Advanced features and techniques

Programming for beginners with the BBC micro:bit

John Davies

2026 May 13

1 Introduction

Introduction

Here is some more information that may help you to move forward and to understand how your program is executed by the computer in the micro:bit.

- A collection of tips and techniques for MakeCode.
- Text-based programming languages and MicroPython
- What goes on behind the scenes of your program?

A separate set of slides explains the use of functions but I cannot cover some topics that are vital for a realistic program. Here are a few.

- The most important is the array, which is a list of values. It might contain a sequence of temperatures as a function of time, for example, or a set of weights that you wish to analyse. MakeCode has special loops for processing lists.
- MakeCode has a Music toolbox for playing melodies and single notes. I loathe anything that makes sounds in (large) classes so I skipped this topic!
- The point of a computer like the micro:bit is to **embed** it in a larger system. An example could be a wheeled robot but we simply don't have time.

2 Tips and techniques

Duplicating programs

You often want to base a new program on an existing one, while keeping the original version.

1. Go to the Home screen of MakeCode (house icon) and click View All next to Projects.
2. Click on the program that you wish to use as a base and click the Duplicate button at top right (it looks like two overlapping sheets of paper if the window is too small to show the text).
3. Give the copy a new name.
4. Select the new program and click the Open button (pen and paper) to start editing the fresh copy.

Duplicating, deleting and moving blocks

Right-clicking on a block in MakeCode gives several handy options:

- Duplicate (including blocks enclosed by the selected block)
- Add Comment (see later)
- Collapse Block (or Expand Block if it has been collapsed to save space)
- Delete Blocks (including blocks enclosed by the selected block)
- Help

If you try to drag a block out of a 'container', such as a handler or if–then–else, it brings the blocks below with it. If you don't want this, you can duplicate the block to make another copy and delete the unwanted parts.

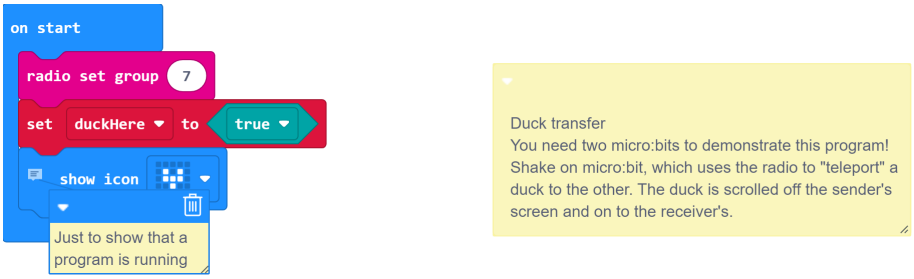
Right-clicking the background of the workspace brings up a different set of options. The Snapshot was very useful for writing these notes!

Comments

A well written program should include **comments** to explain features that are not obvious from the code itself. They are intended for humans that study the program and are ignored by the computer. To quote Commenting on your code,

> Adding comments to your code helps explain to other people reading your code what you have written in code and why you have written your code in a certain way.

Add comments to MakeCode by right-clicking either the background or a block.



Comments that explain weaknesses in the program are particularly useful.

Where do my files go?

- Files are stored on your computer if you use the **offline app**. On Windows it creates a directory Documents/MakeCode/microbit/, within which it creates a directory for each new project. I have no idea where they go on a Mac.
- I don't know where programs are stored if you use MakeCode through the **web browser** rather than the offline app. The support page on Saving projects says "The project files are actually stored as data in the browser's indexed data store".
  Make sure that you always log in with the same account to ensure that you save all your programs in the same place. (I have several Microsoft accounts on my computer, which is not unusual.)
- An **iPad** or **iPhone** stores the files in the app's private space, which is not visible from Files.
- I have no idea about **Android**!

Introduction to functions

A cookery book does not repeat the instructions for common requirements such as pastry every time they are needed. Instead, the recipe for apple pie says something like 'Make 200 g of short crust pastry (page 218)'.

The same is done with a **function** in computer programs:

- The instructions on page 218 are called the **definition** of the function (short crust pastry here).
- The recipe for apple pie **calls** the function and passes the **parameter** 200 g, which specifies how much pastry to make (although this is not always needed).

Functions are widely used in computer programs for two main reasons:

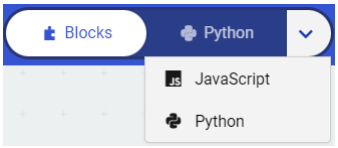
- to avoid repetition (as above)
- to make the program easier to write and understand by putting each distinct activity into a function

Functions in MakeCode are essentially blocks that you design yourself. They are explained in a separate set of slides.

3 Text-based programming languages

Text-based programming languages

If you have programmed in the past, you probably used a text-based language such as FORTRAN, BASIC, Pascal or C. The blocks in MakeCode are wonderful for constructing simple programs but may become frustrating if you want to write something more ambitious. Many other options are available and here are a few.



MakeCode has a selector at the top of the screen that allows you to switch to one of two text-based languages:

- **JavaScript** – the language used to support interactive web pages
- **Python** – a version of a widely used programming language

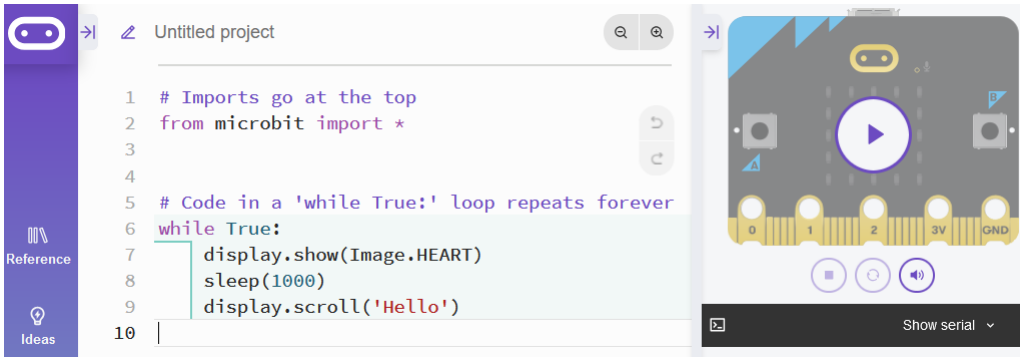
I recommend Python *but not this version*.

Both of these languages are designed to map on to the blocks used in MakeCode so that the user can switch seamlessly between them. Unfortunately this feature constrains the way in which the languages are used, which is why I do not recommend them.

MicroPython

Python is a widely used programming language for desktop computers while **MicroPython** is a version intended for microcontrollers like that on the micro:bit.

You can program in MicroPython using a web-based editor, which includes a simulator, but *this is completely separate from MakeCode.*



Ignore this section if the screenshot of the editor brings back nightmares! MicroPython offers more control than MakeCode and the functions that are equivalent to many blocks provide access to parameters that are not available in the blocks. For example, the brightness of the LEDs can be controlled rather than simply turning them on or off. The user guide is a good introduction with links to the reference manual.

An obvious question is why the micro:bit has two versions of Python. This is answered in the article MakeCode Python and MicroPython. To summarise, the fundamental issue is that Python is a general purpose language and does not include instructions to access the features of the micro:bit. These are added through specific software called an *application programming interface* (API) and the two versions of Python have implemented the API in different ways. The MakeCode version is designed to map on to blocks while that in MicroPython is more conventional, which is why I recommend it.

Many other languages are available but I have tried only these.

4 Behind the scenes

What lies between your code and the processor?

The processor in the nRF52833 is an ARM Cortex-M4F. This includes many advanced features but its basic instructions fall into three categories. They:

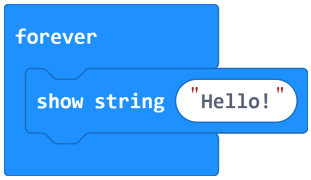
1. **move data** between the registers (fast memories) within the CPU and the main memory. This is like the RCL and STO keys on a pocket calculator.

2. **perform simple calculations** on data in the registers. Some operations are familiar, such as addition, while others are related to Boolean algebra (AND, OR etc).

3. **control the flow** of the program. These instructions support loops, if–then–else statements and so on.

How do our programs relate to these basic instructions?

Hello! program

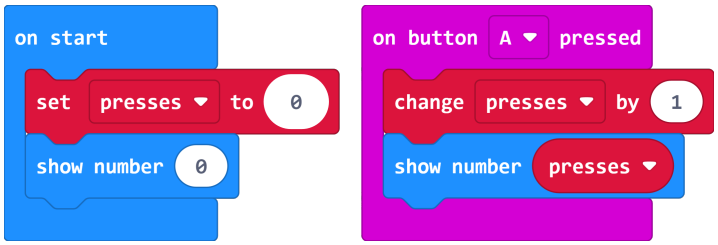


The forever block could translate directly into a basic instruction but what about show string? This block must:

1. convert the string Hello! into a pattern of dots to display on the LEDs, which requires a table to show the pattern for each character
2. put the first character on the 5×5 display
3. wait 150 ms then shift the pattern by one dot so that the text scrolls
4. check whether it has reached the end of the string, so the block has finished

A lot of software lies between show string and the CPU!

Event-driven program with button

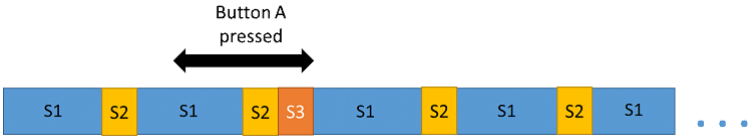


Software in the micro:bit checks the state of each button regularly (every 6 ms) and calls your **handler**, the code within on button A pressed, when an event is detected.

The **scheduler** is a vital piece of software that runs on the micro:bit 'behind the scenes' to ensure that all **tasks** receive their fair share of the CPU.

The scheduler is a central part of any operating system. Even a 'simple' system like the micro:bit is supported by a large library of software, without which it would not be possible to construct programs in the simple way that we have done.

Let's see how the scheduler manages the button and the display in this simple program. The following diagram is taken from The micro:bit – a reactive system. It shows the tasks being run as a function of time across the screen.



S1 task that scrolls the number of presses across the display

S2 task that checks the state of button A, called every 6 ms

S3 handler for button A, called when the button is pressed

This example is inaccurate because the scrolling function S1 runs every 150 ms, which is less often than the polling function S2, every 6 ms. It is possible to react to inputs in a completely different way, using a feature of the computer called an *interrupt*. This is used by the scheduler itself so that it can 'take over' the processor to run tasks at specific times.

Exactly what triggers on button A pressed is explained in the reference manual for On Button Pressed:

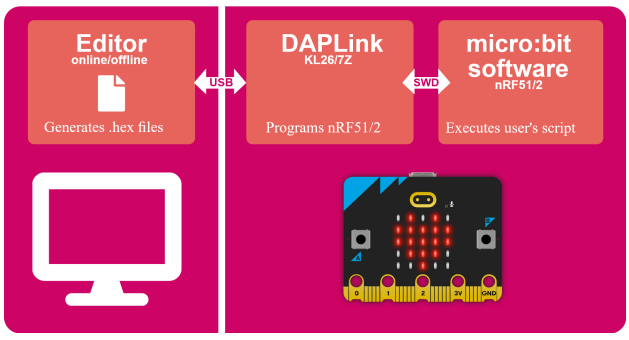
- For button A or B: This handler works when the button is pushed down and released within 1 second.
- For A and B together: This handler works when A and B are both pushed down, then one of them is released within 1.5 seconds of pushing down the second button.

So the event is not triggered by pressing the button but by *releasing* it, provided that you do so quickly enough.

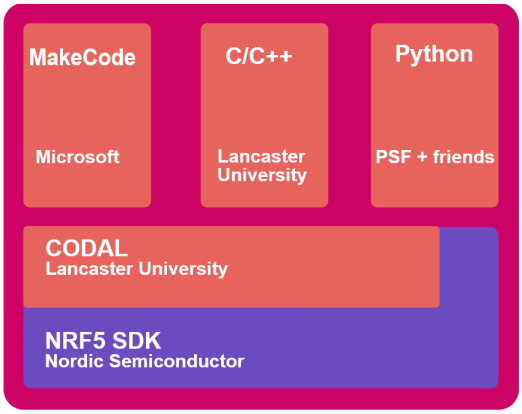
Another example of the scheduler in action is provided by the forever block, which runs its contents every 20 ms. It doesn't simply restart as soon as it gets to the end of its enclosed blocks because that would leave no time for other tasks to run, such as those that check the state of the buttons.

The ARM Cortex-M4F is capable of running a type of operating system called a *real time operating system* or RTOS. As well as a scheduler, this typically passes events to tasks, allows communication between tasks, manages memory and files, and allocates shared resources, such as communication interfaces. The term *real time* means that the system must respond to requests within a specified time so that urgent events are handled before any danger might arise. A small RTOS needs only a few kilobytes of code, a far cry from the operating system on a desktop computer.

The scheduler is only part of the software needed to run your programs on the micro:bit. Several pages on the micro:bit developer community web site show the relation between the various components. This diagram from The micro:bit software ecosystem shows how a program gets from your editor to the micro:bit.



Once on the micro:bit, your program is supported by software called the runtime or *Component Oriented Device Abstraction Layer* (CODAL).



All these layers of software serve two main purposes.

- They greatly simplify the user's programs and permit a clearer structure.
- They hide the details of the hardware from the programmer. This means that the same program could run on different versions of the micro:bit even if the processor were changed. (This has already happened with v1 and v2.)

That's enough!