Micro:bit Practice Laboratory

Bryan Ruddy

13 May 2022

In this week's lectures, we covered the representation of characters by binary digits. In this lab, we will use this new-found knowledge to make pretty pictures show up on a tiny computer. Specifically, we will introduce the BBC micro:bit, a very small computing device designed for education that nonetheless has very similar computer hardware to what you can find in an internet-connected light bulb or similar devices. For those unable to come to campus, and to ease development even for those who are, we will also introduce a web-based simulator for the micro:bit.

The BBC micro:bit

The BBC micro:bit is a small programmable computer. It comes with an on-board display, sensors, and some buttons. For this lab we will simply be interacting with the buttons and display - other functionality will be introduced later.

The micro:bit uses its own modules, with a special, simplified version of Python called MicroPython. You don't get the power of NumPy, but most core Python functionality is supported. Most modules come with documentation; the documentation for the micro:bit modules (and the rest of MicroPython) can be found online on their readthedocs.io page.

To run your code on the micro:bit, you will need to *flash* it to the device: load it from your own computer to it over a USB cable. Whenever you edit your code, you need to re-flash it to see any changes on the micro:bit.

Getting started

For the remaining labs in the course, you will be working with a partner; please choose your partner before signing out a micro:bit for the two of you.

The global chip shortage strikes again—we weren't able to buy any replacement micro:bits this year, so we unfortunately don't have enough for everyone to have a device of their very own.

Software tools

For the computer systems laboratories, we need to work with a Python development environment that supports the micro:bit, and PyCharm will not work. Fortunately, there are a variety of web-based Python development environments that do. The simplest and easiest one to use is produced officially by the micro:bit foundation: `https://python.microbit.org/v/2`. If you use Google Chrome, this editor can connect directly to your micro:bit over USB to load code. This editor comes with extensive and helpful documentation on how to use it; feel free to look through the documentation to figure out how to use the editor.

The official web-based editor does not include a micro:bit simulator, however. The course staff are also actively working to make the simulator support everything we need for the labs, and so there may be updates from week to week. To install the simulator, you will need to do the following:

1. Clone our Git repository somewhere useful on your computer, using the command
   `git clone https://github.com/bryan-ruddy/PythonEditor.git`
2. Navigate into the folder: `cd PythonEditor`
3. Run a command to update some of the simulator's components:
   `git submodule update --init --recursive`

To run the simulator, do the following:

1. Open a command prompt in the PythonEditor folder.
2. Run the simulator using the command
   `python -m http.server --bind 0.0.0.0`
   (If you get an error telling you Python can't be found, try using an Anaconda prompt for your command prompt.)
3. Access the simulator by pointing your web browser to
   `http://localhost:8000/editor.html`
4. When you're done, press CTRL-C in the command prompt to stop the simulator from running.

This command turns your computer into a web server, and is more reliable than just directly opening the HTML file with a browser.

You can also load code from the simulator to a real micro:bit, though the process is not as simple as when using the official editor.

The hardware

Your kit should include the following components:

- A micro:bit
- A very short microUSB cable
- A battery holder (either with batteries inside or accompanied by a pack of AAA batteries)
- Various safety manuals and paperwork

Carefully put the batteries and holder away in your kit; you will not need them for this lab. **Safety note:** Do not ever connect your micro:bit to the USB cable if it is also connected to a battery pack. Doing so is likely to damage the batteries, and may cause them to leak or explode.

Exercise 0

This is your basic 'Hello, world" program, just to get you comfortable with the software and hardware.

1. Run and open the simulator.
2. In the simulator, use the *Load* button to open `exercise_0.py`.
3. Click the *Sim* button to start a simulation of the micro:bit running the code. Observe the output.
4. Change the code so that it prints something else (your name, for example). To re-run the simulation, click *Sim* once to close the existing simulation, and then a second time to start a new one. *If you do not have a physical micro:bit, skip ahead to Exercise 1.*
5. Click the *Download* button to download a special `.hex` file to your computer that contains both the python code and the binary data representing the MicroPython interpreter.
6. Use the USB cable to connect your micro:bit to the computer. You should see it appear just like a flash drive.
7. Copy the `.hex` file onto the micro:bit, then observe what the micro:bit does. It should look a lot like the simulation!
8. To run your code again, press the reset button on the back of the micro:bit.
9. Open the official online editor. It will look very similar to the simulator, but with different buttons at the top.
10. Use the *Load/Save* button to load `exercise_0.py`.
11. Change the code again, so the text it scolls is something new.
12. Use the *Connect* button to make the micro:bit available to the editor.
13. You should now see a *Flash* button at the top of the editor—click it to load your code onto the micro:bit. You should see it scroll your new text.

The simulator is based on a version of the official editor from at least 3 years ago; you can follow the chain of open-source development on GitHub if you're interested.

Congratulations—you have now tried several different ways of writing code and transferring it to your device. From now on, choose the option you find most convenient and that works best for you.

Exercise 1

Here, we will introduce the button interface for the microbit.

1. Run the code from `exercise_1.py` on the micro:bit and/or simulator. Press the left button (`button_a`), and observe the output.
2. Implement a similar display for the right button (`button_b`).

Exercise 2

The objective of this exercise is to make an ASCII character appear based on the interpretation of the input of the buttons. We will be using an 8-bit representation of an ASCII character. The input of these 8 bits will be done with the two buttons on the micro:bit, where one will correspond to zero and one will correspond to one.

You will write code to translate the inputs to a displayable figure, and we have provided you a template to start from in `exercise_2.py`. Ensure that your code reads bits in the order you expect—either smallest first or largest first.

For example:

```
Input: [0]
Interpretation: 0×2^0
Value: 0

Input: [1, 0]
Interpretation: 0×2^0 + 1×2^1
Value: 2

Input: [1, 0, 0]
Interpretation: 0×2^0 + 0×2^1 + 1×2^2
Value: 4
```

This can be generalised into a for loop (which you will be implementing), when the input is arbitrary.

Questions to Ponder

This lab is not assessed, but you might find it helpful to reflect on these questions:

1. In Exercise 2, what happens when you enter an input over the value of 127? What happens when it is under 33? Why is this being produced?
2. Is entering bits in one-by-one an efficient way of inputting characters? If not, how else could these characters be represented to make it more efficient? If so, why?
3. Discuss the merits of using Python to control the display of a tiny computer. Do you think that it poses significant risk, especially since many critical information systems depend on similar code on a daily basis?

Submission Instructions

This lab is not assessed, so no need to submit anything!