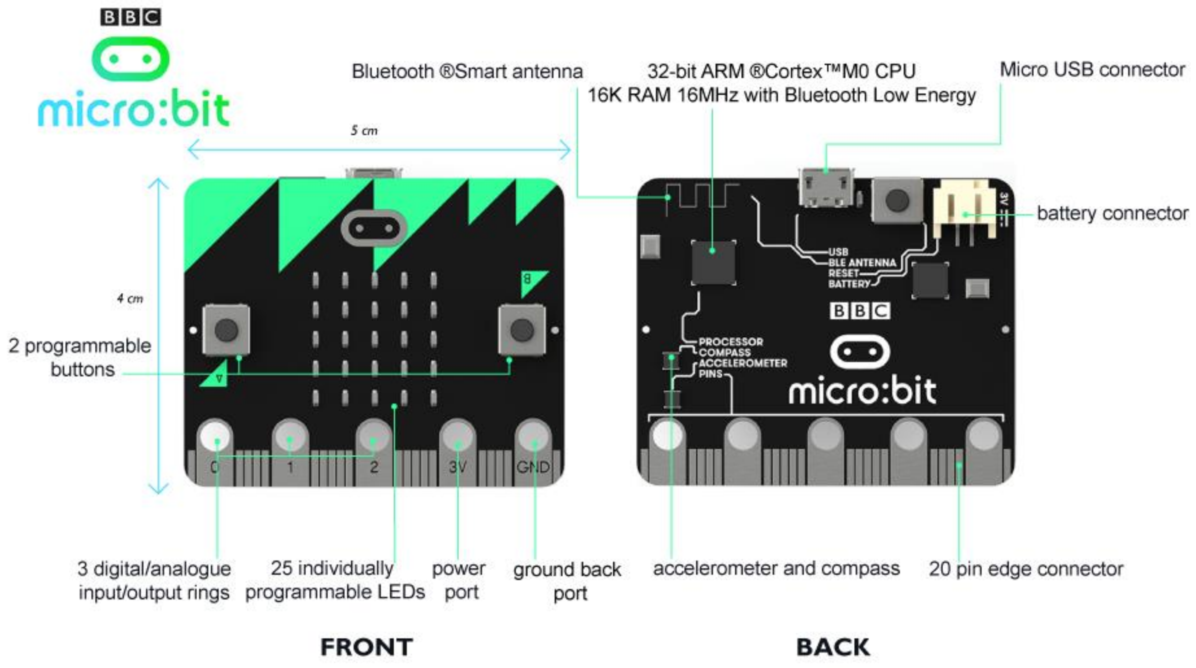
First steps in using BBC micro:bit for control and physical computing

Adrian Oldknow adrian@ccite.org 16th January 2017

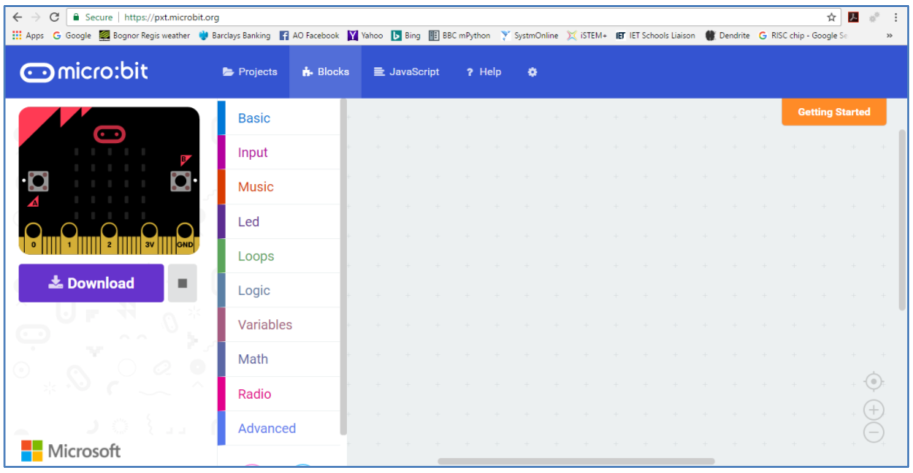
0. Writing your first program (please skip this section if you are already happy how to do it!)

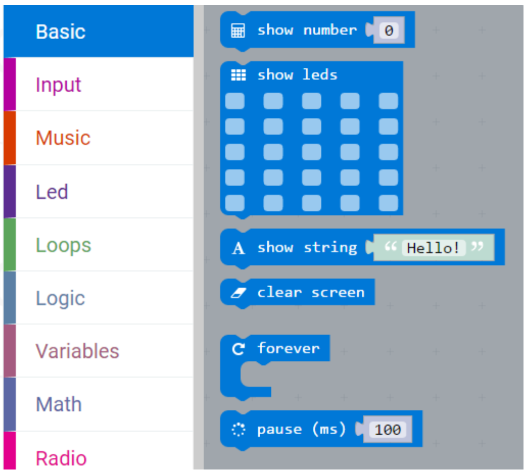

BBC
micro:bit
Bluetooth ®Smart antenna
32-bit ARM ®Cortex™M0 CPU
16K RAM 16MHz with Bluetooth Low Energy
Micro USB connector
5 cm
4 cm
battery connector
2 programmable buttons
3 digital/analogue input/output rings
25 individually programmable LEDs
power port
ground back port
accelerometer and compass
20 pin edge connector
FRONT
BACK

The image above shows the main elements of the BBC micro:bit. One million of these were distributed up to July 2016 to maintained schools in the UK to be given free to all 11-year old students. Since then they have been on general sale for around £15, including a battery case and USB connecting cable. One of the aims of the BBC digital literacy campaign has been to help people understand something now generally called the `*Internet of Things*' (IoT for short). Many modern products use the adjective `*smart*' to mean that they include a computing device to control it, and probably one which can exchange data with other devices wirelessly. Have a close look at the kinds of things this little device has built in to it. In order to make it work you need to connect a power source such as 2 AAA batteries. What is missing is an On/Off switch! As soon as the micro:bit has power it runs whatever program was loaded into it most recently. When you disconnect the power, the micro:bit continues to store the program – and will run it again as soon as you apply power.

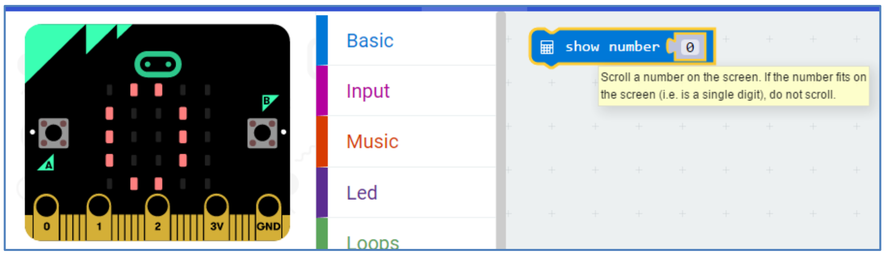
So the first thing we need to find about is the simplest way to get a program to run on it. There are several ways to do this. We will start off with the programming environment developed by Microsoft Research, called the Programming Experience Toolkit, or PXT for short: https://pxt.microbit.org/.

If this is the first time you have used the PXT editor, then you will have a blank program area. Otherwise it will open the last program you built. In which case click on `***Projects***' and select `***New Project***'.

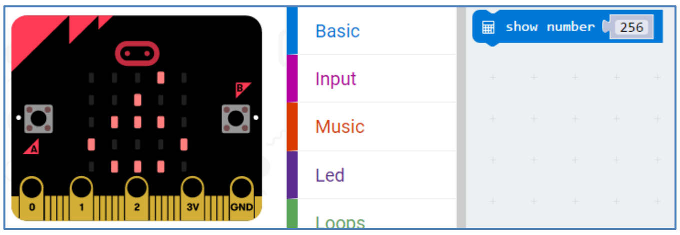




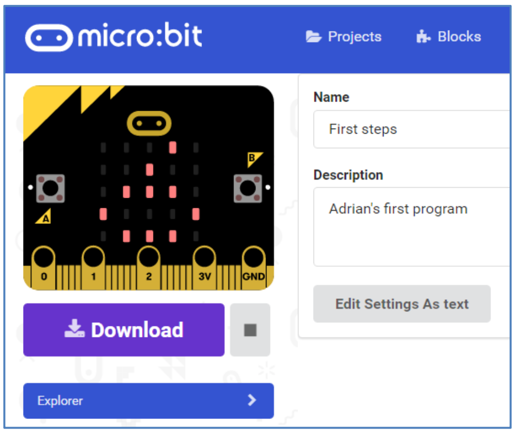
The image of the micro:bit at the top left is a very clever animated device, called an `emulator', which allows you to test programs without even having your own micro:bit! The coloured words in the next column are links to the various building blocks from which you will build your program. If you click on one of these, such as `***Basic***' you will see a collection of blocks from which you select one and drag it to the programming window on the right. Let's try the `***show number***' block. When you click on it a yellow border shows up, which means you have started to run your simple program. You should see that the emulator shows the number you have chosen to display. There will also be some explanation about this block called `***showNumber***'. If you click on this you will open up a new tab with a page of information from the [reference manual](), including an example. Try deleting the zero and entering your own number into the `show number' block. I have entered `256', and you will see that the display scrolls to show each digit in turn, and the program ends with the `6' on the display.



Scroll a number on the screen. If the number fits on the screen (i.e. is a single digit), do not scroll.



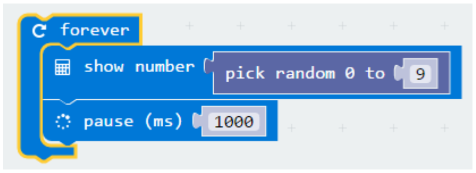
See what happens if you enter something which is not a whole number like `3.14' or `two'. It would be a good idea, now that you have started to build a program, to give it a name like `*first steps*'. Use `***Projects***' and then `***Rename project***'. Close the dialog to get back to the editor.



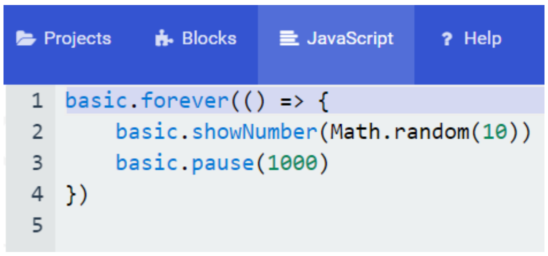
Name
First steps

Description
Adrian's first program

Most programs do something repeatedly, and use what is called a `loop'. The simplest kind of loop is the `***forever***' loop.

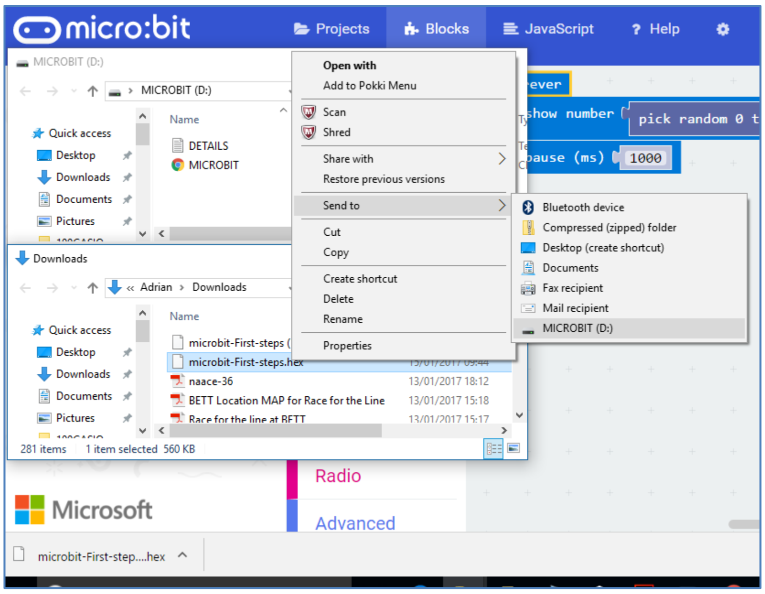


Use the `***Basic***' menu and drag in a `***forever***' block and a `***pause***' block. Place the `***show number***' and `***pause***' blocks within the `jaws' of the `***forever***' loop. From the `***Math***' menu drag in a `***pick random***' block and place it inside the `***show number***' block. Edit the number and change it from 4 to 9. Edit the number in the `***pause***' block and change it from 100 to 1000. So the new program will continually generate a random whole number (called an `integer') between 0 and 9, display it and wait 1000 milliseconds (i.e. 1 second) before doing it all again. Check that this is what the emulator does.

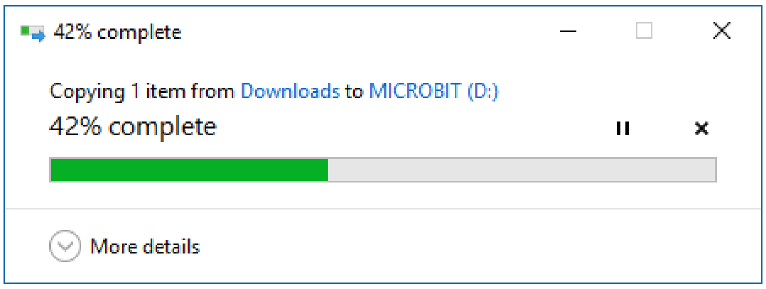


```
basic.forever(() => {
    basic.showNumber(Math.random(10))
    basic.pause(1000)
})
```

Now explore what happens when you click on `***JavaScript***'. This is exactly the same program but written in a text, rather than a graphical, format. It's not as pretty, but it is much more convenient to use for longer programs, especially if you want to print them out. Click on `***Blocks***' to swap back to the graphical version. Now we are ready to send our working program to a micro:bit. This has two stages. The first is for the PXT editor to create a version of the program which the micro:bit can understand. This converts all the lines of the JavaScript program using a number code in what is called `base 16' or `hex' for short. The resulting hex file is stored on your computer, usually in a folder called something like `Downloads'. The second stage is to connect a micro:bit to the computer with a USB cable and to transfer the file to it. Once you have got used to this you should find the whole business of writing a program on a computer, and loading it into the micro:bit, pretty painless! Click on `***Download***'. This opens a window at the bottom of the web-browser display which shows you that a file is being saved.



It is called `microbit-First-steps.hex'. If you right-click on this name you can select the option to `Show in Folder'. This opens your `Downloads' folder. You should see your file at the top with the current date and time. My version has a size of 561 Kb. If you now plug your micro:bit into the USB port of your computer, after a few seconds another window will open showing it as an external storage device with a name something like `MICROBIT (D:)'.



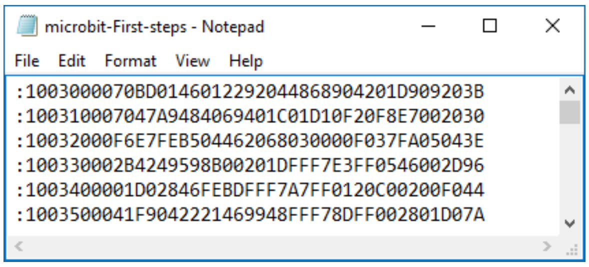
42% complete

Copying 1 item from Downloads to MICROBIT (D:)

42% complete

More details

If you right-click on your hex file, you will have the option to `Send to >' and a list of possible locations. Select `MICROBIT (D:)' and a little dialogue box opens to show that the file is being transferred. You will also see an LED on the back of the micro:bit flash while this happens. As soon as the transfer is complete your program should start to run. At the moment the micro:bit is being powered through the USB cable from the computer. If you disconnect it, you can then attach a battery pack and the program will happily start to run – at least until you disconnect the power.



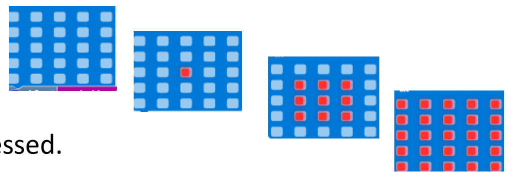
```
:1003000070BD01460122920448689042 01D909203B
```

If you are curious you can open your hex file with something like `Notepad' and see that it does indeed just contain a load of numbers using the characters 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F. Once you have created a hex file you can send it to your friends. They can either send it to their own micro:bits, or open it in the PXT editor. Try it now.

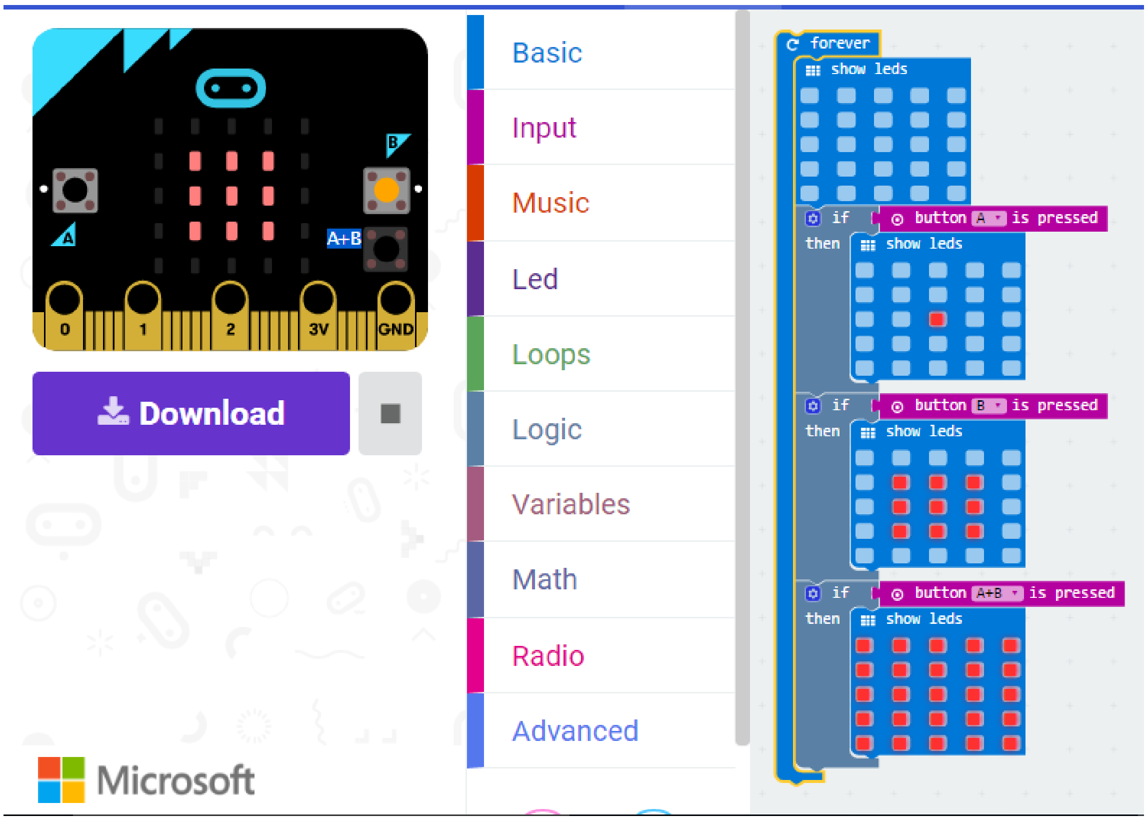
1. Making the micro:bit display respond to its buttons and sensors

The BBC micro:bit is quite a different animal from the Raspberry Pi. The RPi has to have input (keyboard and mouse) and output (TV or monitor) devices connected in order to be useable. The micro:bit has its own input (buttons and sensors) and output (the 25 LED display) devices built in. It can also have other ones attached, such as buzzers, speakers and electronic components. In this section you will learn how. We will start with the simple inputs – the A and B buttons. The micro:bit has a very simple way of accepting inputs from its buttons and using them to control the pattern of LEDs illuminated on the display. Start a New program. We will build a simple 4-state device which tells the micro:bit what to display

(a) when no buttons are pressed:
(b) when button A is pressed:
(c) when button B is pressed:
(d) when both buttons A and B are pressed.



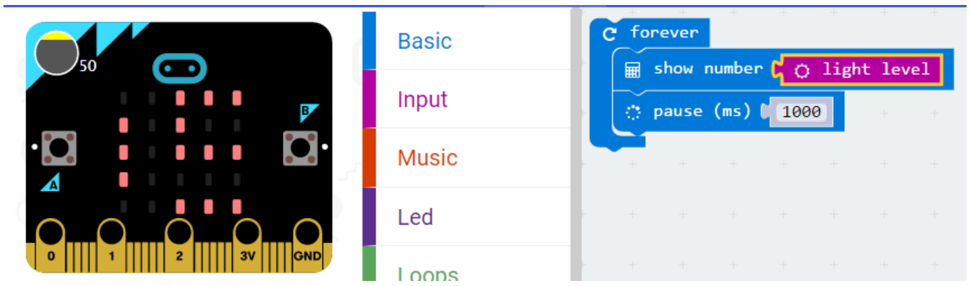
Here is the complete program:



```
forever
  show leds
  if button A is pressed
  then show leds
  if button B is pressed
  then show leds
  if button A+B is pressed
  then show leds
```

The `***show leds***' blocks are `***Basic***' commands. The `***if***' blocks are `***Logic***' commands. The `***button***' blocks are `***Input***' commands. By default, the led display will be blank unless either or both of the buttons are pressed. See what happens to the emulator when you run the program. Note that it creates an extra A+B button! When you are happy, give your program a name like `Push buttons'. Download it as the hex file: `microbit-Push-buttons.hex' and transfer it to your micro:bit. Test that the right things happen when you press any or all of the buttons. Now you have the basic idea we can see how the micro:bit can make decisions about what to display based on its own sensors. The first one we will try is the Light sensor. This is isn't shown on the actual micro:bit, but it uses a clever way to detect light intensity falling on the 25 led display. (The technical details are here.) Start a new program and build the following one. The `***light level***' block is in the

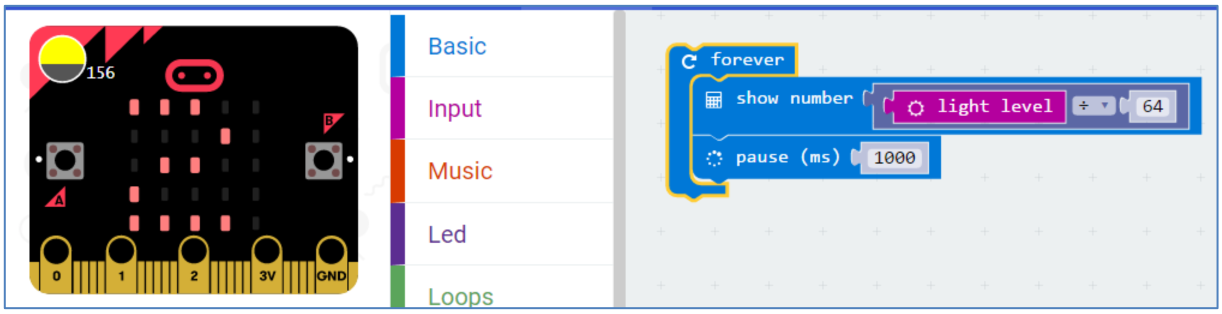
`***Input***' menu.  This returns a value between 0 and 255.  You can simulate this by using your mouse to pull the yellow `blind' up and down over the simulated sensor at the top left of the emulator. Because the result can have up to three digits, the display will scroll to show you the simulated reading.



```
forever
  show number ( light level )
  pause (ms) 1000
```

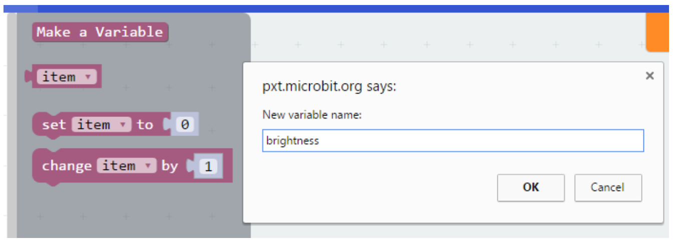
Give your program a title such as `Light level'.  Download it as the `microbit-Light-level.hex' file and transfer it to your micro:bit. Try moving it closer and further from a light source, or shine a torch at the display.

We will now use a little trick to make the `***light level***' easier to work with.  Find the `***divide***' block in the `***Math***' menu and insert it in the `***show number***' block. Edit the number to the right of the `÷' sign to read 64.  If you use a calculator to do a divide operation you will usually get a decimal point in the answer.  Try dividing 156 by 64 and see what you get.  Division on the micro:bit works rather differently.  You only get the whole number (aka `integer') part of the result. Sometimes this operation is called `integer divide'.  Test the resulting program with the emulator.  Now we see that a light level of 156 produces the single digit `2' on the display. What are the only possible numbers this program can display?

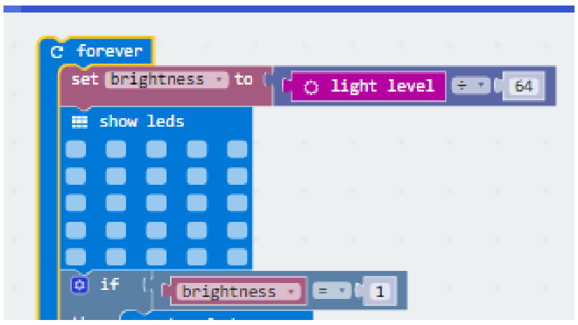


```
forever
  show number ( light level ÷ 64 )
  pause (ms) 1000
```

Don't bother saving, downloading and transferring (aka `flashing') the current program to your micro:bit. We can now merge this idea with our 4-state program, replacing the `***button***' test with a `***light level***' test.  We will now introduce the idea of a `variable'.   In the `***Variables***' menu there is a block called `***Make a Variable***' which allows you to create a new one. Click on `***Projects***' and re-open your `Push buttons' program (or create a new one if you can't find it). Insert a new `***set***' block from the `***Variables***' menu, select `***brightness***' instead of `***item***' as the variable name and insert the `***light level***' and `***divide***' blocks as shown. In each of the `***if***' blocks replace the `***button***' block with an `***equals***' block from the `***Logic***' menu. Insert the variable name `***brightness***' and edit the test value to 1, 2 and 3 in turn.

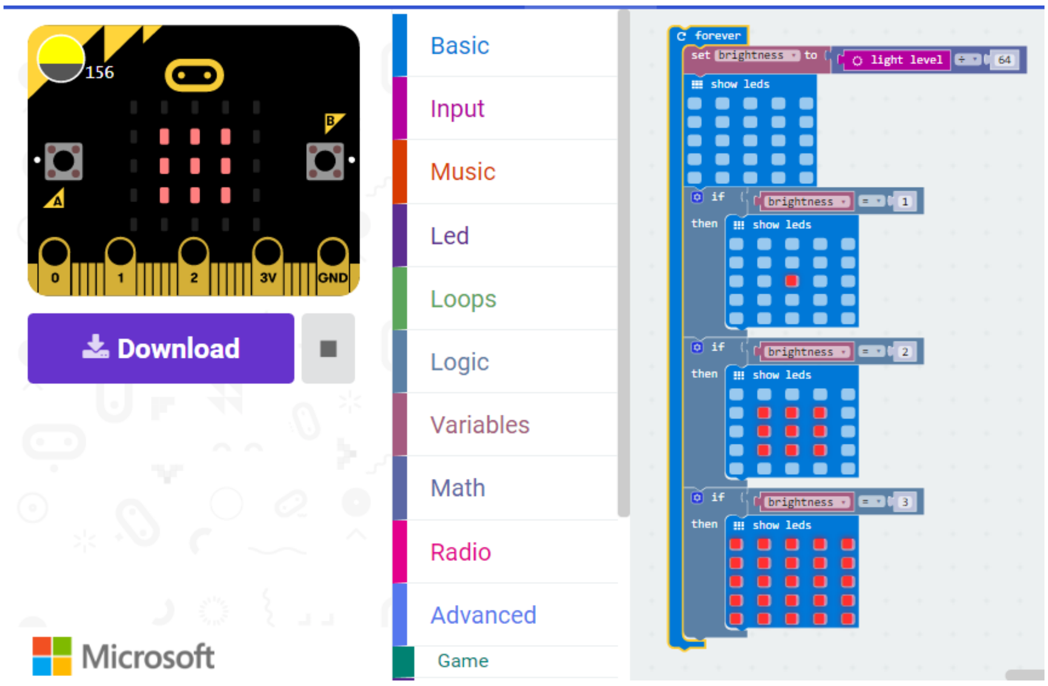


pxt.microbit.org says:
New variable name:
brightness



```
forever
  set brightness to ( light level ÷ 64 )
  show leds
  if ( brightness = 1 )
```

The completed program is shown here. Check it works using the emulator to simulate changes in light intensity.



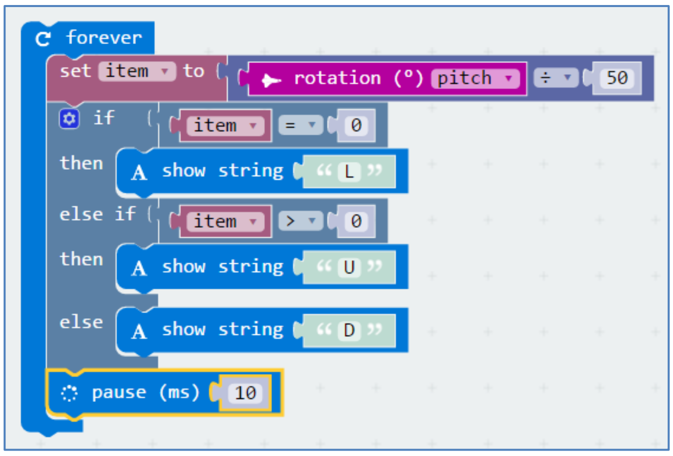
```
forever
    set brightness to ( light level ÷ 64 )
    show leds
        . . . . .
        . . . . .
        . . . . .
        . . . . .
        . . . . .
    if brightness = 1
    then show leds
        . . . . .
        . . . . .
        . . # . .
        . . . . .
        . . . . .
    if brightness = 2
    then show leds
        . . . . .
        . # # # .
        . # # # .
        . # # # .
        . . . . .
    if brightness = 3
    then show leds
        # # # # #
        # # # # #
        # # # # #
        # # # # #
        # # # # #
```

But the logic is all wrong!!! We need more leds when it gets darker, not lighter!

So can you edit the program to fix it? Now you have not only created a program, but also detected and corrected an error. That process is called `debugging'. Save your program and transfer it to your micro:bit to check it now works properly. Move the micro:bit so that the light-levels change and check it works OK.

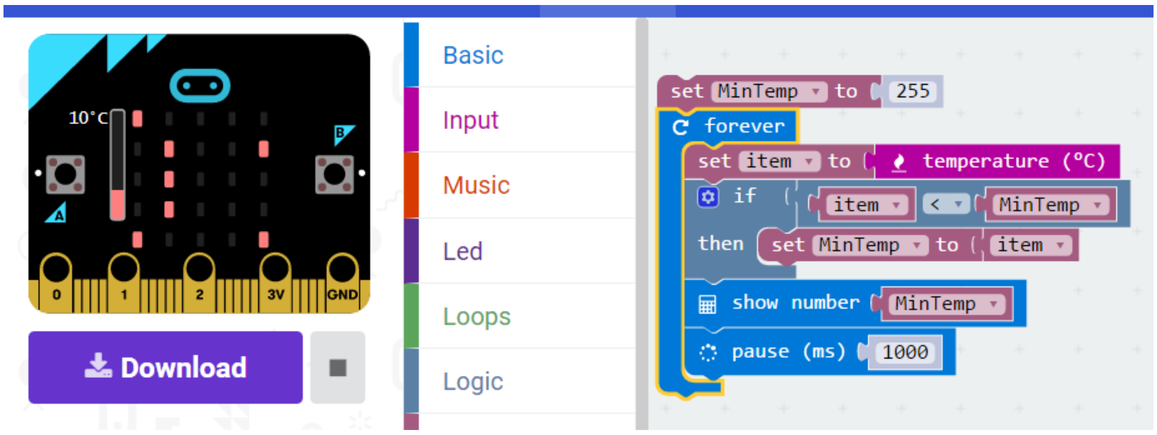
Congratulations. You have built your first Internet of Things device, using the light sensor to switch on and off the leds in the display. That's how most current cars have automatic systems to switch their lights on and off as the light level changes. You could attach your micro:bit to the back of your bike's saddle to create a smart rear lamp. I have saved my version of the corrected program on Dropbox with this link. Check you can copy this program to your computer, open it with the PXT editor and transfer it to your micro:bit.

The micro:bit has several sensors other than temperature or light which can be used to detect motion. These include accelerometers and magnetometers which can be used to detect gestures, collisions and direction. The `***rotation***' block from the `***Input More***' menu can measure the `***pitch***' (forward and back) and `***roll***' (side to side) motion of the micro:bit. This program turns the micro:bit into a spirit-level.



```
forever
    set item to ( rotation (°) pitch ÷ 50 )
    if item = 0
    then show string "L"
    else if item > 0
    then show string "U"
    else show string "D"
    pause (ms) 10
```

So how does the program work? It continuously checks the angle being returned by the `***pitch***' sensor. Dividing this angle by 50 returns a value for the variable `***item***' which you could read using the `***show number***' command from the Basic block. This is a single digit signed number between -4 and +4. We are just going to test whether this is a positive, negative or zero number. If it's zero we display the letter `L' for Level, if it's negative we display `D' for Down, and if it's positive we display `U' for Up. Then we have a slight pause before doing the job again. You can test this with on screen emulator by clicking the mouse somewhere inside the image of the micro:bit at the top left. Check that you can get it to display each one of the 3 letters. So this is another way to use the micro:bit's built-in sensors to control an output. It simulates the way a smart-phone or tablet senses the orientation of the way in which you are holding the device – and so is able always to display text in the right direction for you to read clearly. The last example in this section turns the micro:bit into another kind of instrument – a thermometer with a difference. We will use the variable `***MinTemp***' to record the lowest temperature reached while the micro:bit is awake and sending. This starts with a large number stored in it. Every time the temperature sensor records a value lower than the lowest recorded so far, we replace the value stored in the variable `***MinTemp***' with the current one, saved in `***item***'. You can test the program with the emulator by sliding the simulated thermometer. Once you have transferred the program to the micro:bit, you can detach it, attach batteries and place the micro:bit in the fridge. When I take mine out after half an hour I find the lowest value was 6°C. So my nice bottle of white wine is probably just a bit too cold!

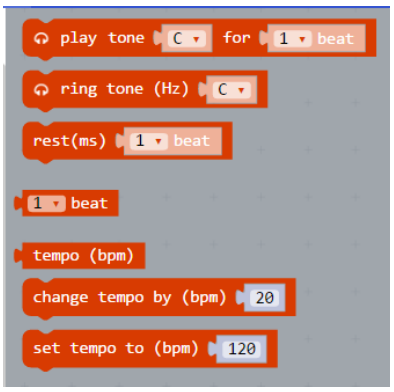


```
set MinTemp to 255
forever
    set item to temperature (°C)
    if item < MinTemp
    then set MinTemp to item
    show number MinTemp
    pause (ms) 1000
```

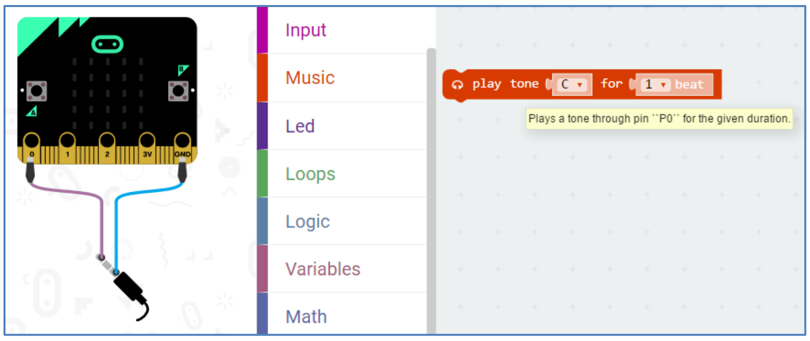
Can you edit the program to record the maximum temperature reached? I suspect it Will not do a micro:bit much good by testing this in a kitchen oven! Can you develop a program which displays the current temperature while also storing the maximum and minimum temperatures reached? Make it display the maximum temperature when the A button is pressed and the minimum when the B button is pressed. Place your micro:bit in a plastic bag and leave it outside for 24 hours to check the max and min temeratures.

2. Connecting external devices

One of the micro:bit menus is called `***Music***', but there is no speaker on the micro:bit itself.



```
play tone C for 1 beat
ring tone (Hz) C
rest(ms) 1 beat
1 beat
tempo (bpm)
change tempo by (bpm) 20
set tempo to (bpm) 120
```

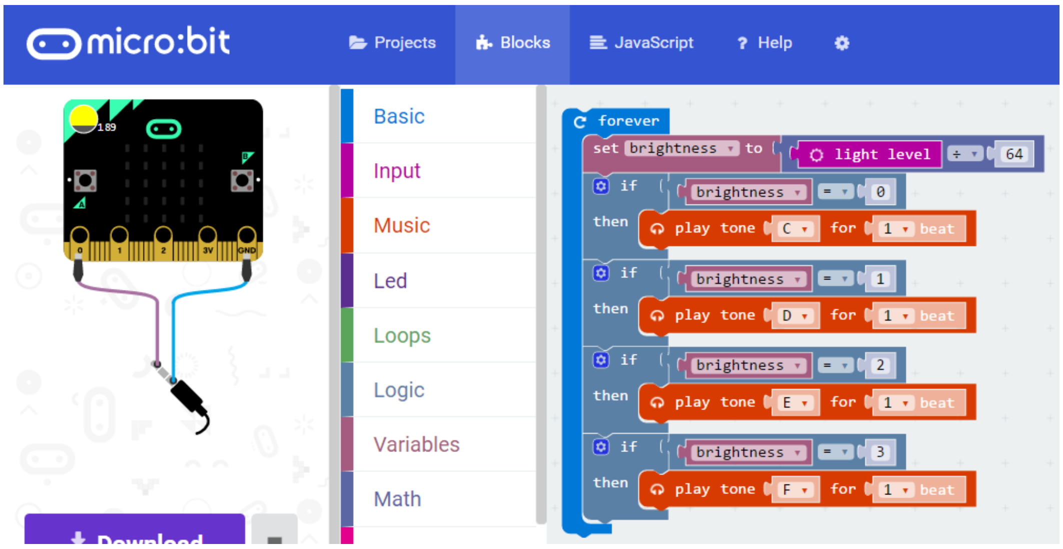


```
play tone C for 1 beat
```

Plays a tone through pin ``P0`` for the given duration.

When you try to run the simple program to play a single note, the emulator suggests that you need to connect `pin0' and `GND' to a headphone or speaker. In the photo below, the bottom left shows a simple black circular buzzer attached to the micro:bit's Pin0 with a green cable and crocodile clips, and to its GND pin with a black cable. You could use the crocodile clips to attach to the separate sections of the jack plug of headphones or speakers.

For £2.25 you can buy a ready-made jack plug adaptor from `Handy Little Modules' or for £5 you can buy the Kitronik M1 power adaptor (bottom right) which has a built in buzzer connected to the right pins. So now you can build a range of alarms to display warning signs on the led array as well, as making noises.

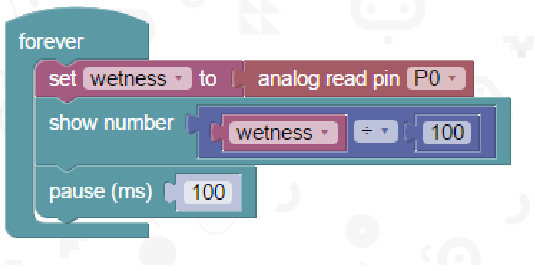


```
forever
    set brightness to ( light level ÷ 64 )
    if ( brightness = 0 )
    then
        play tone C for 1 beat
    if ( brightness = 1 )
    then
        play tone D for 1 beat
    if ( brightness = 2 )
    then
        play tone E for 1 beat
    if ( brightness = 3 )
    then
        play tone F for 1 beat
```

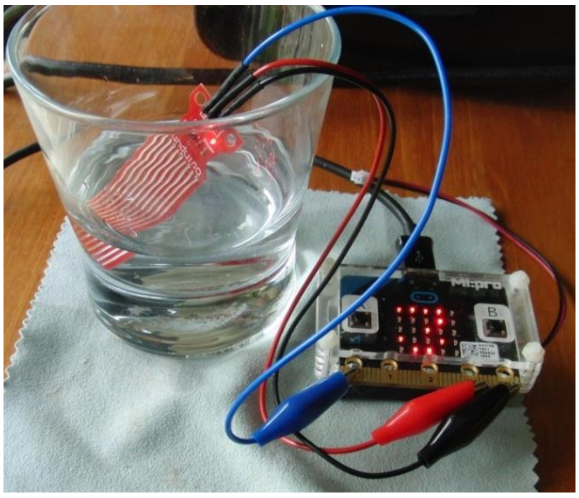
Could you use the temperature sensor to sound an alarm and flash a light if the temperature in the room gets too high, or too cold? This is, of course, the principle used by a thermostat to control the central heating in a home, or the climate control in many modern cars.

As well as external outputs, such as a buzzer, or bright/coloured LEDs, you can also attach external inputs such as sensors. A very nice project to build an automated plant watering system is described here. This uses an external moisture sensor to detect the dampness of the soil in a plant pot. It could be used to sound an alarm so that you are prompted to water the plant. But, better still, the micro:bit could turn on a pump to water the plant automatically. The water sensor costs £3 and the pump costs £5. More tutorials and suggestions are on this site.

Let's test the water sensor. The red lead is connected to 3V, the black to GND and the blue to pin P0. Here is the simple code to check it works:

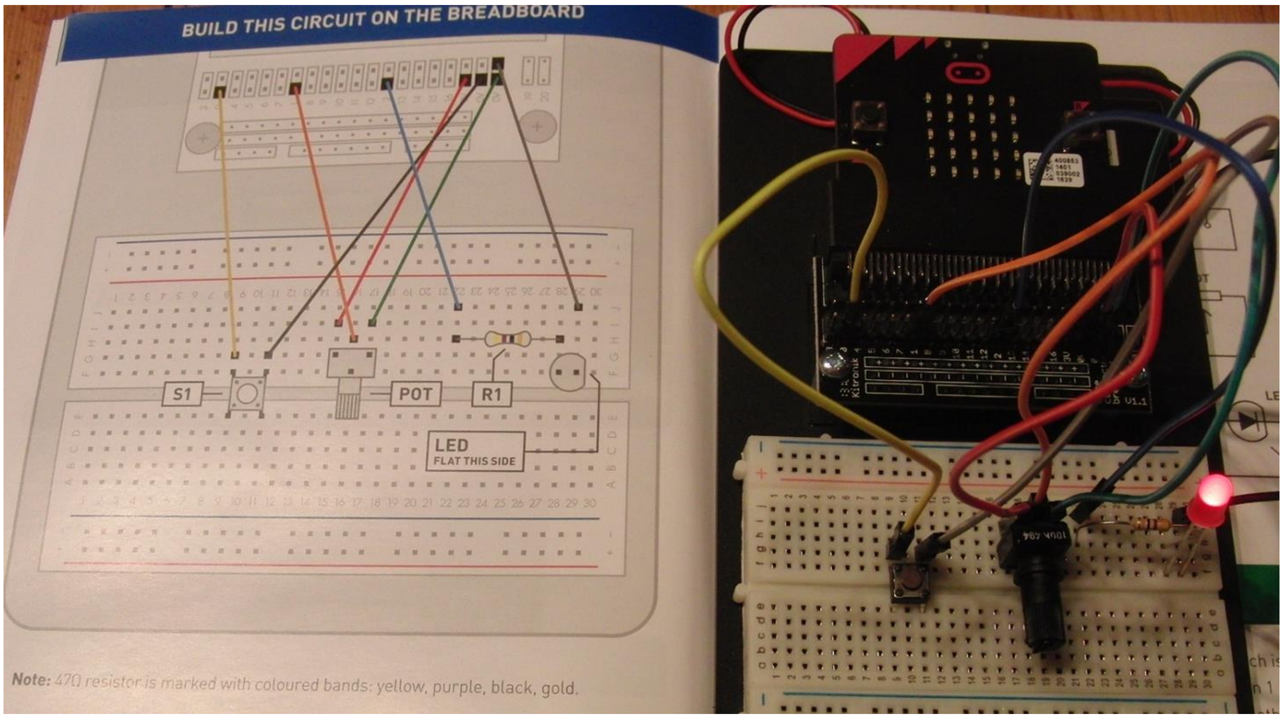


```
forever
    set wetness to analog read pin P0
    show number ( wetness ÷ 100 )
    pause (ms) 100
```



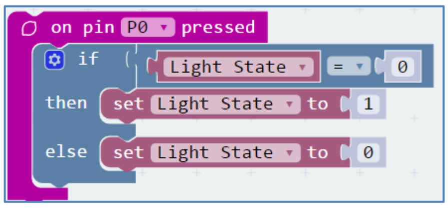
3. Designing a working system – smart lighting

In this section, we will simulate working like an electronic systems designer. We will just use the micro:bit as the smart control device. We will design a system in which a dimmable light emitting diode (LED) responds to the amount of light falling on another electronic component called a light dependent resistor (LDR). This simulates the automatic lighting system in a car or house. These components are widely available and quite cheap. A very convenient resource for this kind of activity is ready made kit, such at the Kitronik's `Inventor's Kit for the BBC micro:bit' costing £25. This consists of an `edge connector' into which you plug your micro:bit. This is connected to two parallel rows of pins which you use to attach wires. This is stuck on a base board along with an object called a `bread-board- with many holes in to allow you to place components, like an LED, and connecting wires. The photograph below shows page 24 of the tutorial with the circuit diagram we need to build. On the right is the assembled Inventor's kit with the programmed micro:bit plugged into the edge connector, with the battery boxed tucked away underneath. The coloured leads slip over the I/O pins connected to the micro:bit through the edge-connector and plug into holes in the bread board. We are using three I/O pins, 0, 1 and 2. Also the GND and +3V power pins. The first test components are a push switch, a potentiometer (variable resistor), a red LED and a 47Ω ohm resistor.


BUILD THIS CIRCUIT ON THE BREADBOARD
S1
POT
R1
LED
FLAT THIS SIDE
Note: 47Ω resistor is marked with coloured bands: yellow, purple, black, gold.
Kitronik

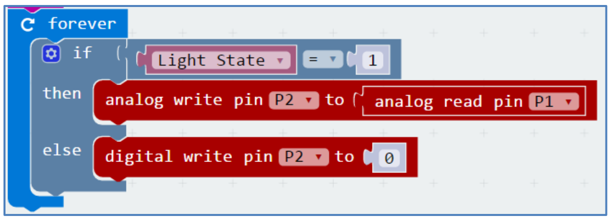
In this first version we will use the potentiometer as a manual dimmer-switch, like we used the A and B buttons in earliest example.

Here is the program written in the current Microsoft PXT editor. I have called it `Dimmer switch'. We use a variable called `***light state***' to tell whether the LED is switched on (1) or off (0). So the first bit of code just tells the micro:bit to use the push switch attached to pin P0 as a `flip-flop' to change the state of the LED.

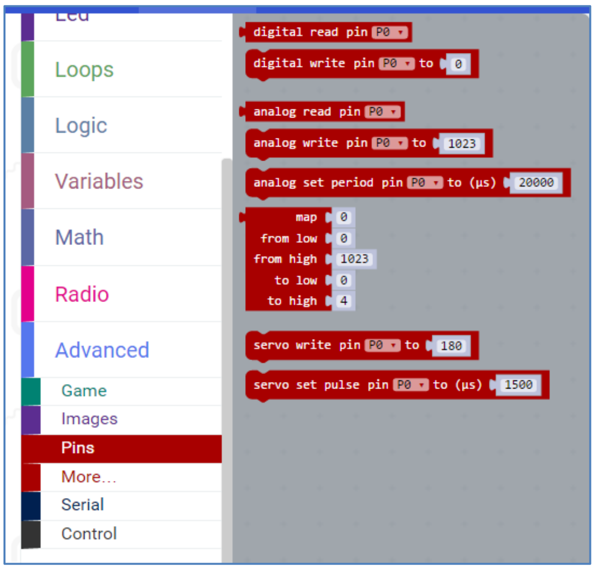


```
on pin P0 pressed
    if Light State = 0
    then set Light State to 1
    else set Light State to 0
```

Now is the time to have a look at the `***Advanced***' blocks menu. The one that we need is called `***Pins***'. This allows you to read values from digital (e.g. a switch) or analog (e.g. a potentiometer) inputs and to send out signals to digital and analog devices.

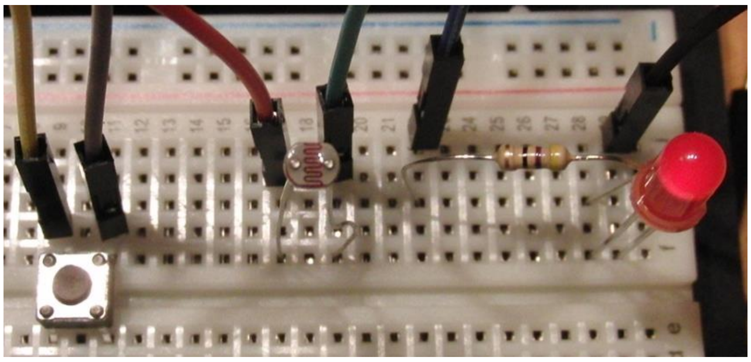


```
forever
    if ( Light State = 1 )
    then
        analog write pin P2 to ( analog read pin P1 )
    else
        digital write pin P2 to 0
```

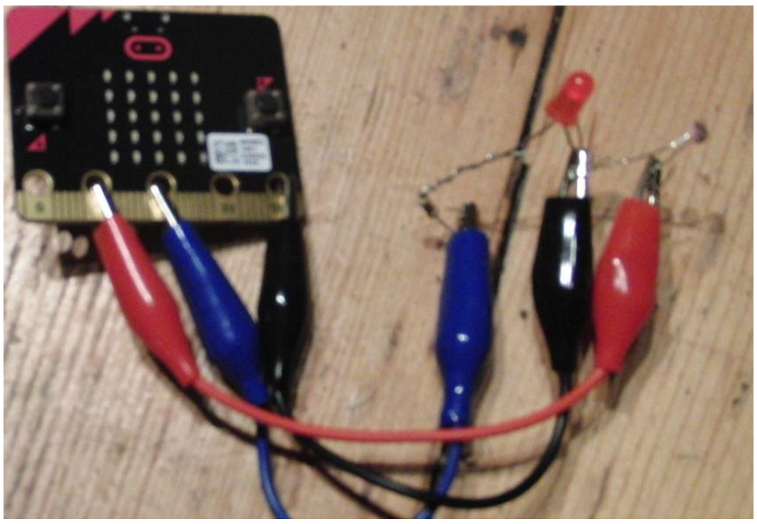


```
digital read pin P0
digital write pin P0 to 0
analog read pin P0
analog write pin P0 to 1023
analog set period pin P0 to (µs) 20000
map 0
  from low 0
  from high 1023
  to low 0
  to high 4
servo write pin P0 to 180
servo set pulse pin P0 to (µs) 1500
```

The second block of the program uses the potentiometer attached to the analog pin P1 to control the brightness of the LED connected to the analog pin P2. When you have transferred the program to the micro:bit you can use the push button to switch the LED on and off. You can turn the spindle attached to the potentiometer to control the brightness of the LED.

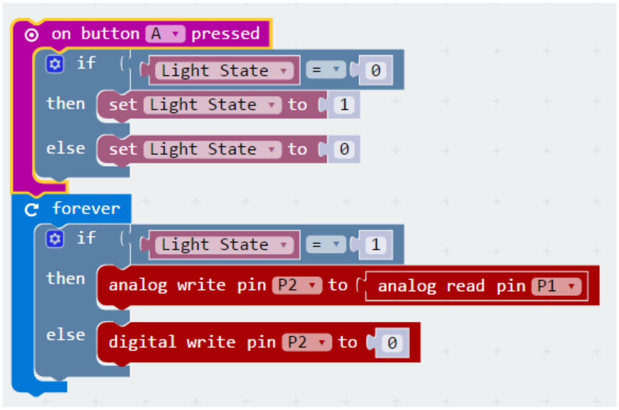


Once your program has been tested and works OK, you can replace the potentiometer with a light dependent resistor. The program doesn't need any changes. But we no longer need the lead to connect the 3V to the potentiometer. The LDR is just connected to GND and P1.



Now we have developed our working system we can dispense with the bread-board and make a bespoke circuit. I have used 3 crocodile clip leads. The black lead connects GND to one leg of the LDR and to the negative leg of the LED. One leg of the resistor is twisted round the positive leg of the LED. The other leg of the resistor is attached with the blue lead to Pim2. The red lead connects the other leg of the LDR to Pin1. There is no point in adding a push switch to the system as we can use button A, say, instead.

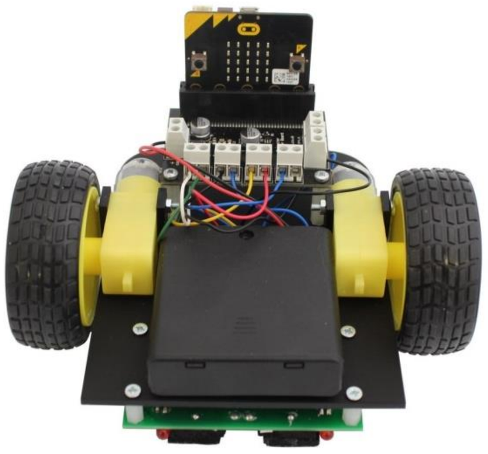
So we just need a very small modification to the program. Of course, if you wanted to market your device you would probably like to put all the components, including the micro:bit and battery, in a nicely designed box for which you could charge a substantial amount!



```
on button A pressed
    if ( Light State = 0 )
    then
        set Light State to 1
    else
        set Light State to 0
forever
    if ( Light State = 1 )
    then
        analog write pin P2 to ( analog read pin P1 )
    else
        digital write pin P2 to 0
```

The 10 well described experiments in the Inventor's kit give a pretty good feel for how to design and control your own circuits and devices. There is now an increasing number of other fun projects for use with micro:bits on the market. Here are a few examples:

Kitronik's `Line-following buggy' at £26



DIMM and UFO from Binary Bots (m:b included, £40 each)

Cheap sensors and accessories are available from `microbit accessories', such as head-phone adaptors, a natty plant watering project and the chick-bot robot. Using the head-phone adaptor you can also attach speakers to the m:b. I also recommend two other devices.

The Kitronik MI:power board costs £5 and provides a robust casing for the m:b as well as compact power from a coin battery.

Once you have designed, tested and de-bugged some interesting systems of your own, please write up what you have done to share it with others. Better still, create a YouTube video and upload it together with your notes and hex program files to somewhere you can share it. Then send the links to your friends.

Happy tinkering!