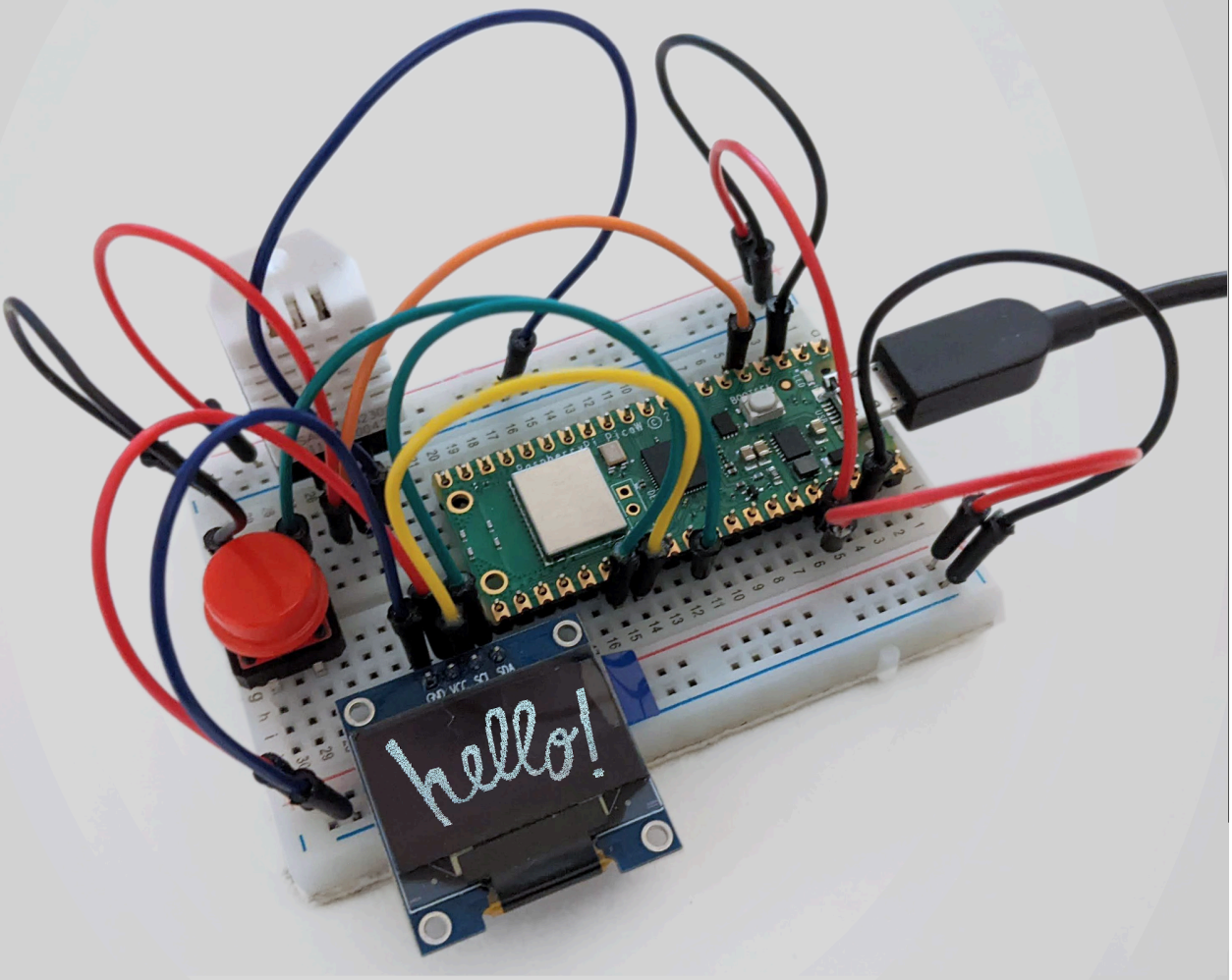
MicroPython

for the Raspberry Pi Pico W



Miguel Grinberg

MicroPython for the Raspberry Pi Pico W

Miguel Grinberg

Jan 18, 2026

Contents

Preface

In 2018, I moved to a new house that was very cold. The heating controller in this house, which was one of the newest Internet enabled models, gave me the option to see and change the heating schedules from my smartphone or computer. Yet, I found that I constantly needed to make manual adjustments because I was cold. After an investigation I determined that the thermostats installed in the house were very inaccurate, often reporting a temperature that was a few degrees higher than what I measured with my own thermometer. Because I could easily start and stop the heating by sending HTTP requests to the heating controller, I decided to build my own thermostats and use them instead of the stock ones. This was the excuse that I needed to learn MicroPython!

I built two Wi-Fi thermostats for less than $10 USD each and installed them in the two levels of the house, and that solved my heating problem without having to make any major modifications in a house I did not own. I hope after you learn how to work with MicroPython you will also come up with ideas for little devices that can improve your life in some way.

This edition of the book has been revised from the 2019 original and adapted to use a Raspberry Pi Pico W[1] microcontroller board in all examples. The examples that do not require an Internet connection also run on the original Raspberry Pi Pico[2] board.

Who This Book Is For

This book is for curious developers interested in learning how to create small Internet enabled devices, in the so-called "Internet of Things" category. The book is written in tutorial form, with each chapter introducing new concepts that build upon the previous material.

Requirements

The first chapter of the book includes a "shopping list" of hardware components that you need to build the examples featured in this book. In this edition, the main component is a Raspberry Pi Pico W microcontroller board, but there are some other components as well. In addition, you will need

[1] https://datasheets.raspberrypi.com/picow/pico-w-product-brief.pdf
[2] https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf

a computer, which will be used to program the microcontroller. You can use any Windows, macOS or Linux computer, as long as it has one available USB port to which the microcontroller can be connected.

How To Work With The Example Code

I have released the complete source code for this book on a GitHub repository[3]. In general, you will not need this repository, as the entire source code for all the examples is shown in this book. Having the complete code examples in a centralized location, however, can help you find mistakes in your own code. Also, in the last chapter of the book I am using some image files and third-party library code that you will be able to download from this repository.

Conventions Used In This Book

This book frequently includes commands that you need to type in a terminal session. For these commands, a $ will be shown as a command prompt. This is a standard prompt for many Linux shells, but may look unfamiliar to Microsoft Windows users. For example:

```
$ python hello.py
hello
```

In a lot of the terminal examples, you are going to be required to have an activated *virtual environment* (do not worry if you don't know what this is yet, you will find out very soon!). For those examples, the prompt will appear as `(venv) $`:

```
(venv) $ python hello.py
hello
```

You will also need to interact with the MicroPython REPL, the interactive Python prompt. Examples that show statements that need to be entered in a MicroPython interpreter session will use a >>> prompt, as in the following example:

```
>>> print('hello!')
hello
```

In all cases, lines that are not prefixed with a $ or >>> prompt, are the output printed by the command, and should not be typed.

[3] https://github.com/miguelgrinberg/micropython-pico-code

1. Welcome

1.1. About This Tutorial

This is a tutorial for Python beginners who want to learn to program devices to interact with the physical world and with the Cloud. You are going to learn how to program with MicroPython[1], a version of the Python language designed to run on microcontrollers[2]. The applications that you are going to learn how to write are going to read data from sensors, display information on small screens, react to the push of a button, and upload or download data from the Internet. Lots of cool stuff!

This tutorial is focused on the software side more than on the hardware side. I'm going to use a high-level "Lego" approach for building circuits, so instead of having to solder components into circuit boards, you'll be building your experiments on breadboards[3], and interconnecting components with jumper wires[4]. You will be able to assemble and disassemble your circuits with ease, and more importantly, reuse components as you move through the tutorial chapters and later when you build your own projects.

1.2. Requirements

To be able to follow this tutorial you just need to have basic coding experience with Python. You do not need to have any previous knowledge of microcontrollers, electronics, or even MicroPython.

You will also need a Windows, Mac or Linux computer with a free USB port, as you will connect the microcontroller board to your computer to program it.

[1] https://micropython.org/
[2] https://en.wikipedia.org/wiki/Microcontroller
[3] https://en.wikipedia.org/wiki/Breadboard
[4] https://en.wikipedia.org/wiki/Jump_wire

1.3. The Shopping List

In this section I will give you a short list with the components that you are going to need to build all the examples featured in this book. As a reference, I'm providing links to Amazon US and UK for most of these components (disclaimer: my Amazon affiliate link is embedded in these links), but you do not need to buy the exact products I'm linking. There are many manufacturers for these items, so use the links I provide as a reference and then if you prefer, buy from your favorite retailer. If you want a recommendation other than Amazon, Ebay is also a great place to shop online for electronic components, and you'll probably find better prices.

1.3.1 Microcontroller Board

MicroPython runs on many types of microcontrollers, but the projects featured in this book are based on the Raspberry Pi Pico W[5] board. There are two editions of this board labeled "Pico W" and "Pico WH". The electronics in these two models are identical, but the WH model comes with headers (the two rows of pins sticking from the bottom in the picture below) pre-soldered into the board. If you are a beginner in electronics, you should buy the "WH" model. If you have some experience with a soldering iron, you can buy the easier to find "W" model and loose headers, and solder them yourself. All the projects in this book are built on a breadboard, so you will need a device with headers.

The cost of these devices is so low that I recommend that you order at least two to have a spare.

Buy from Raspberry Pi Foundation[6]

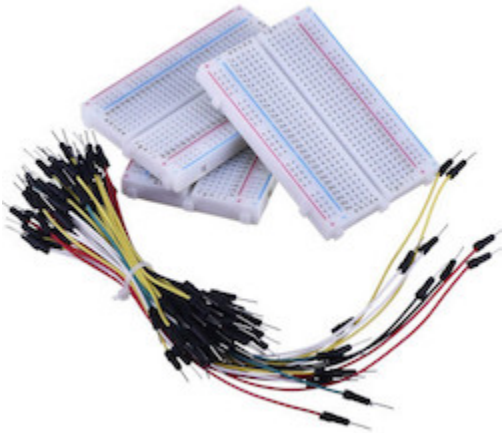
Note that there is an earlier model, the original "Pico" board and its "Pico H" variant with soldered headers. The most important difference is that these older boards do not have Wi-Fi support. If you

[5] https://datasheets.raspberrypi.com/picow/pico-w-product-brief.pdf
[6] https://www.raspberrypi.com/products/raspberry-pi-pico/

already have this board, you can use it instead of the W or WH models, but only for the projects that do not require Internet access.

1.3.2 Breadboard and Jumper Wires

To be able to easily build and take apart your circuits without soldering, you are going to need a breadboard and some jumper wires. I recommend that you get more than one breadboard as well, in case you want to build multiple projects in parallel.



Buy from Amazon US[7] - Buy from Amazon UK[8]

1.3.3 Temperature and Humidity Sensor

To demonstrate how to work with sensors, you are going to use a DHT22 temperature and humidity sensor. This little device reads and reports the temperature and humidity levels in the environment. Depending on where you buy it, you may find models that come with three or four pins. Both are fine for this tutorial.

[7] https://amzn.to/2CCcjww
[8] https://amzn.to/2OiTTFG

Buy from Amazon US[9] - Buy from Amazon UK[10]

1.3.4 Screen

Some projects in this book display information on a small 128x64 pixel OLED screen. Once again, these are so cheap that you may want to get a few.

[9] https://amzn.to/2Tq2CqO
[10] https://amzn.to/2OkjbU4

Buy from Amazon US[11] - Buy from Amazon UK[12]

1.3.5 Push Button

A common use case for circuits is to perform actions after a button push. You are not going to need more than one button at a time, but as with the above you may want to also get a pack, since buttons are always handy to have.

Buy from Amazon US[13] - Buy from Amazon UK[14]

1.3.6 USB to Micro-USB Cable

You will need a USB cable to connect your microcontroller to your computer. This is going to provide power, and also a serial connection through which you'll upload code into the microcontroller memory. The end that plugs into the microcontroller board should have a male Micro-USB connector. The end that plugs into your computer needs to be USB-A or USB-C, depending on the available ports on your computer.

The Micro-USB connector used to be popular to charge Android phones, so you may already have an old charging cable that you can use.

[11] https://amzn.to/2Ygzztq
[12] https://amzn.to/2TWUlQ3
[13] https://amzn.to/2OiPKS8
[14] https://amzn.to/2U6zWrc

Buy from Amazon US[15] - Buy from Amazon UK[16]

1.3.7 Power Bank (Optional)

A power bank is not required for this tutorial, but if you have one, you will be able to use it to power your microcontroller independently of the computer after you have uploaded your code.

1.4. Let's Build Some Stuff!

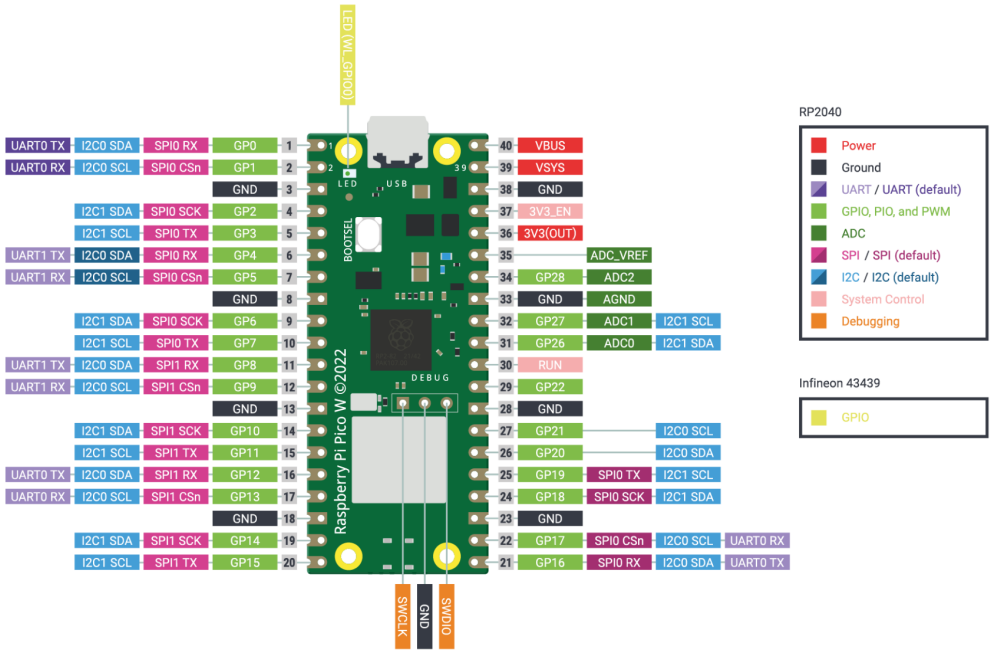
Once you have the materials listed above you are ready to move on to the next chapter and start building stuff!

[15] https://amzn.to/2YgEpqt
[16] https://amzn.to/2JwgCQ1

2. Hello, MicroPython!

2.1. The Raspberry Pi Pico W Microcontroller Board

To begin, I want to give you a very brief overview of the components that are included in your Pico microcontroller. Below you can see a diagram that should more or less match your board:



The official Raspberry Pi Pico W documentation includes a high-resolution PDF[1] of this diagram that I recommend that you download to be able to read the labels.

[1] https://datasheets.raspberrypi.com/picow/PicoW-A4-Pinout.pdf

The RP2040 microcontroller chip is the black box in the middle of the board. The larger silver box below it is the Infineon CYW43439 chip, which provides the Wi-Fi connectivity. There are 20 pins on each side of the board, each with a number and one or more labels such as GP0, 3V3(OUT), GND, etc. The labels associated with each pin are an indication of the pin's function. Since these boards are extremely optimized, most pins can perform multiple functions and thus have multiple labels.

The pin labeled 3V3(OUT) provides electric current. The name derives from the voltage, which is 3.3 volts. This pin can be used to power other components that are part of your circuit. There are also eight GND or "ground" pins, spread on both sides of the board. These provide a ground connection, which is required to close the circuit when you provide power to a connected component. Electric current always flows towards the ground, so when you connect a device to the microcontroller you will be making two actual connections. A connection from the 3V3(OUT) pin will deliver power to the component, and a connection from the component to a GND pin will close the circuit, effectively forcing electrical current to flow through the component. As you will learn later in this tutorial, all components have a clearly labeled pin for input current, typically called VCC, and one for connection to ground also called GND.

The pins labeled with a GP<n> are called GPIO (General-Purpose Input/Output) pins. These are the pins that are used for communication between the microcontroller and other connected devices. The Raspberry Pi Pico W and WH boards have 26 GPIO pins labeled from GP0 to GP28, skipping GP23, GP24 and GP25. The three missing pins, along with GP29, exist but are used internally by the board and do not have a physical pin assigned to them.

Many of the GPIO pins can be programmed to do other tasks instead of GPIO. For example, the three pins labeled ADC0 through ADC2 (which are also pins GP26 to GP28) are "analog" pins. GPIO pins have binary values, either 0 or 1, while analog pins can have a range of values, determined by the amount of electrical current passing through the pin. A fourth ADC pin is also available, and connected internally to a temperature sensor. Note that this book does not make use of the analog pins.

The board has a small built-in LED (Light Emitting Diode[2]), a tiny light bulb that you can see in the top-left part of the diagram, between pins number 2 and 3. This LED is connected to a special internal pin labeled WL_GPIO0. Note that if you are using the original Pico board, the internal LED pin is wired to the internal GP25 pin referenced above. By the end of this chapter you will know how to turn the LED on and off with MicroPython code!

Another interesting pin is RUN, which can be used to send a reset signal to the board. In future chapters you are going to learn how to use this pin to "wake" the microcontroller when it is in a deep sleep state.

The pin labeled VSYS can be used to provide power to the board, as an alternative to the much more convenient micro-USB port. All the examples you are going to build in this tutorial are powered through the micro-USB port, so you are not going to use this pin.

In the top center you have the micro-USB input connector. In this tutorial you will always power

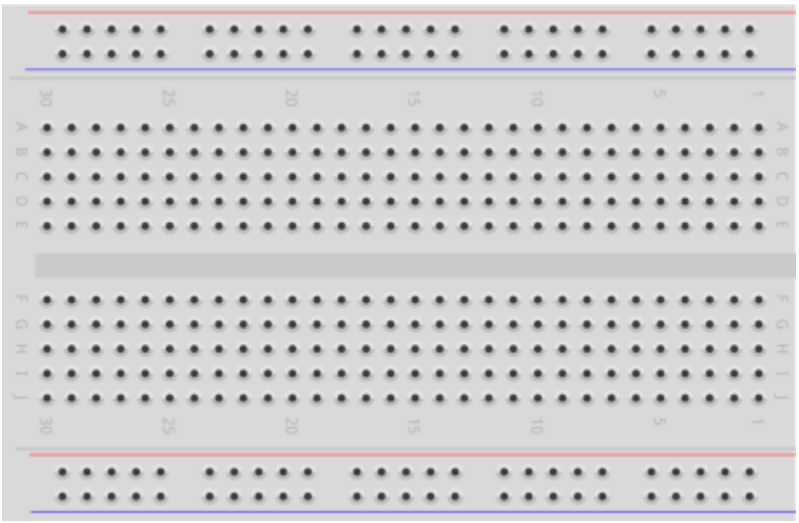
[2] https://en.wikipedia.org/wiki/Light-emitting_diode

the board through this connector. When you connect the other end of the cable to your computer, the board is going to power up, and at the same time, it is going to appear as a serial device in your computer. The serial connection that is established between the computer and the microcontroller board is used to program the board.

To complete this overview, note that the board has a small button labeled `BOOTSEL`, which you can find in the top-left portion of the diagram, below the LED. This button prepares the board to be flashed with a new firmware. You will learn how to use this button to flash the MicroPython language into your board shortly.

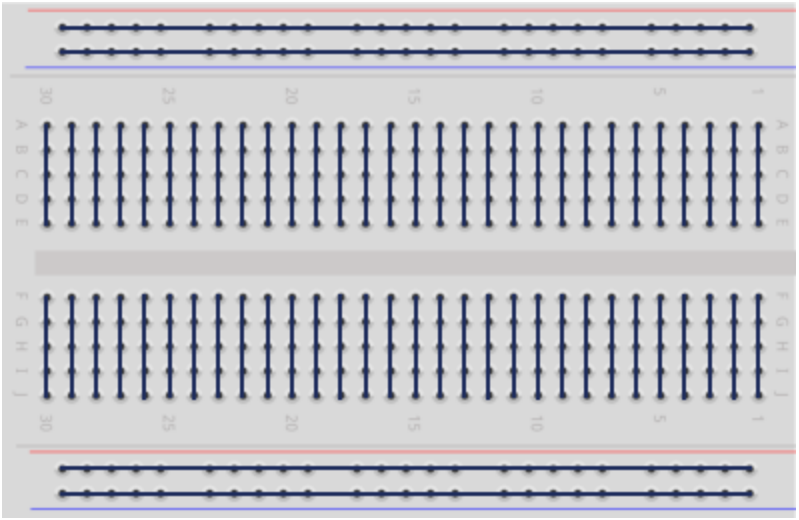
2.2. The Breadboard

The first task that I'm going to show you how to do is to set up the microcontroller board on a breadboard. So first, let me tell you how breadboards work. Here is a diagram that should match the breadboard that you have:



The key to understand how to work with a breadboard is to visualize the internal connections between the holes. Once you know which holes are internally connected, you know that you can create a connection between two pins by inserting them in a pair of connected holes. It's really that simple.

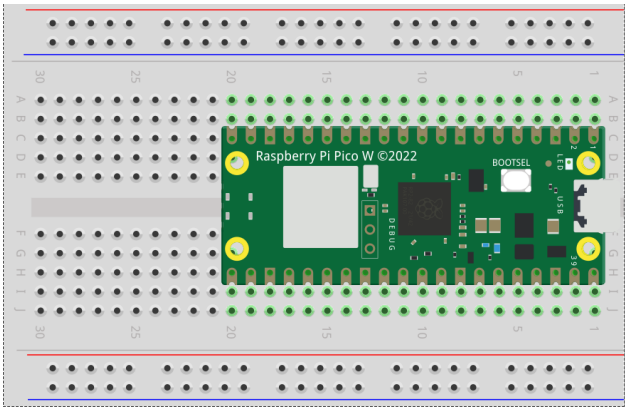
Below I created a diagram that shows these internal connections between holes:



The top two and top bottom rows of holes are usually referred to as the "power strips", and they actually work in a similar way to a household power strip. For example, when you connect a pin that delivers current (such as any of the `3V3(OUT)` pin in the Pico board) into any of the holes in the top row, then all the remaining holes in that row can be used by other devices to draw power for themselves, so effectively this is equivalent to having a direct connection between each device and the power source. This top line is marked with a red line and in some breadboards with a "+" label, which indicates that as a convention, it should be used for current, as in my example. The second row of holes is marked in blue and with a "-" label. The convention is to use the pins in this line for ground connections. There are two more positive and negative power strips at the bottom of the breadboard.

The connections used by the holes in the middle are more tricky to understand. Here the holes are grouped in fives. The ten rows of holes in the two middle sections are labeled with the letters `a` through `j`, but unlike the power strip rows, these rows are not connected. The connections for these pins are vertical, and cover just the group of five vertical holes in each section. The columns are labeled from `1` to `30`, but in many breadboards you can only see a label every five columns. Using these labels, you can see that the `a1`, `b1`, `c1`, `d1`, and `e1` holes are all connected with each other, so any pins that are plugged into these holes will be all connected. Similarly, pins `f1`, `g1`, `h1`, `i1` and `j1` are connected between them, but they do not share a connection with the top five holes.

If you think this is starting to make sense, you are ready to install the microcontroller on the breadboard. Place the breadboard on your desk with a red power strip on top and column 1 to the right, and then align your microcontroller in the middle section, with the micro-USB port facing right. Do not insert it into the breadboard yet. The top 20 pins from the microcontroller board should be in the `c` row of your breadboard, columns 1 on the right to 20 on the left. The bottom strip of microcontroller pins should be aligned with the `h` row. The `a`, `b`, `i` and `j` rows should be free and visible from the top. Here is a diagram:



After you align the pins as shown in the diagram above, gently press on the board until the pins are fully inserted into the holes.

2.3. Wiring the Breadboard's Power Strips

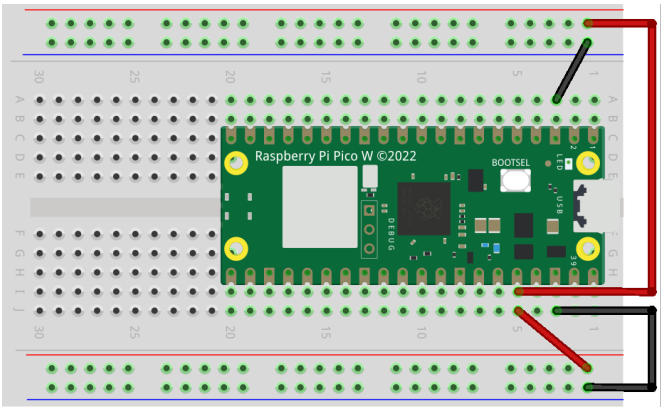
The next task is to set up the delivery of power to the breadboard. I explained above that the top and bottom two rows of holes are used to deliver current and ground to other devices. While in this chapter there are no additional devices that need to make use of power, it is always a good idea to have the power strips properly set up on the breadboard for when they are needed.

To do this you are going to need four jumper wires. If you have different kinds of wires, make sure the ones you select have all male pins. If you have wires of several colors available, you may want to use two red wires for the positive connections, and two black or blue wires for the ground.

The connections that you need to make are four:

- From one of the two holes below the `3V3(OUT)` pin to any hole in the breadboard's red row on the top
- From the other open hole below the `3V3(OUT)` pin to any hole in the breadboard's red row on the bottom
- From a hole next to one of the top `GND` pins (for example, the `a3` hole) to any hole in the breadboard's blue row on the top
- From one of the bottom `GND` pins (for example, the `j2` hole) to any hole in the breadboard's bottom row

The end result should look more or less like the following diagram:



In case this isn't clear, no other component is drawing power from the microcontroller at this time, so these connections that supply power and ground to the breadboard aren't really doing anything yet, but it's nice to have the breadboard in a state that is ready to support additional components later on.

This is enough work in the physical world for this chapter. Now I'm going to take you to the software side and show you how to prepare your computer so that you can install and run MicroPython on your board.

2.4. Setting Up Your Computer

The tool that you are going to use to communicate with the Pico board is written in Python, so what you need to do now (if you haven't yet) is install Python on your computer. If your operating system does not provide a pre-packaged Python 3 installer, you can go to https://python.org to download an official build for any of the supported operating systems. Make sure it is a recent 3.x version.

Once Python is installed, open a terminal window. On Linux or macOS I will assume that you are running a `bash`, `zsh` or similar shell, while on Windows it will be the Command Prompt window. First create a new directory where you are going to store files associated with this tutorial:

```
$ mkdir micropython-pico-tutorial
$ cd micropython-pico-tutorial
```

Remember that the `$` is not part of the command and is shown just so that you know the text that follows has to be typed on the terminal.

The next step is to create a Python virtual environment. This is the recommended way to install packages, so that they are private to your project instead of installed system-wide. You can create a virtual environment with the following command:

```
$ python -m venv venv
```

This command is asking Python to run the `venv` package (the first `venv`) and create a virtual en-

vironment named `venv` (the second `venv`). If you want to use a different name for your virtual environment, replace the second `venv` with name you want to use. After the command completes, you should see a subdirectory with this name. Inside this subdirectory there is a private copy of the Python interpreter.

Note that if the command above does not work, it is possible that in your system the Python interpreter is called `python3` and not `python`. In that case, the command should be:

```
$ python3 -m venv venv
```

Now you are going to activate this new virtual environment. The virtual environment activation configures the Python interpreter installed inside the virtual environment as the currently active Python that is invoked when you type `python` in the command line. This activation is temporary, by the way, nothing in your system is modified. The command to do the activation is different depending on the operating system. If you are using Linux or macOS, the command is:

```
$ source venv/bin/activate
(venv) $ _
```

If you are using Windows, the activation command is:

```
$ venv\Scripts\activate
(venv) $ _
```

See how as a response to running this command, the prompt is modified to indicate that `venv` is activated.

With the virtual environment activated, you are ready to install the package that is going to help you manage your Pico board. For this you are going to use the Python installer utility, called `pip`. This utility is already installed in the virtual environment:

```
(venv) $ pip install rshell
```

At this point your computer should be ready, but I want to give you some additional information on virtual environments in case you are not familiar with them before I move on to the next task.

If you want to deactivate an activated virtual environment and return to the globally available Python interpreter, use the `deactivate` command:

```
(venv) $ deactivate
$ _
```

Something that confuses a lot of beginners is that virtual environment activations are local to each terminal session, so when you work with multiple terminals at the same time, you have to run the activation command in all of them. And if you close a terminal window while the virtual environment was activated, then the activation is lost, and you will need to re-issue it when you open a new terminal. You will also need to repeat the activation of the virtual environment after you reboot your computer.

Just so that this is completely clear, let's assume you've done all the steps above to install `rshell`, then turned your computer off and on again, and now want to resume working on this project. The virtual environment is still stored in your computer, and it still contains the installed packages, but it is not activated, so when you run `python` you will be using the system-wide interpreter, and if you try to run `rshell` you are going to get errors since these are not installed globally. To go back to your virtual environment, you would need to open a terminal window and enter the following commands. For Linux and macOS:

```
$ cd micropython-pico-tutorial
$ source venv/bin/activate
(venv) $ _
```

And if you are on Windows:

```
> cd micropython-pico-tutorial
> venv\Scripts\activate
(venv) > _
```

And now you should be back in business!

2.5. Erasing the Pico Flash Memory

At this point you have your Pico W board nicely set up in the breadboard. Before you learn how to install MicroPython on it, it is a good idea to erase the flash memory of your board. This is especially important if you have used the board before, as this will restore it to brand-new condition and ensure that there are no side effects caused by previously stored files or data.

Firmware files for the Pico board use a *.uf2* extension. The Raspberry Pi Foundation provides a special firmware that when flashed to the board causes the flash memory to be completely erased. Find the link to download the *flash_nuke.uf2* firmware on the Resetting Flash memory[3] section of the Pico documentation. You may want to move this file to the *micropython-pico-tutorial* directory that you created above to keep it close to the project you are going to build.

Now take the USB cable and connect the USB-A or USB-C end to your computer. Next, press the `BOOTSEL` button on your Pico board and while keeping it pressed, plug the Micro-USB end of your cable to the board to power it up. A few seconds later you are going to see a new disk drive labeled `RPI-RP2` mount on your computer.

[3] https://www.raspberrypi.com/documentation/microcontrollers/raspberry-pi-pico.html#resetting-flash-memory

Using your mouse, drag and drop the *flash_nuke.uf2* file that you just downloaded on the Pico disk. The disk will disappear shortly after receiving the firmware file, and with this the flash memory of the board will be erased. On some operating systems you may get a warning that you disconnected the disk drive without ejecting it first. You can ignore this warning in relation to the `RPI-RP2` disk.

2.6. Flashing MicroPython to the Pico Board

The process of installing MicroPython on your board is similar to that of erasing the flash memory, but with a different *.uf2* firmware file.

You can download an official release of MicroPython from micropython.org[4]. The Downloads page has a section for Raspberry Pi Pico W[5] builds, so find that section to access the files. There are several boards that have the word "Pico" in their names, so make sure you find the downloads for the "Pico W" board from the Raspberry Pi Foundation, as firmware files for other boards will not work.

At the time I'm writing this, there are only nightly builds for the Pico W, but by the time you do this you may find one or more official releases as well. Download the latest release or nightly build using your web browser. This should be a file with a name that starts with *rp2-pico-w* and has the standard *.uf2* extension. To keep things organized, also move the file from your downloads folder into the *micropython-pico-tutorial* project directory.

To install MicroPython on the Pico board you have to repeat the flashing process you used before to erase the flash memory. If the Pico board is still connected to your computer, disconnect one of the cable ends. Then press the `BOOTSEL` button and reconnect the cable while keeping the button pressed. Once the `RPI-RP2` disk reappears, drop the MicroPython firmware file on it. As before, the disk will disappear shortly after the file is copied.

[4] https://micropython.org/download
[5] https://micropython.org/download/rp2-pico-w/