Getting started with the BBC micro:bit

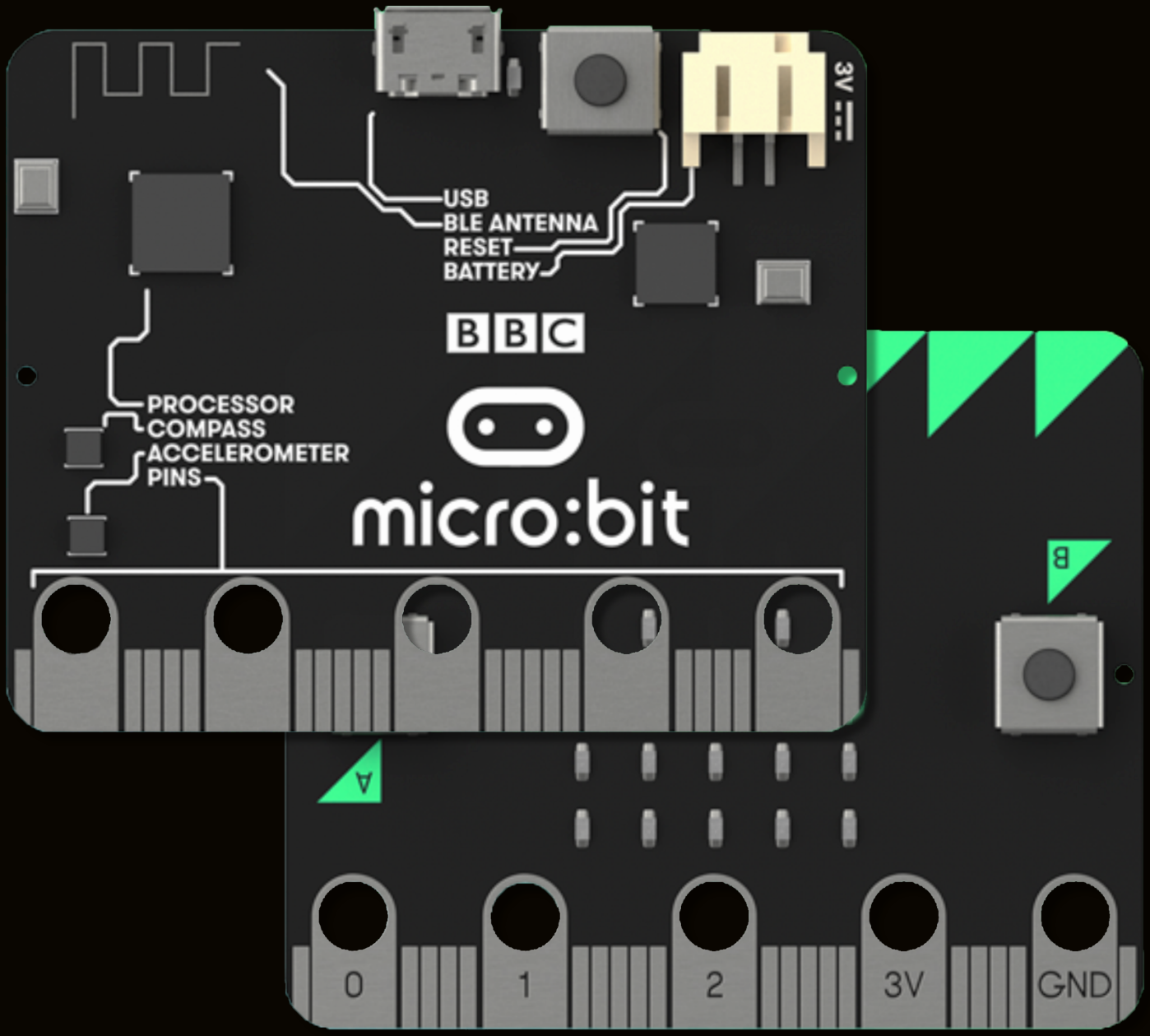


Table of contents

The micro:bit

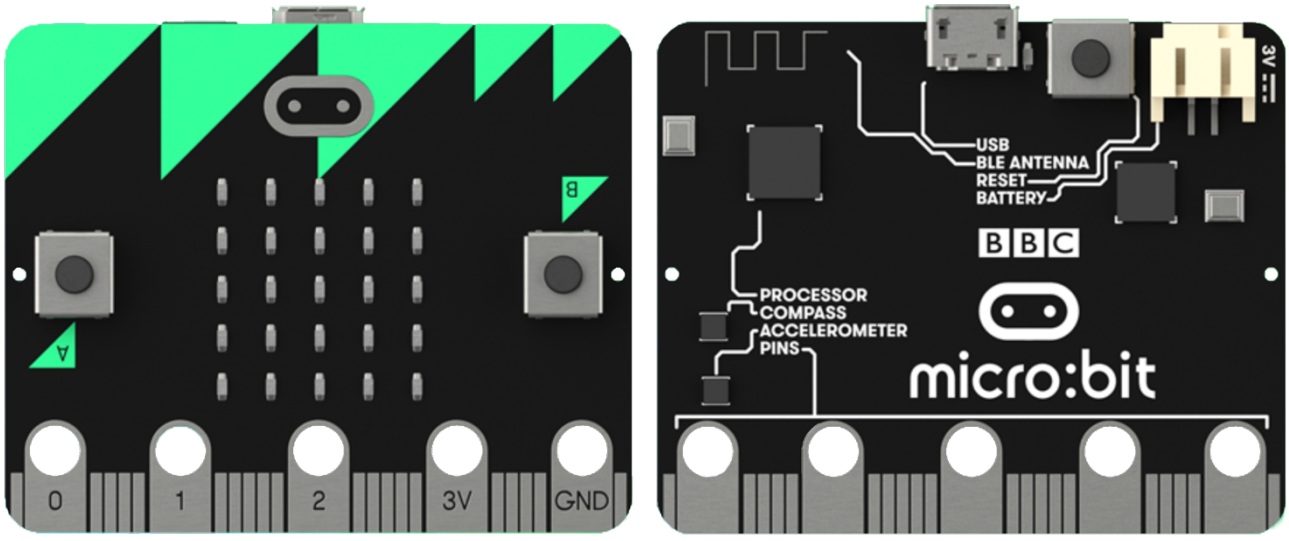
First conceived in 2012 and released in 2016, the BBC micro:bit is a small (4 x 5 cm) **microcontroller board** intended to encourage children to get more involved in computer coding, electronics and **making**.

Micro:bits were given away free to every year 7 (ages 11-12) school-child in Britain in the 2015-16 academic year – around one million boards in total. Once this aim had been achieved, they were put on sale to the public. At present they are available from sellers such as **Pimoroni** and **The PiHut** in the UK.

The micro:bit features 25 LEDs in a 5 x 5 grid, two push-buttons, a compass, an accelerometer, five 'large' physical pins suitable for crocodile clips or banana plugs, and a further 20 small pins available through an edge-connector. They can be powered by an external battery pack or via USB, and communicate through the same USB port or built-in Bluetooth.

The BBC and its partners have provided options for programming via a web-app, mobile devices or desktop software, in multiple languages, and using a variety of interfaces ranging from drag-and-drop, through touch-optimised, to full text.

At a recommended selling price of £13 (generally £16 including an optional battery pack), the micro:bit is a great introduction to coding and electronics.

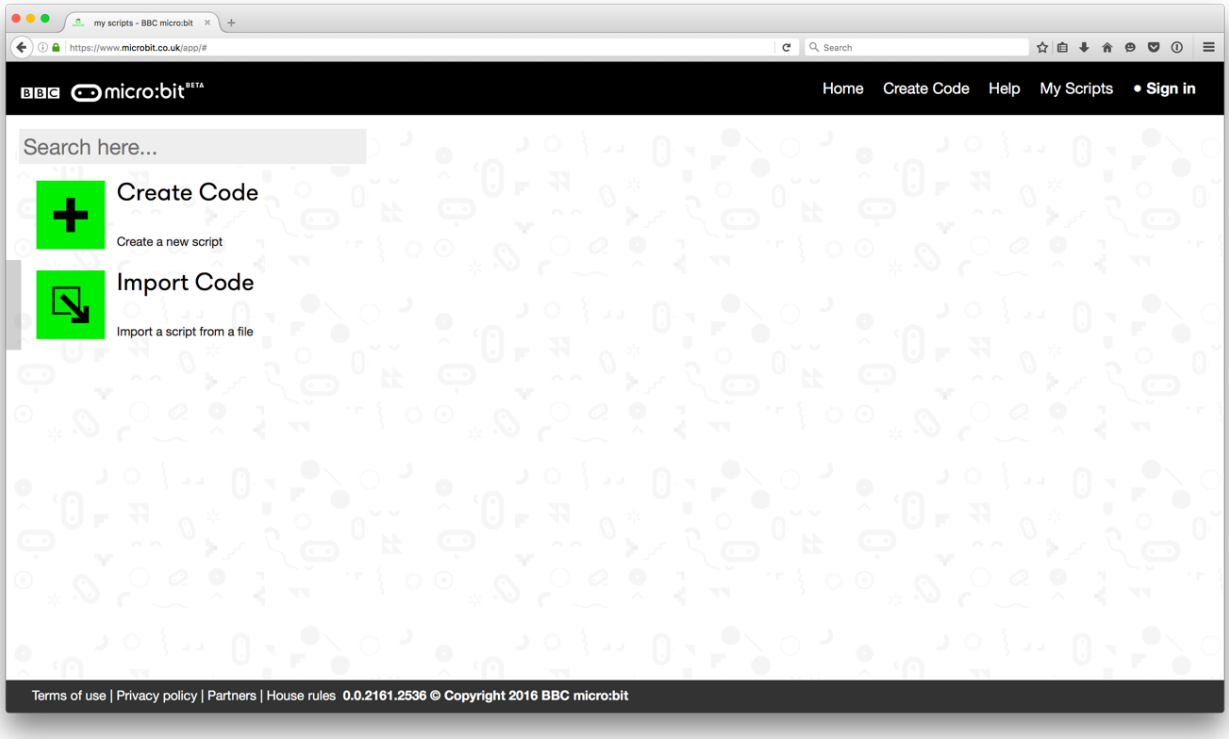


Front and back view of the micro:bit – components are helpfully labelled

Getting up and running

Loading the web app for the first time

At first, most of your time writing code for the micro:bit will be spent in a web browser, using the micro:bit *web-app*. You can find this at microbit.co.uk/app/ – the first time you visit, it should look something like this:

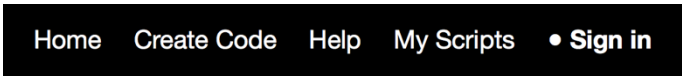


The micro:bit web-app

Depending on your browser and security settings, you may be asked for permission to download and install a file – you should definitely reply '**yes**', because ...

> After you've visited the web-app that first time, you can visit microbit.co.uk/app/ **even if you're not online** and you'll be able to access all your scripts as well!

Let's have a quick look at that page and see what options we have in the **menu-bar**:



Home – takes you to the BBC micro:bit homepage

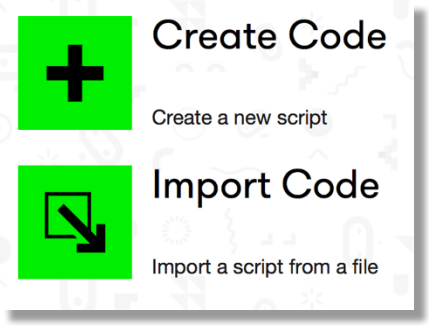
Create Code – start a new script (more on this soon)

Help – opens the help page on the BBC micro:bit site

My Scripts – shows a list of all your scripts, on the same page as the image above (scripts will appear above the green **'Create Code'** box once you have some!)

Sign in – unless you are a teacher at a school that has registered for the site, you can (and should) ignore this option

And then there are the two big, friendly buttons in the main panel:



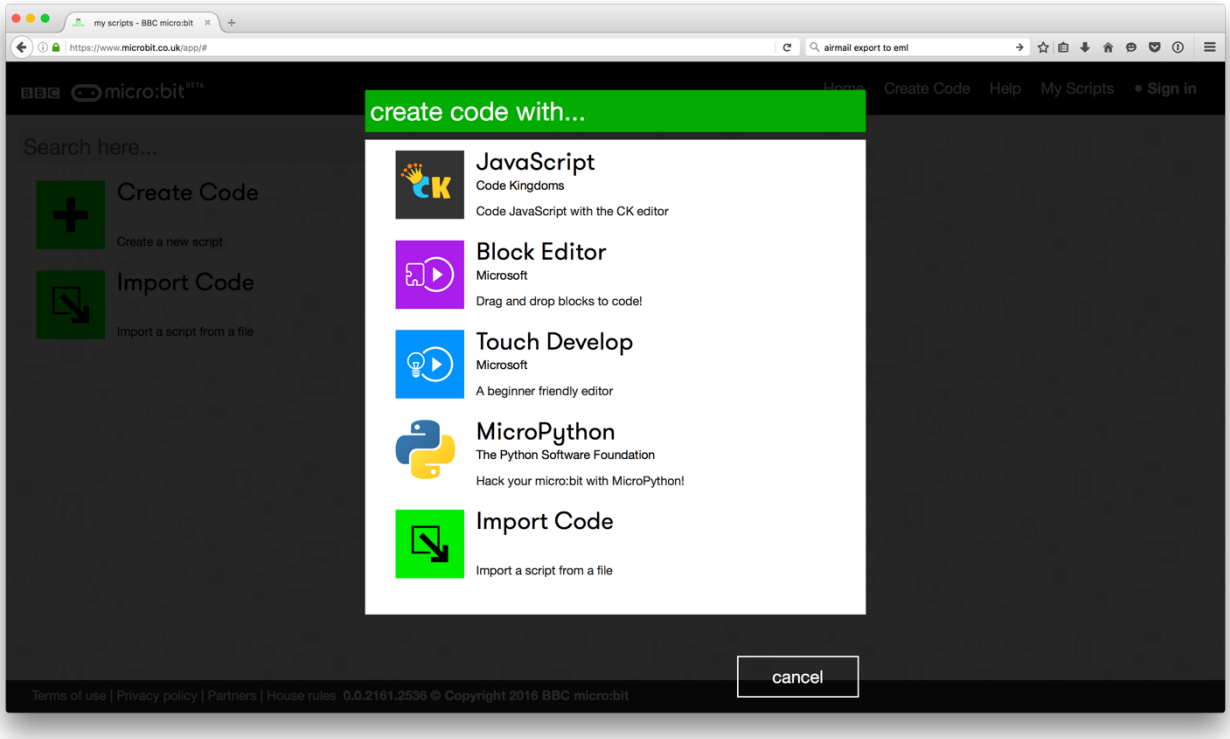
- **Create Code** – start a new script (see the next section)
- **Import Code** – lets you upload either a compiled **.hex** file (in which case it'll also de-compile back to the original code) or a **.jsz** script file

> **A .hex says what?** As we're going to see shortly, there are four ways of writing code for the micro:bit – but the device doesn't understand any of the languages that your code is going to be written in. Instead, the web-app puts your code into a **compiler** which converts it into a series of instructions that the micro:bit *can* understand, and generates a **compiled** file which you can then **flash** to your micro:bit. This compiled code is written in hexidecimal, and the files are called **.hex** files

Creating Code

The next thing to do, naturally enough, is to create your first *script* – so click on **Create Code** (either in the menu bar or that big, friendly green button).

This will bring up the following options screen



Selecting how to create your code

You're offered four different ways that you can write code for the micro:bit, together with an option to **Import Code** – which does exactly the same as it did from the previous screen.

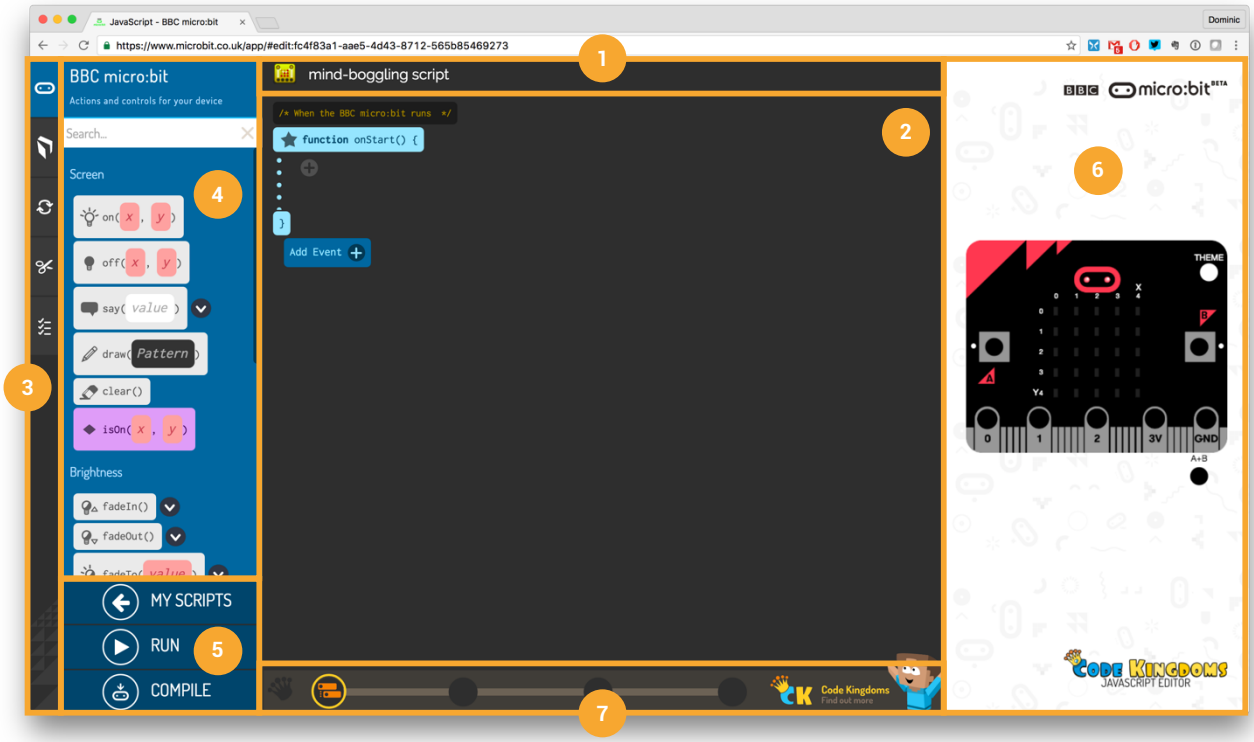
The four coding options are:

Name	What Is It?	Good for
JavaScript Code Kingdoms	A simple but powerful drag-and-drop interface hides the JavaScript underneath	Beginners & intermediates
Block Editor Microsoft	A drag-and-drop editor, visually similar to *Scratch*, hiding TouchDevelop code underneath	Beginners
Touch Develop Microsoft	A coding interface developed specifically to work well on touch-screen devices	Beginners & intermediates
MicroPython Python Foundation	A full programming language based on the widely-used Python	Intermediates & enthusiasts

JavaScript from Code Kingdoms

Your first script

For your first script, we're going to use the JavaScript editor from Code Kingdoms. Click on the CK logo to launch the CK editor.



1	**Script name**	The editor will provide a temporary name – feel free to change it!
2	**Coding panel**	Code is shown (drag-and-drop) or written (text) here
3	**Menu sidebar**	Select tab to show required section in the menu
4	**Menu**	Elements to drag into the coding panel
5	**Action buttons**	Options to return to the script selection window; run script in the simulator; or compile code for transfer to the micro:bit
6	**Simulator**	Simulator for testing scripts
7	**Slider**	Choose how the script is presented – far left for drag-and-drop, far right for pure JavaScript text

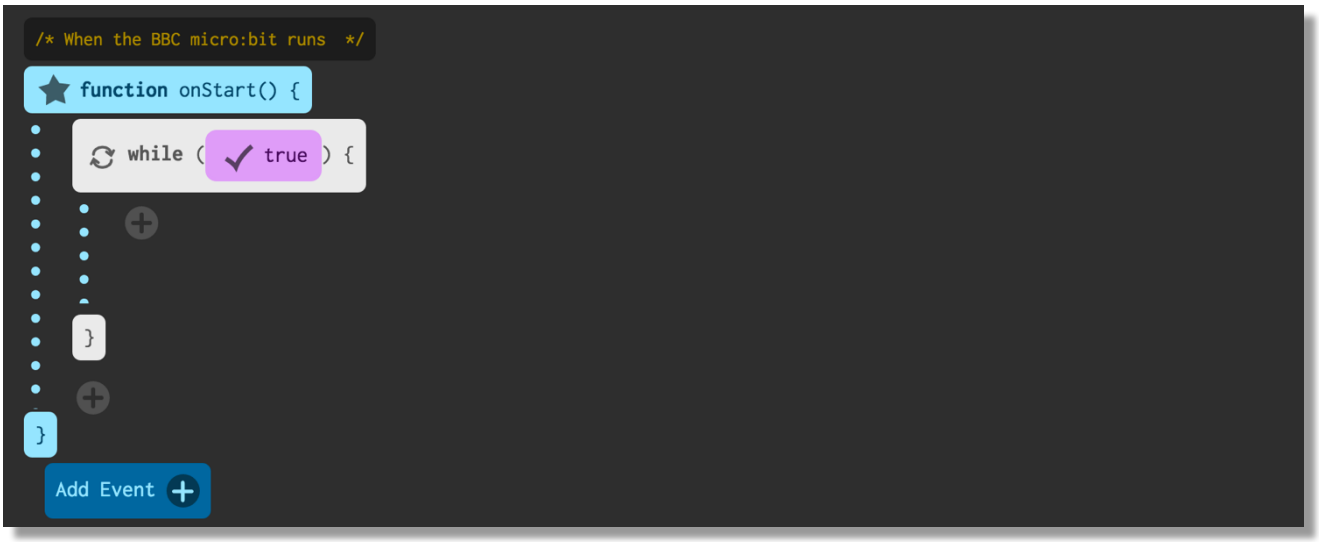
> We're going to write a *very* simple script, which will make an LED on the micro:bit flash on and off. The instructions will give a brief explanation of what's going on, but if you're a beginner don't worry about what's going on – just follow along for now.

Our script will make one of the LEDs on the micro:bit flash on and off. It'll start flashing when we turn the micro:bit on, and keep flashing until we turn it back off again.

To turn this into code in the CK editor:

- Select the **Language** menu by clicking on the icon
- Find the `while ( test ) { }` block and drag it inside the `onStart()` function
- Click on the `test` and select `true`

At this point your script should look like this:



> The `onStart()` function is called when the micro:bit is turned on.

> The `while (test)` block is a **control statement** which loops so long as the test evaluates to true; a `while (true)` test is a common way of creating a loop that will repeat forever (at least, until the micro:bit is turned off again).

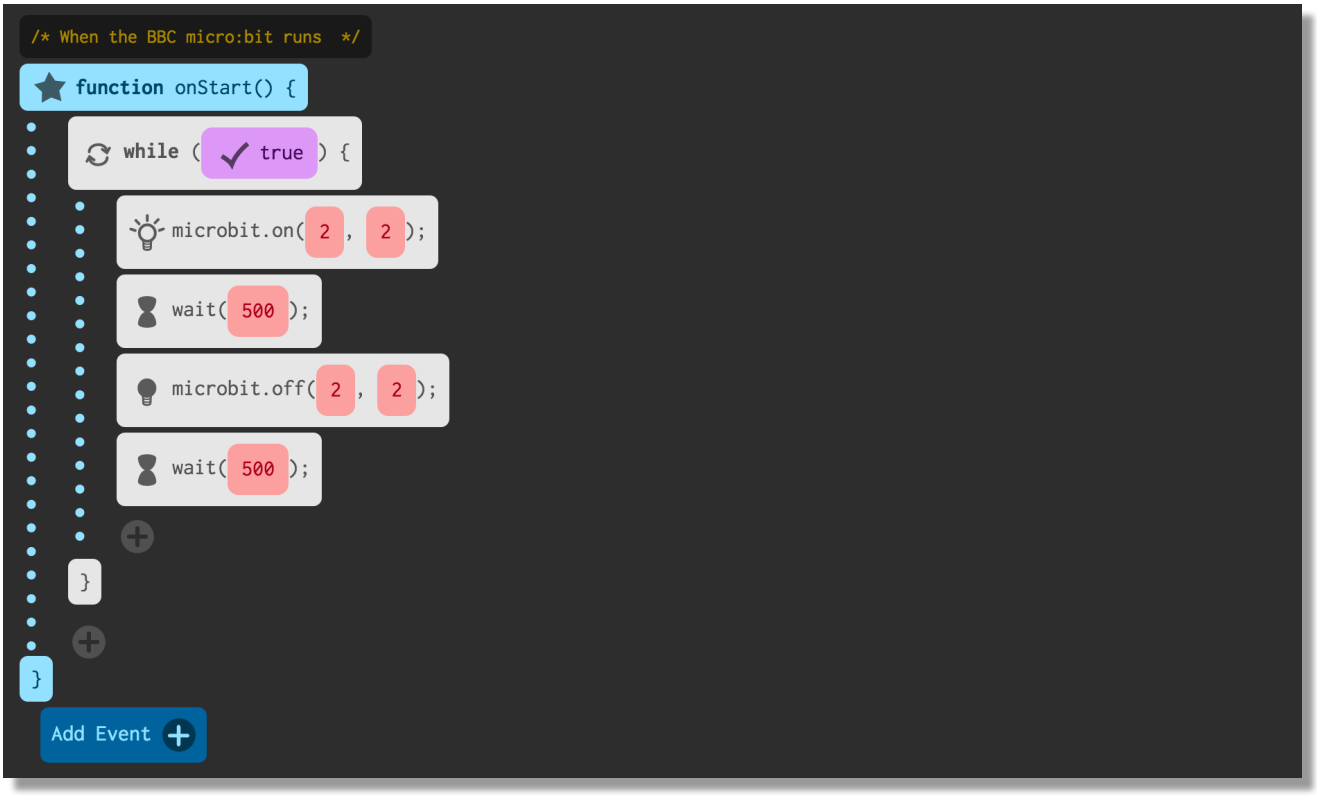
We now have a block of code that will start when the micro:bit is turned on and then loop until it's turned off again – so now we just need to write the code to turn an LED on and off again.

- Select the **BBC micro:bit** menu by clicking on the icon
- Drag out a on(x , y) block and drop it inside the `while` loop
- Click on *x*, then type `2` in the pop-up dialog; do the same with *y*
- Drag out a off(x , y) block and drop it *inside* the `while` loop under the LED-on
- Set the `(x,y)` co-ordinates to `(2,2)` again

You can try running the code on the simulator – just click on the **RUN** button. You'll see the middle LED comes on … and then stays on. This is because although the LED is turning on and off, but **much** faster than we can see. So we need to slow things down a little …

- Select the **Language** menu (that's the icon)
- Drag out a wait(*milliseconds*) block and drop it below the LED-on block
- Drag out a second `wait` block and drop it below the LED-off block
- Choose how quickly you want the LED to blink on and off, and set the two *`milliseconds`* – remember that one second is 1,000 milliseconds!

Your finished script should look something like this:



Flashing the micro:bit

Obviously you can run the script in the simulator again, but wouldn't you much rather see it running on the micro:bit instead?

The way to do this is pretty simple:

- Click on the **COMPILE** button
- A **.hex** file will be downloaded and then a **Success!** banner will appear
- Connect the the micro:bit to your computer using a USB cable; it will show up as an external drive
- Copy the .hex file to the micro:bit drive – it's probably easiest to drag and drop in your file browser
- The yellow status light on the back of the micro:bit will flash to indicate the file is being copied, and then the micro:bit will restart
- Your LED should now be flashing!

> Depending on your computer, you may see a message about the micro:bit not having been removed properly – don't worry about it …

Congratulations, you've just written your first script for the micro:bit and successfully transferred it to the device!

The hidden JavaScript

To finish up our quick survey of the Code Kingdoms editor, let's take a look at the slider at the bottom of the screen. There are four possible settings:

- With the slider in the far left-hand position, we get a full drag-and-drop interface with visual cues (the light-bulbs in the LED-on and LED-off scripts, for instance);
- The second position keeps the drag-and-drop interactions and the block-style design, but drops the additional visual cues;
- The third position again keeps the drag-and-drop interactions but reduces the content to 'pure' JavaScript code – the visual cues and the colored blocks are gone;

- In the fourth and final position, we have pure JavaScript in a text editor. The menu pane still has the code-blocks that we're seen but instead of drag-and-drop, clicking on a block inserts that code in the text editor at the current cursor position

This demonstrates the genius of the CK editor – even if you're working in the full graphical interface, you're **actually** writing JavaScript code under the hood! As an example, here's how our script to blink the LED looks in the full text editor:

```
1  /* When the BBC micro:bit runs   */
2  function onStart(  ) {
3      while (true) {
4
5          microbit.on(2, 2);
6          wait(500);
7          microbit.off(2, 2);
8          wait(500);
9
10     }
11
12
13 }
14
```

JavaScript is a very widely-used scripting language, and is a fundamental part of the technologies powering the internet. Although the micro:bit has a limited implementation of JavaScript, the Code Kingdoms editor is a great way to learn the basics of coding in this very popular language.

Continuing with JavaScript

> It's beyond the scope of this book to go into more detail on learning to code in general, or the specifics of any language in particular. Instead after each section you will be presented with other resources (including books, videos and websites) if you want to go into more depth.

JavaScript on the micro:bit

- The ideal starting place are the interactive tutorials from Code Kingdoms; these can be accessed from within the CK editor by clicking on the  icon, which will bring up the **Choose a Tutorial** menu
- A *cookbook* of the tutorials is also available for download, together with a *cheat sheet*. They can both be found at microbit.co.uk/ck

JavaScript 'at large'

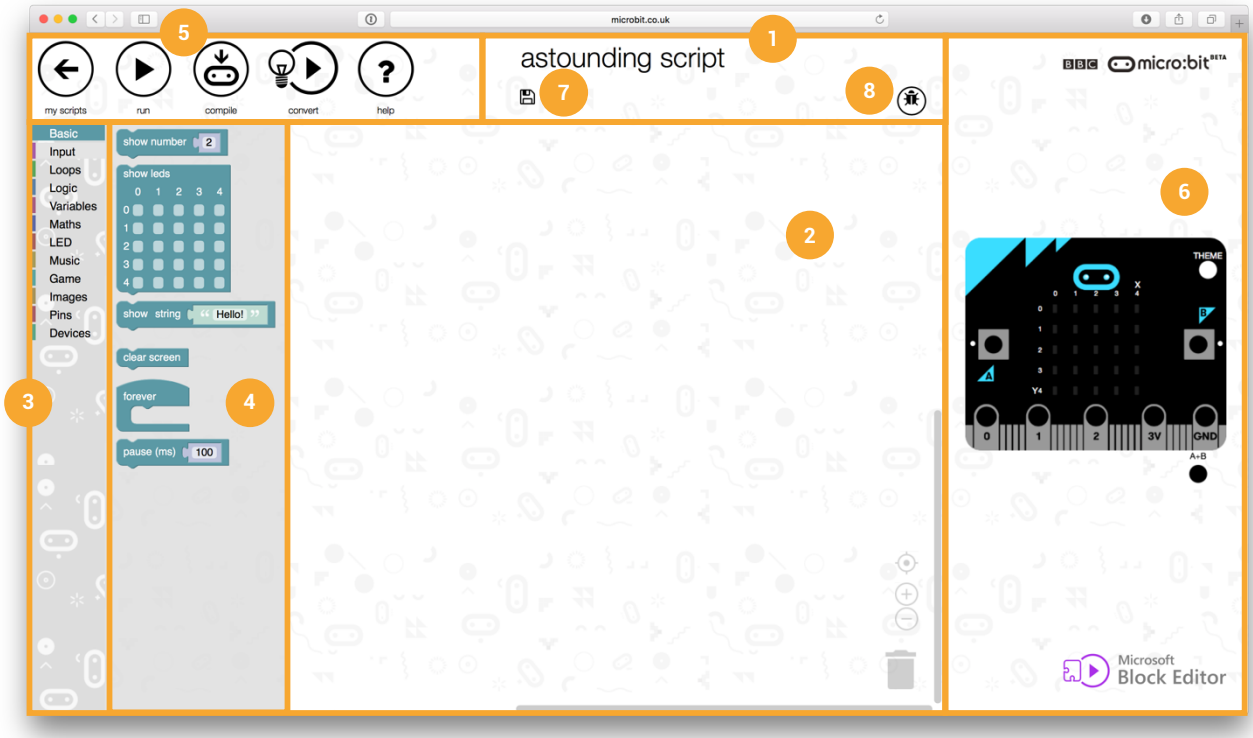
- To learn more about JavaScript 'at large' a good place to start is Codecademy's JavaScript course, which can be found at codecademy.com/learn/javascript. Although there is a paid Pro option, the free elements of the course are more than enough to get a grounding in JavaScript
- Eloquent JavaScript (eloquentjavacsript.net) is an excellent resource once you've learnt the basics of the language
- Once you've got an understanding of the language, codewars (codewars.com) is very useful, providing bite-size challenges to exercise your coding-fu

Block Editor from Microsoft

> We'll take a look at Block Editor and Touch Develop separately, but note that – in the same way that the CK editor presents different ways of coding in JavaScript – Block Editor is a drag-and-drop interface hiding Touch Develop code underneath.

Block Editor

We'll take a quick look at the Block Editor by re-creating our blinking LED script. From the web-app home screen, click on **Create Code** and pick the **Block Editor** option.



1	**Script name**	The editor will provide a temporary name – feel free to change it!
2	**Coding panel**	Drag-and-drop your blocks here to build your code
3	**Menu sidebar**	Bring up the relevant menu
4	**Menu**	Elements to drag into the coding panel; click in the sidebar to show, click again to hide

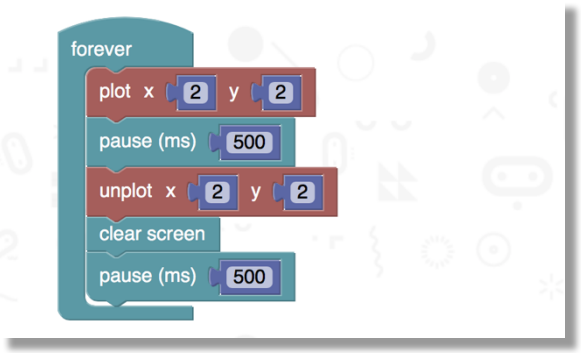
5	**Action buttons**	Options to return to the script selection window; run script in the simulator; compile code for transfer to the micro:bit; convert code to Touch Develop; or open the help
6	**Simulator**	Simulator for testing scripts
7	**Save status**	Status indicator – a icon shows there are unsaved changes, whilst a shows that the script has been saved (auto-saves every few seconds)
8	**Bug monitor**	Summary of any problems with your script

Quick Block Editor Script

To give a quick indication of how the Block Editor works, we'll re-write our flashing LED script.

- Click on **Basic** in the sidebar and drag out a **forever** block into the coding panel
- Click on **LED** in the sidebar and drag out a **plot** and an **unplot** block into the coding panel; move them so that they're inside the **forever** block
- For both the **plot** and an **unplot** blocks, set **x, y** to **2, 2**
- Click on Basic again and drag out two **pause (ms)** blocks – drop one under the **plot** block and the other under the **unplot** block
- Change the pause periods how you want – remember, they're in milliseconds …

Your script should now look something like:



- Click on **run** in the actions menu to see how this looks in the simulator

Copy your script to the micro:bit

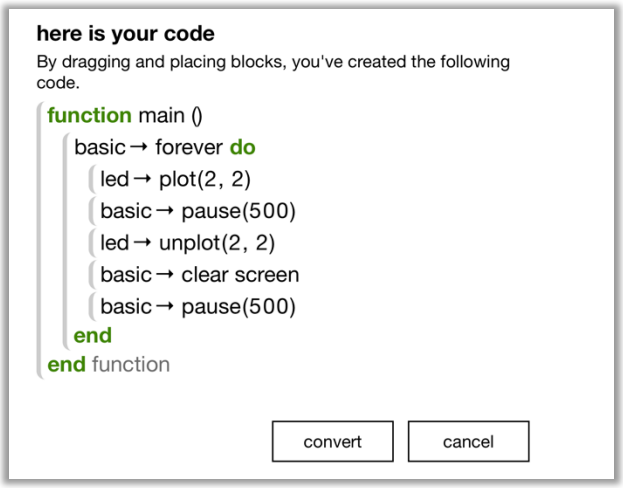
Copying the script to the micro:bit is essentially the same as it was for the CK editor:

- Click on the **compile** button to download your script as a **.hex** file
- Connect the the micro:bit to your computer using a USB cable; it will show up as an external drive
- Copy the .hex file to the micro:bit drive
- The yellow status light on the back of the micro:bit will flash to indicate the file is being copied, and then the micro:bit will restart
- Your LED should now be flashing!

Converting your script to Touch Develop

Microsoft provide a handy way to see how your code looks in Touch Develop – just click on **convert**. This will bring up a pop-up showing how your script looks if written in Touch Develop, and an option to convert it - doing so saves it as a Touch Develop script, but the original Block Editor script will **not** be deleted.

The Touch Develop code generated from my example looks like this:



Continuing with Block Editor

On the micro:bit website

- There's an introductory book, **Learning to program with Microsoft Block Editor**, available on the microbit website – go to microbit.co.uk/blocks/book. Unfortunately at the time of writing this isn't available as a download – you'll just have to read it online …
- There are also a number of lessons for all levels, from Beginner to Advanced, on the website at microbit.co.uk/blocks/lessons. These are written in the form of lesson plans, so if you want to jump straight to the instructions look in the **Quick Links** section of the lesson for the **activity** link.
- And finally there's comprehensive documentation for Block Editor, which can be accessed direct from the editor by hitting the help icon. It's also available at microbit.co.uk/blocks/contents

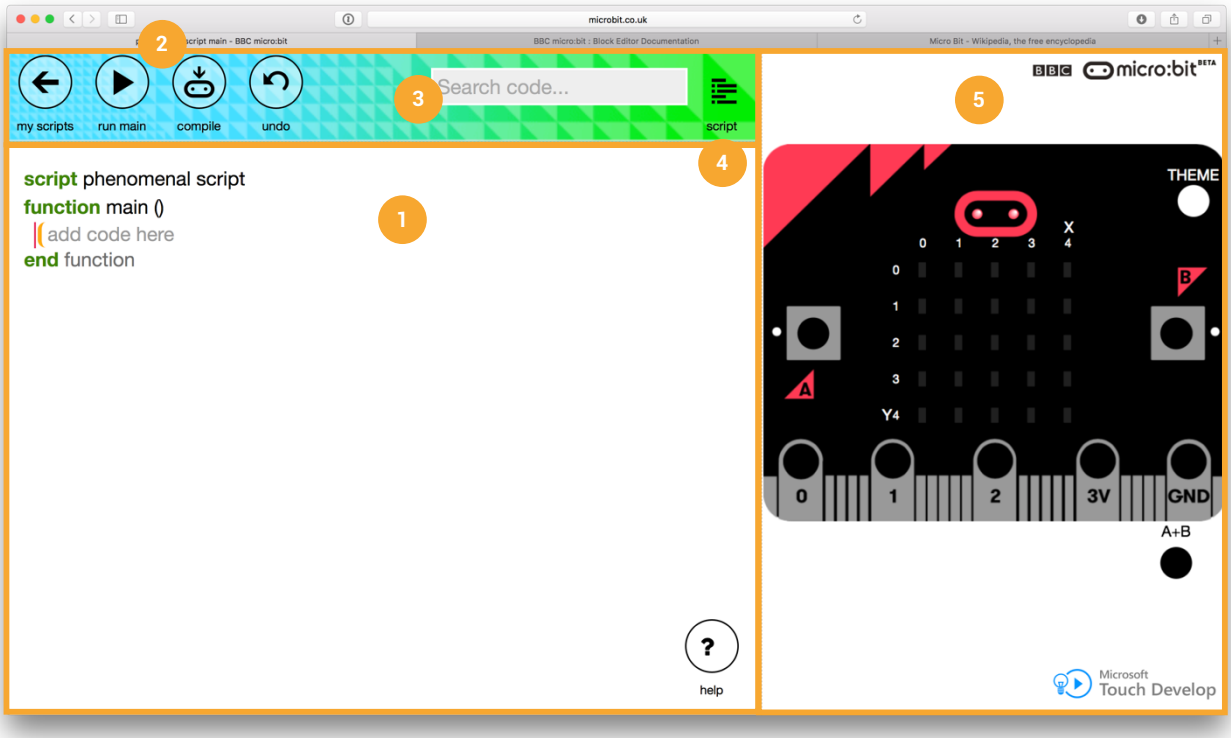
Elsewhere

- Element14 (who were involved in the manufacture of the micro:bit) have an excellent short course 10 BBC micro:bit Projects in 10 Days which is focussed on the Block Editor
- Chris Harvey, from the ICT department at St Olave's school in York, has made a series of videos about the micro:bit available on YouTube. Although he concentrates on JavaScript later in the series, the first three videos introduce the Block Editor – you can find them here

Touch Develop from Microsoft

The Touch Develop interface

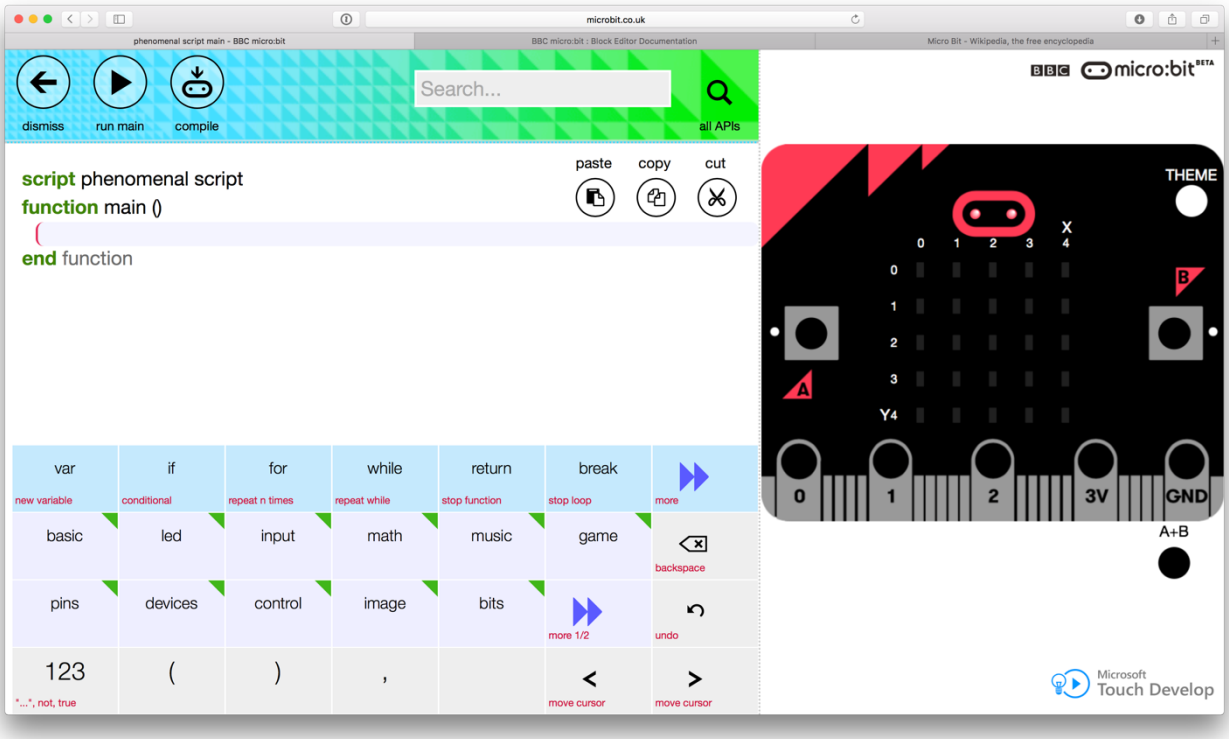
If you followed along with the Block Editor section, you may have already converted the Block Editor version of our blinking LED script into a Touch Develop version. However now we're going to write it from scratch – so go back to the web app, click on **Create Code** and select **Touch Develop** from the list of editors.



1	**Coding panel**	Where you interact with the editor to write your code
2	**Actions panel**	Options to return to the script selection window; run your script in the simulator; compile your script; and (where relevant) undo the last edit
3	**Search bar**	Search your code
4	**Menu (script)**	Splits the coding panel in two and shows a menu in the right-hand side
5	**Simulator**	Simulator for testing scripts

> The Touch Develop interface has been developed to be touch-friendly so that it works well on mobile devices.

Unlike either the CK editor or Block Editor, in Touch Develop the code command blocks aren't visible until you touch (or click) the screen. If we click inside the **main ()** function, the screen will change to bring up the code blocks:



Most of these options are familiar from the sidebar in Block Editor; clicking on one shows the available commands.

A quick Touch Develop script

To get a feel for Touch Develop, we'll quickly recreate our flashing LEDs script:

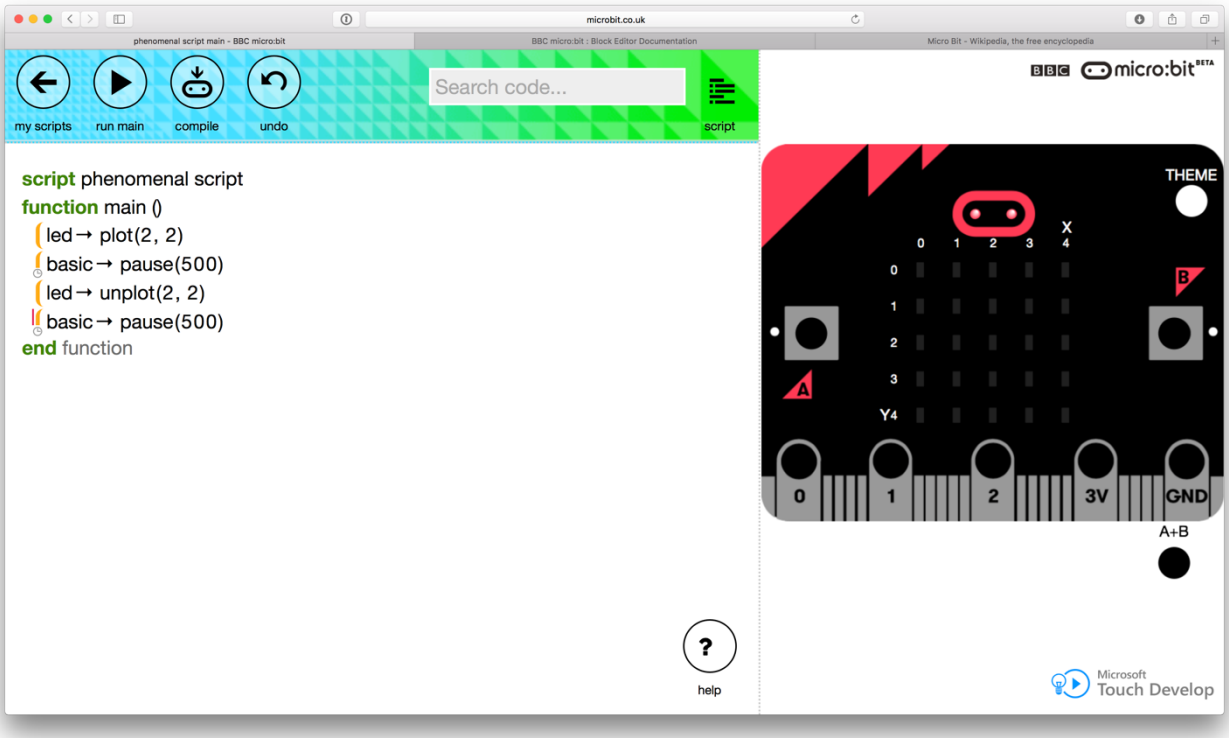
- Click on the grey line that says **add code here** to bring up the code menu
- Click on **led** and then choose **plot** from the list that appears
- In the coding panel the cursor will be flashing in front of the first **0** – click and then type **2**, then click on **move cursor** and type **2** again
- Click anywhere outside of the function to finishing editing

- The finished command should be led → plot(2, 2)

You should now see large ⊕ symbols above and below your line of code (if not, click on the first line again) – clicking on one of these adds a new line of code above or below the highlighted line of code.

- Click on the ⊕ below you **plot** command to create a new row
- Click on **basic** and then **pause**, then choose a time (in milliseconds) to wait
- Click on the ⊕ below your **pause**, and add an **unplot** command (in the **basic** menu) for co-ordinate **(2,2)**
- Add another **pause** command below the **unplot** command

Your finished code should look a little like this



Continuing with Touch Develop

On the micro:bit website

- There's a short introduction at microbit.co.uk/td/editor together with a video at microbit.co.uk/getting-started/touchdevelop-editor
- There are lessons for all levels, from Beginner to Advanced, on the website at microbit.co.uk/td/lessons. As with the Block Editor, these are written in the form of lesson plans, so if you want to jump straight to the instructions look in the **Quick Links** section of the lesson for the **activity** link.

> Note that some of the lessons are repeats of those from the Block Editor, although obviously re-written to suit the Touch Develop interface.

- If you've started with Block Editor and are moving across to Touch Develop, there's a **From Block Editor To Touch Develop** guide at microbit.co.uk/blocks/to-td
- And finally there's comprehensive documentation for Touch Develop, which can be accessed direct from the editor by hitting the help icon. It's also available at microbit.co.uk/td/contents

Elsewhere

There's a huge amount of content on Microsoft's Touch Develop website, including:

- A brief introduction at touchdevelop.com/microbit;
- A getting-started guide at touchdevelop.com/docs/getting-started;
- Tutorials at touchdevelop.com/docs/tutorials;
- And a full set of documentation at https://www.touchdevelop.com/docs, including a language and API references

MicroPython from The Python Software Foundation

The MicroPython interface

The last of the editors on the micro:bit website is MicroPython. Start a new script from the web app by clicking on **Create Code** and select **MicroPython**.

1	**Script name**	The editor will provide a temporary name – feel free to change it!
2	**Coding panel**	Where you interact with the editor to write your code
3	**Actions panel**	Options to return to the script selection window; download a compiled .hex file; access snippets; open help; zoom in or out; see save status; and see bug status
4	**Instructions**	Instructions for running your script on the micro:bit

With MicroPython, we're moving into the realm of a *'traditional'* code editor – it's text-based and doesn't even have a simulator for testing code. To see if your code is working, you will actually need to **flash** your script to the micro:bit and try it out for real!

> MicroPython does offer **some** help – if there's an error in your code, a message will scroll across your micro:bit telling you the first line that is a problem together with some information about the type of error.

Two quick scripts

As you can see, when starting a new script you're provided with some default code:

- The statement `from microbit import *` (which you will probably want to leave at the top of any script you write) **imports** all the **library code** from the `microbit` **module** – essentially, this gives you the commands that you need to easily access the micro:bit

Even without any specific knowledge of the Python / MicroPython language, we can make a pretty good guess what the rest of the code does:

- The rest of the script is inside a `while True:` loop – so it'll start looping when the micro:bit is turned on, and continue until it's turned off again
- `display.scroll("Hello, World")` looks like it'll scroll the text "`Hello, World`" across the micro:bit's display …
- and `display.show(Image.HEART)` looks like it's going to show a heart image …
- and finally, `sleep(2000)` is going to pause the script for 2000 milliseconds (or 2 seconds!)

As another example, here's our blinking LED script written in MicroPython:

```
1  from microbit import *
2
3
4  while True:
5      display.set_pixel(2,2,9)
6      sleep(500)
7      display.set_pixel(2,2,0)
8      sleep(500)
9
10
```

We've seen the `while True:` and `pause()` commands already in the sample script, but in line 5 we have a new command: `display.set_pixel(2,2,9)`

- most likely `display.set_pixel()` is going to turn a pixel on …
- and we can be fairly cofident that the `2,2` refers to the co-ordinate of the pixel that we're turning on …
- but what about that `9` at the end? And come to that, what about the `0` in the instruction on line 7?

A look at the BBC micro:bit MicroPython documentation (which you can access by clicking on **Help** in the editor or visiting microbit-micropython.readthedocs.io/en/latest/) shows that the third digit in the `set_pixel` command is for the brightness of the pixel, with `9` being fully lit and `0` being off – so in MicroPython `set_pixel` is used to turn pixels both on **and** off.

We're going to need a bigger boat

This simple script illustrates a key point about MicroPython – it's a powerful language (and can leverage a lot of the power of 'full-fat' Python as well), but alongside that power comes increased complexity. If you're learning MicroPython, you're probably going to need to read the documentation until you're more experienced!

A different editor

If you're thinking of using MicroPython, it's worth considering downloading an editor rather than working in the web-app. There are several options available, but at the moment the best option for working with micro:bit appears to be **Mu**, available free from codewith.mu. This offers the ability to:

- flash your scripts with one click – no need to download and then copy
- access to the micro:bit's file system
- access the micro:bit REPL (read-eval-print-loop) – essentially an interactive Python terminal, running on the micro:bit

Continuing with MicroPython

Unlike the other editors, there is very little documentation for MicroPython on the microbit site. However there **are** plenty of resources:

- First Steps with Python, at microbit.pythonanywhere.com/help.html
- There is excellent documentation for BBC micro:bit MicroPython, at microbit-micropython.readthedocs.io including a series of tutorials and a complete set of reference documents
- Since MicroPython is a refinement of Python, intended specifically for low-power boards like the micro:bit, it's also worth visiting the site for the Python Software Foundation – try python.org/community/microbit/
- There's an excellent set of videos by Tony DiCola from Adafruit, available at their YouTube channel, youtube.com/user/adafruit/playlists (look for the MicroPython playlist)
- And to go with those videos there are a series of written tutorials – look at the learning portal at Adafruit (learn.adafruit.com/) and search for *micropython*

> Although the Adafruit videos and tutorials are very good, they're primarily aimed at their own 'Feather Huzzah ESP8826' board. The principles are the same, but some of the code may be different – for instance the ESP8266 needs a module called `machine` whilst the micro:bit uses the `microbit` module.

Going mobile

One of the original promises of the micro:bit was that it would be possible to code on, and transfer that code to the micro:bit from, a mobile phone. To their credit, this does work, albeit with a some caveats.

Get connected

The first thing you're going to need is an app on your mobile device. Head to your app store and download the micro:bit app.

Next, you need to pair the micro:bit over Bluetooth:

- In the micro:bit app on your mobile device, tap on Connect
- On the micro:bit, press and hold both the A and B buttons; then *still holding* these buttons, press and then release the Reset button
- If successful, the text **PAIRING** will scroll across the micro:bit screen and then an image will be shown (which identifies the micro:bit)
- Back on your mobile device, you will see a recreation of the micro:bit's LED display – replicate the image from your micro:bit and tap **Next**
- Back on the micro:bit you'll see an arrow pointing to the A button; press this and a six-digit pass code will be shown
- Enter the pass code on your mobile device
- and you're connected

> This is where we hit a couple of those caveats:
>
> - Whenever you plug your micro:bit into USB you will need to repeat the whole pairing process to use Bluetooth again; and
> - If you've used MicroPython, pairing may not even start on the micro:bit

> You may find that holding A+B and then pushing reset doesn't have any effect. If so, connect the micro:bit to a computer over USB and flash **any** script from the web app so long as it's **not** MicroPython. Try again, it should work fine.

Mobile coding

Whether you're paired or not, you can code from your mobile device by tapping on **Create Code** in the app. Rather than open an editor within the app, this will instead open your mobile device's browser and run the web-app.

> Because your scripts are associated with your browser, any scripts you've written on your computer **won't** be on your mobile device, and any scripts you write on the mobile device **won't** appear on your computer.
>
> *The exception is if you're able to sign in which, as we saw in **Getting up and running**, is currently only available to teachers at registered schools.*

This is where Touch Develop starts to really shine as an editor – both CK Editor and Block Editor **work** on a touch-screen device, but Touch Develop works **better**, particularly on a device with a smaller screen. Try writing our flashing LED script in each of the three editors on your mobile device – odds are good that it'll be a more pleasant experience in Touch Develop.

> **Don't** try to use MicroPython – remember, it'll probably break your bluetooth pairing!

Once you're ready to flash your micro:bit (and are connected via bluetooth), tap on **Compile** in the editor on your mobile device. You'll see the normal download message appear, and then should see a screen asking you what action you want to take with the file – tap on **Open in micro:bit** and the file will be flashed to your micro:bit.

> There is also a **Flash** option in the mobile app menu – this will give you a choice of flashing the most recent script you used, or choosing from a list of your scripts. I've had mixed success – it seems to default to opening the CK Editor for some reason.

And that's all there is to coding your micro:bit using a mobile device!

Electronics projects

It's beyond the scope of this getting started guide, but those 5 large pins (and the 20 small ones in the edge connector!) are ideal for getting started with electronics. If this interests you, there are some great resources out there.

Block Editor

- **Hack your headphones** - microbit.co.uk/blocks/lessons/hack-your-headphones
- **Banana keyboard** - microbit.co.uk/blocks/lessons/banana-keyboard

Touch Develop

- **Hack your headphones** - microbit.co.uk/td/lessons/hack-your-headphones
- **Banana keyboard** - microbit.co.uk/td/lessons/banana-keyboard

MicroPython

- Several of the tutorials at microbit-micropython.readthedocs.io/ give examples of dealing with electronics – for instance **Input/Output**, **Music** and **Speech** sections
- Adafruit's MicroPython tutorial videos (youtube.com/user/adafruit/playlists) include some simple examples …
- And again, there are written tutorials to accompany the videos on their learning portal (learn.adafruit.com/)

If you're interested in doing more with electronics, take a look at the range of accessories (including dock connectors, a breakout board and an inventors kit) by **Kitronic**, available from **Pimoroni**.

Glossary

Compiler – a piece of software running on your computer which takes your script and converts it into a file of hexadecimal machine-code that the micro:bit can understand

Flash – when a compiled file is transferred to the micro:bit, it re-writes the flash memory on the device (so the code is still there once the power has been turned off and on again). This is also called **flashing** the device.

Interpreter – a piece of software (in this case running on the micro:bit) which takes your script and interprets it into machine-code instructions that the micro:bit can understand. The MicroPython **REPL** is an interpreter.

Making – a subculture interested in DIY as it relates to electronics, 3D printing, robotics and the like. Take a look at **Make:** magazine (makezine.com) and **Maker Faires** (makerfaire.com)

Microcontroller Board – an electronics board containing a microcontroller – essentially a small computer containing a processor, memory and input/output, all integrated into a single circuit (and known as a *System on Chip* or *SoC*). These are very popular, particularly in **Making** and electronics.

MicroPython REPL – a read-evaluate-print-loop, essentially a MicroPython console running on the micro:bit, available through USB, provided by an **Interpreter** on the micro:bit. Because it's interpreted, any code you run in the REPL will be slower than if you compile it – but it's much quicker to change code through the interpreter than a compiler, making this a better choice when testing out ideas.