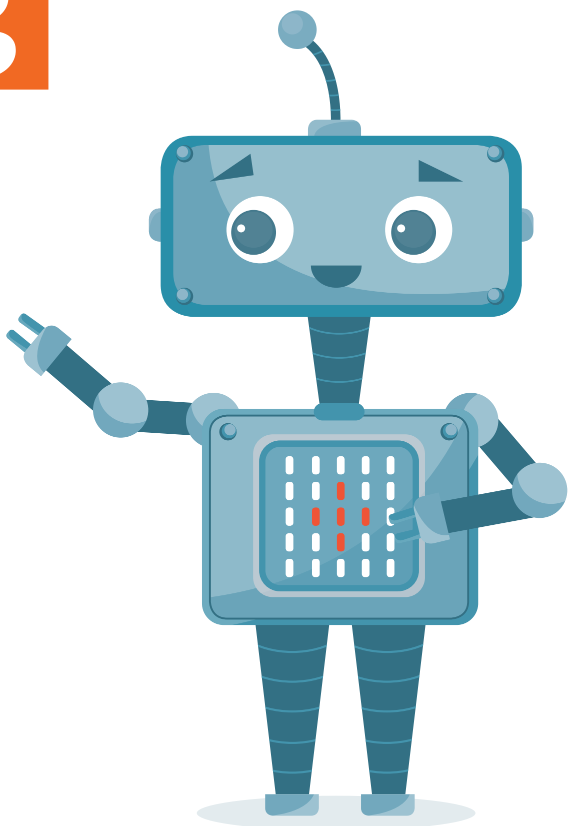
arm School Program

micro:course

learn computing with the micro:bit



Rob Leeman

Arm Education Media is an imprint of Arm Limited, 110 Fulbourn Road, Cambridge, CB1 9NJ, UK

Copyright © 2022 Arm Limited (or its affiliates).
All rights reserved.

No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or any other information storage and retrieval system, without permission in writing from the publisher, except under the following conditions:

Permissions

- You may download this book in PDF format from the Arm.com website for personal, non-commercial use only.
- You may reprint or republish portions of the text for non-commercial, educational or research purposes but only if there is an attribution to Arm Education.

This book and the individual contributions contained in it are protected under copyright by the Publisher (other than as may be noted herein).

Notices

Knowledge and best practice in this field are constantly changing. As new research and experience broaden our understanding, changes in research methods and professional practices may become necessary.

Readers must always rely on their own experience and knowledge in evaluating and using any information, methods, project work, or experiments described herein. In using such information or methods, they should be mindful of their safety and the safety of others, including parties for whom they have a professional responsibility.

To the fullest extent permitted by law, the publisher and the authors, contributors, and editors shall not have any responsibility or liability for any losses, liabilities, claims, damages, costs or expenses resulting from or suffered in connection with the use of the information and materials set out in this textbook.

Such information and materials are protected by intellectual property rights around the world and are copyright © Arm Limited (or its affiliates). All rights are reserved. Any source code, models or other materials set out in this reference book should only be used for non-commercial, educational purposes (and/or subject to the terms of any license that is specified or otherwise provided by Arm). In no event shall purchasing this textbook be construed as granting a license to use any other Arm technology or know-how.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its affiliates) in the US and/or elsewhere. All rights reserved. micro:bit is a trademark of the Micro:bit Educational Foundation. Other brands and names mentioned in this document may be the trademarks of their respective owners. For more information about Arm's trademarks, please visit https://www.arm.com/company/policies/trademarks.

Arm is committed to making the language we use inclusive, meaningful, and respectful. Our goal is to remove and replace non-inclusive language from our vocabulary to reflect our values and represent our global ecosystem.

Arm is working actively with our partners, standards bodies, and the wider ecosystem to adopt a consistent approach to the use of inclusive language and to eradicate and replace offensive terms. We recognize that this will take time. This book may contain references to non-inclusive language; it will be updated with newer terms as those terms are agreed and ratified with the wider community.

Contact us at school@arm.com with questions or comments about this course. You can also report non-inclusive and offensive terminology usage in Arm content at terms@arm.com.

ISBN: 978-1-911531-45-6

For information on all Arm Education Media publications, visit our website at
https://school.arm.com

To report errors or send feedback, please email
school@arm.com

CONTENTS

GETTING STARTED

This book introduces learners to the world of making and programming through a series of real-world challenges that feature the micro:bit.

Learners study the building blocks of Computer Science in a practical and engaging way by making real-world products and solving real-world problems using code. Learners start using a “block-based” language called MakeCode to learn the fundamental concepts of Computer Science whilst applying them to the physical world. Learners then move on to using Python, a text-based programming language, and explore more advanced features of the micro:bit.

Accompanying lesson plans and a digital version of this book can be downloaded from https://school.arm.com

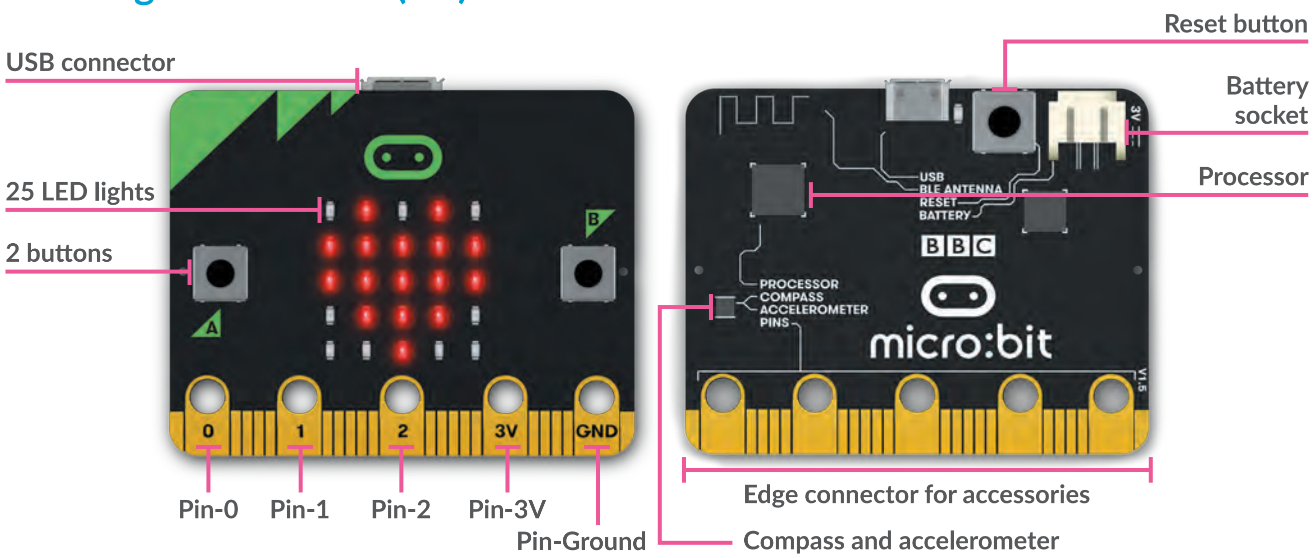
Understanding the micro:bit

Learners often struggle to conceptualize what the micro:bit is and what it is for. It is important to ensure learners understand that a micro:bit can sense, control and react to the real world in many ways. Learners’ creations don’t need to be complex and the making part of the process is as much fun as the coding part. Some projects will only have a sensor and an output (like a traffic counter), while others will have more complex programs (like a remote-controlled robot). The beauty of the micro:bit is its versatility and simplicity. The key to unlocking the power of the micro:bit is understanding the blocks (code) that micro:bit uses and learning how to combine the blocks to solve problems in creative ways. Much can be learned through play and experimentation, but learners progress faster if there is a challenge to overcome or a problem to be solved.

The hardware—Version 1 and Version 2

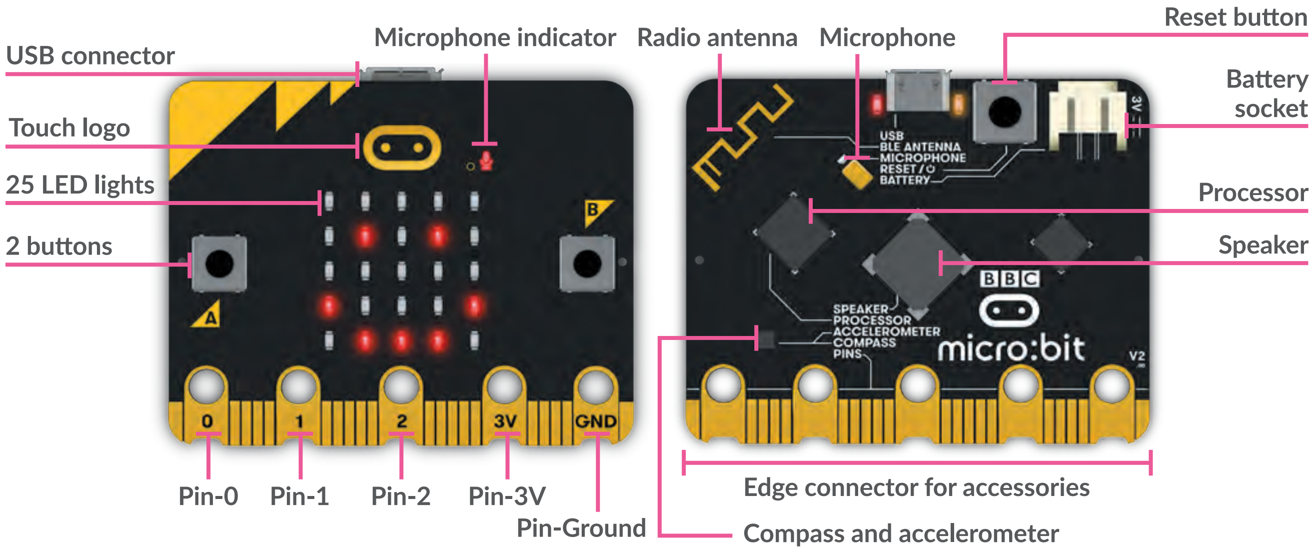
There are now two versions of the micro:bit, the original micro:bit (V1) and the new micro:bit (V2). They look very similar, but the new micro:bit has some new features and hardware that make it even better and allow you to do more with it. You can find all the technical details on the micro:bit website: https://microbit.org/get-started/user-guide/overview/

The original micro:bit (V1)

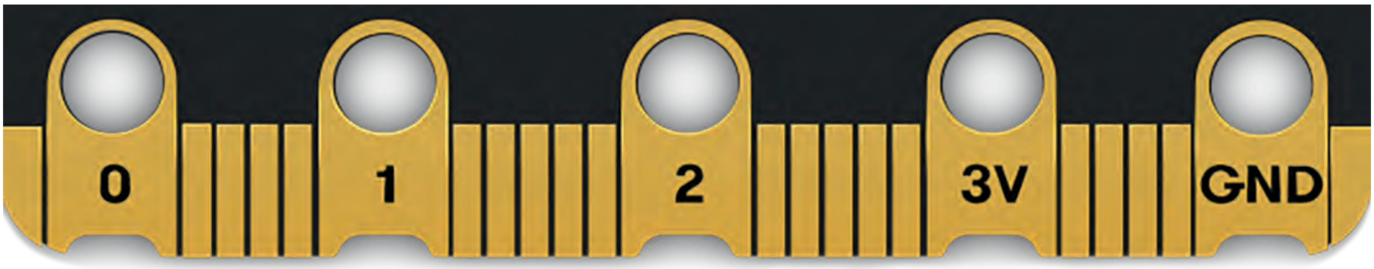


The new micro:bit (V2)

The new micro:bit is packed with new features:



The most obvious way to tell which micro:bit version you have is by looking at the pins. The new micro:bit has notches that help keep crocodile clips attached and prevent them touching the other pins by accident when moved.



The most interesting new features are the built-in speaker and microphone. These remove the need for headphones when playing with the music blocks and allow the use of sound input as well as sound output. There is also an additional button in the form of a touch-sensitive logo on the front. These new features also come with some new blocks in MakeCode.

The interface

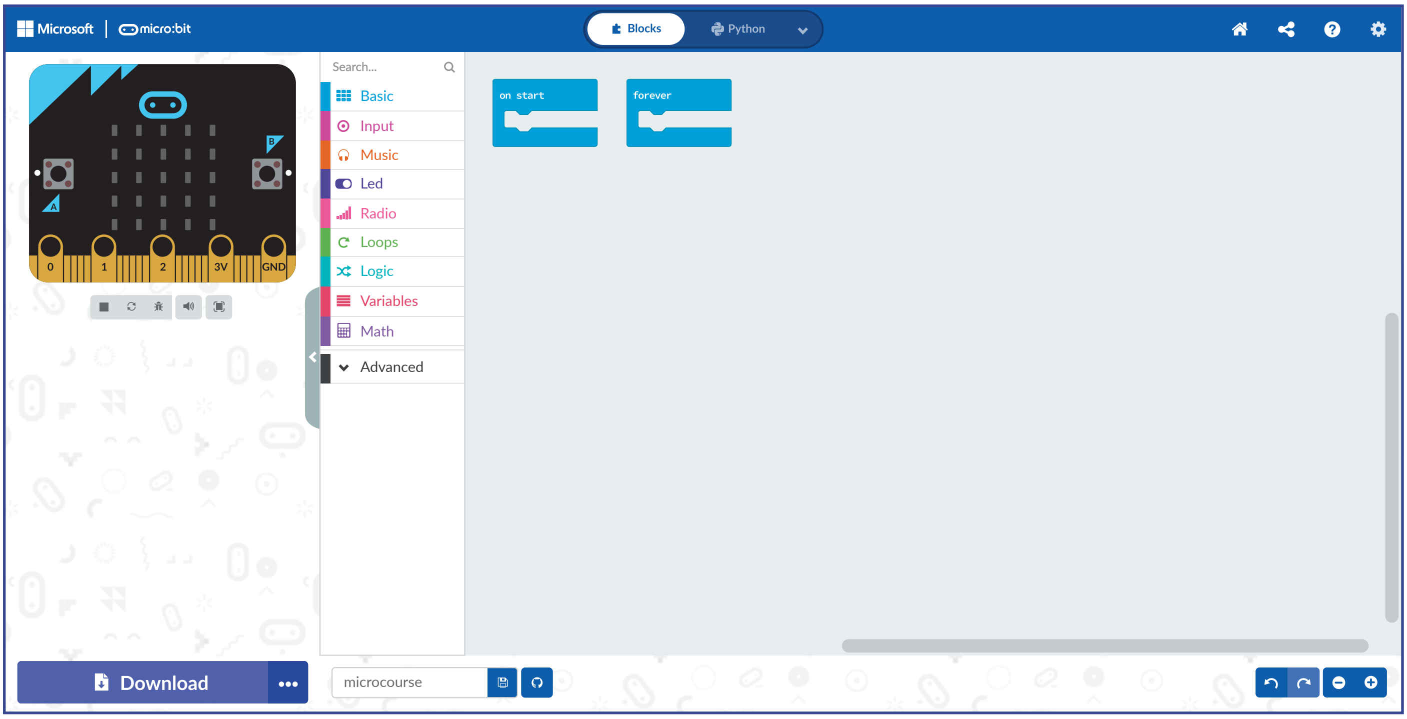
The MakeCode website lets you program your micro:bit using a block-based programming language. These blocks do everything a "proper" programming language does, but they allow you to program visually without worrying about the syntax. You can also program your micro:bit using JavaScript or Python if you want to.

Some blocks, such as the microphone blocks, will only work on the new (V2) micro:bit, but these are clearly labelled. All other blocks will work on both versions.

You can also add additional packages that give you more blocks to play with—more on this later.

PRO *TIP*

If your V1 micro:bit is very old, you may need to flash it as it could need new firmware to work correctly. Detailed instructions are available here: https://microbit.org/get-started/user-guide/firmware/



Simulator and physical micro:bit

The MakeCode website has a built-in simulator that means you don't actually need a physical micro:bit to learn how to use them. This is great for learners, as they can rapidly prototype their ideas and bugfix their code quickly. It removes the need to download the .hex files constantly and flash the micro:bit to see how the changes in code make the micro:bit behave.

Solutions

The .hex and .py solutions to all the projects found in this book are available in our GitHub repository: github.com/arm-university.

QUICKSTART

micro:
project

Setting the scene

In this introductory project we will learn about the basic functionality of the micro:bit and how to program it.

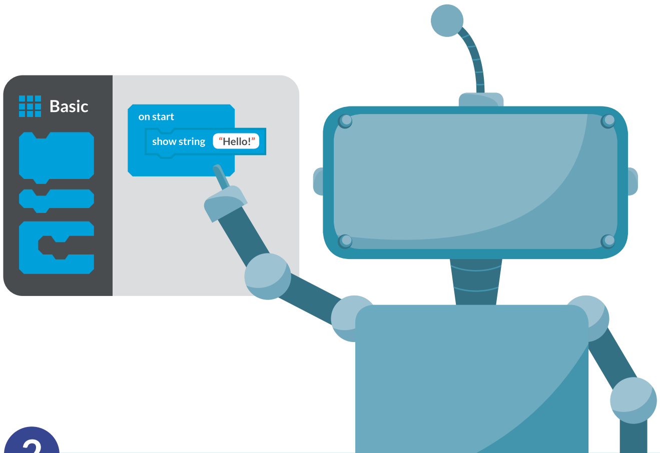
Success criteria

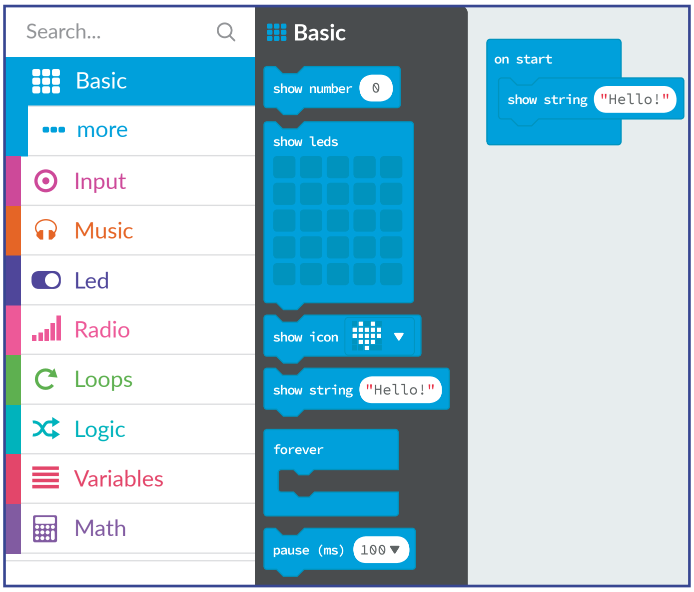
- ☑ Create a simple program using MakeCode
- ☑ Download the program and upload it to the micro:bit

1

Log into your computer and go to https://makecode.microbit.org

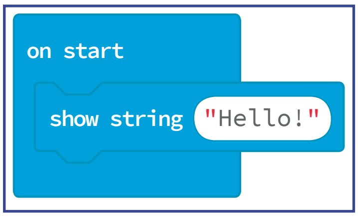
Click the **Basic** tools and drag across the ***on start*** and the ***show string*** blocks as below



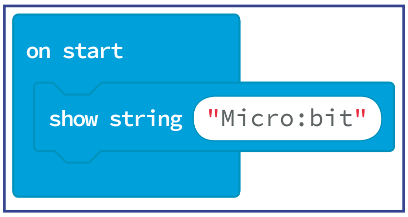


2

Change the text from **Hello!** to something else (maybe your name)



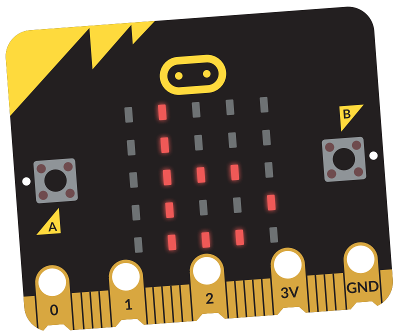
to



3

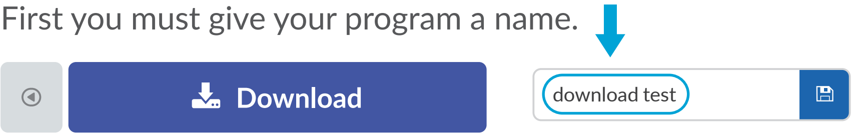
Congratulations!

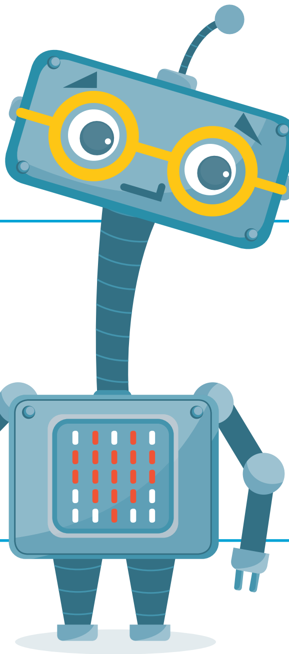
You have created your first program for the micro:bit. Your program will be “simulated” for you so you can see what the output will be:



4

First you must give your program a name.





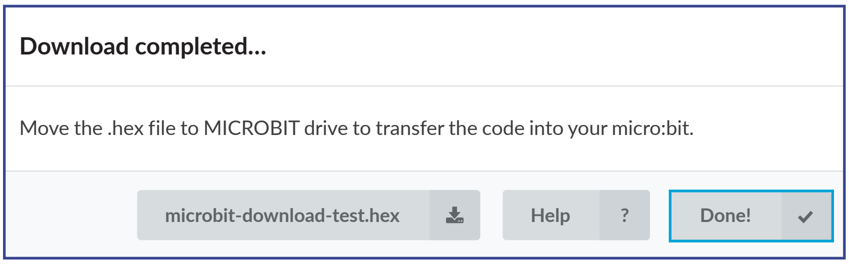
5

You now need to put your program on your micro:bit.
To do this we must Download the **.hex** file and copy/paste it into the micro:bit.

Click the **Download** button

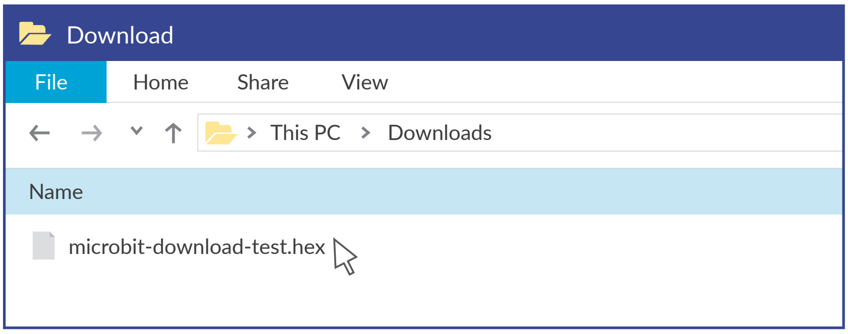
6

You will see this dialogue:



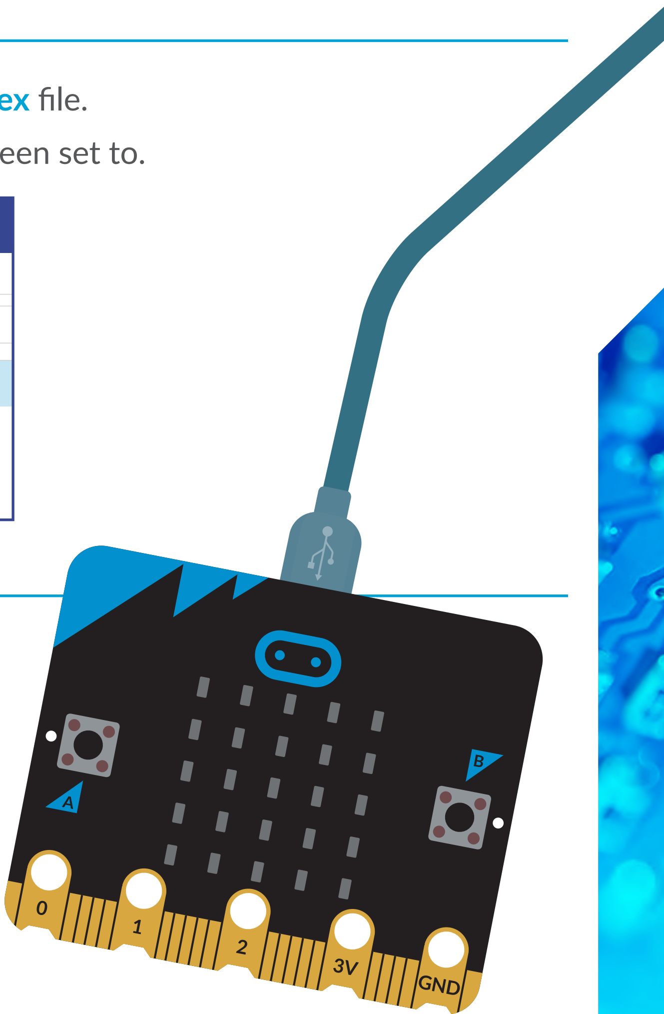
7

If you look in your **Download** folder you should find your **.hex** file.
If not, it will be whatever the default download folder has been set to.



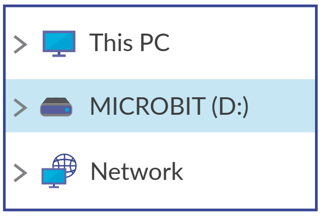
8

If you haven't already done so you will need to connect your micro:bit to your computer using a micro USB cable.

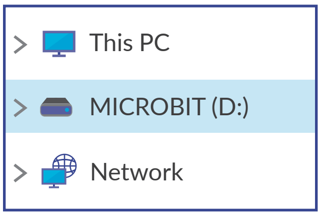


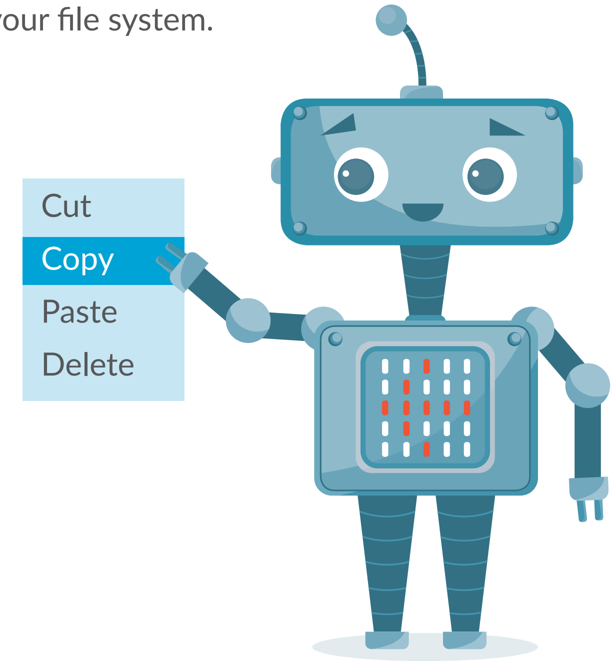
9

Your micro:bit should appear as a removable drive in your file system.



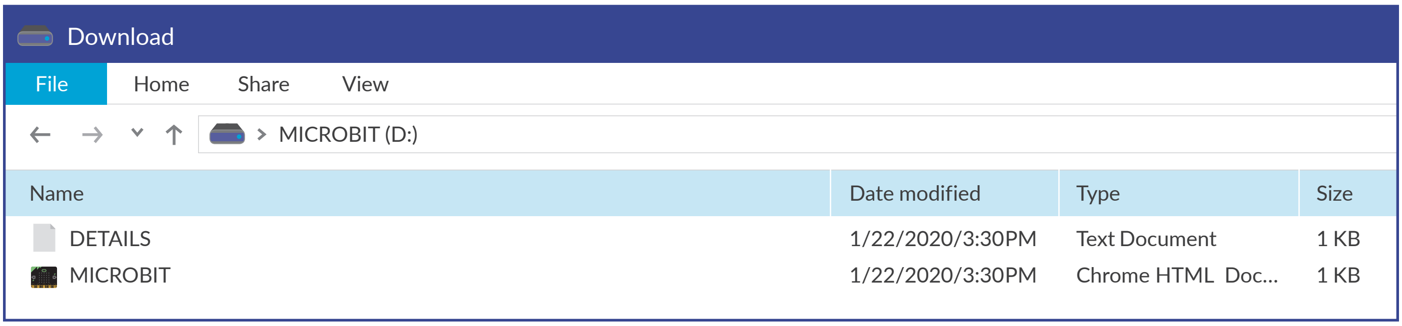
You now need to **Copy** (Ctrl + C) the **.hex** file from your **Download** folder into the micro:bit.





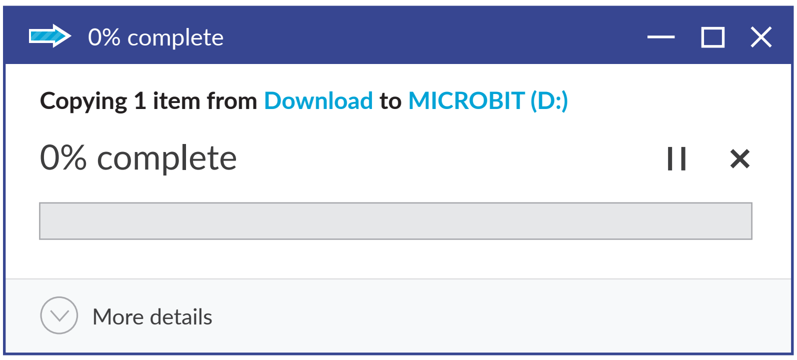
10

You can drag and drop or copy and paste the **.hex** file into the MICROBIT folder. There are some files already on the micro:bit but don't worry about these.



11

When you drag the **.hex** file into the micro:bit you will see a progress window. The orange lights on the back of the micro:bit will flash rapidly while uploading is in progress.



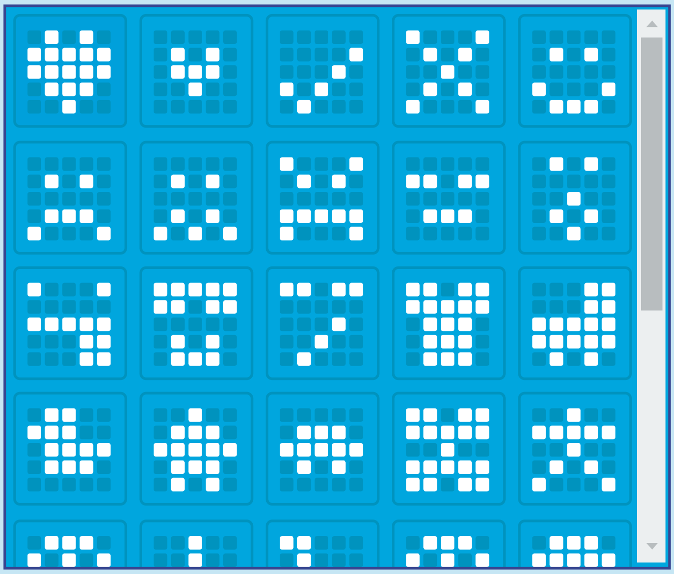
Once the file has transferred the micro:bit window will close and your program will run on the micro:bit.

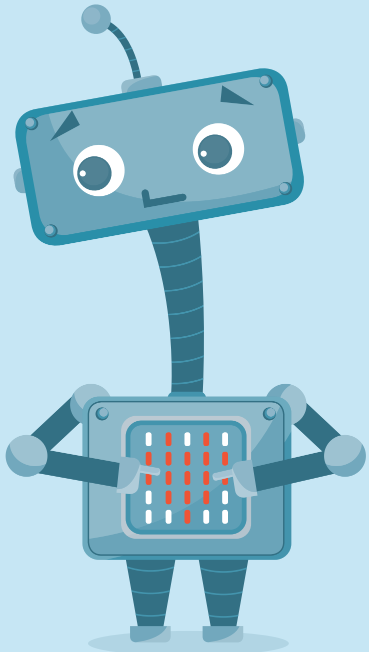
PRO *TIP*

You can sync your micro:bit with MakeCode using WebUSB. This means you can just click "Download" to get your code on your micro:bit. You should get a prompt in MakeCode that will guide you through the process.
https://microbit.org/get-started/user-guide/web-usb/

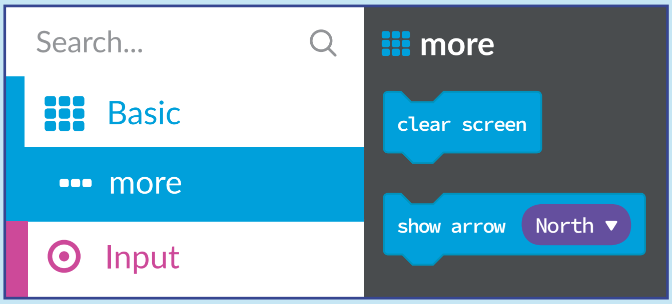
Stretch tasks

- Why did the text only appear once? Make the text appear ***forever***.
- Make a program that displays one of the icons.





- Make your own icon using the ***show leds*** block.
- Make a program that counts down from 5 seconds to 1 seconds and then displays a smiley face for 5 seconds.
- Click the **Input** menu under the Basic blocks menu and then the **...more** icon for more blocks (most of the blocks have more options).



Final thoughts

You have just learned how to:

- Create basic programs using MakeCode
- Name and save your program
- Upload your program to your micro:bit
- Extend your program
- Access more blocks

These are the essential skills for using a micro:bit. You will be doing this a lot in this course!

MULTIPLICATION REVISION APP

micro:project

Setting the scene

In this project we will make a program that gives simple maths problems and also the answers when needed to help young students practice their times tables.

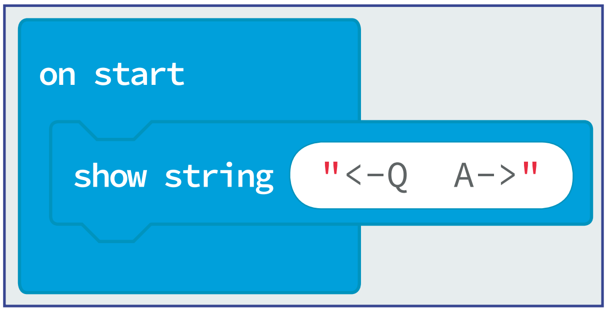
Success criteria

- Has a simple interface when the program starts
- Pressing a button gives a random multiplication problem using numbers between 1 and 10
- Pressing another button gives the answer

1

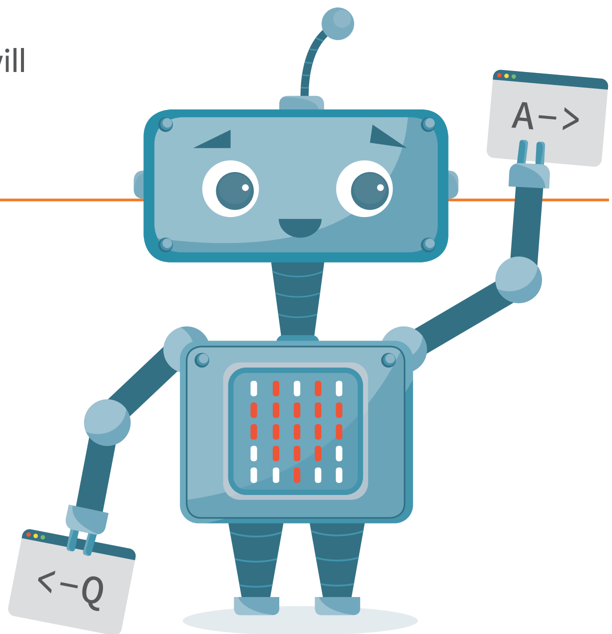
Open https://makecode.microbit.org

The first thing to consider here is how the program will be used. The success criteria requires a simple interface so we need to have some very simple instructions when the program first runs. As the A and B buttons are either side of the LEDs we will program button A to give the question (Q) and button B give the answer (A) and show this with a simple arrow:



2

We now need to program button A. The success criteria requires the numbers to be generated randomly so we will need to use a variable.



3

Variables

A variable is a container for a value, like a number we might use in a sum, or a string (text) that we might use as part of a sentence. One special thing about variables is that their contained values can change.

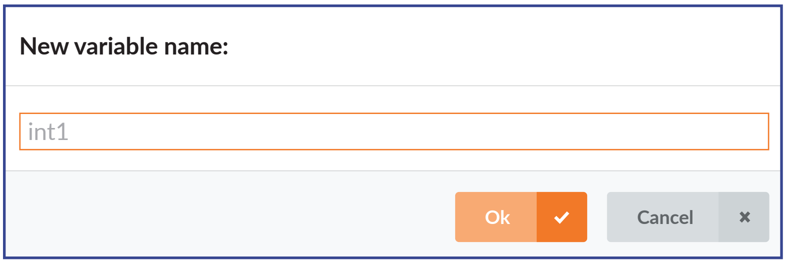
4

We need to create two variables to store the two numbers needed for a multiplication problem:

To create a variable you need to open the '**Variables**' menu

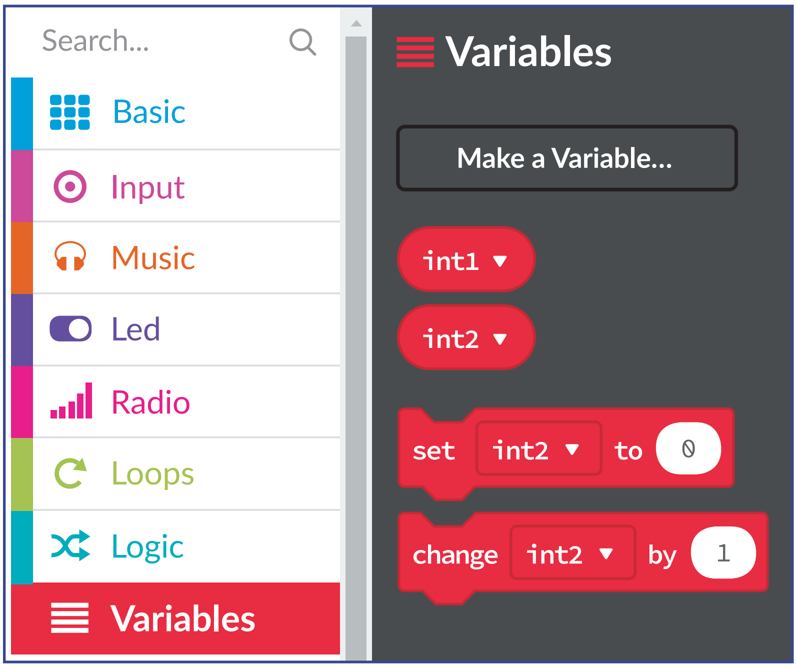
in MakeCode and select "**Make a Variable**"

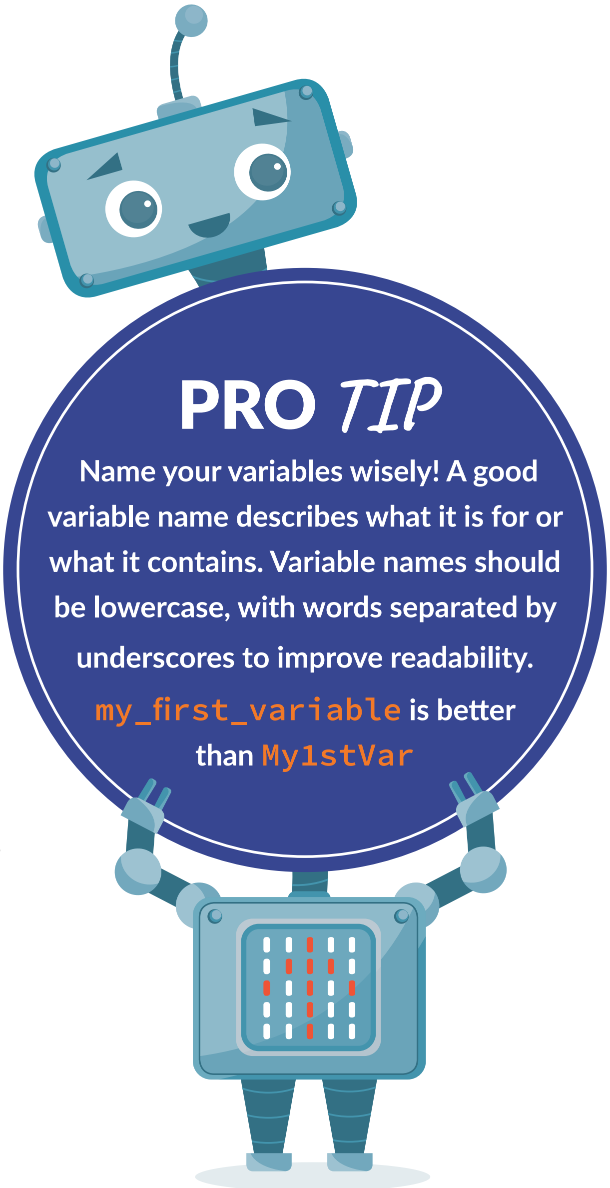
this will open a menu to name the variable:



In this example we have named the variables ***int1*** and ***int2***. Int is short for **integer** which is a computing/math word for whole number.

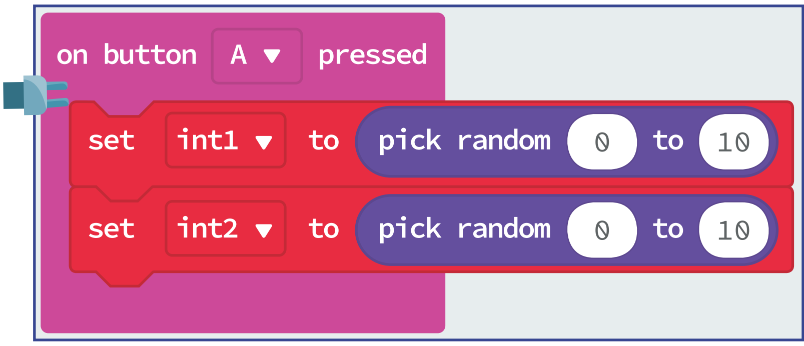
Once you have created the two variables they will appear in the **Variables** menu to be used within other blocks.

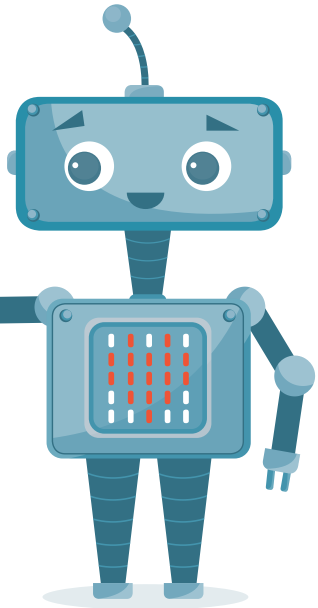




5

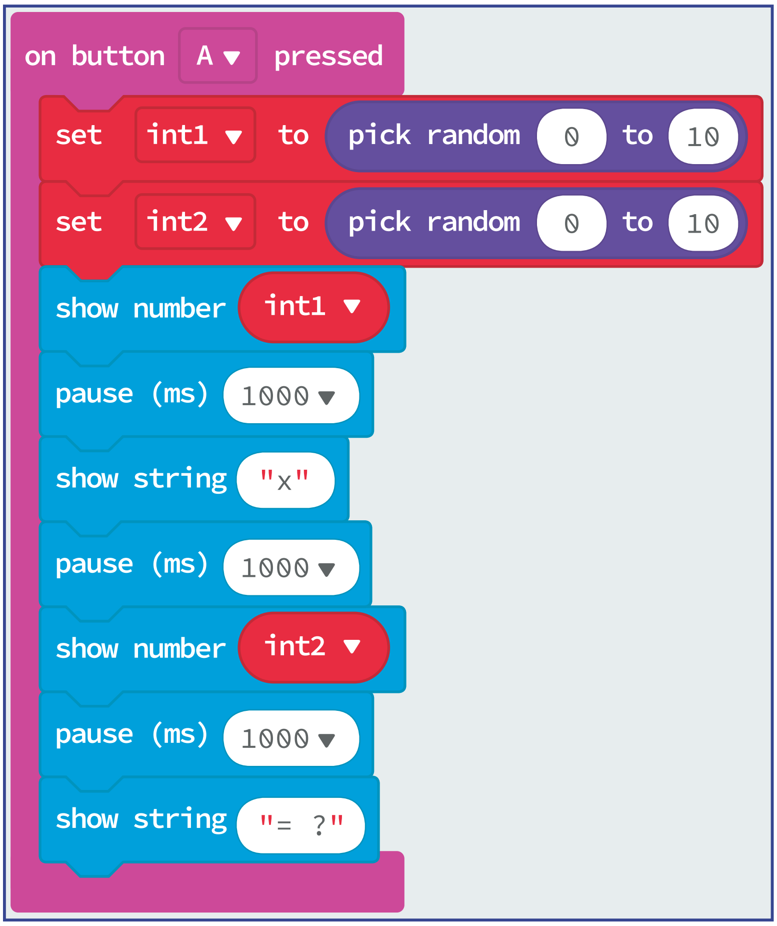
Now you need to use an ***on button A pressed*** block and inside that put a ***set to*** block for both ***int1*** and ***int2*** and then attach a ***pick random number*** from the **Math** blocks. Now change the number from 4 to 10 in the ***pick random*** block.





6

This will now set the two variables to random numbers from 0 to 10 when button A is pressed. We now need to show this on the LEDs so the user can see the question.



Now we can add the ***show number*** blocks to show the random numbers.

To make the program suitable for younger learners we have added a ***pause*** block so the number stays on the LEDs long enough to read.

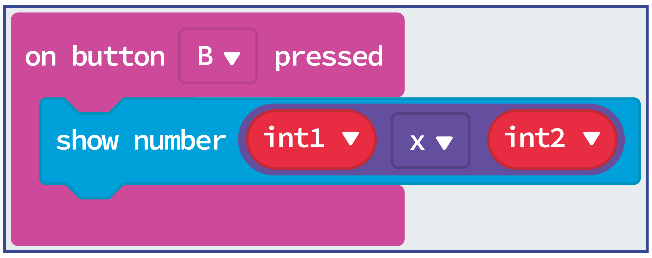
The ***show string "x"*** block represents the multiplication symbol.

That last block is another ***show string*** that displays the "=" sign and then a "?" to show that it is a question.

7 The process so far:

- User is prompted to press button A for a question

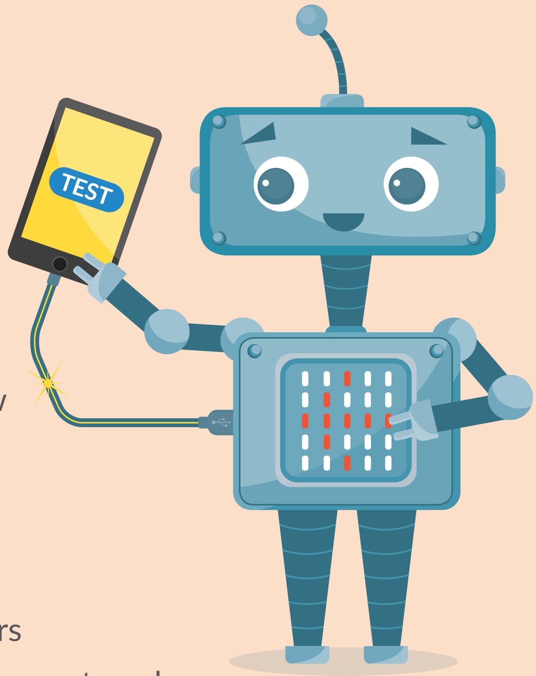
We now need to display the answer as this is also required by the success criteria. We will now program button B to give the answer. As we have stored the randomly generated numbers in variables when button A is pressed we can now multiply them together and show this on the LEDs.



Again we use an ***on button pressed*** block but this time for button B and we also add the ***show number*** block. Inside that block we use another **Math** block to multiply ***int1*** with ***int2***.

Test time!

We have used a few different blocks here so now is the time to test your program and make sure that it behaves as you would expect.



Give your program a name such as **math_app.hex** and **Download** it onto you micro:bit.

Make sure to check that the answer is correct a few times! If not, look at the blocks and see where the problem is and fix it as you go along.

Stretch tasks

- Make the program using the other math operators (+, -, /); you could make the program change the operator when shaken for example.
- Make another micro:bit keep score for two players (you will need to use the Radio blocks).
- Add a hard mode (questions between 0 and 100) when A+B are pressed.

Final thoughts

In this micro:project we have covered:

- Inputs
- Variables
- Random numbers
- Basic math
- Testing and troubleshooting

and combined them to make a useful app. This is what Computing is all about, using the tools to make something useful.

TEMPERATURE SENSOR

micro: project

Setting the scene

In this project, we will explore some of the micro:bit's other sensors and use some new blocks in MakeCode to create a temperature sensor.

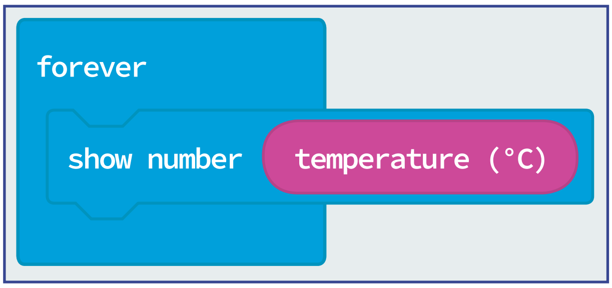
Success criteria

We are going to make a program that senses the temperature and ***if***

- ☑ the temperature is less than 18°C it will show the message "Too cold!!"
- ☑ the temperature is between 18°C and 24°C it will show the mesasge "Just right!"
- ☑ the temperature is more than 24°C it will show the message "Too hot!"

1

Open https://makecode.microbit.org



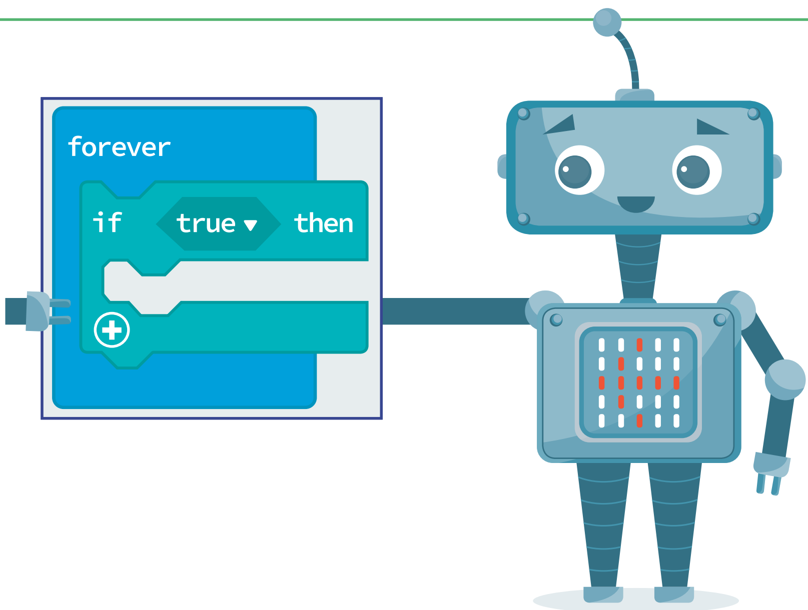
From the basic blocks add a ***forever*** block with a ***show number*** block that contains the ***temperature*** input.

Save this as **TemperatureSensor.hex** and upload it to your micro:bit. You will now see the temperature in degrees centigrade shown continuously across the LEDs. Try moving the micro:bit to warmer and cooler areas to see the changes on the screen (if you have a battery, long enough USB cable or are using a laptop).

We will now use some simple logic to display a message based on the current temperature.

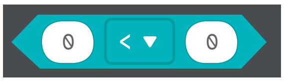
2

To start with the first task, we need to add a forever block to hold the other block so the program will run continously. We then need to add an ***if*** block from the **Logic** blocks.



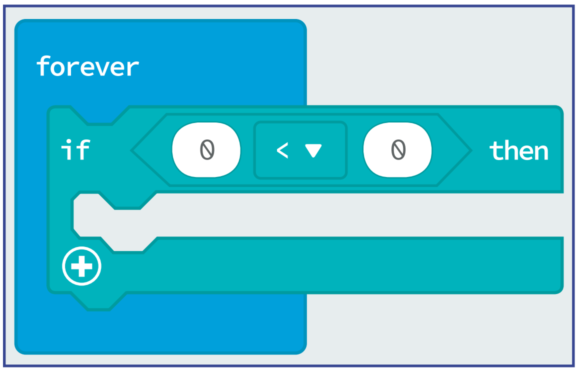
3

This will test if something happens and then will do something that we choose. We want to sense *if* the *temperature* (in °C) is *less than 18* so we now need a compare block (also in the Logic block).



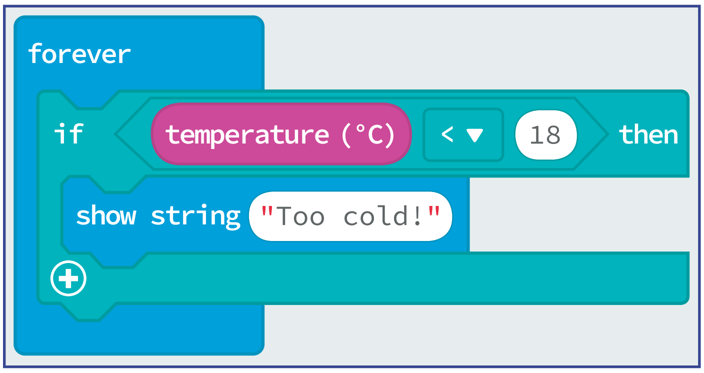
4

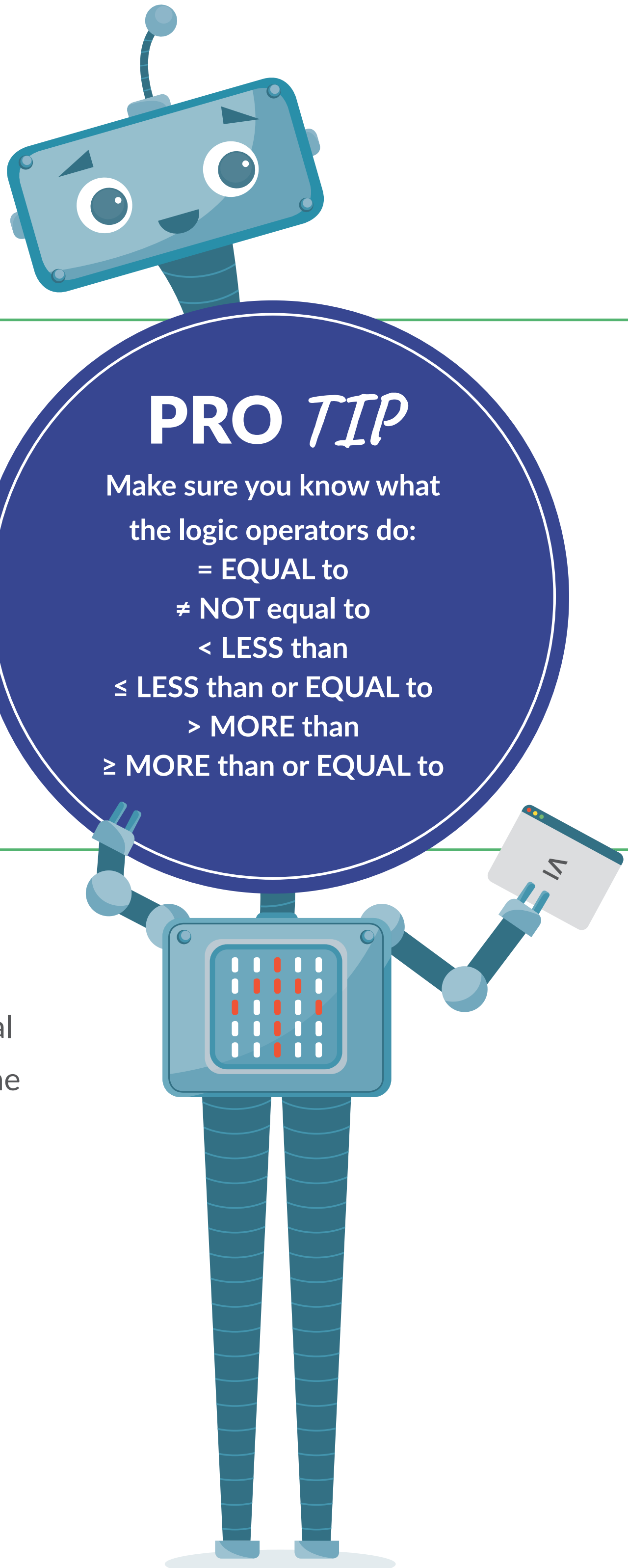
We need to add this to the *if* block



5

We now need to add in the *temperature* input, the desired temperature (18°C) and a message *if* the temperature is *less than 18*°C. The *if* block is a logical test and will return a True or False. It will only run the blocks in the *then* section *if* the test is True.





PRO TIP

Make sure you know what the logic operators do:

= EQUAL to

≠ NOT equal to

< LESS than

≤ LESS than or EQUAL to

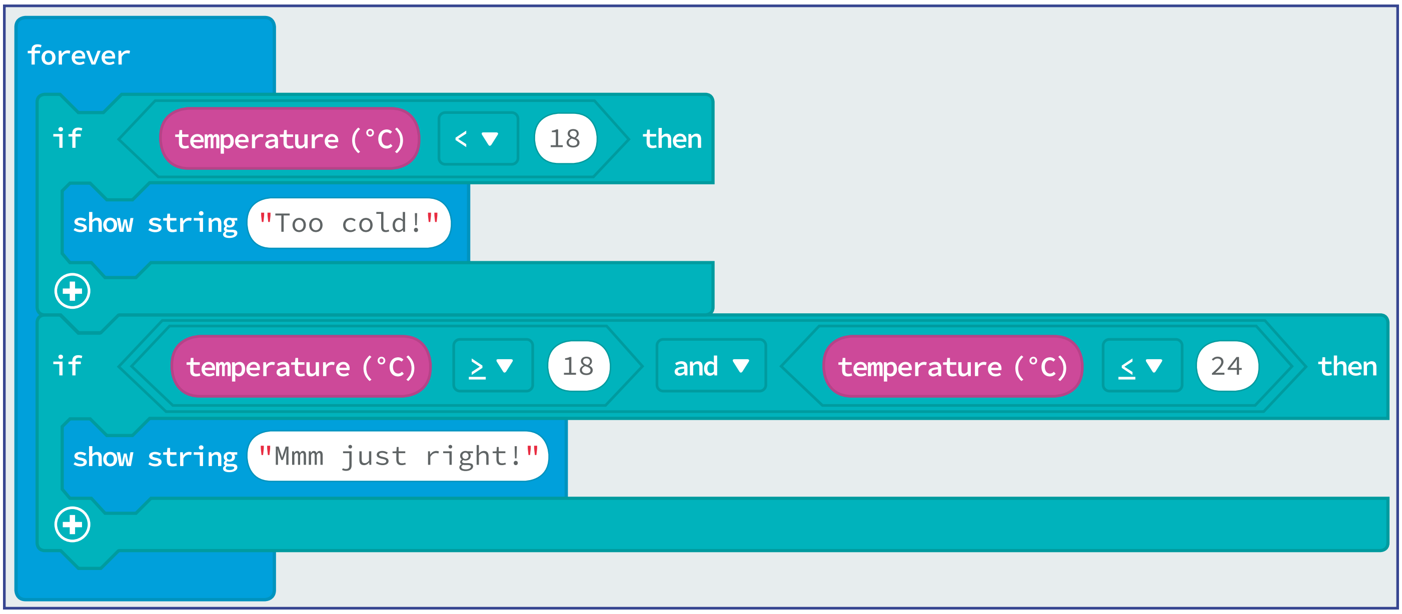
> MORE than

≥ MORE than or EQUAL to

6

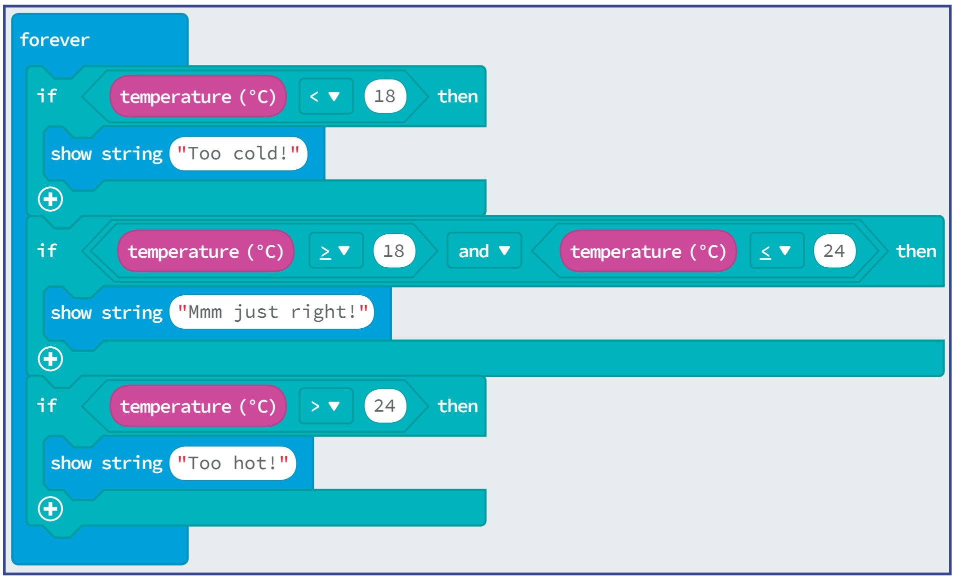
The message “Too cold!” will only show if the temperature is ***less than*** 18°C.

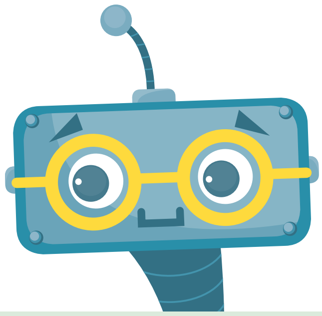
That should now have met the first success criteria. The second success criteria will be slightly more complex as it will need two tests to see if the temperature is above 18°C and below 24°C. To do this we need to use another Logic block with an ***and*** in it to carry out two tests at the same time.



7

The final ***if*** block is very similar to the first block but with a **>24** for the temperature input and “Too hot!” string. Duplicate the first ***if*** block and change the values to speed things up.





Stretch tasks

Here are some challenges that use the micro:bit's inputs:

- Adjust the temperatures to what you think is too cold, just right and too hot.
- Make button A sense the temperature and button B sense the light intensity.
- Swap the ***show string*** for a custom icon.
- Use the ***on pin P0 pressed*** input to generate a random number between 0 and 99.
- Use the ***on button pressed*** blocks to test for ***magnetic force*** (you will need a magnet to trigger this sensor) and use the result to give an output.
- Write down some real world examples of technology that uses if (selection) blocks, logic and sensors to perform a task.

Final thoughts

You have just learned how to:

- Use the micro:bits inputs in a program
- Use computational logic
- Use selection to test an input and return an output

Think about how this program could be extended. If it were too hot, what could we get the program to do instead of just saying it's too hot? What machine could the program turn on?

Research what a **solenoid** is and how it could be used to turn on or off a machine using a program.

MICRO:PET V2

micro: project

Setting the scene

Loneliness and isolation are a real problem for children staying in hospitals for long periods, especially in rural areas. You have been tasked with creating a digital pet that can be played with and can keep people company while they stay in hospital.

Success criteria

The pet must:

- look like a friendly pet (be creative);
- be robust enough to be played with;
- contain a micro:bit that users can interact with;
- have a face to express emotions when interacted with;
- have one or more programmed interactions so it behaves like a pet to keep the user company; and
- use the speaker and microphone to talk and react to touch.

1 Some ideas

Here are some possible ideas that could be programmed for your pet:

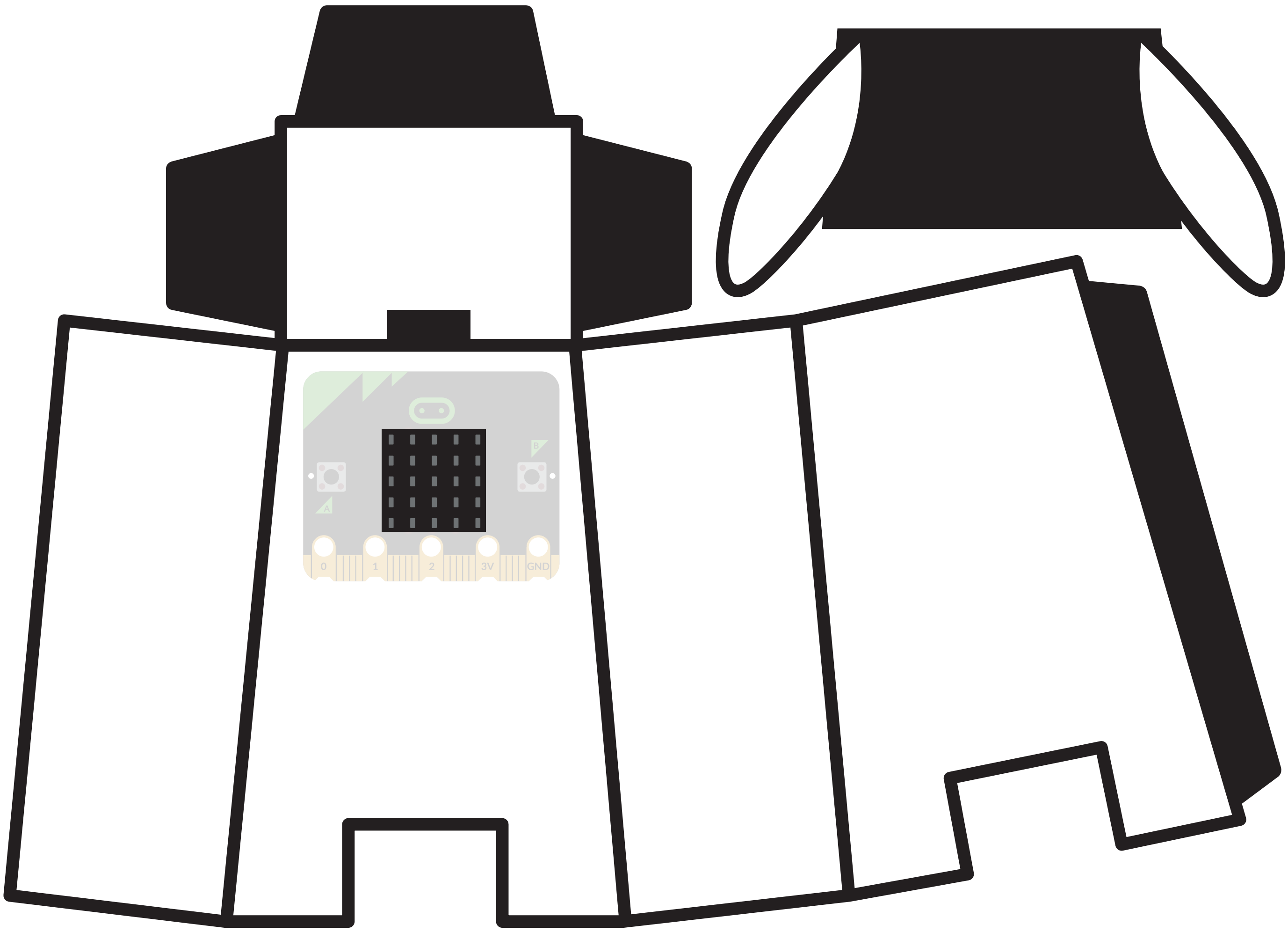
- reacting to playing or shaking using an accelerometer;
- feeding every few second/minutes/hours;
- needing attention, such as getting lonely if not interacted with frequently and making noises to remind you to interact;
- sleeping and waking, depending on a light sensor, and snoring when it is asleep;
- reacting to temperature using a temperature sensor;
- playing mini games, such as rock, paper, scissors;
- communicating and interacting with other pets using radio communication; and
- using other outputs such as sound to make your pet seem like it's really alive (micro:bit V2 only).

2 Design

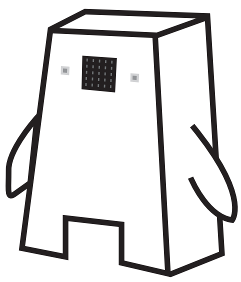
You can go one of two ways: use the net provided as the body of your pet, which you can adapt and decorate, or design your own! If you design your own then you will need to complete the design sheet to justify your design ideas.

The most important thing to remember is to be creative and come up with something novel that meets the success criteria in an interesting way. Think about the needs of the user. Think about how they will interact with the pet, what they would expect a pet to do, and how it would behave.

This is a BLANK net. It would make a minimalist pet on its own, but you are meant to come up with a name and design for your pet. This will give it character and make it seem more alive for the user.



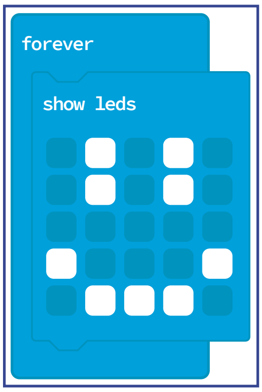
You may also want to design your algorithms. You can do this however you choose, or you could jump straight into MakeCode. When programming your pet, always remember to consider the following.



- How is the user interacting with the pet?
- What are the inputs, processes and outputs?
- Test your pet continuously and adjust and improve it as you go along.
- Keep in mind the success criteria.

Here are some block snippets to help you get started or to help if you get stuck. Try to complete the task without using these snippets if you can.

3 Smiley face for the micro:pet

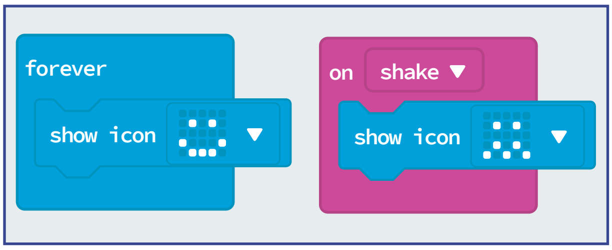


The first success criterion is to make the micro:pet friendly. You can do this by giving it a smiley face. There are some pre-made icons you could use, or you could make your own. Here, the *show leds* block is inside a *forever* block so that the default state is happy.

PRO TIP

Use animation to make your micro:pet more lifelike. Think about how your micro:pet would react, how you can animate this, and what sounds you could use.

4 Reacting to playing or shaking (accelerometer)



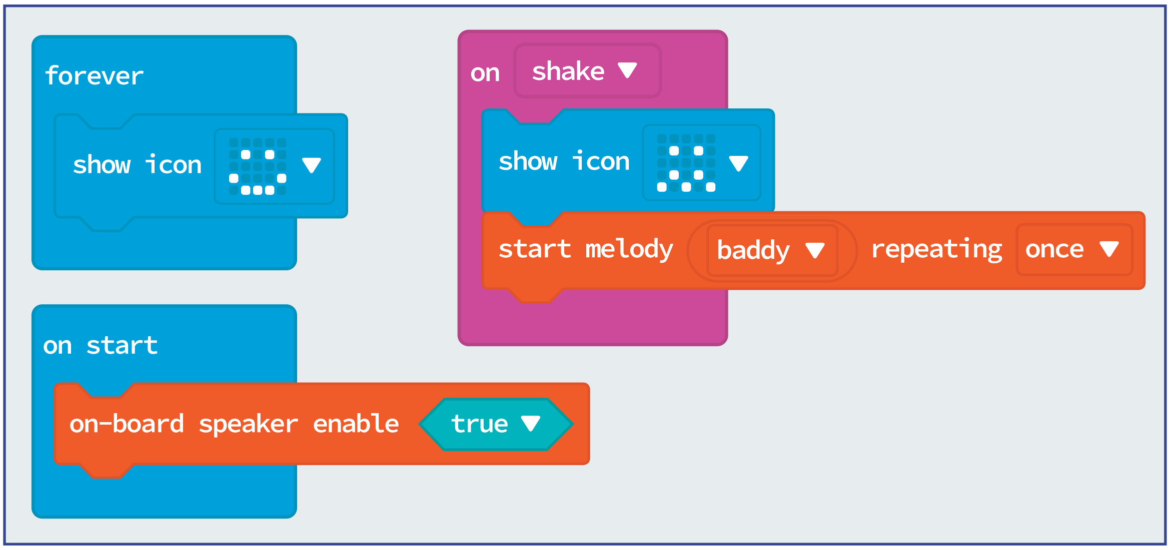
A simple interaction to start with is the micro:pet reacting to being shaken. Here, the micro:bit's accelerometer senses when it is being shaken using the ***on shake*** block. It then shows a sad face to represent the micro:pet being unhappy when it is shaken.

5 Reacting to playing or shaking (accelerometer) (V2)

If you have a micro:bit V2, then you can also add in some sound blocks to accompany your interaction.

The V2 sounds are in the *sound* blocks. You also need to *enable* the speaker *on start* to make it work. You only have to enable it once.

Think about how sounds can be added to all the interactions and what sounds best represent what is happening. Here we use the *baddy* sound as the micro:pet is not happy. Experiment with the sounds available to best suit the interaction.

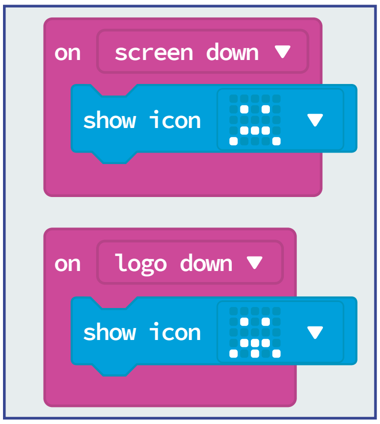


6

Emotions that are affected by interaction

Next, add some other interactions, like the micro:pet being unhappy when it is placed screen down. To do this, the ***input*** blocks use the gyroscopic sensor to find out what orientation the micro:bit has. The micro:bit can be programmed to give outputs such as a sad face.

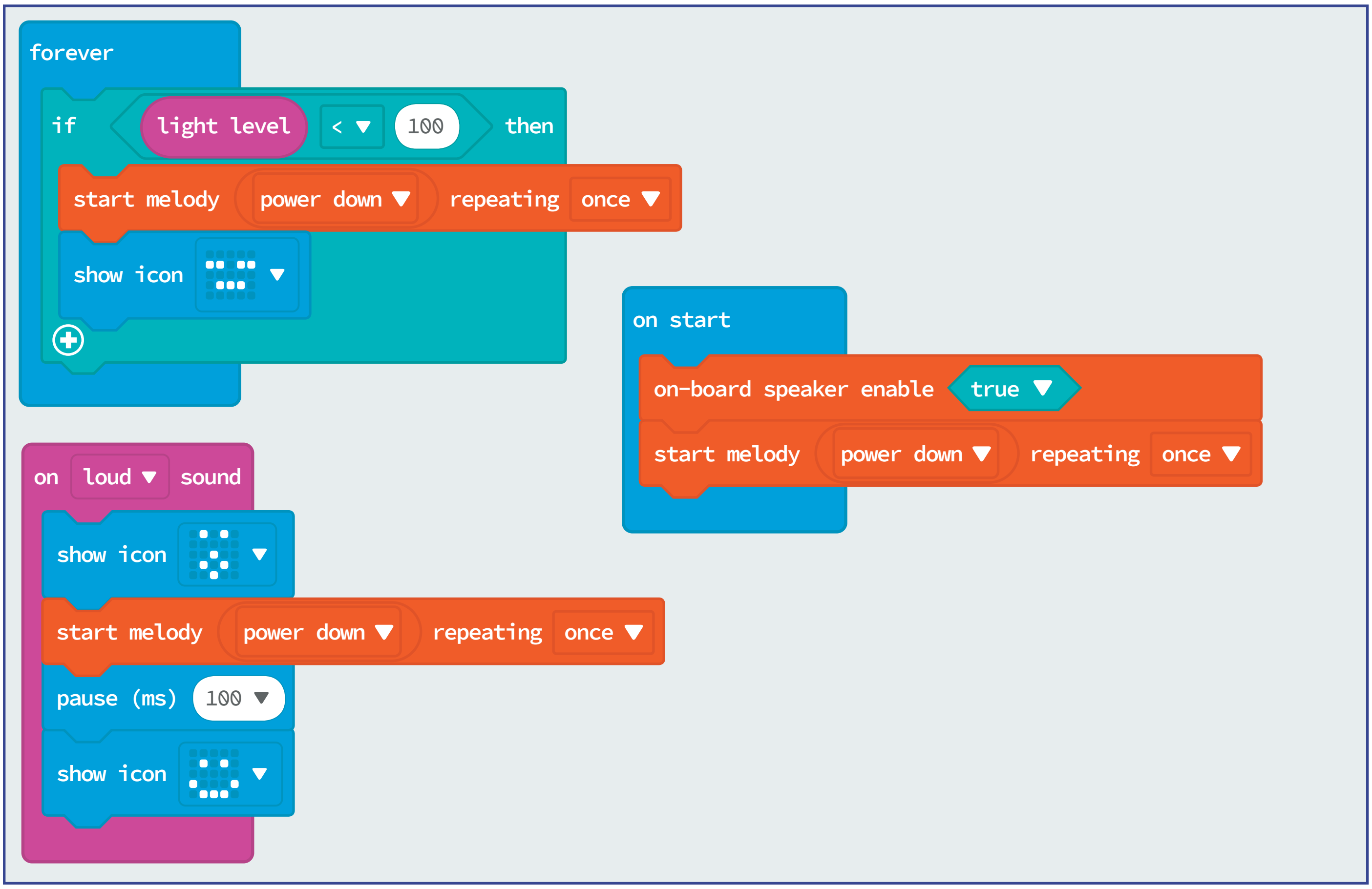
Think about how a micro:pet would react to the available inputs. Then think about how the reaction can be modeled using outputs such as LEDs and sound.



7

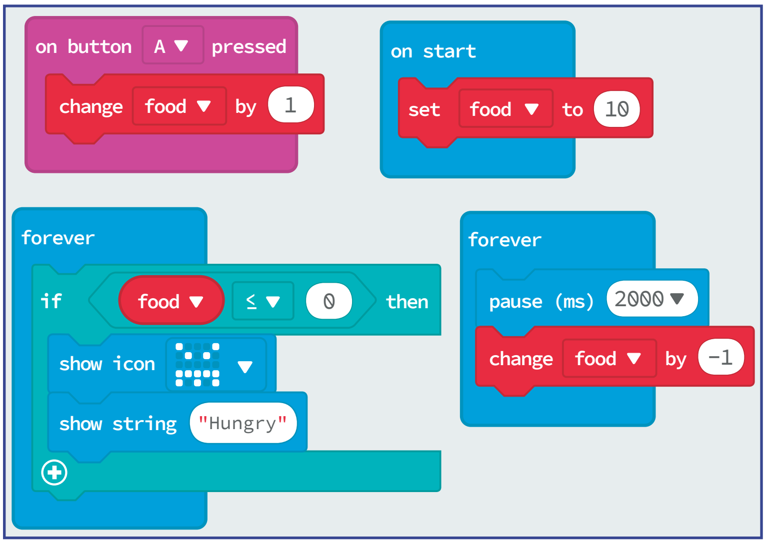
Wakes up when it hears a loud noise (V2)

If you have a micro:bit V2, you can expand these interactions with sounds. Here, we use the microphone on the micro:bit to sense loud noises. We give the micro:pet a surprised face and a sound to represent being shocked by a loud noise.



8 Feeding (every few seconds or minutes)

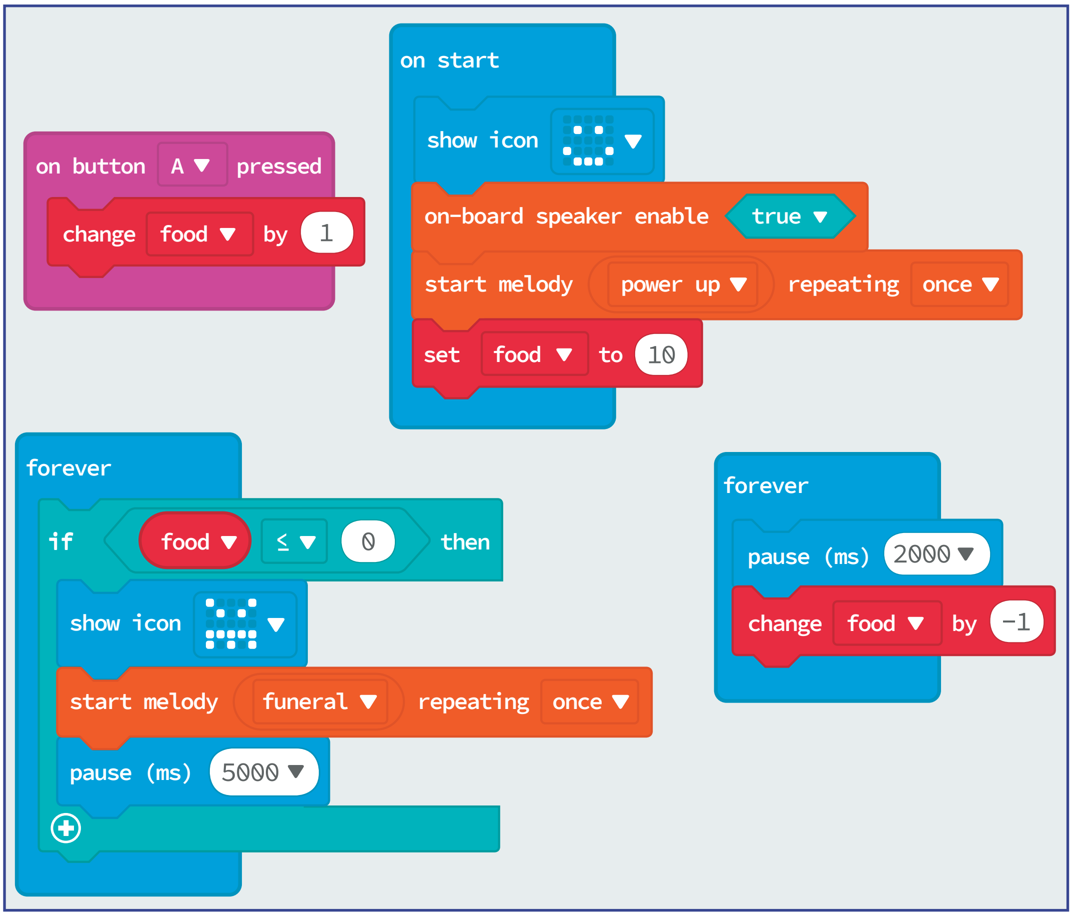
All pets need regular feeding and the micro:pet is no different. Here, we create a variable called *food* and set it to 10 *on start*. The *food* variable goes down by −1 every 2 seconds (you can make this as long as you like). When it gets to 0, the pet gets hungry and lets you know. You can feed the pet by pressing the A button, which increments the *food* variable by 1 on each press.



Think about how the feeding could be made more interactive. Instead of pressing a button, you could use another micro:bit as "food". The *radio* blocks can be used to recognize when the food is near the micro:pet.

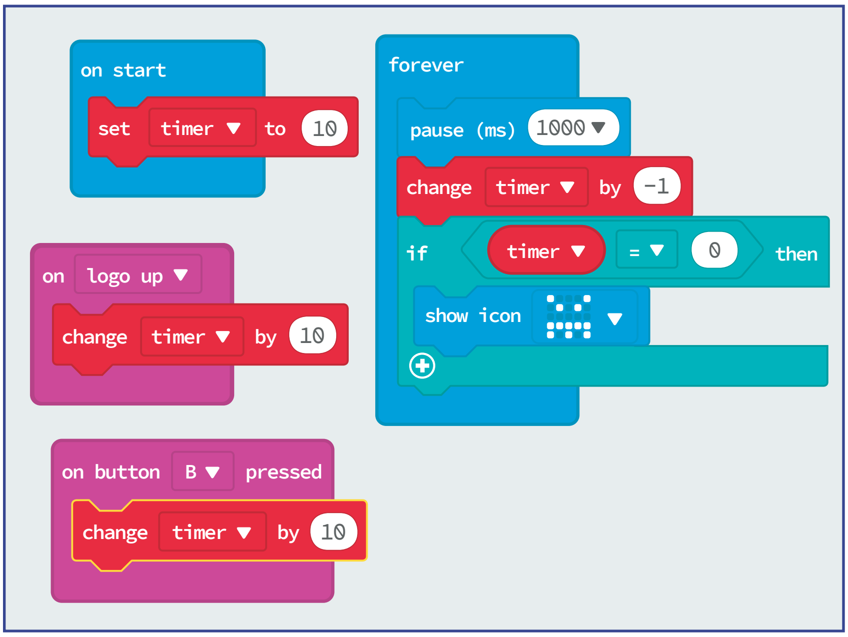
9 Feeding (every few seconds or minutes) (V2)

If you have a micro:bit V2 then you can enhance the feeding interaction with sounds for eating, being hungry, and even being too full.



10 Needing attention (gets lonely if not interacted with frequently)

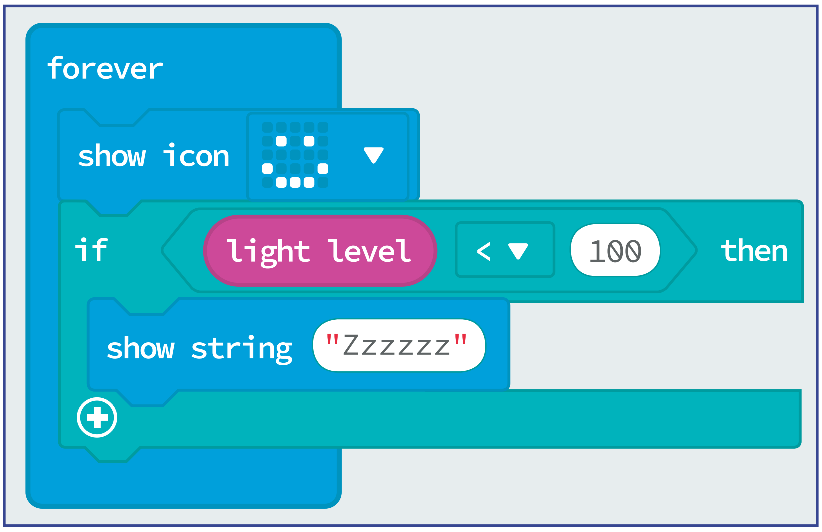
Another interaction to explore is the micro:pet needing attention. This would prompt the user to play with the pet regularly, just like a real pet. Here we use another variable called *timer*. This goes down every second and the micro:pet gets angry when *timer* reaches 0. The *timer* variable represents the attention given to the micro:pet. You can give it attention by pressing the B button or turning the pet upright. Both of these actions increase the *timer* variable by 10.



Getting the "neediness" right will take some experimentation. You don't want the pet to be so needy that it annoys the users.

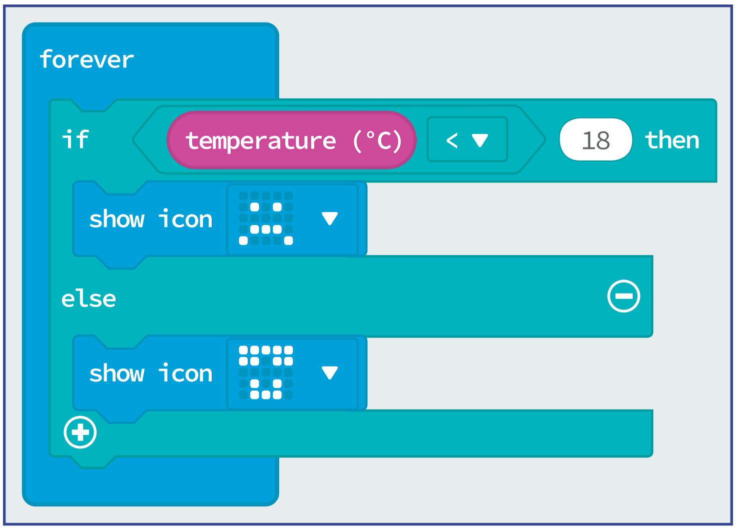
11 Sleeping and waking (light sensor)

Here we also add in an ***if*** block to monitor light levels. If the light level is less than 100 out of 255, then the micro:bit will show "Zzzzzz". This represents the micro:pet going to sleep when it gets dark.



12 Reacting to temperature (temperature sensor)

Similarly, the micro:pet can be programmed to react to the ambient temperature. Here, if the micro:pet gets cold, it shows a sad face.



13 Communication and interaction between micro:pets (advanced)

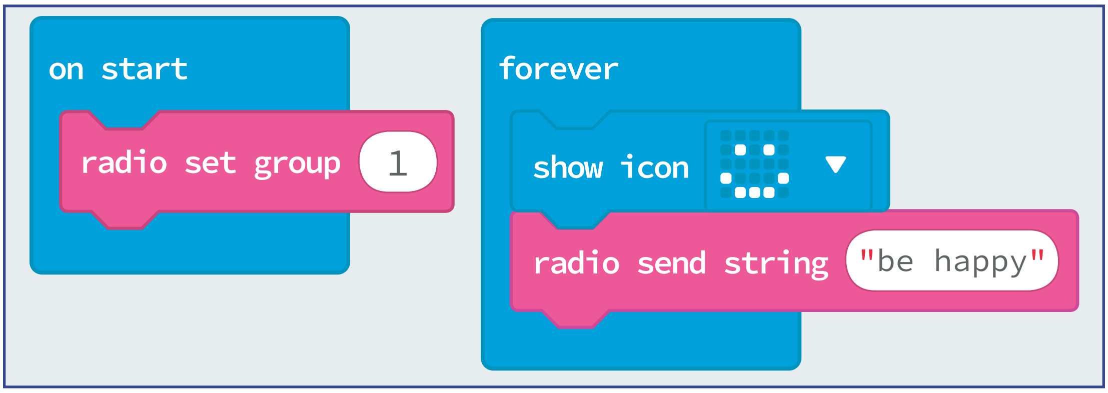
If you have more than one micro:bit, you can create two micro:pets and have them interact with each other! This makes many more types of interaction possible. Here, micro:pet 1 sends a “be happy” string using the *radio* blocks. When micro:pet 2 receives the string, it shows a happy face.

The *radio* blocks can be used to add some advanced features. For example, you could add in games between the micro:pets such as hide and seek, and rock paper scissors, as well as adding in more interactive feeding.

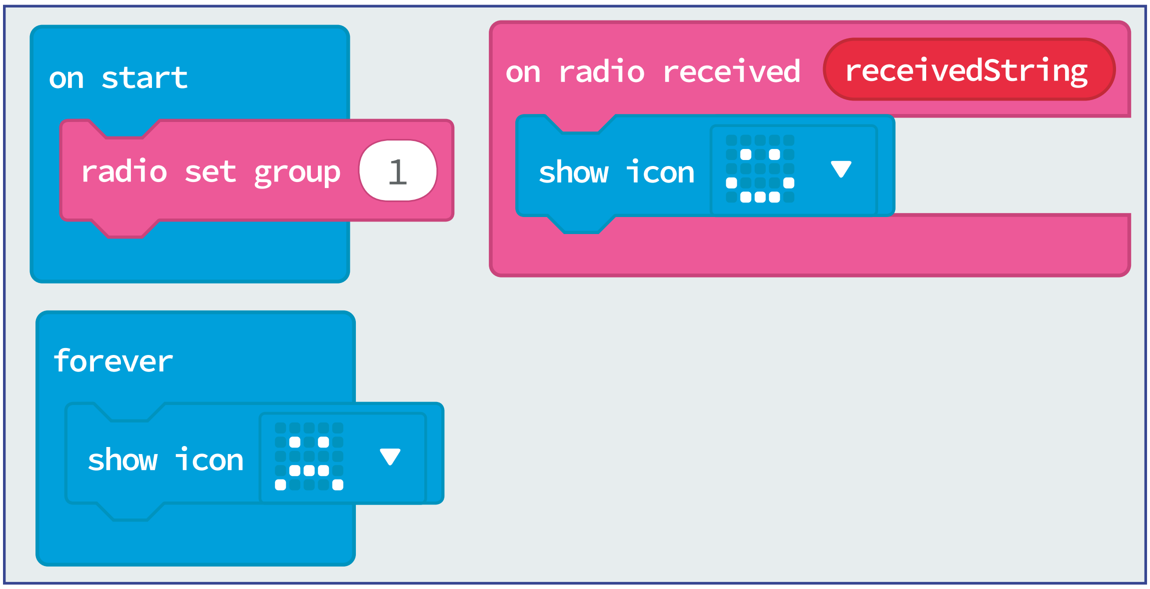
PRO TIP

Both micro:bits need to be on the same radio group to be able to communicate with each other.

Micro:pet 1:



Micro:pet 2:



Stretch tasks

- Use other inputs, such as other types of sensors (this requires additional hardware).
- The medical team at the hospital have asked whether the pet could log any useful data. Investigate what data could be logged to help them.
- Use servo motors to make your pet move.
- Use https://www.plushpal.app/ to create custom gestures for your pet.

Final thoughts

This project introduces the concept and practice of rapidly prototyping a product to meet a need. Mixing together problem solving, making, and computational thinking to make creative and innovative projects is what makes physical computing fun. This project could be taken all the way to a real-life product, and you should think about what other features you could add to better suit the needs of the user. How could it be a better medical device? How could it be better at entertaining a patient? How could it be improved? This idea of iteratively improving products and projects is one we will explore further in other projects.

ROCK PAPER SCISSORS

micro: project

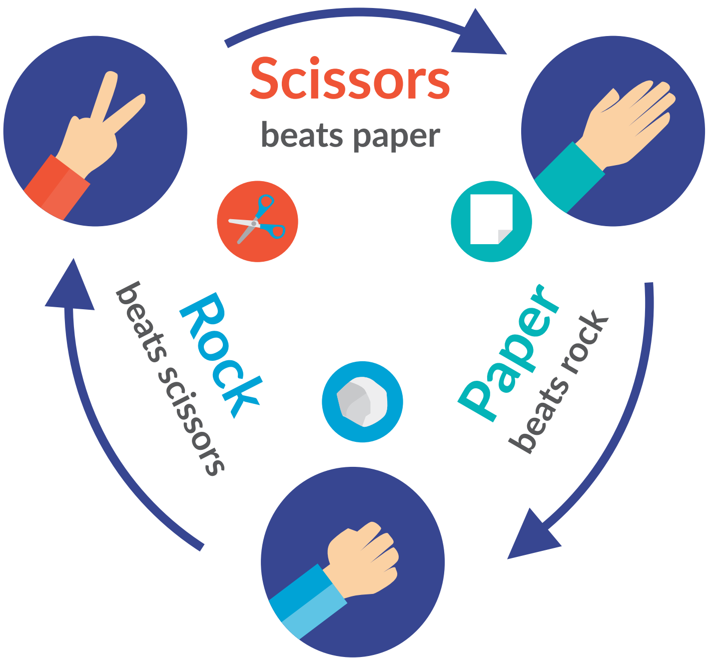
Setting the scene

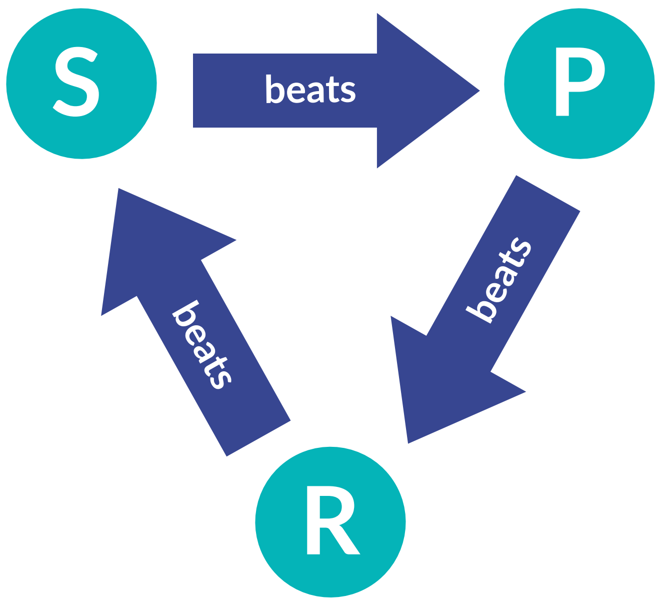
In this project we will explore a more complicated game that involves some logic to determine the winner. The game is the classic rock paper scissors.

For this project, we will make a program on two micro:bits so that you can play rock paper scissors (also known as Roshambo)

Success criteria

- Make a game where S (Scissors) beats P (Paper), P beats R (Rock) and R beats S
- The program will randomly select S, P or R when shaken
- The program will transmit via radio the selection and will determine if it has won or lost the match
- The program will keep count of wins and losses until reset by a button press (A and B)





1

Getting started

To begin, we will solve the second success criteria. We need to create a variable to store the randomly generated S, P or R.

Open https://makecode.microbit.org

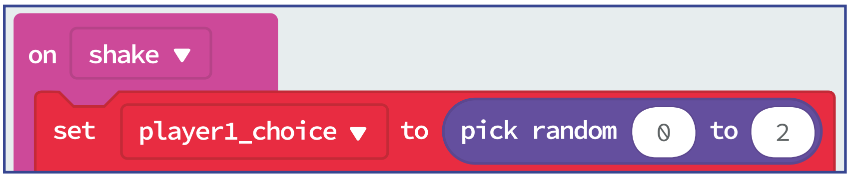
2

Random

We can't get the micro:bit to make a random choice between S, P or R so we need to get it to randomly generate a number (0, 1 and 2) which we can then use to represent the S, P and R. We also can't compare characters (chars) using logic so we must use number for this.

3

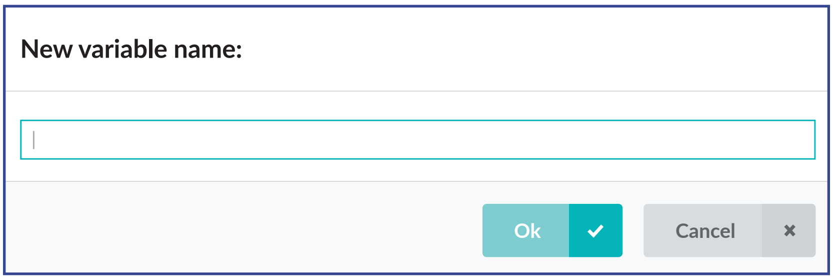
Here you can see we use an ***on shake*** block with a variable called ***player1_choice*** with a **Math** block to create the random number from 0 to 2, we will use 0 as S, 1 as P and 2 as R.



4

To create a variable you need to open the **Variables** menu in MakeCode and select "Make a Variable" button

this will open a menu to name the variable:



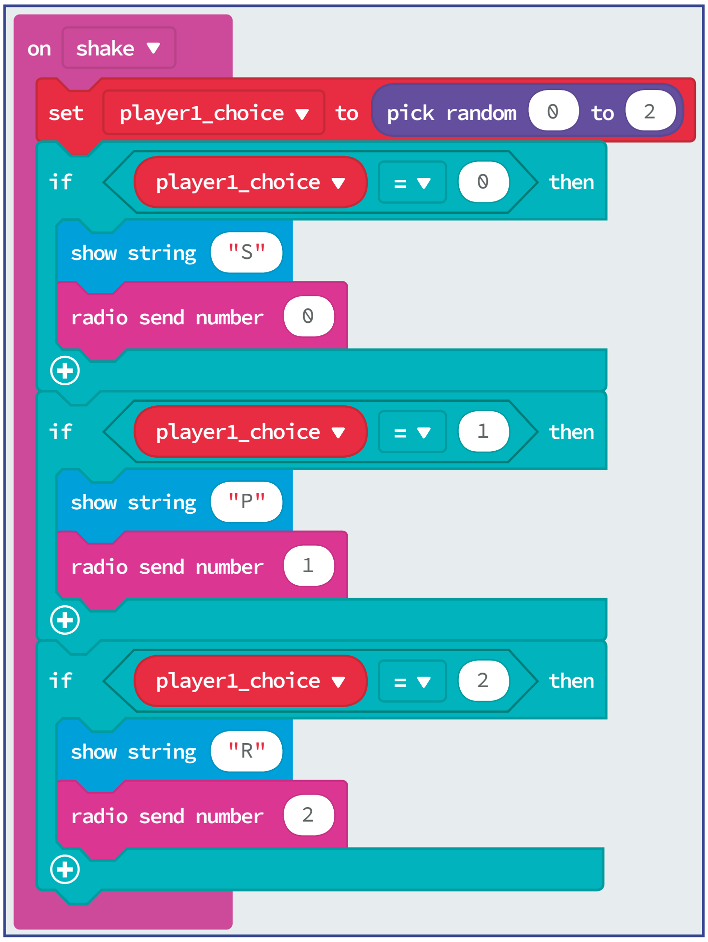
5

We will now use some logic to tell the user (and the other player's micro:bit) what our choice was.

Two players are better than one

For this game to be realistic we will need two players and the code for each will be very similar. Think about how the code for the second player will need to change.

6



To work out who has won we need to create three "tests" with three different outcomes:

- If the random number is **0** then the micro:bit shows **S** and also broadcasts the number **0**
- If the random number is **1** then the micro:bit shows **P** and also broadcasts the number **1**
- If the random number is **0** then the micro:bit shows **R** and also broadcasts the number **2**

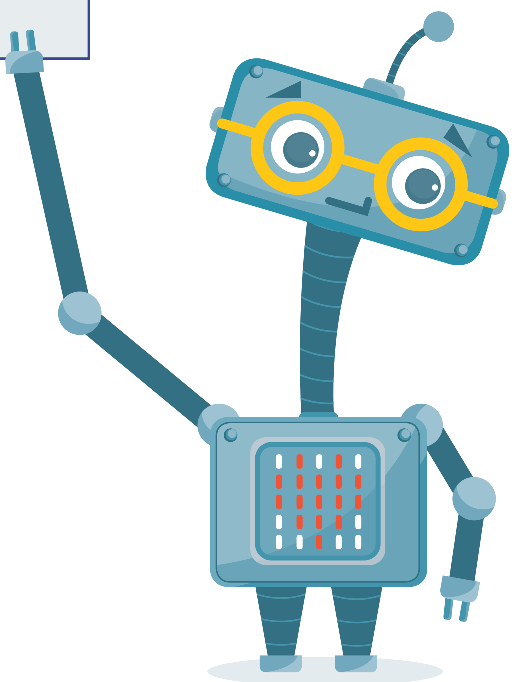
This gives us the basic game where the shake produces either S, P or R. You could play the game just with this program as you can work out the winner yourself but we can make the micro:bits work it out for us!

7

We need to ensure that the two micro:bits are talking to each other on the same channel. To do so we need to set the radio group to the same number on both micro:bits.

So far we have met the following success criteria:

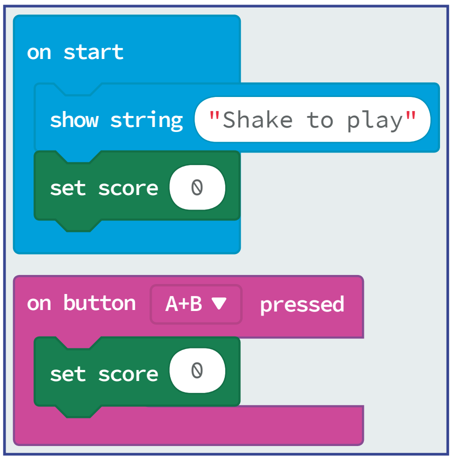
- The program will randomly select S, P or R when shaken
- The program will transmit the selection via radio



8

Keeping score

We will now make the program keep score for each user and also allow the user to reset the score to zero.



Here we create the initial user interface by telling the user to "Shake to play". Then we use the ***set score*** block (in the 'Game' blocks menu) to set the score to 0 when the program first runs and also when **A+B** are pressed.

The score variable

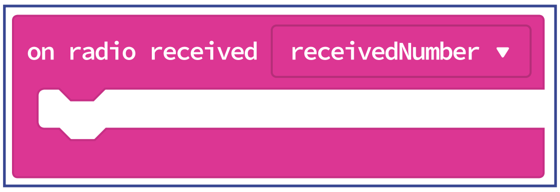
'Score' is a built-in **variable** in the micro:bit used for making games.

9

The game logic

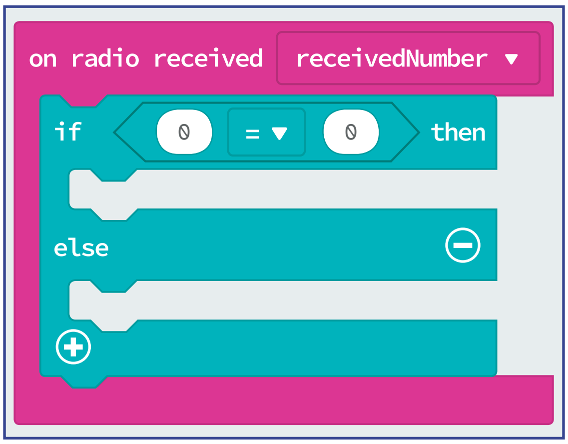
Now for the slightly tricky bit! We have made the program transmit the choice but we now need to make it receive the other player's choice and determine whether we have won, lost, or whether it's a draw.

We are going to do some comparrisons of the variable ***player1_choice*** and ***receivedNumber.*** We have just created the ***player1_choice*** variable and it will contain a number (0-2) if it has been shaken. We now need to make the program listen for the other player's choice and to do this we need the ***on radio received*** block from the Radio blocks.



This block receives the radio signal from the other micro:bit and stores the number in a variable called ***receivedNumber***

We can now compare the number transmitted from the other micro:bit (which represents their randomly selected choice) and compare it to our number (randomly selected choice) to determine who has won.



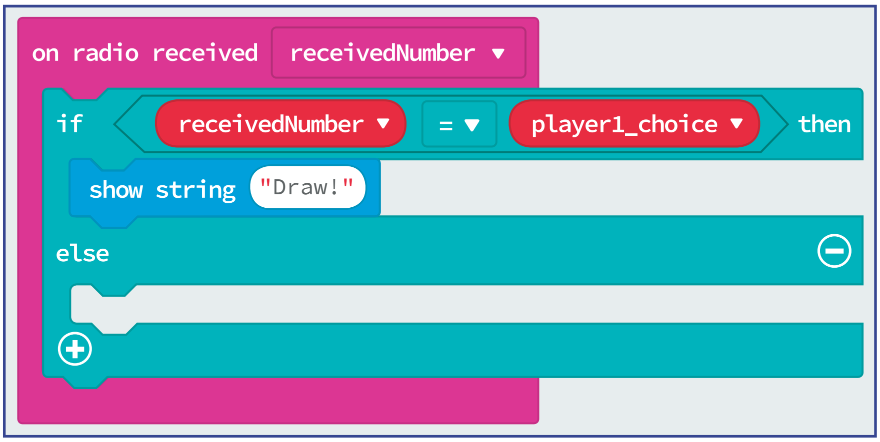
To do this we need an **if then else** block from the logic blocks. Inside that block we will put an = logic block. We are going to compare whether the ***receivedNumber*** is the same as ***player1_choice*** and if it is then the game is a draw.

10

We now need to add an = logic block so we can see if the variable is = to ***receivedNumber***.

Here you can see the = block with the variables inside them.

The **then** part of the **if then else** block shows the string “Draw!” because if both players choose the same the game is a draw.



		Player 1 (player1_choice)		
		S – 0	P – 1	R – 2
Player 2 (receivedNumber)	S – 0	**Draw**	Condition 1	Condition 2
	P – 1	Condition 4	**Draw**	Condition 3
	R – 2	Condition 5	Condition 6	**Draw**

This leaves 6 remaining “conditions” that we need to test for.

Condition	Player 1 (player1_choice)	Player 2 (receivedNumber)
1	Lose	Win
2	Win	Lose
3	Lose	Win
4	Win	Lose
5	Lose	Win
6	Win	Lose

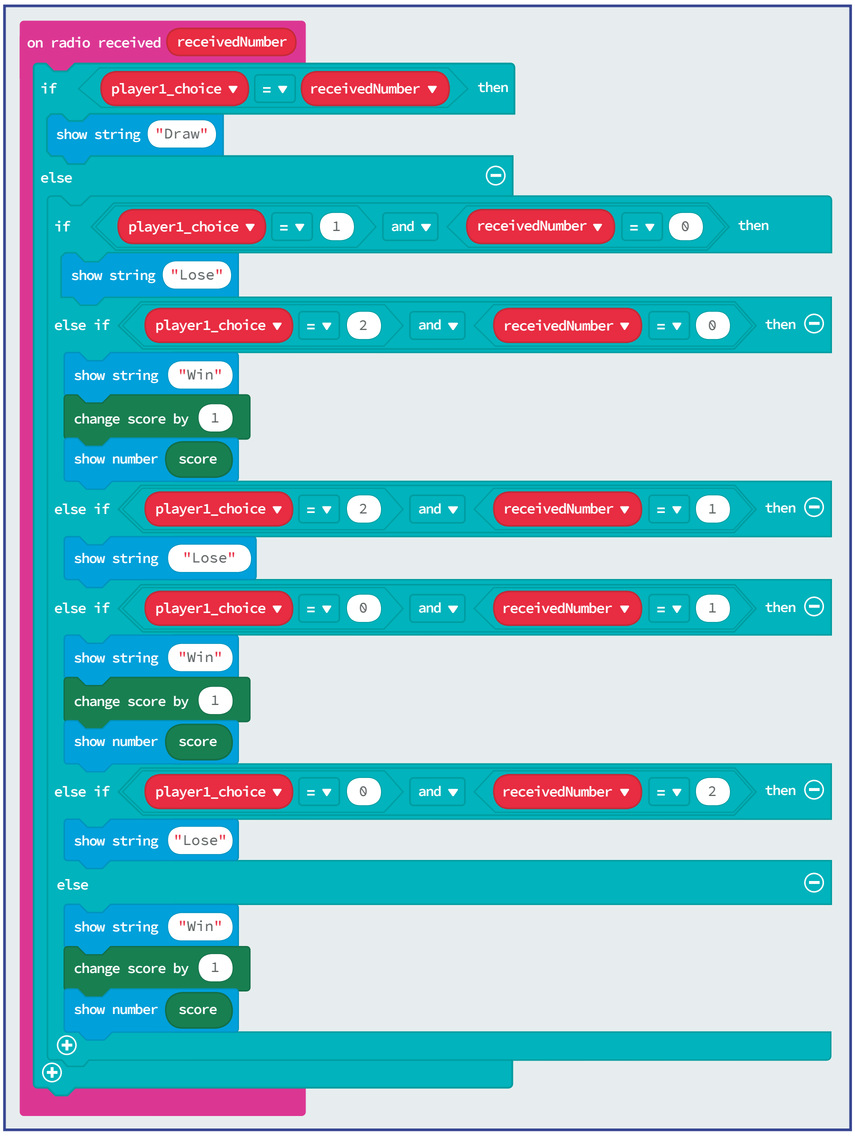
11

Now we need to add some more logic to deal with these 6 conditions of the game. It does not matter which order you check the conditions in, you just need to keep track of which ones you have done with your code blocks. Another coder's solution is likely to be different to yours but that is fine, as long as all combinations are checked.

Add an **else if** block to your blocks to test the next condition and then keep adding others until you have dealt with all the squares in the table. The block show in the example below is for ***player1_choice*** = R (2) and ***receivedNumber*** = S (0)

After each condition has been checked, we then need to adjust the ***score*** depending on who has won. If the player has won, then their score can be increased by changing the built-in ***score*** variable by 1, otherwise they lose. You need to make sure you test all the combinations which are shown in the table above and test your program so that it always works out the correct winner and gives them a point.

Check these blocks carefully. It is easy to miss a block or not change a number and this will make the program behave strangely. The very last condition does not need to be checked because it is the only possible combination left.

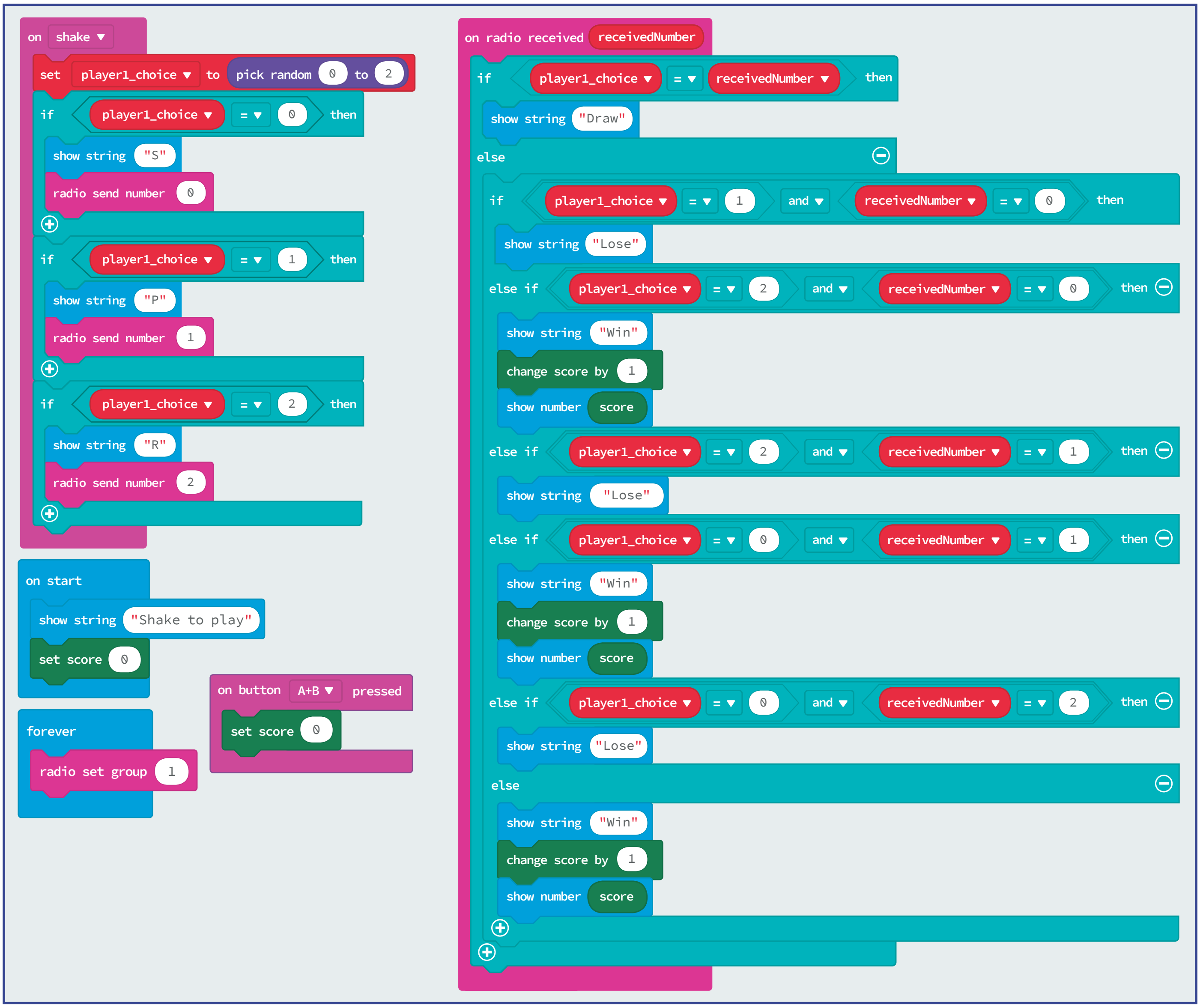


12

Two player

Don't forget that this is a two player game and so this program needs to be uploaded to two micro:bits to play properly. You may want to change the ***player1_choice*** variable to ***player2_choice*** on the second micro:bit but the program will work fine without this change.

The full solution



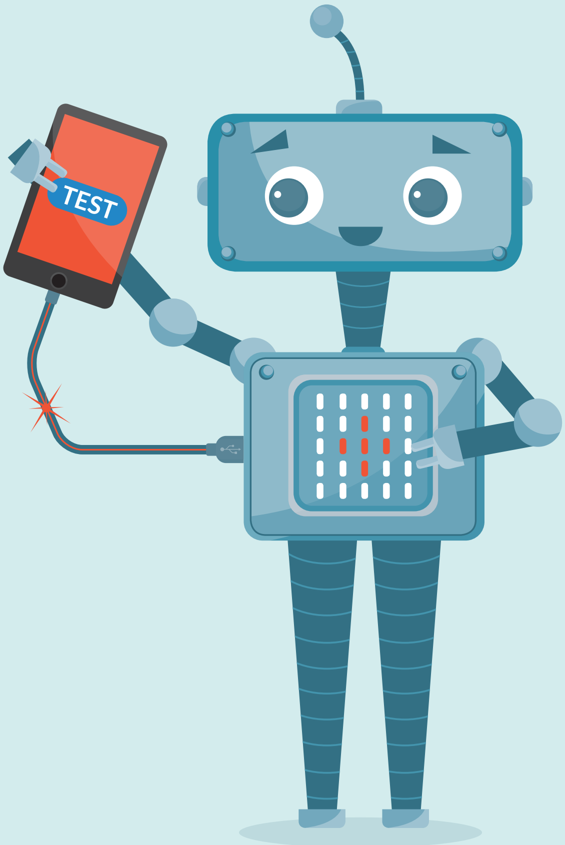
13

We have now met all the success criteria:

- Make a game where S (Scissors) beats P (Paper), P beats R (Rock) and R beats S
- The program will randomly select S, P or R when shaken
- The program will transmit the selection via radio and will determine if it has won or lost the match
- The program will keep count of wins and losses until reset by a button press (A and B)

Test time!

We have used a few different blocks here and lots of blocks within blocks, so now is the time to test your program and make sure that it behaves as you would expect. Make sure to check that the game works correctly a few times!



Stretch tasks

- Change S, P, R to icons for scissors, paper, rock.
- Make the program include three players.
- Add another option V (Vulcan) that beats S but loses to P and R.
- Change the program so the player has to press a button that starts a 3 second countdown before the choice is made.
- Adapt the game to use functions to remove the nested *if* blocks.

Final thoughts

This has been the most complicated program so far and has used some complicated logic. Keep going! There are other ways to make this game and this is just one of them.

SMART CITIES

micro:
project

Success criteria

- Design a street light that changes its behaviour depending on the local conditions
- Design the body of the street light to be efficient and robust
- Design the light housing to minimise light pollution
- Create a program that uses the micro:bit's sensors to make the street light smart
- Create a smart street light that uses LEDs to save electricity usage
- Consider how the data generated by smart street lights could be used
- Suggest ways in which the data collected from the sensors in the street lights across the city could be used by other organisations

1

What is the Internet of Things (IoT)?

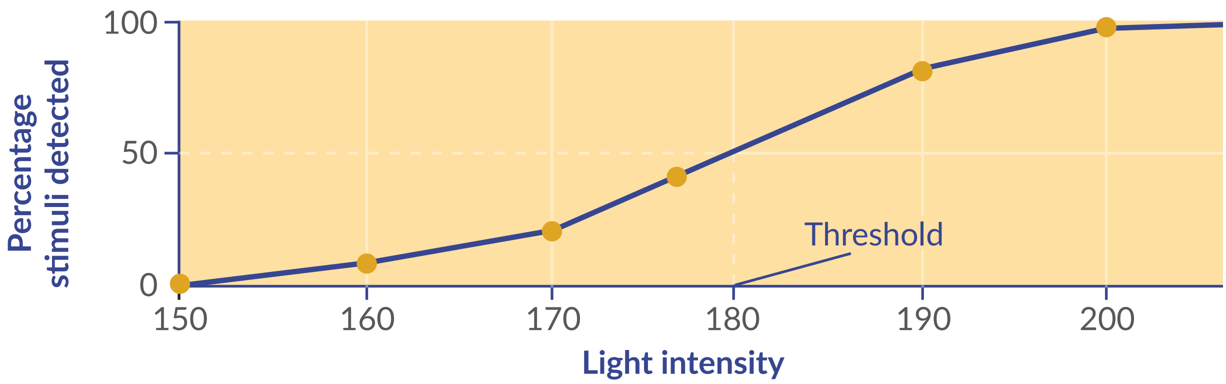
One way in which technology is improving our lives is by adding computers to everything. These computers are tiny but contain sensors that allow the computer to control a system or device. They can also transmit and receive data to and from the internet. This allows much more precise control over the device as digital sensors are more precise that analogue sensors in some applications such as temperature control.

In this project we will be designing and making a "smart" street light for a city. The city currently has traditional street lights and these use a lot of electricity and have sodium bulbs which are expensive to replace.

2

How traditional street lights work

Currently street lights work by either being manually controlled, turned on at a given time or more modern ones have a light sensor that turns on the light when a light threshold is reached.



This is very inefficient as the lights stay on constantly all night at the same brightness.

3

The city wants the new street lights to use sensors to detect the light levels and react to the light levels with differing light intensities. They also want you to fully utilise the sensors on the devices to provide as much data as possible that could be used by other departments in the city. For example, the local weather station may use the temperature data to accurately predict temperature.

4

Additional considerations

- The street light should not exceed the footprint of a traditional street light
- The street lights can be connected to the internet if needed
- The environment agency is very keen for the lights to be as efficient as possible in terms of both materials used and power consumed to minimise the environmental impact
- Consider what other sensors could be added to improve functionality

5

Getting started

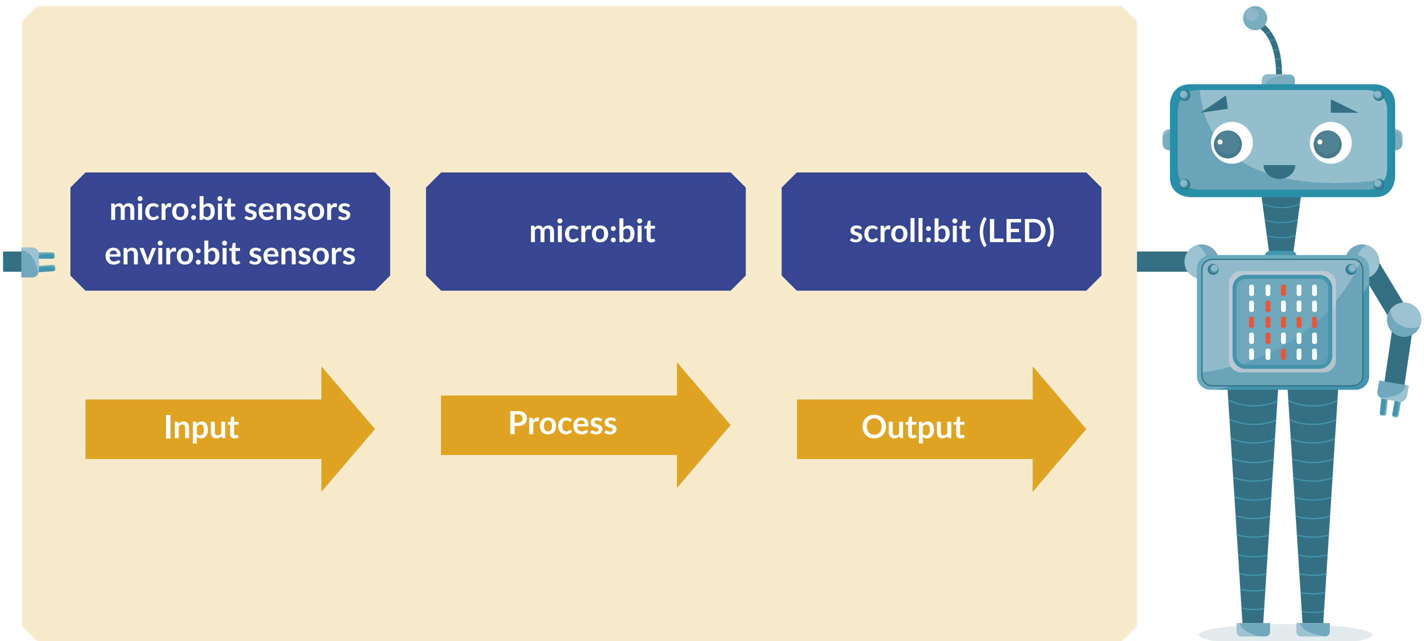
First we need to understand what sensors the micro:bit has that will be useful in our smart light.

We are going to also use two peripherals that add additional sensors to the ones built into the micro:bit. We are going to use an enviro:bit and a scroll:bit.

6

Sensing the surroundings

First, we need to understand what inputs and outputs the micro:bit has and think about how best to use them. We need to decide which inputs we want to use, how we will process that data and the output we want to trigger.



Input	Process	Output
Sensors built into the micro:bit:		
Temperature sensor Light sensor Accelerometer Magnetometer	Trigger outputs based on thresholds, timings etc	LED matrix
External peripherals (enviro:bit and scroll:bit)		
Temperature sensor Pressure sensor Humidity sensor Light and colour sensor Microphone	Controlled by a micro:bit	17x7 white LEDs Individual brightness control Radio Bluetooth

7

Enviro:bit

The enviro:bit is a micro:bit peripheral that adds in several additional environmental sensors that will be useful for a smart street light.

8

Scroll:bit

The scroll:bit will form the lamp part of our smart street lamp. We will use a scroll:bit as it has white LEDs and the intensity can be controlled to maximise efficiency.

9

Get designing!

Now we need to design the system and the hardware for the new light. There are lots of inputs that can be used here and you will need to think carefully about how to use the sensor data to achieve the success criteria. You will also need to be mindful that you have two micro:bits in play and they may need to talk to each other.

Use the design template on the following page to design your product.

Success criteria

Design a street light that changes its behaviour depending on the local conditions

Design the body of the street light to be efficient and robust

Design the light housing to minimise light pollution

Create a program that uses the micro:bit's sensors to make the street light smart

Create a smart street light that uses LEDs to save electricity usage

Consider how the data generated by smart street lights could be used

Suggest ways in which the data collected from the sensors in the street lights across the city could be used by other organisations

Additional considerations

The street light should not exceed the footprint of a traditional street light

The street lights can be connected to the internet if needed

The environment agency is very keen for the lights to be as efficient as possible in terms of both materials used and power consumed to minimise the environmental impact

Consider what other sensors could be added to improve functionality

Materials:

Essentials features:

Nice to have:

Input process output:

How is this design better than a traditional light?

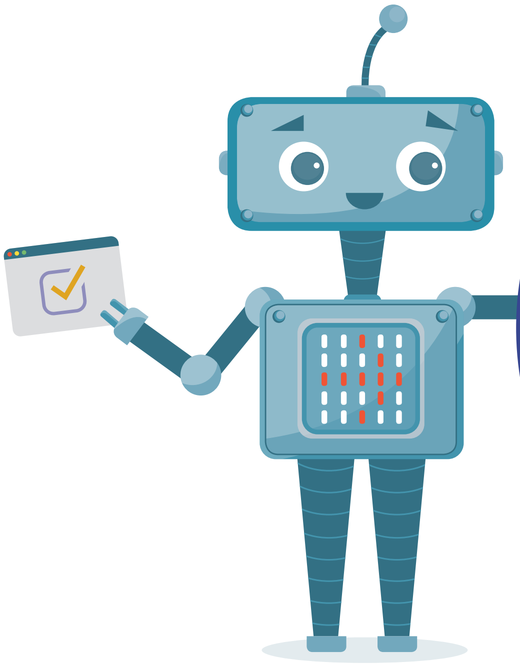
Who does this new design help?

Sketch of the light:

How could it be better?

Team name and branding/logo:

arm School Program



PRO *TIP*

Ensure that you check your success criteria regularly. Think carefully about the design of the light housing and the frame. It will need to be robust enough to cope with high winds and harsh weather.

Stretch tasks

Community notice board

The community want a notice board on the street light, where they can post about lost pets, local events and community groups. People will create a notice at home, send it for approval and then have it displayed.

- Design the system using a flow chart.
- Connect your micro:bit to a monitor to begin creating the noticeboard.

Dog deterrent

Some owners allow their dogs to wee on lamp posts. The council would like to deter them.

- The sign needs to detect moisture from the dog and then frighten it so it does not want to come back.
- Use the moisture sensor to detect the dog.
- Connect a speaker to create a noise.
- Find out about sounds that dogs can hear that humans cannot.

Police interceptors

At the scene of an accident or incident, the police would like to turn up the street light.

- The police officer will be able to turn up the street light using a phone app.
- Find the micro:bit app on the app store.
- Pair the phone to the micro:bit using Bluetooth.

Security light

Local people have asked for a help button so that they feel safer walking at night.

- The help button will instantly set the light to maximum and an alarm will sound.
- A radio signal is sent to alert the control room.
- Use the A and B buttons to prompt different events.
- Can you think of other ways streetlights could help people feel safer at night?

Bin day

Local people have asked for the streetlight to remind them when to put their bins out.

- The system needs to tell them which coloured bin or bins to put out.
- The normal bin day may change when there is a bank holiday.
- Find the website that has a bin calendar for your area.
- Create different coloured light to match the bin colours.

Freezing conditions

The transport services need to know when the temperature has fallen below zero, so that they can grit the streets and pavements.

- The system needs to detect when the temperature has fallen below zero.
- The street light can show a warning picture.
- Create a warning icon for the light to show.
- Use a fridge to test your final system.

Moon tracker

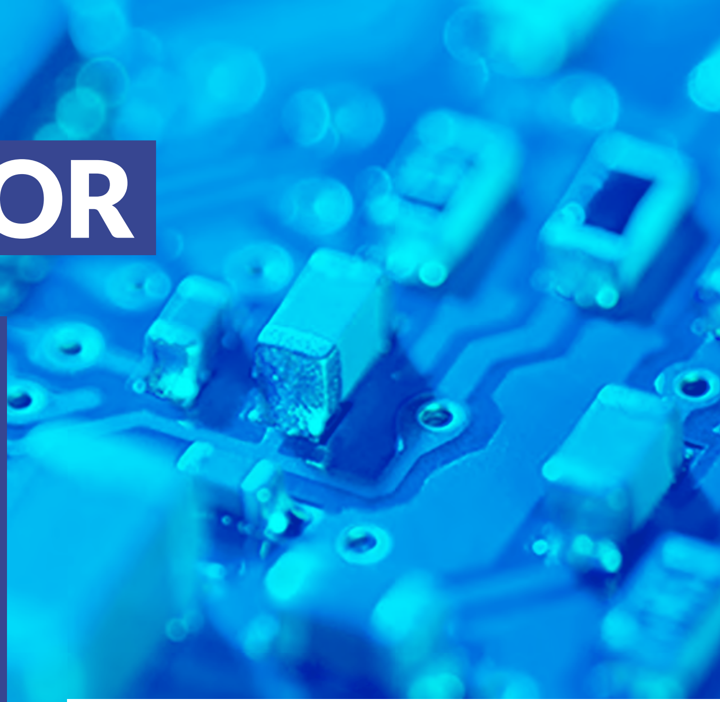
The moonlight can cast shadows. The light could move with the moon so that it is always illuminating the other side, so minimizing shadows.

- The moon rises in the east and sets in the west.
- Learn how to use the servo with the micro:bit to create a 180 degree turn.
- Use the compass sensor to set the direction.

Final thoughts

One potential downside of the IoT revolution is the issue of e-waste. Adding electronics to everything may produce additional waste that contains rare earth metals, unless we consciously design technology to be more readily recyclable and not rely on unsustainable materials. When designing our next project, consider how to minimise e-waste.

TREE PROTECTOR



Getting started

A flourishing life on land is the foundation for our life on this planet. We are all part of the planet's ecosystem and we have caused severe damage to it through deforestation, loss of natural habitats and land degradation. Promoting a sustainable use of our ecosystems and preserving biodiversity is more than just a cause. It is the key to our own survival.

We have been tasked with creating a product that helps protect forests and combats deforestation. The government wants a product that can be attached to trees to alert the authorities if trees are being cut down by illegal loggers.

Success criteria

- ☑ Can be attached to a tree securely
- ☑ Alerts the authorities if:
  – The tree is cut down
- ☑ Gives the authorities the location of the fallen tree

1 Breaking down the problem

We are going to start by creating a program that uses the micro:bit's sensors to allow us to sense when a tree has been cut down. We will start with designing the program.

2 Input, Process, Output (IPO)

We now need to think about the IPO of one of the required features for the product to 'alert authorities if the tree is cut down'. You can use this table to help you design a solution for the other success criteria.

Input	Process	Output
Acceleration and tilt sensor data to sense if tree is falling over	If acceleration of angle of sensor exceeds a threshold then output	Send message to authorities including location data

PRO *TIP*

The authorities will need GPS co-ordinates so that they can intervene if there is illegal logging occurring. We could ‘hard code’ the GPS co-ordinate into the micro:bit or we could use a GPS peripheral to get a live GPS co-ordinate.

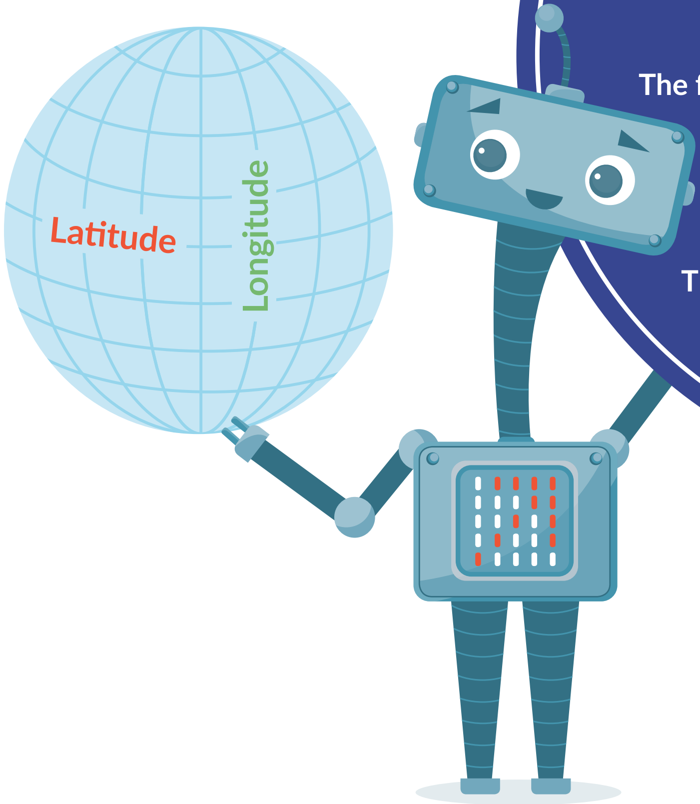
The format of a GPS co-ordinate is made up of a latitude and a longitude:

Latitude: 52.1818424

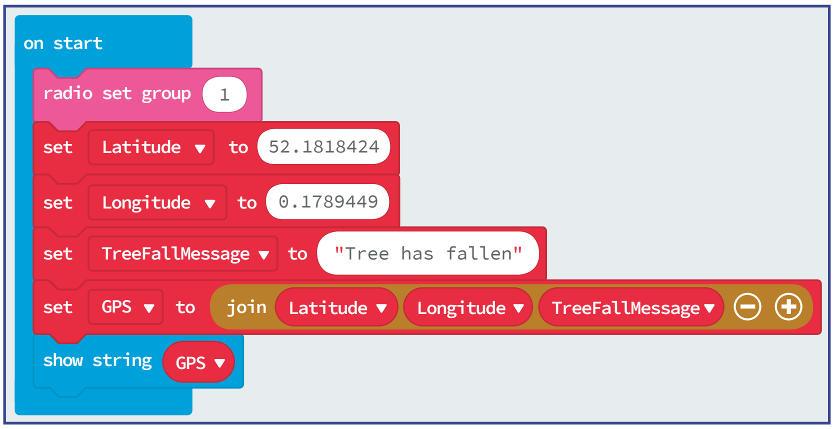
Longitude: 0.1789449

These combined: 52.1818424,0.1789449

Can you work out where this GPS co-ordinate is for?



3 Example code for the node (on the tree)



These blocks run as soon as the micro:bit is powered on.

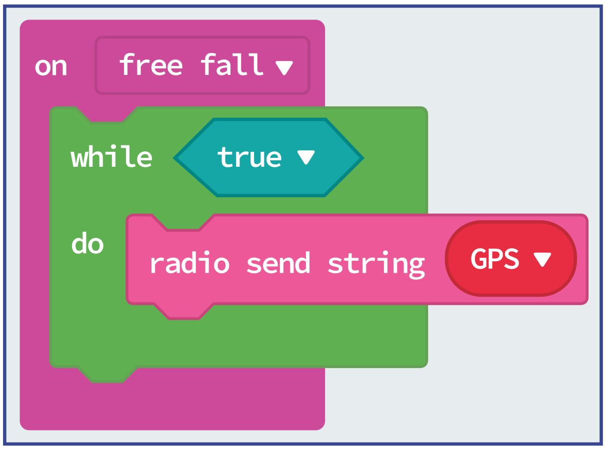
The first action is to set the ‘radio group’ to 1, this also needs to be done on the gateway so that the micro:bits can talk to each other on the same radio frequency. If these numbers are different, it won’t work!

4

Next, 2 variables are created to store the GPS data. Latitude and longitude must be split. Why is this? The micro:bit treats variables as numbers by default and the comma (,) confuses it and so it strips out everything after and including the comma.

5

We then create two more variables, one called 'tree fall message' with a message (drag a string box from the Text menu to allow you to enter the value as a string) to transmit to the authorities and another called GPS that 'joins' the latitude, longitude and the message. The 'GPS' variable is then shown on the micro:bit's screen to show it is working.



6

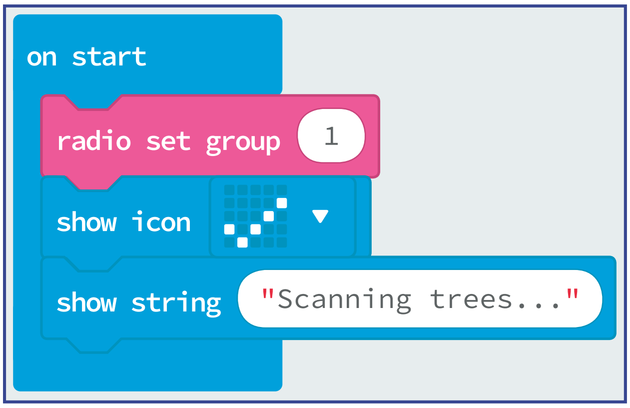
In this example, the micro:bit waits until it senses a 'free fall'. This would happen if the tree was chopped down and it fell (assuming the micro:bit is high enough up the tree!). You will need to test this!

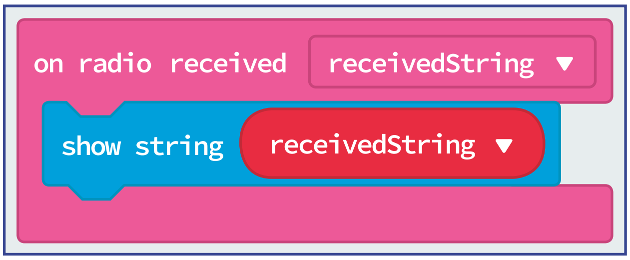
Next, we use a 'while true' loop, we use this so that the blocks inside it will repeat forever, as True will always be True.

The blocks inside the loop will transmit a string (text) which contains the message, latitude and longitude (as a string). These can be received by the 'gateway' and then sent to the authorities who can then come and see if the tree is being chopped down or if it has just fallen over naturally.

7

Example code for the gateway (with the authorities)





This project is designed to be created on a physical micro:bit: a node in one MakeCode project and a gateway in another project. (Second "on start" block is not needed in the MakeCode simulator.) This first block again runs as soon as the micro:bit is powered on. The 'radio group' is set to the same group as the node (this is important).

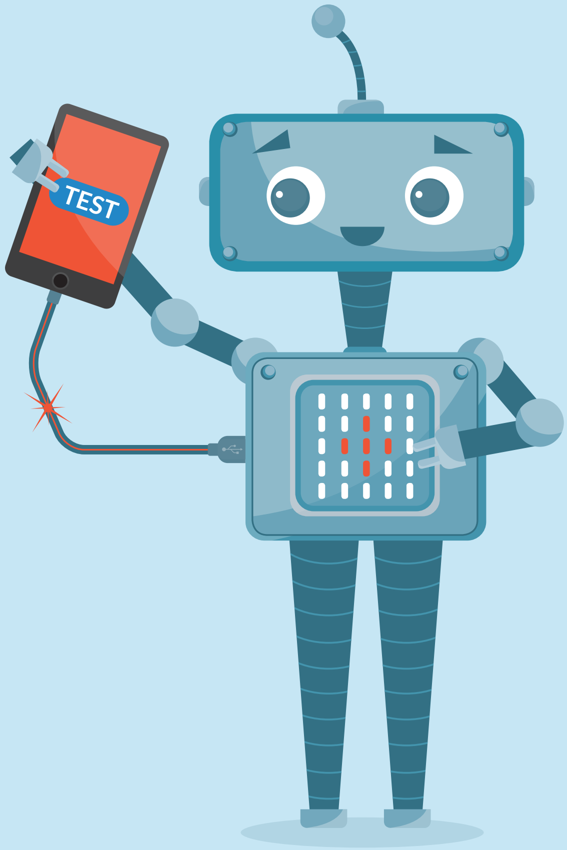
Next, we display a tick and some text to show the program is active.

The next set of blocks waits to receive a string via radio. It stores this string in a variable called 'receivedString' and then shows this string on the LEDs.

This string will be the 'GPS' variable sent by the node and contains the GPS co-ordinates of the fallen tree.

Test time!

The alert to the authorities will need to be tested so that it doesn't send alerts accidently, like if the tree is just blowing in the wind for example. You will need to test the sensitivity and set the thresholds so that they only trigger an alert when they are meant to.



Stretch tasks

- Rather than manually adding the latitude and longitude, adapt your product to be able to get an accurate latitude and longitude from a GPS device.
- Adapt your product so that it doesn't just rely on the micro:bit sensing freefall to alert the authorities. What other sensors or programming techniques could you use to sense if a tree is being cut down?
- Sometimes trees blow over naturally in high winds. The authorities don't want to investigate every time this happens. They have decided to attach your product to lots of trees in the forest but only want to be alerted if more than one tree collapses per day. Can you combine other micro:bits to deliver this feature?
- Having devices in hard to reach places is a potential source of e-waste. One source of e-waste is the use of batteries and these will be difficult to change on this project. How can we power the tree protector without needing to change the batteries? Also, how can we make sure the device itself doesn't become e-waste?

Final thoughts

We can protect life on land together. Here you can see what you can do to contribute. Find organizations to support, information to share and some useful tips for your everyday life that can really make a difference.

https://www.globalgoals.org/15-life-on-land

https://www.hackster.io/149085/panic-button-using-xinabox-micro-bit-and-ubidots-bf4dc8

INTRODUCTION TO DATA LOGGING

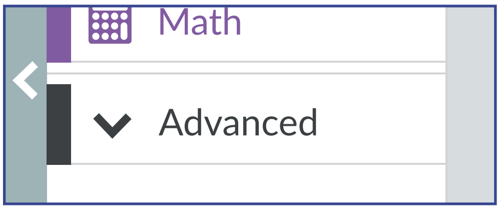
Setting the scene

A new feature of the micro:bit V2 is its ability to log data. This means that it can take readings from its sensors and store them in memory so that the data can be accessed later. This allows you to use the micro:bit to take all kinds of measurements over time and then learn from the results. This is also called “data science”.

Success criteria

- Understand what datalogging is and what it can be used for.
- Be able to add extensions to MakeCode.
- Create a simple data structure to contain light level readings.
- Populate the data structure with light level data.
- View and export the logged data.

1

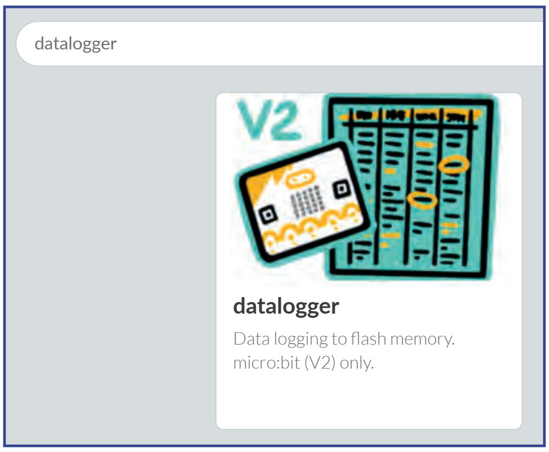


To add the new blocks, click on “Advanced” underneath the basic blocks menu to expand it.

2

At the bottom of the screen, click the “Extensions” button. This will open a new search screen with all the available extensions. There are lots! These extensions add additional functionality to your micro:bit. Lots of them are for specific peripherals like the enviro:bit.

3



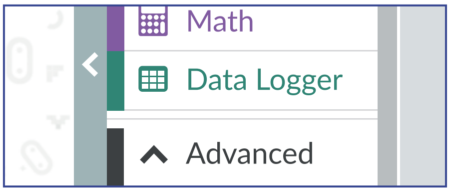
Type “datalogger” into the search box and you should see the new datalogger extension. Notice that “datalogger” has no spaces—you will get many more results if you type “data logger”.

PRO TIP

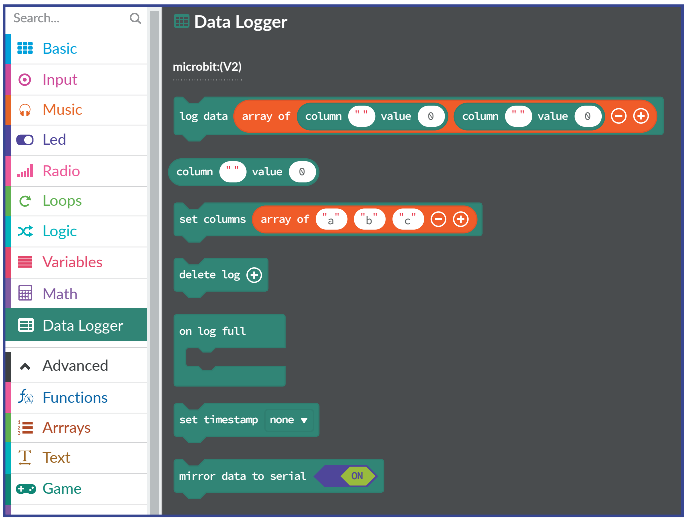
These blocks will only work with the new micro:bit V2.

4

You will now see the Data Logger blocks at the bottom of the basic blocks menu.



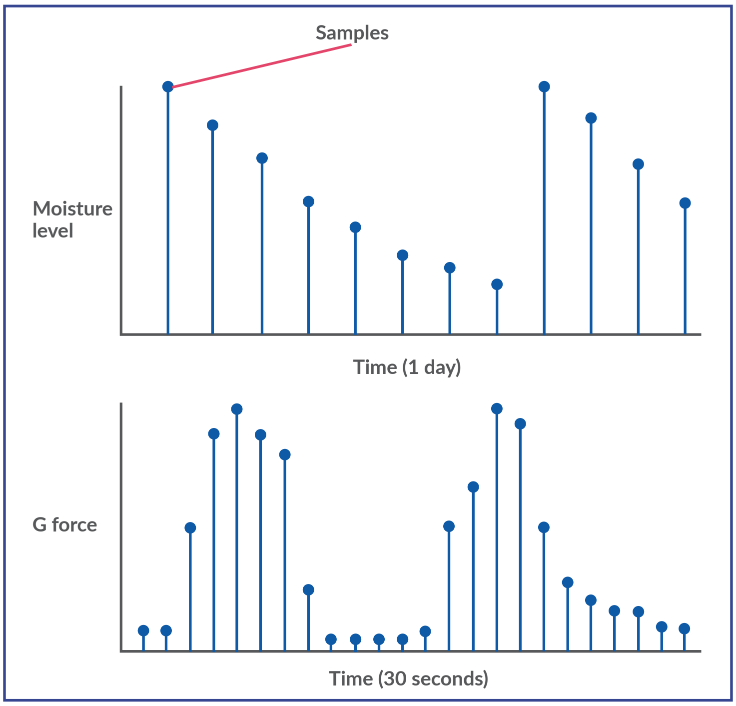
5



These are all the blocks in the datalogger extension. You now have all the extra blocks you need, but let's quickly look at what datalogging is and how it works.

6

Datalogging is where you record data over a given amount of time. That data can be anything the micro:bit can take as an input, for example, temperature or light levels. This data can be sampled quickly or slowly. For example, if you wanted to log the Gs your scooter gets on a halfpipe then you would sample a lot of data over a short time. If you were measuring light and moisture levels for a plant in a greenhouse, you would sample much more slowly for a longer period, like every hour. This sampling rate is also referred to as "automatic logging".



You can already take inputs and process them, but these blocks allow you to store this data and then do something with it later on. This is really useful if you want to analyze data. You can even start to explore the world of data science by applying statistical techniques to analyze your data in clever ways. This is advanced stuff, which we will explore later, but let's start with the basics.

7

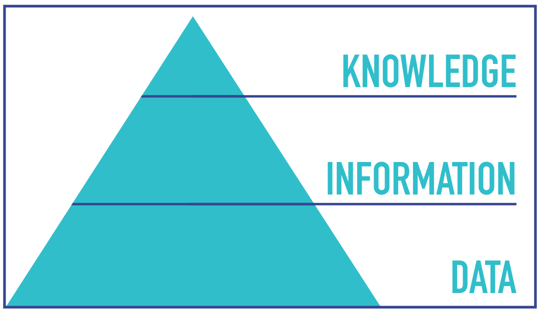
The data is stored in a file called MY_DATA, which you can access on the micro:bit from a PC. You can export the data as a .csv file (for spreadsheets) or view the data by simply clicking on it.

8

The data you record will be stored in an array. This is a fancy word for a table and you may already be familiar with it from using spreadsheets. The array has rows and columns: rows go across from left to right and columns go down from top to bottom.

	A	B	C
1	time (source1)	time (source1)	data.0
2	0	0	2449.7
3	0.007	0.109	2449.85
4	0.007	0.213	2449.96
5	0.007	1.819	2451.5
6	0.007	3.678	2451.74
7	0.007	3.794	2453.52
8	0.008	3.901	2453.63
9	0.008	4.009	2453.74
10	0.008	4.125	2453.85

9



An important concept to understand is the difference between data and information. Data is raw facts and figures, for example, letters and numbers. If this data is shown in a useful format, for example, a graph, then it is information. Information is data with context, or data that has been processed into something with meaning.

Using our greenhouse example, the data on its own is meaningless, just a stream of readings. However, when we know that the readings show how the light levels and moisture levels change over time and how this affects the plants, we have information. Information is useful! Taking that information and analyzing it is called data science. If we do this properly, we can turn our information into knowledge, which we can use to improve what we are logging.

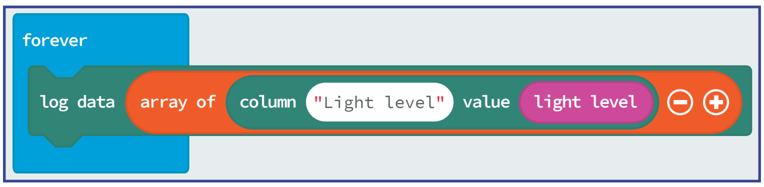
10

You need to label your data so you don't just end up with lots of numbers with no meaning. The label is the heading at the top of a column of readings. The rows are filled in each time an input is sampled.

	A	B	C
1	time (source1)	time (source1)	data.0

11

Let's start with a simple example. Add a ***log data*** block from the Data Logger blocks to a ***forever*** block. Name the column ***Light level*** and add a ***light level*** input block to the value space. This will constantly log the light level and record it.



12

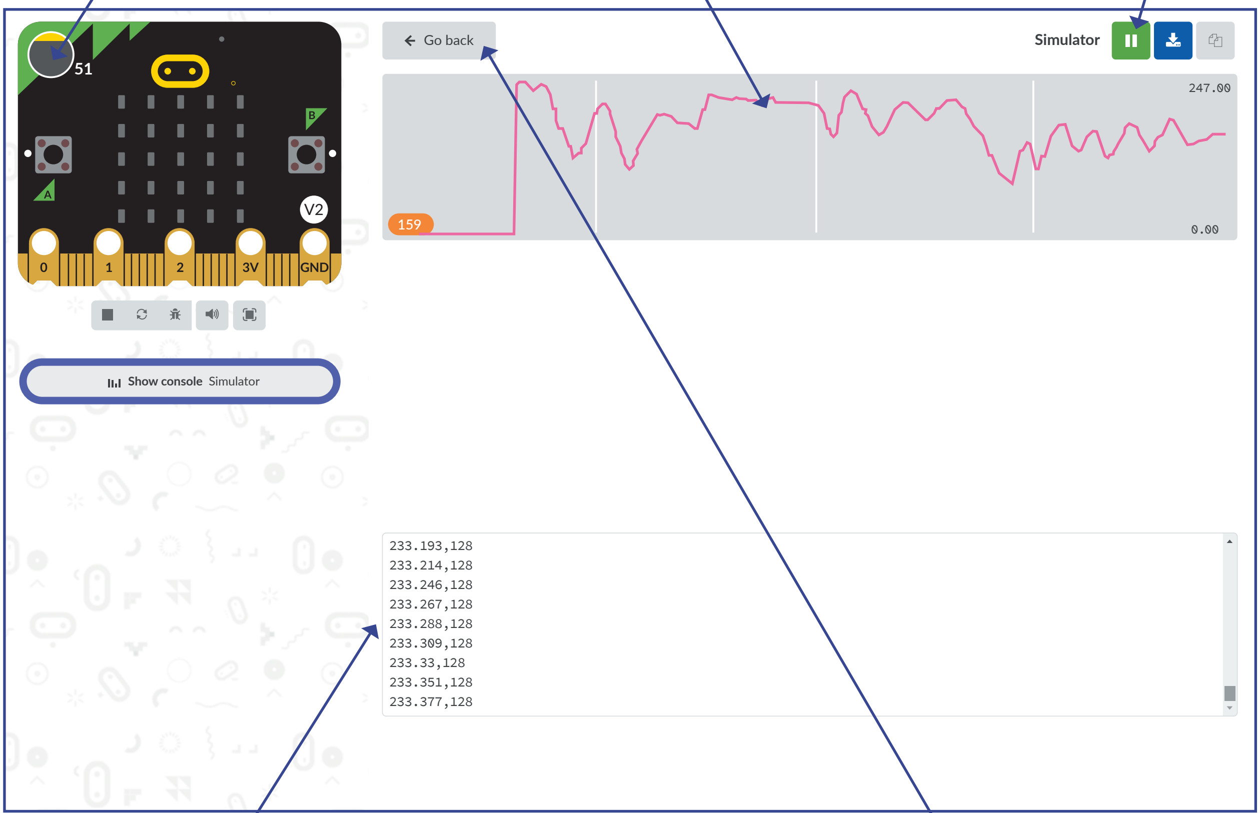
You will notice that this "Show console" button appears under the micro:bit simulator. If you click it, you reach a new screen where you can see the live data being logged from the simulator.

13

You can change the light level by dragging the simulated light level circle to see how the changes affect the data.

The graph shows the data in real time.

You can also pause the record, export the data, or copy the text using these buttons.



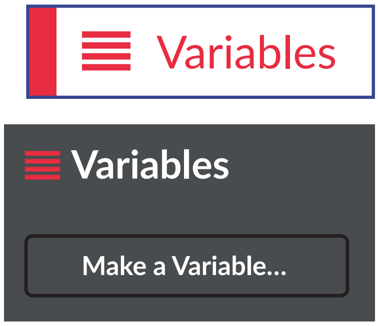
The raw numeric data is shown here.

PRO TIP

Click on the blocks at the top of the screen to get back to your code.

14

The micro:bit only has a limited amount of memory, so you need to program a way to start and stop the logging. Use the buttons for this.



To do this, you need to create a variable. Click on the "Variables" icon, choose "Make a variable", and then name it something sensible like "temperature" or "light_level".

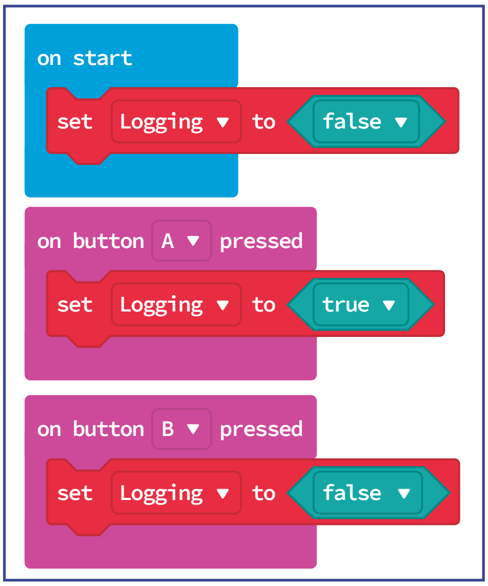
You can now use this variable to control when logging starts and stops by testing whether it is ***true*** or ***false*** (on or off).

15

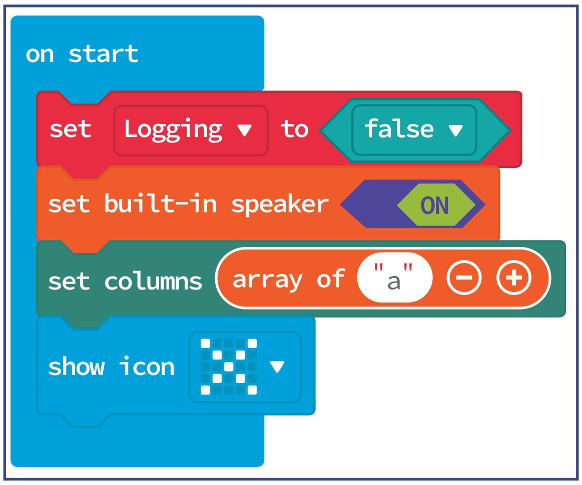
Start by setting the ***logging*** variable to ***false*** on start. This means the micro:bit is not logging as soon as you start it up.

Next, use an ***on button A pressed*** block to set the variable to ***true***. You will use this to start the logging.

Next, use the ***on button B pressed*** block to set the ***logging*** variable to ***false*** to stop the logging.



16



So far, you can't see anything happening, so next add some icons to the LEDs. These will show what state the variable is in, either ***true*** (logging) or ***false*** (not logging).

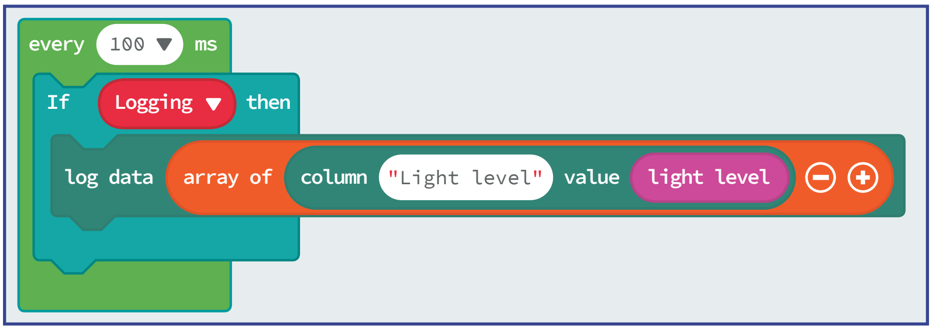
You should also add in some sound blocks to give an audio confirmation that the logging has started and stopped. To do this, enable the built-in speaker.

Set up the column to store the data. The column will hold the light level, with each row being a measurement at a given time.

17

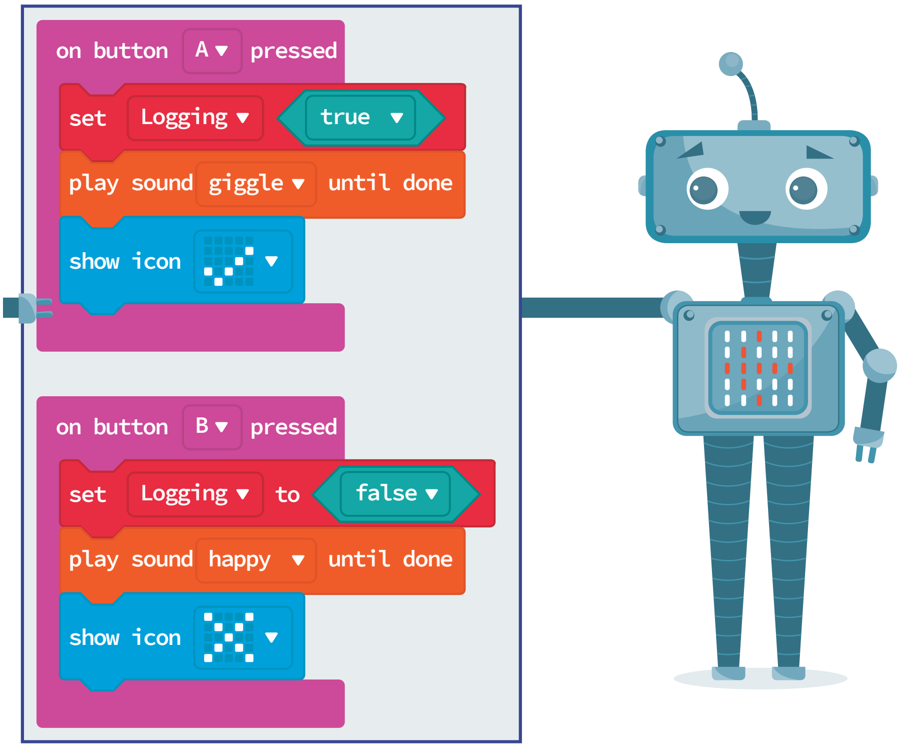
Next, add the blocks that start the datalogging when the ***logging*** variable is ***true***.

Use an ***every x ms*** block so that a reading is taken every x milliseconds. Experiment with changing the number of milliseconds and see how this affects your results. The more readings you take, the quicker you will use up the available memory, but the more accurately you will monitor what is happening.



18

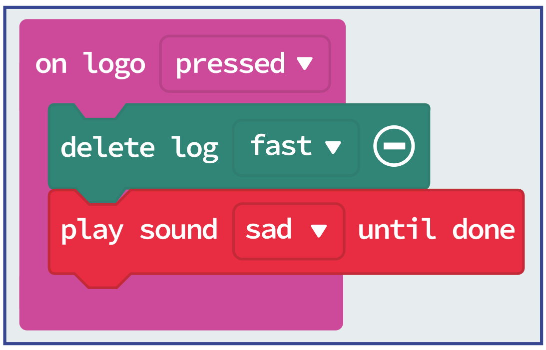
Next, add sounds and icons to the ***on button*** blocks to provide audio feedback.



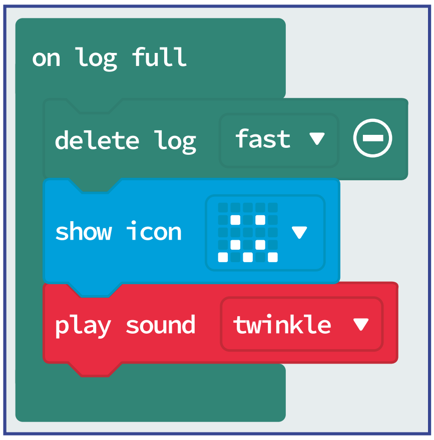
19

These blocks allow you to delete all the data by pressing the micro:bit logo on the front of the micro:bit. You could use an ***on button A+B*** block instead if you think this makes it too easy to accidentally wipe your data.

You can also add the ***sad*** noise to indicate the data being deleted.



20



Now let's add the blocks that allow the micro:bit to keep recording even once the memory is full. These blocks tell the micro:bit to overwrite the oldest data with new data.

Add in an icon to show this is happening as well as a suitable sound. You could even use a ***show string*** block here to show that it is full.

21

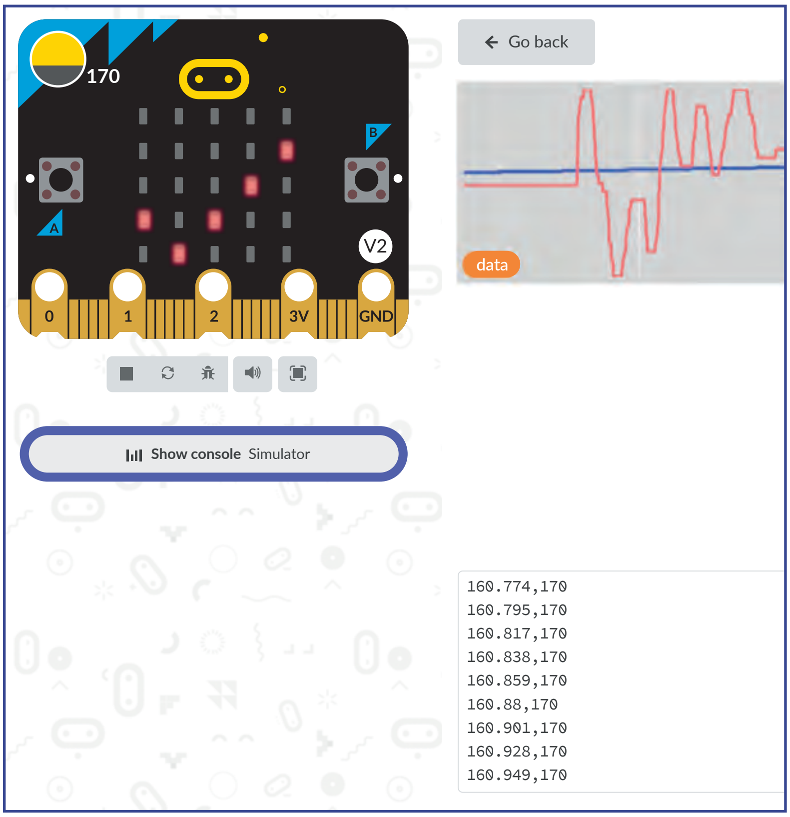
Now see if it all works in the simulator. Click "Show console" to test the functionality.

You may need to click the A button on the simulator to see the button.

22

Test that the buttons do as you expect, that the LEDs show the correct icon, and that the right sounds play.

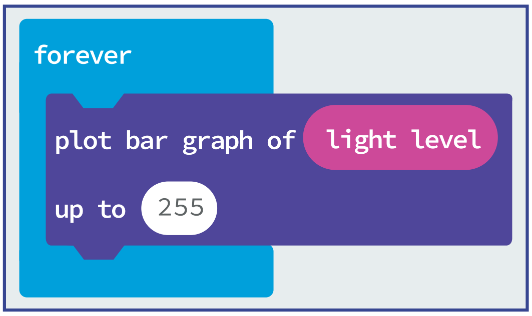
Adjust the light level to see the logs change.



23

Let's now add a bar graph that shows the light level on the micro:bit. This will show how much light you need to get a reading. It's more than you might think!

These blocks simply plot a bar chart on the LEDs to show the light intensity the micro:bit is sensing. This uses the same light sensor that you are using for datalogging so it will help you to troubleshoot your program.



24

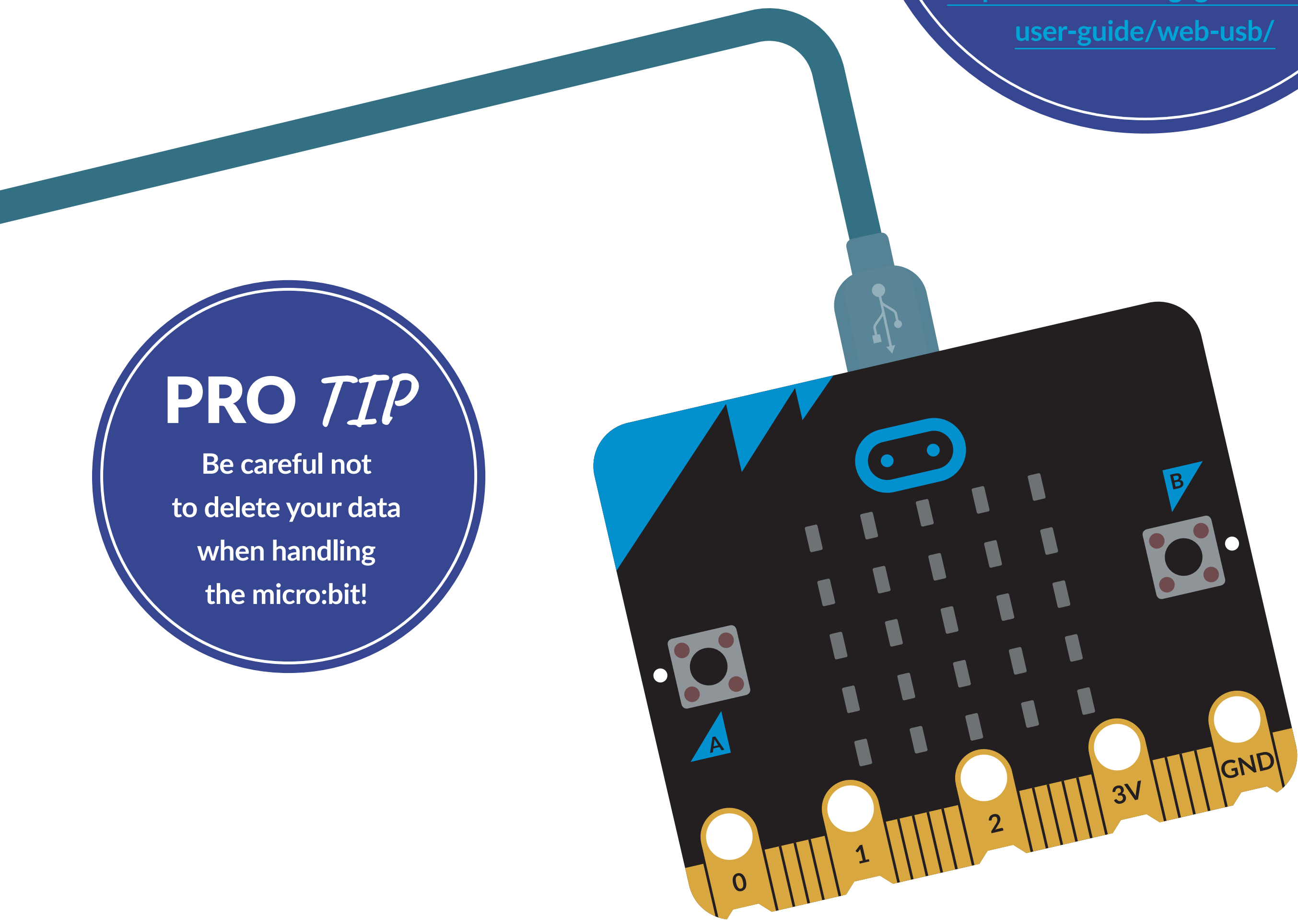
Now test your program on the real micro:bit!

Download the .hex file to your micro:bit. Using a torch or other light source, check that the bar graph is registering the changes in light on the LEDs. If it is, then go ahead and press the A button to record the results.

After a short while, press the B button and then plug your micro:bit back into your PC with the USB cable.

PRO *TIP*

You can sync your micro:bit with MakeCode using WebUSB. This means you can just click "Download" to get your code on your micro:bit. You should get a prompt in MakeCode that will guide you through the process. https://microbit.org/get-started/user-guide/web-usb/



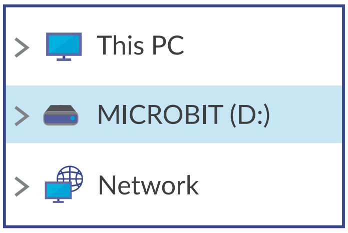
PRO *TIP*

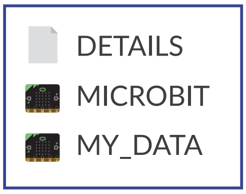
Be careful not to delete your data when handling the micro:bit!

25

When you plug in the micro:bit, you should now be able to see a new file called MY_DATA.

Open it and see what's inside!





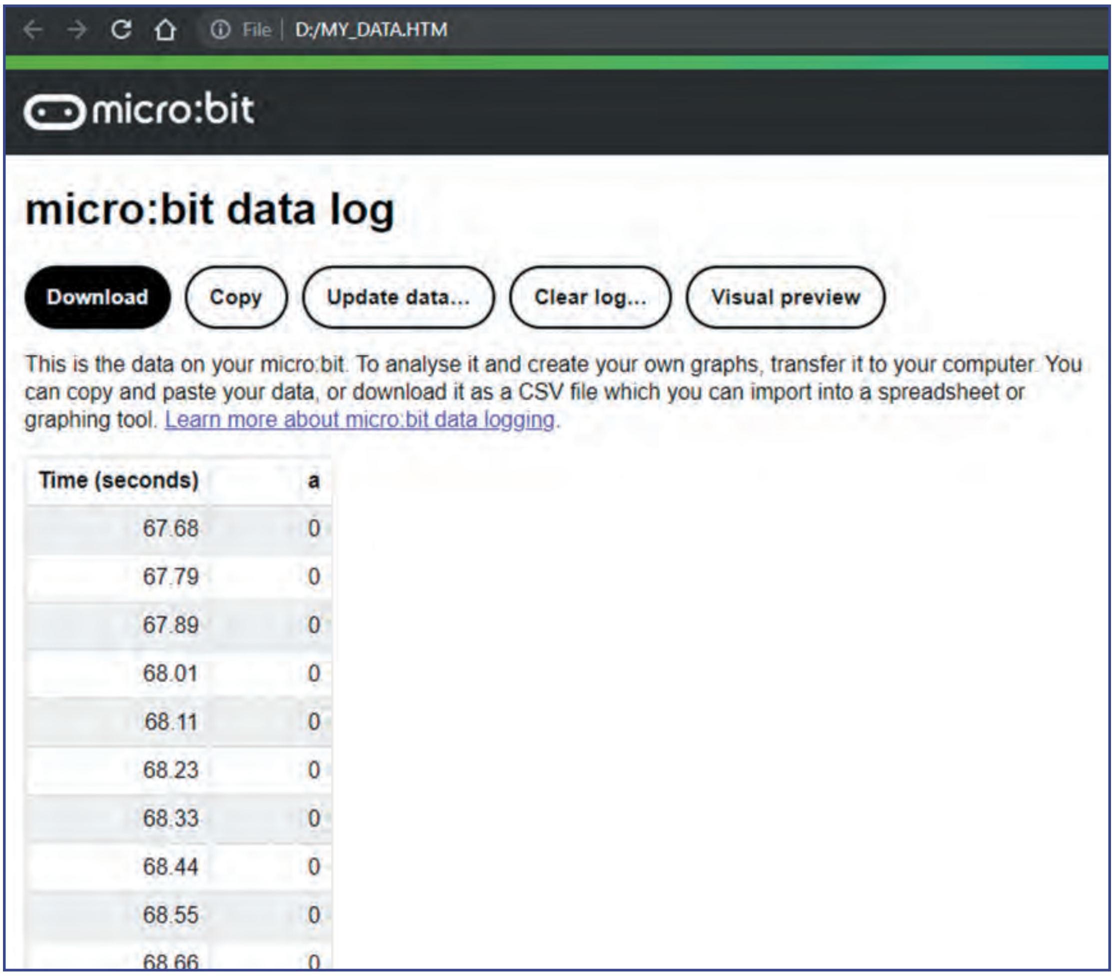
PRO *TIP*

If you get a blank page when you open MY_DATA, you will need to update your micro:bit's firmware to the latest version. There is a guide on this here: https://microbit.org/get-started/user-guide/firmware/

26

This is what your MY_DATA file will look like. You have multiple options now: you can download and save the data as a .csv file (for spreadsheets) or see a visual preview as a chart.

Now the data is gathered, you can analyze it either in a spreadsheet or using Python. See whether any patterns emerge, and whether the data tells you anything interesting. We will get into this side of things in a later project.



Stretch tasks

- Modify the program to take in more than one type of reading. Other readings could include temperature, direction, acceleration, or anything else that the micro:bit can measure.
- Think of a project where measuring something over time would be useful. How could the data help you?
- Download the data as a .csv file and then open it in a spreadsheet application. Create a chart to visualize the data. Choose which chart to use carefully!

Final thoughts

Being able to store data and analyze it later is a very powerful tool. The data you record can help you improve what you are measuring. For example, a farm could measure soil moisture and light levels, see how these affect the crop yield, and consequently improve its profitability. Data can also be used to spot patterns, and you can create programs that do this for you—this is known as machine learning. In our farm example, a form of machine learning could be used in which the machine learns from the data which light and water levels get the highest crop yield. This can be used to automate watering and light levels to maximize the yield. Data science and machine learning can be applied to almost anything and are rapidly changing the world we live in.

OCEAN HEALTH MONITOR

micro: project

Getting started

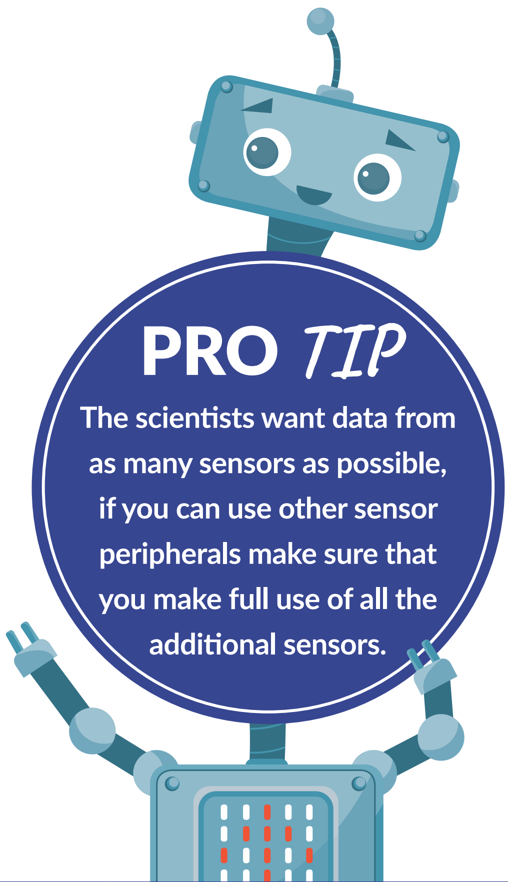
A group of marine scientists have asked you to develop a prototype floating sensor node that they can leave in the ocean. The will transmit data to them so that they can study climate change in the sea.

Success criteria

- Design and create a floating sensor node using a micro:bit
- The beacon must transmit sensor data to a gateway micro:bit every 10 seconds
- The beacon must also transmit its unique ID number (there will be lots of nodes!)
- The gateway micro:bit must be able to show the data on its LED screen

1 Input, Process, Output (IPO)

Node		
Input	**Process**	**Output**
Water temperature	Format data	Transit data via radio
Air temperature	Unique ID number	Transmit unique ID number
Air pressure		
Humidity		
Air quality		
Light intensity		
Accelerometer (waves)		
Compass		



Gateway		
Input	**Process**	**Output**
Data via radio	Format the data	Display the data on the LEDs
Unique ID number via radio		

2 Building the prototype

The prototype should be designed and made to float but this does not need to be tested as the scientists are only interested in the sensor data and not the floating part as this will be made by a specialist boat designer.

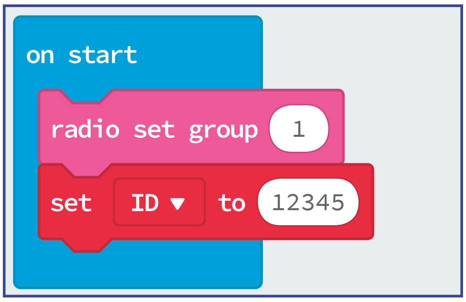
In the following example the air temperature will be sensed and transmitted from the node to the gateway.

PRO *TIP*

The 'on radio received' event can only be created once due to hardware limitations, to work round this you can simply concatenate (join) your sensor data and send it all once, or use more than one micro:bit.

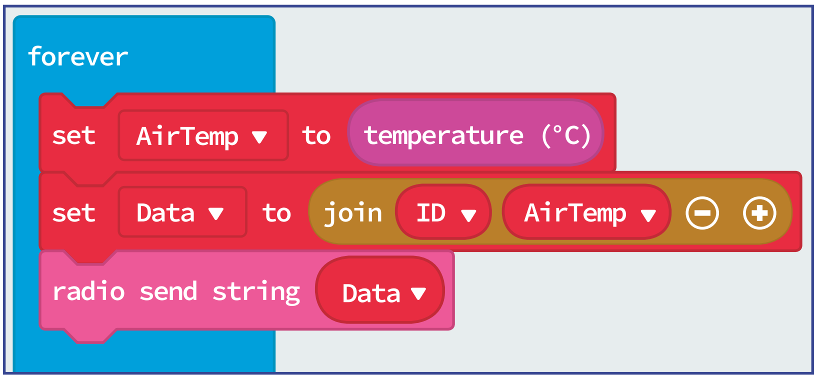
3 Example code 1 (node)

This example sets the radio group to 1, this is important as both the node and the gateway need to be set to the same group or they will not be able to communicate.



A variable is called 'ID' is created to store the node's unique ID.

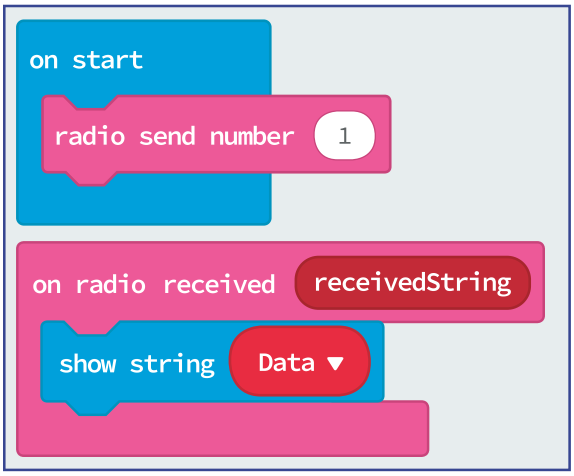
Another variable is created called 'AirTemp' and this is set to the micro:bit temperature sensor reading.



Next, we create another variable called 'Data' to hold the joined (or concatenated) ID and 'AirTemp' variables.

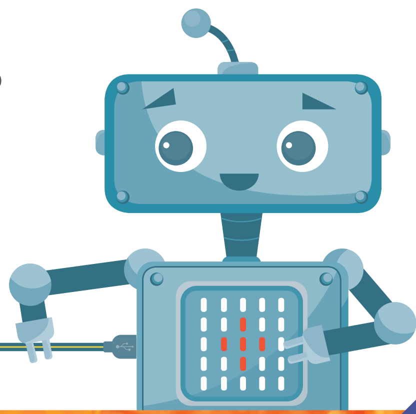
The 'Data' variable is then sent over radio.

4 Example code 1 (gateway)



As with the node, we also need to set the radio group to the same as the node or they will not be able to communicate. (Second "on start" block is not needed in the MakeCode simulator.)

Here, the micro:bit waits to receive the variable 'Data'. It then shows the variable on the LEDs.



5

Graphing the temperature data

Example code 2 (node)

```
on start
    radio set group 1

forever
    set AirTemp ▼ to temperature (°C) ▼
    radio send number AirTemp ▼
```

In this example we again set the node radio group to 1 and will also do the same on the gateway.

Again, we create a variable called 'Air Temp' and set it to the micro:bit's temperature sensor reading.

Then the 'AirTemp' variable is sent over radio to the gateway.

6

Example code 2 (gateway)

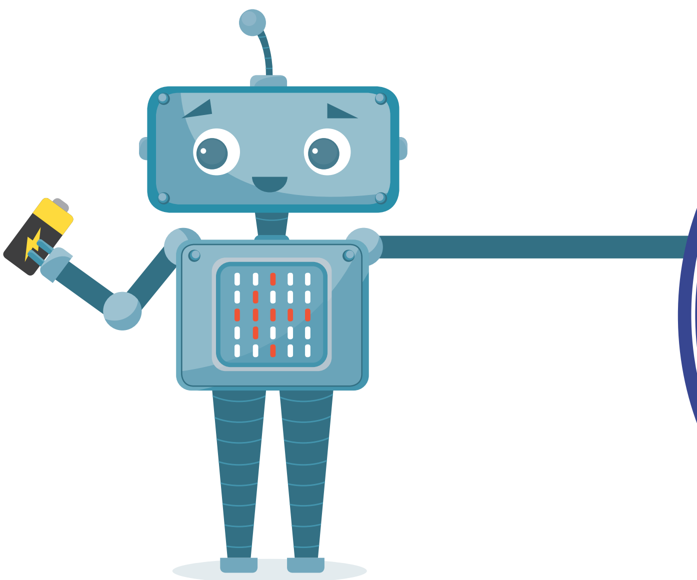
```
on start
    radio send number 1

on radio received receivedNumber
    plot bar graph of AirTemp ▼
    up to 1023
```

As with the node we also need to set the radio group to the same as the node or they will not be able to communicate. (Second "on start" block is not needed in the MakeCode simulator.)

This is where things are a little different!

In this example we take the received variable and plot a bar graph using the LED blocks.

You may need to think about adding some 'pauses' on the node to slow the data transmission rate down to make the batteries last longer!



PRO *TIP*

Sending data in real time can use up batteries on these devices quickly and these monitors need to be at sea for long periods of time. Think about how you can send the data from the nodes less frequently but still give a good representation of temperature changes over time.

7

Graphing the acceleration data

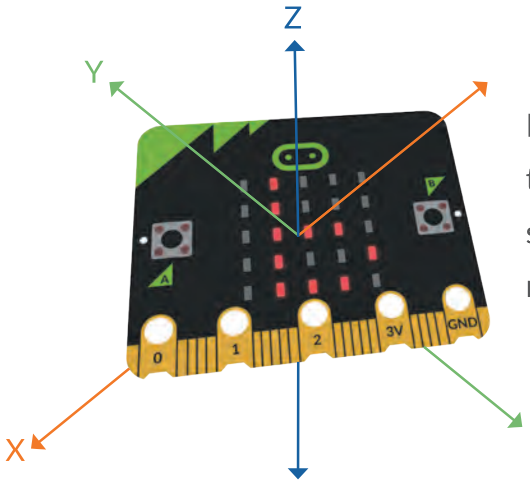
Example code 3 (node)

```
on start
    radio set group 1

forever
    radio send number acceleration (mg) y ▼
```

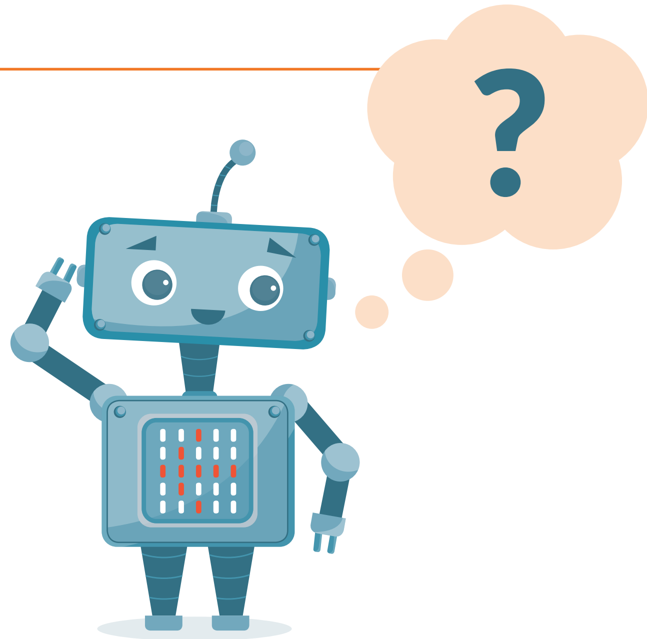
These blocks are set up as in example 2 and set the radio group to 1 (this is also done on the gateway below).

Here the acceleration data is sent over radio. In the previous example we set a variable to the temperature reading and send that. Here the acceleration data is just sent. Do you think this makes a difference?

8



In this example the Y axis data is sent. What is it measuring?



9

Example code 3 (gateway)

```
on start
    radio set group 1

on radio received receivedNumber ▼
    plot bar graph of receivedNumber ▼
    up to 2000
```

As before, the radio group is set to the same as the node so that they can communicate. (Second “on start” block is not needed in the MakeCode simulator.)

The blocks here ‘listen’ for transmitted numbers and then add them to a variable called ‘recievedNumber’.

A graph is then plotted on the LED screen showing a real time graph of the accelerometer data.

Why might this be useful data for the scientists?

10

If you have access to any additional peripherals, you will need some custom extension blocks in MakeCode. To find these blocks you need to click the 'Extensions' tab:

⊕ Extensions

search for the control board that you are using.

Test time!

⚠ **BEWARE!** Micro:bits and peripherals are not waterproof! Electricity and water do not mix well, and you can permanently damage the electronics by getting them wet. You can test this prototype on dry land!

Stretch tasks

- Add a separate temperature sensor that can be submerged to measure the temperature at different depths.
- Using the compass on the micro:bit, modify your prototype to provide a wind direction to the gateway. (You will need to do this on a separate micro:bit.)
- Create another node and have them both transmit different data to the gateway.
- Create a user interface (UI) for the gateway to allow the user to see the data from separate nodes.
- Add a LoRa or WiFi transmitter to your prototype and transmit the data over longer distances.
- Add a pH meter to also measure ocean acidity. (You may need to use an analogue to digital signal converter.)
- Rather than transmitting the data, have the micro:bit log the data locally for the scientists to collect. Alternatively, log and transmit the data for redundancy.
- Technology use is a contributor to global warming due to the amount of electricity that it consumes. Research how technology companies like Arm are creating more efficient technologies to help reduce our carbon footprint. How can we make sure our Ocean Health Monitors don't become e-waste?

Final thoughts

This project explores several different computational techniques to gather useful data to help the marine scientists. Can you think of other types of sensors that may help them? What other data could be useful to them across an ocean?

AUTO-FARMER

Getting started

A group of farmers have some farmland that is at risk of desertification (becoming unusable for farming) and have asked you to develop a prototype to control some LED grow lights and irrigation pumps that will help grow plants on the land. The lights and pumps can only be left on or off and the farmers want a system where the turning on and off is automated depending on the wetness/dryness of the land and the light levels to efficiently use water and electricity.

Success criteria

- ☑ Design and build a prototype that uses a light sensor to turn on a grow light when the light level drops below a certain level.
- ☑ The prototype should also use a moisture sensor to turn on irrigation pumps when the soil is dry.
- ☑ The prototype needs to have a ‘kill switch’ to turn off the lights and pumps.

1

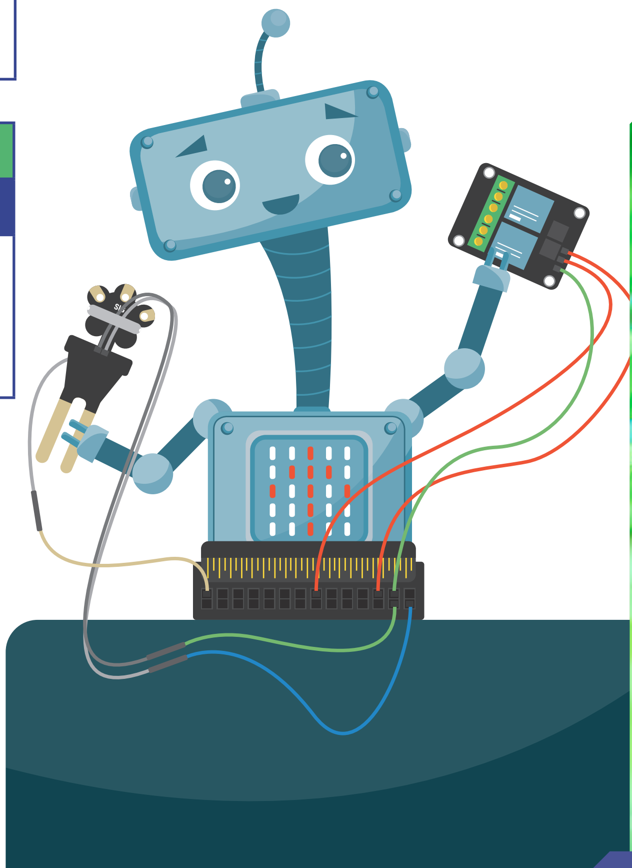
Input, Process, Output (IPO)

Light (LED grow lights)		
Input	**Process**	**Output**
Light level	If light level drops below 45	Turn on light

Water (irrigation pumps)		
Input	**Process**	**Output**
Moisture sensor	If moisture level drops below <user defined value>	Turn on irrigation pumps

Building the prototype

To allow us to interact with irrigation pumps and grow lights, we are going to need to use a relay. A relay is a switch that can be controlled by a microcontroller. We can program the micro:bit to switch the relay on and off given certain conditions (such as light and moisture levels) and the relay will switch on or off the power to the pumps and lights automatically!



2

PINS

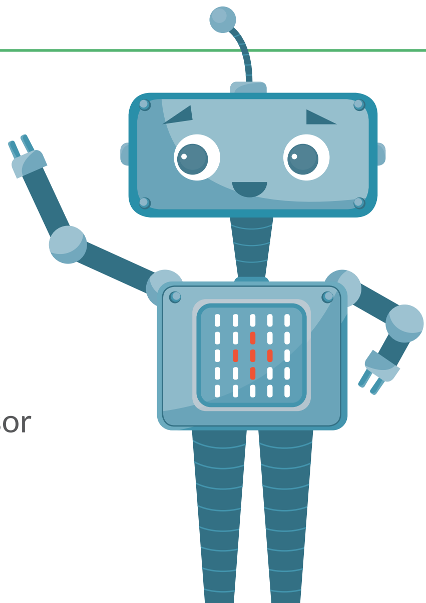
Using the correct pins is important and the table below lists all the available pins on the micro:bit.

Pin	Function 1	Function 2	Description
GND			**Ground for both the relay and the moisture sensor**
GND			Ground
3V3			3.3V
0	Analog In		**Connected to large pin 0 Used for the signal from the moisture sensor**
1	Analog In		Connected to large pin 1
2	Analog In		Connected to large pin 2
3	Analog In	LED Column 1	Controls part of LED array
4	Analog In	LED Column 2	Controls part of LED array
5		Button A	Connected to Button A on micro:bit
6		LED Column 9	Controls part of LED array
7		LED Column 8	Controls part of LED array
8			Open GPIO pin
9		LED Column 7	Controls part of LED array
10	Analog In	LED Column 3	Controls part of LED array
11		Button B	Connected to Button B on micro:bit
12			**Open GPIO pin Used to control relay 1**
13	SCK		GPIO or SPI clock
14	MISO		GPIO or SPI MISO
15	MOSI		GPIO or SPI MOSI
16			**Open GPIO pin Used to control relay 2**
19	SCL		GPIO or I^2 clock
20	SDA		GPIO or I^2 data

3

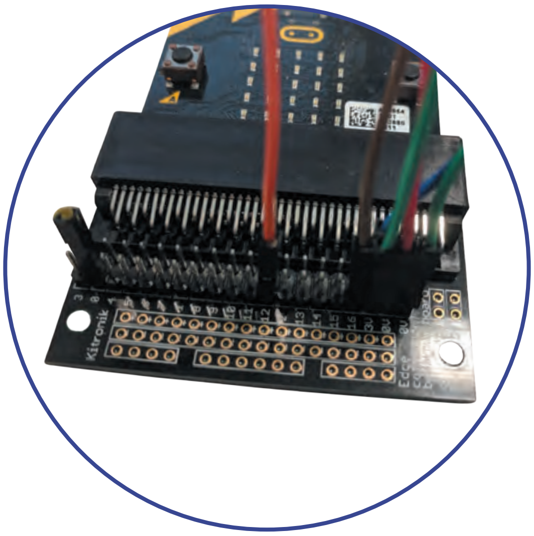
We are going to use:

- Pin 0 for analogue input from the moisture sensor
- Pin 12 for the digital out to turn on relay 1 (IN1)
- Pin 16 for the digital out to turn on relay 2 (IN2)
- Pin 17 (3v) to power both the relay and the moisture sensor
- Ground (GND or 0v) for both the relay and the moisture sensor



Wiring it all up

4

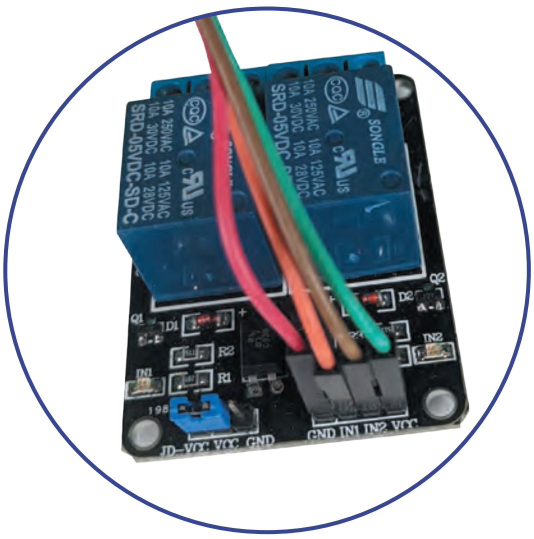


In this image you can see how the micro:bit breakout edge connector is used to connect the header cables to the relay and the moisture sensor.

5

The cables to the moisture sensor had to be doubled up as the connection on the breakout board are male and the connection on the moisture sensor are female. The connection to the moisture sensor is held in place with an elastic band. (This is not ideal, but it works for a prototype and avoids having to solder the connections or use additional header connections.)

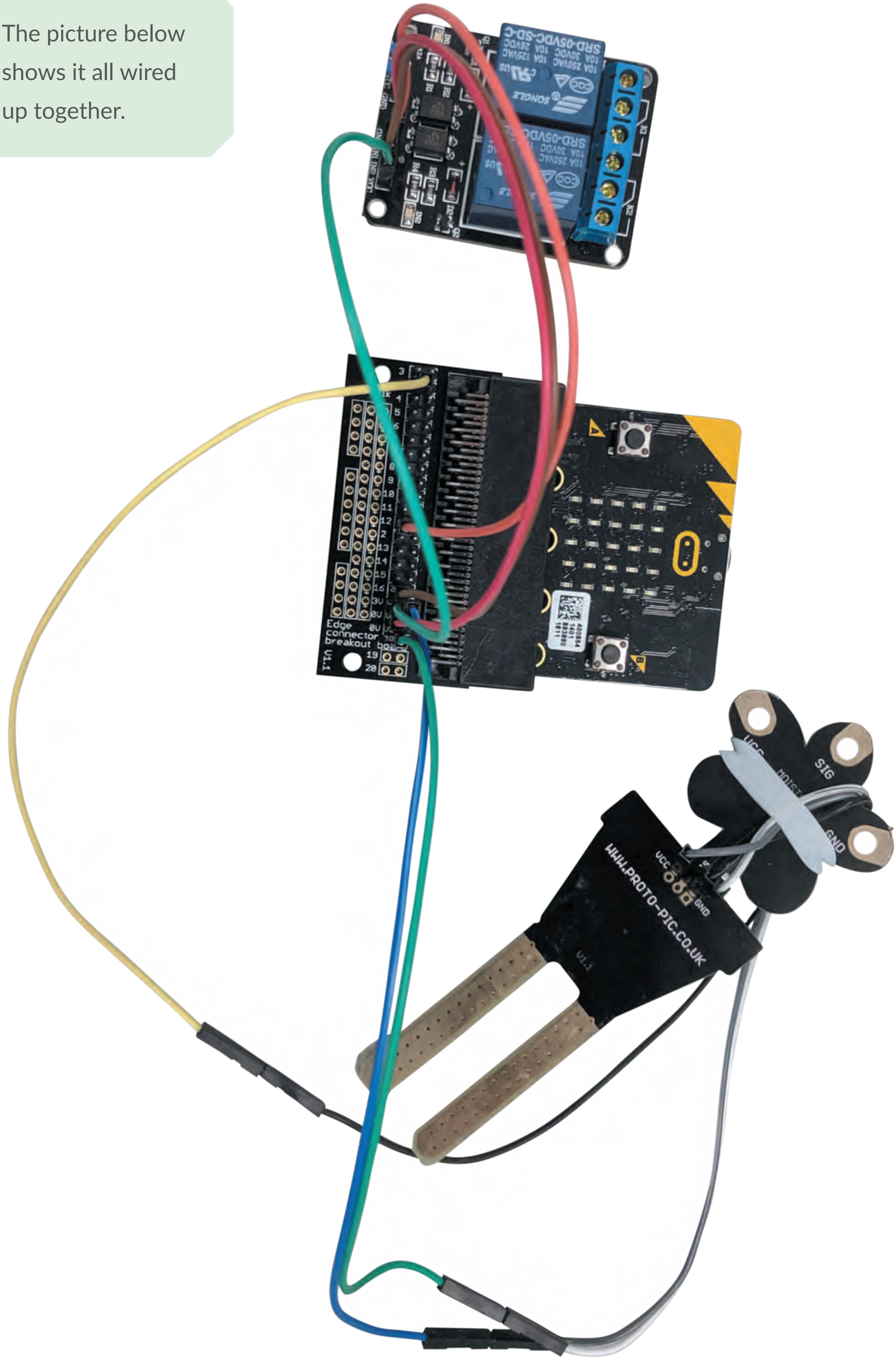
6



Here you can see the header wires on the relay board. Notice the cable colours and how they match the pins above.

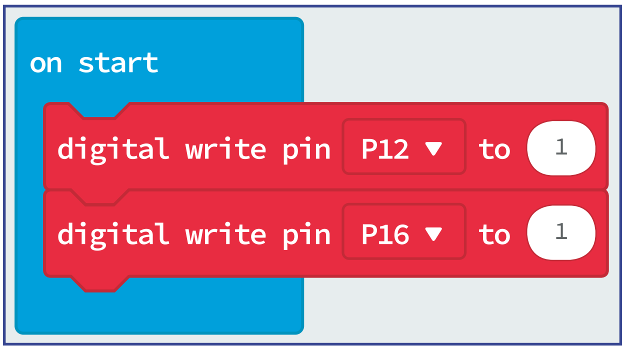
7

The picture below shows it all wired up together.



8

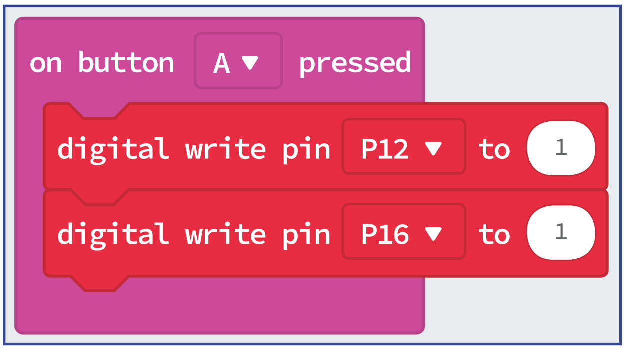
Example code



These blocks set the digital write to 1. This may seem odd as usually 1 = on and 0 = off but the relay being used is 'active low' which means it turns the switch on with a 0 rather than a 1.

This just ensures the prototype starts 'off' when first powered on.

9



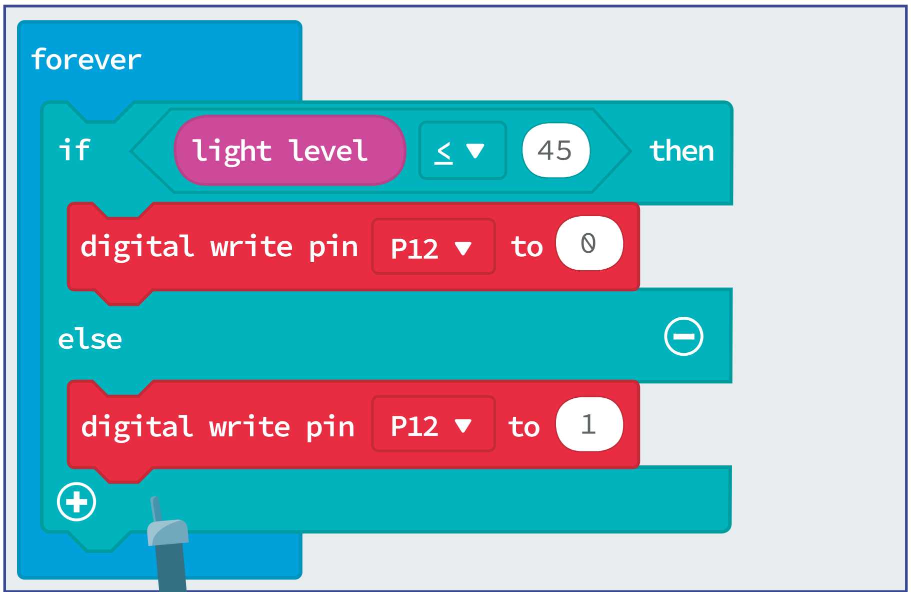
These blocks add the ability to turn off the relay by pressing the A button.

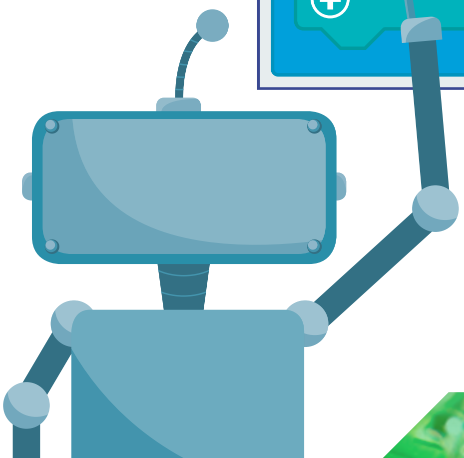
10

These blocks sense if the light level (using the light sensor on the micro:bit) falls below 45, in which case it will turn on pin 12, which would switch the relay switch on, which in turn would power the grow light.

The 'else' section ensures the grow light would be set to off if the light level is too high.

You will need to experiment to see what the light level threshold should be to turn the light on so as not to waste electricity during the day.





11

```
forever
  if analog read pin P0 ≤ 250 then
    digital write pin P16 to 0
    pause (ms) 2000
  else
    digital write pin P16 to 1
    pause (ms) 2000
```

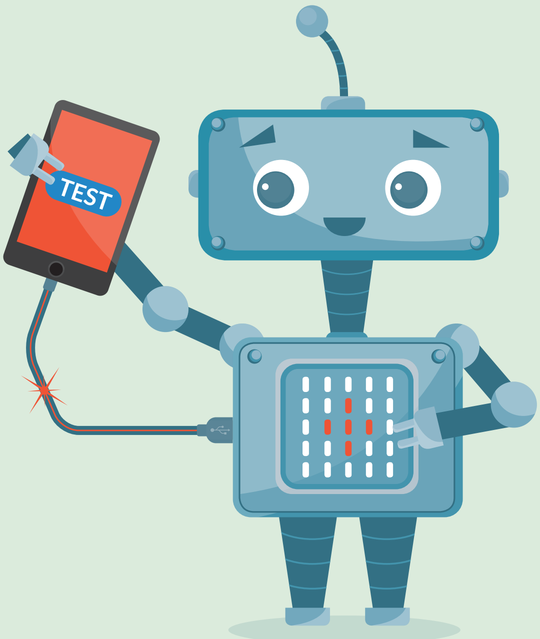
The analogue read pin gives a value between 0 and 1023. The more electricity it senses, the higher the value. If the soil were wet, the water in the soil would conduct a lot of electricity and so the value would be very high. Dry soil would give a low value.

These blocks sense if the analogue read pin value falls below 250 and then turns on pin 16 which would then turn on the irrigation pumps to water the soil.

Pause blocks are needed here or you may find the relay cycles on and off quickly.

Test time!

You will need to test the light and moisture threshold values to make sure that they trigger the relay when they are meant to. The light value worked well under strip lights inside but may not work in actual daylight so this will need to be tested! The moisture values were tested in a cup of water so you will need to test this using some dry soil and some well-watered soil to get the right value. We don't want to waste any water or electricity.



Stretch tasks

- Add in an LED and solenoid to create a fully working prototype.
- Adapt your prototype to use other sensors to control other devices to help the farmers.
- Design a system to automate planting seeds.
- Log all of the data collected locally to allow the farmers to track what works best for their crops.

Final thoughts

This project has some real-world applications as efficient use of water and electricity is important for sustainable farming. Using technology to make things more efficient is a great way to make a difference and to help shape a sustainable future.

OIL SPILL CLEANER UPPER

Getting started

Oil spills do untold damage to eco-systems.

A newly developed material can absorb up to 90 times its own weight in spilled oil and then be squeezed out like a sponge and reused, raising hopes for easier clean-up of oil spill sites.

https://www.newscientist.com/article/2123391-sponge-can-soak-up-and-release-spilled-oil-hundreds-of-times

A group of marine scientists have asked you to develop an algorithm that could be used on a boat drone to drag around a sheet of this smart material to clean up an oil spill.

Success criteria

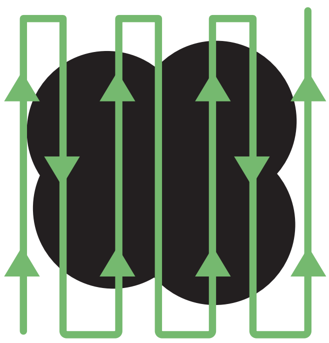
- Build a floating oil spill cleaner upper boat drone that starts with a button press.
- The product should be able to autonomously navigate over an area.
- The product should be made to clean up an oil spill by dragging a 'smart material'.

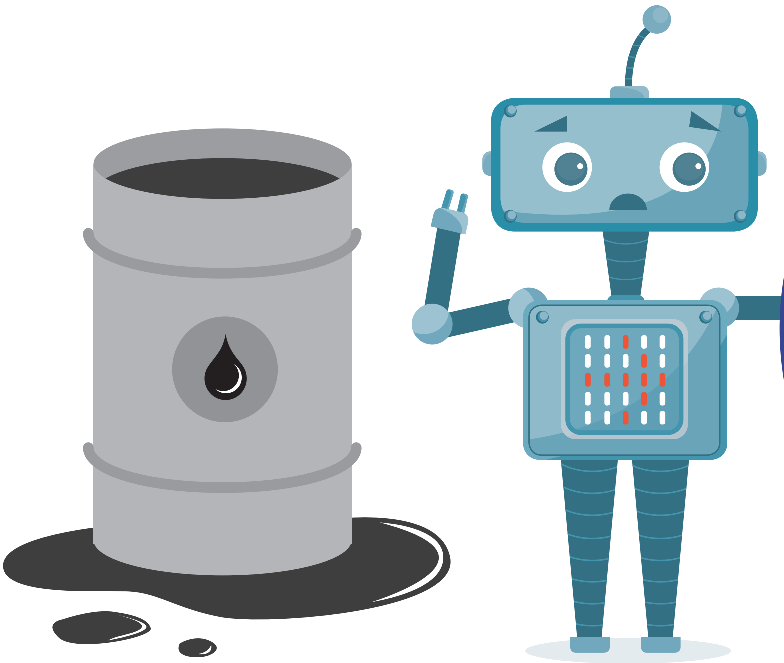
1

Breaking down the problem

The input and output for this problem are simple as the drone boat should start with a button press and should follow a pre-programmed path. Creating the algorithm for the movement is the tricky part and will require some thought.

Input, Process, Output (IPO)

Input	Process	Output
Button press	Algorithm to control the movement of the boat drone in a path to clean up oil: 	Servo motor control



PRO *TIP*

Don't worry about distances at this point. Oil spills can be small or large, and the product just needs to be able autonomously clean an area. Think about how the size of the area can be changed.
We won't have access to any smart material but we can simulate it using a normal sponge.

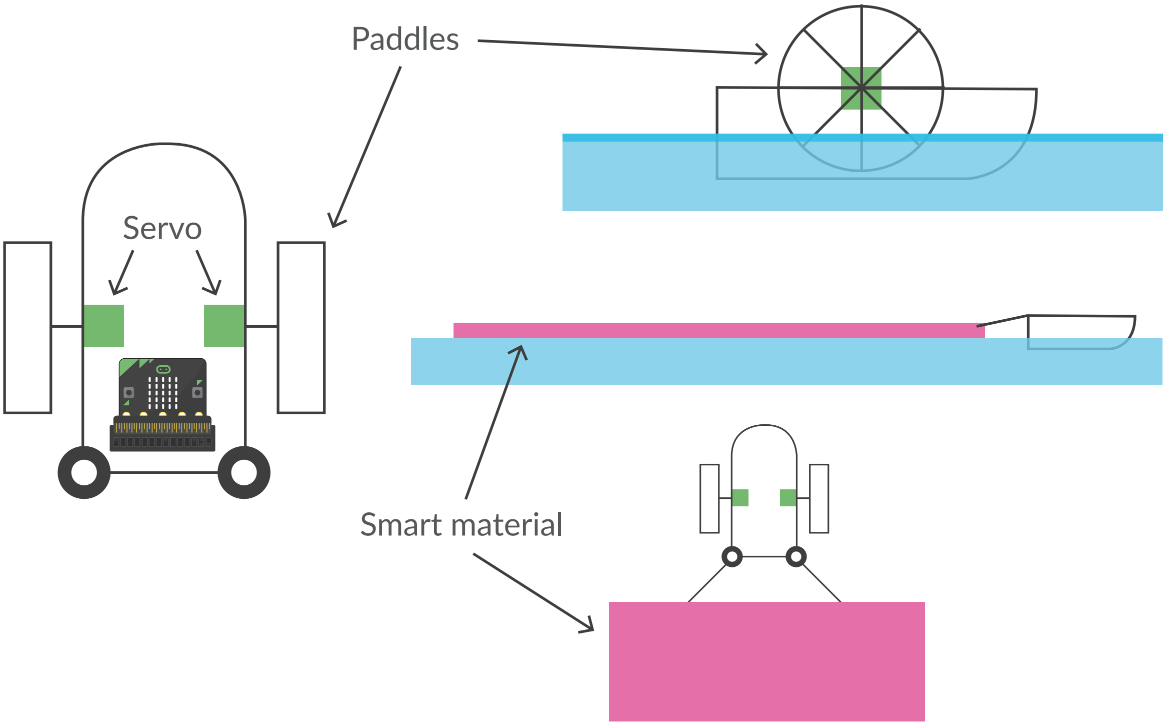
2 Building the product

For this project we need to build a simple boat. You can use anything that is waterproof.

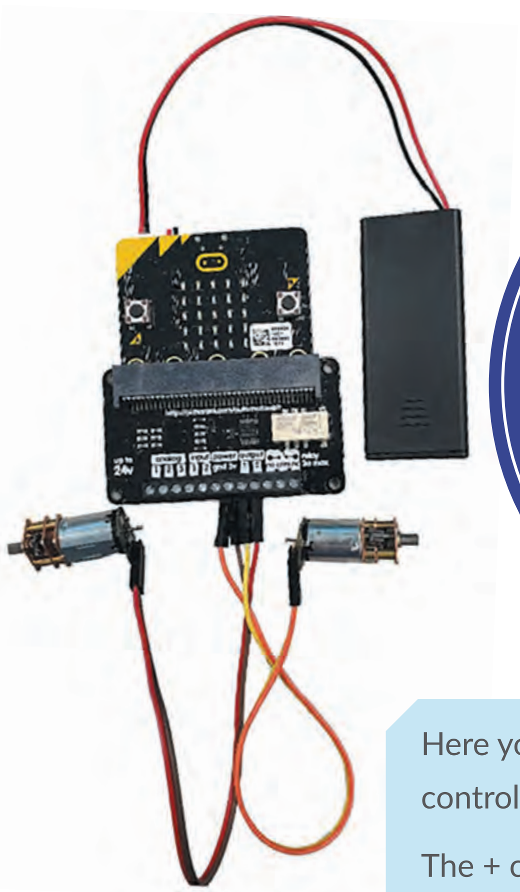
Kit required:

- A micro:bit
- Header wires
- Battery pack
- Boat building materials
- A foam sponge
- A mini screwdriver
- A servo driver board

There are many types of servo controller boards for micro:bit, in this example an 'automation bit' was used.

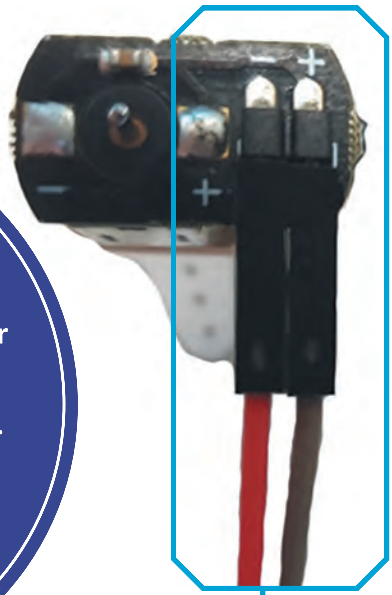


3



PRO *TIP*

There are several extension boards that can be used to power motors and you can power one directly from the micro:bit's pins. If you cannot find this extension board, there will likely be several other options available that do the same thing.



Here you can see how the servo motors are wired to the servo controller and micro:bit.

The + cable from **both** the servos need to go into the 3v opening on the servo control board.

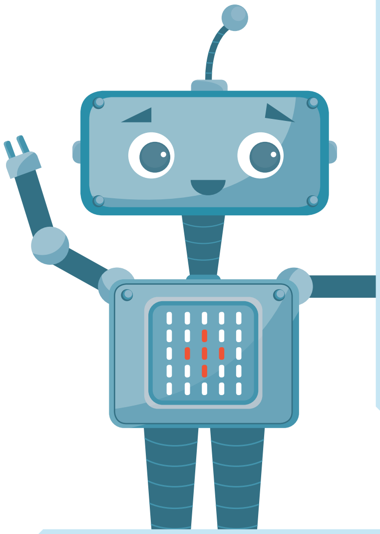
The – cable needs to go into output 1 and 2 respectively.

Pay attention to which side you put them on. In this image:

Output 1 = Right →

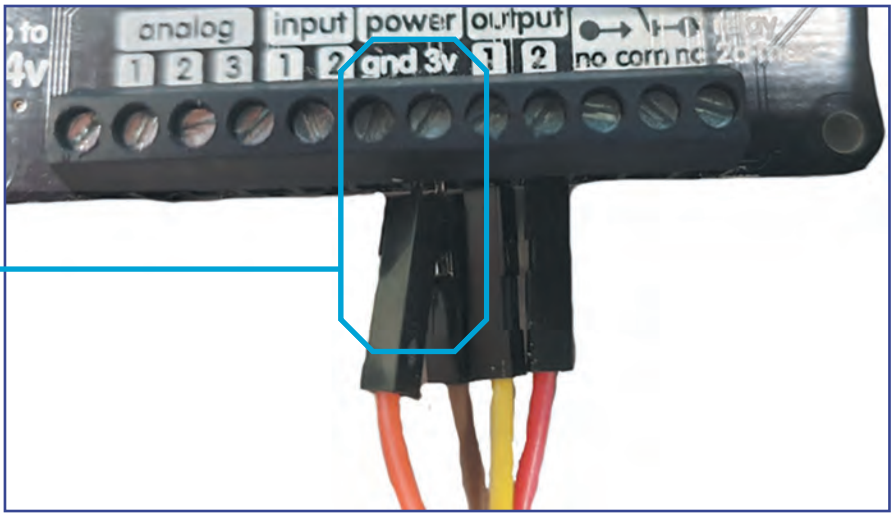
Output 2 = Left ←

You may have a third cable for the servo which is the ground (GND), attach this to the GND terminal on the board if you have this.



Here you can see the two + cables from the servos going into the same 3v terminal.

Other servo control boards may have more than one 3v terminal and so these should be separated if you can.



4

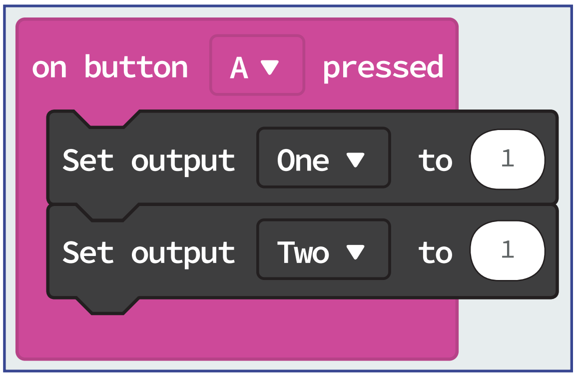
Example code

In this example some custom extension blocks were used. To find these blocks you need to click the 'Extensions' tab and then search for the control board that you are using.

In this example an 'automation bit' was used, but other servo control boards will also work.

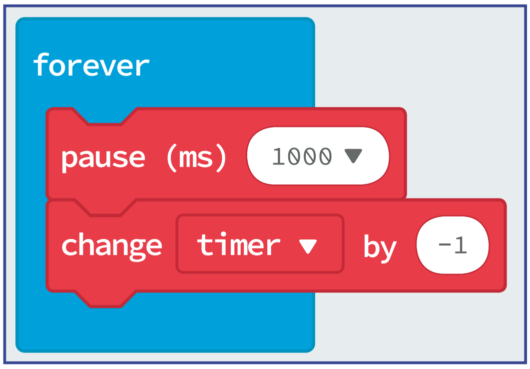
5

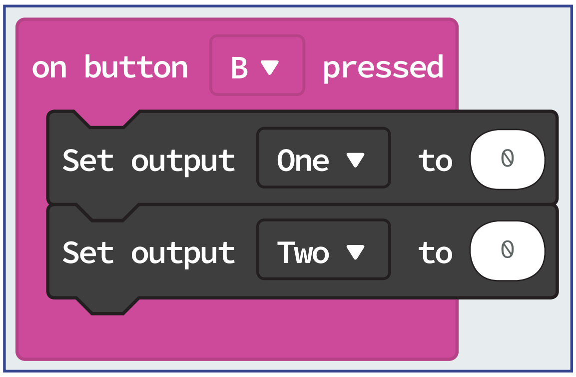
To start with we will create the first part of the algorithm that turns the servos on when a button is pressed:



This simply sets each of the outputs to 1 (on) once button A is pressed.

To get the servos to stay on for different amounts of time we need to create a timer. In this example we use a 'forever' block to change the value of a variable by -1 every second. We can then count down from any number we like by setting the variable to however many seconds we need and then doing something when it reaches 0.

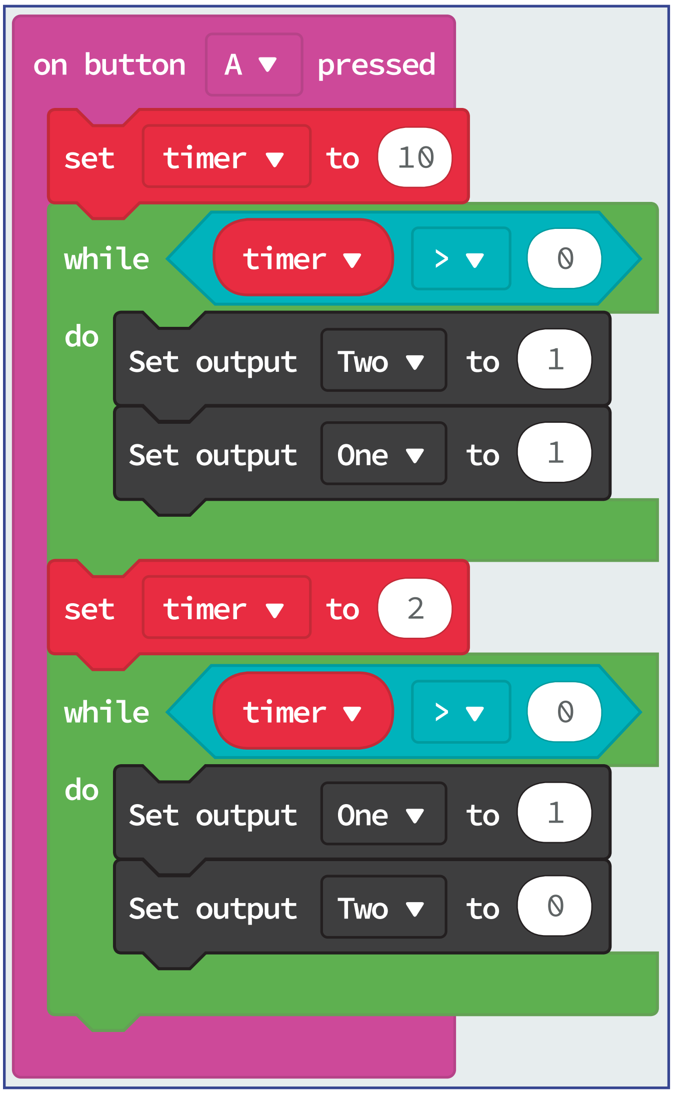




In this example we set the button B to set the outputs to 0 so that we can use it to turn of the robot drone.

We could however use the A/B reset button to do this, but we may need this button for something else later.

6



In this example we use the button A as the trigger to set the timer variable to 10.

We then use a while loop to check if the timer variable is greater than 0, if it is, it turns both outputs on. This would give us 10 seconds of forward motion for the boat.

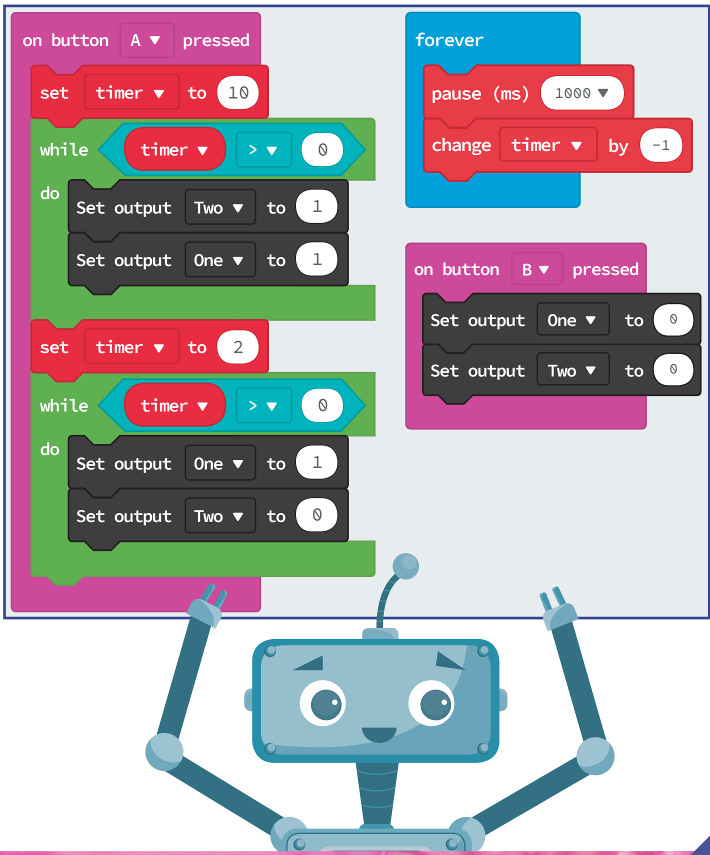
Once the timer variable reaches 0 we set the timer variable to 2 (to give us 2 seconds) and then use another while loop to check if the timer variable is greater than 0. If it is, then it sets only output 1 to 1 (on) so the drone boat will turn to the right.

You will need to experiment with how many seconds it takes to turn 90 degrees.

7

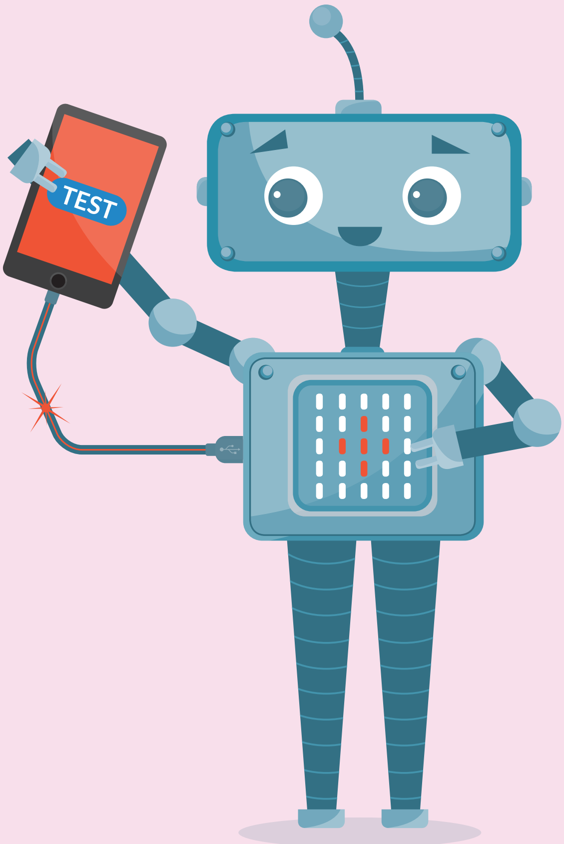
This set of blocks shows the first few steps of the algorithm that automates the oil spill clean-up.

You can use this as a starting point and adapt it to meet the success criteria.



Test time!

BEWARE! *Do not test this in water. Micro:bits and peripherals are not waterproof! Electricity and water do not mix well, and you can permanently damage the electronics by getting them wet.*



Stretch tasks

- Adapt the program so that the navigation is done using the micro:bit's compass so that it turns precisely 90 degrees and can stay on course more accurately.
- Add a moisture sensor so that the boat drone only starts cleaning when in the water.
- In large oil spills, many drones would be used at once. Adapt your programme so that the boat drones don't collide with each other. (You could use the radio blocks for this.)
- Adapt your program so that the 'smart material' is dragged by two drone boats and the smart material is in a long thin sheet. This helps the oil to be squeezed out of it more easily.
- Adapt your program so that the boat drones can return to a 'base' where the smart material can be wrung out and then re-used.
- Adapt your program so that you can control the boat drones direction of travel remotely using another micro:bit.

Final thoughts

Combining smart material and autonomous drones is just one way that technology can help protect the environment. Can you think of other ways that technology can help?

This project is all about cleaning up the environment. The technology we are using to do so can easily become e-waste and add to the problem, unless we ensure it is sustainable and designed to last. Think about how lots of technology is designed to fail after a short time so you have to buy it again, how can we ensure our products don't add to the problem?

TREASURE HUNT

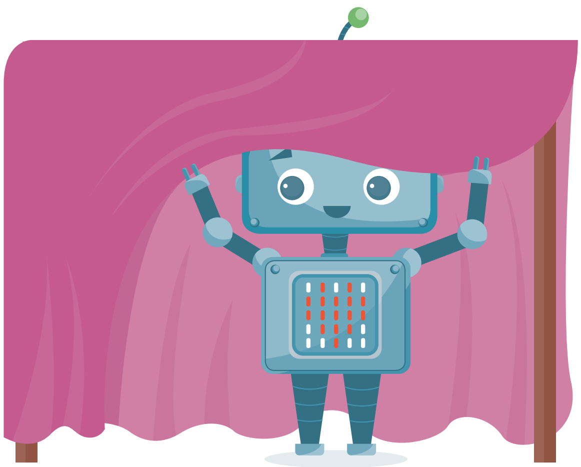
Setting the scene

For this challenge there are three sub-challenges and so you must work in teams. Each team must make:

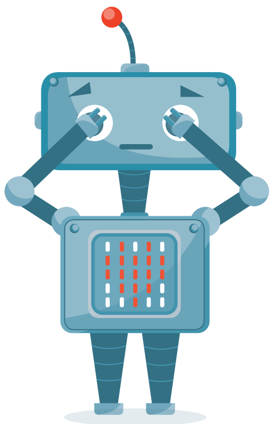
1

A treasure clue beacon

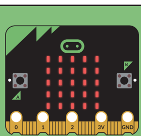
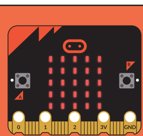
This will be a micro:bit that is hidden by your team and transmits a radio signal so that a 'beacon finder' can find it. The beacon must also transmit a secret code (a single letter) when activated by a passphrase that will be written/printed on a card next to the micro:bit in the hiding place.



Beacon
(hidden somewhere)

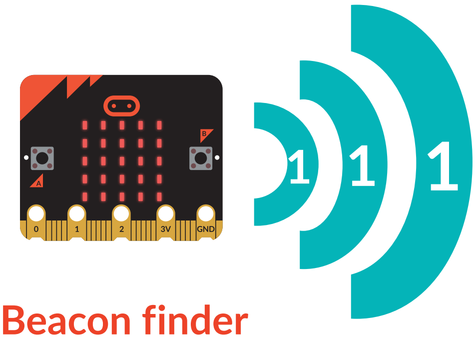


Beacon finder
(with hunter)

 Beacon 1	 Beacon finder
Listens for 1	Transmits 1
When receives 1 transmits password letter	When receives password letter shows on screen

2 A beacon finder

This will be a micro:bit which is programmed to direct the user towards the clue beacons and can be used to get the secret codes needed to unlock the treasure chest.



Beacon finder
(with hunter)

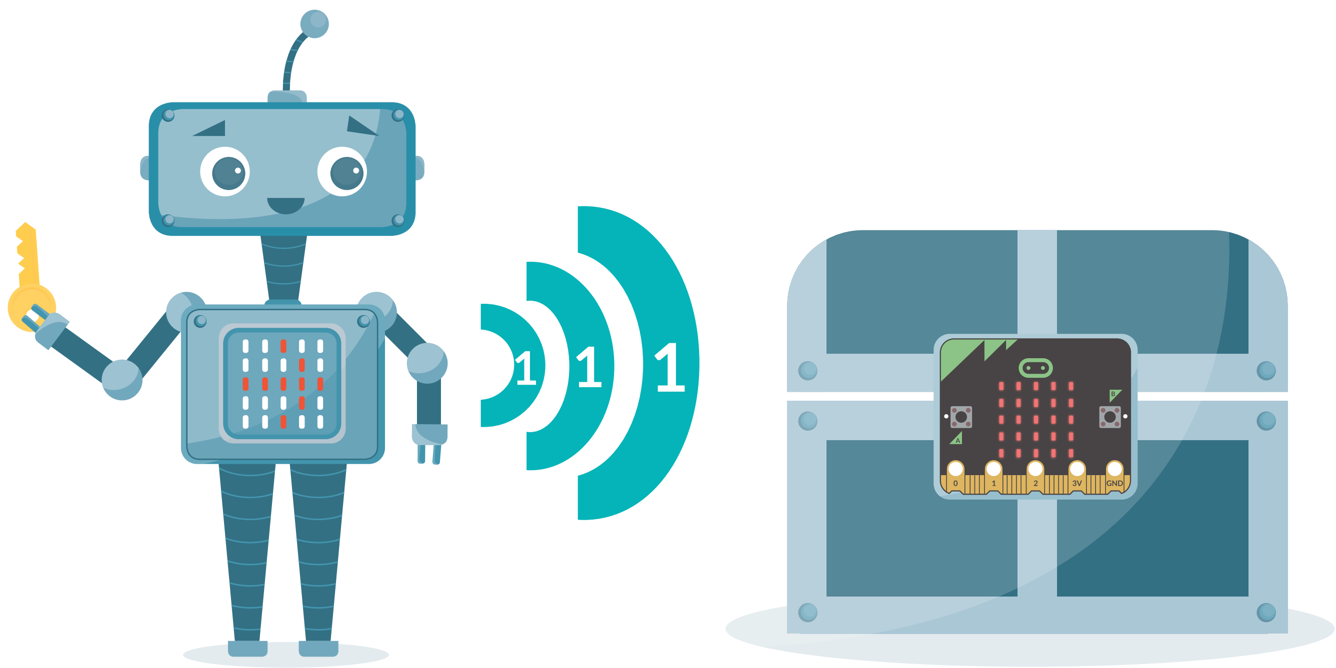
- The beacon finder will transmit the number of the beacon. (This will be on the beacon.)
- The beacon will take this and transmit a letter of the password for the treasure chest.
- Each beacon will transmit 1 letter.
- Each letter will make up the password.
- The password will be used by the key to open the chest.

3 A treasure chest

The treasure chest must be a container which holds the ‘treasure’ and is unlocked when a secret code is transmitted to it. The secret code will be made up of the individual codes transmitted from the clue beacons.

4 A chest key

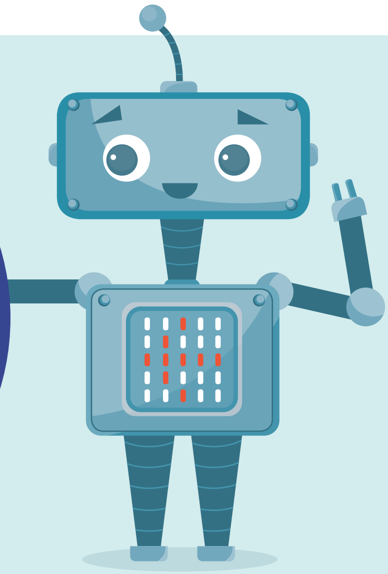
You will need to create a key (a micro:bit) that will transmit the secret code to the treasure chest.

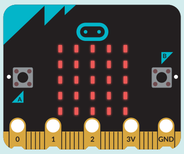


5

PRO *TIP*

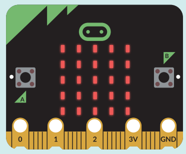
You will need one beacon per letter of the password. The example below shows how the letters of the password are transmitted by the beacons.





Beacon 1

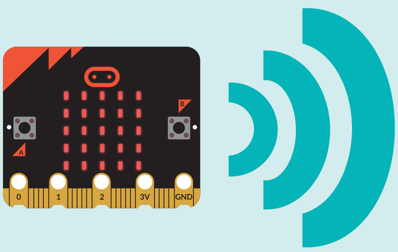
Group ID	Beacon code	Password letter
1	1	A



Beacon 2

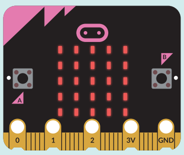
Group ID	Beacon code	Password letter
2	2	C

Beacon codes can be printed on beacons!



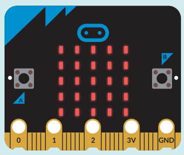
Group ID	Beacon code
1	1
2	2
3	3
4	4

The password in this example is **CATS**



Beacon 3

Group ID	Beacon code	Password letter
3	3	S



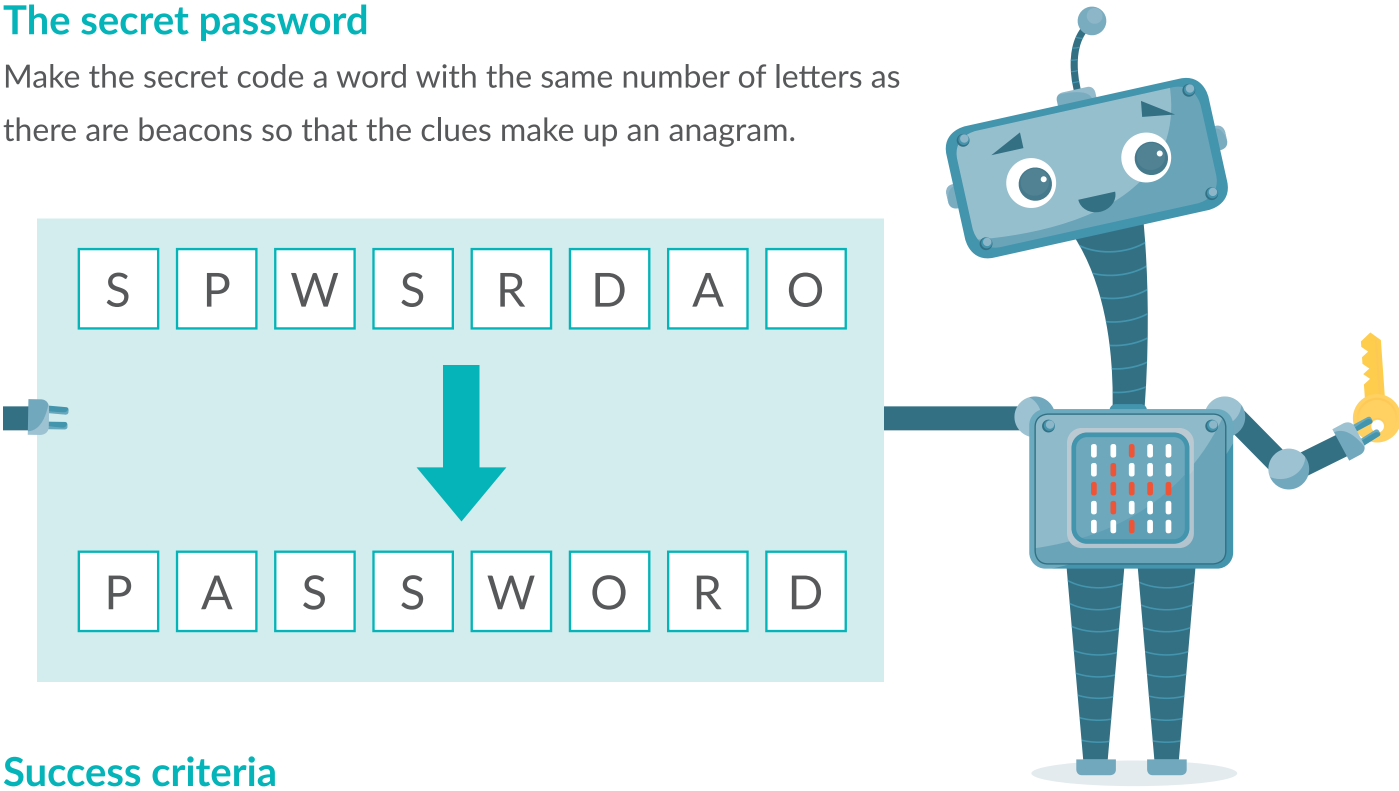
Beacon 4

Group ID	Beacon code	Password letter
4	4	T

6

The secret password

Make the secret code a word with the same number of letters as there are beacons so that the clues make up an anagram.



Success criteria

Beacon	Beacon finder
☑ Each beacon must transmit a letter of the password on group ID 128 with a transmit power of 2 once it receives the correct number on the appropriate group ID. ☑ The beacon must display its beacon number constantly.	☑ Each beacon finder must transmit a number on a group ID with the same number (128) with a transmit power of 2. ☑ The beacon finder must help the user find the beacon. ☑ The beacon finder must transmit the correct number to the beacon. ☑ The beacon finder must store the password letter as a variable.

Treasure chest	Key
☑ The treasure chest should listen to group ID 255 and open when it receives the correct password made up of the password letters transmitted by the beacons. ☑ The treasure chest should open using a servo/motor. ☑ The chest should look like an authentic treasure chest.	☑ The key must be able to receive the complete password and then transmit it to the chest on group ID 255 with a transmit power of 2. ☑ The key must look like a key.

7 Some ideas

Make the secret code a word with the same number of letters as the number of beacons so each beacon transmits one letter that make up an anagram that is the secret word that opens the treasure chest. If an anagram is too hard then the beacons can be numbered so the letters are already in the right order.

8 Design

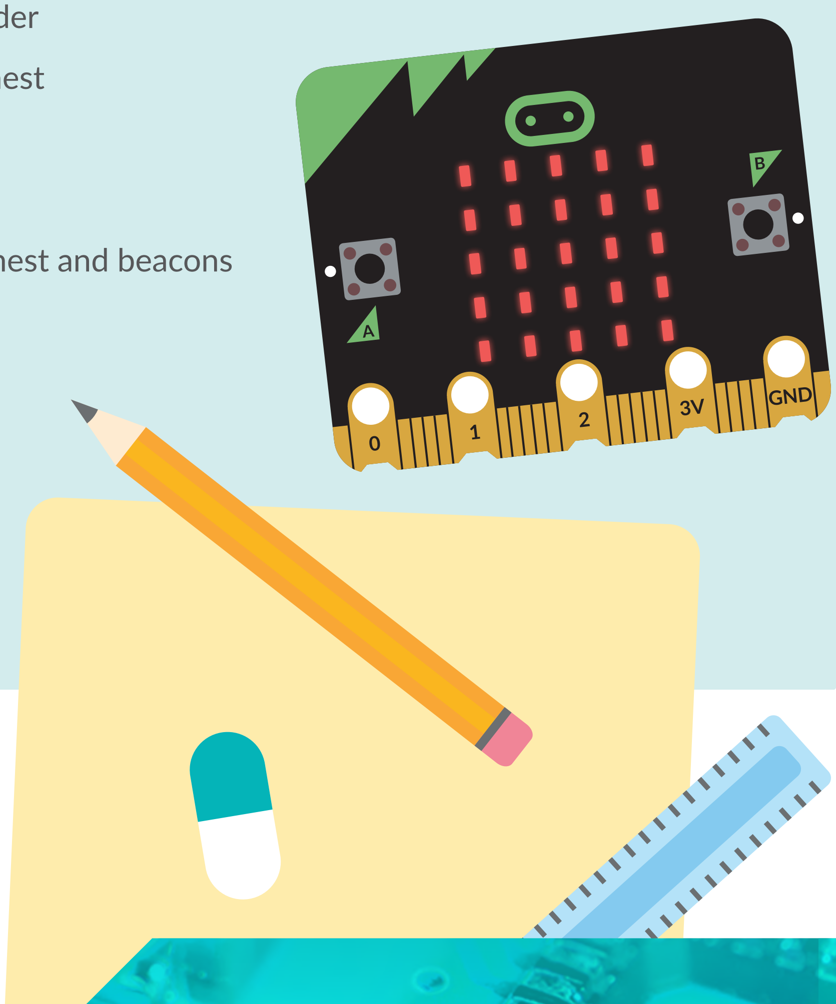
The most important thing to remember is to be creative and come up with something novel that meets the success criteria in an interesting way. Think about the needs of the user and think about how they will interact with the project and what they would expect the product to do.

Stretch tasks

- Use a Caeser Cypher to encrypt the password.
- Create a digital 'lock pick' for the chest.
- Try to work out the password computationally using only some of the letters.
- Once opened, the chest should use light and sound to celebrate!

Equipment needed

- A micro:bit for each letter of the password (~4 is ideal)
- 1 x micro:bit for the beacon finder
- 1 x micro:bit for the treasure chest
- 1 x micro:bit for the key
- **~7 in total**
- Making resources for the key chest and beacons
  - Card
  - Paper
  - Pens/pencils/colours
  - Etc.
- A servo/motor for the chest
- Some treasure!



GETTING STARTED WITH MICROPYTHON

MicroPython allows you to use all the power of Python on the micro:bit. Python is a high level text based language that is used commercially across small, medium and global companies such as Google.

In this guide we will use Python code and it will look like this:

```
from microbit import *
display.scroll("This is what code looks like")
```

To write your code you will need to use an interpreter. You can use an online one here:

https://python.microbit.org/v/1

The interface looks like this:

Important things to remember

Making the leap into using a "proper" programming language is exciting but you need to be careful when typing out the code as any mistakes will make MicroPython throw an error. MicroPython is case sensitive so python and Python are treated as different things. Typing out the code carefully is very important as any typos or missing quotation marks will again throw errors. MicroPython usually tries to tell you what line the error is on but it isn't always completely accurate.

QUICKSTART MICROPYTHON

micro:
project

1

Let’s dive straight in (if you haven’t already) and code our first program. Open the interpreter (https://python.microbit.org/v/1) and you should see the following code:

```
from microbit import *

while True:
    display.scroll(‘Hello, World!’)
    display.show(Image.HEART)
    sleep(2000)
```

2

Let’s look at what each part does:

```
from microbit import *
```

This imports everything MicroPython needs to work with the micro:bit.
(* means "everything", which means all the functionality within Python for working with micro:bits)

```
while True:

        display.scroll(‘Hello, World!’)
```

`while True:` means do whatever comes after this (the indented bit) forever while ‘something’. The ‘something’ in this example is `True`, and `True` will always be `True` so it will run forever! You can add conditions here like

`while x > 5:` but we’ll come to that later.

The next line `display.scoll(‘Hello, World!’)` (as you have probably guessed) displays and scrolls the message ‘Hello, World!’.

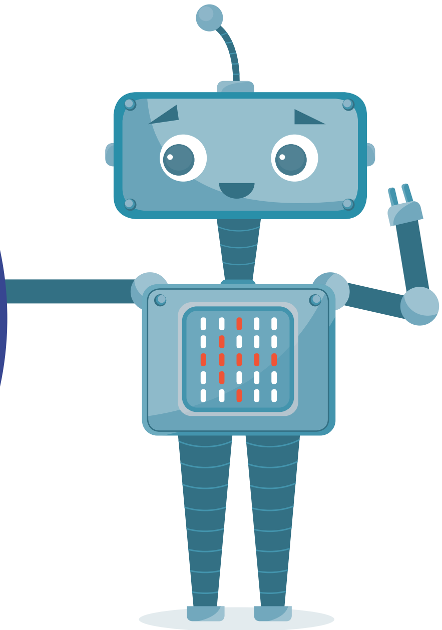
`display.show(Image.HEART)`	The next line is slightly different. It still displays something, but this time it shows it instead of scrolling it. Also, this time it is displaying an image **(Image.HEART)** instead of text, as previous.
`sleep(2000)`	The **sleep(2000)** leaves a gap between the loops of 2000 milliseconds (2 seconds). Then the loop starts again.

Did you notice that after the **display.scroll** there were brackets () and speech marks "" inside them? These are important as they tell MicroPython that what is between the bracket and speech marks is text (or 'string' as it is also called in programming).

PRO *TIP*

When typing these, open and close the brackets and speech marks before adding the contents so that you don't forget to close them. So you would type (' ') and then type the string in the middle. You can use ' ' or " " as long as you are consistent.

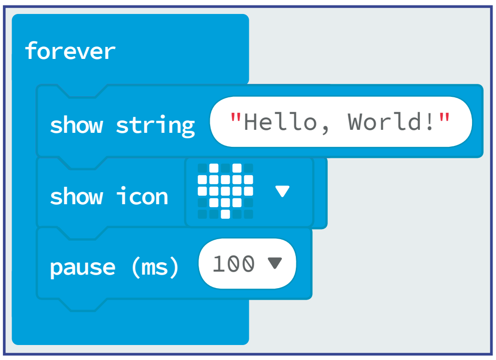
*****Some Python interpreters close them for you automatically!*****



3 Indentation

You may have noticed that all the lines after **while True:** are indented four spaces. This is done to help the reader see what code sits within other code, much like in MakeCode where blocks sit inside each other.

```
while True:
    display.scroll('Hello, World!')
    display.show(Image.HEART)
    sleep(2000)
```



4 Activity:

- Edit the code to scroll your name
- Edit the code to have a longer or shorter `sleep`
- Edit the code to show any other image from these built-in images:

```
Image.HEART          Image.FABULOUS       Image.SQUARE_        Image.TSHIRT
                                           SMALL
Image.HEART_         Image.MEH                                 Image.
 SMALL               Image.YES            Image.RABBIT          ROLLERSKATE
Image.HAPPY          Image.NO             Image.COW            Image.DUCK
Image.SMILE          Image.TRIANGLE       Image.MUSIC_         Image.HOUSE
                                           CROTCHET
Image.SAD            Image.                                    Image.TORTOISE
Image.CONFUSED        CHESSBOARD          Image.MUSIC_         Image.BUTTERFLY
                                           QUAVER
Image.ANGRY          Image.DIAMOND                             Image.
                                          Image.MUSIC_          STICKFIGURE
Image.ASLEEP         Image.DIAMOND_        QUAVERS
                      SMALL                                    Image.GHOST
Image.                                    Image.PITCHFORK
 SURPRISED           Image.SQUARE                              Image.SWORD
                                          Image.XMAS
Image.SILLY                                                    Image.GIRAFFE
                                          Image.PACMAN
                                                               Image.SKULL
                                          Image.TARGET
                                                               Image.UMBRELLA
                                                               Image.SNAKE
```

5 Make your own image

You can also create you own images and give them names to use later. To do this you need to tell the micro:bit which LEDs you want to be on and which need to be off. Unlike in MakeCode we don't have a simulation to draw our image onto.

The micro:bit's LEDs are arranged in a 5x5 grid so to make this easier we can arrange our code to match. Below is the code needed to make your own images. Follow along and make sure you type everything out carefully.

6 Plan your image

Plan your image using the **MicroPython design sheet**. Have a go at a few different designs and try using the different brightness values from 0 – 9. Give your image a name underneath the grid as this will help later on.

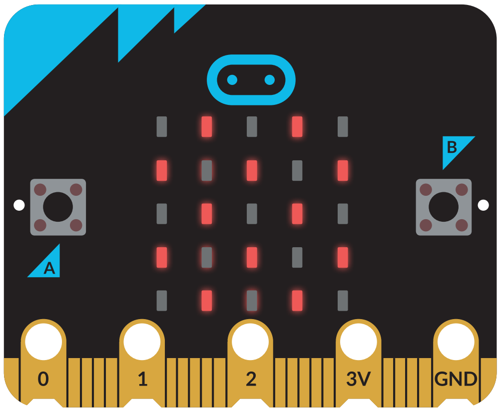
7 Programming your image

```
pattern1 = Image(“:”
                 “:”
                 “:”
                 “:”
                 “”)
```

```
from microbit import *

pattern1 = Image(“09090:”
                 “90909:”
                 “09090:”
                 “90909:”
                 “09090”)

display.show(pattern1)
```

Here we are creating a variable called `pattern1` and we are telling MicroPython that it is an image. We need to specify 5 rows of 5 values (one for each LED). The value can be from 0–9, 0 being off and 9 being the brightest. The code here won't do anything as there are no values, it is empty to show you the structure of the code. We will add the values below.

We then need to tell it which LEDs we want on and off.



Having the code like this makes it easier to read for us but you could have it all in one line like this:

```
pattern1 =
Image(“09090:90909:09090:90909:09090”)
```

Notice how there are fewer `“”` as all the code is on one line.

8 Variables

A variable is a container for a value, like a number we might use in a sum, or a string that we might use as part of a sentence. One special thing about variables is that their contained values can change.

```
my_str = (“Hello”)
my_int = 35
display.show(my_int)
```

A variable can contain anything. Here we have variable `my_str` which holds the string “Hello” and another called `my_int` which holds an integer (number). This would show 35 on the screen.

Stretch tasks

- Try the above code yourself.
- Give the variable `(pattern1)` a different name.
- Make your own image with your own name.
- Experiment with different brightness values (0–9) to get different effects.

9 Getting animated

Next we will create a series of images to make a simple animation and then display them to "play" the animation on the micro:bit. Creating the images that make up the animation is the same as making one but you do more of them!

```
from microbit import *
pattern1 = Image("09090:"
                 "90909:"
                 "09090:"
                 "90909:"
                 "09090")
pattern2 = Image("09090:"
                 "90509:"
                 "05050:"
                 "90509:"
                 "09090")
pattern3 = Image("05050:"
                 "50905:"
                 "09090:"
                 "50905:"
                 "05050")
pattern4 = Image("90909:"
                 "09090:"
                 "90909:"
                 "09090:"
                 "90909")
all_patterns = [pattern1,
pattern2, pattern3, pattern4]
display.show(all_patterns,
delay = 200)
```

Here we have created 4 separate images with some slight differences in the LED values to create an animation. Just copy and paste your first image and change the numbers to speed things up.

You may find it easier to plan your animation first You could use a copy of the planning grids on page 74 or even use a spreadsheet.

After the images we create a list called `all_patterns` and we add all the patterns (1–4) to the list. You can tell it is a list as it uses square brackets `[]`.

Then we display the list with a slight delay.

10 Lists

Lists are exactly what they sound like, they are lists of things. You can add anything to a list and even mix them up.

```
while True:
    if button_a.is_pressed():
display.scroll("A")
```

Then we can display the list after casting (changing it) it to a string data type.

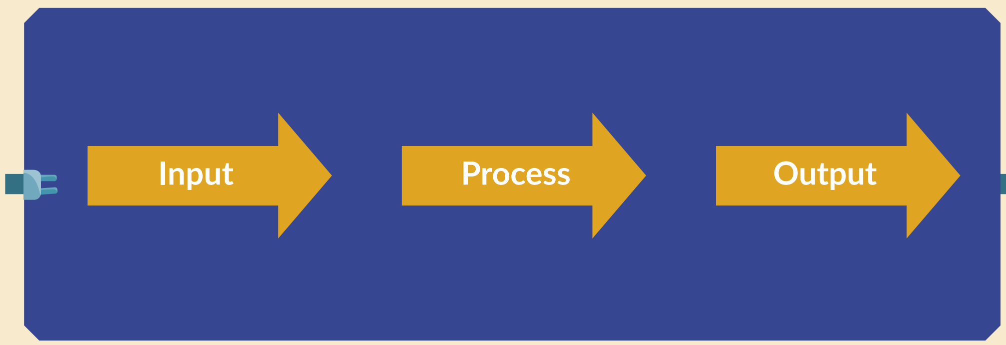
Did you notice that the numbers don't require speech marks (“”)? Numbers don't need them as they represent a value. The **“35”** is a string and so maths can't be performed with it, but can be with **35**. This is because a string and an integer are different **data types**.

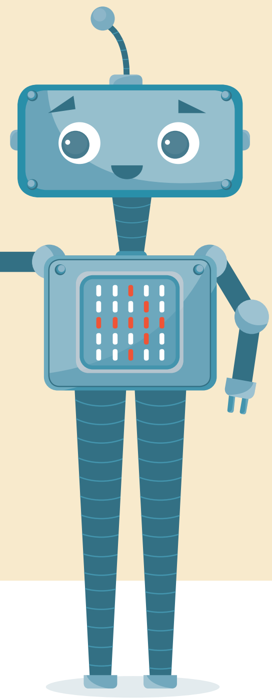
Stretch tasks

- Create your own animation of at least 6 images in a list.
- Experiment with LED brightness values to create effects.
- Make your animation loop continuously.

Final thoughts

So far we have programmed some outputs (LEDs). Next we will look at some inputs (buttons) that can be used to trigger the outputs. This demonstrates the Input, Process, Output model where an input, such as a button being pressed on the micro:bit triggers a process, such as creating a variable with a value (like in the multiplication revision app) that generates an output, such as scrolling the value on the LEDs.



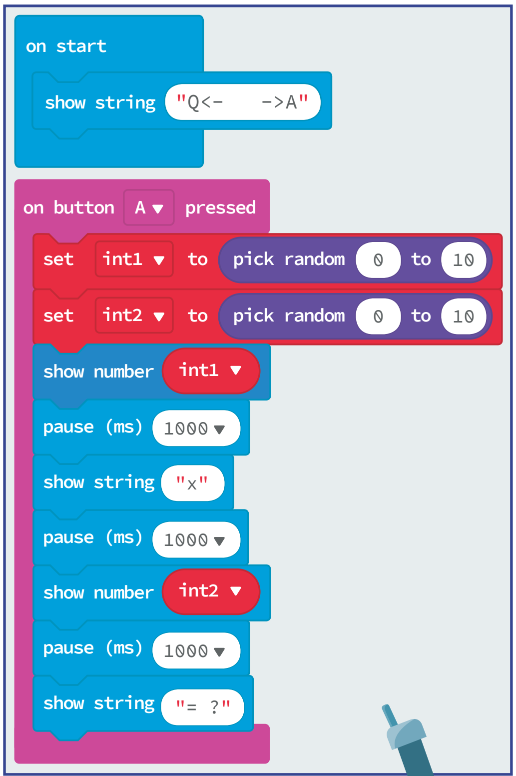


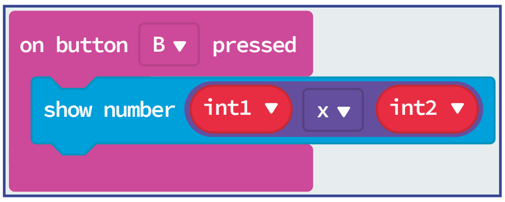
This process can describe many of the activities you will carry out on the micro:bit and it will help you plan your own programs.

MULTIPLICATION REVISION APP 2.0

using MicroPython

1



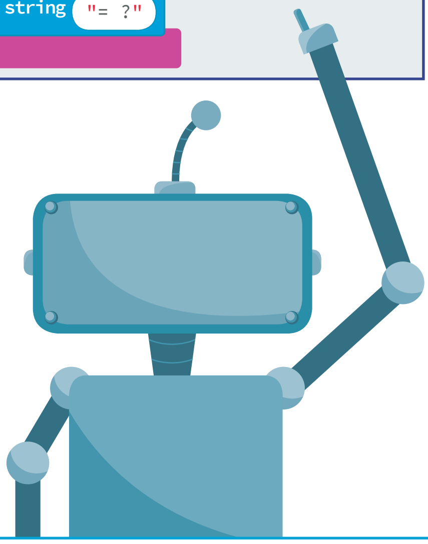


We are going to follow the same process as we did for the MakeCode version at the start of the book, but this time with MicroPython. To remind you of the program, the blocks are on the left.

We are going to replicate this program in MicroPython code.

Before we dive in we must understand some other programming tools:

- while loops – we have seen these before in the Quickstart project
- if, else if, else – also known as selection
- buttons as inputs
- random numbers (functions and modules)



2

Buttons, selection and loops

One of the simplest inputs on the micro:bit are the buttons. We have already used them in some of our programs but we will now look at how to program them in MicroPython. To get a button to work using code we need to use a `while` loop. A `while` loop is a test to see if a condition is met, if it is then it runs the code that comes after it. We use a `while True:` in the same way as the **forever** block in MakeCode.

```
from microbit import *
while True:
    if button_a.is_pressed():
        display.scroll("A")
```

This code waits for button A to be pressed and then scrolls **A** across the screen when it is.

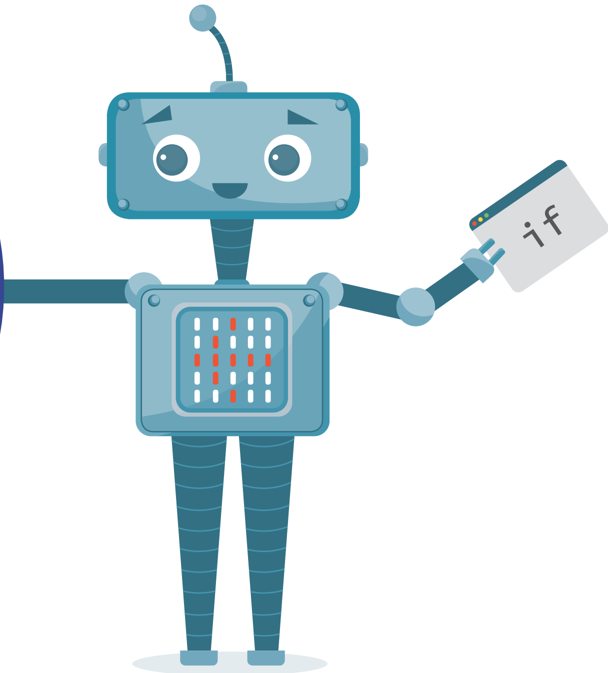
3

In this example we are testing whether `True` is true, which it always will be, so it then moves onto the next line of code where we test `if` button A is pressed, and if it is we display 'A'. You can test for any condition such as while a button is pressed or while a variable equals a number for example.

4

PRO *TIP*

Notice the indentation? This helps us see the structure of the code. The `if` is triggered by the `while` and the `display.scroll` is trigerred by the `if`. Indentation is usually in blocks of four spaces.



So far we have our program waiting for button A to be pressed but if it isn't then nothing happens. We can fix this by adding in an `else`. An `else` can be understood like the word 'otherwise' as it is triggered if the `if` condition is not met, so if button A is not pressed then the `else` code is triggered.

```
from microbit import *
while True:
    if button_a.is_pressed():
        display.scroll("A")
    else:
        display.show(Image.SNAKE)
```

This `else:` block will run if the `if` condition is not met. It helps to keep a good user experience for your program as otherwise the screen would be blank.

There is one more part of the `if` statement to explore and that is the `elif` which stands for **else if**. The `elif` is used when you want to test for more than one condition. We can use this to see if button B is pressed. Rather than having a separate `if` statement we can add it to this one with an `elif`.

```
from microbit import *

while True:
    if button_a.is_pressed():
        display.scroll("A")
    elif button_b.is_pressed():
        display.scroll("B")
    elif button_a.is_pressed() and button_b.is_pressed():
        display.scroll("AB")
    else:
        display.show(Image.SNAKE)



```

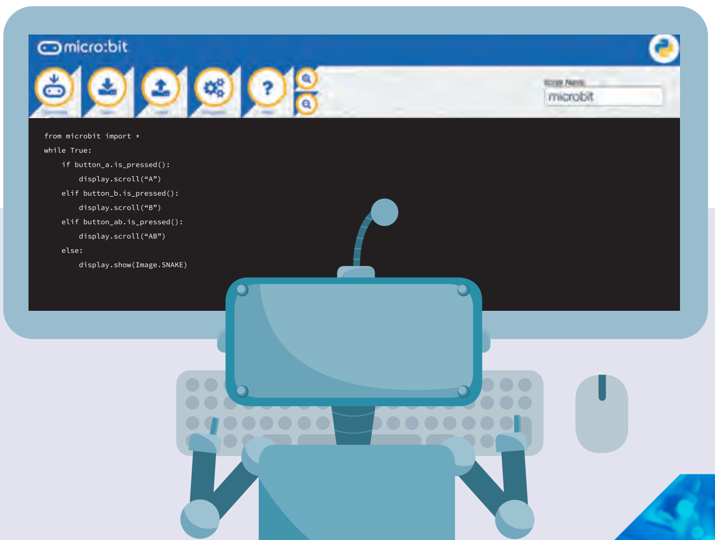
These `elif` allow us to test for more than one condition (a button being pressed in this example). You can have more than one `elif` if you need to and we have added another to test for both A and B being pressed.

Random numbers

We are now going to look at making a random number in MicroPython and then we have all the pieces to make the Math revision app in MicroPython. In MicroPython there is a **function** called `random.randint` that creates a random integer (number). A function is like a mini program built into MicroPython that does a specific job and `random.randit` creates random numbers. There are lots of other functions that do lots of other things and using them makes programming much quicker and easier. This **function** is part of a **module** called 'random' and we need to import it at the start of our program.

```
from microbit import *
import random

```

This imports all the micro:bit modules
This imports the random module



To generate a random number we need to ‘call’ the function and give it some parameters, which contain ‘arguments’. This tells MicroPython what range of numbers to use (0–9 for example).

```
from microbit import *
import random

while True:
    if button_a.is_pressed():
        int1 = random.randint(0,10)
        int2 = random.randint(0,10)


```

Here we have set up our forever (`while True`) loop and have used an `if` statement to see if button A is pressed. If it is, we create a **variable** called `int1` (just like in MakeCode) and then call the `random.randint` function with the parameters `(0,10)` which select a number between 0 and 10. We then do this again for `int2`.

The finished program

```
from microbit import *
import random

while True:
    if button_a.is_pressed():
        int1 = random.randint(0,10)
        int2 = random.randint(0,10)

        display.scroll(int1)
        display.scroll(“x”)
        display.scroll(int2)
        display.scroll(“= ?”)
    elif button_b.is_pressed():
        display.scroll(str(int1 * int2)
    else:
        display.show(“<-Q A->”)
display.clear()
```

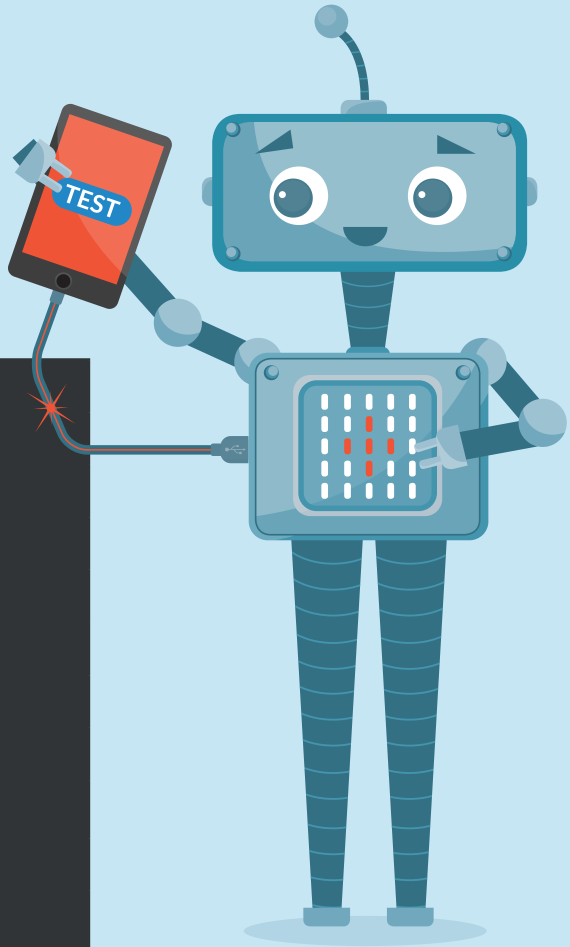
Here we have added some `display.scroll()` to create the math problem on the display and we have added the initial prompt under the `else` so users know what the buttons do (Q = Question, A = Answer).

Test it

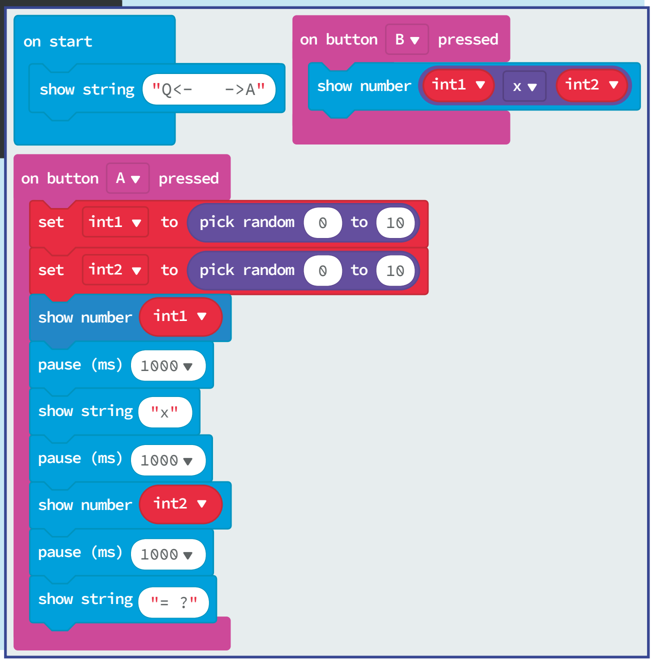
Download the .hex file by clicking the Download button: and copying it to the micro:bit.

Task

- Compare the MakeCode to the MicroPython below to make sure you understand how they are similar
- Draw lines from MicroPython code to the corresponding MakeCode blocks
- Discuss how they are structurally different




```
from microbit import *
import random

while True:
    if button_a.is_pressed():
        int1 = random.randint(0,10)
        int2 = random.randint(0,10)

        display.scroll(int1)
        display.scroll(“x”)
        display.scroll(int2)
        display.scroll(“= ?”)
    elif button_b.is_pressed():
        display.scroll(str(int1 * int2))
    else:
        display.show(“<-Q A->”)
display.clear()
```



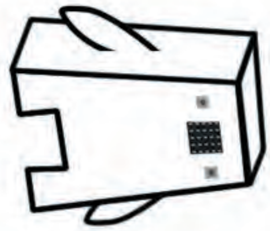
Final thoughts

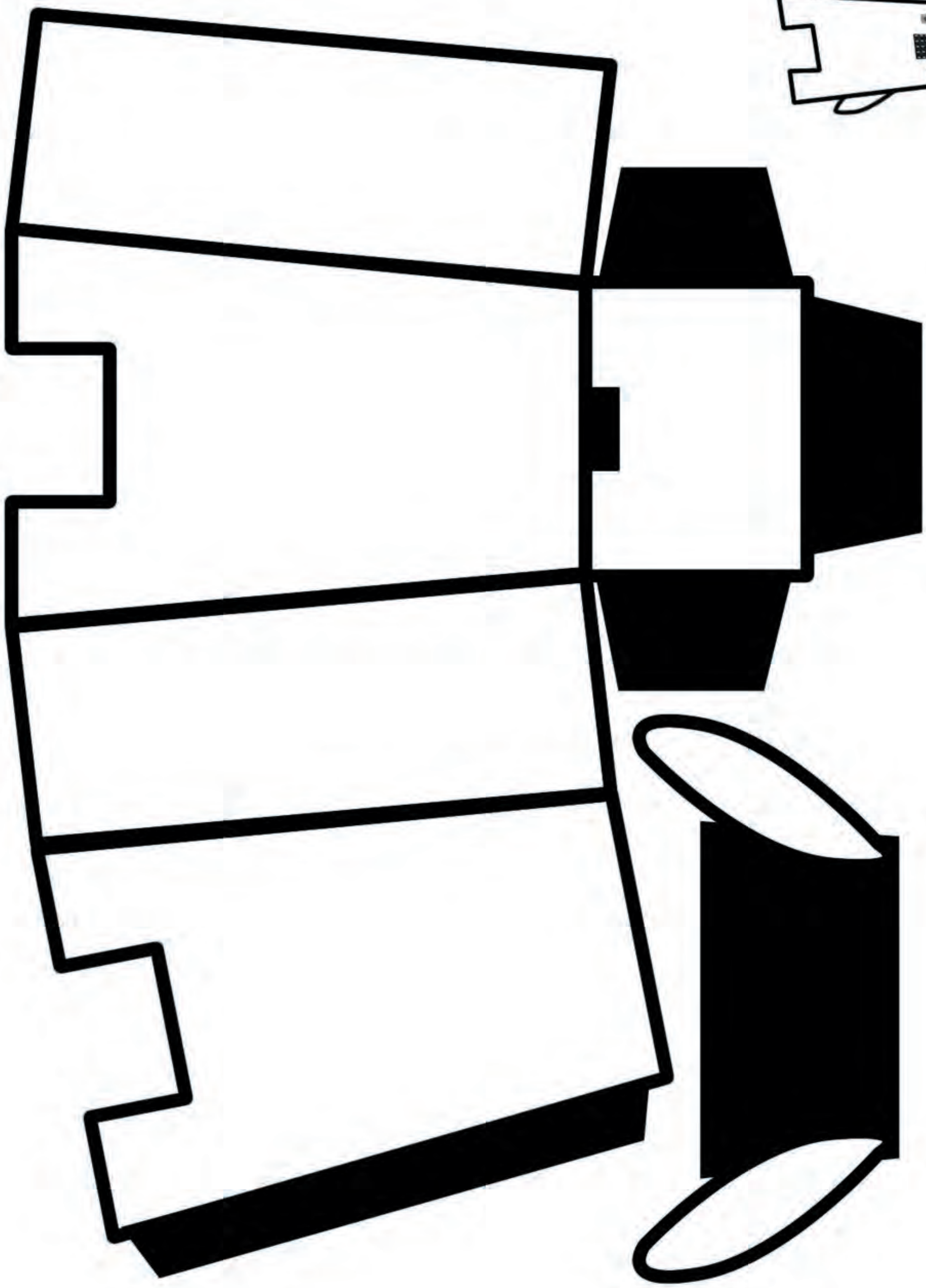
We have learned some of the fundamental building blocks of programming and used them to make a math revision app in real code. We have learned about:

- Variables
- Selection (if, elif, else)
- While loops
- Functions and modules

Stretch tasks

- Explore the micro:bit modules (https://microbit-MicroPython.readthedocs.io/en/latest/microbit.html#functions) and think about how you could use these tools to solve a problem.
- Explore the other functions in the random library (https://microbit-MicroPython.readthedocs.io/en/latest/random.html).
- Explore other modules and built-in functions (https://microbit-MicroPython.readthedocs.io/en/latest/microbit.html) such as:
  - Speech
  - SPI
  - UART
  - Accelerometer





MICROPYTHON DESIGN SHEET

Use this sheet to design your images and animations. Fill the square with number from **0–9** with 0 being off and 9 being the brightest and give it a name.

0	9	0	9	0
0	9	0	9	0
0	0	0	0	0
9	0	0	0	9
0	9	9	9	0

smiley1

OTHER ASP PRODUCTS

The Arm School Program has a wealth of project based learning teaching resources, all freely available at https://school.arm.com.

These include the following.

Arm School Program on YouTube

Visit our YouTube channel for a range of videos to support teaching and learning Computer Science at home and in the classroom. You can also download tasks that encourage learners to apply the skills and knowledge covered in our two playlists, Introduction to Computing with Micro:bit

https://www.youtube.com/watch?v=M4JdhPqmp7A&list=PLXwAdcOl0lCrDF-ympqZC94r59Xpb36GD

and Introduction to MicroPython

https://www.youtube.com/watch?v=XTSNznidJvU&list=PLXwAdcOl0lCpKRep_Gb_YVDfYjcaqZSvA.

Teaching with Physical Computing on edX

Our program of four professional development courses on edX.org introduces teachers to physical computing and how to apply it through project based learning (PBL) in order to maximize engagement and learning in the classroom.

Whether you're new to teaching Computing or you're a specialist Computer Science teacher, Teaching with Physical Computing will help you get to grips with PBL: the pedagogy and its practical application. To access each course free of charge, after selecting 'Enroll' choose 'Audit this course'.

https://www.edx.org/course/teaching-with-physical-computing-course-1-introduction-to-project-based-learning

Smart Schools on Arduino

Our Smart Schools resource provides accessible and engaging projects for teachers and learners that utilize the more advanced features of Arduino in real-world contexts. It presents real problems that need solving with technology, with all the associated teaching and learning resources such as slides, lesson plans, and activities, as well as solution guides and cheat sheets for non-specialists.

The resources are split into UK key stages 3–5 (6th–8th grade, 9th–10th grade and 11th–12th grade respectively), with three courses per key stage.

https://www.arm.com/resources/education/schools/content/arduino-schools-projects

Arduino Projects for Schools

Arduino MKR Projects for Schools is a colorful entry-level resource, which introduces learners to the exciting world of microcontrollers, the Internet of Things, and data science. Learners use both simulators and physical devices to build systems and solve real-life problems.

https://www.arm.com/resources/education/schools/content/arduino-mkr-projects-for-schools

Introduction to Computing Using micro:bit

Journey through all the features of the micro:bit with interactive activities and engaging projects to excite and enthuse learners. The course uses MakeCode as the programming interface and is suitable for learners of all ages and abilities.

https://www.arm.com/resources/education/schools/content/computing-using-microbit

Robotics and Internet of Things Course

Take a journey of learning through the Internet of Things and robotics. Learners apply Arm-based technology to solve interesting and authentic problems, using micro:bits and other exciting peripherals to create autonomous cars and smart cities.

https://www.arm.com/resources/education/schools/content/robotics-and-iot

Introduction to Programming Using MicroPython

Learn about all the programming techniques contained in the UK Computer Science curriculum for 11–16-year-olds using MicroPython to program a micro:bit. This programming course covers the foundational computational techniques required for Computer Science for 14–16-year-olds in the UK.

https://www.arm.com/resources/education/schools/content/programming-using-micropython

Computational Thinking Tasks

These resources cover all the GCSE Computer Science computational thinking techniques for 14–16-year-olds. They contain lots of interactive activities to reinforce understanding and prepare learners for their exams.

https://www.arm.com/resources/education/schools/content/computational-thinking

International Computing Course with micro:bit

This is a complete curriculum designed for schools internationally, covering grades 5 through 7. Each grade includes 40 hours of teaching and learning content and uses the Arm School Program's pedagogical approach of PBL. The course is based on the use of micro:bits and MicroPython.

https://www.arm.com/resources/education/schools/content/microbit-international