Name: ____________________________

An tSraith Shóisearach do Mhúinteoirí

Junior CYCLE for teachers

Control Systems for Mechatronics in Junior Cycle Engineering

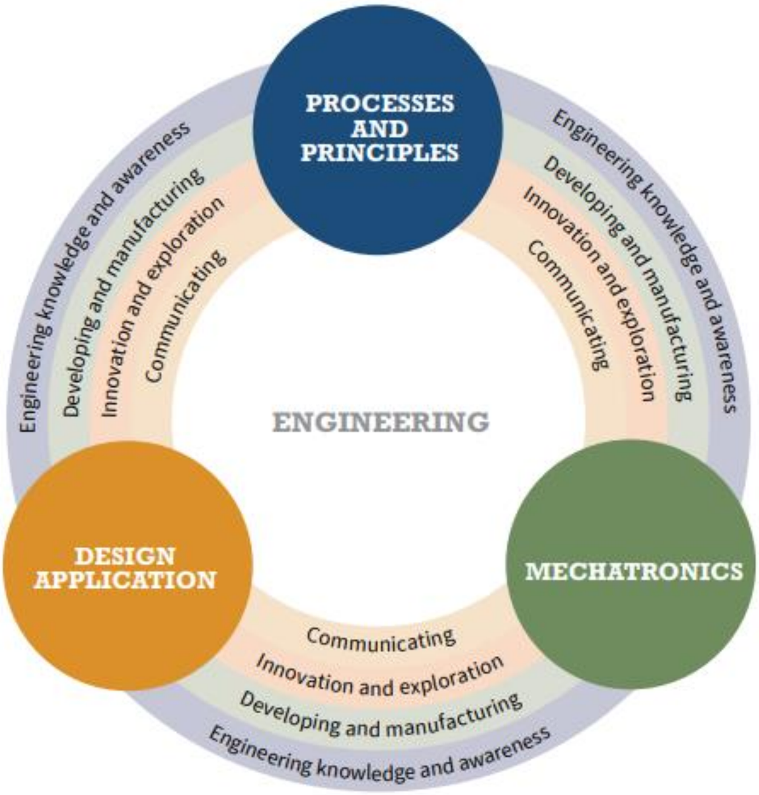
Specification

Ionad Oideachais Mhuineachain

An Roinn Oideachais
Department of Education

www.jct.ie

Introduction:


PROCESSES AND PRINCIPLES
ENGINEERING
DESIGN APPLICATION
MECHATRONICS
Engineering knowledge and awareness
Developing and manufacturing
Innovation and exploration
Communicating

Since its introduction in 2018, the Engineering specification has been taught in schools around the country. The specification introduced a strand titled Mechatronics. This strand's descriptor and learning outcomes reference control technology and the use of code and sequences to solve problems.
The following document contains resources that JCt4 have developed to support teachers utilise software in control systems.

There are many different types of support in this document such as step-by-step instructions how to engage with micro:bit, micro:bit classroom and makecode.microbit.org.

The first section of this resource contains step-by-step modules that take you through coding, wiring, and creating micro:bit circuits.

Page 3 and 4 contains a table of YouTube links to activities associated with the modules and also the page numbers for each resource in this booklet.

There are two methods that allow you to engage with these modules:

1. Following the text instructions
2. Using the step-by-step videos that take you through the process which can be accessed using the QR codes or the hyperlinked table of contents.

The second section of this resource contains links to webinars and live demonstrations that were carried out over the past number of years.

Section 1

Section 2

Supporting Mechatronics in Engineering OLE 2020/2021

This webinar covered many different topics such as developing and testing circuits using TinkerCAD, two different areas of coding using micro:bit, using micro:bit classroom as a resource for further engaging with control technology with students. The coding challenges, and how to code the challenges are used in the webinar are on page 102. The webinar recording in full can be found using the following link: **https://www.youtube.com/watch?v=eVJ6hdb9sCE**

For particular sections of the Video click on the following links:

For engaging with TinkerCAD and circuits:	https://youtu.be/eVJ6hdb9sCE?t=257
Introduction to Micro:bit:	https://youtu.be/eVJ6hdb9sCE?t=815
For engaging with Micro:bit: and Challenges	https://youtu.be/eVJ6hdb9sCE?t=1535
Micro:bit Classroom	https://youtu.be/eVJ6hdb9sCE?t=3368

Classroom Practice and Computer Software in Junior Cycle Engineering Spring OLE 2021/2022

This webinar looked at engaging students with control technology software in an online learning environment. It considered three different school settings and three different areas that the teachers developed with their students. At the end of the session, we engaged with how to program a micro:bit from a phone or tablet. The webinar recording in full can be found using the following link: **https://www.youtube.com/watch?v=lE_IxcVFlgg**

For particular sections of the Video click on the following links:

For engaging with makecode and a phone/tablet:	https://youtu.be/lE_IxcVFlgg?t=3244

Mechatronics in Engineering

Micro:bit is used for the purpose of developing teachers' knowledge, understanding, skills and values in computer software, however, any brand of microcontroller may be used in Mechatronics. This was determined through feedback from teachers as being the most popular choice for a variety of reasons.

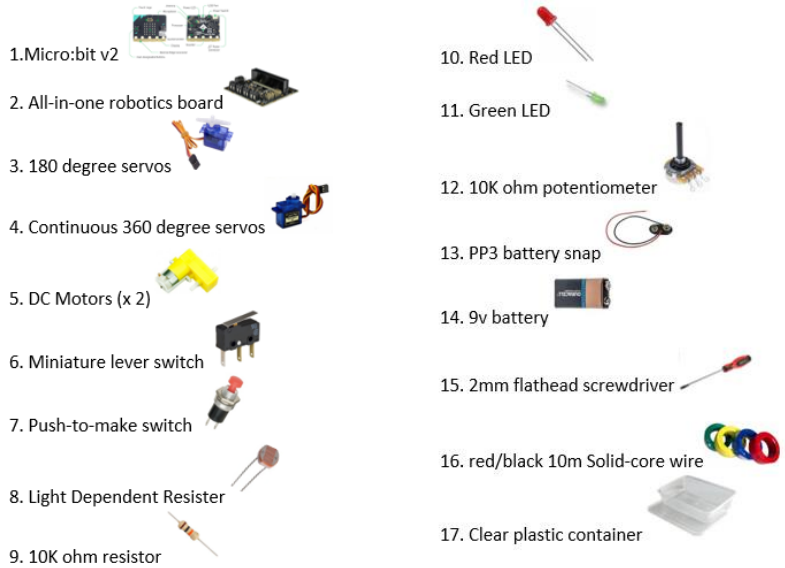
Note: ***<u>Computer Software Technology on its own is not Mechatronics</u>***. A Mechatronic system includes a combination of mechanisms, electronic components and computer software. An ***<u>integrated, non-linear</u>*** approach should be taken to the teaching and learning of mechatronics.

The following pages are structured into modules which aim to develop understanding and skills with particular reference to the strand of Mechatronics in Junior Cycle Engineering.

In total there are 7 modules which progress in complexity and aim to incrementally develop understanding and application in the strand of Mechatronics. Some modules may have more than one activity. Explanatory videos can be found in each module which outline some of the key features and will offer guidance to support the activities.

The following components and materials have been made available to all teachers within the Mechatronics teaching and learning resource packs.

JCt4 Computer Software Component list



1.Micro:bit v2
2. All-in-one robotics board
3. 180 degree servos
4. Continuous 360 degree servos
5. DC Motors (x 2)
6. Miniature lever switch
7. Push-to-make switch
8. Light Dependent Resister
9. 10K ohm resistor
10. Red LED
11. Green LED
12. 10K ohm potentiometer
13. PP3 battery snap
14. 9v battery
15. 2mm flathead screwdriver
16. red/black 10m Solid-core wire
17. Clear plastic container

Teachers should also bring a **Soldering Iron, Solder and Wire Cutter and/or Wire Stripper** to aid any assembly which has not being completed prior to the workshop.

<u>Instruction:</u> When engaging in a self-directed approach, start by reflecting on the activity in the learning log, then watch the associated video prior to engaging in the coding activity.

Module Content

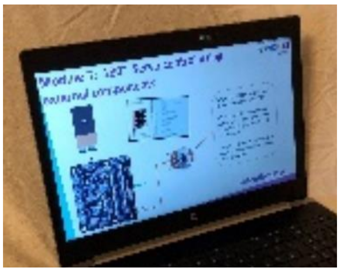
- Getting Started
- Module 1 – Introduction to basic commands
- Module 2 – Simulating Motor Control – 180^{o} and Continuous Servo's
- Module 3 – DC Motors
- Module 4 – Adding Inputs and Outputs
- Module 5 – Control using Potentiometers
- Module 6 – Control using Sensors
- Module 7 – Radio Control

Participating through Self-Direction

Each module is supported through a range of videos and concise instructions in the learning log. Participants will be able to communicate through the assigned breakout room to offer collegial support. The advisor will also be available.

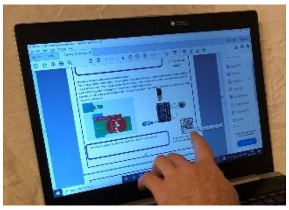
Instructions:

1) Choose a module. Read the relevant pages for that module in the learning log.



2) Watch the video which accompanies the module. The video will explain the problem-solving approach to coding. It will explain the various aspects to the code, and how to find and assemble the blocks for the code.

3) The video can be found by scanning the QR code with the camera on your phone, this will allow you to watch the video on your phone. Alternatively, if you open the learning log on your laptop as a PDF, you will be sent to the video by clicking directly on the link.



4) Assemble the micro:bit and relevant components, download the code and test the code that you have assembled.

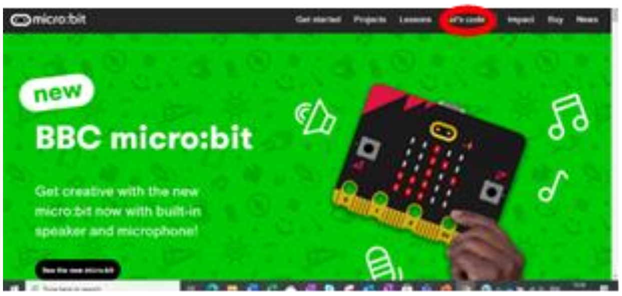
5) Revisit the video to further your understanding on the activity.

Getting Started

This section will show you how to navigate to the MakeCode website which facilitates the programming of a micro:bit.

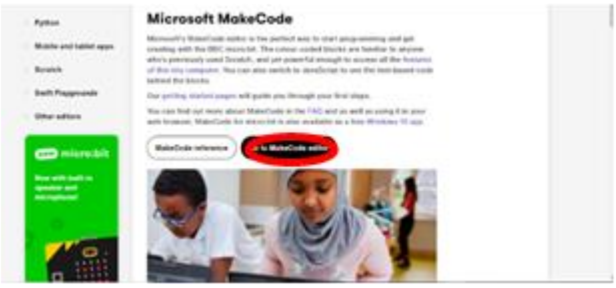
1) Use the following link to direct you towards Makecode.
Url Link: https://makecode.microbit.org/#
Alternatively use the following search words in your browser.
Key Search Words: Micro:bit MakeCode

2) Select 'Lets Code'



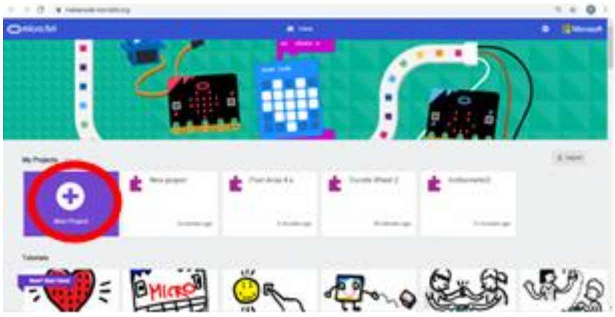
Get creative with the new micro:bit now with built-in speaker and microphone!

3) Select ' Go to MakeCode editor'

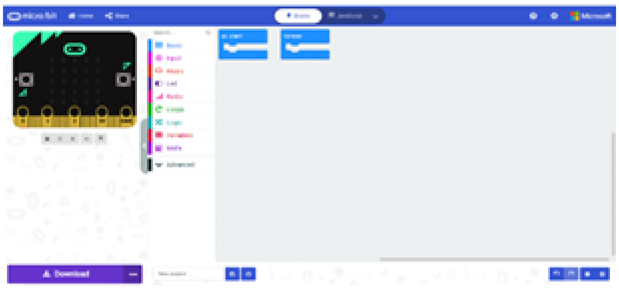


Microsoft MakeCode

4) Select 'New Project'



5) Start Coding!



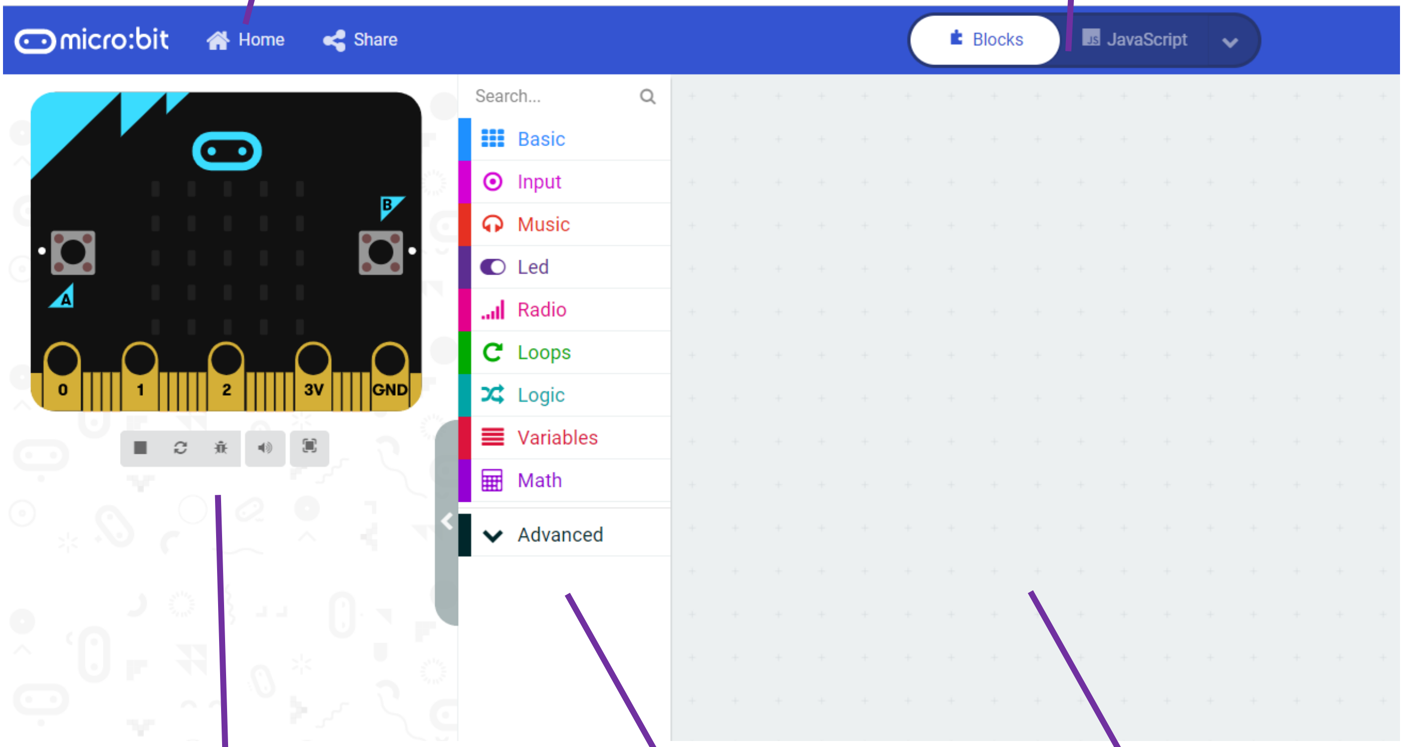
Basic Commands Introduction to 'MakeCode'

The home screen is accessed through this button. From the home screen you can create a new project, access tutorials and any previous projects you may have created.

This button allows you to swap between drag and drop coding blocks and JavaScript. Students may need to be familiar with coding languages to engage in JavaScript.

Video

https://youtu.be/22_lFT1yOU0



This is the simulator where all code can be tested where you have used the original coding blocks

The coding tabs where all the coding blocks are stored.

This area on screen is where the code is populated. This is called the 'Editor'.

Module 1: Introduction to Basic Commands

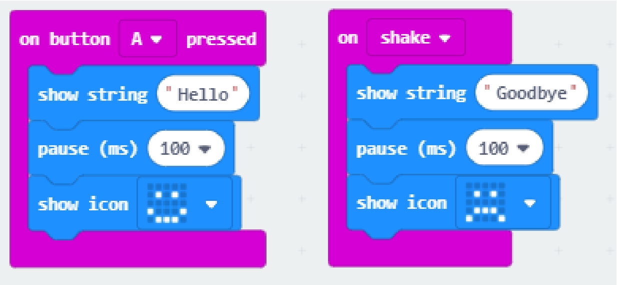
This module will introduce participants to programming using block code on MakeCode. A code is developed to programme the LED matrix to communicate the letters and numbers. This will be activated by pressing a button on the micro:bit. The code will be tested on the simulator. Use the video for support through the supplied link.

Activity A

When button A is pressed, the word 'HELLO' will scroll across the LED matrix. It will pause briefly before an emoji appears on the screen.
When someone shakes the Micro:bit, the word 'Goodbye' will scroll across the screen. It will pause briefly and a separate emoji will appear.

Video

https://youtu.be/57RUjgjG3VU



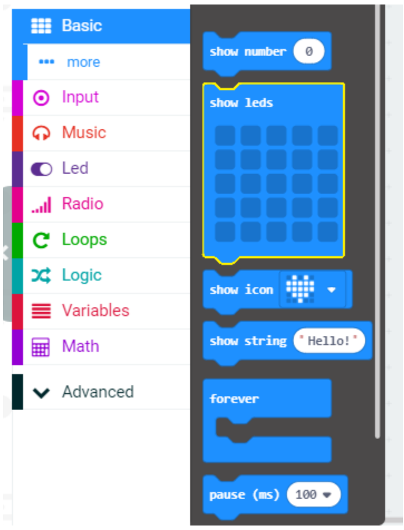
```
on button A ▾ pressed
    show string "Hello"
    pause (ms) 100 ▾
    show icon ▾

on shake ▾
    show string "Goodbye"
    pause (ms) 100 ▾
    show icon ▾
```

This code is intended to fulfil the activity constraints. The following coding tabs are required in the menu:

Inputs: pressing buttons or giving signals to start a function such as shaking the board.
Basic: simple functions such as scrolling text and icons on screen.
Lastly the simulator will help us test the code before uploading to the micro:bit

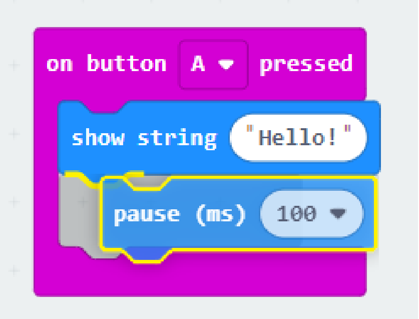
1:

- To start: On the home screen click on the 'input' tab.
- The options will appear on screen for you as in this picture.
- Drag and drop out the required coding block. In this activity it is the 'on button A pressed'.
- Select
- Place 'on button A pressed' anywhere in the editor.



2:

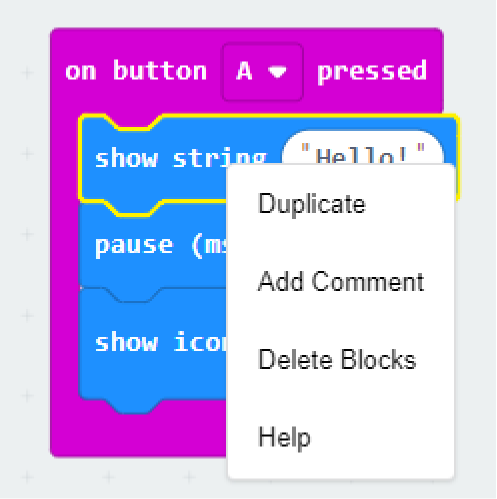
- Next click on the 'Basic' tab and the options will appear as we see them in the picture to the left.
- Choose the 'show string' option and drag it onto the coding area.
- Place is in the gap that's present in the 'on button A pressed' input from step 1.



```
on button A pressed
    show string "Hello!"
    pause (ms) 100
```

3:

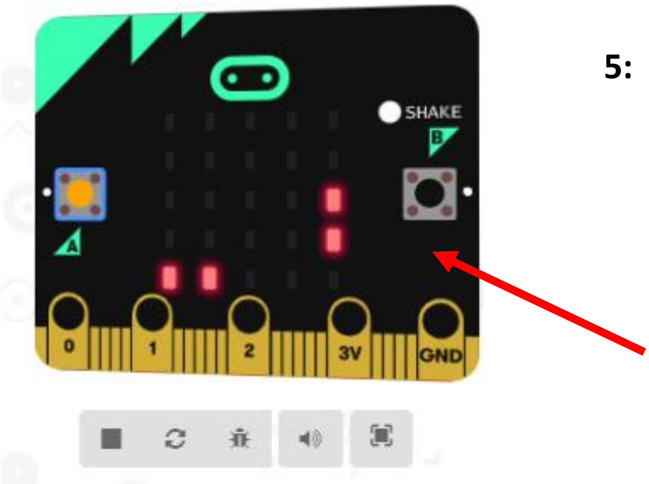
- Return to the 'Basic' tab.
- Select the 'pause' block.
- Drag and drop it into the 'on button A pressed' as shown. You can change the time as you so wish. Time is in milliseconds so 1 second = 1000 milliseconds.
- Using 'Basic' select an icon of your choice, the drop-down arrow beside the face allows you to change the icon.



```
on button A pressed
    show string "Hello!"
    pause (ms
    show icon
```

4:

- Using the 'Input' tab we now find the 'on shake' option
- This time we are replicating some of the code. Instead of re-finding it we can right click on one block and duplicate it.
- Click on the text in the 'show string' block, in order to change the text to goodbye.
- Use the drop-down arrow in order to change the icon again to replicate the code.



5:

- Now we can test the code on the simulator.
- Press button A on the micro:bit on screen and test if your code works as intended.
- To simulate, move the mouse vigorously over the micro:bit. This will simulate the shake in real-life. Alternatively, press the newly added shake button on the simulator in order to test the second part of the code.

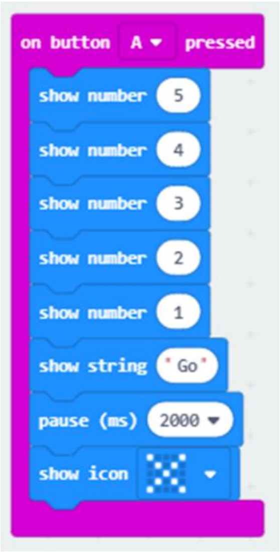
Reflection / Notes:

Activity B

After a countdown of 5 seconds, the display will indicate to pedestrians to cross a road. After 2 seconds, the display will change to a stop icon indicate to additional pedestrians that it is unsafe to cross. Use the video to support you in this activity.

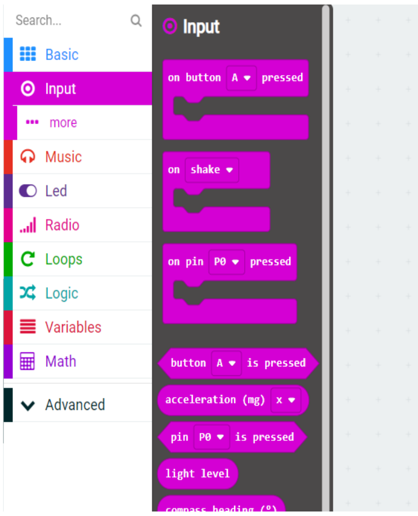
Video

https://youtu.be/fTibjkEyzSc



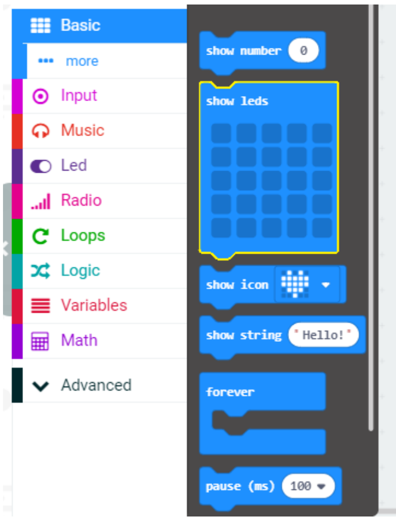
```
on button A pressed
    show number 5
    show number 4
    show number 3
    show number 2
    show number 1
    show string "Go"
    pause (ms) 2000
    show icon
```

Reflection on Observations:



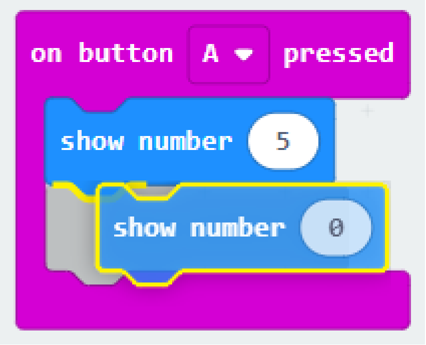
1:

- To start: On the home screen click on the 'input' tab.
- The options will appear for you on screen as in this picture.
- Drag and drop out the coding block you want. In this activity, it is the 'on button A pressed'.
- Place the 'on button A' code anywhere on the area shown in the previous page.



2:

- Next, click on the 'Basic' tab and the options will appear as we see them in the picture to the left.
- Choose the 'show number' option and drag it onto the coding area.
- Place it in the gap that is present in the 'on button A pressed' input from step 1.

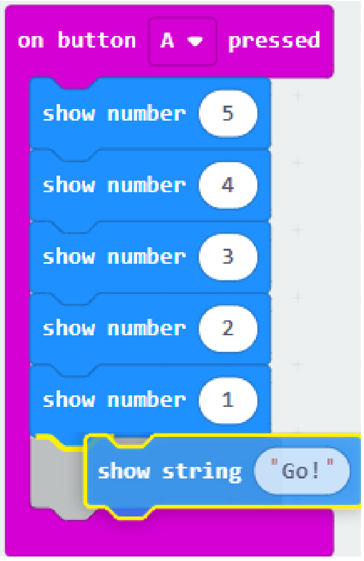


```
on button A pressed
    show number 5
    show number 0
```

3:

- Drag and drop it into the ‘on button A pressed’ as shown.
- Return to the ‘Basic’ and continue to add blocks of ‘show number’. **Note: Add numbers with a reducing** value from 5 to 1.

  Duplicate as shown in the previous module which may help speed up the process.

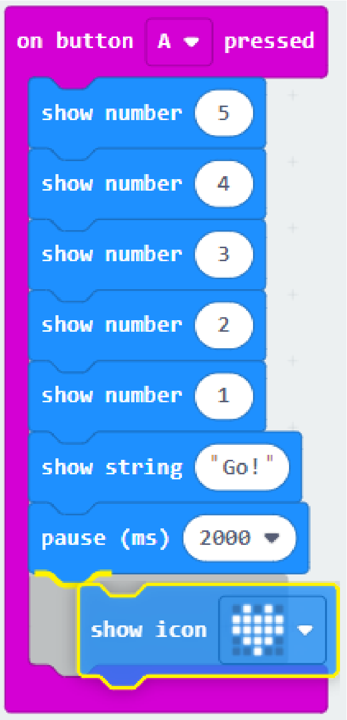


```
on button A pressed
    show number 5
    show number 4
    show number 3
    show number 2
    show number 1
    show string "Go!"
```

4:

- In the ‘Basic’ tab use the ‘show string’ to add text as shown in Module 1, enter any text you wish to indicate to someone to cross the road.
- Return to the ‘Basic’ tab and choose the ‘pause’ block’ add a pause for 2 seconds as per the brief.

❖ **Remember:** Time is in milliseconds so 1 second = 1000 milliseconds.



```
on button A pressed
    show number 5
    show number 4
    show number 3
    show number 2
    show number 1
    show string "Go!"
    pause (ms) 2000
    show icon
```

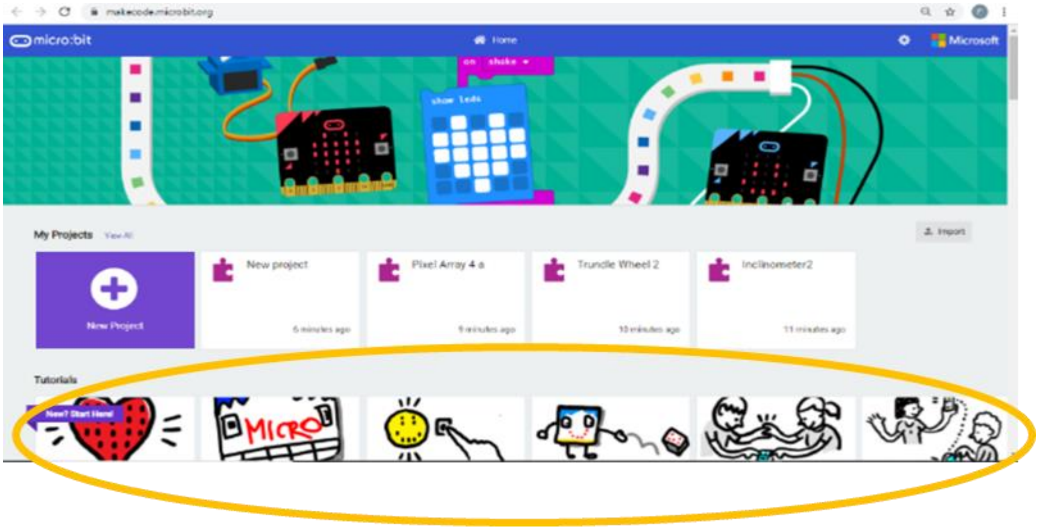
5:

- In the ‘Basic’ tab use the ‘show icon’ to add an icon as shown in Module 1, enter any icon you wish to indicate to someone not to cross the road.
- Use the simulator as shown on page 8 in order to see if this code works.

Activity C

MakeCode has a range of tutorials and sample codes. Take this opportunity to explore the tutorials, and consider how they may be used with your students in the context of Engineering.

1) Return to your Home Page.
2) Scroll through the range of tutorials.
3) **What Code would you think your students may be interested in?**



Reflection on Observations:

Module 2: Simulating Motor Control

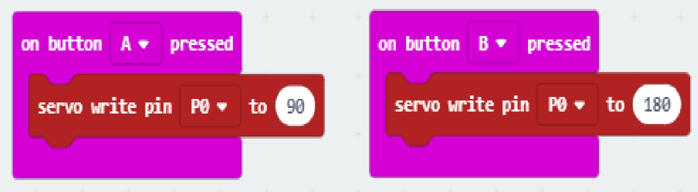
This module will introduce participants to programming a servo motor using block code on MakeCode. A code is developed to programme a servo motor. This will be activated by pressing a button on the micro:bit. The code will be tested on the simulator in each of the 4 activities. Use the video to support you in the activity.

Video

https://youtu.be/K0RuWNR4pcA

Activity A

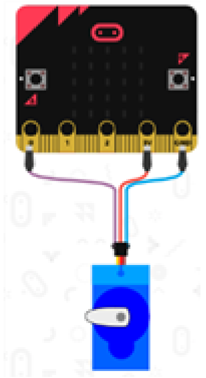
Open and close a barrier using a 180^0 servo motor



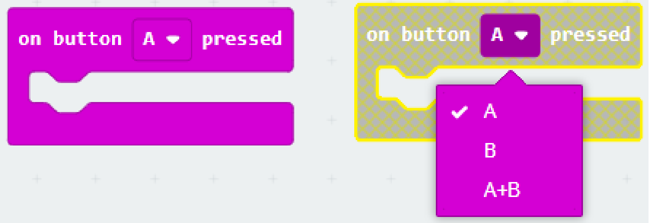
```
on button A pressed
    servo write pin P0 to 90

on button B pressed
    servo write pin P0 to 180
```

This is one possible solution to the task!
Did you consider an alternative solution?

MakeCode simulator



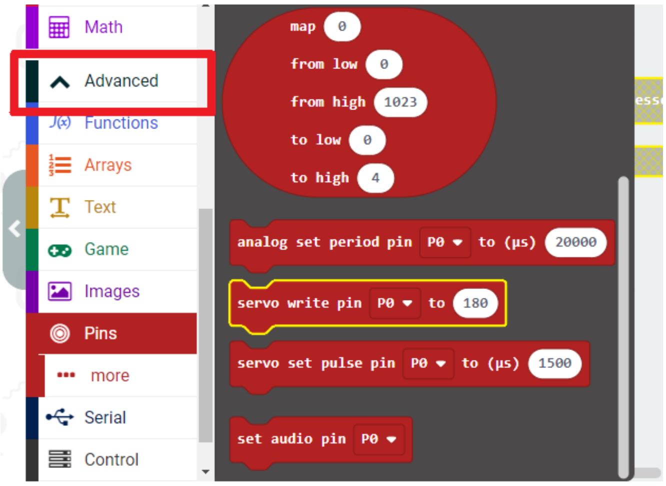
1:



```
on button A pressed

on button A pressed
```

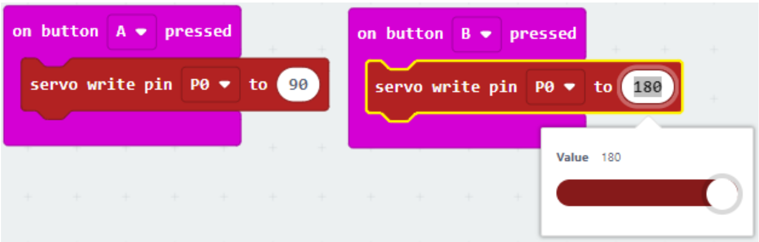
- To start: Click on the 'input' tab and select 'on button A pressed' and bring drop it into the editor.
- Duplicate this block by right clicking and selecting duplicate.
- As you can see on the left, it is yellow as it is not possible to have two commands with the same button.
- Click the drop-down arrow beside 'A' and select 'B'.

2:



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

analog set period pin P0 to (µs) 20000

servo write pin P0 to 180

servo set pulse pin P0 to (µs) 1500

set audio pin P0
```

- In order to find the 'pins' tab, you must click on the 'Advanced' drop down menu at the bottom of the list of tabs.
- From this list choose 'servo write pin P0 to __' and drop it into 'on button A pressed'
- Duplicate, 'servo write pin P0 to __' and add it to 'on button B pressed'.
- Set the degrees to two different values and use the simulator in order to test if it works.



```
on button A ▾ pressed
    servo write pin P0 ▾ to 90

on button B ▾ pressed
    servo write pin P0 ▾ to 180
```

3:

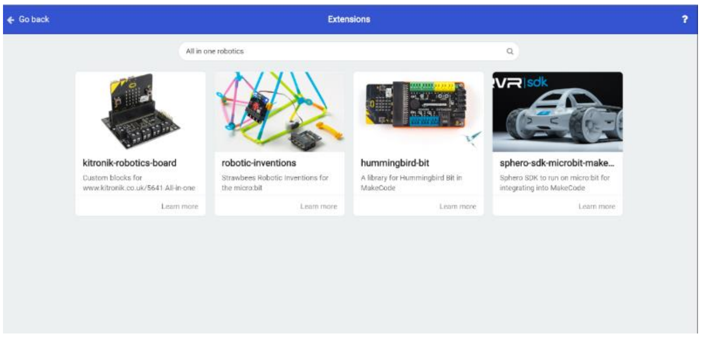
- Set the degrees to 2 different values for example 90^0 and 180^0 and use the simulator in order to test if it works.

Reflection on Observations:

Finding the All-in-one Robotics Board Extension

The All-in-one Robotics board is useful to facilitate the driving of DC and Servo motors using the micro:bit. The micro:bit on its own will only supply motors with a maximum of 3V. However, in most instances the motors used in the engineering workshop will require a higher voltage in order to achieved full speed and torque.   The robotics board will supply the additional required voltage. The programming of the robotics board requires a Robotics block to be added to the MakeCode menu. The following explains how to this. The video in the next activity will also explain how to add the extension.

Summary: In coding tabs / menu select, ‘Advanced’, and then ‘Extensions’. In the search bar type ‘All-in-one robotics board’ and select the option when it appears.

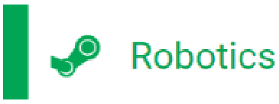


kitronik-robotics-board
Custom blocks for www.kitronik.co.uk/5641 All-in-one
Learn more

robotic-inventions
Strawbees Robotic Inventions for the micro:bit
Learn more

hummingbird-bit
A library for Hummingbird Bit in MakeCode
Learn more

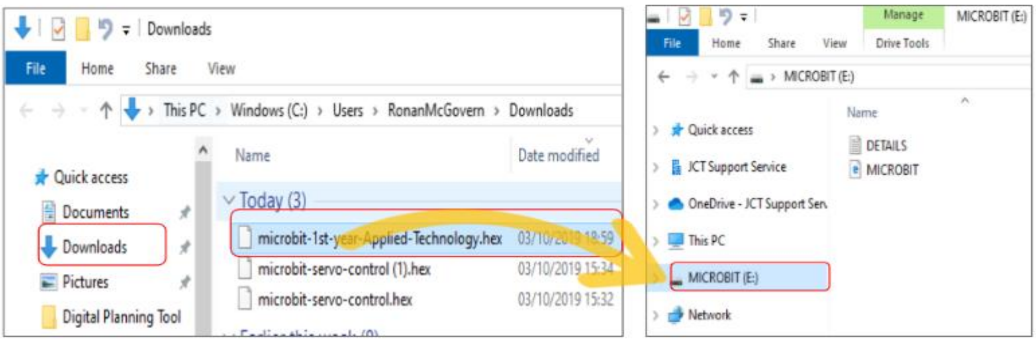
sphero-sdk-microbit-make...
Sphero SDK to run on micro:bit for integrating into MakeCode
Learn more



Downloading Files from MakeCode

When a code has being developed, it can then be downloaded to the micro:bit. The code is called a ‘Hex File’. Following the instructions to learn this procedure.

Transferring your HEX file to your micro:bit



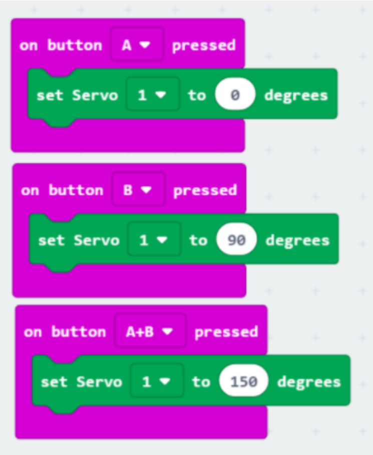
Copy file from your download folder
Paste file into micro:bit drive.

Video

https://www.youtube.com/watch?v=kxHklujresY&feature=youtu.be

Activity B – Servo Motor Control

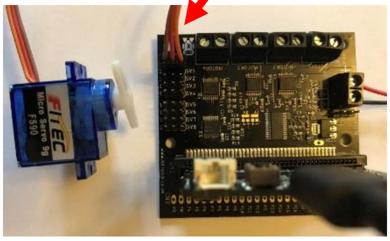
Program a 180⁰ servo to prescribed positions. Build the code and download to the Micro:bit. Use the Video to support you in the activity.



```
on button A pressed
    set Servo 1 to 0 degrees
on button B pressed
    set Servo 1 to 90 degrees
on button A+B pressed
    set Servo 1 to 150 degrees
```

- Press A – Rotate servo full speed clockwise
- Press B – Servo is fully stopped
- Press A+B – Rotate servo anti-clockwise, at 66% of the full speed

The Servo is configured to pin 1. The orange lead is the signal and must be facing to the inside of the board.



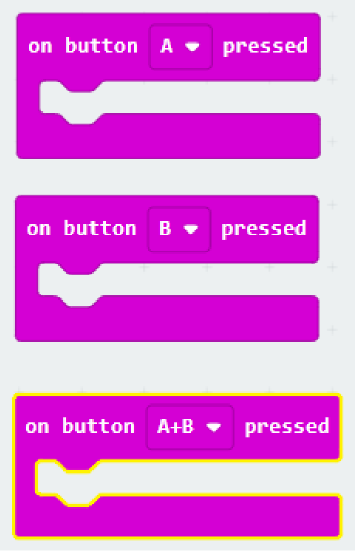
Additional Resources from the Mechatronics Elective 2020

The principles behind coding a servo motor

https://drive.google.com/file/d/1bKDBKxJ8U7KuZNXTGPcFltaB6dOrJqeL/view

Video to a possible design challenge to apply the above code.

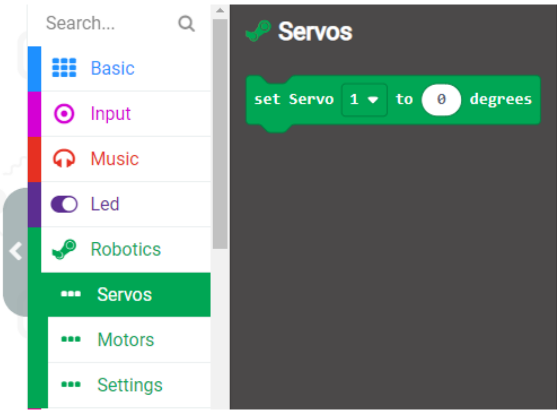
https://www.youtube.com/watch?v=Y1CcKXKZkPI&feature=youtu.be



```
on button A pressed
on button B pressed
on button A+B pressed
```

1:

- To start: Click on the ‘input’ tab. Select ‘on button A pressed’. Duplicate this block twice, so that there are a total of three blocks.
- If you click the drop-down arrow beside ‘A’, you are given more options to select for buttons to be pressed. This is seen with the blocks on the left

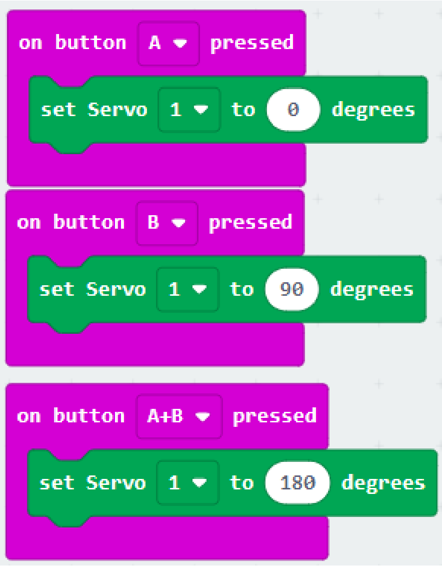


```
set Servo 1 ▾ to 0 degrees
```

2:

- Using the newly added ‘robotics’ tab as shown on page 14, click on ‘servos’
- Use the ‘set servo 1 to __ degrees’ block and add one to each of the inputs.

❖ **Reminder:** right clicking on the block in the coding area allows you to duplicate the blocks over and over again.



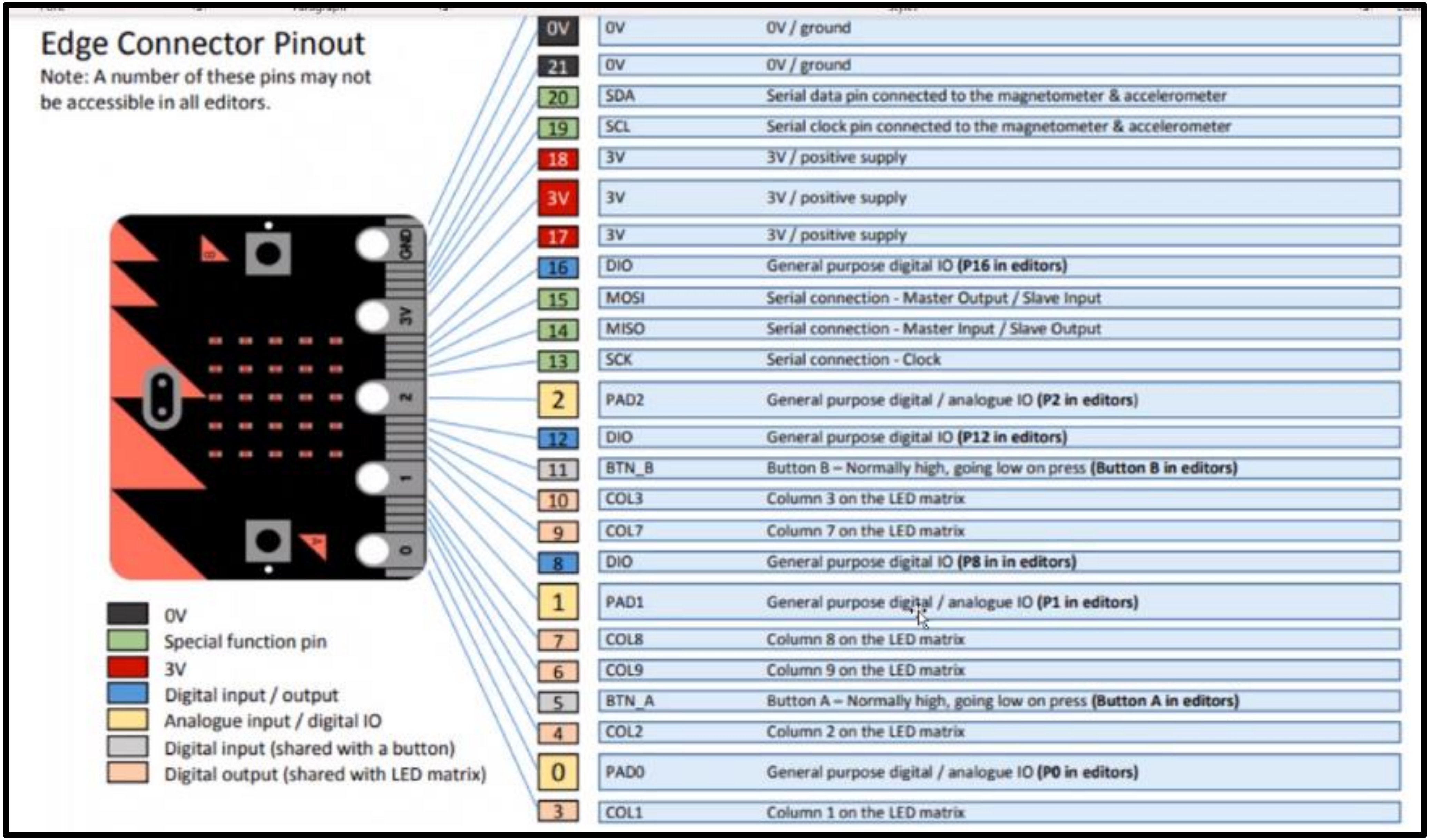
```
on button A ▾ pressed
    set Servo 1 ▾ to 0 degrees

on button B ▾ pressed
    set Servo 1 ▾ to 90 degrees

on button A+B ▾ pressed
    set Servo 1 ▾ to 180 degrees
```

3:

- Change each of the degrees to different values
- From here, download the code to your micro:bit as shown on page 14. Using the ‘All-in-one board’ and a 180-degree servo, try out the code shown.

*If you wish to use more than single 180-degree servo, duplicate the green blocks but change the number ‘servo 1’ to where you connect the second servo to on the All-in-one robotics board

Reflection on Observations:



Edge Connector Pinout

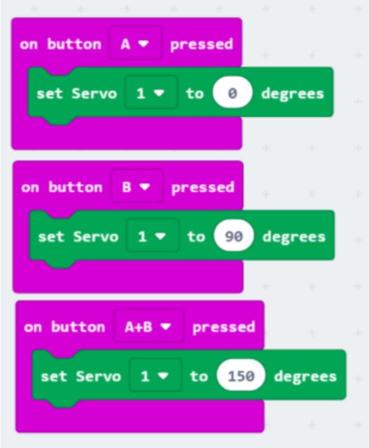
Note: A number of these pins may not be accessible in all editors.

- ⬛ 0V
- 🟩 Special function pin
- 🟥 3V
- 🟦 Digital input / output
- 🟨 Analogue input / digital IO
- ⬜ Digital input (shared with a button)
- 🟧 Digital output (shared with LED matrix)

0V	0V	0V / ground
21	0V	0V / ground
20	SDA	Serial data pin connected to the magnetometer & accelerometer
19	SCL	Serial clock pin connected to the magnetometer & accelerometer
18	3V	3V / positive supply
3V	3V	3V / positive supply
17	3V	3V / positive supply
16	DIO	General purpose digital IO **(P16 in editors)**
15	MOSI	Serial connection - Master Output / Slave Input
14	MISO	Serial connection - Master Input / Slave Output
13	SCK	Serial connection - Clock
2	PAD2	General purpose digital / analogue IO **(P2 in editors)**
12	DIO	General purpose digital IO **(P12 in editors)**
11	BTN_B	Button B – Normally high, going low on press **(Button B in editors)**
10	COL3	Column 3 on the LED matrix
9	COL7	Column 7 on the LED matrix
8	DIO	General purpose digital IO **(P8 in in editors)**
1	PAD1	General purpose digital / analogue IO **(P1 in editors)**
7	COL8	Column 8 on the LED matrix
6	COL9	Column 9 on the LED matrix
5	BTN_A	Button A – Normally high, going low on press **(Button A in editors)**
4	COL2	Column 2 on the LED matrix
0	PAD0	General purpose digital / analogue IO **(P0 in editors)**
3	COL1	Column 1 on the LED matrix

Activity C

Modify the code shown below from the previous activity by changing the value of the angles. Plug the servo into a different pin and modify the code to reflect this change.



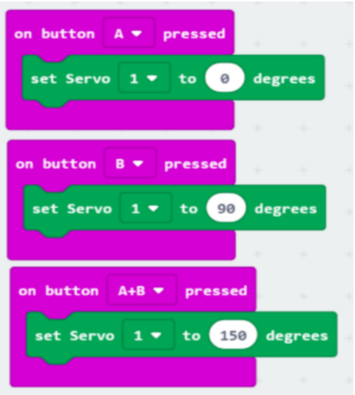
```
on button A ▾ pressed
    set Servo 1 ▾ to 0 degrees

on button B ▾ pressed
    set Servo 1 ▾ to 90 degrees

on button A+B ▾ pressed
    set Servo 1 ▾ to 150 degrees
```

- Place the servo into any other pin number.
- Modify the code to reflect the new pin position.
- Modify the angles
- Download and test the new code.

Watch the stimulus video attached to the QR code.
How might you engage your students in this activity in Engineering?

Activity D

Control the speed and direction of a 360^0 servo. Explore the following piece of code with the supporting video.



```
on button A ▾ pressed
    set Servo 1 ▾ to 0 degrees

on button B ▾ pressed
    set Servo 1 ▾ to 90 degrees

on button A+B ▾ pressed
    set Servo 1 ▾ to 150 degrees
```

- Press A – Rotate servo full speed clockwise
- Press B – Servo is fully stopped
- Press A+B – Rotate servo anticlockwise at 66% of full speed

Download the program and simulate using the All-in-one-Robotics board.

Video

https://www.youtube.com/watch?v=kxHklujresY&feature=youtu.be

Additional Resources from the 2020 Mechatronics Elective

The principles behind coding a 360° servo motor.

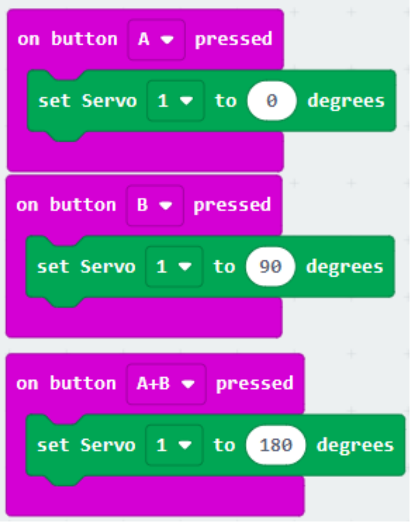
https://drive.google.com/file/d/1K8YIIDU-f28av5g8hgOSR_sUziHp1Su0/view

Video to a design challenge to apply the above code.

https://www.youtube.com/watch?v=hJRMD0gKH_s&feature=youtu.be

Watch the stimulus video attached to the QR code above.
How might you engage your students in this activity in Engineering?

Activity E

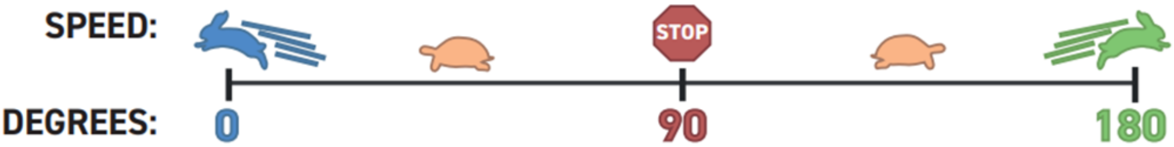


```
on button A ▾ pressed
  set Servo 1 ▾ to 0 degrees

on button B ▾ pressed
  set Servo 1 ▾ to 90 degrees

on button A+B ▾ pressed
  set Servo 1 ▾ to 180 degrees
```

1:

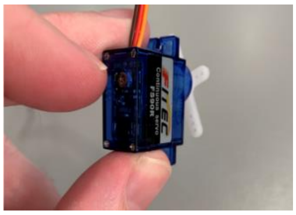
- The coding setup is the same as previously shown in Activity 3.
- When looking at the code it looks identical
- The difference is the way in which the servo works.

❖ A continuous servo is not controllable in terms of degrees. It is controlled in terms of speed in a given direction. 90 degrees is 0 in terms of speed.

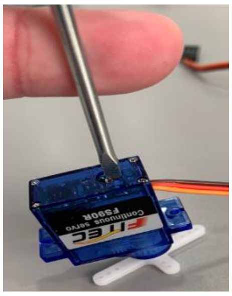


2:

- The diagram above shows the differences between the degrees.
  1. 0 degrees is full speed of the servo turning to the left.
  2. 180 degrees is full speed turning to the right.
- As you approach values towards 90 degrees from 0 or 180 degrees the motor will slow down.



Pic A



Pic B

3: Callibrating the servo.

Sometimes when the 360^0 is programmed to be at stationary, its shaft may shudder slightly. Use the following steps to callibrate a Servo to stop the shudder.

- Program the servo as before using ‘set servo _ to 90 degrees’.
- Upload this code to the micro:bit and if the servo is turning it needs to be calibrated.
- To do this use the screw at the back of the servo. (**Pic A**)
- Using a screwdriver, turn the screw until the motor stops turning, that is it now calibrated to the middle of the servo. (**Pic B**)
- From here create your code to work as you want to now

Module 3: DC Motor Control

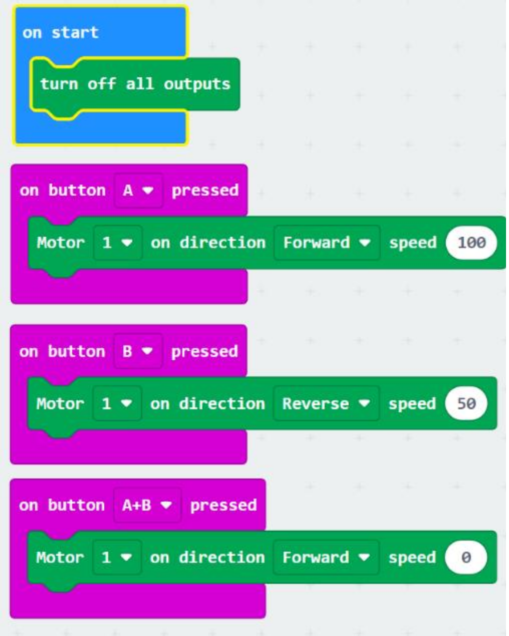
Video

https://www.youtube.com/watch?v=tHA3HGjslWE&feature=youtu.be

This module will introduce participants to programming DC Motors using block code on MakeCode. The ‘Robotics’ block must be added to the MakeCode menu to engage in this activity. Explore the following piece of code with the supporting video.

Activity A

Program a DC motor to prescribed constraints.



```
on start
    turn off all outputs

on button A ▾ pressed
    Motor 1 ▾ on direction Forward ▾ speed 100

on button B ▾ pressed
    Motor 1 ▾ on direction Reverse ▾ speed 50

on button A+B ▾ pressed
    Motor 1 ▾ on direction Forward ▾ speed 0
```

- Press A – Rotate forward full speed.
- Press B – Rotate backwards at half speed.
- Press A+B – Motor stopped.

❖ Download the program and simulate using the All-in-one-Robotics board.

Additional Resources from the Mechatronics Elective 2020

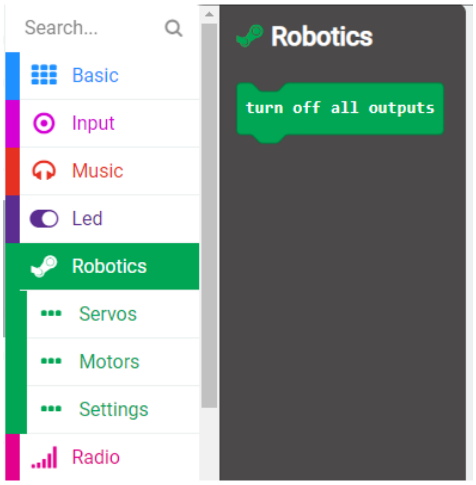
Video explaining the principles behind coding a DC motor

https://drive.google.com/file/d/1pBcswVMnD_HfbkPWe8MkzR3azz7Y4CUL/view

Video to a design challenge to apply the above code.

https://www.youtube.com/watch?v=ErpWsZ5Ef0Y&feature=youtu.be

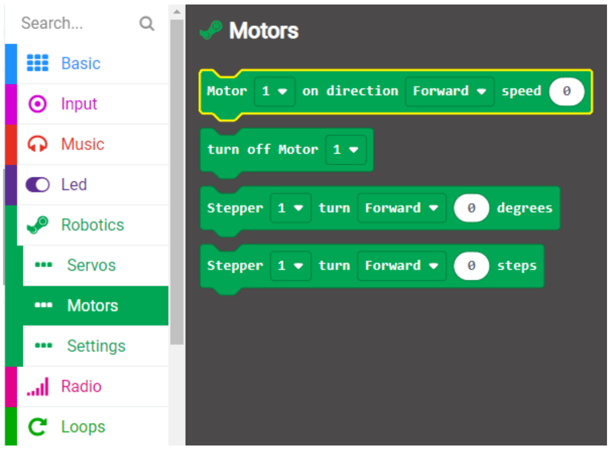
Watch the stimulus video attached to the QR code above.
How might you engage your students in this activity in Engineering?



```
turn off all outputs
```

1:

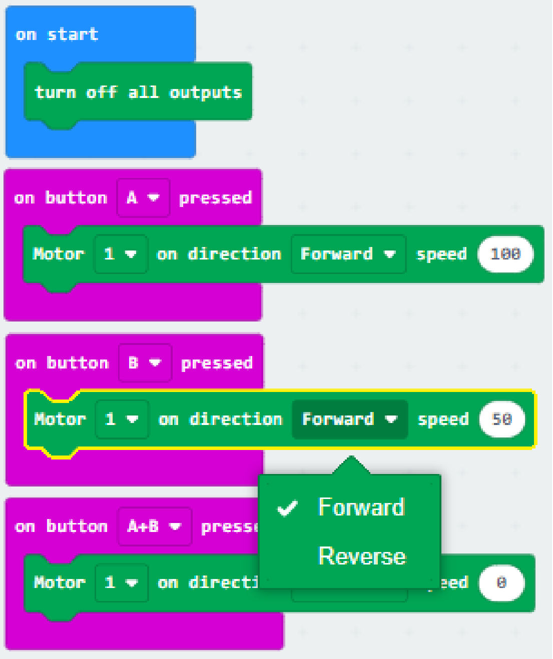
- When creating the code, we will need to make sure that the motors do not start turning when we power up the micro:bit so we use ‘basic’ and the ‘on start’.
- Using the ‘robotics’ tab, click on ‘robotics’ and you will see ‘turn of all outputs’.



```
Motor 1 ▾ on direction Forward ▾ speed 0
turn off Motor 1 ▾
Stepper 1 ▾ turn Forward ▾ 0 degrees
Stepper 1 ▾ turn Forward ▾ 0 steps
```

2:

- Using the ‘robotics’ tab click on ‘motors’.
- Here you will see the blocks for DC motors and Stepper motors.
- Using the ‘motor __ on direction ___ speed __’.
- As you can see, we can control which motor is turning, its turning direction and its speed.
- The speed is a percentage from 0-100 of the total power available to the motor. i.e.: 6V on the All-in-one robotics board will spin slower than 9V power to the board.



```
on start
    turn off all outputs
on button A ▾ pressed
    Motor 1 ▾ on direction Forward ▾ speed 100
on button B ▾ pressed
    Motor 1 ▾ on direction Forward ▾ speed 50
on button A+B ▾ presse
    Motor 1 ▾ on directi          eed 0
```

3:

- In order to change the direction of the motors, click on the drop-down arrow beside the direction.

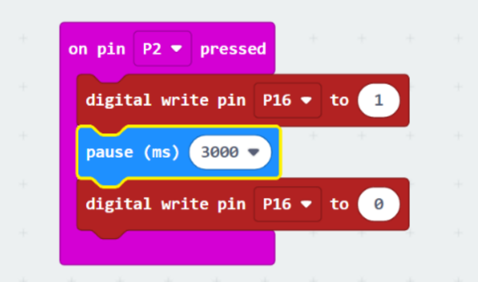
❖ The motors in a project, depending on which way they are wired, may spin in the opposite direction. You may need to run the code first to check this. Then change it to the appropriate direction in order to make them spin the same direction.

Module 4: Adding Inputs and Outputs

External LEDs and switches can be easily added to the All-in-one robotics board. Some components must be soldered to the robotics board in order to simulate any code developed in this activity. Activity: Build the given code with the support of the given video link. Reflect on how you might explore this code further with your students.

Video

https://youtu.be/0f85f1lf4VU



```
on pin P2 ▾ pressed
    digital write pin P16 ▾ to 1
    pause (ms) 3000 ▾
    digital write pin P16 ▾ to 0
```

This code is intended to fulfil the activity constraints.

What is involved in the code?

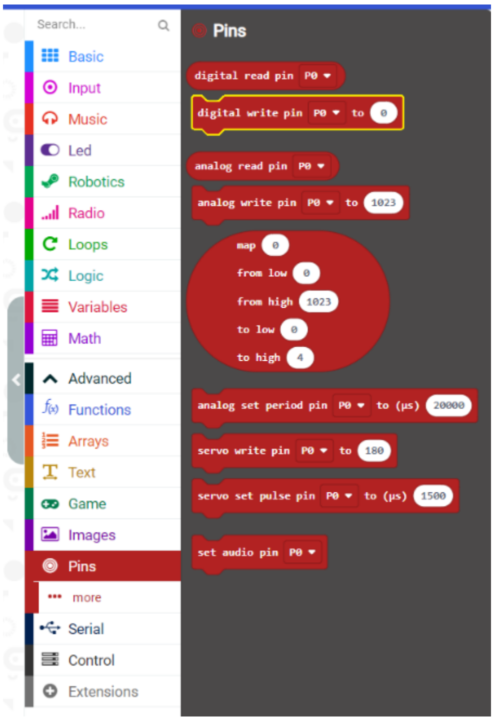
Input: Pressing buttons or giving signals to start a function such as shaking the board.

Basic: Simple functions such as pausing a code.

Pins: This controls the signals from the 'Link Pads' on the 'All-in-one Robotics Board'. This is where the switch and the LED are joined

1:

- Select 'on pin pressed' from the Input tab. Drag and drop it onto the programming editor.



Search...
Basic
Input
Music
Led
Robotics
Radio
Loops
Logic
Variables
Math
Advanced
Functions
Arrays
Text
Game
Images
Pins
more
Serial
Control
Extensions
Pins
digital read pin P0
digital write pin P0 to 0
analog read pin P0
analog write pin P0 to 1023
map 0 from low 0 from high 1023 to low 0 to high 4
analog set period pin P0 to (µs) 20000
servo write pin P0 to 180
servo set pulse pin P0 to (µs) 1500
set audio pin P0

2:

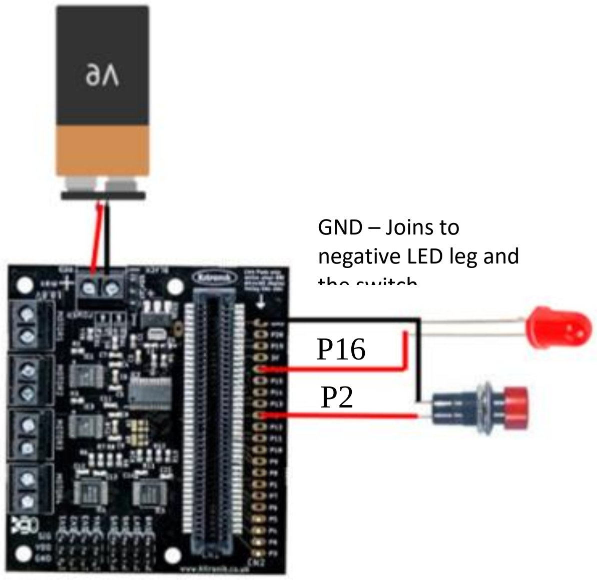
In the menu, click 'Advanced' to extend the menu, select 'Pins'. From the new menu, select 'digital write Pin P0 to 0' and drag and drop it into 'on pin P0 pressed' menu block'. Set P0 to P16 and ensure the output is set to 1.

3:

Return to the 'Basic' block and select 'pause (ms)'. Drop this into the 'on pin P0 pressed' menu block'. Set the pause to 3000ms.

4:

Select a second 'digital write Pin P0 to 0' from the menu or by duplicating the first block, and drag and drop it into 'on pin P0 pressed' menu block' under pause (ms)'. Set P0 to P16 and ensure the output is set to 0.



5:

Download the code to the Micro:bit and test the code using the configured components on the 'all-in-one Robotics Board.

How could this program be applied to an Engineering project?

Activity: Design problem
(Using knowledge built from the previous modules)

- In a model vehicle it is needed to indicate when the vehicle is slowing down or has come to a stop
- Develop the code relevant to replicate this in action

Notes:

Module 5: Servo Control using a Potentiometer

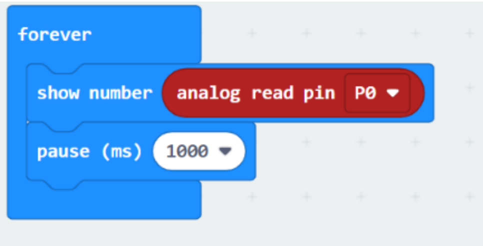
Activity A

Potentiometers are variable resistors. They are often applied in applications in Engineering to vary the resistance in a circuit and often, control the resistance applied to an output such as a motor to adjust speed and torque, and control the angle of a servo motor. In this first activity we will look at how to map a potentiometer to a microbit. This is required in instances where a potentiometer is required to control the angle of a servo. It is necessary as maximum resistances vary from one potentiometer to the next.

Video

https://youtu.be/tPjJrtGBUlM

Explore the following piece of code with the supporting video. The video will explain the context and application for this code, and it will form the basis for activity two in this module.



```
forever
    show number  analog read pin  P0 ▾
    pause (ms)  1000 ▾
```

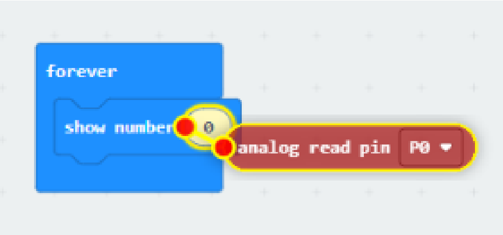
This code is useful to map any potentiometer to the micro:bit. It will allow the minimum resistance to be mapped to 0^0 and the maximum resistance to be mapped to 180^0 on the servo. The blocks required include:

Basic: Simple functions such as pausing a code, 'show number' and 'forever' block.

Pins: This controls the signals from the 'Link Pads' on the 'All-in-one Robotics Board'. This is required to send power to the potentiometer and to measure the 'signal' from the middle leg on the potentiometer.

1:

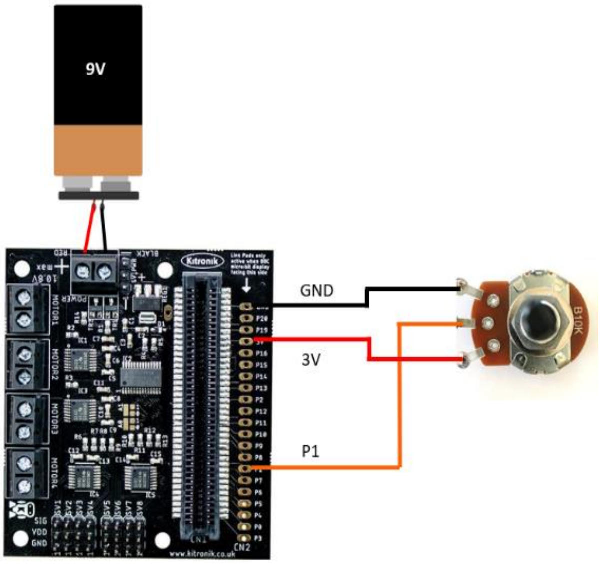
In the 'Basic' menu select 'show number' and drag and drop it into the 'forever' block.



```
forever
    show number  0
    analog read pin  P0 ▾
```

2:

In the menu, click 'Advanced' to extend the menu, select 'Pins'. From the new menu, select 'analog read Pin P0 ' and drag to the left of the 'show number' block until it is highlighted yellow. Release the mouse button to drop 'analog read Pin P0 ' into 'show number'. Change P0 to 1.



3:

In the 'Basic' menu select 'pause (ms)' and drag and drop it into the 'forever' block under 'show number'.

4:

Download the code to the microbit and test on the configured components. The highest resistance for this resistor should be approximately 1022 for this 10kΩ resistor. The lowest resistance should be a number close to 0.

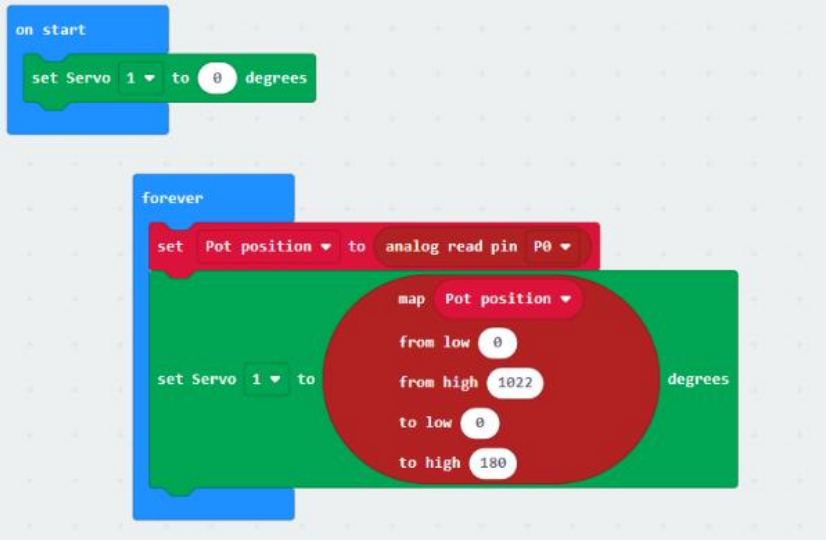
Activity B: Servo control using a Potentiometer

This activity explores code which can be applied in computer software to allow a Servo Motor to be controlled by a potentiometer. This may be applied to Mechanisms such as Linear Actuators. An understanding of the code will be required to engage in the additional code in this activity.

Explore the following piece of code with the supporting video. The video will explain the context and how the code is designed.

Video

https://youtu.be/DZFG7nO2TtA



```
on start
    set Servo 1 to 0 degrees

forever
    set Pot position to analog read pin P0
    set Servo 1 to  map Pot position        degrees
                    from low 0
                    from high 1022
                    to low 0
                    to high 180
```

How might Engineering students apply this code in their projects?

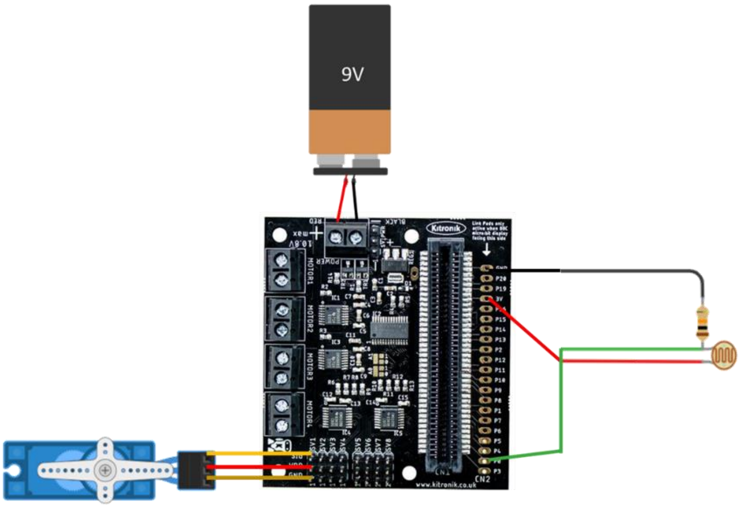
Activity: Design problem

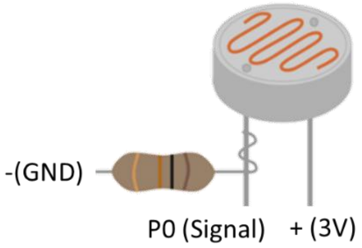
- Canals use a series of locks in order for boats to move through areas of elevation changes
- Create a simulation of the code needed to make the lock gates open and close

HINT: What angle do the levers need to rotate in order to fully open the locks?

Module 6: Sensors

Sensors such as LDRs and Thermistors can be added to the Robotics board as additional sensors. This activity explains how to add and code a LDR and apply it to a geared DC motor and a servo motor.



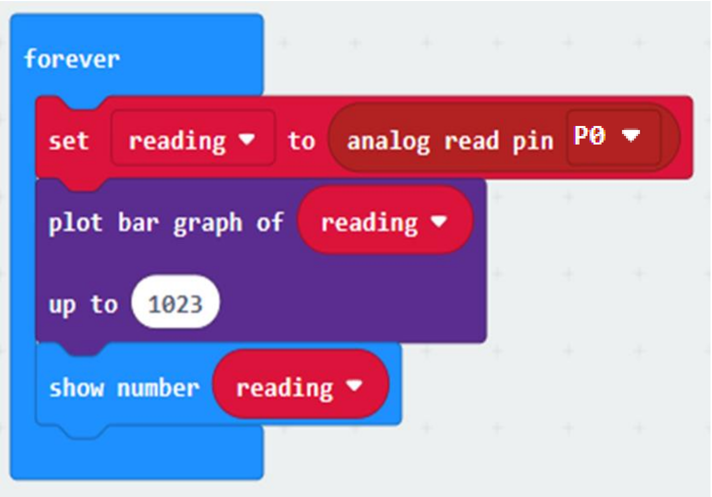


Activity A: Calibrating a sensor:

Video

Why do we calibrate sensors? When using sensors for any reason we should know it's maximum and minimum reading in a given situation as if we make it too sensitive it might not be sensitive enough or the opposite too sensitive. Use the video in the provided link for support.

https://youtu.be/DuBzrziPJso

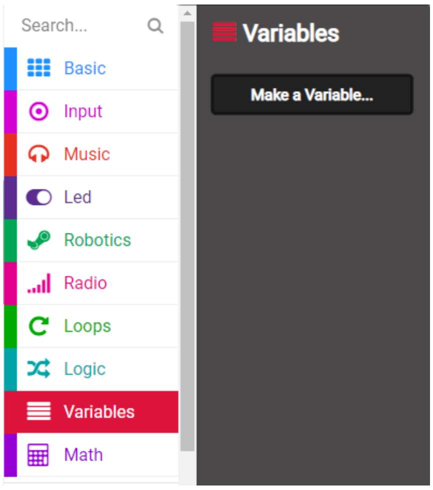


```
forever
    set reading ▾ to analog read pin P0 ▾
    plot bar graph of reading ▾
    up to 1023
    show number reading ▾
```

Using this code you will see the values coming from the sensor on the micro:bit screen

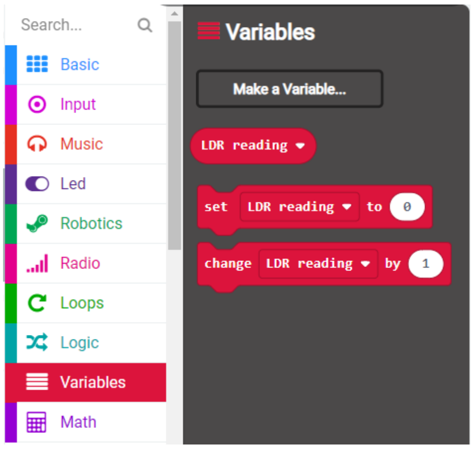
How might you engage your students in this activity in Engineering?

To calibrate the sensor:



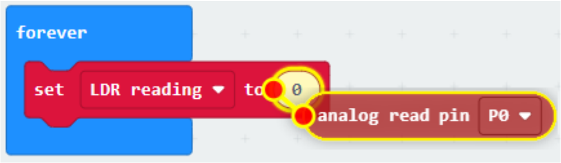
1:

- When we are taking a reading, we will need to create a variable.
- Click on the 'variables' tab and then the 'Make a Variable' button
- Just like in the subject maths, a variable can have any value.
- We will dictate where that variable comes from, in this case, the LDR



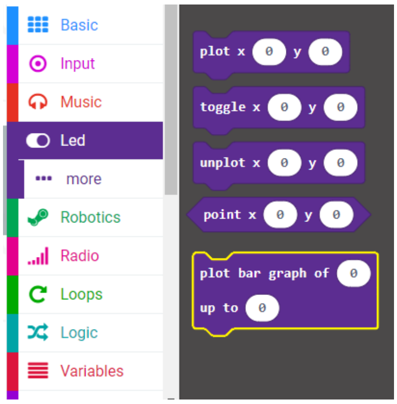
2:

- Name the variable something that will help make it identifiable, in this case LDR reading.
- When the variable is named three blocks appear.
- 'LDR reading' allows for the use of variable to show numbers
- 'set LDR reading to __' allows us to create the link to the analogue value coming from the LDR
- 'Change LDR reading by __' allows us to change the variable which can be useful for a countdown using a loop



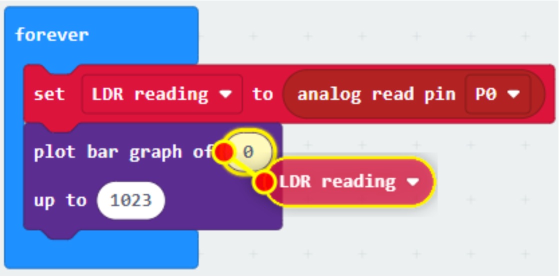
3:

- From 'basic' we use the 'forever' block as we would like this to run continuously in the background.
- Using the 'variables' tab we use 'set LDR reading to __'
- From the 'pins' tab we use the 'analog read pin __' and set it to whichever pin the potential divider is connected to. (check the instructions given to you with the 'mechatronics pack')
- By doing this we have said the reading of the LDR comes from the pin it's connected to.



4:

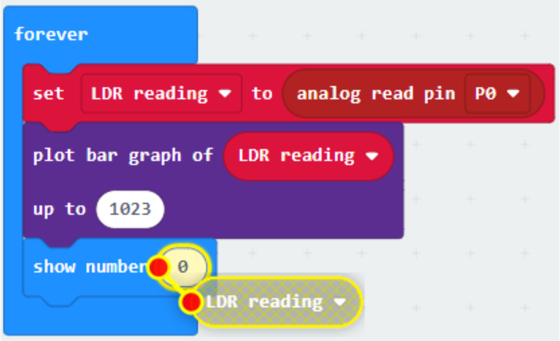
- From the 'Led' tab we can access the 'Plot bar graph of __ up to __'
- This will put a bar-chart on the LED screen and indicate how much of the resistance is left in the LDR
- The bar-chart will show up on screen before the reading of the LDR



```
forever
    set LDR reading ▾ to analog read pin P0 ▾
    plot bar graph of 0  LDR reading ▾
    up to 1023
```

5:

- Using the 'Variables' tab we can plot the 'LDR reading' as shown
- We set the reading up to 1023 as any analog sensor goes between 0 and 1023 in terms of values



```
forever
    set LDR reading ▾ to analog read pin P0 ▾
    plot bar graph of LDR reading ▾
    up to 1023
    show number 0  LDR reading ▾
```

6:

- Using the 'Basic' tab we can use 'show number __' to put the reading on the screen
- From 'Variables' we use 'LDR reading' to show the value coming from the LDR
- Now upload the code to the micro:bit and take the reading of the LDR in the brightest light and in dark conditions as this will be useful in the next activities

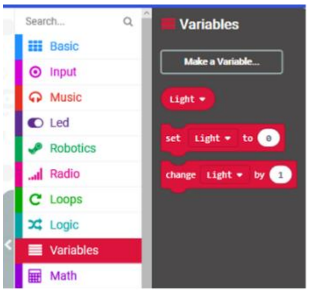
Reflection on Observations:

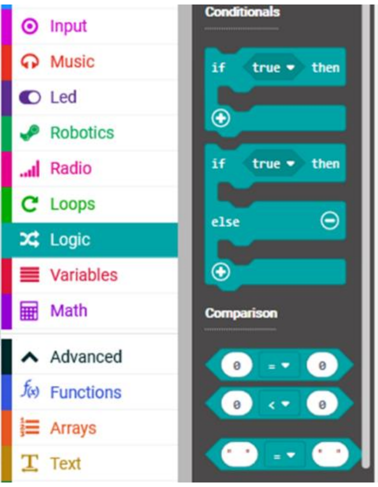
Activity B: using a sensor and DC motor

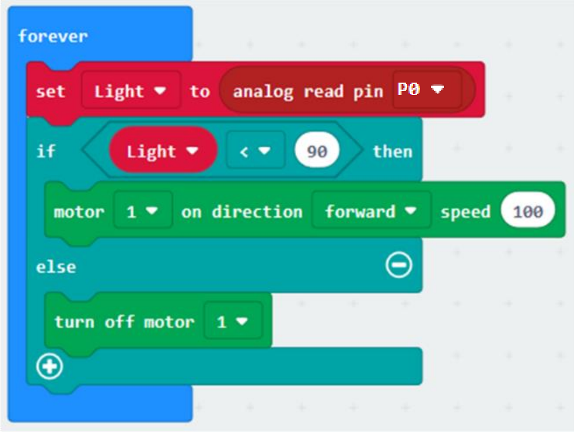
Using the coding blocks seen below, gives a basic way to turn on a DC motor when the light reading is below a set value. Using the calibration code above, could we improve the code? The video in the attached link will support you in this activity.

Video

https://youtu.be/TKpPK-RjGus

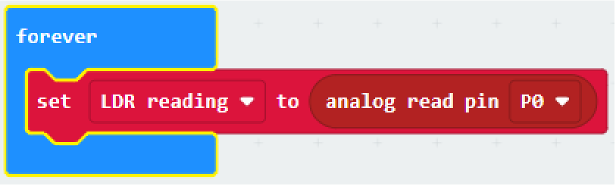






```
forever
    set Light to analog read pin P0
    if Light < 90 then
        motor 1 on direction forward speed 100
    else
        turn off motor 1
```

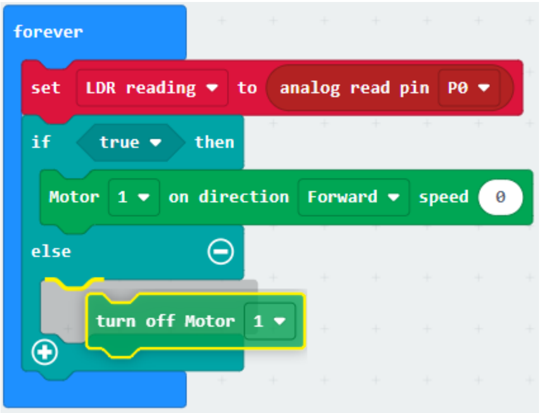
How might you engage your students in this activity in Engineering?



```
forever
    set LDR reading to analog read pin P0
```

1:

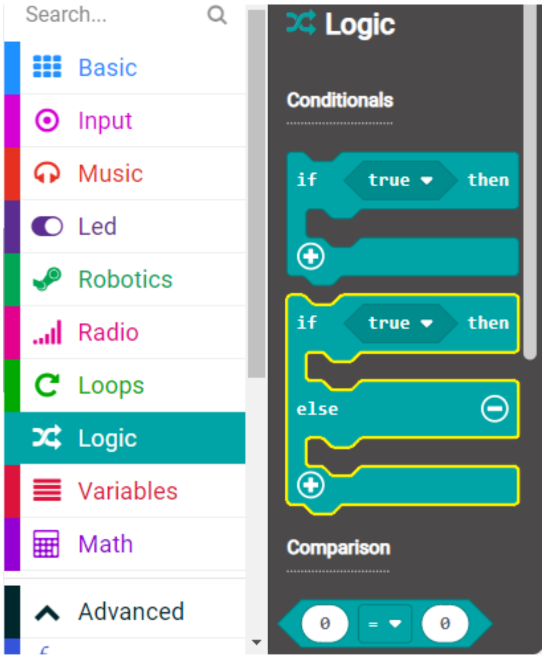
- Using the previous activity and the steps 1-3, recreate the code on the left



```
forever
    set LDR reading to analog read pin P0
    if true then
        Motor 1 on direction Forward speed 0
    else
        turn off Motor 1
```

2:

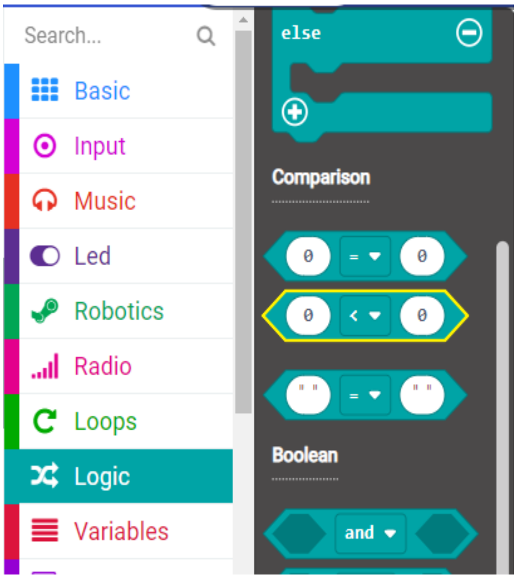
- Using the 'logic' tab we can create a scenario to occur
- If a reading of a value is present the component will be actioned, if not then something else will be actioned
- In this case we can create a code when the value is less than a reading from the LDR then turn on the motor based on the this



3:

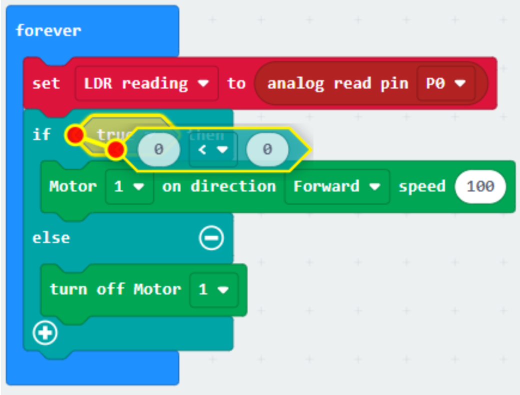
- In the 'if' part of the logic gate, use the 'robotics' tab and place in 'motor __ on direction __ speed __'
- Set the values and directions as desired
- In the 'else' section add 'turn off motor __'

*So far, we have said if something is present then turn the motor on and if it's not there, turn it off

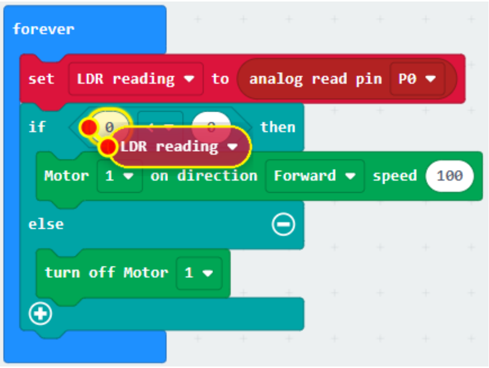


4:

- Using 'logic' we can create a comparison
- Add this to the logic gate as you can see in the bottom picture



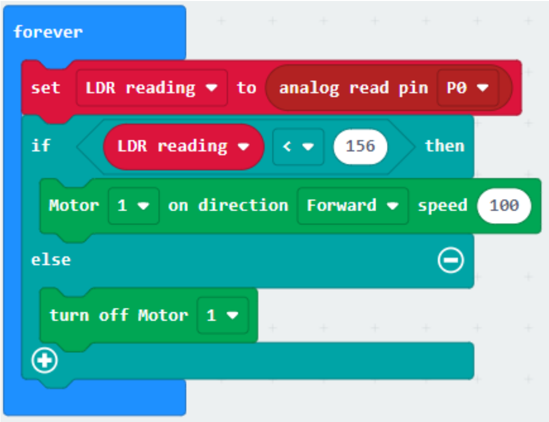
```
forever
    set LDR reading to analog read pin P0
    if 0 < 0 then
        Motor 1 on direction Forward speed 100
    else
        turn off Motor 1
```



```
forever
    set LDR reading to analog read pin P0
    if LDR reading < 0 then
        Motor 1 on direction Forward speed 100
    else
        turn off Motor 1
```

5:

- From 'Variables' bring the 'LDR reading' into the first part of the code
- Change the value of the < (less than) part to suit a value within your range of values taken from the previous activity



```
forever
    set LDR reading ▾ to analog read pin P0 ▾
    if LDR reading ▾ < ▾ 156 then
        Motor 1 ▾ on direction Forward ▾ speed 100
    else ⊖
        turn off Motor 1 ▾
    ⊕
```

6:

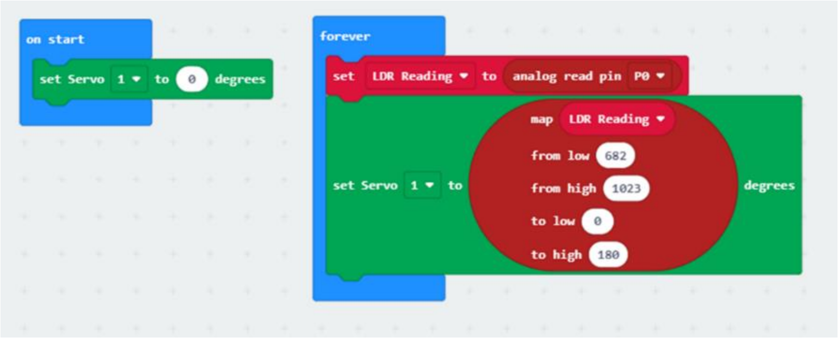
- Once this is done the motor will turn on when the LDR goes below that value and turn off when above that value
- You can swap this function by changing to the ‘>’ (greater to)

Activity C: using a sensor and a servo motor

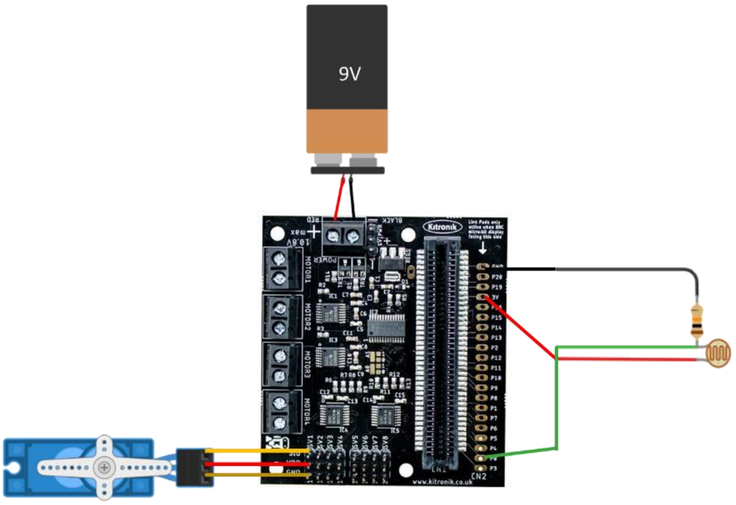
Video

https://youtu.be/4nTvfn4RrNw

Using the previous learning, we can calibrate the servo to move between two values and its sensitivity depends on the range of values from the LDR and the degrees we want to use.



```
on start
    set Servo 1 ▾ to 0 degrees

forever
    set LDR Reading ▾ to analog read pin P0 ▾
    set Servo 1 ▾ to map LDR Reading ▾
                         from low 682
                         from high 1023
                         to low 0
                         to high 180      degrees
```



How might you engage your students in this activity in Engineering?

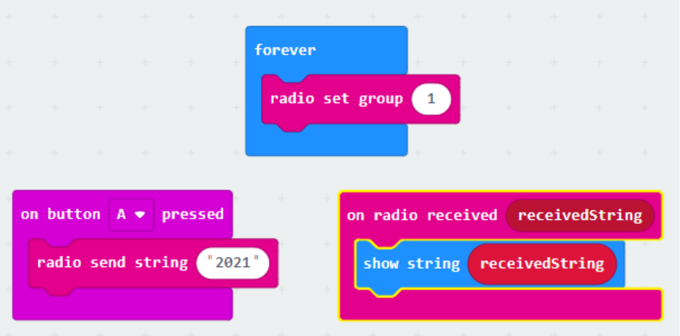
Module 7: Radio Control

Activity A: Radio Send String

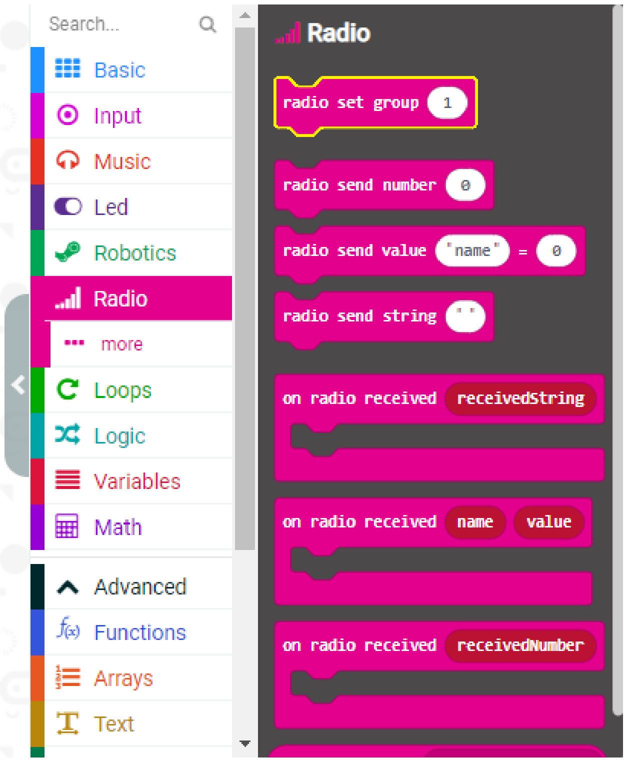
Explore the following piece of code which will allow two Micro:bits to send a ‘String’ from one Micro:bit to the next. Use the supporting video to guide you through this. It will help you to develop your understanding on the code.

Video

https://youtu.be/sisag7fvCcQ



```
forever
    radio set group 1

on button A pressed
    radio send string "2021"

on radio received receivedString
    show string receivedString
```



1:

Start with a ‘forever’ block in the program editor. In the ‘Radio’ tab select ‘radio set group’ and set the number to 1.

2:

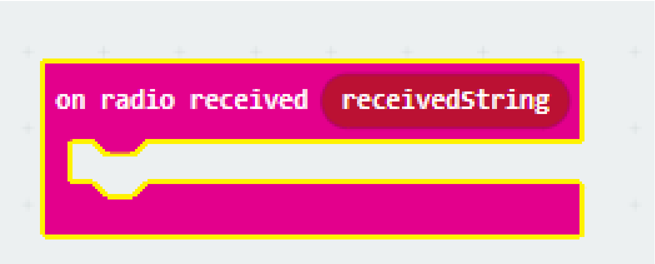
Drag and drop ‘on button A pressed’ into the program editor.

3:

Select ‘radio send string’ from the ‘Radio’ tab and drop it into the ‘on button A pressed’ block. Edit the text in the string to desired message to be communicated on the

4:

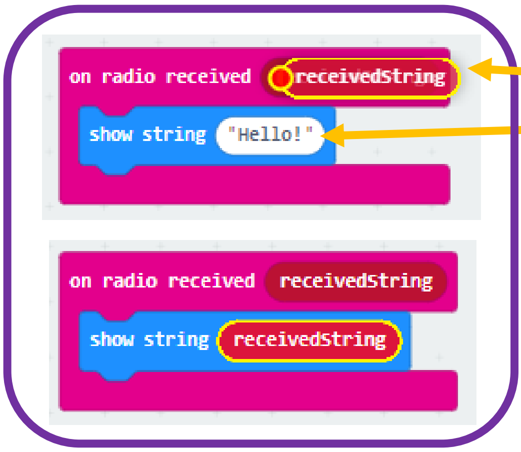
Select ‘radio send string’ from the ‘Radio’ tab and drop it into the ‘on button A pressed’ block. Edit the text in the string to desired message to be communicated on the micro:bit.



```
on radio received receivedString
```

5:

Select ‘on radio received receivedString’ from the ‘Radio’ tab and drop it into editor and drop ‘show string___’ from the ‘basic’ tab into the ‘on radio received’ block.



```
on radio received  receivedString
    show string  "Hello!"
```

```
on radio received  receivedString
    show string  receivedString
```

6:

In the 'on radio received' block, click the 'recievedString' block and drag and drop it into the 'show string' block.

7:

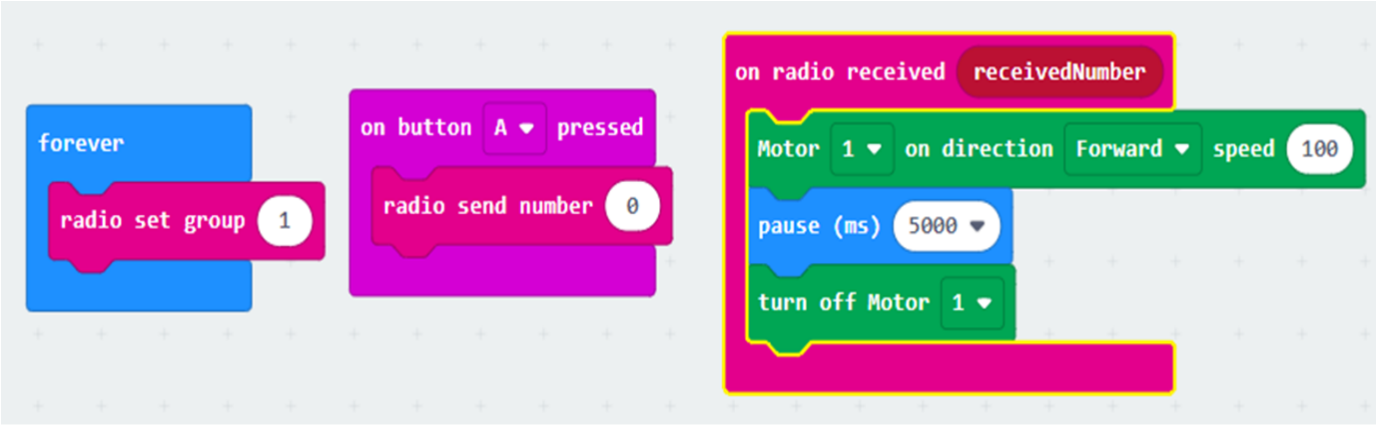
Download the code to **both** micro:bits. Press 'button A' on one micro:bit to send the code to the second micro:bit.

Activity B: Radio controlled DC motor

Explore the following piece of code. When button A is pressed, it will send a radio signal to a second Micro:bit. This will control a DC motor and will turn the motor on in the forward direction for 5 seconds. Use the supporting video to guide through this. It will help you to develop your understanding of the code.

Video

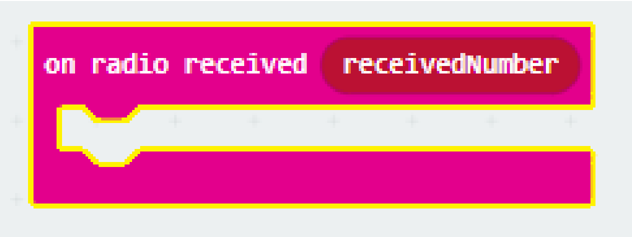
https://youtu.be/tAbFim3SZqM



```
forever
    radio set group  1

on button  A ▾  pressed
    radio send number  0

on radio received  receivedNumber
    Motor  1 ▾  on direction  Forward ▾  speed  100
    pause (ms)  5000 ▾
    turn off Motor  1 ▾
```

NOTE:

This code controls a DC motor. The relevant coding blocks can be found in the 'Robotics' tab. If starting with a new project you will need to add the 'Robotics' tab to the menu. This can be found by selecting 'Advanced' and then 'Extensions' in the menu and searching for 'All-in-one Robotics Board'.



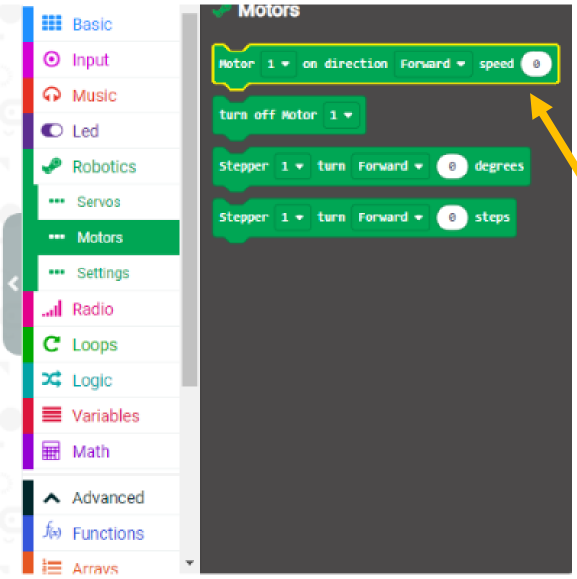
```
on radio received  receivedNumber
```

1:

Start with a 'forever' block in the program editor. In the 'Radio' tab select 'radio set group' and set the number to 1.

2:

From the 'Input' tab select 'on button A pressed' and drop into the editor. Select 'radio send number' from the radio tab and drop it into 'on button A pressed' block.



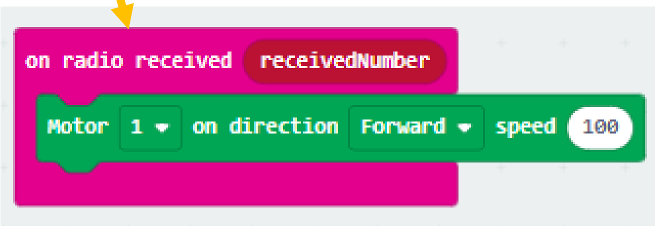
```
Motor 1 ▾ on direction Forward ▾ speed 0
turn off Motor 1 ▾
Stepper 1 ▾ turn Forward ▾ 0 degrees
Stepper 1 ▾ turn Forward ▾ 0 steps
```

3:

Select 'on radio received receivedNumber' from the 'Radio' tab and drop it into editor.

4:

In the 'Robotics' tab, select 'Motors', and from here drap and drop 'Motor __ on direction Forward speed __' into the 'on radio received receivedNumber' block as seen below. Change the speed from 0 to 100.



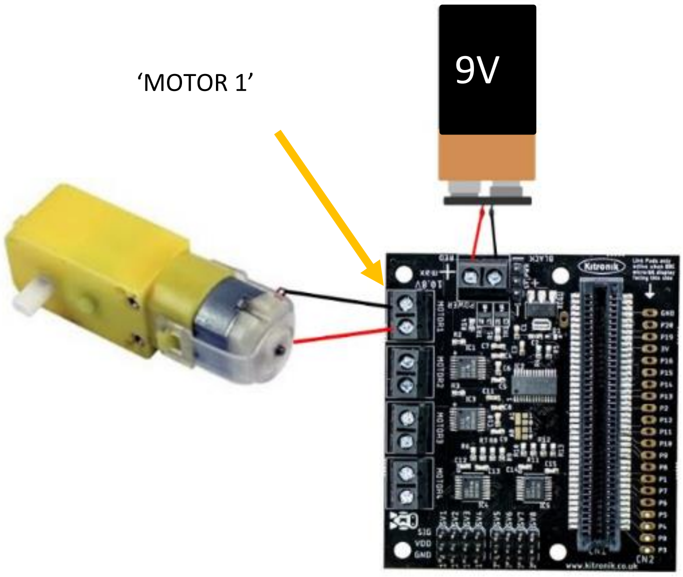
```
on radio received receivedNumber
    Motor 1 ▾ on direction Forward ▾ speed 100
```

5:

In the 'Robotics' tab, select 'Motors', and from here drag and drop 'Motor __ on direction Forward speed __' into the on radio received receivedNumber' block. Change the speed from 0 to 100.

6:

In the 'Basic' tab select the 'Pause (ms) __' block and drop it under the 'Motor __ on direction Forward speed __' block.



7:

Click the 'Robotics' tab in the menu. Select 'turn off all outputs'. Drag and drop it under the 'Pause (ms) __' block.

8:

Download the code to both micro:bits to test it on the All in one robotics board.

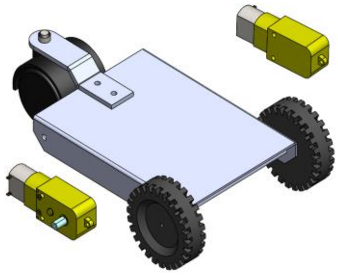
How might code such as this deepen a student's understanding of Mechatronics in their projects ?

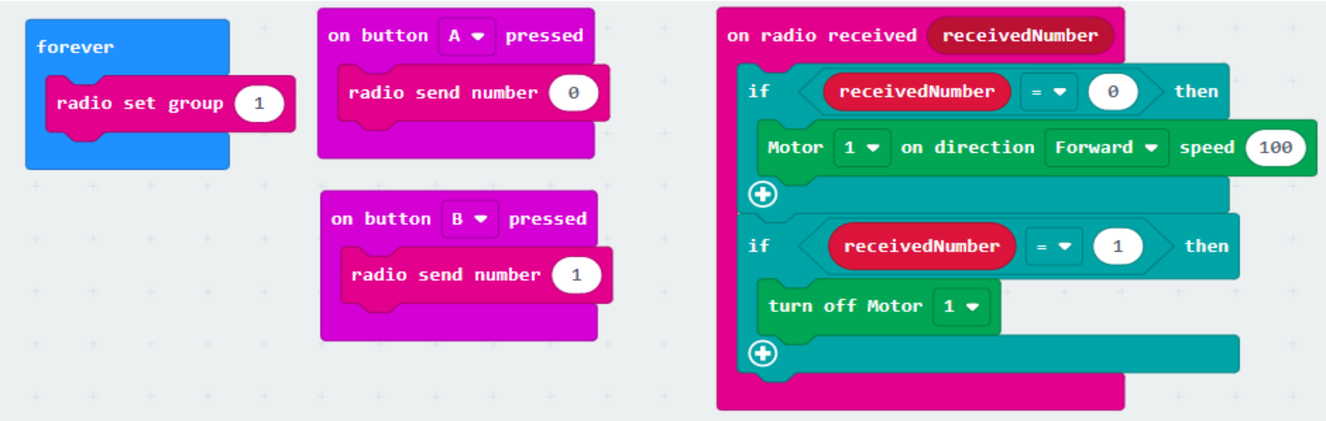
Activity C: remote control of a DC motor

Using the following code as your base you can control the motor using a combination of inputs and sending different values to the micro:bits. Creating different situations using logic gates can allow different control of the motors or any other components.

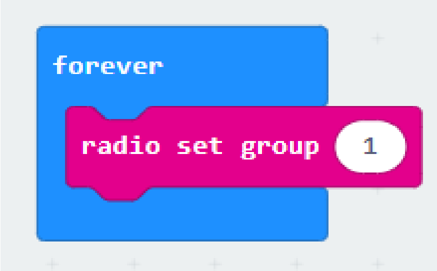
Video

https://youtu.be/YVIOk30kuEw





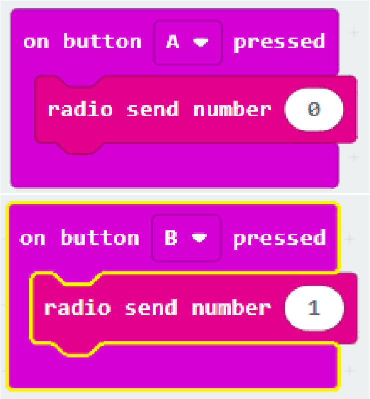
```
forever
    radio set group 1

on button A pressed
    radio send number 0

on button B pressed
    radio send number 1

on radio received receivedNumber
    if receivedNumber = 0 then
        Motor 1 on direction Forward speed 100
    if receivedNumber = 1 then
        turn off Motor 1
```



```
forever
    radio set group 1
```

1:

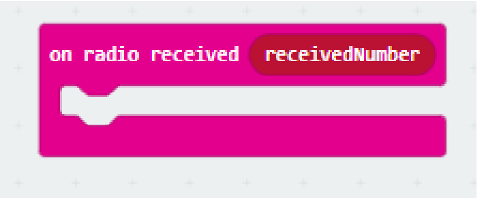
Using your understanding from previous activities, build this block of code and set the 'radio set group __' to '1'



```
on button A pressed
    radio send number 0

on button B pressed
    radio send number 1
```

2:

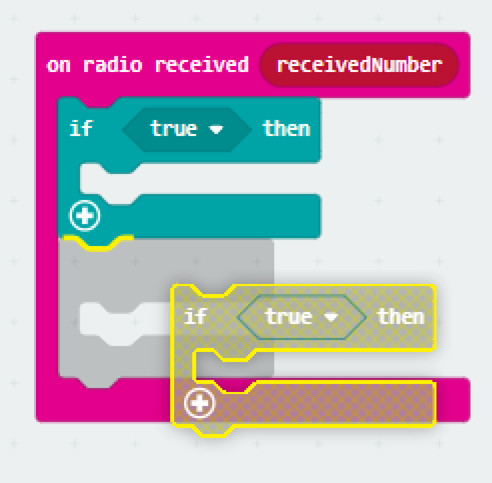
Build two input blocks to send a number using a radio signal as shown opposite. Set the first the first to send the number '0' when button 'A' is pressed (This signal will be used to rotate the motor forward). Set the second block to send the number '1' when button 'B' is pressed (This signal will be used to turn the motor off'.



```
on radio received receivedNumber
```

3:

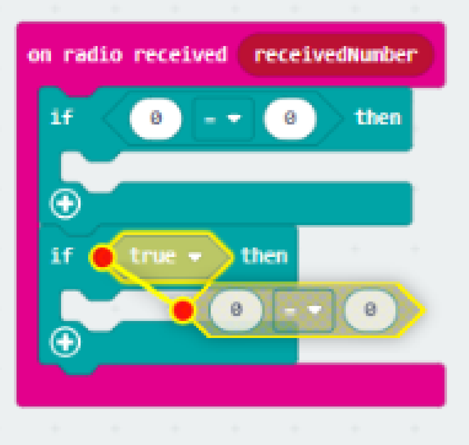
For the final part of the code start with an 'on radio received recievedNumber' block.



```
on radio received receivedNumber
    if true then

    if true then

```

4:

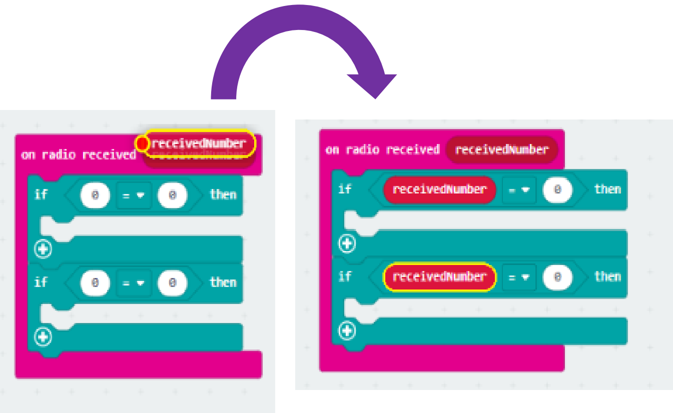
Add the 'if +' conditional block from the 'logic' tab and duplicate it in the 'on radio received' block. This will be used to compare the signal that is sent by button 'A' or 'B' to decide the output.



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

    if true then
        0 = 0
```

5:

Add an '=' 'comparison' block to both of the 'Conditional' blocks.

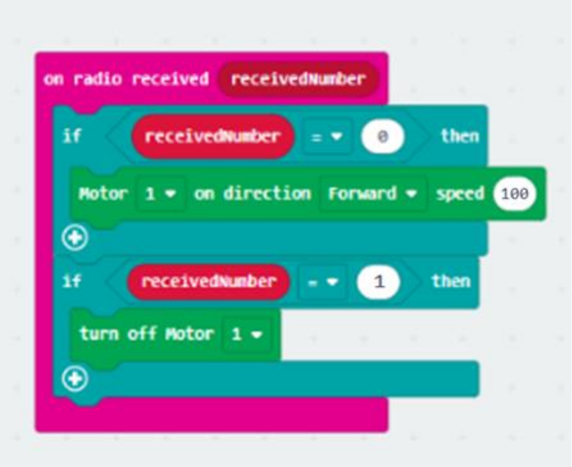


```
on radio received receivedNumber
    if 0 = 0 then

    if 0 = 0 then

```

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

    if receivedNumber = 0 then

```

6:

Select 'recieveNumber' in the 'Radio' block in this code and drop it into each of the 'Conditional' blocks. This will allow the conditional block to compare the number received to '0' or '1'. On this basis we can allow the code to determine the output.



```
on radio received receivedNumber
    if receivedNumber = 0 then
        Motor 1 on direction Forward speed 100

    if receivedNumber = 1 then
        turn off Motor 1

```

7:

- From the 'Robotics' tab add 'motor __ on direction __ speed __' to the first 'Conditional' block. Set the values to achieve the following logic: If the received number '=' 0, then the motor in pin 1 will rotate forward full speed.
- From the 'Robotics' tab add 'turn off motor __' to the second 'Conditional' block. Set the values to achieve the following: If the number received '=' 1, turn the motor in pin 1 off.
- Download and test the code on the configured components.

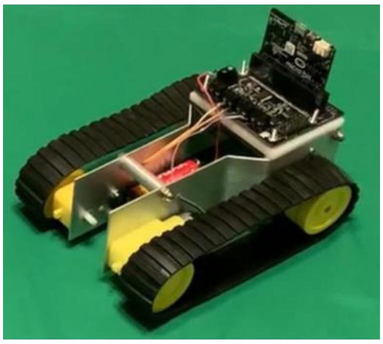
Module 8: Motor Control

Video

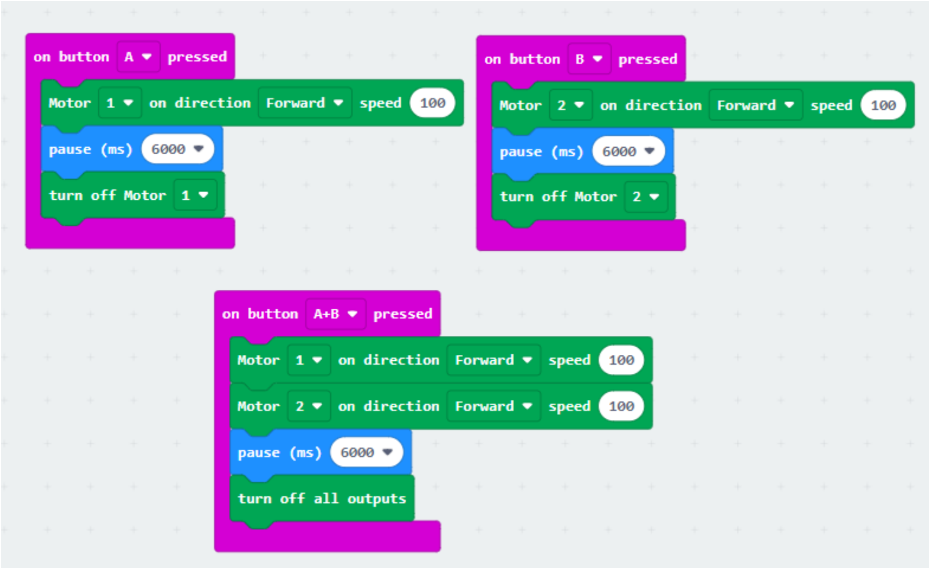
Activity A: DC Motor Control for a Specified Time

In this activity we are going to react to an on-going question from teachers around motor control: When programming a motor, how can I make the motors turn on for a specific amount of time?

In this module we will address this specifically as well as some other areas about motor control. Click or scan the QR code to view the supporting video.



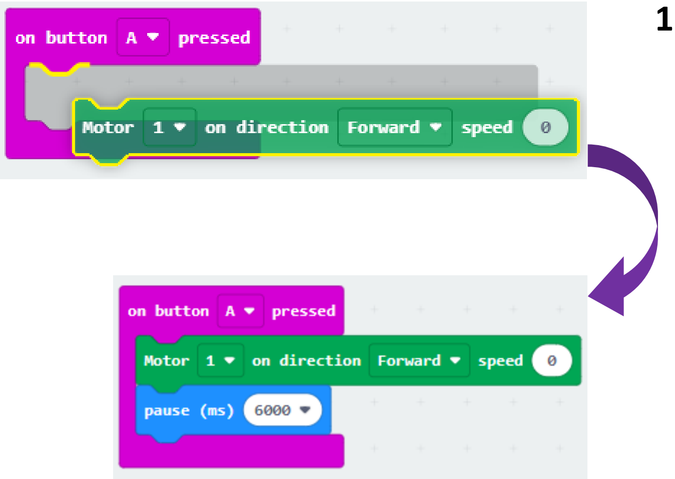
During the CPD session 2020/2021, we set the challenge to program the track vehicle in the picture. This was achieved by most teachers in the CPD. There have been follow-up emails asking for more engagement with the solution.



```
on button A pressed
    Motor 1 on direction Forward speed 100
    pause (ms) 6000
    turn off Motor 1

on button B pressed
    Motor 2 on direction Forward speed 100
    pause (ms) 6000
    turn off Motor 2

on button A+B pressed
    Motor 1 on direction Forward speed 100
    Motor 2 on direction Forward speed 100
    pause (ms) 6000
    turn off all outputs
```

The code seen above will solve this activity. It uses 'input' 'basic' and 'robotics' blocks in order to complete its commands. The 'robotics' extension needs to be added to complete this code. This code works by pressing the buttons for each command. Later in this activity you will be shown how to create the program that runs automatically with the press of one button.

1:



```
on button A pressed
    Motor 1 on direction Forward speed 0
```

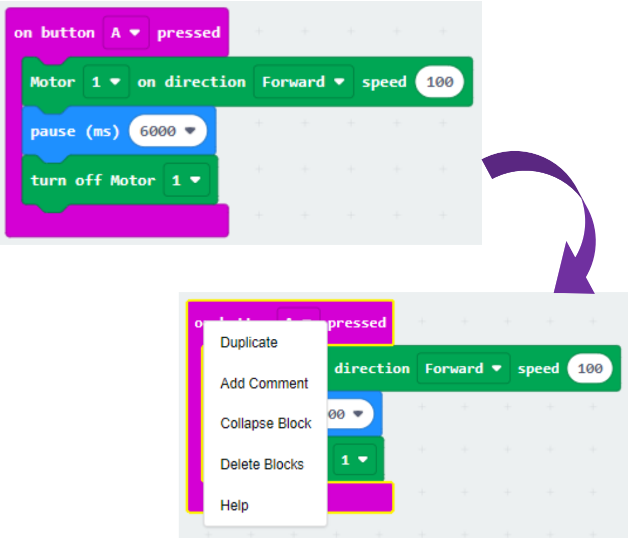
```
on button A pressed
    Motor 1 on direction Forward speed 0
    pause (ms) 6000
```

- Using the 'input' tab, select the 'on button __ pressed'
- Once the 'Robotics' extension has been added, use the 'motor __ on direction ___ speed ___' to turn on the motor and decide on a speed to select from 0-100
- Using the 'basic' tab, use the 'pause' block and choose a certain time to keep the motor running for

*the pause block can allow a command to run for a certain time and also can be a physical pause

** a second in this block is in terms of 1000's. 6000 is 6 seconds and so on

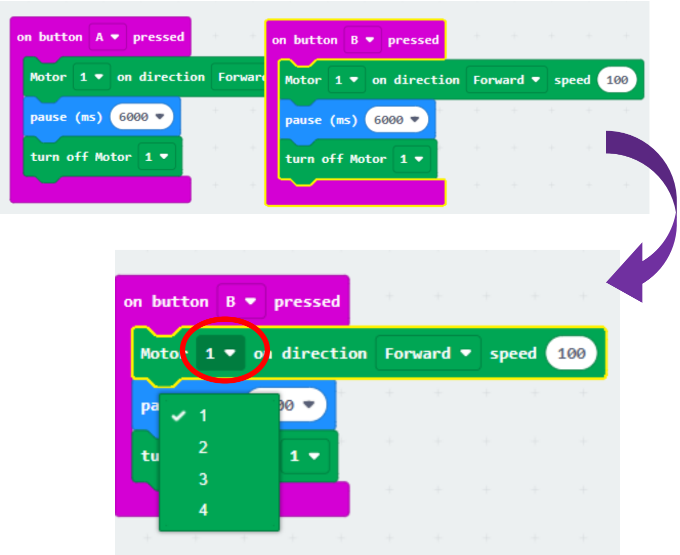
2:



```
on button A pressed
    Motor 1 on direction Forward speed 100
    pause (ms) 6000
    turn off Motor 1
```

- Using the 'robotics' tab we add in the block from the motors option to 'turn off motor __'
- Without using this block the motor will continue to turn and not stop until the motor is told to, even if you tell another motor to turn on in the meantime
- By right clicking on the 'input' block, you can click 'duplicate' and copy the entire set of blocks

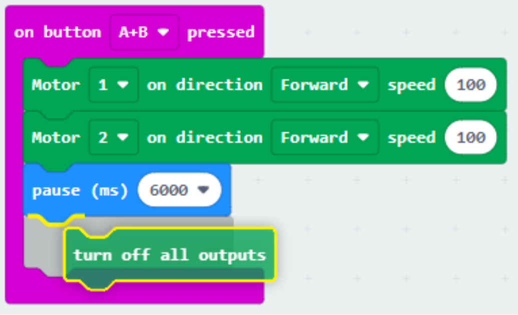
3:



```
on button A pressed
    Motor 1 on direction Forward
    pause (ms) 6000
    turn off Motor 1

on button B pressed
    Motor 1 on direction Forward speed 100
    pause (ms) 6000
    turn off Motor 1
```

```
on button B pressed
    Motor 1 on direction Forward speed 100
```

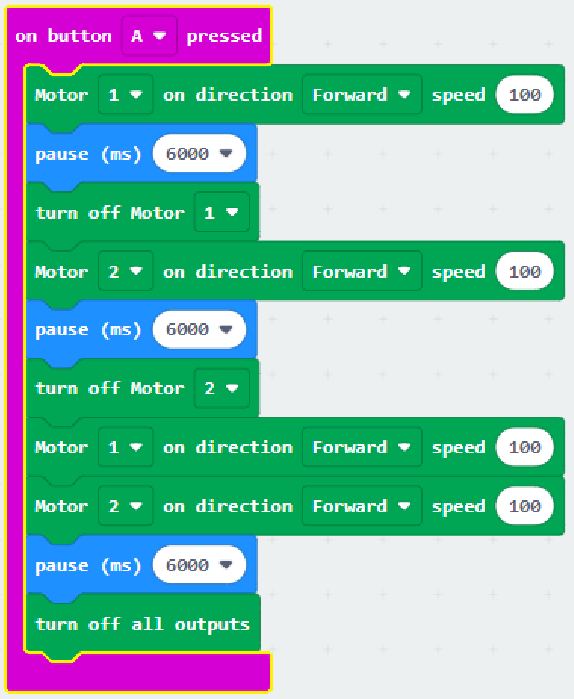
- Using the drop-down-arrow we can change the 'input' command to 'on button B pressed'
- Using the 'motor' block we can change the motor we want to turn by selecting the appropriate terminal block 1, 2, 3, or 4 from the drop-down menu
- Once again, we need to tell this motor to stop, otherwise it will continue to turn indefinitely, so change the number in the 'turn off motor __' to the same corresponding motor as the previous bullet point



```
on button A+B pressed
    Motor 1 on direction Forward speed 100
    Motor 2 on direction Forward speed 100
    pause (ms) 6000
    turn off all outputs
```

4:

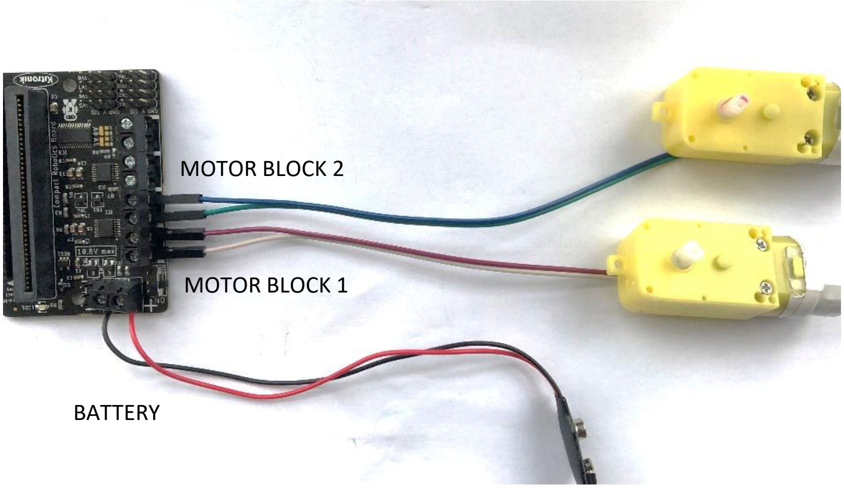
- By right clicking on the 'input' block you can click 'duplicate' and copy the entire set of blocks
- Using the drop-down-arrow we can change the 'input' command to 'on button A+B pressed'
- Using the 'robotics' tab we can add the first motor to turn as shown in step 1
- Using the original 'robotics' tab we can add 'turn off all outputs' to turn off all motors at the same time.



```
on button A pressed
    Motor 1 on direction Forward speed 100
    pause (ms) 6000
    turn off Motor 1
    Motor 2 on direction Forward speed 100
    pause (ms) 6000
    turn off Motor 2
    Motor 1 on direction Forward speed 100
    Motor 2 on direction Forward speed 100
    pause (ms) 6000
    turn off all outputs
```

5:

- It has also been asked 'how could I make the movement automatic?'- when I press button A I want the vehicle to move left and right, then go straight, by itself
- By combining all of the code seen in the previous steps together under the one input, the vehicle will move automatically and stop after 18 seconds using the 'pauses' show on the left



6:

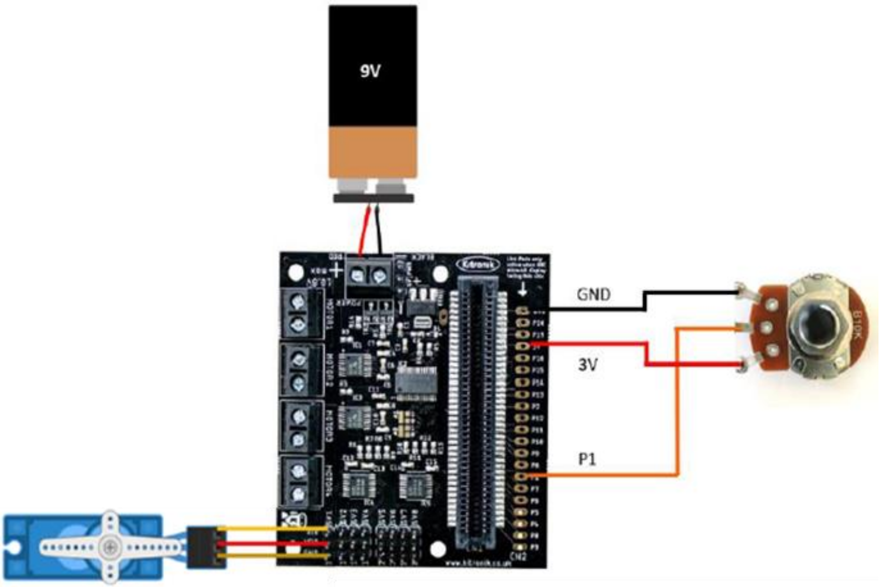
- Configure both motors as shown in the image
- Download and test the code

Note: The codes can be downloaded at the end of each of the previous 5 steps of building the code in this activity. This will build your understanding of each part of the code.

Activity B: Creating a hydrostatic servo motor control (One direction only)

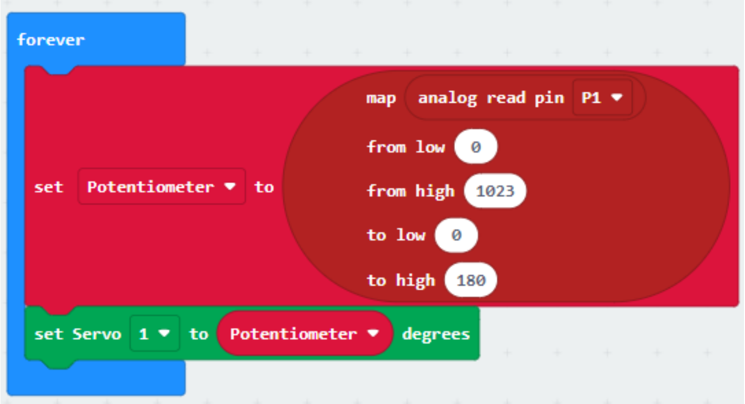
Video

This code aims to mimic a hydrostatic power drive.
In module 2 of the 2020/2021 learning log, it showed how to map a 180-degree servo to the potentiometer. In this module we will show how to use the potentiometer to set the direction and the speed of a continuous servo.
The diagram below shows how to connect the servo to the all-in-one robotics board.
Click or scan the QR code to view the supporting video.



1. GND is connected to one leg on the outside of the potentiometer
2. 3V is connected to the other outside leg
3. The middle leg can be connected to either P0, P1, or P2 to receive a signal

**connecting the outside legs to the board in the opposite way will only affect polarity of the potentiometer, the middle leg must be connected to an analog input



```
forever
    set Potentiometer to map analog read pin P1
                         from low 0
                         from high 1023
                         to low 0
                         to high 180
    set Servo 1 to Potentiometer degrees
```

This code is useful to map any potentiometer to the servo. It will allow the servo to be controlled, in terms of speed and direction by the potentiometer.

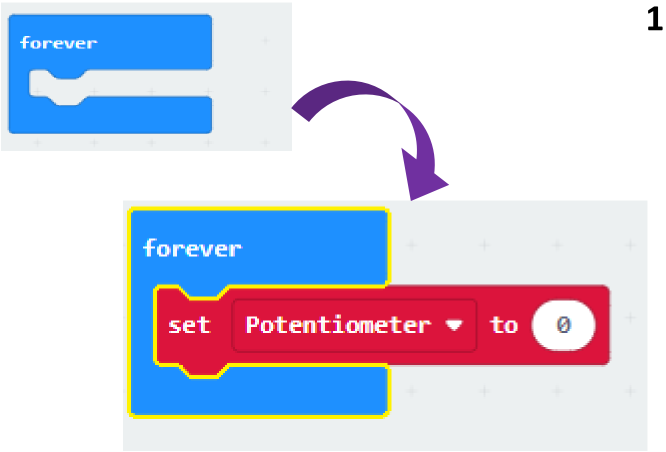
Click or scan the QR code opposite to guide you through the process of mapping the potentiometer to the Servo.

The blocks required include:
Basic: Simple functions such as ‘show number’ and ‘forever’ block.
Pins: This controls the signals from the ‘Link Pads’ on the ‘All-in-one Robotics Board’. This is required to send power to the potentiometer and to measure the ‘signal’ from the middle leg on the potentiometer. This allows us to control motors and servos.

Video – Calibrating the potentiometer

1:

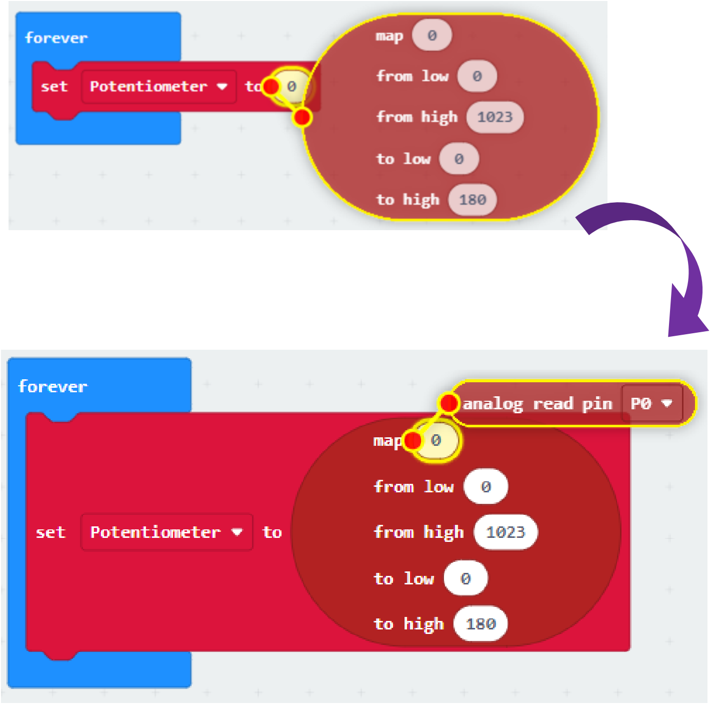
- Using the ‘variables’ tab, create a variable that we can use to set the value of the potentiometer, add it to the ‘forever’ block
- From here we can create whatever variable we so wish and get the reading from the analog sensor, in this case the potentiometer



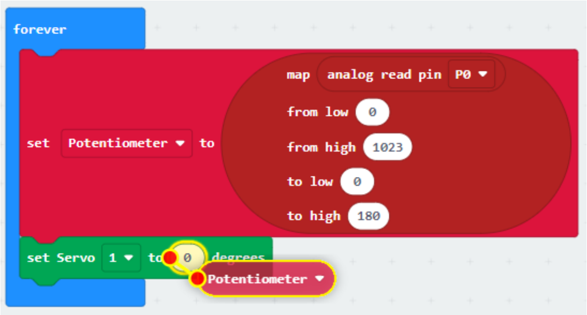
```
forever

forever
  set Potentiometer ▾ to 0
```

2:

- Using the ‘pins’ tab we set the potentiometer to take the reading from whichever pin you have connected the potentiometer to
- We use the ‘map’ block to create a scenario where it is taking a reading from a resistance from 0-1023 and it converts it to degrees from 0-180
- If you wish to make the range of readings smaller and more accurate you can use the QR code at the bottom of page 22 to navigate to a video which will provide help to calibrate the potentiometer
- Using ‘pins’ again we use the analog read pin p__’ to identify what pin the reading is to come from, once again making sure the middle leg of the potentiometer is connected to this pin



```
forever
  set Potentiometer ▾ to 0
    map 0
      from low 0
      from high 1023
      to low 0
      to high 180

analog read pin P0 ▾
forever
  set Potentiometer ▾ to map 0
      from low 0
      from high 1023
      to low 0
      to high 180
```



```
forever
  set Potentiometer ▾ to  map  analog read pin  P0 ▾
                               from low  0
                               from high 1023
                               to low  0
                               to high  180
  set Servo 1 ▾ to 0 degrees
                   Potentiometer ▾
```

3:

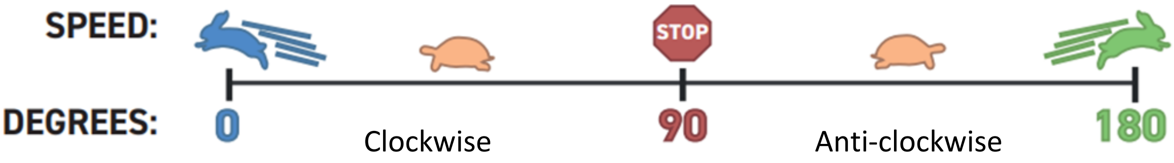
- From the ‘robotics’ tab we use the ‘set servo __ to __ degrees’ which will cause the servo to turn on
- From the ‘Variables’ tab we use the ‘potentiometer’ block. This will align the angle of the servo to the reading on the potentiometer.

**This means that at halfway of the potentiometer’s resistance will be 90°, and at either end of its rotation will be 0° or 180°

4:

- Download and test the code

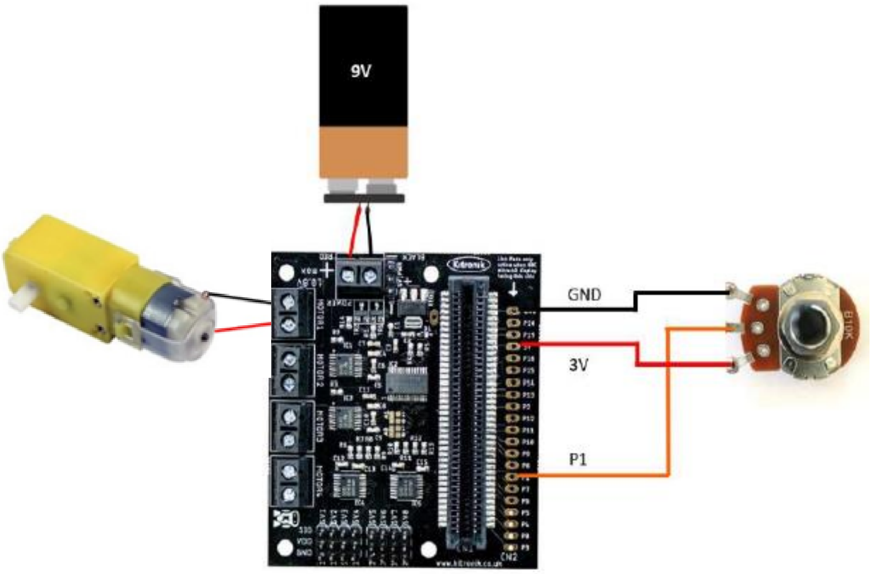
From the previous learning log, we know that with a continuous servo, the following diagram of degrees mapped to speed and direction will work with the servo:



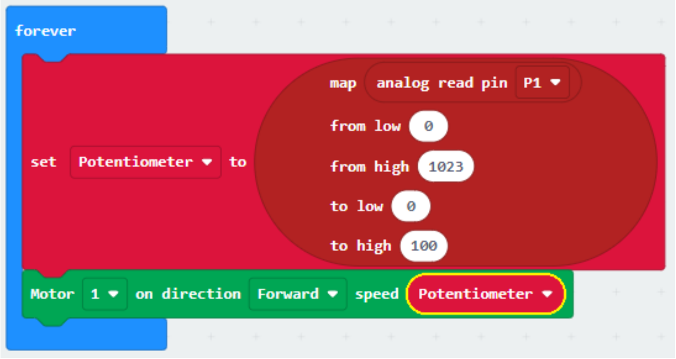
Activity C: Controlling a DC motor using a Potentiometer (One direction only)

Video

This code aims to mimic a hydrostatic power drive.
In the previous activity we have created a hydrostatic servo motor.
In this module we will create a hydrostatic DC control system.



The set-up is the same as before, instead of using the servo, connect the motor to the terminal block



```
forever
    set Potentiometer ▾ to  map  analog read pin P1 ▾
                                 from low 0
                                 from high 1023
                                 to low 0
                                 to high 100
    Motor 1 ▾ on direction Forward ▾ speed Potentiometer ▾
```

1:

- Looking at the solution on the left we can see many similarities to the previous activity.

- There are minor changes that we can see in the code. Firstly, the mapping has changed: instead of 'to low 0 - to high 180' we now see it has become from 'to low 0 – to high 100'

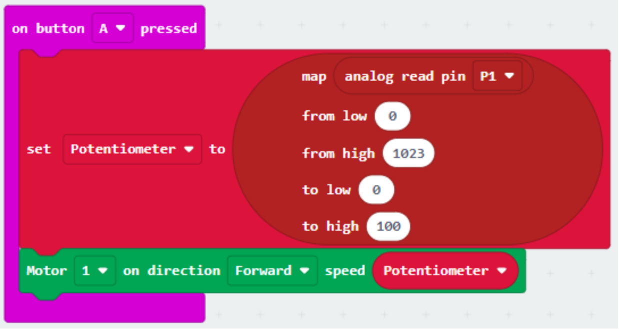
- This is to acknowledge that the motor control is from a speed of 0 – 100

- Similarly, we use the 'robotics' tab to find the motor block and add the potentiometer variable from the 'variables' tab

Note: As before, if you wish to calibrate the sensor and make it more sensitive then you can use this video. Click or scan the QR code to view the support video

Video

Activity 3a: Controlling a DC motor using a Potentiometer



```
on button A ▾ pressed
    set Potentiometer ▾ to  map  analog read pin P1 ▾
                                 from low 0
                                 from high 1023
                                 to low 0
                                 to high 100
    Motor 1 ▾ on direction Forward ▾ speed Potentiometer ▾
```

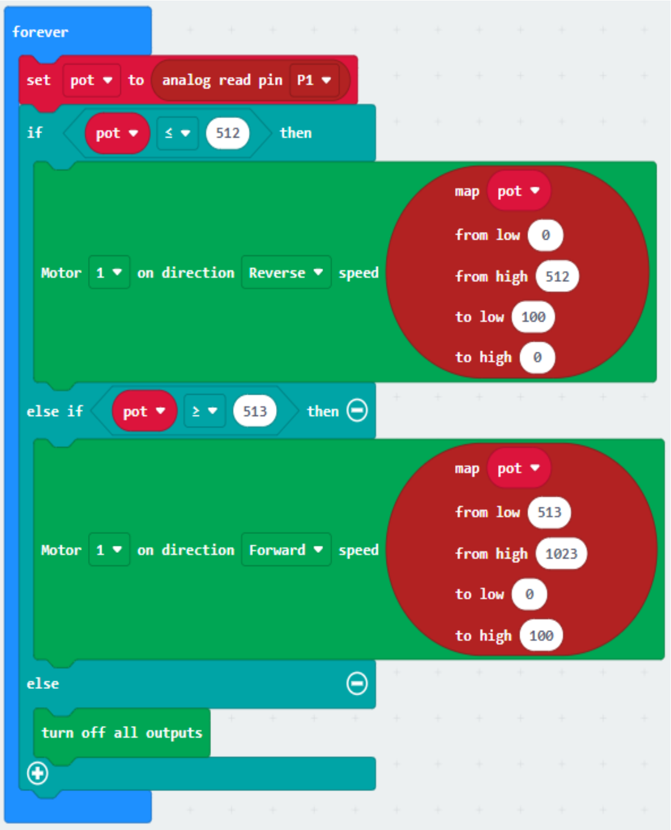
1:

- If you wish to have more control over the motor, you can use the 'input' tab and add it to the already developed code as shown.

- This now starts the motor on button A and then you can control the speed using the potentiometer

- Button A will need to be pressed, each time the potentiometer is adjusted.

Activity D: Controlling a DC motor using a potentiometer (Forward and Reverse)



```
forever
    set pot to analog read pin P1
    if pot ≤ 512 then
        Motor 1 on direction Reverse speed map pot
                                           from low 0
                                           from high 512
                                           to low 100
                                           to high 0
    else if pot ≥ 513 then
        Motor 1 on direction Forward speed map pot
                                           from low 513
                                           from high 1023
                                           to low 0
                                           to high 100
    else
        turn off all outputs
```

Video

This code aims to mimic a hydrostatic power drive.

The code on the left develops on the skills learned so far. Using mapping, motor direction and in the code seen here logic too.

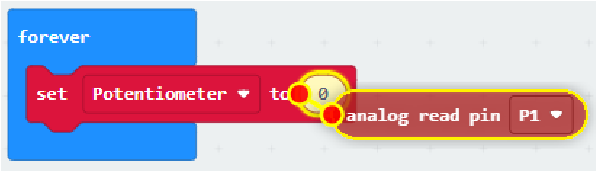
Basic: allows us to have the code running forever

Logic: allows us to create a scenario or a comparison that if one thing present it will do something and if not another.

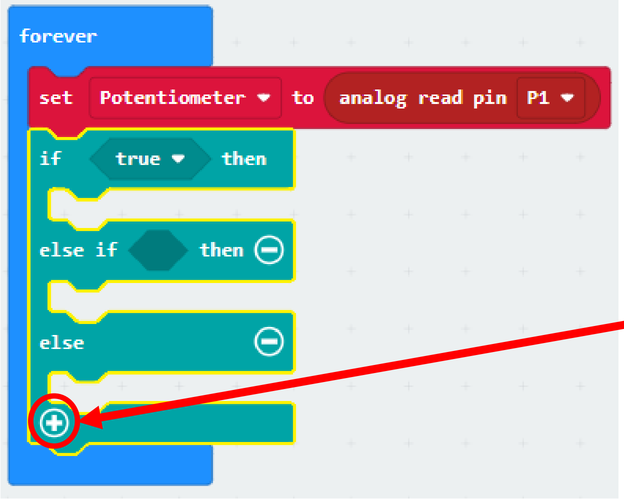
Robotics: motors can be controlled from this tab, but the extension needs to be added.

Pins: allows us to allocate control to a particular pin, in this case where the readings are coming from: the potentiometer.

Variable: as we've seen before, this facilitates the use of the variable resistance of the potentiometer



```
forever
    set Potentiometer to 0
                    analog read pin P1
```



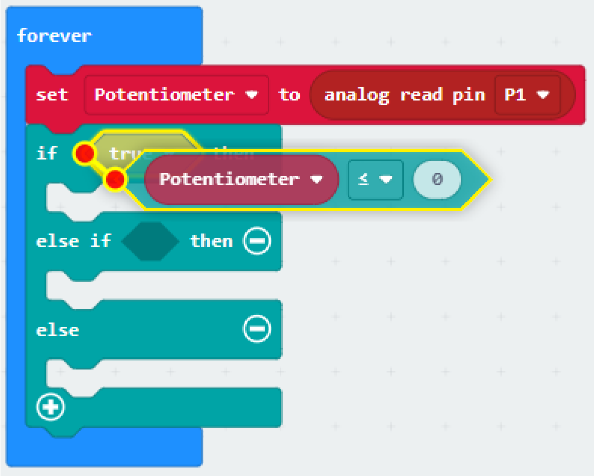
```
forever
    set Potentiometer to analog read pin P1
    if true then
    else if then
    else
```

1:

- As we have seen before, using ‘variable and the ‘pins’ tab we can set the reading from the potentiometer by setting the reading to come from P0,1,2 depending on which pin is connected again to the middle leg of the potentiometer.

- Using the ‘logic tab we can create the following scenarios. If __ then __, else if _ then __, else __’.

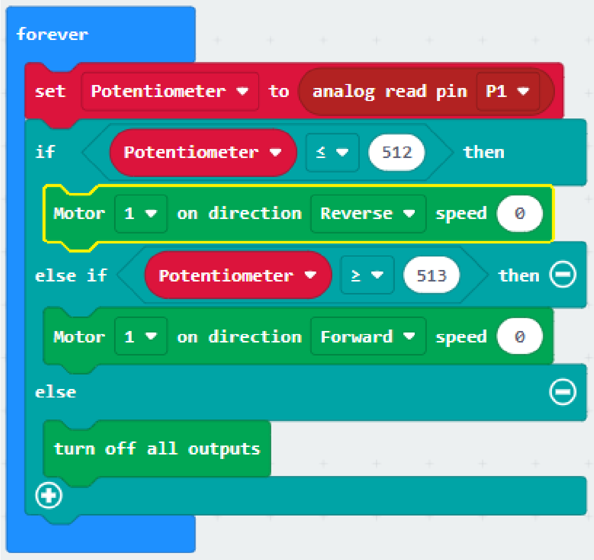
- When you open the tab and use the logic gate If __ then __’ you can click on the plus shown to add an extra scenario.



```
forever
  set Potentiometer to analog read pin P1
  if true then
    Potentiometer ≤ 0
  else if then
  else
```

2:

- Using the 'Comparison' section of the 'logic' tab we create a comparison that when the potentiometer reads less than or equal to a value, then it will turn on the motor
- We use the second block in 'else if __ then' so if the potentiometer output is greater than a reading it will do something else with the motor
- Finally, we use the 'else' section to turn off all outputs if this doesn't exist

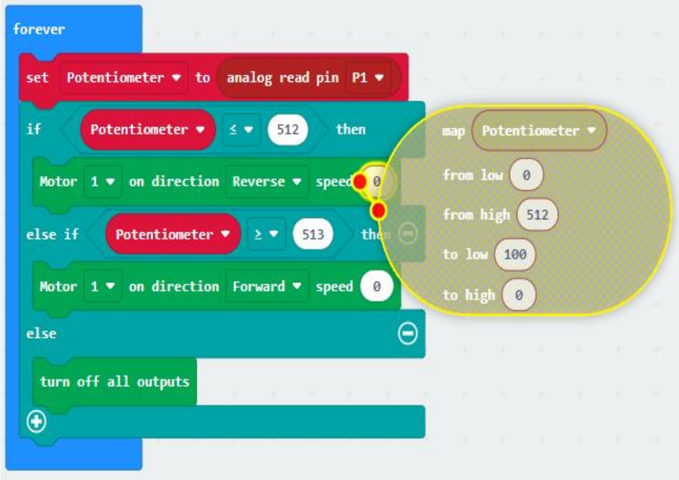


```
forever
  set Potentiometer to analog read pin P1
  if Potentiometer ≤ 512 then
    Motor 1 on direction Reverse speed 0
  else if Potentiometer ≥ 513 then
    Motor 1 on direction Forward speed 0
  else
    turn off all outputs
```

3:

- As you can see here the readings have been set at half of 1023, the maximum reading from our potentiometer ** Use the calibration video used in the previous activity in order to find the middle of the resistance of the potentiometer

- Using the 'robotics' tab we can add the motor controls, changing the directions and eventually turning off all the outputs

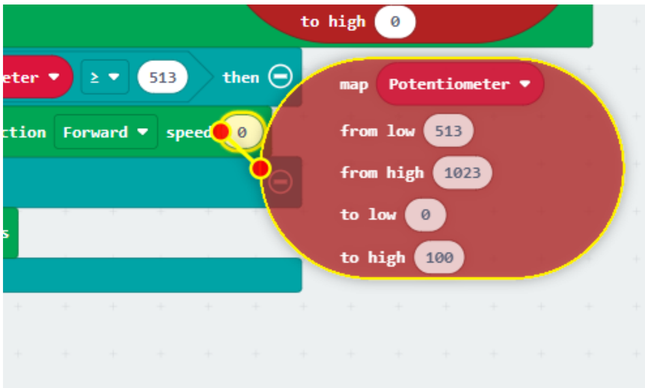


```
forever
  set Potentiometer to analog read pin P1
  if Potentiometer ≤ 512 then
    Motor 1 on direction Reverse speed 0
  else if Potentiometer ≥ 513 then
    Motor 1 on direction Forward speed 0
  else
    turn off all outputs

map Potentiometer
  from low 0
  from high 512
  to low 100
  to high 0
```

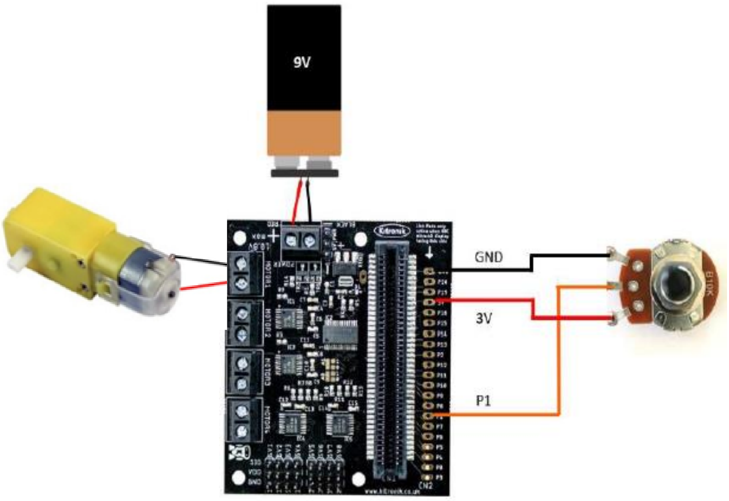
4:

- Using 'pins' and the 'map' block we have used before we set up the situation where the potentiometer gives values of resistance from 0 - half way (512) and this maps to the speed of the motor

- When setting the speed of the motor we need to be conscious of the fact that we want the lowest reading from the potentiometer to be the fastest speed so the low in '100' and the middle resistance of the potentiometer is '0'



```
to high 0
eter ▾  ≥ ▾  513  then ⊖
ction  Forward ▾  speed  0
map  Potentiometer ▾
from low  513
from high  1023
to low  0
to high  100
```



5:

- Using the previous learning, we create a new map where we move from 513 to the top range of the 1023
- From this now we build up the speed from the middle of the resistance up to the top end of the resistance
- We map this speed from 0 up to the maximum speed of the motor
- Looking at the code, we have now said that if the potentiometer is in the middle position, the motor should be turned off, as you turn to the lowest resistance the motor turns in reverse and begins to speed up until you are at 0 resistance, as you turn to the opposite direction from the middle, the direction changes and it speeds up until you hit the maximum resistance

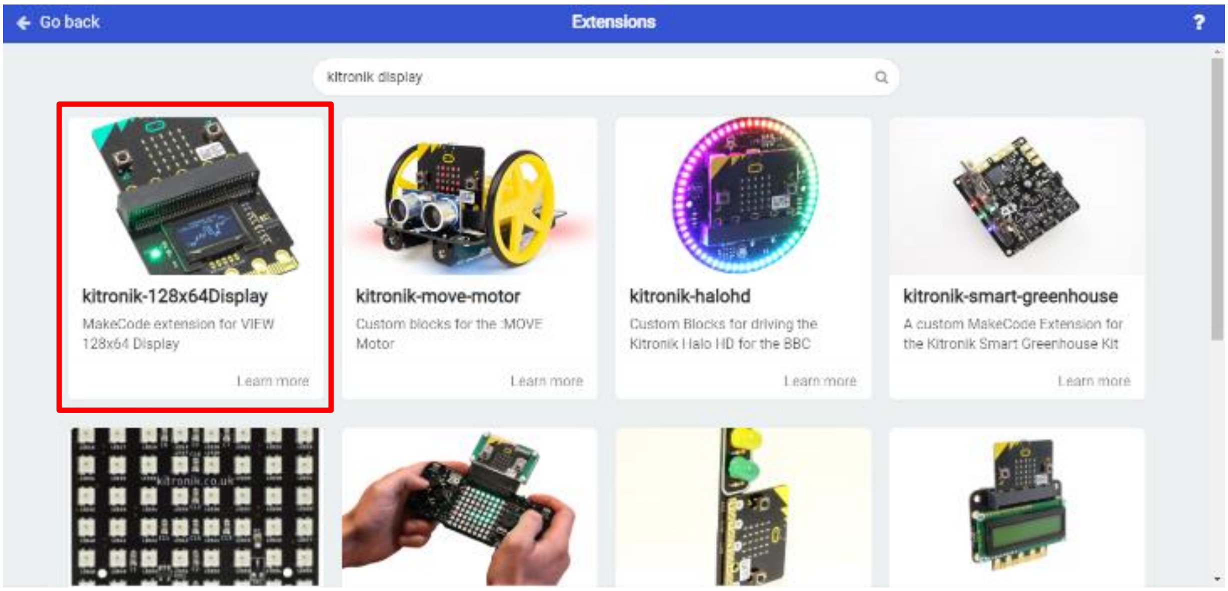
6:

- Configure the components the same way as activity 3 as shown in the image to the left.
- Download and test the code

Module 9: External Screen

In previous modules we have examined how to display a range of text and numerical data on the micro:bit LED display. We can enhance our ability to display data from a range of both onboard and external sensors, as well as displaying lines of text using an external screen. JCt4 have provided a Kitronik 128x64 LED display in the Mechatronics resource which we will use in the following activities.

To engage with this activity, you will need to add in another extension. This one is found in the extension library under the search of Kitronik Display. You can see in the screen shot above which extension to choose.



kitronik display

kitronik-128x64Display
MakeCode extension for VIEW 128x64 Display
Learn more

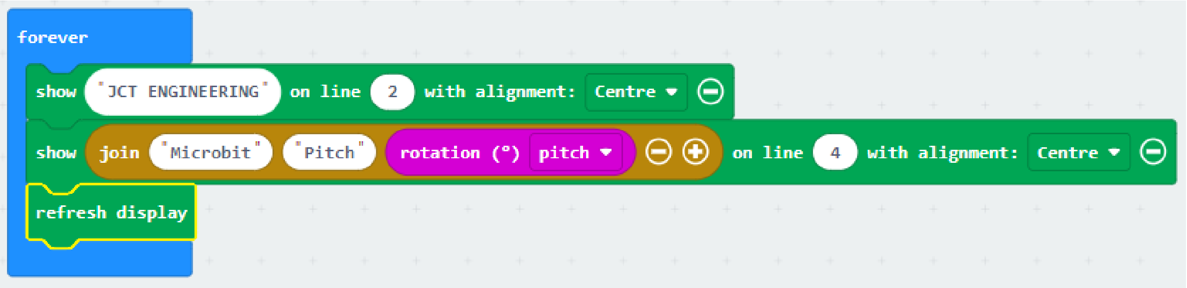
kitronik-move-motor
Custom blocks for the :MOVE Motor
Learn more

kitronik-halohd
Custom Blocks for driving the Kitronik Halo HD for the BBC
Learn more

kitronik-smart-greenhouse
A custom MakeCode Extension for the Kitronik Smart Greenhouse Kit
Learn more

Activity A: Display Text and Internal Sensor Values

Video



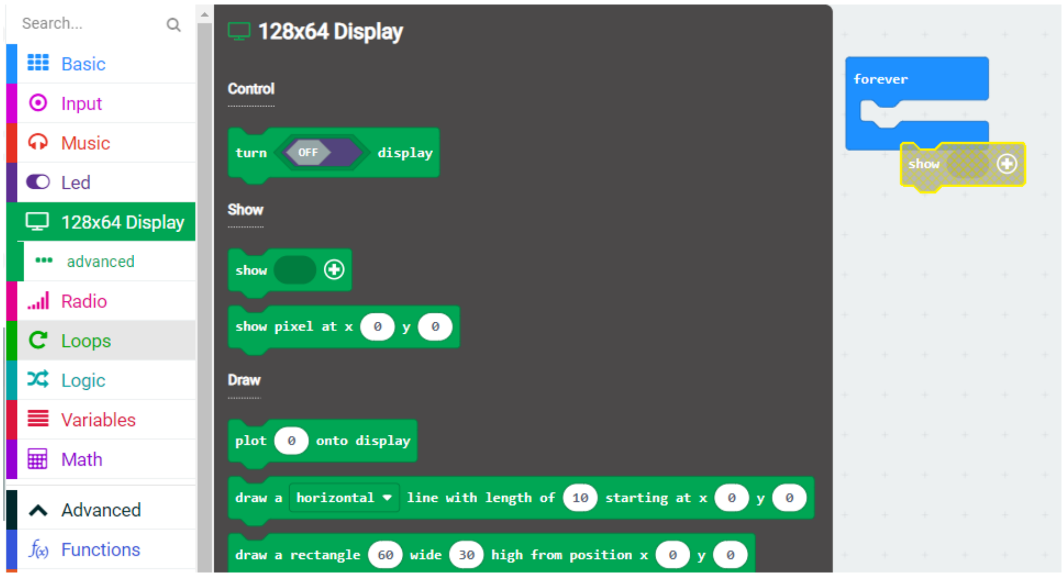
```
forever
    show "JCT ENGINEERING" on line 2 with alignment: Centre
    show join "Microbit" "Pitch" rotation (°) pitch on line 4 with alignment: Centre
    refresh display
```

The code above is the solution to the activity.
It involves some new blocks that have not been engaged with in any other module.

128x64 Display: allows for text, measurements and other images to be shared on the screen

Text: allows words, values, and other variables to be shown on the screen

Input: allows the pitch of the micro:bit to be shown on the screen

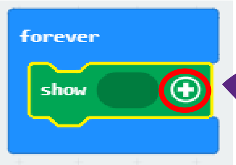


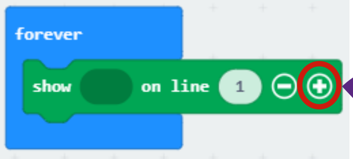
1:

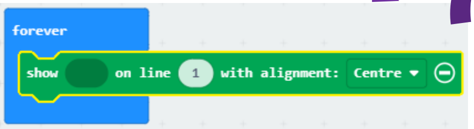
- Using the 128x64 display select the ‘show’ block from the tab
- As shown, drop the block in the ‘forever’ command therefore the text and numbers will end up on the screen forever

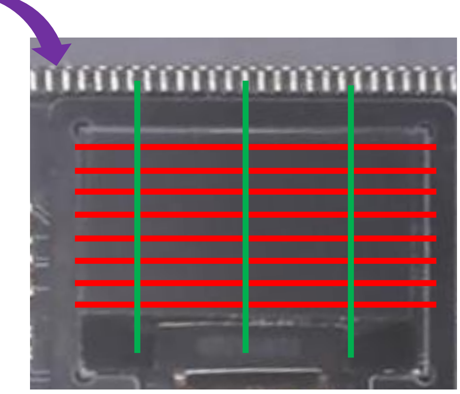
2:

- Using the plus arrow expand out the code block so it reads ‘on line __’
- Using the plus arrow again, add in the alignment of the text or variables





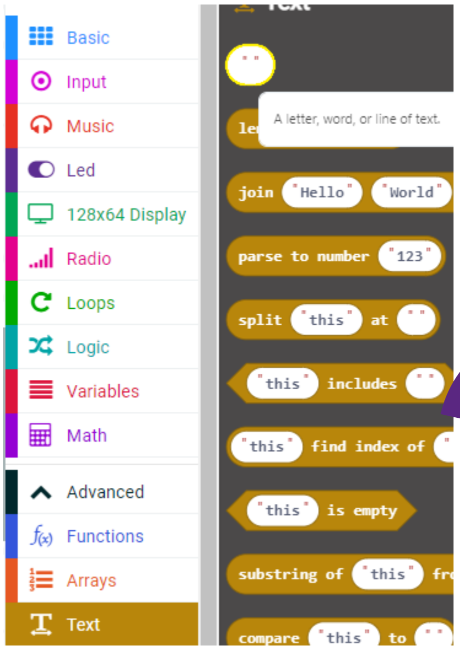




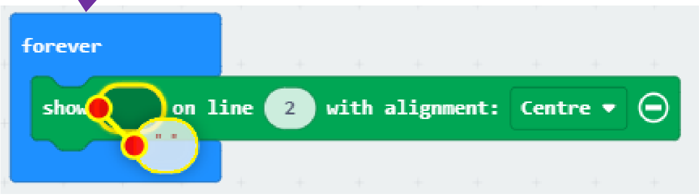
- There are 8 lines that you can show code on the screen (indicated in red)
- There are 3 alignments on the screen (indicated in Green)

3:

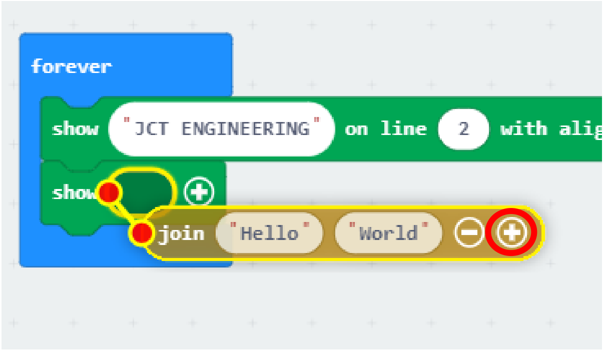
- Using the ‘text’ tab select the block shown
- As seen in the image, this block allows for a letter, word, or line of text to be shown
- Add this block to the screen block as shown in the image below
- Add your desired text to this block, as shown below left, ‘JCT ENGINEERING’ was added



```
" "
A letter, word, or line of text.
join "Hello" "World"
parse to number "123"
split "this" at " "
"this" includes " "
"this" find index of "
"this" is empty
substring of "this" fr
compare "this" to " "
```



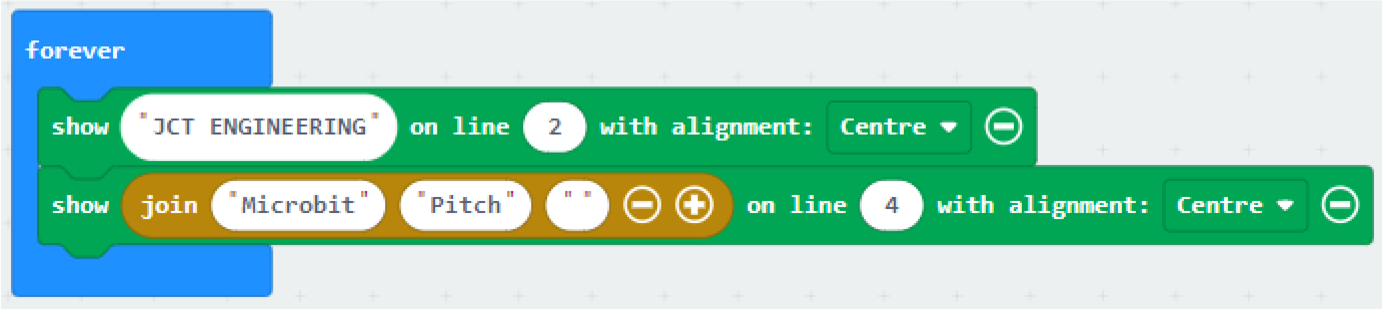
```
forever
  show " " on line 2 with alignment: Centre ▾ ⊖
```



```
forever
  show "JCT ENGINEERING" on line 2 with alig
  show ⊕
    join "Hello" "World" ⊖ ⊕
```

4:

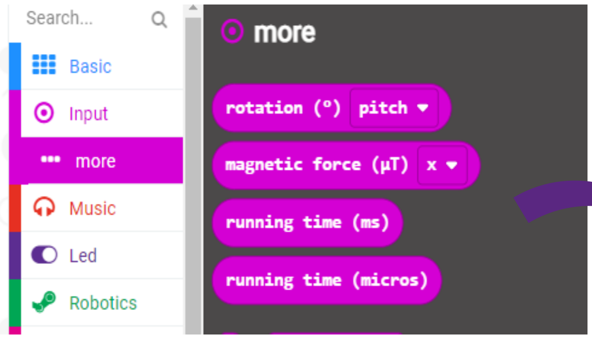
- Using the ‘128x64 display’ tab add the ‘show __’ block again to the forever loop
- Using the text tab select the ‘join __ __’ block add it to the ‘show block’ as shown
- Using the plus button on the block add space for more text



```
forever
  show "JCT ENGINEERING" on line 2 with alignment: Centre ▾ ⊖
  show join "Microbit" "Pitch" " " ⊖ ⊕ on line 4 with alignment: Centre ▾ ⊖
```

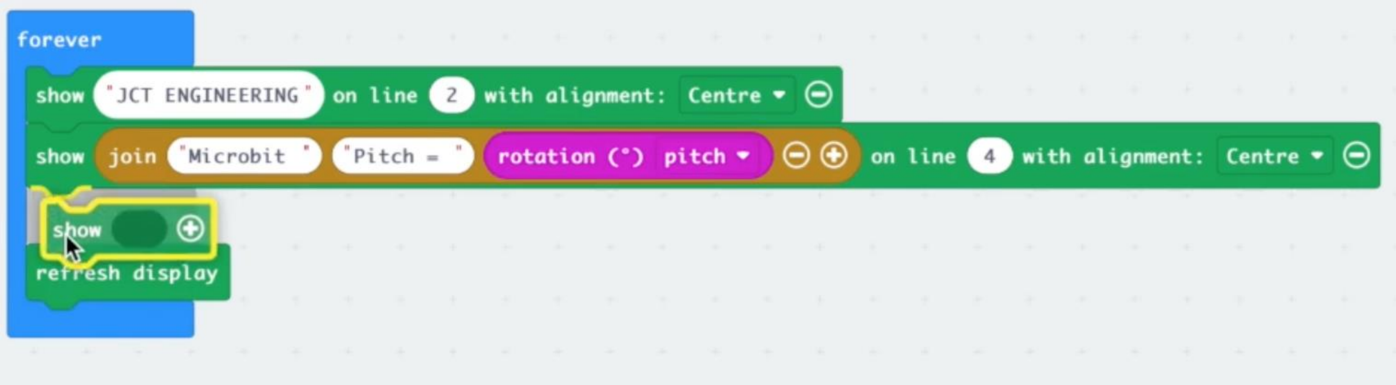
5:

- As shown previously, before using the plus button on the ‘show’ block, you can select on which line you would like to display the text and the alignment
- On screen, it will now show the words Micro:bit and pitch, the next step is to add in the numerical value of the pitch.

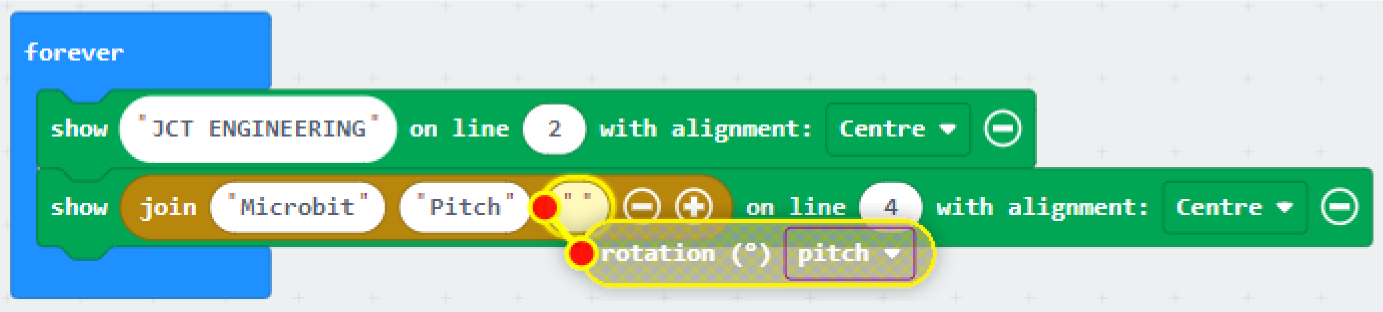


6:

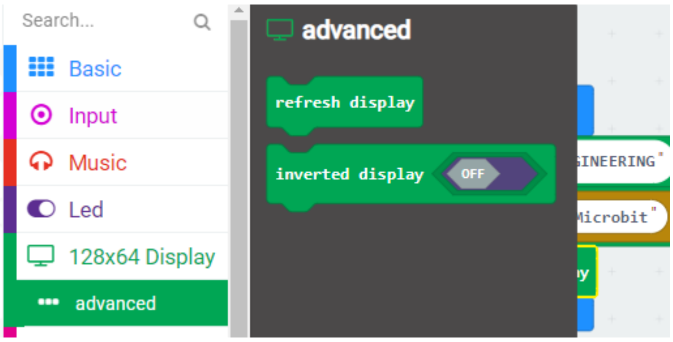
- Using the 'more' option of the 'input' tab use the block 'rotation (°) pitch'
- Add it to the 'join' block as shown below
- This will now allow the reading from the pitch of the micro:bit to come up on the screen.



```
forever
  show "JCT ENGINEERING" on line 2 with alignment: Centre
  show join "Microbit " "Pitch = " rotation (°) pitch on line 4 with alignment: Centre
  show
  refresh display
```



```
forever
  show "JCT ENGINEERING" on line 2 with alignment: Centre
  show join "Microbit" "Pitch" " " on line 4 with alignment: Centre
                        rotation (°) pitch
```



7:

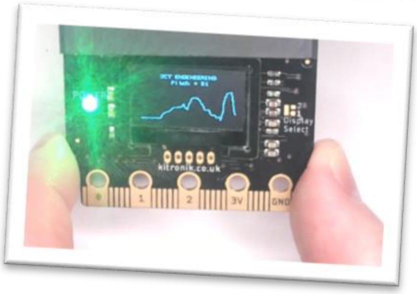
- From the advanced section of the '128x64 Display' tab use the 'refresh display' block in order to constantly refresh the screen and the readings from the pitch

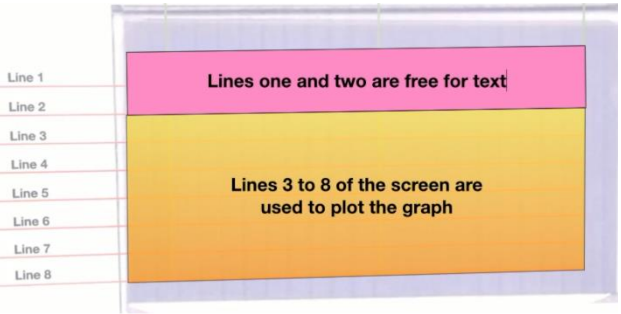
**If this step was not added in the screen would only show the first reading of the pitch and not show any changes in rotation

Video

Activity B: Plot a Graph of Sensor

Sometimes it is desirable to visually represent a set of values. In this activity, we will explore how we can use the graph feature of our Kitronik 128x64 screen to record a range of values in a live graph. We use values obtained from the accelerometer sensor on the Micro:bit to plot the graph.

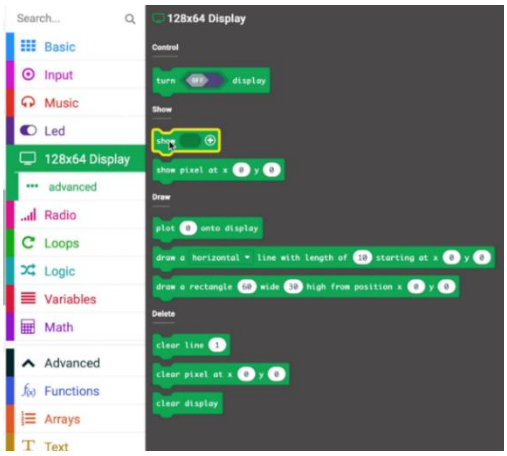




Lines one and two are free for text

Lines 3 to 8 of the screen are used to plot the graph

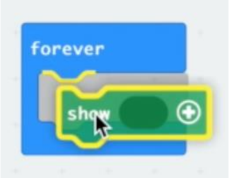
Before we begin this activity, we must first understand how the screen allocates space for the graph to be plotted. From the eight lines available, it reserves lines 1 and 2 for text/data. Lines 3 – 8 are then used as the area for the graph, as shown on the image opposite. Any text on lines 3 – 8 will be plotted over by the graph.



```
turn off display
show
show pixel at x 0 y 0
plot 0 onto display
draw a horizontal line with length of 10 starting at x 0 y 0
draw a rectangle 60 wide 30 high from position x 0 y 0
clear line 1
clear pixel at x 0 y 0
clear display
```

1:

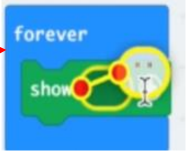
- Open a new Makecode document and add in the 128x64 extension.
- From the '128x64 Display' tab, choose the "show ______" block and place it in the "forever" block as shown.



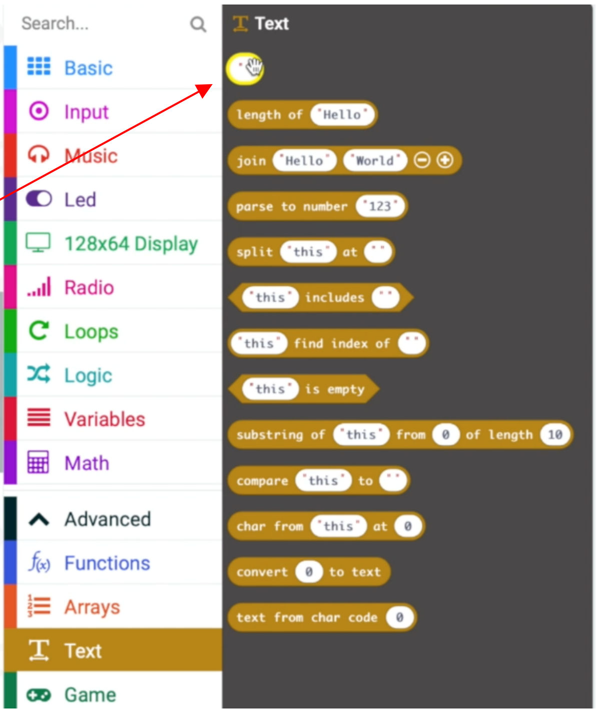
```
forever
  show
```

2:

- We cannot type directly into the 'show___' code. We need to use the 'Text' code blocks. These are found in the 'Text' drawer. (You may need to click on the advanced Tab at the bottom of the code drawers to see this)
- From the 'Text' drawer, choose the "_" block from the top of the drawer and place it in the "show _____" block as shown here.

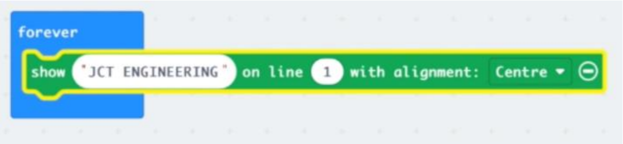


```
forever
  show
```

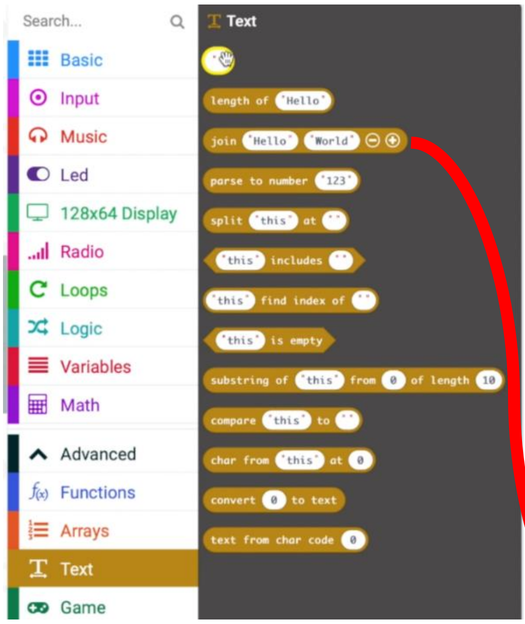


```
" "
length of "Hello"
join "Hello" "World"
parse to number "123"
split "this" at " "
"this" includes " "
"this" find index of " "
"this" is empty
substring of "this" from 0 of length 10
compare "this" to " "
char from "this" at 0
convert 0 to text
text from char code 0
```

- Type "JCT ENGINEERING" in the text box, and place it on line 1, centre aligned.



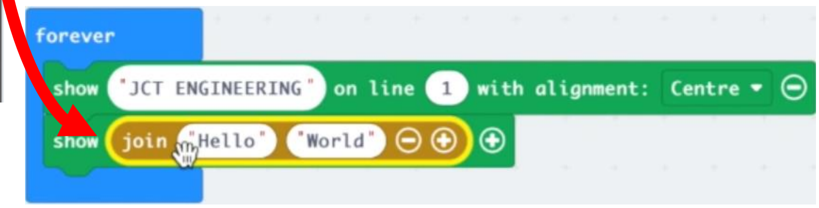
```
forever
  show "JCT ENGINEERING" on line 1 with alignment: Centre
```



3:

- Add another show____ block from the 128x64 Display code drawer.
- From the Text drawer, choose the Join "Hello" "World" block and place it into the show______ block as shown below.

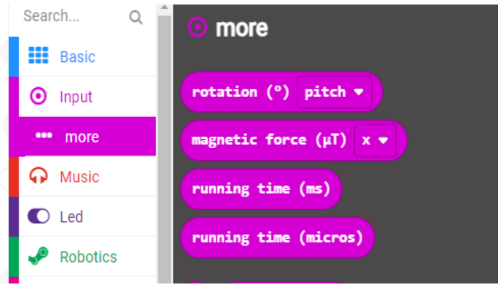
- The "Hello" "World" block allows us to insert text and variables in the one string. We will use this to display the text: "Pitch = XX" on the display.

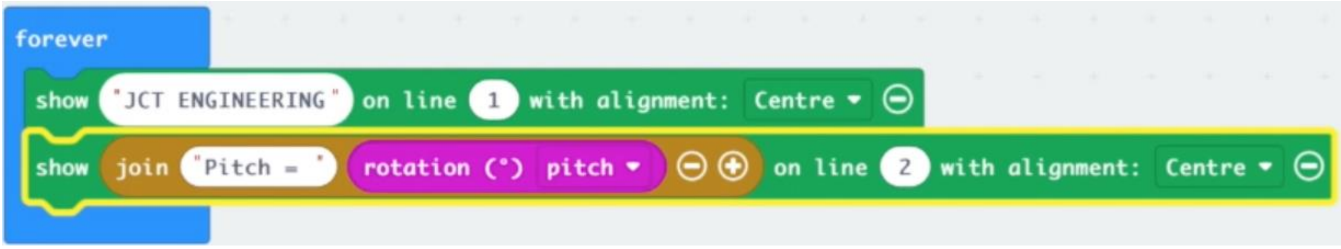


```
forever
  show "JCT ENGINEERING" on line 1 with alignment: Centre ▾ ⊖
  show join "Hello" "World" ⊖ ⊕ ⊕
```

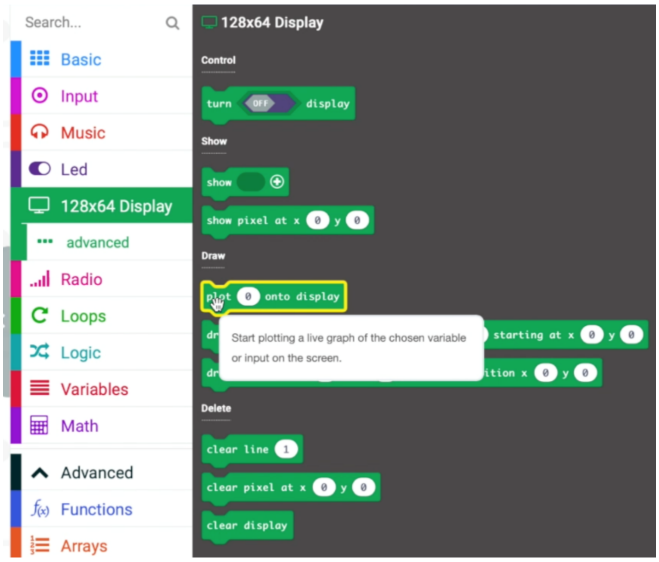
4:

- In the "Hello" section of the join "Hello" "World" block, type in the text "**Pitch =** " as in the example below.
- We want to display the pitch, so, from the Input drawer, click on the ... more tab and select the rotation (°) pitch block and drag it into the second input of the join "Hello" "World" block, also shown below.
- Place this on line 2 with centre alignment.





```
forever
  show "JCT ENGINEERING" on line 1 with alignment: Centre ▾ ⊖
  show join "Pitch = " rotation (°) pitch ▾ ⊖ ⊕ on line 2 with alignment: Centre ▾ ⊖
```



Start plotting a live graph of the chosen variable or input on the screen.

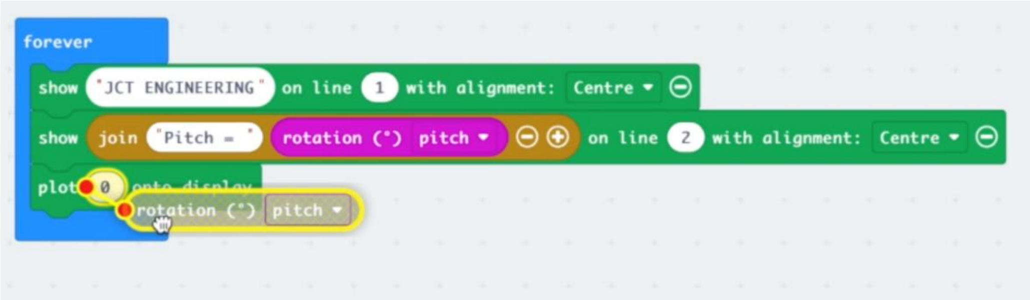
5:

We will now begin to plot the graph.

- From the 128x64 Display drawer, choose the plot__ onto display code block.

This is in the *Draw* section of the drawer and is shown highlighted in the image on the left.

- Drag this out into the forever block and place it under the code previously inserted.



```
forever
  show "JCT ENGINEERING" on line 1 with alignment: Centre
  show join "Pitch = " rotation (°) pitch on line 2 with alignment: Centre
  plot 0 onto display
     rotation (°) pitch
```

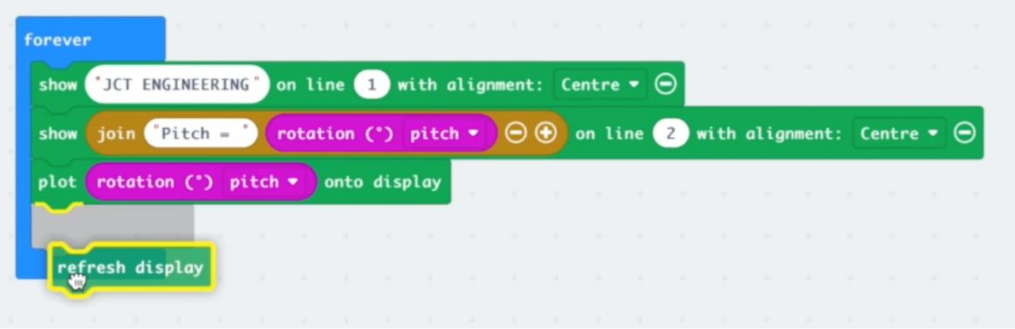
6:

We now want to tell the screen to plot the graph of the pitch. A convenient method to do this is to right click on the rotation (°) pitch code we inserted in step 4 of this activity and select duplicate from the contextual menu that pops up when we right-click.

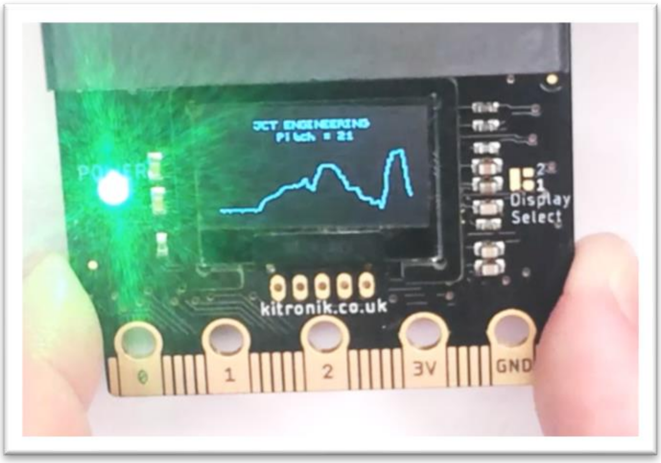
We place the duplicated rotation (°) pitch code into the plot__ onto display code block. The Micro:bit will now plot the values for the pitch onto the display when the code is flashed to it and the screen connected. (Alternatively, you can select the rotation (°) pitch code form the Input drawer, as we did in step 4)

7:

To help limit latency or lags in plotting the graph, we add in the refresh display code from the 128x64 Display drawer.



```
forever
  show "JCT ENGINEERING" on line 1 with alignment: Centre
  show join "Pitch = " rotation (°) pitch on line 2 with alignment: Centre
  plot rotation (°) pitch onto display
  refresh display
```



8:

Flash the completed code above to the micro:bit. Ensure the display is connected or the micro:bit will give an ERROR – no display warning on its LED matrix. When the micro:bit reboots, it will display a titled graph as shown here and the values will update to visually show the variance in pitch as you move the micro:bit. The axes will automatically update to show the extreme values as you tilt the device.

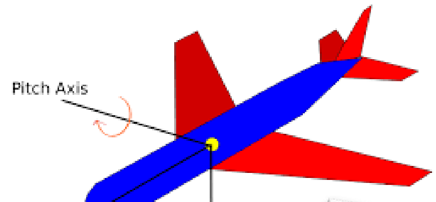
What workshop activities or processes might this activity support in Junior Cycle Engineering?

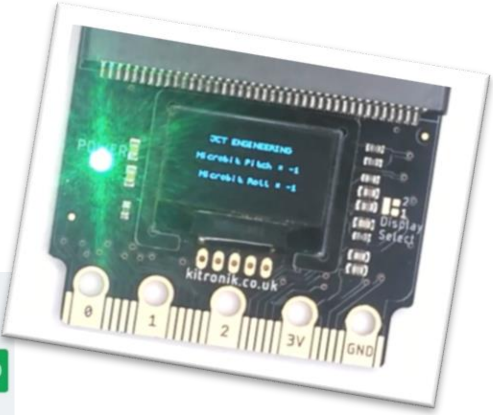
Activity C: Display Values from Multiple Sensors

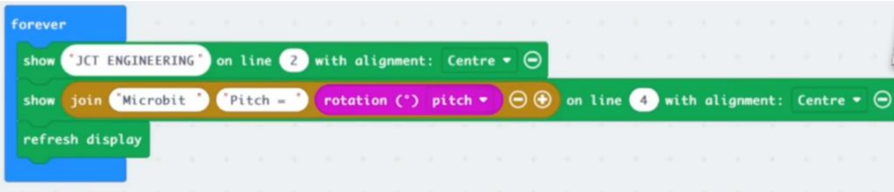
Video

In activity A, we investigated the layout of our screen and how we can display text and numerical data obtained from an onboard sensor. In this activity, we shall now extend on activity A and display two sets of data from the micro:bit's onboard accelerometer, the Pitch and Roll.

Refer to the image of the plane as a quick reminder of the terms Pitch and Roll.



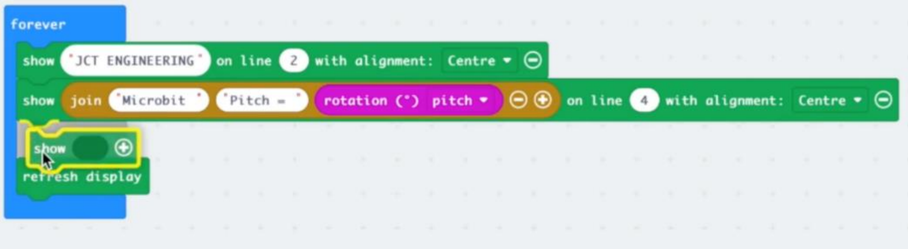




```
forever
  show "JCT ENGINEERING" on line 2 with alignment: Centre
  show join "Microbit " "Pitch = " rotation (°) pitch on line 4 with alignment: Centre
  refresh display
```

1:

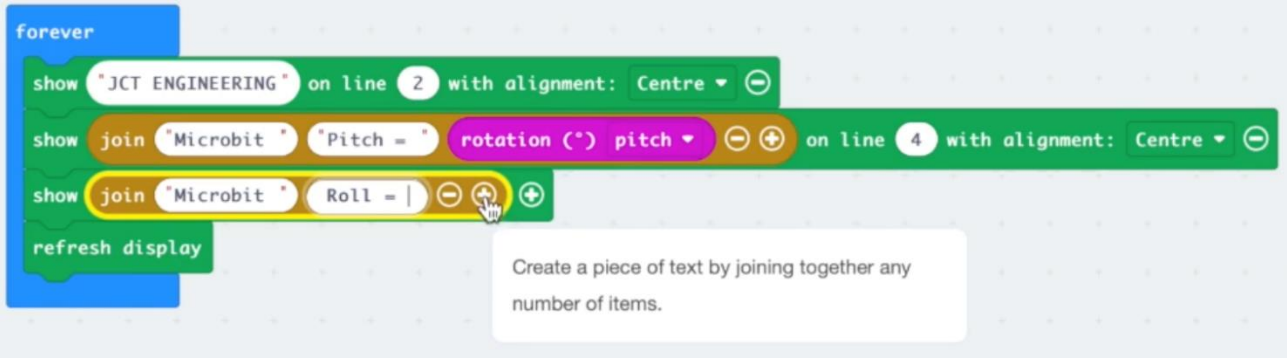
The code used in Activity A is shown above. Set up this code following the instructions in Activity A, or alternatively, duplicate your Activity A file.



```
forever
  show "JCT ENGINEERING" on line 2 with alignment: Centre
  show join "Microbit " "Pitch = " rotation (°) pitch on line 4 with alignment: Centre
  show
  refresh display
```

2:

Add a show_____ block from the 128x64 Display code drawer and place it in the forever block as shown opposite.



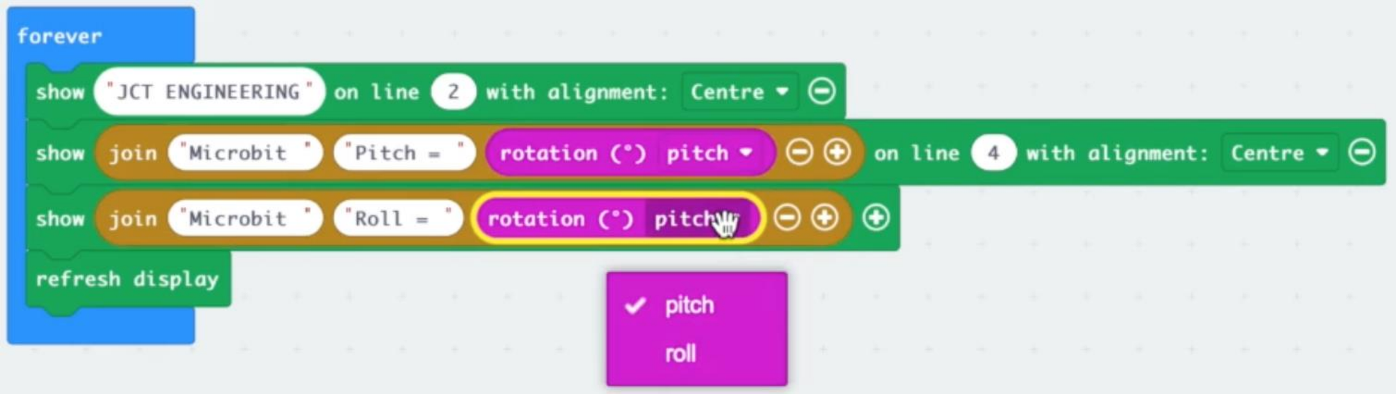
```
forever
  show "JCT ENGINEERING" on line 2 with alignment: Centre
  show join "Microbit " "Pitch = " rotation (°) pitch on line 4 with alignment: Centre
  show join "Microbit " "Roll = |"
  refresh display
```

Create a piece of text by joining together any number of items.

3:

- From the Text drawer, choose the Join "Hello" "World" block and place it into the show______ block as shown above.

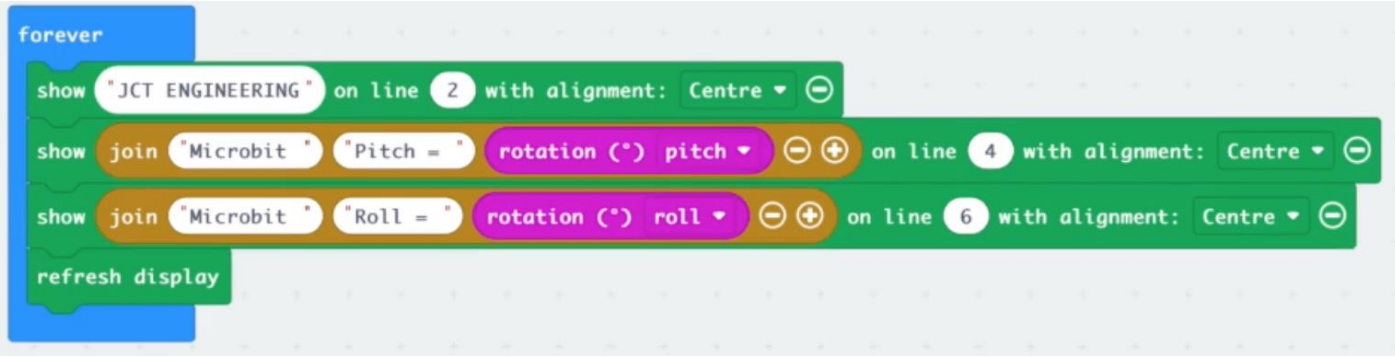
- In the join "Hello" "World" block, insert the text "**Microbit** " (*be sure to include a space after the word, or the screen will bunch the words together*) into the first input space and insert the text "**Roll =** " into the second input box (*again, be sure to include a space after the text*)
- Now click on the **+** symbol, shown above, to add another input box to the block. This will allow us to add the variable we need in the next step.



```
forever
  show "JCT ENGINEERING" on line 2 with alignment: Centre
  show join "Microbit " "Pitch = " rotation (°) pitch on line 4 with alignment: Centre
  show join "Microbit " "Roll = " rotation (°) pitch
  refresh display
```

4:

- We now want to tell the Micro:bit to measure the roll values from the accelerometer. This value is contained in the rotation (°) pitch code.
- As in the previous activity, a convenient method to do this is to right click on the existing rotation (°) pitch code we inserted and select duplicate from the contextual menu that pops up when we right click.
- We place the duplicated rotation (°) pitch code into the third input box of the Join "Hello" "World". By clicking on the drop down to the right of the pitch, choose roll to change the reading that will be displayed on this line.
- Modify this line of the show____ code block so it appears on line 6, with centre alignment.



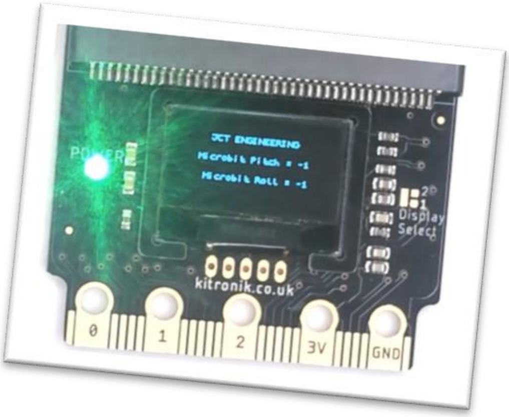
```
forever
  show "JCT ENGINEERING" on line 2 with alignment: Centre
  show join "Microbit " "Pitch = " rotation (°) pitch on line 4 with alignment: Centre
  show join "Microbit " "Roll = " rotation (°) roll on line 6 with alignment: Centre
  refresh display
```

5:

Flash the completed code above to the Micro:bit. Ensure the display is connected or the Micro:bit will give an ERROR – no display warning on its LED matrix.

When the Micro:bit reboots, it will display both the Pitch and roll values that its accelerometer is sensing.

Try carefully tilting and twisting the Micro:bit in various directions to see this in action.



Can you identify a machine or other piece of engineering that monitors pitch and roll values?
How does the machine use this information?

Video

Activity D: Display Readings from an External Sensor Example – A Potentiometer

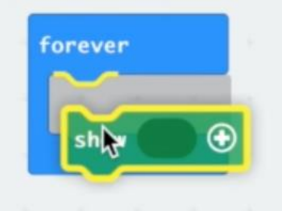
We have used the potentiometer in many activities across the previous modules, 1 -8. In this activity, we will use the screen to display the range of values obtained from the potentiometer in real time.

To set up this activity, connect your potentiometer to the header pins on the Robotics board as follows:

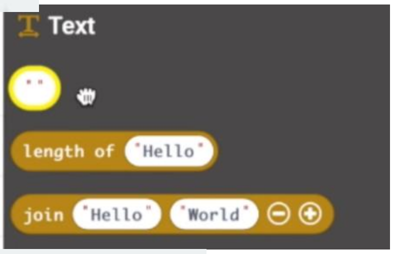
- To set up this activity, connect your potentiometer to the header pins on the Robotics board, ensuring that the wiper (centre pin) is connected to pin 0.

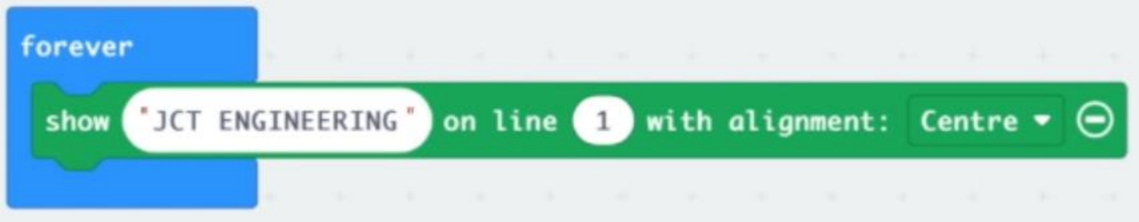
1:

- From the 128x64 Display drawer, select the show____ block.
- As shown opposite, drop the show____ block into the 'forever' command.
- Next, we add a basic text box. To do this, open the Text drawer and select the "_" code from the top of the drawer. Insert this into the input of the show____ block.



```
forever
  show ( )
```





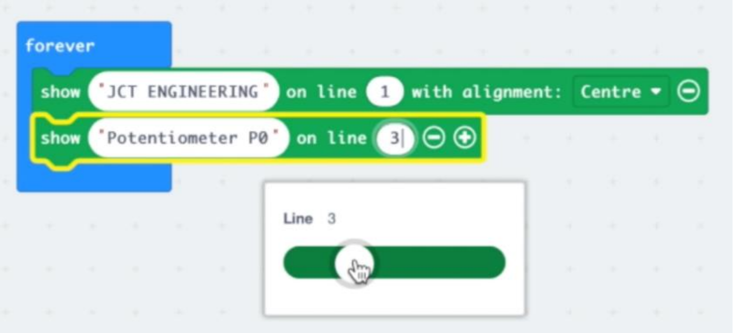
```
forever
  show "JCT ENGINEERING" on line 1 with alignment: Centre ▾
```

2:

In the "_" text input, type in **JCT ENGINEERING** as shown above. Modify the code to show the text on Line 1 with centre alignment.

3:

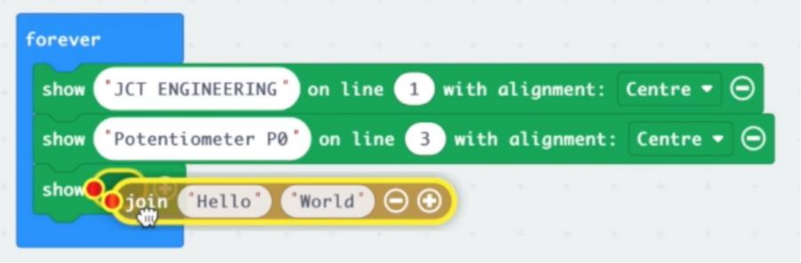
- From the 128x64 Display drawer select the show____ block.
- Place the show____ block into the 'forever' command beneath the previous one.
- As in step one, add a basic text box. To do this, open the Text drawer and select the "_" code from the top of the drawer. Insert this into the input of the show____ block.
- Type the text **Potentiometer P0** into the text input.
- Place the text on line 3, with centre alignment.



```
forever
  show "JCT ENGINEERING" on line 1 with alignment: Centre ▾
  show "Potentiometer P0" on line 3
```

4:

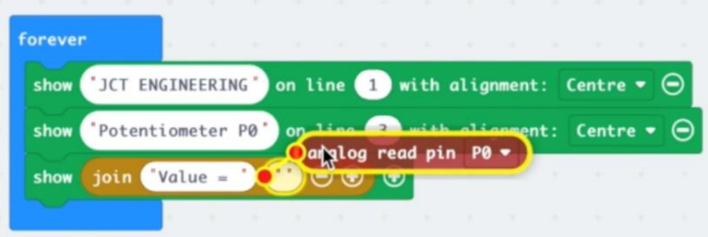
- Add another show_____ block into the 'forever' command beneath the previous one.
- We now want to insert text and a variable so we will use the join "Hello" "World" block.
- To do this, open the Text drawer and select the join "Hello" "World" code from the drawer. Insert this into the input of the show_____ block.
- Type the text "**Value =**" into the first input box.
- Place the text on line 3, with centre alignment.



```
forever
  show "JCT ENGINEERING" on line 1 with alignment: Centre
  show "Potentiometer P0" on line 3 with alignment: Centre
  show join "Hello" "World"
```

5:

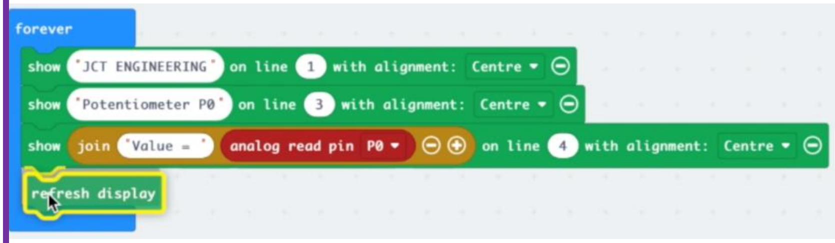
- Type the text **"Value = "** into the first input box of the join "Hello" "World" block. (Remember to add a space after the = sign or the screen will join the values to the text as it displays)
- Now we will add the values from the potentiometer. From the Pins drawer, add the analog read pin P0 block to the second input of the join "Hello" "World" block.



```
forever
  show "JCT ENGINEERING" on line 1 with alignment: Centre
  show "Potentiometer P0" on line 3 with alignment: Centre
  show join "Value = " analog read pin P0
```

6:

- Finally, place the code from step five on line 4 with centre alignment.
- Add a refresh display command to your code to help reduce latency in the memory of your screen.
- The completed code is shown to the left.
- Flashing this to your Micro:bit will make it display the values it is reading from the potentiometer connected on Pin 0.



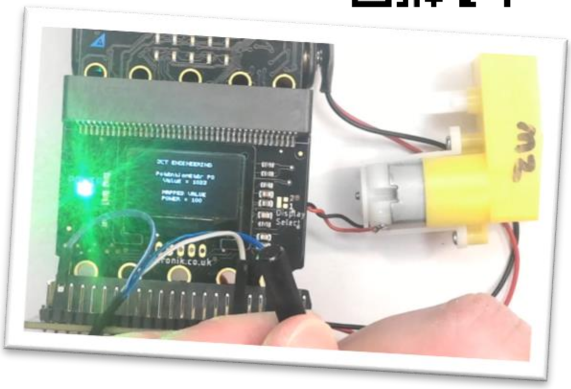
```
forever
  show "JCT ENGINEERING" on line 1 with alignment: Centre
  show "Potentiometer P0" on line 3 with alignment: Centre
  show join "Value = " analog read pin P0 on line 4 with alignment: Centre
  refresh display
```

Has this activity helped you understand how the micro:bit "reads" the signals from the potentiometer? How could this be used to help student understanding of the difference in analog and digital input?

Video

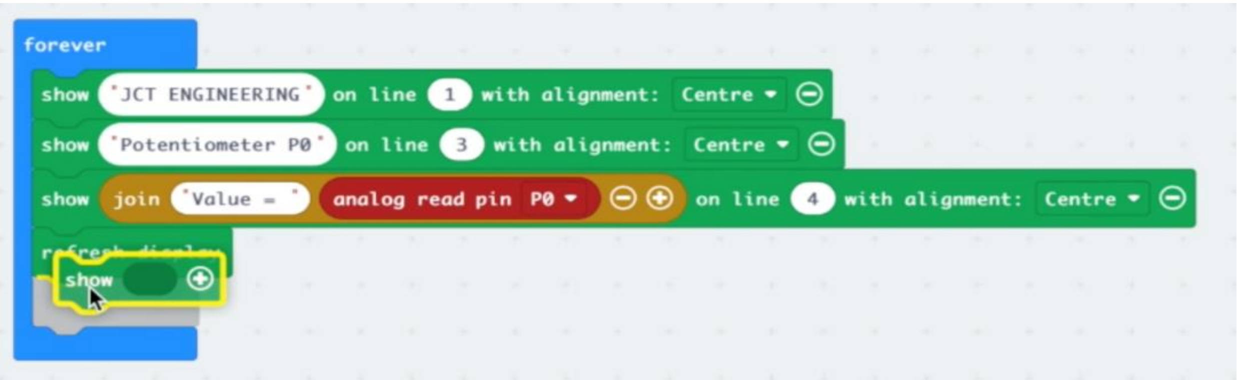
Activity E: Investigating Mapped Values using the 128x64 Display.

The “Map” function is something we regularly use with our Micro:bit, particularly when controlling servo and DC motors. In many projects, smooth operation of our drive systems is dependent on mapped values; where our Micro:bit is relating the speed or direction of the motor to an analog reading. We have explored this extensively in Module 8.
So what exactly is the Micro:bit doing when it maps a value? Using the code and potentiometer set up in Activity D. We shall look a little closer to find out.



1:

- From the 128x64 Display drawer select the show____ block.
- Place the show____ block into the ‘forever’ command beneath the previous code from Activity D.
- Next, add a basic text box. To do this, open the Text drawer and select the “_” code from the top of the drawer. Insert this into the input of the show____ block.
- Type the text “**VALUE = ”** into the text input, remembering to add a space at the end. Place the text on line 4, with centre alignment.

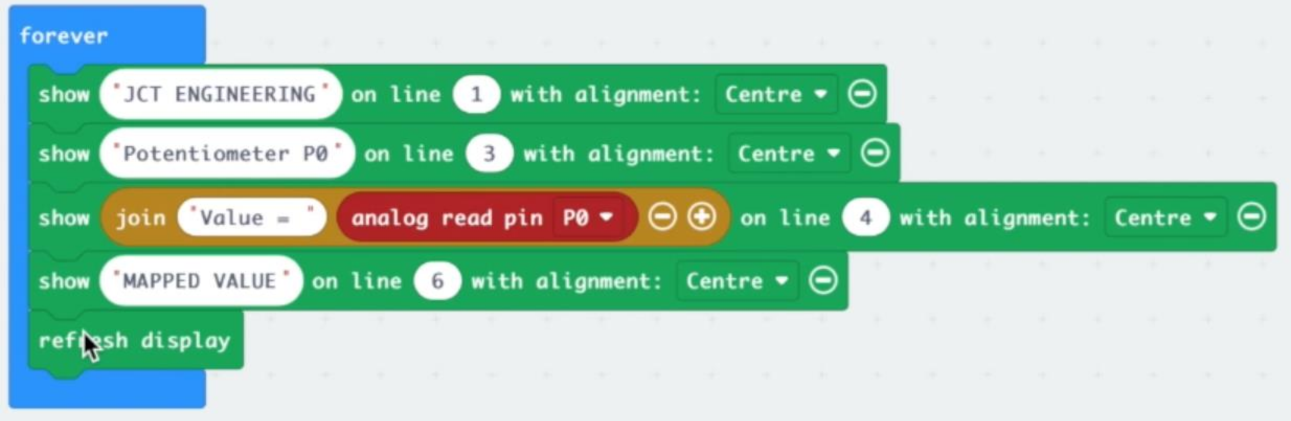


```
forever
    show "JCT ENGINEERING" on line 1 with alignment: Centre
    show "Potentiometer P0" on line 3 with alignment: Centre
    show join "Value = " analog read pin P0 on line 4 with alignment: Centre
    refresh display
    show
```

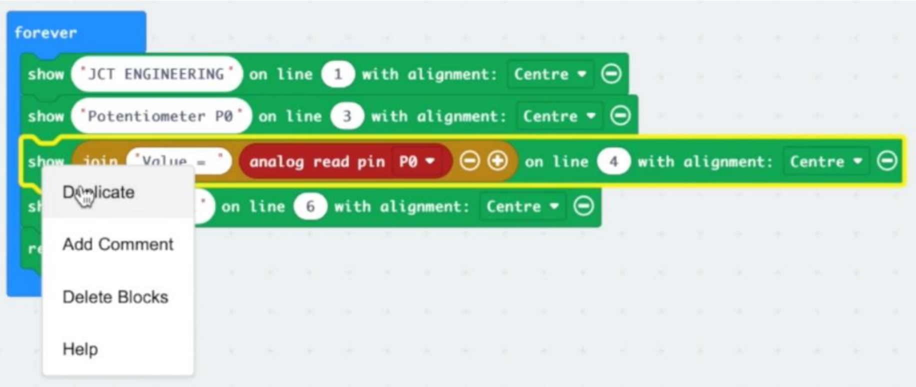
2:

- Type the text “**MAPPED VALUE ”** into the text input added above, remembering to add a space at the end.

- Modify the code to place the text on line 6, again with centre alignment. This is the title text for our mapped values which we shall display on the next line.



```
forever
    show "JCT ENGINEERING" on line 1 with alignment: Centre
    show "Potentiometer P0" on line 3 with alignment: Centre
    show join "Value = " analog read pin P0 on line 4 with alignment: Centre
    show "MAPPED VALUE " on line 6 with alignment: Centre
    refresh display
```



```
forever
  show "JCT ENGINEERING" on line 1 with alignment: Centre
  show "Potentiometer P0" on line 3 with alignment: Centre
  show join "Value = " analog read pin P0 on line 4 with alignment: Centre
  sh... on line 6 with alignment: Centre
```

3:

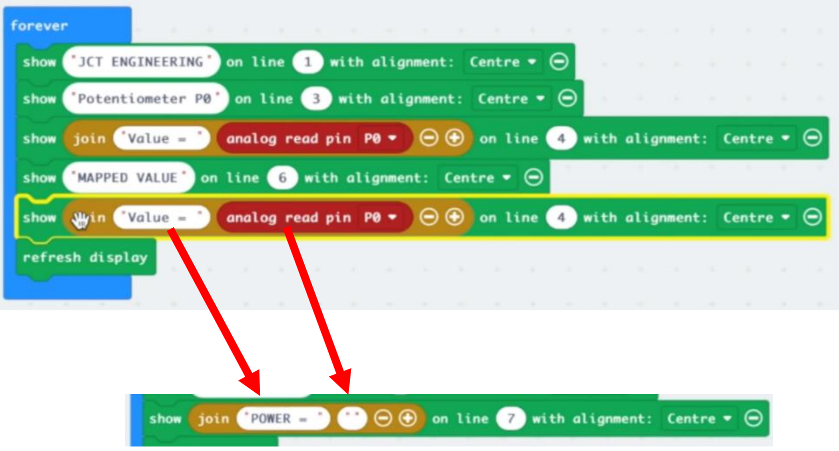
For convenience, we can duplicate a line of code here.

- Right click on the show____ block containing the join "Hello" "World" block.
- Click on *Duplicate* from the contextual menu that appeared when you right clicked.

- Place this duplicate block of code into the forever block as indicated to the right.
- Modify the text in the first input of this line to read **"POWER = "**

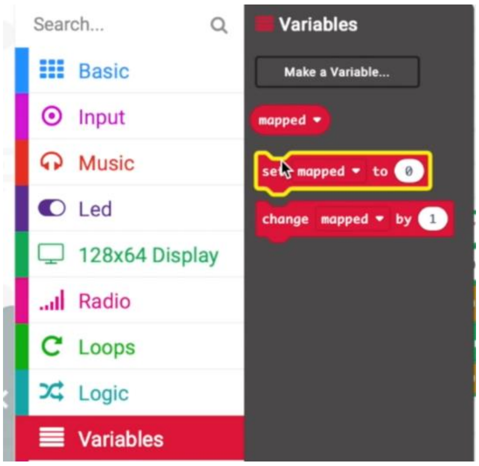
Remember to add a space at the end!

- We delete the analog read pin P0 from the block as well.
- As we will use this line to display the mapped value that the Micro:bit has calculated.
- Modify this code to place the text on line 7 with centre alignment.



```
forever
  show "JCT ENGINEERING" on line 1 with alignment: Centre
  show "Potentiometer P0" on line 3 with alignment: Centre
  show join "Value = " analog read pin P0 on line 4 with alignment: Centre
  show "MAPPED VALUE" on line 6 with alignment: Centre
  show join "Value = " analog read pin P0 on line 4 with alignment: Centre
  refresh display
```

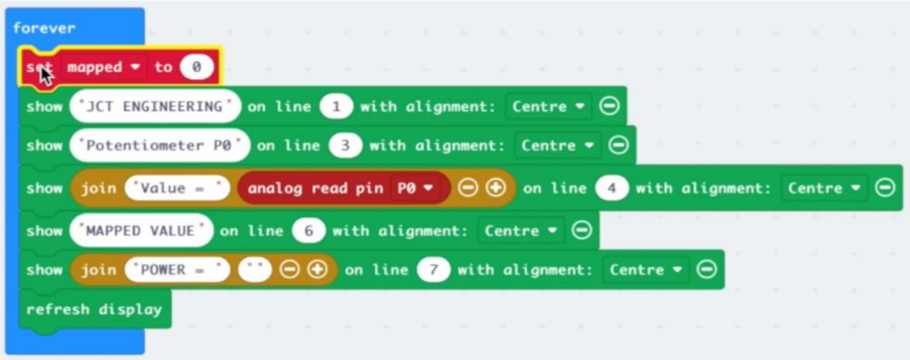
```
show join "POWER = " "" on line 7 with alignment: Centre
```



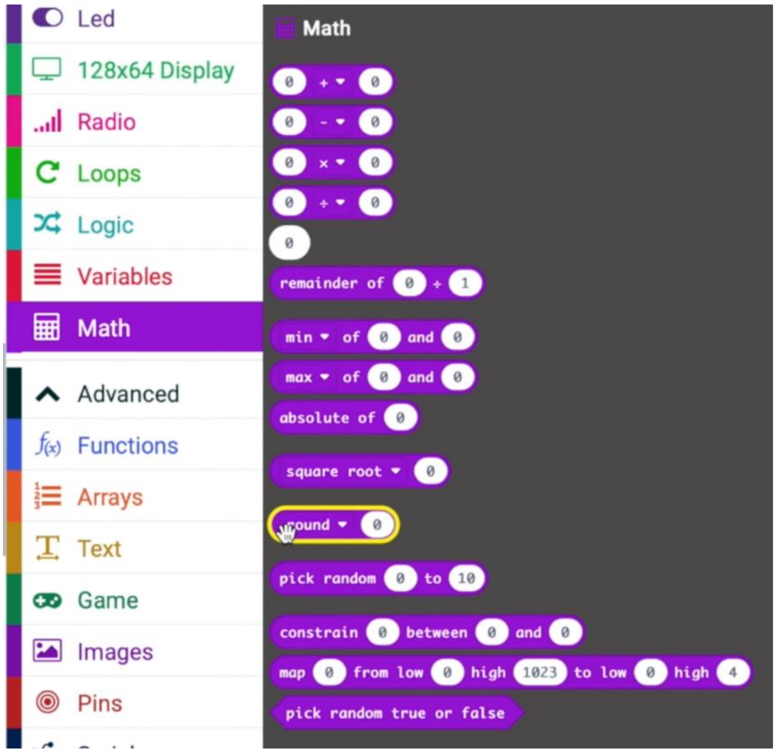
4:

Before we can display a value, we need to create a variable that will become the value we want to display.
We have been through this process previously in Module 8.

- Open the Variables drawer.
- Create a variable called "**mapped**"
- We now need to tell the Micro:bit how to calculate the mapped value. To do this, place the set ____ to _____ block from the variables drawer into the forever block as shown below.



```
forever
  set mapped to 0
  show "JCT ENGINEERING" on line 1 with alignment: Centre
  show "Potentiometer P0" on line 3 with alignment: Centre
  show join "Value = " analog read pin P0 on line 4 with alignment: Centre
  show "MAPPED VALUE" on line 6 with alignment: Centre
  show join "POWER = " "" on line 7 with alignment: Centre
  refresh display
```



5:

In previous activities, we have used the mapped value to control motors. Our reference for the output of the function has been the speed/direction of the motor.

If we display the mapped value directly on the screen, we will see a number with as many decimal places as the screen can fit.

This can be quite confusing and messy to display, so we shall round off the mapped value to the nearest whole number.

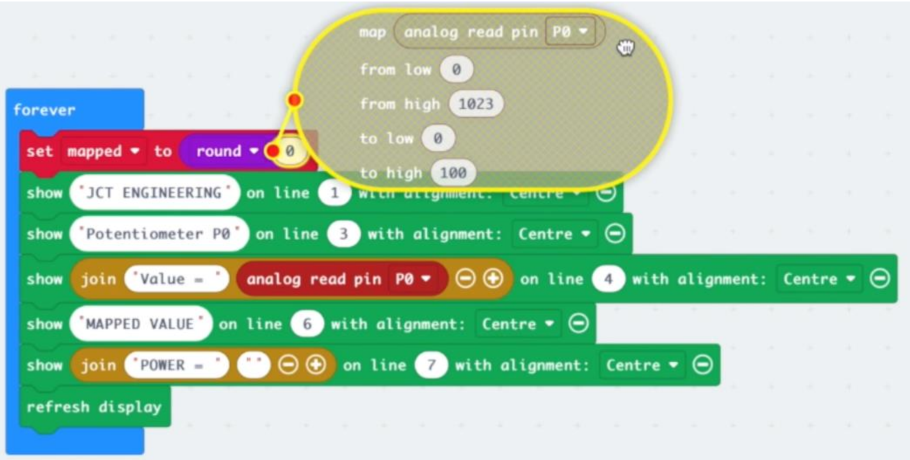
This way we get a simple, clean display of the mapped value.

- To round off the mapped value, we use the round_____ code block from the Math drawer of code.
- Place this block into the set _____ to _____ code from step four.

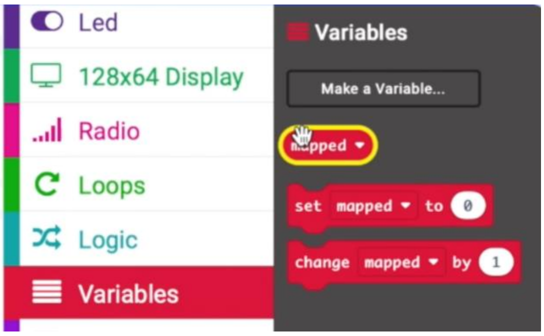
6:

Now that we have set up our mapped variable and rounded it off to the nearest whole number, we must now tell the micro:bit what input we want it to measure, as well as the limits we want it to map to. (e.g., a servo needs values up to 180, but a DC motor needs outputs up to 100.)

- From the Pins drawer, choose the map_____ code and insert it into the round______ code as shown to the right.
- We need to ensure the "to high____" value is 100, as we will use a DC motor later.

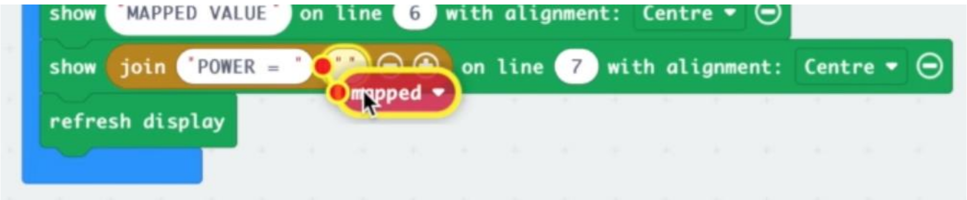


```
forever
  set mapped ▾ to round ▾ ( map ( analog read pin P0 ▾ )
                                from low 0
                                from high 1023
                                to low 0
                                to high 100 )
  show "JCT ENGINEERING" on line 1 with alignment: Centre ▾
  show "Potentiometer P0" on line 3 with alignment: Centre ▾
  show join "Value = " analog read pin P0 ▾ on line 4 with alignment: Centre ▾
  show "MAPPED VALUE" on line 6 with alignment: Centre ▾
  show join "POWER = " "" on line 7 with alignment: Centre ▾
  refresh display
```

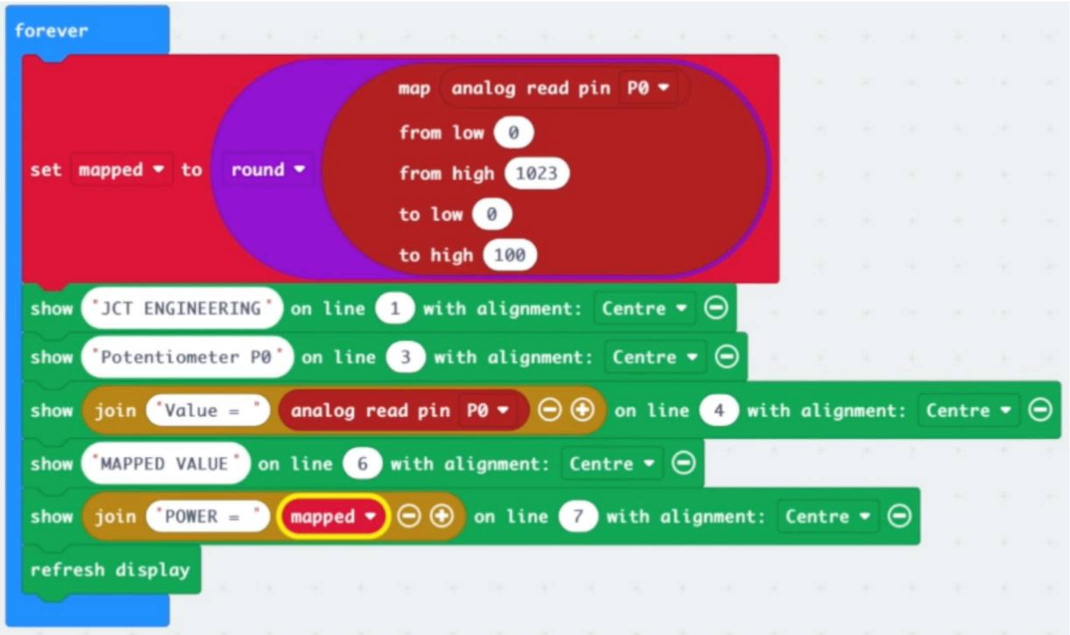


7:

- The Micro:bit now has been instructed to map the values from the potentiometer connected to Pin 0 *(0 -1023)* to a range suitable for power output to the DC motor (*0-100%*).
- To display the result of the Map function, open the Variables drawer and select the mapped variable.
- Drag the mapped variable out and place it in the second input of the join "Hello" "World" block as shown below (where analog read pin P0 used to be before we deleted it in step three).



```
show "MAPPED VALUE" on line 6 with alignment: Centre ▾
show join "POWER = " "" on line 7 with alignment: Centre ▾
                    mapped ▾
refresh display
```



```
forever
    set mapped ▾ to round ▾ ( map analog read pin P0 ▾
                                  from low 0
                                  from high 1023
                                  to low 0
                                  to high 100 )
    show "JCT ENGINEERING" on line 1 with alignment: Centre ▾ ⊖
    show "Potentiometer P0" on line 3 with alignment: Centre ▾ ⊖
    show join "Value = " analog read pin P0 ▾ ⊖⊕ on line 4 with alignment: Centre ▾ ⊖
    show "MAPPED VALUE" on line 6 with alignment: Centre ▾ ⊖
    show join "POWER = " mapped ▾ ⊖⊕ on line 7 with alignment: Centre ▾ ⊖
    refresh display
```

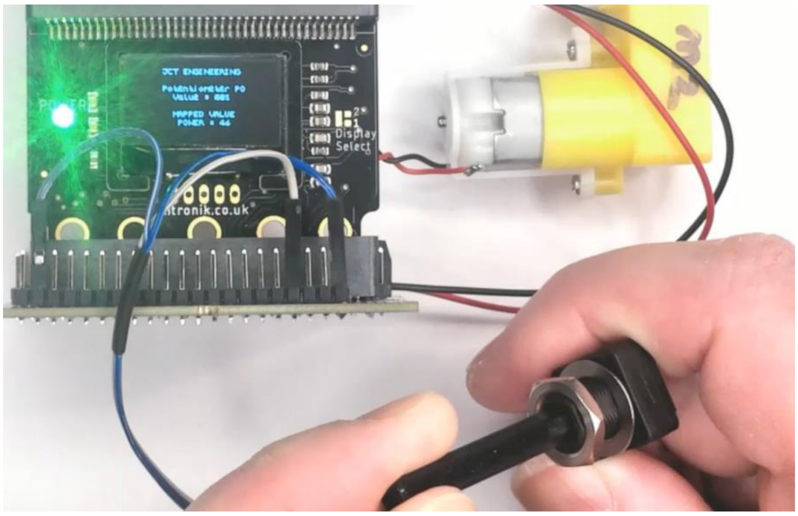
8:

The completed code is shown in the image above.

When downloaded and flashed to the micro:bit, the screen will display the range of values read from the Potentiometer (P0) and the mapped values which are scaled accordingly.

To check the accuracy, cycle from 0 – 1023 on the potentiometer, this will give a power output- from 0 – 100%.

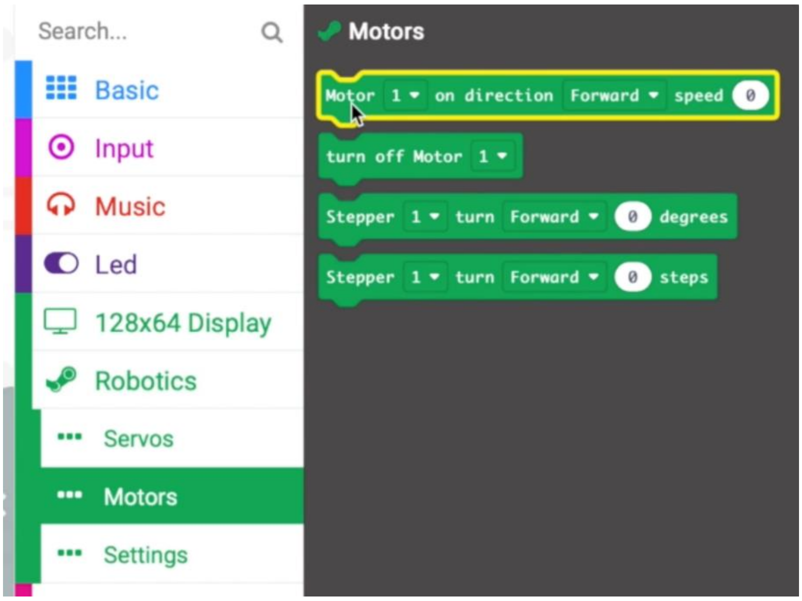
If you rotate the potentiometer to a reading of 512, you should have a mapped value (power) output of 50%.



9:

The purpose of the mapping function in this example is to control the power output to a DC motor, as we did in Module 8, Activity C.

To compare the values on screen to the operation of the motor, let’s add our DC motor on Motor output 1 of the All-In-One Robotics board.



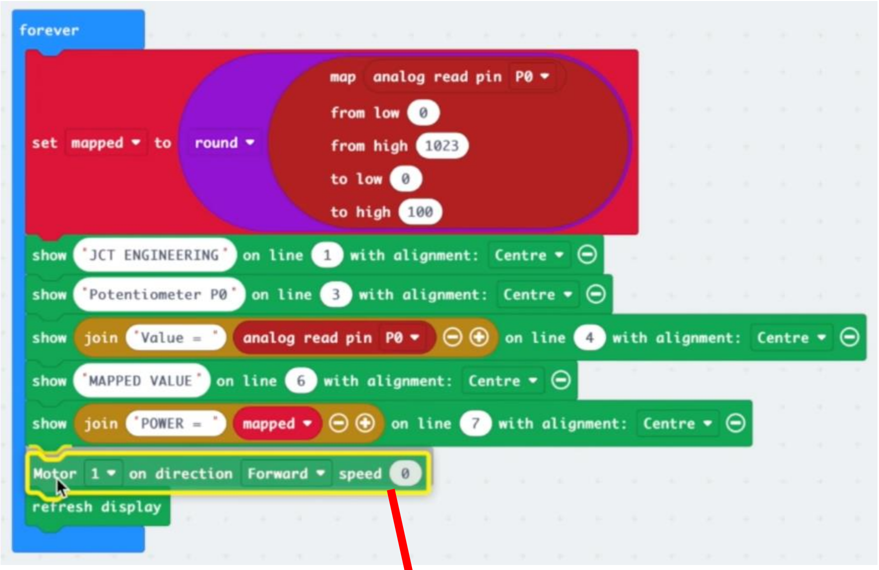
```
Motor 1 ▾ on direction Forward ▾ speed 0
turn off Motor 1 ▾
Stepper 1 ▾ turn Forward ▾ 0 degrees
Stepper 1 ▾ turn Forward ▾ 0 steps
```

- Add the All-In-One Robotics extension to your Makecode window.
- Go to the Motors drawer within the Robotics drawer.
- Choose the Motor ___ on direction ______ speed ______ code block.

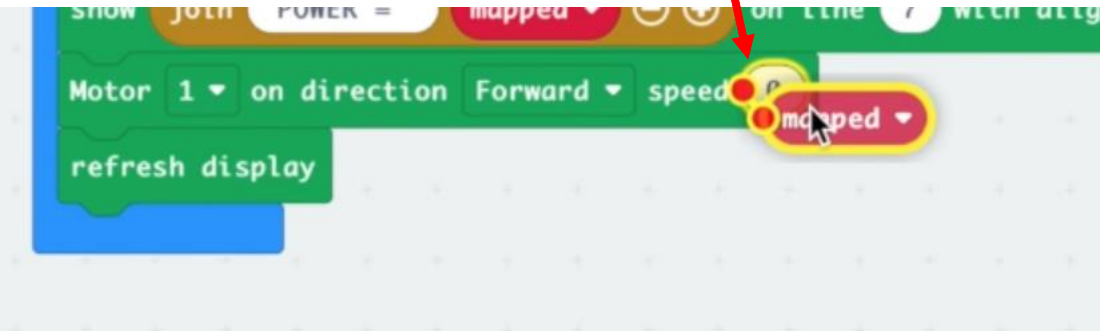
10:

Place the Motor ___ on direction _____ speed _____ code block into the forever block. It can be placed anywhere, but for clarity, let's place it after the last code we inserted.

To instruct the micro:bit what speed to set the motor to, insert the variable mapped into the speed input of the motor____on direction____speed____ code block. The mapped variable can be obtained from the Variables drawer, or it can also be duplicated from existing code in the window.



```
forever
  set mapped ▾ to round ▾ ( map analog read pin P0 ▾
                                from low 0
                                from high 1023
                                to low 0
                                to high 100 )
  show "JCT ENGINEERING" on line 1 with alignment: Centre ▾ ⊖
  show "Potentiometer P0" on line 3 with alignment: Centre ▾ ⊖
  show join "Value = " analog read pin P0 ▾ ⊖⊕ on line 4 with alignment: Centre ▾ ⊖
  show "MAPPED VALUE" on line 6 with alignment: Centre ▾ ⊖
  show join "POWER = " mapped ▾ ⊖⊕ on line 7 with alignment: Centre ▾ ⊖
  Motor 1 ▾ on direction Forward ▾ speed 0
  refresh display
```

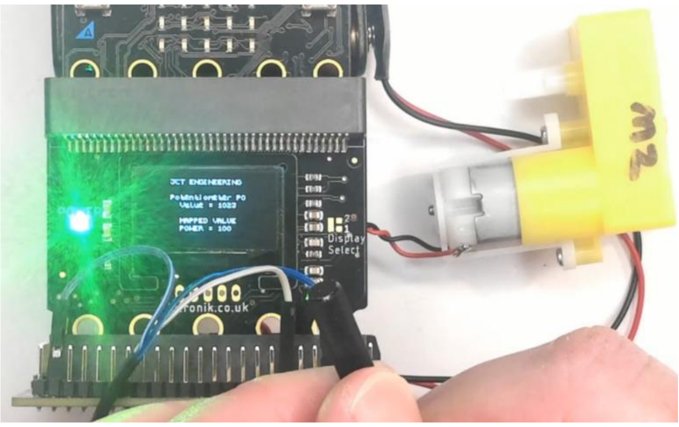


```
show join POWER = mapped ▾ ⊖⊕ on line 7 with alig
Motor 1 ▾ on direction Forward ▾ speed 0  mapped ▾
refresh display
```

When this updated code is downloaded and flashed to your micro:bit, the % power from the battery being sent to the motor will be shown on the screen.

As you increase power by turning the potentiometer, you will hear and see the motor changing speed.

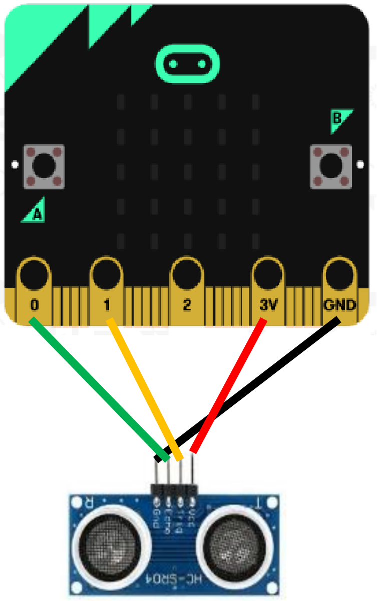
Note: The motor will not begin to turn immediately at 1%, as there is a minimum voltage it must receive before it can overcome inertia, load, friction, etc. Using a 6v battery, you will likely be in the mid-teens before your motor moves.



Has this activity enhanced your understanding of mapped values?
How could this activity support student understanding in mechatronics?
What practical activities in the workshop does this activity lend itself towards?

Module 10: Ultrasonic sensors

In the previous learning log from CPD 2020/2021, we engaged with LDRs in module number 6 by adding them to the Robotics board as additional sensors. This module will explain how to add and code an ultrasonic sensor and apply it to a geared DC motor and a servo motor.

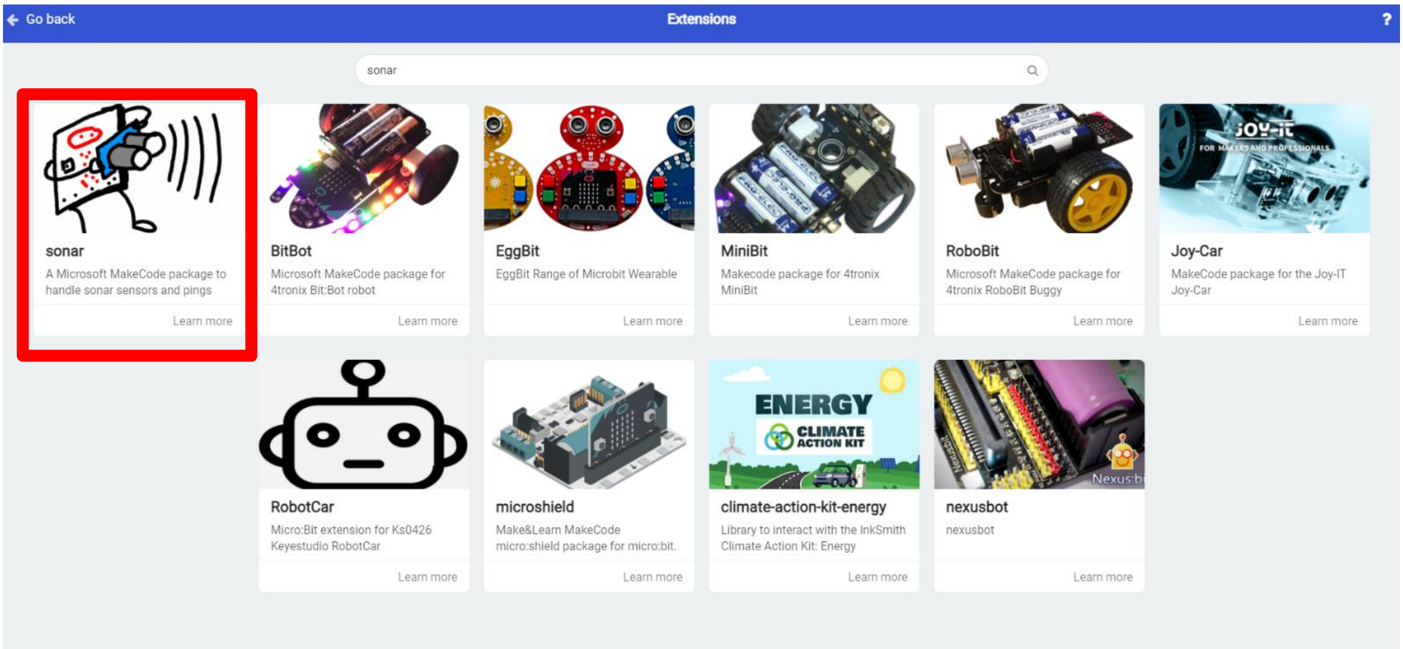


Possible Configurations
VCC: 3v
Gnd: Gnd
*Trig: P0,P1 or P2
*Echo: P0,P1 or P2

*Trig and Echo need to be identified in the code to decide the pin they will join to on the micro:bit or All-in-one Robotic board.

Videos supporting the activities in this module include instructions which outline how to configure the ultrasonic sensor to the micro:bit and Robotics board.

Extensions
To use the ultrasonic sensor, you need to add in the extension for the code to Make:code. Search for sonar in the 'extensions' library and add the one outlined in the diagram below.



sonar

sonar
A Microsoft MakeCode package to handle sonar sensors and pings
Learn more

BitBot
Microsoft MakeCode package for 4tronix Bit:Bot robot
Learn more

EggBit
EggBit Range of Microbit Wearable
Learn more

MiniBit
Makecode package for 4tronix MiniBit
Learn more

RoboBit
Microsoft MakeCode package for 4tronix RoboBit Buggy
Learn more

Joy-Car
MakeCode package for the Joy-IT Joy-Car
Learn more

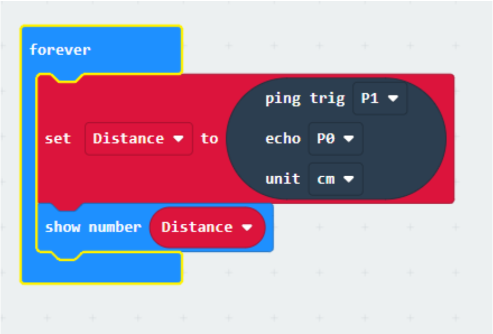
RobotCar
Micro:Bit extension for Ks0426 Keyestudio RobotCar
Learn more

microshield
Make&Learn MakeCode micro:shield package for micro:bit.
Learn more

climate-action-kit-energy
Library to interact with the InkSmith Climate Action Kit: Energy
Learn more

nexusbot
nexusbot
Learn more

Activity A: Calibrating the Ultrasonic

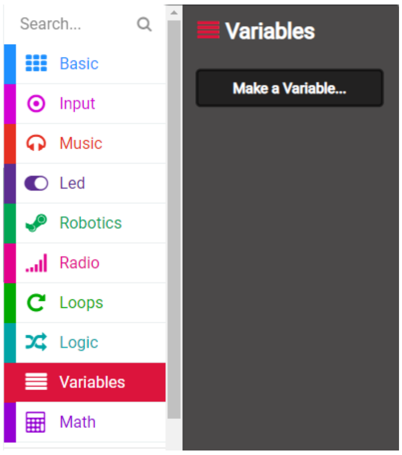


```
forever
    set Distance to ping trig P1
                         echo P0
                         unit cm
    show number Distance
```

Video

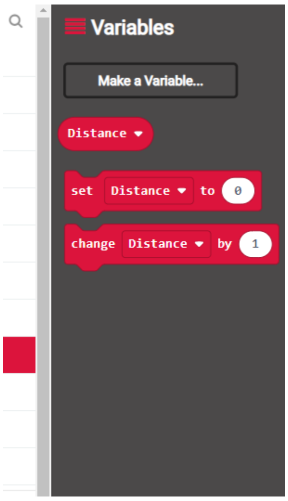
Why do we calibrate sensors? When using sensors for any reason, we should know it's maximum and minimum reading for a given situation. Without the readings, we might make the circuit not sensitive enough, or the opposite, too sensitive. You can access the video on how to calibrate a sensor by clicking or scanning the QR code.

How might you engage your students in this activity in Engineering?



1:

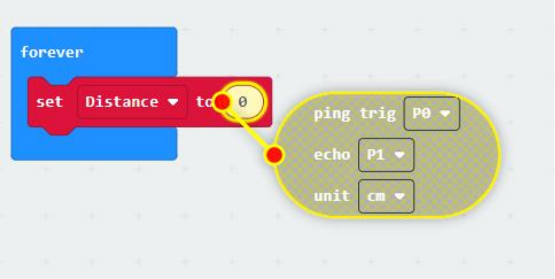
- When we are taking a reading, we will need to create a variable.
- Click on the variables tab and then the 'Make a Variable' button
- Just like in the subject of Maths, a variable can have any value.
- We will dictate where that variable comes from, in this case, the ultrasonic sensor



2:

- Name the variable something that will help make it identifiable, in this case 'distance'.

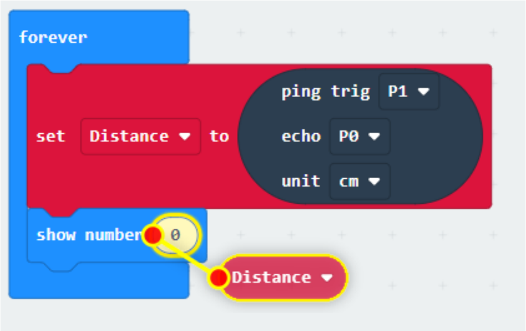
- When the variable is named, three blocks appear.
- 'Distance' allows for the use of variable to show numbers.
- 'set distance to __' allows us to create the link to the analogue value coming from the LDR
- 'Change distance by __' allows us to change the variable which can be useful for a countdown using a loop.



```
forever
  set Distance ▾ to 0
  ping trig P0 ▾
       echo P1 ▾
       unit cm ▾
```

3:

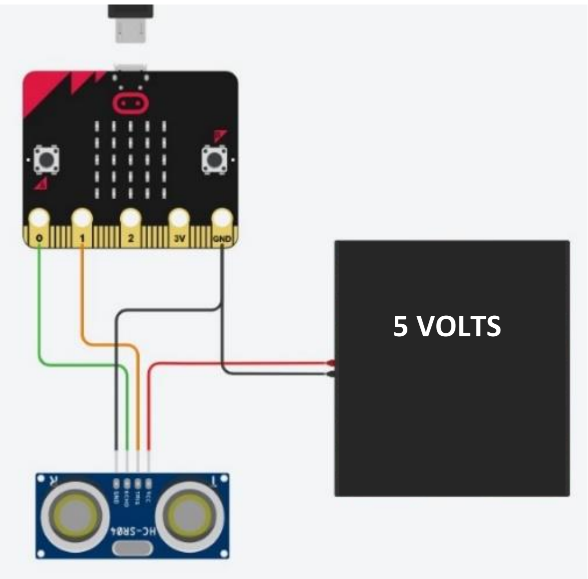
- From ‘basic’ we use the ‘forever’ block as we would like this to run continuously in the background.
- Using the ‘variables’ tab we use ‘set distance reading to __’ .
- From the ‘sonar’ tab we use the ‘ping trig P0, echo p0, unit µs’ and set it to whichever pins the ping and trig are connected to. Also choose your unit of measurement i.e., cm or inches.
- By doing this, we have said which pins the readings are coming from to measure the distance.



```
forever
  set Distance ▾ to ping trig P1 ▾
                    echo P0 ▾
                    unit cm ▾
  show number 0
  Distance ▾
```

4:

- Using the ‘Basic’ tab we can use ‘show number __’ to put the reading on the screen
- From ‘Variables’ we use ‘distance’ to show the value coming from the ultrasonic sensor.
- Now upload the code to the micro:bit and take the reading of the ultrasonic sensor, based on how far it is from an object, as this will be useful in the next activities.
- ***Download and test the code and configure the components as shown at the start of this module on page 45.***
- Place an object in front of the sensor and the reading will appear as a number in cm on the micro:bit screen.



Note: In some instances, the micro:bit may not be able to power the ultrasonic sensor. The LEDs may constantly be reading 0 (zero). If this is the case a separate power supply may be needed as shown on the image.

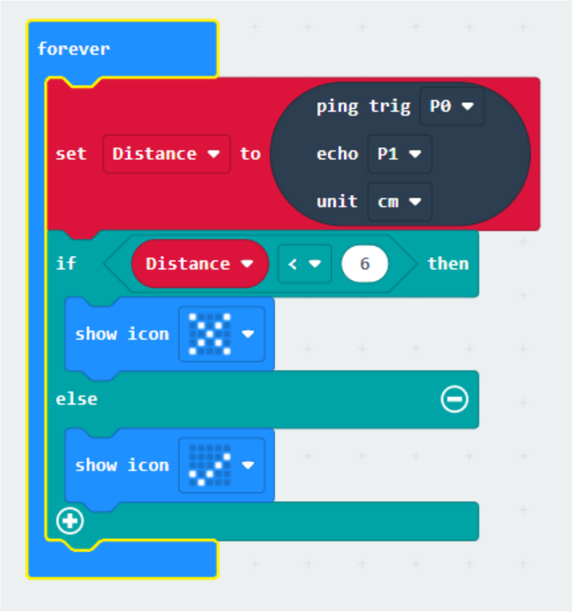
Green = TRIGG to P0
Orange = Echo to P1
Black = GND to GND
Red = VCC to 5 volts +

Reflection on your observations:

Activity B: Using the Ultrasonic Sensor to Show a Warning using the Micro:bit

Video

This activity explores code which can, using the screen on the micro:bit, send a signal that you are too close to an object by getting a signal from the ultrasonic sensor. Click or scan the QR code to access the supporting video for this activity.



```
forever
    set Distance to ping trig P0 echo P1 unit cm
    if Distance < 6 then
        show icon
    else
        show icon
```

This code is intended to fulfil the activity constraints.

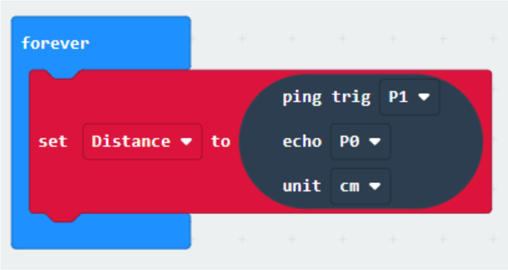
What is involved in the code?

Basic: Simple functions such as pausing a code.

Logic: Create a logic pattern where, if something is present it will do something, otherwise it will not.

Variables: This is a programmable unknown value that can be set to any value, in this instance the reading from the ultrasonic sensor.

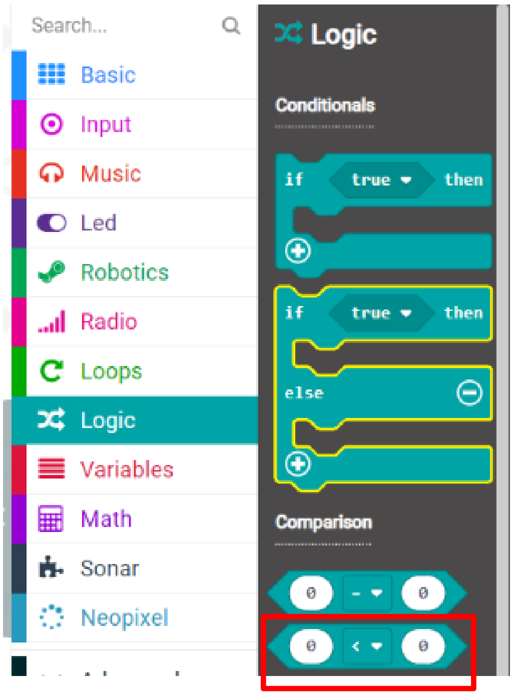
Sonar: taking the readings from the sonar using the pins and values set in the code.



```
forever
    set Distance to ping trig P1 echo P0 unit cm
```

1:

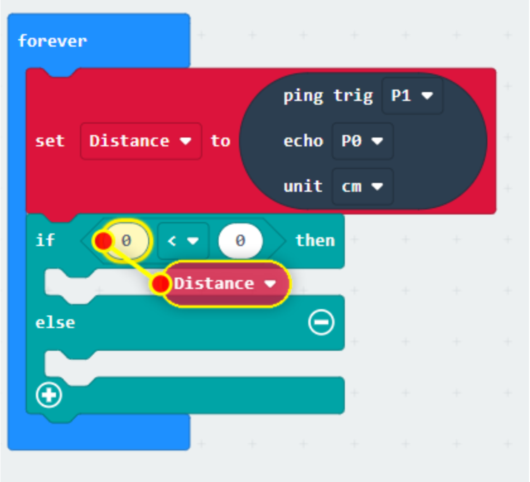
- Using the first 3 steps from ‘Activity 1’ your coding blocks should be aligned with the diagram on the left.



2:

- Using the ‘logic’ tab, use the ‘If true then __ else __’ Block and add it to the code.
- Using the comparison section as shown in the image, select the ‘___ < ___’ option.
- Add it to the ‘true’ part of the logic gate.

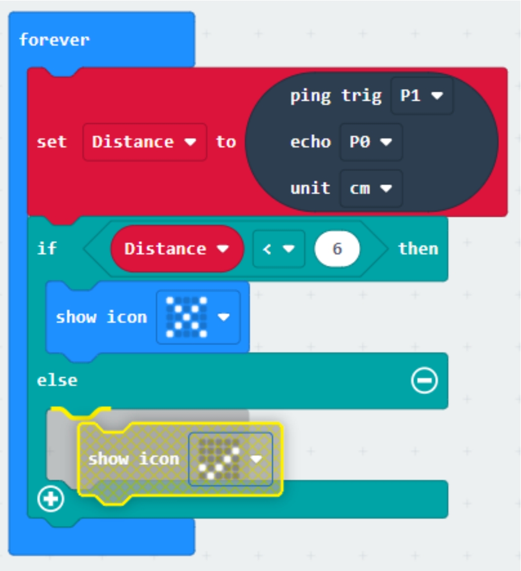
*This has now created a comparison stating that ‘if ___ < ___ then do _____ if it’s not there do ____’



```
forever
  set Distance to ping trig P1
                       echo P0
                       unit cm
  if 0 < 0 then
  Distance
  else
```

3:

- Using the ‘variables’ tab, select the ‘distance’ block and add it to the comparison block as shown.
- From here you can create a distance stated using the calibration used from ‘Activity 1’
- Once you have decided on a distance that you would like to warn people that they are too close, enter it into the numerical option of the comparison.



```
forever
  set Distance to ping trig P1
                       echo P0
                       unit cm
  if Distance < 6 then
    show icon
  else
    show icon
```

4:

- For the option code here, we have used 6cm as the distance we want from the ultrasonic sensor for the micro:bit to warn someone they are too close.
- Using the ‘basic’ tab, select the ‘show icon’ choosing which icon you want to show on screen, in our case an ‘X’ or a ‘✓’ .

So, what is happening in this code?
We are using the sensor to detect the distances in cm. When the distance is greater than 6 cm from the ultrasonic sensor it shows a ‘✓’ on screen but when you go closer than 6 cm from the sensor the icon changes to a ‘X’ and you are warned you are too close.

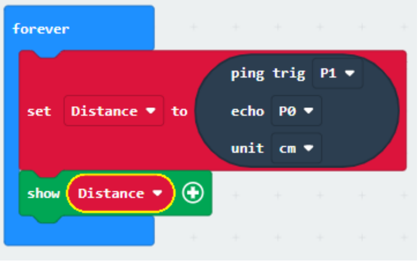
Download and test the code with the micro:bit. The Ultrasonic sensor should be configured to the board the same way as shown in activity 1 of this module on page 45.

If your micro:bit cannot get a reading from the microbit, use the configuration for the components on page 47.

How could you use this with your students?

Activity C: Using the Ultrasonic Sensor to Show a Code on an External Screen

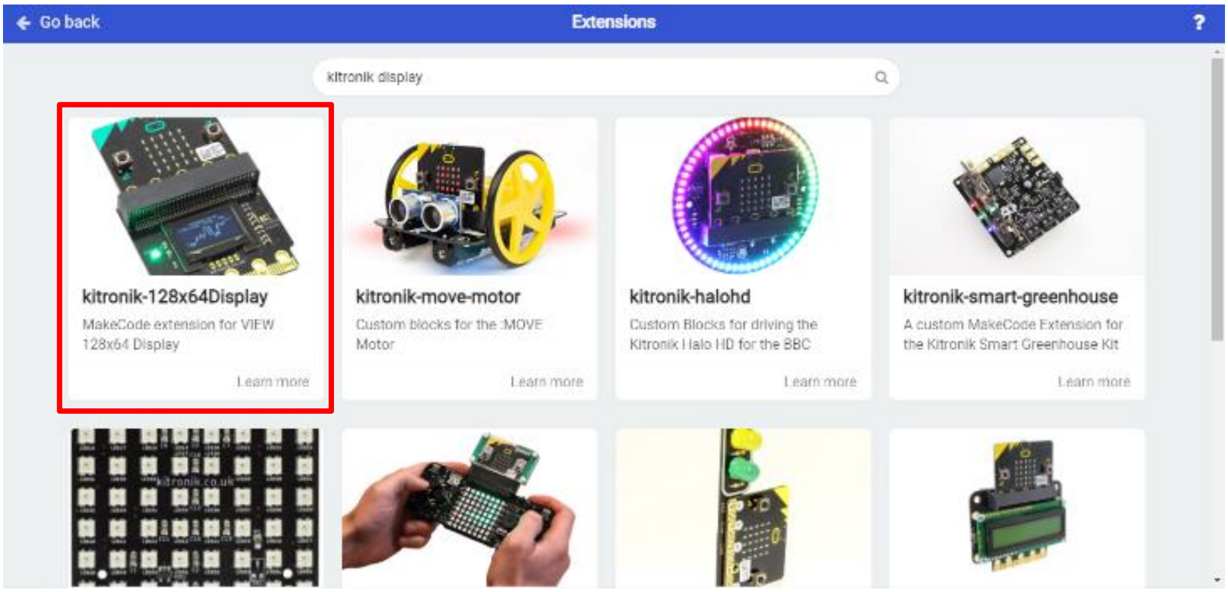
Video



```
forever
  set Distance ▾ to  ping trig P1 ▾
                     echo P0 ▾
                     unit cm ▾
  show Distance ▾ ⊕
```

The code on the left is intended to fulfil the activity. The following coding tabs are required in the menu:

- Basic: the forever block keeps the code running in the background for the time the micro:bit is powered up.
- Variables: tab allows you to set your measurement from any source, in this case the ultrasonic sensor.
- 128x64: the extension when added will allow you to put many different things on the screen.



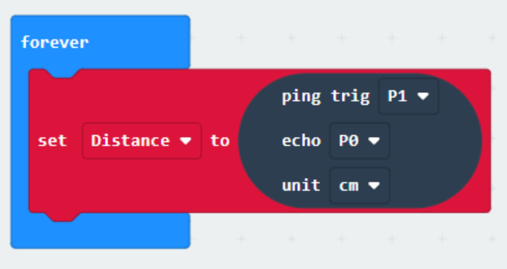
kitronik-128x64Display
MakeCode extension for VIEW 128x64 Display

kitronik-move-motor
Custom blocks for the :MOVE Motor

kitronik-halohd
Custom Blocks for driving the Kitronik Halo HD for the BBC

kitronik-smart-greenhouse
A custom MakeCode Extension for the Kitronik Smart Greenhouse Kit

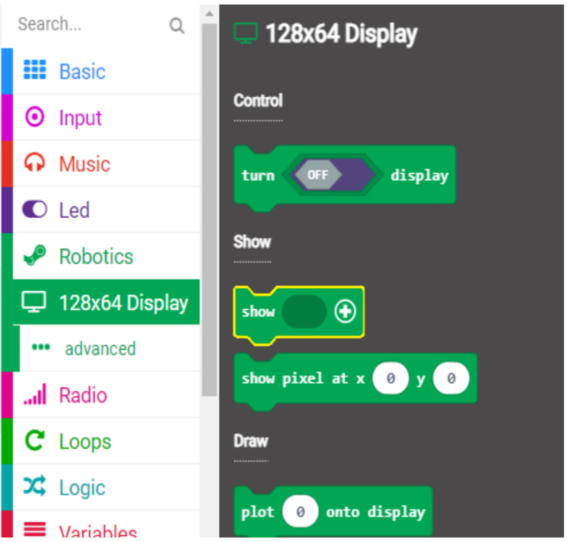
To engage with this activity, you will need to add in another extension. This one is found in the extension library under the search of Kitronik Display. You can see in the screen shot above which extension to choose.



```
forever
  set Distance ▾ to  ping trig P1 ▾
                     echo P0 ▾
                     unit cm ▾
```

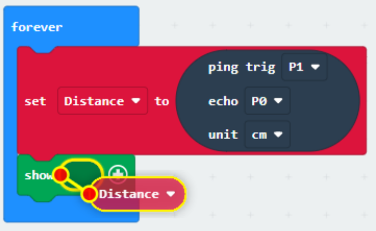
1:

- Using the first 3 steps from 'Activity 1'. Your coding blocks should be aligned with the diagram on the left.



2:

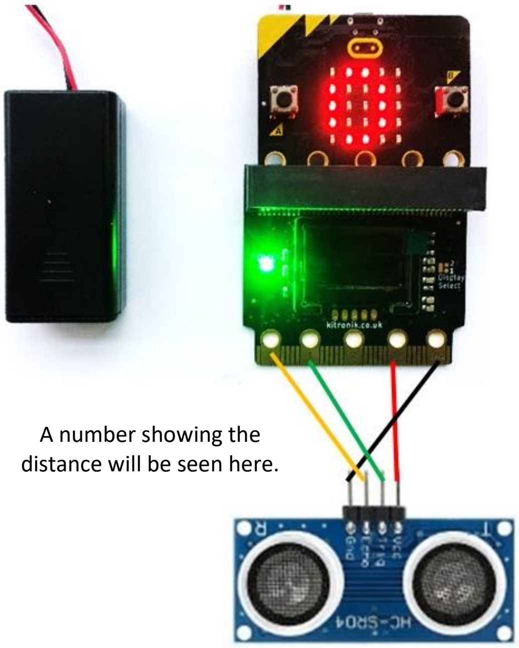
- Once you have added the extension, you can now find the blocks under ‘128x64 Display’
- There are many different options to choose from to define what you would like to show on screen
- As you can see in the picture to the left, there’s many different options for you to show different pixels on screen
- The option we need to show the reading from the ultrasonic sensor is ‘show___’



```
forever
  set Distance to ping trig P1
                       echo P0
                       unit cm
  show Distance
```

3:

- Using the ‘variables’ tab, add in the ‘distance’ block into the show block as shown
- This now tells the screen to display the distance (in cm) being read by the ultrasonic sensor.



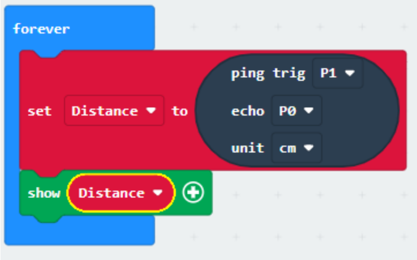
A number showing the distance will be seen here.

4:

- The code may be downloaded and tested at this point before progressing to the next part of the activity.
- Configure the micro:bit to the screen and ultrasonic sensor to the screen temporarily using crocodile clips.
- The distance will be visible in the top left of the screen as highlighted.

Configuration of the sensor in the image is as follows:
Black: GND to GND
Red: VCC to 3V
Green: TRIGG to P1

Application Challenge: using the code to make a reversing sensor

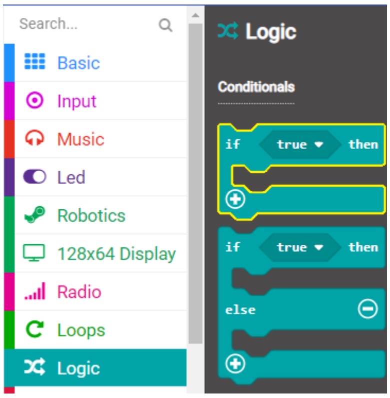


```
forever
  set Distance to ping trig P1
                       echo P0
                       unit cm
  show Distance
```

1:

- Once the code has been set up like this, we can add different commands to do different commands
- The code here mimics in a similar way to what can happen on screen in a vehicle when using parking sensors
- In order to make a sensor, we will need different sounds depending on the distance from the sensor
- We also need to allow for different distances to differentiate between the sounds

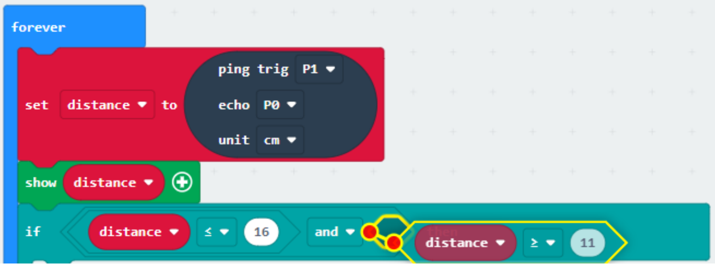
2:



```
if true then
+

if true then
else
+
```

- Using logic gates, we can create a sequence which will allow different reactions for when different readings are taken by the ultrasonic
- Using the 'if true then', we can set up a few different scenarios
- We can use the comparison section to add in a block that references a distance less than a given measurement from the sensor

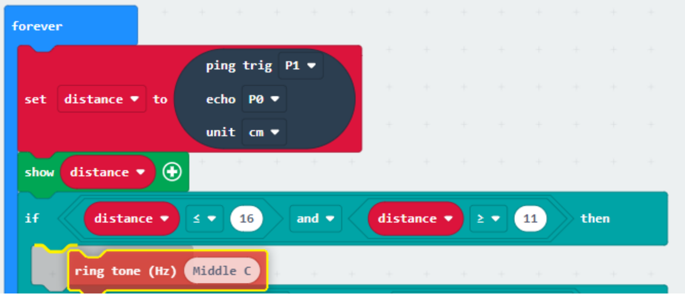
3:



```
forever
  set distance to ping trig P1
                        echo P0
                        unit cm
  show distance
  if distance ≤ 16 and distance ≥ 11
```

- When using the 'variable' in the place of numbers we have created the premise that if the object sensed is within a certain distance
- Using the 'Boolean' block 'and' we can set a range of distances as shown from 16 to 11
- We use a comparison block adding the variable 'distance' to the start of it
- Making sure that the distances read from ≤ to ≥ therefore locking it between the 2 values

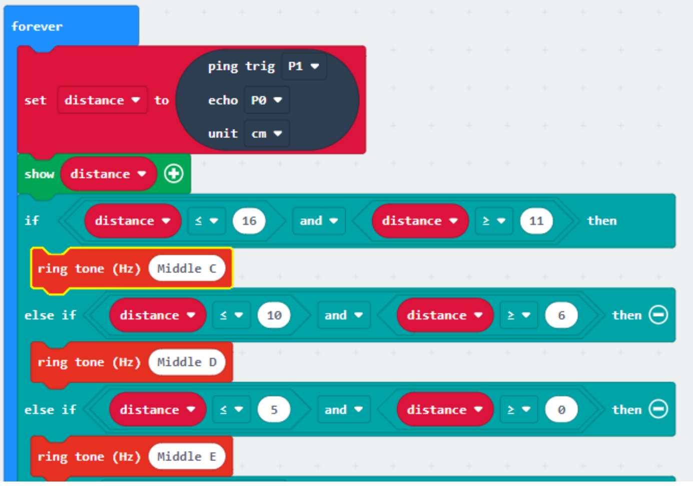
4:



```
forever
  set distance to ping trig P1
                        echo P0
                        unit cm
  show distance
  if distance ≤ 16 and distance ≥ 11 then
    ring tone (Hz) Middle C
```

- Using the 'music' tab we can add a tone to be played
- 'ring tone (Hz) _____' from here you can choose the tone you want to play. This example starts at 'middle c'
- From here we can duplicate the blocks and add more using the left click on the mouse
- You can change the tone to simulate a vehicle getting progressively closer to an object

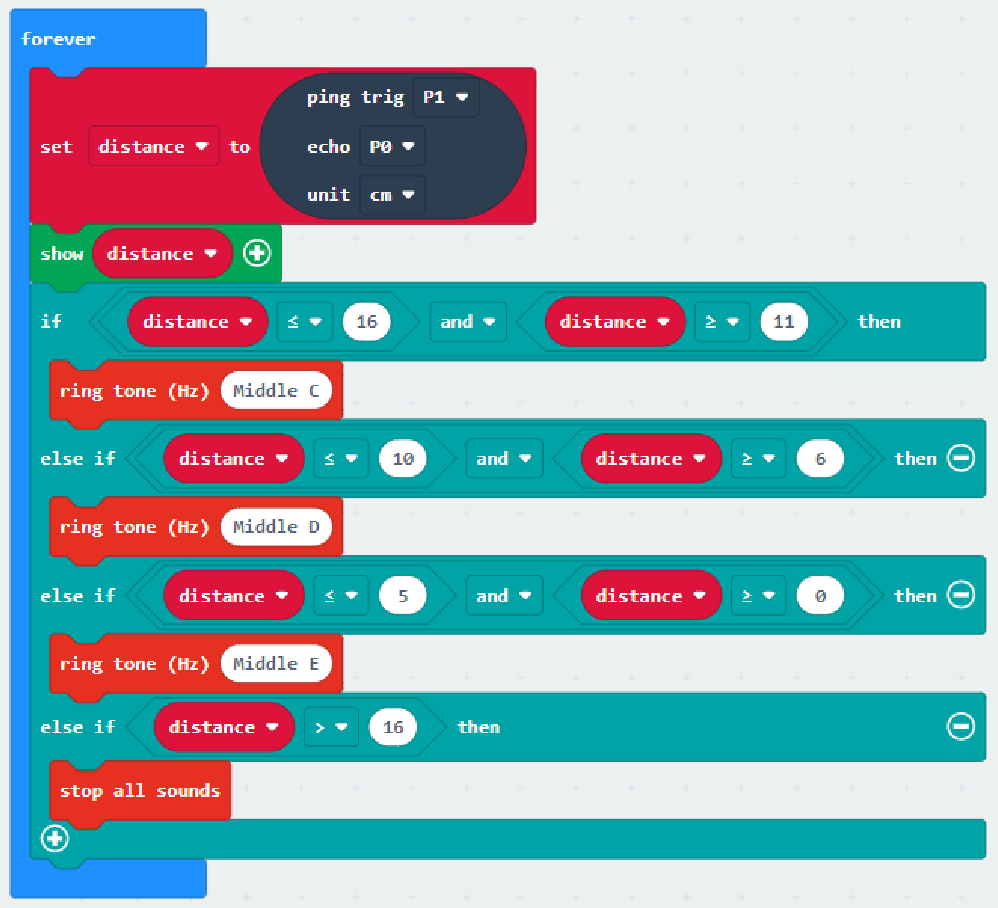
5:



```
forever
  set distance to ping trig P1
                        echo P0
                        unit cm
  show distance
  if distance ≤ 16 and distance ≥ 11 then
    ring tone (Hz) Middle C
  else if distance ≤ 10 and distance ≥ 6 then
    ring tone (Hz) Middle D
  else if distance ≤ 5 and distance ≥ 0 then
    ring tone (Hz) Middle E
```

- As you can see here that the progression is indicated by the numbers decreasing and the tone is increasing in pitch

- The code needs to tell the tone to stop if it's been started. We can do that by using the greater than comparison and 'stop all sounds'



```
forever
  set distance to ping trig P1
                  echo P0
                  unit cm
  show distance
  if distance ≤ 16 and distance ≥ 11 then
    ring tone (Hz) Middle C
  else if distance ≤ 10 and distance ≥ 6 then
    ring tone (Hz) Middle D
  else if distance ≤ 5 and distance ≥ 0 then
    ring tone (Hz) Middle E
  else if distance > 16 then
    stop all sounds
```

6:

- This code will solve the challenge
- Are there any other alternative options?
- What code would you use?

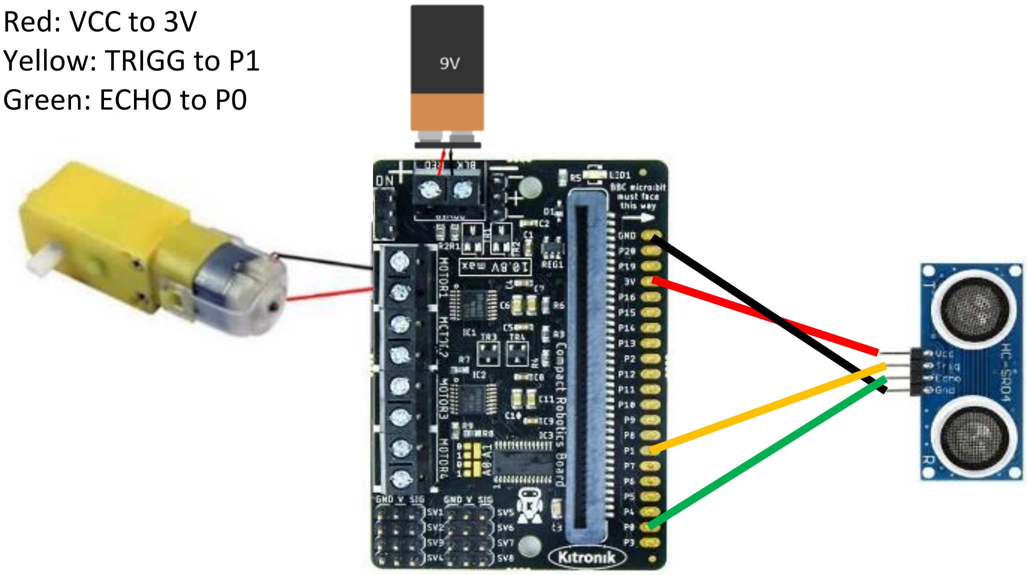
How could you use this with your students? What way could you integrate this into project/classroom activities?

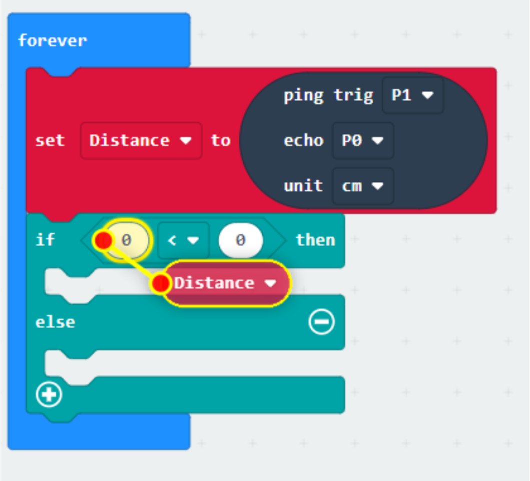
Activity D: Using the Ultrasonic Sensor to Turn Off a Motor

Video

The DC Motor can be controlled by the ultrasonic sensor using the all-in-one robotics board. The components can be configured like the image below. Click or scan the QR code to access the supporting video for this activity.
The configuration of the components is outlined in the image below and as follows:

Black: GND to GND
Red: VCC to 3V
Yellow: TRIGG to P1
Green: ECHO to P0

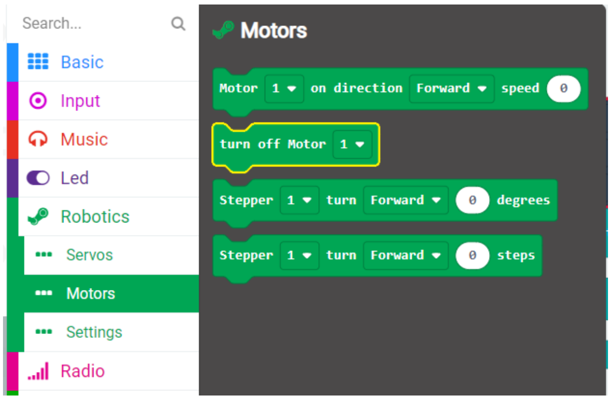




```
forever
    set Distance ▾ to ping trig P1 ▾
                      echo P0 ▾
                      unit cm ▾
    if 0 < ▾ 0 then
        Distance ▾
    else
    ⊕
```

1:

- Using the learning from ‘activity 2’, we can build the code to the same point as shown on the left
- This code is the building block for our task, to stop the motor when we get too close to an object
- From here, you can again choose a distance that you would like the motor to stop from



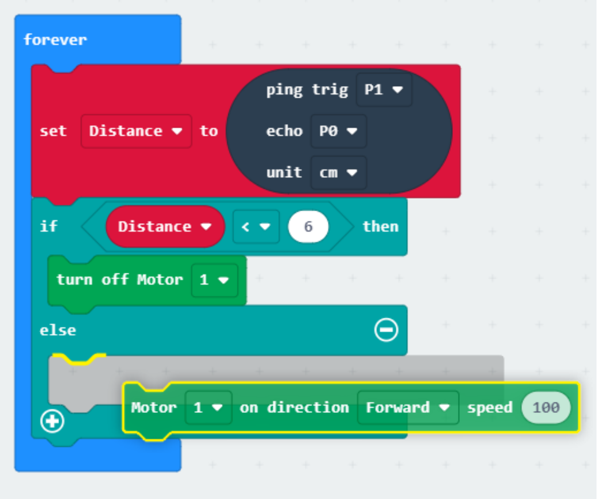
```
Motor 1 ▾ on direction Forward ▾ speed 0
turn off Motor 1 ▾
Stepper 1 ▾ turn Forward ▾ 0 degrees
Stepper 1 ▾ turn Forward ▾ 0 steps
```

2:

- Using the ‘turn off motor __’ from the robotics tab, we can tell the motor to turn off inside this distance

*the ‘all-in-one robotics board’ extension needs to be added to access this tab

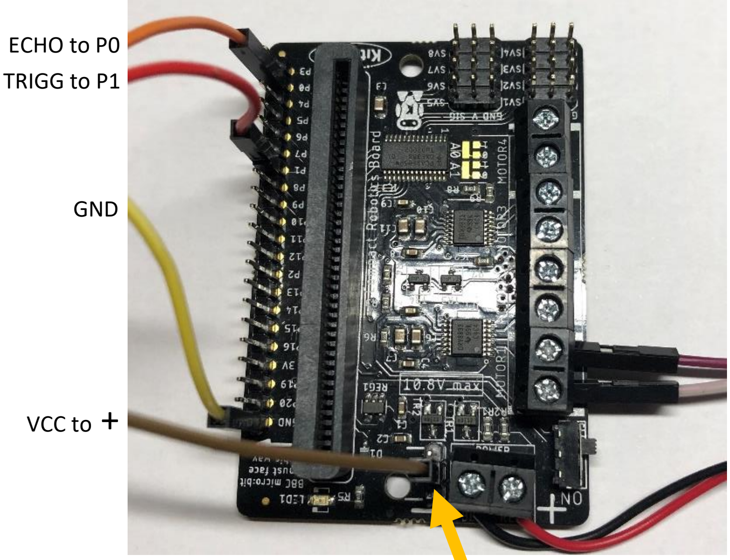
- We will also be using the ‘motor __ on direction ___ speed ___’ block to make the DC motor turn



```
forever
    set Distance to ping trig P1
                         echo P0
                         unit cm
    if Distance < 6 then
        turn off Motor 1
    else
        Motor 1 on direction Forward speed 100
```

3:

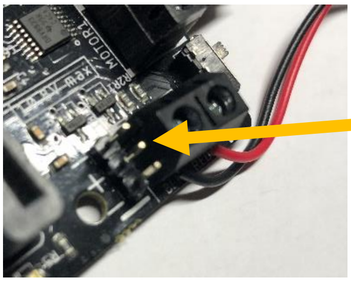
- By adding the ‘turn off motor__’ we have now told the motor to turn off when the ultrasonic sensor is 6cm away from an object
- Adding in the ‘Motor __ on Direction__ speed__’ tells the system that it’s to go until it is 6cm from an object
- ***Download and test the code on the configured components as shown in the image on the previous page.***



Note: If the all-in-one robotics board is not able to power the ultrasonic sensor from the pin headers on the link pads, it is possible to supply additional power. This can be taken from the middle of the three auxiliary power pins next to the terminal block for the battery as indicated in the image opposite. To do this, disconnect the VCC from the 3V supply and connect it to the middle (labelled +) beside the terminal block. This will supply the Ultrasonic sensor with the same voltage being supplied by the battery, rather than the 3v available on the 3v header pin.

Warning: Connecting an incorrect wire from the sensor can damage the sensor!

If connecting to the auxiliary power pin, ensure the battery voltage does not exceed 6v.



There are 3 pins beside the battery terminal block. There are 3 auxiliary power pins beside the battery terminal block.

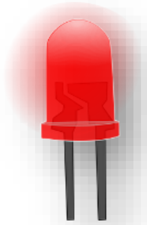
Connect the VCC to the middle which is the + .

Challenge:

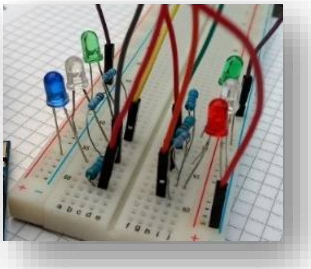
What are the implications of this code? E.g., the vehicle will stop when it gets within 6cm of an object, what could we do to change this? What could we do to make it change direction instead?

Module 11: Addressable LEDs (Neopixels)

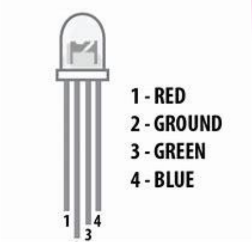
Adding lots of LEDs into an engineering project can be a complicated process, often ending up as a spider's web of switches, wires, resistors, and LEDs. Addressable LEDs are the latest evolution of LED technology. They can make this process easier. A brief overview is illustrated below.



A basic LED requires two wires and emits one colour, similar to a traditional bulb.



Adding multiple LEDs, resistors and wires becomes messy and frustrating.



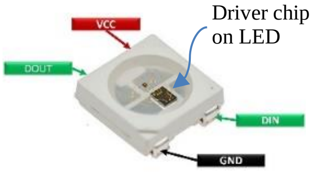
Red, Green, Blue (RGB) LEDs can display multiple colours and are neater.

RGB LED strips are long lengths of LEDs which can all display multiple colours.

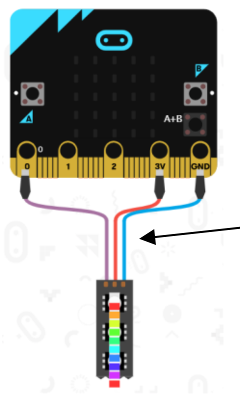
RGB LED strips are commonly used too for decorative lighting as shown.

Neopixels are unique because each LED contains a controller chip, so each one can be a different colour and they can be animated using code.

The driver chip in Neopixels allows long lengths of LEDs to be controlled via a single data signal. This data signal needs a controller such as our Micro:bit. The data signal needs only one wire to communicate between the controller and the strip, so attaching a long length of LEDs can be as simple as connecting a servo motor.



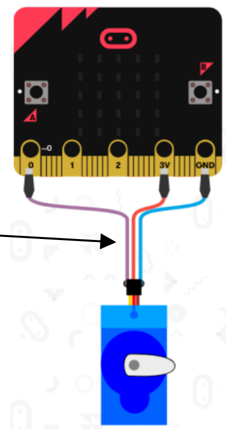
Driver chip on LED



Neopixel strip

Simple three wire connection

- ❖ Ground (-ve)
- ❖ Power (+ve)
- ❖ Data signal

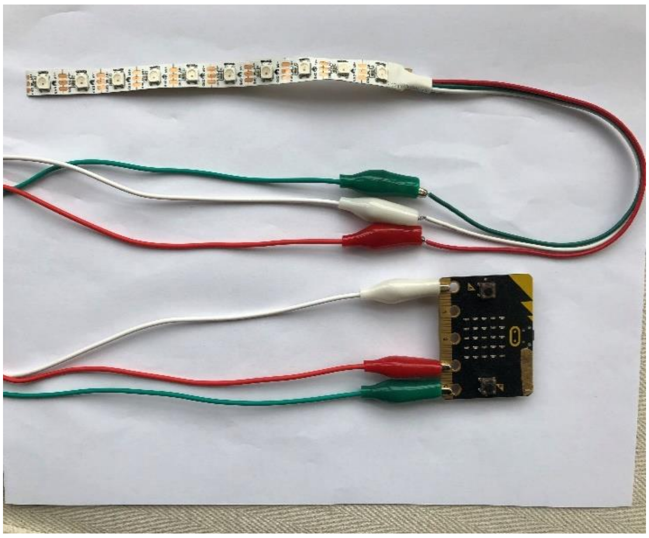


Servo motor

Connecting your Neopixel strip to the Micro:bit

Video

Use the video titled ‘Getting to know your neopixels’ to guide you through this section. It can be accessed by clicking or scanning the QR code opposite.



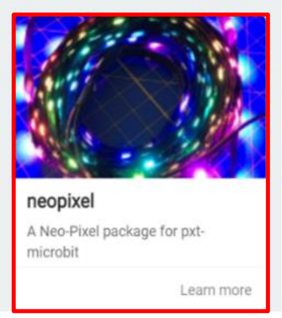
The video will explain the principle of operation of the LED and the challenges of wiring it to the micro:bit. The image and instructions are provided below for your convenience and as a quick reference.

- The red on the Neopixel joins to 3V.
- The white lead on the Neopixel joins to the GND on the micro:bit.
- The middle lead (green) on the Neopixel joins to P0 on the micro:bit.

This configuration applies to each of the following activities A-E in this module.

Ensure there is very good contact between the crocodile clip and the wires on the LED to ensure a good connection.

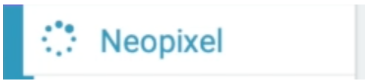
Adding the Neopixel extension to Makecode for micro:bit

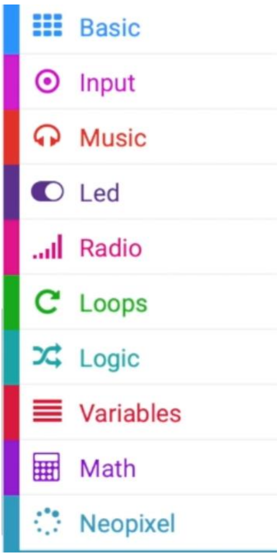


neopixel

A Neo-Pixel package for pxt-microbit

For your micro:bit to communicate correctly with your LED strip, you need to add the Neopixel extension to your Makecode session. This adds the Neopixel drawer to Makecode, which contains blocks of code you can use to control your LED strip.





From the code drawers on the left, the following are required in this module and are colour coded throughout the activities.

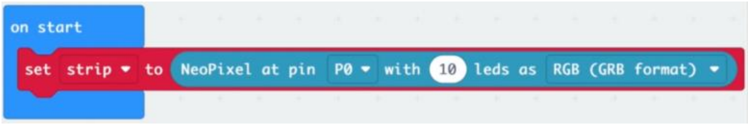
- The Basic drawer
- The Input drawer
- The Loops drawer
- The Variables drawer
- The Neopixel drawer

Video

Activity A: Light it up!
Illuminating your Neopixel

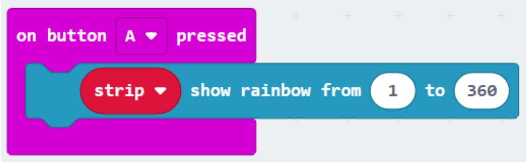
This activity explores code which can be used to illuminate and clear your Neopixel strip. This is done using basic inputs and Neopixel code. This is worked through in detail in the accompanying video. Click or scan the QR code to access the video.

Lighting up the rainbow



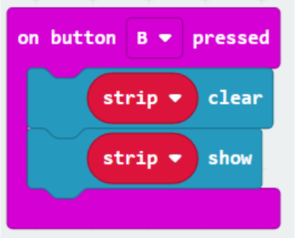
```
on start
  set strip to NeoPixel at pin P0 with 10 leds as RGB (GRB format)
```

- Find the set strip to... block at the top of the Neopixel drawer.
- Drag this into the on start block.
- Update the set strip to... block to contain **10** LEDs. This lets the micro:bit know how many instructions it should send.



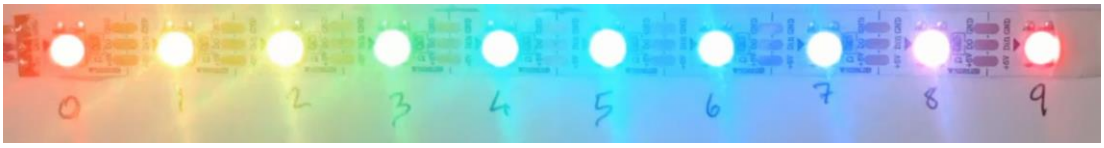
```
on button A pressed
  strip show rainbow from 1 to 360
```

- Add the show rainbow block from the Neopixel drawer to the on button A pressed input. – This turns on the rainbow.



```
on button B pressed
  strip clear
  strip show
```

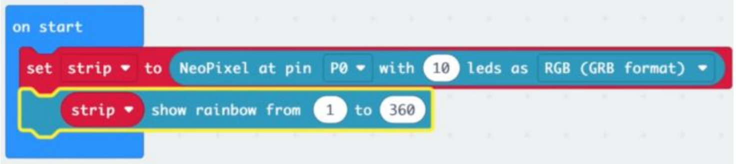
- The strip is turned off by adding the strip clear and strip show blocks to the on button B pressed block.



Video

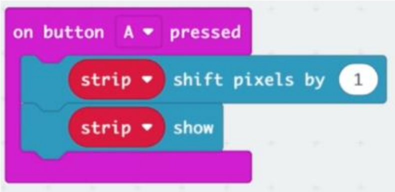
Activity B: Basic Animation
Using Shift & Rotate commands

This activity introduces code which can be used to animate the Neopixel strip. This is a follow on from the previous activity and again basic inputs are used to create effects on the strip. This is worked through in detail in the accompanying video.



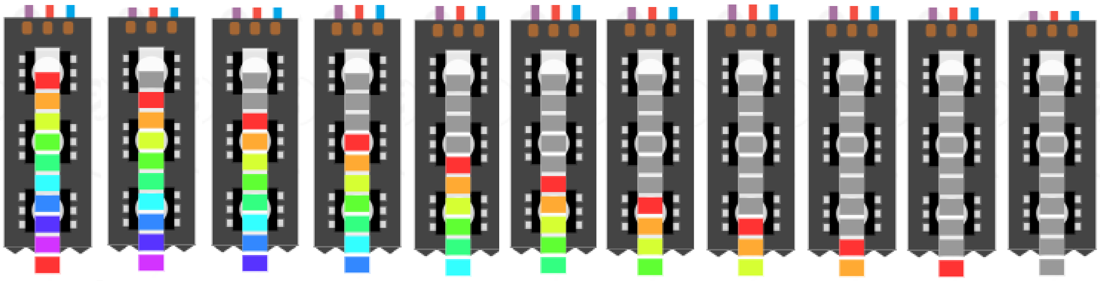
```
on start
  set strip to NeoPixel at pin P0 with 10 leds as RGB (GRB format)
  strip show rainbow from 1 to 360
```

- Like the last activity, drag the set strip to... block onto the on start block, updating it to show 10 LEDs.
- Now add the show rainbow block into the on start block as shown – *This turns on the rainbow automatically.*

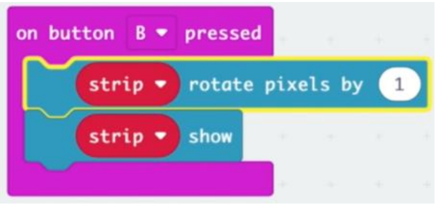


```
on button A pressed
  strip shift pixels by 1
  strip show
```

- Add a on button A pressed block and put a strip shift pixels and a strip show block into it. -This pushes the LEDs from the start to the end of the strip. It will move forward one LED each time it is pushed.



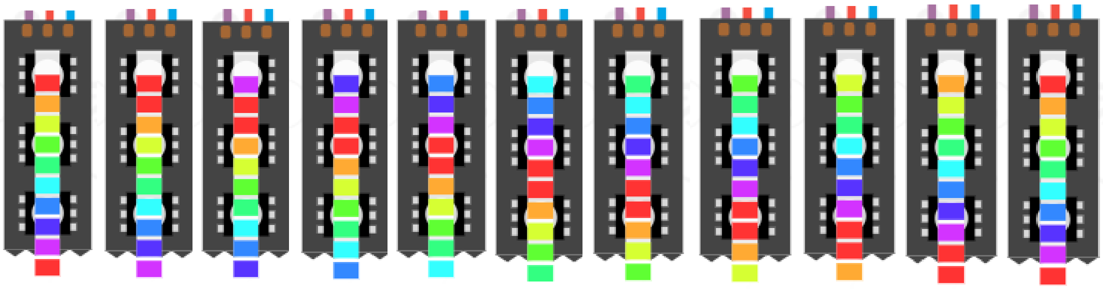
This shows the effect of shifting the pixels by one each time button A is pressed. The LEDs are pushed from the start of the line until they are all pushed off. The strip will then be turned off.



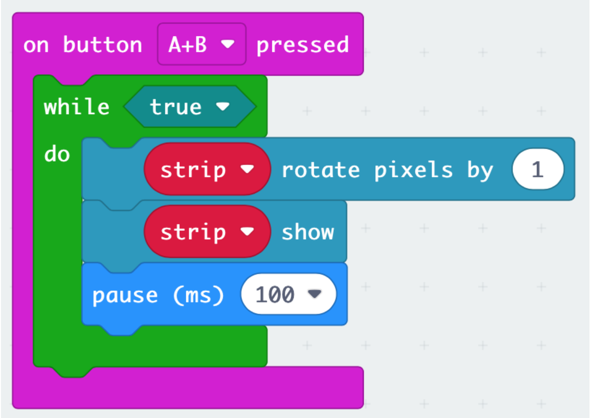
```
on button B ▾ pressed
    strip ▾ rotate pixels by 1
    strip ▾ show
```

Rotating Pixels

- Add an on button B pressed block.
- Insert the strip rotate pixels by block and the strip show blocks into it. – This tells the micro:bit to push the LEDs forward one step, but with this command the LED pushed off at the end re-joins the back to the queue.



This shows the effect of rotating all the pixels by one each time button B is pressed. The LEDs that are pushed off the line go back to the end of the line and the cycle repeats continuously.



```
on button A+B ▾ pressed
    while true ▾
    do
        strip ▾ rotate pixels by 1
        strip ▾ show
        pause (ms) 100 ▾
```

To automate the rotation:

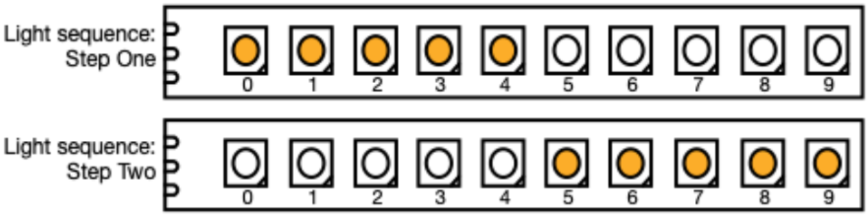
- Insert an on button A+B pressed input block.
- Add the strip rotate pixels and the strip show blocks as in the previous activity.
- Now add the while block from the loop drawer and modify it to "true" as shown opposite. This creates a continuous loop effect for our rotation after button A+B are pressed.
- Add a pause block to slow down the loop speed, making it easier to see.

Download and test the code on the micro:bit.

Activity C: School Ahead Warning Beacons

Video

This activity explores a practical example of how to manually create a flashing beacon on the Neopixel, and how to create an automated sequence also. There is a follow up creative activity to accompany this coding activity. Click or scan the QR code to access the video.

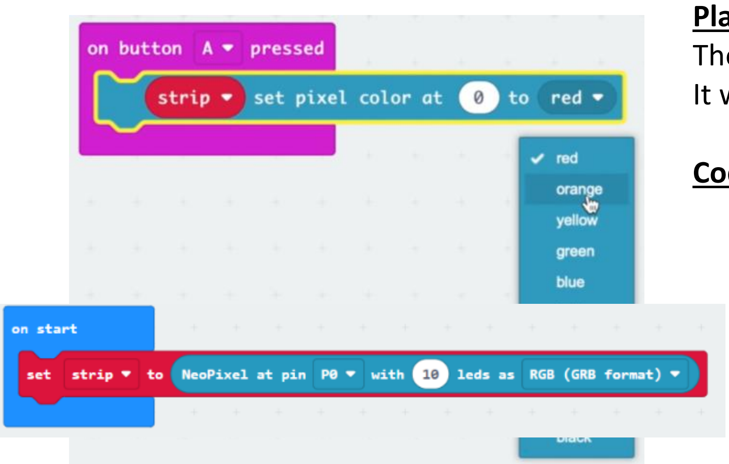


Planning the solution

The image above is the plan for the school beacon. It will be used to guide the coding.

Coding the solution - step one

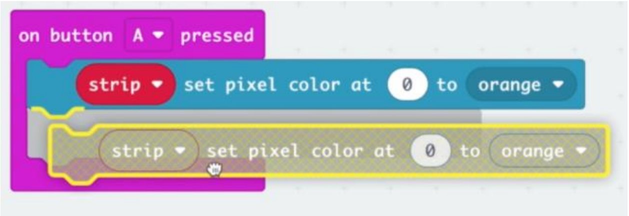
- The set strip to… block has been added to the on start block and updated to 10 LEDs, as in the previous activities.
- In the on button A pressed input, add the set pixel color at___to ___ This allows us to set any individual led to a colour of our choice. – From the plan above, we have set led no:0 to orange.



```
on button A ▾ pressed
    strip ▾ set pixel color at 0 to red ▾
```

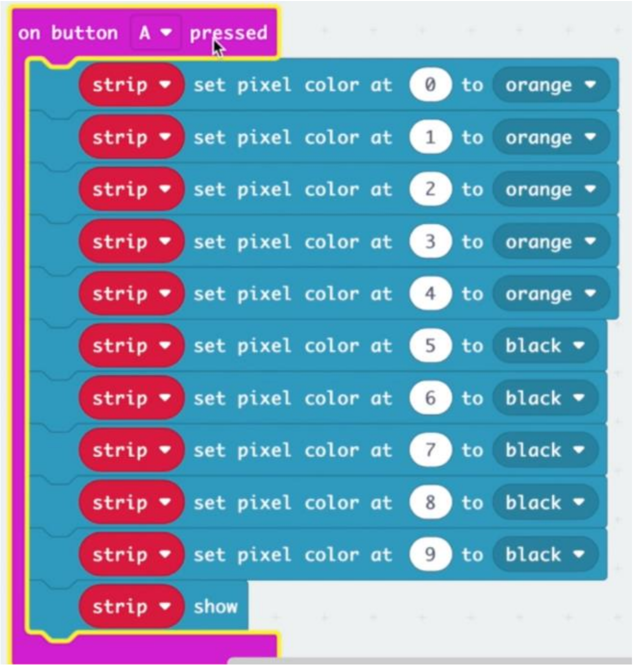
```
on start
    set strip ▾ to NeoPixel at pin P0 ▾ with 10 leds as RGB (GRB format) ▾
```

- The block is added again and this time we set led no:1 to orange.



```
on button A ▾ pressed
    strip ▾ set pixel color at 0 to orange ▾
    strip ▾ set pixel color at 0 to orange ▾
```

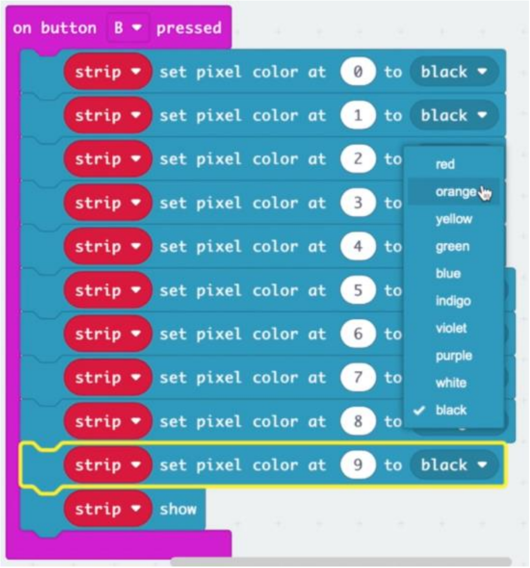
- As seen opposite, we repeat this for all 10 LEDs, assigning the correct number and colour to the LED as we had set out in the plan above. (0,1,2,3,4 are orange. 5,6,7,8,9 are black, i.e. turned off)



```
on button A ▾ pressed
    strip ▾ set pixel color at 0 to orange ▾
    strip ▾ set pixel color at 1 to orange ▾
    strip ▾ set pixel color at 2 to orange ▾
    strip ▾ set pixel color at 3 to orange ▾
    strip ▾ set pixel color at 4 to orange ▾
    strip ▾ set pixel color at 5 to black ▾
    strip ▾ set pixel color at 6 to black ▾
    strip ▾ set pixel color at 7 to black ▾
    strip ▾ set pixel color at 8 to black ▾
    strip ▾ set pixel color at 9 to black ▾
    strip ▾ show
```

- The strip show block is added to turn on the LEDs to the set colour.

- The strip is turned off by adding the strip clear and strip show blocks to the on button B pressed block.

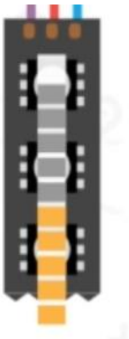


```
on button B ▾ pressed
    strip ▾ set pixel color at 0 to black ▾
    strip ▾ set pixel color at 1 to black ▾
    strip ▾ set pixel color at 2 to
    strip ▾ set pixel color at 3 to
    strip ▾ set pixel color at 4 to
    strip ▾ set pixel color at 5 to
    strip ▾ set pixel color at 6 to
    strip ▾ set pixel color at 7 to
    strip ▾ set pixel color at 8 to
    strip ▾ set pixel color at 9 to black ▾
    strip ▾ show
```

Coding the solution – step two

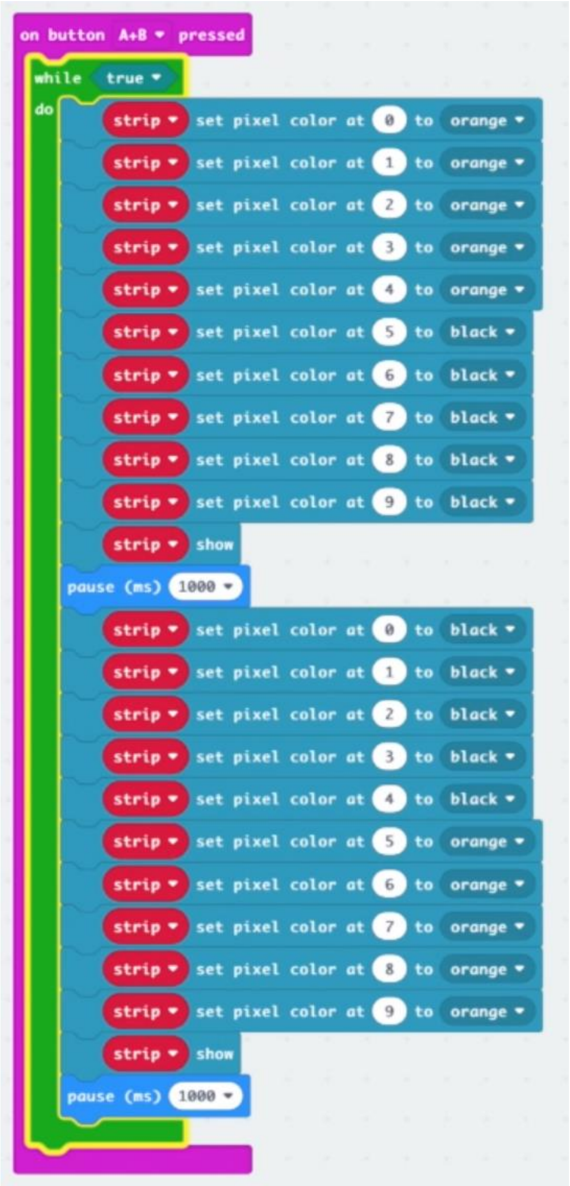
To code the second step from the plan above, we can duplicate the on button A pressed block and change it to an on button B pressed. These save having to re-build all the blocks of code.

- As seen opposite, we update all 10 LEDs, assigning the colour from the plan above. This time LED’s 0,1,2,3,4 are black, while 5,6,7,8,9 are now orange.



- The strip show block is again needed to turn on the LEDs to the set colour.
- By pushing buttons A or B on the micro:bit, we can create the alternating flashing beacon.

Download and test the code on the micro:bit.



```
on button A+B ▾ pressed
    while true ▾
    do
        strip ▾ set pixel color at 0 to orange ▾
        strip ▾ set pixel color at 1 to orange ▾
        strip ▾ set pixel color at 2 to orange ▾
        strip ▾ set pixel color at 3 to orange ▾
        strip ▾ set pixel color at 4 to orange ▾
        strip ▾ set pixel color at 5 to black ▾
        strip ▾ set pixel color at 6 to black ▾
        strip ▾ set pixel color at 7 to black ▾
        strip ▾ set pixel color at 8 to black ▾
        strip ▾ set pixel color at 9 to black ▾
        strip ▾ show
        pause (ms) 1000 ▾
        strip ▾ set pixel color at 0 to black ▾
        strip ▾ set pixel color at 1 to black ▾
        strip ▾ set pixel color at 2 to black ▾
        strip ▾ set pixel color at 3 to black ▾
        strip ▾ set pixel color at 4 to black ▾
        strip ▾ set pixel color at 5 to orange ▾
        strip ▾ set pixel color at 6 to orange ▾
        strip ▾ set pixel color at 7 to orange ▾
        strip ▾ set pixel color at 8 to orange ▾
        strip ▾ set pixel color at 9 to orange ▾
        strip ▾ show
        pause (ms) 1000 ▾
```

Automating the beacon

While the beacon can be manually operated, it is much more practical to do this automatically. This can be easily achieved.

Coding the solution

- Duplicate the on button A pressed block and change it to a new on button A+B pressed block. This block contains the code for the first step of the beacon flash, from button A.
- We now copy the code for the second step of the beacon flash from the on button B pressed block. We add this code to our new on button A+B pressed block.
- We must add a pause after each step to tell the micro:bit how long to leave the LEDs on for.
- Finally, we add a loop to automate the flash sequence by choosing the while true loop and adding it as shown opposite.
- Pressing A+B will now operate the flashing beacon.

Download and test the code on the micro:bit.

Design Challenge:

An ambulance supplier intends to upgrade their fleet to energy saving LED strips. The strips will be located on the front top part of the body, in the same position as the lights shown in the image.

- **Design**, **code**, and **test** a beacon display that will be suitable for the ambulance.
- **Apply** your code to your Neopixel strip.

Neopixel project planning sheet

Use this to help you design and code your Neopixel strip for your projects.
Shade or colour your design to guide your coding.
A sample design for the school beacon is shown.

	0	1	2	3	4	5	6	7	8	9
Light sequence: Step One	●	●	●	●	●	○	○	○	○	○
Light sequence: Step Two	○	○	○	○	○	●	●	●	●	●

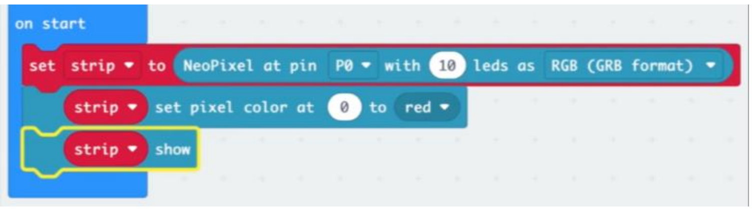
Project name: ______________________________

	0	1	2	3	4	5	6	7	8	9
Light sequence: Step One	○	○	○	○	○	○	○	○	○	○
Light sequence: Step Two	○	○	○	○	○	○	○	○	○	○
Light sequence: Step Three	○	○	○	○	○	○	○	○	○	○
Light sequence: Step Four	○	○	○	○	○	○	○	○	○	○
Light sequence: Step Five	○	○	○	○	○	○	○	○	○	○
Light sequence: Step Six	○	○	○	○	○	○	○	○	○	○
Light sequence: Step Seven	○	○	○	○	○	○	○	○	○	○
Light sequence: Step Eight	○	○	○	○	○	○	○	○	○	○

Activity D: The Larson Scanner (Knight Rider)

Video

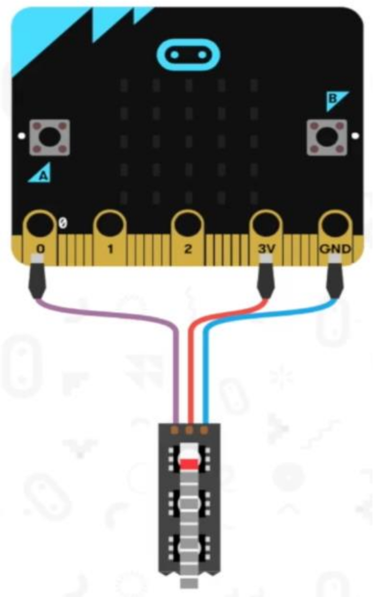
This activity explores more code which can be used to animate sequences on your Neopixel strip. We will investigate a running sequence of back-and-forth light. This is a progression from Activity B and C, which contain prerequisite knowledge. Click or scan the QR code opposite to access the video to guide you through the activity.

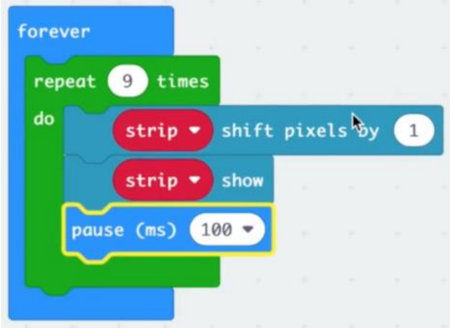


```
on start
    set strip ▾ to NeoPixel at pin P0 ▾ with 10 leds as RGB (GRB format) ▾
    strip ▾ set pixel color at 0 to red ▾
    strip ▾ show
```

Coding the solution - step one

- As shown above, the set strip to… block has been added to the on start block and updated to 10 LEDs, as in the previous activities. As shown above, the set strip to… block has been added to the on start block and updated to 10 LEDs, as in the previous activities.
- Adding the set pixel color at___to ___ has been added and modified to set pixel 0 to red.
- When the strip show block is added, it will turn the first led on the strip to red. This is shown opposite.

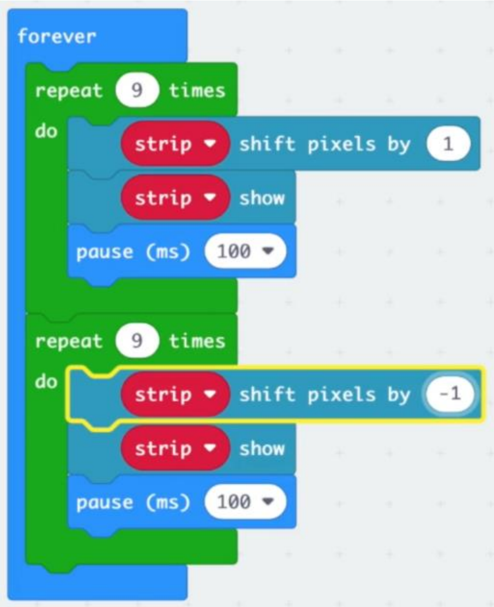




```
forever
    repeat 9 times
    do
        strip ▾ shift pixels by 1
        strip ▾ show
        pause (ms) 100 ▾
```

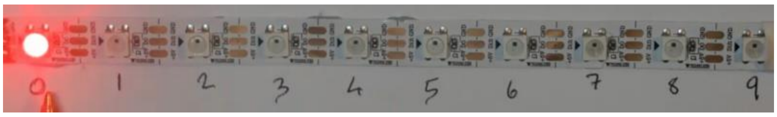
- Returning to code we used in activity B, we add the strip shift pixels block and the strip show block. This pushes the red LED one step up the line.
- We need this step to happen nine times, so we add a loop by adding the repeat __ times block and modifying it to 9. This makes the step repeat nine times, bringing the LED to the end of the strip.
- Adding a pause block slows down the steps so we can see the LED move.

- The LED is currently starting at position 0 and shifting to the end of the strip, where it pushed off and appears to turn off. **We need to now tell it to step backwards from 9**.
- We do this by duplicating the repeat block and all the code inside it. We add it in the forever loop underneath itself as shown opposite.
- To tell the LED to shift in reverse, we modify the shift pixels by ___ code to shift by **-1**. Now our red LED will travel over and back from position 0 to 9 forever.



```
forever
    repeat 9 times
    do
        strip ▾ shift pixels by 1
        strip ▾ show
        pause (ms) 100 ▾
    repeat 9 times
    do
        strip ▾ shift pixels by -1
        strip ▾ show
        pause (ms) 100 ▾
```

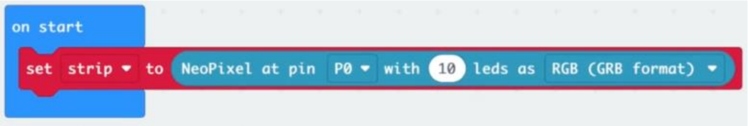
> **Challenge:** How could we modify our code to alter the speed of the scanner? Experiment with making it double, and then half it's speed.



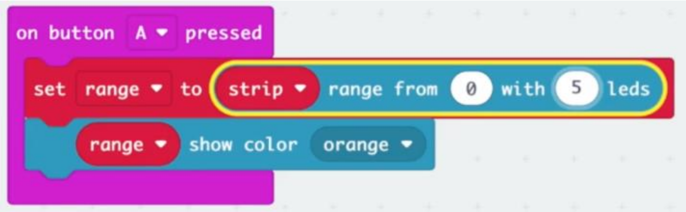
Activity E: Dynamic Indicators (Audi Style)

Video

This is an advanced activity. We explore code which can be used to create dynamic “Audi” indicators. Many of the concepts are established in previous activities. This activity should be attempted after completing these previous activities. This is worked through in detail in the accompanying video. Click or scan the QR code to access the video.

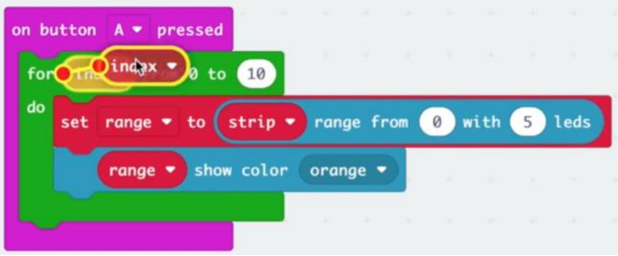


```
on start
    set strip ▾ to NeoPixel at pin P0 ▾ with 10 leds as RGB (GRB format) ▾
```

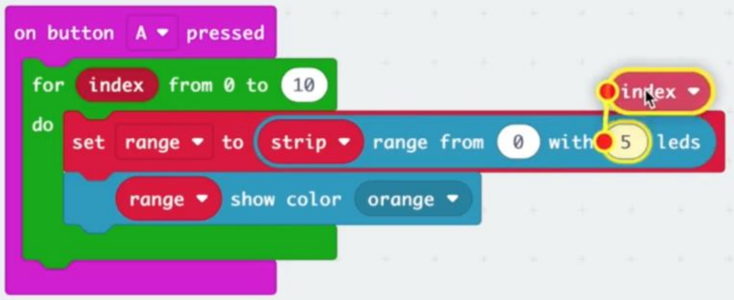


```
on button A ▾ pressed
    set range ▾ to strip ▾ range from 0 with 5 leds
    range ▾ show color orange ▾
```

- We begin this activity as usual, by adding the set strip to code to the on start block and by modifying it to have 10 LEDs.
- Using an on button A pressed block, we now add the set range to___ block. This code sets up a **group** of LEDs called a range. *This is useful to us as we can then vary the size of the group*. The default values are to start the group at LED 0 and to add 5 LEDs to it.
- We next add the strip show colour____ block and modify it using the drop down to read range show colour___. We can now set the colour of the group as we decide to. We choose orange from the drop down.



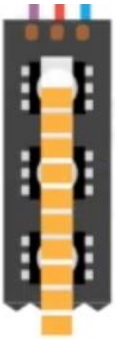
```
on button A ▾ pressed
    for index ▾ 0 to 10
    do
        set range ▾ to strip ▾ range from 0 with 5 leds
        range ▾ show color orange ▾
```

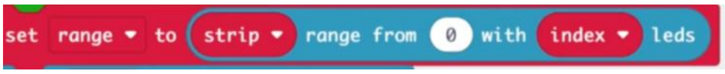


```
on button A ▾ pressed
    for index from 0 to 10                                  index ▾
    do
        set range ▾ to strip ▾ range from 0 with 5 leds
        range ▾ show color orange ▾
```

We have set up a group of LEDs called a range. We have also defined the colour of the group. The key to making the dynamic indicator light function is to add LEDs to the group to make it appear to “grow.” We do this using a loop.

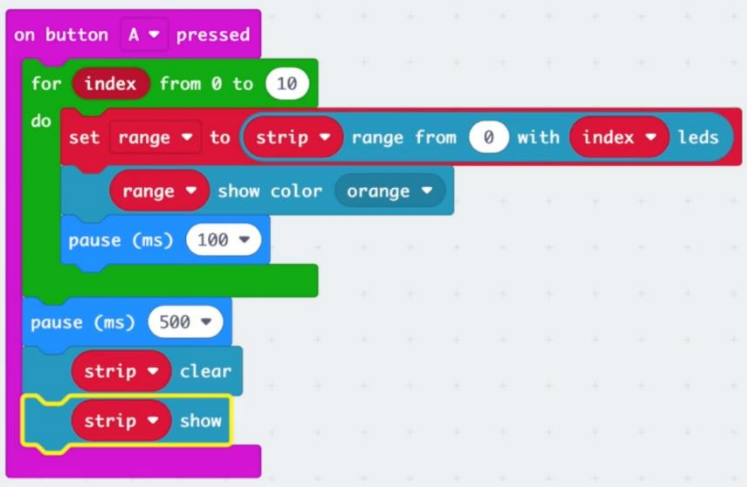
- From the loop drawer, we add the for index from 0 to ___ block. This automatically creates a variable called index. This will be the size of our group. This block counts from 0 to whatever number we decide. We modify it to count from 0 to 10, as we have 10 LEDs on our strip.
- The important step in this process is to now tell the micro:bit that the size of the group should be whatever number the for index from loop has currently counted to. It will vary between 0 to 10, so the group will also be between 0 and 10.
- To do this, click on the index variable in the loop itself as shown above and drag it into the last variable opening on the set range to___ block, as shown above on the right.
- The block should look like the one shown below, and the simulator should look like the image shown on the right.





```
set range ▾ to strip ▾ range from 0 with index ▾ leds
```

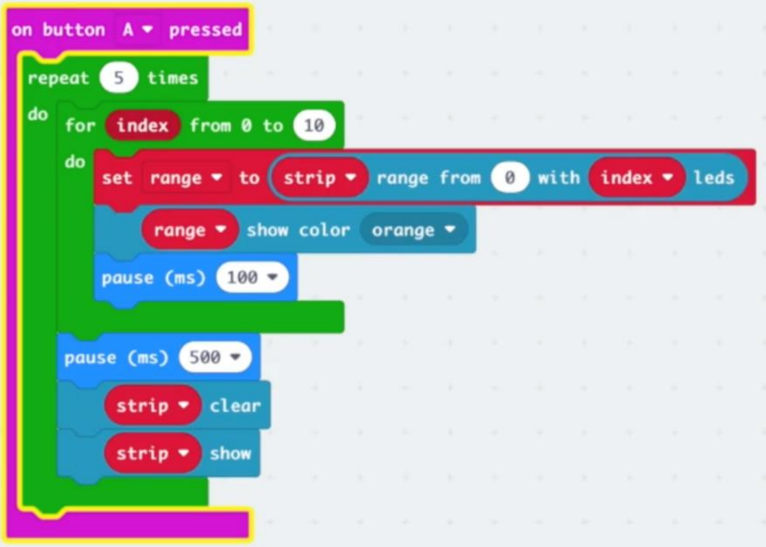
All the LEDs on the strip appear currently lit up orange, as the orange range has increased from 0 LEDs to 10 LEDs so fast that we couldn't see it. To get the effect of the growing indicator, we need to slow the "range growing" process down.



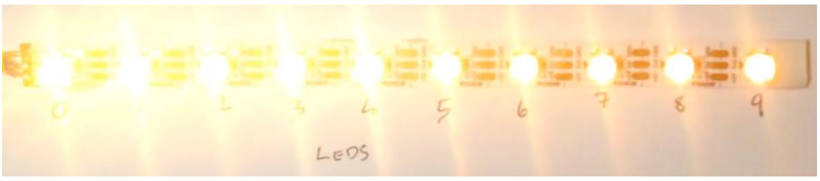
```
on button A pressed
    for index from 0 to 10
    do
        set range to strip range from 0 with index leds
        range show color orange
        pause (ms) 100
    pause (ms) 500
    strip clear
    strip show
```

- Add a pause block into the for index from___ loop. Place it under the range show colour block as shown on the left. We can leave the duration at the default 100ms. This tells the micro:bit to pause for 100ms each time it adds another LED to the group. We should now be able to see the group growing.
- To clear the line and start again, we add a pause block after the for index from___ loop. Choose 500ms for a nice effect.
- After the line has been fully lit for 500ms, we turn off all the LEDs by adding the strip clear and strip show blocks at the end.

- Now, we have the correct animation for our Audi style dynamic indicators working correctly. Let's choose the number of times this sequence should happen once we press the input button A.
- To repeat the animation, we add a repeat___ times loop into the on button A pressed block, ensuring all the other code snaps inside the repeat loop as we drag it in. This should happen easily if you drag the repeat ___times loop in from outside the left edge of the on button A pressed block.
- Modify your code to give a suitable number of flashes. In the code shown on the right, a repeat of 5 times is shown.



```
on button A pressed
    repeat 5 times
    do
        for index from 0 to 10
        do
            set range to strip range from 0 with index leds
            range show color orange
            pause (ms) 100
        pause (ms) 500
        strip clear
        strip show
```



The Neopixel strip displaying the code in action.

Further explorations…
Look back at the way we have assembled the code. Try to experiment with different values to speed up, slow down, or shorten sequences.
If we change the number in the for index from loop to a number below 10 what happens?

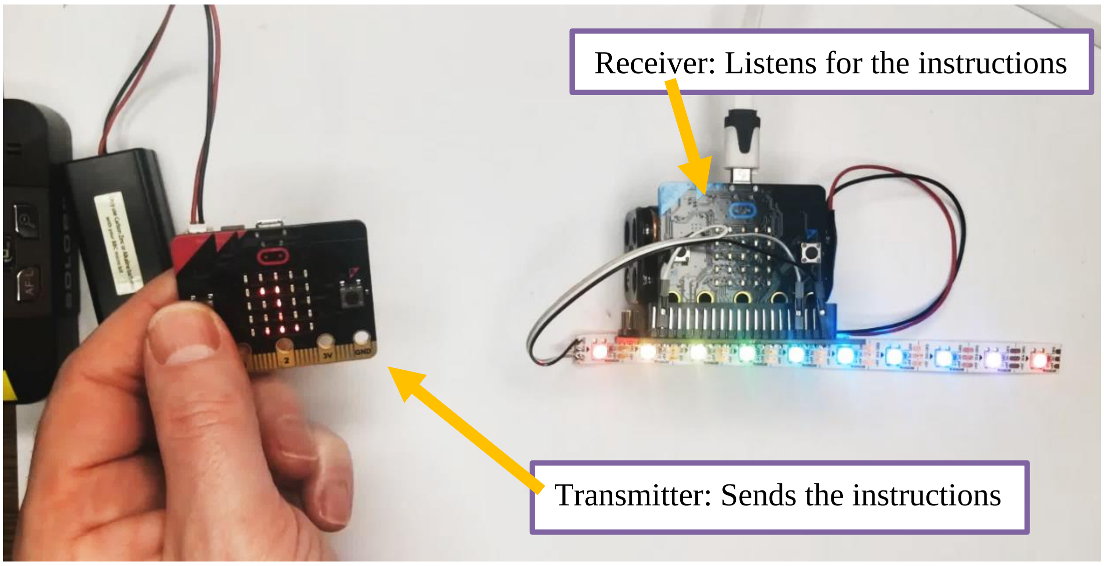
Module 12: Radio Control 2

Video

Activity A: Remote Neopixel Control

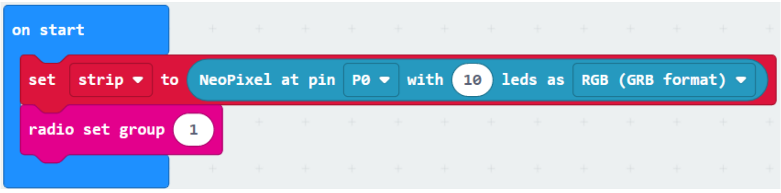
As we've seen in the previous module the Neopixels can be controlled and lit up individually. Using the micro:bit bluetooth feature, we can set up a remote display that can be controlled from a distance using a second micro:bit. The following activity will show you how to create the command and control the Neopixels.
Scan or click the QR code to use the Video to support you in the activity.
Note: Before commencing this activity, ensure you have connected the Neopixel strip to your Micro:bit as described at the beginning of Module 11.

** The code below for this activity can be completed as one program, and downloaded to two separate micro:bits. Alternatively, the code for the transmitter can be programmed separately from the code for the receiver and downloaded separately to the appropriate micro:bit. The video programs the code for the transmitter and receiver separately, and downloads the correct code to the appropriate micro:bit.


Receiver: Listens for the instructions
Transmitter: Sends the instructions

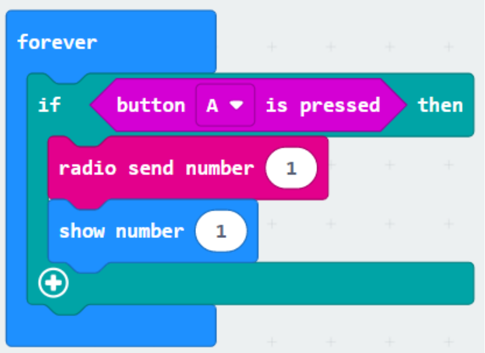
The following code seen above will solve the activity. The program is developed using the 'input', 'basic' and 'radio', and the Neopixels extension. Using this code, you will be able to remotely turn the Neopixel strip on, off, and rotate the lights upon pressing the button on a micro:bit that is part of the same radio group. We will do this by sending one of three numbers: **0**= Off, **1**=On, **2**=Rotate pixels.

1:



```
on start
    set strip ▾ to NeoPixel at pin P0 ▾ with 10 leds as RGB (GRB format) ▾
    radio set group 1
```

- As shown in the previous module, we set up the strip of Neopixels using the set strip to... block at the top of the Neopixel drawer.
- Drag this into the on start block.
- Update the set strip to... block to contain **10** LEDs.
- From the Radio drawer, select the radio set group__ block and add it as shown. The number shown in this block is the channel we are using. Micro:bits using the same channel can "talk" to each other.



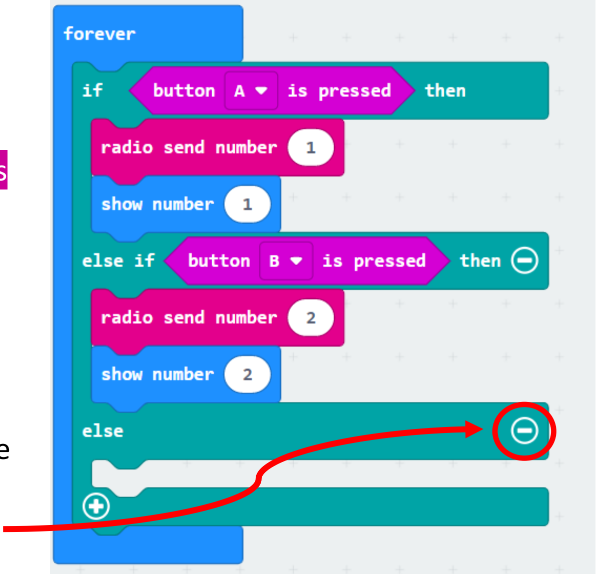
```
forever
  if button A is pressed then
    radio send number 1
    show number 1
```

2:

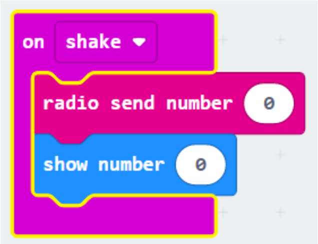
- In Module 7, in the 2020/2021 CPD session, we showed how to create a signal, and a number to be sent to the other Micro:bit.
- In the forever block, we add a conditional if ___ then ___. This can be found in the Logic drawer.
- Using the 'Input' tab we choose the if button A is pressed and place it in the conditional as shown.
- From the Radio drawer, we add the radio send number__ and modify it to read **1**. For visual feedback, add the show number___ block from the Basic drawer and modify it to read the number **1**.

3:

- We now create the next branch of the conditional. We add in the if button A is pressed and modify it to read if button B is pressed.
- Similar to step two, we now add the radio send number___ and show number ___ blocks.
- Modify both blocks to read the number **2**, as shown opposite.
- We can delete the last "*else*" branch of the code as we won't need it. Do this by clicking the minus on the same branch as the *else*.



```
forever
  if button A is pressed then
    radio send number 1
    show number 1
  else if button B is pressed then ⊖
    radio send number 2
    show number 2
  else ⊖
```



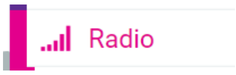
```
on shake
  radio send number 0
  show number 0
```

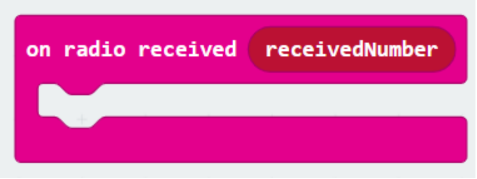
4:

- Now that we have set up the commands to send the *on* and *rotate* numbers, we can now add in the *off* command. This will be sent when the micro:bit is shaken.
- In the Input drawer, select the on shake block and drag it into the editor window.
- As in step two above, add in the radio send number___ and the show number___ blocks. Modify these to the number **0**, which will be the *off* command.
- This concludes the transmitter part of the coding. We will now investigate the receiver part of the coding process.

5:

- Now that we have set up the code for the micro:bit to transmit the numbers 0 for off, 1 for on, and 2 for rotate, we now tell the micro:bit what to do when it receives or "hears" these numbers.
- We begin this process by opening the Radio drawer and selecting the on radio received 'received number' block and dragging it into the editor window.
- We will use the if___ else___ conditional from the Logic drawer to code this part.

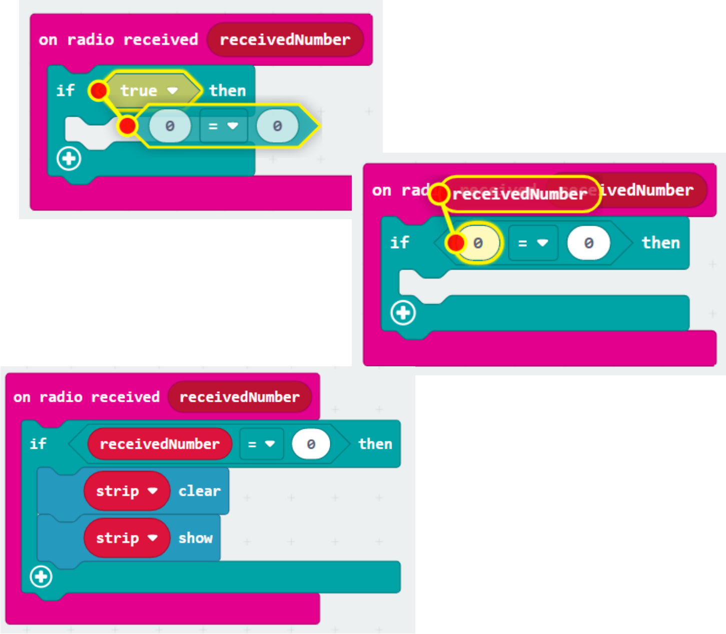




```
on radio received receivedNumber
```

6:

- When the if___ else___ block has been added, we add the comparison ___=___ also from the Logic drawer and place it in the if___else___ block as shown.
- To tell the micro:bit to compare the received number to something, select the 'received number' variable within the block and drag it into the first part of the ___=___ comparison block as shown on the left.
- As we want the number 0 to turn off the strip, under the comparison that the *if* received number = 0, add in the strip clear and strip show blocks from the Neopixel drawer.



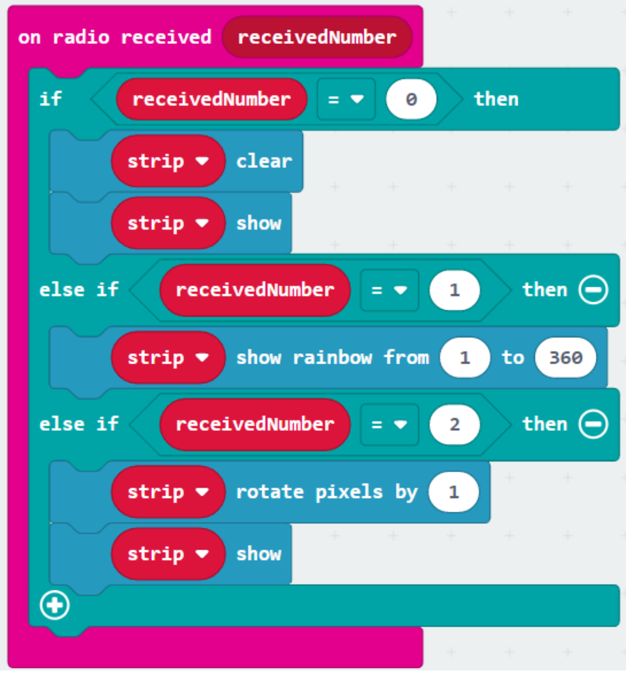
```
on radio received receivedNumber
    if true then
    0 = 0
```

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

```
on radio received receivedNumber
    if receivedNumber = 0 then
        strip clear
        strip show
```

7:

- Similarly, we now add in the extra *else if* branches to the if___ else___ code so that *if* received number = 1, turn on the strip by adding the strip show rainbow code underneath.
- Adding another *else if* branch. Edit this code to read, *if* received number = 2 then rotate the pixels by adding the strip rotate pixels and strip show codes from the Neopixels drawer.
- Download the code to two micro:bits and test it.
- This code can be used in either the transmitting or receiving Micro:bit.



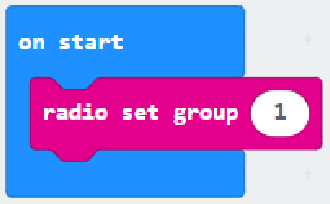
```
on radio received receivedNumber
    if receivedNumber = 0 then
        strip clear
        strip show
    else if receivedNumber = 1 then
        strip show rainbow from 1 to 360
    else if receivedNumber = 2 then
        strip rotate pixels by 1
        strip show
```

Activity B: Pitch (Tilt) Wireless Motor Control of a DC Motor

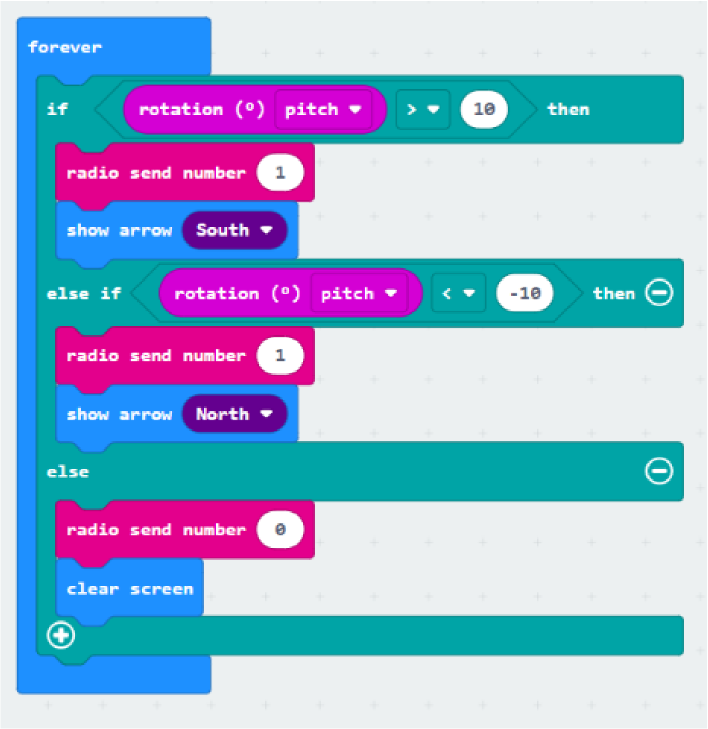
The following code will allow the motor to be controlled by tilting the micro:bit forward and back. The control of the motor will be dependent on the information sent to it by the rotation as seen on the code below. *Scan or click the QR code to use the video to support you in the activity.*

Video

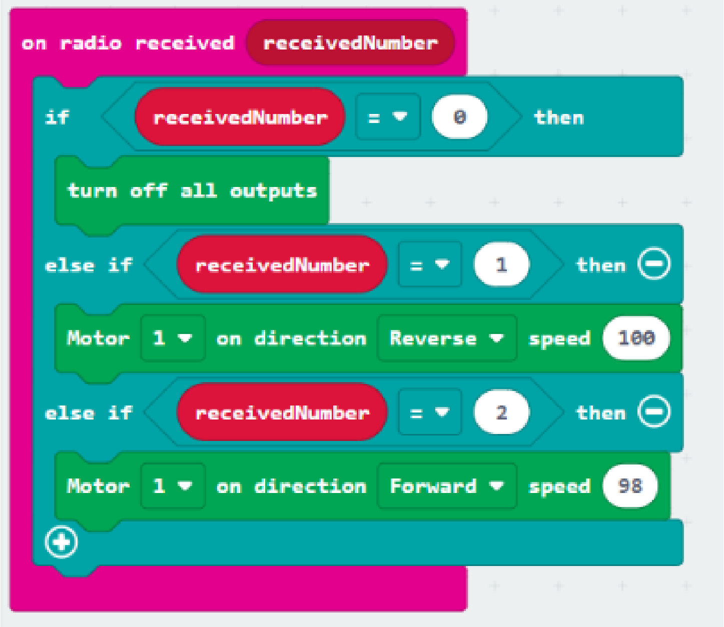
** The following code can be completed in one go and downloaded to two separate micro:bits as shown below. Alternatively, it can be split in two and programmed to two separate micro:bits as shown in the video. In that instance, the radio received blocks are programmed into the micro:bit that's added to the product.



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



```
forever
    if rotation (°) pitch ▾ > ▾ 10 then
        radio send number 1
        show arrow South ▾
    else if rotation (°) pitch ▾ < ▾ -10 then ⊖
        radio send number 1
        show arrow North ▾
    else ⊖
        radio send number 0
        clear screen
    ⊕
```



```
on radio received receivedNumber
    if receivedNumber = ▾ 0 then
        turn off all outputs
    else if receivedNumber = ▾ 1 then ⊖
        Motor 1 ▾ on direction Reverse ▾ speed 100
    else if receivedNumber = ▾ 2 then ⊖
        Motor 1 ▾ on direction Forward ▾ speed 98
    ⊕
```

This code is the solution to the challenge:

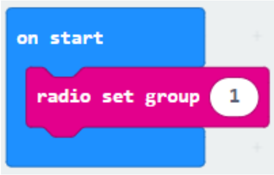
Basic: allows our calculations to run in the background forever or to create a set up from the start of the program

Variables: allow us to map values and use them at different points in the code

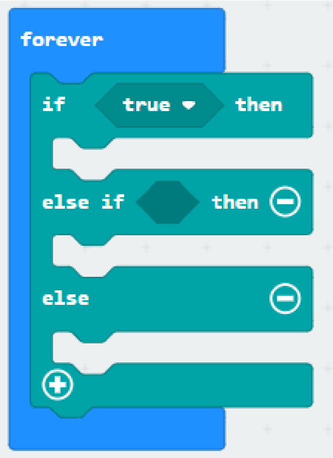
Inputs: allows control with the accelerometer

Logic: allows us to set scenarios in place

Radio: sends a signal to the other micro:bit in the group. The radio tab is also used to tell the motors what to do when the signal is received



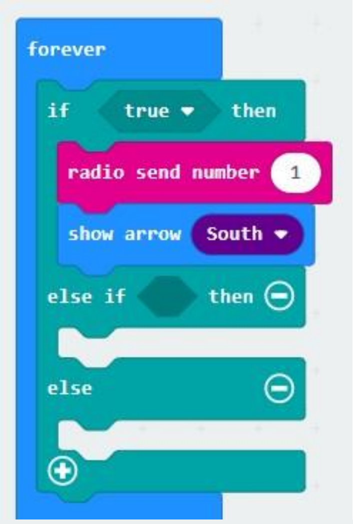
```
on start
    radio set group 1
```



```
forever
    if true then
    else if ⬡ then ⊖
    else ⊖
    ⊕
```

1:

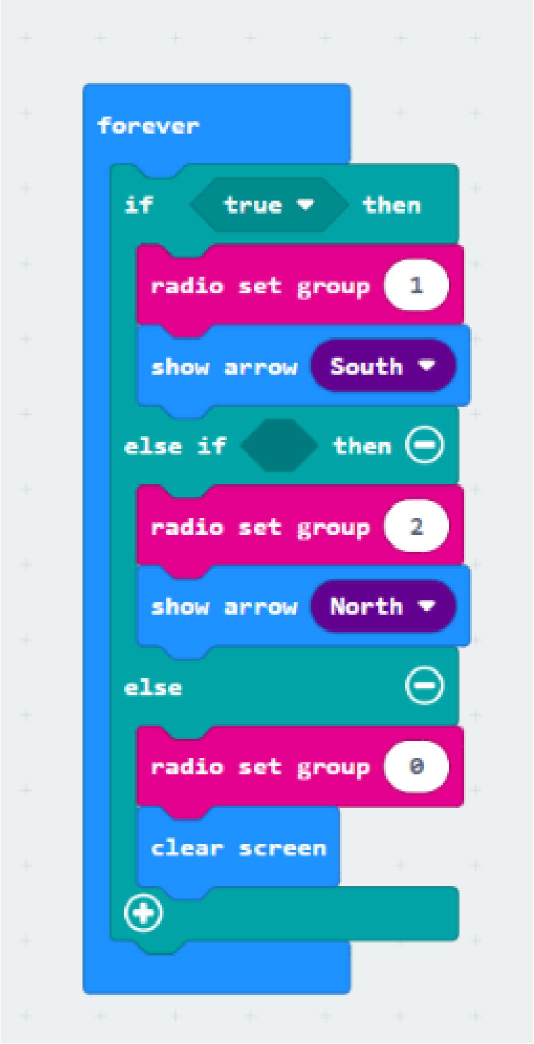
- Using the ‘on-start’ block set the radio group to any value you wish. The ‘radio’ tab has the block needed to do this.
- Using the ‘forever’ block and the ‘logic’ tab we create the blocks shown on the left. Module 8 Activity 4 shows how to add ‘else if__ then else’ by clicking the + icon on the bottom left corner.



```
forever
    if true then
        radio send number 1
        show arrow South
    else if ⬡ then ⊖
    else ⊖
    ⊕
```

2:

- Using the ‘basic’ tab, add in the ‘show arrow South’ as shown on the left. This indicates the intended direction of the motor when moving.
- When directing the motor, a signal needs to be sent to the other micro:bit to tell it to start the motor.
- Using the ‘radio’ tab ‘radio send number __’, set the assign a number to this function.

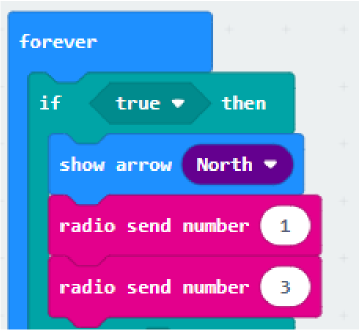


```
forever
    if true then
        radio set group 1
        show arrow South
    else if ⬡ then ⊖
        radio set group 2
        show arrow North
    else ⊖
        radio set group 0
        clear screen
    ⊕
```

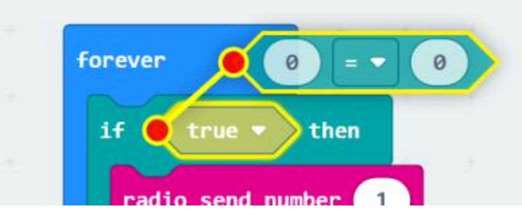
3:

- Using the same blocks as the previous step, change the values as seen.
- By doing this, the direction is being changed on screen to reverse, and a different signal is being sent to the micro:bit.
- Using ‘basic’ and the ‘clear screen’ block, the value now being sent to the other micro:bit turns to 0.
- This indicates that the motor should turn off, but this will only happen when the code is interpreted.

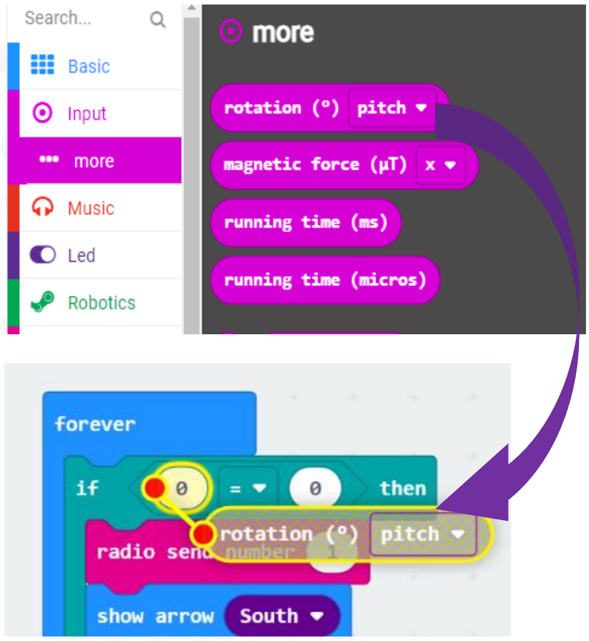
**if 2 or more motors wish to be ran at the same time, then sending many different values can allow this to happen as shown below.



```
forever
    if true then
        show arrow North
        radio send number 1
        radio send number 3
```



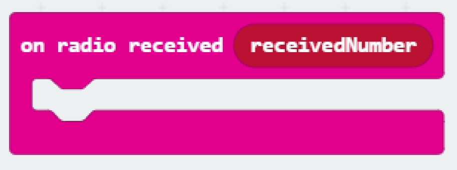
```
forever
  if true then
    radio send number 1
0 = 0
```



```
forever
  if 0 = 0 then
    radio send number 1
    show arrow South
rotation (°) pitch
```

4:

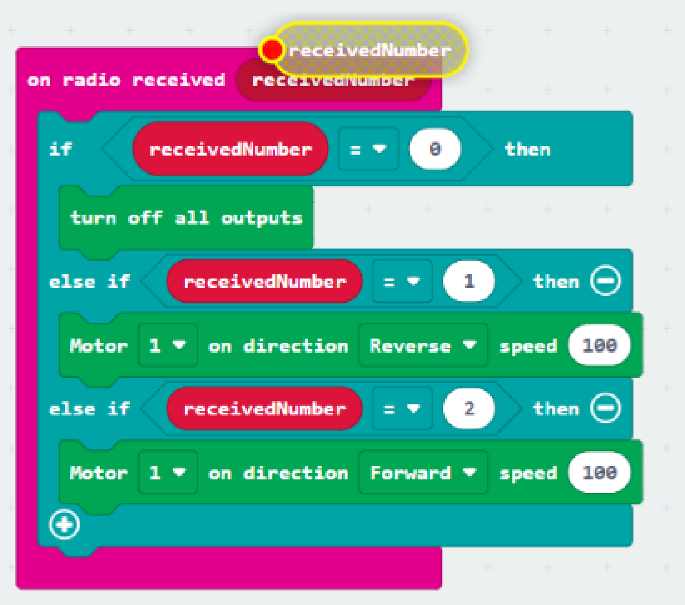
- As previously shown, set up a comparison block in both of the spaces allowed in the logic gate, use the 'less than' and 'greater than' comparison
- Using the 'more' option of the 'input' tab, use the block 'rotation (°) pitch'
- Add the 'rotation (°) pitch' block to the comparison block as shown to the left
- After this, the degrees are set to the desired angle of rotation, in this instance -10° and 10°
- According to the code, now there is a range of 20° from -10° below the micro:bit being horizontal, to 10° above the micro:bit being horizontal, where the motor will be turned off



```
on radio received receivedNumber
```

5:

- Using the 'Radio' tab, use the 'on radio received receivedNumber'. The scenarios can be set for the numbers that are sent to do something.



```
on radio received receivedNumber
  if receivedNumber = 0 then
    turn off all outputs
  else if receivedNumber = 1 then
    Motor 1 on direction Reverse speed 100
  else if receivedNumber = 2 then
    Motor 1 on direction Forward speed 100
receivedNumber
```

6:

- Using the 'receivedNumber' block as shown, add it to the comparisons created as shown on the left
- The comparisons are now '=' and the numbers added correspond to the numbers being sent from step 3
- Using the 'robotics' tab, add in the motor control block. Assign the direction and speed as desired
- Using the 'if' blocks and as shown, set the premise that if the number sent is the value 1, it will turn reverse, coinciding with the south arrow
- If the value is 2, it will go forwards, coinciding with the north arrow. If the value is 0, it will stop the motors
- Depending on the angle the micro:bit is tilted at, a number will be transmitted as per the code
- Download and test the code

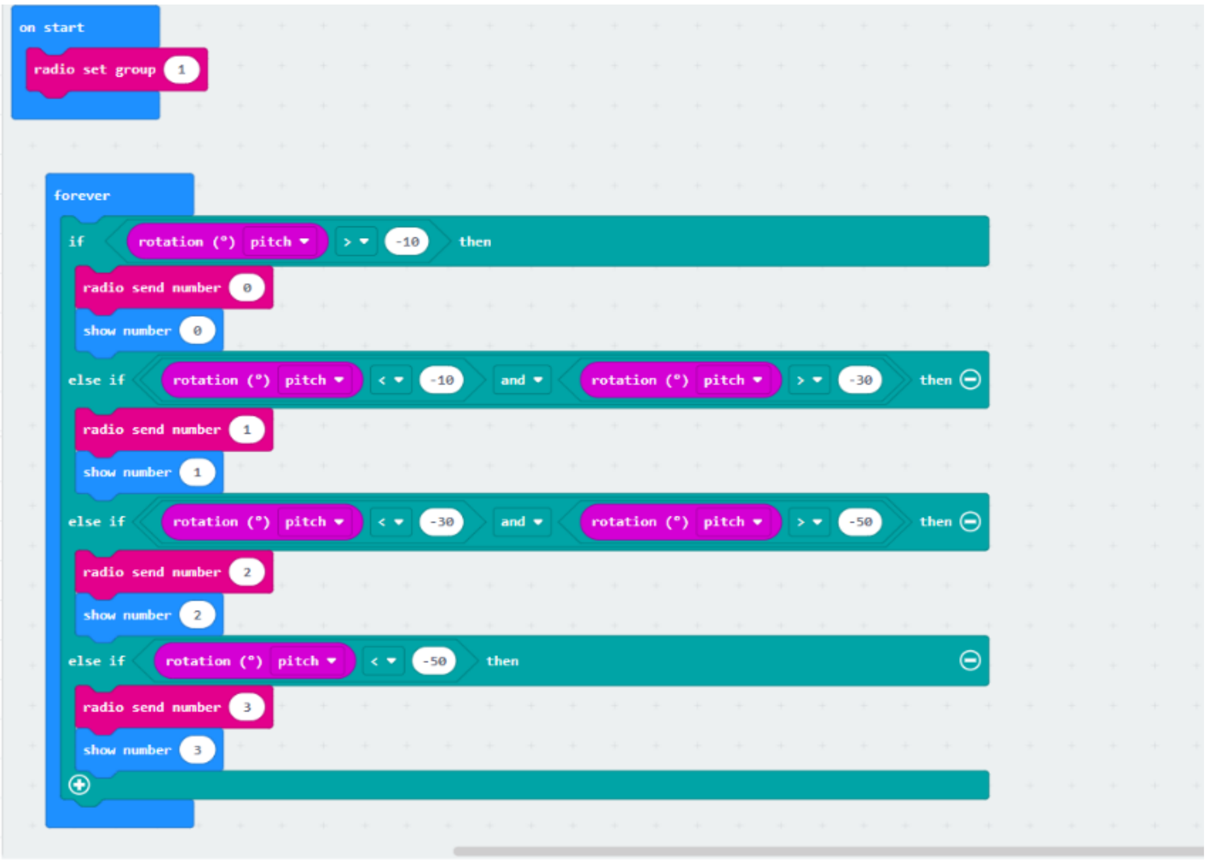
Note: If the code for the Transmitter and the Receiver are in separate programs, the code for the transmitter should be downloaded to the micro:bit that will be the remote control. Download the receiver code to the micro:bit to be fitted to the All-In-One Robotics board.
This is demonstrated in the video accessible through the QR code.

Activity C: Pitch (Tilt) Shifting Gearbox for a DC Motor

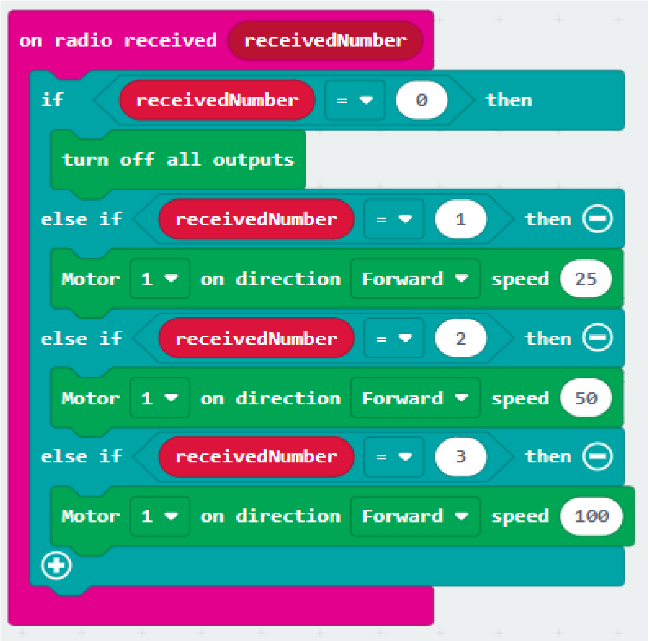
Video

In this activity, the code is developed to speed up the motor or slow it down, dependent on the angle of rotation of the micro:bit. The code builds further on the learning of the previous module and some of the steps are similar to the content already engaged with. Scan or click the QR code to use the video to support you in the activity.

** The following code can be completed in one go and downloaded to two separate micro:bits as shown below. Alternatively, it can be split in two and programmed to two separate micro:bits as shown in the video. In that instance, the radio received blocks are programmed into the micro:bit that's added to the product.



```
on start
    radio set group 1

forever
    if rotation (°) pitch > -10 then
        radio send number 0
        show number 0
    else if rotation (°) pitch < -10 and rotation (°) pitch > -30 then
        radio send number 1
        show number 1
    else if rotation (°) pitch < -30 and rotation (°) pitch > -50 then
        radio send number 2
        show number 2
    else if rotation (°) pitch < -50 then
        radio send number 3
        show number 3
```



```
on radio received receivedNumber
    if receivedNumber = 0 then
        turn off all outputs
    else if receivedNumber = 1 then
        Motor 1 on direction Forward speed 25
    else if receivedNumber = 2 then
        Motor 1 on direction Forward speed 50
    else if receivedNumber = 3 then
        Motor 1 on direction Forward speed 100
```

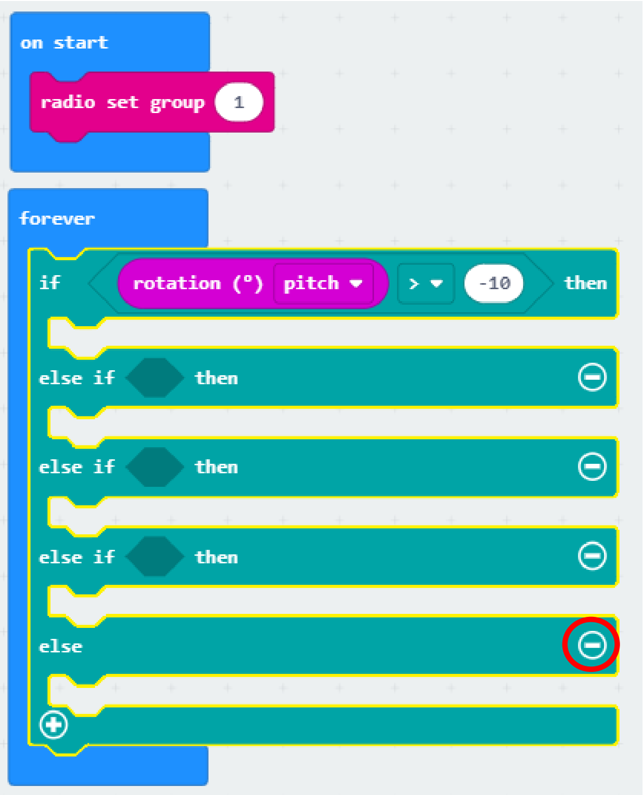
This code is the solution to the challenge:

Basic: allows our calculations to run in the background forever

Inputs: allows control from different sources

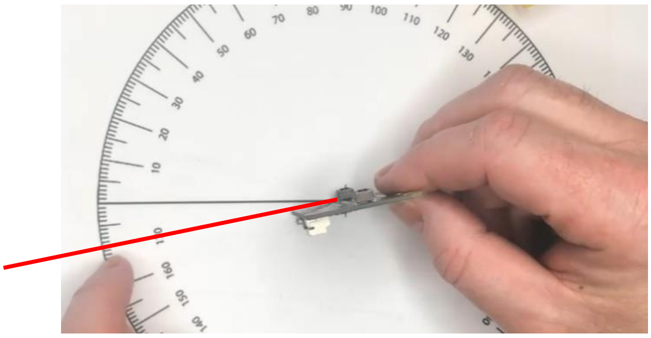
Logic: allows us to set scenarios in place

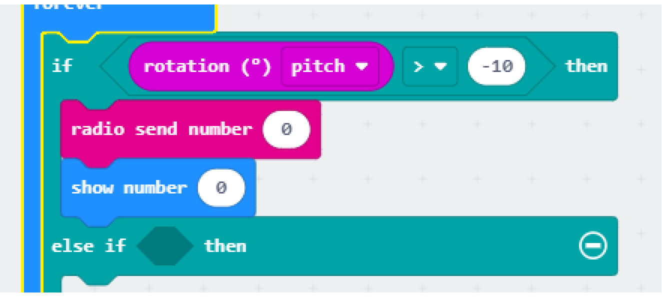
1:



```
on start
    radio set group 1
forever
    if rotation (°) pitch > -10 then
    else if then
    else if then
    else if then
    else
```

- Using the 'on start' block from the 'basic', we now set the micro:bits to talk to each other using the radio set group __. As shown at the start of the module, any micro:bit programmed into this group will communicate together.
- Using the 'logic' tab, set up the comparisons with ' if ' and ' else if ' as shown. These comparisons will set up the premise that sends values based on angles of rotation.
- By using the minus, the else of the logic code can be removed as the solution to the code on the previous page does not need it.
- As shown in step 4 from the previous activity, in the 'more' section of the 'input' tab, create a comparison with the 'logic' tab as shown.
- Set the value to '> -10'. The micro:bit will now be set to work from -10° upwards.

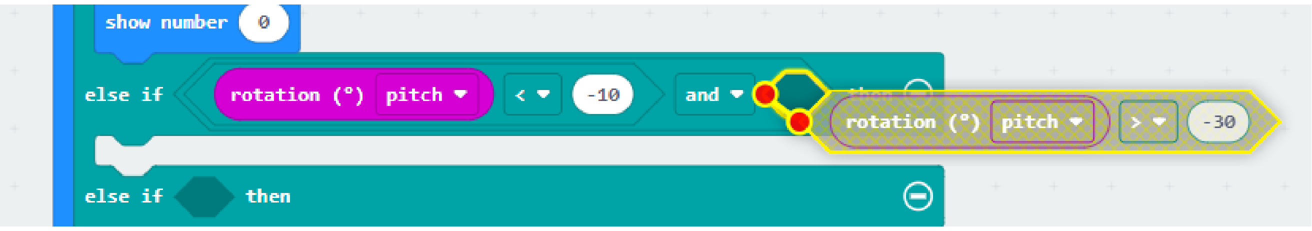




```
forever
    if rotation (°) pitch > -10 then
        radio send number 0
        show number 0
    else if then
```

2:

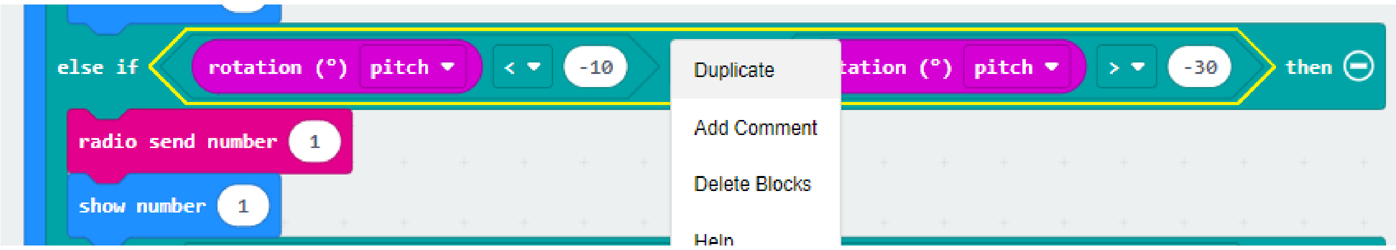
- Using 'radio', create the scenario where the 'radio send number 0' block, and the and 'show number 0' block set the premise that the micro:bit will send the number 0 and show the 0 on screen.



```
show number 0
else if rotation (°) pitch < -10 and   rotation (°) pitch > -30
else if then
```

3:

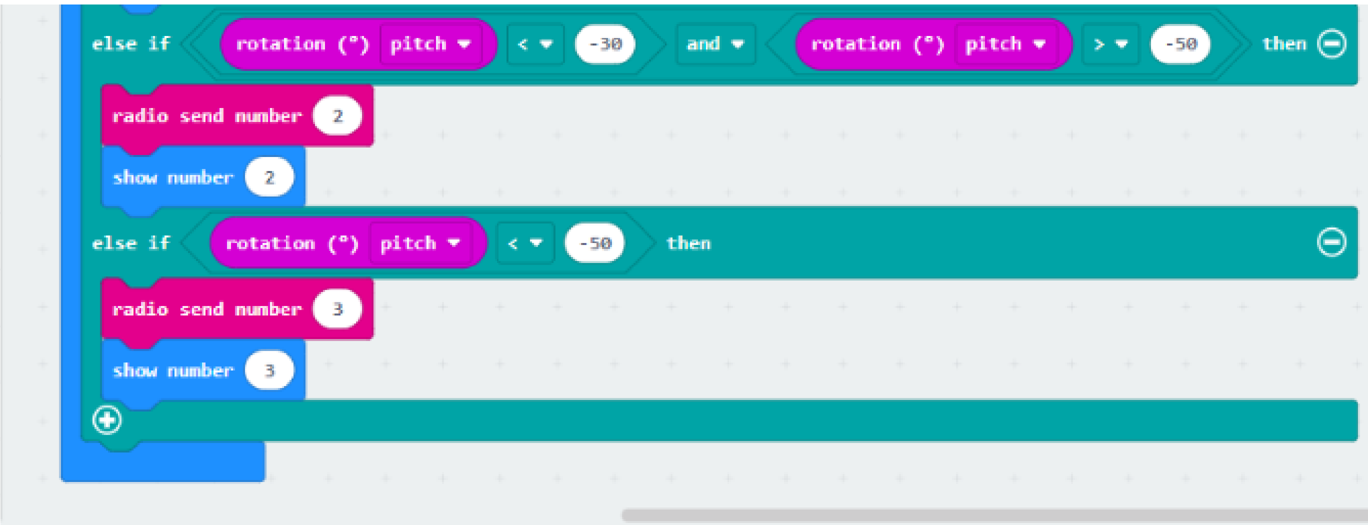
- Using the 'logic' tab, create a comparison with the 'and' block. Add it to the 'else if' section of the logic block
- Using the 'comparison' block again, and the 'rotation (°) pitch' from here, an inequality range can be setup. We can set the pitch between -10° 'and' -30°. From this, the rotation of the micro:bit will make a change from -10° upwards and then from < -10° to >-30°
- Add a number to send and show this number on screen as seen in the previous step



```
else if rotation (°) pitch < -10   ation (°) pitch > -30 then
  radio send number 1
  show number 1
```

4:

- By right clicking on the 'comparison' block, duplicate this inequality and add it to the next 'else if'
- Change the values of the rotation to -30° and -50°
- Using the 'radio send number __' and 'show number__', create a new number to send and show on screen

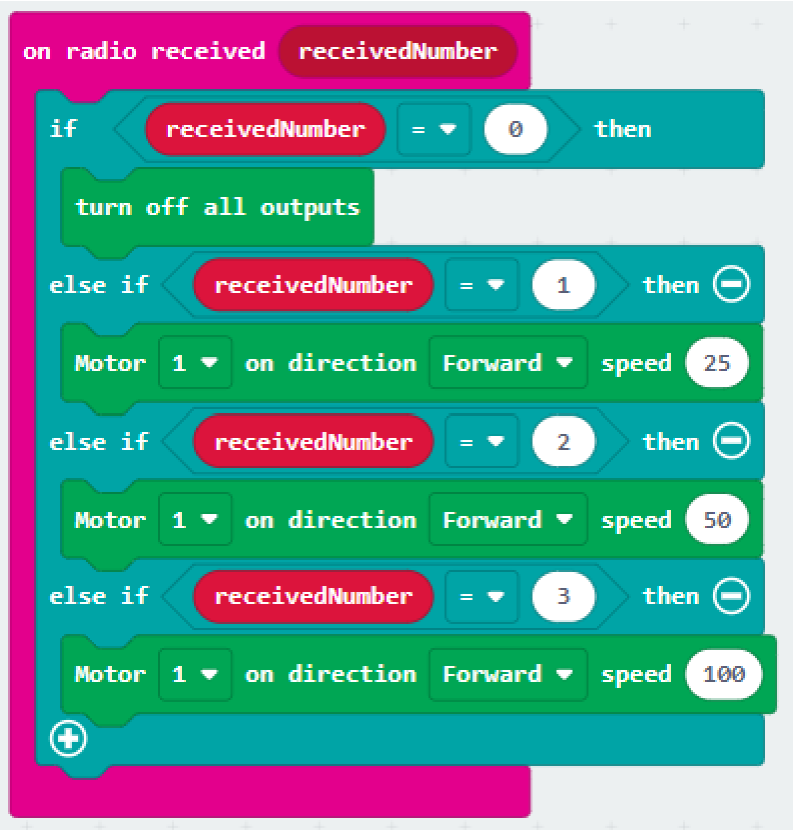


```
else if rotation (°) pitch < -30 and rotation (°) pitch > -50 then
  radio send number 2
  show number 2
else if rotation (°) pitch < -50 then
  radio send number 3
  show number 3
```

5:

- The last part of the code is to set the value for the top speed. Using one comparison, and stating that if the value of rotation is <-50° to send a final number, in this case 3

**to make the motor speed more sensitive, make the range of values smaller; instead of 20° choose another range. And if you wish to add extra speeds, add more 'else ifs' and 'radio send numbers'



```
on radio received receivedNumber
    if receivedNumber = 0 then
        turn off all outputs
    else if receivedNumber = 1 then
        Motor 1 on direction Forward speed 25
    else if receivedNumber = 2 then
        Motor 1 on direction Forward speed 50
    else if receivedNumber = 3 then
        Motor 1 on direction Forward speed 100
```

6:

- The final step of the code is the same as step 6 in activity 2 in this module. As you can see above, the code is built on the numbers sent from the previous steps.
- If the numbers match what has been sent you can see how the motor speeds up.
- ***Download and test the code on the micro:bit.***

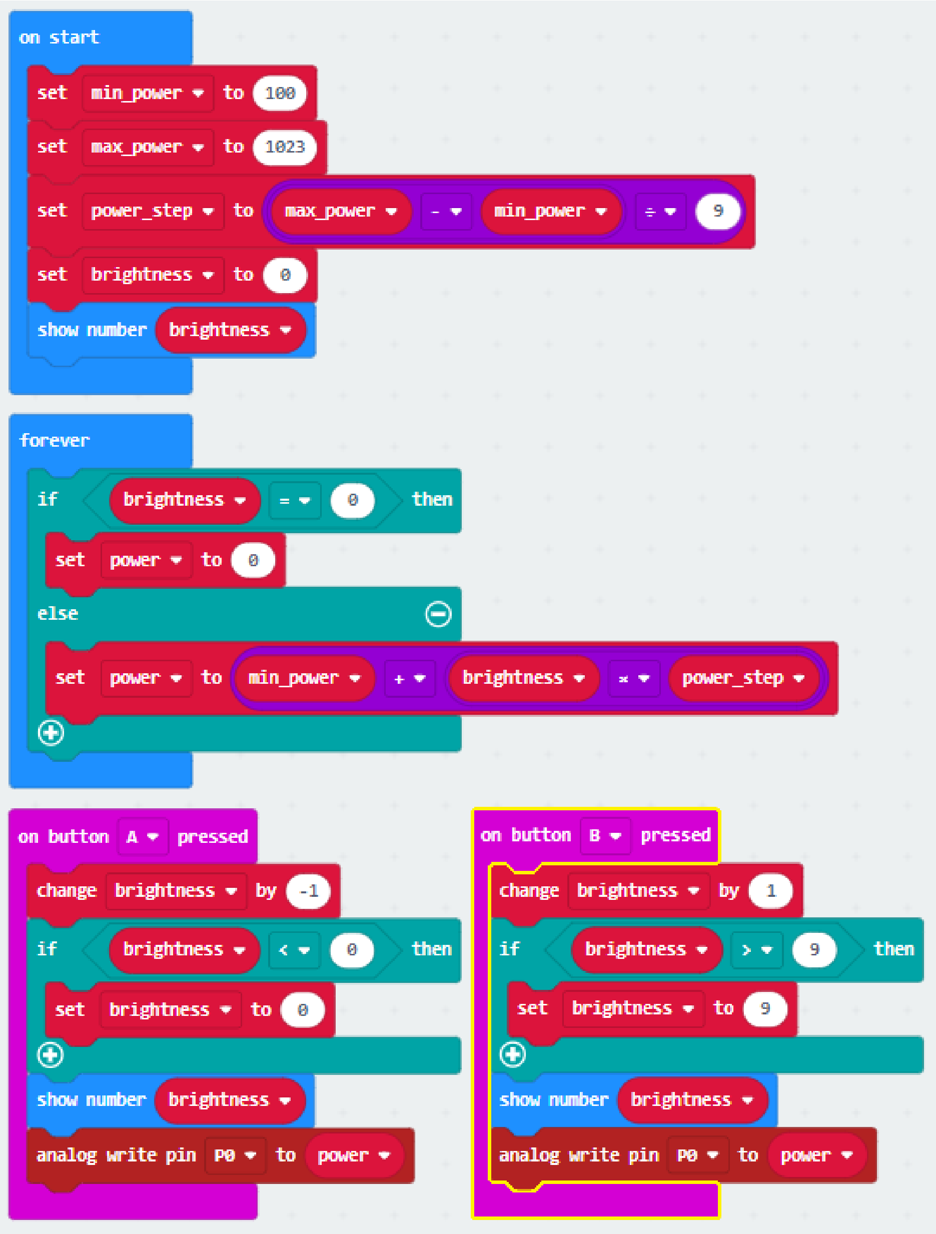
Note: If the code for the Transmitter and the Receiver are in separate programs, the code for the transmitter should be downloaded to the micro:bit that will be the remote control. Download the receiver code to the micro:bit to be fitted to the All-in-one Robotics board.
This is demonstrated in the video accessible through the QR code.

Module 13: Dimming an LED using Pulse Width Modulation

Video

In this Module, we will learn how to create code that will dim an LED. It is done by creating code with blocks that have not been used before. We create math problems that allow for the power going to the LED to be controlled and therefore making the LED shine brighter or dimmer.

The new learning in this module is the development of a math problem and creating a premise whereby dividing the output reading from the micro:bit can create steppingstones to map values to. Scan or click the QR code to use the Video to support you in the activity.



```
on start
    set min_power to 100
    set max_power to 1023
    set power_step to max_power - min_power ÷ 9
    set brightness to 0
    show number brightness

forever
    if brightness = 0 then
        set power to 0
    else
        set power to min_power + brightness × power_step

on button A pressed
    change brightness by -1
    if brightness < 0 then
        set brightness to 0
    show number brightness
    analog write pin P0 to power

on button B pressed
    change brightness by 1
    if brightness > 9 then
        set brightness to 9
    show number brightness
    analog write pin P0 to power
```

This code is the solution to the challenge:

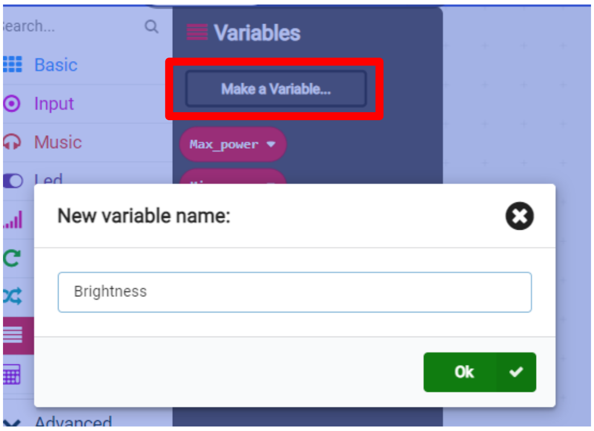
Basic: allows our calculations to run in the background forever

Variables: allow us to map values and use them at different points in the code

Inputs: allows control from different sources

Logic: allows us to set scenarios in place

Pins: Allows for the sending of the signal to the LED and turn it on with a certain power output



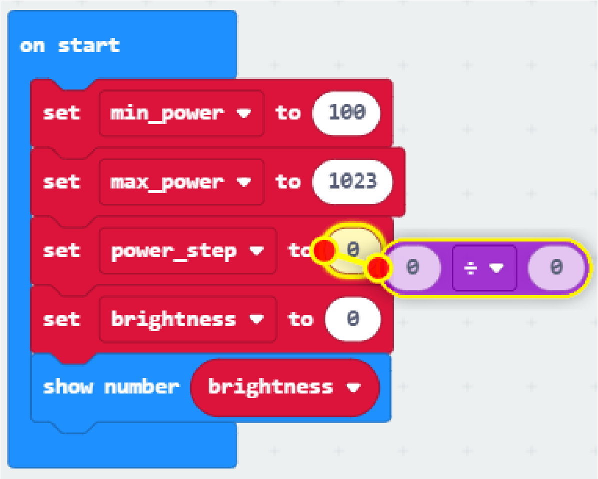
New variable name:

Brightness

1:

- Using the 'variables' tab and the 'make a variable' option, create five different variables:

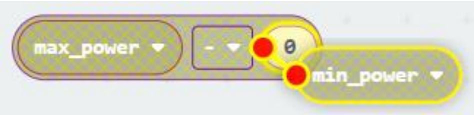
1. Min_power
2. Max_power
3. Power_step
4. Brightness
5. Power



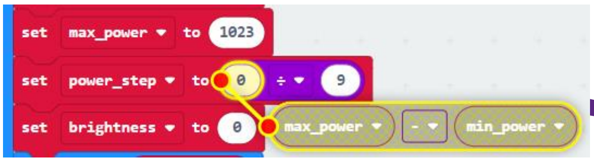
```
on start
  set min_power ▾ to 100
  set max_power ▾ to 1023
  set power_step ▾ to 0      0 ÷ ▾ 0
  set brightness ▾ to 0
  show number brightness ▾
```

2:

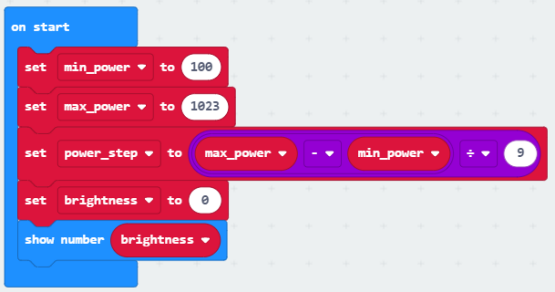
- Using the 'on start' block and the 'variables' tab, we set up the blocks seen using 'set __ to __'
- Using the codes seen on the left, set the 'Min_power' to 100
- 'Max_power' to 1023
- 'Set Brightness' to 0
- 'Set Power_step' to, using the 'Math' tab add in the '__ ÷ __' block



```
max_power ▾  - ▾  0
                  min_power ▾
```



```
set max_power ▾ to 1023
set power_step ▾ to 0 ÷ ▾ 9
set brightness ▾ to 0    max_power ▾ - ▾ min_power ▾
```

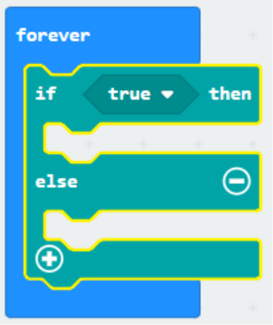


```
on start
  set min_power ▾ to 100
  set max_power ▾ to 1023
  set power_step ▾ to max_power ▾ - ▾ min_power ▾ ÷ ▾ 9
  set brightness ▾ to 0
  show number brightness ▾
```

3:

- Using the 'math' tab, use the '__ - __'. From the 'variables' tab we set up the difference between the max_power and the min_power
- Using this block, add this block to the 'set power_step to' block before the division sign as shown
- From here, we then put the number 9 into the 'math' block to signify that there are 9 steps in the brightness of the LED

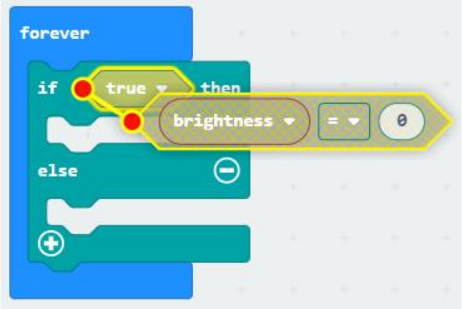
Having completed this step, the minimum value has been set at the low range of the output value. The maximum value has been set at the highest value of the output. When the micro:bit has been started, the LED will be turned off. By using the math function, we have stated there are 9 steps in the function.



```
forever
  if true then

  else

```

4:

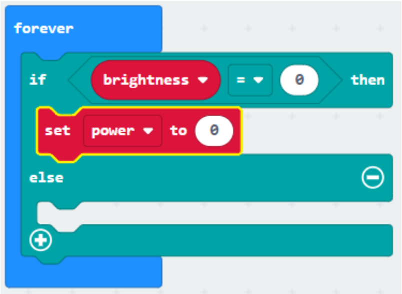
- Using the 'forever' block, the following will set up the premise that the LED power will come from the sum of the Min power and the brightness level required.
- This will create a scenario where the power is to be connected to the brightness
- The 'logic' tab is used to set up a scenario where the power is linked to the brightness of the LED



```
forever
  if true then
     brightness = 0
  else

```

5:

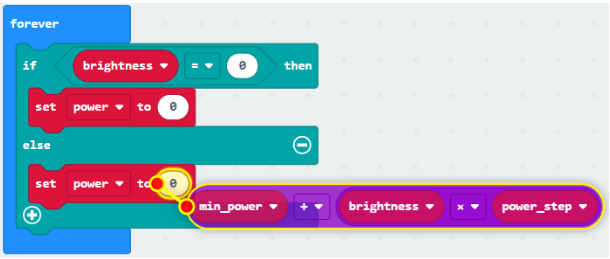
- Using the 'comparison' and the 'brightness' variable, create the diamond shaped block as shown.
- Add the comparison to the 'if true' section of the logic block so that it reads 'if brightness = 0'



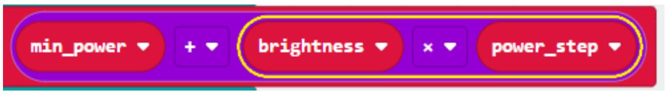
```
forever
  if brightness = 0 then
    set power to 0
  else

```

6:

- Using the 'set power to __' from the variables tab, add it to the logic gate as shown. Leave the value set to 0



```
forever
  if brightness = 0 then
    set power to 0
  else
    set power to 0
                   min_power + brightness x power_step
```



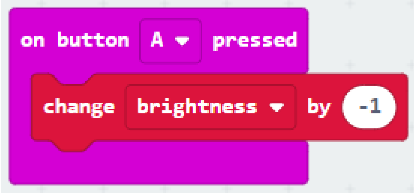
```
min_power + brightness x power_step
```

7:

- Using the 'set power to __' from the previous step again, add it to the 'else' section of the logic gate
- Using the 'math' block again as shown in step 3, build it up as shown in the image on the left
- It should now read 'set power to Min_power + (Brightness x Power_step)*

*The brackets are used to show that Brightness and Power_step are connected in a separate 'math' block, just like in step 3.

8:

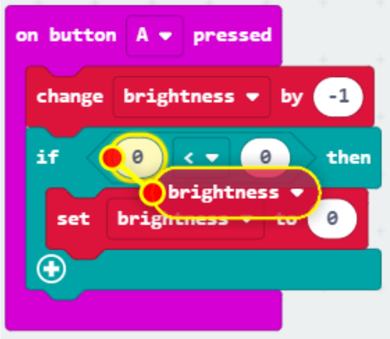


```
on button A ▾ pressed
    change brightness ▾ by -1
```

- Using the 'input' tab, select the 'on button A pressed'
- Using the 'variables' tab, use the 'change brightness by __' block

Note: *if the block is not available in the variables tab, use the change __ by and click on the drop down. Find the brightness variable*

9:

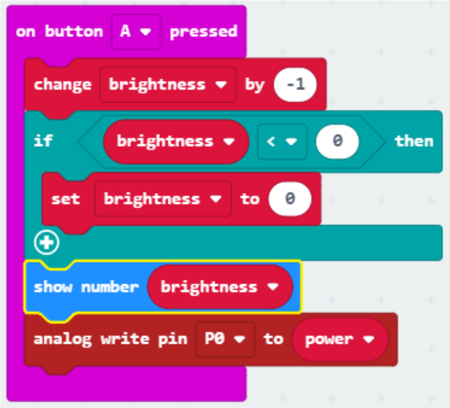


```
on button A ▾ pressed
    change brightness ▾ by -1
    if 0 < ▾ 0 then
         brightness ▾
        set brightness ▾ to 0
    ⊕
```

- Using the 'logic' tab, create a comparison as shown before
- From the 'variables' set the logic gate to 'if brightness < 0 then'
- Again, from the variables tab, use the set brightness to 0'

By doing this, we have set a limit. If someone presses the button continuously, it will not make a difference as the LED will be at brightness 0

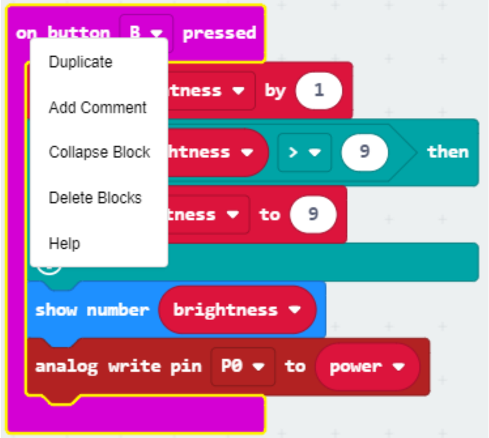
10:



```
on button A ▾ pressed
    change brightness ▾ by -1
    if brightness ▾ < ▾ 0 then
        set brightness ▾ to 0
    ⊕
    show number brightness ▾
    analog write pin P0 ▾ to power ▾
```

- Using the 'basic' tab, use the 'show number' block using 'variables'. Add in the brightness to the 'show number' block. This will show the value on screen
- Using the 'pins' tab, use the 'analog write pin P0 to __'
- Using the 'Variables' tab, add the power block as shown
- This has now set the brightness of the LED to the scale from 0-9

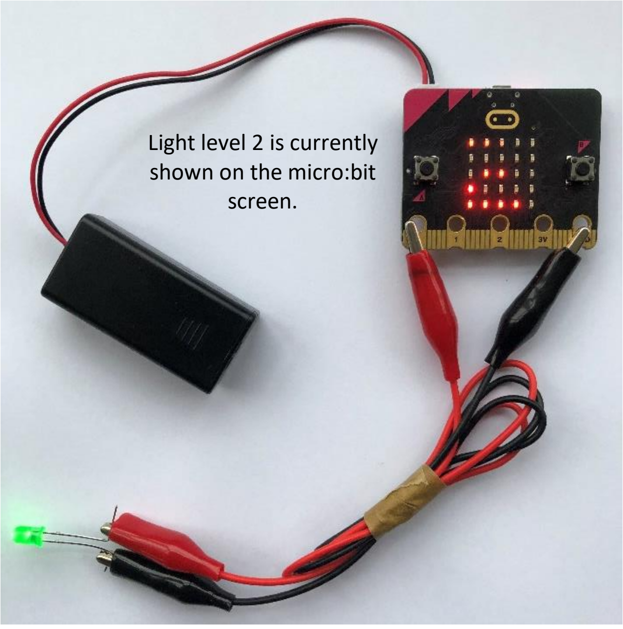
11:



```
on button B ▾ pressed
    tness ▾ by 1
    htness ▾ > ▾ 9 then
    ness ▾ to 9
    show number brightness ▾
    analog write pin P0 ▾ to power ▾
```

- Right click on the 'input' block and you will see the list shown. Click on the word duplicate
- This will allow for all the code to be copied and will re-create the blocks needed
- Using the blocks, we change 'on button A pressed' to 'on button B pressed'
- Change the 'change brightness by -1' to '1'
- Change the 'if brightness < 0' to brightness >9' and change the 'set brightness to 9' as shown.

Note: *Now we have set an upper limit and increase the brightness as you press button B*



Light level 2 is currently shown on the micro:bit screen.

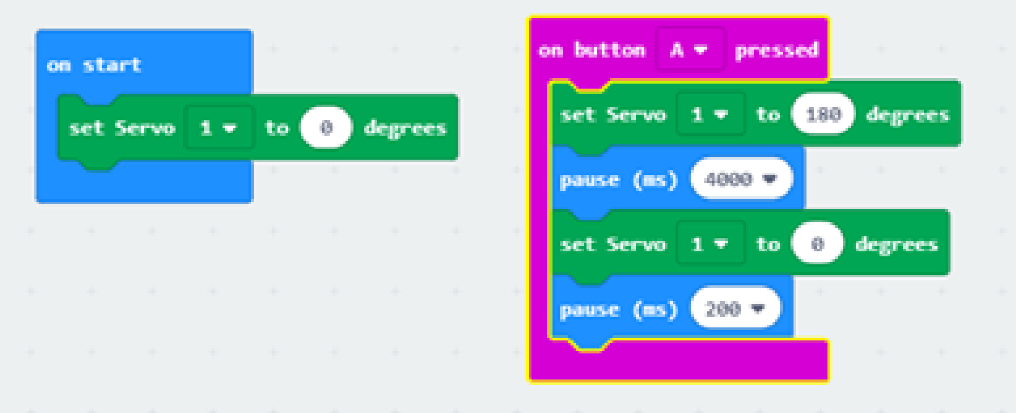
- Download the code to the micro:bit to test it
- The positive on the LED joins to P0
- The negative of the LED joins to GND
- The level of light will incrementally get brighter as button B is pressed, and darker as button A is pressed

Supporting Mechatronics in Engineering – OLE 20/21

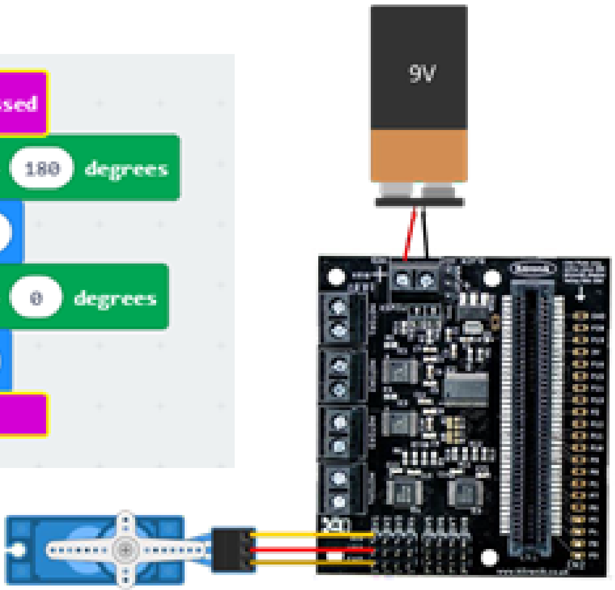
Coding an Angular Servo to any Degree

Micro:bit activity:

Introduction activity: moving a servo



```
on start
    set Servo 1 ▾ to 0 degrees

on button A ▾ pressed
    set Servo 1 ▾ to 180 degrees
    pause (ms) 4000 ▾
    set Servo 1 ▾ to 0 degrees
    pause (ms) 200 ▾
```



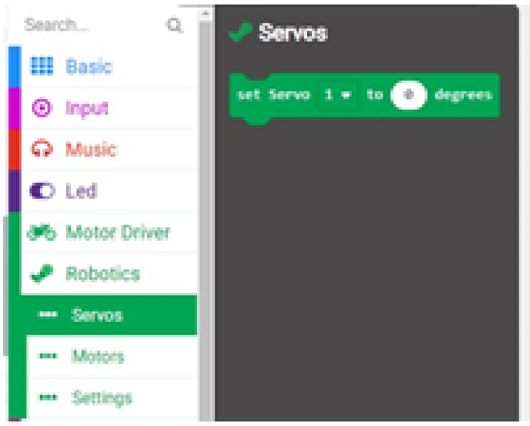
All colour blocks refer to a specific command in make code

Blue is for Basic

Pink is for Input

Red is for Music

Green is for Robotics or Motor Driver



Steps

1. Using the 'Basic' tab, select the 'on start' block in the coding section, and drop 'set servo 1 to 0 degrees' in the gap to create a chain
2. Using the 'Basic' tab, place an 'on button pressed' block and choose which input you want, A, B or A+B pressed
3. Using the 'Robotics' tab, place the 'set servo 1 to 0 degrees' and change the number to 180 or any other number
4. Using the 'Basic' tab, place 'the pause (ms) 0' block and change it to '4000'
5. Using the 'Robotics' tab, place the 'set servo 1 to 0 degrees' and change the number to 0
6. Using the 'Basic' tab. place 'the pause (ms) 0' block and change it to '200'

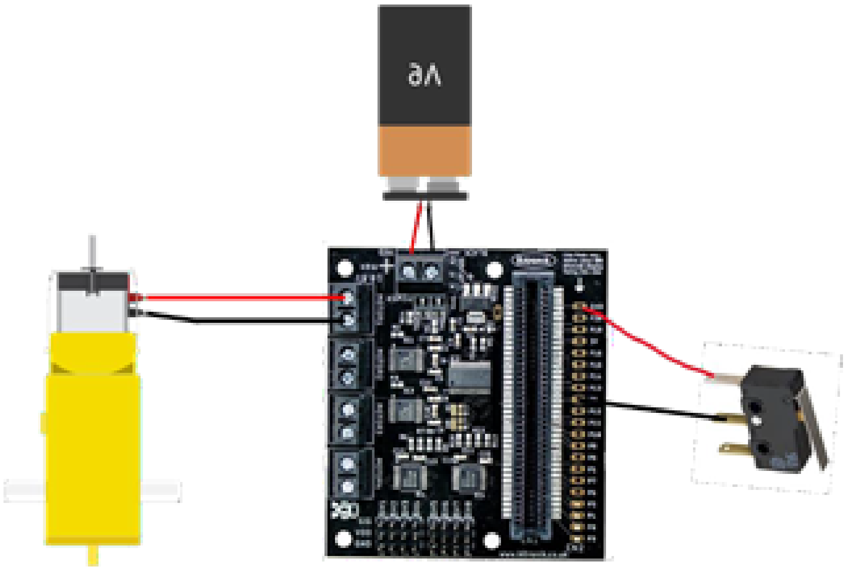
Coding an External Switch to Function

The 'limit switch' is soldered into two connection points on the track pad:

COM leg – GND
NO leg - P2

Why?

From page 3 of the learning log we can see that P2 is an input but we could also use P0 or P1. If we have more than one switch, we could include them on those inputs. Connecting it to 'GND' completes the circuits like any simple circuit



All colour blocks refer to a specific command in make code

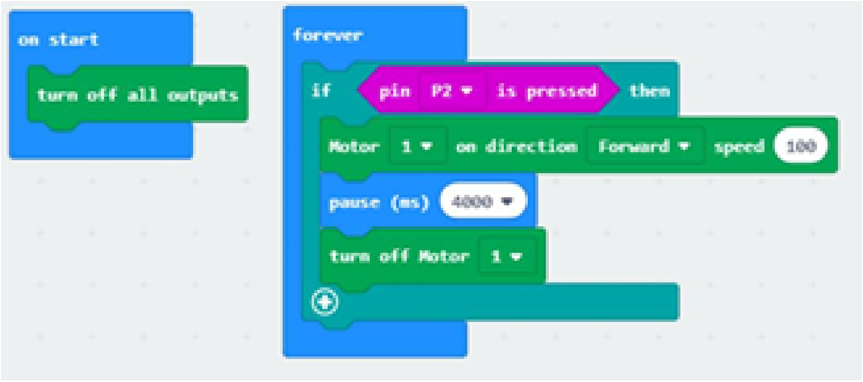
Blue is for Basic

Pink is for Input

Red is for Music

Green is for Robotics or Motor Driver

Aqua is for Logic



```
on start
    turn off all outputs

forever
    if  pin P2 ▾ is pressed  then
        Motor 1 ▾ on direction Forward ▾ speed 100
        pause (ms) 4000 ▾
        turn off Motor 1 ▾
    ⊕
```

1. Using the 'Basic' tab, place 'on start' into the code screen and add 'turn off all outputs' to the middle of it.
2. Using the 'Basic' tab, 'forever' loop button to be brought out, it means that this program will run always until a new one is put on the micro:bit
3. From the 'Logic' tab, bring out the 'if true then' option
4. From the 'Input' tab, bring out the 'pin P0 is pressed' option that looks like the diagram and drag it and place it over the true option on the 'if true then' block
5. Using the drop-down button change P0 to P2 (where we soldered in the limit switch)
6. From the 'Robotics' tab, bring 'out motor 1 on direction forward speed 0' you can use the drop down to change the direction of rotation of the motor and a value of 1-100 for the speed of the motor 100 being the fastest
7. From the 'Basic' tab, bring out the 'pause (ms) 0' block, change the seconds to control the length of time the motor will run
8. Finally, from the 'Robotics' tab, bring the 'turn off motor 1' to the chain

An tSraith Shóisearach do Mhúinteoirí

Junior CYCLE for teachers

Contact Details

Administrative Office:

Monaghan Ed. Centre,

Armagh Road,

Monaghan.

www.metc.ie

Director's Office:

LMETB,

Chapel Street,

Dundalk.

For all queries please contact:

info@jct.ie

Key websites:

www.jct.ie

www.curriculumonline.ie

www.ncca.ie

Follow us on Twitter:

@JCforTeachers

@JCt4ed

QR code - Feedback form

JUNIOR CYCLE FOR TEACHERS JCt4 An tSraith Shóisearach do Mhúinteoirí