
Jumpy Bun - CircuitPython Game

Mr. Coxall

Jan 23, 2020

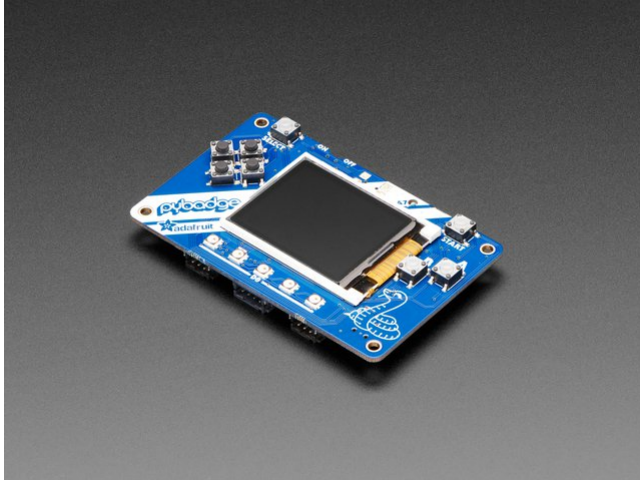
Contents

| | | |
|----------|------------------------------|-----------|
| 1 | Install CircuitPython | 5 |
| 2 | Your IDE | 7 |
| 2.1 | Hello, World! | 8 |
| 3 | Image Banks | 11 |
| 4 | Game | 13 |
| 4.1 | Background | 13 |
| 4.2 | Space Ship | 13 |
| 5 | Menu System | 15 |
| 5.1 | Start Scene | 15 |
| 5.2 | Splash Scene | 15 |
| 5.3 | Game Over Scene | 15 |

In this project we will be making an old school style video game for the [Adafruit PyBadge](#). We will be using [CircuitPython](#) and the [stage library](#) to create a [Sheep it Up](#) like game. The stage library makes it easy to make classic video games, with helper libraries for sound, sprites and collision detection. The game will also work on other variants of PyBadge hardware, like the [PyGamer](#) and the [EdgeBadge](#). The full completed game code with all the assets can be found [here](#).

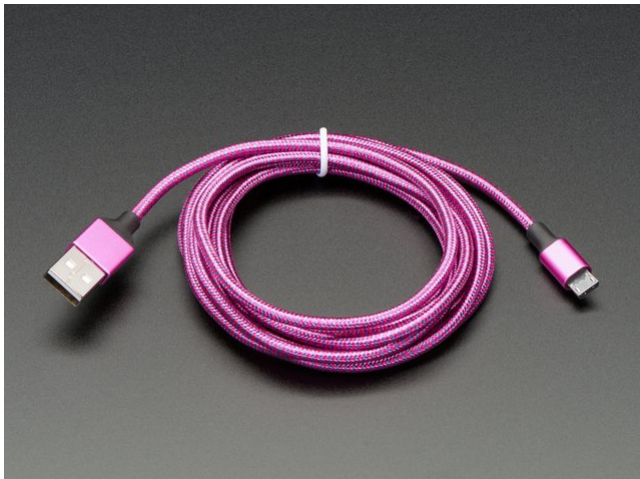
Parts

You will need the following items:



Adafruit PyBadge for MakeCode Arcade, CircuitPython or Arduino

PRODUCT ID: 4200

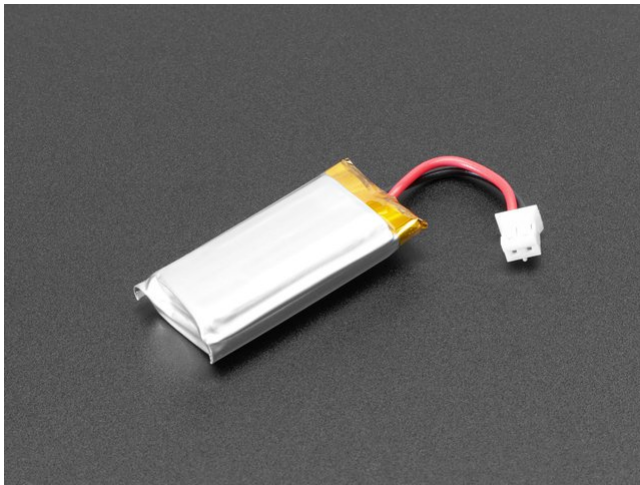


Pink and Purple Braided USB A to Micro B Cable - 2 meter long

PRODUCT ID: 4148

So you can move your CircuitPython code onto the PyBadge.

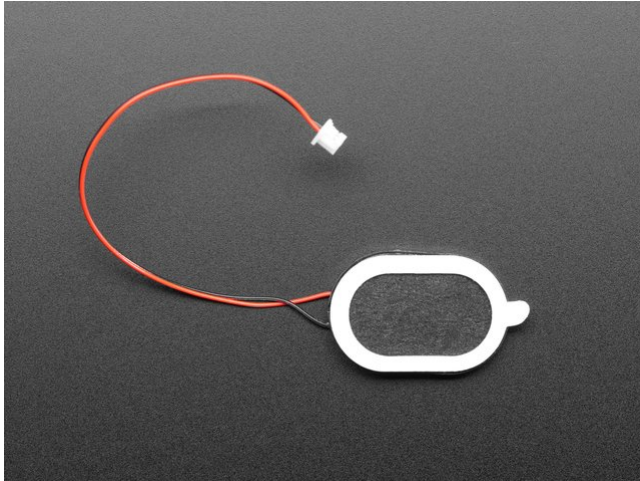
You might also want:



Lithium Ion Polymer Battery Ideal For Feathers - 3.7V 400mAh

PRODUCT ID: 3898

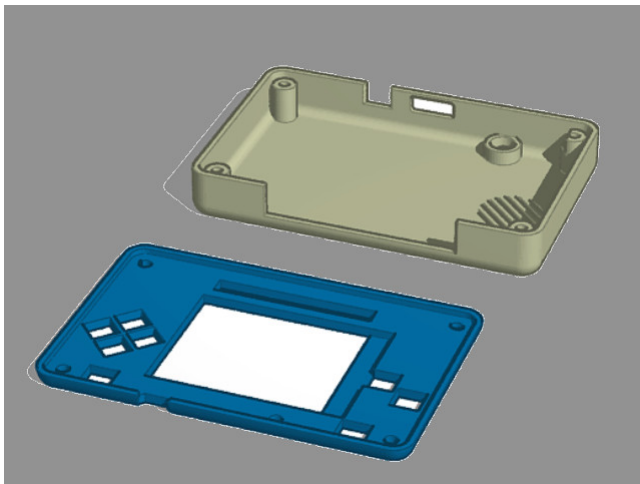
So that you can play the game without having it attached to a computer with a USB cable.



Mini Oval Speaker - 8 Ohm 1 Watt

PRODUCT ID: 3923

If you want lots of sound. Be warned, the built in speaker is actually pretty loud.



3D Printed Case

I did not create this case. I [altered Adafruit's design](#). One of the screw posts was hitting the built in speaker and the case was not closing properly. I also added a piece of plastic over the display ribbon cable, to keep it better protected. You will need 4 x 3M screws to hold the case together.

Install CircuitPython

Fig. 1: Clearing the PyBadge and loading the CircuitPython UF2 file

Before doing anything else, you should delete everything already on your PyBadge and install the latest version of CircuitPython onto it. This ensures you have a clean build with all the latest updates and no leftover files floating around. Adafruit has an excellent quick start guide [here](#) to step you through the process of getting the latest build of CircuitPython onto your PyBadge. Adafruit also has a more detailed comprehensive version of all the steps with complete explanations [here](#) you can use, if this is your first time loading CircuitPython onto your PyBadge.

Just a reminder, if you are having any problems loading CircuitPython onto your PyBadge, ensure that you are using a USB cable that not only provides power, but also provides a data link. Many USB cables you buy are only for charging, not transferring data as well. Once the CircuitPython is all loaded, come on back to continue the tutorial.

CHAPTER 2

Your IDE

One of the great things about CircuitPython hardware is that it just automatically shows up as a USB drive when you attach it to your computer. This means that you can access and save your code using any text editor. This is particularly helpful in schools, where computers are likely to be locked down so students can not load anything. Also students might be using Chromebooks, where only “authorized” Chrome extensions can be loaded.

If you are working on a Chromebook, the easiest way to start coding is to just use the built in [Text app](#). As soon as you open or save a file with a *.py extension, it will know it is Python code and automatically start syntax highlighting.

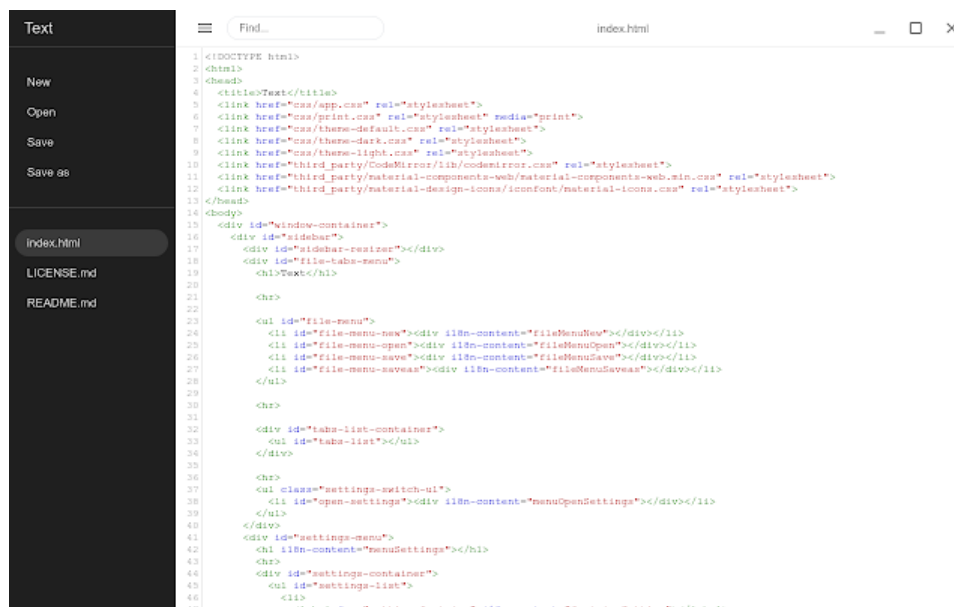


Fig. 1: Chromebook Text app

If you are using a non-Chromebook computer, your best bet for an IDE is [Mu](#). You can get it for Windows, Mac, Raspberry Pi and Linux. It works seamlessly with CircuitPython and the serial console will give you much needed debugging information. You can download Mu [here](#).



Fig. 2: Mu IDE

Since with CircuitPython devices you are just writing Python files to a USB drive, you are more than welcome to use any IDE that you are familiar using.

2.1 Hello, World!

Yes, you know that first program you should always run when starting a new coding adventure, just to ensure everything is running correctly! Once you have access to your IDE and you have CircuitPython loaded, you should make sure everything is working before you move on. To do this we will do the traditional “Hello, World!” program. By default CircuitPython looks for a file called `code.py` in the root directory of the PyBadge to start up. You will place the following code in the `code.py` file:

```
1 print("Hello, World!")
```

As soon as you save the file onto the PyBadge, the screen should flash and you should see something like:

Although this code does work just as is, it is always nice to ensure we are following proper coding conventions, including style and comments. Here is a better version of Hello, World! You will notice that I have a call to a `main()` function. This is common in Python code but not normally seen in CircuitPython. I am including it because by breaking the code into different functions to match different scenes, eventually will be really helpful.

```
1 #!/usr/bin/env python3
2
3 # Created by : Mr. Coxall
4 # Created on : January 2020
5 # This program prints out Hello, World! onto the PyBadge
6
7
```

(continues on next page)

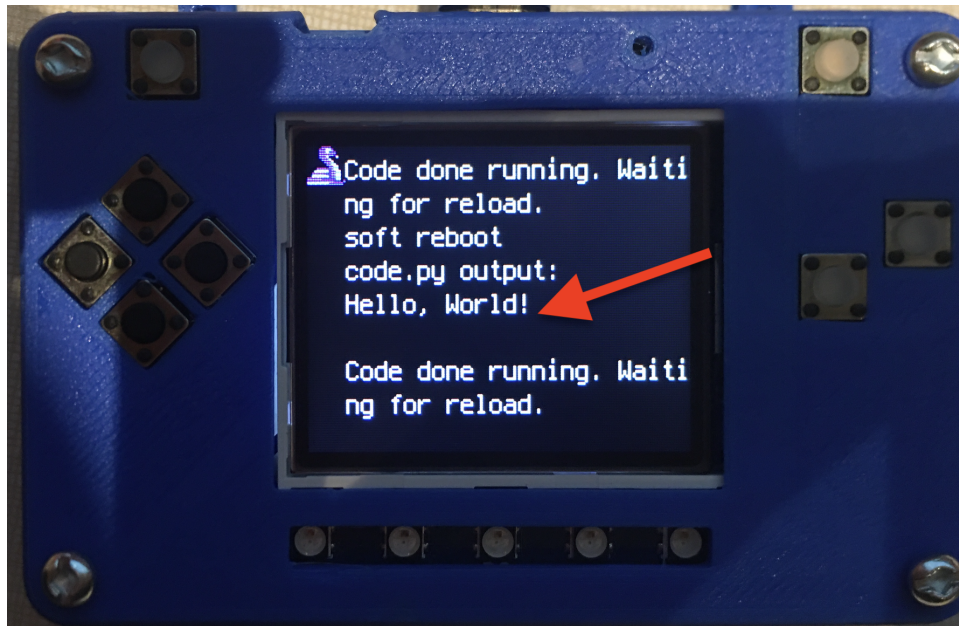


Fig. 3: Hello, World! program on PyBadge

(continued from previous page)

```
8 def main():
9     # this function prints out Hello, World! onto the PyBadge
10    print("Hello, World!")
11
12
13 if __name__ == "__main__":
14    main()
```

Congratulations, we are ready to start.

CHAPTER 3

Image Banks

Before we can start coding a video game, we need to have the artwork and other assets. The stage library from CircuitPython we will be using is designed to import an “image bank”. These image banks are 16 sprites staked on top of each other, each with a resolution of 16x16 pixels. This means the resulting image bank is 16x256 pixels in size. Also the image bank **must** be saved as a 16-color BMP file, with a pallet of 16 colors. To get a sprite image to show up on the screen, we will load an image bank into memory, select the image from the bank we want to use and then tell CircuitPython where we would like it placed on the screen.

For sound, the stage library can play back *.wav files in PCM 16-bit Mono Wave files at 22KHz sample rate. Adafruit has a great learning guide on how to save your sound files to the correct format [here](#).

If you do not want to get into creating your own assets, other people have already made assets available to use. All the assets for this guide can be found in the GitHub repo [here](#):

- [space aliens image bank](#)
- [coin sound](#)
- [pew sound](#)
- [boom sound](#)
- [crash sound](#)

Please download the assets and place them on the PyBadge, in the root directory. Your previous “Hello, World!” program should restart and run again each time you load a new file onto the PyBadge, hopefully with no errors once more.

Assets from other people can be found [here](#).

X

4.1 Background

X

4.2 Space Ship

X

X

5.1 Start Scene

X

5.2 Splash Scene

T

5.3 Game Over Scene

T