5th Edition
Fully updated for Raspberry Pi 5

The Official Raspberry Pi Beginner's Guide

How to use your new computer



Raspberry Pi PRESS

Gareth Halfacree

The Official Raspberry Pi Beginner's Guide, 5th Edition

The Official Raspberry Pi Beginner's Guide
by Gareth Halfacree
ISBN: 978-1-912047-26-0
Copyright © 2024 Gareth Halfacree
Printed in the United Kingdom
Published by Raspberry Pi, Ltd., 194 Science Park, Cambridge, CB4 0AB

Editors: Brian Jepson, Liz Upton
Interior Designer: Sara Parodi
Production: Nellie McKesson
Photographer: Brian O'Halloran
Illustrator: Sam Alder
Graphics Editor: Natalie Turner
Publishing Director: Brian Jepson
Head of Design: Jack Willis
CEO: Eben Upton

July 2024: Fifth Edition, Second Printing
October 2023: Fifth Edition
November 2020: Fourth Edition
November 2019: Third Edition
June 2019: Second Edition
December 2018: First Edition

The publisher, and contributors accept no responsibility in respect of any omissions or errors relating to goods, products or services referred to or advertised in this book. Except where otherwise noted, the content of this book is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported (CC BY-NC-SA 3.0).

Table of Contents

v **Welcome to the Official Raspberry Pi Beginner's Guide**
vii **About the author**

Chapter 1
1 **Get to know your Raspberry Pi**
Introducing your new credit-card-sized computer. Take a guided tour of the Raspberry Pi, find out how it works, and discover some of the amazing things you can do with it.

Chapter 2
15 **Getting started with your Raspberry Pi**
Discover the essential items you'll need for your Raspberry Pi, and learn how to connect them all to get it set up and working.

Chapter 3
31 **Using your Raspberry Pi**
Learn about the Raspberry Pi operating system.

Chapter 4
53 **Programming with Scratch 3**
Learn how to start coding using Scratch, a block-based programming language.

Chapter 5
89 **Programming with Python**
Now you've got to grips with Scratch, we'll show you how to do text-based coding with Python.

Chapter 6
121 **Physical computing with Scratch and Python**
There's more to coding than doing things on screen — you can also control electronic components connected to your Raspberry Pi's GPIO pins.

Chapter 7
155 **Physical computing with the Sense HAT**
As used on the International Space Station, the Sense HAT is a multifunctional add-on board for Raspberry Pi, equipped with sensors and an LED matrix display.

Chapter 8
201 **Raspberry Pi Camera Modules**
Connecting a Camera Module or HQ Camera to your Raspberry Pi enables you to take high-resolution photos and shoot videos, and create amazing computer vision projects.

Chapter 9
219 **Raspberry Pi Pico and Pico W**
Raspberry Pi Pico and Pico W bring a whole new dimension to your physical computing projects.

Appendices

239 **Appendix A**
Install an operating system to a microSD card

245 **Appendix B**
Installing and uninstalling software

251 **Appendix C**
The command-line interface

259 **Appendix D**
Further reading

269 **Appendix E**
Raspberry Pi Configuration Tool

277 **Appendix F**
Raspberry Pi specifications

Welcome to the Official Raspberry Pi Beginner's Guide

We think you're going to love your Raspberry Pi. Whichever model you have — a standard Raspberry Pi board, the compact Raspberry Pi Zero 2 W, or the Raspberry Pi 400 with integrated keyboard — this affordable computer can be used to learn coding, build robots, and create all kinds of weird and wonderful projects.

Raspberry Pi is capable of doing all the things you'd expect from a computer — everything from browsing the internet and playing games, to watching movies and listening to music. But your Raspberry Pi is much more than a modern computer.

With a Raspberry Pi you can get right into the heart of a computer. You get to set up your own operating system, and can connect wires and circuits directly to its GPIO pins. It was designed to teach young people how to program in languages like Scratch and Python, and all the major programming languages are included with the official operating system. With Raspberry Pi Pico, you can create unobtrusive, low-power projects that interact with the physical world.

The world needs programmers more than ever, and Raspberry Pi has ignited a love of computer science and technology in a new generation.

People of all ages use Raspberry Pi to create exciting projects: everything from retro games consoles to internet-connected weather stations.

So if you want to make games, build robots, or hack a variety of amazing projects, then this book is here to help you get started.

You can find example code and other information about this book, including errata, in its GitHub repository at **rptl.io/bg-resources**. If you've found what you believe is a mistake or error in the book, please let us know by using our errata submission form at **rptl.io/bg-errata**.

About the author

Gareth Halfacree is a freelance technology journalist, writer, and former system administrator in the education sector. With a passion for open-source software and hardware, he was an early adopter of the Raspberry Pi platform and has written several publications on its capabilities and flexibility. He can be found on Mastodon as **@ghalfacree@mastodon.social** or via his website at **freelance.halfacree.co.uk**.

Colophon

Raspberry Pi is an affordable way to do something useful, or to do something fun.

Democratising technology — providing access to tools — has been our motivation since the Raspberry Pi project began. By driving down the cost of general-purpose computing to below $5, we've opened up the ability for anybody to use computers in projects that used to require prohibitive amounts of capital. Today, with barriers to entry being removed, we see Raspberry Pi computers being used everywhere from interactive museum exhibits and schools to national postal sorting offices and government call centres. Kitchen table businesses all over the world have been able to scale and find success in a way that just wasn't possible in a world where integrating technology meant spending large sums on laptops and PCs.

Raspberry Pi removes the high entry cost to computing for people across all demographics: while children can benefit from a computing education that previously wasn't open to them, many adults have also historically been priced out of using computers for enterprise, entertainment, and creativity. Raspberry Pi eliminates those barriers.

Raspberry Pi Press

store.rpipress.cc

Raspberry Pi Press is your essential bookshelf for computing, gaming, and hands-on making. We are the publishing imprint of Raspberry Pi Ltd, part of the Raspberry Pi Foundation. From building a PC to building a cabinet, discover your passion, learn new skills, and make awesome stuff with our extensive range of books and magazines.

The MagPi

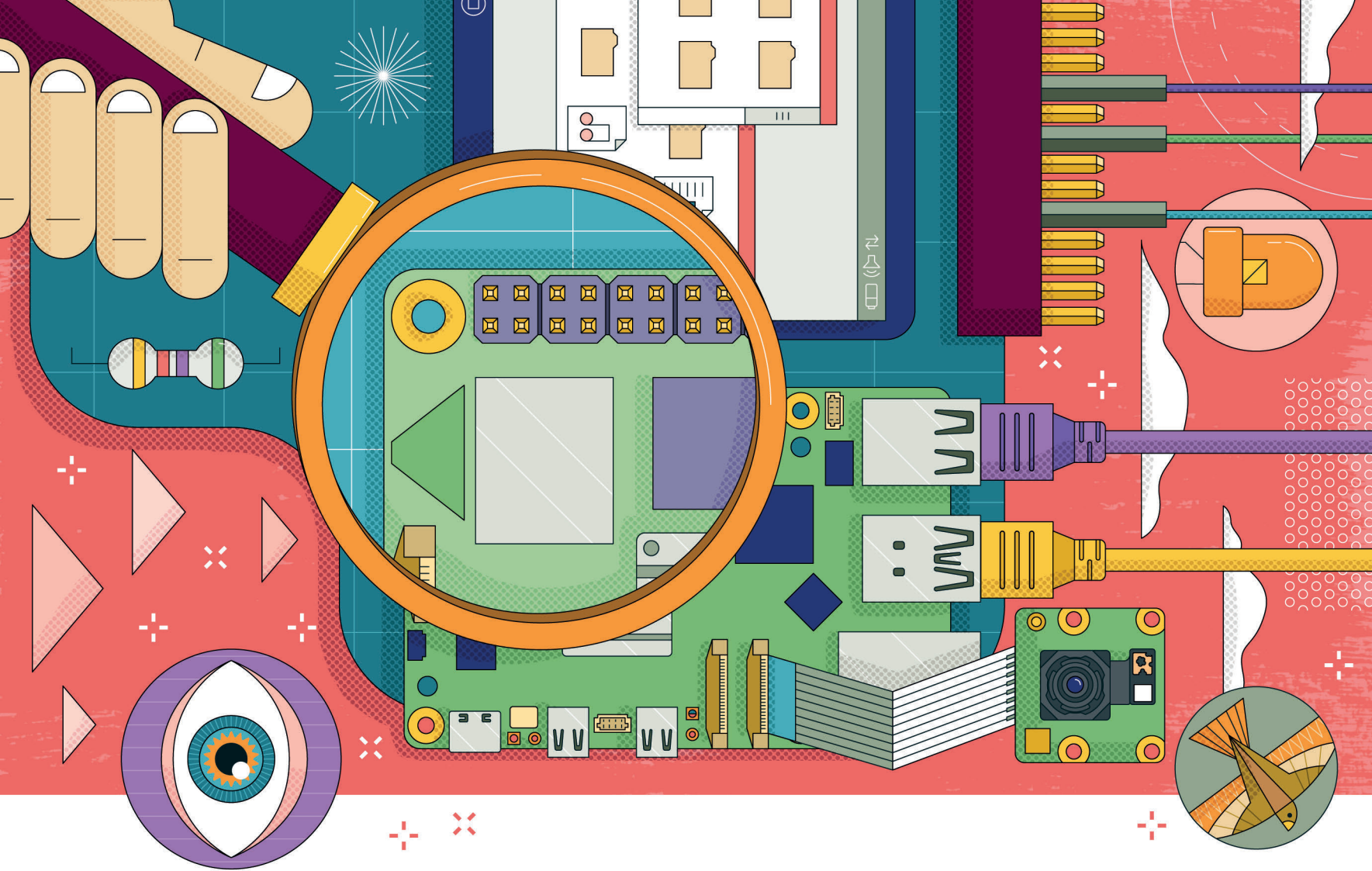
magpi.raspberrypi.com

The MagPi is the official Raspberry Pi magazine. Written for the Raspberry Pi community, it is packed with Pi-themed projects, computing and electronics tutorials, how-to guides, and the latest community news and events.

HackSpace

hackspace.raspberrypi.com

HackSpace magazine is filled with projects for fixers and tinkerers of all abilities. We'll teach you new techniques and give you refreshers on familiar ones, from 3D printing, laser cutting, and woodworking to electronics and the Internet of Things. *HackSpace* will inspire you to dream bigger and build better.



Chapter 1

Get to know your Raspberry Pi

Introducing your new credit-card-sized computer. Take a guided tour of the Raspberry Pi, find out how it works, and discover some of the amazing things you can do with it.

Raspberry Pi is a remarkable device: a fully functional computer in a tiny, low-cost package. Whether you're looking for a device you can use to browse the web or play games, are interested in learning how to write your own programs, or are looking to create your own circuits and physical devices, Raspberry Pi — and its amazing community — will support you every step of the way.

Raspberry Pi is known as a *single-board computer*, which means exactly what it sounds like: it's a computer, just like a desktop, laptop, or smartphone, but built on a single *printed circuit board*. Like most single-board computers, Raspberry Pi is small — it has roughly the same footprint as a credit card — but that doesn't mean it's not powerful: a Raspberry Pi can do anything a bigger and more power-hungry computer can do, from browsing the web and playing games to driving other devices.

The Raspberry Pi family was born from a desire to encourage more hands-on computer education around the world. Its creators, who joined together to form the non-profit Raspberry Pi Foundation, had little idea that it would prove so popular: the few thousand built to test the waters in 2012 sold out immediately, and more than fifty million have been shipped all over the world in the years since. These boards have found their way into homes, classrooms, offices, data centres, factories, and even self-piloting boats and satellites.

Various models of Raspberry Pi have been released since the original Model B, each bringing either improved specifications or features specific to a particular use-case. The Raspberry Pi Zero family, for example, is a tiny version of the full-size Raspberry Pi which drops a few features — in particular the multiple

USB ports and wired network port — in favour of a significantly smaller layout and reduced power requirements.

All Raspberry Pi models have one thing in common, though: they're *compatible*, meaning that most software written for one model will run on any other model. It's even possible to take the very latest version of Raspberry Pi's operating system and run it on an original pre-launch Model B prototype. It will run more slowly, it's true, but it will still run.

Throughout this book you'll learn about Raspberry Pi 4 Model B, Raspberry Pi 5, Raspberry Pi 400, and Raspberry Pi Zero 2 W: the latest and most powerful versions of Raspberry Pi. Everything you learn can be easily applied to other models in the Raspberry Pi family, so don't worry if you're using a different model or revision.



RASPBERRY PI 400

If you have a Raspberry Pi 400, the circuit board is built into the keyboard case. Read on to learn about all the components that make Raspberry Pi tick, or skip to "Raspberry Pi 400" on page 9 for a tour of your desktop device.



RASPBERRY PI ZERO 2 W

If you have a Raspberry Pi Zero 2 W, some of the ports and components look different when compared to the Raspberry Pi 5. Read on to learn about what each component does, or skip to "Raspberry Pi Zero 2 W" on page 11 to learn more about your device.

A guided tour of Raspberry Pi

Unlike a traditional computer, which hides its inner workings in a case, a standard Raspberry Pi has all its components, ports, and features out on display — although you can buy a case to provide extra protection, if you'd prefer. This makes it a great tool for learning about what the various parts of a computer do, and also makes it easy to learn what goes where when it comes time to plug in the various other pieces of hardware — known as *peripherals* — you'll need to get started.

Figure 1-1 shows a Raspberry Pi 5 from above. When you're using a Raspberry Pi with this book, try to keep it oriented the same way as in the pictures; otherwise it can get confusing when it comes to using things like the GPIO header (detailed in Chapter 6, *Physical computing with Scratch and Python*).

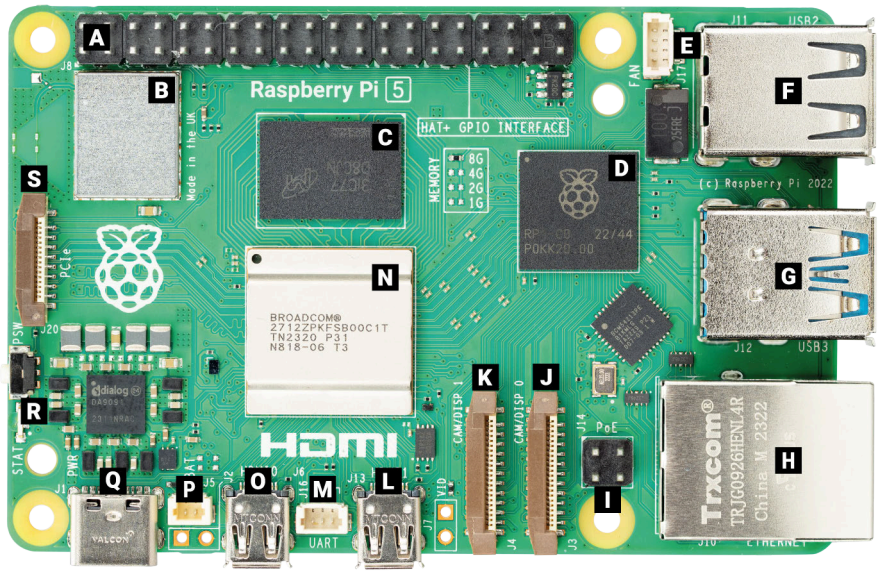


Figure 1-1 Raspberry Pi 5

A **GPIO header**

B **Wireless**

C **RAM**

D **RP1 I/O controller chip**

E **Connector for fan**

F **USB 2.0**

G **USB 3.0**

H **Ethernet port**

I **Power-over-Ethernet (PoE) pins**

J **CSI/DSI camera/display port 0**

K **CSI/DSI camera/display port 1**

L **Micro HDMI 1**

M **Connector for UART serial port**

N **System-on-chip**

O **Micro HDMI 0**

P **Real-time clock battery header**

Q **USB Type-C power in**

R **Power button**

S **Connector for PCI Express (PCIe)**

While it may look like there's an overwhelming amount packed into such a tiny board, a Raspberry Pi is very simple to understand — starting with its *components*, the inner workings that make the device tick.

Raspberry Pi's components

Like any computer, Raspberry Pi is made up of many components, each of which has a role to play in making it work. The first, and arguably most important, of these can be found just to the left of the centre point of the board (**Figure 1-2**), covered in a metal cap: this is the *system-on-chip* (SoC).

The name 'system-on-chip' is a great indicator of what you would find if you prised the metal cover off: a silicon chip, known as an *integrated circuit*, that contains the bulk of Raspberry Pi's system. This integrated circuit includes a *central processing unit* (CPU), commonly thought of as the 'brain' of a computer, and a *graphics processing unit* (GPU), which handles the visual rendering and display output side of things.

A brain is no good without memory, however, and just above the SoC you'll find exactly that: a small, black, plastic-covered rectangular chip (**Figure 1-3**). This is Raspberry Pi's *random-access memory (RAM)*. When you're working on Raspberry Pi, it's the RAM that holds what you're doing. Saving your work moves this data to the more permanent storage of the microSD card. Together, these components form Raspberry Pi's *volatile* and *non-volatile memory*: the volatile RAM loses its contents whenever Raspberry Pi loses power, while the non-volatile memory in the microSD card keeps its contents.



Figure 1-2
Raspberry Pi's system-on-chip (SoC)

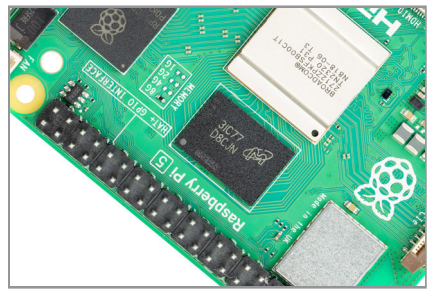


Figure 1-3
Raspberry Pi's random access memory (RAM)

At the top left of the board you'll find another metal lid (**Figure 1-4**) covering the *radio*, the component that Raspberry Pi the ability to communicate with devices wirelessly. In fact, the radio itself fills the role of two other common components: a *WiFi radio* that connects to computer networks; and a *Bluetooth radio* that connects to peripherals like mice and sends or receives data from nearby smart devices like sensors or smartphones.

Another black, plastic-covered chip marked with the Raspberry Pi logo can be found on the right side of the board, near the USB ports (**Figure 1-5**). This

is *RP1*, a custom I/O controller chip which communicates with the four USB ports, the Ethernet port, and most low-speed interfaces to other hardware.

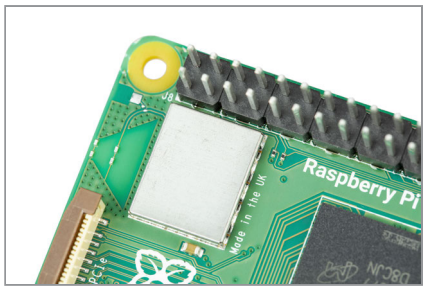


Figure 1-4
Raspberry Pi's radio module

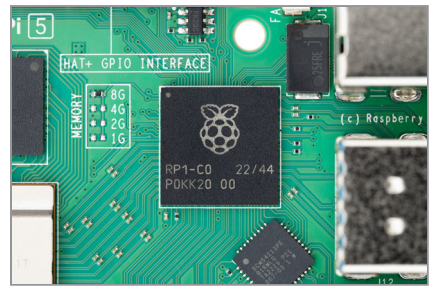


Figure 1-5
Raspberry Pi's RP1 controller chip

Another black chip, smaller than the rest, can be found a little bit above the USB C power connector at the bottom left of the board (**Figure 1-6**). This is known as a *power management integrated circuit (PMIC)*; it takes the power that comes in from the USB C port and turns it into the power your Raspberry Pi needs to run.

The final black chip, below RP1 and positioned at a jaunty angle, helps the RP1 in handling Raspberry Pi's Ethernet port. It provides what is known as an *Ethernet PHY*, providing the *physical* interface which sits between the Ethernet port itself and the Ethernet controller in the RP1 chip.

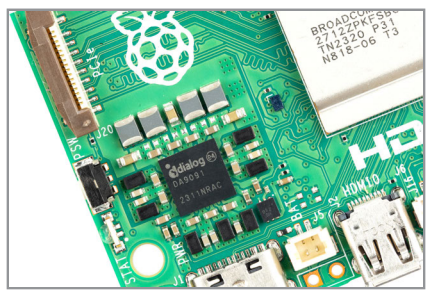


Figure 1-6
Raspberry Pi's power management integrated circuit (PMIC)

Don't worry if this seems like a lot to take in: you don't need to know what each component is or where to find it on the board in order to use Raspberry Pi.

Raspberry Pi's ports

Raspberry Pi has a range of ports, starting with four *Universal Serial Bus (USB) ports* (**Figure 1-7**) at the middle and top of its right-hand edge. These ports let you connect any USB-compatible peripheral — like keyboards, mice, digital cameras, and flash drives — to your Raspberry Pi. Speaking technically, there are two types of USB ports on Raspberry Pi, each relating to a different Universal Serial Bus standard: the ones with black plastic inside are USB 2.0 ports and the ones with blue plastic are newer and faster USB 3.0 ports.

Next to the USB ports is an *Ethernet port*, also known as a *network port* (**Figure 1-8**). You can use this port to connect your Raspberry Pi to a wired computer network with a cable that uses what is known as an RJ45 connector. If you look closely at the Ethernet port, you'll see two light-emitting diodes (LEDs) at the bottom. These are status lights which, when lit or blinking, let you know the connection is working.

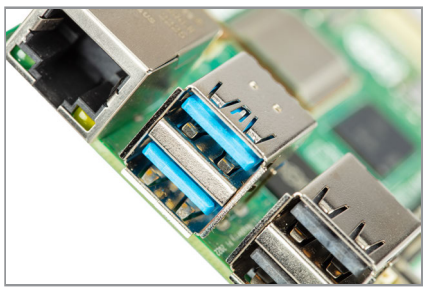


Figure 1-7
Raspberry Pi's USB ports

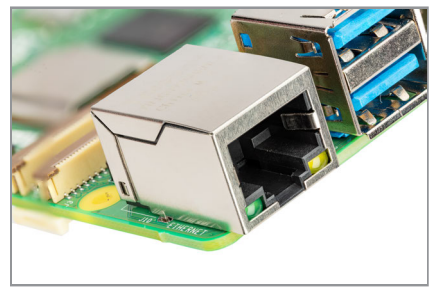


Figure 1-8
Raspberry Pi's Ethernet port

Just to the left of the Ethernet port, on the bottom edge of Raspberry Pi, is a *Power-over-Ethernet (PoE) connector* (**Figure 1-9**). This connector, when paired with the Raspberry Pi 5 PoE+ *HAT — Hardware Attached on Top*, a special add-on board designed for Raspberry Pi — and a suitable PoE-capable network switch, lets you power Raspberry Pi from its Ethernet port without having to connect anything to the USB Type-C connector. The same connector is also available on Raspberry Pi 4, though in a different location; Raspberry Pi 4 and Raspberry Pi 5 use different HATs for PoE support.

Directly to the left of the PoE connector are a pair of strange-looking connectors with plastic flaps you can pull up; these are the *camera and display connectors*, also known as the *Camera Serial Interface (CSI) and Display Serial Interface(DSI) ports* (**Figure 1-10**).

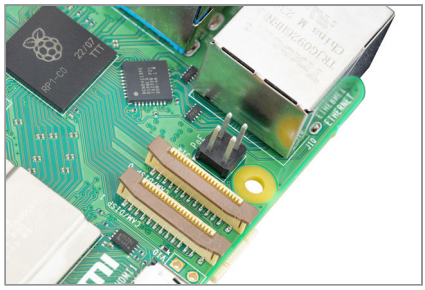


Figure 1-9
Raspberry Pi's Power-over-Ethernet connector

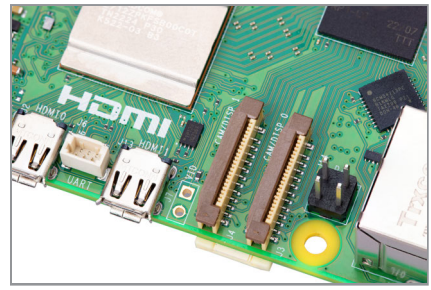


Figure 1-10
Raspberry Pi's camera and display connectors

You can use these connectors to connect a DSI-compatible display like the Raspberry Pi Touchscreen Display or the specially designed Raspberry Pi Camera Module family (see **Figure 1-11**). You'll learn more about camera modules in Chapter 8, *Raspberry Pi Camera Modules*. Either port can act as a camera input or display output so you can have two CSI cameras, two DSI displays, or one CSI camera and one DSI display running on a single Raspberry Pi 5.

To the left of the camera and display connectors, still on the bottom edge of the board, are the *micro High Definition Multimedia Interface (micro HDMI) ports*, which are smaller versions of the connectors you can find on a games console, set-top box, or TV (**Figure 1-12**). The 'multimedia' part of its name means that it carries both audio and video signals, while 'high-definition' means you can expect excellent quality from both signals. You'll use these micro HDMI ports to connect Raspberry Pi to one or two display devices, such as a computer monitor, TV, or projector.

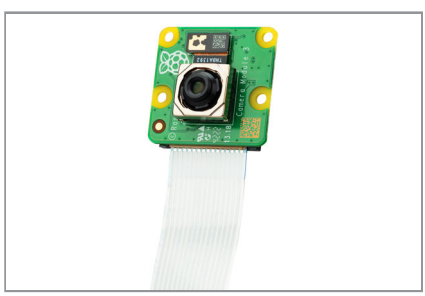


Figure 1-11
Raspberry Pi's camera module

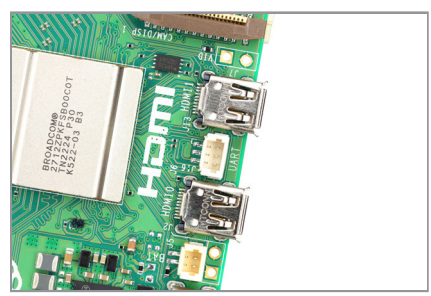


Figure 1-12
Raspberry Pi's micro HDMI ports

Between the two micro HDMI ports is a small connector labelled 'UART,' which provides access to a *Universal Asynchronous Receiver-Transmitter (UART) serial port*. You won't use that port in this book, but you may need it in the future for communicating with, or troubleshooting, more complex projects.

To the left of the micro HDMI ports is another small connector labelled 'BAT', where you can connect a small battery to keep Raspberry Pi's *real-time clock (RTC)* ticking, even when it's disconnected from its power supply. You don't need to connect a battery to use Raspberry Pi, though, since it will automatically update its clock when turned on, so long as it has access to the internet.

At the bottom left of the board is a *USB C power port* (**Figure 1-13**), used to provide Raspberry Pi with power through a compatible USB C power supply. The USB C port is a common sight on smartphones, tablets, and other portable devices. While you could use a standard mobile charger to power your Raspberry Pi, for best results you should use the official Raspberry Pi USB-C Power Supply: it's better at coping with the sudden changes in power requirements that can occur when your Raspberry Pi is working particularly hard.

At the left edge of the board is a small button facing outwards. This is Raspberry Pi 5's *power button*, used to safely shut down your Raspberry Pi when you're finished with it. This button is not available on Raspberry Pi 4 or older boards.

Above the power button is another connector (**Figure 1-14**), which, at first glance, looks like a smaller version of the CSI and DSI connectors. This almost-familiar connector connects to Raspberry Pi's *PCI Express (PCIe) bus*: a high-speed interface for add-on hardware like Solid State Disks (SSDs). To use the PCIe bus, you'll need the Raspberry Pi PCIe HAT add-on to convert this compact connector to a more common *M.2-standard PCIe slot*. You don't need the HAT to make full use of Raspberry Pi, though, so feel free to ignore this connector until you need it.

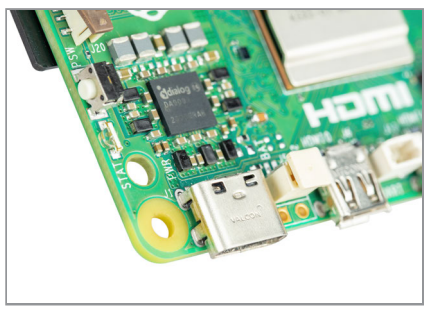


Figure 1-13
Raspberry Pi's USB Type-C power port

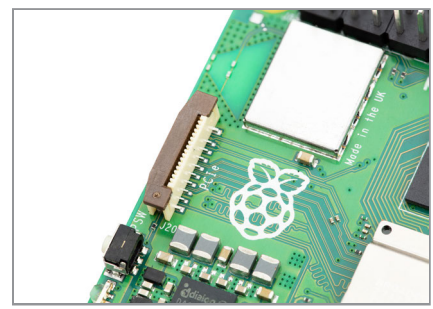


Figure 1-14
Raspberry Pi's connector for PCI Express.

At the top edge of the board are 40 metal pins, split into two rows of 20 pins (**Figure 1-15**). These pins make up the *GPIO (general-purpose input/output) header*, an important feature of Raspberry Pi, that lets it talk to additional hardware from LEDs and buttons all the way to temperature sensors, joysticks, and pulse-rate monitors. You'll learn more about the GPIO header in Chapter 6, *Physical computing with Scratch and Python*.

There's one final port on Raspberry Pi, but you won't see it until you turn the board over. Here on the underside of the board you'll find a *microSD card connector* positioned almost exactly underneath the top-side's connector marked 'PCIe' (**Figure 1-16**). This connector is for Raspberry Pi's storage device: the microSD card inserted in here contains all the files you save, all the software you install, and the operating system that makes your Raspberry Pi run. It's also possible to run your Raspberry Pi without a microSD card by loading its software over the network, from a USB drive, or from an M.2 SSD. For this book we'll keep it simple and focus on using a microSD card as the main storage device.

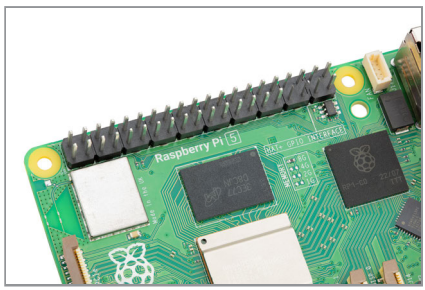


Figure 1-15
Raspberry Pi's GPIO header

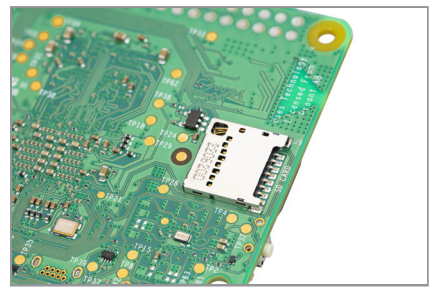


Figure 1-16
Raspberry Pi's microSD card connector

Raspberry Pi 400

Raspberry Pi 400 takes the same components as Raspberry Pi 4, including the system-on-chip and memory, but places them inside a convenient keyboard housing. As well as protecting the electronics, the keyboard housing takes up less room on your desk and helps keep your cables tidy.

While you can't easily see the internal components, you *can* see the external bits and pieces, starting with the keyboard itself (**Figure 1-17**).

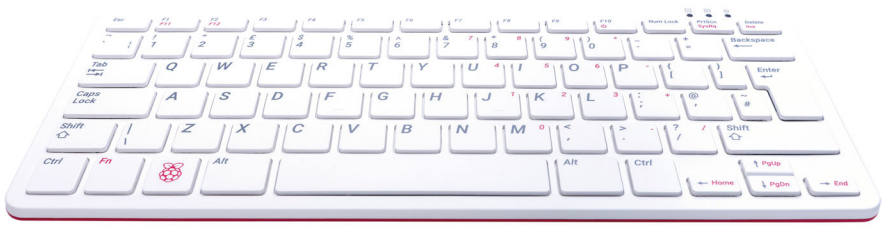


Figure 1-17 Raspberry Pi 400 has an integrated keyboard

In the upper right corner are three light-emitting diodes (LEDs): the first lights up when you press the **Num Lock** key, which switches some of the keys to act like the ten-key number pad on a full-size keyboard; the second lights up when you press **Caps Lock**, which makes the letter keys upper-case rather than lower-case; and the last lights up when Raspberry Pi 400 is powered on.

At the back of Raspberry Pi 400 (**Figure 1-18**) are the ports. The left-most port is the general-purpose input/output (GPIO) header. This is the same header shown in **Figure 1-15**, but flipped: the first pin, Pin 1, is at the top-right, while the last pin, Pin 40, is at the bottom-left. You can find out more about the GPIO header in Chapter 6, *Physical computing with Scratch and Python*.



Figure 1-18 The ports are found at the rear of Raspberry Pi 400

Next to the GPIO header is the microSD card slot. Like the slot on the underside of Raspberry Pi 5, this holds the microSD card that serves as storage for Raspberry Pi 400's operating system, applications, and other data. A microSD card comes pre-installed in the Raspberry Pi 400 Personal Computer Kit. To remove it push gently on the card until it clicks and springs out, then pull the card the rest of the way out. When you put the card back in, make sure the shiny metal contacts are facing downwards. Push the card in gently until it clicks, which means it's locked into place.

The next two ports are the micro HDMI ports, used to connect a monitor, TV, or other display. Like Raspberry Pi 4 and Raspberry Pi 5, Raspberry Pi 400 supports up to two HDMI displays. Next to these is the USB C power port, used to connect an official Raspberry Pi Power Supply, or any other compatible USB C power supply.

The two blue ports are USB 3.0 ports, which provide a high-speed connection to devices like solid-state drives (SSDs), memory sticks, printers, and more. The white port to the right of these is a lower-speed USB 2.0 port, which you can use for the Raspberry Pi Mouse included with the Raspberry Pi 400 Personal Computer Kit.

The final port is a gigabit Ethernet network port, used to connect Raspberry Pi 400 to your network using an Ethernet cable as an alternative to using the device's built-in WiFi radio. You can read more about connecting Raspberry Pi 400 to a network in Chapter 2, *Getting started with your Raspberry Pi*.

Raspberry Pi Zero 2 W

Raspberry Pi Zero 2 W (**Figure 1-19**) is designed to offer many of the same features as the other models in the Raspberry Pi family, but in a much more compact design. It's cheaper and draws less power, but it also lacks a few ports found on the larger models.

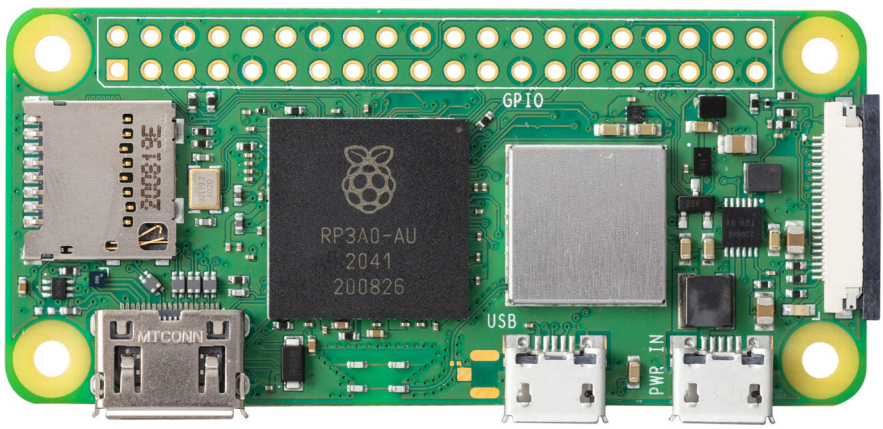


Figure 1-19 Raspberry Pi Zero 2 W

Unlike Raspberry Pi 5 and Raspberry Pi 400, Raspberry Pi Zero 2 W lacks a wired Ethernet port. You can still connect it to a network, but only using a WiFi connection. You'll learn more about connecting Raspberry Pi Zero 2 W to a network in Chapter 2, *Getting started with your Raspberry Pi*.

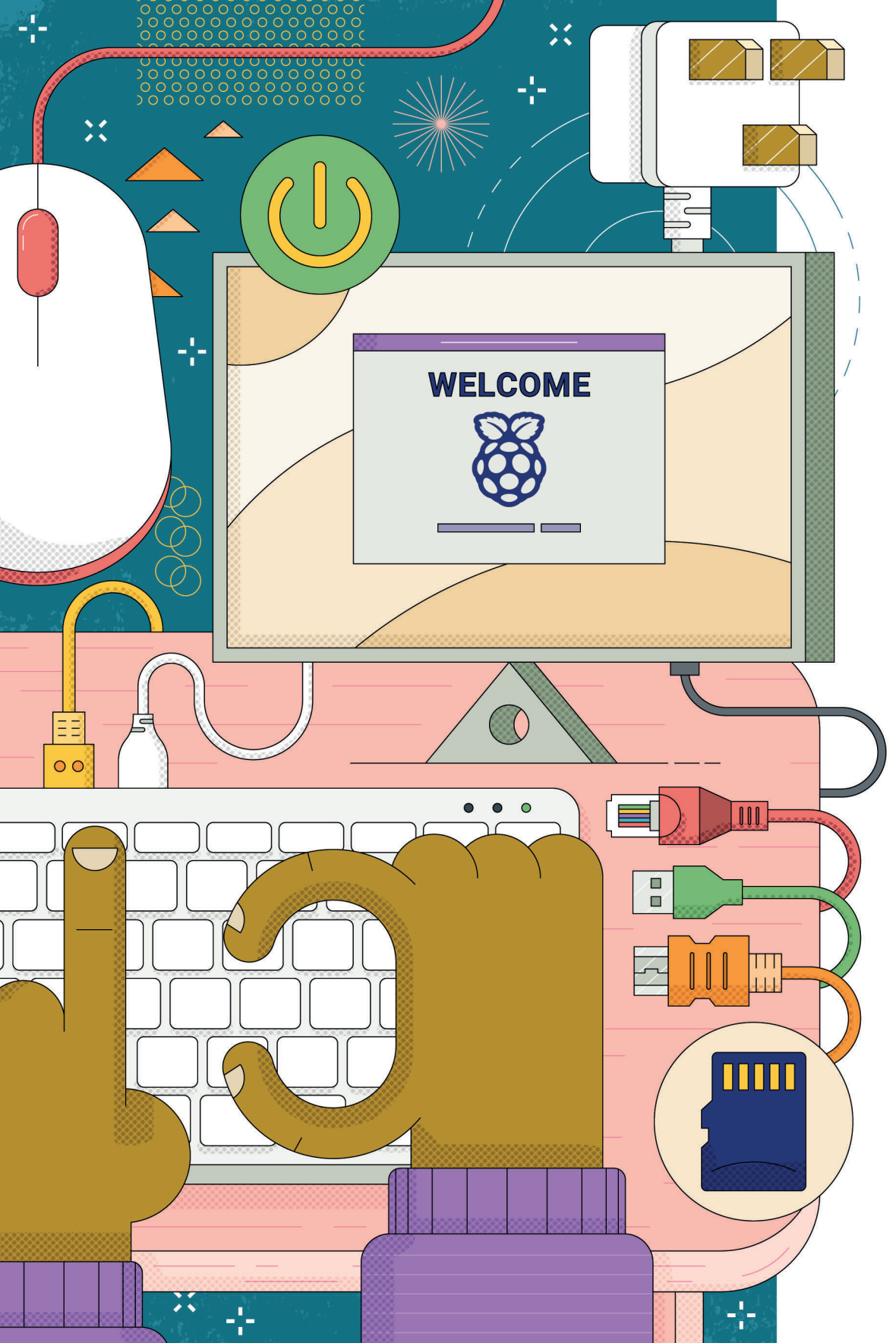
You should also notice a difference in the system-on-chip: it's black instead of silver and there's no separate RAM chip visible. This is because the two parts — SoC and RAM — are combined into one chip, marked with an etched Raspberry Pi logo, and placed roughly in the middle of the board.

The far left of the board has the usual microSD card slot for storage, and below that is a single mini HDMI port for video and audio. Unlike Raspberry Pi 5 and Raspberry Pi 400, Raspberry Pi Zero 2 W only supports a single display.

To the right are two micro-USB ports: the left-hand port, marked 'USB', is a USB On-The-Go (OTG) port which is compatible with OTG adapters to connect keyboards, mice, USB hubs, or other peripherals; the right-hand port, marked 'PWR IN', is the power connector. You can't use a power supply designed for Raspberry Pi 4 or Raspberry Pi 400 with Raspberry Pi Zero 2 W, since they use different connectors.

At the very right of the board is a Camera Serial Interface you can use to connect a Raspberry Pi Camera Module. You'll learn more about that in Chapter 8, *Raspberry Pi Camera Modules*.

Finally, Raspberry Pi Zero 2 W has the same 40-pin general-purpose input/output (GPIO) header as its bigger siblings, but it's supplied *unpopulated*. This means it doesn't have any pins fitted. If you want to use the GPIO header you'll need to solder a 2×20 2.54mm *pin header* in place — or purchase the Raspberry Pi Zero 2 WH, which has a header already soldered into place.


WELCOME

Chapter 2

Getting started with your Raspberry Pi

Discover the essential items you'll need for your Raspberry Pi, and learn how to connect them all to get it set up and working.

Raspberry Pi is designed to be as quick and easy to set up and use as possible, but — like any computer — it relies on various external components called *peripherals*. While it's easy to take a look at the bare circuit board of Raspberry Pi — which looks significantly different to the encased, closed-off computers you may be used to — and worry that things are about to get complicated, that's not the case. You can be up and running with your Raspberry Pi in well under ten minutes if you follow the steps in this guide.

If you received this book in a Raspberry Pi Desktop Kit or with a Raspberry Pi 400, you'll already have almost everything you need to get started. All you need to provide is a computer monitor or a TV with an HDMI connection — the same type of connector used by set-top boxes, Blu-ray players, and games consoles — so you can see what your Raspberry Pi is doing.

If you picked up your Raspberry Pi without accessories, then you'll also need:

- **USB power supply** — A 5V power supply rated at 5 amps (5A) and with a USB C connector for Raspberry Pi 5, a 5V power supply rated at 3 amps (3A) and with a USB C connector for Raspberry Pi 4 Model B or Raspberry Pi 400, or a 5V power supply rated at 2.5 amps (2.5A) and with a micro USB connector for Raspberry Pi Zero 2 W. The Official Raspberry Pi Power Supplies are recommended, as they are designed to cope with the quickly switching power demands of Raspberry Pi. Third-party power supplies may not be able to negotiate current, and may cause power issues with your Raspberry Pi.

- **microSD card** — The microSD card acts as your Raspberry Pi's permanent storage. All the files you create and all the software you install, along with the operating system itself, are stored on the card. An 8GB card is enough to get you started, though a 16GB one offers more room to grow. The Raspberry Pi Desktop Kits includes a microSD card with Raspberry Pi OS pre-installed; see Appendix A, *Install an operating system to a microSD card* for instructions on installing an operating system (OS) onto a blank card.



Figure 2-1
USB power supply

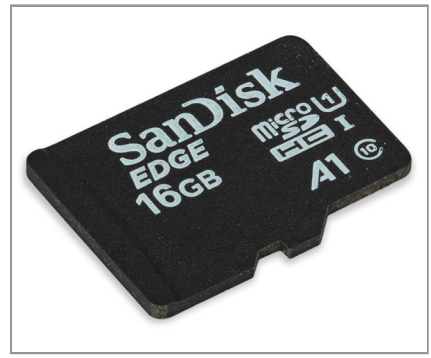


Figure 2-2
microSD card

- **USB keyboard and mouse** — The keyboard and mouse allow you to control your Raspberry Pi. Almost any wired or wireless keyboard and mouse with a USB connector will work with Raspberry Pi, though some gaming-style keyboards with colourful lights may draw too much power to be used reliably. Raspberry Pi Zero 2 W needs a micro USB OTG adapter, and if you want to plug in more than one USB device at a time you'll need a powered USB hub.

- **HDMI cable** — This carries sound and images from your Raspberry Pi to your TV or monitor. Raspberry Pi 4, 5, and Raspberry Pi 400 need a cable with a micro HDMI connector at one end, while Raspberry Pi Zero 2 W needs a cable with a mini HDMI connector; the other end should have a full-size HDMI connector for your display. You can also use a micro or mini HDMI-to-HDMI adapter along with a standard, full-size HDMI cable. If you're using a monitor without an HDMI socket, you can buy adapters to convert to DVI-D, DisplayPort, or VGA connectors.



Figure 2-3
USB keyboard



Figure 2-4
HDMI cable

Raspberry Pi is safe to use without a case, providing you don't place it on a metal surface, which could conduct electricity and cause a short-circuit. An optional case, however, can provide additional protection; the Desktop Kit includes the Official Raspberry Pi Case, while third-party cases are available from all good stockists.

If you want to use Raspberry Pi 4, Raspberry Pi 5, or Raspberry Pi 400 on a wired network, rather than a Wi-Fi network, you'll also need an Ethernet network cable. This should be connected at one end to your network's switch or router. If you're planning to use Raspberry Pi's built-in wireless radio, you won't need a cable; you will, however, need to know the name and key or passphrase for your wireless network.

RASPBERRY PI 400 SETUP

The following instructions are for setting up Raspberry Pi 5 or another bare-board member of the Raspberry Pi family. For Raspberry Pi 400, see "Setting up Raspberry Pi 400" on page 26.

Setting up the hardware

Begin by unpacking your Raspberry Pi from its box. Raspberry Pi is a robust piece of hardware, but that doesn't mean it's indestructible; try to get into the habit of holding the board by the edges, rather than on its flat sides, and be extra careful around the raised metal pins. If these pins are bent, at best it'll make using add-on boards and other extra hardware difficult and, at worst, might cause a short-circuit that will damage your Raspberry Pi.

If you haven't done so already, have a look at Chapter 1, *Get to know your Raspberry Pi*, for details on exactly where the various ports are and what they do.

Assembling the Raspberry Pi Case

If you're installing Raspberry Pi 5 in a case, this should be your first step. If you're using the Official Raspberry Pi Case, begin by splitting it into its three individual pieces: the red base, fan assembly and frame, and white lid.

Take the base and hold it so that the raised end is to your left and the lower end to your right.

Hold your Raspberry Pi 5, with no microSD card inserted, by its USB and Ethernet ports, at a slight angle. Gently lower the other side down into the base, so it looks like **Figure 2-5**. You should feel and hear a click as you seat it flat against the base.



SETTING UP THE FAN ASSEMBLY

The fan should have arrived inserted into the fan assembly, and the fan assembly should already be inserted into its frame when you take it out of the box. If not, you can click it all together (see **Figure 2-7**).

Next, plug the fan's white JST connector into the fan socket on the Raspberry Pi 5 as shown in **Figure 2-6**. It will only fit one way around, so you don't need to worry about connecting it backwards.

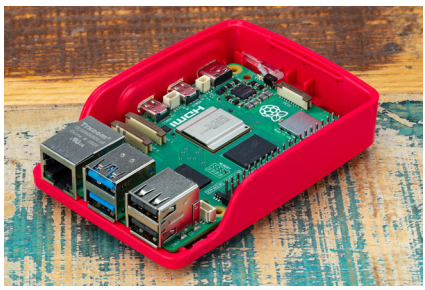


Figure 2-5
The Raspberry Pi 5 placed in its case

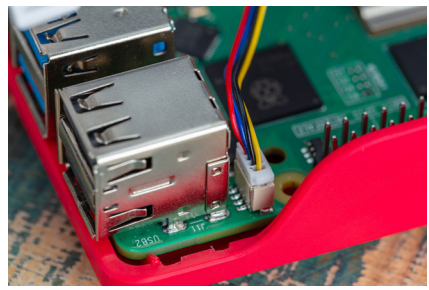


Figure 2-6
Plugging in the fan connector

Click the fan assembly and frame into place as shown in **Figure 2-7**, and gently push down until you feel and hear a click.

If you want to cover everything in the case up, take the optional white lid and position it so that the Raspberry Pi logo is over the USB and Ethernet connectors on Raspberry Pi 5, as shown in **Figure 2-8**. To fasten it into place, gently push down on the middle of the lid until you hear a click.



Figure 2-7
Attaching the fan assembly and frame



Figure 2-8
Placing the lid on top of the case

HATS AND LIDS

You can fit a HAT (Hardware Attached on Top) directly on top of Raspberry Pi 5 by removing the fan assembly, or you can stack it on top of the fan assembly and frame using 14mm high standoffs and a 19mm GPIO extender. These will be available separately from Authorised Resellers.

Assembling the Raspberry Pi Zero Case

If you're installing Raspberry Pi Zero 2 W in a case, this should be your first step. If you're using the Official Raspberry Pi Zero Case, begin by unpacking it. You should have four pieces: a red base and three white lids.

If you're using Raspberry Pi Zero 2, you want to use the solid lid. If you're going to be using the GPIO header, about which you'll learn more in Chapter 6, *Physical computing with Scratch and Python*, pick the lid with the long rectangular hole in it. If you have a Camera Module 1 or 2, pick the lid with the circular hole.

The Camera Module 3 and High Quality (HQ) Camera Module are not compatible with the Raspberry Pi Zero Case camera lid and must be used outside the case; there's a cut-out at the end of the Raspberry Pi Zero Case for the camera cable.

Take the base and place it flat on the table so that the cut-outs for the ports are facing towards you as shown in **Figure 2-9**.

Holding your Raspberry Pi Zero (with the microSD card inserted) by the edges of the board, line it up so the small circular posts in the corners of the base go into the mounting holes on the corners of Raspberry Pi Zero 2 W's circuit board. When they're lined up (**Figure 2-10**), gently push Raspberry Pi Zero 2 W downwards until you hear a click and the ports are lined up with the cut-outs in the base.



Figure 2-9
The Raspberry Pi Zero case

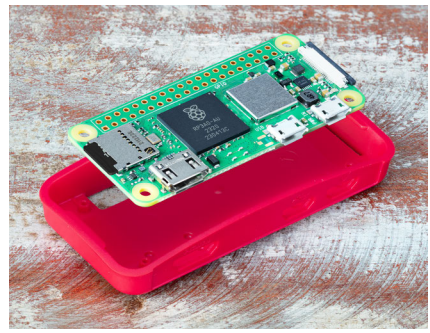


Figure 2-10
Placing the Zero in its case

Take your chosen white lid and place it on top of the Raspberry Pi Zero Case base, as shown in **Figure 2-11**. If you're using the Camera Module lid, make sure that the cable isn't trapped. When the lid is in place, gently push it down until you hear a click.



CAMERA MODULE AND THE ZERO CASE

If you're using a Raspberry Pi Camera Module, use the lid with the circular hole. Line the Camera Module's mounting holes up with the cross-shaped posts in the lid, so that the camera's connector is facing towards the logo on the lid. Click it into place. Gently push the bar on the camera connector away from your Raspberry Pi, then push the narrower end of the included camera ribbon cable into the connector before pushing the bar back into place. Connect the wider end of the cable to the Camera Module in the same way. For more information on installing the Camera Module, see Chapter 8, *Raspberry Pi Camera Modules*.

At this point, you can also stick the included rubber feet to the bottom of the case (see **Figure 2-12**): flip it over, peel the feet off the backing sheet, and stick them in the circular indentations on the base to provide a better grip on your desk.



Figure 2-11
Attaching the lid



Figure 2-12
Attaching the feet

Connecting the microSD card

To install the microSD card, which is Raspberry Pi's *storage*, turn your Raspberry Pi (in its case if you're using one) upside-down and slide the card into the microSD slot with the card's label side facing away from Raspberry Pi. It will only fit in one way, and should slide home without too much pressure (see **Figure 2-13**).

The microSD card will slide into the connector, then stop without a click.

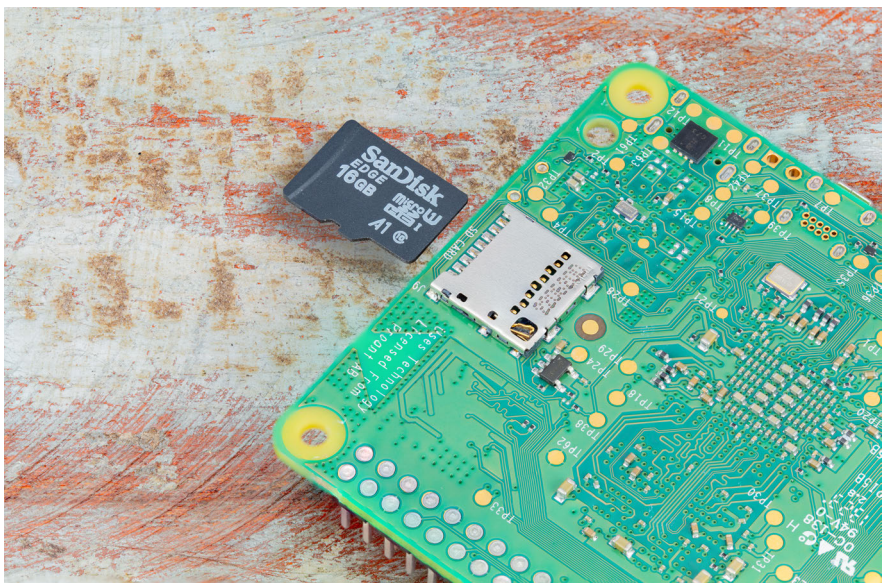


Figure 2-13 Inserting the microSD card

For Raspberry Pi Zero 2 W, the microSD slot is on the top at the left-hand side. Insert the card with the label facing away from your Raspberry Pi.

If you want to remove it again in the future, simply grip the end of the card and pull it gently out. If you're using an older model of Raspberry Pi, you'll need to give the card a gentle push first to unlock it; this isn't necessary with a Raspberry Pi 3, 4, 5, or any model of Raspberry Pi Zero.

Connecting a keyboard and mouse

Connect the keyboard's USB cable to any of the four USB ports (black USB 2.0 or blue USB 3.0) on Raspberry Pi as shown in **Figure 2-14**. If you're using the Official Raspberry Pi Keyboard, there's a USB port on its back for the mouse. Otherwise, just connect the USB cable from your mouse to another USB port on Raspberry Pi.

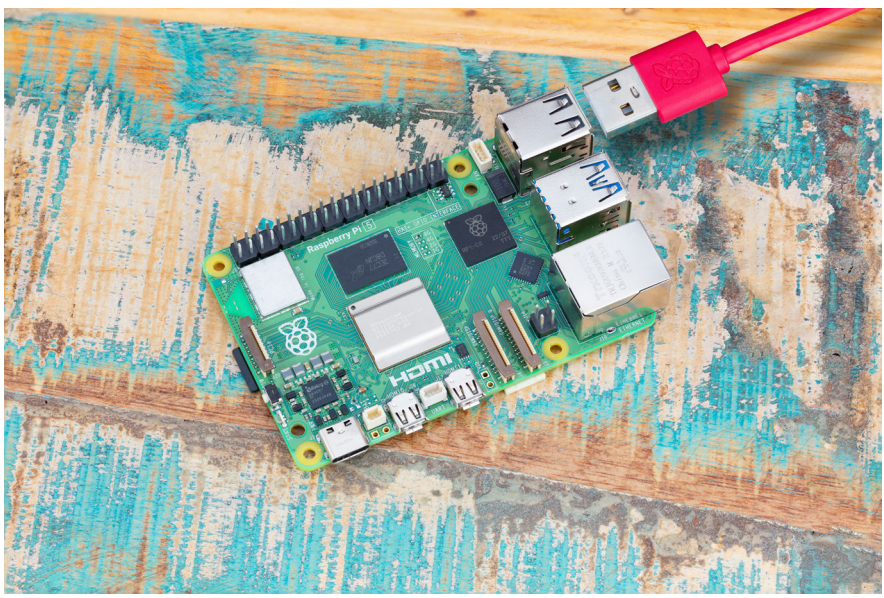


Figure 2-14 Plugging a USB cable into a Raspberry Pi 5

For Raspberry Pi Zero 2 W, you'll need to use a micro USB OTG adapter cable. Insert this into the left-hand micro USB port, then connect the USB cable from your keyboard to the USB OTG adapter.

If you are using a keyboard with a separate mouse, rather than one with a built-in touchpad, you'll also need to use a powered USB hub. Connect the micro USB OTG adapter cable as above, then connect the hub's USB cable to

the USB OTG adapter before connecting your keyboard and mouse to the USB hub. Finally, connect the hub's power adapter and switch it on.

The USB connectors for the keyboard and mouse should slide home without too much pressure; if you're having to force the connector in, there's something wrong. Check that the USB connector is the right way up!

KEYBOARD AND MOUSE

The keyboard and mouse act as your main means of telling Raspberry Pi what to do; in computing, these are known as *input devices*, in contrast with the display, which is an *output device*.

Connecting a display

For Raspberry Pi 4 and Raspberry Pi 5, take the micro HDMI cable and connect the smaller end to the micro HDMI port closest to the USB Type-C port on your Raspberry Pi as shown in **Figure 2-15**. Connect the other end to your display.

For Raspberry Pi Zero 2 W (**Figure 2-16**), take the mini HDMI cable and connect the smaller end to the mini HDMI port at the left-hand side of your Raspberry Pi, under the microSD slot. The other end connects to your display.



Figure 2-15
Connecting the HDMI cable to a Raspberry Pi 5



Figure 2-16
Connecting the HDMI cable to a Raspberry Pi Zero

If your display has more than one HDMI port, look for a port number next to the connector itself; you'll need to switch the TV to this input to see your Raspberry Pi's display. If you can't see a port number, don't worry, you can simply switch through each input in turn until you find Raspberry Pi.



TV CONNECTION

If your TV or monitor doesn't have an HDMI connector, that doesn't mean you can't use a Raspberry Pi. Adapter cables, available from any electronics stockist, will allow you to convert the micro or mini HDMI port on your Raspberry Pi to DVI-D, DisplayPort, or VGA for use with other computer monitors.

Connecting a network cable (optional)

To connect your Raspberry Pi to a wired network, take a network cable — known as an *Ethernet* cable — and push it into Raspberry Pi's Ethernet port, with the plastic clip facing downwards, until you hear a click (see **Figure 2-17**). If you need to remove the cable, squeeze the plastic clip inwards towards the plug and gently slide the cable free again.

The other end of your network cable should be connected to any free port on your network hub, switch, or router in the same way.

Connecting a power supply

Connecting your Raspberry Pi to a power supply is the final step in the hardware setup process. It's the last thing you'll do before you start setting up its software. Your Raspberry Pi will turn on as soon as it's connected to a live power supply.

For Raspberry Pi 4 and Raspberry Pi 5, connect the USB C end of the power supply cable to the USB C power connector on your Raspberry Pi as shown in **Figure 2-18**. It can go in either way around, and should slide home gently. If your power supply has a detachable cable, make sure the other end is plugged into the body of the power supply.



WARNING!

Raspberry Pi 5 needs a 5V power supply capable of delivering 5A of current, and a suitable E-Marked USB C cable. If you connect a lower-current power supply, including the Official Raspberry Pi 4 Power Supply, Raspberry Pi 5's USB ports will be limited to low-power devices only.



Figure 2-17
Connecting Raspberry Pi 5 to Ethernet

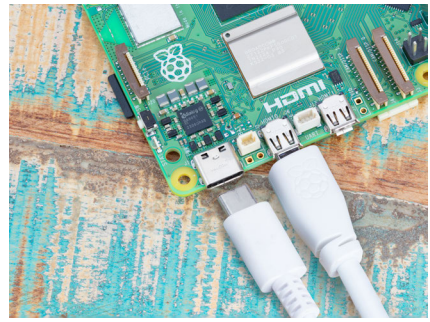


Figure 2-18
Powering your Raspberry Pi 5

For Raspberry Pi Zero 2 W, connect the micro USB end of the power supply cable to the right-hand micro USB port on your Raspberry Pi. It can only go in one way up, so make sure to check its orientation before pushing it gently in.

Congratulations: you have put your Raspberry Pi together!

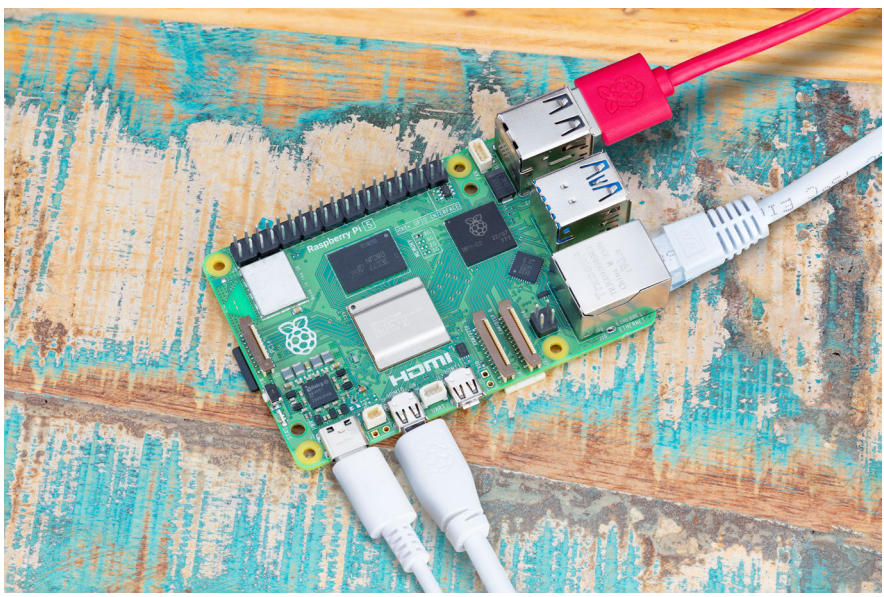


Figure 2-19 Your Raspberry Pi is ready to go!

Finally, connect the power supply to a mains socket, switch the socket on, and your Raspberry Pi will immediately start running.

You'll briefly see a rainbow-coloured cube followed by an informational screen with a Raspberry Pi logo on it. You may also see a blue screen appear

as the operating system resizes itself to make full use of your microSD card. If you see a black screen, wait a few minutes; the first time Raspberry Pi boots it will perform some housekeeping in the background, which can take a little time.

After a while you'll see the Raspberry Pi OS Welcome Wizard, as in **Figure 2-20**. Your operating system is now ready to be configured, which you'll learn to do in Chapter 3, *Using your Raspberry Pi*.



Figure 2-20 The Raspberry Pi OS Welcome Wizard

Setting up Raspberry Pi 400

Unlike Raspberry Pi 4, Raspberry Pi 400 comes with a built-in keyboard and the microSD card already installed. You'll still need to connect a few cables to get started, but it should only take you a few minutes.

Connecting a mouse

Raspberry Pi 400's keyboard is already connected, leaving you with just the mouse to add. Take the USB cable at the end of the mouse and insert it into any of the three USB ports (2.0 or 3.0) on the rear panel of your Raspberry Pi 400. If you want to save the two high-speed USB 3.0 ports for other accessories, use the white USB 2.0 port.

The USB connector should slide home without too much pressure (see **Figure 2-21**). If you're having to force the connector in, there's something wrong. Check that the USB connector is the right way up!

Connecting a display

Take the micro HDMI cable and connect the smaller end to the micro HDMI port closest to the microSD slot on your Raspberry Pi 400, and the other end to your display, as shown in **Figure 2-22**. If your display has more than one HDMI port, look for a port number next to the connector itself; you'll need to switch the TV or monitor to this input to see Raspberry Pi's display. If you can't see a port number, don't worry: you can simply switch through each input in turn until you find Raspberry Pi.



Figure 2-21
Plugging a USB cable into a Raspberry Pi 400



Figure 2-22
Connecting the HDMI cable to a Raspberry Pi 400

Connecting a network cable (optional)

To connect your Raspberry Pi 400 to a wired network, take a network cable — known as an Ethernet cable — and push it into Raspberry Pi 400's Ethernet port, with the plastic clip facing upwards, until you hear a click (**Figure 2-23**). If you need to remove the cable, just squeeze the plastic clip inwards towards the plug and gently slide the cable free again.

The other end of your network cable should be connected to any free port on your network hub, switch, or router in the same way.

Connecting a power supply

Connecting Raspberry Pi 400 to a power supply is the very last step in the hardware setup process, and it's one you should do only when you're ready to set up its software. Raspberry Pi 400 does not have a power switch and will turn on as soon as it is connected to a live power supply.

First, connect the USB Type-C end of the power supply cable to the USB Type-C power connector on Raspberry Pi. It can go in either way around and should slide home gently. If your power supply has a detachable cable, make sure the other end is plugged into the body of the power supply.



Figure 2-23 Connecting Raspberry Pi 400 to Ethernet

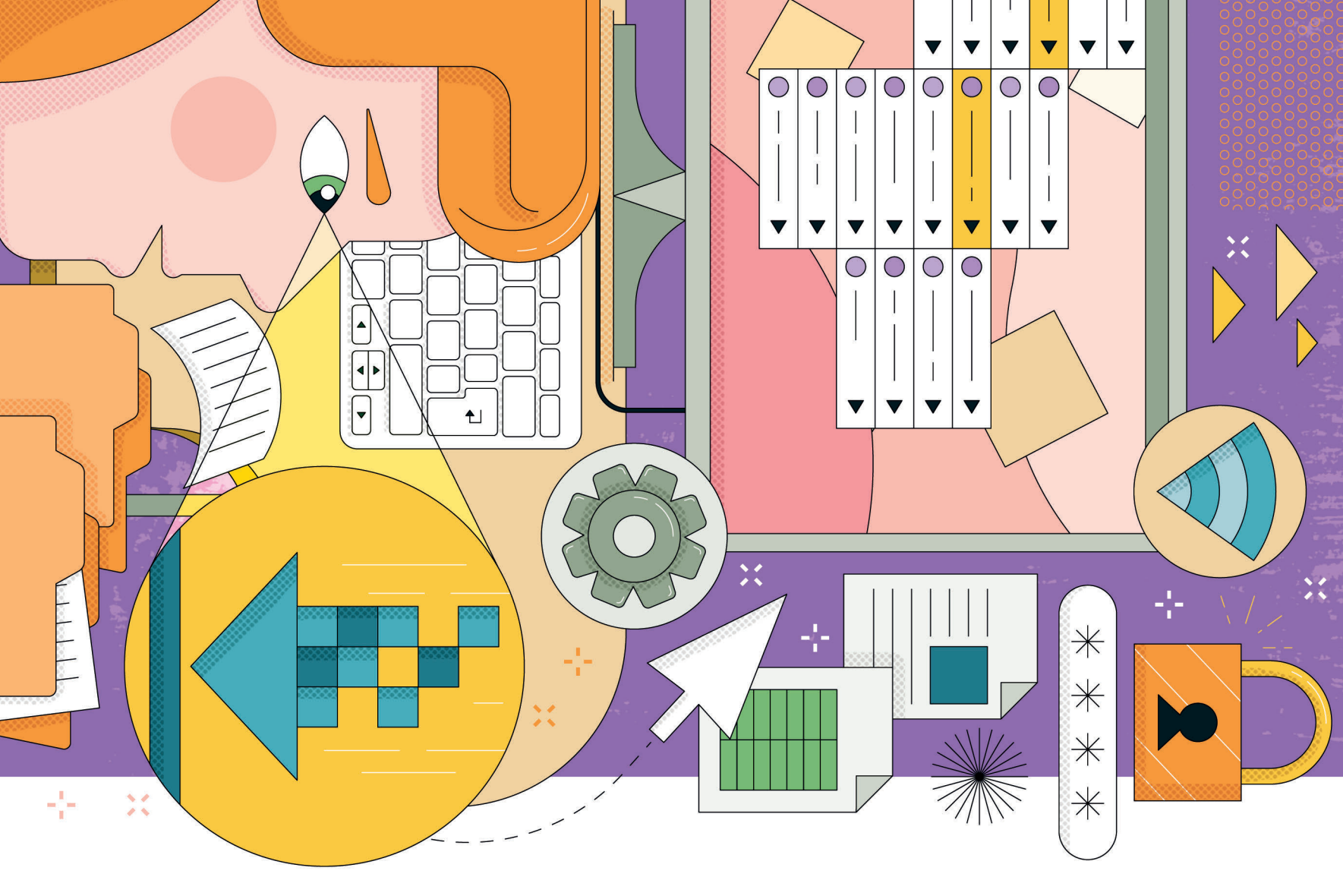
Finally, connect the power supply to a mains socket and switch the socket on: your Raspberry Pi 400 will immediately start running. Congratulations, you have put your Raspberry Pi 400 together (**Figure 2-24**)!



Figure 2-24 Your Raspberry Pi 400 is all wired up!

You'll briefly see a rainbow-coloured cube followed by an informational screen with a Raspberry Pi logo on it. You may also see a blue screen appear as the operating system resizes itself to make full use of your microSD card. If you see a black screen, wait a few minutes: the first time Raspberry Pi boots it will perform some housekeeping in the background, which can take a little time.

After a while you'll see the Raspberry Pi OS Welcome Wizard, as shown earlier in **Figure 2-20**. Your operating system is now ready to be configured, which you'll learn to do in Chapter 3, *Using your Raspberry Pi*.



Chapter 3

Using your Raspberry Pi

Learn about the Raspberry Pi operating system.

Your Raspberry Pi can run a wide range of software, including a number of different operating systems — the core software that makes a computer run. The most popular of these, and the official operating system from Raspberry Pi, is Raspberry Pi OS. Based on Debian Linux, it is tailor-made for Raspberry Pi and comes with a range of extra software pre-installed and ready to go.

If you've only ever used Microsoft Windows or Apple macOS, don't worry: Raspberry Pi OS is based on the same intuitive windows, icons, menus, and pointer (WIMP) principles, and should quickly become familiar.

Read on to get started and find out more about some of the bundled software.

The Welcome Wizard

The first time you run Raspberry Pi OS, you'll see the Welcome Wizard (**Figure 3-1**). This helpful tool will walk you through changing some settings in Raspberry Pi OS, known as the *configuration*, to match how and where you will be using your Raspberry Pi.

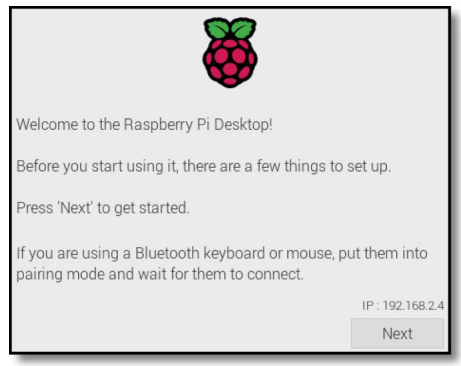


Figure 3-1 The Welcome Wizard

Click the **Next** button, then choose your country, language, and time zone by clicking on each drop-down box in turn and selecting your answer from the list (**Figure 3-2**). If you are using a US-layout keyboard, click on the check box to make sure Raspberry Pi OS uses the correct keyboard layout. If you want the desktop and programs to appear in English, regardless of your country's native language, click on the **Use English language** checkbox to tick it. When you're finished, click **Next**.



Figure 3-2 Selecting a language, among other options

The next screen will ask you to choose a name and password for your user account (**Figure 3-3**). Choose a name — it can be anything you like, but it must start with a letter and can only contain lower-case letters, digits, and hyphens. Then you'll need to create a memorable password. You'll be asked to type the password twice to make sure you didn't make any mistakes that could lock you out of your new account. When you're happy with your choices, click **Next**.

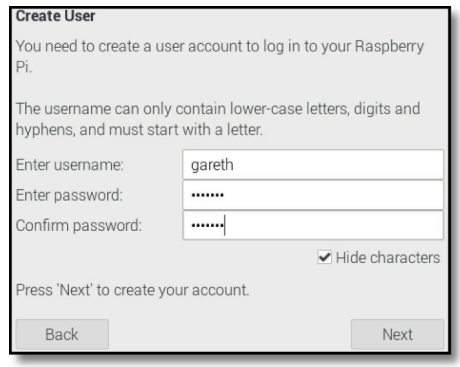


Figure 3-3 Setting a new password

The following screen will allow you to choose your Wi-Fi network from a list (**Figure 3-4**).

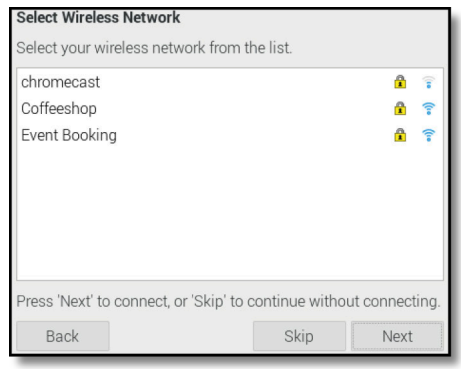


Figure 3-4 Choosing a wireless network

WIRELESS NETWORKING

Built-in wireless networking is only available on the Raspberry Pi 3, Raspberry Pi 4, Raspberry Pi 5, and Raspberry Pi Zero W and Zero 2 W families. If you want to use a different model of Raspberry Pi with a wireless network, you'll need a USB Wi-Fi adapter.

Scroll through the list of networks with the mouse or keyboard, find your network's name, click on it, then click **Next**. Assuming that your wireless network is secure (it really should be), you'll be asked for its password (also known as its pre-shared key). If you don't use a custom password, the default is normally

written on a card that comes with the router, or on the bottom or back of the router itself. Click **Next** to connect to the network. If you don't want to connect to a wireless network, click **Skip**.

Next you will be asked to choose your default *web browser* from the two preinstalled in Raspberry Pi OS: Google's Chromium, the default, and Mozilla's Firefox (**Figure 3-5**). For now, just leave Chromium selected as the default, so you can follow along with this book; you can always switch to Firefox later, if you'd prefer. If you do change the default browser, you can also choose to uninstall the non-default browser to save space on your microSD card. Just tick the box when you are offered the option, and click the **Next** button.



Figure 3-5 Selecting a browser

The next screen will allow you to check for and install updates for Raspberry Pi OS and the other software on Raspberry Pi (**Figure 3-6**). Raspberry Pi OS is regularly updated to fix bugs, add new features, and improve performance. To install these updates, click **Next**. Otherwise, click **Skip**. Downloading the updates can take several minutes, so be patient.

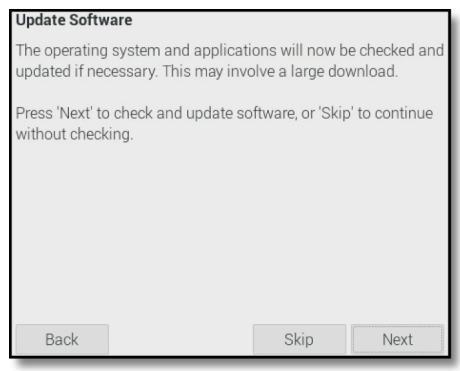


Figure 3-6 Checking for updates

When the updates are installed, a window saying 'System is up to date' will appear; click the **OK** button.

The final screen of the Welcome Wizard (**Figure 3-7**) provides one last bit of information: certain changes made will only take effect when you restart your Raspberry Pi (a process also known as rebooting). Click the **Restart** button and your Raspberry Pi will restart. From now on, the Welcome Wizard won't appear; its job is done, and your Raspberry Pi is ready to use.

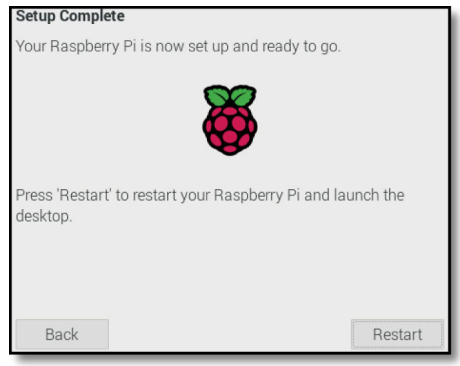


Figure 3-7 Restarting Raspberry Pi

WARNING!

If, after your Raspberry Pi starts up, you see a message in the top-right corner telling you "this power supply is not capable of supplying 5A," it means you're using a power supply which can't supply the 5V at 5A required by Raspberry Pi 5. You should replace your power supply with one supporting Raspberry Pi 5, like the Official Raspberry Pi 5 Power Supply. You can also ignore the warning and continue to use Raspberry Pi 5, but certain high-power USB devices — like hard drives — won't work.

If you see a "low voltage warning" message, accompanied by a lightning-bolt symbol, you should stop using your Raspberry Pi until you can replace the power supply: *voltage droop* from low-quality power supplies can cause Raspberry Pi to crash, losing your work.

Navigating the desktop

The version of Raspberry Pi OS installed on most Raspberry Pi boards is properly known as 'Raspberry Pi OS with desktop', referring to its main graphical user interface (**Figure 3-8**). The bulk of this desktop is taken up with a wallpaper picture (**A** in **Figure 3-8**), on top of which the programs you run will appear. At the top of the desktop is a taskbar (**B**), which allows you to launch your installed programs. These are then indicated by tasks (**C**) in the taskbar.

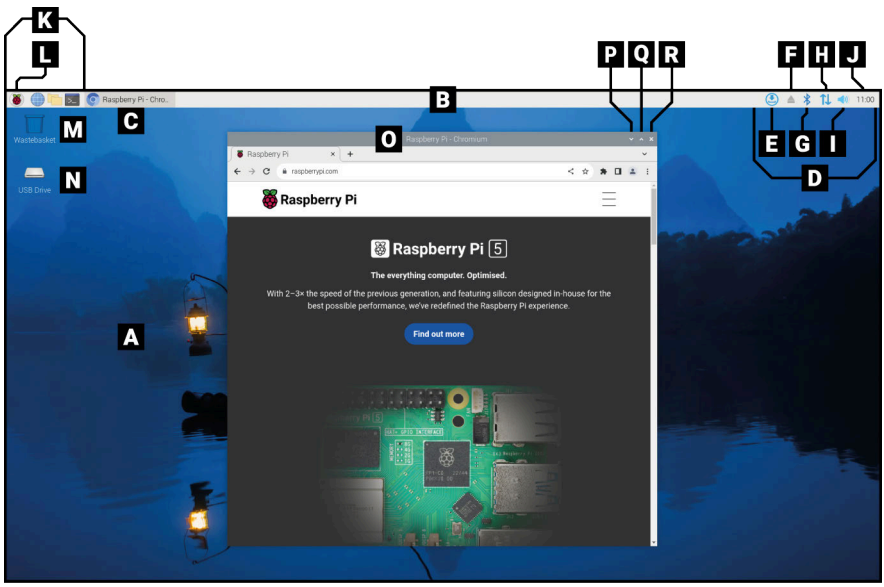


Figure 3-8 The Raspberry Pi OS desktop

A Wallpaper

B Taskbar

C Task

D System Tray

E Software Update icon

F Media eject

G Bluetooth icon

H Network icon

I Volume icon

J Clock

K Launcher

L Menu (or Raspberry Pi icon)

M Wastebasket icon

N Removable drive icon

O Window title bar

P Minimise

Q Maximise

R Close

The right-hand side of the menu bar houses the *system tray* (**D**). The software update icon (**E**) appears only when there are updates to Raspberry Pi OS and its applications. If you have any *removable storage*, such as USB memory sticks, connected to Raspberry Pi you'll see an eject symbol (**F**); click this to safely eject and remove them. On the far right is the clock (**J**); click it to bring up a digital calendar (**Figure 3-9**).

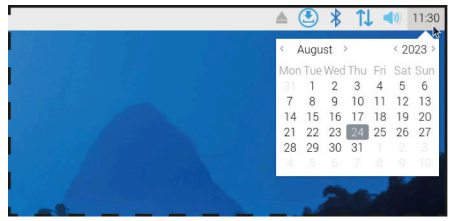


Figure 3-9 The digital calendar

Next to this is a speaker icon (**I**). Click on it with the left mouse button to adjust your Raspberry Pi's audio volume, or click with the right mouse button to choose which output Raspberry Pi should use for its sound. Next to that is a network icon (**H**); if you're connected to a wireless network you'll see the signal strength displayed as a series of bars, but if you're connected to a wired network you'll just see two arrows. Clicking the network icon will bring up a list of nearby wireless networks (**Figure 3-10**), while clicking on the Bluetooth icon (**G**) next to that will allow you to connect to a nearby Bluetooth device.

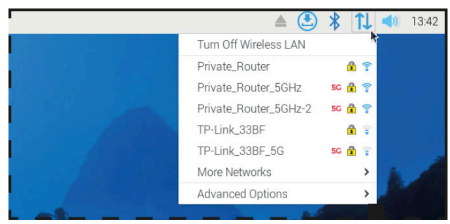


Figure 3-10 Listing nearby wireless networks

The left-hand side of the menu bar is home to the *launcher* (**K**), which is where you'll find the programs installed alongside Raspberry Pi OS. Some of these are visible as shortcut icons; others are hidden away in the menu, which you can bring up by clicking the Raspberry Pi icon (**L**) to the far left (**Figure 3-11**).

The programs in the menu are split into categories. Each category's name tells you what to expect: the **Programming** category contains software designed to help you write your own programs — as explained starting in Chapter 4, *Programming with Scratch 3* — and Games will help you while away the hours.

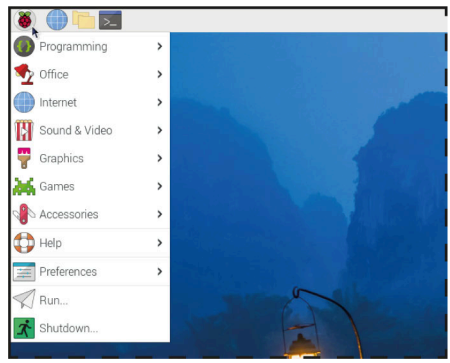


Figure 3-11 The Raspberry Pi menu

Not every program will be detailed in this guide, so feel free to experiment with them to learn more. On the desktop, you'll find the Wastebasket (**M**) and any external storage devices (**N**) connected to your Raspberry Pi.

The Chromium web browser

To practise using your Raspberry Pi, start by loading the Chromium web browser: click on the Raspberry Pi icon at the top-left to bring up the menu, move your mouse pointer to select the Internet category, and click on **Chromium Web Browser** to load it.

If you've used Google's Chrome browser on another computer, the Chromium browser will be immediately familiar. Chromium lets you visit websites, play videos, games, and even communicate with people all over the world on forums and chat sites.

Start using Chromium by maximising its window so it fills the screen: find the three icons at the top-right of the Chromium window title bar (**O**) and click on the middle, up-arrow icon (**Q**). This is the *maximise* button. To the left of maximise is *minimise* (**P**), which will hide a window until you click on it in the taskbar at the top of the screen. The cross to the right of maximise is *close* (**R**), which does exactly what you'd expect: it closes the window.



CLOSE AND SAVE

Closing a window before you've saved any work you've done is a bad idea; while many programs will warn you to save when you click the close button, others won't.

The first time you run the Chromium web browser, the Raspberry Pi website should load automatically, as shown in **Figure 3-12**. If not (or to visit other websites), click in the address bar at the top of the Chromium window — the big white bar with a magnifying glass on the left-hand side — and type **raspberrypi.com** (or the address of the website you want to visit), then press the **ENTER** key on your keyboard. The Raspberry Pi website will load.

You can also type searches into the address bar: try searching for 'Raspberry Pi', 'Raspberry Pi OS', or 'retro gaming'.

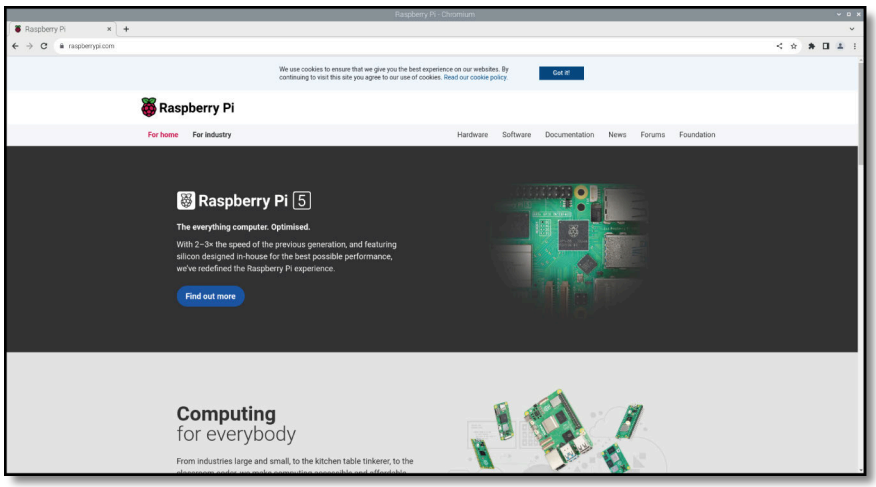


Figure 3-12 The Raspberry Pi website in Chromium

The first time you load Chromium, it may bring up several *tabs* along the top of the window. To switch to a different tab, click on it; to close a tab without closing Chromium itself, click the cross on the right-hand side of the tab you want to close.

To open a new tab, which is a handy way of having multiple websites open without having to juggle multiple Chromium windows, either click on the tab button to the right of the last tab in the list, or hold down the **CTRL** key on the keyboard and press the **T** key before letting go of **CTRL**.

When you're finished with Chromium, click the close button at the top-right of the window.

The File Manager

Files you save — for example, programs, videos, images — all go into your *home directory*. To see the home directory, click on the Raspberry Pi icon again to bring up the menu, move the mouse pointer to select **Accessories**, then click on **File Manager** to load it (**Figure 3-13**).

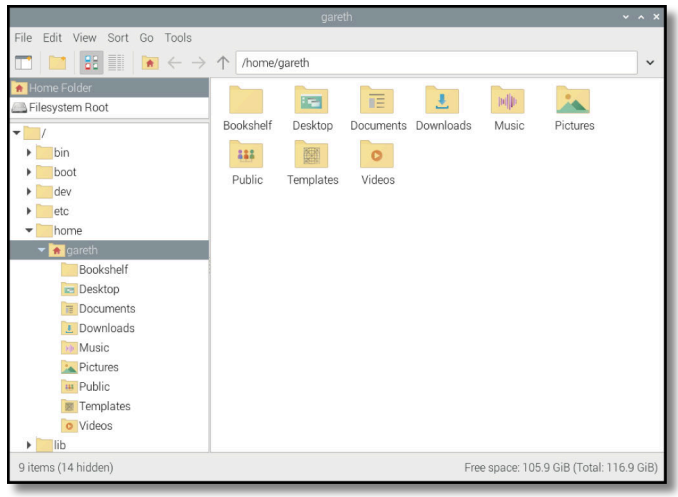


Figure 3-13 The file manager

The File Manager lets you browse the files and folders, also known as *directories*, on Raspberry Pi's microSD card, as well as those on any removable storage devices — like USB flash drives — you have connected to your Raspberry Pi's USB ports. When you first open it, it automatically goes to your home directory. In here you'll find a series of other folders, known as *subdirectories*, which — like the menu — are arranged in categories. The main subdirectories are:

- **Bookshelf** — This contains digital copies of books and magazines from Raspberry Pi Press. You can read and download books with the Bookshelf application in the Help section of the menu.
- **Desktop** — This folder is what you see when you first load Raspberry Pi OS. If you save a file in here it will appear on the desktop, making it easy to find and load.
- **Documents** — Home to most of the text files you'll create, from short stories to recipes.
- **Downloads** — When you download a file from the internet using the Chromium web browser, it will be automatically saved in Downloads.

- **Music** — Any music you create or download can be stored here.
- **Pictures** — This folder is specifically for pictures, known in technical terms as *image files*.
- **Public** — While most of your files are private, anything you put in Public will be available to other users of your Raspberry Pi, even if they have their own username and password.
- **Templates** — This folder contains any templates — blank documents with a basic layout or structure already in place — which have been installed by your applications or created by you.
- **Videos** — A folder for videos, and the first place most video-playing programs will check for content.

The File Manager window itself is split into two main panes: the left pane shows the directories on your Raspberry Pi, and the right pane shows the files and subdirectories of the directory selected in the left pane.

If you plug a removable storage device into the Raspberry Pi's USB port, a window will pop up asking if you'd like to open it in the File Manager (**Figure 3-14**). Click **OK** and you'll be able to see its files and directories.

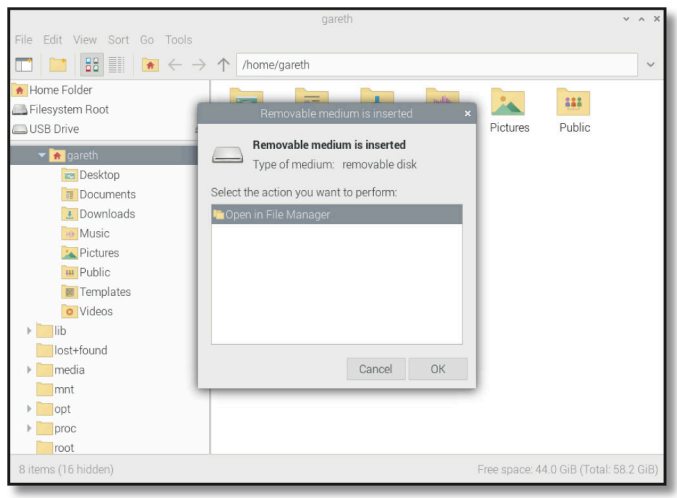


Figure 3-14 Inserting a removable storage device

You can easily *drag and drop* files between Raspberry Pi's microSD card and a removable device. With your home directory and the removable device open in separate File Manager windows, move your mouse pointer to the file you

want to copy, click and hold the left mouse button down, slide your mouse pointer to the other window, and let go of the mouse button (**Figure 3-15**).

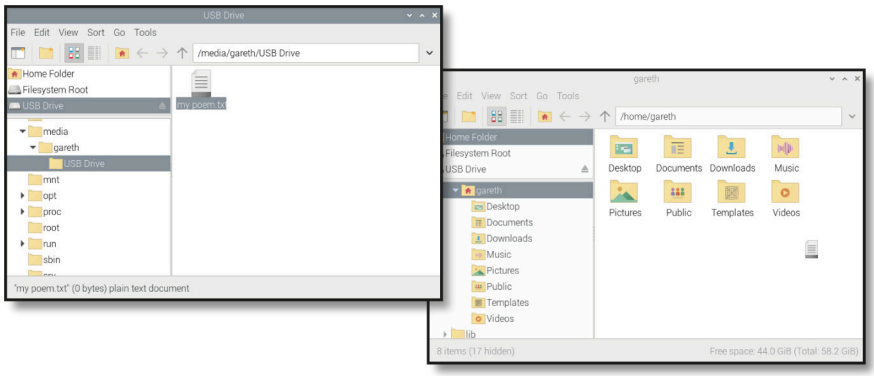


Figure 3-15 Dragging and dropping a file

An easy way to copy a file is to click once on the file, click the **Edit** menu, click **Copy**, click the other window, then click the **Edit** menu and click **Paste**.

The Cut option, also available in the Edit menu, is similar, but it deletes the file from its original home after making the copy. Both options can also be used through the keyboard shortcuts **CTRL+C** (copy) or **CTRL+X** (cut), and **CTRL+V** (paste).

KEYBOARD SHORTCUTS

When you see a keyboard shortcut like **CTRL+C**, it means to hold down the first key on the keyboard (**CTRL**), press the second key (**C**), then let go of both keys.

When you've finished experimenting, close the File Manager by clicking the close button at the very top-right of the window. If you have more than one window open, close them all. If you connected a removable storage device to your Raspberry Pi, eject it by clicking the eject button at the top-right of the screen, finding it in the list, and clicking on it before unplugging it.

EJECT DEVICES

Always use the eject button before unplugging an external storage device. If you don't, the files on it may become corrupt and unusable.

The Recommended Software tool

Raspberry Pi OS comes with a wide range of software already installed, but your Raspberry Pi is compatible with even more. A selection of the best of this software can be found in the Recommended Software tool.

Note that the Recommended Software tool needs a connection to the internet. If your Raspberry Pi is connected, click on the Raspberry Pi icon, move your mouse pointer to **Preferences**, and click on **Recommended Software**. The tool will load and start downloading information about available software.

After a few seconds, a list of compatible software packages will appear (**Figure 3-16**). These, like the software in the Raspberry Pi menu, are arranged into various categories. Click on a category in the pane on the left to see software from that category, or click **All Programs** to see everything.

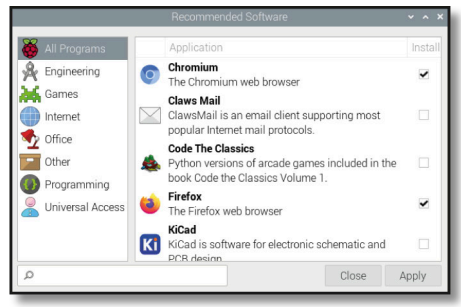


Figure 3-16 The Recommended Software tool

If a piece of software has a tick next to it, it's already installed on your Raspberry Pi. If it doesn't, you can click on the check box next to it to add a tick and mark it for installation. You can mark as many pieces of software as you like before installing them all at once, but if you're using a smaller-than-recommended microSD card you may not have room for them all.

PRE-INSTALLED APPLICATIONS

Some versions of Raspberry Pi OS come with more software installed than others. If the Recommended Software Tool says Code the Classics is already installed — if there's already a tick in the checkbox — you can choose something else from the list to install instead.

There's software available for Raspberry Pi OS to perform a wide range of tasks, including a selection of games written for the book *Code the Classics,*

Volume 1 — a walk through the history of gaming which teaches you how to write your own games in Python, available at **store.rpipress.cc**.

To install the *Code the Classics* games, click on the checkbox next to **Code the Classics** to tick it; you may need to scroll down the list of applications to see it. You'll see the text *(will be installed)* appear to the right of the application you selected, as shown in **Figure 3-17**.

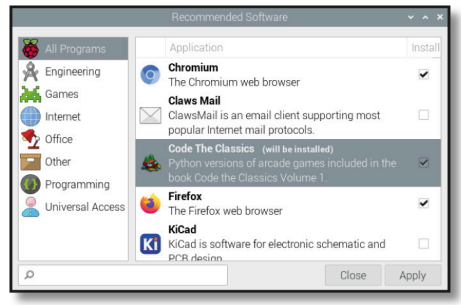


Figure 3-17 Selecting Code the Classics for installation

Click **Apply** to install the software; you'll be asked to enter your password. It will take up to a minute, depending on the speed of your internet connection, to install (**Figure 3-18**). Once the process is finished, you'll see a message telling you that installation is complete. Click **OK** to close the dialogue box, then click the **Close** button to close the Recommended Software tool.

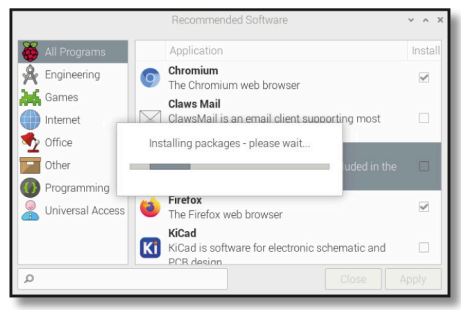


Figure 3-18 Installing Code the Classics

If you change your mind about software you've installed, you can free up space by uninstalling it. Just load the Recommended Software tool again, find the software in the list, and click the checkbox to remove the tick. When you click **Apply**, the software will be removed, but any files you've created with it and saved in your Documents folder will remain.

Another tool for installing or uninstalling software, the Add/Remove Software tool, can be found in the same Preferences category of the Raspberry Pi menu. This offers a wider selection of software beyond the list of recommended software. Learn how to use the Add/Remove Software tool in Appendix B, *Installing and uninstalling software*.

The LibreOffice productivity suite

For another taste of what Raspberry Pi can do, click on the Raspberry Pi icon, move your mouse pointer to **Office**, and click on **LibreOffice Writer**. This will load the word processor portion of LibreOffice (**Figure 3-19**), a popular open-source productivity suite.

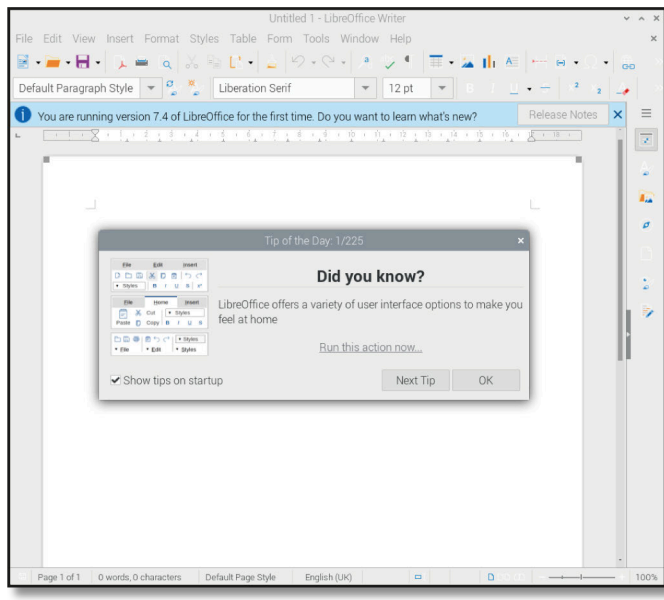


Figure 3-19 The LibreOffice Writer program

NO LIBREOFFICE?

If you don't have an **Office** category in your Raspberry Pi menu, or if you can't find LibreOffice Writer in there, it may not be installed. Go back to the Recommended Software tool and install it there before proceeding with this section.

A word processor lets you write and format documents: you can change the font style, colour, size, add effects, and even insert pictures, charts, tables, and other content. A word processor also lets you check your work for mistakes, highlighting spelling and grammar problems in red and green respectively as you type.

Begin by writing a paragraph so you can experiment with formatting. If you're feeling particularly keen, you could write about what you've learned about Raspberry Pi and its software so far. Explore the different icons at the top of the window to see what they do: see if you can make your writing bigger and change its colour. If you're not sure how to do this, simply move your mouse pointer over each icon to display a 'tool tip' telling you what that icon does. When you're satisfied, click the **File** menu and the **Save** option to save your work (**Figure 3-20**). Give it a name and click the **Save** button.

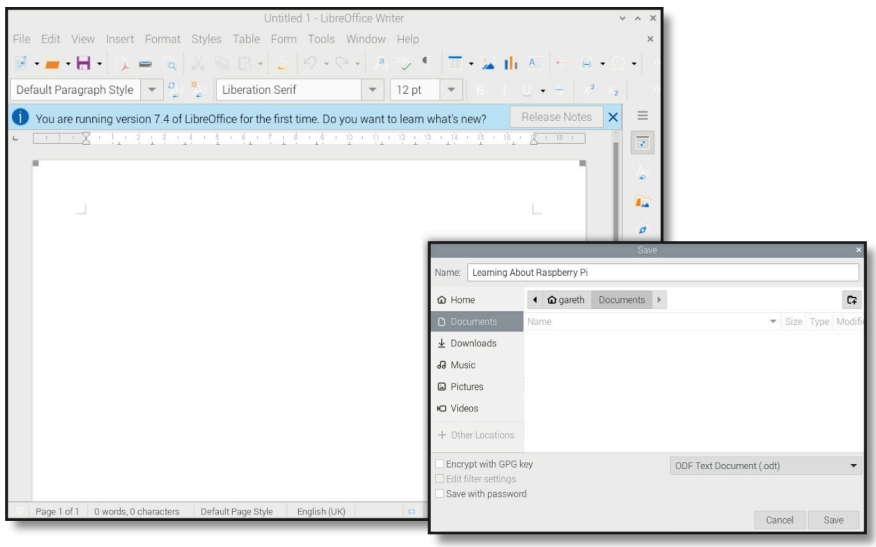


Figure 3-20 Saving a document



SAVE YOUR WORK

Get in the habit of saving your work, even if you haven't finished it yet. It will save you a lot of trouble if there's a power cut and you're interrupted part-way through!

LibreOffice Writer is only part of the overall LibreOffice productivity suite. The other parts, which you'll find in the same Office menu category as LibreOffice Writer, are:

- **LibreOffice Base** — A database: a tool for storing information, looking it up quickly, and analysing it.
- **LibreOffice Calc** — A spreadsheet: a tool for handling numbers and creating charts and graphs.
- **LibreOffice Draw** — An illustration program: a tool for creating pictures and diagrams.
- **LibreOffice Impress** — A presentation program: for creating slides and running slideshows.
- **LibreOffice Math** — A formula editor: for creating properly-formatted mathematical formulae which can be used in other documents.

LibreOffice is also available for other computers and operating systems. If you enjoy using it on your Raspberry Pi, you can download it for free from **libreoffice.org** and install it on any Microsoft Windows, Apple macOS, or Linux computer. You can close LibreOffice Writer by clicking the close button at the top-right of the window.

GETTING HELP



Most programs include a Help menu which has everything from information about what the program is to guides on how to use it. If you ever feel lost or overwhelmed by a program, look for the Help menu to reorient yourself.

Raspberry Pi Configuration tool

The last program you'll learn about in this chapter is known as the Raspberry Pi Configuration tool, and it's a lot like the Welcome Wizard you used at the start: it allows you to change various settings in Raspberry Pi OS. Click on the Raspberry Pi icon, move your mouse pointer to select the **Preferences** category, then click on **Raspberry Pi Configuration** to load it (**Figure 3-21**).

The tool is split into five tabs. The first of these is **System**: this allows you to change the password of your account, set a host name — the name your Raspberry Pi uses on your local wireless or wired network — and alter a range of other settings, including choosing a default web browser. The majority of these shouldn't need changing. Click on the **Display** tab to bring up the next

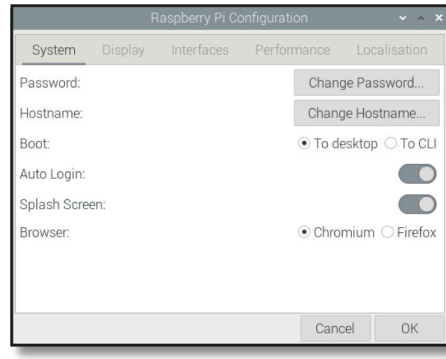


Figure 3-21 The Raspberry Pi Configuration tool

category. Here you can alter the screen display settings, if needed, to suit your TV or monitor.



MORE DETAILS

This brief overview is simply to get you used to the tool. More detailed information on each of its settings can be found in Appendix E, *Raspberry Pi Configuration Tool*.

The **Interfaces** tab offers a range of settings, all of which (except for **Serial Console** and **Serial Port**) start off disabled. These settings should only be changed if you're adding new hardware, and even then only if instructed by the hardware's manufacturer. The exceptions to this rule are **SSH**, which enables a 'Secure Shell' and lets you log into Raspberry Pi from another computer on your network using an SSH client; **VNC**, which enables a 'Virtual Network Computer' and lets you see and control the Raspberry Pi OS desktop from another computer on your network using a VNC client; and **Remote GPIO**, which lets you use Raspberry Pi's GPIO pins — about which you'll learn more in Chapter 6, *Physical computing with Scratch and Python* — from another computer on your network.

Click on the **Performance** tab to see the fourth category. Here you configure the **overlay file system**, which is a way to run your Raspberry Pi without writing changes to the microSD card. It's not something you'll need to do in most cases, so most users can just leave this section as-is.

Finally, click on the **Localisation** tab to see the last category. Here you can change your locale, which controls things like the language used in Raspberry Pi OS and how numbers are displayed; change the time zone; change the key-

board layout; and set your country for Wi-Fi purposes. For now, though, just click on **Cancel** to close the tool without making any changes.

> **WARNING!**
>
> Different countries have different rules about what frequencies a Wi-Fi radio can use. Setting the Wi-Fi country in the Raspberry Pi Configuration Tool to a different country from the one you're actually in is likely to make it struggle to connect to your networks and can even be illegal under radio licensing laws — so don't do it!

Software updates

Raspberry Pi OS receives frequent updates, which add new features or fix bugs. If Raspberry Pi is connected to a network via an Ethernet cable or Wi-Fi, it will automatically check for updates and let you know if any are ready to be installed with a small icon in the system tray (it looks like an arrow pointing down into a tray, surrounded by a circle).

If you see this icon at the top-right of your desktop, there are updates ready to install. Click the icon then click **Install Updates** to download and install them. If you'd prefer to see what the updates are first, click **Show Updates** to see a list (**Figure 3-22**).

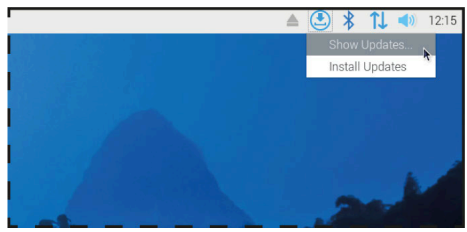


Figure 3-22 Using the software update tool

The time it takes to install updates varies depending on how many there are and how fast your internet connection is, but it should only take a few minutes. After the updates are installed, the icon will disappear from the system tray until there are more updates to install.

Some updates are designed to improve the security of Raspberry Pi OS. It's important to use the software update tool to keep your operating system up-to-date!

Shutting down

Now you've explored the Raspberry Pi OS desktop, it's time to learn a very important skill: safely shutting your Raspberry Pi down. Like any computer, Raspberry Pi keeps the files you're working on in *volatile memory* — memory which is emptied when the system is switched off. For documents you're creating, it's enough to save each in turn — which moves the file from volatile memory to *non-volatile memory* (the microSD card) — to ensure you don't lose anything.

The documents you're working on aren't the only files open, though. Raspberry Pi OS itself keeps a number of files open while it's running, and pulling the power cable from your Raspberry Pi while these are still open can result in the operating system becoming corrupt and needing to be reinstalled.

To prevent this from happening, you need to make sure you tell Raspberry Pi OS to save all its files and prepare to be powered off — a process known as *shutting down* the operating system.

Click on the Raspberry Pi icon at the top left of the desktop and then click on **Shutdown**. A window will appear with three options (**Figure 3-23**): **Shutdown**, **Reboot**, and **Logout**. **Shutdown** is the option you'll use most: clicking on this will tell Raspberry Pi OS to close all open software and files, then shut the Raspberry Pi down. Once the display has gone black, wait a few seconds until the flashing green light on your Raspberry Pi goes off, after which it's safe to turn off the power supply.

If you press the button once, you'll see the same window appear as if you'd clicked the Raspberry Pi icon followed by **Shutdown**; press the power button again when the window is visible and Raspberry Pi will shut down safely.

If you press and hold the power button for longer, it will perform a *hard shutdown* — effectively the same as if you'd just turned the power off. Only do this if your Raspberry Pi isn't responding to your instructions and you can't shut down any other way, as it runs the risk of corrupting your files or operating system.

To turn Raspberry Pi back on disconnect then reconnect the power cable, or toggle the power at the wall socket.

Reboot goes through a similar process to **Shutdown**, closing everything down, but instead of turning Raspberry Pi's power off, it restarts Raspberry Pi — as if you'd chosen **Shutdown**, then disconnected and reconnected the power cable. You'll need to use **Reboot** if you make certain changes which require a restart of the operating system — such as installing certain updates to its core software — or if some software has gone wrong, known as *crashing*, and left Raspberry Pi OS in an unusable state.

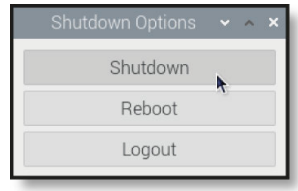


Figure 3-23
Shutting down Raspberry Pi

Logout is useful if you have more than one user account on your Raspberry Pi: it closes any programs you currently have open and takes you to a login screen on which you are prompted for a username and password. If you hit Logout by mistake and want to get back in, simply type the username and whatever password you chose in the Welcome Wizard at the start of this chapter.

WARNING!

Never remove the power cable from a Raspberry Pi or turn the power supply off at the wall without shutting down first. Doing so is likely to corrupt the operating system, and you could also lose any files you have created or downloaded.



Chapter 4

Programming with Scratch 3

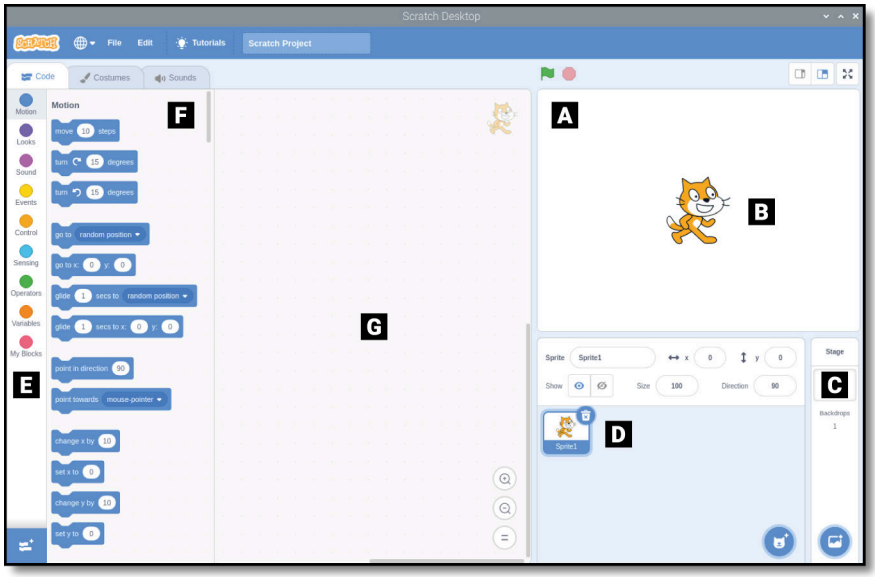
Learn how to start coding using Scratch, a block-based programming language.

Using Raspberry Pi isn't just about using software other people have created: it's also about creating your own software, based on almost anything your imagination can conjure. Whether or not you have previous experience with creating your own programs — a process known as *programming* or *coding* — you'll find Raspberry Pi is a great platform for creation and experimentation.

Key to the accessibility of coding on Raspberry Pi is Scratch, a visual programming language developed by the Massachusetts Institute of Technology (MIT). Whereas traditional programming languages require text-based instructions for the computer to carry out, in much the same way as you might write a recipe for baking a cake, Scratch has you build your program step-by-step using blocks — pre-written chunks of code disguised as colour-coded jigsaw pieces.

Scratch is a great first language for budding coders of any age, but don't be fooled by its friendly appearance: it's still a powerful and fully functional programming environment which you can use to create everything from simple games and animations through to complex interactive robotics projects.

Introducing the Scratch 3 interface



A **Stage area**

B **Sprite**

C **Stage controls**

D **Sprites list**

E **Blocks palette**

F **Blocks**

G **Code area**

Like actors in a play, your characters move around the stage (**A**) under the control of your Scratch program. These characters are known as sprites (**B**). To change the stage, such as to add your own background, use the stage controls (**C**). All the sprites you have created or loaded are in the Sprites list (**D**).

All the blocks available for your program appear in the blocks palette (**E**), which features colour-coded categories. Blocks (**F**) are pre-written chunks of code. You build your program in the code area (**G**) by dragging and dropping blocks from the blocks palette to form scripts.



SCRATCH VERSIONS

There are two versions of Scratch available for Raspberry Pi OS: Scratch and Scratch 3. This book is written with Scratch 3 in mind, which is only compatible with Raspberry Pi 4, Raspberry Pi 5, and Raspberry Pi 400.

INSTALLING SCRATCH

If you can't find Scratch 3 in the **Programming** menu, it may not be installed in your version of Raspberry Pi OS. Turn to "The Recommended Software tool" on page 43 and use the instructions there to install Scratch 3 from the **Programming** category, then come back here once it's installed.

Your first Scratch program: Hello, World!

Scratch 3 loads like any other program on Raspberry Pi: click on the Raspberry Pi icon to load the Raspberry Pi menu, move the cursor to the **Programming** section, and click on Scratch 3. After a few seconds, the Scratch 3 user interface will display. You may see a message about data collection: you can click **Yes** if you're happy to submit usage data to the Scratch Team, otherwise click **No**. Scratch will finish loading once you've made your choice.

Most programming languages need you to tell the computer what to do through written instructions, but Scratch is different. Start by clicking on the **Looks** category in the blocks palette, found at the left of the Scratch window. This brings up the purple blocks under that category. Find the say Hello! block, click and hold the left mouse button on it, and drag it over to the code area at the centre of the Scratch window before letting go of the mouse button (**Figure 4-1**).

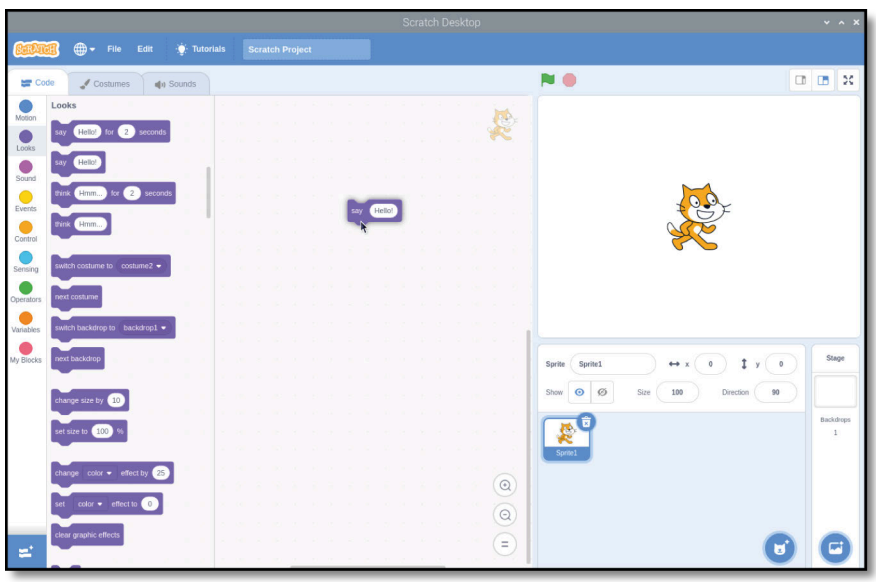


Figure 4-1 Drag and drop the block into the code area

Look at the shape of the block you've just dropped: it has a hole at the top and a matching part sticking out at the bottom. Like a jigsaw piece, this shows you that the block is expecting to have something above it and something below it. For this program, that something above is a *trigger*.

Click on the **Events** category of the blocks palette, coloured gold, then click and drag the when clicked block — known as a *hat* block — onto the code area. Position it so that the bit sticking out of the bottom connects into the hole at the top of your say Hello! block until you see a white outline, then let go of the mouse button. You don't have to be precise; if it's close enough, the block will snap into place. If it doesn't, click and hold on it again to adjust its position until it does.



Your program is now complete. To make it work, known as *running* the program, click the green flag icon at the top-left of the stage area. If all has gone well, the cat sprite on the stage will greet you with a cheery 'Hello!' (**Figure 4-2**) — your first program is a success!

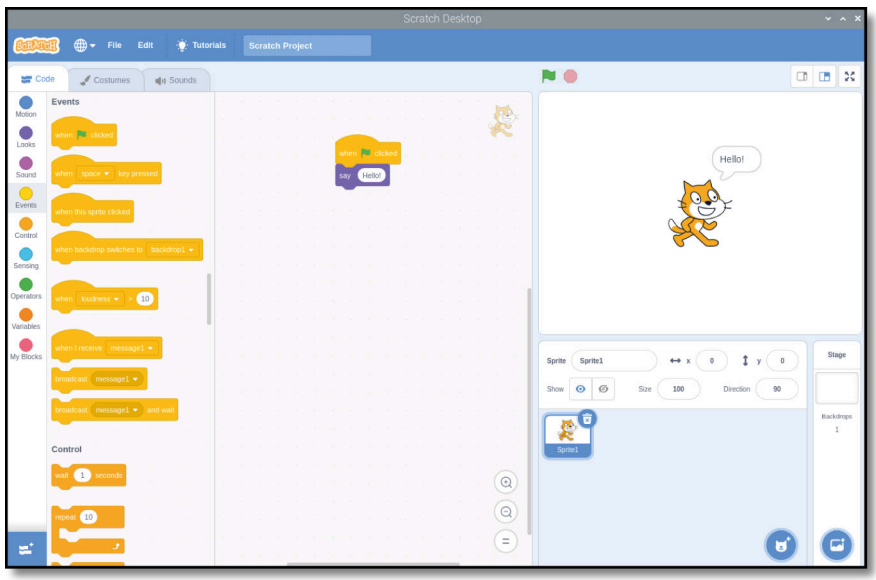


Figure 4-2 Click the green flag above the stage and the cat will say 'Hello'

Before moving on, name and save your program. Click on the **File** menu, then **Save to your computer**. Type in a name and click the **Save** button (**Figure 4-3**).

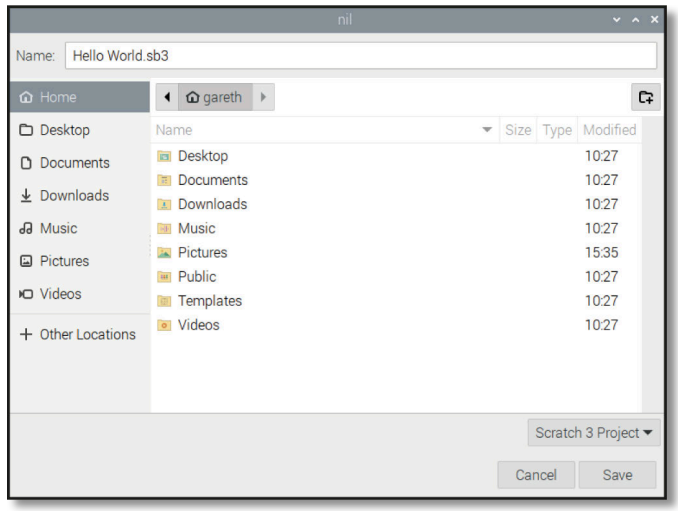


Figure 4-3 Save your program with a memorable name

WHAT CAN IT SAY?

Some blocks in Scratch can be modified. Try clicking on the word '`Hello!`', then type something else and click the green flag again. What happens on the stage?

Next steps: sequencing

Your program has two blocks, but it only has one real instruction: to say '`Hello!`' every time the program runs. To do more, you need to know about *sequencing*. Computer programs, at their simplest, are a list of instructions, just like a recipe. Each instruction follows on from the last in a logical progression known as a *linear sequence*.

Start by clicking and dragging the say Hello! block from the code area back to the blocks palette (**Figure 4-4**). This deletes the block, removing it from your program and leaving just the **Trigger** block, when clicked.

Click on the **Motion** category in the blocks palette, then click and drag the move 10 steps block so it locks into place under the trigger block on the code area.

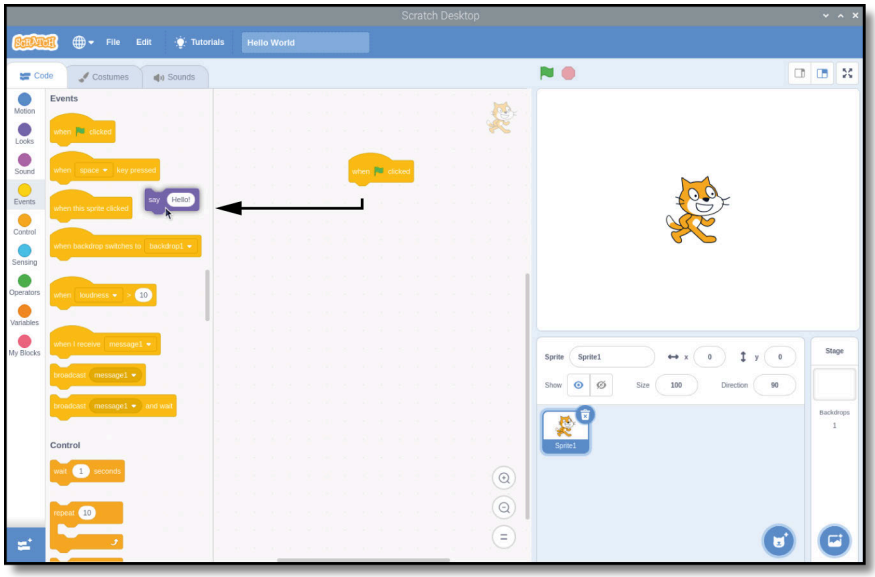
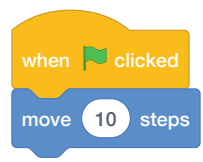
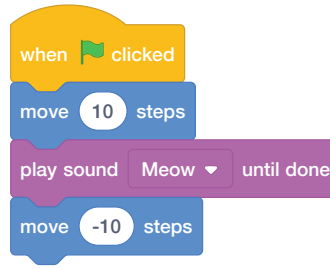


Figure 4-4 To delete a block, simply drag it out of the code area

As the name suggests, this tells your sprite — the cat — to move a set number of steps in the direction it's currently facing.



Next, add more instructions to your program to create a *sequence*: click on the **Sound** category, colour-coded pink, then click and drag the play sound Meow until done block so it locks underneath the move 10 steps block. Now keep the sequence going: click on the **Motion** category again and drag another move 10 steps block underneath your **Sound** block, but this time change **`10`** to **`-10`** to create a move -10 steps block.



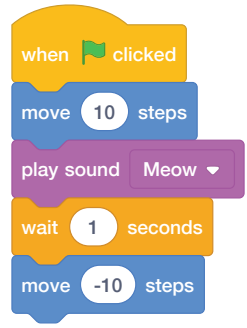
Click the green flag above the stage to run the program. You'll see the cat move to the right, make a 'meow' sound — make sure you've got speakers or headphones connected to hear it — then move back to the start again. Every time you click the green flag the cat will repeat these actions.

Congratulations! You've created a sequence of instructions, which Scratch is running through one at a time, top to bottom. While Scratch will only run one instruction at a time from the sequence, it does so very quickly: try deleting the play sound Meow until done block by clicking and dragging the bottom move -10 steps block to detach it, dragging the play sound Meow until done block to the blocks palette, then replacing it with the simpler play sound Meow block before dragging your move -10 steps block back onto the bottom of your program.



Click the green flag to run your program again. Only this time the cat sprite doesn't seem to move. The sprite *is* moving, but it moves back again so quickly that it appears to be standing still. This is because using the play sound Meow block doesn't mean that the program will wait for the sound to finish playing before the next step. Because Raspberry Pi 'thinks' so quickly, the next instruction runs before you can see the cat sprite move.

There's another way to fix this, beyond using the play sound Meow until done block: click on the light orange **Control** category of the blocks palette, then click and drag a wait 1 seconds block between the play sound Meow block and the bottom move -10 steps block.



Click the green flag to run your program one last time, and you'll see that the cat sprite waits for a second after moving to the right before moving back again. This is known as a *delay*, and is key to controlling how long your sequence of instructions takes to run.

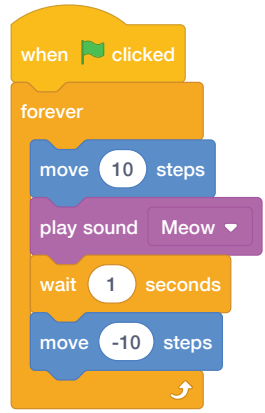
CHALLENGE: ADD MORE STEPS

Try adding more steps to your sequence and changing the values in the existing steps. What happens when the number of steps in one move block doesn't match the number of steps in another? What happens if you try to play a sound while another sound is still playing?

Looping the loop

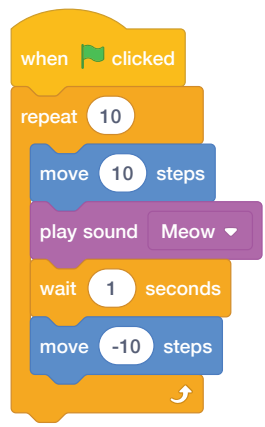
The sequence you've created so far runs only once. You click the green flag, the cat sprite moves and meows, and then the program stops until you click the green flag again. It doesn't have to stop, though, because Scratch includes a type of **Control** block known as a *loop*.

Click on the **Control** category in the blocks palette and find the forever block. Click and drag this into the code area, then drop it underneath the when clicked block and above the first move 10 steps block.



The C-shaped **Forever** block automatically grows to surround the other blocks in your sequence. Click the green flag now and you'll quickly see what the forever block does: instead of your program running once and finishing, it will run over and over again — quite literally forever. In programming, this is known as an *infinite loop* — a loop that never ends.

If the sound of constant meowing is getting to be a little much, click the red octagon next to the green flag above the stage area to stop your program. To change the loop type, pull the first move 10 steps block, and the blocks beneath it, out of the forever block, then drop them underneath the when clicked block. Click and drag the forever block to the blocks palette to delete it, then click and drag the repeat 10 block under the when clicked block so it goes around the other blocks.



Click the green flag to run your new program. At first, it seems to be doing the same thing as your original version: repeating your sequence of instructions over and over again. This time, though, rather than continuing forever, the loop will finish after ten repetitions. This is known as a *definite loop* —

you define when it will finish. Loops are powerful tools, and most programs, especially games and sensing programs, make heavy use of both infinite and definite loops.



WHAT HAPPENS NOW?

What happens if you change the number in the loop block to make it larger? What happens if it's smaller? What happens if you put the number 0 in the loop block?

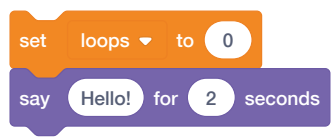
Variables and conditionals

The final concepts you'll need to understand before beginning to code Scratch programs in earnest are closely related: *variables* and *conditionals*. A variable is, as the name suggests, a value which can vary — in other words, change — over time and under control of the program. A variable has two main properties: its name, and the value it stores. That value doesn't have to be a number, either. It can be numbers, text, true-or-false (also known as *boolean values*), or completely empty — known as a *null value*.

Variables are powerful tools. Think of the things you have to track in a game: the health of a character, the speed of moving object, the level being played, and the score. All of these are tracked as variables.

First, click the **File** menu and save your existing program by clicking **Save to your computer**. If you saved the program earlier, you'll be asked if you want to overwrite it, replacing the old saved copy with your new, up-to-date version. Next, click **File** and then **New** to start a new, blank project (click **OK** when asked if you want to replace the contents of the current project). Click on the dark orange **Variables** category in the blocks palette, then the **Make a Variable** button. Type **loops** as the variable name (**Figure 4-5**), then click **OK** to make a series of blocks appear in the blocks palette.

Click and drag the set loops to 0 block to the code area. This tells your program to *initialise* the variable with a value of 0. Next, click on the **Looks** category of the blocks palette and drag the say Hello! for 2 seconds block under your set loops to 0 block.



As you found earlier, the say Hello! blocks cause the cat sprite to say whatever is written in them. Rather than writing the message in the block yourself,

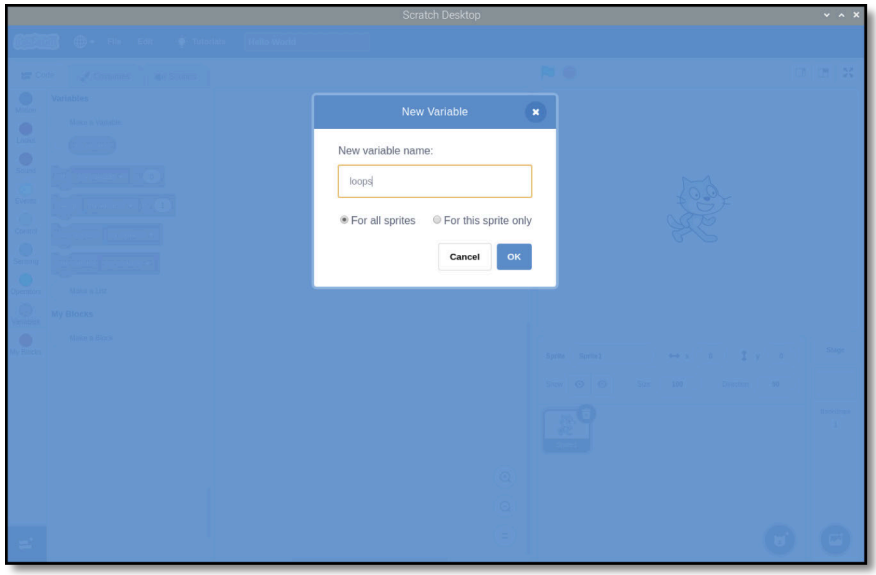
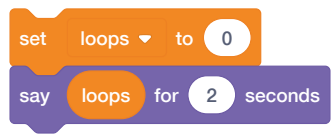


Figure 4-5 Give your new variable a name

though, you can use a variable instead. Click back onto the **Variables** category in the blocks palette, then click and drag the rounded loops block — known as a *reporter block*, found at the top of the list with a tick-box next to it — over the word **Hello!** in your say Hello! for 2 seconds block. This creates a new, combined block: say loops for 2 seconds.



Click on the **Events** category in the blocks palette, then click and drag the when clicked block to place it on top of your sequence of blocks. Click the green flag above the stage area, and you'll see the cat sprite say '**0**'. (**Figure 4-6**) — the value you gave to the variable **loops**.

Variables aren't unchanging, though. Click on the **Variables** category in the blocks palette, then click and drag the change loops by 1 block to the bottom of your sequence.

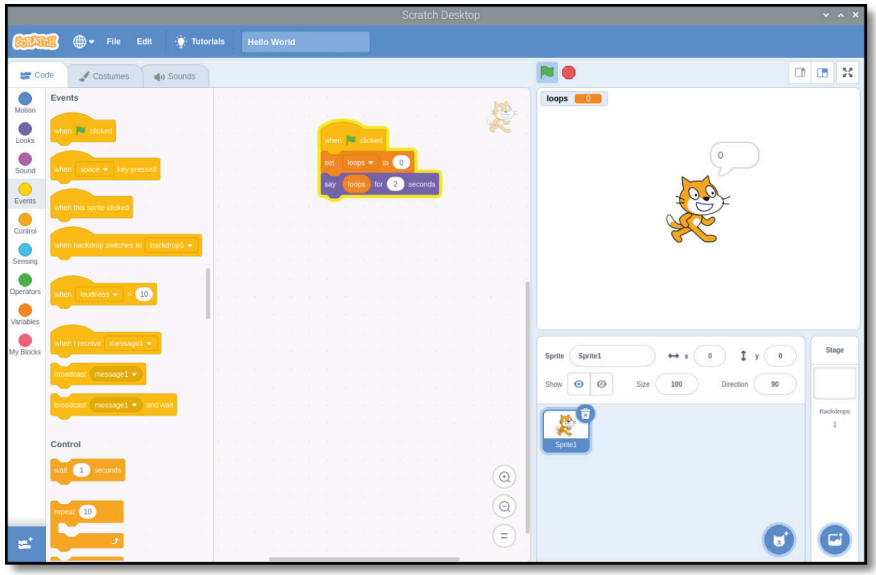
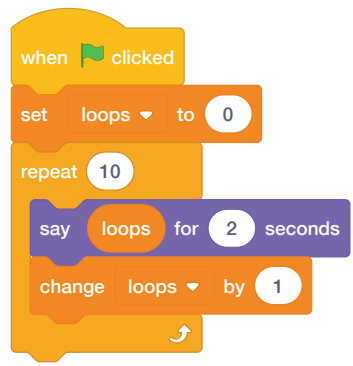


Figure 4-6 This time the cat will say the value of the variable

Next, click on the **Control** category, then click and drag a repeat 10 block and drop it so that it starts directly beneath your set loops to 0 block and wraps around the remaining blocks in your sequence.



Click the green flag again. This time, you'll see the cat count upwards from 0 to 9. This works because your program is now changing, or *modifying*, the variable itself: every time the loop runs, the program adds one to the value in the **loops** variable (**Figure 4-7**).

COUNTING FROM ZERO

Although the loop you've created runs ten times, the cat sprite only counts up to nine. This is because we're starting with a value of zero for our variable. Including zero and nine, there are ten numbers between zero and nine — so the program stops before the cat ever says '10'. To change this, you could set the variable's initial value to 1 instead of 0.

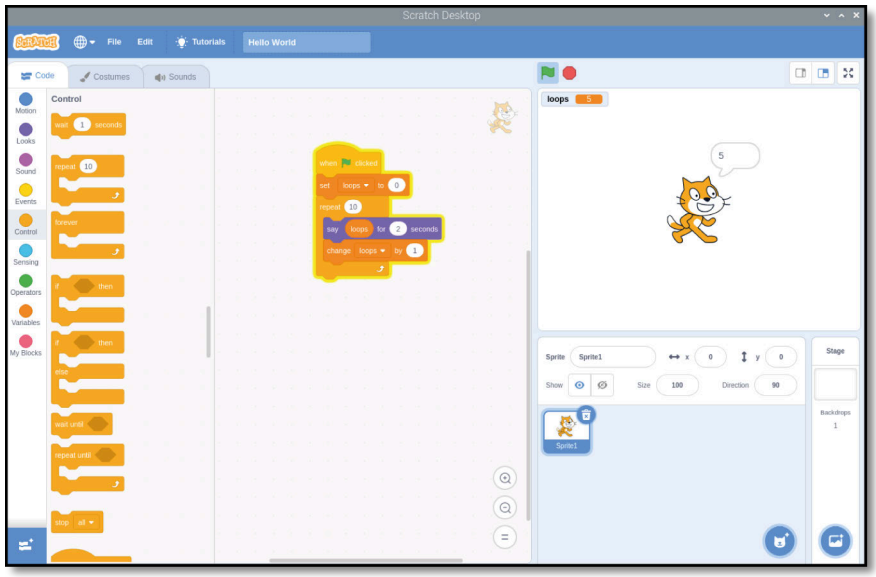
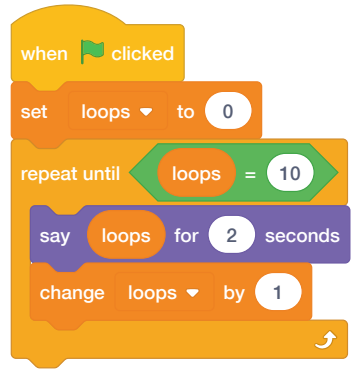


Figure 4-7 Thanks to the loop, the cat now counts upwards

You can do more with a variable than modify it. Click and drag the say loops for 2 seconds block to break it out of the repeat 10 block and drop it below the repeat 10 block. Click and drag the repeat 10 block to the blocks palette to delete it, then replace it with a repeat until block, making sure the block is connected to the bottom of the set loops to 0 block. It should surround both of the other blocks in your sequence. Next, click the **Operators** category in the blocks palette, colour-coded green, then click and drag the diamond-shaped = block and drop it on the matching diamond-shaped hole in the repeat until block.

This **Operators** block lets you compare two values, including variables. Click on the **Variables** category, drag the loops reporter block into the empty space in the = **Operators** block, then click on the space with `50` in it and type the number `10`.



Click on the green flag above the stage area and you'll find the program works the same way as before: the cat sprite counts from 0 up to 9 (**Figure 4-8**) and then the program stops. This is because the repeat until block is working in exactly the same way as the repeat 10 block, but rather than counting the number of loops itself, it's comparing the value of the **loops** variable to the value you typed to the right of the block. When the **loops** variable reaches 10, the program stops.

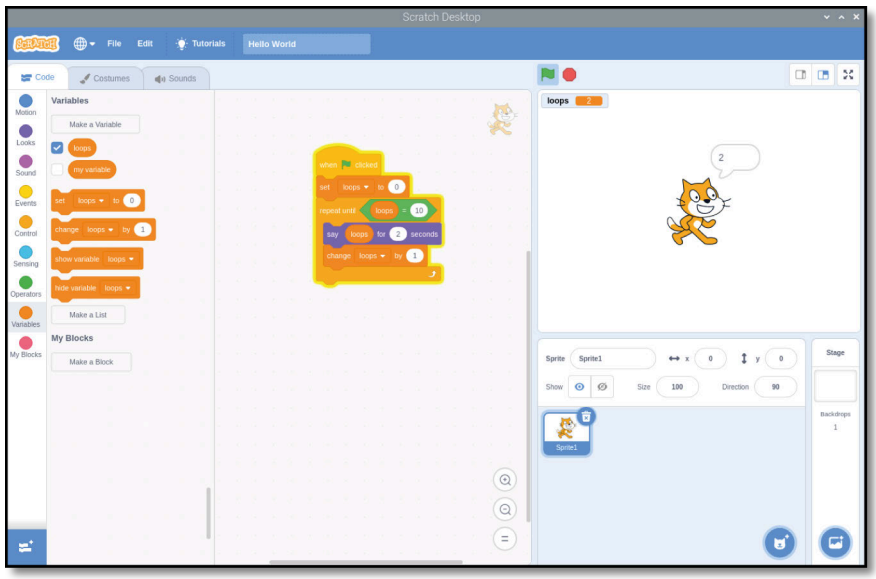


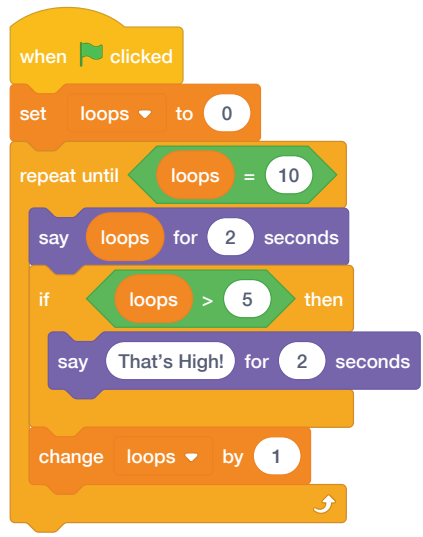
Figure 4-8 Using a 'repeat until' block with a comparative operator

This is known as a *comparative operator*: it literally compares two values. Click on the **Operators** category of the blocks palette and find the two other diamond-shaped blocks above and below the one with the = symbol. These are also comparative operators: < compares two values and is triggered when

the value on the left is smaller than the one on the right, and `>` triggers when the value on the left is larger than the one on the right.

Click on the **Control** category of the blocks palette, find the if then block, then click and drag it to the code area before dropping it directly beneath the say loops for 2 seconds block. It will automatically surround the change loops by 1 block, so click and drag on that block to move it so it connects to the bottom of your if then block instead. Click on the **Looks** category of the blocks palette, then click and drag a say Hello! for 2 seconds block and drop it inside your if then block. Click on the **Operators** category of the blocks palette, then click and drag the < > block into the diamond-shaped hole in your if then block.

The if then block is a **Conditional** block, which means the blocks inside it will only run if a certain condition is met. Click on the **Variables** category of the blocks palette, drag and drop the loops reporter block into the empty space in your < > block, then click on the space with `50` in it and type the number `5`. Finally, click on the word `Hello!` in your say Hello! for 2 seconds block and type `That's high!`.



Click the green flag to run your program. At first, it will work as before with the cat sprite counting upwards from zero. When the number reaches six, the first number greater than five, the if then block will begin to trigger and the cat sprite will comment on how high the numbers are getting (**Figure 4-9**). Congratulations: you can now work with variables and conditionals!

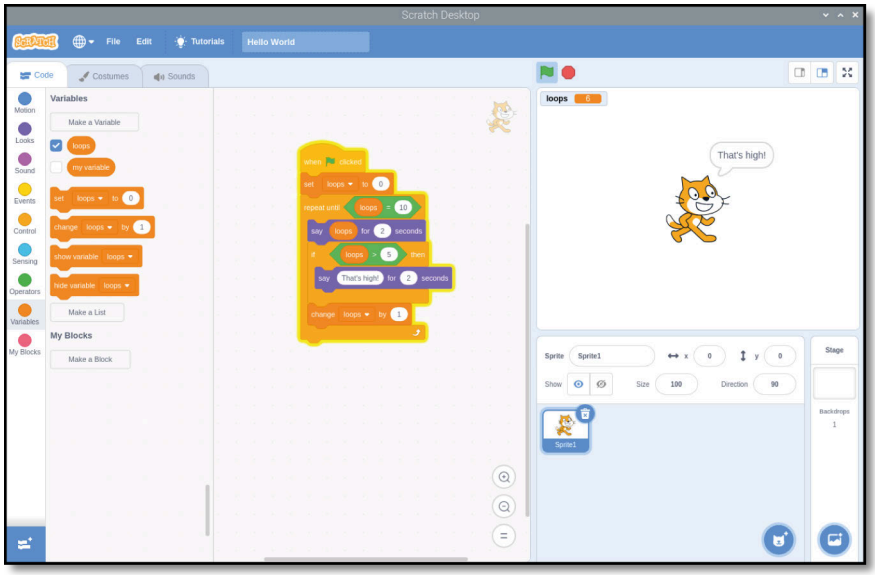


Figure 4-9 The cat makes a comment when the number reaches six

CHALLENGE: HIGH AND LOW

How could you change the program so the cat sprite comments on how low the numbers below five are instead? Can you change it so that the cat will comment on both high and low numbers? Experiment with the if then else block to make this easier!

Project 1: Astronaut reaction timer

Now that you understand how Scratch works, it's time to make something a little more interactive: a reaction timer, designed to honour British ESA astronaut Tim Peake and his time aboard the International Space Station.

Save your existing program, if you want to keep it, then open a new project by clicking on **File** and **New**. Before you begin, give it a name by clicking on **File** and **Save to your computer**: call it 'Astronaut reaction timer'.

This project relies on two images — one as a stage background, one as a sprite — which are not included in Scratch's built-in resources. To download them, click on the Raspberry Pi icon to load the Raspberry Pi menu, move the mouse pointer to **Internet**, and click on **Chromium Web Browser**. When the browser has loaded, type **rptl.io/astro-bg** into the address bar, then press the **ENTER**

key. Right-click on the picture of space and click on **Save image as**, then click on the **Save** button (**Figure 4-10**). Click back into the address bar, and type **rptl.io/astro-sprite** followed by the **ENTER** key.

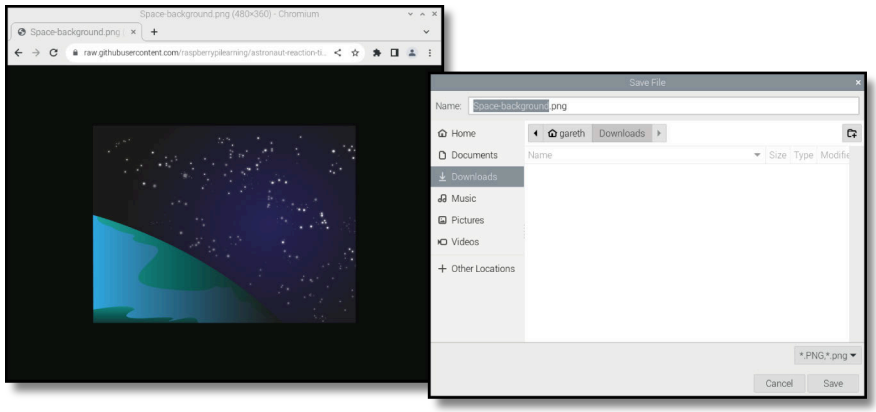


Figure 4-10 Save the background image

Again, right-click on the picture of Tim Peake and click on **Save image as**, then choose the **Downloads** folder and click on the **Save** button. With those two images saved, you can close Chromium or leave it open and use the taskbar to switch back to Scratch 3.

USER INTERFACE

If you've been following this chapter from the start, you should be familiar with the Scratch 3 user interface. The following project instructions will rely on you knowing where things are; if you forget where to find something, look back at the picture of the user interface at the start of this chapter for a reminder.

Right-click the cat sprite in the list and click **delete**. Hover the mouse pointer over the **Choose a Backdrop** icon . Next, click the **Upload Backdrop** icon from the list that appears.

Find the **Space-background.png** file in the **Downloads** folder, click on it to select it, then click **OK**. The plain white stage background will change to the picture of space, and the code area will be replaced by the backdrops area (**Figure 4-11**). Here you can draw over the backdrop, but for now just click on the tab marked **Code** at the top of the Scratch 3 window.

Upload your new sprite by hovering your mouse pointer over the **Choose a Sprite** icon . Next, click the **Upload Sprite** icon at the top of the list that appears. Find the file **Astronaut-Tim.png** in the Downloads folder, click to

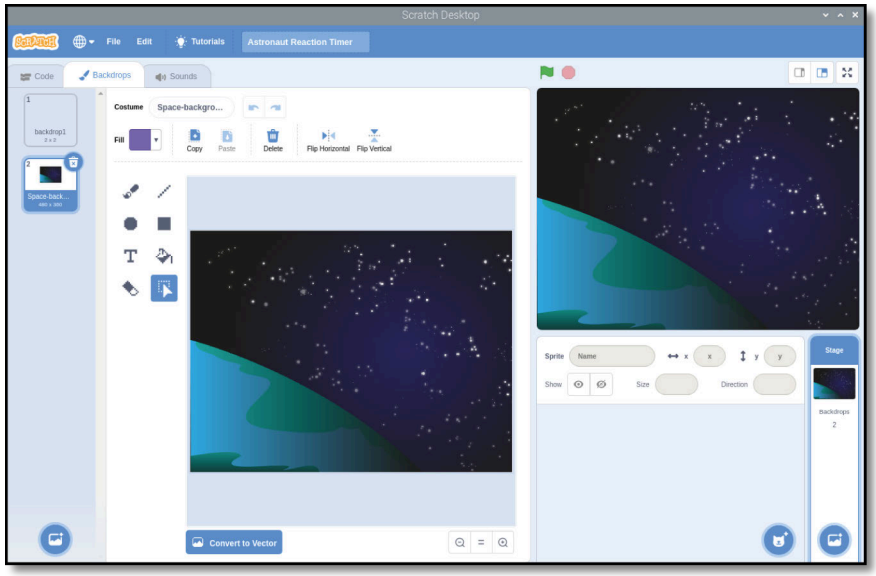
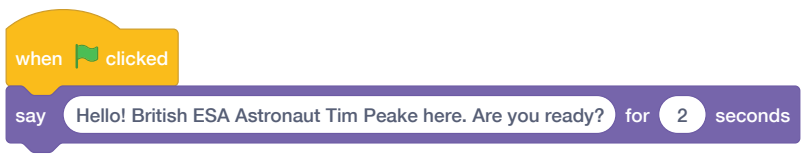


Figure 4-11 The space background appears on the stage

select it, then click **OK**. The sprite appears on the stage automatically, but it might not be in the middle of the stage: click and drag it with the mouse and drop it so it's near the lower middle (**Figure 4-12**).

With your new background and sprite in place, you're ready to create your program, so click the **Code** tab. Start by creating a new variable called `time`, making sure that **For all sprites** is selected before clicking **OK**. Click on your sprite — either on the stage or in the sprite pane — to select it, then add a when clicked block from the **Events** category to the code area. Next, add a say Hello! for 2 seconds block from the **Looks** category, then click on it to change it to say `Hello! British ESA Astronaut Tim Peake here. Are you ready?`



Add a wait 1 seconds block from the **Control** category, then a say Hello! block. Change this block to say '`Hit Space!`', then add a reset timer block from the **Sensing** category. This controls a special variable built into

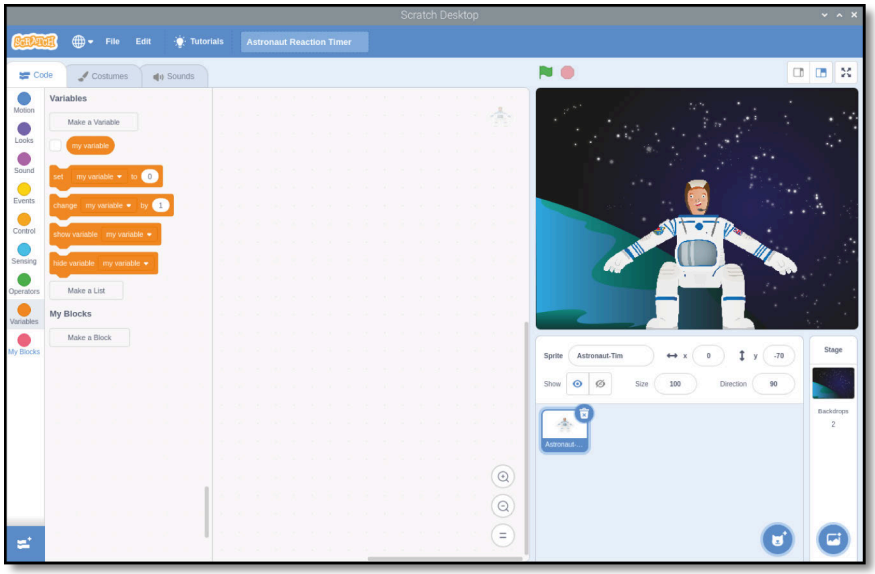
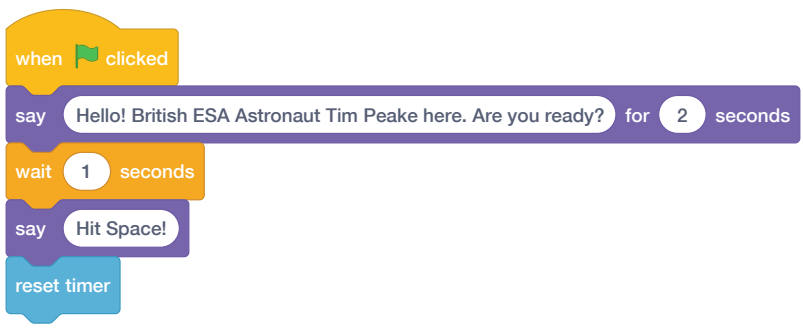
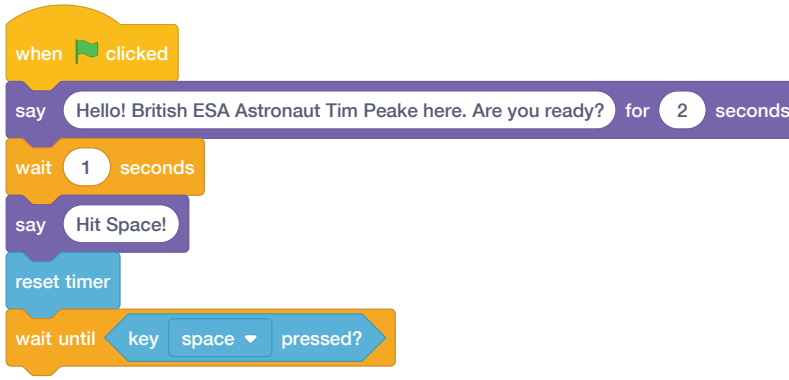


Figure 4-12 Drag the astronaut sprite to the lower middle of the stage

Scratch for timing things and will be used to time how quickly you can react in the game.

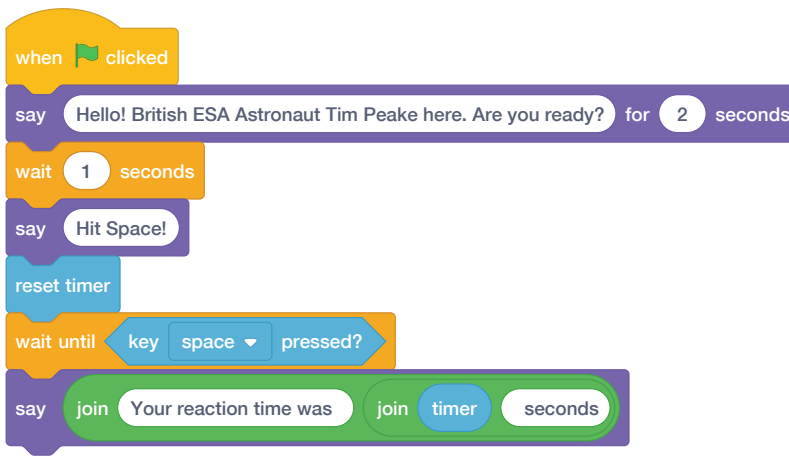


Add a wait until **Control** block, then drag a key space pressed? **Sensing** block into its white space. This will pause the program until you press the **SPACE** key on the keyboard, but the timer will continue to run — counting exactly how much time has passed between the message telling you to '`Hit Space!`' and when you actually press the **SPACE** key.



You now need Tim to tell you how long you took to press the **SPACE** key, but in a way that's easy for you to read. To do this, you'll need a join **Operators** block. This takes two values, including variables, and joins them together one after the other — known as *concatenation*.

Start with a say Hello! block, then drag and drop a join **Operators** block over the word `Hello!`. Click on `apple` and type `Your reaction time was` — make sure you have a blank space at the end — then drag another **Join** block over the top of `banana` in the second box. Drag a timer **Reporting** block from the **Sensing** category into what is now the middle box, and type `seconds`. into the last box — make sure you include a blank space at the start.



Finally, drag a set my variable to 0 **Variables** block onto the end of your sequence. Click on the drop-down arrow next to '**my variable**' and click on '**time**' from the list, then replace the `0` with a timer **Reporting** block from the **Sensing** category. Your game is now ready to test by clicking on the green flag above the stage. Get ready, and as soon as you see the message '`Hit Space!`', press the **SPACE** key as quickly as you can (**Figure 4-13**).

See if you can beat our score!

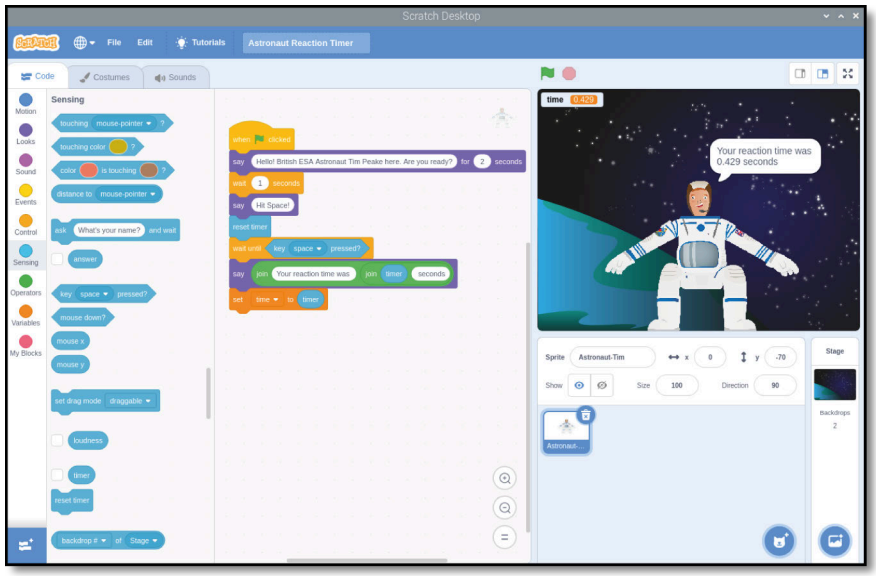
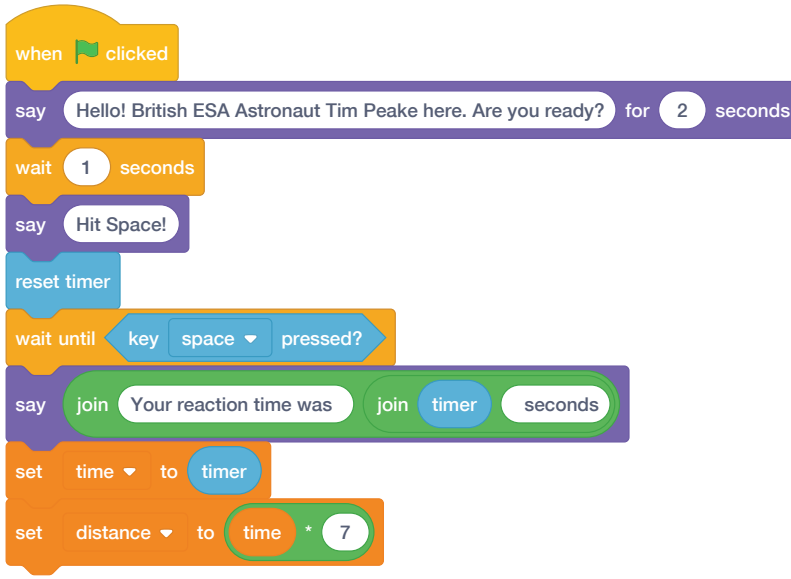


Figure 4-13 Time to play the game!

You can extend this project further by having it calculate roughly how far the International Space Station has travelled in the time it took you to press the **SPACE** key, based on the station's published speed of seven kilometres per second. First, create a new variable called `distance`. Notice how the blocks in the **Variables** category automatically change to show the new variable, but the existing time variable blocks in your program remain the same.

Add a set distance to 0 block, then drag a (*) **Operators** block — which indicates multiplication — over the `0`. Drag a time **Reporting** block over the first blank space, then type the number `7` in the second space. When you're finished, your combined block reads set distance to time * 7. This will take the time it took you to press the **SPACE** key and multiply it by seven, to get the distance in kilometres the ISS has travelled.



Add a wait 1 seconds block and change it to `4`. Next, drag another say Hello! block onto the end of your sequence and add two join blocks, just as you did before. In the first space, over `apple`, type `In that time the ISS travels around `, remembering a space at the end; in the `banana` space, type `kilometres.`, again remembering a space at the start.

Finally, drag a round **Operators** block into the middle `apple` space, then drag a distance **Reporting** block into the new blank space it creates. The round block rounds numbers up or down to their nearest whole number, so instead of a hyper-accurate but hard-to-read number of kilometres you'll get an easy-to-read whole number.

Click the green flag to run your program and see how far the ISS travels in the time it takes you to hit the **SPACE** key (**Figure 4-14**). Remember to save your program when you're finished, so you can easily load it again in the future without having to start from the beginning!

CHALLENGE: CUSTOM ARTWORK

You can click on a sprite or background, then click the **Costumes** or **Backdrops** tab to bring up an editor with drawing tools. Can you draw your own characters and background and edit the code to have your character say something different?

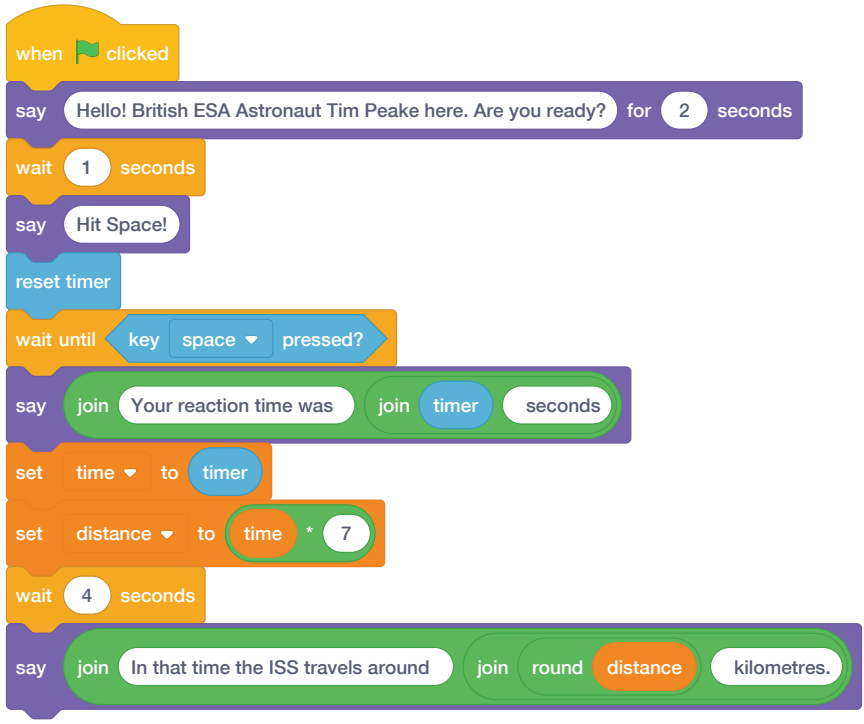




Figure 4-14 Tim tells you how far the ISS has travelled

Project 2: Synchronised swimming

Most games use more than a single button. This project demonstrates that, by offering two-button control using the left and right arrow keys on the keyboard.

Create a new project and save it as 'Synchronised swimming'. Click on the **Stage** in the stage control section, then click the **Backdrops** tab at the top left. Click the **Convert to Bitmap** button below the backdrop. Choose a water-like blue colour from the **Fill** palette and click on the **Fill** icon . Next, click on the chequered backdrop to fill it with blue (**Figure 4-15**).

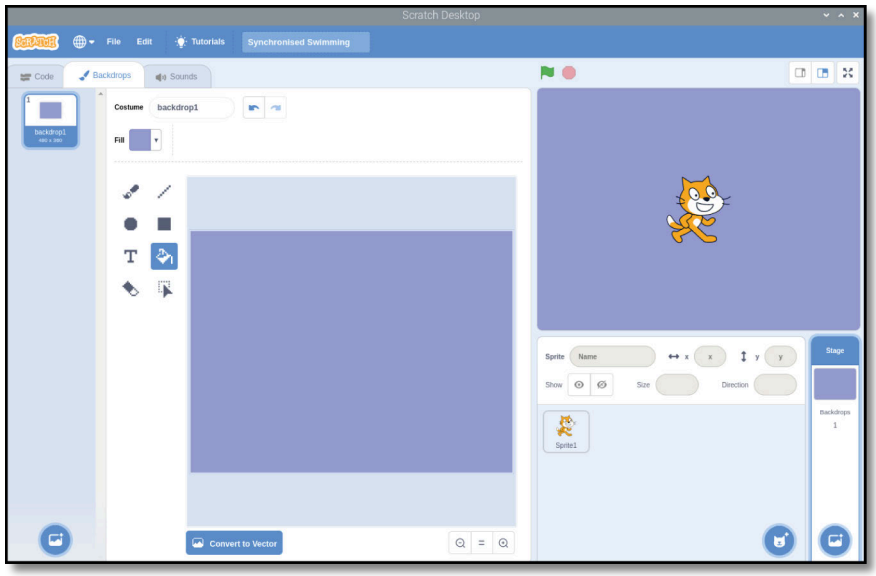


Figure 4-15 Fill the background with a blue colour

Right-click the cat sprite in the list and click **delete**. Click the **Choose a Sprite** icon to see a list of built-in sprites. Click on the **Animals** category, then '**Cat Flying**' (**Figure 4-16**), then **OK**. This sprite also works well for swimming projects.

Click the new sprite, then drag two when space key pressed **Events** blocks into the code area. Click on the small down-arrow next to the word 'space' on the first block and choose **left arrow** from the list of possible options. Drag a turn 15 degrees **Motion** block under your when left arrow key pressed block, then do the same with your second **Events** block except choosing **right arrow** from the list and using a turn 15 degrees **Motion** block.

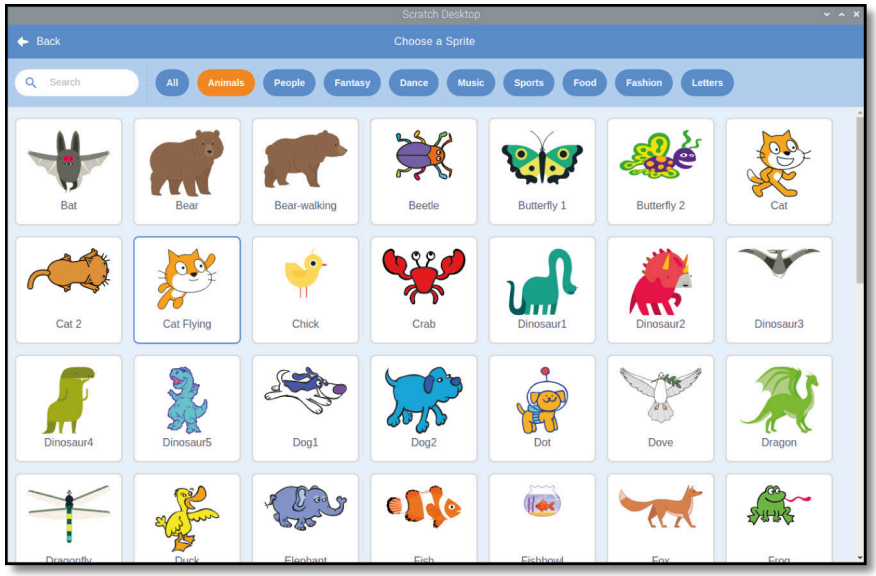
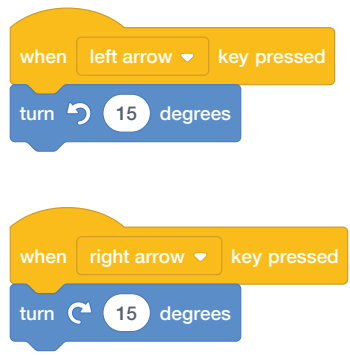


Figure 4-16 Choose a sprite from the library



Press the left or right arrow key to test your program. You'll see the cat sprite turning as you do, matching the direction you're choosing on the keyboard. Notice how you didn't need to click on the green flag this time? This is because the **Events** trigger blocks you have used are always active, even when the program isn't 'running' in the usual sense.

Do the same steps twice again, but this time choosing **up arrow** and **down arrow** for the **Events** trigger blocks, then move 10 steps and move -10 steps for the **Motion** blocks. Press the arrow keys now and you'll see your cat can turn around and swim forwards and backwards too!









To make the cat sprite's motion more realistic, you can change how it appears — known in Scratch terms as a *costume*. Click on the cat sprite, then click on the **Costumes** tab above the blocks palette. Click on the '**cat flying-a**' costume and click on the bin icon that appears at its top-right corner to delete it. Next, click on the '**cat flying-b**' costume and use the name box at the top to rename it to 'right' (**Figure 4-17**).

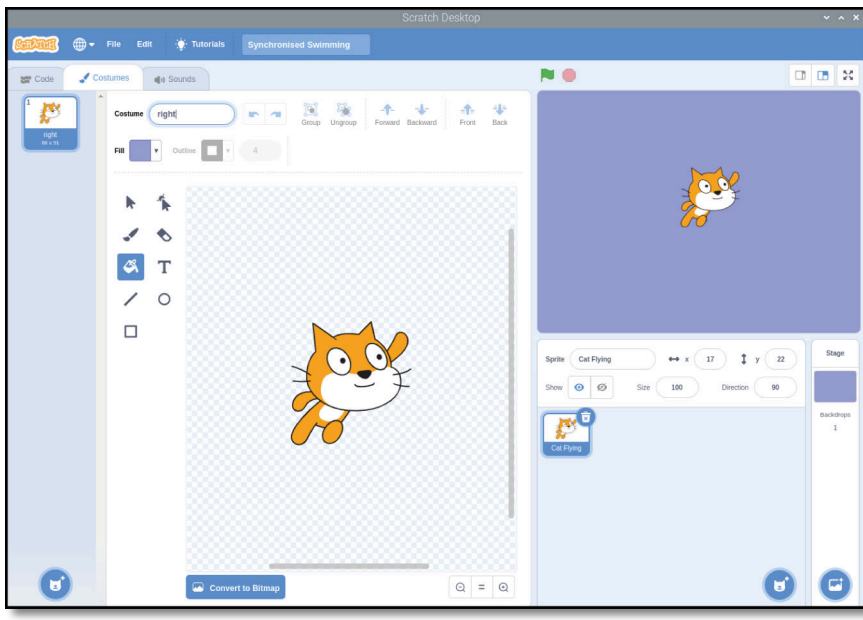


Figure 4-17 Rename the costume as 'right'

Right-click on the newly renamed 'right' costume and click **duplicate** to create a copy. Click on this copy to select it, then click the **Select** icon . Next, click the **Flip Horizontal** icon . Finally, rename the duplicate costume to 'left' (**Figure 4-18**). You'll finish with two 'costumes' for your sprite, which are

exact mirror images: one called ‘right’ with the cat facing right, and one called ‘left’ with the cat facing left.

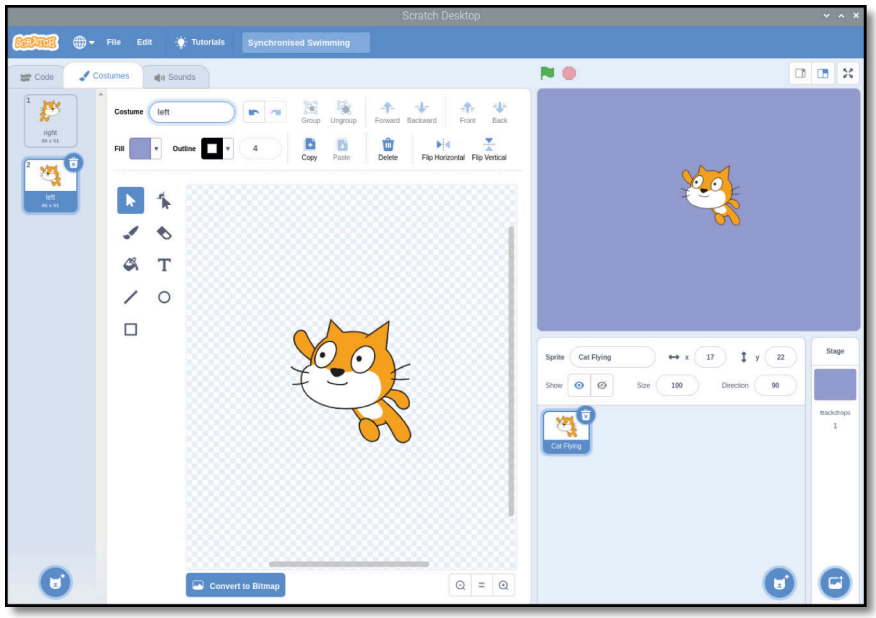
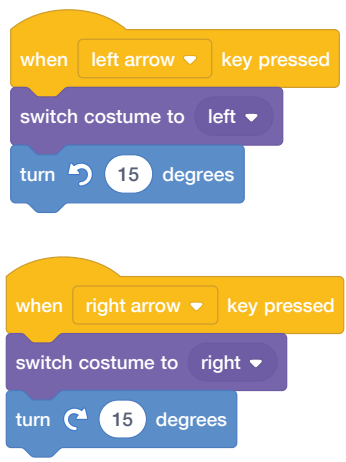
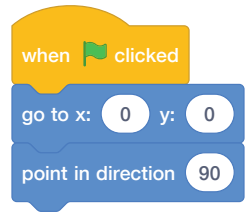


Figure 4-18 Duplicate the costume, flip it, and name it ‘left’

Click on the **Code** tab above the costume area, then drag two switch costume to left **Looks** blocks under your left arrow and right arrow **Events** blocks, changing the one under the right arrow block to read switch costume to right. Try the arrow keys again; the cat now seems to turn to face the direction it’s swimming.



For Olympic-style synchronised swimming, though, we need more swimmers, and we need a way to reset the cat sprite's position. Add a when clicked **Events** block, then underneath add a go to x: 0 y: 0 **Motion** block — changing the values if necessary — and a point in direction 90 **Motion** block. Now, when you click the green flag, the cat moves to the middle of the stage and faces to the right.



To create more swimmers, add a repeat 6 block — changing the default value of '**10**' — and add a create clone of myself **Control** block inside it. To make it so the swimmers aren't all swimming in the same direction, add a turn 60 degrees block above the create clone block but still inside the repeat 6 block. Click the green flag and try the arrow keys now to see your swimmers come to life!



To complete the Olympic feel, you'll need to add some music. Click on the **Sounds** tab above the blocks palette, then click the **Choose a Sound** icon . Click on the **Loops** category, then browse through the list (**Figure 4-19**) until you find some music you like — we went with '**Dance Around**'. Click on the **OK** button to choose the music, then click on the **Code** tab to open the code area again.

Add another when clicked **Events** block to your code area, then add a forever **Control** block. Inside this **Control** block, add a play sound dance around until done block — remembering to look for the name

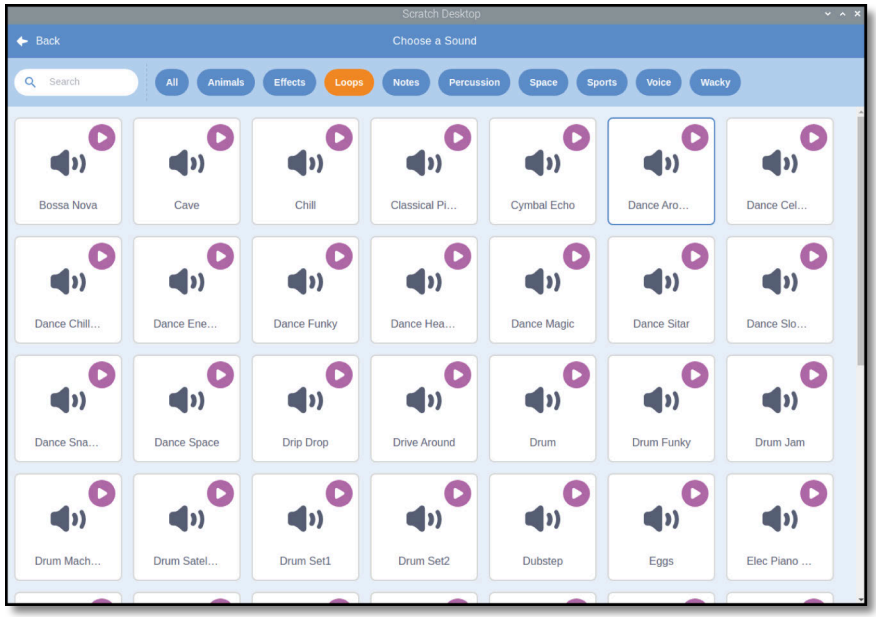
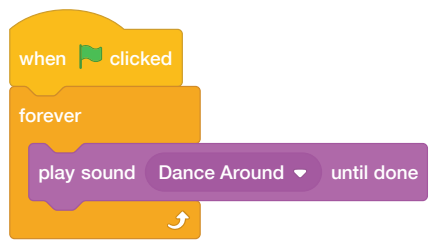


Figure 4-19 Select a music loop from the sound library

of whatever piece of music you chose — and click the green flag to test your new program. If you want to stop the music, click the stop button (the red octagon) to stop the program and silence the sound!

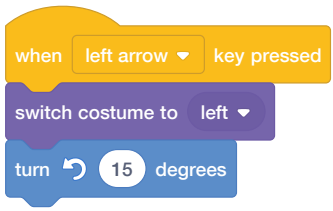


Finally, you can simulate a full dancing routine by adding a new event trigger to your program. Add a when space key pressed **Events** block, then a switch costume to right block. Underneath this, add a repeat 36 block — remember to change the default value — and inside this a turn 10 degrees block and a move 10 steps block.

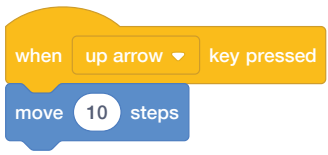
Click the green flag to start the program, then press the **SPACE** key to try out the new routine (**Figure 4-20**), and save your program when you're finished!

CHALLENGE: CUSTOM ROUTINE

Can you create your own synchronised swimming routine using loops? What would you need to change if you wanted more (or fewer) swimmers? Can you add multiple swimming routines which can be triggered using different keyboard keys?

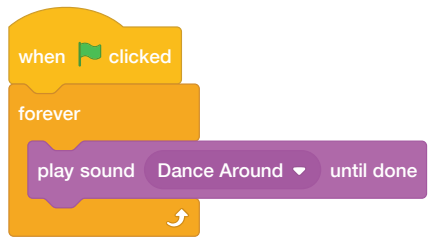














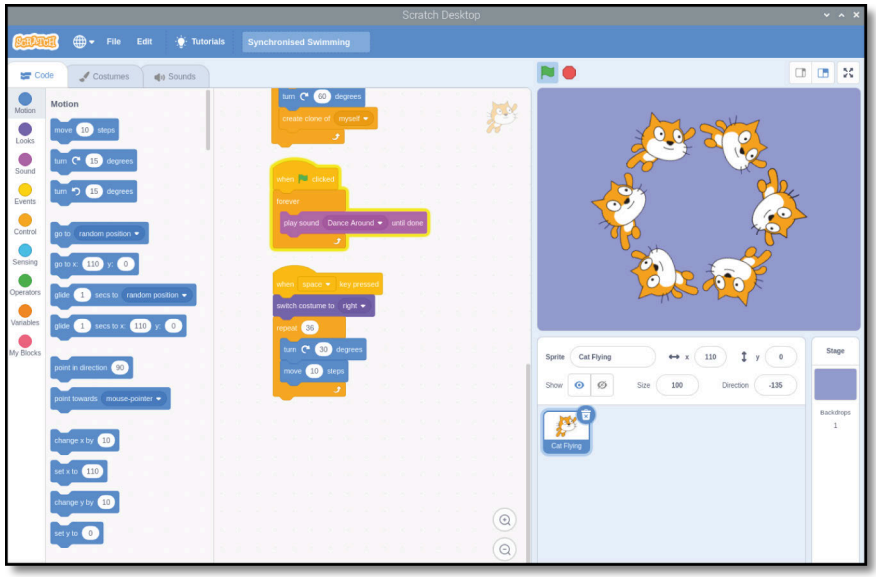


Figure 4-20 The finished synchronised swimming routine

Project 3: Archery game

Now you're getting to be a bit of an expert at Scratch, it's time to work on something a little more challenging: a game of archery, where the player must hit a target with a randomly swaying bow and arrow.

Start by opening the Chromium web browser and typing **rptl.io/archery** into the address bar, followed by the **ENTER** key. The resources for the game are contained in a zip file, so you'll need to unzip it. To do so, right-click the file and select **Extract Here**. Switch back to Scratch 3 and click on the **File** menu followed by **Load from your computer**. Click on **ArcheryResources.sb3** followed by the **Open** button. You'll be asked if you want to replace the contents of your current project. If you haven't saved your changes, click **Cancel** and save them, then click **OK**.

The project you've just loaded contains a backdrop and a sprite (**Figure 4-21**), but none of the actual code to make a game: adding that is your job. Start by adding a when clicked block, then a broadcast message1 block. Click on the down arrow at the end of the block, and then click '**New Message**' and type in '`new arrow`' before clicking the **OK** button. Your block now reads broadcast new arrow.

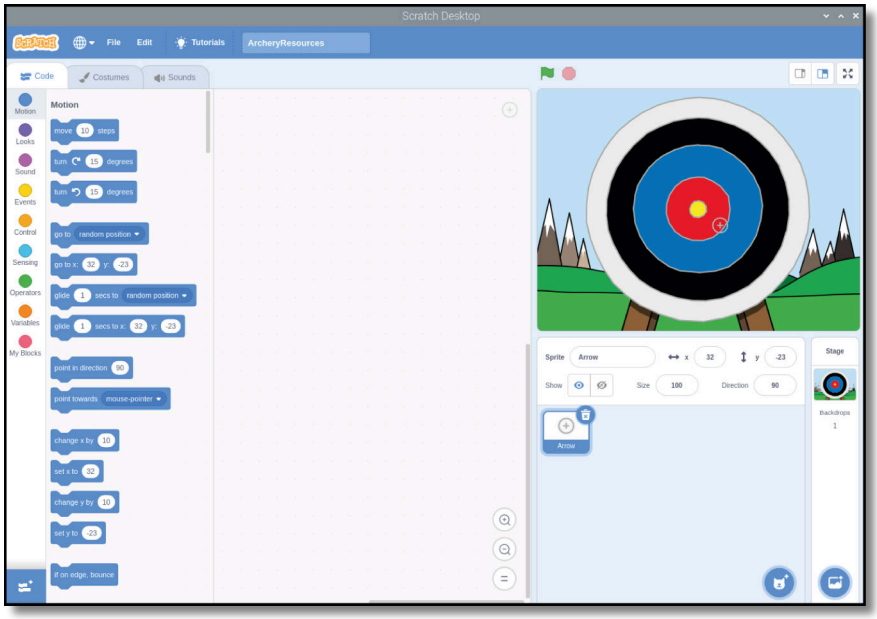
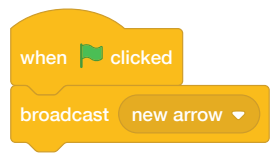
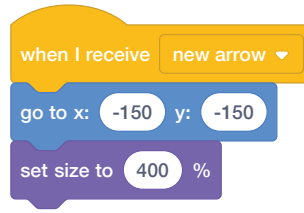


Figure 4-21 Resources project loaded for the archery game



A broadcast is a message from one part of your program which can be received by any other part of your program. To have it actually do something, add a when I receive message1 block, and again change it to read when I receive new arrow. This time you can just click on the down arrow and choose **new arrow** from the list; you don't have to create the message again.

Below your when I receive new arrow block, add a go to x: -150 y: -150 block and a set size to 400 % block. Remember that these aren't the default values for those blocks, so you'll need to change them once you've dragged them onto the code area. Click on the green flag to see what you've done so far: the arrow sprite, which the player uses to aim at the target, will jump to the bottom-left of the stage and quadruple in size.

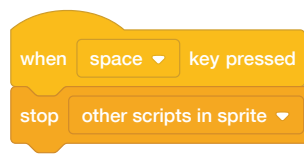


To give the player a challenge, add movement to simulate the swaying of the bow as it's drawn back and the archer aims. Drag a forever block, followed by a glide 1 seconds to x: -150 y: -150 block. Edit the first white box to say `0.5` instead of `1`, then put a pick random -150 to 150 **Operators** block in each of the other two white boxes. This means the arrow will drift around the stage in a random direction, for a random distance — making it far harder to hit the target!



Click the green flag again, and you'll see what that block does: your arrow sprite is now drifting around the stage, covering different parts of the target. At the moment, though, you have no way to shoot the arrow at the target.

Drag a when space key pressed block into your code area, followed by a stop all **Control** block. Click the down arrow at the end of the block and change it to a stop other scripts in sprite block.



If you'd stopped your program to add the new blocks, click the green flag to start it again and then press the **SPACE** key: you'll see the arrow sprite stop moving. That's a start, but you need to make it look like the arrow is flying to the target. Add a repeat 50 block followed by a change size by -10 block, then click the green flag to test your game again. This time, the arrow appears to be flying away from you and towards the target.

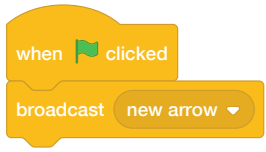


To make the game more fun, you should add a way to keep score. Still in the same stack of blocks, add an if then block — making sure it's below the repeat 50 block and not inside it — with a touching color? **Sensing** block in its diamond-shaped gap. To choose the correct colour, click on coloured box at the end of the **Sensing** block, then the **Eye Dropper** icon . Next, click on the yellow bull's-eye of your target on the stage.

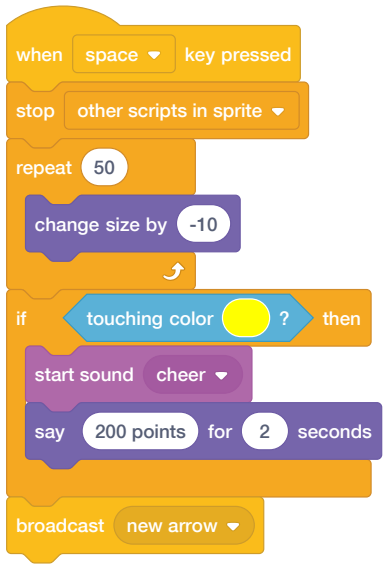
Add a start sound cheer block and a say 200 points for 2 seconds block inside the if then block so the player knows they scored. Finally, add a broadcast new arrow block to the very bottom of the block stack, below and outside the if then block, to give the player another arrow each time they fire one. Click the green flag to start your game and try to hit the yellow bull's-eye: when you do, you'll be rewarded with a cheer from the crowd and a 200-point score!

The game works at this point, but may be a little too challenging. Using what you've learnt in this chapter, try adding scores for hitting parts of the target other than the bull's-eye: 100 points for red, 50 points for blue, and so on.

For more Scratch projects to try, see Appendix D, *Further reading*.







CHALLENGE: CAN YOU IMPROVE IT?

How would you make the game easier? How would you make it more difficult? Can you use variables to have the player's score increase as they fire more arrows? Can you add a countdown timer to put more pressure on the player?


Hello,
world!
def
#

Chapter 5

Programming with Python

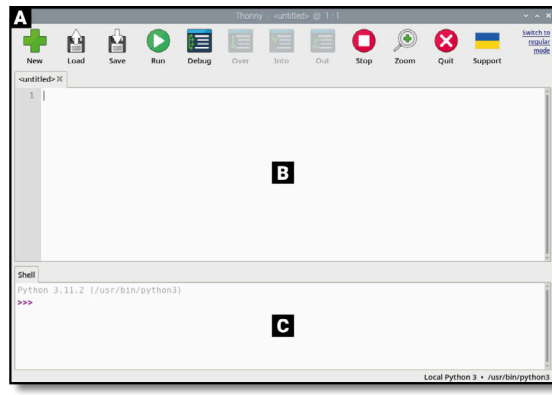
Now you've got to grips with Scratch, we'll show you how to do text-based coding with Python.

Named after the Monty Python comedy troupe, Guido van Rossum's Python has grown from a hobby project first released to the public in 1991 to a much-loved programming language that powers a wide range of projects. Unlike the visual environment of Scratch, Python is text based: you write instructions, using a simplified language and specific format, which the computer then carries out.

Python is a great next step for those who have already used Scratch, offering increased flexibility and a more traditional programming environment. That's not to say it's difficult to learn, though: with a little practice, anyone can write Python programs for everything from simple calculations through to surprisingly complicated games.

This chapter builds on terms and concepts introduced in Chapter 4, *Programming with Scratch 3*. If you haven't worked through the exercises in that chapter yet, you'll find this chapter easier to follow if you go back and do so first.

Introducing the Thonny Python IDE



A **Toolbar**

B **Script area**

C **Line numbers**

D **Python shell**

Thonny's 'Simple Mode' interface uses a bar of friendly icons (**A**) as its menu, allowing you to create, save, load, and run your Python programs, as well as test them in various ways. The script area (**B**) is where your Python programs are written, and is split into a main area for your program and a small side margin for showing line numbers (**C**). The Python shell (**D**) allows you to type individual instructions which are then run as soon as you press the **ENTER** key, and also provides information about running programs.



THONNY MODES

Thonny has two main versions of its interface: 'Regular Mode' and a 'Simple Mode', which is better for beginners. This chapter uses Simple Mode, which is loaded by default when you open Thonny from the **Programming** section of the Raspberry Pi menu.

Your first Python program: Hello, World!

Like the other pre-installed programs on Raspberry Pi, Thonny is available from the menu: click on the Raspberry Pi icon, move the cursor to the **Programming** section, and click on **Thonny Python IDE**. After a few seconds, the Thonny user interface (Simple Mode by default) will load.

Thonny is a package known as an *integrated development environment (IDE)*, a complicated-sounding name with a simple explanation. It gathers together, or *integrates*, all the different tools you need to write, or *develop*, software

into a single user interface, or *environment*. There are lots of IDEs available, some of which support many different programming languages, while others, like Thonny, focus on supporting a single language.

Unlike Scratch, which gives you visual building blocks as a basis for your program, Python is a more traditional programming language where everything is written down. Start your first program by clicking on the Python shell area at the bottom of the Thonny window, then type the following instruction before pressing the **ENTER** key:

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

When you press **ENTER**, you'll see that your program begins to run instantly. Python will respond, in the same shell area, with the message 'Hello, World!' (**Figure 5-1**), just as you asked. That's because the shell is a direct line to the Python *interpreter*, whose job it is to look at your instructions and *interpret* what they mean. This is known as *interactive mode*, and you can think of it like a face-to-face conversation with someone: as soon as you finish what you're saying, the other person will respond, then wait for whatever you say next.

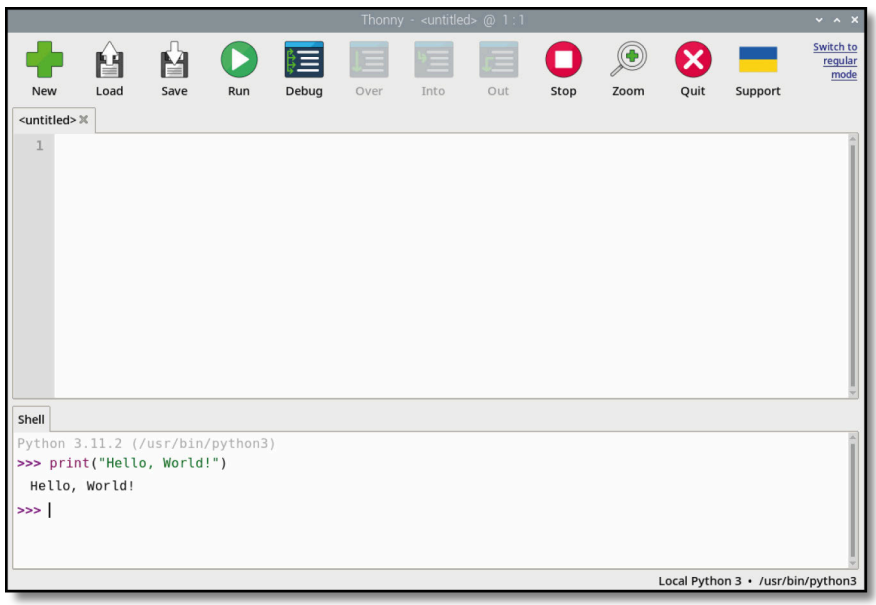


Figure 5-1 Python prints the 'Hello, World!' message in the shell area



SYNTAX ERROR

If your program doesn't run but instead prints a 'syntax error' message to the shell area, there's a mistake somewhere in what you've written. Python needs its instructions to be written in a very specific way: if you miss a bracket or a quotation mark, spell 'print' wrong or give it a capital P, or add extra symbols somewhere in the instruction, your program won't run. Try typing the instruction again, and make sure it matches the version in this book before pressing the **ENTER** key.

You don't have to use Python in interactive mode, though. Click on the script area in the middle of the Thonny window, then type your instruction again:

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

When you press the **ENTER** key this time, all you get is a new, blank line in the script area. To make this version of your program work, you have to click the **Run** icon in the Thonny toolbar. Before you do that, though, you should click the **Save** icon. Give your program a descriptive name, like **Hello World.py** and click the **OK** button. Once you've saved your program, click the **Run** icon and you'll see two messages appear in the Python shell area (**Figure 5-2**):

```
>>> %Run 'Hello World.py'
 Hello, World!
```

The first of these lines is an instruction from Thonny telling the Python interpreter to run the program. The second is the output of the program — the message you told Python to print. Congratulations: you've now written and run your first Python program in both interactive and script modes!



CHALLENGE: NEW MESSAGE

Can you change the message the Python program prints as its output? If you wanted to add more messages, would you use interactive mode or script mode? What happens if you remove the brackets or the quotation marks from the program and then try to run it?

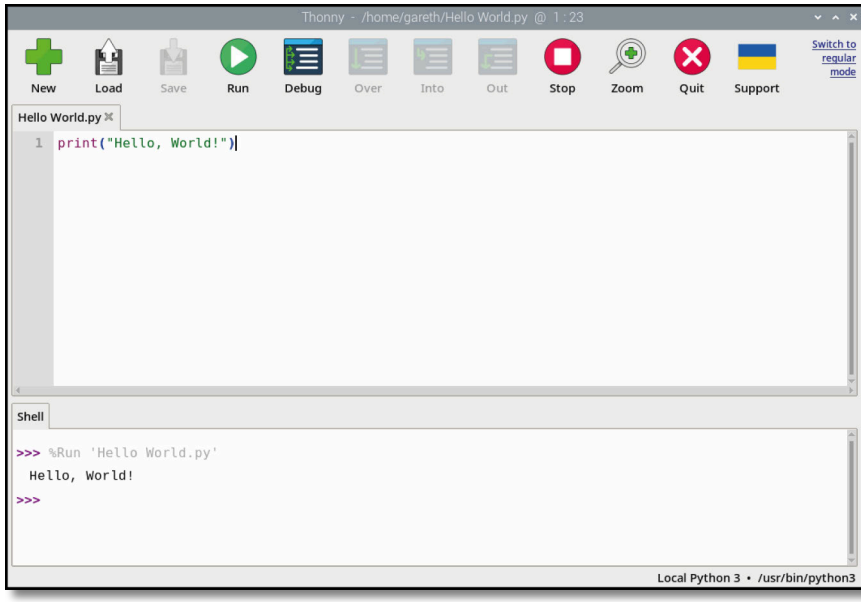


Figure 5-2 Running your simple program

Next steps: loops and code indentation

Just as Scratch uses stacks of jigsaw-like blocks to control which bits of the program are connected to which other bits, Python has its own way of controlling the sequence in which its programs run: *indentation*. Create a new program by clicking on the **New** icon in the Thonny toolbar. You won't lose your existing program; instead, Thonny will create a new tab above the script area. Start by typing in the following:

```
print("Loop starting!")
for i in range(10):
```

The first line prints a simple message to the shell, just like your Hello World program. The second begins a *definite* loop, which works in the same way as in Scratch: a counter, `i`, is assigned to the loop and is given a series of numbers to count. This is the `range` instruction, which tells the program to start at the number 0 and work upwards towards, but never reaching, the number 10. The colon symbol (`:`) tells Python that the next instruction should be part of the loop.

In Scratch, the instructions to be included in the loop are literally included inside the C-shaped block. Python uses a different approach: indenting code. The next line starts with four blank spaces, which Thonny should have added when you pressed **ENTER** after line 2:

```
    print("Loop number", i)
```

The blank spaces push this line inwards compared to the other lines. This indentation is how Python tells the difference between instructions outside the loop and instructions inside the loop; the indented code is *nested*.

You'll notice that when you pressed **ENTER** at the end of the third line, Thonny automatically indented the next line, assuming it would be part of the loop. To remove this, just press the **BACKSPACE** key once before typing the fourth line:

```
print("Loop finished!")
```

Your four-line program is now complete. The first line sits outside the loop and will only run once. The second line sets up the loop; the third sits inside the loop and will run once for each time the loop loops. The fourth line sits outside the loop once again.

```
print("Loop starting!")
for i in range(10):
    print("Loop number", i)
print("Loop finished!")
```

Click the **Save** icon, save the program as **Indentation.py**, then click the **Run** icon and look at the shell area for your program's output (**Figure 5-3**):

```
Loop starting!
Loop number 0
Loop number 1
Loop number 2
Loop number 3
Loop number 4
Loop number 5
Loop number 6
Loop number 7
Loop number 8
Loop number 9
Loop finished!
```

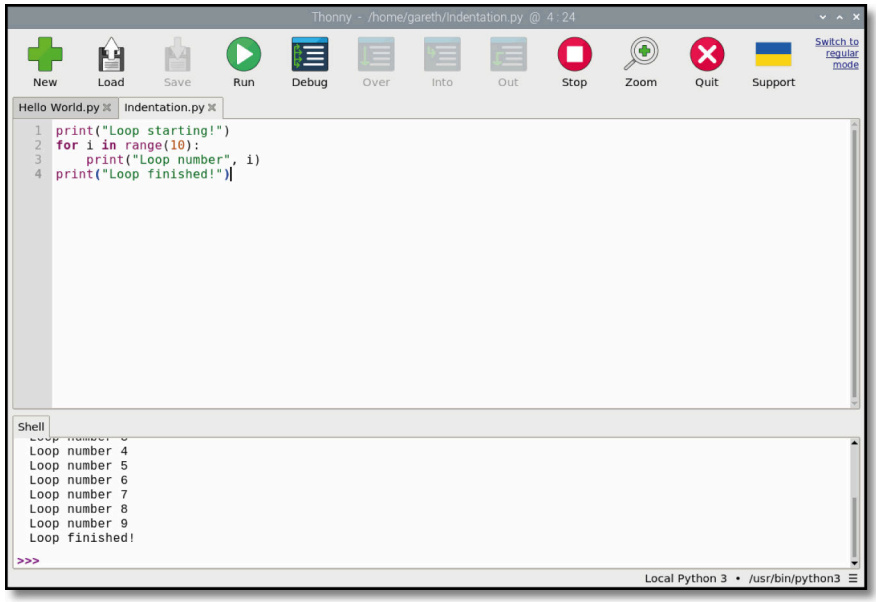


Figure 5-3 Executing a loop

COUNT FROM ZERO

Python is a zero-indexed language — meaning it starts counting from 0, not from 1 — which is why your program prints the numbers 0 through 9 rather than 1 through 10. If you wanted to, you could change this behaviour by switching the `range(10)` instruction to `range(1, 11)` — or any other numbers you like.

Indentation is a powerful and integral part of Python, and it's also one of the most common reasons for a program to not work as you expect. When looking for problems in a program, a process known as *debugging*, always double-check the indentation — especially when you begin nesting loops within loops.

Python also supports *infinite* loops, which run without end. To change your program from a definite loop to an infinite loop, edit line 2 to read:

```
while True:
```

If you click the **Run** icon now, you'll get an error: `name 'i' is not defined`. This is because you've deleted the line which created and assigned a value to the variable `i`.

To fix this, simply edit line 3 so it no longer uses the variable:

```
print("Loop running!")
```

Click the **Run** icon, and — if you're quick — you'll see the '`Loop starting!`' message followed by a never-ending string of '`Loop running`' messages (**Figure 5-4**). The '`Loop finished!`' message will never print, because the loop has no end: every time Python finishes printing the '`Loop running!`' message, it goes back to the beginning of the loop and prints it again.

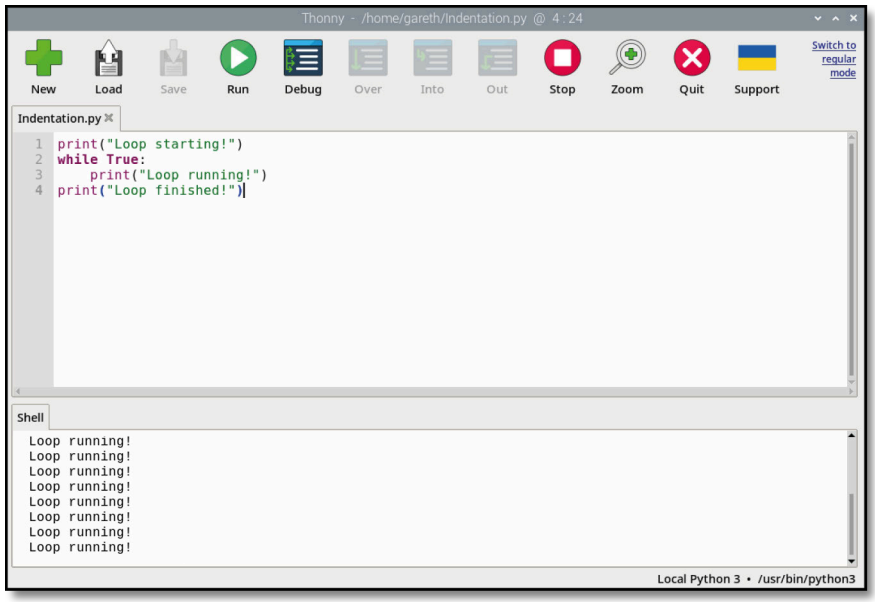


Figure 5-4 An infinite loop keeps going until you stop the program

Click the **Stop** icon on the Thonny toolbar to tell the program to stop what it's doing — known as *interrupting* the program. You'll see a message appear in the Python shell area, and the program will stop without ever reaching line 4.

CHALLENGE: LOOP THE LOOP

Can you change the loop back into a definite loop? Can you add a second definite loop to the program? How would you add a loop within a loop, and how would you expect that to work?

Conditionals and variables

Variables in Python, as in all programming languages, exist for more than just controlling loops. Start a new program by clicking the **New** icon on the Thonny menu, then type the following into the script area:

```
userName = input("What is your name? ")
```

Click the **Save** icon, save your program as **Name Test.py**, click **Run**, and watch what happens in the shell area. You should see a prompt asking for your name. Type your name into the shell area, followed by **ENTER**. Because that's the only instruction in your program, nothing else will happen (**Figure 5-5**). If you want to do anything with the data you placed into the variable, you'll need more lines in your program.

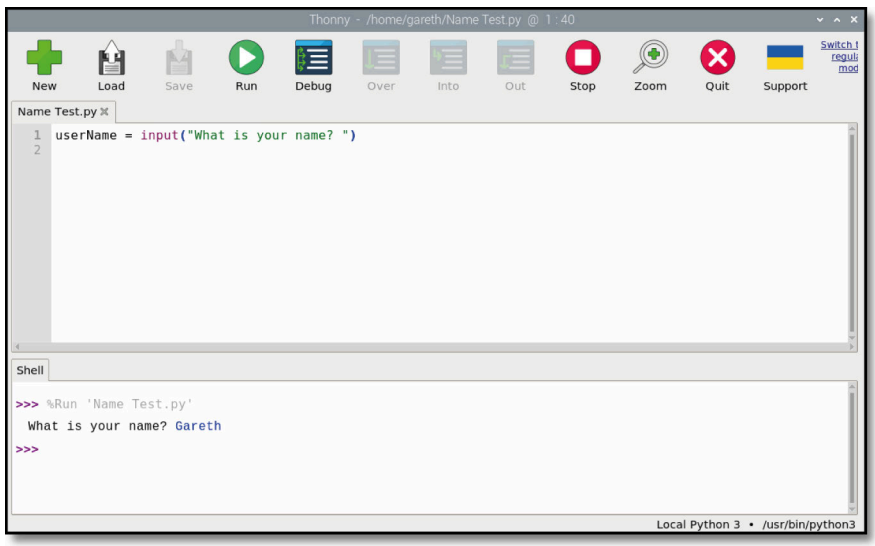


Figure 5-5 The `input` function lets you ask a user for some text input

To make your program do something useful with the name, add a *conditional statement* by typing the following:

```
if userName == "Clark Kent":
    print("You are Superman!")
else:
    print("You are not Superman!")
```

Remember that when Thonny sees that your code needs to be indented, it will do so automatically — but it doesn't know when your code needs to stop being indented, so you'll have to delete the spaces yourself before you type `else:`.

Click **Run** and enter your name into the shell area. Unless your name happens to be Clark Kent, you'll see the message 'You are not Superman!'. Click **Run** again, and this time type the name 'Clark Kent' — making sure to write it exactly as in the program, with a capital C and K. This time, the program recognises that you are, in fact, Superman (**Figure 5-6**).

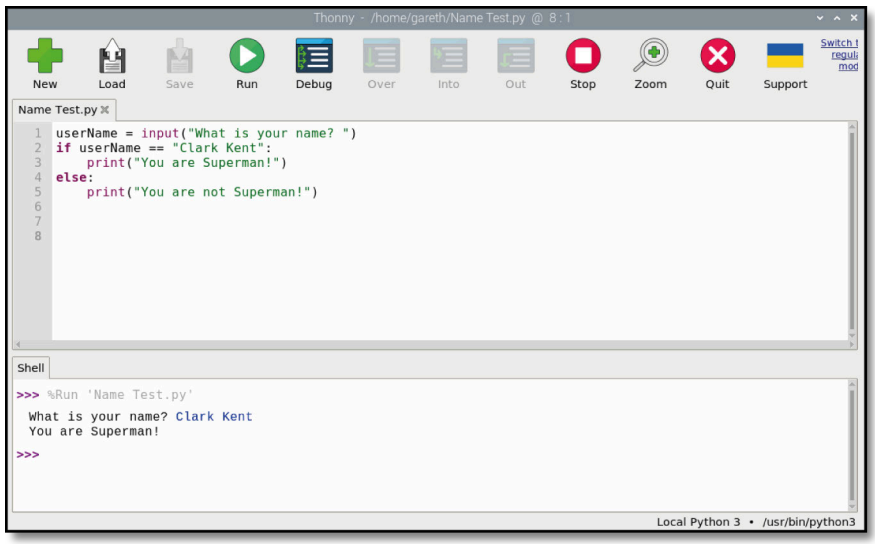


Figure 5-6 Shouldn't you be out saving the world?

The `==` symbols tell Python to do a direct comparison, looking to see if the variable `userName` matches the text — known as a *string* — in your program. If you're working with numbers, there are other comparisons you can make: `>` to see if a number is greater than another number, `<` to see if it's less than, `=>` to see if it's equal to or greater than, and `=<` to see if it's equal to or less than. There's also `!=`, which means not equal to; it's the exact opposite of `==`. These symbols are known as *comparison operators*.



USING = AND ==

The key to using variables is to learn the difference between `=` and `==`. Remember: `=` means 'make this variable equal to this value', while `==` means 'check to see if the variable is equal to this value'. Mixing them up is a sure way to end up with a program that doesn't work!

You can also use comparison operators in loops. Delete lines 2 to 5, then type the following in their place:

```
while userName != "Clark Kent":
    print("You are not Superman - try again!")
    userName = input ("What is your name? ")
print("You are Superman!")
```

Click the **Run** icon again. This time, instead of quitting, the program will keep asking for your name until it confirms that you are Superman (**Figure 5-7**) — sort of like a very simple password. To get out of the loop, either type 'Clark Kent' or click the **Stop** icon on the Thonny toolbar. Congratulations: you now know how to use conditionals and variables!

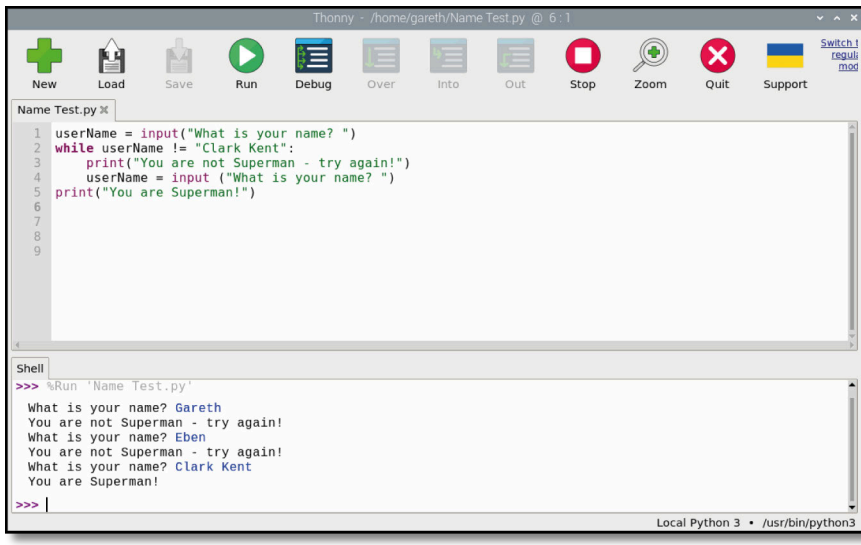


Figure 5-7 It will keep asking for your name until you say it's 'Clark Kent'

CHALLENGE: ADD MORE QUESTIONS

Can you change the program to ask more than one question, storing the answers in multiple variables? Can you make a program which uses conditionals and comparison operators to print whether a number typed in by the user is higher or lower than 5, like the program you created in Chapter 4, *Programming with Scratch 3*?

Project 1: Turtle Snowflakes

Now you understand how Python works, it's time to play with graphics and create a snowflake using a tool known as a *turtle*.

Turtles are robots shaped like their animal namesakes, which are designed to move in a straight line, turn, and to lift and lower a pen. Simply put, a turtle — whether physical or digital — will start or stop drawing a line as it moves. Unlike some other languages, namely Logo and its many variants, Python doesn't have a built-in turtle tool, but it does come with a *library* of add-on code to give it turtle power. Libraries are bundles of code which add new instructions to expand the capabilities of Python, and are brought into your own programs using an `import` command.

Create a new program by clicking on the **New** icon, then type the following:

```
import turtle
```

When using instructions included in a library, you have to use the library name followed by a full stop, then the instruction name. That can be annoying to type out every time, so you can assign an instruction to a variable with a short name. It could be as short as just one letter, but we thought it might be nice for it to double as a pet name for the turtle. Type the following:

```
pat = turtle.Turtle()
```

To test your program out, you'll need to give your turtle something to do. Type:

```
pat.forward(100)
```

Click the **Save** icon, save your program as **Turtle Snowflakes.py**, then click the **Run** icon and a new window called 'Turtle Graphics' will appear showing the result of your program: your turtle, Pat, will move forwards 100 units, drawing a straight line (**Figure 5-8**).

Switch back to the main Thonny window — if it's hidden behind the Turtle Graphics window, either click the minimise button on the Turtle Graphics window or click on the Thonny entry in the task bar at the top of the screen. Once you've brought the Thonny window to the front, click **Stop** to close the Turtle Graphics window.

Typing out every single movement instruction to draw something more complex by hand would be tedious, so delete line 3 and create a loop to do the hard work of creating shapes:

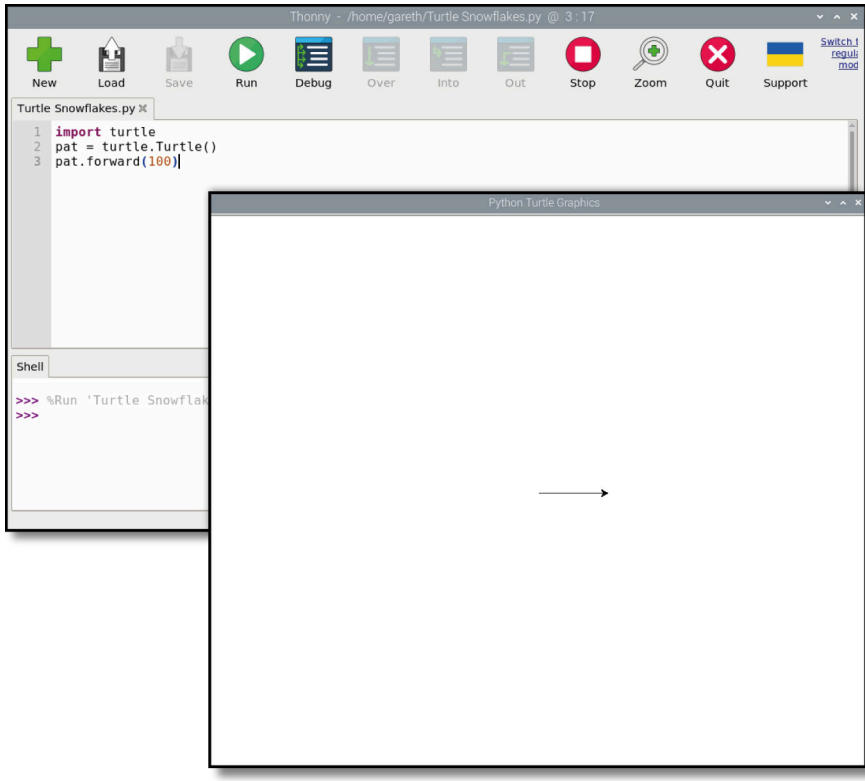


Figure 5-8 The turtle moves forward to draw a straight line

```
for i in range(2):
    pat.forward(100)
    pat.right(60)
    pat.forward(100)
    pat.right(120)
```

Run your program, and Pat will draw a single parallelogram (**Figure 5-9**).

To turn that into a snowflake-like shape, click **Stop** in the main Thonny window and create a loop around your loop by adding the following as line 3:

```
for i in range(10):
```

...and the following at the bottom of your program:

```
    pat.right(36)
```

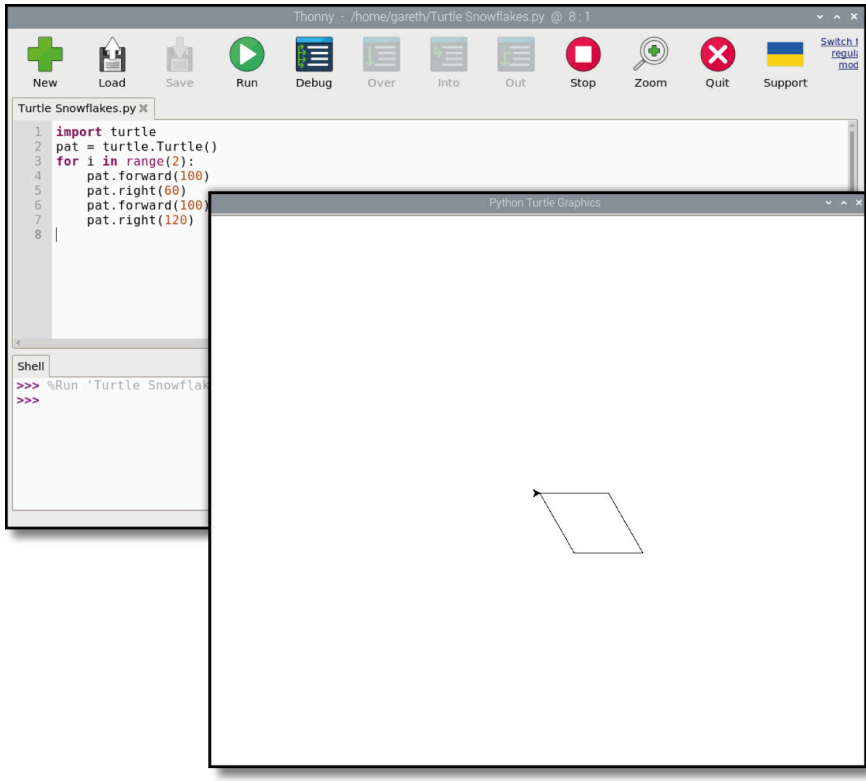


Figure 5-9 By combining turns and movements, you can draw shapes

Your program won't run as it is, because the existing loop isn't indented correctly. To fix that, click on the start of each line in the existing loop — lines 4 through 8 — and press the **SPACE** key four times to correct the indentation. Your program should now look like this:

```
import turtle
pat = turtle.Turtle()
for i in range(10):
    for i in range(2):
        pat.forward(100)
        pat.right(60)
        pat.forward(100)
        pat.right(120)
    pat.right(36)
```

Click the **Run** icon , and watch the turtle: it'll draw a parallelogram, as before, but when it's done it'll turn 36 degrees and draw another, then another, and so on until there are ten overlapping parallelograms on the screen — looking a little like a snowflake (**Figure 5-10**).

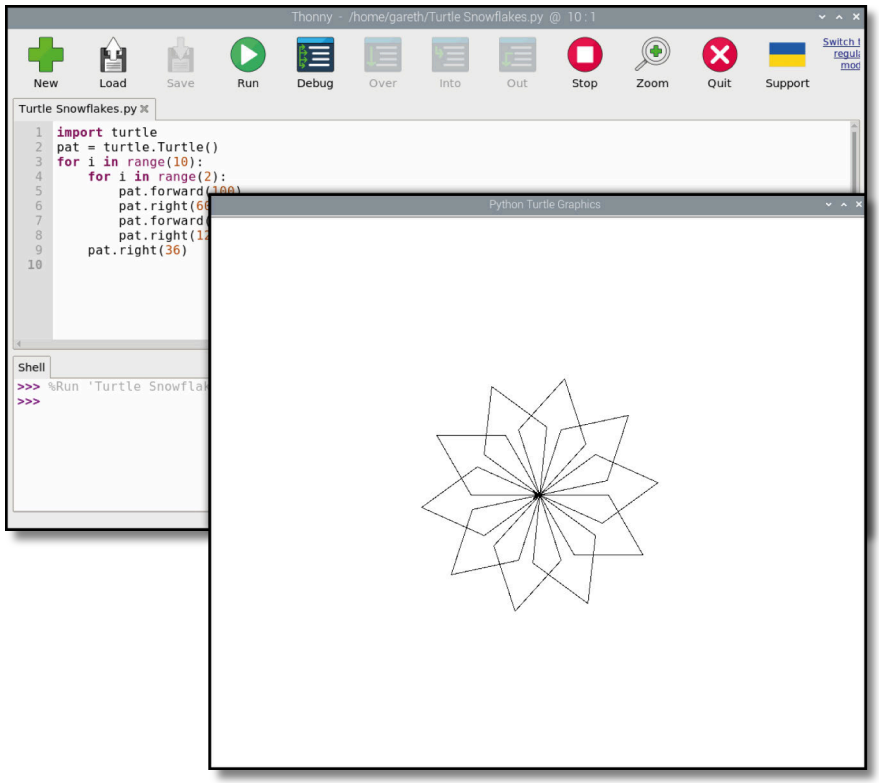


Figure 5-10 Repeating the shape to make a more complex one

While a robotic turtle draws in a single colour on a large piece of paper, Python's simulated turtle can use a range of colours. Add the following as a new line 3 and 4, pushing the existing lines down:

```
turtle.Screen().bgcolor("blue")
pat.color("cyan")
```

Run your program again and you'll see the effect of your new code: the background colour of the Turtle Graphics window has changed to blue, and the snowflake is now cyan (**Figure 5-11**).

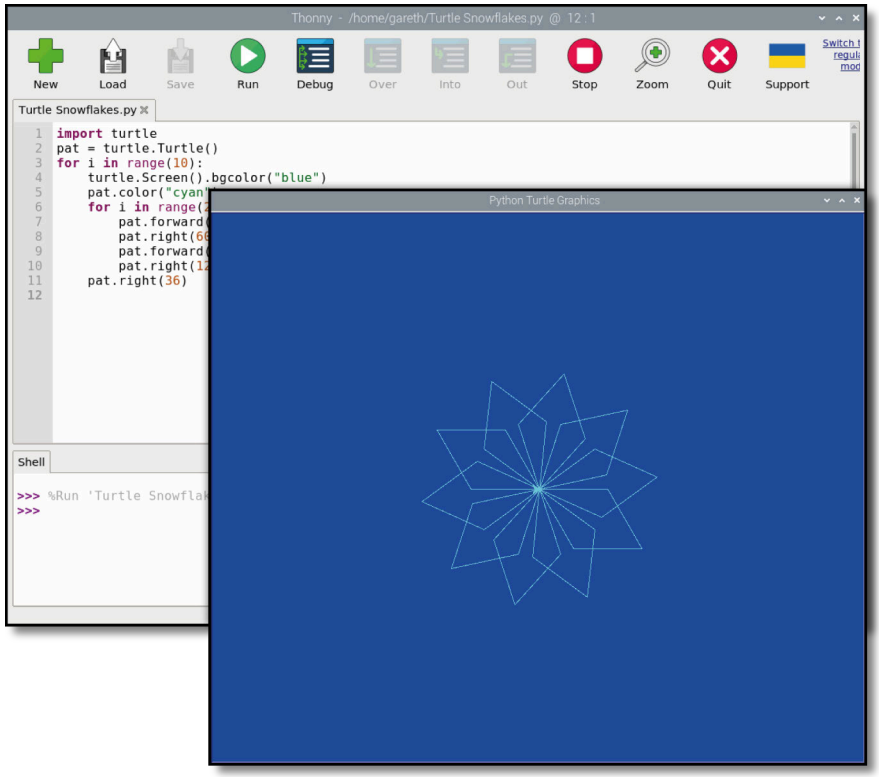


Figure 5-11 Changing the background and snowflake colours

You can also choose the colours randomly from a selection, using the `random` library. Go back to the top of your program and insert the following as line 2:

```
import random
```

Change the background colour in what is now line 4 from 'blue' to 'grey', then create a new variable called `colours` as a new line 5:

```
colours = ["cyan", "purple", "white", "blue"]
```

This type of variable is called a *list*, and is marked by square brackets. In this case, the list is filled with possible colours for the snowflake segments, but you still need to tell Python to choose one each time the loop repeats. At the

very end of the program, enter the following — making sure it's indented with four spaces so it forms part of the outer loop, just like the line above it:

```
    pat.color(random.choice(colours))
```

U.S. SPELLINGS

Many programming languages use American English spellings, and Python is no exception: the command for changing the colour of the turtle's pen is spelled `color`, and if you spell it the British English way as `colour` it simply won't work. Variables, though, can have any spelling you like — which is why you're able to call your new variable `colours` and have Python understand.

Click the **Run** icon and the snowflake-stroke-ninja-star will be drawn again. This time, though, Python will choose a random colour from your list as it draws each petal — giving the snowflake a multicolour finish, as shown in **Figure 5-12**.

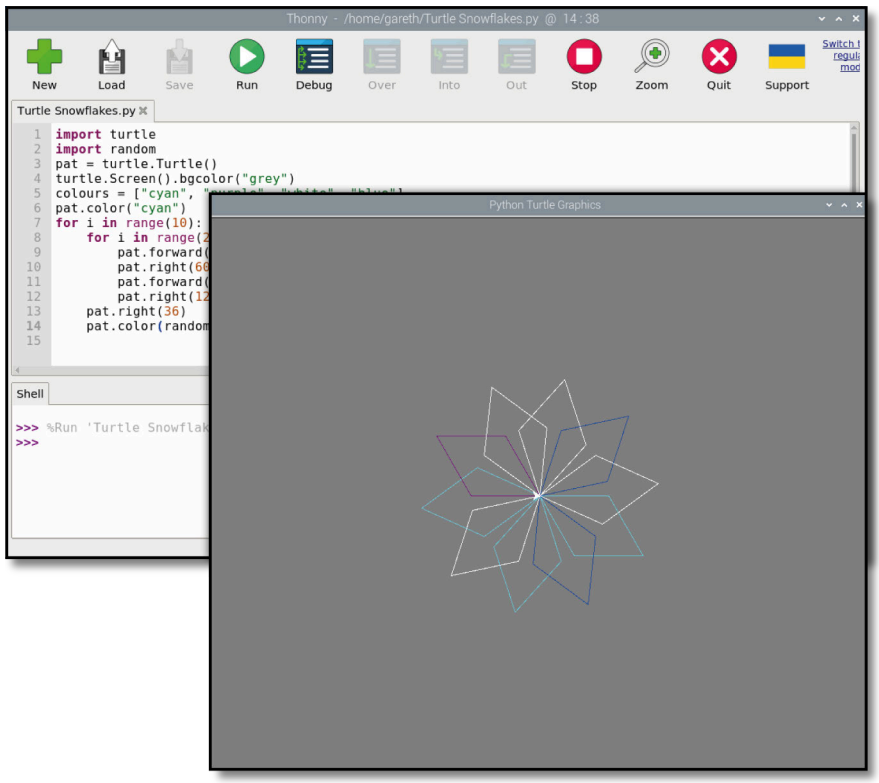


Figure 5-12 Using random colours for the 'petals'

To make the snowflake look less like a ninja star and more like an actual snowflake, add a new line 6, directly below your `colours` list, and type the following:

```
pat.penup()
pat.forward(90)
pat.left(45)
pat.pendown()
```

The `penup` and `pendown` instructions would move a physical pen off and on to the paper if used with a turtle robot, but in the virtual world it just tells your turtle when to stop and start drawing lines. This time, rather than using a loop, you're going to create a *function* — a segment of code which you can call at any time, essentially creating your very own Python instruction.

Start by deleting the code for drawing your parallelogram-based snowflakes: that's everything between and including the `pat.color("cyan")` instruction on line 10 through to `pat.right(36)` on line 17. Leave the `pat.color(random.choice(colours))` instruction, but add a hash symbol (`#`) at the start of the line. This is known as *commenting out* an instruction, which means that Python will ignore it when the program runs. You can use comments to add explanations to your code, which will make it a lot easier to understand when you come back to it a few months later, or if you send it on to someone else.

Create your function, which we'll call `branch`, by typing the following instruction onto line 10, below `pat.pendown()`:

```
def branch():
```

This *defines* your function by giving it a name, `branch`. When you press the **ENTER** key, Thonny will automatically add indentation for the function's instructions. Type the following, making sure to pay close attention to indentation — because at one point you're going to be nesting code three indentation levels deep!

```
    for i in range(3):
        for i in range(3):
            pat.forward(30)
            pat.backward(30)
            pat.right(45)
        pat.left(90)
        pat.backward(30)
        pat.left(45)
    pat.right(90)
    pat.forward(90)
```

Finally, create a new loop at the bottom of your program — but above the commented-out `color` line — to run, or *call*, your new function:

```
for i in range(8):
    branch()
    pat.left(45)
```

Your finished program should look like this:

```
import turtle
import random

pat = turtle.Turtle()
turtle.Screen().bgcolor("grey")
colours = ["cyan", "purple", "white", "blue"]

pat.penup()
pat.forward(90)
pat.left(45)
pat.pendown()

def branch():
    for i in range(3):
        for i in range(3):
            pat.forward(30)
            pat.backward(30)
            pat.right(45)
        pat.left(90)
        pat.backward(30)
        pat.left(45)
    pat.right(90)
    pat.forward(90)

for i in range(8):
    branch()
    pat.left(45)
#    pat.color(random.choice(colours))
```

Click on **Run** and watch the graphics window as Pat draws based on your instructions. Congratulations: your snowflake now looks a lot more like a snowflake (**Figure 5-13**)!

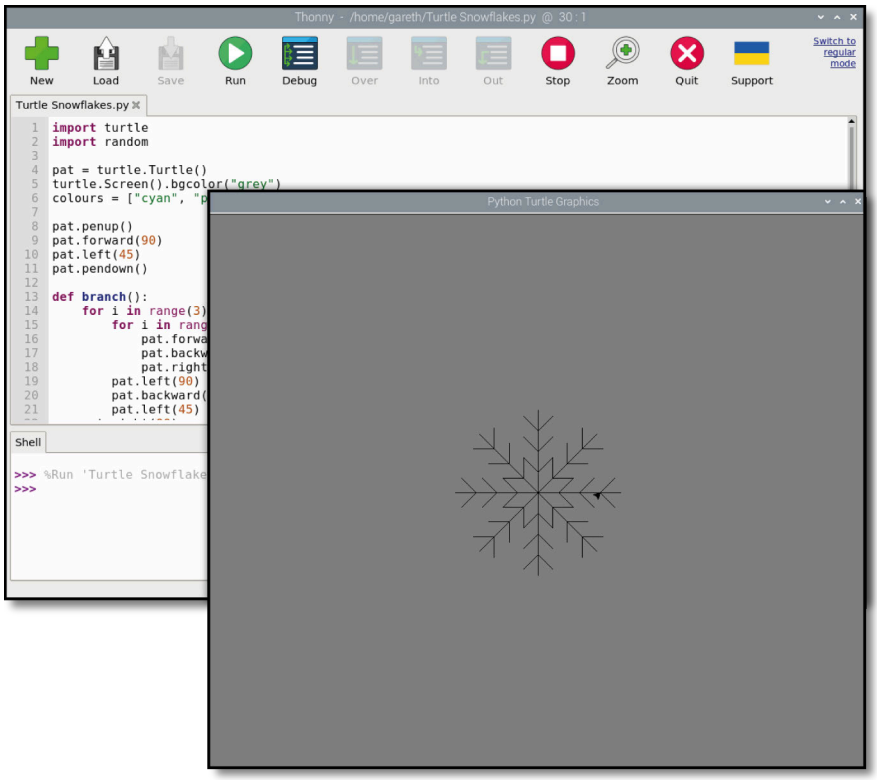


Figure 5-13 Extra branches make it look like a snowflake



CHALLENGE: WHAT'S NEXT?

Can you use your commented-out instruction to have the branches of the snowflake drawn in different colours? Can you create a 'snowflake' function, and use it to draw lots of snowflakes on the screen? Can you make your program change the size and colour of the snowflakes at random?

Project 2: Scary Spot the Difference

Python can handle pictures and sounds as well as turtle-based graphics, and those can be used to great effect as a prank on your friends — a spot-the-difference game with a scary secret at its heart, perfect for Halloween!

This project needs two images — your spot-the-difference image plus a 'scary' surprise image — and a sound file. Click on the Raspberry Pi icon to load the Raspberry Pi menu, choose the **Internet** category, and click on **Chromium Web Browser**. Once it's open, type **rptl.io/spot-pic** into the address bar followed by the **ENTER** key. Right-click on the picture and click on **Save image as**, choose the **Home** folder from the list on the left-hand side, then click **Save**. Click back on Chromium's address bar, then type **rptl.io/scary-pic** followed by the **ENTER** key. As before, right-click the picture, click **Save image as**, choose the **Home** folder, then click **Save**.

To get the sound file, click back into the address bar and type **rptl.io/scream** followed by the **ENTER** key. This file — the sound of a scream to give your player a real surprise — will download automatically. It needs to be moved to the **Home** folder before you can use it. Open a new File Manager window, go to your **Downloads** folder, and find the **en_images_....wav** file you just downloaded. Right-click the file and click **Rename**, then change its name to **scream.wav**. Right-click **scream.wav**, then click **Cut**. Finally, click on **Home Folder** at the top-left of the file manager, right-click in any empty space in the large file viewing window on the right, and click **Paste**. You can now close the Chromium and File Manager windows.

Click the **New** icon ✚ in the Thonny toolbar to begin a new project. As before, you're going use a library to extend Python's capabilities. This time it's the `pygame` library, which, as the name suggests, was created with games in mind. Type the following:

```
import pygame
```

You'll need some parts from other libraries and a subsection of the Pygame library too. Import these by typing the following:

```
from pygame.locals import *
from time import sleep
from random import randrange
```

The `from` instruction works differently to the `import` instruction by allowing you to import only the parts of a library you need rather than the whole library. Next, you need to set up Pygame; this is called *initialisation*. Pygame

needs to know the width and height of the player's monitor or TV, known as its *resolution*. Type the following:

```
pygame.init()
width = pygame.display.Info().current_w
height = pygame.display.Info().current_h
```

The final step in setting up Pygame is to create its main window, which Pygame calls a screen. Type the following:

```
screen = pygame.display.set_mode((width, height))
pygame.display.update()
# Your code goes here
pygame.quit()
```

Note the commented line in the middle: this is where your program will go. For now, though, click the **Save** icon, save your program as **Spot the Difference.py**, then click the **Run** icon and watch. Pygame will create a window, fill it with a black background, and then almost immediately close the window as it reaches the instruction to quit. Aside from a short message in the shell (**Figure 5-14**), the program hasn't achieved much so far.

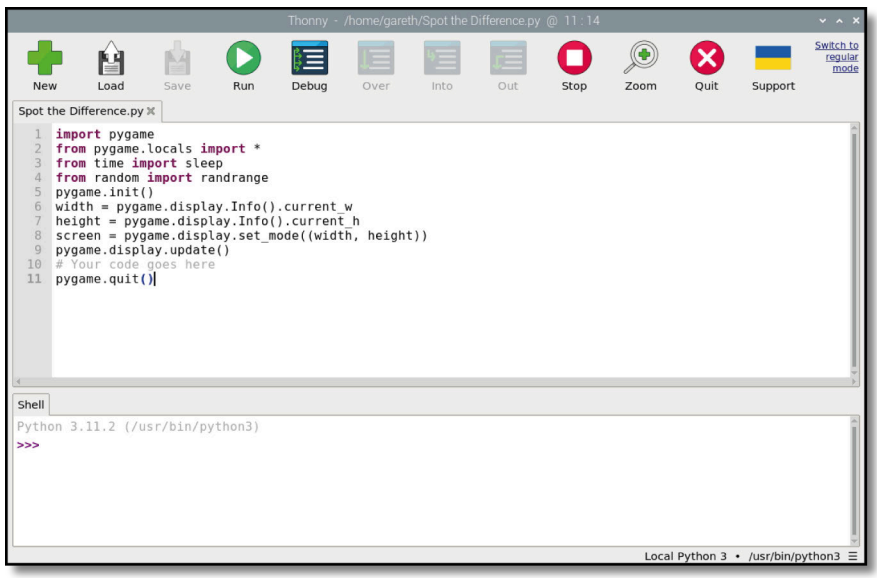


Figure 5-14 Your program is functional, but doesn't do much yet

To display your spot-the-difference image, delete the comment above `pygame.quit()` and type the following in the space:

```
difference = pygame.image.load('spot_the_diff.png')
```

To make sure the image fills the screen, you need to scale it to match your player's monitor or TV's resolution. Type the following:

```
difference = pygame.transform.scale(difference, (width, height))
```

Now that the image is in memory, you need to tell Pygame to actually display it on the screen — a process known as *blitting*, or a *bit block transfer*. Type the following:

```
screen.blit(difference, (0, 0))
pygame.display.update()
```

The first of these lines copies the image onto the screen, starting at the top-left corner; the second tells Pygame to redraw the screen. Without this second line, the image will be in the correct place in memory, but you'll never see it!

Click the **Run** icon, and the image, shown in **Figure 5-15**, will briefly appear on screen.



Figure 5-15 Your spot-the-difference image

To have the image on screen for a longer period of time, add the following line just above `pygame.quit()`:

```
sleep(3)
```

Click the **Run** icon again, and the image will stay on the screen longer. Add your surprise image by typing the following just below the line `pygame.display.update()`:

```
zombie = pygame.image.load('scary_face.png')
zombie = pygame.transform.scale(zombie, (width, height))
```

Add a delay, so the zombie image doesn't appear right away:

```
sleep(3)
```

Then blit the image to the screen and update to show it to the player:

```
screen.blit(zombie, (0,0))
pygame.display.update()
```

Click the **Run** icon and watch what happens: Pygame will load your spot-the-difference image, but after three seconds it will be replaced with the scary zombie (**Figure 5-16**)!

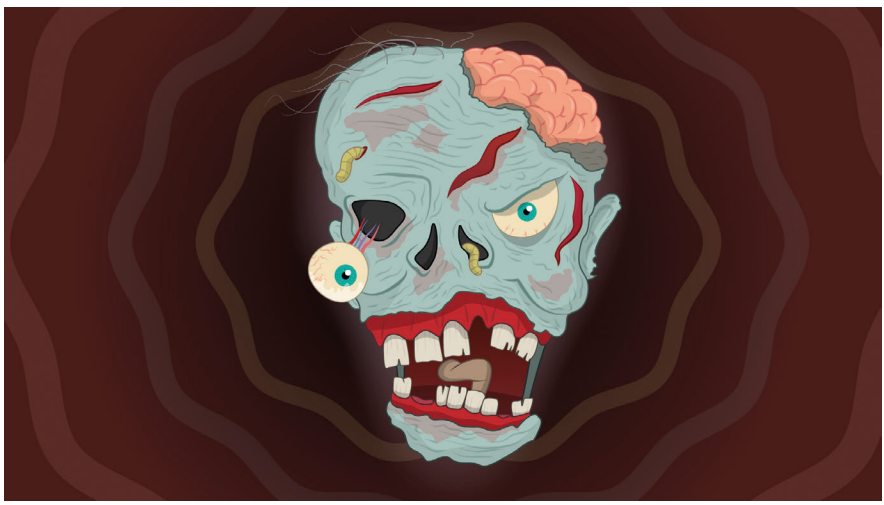


Figure 5-16 It'll give someone a scary surprise

Having the delay set at three seconds makes things a bit too predictable, though. Change the line `sleep(3)` above `screen.blit(zombie, (0,0))` to:

```
sleep(randrange(5, 15))
```

This picks a random number between 5 and 15 and delays the program for that long. Next, add the following line just above your `sleep` instruction to load the scream sound file:

```
scream = pygame.mixer.Sound('scream.wav')
```

Type the following on a new line after your sleep instruction to start playing the sound. It should kick in just ahead of the scary image actually being shown to the player:

```
scream.play()
```

Finally, tell Pygame to stop playing the sound by typing the following line just above `pygame.quit()`:

```
scream.stop()
```

Click the **Run** icon and admire your handiwork: after a few seconds of innocent spot-the-difference fun, your scary zombie will appear alongside a blood-curdling shriek — sure to give your friends a fright! If you find that the zombie picture appears before the sound starts playing, you can compensate by adding a small delay just after your `scream.play()` instruction and before your `screen.blit` instruction:

```
sleep(0.4)
```

Your finished program should look like this:

```
import pygame
from pygame.locals import *
from time import sleep
from random import randrange

pygame.init()
width = pygame.display.Info().current_w
height = pygame.display.Info().current_h
screen = pygame.display.set_mode((width, height))
pygame.display.update()

difference = pygame.image.load('spot_the_diff.png')
difference = pygame.transform.scale(difference, (width, height))
screen.blit(difference, (0, 0))
pygame.display.update()

zombie = pygame.image.load('scary_face.png')
```

```
zombie = pygame.transform.scale (zombie, (width, height))
scream = pygame.mixer.Sound('scream.wav')
sleep(randrange(5, 15))
scream.play()
screen.blit(zombie, (0,0))
pygame.display.update()

sleep(3)
scream.stop()
pygame.quit()
```

Now all that's left to do is to invite your friends to play spot-the-difference — and to make sure their speakers are turned up, of course!



CHALLENGE: ALTER THE LOOK

Can you change the images to make the prank more appropriate for other events, like Christmas? Can you draw your own spot-the-difference and scary images (using a graphics editor such as GIMP)? Could you track the user clicking on a difference to make it more convincing?

Project 3: Text Adventure

Now that you're getting the hang of Python, it's time to use Pygame to make something a little more complicated: a fully-functional text-based maze game based on classic text adventures. Also known as interactive fiction, these games date back to when computers couldn't handle complex graphics, but they still have their fans who argue that no graphics will ever be as vivid as those you have in your imagination.

This program is quite a bit more complex than the others in this chapter. To make things easier you will start with a partially-written version. Open the Chromium Web Browser and go to **rptl.io/text-adventure**.

Chromium will load the code for the program in the browser. Right-click the browser page, choose **Save As**, and save the file as **text-adventure.py** to your Downloads folder, but it may warn you that this type of file — a Python program — could harm your computer. You've downloaded the file from a trusted source, so click on the **Keep** button if the warning message appears at the bottom of the screen. Go back to Thonny, then click the **Load** icon . Find the file, **text-adventure.py**, in your **Downloads** folder and click the **Load** button.

Start by clicking the **Run** icon to familiarise yourself with how a text adventure works. The game's output appears in the shell area at the bottom of the Thonny window. If necessary, you can make the Thonny window larger by clicking on the maximise button to make it easier to read.

As it stands now, the game is very simple: there are two rooms and no objects. The player starts in the **Hall**, the first of the two rooms. To go to the **Kitchen**, simply type '**go south**' followed by the **ENTER** key (**Figure 5-17**). When you're in the **Kitchen**, you can type '**go north**' to return to the **Hall**. You can also try typing '**go west**' and '**go east**', but as there aren't any rooms in those directions the game will show you an error message.

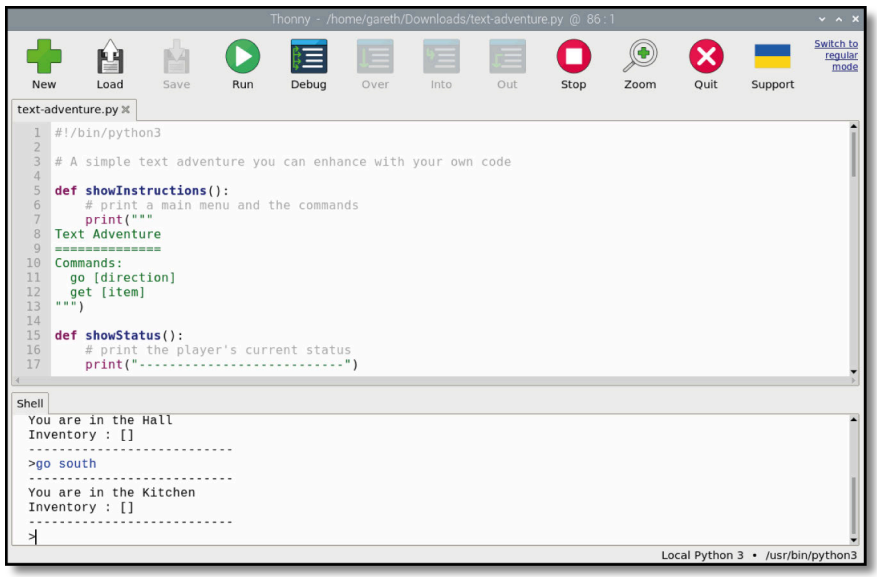


Figure 5-17 There are only two rooms so far

Press the **Stop** icon to stop the program, then scroll down to line 30 of the program in the script area to find a variable called **rooms**. This type of variable is known as a *dictionary*, and it defines the rooms, their exits, and which room a given exit leads to.

To make the game more interesting, let's add another room: a **Dining Room** to the east of the **Hall**.

Find the `rooms` variable in the scripts area, and extend it by adding a comma symbol (`,`) after the `}` on line 38, then typing the following:

```
'Dining Room' : {
    'west' : 'Hall'
}
```

You'll also need a new exit in the `Hall`, as one isn't automatically created for you. Go to the end of line 33, add a comma, then add the following line:

```
    'east' : 'Dining Room'
```

Click the **Run** icon and visit your new room: type '`go east`' while in the `Hall` to enter the `Dining Room` (**Figure 5-18**), and type '`go west`' while in the `Dining Room` to re-enter the `Hall`. Congratulations: you've made a room of your own!

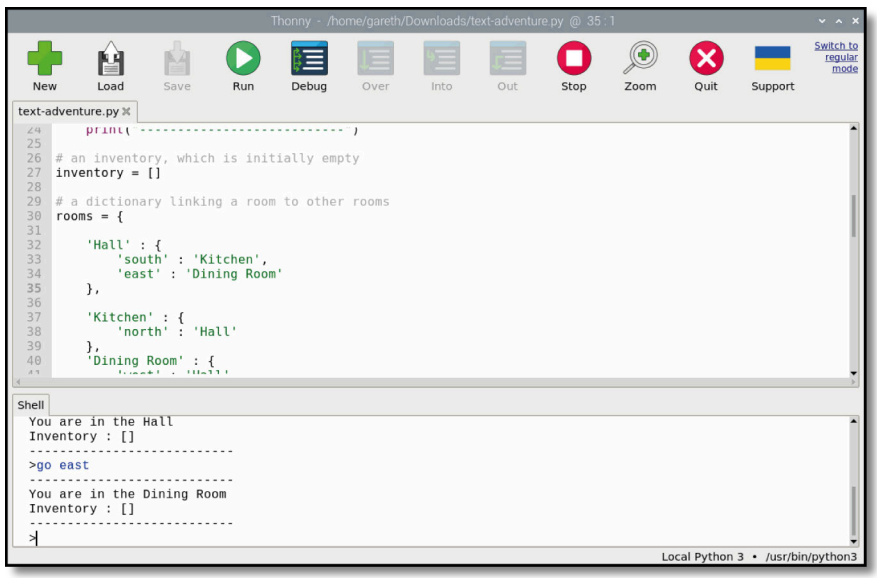


Figure 5-18 You have added another room

Empty rooms aren't much fun, though. To add an item to a room, you need to modify that room's dictionary. Click the **Stop** icon to stop the program. Find the `Hall` dictionary in the scripts area, add a comma to the end of the line `'east' : 'Dining Room'`, press **ENTER**, then type this line:

```
    'item' : 'key'
```

Click the **Run** icon again. This time, the game will tell you that you can see your new item: a key. Type '`get key`' (**Figure 5-19**) to pick it up and add it to the list of items you're carrying — known as your *inventory*. Your inventory stays with you as you travel from room to room.

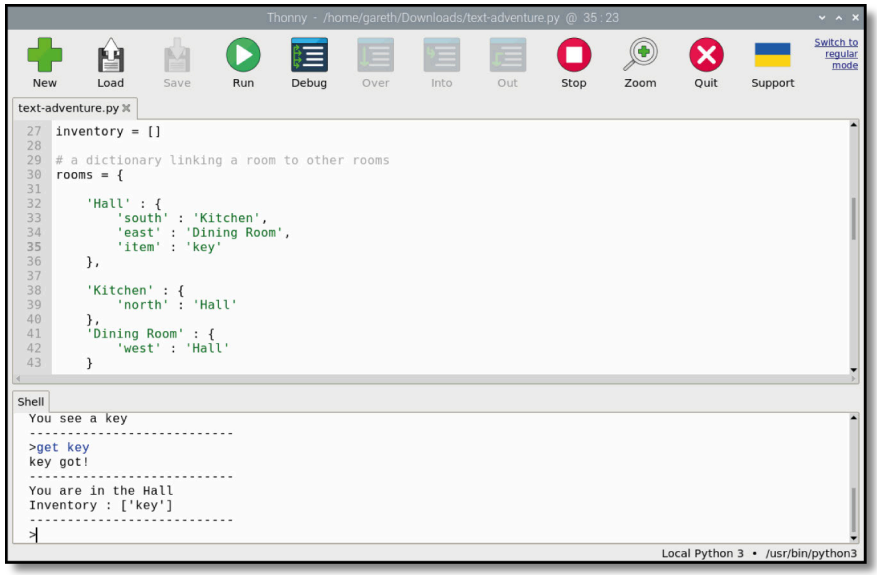


Figure 5-19 The collected key is added to your inventory

Click the **Stop** icon and make the game even more interesting by adding a monster to avoid. Find the `Kitchen` dictionary entry, and add a '`monster`' item in the same way as you added the '`key`' item — remembering to add a comma to the end of the line above:

```
        'item' : 'monster'
```

You need to add some logic to enable the monster to attack the player. Scroll to the bottom of the program in the script area and add the following lines — including the comment, marked with a hash symbol, which will help you understand the program if you come back to it another day. Be sure to indent the lines and type everything between `if` and the colon (`:`) on one line:

```
# player loses if they enter a room with a monster
if 'item' in rooms[currentRoom]
      and 'monster' in rooms[currentRoom]['item']:
    print('A monster has got you... GAME OVER!')
    break
```

Click the **Run** icon and try going into the Kitchen (**Figure 5-20**) — the monster won't be too impressed when you do!

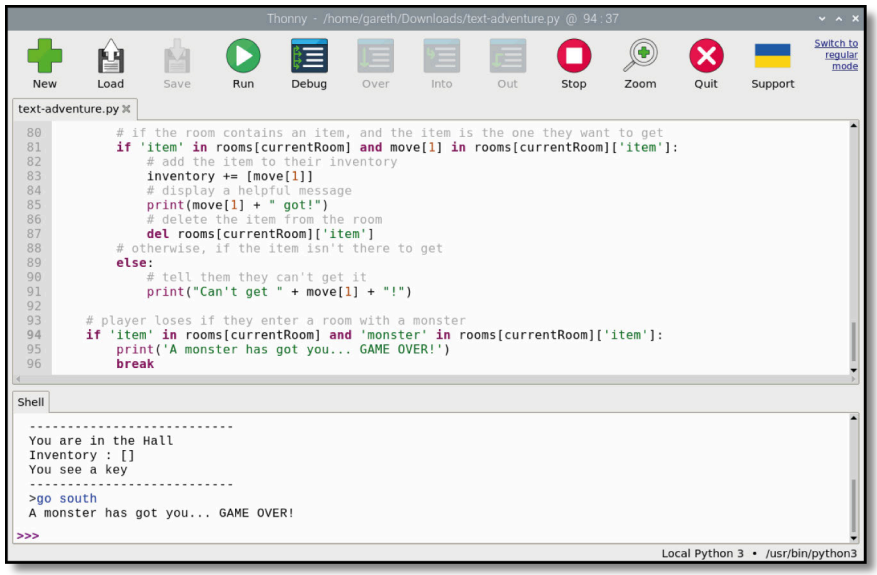


Figure 5-20 Never mind rats, there's a monster in the kitchen

To turn this adventure into a proper game, you should add more items, another room, and the ability to 'win' by leaving the house with all the items safely stashed in your inventory. Start by adding another room as before with the `Dining Room` — only this time, it's a `Garden`. Add an exit from the `Dining Room` dictionary, remembering to add a comma to the end of the line above:

```
        'south' : 'Garden'
```

Then add your new room to the main `rooms` dictionary, again remembering to add a comma after the `}` on the line above:

```
    'Garden' : {
        'north' : 'Dining Room'
    }
```

Add a 'potion' object to the `Dining Room` dictionary, again remembering to add the necessary comma to the line above:

```
        'item' : 'potion'
```

Finally, scroll to the bottom of the program and add the logic required to check if the player has all the items and, if so, tell them they've won the game (be sure indent the lines and to type everything between `if` and the colon (`:`) on one line):

```
# player wins if they get to the garden with a key and a potion
if currentRoom == 'Garden' and 'key' in inventory
      and 'potion' in inventory:
   print('You escaped the house... YOU WIN!')
   break
```

Click the **Run** icon and try to finish the game by picking up the key and the potion before going to the garden. Remember not to enter the `Kitchen`, because that's where the monster is!

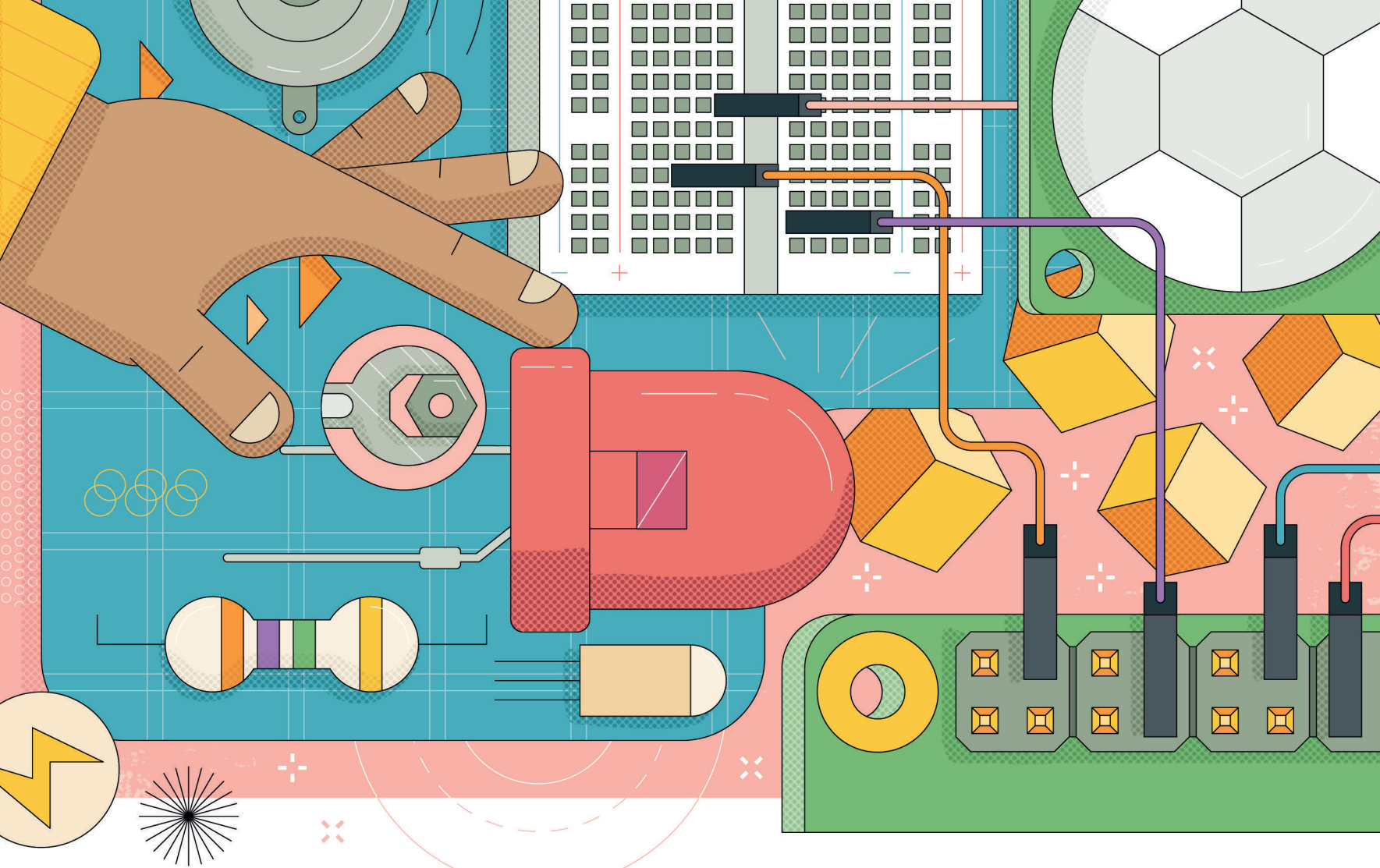
As a last tweak for the game, add some instructions telling the player how to complete the game. Scroll to the top of the program, where the function `showInstructions()` is defined, and add the following:

```
Get to the Garden with a key and a potion
Avoid the monsters!
```

Run the game one last time, and you'll see your new instructions appear at the very start. Congratulations, you've made an interactive text-based maze game!

CHALLENGE: EXPAND THE GAME

Can you add more rooms to make the game last longer? Can you add an item to protect you from the monster? How would you add a weapon to slay the monster? Can you add rooms that are above and below the existing rooms, accessed by stairs?



Chapter 6

Physical computing with Scratch and Python

There's more to coding than doing things on screen — you can also control electronic components connected to your Raspberry Pi's GPIO pins.

When people think of 'programming' or 'coding', they're usually — and naturally — thinking about software. Coding can be about more than just software, though: it can also affect the real world through hardware. This is known as *physical computing*.

As the name suggests, physical computing is all about controlling things in the real world with your programs: using hardware alongside software. When you set the program on your washing machine, change the temperature on your programmable thermostat, or press a button at traffic lights to cross the road safely, you're using physical computing.

Raspberry Pi is a great device for learning about physical computing thanks to one key feature: the *general-purpose input/output* (*GPIO*) header.

Introducing the GPIO header

At the top edge of Raspberry Pi's circuit board, or at the back of Raspberry Pi 400, you'll find two rows of metal pins. This is the GPIO (general-purpose input/output) header and it's there so you can connect hardware like LEDs and switches to Raspberry Pi, and control them using programs you create. These pins can be used for both input and output.

Raspberry Pi's GPIO header is made up of 40 male pins as shown in **Figure 6-1**. Some pins are available for you to use in your physical computing projects, some pins provide power, and other pins are used to communicate with add-on hardware like the Sense HAT (see Chapter 7, *Physical computing with the Sense HAT*).

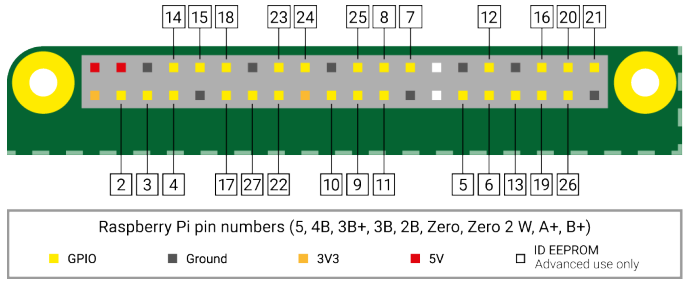


Figure 6-1 Raspberry Pi GPIO pinout

Raspberry Pi 400 has the same GPIO header with all the same pins, but it's turned upside-down compared to other Raspberry Pi models. **Figure 6-2** assumes you're looking at the GPIO header from the back of Raspberry Pi 400. Always double-check your wiring when connecting anything to Raspberry Pi 400's GPIO header — it's easy to forget, despite the 'Pin 40' and 'Pin 1' labels on the case!

Raspberry Pi Zero 2 W has a GPIO header too, but doesn't have header pins attached. If you want to do physical computing with Raspberry Pi Zero 2 W, or another model in the Raspberry Pi Zero family, you'll need to *solder* the pins into place using a soldering iron. If that sounds a little adventurous for now, check with an approved Raspberry Pi reseller for a Raspberry Pi Zero 2 WH with the header pins already soldered into place for you.



GPIO EXTENSIONS

It's perfectly possible to use Raspberry Pi 400's GPIO header as-is, but you may find it easier to use an extension. With an extension, the pins are brought around to the side of Raspberry Pi 400, meaning you can check and adjust your wiring without having to keep going around the back.

Compatible extensions include the Black HAT Hack3r range from **pimoroni.com** and the Pi T-Cobbler Plus from **adafruit.com**.

If you buy an extension, always check how it is wired. Some, like the Pi T-Cobbler Plus, change the layout of the GPIO pins. When in doubt, always use the manufacturer's instructions rather than the pin diagrams shown in this book.

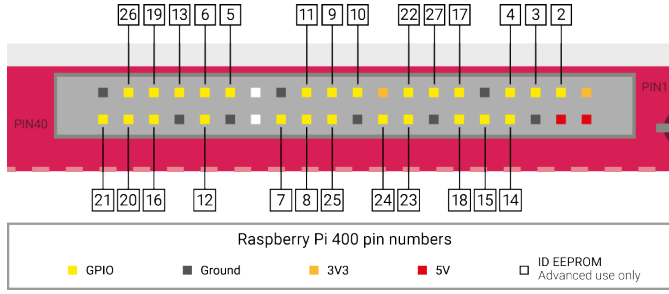


Figure 6-2 Raspberry Pi 400 GPIO pinout

There are several categories of pin types, each of which has a particular function:

3V3	3.3 volts power	A permanently-on source of 3.3V power, the same voltage Raspberry Pi runs at internally
5V5	5 volts power	A permanently-on source of 5V power, the same voltage as Raspberry Pi takes in at the USB C power connector
Ground (GND)	0 volts ground	A ground connection, used to complete a circuit connected to power source
GPIO XX	General-purpose input/output pin number 'XX'	The GPIO pins available for your programs, identified by a number from 2 to 27
ID EEPROM	Reserved special-purpose pins	Pins reserved for use with Hardware Attached on Top (HAT) and other accessories



WARNING!

Raspberry Pi's GPIO header is a fun and safe way to experiment with physical computing, but it must be treated with care. Be careful not to bend the pins when connecting and disconnecting hardware. Never connect two pins directly together, accidentally or deliberately, unless expressly told to do so in a project's instructions. If you do this you'll create a *short circuit* and, depending on the pins, can permanently damage your Raspberry Pi.

Electronic components

The GPIO header is only part of what you'll need to begin working with physical computing. You'll also need some electrical components, the devices you'll control from the GPIO header. There are thousands of different components available, but most GPIO projects are made using the following common parts.

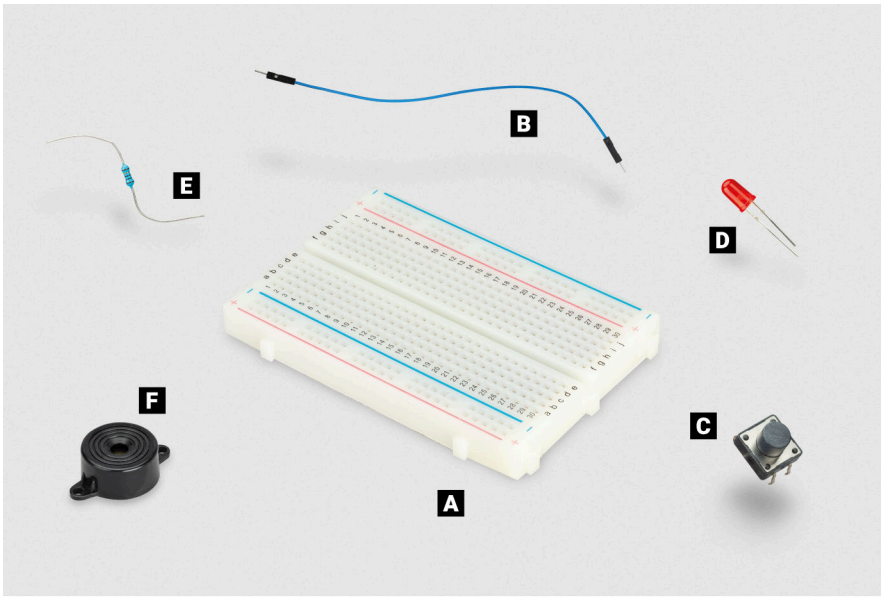


Figure 6-3 Common electronic components

A **Breadboard**

B **Jumper wire**

C **Momentary switch**

D **Light-emitting diode (LED)**

E **Resistor**

F **Piezoelectric buzzer**

A **breadboard** (**A**), also known as a *solderless breadboard*, can make physical computing projects considerably easier. Rather than having a bunch of separate components which need to be connected with wires, a breadboard lets you insert components and have them connected through metal tracks which are hidden beneath its surface. Many breadboards also include sections for power distribution, making it even easier to build your circuits. You don't need a breadboard to get started with physical computing, but it certainly helps.

Jumper wires (**B**), also known as *jumper leads*, connect components to your Raspberry Pi and, if you're not using a breadboard, to each other. They are available in three versions: male-to-female (M2F), which you'll need to connect your breadboard to the GPIO pins; female-to-female (F2F), which can be used to connect individual components together if you're not using a breadboard; and male-to-male (M2M), which is used to make connections from one part of a breadboard to another. Depending on your project, you may need all three types of jumper wire. If you're using a breadboard, you can usually get away with just M2F and M2M jumper wires.

A **momentary switch** (**C**) is the type of switch you might find on controllers for a game console. Commonly available with two or four legs — either type will work with a Raspberry Pi — the push-button is an input device: you can tell your program to watch out for it being pushed and then perform a task. Another common switch type is a *latching switch*: while a push-button is only active when you're holding it down, a latching switch — like a light switch — activates when you toggle it once, then stays active until you toggle it again.

A **light-emitting diode** (**LED**, **D**) is an *output device* which you can control directly from your program. An LED lights up when it's powered on, and you'll find them all over your house: from the small ones which let you know when you've left your washing machine switched on, to the large ones you might have lighting up your rooms. LEDs are available in a wide range of shapes, colours, and sizes, but not all are suitable for use with Raspberry Pi: avoid any which say they are designed for 5V or 12V power supplies.

Resistors (**E**) are components which control the flow of *electrical current* and are available in different values, measured using a unit called *ohms* (Ω). The higher the number of ohms, the more resistance is provided. For Raspberry Pi physical computing projects, their most common use is to protect LEDs from drawing too much current and damaging themselves or your Raspberry Pi; for this you'll want resistors rated at around 330Ω, though many electrical suppliers sell handy packs containing a number of different commonly used values to give you more flexibility.

A **piezoelectric buzzer** (**F**), usually just called a buzzer or a sounder, is another output device. Whereas an LED produces light, a buzzer produces a buzzing noise. Inside the buzzer's plastic housing are a pair of metal plates. When active,

these plates vibrate against each other to produce the buzzing sound. There are two types of buzzers: *active buzzers* and *passive buzzers*. Make sure to get an active buzzer, as these are the simplest to use.

Other common electrical components include motors, which need a special control board before they can be connected to Raspberry Pi, infrared sensors which detect movement, temperature and humidity sensors which can be used to predict the weather; and light-dependent resistors (LDRs) — input devices which operate like a reverse LED by detecting light.

Sellers all over the world provide components for physical computing with Raspberry Pi, either as individual parts or in kits which provide everything you need to get started. To find sellers, visit **rptl.io/products**, click on **Raspberry Pi 5**, and click the **Buy now** button to see a list of Raspberry Pi partner online stores and approved resellers for your country or region.

To complete the projects in this chapter, you should have at least:

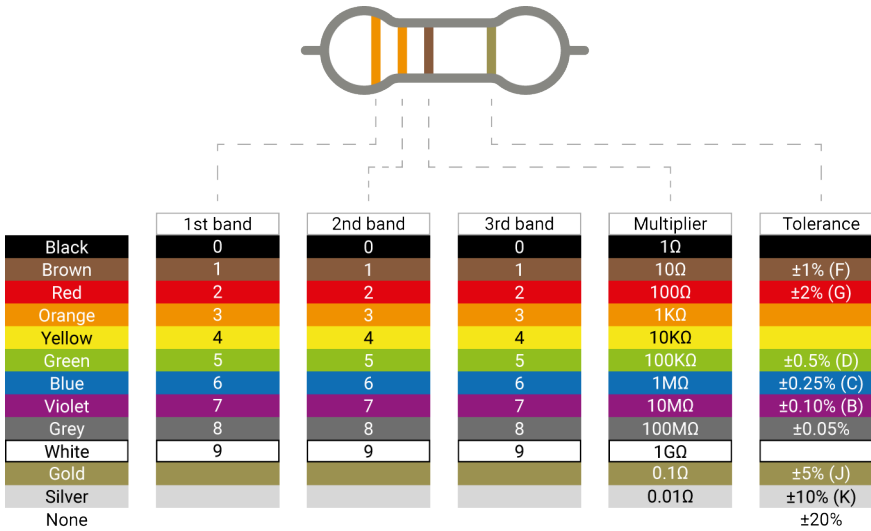
- 3 × LEDs: red, green, and yellow or amber
- 2 × push-button switches
- 1 × active buzzer
- Male-to-female (M2F) and female-to-female (F2F) jumper wires
- Optionally, a breadboard and male-to-male (M2M) jumper wires

Reading resistor colour codes

Resistors come in a wide range of values, from zero-resistance versions which are effectively just pieces of wire to high-resistance versions the size of your leg. Very few of these resistors have their values printed on them in numbers. Instead, they use a special code (**Figure 6-4**) printed as coloured stripes or bands around the body of the resistor.

To read the value of a resistor, position it so the group of bands is to the left and the lone band is to the right. Starting from the first band, look its colour up in the '1st/2nd Band' column of the table to get the first and second digits. This example has two orange bands, which both mean a value of '3' for a total of '33'. If your resistor has four grouped bands instead of three, note down the value of the third band too (for five/six-band resistors, see **rptl.io/5-6-band**).

Moving onto the last grouped band — the third or fourth — look its colour up in the 'Multiplier' column. This tells you what you need to multiply your current number by to get the actual value of the resistor. This example has a brown



	1st band	2nd band	3rd band	Multiplier	Tolerance
Black	0	0	0	1Ω	
Brown	1	1	1	10Ω	±1% (F)
Red	2	2	2	100Ω	±2% (G)
Orange	3	3	3	1KΩ	
Yellow	4	4	4	10KΩ	
Green	5	5	5	100KΩ	±0.5% (D)
Blue	6	6	6	1MΩ	±0.25% (C)
Violet	7	7	7	10MΩ	±0.10% (B)
Grey	8	8	8	100MΩ	±0.05%
White	9	9	9	1GΩ	
Gold				0.1Ω	±5% (J)
Silver				0.01Ω	±10% (K)
None					±20%

Figure 6-4 Resistor colour codes

band, which means '$\times 10^{1}$'. That may look confusing, but it's simply *scientific notation*: '$\times 10^{1}$' simply means 'add one zero to the end of your number'. If it were blue, for $\times 10^{6}$', it would mean 'add six zeroes to the end of your number'.

Taking 33 from the orange bands, plus the added zero from the brown band, gives us 330 — which is the value of the resistor, measured in ohms. The final band, the one on the right, is the *tolerance* of the resistor. This is simply how close to its rated value it is likely to be. Cheaper resistors might have a silver band, indicating a tolerance 10% higher or lower than its rating, or no last band at all, indicating a tolerance 20% higher or lower. The most expensive resistors have a grey band, indicating a tolerance within 0.05% of its rating. For hobbyist projects, accuracy isn't that important: any tolerance will usually work fine.

If your resistor value goes above 1000 ohms (1000Ω), it is usually rated in kilohms (kΩ); if it goes above a million ohms, those are megohms (MΩ). A 2200Ω resistor would be written as 2.2kΩ; a 2,200,000Ω resistor would be written as 2.2MΩ.

CAN YOU WORK IT OUT?

What colour bands would a 100Ω resistor have? What colour bands would a 2.2MΩ resistor have? If you wanted to find the cheapest resistors, what colour tolerance band would you look for?

Your first physical computing program: Hello, LED!

Just as printing 'Hello, World' to the screen is a fantastic first step in learning a programming language, making an LED light up is the traditional introduction to learning physical computing. For this project, you'll need an LED and a 330 ohm (330Ω) resistor, or as close to 330Ω as you can find, plus female-to-female (F2F) jumper wires.

RESISTANCE IS VITAL

The resistor is a vital component in this circuit: it protects your Raspberry Pi and the LED by limiting the amount of electrical current the LED can draw. Without it, the LED can pull too much current and burn itself — or your Raspberry Pi — out. When used like this, the resistor is known as a *current-limiting resistor*. The exact value of the resistor you need depends on the LED you're using, but 330Ω works for most common LEDs. The higher the value, the dimmer the LED; the lower the value, the brighter the LED.

Never connect an LED to a Raspberry Pi without a current-limiting resistor, unless you know the LED has a built-in resistor of appropriate value.

Start by checking that your LED works. Turn your Raspberry Pi so the GPIO header is in two vertical strips to the right-hand side. Connect one end of your 330Ω resistor to the first 3.3V pin (labelled 3V3 in **Figure 6-5**) using a female-to-female jumper wire, then connect the other end to the long leg — the positive lead, or anode — of your LED with another female-to-female jumper wire. Take a last female-to-female jumper wire, and connect the short leg — the negative lead, or cathode — of your LED to the first ground pin (labelled GND in **Figure 6-5**).

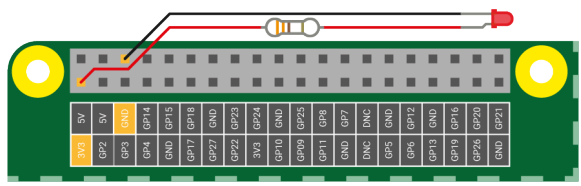


Figure 6-5 Wire your LED to these pins — don't forget the resistor!

While your Raspberry Pi is on, the LED should light up. If it doesn't, double-check your circuit: make sure you haven't used too high a resistor value, that all the wires are properly connected, and that you've definitely picked the right GPIO pins to match the diagram. Also check the legs of the LED, as LEDs will only work one way around: make sure the longer leg is connected to the positive side of the circuit, and the shorter leg to the negative.

Once your LED is working, it's time to program it. Disconnect the jumper wire from the 3.3V pin (labelled 3V3 in **Figure 6-6**) and connect it to pin 25 of the GPIO (labelled GP25 in **Figure 6-6**). The LED will switch off, but don't worry — that's normal.

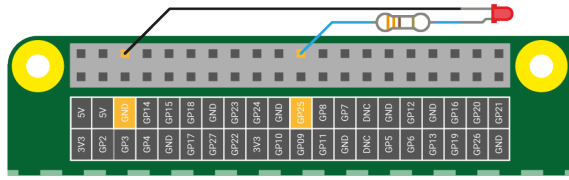


Figure 6-6 Disconnect the wire from 3V3 and connect it to pin 25 of the GPIO

You are now ready to create a Scratch or Python program to turn your LED on and off.

CODING KNOWLEDGE

The projects in this chapter need you to be comfortable with using Scratch 3 and the Thonny Python integrated development environment (IDE). If you haven't already done so, turn to Chapter 4, *Programming with Scratch 3*, and Chapter 5, *Programming with Python*, and work through those projects first.

If you don't have Scratch 3 installed already, follow the instructions in "The Recommended Software tool" on page 43 to install it.

LED control in Scratch

Load Scratch 3 and click on the **Add Extension** icon . Scroll down to find the **Raspberry Pi GPIO** extension (**Figure 6-7**), then click on it. This loads the blocks you need to control Raspberry Pi's GPIO header from Scratch 3. You'll see the new blocks appear in the blocks palette; when you need them, they'll be available in the Raspberry Pi GPIO category.

WARNING!

At the time of writing, Scratch 3 had not received an update to support Raspberry Pi 5. If you're having problems, check for an update (see "Software updates" on page 49). If no update is available, check this book's GitHub repository at **rptl.io/bg-resources** for status information regarding a forthcoming update.

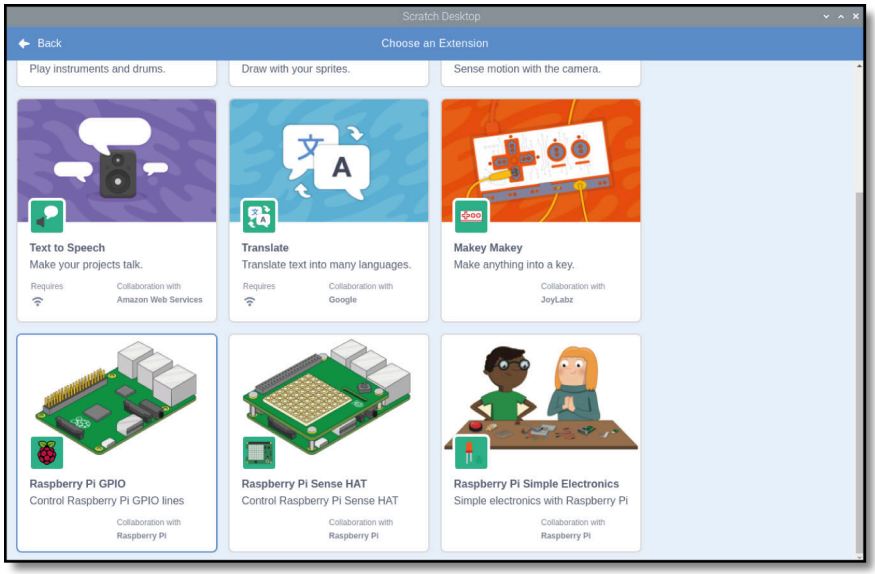
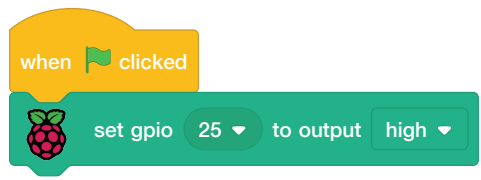
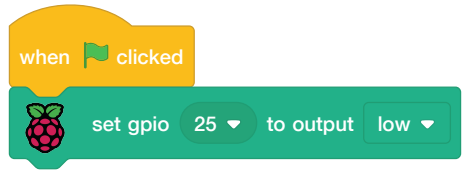


Figure 6-7 Add the Raspberry Pi GPIO extension to Scratch 3

Start by dragging a when clicked **Events** block onto the code area, then place a green set gpio to output high block underneath it. You'll need to choose the number of the pin you're using: click on the small arrow to open the drop-down selection and click on **25** to tell Scratch you're controlling pin 25 of the GPIO.



Click the green flag to run your program. You'll see your LED light up. Congratulations: you've programmed your first physical computing project! Click the red octagon to stop your program: notice how the LED stays lit? That's because your program only ever told your Raspberry Pi to turn the LED on — that's what the `output high` part of your set gpio 25 to output high block means. To turn it off again, click on the down arrow at the end of the block and choose '**low**' from the list.



Click the green flag again, and this time your program will turn the LED off. To make things more interesting, add an orange forever **Control** block and a couple of orange wait 1 seconds blocks to create a program to flash the LED on and off every second.



Click the green flag and watch your LED: it will turn on for a second, turn off for a second, turn on for a second, and keep repeating that pattern until you click the red octagon to stop it. See what happens when you click the octagon while the LED is in its on or off states.

CHALLENGE: CAN YOU ALTER IT?

How would you change the program to make the LED stay on for longer? What about staying off for longer? What's the smallest delay you can use while still seeing the LED switch on and off?

LED control in Python

Load Thonny from the **Programming** section of the Raspberry Pi menu, then click the **New** button to start a new project and **Save** to save it as **Hello LED.py**. To use the GPIO pins from Python, you need a library called GPIO Zero. For this project, you only need the part of the library for working with LEDs. Import just this section of the library by typing the following into the Python shell area:

```
from gpiozero import LED
```

Next, you need to let GPIO Zero know which GPIO pin the LED is connected to. Type the following:

```
led = LED(25)
```

Together, these two lines give Python the ability to control LEDs connected to your Raspberry Pi's GPIO pins and tell it which pin — or pins, if you have more than one LED in your circuit — to control. To control the LED and switch it on, type the following:

```
led.on()
```

To switch the LED off again, type:

```
led.off()
```

Congratulations, you are now controlling your Raspberry Pi's GPIO pins in Python! Try typing those two instructions again. If the LED is already off, `led.off()` won't do anything; the same is true if the LED is already on and you type `led.on()`.

To write your own program, type the following into the script area:

```
from gpiozero import LED
from time import sleep
led = LED(25)
while True:
    led.on()
    sleep(1)
    led.off()
    sleep(1)
```

This program imports the `LED` function from the gpiozero (GPIO Zero) library and the `sleep` function from the time library, then constructs an infinite loop to turn the LED on for a second, turn it off for a second, and repeat. Click the **Run** button to see it in action: your LED will begin to flash.

As with the Scratch program, make a note of the program's behaviour when you click the **Stop** button while the LED is on, and observe what happens if you click while the LED is off.

CHALLENGE: LONGER LIGHT-UP

How would you change the program to make the LED stay on for longer? What about staying off for longer? What's the smallest delay you can use while still seeing the LED switch on and off?

Using a breadboard

The next projects in this chapter will be much easier to complete if you're using a breadboard (**Figure 6-8**) to hold the components and make the electrical connections.

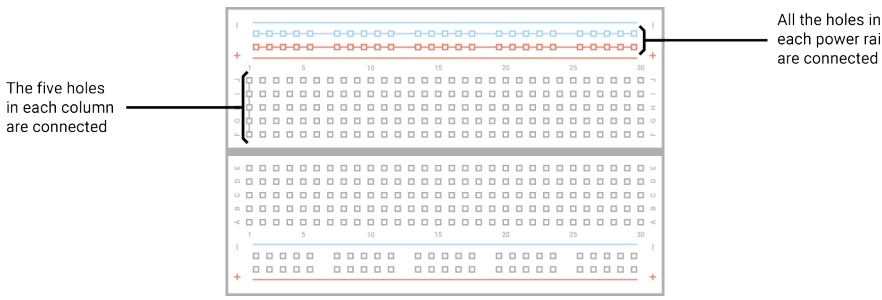


Figure 6-8 A solderless breadboard

A breadboard is covered with holes which are spaced 2.54mm apart to match most components. Under these holes are metal strips (terminals) which act like the jumper wires you've been using until now. These run in columns on the board, with most boards having a gap down the middle to split them in two halves. Many breadboards also have letters going up the left side and numbers on the top and bottom. These allow you to find a particular hole: A1 is the bottom-left corner, B1 is the hole just above it, while B2 is one hole to the right. A1 is connected to B1 by the hidden metal strips, but no number hole is ever connected to a different number hole unless you add a jumper wire yourself.

Larger breadboards also have strips of holes along the top and bottom, typically marked with red and black or red and blue stripes. These are the *power rails*, and are designed to make wiring easier: you can connect a single wire from your Raspberry Pi's ground pin to one of the power rails — typically marked with a blue or black stripe and a minus symbol — to provide a *common ground* for lots of components on the breadboard, and you can do the same if your circuit needs 3.3V or 5V power.

Adding electronic components to a breadboard is simple: just line their leads (the sticky-out metal parts) up with the holes and gently push until the component is in place. For connections you need to make beyond those the breadboard makes for you, you can use male-to-male (M2M) jumper wires; for connections from the breadboard to the Raspberry Pi, use male-to-female (M2F) jumper wires.



WARNING

Never try to cram more than one component lead or jumper wire into a single hole on the breadboard. Remember: holes are connected in columns, aside from the split in the middle, so a component lead in A1 is electrically connected to anything you add to B1, C1, D1, and E1.

Next steps: reading a button

Outputs like LEDs are one thing, but the 'input/output' part of 'GPIO' means you can use pins as inputs too. For this project, you'll need a breadboard, male-to-male (M2M) and male-to-female (M2F) jumper wires, and a push-button switch. If you don't have a breadboard you can use female-to-female (F2F) jumper wires, but the button will be much harder to press without accidentally breaking the circuit.

Start by adding the push-button to your breadboard. If your push-button has only two legs, make sure they're in different numbered holes of the breadboard; if it has four legs, turn it so the sides the legs come out from are aligned as shown in **Figure 6-9**. Connect the ground rail of your breadboard to a ground pin of your Raspberry Pi (marked GND) with a male-to-female jumper wire, then connect one leg of your push-button to the ground rail with a male-to-male jumper wire. Finally, connect the other leg — the one on the same side as the leg you just connected, if using a four-leg switch — to the GPIO 2 pin (marked GP2) of your Raspberry Pi with a male-to-female jumper wire.

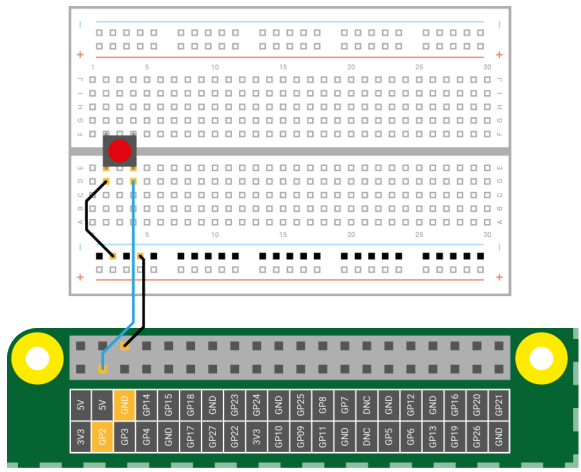
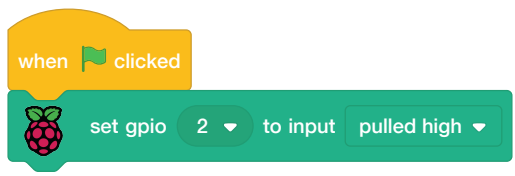


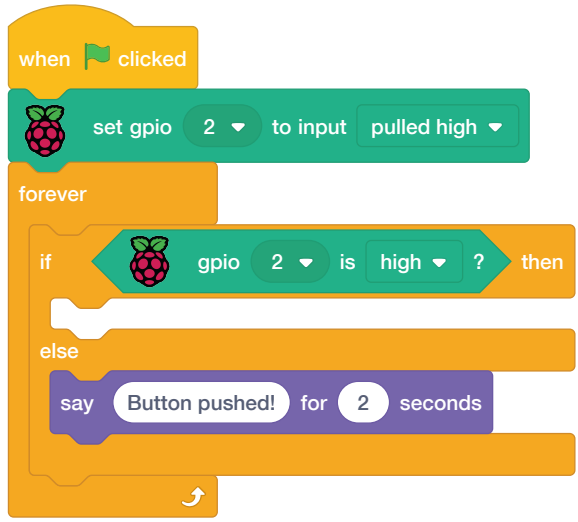
Figure 6-9 Wiring a push-button to the GPIO pins

Reading a button in Scratch

Start a new Scratch program and drag an orange when clicked block onto the code area. Connect a green set gpio to input pulled high block, and select the number **2** from the drop-down to match the GPIO pin you used for the push-button.



If you click the green flag now, nothing will happen. That's because you've told Scratch to use the pin as an input, but not what to do with that input. Drag an orange forever block to the end of your sequence, then drag an orange if then else block inside it. Find the green gpio is high? block, drag it into the diamond-shaped space in the if then part of the orange block, and use the drop-down to select the number **2** to tell it which GPIO pin to check. Drag a violet say Hello! for 2 seconds block into the else part of the orange block, and edit it to say '`Button pushed!`'. Leave the space between if and else in the orange block empty for now.



There's a lot going on here. Start by testing your program: click the green flag, then push the button on your breadboard. Your sprite should tell you that the button has been pushed. Congratulations: you've successfully read an input from the GPIO pin!

Because the space between `if` and `else` in the orange block is empty for now, nothing happens when `gpio 2 is high?` evaluates to true. The code that runs when the button is pushed is in the `else` part of the block. This seems confusing: surely pressing the button makes it go high? In fact, it's the opposite: Raspberry Pi's GPIO pins are normally high, or on, when set as an input, and pushing the button pulls them down to low.

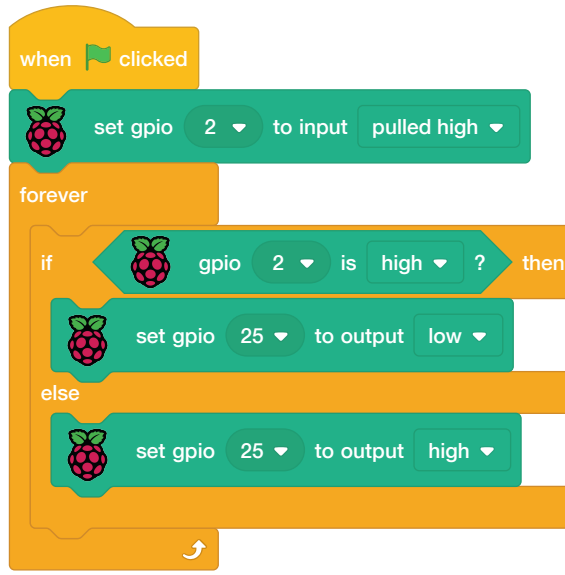
Look at your circuit again: see how the button is connected to the GPIO 2 pin, which is providing the positive part of the circuit, and the ground pin. When the button is pushed, the voltage on the GPIO pin is pulled low through the ground pin, and your Scratch program stops running the code (if any) in your `if gpio 2 is high? then` block and instead runs the code in the `else` part of the block.

If that all sounds perplexing, just remember this: a button on a Raspberry Pi GPIO pin is considered pushed when the pin goes low, not when it goes high!

To extend your program further, add the LED and resistor back into the circuit. Remember to connect the resistor to pin 25 of the GPIO and the long leg of the LED, and the shorter leg of the LED to the ground rail on your breadboard.

Drag the `say Button pushed! for 2 seconds` block off the code area to the block palette to delete it, then replace it with a green `set gpio 25 to output high` block, making sure to change the GPIO number using the drop-down arrow.

Drag a green set gpio 25 to output low block — remembering to change the GPIO number from its default — into the currently empty if gpio 2 is high? then part of the block.



Click the green flag and push the button. The LED will light up as long as you're holding the button down; let go, and it will go dark again. Congratulations: you're controlling one GPIO pin based on an input from another!

CHALLENGE: MAKE IT STAY LIT

How would you change the program to make the LED stay on for a few seconds, even after you let go of the button? What would you need to change to have the LED on while you're not pressing the button and off while you are?

Reading a button in Python

Click the **New** button in Thonny to start a new project, and the **Save** button to save it as **Button Input.py**. Using a GPIO pin as an input for a button is very similar to using a pin as an output for an LED, but you need to import a different part of the GPIO Zero library. Type the following into the script area:

```
from gpiozero import Button
button = Button(2)
```

To have code run when the button is pressed, GPIO Zero provides the `wait_for_press` function. Type the following:

```
button.wait_for_press()
print("You pushed me!")
```

Click the **Run** button, then press the push-button switch. Your message will print to the Python shell at the bottom of the Thonny window. Congratulations: you've successfully read an input from the GPIO pin!

If you want to try your program again, you'll need to click the **Run** button again. Because there's no loop in the program, it quits as soon as it finishes printing the message to the shell.

To extend your program further, add the LED and resistor back into the circuit if you haven't already done so: remember to connect the resistor to pin 25 of the GPIO and the long leg of the LED, and the shorter leg of the LED to the ground rail on your breadboard.

To control an LED as well as read a button, you'll need to import both the `Button` and `LED` functions from the GPIO Zero library. You'll also need the `sleep` function from the `time` library. Go back to the top of your program, and type in the following as the new first two lines:

```
from gpiozero import LED
from time import sleep
```

Below the line `button = Button(2)`, type:

```
led = LED(25)
```

Delete the line `print("You pushed me!")` and replace it with:

```
led.on()
sleep(3)
led.off()
```

Your finished program should look like this:

```
from gpiozero import LED
from time import sleep
from gpiozero import Button

button = Button(2)
led = LED(25)
button.wait_for_press()
led.on()
sleep(3)
led.off()
```

Click the **Run** button, then press the push-button switch: the LED will come on for three seconds, then turn off again and the program will exit. Congratulations: you can control an LED using a button input in Python!

CHALLENGE: ADD A LOOP



How would you add a loop to make the program repeat instead of exiting after one button press? What would you need to change to make the LED turn on while you're not pressing the button, and off while you are?

Make some noise: controlling a buzzer

LEDs are a great output device, but not much use if you're not looking at them. The solution: buzzers, which make a noise audible anywhere in the room. For this project you'll need a breadboard, male-to-female (M2F) jumper wires, and an active buzzer. If you don't have a breadboard, you can connect the buzzer using female-to-female (F2F) jumper wires instead.

An active buzzer can be treated exactly like an LED in terms of circuitry and programming. Repeat the circuit you made for the LED, but replace the LED with the active buzzer and leave the resistor out, as the buzzer will need more current to work. Connect one leg of the buzzer to pin 15 of the GPIO (labelled GP15 in **Figure 6-10**) and the other to the ground pin (labelled GND in the diagram) using your breadboard and male-to-female jumper wires.

If your buzzer has three legs, make sure the leg marked with a minus symbol (-) is connected to the ground pin, and the leg marked with 'S' or 'SIGNAL' is connected to pin 15, then connect the remaining leg — usually the middle leg — to the 3.3V pin (labelled 3V3.)

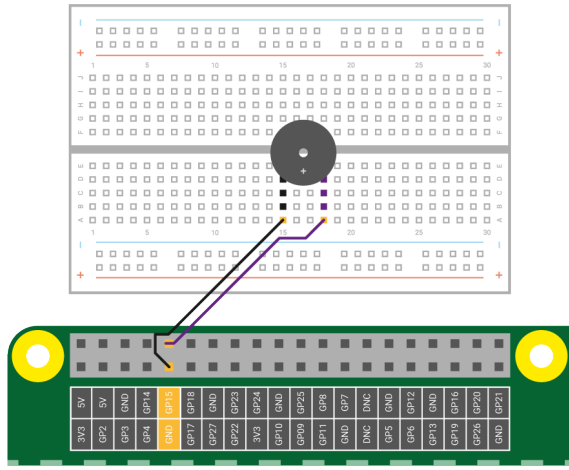


Figure 6-10 Connecting a buzzer to the GPIO pins

Controlling a buzzer in Scratch

Recreate the program you used to make the LED flash — or load it, if you saved it earlier. Use the drop-down in the green set gpio to output high blocks to select number **15**, so Scratch is controlling the correct GPIO pin.



Click the green flag, and your buzzer will begin to buzz: one second on, one second off. If you only hear the buzzer clicking once a second, your buzzer is passive, not active. While an active buzzer generates a rapidly changing signal, known as an *oscillation*, to make the metal plates vibrate, a passive buzzer needs to receive an external oscillation signal rather than producing one itself.

When you turn it on using Scratch, the plates only move once and stop, making the 'click' sound until the next time your program switches the pin on or off.

Click the red octagon to stop your buzzer, but do so when it's not making a sound, otherwise the buzzer will buzz until you run your program again!

CHALLENGE: CHANGE THE BUZZ

How could you change the program to make the buzzer sound for a shorter time? Can you build a circuit so the buzzer is controlled by a button?

Controlling a buzzer in Python

Controlling an active buzzer through the GPIO Zero library is almost identical to controlling an LED, in that it has on and off states. You need a different, function, though: `Buzzer`. Start a new project in Thonny and save it as **Buzzer.py**, then type the following:

```
from gpiozero import Buzzer
from time import sleep
```

As with LEDs, GPIO Zero needs to know which pin your buzzer is connected to in order to control it. Type the following:

```
buzzer = Buzzer(15)
```

From here, your program is almost identical to the one you wrote to control the LED; the only difference (apart from a different GPIO pin number) is you're using `buzzer` in place of `led`. Type the following:

```
while True:
    buzzer.on()
    sleep(1)
    buzzer.off()
    sleep(1)
```

Click the **Run** button and your buzzer will begin to buzz: one second on, and one second off. If you are using a passive buzzer rather than an active buzzer, you'll only hear a brief click every second instead of a continuous buzz.

Click the **Stop** button to exit the program, but make sure the buzzer isn't making a sound at the time, or it will continue to buzz until you run your program again!

Scratch project: Traffic Lights

Now that you know how to use buttons, buzzers, and LEDs as inputs and outputs, you're ready to build an example of real-world computing: traffic lights, complete with a button you can press to cross the road. For this project, you'll need a breadboard; red, yellow, and green LEDs; three 330Ω resistors; a buzzer; a push-button switch; and a selection of male-to-male (M2M) and male-to-female (M2F) jumper wires.

Start by building the circuit (**Figure 6-11**), connecting the buzzer to pin 15 of the GPIO (labelled GP15 in **Figure 6-11**), the red LED to pin 25 (labelled GP25), the yellow LED to pin 8 (GP8), the green LED to pin 7 (GP7), and the switch to pin 2 (GP2). Remember to connect the 330Ω resistors between the GPIO pins and the long legs of the LEDs, and connect the second legs on all your components to the ground rail of your breadboard. Finally, connect the ground rail to a ground pin (labelled GND) on Raspberry Pi to complete the circuit.

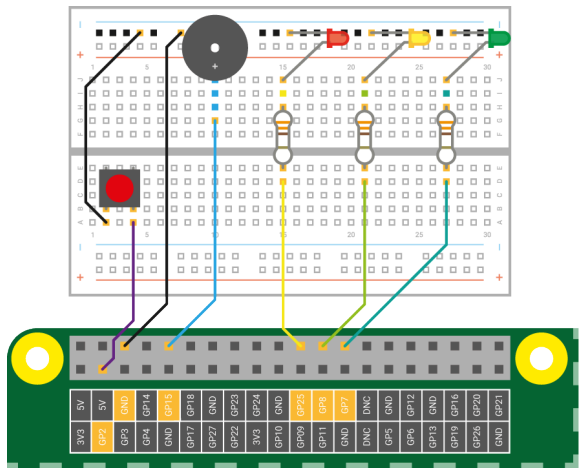
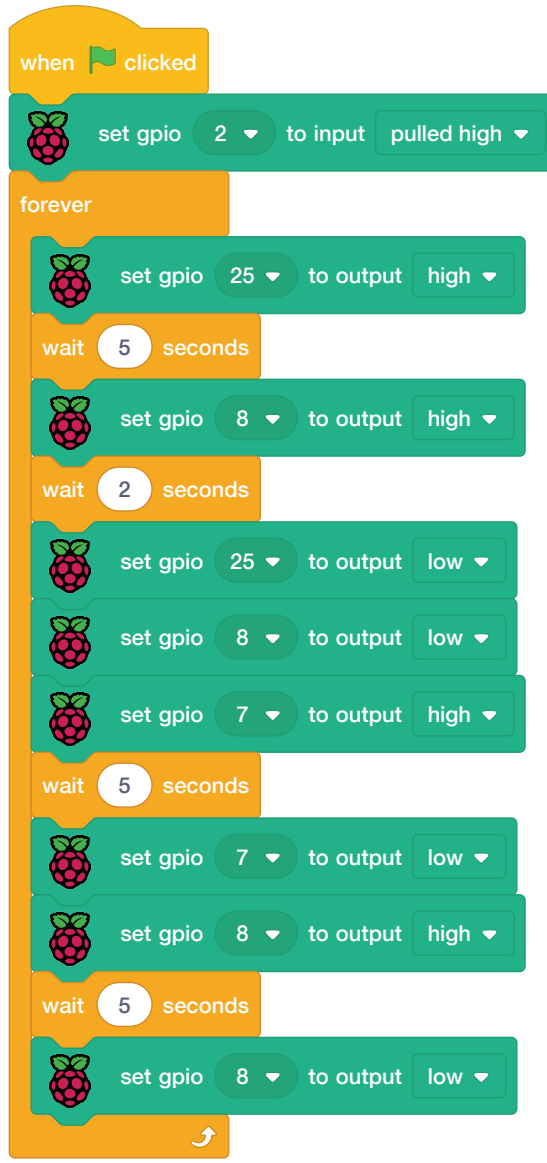


Figure 6-11 Wiring diagram for the Traffic Lights project

Start a new Scratch 3 project, then drag a when clicked block onto the code area. Next, you'll need to tell Scratch that pin 2 of the GPIO, which is connected to the push-button switch, is an input rather than an output. Drag a green set gpio to input pulled high block from the **Raspberry Pi GPIO** category of the blocks palette under your when clicked block. Click on the down arrow next to **0** and select **2** from the drop-down list.

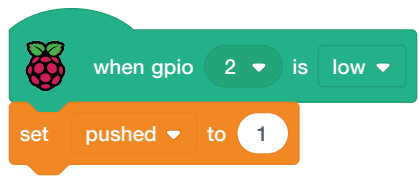
Next, you need to create your traffic light sequence. Drag an orange forever block into your program, then fill it with blocks to turn the traffic light LEDs on and off in a pattern. Remember which GPIO pins have which component attached: when you're using pin 25 you're using the red LED, pin 8 is the yellow LED, and pin 7 is the green LED.



Click the green flag and watch your LEDs: first the red will light, then both the red and yellow, then the green, then the yellow, and finally the sequence repeats, starting with the red light again. This pattern matches the one used by traffic lights in the UK; you can edit the sequence to match patterns in other countries, if you wish.

To simulate a pedestrian crossing, you need your program to watch for the button being pressed. Click the red octagon to stop your program if it's currently running. Drag an orange `if then else` block onto your script area and connect it so it's directly beneath your `forever` block, with your traffic light sequence in the `if then` section. Leave the diamond-shaped gap empty for now.

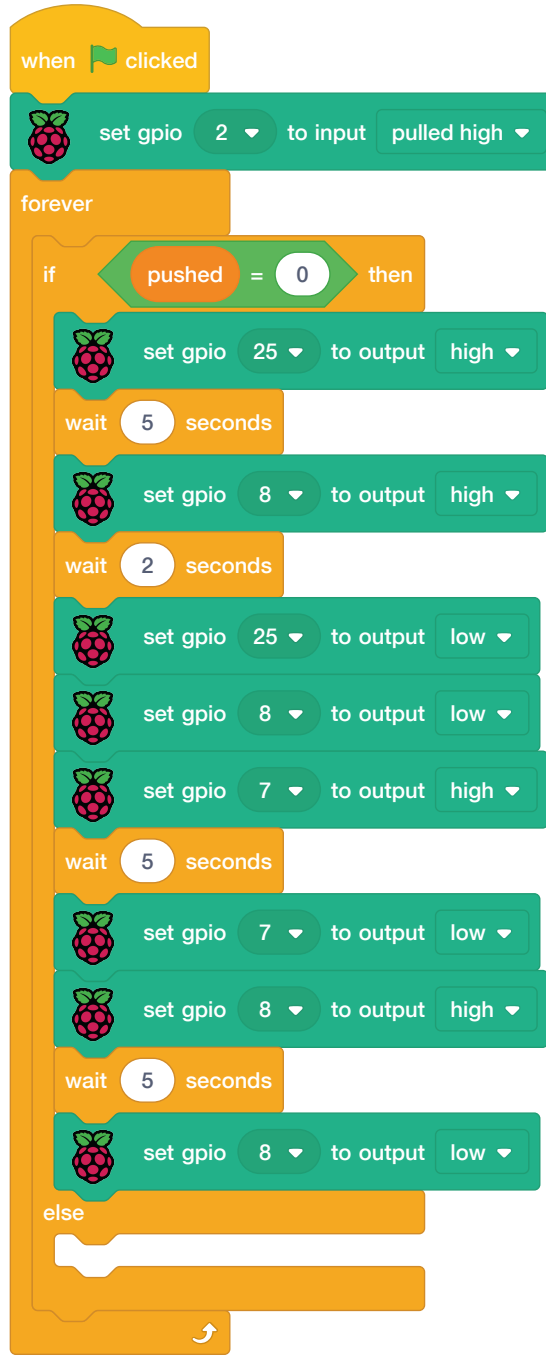
A real pedestrian crossing doesn't change the light to red as soon as the button is pushed, but instead waits for the next red light in the sequence. To build that into your own program, drag a green `when gpio is low` block onto the code area and select `2` from its drop-down list. Create a new variable named **`pushed`**, then drag an orange `set pushed to 1` block underneath it.



This block stack watches out for the button being pushed, then sets the variable **`pushed`** to 1. Setting a variable this way lets you store the fact the button has been pushed, even though you're not going to act on it right away.

Go back to your original block stack and find the `if then` block. Drag a green diamond-shaped `=` operator block into the `if then` block's diamond blank, then drag a dark orange `pushed` reporter block into the first blank space. Type **`0`** over the **`50`** on the right-hand side of the block.

Click the green flag and watch the traffic lights go through their sequence. When you're ready, press the push-button switch: at first it will look like nothing is happening, but once the sequence reaches its end — with just the yellow LED lit — the traffic lights will go off and stay off, thanks to your **`pushed`** variable.


when clicked
set gpio 2 to input pulled high
forever
if pushed = 0 then
set gpio 25 to output high
wait 5 seconds
set gpio 8 to output high
wait 2 seconds
set gpio 25 to output low
set gpio 8 to output low
set gpio 7 to output high
wait 5 seconds
set gpio 7 to output low
set gpio 8 to output high
wait 5 seconds
set gpio 8 to output low
else

All that's left is to make your pedestrian crossing button actually do something other than turn the lights off. In the main block stack, find the else block and drag a set gpio 25 to output high block into it — remember to change the default GPIO pin number to match the pin your red LED is connected to.

Beneath that, still in the else block, create a pattern for the buzzer: drag an orange repeat 10 block, then fill it with a green set gpio 15 to output high, an orange wait 0.2 seconds, a green set gpio 15 to output low, and another orange wait 0.2 seconds block, changing the GPIO pin values to match the pin for the buzzer.

Finally, beneath the bottom of your repeat 10 block but still in the else block, add a green set gpio 25 to output low block and a dark orange set pushed to 0 block. The last block resets the variable that stores the button press, so the buzzer sequence doesn't just repeat forever.

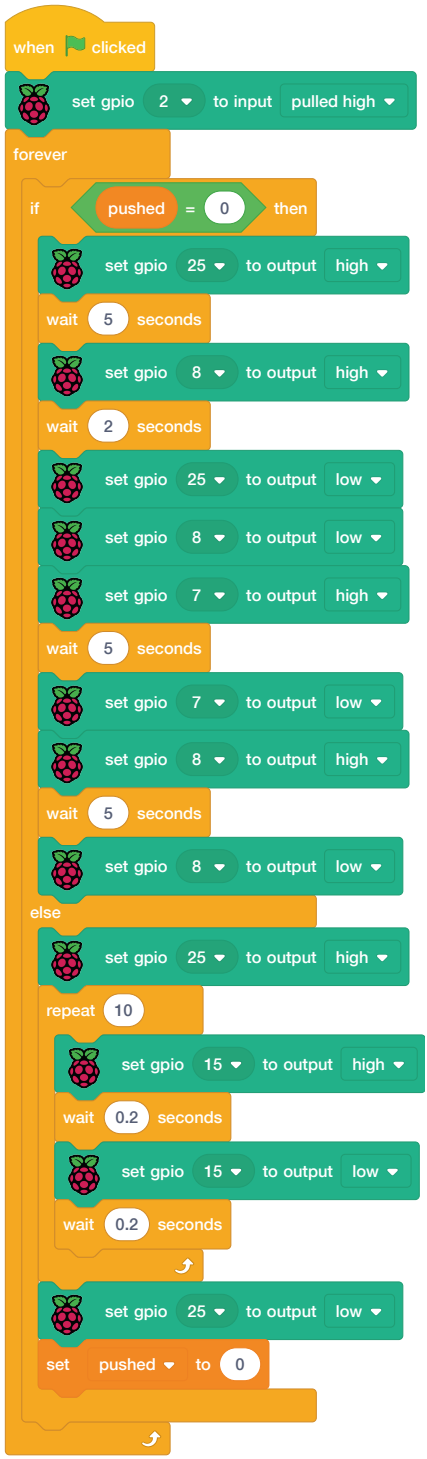
Click the green flag, then push the switch on your breadboard. After the sequence finishes, you'll see the red light go on and hear the buzzer sound to let pedestrians know it's safe to cross. After a couple of seconds, the buzzer will stop and the traffic light sequence will start again and continue until the next time you press the button.

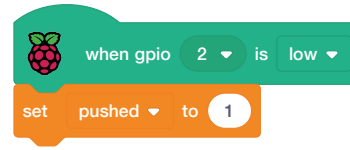
Congratulations: you have programmed your own fully functional set of traffic lights, complete with pedestrian crossing!



CHALLENGE: CAN YOU IMPROVE IT?

Can you change the program to give the pedestrian longer to cross? Can you find information about other countries' traffic light patterns and reprogram your lights to match? How could you make the LEDs less bright?


when clicked
set gpio 2 to input pulled high
forever
if pushed = 0 then
set gpio 25 to output high
wait 5 seconds
set gpio 8 to output high
wait 2 seconds
set gpio 25 to output low
set gpio 8 to output low
set gpio 7 to output high
wait 5 seconds
set gpio 7 to output low
set gpio 8 to output high
wait 5 seconds
set gpio 8 to output low
else
set gpio 25 to output high
repeat 10
set gpio 15 to output high
wait 0.2 seconds
set gpio 15 to output low
wait 0.2 seconds
set gpio 25 to output low
set pushed to 0


when gpio 2 is low
set pushed to 1

Python project: Quick Reaction Game

Now that you know how to use buttons and LEDs as inputs and outputs, you're ready to build an example of real-world computing: a two-player quick-reaction game, designed to see who has the fastest reaction time! For this project you'll need a breadboard, an LED, a 330Ω resistor, two push-button switches, some male-to-female (M2F) jumper wires, and some male-to-male (M2M) jumper wires.

Start by building the circuit (**Figure 6-12**): connect the first switch at the left-hand side of your breadboard to pin 14 of the GPIO (labelled GP14 in **Figure 6-12**). The second switch at the right-hand side of your breadboard goes to pin 15 (labelled GP15); the LED's longer leg connects to the 330Ω resistor, which then connects to pin 4 of the GPIO (labelled GP4). The second leg on all your components connects to your breadboard's ground rail. Finally, connect the ground rail to your Raspberry Pi's ground pin (labelled GND).

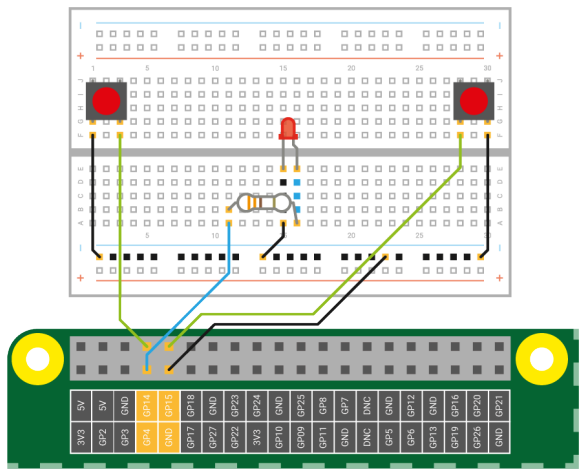


Figure 6-12 Wiring diagram for the Quick Reaction Game

Start a new project in Thonny and save it as **Reaction Game.py**. You're going to be using the `LED` and `button` functions from the GPIO Zero library, and the `sleep` function from the time library. Rather than importing the two GPIO Zero functions on two separate lines, though, you can save time and import them together using a comma symbol (`,`) to separate them. Type the following in the script area:

```
from gpiozero import LED, Button
from time import sleep
```

As before, you'll need to tell GPIO Zero which pins the two buttons and the LED are connected to. Type the following:

```
led = LED(4)
right_button = Button(15)
left_button = Button(14)
```

Now add instructions to turn the LED on and off, so you can check it's working correctly:

```
led.on()
sleep(5)
led.off()
```

Click the **Run** button. The LED will turn on for five seconds, then turn off. Then the program will quit. For the purposes of a reaction game, having the LED go off after exactly five seconds every time is a bit predictable. Add the following below the line `from time import sleep`:

```
from random import uniform
```

The `random` library, as its name suggests, lets you generate random numbers (in this case with a uniform distribution — see **rptl.io/uniform-dist**). Find the line `sleep(5)` and change it to read:

```
sleep(uniform(5, 10))
```

Click the **Run** button again: this time the LED will stay lit for a random number of seconds between 5 and 10. Count to see how long it takes for the LED to go off, then click the **Run** button a few more times. You'll see the time is different for each run, making the program less predictable.

To turn the buttons into triggers for each player, you need to add a function. Go to the very bottom of your program and type the following:

```
def pressed(button):
    print(str(button.pin.number) + " won the game")
```

Remember that Python uses indentation to understand which lines are part of your function; Thonny will automatically indent the second line for you.

Finally, add the following two lines to detect when players press the buttons — remember that they must not be indented, or else Python will treat them as part of your function.

```
right_button.when_pressed = pressed
left_button.when_pressed = pressed
```

Run your program, and this time try to press one of the two buttons as soon as the LED goes out. You'll see a message showing which button was pressed first printed in the Python shell at the bottom of the Thonny window. Unfortunately, you'll see the same message each time either button is pushed, using the pin number rather than a friendly name for the button.

To fix that, start by asking the players for their names. Underneath the line `from random import uniform`, type the following:

```
left_name = input("Left player name is ")
right_name = input("Right player name is ")
```

Go back to your function and replace the line `print(str(button.pin.number) + " won the game")` with:

```
    if button.pin.number == 14:
        print (left_name + " won the game")
    else:
        print(right_name + " won the game")
```

Click the **Run** button, then type the names of both players into the Python shell area. When you press the button this time — as quickly as you can after the LED goes out — you'll see the player's name displayed instead of the pin number.

To fix the problem of all button presses being considered a win, you'll need to add a new function from the sys — short for *system* — library: `exit`. Under the last `import` line, type the following:

```
from os import _exit
```

Then at the end of your function, under the line `print(right_name + " won the game")`, type the following:

```
    _exit(0)
```

The indentation is important here: `_exit(0)` should be indented by four spaces, lining up with `else` two lines above it, and `if` two lines above that. This instruction tells Python to stop the program after the first button is pressed, meaning the player who presses the button too late doesn't get a reward for losing!

Your finished program should look like this:

```
from gpiozero import LED, Button
from time import sleep
from random import uniform
from os import _exit

left_name = input("Left player name is ")
right_name = input ("Right player name is ")
led = LED(4)
right_button = Button(15)
left_button = Button(14)

led.on()
sleep(uniform(5, 10))
led.off()

def pressed(button):
    if button.pin.number == 14:
        print(left_name + " won the game")
    else:
        print(right_name + " won the game")
    _exit(0)

right_button.when_pressed = pressed
left_button.when_pressed = pressed
```

Click the **Run** button, enter the players' names, wait for the LED to go off, press a button, and you'll see the name of the winning player. You'll also see a message from Python itself: `Process ended with exit code 0.` This means that Python received your `_exit(0)` command and halted the program, and is ready for its next instructions. If you want to play again, click the **Run** button once more!

Congratulations: you've made your own physical game!

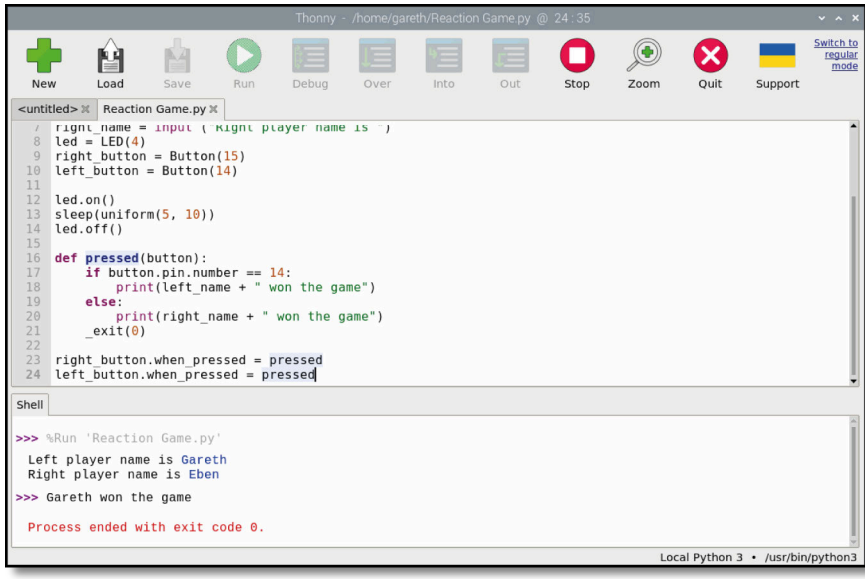
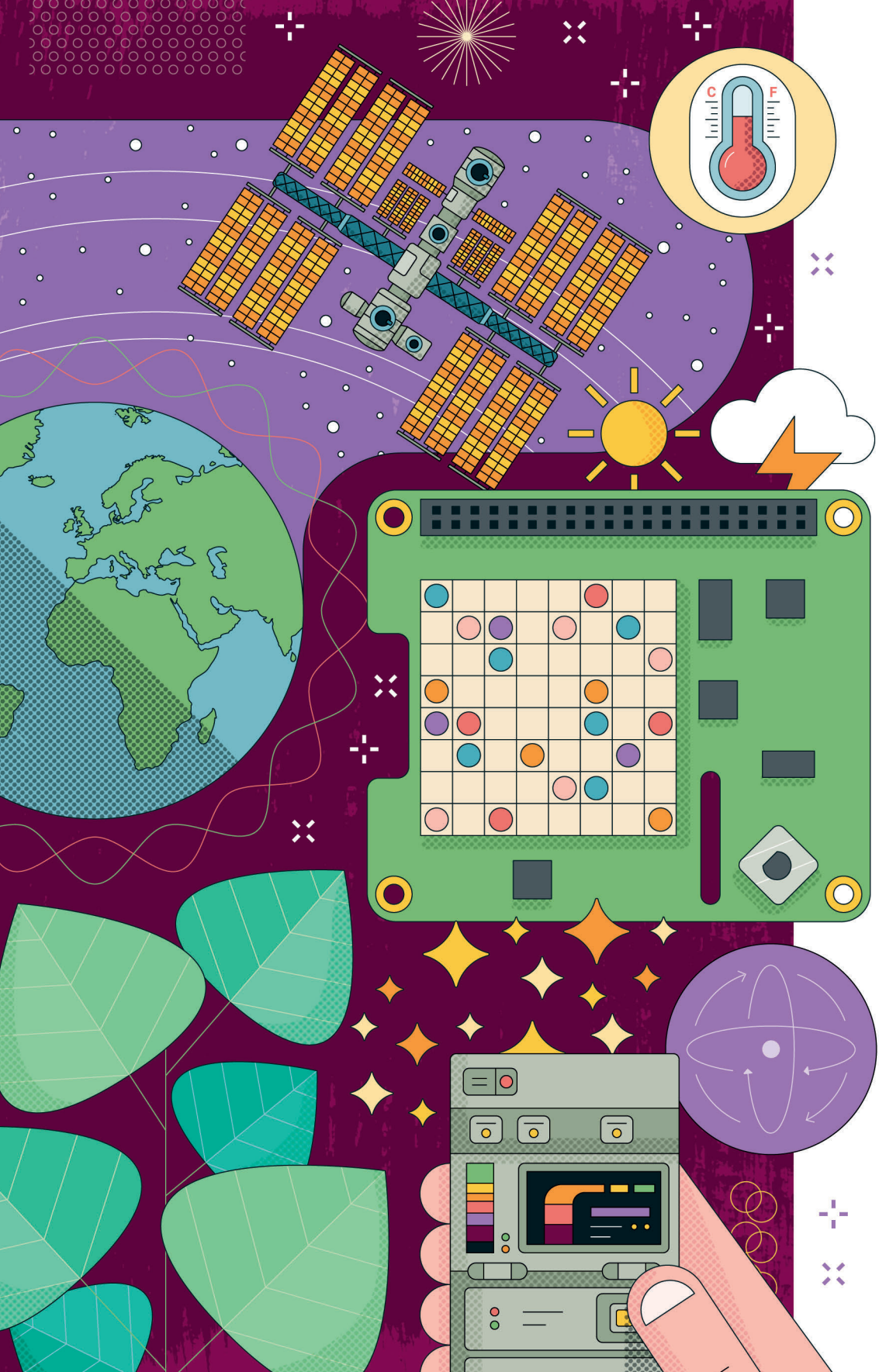


Figure 6-13 Whoever is first to press the button after the light goes out will be declared the winner



CHALLENGE: IMPROVE THE GAME

Can you add a loop, so the game runs continuously? Remember to remove the `_exit(0)` instruction first! Can you add a score counter, so you can see who is winning over multiple rounds? What about a timer, so you can see just how long it took you to react to the light going off?


C
F

Chapter 7

Physical computing with the Sense HAT

As used on the International Space Station, the Sense HAT is a multifunctional add-on board for Raspberry Pi, equipped with sensors and an LED matrix display.

Raspberry Pi comes with support for a special type of add-on board called *Hardware Attached on Top (HAT)*. HATs can add everything from microphones and lights to electronic relays and screens to Raspberry Pi, but one HAT in particular is very special: the Sense HAT.

WARNING!

At the time of writing, neither Scratch 3 nor the Sense HAT Emulator software have been updated to support Raspberry Pi 5, and the Sense HAT Emulator contained a bug that prevents it from running on the latest version of Raspberry Pi OS (see **rptl.io/sense-emu-fix** for a partial workaround). If you're having problems, check for updates (see Software Updates in Chapter 3, *Using your Raspberry Pi*).

The Sense HAT was designed specially for the Astro Pi space mission. A joint project between the Raspberry Pi Foundation, the UK Space Agency, and the European Space Agency (ESA), Astro Pi saw Raspberry Pi boards, cameras and Sense HATs carried up to the International Space Station (ISS) aboard an Orbital Science Cygnus cargo rocket. Since safely reaching orbit high above the Earth, the Raspberry Pis — nicknamed Ed and Izzy by the astronauts — have been used to run code and carry out scientific experiments contributed by tens of thousands of schoolchildren from dozens of countries across Europe. New, updated Raspberry Pi hardware (Raspberry Pi 4s nicknamed Flora, Fauna, and Fungi) was sent to the ISS in 2022. If you're in Europe and under 19

years old, you can find out how to run your own code and experiments in space at **astro-pi.org**.

The same Sense HAT hardware that's in use on the ISS can be found here on Earth, too, at all Raspberry Pi retailers — and if you don't want to buy a Sense HAT right now, you can simulate one in software.



REAL OR SIMULATED

This chapter is easier to follow if you use a real Sense HAT attached to a Raspberry Pi's GPIO header, but anyone who doesn't have one can skip "Installing the Sense HAT" on page 158 and try the projects out in the Sense HAT Emulator instead.

Introducing the Sense HAT

The Sense HAT (**Figure 7-1**) is a powerful, multifunctional add-on for Raspberry Pi. As well as an 8×8 matrix of 64 red, green, and blue (RGB) programmable LEDs which can be controlled to produce any colour from a range of millions, the Sense HAT includes a five-way joystick controller and six (or seven, on later models) on-board sensors.

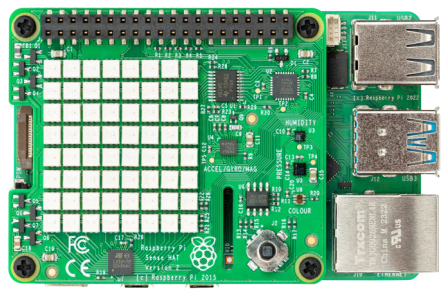


Figure 7-1 The Sense HAT

- **Gyroscope sensor** — Used to sense changes in angle over time, technically known as *angular velocity*, the gyroscope sensor can tell when you rotate the Sense HAT on any one of this three axes — and how quickly it's rotating.

- **Accelerometer** — Similar to the gyroscope sensor, but rather than monitoring an angle, it measures acceleration force in multiple directions. Combined, readings (data) from the accelerometer and the gyroscope sensor can help you track where a Sense HAT is pointing, and how it's being moved.

- **Magnetometer** — Measures the strength of a magnetic field. The magnetometer is another sensor which can help track the Sense HAT's movements: by measuring the Earth's natural magnetic field, the magnetometer can figure out the direction of magnetic north. The same sensor can also be used to detect metallic objects, and even electrical fields. All three of these sensors are built into a single chip, labelled **ACCEL/GYRO/MAG** on the Sense HAT's circuit board.

- **Humidity sensor** — Measures the amount of water vapour in the air (the *relative humidity*). Relative humidity can range from 0%, when no water vapour is present at all, to 100%, when the air is completely saturated. Humidity data can also help detect when it might be about to rain.

- **Barometric pressure sensor** — also known as the *barometer*, this measures air pressure. Although most people will be familiar with barometric pressure from the weather forecast, the barometer has a secret second use: it can track when you're climbing up or down a hill or mountain, as the air gets thinner and the pressure decreases the further you get from Earth's sea level.

- **Temperature sensor** — Measures how hot or cold the surrounding environment is. This measurement can be affected by how hot or cold the Sense HAT itself is: if you're using a case, you may find your readings are higher than you expect. The Sense HAT doesn't have a separate temperature sensor; instead, it uses temperature sensors built into the humidity and barometric pressure sensors. A program can use one or both of these sensors: it's up to you.

- **Colour and brightness sensor** — Only available on Sense HAT V2, the colour and brightness sensor picks up the light around you and reports on its intensity — great for projects in which you want to automatically dim and brighten the LEDs according to how well-lit your room is. The sensor can also be used to report on the colour of incoming light. Its readings will be affected by the light coming from Sense HAT's own LED matrix, so consider this when designing your experiments. This is the only sensor you can't emulate using the Sense HAT Emulator; you'll need a real Sense HAT V2 to use it.



SENSE HAT ON RASPBERRY PI 400

The Sense HAT is fully compatible with Raspberry Pi 400, and can be inserted directly into the GPIO header on the back. Doing so, however, means that the LEDs will be facing away from you and the board will be oriented upside-down.

To fix this, you'll need a GPIO extension cable or board. Compatible extensions include the Black HAT Hack3r range from **pimoroni.com;** you can use the Sense HAT with the Black HAT Hack3r board itself, or simply use the included 40-pin ribbon cable as an extension. Always check the manufacturer's instructions, though, to be sure you're connecting the cable and Sense HAT the right way around!

Installing the Sense HAT

Start by unpacking your Sense HAT and making sure you have all the pieces: you should have the Sense HAT itself, four metal or plastic pillars known as *spacers*, and eight screws. You may also have some metal pins mounted in a black plastic strip, like the GPIO pins on Raspberry Pi; if so, push this strip pin-side-up through the bottom of the Sense HAT until you hear a click.

The spacers are designed to stop the Sense HAT from bending and flexing as you use the joystick. While the Sense HAT will work without them being installed, using them will help protect your Sense HAT, Raspberry Pi, and GPIO header from damage.

If you're using the Sense HAT with Raspberry Pi Zero 2 W, you won't be able to use all four spacers. You'll also need to have soldered some pins onto the GPIO header, or have purchased your board from a reseller who has done it for you.



WARNING!

Hardware Attached on Top (HAT) modules should only ever be plugged into and removed from the GPIO header while your Raspberry Pi is switched off and disconnected from its power supply. Always be careful to keep the HAT flat when installing it, and double-check that it's lined up with the GPIO header pins before pushing it down.

Install the spacers by pushing four of the screws up from underneath your Raspberry Pi through the four mounting holes at each corner. Twist the spacers onto the screws. Push the Sense HAT down onto your Raspberry Pi's GPIO header, making sure to line it up properly with the pins underneath and to keep it as flat as possible.

Finally, screw the final four screws through the mounting holes on the Sense HAT and into the spacers you installed earlier. If it's installed properly, the Sense HAT should be flat and level, and shouldn't bend or wobble as you push on its joystick.

Plug the power back into your Raspberry Pi, and you'll see the LEDs on the Sense HAT light up in a rainbow pattern (**Figure 7-2**), then go dark again. Your Sense HAT is now installed!

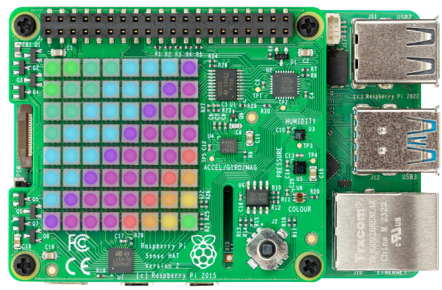


Figure 7-2 A rainbow pattern appears when the power is first turned on

If you want to remove the Sense HAT again, undo the top screws, lift the HAT off — being careful not to bend the pins on the GPIO header, as the HAT holds on quite tightly (you may need to prise it off gently) — then remove the spacers from Raspberry Pi.

You'll need some software to program the Sense HAT, which may not be already installed. If you can't find Scratch 3 and the Sense HAT Emulator in the **Programming** section of the Raspberry menu, head to Chapter 3, *Using your Raspberry Pi*, and follow the instructions in the Recommended Software section to install Scratch 3. Follow the instructions in Appendix B, *Installing and uninstalling software* to install the Sense HAT Emulator (**sense-emu-tools**).

PROGRAMMING EXPERIENCE

This chapter assumes experience with Scratch 3 or Python and the Thonny integrated development environment (IDE). If you haven't done so already, please turn to Chapter 4, *Programming with Scratch 3*, or Chapter 5, *Programming with Python*, and work through the projects in those chapters first.

Hello, Sense HAT!

As with all programming projects, there's an obvious place to start with the Sense HAT: scrolling a welcome message across its LED display. If you're using the Sense HAT emulator, load it now by clicking on the Raspberry Pi icon, choosing the **Programming** category, and clicking on **Sense HAT Emulator**.

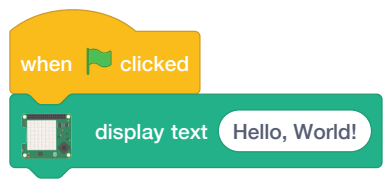
Greetings from Scratch

Load Scratch 3 from the Raspberry Pi menu. Click the **Add Extension** button at the bottom-left of the Scratch window. Click on the **Raspberry Pi Sense HAT** extension (**Figure 7-3**). This loads the blocks you need to control the various features of the Sense HAT, including its LED display. When you need them, you'll find them in the **Raspberry Pi Sense HAT** category.



Figure 7-3 Adding the Raspberry Pi Sense HAT extension to Scratch 3

Start by dragging a when clicked **Events** block onto the script area, then drag a display text Hello! block directly underneath it. Edit the text so that the block reads display text Hello, World!.



Click the green flag on the stage area and watch your Sense HAT or the Sense HAT emulator: the message will scroll slowly across Sense HAT's LED matrix, lighting up the LED pixels to form each letter in turn (**Figure 7-4**). Congratulations: your program's a success!

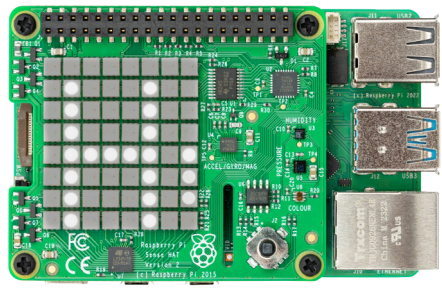


Figure 7-4 Your message scrolls across the LED matrix

Now that you can scroll a simple message, it's time to control how that message is displayed. As well as being able to modify the message, you can alter the rotation — which direction the message is displayed in. Drag a set rotation to 0 degrees block from the blocks palette and insert it below when clicked and above display text Hello, World!. Click on the down arrow next to **0** and change it to **90**.

Click the green flag and you'll see the same message as before, but rather than scrolling left-to-right, it will scroll bottom-to-top (**Figure 7-5**) — you'll need to turn your head, or the Sense HAT, to read it!

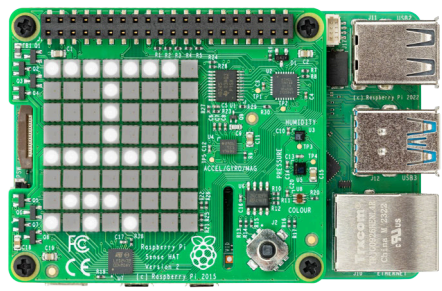


Figure 7-5 This time the message scrolls vertically

Now change the rotation back to 0, and drag a set colour block between set rotation to 0 degrees and display text Hello, World!. Click on the colour at the end of the block to bring up Scratch's colour picker, and find a nice bright yellow colour. Now click the green flag to see how your program's output has changed (**Figure 7-6**).

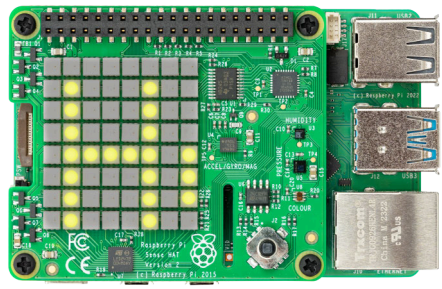


Figure 7-6 Changing the colour of the text

Finally, drag a set background block between set colour and display text Hello, World!. Click on the colour to bring up the colour picker again. This time, choosing a colour doesn't affect the LEDs that make up the message. It changes the LEDs that don't: the background. Find a nice blue colour, then click the green flag again: this time your message will be in a bright yellow on a blue background. Try changing these colours to find your favourite combination — not all colours work well together!

As well as being able to scroll entire messages, you can show individual letters. Drag your display text Hello, World! block off the script area to delete it, then drag a display character A block onto the script area in its place.

Click the green flag, and you'll see the difference: this block shows only one letter at a time, and the letter stays on the Sense HAT until you tell it otherwise without scrolling or disappearing. The same colour control blocks apply to this block as the display text block: try changing the letter's colour to red (**Figure 7-7**).



CHALLENGE: REPEAT THE MESSAGE

Can you use your knowledge of loops to make a scrolling message repeat itself? Can you create a program that spells out a word letter-by-letter, using different colours?

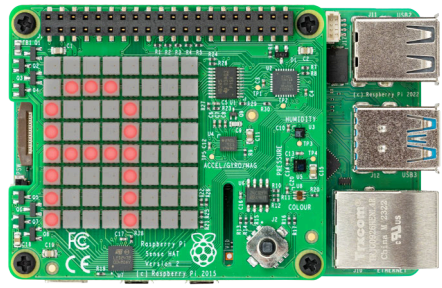


Figure 7-7 Displaying a single letter

Greetings from Python

Load Thonny by clicking on the Raspberry icon, choosing **Programming**, and clicking on **Thonny**. If you're using the Sense HAT emulator and it gets covered by the Thonny window, click and hold the mouse button on either window's title bar — at the top, in blue — and drag it to move it around the desktop until you can see both windows.

PYTHON LINE CHANGE

Python code written for a physical Sense HAT runs on the Sense HAT emulator, and vice-versa, with only one change. If you're using the Sense HAT emulator with Python you'll need to change the line `from sense_hat import SenseHat` in all the programs from this chapter to `from sense_emu import SenseHat` instead. If you want to then run them on a physical Sense HAT again, just change the line back!

To use the Sense HAT, or Sense HAT emulator, in a Python program, you need to import the Sense HAT library. Type the following into the script area, remembering to use `sense_emu` (in place of `sense_hat`) if you're using the Sense HAT emulator:

```
from sense_hat import SenseHat
sense = SenseHat()
```

The Sense HAT library has a simple function for taking a message, formatting it so that it can be shown on the LED display, and scrolling it smoothly. Type the following:

```
sense.show_message("Hello, World!")
```

Save your program as **Hello Sense HAT.py**, then click the **Run** button. You'll see your message scroll slowly across the Sense HAT's LED matrix, lighting up the LED pixels to form each letter in turn (**Figure 7-8**). Congratulations: your program's a success!

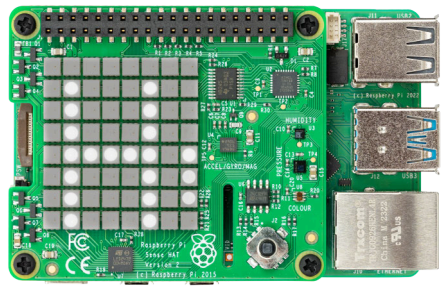


Figure 7-8 Scrolling a message across the LED matrix

The `show_message()` function has more tricks up its sleeve than that, though. Go back to your program and edit the last line so it says:

```
sense.show_message("Hello, World!", text_colour=(255, 255, 0),
                   back_colour=(0, 0, 255), scroll_speed=(0.05))
```

These extra instructions, separated by commas, are known as *parameters*, and they control various aspects of the `show_message()` function. The simplest is `scroll_speed=()`, which changes how quickly the message scrolls across the screen. A value of 0.05 in here scrolls at roughly twice the usual speed. The bigger the number, the lower the speed.

The `text_colour=()` and `back_colour=()` parameters — spelled in the British English way, unlike most Python instructions — set the colour of the writing and the background respectively. They don't accept the names of colours, though; you have to define the colour you want using a trio of numbers. The first number represents the amount of red in the colour, from 0 for no red at all to 255 for as much red as possible; the second number is the amount of green in the colour; and the third number the amount of blue. Together, these are known as *RGB* — for red, green, and blue.

Click on the **Run** icon and watch the Sense HAT: this time, the message will scroll considerably faster, and be displayed in a bright yellow on a blue background (**Figure 7-9**). Try changing the parameters to find a speed and colour combination that works for you.

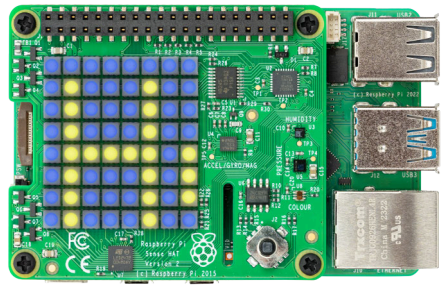


Figure 7-9 Changing the colour of the message and background

If you want to use friendly names instead of RGB values to set your colours, you'll need to create variables. Above your `sense.show_message()` line, add the following:

```
yellow = (255, 255, 0)
blue = (0, 0, 255)
```

Go back to your `sense.show_message()` line and edit it so it reads:

```
sense.show_message("Hello, World!", text_colour=(yellow),
                   back_colour=(blue), scroll_speed=(0.05))
```

Click the **Run** icon again, and you'll see that nothing has changed: your message is still in yellow on a blue background. This time, though, you've used the variable names to make your code more readable. Instead of a string of numbers, the code explains what colour it's setting. You can define as many colours as you like: try adding a variable called `red` with the values 255, 0, and 0; a variable called `white` with the values 255, 255, 255; and a variable called `black` with the values 0, 0, and 0.

As well as being able to scroll full messages, you can display individual letters. Delete your `sense.show_message()` line altogether, and type the following in its place:

```
sense.show_letter("A")
```

Click **Run**, and you'll see the letter 'A' appear on the Sense HAT's display. This time, it'll stay there: individual letters, unlike messages, don't automatically scroll. You can control `sense.show_letter()` with the same colour parameters as `sense.show_message()`, too: try changing the colour of the letter to red (**Figure 7-10**).

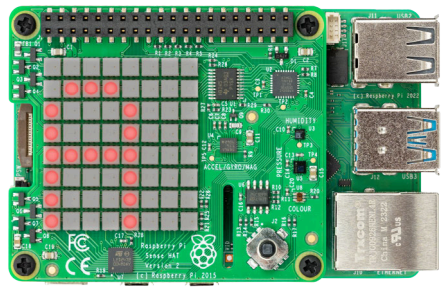


Figure 7-10 Displaying a single letter



CHALLENGE: REPEAT THE MESSAGE

Can you use your knowledge of loops to make a scrolling message repeat itself? Can you create a program that spells out a word letter-by-letter, using different colours? How fast can you make a message scroll?

Next steps: Drawing with light

The Sense HAT's LED display isn't just for messages: you can display pictures, too. Each LED can be treated as a single pixel — short for *picture element* — in an image of your choosing. This allows you to jazz up your programs with pictures and even animation.

To create drawings, you need to be able to change individual LEDs. To do that, you'll need to understand how the Sense HAT's LED matrix is laid out. Then you'll be able to write a program that turns the correct LEDs on or off.

There are eight LEDs in each row of the display, and eight in each column (**Figure 7-11**). When counting the LEDs, though, you should start at 0 and end at 7, like most programming languages do. The first LED is in the top-left corner, the last is in the bottom-right. Using the numbers from the rows and columns, you can find the *coordinates* of any LED on the matrix. The blue LED in the pictured matrix is at coordinates 0, 1; the red LED is at coordinates 7, 4. The X axis coordinate comes first and increases across the matrix, followed by the Y axis, which increases down the matrix.

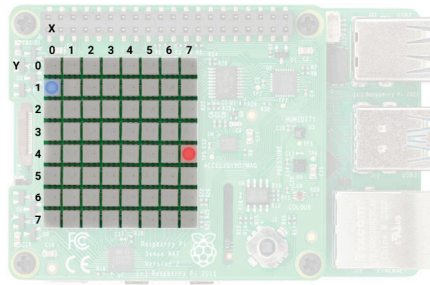
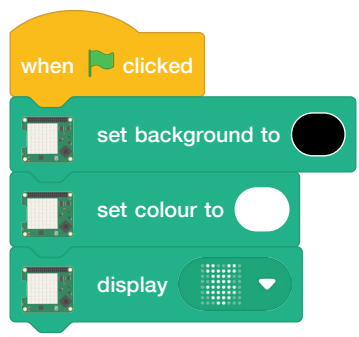


Figure 7-11 LED matrix coordinates system

When planning pictures to draw on the Sense HAT, it may help to draw them by hand first, on gridded paper, or you can plan things out in a spreadsheet such as LibreOffice Calc.

Pictures in Scratch

Start a new project in Scratch, saving your existing project if you want to keep it. If you've been working through the projects in this chapter, Scratch 3 will keep the Raspberry Pi Sense HAT extension loaded; if you have closed and reopened Scratch 3 since your last project, load the extension using the **Add Extension** button. Drag a when clicked **Events** block onto the code area, then drag set background and set colour blocks underneath it. Edit both to set the background colour to black, and the colour to white. Make black by sliding the **Brightness** and **Saturation** sliders to 0; make white by sliding **Brightness** to 100 and **Saturation** to 0. You'll need to do this at the start of every Sense HAT program, otherwise Scratch will simply use the last colours you chose — even if you chose them in a different program. Finally, drag a display raspberry block to the bottom of your program.



Click the green flag: you'll see the Sense HAT's LEDs light up a raspberry shape (**Figure 7-12**).

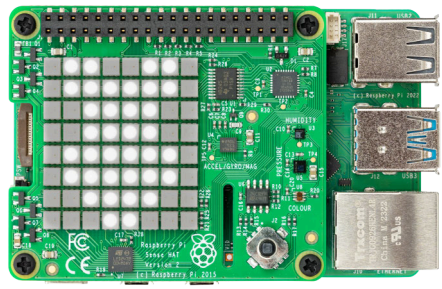


Figure 7-12 Displaying the raspberry shape with Scratch



WARNING

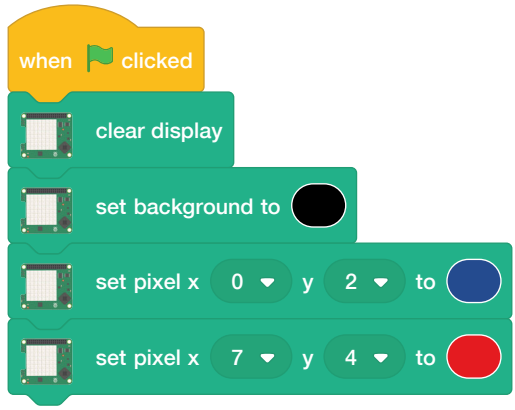
When the LEDs are bright white, as in this code example, avoid looking directly at them — they're bright enough to hurt your eyes.

You're not limited to the pre-set raspberry shape, either. Click on the down-arrow next to the raspberry image to activate drawing mode. You can click on any LED on the pattern individually to switch it on or off, while the two buttons at the bottom set all LEDs to on or off. Try drawing your own pattern now, then click the green arrow to see it on the Sense HAT. Also try changing the colour and the background colour using the blocks above.

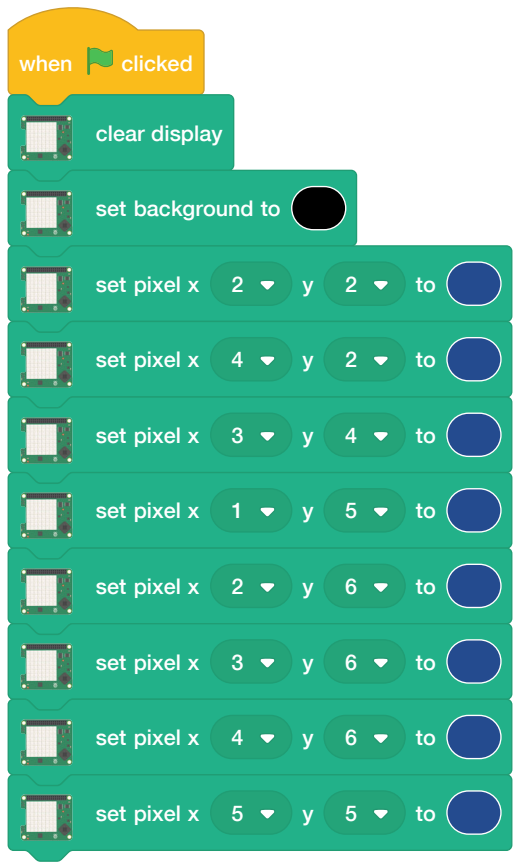
When you've finished, drag the three blocks into the blocks palette to delete them, and place a clear display block under when clicked; click the green flag, and all the LEDs will switch off.

To make a picture, you need to be able to control individual pixels and to assign them different colours. You can do this by chaining edited display raspberry blocks with set colour blocks, or you can address each pixel individually. Try creating your own version of the LED matrix example pictured at the start of this section. Two specifically selected LEDs are lit up in red and blue. Leave the clear display block at the top of your program and drag a set background block underneath it. Change the set background block to black, then drag two set pixel x 0 y 0 blocks underneath it. Finally, edit these blocks as shown.

Click the green flag, and you'll see your LEDs light up to match the matrix image (**Figure 7-13**). Congratulations: you can control individual LEDs!



Edit your existing set pixel blocks as follows, and drag more onto the bottom until you have created the following program.



Before you click the green flag, see if you can guess what picture is going to appear based on the LED matrix coordinates you've used. Now run your program and see if you're right!



CHALLENGE: NEW DESIGNS

Can you design more pictures? Try getting some graph paper and using it to plan out your picture by hand. Can you draw the picture on your Sense HAT and make the colours change?

Pictures in Python

Start a new program in Thonny and save it as Sense HAT Drawing, then type the following — remembering to use `sense_emu` (in place of `sense_hat`) if you're using the emulator:

```
from sense_hat import SenseHat
sense = SenseHat()
```

Remember that you need both these lines your program in order to use the Sense HAT. Next, type:

```
sense.clear(255, 255, 255)
```

While not looking directly at the Sense HAT's LEDs, click the **Run** icon: you should see them all turn a bright white (**Figure 7-13**) — which is why you shouldn't be looking directly at them when you run your program!

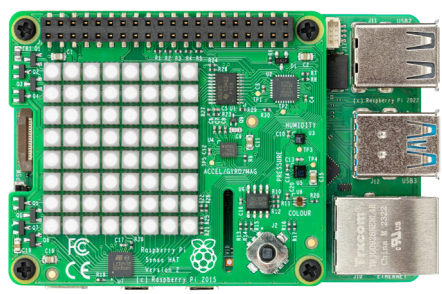


Figure 7-13 Turning all the LEDs on

WARNING

When the LEDs are bright white, avoid looking directly at them — they're bright enough to hurt your eyes.

The `sense.clear()` command is designed to clear the LEDs of any previous programming, but accepts RGB colour parameters — meaning you can change the display to any colour you like. Try editing the line to:

```
sense.clear(0, 255, 0)
```

Click **Run**, and the Sense HAT will go bright green (**Figure 7-14**). Experiment with different colours, or add the colour-name variables you created when you wrote your Hello World program to make things easier to read.

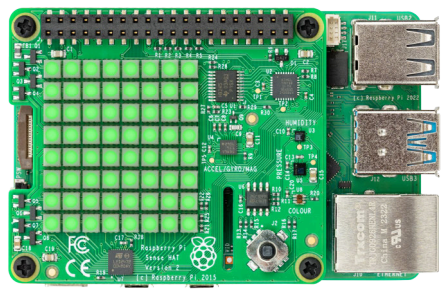


Figure 7-14 The LED matrix lit up in bright green

To clear the LEDs, you need to use the RGB values for black: 0 red, 0 blue, and 0 green. There's an easier way, though. Edit the line of your program to read:

```
sense.clear()
```

The Sense HAT will go dark. This is because for the `sense.clear()` function, having nothing between the brackets is equivalent to telling it to turn all LEDS to black — i.e. switch them off (**Figure 7-15**). When you need to completely clear the LEDs in your programs, that's the function you should use.

To create your own version of the LED matrix pictured earlier in this chapter, with two specifically selected LEDs lit up in red and blue, add the following lines to your program after `sense.clear()`:

```
sense.set_pixel(0, 2, (0, 0, 255))
sense.set_pixel(7, 4, (255, 0, 0))
```

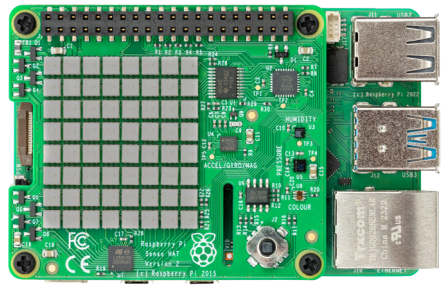


Figure 7-15 Use the `sense.clear` function to turn off all the LEDs

The first two numbers are the pixel's location on the matrix, with the X (across) axis followed by the Y (down) axis. The second set of numbers, surrounded by their own brackets, are the RGB values for the pixel's colour. Click the **Run** button, and you'll see the effect: two LEDs on your Sense HAT will light up, as shown in **Figure 7-11**.

Delete those two lines, and type in the following:

```
sense.set_pixel(2, 2, (0, 0, 255))
sense.set_pixel(4, 2, (0, 0, 255))
sense.set_pixel(3, 4, (100, 0, 0))
sense.set_pixel(1, 5, (255, 0, 0))
sense.set_pixel(2, 6, (255, 0, 0))
sense.set_pixel(3, 6, (255, 0, 0))
sense.set_pixel(4, 6, (255, 0, 0))
sense.set_pixel(5, 5, (255, 0, 0))
```

Before you click **Run**, look at the coordinates and compare them to the matrix: can you guess what picture these instructions are going to draw? Click **Run** to find out if you're right!

Drawing a detailed picture using individual `set_pixel()` functions is slow, though. To speed things up, you can change multiple pixels at the same time. Delete all your `set_pixel()` lines and type the following:

```
g = (0, 255, 0)
b = (0, 0, 0)
creeper_pixels = [
    g, g, g, g, g, g, g, g,
    g, g, g, g, g, g, g, g,
```

```
    g, b, b, g, g, b, b, g,
    g, b, b, g, g, b, b, g,
    g, g, g, b, b, g, g, g,
    g, g, b, b, b, b, g, g,
    g, g, b, b, b, b, g, g,
    g, g, b, g, g, b, g, g
]
sense.set_pixels(creeper_pixels)
```

There's a lot there, but start by clicking **Run** to see if you recognise a certain little creeper. The first two lines create two variables to hold colours: green and black. To make the code for the drawing easier to write and read, the variables are single letters: '`g`' for green and '`b`' for black.

The next block of code creates a variable which holds colour values for all 64 pixels on the LED matrix, separated by commas and enclosed between square brackets. Instead of numbers, though, it uses the colour variables you created earlier: look closely, remembering '`g`' is for green and '`b`' is for black, and you can already see the picture that will appear (**Figure 7-16**).

Finally, `sense.set_pixels(creeper_pixels)` takes that variable and uses the `sense.set_pixels()` function to draw on the entire matrix at once. Much easier than trying to draw pixel-by-pixel!

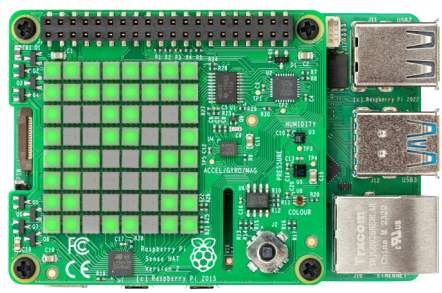


Figure 7-16 Displaying an image on the matrix

You can also rotate and flip images, either as a way to show images the right way up when your Sense HAT is turned around, or as a way to create simple animations using a single asymmetrical image.

Start by editing your `creeper_pixels` variable to close his left eye, by replacing the four '`b`' pixels, starting with the first two on the third line and then the first two on the fourth line, with '`g`':

```
creeper_pixels = [
    g, g, g, g, g, g, g, g,
    g, g, g, g, g, g, g, g,
    g, g, g, g, g, b, b, g,
    g, g, g, g, g, b, b, g,
    g, g, g, b, b, g, g, g,
    g, g, b, b, b, b, g, g,
    g, g, b, b, b, b, g, g,
    g, g, b, g, g, b, g, g
]
```

Click **Run**, and you'll see the creeper's left eye close (**Figure 7-17**). To make an animation, go to the top of your program and add the line:

```
from time import sleep
```

Then go to the bottom and type:

```
while True:
    sleep(1)
    sense.flip_h()
```

Click **Run**, and watch the creeper as it blinks its eyes, one at a time!

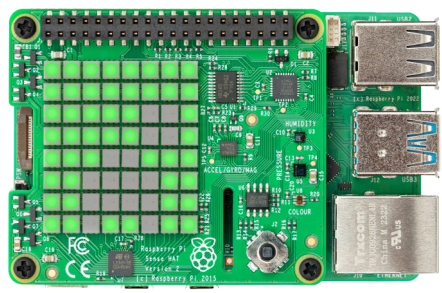


Figure 7-17 Showing a simple two-frame animation

The `flip_h()` function flips an image on the horizontal axis. If you want to flip an image on its vertical axis, replace `sense.flip_h()` with `sense.flip_v()` instead. You can also rotate an image by 0, 90, 180, or 270 degrees using `sense.set_rotation(90)`, changing the number according to how many degrees you want to rotate the image. Try using this to have the creeper spin around instead of blinking!

CHALLENGE: NEW DESIGNS

Can you design more pictures and animations? Try getting some graph paper and using it to plan out your picture by hand, to make writing the variable easier. Can you draw a picture and make the colours change? Remember: you can change the variables after you've already used them once.

Sensing the world around you

The Sense HAT's real power lies in its sensors. These allow you to take readings of everything from temperature to acceleration, and use the information they give you in your programs.

EMULATING THE SENSORS

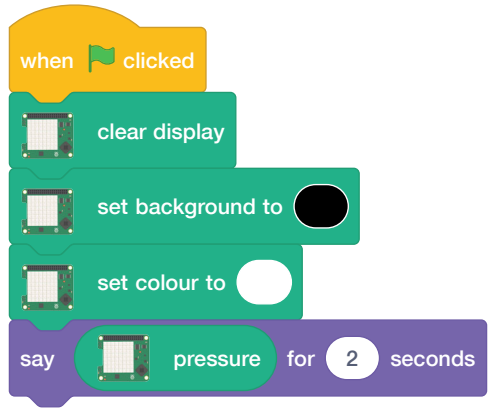
If you're using the Sense HAT Emulator, you'll need to enable inertial and environmental sensor simulation: in the Emulator, click **Edit**, then **Preferences**, then tick them, if they are not already ticked. In the same menu, choose **180°..360°|0°..180°** under **Orientation Scale** to make sure the numbers in the Emulator match the numbers reported by Scratch and Python, then click the Close button.

Environmental sensing

The barometric pressure sensor, humidity sensor, and temperature sensor are all environmental sensors; they take measurements from the environment surrounding the Sense HAT.

Environmental sensing in Scratch

Start a new program in Scratch, saving your old one if you wish, and add the **Raspberry Pi Sense HAT** extension if it isn't already loaded. Drag a when clicked **Events** block onto your code area, then a clear display block underneath, and a set background to black block underneath that. Next, add a set colour to white block — use the **Brightness** and **Saturation** sliders to choose the correct colour. It's always a good idea to do this at the start of your programs, as it will make sure the Sense HAT isn't showing anything left over from an old program, while guaranteeing what colours you're using. Drag a say Hello! for 2 seconds **Looks** block directly underneath your existing blocks. To take a reading from the pressure sensor, find the pressure block in the **Raspberry Pi Sense HAT** category and drag it over the word '`Hello!`' in your say Hello! for 2 seconds block.



Click the green flag, and the Scratch cat will tell you the current reading from the pressure sensor in *millibars*. After two seconds, the message will disappear. Try blowing on the Sense HAT (or moving the **Pressure** slide up in the emulator) and clicking the green flag to run the program again; you should see a higher reading this time (**Figure 7-18**).

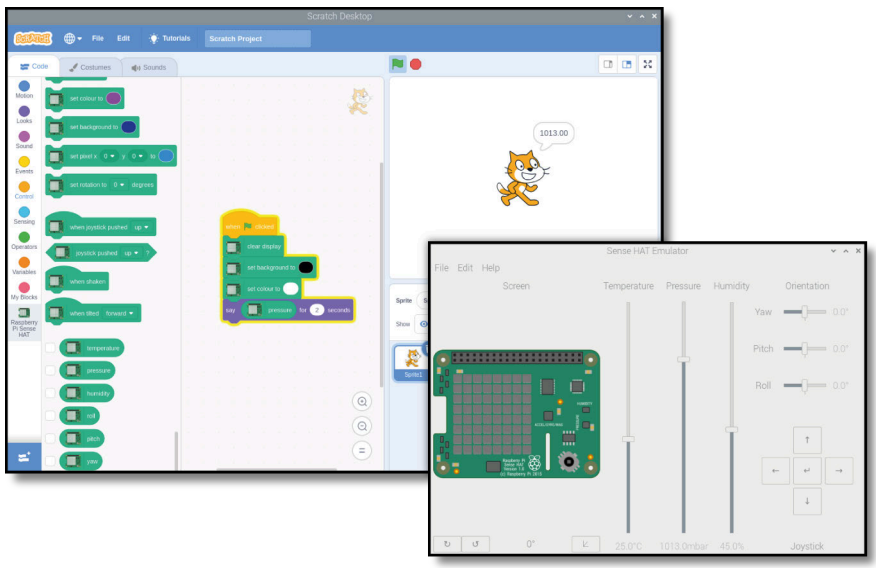


Figure 7-18 Showing the pressure sensor reading



CHANGING VALUES

If you're using the Sense HAT emulator, you can change the values reported by each of the emulated sensors using its sliders and buttons. Try sliding the pressure sensor setting down towards the bottom, then clicking the green flag again.

To switch to the humidity sensor, delete the pressure block and replace it with humidity. Run your program again, and you'll see the current relative humidity of your room. Again, you can try running the program while blowing on the Sense HAT (or moving the emulator's **Humidity** slider up) to change the reading (**Figure 7-19**) — your breath is surprisingly humid!

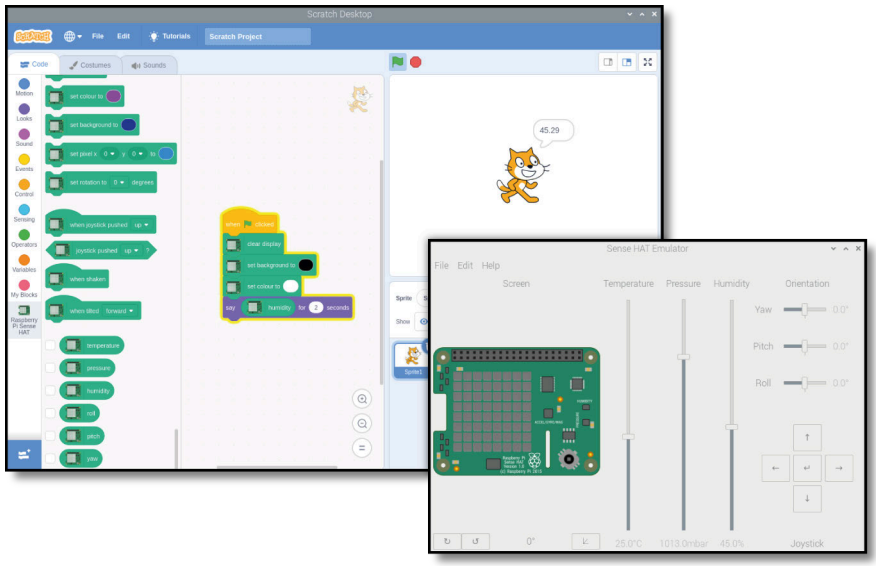


Figure 7-19 Displaying the reading from the humidity sensor

Using the temperature sensor is as easy as deleting the humidity block and replacing it with temperature, then running your program again. You'll see a temperature in degrees Celsius (**Figure 7-20**). This might not be the exact temperature of your room, however: your Raspberry Pi generates heat all the time it's running, and this warms the Sense HAT and its sensors too.

CHALLENGE: SCROLL AND LOOP

Can you change your program to take a reading from each of the sensors in turn, then scroll them across the LED matrix rather than printing them to the stage area? Can you make your program loop, so it's constantly printing the current environmental conditions?

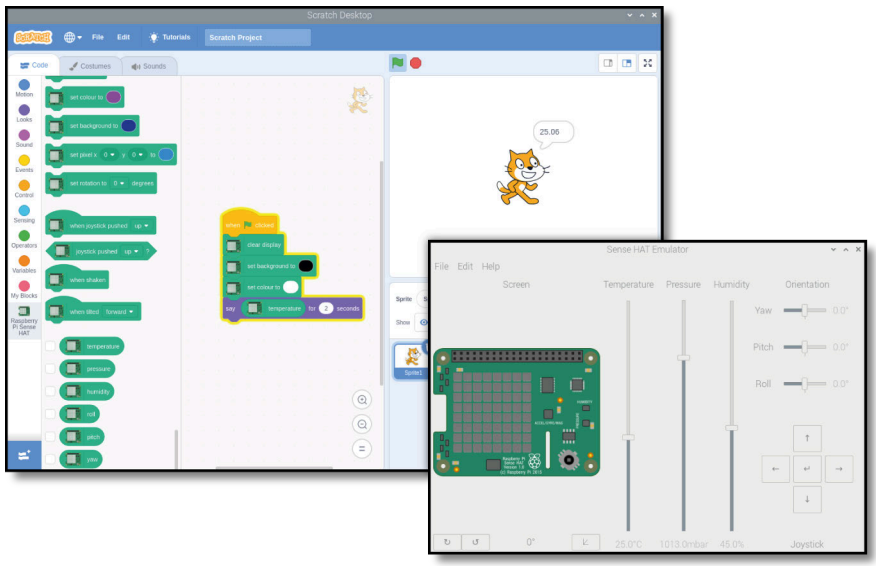


Figure 7-20 Displaying the temperature sensor reading

Environmental sensing in Python

To start taking readings from sensors, create a new program in Thonny and save it as **Sense HAT Sensors.py**. Type the following into the script area — you will have to do this every time you use the Sense HAT — and remember to use `sense_emu` if you're using the emulator:

```
from sense_hat import SenseHat
sense = SenseHat()
sense.clear()
```

It's always a good idea to include `sense.clear()` at the start of your programs, just in case the Sense HAT's display is still showing something from the last program it ran.

To take a reading from the pressure sensor, type:

```
pressure = sense.get_pressure()
print(pressure)
```

Click **Run** and you'll see a number printed to the Python shell at the bottom of the Thonny window. This is the air pressure reading detected by the barometric pressure sensor, in *millibars* (**Figure 7-21**).

CHANGING VALUES

If you're using the Sense HAT emulator, you can change the values reported by each of the emulated sensors using its sliders and buttons. Try sliding the pressure sensor setting down towards the bottom, then clicking **Run** again.

Try blowing on the Sense HAT (or moving the **Pressure** slider up in the emulator) while clicking the **Run** icon again; the number should be higher this time.

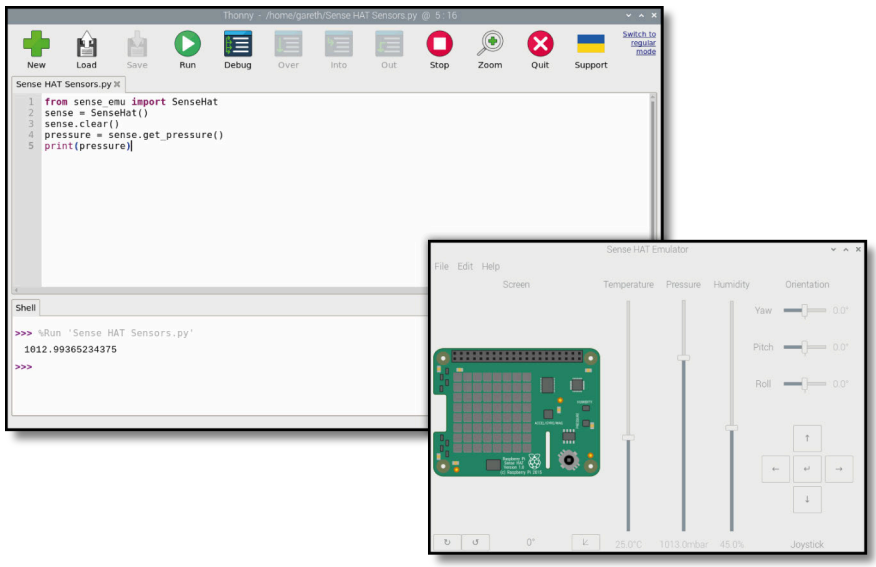


Figure 7-21 Printing a pressure reading from the Sense HAT

To switch to the humidity sensor, remove the last two lines of code and replace them with:

```
humidity = sense.get_humidity()
print(humidity)
```

Click **Run** and you'll see another number printed to the Python shell: this time, it's the current relative humidity of your room as a percentage. Again, you can blow on the Sense HAT (or move the emulator's **Humidity** slider up) and you'll see it go up when you run your program again (**Figure 7-22**) — your breath is surprisingly humid!

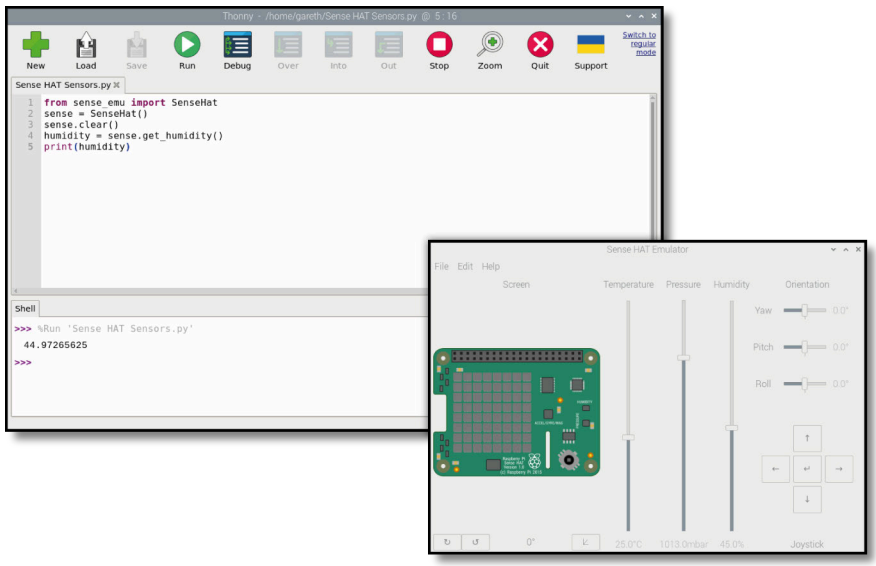


Figure 7-22 Displaying the humidity sensor reading

For the temperature sensor, remove the last two lines of your program and replace them with:

```
temp = sense.get_temperature()
print(temp)
```

Click **Run** again, and you'll see a temperature in degrees Celsius (**Figure 7-23**). This might not be the exact temperature of your room, however: your Raspberry Pi generates heat all the time it's running, and this warms the Sense HAT and its sensors too.

Normally the Sense HAT reports the temperature based on a reading from the temperature sensor built into the humidity sensor. If you want to use the reading from the pressure sensor instead, you should use `sense.get_temperature_from_pressure()`. It's also possible to combine the two readings to get an average, which may be more accurate than using either sensor alone. To do this, delete the last two lines of your program and type:

```
htemp = sense.get_temperature()
ptemp = sense.get_temperature_from_pressure()
temp = (htemp + ptemp) / 2
print(temp)
```

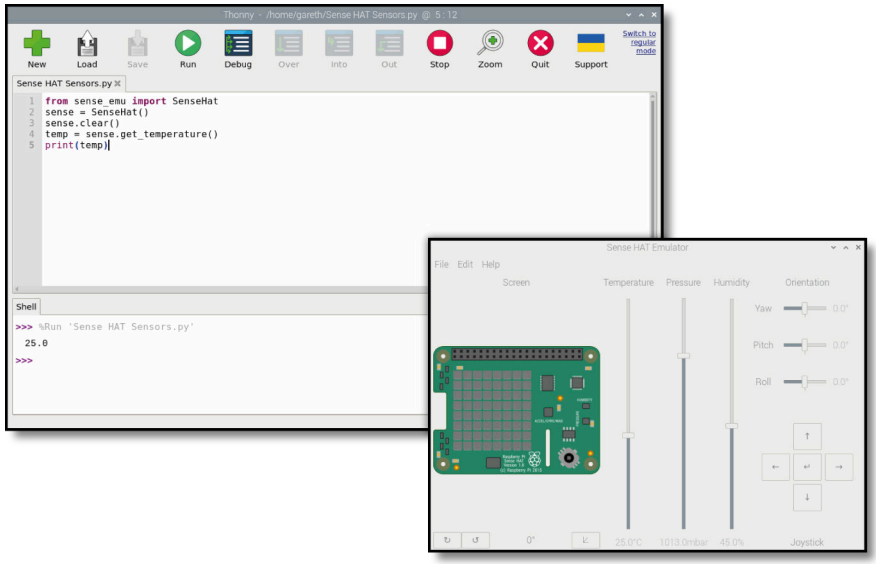


Figure 7-23 Showing the current temperature reading

Click the **Run** icon, and you'll see a number printed to the Python console (**Figure 7-24**). This time, it's based on readings from both sensors, which you've added together and divided by two — the number of readings — to get an average of both. If you're using the emulator, all three methods — humidity, pressure, and average — will show around the same number.

CHALLENGE: SCROLL AND LOOP

Can you change your program to take a reading from each of the sensors in turn, then scroll them across the LED matrix rather than printing them to the shell? Can you make your program loop, so it's constantly printing the current environmental conditions?

Inertial sensing

The gyroscopic sensor, accelerometer, and magnetometer combine to form what is known as an *inertial measurement unit (IMU)*. While, technically speaking, these sensors take measurements from the surrounding environment just like the environmental sensors — the magnetometer, for example, measures magnetic field strength — they're usually used for data about the movement of the Sense HAT itself. The IMU is the sum of multiple sensors.

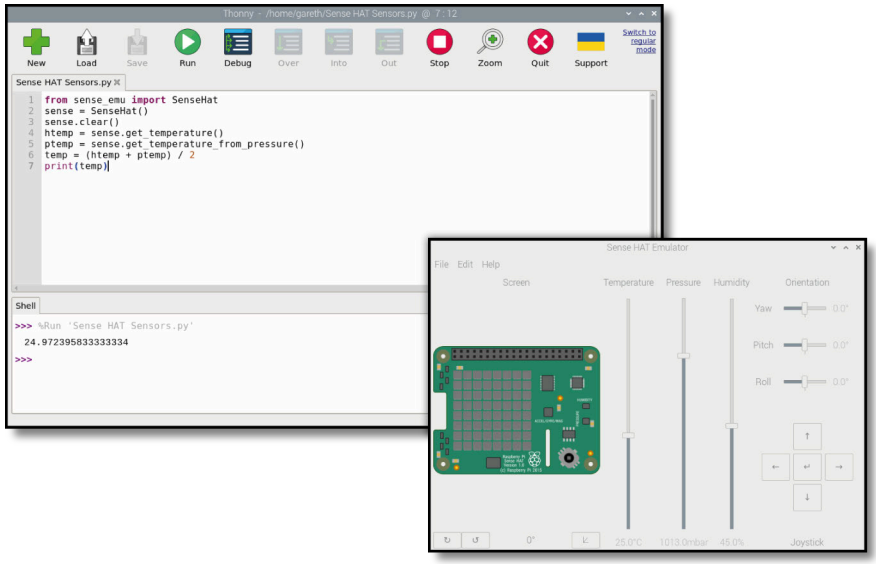


Figure 7-24 A temperature based on the readings from both sensors

Some programming languages allow you to take readings from each sensor independently, while others will only give you a combined reading.

Before you can make sense of the IMU, you need to understand how things move. The Sense HAT, and the Raspberry Pi it's attached to, can move along three spatial axes: side-to-side on the X axis; forwards and backwards on the Y axis; and up and down on the Z axis (**Figure 7-25**). It can also rotate along these three same axes, but their names change: rotating on the X axis is called *roll*, rotating on the Y axis is called *pitch*, and rotating on the Z axis is called *yaw*. When you rotate the Sense HAT along its short axis, you're adjusting its pitch; rotate along its long axis, and that's roll. Spin it around while keeping it flat on the table, and you're adjusting its yaw. Think of them like an aeroplane: when it's taking off, it increases its pitch to climb. When it's doing a victory roll, that's literally it spinning along its roll axis; when it's using its rudder to turn like a car would, without rolling, that's yaw.

Inertial sensing in Scratch

Start a new program in Scratch and load the **Raspberry Pi Sense HAT** extension, if it's not already loaded. Start your program in the same way as before: drag a when clicked **Events** block onto your code area, then drag a clear display block underneath it followed by dragging and editing a set background to black and a set colour to white block. Next, drag a forever

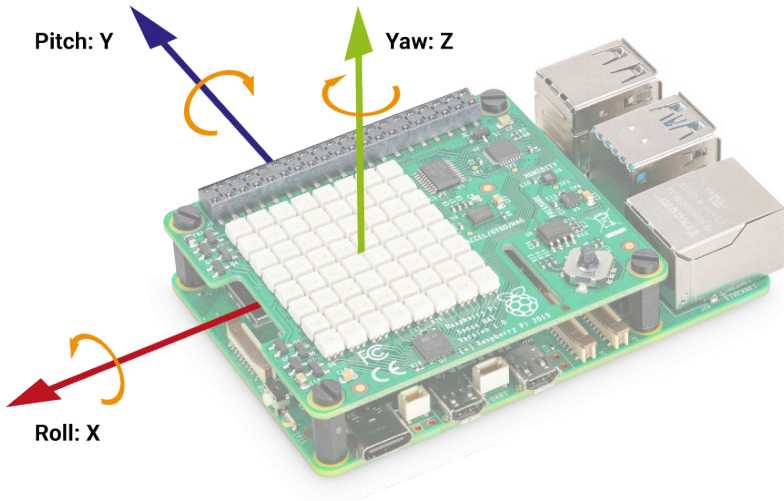
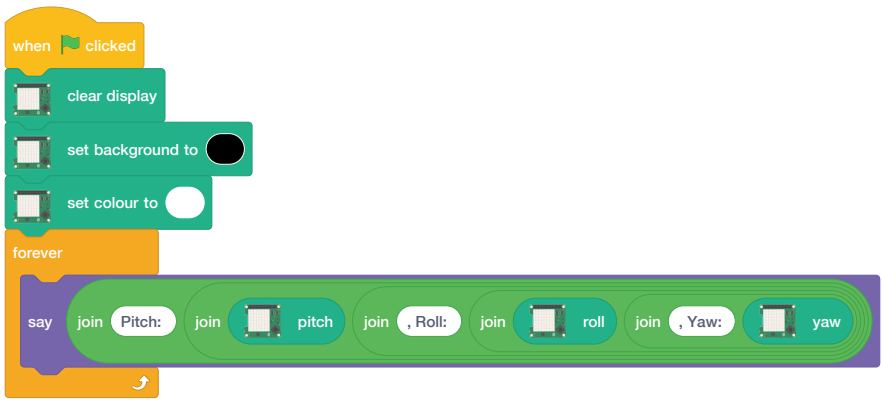


Figure 7-25 The spatial axes of the Sense HAT's IMU

block to the bottom of your existing blocks and fill it with a say Hello! block. To show a reading for each of the three axes of the IMU — pitch, roll, and yaw — you'll need to add join **Operator** blocks plus the corresponding **Raspberry Pi Sense HAT** blocks. Remember to include spaces and commas, so that the output is easy to read.



Click the green flag to run your program, and try moving the Sense HAT and Raspberry Pi around — being careful not to dislodge any cables! As you tilt the Sense HAT through its three axes, you'll see the pitch, roll, and yaw values change accordingly (**Figure 7-26**).

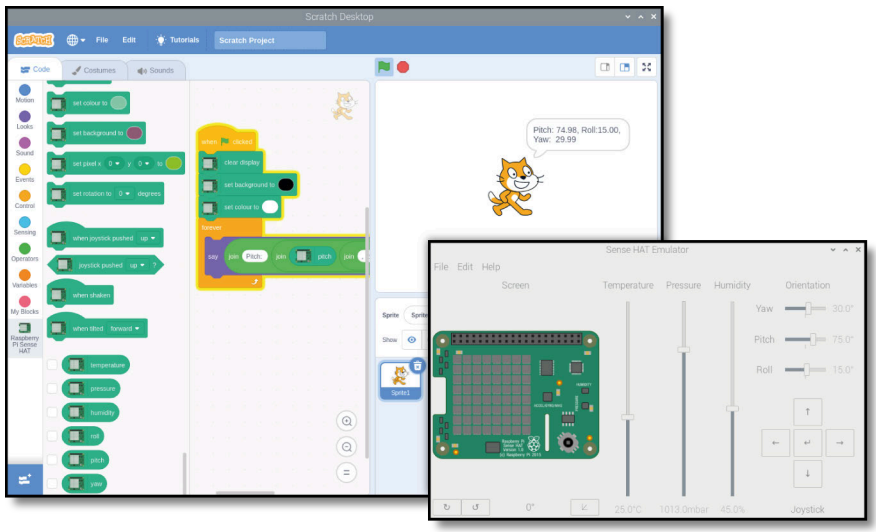


Figure 7-26 Displaying the pitch, roll, and yaw values

Inertial sensing in Python

Start a new program in Thonny and save it as **Sense HAT Movement.py**. Fill in the usual starting lines, remembering to use `sense_emu` if you're using the Sense HAT emulator:

```
from sense_hat import SenseHat
sense = SenseHat()
sense.clear()
```

To use information from the IMU to work out the current orientation of the Sense HAT on its three axes, type the following:

```
orientation = sense.get_orientation()
pitch = orientation["pitch"]
roll = orientation["roll"]
yaw = orientation["yaw"]
print("pitch {0} roll {1} yaw {2}".format(pitch, roll, yaw))
```

Click **Run** and you'll see readings for the Sense HAT's orientation split across the three axes (**Figure 7-27**). Try rotating the Sense HAT and clicking **Run** again. You should see the numbers change to reflect its new orientation.

The IMU can do more than measure orientation, though: it can also detect movement. To get accurate readings for movement, the IMU needs to be read

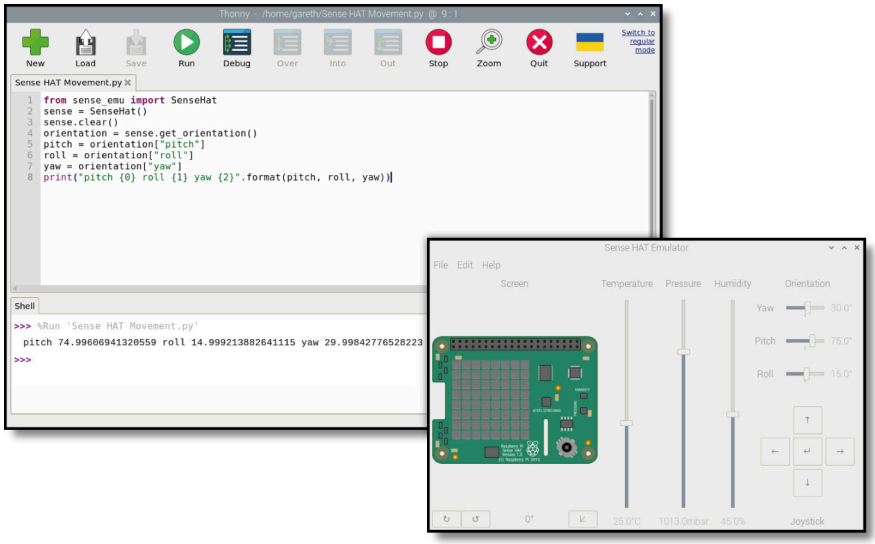


Figure 7-27 Showing the Sense HAT's pitch, roll, and yaw values

frequently in a loop. Taking a single reading won't give you any useful information when it comes to detecting movement. Delete everything after `sense.clear()` then type the following code:

```
while True:
    acceleration = sense.get_accelerometer_raw()
    x = acceleration["x"]
    y = acceleration["y"]
    z = acceleration["z"]
```

You now have variables containing the current accelerometer readings for the three spatial axes: X, or left and right; Y, or forwards and backwards; and Z, or up or down. The numbers from the accelerometer sensor can be difficult to read, so type the following to make them easier to understand by rounding them to the nearest whole number:

```
    x = round(x)
    y = round(y)
    z = round(z)
```

Finally, print the three values by typing the following line:

```
    print("x={0}, y={1}, z={2}".format(x, y, z))
```

Click **Run**, and you'll see values from the accelerometer printed in the Python shell area (**Figure 7-28**). Unlike the values from your previous program, these will print continuously. To stop them printing, click the red **Stop** button to stop the program.

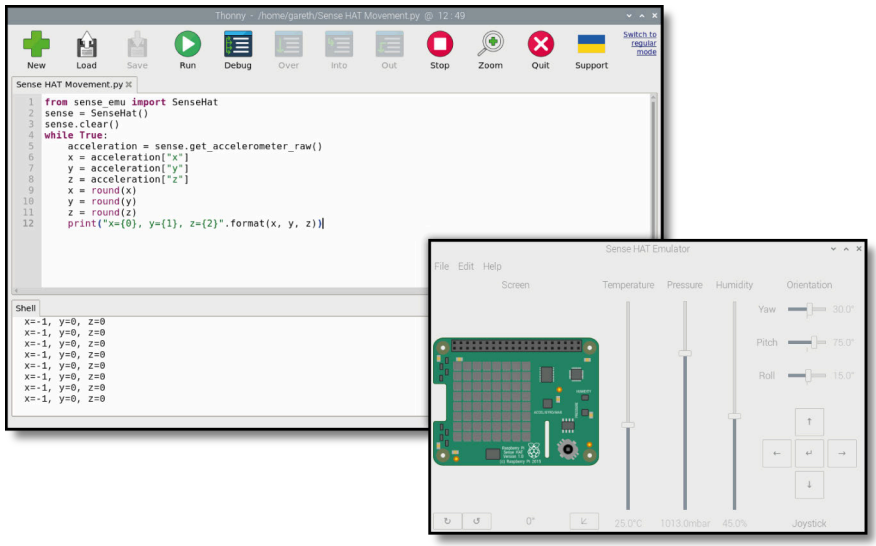


Figure 7-28 Accelerometer readings rounded to the nearest whole number

You may have noticed that the accelerometer is telling you that one of the axes — the Z axis, if your Raspberry Pi is flat on the table — has an acceleration value of 1.0 gravities (1G), yet the Sense HAT isn't moving. That's because it's detecting the Earth's gravitational pull. Gravity is the force that is pulling the Sense HAT down towards the centre of the Earth, and the reason why, if you knock it off your desk, it'll fall to the floor.

With your program running, try carefully picking the Sense HAT and your Raspberry Pi up and rotating them around — but make sure not to dislodge any of the cables! With the Raspberry Pi's network and USB ports pointing to the floor, you'll see the values change so the Z axis reads 0G and the X axis now reads 1G. Turn it again so the HDMI and power ports are pointing to the floor, and now you'll see that it's the Y axis that reads 1G. If you do the opposite and orient the Raspberry Pi so that the HDMI port is pointing at the ceiling, you'll see -1G on the Y axis instead.

Using the knowledge that the Earth's gravity is roughly 1G, along with your understanding of the spatial axes, you can use readings from the accelerometer to figure out which way is down — and, likewise, which way is up. You can also use it to detect movement: try carefully shaking the Sense HAT and

Raspberry Pi, and watch the numbers as you do. The harder you shake, the greater the acceleration.

When you're using `sense.get_accelerometer_raw()`, you're telling the Sense HAT to turn off the other two sensors in the IMU — the gyroscopic sensor and the magnetometer — and return data purely from the accelerometer. You can do the same thing with the other sensors too.

Find the line `acceleration = sense.get_accelerometer_raw()` and change it to:

```
orientation = sense.get_gyroscope_raw()
```

Change the word `acceleration` on all three following lines to `orientation`. Click **Run**, and you'll see the orientation of the Sense HAT for all three axes, rounded to the nearest whole number. Unlike the last time you checked orientation, though, this time the data is only coming from the gyroscope, without using the accelerometer or magnetometer. This can be useful if you want to know the orientation of a moving Sense HAT on the back of a robot, for example, without the movement confusing things. It's also helpful if you're using the Sense HAT near a strong magnetic field.

Stop your program by clicking on the red **Stop** button. To use the magnetometer, delete everything from your program except for the first four lines, then type the following below the `while True` line:

```
north = sense.get_compass()
print(north)
```

Run your program and you'll see the direction of magnetic north printed repeatedly to the Python shell area. Carefully rotate the Raspberry Pi, and you'll see the heading change as the Sense HAT's orientation relative to north shifts: you've built a compass! If you have a magnet — a fridge magnet will do — try moving it around the Sense HAT to see what that does to the magnetometer's readings.

CHALLENGE: AUTO-ROTATE



Using what you've learned about the LED matrix and the inertial measurement unit's sensors, can you write a program that rotates an image depending on the position of the Sense HAT?

Joystick control

The Sense HAT's joystick, found in the bottom-right corner, may be small, but it's surprisingly powerful: as well as being able to recognise inputs in four directions — up, down, left, and right — it also has a fifth input, accessed by pressing down on the joystick and using it like a push-button switch.



WARNING!

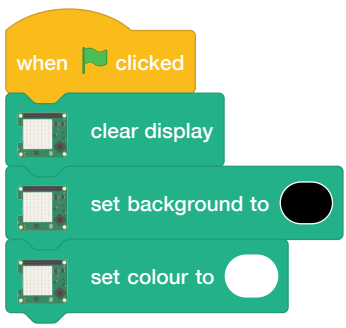
The Sense HAT joystick should only be used if you've fitted the spacers as described at the start of this chapter. Without the spacers, pushing down on the joystick can flex the Sense HAT board and damage both the Sense HAT and Raspberry Pi's GPIO header.

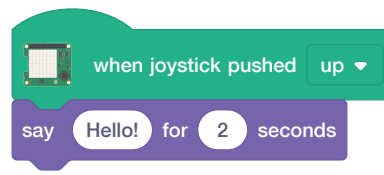
Joystick control in Scratch

Start a new program in Scratch with the **Raspberry Pi Sense HAT** extension loaded. As before, drag a when clicked block underneath it, followed by dragging and editing a clear display block underneath. Then add a set background to black and a set colour to white block.

In Scratch, the Sense HAT's joystick maps to the cursor keys on the keyboard: pushing the joystick up is equivalent to pressing the up-arrow key, and pushing it down is the same as pushing the down-arrow key. Pushing it left or right does the same as the left and right arrow keys. Pressing the joystick down like a push-button switch is equivalent to pressing the **ENTER** key.

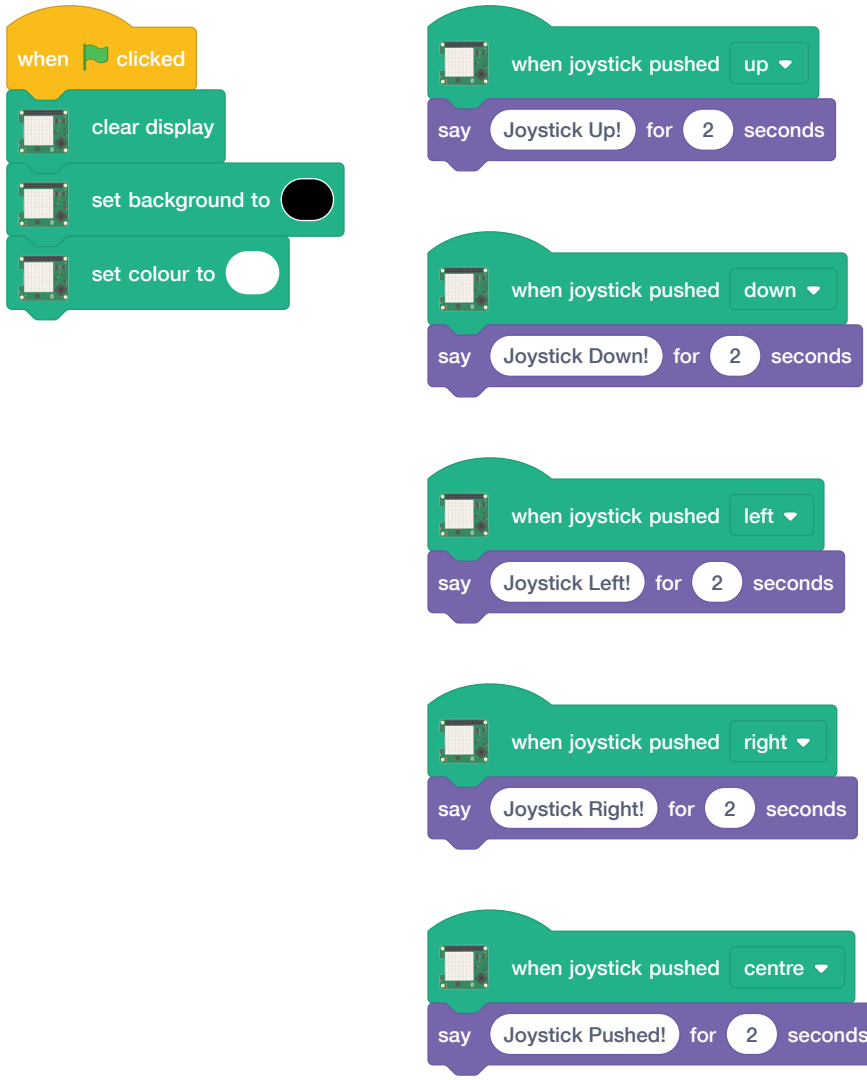
Drag a when joystick pushed up block onto your code area. Then, to give it something to do, drag a say Hello! for 2 seconds block under it.





Push the joystick upwards and you'll see the Scratch cat say a cheery "Hello!" Joystick control is only available on the physical Sense HAT. When using the Sense HAT Emulator, use the corresponding keys on your keyboard to simulate joystick presses instead.

Next, change say Hello! to say Joystick Up!, and add **Events** and **Looks** blocks until you have something to say for each of the five ways the joystick can be pressed. Try pushing the joystick in various directions, and watch as messages appear!



FINAL CHALLENGE

Can you use the Sense HAT's joystick to control a Scratch sprite on the stage area? Can you make it so that if the sprite collects another sprite, representing an object, the Sense HAT's LED matrix displays a message?

Joystick control in Python

Start a new program in Thonny and save it as Sense HAT Joystick. Begin with the usual three lines that set up the Sense HAT and clear the LED matrix — remembering to use `sense_emu` (in place of `sense_hat`) if you're using the emulator:

```
from sense_hat import SenseHat
sense = SenseHat()
sense.clear()
```

Next, set up an infinite loop:

```
while True:
```

Then tell Python to listen for inputs from the Sense HAT joystick with the following line, which Thonny will automatically indent for you:

```
    for event in sense.stick.get_events():
```

Finally, add the following line — which, again Thonny will indent for you — to actually do something when a joystick press is detected:

```
        print(event.direction, event.action)
```

Click **Run**, and try pushing the joystick in various directions. You'll see the direction you've chosen printed to the Python shell area: up, down, left and right; and middle for when you've pushed the joystick down like a push-button switch.

You'll also see that you're given two events each time you push the joystick once: one event, `pressed`, for when you first push in a direction; the other event, `released`, for when the joystick returns to centre.

You can use this in your programs: think of a character in a game, which could be made to start moving when the joystick is pressed in one direction, and then made to stop as soon as it's released.

You can also use the joystick to trigger functions, rather than being limited to using a `for` loop. Delete everything below `sense.clear()`, and type the following:

```
def red():
    sense.clear(255, 0, 0)

def blue():
```

```
    sense.clear(0, 0, 255)

def green():
    sense.clear(0, 255, 0)

def yellow():
    sense.clear(255, 255, 0)
```

These functions change the whole Sense HAT LED matrix to a single colour: red, blue, green, or yellow. This is going to make observing that your program works extremely easy! To actually trigger them, you need to tell Python which function goes with which joystick input. Type the following lines:

```
sense.stick.direction_up = red
sense.stick.direction_down = blue
sense.stick.direction_left = green
sense.stick.direction_right = yellow
sense.stick.direction_middle = sense.clear
```

Finally, the program needs an infinite loop — known as the *main* loop — in order to keep running. This means you'll have to keep watching for joystick inputs, rather than just running through the code you've written once and then quitting. Type the following two lines:

```
while True:
    pass
```

Your completed program should look like this:

```
from sense_hat import SenseHat
sense = SenseHat()
sense.clear()

def red():
    sense.clear(255, 0, 0)

def blue():
    sense.clear(0, 0, 255)

def green():
    sense.clear(0, 255, 0)

def yellow():
    sense.clear(255, 255, 0)

sense.stick.direction_up = red
```

```
sense.stick.direction_down = blue
sense.stick.direction_left = green
sense.stick.direction_right = yellow
sense.stick.direction_middle = sense.clear

while True:
    pass
```

Click **Run**, and try moving the joystick: you'll see the LEDs light up in glorious colour. To turn the LEDs off, push the joystick like a push-button. The `middle` direction is set to use the `sense.clear()` function to turn them all off. Congratulations: you can capture input from the joystick!



FINAL CHALLENGE

Can you use what you've learned to draw an image to the screen, then have it rotated in whatever direction the joystick is pushed? Can you make the middle input switch between more than one picture?

Scratch project: Sense HAT Sparkler

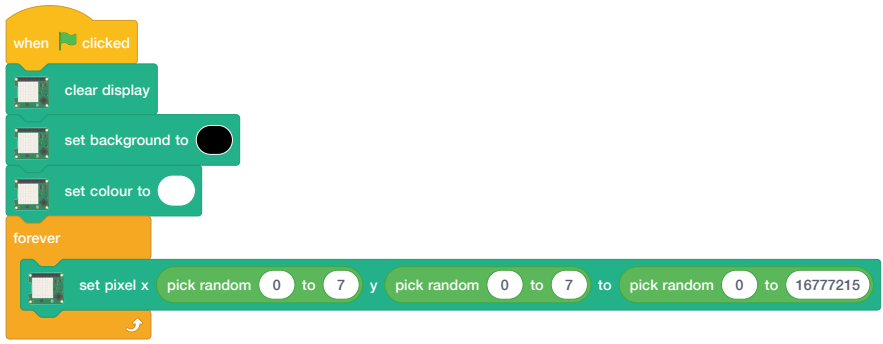
Now you know your way around the Sense HAT, it's time to put everything you've learned together to build a heat-sensitive sparkler — a device which is at its happiest when it's cold and which gradually slows down the hotter it gets.

Start a new Scratch project and add the Raspberry Pi Sense HAT extension, if not already loaded. As always, begin with four blocks: start with when clicked with a clear display block underneath. You'll need a set background to black, and a set colour to white, remembering you'll have to change the colours from their default settings.

Start by creating a simple, but artistic, sparkler. Drag a forever block onto the code area, then fill it with a set pixel x 0 y 0 to colour block. Rather than using set numbers, fill in each of the x, y, and colour sections of that block with a pick random 1 to 10 **Operators** block.

The values 1 to 10 aren't very useful here, so you need to do some editing. The first two numbers in the set pixel block are the X and Y coordinates of the pixel on the LED matrix, which means they should be numbered between 0 and 7. Change the first two blocks to read pick random 0 to 7.

The next section is the colour the pixel should be set to. When you're using the colour selector, the colour you choose is shown directly in the script area; internally, though, the colours are represented by a number, and you can use the number directly. Edit the last **Pick random** block to read pick random 0 to 16777215.



Click the green flag and you'll see the LEDs on the Sense HAT begin to light up in random colours (**Figure 7-29**). Congratulations: you've made an electronic sparkler!

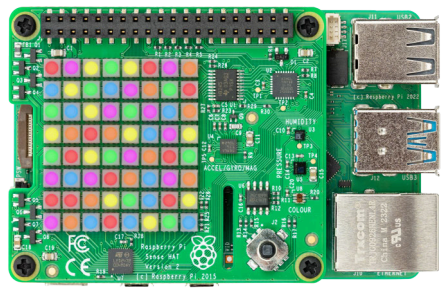
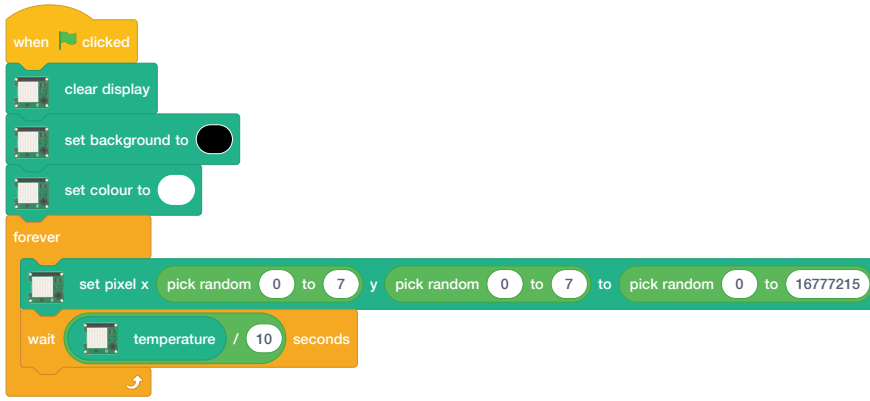


Figure 7-29 Lighting the pixels in random colours

The sparkler isn't very interactive. To change that, start by dragging a wait 1 seconds block so it's under the set pixel block but within the forever block. Drag a / **Operators** block over the 1, then type 10 in its second space. Finally, drag a temperature block over the first space in the divide **Operator** block.



Click the green flag and you'll notice — unless you are somewhere very cold — that the sparkler is considerably slower than before. That's because you've created a temperature-dependent delay: the program now waits *the current temperature divided by 10* number of seconds before each loop. If the temperature in your room is 20°C, the program will wait two seconds before looping; if the temperature is 10°C, it'll wait one second; if it's under 10°C, it'll wait less than a second.

If your Sense HAT is reading a negative temperature — below 0°C, the freezing point of water — it'll try to wait less than 0 seconds. Because that's impossible — without inventing time travel, anyway — you'll see the same effect as though it was waiting 0 seconds. Congratulations: you have learned how to integrate the Sense HAT's features into programs of your own!

Python project: Sense HAT Tricorder

Now you know your way around the Sense HAT, it's time to put everything you've learned together to build a tricorder — a device immediately familiar to fans of a certain science-fiction franchise. The fictional tricorder uses different sensors to report on its surroundings.

Start a new project in Thonny and save it as **Tricorder.py**, then start with the lines you need to use every time you begin a Sense HAT program in Python, remembering to use `sense_emu` if you're using the Sense HAT emulator:

```
from sense_hat import SenseHat
sense = SenseHat()
sense.clear()
```

Next, you need to start defining functions for each of the Sense HAT's sensors. Start with the inertial measurement unit by typing:

```
def orientation():
    orientation = sense.get_orientation()
    pitch = orientation["pitch"]
    roll = orientation["roll"]
    yaw = orientation["yaw"]
```

Because you're going to be scrolling the results from the sensor across the LEDs, it makes sense to round the numbers so you don't end up waiting for dozens of decimal places. Rather than whole numbers, round them to one decimal place by typing the following:

```
    pitch = round(pitch, 1)
    roll = round(roll, 1)
    yaw = round(yaw, 1)
```

Finally, you need to tell Python to scroll the results to the LEDs, so the tricorder works as a hand-held device without needing to be connected to a monitor or TV:

```
    sense.show_message("Pitch {0}, Roll {1}, Yaw {2}".
                       format(pitch, roll, yaw))
```

Now that you have a full function for reading and displaying the orientation from the IMU, you need to create similar functions for each of the other sensors. Start with the temperature sensor:

```
def temperature():
    temp = sense.get_temperature()
    temp = round(temp, 1)
    sense.show_message("Temperature: %s degrees Celsius" % temp)
```

Look carefully at the line which prints the result to the LEDs: the `%s` is known as a *placeholder*, and gets replaced with the content of the variable `temp`. Using this, you can format the output nicely with a label, 'Temperature:', and a unit of measurement, 'degrees Celsius,' which makes your program a lot more friendly.

Next, define a function for the humidity sensor:

```
def humidity():
    humidity = sense.get_humidity()
    humidity = round(humidity, 1)
    sense.show_message("Humidity: %s percent" % humidity)
```

Then the pressure sensor:

```
def pressure():
    pressure = sense.get_pressure()
    pressure = round(pressure, 1)
    sense.show_message("Pressure: %s millibars" % pressure)
```

Finally, define a function for the compass reading from the magnetometer:

```
def compass():
    for i in range(0, 10):
        north = sense.get_compass()
    north = round(north, 1)
    sense.show_message("North: %s degrees" % north)
```

The short `for` loop in this function takes ten readings from the magnetometer to ensure that it has enough data to give you an accurate result. If you find that the reported value keeps shifting, try extending it to 20, 30, or even 100 loops to improve the accuracy further.

Your program now has five functions, each of which takes a reading from one of the Sense HAT's sensors and scrolls them across the LEDs. It needs a way to choose which sensor you want to use, and the joystick is perfect for that.

Type the following:

```
sense.stick.direction_up = orientation
sense.stick.direction_right = temperature
sense.stick.direction_down = compass
sense.stick.direction_left = humidity
sense.stick.direction_middle = pressure
```

These lines assign a sensor to each of the five possible directions on the joystick. `Up` reads from the orientation sensor; `down` reads from the magnetometer; `left` reads from the humidity sensor; `right` from the temperature sensor; and pressing the stick in the `middle` reads from the pressure sensor.

Finally, you need a main loop so the program will keep listening out for joystick presses and not just immediately quit. At the very bottom of your program, type the following:

```
while True:
    pass
```

Your completed program should look like this:

```
from sense_hat import SenseHat
sense = SenseHat()
sense.clear()

def orientation():
    orientation = sense.get_orientation()
    pitch = orientation["pitch"]
    roll = orientation["roll"]
    yaw = orientation["yaw"]

    pitch = round(pitch, 1)
    roll = round(roll, 1)
    yaw = round(yaw, 1)

    sense.show_message("Pitch {0}, Roll {1}, Yaw {2}".
                       format(pitch, roll, yaw))

def temperature():
    temp = sense.get_temperature()
    temp = round(temp, 1)
    sense.show_message("Temperature: %s degrees Celsius" % temp)

def humidity():
    humidity = sense.get_humidity()
    humidity = round(humidity, 1)
    sense.show_message("Humidity: %s percent" % humidity)

def pressure():
    pressure = sense.get_pressure()
    pressure = round(pressure, 1)
    sense.show_message("Pressure: %s millibars" % pressure)

def compass():
    for i in range(0, 10):
        north = sense.get_compass()
    north = round(north, 1)
    sense.show_message("North: %s degrees" % north)

sense.stick.direction_up = orientation
sense.stick.direction_right = temperature
sense.stick.direction_down = compass
sense.stick.direction_left = humidity
sense.stick.direction_middle = pressure

while True:
    pass
```

Click **Run**, and try moving the joystick to take a reading from one of the sensors (**Figure 7-30**). When it has finished scrolling the result, press a different direction. Congratulations: you've built a hand-held tricorder that would make the United Federation of Planets proud!

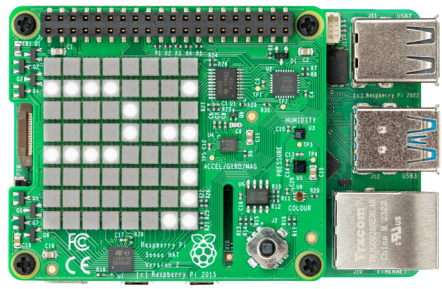
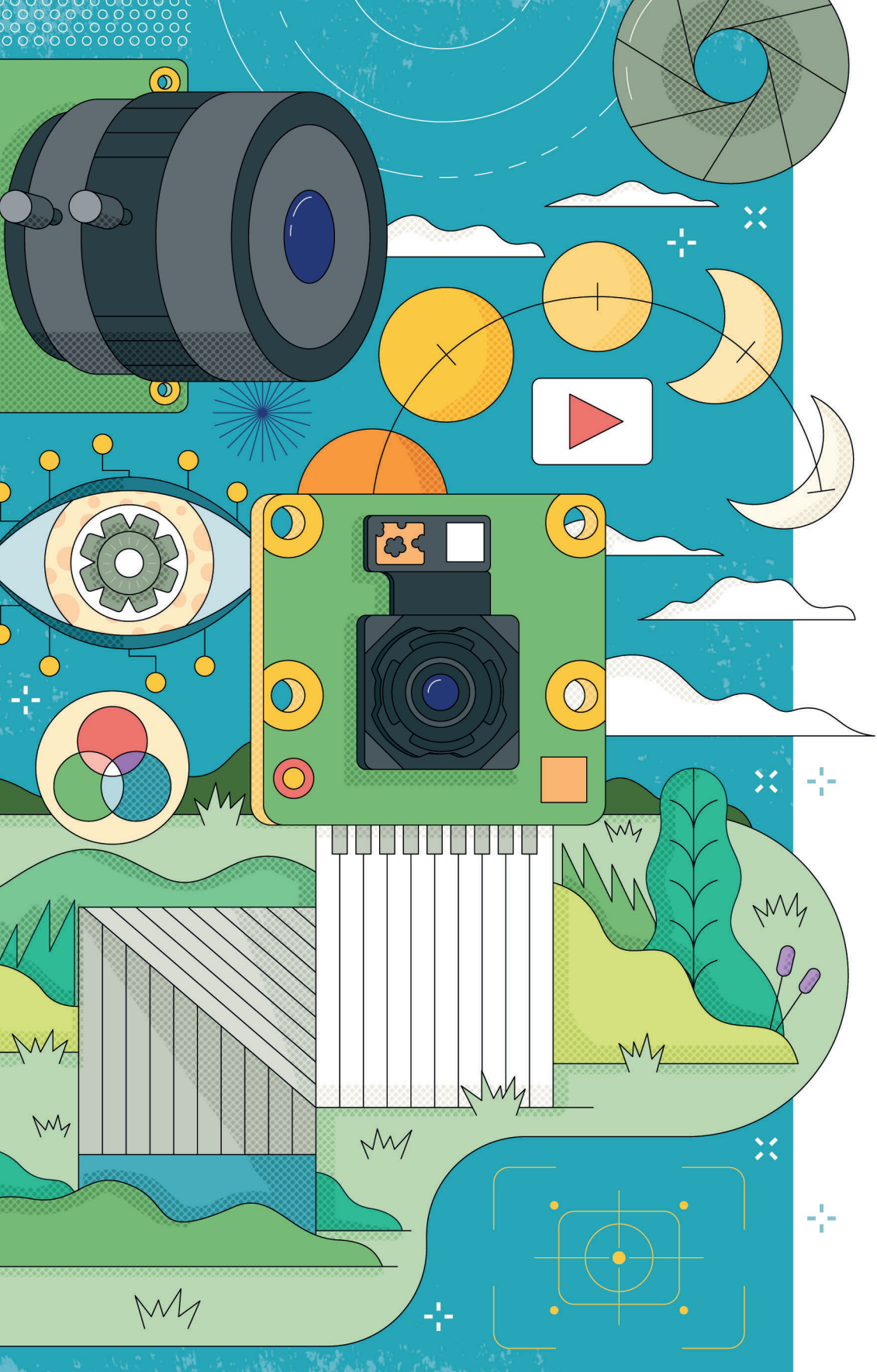


Figure 7-30 Each reading scrolls across the display

For more Sense HAT projects, including an example of how to use the colour sensor on Sense HAT V2, follow the links in Appendix D, *Further reading*.



Chapter 8

Raspberry Pi Camera Modules

Connecting a Camera Module or HQ Camera to your Raspberry Pi enables you to take high-resolution photos and shoot videos, and create amazing computer vision projects.

If you've ever wanted to build something that can see for itself — known in the robotics field as *computer vision* — then Raspberry Pi's optional Camera Module 3 (**Figure 8-1**), High Quality Camera (HQ Camera), or Global Shutter Camera are must-have accessories. Small, square circuit boards with a thin ribbon cable, the three Camera Modules connect to the Camera Serial Interface (CSI) port on your Raspberry Pi and provide high-resolution still images and moving video signals, which can be used as-is, or integrated into your own programs.

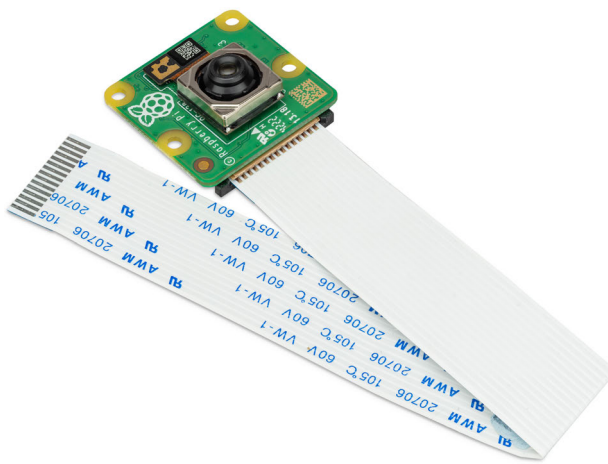


Figure 8-1 Raspberry Pi Camera Module 3



RASPBERRY PI 400

The Raspberry Pi Camera Modules are not compatible with the Raspberry Pi 400 desktop computer. You can use USB webcams as an alternative, but you won't be able to use the software tools shown in this chapter with a Raspberry Pi 400.

Camera variants

There are multiple types of Raspberry Pi Camera Module available, and the model you need will depend on what you're capturing: the standard Camera Module 3, the 'NoIR' version, the High Quality (HQ) Camera, and the Global Shutter Camera. If you want to take normal pictures and video in well-lit environments, you should use the standard Camera Module 3, or the Camera Module 3 Wide for a wider field of view.

If you want to be able to swap lenses and are looking for the best picture quality, use the HQ Camera Module. The NoIR Camera Module 3 — so called because it has no infrared (IR) filter — is designed for use with infrared light sources to take pictures and video in total darkness, and is also available in a wide-angle version. If you're building a nest box, security camera, or other project involving night vision, you want the NoIR version — but remember to buy an infrared light source at the same time! Finally, the Global Shutter Camera captures the entire image at once, rather than line-by-line, making it suitable for high-speed photography and computer vision work.

Raspberry Pi Camera Module 3

The Raspberry Pi Camera Module 3, in both standard and NoIR versions, is built around a Sony IMX708 image sensor. This is a *12 megapixel sensor*, meaning it captures images with up to 12 million pixels in them. This results in a maximum image size of 4608 pixels wide by 2592 pixels tall. There are two lens options for Raspberry Pi Camera Module 3: the standard lens, which captures a field of view 75 degrees wide; and the wide-angle lens, which has a 120 degree field of view.

In addition to still photographs, the Raspberry Pi Camera Module 3 can capture video footage at Full HD resolution (1080p) at a rate of 50 frames per second (50 fps). For smoother motion, or even to create a slow-motion effect, the camera can be set to capture at a higher frame rate by lowering the resolution: you get 100 fps at 720p resolution and 120 fps at 480p (VGA) resolution. The module has one last trick up its sleeve compared to earlier versions: it offers *autofocus*, meaning it can automatically adjust the focal point of the lens for close-up or distant subjects.

Raspberry Pi High Quality Camera

The High Quality Camera uses a 12.3 megapixel Sony IMX477 sensor. This sensor is larger than the one in the standard and NoIR Camera Modules — meaning it can gather more light which leads to higher-quality images. Unlike the Camera Modules, though, the HQ Camera doesn't include a lens, without which it can't take any pictures or videos. You can use any lens with a C or CS mount, and you can use other lens mounts with an appropriate C or CS mount adapter. An alternative version of the High Quality Camera is available for use with M12-mount lenses.

Raspberry Pi Global Shutter Camera

The Global Shutter Camera uses a 1.6 megapixel Sony IMX296 sensor. Although it provides lower resolution than either the standard Raspberry Pi Camera Module or the High Quality Camera, its ability to capture the entire image at once means it excels at capturing rapidly-moving subjects without the distortion you can get with a rolling shutter camera. Like the High Quality Camera, it comes without a lens, and supports the same C and CS mount lenses; unlike the High Quality Camera, there is no M12-mount version at the time of writing.

Raspberry Pi Camera Module 2

The earlier Raspberry Pi Camera Module 2 and its NoIR variant are based on a Sony IMX219 image sensor. This is an 8 megapixel sensor, so it can take pictures with up to 8 million pixels in them measuring 3280 pixels wide by 2464 tall. Along with still images, the Camera Module can capture Full HD resolution (1080p) video at 30 frames per second (30 fps) with higher frame rates available at lower resolutions: 60fps for 720p video footage and up to 90 fps for 480p (VGA) footage.

RASPBERRY PI ZERO AND RASPBERRY PI 5



All models of the Raspberry Pi Camera Module are compatible with Raspberry Pi Zero 2 W; newer versions of the original Raspberry Pi Zero and Zero W; and Raspberry Pi 5. If you're using a Raspberry Pi 5, you'll need a different ribbon cable than the one you might have used with Raspberry Pi 4 and earlier models.

Ask your favourite Authorised Reseller for a suitable cable: the wider end goes into the camera, while the narrower end goes into Raspberry Pi.

Installing the camera

Like any hardware add-on, the Camera Module or HQ Camera should only be connected to (or disconnected from) your Raspberry Pi when the power is off, and the power cable is unplugged. If your Raspberry Pi is turned on, choose **Shutdown** from the Raspberry Pi menu, wait for it to power off, and unplug it.

In most cases, the ribbon cable will already be connected to the Camera Module or HQ Camera. If it isn't, turn your camera board upside-down so the sensor is on the bottom, and look for a flat plastic connector. Carefully hook your fingernails around the sticking-out edges and pull outwards until the connector pulls part-way out. Slide the ribbon cable, with the silver or gold edges downwards and the plastic reinforcement facing upwards, under the flap you just pulled out, then push the flap gently back into place with a click (**Figure 8-2**). If the cable is installed properly, it will lie straight and won't come out if you give it a gentle tug. If it's not seated correctly, pull the flap out and try again.

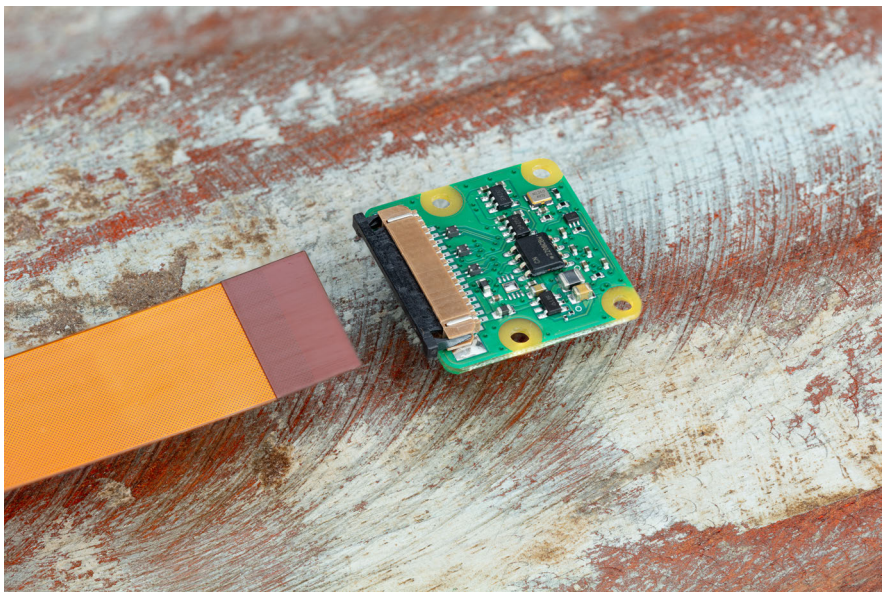


Figure 8-2 Connecting the ribbon cable to the Camera Module

Install the other end of the cable the same way. Find the lower of the two camera/display ports, marked 'CAM/DISP 0,' on Raspberry Pi 5 or the single camera port on Raspberry Pi 4, Raspberry Pi Zero 2 W, and earlier models, and pull the small plastic cover up. If your Raspberry Pi is installed in a case, you might find it easier to remove it first.

With Raspberry Pi 5 positioned so the GPIO header is to the right and the HDMI ports are to the left, slide the ribbon cable in so the silver or gold edges are facing you and the plastic reinforcement is on the opposite side (**Figure 8-3**). Then gently push the flap back into place.

For Raspberry Pi 4 and earlier models, the ribbon cable should be the other way around, with the plastic reinforcement facing towards you and the silver or gold edges on the opposite side. If you are using a Raspberry Pi Zero 2 W or an older Raspberry Pi Zero, the silver or gold edges should be pointing down towards the table and the plastic up towards the ceiling. If the cable is installed properly, it'll sit straight and won't come out if you give it a gentle tug. If it's not seated correctly, pull the flap out and try again.

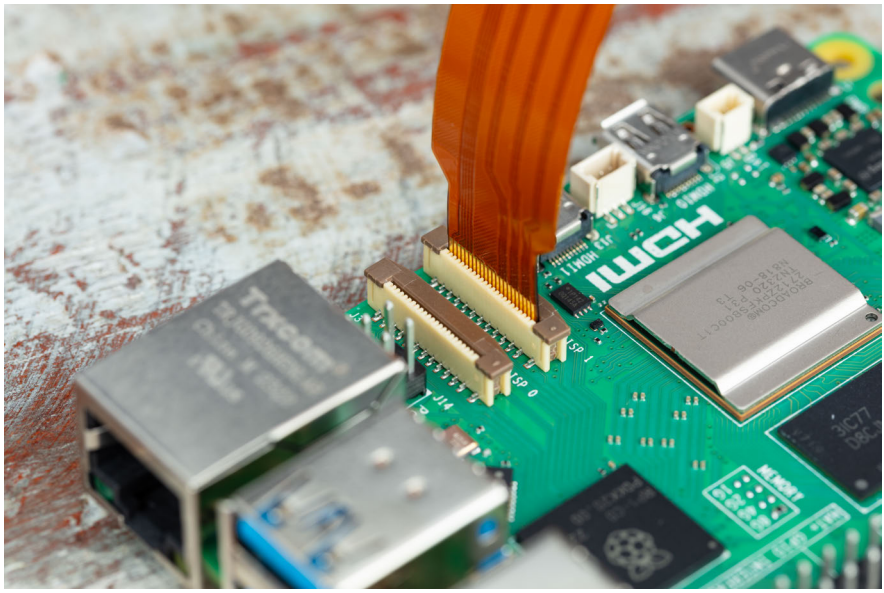


Figure 8-3 Connecting the ribbon cable to the Camera/CSI port on Raspberry Pi

The Camera Module may come with a small piece of blue plastic covering the lens to protect it from scratches during manufacturing, shipping, and installation. Find the small flap of loose plastic and pull the cover gently off the lens to get the camera ready for use.

Connect the power supply back to Raspberry Pi and let it load Raspberry Pi OS.

ADJUSTING FOCUS

All versions of Raspberry Pi Camera Module 3 include a motorised autofocus system, which can adjust the focal point of the lens between close-up and distant objects. Raspberry Pi Camera Module 2 uses a lens which includes limited manual focus adjustment. It is packaged with a small tool for turning the lens and adjusting the focus.

Testing the camera

To confirm that your Camera Module or HQ Camera is properly installed, you can use the **rpicam** tools. These are designed to capture images from the camera using Raspberry Pi's *command-line interface (CLI)*.

Unlike the programs you've been using so far, you won't find the rpicam tools in the menu. Instead, click on the Raspberry Pi icon to load the menu, choose the **Accessories** category, and click on **Terminal**. A black window with green and blue text will appear (**Figure 8-4**): this is the *terminal*, which allows you to access the command-line interface.

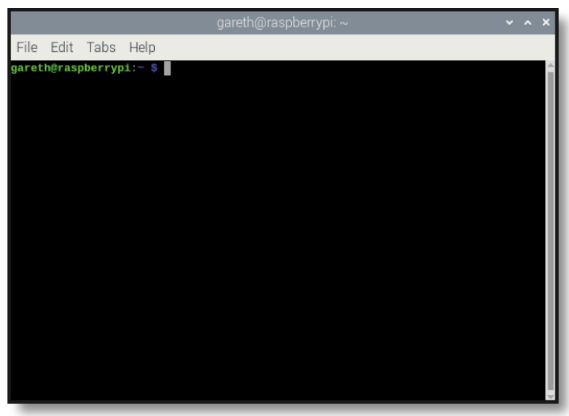


Figure 8-4 Open a Terminal window to enter commands

To capture an image with the camera, type the following into the terminal:

```
rpicam-still -o test.jpg
```

As soon as you hit **ENTER**, you'll see a window with a view of what the camera sees appear on-screen (**Figure 8-5**). This is called the *live preview* and, unless you tell `rpicam-still` otherwise, it will last for five seconds. After those five seconds are up, the camera will capture a single still picture and save it in

your **home/<username>** folder with the name **test.jpg**. If you want to capture another, type the same command again — but make sure you change the output file name after the **-o**, or you'll save over the top of your first picture!

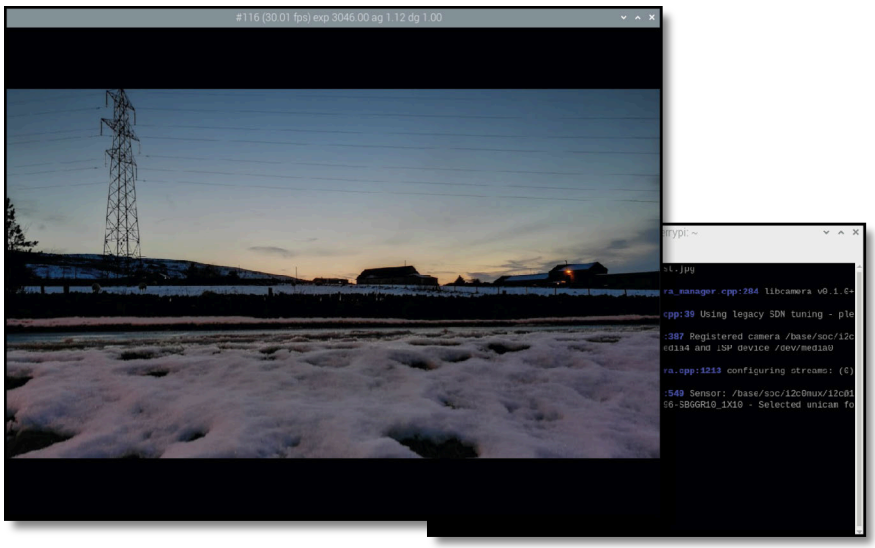


Figure 8-5 The live preview from the camera

If the live preview was upside-down, you need to tell **rpicam-still** that the camera is rotated. The Camera Module is designed to have the ribbon cable coming out of the bottom edge. If it's coming out of the sides or the top, as with some third-party camera mount accessories, you can rotate the image by 90, 180, or 270 degrees using the **--rotation** switch. For a camera mounted with the cable coming out of the top, use the following command:

```
rpicam-still --rotation 180 -o test.jpg
```

If the ribbon cable is coming out of the right-hand edge, use a rotation value of 90 degrees; if it's coming out of the left-hand edge, use 270 degrees. If your original capture was at the wrong angle, try another using the **--rotation** switch to correct it.

To see your picture, open the **File Manager** from the **Accessories** category of the Raspberry Pi menu: the image you've taken, called **test.jpg**, will be in your **home/<username>** folder. Find it in the list of files, then double-click the picture to load it in an image viewer (**Figure 8-6**). You can also attach the image to emails, upload it to websites via the browser, or drag it to an external storage device.

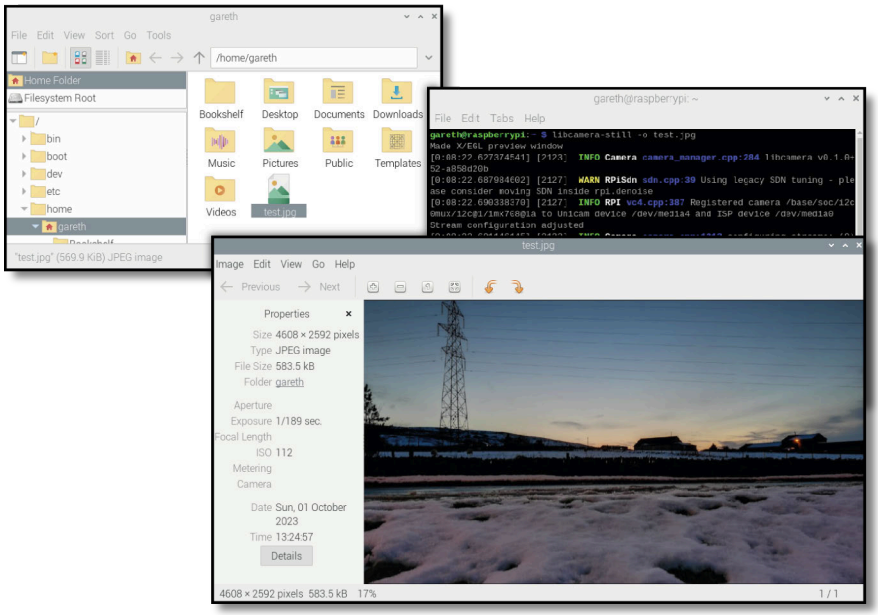


Figure 8-6 Opening the captured image

Raspberry Pi Camera Module 3 includes the ability to adjust the focal point of the image using a motorised autofocus system. This is enabled by default: when you capture a picture, the camera module will automatically adjust its focus to make the picture as clear as possible using what is known as *continuous autofocus*.

As the name suggests, continuous autofocus constantly adjusts the focal point right up until the moment the picture is captured. If you're capturing multiple pictures, or shooting video, it will continue to adjust the focus as you work. If something moves between the camera and the subject, the camera will switch its focus automatically.

There are other autofocus modes you can use if continuous autofocus isn't delivering the results you need. You can read about these in the *Advanced camera settings* reference section at the end of this chapter.

Capturing video

Your Camera Module isn't restricted to just capturing still images: it can also record video using a tool called `rpicam-vid`.



MAKE ROOM, MAKE ROOM

Recording video can take a lot of storage space. If you're planning to record a lot of video, make sure you have a large microSD card. You could also invest in a USB flash drive or other external storage.

By default, the rpicam tools will save files to whatever folder they're launched from. So make sure to change directories so you are saving to your chosen storage device. You can read about changing directories in the terminal in Appendix C, *The command-line interface*.

To record a short video, type the following in the terminal:

```
rpicam-vid -t 10000 -o test.h264
```

As before, you'll see the preview window appear. This time, though, instead of counting down and capturing a single still image, the camera will record ten seconds of video to a file. When the recording is finished, the preview window will automatically close.

If you want to capture a longer video, change the number after `-t` to the length of recording you're after in milliseconds. For example, to take a ten-minute recording you would type:

```
rpicam-vid -t 600000 -o test2.h264
```

To play your video back, find it in the file manager and double-click the video file to load it in the VLC video player (**Figure 8-7**). Your video will open and begin playing.

Video captured by `rpicam-vid` comes in a format called a *bitstream*. The way a bitstream works is a bit different from the video files you might be used to. Usually, files contain multiple parts: the video, any audio captured alongside the video, timecode information about when each frame should be displayed, and additional information known as *metadata*. A bitstream is different. It has none of this: it's just pure video data.

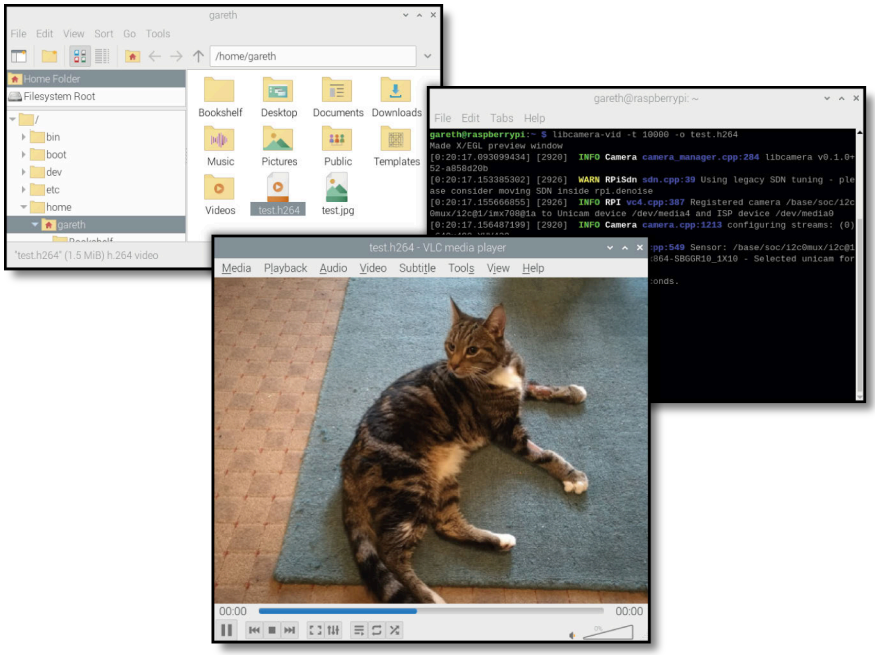


Figure 8-7 Opening the captured video

To ensure your video files play back on as many software platforms as possible, including software running on computers other than Raspberry Pi, you'll need to process it into a *container*. On Raspberry Pi 5, you can output to the MP4 container format directly by specifying the **mp4** file extension:

```
rpicam-vid -t 600000 -o test2.mp4
```

On earlier models of Raspberry Pi, you'll need to capture some missing information: *frame timing*. In the terminal, record a new video — but this time tell `rpicam-vid` to record timing information into a file called **timestamps.txt**:

```
rpicam-vid -t 10000 --save-pts timestamps.txt -o test-time.h264
```

When you open the video folder in the file manager, you'll see two files: the video bitstream, **test-time.h264**, and the **timestamps.txt** file (**Figure 8-8**).

To combine these two files into a single container suitable for playback on other devices, use the **mkvmerge** tool. This takes the video, merges it with the timestamps, and outputs a video container file known as a *Matroska Video file* or *MKV*.

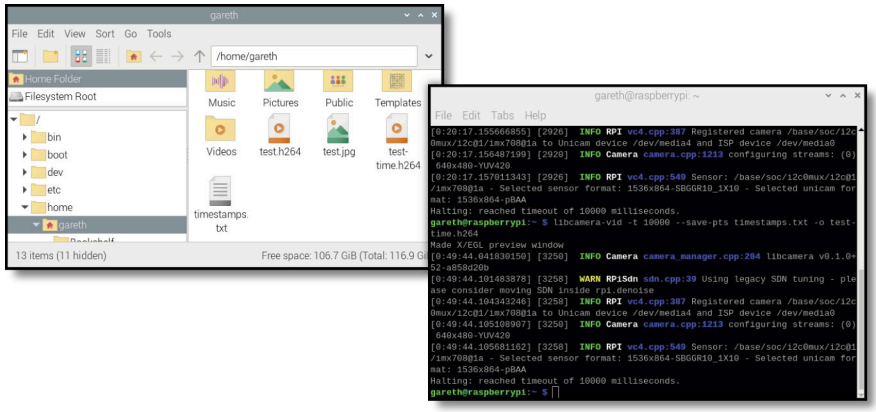


Figure 8-8 A video file with a separate timestamp file

In the command line, type (the \ is a special character that lets you break the command across two lines):

```
mkvmerge --timecodes 0:timestamps.txt test-time.h264 \
   -o test-time.mkv
```

You'll now have a third file, **test-time.mkv**. Double-click this file in the file manager to load it in VLC, and you'll see the video you recorded play back without skipping or dropping any frames. If you want to transfer the video to a removable drive for playback on another computer you only need the MKV file, and can delete the H264 and TXT files.

Always remember to save timestamps with your video if you want to create a file that will play back properly on as many computers as possible. It's not easy to go back and create them after recording!

Time-lapse photography

There's another trick your camera module can pull off: *time-lapse photography*. In time-lapse photography, still pictures are taken over a period of time at regular intervals, to capture changes that happen too slowly for the naked eye to observe. It's a great tool for watching how the weather changes over the period of a day, or how a flower grows and blooms over a period of months. You can even use time-lapse techniques to make your own stop-motion animation!

To start a time-lapse photography session, type the following in the terminal to make a new directory and change into it. This helps keep all the files you capture neatly in one place:

```
mkdir timelapse
cd timelapse
```

Then begin capturing by typing:

```
rpicam-still --width 1920 --height 1080 -t 100000 \
   --timelapse 10000 -o %05d.jpg
```

The output filename is a little different this time around: `%05d` tells `rpicam-still` to use numbers, starting at 00000 and counting upwards, as the filename. Without this, it would automatically overwrite older pictures every time it took a new one, and you'd only have one picture to show for your efforts.

The `--width` and `--height` switches control the *resolution* of the images captured. In this case, we're setting the images to a width of 1920 pixels and a height of 1080 pixels — the same resolution as a Full HD video file.

The `-t` switch acts as it did before, setting up a timer for how long the camera should run. In this case, it's 100,000 milliseconds (100 seconds).

Finally, the `--timelapse` switch tells `rpicam-still` how long to wait between pictures. Here it's set to 10,000 milliseconds (ten seconds). Because it won't take any photos until the first ten seconds have elapsed, you'll get a total of nine photos.

Leave `rpicam-still` running for 100 seconds, then open the timelapse directory in your file manager. You'll see nine individual photos, each labelled with a number started at 00000 (**Figure 8-9**).

To combine these pictures into an animation, use the **ffmpeg** tool. Type:

```
ffmpeg -r 0.5 -i %05d.jpg -r 15 animation.mp4
```

This tells ffmpeg to interpret the pictures you captured as though they were a video running at 0.5 frames per second, and use them to produce an animated video running at 15 frames per second.

Double-click the file **animation.mp4** to play it back in VLC. You'll see each of the photos you took appear one after the other (**Figure 8-10**).

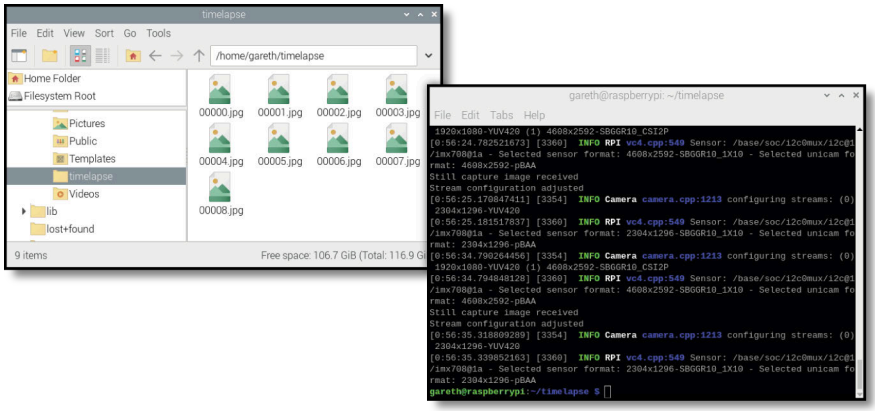


Figure 8-9 Photos taken during a time-lapse session

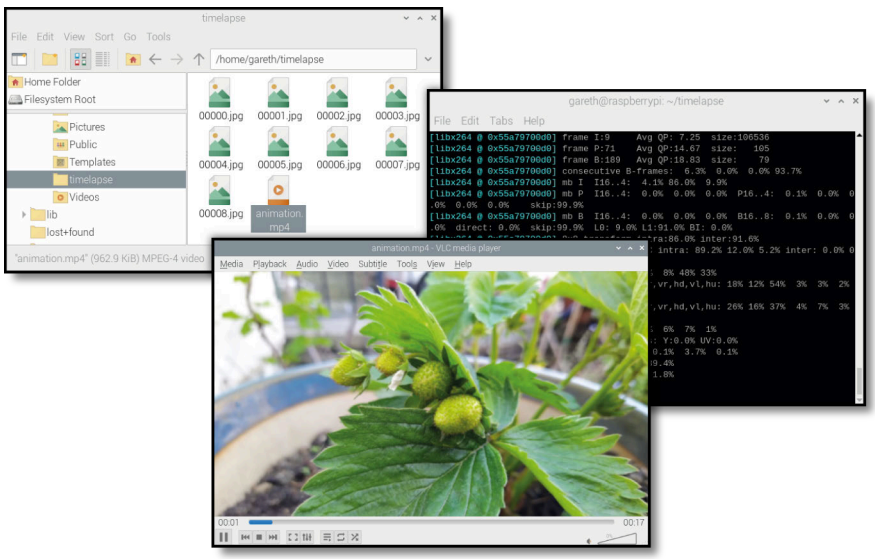


Figure 8-10 Playing back a timelapse animation

To make the animation faster, try changing the input frame rate from 0.5 frames per second to 1 or more; to make it slower, try decreasing it to 0.2 or lower.

Why not try making your own stop-motion video? Position toys in front of the camera and begin a time-lapse session, then move them into new positions just after each photo has been taken. Remember to move your hands out of shot before each photo is taken!

Advanced camera settings

Both `rpicam-still` and `rpicam-vid` support a range of advanced settings, giving you finer control over settings like resolution — the size of the image or video you capture. Higher-resolution images and videos are of a higher quality, but take up a correspondingly larger amount of storage space — so be careful when experimenting!

rpicam-still and rpicam-vid

The settings below can be used with both `rpicam-still` and `rpicam-vid` by adding them to the command you type at the terminal.

`--autofocus-mode`

Configures the autofocus system on Raspberry Pi Camera Module 3. Possible options are: `continuous`, the default mode; `manual`, which disables autofocus entirely; and `auto`, which performs a single autofocus operation when the camera first starts up. This setting has no effect on other Camera Module versions.

`--autofocus-range`

Sets the range for the Raspberry Pi Camera Module 3 autofocus system. If you find the autofocus system is struggling to lock on to your subject, changing the range here can help. Possible options are: `normal`, the default setting; `macro`, which prioritises close-up objects; and `full`, which can focus both extremely close-up and all the way to the horizon.

`--lens-position`

Manually controls the focal point of the lens, for use with the `--autofocus-mode manual` setting. This allows you to set the point where the lens focuses using a unit called *dioptres*, which are equal to one divided by the focal point distance in metres. To set the camera to focus on 0.5 m (50 cm), for example, use `--lens-position 2`; to set it to focus at 10m, use `--lens-position 0.1`. A value of 0.0 represents a focal point of infinity — the farthest the camera can focus.

`--width --height`

Sets the image or video resolution. To capture a Full HD (1920×1080) video, for example, use these arguments with `rpicam-vid`:

```
-t 10000 --width 1920 --height 1080 -o bigtest.h264
```

`--rotation`

Rotates the image from 0 degrees, the default, through 90, 180, and 270 degrees. If your camera is mounted so the ribbon cable isn't coming out of the bottom, this setting will let you capture images and video the right way up.

`--hflip --vflip`

Flips the image or video along the horizontal axis — like a mirror — and/or the vertical axis.

`--sharpness`

Allows you to make the captured image or video look clearer by applying a sharpening filter. Values above 1.0 increase the sharpness above the default; values below 1.0 decrease the sharpness.

`--contrast`

Increases or decreases the contrast of the captured image or video. Values above 1.0 increase the contrast above the default. Values below 1.0 decrease the contrast.

`--brightness`

Increases or decreases the brightness of the image or video. Decreasing the value from the default of 0.0 will make the image darker until you reach the minimum value of -1.0, a completely black image. Increasing the value will make the image lighter until you reach the maximum value of 1.0, a completely white image.

`--saturation`

Increases or decreases the image or video's colour saturation. Decreasing the value from the default 1.0 will make colours more muted until you reach the minimum value of 0.0, a completely greyscale image with no colour at all. Values above 1.0 will make the colours more vibrant.

`--ev`

Sets an exposure compensation value, from -10 to 10, controlling how the camera's gain control works. Usually, the default value of 0 gives best results. If your camera is capturing images that are too dark, you can increase the value; if they're too light, decrease the value.

`--metering`

Sets the metering mode for the automatic exposure and automatic gain controls. The default value, **`centre`**, usually provides the best results; you can override this to choose **`spot`** or **`average`** metering if you'd prefer.

`--exposure`

Switches between the default exposure mode, **`normal`**, and a **`sport`** exposure mode designed for fast-moving subjects.

`--awb`

Allows you to change the automatic white balance algorithm from the default auto mode to: **`incandescent`**, **`tungsten`**, **`fluorescent`**, **`indoor`**, **`daylight`**, or **`cloudy`**.

rpicam-still

The following options are available in **`rpicam-still`**:

`-q`

Sets the quality of the captured JPEG image, from 0 to 100 where 0 is minimum quality and the smallest file size and 100 is maximum quality and the largest file size. The default quality is 93.

`--datetime`

Uses the current date and time — in the format two-digit month, two-digit day, minutes, hours, seconds — as the output filename. Use instead of **`-o`**.

`--timestamp`

Similar to **`--datetime`**, but sets the output filename to the number of seconds since the start of 1970 — known as the *UNIX epoch*.

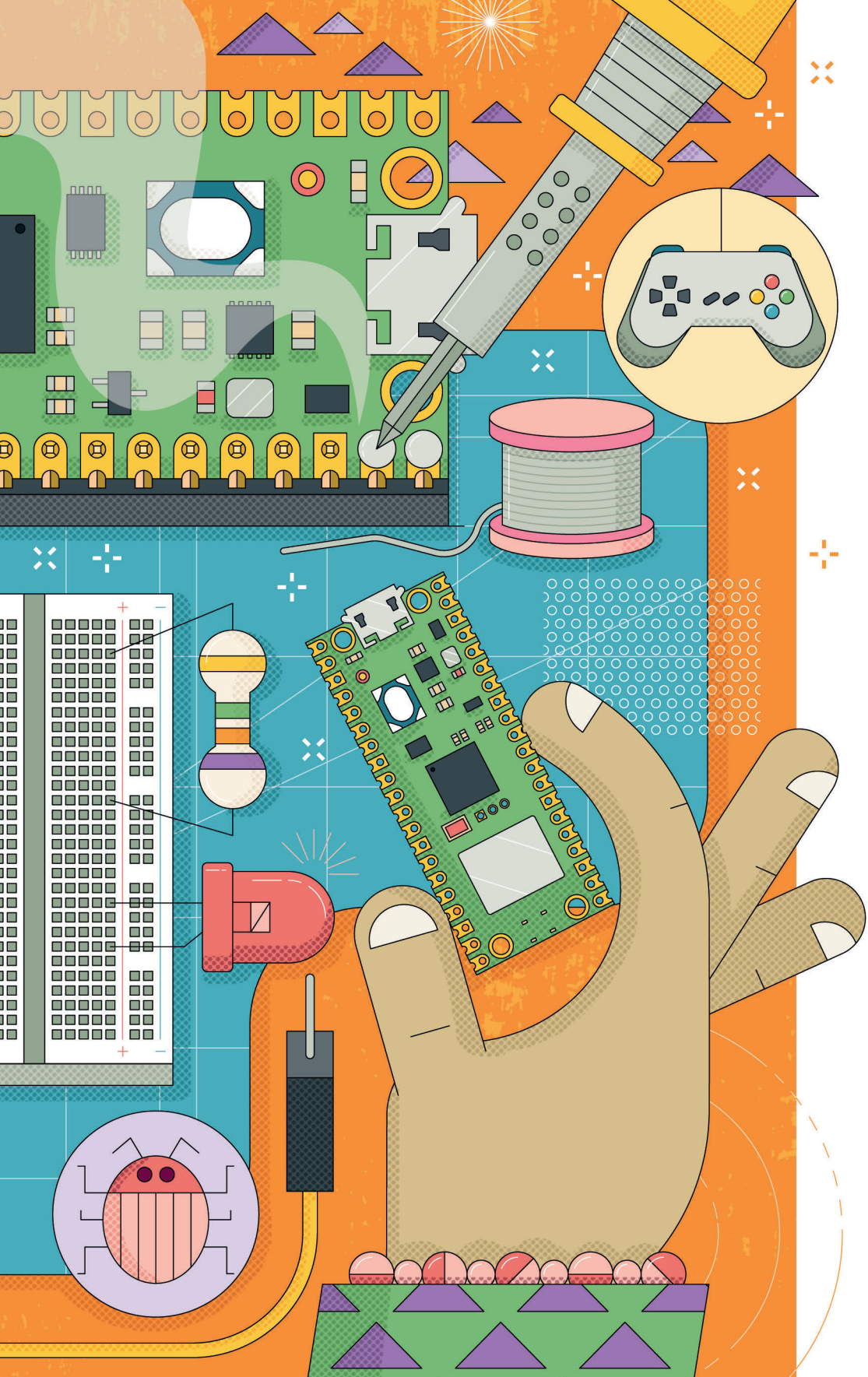
`-k`

Captures a still image when you press the **ENTER** key, rather than automatically capturing after a delay. If you want to cancel a capture, type **x** followed by **ENTER**. Works best with the timeout, `-t`, set to 0. `rpicam-vid` has a similar `-k` switch, but it works a little differently, and uses the Enter key to toggle between recording and pausing, starting in recording mode. When you've finished, type **x** followed by **ENTER** to quit.

DIGGING DEEPER



This chapter covers the most common switches for the rpicam apps, but there are plenty more. A full technical rundown of rpicam, including how it differs from the older raspivid and raspistill applications, is available at **rptl.io/camera-software.**



Chapter 9

Raspberry Pi Pico and Pico W

Raspberry Pi Pico and Pico W bring a whole new dimension to your physical computing projects.

Raspberry Pi Pico and Pico W are *microcontroller development boards*. They're designed for experimenting with physical computing using a special type of processor: a *microcontroller*. The size of a stick of gum, Raspberry Pi Pico and Pico W pack a surprising amount of power thanks to the chip at the centre of the board: an RP2040 microcontroller.

Raspberry Pi Pico and Pico W aren't designed to replace Raspberry Pi, which is an entirely different class of device known as a single-board computer. You might use Raspberry Pi to play games, write software, or browse the web, as you've seen earlier in this book. Raspberry Pi Pico is designed for physical computing projects, where it is used to control anything from LEDs and buttons to sensors, motors, and even other microcontrollers.

You can do physical computing work with your Raspberry Pi, too, thanks to its general-purpose input/output (GPIO) pins, but there are advantages to using a microcontroller development board instead of a single-board computer. Raspberry Pi Pico is smaller, cheaper, and offers some features specific to physical computing, like high-precision timers and programmable input/output systems.

This chapter isn't designed to be an exhaustive guide to what you can do with Raspberry Pi Pico and Pico W, and you don't need to buy a Pico to get the most out of your Raspberry Pi. If you already have a Raspberry Pi Pico or Pico W, or would just like to learn more about them, this chapter will serve as an introduction to their key features.

For a full view of the features of Raspberry Pi Pico and Pico W, pick up the book *Get Started with MicroPython on Raspberry Pi Pico*.

A guided tour of Raspberry Pi Pico

Raspberry Pi Pico — 'Pico' for short — is a lot smaller than even Raspberry Pi Zero, the most compact of Raspberry Pi's single-board computer family. Despite this, it includes a lot of features — all accessible using the pins around the edge of the board. It's available in two versions, Raspberry Pi Pico and Raspberry Pi Pico W; you'll see the difference between the two later.

Figure 9-1 shows your Raspberry Pi Pico as seen from above. If you look at the longer edges, you'll see gold-coloured sections with small holes. These are the pins which provide the RP2040 microcontroller with connections to the outside world — known as its input/output (I/O).

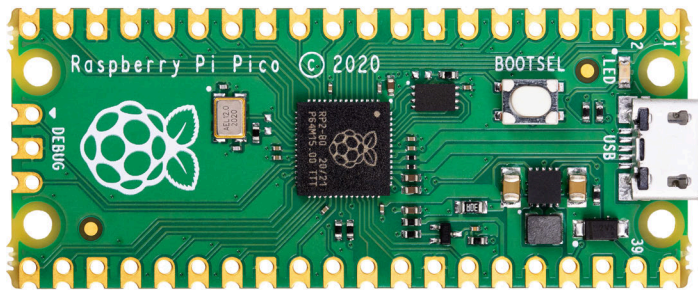


Figure 9-1 The top of the board

The pins on your Pico are very similar to the pins that make up the general-purpose input/output (GPIO) header on your Raspberry Pi — but while most Raspberry Pi single-board computers come with the physical metal pins already attached, Raspberry Pi Pico and Pico W do not.

If you want to buy a Pico with headers mounted, look for Raspberry Pi Pico H and Pico WH instead. There's a good reason to offer models without headers attached: look at the outer edge of the circuit board and you'll see it's bumpy, with little circular cut-outs (**Figure 9-2**).

These bumps create what is called a *castellated circuit board*, which can be soldered on top of other circuit boards without using any physical metal pins. It's very helpful in builds where you need to keep the height to a minimum, making for a smaller finished project. If you buy an off-the-shelf gadget powered by Raspberry Pi Pico or Pico W, it'll almost certainly be fitted using the castellations.

The holes just inwards from the bumps are to accommodate 2.54mm male pin headers. You'll recognise them as the same type of pins used on the bigger Raspberry Pi's GPIO header. By soldering these in place pointing downwards, you can push your Pico into a *solderless breadboard* to make connecting and disconnecting new hardware as easy as possible — great for experiments and rapid prototyping!

The chip at the centre of your Pico (**Figure 9-3**) is an RP2040 microcontroller. This is a *custom integrated circuit* (IC), designed and built by Raspberry Pi to operate as the brains of your Pico and other microcontroller-based devices. If you look at it closely, you'll see a Raspberry Pi logo etched into the top of the chip along with a series of letters and numbers which let engineers track when and where the chip was made.

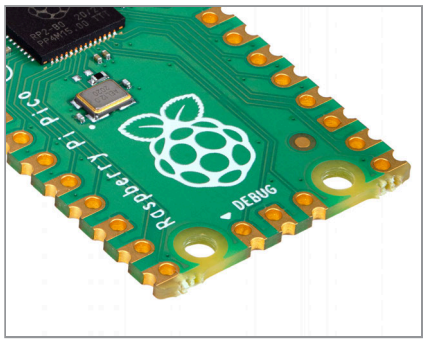


Figure 9-2
Castellation

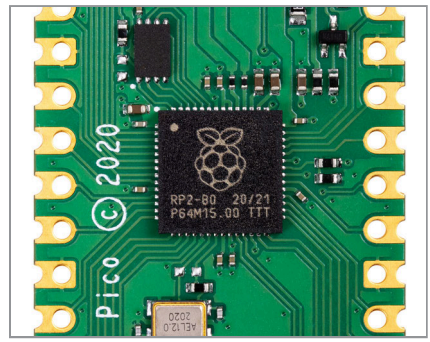


Figure 9-3
RP2040 chip

At the top of your Pico is a *micro USB port* (**Figure 9-4**). This provides power to make your Pico run, and also sends and receives data that lets your Pico talk to a Raspberry Pi or another computer via its USB port. This is how you'll load programs onto your Pico.

If you hold your Pico up and look at the micro USB port head-on, you'll see it's shaped so it's narrower at the bottom and wider at the top. Take a micro USB cable, and you'll see its connector is the same.

The micro USB cable will only go into the micro USB port on your Pico one way up. When you're connecting it, make sure to line the narrow and wide sides up the right way around — you could damage your Pico if you try to brute-force the micro USB cable in the wrong way up!

Just below the micro USB port is a small button marked 'BOOTSEL' (**Figure 9-5**). 'BOOTSEL' is short for *boot selection*, which switches your Pico between two start-up modes when it's first switched on. You'll use the boot selection button later, as you get your Pico ready for programming.

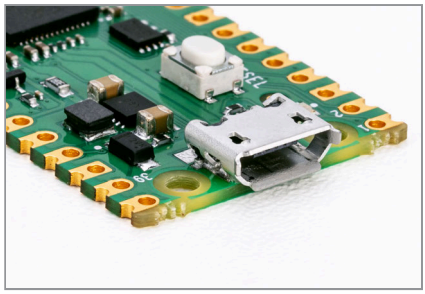


Figure 9-4
micro USB port

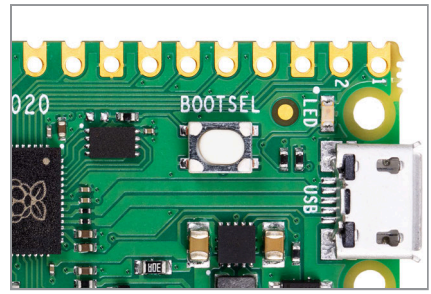


Figure 9-5
Boot selection switch

At the bottom of your Pico are three smaller gold pads with the word 'DEBUG' above them (**Figure 9-6**). These are designed for debugging, or finding errors, in programs running on the Pico, using a special tool called a *debugger*. You won't need to use the debug header at first, but you may find it useful as you write larger and more complicated programs. On some Raspberry Pi Pico models, the debug pads are replaced by a small, three-pin connector.

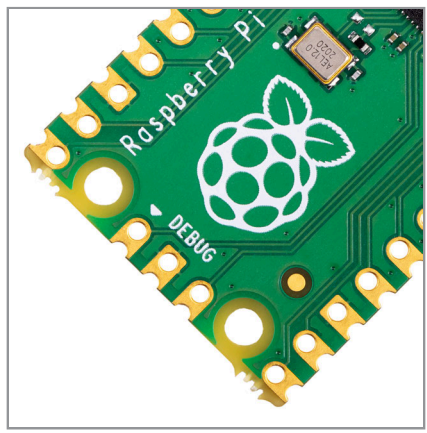


Figure 9-6
Debug pads

Turn your Pico over and you'll see the underside has writing on it (**Figure 9-7**). This printed text is known as a *silk-screen layer*, and labels each of the pins with its core function. You'll see things like 'GP0' and 'GP1', 'GND', 'RUN', and '3V3'. If you ever forget which pin is which, these labels will tell you — but you won't be able to see them when the Pico is pushed into a breadboard, so we've printed full pinout diagrams in this book for easier reference.

You might have noticed that not all the labels line up with their pins. The small holes at the top and bottom of the board are mounting holes, designed

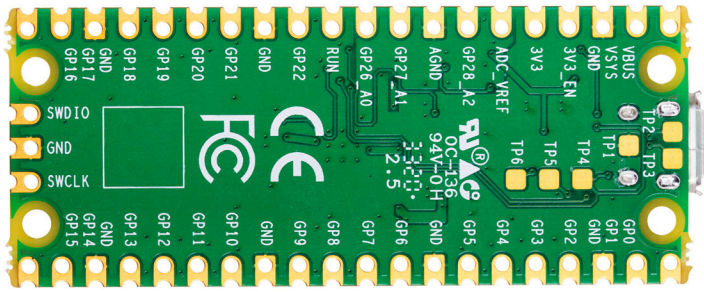


Figure 9-7 Labelled underside

to allow you to fix your Pico to projects more permanently, using screws or nuts and bolts. Where the holes get in the way of the labelling, the labels are pushed further up or down the board: looking at the top-right. So 'VBUS' is the first pin on the right, 'VSYS' the second, and 'GND' the third.

You'll also see some flat, gold pads labelled with 'TP' and a number. These are test points, and are designed for engineers to quickly check that a Raspberry Pi Pico is working after it has been assembled at the factory — you won't be using them yourself. Depending on the test pad, the engineer might use a multimeter or an oscilloscope to check that your Pico is working properly before it's packaged up and shipped to you.

If you have a Raspberry Pi Pico W or Pico WH, you'll find another piece of hardware on the board: a silver metal rectangle (**Figure 9-8**). This is a shield for a wireless module, like the one on Raspberry Pi 4 and Raspberry Pi 5, which can be used to connect your Pico to a Wi-Fi network or to Bluetooth devices. It's connected to a small antenna which sits at the very bottom of the board — which is why you'll find the debug pads or connector closer to the middle of the board on Raspberry Pi Pico W and Pico WH.

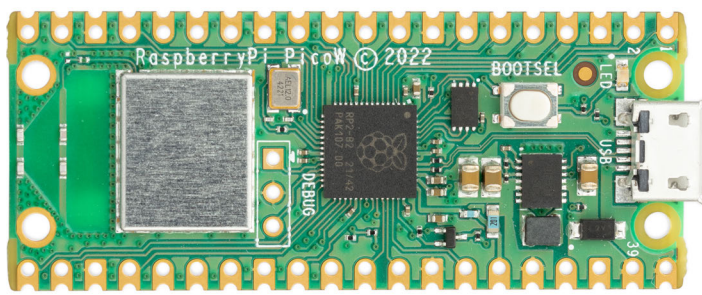


Figure 9-8 The Raspberry Pi Pico W wireless module and antenna

Header pins

When you unpack your Raspberry Pi Pico or Pico W, you'll notice that it is completely flat. There are no metal pins sticking out from the sides, like you'd find on the GPIO header of your Raspberry Pi or on the Raspberry Pi Pico H and Pico WH. You can use the castellations to attach your Pico to another circuit board, or to directly solder wires for a project where your Pico will be permanently fixed in place.

The easiest way to use your Pico, though, is to connect it to a breadboard — and for that, you'll need to attach pin headers. Putting pin headers on Raspberry Pi Pico requires a soldering iron, which heats up the pins and pads so that they can be connected using a soft metal alloy called *solder*.

For the introductory projects in this chapter, you won't need to connect any pins to your Pico. If you want to build more complicated projects, though, you can find out how to safely solder the pins into place in Chapter 1 of *Get Started with MicroPython on Raspberry Pi Pico*. You can also check to see if your favourite Raspberry Pi reseller stocks a version of Raspberry Pi Pico with the header pins already soldered on. These are known as Raspberry Pi Pico H and Raspberry Pi Pico WH for the standard and Wi-Fi versions respectively.

Installing MicroPython

Like your Raspberry Pi, you can program Raspberry Pi Pico in Python. Because it's a microcontroller rather than a single-board computer, though, it needs a special version known as *MicroPython*.

MicroPython works just like normal Python, and you can use the same Thonny IDE as when programming Raspberry Pi. There are some features of regular Python missing in MicroPython, however, and other features are added such as special libraries for microcontrollers and their peripherals.

Before you can program your Pico in MicroPython, you need to download and install the *firmware*. Start by plugging a micro USB cable into the micro USB port on your Pico — make sure it's the right way up before gently pushing it in the rest of the way.



WARNING

To install MicroPython onto your Pico, you'll need to download it from the internet. You'll only have to do this once: after MicroPython is installed, it will stay on your Pico unless you decide to replace it with something else in the future.

Hold down the '**BOOTSEL**' button on the top of your Pico. Then, while still holding it down, connect the other end of the micro USB cable to one of the USB ports on your computer. Count to three, then let go of the button.

> **NOTE**
>
> On macOS, you may be asked whether you want to "**Allow accessory to connect**" when you plug the Pico into your computer. You will need to click **Allow** to permit it. After you install MicroPython onto your Pico, macOS may ask the question a second time, because it now looks like a different device.

After a few more seconds you should see your Pico appear as a removable drive, as though you'd connected a USB flash drive or external hard drive. On a Raspberry Pi, you'll see a pop-up asking if you'd like to open the drive in the File Manager. Make sure **Open in File Manager** is selected and click **OK**.

In the File Manager window, you'll see two files on your Pico (**Figure 9-9**): **INDEX.HTM** and **INFO_UF2.TXT**. The second file contains information about your Pico, such as the version of the bootloader it's currently running. The first file, **INDEX.HTM**, is a link to the Raspberry Pi Pico website. Double-click on this file or open your web browser and type **rptl.io/microcontroller-docs** into the address bar.

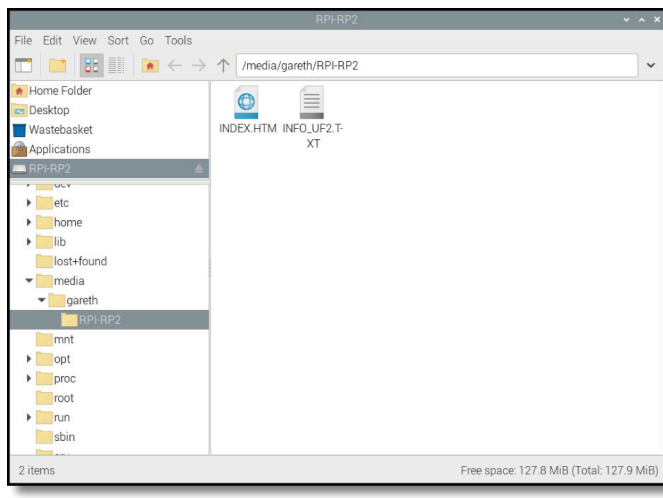


Figure 9-9 You'll see two files on your Raspberry Pi Pico

When the web page opens, you'll see information about Raspberry Pi's microcontroller and development boards, including Raspberry Pi Pico and Pico W. Click on the MicroPython box to go to the firmware download page. Scroll down to the section labelled **Drag-and-Drop MicroPython**, as shown in **Figure 9-10**,

and find the link for the version of MicroPython for your board. There's one for Raspberry Pi Pico and Pico H, and another for Raspberry Pi Pico W and Pico WH. Click on the link to download the appropriate UF2 file. If you accidentally download the wrong file, don't worry; you can come back to the page at any time and flash new firmware onto your device using the same process.

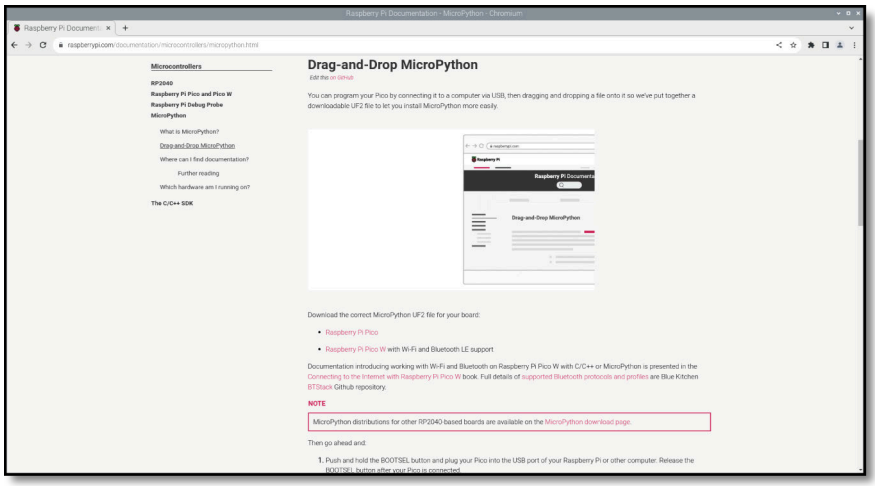


Figure 9-10 Click the link to download the MicroPython firmware

Open a new File Manager window, then navigate to your **Downloads** folder and find the file you just downloaded. It will be called '**rp2-pico**' or '**rp2-pico-w**' followed by a date, some text and numbers which are used to tell different firmware builds apart, along with the extension '**uf2**'.



NOTE

To find the Downloads folder on your Raspberry Pi, click the Raspberry Pi menu, choose **Accessories**, and open the **File Manager**. Next, look for **Downloads** in the list of folders to the left of the **File Manager** window. You may have to scroll down the list to find it, depending on how many folders you have on your Raspberry Pi.

Click and hold the mouse button on the UF2 file, then drag it to the other window that's open on your Pico's removable storage drive. Hover it over that window and let go of the mouse button to drop the file onto your Pico, as shown in **Figure 9-11**.

After a few seconds you'll see your Pico drive window disappear from **File Manager**, **Explorer**, or **Finder**, and you may also see a warning that a drive was removed without being ejected. Don't worry, that's supposed to happen! When you dragged the MicroPython firmware file onto your Pico, you told it

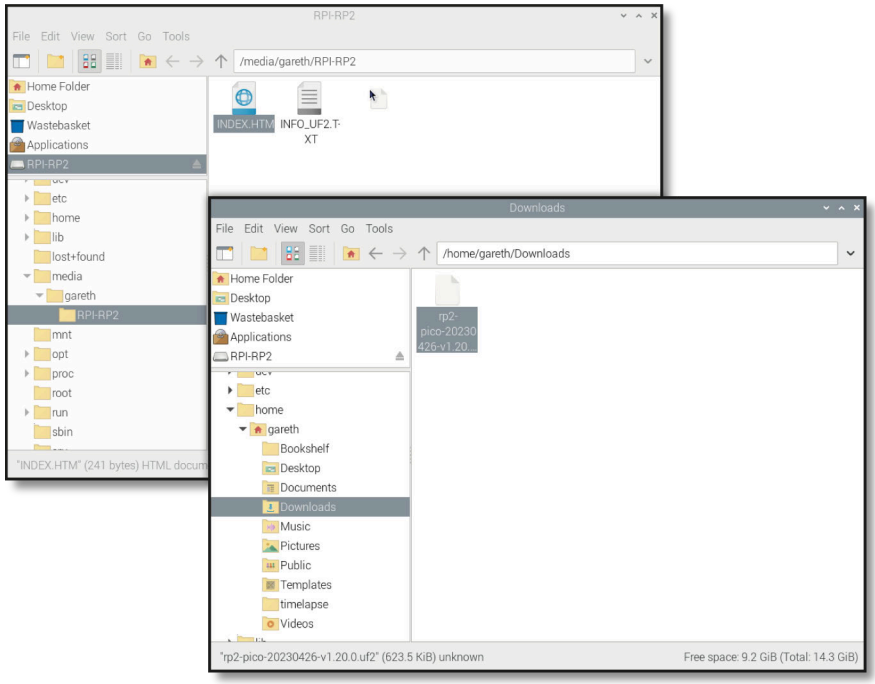


Figure 9-11 Drag the MicroPython firmware file to your Raspberry Pi Pico

to flash the firmware onto its internal storage. To do that, your Pico switches out of the special mode you put it in with the 'BOOTSEL' button, flashes the new firmware, and then loads it — meaning that your Pico is now running MicroPython.

Congratulations: you're now ready to get started with MicroPython on your Raspberry Pi Pico!

FURTHER READING

The webpage linked from **INDEX.HTM** isn't just a place to download MicroPython. It also hosts plenty of additional resources. Click on the tabs and scroll to access guides, projects, and the *databook* collection — a bookshelf of detailed technical documentation covering everything from the inner workings of the RP2040 microcontroller which powers your Pico, to programming it in both the Python and C/C++ languages.

Your Pico's pins

Your Pico talks to hardware through the series of pins along both its edges. Most work as programmable input/output (PIO) pins, meaning they can be programmed to act as either an input or an output, and have no preset purpose of their own until you assign one. Some pins have extra features and alternative modes for communicating with more complicated hardware; others have a specific purpose, providing connections for things like power.

Raspberry Pi Pico's 40 pins are labelled on the underside of the board, with three also labelled with their numbers on the top of the board: Pin 1, Pin 2, and Pin 39. These top labels help you remember how the numbering works: Pin 1 is at the top-left as you look at the board from above, with the micro USB port to the upper side. Pin 20 is the bottom-left, Pin 21 the bottom-right, and Pin 39 one below the top-right with the unlabelled Pin 40 above it. The labelling on the underside is more thorough, but you won't be able to see it when your Pico is plugged into a breadboard!

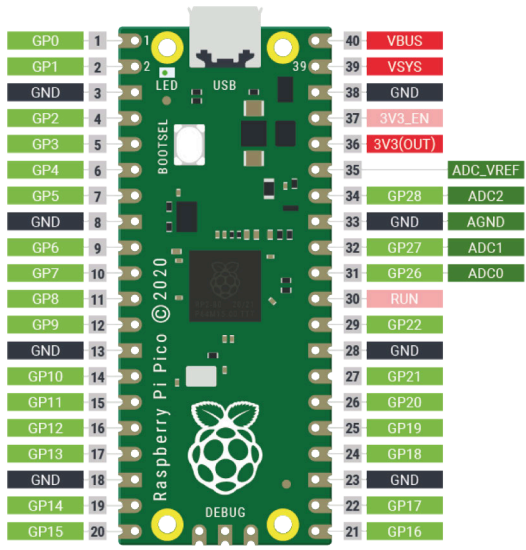


Figure 9-12 The Raspberry Pi Pico's pins, seen from the top of the board

On the Raspberry Pi Pico, pins are usually referred to by their functions (see **Figure 9-12**) rather than by number. There are several categories of pin types, each of which has a particular function:

- **3V3** — *3.3 volts power* — A source of 3.3V power generated from the VSYS input. This power supply can be switched on and off using the 3V3_EN pin above it, which also switches your Pico off.

- **VSYS** — *~2-5 volts power* — A pin directly connected to your Pico's internal power supply, which cannot be switched off without also switching the Pico off.
- **VBUS** — *5 volts power* — A source of 5V power taken from your Pico's micro USB port, and used to power hardware which needs more than 3.3V.
- **GND** — *0 volts ground* — A ground connection, used to complete a circuit connected to a power source. Several GND pins are dotted around your Pico to make wiring easier.
- **GPxx** — *General-purpose input/output pin number 'xx'* — The GPIO pins available for your program, labelled GP0 through to GP28.
- **GPxx_ADCx** — *General-purpose input/output pin number 'xx', with analogue input number 'x'* — A GPIO pin which ends in ADC and a number can be used as an analogue input as well as a digital input or output — but not both at the same time.
- **ADC_VREF** — *Analogue-to-digital converter (ADC) voltage reference* — A special input pin which sets a reference voltage for any analogue inputs.
- **AGND** — *Analogue-to-digital converter (ADC) 0 volts ground* — A special ground connection for use with the ADC_VREF pin.
- **RUN** — *Enables or disables your Pico* — The RUN header is used to start and stop your Pico from another microcontroller or other controlling device.

Connecting Thonny to Pico

Begin by loading Thonny: click the Raspberry Pi menu at the top-left on your screen, move the mouse to the **Programming** section, and click on **Thonny**.

With your Pico connected to your Raspberry Pi, click on the words **Local Python 3** at the bottom-right of the Thonny window. This shows your current interpreter, which is responsible for taking the instructions you type and turning them into code that the computer, or microcontroller, can understand and run. Normally the interpreter is the copy of Python running on your Raspberry Pi, but it needs to be changed to run your MicroPython programs on your Pico.

Look for 'MicroPython (Raspberry Pi Pico)' (**Figure 9-13**) in the list that appears, and click on it. If you can't see it in the list, double-check that your Pico is properly plugged into the micro USB cable, and that the micro USB cable is properly plugged into your Raspberry Pi or other computer.

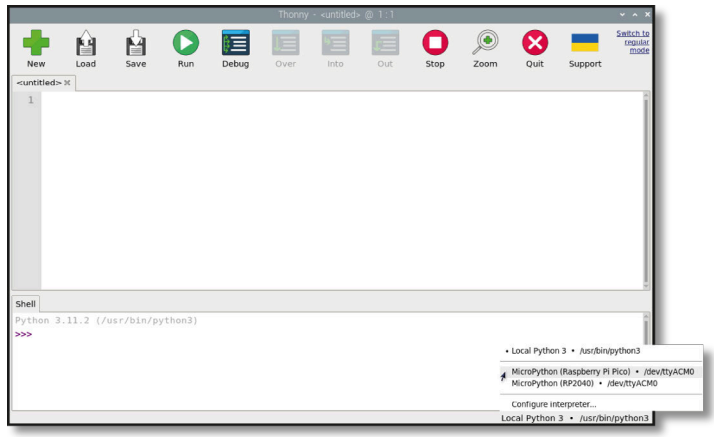


Figure 9-13 Choosing a Python interpreter



PYTHON PROFESSIONALS

This chapter assumes you're familiar with the Thonny IDE and writing simple Python programs. If you haven't done so already, work through the projects in Chapter 5, *Programming with Python* before continuing with this chapter.



INTERPRETER SWITCHING

Choosing the interpreter picks where and how your program will run: when you choose **MicroPython (Raspberry Pi Pico)**, programs will run on your Pico; picking **Local Python 3** means programs will run on your Raspberry Pi instead.

If you find programs aren't running where you'd expect, make sure to check which interpreter Thonny is set to use!

Your first MicroPython program: Hello, World!

You can check that everything's working the same way you learned to write Python programs on Raspberry Pi: by writing a simple 'Hello, World!' program. First, click on the Python shell area at the bottom of the Thonny window, just to the right of the bottom `>>>` symbols, and type the following instruction before pressing the **ENTER** key:

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

When you press **ENTER**, you'll see that your program begins to run instantly: Python will respond, in the same shell area, with the message '`Hello, World!`' (**Figure 9-14**), just as you asked. That's because the shell is a direct line to the MicroPython interpreter running on your Pico, whose job it is to look at your instructions and interpret what they mean. This interactive mode works the same as when you're programming your Raspberry Pi: instructions written in the shell area are acted on immediately, with no delay. The only difference: they're sent to your Pico to run them, and any result — in this case the message 'Hello, World!' — is sent back to Raspberry Pi to be displayed.

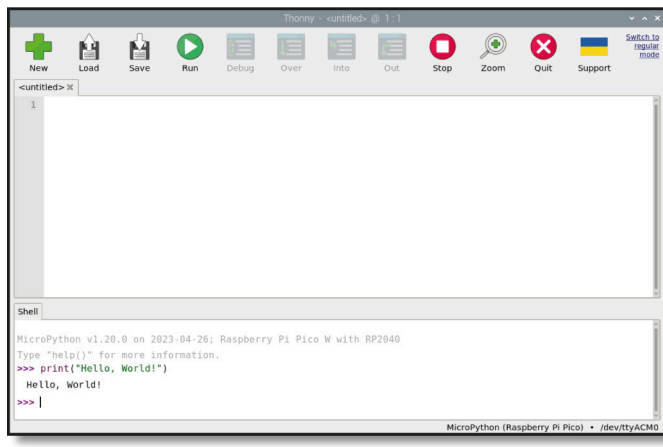


Figure 9-14 MicroPython prints the 'Hello, World!' message in the shell area

You don't have to program your Pico (or your Raspberry Pi) in interactive mode. Click on the script area in the middle of the Thonny window, then type your program again:

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

When you press the **ENTER** key this time, nothing happens — except that you get a new, blank line in the script area. To make this version of your program work, you'll have to click the **Run** icon in the Thonny toolbar.

Even though this is a simple program, you'll want to get in the habit of saving your work. Before you run your program, click the **Save** icon. You'll be asked whether you want to save your program to '**This computer**', meaning your Raspberry Pi or whatever other computer you're running Thonny on, or to '**Raspberry Pi Pico**' (**Figure 9-15**). Click **Raspberry Pi Pico**, then type a descriptive name like **Hello World.py** and click the OK button.

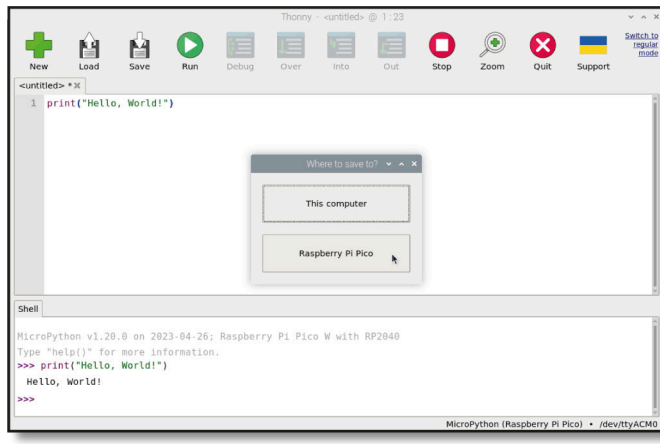


Figure 9-15 Saving a program to Pico

Click the **Run** icon now. It will run automatically on your Pico. You'll see two messages appear in the shell area at the bottom of the Thonny window:

```
>>> %Run -c $EDITOR_CONTENT
Hello, World!
```

The first of these lines is an instruction from Thonny telling the MicroPython interpreter on your Pico to run the code that's in the script area (the EDITOR_CONTENT). The second is the output of the program — the message you told MicroPython to print. Congratulations: now you've written two MicroPython programs, in interactive and script modes, and you've successfully run them on your Pico!

There's just one more piece to the puzzle: loading your program again. Close Thonny by pressing the X at the top-right of the window on Windows or Linux (use the close button at the top-left of the window on macOS), then launch Thonny again. This time, instead of writing a new program, click the **Load** icon in the Thonny toolbar. You'll be asked whether you want to save it to '**This computer**' or your '**Raspberry Pi Pico**' again. Click **Raspberry Pi Pico** and you'll see a list of all the programs you've saved to your Pico.

A PICO FULL OF PROGRAMS

When you tell Thonny to save your program on the Pico, it means that the programs are stored on the Pico itself. If you unplug your Pico and plug it into a different computer, your programs will still be where you saved them: on your very own Pico.

Find **Hello_World.py** in the list — if your Pico is new, it will be the only file there. Click to select it, then click OK. Your program will load into Thonny, ready to be edited, or for you to run it again.

CHALLENGE: NEW MESSAGE

Can you change the message the Python program prints as its output? If you wanted to add more messages, would you use interactive mode or script mode? What happens if you remove the brackets or the quotation marks from the program and then try to run it again?

Your first physical computing program: Hello, LED!

Just as printing 'Hello, World' to the screen is the usual first step in learning a programming language, making an LED light up is the traditional introduction to learning physical computing on a new platform. You can get started without any additional components, too: your Raspberry Pi Pico has a small LED, known as a *surface-mount device (SMD) LED*, on top.

Start by finding the LED: it's the small rectangular component to the left of the micro USB port at the top of the board (**Figure 9-16**), marked 'LED'.

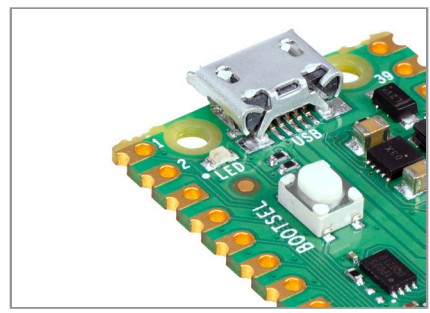


Figure 9-16
The on-board LED is found to the left of the micro USB connector

The on-board LED is connected to one of RP2040's general-purpose input/output pins, GP25. This is one of the 'missing' GPIO pins provided by the RP2040 microcontroller, but not broken out to a physical pin on the edge of your Pico. While you can't connect any hardware to the pin (other than the on-board LED), it can be treated just the same as any other GPIO pin within

your programs. It's a simple way to add an output to your programs without needing any extra components.

Click the **New** icon ✚ in Thonny and start your program with the following line:

```
import machine
```

This short line of code is key to working with MicroPython on your Pico. It loads, or *imports*, a collection of MicroPython code known as a *library* — in this case, the `machine` library. The `machine` library contains all the instructions MicroPython needs to communicate with the Pico and other MicroPython-compatible devices, extending the language for physical computing. Without this line, you won't be able to control any of your Pico's GPIO pins — and you won't be able to make the on-board LED light up.

The `machine` library exposes what is known as an *application programming interface (API)*. The name sounds complicated, but describes exactly what it does: it provides a way for your program, or the *application*, to communicate with the Pico via an *interface*.

The next line of your program provides an example of the `machine` library's API:

```
led_onboard = machine.Pin("LED", machine.Pin.OUT)
```

This line defines an object called `led_onboard`, which offers a friendly name you can use to refer to the on-board LED later in your program. It's technically possible to use any name here, but it's best to stick with names which describe the variable's purpose, to make the program easier to read and understand.

The second part of the line calls the `Pin` function in the machine library. This function, as its name suggests, is designed for handling your Pico's GPIO pins. At the moment, none of the GPIO pins — including GP25, the pin connected to the on-board LED — know what they're supposed to be doing. The first argument, `LED`, is a special *macro* which is assigned to the on-board LED, which you can use instead of having to remember number of its pin. The second, `machine.Pin.OUT`, tells Pico the pin should be used as an *output* rather than an *input*.

That line alone is enough to set the pin up, but it won't light the LED. To do that, you need to tell your Pico to actually turn the pin on. Type the following code on the next line:

```
led_onboard.value(1)
```

This line is also using the machine library's API. Your earlier line created the object `led_onboard` as an output on pin GP25, using the `LED` macro; this line takes the object and sets its *value* to 1 for 'on'. It could also set the value to 0, for 'off'.

PIN NUMBERS

The GPIO pins on your Pico are usually referred to using their full names: GP25 for the pin connected to the on-board LED, for example. In MicroPython, though, the letters G and P are dropped — so, if you're using the pin number rather than the `LED` macro, make sure you write '25' rather than 'GP25' in your program or it won't work!

Click the **Run** button and save the program on your Pico as **Blink.py**. You'll see the LED light up. Congratulations: you've written your first physical computing program!

You'll notice, however, that the LED stays lit. That's because your program tells the Pico to turn it on, but never tells it to turn it off. You can add another line at the bottom of your program:

```
led_onboard.value(0)
```

Run the program this time, though, and the LED never seems to light up. That's because your Pico works very, very quickly — much faster than you can see with the naked eye. The LED is lighting up, but for such a short time that it appears to remain dark. To fix that, you need to slow your program down by introducing a delay.

Go back to the top of your program: click to move your cursor to the end of the first line and press **ENTER** to insert a new second line. On this line, type:

```
import time
```

Like `import machine`, this line imports a new library into MicroPython: the `time` library. This library handles everything to do with time, from measuring it to inserting delays into your programs.

Click on the end of the line `led_onboard.value(1)`, then press **ENTER** to insert a new line. Type:

```
time.sleep(5)
```

This calls the `sleep` function from the `time` library, which makes your program pause for the number of seconds you typed: in this case, five seconds.

Click the **Run** button again. This time you'll see the on-board LED on your Pico light up, stay lit for five seconds — try counting along — and go out again.

Finally, it's time to make the LED blink. To do that, you'll need to create a loop. Rewrite your program so it matches the one below:

```
import machine
import time

led_onboard = machine.Pin(LED, machine.Pin.OUT)

while True:
    led_onboard.value(1)
    time.sleep(5)
    led_onboard.value(0)
    time.sleep(5)
```

Remember that the lines inside the loop need to be indented by four spaces, so MicroPython knows they form the loop. Click the **Run** icon again, and you'll see the LED switch on for five seconds, switch off for five seconds, and switch on again, constantly repeating in an infinite loop. The LED will continue to flash until you click the **Stop** icon to cancel your program and reset your Pico.

There's another way to handle the same job, too: using a *toggle*, rather than setting the LED's output to 0 or 1 explicitly. Delete the last four lines of your program and replace them so it looks like this:

```
import machine
import time

led_onboard = machine.Pin(LED, machine.Pin.OUT)

while True:
    led_onboard.toggle()
    time.sleep(5)
```

Run your program again. You'll see the same activity as before: the on-board LED will light up for five seconds, then go out for five seconds, then light up again in an infinite loop. This time, though, your program is two lines shorter: you've *optimised* it. Available on all digital output pins, `toggle()` simply switches between on and off: if the pin is currently on, `toggle()` switches it off; if it's off, `toggle()` switches it on.

CHALLENGE: LONGER LIGHT-UP

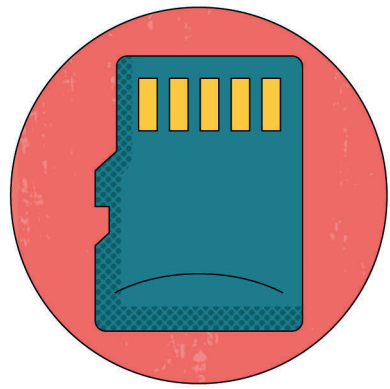


How would you change your program to make the LED stay on for longer? What about staying off for longer? What's the smallest delay you can use while still being able to see the LED blink on and off?

Congratulations: you've learned what a microcontroller is, how to connect Raspberry Pi Pico to your Raspberry Pi, how to write MicroPython programs, and how to toggle an LED by controlling a pin on the Pico!

There's much more to learn about your Raspberry Pi Pico: using it with a breadboard, connecting additional hardware like LEDs, buttons, motion sensors or screens, and even making use of advanced features like its *analogue to digital converters (ADCs)* and *programmable input/output (PIO)* capabilities. And that's without considering connecting it to your network to begin experimenting with the *Internet of Things (IoT)*.

To learn more, pick up a copy of *Get Started with MicroPython on Raspberry Pi Pico*. It's available at all good booksellers, online and in print.



Appendix A

Install an operating system to a microSD card

You can buy microSD cards with Raspberry Pi OS pre-installed on them from all good Raspberry Pi retailers, so you can get started with Raspberry Pi quickly and easily. Pre-loaded microSD cards are also bundled with the Raspberry Pi Desktop Kit and Raspberry Pi 400.

If you'd prefer to install the operating system yourself on a blank microSD card, you can do this easily using Raspberry Pi Imager. If you're using Raspberry Pi 4, Raspberry Pi 400, or Raspberry Pi 5, you can also download and install the operating system over the network directly on your device.

WARNING!

If you've purchased a microSD card with Raspberry Pi OS already pre-installed, you don't need to do anything else other than plug it into your Raspberry Pi. These instructions are for installing Raspberry Pi OS on blank microSD cards, or on cards you've used before and want to repurpose. Carrying out these instructions on a microSD card with files on it will delete those files, so make sure you've backed things up first!

Downloading Raspberry Pi Imager

Based on Debian, Raspberry Pi OS is the official operating system for Raspberry Pi. The easiest way to install Raspberry Pi OS on a microSD card for your Raspberry Pi is to use the Raspberry Pi Imager tool, downloadable from **rptl.io/imager**.

The Raspberry Pi Imager application is available for Windows, macOS, and Ubuntu Linux computers, so choose the relevant version for your system. If the only computer you have access to is your Raspberry Pi, skip to "Running Raspberry Pi Imager over the network" to see if it's possible to run the tool directly on your Raspberry Pi. If not, you'll need to purchase a microSD card with the operating system already installed from a Raspberry Pi reseller — or ask a friend if they can install it onto your microSD card for you.

On macOS, double-click the downloaded **DMG** file. You may need to change your Privacy & Security setting to allow apps downloaded from the App Store and identified developers to enable it to run. You can then drag the **Raspberry Pi Imager** icon into the Applications folder.

On a Windows PC, double-click the downloaded **EXE** file. When prompted, select the **Yes** button to enable it to run. Then click the **Install** button to start the installation.

On Ubuntu Linux, double-click the downloaded **DEB** file to open the Software Centre with the package selected, then follow the on-screen instructions to install Raspberry Pi Imager.

You can now attach your microSD card to your computer. You'll need a USB adapter unless your computer has a card reader built in — many laptops do, but not many desktops. Note that the microSD card doesn't need to be pre-formatted.

Launch the Raspberry Pi Imager application, then skip to "Writing the OS to the microSD card" on page 241.

Running Raspberry Pi Imager over the network

Raspberry Pi 4 and Raspberry Pi 400 all include the ability to run Raspberry Pi Imager themselves, loading it over the network without the need to use a separate desktop or laptop computer.



WARNING!

At the time of this writing, network installation is not supported on Raspberry Pi 5, but it will be available in a future firmware update.

To run Raspberry Pi Imager directly, you'll need your Raspberry Pi, a blank microSD card, a keyboard (if you're not using the Raspberry Pi 400's built-in keyboard), a TV or monitor, and an Ethernet cable connected to your modem or router. Note that installation over a Wi-Fi connection isn't supported.

Insert your blank microSD card into your Raspberry Pi's microSD slot, and connect the keyboard, Ethernet cable, and USB power supply. If you're reusing an old microSD card, hold down the **Shift** key on the keyboard as Raspberry Pi boots up to load the network installer; if your microSD is blank, the installer will load automatically.

When you see the network installer screen, hold down the **Shift** key to begin the installation process. The installer will automatically download a special version of Raspberry Pi Imager and load it on your Raspberry Pi as shown in **Figure A-1**. Once it's downloaded, you'll see a screen exactly like the stand-alone version of Raspberry Pi Imager, complete with options to choose an operating system and a storage device for installation.

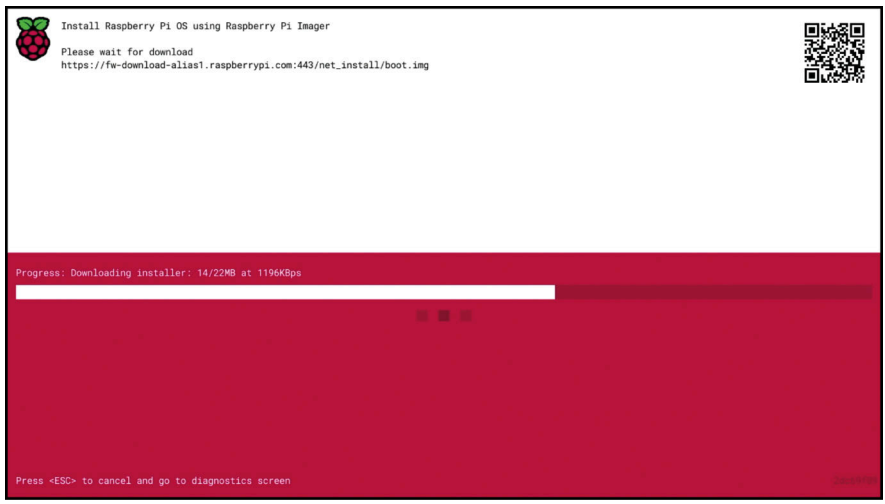


Figure A-1 Installing Raspberry Pi OS over the network

Writing the OS to the microSD card

Click the **Choose Device** button to select which model of Raspberry Pi you have, and you'll see the screen shown in **Figure A-2**. Find your Raspberry Pi in the list and click it. Next, Click **Choose OS** to select which operating system you would like to install, and the screen shown in **Figure A-3** will appear.

The top option is standard Raspberry Pi OS with desktop — if you'd prefer the slimmed-down Lite version, or the Full version ('Raspberry Pi OS with desktop and recommended software'), select **Raspberry Pi OS (other)**.

You can also scroll down the list to see a range of third-party operating systems compatible with Raspberry Pi. Depending on your model of Raspberry Pi,

these may range from general-purpose operating systems like Ubuntu Linux and RISC OS Pi to operating systems tailored for home entertainment, gaming, emulation, 3D printing, digital signage, and more.

Near the very bottom of the list, you'll find **Erase**; this will wipe the microSD card and all the data on it.

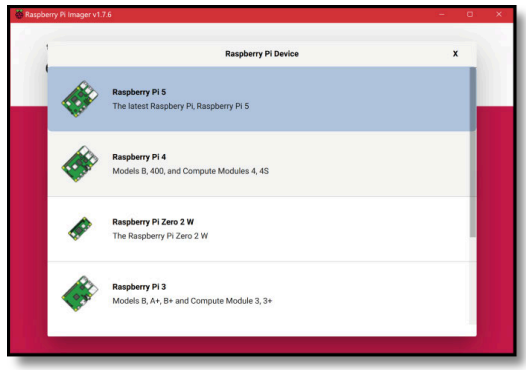


Figure A-2 Choosing your model of Raspberry Pi

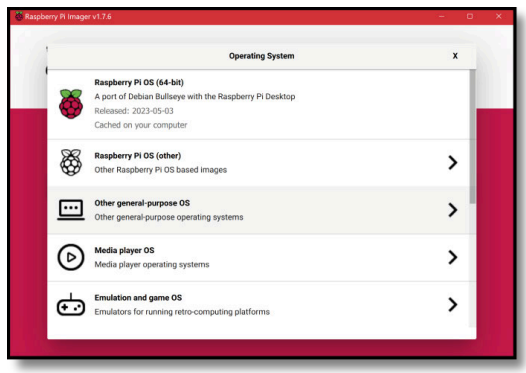


Figure A-3 Choosing an operating system



32-BIT VERSUS 64-BIT

After you select a model of Raspberry Pi, you will only be offered images that are compatible with your model. If Raspberry Pi OS (64-bit) is among the options, as would be the case with Raspberry Pi 4 or Raspberry Pi 5, choose the 64-bit option unless you have a compelling need to install a 32-bit version of the operating system.

If there's an operating system you want to try which isn't in the list, you can still install it using Raspberry Pi Imager. Just head to the operating system's

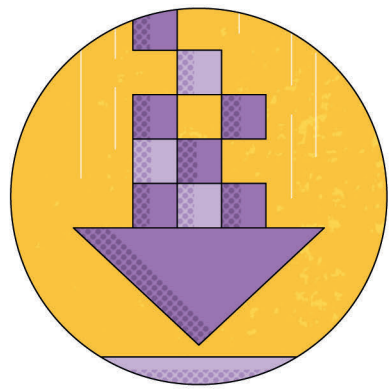
website, download the image, then pick the **Use custom** option from the bottom of the **Choose OS** list.

With an OS selected, click the **Choose Storage** button and select your microSD card. Usually, it'll be the only storage device in the list. If you see more than one storage device — this usually happens if you have another microSD card or a USB flash drive attached to your computer — then be very careful to pick the right device, or you could wipe your drive and lose all your data. If in doubt, just close Raspberry Pi Imager, disconnect all removable drives except your target microSD card, and open Raspberry Pi Imager again.

Finally, click the **Next** button and you'll be asked whether you want to customise the operating system. If you are running the Lite version, you will need to go through this step because it allows you to configure your username, password, wireless network connection, and more without needing to connect a keyboard, mouse, and monitor.

Next, Raspberry Pi Imager will ask you to confirm whether you should write over the contents of your SD card, and if you click **Yes**, it will begin. Wait while the utility writes the selected OS to your card and then verifies it. When the operating system has been written to the card, you can remove the microSD card from your desktop or laptop computer and insert it into your Raspberry Pi to boot into your new operating system. If you wrote the new OS on your Raspberry Pi itself using the network boot feature, simply turn Raspberry Pi off and back on again to load your new OS.

Always make sure the write process has finished before removing the microSD card or switching your Raspberry Pi off. If the process is interrupted part-way through, your new OS won't work properly. If that happens, simply start the write process again to overwrite the damaged OS and replace it with a working copy.



Appendix B

Installing and uninstalling software

Raspberry Pi OS comes with a selection of popular software packages, hand-picked by the team at Raspberry Pi, but these are not the only packages that will work on a Raspberry Pi. Using the following instructions, you can browse additional software, install it, and uninstall it again — expanding the capabilities of your Raspberry Pi.

The instructions in this appendix are supplementary to those in Chapter 3, *Using your Raspberry Pi*, which explains how to use the **Recommended Software** tool.

Browsing available software

To see and search the list of software packages available for Raspberry Pi OS using its *software repositories*, click the Raspberry Pi icon to load the menu, select the Preferences category, then click on **Add/Remove Software**. After a few seconds, the tool's window will appear as shown in **Figure B-1**.

The left-hand side of the **Add/Remove Software** window contains a list of categories. They're the same categories that you will see in the main menu when you click on the Raspberry Pi icon.

Clicking on one category will show you a list of the software available for it. You can also enter a search term in the box at the top-left of the window, such as 'text editor' or 'game', and see a list of matching software packages, which can come from any category. Clicking on any package brings up additional

information about it in the space at the bottom of the window as shown in **Figure B-2.**

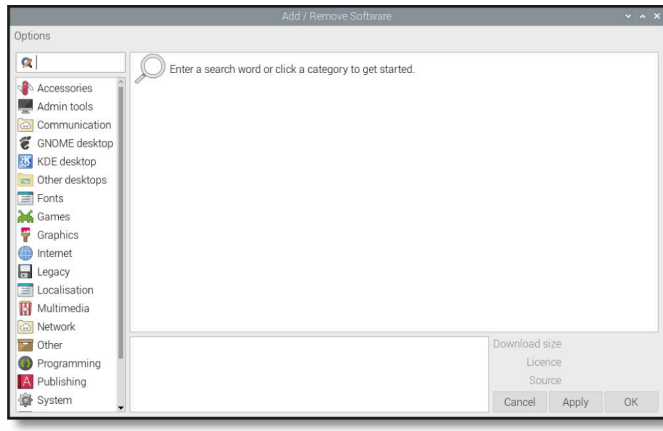


Figure B-1 The **Add/Remove Software** window

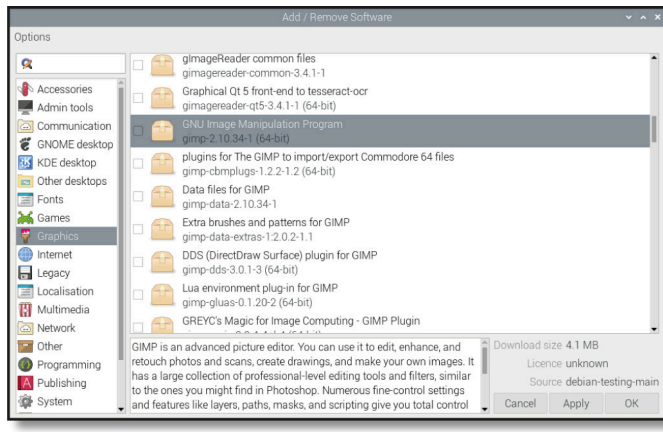


Figure B-2 Additional package information

If the category you've chosen has lots of software packages available, it may take some time for the **Add/Remove Software** tool to finish working through the list.

Installing software

To select a package for installation, check the box next to it by clicking on it. You can install more than one package at once: keep clicking to add more packages. The icon next to the package will change to an open box with a '+' symbol, as shown in **Figure B-3**, to confirm that it's going to be installed.

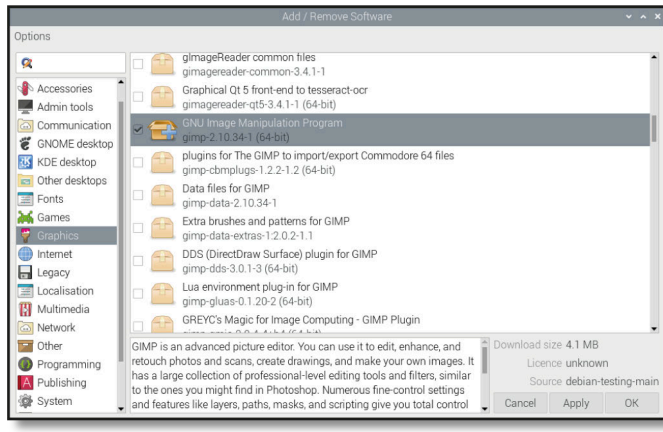


Figure B-3 Selecting a package for installation

When you're happy with your choices, click either the **OK** or **Apply** button. the only difference is that **OK** will close the **Add/Remove Software** tool when your software is installed, while the **Apply** button leaves it open. You'll be asked to enter your password (**Figure B-4**), to confirm your identity — you wouldn't want anyone else to be able to add or remove software from your Raspberry Pi, after all!

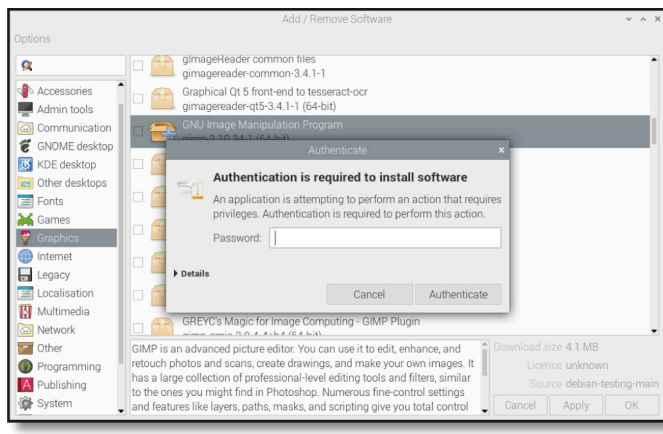


Figure B-4 Authenticating your identity

You may find that when you install a single package, other packages are installed alongside it. These are *dependencies*: packages that the software you chose to install needs in order to work, like sound effect bundles for a game, or a database to go with a web server.

Once the software is installed, you should be able to find it by clicking the Raspberry Pi icon to load the menu and then choosing the software package's category (see **Figure B-5**). Keep in mind that the menu category isn't always the same as the category in the **Add/Remove Software** tool, and some software doesn't have an entry in the menu at all. This software is known as *command-line software*, and needs to be run at the terminal. For more information on the command line and the terminal, turn to Appendix C, *The command-line interface*.

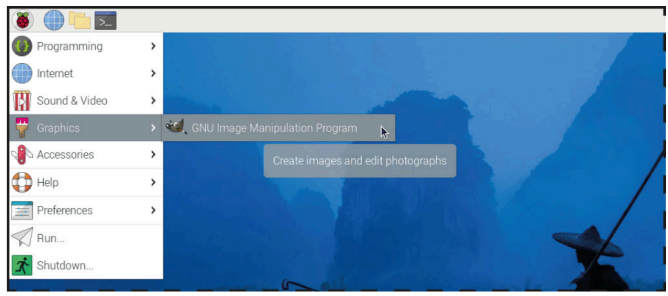


Figure B-5 Finding the software you just installed

Uninstalling software

To select a package for removal, or *uninstallation*, find it in the list of packages — the search function is handy here — and uncheck the box next to it by clicking on it. You can uninstall more than one package at once: just keep clicking to remove more packages. The icon next to the package will change to an open box next to a small recycle bin, to confirm that it's going to be uninstalled (see **Figure B-6**).

As before, you can click **OK** or **Apply** to begin uninstalling the selected software packages. You'll be asked to confirm your password, unless you did so within the last few minutes, and you may also be prompted to confirm that you want to remove any dependencies relating to your software package as well (see **Figure B-7**). When the uninstallation has finished, the software will disappear from the Raspberry Pi menu, but files you created using the software — pictures for a graphics package, for example, or saves for a game — won't be removed.

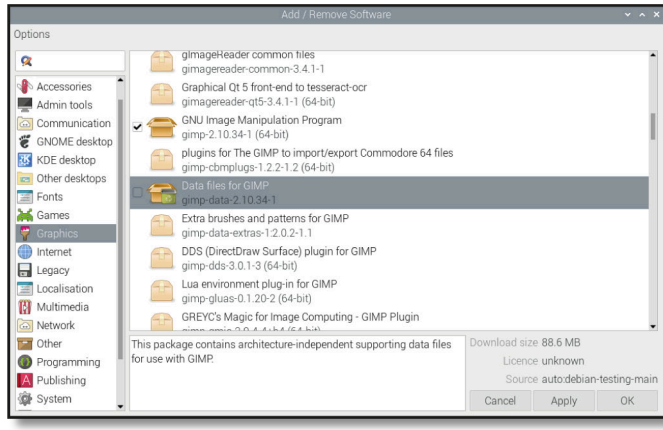


Figure B-6 Selecting a package for removal

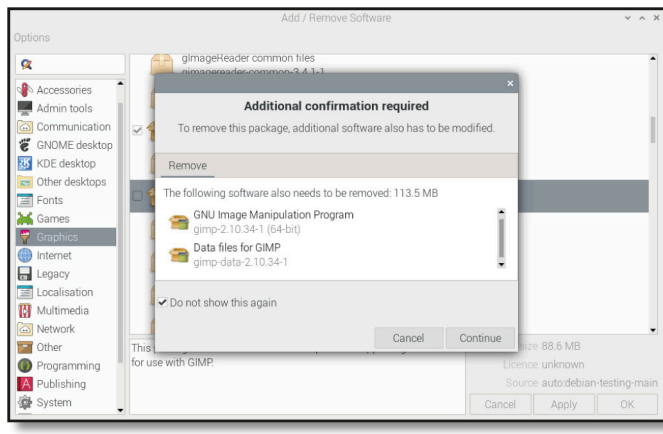
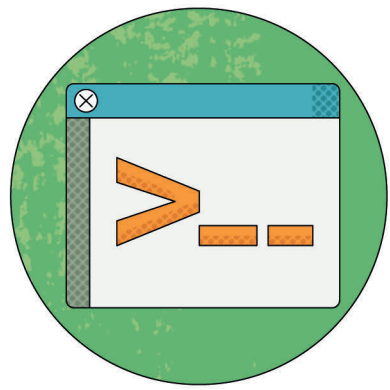


Figure B-7 Confirming whether to remove dependencies

WARNING!

All software installed in Raspberry Pi OS appears in **Add/Remove Software**, including software required for your Raspberry Pi to run. It's possible to remove packages that the desktop needs to load. To avoid this, don't uninstall things unless you're sure you no longer need them. You can reinstall Raspberry Pi OS by following the instructions in Appendix A, *Install an operating system to a microSD card*.



Appendix C

The command-line interface

While you can manage most of the software on a Raspberry Pi through the desktop, some can only be accessed using a text-based mode known as the *command-line interface (CLI)* in an application called Terminal. Most users will never need to use the CLI, but for those who want to learn more, this appendix offers a basic introduction.

Loading the Terminal

The CLI is accessed through the Terminal, a software package which loads what is technically known as a *virtual teletype (VTY) terminal*, a name dating back to the early days of computers when users issued commands via a large electromechanical typewriter rather than a keyboard and monitor. To load the Terminal package, click on the Raspberry Icon to load the menu, choose the **Accessories** category, then click on **Terminal**. The Terminal window will appear as shown in **Figure C-1**.

The Terminal window can be dragged around the desktop, resized, maximised, and minimised just like any other window. You can also make the writing in it bigger if it's hard to see, or smaller if you want to fit more in the window: click the **Edit** menu and choose **Zoom In** or **Zoom Out**, or press and hold the **CTRL+SHIFT** key on the keyboard followed by **+** or **-**.

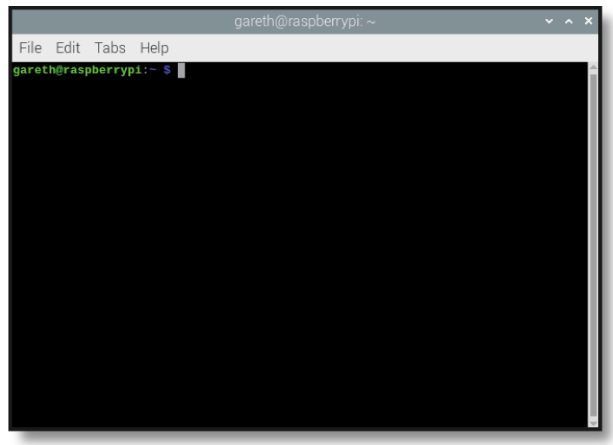


Figure C-1 The Terminal window

The prompt

The first thing you see in a terminal is the *prompt*, which is waiting for your instructions. The prompt on a Raspberry Pi running Raspberry Pi OS looks like this:

```
username@raspberrypi:~ $
```

The first part of the prompt, `username`, will be your username. The second part, after the `@`, is the host name of the computer you're using, which is `raspberrypi` by default. After the '`:`' is a tilde, `~`, which is a shorthand way of referring to your home directory and represents your *current working directory (CWD)*. Finally, the `$` symbol indicates that your user is an *unprivileged user*, meaning that you will need to elevate your permissions before carrying out certain tasks like adding or removing software.

Getting around

Try typing the following then pressing the **ENTER** key:

```
cd Desktop
```

You'll see the prompt change to:

```
pi@raspberrypi:~/Desktop $
```

That shows you that your current working directory has changed: you were in your home directory before, indicated by the `~` symbol, and now you're in the **Desktop** subdirectory underneath your home directory. To do that, you used the `cd` command — *change directory*.



CORRECT CASE

Raspberry Pi OS's command-line interface is case-sensitive, meaning that it matters when commands or names have upper- and lower-case letters. If you received a 'no such file or directory' message when you tried to change directories, check that you had a capital D at the start of **Desktop**.

There are four ways to go back to your home directory: try each in turn, changing back into the **Desktop** subdirectory each time. The first is:

```
cd ..
```

The `..` symbols are another shortcut, this time for 'the directory above this one', also known as the *parent directory*. Because the directory above **Desktop** is your home directory, this returns you there. Change back into the **Desktop** subdirectory, and try the second method:

```
cd ~
```

This uses the `~` symbol, and literally means 'change into my home directory'. Unlike `cd ..`, which just takes you to the parent directory of whatever directory you're currently in, this command works from anywhere — but there's an easier way:

```
cd
```

Without being given the name of a directory, `cd` simply defaults to going back to your home directory.

There's another way to get back to your home directory (replace `username` with your actual username):

```
cd /home/username
```

This uses what is called an *absolute path*, which will work regardless of the current working directory. So, like `cd` on its own or `cd ~`, this will return you to your home directory from wherever you are. Unlike the other methods, though, it needs you to know your username.

Handling files

To practise working with files, change (`cd`) to the **Desktop** directory and type the following:

```
touch Test
```

You'll see a file called **Test** appear on the desktop. The `touch` command is normally used to update the date and time information on a file, but if, as in this case, the file doesn't exist, it creates it.

Try the following:

```
cp Test Test2
```

You'll see another file, **Test2**, appear on the desktop. This is a *copy* of the original file, identical in every way. Delete it by typing:

```
rm Test2
```

This *removes* the file, and you'll see it disappear.



WARNING!

When you delete files using the graphical File Manager, it stores them in the Wastebasket for later retrieval just in case you change your mind. Files deleted using `rm` are gone for good, and won't go via the Wastebasket. Make sure you type with care!

Next, try:

```
mv Test Test2
```

This command *moves* the file, and you'll see your original **Test** file disappear and be replaced by **Test2**. The move command, `mv`, can be used like this to rename files.

When you're not on the desktop, though, you still need to be able to see what files are in a directory. Type:

```
ls
```

This command *lists* the contents of the current directory, or any other directory you give it. For more details, including listing any hidden files and reporting the sizes of files, try adding some switches:

```
ls -larth
```

These switches control the `ls` command: `l` switches its output into a long vertical list; `a` tells it to show all files and directories, including ones that would normally be hidden. The `r` switch reverses the normal sort order; `t` sorts by modification time, which combined with `r` gives you the oldest files at the top of the list and the newest files at the bottom. And `h` uses human-readable file sizes, making the list easier to understand.

Running programs

Some programs can only be run at the command line, while others have both graphical and command-line interfaces. An example of the latter is the Raspberry Pi Software Configuration Tool, which you would normally load from the raspberry icon menu.

To experiment with using the Software Configuration Tool at the command line, type:

```
raspi-config
```

You'll be shown an error message telling you that the software can only be run as *root*, the superuser account on your Raspberry Pi, because of your user account's status as an unprivileged user. It will also tell you how to run the software as the root, by typing:

```
sudo raspi-config
```

The `sudo` part of the command means *switch-user do*, and tells Raspberry Pi OS to run the command as the root user. The Raspberry Pi Software Configuration tool will appear as shown in **Figure C-2**.

You'll only need to use `sudo` when a program needs elevated *privileges*, such as when it's installing or uninstalling software or adjusting system settings. A game, for example, should never be run using `sudo`.

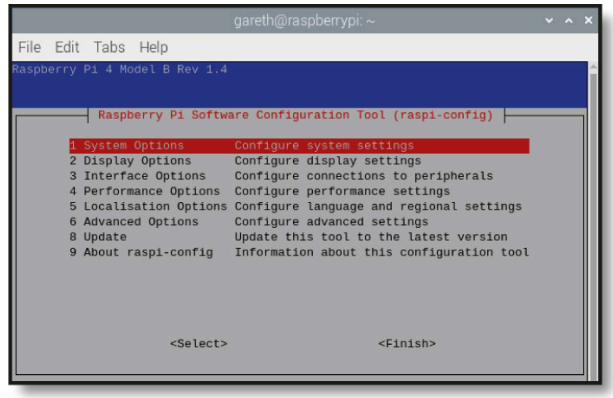


Figure C-2 The Raspberry Pi Software Configuration tool

Press the **TAB** key twice to select Finish and press **ENTER** to quit the Raspberry Pi Software Configuration Tool and return to the command-line interface. Finally, type:

```
exit
```

This will end your command-line interface session and close the Terminal app.

Using the TTYs

The Terminal application isn't the only way to use the command-line interface: you can also switch to one of a number of already-running terminals known as the *teletypes* or *TTYs*. Hold the **CTRL** and **ALT** keys on your keyboard and press the **F2** key to switch to tty2 (see **Figure C-3**).



Figure C-3 One of the TTYs

You'll need to log in again with your username and password, after which you can use the command-line interface just like in the Terminal application. Using these TTYs is handy when, for whatever reason, the main desktop interface isn't working.

To switch away from the TTY, press and hold **CTRL+ALT**, then press **F7**: the desktop will reappear. Press **CTRL+ALT+F2** again and you'll switch back to tty2 — and anything you were running in it will still be there.

Before switching again, type:

```
exit
```

Then press **CTRL+ALT+F7** to get back to the desktop. The reason for exiting before switching away from the TTY is that anybody with access to the keyboard can switch to a TTY, and if you're still logged in they'll be able to access your account without having to know your password!

Congratulations: you've taken your first steps in mastering the Raspberry Pi OS command-line interface!



Appendix D

Further reading

The Official Raspberry Pi Beginner's Guide is designed to get you started with your Raspberry Pi, but it's by no means a complete look at everything you can do. The Raspberry Pi community is globe-spanning and vast, with people using them for everything from games and sensing applications to robotics and artificial intelligence. You'll find a huge amount of inspiration out there.

Each page in this appendix highlights some sources of project ideas, lesson plans, and other material which act as a great next step now you've worked your way through the *Beginner's Guide*.

Bookshelf

Raspberry Pi icon > Help > Bookshelf



Figure D-1 The Bookshelf application

Bookshelf (shown in **Figure D-1**) is an application included with Raspberry Pi OS which lets you browse, download, and read digital versions of Raspberry Pi Press publications. Load it by clicking on the Raspberry Pi icon, select Help, and click **Bookshelf**; then browse from a range of magazines and books, all free to download and read at your leisure.

Raspberry Pi news

raspberrypi.com/news

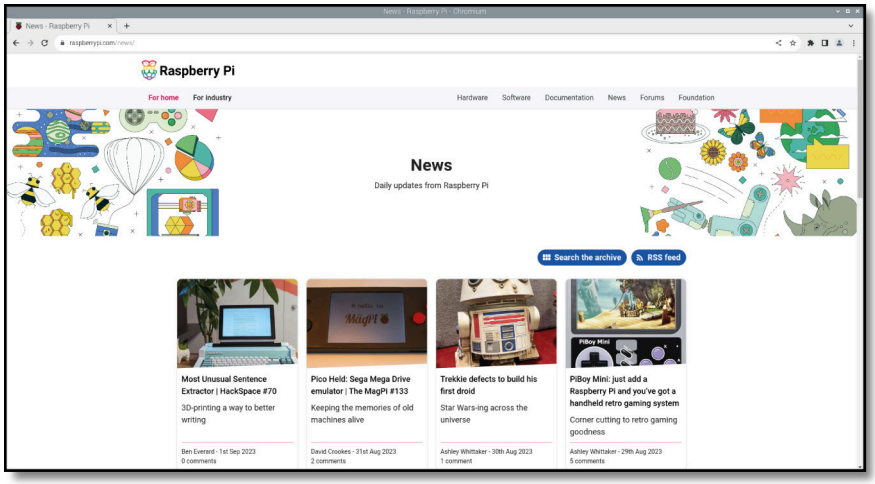


Figure D-2 Raspberry Pi news

Every weekday, you'll find a new article, covering announcements on new Raspberry Pi computers and accessories, the latest software updates, and round-ups of community projects — as well as updates from Raspberry Pi Press publications including The MagPi and HackSpace Magazine (**Figure D-2**).

Raspberry Pi Projects

rpf.io/projects

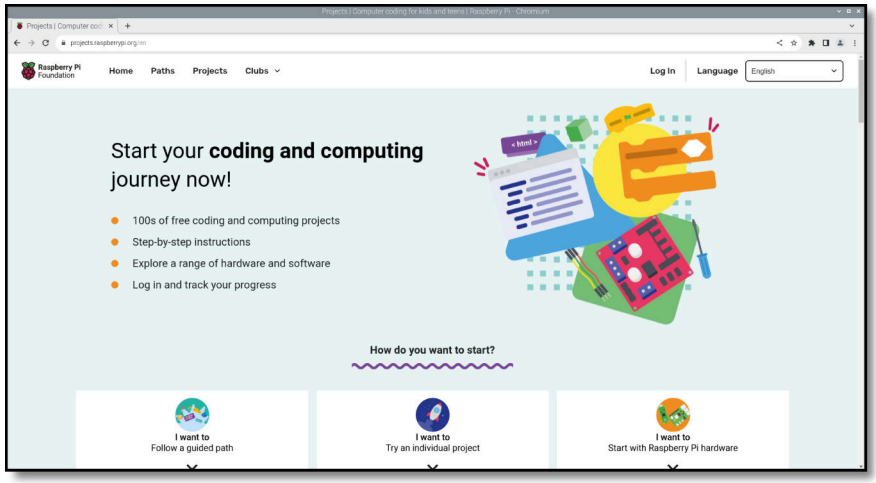


Figure D-3 Raspberry Pi projects

The official Raspberry Pi Projects site from the Raspberry Pi Foundation (**Figure D-3**) offers step-by-step project tutorials in a range of categories, from making games and music to building your own website or Raspberry Pi-powered robot. Most projects are available in a variety of languages, too, and cover a range of difficulty levels suitable for everyone from absolute beginners to experienced makers.

Raspberry Pi Education

rpf.io/education

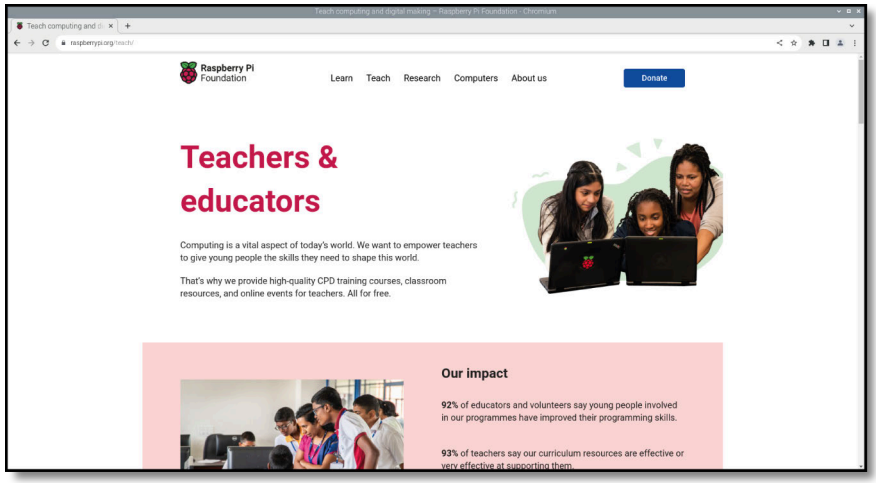


Figure D-4 The Raspberry Pi education site

The official Raspberry Pi Education site (**Figure D-4**) offers newsletters, online training, and projects with educators firmly in mind. The site also links to additional resources including free training programmes, Code Club and CoderDojo volunteer-driven coding programmes, and more.

The Raspberry Pi Forums

rptl.io/forums

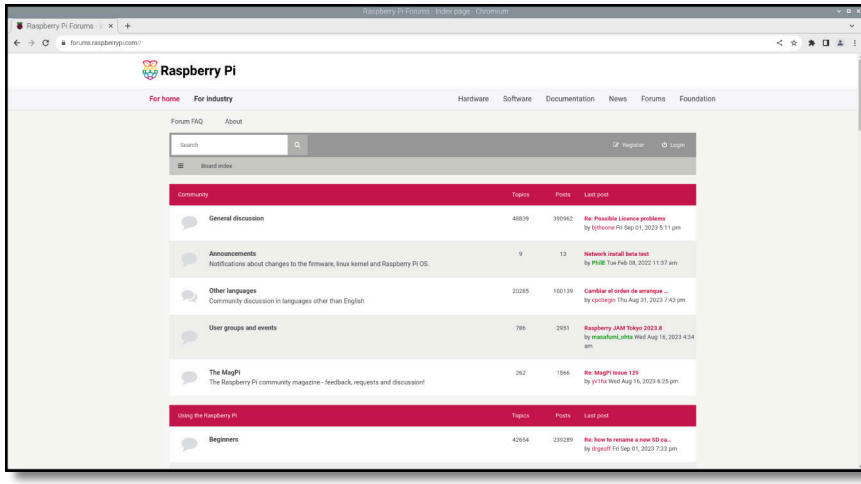


Figure D-5 The Raspberry Pi forums

The Raspberry Pi Forums, shown in **Figure D-5**, are where Raspberry Pi fans can get together and chat about everything from beginner's issues to deeply technical topics — and there's even an 'off-topic' area for general chatting!

The MagPi magazine

magpi.cc

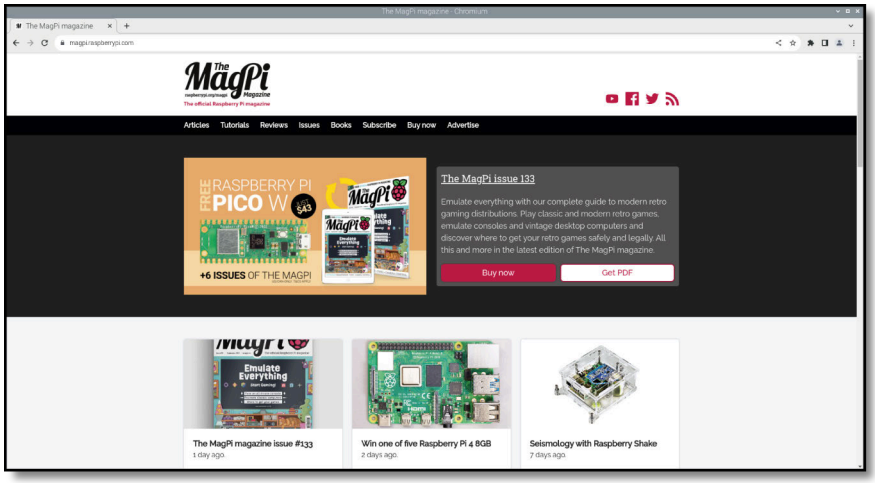


Figure D-6 The MagPi magazine

The official Raspberry Pi magazine, *The MagPi* is a monthly publication which covers everything from tutorials and guides to reviews and news, supported in no small part by the worldwide Raspberry Pi community (**Figure D-6**). Copies are available in all good newsagents and supermarkets, and can also be downloaded digitally free of charge under the Creative Commons licence. *The MagPi* also publishes books and bookazines on a variety of topics, which are available to buy in printed format or to download for free.

HackSpace magazine

hsmag.cc

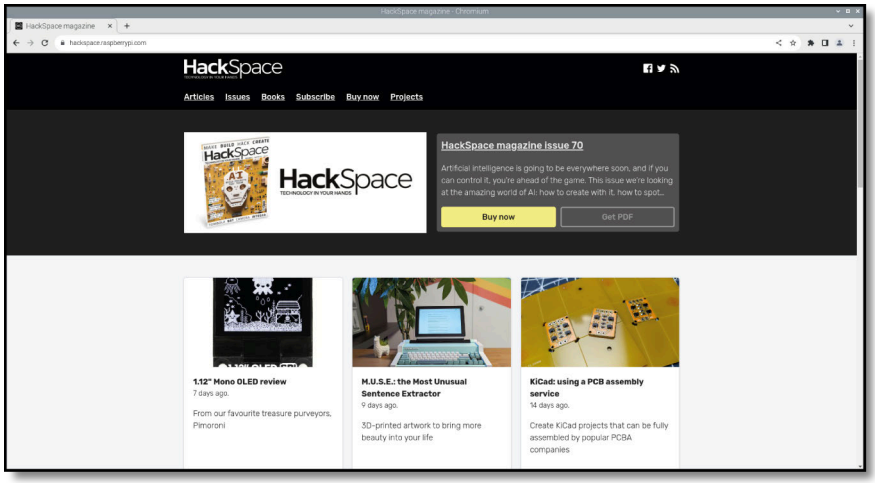
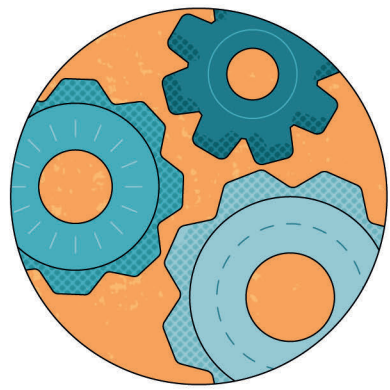


Figure D-7 HackSpace magazine

HackSpace magazine takes a look at the maker community with hardware and software reviews, tutorials, and interviews (**Figure D-7**). If you're interested in broadening your horizons beyond Raspberry Pi, *HackSpace* is a great place to start — it can be found in print at supermarkets and newsagents and is available as a digital download.



Appendix E

Raspberry Pi Configuration Tool

The Raspberry Pi Configuration Tool is a powerful package for adjusting settings on your Raspberry Pi, from the interfaces available to programs, to the way you control your Raspberry Pi over a network. It can seem a little daunting to newcomers, though, so this appendix will walk you through each of the settings in turn and explain what they do.

> **WARNING!**
>
> Unless you know you need to change a particular setting, it's best to leave the Raspberry Pi Configuration Tool alone. If you're adding new hardware to your Raspberry Pi, such as an audio HAT, the instructions should tell you which setting to change; otherwise, the default settings should generally be left as they are.



You can load the Raspberry Pi Configuration Tool from the Raspberry Pi menu, under the **Preferences** category. It can also be run from the command-line interface or a terminal using the command `raspi-config`. The layouts of the command-line version and the graphical version are different, with options appearing in different categories, depending on which version you use. This appendix is based on the graphical version.

System tab

The System tab (**Figure E-1**) displays options which control Raspberry Pi OS system settings.

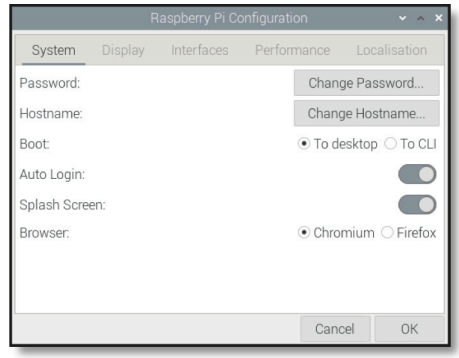


Figure E-1 The System tab

- **Password** — Click the **Change Password** button to set a new password for your current user account.
- **Hostname** — The hostname is the name by which a Raspberry Pi identifies itself on networks. If you have more than one Raspberry Pi on the same network, they must each have a unique name of their own. Click the **Change Hostname** button to choose a new one.
- **Boot** — Setting this to **To Desktop** (the default) loads the familiar Raspberry Pi OS desktop; setting it to **To CLI** loads the command-line interface as described in Appendix C, *The command-line interface*.
- **Auto Login** — When enabled (the default), Raspberry Pi OS will load the desktop without needing you to type in your username and password.
- **Splash Screen** — When enabled (the default), Raspberry Pi OS's boot messages are hidden behind a graphical splash screen.
- **Browser** — Allows you to switch between Google's Chromium (the default) and Mozilla's Firefox as your default web browser.

Display tab

The Display tab (**Figure E-2**) shows settings which control how the screen is displayed.

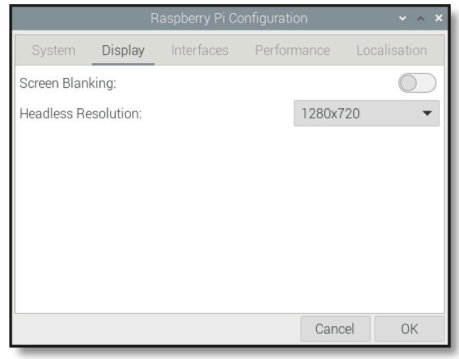


Figure E-2 The Display tab

- **Screen Blanking** — This option allows you turn screen blanking on and off. When enabled, your Raspberry Pi will turn the display black if you haven't used it for a few minutes; this protects your TV or monitor from any damage which might be caused by displaying a static image for long periods of time.
- **Headless Resolution** — This option controls the resolution of the virtual desktop when you're using Raspberry Pi without a monitor or TV attached — something known as *headless operation*.

Interfaces tab

The Interfaces tab (**Figure E-3**) displays the settings which control the hardware interfaces on your Raspberry Pi.

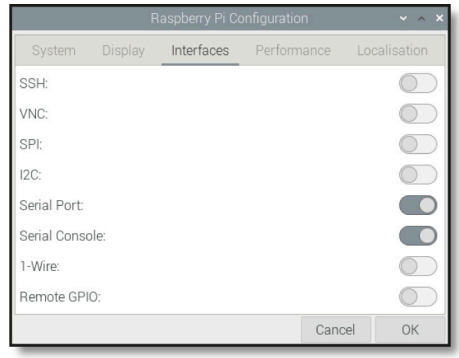


Figure E-3 The Interfaces tab

- **SSH** — Enables or disables the Secure Shell (SSH) interface. It allows you to open a command-line interface on Raspberry Pi from another computer on your network using an SSH client.
- **VNC** — Enables or disables the Virtual Network Computing (VNC) interface. It allows you to view the desktop of your Raspberry Pi from another computer on your network using a VNC client.
- **SPI** — Enables or disables the Serial Peripheral Interface (SPI), used to control certain components that connect to Raspberry Pi's GPIO pins.
- **I2C** — Enables or disables the Inter-Integrated Circuit (I^2C) interface, used to control certain components that connect to the GPIO pins.
- **Serial Port** — Enables or disables Raspberry Pi's serial port, available on the GPIO pins.
- **Serial Console** — Enables or disables the serial console, a command-line interface available on the serial port. This option is only available if the Serial Port setting above is enabled.
- **1-Wire** — Enables or disables the 1-Wire interface, used to control some hardware add-ons which connect to Raspberry Pi's GPIO pins.
- **Remote GPIO** — Enables or disables a network service which allows you to control Raspberry Pi's GPIO pins from another computer on your network using the GPIO Zero library. More information on remote GPIO is available from **gpiozero.readthedocs.io**.

Performance tab

The Performance tab (**Figure E-4**) shows settings which control the performance of your Raspberry Pi.

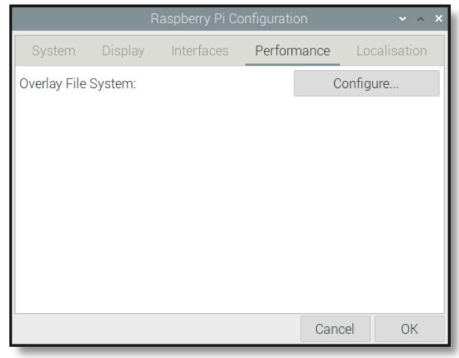


Figure E-4 The Performance tab

- **Overlay File System** — Allows you to lock Raspberry Pi's file system down so that changes only get made to a virtual disk held in memory rather than being written to the microSD card, so your changes are lost and you go back to a clean state whenever you reboot.

Models of Raspberry Pi prior to Raspberry Pi 5 will also have the following options available:

- **Case Fan** — Allows you to enable or disable an optional cooling fan connected to Raspberry Pi's GPIO header, designed to keep the processor cool in warmer environments or under extreme load. A compatible fan for the Official Raspberry Pi 4 Case is available from **rptl.io/casefan**.
- **Fan GPIO** — The cooling fan is normally connected to GPIO Pin 14. If you have something else connected to this pin, you can choose another GPIO pin here.
- **Fan Temperature** — The minimum temperature, in degrees Celsius, at which the fan should spin. Until Raspberry Pi's processor reaches this temperature, the fan will remain off to keep things quiet.

Localisation tab

The Localisation tab (**Figure E-5**) holds settings which control which region your Raspberry Pi is designed to operate in, including keyboard layout settings.

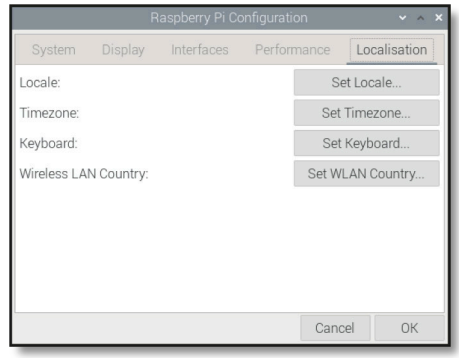


Figure E-5 The Localisation tab

- **Locale** — Allows you to choose your locale, a system setting which includes language, country, and character set. Please note that changing the language here will only change the displayed language in applications for which a translation is available, and won't affect any documents you've created or downloaded.
- **Timezone** — Allows you to choose your regional time zone, selecting an area of the world followed by the closest city. If your Raspberry Pi is connected to the network but the clock is showing the wrong time, it's usually caused by choosing the wrong time zone.
- **Keyboard** — Allows you to choose your keyboard type, language, and layout. If you find your keyboard outputs the wrong letters or symbols, you can correct it here.
- **Wireless LAN Country** — Allows you to set your country for radio regulation purposes. Be sure to select the country in which your Raspberry Pi is being used: selecting a different country may make it impossible to connect to nearby wireless LAN access points and can be a breach of broadcasting law. A country must be set before the wireless LAN radio can be used.



Appendix F

Raspberry Pi specifications

The components and features of a computer are its *specifications*, and a look at the specifications gives you the information you need to compare two computers. These specifications can seem confusing. You don't need to know or understand them to use a Raspberry Pi, but they are included here for the curious reader.

Raspberry Pi 5

Raspberry Pi 5's system-on-chip (SoC) is a Broadcom BCM2712, which you'll see written on its metal lid if you look closely enough. This features four 64-bit Arm Cortex-A76 central processing unit (CPU) cores, each running at 2.4GHz, and a Broadcom VideoCore VII graphics processing unit (GPU) for video tasks and for 3D rendering work such as games, running at 800MHz.

The SoC is connected to 4GB or 8GB of LPDDR4X (Low-Power Double-Data-Rate 4) RAM (random-access memory) which runs at 4,267MHz. This memory is shared between the central processor and the graphics processor. The microSD card slot supports up to 512GB of storage.

The Ethernet port supports up to gigabit (1000Mbps, 1000-Base-T) connections, while the radio supports 802.11ac Wi-Fi networks running on the 2.4GHz and 5GHz frequency bands, Bluetooth 5.0, and Bluetooth Low Energy (BLE) connections.

Raspberry Pi 5 has two USB 2.0 and two USB 3.0 ports for peripherals. It also has a connector for a single high-speed PCI Express (PCIe) 2.0 lane. Using an optional HAT accessory, this connector can be used to add high-speed M.2 Solid State Drive (SSD) storage, accelerators for machine learning (ML) and computer vision (CV), and other hardware.

Raspberry Pi 4 and 400

- **CPU** — 64-bit quad-core Arm Cortex-A72 (Broadcom BCM2711) at 1.5GHz or 1.8GHz (Raspberry Pi 400)
- **GPU** — VideoCore VI at 500MHz
- **RAM** — 1GB, 2GB, 4GB (Raspberry Pi 400), or 8GB of LPDDR4
- **Networking** — 1 × Gigabit Ethernet, dual-band 802.11ac, Bluetooth 5.0, BLE
- **Audio/Video Outputs** — 1 × 3.5mm analogue AV jack (Raspberry Pi 4 only), 2 × micro-HDMI 2.0
- **Peripheral Connectivity** — 2 × USB 2.0 ports, 2 × USB 3.0 ports, 1 × CSI (Raspberry Pi 4 only), 1 × DSI (Raspberry Pi 4 only)
- **Storage** — 1 × microSD up to 512GB (16GB in Raspberry Pi 400 kit)
- **Power** — 5V at 3A via USB C, PoE (with additional HAT, Raspberry Pi 4 only)
- **Extras** — 40-pin GPIO header

Raspberry Pi Zero 2 W

- **CPU** — 64-bit quad-core Arm Cortex-A53 at 1GHz (Broadcom BCM2710)
- **GPU** — VideoCore IV at 400MHz
- **RAM** — 512MB of LPDDR2
- **Networking** — Single-band 802.11b/g/n, Bluetooth 4.2, BLE
- **Audio/Video Outputs** — 1 × Mini-HDMI
- **Peripheral Connectivity** — 1 × Micro USB OTG 2.0 Port, 1 × CSI
- **Storage** — 1 × microSD up to 512GB
- **Power** — 5 volts at 2.5 amps via micro USB
- **Extras** — 40-pin GPIO header (unpopulated)

Raspberry Pi is a small, clever, British-built computer that's packed with potential. Made using a desktop-class, energy-efficient processor, Raspberry Pi is designed to help you learn coding, discover how computers work, and build your own amazing things. This book was written to show you just how easy it is to get started.

Learn how to:

- Set up your Raspberry Pi, install its operating system, and start using this fully functional computer.
- Start coding projects, with step-by-step guides using the Scratch 3, Python, and MicroPython programming languages.
- Experiment with connecting electronic components, and have fun creating amazing projects.

New in the 5th edition:

- Updated for the latest Raspberry Pi computers: Raspberry Pi 5 and Raspberry Pi Zero 2 W.
- Covers the latest Raspberry Pi OS.
- Includes a new chapter on the Raspberry Pi Pico!



Computers | Hardware | General

ISBN 978-1-912047-26-0

UK £19.99
US $24.99

Raspberry Pi PRESS

raspberrypi.com