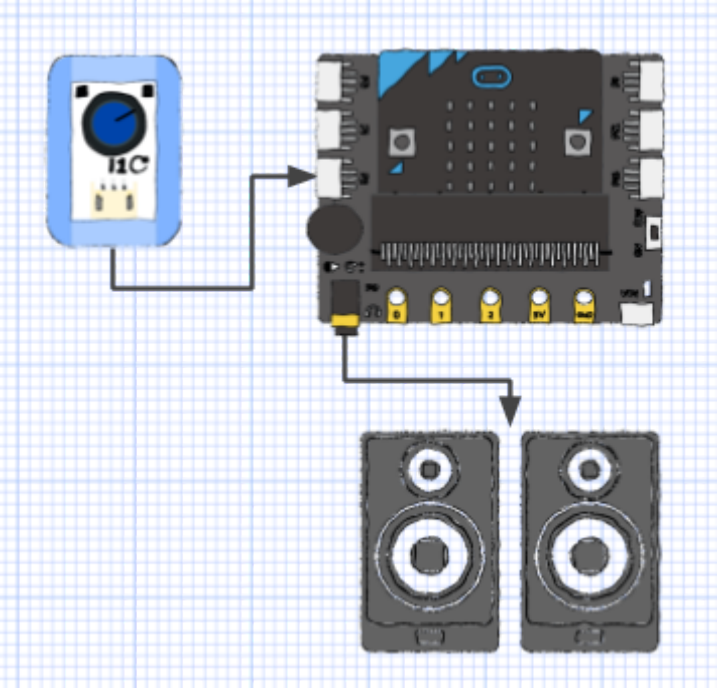
   Add the equation to calculate wavelength for a given frequency, and test it with Middle C.



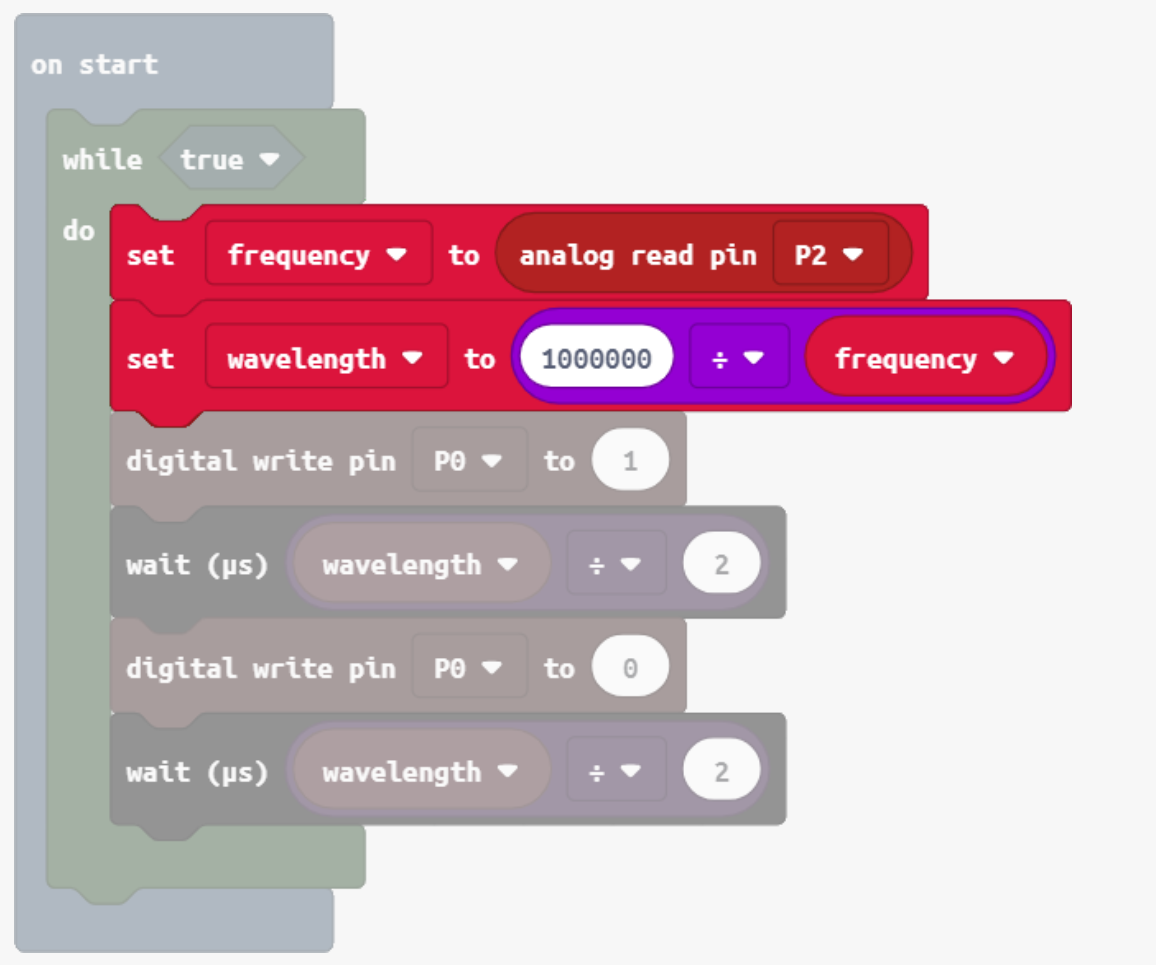
Humans are particularly sensitive to frequencies between 2000 and 4000 Hz.

These are the frequencies babies make when crying.

4. Now for some fun! Attach the Rotation Sensor to **P2** on the Boson Expansion Board.



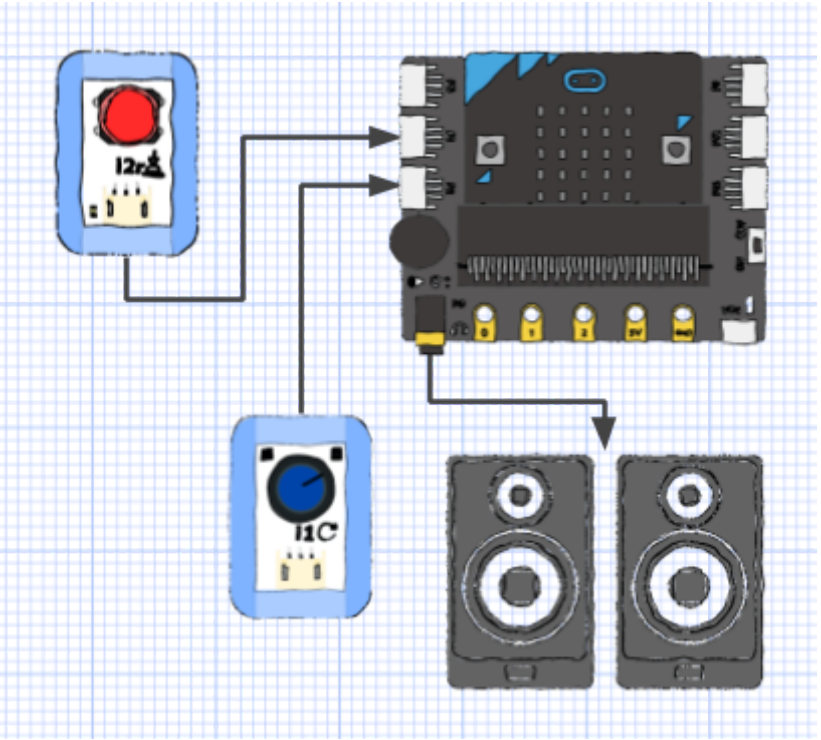
5. A couple of simple changes and we have an adjustable tone!



Multi-mode

6. Let's construct our multi-mode sound effects board.

   Attach the push button to **P1**.



Our setup will be:

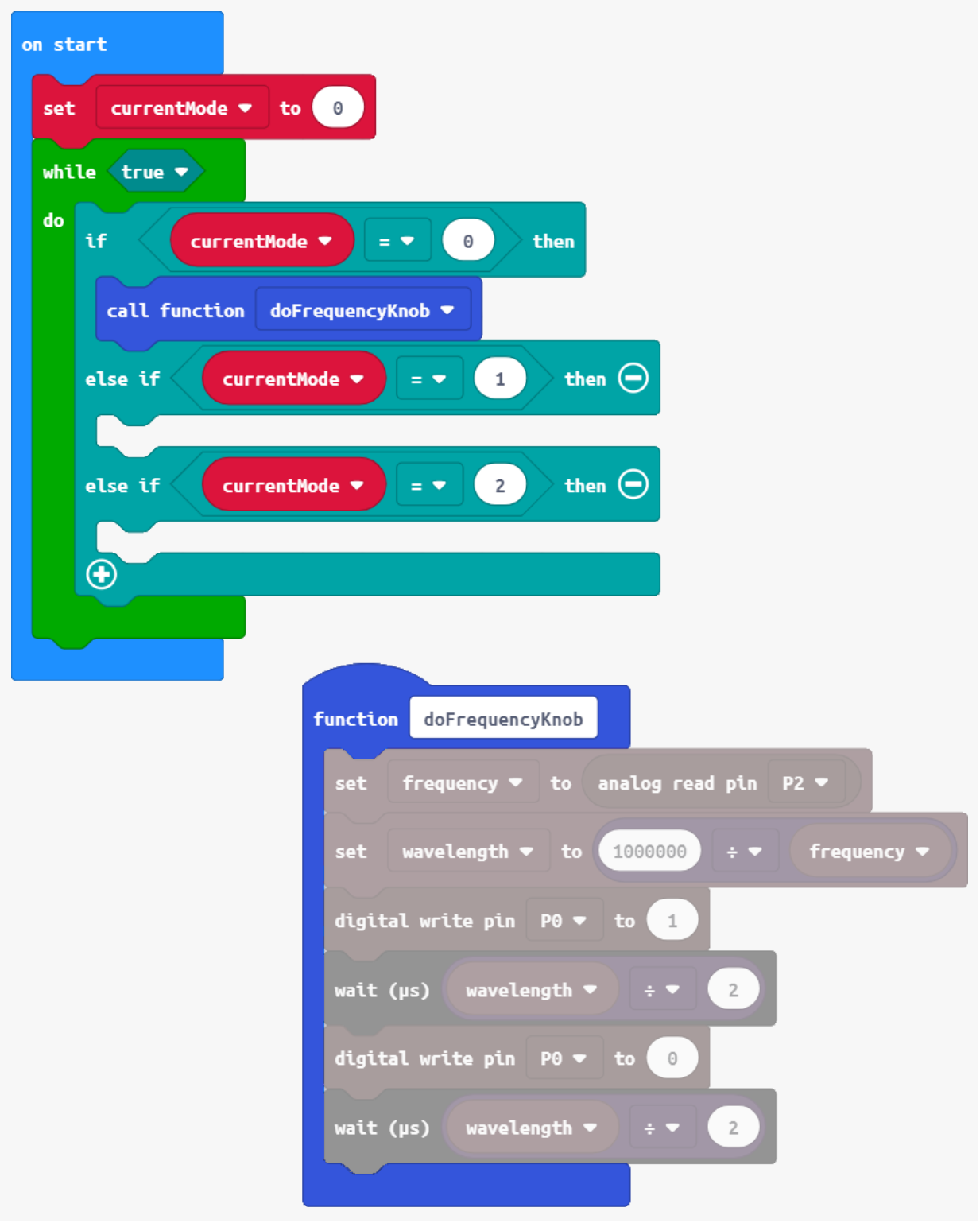
mode 0 → *Frequency knob* (already done)

mode 1 → *Light theramin* (use micro:bit light sensor instead of knob)

mode 2 → *Pew-pew sounds* (generate with a loop)

mode 3 onwards → *Add your own!*

7. We'll use a number variable **currentMode** to keep track of which mode we are in.

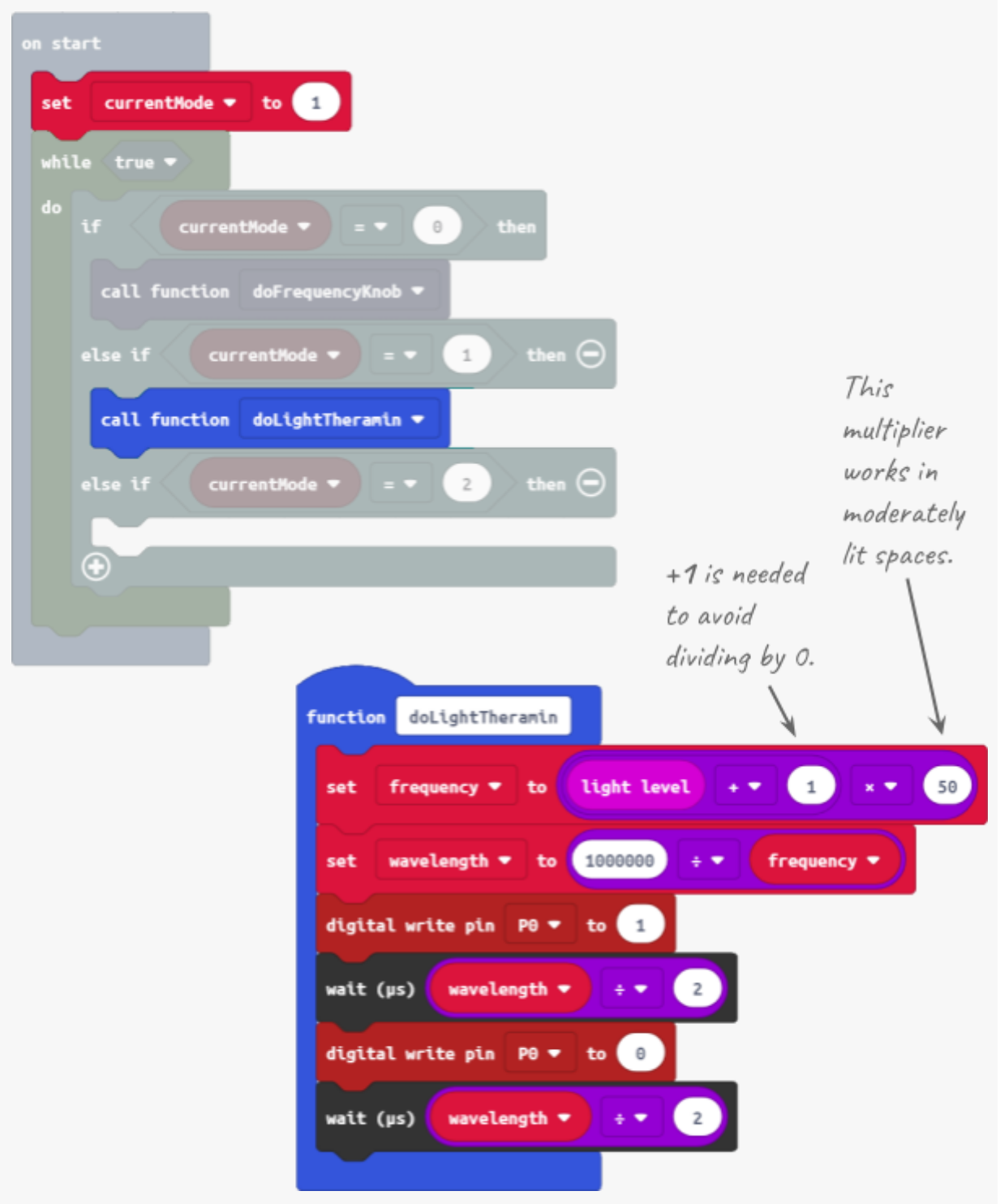


8. A light theramin makes a spooky sound when you wave your hand near the micro:bit.

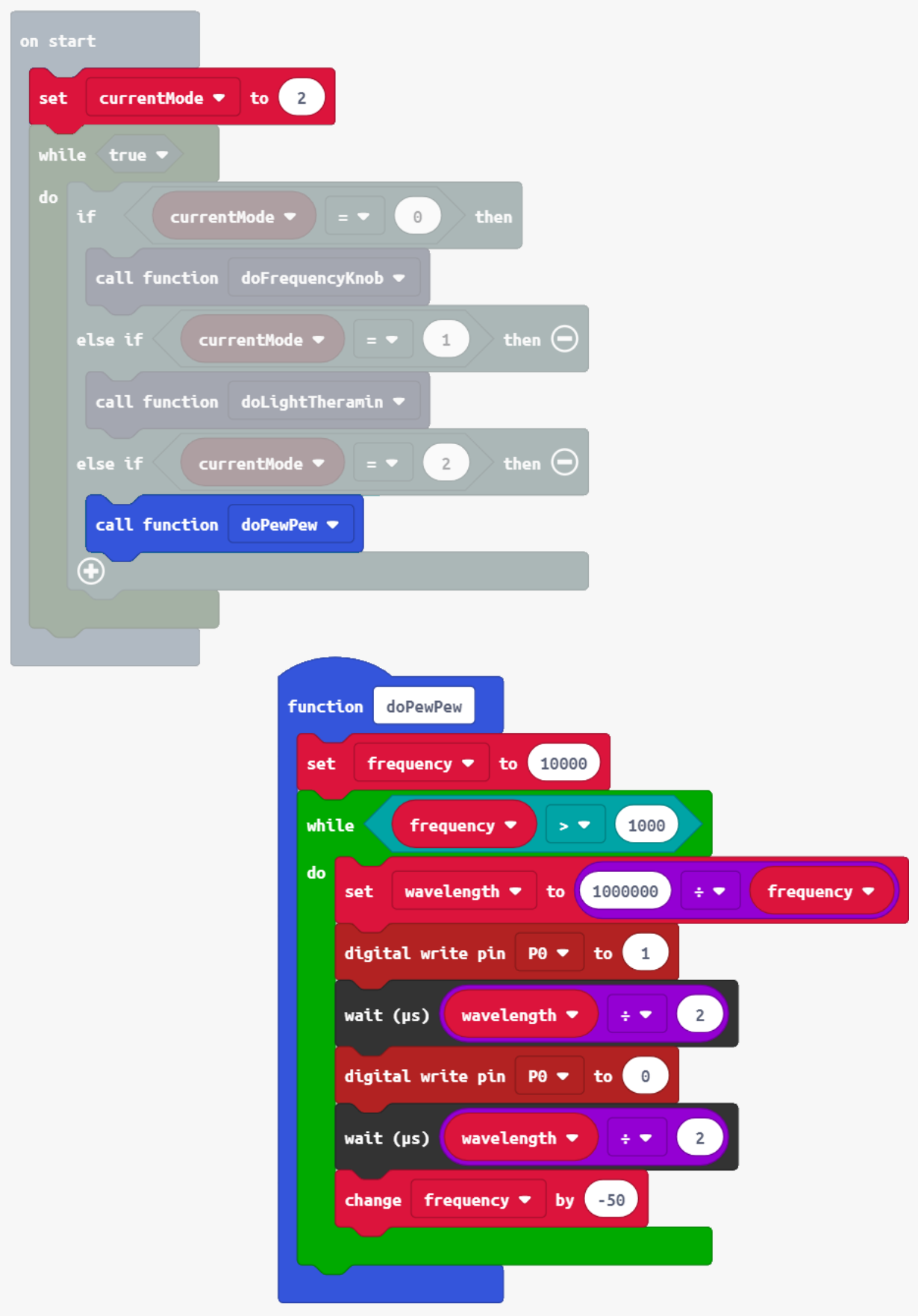
The micro:bit's own LEDs serve as its light sensors.

Full daylight can give a value up to 255, but this quickly drops and can stay below 20 in a dark-to-moderately lit room.

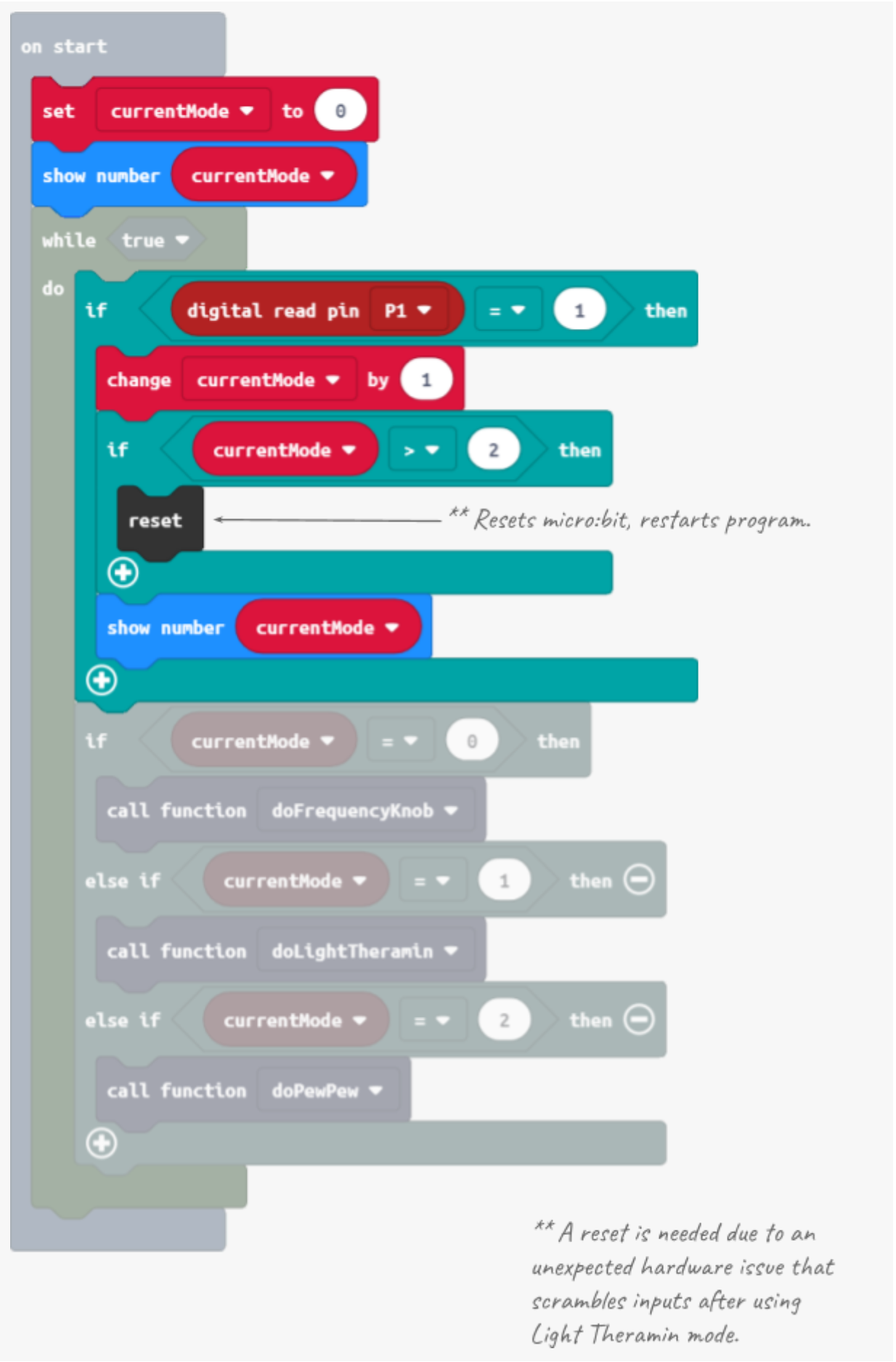
Don't delete your first function, ***doFrequencyKnob****! Make a new one* ***doLightTheramin****.*



9. The third mode is a “pew-pew” sound, using a loop to quickly slide down the frequency.
   Don’t delete your first two functions! Make a new one ***doPewPew****.*



10. Finally, add code for the push button (on **P1**) to cycle through the modes.



Maybe use headphones?

C. Reverse the PewPew sounds to go up instead of down.

D. In Frequency Knob mode, allow the user to press Button A to get a once-off display of the current frequency.

E. Add a fourth mode. Use the Music command **ring tone** to play a random frequency between 1000 and 20000 Hz.

F. Use the RGB LED Strip to show the current mode.

Old-school sounds

The earliest synthesisers and gaming consoles used these techniques to make sound effects.

- **Music speed up** - Set the micro:bit to play a tune, then use the rotation sensor to adjust the tempo.
- **Superboard** - Combine two micro:bits with radio to make a super sound effects board with twice the buttons and knobs.
- **Making waves** - Research how synthesisers create sounds (you might want to start here), and try adding waveforms to make new sounds.
- **Game sound effects** - Use the techniques learned to make sound effects for a micro:bit game.

ACTIVITY 2.5 - Swarm intruder alert system

Goals

- Build a peer-to-peer, expandable alert system.
- Use multiple sensors to detect an intruder.
- Adapt the micro:bit's light sensing to detect sudden shadows.
- Connect micro:bits with radio for a truly expandable system.

JavaScript
parallel document

Python
parallel document

BUILD

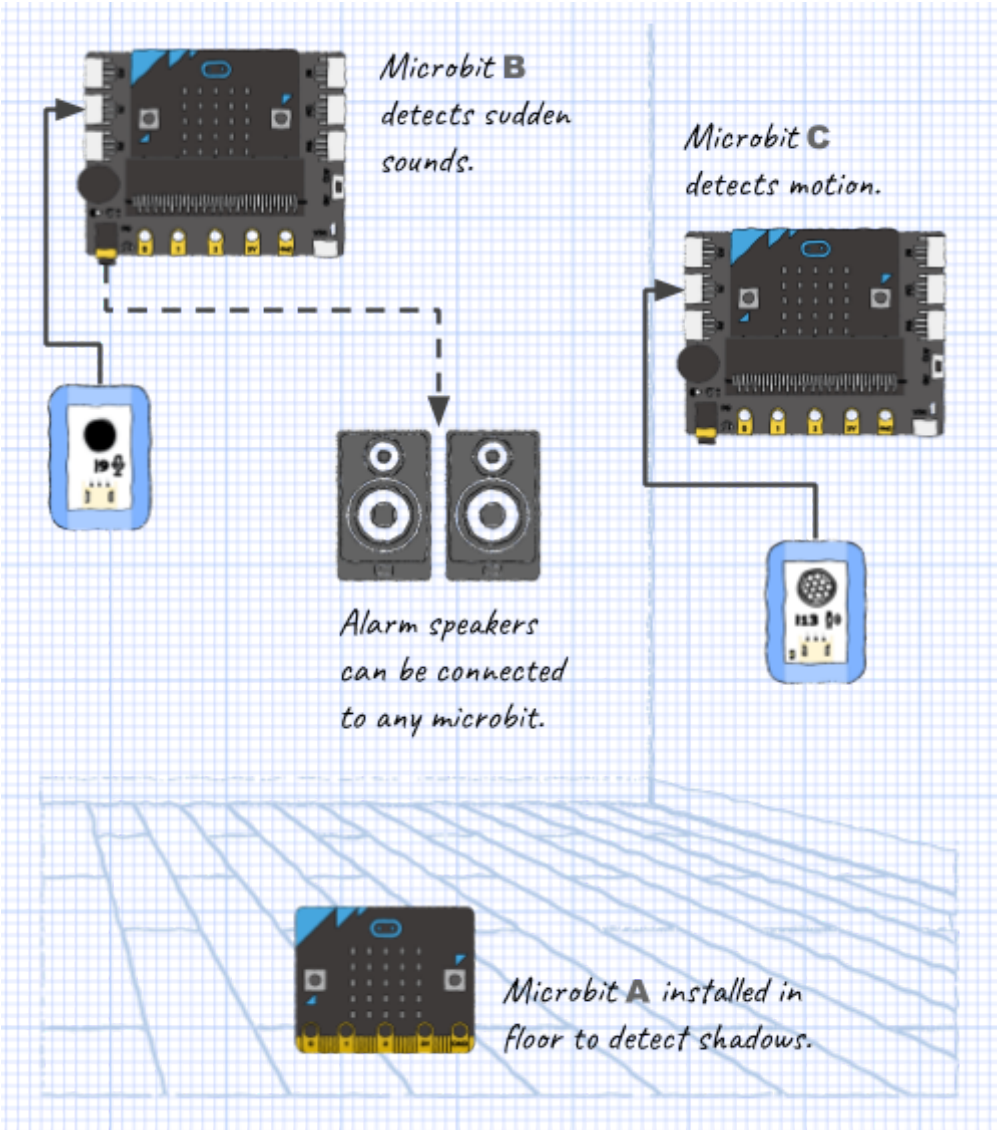
Design overview

We will build a peer-to-peer alarm system, with:

- a floor-mounted micro:bit to detect shadows on the floor,
- a wall-mounted micro:bit to detect motion,
- a ceiling-mounted micro:bit to detect sounds.

*If **any 2** micro:bits are tripped, the whole system will go into alarm.*

This makes a distributed system that is difficult to disarm.



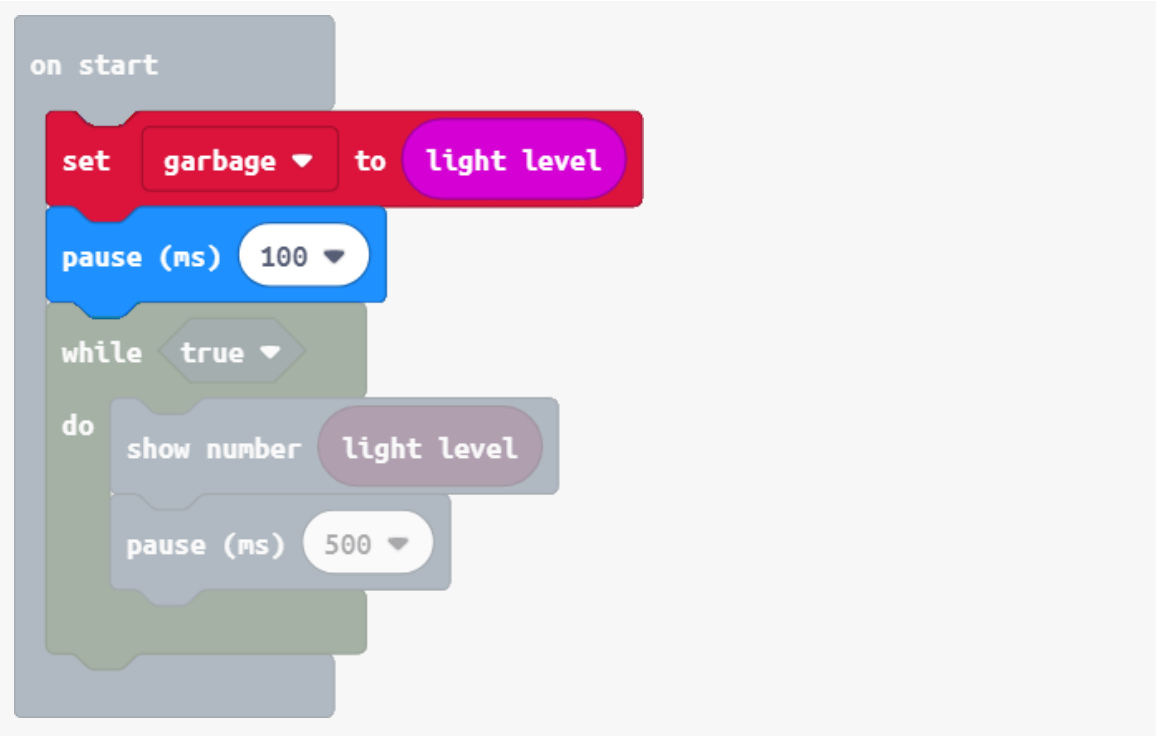
1. Start with micro:bit **A**. It doesn't need Boson Expansion Board.

   This quick program keeps displaying the micro:bit's light sensor reading.

The light sensor range is counter-intuitive. Full daylight can give a value up to **255**, but this quickly drops and can stay below **20** in a dark-to-moderately lit room.

2. Did you notice a bad reading? The very first reading is always 255! This is a quirk when we first start up the sensor.

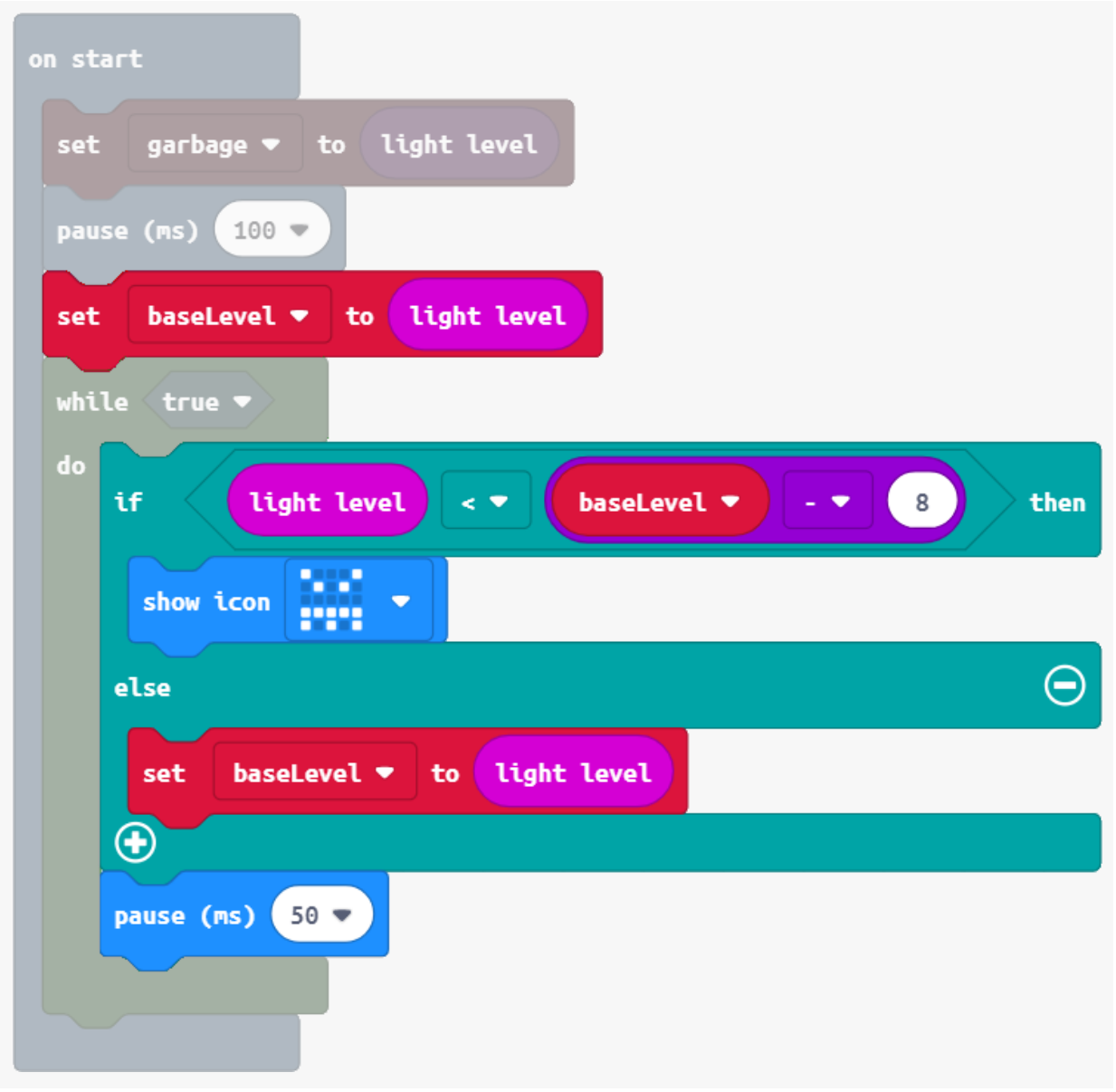
   This is easily solved. Just dump the first reading in a **garbage** variable and wait a moment before continuing.



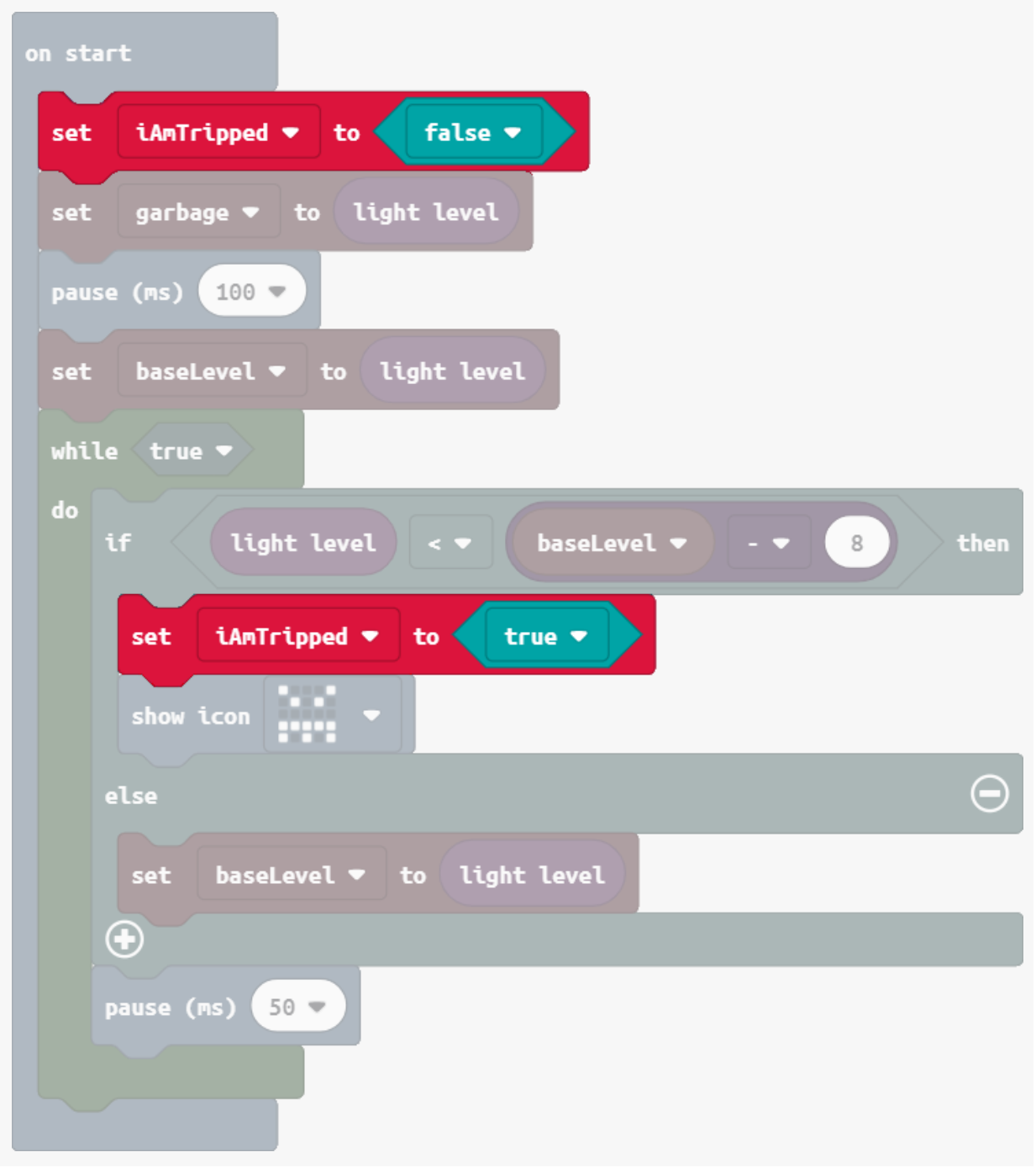
3. Over the day, light levels will naturally change in a room. To detect a sudden light change, we'll need a moving base light level.

   Here's our plan in pseudocode:

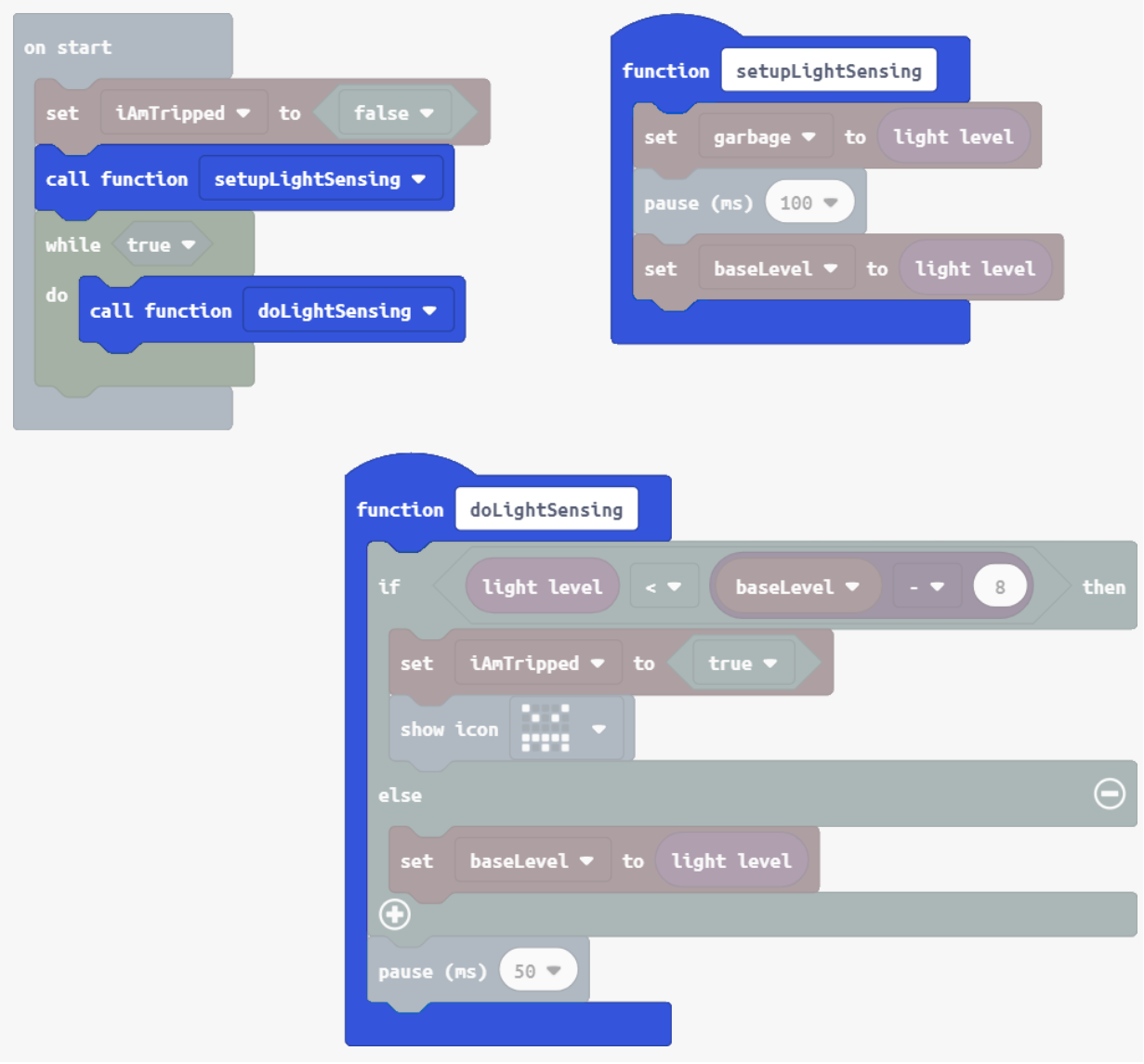
```
Start sensor (take garbage reading)
baseLevel ← current light level
REPEAT forever
    IF current light level is much less than baseLevel THEN
        This micro:bit is tripped!
    ELSE
        baseLevel ← current light level
    END IF
END REPEAT
```



4. Introduce a variable to indicate this micro:bit has been tripped.



5. Now, separate the sensing code into functions.

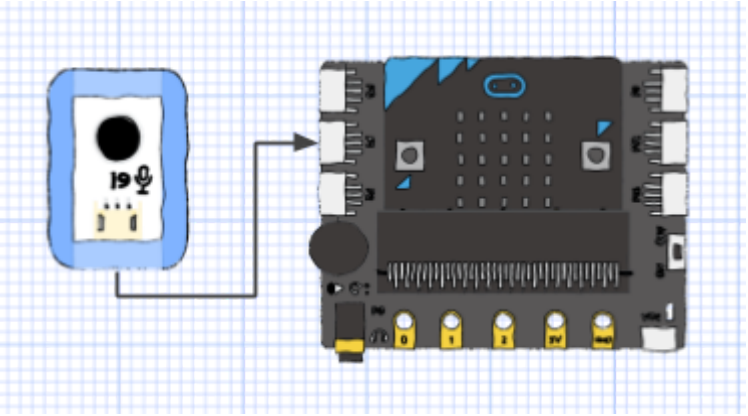


Don't lose this program!

We're moving onto the next micro:bit, but we'll come back to this program later.

6. Insert micro:bit **B** into a Boson Expansion Board.

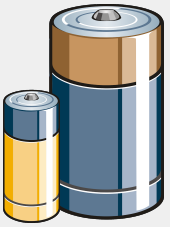
   Attach the Sound Sensor to **P1**.



Boson power

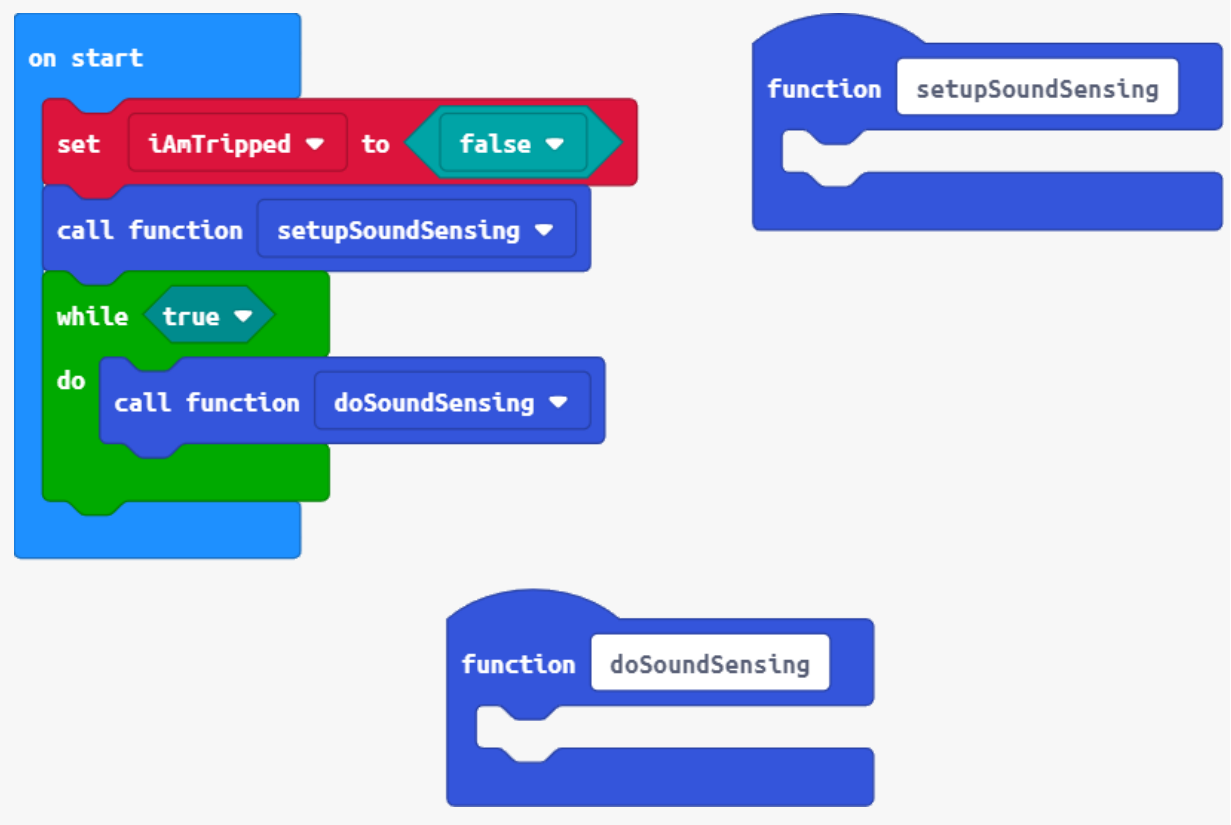
The most reliable solution is to power this micro:bit via the **VIN** USB socket on the Boson Expansion Board. See THREE QUICK TIPS.

Results may vary if the micro:bit's power is used.



7. Our second micro:bit will sense a sudden change in sound level.

   Begin with the same basic structure as micro:bit **A**.



8. As with the light sensor, we'll use a moving base level.

baseLevel ← *current sound level*

REPEAT forever

IF current sound level is much more than ***baseLevel*** *THEN*

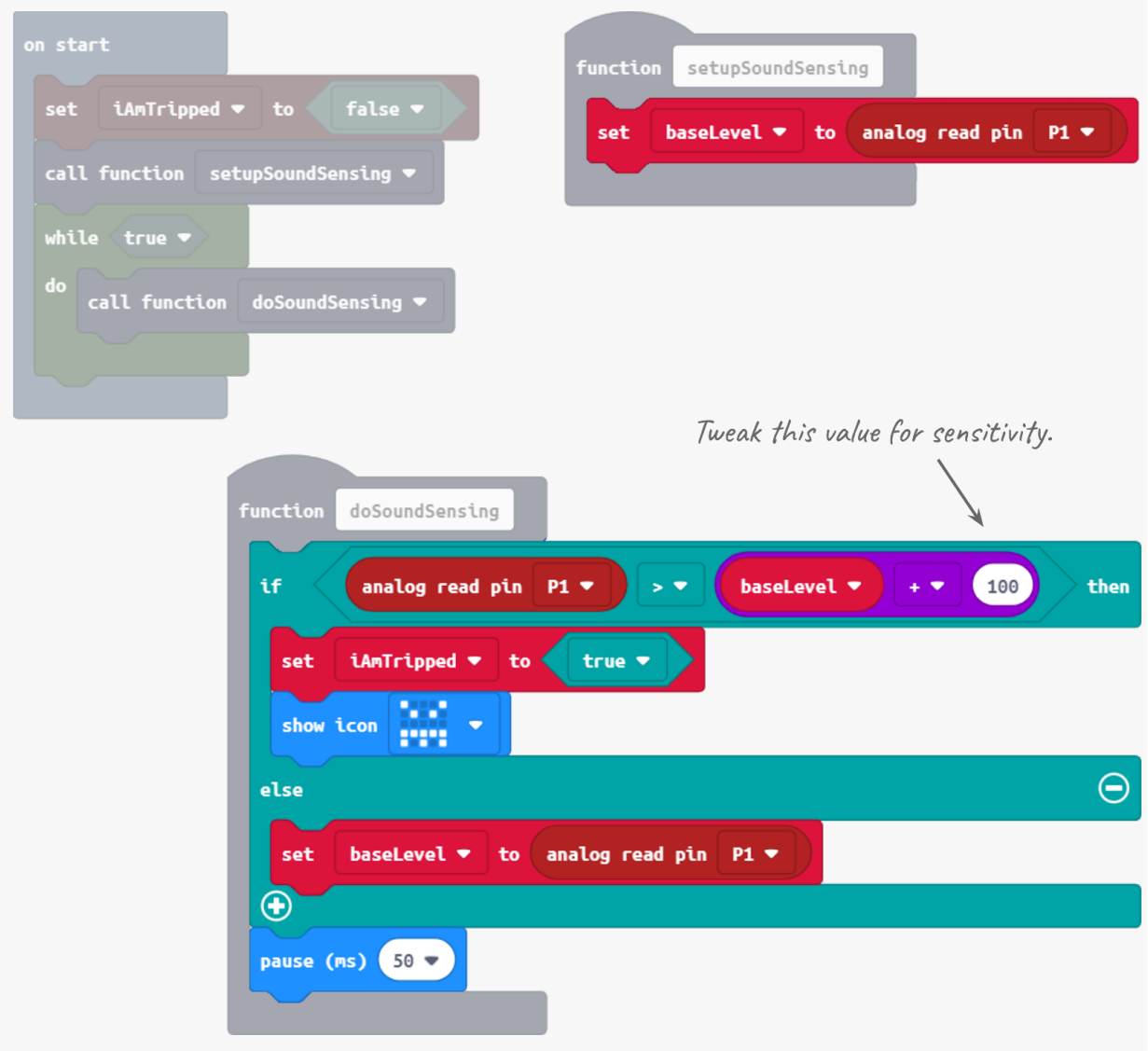
This micro:bit is tripped!

ELSE

baseLevel ← *current sound level*

END IF

END REPEAT

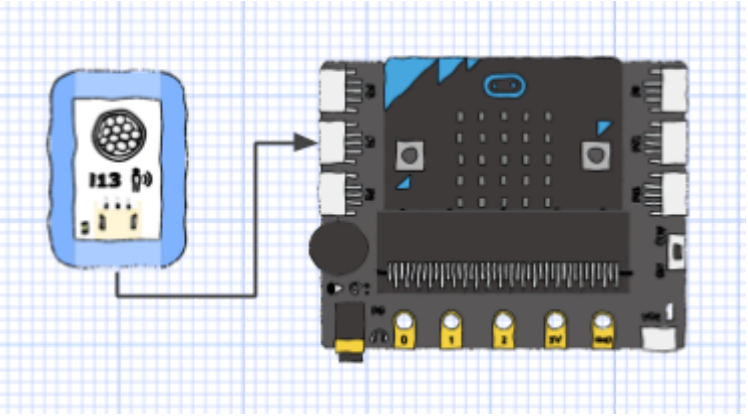


Don't lose this program!

We're moving onto the last micro:bit, but we'll come back to this program later.

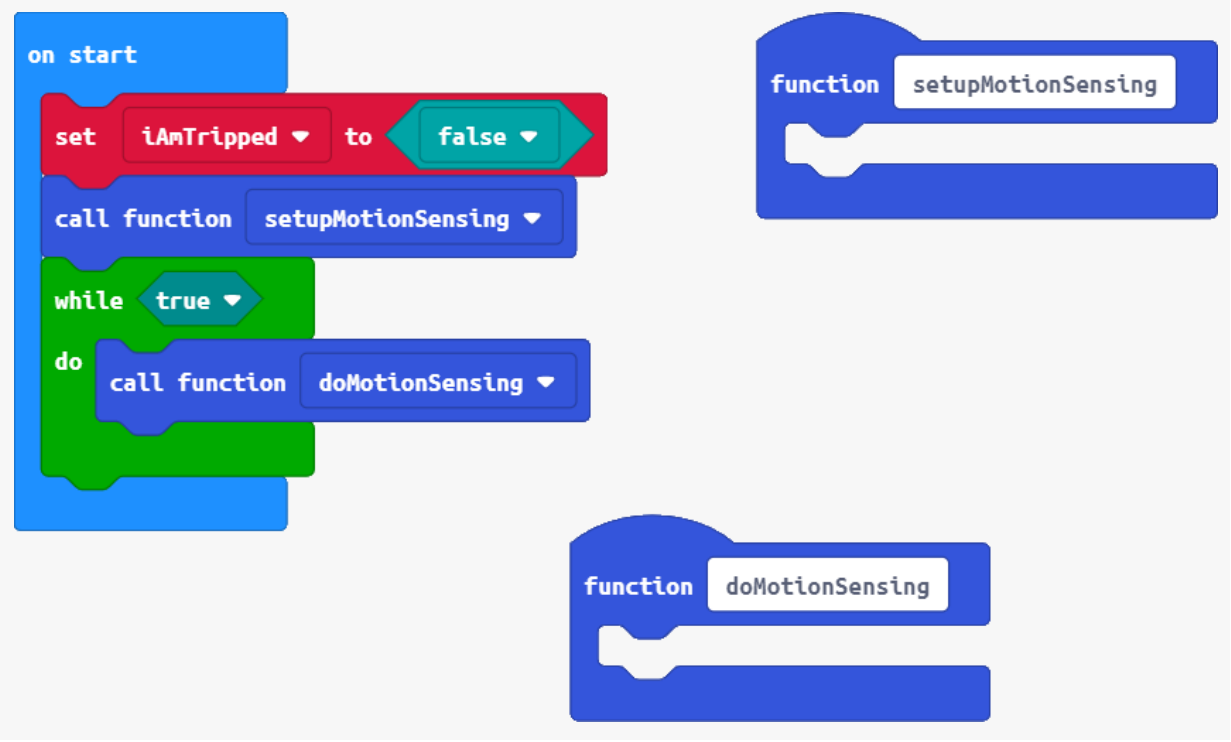
9. Insert micro:bit **C** into a Boson Expansion Board.

   Attach the Motion Sensor to **P1**.

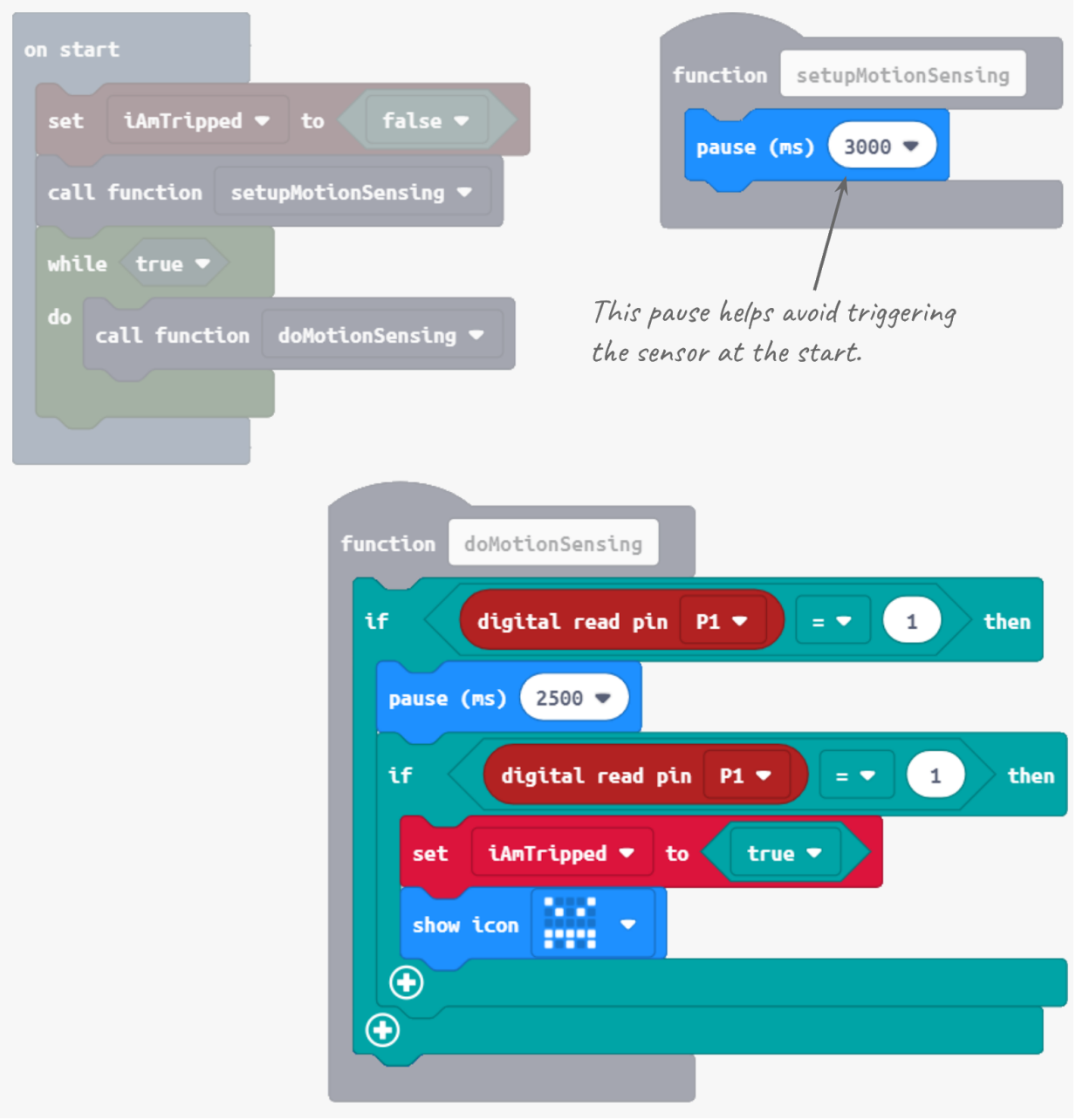


10. Our third micro:bit will be triggered by the motion sensor, just like in ACTIVITY 2.2.

    Begin with the same basic structure as the other two micro:bits.

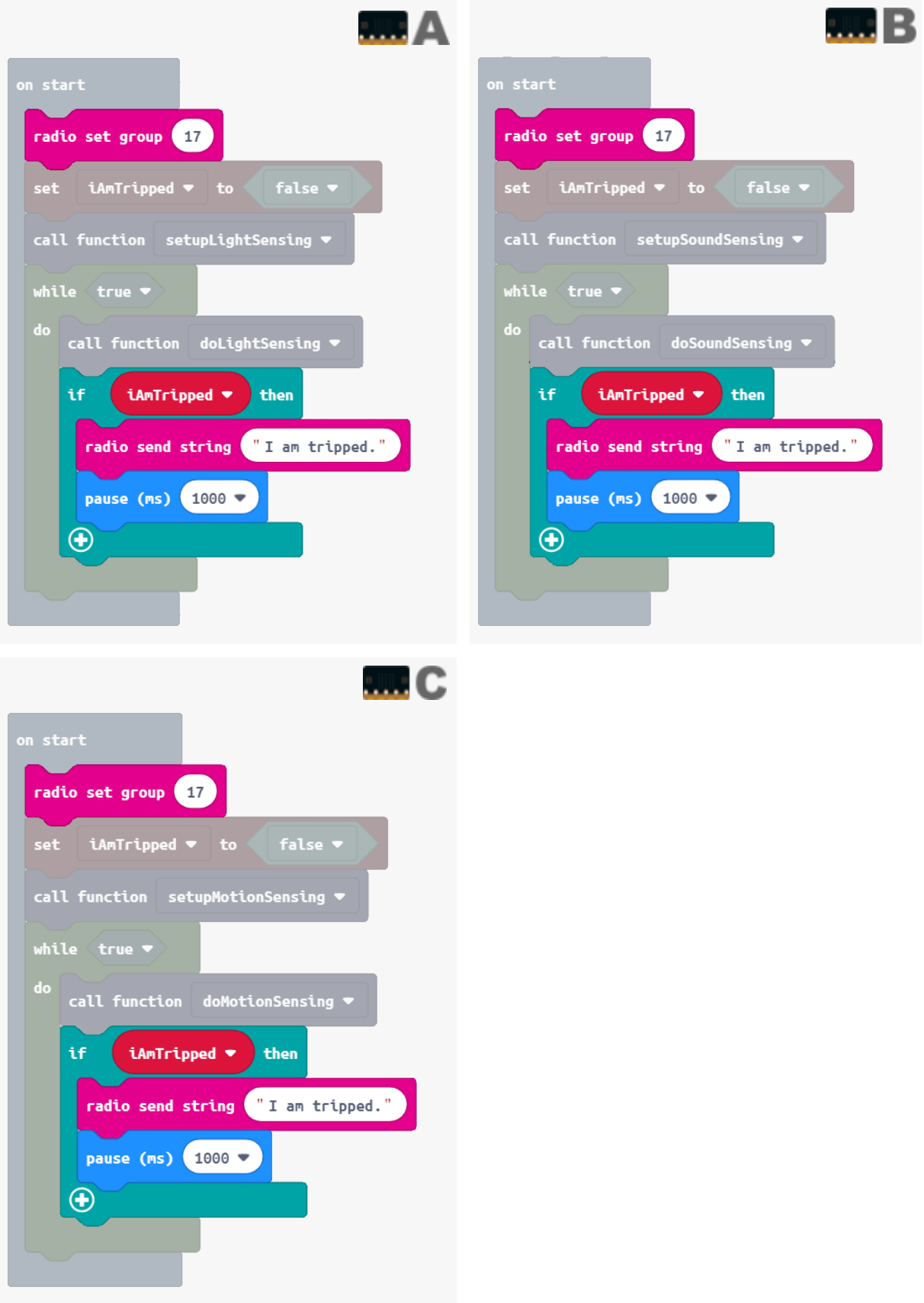


11. Motion is confirmed if the sensor is active for a full 2.5 seconds.

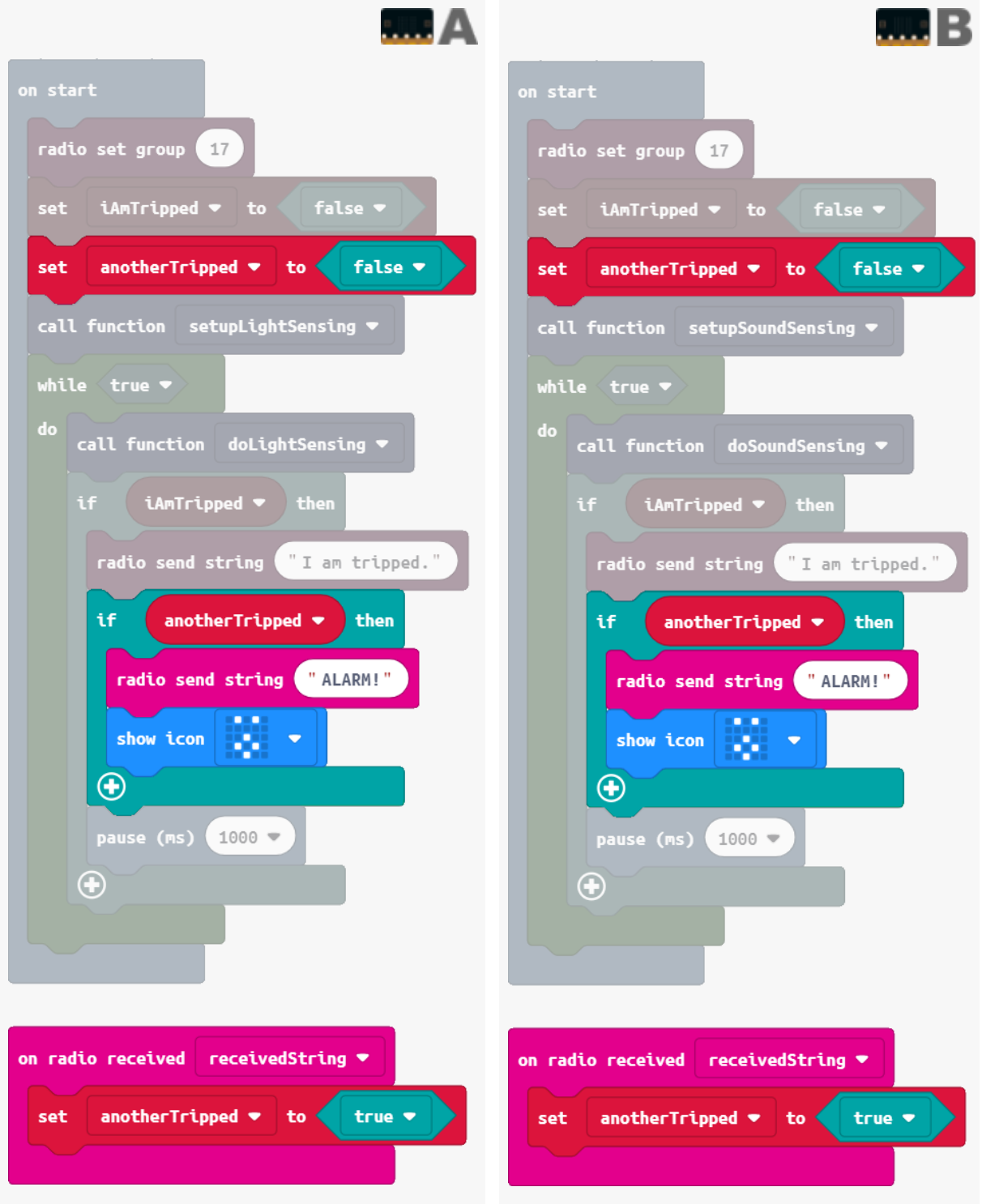


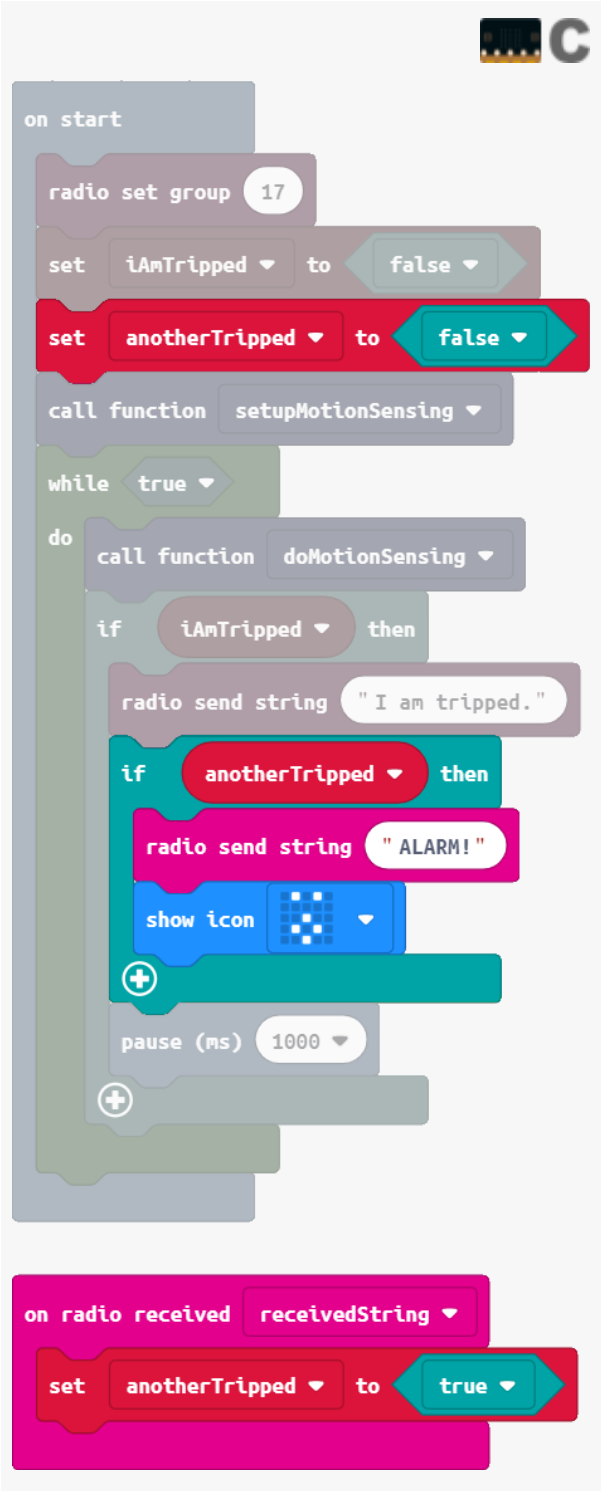
Hitting the alarm

11. If any one of the micro:bits is tripped, it should begin sending messages every second.



12. If a micro:bit is tripped *and* it gets a message from another one, it's time to hit the alarm.




on start
radio set group 17
set iAmTripped to false
set anotherTripped to false
call function setupMotionSensing
while true
do
call function doMotionSensing
if iAmTripped then
radio send string " I am tripped."
if anotherTripped then
radio send string " ALARM!"
show icon
pause (ms) 1000
on radio received receivedString
set anotherTripped to true

Alarm! Alarm!

A. Add code so the micro:bits play a melody once the alarm is going.
(Speakers can be attached to just one of them.)

B. Increase the initial time delay before micro:bit **C** (motion sensing) becomes active. This should be at least 10 seconds to allow people to leave the premises.

C. Intruders sometimes have torches. For micro:bit **A** (light sensing), add code so it also trips if the light level rises suddenly.

D. An untripped micro:bit may not go into alarm mode, even if the other two do. Fix this in the common code by making sure the ALARM! message is properly received and acted on.

Over to you

We've now explored all the modules in the Boson Starter Kit. What else can you make?

- More improvements to our intruder alert system:
    - **Hidden reset** - The reset button is out of sight. Add code so that a press of button B on *any one* micro:bit forces a reset on *all three*.
    - **Reset code** - Improve your button B mass reset with a trick or password (eg. you must hold the button for 3 seconds, or you must follow it with a combination of A and B.)
    - **Accelerometer alarm** - This is a fourth micro:bit design. Attach it to a door or some object in the house that is likely to move in case of an intrusion.
- **???** -
- **Testing heat sensor** - The motion sensor should only respond to changes in infrared (heat). Build a tester that uses the mini servo to wave a piece of cardboard or LEGO in front of the motion sensor.

MODULE 3

Real-world projects with BBC micro:bit and **BOSON Science Kit**.

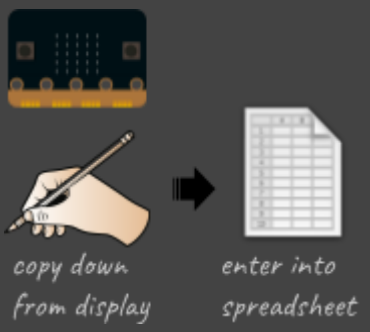
THREE WAYS to get data off the micro:bit

You recorded temperature readings once per hour over 24 hours, and stored them in an array. How can you get this data off the micro:bit for analysis in a spreadsheet?

1 Code a loop to **scroll** each array value on the micro:bit's display when Button A is pressed. Button B moves to the next value.

Write down each value and/or enter the values manually into a spreadsheet.

Is your micro:bit fixed somewhere? Use **radio** in your code to send the values to another micro:bit first.



2 Use **Serial** commands in your code to send text to a computer via USB.

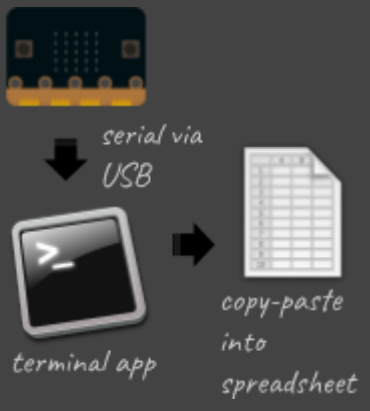
The text appears in a **terminal** app on the computer, and can be copied-pasted from there.

Coding in Python only? Offline environment Mu includes a terminal *(Windows, Mac or Linux)*.

Coding in Visual or JavaScript? Try the MakeCode for micro:bit app *(Windows 10 required)*.

Just want a terminal alone?

- Mac and Linux follow specific instructions.
- Windows install drivers and terminal app.



3 Use a **mobile app** like Bitty Data Logger to receive data on your iOS or Android device.

Data is sent to the device over Bluetooth, and can be shared as a spreadsheet from the app.

Note:

- Bitty gathers data *directly* from micro:bit's internal sensors (accelerometer, temperature, magnetometer). It does not give access to data stored in variables / arrays.
- App is not free.

ACTIVITY 3.1 - Smart robotic fan

Goals

- Create an “old fashioned” robotic swinging fan that goes based on humidity and temperature.
- Test the Boson Humidity Sensor and Mini Servo Module.
- Use the OLED module to help test sensor values.
- Use a **Maths formula** to control behaviour.

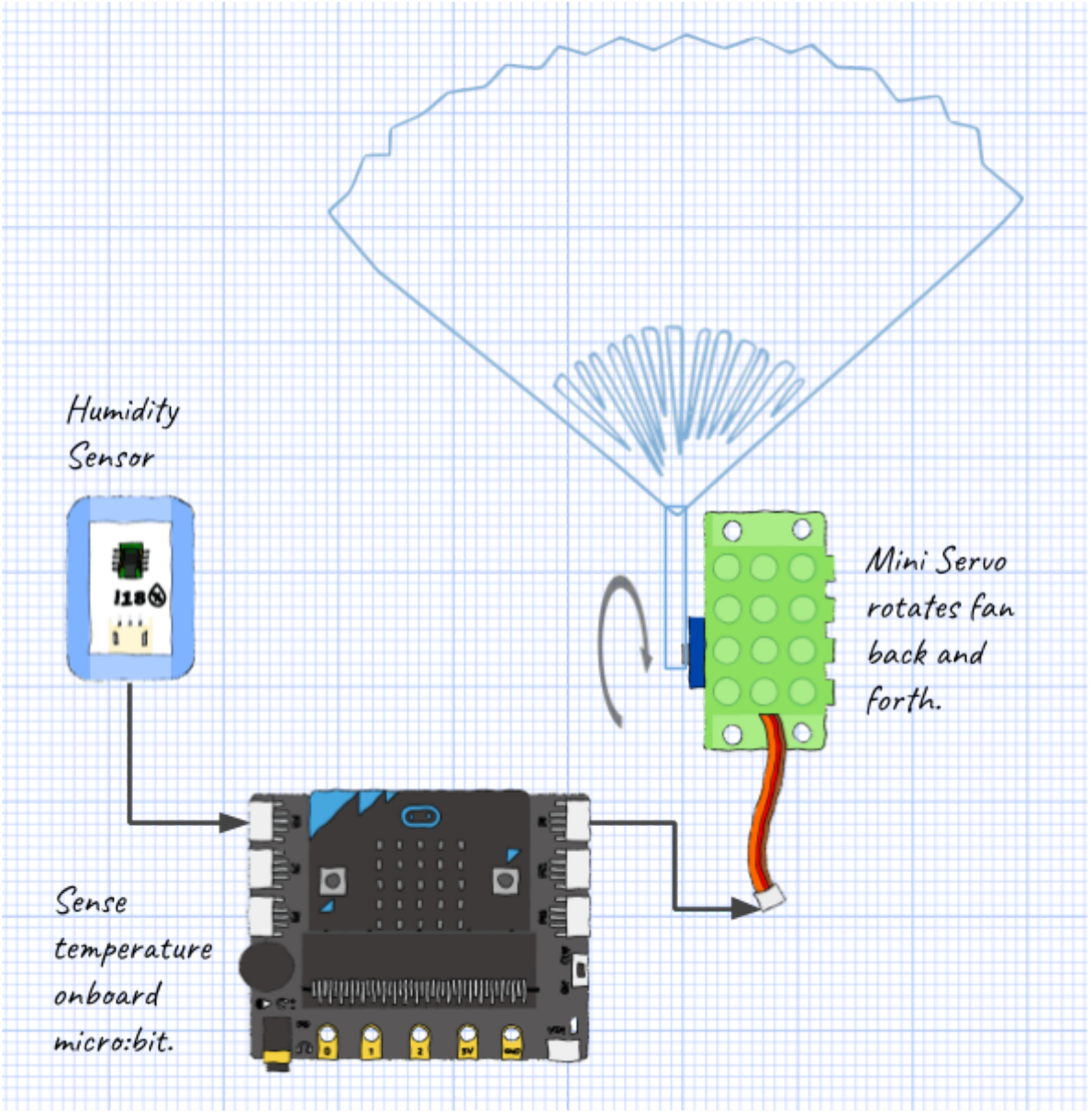
JavaScript
parallel document

Python
parallel document

Design overview

BUILD

We will build a fan that swings back and forth if the temperature *and* humidity are too high.



To measure both temperature *and* humidity together, we'll use a value called the **discomfort level**. The fan will move when the **discomfort level** gets too high.

$$\text{discomfort level} = \text{humidity in \%} + (\text{temperature in °C} \times 20)$$

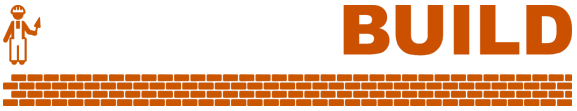
How uncomfortable is it?

Our formula is simplified. Researchers actually use many factors to determine "comfort" including:

- temperature,
- humidity,
- air speed,
- metabolic rate (how active is your body right now?),
- clothing.

See the CBE Thermal Comfort Tool for more information.

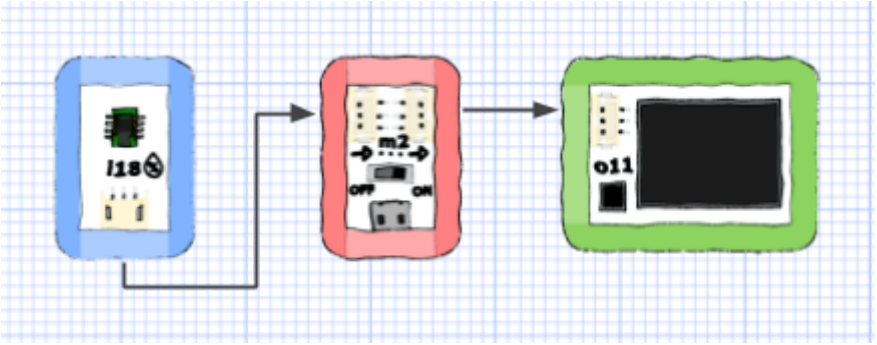
Testing the Humidity Sensor



1. First, we'll test the Humidity Sensor without the micro:bit.

   Attach the Humidity Sensor through the small Mainboard to the OLED Module as shown.

   (Remember to give the Mainboard power via USB lead.)



2. Tap the button on the OLED Module until you see **Air Humidity**. The OLED Module presents the reading as a percentage.

3. Now tap the button until you see **Analog Data**. This is the value that will go to the micro:bit. What do you notice about it?

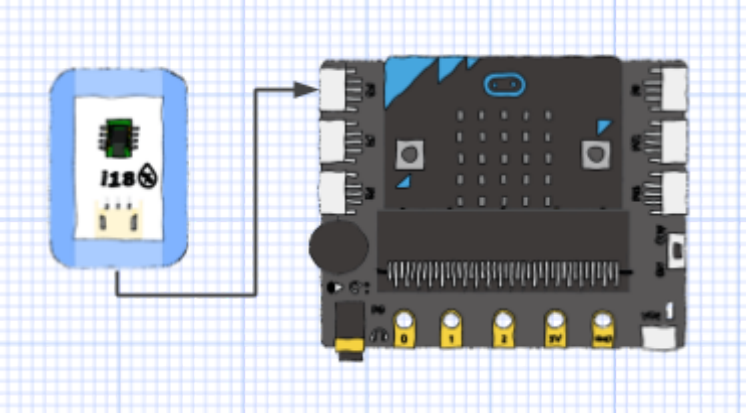
The **Boson Humidity Sensor** reads the relative humidity in the environment.

It gives a value (between 0 and 1023) which is approximately 10 times the reading in percentage form.

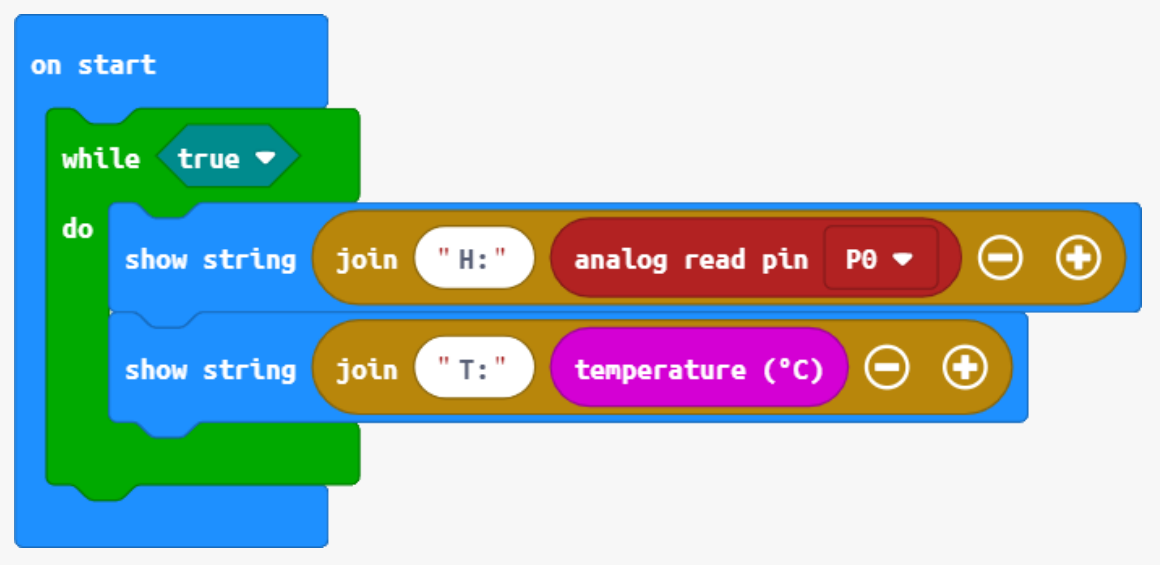
eg. **45%** humidity gives the value **450**.

Calculating discomfort

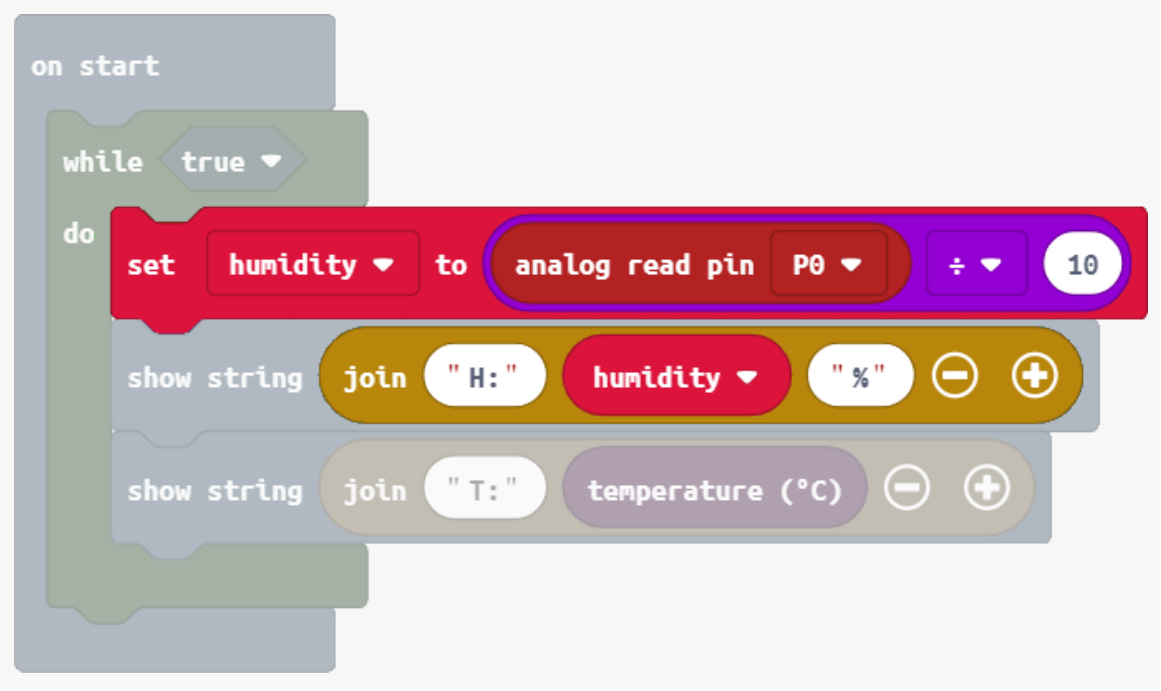
4. Now attach the Humidity Sensor to **P0** on the Boson expansion board.



5. Make a loop to keep getting the Humidity Sensor reading on **P0**, as well as the micro:bit's temperature reading, and scroll them on screen.



6. Divide the sensor value by 10 to get the actual humidity in %.

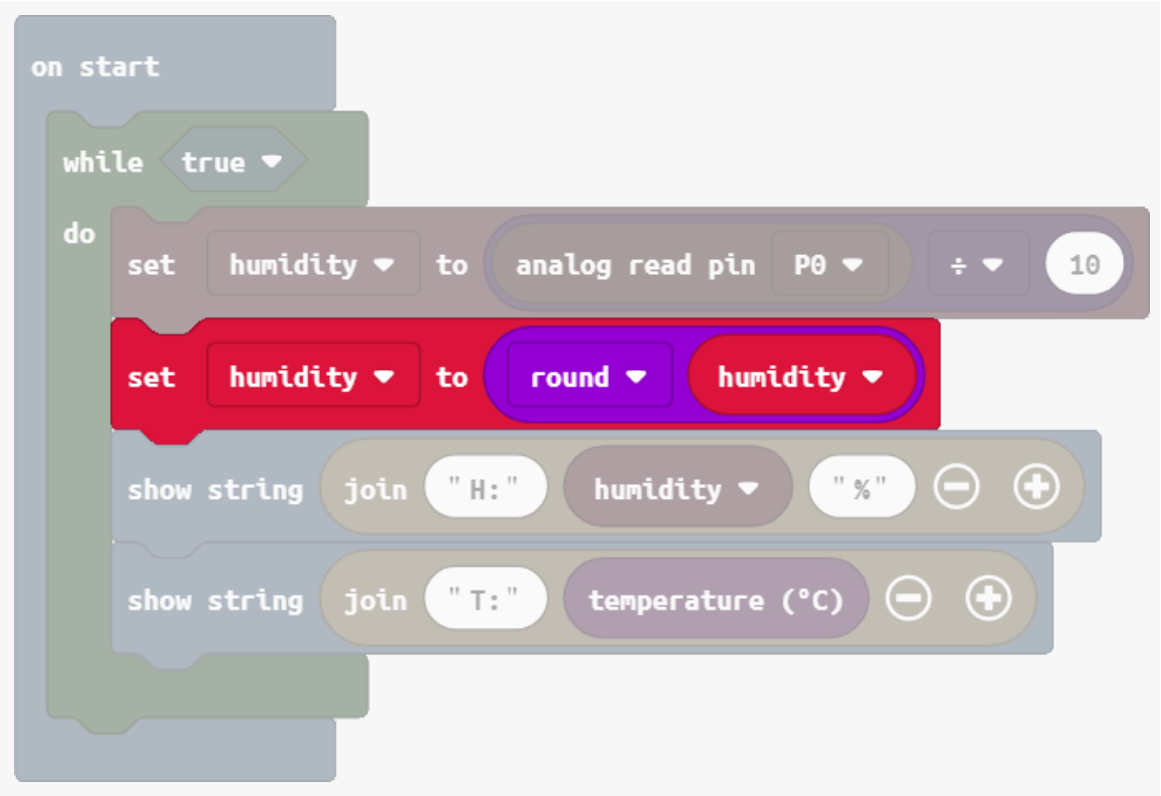


Did you get a strange value after that calculation?

eg. Instead of **44.3**, the micro:bit scrolled **44.299999999999998**?

This is a Maths quirk on the microcontroller, and it may not occur on the preview when coding. To deal with it, we'll use the Maths **round** command.

7. Round **humidity** to the nearest whole number before displaying it.

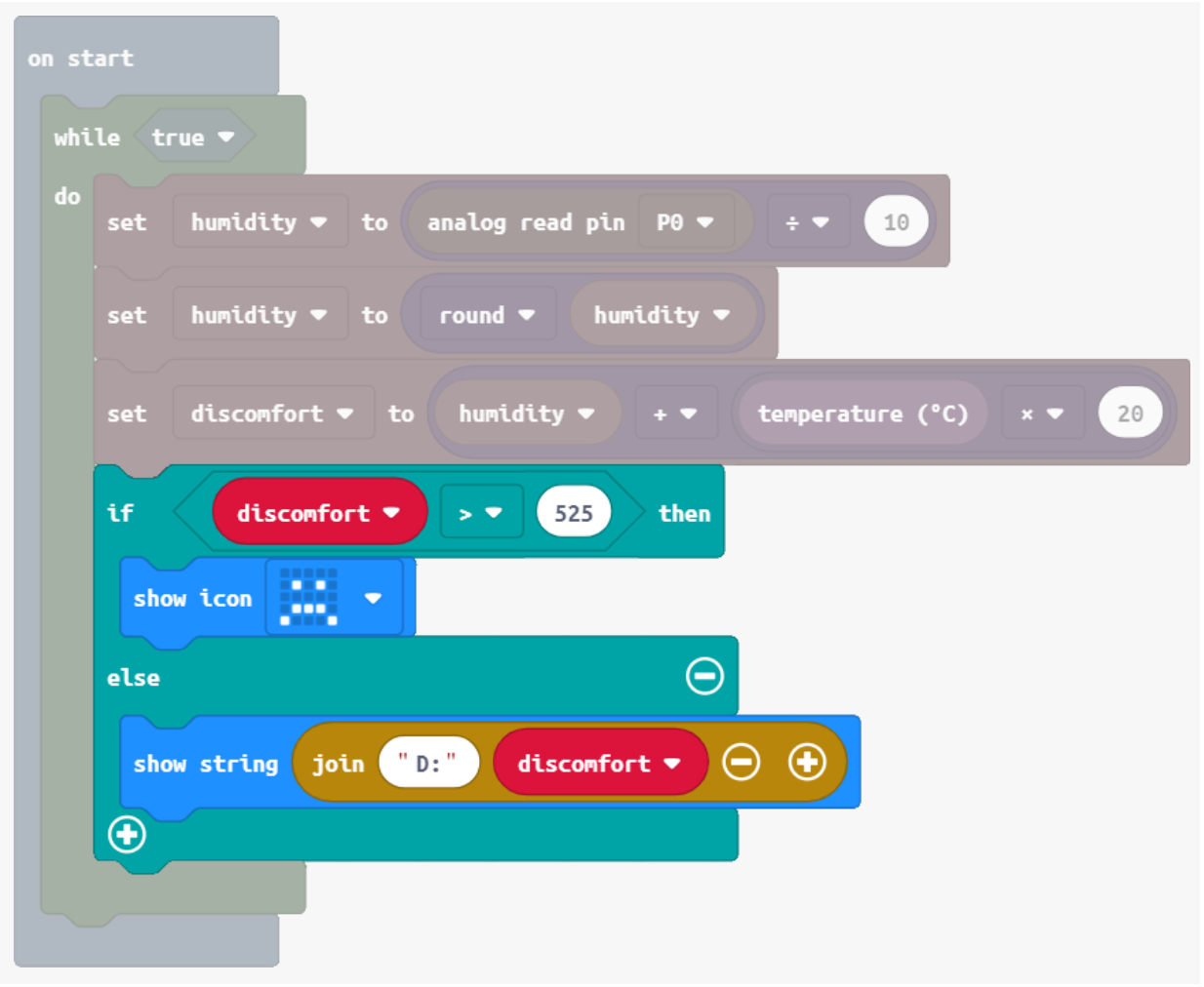


8. Now, calculate the **discomfort level**. The formula is:

$$\text{discomfort level} = \text{humidity in \%} + (\text{temperature in °C} \times 20)$$

Note down the value you get. Depending on your classroom environment, it should be between 400 and 600.

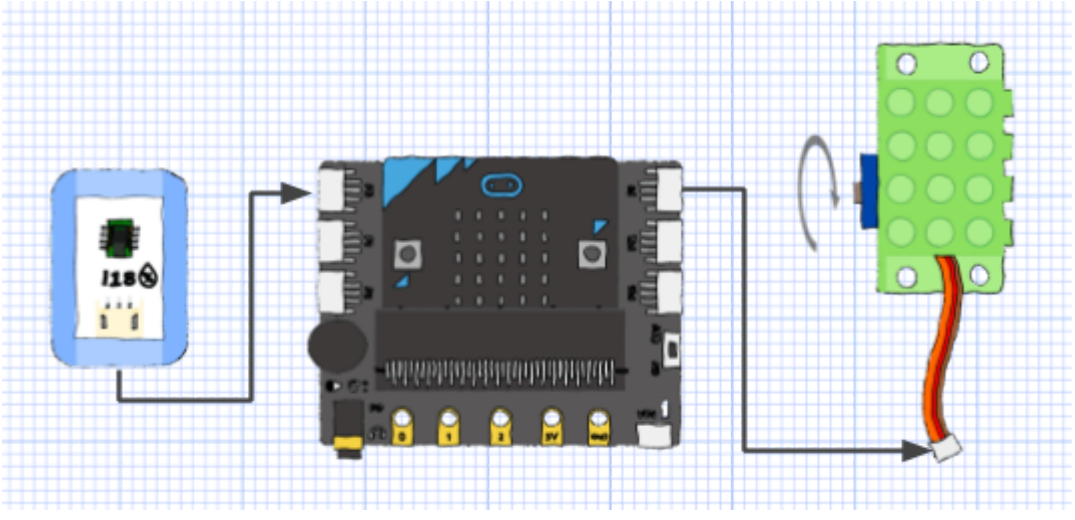
9. Breathe (“haaaa”) on the Humidity Sensor. **Discomfort level** should rise by at least 15, then fall back down slowly. Find your threshold value (eg. 525) to show an alert icon.



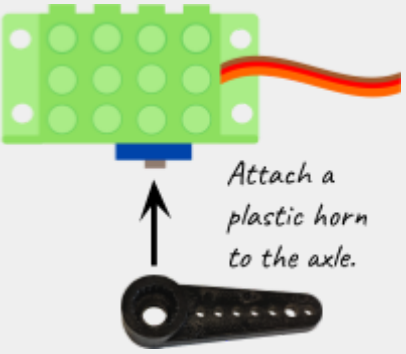
Driving the fan

BUILD

10. For the old-fashioned fan, we’re going to use the Mini Servo. Attach it to port **P8**.

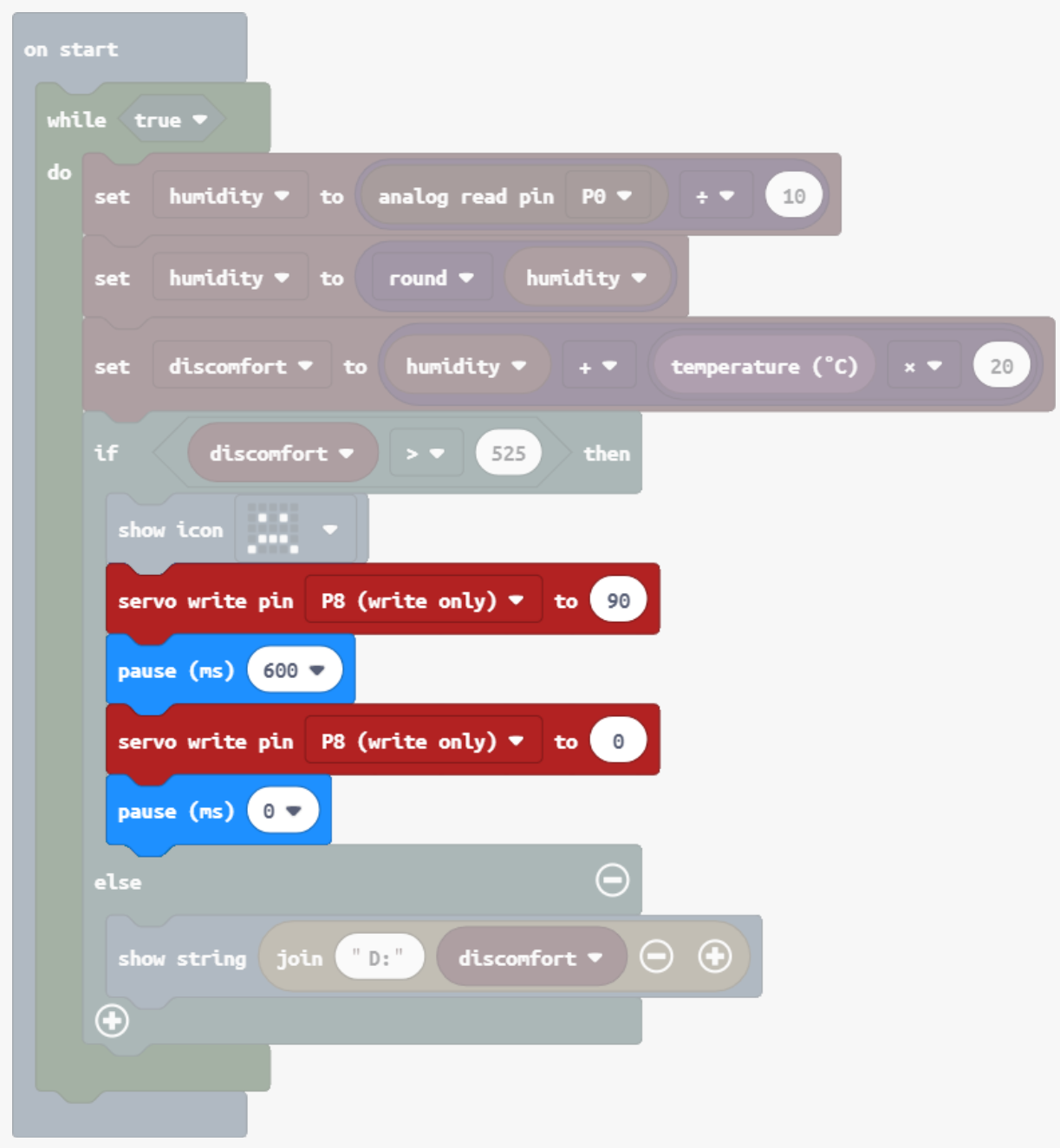


See ACTIVITY 2.2 for an introduction to the **servo motor**.



11. Turn the servo back and forth 90° each time the discomfort is found to be too high.

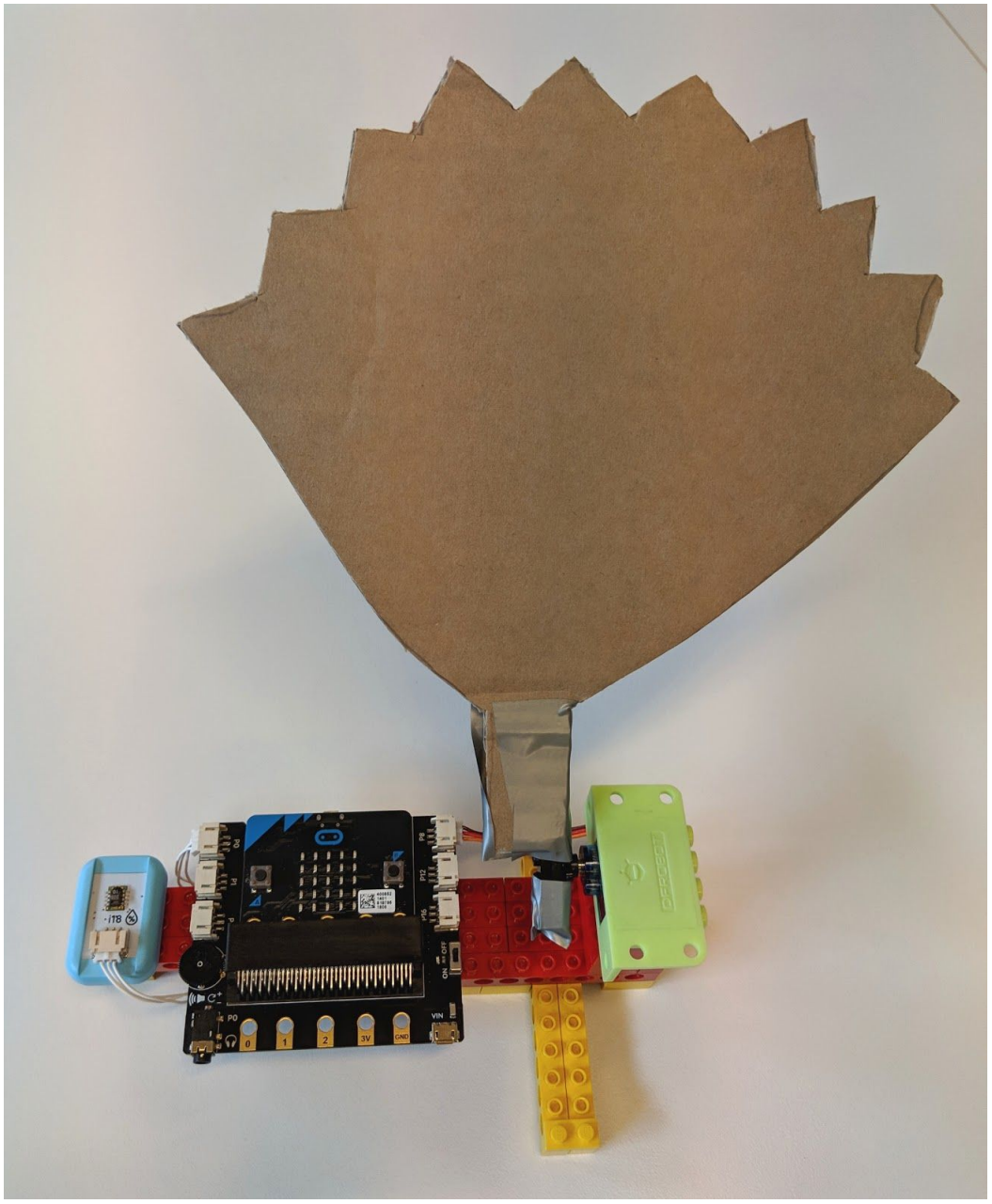
    *TIP: To get enough power, attach USB power to the **VIN** port on the Boson Expansion Board.*



Make the mechanism

12. Use materials of your choice to build a platform for your sensors and fan.

    This construction is built with LEGO, cardboard and duct tape:

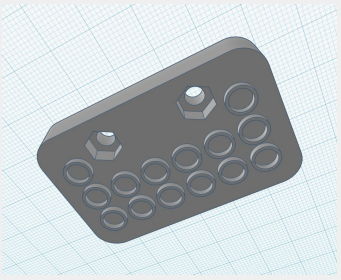


Manufacturing your own parts

If you have access to a 3D printer or laser cutter, you can design and build your own pieces.

This LEGO mount attaches to the Boson Expansion Board with the standard Boson screws and hex nuts.

Find it on Tinkercad here.



“Fan”-made sequels

TINKER

A. Double the fans! Add the Boson Mini Fan to **P12** and have it turn on at the same time as the swinging fan.

B. Edit your code to adjust the formula for **discomfort level**, giving humidity more influence:

$$\text{discomfort level} = (\text{humidity in \%} \times 2) + (\text{temperature in °C} \times 20)$$

C. Instead of scrolling the discomfort level on-screen, use custom icons to quickly show if humidity rose or fell since last checked.

D. Conditions in a room are hard to predict.
Add the rotation sensor to **P1**. Edit your program so that the rotation sensor can adjust the **discomfort** threshold for turning on the fan.

Sensors and servos

JUMP OFF

It’s fun to activate moving parts via sensors!

- One more improvement for our fan system:
    - **Motion sensing system** - Use the motion sensor (Boson PIR Sensor) to deactivate the fan if there is no one around after 30 seconds.
- **Sensor calibrator** - The Boson Temperature Sensor works a little differently from the one on the micro:bit. It does not give degrees Celsius.
    - Write a program to display a reading from both sensors whenever you press button A.
    - Take readings in 3 different places (eg. take one after leaving it in the fridge).
    - Compare the values in each place to discover a common divisor. Use it to convert the readings from the Boson Temperature Sensor into degrees Celsius.
- **Button surprise** - When the user presses the Boson Push Button, a surprise arm swings around to tap them on the hand.
- **Classroom noise alert** - Use the Mini Servo to swing up a “BE QUIET!” sign if the sound level in the classroom is too high.

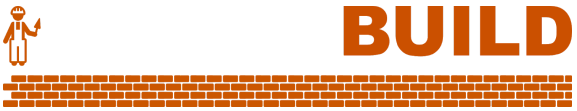
ACTIVITY 3.2 - Portable weather station

Goals

- Create a portable sensor array to collect light, humidity and temperature data.
- Test the Boson Light and Temperature Sensors.
- Use **Serial communication** to send data to computer.
- Use spreadsheet software to analyse and present the data.

JavaScript
parallel document

Python
parallel document



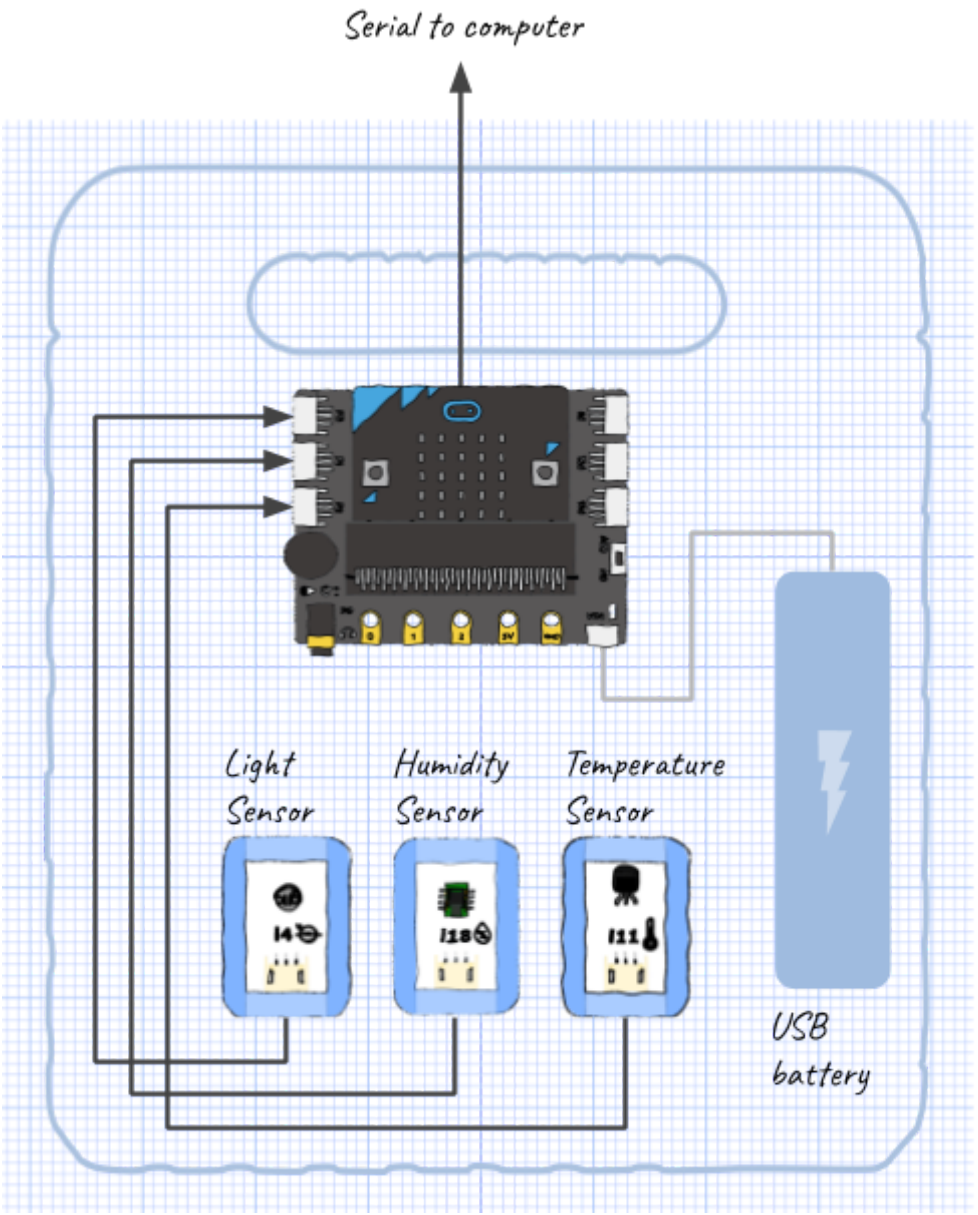
Design overview

The device operates over **2 phases**.

In **Phase 1**, it gathers 20 readings from all three sensors over the course of 3 minutes. (No need for computer connection.)

In **Phase 2**, we connect USB to the computer and the device sends the data over serial.

The data can then be opened in spreadsheet software.



Before you begin

This lesson introduces Serial communication, described in THREE WAYS as Option 2. For this example, we will use the terminal software Tera Term on the computer.

Whichever terminal software is chosen, *using Serial in Windows requires installing software* on the computer.

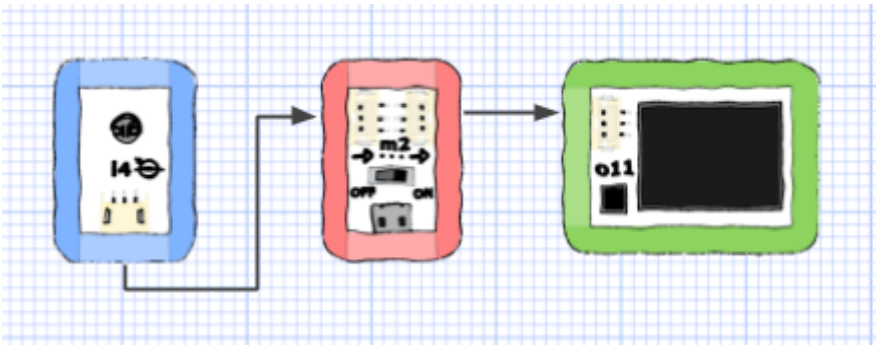
Testing the Light Sensor

BUILD

1. Test the Light Sensor without the micro:bit.

   Attach the Light Sensor through the small Mainboard to the OLED Module as shown.

   (Remember to give the Mainboard power via USB lead.)

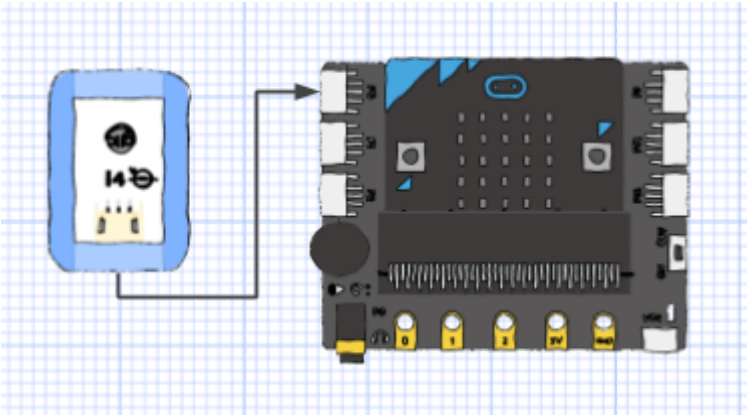


2. Tap the button until you see **Analog Data**. This is the value that will go to the micro:bit.

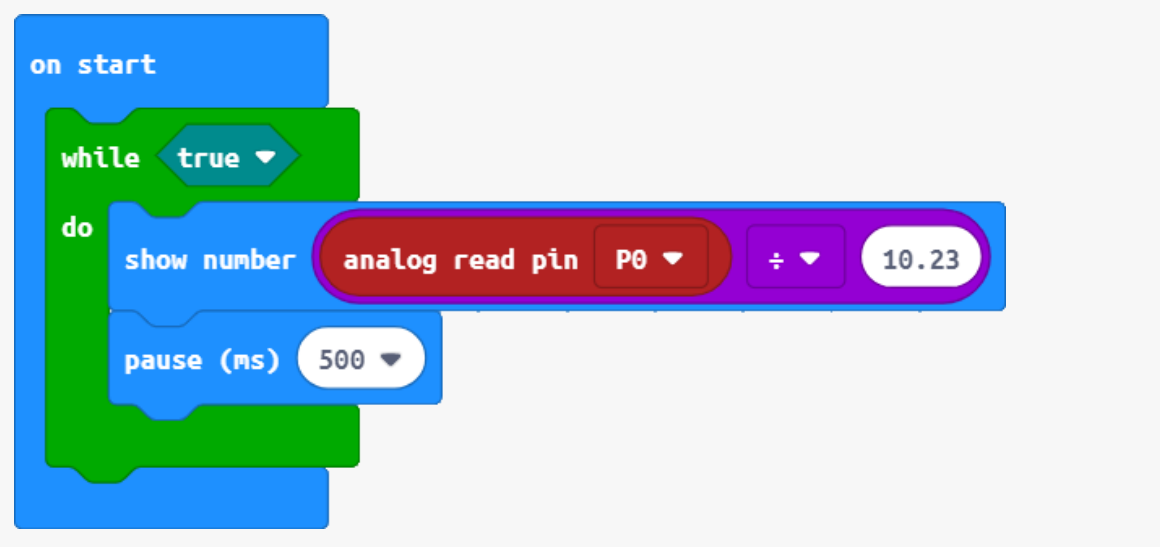
The **Boson Light Sensor** measures light intensity.

It gives a value between 0 (0% intensity) and 1023 (100% intensity).

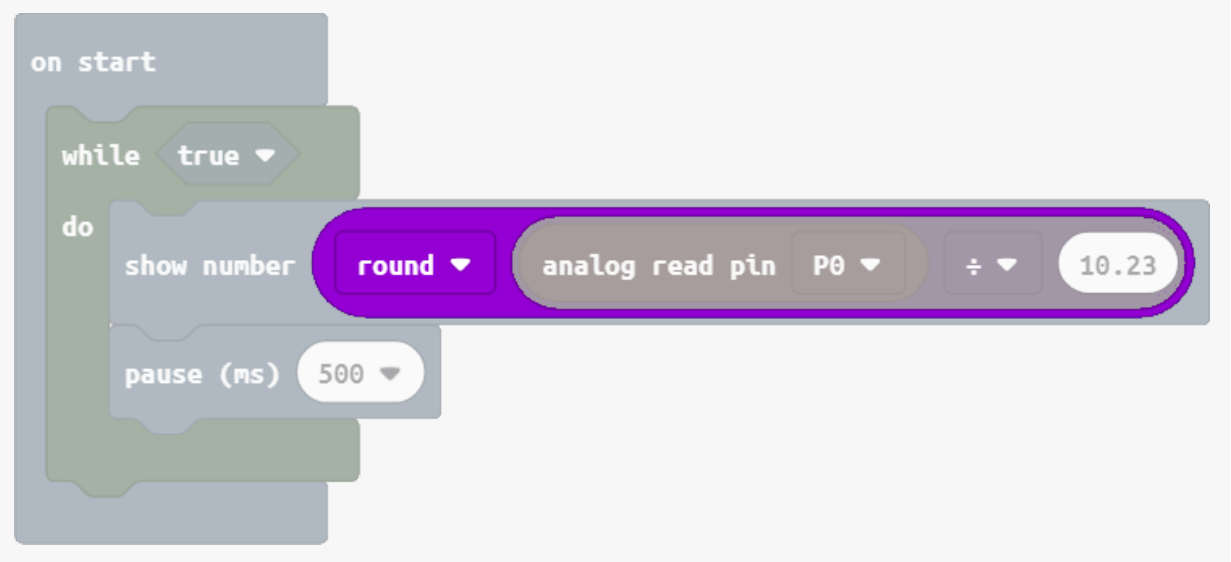
3. Now attach the Light Sensor to **P0** on the Boson expansion board.



4. To convert the range **0** → **1023** into **0** → **100%**, we simply divide by 10.23.



5. As with our reading in ACTIVITY 3.1, we'll use **round** to deal with any long decimals.



Rounding

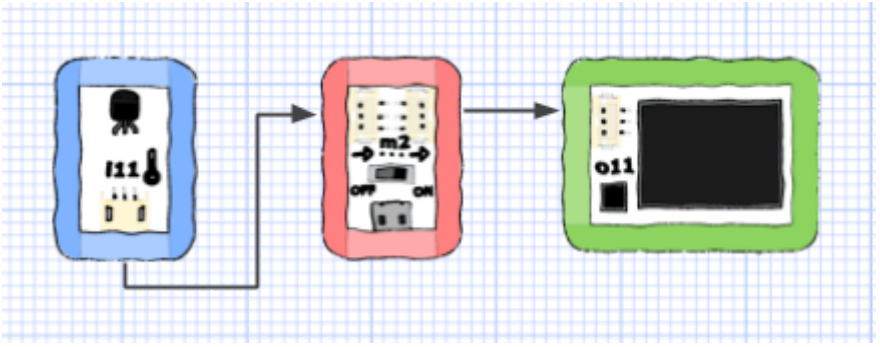
This strict rounding makes sense when using the small LED display on the micro:bit, but it will not be necessary for the serial output.

Testing the Temperature Sensor

4. Test the Light Sensor without the micro:bit.

   Attach the Light Sensor through the small Mainboard to the OLED Module as shown.

   (Remember to give the Mainboard power via USB lead.)



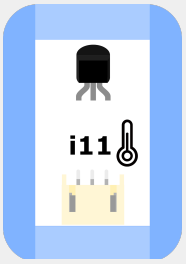
5. Tap the button until you see **i11 Temperature**. The OLED Module presents the reading in degrees Celsius and degrees Fahrenheit.

6. Now tap the button until you see **Analog Data**. This is the value that will go to the micro:bit.

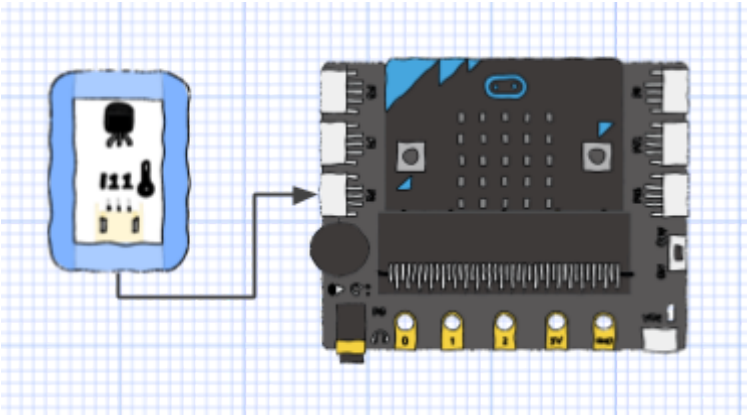
The value from the **Boson Temperature Sensor** ranges between 0 and 1023. How do we convert for temperature?

The rate is 33.33°C per volt. Since the micro:bit operates at 3.3 V, the highest possible value of 1023 would be 109.989°C.

So, take any value and **multiply by 109.989 then divide by 1023**.



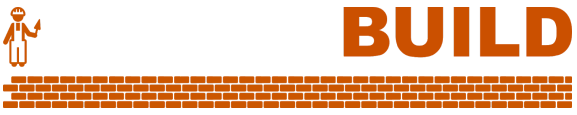
7. Now attach the Temperature Sensor to **P2** on the Boson expansion board.



8. Read in the value and convert to °C: **multiply by 109.989 then divide by 1023**.

9. Again, we'll use **round** to deal with any long decimals.

Practicing Serial output



10. Ensure your micro:bit is plugged into your computer, then run Tera Term. (Tera Term must be installed. It can be downloaded from ttssh2.osdn.jp/index.html.en)

11. Set up the connection as per the instructions at microbit.co.uk/td/serial-library.
    a. Click **File** -> **New Connection…**
        - Choose **Serial**, then press **OK**.
    b. Click **Setup** -> **Serial port...**
        - Set **Speed** to **115200**,
        - Set **Data** to **8 bit**,
        - Set **Parity** to **none**,
        - Set **Stop bits** to **1 bit**,
        - Press **OK**.

12. Let’s send something from the micro:bit to the computer.

    Download this code to the micro:bit as normal. The message should appear in Tera Term.

13. Let’s practice sending three numbers, separated by commas.

The power of serial

Now you can get big amounts of data off the micro:bit super fast.

But that’s not all. With serial, you can even type messages back from the computer to the micro:bit!

What sort of cool things could you do now? Here’s some ideas:

- chatbot
- text adventure
- ASCII art

Putting it all together

14. First, construct your portable unit as per the design overview for this lesson.

    This LEGO unit holds a small battery charger connected to the VIN port on the Boson Expansion Board.

15. Here's our plan.

    Once the code is downloaded to the micro:bit, disconnect from the computer for PHASE 1. Connect to the computer and open Tera Term *before* pressing button B for PHASE 2.

16. Here’s the code for PHASE 1 (collecting the data).

Objects

In the code above, the readings from the three sensors are placed in separate, parallel arrays.

Another approach would be to use **object-oriented programming** (JavaScript and Python only). With this approach, a Reading object would be created every 10 seconds, containing all three readings (light, humidity and temperature). It could also include a timestamp. Each Reading object would then be added to an array for storage.

17. Now add the code for PHASE 2 (sending the data via serial).

Using the data

18. If all went well, your Tera Term is filled with data, nicely separated by commas.

    Click **File** -> **Log…**

    - Set **File name** to **data.csv**,
    - Tick **Include screen buffer**,
    - Press **OK**.

19. You now have a .csv file that can be opened in spreadsheet software like *Microsoft Excel*, *Google Sheets*, or *Numbers*.

    Make a line graph of your data and begin your analysis.

Data reporting

A. Before the serial communication begins, add code to show the average temperature for the whole experiment.

B. Use the micro:bit's own light sensor to take an additional set of readings. Which one reacts faster?

C. Use the micro:bit's own temperature sensor to take an additional set of readings. Which one reacts faster?

Serial for breakfast

Now that you have serial communication, what other data could you collect and analyse?

- **Micro-climate experiments** - Have micro:bits collect data throughout a day (eg. in the house, in shaded vegetation, outside in the sun). Compare the readings through the day. Identify the times of the day when the climate conditions are most different and the times of the day when the conditions are most similar.
- **Data relayer** - Use the radio functionality to set up a micro:bit as a data relayer for other micro:bits. The data relayer immediately sends all received radio messages via serial to the computer.
- **Chatbot** - Use two-way serial communication to turn your micro:bit into a chatbot. It responds to things you say by looking for keywords in your input. You can still use the micro:bit's own features and Boson modules to make things even more fun!
- **Text adventure** - Use two-way serial communication to write a text adventure, with the addition of pictures, sounds and tactile responses using the micro:bit's screen and Boson modules.

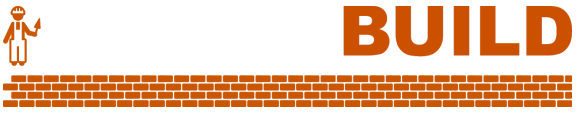
ACTIVITY 3.3 - Robotic plant waterer

Goals

- Create an automated system to water a plant under the appropriate conditions.
- Test the Boson Soil Moisture Sensor.
- Combine factors from multiple sensors to determine watering.
- Control a servo to sprinkle gravity-fed water on a plant.

JavaScript
parallel document

Python
parallel document



Design overview

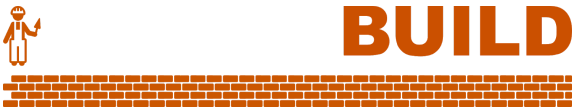
Our system will water a plant only under certain conditions.

It uses:

- light intensity from a **Light Sensor** to determine if it is the best time to water,
- soil wetness from a **Soil Moisture Sensor** to factor into watering frequency,
- air humidity from a **Humidity Sensor** to factor into watering frequency.

When it is time to water, a **Mini Servo** rotates the end of a flexible pipe from a container of water.

Testing the Soil Moisture Sensor



1. Test the Soil Moisture Sensor without the micro:bit.

   Attach the sensor through the small Mainboard to the OLED Module as shown.

   (Remember to give the Mainboard power via USB lead.)

Working with water

Water and circuits don't mix!

- Water should *not* touch the BBC micro:bit or Boson Expansion Board.
- Water should *not* touch the main part of any Boson module.
- Only dip the metal rods at the bottom of the Soil Moisture Sensor. Do *not* bury the whole module.

2. Tap the button until you see **Analog Data**. This is the value that will go to the micro:bit.

   Note down the values when you dip the sensor in dry soil, wet soil and water. Your body is also conductive!

The **Boson Soil Moisture Sensor** measures the conductivity between its two rods. Since water is an effective conductor of electricity, wet soil can be distinguished from dry soil.

Theoretically, the sensor can give a value between 0 (driest) and 1023 (wettest).

3. Now attach all three sensors to the Boson expansion board:
    - Light Sensor to **P0**
    - Humidity Sensor to **P1**
    - Soil Moisture Sensor to **P2**

4. Our first program will simply keep displaying the readings from each sensor.

5. Here's our plan. We check every second to decide if another short watering is needed. NOTE: In this solution, we won't vary watering time. Each watering is short.

```
REPEAT forever

    lightIntensity ← read from sensor P0
    airHumidity ← read from sensor P1
    soilMoisture ← read from sensor P2
    airTemperature ← read from micro:bit

    Calculate airDrynessFactor from airHumidity
    Calculate soilDrynessFactor from soilMoisture
    Calculate temperatureFactor from airTemperature
    Decide waterNeeded based on airDrynessFactor, soilDrynessFactor and
        temperatureFactor

    IF waterNeeded AND lightIntensity indicates dusk / dawn THEN
        Water the plant a little
    END IF
    Wait for 1 second

END REPEAT
```

6. We'll begin by storing the raw sensor values in variables.

7. Send the values to the computer as we go. See ACTIVITY 3.2 on serial communication.

8. Calculate **factors** to help decide if watering is needed. This means weighing the influence of the air humidity, soil moisture and air temperature.

9. Finally, it’s time to set a boolean variable **waterNeeded** for whether to water now or not.

10. Attach the mini servo at **P8**.

See ACTIVITY 2.2 for an introduction to the **servo motor**.

We will use it to rotate the end of a flexible tube with gravity-fed water from a container.

11. A simple function **waterPlant** will rotate the servo to water the plant for a fixed time.
    You may need to adjust the angles depending on your built design.

12. Dawn and dusk are often considered the best times to water a plant. We'll activate watering only if the light sensor reading is appropriate *AND* the **waterNeeded** boolean is true.

13. This is the complete program.

14. One approach to the watering system is a gravity-fed, flexible tube from a container of water.

    The construction below is built from cardboard and small wooden paddles, with a simple flexible tube from a hardware store.

Water variation

A. Modify the formula for **temperatureFactor** so it might better suit tropical plants:

   temperatureFactor = (airTemperature - 28) × 45

B. Create a counter to record how many waterings have happened since the program started. Show the counter every time button A is pressed.

C. Measure how many millilitres of water each second of watering delivers. Change the counter from B above to record litres used since the program started.

D. The present watering mechanism can lead to some spills as the watering lever drops down, come up with a solution to this mechanical problem.

The seed of an idea

Smart robotic gardens are one approach to saving water and giving plants individual care.

- **More features for the watering system**
  - Create an alarm reminding the user to refill the water container. The alarm could be set off after multiple waterings that did not prevent the soil from getting drier (ie. empty waterings).
  - Log the amount of watering required over a period of days or weeks, and see if you can correlate it with weather conditions over the time.
- **Seed germination experiment** - Perform an experiment with two sets of seeds planted in different soil conditions (eg. wetter vs. drier soil, brighter vs. darker chamber). Use the sensors to monitor the conditions and keep them stable so that your experiment has accuracy.
- **Automated greenhouse -** Build a small greenhouse model. Use mini servos to open windows when the air or soil inside is too hot.

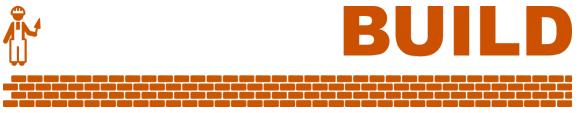
ACTIVITY 3.4 - Lie detector

Goals

- Create a lie detector that detects changes in heart rate.
- Test the Boson Heart Rate Sensor.
- Use a **queue array** to maintain a running average of 9 heartbeats for active heart rate monitoring.

JavaScript
parallel document

Python
parallel document



Design overview

We will build a lie detector with:

- a heart beat sensor to detect each heart beat from a finger,
- an RGB LED strip to show whether the heart rate is higher or lower than a baseline.

Testing the Heart Rate Sensor

1. Attach the Heart Rate Sensor to **P0** on the Boson expansion board.

The **Boson Heat Rate Sensor** is digital. For each heartbeat, it gives an ON (1) value, then an OFF (0) value.

It also shows the reading on its own blue LED.

- *Always use the **VIN USB port** to provide power to the Boson Expansion Board.*
- *Place a **finger** on the sensor as shown. Do not press too hard.*
- ***Hold still** for at least 10 seconds while the sensor adjusts to your pressure.*

2. We'll use a loop to plot a single LED when a heartbeat comes in from the sensor.

3. You may find the reading jitters a bit on the way down, even after the sensor adjusts to your pressure.

   Let's improve the code so that it waits while the reading is still 1, then pauses for 250 ms once the reading drops to 0.

Why not the Heart icon?

We're using an LED **plot** rather than a **show icon**. This is because the show icon block contains an in-built delay of 400ms.

If you want to remove that delay: Switch to Javascript, add `,0` to the end of the command, then switch back.

4. We'll count beats over a 15-second period, then multiply by 4 to get the beats per minute.

```
Scroll "Place finger."
Wait 10 seconds for the sensor to adjust
Show a tick icon

numberOfBeats ← 0
REPEAT for 15 seconds only
    IF a heartbeat is detected
        Add 1 to numberOfBeats
    END IF
END REPEAT

heartRate ← numberOfBeats × 4
Scroll heartRate, " bpm"
```

5. Here's the start code.

6. To loop for 15-seconds only, use a variable **finishingTime** and the micro:bit's running time.

7. Now put in the heartbeat code from before, to show each heartbeat on the display.

8. We can now add code for recording the number of heartbeats, and calculating the heart rate.

Ongoing heart rate

9. So far, our code can determine heart rate once after a period of time. To actively monitor heart rate, we'll need something more sophisticated.

1

Set up an array for recording 10 **beat times**.

Each of these is a record of the running time when a beat occurred.

Initialise all values to 0.

2

When we detect a heartbeat:

- remove the first element from the array,
- push the recorded time for the heartbeat onto the end. This shuffles all the others up.

3

Keep going. Soon the array is filled with recorded heartbeat times (measured in milliseconds since the program began).

4

Now we can find the time interval between 9 heartbeats at any time. Subtract the earliest (index 0) from the latest (index 9).

nineBeatInterval = 20140 - 12882
= 7258 ms

averageInterval = 7258 ÷ 9
= 806 ms

From here, we can calculate the heartrate.

heartRate = 60000 ÷ 806
= 74 beats per minute

10. Start with our earlier code for displaying heartbeats.

11. Now, add the array code.

Queue

This way of using an array is called a **queue**.
Values are pushed onto one end, and shifted off the other end.

12. Add the calculation code.

13. To check the calculated **heartRate**, simply add an event to display it whenever button A is pressed.

    Remember to wait until at least 9 heartbeats have been recorded.

14. If heart rate rises when a person is lying, we'll need to get a *baseline* heart rate first. We'll wait 35 seconds from when the program starts before storing this.

    Here's the overall plan:

```
Scroll "Place finger."
Wait 10 seconds for the sensor to adjust
Show a tick icon

baselineHeartRate ← -1
REPEAT forever
    Calculate heartRate using heartbeat array (see previous program)
    IF baselineHeartRate = -1 THEN
        IF running time > 35 seconds THEN
            baselineHeartRate ← heartRate
        END IF
    ELSE
        Set RGB LED strip to show deviation of heartRate from baselineHeartRate
    END IF
END REPEAT
```

15. Connect the LED RGB Strip to **P8** on the Boson Expansion Board.

16. Add starting code to the previous program, and tidy the heart rate calculation into a function.

17. Introduce the **baselineHeartRate** variable.

18. Finally, add the code to set up and use the RGB LED strip. Show three green lights for the baseline heart rate, then add a light for every 2 beats per minute the heart rate rises above it.

Listen to your heart

A. Use the solution to find someone's heart rate after some exercise.

B. Based on the result in part A, change the code so that all the lights turn blue when the "activity" heart rate is reached again.

C. Add a sound for each heart beat, so that you can listen to your own heartbeat like a stethoscope.

Heart and queue

What else could you do with the skills in this activity?

- **Experimenting with the design**
    - Place the waterproof temperature sensor in warm water and have a person place one hand in the water as you add ice. Is there a correlation between their heart rate and the temperature of the water?
    - Use the waterproof temperature sensor in a different way - to detect subtle rises in a person's skin temperature when nervous.
    - Activate and increase the speed of a mini fan when heart rate goes up.
- **Heartbeat music** - Use the **set tempo** command to match the beat of music to a heartbeat.
- **Message queue** - Create a queue array for messages received over the micro:bit's radio. Received messages are pushed onto the queue, then shifted off and displayed whenever a button is pressed.

ACTIVITY 3.5 - Perfect pH advisor

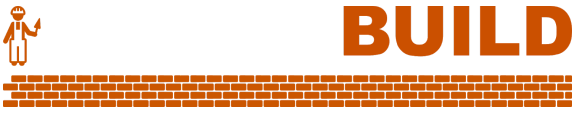
Goals

- Set up the micro:bit to guide you to a desired pH level in a solution.
- Test the Boson pH Sensor.
- Test the Boson Waterproof Temperature Sensor.
- Use the micro:bit buttons to set a desired pH level.

JavaScript
parallel document

Python
parallel document

Design overview



Use **Rotation Sensor** to choose a target pH for a liquid, between 5.5 and 8.5.

The micro:bit actively monitors

- pH of the liquid with **pH Sensor**,
- temperature of the liquid with **Waterproof Temperature Sensor**.

The micro:bit advises on its display if the liquid is too acidic, too alkaline, or on target.

Before you begin

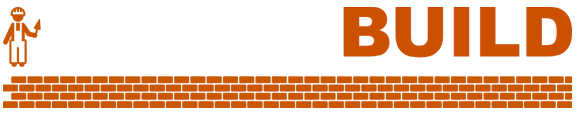
Liquid and circuits don't mix!

- Liquid should *not* touch the BBC micro:bit or Boson Expansion Board. Liquid should *not* touch the main part of any Boson module.

No **hot** liquid!

- Sensor leads and casings are not made for high liquid temperatures. Keep your liquid below 50°C.

Testing the pH Sensor



1. Carefully remove the cap from the pH Sensor tube (without spilling the fluid inside), then dip the exposed end of the tube in water.

   Attach the sensor through the small Mainboard to the OLED Module as shown. *(Remember to give the Mainboard power via USB lead.)*

2. Tap the button until you see **i17 PH**.

   Tap water should have a pH between 6.5 and 8.5.

The **Boson pH Sensor** is a delicate instrument.

- Avoid sticky or sugary liquids.
- When done,
  - *rinse* the end of the tube under water,
  - *reattach* the cap to the tube.
    The cap contains a Potassium Chloride solution (3.3 mol/L) to keep the sensitive electrode moist.

3. Attach the pH Sensor to **P1** on the Boson expansion board.

4. We'll use a loop to keep scrolling the analog value from **P1**.

5. To convert this value to pH, multiply by 14, then divide by 1023.

6. Our pH value has too many digits to display nicely. To round it to 1 decimal place, we'll need to break it into a whole digit and a decimal digit.

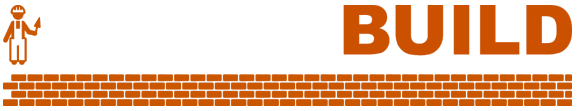
$$pH = 7.4$$

$$wholeDigit = 7$$

$$\begin{aligned} decimalDigit &= (pH - wholeDigit) \times 10 \\ &= (7.4 - 7) \times 10 \\ &= 0.4 \times 10 \\ &= 4 \end{aligned}$$

7. Finally, we can show both digits by concatenating them with a '.' in between.

Testing the Waterproof Temperature Sensor



8. Test the Waterproof Temperature Sensor without the micro:bit.

   Dip the end piece in water and attach the sensor through the small Mainboard to the OLED Module as shown. *(Remember to give the Mainboard power via USB lead.)*

9. Tap the button until you see **i19 Temperature**. (Do not read **i11 Temperature**!)

   Dry the end piece and hold it in your hand to see the difference as it warms.

10. Attach the Waterproof Temperature Sensor to **P0** on the Boson expansion board.

11. Again, we'll use a loop to keep scrolling the analog value. What do you notice about higher temperatures (inside your hand) and lower temperatures (in the water)?

The **Boson Waterproof Temperature Sensor** gives very different analog values to the other temperature sensor we used in ACTIVITY 3.2.

As shown in the graph below, the relationship is *curved* and *negative* (value is lower when temperature is higher).

How can we convert with these analog values to temperatures?

- One option would be to find the formula for the whole relationship. But Microsoft Excel's best matching trendline is a 4th-order polynomial!
- An easier option is to isolate the useful temperature range (0°C to 50°C) and estimate with a simple linear equation.

12. Let's invert the formula to convert the analog value to degrees, then output it.

$$value = -10.2 \times temperature + 795$$

$$\therefore temperature = (value - 795) \div -10.2$$

13. The full program has two phases: **set the target pH**, then **monitor and advise**.

```
Scroll "Choose target."
REPEAT until Button B is pressed
    targetPH ← Rotation Sensor reading mapped between 5.5 and 8.5
    Scroll targetPH
END REPEAT
Scroll "Target set."

REPEAT forever
    actualPH ← pH Sensor reading
    IF actualPH < targetPH - 0.1
        Scroll "Too acidic."
    ELSE IF actualPH > targetPH + 0.1
        Scroll "Too alkaline."
    ELSE
        Show tick icon
        Scroll "On target!"
    END IF
END REPEAT
```

14. Attach all three sensors as per the complete diagram at the start of this activity.

Common household items like vinegar and sodium bicarbonate can be added to adjust the pH of water.

15. The rotation sensor on **P2** gives a reading from 0 → 1023.
    Map this to 5.5 → 8.5 for the target pH.

16. Place the decimal display code we developed earlier into a function to display the target pH.

17. Next, begin the second phase of the program.

18. The guidance logic is simple. The actual pH is good if it is within 0.1 either side of the target.

My eyes! The goggles do nothing!

A. Replace the current messages “too acidic” and “too alkaline” with icons for quicker advice.

B. Allow the user to press Button A to display the actual pH at any time.

C. Using the code developed for testing the Waterproof Temperature Sensor, add icons for “reduce temperature” and “raise temperature” on the micro:bit display, after the pH icons. Assume the desired temperature is 20°C.

D. Add the LED strip to show the temperature variance from part C, instead of icons.

More experimentation needed

What else could you do with the skills in this activity?

- **Further improvements to the pH advisor:**
    - Use serial communication to add data logging to the system. For example, you could gather evidence for whether fizzy drinks lose their acidity as they go flat.
      (Note, you may find that carbonated water has a different result to cola, which also contains phosphoric acid.)
    - Use a traditional pH indicator to verify the accuracy of the pH sensor.
    - Add temperature compensation. Measured pH actually varies depending on the temperature of the liquid being measured. A bit of Maths can be used to compensate for this, so that all pH readings are equivalent to room temperature (eg. 25°C).
- **Aquaponics monitor** - pH is one of the measurements used to monitor aquaponics and hydroponics systems. The pH sensor can form one part of a complete system.
- **Triple thermometer** - You now have access to three means of measuring temperature; the Boson Waterproof Temperature Sensor, the Boson Temperature Sensor and the micro:bit’s own temperature sensor. Using radio and/or serial communication, take some accurate measurements to compare the accuracy and speed of the three sensors.

MORE MODULES

Looking for more inputs and ouputs to connect to the micro:bit?

1 The **Gravity** range of modules can be connected to the Boson Expansion Board, even with the same leads.

The range includes over 100 sensors and actuators, including sensors for touch, magnetic fields, tilting, gases and air, flame, distance, joystick control and many more.

Note:

- Not all gravity sensors may work with the micro:bit. They are relatively cheap and can be purchased individually for testing.
- Gravity sensors do not have the Boson child-friendly, magnetic backpieces.
- See also the Micro:Mate, a smaller expansion board for micro:bit perfect for Gravity's own leads.

2 The **Boson Inventor Kit** contains additional modules for input (tilt, self-locking switch) and output (ultra bright LED, buzzer, Gared motor).

It also includes various logic modules, which function like logic gates between other connections.

3 Build your own sensors and circuits out of cardboard, conductive tape or thread, and raw electronic components.

Use alligator clips to connect to the large pins on the bottom of the micro:bit, or at the bottom of the Boson Expansion Board.

For more serious circuitry, try your hand at breadboarding and prototyping. Both types of expansions are readily available for the micro:bit.

APPENDIX A - TINKER SOLUTIONS

Solutions for challenges.

ACTIVITY 1.1

A. Make a 4 second pause between each “Hello”.

```
from microbit import *
while True:
    display.scroll('Hello,')
    sleep(4000)
```

B. Instead of looping forever, limit the loop to exactly 5 times.

or

```
from microbit import *
for index in range(5):
    display.scroll('Hello,')
```

C. Now change the loop to happen 8 times.

or

```
from microbit import *
for index in range(8):
    display.scroll('Hello,')
```

D. Have the program show the index as it loops: “Hello 0”, “Hello 1”, “Hello 2”, etc.

```
from microbit import *
for index in range(6):
    display.scroll('Hello ' + str(index))
```

ACTIVITY 1.2

A. Change just the last line of the program so that the answer is always “Don’t count on it.”

“Don’t count on it” is at position **5** in the array, because the first value “It is certain” is at position **0**.

B. Add some more answers of your own. Remember to increase the limit of the random number.

C. Put in a loop so that the program starts again, without you having to press reset.

D. Modify the program to have a cheat. Pressing button B always gives “It is certain”.

We set the empty loop to continue to wait as long as *neither* button A nor button B is pressed.

Then, we use **if/else** to check which of the two buttons was actually pressed.

ACTIVITY 1.3

A. Between 2 and 4 seconds is too long to wait at the start. Change it to between 1 and 3 seconds.

B. Change the program so that it shows a smiley face every time the user achieves a time lower than half a second, a meh face if lower than a second, and a sad face otherwise.

C. Put in a loop so that the program starts again, without you having to press reset.

D. Now limit the game to exactly 3 rounds, then say “Game over”.

A. For fun, have the pixel light up in a different random position each time you play.

ACTIVITY 1.4

A. Change the game to count touches on one of the micro:bit's pins, instead of the button.

B. Make it a random time limit. Don’t forget to tell the player the time limit before the game starts.

C. Give the player a target to win the game. eg. 15 presses in 5 seconds.

D. Run the original game three times. When finished all three games, display the average score.

The solution is to use an array **scores** to hold the scores.

The index for the loop is called **round**, because it is the number for each round of the game (Round 0, Round 1 and Round 2).

As soon as each round is complete, we still display the score, but we also push that score onto the array **scores**.

Finally, when all rounds are done, we calculate the average by first summing the scores stored in the array at positions 0, 1 and 2.

ACTIVITY 1.5

A. Detect a collision. A collision is when the player and the brick are in exactly the same position. When this happens, blink the overlapping LED two times, then resume the game.

   *We add code to the **main routine**.*

B. Create a variable to count collisions. End the game at exactly 3 collisions.

*Again, add code to the **main routine**.*

C. Create a score variable and increase it by 1 every time the player successfully dodges the brick. Print the score when the game is over.

*Introduce score at the start of the **main routine**.*

*The **moveBrick** function is where we discover the brick reaching the bottom, so increase score there.*

*Finally, add the score printout at the end of the **main routine**.*

D. When the game is over, congratulate the player if the score is greater than 20.

*Again, add code to the **main routine**.*

ACTIVITY 1.6

A. Add extra check to the end of the program.

B. Add a function to display dots based on **myRoll**'s value.

C. Be sure to include the resetting of the roll variables to 0 inside the loop.

D. Add a variable **myWins** to keep the number of wins.

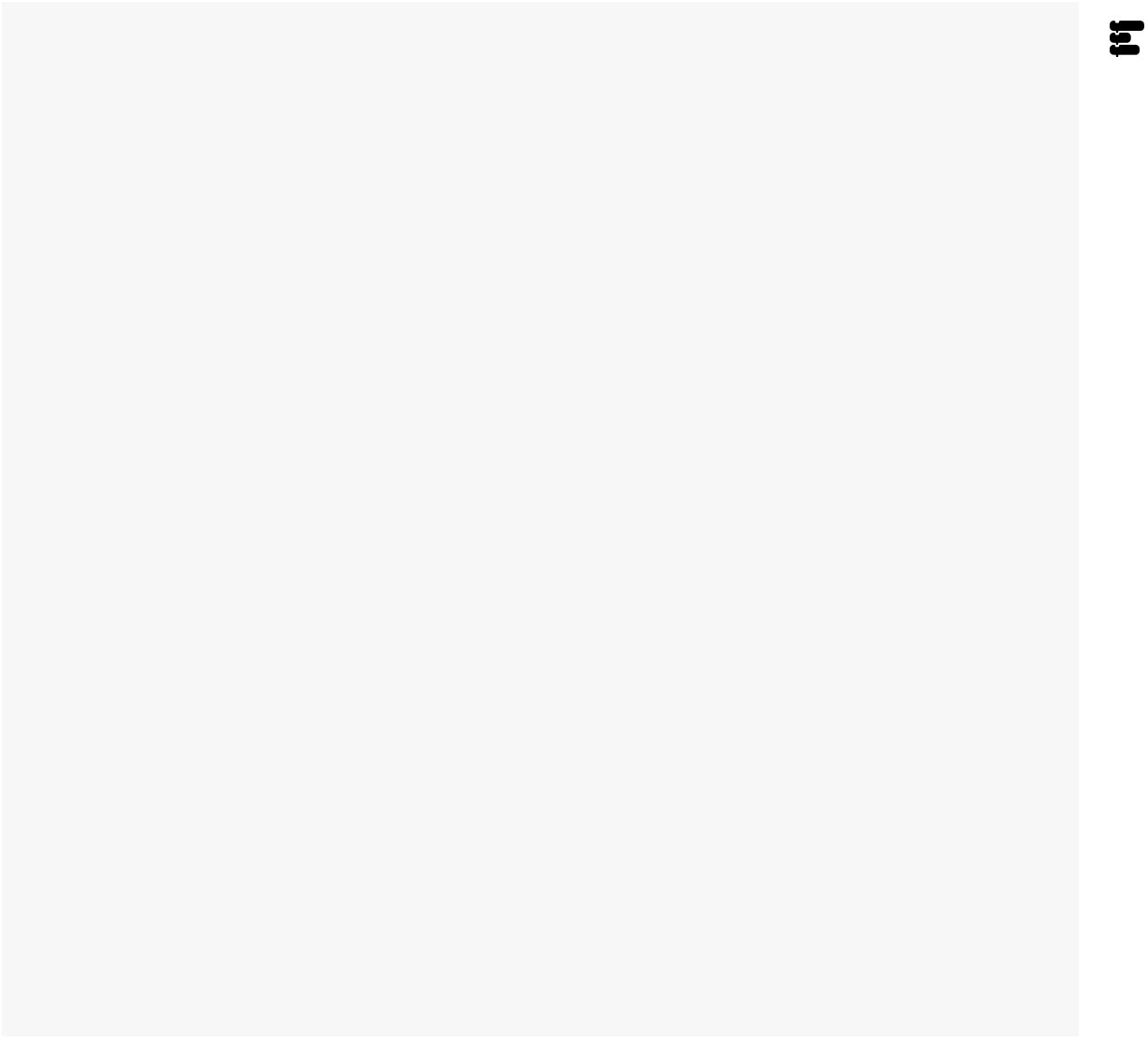
E. Write a function to show icons for a ticking clock.

ACTIVITY 2.1

A. No code change needed. Simply swap the modules.

B. A simple solution is to use the in-built graph command (Visual and JavaScript only).

*Add code to the **main routine**.*



Alternatively, write separate code for a custom solution. This function plots the sound level as a crude bar graph using 5 horizontal lines. Could you improve it?

Is the micro:bit's display too limited?

With just one colour (red), it might seem like only a small amount of detail can be conveyed.

But remember:

- Each pixel can be lit separately.
- Each pixel can have its own brightness level (0 to 255).

C. Add a simple function to toggle the fan on and off, and use an empty to loop to wait until button is released.

D. Use a variable **clapThreshold** for the number where a clap is detected, then set this variable from the rotation sensor.

*Add code to the **main routine**.*

ACTIVITY 2.2

A. This simple solution chooses random positions between 25° and 75° for the servo. We've also shortened the pause to 500 ms while the servo is moving.

*Change code in the **main routine**.*

B. Attach the LED to **P12** on the Boson Expansion Board.

This new flashing function replaces the 500 ms pause while the servo is moving.

C. Add a simple check so the main program only proceeds when there is enough light.

D. Rename the variable **noOfWaves** to **noOfActivations,** and simply move the change command.

ACTIVITY 2.3

A. This requires a simple adjustment to the lowest LED’s condition:

B. Inside the loop, set the brightness so that your sensor’s upper limit is mapped down to 255. In this example, we divide the reading by 2.4.

C. Use a boolean variable **brightnessActive** to remember whether the brightness effect is turned on.

D. Add the command inside the loop.

ACTIVITY 2.4

A. The wait time affects the pitch, because you are changing the wavelength of the soundwave.

B. This program produces a falling siren sound.

C. Reversing the PewPew makes a car alarm sound.

D. Add a check for button A inside the **doFrequencyKnob** function.

E. The fourth mode is small enough that it doesn't require a separate function.

F. Remember to set up the LED strip at the start of the program.

ACTIVITY 2.5

A. Add the necessary Music command into all three micro:bit programs.

The existing code below is from micro:bit **C***, but the added code is the same for all three.*

B. Here’s the complete code for microbit **C** (motion sensing) so far.

C. Modify the **doLightSensing** function:

D. Check the incoming message for the word “ALARM!”, then trip this micro:bit too.

The existing code below is from micro:bit **A***, but the added code is the same for all three.*

ACTIVITY 3.1

A. Activate the mini fan at full speed by writing 1 (on) to **P12**. Deactivate it by writing 0 (off).

B. Simply modify the formula for calculating **discomfort**:

Order of operations

Imagine each Math block surrounded by brackets. The operation inside the block will be performed first. But the purple lines can be subtle and easy to miss.

For the above example:

✓ correct

✗ incorrect

C. Use a variable **previousHumidity** to keep the most recent humidity reading for comparing.

D. The rotation sensor on **P1** gives a reading between 0 and 1023, so the **discomfort** value can be compared directly with it. Just a small turn of the knob will affect the sensitivity of the system.

ACTIVITY 3.2

A. Write a function **calculateAverageTemperature** to do the Maths work. Call it from the main program.

B. Add a new variable **internalLight**, and a new array **internalLightReadings**.

The micro:bit's sensor ranges from 0 to 255, so divide by 2.55 to get a percentage value.

C. Add a new array **internalTempReadings**.

The micro:bit gives temperature in degrees Celsius directly.

ACTIVITY 3.3

A. Just a simple adjustment to the Maths for **temperatureFactor**:

B. Add a variable **noOfWateringsDone**, and increment it after each watering.

C. The solution below supposes that each watering uses 41mL.

Replace the variable **noOfWateringsDone** with **noOfLitres** and adjust the increment.

D. Engineered solutions will vary.

ACTIVITY 3.4

A. Everyone is different. You may find an “activity” heart rate between 100 and 170 bpm.

B. Add a check to your **updateLEDStrip** function. *We will use 140 as the “activity” heart rate.*

C. A short tone is enough. *Add to the **main program**.*

ACTIVITY 3.5

A. In the solution below, we invert the advice with an icon for “add more sodium bicarbonate” or “add more vinegar”.

B. Create a similar function to **displayTargetPH**, then call it when Button A is pressed.

C. The temperature check follows the pH check. A short pause ensures the icons are seen.

D. Connect the LED strip to **P8**.
Light up the central LED in the strip **green** when the correct temperature is reached. Light LEDs below **blue** for lower temperatures and LEDs above **red** for higher temperatures.

Program continues over page.

APPENDIX B - COURSE METHODOLOGY

The pedagogy behind this course.

Build → Tinker → Jump Off

The sequence of , TINKER and JUMP OFF is a deliberate reflection of successful classroom practice in the experience of the writers of this resource.

In the stage, concepts are taught progressively with all code and connections given.

Rather than have students merely copy the code, use this time to explore. eg .What would happen if you changed this value? Can we make it work with the other button instead?

The TINKER stage is a series of challenges (some easy, some more challenging) to modify the program, usually to augment and improve the solution made earlier.

There are often multiple solutions, but sample solutions are given in APPENDIX A.

JUMP OFF contains ideas for fresh projects based on the skills covered so far, as well as occasional overhaul ideas for an existing project.

Depending on complexity, the suggestions in this section may take one lesson or many lessons, or could form the basis for a large project.

Single sequence code

Like MIT's *Scratch*, the environment at makecode.microbit.org allows for event trigger blocks (event handlers) such as **on button pressed** and **forever**. However, this course keeps all code in only one sequence, wherever possible, for the following reasons:

1. **Encouraging better code sequence understanding.** Misuse of event triggers can lead to messy and unmanageable code, difficult to read and debug. While it is common for user interfaces to be coded with multiple threads, a single thread encourages careful thinking in planning the flow of the program.

2. **Avoiding *race conditions*.** Event triggers can often lead to situations where two code threads are attempting to read or write to the same resource or variable, leading to unpredictable results. Even worse, in the makecode environment, a thread of code without pauses of any kind can prevent other threads from running at all.

3. **Readability of code.** While event triggers look attractive and readable as visual code, they appear in JavaScript as *anonymous functions*, a relatively advanced technique that is difficult to read and understand before the introduction of regular functions.

4. **Integrity across languages.** Event triggers are not available in Python for micro:bit.

✗ Using event trigger blocks

The button cannot interrupt the bar graph to display the temperature as text. Both threads are trying to access the display simultaneously.

✓ Using one sequence

A loop ensures that the program keeps checking if button A is being pressed. The bar graph does not return until after the string has been shown.