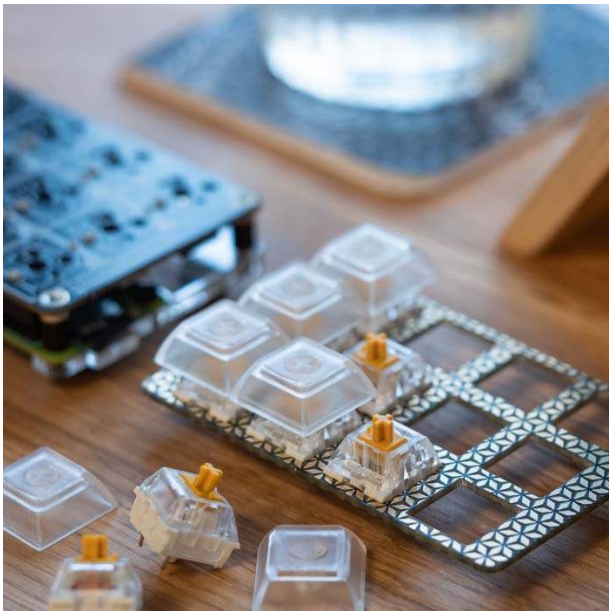
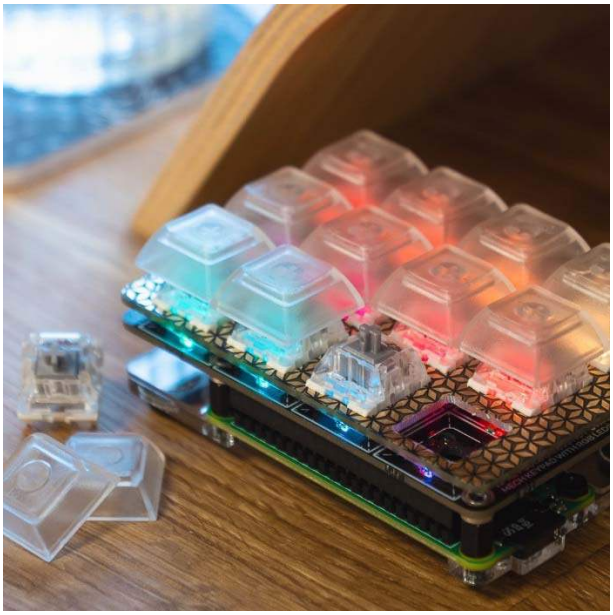
PIMORONI
Purveyors of fine Maker goods, worldwide





Keybow Mini Mechanical Keyboard Kit

- PIM420

Keybow is an easy-to-build, solderless, DIY mini mechanical keyboard. It's Raspberry Pi-powered, with twelve illuminated keys, hot-swap clicky or linear switches, clear keycaps, and awesome customisable layouts and macros. It's the ultimate macro pad.

This kit has everything you need* to build your own mini mechanical keyboard. It's a fun, affordable, first step into the world of mechanical keyboards, with high-quality clicky (Gold) or linear (Silver) Kailh Speed switches and clear DSA-profile key caps that look incredible when lit up with the per-key RGB lighting. The fancy hot-swap Kailh sockets mean that there's absolutely no soldering required!

Kit includes*

- Raspberry Pi Zero WH
- Keybow PCB
- Switch plate
- Twelve Kailh Speed switches (Gold or Silver)
- Twelve clear DSA-profile key caps
- Acrylic baseplate
- Fixings and feet
- Micro-USB cable
- Comes in a reusable kit box

**Just add your own micro-SD card*

Use Keybow as a hotkey pad for your favourite program like Adobe Lightroom, a custom games controller, to trigger clips, tracks, or effects in Ableton Live, or to paste frequently-used text or code snippets. So, if you want to open your web browser and search for cat GIFs all with a single keypress, we've got you covered. Because all your key and lighting customisation is stored on the device, it's completely portable too, meaning you can switch your setups between any machine you like.

Keybow is powered by a Raspberry Pi Zero WH (with pre-soldered header), and uses the Zero's USB HID gadget mode so that it appears as a real keyboard when plugged into your computer with the included USB cable. We've built a completely custom, stripped-down, RAM-disk-based Keybow OSwith a Lua interface to customise the layout and lighting on your Keybow. It's Windows, Mac, and Linux-compatible.

Keybow features

- Per-key RGB LEDs (APA102)
- Kailh hot-swap switch sockets (for Cherry MX-compatible switches)
- 40-pin female header
- I2C breakout header for add-ons
- Custom Keybow OS
- Compatible with Raspberry Pi 3B+, 3, 2, B+, A+, Zero, and Zero W
- Assembled size: 85x56.5x38mm

Here's a little video of us using Keybow to control Ableton Live, muting tracks and effects, and switching between plugins.

Construction

The Keybow PCB has a 40-pin female header, like a regular Pi HAT, that plugs onto the 40-pin male header on the included Raspberry Pi Zero WH. The Pi is attached to the acrylic baseplate and shim, and the whole thing is rigidly held together by metal standoffs. Rubber feet on the baseplate stop Keybow from slipping around on your desk.

We've got a full tutorial on how to assemble your Keybow here.

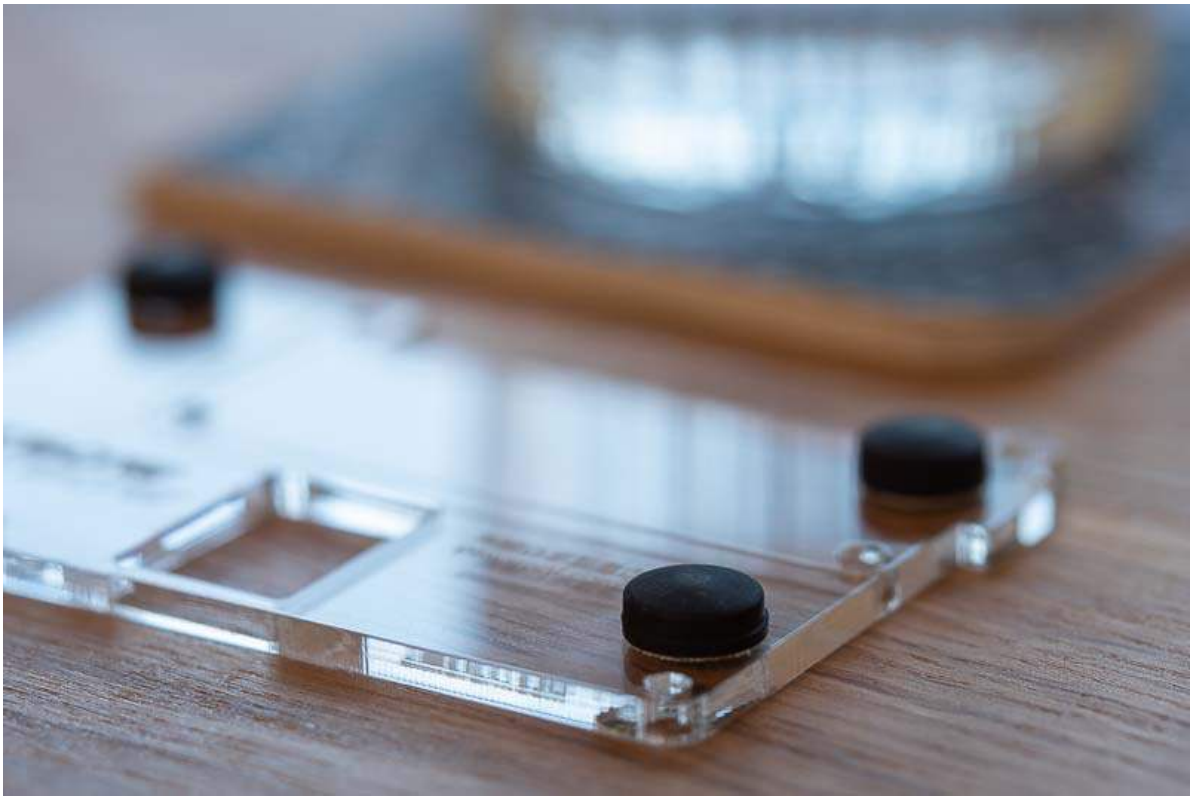
Assembling Keybow

In this tutorial, you'll learn how to assemble your Keybow mini mechanical keyboard. Assembly should take 15 to 20 minutes, and the only tool you'll need is a Phillips screwdriver. We'll fit the Raspberry Pi Zero WH to the acrylic baseplate first, then fit the Keybow PCB, and last of all fit the switches and key caps.

Attaching the Raspberry Pi Zero WH to the baseplate

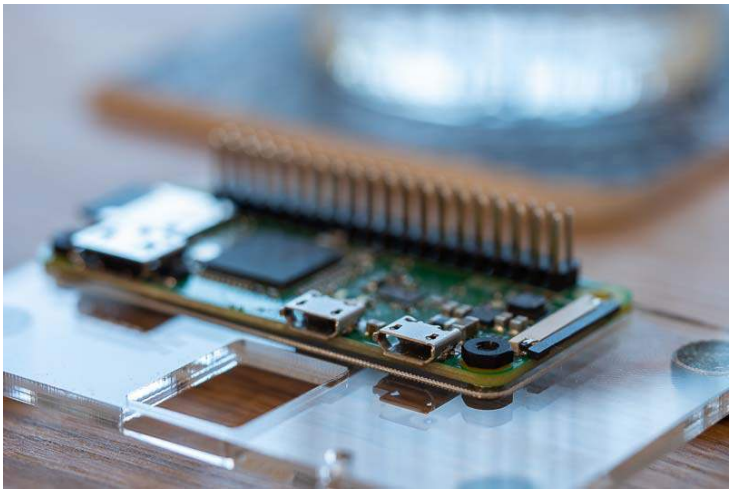
We'll begin by attaching the rubber feet to the thicker acrylic plate.

Peel the protective film off the two acrylic pieces very carefully. The thinner piece is especially fragile, so be really careful when peeling the film off this one. Take the thicker piece and turn it so that you can read the Keybow text. Stick the four rubber, self-adhesive feet to the acrylic where the four outlined circles are.



Next, we'll attach the Raspberry Pi Zero WH to the acrylic baseplate.

Remove the Zero WH from its antistatic bag. Flip the baseplate over, so that the rubber feet are now sitting on the surface on which you're working, and the outline of the Zero WH is at the top left corner. Sit the Zero WH on this space, with the GPIO pins towards the top of the acrylic baseplate (the solder joints underneath the Zero WH's header should sit neatly in the cutout at the top). Use two of the M2.5 metal screws and two of the plastic nuts to attach the Zero WH using the *bottom* pair of mounting holes (the ones further away from the Zero WH's GPIO pins).

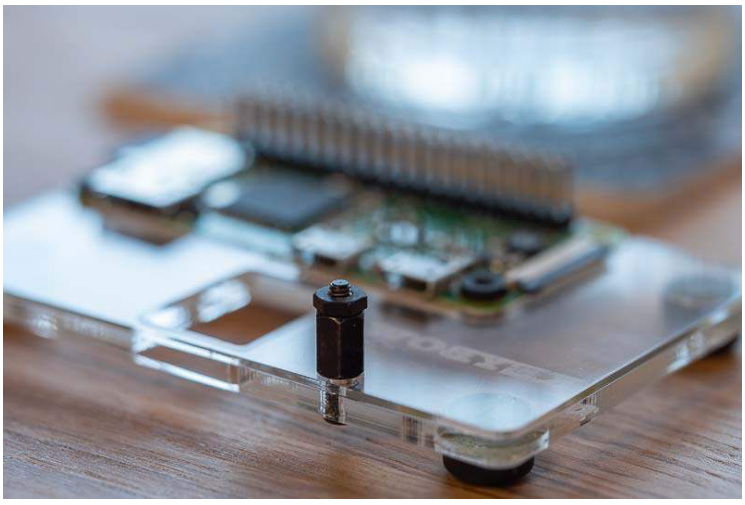


The thinner acrylic piece is a shim layer that levels up the metal standoffs that attach the Keybow PCB to the baseplate and Zero WH, so that they all sit at the same height. We'll fit it now, using one of the standoffs.

Slot the shim layer next to the Zero WH on the baseplate; it'll only fit one way. Take one of the metal standoffs and one of the metal M2.5 screws. Push the screw through the hole at the bottom right corner of the baseplate and shim layer, from below, then screw the female end of the metal standoff onto the screw.

Take care not to overtighten any of the metal screws, as you'll risk cracking the acrylic.

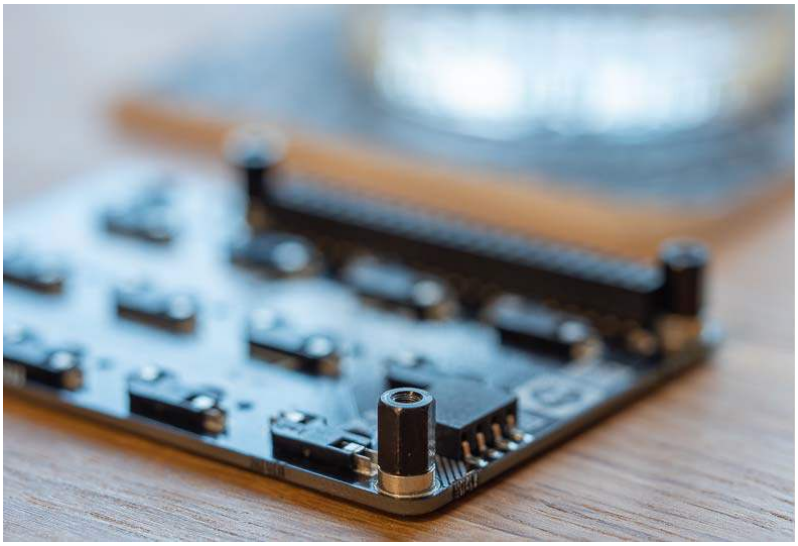
Lastly, screw the remaining plastic nut all the way onto the male thread on the top of the standoff that you just fitted. This will sit in the mounting hole on the Keybow PCB that doesn't have a threaded metal post and keep that corner level.



Don't fit any of the other standoffs to the acrylic baseplate yet!

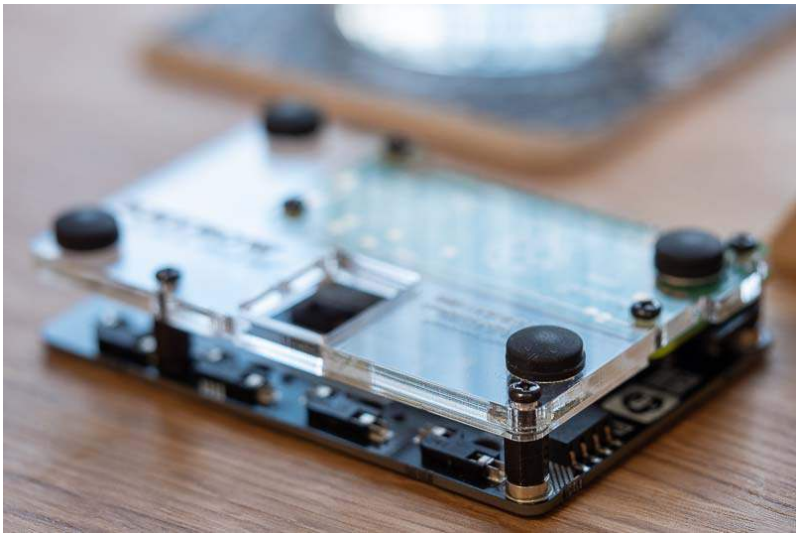
Attaching the Keybow PCB

The three other mounting holes on the Keybow PCB have threaded metal posts. We'll be screwing the male-threaded ends of the metal standoffs into these metal posts. Peel off the little amber pieces of protective film off the posts and screw in the standoffs.



You can now push your Keybow PCB's female GPIO header down onto the male GPIO pins on the Zero WH. Make sure that all the pins are lined up correctly. There will be a little gap left between the headers, but don't worry because they'll still be making good electrical contact.

Use the remaining three metal M2.5 screws to attach the standoffs to the acrylic baseplate, again taking care not to overtighten them and crack the acrylic.

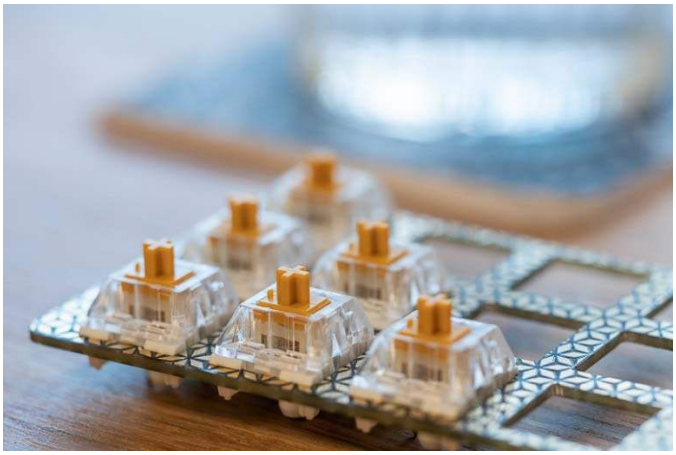


Mounting the switches and key caps

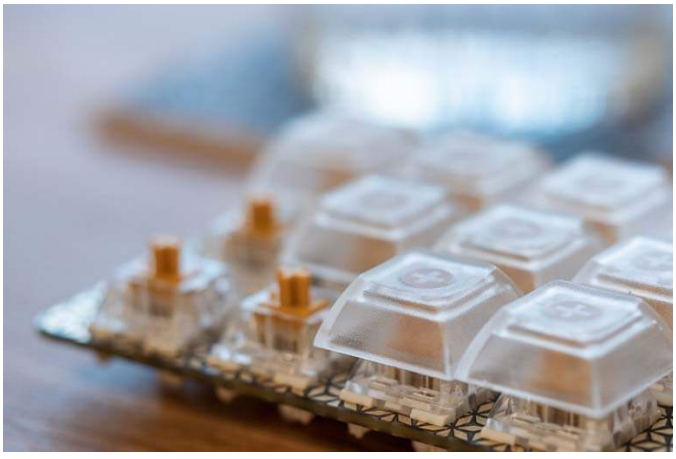
The switches push tightly into the PCB switch plate, and then the whole plate with switches mounted pushes down onto the Keybow PCB, with the pins on the switches being gripped in the hot-swap sockets.

It's important that you orient the switches the right way round when pushing them into the plate. If you look carefully at the switches, you'll see they have a little cavity underneath on one edge. If you turn the switch plate so that the black and gold side is facing upwards, and the KEYBOW text is at the right hand side, then the cavities on the switches should all be at the top.

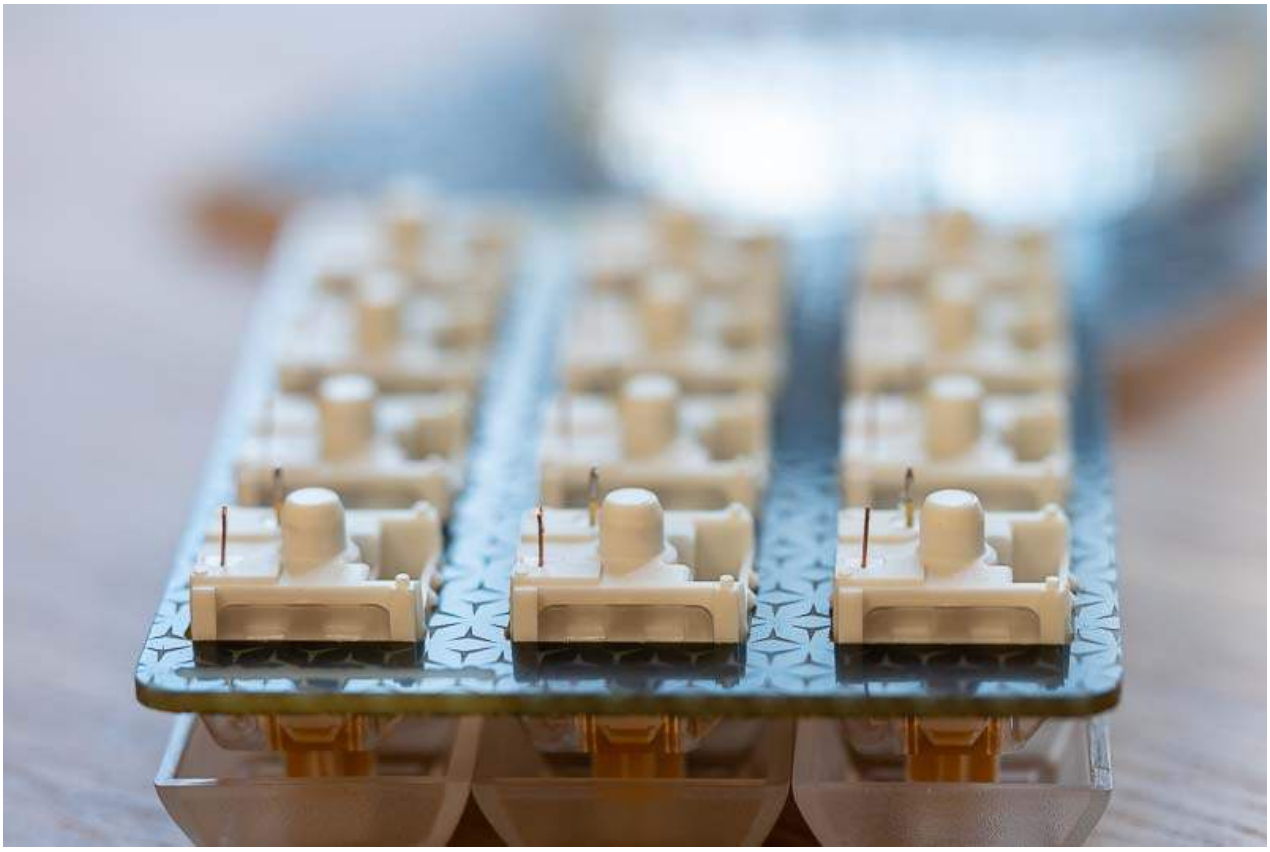
Push each switch into the plate, so that they sit flush. They're quite a tight fit, but they should click in when they're properly fitted.



Next, we'll mount the key caps. It doesn't matter which way round they go, as they're completely symmetrical. Push them all the way down onto the stems on the switches.



Flip the switch plate with switches and key caps mounted and take a look at all of the pins on the switches. Sometimes, they can get bent slightly in transit, but they all need to be straight to fit correctly into the hot-swap sockets. You can gently bend them back into position if you need to.



Turn your Keybow PCB assembly and switch plate so that the KEYBOW logos are both at the same side. Align the two pieces with each other, and gently sit the switch plate and switches in the correct location with the pins on the switches in the sockets on the PCB. Once you're happy that they're all correctly aligned, then push the switches down into the sockets. It's best to hold the whole thing at both sides and apply even pressure, so that they all go in straight. The bottoms of all of the switches should sit flush with the Keybow PCB.

Next steps

The next step is to set up the Keybow software and customise your key mappings. We'll cover all of that in the Setting up the Keybow OS tutorial.

Setting up the Keybow Operating System (OS)

In this tutorial, we'll look a little at how the Keybow OS works (you don't really need to worry about this bit too much, but you might find it interesting and useful), then we'll get onto how to set up a micro-SD card with the software. Last, we'll look at simple examples of how to customise the keys and lights on your Keybow.

How it all works

The Keybow OS is a custom OS that runs on the Raspberry Pi Zero WH that's included in your Keybow kit. It consists of a few different unique parts:

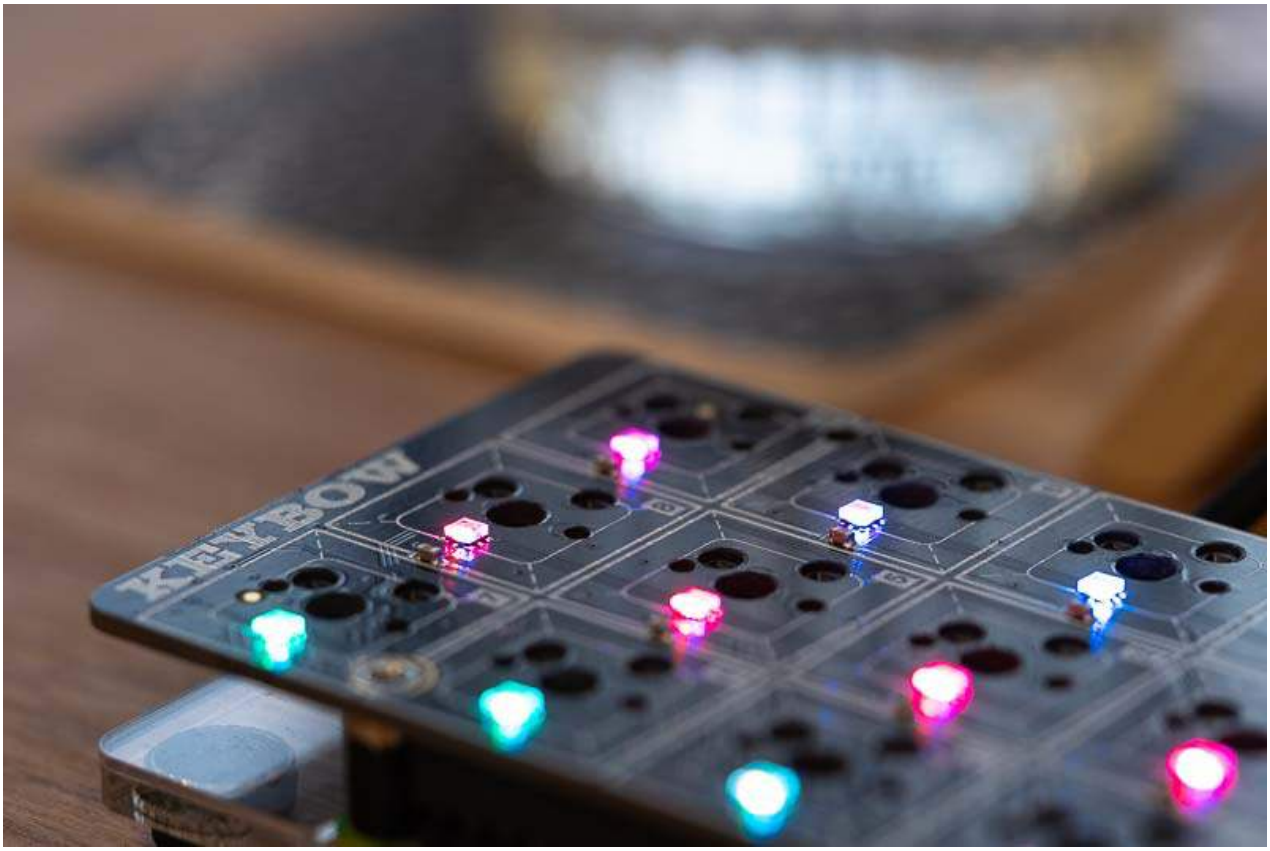
1. A stripped-down OS based on Raspbian
2. A ramdisk, into which the entire OS is loaded at boot
3. Lua files for setting the key mappings, layouts, and lighting
4. Some C code behind the scenes that sets up and controls the USB HID that Keybow uses to act as a keyboard

The ramdisk means that once the OS is loaded into it during boot, the SD card is no longer accessed or required. This avoids any problems with SD card corruption that can occur when you unplug your Raspberry Pi without safely shutting down first.

Lua is a basic but powerful programming language that works really well on embedded and low-power devices. It has a clean, simple syntax and is ideal for the interface to Keybows layouts and lighting.

Each of the twelve keys on Keybow is linked to a single GPIO pin on the Pi Zero W, and the APA102 LEDs to another two pins (the SPI pins). When a key is pressed, the change on that GPIO pin is detected, translated to a HID keycode by the Lua and C code, then sent out through the USB on-the-go port on the Pi Zero W and to the connected computer.

The LEDs under each key are addressable and in a single chain, so they can be individually controlled. We've got two different ways of setting the LEDs: individually in the `keys.lua` file, or by creating a 24-bit PNG file that will be animated across the keys (more on that later).



Setting up your Keybow OS micro-SD card

Because the Keybow OS is so stripped-down and small, you can get away with using a fairly low capacity micro-SD card; anything bigger than 1GB will be plenty.

Pop your micro-SD card into your computer, and format it in FAT32 format. We recommend using the SD Association's SD Memory Card Formatter app, which is cross-platform and generally very reliable. You can choose "quick format", and call the card whatever you like, e.g. "keybow".

Download the zip file with the Keybow OS on by clicking here. Unzip it, then drag all of the files *inside* the "sdcard" folder (i.e. not the folder itself) across to the micro-SD card.

Booting your Keybow for the first time!

Eject the prepared micro-SD card and pop it into the micro-SD card slot on the Pi Zero WH. Plug the micro-B end of the USB cable into the on-the-go port on the Pi Zero WH; it's the one that lines up with the cut-out in the Keybow baseplate.

Assuming you've prepared your SD card properly, after about 10-15 seconds, you should see the lights on your Keybow animating through some blue, pink, and orange hues. You might see a few LEDs light randomly when you first plug the power in; this is normal and will disappear once it's fully booted.

If you don't see the keys lighting up and animating, then something has gone wrong! Try the following troubleshooting tips:

- Did you definitely format your micro-SD card with a single FAT32 partition?
- Make sure that the files on the SD card aren't inside a folder
- Make sure that you copied all of the files across from the unzipped folder
- Is the micro-SD card plugged in fully, and is the micro-USB cable plugged in fully?
- Is the Keybow PCB pushed onto the Pi's pins as far as it will go? It should look like the picture below
- Is the green activity LED blinking on your Pi Zero WH? If not, then it should be!
- Try a different micro-SD card, if you have one
- Remove the baseplate, and plug a mini HDMI cable in. When you plug the power into the Pi Zero WH, you should see a bunch of text appearing on the screen.

If, after all of those steps, your Keybow is still not working, then pop a post on our forums at https://forums.pimoroni.com and we'll do our best to get it working!

How the key mappings work

By default, Keybow is set up as a number pad with the numbers 0-9, full stop, and enter, but you can easily customise the twelve keys on Keybow to be whatever key you like. To do this, you'll need to unplug the USB cable from your Keybow and pop the micro-SD card out and into your computer.

For most users, the only files you'll need to worry about are the `keys.lua` file and the layout files in the `layouts` folder. The `layouts` folder contains some example layouts, with the `default.lua` layout being the default numberpad.

You'll see that there's a line that says `require "layouts/default"` towards the top of the `keys.lua` file. This is linking and enabling the default mapping in `layouts/default.lua`. Let's look at that `default.lua` file:

```
require "keybow"

-- Standard number pad mapping --

-- Key mappings --

function handle_key_00(pressed)
    keybow.set_key("0", pressed)
end

function handle_key_01(pressed)
    keybow.set_key(".", pressed)
end

function handle_key_02(pressed)
    keybow.set_key(keybow.ENTER, pressed)
end

function handle_key_03(pressed)
    keybow.set_key("1", pressed)
end
```

```
function handle_key_04(pressed)
    keybow.set_key("2", pressed)
end

function handle_key_05(pressed)
    keybow.set_key("3", pressed)
end

function handle_key_06(pressed)
    keybow.set_key("4", pressed)
end

function handle_key_07(pressed)
    keybow.set_key("5", pressed)
end

function handle_key_08(pressed)
    keybow.set_key("6", pressed)
end

function handle_key_09(pressed)
    keybow.set_key("7", pressed)
end

function handle_key_10(pressed)
    keybow.set_key("8", pressed)
end

function handle_key_11(pressed)
    keybow.set_key("9", pressed)
end
```

You'll see that there are twelve functions, one for each key on Keybow. They're numbered `00`, `01`, and so on, up to `11`. The numberings go from bottom left to top right, and start at the bottom left key when Keybow is in portrait orientation with the USB cable at the right hand side.

Each function has a single line inside saying e.g. `keybow.set_key("0", pressed)`, which means "set this key to send 0 when pressed". You can change this to any number, letter, punctuation mark, or other character on a standard keyboard (we'll get to special keys like space, control, tab, enter, and so on, in a moment).

Changing the `pressed` at the end of those lines to `not pressed`, as follows: `keybow.set_key("0", not pressed)`, will cause the 0 to be sent when the key is released rather than pressed. You can have a `pressed` and `not pressed` sent by a single key, just by putting two lines within the function, like this, which will send 0 when the key is pressed and 1 when it is released again:

```
function handle_key_00(pressed)
    keybow.set_key("0", pressed)
    keybow.set_key("1", not pressed)
end
```

Take a look at the `handle_key_02` function now. You'll see that, rather than having a character like `"0"` in quote marks, it has `keybow.ENTER`. There are a number of special keys defined as variables in the `keybow.lua` file, that you can map to keys on Keybow:

```
keybow.LEFT_CTRL
keybow.LEFT_SHIFT
keybow.LEFT_ALT
keybow.LEFT_META

keybow.RIGHT_CTRL
keybow.RIGHT_SHIFT
keybow.RIGHT_ALT
keybow.RIGHT_META

keybow.ENTER
keybow.ESC
keybow.BACKSPACE
```

```
keybow.TAB
keybow.SPACE
keybow.CAPSLOCK

keybow.LEFT_ARROW
keybow.RIGHT_ARROW
keybow.UP_ARROW
keybow.DOWN_ARROW

keybow.F1
keybow.F2
keybow.F3
keybow.F4
keybow.F5
keybow.F6
keybow.F7
keybow.F8
keybow.F9
keybow.F10
keybow.F11
keybow.F12
```

Note that you don't need quote marks around these, as they're variables.

Creating a custom key layout

Our recommended way of creating your own custom layout is to create a new lua file inside the `layouts` folder, for instance `mylayout.lua`, and then link that layout in the `keys.lua`, so your `keys.lua` file would look like this:

```
require "keybow"

-- require "layouts/default" -- Numberpad

-- Custom layouts (uncomment to enable) --
```

```
-- require "layouts/boilerplate" -- Handy bits of boilerplate text like Lorem Ipsum
-- require "layouts/lightroom" -- Handy hotkeys for Adobe Lightroom Classic CC
-- require "layouts/pico8" -- Controls for Pico-8

require "layouts/mylayout" -- My custom layout
```

Notice that we've commented out the line that was linking the `default` layout by adding two hyphens at the beginning of the line, and that we've left off the `.lua`from the end of the `mylayout.lua` filename when linking it on the `require...`line.

It's important that you only have one layout file linked (i.e. uncommented) in your `keys.lua` file at any one time, or things will get crazy!

To see some examples of custom layouts, you can look at the other layout files in the `layouts` folder, like `lightroom.lua` and `pico8.lua`. There's also a blank layout, `blank.lua`, that you can use as a template.

Here are the ready-made layouts that we've included with Keybow OS:

- Default numberpad - numbers 0-9, full stop, and enter
- Boilerplate - example of how to enter whole strings of text like Lorem Ipsum placeholder text, or frequently-used code like the Python shebang
- Lightroom - hotkeys for Adobe Lightroom Classic CC
- Pico-8 - a Pico-8 gamepad with directional, action, and various function keys

Customising the lighting on Keybow

There are two ways to customise the lighting on Keybow.

Using a PNG image

The first, and easiest, way to customise the lighting is with a 24-bit PNG file. If you create a PNG file that is 12 pixels wide, then the colours of those 12 pixels will be mapped to each of the 12 LEDs under Keybow's keys. If you make the PNG file taller than 1 pixel, then this will create an animation on each LED, cycling through the colours in each column of the image from top to bottom, then looping back to the top. The animations are displayed at 60fps.

If you create an image that is just 1 pixel wide and several pixels tall, then the animation will be duplicated to all of the LEDs on Keybow.

The pattern that is used by default is `default.png` on the Keybow micro-SD card, so you can change the pattern simply by replacing that file with another called`default.png`. We've put a bunch of example patterns in the `patterns` folder.

Note that if you want your animation to loop smoothly, then you'll have to make the pixels at the very bottom of you PNG match up with those at the very top.

Manually setting each LED

The second method of customising the LEDs on your Keybow is by setting them manually is the `setup` function of your layout file. Here's an example from our Pico-8 controller layout:

```
function setup() -- Set custom lights up
    keybow.auto_lights(false)
    keybow.clear_lights()
    keybow.set_pixel(0, 255, 255, 0) -- Green
    keybow.set_pixel(1, 255, 255, 0) -- Green
    keybow.set_pixel(2, 0, 255, 255) -- Cyan
    keybow.set_pixel(3, 255, 0, 255) -- Magenta
```

```
    keybow.set_pixel(4, 0, 255, 255) -- Cyan
    keybow.set_pixel(5, 0, 255, 255) -- Cyan
    keybow.set_pixel(6, 255, 0, 255) -- Magenta
    keybow.set_pixel(7, 255, 0, 255) -- Magenta
    keybow.set_pixel(8, 0, 255, 255) -- Cyan
    keybow.set_pixel(9, 255, 0, 255) -- Magenta
    keybow.set_pixel(10, 0, 255, 255) -- Cyan
    keybow.set_pixel(11, 0, 255, 255) -- Cyan
end
```

The `setup` function is run when the `keys.lua` file is first loaded.
The `keybow.auto_lights(false)` and `keybow.clear_lights()` lines disable the PNG animation and clear any LEDs that are lit.

The other lines set each of the twelve pixels using `keybow.set_pixel(pixel, r, g, b)` where `pixel` is the pixel number and `r`, `g`, and `b` are RGB colour values from 0 to 255. The LED are numbered in the same order as the keys, from bottom left to top right, and start at the bottom left key when Keybow is in portrait orientation.

Rather than setting these pixels in the `setup` function, you can set them in the `handle_key` function for the keys themselves, meaning that you can have them come on, or change colour, when the keys are pressed and/or released. Here's an example where key 0 is red normally, but changes to green when pressed:

```
function setup() -- Set custom lights up
    keybow.auto_lights(false)
    keybow.clear_lights()
end

function handle_key_00(pressed)
    if pressed then
        keybow.set_key("0", pressed)
        keybow.set_pixel(0, 0, 255, 0)
    else
        keybow.set_pixel(0, 255, 0, 0)
    end
end
```

Advanced customisation

We'll cover advanced customisation of Keybow with snippets and macros in a further tutorial: Using macros and snippets with Keybow.

That tutorial covers binding multiple keypresses to a single key to form macros and the use of our ready-made Windows and Mac snippets. This is where Keybow starts to get really powerful!

Using macros and snippets with Keybow

The real power of Keybow is in its ability to become a macro keyboard, triggering a whole series of keypresses from just a single key, for example "control-alt-delete" or typing "The quick brown fox jumps over the lazy dog". This is great for setting up actions that you can never quite remember, like the shortcut to import photos to Lightroom (command/control-shift-I), by just linking them to a single key.

You could also create a Keybow layout that stored all your frequently-used bits of text, like code snippets, email templates, or even complex sets of actions like opening a web browser, then entering an URL or search term and hitting enter. The world is your lobster!

As well as these custom actions, we've put together a couple of files full of handy snippets for Windows and Mac; these are common actions like peeking the desktop, snapping windows to one side or the other, a Spotlight search on Mac, and lots more. You can link these straight to keypresses in your layout files.

We'll assume that you've followed the first two Keybow tutorials already, but if you haven't then go back over them before this one:

- Assembling Keybow
- Setting up the Keybow OS

Creating a simple macro

We'll create a simple Windows-R (opens the run menu on Windows; if you're on Mac then these keys will refresh your browser page) macro as an example of how to put them together, and how to use modifier keys and the tap key function.

Unplug the USB cable from your Keybow and pop the micro-SD card out and put it in your computer. In the the `layouts` folder, create a new layout called `macros.lua` and add the following to it:

```
require "keybow"


function handle_key_00(pressed)
    if pressed then
        keybow.set_modifier(keybow.LEFT_META, keybow.KEY_DOWN)
        keybow.tap_key("r", pressed)
        keybow.set_modifier(keybow.LEFT_META, keybow.KEY_UP)
    end
end
```

Let's break down what the code does.

The macro is bound to key 0, that is the bottom left key when Keybow is in portrait orientation with the USB cable at the right hand side.

We're running this macro when the key is pressed (as opposed to when it's released), hence the `if pressed then`.

The line `keybow.set_modifier(keybow.LEFT_META, keybow.KEY_DOWN)` uses the `set_modifier` function that allows you to keep a modifier key held down while you press another key or keys, and tells Keybow to keep the key held down - `keybow.KEY_DOWN`. `keybow.LEFT_META` represents the Windows key on Windows or the command key on Mac.

Next, we use the `tap_key` function, which simulates a quick press and release of a key, to tap the "r" key - `keybow.tap_key("r", pressed)`.

Last of all, we use the `set_modifier` function again to release the `keybow.LEFT_META` (Windows) key - `keybow.set_modifier(keybow.LEFT_META, keybow.KEY_UP)`, before closing both the `if` statement and the `handle_key`function with `end`.

Save your new `macros.lua` file in the `layouts` folder, and then open the `keys.lua` file and link and enable your new layout. The file should look something like this:

```lua
require "keybow"


-- require "layouts/default" -- Numberpad


-- Custom layouts (uncomment to enable) --


-- require "layouts/boilerplate" -- Handy bits of boilerplate text like Lorem Ipsum
-- require "layouts/lightroom" -- Handy hotkeys for Adobe Lightroom Classic CC
-- require "layouts/pico8" -- Controls for Pico-8


require "layouts/macros" -- Macros layout
```

Save the `keys.lua` filem, eject and pop the micro-SD card out and into your Keybow, then plug it in with its USB cable. Once booted, give your new macro a try!

Entering strings of text

Entering strings of text can be really handy. It might be a snippet of text that you use frequently, like an address or some Lorem Ipsum placeholder text, or it could be a search term that gets entered once you've opened your web browser with a macro.

Here, we'll extend our Windows-R run menu macro that we made and type "cmd" to open the command prompt, and then tap enter to open it. We'll also use the `keybow.sleep()` function to introduce a couple of short pauses between the parts of the macro.

Disconnect your Keybow, remove the micro-SD card again, and pop it into your computer. Open the `macros.lua` file.

Within the `handle_key_00` function that we added our macro to, add a `keybow.sleep(500)` straight after the `keybow.set_modifier(keybow.LEFT_META, keybow.KEY_UP)` line. This will introduce a half-second (500 milliseconds) pause before the next line in our macro.

Next, we'll use the `keybow.text()` function to type our "cmd" text. Add the line `keybow.text("cmd")` below the `keybow.sleep(500)` line.

You can use the `keybow.text()` function with any of the text characters on your keyboard, including shifted characters (uppercase letters, !, @, £, etc.), and also spaces, and even tabs and new lines with `\t` and `\n` respectively.

Add another `keybow.sleep(500)` below that last line to introduce another small pause.

Last of all, we'll add the line `keybow.tap_enter()` to tap the enter key. There are `keybow.tap_` functions for several of the most commonly used keys like enter, space, and tab, as well as the `tap_key()` function that we used earlier, to which you can pass any key to tap it. You can see them all towards the bottom of the `keybow.lua` file.

Your whole `macros.lua` file should now look like this:

```
require "keybow"


function handle_key_00(pressed)
    if pressed then
        keybow.set_modifier(keybow.LEFT_META, keybow.KEY_DOWN)
        keybow.tap_key("r", pressed)
        keybow.set_modifier(keybow.LEFT_META, keybow.KEY_UP)
        keybow.sleep(500)
        keybow.text("cmd")
        keybow.sleep(500)
        keybow.tap_enter()
    end
end
```

Save that file, eject the micro-SD card, and pop it into your Keybow. Does the macro work, and open the command prompt?

If you're using a Mac, then you can change this macro example to hold command, tap space, and then type something like "chrome" to open Google Chrome.

Using the ready-made snippets for Windows and Mac

We've included a folder called `snippets` with two files in, `windows_snippets` and `mac_snippets`. These files have a whole bunch of ready made functions - hotkeys and macros - that you can use directly in your layouts. They cover basic things like switching between app windows, snapping windows, opening spotlight or Windows search, and a lot more.

We'll put together an example that uses a couple of the Mac snippets to do something that you probably do several times a day, but now you can do it with just a single keypress - search for cat GIFs!

Let's add this new macro to our 'macros.lua' file that we've been using, and assign it to key 1 (the middle bottom key). Think about the steps involved in doing our cat gifs search. We have to:

1. Open Spotlight
2. Get Spotlight to open Safari
3. Focus on the smart search field
4. Type in "cat gifs" and press enter

Luckily, there are snippets we can use to do the trickier steps of a Spotlight search and the Safari search, so it's just a case of adding two lines rather than a whole bunch.

Disconnect your Keybow, remove the micro-SD card, and pop it into your computer. Open the `macros.lua` file.

At the top of the file, just below the `require "keybow"` line, add `require "snippets/mac_snippets"`. This will let us use functions from the `mac_snippets.lua` file in the `snippets` folder.

Create a new `handle_key` function at the bottom of your macros file, that looks like this:

```
function handle_key_01(pressed)
    if pressed then
        mac_snippets.spotlight("safari")
        mac_snippets.safari_search("cat gifs")
    end
end
```

Remarkably easy, huh? The `spotlight` and `safari_search` snippets take a string as input and handle the other parts of the process, i.e. the keyboard shortcut, entering the text, and then pressing enter.

Your whole `macros.lua` file should now look like this:

```
require "keybow"
require "snippets/mac_snippets"

function handle_key_00(pressed)
    if pressed then
        keybow.set_modifier(keybow.LEFT_META, keybow.KEY_DOWN)
        keybow.tap_key("r", pressed)
        keybow.set_modifier(keybow.LEFT_META, keybow.KEY_UP)
        keybow.sleep(500)
        keybow.text("cmd")
        keybow.sleep(500)
        keybow.tap_enter()
    end
end

function handle_key_01(pressed)
    if pressed then
        mac_snippets.spotlight("safari")
        mac_snippets.safari_search("cat gifs")
    end
end
```

Eject the micro-SD card, put it into your Keybow, then plug it into your computer. Press key 1 and... SHAZAM... cat GIFs!

Take a look though the `windows_snippets` and `mac_snippets` files in the `snippets` folder to see all of the functionality included, everything from general navigation shortcuts, to screenshots, to specific app shortcuts for Safari, Chrome, and email.

Mechanical switches

Keybow comes with your choice of Kailh Speed Gold (clicky) or Silver (linear, non-clicky) switches. Both switches are light and smooth, and the gold switches have a satisying click when pressed.

We've chosen clear DSA key caps for Keybow, as they show off the per-key RGB LEDs really well. The slightly frosted finish on the clear key caps diffuses the light beautifully. Being DSA, the caps have a flat profile that suits small size of Keybow.

The switches slot into the PCB switch plate to hold them securely, and then push into the Kailh hot-swap sockets on the Keybow PCB. This means that there's no soldering required, and you can easily change out the switches in the future, if you wish.

Note that if you want to use different switches with Keybow, then you'll need to ensure that they have a recess on the underside for surface-mount LEDs.

Lighting

We've used the same tiny APA102 RGB LEDs that we use on our Picade Plasma PCBs, and there's one under each of the twelve keys. The LEDs sit in the cavity on the underside of the switch and shine up through, into the key cap.

There's a nifty way to light and animate the LEDs on Keybow. You can create a PNG file with a coloured gradient or pattern, and it will be animated across the LEDs from the top of the image to the bottom. The width of the PNG determines how it's displayed.

You can also manually set the LEDs on one or more keys, overriding the animation, or have them only light up when pressed.

There's a bunch of example animations to use, or you can create your own in your favourite graphics program.

Key mappings and layouts

The power of Keybow is in how customisable it is. You can map each of the twelve keys to whichever keyboard keys you want, or even have them trigger a whole series of keypresses or strings of text to be entered.

Our Keybow software uses the on-the-go micro-USB port on the Raspberry Pi Zero WH and USB HID gadget mode, so that it appears as a regular USB keyboard device when plugged into a computer.

The custom, stripped-down OS runs on a RAM-disk, meaning that it boots and runs quickly, it's robust against being unplugged, and there's no risk of SD card corruption.

To customise your Keybow layout and lighting, just pop the micro-SD card out and edit the keys.lua file on your computer.

We've included a bunch of useful code snippets and helper functions for Windows and Mac that can be used in your Keybow layouts, as well as whole example layouts to turn your Keybow into things like an Adobe Lightroom hotkey pad, a Pico-8 games controller, or just a regular numberpad.

You can read how to set up the Keybow OS and how to create your own macros and key layouts here on our learning portal.