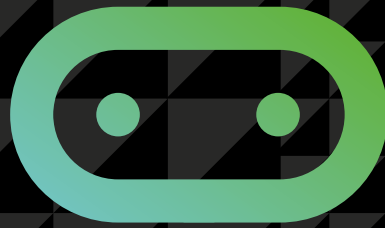


BBC



micro:bit

QUICK START GUIDE FOR TEACHERS

Hi there.

Wanna code?



- Tips and advice for getting to grips with the BBC micro:bit
- Step-by-step coding challenges with clear solutions
- Guidance on creating and sharing your own programs

Supported by



Microsoft

Foreword

Computational thinking is the thought processes involved in formulating a problem and expressing its solution(s) in a way that a computer – human or machine – can carry out effectively. In 2006, I began to advocate that everyone, regardless of profession, of career, or of age, can benefit from learning how to think computationally^[1]. With the BBC micro:bit, the BBC and its partners, including Microsoft, catalyze our realization of this dream. Children can learn how to think computationally by first formulating a problem and conceptualizing a solution. Then, by expressing their solution using a code editor, such as Microsoft Touch Develop, and by compiling and running their program on the BBC micro:bit, they can see their code come alive!

I commend the BBC and the UK for their leadership in the *Make It Digital* initiative. Teaching children at an early age the fundamentals of computing helps provide them with the programming skills and the computational thinking skills they will need to function in the 21st century workforce. Programming the BBC micro:bit will teach children basic coding concepts, such as variables, types, procedures, iteration, and conditionals. Solving problems with the BBC micro:bit will expose children to computational thinking skills, such as abstraction, decomposition, pattern matching, algorithm design, and data representation. Students knowledgeable with these skills will be in high demand by all industrial, government, and academic sectors, not just information technology.

Most importantly, the BBC micro:bit will introduce children to the joy of computing. Making one's personal device do whatever one wants is empowering. Programming the BBC micro:bit will tap into a child's imagination and creativity. The device and programming environment provide a playful way to explore a multitude of computational behaviors.

I am thrilled to see this Quick Start Guide for Teachers, which is rich with examples that show how hands-on coding is easy and natural. Let's have fun together!

Jeannette M. Wing

Corporate Vice President, Microsoft Research
25 May 2015



[1] Jeannette M. Wing, "Computational Thinking", Communications of the ACM 49 (3): 33, 2006.

The <i>Make It Digital</i> initiative	02
The BBC micro:bit	04
The BBC micro:bit website	06
Getting started with the Microsoft Touch Develop Editor	08
Uploading programs to the BBC micro:bit	10
Coding building blocks	11
Lessons	12
Solutions	31
The BBC micro:bit and the curriculum	32

Acknowledgements

Written by: Miles Berry and Ray Chambers
Additional material written by: Ross Lowe

Consultant: Roger Davies
Project managed and developed by: Becca Law
Cover and text design: me&him Design

Typeset in GT Walsheim Regular, Bold and Helvetica TT Regular, Oblique, Bold by **me&him Design**.

A catalogue record for this title is available from the British Library.
ISBN: 978-1-471-86382-0

Printed in the UK by
Ashford Colour Press Ltd

Acknowledgements

Thank you to the following people who contributed their ideas and support: Steve Connolly and Debbie Smith, Hodder Education; Ray Chambers, Uppingham Community College; Miles Berry, University of Roehampton; Roger Davies, Queen Elizabeth School; Clare Riley, Jeannette Wing, Thomas Ball, Peli de Halleux, Michal Moskal, Eric Anderson, Jonathan Protzenko, Steve Hodges, Deirdre Quarnstrom, Vardhani Mellacheruvu, Lori Ada Kilty, Darren Gehring, Genevieve Payzer, Gladys Hilerio, Kevin Chen, Michael Braun and Judith Bishop from Microsoft and the BBC team.

This content is licensed under
Attribution-NonCommercial-ShareAlike
CC BY-NC-SA

This licence lets others remix, tweak, and build upon your work non-commercially, as long as they credit you and license their new creations under the identical terms.

See - <https://creativecommons.org/licenses/by-nc-sa/4.0/>

Although every effort has been made to ensure that website addresses are correct at time of going to press, Microsoft cannot be held responsible for the content of any website mentioned. It is sometimes possible to find a relocated web page by typing in the address of the home page for a website in the URL window of your browser.

The *Make It Digital* initiative

In March 2015, the BBC launched *Make It Digital* – a major UK-wide initiative to inspire a new generation to get creative with coding, programming and digital technology.

The project aims to put digital creativity in the spotlight like never before, and to help build the nation's digital skills, through an ambitious range of new programmes, partnerships and projects.

These include:

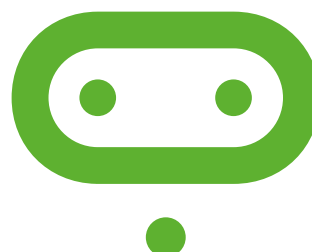
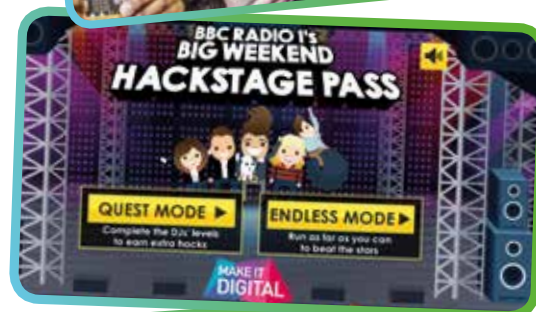
- A major partnership to develop and give a BBC micro:bit coding device to all Year 7 children across the UK, for free, to inspire a future generation.
- A season of programmes and online activity involving the BBC's biggest and best-loved brands, including Doctor Who, Eastenders, Radio 1, The One Show, Children in Need, BBC Weather and many more, including a new documentary on Bletchley Park.
- The *Make It Digital* Traineeship to create life-changing opportunities for up to 5,000 young unemployed people; the largest traineeship of its kind.
- Partnerships with around 50 major organisations across the UK.
- A range of formal education activities and events, including Bitesize, Live Lessons and School Report.

The BBC micro:bit initiative

Back in the 1980s, the BBC Micro was used extensively in primary and secondary schools and was instrumental in inspiring a generation of technology pioneers. Nowadays, computing and digital technology can be found everywhere, but the emphasis seems to have shifted from the creation of technology to the consumption of it.

As part of the *Make It Digital* initiative, the BBC has collaborated with over 25 organisations to create the BBC micro:bit, a personal programmable device, which will be provided, free of charge, to every child in Year 7 across the UK.

It provides an exciting and accessible introduction to coding on a simple hardware platform. Its purpose is to enthuse, excite and empower a new generation of digitally-creative young people. This is why the BBC micro:bits are specifically designed for young people themselves to own.



The BBC micro:bit device

The BBC micro:bit is a very simple computer. It is programmed by using another device (smartphone, tablet, PC, iPad etc.) to write the program, which is then compiled and downloaded onto the BBC micro:bit. The newly programmed BBC micro:bit can be disconnected and will run the program, just like other embedded devices, such as a digital watch, a GPS device or a pocket calculator.

The device has a display made up of 25 LEDs and some simple input controls that can be used in a number of ways. It is small enough to slip into a pocket or even wear.

The BBC micro:bit offers a gentle introduction to programming and making: switch on, program it to do something fun, wear it, customise it, and put new ideas into action. It can be programmed to show words or shapes, tell the time or play games.

It is designed to be a starting point to get young people interested in coding so they can move on to other, more sophisticated devices in future. The BBC micro:bit has an accelerometer, which can detect movement, and it can connect and communicate with other devices, including Arduino, Galileo and Raspberry Pi.

It offers a natural progression from screen-based programming using visual languages, and can lead on to more complex, text-based programming.

The BBC micro:bit also has Bluetooth Low Energy, allowing it to be part of the 'Internet of Things' – the extension of the internet beyond computers and smartphones to include other embedded systems, from fridges to cars, and even home central heating systems.

Supporting learning

The BBC and its partners recognise that a hands-on learning experience can help young people to grasp the computing curricula in ways that on-screen coding activities and traditional classroom learning cannot.

The BBC micro:bit can help learners to develop their understanding of physical technology and computing, offering the opportunity to apply complex thinking, analytical and problem-solving strategies.

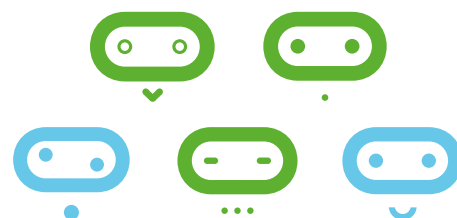
Inspirational content on BBC radio and television will raise awareness of the BBC micro:bit, while teachers, parents and young people will be encouraged and supported to get the most out of the device through a rich range of online resources and real-world events created by the BBC and partners.

Partnerships

More than 25 organisations have been involved in this pioneering partnership. See www.microbit.co.uk/partners for more information.

See section on Coding building blocks on **page 11** and Lessons on **pages 12–30**.

You can share links to tutorials and code with other BBC microbit users via the dedicated BBC micro:bit CAS forum at: computingatschool.org.uk/



The BBC micro:bit:

What is it designed to do?

The BBC micro:bit is a very simple computer. A computer is a machine that accepts input, processes this according to stored instructions and then produces output. All three of these elements are present on the BBC micro:bit's printed circuit board.

Front of board

LED

Coordinates start at (0,0) in top left-hand corner. In computing, displays start at the top left-hand corner so, in coding terms, this is (0,0). This is different from mathematics and graphs where (0,0) is the bottom left corner. It is important to note this is also relative, so if the screen rotates (0,0) is still the top left corner of the screen. See Lesson 4 for the use of coordinates in the Catch the egg game.

LED MATRIX

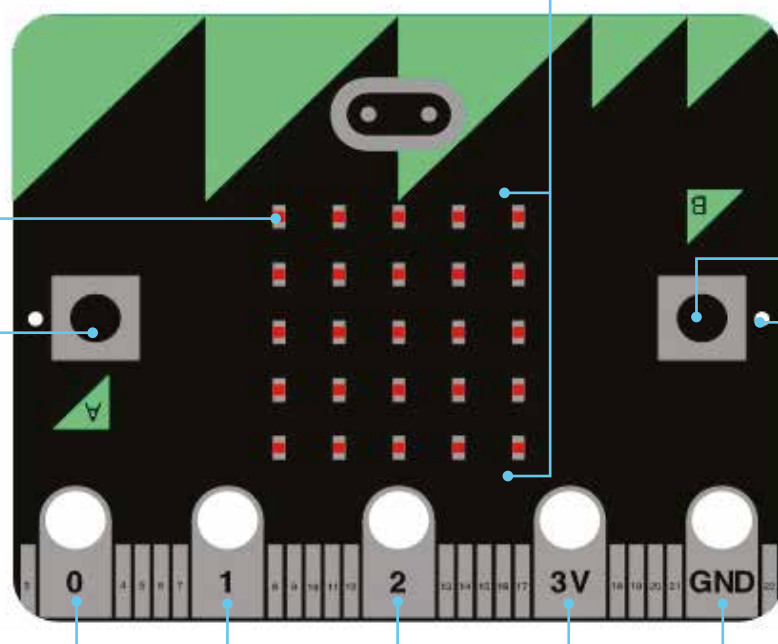
5 × 5 array of light emitting diodes (LEDs), which can each be set to on / off. The brightness of the set of LEDs as a whole can also be controlled.

BUTTON B

See Button A.

HOLES

Holes for sewing, mounting and hanging.



BUTTON A

A form of input. The BBC micro:bit detects when this button is being pressed. This is a push-to-make switch (pressing it completes an electrical circuit).

PINS P0, P1, P2

Pins for attaching external sensors, like thermometers or moisture detectors, and actuators, like turning a motor on, so students can build projects with them like a plant watering alarm. Can be either input or output and either digital or analogue.

3V AND GND

Enable a user to power an external device, like a motor, using the battery or USB. They also enable capacitive touch (using an object as a switch).

Back of board

BLUETOOTH LOW ENERGY ANTENNA

A messaging service, built for the Internet of Things, so devices can talk to each other. The BBC micro:bit will be a peripheral device and it can talk to a central device like a smartphone or tablet (or a laptop that has BLE). This means the BBC micro:bit can send signals to and receive signals from a central device. BLE will be used to 'flash' new programs onto the BBC micro:bit and to allow the BBC micro:bit to communicate with a computer or an internet connection.

USB PLUG

Programs can be downloaded from Windows and Macs onto the BBC micro:bit via a USB data connection. The USB connects the BBC micro:bit to a computer. This means the BBC micro:bit can send data to and receive data from the computer. The USB will be used to 'flash' new programs onto the BBC micro:bit and to allow the BBC micro:bit to communicate with a computer or an internet connection.

STATUS LED

Flashes yellow when the system wants to tell the user something has happened.

BUTTON R

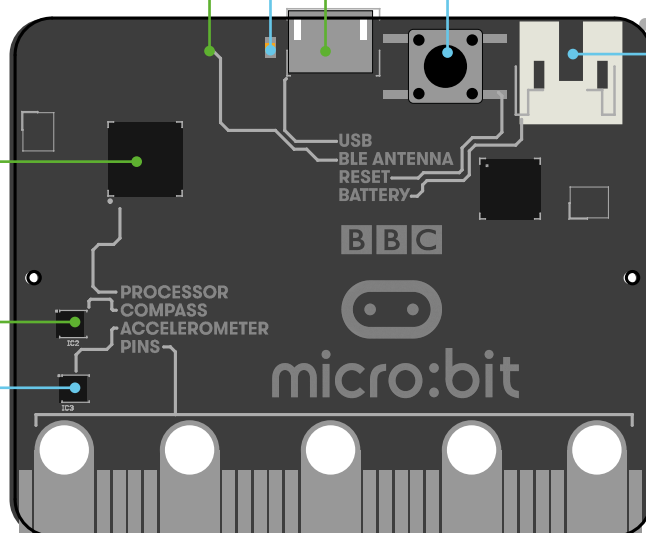
System button, which has various uses. Has to be pressed to 'flash' new code onto the device over BLE.

PROCESSOR

All the BBC micro:bit's programs and any data are stored on the small silicon-chip micro-controller. This tiny chip designed by ARM has 128kB flash memory and 16kB RAM memory; a tiny fraction of the memory on a smartphone.

BATTERY

This socket connects the external battery pack (containing two AAA batteries) to the board. The battery pack is attached physically to the board with a Velcro patch.



COMPASS

A sensor to detect magnetic fields, like the Earth's, allowing the direction of the BBC micro:bit to be determined and converted to a digital form that can be used in BBC micro:bit programs. Output from the compass is degrees.

ACCELEROMETER

Converts analogue information about how quickly the BBC micro:bit's speed changes to a digital form that can be used in BBC micro:bit programs. Output from the accelerometer is in milli-g. Allows the BBC micro:bit to be used to control movement of on-screen characters such as Kodu.

A note about ARM

ARM designs the processors for most mobile phones and embedded systems (such as smart thermostats, car engine controllers and the processors inside digital cameras), and was founded by members of the original BBC Micro team!

BBC micro:bit is based on ARM's mbed platform for embedded systems, but programming the BBC micro:bit is very straightforward.

A note about machine code

Machine code is the language the CPU (central processing unit) of a computer understands, but it isn't very readable by humans as it is made up of numbers.

Machine code is known as a low level language. High level languages, such as Microsoft Block Editor or Microsoft Touch Develop, are readable/understandable by humans. A program written in a high level language, like Microsoft Touch Develop, has to be compiled (translated) into machine code that the processor 'understands' (see page 10).

The BBC micro:bit website

The BBC micro:bit website www.microbit.co.uk is the starting point for learning about and programming on the BBC micro:bit. There is extensive help available on the site, including scaffolded, step-by-step tutorials for programming the BBC micro:bit.



01 Sign in

To sign in:

- enter your facilitator code provided (email BBCmicrobit@bbc.co.uk if you don't have one)
- authenticate your account by entering a username and password from an existing account (Facebook, Microsoft, Office 365)
- read and agree to the terms of use.

02 My Scripts

Click on My Scripts to:

- save and retrieve scripts
- compile code and 'flash' it to a BBC micro:bit
- publish scripts to the BBC micro:bit website
- share scripts with other BBC micro:bit users
- set up groups, with access codes, for your students to join.



For more information visit www.microbit.co.uk/help.

03 Create Code

Click here to choose and select an available code editor (see below) to start creating programs for the BBC micro:bit.

All code editors come with a BBC micro:bit simulator, so you can test ideas and code on screen without having to have a BBC micro:bit plugged into the computer to run the code you write. Two key editors are currently available to program the BBC micro:bit.

Microsoft Block Editor

This is a graphical, drag and drop code editor, where coding blocks snap together. It's quite similar to Scratch, which many students may have encountered in primary school. There's support for the main input and output functions of the BBC micro:bit, as well as standard programming constructs such as sequence, selection, repetition and variables.

It's easy to start a project in Microsoft Block Editor and then convert it to a Microsoft Touch Develop script.

Microsoft Touch Develop Editor

The Microsoft Touch Develop Editor sits between visual, block-based languages, such as Blockly, and traditional, text-based programming languages, such as Python. The editor is based on the Microsoft Touch Develop programming language and comes with a BBC micro:bit library of commands installed.

Like other text-based programming languages, Microsoft Touch Develop provides a great deal of flexibility: as well as supporting input, output, sequence, selection, repetition and variables, there's also support for user-defined functions, making this a good choice for developing the ideas of decomposition and abstraction.

Other code editors

A number of other code editors will be available from the Autumn term 2015. See www.microbit.co.uk/create-code for up-to-date information on which code editors are available.

04 Discover

Use this drop-down menu to explore the BBC micro:bit website, including tutorials and projects.

Tutorials

The BBC micro:bit site offers different types of coding tutorials. Some tutorials are interactive and lead you through the creation of a program step-by-step with on-screen tips. Others present guided challenges with fewer instructions. Here we provide support by signposting key instructions and routes through projects.

Projects

This is a bank of BBC micro:bit projects created by other coders for you to explore, use and adapt. Some of these projects have been created by the BBC and BBC micro:bit partners, but most will be written by young people themselves and other BBC micro:bit users. These are a great starting point for seeing just what the BBC micro:bit can do, as well as learning how to program it. It's much easier to take someone else's program and edit it to make it work a little (or a lot) differently, than having to start programming from a blank screen.

06 About

Everything you could want to know about your BBC micro:bit and its features.

07 Getting Started

These are video and step-by-step guides for getting started with the BBC micro:bit. These videos and guides walk you through the process of starting to write some code, including switching between the different code editors available, saving projects, installing the loader software on your computer, connecting the BBC micro:bit and uploading code to it via the loader. This content will be continually updated.

08 Teachers and Parents

This introduces the BBC micro:bit in the context of its use in the classroom and at home. It contains useful information for anyone supporting children on their BBC micro:bit journey.

05 Search

Search through learning resources, videos, tutorials and more.

09 Help

Here you will find frequently asked questions and where to go for additional support.

Getting started with the Microsoft Touch Develop Editor

Once you've signed in to the BBC micro:bit website (see details on page 6), you can get started with coding!

01

Type www.microbit.co.uk into your web browser.



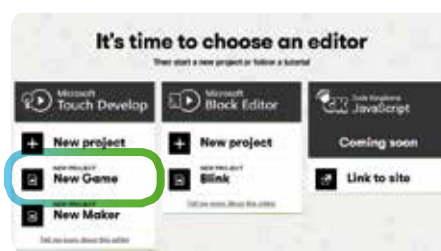
02

Click **Create Code**.



03

In the Microsoft Touch Develop section, click the second link down.



04

Have a go at clicking some of the buttons on screen to see what they do.

CODING AREA

This is where all your coding takes place.

RUN

Click to run a program. The simulator on the right-hand side of the screen will show the code in action.

UNDO

Click to undo any changes to your code.

MY SCRIPTS

Click to return to any previous scripts you've written in the Microsoft Touch Develop Editor.

COMPILE

Click to compile your program to allow it to run on the BBC micro:bit (see page 10 for more information about the compilation process).

SCRIPT

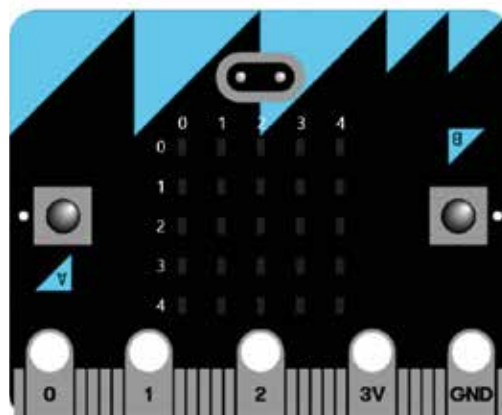
Click to display all the functions you're working with, any libraries you're using as part of your code (these are sets of functions developed by other people that you can use in your script) and any global data. The button isn't shown when this list is already visible.



```
function main ()
  basic -> forever do
    led -> plot(2, 2)
    basic -> pause(200)
    basic -> clear screen
    basic -> pause(200)
  end
end function
```

RAD BLINK

BBC micro:bit



05

Click on the `led->plot(2, 2)` instruction that's already in the script. A code keyboard will appear at the bottom of the screen.

+

Click to add instructions above or below the selected line.

CUT / COPY

Click to cut or copy selected code.

SIMULATOR

All BBC micro:bit code editors include a simulator, which shows how your program will execute.

This means you can:

- start writing code for the BBC micro:bit even if you don't have the actual device
- test programs on the simulator to ensure they work before downloading the code to the BBC micro:bit.

PASTE

Click to paste in cut or copied code.

ALL COMMANDS

Search for appropriate code/ functions to add to your programs.

CODE KEYBOARD

The keyboard makes it easy to edit your code on a touch screen device or just using a mouse. If you're working with a normal keyboard, you can enter language commands by just typing, and you'll see the possible command completions appear automatically.



Where next?

Use Microsoft Touch Develop

Get coding using the lessons on pages 12–30 in this book. Once you've completed all four, visit www.microbit.co.uk/start-guide/certificate to pick up your certificate!

There's more to Microsoft Touch Develop than the BBC micro:bit! Visit touchdevelop.com to make games and apps and stretch your digital creativity!

Try other code editors

A number of different code editors are available to program the BBC micro:bit. All of these can be found online at www.microbit.co.uk:

Code Kingdoms JavaScript Editor

A visual editor that uses introductory chunks of JavaScript. You can move to text-based input as your skills progress.

MicroPython Editor

Hack your BBC micro:bit with Python, an easy-to-learn programming language for everyone, from kids to teachers to professional software engineers.

Kodu Game Lab

Use the BBC micro:bit to control Kodu characters, display text and animations, interface with the real world, and much more!
www.kodugamelab.com/bbc-microbit/

Samsung Android App

An app for on-the-go BBC micro:bit programming. It will also enable BBC micro:bits to talk to and control phones and tablets! Teacher resources coming soon.

Uploading programs to the BBC micro:bit

How does my program get onto the BBC micro:bit?

For your program to work on the BBC micro:bit, first it has to be compiled. Compiling means to translate a program into a more efficient computer language.

When you hit the compile button on the Microsoft Touch Develop Editor interface, your program is actually compiled twice.

First, your program is translated into a C++ program. C++ is a very popular language for programming software systems, both large (like Microsoft Windows) and small (like the BBC micro:bit).

Second, the C++ program is translated into a binary file that contains the machine code in the instruction set used by the ARM processor that is on your BBC micro:bit. Compiling to C++ actually happens in the web browser itself, and then the C++ code is sent over the internet to a server (at developer.mbed.org) which compiles the C++ code to the ARM machine code (the hex file), which then gets sent back to your browser. When you drag the hex file over to the drive for your BBC micro:bit, your ARM binary program is installed and begins to run.

The BBC micro:bit hardware is built using ARM's open source mbed platform. This means that as well as using Microsoft Touch Develop and the other editors on the BBC micro:bit site, it is possible for more confident coders to program the BBC micro:bit using industry-standard development tools, including ARM's online C++ compiler at developer.mbed.org.

Getting your programs onto the BBC micro:bit

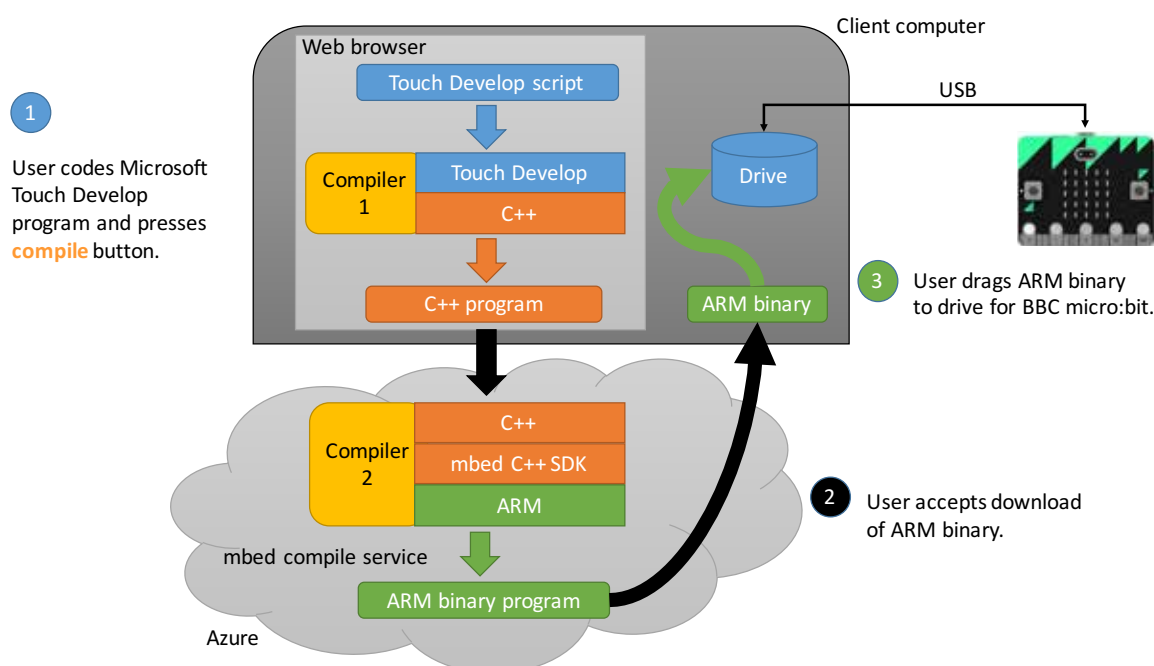
The last stage – getting your program onto the BBC micro:bit itself – is quite easy:

- Hit the compile button in the code editor. A .hex file will be created.
- Plug the BBC micro:bit in to your computer's USB port using a standard micro USB cable (supplied). The BBC micro:bit should show up as a USB storage device.
- Drag the .hex file onto the drive that corresponds to the BBC micro:bit. Once the system LED has stopped flashing, press the reset button on the back of the BBC micro:bit to start the program.

Once a program is uploaded to the BBC micro:bit, the device can be unplugged and will run independently, as long as the user has attached a battery pack.



The BBC micro:bit compiling process



Coding building blocks

It's easy to get started with coding on the BBC micro:bit. The images below show the code you need (both in the Microsoft Block and Microsoft Touch Develop Editors) to make your BBC micro:bit do simple things. These could be used to kick off your first BBC micro:bit coding sessions with your students and can also be used in more complex projects.

Activity

Scrolling text

Code in Microsoft Block

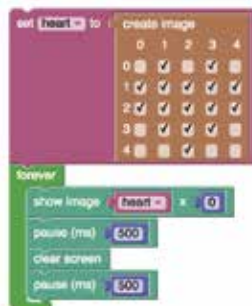


Code in Microsoft Touch Develop

Online tutorial:
www.microbit.co.uk/td/tutorials/scroll-text

```
function main ()
  basic ← forever do
    basic ← show string("Hello World!", 100)
  end
end function
```

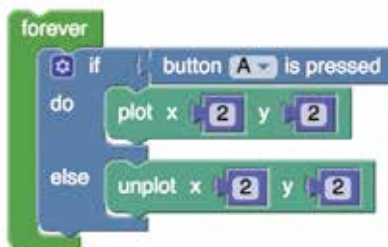
Flashing heart image



Online tutorial:
www.microbit.co.uk/td/tutorials/flashing-heart

```
function main ()
  var heart := image ← create image(♥)
  basic ← forever do
    heart ← show image(0)
    basic ← pause(500)
    basic ← clear screen
    basic ← pause(500)
  end
end function
```

Press Button A to turn on a light



Online tutorial:
www.microbit.co.uk/td/tutorials/button-light

```
function main ()
  basic ← forever do
    If input = button is pressed("A") then
      led ← plot(2, 2)
    else
      led ← unplot(2, 2)
    end if
  end
  do nothing
end function
```


Lesson 1: Digital key chain

Programming a Minecraft Creeper face using the image editor within Microsoft Touch Develop

Outcome

Display of a Creeper face (similar to the character seen in Minecraft) on the BBC micro:bit LED display:

- By default, all of the LEDs (or lights) are off.
- There will be a single state (Minecraft Creeper face).
- The image will turn off after 3 seconds.

Tutorials

For a guided tutorial go to

www.microbit.co.uk/td/tutorials/digital-key-chain

Decomposing the problem

This lesson can be decomposed into four parts:

1. Design how our single state will look (which LEDs will be switched on to display our Creeper face).
2. Use the image editor to turn on the required LEDs.
3. Create a timer to pause the image for 3 seconds.
4. Reset the display to its original state: OFF.



Design how each **state** will look

Before we start to code, we need to plan what our single state will look like.



01

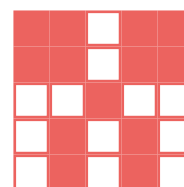
Draw a 5x5 grid and colour in the boxes to show what the Minecraft Creeper face will look like.

You don't have to program a Creeper face. The image could be anything you like.

Key

LED on =

LED off =



CREEPER FACE

Why not try out different images?



Use the image editor to **turn on** the required LEDs

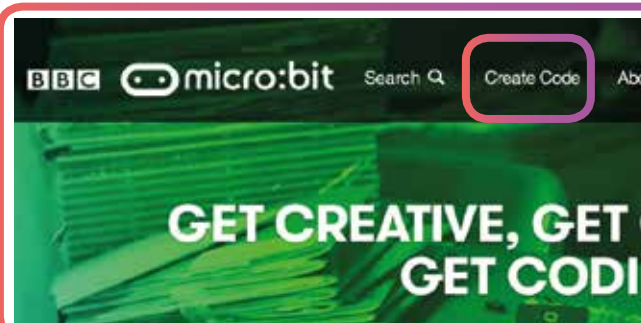
We need to specify which LEDs will be ON to display the Creeper face.



02

Start by opening a new browser window and typing www.microbit.co.uk in the address bar.

Click on **Create Code**. In **Microsoft Block Editor**, click **New project**. Type in a name for your script, such as **Creeper**. Click on **create**.



03

A blank coding environment will appear (see picture to the right).

Select the **Images** button from the menu on the left.



04

The **Images** section includes blocks that control the creation and display of an image on the BBC micro:bit through LEDs.

Select the **show image** block.



05

You will notice that an offset value of **0** is displayed.

Changing this allows you to show your image in different positions on the BBC micro:bit display.



06

We now want to select which LEDs will be ON for our Creeper face.

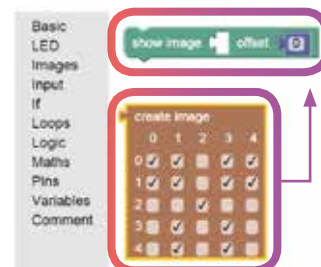
Select the **Images** button then the **create image** block. Tick the boxes in the block to make the shape of the Creeper face, as shown in the image.



07

Drag the **create image** block into the empty position on the **show image** block.

This will make sure that the Creeper face appears when you press the **run** button.



It's important that we test our programs regularly. This allows us to debug the program and fix any errors.



08

Press the **run** button to test your program. What does it look like on the simulator? If it doesn't work as expected, go back and try to find and correct the problem.



Create a timer to **pause** for 3 seconds

To display the Creeper face for a short period of time, we need to add a timer.

09

From the **Basic** menu, select the **pause** block. The BBC micro:bit uses milliseconds as input, so 1000 is equivalent to 1 second.

We want to pause for 3 seconds, so change the number to **3000**.

Drag the block upwards so it snaps into place below the **show image** block.



Reset the display to its original state: **OFF**

To finish our program, we're going to turn all of the LEDs off. This will help to prolong the battery life of the BBC micro:bit.

10

Click on the **LED** menu. Select the **clear screen** block and snap it under the **pause** block. This will make sure that all of the LEDs are turned off after the Creeper face has displayed for 3 seconds.



11

You should now have a finished program which will display a Creeper face!



Do your own thing!

- Change the pattern in the **create image** block to show your own design.
- Instead of clearing the display, add another **show image** and **pause** block to create a simple two-state animation. Can you experiment with the brightness of the Creeper image between face changes?

A solution for the complete digital key chain code can be found on page 31. The working code can be found at www.microbit.co.uk/start-guide/solutions/digital-key-chain.

Lesson 2: Rock, paper, scissors game

Programming a game of 'rock, paper, scissors' using the accelerometer in Microsoft Touch Develop

Outcome

A simple 'rock, paper, scissors' game. Each time you shake the BBC micro:bit, it will randomly choose one of three shapes: rock, paper or scissors.

Tutorials

For a guided tutorial, go to www.microbit.co.uk/td/lessons/rock-paper-scissors/tutorial

Decomposing the problem

This lesson can be decomposed into five parts:

1. Design how each shape will look.
2. Create an image that contains three frames; one for each of the three shapes.
3. Randomly generate an **offset** that will choose which of these three frames is displayed.
4. Display the image on the BBC micro:bit.
5. Create an **event handler** to detect when the BBC micro:bit is shaken.



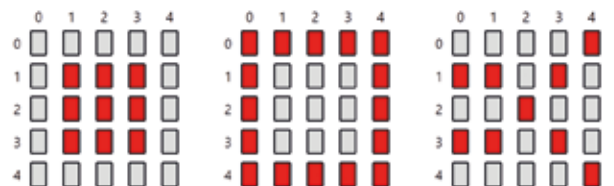
Design the shapes

Before we start to code, we need to plan what the rock, paper and scissors images will look like.



01

Draw three 5x5 grids and colour in the boxes (representing LEDs) to show what the rock, paper and scissors images will look like.



Use the image editor to create an image with **three frames**

We need to specify which LEDs will be ON for each of our three frames.



02

Open a new browser window and type www.microbit.co.uk in the address bar. Click on **Create Code**. In **Microsoft Touch Develop**, click **New project**. Type in a name for your script, such as **rock paper scissors**. Click on **create**.



03

We need to start by creating an image that contains three frames: rock, paper and scissors. Click on **do nothing** and then select **image**, followed by **create image**.

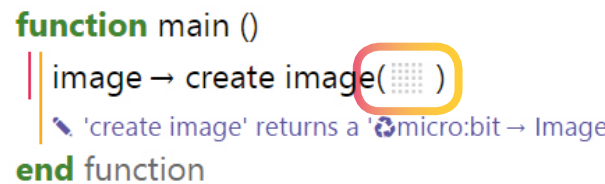


image -- Extensions


 create image

04

Next, select the 5x5 grid (representing the BBC micro:bit LEDs) to create each frame. This will bring up a visual editor, which allows you to control which of the LEDs you want to turn on for a given frame.

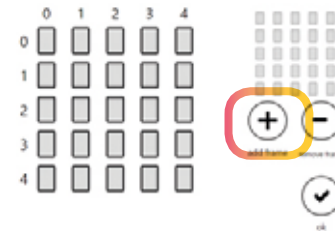
function main ()


 image → create image()
'create image' returns a  micro:bit → Image

end function

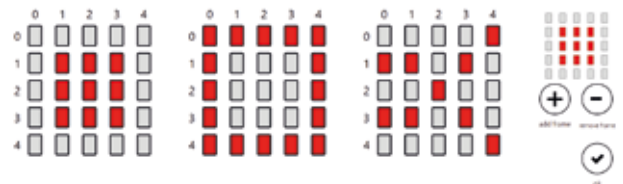
05

Add two more frames to the editor by clicking on **add frame** twice.



06

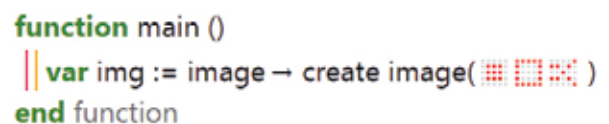

Now, click on the LED boxes corresponding to those you coloured in for step 1, so that each frame represents one of the three shapes. Click on the **ok** button when you have finished.



07

Now we need to store this image in a variable. Click **store in variable** from the keyboard

function main ()


 var img := image → create image()

end function

Using offset

Offset allows us to specify the number of pixels the image is shifted to the right or left on the BBC micro:bit display. We will use offset to choose the frame in our img variable that we want to show.



08

The BBC micro:bit will show one frame of our image at a time by displaying the image with an offset of either 0, 5, or 10. (An offset of 0 will display the first image, 5 the second and 10 the last.)

We want the BBC micro:bit to randomly choose one of these offset values when we run the script.

Add a new line of code by clicking the **+** button. Then click **math** followed by **random**. By default, the random number generator has a limit of 2. This means that it will randomly generate a number between 0 and 1. We need to generate three random numbers, so to fix this, change the **2** inside the brackets to a **3**.

```
function main ()
  var img := image → create image( [img icon] )
  math → random(3)
  ⓘ 'random' returns a 'Number'; tap 'store in var' to store
end function
```

09

We have created code to generate a random number between 0 and 2. But we actually need our BBC micro:bit to randomly generate either 0, 5, or 10 (to match our offset values). We can resolve this by multiplying the randomly generated number (between 0 and 2) by 5 (resulting in either 0, 5 or 10). Use the keyboard to insert ***** and then type **5** (as shown).

```
function main ()
  var img := image → create image( [img icon] )
  math → random(3) * 5
  ⓘ '*' returns a 'Number'; tap 'store in var' to store it in a
end function
```

10

We have created a random number that is either 0, 5, or 10. However, the value will be lost if we don't do anything with it. Let's store it in a variable that's called **offset**. Click **store in variable** from the keyboard. Rename **x** to **offset**.

```
function main ()
  var img := image → create image( [img icon] )
  var offset := math → random(3) * 5
end function
```

11

Now we need to display the image with the appropriate offset on the BBC micro:bit. Add a new line of code by pressing the **+** button. Then, on the keyboard click **img** and then **show image**. This will display the image on the BBC micro:bit.

```
function main ()
  var img := image → create image( [img icon] )
  var offset := math → random(3) * 5
  img → show image(0)
end function
```

12

By default, the offset of the **show image** is **0**, as indicated by the number in the brackets. However, we want to replace the **0** with the variable **offset** that we generated in steps 8–11. Delete the **0**, and then select the **offset** variable from the keyboard below.

```
function main ()
  var img := image → create image( [img icon] )
  var offset := math → random(3) * 5
  ⓘ img → show image(offset)
end function
```



The on-shake event



This event is triggered when the BBC micro:bit is shaken vigorously. The BBC micro:bit detects the shaking movement through its **accelerometer**.

13

Start by adding an **on shake** condition to detect when you shake the BBC micro:bit. Add a new line of code, then click on **input** followed by **on shake**. (You may need to click on **more** to display the **input** option.)

input -- Extensions

→ on logo down

→ on shake

→ on fall

→ on lo up

14

You have created the event handler, but nothing will happen yet when the BBC micro:bit is shaken. We need to fix this by moving all the code we have created so far into the **on shake** condition. Begin by selecting the first line of code and then clicking **cut**.

```
function main ()
  var img := image → create image(
  var offset := math → random(3) * 5
  img → show image(offset)
  input → on shake do
    do nothing
  end
end function
```

15

Select the **do nothing** line inside the **on shake** event handler, and then click **paste**. Cut and paste the remaining two lines of code so that your screen looks like the image on the right.

```
function main ()
  input → on shake do
    var img := image → create image(
    var offset := math → random(3) * 5
    img → show image(offset)
  end
end function
```

Run your program

See how many times you can win against the BBC micro:bit. Challenge your students to see how many times they can win against the BBC micro:bit.



Do it yourself!

Can you program the BBC micro:bit to track the scores of your games with the BBC micro:bit?

- You will first need to create a variable for the times you win against the BBC micro:bit.
- Add code so that your score goes up each time you press Button A.
- Add code to display the score on the BBC micro:bit.
- Repeat the previous step for when you lose against the BBC micro:bit. Add code to reduce your score each time you press Button B, then display the score after either button is pressed.



A solution for the complete rock, paper, scissors game code can be found on page 31. The solution to these challenges can be found at www.microbit.co.uk/td/lessons/rock-paper-scissors/challenges.

Lesson 3: Digital pet

Programming an animated pet using variables and functions

Outcome

A digital pet (similar to Tamagotchis from the 1990s) with different states that can be controlled by pressing Buttons A and B (our input). The idea is that our digital pet needs to be taken care of.

- The default state of the pet is **AWAKE**.
- Button A will stroke the pet, causing it to fall **ASLEEP**.
- Button B will feed the pet, so it is **EATING**.

Tutorials

For a guided tutorial go to www.microbit.co.uk/td/lessons/digital-pet/tutorial

Decomposing the problem

This lesson can be decomposed into four parts:

1. Design how each state will look (which LEDs will be switched on).
2. Create a function that tells our BBC micro:bit which LEDs to turn on for each state.
3. Create a **forever loop** to continually update the state.
4. Create conditional statements to specify which function to run if a particular button is pressed, e.g. if Button A pressed then go to **ASLEEP** state.

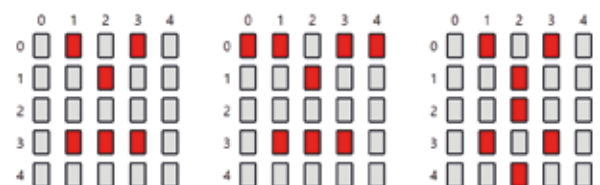
Design how each **state** will look

Before we start to code, we need to plan out what our pet will look like when it is **AWAKE**, **ASLEEP** or **EATING**.



01

Draw a 5x5 grid and colour in the boxes (representing LEDs) to show what your pet will look like at different times, for example: **AWAKE**, **ASLEEP**, **EATING** (as shown on the right).



02

Start by opening a new browser window and typing www.microbit.co.uk in the address bar. Click on [Create Code](#). In [Microsoft Touch Develop](#), click [New project](#). Type in a name for your script, such as **Digital pet**. Click on [create](#).



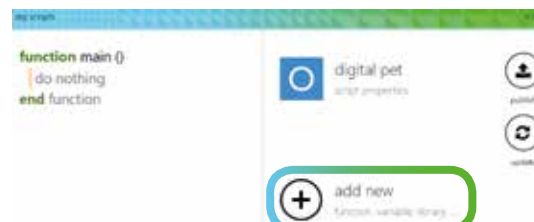
Create a function for each **state**

We're going to start by programming a function for the different states of our digital pet (e.g. which LEDs are ON and which are OFF for each state). We're going to do this first because we will want to call on these functions later, without having to leave the main process.



03

To create the first function (for our **AWAKE** face), select the **script** button in the top right of the screen. Click on the **add new (+)** button to bring up your resource menu.



04

Select **function** from the menu. This will bring you to a new coding area (as shown on the right). Notice that it will say **do stuff** at the top. This is the current name of the function. Click on **do stuff** and rename your function to **set awake**. Click **ok**.

```
function do stuff ()
| do nothing
end function
```

05

Now that we have named our function, we need to specify which LEDs should be turned on for the **AWAKE** state. We can do this using the image editor. Click **do nothing** in the coding area. A keyboard will pop up at the bottom of the screen. Select **image**. Notice how the keyboard changes. Select **create image**.

```
function set awake ()
| do nothing
end function
```

06

Click on the 5x5 grid within the brackets. You will notice that none of the LEDs are currently lit. Select the LEDs that you want to be on when your digital pet is in the **AWAKE** state. When you have finished, click on **ok** at the bottom right of the pane.

```
function set awake ()
| image → create image( )
| 'create image' returns a micro:bit →
| property on it
end function
```

07

You'll notice an error message below your line of code. The BBC micro:bit has created an image, but the image will be lost if we don't do anything with it. Let's store this image in a variable and name it **img**. Click on **store in var** at the bottom to store this image.

```
function set awake ()
| var img := image → create image( )
end function
```

08

To show this image on the BBC micro:bit, click the **+** button to create a new line of code. Click on the image variable name **img**.



09

To display this image, click on **show image**.

```
function set awake ()
  var img := image → create image( )
  img → show image(0)
end function
```

10

We have created a function to display our digital pet in its **AWAKE** state. However, this function isn't yet 'called' by the main method. Go back to the main method by clicking on **script** followed by **main**. Click on **do nothing**, then **code**, followed by **set awake**.

```
function main ()
  set awake
end function
```

Test your code

Try testing your code at this point. You will see the **AWAKE** state after pressing the **reset** button on the BBC micro:bit. However, there are no ways to interact with this pet. Let's fix this!



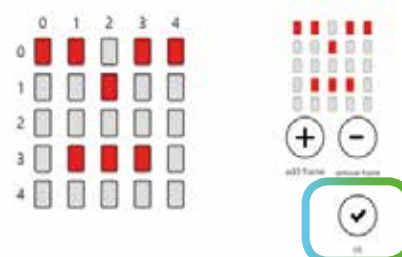
11

Now let's create another function to put our digital pet to sleep. Click on **script**, followed by **add new**, and then **function**. Rename the script to **set sleep** using the same method as you did for **set awake**.

```
function set sleep ()
  do nothing
end function
```

12

Just as you did for **set awake**, create an image for the sleep mode and store it in a variable named **img**. You can do this by clicking on **image** and then **create image**. Next, click on the 5x5 grid. Plot the LEDs as shown, and click on **ok** when you are finished. Finally, click on **store in var** at the bottom.



13

After you have stored the picture in a variable, don't forget to display it as well! To do this, click on the **+** button to insert a new line of code and select **img** followed by **show image**.

```
function set sleep ()
  var img := image → create image( )
  img → show image(0)
end function
```

Create a **forever loop** to update the display regularly

We now want to create some code that makes the display update regularly. We can do this by using a **forever loop**.



14

Click on **script** and **main** to return to our main method. To create a forever loop so that our program will continuously update the state of our digital pet, click on the **+** button to add a new line of code, then click on **basic**, followed by **forever**.

```
function main ()
  ▷ set awake
  basic → forever do
    do nothing
  end
end function
```

15

We now need to add an **IF** statement to specify which function to run when Button A is pressed. When Button A is pressed, we want to display the **ASLEEP** state. Click **do nothing**, and then **if**. Now, we need to add a condition within the **IF** statement. We want to know if Button A is pressed, so click **input**, **button is pressed** followed by **"A"**.

```
function main ()
  ▷ set awake
  basic → forever do
    if input → button is pressed("A") then
      do nothing
    else do nothing end if
  end
end function
```

16

Now we need to display the **SLEEP** state when Button A is pressed. Inside the **IF** statement, call the function, **set sleep**. You can do this by clicking on **do nothing** below the **IF** statement, then **code**, followed by **set sleep**.

```
function main ()
  ▷ set awake
  basic → forever do
    if input → button is pressed("A") then
      ▷ set sleep
    else do nothing end if
  end
end function
```

17

At the moment, our digital pet will be set to sleep for only a split second because the forever loop executes the code very quickly. So, let's add a pause of 5000 milliseconds (5 seconds) after our pet is set to sleep. To do this, add a new line of code by clicking the **+** button, then select **basic**, followed by **pause**. Change **100** to **5000**.

```
function main ()
  ▷ set awake
  basic → forever do
    if input → button is pressed("A") then
      ▷ set sleep
      basic → pause(5000)
    else do nothing end if
  end
end function
```

18

We now have code that specifies what happens when Button A is pressed. But what happens if Button A is not pressed? We want the default state for our pet to be **AWAKE** so we need to move **set awake** to inside the **ELSE** statement, which is found below the **IF** statement. To do this, select **set awake** and then click on the **cut** button.



```
function main ()
  nothing
  ▷ set awake
  basic → forever do
    if input → button is pressed("A") then
      ▷ set sleep
      basic → pause(5000)
    else do nothing end if
  end
```

19

Now, select **do nothing** inside of the **ELSE** statement, and select **paste**.

You have now successfully completed your digital pet!

```
function main ()
  basic → forever do
    if input → button is pressed("A") then
      ▷ set sleep
      basic → pause(5000)
    else
      ▷ set awake
    end if
  end
end function
```

Do it yourself!

- Can you add another function called **set eat** so that you can feed your pet?
- Try to feed your pet using Button B.
- Can you add code so your pet tells you when it is going to sleep?
- Can you keep track of how many times you feed your pet, and display that number when the BBC micro:bit is shaken?



A solution for the complete digital pet code can be found on page 31. The solution to these challenges can be found at www.microbit.co.uk/td/lessons/digital-pet/challenges

Lesson 4: Catch the egg game

Programming a game of 'catch the egg' using the accelerometer in Microsoft Touch Develop

Outcome

A 'catch the egg' game in which an egg (represented by a single LED) 'falls' from the top of the BBC micro:bit display and can be caught in a moveable basket at the bottom of the display. The script includes code for the accelerometer, which allows a user to control the position of the basket when the device is tilted:

- By default, the first 'egg' LED starts to drop from the centre of the top line of the display.
- The subsequent 'eggs' will then fall from random positions at the top of the display.
- The 'basket' will be moved by tilting the BBC micro:bit.

Tutorials

For a guided tutorial go to www.microbit.co.uk/tutorials/lessons/catch-the-egg-game/tutorial

Decomposing the problem

This lesson can be decomposed into seven parts:

1. Create the **global variables** for the game.
2. Assign initial values to each of the global variables.
3. Plot the starting positions of the LEDs.
4. Create a **forever loop** to update the display regularly.
5. Get the 'egg' to drop down the LED display.
6. Change the position of the basket using the accelerometer functionality.
7. Use **IF** conditions to check the final position of the egg.

Create the **global variables** for the game

Global variables are different from local variables (which only work inside a single loop). Global variables are accessible from any part of our program.



01

Start by opening a new browser window and typing www.microbit.co.uk in the address bar. Click on **Create Code**. In **Microsoft Touch Develop**, click **New project**. Type in a name for your script, such as **Catch the egg**. Click on **create**.



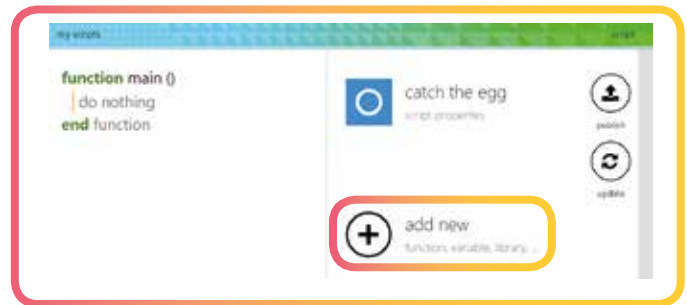
02

We're going to start by creating a number of **global variables** that will be accessible from any part of our program. To begin, click on the **script** button in the top right-hand corner.



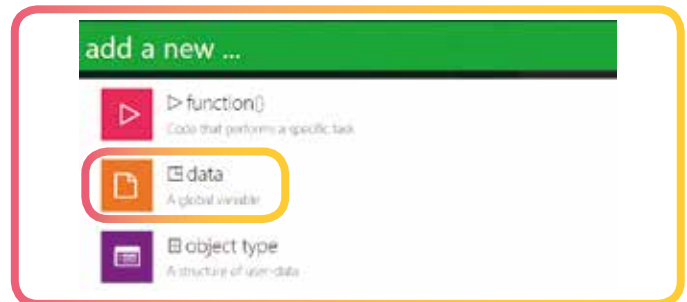
03

A menu will pop up, which allows you to add other features to your program. In this case, we're going to add in a global variable by clicking on the **+** button.



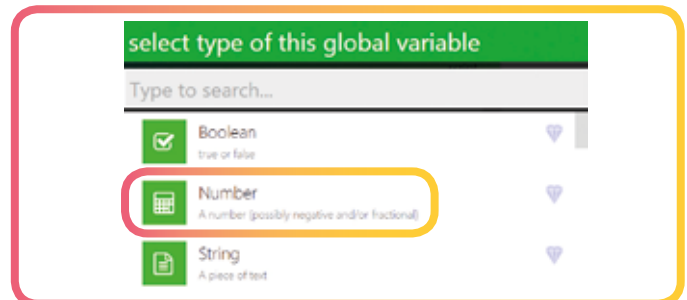
04

Select the **data** button from the menu that appears.



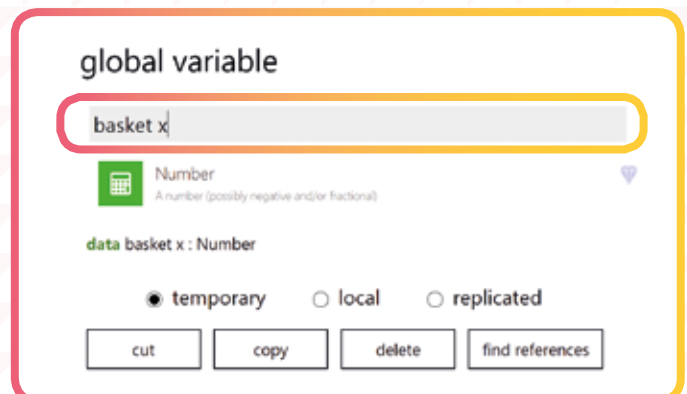
05

Now it's time to select a data type. For this program we're going to use **Number** for our variables. This is because we're going to be using x and y coordinates to designate the position of our basket to catch the falling eggs.



06

We're going to start by creating a variable for the x-position of the basket we will use to catch the egg. Type in **basket x**, then click the **ok** button. You should see your **basket x** variable in **vars**.



Naming variables

Explain, or remind students, to be as descriptive as possible when naming variables (e.g. score, timer, etc.) rather than using generic names (e.g. variable 1, variable 2, etc.). It's much easier to find and fix problems with variables when you can easily work out which one isn't working.



07

Repeat steps 2–6 to create all the variables for your game. You will need to create three variables in total: **basket x**, **egg x** (to control the horizontal position of the egg) and **egg y** (to control the vertical position of the egg).

Once you have finished setting up each of the variables, you should have something which looks like the image on the right.

vars



Assign initial values to each of the **global variables**

As with any programming language, when you declare your variables, you need to set them to a value. This value can be manipulated and changed later on.



08

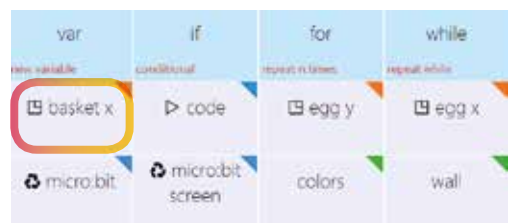
Return to your main function. Click **do nothing** below the **main** function, then select the **data** button from the keyboard.

function main ()

|| do nothing
end function

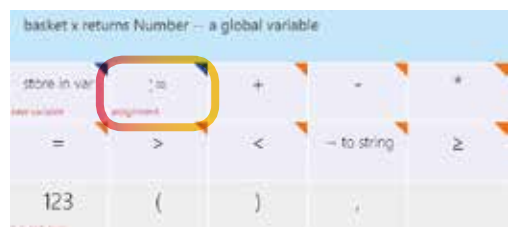
09

Select the variable **basket x** to begin with. Remember: **basket x** controls the x-position of the basket.



10

To assign a value to the **basket x** variable, select the **assignment** (**:=**) button from the keyboard. We want the basket to sit in the centre of the bottom row of the display, so type in **2**. Click on the **+** button below the **basket x** variable to add lines for the other two variables.



11

Assign values to the remaining two variables by following the step above. We want the egg to start falling from the top centre of the display at the beginning of our program, so set **egg x** to **2** (middle) and **egg y** to **0** (top).



Plot the starting positions of the LEDs

All LEDs on the BBC micro:bit display are OFF by default. We're going to program the BBC micro:bit to plot our first lights.



12

Click the **+** button to add another line of code below your assigned variables. Select **led** from the keyboard and then select the **plot** button (on screen 2 of the keyboard). You should now have something which looks like the image shown on the right.

```
function main ()
  basket x := 2
  egg x := 2
  egg y := 0
  led → plot(0, 0)
end function
```

13

The code shown above will only turn on a LED at position (0,0). We need to remove each of these values and replace them with our **egg x** and **egg y** value. Delete the first **0** and replace it with **egg x**. Repeat this process to select **egg y** for the second **0**.

```
function main ()
  basket x := 2
  egg x := 2
  egg y := 0
  led → plot(egg x, egg y)
end function
```

14

We now need to plot the starting position of the basket (**basket x**). Click the **+** button to add a new line of code. Then select **led** from the keyboard and then select the **plot** button. Change the first **0** to the **basket x** variable and the second **0** to **4**.

```
function main ()
  basket x := 2
  egg x := 2
  egg y := 0
  led → plot(egg x, egg y)
  led → plot(basket x, 4)
end function
```

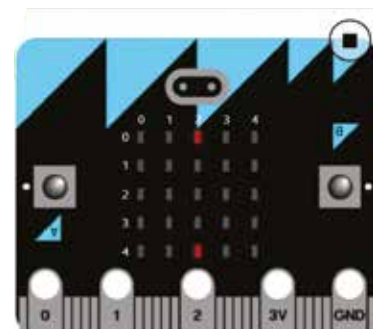


Test your code

It's important to regularly test that our code is working correctly so we can debug any errors. Regularly remind students of the importance of testing.

15

At this point in the program we're going to run the script by selecting the **run** button. You should see something similar to the simulator on the right when you run your program.



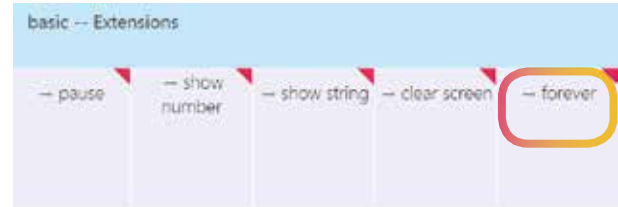


Create a **forever loop** to update the display regularly

Our next section of code requires you to get the display to update regularly. We do this by using a **forever loop**. This is to make sure that the program is always running. If we didn't use a **forever loop**, then we would have to write lots of lines of code to simulate our outcome.

16

Add a new line of code by pressing the **+** button. Select **basic**, and then **forever**.



Get the 'egg' to drop down the LED display

We now want to get the egg to look like it is dropping down the display.



17

Before we can light up the second LED (below the first), we need to unplot the original LED. This will ensure a smooth transition from one lit LED to another – to create the impression the egg is falling. Select **led** from the keyboard and **unplot**. Unplot both your basket position (**basket x**) and your egg position (**egg x**, **egg y**), as shown in the picture.

```
basic → forever do
  | led → unplot(□ basket x, 4)
  | led → unplot(□ egg x, □ egg y)
end
```

18

To get the egg to move down the display, we need to change the vertical position of the egg (**egg y**). We can do this by adding 1 to the value of **egg y**, each second. Select **egg y** and assign (**:=**) the variable to **egg y + 1**.

```
basic → forever do
  | led → unplot(□ basket x, 4)
  | led → unplot(□ egg x, □ egg y)
  | □ egg y := □ egg y + 1
end
```

19

You now need to plot the egg in its new position, using **plot** as we have previously (see step 12). We will need to slow down the movement of the egg. Select **basic** and **pause**, and input **300** milliseconds. This will allow you to see the lights fall down the screen at a slower pace.

```
basic → forever do
  | led → unplot(□ basket x, 4)
  | led → unplot(□ egg x, □ egg y)
  | □ egg y := □ egg y + 1
  | led → plot(□ egg x, □ egg y)
  | basic → pause(300)
end
```



Run your program, you should notice the egg fall down the board.



Change the position of the basket using the **accelerometer** functionality

It's now time to change the position of the basket. You can do this by using the **accelerometer** within the BBC micro:bit. You may have used an accelerometer in your smartphone when playing games previously.

20

We're going to add the next line of code above the **pause**. Select **input** from the keyboard. To use the accelerometer you will need to select the **acceleration** button. Now select **store in var**. The program sets the acceleration to left and right. This is the default position ("**x**"); you can also control up and down ("**y**"). You should have something like this:

```
egg y := egg y + 1
led → plot(egg x, egg y)
var acc x := input → acceleration("x")
basic → pause(300)
```

21

We now need to work out the position of the accelerometer and then turn the LEDs on. Select the **basket x** variable from the **data** menu in the keyboard. Select the **assignment** (**:=**) button.

```
add egg y := egg y + 1
led → plot(egg x, egg y)
+ acc x := input → acceleration("x")
basket x :=
  the operator needs something after it (ID154)
+ basic → pause(300)
  nothing
```

22

Type in number **2**, **+** and then select the **math** library from the menu at the bottom. We need the math library in Microsoft Touch Develop so that we can do our maths calculations and work out the position of the accelerometer.

```
add egg y := egg y + 1
led → plot(egg x, egg y)
+ acc x := input → acceleration("x")
+ basket x := 2 + math
  '+' expects Number here, got Math (ID103)
+ basic → pause(300)
  nothing
```

23

Select the **min** button. Replace the first number with **2** and then **math** and then **max**. This code is finding the highest and lowest values that the board could move to the left and right. Your code should look like the picture, on the right.

```
led → plot(egg x, egg y)
var acc x := input → acceleration("x")
basket x := 2 + math → min(2, math → max(0, 0))
basic → pause(300)
```

24

To finish, we need to replace the maximum values with **-2** and then the variable we created. In this case it's called **acc x**. Divide it by **200**.

```
led → plot(egg x, egg y)
var acc x := input → acceleration("x")
basket x := 2 + math → min(2, math → max(-2, acc x / 200))
basic → pause(300)
```

25

You now need to add a line above the **pause** line to plot the basket. To do this, select **led** and **plot** and click on **data** to use **basket x**.

```

┌ basket x := 2 + math → min(2, math → max(-2, acc x / 200))
└ led → plot(┌ basket x, 4)
basic → pause(300)

```



Test your program on the device

Before you run the program on your device, use the simulator to test that it works on screen. Use the mouse to simulate accelerometer input. You should now notice that the lights change when you move the BBC micro:bit left and right. You should also notice that the egg falls from the top of the display.



Use IF statement to check the final position of the 'egg'

We now want to make sure that the egg moves back to the top of the screen when it gets to the bottom of the board. We will need to use an **IF** statement to do this.



26

Select **if** from the menu and create a conditional statement for if **egg y** is greater than 4. This means the egg is off the display because 4 is the bottom of the board.

```

┌ led → plot(┌ basket x, 4)
└ if ┌ egg y > 4 then
    | do nothing
    └ else do nothing end if
basic → pause(300)

```

27

Inside this **IF** statement, we're going to tell the egg to find a new position at the top of the screen. We're going to set the value to **-1** so that the egg is hidden just above the board before it appears at 0. We're then going to set the position of **egg x** using the **math** library to find a random value. Can you repeat the code shown? If you test your program now, you will notice that the egg keeps falling down the screen in random positions.

```

┌ if ┌ egg y > 4 then
    | ┌ egg y := -1
    | ┌ egg x := math → random(5)
    └ else do nothing end if
└ basic → pause(300)
end

```

Do your own thing!

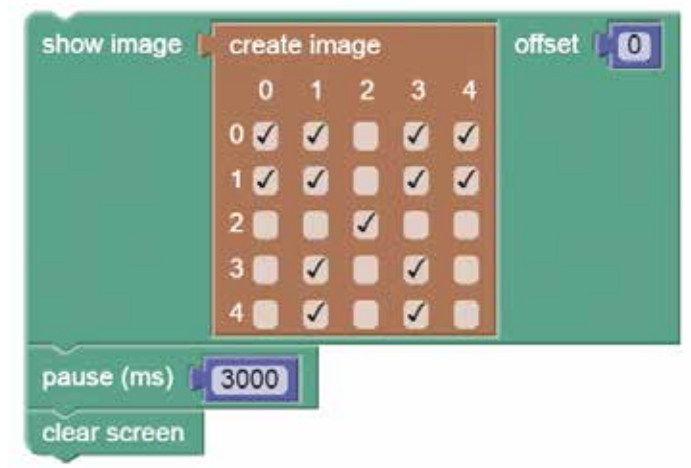
- Now that the egg falls down the display, use an **IF** statement to detect if you caught the egg in the basket.
- We can keep track of the score now that we can detect when the egg is caught in the basket! Add the game library to your script so you can keep track of how many eggs have been caught with the add score function.
- Don't forget to remove one life when an egg falls off the display. This way when the player runs out of lives, the game will be over and the score will be displayed.
- Make the game harder as you catch more eggs. To do this, create a global variable called **falling pause** to keep track of how fast the egg falls. After every 5 catches, subtract 50 from the value of **falling pause** so the egg will keep falling faster down the display as the basket fills up with eggs.



A solution for the complete catch the egg game code can be found on page 31. The solution to these challenges can be found at www.microbit.co.uk/td/lessons/catch-the-egg-game/challenges

Solutions

Lesson 1: Digital key chain



Lesson 3: Digital pet

```
function main ()
  basic → forever do
    if input → button is pressed("A") then
      ▷ set sleep
      basic → pause(5000)
    else
      ▷ set awake
    end if
  end
end function
```

Lesson 2: Rock, paper, scissors

```
function main ()
  input → on shake do
    var img := image → create image( [rock] )
    var offset := math → random(3) * 5
    img → show image(offset)
  end
end function
```

Lesson 4: Catch the egg game

```
function main ()
  basket x := 2
  egg x := 2
  egg y := 0
  led → plot(egg x, egg y)
  led → plot(basket x, 4)
  basic → forever do
    led → unplot(basket x, 4)
    led → unplot(egg x, egg y)
    egg y := egg y + 1
    led → plot(egg x, egg y)
    var acc x := input → acceleration("x")
    basket x := 2 + math → min(2, math → max(-2, acc x / 200))
    led → plot(basket x, 4)
    if egg y > 4 then
      egg y := -1
      egg x := math → random(5)
    else do nothing end if
    basic → pause(300)
  end
end function
```


The BBC micro:bit and the curriculum

Although the BBC micro:bit has been designed with young people's own independent use in mind, for schools in England following the new computing curriculum, BBC micro:bit has the potential to be a really interesting platform for exploring lots of the required content.

KS3 Computing PoS Subject content

design, use and evaluate computational abstractions that model the state and behaviour of real-world problems and physical systems

understand several key algorithms that reflect computational thinking [for example, ones for sorting and searching]; use logical reasoning to compare the utility of alternative algorithms for the same problem

use two or more programming languages, at least one of which is textual, to solve a variety of computational problems; make appropriate use of data structures [for example, lists, tables or arrays]; design and develop modular programs that use procedures or functions

understand simple Boolean logic [for example, AND, OR and NOT] and some of its uses in circuits and programming; understand how numbers can be represented in binary, and be able to carry out simple operations on binary numbers [for example, binary addition, and conversion between binary and decimal]

understand the hardware and software components that make up computer systems, and how they communicate with one another and with other systems

understand how instructions are stored and executed within a computer system; understand how data of various types (including text, sounds and pictures) can be represented and manipulated digitally, in the form of binary digits

undertake creative projects that involve selecting, using, and combining multiple applications, preferably across a range of devices, to achieve challenging goals, including collecting and analysing data and meeting the needs of known users

create, re-use, revise and re-purpose digital artefacts for a given audience, with attention to trustworthiness, design and usability

understand a range of ways to use technology safely, respectfully, responsibly and securely, including protecting their online identity and privacy; recognise inappropriate content, contact and conduct and know how to report concerns

BBC micro:bit contexts

Students can learn much about the idea of abstraction by thinking about the different layers of systems that have to operate together to make the BBC micro:bit work, as illustrated by the relationship of Microsoft Touch Develop or Blockly to C++ and to the ARM mbed machine code that runs on the chip itself.

There's scope here to get students thinking algorithmically, carefully planning their programs before they write any code. Some key algorithms could be implemented on the BBC micro:bit too, from finite state machines (Lesson 3: Digital pet) to 'guess my number' games using binary search.

Students could compare programming the same algorithm in both the Microsoft Block and Microsoft Touch Develop Editors. They can also learn to design and develop modular programs using user-defined functions in Microsoft Touch Develop.

There's chance to explore Boolean logic using the AND, OR and NOT operators built in to the language and the A and B input buttons on the BBC micro:bit.

The 25-pixel display lends itself to investigating binary representation, both for images, creating simple bitmap sprites, and for numbers, using it to display numbers up to 2^{25} using binary place value! Why not create a binary counter or even a clock using the BBC micro:bit?

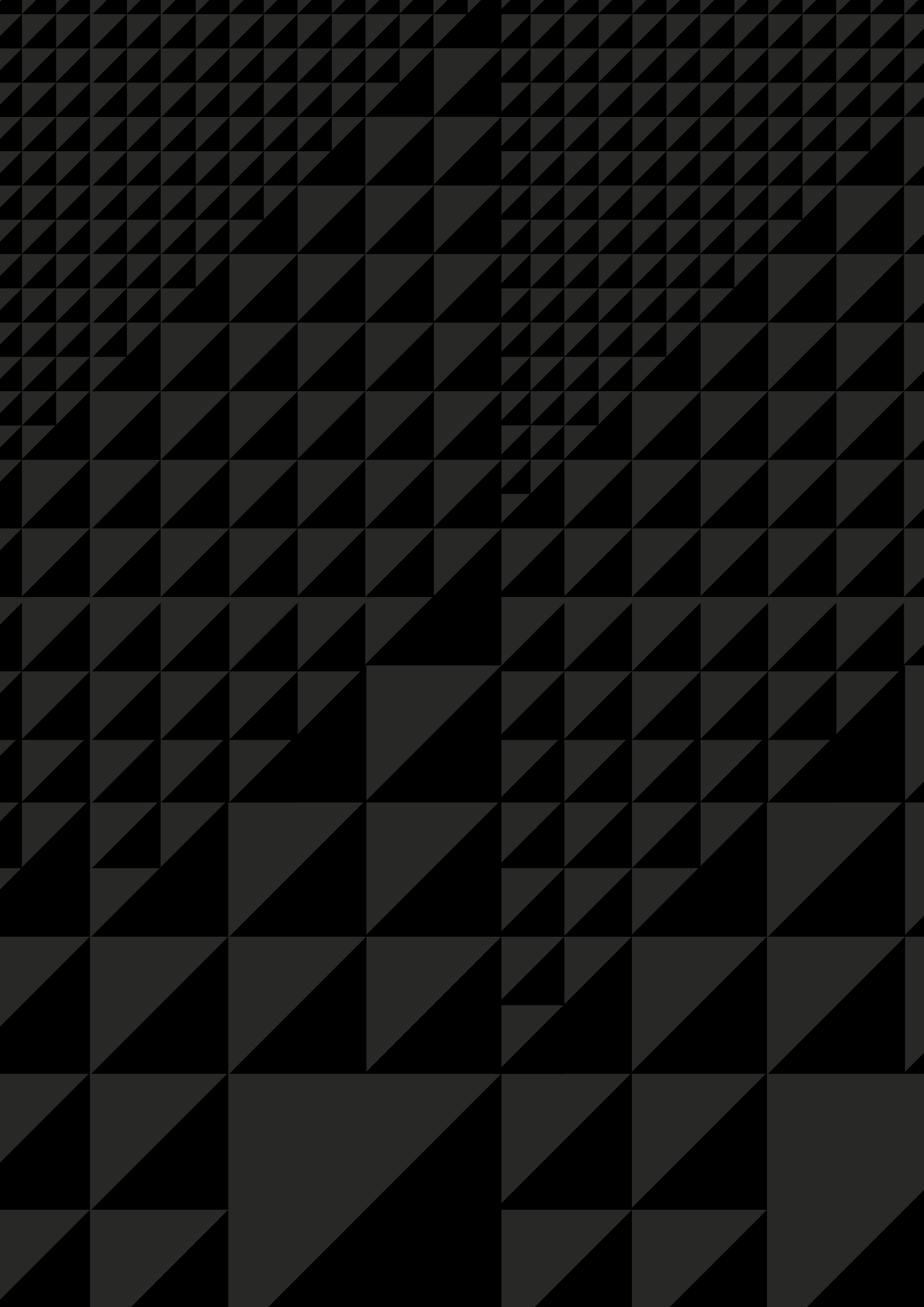
As it's a simple system, the BBC micro:bit provides a more accessible way for students to grasp complex ideas of how hardware and software systems behave and communicate.

The use of compiled machine code here might be part of a unit of work exploring how instructions are stored and executed in computers.

There's ample scope for creative projects here, achieving challenging goals and meeting the needs of known users. The limitations of the BBC micro:bit interface make it a great way to think creatively about design and usability.

Remixing code via the BBC micro:bit site provides some great opportunities for working with 'digital artefacts' produced by others.

Participating in the BBC micro:bit online community provides an opportunity to emphasise the need for respect and responsibility when working online.



BBC



micro:bit

The BBC micro:bit Quickstart Guide for Teachers is designed to support educators in effective use of the BBC micro:bit devices distributed to all Year 7 students in the United Kingdom as part of the BBC's *Make It Digital* initiative.

Supported by Microsoft and published by Hodder Education, this indispensable guide features:

- An introduction to the *Make It Digital* initiative
- An outline of what the BBC micro:bit is and what it is designed to do
- Advice on how teachers and students can get the most out of the BBC micro:bit device, including how the hardware and supporting services work (including the BBC micro:bit website, code editors and code compiler)
- Guidance on how to get started with creating programs for the BBC micro:bit using the Microsoft Touch Develop Editor, and how to compile them and upload them to your device
- Coding lessons of varying difficulty with step-by-step walkthroughs and solutions for each activity
- Curriculum references, providing educators with opportunities to introduce key computational thinking concepts and map outcomes back to aspects of the English computing programme of study



9 781471 863820 >