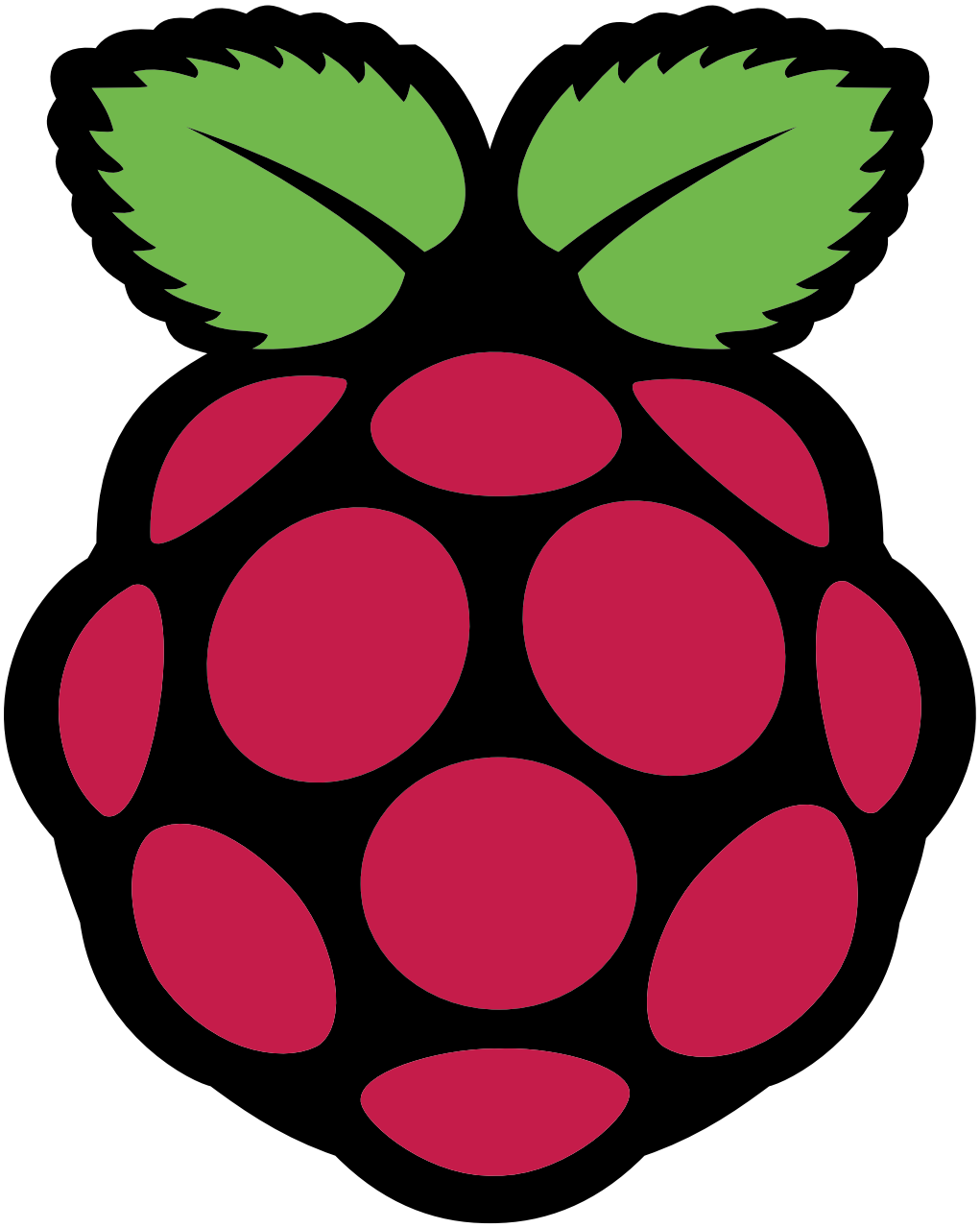
FREE KIT **NOT INCLUDED** with **digital edition**

THE *Official* RASPBERRY PI BEGINNER'S BOOK



FROM THE MAKERS OF The MagPi THE OFFICIAL RASPBERRY PI MAGAZINE

The MagPi ESSENTIALS

LEARN | CODE | MAKE

OUT NOW IN PRINT

ONLY £3.99

from

raspberrypi.org/magpi

The MagPi ESSENTIALS — From the makers of the official Raspberry Pi magazine

GET THEM DIGITALLY:

Available on the App Store

GET IT ON Google Play

WELCOME TO THE OFFICIAL RASPBERRY PI BEGINNER'S BOOK

You are holding, in your hands, a piece of history. The Official Raspberry Pi Beginner's Book isn't just a book about a computer: It's a book *with a computer*.

Not just any computer either. The Pi Zero W is an incredibly well-designed microcomputer. And it's the most creative programmable computer on earth.

With a Raspberry Pi, you can hack, make and build all kinds of different things. It could be a digital camera, or a retro games console, or a home media centre. Or a sensor on board the International Space Station, or a programmable Minecraft machine.

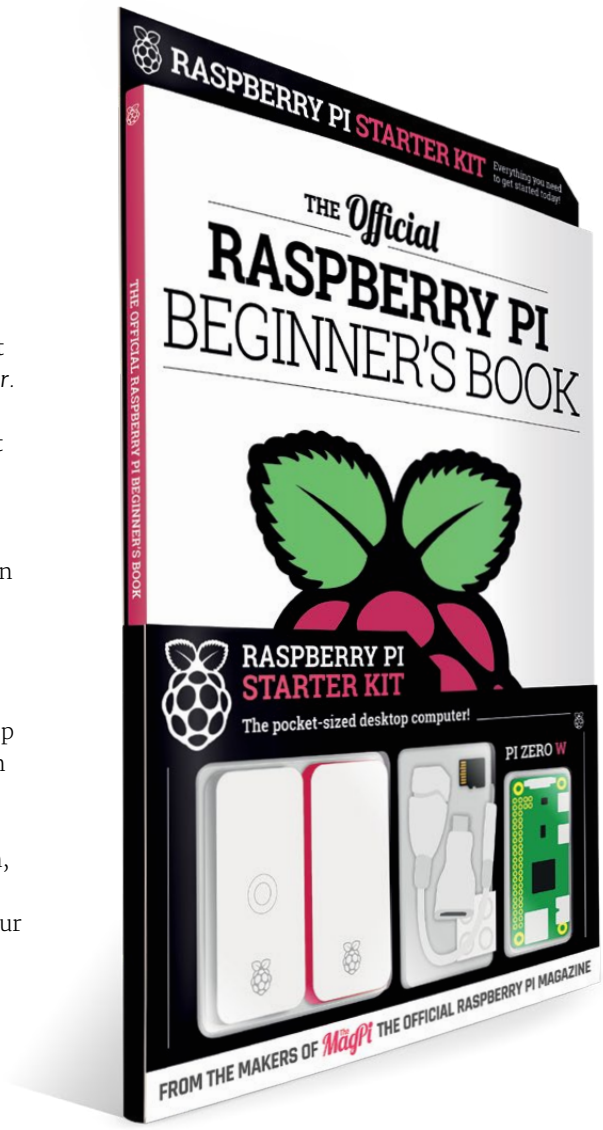
Everything you need to get started with a Raspberry Pi computer is inside this kit.

Our tutorials will guide you, step-by-step, from setting up the Pi Zero W hardware to learning how to use the Raspbian operating system, through to hacking electronics with the GPIO pins on your Pi Zero W.

You'll also learn how to program a computer with Python, the world's best programming language. We show coding the fun way, by hacking worlds in Minecraft and making your own games.

We're so glad you've got The Official Raspberry Pi Beginner's Book. Now let's get started...

Lucy Hattersley
Editor



FIND US ONLINE raspberrypi.org/magpi **GET IN TOUCH** magpi@raspberrypi.org

The MagPi

Available on the App Store

Get it on Google play

CC BY NC SA

EDITORIAL
Publishing Director: **Russell Barnes**
Editor: **Lucy Hattersley**
Features Editor: **Rob Zwetsloot**
Sub Editors: **Phil King, Jem Roberts**

DESIGN
Critical Media: **criticalmedia.co.uk**
Head of Design: **Dougal Matthews**
Designers: **Lee Allen, Daiva Bumelyte, and Mike Kay**
Illustrator: **Sam Alder**

PUBLISHING
For advertising & licensing:
russell@raspberrypi.org
Comms Director: **Liz Upton**
CEO: **Eben Upton**

DISTRIBUTION
Seymour Distribution Ltd
2 East Poultry Ave
London
EC1A 9PT | **+44 (0)207 429 4000**

MAGAZINE SUBSCRIPTIONS
Select Publisher Services Ltd
PO Box 6337
Bournemouth
BH1 9EH | **+44 (0)1202 586 848**

CONTRIBUTORS
Phil King, Laura Sach, Marc Scott, K.G. Orphanides, and Clive Webster

recycle
When you have finished with this magazine, please recycle it.

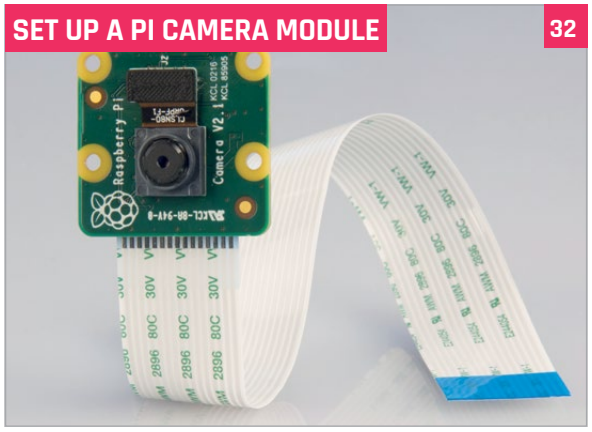
This bookazine is printed on paper sourced from sustainable forests and the printer operates an environmental management system which has been assessed as conforming to ISO 14001.

 ISBN: 978-1-912047-70-3.

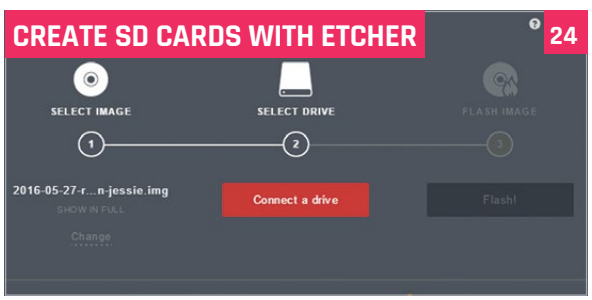
Contents

GETTING STARTED WITH YOUR PI ZERO W 06

SET UP A PI CAMERA MODULE 32



CREATE SD CARDS WITH ETCHER 24



Contents

GETTING STARTED WITH PI ZERO W

Ultra-low-cost, super-tiny, and incredibly powerful, the Pi Zero W is the tiniest Raspberry Pi computer

The Pi Zero is an ultra-low-cost and incredibly small microcomputer packed onto a single board.

It's the computer included with The Official Raspberry Pi Beginner's book, and is small enough to fit in the palm of your hand.

For all that, the Pi Zero W is packed with enough power to handle demanding computer projects.

Despite its diminutive stature, the Pi Zero W is no toy. The Pi Zero W is a fully fledged microcomputer with a 1GHz ARM CPU and 512MB RAM. It packs enough technology to run the full version of Raspbian, just the same as the larger, and more powerful, Raspberry Pi 3.

The Pi Zero W is a rewarding device that's ideal for creating Internet of Things, wearable, and embedded projects.

To keep the size down, the Pi Zero features a smaller-than-normal mini HDMI socket. You'll almost certainly need a mini HDMI-to-HDMI adapter or cable to connect the Raspberry Pi to a television or monitor.

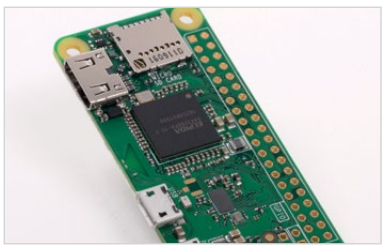
The Pi Zero board uses the same micro USB power input as other Raspberry Pi devices, and you can use an official adapter or salvage a high-quality power supply from a mobile phone (2A output is recommended).

Ports are minimal on the Pi Zero, and it sports a single USB port that's

PI ZERO W

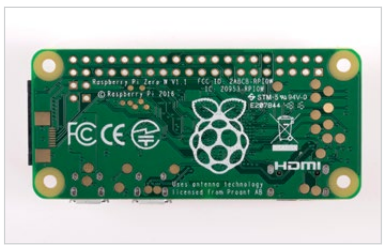
Powerful processor

The Pi Zero W packs a sizzling 1GHz single-core ARM 11 CPU with 512MB RAM. Despite its diminutive size, it's 40 percent faster than the original Raspberry Pi model.



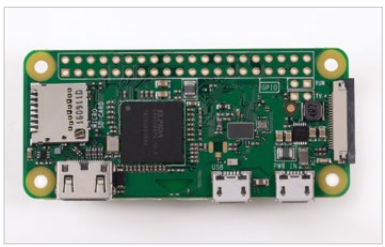
Tiny form factor

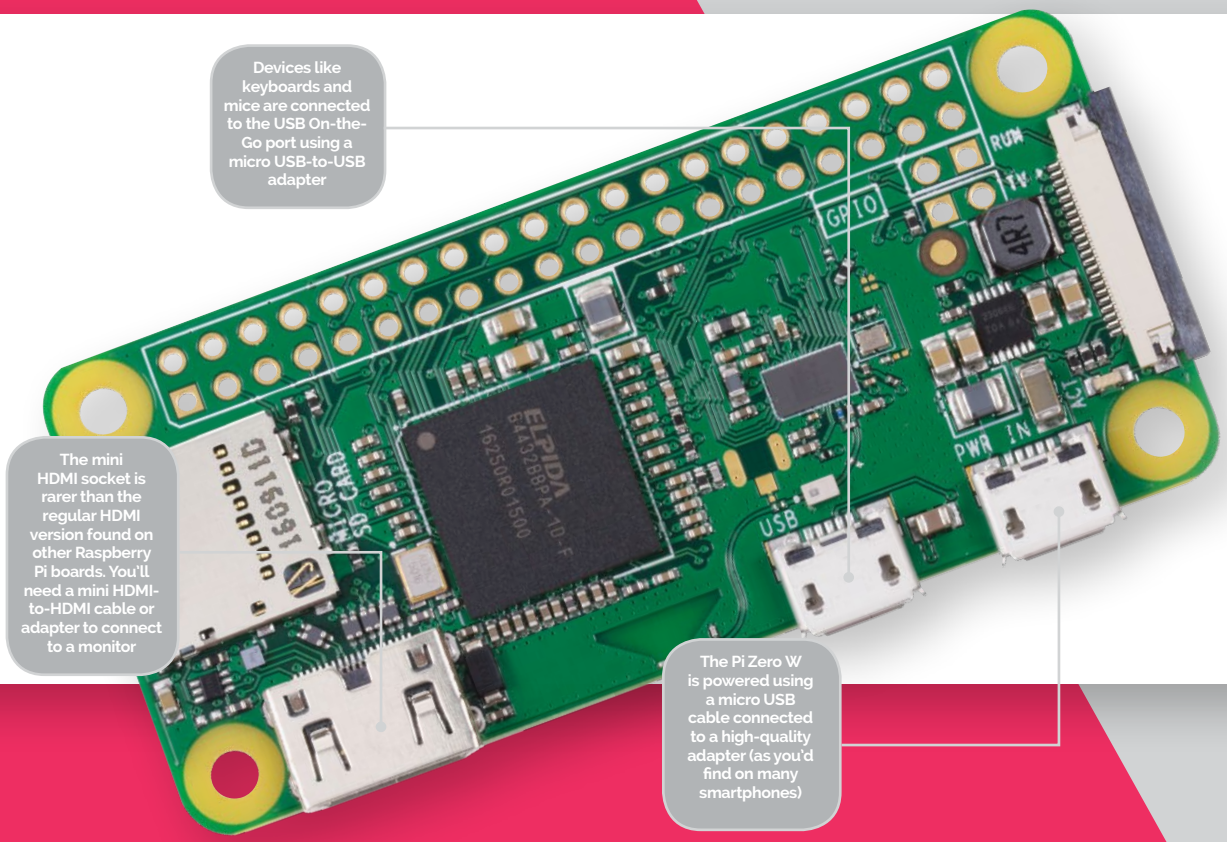
The Pi Zero W offers a full computer experience, complete with the Raspbian operating system, and is only a third the size of the original Raspberry Pi.



GPIO to go

The full GPIO header sits along the side of the Pi Zero board. These holes enable makers to attach hardware to the Pi Zero, and you can experiment with electronics projects.





smaller than a regular one. You'll need a micro USB-to-USB adapter to connect your keyboard. You may also want a USB hub to connect a mouse and other devices like a USB camera.

The Pi Zero W has a built-in camera connector. Like the other Raspberry Pi devices, you can connect a Raspberry Pi Camera Module or NoIR Camera Module directly to the Pi Zero W. This enables you to turn the Pi Zero W into a super-low-cost camera for taking photos and recording videos.

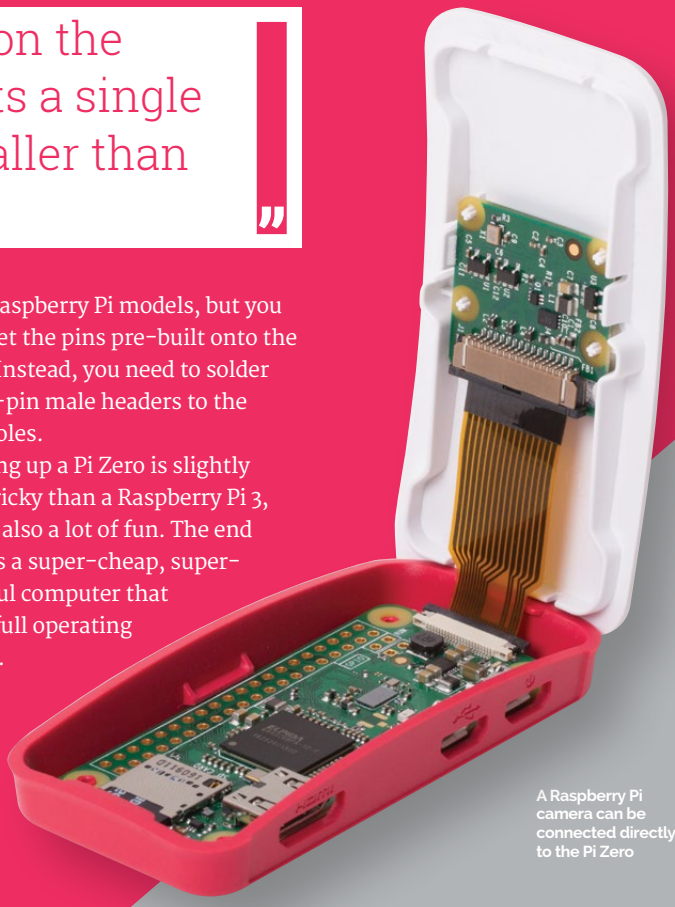
Thanks to the low power draw of the Pi Zero W, this is ideal for time-lapse photography. You just set it up and let it get on with it.

Hooking a Pi Zero W up to the internet is easy, thanks to built-in wireless networking. It also features Bluetooth so you can attach wireless devices.

Amazingly, the Pi Zero even has the full 40-pin GPIO header of the other Raspberry Pi models, but you don't get the pins pre-built onto the board. Instead, you need to solder two 20-pin male headers to the GPIO holes.

> "Ports are minimal on the Pi Zero, and it sports a single USB port that's smaller than a regular one"

Setting up a Pi Zero is slightly more tricky than a Raspberry Pi 3, but it's also a lot of fun. The end result is a super-cheap, super-powerful computer that runs a full operating system.



A Raspberry Pi camera can be connected directly to the Pi Zero

RASPBERRY PI 3

Creating amazing projects is easy with a Raspberry Pi 3. It features a more powerful processor than the Pi Zero W, and comes with GPIO pins soldered to the board

The Raspberry Pi 3 is a wonderful microcomputer that brims with potential.

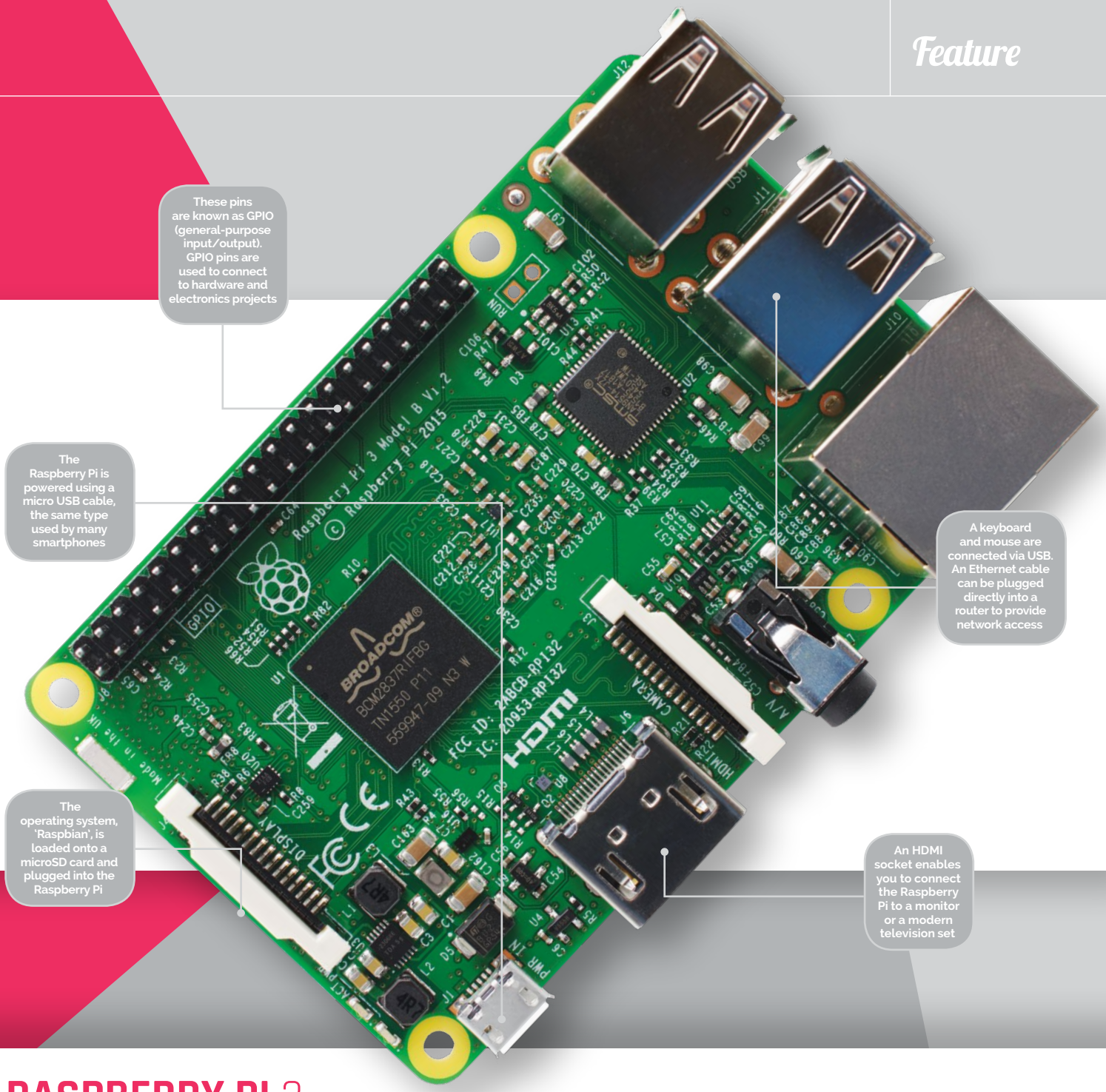
The Raspberry Pi 3 board is slightly larger than the free Pi Zero W included in your Starter Kit. It features a faster 1.2GHz ARM CPU, and slightly more RAM (1GB vs the 512MB in the Pi Zero W). It also has four standard USB ports, making it easier to connect USB devices, and an Ethernet socket. Last, but not least, it has the GPIO pins soldered to the board, for easy electronics.

However, with either a Raspberry Pi 3 or Pi Zero W you can build robots, learn to code, and create all kinds of weird and wonderful projects.

Hackers and enthusiasts have turned Raspberry Pi boards into fully automated weather stations, internet-connected beehives, motorised skateboards, and much more. The only limit is your imagination.

Creating projects with a Raspberry Pi is fun once you've mastered the basics.

So in the guide that follows, we're going to take you from newbie zero to Raspberry Pi hero. Grab your Pi Zero W and let's get started.



RASPBERRY PI 3

The Raspberry Pi 3 is the latest model, and the version with the most features

SD card

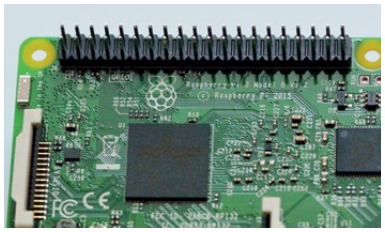
On the underside of the Raspberry Pi 3 board is the SD card slot. You preload the operating system onto a microSD card and use it to boot up the Raspberry Pi.

Wireless network

The Pi 3 and Pi Zero W both feature built-in wireless LAN and Bluetooth. This enables you to connect to a wireless router and get online without using a WiFi dongle.

1.2GHz ARM CPU

Featuring the latest 1.2GHz quad-core ARM CPU (central processing unit), the Raspberry Pi 3 is faster than many smartphones, and powerful enough to be used as a desktop computer.



EQUIPMENT YOU'LL NEED

All the kit you need to get a Raspberry Pi up and running for the first time

You don't require much to get your Raspberry Pi started: a smartphone charger, a recycled HDMI cable, and a keyboard and mouse are all you need.

Most items can be sourced from computer hardware around the house, or begged and borrowed from friends and family. If you're looking for the ultimate in low-cost computing; the Raspberry Pi is it.

You should be able to source, salvage, and scavenge most equipment you need to get a Raspberry Pi up and running.

To get the most out of your Raspberry Pi in the long term, though, you should use high-quality components.

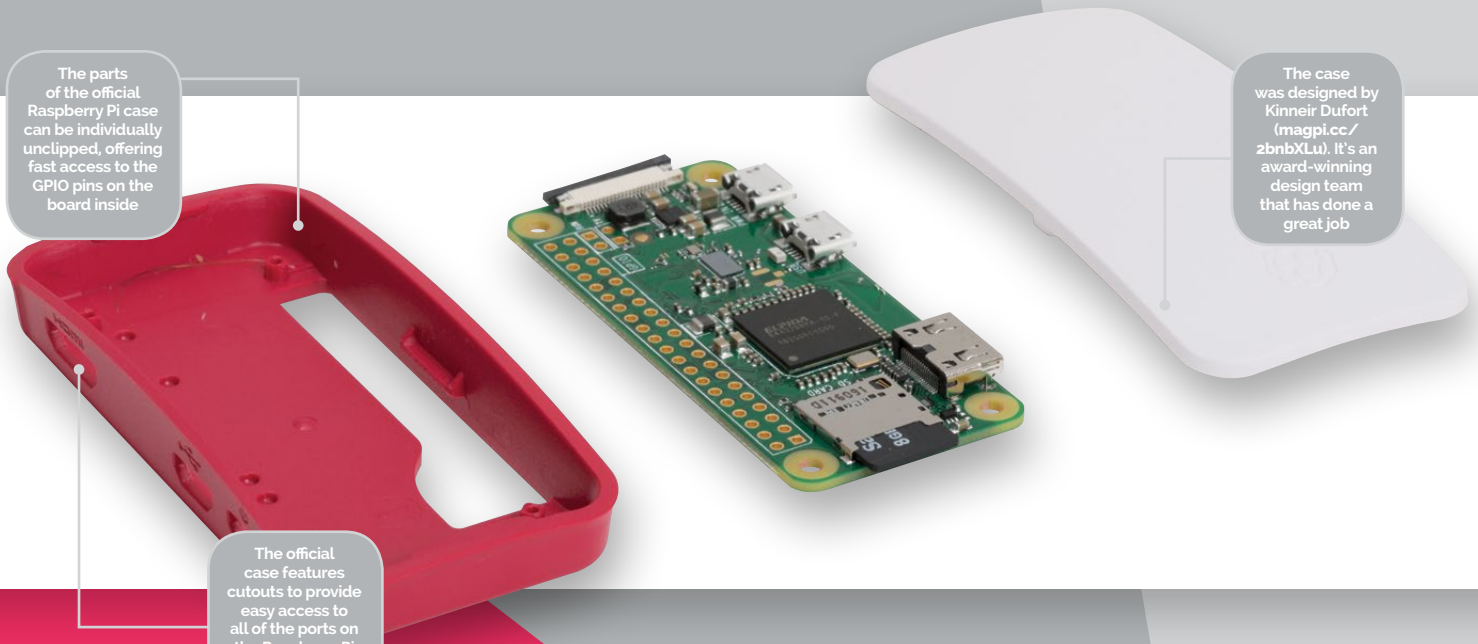
Not all USB power adapters are born equal. A reliable branded adapter will provide a steady stream of power, even when you attach multiple devices to the Pi.

Any equipment you can't recycle can be picked up from the Raspberry Pi Shop (**magpi.cc/2bnamFF**) or from distributors like Element14 (**element14.com**), Allied Electronics (**alliedelec.com**), and RS Components (**magpi.cc/2bnapBl**).

IN YOUR KIT: MICRO SD CARD

The microSD card in your kit acts as the hard drive for your Raspberry Pi. You install the Raspbian operating system onto the card, then all your documents, files, and projects are saved to it as you work. The microSD card in your Starter Kit is a high-quality piece of equipment and comes with NOOBS pre-installed.

If you want to use a larger card, and are wondering which brand and type to get, take a look at the results from benchmark tests done by Raspberry Pi fan Jeff Geerling. Some cards run up to four times as fast as others. You can read more at **magpi.cc/2bncFs3**.



HDMI cable

An HDMI cable is the easiest way to connect your Raspberry Pi to a computer monitor or television. You don't need an expensive one, and most people recycle one from an old games console or DVD player.

USB power

A good 2A or 2.5A power supply provides you with enough power to run your Pi Zero. Many people use an Android or iPhone adaptor and micro USB cable. Or you can buy an official Universal Power Supply (**magpi.cc/2a14pye**).

Keyboard and Mouse

Any standard USB keyboard and mouse can be used to control your Pi Zero W. Ones with two buttons (non-Apple mice) tend to work better. You can use a Bluetooth keyboard and/or mouse, but wired devices connected via USB make it easier to set up your Raspberry Pi.

USB hub

The Pi Zero W has a single USB socket. You may be able to connect your wired mouse to a spare USB socket on your keyboard. Otherwise, you'll need to use a USB hub like this (**magpi.cc/2ykHg8r**). Connect both the mouse and keyboard to the hub, and the hub to Pi Zero W.

SET UP YOUR PI ZERO W

You'll Need

- Your Starter Kit
- USB mouse
- HDMI cable
- micro USB power supply

Discover how to set up your Pi Zero W Starter Kit and start using it

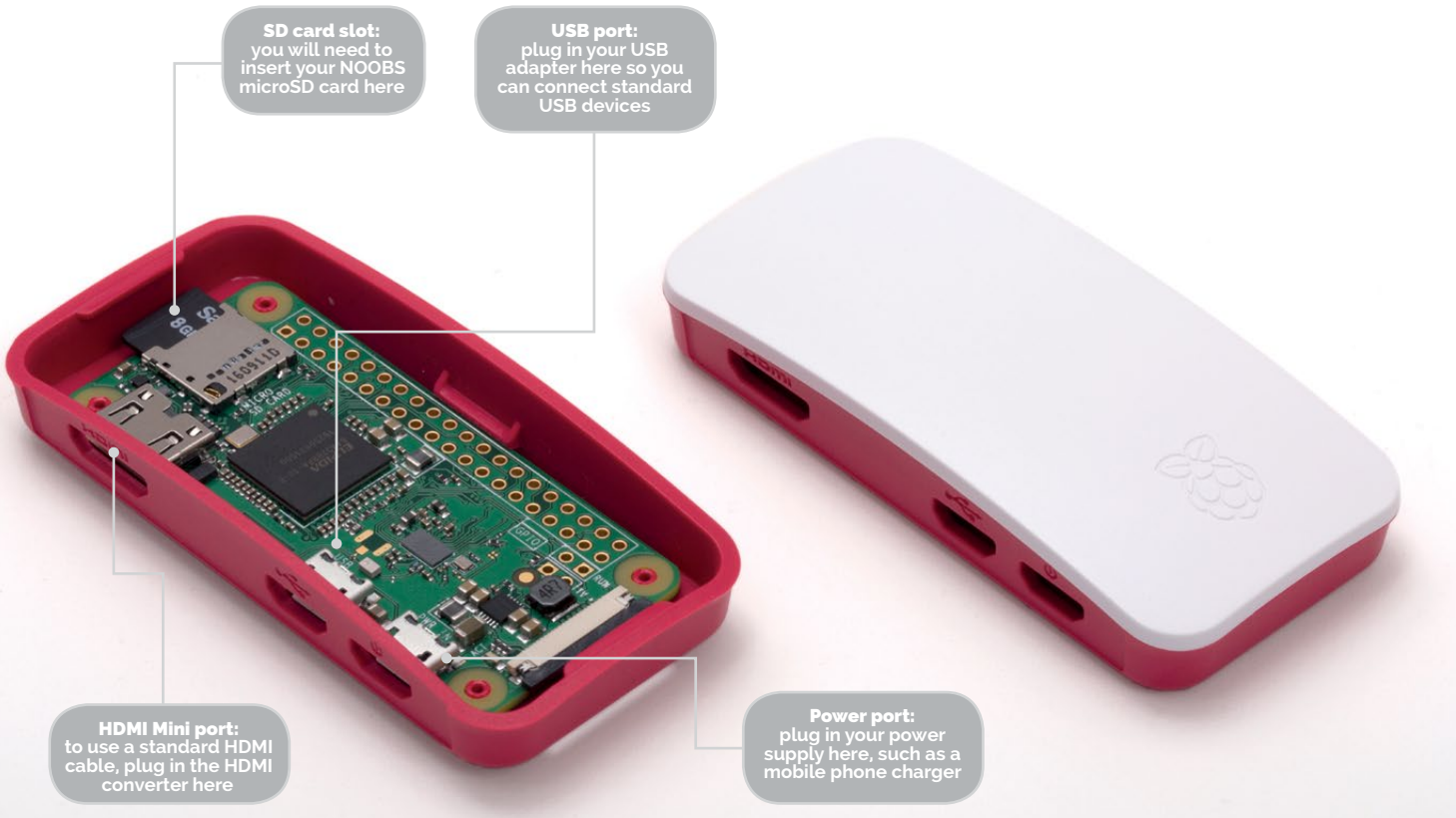
So, you've unpacked your Starter Kit and are wondering what to do with all the bits? Fear not, as here we'll guide you step by step through the process of setting up the hardware and installing the Raspbian operating system.

The Pi Zero W may be tiny, but it's a fully fledged computer. To keep the size of the board down, however, there are a few space-saving features. Instead of standard-size USB ports, it features a couple of smaller micro USB ports. One of these is for connecting a power supply, while the other is for plugging in standard USB devices – via the USB cable adapter in the Starter Kit.

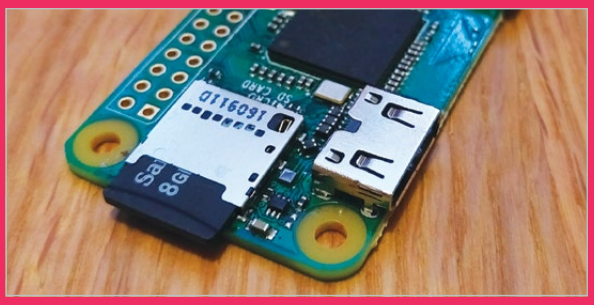
Since there's only one USB port, to connect both a keyboard and mouse you'll either have to daisy-chain them using the USB port on the keyboard (if it has one) or use a USB hub.

On the same edge of the board, you'll also see a larger port. This is the HDMI Mini connector for connecting your Pi Zero W to a TV or monitor, via a cable plugged into the HDMI adapter in the Starter Kit.

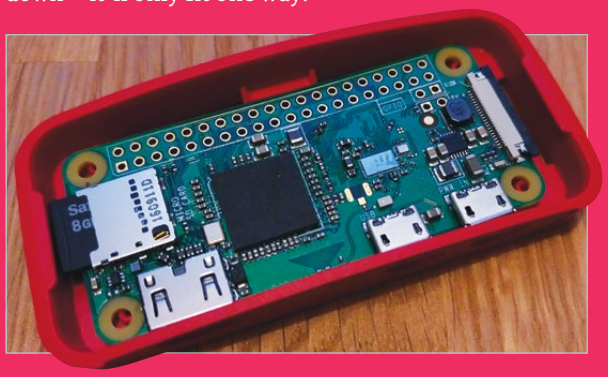
Near the HDMI Mini port is a silver slot. This is where you'll insert your microSD card from the Starter Kit. Let's start putting it all together...



HOW TO: SET UP YOUR PI ZERO W



>STEP-01
Insert microSD card
That silver slot at one end of the Pi Zero W takes microSD cards like the NOOBS one in your Starter Kit. Insert it into the slot with its metal contacts facing down – it'll only fit one way.

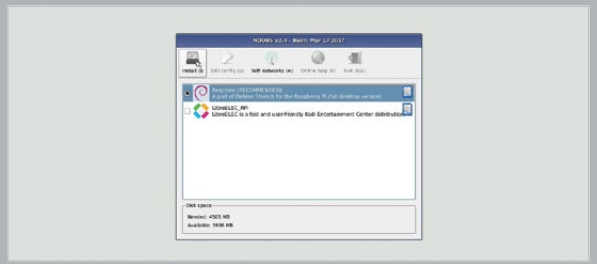


>STEP-02
Put it in the case
Now that you have the microSD inserted, you can place your Pi Zero W into the red base of the case, making sure the HDMI and USB/power ports line up with the cutouts in the case.

>STEP-03
Plug in a mouse and keyboard
Insert the USB cable adapter into the Pi Zero W port marked 'USB'. You can now plug in a standard USB mouse and keyboard. You may need a USB hub (**magpi.cc/2ykHg8r**) to connect both devices.

>STEP-04
Connect to a TV
The larger port next to the 'HDMI' marking on the Pi Zero W is an HDMI Mini port. Plug the HDMI adapter from the kit into it, then connect a standard HDMI cable from it to your TV or monitor.

>STEP-05
Power it up
The micro USB port marked 'PWR IN' is for the power supply. It's best to use an official Raspberry Pi one, but a typical mobile phone charger will usually provide enough power. Now turn it on.



>STEP-06
Install Raspbian
A menu will appear on screen. Click the checkbox to the left of Raspbian, then click Install – it'll take a while. When prompted, click OK to start up Raspbian and in a little while you'll see its standard desktop interface.

INSTALLING RASPBIAN

Discover how to use NOOBS to quickly set up the Raspbian operating system on your Raspberry Pi

Before you start using your Raspberry Pi, it needs to have an operating system (OS). This is the software used to start the hardware, and open and close programs.

Many computers use a specific operating system tied to the hardware. You'll probably be used to Windows on a PC and OS X on a Mac computer.

Most Raspberry Pi owners use an open-source operating system called Raspbian, which is based on Linux. The current version is based on a version of Linux called Debian Jessie, hence the name Raspbian (sometimes you'll hear it called 'Raspbian Jessie').

Linux is like Windows and Mac OS X, but more fun because it's open source, so anybody can view the source code and improve it.

You can install a range of different OSes on a Raspberry Pi, some based on other versions of Linux, others based on Windows, and even completely unique environments like RISC OS.

Raspbian is the official OS and the one most beginners should start with. It's the simplest to install, easiest to use, and most projects and tutorials use Raspbian as their base.

Start with NOOBS

There are two approaches to installing Raspbian and other operating systems. Beginners should start with NOOBS (New Out Of Box Software). More advanced users may copy an image file containing a whole operating system directly to the SD card.

In your Starter Kit is a microSD card pre-formatted with NOOBS.

If you wish to use a different microSD card, you must first wipe it using the Windows FAT32 format The easiest way to do this on a Mac or Windows PC is to use a program called SD Card Formatter (**magpi.cc/2bncvkm**).

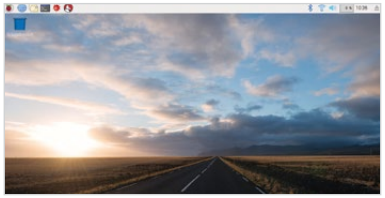
Connect your microSD card to a Mac or Windows PC, typically using a microSD-to-SD card adapter or a USB card reader, and use SD Card Formatter to erase the card.

Next, download the NOOBS ZIP file from **magpi.cc/2bnf5XF**. Extract the contents of the file and open the NOOBS folder. Copy the

AVAILABLE OSES

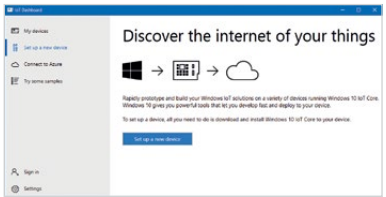
Raspbian

The official operating system is the easiest to use, and the one beginners should start with. It works a lot like other popular operating systems.



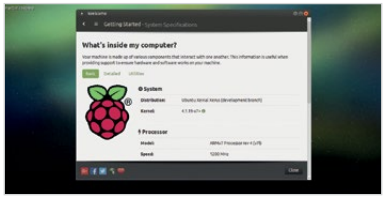
Windows 10 IoT Core

Not the full version of Windows, sadly, but Windows 10 IoT Core enables programmers to run Internet of Things and embedded projects.



Ubuntu MATE

Ubuntu is one of the world's most popular Linux operating systems, and Ubuntu MATE is a lightweight version that runs just fine on the Raspberry Pi.



NOOBS automates the process of installing Raspbian. Select the Raspbian option and click on Install to run it

NOOBS automatically copies all the files needed to run Raspbian onto your SD card

contents across to the root of the SD card. See the 'Setting up NOOBS' steps for more information.

With the NOOBS files copied across, remove the microSD card from your computer and slot it into your Raspberry Pi. Now connect the keyboard, mouse, and HDMI cable. Finally, attach the USB power to boot up the Raspberry Pi.

The Raspberry Pi will boot, displaying the NOOBS installer. By default it only has one option, 'Raspbian [RECOMMENDED]'. Place a tick next to Raspbian and click Install. Click Yes in the Confirm alert to begin installing Raspbian.

Now you just need to wait while the Raspbian file system is extracted. When it's finished, you'll see the Raspbian desktop and the message 'OS(es) Installed Successfully'. Click OK to start using your Raspberry Pi.

Installing image files

Installing an operating system from an image file is a slightly more complex procedure, but one that more advanced (and Pi Zero) users should learn. Image files are copied differently in Windows, compared to Linux and Mac computers.

In both systems, you format the microSD card to FAT32 as usual, then you download the operating system as an image file, a large file ending in '.img'. This file is then copied bit by bit as an exact replica to the microSD card.

On a Windows PC, you will copy the image file using an app called Win32DiskImager (**magpi.cc/2bndEsr**). On Mac and Linux machines, most users copy the file using a command called 'dd' in the Terminal.

Full instructions for copying image files for Windows, Mac, and Linux can be found on the Raspberry Pi website (**magpi.cc/1V50j8E**).

A good alternative for Mac owners is a program called Apple Pi Baker (**magpi.cc/2bcD53z**). This program enables you to pick the image file and the SD card, and then handles the copying automatically.

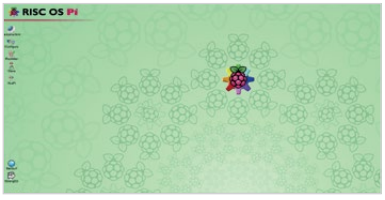
Learning how to copy image files is essential if you want to use operating systems other than Raspbian. Beginners should stick with NOOBS and Raspbian.

OSMC

OSMC (Open Source Media Centre) is an easy way to transform your Raspberry Pi into a video and audio player.

RISC OS

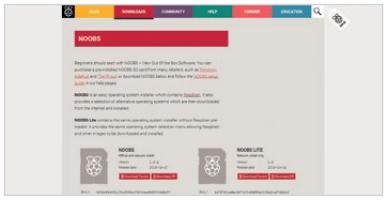
RISC OS is an operating system originally designed by Acorn Computers for ARM-based systems. It's very light and completely different.



SETTING UP NOOBS

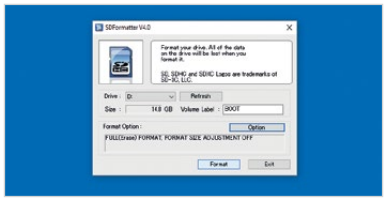
Download NOOBS

The microSD card in your kit already has NOOBS installed on it. So only use these instructions to re-install NOOBS. See page 22 for more information on NOOBS. In a browser, visit **magpi.cc/2bnf5XF**. Click on Download ZIP. Then right-click the file in Windows and choose Extract All, then Extract.



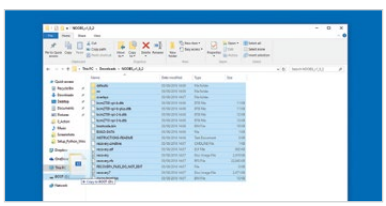
Format SD card

Open SD Card Formatter and you'll see the card in the Drive letter. Change the Volume Label to BOOT so you can identify it later. Now click Option and change Format Type to Full (Erase). Ensure Format Size Adjustment is set to Off and click OK. Click Format, then OK. Click Exit to close SD Card Formatter when it's finished.



Copy NOOBS files

Open the freshly extracted folder so you can view all the files. It should have folders called **defaults**, **os**, and **overlays**, and files including **bootcode.bin** and **recovery**. Select all of the files and drag them onto the BOOT icon in the sidebar. This copies all of the files inside the NOOBS folder to the root of the SD card. It's important to copy the files inside NOOBS, and not the NOOBS folder itself.



USING RASPBIAN

Getting to grips with the Raspberry Pi's official operating system

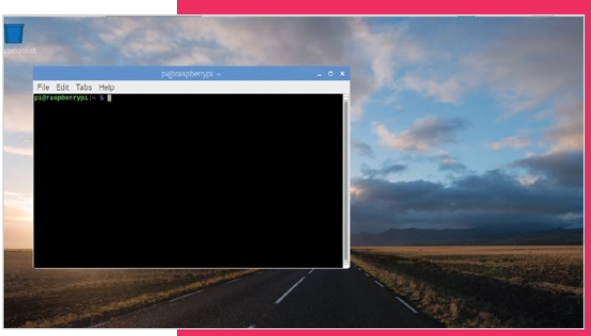
A Raspberry Pi can run many operating systems (OSes), but Raspbian is the official OS and the one most newcomers will start with.

Raspbian is a Linux operating system based on the popular Debian distribution. Fully customised for the Raspberry Pi hardware, it's usually a trouble-free experience using a Raspberry Pi with Raspbian.

One aspect of Linux that will be new to Windows and Mac users is being able to choose from different graphical interfaces. Raspbian includes one based on LXDE, which stands for 'Lightweight X11 Desktop Environment'.

This heavily modified version of LXDE enables you to use a Raspberry Pi as you would another computer. At the top left is the Menu button (raspberry icon), which offers access to most of the programs and apps installed. Programs open in windows, which you can switch between, minimise, maximise, and close using buttons.

As older readers may know, computers didn't always have windows; instead, in the past, most users used a command-line interface and entered text commands to start programs.



You'll learn how to use the Terminal and control your Raspberry Pi computer using text commands

Terminal velocity

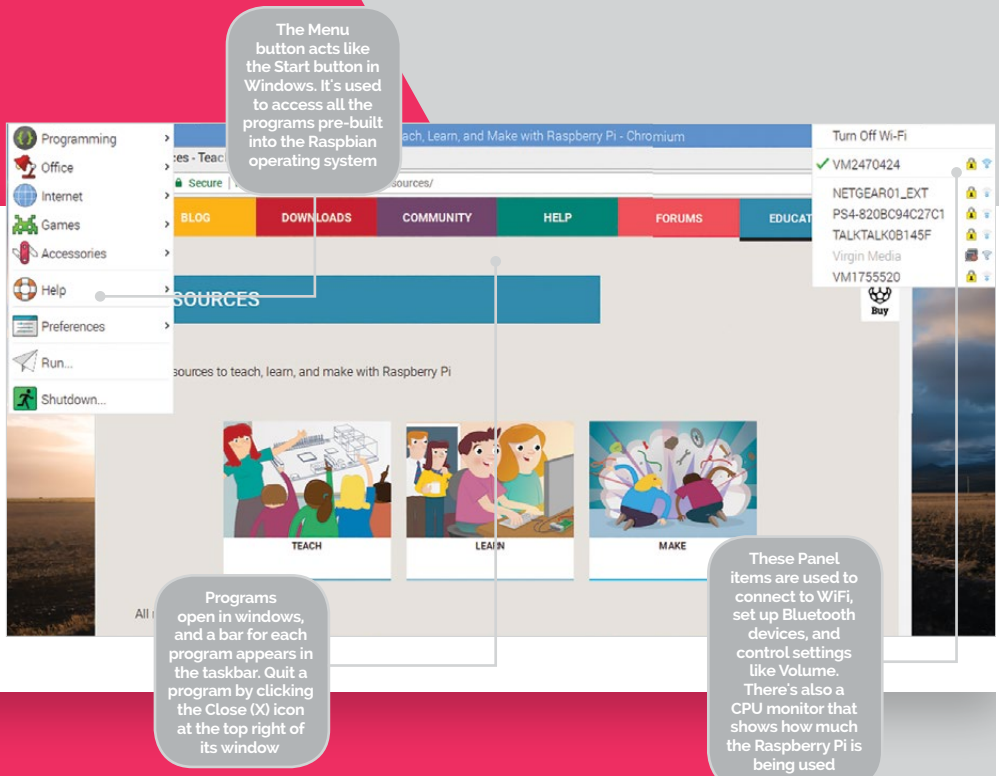
In Raspbian, you'll probably spend some time working under the hood of the desktop in a command-line environment. Next to the Menu button is the Terminal, a program that enables you to enter Linux text commands. Learning how Linux works, and how to create programs that run from the command line, is part of the joy of owning a Raspberry Pi. It's a return to classic computing where you need to learn how things actually work.

Raspbian is a great environment for learning to code. Along with easy access to the command line, you get all kinds of programming environments built in: everything from MIT's Scratch to Python and Java. You even get a full working version of Mathematica, a cool maths environment that normally costs £190 to buy, with access to real-world data.

Your kit comes with a microSD card pre-formatted with the NOOBS software. This saves you from having to install the operating system manually

Office worker

It isn't just about programming, though. You can use your Raspberry Pi as a desktop computer, and the operating system comes with LibreOffice built in. This is a full office suite of programs, similar to Microsoft Office. Its programs include Writer (word processing), Calc (spreadsheets),



Impress (presentations), Draw (vector graphics and flowcharts), Base (databases), and Math (formula editing).

Raspbian connects to the internet, and has a built-in web browser based on Google Chromium, and a more lightweight one called Epiphany (started from the Terminal be entering `epiphany-browser`). You also get an email client called 'Claws Mail'. Go to Menu > Internet.

The Pi Zero W connects to the internet using wireless LAN. It's easy to connect to a wireless network using the wireless networking icon in the top right.

Settings and software

You can adjust the settings for your Raspberry Pi in two ways: using the desktop interface or a Terminal program called raspi-config.

Choose Menu > Preferences to find a collection of different system settings. Add / Remove Software can be used to find and remove packages from Raspbian.

Appearance Settings, Audio Device Settings, Main Menu Editor, and Mouse & Keyboard Settings all adjust appearance and interaction with Raspbian. Most of the options are self-explanatory.

The Raspberry Pi Configuration choice provides more in-depth options. Here you can change your password (**raspberry** by default) and the host name of the Pi on the network (**raspberrypi** by default). You can choose to boot to the desktop or the command-line interface (CLI), and enable and disable various hardware interface options.

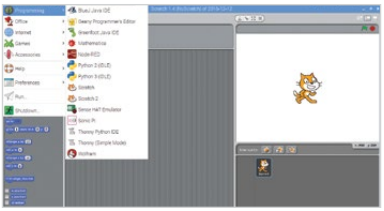
Raspi-config offers even more detailed options. Open a Terminal window and then enter `sudo raspi-config`. A blue screen with options in a grey box appears. Use the up and down arrow keys to move between options; press the right and left arrow keys to move into an option (and back to the main menu). More information on these options can be found at **magpi.cc/2bnfuJF**.

The important thing about Raspbian is not to worry about experimenting with different options and settings. Feel free to explore the menus, command line, and configuration settings. You can always reset your microSD card with NOOBS and start again.

USING THE RASPBIAN INTERFACE

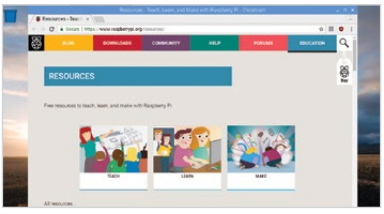
Programming tools

Raspbian comes with a selection of coding tools, found under Menu > Programming. Scratch makes it easy to learn programming concepts, and popular languages like Python and Java are ready to use right out of the box.



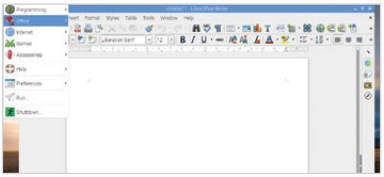
Web software

A web browser called Chromium is built into Raspbian, along with an email program called Claws Mail. There are links to Raspberry Pi Resources and *The MagPi* under Menu > Internet.



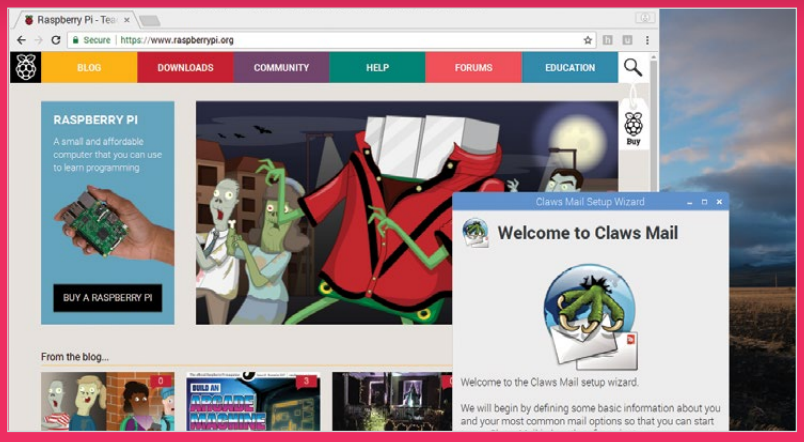
Office suite

Raspbian features powerful LibreOffice programs like Writer and Impress. These are the equivalent of Microsoft Office apps and enable you to create documents on your Raspberry Pi.



SETTING UP THE INTERNET

Get online wirelessly and quickly, with this guide to setting up wireless networks on your Raspberry Pi



A wireless internet connection enables you to get help online and set up apps like Claws Mail

The Raspberry Pi is best when connected to the internet. You can use it to browse the web, play online videos, and send and receive emails. More importantly, you can get the latest updates and install the software packages you need for any project.

To do this, you'll need to get online. With the Pi Zero W in your kit, this is easier than ever, because it has a wireless antenna built into the board.

Some other models of Raspberry Pi, including the older Pi Zero, require a WiFi dongle connected to a spare USB port.

With wireless added to your Raspberry Pi, it's easy to get online. Boot into the Raspbian desktop and look for the Wireless & Wired Network icon in the Panel (on the top right of the display).

Click the icon and you'll see a list of all the local WiFi networks. Choose your network and (if you have one) enter your password, also called the 'Pre Shared Key'.

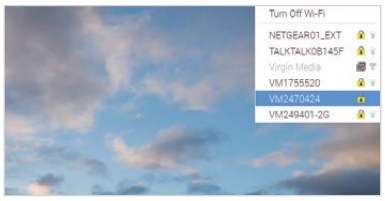
The Raspberry Pi connects to the wireless network, enabling you to get online. In this respect it's pretty much like any other computer that connects to WiFi; it will even remember the password for next time.

Once you're online, you can use the Chromium browser to fetch webpages. Click Web Browser in the Launch Bar.

CONNECTING TO A WIRELESS NETWORK

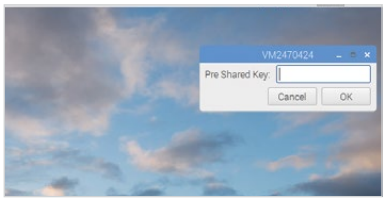
Check for networks

Click on the Wireless & Wired Network icon in the top right. Raspbian will display a list of all the wireless networks available in your local area. Click the one that's yours.



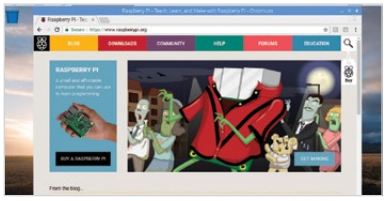
Enter your password

Enter your WiFi password in the Pre Shared Key field and click on OK. The network symbol will switch to a wireless symbol and you'll be connected.



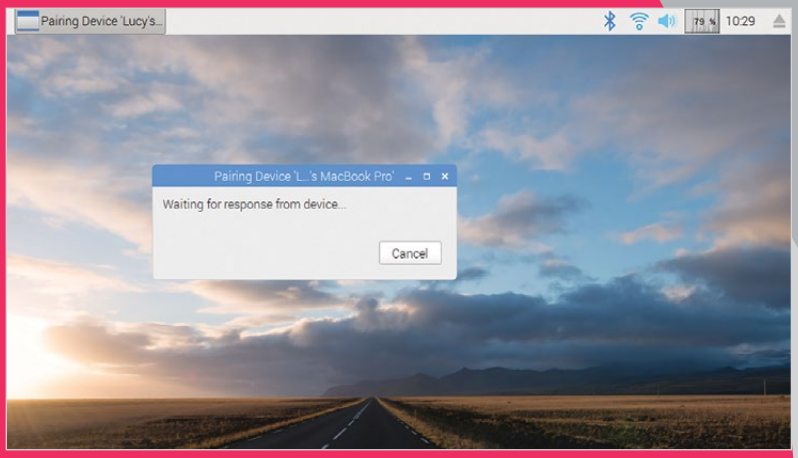
Test your connection

Test your internet connection by opening a webpage. Click on Web Browser in the Launch Bar and enter **www.raspberrypi.org** in the URL field. Press **RETURN** to load the page.



SETTING UP BLUETOOTH

Connect wirelessly to nearby devices with Bluetooth technology



Devices connected by Bluetooth work wirelessly with your Raspberry Pi

Bluetooth is another piece of technology that has been added to the Pi Zero W board. With Bluetooth you can connect wireless devices, such as mice and keyboards, directly to your Raspberry Pi.

As with wireless LAN, if you own an older Raspberry Pi model or Pi Zero, you'll need to attach a USB dongle to use Bluetooth devices.

With Bluetooth hardware on your Raspberry Pi W, it's easy to connect to a device wirelessly, a process known as 'pairing'.

You can pair wireless gaming controllers, like a PlayStation joypad, or Android smartphones. Many Raspberry Pi projects make use of Bluetooth, enabling the Raspberry Pi to communicate with nearby electronic components and devices.

The easiest way to test out Bluetooth is to set up a wireless mouse or keyboard; both are fairly easy devices to come by.

In some ways, the process is similar to connecting to a WiFi network, but the Bluetooth device you want to connect to must be set to pairing mode first. This is also known as making the device 'discoverable'. Putting a device into pairing mode varies by device; holding down the power button until an LED flashes is fairly commonplace, but check with the instructions for your device.

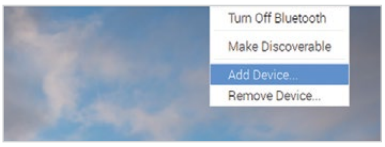
You then use the Bluetooth icon in the Raspbian desktop Panel to connect to the device: choose Bluetooth > Add Device.

It's possible to put your Raspberry Pi into pairing mode by choosing **Bluetooth > Make Discoverable** from the Panel. Then you can connect to your Raspberry Pi from other Bluetooth devices like mobile phones.

SETTING UP A BLUETOOTH DEVICE

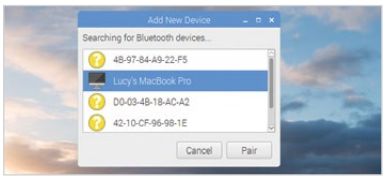
Pairing mode

Start by putting your Bluetooth device in Pairing / Discoverable mode. We're using an Apple wireless keyboard here. Hold down the power button until the LED flashes. Click Bluetooth in the Panel and choose Add Device.



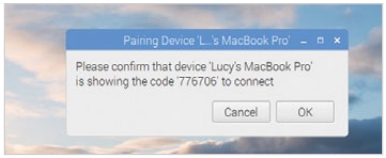
Add new device

The Add New Device window opens and will scan for nearby Bluetooth devices. Some will have names, others just identifying numbers (check on the device). Choose a device from the list and click Pair.



Enter code

The Pi now attempts to pair with the Bluetooth device. You'll be asked to enter a code on the keyboard; press the buttons and **RETURN**. You can now start using the Bluetooth device with your Raspberry Pi.



GETTING TO KNOW GPIO

Discover the joy of electronics by hooking up components, wires, and hardware to the pins on a Raspberry Pi board

One of the most powerful and fun features of the Raspberry Pi is the row of pins at the top. Known as 'GPIO' (general-purpose input/output), these pins enable you to hook up the Raspberry Pi to additional hardware and electronics.

On a larger board, such as the Raspberry Pi 3, the pins are soldered in place. With the Pi Zero W included in your kit, you'll need to solder the GPIO pins to the board.

There are lots of hardware attachments for the Raspberry Pi that connect directly to the GPIO pins. Many are known as HATs (Hardware Attached on Top). These connect directly to the GPIO and sit on top of the Raspberry Pi. More importantly, HATs are designed to work as soon as you connect them to the Raspberry Pi, so hardware branded as a HAT is easier to set up.

> "You can connect the GPIO pins to all kinds of electronic circuitry and control it"

The real joy of GPIO isn't using pre-made hardware, but building your own electronics projects. You can connect the GPIO pins to all kinds of electronic circuitry and control it using the Raspberry Pi. With the right cables, you can hook the GPIO pins up to switches, buttons, sensors, buzzers, and all manner of electronic gizmos and widgets. These are used to learn all about electronics hardware and circuit building.

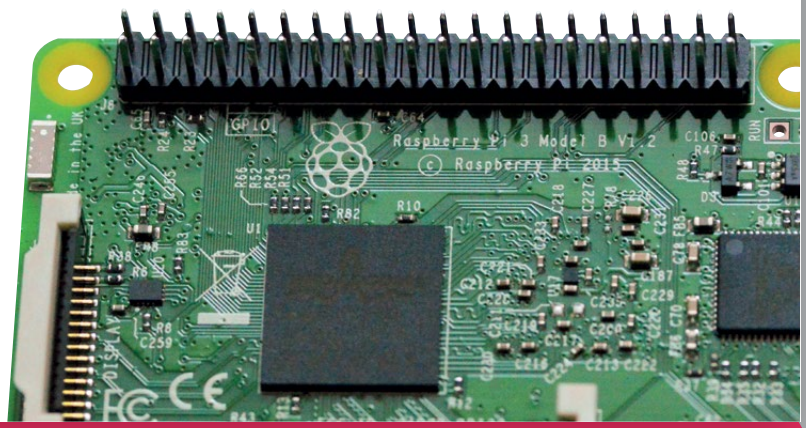
BREADBOARDS AND BREAKOUTS

Electronic components are plugged into the holes on the breadboard, and components in adjacent holes are linked. In this way, you can build up a test circuit without having to actually solder components together.

If you follow the instructions, connecting directly to the GPIO pins on a Raspberry Pi is safe, but randomly plugging in wires and power sources to the Raspberry Pi may cause bad things to happen, especially plugging in devices that use a lot of power (like motors).

Because of this, many electronics enthusiasts use a device known as a 'breakout cable' between the Raspberry Pi and breadboard. The breakout cable plugs into the GPIO pins, and into the breadboard.

There are also devices like the Explorer HAT that combine a breakout with a breadboard and enable you to create prototype circuits.



While it's possible to wire parts directly to the GPIO pins, most tinkerers place electronic components in a breadboard and connect this to the Raspberry Pi. Unlike the type used for cutting bread, an electronic breadboard is a plastic slab with lots of holes in it.

Wiring a breadboard (or circuit) directly to the GPIO pins is generally safe, as long as you avoid circuits with external power sources. Most tinkerers invest in a breakout cable to go with the breadboard (see 'Breadboards and breakouts').

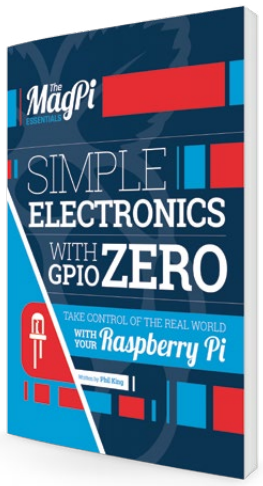
With your circuit set up, you then control the GPIO pins in a programming environment like Python or Scratch. GPIO pins are set to input or output mode. GPIO outputs are easy because the pin is switched on or off (known as HIGH or LOW in computing terms).

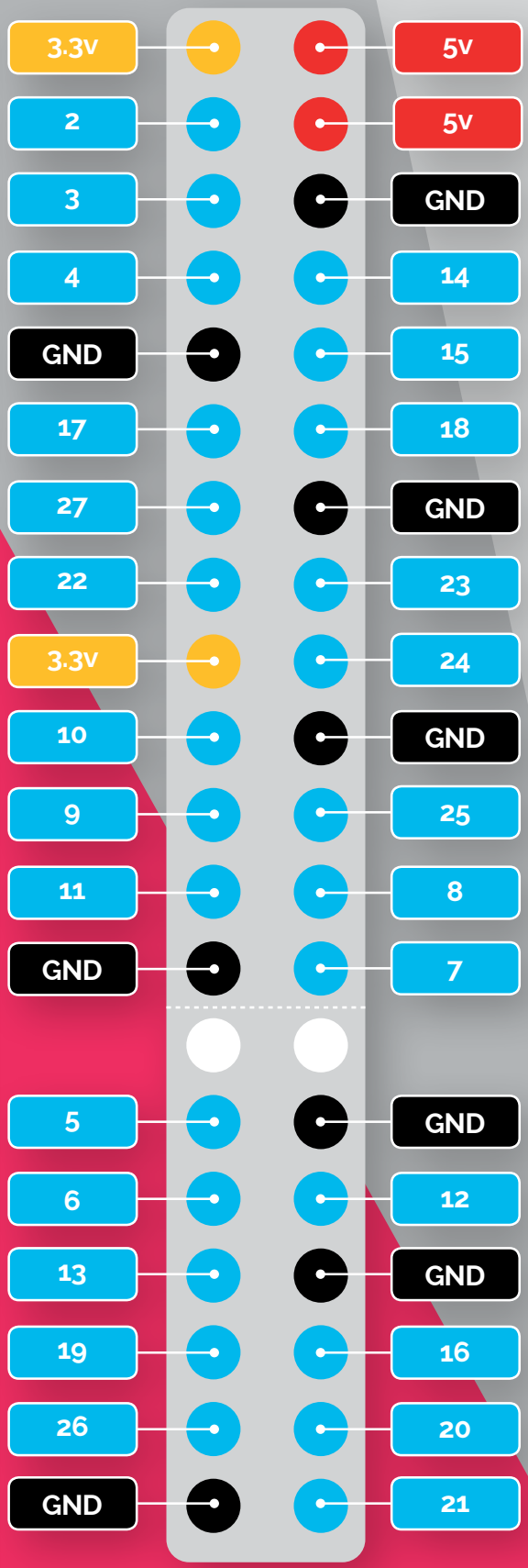
GPIO input is a bit more tricky. In this case, the GPIO pin is set to HIGH or LOW and responds to a change from a circuit. A button (or other electronic component) can change the circuit from LOW to HIGH, or HIGH to LOW, with the Raspberry Pi coded to respond accordingly. This is often referred to as 'pull up' or 'pull down'. Don't worry: if this all sounds complicated, you can get started by using GPIO Zero to make programming much easier.

Never underestimate the pure fun you can get from a little computer, a bunch of pins, and a handful of electronic components. Discovering how to use GPIO is a great way to spend your time.

GPIO ZERO ESSENTIALS

Learning to use the GPIO pins is the route to having real fun with a Raspberry Pi. It's a big subject, with lots of tricks and tinkering to discover. Our *GPIO Zero Essentials* book teaches you the basics (and beyond) of using the GPIO port with the GPIO Zero Python library. See **magpi.cc/GPIOZero-book** for more information.





There are 40 GPIO pins, each with a specific function. Use this image as a handy guide whenever you're programming electronics

BEGINNER'S GUIDE TO NOOBS

The easiest way to set up a Raspberry Pi with Raspbian, and other operating systems, is to use the NOOBS installer

You'll Need

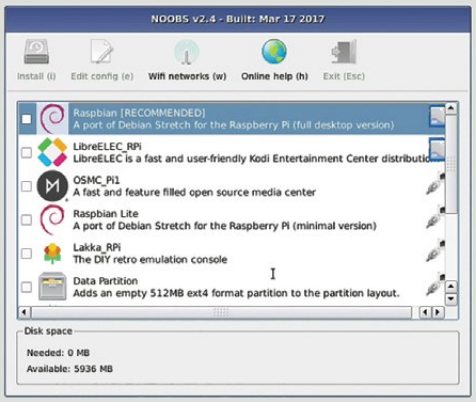
- micro SD card (8GB or larger)
- NOOBS installation files
- Mac, PC or Linux machine

IMPORTANT

Your starter kit micro SD card is already set-up with NOOBS and ready to use

Click the Install button to format your micro SD card and install the selected operating system

Connect to a wireless network (or attach an Ethernet cable) to access a wider range of operating systems



The 'Raspbian [RECOMMENDED]' option is the official operating system and is available offline. The 'x' next to it indicates that this OS is going to be installed

One of the things we love about the Raspberry Pi is just how easy it is to get started. A lot of this is down to a custom, simple-to-use installer called NOOBS.

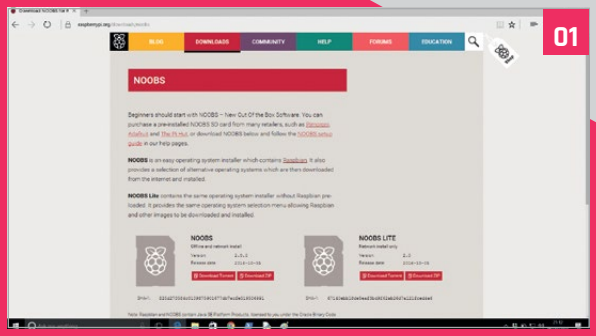
NOOBS (or 'New Out Of Box Software', to give it its full name) is a unique installation image and an essential tool for newcomers. With NOOBS loaded on a micro SD card, you can install a wonderful range of operating systems for your Raspberry Pi.

Your kit contains a micro SD card pre-loaded with NOOBS. When you start up a Raspberry Pi with NOOBS for the first time, you're given the option to install the Raspbian operating system. Connect the Raspberry Pi to a network and you'll also get a bunch of other operating systems to choose from.

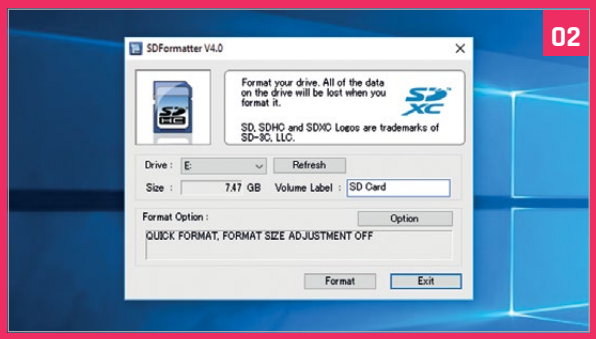
This tutorial shows you how to format any micro SD card and turn it into a NOOBS installer.

From there it's just a matter of picking the operating system you want and letting NOOBS do its thing. The NOOBS installer wipes the micro SD card and sets up the operating system for you. When the Raspberry Pi restarts, you'll no longer see NOOBS, just your operating system.

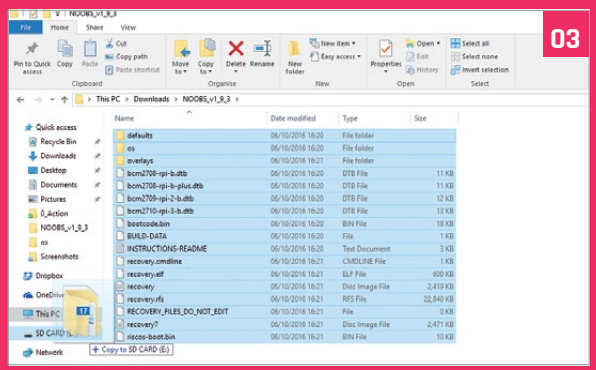
In this tutorial, we're going to help absolute newcomers install Raspbian (the official operating system). But don't forget: you can use NOOBS to experiment with other operating systems. NOOBS isn't just great for beginners; it's also ideal for exploring what other operating systems have to offer.



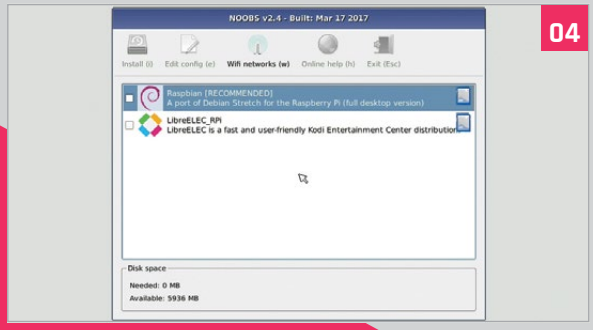
>STEP-01
Download NOOBS
Open your web browser and visit **raspberrypi.org/downloads/noobs/**. Click on Download ZIP under 'NOOBS Offline and network install'. Save the ZIP file to your Downloads folder and extract its contents.



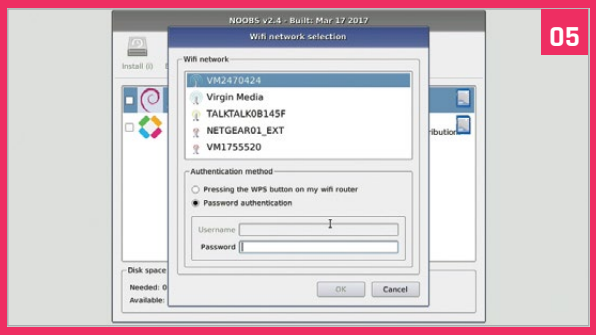
>STEP-02
SD card
Download SD Card Formatter from **sdcard.org** and open the program (click Yes in the User Account Control alert on Windows). Attach your micro SD card to the computer, and the card will appear in Drive. Enter 'SD CARD' in the Volume Label so you can identify it in the next step. Click Format (enter your password on a Mac). Answer OK to the alerts.



>STEP-03
Copy the files
Return to your Downloads and open the folder containing the NOOBS files. Make sure you're looking at the files inside the folder, and not the folder itself. Select all the files in the NOOBS folder and drag them to the SD CARD folder (in the sidebar).



>STEP-04
Power up
Eject the micro SD card from your computer. Place it into your Raspberry Pi and power it up. You'll be greeted by the NOOBS v1.9 screen. If you haven't connected to the internet, you'll only see a single option: 'Raspbian [RECOMMENDED]'.



>STEP-05
Connect to network
To access more operating system options, connect the Raspberry Pi to a network. Attach an Ethernet cable or click 'WiFi networks'. Choose your wireless network and enter the password. Click OK. You'll now see a wider range of options (as shown in the main image opposite).

>STEP-06
Install Raspbian
We're going to go with Raspbian, so click to put an X in the box next to 'Raspbian [RECOMMENDED]' and click Install. Click Yes in the alert window. The NOOBS software is copied to the micro SD card. NOOBS displays 'OS(es) Installed Successfully' when the software is installed. Click OK and the Raspberry Pi will restart and boot into the operating system.

CREATE SD CARDS WITH ETCHER

You'll Need

- Raspberry Pi
- micro SD card
- Etcher

The easiest way to burn OS image files to your Raspberry Pi SD cards

Copying operating system (typically Raspbian) image files to a micro SD card is an essential part of getting started with a Raspberry Pi. It can be a long-winded process, and is often difficult for newcomers to grasp.

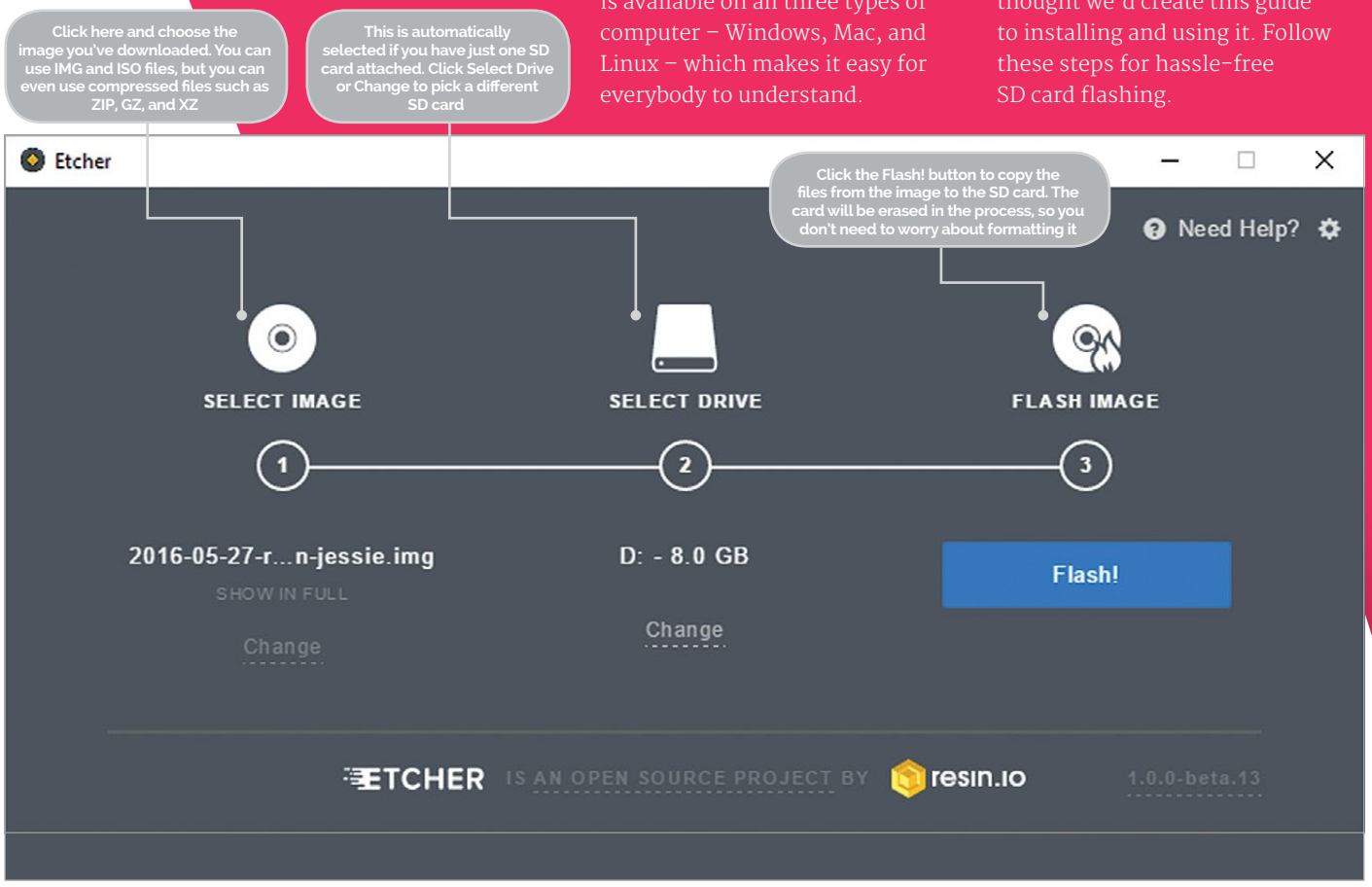
Mac and Linux users typically use the **dd** command in the terminal, while Windows users require a program such as Win32DiskImager.

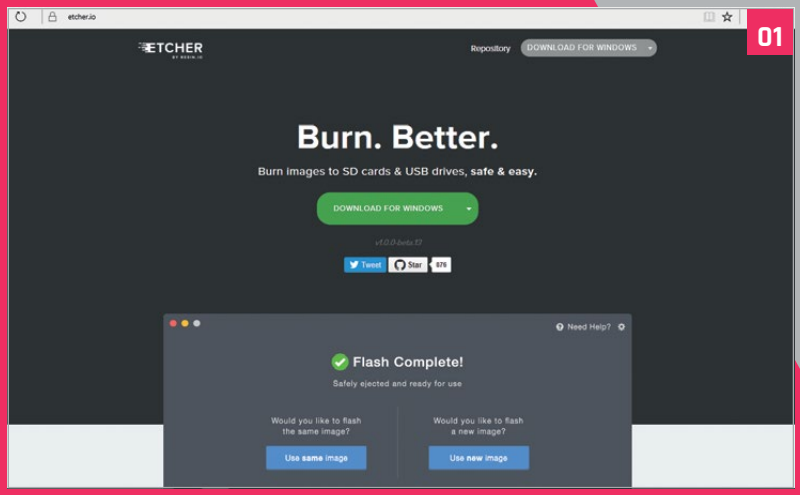
So we were pleased to come across Etcher (**etcher.io**). Etcher turns the whole process of flashing an OS image file into three simple steps: Select Image, Select Drive, and Flash Image.

More importantly, the same program, with the same interface, is available on all three types of computer – Windows, Mac, and Linux – which makes it easy for everybody to understand.

Etcher takes a lot of the stress out of flashing a drive. Etcher won't write to your hard drive volumes unless you check Unsafe Mode in Settings. Unsafe Mode is handy if you want to flash a USB thumb drive or other internal drive, but it's disabled by default, making the process safer for newcomers.

We like Etcher so much, we thought we'd create this guide to installing and using it. Follow these steps for hassle-free SD card flashing.





>STEP-01
Install in Windows or Mac

Download and install Etcher from the **etcher.io** website. Double-click the .exe file in Windows and follow the Etcher setup wizard. Drag the Etcher app to your Applications folder on a Mac and double-click to open it. In Windows, run Etcher in Administrator Mode: right-click on Etcher and choose 'Run as administrator'.

>STEP-02
Install on Linux

Download the AppImage file from the Etcher website. Open a terminal window and enter:

```
cd Downloads
chmod a+x Etcher-linux-x64.AppImage
./Etcher-linux-x64.AppImage
```

>STEP-03
Download your OS image

Download a copy of the latest Raspbian image from **raspberrypi.org/downloads** (or the OS image you want to install). Unzip the file after it has downloaded. Double-click the file in Mac or Linux (or use **unzip** in a terminal window). In Windows, right-click the file and choose Extract All. Etcher can install directly from a ZIP file, but the process takes a lot longer.

>STEP-04
Select the image

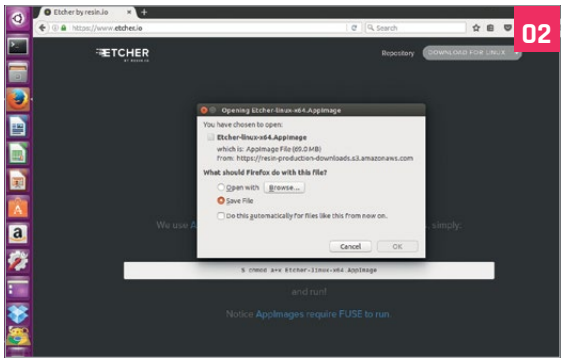
Click **Select Image** in Etcher. Use the file manager window and locate the image you unzipped in the previous step. Click **Open**. The image will appear under Select Image, and **Connect a drive** will highlight red.

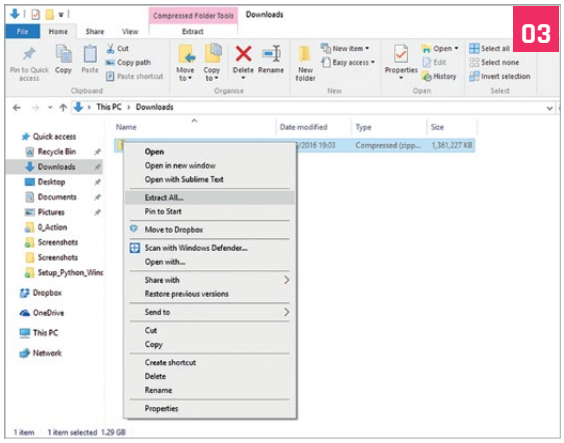
>STEP-05
Insert your SD card

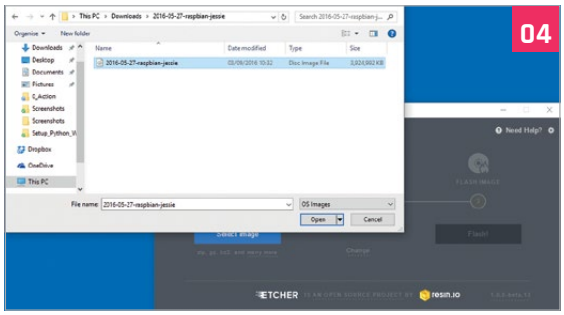
Attach your SD card to the computer. Etcher will select it automatically. Etcher won't write to your hard drives by default, but check that the SD card is listed correctly. Now click **Flash!** to write the image file to the SD card.

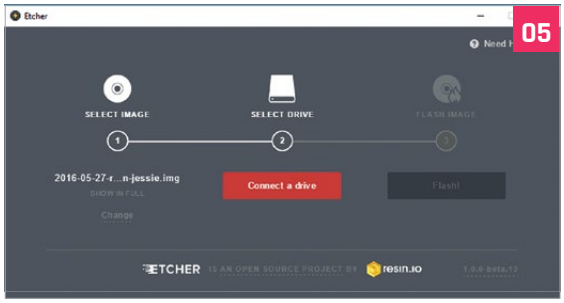
>STEP-06
Writing the image

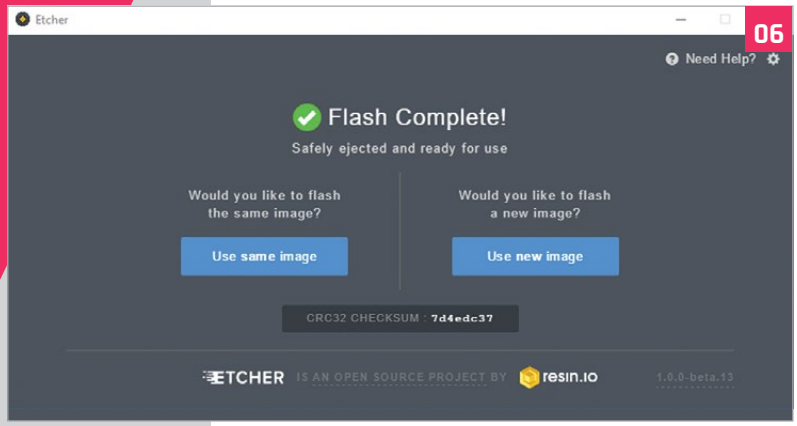
Etcher will format the SD card, before writing and verifying the image; this is shown by a progress bar. When done, remove the SD card, insert it into your Raspberry Pi, and power it up. If you want to flash another SD card with the same image, insert it and click **Use Same Image**.











MASTER THE RASPBERRY PI CONFIGURATION TOOL

Learn your way around the configuration tool found in Raspbian

You'll Need

- Raspberry Pi
- Raspbian Jessie with PIXEL

One of the best features in Raspbian these days is the desktop Raspberry Pi Configuration tool.

Located inside the **Preferences** option in the desktop **Menu**, this enables you to configure the hardware and software settings of your Raspberry Pi.

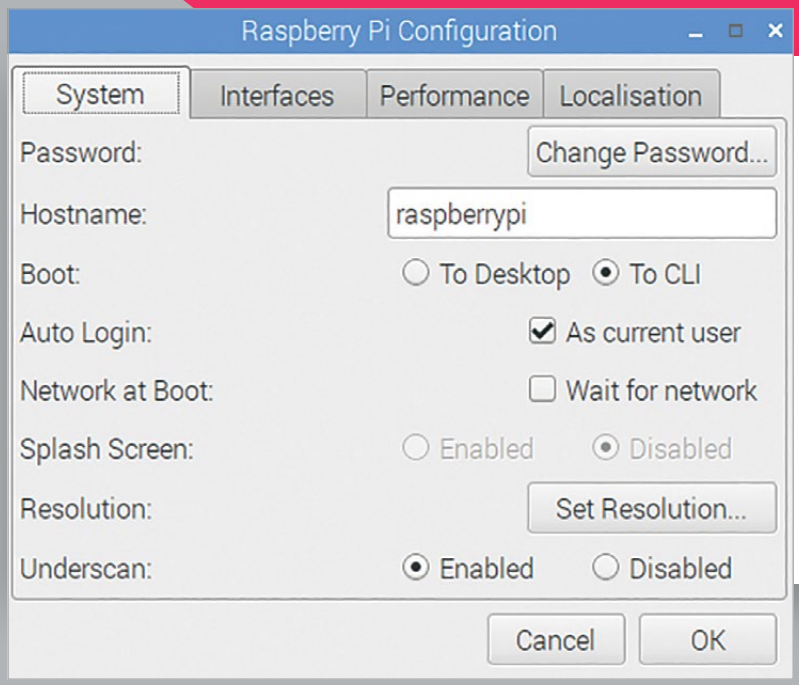
The Raspberry Pi Configuration tool works alongside the old raspi-config tool, which can still be accessed through the terminal using `sudo raspi-config`.

However, the new Configuration tool uses a graphical interface, making it much easier for newcomers. It offers the same options, but with a neater interface. Adjustments made in one tool affect the other. As a result, you can use either tool, or both.

Presently, the Raspberry Pi Configuration tool displays four tabs: System, Interfaces, Performance, and Localisation.

System is where you'll find the most useful tools. In this area you can expand the file system, change the password, and adjust login options. **Interfaces** contains options for activating hardware and software features. **Performance** is used to access overclock modes, and change the amount of RAM allocated to the GPU. The final tab, **Localisation**, enables you to adjust the locale, time zone, keyboard, and WiFi country for your Raspberry Pi.

There's a bunch of powerful features in the Raspberry Pi Configuration tool. As a result, learning its options makes you a better Raspberry Pi owner.



System

Options to expand the file system and change password and hostname sit alongside various login choices.

Interfaces

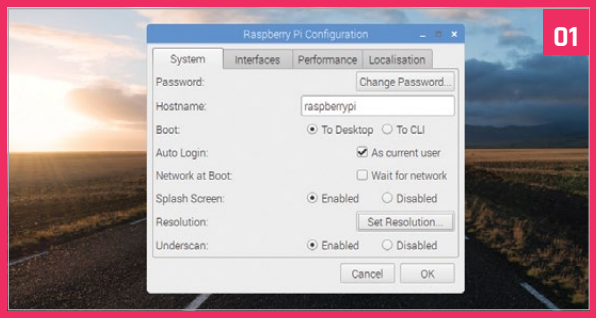
Support for the various hardware and software features, such as Camera Module, SSH, and VNC.

Performance

Overclocking and GPU memory options can improve the performance of a Raspberry Pi.

Localisation

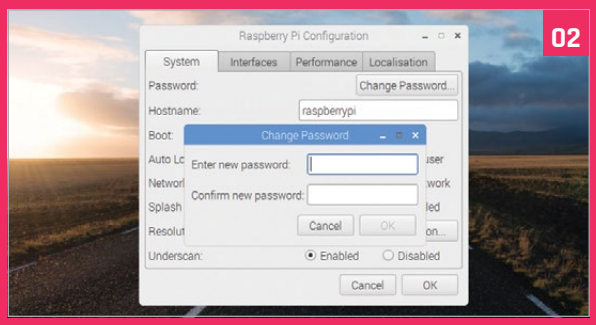
Set up an international keyboard, global WiFi options, and adjust the locale and time zone.



>OPTION-01

Open Raspberry Pi Configuration

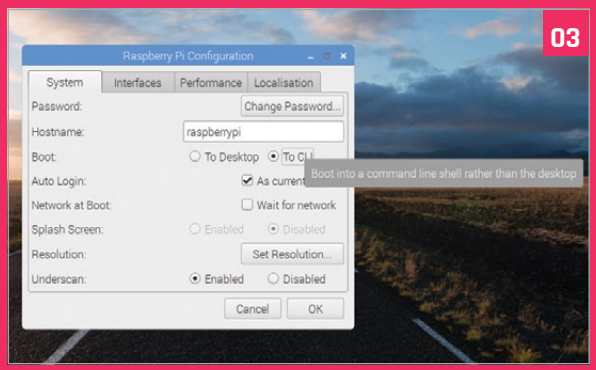
Open **Menu > Preferences > Raspberry Pi Configuration.** The Raspberry Pi Configuration tool has four tabs: System, Interfaces, Performance, and Localisation. Lets start with System, here you will find all the key settings to change the way your Raspberry Pi starts up and functions.



>OPTION-02

Hostname and password

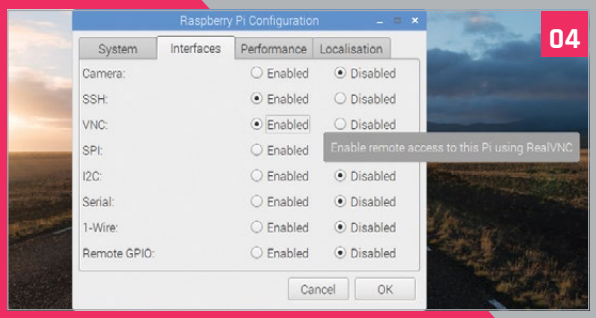
Updating your password is a good idea (especially if you connect it to a network). Click Change Password. Enter the same password into both fields and click OK. You can also personalise your hostname. Note that the hostname – used to identify the Pi on your network – isn't the same as your user name (which is 'pi').



>OPTION-03

Login options

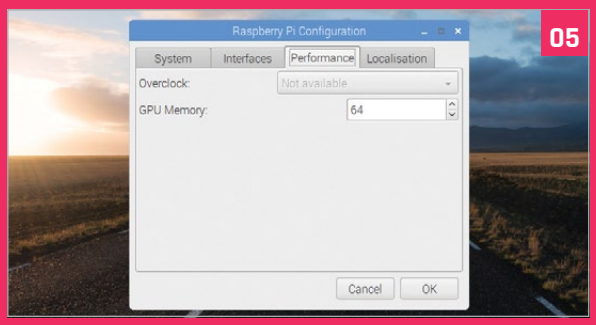
Below the Hostname setting sit various boot options. Choose **To CLI** to boot into the command line instead of the desktop. You can also opt to disable the splash screen and/or remove the auto login. You need to restart Raspbian for any of these to take effect. Click OK and Yes to reboot your Raspberry Pi.



>OPTION-04

Interfaces

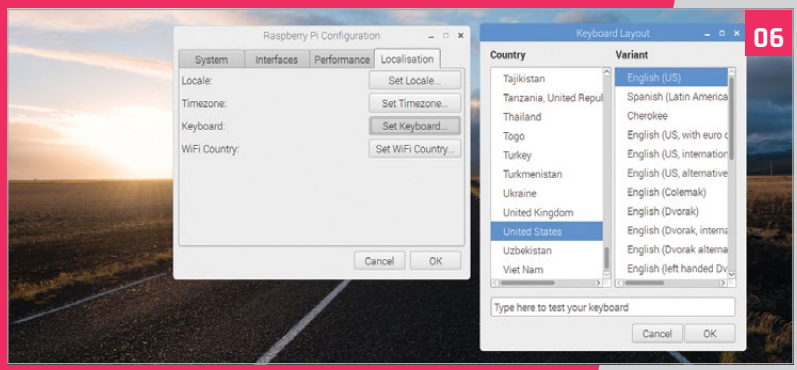
Reopen the Configuration tool and click on Interfaces to view the available options. Set Camera to Enabled if you plan on using the Raspberry Pi Camera Module. Now you'll be able to take images directly from the camera. Set VNC to Enabled if you plan on using VNC to remotely access your Raspberry Pi.



>OPTION-05

Performance

Click on Performance to view the two options here: Overclock and GPU Memory. Overclocking isn't available for the Raspberry Pi 3 yet, so this option will be greyed out. You can adjust the amount of RAM, in megabytes, allocated to the GPU (64 is the default, and is fine for most tasks). Leave it set as 64 on a Pi Zero W.



>OPTION-06

Localisation

Under the Localisation tab sit various international options. Click on Set Keyboard if you're using an international keyboard. If you own a US keyboard, click Set Keyboard. Now choose United States under country and English (US) as the Variant. Click OK and select Yes to reboot the Raspberry Pi.

HOW TO BUILD AN HTPC MEDIA CENTRE

Build your home theatre media centre with a Raspberry Pi and OSMC

You'll Need

- OSMC
- Pi Zero or Raspberry Pi 2/3
- Keyboard for setup
- Smartphone for remote control

Building a media centre, or HTPC (home theatre PC) is one of the most rewarding projects a newcomer can complete.

It's easy to turn a Raspberry Pi into an HTPC, and you can quickly hook it up to your television.

Using an HTPC, you can play video and music files, and stream video and audio from online services. And unlike proprietary systems, such as Apple TV and Google Chromecast, you can use it to play just about any media format.

You can add streaming services to your HTPC, such as BBC iPlayer and Soma radio stations. These enable you to play media directly from the internet.

There are several pieces of software available for setting up an HTPC. Two options you'll find on the Raspberry Pi Foundation's Downloads page are OSMC and LibreELEC. Both are similar and run a media player called Kodi.

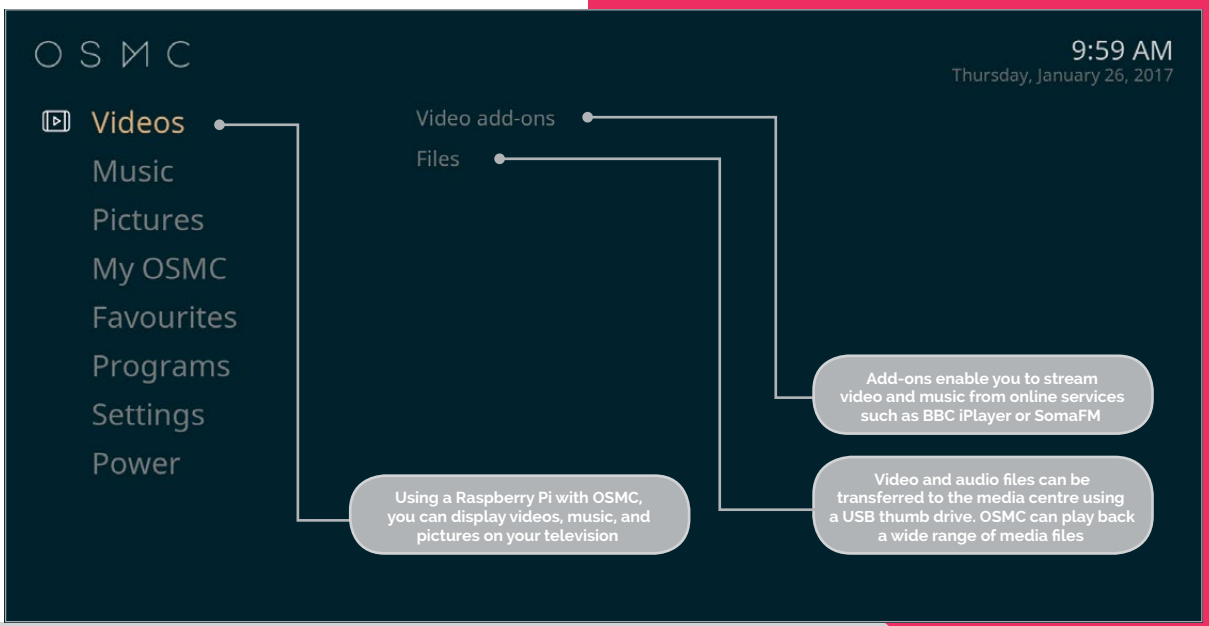
In this guide, we're going to opt for OSMC. It's a free and open-source media centre built on top of Debian, and is reliable and easy to use.

Setting up OSMC on a Raspberry Pi is a straightforward and rewarding project. Let's get started.

INSTALLING OSMC ON A RASPBERRY PI

>STEP-01
Installation

First, install the OSMC image to an SD card. Visit **magpi.cc/2jqbw8R** in your web browser and download the OSMC disk image for Raspberry Pi Zero or Raspberry Pi 2/3. Flash the image to an SD card using Etcher (**etcher.io**).

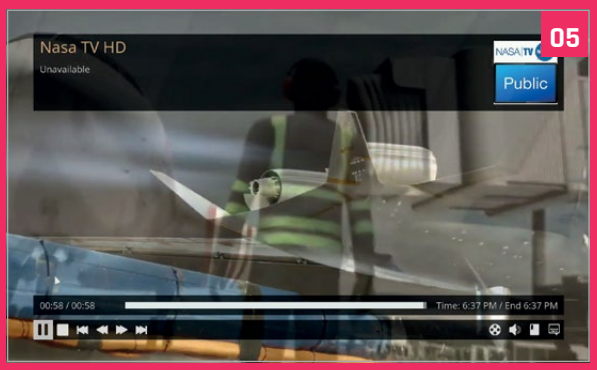


>STEP-02
Setup
Insert the SD card into your Raspberry Pi. Connect a keyboard and use an HDMI cable to connect a display. Power up and go through the Welcome section. It's a good idea to keep SSH enabled (the default option).

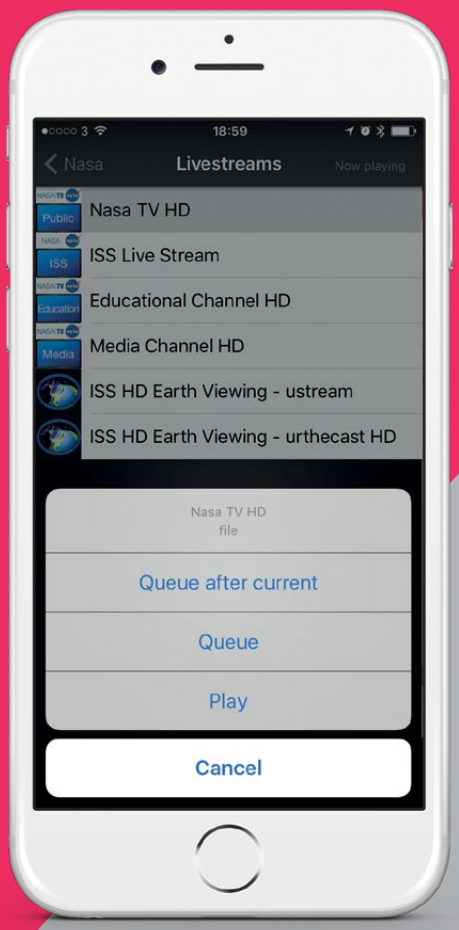
>STEP-03
Video files
To test video playback, we've downloaded a copyright-free video file of a movie called *Return of the Kung Fu Dragon* (**magpi.cc/2jxKryi**). Download the Ogg Video version and then transfer the file to a USB flash drive.

>STEP-04
Play video
Connect the flash drive to the Raspberry Pi. In the OSMC interface, select Videos > Files and the flash drive. Here you'll see the Ogg video file you copied across. Select it, and the video will start playing.

>STEP-05
Streaming services
Select Videos > Video Add-ons > Get More. Scroll down the services to find Nasa and select Install. Press **ESC** to get back to the main screen and select Videos > Nasa. Choose Livestreams and NASA TV to watch the station. For keyboard controls, visit the Kodi wiki (**magpi.cc/2jqjKoC**).

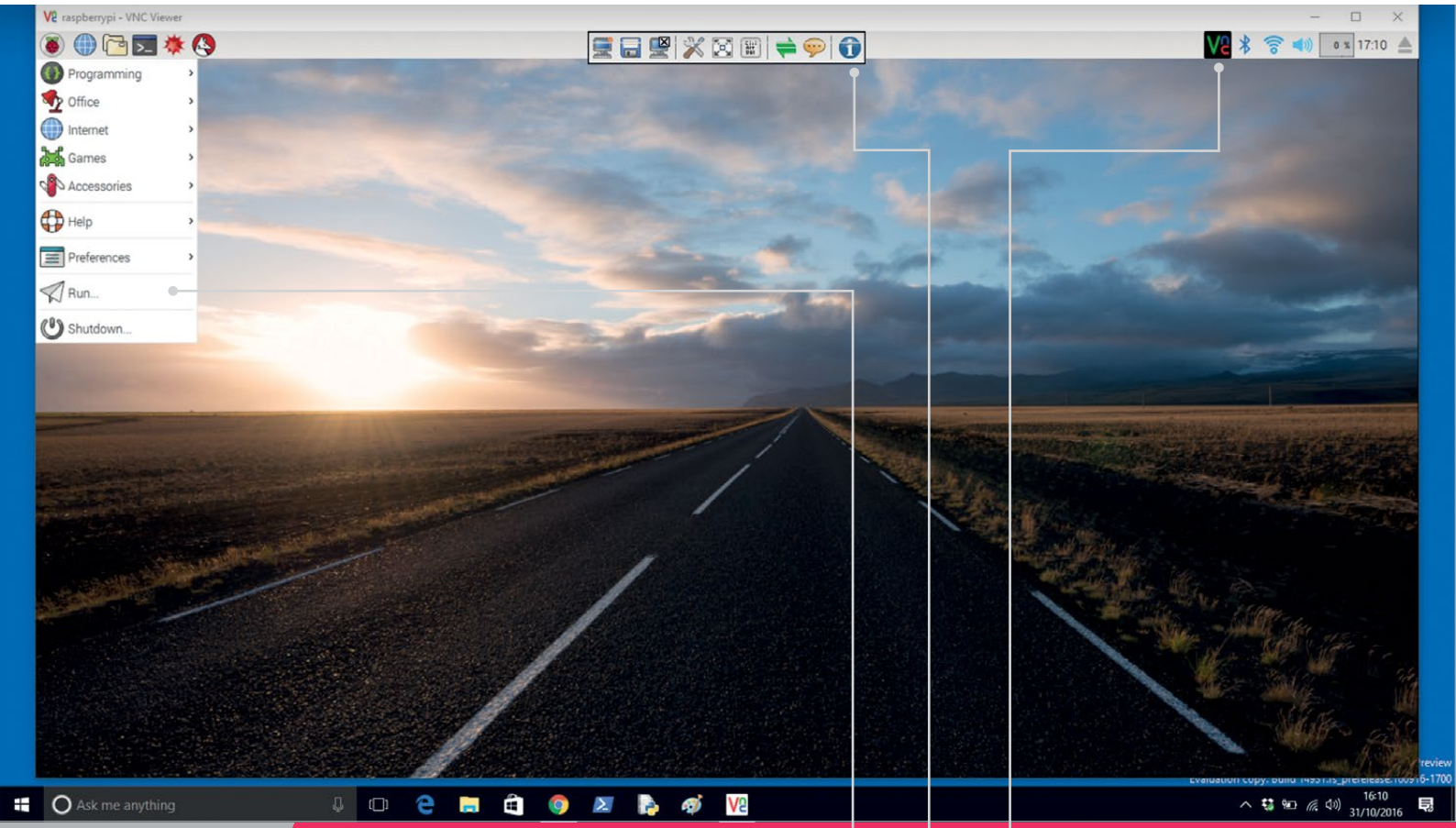


>STEP-06
Remote control
Open the app store on your phone and search for Official Kodi Remote (Kore on Android). Open the app, and click Add Host then Find Kodi. It should find the Raspberry Pi on your network. Click Save. Now use the app as a remote control.



BEGINNER'S GUIDE TO VNC

Remotely control your Pi from another computer with VNC Server and Viewer



You'll Need

- Raspberry Pi
- Raspbian
- VNC Viewer on your computer, smartphone or tablet device

VNC (Virtual Network Computing) is a great technology included with Raspbian.

With VNC, you can remotely control your Raspberry Pi from another computer, such as a PC or Mac, or even another Raspberry Pi board.

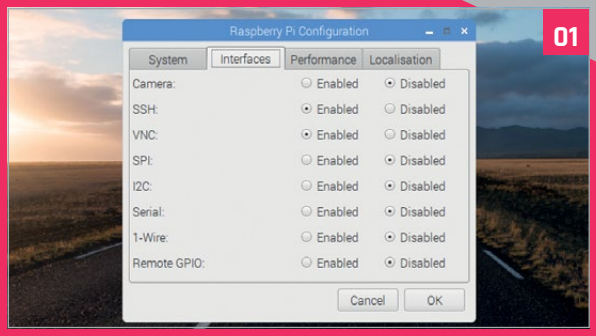
Sometimes it's not convenient to work directly on a Raspberry Pi. This can be because it's not easy to access, or because your keyboard and monitor are being used on your main computer.

With VNC, you can open the Raspbian desktop interface from your Raspberry Pi inside a window on your computer. You can even use VNC apps on your smartphone or tablet to control your Raspberry Pi.

By default, VNC Server from RealVNC gives you direct control over your Raspberry Pi, just as though you were sitting in front of it.

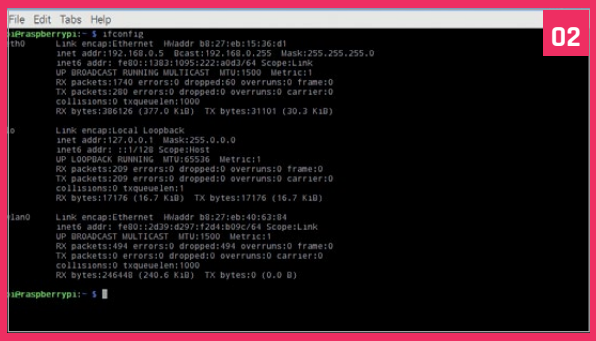
This is great for controlling lightweight Raspbian projects, such as IoT builds.

As we'll see in the steps, Raspbian includes VNC Server by default. However, you will need to enable it yourself. From then on, VNC Server will be loaded every time you switch on your Raspberry Pi.



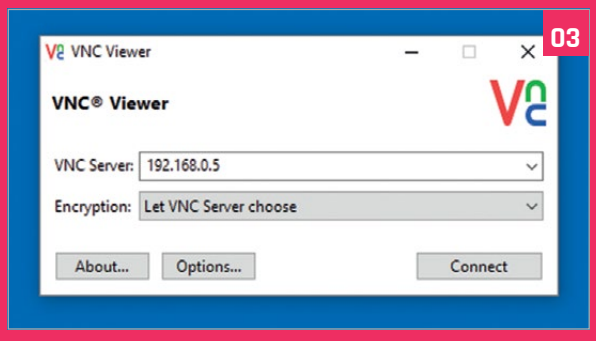
>STEP-01
Enable VNC

You need to enable VNC on your Raspberry Pi before you can use it. Choose **Menu > Preferences > Raspberry Pi Configuration**. Click on Interfaces and set the VNC option to Enabled. Click OK.



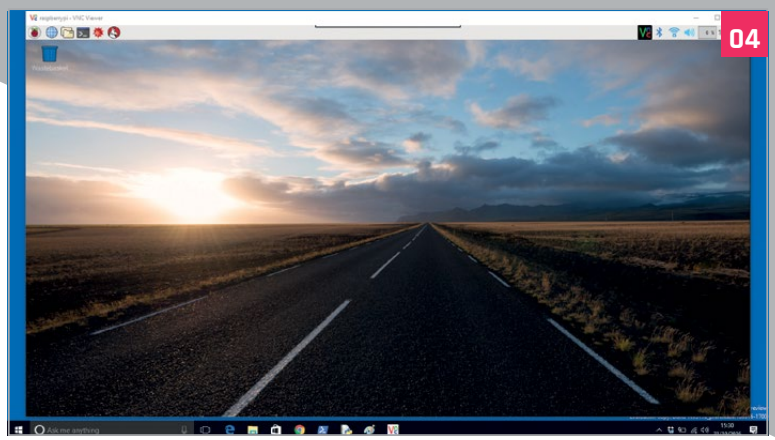
>STEP-02
Network

You'll need to know which IP address your Raspberry Pi is using to connect to it remotely. Open a Terminal window and enter **`ifconfig`**. Check for the four numbers next to **`inet addr`**. They'll be under **`eth0`** if you're connected using an Ethernet cable, or **`wlan0`** if you've connected to a wireless network. It'll look a bit like this: 192.168.0.5.



>STEP-03
Download VNC Viewer

Download and install VNC Viewer on your computer from RealVNC (**magpi.cc/1M4uzfG**). Open the app and enter the IP address from the previous step into the VNC Server field. Ensure Encryption is set to 'Let VNC Server choose', and click Connect then OK. The first time you connect, it will display a 'VNC Server not recognized' alert. Click Continue.

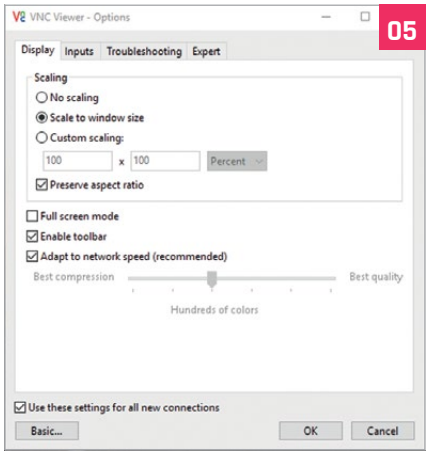


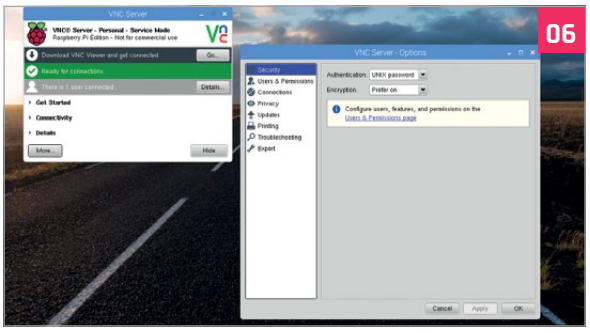
>STEP-04
Desktop window

Enter **pi** into the Username field, and **raspberry** (or your Raspbian password) in the Password field. Click OK. The Raspbian desktop will appear inside a window on your computer. You can click on the Menu, open programs, and run Terminal commands on your Raspberry Pi. If you have your Pi connected to a monitor, you'll see it move as you remotely control it.

>STEP-05
Toolbar settings

At the top of the virtual window is a small white strip. Hover the pointer over it to reveal a set of control icons known as the Toolbar. Full-Screen Mode is one of the most useful. Click Settings to the left to access options; the Advanced button near the bottom expands the list of settings.





>STEP-06
VNC Server

Another set of options is found inside the VNC Server app on your Raspberry Pi. Click on the VNC Server icon to view the server window. Click on Details to see which users are connected (normally this will be just one). Click **More > Option** to view detailed settings for the VNC Server software.

GET STARTED WITH THE PI CAMERA

Snap photos from your Raspberry Pi using its special, programmable camera

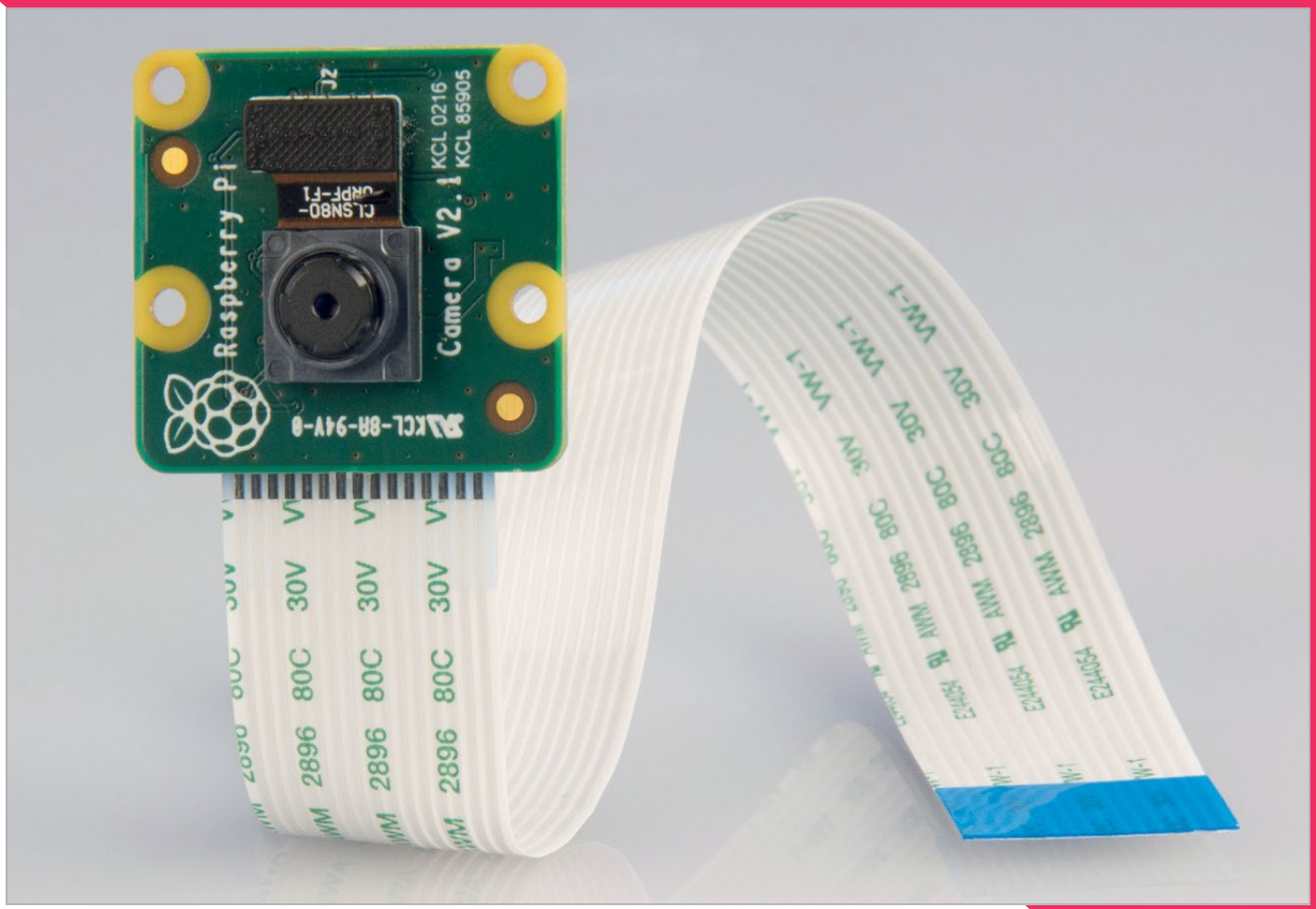
You'll Need

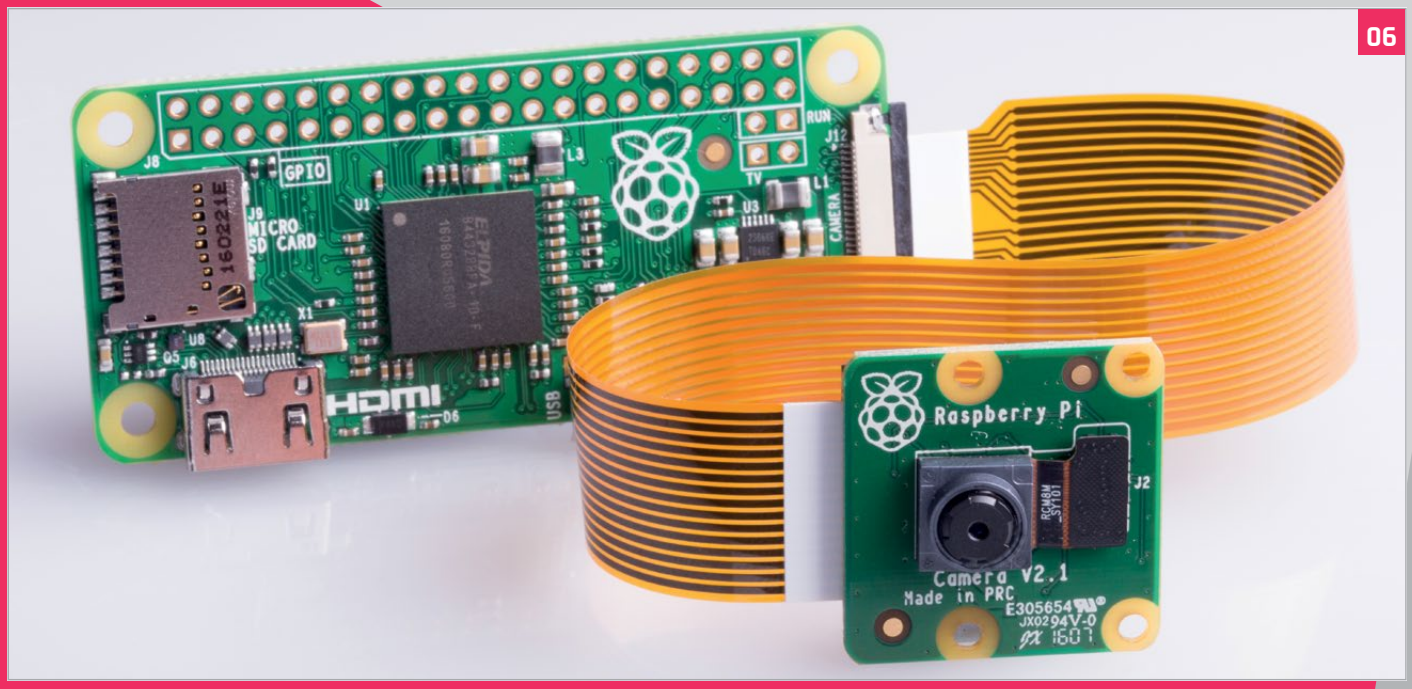
- Raspberry Pi
- Raspbian
- Pi Camera Module
  magpi.cc/28ljlsz

The Raspberry Pi has a few mysterious connectors on it that you wouldn't normally use when hooking it up. We've covered the GPIO pins in a previous issue, but now we're going to move onto the CSI port. You'll find this located between the HDMI and audio jack on a normal Pi, and on the edge of your Pi Zero W. CSI stands for Camera Serial Interface and, as the name suggests, it's used to connect a camera to the Raspberry Pi. Not just any camera either: specifically, the Raspberry Pi Camera Module.

The Camera Module, so called because it looks like a piece of circuit board and is attached via a ribbon cable, is a special programmable camera for the Raspberry Pi. It can take photos and video, and has many extra functions like time-lapse photography and slow-motion recording. It's fairly easy to control from the command line, or by using specific code in a Python script.

With the addition of a camera port to the Pi Zero W, every Raspberry Pi can use the camera for some cool and fun projects. Here's how to get started with it.

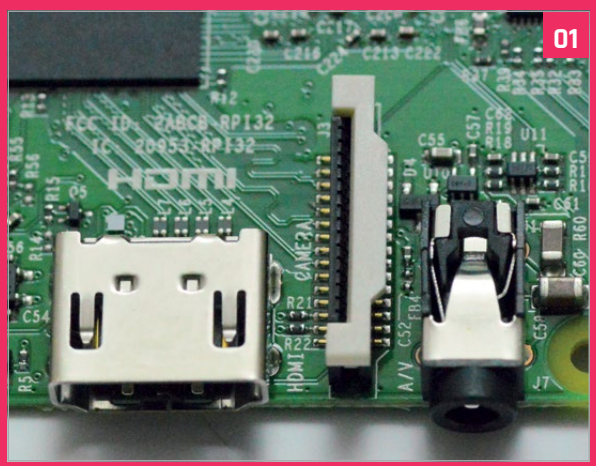


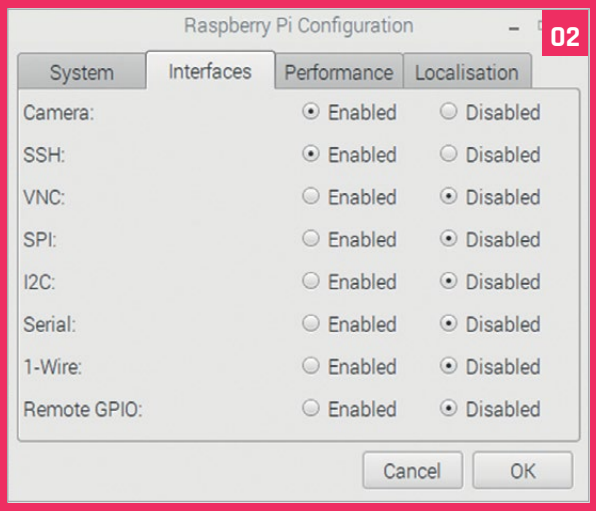


>STEP-01
Pi Zero W adaptor cable
The Pi Zero W has a small connector to attach the camera module. You will have an adapter cable included in your kit. Locate the adaptor cable supplied with your Pi Zero W.

>STEP-02
Attach the camera
With the Raspberry Pi turned off, gently lift up the plastic catch of the CSI connector. Take the end of the ribbon and insert it into the slot, with the silver connectors facing towards the HDMI port, before pushing the catch firmly back down.

>STEP-03
Enable the camera
Boot up the Raspberry Pi. Once you're in the desktop environment, head to **Menu > Preferences** and select Raspberry Pi Configuration. Go to the Interfaces tab and click the option to enable the camera.





>STEP-04
Taking a photo
You can take a photo from the terminal by typing `raspistill -o image.jpg`. This will show a preview for five seconds, then shoot an image at maximum resolution and save it as **image.jpg** by default.

>STEP-05
Recording video
Video is a little more complex: you need to tell it how long the video is in milliseconds. For ten seconds, use `raspivideo -t 10000 -o video.h264`. It'll preview what you're shooting, and the file will be at 720p.

>STEP-06
Python programming
Controlling the camera with Python is easy: all you need to do is import the `picamera` module at the beginning of your script. There's some info on how it works here: **magpi.cc/2gSZf9L**.

SWITCH TO THE COMMAND LINE

By using the command line, you are able to work faster and smarter. Discover how to get started today...

You'll Need

- Raspberry Pi
- Raspbian with PIXEL

Unless you grew up in the 1980s or earlier, the chances are that you are accustomed to using only GUIs (graphical user interfaces) and desktop environments.

There's really nothing wrong with GUIs, and Raspbian comes with a rather fine desktop interface.

But beneath the icons sits a whole other world: the command line. This is where your real computer is. With the command line, you're not locked into doing just what desktop applications enable you to do. You can do just about anything to your computer, and you can do it much faster.

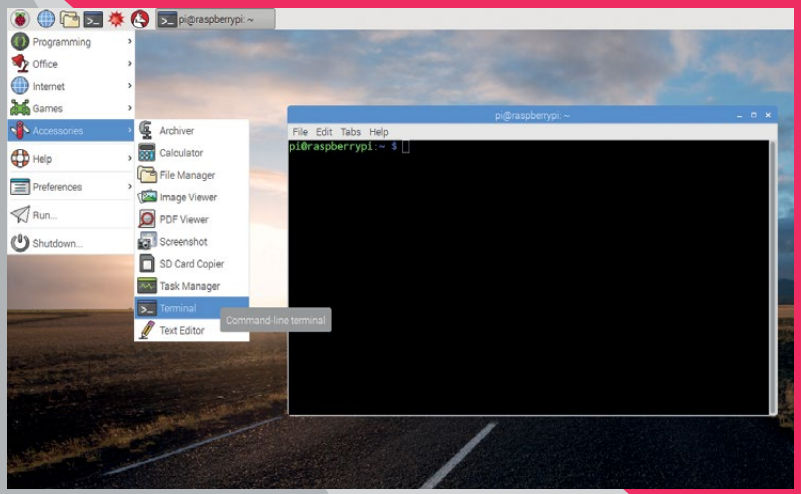
Think of it like driving a car. If you've only ever used a GUI then you're driving an automatic. The command line is like switching to manual. It's a lot trickier, but you get far more control and feel like a proper driver.

The command line can be daunting for newcomers, but it really needn't be. With just a few commands, you can master the command line.

Typing commands

When you boot a Raspberry Pi, you start by default inside the desktop interface.

The fastest way to get access to the command line is through the Terminal app.



Most people access the command line through the Terminal app in the desktop interface

Click on the Terminal icon in the top menu bar (or choose Menu > Accessories > Terminal). A window opens with a black background and some green and blue text. You will see the command prompt:

```
pi@raspberrypi:~ $
```

You are now at the command line. You enter commands using the text interface. Enter **`echo Hello World`** and press **RETURN**, you'll see 'Hello World' printed on the line. Below this is another **$** prompt, ready to accept another command.

Most users get to the command line via the Terminal app, but there is another way known as 'virtual console'. Press **CTRL+ALT+F1** and the desktop will vanish. A black screen appears, displaying 'Raspbian (or Debian) GNU/Linux 8 raspberry pi tty' and below it, 'raspberrypi login'. If you are not automatically logged in , enter **pi** and press **RETURN**, then enter your password (**raspberry** by default).

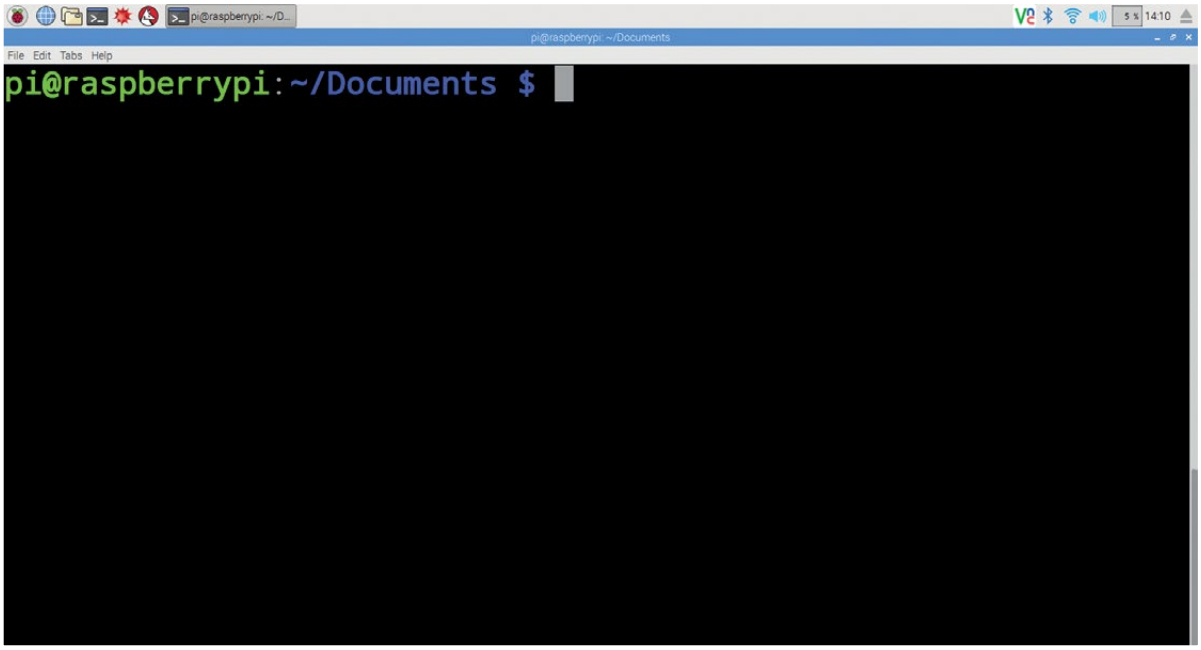
You can now use the command line in full-screen mode. You can get back to the PIXEL desktop using **CTRL+ALT+F7** and switch back to the virtual console using **CTRL+ALT+F1**. Additional virtual consoles can be accessed using **CTRL+ALT+F2** to **F6**. Each has its own login and operates independently.

If you prefer the command line, you can boot Raspbian directly to the command line instead of the desktop interface. Open Raspberry Pi Configuration (Menu > Preferences > Raspberry Pi Configuration). Change the Boot setting to 'To CLI' and click OK. Now when you reboot, you'll start in the command line (enter **`startx`** to start up the desktop).

Locate yourself

The first thing you need to learn is how to find out where you are. You are in your home folder by default. Enter the following command and press **RETURN**:

```
pwd
```



pi@
The first part of the command line is your user name followed by an @ symbol. You can see this on the command line by entering `whoami`.

raspberrypi
After the @ comes your host name. It is the name of your computer: 'raspberrypi' by default.

~/Documents
After the host name is your current working directory. This displays just '~' when you are in your home folder.

$
The dollar sign shows you're operating as a normal user .

This command is short for 'print working directory' and it tells you where you are. The command line will return `/home/pi`.

The home folder is the same one that appears by default when you open the File Manager app. You view the files and directories inside the working directory using the list (`ls`) command:

```
ls
```

Here you'll see the same directories (or folders) that are in the File Manager app: **Desktop**, **Downloads**, **Documents**, and so on.

The file path

Before going any further with directories, you need to understand the file path and the difference between a 'relative' and 'absolute' path.

Files are placed inside folders (which are called 'directories' in the command line). In the visual GUI, you can see these as folders that you open, revealing files and more folders. Your home folder contains a **Documents** folder, and inside that are three more folders: **Blue J Projects**, **Greenfoot Projects**, and **Scratch Projects**.

```
/home/pi/Documents/Scratch\ Projects
```

In the file path above, the first slash is the root of your hard drive. Here you have a directory called **home** that contains all users. In here is another directory called **pi** (that's you), and inside that is another directory called **Documents**, and inside that is one called **Scratch Projects**.

The eagle-eyed reader may have noticed the weird backslash character: '\'. You can't have spaces in file names, so you use a backslash followed by a space at

UNDERSTAND THE LANGUAGE

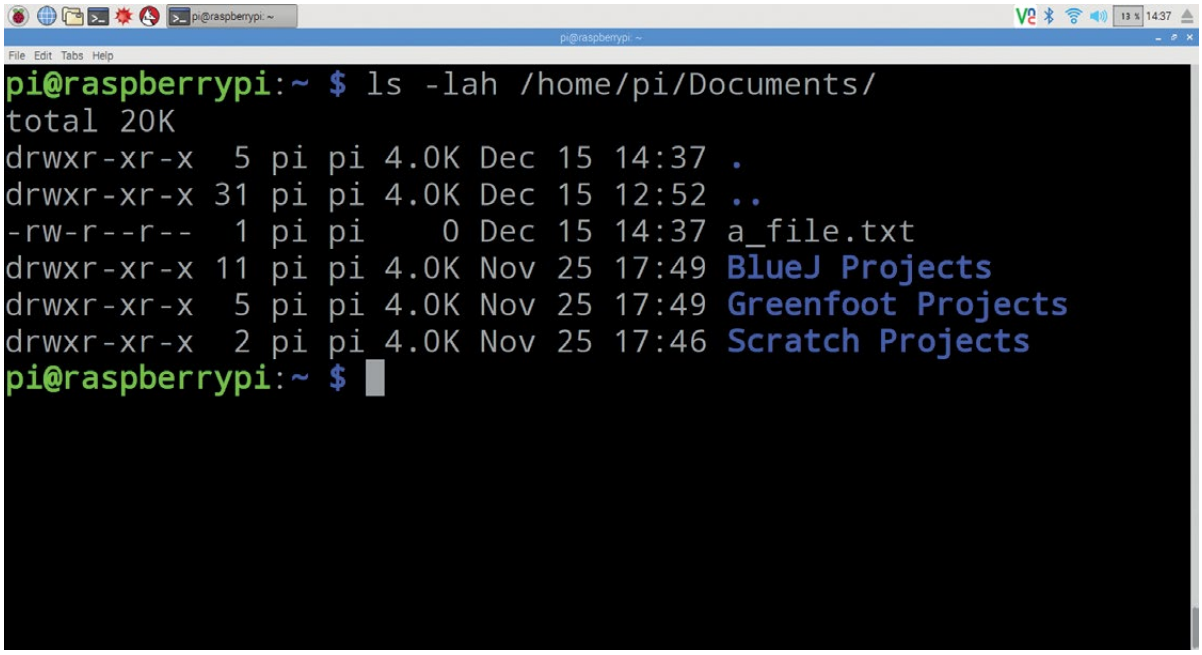
There's a lot of confusing jargon thrown around related to the command line. Terms like command line, shell, and terminal are often used interchangeably. Each has a precise meaning...

- **TERMINAL:** This is the program you use to access the command line from the PIXEL desktop in Raspbian (its full name is LXTerminal).
- **CONSOLE:** This is a physical terminal display with a keyboard. Consoles used to be empty computers that connected to a large mainframe computer.
- **VIRTUAL CONSOLE:** These are virtual versions of a physical console. In Linux, you have multiple virtual consoles accessed using **CTRL+ALT** and the function keys.
- **TTY:** Teletypewriter. In Linux, tty is used to display which virtual console you are using: tty1, tty2, and so on.
- **COMMAND LINE:** This is the text-based environment in general or the specific line you are working on. The command line starts with a dollar sign ($), known as the 'prompt'.
- **SHELL:** This is a command-line interpreter. It surrounds the computer's kernel (hence the name). To get to the kernel, you go through the shell. The shell interprets your text commands and turns them into code the kernel understands.
- **BASH:** This stands for 'Bourne Again Shell' and is the type of shell used in Debian (the version of Linux upon which Raspbian is based).

ls
The first part of a command is the command itself. Here we have ls, which lists the contents of a directory.

-lah
After the command come options. These start with a hyphen and are typically single letters. Each modifies the command. Here we have 'l', 'a', and 'h'. These stand for long listing mode, all files, and human-readable.

/home/pi/Documents
The final part of the command is the arguments. These are often file names or file paths. Here we are listing an absolute (direct) path to the Documents directory. If you omit the argument, it'll display the contents of the current directory.



the command line. Most of the time you'll also use the **TAB** button to quickly enter long file names (see 'Tab completion').

File paths come in two types: relative and absolute. Relative paths are 'relative' to your working directory, which is **/home/pi/** when you start. Entering **ls** alone shows the contents of the current directory. You can view the contents of a directory inside your working directory using **ls** and its name:

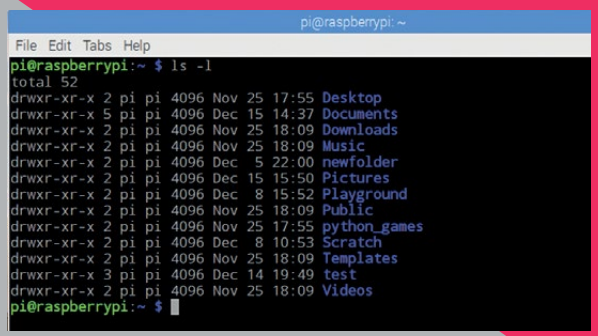
```
ls Documents
```

You can also view the contents of the directory above you using two dots (..):

```
ls ..
```

This displays files relative to where you currently are in the file system. If you moved into the **Downloads** folder and entered **ls Documents**, it'd cause an error, because there is no **Documents** directory inside the **Downloads** folder.

An absolute path, on the other hand, always starts with a slash '/', which is the root directory (the base of your hard drive).



The command line can be used to manage files and directories on your system

Enter:

```
ls /
```

...to view the root directory. Here you'll see all the directories and files that make up Linux. You'll see directories like **bin** (for binaries), **boot** (files used to start up the system), and **home**, which contains your user folder.

Enter:

```
ls /home/pi
```

...and you'll view the contents of your home folder, just as if you had entered **ls** from within it.

You can use absolute paths no matter what your working directory might be, because they always start from the root.

Moving around

Up until now we've stayed in the home folder and looked around using **ls**. You move from one directory and another using the **cd** (change directory) command:

```
cd Documents
```

Now enter:

```
pwd
```

...and you'll see a different working path: /home/pi/Documents. To move back up a directory (known as the 'parent' directory), you use two dots.

```
cd ..
```

Enter **pwd** again and you're back in the home folder. Now try it using an absolute path. Enter:

```
cd /
```

...and you'll be in the root directory. Enter **ls** to view the folders at the base of your hard drive. Enter:

```
cd /home/pi
```

...to move back to your home folder. There's a shortcut for this:

```
cd ~
```

The tilde (~) character is a shortcut for your home folder. You can use it at the start of an absolute directory too. For instance, entering:

```
cd ~/Downloads
```

...moves to your **Downloads** folder no matter where you are in the system.

Files

Throughout the file system, you'll find various types of files. A good selection is in the **python_games** folder, so enter:

```
cd ~/python_games
ls -l
```

The **-l** part (an option) makes it use 'long listing' mode, which displays items with lots of information:

```
-rw-rw-r--  1  pi  pi  973  Jan 27 2015
4row_arrow.png
```

From left to right, each item is:

- **Permissions:** The users and groups that can access a file.
- **Hard links:** The number of files that are linked to this file.
- **Owner:** The person who owns the file. Usually either **pi** or **root**.
- **Group:** The group the file belongs to.
- **File size:** The name of the file.
- **Modification:** When the file was last changed.
- **File name:** The name of the file.

The most obscure item is the list of letters and hyphens that make up the permissions. The first letter will be either a '-' or a 'd' depending on whether it's a file or a directory. Our **4row_arrow.png** is a file, so it's a '-'.

After that are nine letters arranged into three groups of three (see **Fig 1** overleaf):

- **Owner:** Typically this will be the person who created the account.
- **Group:** This is a group of users. You have only one group, **pi**, by default, containing just one user (also **pi**).
- **Other:** These are users from other systems.

Each of the three groups contains letters: 'rwx'. These letters are always in that order and each is either the letter or a hyphen. A letter indicates that the person, group, or other has access to read, write, or execute the file. A hyphen means they don't have that level of access.

Some examples include:

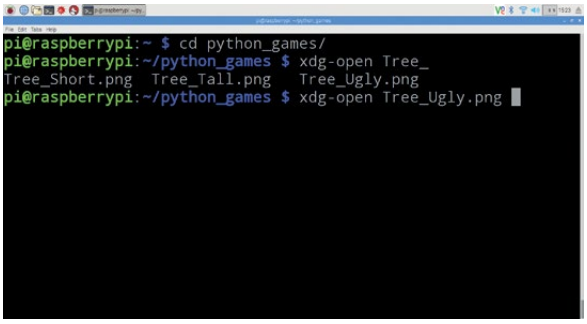
- **rwx** read, write, and execute
- **rw-** read, write, but don't execute
- **r-x** read and execute
- **r--** read only

Now that you've discovered how to move around the file system from the command line, it's time to learn what else you can do.

Take command

One of the first commands you need to learn is **mkdir**. This stands for 'make directory'. Move to the home folder and create a new directory called **test**:

```
cd ~
mkdir test
cd test
```

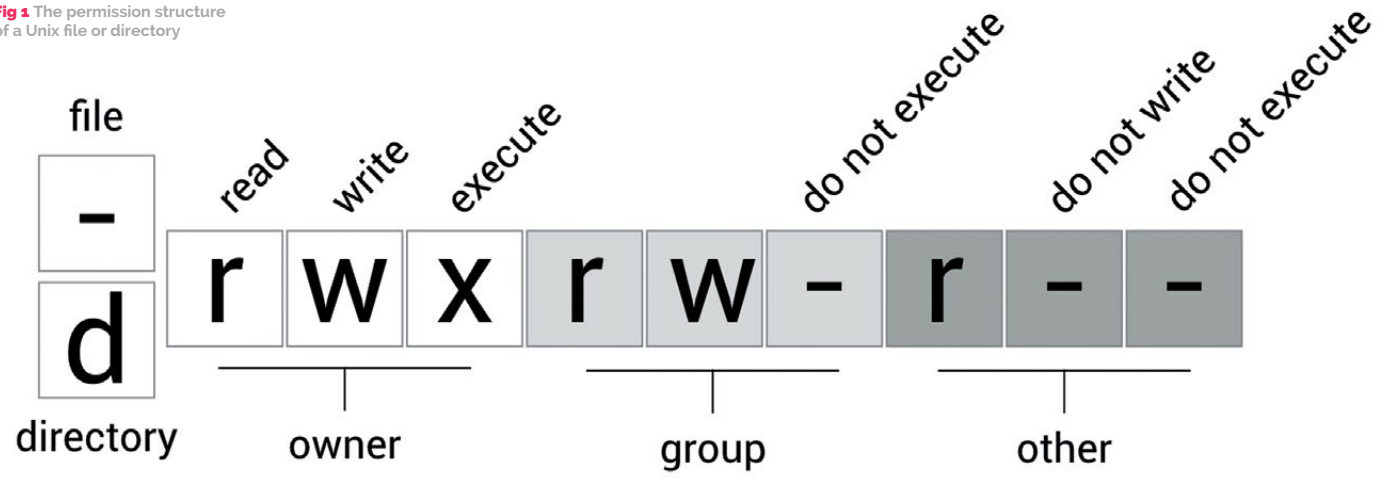


TAB COMPLETION

The single most useful tip you'll ever learn for the command line is tab completion. Pressing **TAB** at any time when entering a working path attempts to complete the file or directory name for you. Use **cd python_games** and enter **xdg-open Tr**, then press the **TAB** key. Notice how it fills it out to **'xdg-open Tree_'**. There are three files starting with Tree. Quickly press **TAB** twice and you'll see them: **Tree_Short.png**, **Tree_Tall.png** and **Tree_Ugly.png**. Enter **S**, **T**, or **U** and press **TAB** again to fill out the whole file name. Press **RETURN** to open it.

Tab completion can be invaluable for entering long file names packed with letters, numbers, and punctuation.

Fig 1 The permission structure of a Unix file or directory



To create files, you use a rather odd command called **touch**. Officially, touch is used to update the modification time of files (reach out and 'touch' them). If you touch a file, it updates the time next to it to the current time.

Few people use **touch** for that. A happy by-product of the command is that if you touch a file that doesn't exist, it creates an empty file. Enter:

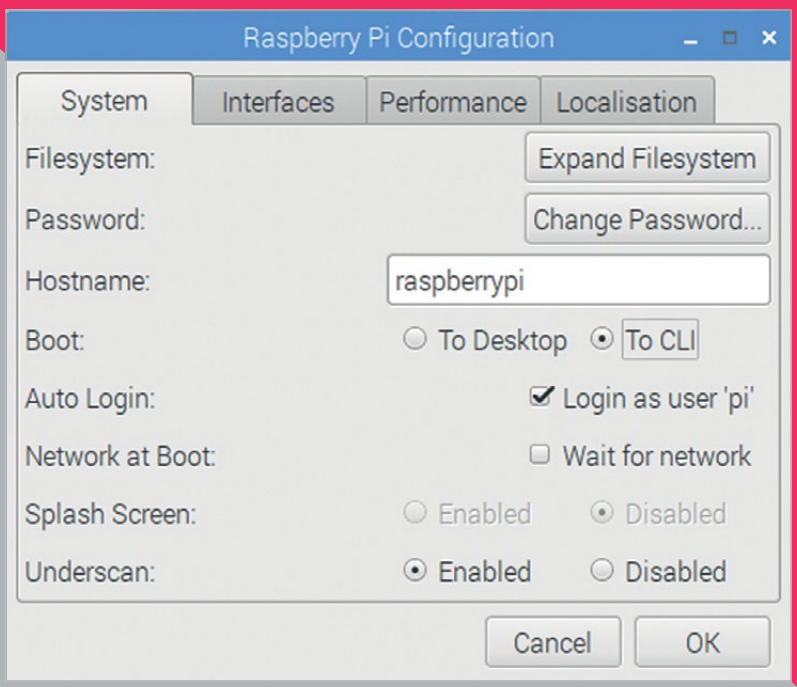
```
touch test.txt
```

You'll create a blank file called **test.txt**. Enter **ls -l** and you'll see the new file along with all its details. Notice that the file size is 0. This is because the file is completely empty.

We can edit the contents of the file using a text editor called nano:

```
nano test.txt
```

You can set Raspbian to boot into the command line (instead of the graphical interface) from the Raspberry Pi Configuration settings



You can enter and edit text in nano, but the Save and Exit commands predate the traditional **CTRL+S**, **CTRL+W** standards. Enter a single line, 'Hello World', and press **CTRL+O** followed by **ENTER** to save the file. Now press **CTRL+X** to exit.

Enter **ls -l** again; you'll notice that the file size has changed from 0 to 12. This is one for each letter (including space) and a newline marker at the end (you can see this character using **od -c test.txt** if you're curious).

Let's try deleting files. This command removes the file:

```
rm test.txt
```

Now move up to its parent directory and use another command, **rmdir**, to remove the empty **test** directory.

```
cd ..
rmdir test
```

Unfortunately, you'll rarely use **rmdir** to remove directories, because they typically have files inside them. You can see how this fails using these commands:

```
mkdir test
touch test/test_file.txt
rmdir test
```

It will say **rmdir: failed to remove 'test': Directory not empty**. The solution is to use **rm** with the **-R** option. This option stands for 'recursive' and means it goes inside each directory (and subdirectory) and removes every file and directory. Be careful when using **rm -R**, as it's permanent and deletes everything inside. Enter:

```
rm -R test
```

The **test** directory and everything inside will disappear. Unlike the desktop environment, there is no Wastebasket in the command line. When you remove files, they're instantly gone for good.

Options

Most commands have options that affect how they work. It's common to use these three options with the **ls** command:

```
ls -lah
```

Options start with a single hyphen '-' followed the letter for each option. The three options used here are:

- l = long listing format
- a = all including hidden files
- h = human-readable (makes large file sizes more readable)

Options are case-sensitive. So **ls -l** and **ls -L** are two different things (small 'l' is long listing format; large 'L' is dereference mode). Sometimes options are listed out in full. These start with two hyphens and have a single hyphen for spaces. This command is the same as **ls -lah**:

```
ls -l --all --human-readable
```

But it's more common to see (and use) the single letter approach.

Sudo

Sudo stands for 'substitute user do', although it's often also called 'superuser do'. If you have multiple users on your system, it can be used to perform commands as another user.

It's mostly used to get root access to your Linux installation. There is an account that controls your Pi user, called 'root'. This is an all-powerful account, which can change just about anything on your system.

Your default account can view files in the root of the hard drive, but it can't create or delete files at the root. Enter:

```
cd /
touch test.txt
```

You'll see **touch: cannot touch 'test.txt': Permission denied**. However, enter:

```
sudo touch test.txt
```

...and the **test.txt** file will be created on the root of your hard drive. You can see it using **ls -l**.

Now try to delete it:

```
rm test.txt
```

It will say **rm: remove write-protected regular empty file 'test.txt'?"** Enter **Y** and it'll say **rm: cannot remove 'test.txt': Permission denied**. You need to use sudo to remove the file:

```
sudo rm test.txt
```

You can see why sudo is such a powerful tool. Without it, you couldn't install software using apt or apt-get. But with it, you can remove or delete vital system files. Enter **ls /bin** and you'll see many programs (known as 'binaries') used by Linux. These include the **ls** command you just used. Accidentally deleting these files could make your system unstable.

So use sudo with care. In Raspbian you don't need to enter the password to use sudo. On many other Linux systems, however, you will be asked for the password before you can use sudo.

What's up, man?

There are lots of ways of getting help inside the command line. The first command you should turn to is **man**. This stands for 'manual' and gives you instructions on Linux commands and tools. Enter:

```
man ls
```

...and you'll see the manual for the list command. Notice under the SYNOPSIS it says:

```
ls [OPTION]... [FILE]...
```

This shows you the structure of the command. Almost all commands are in the 'command, option, argument' structure, although some arguments have more than one [FILE] argument (such as copy, which requires a source and destination file).

Press the space bar to move down the instructions. Here you will see a list of all the available options. With **man**, you can get detailed information on just about every tool on the command line. You can even get a manual for the **man** command with:

```
man man
```

If you need a quick reminder on how to use a command, try using it with **-h** or **--help** as an option:

```
touch --help
```

...tells you what options are available with the **touch** command. You can use this with many command-line tools to get a quick refresher on how they work.

Moving from a GUI to a command line is a vital skill for hackers and coders. Everything on your computer, from programs to preferences, is stored in the file system somewhere. Learning to use the command line makes you a more capable Raspberry Pi user.

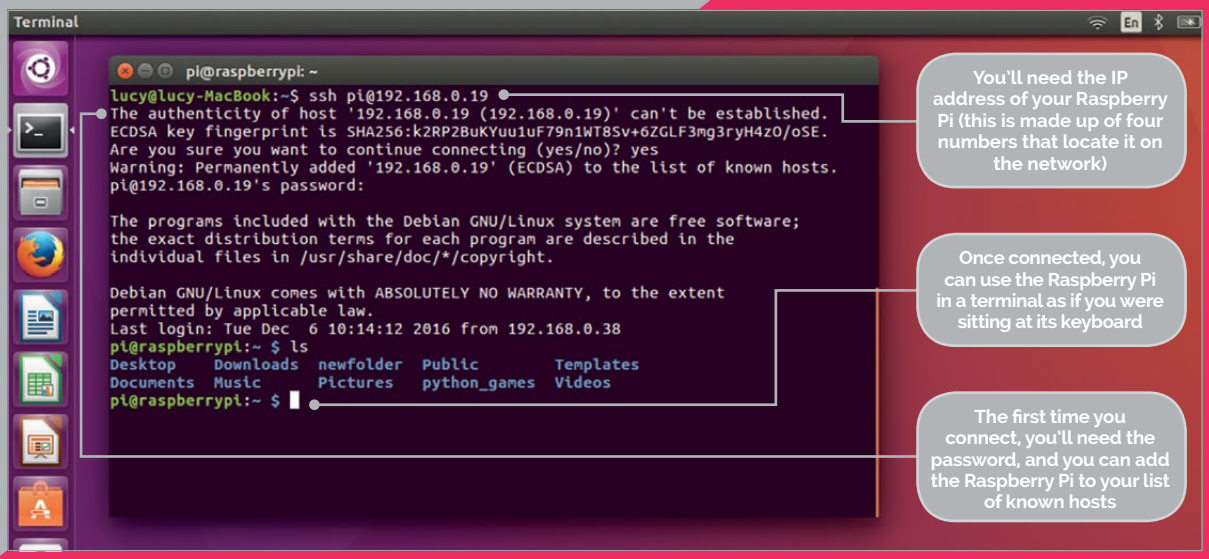
So, the next time you make a file, move a file, or delete something, don't head to the File Manager. Open Terminal and perform tasks from the command line. Soon it'll be second nature.

BEGINNER'S GUIDE TO SSH

Connect to your Raspberry Pi remotely through a terminal using Secure Shell

You'll Need

- Raspberry Pi
- Raspbian with PIXEL
- SSH client



Secure Shell (better known as 'SSH') is an encrypted networking technology that enables you to manage computers from the command line over a network.

SSH is handy if you want to quickly connect to your Raspberry Pi from a terminal on another computer. It's also ideal for lightweight distro installations that don't have an interface. It's especially useful when creating Internet of Things (IoT) projects, as these may be embedded and not require a desktop.

We've already looked at VNC (Virtual Network Computing), and SSH offers a similar service. But while VNC shares the entire desktop, SSH works from the command line.

On Linux PCs and Macs, you don't need to install any software to start using SSH. Linux and Mac OS X have the SSH command-line application installed by default; you can view its manual in the terminal using **man VNC**.

On Windows you will need to download an SSH client; the most commonly used one is called PuTTY. Download the PuTTY software from Simon Tatham's website (**magpi.cc/2hb1IiW**).

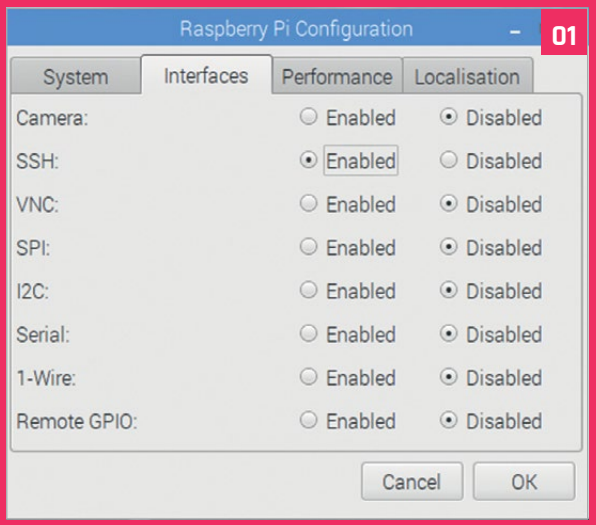
You'll need to use the password for your Raspberry Pi to log in using SSH. For security reasons, we recommend changing the default password.

SSH uses an encrypted network, so it doesn't send your password as plain text. More advanced users can control the encryption keys, using ssh-keygen. For now, we'll look at setting up and using SSH.

>STEP-01

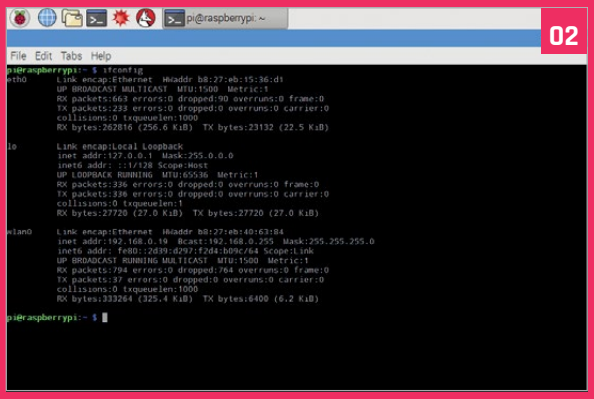
Activate SSH

You need to turn on SSH in Raspbian before you can use it to connect remotely to your Raspberry Pi. In the desktop interface, choose **Menu > Preferences > Raspberry Pi Configuration**. Click on Interfaces and set SSH to Enabled. Click OK.



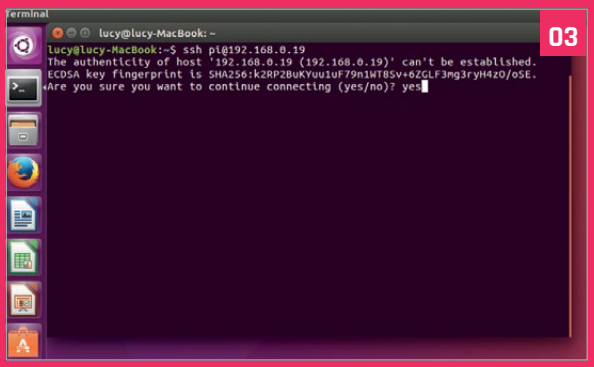
>STEP-02
Get the IP address

Connect your Raspberry Pi to a local network. Use a wireless network, or connect the Raspberry Pi directly to a router with an Ethernet cable. Open a terminal and enter **`ifconfig`** to find the IP address. With Ethernet, it'll be the four numbers next to **`inet addr:`**, such as 192.168.0.27. If you're connected wirelessly, look for similar numbers under **`wlan0`**.



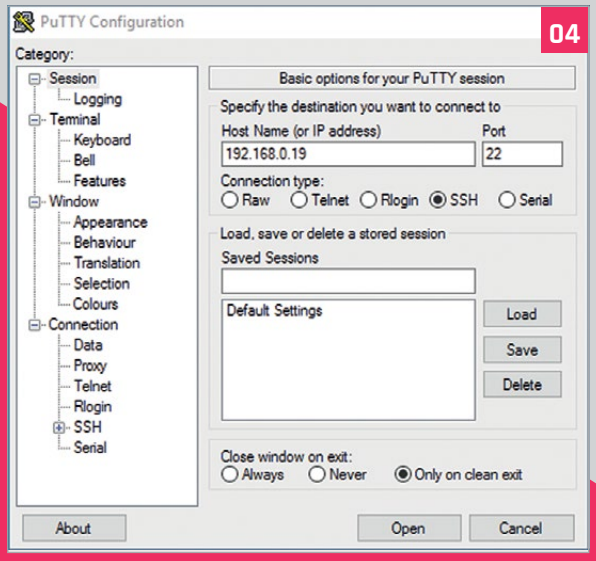
>STEP-03
SSH

On a Linux or Mac, open a terminal and enter **ssh pi@youripaddress**. On our network, that's **ssh pi@192.168.0.19**. The first time, you'll get this message: 'The authenticity of host (192.168.0.19') can't be established. ECDSA key fingerprint is SHA256:' followed by a long cryptographic hash of letters and numbers. It will say 'Are you sure you want to continue connecting?'. Enter **yes** and press **RETURN**.



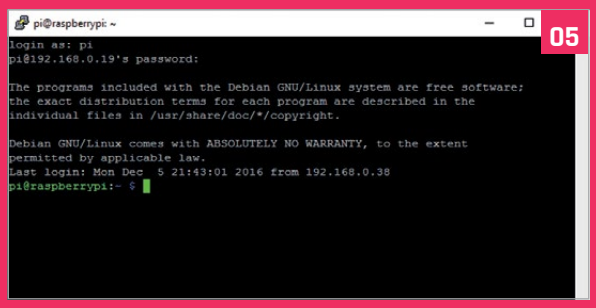
>STEP-04
PuTTY

On a PC you'll need to install PuTTY. Download the **putty.exe** file and click Run. The PuTTY Configuration window appears with basic options. Enter the IP address of your Raspberry Pi in the 'Host Name (Or IP Address)' field. Don't change the 'Port' field. Click Open. You will get a PuTTY 'Security Alert' field. Click Yes. The terminal window displays 'login as:' Enter **pi** and press **RETURN**. Now enter the password for your Raspberry Pi.



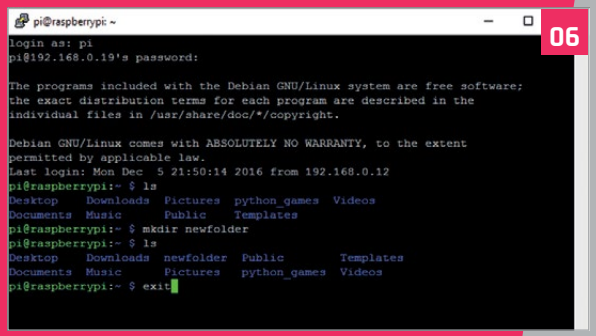
>STEP-05
The command line

You will now see your usual command line replaced with **`pi@raspberrypi: ~$`**. You are now logged in and working on the command line from your Raspberry Pi. Enter **`ls`** and you'll see **python_games** along with the other unique Raspberry Pi folders and files. You can create, edit, move, and work with files as if you were using a terminal on your Raspberry Pi.



>STEP-06
Exiting

There are limitations over VNC. You can't open programs with a graphical interface, so you'll need to use command-line alternatives (such as nano or vim instead of Leafpad for text editing). It's not as easy to share files using SSH as it is with VNC, but for fast command-line editing, it's hard to beat. Enter **`exit`** at the command line to finish.

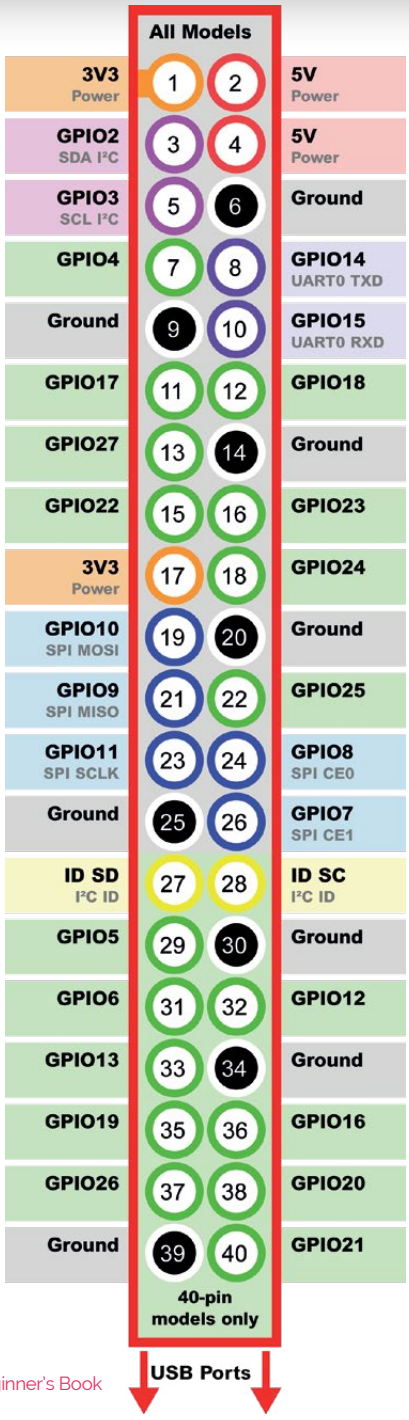


BEGINNER'S GUIDE TO GPIO ZERO

Discover the easy way to control GPIO pins on your Raspberry Pi

GPIO Zero Layout Guide

GPIO Zero uses the BCM number system to identify pins. These are the numbers referenced on the outside of the diagram, such as 18 for GPIO18. The board numbers (the ones on the inside) are just to help you count the pins. You can use any pin here marked GPIO, but the ones in blue double up with other functions. For this reason, we will stick to the ones marked in green in this tutorial.



You'll Need

- Rasbperry Pi
- Breadboard
- LEDs
- Button
- Jumper cables

The 40 metal pins on your Raspberry Pi board are known as General Purpose Input/Output pins, or GPIO pins for short.

These pins are your connection between the virtual world of computer code and the real world. With GPIO, you can connect circuit components up to your Raspberry Pi.

It's easy to attach LED lights, buttons, buzzers, and all kinds of electronic components to your Raspberry Pi. Typically you connect these components to a breadboard, at least when you are starting out. A breadboard is a plastic prototyping board used to link circuit components together without having to connect them physically. The legs of the components, along with cables, are pushed into adjoining holes on the breadboard.

You then hook the breadboard up to the GPIO pins on a Raspberry Pi (again, using jumper cables). You'll need male-to-male jumper cables to connect breadboard components together, and male-to-female jumper cables to connect the breadboard to the GPIO pins on your Raspberry Pi.

Different GPIO pins have different qualities. Some provide constant power, either at 5 volts or 3.3 volts. Others are ground pins, which should be used to complete a circuit.

Perhaps the most interesting GPIO pins are the ones that can be programmed. These can be turned on and off, powering up components (like LED lights) from code. Alternatively, they can be set to respond to voltage change inputs, such as a button push. Your code can then respond to these inputs.

Typically, you'll program GPIO pins using Python, although Scratch, Java, and a host of other languages can be used to control GPIO.

Programming GPIO in Python used to be quite a detailed task. But a new library, called GPIO Zero, simplifies things massively. With GPIO Zero you can quickly connect components and start using them with just a few simple commands.

>STEP-01
Set up a circuit

Programming GPIO pins is a relatively straightforward process, but remember that there are a few steps required to perform the most basic of tasks.

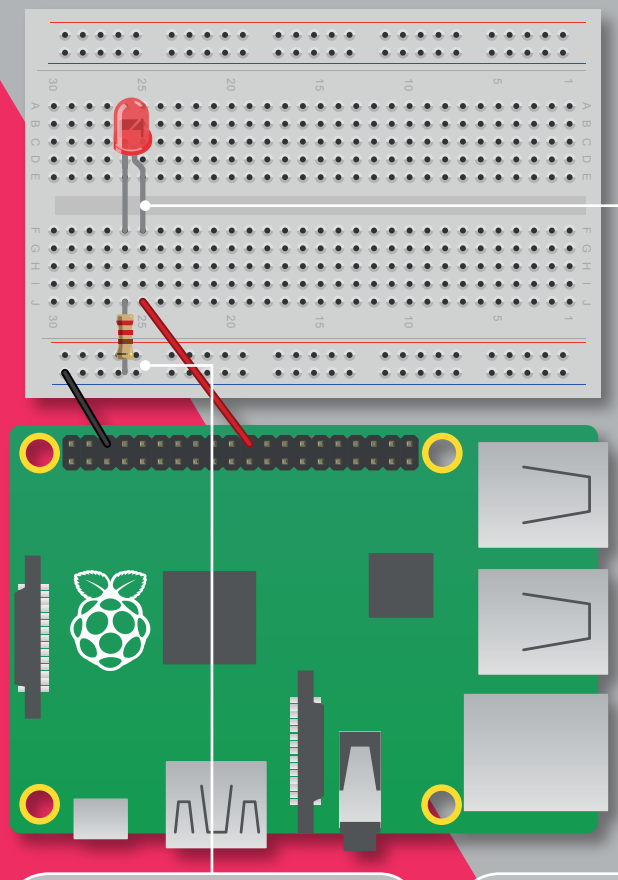
We're going to start by wiring up a single LED to GPIO25 as shown in this diagram. If you don't know how to use a breadboard, take a look back a couple of pages in this magazine. This circuit is extremely similar to the one we used there, except that instead of the long leg of the LED being connected to a live pin, it is connected to a programmable GPIO pin. This makes the LED light up when the GPIO pin is turned on in our program.

Use a female-to-male jumper lead to connect the GND pin to the ground rail on the breadboard.

Insert one end of a resistor into a hole on the ground rail, and the other end into a hole on the breadboard.

Connect the short leg of the LED into a hole on the same line and the longer pin on the line next to it.

Finally, take another male-to-female jumper wire. Place one end in a hole on the same row as the longer LED leg. Connect the other end to GPIO18.



A resistor is used to prevent the LED from burning itself out. The smaller the resistor, the brighter the light will be, but don't burn your LED out. A 330Ω resistor is a good start, though you can use other sizes. However, if you pick one that's too high, you may not be able to see the LED

The LED has two legs. The short one is connected to the resistor (and to the ground pin). The long leg is connected to GPIO25 on the Raspberry Pi

>STEP-02
Regular GPIO

If we were going to light up this LED using regular code rather than GPIO Zero, this is what we would have to write (don't enter this code: it's just an example):

```
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(25, GPIO.OUT)
GPIO.output(25,GPIO.HIGH)
```

It's not impossible to decipher. But this code is fussy, and concepts like 'OUT' and 'HIGH' get in the way of understanding the relationship between the code and the light. With GPIO Zero it's a lot easier.

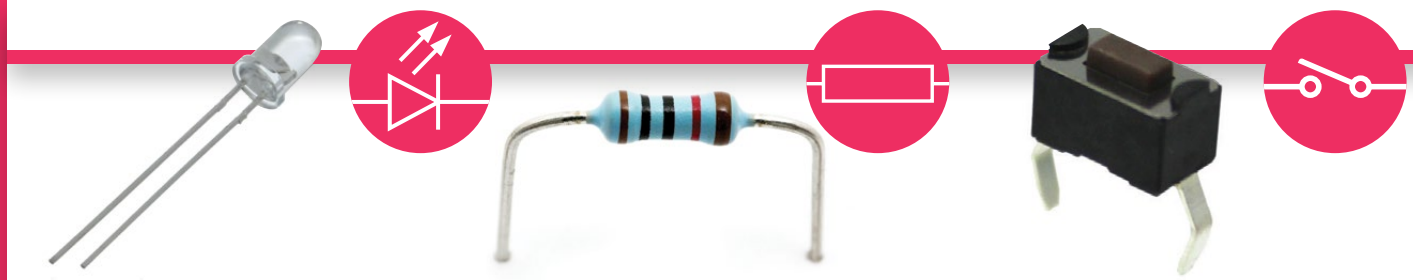
>STEP-03
GPIO Zero

GPIO Zero simplifies things. The same code in GPIO Zero looks like this:

```
from gpiozero import LED
led = LED(25)
led.on()
```

Enter each line of the code above into the Python shell one line at a time. Press **RETURN** after each line. When you enter `led.on()` the LED will light up.

As well as consisting of fewer lines, this program is easier for young students to understand.



LED
LEDs are easy to start with. LEDs have one leg longer than the other. They only work one way around.

RESISTOR
Always use a resistor with LED lights. This prevents them from burning out from high voltage.

BUTTON SWITCH
A button switch completes a circuit connection when it's pressed. Code can respond to button pushes.

The first line of the program is where we import the **gpiozero** library into Python.

The second line creates a LED object, with the pin number as the argument (inside the brackets).

The third line tells the pin to switch on.

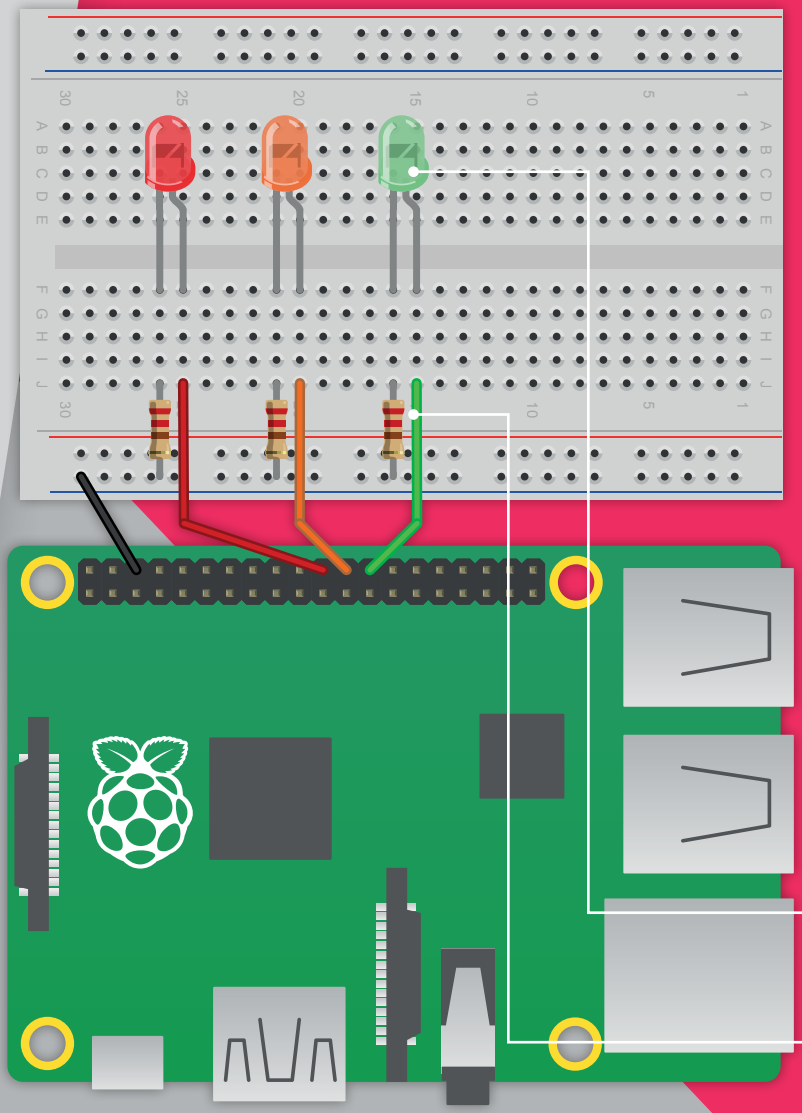
Now enter **led.off()**. The light will switch off.

>STEP-04
Blink

The real joy of GPIO Zero is that it lets you perform rather complex tasks with simple instructions. Creating a blinking LED with more standard code requires you to import the **time** module, set an LED to on, pause for a period, then set the LED to off, and repeat in a loop. This process makes it difficult to perform other code tasks at the same time. In GPIO Zero, however, you simply enter this line:

```
led.blink()
```

The light will start blinking on and off at one-second intervals. Enter **led.off()** to stop it.



The short leg of each LED is connected to a resistor. Each longer LED is connected a separate GPIO pin. These pins are used to turn on, or off, the power to each light

All three resistors are connected to the ground rail. This rail is connected to a single GND pin on the Raspberry Pi

>STEP-05
More control

One neat thing about GPIO Zero commands is that you can enter arguments inside the brackets. Enter **led.blink(** and stop at the open bracket. A yellow box appears showing the text '**on_time=1, off_time=1, n=none, background=True)**'.

These are the parameters available for the **blink** method. They are the number of seconds for which a light stays on and then stays off, how many times the light blinks, and whether you can add more code while the light is blinking.

After each argument is the default value: one second on, one second off, none (which means the light blinks until you say otherwise), and True (which lets you carry on adding code while the light is flashing). To set default values, add a number for each value (from left to right):

```
led.blink(4,2)
```

The light comes on for four seconds, and off for two seconds. Enter **led.off()** to stop it. Alternatively, you can add the item and equals sign to pick a value to change (and keep the defaults).

```
led.blink(n=3)
```

The light will blink three times and stop.

>STEP-06
Traffic lights

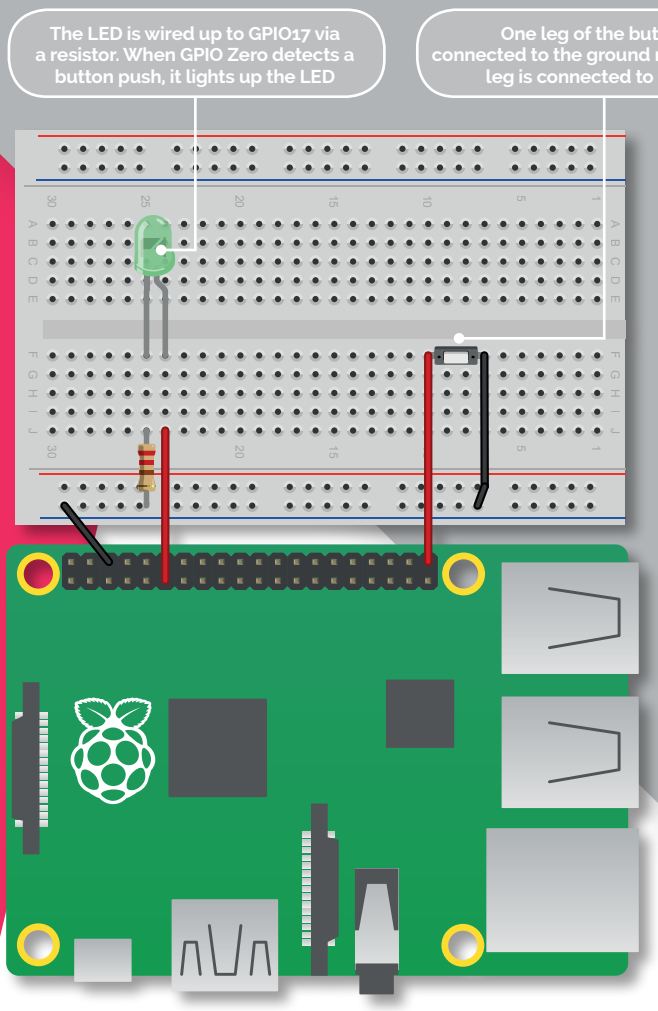
Let's take our LEDs and build something a little more complex. We're going to add another two LEDs to our circuit (three in total). We'll use one red LED, one amber, and one green.

Connect the new LEDs into the circuit using another two resistors to connect their shorter legs to the ground rail.

Connect the longer leg of the amber LED to GPIO8 and the longer leg of the green LED to GPIO7. These are the two pins next to GPIO25, so you have your LEDs all together.

Now we need to create a program to control our traffic lights. Open Python 3 (IDLE) and choose **File > New File**.

Type out the code from **traffic_lights.py** and choose **Run > Run Module** (or press **F5**). The Python program will run, and you'll see your traffic lights in action.



>STEP-07

Adding a button

Now we're going to add a button to our circuit and connect it to GPIO21 with an LED wired up to GPIO17. Buttons are wired up in a similar fashion to LEDs, using female-to-male jumper leads.

Like LEDs, buttons have legs. One leg is wired to a GND pin (via the same ground rail you've been using for LEDs); the other leg is connected to a GPIO pin. Some buttons have four legs, so you can connect them to the breadboard with their legs straddling the central groove (see the breadboard diagram).

>STEP-08

Button responder

Unlike LEDs, you don't need to use a resistor with a button. The button's legs are the same length, and it typically doesn't matter which way around you hook it up to the breadboard.

When the button is pushed, it forms a connection between the ground rail and the pin, completing a circuit. The Raspberry Pi is set up, using GPIO Zero, to detect this connection and respond.

Open Python shell and create a new file. Enter the code from **button.py**, save the file, and run the code. When you push the button, the LED will light up.

Language
>PYTHON

DOWNLOAD:
magpi.cc/2ehTqVq

button.py

```
from gpiozero import LED, Button
from signal import pause

led = LED(17)
button = Button(21)

button.when_pressed = led.on
button.when_released = led.off

pause()
```

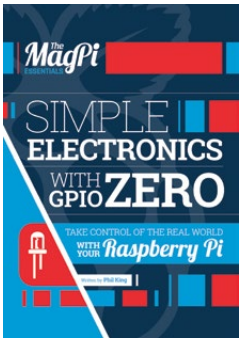
traffic_lights.py

```
from gpiozero import LED
from time import sleep

red = LED(25)
amber = LED(8)
green = LED(7)

green.on()
amber.off()
red.off()

while True:
    sleep(10)
    green.off()
    amber.on()
    sleep(1)
    amber.off()
    red.on()
    sleep(10)
    amber.on()
    sleep(1)
    green.on()
    amber.off()
    red.off()
```

Going further

You can achieve a lot more with GPIO Zero than controlling LEDs and buttons.

For a more detailed look at how you can detect motion, control robots, read sensor information (such as movement sensors or thermometers), *The MagPi*'s Phil King has created a fantastic guide: *MagPi Essentials: Simple Electronics with GPIO Zero*. Learn more at **magpi.cc/Back-issues**.



HOW TO USE A BREADBOARD

This humble plastic block full of holes can be used to create just about anything

You'll Need

- Breadboard
- LED light
- Resistor
- Male-to-female jumper leads
- Male-to-male jumper leads

Most of our projects are tested using a small piece of plastic known as a breadboard. Officially, it's known as a 'solderless breadboard' because it enables you to use circuit parts without soldering them together.

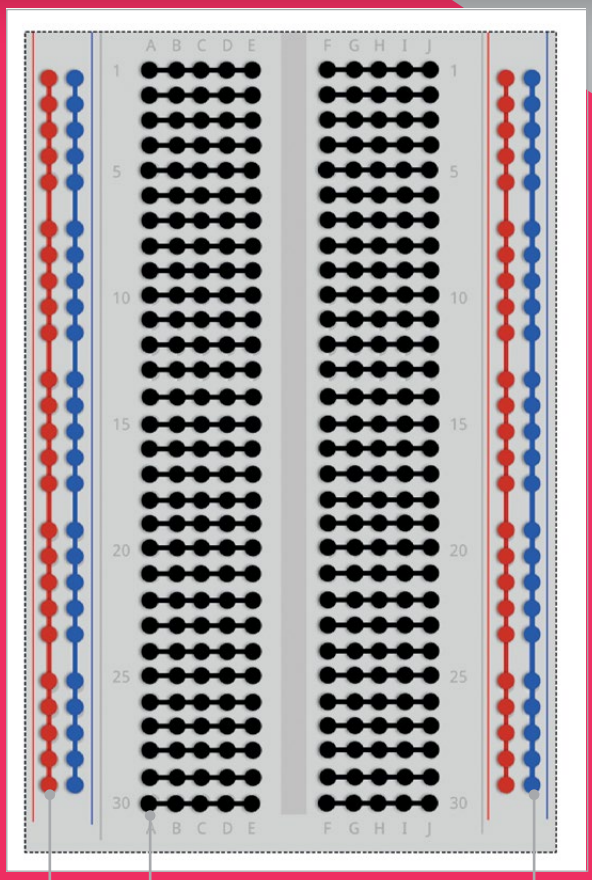
Electrical components are connected by pushing them into the holes in a breadboard. These holes are connected in strips, as shown in the main image. If you push a wire, or a different component, into one hole in a strip, and another wire into the hole next to it, it's as if you'd physically joined (or soldered) the two wires.

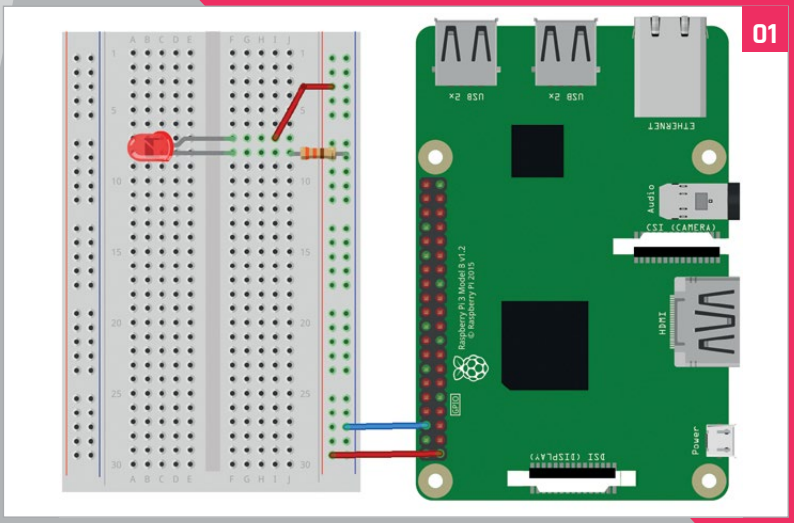
In the old days, people would either solder wire components together on an actual breadboard, or they'd wrap wires together around nails in a pinboard.

For a lot of Raspberry Pi fans, using a breadboard is part of life. But for many newcomers this quirky piece of kit is baffling: a smorgasbord of holes arranged in rows and columns that seem to make little sense.

So we think it's a great idea to read this beginner's guide to how a breadboard works. In this tutorial, we'll explain how these holes are arranged, and how to set up a circuit on your breadboard.

If you already know all this, feel free to move on. If not, stick around and learn about one of the most fun things you can do: building your own circuits and hooking hardware up to your Raspberry Pi.

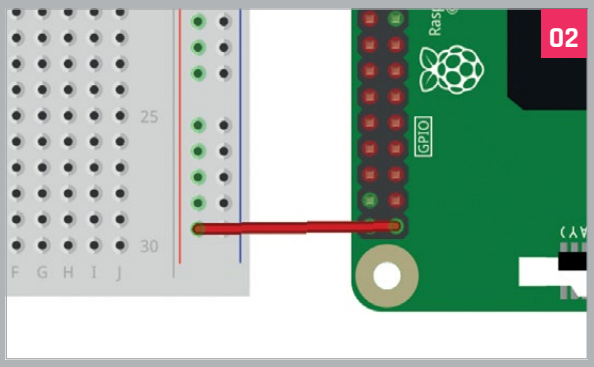




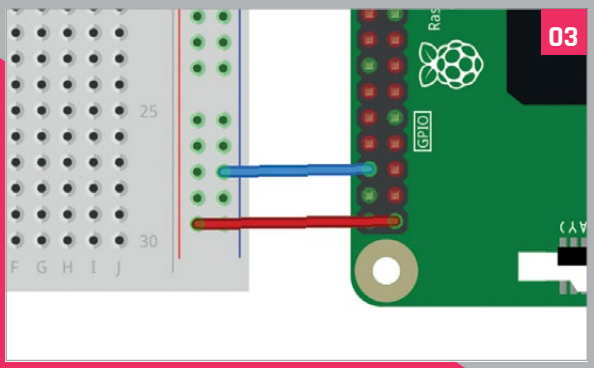
>STEP-01

Fritzing diagram

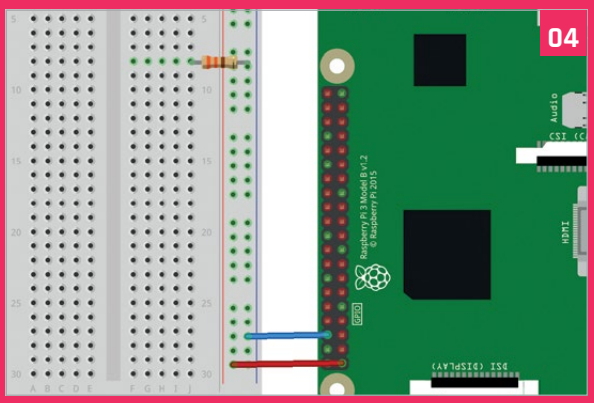
Circuit diagrams can be a little hard to understand for the novice. So we use visual breadboard diagrams, like this. This complete diagram uses the power and ground pins from a Raspberry Pi to light up an LED.



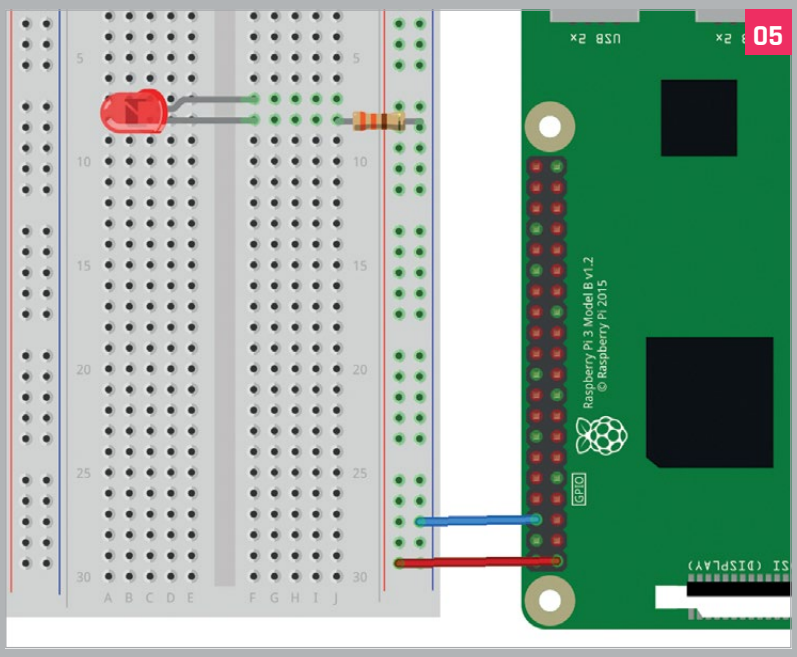
>STEP-02
Wire up the live rail
Take a female-to-male jumper lead (the colour of the wire doesn't matter) and connect the female end to a 5V pin on the Raspberry Pi. Place the male end of the lead into a hole on the red rail on the breadboard.



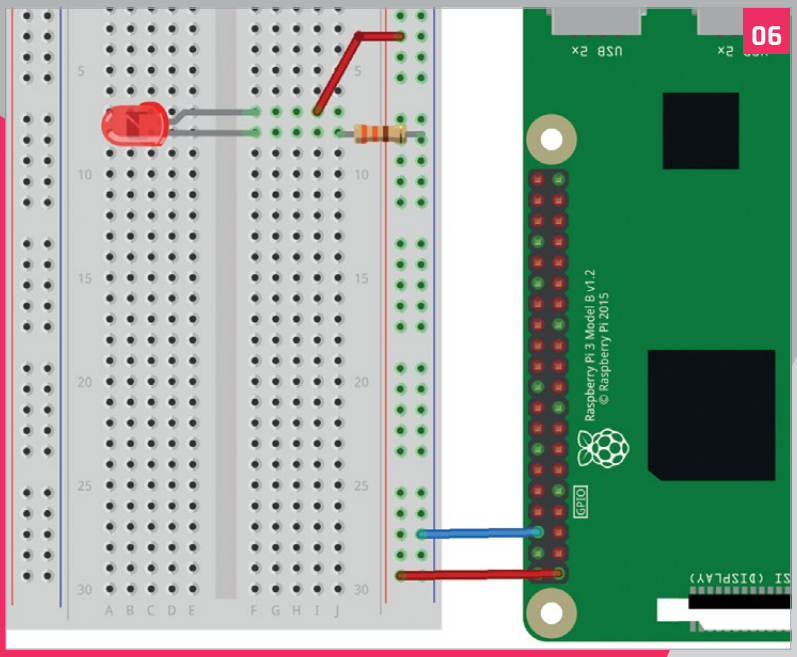
>STEP-03
The ground rail
Take another female-to-male jumper and connect the female end to a ground (GND) pin on the Raspberry Pi. The male end goes into a hole on the blue (ground) rail. All blue holes now act as a ground pin.



>STEP-04
Add a resistor
Take a resistor and connect one leg of it to a hole on the ground rail of the breadboard. It's now linked to the ground pin of the Raspberry Pi (via the jumper lead we used in the previous step). Take the other leg and connect it to a hole on the main breadboard.



>STEP-05
Add the LED
Take an LED component and look at the legs. Notice that one of the legs is shorter than the other. Place the shorter leg in a hole on the same row as the resistor. This leg is now connected to the resistor (which is linked to the ground rail, and therefore to the ground pin on the Raspberry Pi).



>STEP-06
Wire it up
Place the longer leg in a hole on the next row along. Now take another male-to-male jumper lead and place one end in the hole next to the long leg of the LED. Place the other end in a hole on the red live rail to complete the circuit. The LED lights up.

CODE IN PYTHON WITH THONNY

Use the new Thonny IDE to understand what's going on in your code

You'll Need

- Raspberry Pi
- Latest version of Raspbian
- Thonny

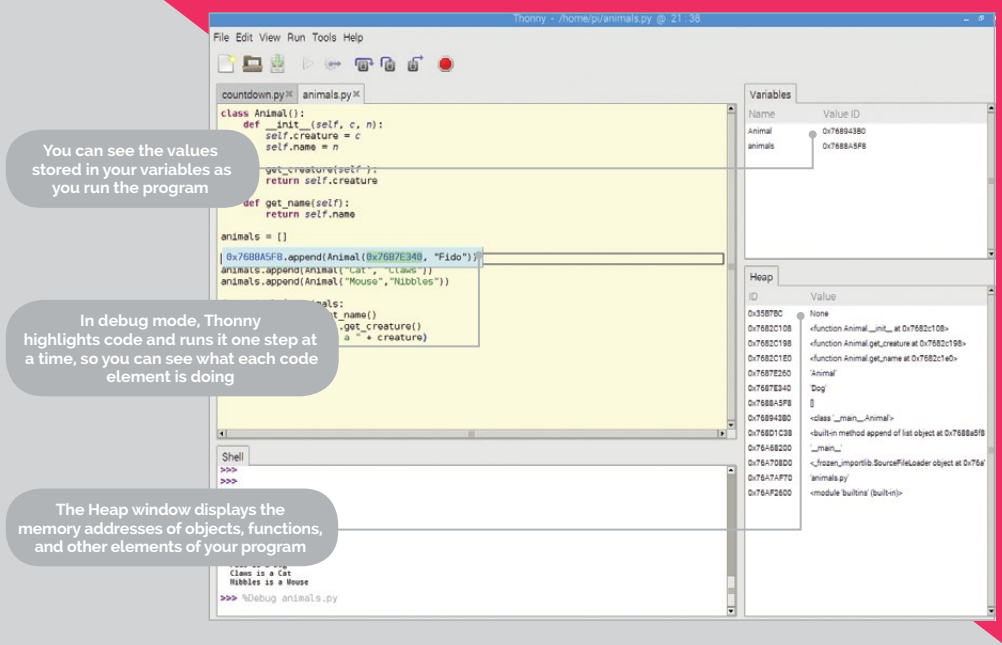
Thonny is a new IDE (integrated development environment) bundled with the latest version of the Raspbian operating system. Using Thonny, it's now much easier to learn to code. Thonny comes with Python 3.6 built in, so you don't need to install anything. Just open up the program, which you'll find under Menu > Programming. It offers a lot of advanced features not currently available in the Python 3 (IDLE) program, which is still included with Raspbian.

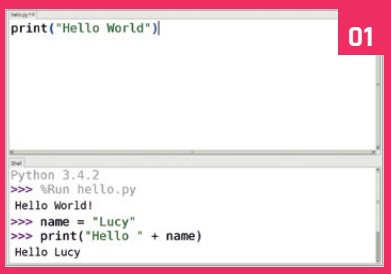
When you start Thonny, you'll see a new script editor and a shell. As with Python 2/3 IDLE, you enter a program in the script editor and run it in the shell. You can then use the shell to interact directly with the program; accessing variables, objects, and other program features.

Thonny has a range of additional features that are perfect for learning programming. One of the best features is a powerful, but easy-to-use, debug mode. Instead of running your program, it steps through the code line by line. You can see the variables and objects being created, and values being passed into functions or assessed by comparators.

You often find debuggers in powerful IDEs, but they tend to require you to manually set breakpoints (places where the program freezes so you can examine the code). The approach in Thonny is much more straightforward. It also has a range of panels that enable you to inspect various items, such as variables, objects, and the heap (the memory space where items are stored).

There's some pretty good stuff in Thonny for young coders. The ability to step through your programs makes it much easier to understand what happens when you hit Run.





>STEP-01

How to use Thonny

Click the Raspberry Pi Menu icon in the top left of the screen and choose Programming > Thonny Python IDE. We've used File > Increase Font Size so you can see the text more clearly. Enter this line of code in the script editor:

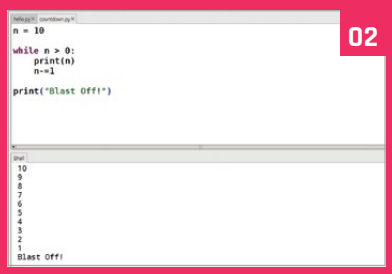
```
print("Hello World!")
```

Now choose File > Save and name the program **hello.py**. Click 'Run current script' (or press **F5**) to see the output in the shell. As with

DOWNLOAD:
magpi.cc/2t8CEhG

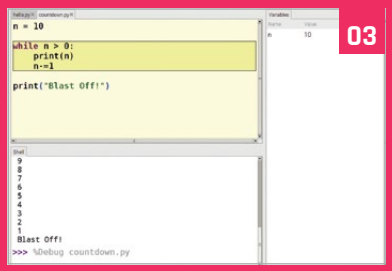
IDLE, you can also enter commands directly in the shell, such as:

```
name = "Lucy"
print("Hello " + name)
```



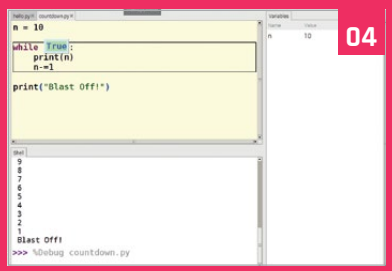
>STEP-02
Countdown

Let's see how you can walk through a file and see a variable change. Create a new script (File > New) and enter the code in **countdown.py**. Click Run and the code will display '10, 9 … 2, 1, Blast Off!' The **n** variable starts at 10. A **while** loop prints it, and decreases its value as long as it remains above zero.



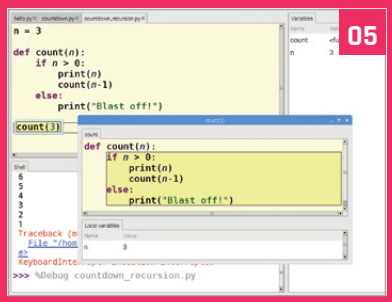
>STEP-03
Debug

Choose View > Variables and a new window appears displaying **n** and its current value (which is zero). Now let's run through it one step at a time. Click 'Debug current script'. The first line will be highlighted. Click Step Into and the value will be highlighted. Click it again, and both **n** and **10** are placed in the Variables window.



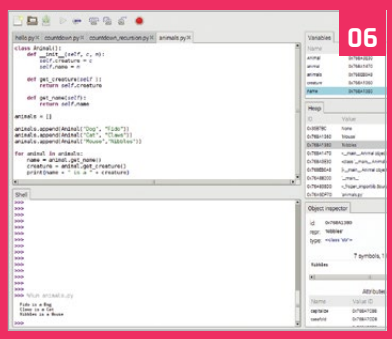
>STEP-04
Step through

Keep clicking Step Into and you will see the value of variable **n** (which is 10) added to the comparator and evaluated to **True**. Then the **while** loop will activate, the value of **n** will be displayed to the console, and **n** will decrease by 1. Click Step Out to run through the **while** loop and back to the main code.



>STEP-05
Recursion

Thonny's debug mode makes it easier to understand concepts such as recursion. Our **countdown_recursion.py** program runs a countdown recursively (a function which calls itself from inside itself). When the function calls itself, a new window appears with the function. Keep stepping through to see the values updated.



>STEP-06
Heap and objects

For a more detailed view, enter View > Heap and View > Objects. Now, as you work with object-oriented code, you can select objects in the Heap or Variables window and use the Object Inspector to check their type and attributes. The **animals.py** code creates animal objects with creature and name instance variables.

animals.py

```
class Animal():
  def __init__(self, c, n):
    self.creature = c
    self.name = n

  def get_creature(self ):
    return self.creature

  def get_name(self):
    return self.name

animals = []

animals.append(Animal("Dog", "Fido"))
animals.append(Animal("Cat", "Claws"))
animals.append(Animal("Mouse","Nibbles"))

for animal in animals:
  name = animal.get_name()
  creature = animal.get_creature()
  print(name + " is a " + creature)
```

countdown_recursion.py

```
n = 3

def count(n):
  if n > 0:
    print(n)
    count(n-1)
  else:
    print("Blast off!")

count(n)
```

countdown.py

```
n = 10

while n > 0:
  print(n)
  n-=1

print("Blast Off!")
```

SET UP A FILE SERVER

Turn your Raspberry Pi into a file server to back up and share content from anywhere on your local network

You'll Need

- A 32GB micro SD card
- Raspberry Pi 2/3
- Monitor, keyboard and mouse (for setup)
- Wired Ethernet connection
- NOOBS **magpi.cc/2bnf5XF**

It's easy to use a Pi as a simple file server where you can store backups and share files from all the other computers on your network. Samba is the Linux implementation of the SMB/CIFS file sharing standard used by Windows PCs and Apple computers, and widely supported by media streamers, games consoles and mobile apps.

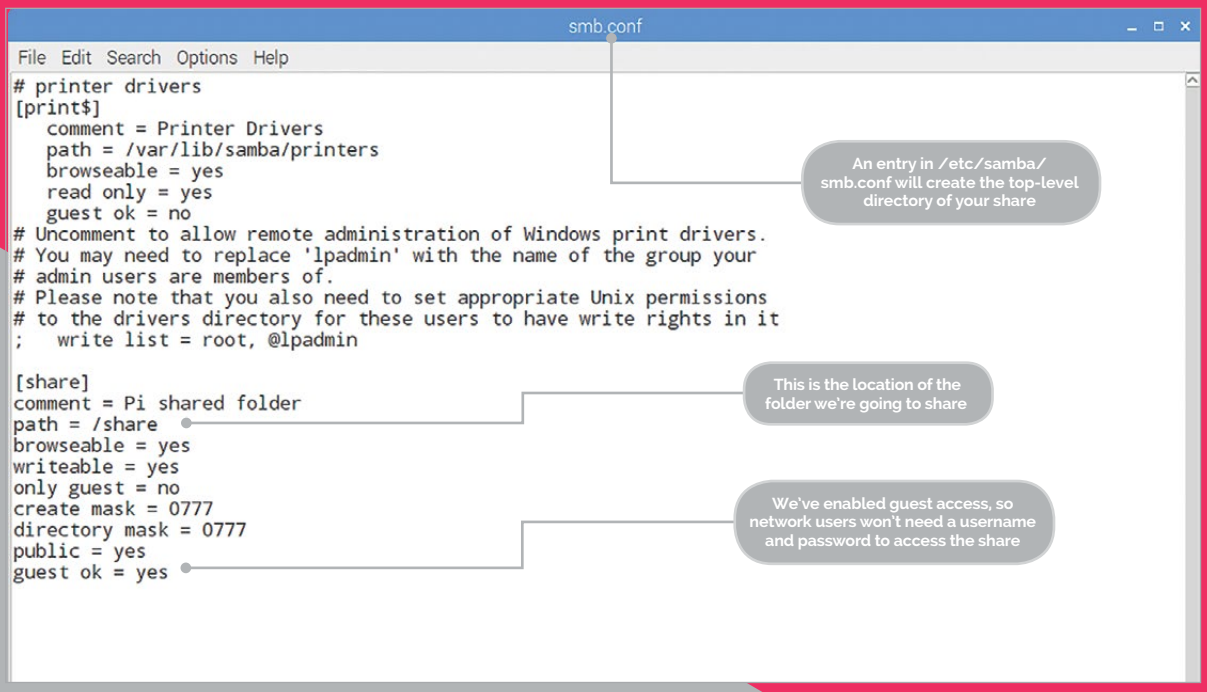
This tutorial assumes that you'll use a keyboard, mouse, and monitor to set up your file server, but you can alternatively enable SSH (**magpi.cc/1GULmTr**) and connect to it remotely from another computer on your local network.

We also assume you're using a 32GB (or smaller) micro SD card, which provides a reasonable amount of storage space without requiring any extra steps to make it accessible. However, if you need extra storage, it's easy to mount a large external USB drive and create a Samba entry for it.

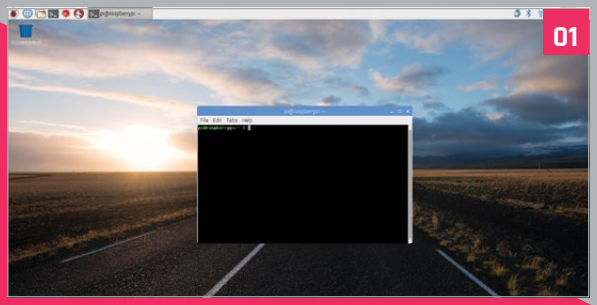
Alternatively, if you want to keep things compact, you can install Raspbian on micro SD cards of up to 256GB, although we suggest checking online (non-working SD cards: **magpi.cc/2q97aGO**) before you buy to make sure you get one that's fully compatible with the Raspberry Pi.

Once set up, you can mount your home file server on all the other computers on your network, and use it as a convenient place to store everything from music files you want to share with your housemates, to backups of important documents and save-game files you'd like to share between computers.

We recommend using a wired Ethernet connection for stability and fast transfer speeds. The project will still work if you connect your Pi via WiFi, although performance will be affected, particularly when it comes to copying over large files.



>STEP-01
How to: Set up Samba
Start with a fresh installation of Raspbian. Download the latest version of NOOBS (**magpi.cc/2bnf5XF**) and copy it to a blank micro SD card that's been formatted as fat32. Plug the micro SD card into your Pi, boot it up and opt to install Raspbian.



>STEP-02
Install Samba
Samba is available in Raspbian's standard software repositories. We're going to update our repository index, make sure our operating system is fully updated, and install Samba using **apt-get**. Open a Terminal and type:

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install samba samba-common-bin
```

>STEP-03
Create your shared directory
We're going to create a dedicated shared directory on our Pi's micro SD hard disk. You can put it anywhere, but ours will be at the top level of the root file system.

```
sudo mkdir -m 1777 /share
```

This command sets the sticky bit (1) to help prevent the directory from being accidentally deleted and gives everyone read/write/execute (777) permissions on it.

>STEP-04
Configure Samba to share your new directory
Edit Samba's config files to make the file share visible to the Windows PCs on the network.

```
sudo leafpad /etc/samba/smb.conf
```

In our example, you'll need to add the following entry:

```
[share]
Comment =  Pi shared folder
Path = /share
Browseable = yes
Writeable = Yes
only guest = no
create mask = 0777
directory mask = 0777
Public = yes
Guest ok = yes
```

This means that anyone will be able to read, write, and execute files in the share, either by logging in as a Samba user (which we'll set up below) or as a guest. If you don't want to allow guest users, omit the **guest ok = yes** line.

You could also use Samba to share a user's home directory so they can access it from elsewhere on the network, or to share a larger external hard disk that lives at a fixed mount point. Just create a **smb.conf** entry for any path you want to share, and it'll be made available across your network when you restart Samba.

>STEP-05
Create a user and start Samba
Before we start the server, you'll want to set a Samba password – this is not the same as your standard default password (raspberry), but there's no harm in reusing this if you want to, as this is a low-security, local network project.

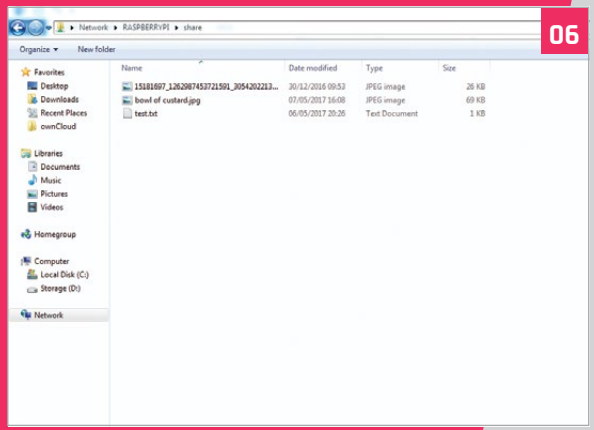
```
sudo smbpasswd -a pi
```

Then set a password as prompted. Finally, let's restart Samba:

```
sudo /etc/init.d/samba restart
```

From now on, Samba will start automatically whenever you power on your Pi. Once you've made sure that you can locate your shared folder on the network, you can safely disconnect the mouse, monitor, and keyboard from your Pi and just leave it running as a headless file server.

>STEP-06
Find your Pi on the network
You'll now be able to find your Raspberry Pi file server (named **RASPBERRYPI** by default) from any device on your local network. If you've left **smb.conf**'s default settings as they are, it will appear in a Windows network workgroup called **WORKGROUP**.



BUILD AN INTRANET WEB SERVER

Build a local HTML server with Apache

You'll Need

- Raspberry Pi 2/3
- Monitor, keyboard, and mouse (for setup)
- Wired Ethernet connection
- micro SD card with NOOBS **magpi.cc/2bnf5XF**

If you want to get to grips with how the web works, one of the most entertaining ways to learn is to build your own local intranet web server to display simple – or even complex – internal websites.

A Raspberry Pi is an ideal server for small websites that don't require the capacity or server-side processing power of a more powerful computer, and it's an ideal development environment if you're to use HTML.

It can host a personal blog or to-do list, keep a web-based calendar for the household, hold your family photo albums, or simply host a website you're developing before you're ready to share it with the world.

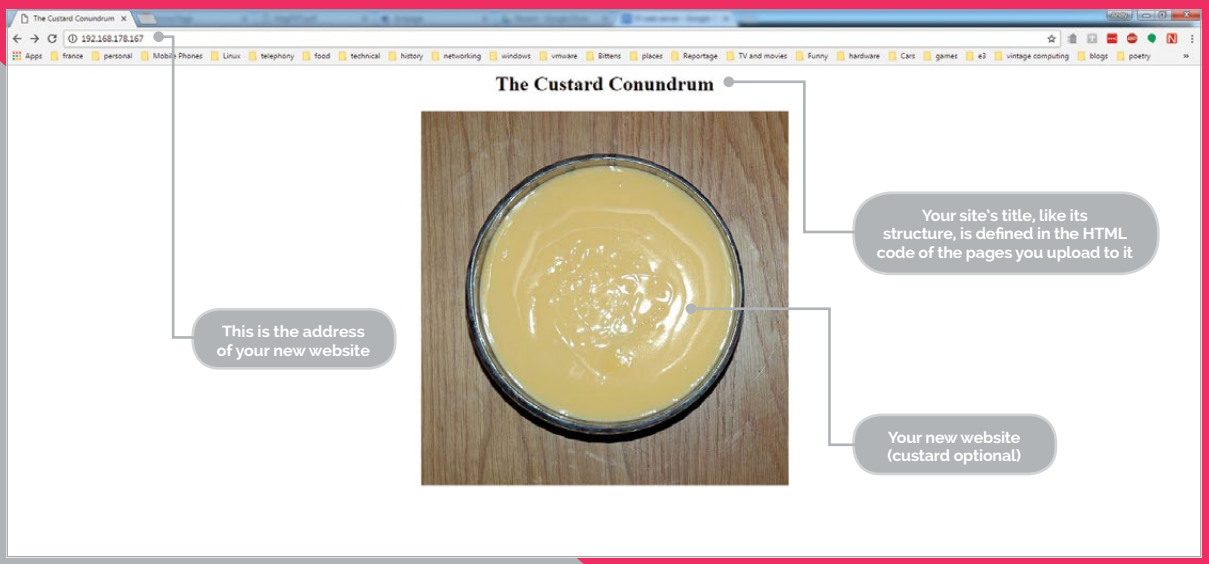
Much of the world wide web is built on LAMP – Linux, Apache, MySQL, PHP – often with a content management system (CMS) on top to make it easy to create complex websites with little knowledge of HTML or PHP.

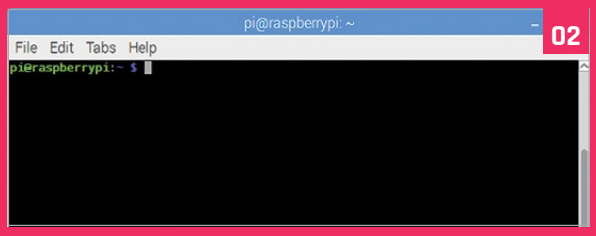
This tutorial will take you through the basics of getting your server's environment set up. We don't go as far as installing a CMS, but by the end of the following steps, your Pi will be ready to immediately display ordinary 'flat' HTML webpages. If you add the optional step of installing the MySQL database back end and PHP interpreter, you should be all set to install and configure most CMSs by following the instructions they supply.

>STEP-01

Install the OS

Hook up the keyboard and mouse to the Pi and connect it to an HDMI monitor. Copy NOOBS to a FAT32-formatted micro SD card; insert it and power up the Pi. Opt to install Raspbian with the PIXEL window manager. This will take a few minutes.





02

>STEP-02
Check the network address
Once the Pi has rebooted, open a Terminal window and run:

```
ifconfig
```

Make a note of the 'inet addr' value for eth0. This will be the IP address of your web server. It's a good idea to assign the Pi a static DHCP reservation on your router so the Pi will keep that address permanently.

>STEP-03
Update your Pi and install Apache
Run the following commands in the Terminal to make sure Raspbian is up to date. Adding **-y** to the end of **apt-get** commands instructs the program to automatically answer yes to any questions rather than waiting for you to type **Y** or **N**.

```
sudo apt-get update
sudo apt-get upgrade -y
```

Once this is complete, it's time to install your new Apache web server:

```
sudo apt-get install apache2 -y
```

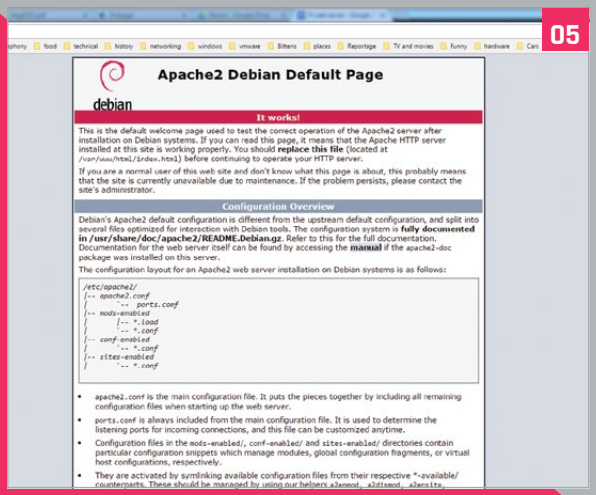
Apache is the main piece of software you need to serve webpages to client PCs.

>STEP-04
Add PHP and MySQL (optional)
Many websites use content management systems, such as WordPress. These require PHP and MySQL, so if you want to experiment with CMS-driven sites further down the line, this is an ideal time to add the software you'll need to render them. You can install both at once using the following command:

```
sudo apt-get install php5 mysql-server -y
```

Chose a strong password for the MySQL 'root' user and note it down somewhere safe.

>STEP-05
Test Apache
Open your web browser of choice, either on the Pi or on another PC on your local network, and enter the IP address from step 2 into the address bar. You should see the Apache 2 Debian default page, with a red banner and the words 'It Works!' across it.



05

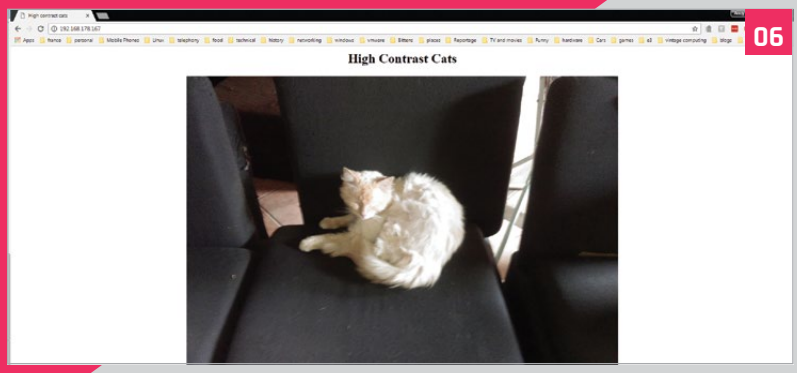
>STEP-06
Start building your site
Your website's files are located in the **/var/www/html** directory. Run the following to make this folder accessible to the default user, **pi**.

```
sudo chown -R pi /var/www/html
```

Delete the **index.html** file found there and upload your new intranet site to that folder. Your new site should now be accessible from your local network on the address you used in step 5.

Your Pi is an ideal environment for web development, so you can simply drop the HTML files you're working on into your web server directory. If you want to experiment with more advanced options, you can enable FTP and SSH for remote access to your web server from other computers on your local network. You can also install a CMS such as WordPress, which you'll be able to access from a browser on your local network to create content-rich websites.

While this is strictly a local web server project, the same software and processes go into an internet-facing server, although that will require firewall configuration and extra security measures that are beyond the scope of this tutorial.



06

HOW TO BACK UP YOUR RASPBERRY PI

Back up the files on a Raspberry Pi so you can quickly restore your files

You'll Need

- Raspberry Pi
- SD card
- Flash drive

One of the great things about using a Raspberry Pi is how easy it is to reset the whole system. If you're working on a project and you completely mess something up, it only takes minutes to reinstall the operating system and start again. Swift restoration is just one thing that makes the Raspberry Pi ideal for experimentation.

This constant state of being 'ready-to-refresh' can be a double-edged sword, though. The Raspberry Pi is a useful microcomputer: you can set up email and use the internet, write documents, install and use coding tools, and work on all manner of long-term projects.

If you use a Raspberry Pi for a long time, you'll want to think about a backup solution. A backup enables you to restore the Raspbian OS and your files.

Fortunately, there are several different ways to go about backing up the OS and your files.

Option 1: Copy the SD card image

The simplest way to back up your Raspberry Pi is to copy the entire SD card as an image.

This technique is the reverse of flashing your SD card when installing an OS to it. Instead of copying an image file from your computer to the SD card, you copy the entire SD card to an image file on your computer. This is, in fact, how image files are created in the first place.

Power down your Raspberry Pi and remove the SD card. Place it into an SD card reader and connect it to your computer.

Open a Terminal window on a Mac or Linux computer, and enter **df**. Take a look at what drives you have on your system. Now attach the SD card to your computer, and enter **df** again.

Spot the newly mounted drive: on a Linux machine, it will be something like /dev/sdb1, and on a Mac it will say /dev/disk2s1. The numbers may be different, so be sure to check carefully.

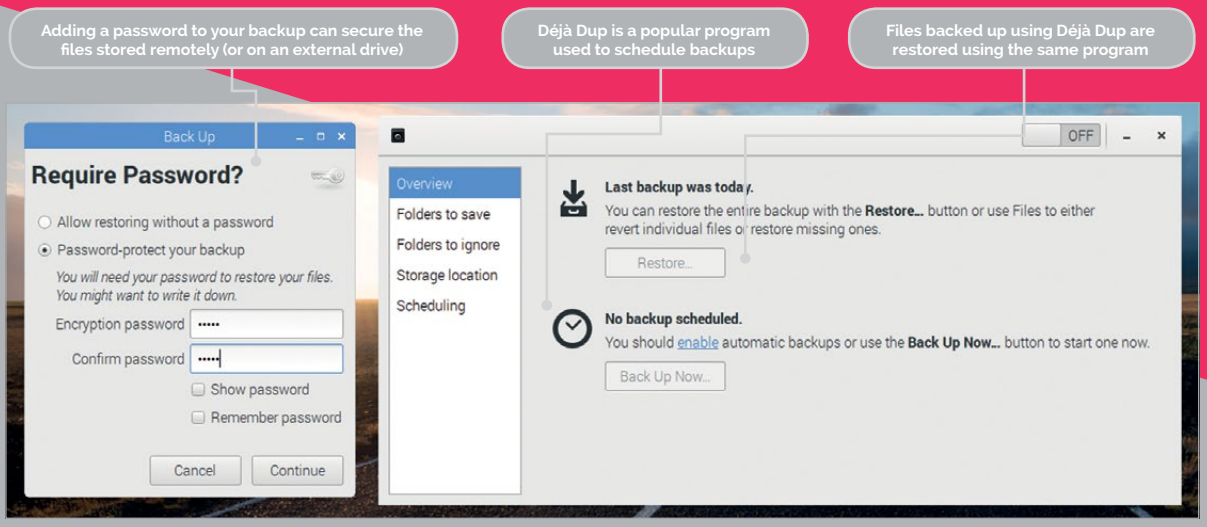
On Linux:

```
sudo dd bs=4M if=/dev/sdb of=raspbian.img
```

On a Mac:

```
sudo dd bs=4m if=/dev/rdisk2 of=raspbian.img
```

You can then use the **raspbian.img** file to restore your entire operating system (in its current state) to an SD card using **dd** in reverse, or by using an app such as Etcher (**etcher.io**) to flash the SD card.



In Windows, you back up the SD card using Win32 Disk Imager (**magpi.cc/2bndEsr**).

Open the program and click Yes to the security alert window. Enter **C:\raspberrypi.img** in the Image File text box and click Read. The SD card will be written to the image file. When it says 'Read Successful', you can click OK.

Option 2: Back up the Home folder

The challenge with turning the SD card into an image file is doing it on a regular basis. You have to remove the SD card from your Raspberry Pi, attach it to a Mac or PC, and complete the whole backup.

The second backup option is to back up just your Home folder as a compressed file. Uncompressing the file enables you to browse and restore individual files and directories. Use the Terminal on your Raspberry Pi:

```
cd /home/
sudo tar czf pi_home.tar.gz pi
```

Copy the **pi_home.tar.gz** file to a USB flash drive for safe keeping.

Option 3: Schedule backups

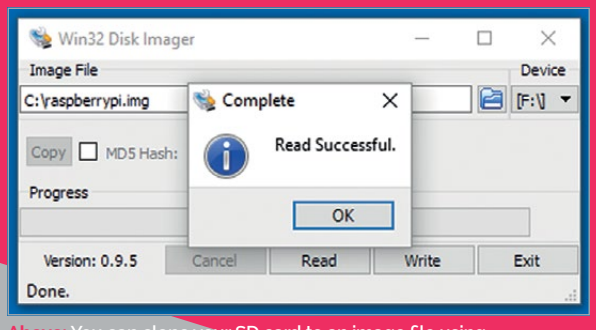
It is possible to schedule backups manually by scheduling the tar command as a cron job. See 'Scheduling Tasks with Cron' (**magpi.cc/2kg73Xs**).

Some Raspberry Pi users choose to use rsync (**magpi.cc/2kgchCH**) instead of tar, because this smartly copies updated files rather than the whole system. But you need to create an exclude file that ignores the contents of system folders. Take a look at this Stack Exchange discussion if you're interested (**magpi.cc/2kg1qIQ**).

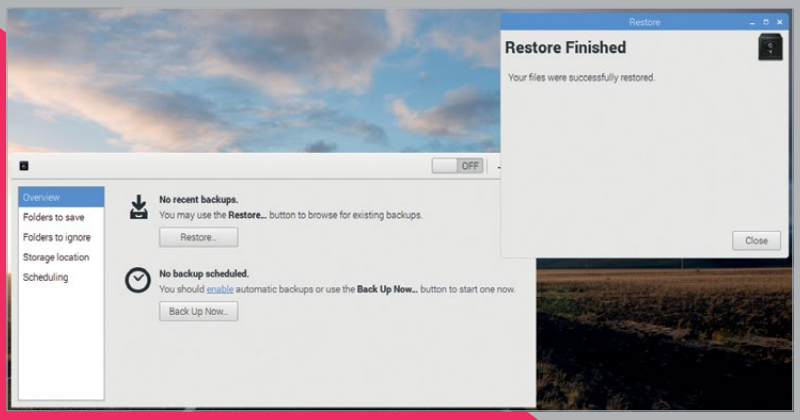
An easier approach is to use a free program called Déjà Dup. This automates rsync and gives it a user-friendly interface. It's an easy program to use, and you can back up your Raspberry Pi using Amazon S3, SSH, FTP, or by copying the files directly to a flash drive.

You can pick directories to include and exclude, and it's easy to restore backups. Déjà Dup is installed by default on Ubuntu, and is easy to install in Raspbian:

```
sudo apt-get update && sudo apt-get upgrade
sudo apt-get install deja-dup
```



Above: You can clone your SD card to an image file using dd on a Mac or Linux, and Win32 Disk Imager on a PC



Above: Files can be restored individually, or you can restore the whole of your Home folder to a fresh installation of Raspbian

Now you'll find a new program called Backups in Menu > Accessories.

Click on Folders to save and check that Home (pi) is listed. Folders to ignore should include Rubbish Bin.

Now click on Storage Location and change Amazon S3 to your flash drive.

Go back to Overview and click Back Up Now. Déjà Dup will start the first backup of your Home folder.

You'll be asked if you want to enter an encryption password. This is a good idea for security. Enter the same password in the Encryption Password and Confirm Password fields. Click Continue.

The first backup may take a while, depending on the size of your Home folder.

When Déjà Dup has finished backing up successfully, you can set up a schedule. Set the Off button to On. Click Scheduling and change Week to Day.

Backup will now store the contents of your Home folder to the flash drive on a daily basis, ready to be restored whenever you need it.

Restoring files

If you've had a total system failure and had to wipe your SD card, you'll need to reinstall Déjà Dup (follow the instructions earlier).

Make sure your backup flash drive is attached to the Raspberry Pi. Now open Accessories > Backups.

Click Restore to reinstall any missing files or directories. A 'Restore From Where?' window will open if you are installing from a fresh Raspbian image. Choose your flash drive using the Backup Location. Click Forward.

Choose the date to pick a backup from which you want to restore and click Forward. You have two options: restore files to original locations and restore to a particular folder. Choose to restore files to their original locations. Click Forward and Restore.

If you entered a password, you have to enter it in the Encryption Password field. Click Continue.

All the files from the Home folder in your previous backup will now be restored. You will see 'Restore Finished' when all your files are back. Click Close. You will now find all the files and directories from your Home folder reinstalled.

POWER YOUR RASPBERRY PI

Don't let your Raspberry Pi suffer from a shortage of volts

You'll Need

- Raspberry Pi
- Raspbian OS
- Power supply

Powering your Raspberry Pi is, on the surface, a remarkably simple affair. All Raspberry Pi boards are designed to use the same micro USB power socket as many smartphones. All you need is a spare USB adapter, and it will provide power to your Raspberry Pi.

That's the basic requirement. Of course, this being a board for makers and hackers, there's far more to it than that. Learning how to properly supply the right amount of power to your Raspberry Pi board is important as you start to create complex projects.

A 5V micro USB typically powers the Raspberry Pi. But how much current (in milliamps or amps) the Pi requires to function depends on your usage.

The recommended amount is between 700mA for a Raspberry Pi Model A, and up to 2.5A for a Raspberry Pi 3 Model B (see 'Power Supply Requirements' box).

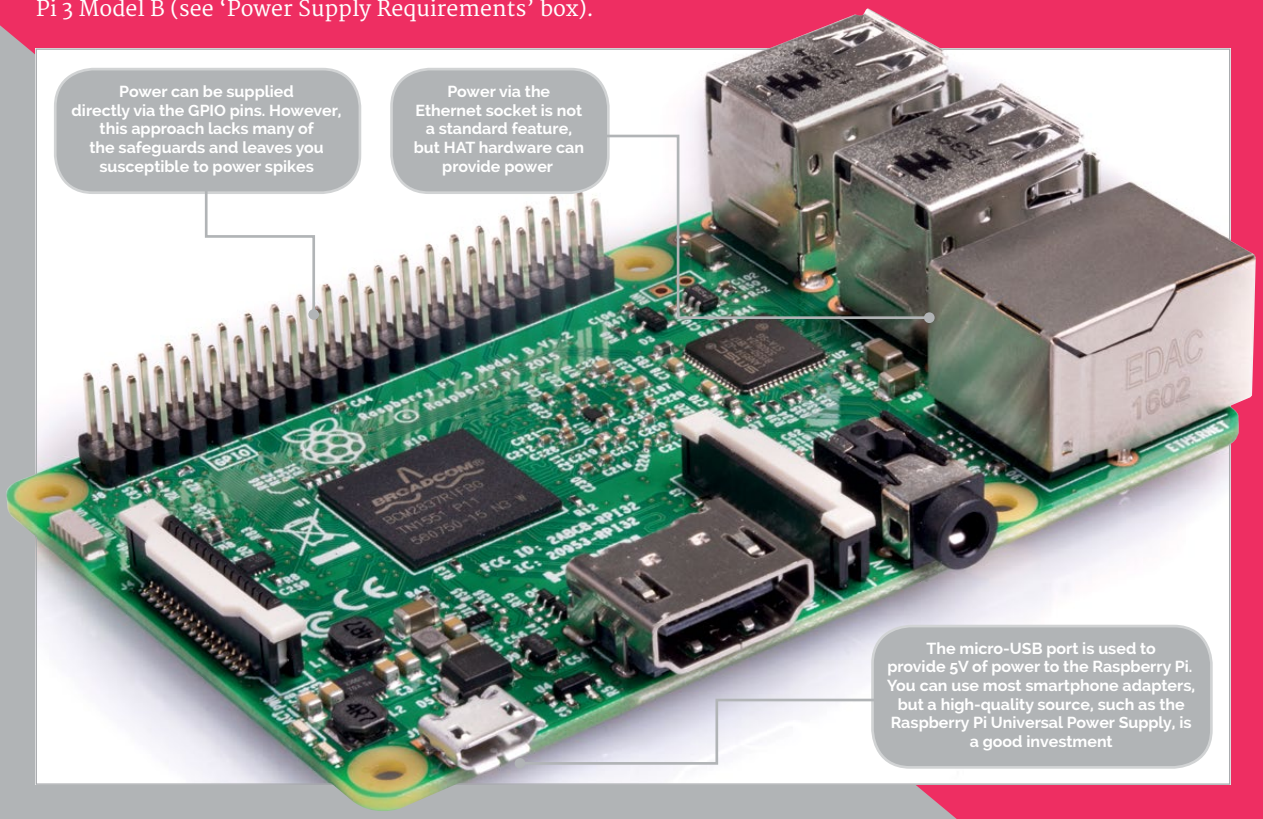
The Raspberry Pi boards typically draw much lower amounts, between 200 and 500mA.

Usage depends on what you're doing with the Pi. Playing video and browsing the web draws more power than idling and booting. It also depends on what devices you have connected; some keyboards and mice draw more power than others.

Low power warning

So, the Raspberry Pi can function on lower power supplies, but it may have problems when you start to do demanding tasks or add peripherals.

The Raspbian operating system comes with a low voltage indicator icon. This yellow lightning bolt appears in the top-right of the screen. It is used to indicate an under-voltage warning.



The Raspberry Pi Universal Power Supply is a reliable source of power for your board

Drawing too much power from the Raspberry Pi board isn't dangerous, but it can cause erratic behaviour. If you want to attach several performance-heavy peripherals to your Raspberry Pi, it's best to use a powered USB hub.

Official supply

The Raspberry Pi Universal Power Supply (**magpi.cc/2a14pye**) is a dependable power source. It'll keep feeding your Pi the steady 2A it needs for proper performance. If you're worried about getting stable power, then it's a good idea to invest in a suitable adapter.

Power via GPIO

A more technical (and dangerous) way to power the Raspberry Pi is via the GPIO pins.

The 5V GPIO pins on a Raspberry Pi are connected to the 5V rail. Typically they provide the remaining power from the Raspberry Pi (that isn't being used to power the board itself). So you can hook up the GPIO 5V pins to a 5V power source and feed power directly to the board.

```
Connect a 5V source to Pin #2 (5V).
Connect the ground of that source to Pin #6 (GND).
```

Please be aware that there is no regulation or fuse protection on the GPIO to protect from over-voltage or current spikes.

If an incorrect voltage is applied, or a current spike occurs on the line, you can permanently damage your Raspberry Pi.

Power Usage

	Pi 1 (B+)	Pi 2 B	Pi 3 B	Zero
Boot	Max 0.26A	0.40A	0.75A	0.20A
	Avg 0.22A	0.22A	0.35A	0.15A
Idle	Avg 0.20A	0.22A	0.3a0A	0.10A
Video playback (H.264)	Max 0.30A	0.36A	0.55A	0.23A
	Avg 0.22A	0.28A	0.33A	0.16A
Stress	Max 0.35A	0.82A	1.34A	0.35A
	Avg 0.32A	0.75A	0.85A	0.23A

Test conditions used a standard Raspbian image (26 Feb 2016), at room temperature, connected to an HDMI monitor, USB keyboard, and mouse. For the Model 3B it was connected to a WiFi access point. All these power measurements do not take into account power consumption from additional USB devices; these measurements can easily be exceeded with multiple additional USB devices connected or when using a HAT.

Power via HAT

If you're looking to power the Raspberry Pi via the GPIO pins, then it's best to go via a HAT. The hardware sits on top of the GPIO pins and adds the safety features you need.

Power over Ethernet

Supplying power to your Raspberry Pi via the Ethernet cable is an attractive proposition, but it's not provided as standard. If you're interested in providing power to your Raspberry Pi via a network cable, then take a look at the Pi PoE Switch HAT (**magpi.cc/2lhnDDT**).

Mobile power

It is possible to provide power to the device from a battery pack designed for mobile phone charging. Companies such as Anker (**anker.com**) and Poweradd (**ipoweradd.com**) are known for their mobile charging solutions.

But if you want something a little more serious, try the Zero LiPo. This HAT supplies power from lithium batteries via the GPIO pins with safety management features (**magpi.cc/2yizv2J**).

Power Supply Requirements

Product	Recommended PSU current capacity	Maximum total USB peripheral current draw	Typical bare-board active current consumption
Raspberry Pi Model A	700mA	500mA	200mA
Raspberry Pi Model B	1.2A	500mA	500mA
Raspberry Pi Model A+	700mA	500mA	180mA
Raspberry Pi Model B+	1.8A	600mA/1.2A (switchable)	330mA
Raspberry Pi 2 Model B	1.8A	600mA/1.2A (switchable)	350mA
Raspberry Pi 3 Model B	2.5A	1.2A	~400mA
Pi Zero / Pi Zero W	1.2A	1.2A	200mA

GET HELP FROM THE COMMAND LINE

The basic commands you need to get advice from the Linux command line

In our Switch to the command line tutorial (*Page* 34), we looked briefly at 'man', the manual you can access from the command line.

The man tool is so important that we think it deserves a more thorough explanation. And man isn't alone in offering help on the command line. Other commands like **whatis**, **info**, and **apropos** all offer support and assistance. And let's face it, support and assistance are what you will often need at the command line.

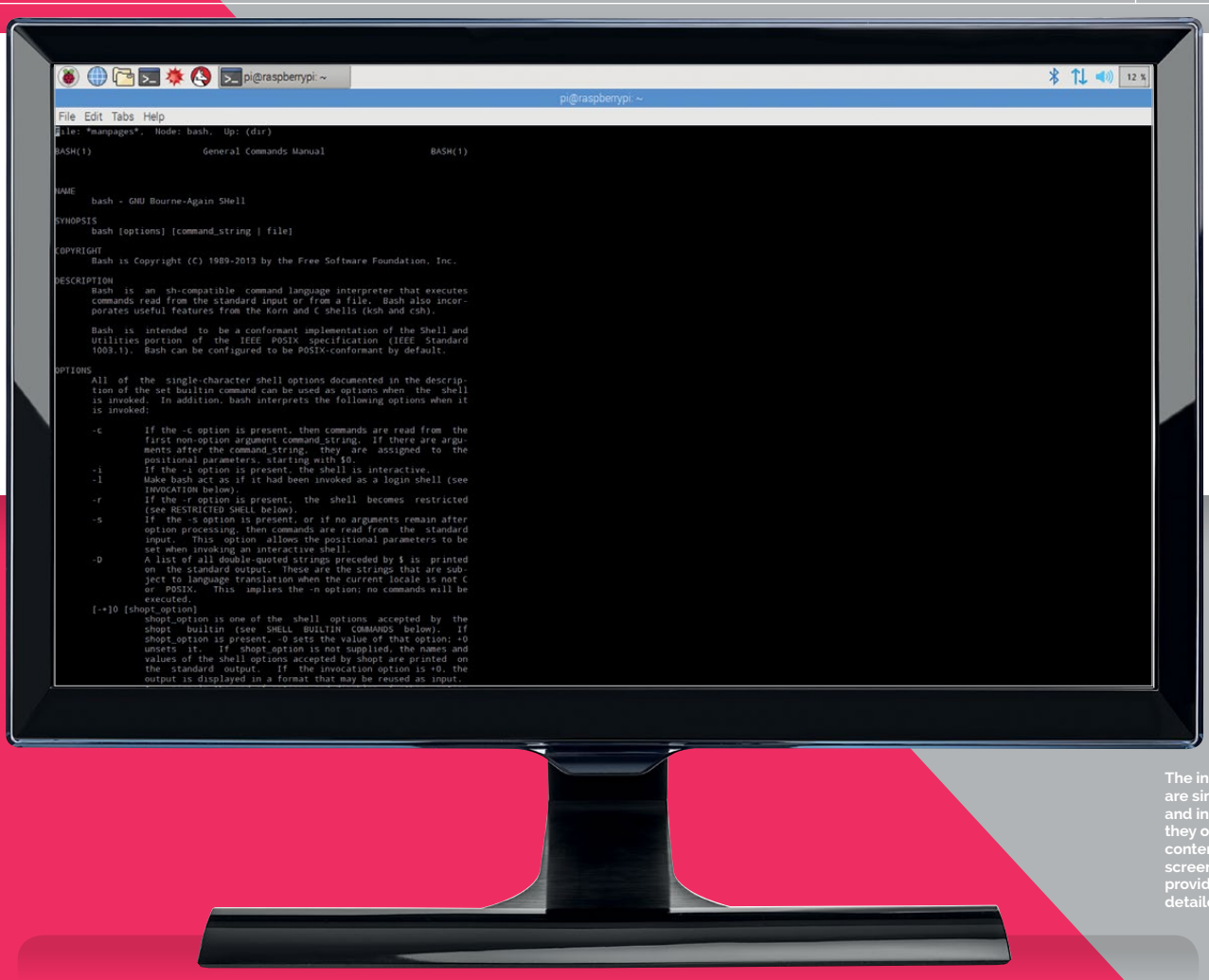
Even seasoned coders don't always know the correct command to type into the Linux terminal. This guide is all about the various ways to get help at the command line, so no matter what command you come across, you'll be able to find out more information on how to use it.

man

Your first point of call for getting help on the command line is man (short for 'manual'). Enter **man** followed by the name of a command to get detailed information about it. For instance, enter:

```
man passwd
```

...and you will see detailed information about the tool used to change your password. Man screens are displayed one page at a time. Press the **SPACE** bar to



The info screens are similar to man, and in many cases, they offer the same content. But some info screens (like bash) provide much more detailed content

move to the next page, and press **Q** to exit the page and return to the command prompt.

Man pages can be a bit tricky to read at first, but you'll soon get the hang of it.

At the top are the Name, Synopsis, and Description sections. Read these to get an overview of the command. Below them you'll find options and parameters; read these carefully to discover ways to expand your usage of each command. It's a good idea to use man on any commands you know, and read the manual for any new Linux commands you come across.

You can even read a man page for man:

```
man man
```

Press **H** in the man screen to view a summary of navigational key presses. These are worth learning so you can do more than press space to move to the next page.

Man's lesser-known partner is 'info', which is used to display information pages associated with commands. Sometimes these are the same as the man pages. In other cases they provide a different description. Try these:

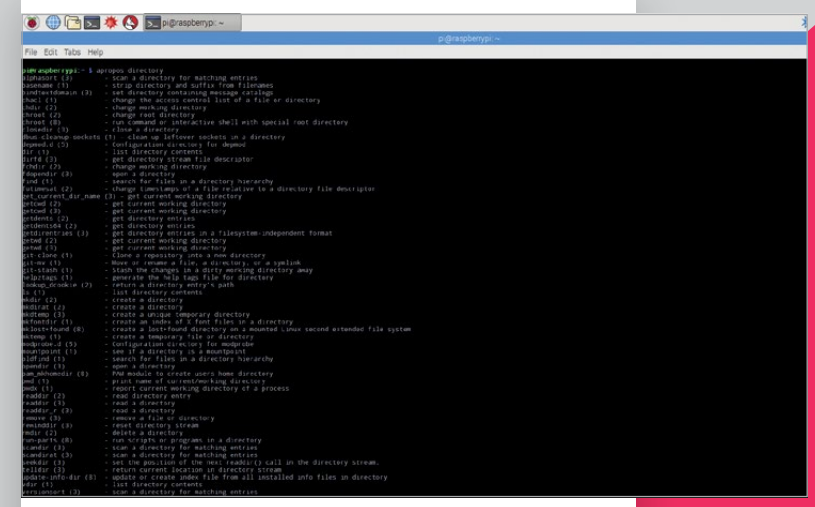
```
man bash
info bash
```

While **man bash** gives you a brief description of the GNU Bourne-Again Shell and the options used with the bash command; **info bash** gives the whole history and hundreds of pages of detailed information.

Press **H** on an info screen to view the controls for navigating such long documents. As well as **SPACE** to move down, you use **DELETE** to go back a screen, **TAB** to highlight links, and **RETURN** to use them. Press **Q** to exit the help screen.

```
File Edit Tabs Help
pi@raspberrypi:~ $ mktemp --help
Usage: mktemp [OPTION]... [TEMPLATE]
Create a temporary file or directory, safely, and print its name.
TEMPLATE must contain at least 3 consecutive 'X's in last component.
If TEMPLATE is not specified, use tmp.XXXXXXXXXX, and --tmpdir is implied.
Files are created u+rw, and directories u+rwx, minus umask restrictions.

  -d, --directory     create a directory, not a file
  -u, --dry-run       do not create anything; merely print a name (unsafe)
  -q, --quiet         suppress diagnostics about file/dir-creation failure
      --suffix=SUFF   append SUFF to TEMPLATE; SUFF must not contain a slash.
                        This option is implied if TEMPLATE does not end in X
  -p DIR, --tmpdir[=DIR]  interpret TEMPLATE relative to DIR; if DIR is not
                        specified, use $TMPDIR if set, else /tmp.  With
                        this option, TEMPLATE must not be an absolute name;
                        unlike with -t, TEMPLATE may contain slashes, but
                        mktemp creates only the final component
  -t                  interpret TEMPLATE as a single file name component,
                        relative to a directory: $TMPDIR, if set; else the
                        directory specified via -p; else /tmp [deprecated]
      --help     display this help and exit
      --version  output version information and exit

GNU coreutils online help: <http://www.gnu.org/software/coreutils/>
Full documentation at: <http://www.gnu.org/software/coreutils/mktemp>
or available locally via: info '(coreutils) mktemp invocation'
pi@raspberrypi:~ $
```

Many commands feature a built-in help option, accessed with `-h` or `--help`. Using it offers a brief outline



Apropos is used to search the manual for matching words. With it, you can find commands based on subjects, such as directory, password, or links

FINDING COMMANDS

As you become more familiar with man and info, you'll start searching for commands to look up. Here, the **man -k** command comes in useful. In particular, try this:

```
man -k directory | more
```

This command lists all available man entries. Press **SPACE** to run through them one at a time. The **man -k** option is worth remembering. If you use **man man**, it tells you the **-k** option is 'equivalent to apropos'. Apropos is used to search manual page names and descriptions. It's a handy way to find commands when you don't know their names.

For instance, enter:

```
apropos directory
```

...and you'll get a list of all the commands that have the word 'directory' in their description or page name. Here you'll find common commands such as **ls**, **cd**, and **pwd**, but you'll also find less obvious commands, such as **mktemp**.

Next to each command is a number, like (1) or (2). These correspond to the section numbers of the manual (view using **man man**).

The section numbers are useful for guiding you to the commands that can be used on the command line. As a general rule, **1:** Executable programs or shell commands, and **2:** System calls, both tend to be worth investigating. Higher numbers are for library calls, special files, and kernel routines for advanced users.

You can find out more information about any command using man:

```
man mktemp
```

This command gives you detailed information on how to create temporary directories.

TAB AUTOCOMPLETE

Another way to find files is to use 'tab autocomplete'. By pressing the **TAB** key, you can automatically complete commands, files, and directories on the command line. If you're not doing so already, learn to press **TAB** a lot on the command line: it's a good way to discover new commands.

Take the **apt** tool, for example. There are **apt-get** and **apt-cache**, but did you know about **apt-config** and **apt-key**?

Enter:

```
apt
```

And press the **TAB** key twice. It will display all the different types of apt available.

You can even run through the letters of the alphabet.

Enter the letter 'a':

```
a
```

And press **TAB** twice to view all the commands beginning with 'a'. You can then use man to look up commands. It's a great way to broaden your knowledge of the command line.

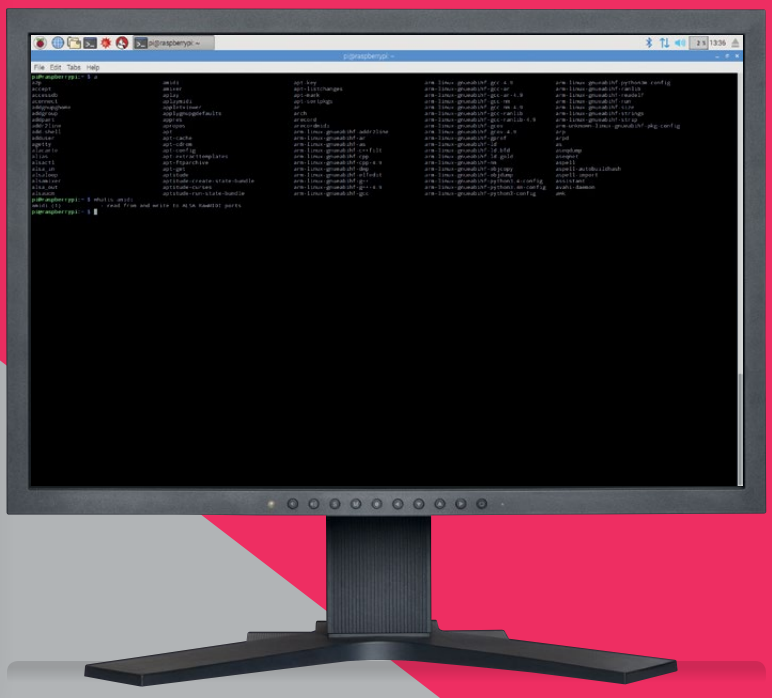
EXPRESS HELP

Many commands also offer a help feature as an option. Help is typically accessed using **-h** or **--help**:

```
mktemp --help
```

Enter a single letter and press TAB twice to view all the commands available in that letter. Then use whatis to get a short one-line description of each command

This command displays the options offered by the mktemp command. It's the same as the first page of mktemp's man file, but saves you digging in and out of the full document.



Not all commands make use of **--help**. Some, like **ls --help**, display the full man document (you can pipe this through **less**:

```
ls --help | less
```

...but it's typically easier to use **man ls**. Some commands don't implement the help option at all.

```
pwd --help
```

...returns 'invalid option'. But it's worth trying when you are experimenting with new commands.

One final command worth using when searching for commands is whatis:

```
whatis pwd
```

This example returns 'print name of current/working directory'. Often, this brief description is enough to let you know what it does, or at least tell you if it's something you'd like to investigate further with man or info.

These are just some of the tools you can use to get help at the command line in Linux. While the command line may seem intimidating at first, you're far from alone in this text-only environment.

WEB SEARCH

One of the advantages when using a desktop interface, like Raspbian, is that a web browser – and a search engine – is just a click away.

Getting online from the command line is a lot easier than you'd imagine. There are many different text-based web browsers that enable you to access Google, Bing, DuckDuckGo, and other websites without having to boot into the PIXEL desktop interface.

We're going to use:

```
sudo apt-get update && sudo apt-get upgrade
sudo apt-get install elinks
```

Now you can open the web browser from the command line using:

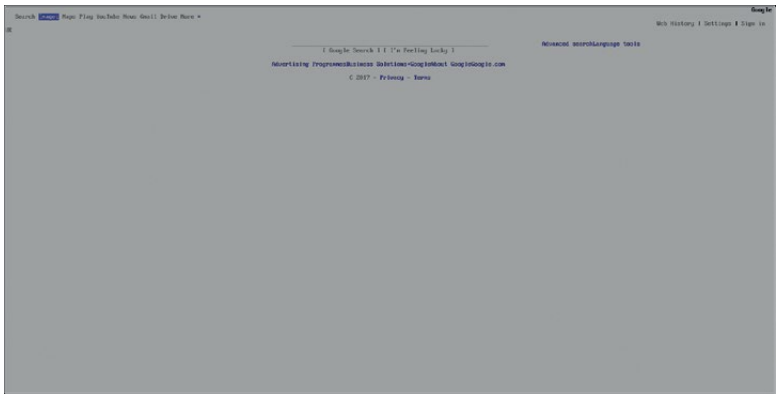
```
elinks
```

The elinks interface is full-screen, so it replaces the command line. Press **g** to open a URL field. You can enter full URLs, such **http://www.google.com** or just shortened versions, such as **raspberrypi.com**.

Better yet, there are a few key bindings for helpful sites. Press **g** then enter these shortcuts:

- **d** – dict.org search
- **sd** – Slashdot
- **g** – Google search

You can also enter Google search terms in the URL field. Press **g**, then enter 'g the magpi' to search for our website in Google.



Other keyboard shortcuts can be used to navigate the program:

- **g** – Goto URL
- **Down Arrow** – Next link
- **Up Arrow** – Previous link
- **Return** – Select link
- **Left Arrow** – Back
- **u** – Forward
- **q** – Quit
- **.** – Toggle link numbering
- **%** – Toggle colours
- **t** – New tab
- **T** – Open link in new tab
- **>** – Next tab
- **<** – Previous tab
- **c** – Close tab

GETTING STARTED WITH MINECRAFT: PI EDITION

Get off to a good start with Minecraft: Pi Edition.
Play the game and write your first program using the API.

If you've never played Minecraft and you want to be a master block builder, we'll help you get stuck into the game, build a house, and get started with the API.

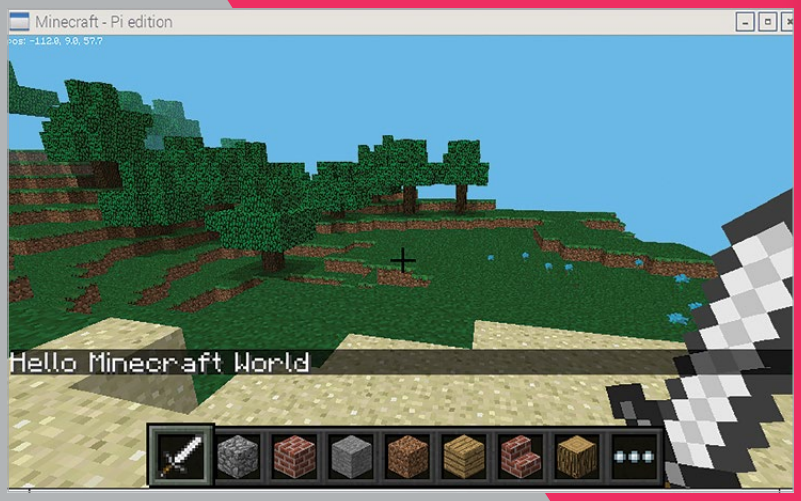
Minecraft is a game which has achieved monumental success; more than 120 million copies have been sold across all its versions. Not bad for a game which doesn't really have a point! If it does have a point, as an indie sandbox game, it's to make stuff. And people have really made stuff, from fully functioning computers to scale models of the Starship Enterprise.

The best things about Minecraft: Pi Edition are that it's free, and that it comes with an API. You don't get this with any other version of Minecraft.

Minecraft is installed by default on Raspbian. If you have an older version, you can get it by opening a Terminal (Menu > Accessories > Terminal) and typing these commands, pressing **ENTER** after each:

```
sudo apt-get update
sudo apt-get install minecraft-pi
```

Use the API to write a 'Hello Minecraft World' program



Playing the game

Click Menu > Games > Minecraft: Pi Edition to run the game. Minecraft: Pi Edition offers one playing mode, Classic, which is all about exploring and building. Click Start Game, then click Create New (or choose an existing world) to enter a world.

- **The mouse** changes where you look
- **Holding the left button** destroys blocks
- **Right button** places blocks
- **W, S, A, D** move you forward, backward, left, and right
- **1, 2, 3, 4, 5, 6, 7, 8** change what you are holding
- **E** opens the inventory
- **ESC** takes you back and to the **Menu**
- **SPACE** is jump. Double-tapping it makes you fly or stop flying

The API

The API (application programming interface) allows you to write programs which control, alter and interact with the Minecraft world, unlocking a whole load of Minecraft hacking. How about creating massive houses at the click of a button; writing a game which uses a LED and buzzer to help you find a block; or recreating Nintendo's Splatoon in Minecraft?

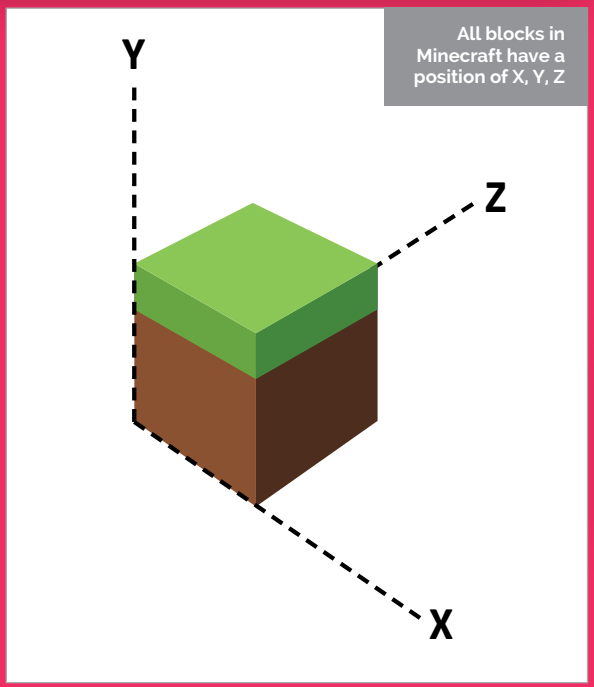
The API works by changing the world as the game is being played, allowing you to:

- Get the player's position
- Change (or set) the player's position
- Discover the type of block at a specific location
- Change a block
- Change the camera angle
- Post messages to the player

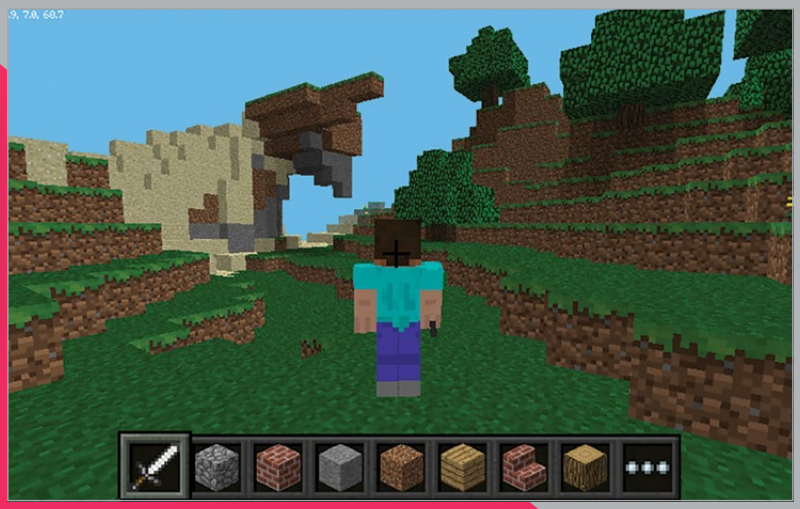
Hello Minecraft World

The first program all programmers create when learning something new is called Hello World, which puts the phrase 'Hello World' on the screen. You're going to do the same, but in Minecraft:

01. Go to the Minecraft menu with **ESC**, but leave the game running (you can minimise it).
02. Open IDLE by clicking Menu > Programming > Python 3 (IDLE).
03. Use File > New File to create a new program and save it as **hellominecraftworld.py**.
04. At the top of your program, type the following code to import the `minecraft` module, which will allow you to use the API and talk to the game:
```
import mcpi.minecraft as minecraft
```
05. On the next line, create a connection from your program to Minecraft and call it `mc`:
```
mc = minecraft.Minecraft.create()
```
06. Use your Minecraft connection and the function `postToChat()` to put a message in the chat window on a third line:
```
mc.postToChat("Hello Minecraft World")
```
07. Run your program by clicking Run > Run Module.



All blocks in Minecraft have a position of X, Y, Z



Minecraft: Pi Edition has an API you can use to program the game

Switch back to Minecraft, click 'Back to game', and you should see the message 'Hello Minecraft World' on the screen. Be quick, though, as the message only stays on the screen for ten seconds.

Any errors will appear in red text in the Python Shell window. Check your code carefully for spelling mistakes, and ensure that you have used the right upper- or lower-case letters.

When you have successfully made the message appear on the screen, try changing it and running the program again.

WATCH FOR RED TEXT

Any errors in your program will appear in the Python Shell in red text.

Teleportation

Using your new Python programming skills and the Minecraft API, you can teleport Steve around the world by adding just one more line of code to your program.

Minecraft is a world of blocks, all about 1m × 1m × 1m. The player and every block in the world has a position made up of x, y, and z: x and z are the horizontal positions and y is the vertical. By changing the player's x, y, and z position, you can teleport them wherever you want.

The player starts at position x = 0, y = 0, z = 0, which is the spawn point, and the player's current position is shown at the top left of the screen.

Add the following code to your 'Hello Minecraft World' program to teleport the player to position x = 0, y = 50, z = 0, which will put your player 50 blocks up in the air:

01. Teleport the player by setting their position:
```
mc.player.setPos(0, 50, 0)
```
02. Run your program by clicking Run > Run Module.
03. Quickly switch back to Minecraft to see your player fall to the floor (unless in flying mode).

Try changing the values in `setPos()` to teleport your player to different places around the world. Use values -125 to 125 for x and z and -64 to 64 for y, otherwise the player will be teleported outside the world.

PYTHON IS CASE-SENSITIVE

Beware of upper- and lower-case letters: 'Minecraft' and 'minecraft' are different things to Python.

MINECRAFT PI CODING TIPS

You'll Need

- Raspbian
- Minecraft: Pi Edition
- Python 2 editor (IDLE)
- Getting started with Minecraft: Pi Edition: bit.ly/1Epg9LKC

If you've completed the Minecraft Pi learning resources at **raspberrypi.org**, check out these pro tips and mini programs to learn more about the coding in Minecraft...

BUILD A HOUSE

The quickest way to make a house in *Minecraft: Pi Edition* is to use code and the API. By programming a house rather than building it by hand, it can be any size you want – 10 blocks across or 100!

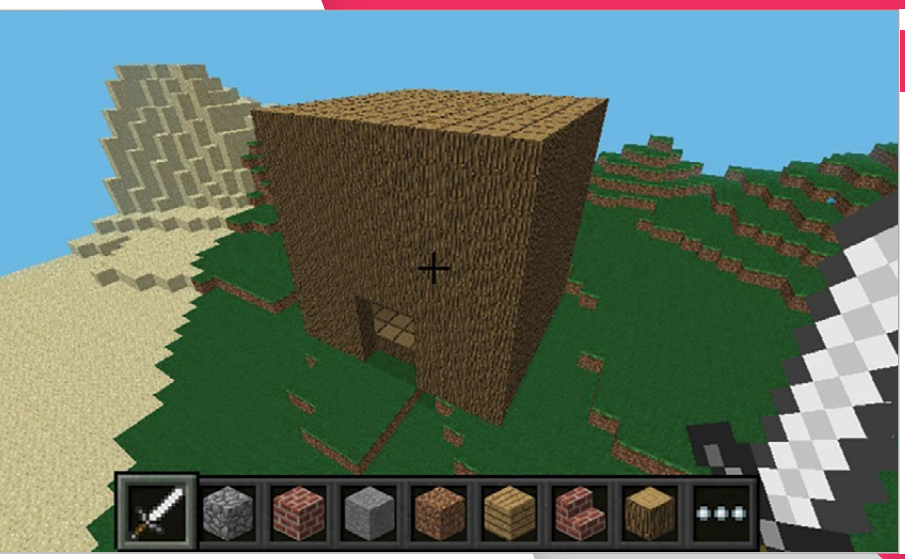
Create a simple program which will use the `setBlocks()` function, once to create a cube 10×10×10 of wood (17) and then again to create a cube of air (0) 9×9×9 inside the wooden cube.

```
from mcpi import minecraft
mc = minecraft.Minecraft.create()

pos = mc.player.getTilePos()
mc.setBlocks(pos.x + 0, pos.y + 0, pos.z + 0,
        pos.x + 10, pos.y + 10, pos.z + 10, 17)

mc.setBlocks(pos.x + 1, pos.y + 1, pos.z + 1,
          pos.x + 9, pos.y + 9, pos.z + 9, 0)
```

You can then use `setBlocks()` again to create an entrance by building another block of air (0).

```
mc.setBlocks(pos.x + 4, pos.y, pos.z,
      pos.x + 6, pos.y + 3, pos.z, 0)
```

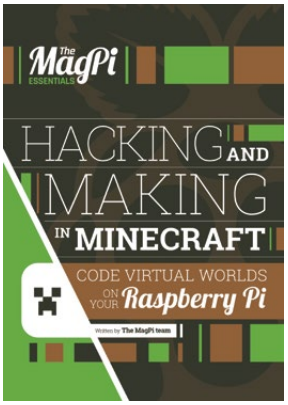
The limits of coding a house are endless – why not add a stone roof, a wool floor, some torches to the outside?

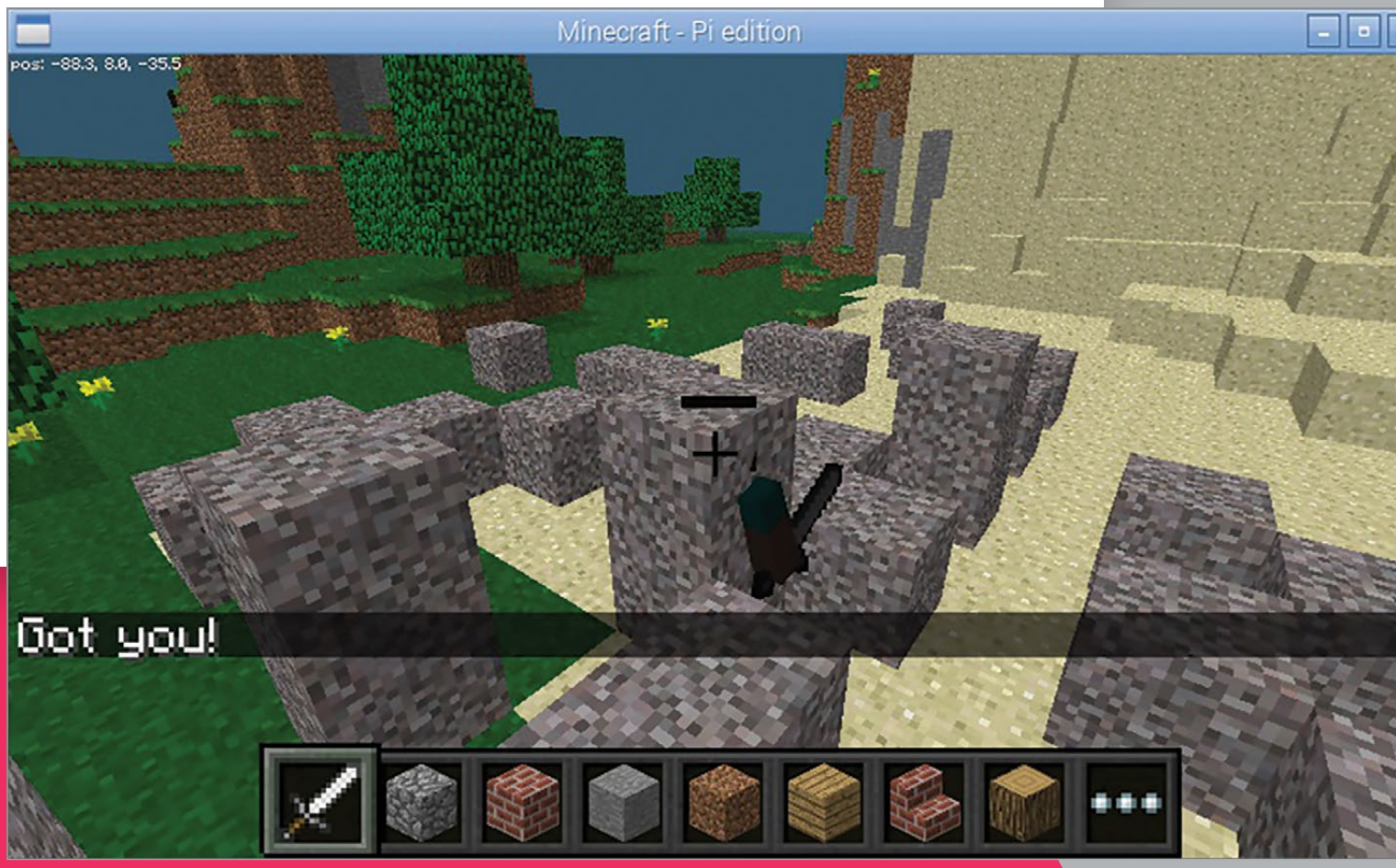


Below Create massive houses in the blink of an eye using just a few lines of code

MINECRAFT ESSENTIALS

Love this Raspberry Pi feature and want to do more modding with Minecraft? Check out our Essentials book, Hacking and Making with Minecraft. Inside you'll discover how to build your own personal world, control GPIO pins and electronics and build your own Minecraft projects: magpi.cc/Minecraft-book





Below Use blocks affected by gravity such as gravel to create your own *Minecraft* mini-game

> Using the gravity effect of blocks is a great way to add something new to your Minecraft programs

USE GRAVITY-EFFECTED BLOCKS

Sand and gravel block types in *Minecraft* are affected by gravity and will fall down if the block below is air.

The same gravity effect occurs if a block is placed in the world using the API. So if you were to create a block of gravel (13) 25 blocks above the player, it would fall on the player's head. In a new program, type:

```
from mcpi import minecraft
mc = minecraft.Minecraft.create()
pos = mc.player.getTilePos()
mc.setBlock(pos.x, pos.y + 25, pos.z, 13)
```

Using the gravity effect of blocks is a great way to add something new to your *Minecraft* programs. Here is a simple program which loops until it manages to drop a gravel block onto Steve's head. If Steve wants to stay in the game, he has to keep moving so the gravel misses him. Start a new program:

```
from mcpi import minecraft
from time import sleep
mc = minecraft.Minecraft.create()

pos = mc.player.getTilePos()

while mc.getBlock(pos.x, pos.y, pos.z) !=
13:
    mc.setBlock(pos.x, pos.y + 25, pos.z,
13)
    sleep(1)
    pos = mc.player.getTilePos()

mc.postToChat("Got you!")
```

Below Change the position of the 'camera' in *Minecraft* and get a different view of the world

CHANGE THE CAMERA

Bored of always following Steve around? You can alter the position of the 'camera' in *Minecraft* to change how you see the world.

You can change the camera to follow Steve while looking directly down at him, or to look down at the world from any coordinate in *Minecraft*.

The **camera.setFollow()** function will change your view so you are looking down at Steve. In a new script:

```
from mcpi import Minecraft
mc = minecraft.Minecraft.create()

mc.camera.setFollow()
```

To change the camera to look down on any position, you use the **camera.setFixed()** function before using **camera.setPos()** to change the position of the camera. If you wanted to set the camera 25 blocks above the spawn position, you would use:

```
mc.camera.setFixed()
  mc.camera.setPos(0,25,0)
```

To set the camera back to normal, you would use the **camera.setNormal()** function.

```
mc.camera.setNormal()
```

Using the camera functions, you could hide a diamond block (57) in the world, then tease the player by changing the camera to show them where it is before challenging them to find it. Try this in a new program:

```
from mcpi import minecraft
from time import sleep
mc = minecraft.Minecraft.create()

mc.postToChat("Here is the diamond
block I have hidden.")
mc.setBlock(100,25,100,57)
mc.camera.setFixed()
mc.camera.setPos(100,30,100)
sleep(10)

mc.postToChat("Go find it!")
mc.camera.setNormal()
```

You could change the program above to drop the diamond block in a random position and use **getHeight()** so the diamond block is always on the top of the world.

MAKE THE WORLD 'READ-ONLY'

Are you fed up with Steve having free rein to destroy your beautifully crafted world? Or would you prefer it if lava didn't burn down your creation?

Using the **setting()** function in the API, you can make your world 'immutable' – something which is unable to be changed. Start a new script with:

```
from mcpi import minecraft
mc = minecraft.Minecraft.create()

#make the world read-only
mc.setting("world_immutable", True)
```

Now, the only way you can change the world is through code – any attempt to place or destroy blocks in the game won't work.

> "Would you prefer it if lava didn't burn down your creation?"

You can make your world writable (or mutable) again by making the setting **False**.

```
#make the world writable
mc.setting("world_immutable", False)
```

You can use this setting to create a new script which will pit your building skills against a friend, giving you 1 minute to make the best building you can before turning the world read-only again.

```
from mcpi import minecraft
from time import sleep
mc = minecraft.Minecraft.create()

mc.setting("world_immutable", True)

mc.postToChat("In a moment you will have
1 minute to create the best building.")
sleep(10)
mc.postToChat("Go")

mc.setting("world_immutable", False)
sleep(60)
mc.postToChat("Stop - Who's is the
best?")
mc.setting("world_immutable", True)
```

LEARN THE HEIGHT OF THE WORLD

If you want to code structures to always be 'on top' of the land, you need to know how high the world is – or, put another way, how far the air comes down!

In *Minecraft* the height is the Y coordinate, while X and Z are the horizontal dimensions – if you pass X and Z coordinates to the API function **getHeight()**, it will return the Y coordinate. In a new program type:

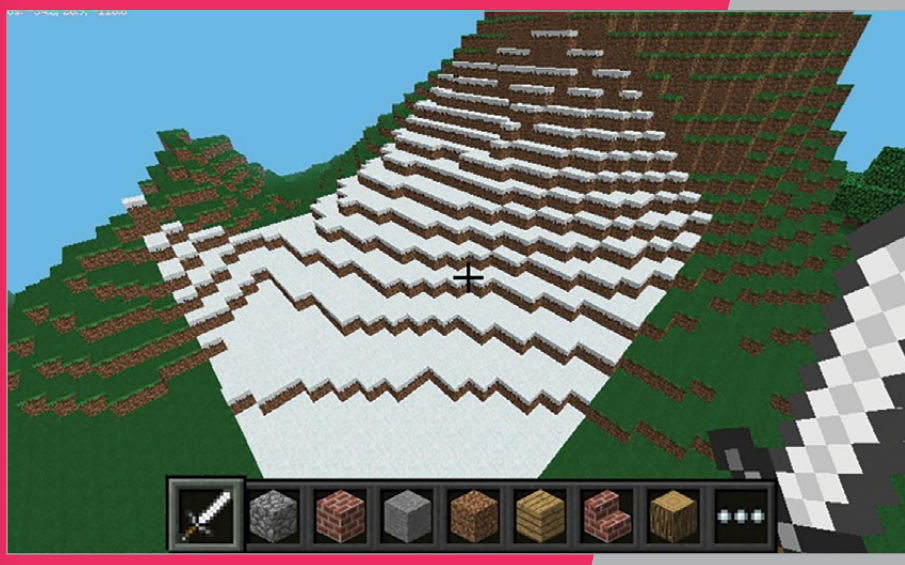
```
from mcpi import minecraft
mc = minecraft.Minecraft.create()

y = mc.getHeight(0,0)

mc.postToChat("Height of the world at spawn
is")

mc.postToChat(y)
```

If you know the height of the world, you can cover the top layer of world in a different type of block by looping through the X and Z coordinates. What about covering the world in snow?

You can do this in a new script by looping through the coordinates around your player, finding the height for that position and setting the block to snow (78).

```
from mcpi import minecraft
mc = minecraft.Minecraft.create()

pos = mc.player.getTilePos()

for x in range(pos.x, pos.x + 10):
    for z in range(pos.z, pos.z + 10):
        y = mc.getHeight(x,z)
        mc.setBlock(x,y,z,78)
```

What other types of block could you cover the world in? Lava perhaps?!



Above Cover *Minecraft* in snow by using the API to find the height of the world

MORE MINECRAFT CODING TIPS & TRICKS

Have you exhausted the Minecraft: Pi basics available from **raspberrypi.org/resources**? Have you completed our Minecraft Pi coding tips? Here are another five tips and mini-programs to experiment with...

You'll Need

- Raspbian
- Minecraft: Pi Edition
- Python 2 editor (IDLE)

BACK UP AND RESTORE MINECRAFT WORLDS

Ever accidentally set off a load of TNT and wished you hadn't? It's at times like these it's a good idea to have a backup of your whole Minecraft world so you can restore it back to normal.

Well, you can. *Minecraft: Pi Edition* stores all of your worlds in a directory on your Raspberry Pi's SD card, so by using the terminal and a few commands, you can find your favourite world and back it up to a file. Open a Terminal window with: **Menu > Accessories > Terminal.**

```
pi@rpi2 ~ $ cd ~/.minecraft/games/com.mojang/minecraftWorlds
pi@rpi2 ~/.minecraft/games/com.mojang/minecraftWorlds $ ls
world  world-  world--
pi@rpi2 ~/.minecraft/games/com.mojang/minecraftWorlds $
  tar czf world--backup.tar.gz world--
pi@rpi2 ~/.minecraft/games/com.mojang/minecraftWorlds $ ls
world  world-  world--  world--backup.tar.gz
pi@rpi2 ~/.minecraft/games/com.mojang/minecraftWorlds $
  tar xzf world--backup.tar.gz
pi@rpi2 ~/.minecraft/games/com.mojang/minecraftWorlds $
```

Below Back up your *Minecraft* worlds as a compressed file

Next, change directory to the **minecraftWorlds** directory using the following command:

```
cd ~/.minecraft/games/com.mojang/minecraftWorlds
```

Each world is saved in its own directory and named the same as what's displayed in the *Minecraft* 'Select World' screen. Use the **ls** terminal command when you're in that directory to see your saved worlds. To make a backup of **world--**, use the **tar** command to create a compressed file:

```
tar czf world--backup.tar.gz world--
```

tar is the command, **czf** tells it to Create a Zipped File, **world-backup.tar.gz** is the name of the backup file, and **world--** is the directory of the world you want to back up.

Now, the next time you want to restore your world, all you have to do is use the following command:

```
tar xzf world--backup.tar.gz
```

Be warned: once entered, there is no going back!

Above Use torches like a pro, automatically placing them around a block

USING TORCHES

If you want to get some light into your *Minecraft* world, you need to create torches. You can do so using the API, but you need to know how to place the torches around the block you want to attach them to.

Torches have their own block type and take up an entire block space in *Minecraft* (even though it looks like they take up a small amount of it), which is why you can't have more than one torch in a block.

> "You could change the code to make a tower of stone"

When torches are created using the **`setBlock()`** API function, it automatically connects the torch to the block which is next to it, to the north, south, east, west, or on top.

Use the following code to create a block of stone above the player and place torches all around it, and one on top.

```
from mcpi import minecraft
from mcpi import block

mc = minecraft.Minecraft.create()

pos = mc.player.getTilePos()

mc.setBlock(pos.x, pos.y + 2, pos.z, block.STONE)

# create torches
# on top
mc.setBlock(pos.x, pos.y + 3, pos.z, block.TORCH)
# to the east
mc.setBlock(pos.x + 1, pos.y + 2, pos.z, block.TORCH)
# to the west
mc.setBlock(pos.x - 1, pos.y + 2, pos.z, block.TORCH)
# to the north
mc.setBlock(pos.x, pos.y + 2, pos.z - 1, block.TORCH)
# to the south
mc.setBlock(pos.x, pos.y + 2, pos.z + 1, block.TORCH)
```

You could change the code to make a tower of stone with torches all around, to provide a beacon to help you find your way.

FIND OUT WHEN BLOCKS ARE HIT

When Steve hits a block by right-clicking with a sword in *Minecraft*, it creates a 'hit event'; you can get this using the API and it'll tell you who hit the block, its position, and what face (i.e. top, bottom, left, right) it was hit on.

You use the function **events.pollBlockHits()** to get a list of events that have occurred since it was last called. You can then loop through events using a **for** loop.

```
from mcpi import minecraft

mc = minecraft.Minecraft.create()

while True:
    hitsList = mc.events.pollBlockHits()
    for hit in hitsList:
        mc.postToChat("A block was hit
(who, position, face)")
        mc.postToChat(hit.entityId)
        mc.postToChat(hit.pos)
        mc.postToChat(hit.face)
```

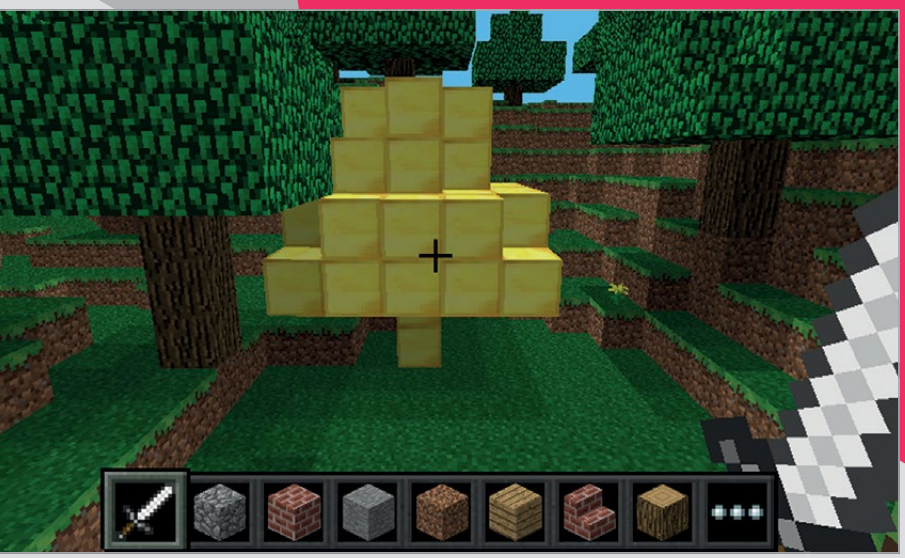
Start up *Minecraft*, run the program above, and experiment with hitting some blocks by holding a sword and right-clicking the block. By using the position and the **getBlock()** function, you can find out the type of block (e.g. stone, dirt, grass) that was hit:

```
        blockType = mc.getBlock(hit.pos)
        mc.postToChat(blockType)
```

Or even better, change the block which was hit, using **setBlock()** to give Steve the Midas touch and make every block he hits turn to gold:

```
        mc.setBlock(hit.pos, 41)
```

Have a think about what other things you can make happen in *Minecraft: Pi* using block hit events.



Below Give Steve the Midas touch and turn blocks into gold!

SAVE AND RESTORE CHECKPOINTS

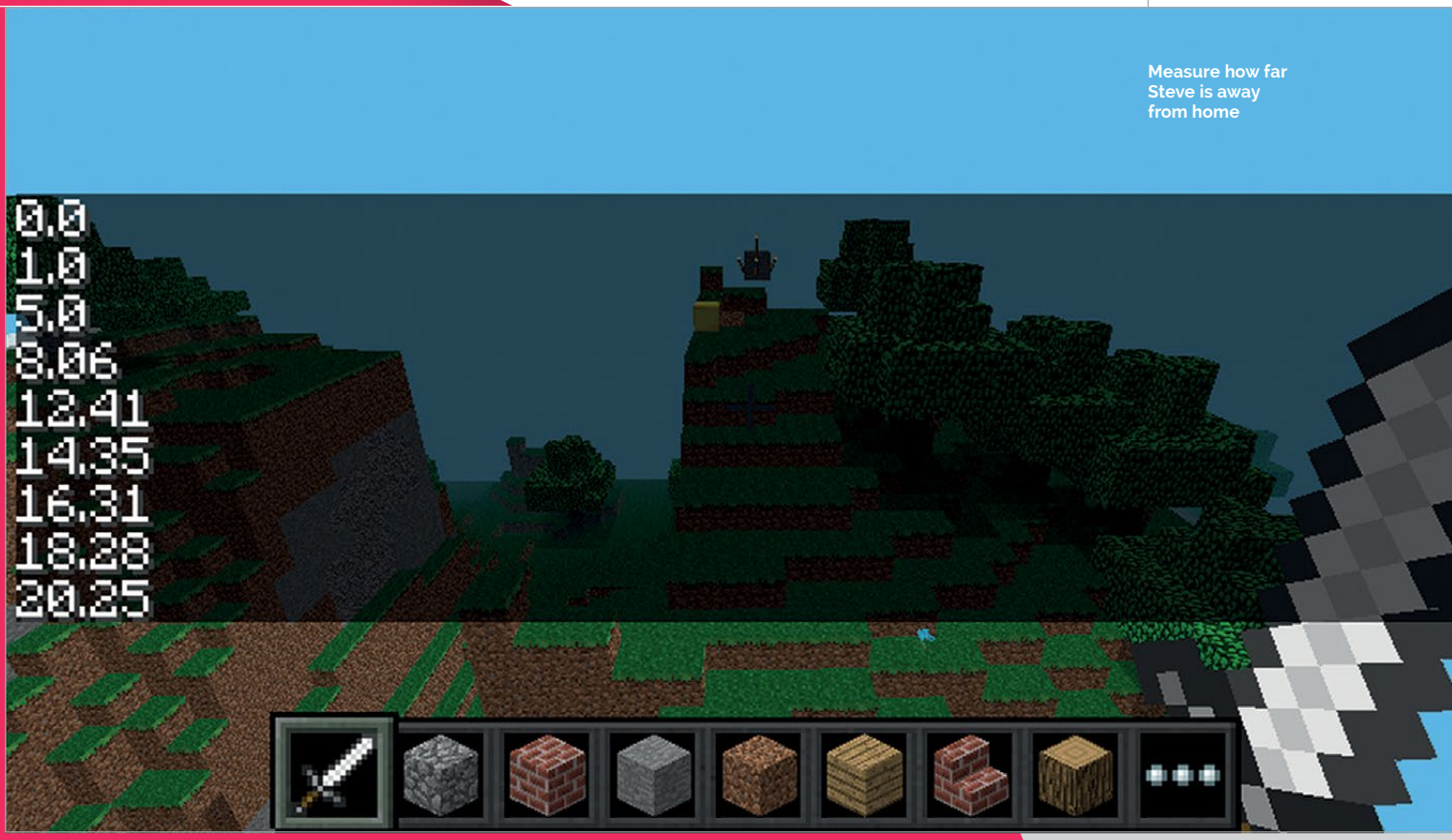
Checkpoints let you create in-game mini-backups so you can undo changes that have been made. You can use the **saveCheckpoint()** API function to make a temporary copy of your world; when you use **restoreCheckpoint()**, this copy is used to put your world back to how it was when you saved the checkpoint.

```
from mcpi import minecraft

mc = minecraft.Minecraft.create()

mc.saveCheckpoint()

mc.restoreCheckpoint()
```

> "Every 30 seconds, your program will save a checkpoint"

You can use the checkpoint functions to create a program which will allow you to 'undo' any unwanted changes you make to your *Minecraft* world. Every 30 seconds, your program will save a checkpoint and if you ever want to go back to it, just hit a block.

```
from mcpi import minecraft
from time import sleep

mc = minecraft.Minecraft.create()

count = 0

while True:
    #every 30 secs save a checkpoint
    if count % 30 == 0:
        mc.saveCheckpoint()
        mc.postToChat("Checkpoint saved")
    count = count + 1
    sleep(1)

    #if a block is hit, restore checkpoint
    if mc.events.pollBlockHits():
        mc.restoreCheckpoint()
        mc.postToChat("Restoring
     checkpoint")
```



Measure how far Steve is away from home

CALCULATING THE DISTANCE BETWEEN TWO BLOCKS

When coding *Minecraft*, it's really useful to know the distance between two blocks – and by using some (fairly) simple maths, we can work it out. This can be used in loads of fun ways, such as a hide and seek game where a diamond block is hidden and Steve is told whether he is getting colder or warmer.
The maths works like this:

01 Calculate the difference between the x, y & z coordinates of the two positions

02 Multiply the difference by itself (its square)

03 Add all the squares together

04 The distance equals the square root of the total above

This program uses this calculation to display how far the player is from where they started. So the further they move away, the greater the distance. See how it works by copying the following code example into IDLE or your favourite text editor (don't forget to save it with the **.py** file extension):

```
from mcpi import minecraft
from math import sqrt
from time import sleep

mc = minecraft.Minecraft.create()

startPos = mc.player.getTilePos()

while True:

    posNow = mc.player.getTilePos()

    xDiff = startPos.x - posNow.x
    yDiff = startPos.y - posNow.y
    zDiff = startPos.z - posNow.z

    xSquare = xDiff * xDiff
    ySquare = yDiff * yDiff
    zSquare = zDiff * zDiff

    total = xSquare + ySquare + zSquare

    distance = sqrt(total)

    mc.postToChat(distance)

    sleep(1)
```

Try changing the program to show the distance between the player and a random diamond block you have created.

Beginner's Guide to CODING

Discover the joy and art of computer programming with your Raspberry Pi

Learning to code is one of the most profoundly life-changing things you can do. This has always been true, but learning to code is increasingly important in the modern world.

The reason the Raspberry Pi was created was to challenge a drop in computer science applications at Cambridge University. Modern computers, and especially games consoles, were fun and powerful, but not easily programmable.

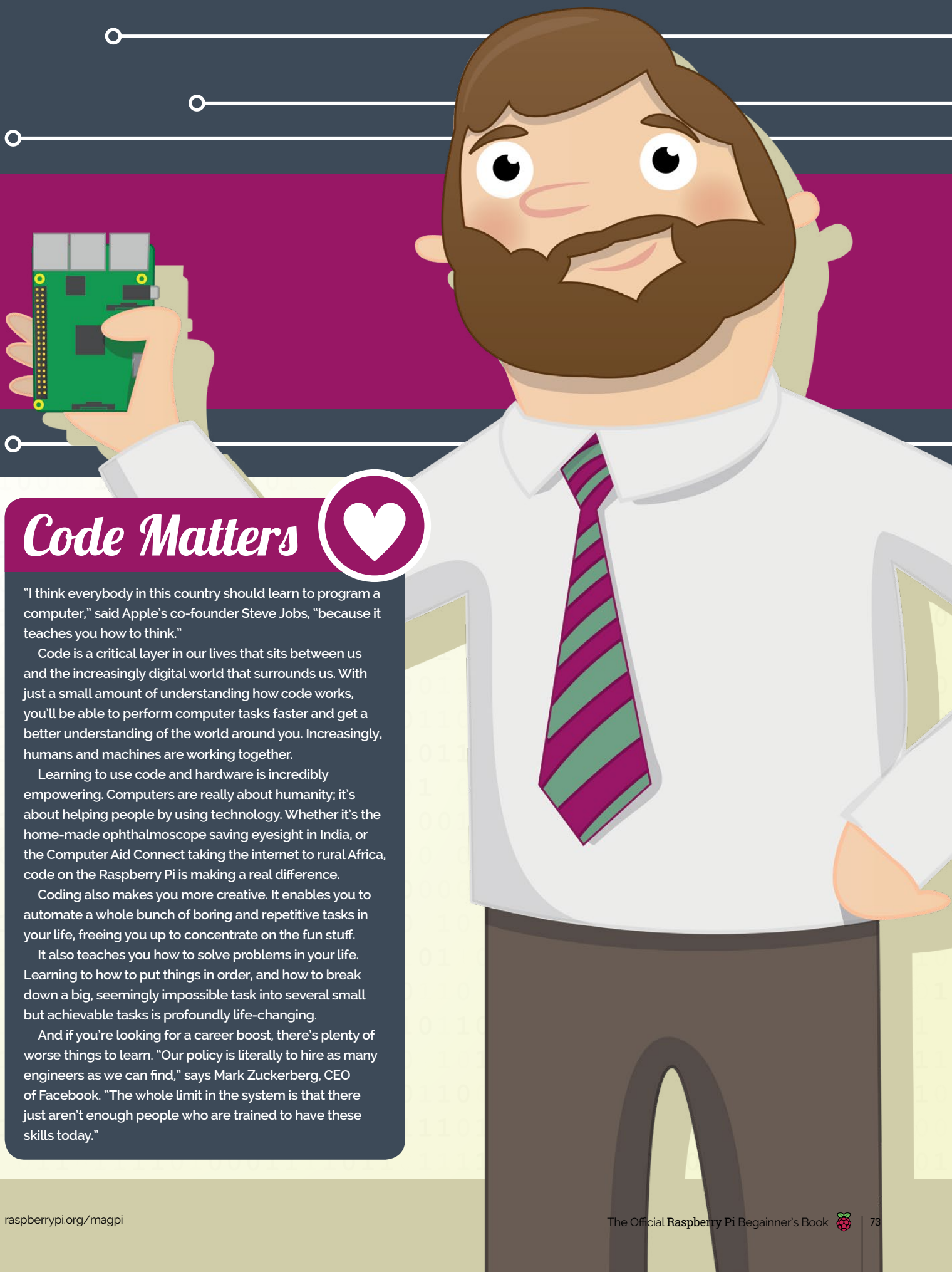
The maker community fell in love with the Raspberry Pi, thanks to its cheap and hackable nature. Building and tinkering are the primary reasons we love Raspberry Pi. Great projects use a combination of hardware and software together.

So, whether you're a hacker learning to make better projects, or a would-be coder looking for a better career, this feature is set to help you on your way.

The good news is that you don't need to be a genius to know coding, just as you don't have to be a genius to read and write. It's actually pretty simple once you learn a few simple concepts like variables, branching, and loops.

Perhaps you're brand-new to coding. Maybe you did a little BASIC in school, or used old languages like Pascal and Fortran. Or maybe you're already knee-deep in projects and just want to learn the language that controls them.

Wherever you're coming from, we're here to walk you through the basic concepts of computer programming. We'll demystify the whole process of code, so you can get a better understanding of what's going on inside your Raspberry Pi.



Code Matters

"I think everybody in this country should learn to program a computer," said Apple's co-founder Steve Jobs, "because it teaches you how to think."

Code is a critical layer in our lives that sits between us and the increasingly digital world that surrounds us. With just a small amount of understanding how code works, you'll be able to perform computer tasks faster and get a better understanding of the world around you. Increasingly, humans and machines are working together.

Learning to use code and hardware is incredibly empowering. Computers are really about humanity; it's about helping people by using technology. Whether it's the home-made ophthalmoscope saving eyesight in India, or the Computer Aid Connect taking the internet to rural Africa, code on the Raspberry Pi is making a real difference.

Coding also makes you more creative. It enables you to automate a whole bunch of boring and repetitive tasks in your life, freeing you up to concentrate on the fun stuff.

It also teaches you how to solve problems in your life. Learning to how to put things in order, and how to break down a big, seemingly impossible task into several small but achievable tasks is profoundly life-changing.

And if you're looking for a career boost, there's plenty of worse things to learn. "Our policy is literally to hire as many engineers as we can find," says Mark Zuckerberg, CEO of Facebook. "The whole limit in the system is that there just aren't enough people who are trained to have these skills today."

What is a Program?

Discover the building blocks of software and learn what goes on inside a program

Which Python?

Python 2 and Python 3 are both commonly used. Python 3 is the future, so we're going with it. Lots of courses still teach Python 2, and it's not a bad idea to take a closer look at the differences between the two: **magpi.cc/2gP6zX3**

Before you go any further, let's look at what a program actually is. The dictionary definition is a "set of instructions that makes a computer do a particular thing."

A computer program is a lot like a recipe. It has a list of ingredients, called 'variables', and a list of instructions, known as 'statements' or 'functions'. You follow the instructions from the recipe one line at a time and end up with a tasty cake – and that's no lie.

The real miracle of computers, however, is that they can do the same thing repeatedly. So you can get a machine to bake a thousand cakes without ever getting tired. A program may contain loops that make it do the same thing over and over again.

Programs also make decisions, and different paths through a program can be taken. Your recipe could make a scrummy chocolate cake or a delightful batch of doughnuts, depending on the variables (the ingredients) it has.

One thing that may surprise you when you begin programming is just how little you need to know to get started. With just a few variables, a smattering of flow, and some functions, you can get a computer doing all the hard work for you.

Inside your Pi

At the heart of your Raspberry Pi are billions of voltage switches known as binary digits (or 'bits' for short). There are 8,589,934,592 of them in its 1GB of RAM, to be exact. All these switches can be set to high or low, which is typically represented as 0 (for low or off) and 1 (for high or on). Everything you see on the screen, hear from the speakers, and type on the keyboard is billions of switches being turned on and off.

Obviously, it's not that easy for humans to talk directly to computers. It's possible to use machine language and send binary instructions directly to a computer, but this isn't where any sane person starts (or ends if they want to remain sane).

Instead, we use a coding language to program. This is written using easy-to-understand functions like `print()`. These are then interpreted into machine language, which the computer understands.

We're going to use Python to learn to code. Python is a truly great programming language. It has a rich syntax that's free from clutter; you don't have to worry about things like curly braces and static typing that crop up in more complicated languages like Java.

With Python, you can just create code, run it, and get things done. Python is one of the languages found most commonly on The Raspberry Pi, so learning it here will help you understand lots of the code used in projects.

Compiled vs Interpreted

Python is an 'interpreted language'. You write the code and then run the program. Under the hood, it's being translated and runs on the fly. Some languages, such as C and Java, are compiled. You write the program, then compile it to get a build file (written in machine code), then you run the build. It's a faff you can do without for now.

IDE and IDLE

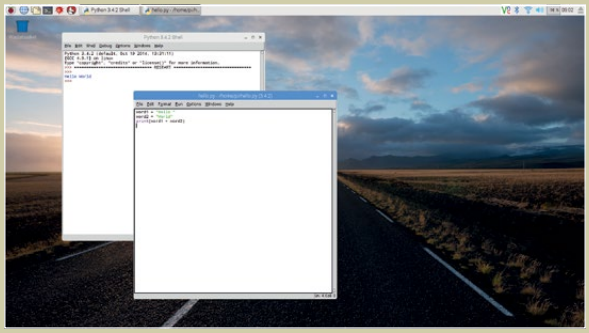
You don't have to write Python programs using a text editor like Leafpad and run them in the terminal. Instead, you can use a neat all-in-one solution, known as an 'IDE' (integrated development environment).

IDEs combine a text editor with program-running functionality. Often, they'll include fancy features like debugging and text completion.

Click **Menu > Programming > Python 3 (IDLE)**, and you'll get a new window called 'Python 3.4.2 Shell:'. This Shell works just like Python on the command line. Enter `print("Hello World")` to see the message.

You can also create programs in a built-in file editor. Choose **File > New File**. Enter this program in the window marked 'Untitled':

```
word1 = "Hello "
word2 = "World"
print(word1 + word2)
```



Above Python IDLE makes it easy to create programs and run them without having to use the command line

Don't forget to include the space after 'Hello'. Choose **File > Save As** and save it as **hello.py**. Now press **F5** on your keyboard to run the program. (Or choose **Run > Run Module**). It'll display 'Hello World' in the Shell.

The advantage of using Python IDLE is that you can inspect the program in the Shell. Enter `word1`, and you'll see 'Hello '. Enter `word2` and you'll see 'World'. This ability to inspect and use the variables in your program makes it a lot easier to experiment with programming and detect bugs (problems in your code).

Why Python?

There are a lot of programming languages out there, and they all offer something special. Python is a great option for beginners. Its syntax (the use of words and symbols) is easy to read. And it scales all the way up to industrial, medical, and scientific purposes, so it's ideal for beginners and experts alike.

Python in the terminal

You don't need to do anything to set up Python on your Raspberry Pi. Open a terminal in Raspbian and enter `python --version`. It will display 'Python 2.7.9'. Enter `python3 --version` and you'll see 'Python 3.4.2'.

We're going to use Python 3 in this feature (see 'Which Python?' boxout). You can open Python 3 in the terminal by just typing `python3`.

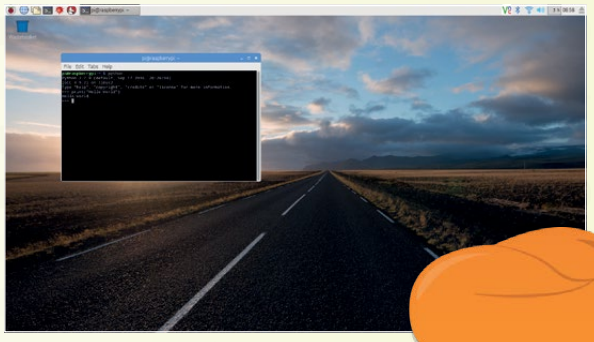
The '$' command-line prompt will be replaced with '>>>'. Here you can enter Python commands directly, just as you would terminal commands.

It's tradition to christen any new language by displaying 'Hello World'. Enter `print("Hello World")` and press **RETURN**. You'll see 'Hello World' outputted on the line below.

Using the Shell is known as Interactive Mode. You can interact directly with the code. It's handy for doing maths; enter `1920 * 1080` to get the answer: 2073600.

Mostly, you create Python programs using a regular text editor and save the files with a '.py' extension. Don't use a word processor like LibreOffice Writer, though – it'll add formatting and mess up the code.

Use a plain text editor like Leafpad (**Menu > Accessories > Text Editor**). Here you can enter your code, save it as a program, and then run the file in the terminal. Enter `python3 yourprogram.py` at the command line to run a program.



Left Python comes pre-installed in the Raspbian operating system and you can use it at the command line

Variables

Variables are all-purpose containers that you use to store data and objects

Python types

Python has five standard data types:

- Number
- String
- List
- Tuple
- Dictionary

Foo bar?

You'll come across 'foo' and 'bar' a lot when learning to code. These are dummy placeholders and don't mean anything. They could be zig and zag or bim and bam. Nobody's quite sure, but it might be related to the expression 'fubar' from the Vietnamese war.

If you've created a science project or experiment, you may have come across variables. In science, a variable is any factor that you can control, change or measure.

In computer programming, variables are used to store things in your program. They could be names, numbers, labels, and tags: all the stuff your program needs.

In Python, you write the name of a variable then a single equals sign and the word, number or object you want to put in it.

Enter this code directly into the Shell:

```
foo = 1
bar = 2
```

Remember: the variable name is on the left, and the thing it contains is on the right. Imagine you've got two plastic cups, and you've scrawled 'foo' on the first and 'bar' on the second. You put a number 1 in foo and a number 2 in bar.

If you ever want to get the number again, you just look in the cup. You do this in Python by just using the variable name:

```
foo
bar
```

You can also print out variables by passing them into a **print** function:

```
print(foo)
print(bar)
```

Variables can also be used to contain 'strings'. These are groups of letters (and other characters) that form words, phrases or other text.

Creating a string variable in Python is pretty much the same as creating an integer, except you surround the text with single (**' '**) or double (**" "**) quotes.

Using double quotes makes it easier to include apostrophes, such as **print("Don't worry. Be Happy")**. This line would break after 'Don' if you used single quotes – **print('Don't worry, be happy')** – so use double quotes for now.

Why variables count

Variables make it much easier to change parts of your code. Say you've got an excellent coding job at Nursery Rhymes Inc and you've written a classic:

```
print("Polly put the kettle on")
print("Polly put the kettle on")
print("Polly put the kettle on")
print("We'll all have tea")
```

The head of marketing comes in and says "our data shows that Polly isn't trending with the millennial demographic." You say "Huh!" and he barks "Change Polly to Dolly."

You now have to go through and change the variable in all three lines. What a downer! But what if you'd written thousands of lines of code and they all needed to change? You'd be there all week.

With variables, you define the variable once and then use it in your code. Then it's ready for changing

at any time:

```
name = "Polly"

print(name + " put the kettle on)
print(name + " put the kettle on)
print(name + " put the kettle on)
print("We'll all have tea")
```

This code prints out the same classic nursery rhyme. But if you want to change the name of our character, you only have to change it in one place:

```
name = "Dolly"
```

…and the poem will update on every line.

What's your type?

When you create a variable in Python, it's automatically assigned a type based on what it is. You can check this using the **type()** function. In the shell interface, enter:

```
foo = "Ten"
bar = 10
```

Now use the **type()** function to check the type of each variable:

```
type(foo)
type(bar)
```

It will say **<class 'str'>** for **foo**, and **<class 'int'>** for **bar**. This concept is important, because different types work together in a variety of ways, and they don't always play nicely together.

For example, if you add together two strings they are combined:

```
name = "Harry"
job = "Wizard"
print("Yer a " + job + " " + name)
```

This prints the message "Yer a Wizard Harry". The strings are concatenated (that's a fancy programming term for 'joined up'). Numbers, though, work completely differently. Let's try a bit of maths:

```
number1 = 6
number2 = 9

print(number1 + number2)
```

Instead of concatenating 6 and 9 together to give 69, Python performs a bit of maths, and you get the answer '15'.

Type casting

So what happens when you want to add a string and an integer together?

```
name = "Ben"
number = 10
print(name + number)
```

You'll get an error message: 'TypeError: Can't convert 'int' object to str implicitly'. This error is because Python can't add together a string and an integer, because they work differently. Ah, but not so fast! You can multiply strings and integers:

```
print(name * number)
```

It'll print 'Ben' ten times: you'll get 'BenBenBenBenBenBenBenBenBenBen'.

If you want to print out 'Ben10', you'll need to convert the integer to a string. You do this using a **str()** function and putting the integer inside the brackets. Here we do that, and store the result in a new variable called **number_as_string**:

```
number_as_string = str(number)
print(name + number_as_string)
```

This code will print out the name 'Ben10'. This concept is known as 'type casting': converting a variable from one type to another.

You can also cast strings into integers using the **int()** function. This is particularly useful when you use **input()** to get a number from the user; the input is stored as a string. Let's create a program that asks for a number and exponent and raises the number to the power of the exponent (using the '**' symbol):

```
number = input("Enter a number: ")
exponent = input("Enter an exponent: ")
result = int(number) ** int(exponent)
```

Our first two variables, **number** and **exponent**, are strings, while our third, **result**, is an integer. We could just print out the result:

```
print(result)
```

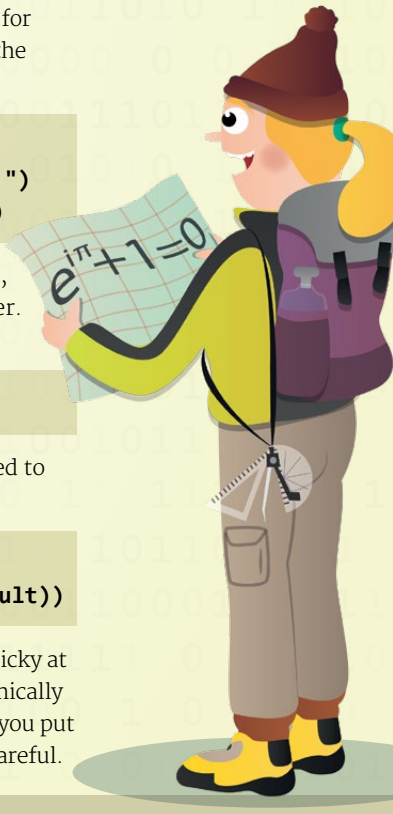
But if we wanted to include a message, we need to type cast **result** to a string:

```
print(number + " raised to the power
      " + exponent + " is " + str(result))
```

Variables, types, and type casting can be a bit tricky at first. Python is a lot easier to use because it dynamically changes the type of a variable to match the thing you put in it. However, it does mean you have to be a bit careful.

What to call a variable?

Variable names should be lower-case words separated by an underscore '_'. They can include numbers, but must start with a letter. You can call variables pretty much anything, but there's a small list of reserved keywords you should avoid (**magpi.cc/2h7MH1y**). It's a good idea to call them something that will be obvious when you use them in your program, like 'student_name' or 'person_age'.



Controlling flow with While & For

Get your program to do all the hard work with while and for loops

Comparison operators

These comparison operators are commonly used in conditions to determine if something is True or False:

==	equal
!=	not equal
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
<>	less than or greater than

Computers are great because they don't mind doing the same stuff over and over again. Their hard-working nature makes computers ideal for doing grunt work.

When looking at variables earlier, we printed out this nursery rhyme:

```
print("Polly put the kettle on")
print("Polly put the kettle on")
print("Polly put the kettle on")
print("We'll all have tea")
```

We didn't like the repetition of Polly, so we replaced it with a variable. But this code is foolish in another way: you have to write out the same **print** line three times.

We're going to use a loop to get rid of the repetition. The first loop we're going to look at is a 'while loop'. In Python 3 IDLE, create a new file and save it as **polly.py**; enter the code from the top of the next page.

We start with two variables:

```
name = "Polly"
counter = 0
```

Then we use the **while** statement followed by a condition: **counter < 3**.

On the next line down, you press the space bar four times to indent the code. Don't press the **TAB** key (see 'Tabs or spaces?' boxout).

```
while counter < 3:
    print(name + " put the kettle on")
    counter = counter + 1
```

The **<** symbol stands for 'less than'. It checks if the item on the left is less than the item on the right. In this case, it sees if the variable counter (which starts at 0) is less than 3. This condition is known as 'True'; if it wasn't, it'd be known as 'False'.

Finally, enter the last line of code:

```
print("We'll all have tea")
```

Save and run the program (press **F5**). It will print 'Polly put the kettle on' three times and then 'We'll all have tea'.

While, condition and indent

There are three things here: the while statement, the condition, and the indented text, organised like this:

```
while condition:
    indent
```

Imagine a three-way chat between all three items in our **polly.py** program:

Tabs or spaces?

There's a massive nerd debate about whether to use spaces or tabs when indenting code. There are valid arguments on both sides, which you can learn in this clip from HBO's comedy *Silicon Valley* (**magpi.cc/2gZde0M**). Use spaces for now. When you're a hardcore coder, you can make the argument for tabs.

While: "Hey Condition! What's your status?"
Condition: "True! The counter is 0. It's less than 3."
Indent: "OK, guys. I'll print out 'Polly put the kettle on' and increase the counter by 1. What's next?"

While: "Hey Condition. What's your status?"
Condition: "True! The counter is now 1."
Indent: "OK. I'm printing out another 'Polly put the kettle on' and increasing the counter by 1."

This goes on till the counter hits 3.

While: "Hey Condition. What's your status?"
Condition: "False! The counter is now 3, which isn't less than 3."
While: "OK guys. We're done!"

The program doesn't run the indented code, but moves to the single **print** at the end: 'We'll all have tea'.

For and lists

The next type of loop is known as 'for'. This is designed to work with lists.

Lists are a type of variable that contain multiple items (strings, numbers, or even other variables). Create a list by putting items inside square brackets:

```
banana_splits = ["Bingo", "Fleegle",
                 "Drooper", "Snorky"]
```

Now enter **banana_splits** in the Shell to view the list. It will display the four names inside the square brackets. You can access each item individually using the variable name and square brackets. Enter:

```
banana_splits[0]
```

...and you'll get 'Bingo'. Lists in Python are zero-indexed; that means the first item in the list is [0]. Here are each of the items. Type them into the Shell to get the names returned:

polly.py

```
name = "Polly"
counter = 0

while counter < 3:
    print(name + " put the kettle on")
    counter = counter + 1

print("We'll all have tea")
```

```
banana_splits[0] # "Bingo"
banana_splits[1] # "Fleegle"
banana_splits[2] # "Drooper"
banana_splits[3] # "Snorky"
```

Zero-indexed lists can be confusing at first. Just remember that you're counting from 0. A for loop makes it easy to iterate over items in a list. Create this program and save it as **splits.py**:

```
banana_splits = ["Bingo", "Fleegle",
                 "Drooper", "Snorky"]

for banana_split in banana_splits:
    print(banana_split)
```

It doesn't matter what you use as the variable in a for loop, as long as you remember to use it in your indented code. You could put:

```
for dude in banana_splits:
    print(dude)
```

It's common to name the list as something plural (such as 'names', 'pages', and 'items') and use the singular version without the 's' for the 'in' variable: 'for name in names', 'for page in pages', and so on.

Infinite loops

You must be careful to change the counter in a while loop, or you'll get an infinite loop. If you delete the line **counter = counter + 1** from our while loop, it will run forever: it never goes above 0, so the indented code runs over and over again. This bug is known as an 'infinite loop' and is a bad thing to have in your programs.

Conditional Branching

Give your programs some brains with conditional branching

Logical operators

You can combine conditions together using logical operators.

and Both operands are true: (a and b) is True

or Any operator is true: (a or b) is True

not Checks if something is false: not (a and b) is True if both a and b are False.

Your programs have been slowly getting more powerful. We've learned to run instructions in procedural order, replaced parts of our program with variables, and looped over the code.

But another important part of programming is called 'conditional branching'. Branching is where a program decides whether to do something or not.

Of course, a program doesn't just decide whether or not to do things on a whim: we use the sturdy world of logic here.

The start of all this is the powerful 'if' statement. It looks similar to a loop, but runs just once. The if statement asks if a condition is True. If it is, then it runs the indented code:

```
if True:
    print("Hello World")
```

Run this program, and it'll display 'Hello World'. Now change the if statement to False:

```
if False:
    print("Hello World")
```

...and nothing will happen.

Of course, you can't just write True and False. Instead, you create a condition which evaluates to True or False; a common one is the equals sign (==). This checks whether both items on either side are the same. Create a new file and enter the code from **password1.py**. This code is a simple program that asks you to enter a password; if you enter the correct password, 'qwerty', it displays 'Welcome'.

Be careful not to confuse the equals logic operator == with the single equals sign =. While the double equals sign checks that both sides are the same, the single equals sign makes both sides the same. Getting == and = mixed up is a common mistake for rookie coders.

What else

After if, the next conditional branch control you need to learn is 'else'. This command is a companion to if and runs as an alternative version. When the if branch is True, it runs; when the if branch is False, the else branch runs.

```
if True:
   print("The first branch ran")
else:
   print("The second branch ran")
```

Run this program and you'll see 'The first branch ran'. But change True to False:

```
if False:
   print("The first branch ran")
else:
   print("The second branch ran")
```

...and you'll see 'The second branch ran'. Let's use this to expand our password program. Enter the code from **password2.py**.

Run the program again. If you get the password correct now, you'll get a welcome message. Otherwise, you'll get an 'incorrect password' message.

Elif

The third branching statement you need to know is 'elif'. This statement stands for 'else if', and sits between the if and else statements. Let's look at an elif statement. Enter this code:

```
if False:
    print("The first block of code ran")
elif True:
    print("The second block of code ran")
else:
    print("The third block of code ran")
```

Run this program and you'll find it skips the first if statement, but runs the elif statement. You'll get 'The second block of code ran'.

The else statement doesn't have a True or False condition; it runs so long as neither the if or elif statements are True. (Note that the else statement here, as always, is optional; you can just have if and elif.)

But what happens if you change both the if and elif conditions to True? Give it a try and see whether just if runs, or elif, or both. Experiment with removing the else statement and play around. It'll help you get the hang of the if, elif, and else statements.

FizzBuzz

We're going to show you a common program used in computer programming interviews. It's a classic called 'FizzBuzz', and it shows that you understand if, else, and elif statements.

First, you need to know about the modulo operator (**%**). This is used to get the remainder from a division and is similar to a divide operator. Take this function:

```
10 / 4 == 2.5
```

If we use a modulo instead, we get this:

```
10 % 4 == 2
```

Modulo turns out to be handy in lots of ways. You can use **% 2** to figure out if a number is odd or even:

```
10 % 2 == 0 # this is even
11 % 2 == 1 # this is odd
```

This program works out if a number is odd or even:

```
number = 10

if number % 2 == 0:
    print("The number is even")
else:
    print("The number is odd")
```

OK – let's move on to FizzBuzz.

password.py

```
password = "qwerty"
attempt = input("Enter password: ")

if attempt == password:
    print("Welcome")
```

password2.py

```
password = "qwerty"
attempt = input("Enter password: ")

if attempt == password:
    print("Welcome")
else:
    print("Incorrect password!")
```

Writing FizzBuzz

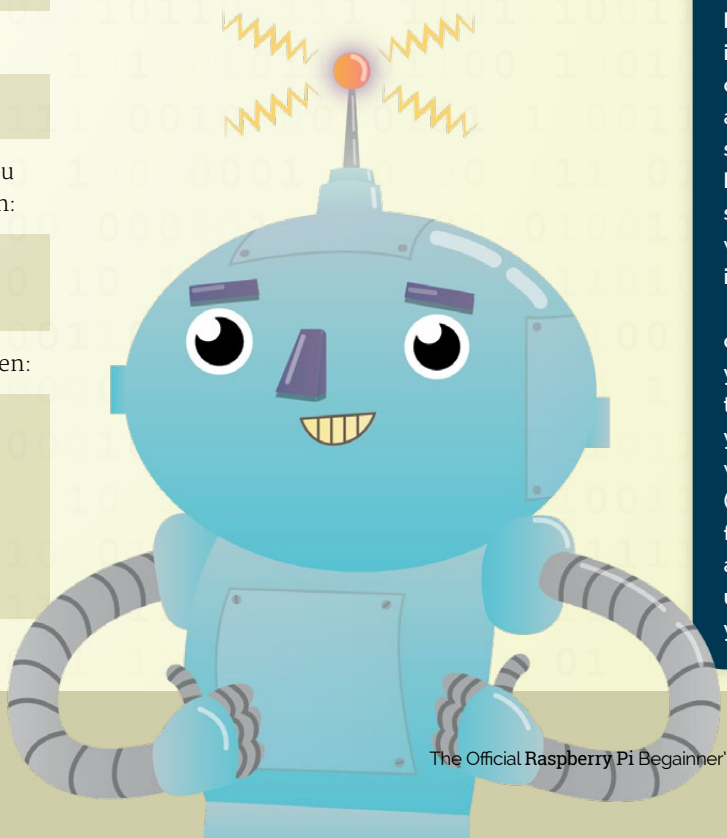
The brief for our FizzBuzz is to print the numbers from 1 to 100. If a number is divisible by three (such as 3, 6, and 9), then you print 'Fizz' instead of the number; if the number is divisible by five, you print 'Buzz' instead.

But if a number is divisible by both 3 and 5, such as the number 15, then you print 'FizzBuzz'.

We're also introducing a new element in FizzBuzz: the 'and' statement. This checks if two conditions are both True: that the number can be divided by both 3 and 5. It only returns True if both conditions are True.

There are three main logical operators: and, or, and not. The first two are relatively straightforward, but the 'not' operator can be more confusing at first. Don't worry about it too much; you'll get the hang of it with practice.

Enter the **fizzbuzz.py** code from page 83 to practise using if, else, and elif elements and logical operators.



Comments

A mark of a good programmer is to use comments in your programs. Comments are used to explain bits of your program to humans. They are completely ignored by the computer.

In Python, you start a comment line with a hash symbol (#). It can be on a line on it own, or it can come right after a line of code. As soon as Python hits the #, it'll stop translating whatever follows into machine code.

Comments help other users to read your program, but they will also help you understand what you're doing (long after you've forgotten). It's a good habit to use comments in your programs.

Creating Functions

Create the building blocks of code and make more robust programs

You've come a long way since your first 'Hello World'. Your programs now check for conditions and loop over themselves.

You're now writing programs that are known as 'Turing complete', named after Alan Turing, the father of computer science and artificial intelligence, who hacked the German Enigma code in WWII.

Now we're going to take things a little further. We're going to introduce you to a form of modularity called functions.

Functions are blocks of code that you write once and can repeat anywhere. It's a little like being able to write a block of text once, and then paste it whenever you need it.

Spotting a function

Python is packed with built-in functions, and you've already been using them in your programs. Commands like **`print()`**, **`len()`**, and **`type()`** are all functions. They're easy to spot: a small command starting with a lower-case letter and followed by a pair of parentheses '()'.

Python documentation

You can browse or download a copy of the Python documentation directly from the Python website at **python.org/doc**. Python has a whole bunch of built-in functions. You can view a list of all the built-in functions on the Python documentation website (**magpi.cc/2gPsGK3**).

Using functions

Let's take a look at a function called **`abs()`**. It stands for 'absolute', and returns the absolute value of any number you pass into it (the bit you pass in is called the 'argument').

An absolute number is the positive of any number, so if you write **`abs(-2)`** you get 2 back. Try this in the Shell:

```
abs(2) # returns 2
abs(-2) # returns 2
```

You can store the returned result as a variable:

```
positive_number = abs(-10)
```

We find it easier to read a function backwards, from right to left. The value is passed into the parentheses, then the function cranks it and returns a new value. This is passed left and stored into the variable.

Defining a function

The great thing about Python is that you don't just use the built-in functions: you get to make your own. These are called 'user-defined functions'.

You create a function using the **`def`** keyword, followed by the function name and parentheses. Inside the parentheses, you list the parameters. These are the same as the arguments, only inside the definition they are called 'parameters'.

```
def function(parameter):
    return parameter
```

Our function here doesn't do anything; it simply accepts a parameter and returns it.

At the end of the function definition is a colon (**:**). The function code is indented by four spaces, just like a loop or if/else branch.

The code inside the indentation runs when you call the function. Functions typically include a **return** statement which passes back an expression.

Working functions

We're going to create a function that prints the lyrics to Happy Birthday.

Type out the **happy_birthday.py** code from the listing, then run it. In the Shell, enter:

```
happy_birthday("Lucy")
```

This function call uses the string 'Lucy' as the argument. This string is passed into the function as the parameter and is then available for use in the indented code inside the function.

Return statements

Many functions don't just run a block of code; they also return something to the function call.

We saw this in **abs()**, which returned the absolute value of a number. This can be stored in a variable.

In fact, we're going to recreate the **abs()** function, so you can see how it's working behind the scenes.

In maths, you invert a positive/negative value by multiplying a negative number by -1, like this:

```
10 * -1 = -10
-10 * -1 = 10
```

We need to create a function that takes a number as a parameter and checks if it's negative. If so, it multiplies it by -1; if it's positive, it simply returns the number. We're going to call our function **absolute()**.

Enter the code in **absolute.py**. When the function hits either of the **return** statements, it returns the value of the number (either on its own or multiplied by -1). It then exits the function.

Run the **absolute.py** code and enter the following in the Shell:

```
absolute(10)
absolute(-10)
```

Our last program listing is a classic known as 'FizzBuzz'; as mentioned on page 81, it will help you to understand if, else, and elif.

You also need to know the modulo operator (**%**) for FizzBuzz. This operator returns the remainder from a division. If you don't know how modulo works, watch this video (**magpi.cc/2h5XNRO**).

Now work through the code in **fizzbuzz.py**.

happy_birthday.py

```
def happy_birthday(name):
    count = 0
    while count < 4:
        if count != 2:
            print("Happy birthday to you")
        else:
            print("Happy birthday dear " + name)
        count += 1
```

absolute.py

```
def absolute(number):
    if number < 0:
        return number * -1
    else:
        return number
```

fizzbuzz.py

```
count = 0
end = 100

while count <= end:
    if count % 5 == 0 and count % 3 == 0:
        print("FizzBuzz")
    elif count % 3 == 0:
        print("Fizz")
    elif count % 5 == 0:
        print("Buzz")
    else:
        print(count)

    count += 1
```

Going further

Here are some resources you will find useful.

GPIO Zero Essentials – magpi.cc/2bA3ZP7
This Essentials guide book explains how the GPIO Zero Python module provides access to a bunch of features. These are used to hook up electronics to your Raspberry Pi via the GPIO pins.

FutureLearn – magpi.cc/2h5Sthf
The Raspberry Pi Foundation has two new online training courses: Teaching Physical Computing with Raspberry Pi and Python, and Teaching Programming in Primary Schools.

Learning Python – magpi.cc/2h2opWC
This tutorial provided by The Raspberry Pi Foundation has files you can download. You download the file, called **intro.py**, using this command in a Terminal: `wget http://goo.gl/0ZDOdX -O intro.py --no-check-certificate`. Open the **intro.py** file in IDLE; all the instructions are in the file.

Importing Code

Stand on the shoulders of giants by importing other programmers' code

Pygame

If you want to learn more about Pygame, check out *Make Games With Python*, our free Essentials Guide to the Pygame module. **magpi.cc/2h2m0vh**

This being the modern world, you're not supposed to do all the work on your own. Instead, you will often stand on the shoulders of other programmers who have done the groundwork for you.

Your programs can import code created by other people using the **import** statement. This enables you to import modules and use their functions – only they're now known as 'methods'.

You import the module at the command line, and then access the functions using dot notation. This is where you list the module, followed by a dot (**.**), then the method.

A common module to use is **math**. This allows you to access lots of maths methods. Open a Python Shell and enter:

```
import math
```

You now have access to all the methods in **math**. You won't notice any difference, but if you type:

```
type(math)
```

...it will say '<class 'module'>'. Let's try out dot notation now. Type **math** followed by a dot and the name of the method (function) you want to use:

```
math.sqrt(16)
```

This gives the square root of 16, which is 4.

Some methods have more than one argument. The **math.pow()** method raises a number to an exponent:

```
math.pow(64,3)
```

This returns 262144.0.

You can also access constant values from a module, which are fixed variables contained in the module. These are like functions/methods, but without the parentheses.

```
math.pi
```

This returns pi to 15 decimal spaces: 3.141592653589793.

```
math.e
```

This returns Euler's number to 15 decimal spaces: 2.718281828459045.

It's also possible to import methods and constants from modules using **from**. This enables you to use them inside your programs without dot notation (like regular functions). For example:

```
from math import pi
from math import e
from math import pow
```

Now, whenever you type **pi** or **e**, you'll get pi and Euler's number. You can also use **pow()** just like a regular function. You can change the name of the function as you import it with **as**:

```
from math import pi as p
```

Now when you enter **p** you'll get pi to 15 decimal spaces. Don't go crazy renaming functions with **as**, but it's common to see some methods and constants imported as single letters.

By creating your own functions, and importing those created by other people in modules, you can vastly improve the capabilities of your programs.

We're going to take everything we've learnt and use it to create a game of Pong; this is one of the world's first videogames.

Write out the code carefully in **pong.py**. Here you'll find variables, functions, loops, and conditional branching: all the stuff we've talked about. Hopefully, you'll now be able to decipher most of this code.

If you're interested in taking Pong further, this program is similar to a version of a Pygame program by Trever Appleton (**magpi.cc/2hgkOUX**). His version has a scorecard and more advanced code. We've kept ours simple so it's easier to start with.

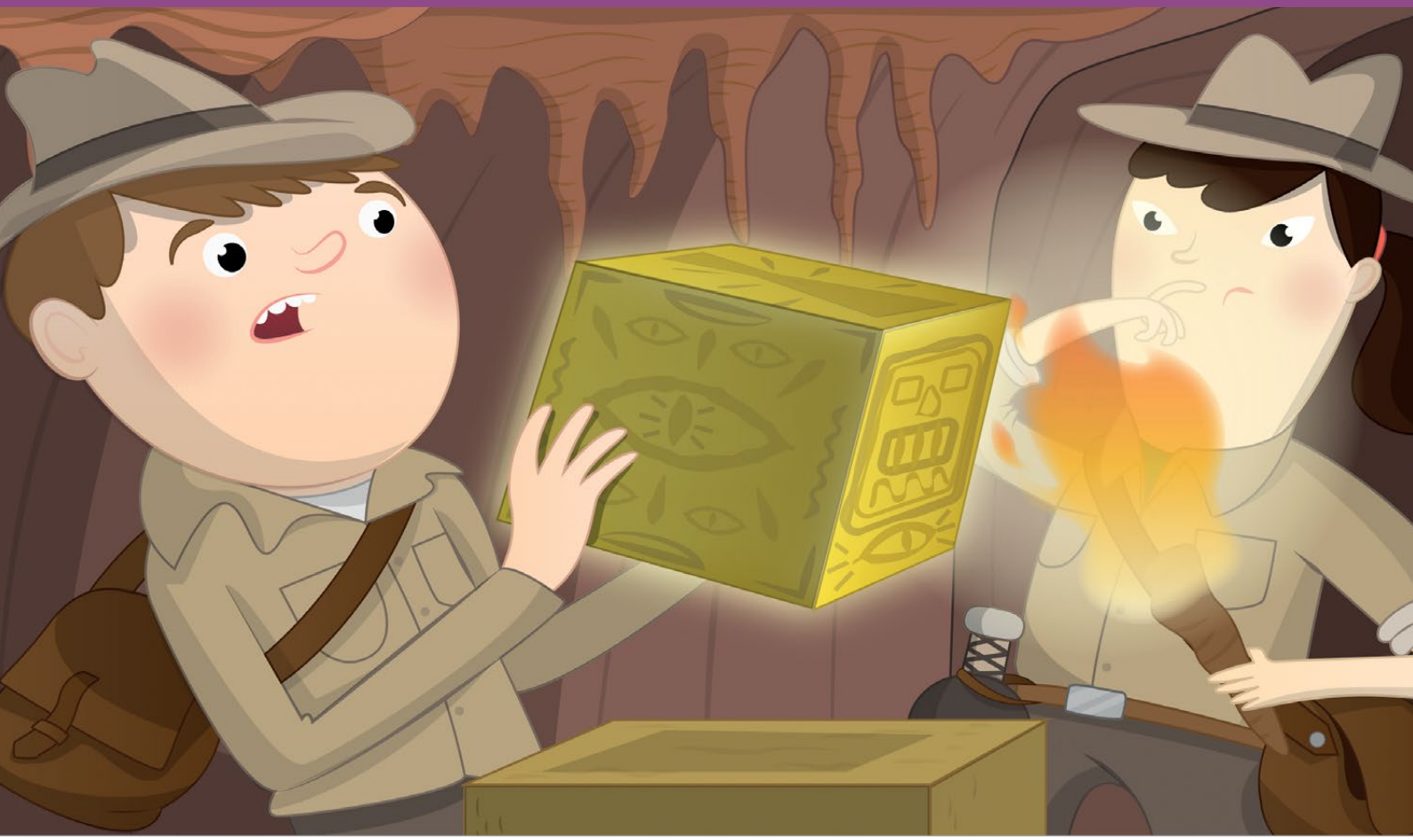
Hopefully this isn't the end of your Python, or programming, journey. There are lots of places you can learn programming from. And you can find more programming resources for you in every issue of *The MagPi* (**magpi.cc**).

Pong.py

```
import pygame, sys
from pygame.locals import *

# Set up game variables
window_width = 400
window_height = 300
line_thickness = 10
paddle_size = 50 # try making this smaller for a harder game
paddle_offset = 20

# Set up colour variables
black = (0  ,0  ,0  ) # variables inside brackets are 'tuples'
white = (255,255,255) # tuples are like lists but the values don't change

# Ball variables (x, y Cartesian coordinates)
# Start position middle of horizontal and vertical arena
ballX = window_width/2 - line_thickness/2
ballY = window_height/2 - line_thickness/2

# Variables to track ball direction
ballDirX = -1 ## -1 = left 1 = right
ballDirY = -1 ## -1 = up 1 = down

# Starting position in middle of game arena
playerOnePosition = (window_height - paddle_size) /2
playerTwoPosition = (window_height - paddle_size) /2

# Create rectangles for ball and paddles
paddle1 = pygame.Rect(paddle_offset,playerOnePosition, line_thickness,paddle_size)
paddle2 = pygame.Rect(window_width - paddle_offset - line_thickness, playerTwoPosition, line_thickness,paddle_size)
ball = pygame.Rect(ballX, ballY, line_thickness, line_thickness)

# Function to draw the arena
def drawArena():
    screen.fill((0,0,0))
    # Draw outline of arena
    pygame.draw.rect(screen, white, ((0,0),(window_width,window_height)), line_thickness*2)
    # Draw centre line
    pygame.draw.line(screen, white, (int(window_width/2)),0),((int(window_width/2)),window_height), (int(line_thickness/4)))

# Function to draw the paddles
def drawPaddle(paddle):
    # Stop the paddle moving too low
    if paddle.bottom > window_height - line_thickness:
        paddle.bottom = window_height- line_thickness
    # Stop the paddle moving too high
    elif paddle.top < line_thickness:
        paddle.top = line_thickness
    # Draws paddle
    pygame.draw.rect(screen, white, paddle)

# Function to draw the ball
def drawBall(ball):
    pygame.draw.rect(screen, white, ball)

# Function to move the ball
def moveBall(ball, ballDirX, ballDirY):
    ball.x += ballDirX
    ball.y += ballDirY
    return ball # returns new position

# Function checks for collision with wall and changes ball direction
def checkEdgeCollision(ball, ballDirX, ballDirY):
    if ball.top == (line_thickness) or ball.bottom == (window_height - line_thickness):
        ballDirY = ballDirY * -1
    if ball.left == (line_thickness) or ball.right == (window_width - line_thickness):
        ballDirX = ballDirX * -1
    return ballDirX, ballDirY # return new direction

# Function checks if ball has hit paddle
def checkHitBall(ball, paddle1, paddle2, ballDirX):
    if ballDirX == -1 and paddle1.right == ball.left and paddle1.top < ball.top and paddle1.bottom > ball.bottom:
        return -1 # return new direction (right)
    elif ballDirX == 1 and paddle2.left == ball.right and paddle2.top < ball.top and paddle2.bottom > ball.bottom:
        return -1 # return new direction (right)
    else:
        return 1 # return new direction (left)

# Function for AI of computer player
def artificialIntelligence(ball, ballDirX, paddle2):
    # Ball is moving away from paddle, move bat to centre
    if ballDirX == -1:
        if paddle2.centery < (window_height/2):
            paddle2.y += 1
        elif paddle2.centery > (window_height/2):
            paddle2.y -= 1
    # Ball moving towards bat, track its movement
    elif ballDirX == 1:
        if paddle2.centery < ball.centery:
            paddle2.y += 1
        else:
            paddle2.y -=1
    return paddle2

# Initialise the window
screen = pygame.display.set_mode((window_width,window_height))
pygame.display.set_caption('Pong') # Displays in the window

# Draw the arena and paddles
drawArena()
drawPaddle(paddle1)
drawPaddle(paddle2)
drawBall(ball)

# Make cursor invisible
pygame.mouse.set_visible(0)

# Main game runs in this loop
while True: # infinite loop. Press Ctrl-C to quit game
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            sys.exit()
        # Mouse movement
        elif event.type == MOUSEMOTION:
            mousex, mousey = event.pos
            paddle1.y = mousey

    drawArena()
    drawPaddle(paddle1)
    drawPaddle(paddle2)
    drawBall(ball)

    ball = moveBall(ball, ballDirX, ballDirY)
    ballDirX, ballDirY = checkEdgeCollision(ball, ballDirX, ballDirY)
    ballDirX = ballDirX * checkHitBall(ball, paddle1, paddle2, ballDirX)
    paddle2 = artificialIntelligence (ball, ballDirX, paddle2)
    pygame.display.update()

```

UNDERSTAND OBJECT-ORIENTED PROGRAMMING

Get your head around OOP by using Scratch and Python to create the same programs



BEYOND PROCEDURE

When you first start programming, you'll begin by writing procedural code.

In good old-fashioned procedural programming, you typically create all your variables at the start of a program. Then you make some function definitions (these are blocks of reusable code).

We looked at procedural programming in Beginner's guide to coding (page 72).

OOP takes all the building blocks of procedural programming – variables, functions, loops, conditions – but bundles them into self-contained blocks.

Most coders create procedural scripts that import objects (from modules and packages). So you're using objects without even realising it.

OOP concepts are found in almost all modern programming languages, including Python and Java.

In the modern world, almost all of the code you will encounter is created in a style called 'object-oriented programming', or OOP for short.

If you grew up with OOP, it is the obvious way to create computer programs.

In OOP, the code is used to create objects. These represent real-world things: a dog, a chair, or the wheels on a car.

Objects contain both the variables that make that thing: a person's height, age, or a name for example. They also contain the functions that object can perform: a dog can jump, or walk, or run; a wheel can rotate.

OOP bundles both the variables and functions together. This style makes it super-easy to cut and paste the code from one program to another. You don't even need to cut and paste, in fact: you use import statements to get functions and the variables they need.

With OOP, you don't need to create an object for a dog: you just find one somebody else has made and import it to your program.

Importing knowledge

At the start of most programs, you'll find a bunch of import statements. These are used to paste in code which has been created by other people.

OOP isn't perfect. It can be accused of overkill. "The problem with object-oriented languages is they've got all this implicit environment that they carry around with them," says Joe Armstrong, creator of Erlang. "You wanted a banana but what you got was a gorilla holding the banana, and the entire jungle."

There's also a whole bunch of decorative terminology surrounding OOP. You'll encounter lots of strange words like 'encapsulation' and 'instantiation'. These make the concept appear much more complicated than it is, and can also be rather off-putting to newcomers.

So OOP is a bit wordy and lends itself to navel-gazing. Many makers, hackers, and coders struggle to understand OOP, and indeed you can get a long way without understanding it.

Young coders, on the other hand, are increasingly introduced to programming via Scratch.

Software like Scratch is included with Raspbian on a Raspberry Pi, and it is designed specifically to teach students OOP stealthily.

In Scratch, objects are called 'Sprites'. They resemble video game characters. The idea is that children brought up on Scratch will inherently feel at home with objects when they migrate to a language like Python.

CREATE BUNCO IN SCRATCH

Create a game in Scratch where players play dice with one another

You'll Need

- Raspbian
- Scratch online

First, we're going to create a game in Scratch. Then we'll recreate it in Python so you can see how it works in both languages.

Our dice game is based on Bunco (**magpi.cc/2hoZNcj**). It's a popular parlour game played in North America.

We've made the rules a little simpler. Each player rolls three dice and counts up the score. The player with the highest score wins.

We need two players, each with their own set of dice. Each player then rolls the dice, looks at their dice, and compares them with the dice of the other player.

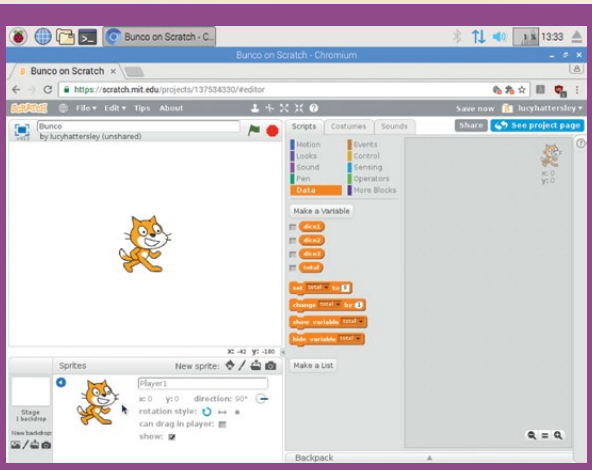
If they have the same score, both call it a draw. If a player spots that their total is higher than the other person's, they shout out "I win!".

This game introduces you to the concept of local variables. Each sprite has three local variables: their own set of dice. They can also look at the variables (or dice) that are local to other sprites.

The opposite of a local variable is a global variable. This is as if both players rolled a single set of dice and shared the result. They'd always draw.

Scratch works slightly differently to Python. In Scratch, you create one sprite and then clone (duplicate) it to create a second sprite. In Python, you create a blueprint for your sprites (known as a 'class') and then stamp out two player objects. We'll come to Python in a bit.

Let's create the dice game in Scratch first...



>STEP-01

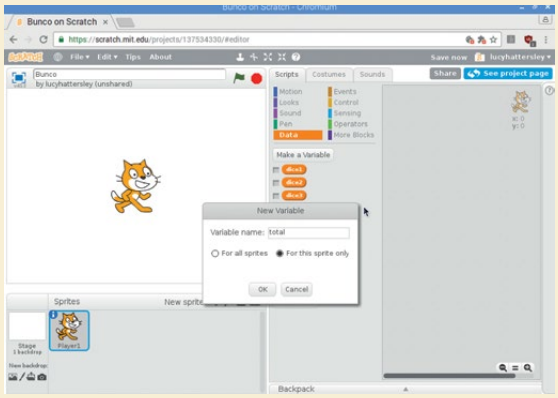
Scratch

Open the web browser and visit **scratch.mit.edu** to open Scratch 2.0. We need the clone features from 2.0, so don't use the Scratch 1.4 app. Log in (or create an account if you're new to Scratch). Create a new project and you'll see a single Scratch Cat sprite on the screen. Click the 'i' symbol next to the sprite in the bottom-left. Change its name to Player1.

>STEP-02

Three dice

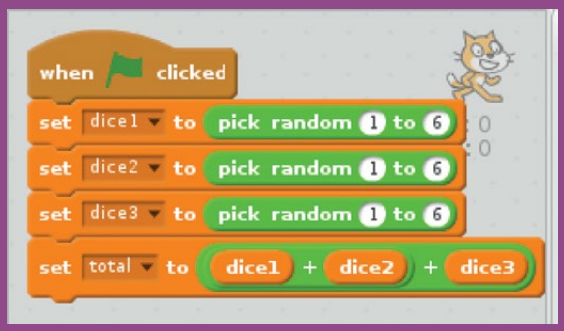
Click on Data and then Make a Variable. Enter **dice1** in the Variable name field and select 'For this sprite only'. Click OK and **dice1** appears in the blocks palette. Repeat the process to create **dice2** and **dice3**. Finally, create another variable called **total**. Remember to choose 'For this sprite only' for all three dice and total.



>STEP-03

Throw the dice

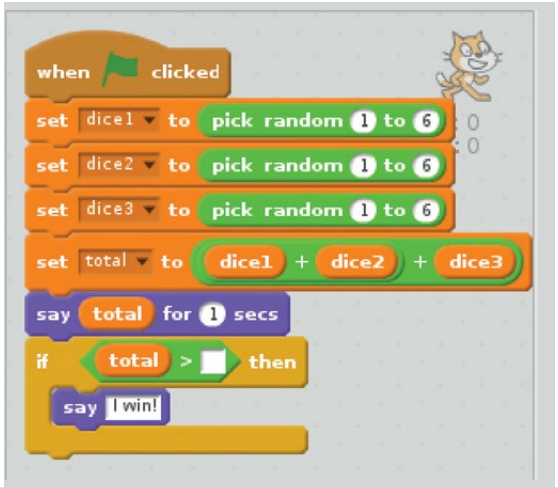
Click on Events and drag a **when green flag clicked** block to the scripts area. Below this you need to add three **set diceN to pick random 1 to 6** blocks. Below these blocks, add **set total to dice1 + dice2 + dice3** (you need to drag one **( ) + ( )** block inside another to add up three blocks.



>STEP-04

Speak

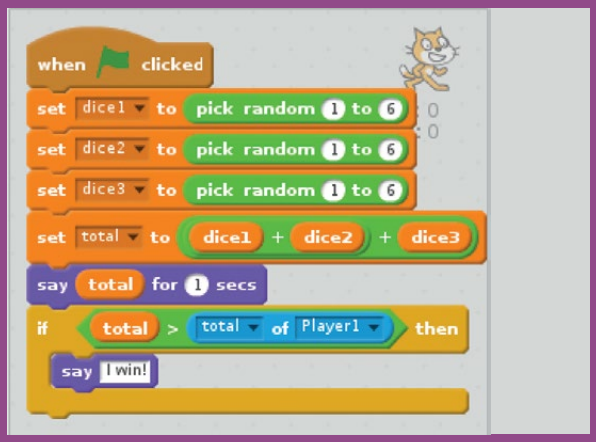
Now drag a **say for** block and attach it to the end of the code. Change it to **say total for 1 secs**. Below that, drag an **if** block. Inside, add a **[ ] > [ ]** (greater-than) block. Drag the **total** variable to the left side of the greater-than block. Drag a **say** block inside the **if** block and change 'Hello!' to 'I win!'.



>STEP-05

Compare scores

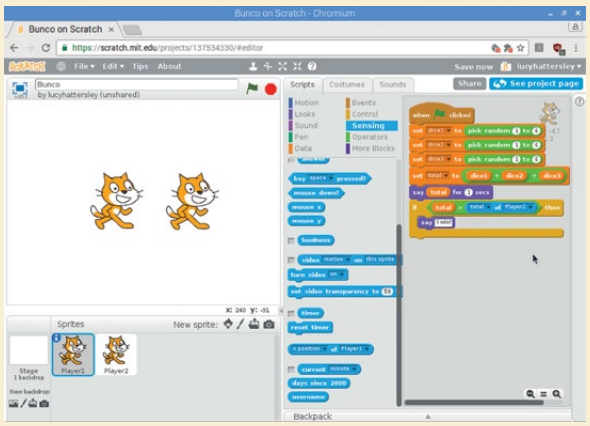
We've only got one sprite so far. But we're about to add another and compare one sprite's total with the other. Choose the Sensing selection of blocks and look for one marked **x-position of Player1**. Change '**x-position**' to total. Drag the block into the correct side of the greater-than block.



>STEP-06

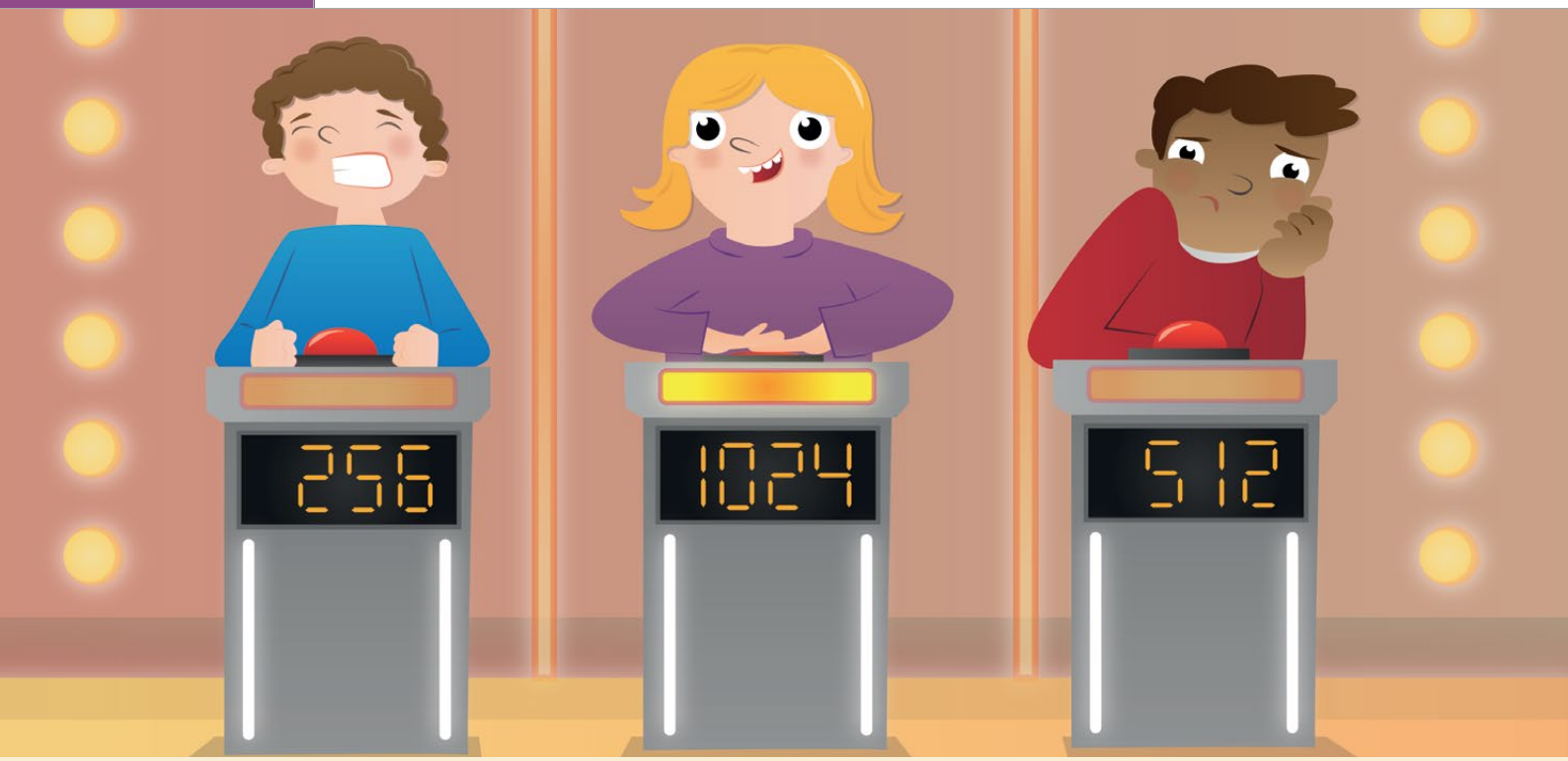
Player 2

Our Player 1 is ready. Now we're going to clone the sprite to create a Player 2. Right-click the sprite and choose Duplicate. The new cat sprite will be automatically called 'Player2'. Click on Player1 in the Sprites window. Change the **total of Player1** block to **total of Player2** (as shown below). Now Player1 compares their total to Player2 (and Player2 is comparing theirs to Player1). Click the green flag to run the program and see which player wins.



SCOPE

The concept of scope is important in object-oriented programming. In Scratch, it's so simple that you may not even notice it. But our two players each have their own **dice1**, **dice2**, and **dice3** variables, plus a **total** variable. These variables are local in scope. When Player1 announces **total**, it's their total. If both sprites accessed the same total, it would be global in scope. Variables in objects are local in scope.



CREATE BUNCO IN PYTHON

CLASS AND INSTANCE

One of the biggest differences between creating sprites in Scratch and objects in Python is that you create an object using a class. This code acts as a blueprint for the object.

In Scratch, you create a sprite and then duplicate it. The second sprite has the same functions as the first. It also has its own set of variables.

In Python (and other programming languages), things work somewhat differently. You don't create an object directly. Instead, you create a blueprint for the objects. This blueprint is known as the 'class'. Don't think of a school, though. Class here means a category of similar items. It's rather like a 'Class M' planet in Star Trek: though different, these are all Earth-like planets.

Once you've created your class, you use it to create objects. These are known as 'instances' or 'object instances'. They all share similar properties. They all have the same variables and functions (called 'methods'). In Scratch, we create one sprite and then duplicate it (to get two sprites). In Python, we create one class definition. We then use this to create two object instances.

Recreate our game of Bunco in Python

Our basic version of Bunco runs just fine in Scratch. Now we're going to recreate it in Python. The translation will help us get a good understanding of how objects work.

First, think about how we could make the game procedurally.

There is a module called **random** that we can import to create random numbers. So we'd need to import that. Then we could create a list for each player. And use the **`randint`** function to add three random numbers between one and six.

We could then use an **`if else`** block with the **`sum()`** function to add up each player's numbers. The player with the highest score wins.

Type out the code from **bunco_procedural.py** to test the program.

There are two problems with this procedural. Bunco is a much more complex game in real life. It is played in six rounds, and players score 21 points

Feature

Language
>PYTHON

DOWNLOAD:
magpi.cc/2jxE6WJ

if they roll all three dice that match the number of the round (three 1s in round 1, or three 2s in round 2, and so on). That's known as rolling a 'bunco'.

We're not going to create all that complexity here. But we are going to add extra types of player. Cheats! One scoundrel has loaded dice; the other rapscallion swaps out one die for a six.

We're then going to play thousands of games and see who wins.

This complexity would be extremely difficult in procedural programming. It requires us to rethink our approach to Bunco. And OOP is the answer.

OOPs upside your head

Instead of creating a list of variables for each player at the start, we're going to create a class called **Player**.

The code in **bunco_oop.py** represents a dice player. We then use it to create two players (see 'Class and Instance').

As with our procedural code, we start by importing the **randint** module.

Now we define our player objects. To do this, we create a class definition. It looks like this:

```
class Player:
```

Inside the class definition is indented code that describes the player object.

Notice that the class name is capitalised and, unlike function definitions, there are no parentheses.

The first thing we need to add is a list to contain the dice. Normally this would be just **dice = []**. But if we wrote it like this:

```
class Player:
    dice = []
```

...we'd have a problem. This code is equivalent to choosing 'For all sprites' in Scratch. Every player created using this code would share a single set of dice and get the same results. We want to use the equivalent of 'For this sprite only'.

To ensure that all our players have their own set of dice, we need to wrap the **dice = []** list inside a quirky function called **__init__()**.

It looks like this:

```
class Player:
    def __init__(self):
        self.dice = []
```

The **__init__()** function runs when you use a class to create an object.

This is known as a 'constructor' or 'initialiser'.

Later on, when we use this **Player** class to create player objects, the **__init__()** code runs each time

bunco_procedural.py

```
import random

player1_dice = []
player2_dice = []

for i in range(3):
  player1_dice.append(random.randint(1,6))
  player2_dice.append(random.randint(1,6))

print("Player 1 rolled" + str(player1_dice))
print("Player 2 rolled" + str(player2_dice))

if sum(player1_dice) == sum(player2_dice):
  print("Draw")
elif sum(player1_dice) > sum(player2_dice):
  print("Player 1 wins")
else:
  print("Player 2 wins")
```

bunco_oop.py

```
from random import randint

class Player:
  def __init__(self):
    self.dice = []

  def roll(self):
    self.dice = [] # clears current dice
    for i in range(3):
      self.dice.append(randint(1,6))

  def get_dice(self):
    return self.dice

player1 = Player()
player2 = Player()

player1.roll()
player2.roll()

print("Player 1 rolled" + str(player1.get_dice()))
print("Player 2 rolled" + str(player2.get_dice()))

if sum(player1.get_dice()) == sum(player2.get_dice()):
  print("Draw!")
elif sum(player1.get_dice()) > sum(player2.get_dice()):
  print("Player 1 wins!")
else:
  print("Player 2 wins!")
```

automatically. It creates a separate set of dice for each of our players.

The 'self' bit also needs explaining. Variables, like our **dice = []** list are normally disposed of when a function ends (or is returned).

So if we just put **dice = []**, the list would be created by **__init__()**, then immediately vanish.

Python gets around this problem using the keyword 'self'. You put 'self' inside the parentheses of the **__init__()**:

```
def __init__(self)
```

Then we use **self**, followed by a dot, to store the variable in this version of the object.

```
self.dice = []
```

You then use **self.** in functions when you want to access or change a variable, by writing **self** inside the parentheses of the function. Like this:

```
def roll(self):
```

The concept can be mind-boggling (it's passing a version of itself into itself). So focus on the practical steps rather than the esoteric theory of how it works:

- Put a special function at the start of a class called **__init__(self)**.
- Put the variables you want to use inside **init**.
- Create the variables with **self.**, like **self.name** or **self.age** or **self.dice = []**.
- Place **self** inside the parentheses of functions that need to access the variables.
- Use **self.** and the name of the variable inside the function to use it.

Got that? Don't worry too much if it seems weird. That's the hardest part and it will get easier with practice.

Now we've got our dice list sorted, what about the other functions?

Methods in the madness

Now that our class has a list for the dice, we need to roll the dice. For that, we'll create a more regular function definition.

```
def roll(self):
  dice = [] # clears any current dice
  for i in range(3):
    self.dice.append(randint(1,6))
```

An object's functions are called 'methods', but they are created in the same way.

There are lots of different types of methods, and you can create whatever you like, but common ones are called 'setters' and 'getters'.

Our **roll** method is a 'setter'. It sets the **dice** list to three random numbers.

What do you think a 'getter' does? That's right. It gets the variables inside the object and returns them. We have just the one getter:

```
def get_dice(self):
  return self.dice
```

Getters and setters seem a bit odd at first. After all, you could just reach into an object and access the variables.

Well, you could, at least in Python, but this is considered a bad thing to do. One of the points of OOP is that objects contain their variables and keep them safe from other objects. So you don't just reach inside an object and access variables.

Instead, you create methods (functions) that set the variables and get them. Then you use these methods to set and get variables.

Now we've created our class definition, we can use it to create objects.

Create away

You create objects just as you would a variable. You use the assigns operator (**=**). We're going to create two dice players:

```
player1 = Player()
player2 = Player()
```

Note that **player1** and **player2** are not called 'variables'. They are called 'object instances'.

We access the object instance's methods using dot notation. That is where you use the name of the object instance, followed by a dot, then the name of the method you want to use.

We created a method, **get_dice()**, that returns the dice stored. We would access this method using dot notation, such as **player1.get_dice()**.

First, we use the **roll** method to get each player object to roll its dice:

```
player1.roll()
player2.roll()
```

The rest of our **bunco_oop.py** program is really very similar to **bunco_procedural.py**. The difference is that here we use the **.get_dice()** method in place of **sum()**.

Inheritance cheats

We've already mentioned that one advantage of OOP is that we can create hundreds or thousands of players with their own set of variables.

There's little point in our program as it is, as the players are all using the same dice and have the same chance of winning. What would happen if we introduced some cheats?

We're going to create a cheat who swaps out one die for a six. The blighter.

Our cheat is not alone. We have another one who uses three loaded dice. They always roll one higher (unless they're already a six).

Both cheats would beat a regular player over a few hundred games. But which of the cheats would win against the other?

It's pretty close. To find out the answer, we'll need to run a simulation that plays hundreds of thousands of games.

We're going to create our cheats using a technique called inheritance. This technique is where you create

> "But which cheat would win against the other?"

a class that takes on all the properties (instance variables and methods) of another class. It also adds a few of its own.

Think of a child inheriting its parents' features. It might get its dad's big nose but grow knobbly knees all by itself.

Our cheats will inherit the same **dice** and **roll** functions as the parent, but they will have cheat functions all of their own.

Our **bunco_module.py** program defines the **Player()** class and two children:

```
class Player:
class Cheat_Swapper(Player):
class Cheat_Loaded_Dice(Player):
```

Objects that inherit from a parent are defined using the same class keyword.

However, the name of the parent is placed inside parentheses of the child.

Our two cheats inherit all the variables and methods (functions) from the parent. So they already have a **dice** list and **roll()** and **get_dice()** methods.

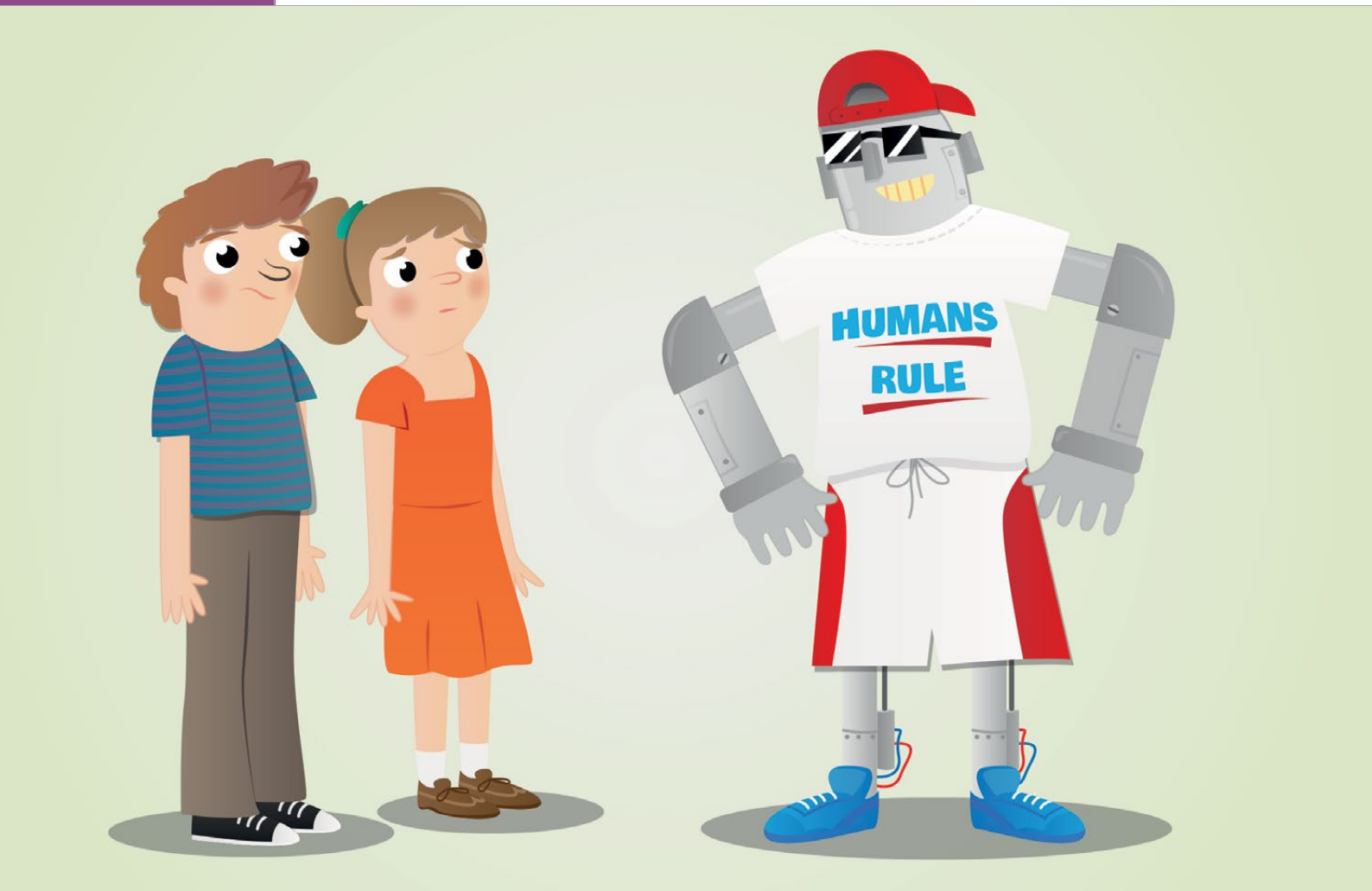
We now give each cheat an additional method, called **cheat**. This is implemented in a different way for each type of cheat.

bunco_module.py

```
from random import randint

class Player:
  def __init__(self):
    self.dice = []

  def roll(self):
    self.dice = [] # clears current dice
    for i in range(3):
      self.dice.append(randint(1,6))

  def get_dice(self):
    return self.dice

class Cheat_Swapper(Player):
  def cheat(self):
    self.dice[-1] = 6

class Cheat_Loaded_Dice(Player):
  def cheat(self):
    i = 0
    while i < len(self.dice):
      if self.dice[i] < 6:
        self.dice[i] += 1
      i += 1
```

bunco_single_test.py

```
from bunco_module import Player
from bunco_module import Cheat_Swapper
from bunco_module import Cheat_Loaded_Dice

cheater1 = Cheat_Swapper()
cheater2 = Cheat_Loaded_Dice()

cheater1.roll()
cheater2.roll()

cheater1.cheat()
cheater2.cheat()

print("Cheater 1 rolled" + str(cheater1.get_dice()))
print("Cheater 2 rolled" + str(cheater2.get_dice()))

if sum(cheater1.get_dice()) == sum(cheater2.get_dice()):
  print("Draw!")

elif sum(cheater1.get_dice()) > sum(cheater2.get_dice()):
  print("Cheater 1 wins!")

else:
  print("Cheater 2 wins!")
```



The **Cheat_Swapper** class definition has a relatively straightforward cheat method:

```
class Cheat_Swapper(Player):
  def cheat(self):
    self.dice[-1] = 6
```

Cheat_Swapper's cheat method finds the last item in the **dice** list and sets it to 6.

Our **Cheat_Loaded_Dice** class definition has a slightly more complicated cheat method:

```
class Cheat_Loaded_Dice(Player):
  def cheat(self):
    i = 0
    while i < len(self.dice):
      if self.dice[i] < 6:
        self.dice[i] += 1
      i += 1
```

This method iterates through the dice in the list, checking whether each die is lower than six. If so, it increases its value by one.

Make sure that you create the code in **bunco_module.py** and save it with the same name.

Notice that this code doesn't have any procedural programming below it. This is because we're going to import it (so you can see what happens when you use **import** in your Python programs).

Now we will create the code that uses these objects in a separate file. Enter the code from the **bunco_single_test.py** listing and make sure you save it in the same directory as **bunco_module.py**.

The first line imports the **Player** class definitions from our **bunco_module.py**.

```
from bunco_module import Player
```

Extra points to you if you spotted that **bunco_module** is listed without the '.py' file extension. This is how you import code from other files into your program.

The **import Player** line pastes in the **class Player** code from **bunco_module.py**. It's as if you had included that code in your program.

Compare this line to the **from random import randint** code at the start of **bunco_module.py**. The idea is the same.

We import the other two class definitions we created:

```
from bunco_module import Cheat_Swapper
from bunco_module import Cheat_Loaded_Dice
```

The rest of the **bunco_single_test.py** code creates the same game as our earlier **bunco_oop.py** program.

Now we create two object instances using the **Cheat_Swapper** and **Cheat_Loaded_Dice** definitions we imported from **bunco_module**:

```
cheater1 = Cheat_Swapper()
cheater2 = Cheat_Loaded_Dice()
```

We then use the **roll()** method. Notice that neither **Cheat_Swapper()** or **Cheat_Loaded_Dice()** has a **roll** method definition. This is a function they both inherit from their parent class, **Player()**:

```
cheater1.roll()
cheater2.roll()
```

Next we call the **cheat()** method from each object instance:

```
cheater1.cheat()
cheater2.cheat()
```

Although each object has a method called **cheat()**, the objects have different implementations. So **cheater1** changes the last die to a 6, and **cheater2** increases each individual die's value by one.

Run the program by pressing **F5** and see which of the players wins. Run it again and you'll get different results. Keep running the program and you'll find it's a pretty close call.

Look inside the folder containing the code and you'll see a new file has appeared called **bunco_module.pyc**. This is a 'compiled file' and is created the first time you run a program that imports code. You don't usually see compiled files because you import code tucked away inside Python on your computer. Don't worry about it. You can't open and make sense of it in a text editor. Delete it if you wish. It'll be recreated when you use **bunco_module.py** in our final program. You can just ignore it for now.

To discover which of the two cheats has the edge, we need to run a simulation. We need to play hundreds of thousands of games and keep track of who wins the games.

Our final program, **bunco_simulation.py**, does just that. This program brings together everything we've learned about OOP. The code in **bunco_simulation.py** creates two cheats and plays 100,000 games. It imports class definitions from our **bunco_module.py** program (so make sure you save it in the same folder).

bunco_simulation.py

```
from bunco_module import *

swapper = Cheat_Swapper()
loaded_dice = Cheat_Loaded_Dice()


swapper_score = 0
loaded_dice_score = 0

number_of_games = 100000
game_number = 0

print("Simulation running")
print("==================")
while game_number < number_of_games:
  swapper.roll()
  loaded_dice.roll()

  swapper.cheat()
  loaded_dice.cheat()

  #Remove # before print statements to see simulation running
  #Simulation takes approximately one hour to run with print
  #statements or ten seconds with print statements
  #commented out

  #print("Cheater 1 rolled" + str(swapper.get_dice()))
  #print("Cheater 2 rolled" + str(loaded_dice.get_dice()))

  if sum(swapper.get_dice()) == sum(loaded_dice.get_dice()):
    #print("Draw!")
    pass

  elif sum(swapper.get_dice()) > sum(loaded_dice.get_dice()):
    #print("Dice swapper wins!")
    swapper_score+= 1

  else:
    #print("Loaded dice wins!")
    loaded_dice_score += 1

  game_number += 1

print("Simulation complete")
print("-------------------")
print("Final scores")
print("------------")
print("Swapper won: " + str(swapper_score))
print("Loaded dice won: " + str(loaded_dice_score))

if swapper_score == loaded_dice_score:
  print("Game was drawn")
elif swapper_score > loaded_dice_score:
  print("Swapper won most games")
else:
  print("Loaded dice won most games")
```

CLONE, COPY & CONTRIBUTE TO CODE WITH GIT

Use Git version control software to work on your own code and contribute to open-source projects. By **Marc Scott**

You'll Need

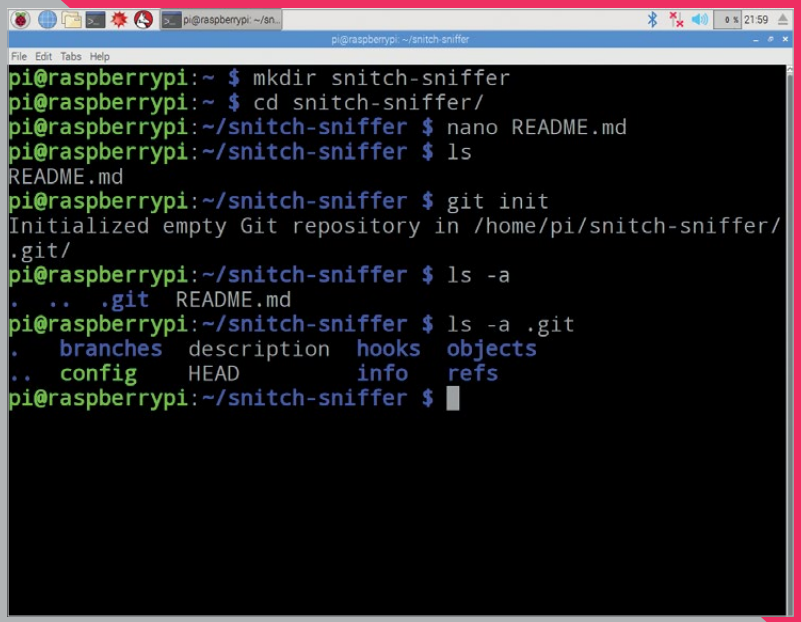
- Raspberry Pi
- Raspbian
- Git

Git is powerful software that lets you clone, copy, and contribute to code projects. Discover how to master Git and you'll be a much better project maker.

If you're working on a Raspberry Pi, then congratulations: Git already is installed in Raspbian by default! If you are on another Linux build, or ever need to install Git on another Linux system, this is all you need:

```
sudo apt install git
```

You're going to be working in a Terminal window for this tutorial, so open it by clicking on the icon on the desktop, or by pressing **CTRL+ALT+T** on the keyboard.



Initialising a directory of code (and other files) adds a hidden .git directory to it; this directory is used to track changes

The first thing to do is to tell Git who you are. This is important, as Git can be used collaboratively by lots of people, so it needs to know who made changes to which files. Use your user name and email address.

```
git config --global user.name "Harry Potter"

git config --global user.email "h.potter@hogwarts.prof"
```

Next, you need to tell Git which text editor you want to use. If you don't have any particularly strong feelings about text editors, then type:

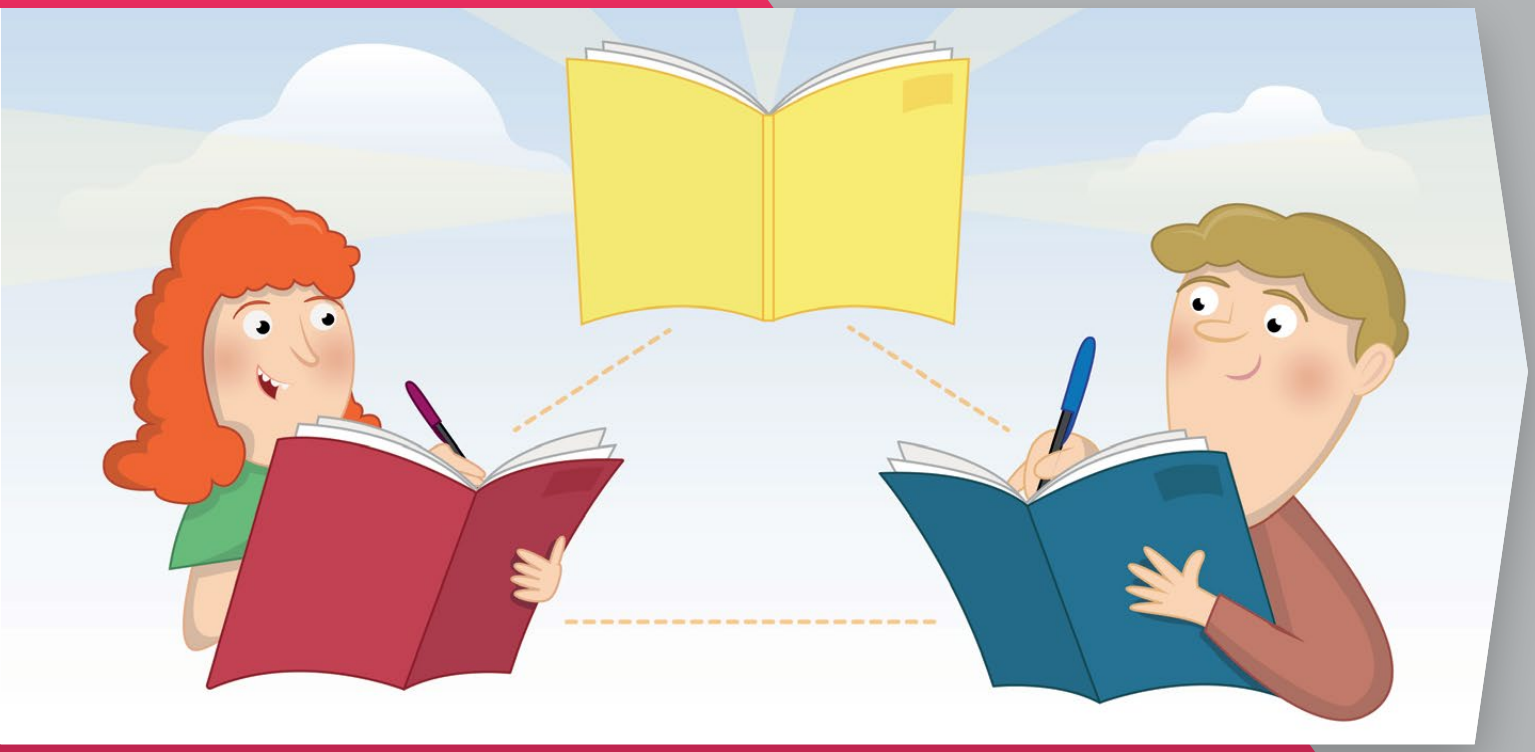
```
git config --global core.editor nano
```

Now you are ready to start using Git to manage your projects.

The magical briefcase

One way to think about Git is to imagine a magical briefcase. You can pull documents out of the briefcase and work on them whenever you like. Once you've finished your work, you can put the documents back into the briefcase, and the case remembers what changes you made to all the documents inside.

What's really clever is that this briefcase can be synchronised with another master briefcase that lives in the clouds. Whenever you like, you can tell your briefcase to copy the contents of all the documents to the master briefcase. If you lose your own briefcase, you don't have to worry, as you can



just get a new one and grab all the documents and writing from the master briefcase.

That's not all, though. The other people working on your project also have magical briefcases, and they also keep their documents synchronised with the master briefcase. This means that you can all work on a project together. If someone else has a better answer to a project question than you do, you can copy their answer from the master briefcase to your document.

Start a project

So you want to start a new project? Maybe it's a special ultrasonic range finder for tracking flying objects in the air. You'll want a directory on your computer for all your files to sit in, so the first thing to do is create that directory.

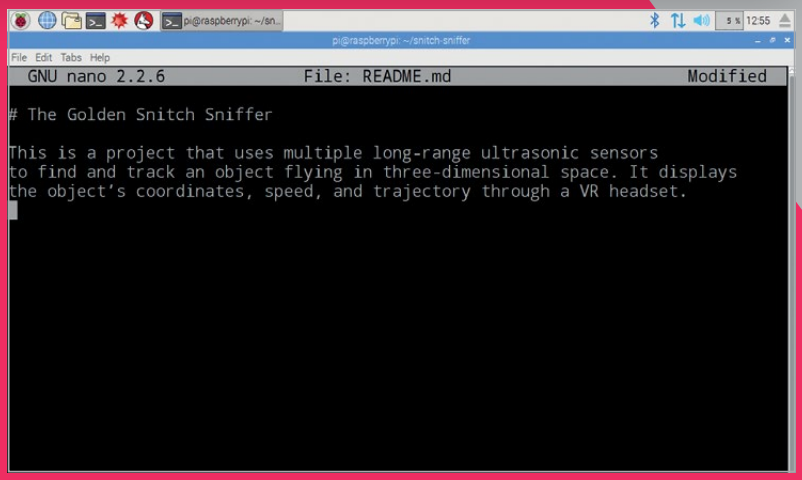
In the Terminal, you can use the **mkdir** (make directory) command to create a new directory.

```
mkdir snitch-sniffer
```

Now you want to go into that directory. You can use the **cd** (change directory) command to do this.

```
cd snitch-sniffer
```

Next, you can create a file that will tell people what the project is about. You can use any text editor to do this, but in this example, nano is used to create a file called **README.md**. The .md extension stands for Markdown, which is a markup language. You can learn more about Markdown on the Daring Fireball website (**magpi.cc/2scx1iu**).



Use nano to edit the README.md text file

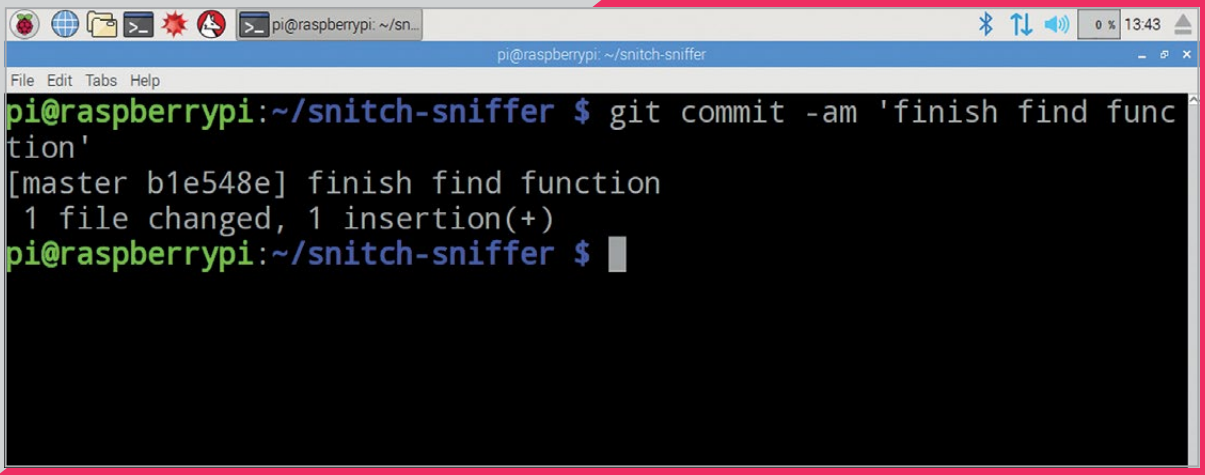
```
nano README.md
```

This command opens up the file in the Terminal. You can now give the file a title and write a short explanation of what your project is about.

```
# The Golden Snitch Sniffer

This is a project that uses multiple long-
range ultrasonic sensors to find and track
an object flying in three-dimensional space.
It displays the object's coordinates,
speed, and trajectory through a VR headset.
```

Pressing **CTRL+X** will cause a save prompt to appear. You can type **Y** to save and then hit **ENTER** to close nano. Your file should have been created and will now be sitting in your directory. You can type **ls** in the Terminal to see a list of files.

Perform a new commit every time you make a change to your file



Create the magic briefcase

At the moment, the directory is just like any other directory on your system. You now need to make the magical briefcase part. This is known as a Git repository, and it takes the form of a hidden directory that keeps track of all the changes to the working directory. Type the following:

```
git init
```

This creates the repository, which from now on will just be called a repo.

If you type **ls** again, nothing will appear to have changed. You can use **ls -a** to see all the hidden files and directories, though.

You should now see something like this in your Terminal window:

```
.  ..  .git  README.md
```

That .git directory is the repo skeleton. You can have a look inside it by typing **ls -a .git**.

This should bring up something like:

```
branches config description HEAD hooks
info objects refs
```

You don't need to worry about this directory at all now. Just know that it is there and that it's tracking all the changes to the parent directory **snitch-sniffer**.

Add your books

So you now have the magic briefcase, but you haven't yet added anything to it. That **README.md** file hasn't been placed into the briefcase yet. You need to tell Git that you want to add the **README.md** file to the repo. To do this, type:

```
git add README.md
```

Sometimes it's easier to just add everything to the repo, though, rather than adding individual files. To do this, you type:

```
git add --all
```

Now Git knows that it needs to keep track of all the changes that happen to the **README.md** file. You can have a look at the status of your repo at any time by typing:

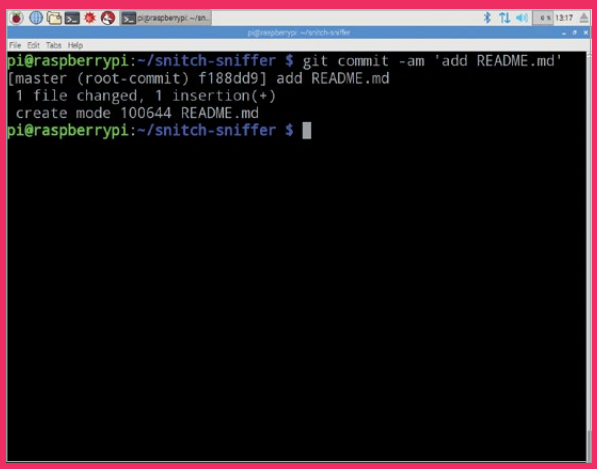
```
git status
```

You should see something like this:

```
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   README.md
```

This is telling you that the **README.md** file has not yet been committed. This means that although Git knows about the file, it doesn't yet have any of the file's contents stored. The simplest way to do a commit is by typing:

```
git commit -am 'add README.md'
```



Add files to the Git repo and commit the changes with a comment

This commits all the changes you have made in the directory to the Git repo, and adds a message saying what you did. The message can be anything really, but it's best to keep it fairly short yet descriptive of what you changed.

Time travel

Now that you have set up your repo, it's time to get on with the project. Create two new files and store them in your **snitch-sniffer** directory:

```
touch snitch-sniffer.py quidditch-rules.json
```

Typing **ls** reveals those files.

```
README.md  quidditch-rules.json  snitch-
sniffer.py
```

The new files need to be added to the Git repo and then committed.

```
git add --all

git commit -am 'add json rules and python
program'
```

Carry on working on your code (enter **nano snitch-sniffer.py** and add some dummy code). Every time you make a change to the file, perform a new commit.

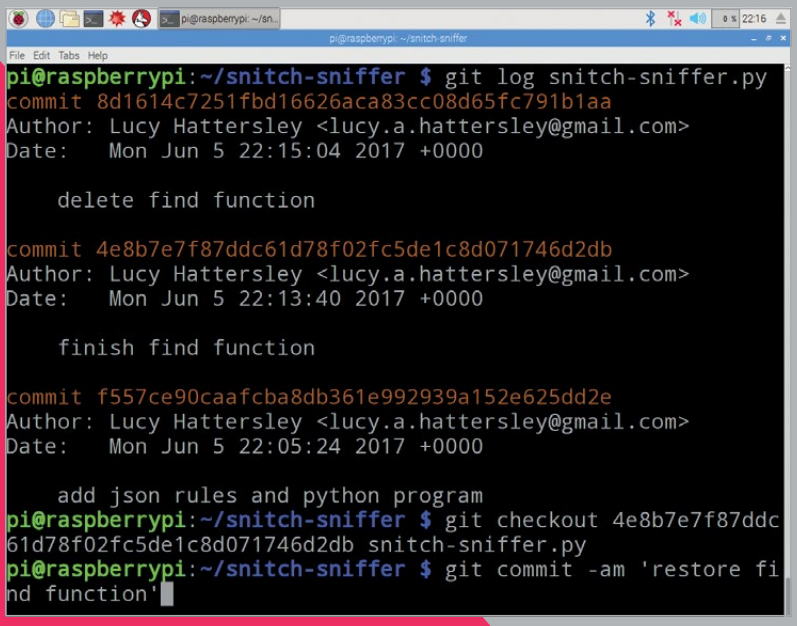
```
git commit -am 'finish find function'
```

Now imagine that you've made a horrible mistake. You've been working for a while, and you've deleted your **find_snitch()** function, and then performed a commit. With Git, it's easy to go back in time and restore an earlier version of any of your files. Let's first look at the commit history of the file.

```
git log snitch-sniffer.py
```

This produces something like this:

```
commit
12c4c693e95438ceadcf3f4fb39c83ce1ade712f
Author: Harry Potter <h.potter@hogwarts.prog>
Date:   Fri Mar 3 20:27:17 2017 +0000

    delete find function

commit
5fd772a292c019a7cf3012b1156685280d4a7d2d
Author: Harry Potter <h.potter@hogwarts.prog>
Date:   Fri Mar 3 20:24:52 2017 +0000

    finish find function
```



A great thing about Git is that it tracks changes so that you can restore an earlier version of your code at any time

```
commit
127545c19794b5fe869dd22d0cf57bf8820c5794
Author: Harry Potter <h.potter@hogwarts.prog>
Date:   Fri Mar 3 20:20:18 2017 +0000

    add json rules and python program
```

You can see that the last commit (the one at the top) was where the function was deleted. Luckily, the commit message has made it easy to see what was done, which is why commit messages are important. However, typing:

```
git log -p snitch-sniffer.py
```

...would have shown the changed contents of the file if the commit message wasn't clear enough.

You can now get back the version of the file from the previous commit. The long string of characters after the word 'commit' is called a hash, and is used by Git to keep track of files. In this case, the commit that needs to be restored is **5fd772a292c019a7cf3012b1156685280d4a7d2d**. Typing the following will get the file back to the way it was:

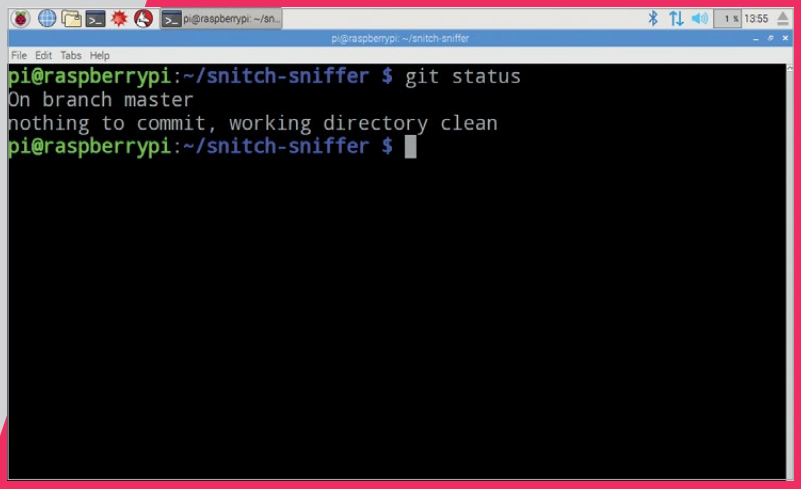
```
git checkout
5fd772a292c019a7cf3012b1156685280d4a7d2d
snitch-sniffer.py
```

The file will be restored, and you can now commit this change.

```
git commit -am 'restore find function'
```

Making major changes

Imagine you're talking to someone about your amazing project, and they have a cool idea for some



You can always check the current status of your repo

changes you could make to improve it. They suggest using lidar rather than ultrasonic sensors. The changes are quite large, though, and you're worried that if you make them, you might break the project.

You could make a copy of the directory and start working on this copy, but to keep using Git, you'd have to make an entirely new repo. This could all get quite confusing. Luckily, Git has a feature called branches. Using a branch allows you to make copies without losing or altering your original work.

First, you can have a look at your repo's current status.

```
git status
```

This should show something like this:

```
On branch master
nothing to commit, working directory clean
```

Now you can make a new branch in the repo, which lets you work on your amazing new adaptation.

```
git checkout -b lidar-version
```

Now **git status** will show you something like this:

```
On branch lidar-version

nothing to commit, working directory clean
```

This tells you that you are on the lidar-version branch. To view all the branches in your repo, you can type **git branch**, which will show something like this:

```
* lidar-version
  master
```

You can now work on the lidar-version branch without altering your master branch. If you try out the new approach and find it doesn't work, you can simply delete the branch using **git branch -D lidar-version**.

However, if it all works well, you can merge the branch back into your master branch. First, you'll need to make sure all your changes are committed and then switch back to the master branch.

```
git checkout master
```

Then you can merge the version into the master branch:

```
git merge lidar-version
```

Warning: you can cause problems with a merge if you're working on two branches at the same time, as Git won't know which changes are the ones you want to keep. For this reason, it's best to work on one branch at a time.

Setting up a Git service

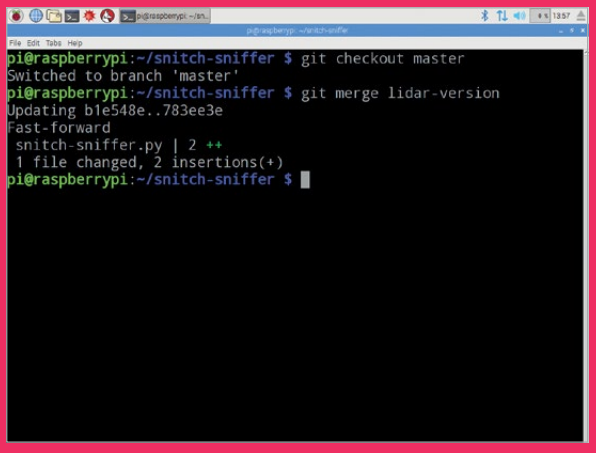
Now that you know how to do the basics in Git, it is time to learn how to use it to its full potential: sharing your work and collaborating with others.

There are lots of services that will host your Git repo for you, free of charge. GitLab is one such service, and BitBucket is another. In this resource, you are going to be using GitHub (**github.com**), which is one of the more popular services.

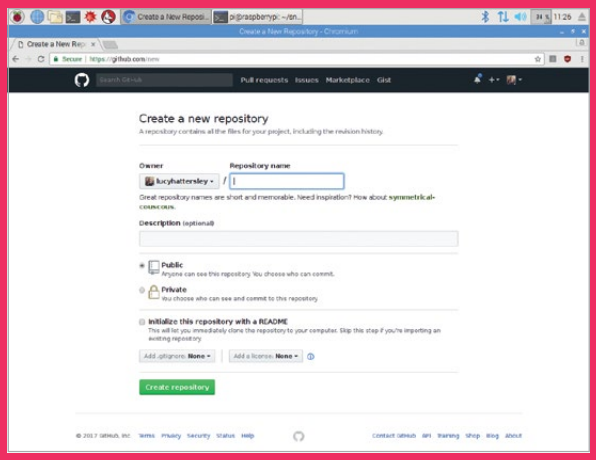
The first thing to do is to register for an account on GitHub: just choose the free plan. Now that you have an account, you can create a **snitch-sniffer** repo on GitHub. Click the New Repository button to the right of Your Repositories. Give the repo a name (enter **snitch-sniffer** in the Repository Name field, and a short description, such as '3D tracking of objects in space' and click Create Repository). This will bring up a page of instructions. As you already have a repo ready to push to GitHub, all you need to do is make sure you are in your project directory in a Terminal window, and type:

```
git remote add origin https://github.com/
harrypotter/scratch-sniffer.git
```

...and then:



Merging puts the changes you have made into your master branch



Use the GitHub website to create a repository (repo)

```
git push -u origin master
```

You will need to enter **yes** in the command line to accept the authentication token, then your GitHub user name and password. If you look on GitHub, you should now be able to see your repo, along with the displayed **README.md** file that you wrote.

Any time you make changes to your project, and you want to push them up to GitHub, you can just type:

```
git push origin master
```

You may see an RSA fingerprint warning. Enter **yes**. If you are working on a different branch, you would type:

```
git push origin <branch-name>
```

Collaborative working

The true power of services like GitHub becomes apparent when you start working with other people. GitHub allows other people to make their own copies of your projects, or you to make copies of theirs. Either of you can then make improvements to the project and then push the improvements up to GitHub for everyone to share.

This tutorial started as a GitHub repo. You can find it at **magpi.cc/2rM1cow**. That means that if you found a mistake in the resource, or if you just wanted to make some improvements, you can. There are two main ways to get involved with other people's projects: issues and pull requests.

GitHub issues

The copy editors at Raspberry Pi are pretty amazing, so the chances of you spotting a typo in this tutorial are pretty slim. You might spot a mistake in some of the code, though, and that's where you can help out. Take this bit of code for example:

```
print('Hello World!)
```

Let's see how you could help fix this error. Head on over to **magpi.cc/2rM1cow**, and make sure you are logged in. Find the Issues tab. You can now create a new issue, and give a description. Once that's completed, the maintainers of the repo will be able to reply to you, and close the issue once it's fixed.

Pull requests

Issues are great, but if you want to help out even more, the project maintainers are usually very happy for you to fix or improve projects yourself. To do this, you need to make your own copy of the repo so that you can work on it.

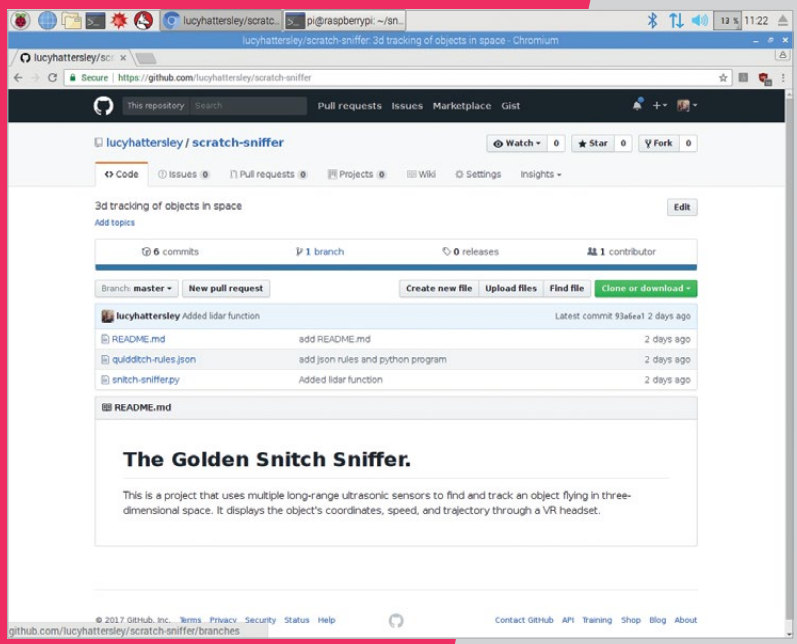
On the project's main page, find the Fork button and click it. You will now have a copy of the repo. You should see a Clone or Download button. Clicking this will reveal the uniform resource identifier (URI) of the repo. Using the Terminal, you can clone the repo to your computer with **git clone**:

```
git clone https://github.com/HelpfulUser/
getting-started-with-git.git
```

All the files and directories will now be on your computer. Go ahead and make the changes you want, then commit them and push them back up to GitHub, just as you would normally do. Here your commit message is particularly important, as it will explain to the resource's original owner the changes you have made.

You can now head back over to GitHub. Find the button that says New Pull Request. Click the button and then click on the Create Pull Request button. Your commit message will be there, but you can change it and even add a more detailed description if you like.

Once you're happy, click the Create Pull Request button. The maintainer of the repo will then be able to see your pull request. They can then choose to merge it into their repo or close it.



The snitch-sniffer repo, showing branches, commits, and other information

CODE WITH THE SENSE HAT EMULATOR

Practise coding for the Sense HAT hardware using the Sense HAT emulator built into Raspbian

You'll Need

- Raspberry Pi
- Raspbian
- Sense HAT emulator

The Sense HAT is one of the most important pieces of Raspberry Pi hardware. The board was developed to travel aboard the International Space Station (ISS) as part of the Astro Pi mission. It was also made available to buy, and schoolkids around the world use it to develop code – some of which runs in space as part of a series of competitions.

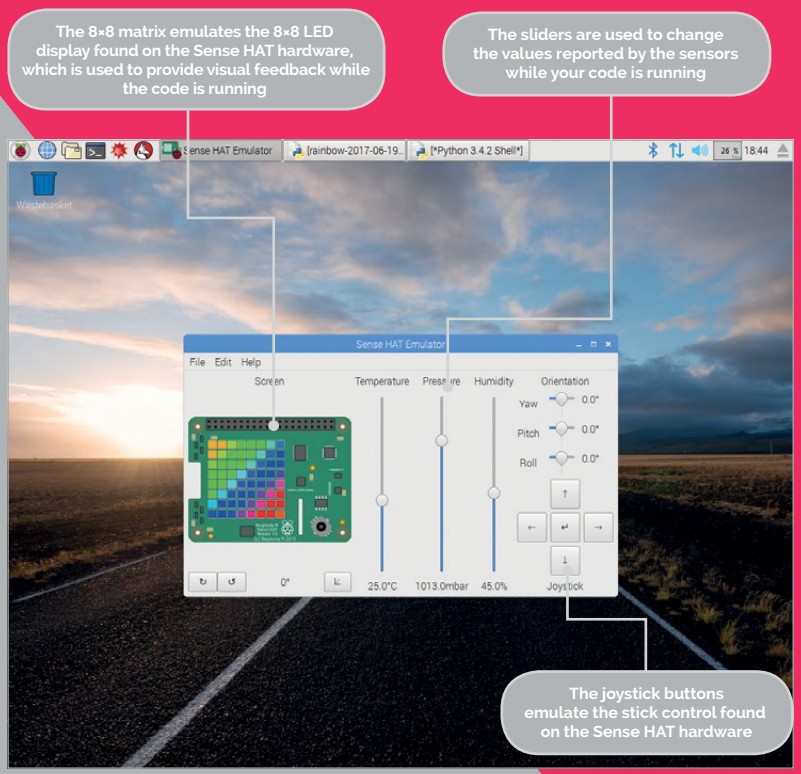
The Sense HAT adds various sensors to the Raspberry Pi: gyroscope, accelerometer, magnetometer, temperature, barometric pressure, and humidity.

The Sense HAT emulator was developed by Dave Jones (**github.com/waveform80**). It is intended for people who own a Raspberry Pi, but not a Sense HAT.

You develop code for the Sense HAT and run it in the emulator. A visual representation of the Sense HAT hardware appears, and a range of sliders and buttons can be used to emulate the Sense HAT's features.

The sliders are used to change the values reported by the sensors while your code is running. You can increase the pressure and humidity that the Sense HAT hardware would detect, and check that your system responds accordingly.

The Sense HAT emulator is a great option for somebody who wants to develop code for the Astro Pi mission, but doesn't have access to the Sense HAT hardware. It's also a great environment for testing code, because you can manually adjust the values reported via the sensors.

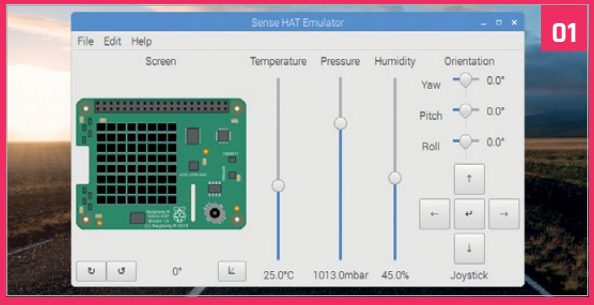


HOW TO: START USING THE SENSE HAT EMULATOR

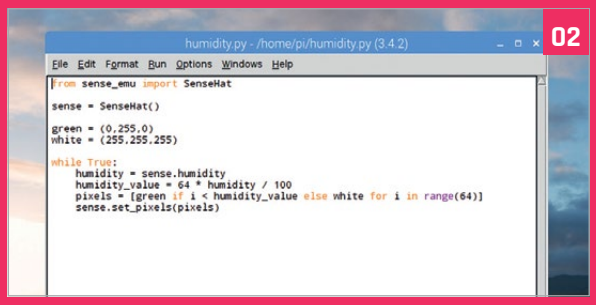
>STEP-01

Start up

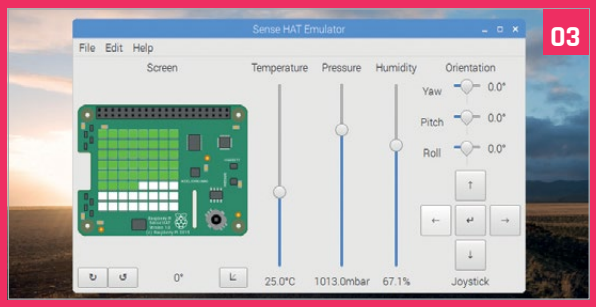
You can access the Sense HAT emulator from the Raspbian desktop menu, under Programming. The emulator closely simulates the experience of attaching the Sense HAT hardware to your Pi. You can read from the sensors or write to the LED matrix using multiple Python processes.



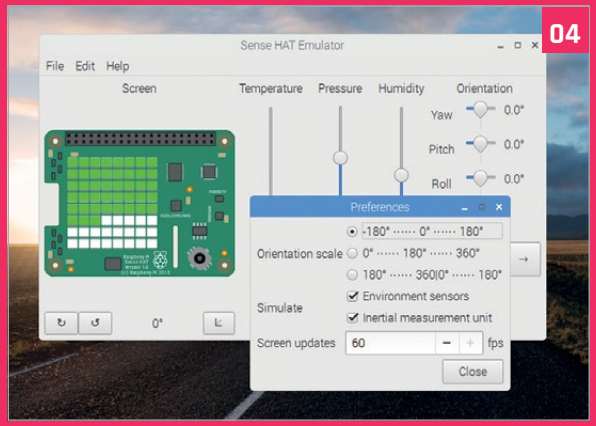
>STEP-02
Code
Open IDLE (Programming > Python 3) and choose File > New. Enter the code from **humidity.py**. This program adjusts the number of green and white pixels displayed on the LED, depending on the detected humidity.



>STEP-03
Run and adjust
Run the program in IDLE (Run > Run Module) and the Sense HAT image will appear and display some green LEDs. Adjust the humidity slider and watch the number of green LEDs change to match the new readings.



>STEP-04
Preferences
There are some preferences that you can adjust to change the behaviour of the emulator. Choose Edit > Preferences. Increase the Screen updates value to provide a more realistic experience of the behaviour of the hardware sensors. You'll see that the values being returned in your code drift according to the known error tolerances of the physical sensors used on the Sense HAT.

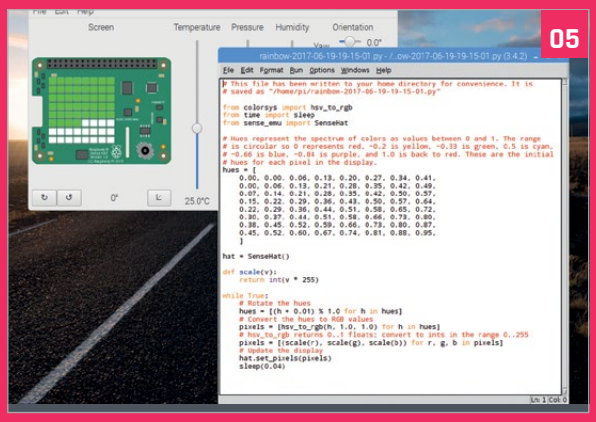


humidity.py

Language
>PYTHON

```
from sense_emu import SenseHat

sense = SenseHat()

green = (0,255,0)
white = (255,255,255)

while True:
    humidity = sense.humidity
    humidity_value = 64 * humidity / 100
    pixels = [green if i < humidity_value
                        else white for i in range(64)]
    sense.set_pixels(pixels)
```

>STEP-05
Code examples
If you're new to the Sense HAT, you can copy and paste a range of example code from the Raspberry Pi educational resources page. Projects include a getting started guide (**magpi.cc/2rvSPSB**) and a random number program (**magpi.cc/2rvpOXq**). You will also find lots of examples under File > Open Example. These will be written to your home directory.



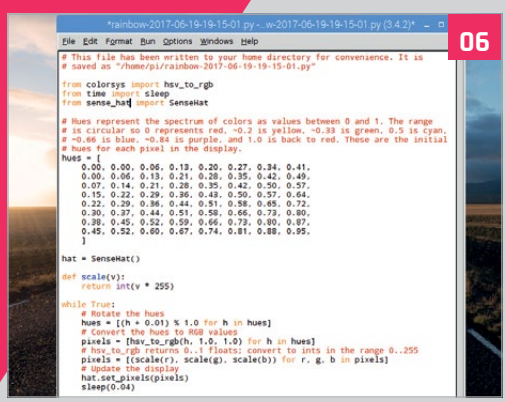
>STEP-06
Port to Sense HAT
If you want to port your emulator code to a physical Sense HAT, you just need to change:

```
sense_emu
```

to...

```
sense_hat
```

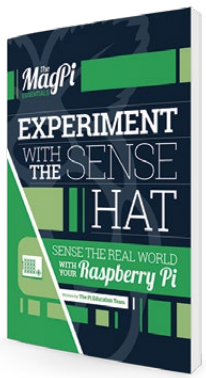
Reverse this if you're porting a physical Sense HAT program to the emulator, perhaps from one of the online educational resources.



Sense HAT Essentials

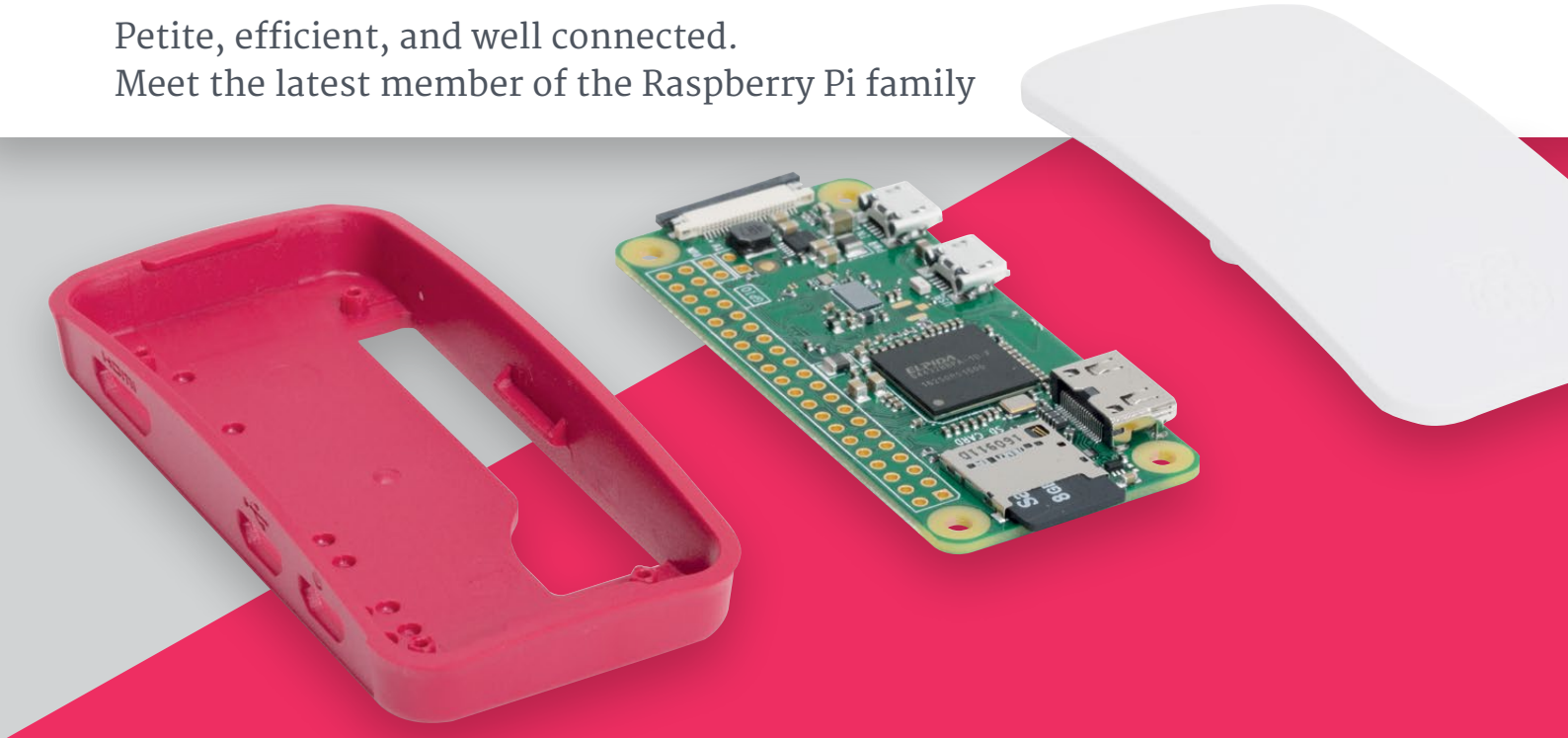
If you're interested in learning more about the Sense HAT, check out our Sense HAT Essentials guide, *Experiment with the Sense HAT*.

magpi.cc/Sense-HAT-book



PI ZERO W

Petite, efficient, and well connected.
Meet the latest member of the Raspberry Pi family



The Official Raspberry Pi Beginner's Book comes with a free computer, the incredible Pi Zero W.

The Pi Zero W is the same small Pi Zero revolutionary board that rocked the maker community, but it packs built-in wireless LAN networking and Bluetooth. It also has an incredible new case. With switchable lids, this enables the Pi Zero W to function as a camera, provides access to its GPIO pins, or lets it act as a secure node.

A third of the size of a Raspberry Pi 3, the Pi Zero W still has enough power to run a full operating system like Raspbian.

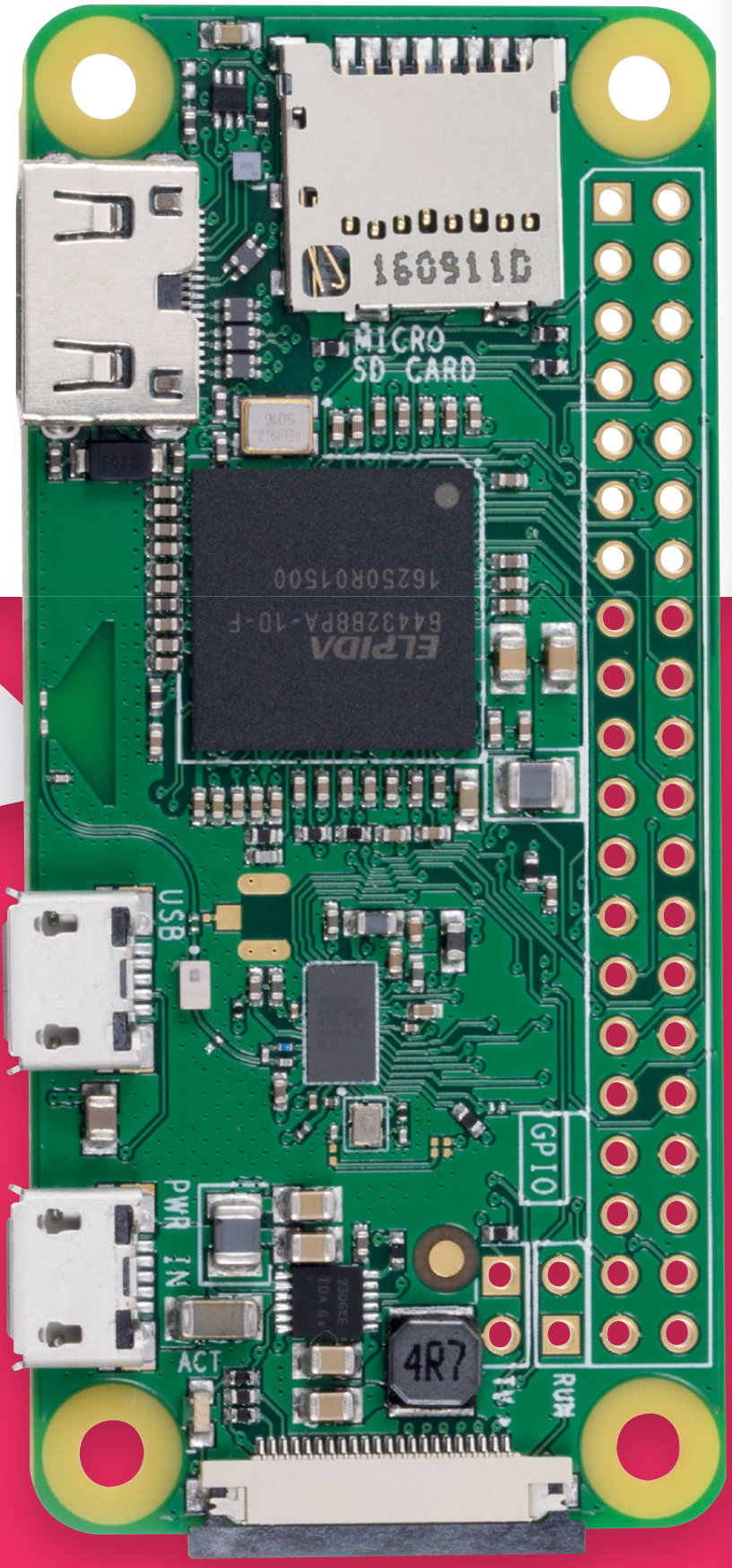
This update to the Pi Zero has the same small form factor, but crams in wireless LAN networking too. With the Pi Zero W, you can connect directly to a local wireless network without having to attach a USB dongle.

The Pi Zero W gets the same wireless networking capabilities first deployed on the Raspberry Pi 3.

Squeezing wireless networking onto a Pi Zero board hasn't been easy, and the way Raspberry Pi went about it has shown considerable ingenuity. The new Pi Zero case is also a thing of wonder.

But it's what you can now do with a Pi Zero W that is going to make a difference.

The Pi Zero W can connect automatically to a local network. With its small size and low-energy footprint, it's ideal as a node for projects around the home, as well as IoT and wearable projects.



PI ZERO W

Technical Specifications

Dimensions:
65mm × 30mm × 5mm

SoC:
Broadcom BCM2835

CPU:
ARM11 running at 1GHz

RAM:
512MB

Wireless:
2.4GHz 802.11n wireless LAN

Bluetooth:
Bluetooth 4.1 LE

Power:
5V, supplied via micro USB connector

Video & Audio:
1080P HD video & stereo audio
via mini-HDMI connector

Storage:
MicroSD card

Output:
Micro USB

GPIO:
40-pin GPIO, unpopulated

Pins:
Run mode, unpopulated;
RCA composite, unpopulated

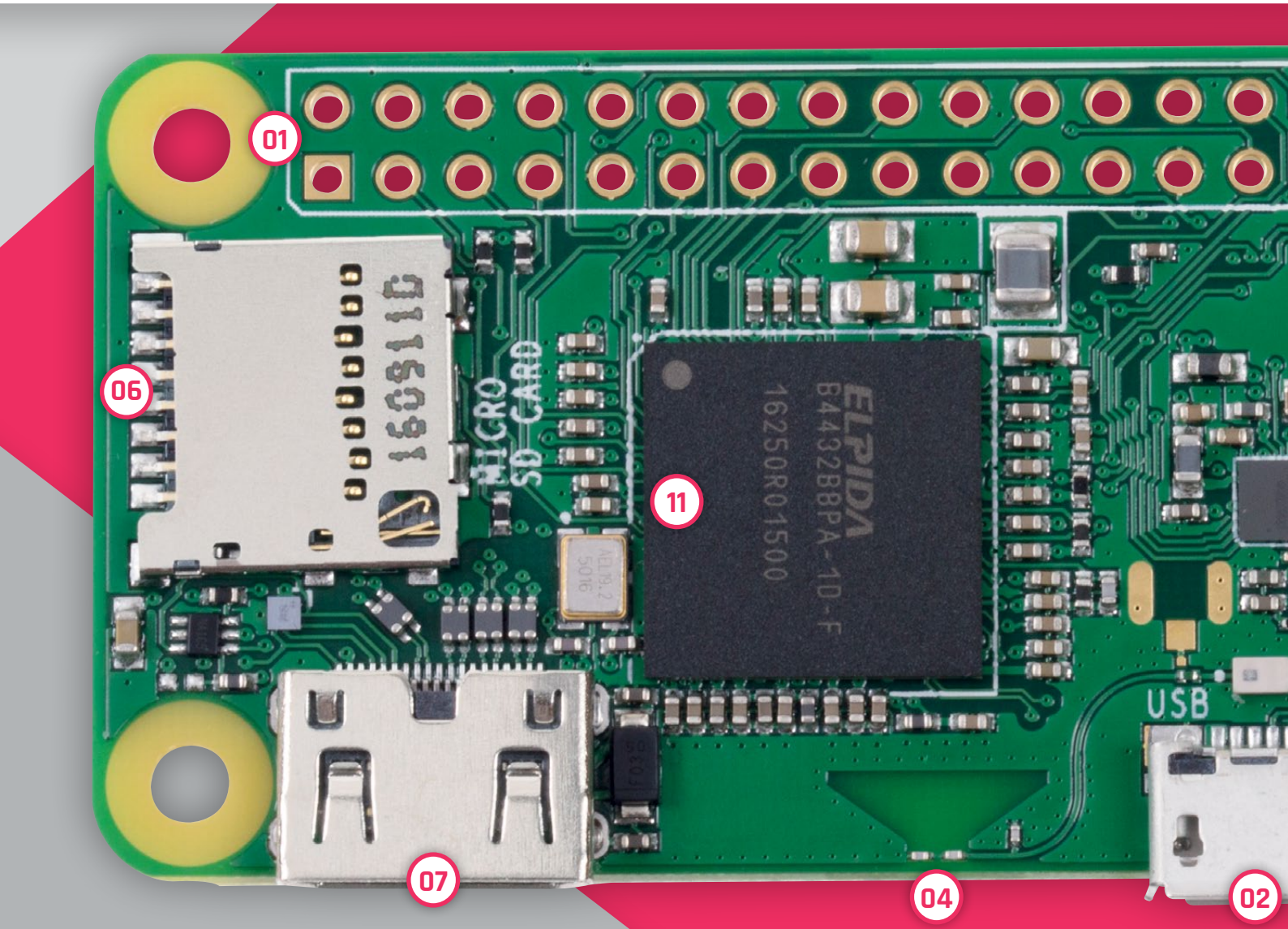
THE BOARD

Up close with the technology packed onto a Pi Zero W board

The technology found in the Pi Zero W is similar to that of the original Pi Zero, only now everything has had to make space for an incredibly smart radio antenna. This design enables the minute Pi Zero board to pack a complete wireless LAN and Bluetooth connection module.

The Pi Zero W contains all of the technology from the original Pi Zero, but with a slightly redesigned board.

The technical surge forward is thanks to the presence of 2.4GHz 802.11n wireless LAN and Bluetooth



4.1 LE. These make the Pi Zero W much more accessible to makers.

It reminded us how packed with technology the Pi Zero is. It features a full 40-pin GPIO header (unpopulated, so you will need to solder the pins on yourself).

Two extra TV pins are used to connect the Pi Zero W to an RCA television, while two pins marked RUN can be used to plug in a reset button.

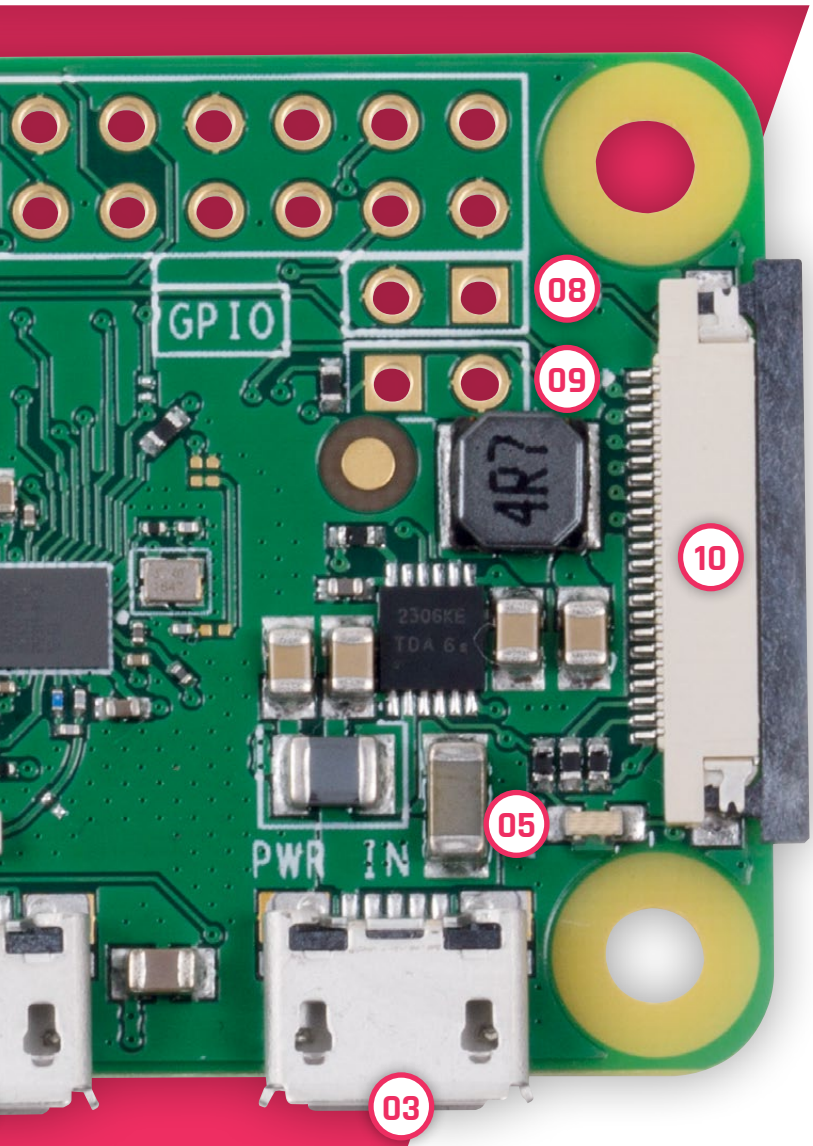
A single USB On-The-Go connection allows you to hook up devices to the Pi Zero W, and the mini HDMI socket connects the board to monitors and TVs.

To hook up stock cables, you'll want a Raspberry Pi Zero Adapter Kit (£4, **magpi.cc/2kPZNzn**) to get the most from your Pi Zero W.

A CSI connection supports the Camera Module v2 (**magpi.cc/28IjIsz**). This adds a Sony IMX219 8-megapixel camera to the Pi Zero W. It's especially pertinent thanks to the new case.

The processor remains a Broadcom BCM2835 containing an ARM11 running up to 1GHz. The Pi Zero W is a full computer, capable of running Raspbian and other Linux operating systems.

> "The Pi Zero W is a full computer, capable of running Raspbian and other Linux operating systems"



01. GPIO header
Pi Zero W has the same 40-pin GPIO header as the larger Raspberry Pi models, but it's unpopulated. Adding pins to the Pi Zero W board is a great first soldering project.

02. Micro USB
A single micro USB port is used to provide connectivity. Only now you don't need to add a USB WiFi dongle, so it'll be easier to connect just a keyboard and mouse.

03. Micro USB power
The second micro USB port is used to provide power to the board.

04. Wireless LAN
This triangular shape is used to boost the wireless connection. The signal bounces around the triangle and connects to the points at the bottom (before being fed up the pipes to the side).

05. Bluetooth 4.1
The Pi Zero also connects to Bluetooth 4.1. This connectivity makes it easy to hook up wireless devices and share files with the board.

06. MicroSD card
A microSD card is used to provide the storage for the Pi Zero W.

07. Mini HDMI
A mini HDMI connection is used to connect the Pi Zero W to a TV or monitor.

08. RUN pins
The two RUN pins can be used to wire up a reset button.

09. TV pins
The TV pins can be used to add an RCA video connection. Perfect for hooking up a Pi Zero W to older television sets (for retro gaming perfection).

10. Camera Module
A camera port was added to the Pi Zero 1.3 and remains on the Pi Zero W. The Camera Module is tightly integrated with the Pi Zero thanks to the new case.

11. Broadcom BCM2835
This Broadcom BCM2835 SoC is the heart of the Pi Zero W. It contains An ARM 11 CPU running at up to 1GHz.

BEHIND THE PI ZERO W

We chat with Roger Thornton, principal hardware engineer at the Raspberry Pi Foundation

Roger Thornton is the person in charge of the Raspberry Pi Zero W project: developing the new board has been a labour of love for him.

The first Raspberry Pi product to gain wireless networking was the Raspberry Pi 3 (released shortly after the original Zero). Developing this product gave Raspberry Pi the experience they needed to bring wireless networking to the Zero.

There was no easy way the wireless layout from the larger Raspberry Pi 3 board would fit on the diminutive Zero. "We thought it was impossible," muses Roger, "but we were getting better at squeezing features onto products, so we did a little playing around with the design." According to Roger, there was a lot of "pushing and shoving" to fit all the components into a restricted area, and onto one side of the board.

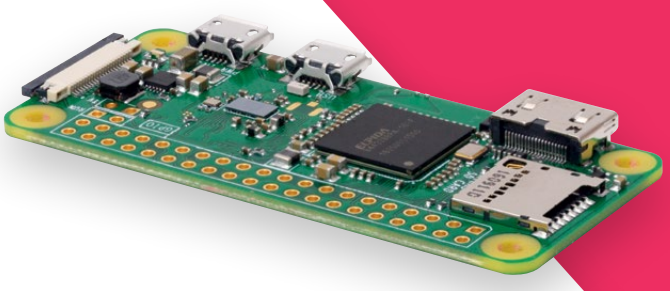
The Raspberry Pi 3 layout was based on a two-sided Broadcom reference design, with a 'chip' antenna. In contrast, the Zero W uses a single-sided layout with a PCB antenna; this is the trapezoidal shape between the mini HDMI and micro USB sockets on the bottom edge of the board.

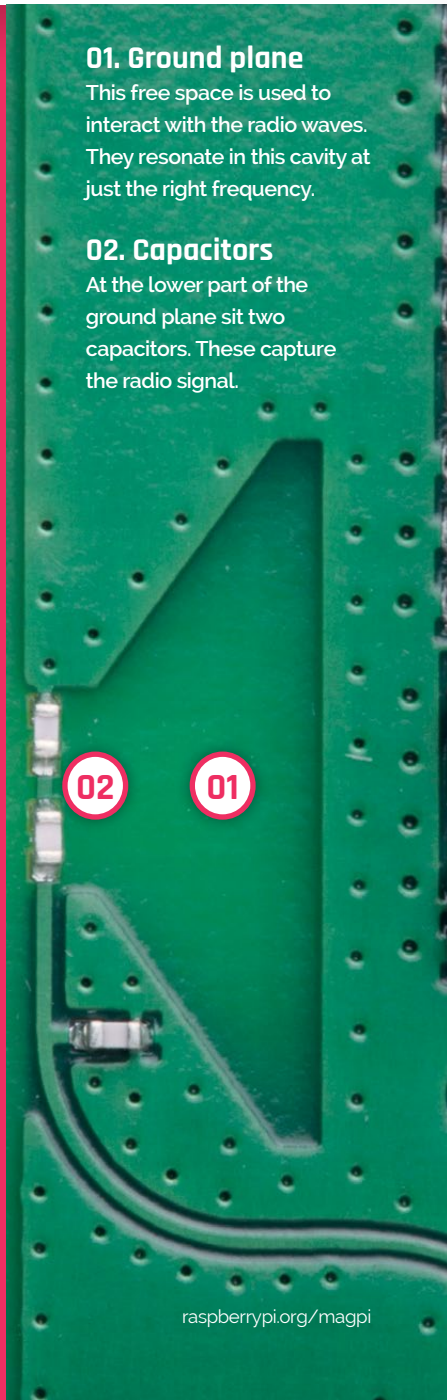
> "They're very clever boffins. It's a really neat design"

"The Raspberry Pi 3 antenna is a surface-mount component," explains Roger, "whereas the Zero W antenna is a resonant cavity which is formed by etching away copper on each layer of the PCB structure."

The technology is licensed from a Swedish company called Proant (you can see the credit on the reverse of the Zero W board).

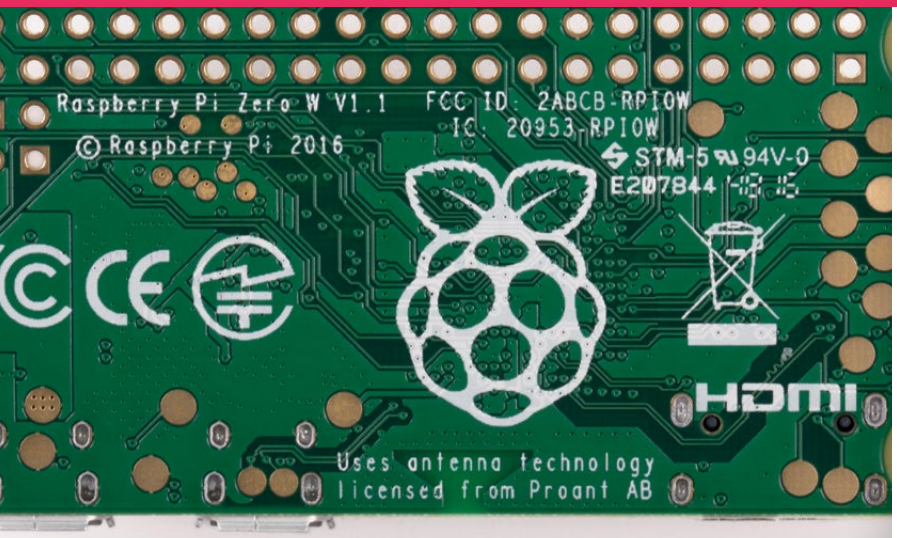
"They're very clever boffins," says Roger. "It's a really neat design."



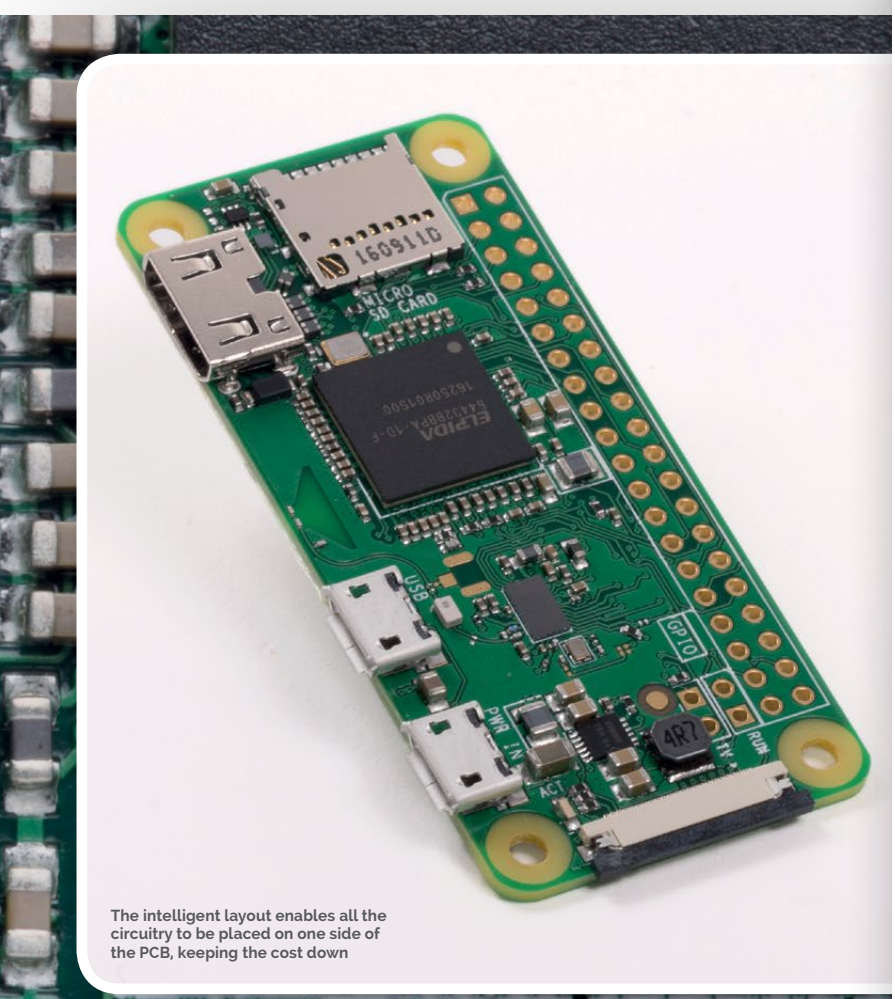


01. Ground plane
This free space is used to interact with the radio waves. They resonate in this cavity at just the right frequency.

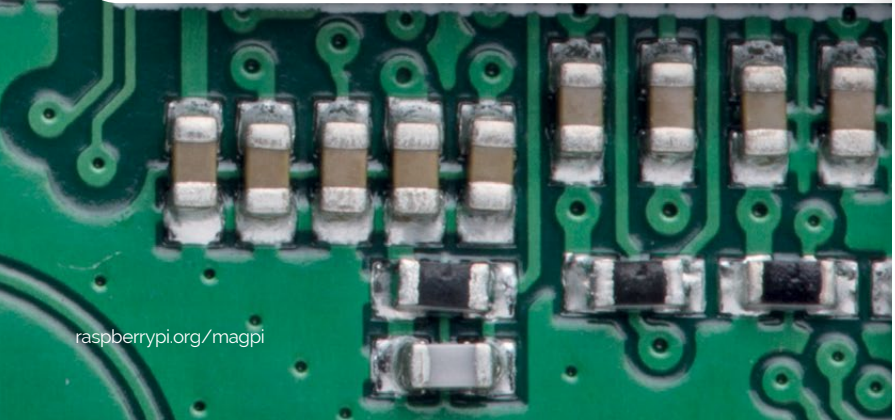
02. Capacitors
At the lower part of the ground plane sit two capacitors. These capture the radio signal.



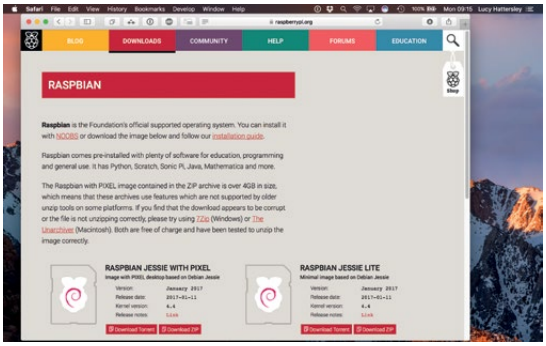
The white text is added to the PCB using a silk-screen technique. These markings are used to provide information and branding for the board



The intelligent layout enables all the circuitry to be placed on one side of the PCB, keeping the cost down

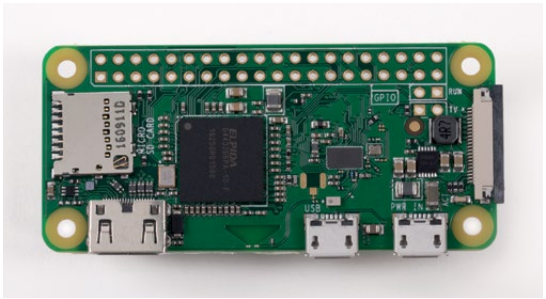


SET UP A **PI ZERO W**



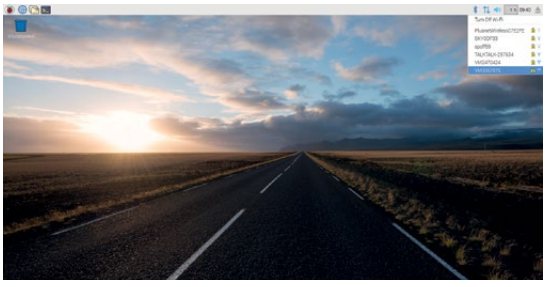
>STEP 01
Download Raspbian

You'll need the latest version of Raspbian with PIXEL from **raspberrypi.org/downloads**. Make sure you get the most recent update, as the latest version adds Pi Zero W support. Put your Pi Zero W inside its case.



>STEP 02
Set up the Pi Zero W

Flash the microSD card and insert it into your Pi Zero W. Use a USB On-The-Go adaptor to connect your keyboard and mouse to the micro USB port marked 'USB' (chain the mouse via a keyboard USB if possible). Use a mini HDMI to HDMI cable (or adapter) to connect the Pi Zero W to a monitor.



>STEP 03
Power on

Connect a power supply to the micro USB port marked 'PWR'. The Pi Zero W will power up into Raspbian with PIXEL. Click on the wireless icon in the top-right of the screen to locate your local wireless network. Enter your wireless password to connect the Pi Zero W to the network.

PI ZERO W CASE

A whole new way to hold your Raspberry Pi

Alongside your Pi Zero W is a fantastic case.

The design is similar to the stunning Raspberry Pi 2/3 case. The same design team, Kinneir Dufort, is responsible for creating the Pi Zero W case.

It's an amazing case which shares a lot of design DNA with the original. It also comes in the same red and white colour scheme.

The case is composed of two parts: a red base and a white lid. They clip together to contain the Pi Zero. On the rear of the red case is a cut-out to allow room for GPIO pin edges and solder.

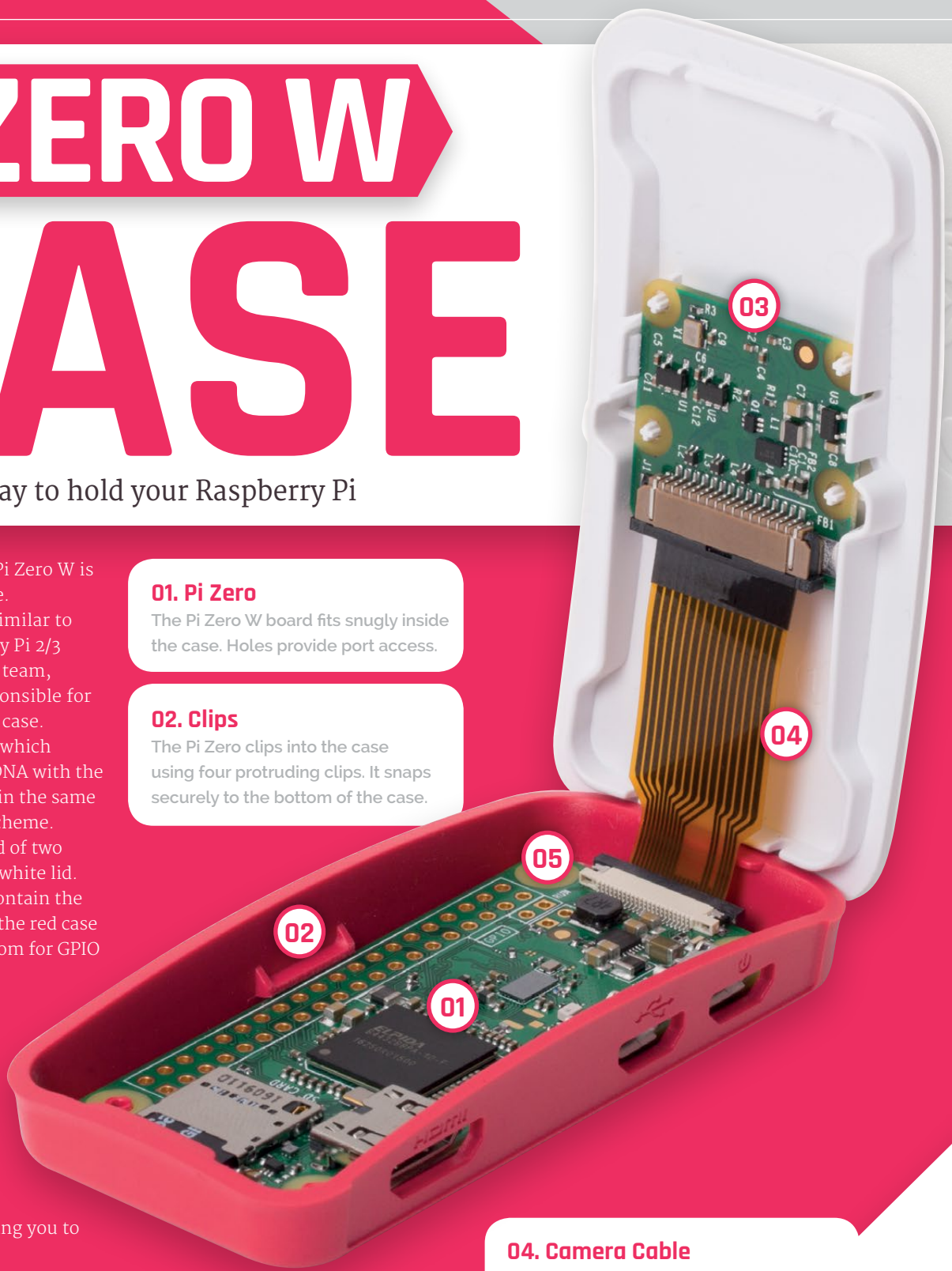
The white cover is interesting. You don't get just one lid; three are available...

The first is a solid cover that keeps your Pi Zero W components secure.

The second has a rectangular slot enabling you to access the GPIO pins.

The third is the most striking. This cover contains a holder for the Camera Module. The camera clips onto the inside of the lid, and a perfectly sized gap enables it to take photos through the cover.

Each Pi Zero W case comes with all three covers, so you can use it whichever way you choose.



01. Pi Zero

The Pi Zero W board fits snugly inside the case. Holes provide port access.

02. Clips

The Pi Zero clips into the case using four protruding clips. It snaps securely to the bottom of the case.

03. Camera Module

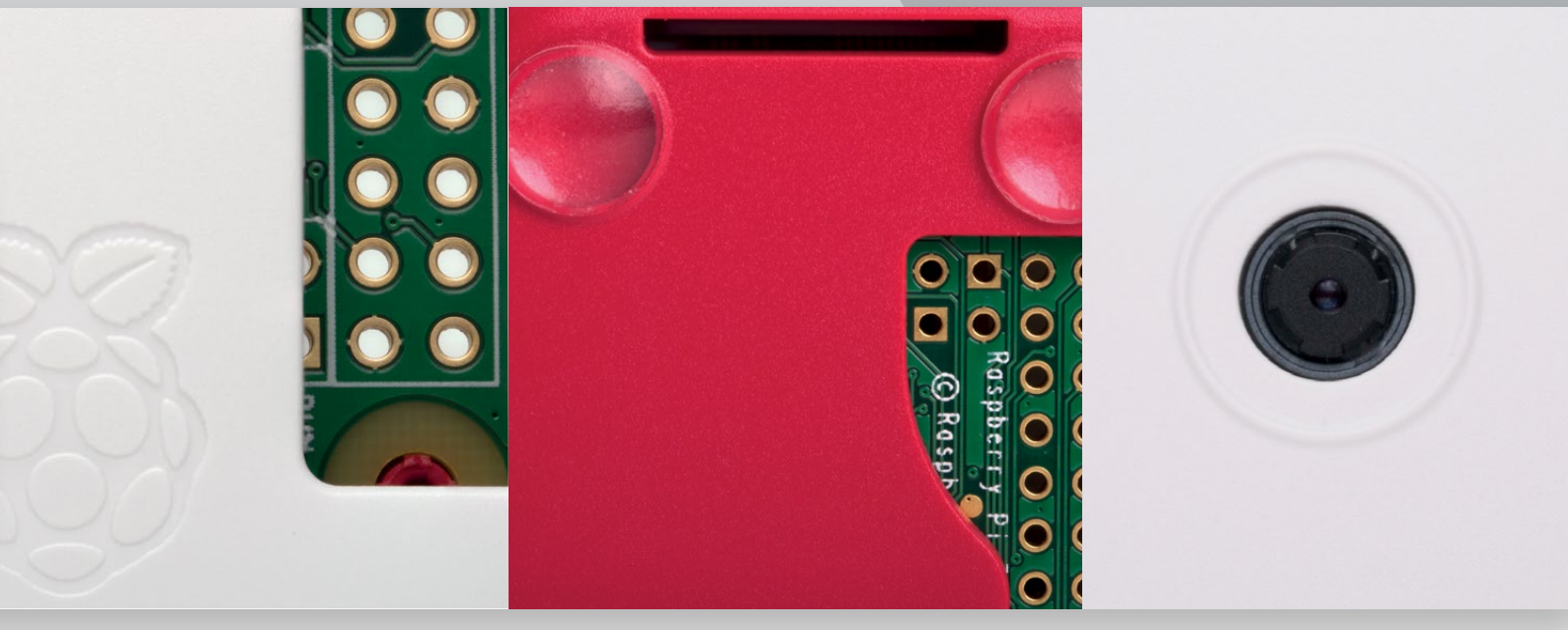
The Camera Module is attached to the inside of the camera case lid. Like the Pi Zero W board, it snaps into place with the four plastic protrusions.

04. Camera Cable

The Pi Zero W case comes with a Pi Zero W camera cable. With this, you connect the Camera Module to the Pi Zero W board.

05. GPIO header

The case lids are interchangeable, and one lid enables direct access to the GPIO pins on the Pi Zero W.



THREE COVERS

Secure cover

The safe cover clips on to the Raspberry Pi case, completely sealing up the Raspberry Pi. Perfectly shaped holes on the side of the case enable you to access the USB and HDMI ports. It makes for the cutest computer around.

GPIO cover

The GPIO cover has a cut-out section that enables you to access the GPIO pins without removing the lid. Another cut-out in the base allows room for solder and pin edges (on the reverse of the board), but could be used for GPIO access too.

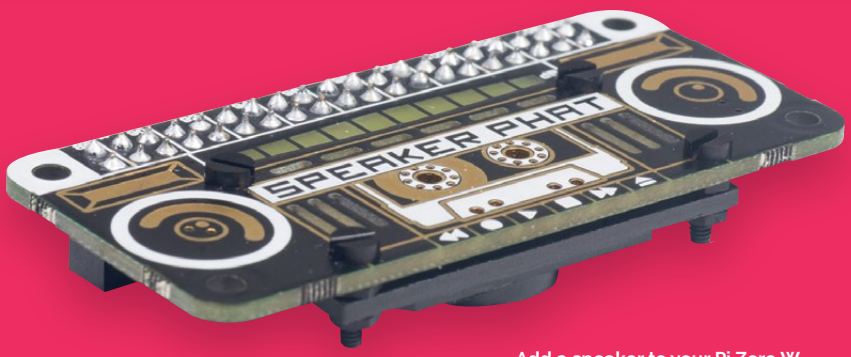
Camera cover

We think the camera cover is the star attraction. It holds the Camera Module in place and resembles a digital camera. We can't wait to see what Pi Zero W projects the community makes with integrated wireless and camera functionality.

PI ZERO W PROJECTS

Five great project ideas for you to build with your new Pi Zero W

AMAZON ECHO



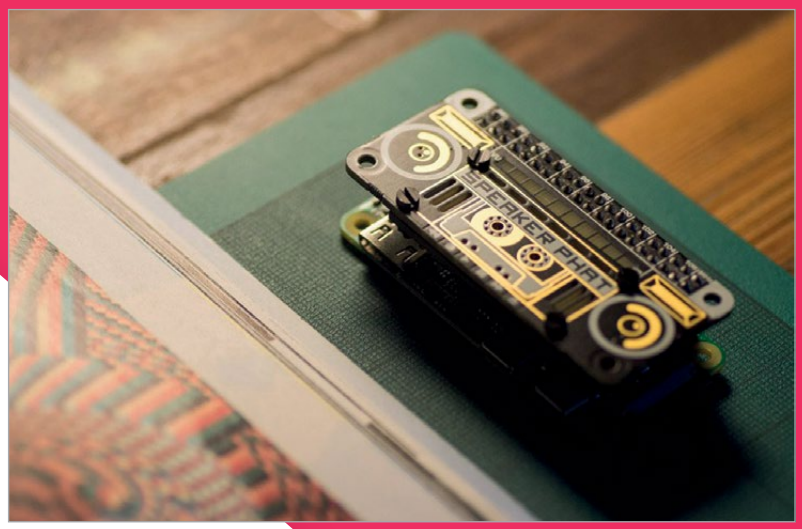
Add a speaker to your Pi Zero W using a Speaker pHAT

Amazon Alexa is an intelligent assistant making a lot of waves in the tech community. Amazon sells two Alexa products: the Amazon Echo and Echo Dot.

Amazon has also created a homebrew version of Alexa for the Raspberry Pi.

Now that the Pi Zero W has wireless LAN, it's a perfect portable Amazon Alexa device. All you need are some speakers and a USB microphone.

Pimoroni's Speaker pHAT (£12, **magpi.cc/2kXdZsE**) is a neat solution. Combine it with a USB microphone using the USB to USB On-The-Go adapter.



Right Use a speaker and microphone to turn your Pi Zero W into a hands-free Amazon Alexa Echo

Previously, you also had to add a button to the Raspberry Pi version of Alexa, but a recent update has added an 'always on' mode to the Raspberry Pi build. Alexa now listens for a vocal cue, such as the word 'Alexa' and then starts monitoring. You can change the vocal signal to any name of your choosing; 'computer' is a popular choice for Star Trek fans.

A GitHub repository has all the information you need to set up Amazon Alexa on a Raspberry Pi (**magpi.cc/2kXrdW2**). There is a complete guide here (**magpi.cc/2kXfPcO**). Frederick Vandenbosch has written a guide to installing Alexa on a Raspberry Pi Zero (**magpi.cc/2kXkY4A**).

WEARABLE CAMERA

The Pi Zero W case with camera lid lends itself perfectly to making a mini camera. You could add a button to the case using Sugru to build a digicam, or use a Bluetooth camera button (as found on selfie sticks).

We also like the idea of making a wearable camera. Adafruit has a great tutorial on creating this project. Adafruit used a unique 3D-printed case, but now you can simply use the Pi Zero W case (**magpi.cc/2kXASMw**).

MOTION-SENSING CAMERA

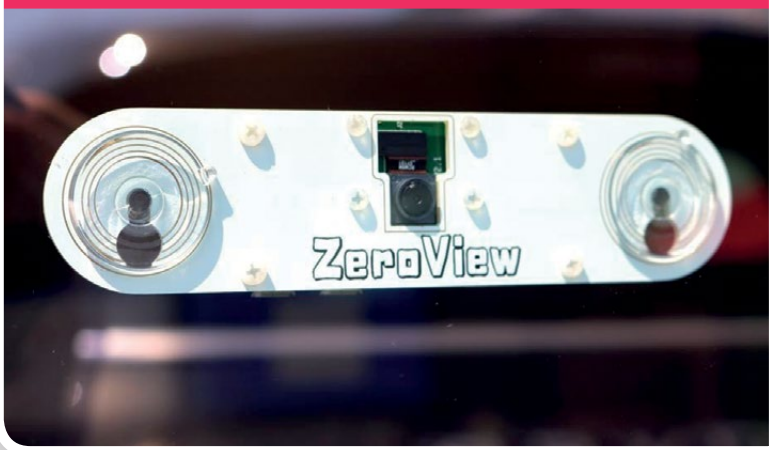
The low power draw of the Pi Zero W also makes it ideal for use as a security camera, or as an outdoor wildlife camera. Hook it up to a Zero LiPo (**magpi.cc/2kXI8YW**) and you'll get a steady stream of power. Connect the Pi Zero camera and use the wireless LAN to send alerts when it detects motion. Take a look at Mark West's project (**magpi.cc/2kXznhq**).

FILE SERVER

The low power draw of the Pi Zero W and its wireless connectivity make it ideal for use as a connected node. Add some storage and turn it into always-on network storage. This has always been a great Raspberry Pi project, but it is especially easy with the Zero W's built-in wireless connectivity.

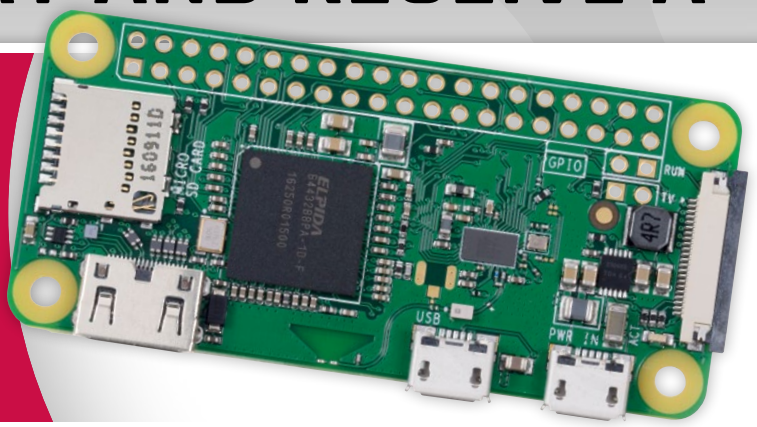
SMART CAR DASH

The Pi Zero W is easy to mount on a car dashboard using a Zero View mount (**magpi.cc/2kXIsa2**). You can use it as a dash cam to record footage. With the wireless LAN, you can connect to a personal hotspot (from your mobile phone). Consider checking out the open-source Dride project for software (**magpi.cc/2lnBwjt**).



SUBSCRIBE TODAY AND RECEIVE A

FREE PI ZERO W



PLUS OFFICIAL **PI ZERO CASE** WITH 3 COVERS

AND FREE CAMERA MODULE CONNECTOR AND USB / HDMI CONVERTER CABLES

Subscribe in print for 12 months today and receive:

- A free Pi Zero W (the latest model)
- Free Pi Zero W case with three covers
- Free Camera Module connector
- Free USB and HDMI converter cables

Other benefits:

- Save up to 25% on the price
- Free delivery to your door
- Exclusive Pi offers and discounts
- Get every issue first (before stores)

Pricing

Get six issues:

£30 (UK)

£45 (EU)

$69 (USA)

£50 (Rest of World)

Subscribe for a year:

£55 (UK)

£80 (EU)

$129 (USA)

£90 (Rest of World)

FREE PI ZERO W!

Get three issues:

£12.99 (UK) **(Direct Debit)**

$37.50 (US) **(quarterly)**

How to subscribe:

- **magpi.cc/Subs-2** (UK / ROW)
- **imsnews.com/magpi** (USA)
- Call +44(0)1202 586848 (UK/ROW)
- Call 800 428 3003 (USA)

Available on the App Store

GET IT ON Google Play

SUBSCRIPTION FORM

YES! I'd like to subscribe to *The MagPi* magazine and save money

This subscription is: ☐ For me ☐ A gift for someone*

Mag#63

YOUR DETAILS Mr ☐ Mrs ☐ Miss ☐ Ms ☐

First name ..... Surname .....

Address .....

.....

Postcode ..... Email .....

Daytime phone ..... Mobile .....

**If giving* The MagPi *as a gift, please complete both your own details (above) and the recipient's (below).*

GIFT RECIPIENT'S DETAILS ONLY Mr ☐ Mrs ☐ Miss ☐ Ms ☐

First name ..... Surname .....

Address .....

Postcode ..... Email .....

PAYMENT OPTIONS

1 DIRECT DEBIT PAYMENT £12.99 every 3 issues (UK only)

Instruction to your bank or building society to pay by Direct Debit

DIRECT Debit

Please fill in the form and send to:
The MagPi, Select Publisher Services Ltd,
PO Box 6337, Bournemouth BH1 9EH

Service user number 8 3 8 7 7 3

Name and full postal address of your bank or building society:

To: The Manager Bank/building society .....

Address .....

.....

..... Postcode .....

Name(s) of account holder(s) .....

Branch sort code ☐☐☐☐☐☐ Account number ☐☐☐☐☐☐☐☐

Reference ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐ (Official use only)

Instruction to your bank or building society

Please pay Select Publisher Services Ltd Direct Debits from the account detailed in this instruction subject to the safeguards assured by the Direct Debit Guarantee. I understand that this instruction may remain with Select Publisher Services Ltd and, if so, details will be passed electronically to my bank/building society.

Signature ..... Date ☐☐/☐☐/☐☐

Banks and building societies may not accept Direct Debit instructions for some types of account.

SUBSCRIPTION PRICING WHEN PAYING BY CHEQUE OR CREDIT/DEBIT CARD

6 ISSUES ☐ UK £30 ☐ Europe £45 ☐ Rest of World £50

12 ISSUES ☐ UK £55 ☐ Europe £80 ☐ Rest of World £90

2 CHEQUE

I enclose a cheque for ..... (made payable to Select Publisher Services Ltd)

3 CREDIT/DEBIT CARD ☐ Visa ☐ MasterCard ☐ Maestro ☐ Switch

Card number ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

Expiry date ☐☐ ☐☐ Valid from ☐☐ ☐☐ (if shown)

Issue number ☐☐ (if shown) Security number ☐☐☐

(last 3 digits on the back of the card)

Signature ..... Date ☐☐/☐☐/☐☐

I would like my subscription to begin from issue ..... (month + year)

RETURN THIS FORM TO:

MagPi Magazine Subscriptions, Select Publisher Services Ltd, PO Box 6337, Bournemouth BH1 9EH

☐ Please tick this box if you DO NOT want to receive any other information from Select Publisher Services Ltd.

☐ Please tick this box if you DO NOT want to receive any other information from other companies.

☐ Please tick this box if you DO NOT want to subscribe to The MagPi newsletter.