9 Awesome Projects
written especially for young people!

Adventures in Raspberry Pi®

Third Edition

Get busy with your Raspberry Pi!
Guide yourself through the complex activities of programming to create these amazing projects!

Carrie Anne Philbin

WILEY

Adventures in Raspberry Pi®

Third Edition

Adventures in Raspberry Pi®

Third Edition

Carrie Anne Philbin

WILEY

Adventures in Raspberry Pi®, Third Edition

This edition first published 2017

Registered office

John Wiley & Sons, Inc., 111 River St, Hoboken, NJ 07030-5774.

For details of our global editorial offices, for customer services and for information about how to apply for permission to reuse the copyright material in this book please see our website at www.wiley.com.

Library of Congress Control Number: 2017937739

A catalogue record for this book is available from the British Library.

ISBN 978-1-119-26906-9 (paperback); ISBN 978-1-119-26907-6 (ePub); 978-1-119-26908-3 (ePDF)

Set in 10/12.5 Chaparral Pro Light by SPi Global

Printed in the United States of America by Command Web Missouri

For Mum & Dad—my best teachers.

Publisher's Acknowledgements

Some of the people who helped bring this book to market include the following:

Editorial

Series Creator: Carrie Anne Philbin
VP Consumer and Technology Publishing Director: Michelle Leete
Associate Director—Book Content Management: Martin Tribe
Professional Technology & Strategy Director: Barry Pruett
Acquisitions Editor: Jody Lefevere
Project Editor: Charlotte Kughen
Technical Editor: Martin O'Hanlon
Editorial Manager: Mary Beth Wakefield
Editorial Assistant: Matthew Lowe
Production Editor: Athiyappan Lalith Kumar

Marketing

Marketing Manager: Lorna Mein
Marketing Assistant: Polly Thomas

About the Author

CARRIE ANNE PHILBIN is an award winning Computing & ICT Teacher, Author & YouTuber. She currently works as Director of Education for the Raspberry Pi Foundation to put the power of computing and digital making into the hands of people all over the world. She also volunteers on the board for Computing At School and as a Director of the Python Software Foundation. Alongside her work in formal education in the UK, Carrie Anne also hosts a Computer Science series on popular educational YouTube channel Crash Course (`www.youtube.com/crashcourse`) and creates technology tutorials on her own channel the Geek Gurl Diaries (`www.geekgurldiaries.co.uk/`).

Acknowledgments

I would like to express my deep gratitude to the Raspberry Pi Foundation for allowing me to set my creativity free on their marvellous inventions. In particular I'd like to thank Alex Bradbury, Dr Sam Aaron and Nicholas Tollervey for their enthusiastic encouragement and patient guidance. Their willingness to give their time so generously is very much appreciated. I would also like to thank Martin O'Hanlon and the Raspberry Pi Community for their useful critiques of this work. Thanks to Jennifer Mayberry for her design work and Sarah Wright for much of the art.

Special thanks should also be given to the staff of Pimoroni for providing necessary equipment in order to complete elements of this book, as well as members of CAS #include and the Rainham Library Book Club, for keeping my progress on schedule with their kind words of encouragement. Thanks to my colleagues at Raspberry Pi, PyConUK and the wider community who have been parts of these adventures for many years.

My special thanks are also extended to my good friends—Emma, Sian, Helen, Viv and Kylie—who are a constant source of inspiration in my life.

Finally, I wish to thank my parents, husband, brother and sister-in-law for their patience, support and encouragement throughout.

Contents

Introduction

ARE YOU AN intrepid adventurer? Do you like to try new things and learn new skills? Would you like to be a pioneer in creating technology? Do you own a Raspberry Pi, or are you considering getting one? If the answer is a resounding "Yes!" then this is the book for you.

What Is the Raspberry Pi and What Can You Do With It?

The Raspberry Pi is a computer. A very small computer. In fact, it is roughly the size of a credit card. Don't be fooled by its size; as we know, good things come in small packages. However, the Raspberry Pi does not come in a package at all. It does not come in a case (although you can build one, as discussed in Adventure 1) and its circuit board and chips are fully visible, as you can see in Figure 1. You can plug a Raspberry Pi into a digital TV or monitor and use a USB keyboard and mouse with it, making it very easy to use. Because of its size, you can easily transport it anywhere.

The Raspberry Pi gives you the opportunity to build and control a device that does what you want it to do. For example, you can deploy your very own robot arm, controlled by a program that you have written. You can design and create your own role-playing game, or produce beautiful computer art or music, all by using code.

Just because the Raspberry Pi is small doesn't mean you can't do big things with it. Here are just a few examples of some incredible Pi projects:

- Launching teddy bears into space using high altitude ballooning (www.raspberrypi.org/archives/4715)
- The ultimate bird feeder—it's solar-powered, takes photographs and tweets images of birds! (www.raspberrypi.org/archives/4832)
- Crazy customised Halloween costumes like Doc Brown from Back to the Future (www.raspberrypi.org/archives/4856)
- A robotic sailboat (www.raspberrypi.org/archives/4109)
- Pi-controlled sculptures like the 15-foot tall Mens Amplio with a brain that lights up (www.raspberrypi.org/archives/4667)

In the final adventure of this book, you use your Pi to build a jukebox that plays your favorite tunes and displays track information on an LCD screen. And with the skills you learn throughout the book, you'll be ready to dream up your own exciting projects—and create them.

FIGURE 1 Raspberry Pi computers.

Who Should Read This Book?

Adventures in Raspberry Pi is for any young person who has an interest in making things happen using computing. You might perhaps be unsure of how to get started or want to further your current skills. Whatever your reasons, this book will be your guide for a journey with your Raspberry Pi, the most important item in your backpack. Your trek will take you from setting up your Pi, through learning the basics of programming, to discovering how to create your own project. By the end of your adventures you will have acquired the skills you need to become a pioneer of technology!

What You Will Learn

This book will help you discover some of the amazing things you can do with your new Raspberry Pi, and introduce you to many of the developer tools and projects available to you. With this book, you learn how to set up and use your Raspberry Pi easily so that you can experience its potential for yourself. You learn the skills you need to design and create your own computing projects.

You find out that you can give instructions to your Raspberry Pi in a variety of ways, using different programming languages and tools. The adventures in the book allow you to experience programming using Scratch, Turtle Graphics, Python, Sonic Pi and Minecraft Pi.

You also learn some computing (and electronics) concepts that you can apply to other devices and programming situations. Many fundamental computing concepts are similar for all programming languages, so once you understand the basics of programming in one language you can apply that knowledge to others very easily.

What You Need for the Projects

First and foremost, of course, you need a Raspberry Pi. If you don't already own one, you can buy a Raspberry Pi from a distributor in your country. You also need a monitor or other screen, a mouse and a keyboard to connect to your Raspberry Pi.

Each chapter—adventure—in the book notes any special items you need to build the project covered in that adventure. Along with your Pi, some projects require Internet access to enable you to download software or other materials. You need headphones or speakers to listen to the music you make in Adventure 7. For the projects in Adventures 8 through 10, you need some specific cables, wires, LEDs, resistors and other hardware. You can purchase these items from your local electronics store, or from various online retailers.

As final ingredients, you need some curiosity and a willingness to try new skills!

How This Book Is Organised

Every chapter of the book is a separate adventure, teaching you to use new skills and concepts while you create a project. The book is organised so that as you progress, the concepts and projects get more complex, building on what you learned in earlier adventures. Each chapter begins with an introduction to the language or tool for that adventure, provides instructions for downloading, installing, and setting up whatever you need, and usually gives you a short task to help you become familiar with the tool. After you've got the basics, I lead you step by step through the instructions for the main project.

In Adventures 1 and 2, you learn how to get started with your equipment and use common text commands, perhaps for the first time. These two chapters are necessary for the beginner Pi explorer, as further adventures depend on the skills covered here.

The two most common ways to program a Raspberry Pi are to use the Scratch or Python languages that come preinstalled on the Pi's main operating system, Raspbian. Adventures 3, 4 and 5 get you started with the basics of these languages.

In Adventure 3, you use Scratch, a simple drag-and-drop programming language, to design and create your own computer game, while getting an introduction to the programming concepts of loops and variables. Adventure 4 is a bridge between Scratch and the more conventional programming language, Python. In this adventure, you use Turtle Graphics to create shapes and spirals with both programming languages. In Adventure 5, you learn how to create an adventure game program that asks for user input, uses lists, imports functions and prints text to the screen, all using text commands written in the programming language Python.

Adventures 6 and 7 take programming on the Raspberry Pi further by looking at two developer tools that you can download and use with the Raspberry Pi: Minecraft Pi and Sonic Pi. Minecraft Pi enables you to interact with and adapt the popular computer game Minecraft, using Python code to build your own transporter. With Sonic Pi, you can create electronic music by writing programs.

Another exciting aspect of using the Raspberry Pi is that it gives you the option to add on to the main board by using GPIO pins. Adventure 8 looks at the GPIO pins in more detail, introducing you to electronics and computer programming while you build a program that uses a marshmallow to make a light blink (yes, you read that right).

You don't always need to program individual electronic components; you can also buy specially designed hardware that fits on top of the GPIO pins on the Raspberry Pi, called HATs. In Adventure 9, you learn more about the features of the Sense HAT and the Explorer HAT before programming them with Python.

Adventure 10 draws on the computing concepts and skills learned through completing the preceding adventures in this book to create one big project—a jukebox. In this chapter, you learn how to plan, design and create a project from start to finish.

Finally, the Appendix suggests where you might go next to learn more about the different aspects of computer science and Raspberry Pi—including how to locate or set up your own club to share project ideas with others.

The Companion Website

Throughout this book you'll find references to the Adventures in Raspberry Pi companion website, `www.wiley.com/go/adventuresinrp3E`. (It's a good idea to bookmark that site so you can return to it as you need to.) The website includes video tutorials to help you out if you get stuck, and code files for some of the more extensive projects.

Conventions

Throughout the book, there are some special boxes to guide and support you. They use the following key:

These boxes explain complex computing concepts or terms.

These boxes are hints to make life easier.

These boxes include important warnings to keep you and your Raspberry Pi safe when completing a step or project.

These boxes feature quick quizzes for you to test your understanding or make you think more about the topic.

These boxes provide explanations or additional information about the topic at hand.

These boxes point you to videos on the companion website that walk you through the tasks in the adventure.

You will also find two sets of sidebars in the book. Challenge sidebars ask you how you might expand on the projects in the book to make changes or add new features. Digging into the Code sidebars explain some of the special syntax or programming language, to give you a better understanding of the computer languages.

When following steps or instructions using code, especially in adventures using Python, you should type in the code as set out by the instructions. Sometimes you need to type a very long line of code, longer than will fit on a single line in this book. If you see a ↩ symbol at the end of a line of code, it means that line and the following line are part of a single code line, so you should type them as one line, not on separate lines. For example the following code should be typed on one line, not two:

```
print("Welcome to Adventures in Raspberry Pi by ↩
Carrie Anne Philbin")
```

Most chapters include a Quick Reference Table at the end to sum up the main commands or concepts from the chapter. You can refer to these guides when you need a refresher on the commands.

Whenever you complete a chapter, you unlock an achievement and collect a new badge. You can collect badges to represent these achievements from the Adventures in Raspberry Pi companion website (www.wiley.com/go/adventuresinrp3E).

Reaching Out

The Appendix explains ways you can take your Raspberry Pi knowledge further, with references to websites, organisations, videos and other resources. Many of those resources include forums where you can ask questions or get in touch with other Raspberry Pi users.

You can also contact me by sending me a message through my website, www.geekgurldiaries.co.uk.

Time to start your adventures!

Adventure 1

You Have a Raspberry Pi. Now What?

IN THE PAGES of this book you discover how to do great things with your Raspberry Pi. You create art and music, programs, games, even create your own jukebox! But first, you need to get your system working.

If you are new to Raspberry Pi, the initial tasks of getting it set up and running might seem a little daunting, but it is not that complicated to do. By setting up the Raspberry Pi yourself you learn more about how it and other computers work. You will encounter technical jargon and procedures that you may not have come across before. In this chapter, I show you how to set up your Raspberry Pi so it is ready for you to use for the first time. I explain what **hardware** and **software** you need, and tell you how to put it

Hardware refers to the physical elements of the computer that you can see and touch. This includes everything inside the computer case, known as components.

Software is the term given to the programs that run on the computer system. Programs are what make the hardware work, for example by making a calculation or organising your files. There are two main types of software: systems software, which runs and manages your computer; and application software, which performs a specific task or function.

all together into a working system. You also find out how to create a backup copy of your system in case you need to replace it at some stage in the future.

What Hardware Do You Need?

Of course, the first thing you need is a Raspberry Pi. If you have used games consoles or computing devices before, you'll notice something different about Raspberry Pi—it doesn't come with a power supply, a charger or any connecting cables. It doesn't have a storage device to keep your programs on either—or even a case!

So, to get started, you first need to get the following hardware together (see Figure 1-1):

- A Raspberry Pi
- A 2A (amp) micro **USB** power adapter
- A USB keyboard and mouse
- A desktop computer or laptop with an SD card reader/writer—this is to enable you to prepare an SD card with the software you need to run your Raspberry Pi
- An 8GB micro **SD card**
- An **HDMI** cable—you will be using this with an HDMI TV or monitor
- A monitor or TV

When I refer to SD cards in this chapter and throughout this book, I am also referring to micro SD cards, which the Raspberry Pi model B+ and subsequent models of boards (for example, Pi 2, Pi 3 and so on) use.

HDMI stands for *High-Definition Multimedia Interface*. HDMI devices are used to transfer video and audio data from a source device—such as your Raspberry Pi—to a compatible HDMI device like a digital TV or monitor.

USB stands for *Universal Serial Bus*. You have probably used a USB port on a computer to plug in a webcam or a portable memory device like a memory stick.

An **SD card**, or *Secure Digital memory card*, stores data or information. SD cards are most often used in digital cameras to store images that can then be transferred to a computer using an SD card reader. A micro SD card is much smaller in physical size, and the Model B+ uses them instead of a standard SD card.

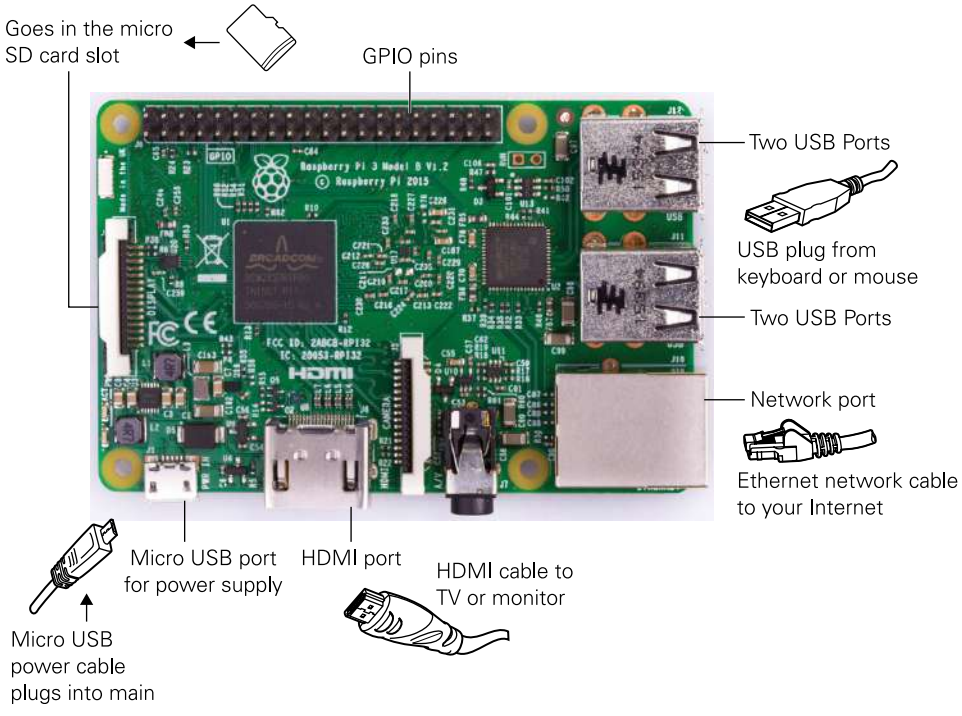


FIGURE 1-1 The essential hardware you'll need before you can use your Raspberry Pi.

What Other Equipment Is Helpful?

The following additional accessories are not vital, but you might want to consider acquiring some of them to improve your Raspberry Pi experience.

- **A case**—To protect your Raspberry Pi from damage and make it easier for you to carry, think about buying a case like the PiBow shown in Figure 1-2, designed and manufactured by Pimoroni (`https://shop.pimoroni.com`). The great thing about this case is that it's colourful and fun, and the ports are also labelled to remind you where each cable should be inserted.

  If you don't want to spend cash on a case, why not create your own by using the Raspberry Pi Punnet? This template can be printed onto card stock, and then cut out and folded into a box. You can really let yourself get creative here and customise your case using pens, paints, stickers or coloured card stock to create a masterpiece. You can download a template for the original Raspberry Pi from this site: `http://squareitround.co.uk/Resources/Punnet_net_Mk1.pdf`.

FIGURE 1-2 The PiBow case can help protect your Raspberry Pi.
Reproduced by permission of Pimoroni

Want a sturdier case? Build one with LEGO blocks! You can find instructions to build the LEGO Raspberry Pi case shown in Figure 1-3 on the official LEGO website, which you can reach through this shortened link: `http://bit.ly/1iF6PNE`.

- **A few spare SD cards**—It's worth having a few extra cards just in case the one you're using becomes corrupted or stops working for any reason. They are also useful for backing up your files and projects—I explain how to do this at the end of the chapter.
- **An SD card reader/writer**—You need an **SD card reader/writer** to enable you to put the Raspberry Pi operating system software onto an SD card. You download the operating system software onto your computer, plug the card reader into a USB port on your computer and use it to copy the OS onto an SD card that you can then load onto your Raspberry Pi. Many desktop computers and laptops are already fitted with an SD card reader and writer but if your computer or laptop doesn't have one, you will have to get an external USB card reader.
- **A Raspberry Pi camera module**—The Raspberry Pi camera module is a Raspberry Pi camera board accessory for the Pi. It connects to the Pi with a flex cable and can be used to take digital images of whatever the camera is pointed at.

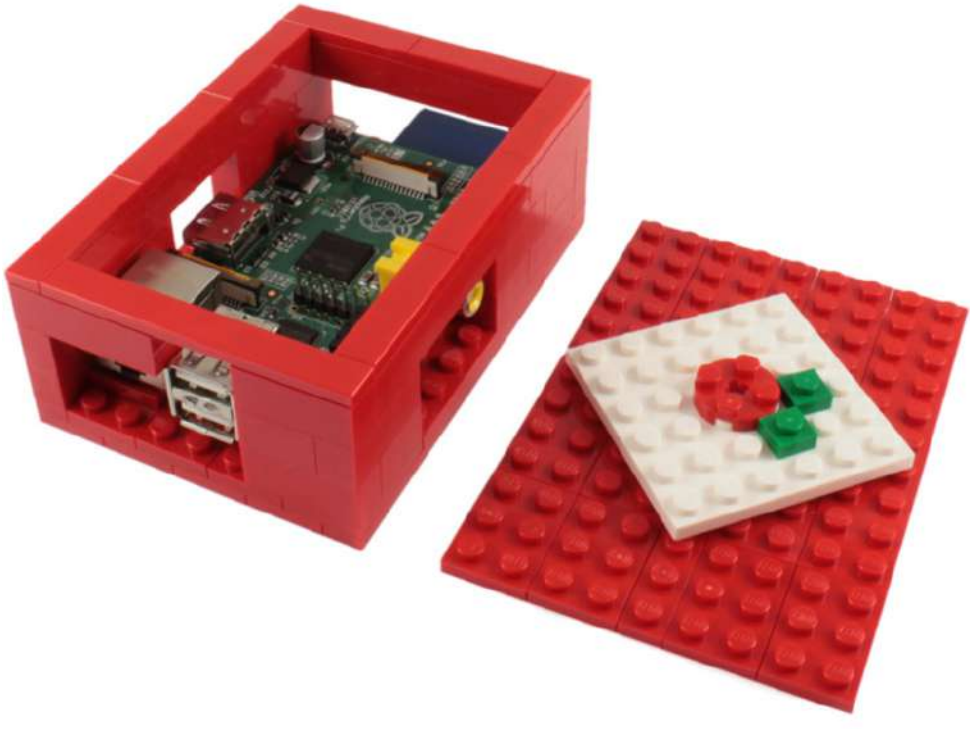


FIGURE 1-3 Build a LEGO case for your Raspberry Pi.
Reproduced by permission of The Daily Brick

Setting Up the Raspberry Pi

Getting your Raspberry Pi up and running takes just three main steps. First, you need to download the operating system software and copy it onto an SD card. Next, you hook up the hardware—the mouse, keyboard and other components. Finally, you install the software onto your Pi and configure a few settings. The next few sections walk you through this process for a smooth launch. Don't worry: Doing the actual steps is much easier than reading these instructions!

For a video that walks you through the steps of setting up your Raspberry Pi, visit the companion website at www.wiley.com/go/adventuresinrp3E. Click the Videos tab and select the SettingUpRaspberryPi file.

Downloading and Copying the Raspbian Operating System

All personal computing devices need an **operating system** (OS) to make them run. You've probably used a computer or laptop before, and the likelihood is that your computer's operating system was Microsoft Windows for a PC, or Mac OS X for a Mac computer or Macbook. The Raspberry Pi can run a number of operating systems, but the OS most people use is **Raspbian**, which is a distribution of the free **Linux** operating system. The projects in this book assume you are using Raspbian on your Raspberry Pi, and the instructions in this section tell you how to download and install it.

Raspbian was created by a community of thousands of volunteers world-wide. You can connect to this community and learn more about Raspbian and Linux at `www.raspbian.org`.

Preparing an SD Card to Store Your Software

A desktop or laptop computer uses a permanent storage device called a **hard drive** to store information and applications. The Raspberry Pi doesn't have a hard drive, however, so your operating system, applications, and information all have to be stored on a removable SD card or micro SD card. This type of storage, known as **flash memory**, is the same as the kind you use with a digital camera to store all your photographs.

Before you plug in all the cables and so on, you first need to prepare (or flash) an SD card with the software the Raspberry Pi needs in order to run. This means that you format your SD card or micro SD card and copy the free Raspbian OS onto it. If you don't do this step, your Raspberry Pi won't recognise it as a storage device (like the hard drive of your computer) from which you can **boot** software. Don't worry if these terms are unfamiliar to you—all will become clear as you read through this section.

MORE ABOUT OPERATING SYSTEMS

An **operating system** (**OS**) is a type of software that allows people to create, store and manage files and applications that contain information on a computer. Examples of popular operating systems include Microsoft Windows, Mac OS X and Linux.

Linux is a free, open source operating system. This means that the programming code it is made with is free and open for anyone to look at and possibly improve on. There are many versions, or distributions, of the Linux operating system available. **Raspbian**, the OS you use on the Raspberry Pi, is a Linux distribution. You may have heard of some other well-known Linux distributions, such as Ubuntu, Debian or Fedora.

The first thing a computer does when you turn it on is to start up, or **boot**, the operating system.

You can buy SD cards with the Raspbian software preloaded onto them. This type of card allows you to get up and running straight away and you can skip the instructions on how to install the Raspbian software. However, I recommend you walk through the installation steps in this chapter, rather than using a preloaded card. It's useful to learn how to complete the formatting process yourself so that you understand how it's done and can start fresh if anything goes wrong.

Your card must be formatted, as described in the following steps, before any software is loaded onto it.

1. The best way to ensure that the card is formatted correctly for use is to download, install and use SD Formatter 4.0 (`www.sdcard.org/downloads/formatter_4`) from the SD Association to your desktop or laptop computer. (The built-in Windows formatting tool only formats the first partition and not the entire disk, so it is important that you use the SD formatter 4.0 tool instead.)
2. To download SD Formatter, follow the link in Step 1, and select either SD Formatter 4.0 for Windows Download or SD Formatter 4.0 for MAC Download. Read and agree to the terms and your download begins. Once the download is complete, extract the file by clicking on Extract All and then run the setup application following the onscreen steps.
3. With the SD Formatter installed on your computer, run the application. Make sure that it has the right drive selected for your card; for example it might be labeled D: or F: (see Figure 1-4).You can find out which drive is your SD card by looking in My Computer on a Windows computer or using Finder on Mac OS X.
4. Click the Option button and select FULL (erase) from the drop-down menu. When you are ready, double-check that you have the correct drive selected, and click Format.

The program wipes all data from the card, so make sure you select the correct drive!

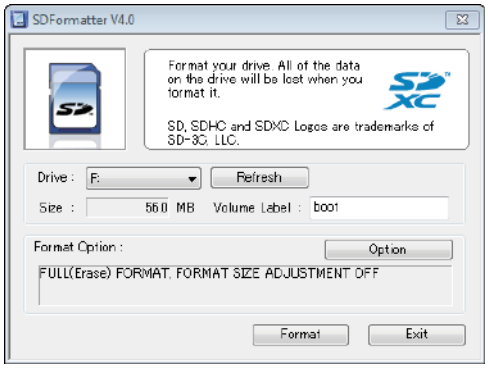


FIGURE 1-4 Formatting an SD Card using the SD Formatter application

Making It Easy with NOOBS

With your SD card formatted, you're ready to copy the Raspbian software onto it. The New Out Of Box Software (NOOBS) produced by the Raspberry Pi Foundation allows you to copy the files you need straight onto the SD card like you would do with photo or document files. It gives you the option of selecting which operating system you want to install, and even provides recovery should you accidentally delete all your software files.

If you are using a micro SD card, then you may need an adapter so that it fits an SD card reader slot. The official Raspberry Pi NOOBS card has a micro SD card inside it, so you can use it as an adapter.

All the projects in this book are designed to run using the Raspbian OS included in the NOOBS software. I recommend ensuring that you use the latest version of NOOBS before starting any projects in this book; otherwise you may have difficulty getting some of the projects to work.

First, you need to download NOOBS onto a desktop or laptop computer with an SD card reader. After you download the software, you save it to an SD card for use with your Raspberry Pi. The following steps walk you through the process:

1. Navigate to the Raspberry Pi website at `www.raspberrypi.org` and click the Downloads tab at the top of the page. The New Out Of Box Software that you want to download is at the top of the page. Click the link to select the latest NOOBS .zip file.

   The download file is a compressed file. Save the compressed file to your desktop or laptop computer, and then extract the files by right-clicking on the file and

selecting the Extract All option (on a Microsoft Windows computer). You will then be given the option to extract the files to a directory or folder of your choice so that you will easily be able to find it after the extraction is complete, as shown in Figure 1-5.

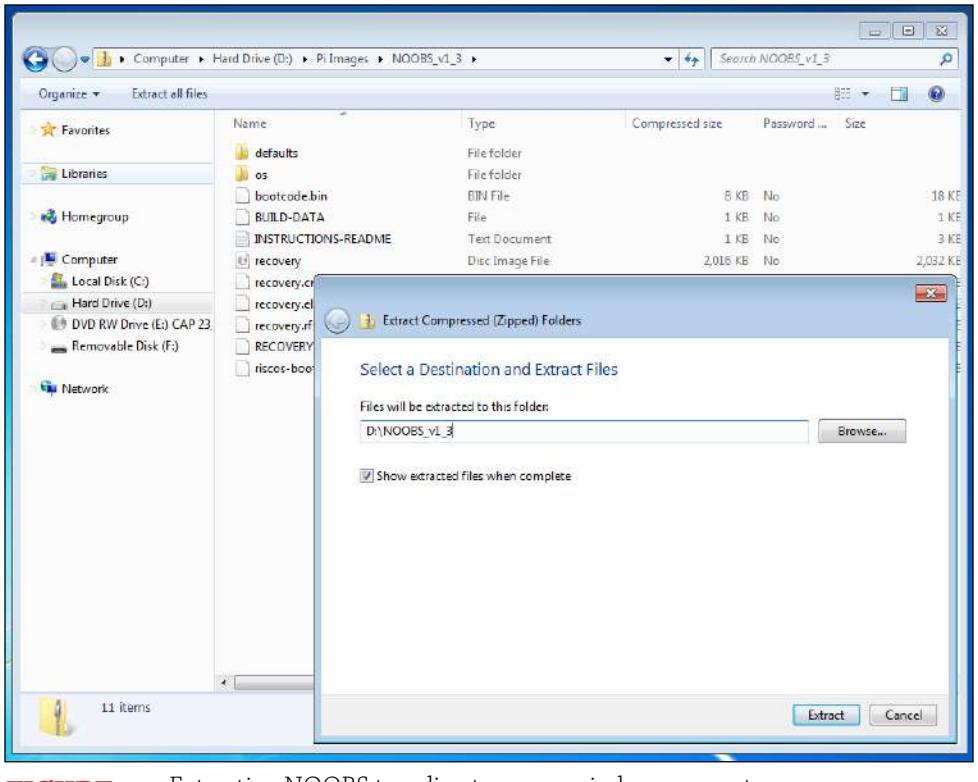


FIGURE 1-5 Extracting NOOBS to a directory on a windows computer

2. Place your formatted card into the card reader slot on your desktop computer or laptop. Now copy the extracted NOOBS files from the directory or folder on your computer and paste them onto your newly formatted SD card. You can do this either by dragging the files from one window to another, or by highlighting them all with your mouse, right-clicking, and copy/pasting the files onto the card.

You should always download the latest version of NOOBS as the software is being updated all the time. The latest version is usually listed at the top of the page with a version number.

Plugging in the Hardware

Now it's time to get your Raspberry Pi up and running. Find yourself a solid surface, like a desk or table, big enough to hold all your equipment. Make sure it's near some main plug sockets. Ideally, you should also have access to a network device like a router because you will likely want to access the Internet on your Raspberry Pi at some point, but you don't need access to a router to set up your Pi.

As noted in Step 5 of the following instructions, do not plug in the power supply until you have completed the first four steps.

Set up your Raspberry Pi using the following steps:

1. Place the card with the NOOBS files you have loaded onto it into the card slot of your Raspberry Pi on the back.
2. Plug a USB keyboard and mouse into the USB slots on the Pi.
3. Connect the HDMI cable from your TV or monitor to the HDMI port on the Pi and turn on your TV or monitor. Some TVs and monitors accept input from lots of different sources, so you may have to make sure that you set your TV or monitor to the HDMI setting. Some TVs and monitors will auto-detect the HDMI when you power up your Raspberry Pi.
4. If you think you'll be using the Internet on your Pi, connect a network cable to the Ethernet port.
5. Start the Raspberry Pi by plugging in the micro USB power supply. It is important to do this step *last*, as the Pi does not have a power button so the boot process begins as soon as you plug in the Pi.

   Okay, your Pi is running!

Installing and Configuring the Software

When you power up your Raspberry Pi with a NOOBS SD card for the first time you need to complete the setup of the software.

The new system loads and begins to resize the SD card's partition. Partitions are used to separate parts of a storage device from each other. Once NOOBS completes this task your card has three partitions: one called the boot partition, which holds all the files needed to start and run your Pi; one called the recovery partition; and one to store any files that you create, or applications that you add later.

The NOOBS software gives you the choice of installing one of several different operating systems, including RaspBMC and Pidora as well as Raspbian. To use the projects in this book, you should install Raspbian. See Figure 1-6.

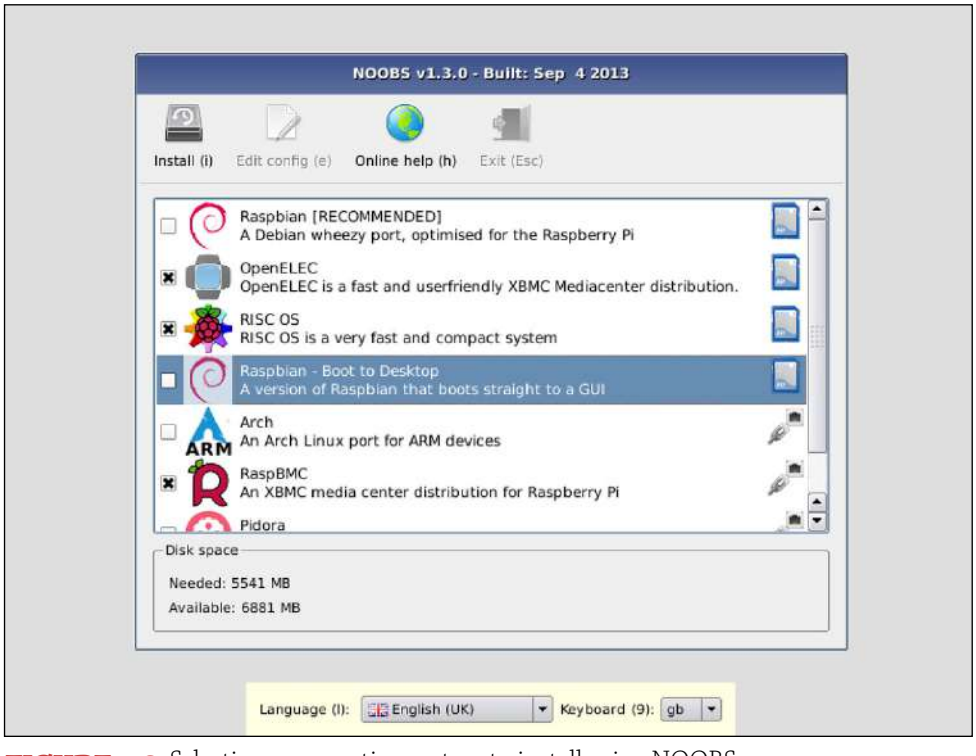


FIGURE 1-6 Selecting an operating system to install using NOOBS

In future releases of NOOBS you will be able to install more than one operating system at a time from the list provided. You might like to try another OS, such as Windows 10 IoT (*Internet of Things*) or Ubuntu Mate, at a later date. You can always use NOOBS on a different SD card, configured for another OS.

Follow these steps to install Raspbian:

1. Select the operating system that you want to install—Raspbian—and click Install. At this point you can also change any language settings.
2. You see a warning asking if you are sure you want to install the operating system as it will overwrite any file system already on the card. Click Yes.

3. Wait for the operating system to be installed by following the progress bar on the screen. Once complete, the Pi boots to the graphical user interface for the Raspbian operating system.

   After installation of the operating system, the desktop automatically loads. You may want to configure some of the settings for your software. You do not need to change any of the settings at this time, as you have the option of coming back to this window (see Figure 1-7) whenever you use your Pi by clicking the Menu icon and selecting Preferences ➪ Raspberry Pi Configuration.

   - **Localisation**—Selecting this option lets you set the language and time zone for your Raspberry Pi. For example if you are in the UK, you may want to set your language to English and time zone to GMT—Greenwich Mean Time.
   - **Interfaces**—In this menu you can enable or disable the interface options. For example, if you have a Raspberry Pi Camera (a Raspberry Pi accessory), you should enable it here so that you are able to use it.
   - **System**—Use this option to disable auto login and change the default password to add some security measures to your setup.

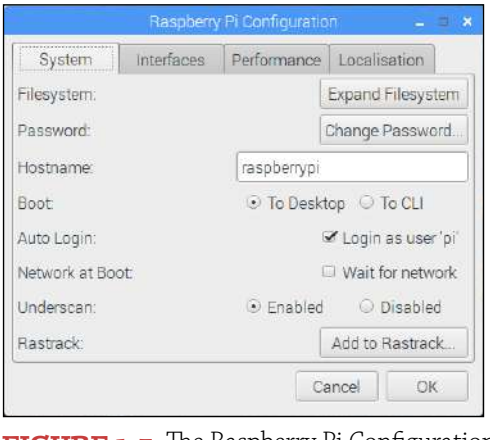


FIGURE 1-7 The Raspberry Pi Configuration window

Congratulations! You've successfully installed the software required to be able to use your Raspberry Pi! Read on to explore the **graphical user interface** (see Figure 1-8) of Raspbian.

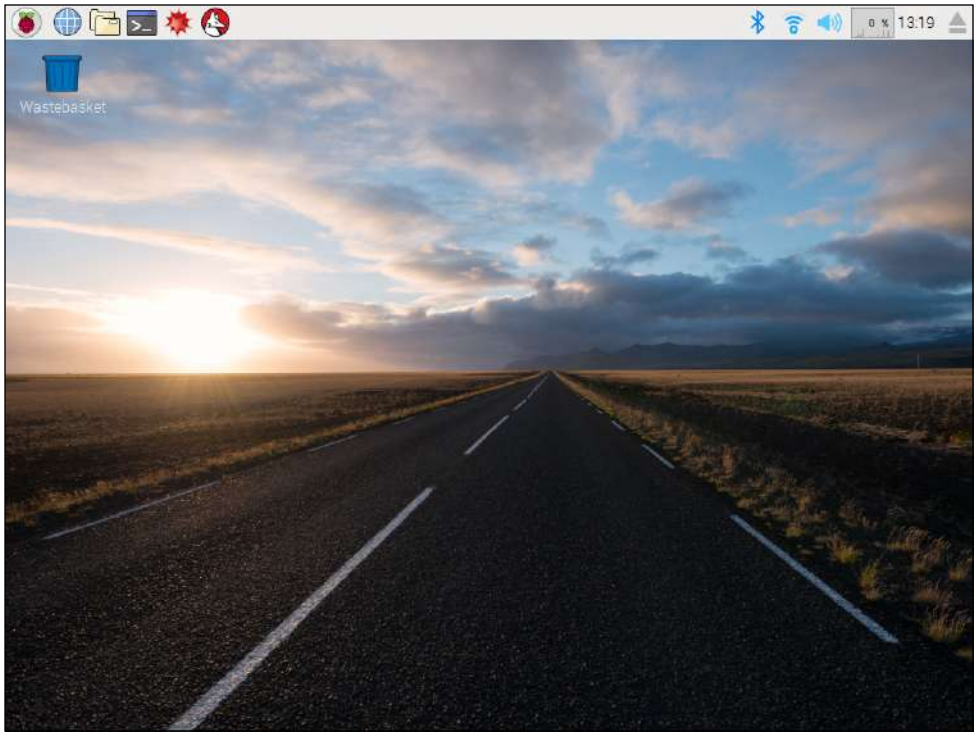


FIGURE 1-8 The Raspbian desktop

You may be used to using desktop computers or laptops that have windows, a mouse pointer, and a desktop. This is typical of a **graphical user interface**, or **GUI**.

USING NOOBS FOR RECOVERY

If anything goes wrong at any time after first running your Pi with NOOBS and installing an operating system—for example, if you manage to corrupt your file system, or if you would like to try one of the other operating systems packaged in with NOOBS—simply hold the Shift key when booting your Raspberry Pi and you go to the recovery screen.

Exploring the Desktop in Raspbian

As you just saw, plugging in the power supply and booting the Raspberry Pi loads the graphical user interface of the Raspbian operating system. You will see the default Raspbian desktop shown in Figure 1-8, with a taskbar across the top with the time on the far right, and the main menu button icon (containing the Raspberry Pi logo) on the far left. Some of the most commonly used applications can be launched from the Main Menu such as Scratch (see Adventure 3), Python (see Adventure 5), and a web browser that you can use to browse the World Wide Web if your Pi is connected to the Internet via an Ethernet cable or Wi-Fi. There are even some games for you to try out using PyGame. Spend a few minutes checking out what applications are available by clicking on the main menu and then each of the submenus in turn. Figure 1-9 shows the applications available under the Programming submenu.

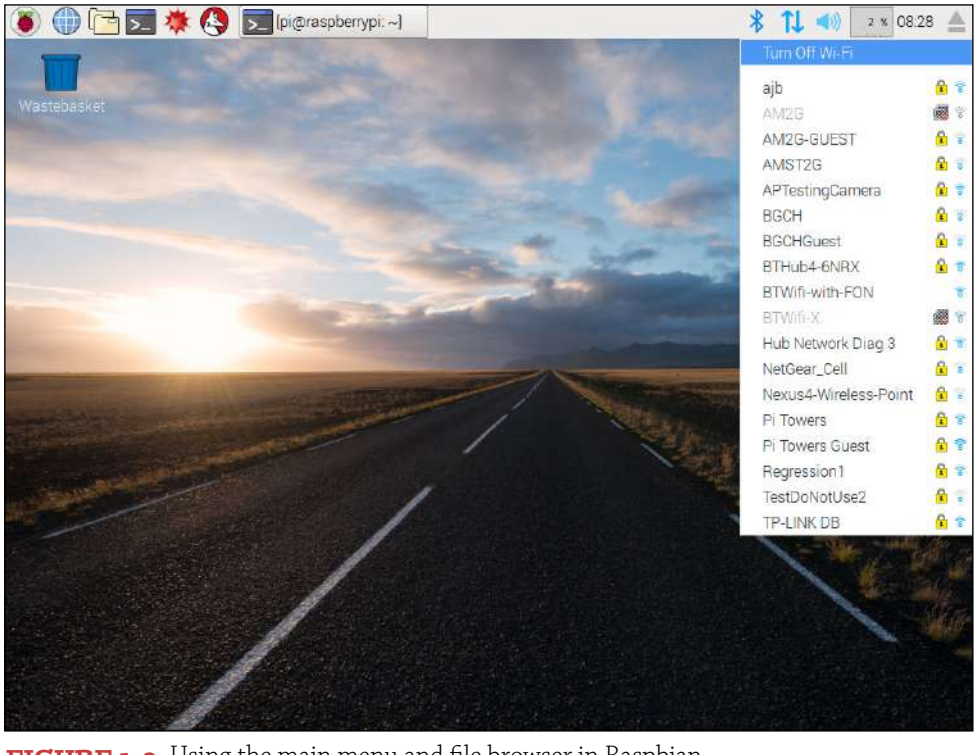


FIGURE 1-9 Using the main menu and file browser in Raspbian

To learn a little about how your Raspberry Pi is set up, try the following steps:

1. Click the main menu icon in the left corner of the taskbar.
2. From the main menu, select Accessories ➪ File Manager to open the file browser. If you are logged in as the user ***pi***, the browser displays the contents of your home directory (`/home/pi`). This directory is fairly empty as you are yet to begin your programming adventures with Raspberry Pi. You see a directory called `Desktop`. Right-click with your mouse on the desktop to open a drop-down menu and select Create New ➪ Empty File. Name the file hello and click Ok. You see an icon appear on the desktop, and if you double-click the directory folder Desktop in the file manager window, you see that it contains the hello file too.

Shutting Down Your Raspberry Pi

When you shut down your Raspberry Pi, it is very important that you don't simply remove the power supply, but make sure you always instruct the OS to shut down safely. The Raspbian OS has a Shutdown button for shutting down your Raspberry Pi cleanly if you are using the GUI—just click the main menu icon followed by Shutdown to open a menu with options to shut down, reboot and log out.

However, if you are not using the GUI, you don't see this icon and need to use a text command to shut down your Raspberry Pi. You can learn how to do this in Adventure 2, in the section "Using Shutdown and Restart Commands".

Connecting to a Wi-Fi Network

To browse the Internet or to download and install extra applications or programming libraries, you need to connect your Raspberry Pi to the Internet. You can do this either by using an Ethernet cable connected to an Internet-enabled router or by configuring the Wi-Fi settings as described in the following steps:

1. Wi-Fi connections can be made via the Wi-Fi network icon on the taskbar, located next to the volume setting icon. If you are using a Raspberry Pi 3, or an earlier model with a Wi-Fi adapter plugged into a USB port on the Raspberry Pi, click this icon to see a list of available Wi-Fi networks as shown in Figure 1-10
2. While the Raspberry Pi is searching for a wireless network, you see a `No Apps found - scanning...` message.

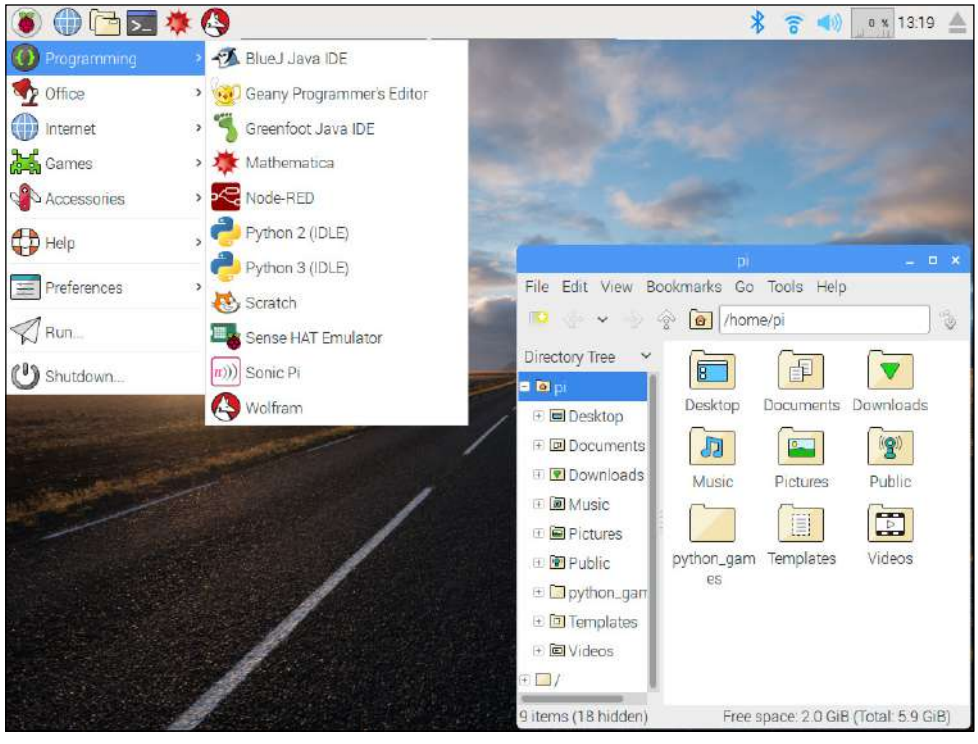


FIGURE 1-10 Connecting to a Wi-Fi network

3. Click the network that you want to connect to. Once you've made your selection, you may be asked to enter a password or network key to continue with the connection.

4. Enter the password or network key and press Ok and wait for the network icon to stop flashing and show the signal strength.

Backing Up an SD Card Image

You have only used your Raspberry Pi once so far, but you have already made changes to the configuration of the operating system. As you move through the projects in this book, you may want to back up your SD Card to a computer or laptop as you go along to make sure you don't lose any of your work if your SD card or micro SD card stops working for any reason. It is very easy to do this using a free Windows application called Win32 Disk Imager. Download this application from `http://sourceforge.net/projects/win32diskimager` before continuing with the following steps.

1. If you have not yet shut down your Raspberry Pi, follow the instructions in the previous section to do so now, or by typing the following command into a Terminal widow:

```
sudo halt
```

2. Take your SD card out of your Pi and place it into your computer's SD card reader.
3. Run the Win32 Disk Imager on your desktop computer by locating the folder into which you extracted it and double-clicking the application icon.
4. In the Image File dialog box (see Figure 1-11), type a name of your choice for your backed-up image—Adventures_In_Pi, for example.
5. Click the folder icon to browse to a location on your computer where you would like to save your backed-up image.
6. Click Read to copy the image from the SD card to your computer.
7. Wait for the progress bar to become full before closing the software and removing your SD card.

In the future, when you become more skilled, it's likely you will have multiple cards with different project images on them. You can save all these to your computer separately. It's a good idea to back them up in this way, to keep all your files safe. It is also best to store one image on one card and use separate cards for each project you work on.

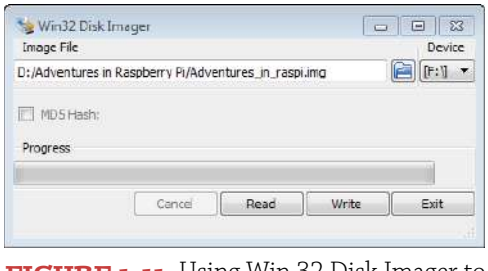


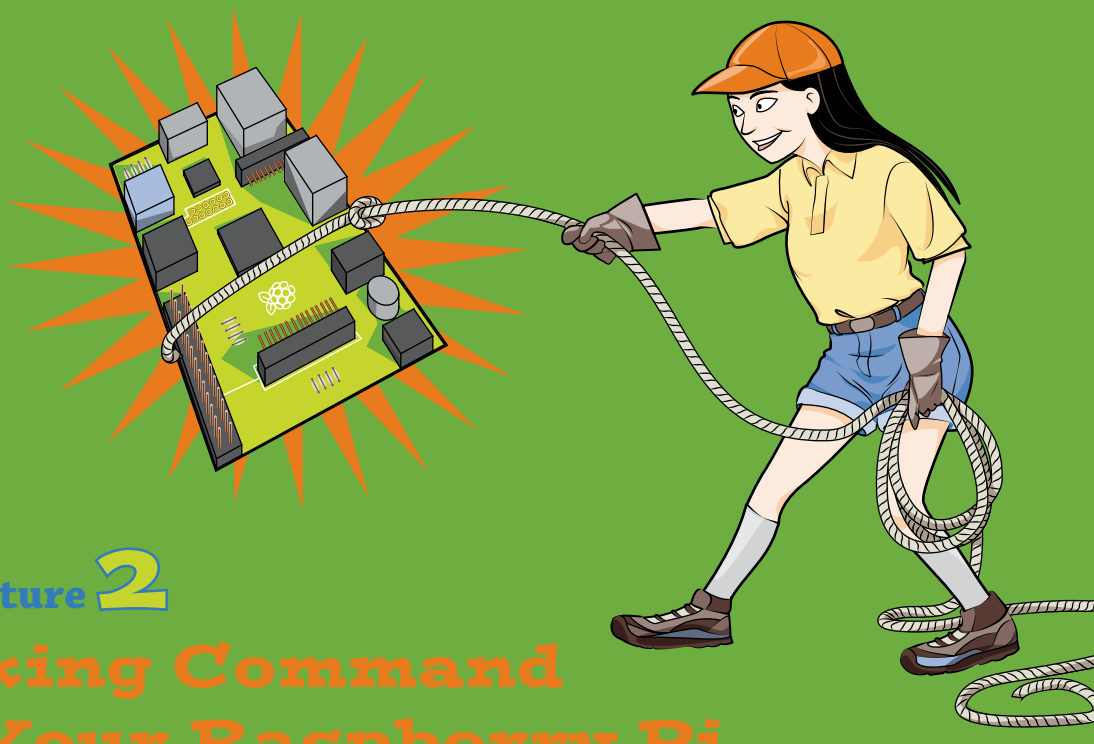
FIGURE 1-11 Using Win 32 Disk Imager to make a backup of an SD card or micro SD card

Raspberry Pi Startup Command Quick Reference Table	
Command	**Description**
`sudo`	Gives the user root or super user permissions.
`sudo halt`	Shuts down (halts) the power to the Raspberry Pi.
`sudo reboot`	Shuts down the power to the Raspberry Pi and then restarts it.

Achievement Unlocked: Your Raspberry Pi is up and running!

In the Next Adventure

In Adventure 2, you learn about the power of text commands. You use commands to instruct your Raspberry Pi and discover how to navigate the file system, launch programs and download more applications to use with your Pi.



Adventure 2
Taking Command of Your Raspberry Pi

NOW THAT YOUR Raspberry Pi is up and running, how do you tell it what you want it to do? Well, there are a number of ways to communicate with computers, depending on what operating system (OS) it uses. Many of today's OSs—like Microsoft Windows and MAC OS X—use graphical user interfaces (GUIs). These have icons that you click with a mouse, making the computer very easy to use. Raspbian, the OS you are using on the Raspberry Pi, has a GUI (see Figure 2-1), which loads after boot.

If you use the Raspbian GUI, you simply click the icons to access the software programs. As an alternative to the GUI, you can communicate with the Raspberry Pi using text-based instructions, known as **commands**, without the need for a GUI. This form of communication is called a **command-line interface**, and the window into which you type the commands is called a **terminal**. Although the GUI might be more user friendly and easier to understand than text commands, text commands can be faster when you become more experienced in using them. You can also do more things with text commands, such as writing scripts, which are small programs that combine a series of commands to carry out routine computing tasks. In later adventures you write your own scripts to make something happen. When you first load a command-line terminal, you see the $ prompt on the screen, which tells you that the computer is waiting for you to type a command.

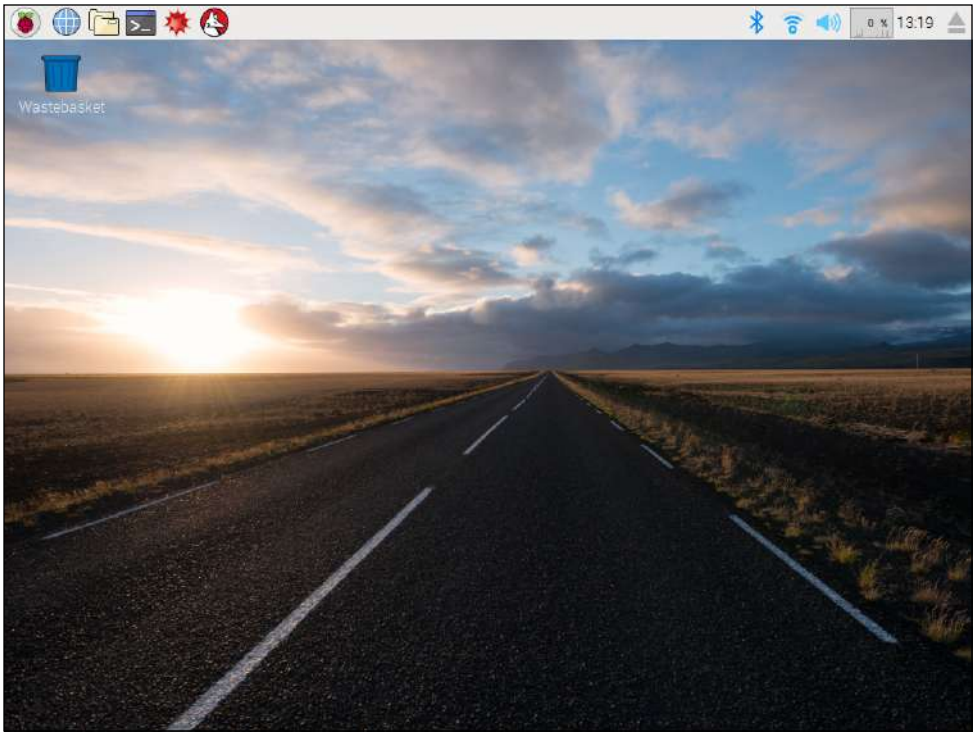


FIGURE 2-1 The Raspberry Pi GUI

A **command-line interface** (**CLI**) allows you to communicate with a computer using text commands.

A **terminal** is a screen window that gives you access to the command-line interface. The graphical Terminal is an example.

By pressing CTRL+ALT and one of the function keys between F1 and F6, you can switch among six different virtual terminals. You can log in to any of the terminals and type commands at the prompt. Press CTRL+ALT+F7 to go back to the desktop environment.

Here's an example of a way to use the command line: Suppose you have written a book report on Hamlet and you want your Raspberry Pi to delete it. You might try typing the following command at the prompt:

```
Delete the file hamlet.doc
```

If you do this, the file won't be deleted, and you get this error message:

```
bash: Delete: command not found
```

You get the error because you can't just type any command and expect the Raspberry Pi to understand it! It only responds to a command when it is composed of a particular set of defined words that it already understands. These commands can be very specific and often need to be typed in a certain order to work. To delete your report, you must use a command that Raspberry Pi understands. In this case, you need to use the `rm` (remove) command:

```
rm hamlet.doc
```

If you learn these commands, you aren't limited to using a GUI, and you can navigate file systems and program the computer by using simple text commands. This can sometimes be faster and more convenient than doing the same tasks in a GUI. Many of the projects and tutorials in this book use some text commands, so this chapter introduces you to some basic commands that help you save some time.

Exploring the Terminal

In this section, you become familiar with some common Linux text commands by using the graphical Terminal within the Raspbian desktop environment, as shown in Figure 2-2. You can open this terminal in one of two ways:

- Open the Terminal from the taskbar by clicking on the icon with your mouse.
- Select Terminal from Accessories in the main menu.

To see a video about using the Terminal and the other tasks in this adventure, visit the companion website at `www.wiley.com/go/adventuresinrp3E` and select CommandLine.

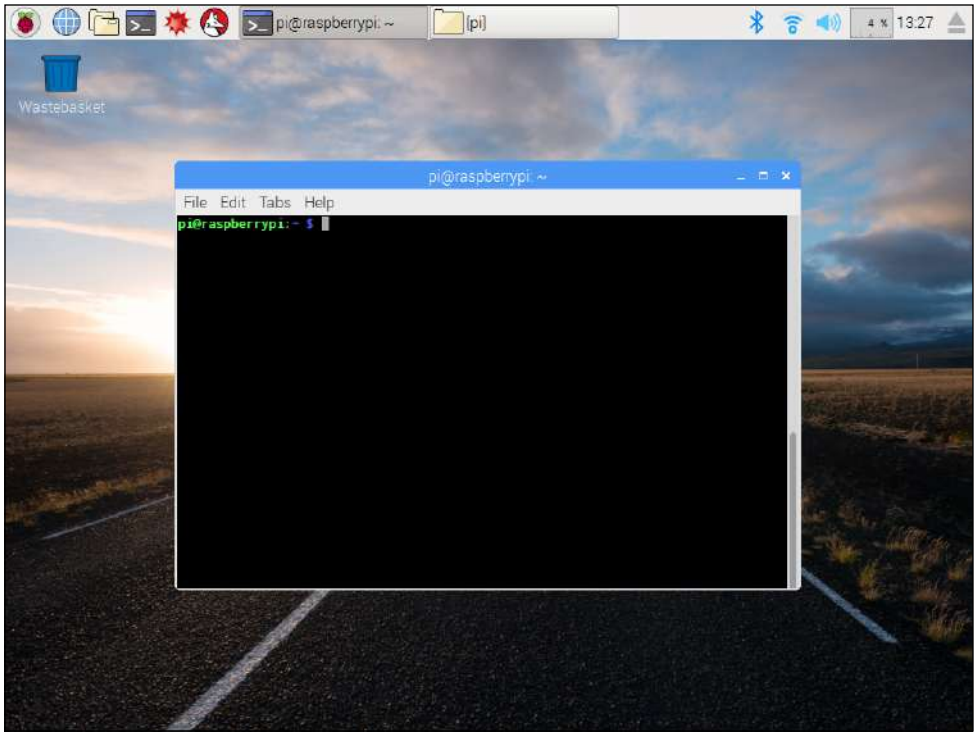


FIGURE 2-2 The Terminal open in the Raspbian Desktop Environment

When Terminal is loaded, you see a black screen with the line `pi@raspberrypi: ~ $`, as shown in Figure 2-2. Let's break down the parts of this line:

- `pi` refers to your username. You are logged in as the user pi.
- `raspberrypi` is the **hostname** of your device; this name identifies it on a network. So you are the user pi on the device *raspberrypi*.
- After the hostname comes the current directory, which is represented by the `~` symbol (this symbol is called a tilde). This is a short way of referring to your home directory, which is `/home/pi` when written in full.
- Finally, the Raspberry Pi asks you what you want it to do, by displaying the `$` symbol as a prompt to enter a text command (see Figure 2-3).

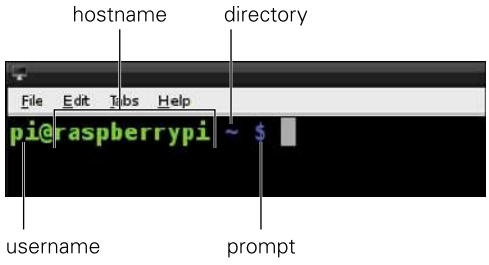


FIGURE 2-3 Breakdown of `pi@raspberrypi ~ $`

Now try to interact with the computer by typing a command: type **date** into the terminal and press Enter.

The Raspberry Pi tells you the date and time without needing to display a graphical clock, and the command provides you with a response quickly.

> A **hostname** is a word that identifies a computing device on a network. The hostname of the Raspberry Pi is `raspberrypi`.
>
> 
>

Commands for Navigating Through Your File System

One of the most important tasks of an OS is to organise files and folders (called directories in Linux). These files and directories are organised into a tree-like structure, in which directories can contain other directories and files. The File Manager, shown in Figure 2-4, is a tool that lets you see what that structure looks like graphically. You can access File Manager by clicking the icon next to the main menu at the top of the desktop environment, followed by Accessories and then File Manager.

You can use simple commands to navigate through the file system using the command line. As with any navigation, you need to know where you are—your starting point—before you can find your way forward. Type **pwd** at the terminal prompt and press Enter:

```
pi@raspberrypi ~ $ pwd
```

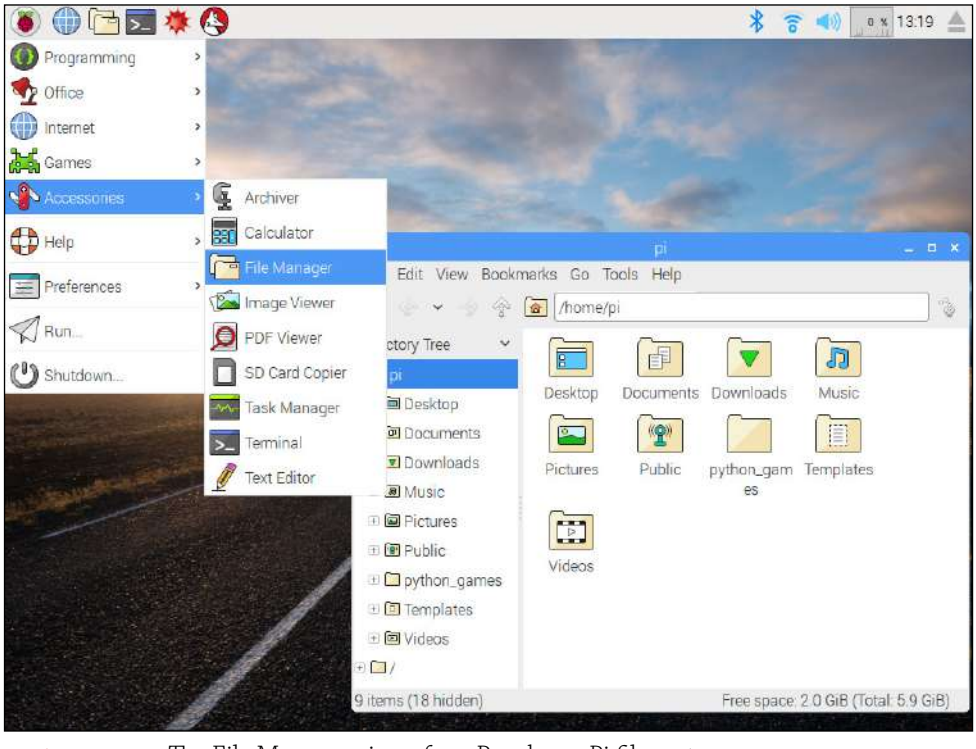


FIGURE 2-4 The File Manager view of my Raspberry Pi file system

The Raspberry Pi responds by displaying this line on the screen:

```
/home/pi
```

The `pwd` command asks the Raspberry Pi to print the working directory, or show which directory you are currently working in. The `/home/pi` response that appears shows that you are currently located in the `pi` directory, which is inside the `home` directory.

As you can see in Figure 2-5, if you give the command `ls`, it tells your Raspberry Pi to `output` a list of the files and directories in the current directory. To see a list of files and directories that are in `/home/pi` (the `pi` directory in your Raspberry Pi's home directory), type `ls` at the `$` prompt and press Enter:

```
pi@raspberrypi ~ $ ls
```

Output refers to the data that your computer gives in response, after you have typed in a command.

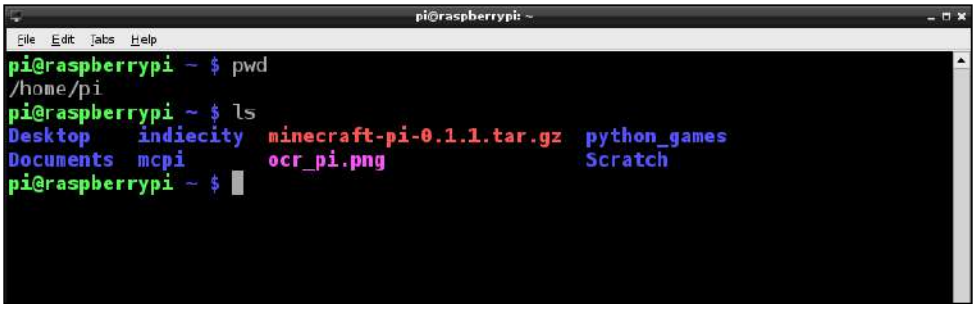


FIGURE 2-5 Using the **ls** command to show a list of the files and directories on my Raspberry Pi

Figure 2-5 shows that inside this particular user's `/home/pi` directory there are six directories listed in blue (such as Desktop and Scratch), as well as one file in pink (`ocr_pi.png`) and one in red (`minecraft-pi-0.1.1.tar.gz`). This list does not really tell us much about those files or directories. To learn more about the contents of the directory you are in, type **ls -l**. You are still using the `ls` command, but this time adding the `-l` (which stands for *long*) **parameter** or option. This tells your Raspberry Pi to show the list in a longer, more detailed format:

```
pi@raspberrypi ~ $ ls -l
```

Parameters modify the way that the standard command works (a bit like ticking a tick box in a GUI program). Most Linux commands have lots of parameters that modify the way that they work. Just to be clear: the `-l` parameter is a lowercase L, not the numeral 1.

As you can see in Figure 2-6, the Raspberry Pi now gives you more information about the files and directories that were listed in Figure 2-5. This information includes the size of the file, the date it was created, the owner of the file and what kind of permission you have to access it.

To move between directories or folders in the tree-like structure, you can use the `cd` (***change directory***) command. Try moving into the `Desktop` directory by typing **cd Desktop** at the prompt:

```
pi@raspberrypi ~ $ cd Desktop
```

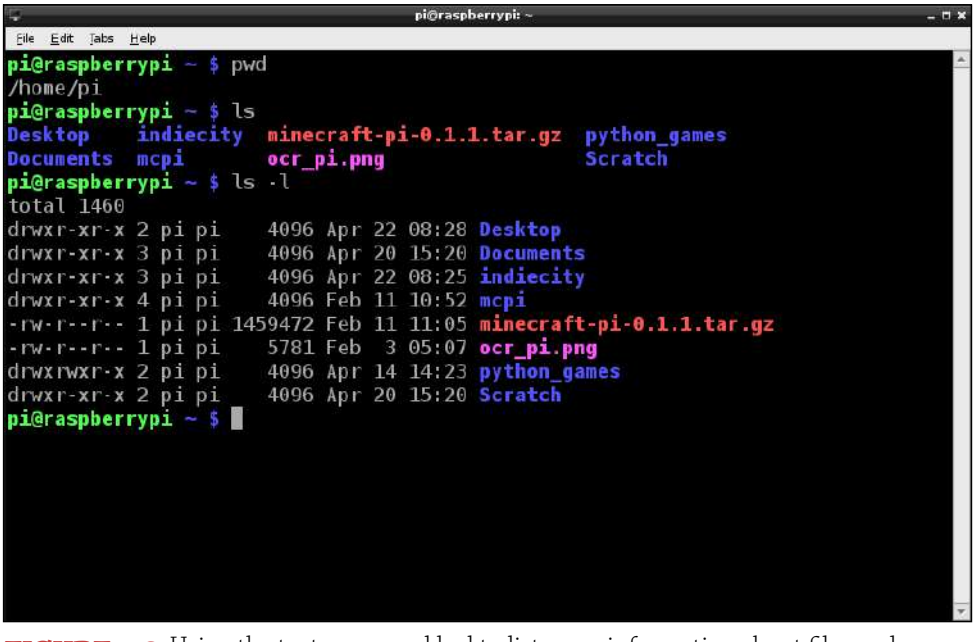


FIGURE 2-6 Using the text command ls -l to list more information about files and directories

The next prompt from Raspberry Pi reads like this:

```
pi@raspberrypi ~/Desktop $
```

Notice how `~/Desktop` has appeared as part of the prompt. This reminds you that you are now in the `Desktop` directory, which in turn is inside your home directory.

You are now inside `/home/pi/Desktop`.

To go back to `/home/pi`, type the command **cd ..** (that's `cd` followed by a space and two full stops).

```
pi@raspberrypi ~ $ cd ..
```

The `cd ..` command moves you up the directory tree to the parent directory. For example, if you are in `/home/pi/Desktop` and type **cd ..** you move upwards through the file system to `/home/pi`. To check where you are in the file system at any time, type **pwd** to see your current directory, as shown in Figure 2-7.

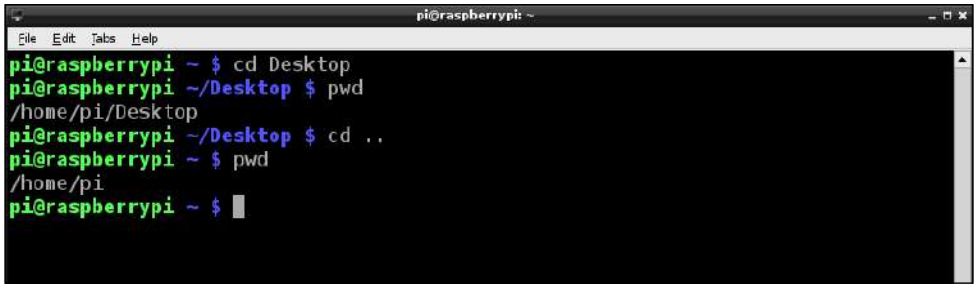


FIGURE 2-7 Navigating the file system in Raspberry Pi's Terminal

If you are ever lost in the file system and can't remember where you are, just type **pwd** to find out.

Understanding sudo

When you are logged into Raspberry Pi as the *user pi*, you have only limited access to the ability to perform tasks on the device. This restriction prevents you from accidentally deleting important files. Sometimes, however, you need to do things that affect the whole system, such as installing a new program or adding a new user. The `sudo` command lets you temporarily act as the *super user* (or *root user*) and gives you permission to do whatever you want on the system. This includes deleting every file on your disk, so you must be very careful when you are logged in as sudo!

You may need to use `sudo` for some applications. Can you think why?

Some applications require the ability to make changes to protected parts of the system or to interact with protected devices, and so must run as an administrator. For instance, if you run the `apt-get` command to install or upgrade an application you must run it with `sudo` or else it fails because it doesn't have permission to update the necessary files.

Launching Programs from the Command Line

You can use text commands to launch programs from the command line, too, which is often quicker than navigating the main menu and clicking an icon. It is also handy to be able to do this if you do not have a mouse plugged in.

Try this. In Terminal, type the following command at the `$` prompt:

```
leafpad
```

The leafpad application on the Raspberry Pi opens. (The leafpad application is a text editor in which you can enter text. It is included with your Raspberry Pi installation.)

Managing Files and Directories

You may sometimes want to create files, or copy, move or delete them. The following Linux commands are useful for managing your files:

- `cat`—displays the contents of the text file
- `cp`—makes another copy of a file
- `mv`—moves a file to a new location
- `rm`—deletes (removes) a file
- `mkdir`—makes a directory
- `rmdir`—deletes (removes) a directory
- `clear`—allows you to clear the terminal

Type the following commands into the Terminal window, in the order they're given here. See if you can explain what is happening at each step:

```
pwd
cd desktop
ls
touch hello
leafpad hello
rm hello
cd ..
```

Installing and Updating Applications

The New Out Of Box Software, or NOOBS, that you learned about in Adventure 1 includes some applications that are installed along with the Raspbian operating system. You can see these application icons on the desktop and also in the main menu. In future adventures you use Scratch and Python 3 (IDLE), which are already installed.

If you connected your Raspberry Pi to the Internet as described in Adventure 1, you can use text commands to download, install and update additional applications that you may want to use.

Downloading and Installing Applications

It's easy to find new applications and install them on your Raspberry Pi. Try typing the following command in a terminal and then pressing Enter:

```
sudo apt-get install vlc
```

Video LAN, or vlc for short, is an application that enables you to play media files on your Raspberry Pi. The `sudo apt-get` command tells Raspberry Pi to use the Internet connection to find and install the application on your Raspberry Pi. This command requires sudo because by installing a new application you are asking the Raspberry Pi to change system files.

You see the terminal window fill with writing and, after a few seconds, the terminal asks you to check that there is enough space on your SD card to install the application. At this point you can either press the Y key for yes, to continue with the installation, or the N key for no, to cancel the installation. Figure 2-8 shows the screen display for the vlc installation process. About halfway down the screen, you see the question "Do you want to continue [Y/n]?" and see that I answered yes (y) to install the application.

Remember from earlier in this chapter that you can use a command to open an application. After you have installed vlc, you can open it and see it work by typing **vlc** into a terminal window, followed by the name of a video or sound file. Try typing the following command in a terminal and then pressing Enter:

```
vlc /home/pi/python_games/match1.wav
```

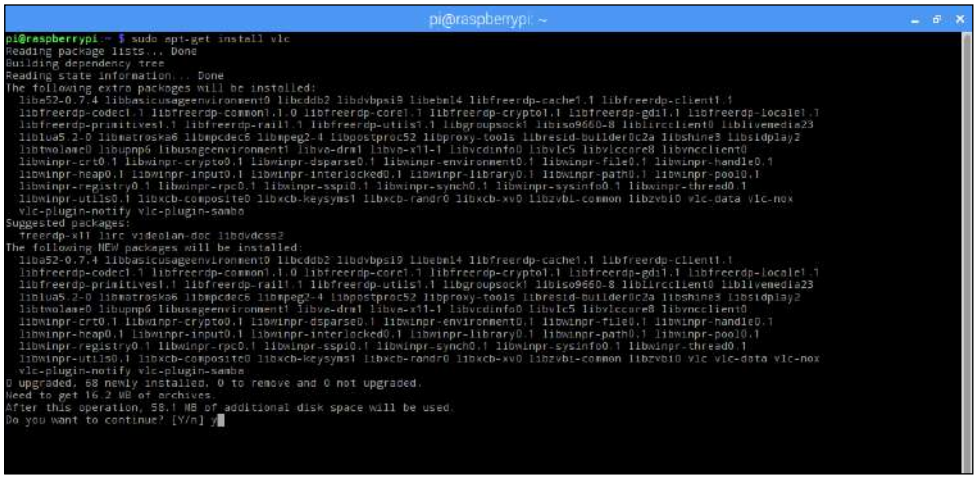


FIGURE 2-8 Using the `apt-get install` command to download and install the vlc application

vlc opens and plays the `.wav` sound file. Don't forget to plug in headphones or speakers if your monitor or TV does not have sound so you can hear, as well as see, your media playing!

Learning More About an Application

Each Linux application or command has a "manual" file that gives a description of the application and lists the options or features that are available. To read the manual for any application, use the command `man` followed by the name of the application. For example, to see the manual for the vlc text editor, simply type **man vlc**. Figure 2-9 shows the manual for the vlc application.

The manual lists the options available with the application and shows you any extra functions that are available for you to use. For the vlc application, for example, you can choose to repeat all media files in a playlist by using the command vlc -L It is always a good idea to read an application's manual, and it can be very helpful if you forget the order in which you need to write the commands.

Upgrading Your Apps

It is good practice to upgrade the applications that you have installed approximately once every two weeks, or before you download and install a new application. Upgrades for an application may provide new features, correct "bugs" that have been causing problems in the application and resolve security issues that may threaten your system.

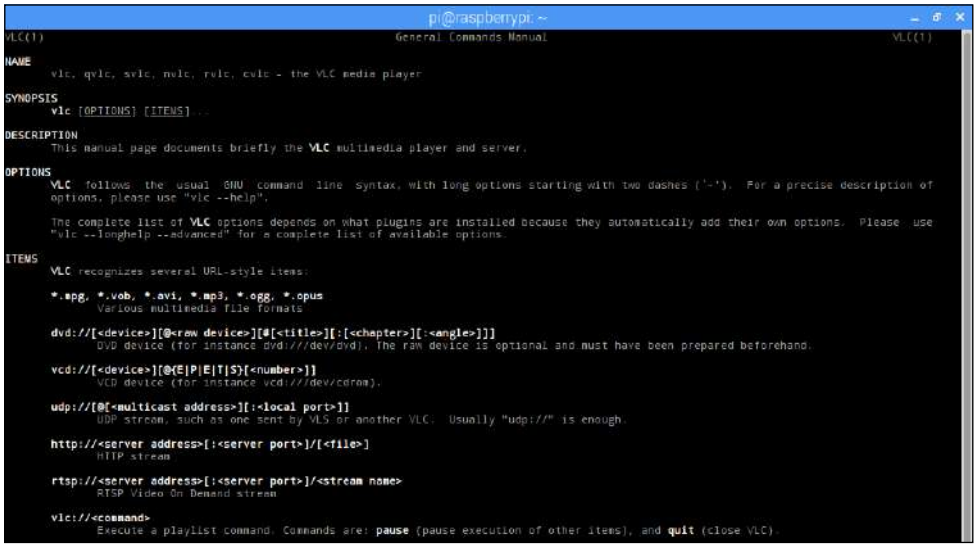


FIGURE 2-9 Accessing an application's manual to learn more

To upgrade your applications, you should first type the following command to download information about any new versions of applications that are available:

```
sudo apt-get update
```

Next, type the following command to actually install the upgrades:

```
sudo apt-get upgrade
```

Editing Files

The nano command opens a text editor, which is an application that allows you to edit text files. This program is useful if you want to make changes to lines of code or individual settings inside a file. The following instructions walk you through the stages of using nano to create and edit a text file (see Figure 2-10).

1. Create a text file on the desktop. To do this, first use the cd command to navigate to the Desktop directory, and then use the nano command, like this:

```
pi@raspberrypi ~ $ cd Desktop
pi@raspberrypi ~/Desktop $ nano hello
```

2. The nano program opens a text file called hello. You can type anything you like into this text file—as you can see in Figure 2-10, I typed "Hello Raspberry Pi Adventurers!" Try typing your favourite quote from a film or the lyrics of a song.

3. To exit the text editor, press CTRL+X. The following message appears:

```
Save modified buffer (ANSWERING "No" WILL DESTROY CHANGES) ?
```

If you want to save the changes you have made to the file, press Y for yes. If you do not want to make any changes, press N for no.

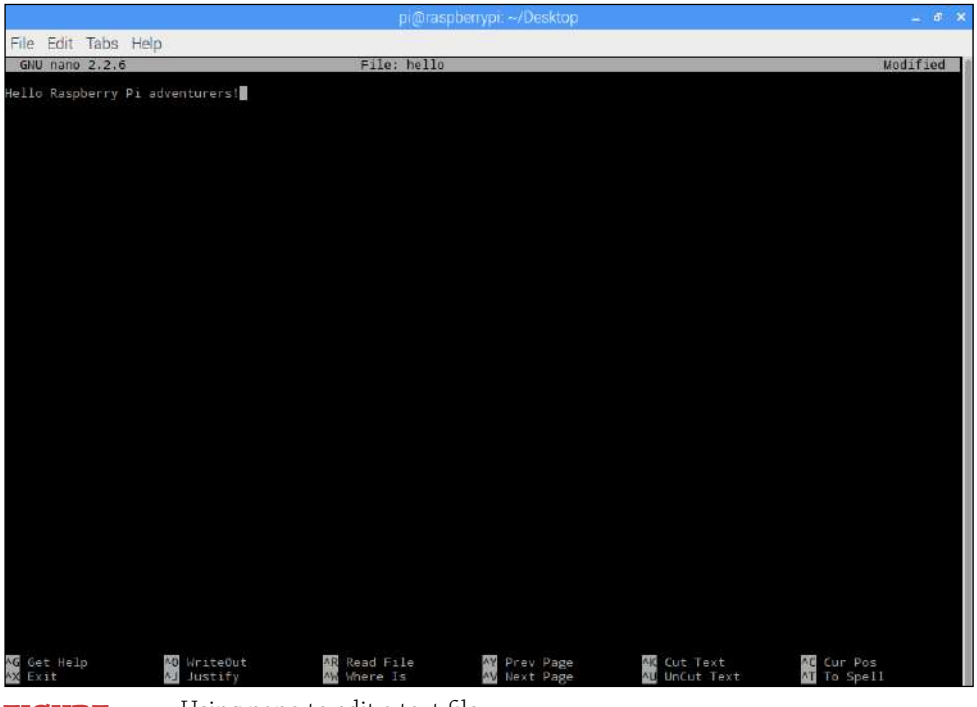


FIGURE 2-10 Using nano to edit a text file

To learn more about using nano to edit files, run the `man nano` command and read the manual.

Using Shutdown and Restart Commands

When you start a Raspberry Pi, the device follows a set of instructions to load the OS. Similarly, when you shut it down, it should follow a set of instructions to shut down the OS in such a way that the file system stored on the SD card stays complete and uncorrupted. It is therefore very important that you don't simply remove the power supply but make sure you always instruct the OS to shut down safely. From the GUI, you can simply use the Shutdown icon from the main menu to shut down the

Raspberry Pi, as described in Adventure 1. Alternatively, from the Terminal, you do this by using the `shutdown` command.

First, make sure that you close any open applications. After all applications are closed, type the following command to start the shutdown process from the command line:

```
sudo halt
```

The `-h` option in this command stands for halt. When the system is "halted", it is safe to remove the power.

Sometimes, you might simply want to restart the OS. The following command, with the `-r` (restart) option, shuts down and then restarts the Raspberry Pi:

```
sudo reboot
```

Continuing Your Text Adventure

To learn more about commands and using terminals in Linux, click the Debian Reference shortcut on the desktop of the Raspberry Pi and explore the information in that program. To recall or review the commands in this chapter, refer to the following Quick Reference Table.

Command Line Quick Reference Table

Command	Description
`cat`	Displays the contents (catalog) of the text file.
`cd`	Changes directory. For example, the command cd Desktop moves you into the Desktop directory.
`cd ..`	Moves you up the directory tree to the parent directory.
`cp`	Makes another copy of a file.
`clear`	Allows you to clear the terminal.
`date`	Displays the time and date.
`ls`	Displays a list of files and folders in the current directory.
`ls -l`	Provides a list that includes more detail about the files. The `-l` parameter is a lowercase L (for *long*), not the numeral 1.
`man`	Displays the manual or description file for the command.
`mv`	Moves a file to a new location.
`mkdir`	Makes a directory.

continued

Command Line Quick Reference Table continued	
Command	**Description**
`nano`	Opens the nano text editor. To open a specific text file, add the filename; for example, nano hello opens the hello text file.
`pwd`	Prints the working directory (shows which directory you are currently working in).
`rm xxx`	Deletes (removes) the file named xxx.
`rmdir`	Deletes (removes) a directory.
`sudo`	Gives the user root or super user permissions.
`sudo apt-get install xxx`	Tells the Raspberry Pi to use the Internet to find, download and install the xxx application.
`sudo apt-get update`	Downloads information about any new versions available for applications on your Raspberry Pi.
`sudo apt-get upgrade`	Installs available upgrades for all applications on your Raspberry Pi.
`sudo halt`	Shuts down (halts) the power to the Raspberry Pi.
`sudo reboot`	Shuts down the power to the Raspberry Pi and then restarts it.

Achievement Unlocked: Your Raspberry Pi responds to your commands!

In the Next Adventure

In Adventure 3, you learn some basic programming skills. You create a crazy monkey animation and a role-playing adventure game using the graphical programming language and environment known as Scratch.

Adventure 3
Creating Stories and Games with Scratch

IF YOU CAN put together a jigsaw puzzle, you can create a computer program using Scratch! With just a few clicks, you can have a bat fly around a castle, make a ninja sneak past a guard or conjure up a flock of butterflies floating through a garden.

Scratch was developed by the Massachusetts Institute of Technology (MIT) Media Lab to help young students learn basic control and programming concepts (`https://scratch.mit.edu`). It is free to use and has become very popular throughout the world. There is even an International Scratch Day, which is held every year to celebrate and share the things people have created using the language.

Scratch is a graphical programming language. This means that, instead of writing text commands, you connect blocks of code together to make a script that makes something happen. You can use Scratch to create interactive stories and computer games in which you draw the scenery (called the **stage**) and the characters (called **sprites**). You can also create music and art with Scratch.

In this adventure, you begin by creating the Scratch equivalent of a typical Hello World! computer program and making Scratch Cat say "Hello". After that, you create a program with a monkey who moves around the screen and changes his facial expression. Finally, you create an entire role-playing game incorporating a variety of backgrounds and different ways to win points.

Sprites are the characters that can be programmed to do something in Scratch. The sprites wear costumes that can be customised.

The **stage** refers to the background for the sprites. You can add scripts to the stage to allow the sprites to interact with it—for example, you might draw a wall that stops your sprite from moving beyond a certain point.

Getting Started with Scratch

If you are using the latest version of the Raspberry Pi operating system Raspbian, Scratch is already installed and you see the Scratch cat icon (see Figure 3-1) in the Programming Menu. To open Scratch, open the main menu at the top of the screen, navigate to Programming and click Scratch (see Figure 3-2).

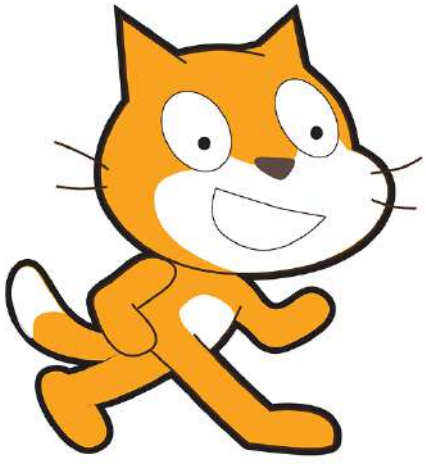


FIGURE 3-1 The Scratch cat

You may have used Scratch in school but it might look a little different on your Raspberry Pi. This is because you are using Scratch version 1.4 on the Raspberry Pi, and there are other versions, including Scratch 2.0, which you may use in school or code clubs through a web browser. If you are using a Raspberry Pi 2 or higher, then you can use Scratch 2.0 in the web browser Chromium, by visiting scratch.mit.edu.

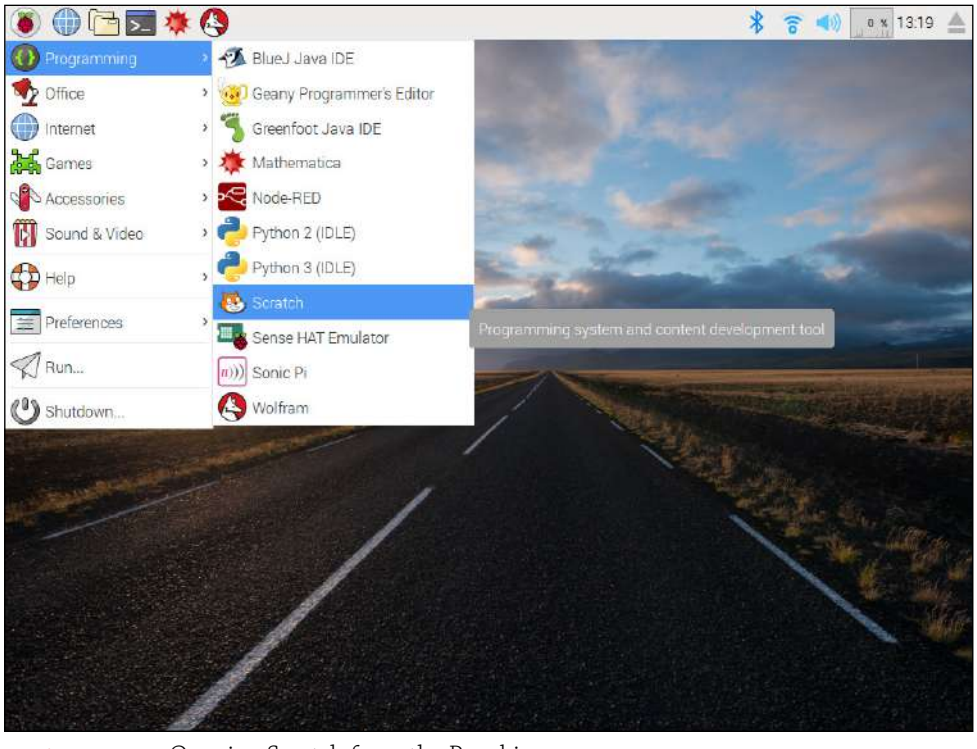


FIGURE 3-2 Opening Scratch from the Raspbian menu

The Scratch Interface

The interface for Scratch includes four main panes, identified in Figure 3-3:

- **Stage**—Your animations, stories and games are displayed in this pane so that you can see what happens as you add backgrounds, characters and scripts to your creations. The stage is set on a grid with an x axis and a y axis so that you can program events or actions to take place at a specific location on the stage; for example, you can have a star appear in the top right corner of the stage by giving the appropriate x and y coordinates.
- **Sprites palette**—This pane displays the sprites or characters that you create for your project. To see or edit scripts or costumes for a sprite, click the sprite in the palette.
- **Blocks palette**—The Blocks palette is divided into two portions. The top portion has eight labels—Motion, Looks, Sound, Pen, Control, Sensing, Operators and Variables—each of which corresponds to a group of blocks that you can use to program your projects. Click a label, and the blocks available for you to use appear in the lower portion of the pane. To form scripts, you select the blocks you want to use, drag them onto the Scripts tab (see the next bullet) and fit them together.

- **Scripts tab**—The centre pane of the interface has three tabs along the top: Scripts, Costumes or Backgrounds, and Sounds. When the Scripts tab is selected, you can drag the programming blocks into this pane and fit them together to build your scripts.

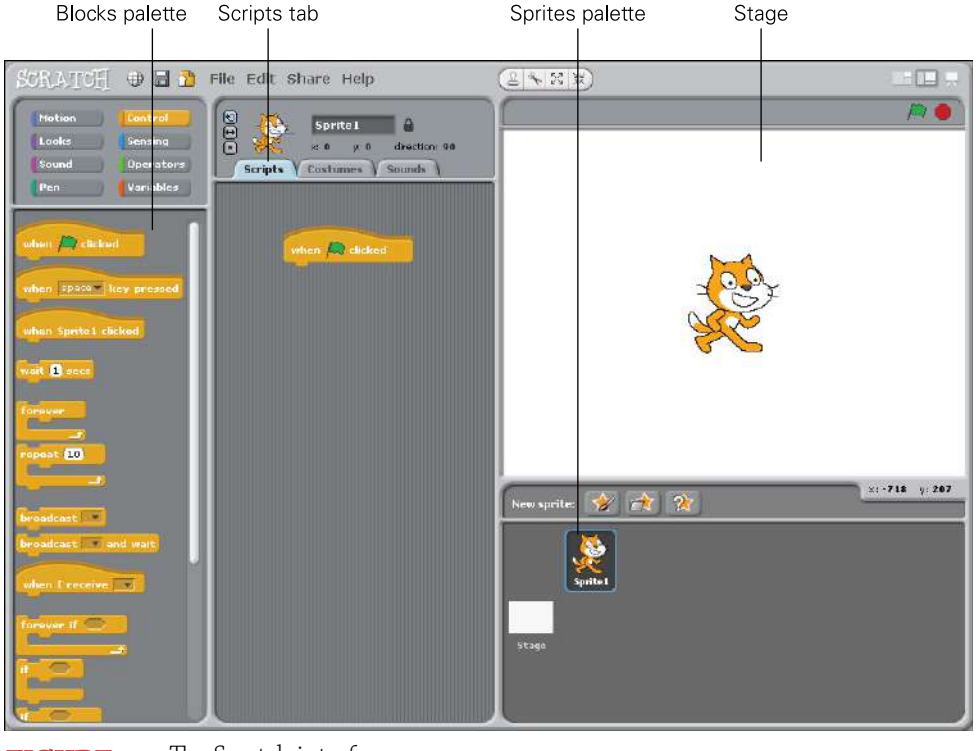


FIGURE 3-3 The Scratch interface

A Quick Hello from Scratch Cat

The best way to learn how to use Scratch is simply to use it! In this project, you learn the basics of using Scratch by following some simple instructions.

1. To begin, make sure that the cat sprite with the label Sprite1 is selected in the Sprites palette and the label reads Sprite1.
2. In the centre pane, click the Scripts tab. You will drag blocks onto this tab to create a "script" that tells your program what actions to perform.

Notice that the cat and the label Sprite1 appear at the top of the tab to indicate that the current script will be applied to that sprite. Always check that you have selected the correct item before working in the Scripts tab.

3. Click the Control label at the top of the Blocks palette to see all the control blocks available.
4. Drag the control block labelled `when`  `clicked` onto the Scripts tab, as shown in Figure 3-4. This control block is the "start button" of your program. It means that when the green flag above the stage is clicked, the script you have created runs.
5. Click Motion at the top of the Blocks palette to see the available motion blocks. From the list of choices, drag the `move 10 steps` block to the Scripts tab and connect it with the control block that you placed in Step 4, as shown in Figure 3-4.

Some of the blocks have sections in the code that you can customise. For example, in the motion block `move 10 steps`, you can change the 10 to any number you like.

6. Click Looks in the Blocks palette, drag `say Hello!` to the Scripts tab and connect it to the motion block you placed in Step 5.
7. Click Sound in the Blocks palette, drag `play sound meow` to the Scripts tab and connect it to the looks block you placed in Step 6.
8. Finally, save your file, and then press the green flag in the top-right corner above the stage to see your script work.

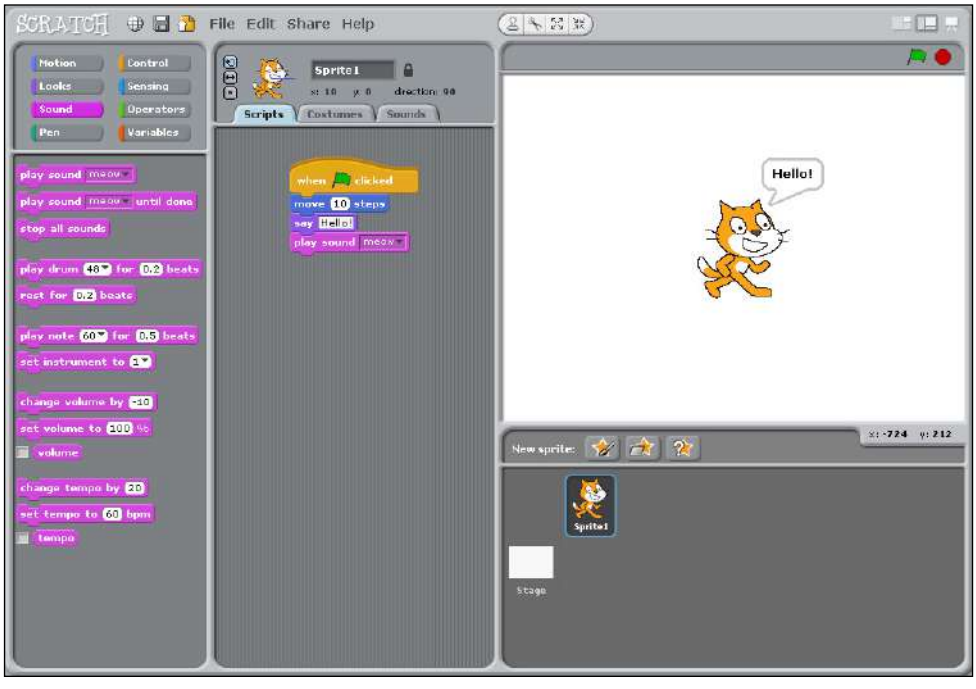


FIGURE 3-4 The blocks for a quick Hello from Scratch cat

When completing a Scratch program containing scripts, it is good practice to save your work at regular points and test that it works. To do this, click File➪Save As, name your file and click Ok. Then press the appropriate key to start your script. Scratch files can be saved into the Scratch Projects folder on your Raspberry Pi. As you move through the instructions in this chapter, I remind you to save your work at the end of each section.

Congratulations! You have written your first program using Scratch. Of course, you can do much more with Scratch than just move a cat around the screen. Next, you take a look at the parts of Scratch that you can design yourself—the stage and costumes.

Setting the Stage

If you are creating an animated story or computer game using Scratch, you want to set the scene by changing the background from its plain white default. You can do this in two ways: by designing and drawing your own background or by selecting an image from the Scratch library.

To prepare to change the background image, click the Stage icon, which is situated next to the Sprites palette. Select the Backgrounds tab. You have a choice between editing the current background or adding a new one:

- To edit the current background, click the Edit button in the Backgrounds tab in the centre of the screen and the Paint Editor window appears, as shown in Figure 3-5. Use the drawing tools to draw a setting for your animation or game. For example, you might want to draw a room or a maze.
- To add a new background, either click the Paint button to open the Paint Editor window and create a scene, or click the Import button to use a scene from the image library (see Figure 3-6).

  Scratch also has an option for you to use webcam images as backgrounds for your Scratch projects. Click the Camera button to access this option. Before you can use this function you need to have either a webcam plugged into a USB port or the Raspberry Pi Cam plugged into the camera slot on the Pi board.

To see a video that walks you through the Scratch interface and shows you how to create backgrounds and characters, visit the companion website at www.wiley.com/go/adventuresinrp3E. Click the Videos tab and select the QuickHelloFromScratch file.

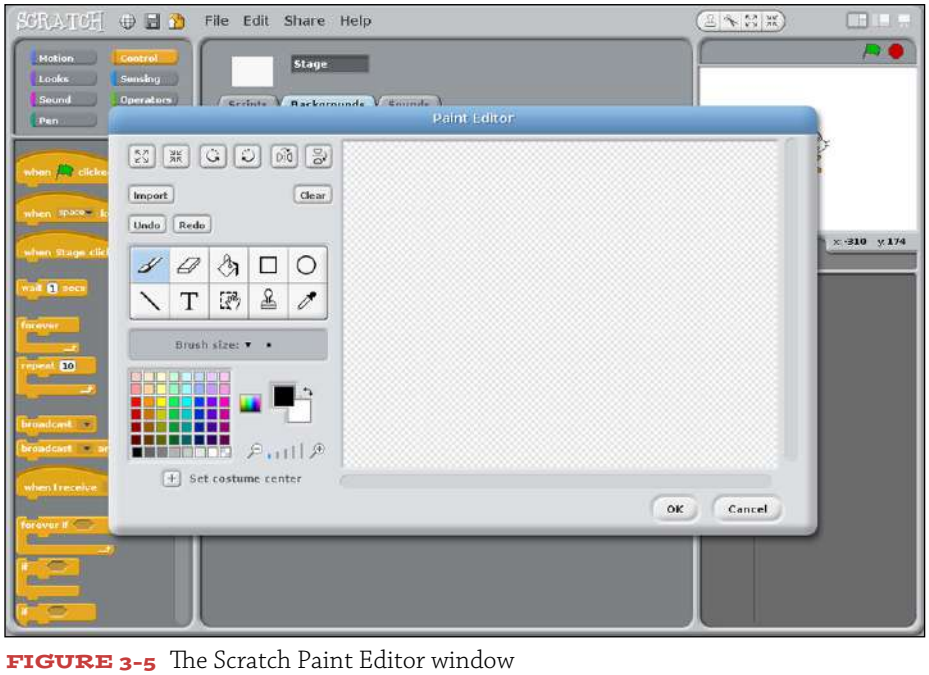


FIGURE 3-5 The Scratch Paint Editor window

FIGURE 3-6 The Import Background window

Creating Costumes and Original Sprites

Of course, you won't always want to use Scratch Cat as your sprite—you may want other animals, people, astronauts, flowers or even a basketball! Scratch has its own sprite library, much like the background image library, which you can use to get more sprites for your project. You can also edit these sprites, or simply draw original sprites of your own.

Using the Scratch Sprite Image Library

To use a sprite design from the Scratch Sprite Library click the Choose New Sprite from File icon at the top of the Sprites palette (the icon with the folder and star), as shown in Figure 3-7. Browse through the Things, People and Animals folders until you find a sprite you want to use. Select it and click OK to add it to your Sprites palette.

Editing an Existing Sprite

Select a sprite from the Sprites palette. In the centre pane, click the Costumes tab to switch from Scripts to Costumes. To edit the sprite, click the Edit button next to the picture of your sprite under costume1 to open the Paint Editor window (see Figure 3-8). You can then use the drawing tools to add your own enhancements to the Scratch cat sprite—why not give it a cape or a moustache? You'll learn more about how to change costumes to create variations of sprites in the next section.

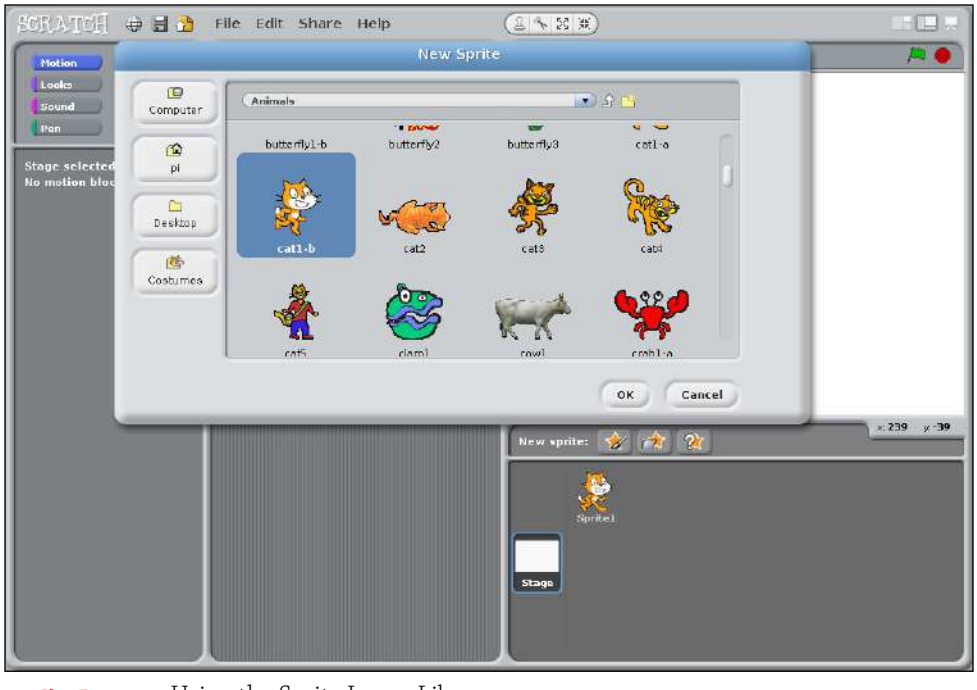


FIGURE 3-7 Using the Sprite Image Library

FIGURE 3-8 Editing an existing sprite using the Paint Editor—love the moustache!

Creating Your Own Original Sprites

To create original sprites of your own, click the Paint New Sprite icon above the Sprites palette (the icon with the paintbrush and star). The Paint Editor window is displayed, and you can use a freehand paintbrush or shapes to create your own characters.

Play around with Scratch a bit to get comfortable with the different aspects of the application. When you have a good feel for how it works, move on to the next section to create an animated monkey!

For a video that walks you through the Crazy Monkey Animation project, visit the companion website at www.wiley.com/go/adventuresinrp3E. Click the Videos tab and select the CrazyMonkey file.

Animating a Crazy Monkey

It's only natural for adventurers to come across challenges, especially on an expedition through a wild jungle. A crazy monkey jumping all over the screen with a variety of facial expressions is definitely going to be a fun challenge!

1. Open a new file by selecting File➪New. Delete the Scratch Cat sprite by right-clicking it and selecting Delete from the menu that is displayed.

   For this project, you need a jungle style background and a monkey sprite. You can draw your own using the Paint Editor in Scratch, or you can use the forest background and the monkey sprite from the image libraries as described in the previous section. Alternatively, if your Raspberry Pi is connected to the Internet, you can download the jungle background and monkey sprite used in this project from the Adventures in Raspberry Pi website at www.wiley.com/go/adventuresinrp3E.

2. With the monkey sprite selected, click the Costumes tab. Rename the costume to Monkey1 by clicking the sprite name above the Edit button and typing the new name. Click the Copy button to make a duplicate of the monkey. You should now see two monkeys on the Costumes tab: Monkey1 and Monkey2.

3. The next step is to change Monkey2's expression. Click the Edit button for Monkey2 to open the Paint Editor. Use the paintbrush tool to erase the mouth on the monkey and replace it with a different one (see Figure 3-9). You can continue to duplicate the monkey as many times as you like, changing the expression on each one.

FIGURE 3-9 Duplicating a sprite and changing the costume. Notice the different expression on each monkey's face.

4. Continue to make copies of the costume and edit each new copy with a different face. You can even change the eyes or move the tail!

5. Now click the Scripts tab. You are going to create a set of blocks to switch among the costumes that you have created. In the Blocks palette, click Control and drag the `when`  `clicked` block onto the Scripts tab.

6. Click Looks in the Blocks palette and add `switch to costume Monkey1` (see Figure 3-10). You can use the drop-down arrow in this block to select the costume (expression) you want to start with.

FIGURE 3-10 First steps in the Crazy Monkey script

7. In the Blocks palette, click Control to switch to the control blocks. Under the first block in the Scripts tab, add the control block `forever`. Inside the `forever` block add the control block `wait 1 secs` and the looks block `next costume`. Figure 3-11 shows the script at this point.

   The `forever` block is a **loop**. It runs the same sequence of blocks over and over again until you stop the program. In this case, you are making the monkey change his facial expressions, over and over again. In computing, this kind of repetition is called **iteration**.

FIGURE 3-11 The Crazy Monkey script with a `forever` block

What do you think will happen when you click the green flag? What do you think will happen if you press the red octagon? Try it and find out if you are correct.

Next, you add some more animation to your monkey by having him move as well as change facial expressions.

8. With the Scripts tab still selected, underneath your first block add another `when`  `clicked` control block.
9. Add the motion block `go to x:0 y:0`.

The stage area in Scratch uses the coordinates x and y to refer to where your sprite will appear on the stage. If you want your sprite to begin in the middle of the stage, use `x:0` and `y:0`. If you want your sprite to appear in the top left portion of the stage, use `x:-163` and `y:11`. Notice that when you select a sprite with your mouse you can see its coordinates at the top of the Script tab (see Figure 3-12). Move the sprite with your mouse and watch as the coordinates change to reflect the new position. The library also includes a handy graph background that shows the x and y axes; you can import that graph from the library to help you.

10. Again, you need to use a `forever` control block to repeat some instructions. Inside the `forever` block, add three blocks from the Motion Blocks palette: `move 10 steps`; `turn 15 degrees right`; and `if on edge, bounce`.
11. Note the three small buttons next to the monkey sprite at the top of the Scripts tab (see Figure 3-12 for reference). These control a sprite's rotation. Click the middle button, which directs the sprite to `only face left and right`. This adds more animation to the monkey sprite and allows him to move more than just his mouth.

Save your animation by clicking File➪Save As and naming it `jungle animation` inside the Scratch Projects folder.

Figure 3-12 shows the completed script. Notice that I have added a third block of commands to include a sound effect for the animation.

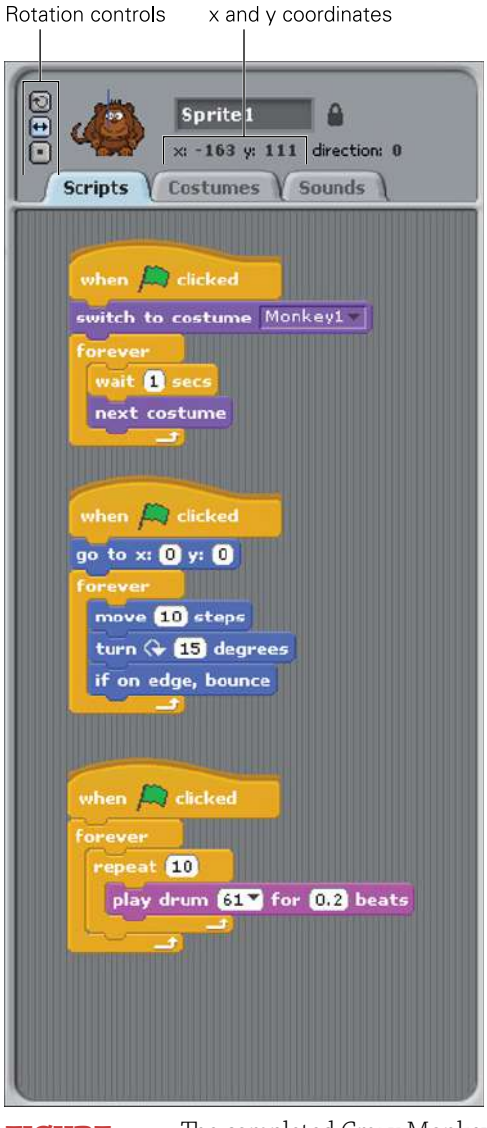
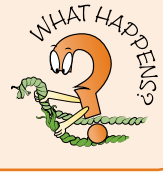


FIGURE 3-12 The completed Crazy Monkey animation script in Scratch

What do you think the monkey will do now? Will he still change costumes? Click the green flag to find out!



CHALLENGE

If a moving monkey with different faces is not exciting enough, you can add some sound effects. Why not have a go yourself? What block do you think you should begin with? See the lowest block of commands in Figure 3-12 if you need help.

Creating an Adventure Role-Playing Game

Now that you have conquered the crazy monkey in the jungle, it's time to move on to the next level by creating a first-person adventure role-playing game in Scratch.

In this project, you learn some programming concepts that are common no matter what programming language you use (such as loops, if statements and variables) by creating a game in which a single player can move through locations or rooms to reach a magical key, trying to avoid deadly flowers along the way.

For a video of the Adventure Game project, visit the companion website at `www.wiley.com/go/adventuresinrp3E`. From the Videos tab, select the ScratchAdventureGame file.

Creating Your Sprite and Stage

To begin, you need to draw an Adventurer sprite from a top-down or bird's eye view; see the sprite in Figure 3-13 for an example. (Don't forget to delete the Scratch Cat sprite first by right-clicking it with your mouse and selecting Delete from the menu that appears.)

Click the paintbrush icon next to New Sprite to open the Paint Editor window, and use the tools to create your sprite. Make sure that you draw your Adventurer sprite facing towards the right, as this will become important later in this project. You also need to create an outside cave and inside cave stage background, labelling the locations `Outside` and `Inside`, respectively. So that you do not lose any of your work in this project, you should save your work as you go along. Refer to the "Setting the Stage" section earlier in this adventure for a reminder on how to create a background. Figure 3-13 shows my version of the completed game in progress.

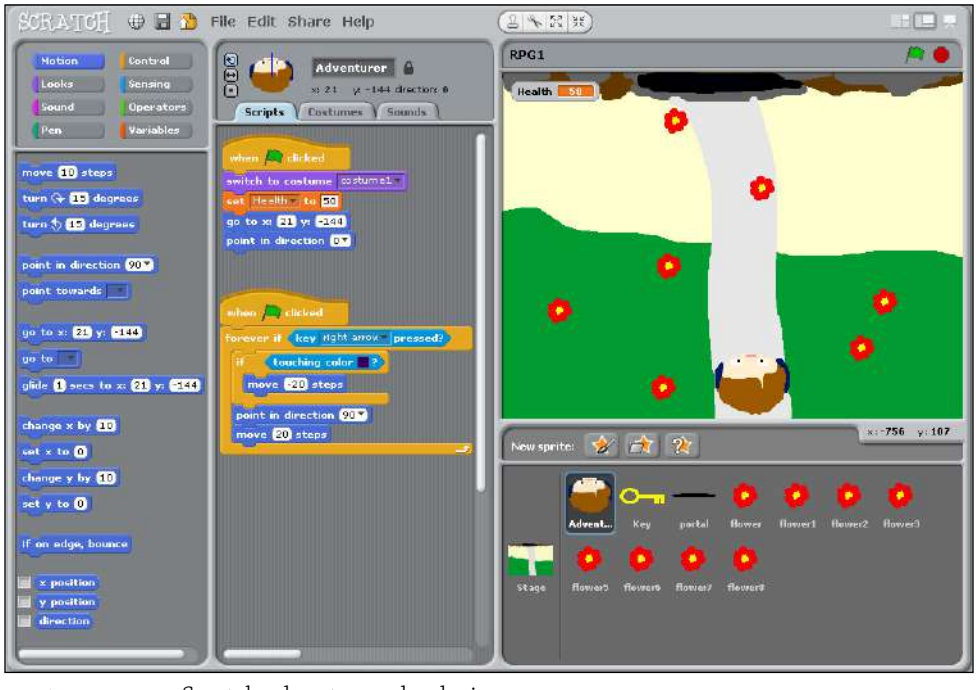


FIGURE 3-13 Scratch adventure role-playing game

Setting the Start Position of the Adventurer Sprite

When the game begins, the Adventurer character always starts at a certain point on the stage; after all, an adventure needs to start somewhere.

1. To set the start point, click the Adventurer Sprite and position it on the stage where you want to start the game.
2. On the Scripts tab, drag the control block `when`  `clicked` and the motion block `go to x: y:` (see Figure 3-14).
3. Set the x and y coordinates to where your Adventurer sprite is located on the stage. You can find its exact coordinates by looking near the top of the Scripts tab, between the sprite's name and the tab labels.
4. You can also add a Motion block to `point in direction _` if you want the Adventurer sprite to be looking up, right, left or down at the start of the game. The value you type into the block determines the direction the sprite faces: Use the value 0 to make the sprite look up, –90 for the left, 90 for the right or 180 for down. Try changing the value to see how your sprite responds (see Figure 3-14).

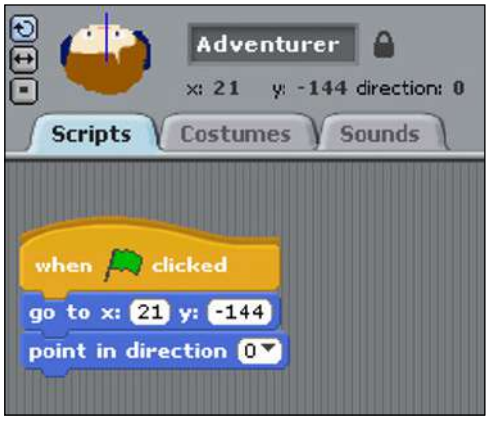


FIGURE 3-14 Setting the start position for your sprite

Creating Variables: Including Health Points for the Adventurer Sprite

In role-playing games, players (or rather, their characters) typically start with a certain number of 'health points' or 'lives'. As you play the game, these points may decrease as you encounter foes or increase when you collect certain objects or overcome obstacles. You can create **variables** in Scratch to allow for values that change, and use these variables inside different scripts. You can set the health points to a certain value—for example, 50—at the start of the game and create scripts that will add or take away points when triggered—for instance, finding a useful tool might add 10 health points. This feature makes the game more interesting.

A **variable** holds a value that can be changed. The health variable in your adventure role-playing game is an example of a value that can be changed and used inside different scripts.

1. To create a variable, click Variables in the Blocks palette and then click Make a Variable. The New Variable window opens and asks you to type a name for your variable.
2. Name your variable `Health` and ensure that For *all sprites* is checked before clicking OK. Figure 3-15 shows the window with the correct settings.
3. You'll see some orange blocks added to your Variables palette, and a small counter box appears on the stage (see Figure 3-18).

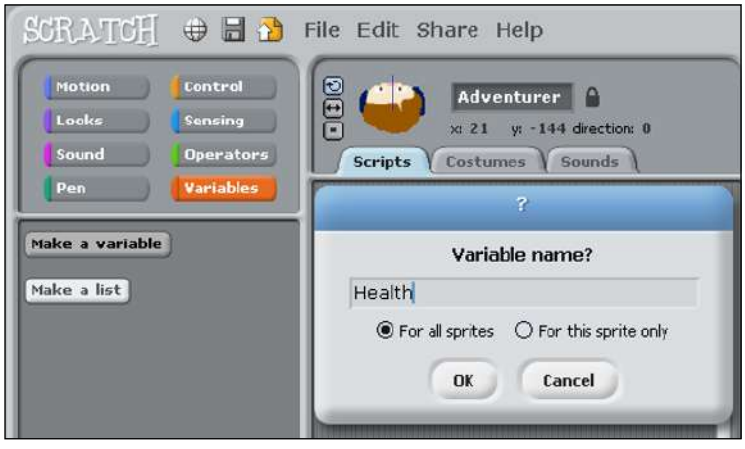


FIGURE 3-15 Creating a variable

4. Add the block `Set Health to 0` to the starting point script you have already created. You can then change the value of `Health` from 0 to 50. This means at the start of the game when the green flag is clicked, the Adventurer sprite starts with 50 health points. Figure 3-16 shows the script up to this point.

5. Remember to save the work you have done so far by clicking File➪Save. Then test that your scripts work by clicking the green flag icon.

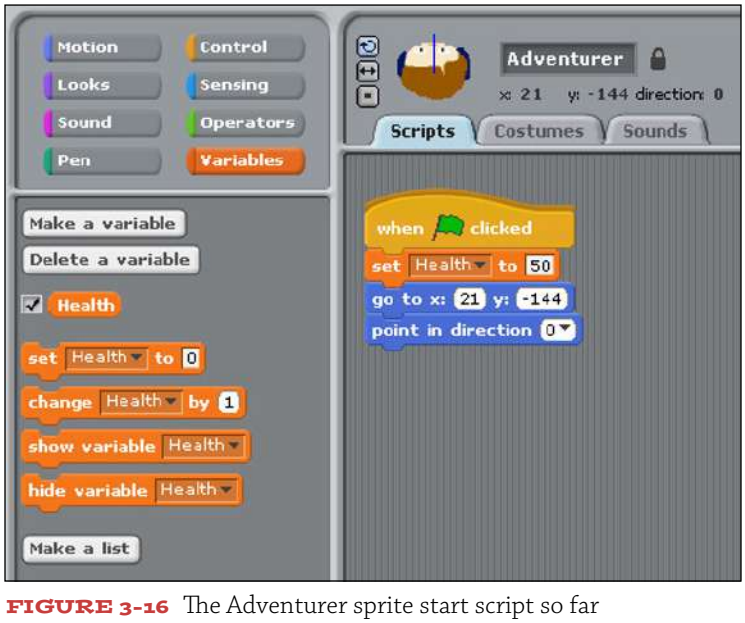


FIGURE 3-16 The Adventurer sprite start script so far

Controlling the Direction and Movement of the Adventurer Sprite

An important part of any computer game is being able to control the movement of a sprite using keys or buttons. In Scratch you are able to create more than one script for a sprite or for a stage, which can be run in parallel with each other. These multiple scripts running at the same time are called **threads** in computing. You need to create a number of scripts for the Adventurer sprite to be able to control the sprite's movement.

1. Underneath the first script you created to set the start position and health variable, add a new `when`  `clicked` control block.
2. Add a `forever if` block. Notice that this block has a hexagon shape in it. This is designed so that you can add extra blocks, such as operators or sensing blocks, to insert a condition. This means that the blocks contained within the `forever if` loop run only if the condition is met. For this project, click Sensing in the Blocks palette and add a `key right arrow pressed` block by dragging it with your mouse and placing it inside the hexagon shape. This creates a **conditional** statement. Notice that you can change which key is referenced by using the drop-down menu on the block. This allows you to set different keys for different directions of movement, as you do later in this project.

In computing, a **conditional** statement is one in which an action will be taken only if a certain condition is true. For example, in the block you create in Step 2, the `forever` part of the `forever if` block loops the sequence of instructions contained within its structure only if the right arrow key on the keyboard is pressed.

3. Add the motion block `point in direction` inside the `forever if` loop and set it to 90, which points the sprite to the right.
4. Underneath this block and still inside the `forever if` loop, add the motion block `move 10 steps` and change the value of steps to 20. See Figure 3-17 to see how the script should look at this point.
5. Test that this script works by clicking the green flag to start all the scripts, and then pressing the right arrow on the keyboard.
6. Create three more scripts like this one for the left, up and down keys, changing the values to move the Adventurer sprite in the correct direction for each arrow key. Save and test your file.

FIGURE 3-17 Controlling the direction and movement of the Adventurer Sprite

Entering a Cave and Switching Backgrounds

The action in a role-playing game typically involves the characters moving among different scenes. In this game, the Adventurer sprite begins outside a cave and has to cross the stage to get to the cave entrance before going inside. At the start of this project you were asked to create two backgrounds for the stage: one outside the cave that you have been using so far, and one for the inside of the cave. But how do you get the Adventurer sprite to switch between these backgrounds? You need a script for each of the backgrounds, another script for the Adventurer sprite and another for the stage.

Adding a Script to Make the Adventurer Sprite Move Between Backgrounds

Scratch programs are built with a collection of small scripts. Sometimes you need a way to communicate among these scripts when certain things happen in your adventure game—for example, when you move the Adventurer sprite from one location or background to another. This next script uses a new sprite to act as a trigger, so that when the Adventurer sprite touches it, a message is **broadcast** to all the sprites and the stage at the same time, to alert them to the change.

1. First, you need to create a new sprite to act as the cave entrance. Simply click the Paint New Sprite button (the paintbrush icon) on the Sprites palette and use the circle tool to draw an ellipse that matches the entrance of your cave (see Figure 3-18). This ellipse is your new sprite.
2. Name the sprite portal and position it at your cave entrance on the opposite side of the stage from where your Adventurer sprite starts. This new portal sprite acts as the trigger to move between the two backgrounds.
3. Select the Adventurer sprite from the sprite palette and add another new script to act as a thread on the Adventurer sprite Scripts tab. Add a `when`  `clicked` control block onto the Scripts tab of the Adventurer sprite, and add the wait `until_` block to it. This is another type of conditional block, but unlike with the

forever loop, the sequence of instructions happens only once when the condition is true. In this case the condition is true when the Adventurer sprite 'touches' the portal sprite.

4. Add the sensing block touching_ into the hexagon space on the wait until_ control block, and use the drop-down menu to select the portal sprite, so the block now reads wait until touching portal.

5. Add the control block broadcast_ and select New from the drop-down menu. Name the new broadcast message Level.

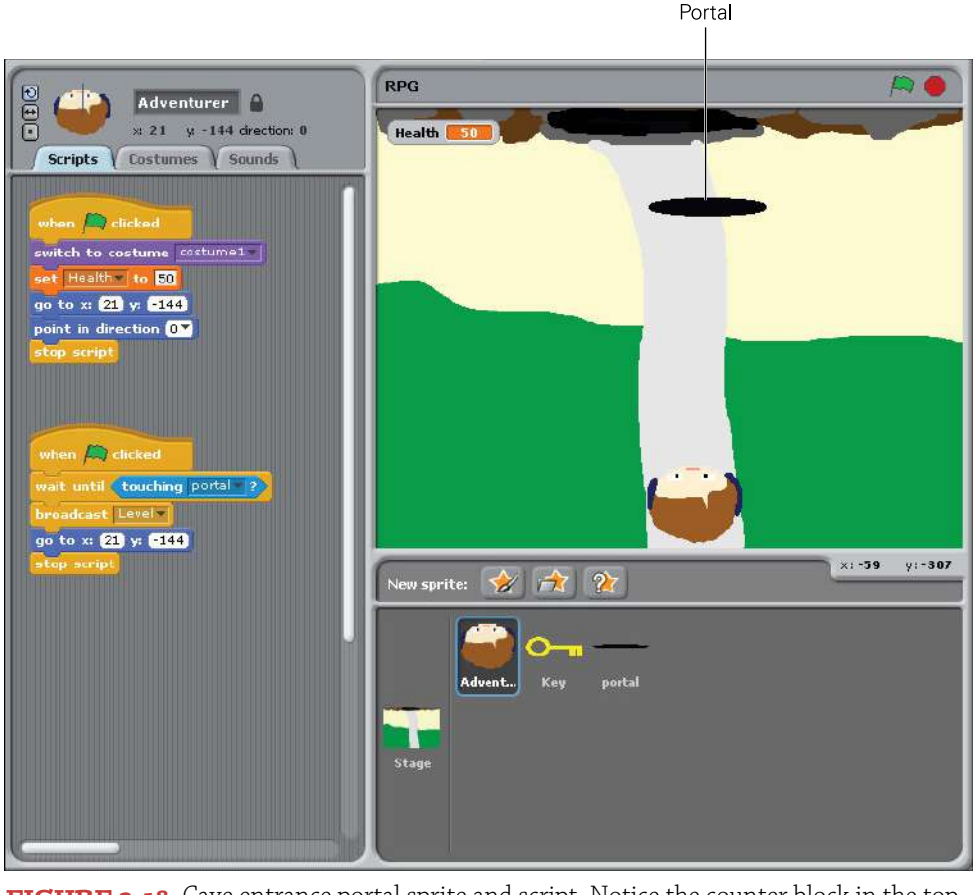


FIGURE 3-18 Cave entrance portal sprite and script. Notice the counter block in the top left corner of the stage.

A **broadcast** message is used to coordinate the actions of different sprites and the stage. It keeps all the scripts running for each sprite and keeps the stage synchronized. In this case, when the Adventurer sprite touches the portal, a message called Level is broadcast. This message triggers a script for the stage to switch to the `Inside` cave background that you created.

Adding a Script to Switch the Stage

So far, you have created a portal sprite that broadcasts a message when touched by the Adventurer sprite, but that does not solve the problem of switching the backgrounds of the stage. You need to add a script to the stage so that it responds to the message that was broadcast when the portal sprite was touched.

1. Click the Stage icon in the Sprites palette. Add the control block `when`  `clicked`, followed by a `switch to background_` looks block, and select `Outside` from the drop-down menu. Make sure that you have labelled your stage backgrounds as `Inside` and `Outside` to correspond with your location designs.
2. Add another control block, but this time use `when I receive` and select the broadcast message **Level** from the drop-down menu. Add the `switch to background_` looks block and choose Inside from the drop-down menu. Figure 3-19 shows this script. Save your work so far and test to see if it works.

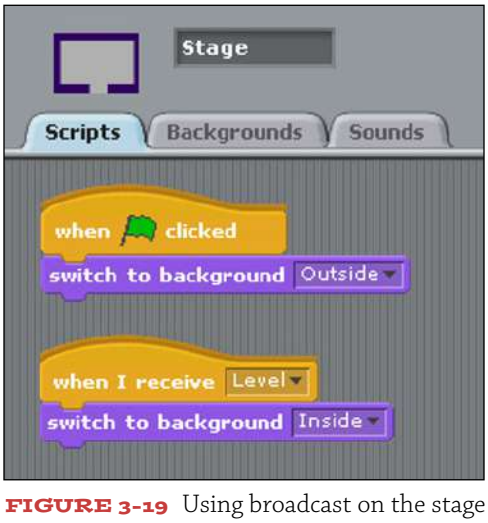


FIGURE 3-19 Using broadcast on the stage to switch between backgrounds

3. Whenever you are creating more complicated scripts for multiple sprites and on the stage, it is a good idea to test that they work. Click the green flag and use the arrow keys on your keyboard to navigate the Adventurer sprite to the portal and see what happens.

4. You may find that your Adventurer sprite is not positioned at the entrance of your `Inside` cave background. Simply drag the Adventurer sprite to where you would like it to start from on this stage background, make a note of the x and y coordinates for this position and, on the script you have been making to move the sprite between backgrounds (refer to Figure 3-20), add the motion block `go to x:0 y:0`. Replace the values with the new coordinates.

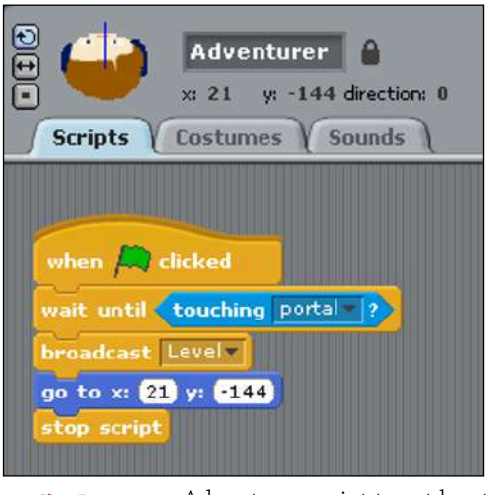


FIGURE 3-20 Adventurer script to set location on new background

Creating an Enchanted Key to Exit the Cave and Giving Extra Health Points

Rather than making another portal sprite to move to another level or background, why not introduce a new sprite that behaves like a magic object?

1. Using the Sprites palette, click the Choose New Sprite From File icon and select Key1 from the Things folder. Rename the sprite by clicking in the name box above the centre tabs and typing `Key`.

2. Just like the script that you created to move the Adventurer sprite to the inside of the cave, the script required for this sprite uses the `wait until touching _` conditional and a new broadcast message called `new_level`, only this time the Key sprite waits until it is touched by the Adventurer sprite to trigger the action, as shown in Figure 3-21.

3. The script works only if you add another script to the stage so that when it receives the new broadcast message for the `new_level`, it switches to the `Outside` background. Click the Stage icon in the Sprites palette and follow the steps in the preceding section ("Adding a Script to Switch the Stage") to add a new script that says: "When I receive new_level, switch to background Outside".

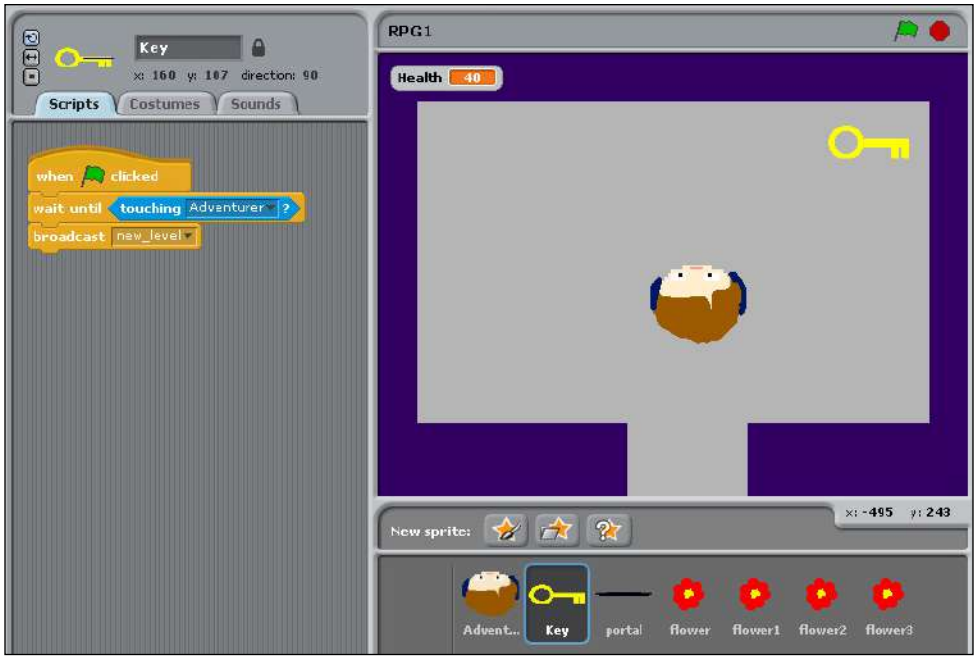


FIGURE 3-21 The enchanted key script

CHALLENGE

You can have a little fun by adding looks blocks to make the key say something when it's touched, and a variable block to increase the Adventurer sprite's health points. Try adding blocks to the script for those actions. Refer to Figure 3-22 if you need help.

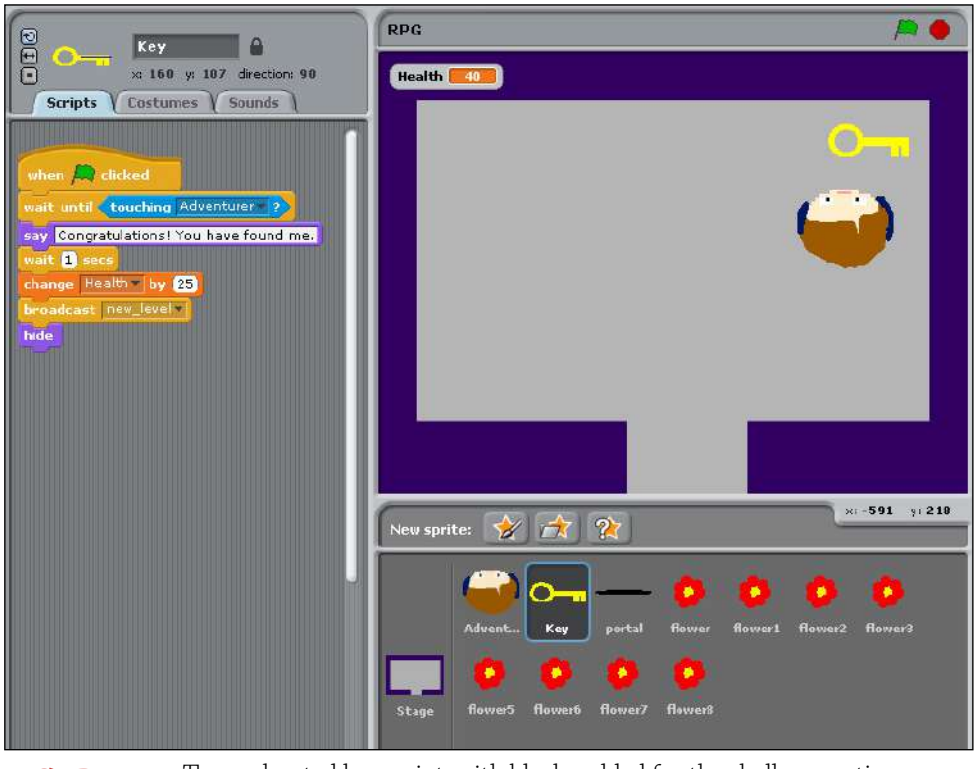


FIGURE 3-22 The enchanted key script, with blocks added for the challenge actions

Using if Statements to Show and Hide Sprites

As it stands now, when you play this game the new sprites that you have created so far, such as the portal sprite, remain visible even when you change backgrounds. This is a little confusing. You really want the portal to show only on the first background, `Outside`, and the key to show only on the second background, `Inside`. In this section, you learn how to add an **if. . .else** block to address this problem.

If and **if. . .else** statements are common constructs in computer programming. When you use an `if` statement, you are asking for a condition to be met, and then making something happen if the condition set is true. For example: If it is raining, then put up an umbrella. You can add another action for when the condition is false using the **else** command. For example: If it is raining, then put up an umbrella; else, wear sunglasses.

The script you create in the following steps tells the portal sprite to appear only when background 1—the outside of the cave—is shown, and to remain hidden the rest of the time.

1. In the portal sprite's Scripts tab, add the control block `when`  `clicked` and attach a `forever` looping control block to it.
2. Add the control block `if else` inside the `forever` loop and drag the operator block `0 = 0` into the `if` hexagon condition (refer to Figure 3-23).
3. Place the sensing block `of` inside the first 0 of the operator block, and add the value 1 to the second part of the same block.
4. Using the drop-down menus on the sensing block, and starting with the second value after `of`, change it to `stage` and then change the first value to `background #`.
5. Finally, add the looks block `show` under the `if` condition and the looks block `hide` under `else`. Save and test your script.

CHALLENGE

What amendments to this script would you need to create to hide the key on the `Outside` background and show the key on the `Inside` background? Try making these changes to your script.

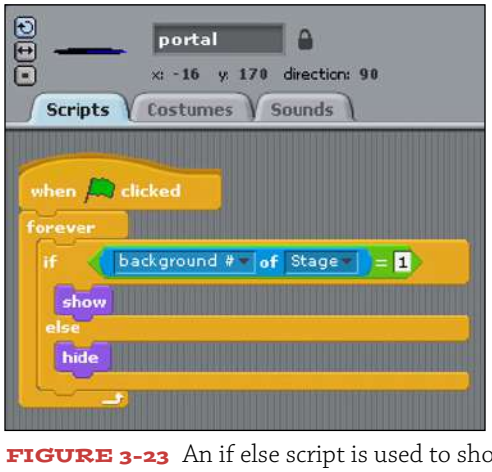


FIGURE 3-23 An if else script is used to show and hide sprites in Scratch.

Creating Health-Point-Stealing Sprites

The Adventurer sprite can now move around the stage and move between stage backgrounds using the broadcast message. However, as it stands, this will be a really easy game to play and anyone playing the game may tire of it very quickly! You can make the journey to the cave entrance more difficult for the player by adding obstacles that will steal health points.

1. To add some obstacles to make it more difficult for the Adventurer sprite to get around the stage, click the Paint New Sprite icon to create a new sprite. Use the tools to draw a flower as your new sprite, and name it *flower*.
2. Add a script to the flower sprite by selecting it from the Sprites panel and then dragging the control block `when`  `clicked` onto the Scripts tab.
3. The obstacle needs to be a constant danger for the sprite, so you need a script that is constantly running. To achieve that, add the `forever if` block. This time the condition is *if the Adventurer sprite is touching the flower then remove a certain number of health points*. Add the sensing block `touching _`. Figure 3-24 shows the completed block.
4. Inside the `forever if` block, add the variable block `change health by 1` and change the value from 0 to –10 so that it removes health points from the player.
5. Add the looks block `say ouch! for 2 secs` after the variable block inside the `forever if` loop block.
6. Remember to add the `if else` script to hide the flower sprite after the player enters the cave, and then save and test your script.

FIGURE 3-24 Health-point-stealing flower sprite script

Don't forget to add the script to hide the flower sprite after the Adventurer sprite enters the cave. After you have added this script, you can duplicate the flower to cover the stage with more obstacles, making the game more interesting. To duplicate the flower sprite, right-click it in the Sprites palette and click Duplicate. You can do this as many times as you like to make multiple copies.

Improving the Movement of the Adventurer Sprite Using if Blocks

The player controls the Adventurer sprite by using the keys on the keyboard. This works well when the sprite is on the stage using the `Outside` background, as there are no walls. However, when it enters the cave, the sprite appears to be walking over the walls. You can use another `if` condition to help stop this from happening.

1. Click the Adventurer sprite and locate the four scripts that control movement using the arrow keys on the keyboard (refer to Figure 3-17).
2. On one of the scripts, add a control block `if` inside the `forever if` loop, above the `point in direction 90` block that is already there.
3. Add the sensing block `touching color` into the blank hexagon on the `if` loop you have just added. Select the colour of the walls by clicking the coloured square box; this action turns the mouse cursor icon into a droplet icon, and you can select the walls with your mouse, getting the exact colour match required.
4. Add the motion block `move 10 steps` inside the `if` loop and set the value to –20 steps. Figure 3-25 shows the new script.
5. Add the same extra piece of script to the three remaining control scripts for each key. Don't forget to save the file and test that your script works.

Creating a Game Over Screen

Typically in role-playing games, when a player loses all her health points, the game ends and a Game Over screen is displayed. Follow these steps to create a Game Over screen:

1. First you need to add a new Game Over background to the stage. You can either paint a new one or duplicate one of the existing backgrounds and edit it to display Game Over across it in large letters (see Figure 3-26).
2. Add another script to the Adventurer sprite. Click the sprite in the Sprites palette, and then drag the control block `when`  `clicked` onto the Scripts tab.

FIGURE 3-25 Completed script to control movement of right key

3. Add a **forever** control block underneath, and an `if` control block inside the `forever` block.
4. Drag the operator block `0 < 0` inside the `if` blank hexagon, Add the variable `health` inside the left side of the **<** sign and type the value `0.1` in the right side.
5. Add the control block `broadcast` and create a new broadcast message called `Game Over`.

The code of this script states that if the health of the Adventurer sprite is less than 0.1, the Game Over message is broadcast to all the sprites and the stage. You need to add the following script to the stage to listen for this broadcast message to end the game.

```
When I receive 'Game Over'
Switch to background 'Game Over'
Stop All
```

The Stop All control block stops all the scripts in Scratch from running, ending the game.

Save the file and run the program to ensure it works as expected. If not, check your work and correct it if you need to.

FIGURE 3-26 Creating a Game Over screen in Scratch

Ideas for Improvements to Your Game

Now that you've learned how to use Scratch, you may want to continue to improve your game. Here are some further ideas to keep you going:

- Try setting some random events to happen during the game.
- Add music and sound effects to make it more exciting for the player.
- Create more sprites for the Adventurer sprite to interact with.
- Use a MaKey MaKey invention kit to create a custom game pad to match your game. You can learn more about MaKey MaKey and order a kit at `www.makeymakey.com`.

For the complete guide to Scratch, download the Scratch Reference Guide from `http://download.scratch.mit.edu/ScratchReferenceGuide14.pdf`.

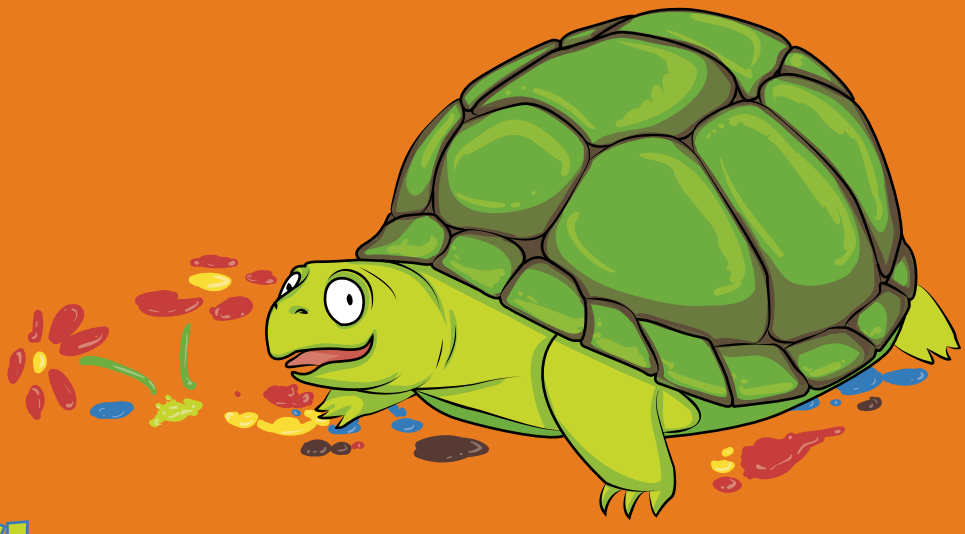
Scratch Command Quick Reference Table	
Command	**Description**
Control Blocks	
`broadcast x`	Sends a message to all the sprites and the stage which can be used to synchronize scripts across multiple sprites and the stage.
`forever`	Repeatedly iterates actions within set.
`forever if`	Checks whether a condition is true, over and over. If the condition is true the program runs the blocks inside.
`if...else`	If the condition is true, the program runs the blocks inside the if section. If not, it runs the blocks inside the else section.
`repeat x`	Sets number of times for action to repeat.
`stop all`	Stops all scripts for all sprites.
`wait x secs`	Sets time before executing next command.
`when`  `clicked`	Begins script when green flag icon is clicked.
`when I receive x`	Begins script when a set broadcast message is heard.
`When x key pressed`	Begins script when designated key is pressed.
Motion Blocks	
`change x by _`	Changes sprite's position on the stage x axis by a specified amount.
`change y by _`	Changes sprite's position on the stage y axis by a specified amount.
`go to x:_ y:_`	Moves sprite to set x and y coordinates on the stage.
`if on edge, bounce`	Turns sprite in the opposite direction if it touches the edge of the stage.
`move x steps`	Moves sprite forward or backwards *x* number of steps.
`point in direction x`	Points sprite in direction *x*.
`point towards x`	Points sprite towards another sprite or a mouse cursor.
`set x to _`	Sets sprite's position on the stage x axis to a designated place.

Command	Description
Control Blocks	
`set y to _`	Sets sprite's position on the stage y axis to a designated place.
`turn (clockwise) x degrees`	Rotates sprite clockwise *x* degrees.
`turn (anti-clockwise) x degrees`	Rotates sprite anti-clockwise *x* degrees.
Looks Blocks	
`change size by x`	Changes sprite's size by *x* amount.
`hide`	Hides a sprite from the stage.
`next costume`	Changes sprite's costume to the next costume in the list.
Command	**Description**
`say xxx`	Shows sprite's speech bubble saying *xxx*.
`set size to x`	Sets a sprite's size to *x* percent of its original size.
`show`	Makes a sprite appear on the stage.
`switch to background x`	Changes the background of the stage.
`switch to costume x`	Changes the costume of a sprite.
`think xxx`	Shows sprite's thought bubble thinking *xxx*.
Variables Blocks	
`Change variable by x`	Changes the variable by *x* amount.
`Make a variable`	Creates a new variable that you can name for either a single sprite or for all sprites.
`Set variable to x`	Sets the variable value to *x*.
Sensing Blocks	
`key x pressed`	If *x* key on the keyboard is pressed then reports true.
`touching color x`	If a sprite is touching a designated colour then reports true.
`touching x`	If a sprite is touching designated sprite, edge or mouse cursor, then reports true.

Achievement Unlocked: You have created a program using Scratch!

In the Next Adventure

In the next adventure, you learn how to program art using Turtle Graphics. In the first half, you use Scratch. In the second half, you use the programming language Python, and you're introduced to the Python programming environment and some commands to enable you to draw shapes and repeat them to make patterns.



Adventure 4

Programming Shapes with Turtle Graphics

SUPPOSE YOU COULD pick up a turtle, dip his tail into coloured ink, place him on a piece of paper and make him walk around so that his tail paints a spiral shape, a pentagon or a noughts and crosses grid? This adventure introduces you to different ways that you can create shapes or line drawings using code.

You use a module called *Turtle Graphics* that works by directing a cursor (*or turtle*) around the screen using movement instructions; see an example of the result in Figure 4-1. This movement leaves a colour trail like a pen, which means you are able to program a computer to draw. Turtle Graphics was originally a feature of the programming language LOGO (Logic Oriented Graphic Oriented), which was designed to teach young people how to program using a logical sequence of steps by means of an onscreen cursor called *a turtle*. LOGO continues to be a very popular way to learn logic and sequencing in computer programming. Both Scratch and Python include `turtle` modules that can be used to create shapes, drawings and patterns.

This adventure draws on many of the computing concepts you have already used in previous tutorials in this book, such as sequencing, variables and loops, to create shapes and spirals in both Scratch and Python programming environments on the Raspberry Pi.

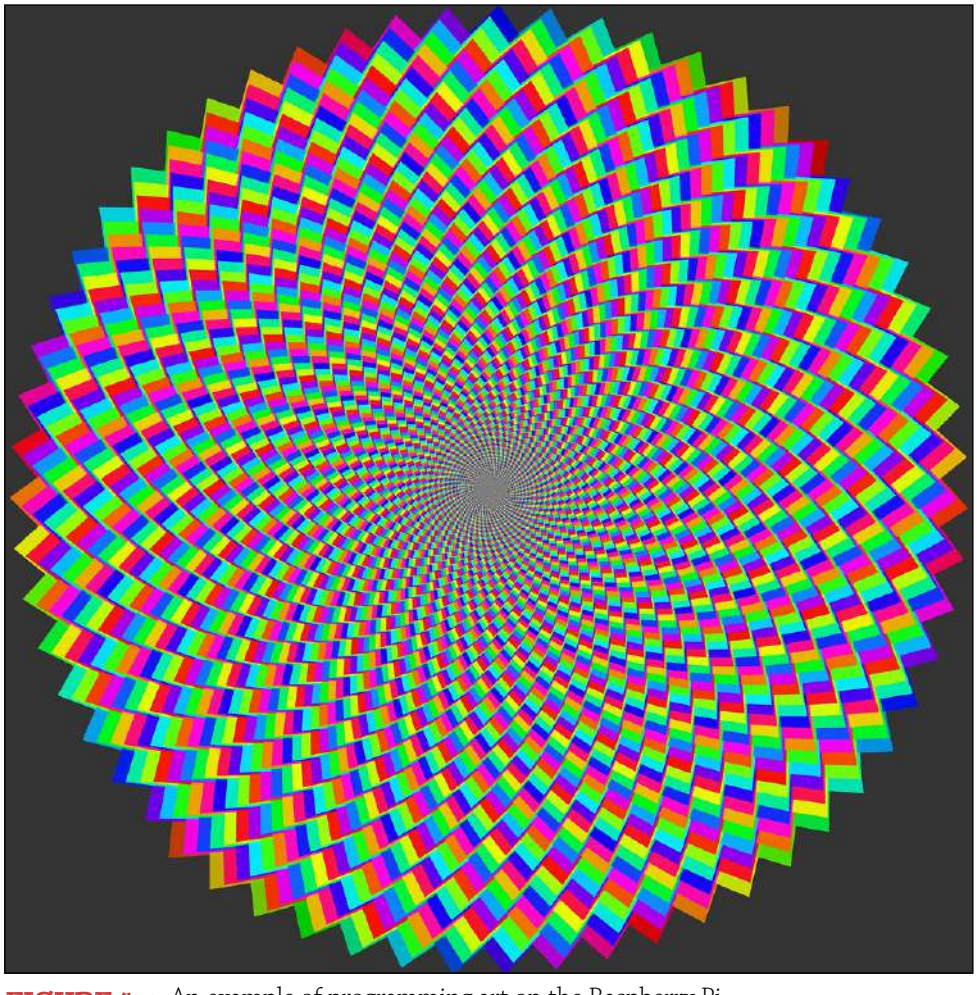


FIGURE 4-1 An example of programming art on the Raspberry Pi

Scratch Turtle Graphics

In this part of the adventure, you learn to use the basic features of Turtle Graphics in Scratch by writing a script that turns any sprite into a "pen" to draw lines and shapes on your stage.

To see a tutorial on creating shapes with Scratch, visit the companion website at www.wiley.com/go/adventuresinrp3E. Click the Videos tab and select the ScratchShapes file.

Using Pen Down and Pen Up

The `pen down` and `pen up` blocks instruct the sprite to start and stop drawing, in the same way as you touch your pen to paper to draw a line and then lift it.

The "turtle" in the following project refers to whatever sprite you choose to act as your pen. It doesn't have to look like a turtle—in fact, the sprite library doesn't include a turtle image, but you can create one using the Paint Editor if you like.

To draw some of the shapes and spirals in this adventure using Scratch, you need lots of space on the stage. Sometimes the "turtle" can go off screen, and this can result in messy shapes! To make sure that you have plenty of space, click the Switch to Full Stage icon above the stage on the right-hand side of the screen.

1. Open Scratch as you did in Adventure 3 by using the main menu to select Programming➪Scratch. Once the application has loaded, maximise the size of the stage by clicking the Switch to Full Stage icon above the stage on the right side of the screen.
2. You can use the default Scratch Cat sprite as your "turtle". However, you may find it easier to give directions to the sprite if you have a bird's-eye (top-down) view. To change the sprite to a bird's-eye view of the Scratch Cat, right-click the Scratch Cat sprite in the Sprites palette and select Delete from the menu. Next, click Choose New Sprite from File and browse to select Cat2 inside the Animals directory. (Refer to Figure 3-3 from Adventure 3 for a reminder of how the Scratch interface is set up.)

Keep in mind that the sprite directing the cursor in Turtle Graphics is always called a turtle, even though the image might be a cat, a person or something else.

3. Remember when starting any script in Scratch you need a trigger, so select the `When` `clicked` control block and drag it onto the Scripts tab of the turtle sprite.
4. Add the `pen down` block from the Pen blocks palette. This begins the drawing.
5. Add the motion block `Move 10 steps` and change the value to 100.
6. Add the `pen up` block to end the line drawing. Figure 4-2 shows the result.
7. Select File➪Save As to name and save your file.

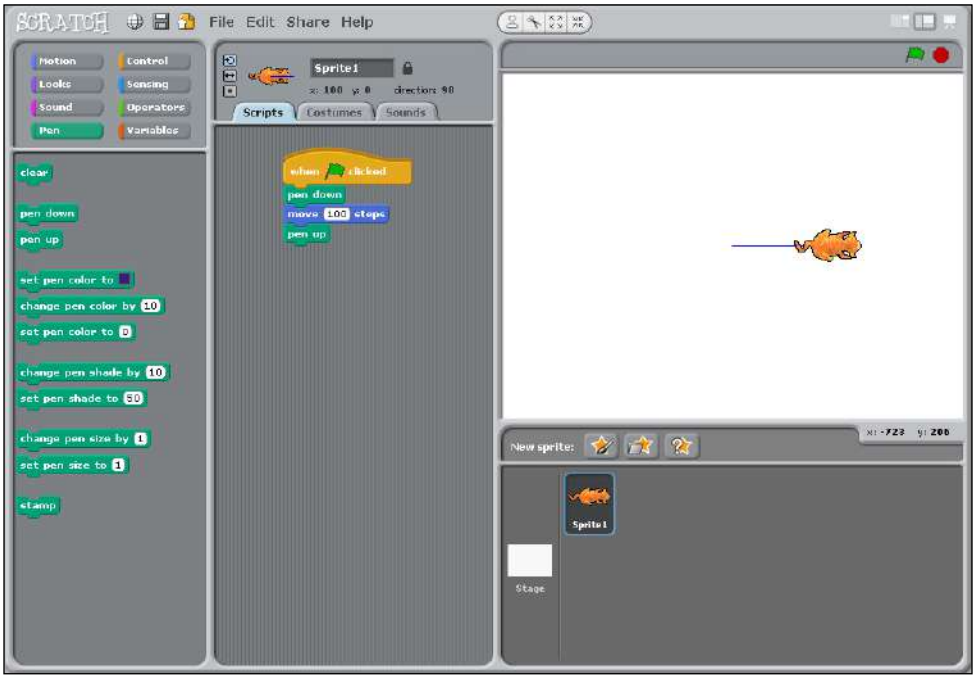


FIGURE 4-2 Using `pen down` and `pen up` to draw a line in Scratch

CHALLENGE

How could you add to the script to turn the turtle round and draw another line?

What does the `pen up` command do?

Drawing Simple Shapes

Now draw a pentagon using the script that you already created, by using the following steps.

1. Add the motion block `turn 15 degrees` underneath the `move 100 steps` block and change the value from 15 to 72.
2. So far, the script will have drawn only one side of the five-sided pentagon shape. You could add five more `move` and `turn` blocks to the sequence, or you could just add a loop to repeat, or **iterate**, the instruction five times. Add the control block `repeat` under the `pen down` block to contain the motion blocks and change the value to 5 (see Figure 4-3).

3. Click the green flag to test that your turtle draws a pentagon.
4. Click File➪Save As to save your changes.

The turtle can turn 360 degrees in a circle, either left or right. This helps you draw shapes. For example, to draw a square you would change the value of degrees to 90 to get the right-angled turn needed before drawing the next line.

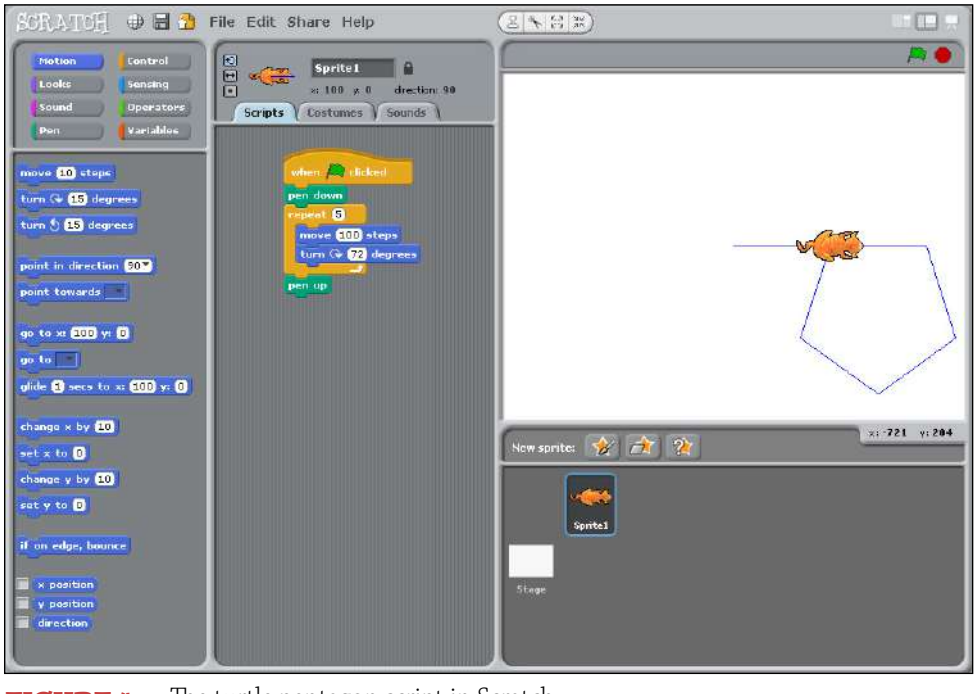


FIGURE 4-3 The turtle pentagon script in Scratch

CHALLENGE

If the value of steps and the value of degrees to turn were set to 1, and the repeat value set to 360, what shape do you think the turtle would draw?

Now that you have created a pentagon shape, how could you go about making a hexagon shape, or an octagon shape?

Using "clear" and Setting a Start Point

You may have noticed that every time you press the button, the image you have just drawn remains on the screen. This can be frustrating when you are testing scripts. To ensure you have a clear stage every time you run a script, add the `clear` block from the pen palette underneath the starting control block `When clicked`, as shown in Figure 4-4. This block tells the program to remove the previous action before proceeding.

Also notice that the turtle begins its drawing wherever it is located. This can affect your drawing, and part of it may end up offscreen if the turtle is positioned too close to the edge of the stage. To avoid this, you can add the motion block `go to x: 0 y:0` to set the start point of the pen, just like you did in Adventure 3 for the Adventurer sprite in the final game project. Remember that Scratch uses x and y coordinates, with `x:0` and `y:0` being the middle of the stage. You can also set the direction that the sprite faces before it draws your shape by using the motion block `point in direction 90` after the start coordinates. Figure 4-4 shows these blocks added to the script.

Using Variables Instead of Values

It is more logical to set **variables** for values that you want to use several times in programming. In your pentagon drawing, you used values for the length of the side of the shape (100 steps), the angle of the turn (72 degrees) and the number of sides (repeat 5). In this part of the project, you create variables that make it easier to create similar shapes in the future.

1. In the blocks palette, click Variables ➪ Make a Variable. You need to make three variables, called `Number_Sides`, `Angle` and `Side_Length`.
2. Drag the three new variable blocks onto the script and set the value of `Number_Sides` to 5, `Angle to` 72 and `Side_Length` to 100.
3. Underneath the variables blocks, add your simple shape script, but now instead of the values that you typed into the boxes, add the variable names (see Figure 4-4).
4. Now that you are using variables, you no longer need to calculate the angle of rotation. Instead, you can set the `Angle` variable to divide 360 (the number of degrees in a circle) by the number of sides that you set. To do this, drag the operators block `0 / 0` and replace the value 72 with it. Then type 360 into the left box, and drag the variable block `Number_Sides` into the right box.
5. If you change the number of sides to 6, the script draws a hexagon; if you change the number of sides to 4, it draws a square, and so on.

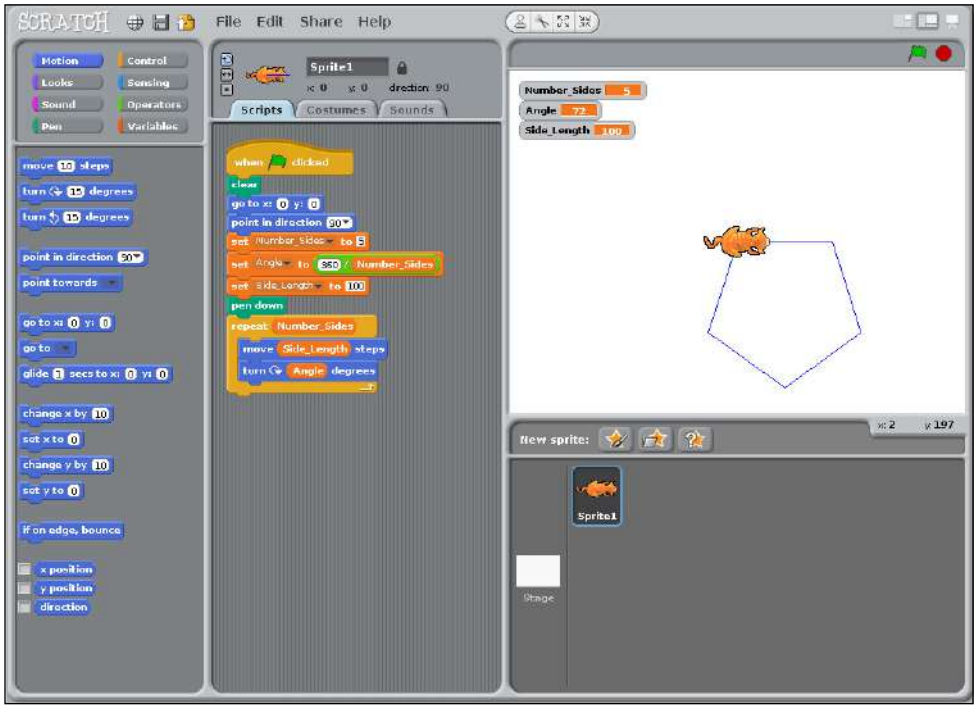


FIGURE 4-4 Using variables instead of values in Scratch

Changing the Size and Colour of the Pen

To make your drawings look more interesting, you can also change the colour of the pen and the thickness of the line.

1. Add the pen block `set pen colour to` to your pentagon script, after the `point in direction 90` block and before the `repeat` block.
2. Click the coloured square and use the eyedropper tool to select the shade you want to use from the colour palette.
3. You can also use the `set pen size to` block to change the thickness of the line. The higher the value you set, the thicker the line will be. Add this block to the script under the `set pen colour to` block and before the `repeat` block, and change the thickness of the pen line used in the pentagon script to 5.
4. Click the green flag to run the script. Figure 4-5 shows the results.

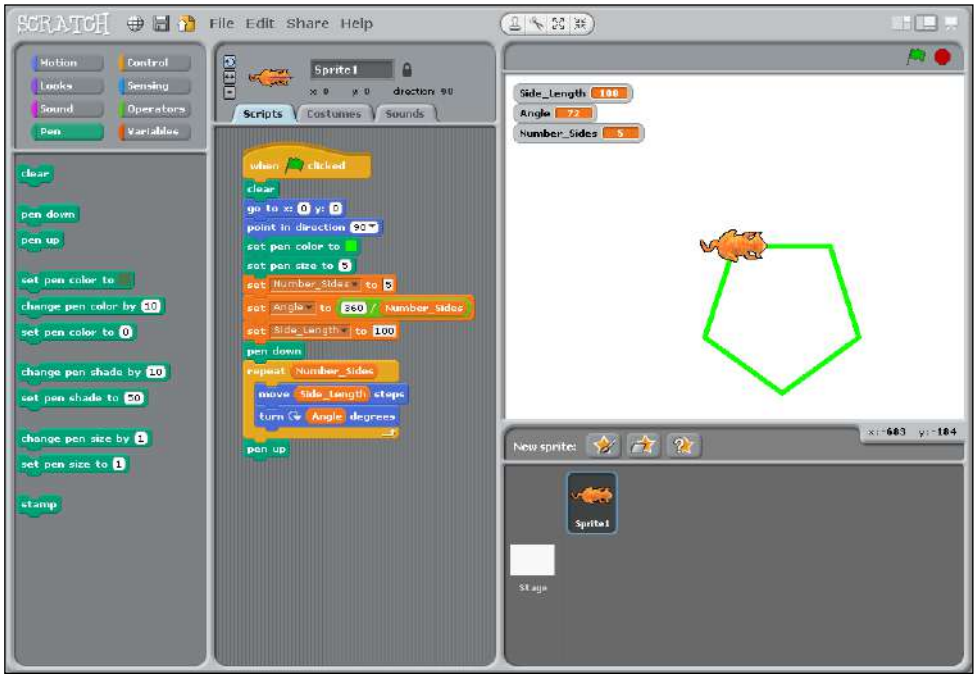


FIGURE 4-5 Setting the colour and size of the pen in Scratch

Creating Spiral Patterns

Once you have mastered drawing a single shape, you can start to think of ways to repeat the shape over and over to make a spiral pattern.

Add to the pentagon script a `repeat` control block and a `turn degrees` block, so that it looks like Figure 4-6.

To make your spiral look more colourful, you can add the pen block `Change pen colour by 10` underneath the motion block `turn 15 degrees`. This changes the colour of the pen after each pentagon shape has been drawn. Figure 4-7 shows the final script and the result.

Using User Input to Determine the Number of Sides

It is always more fun to allow a user to interact with a program you have created. In your turtle script, you can ask a user to set the value for the number of sides a shape in the spiral can have.

1. Add the sensing block `ask What's your name? And wait` underneath the start point blocks and before the set variable blocks.

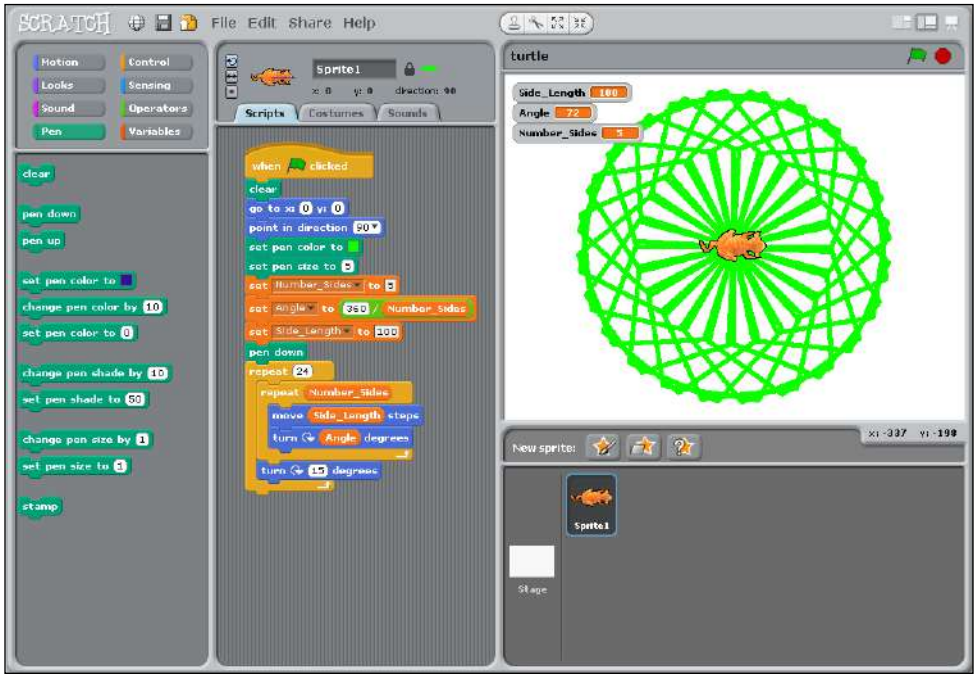


FIGURE 4-6 Creating repeating pentagon spirals

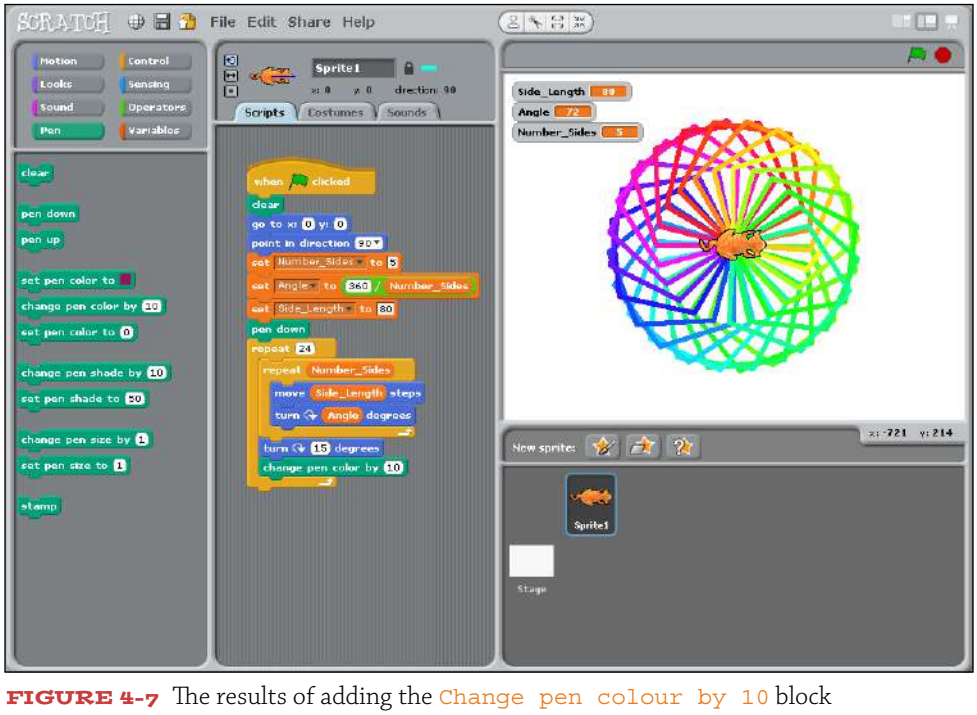


FIGURE 4-7 The results of adding the `Change pen colour by 10` block

2. Change the question to "How many sides would you like your shape to have?"
3. Add the sensing block `answer` to the `set Number_Sides` block, where you normally type the value. Your script should look like the one shown in Figure 4-8.
4. Now run your script. It asks the user how many sides she wants before it draws the spiral.

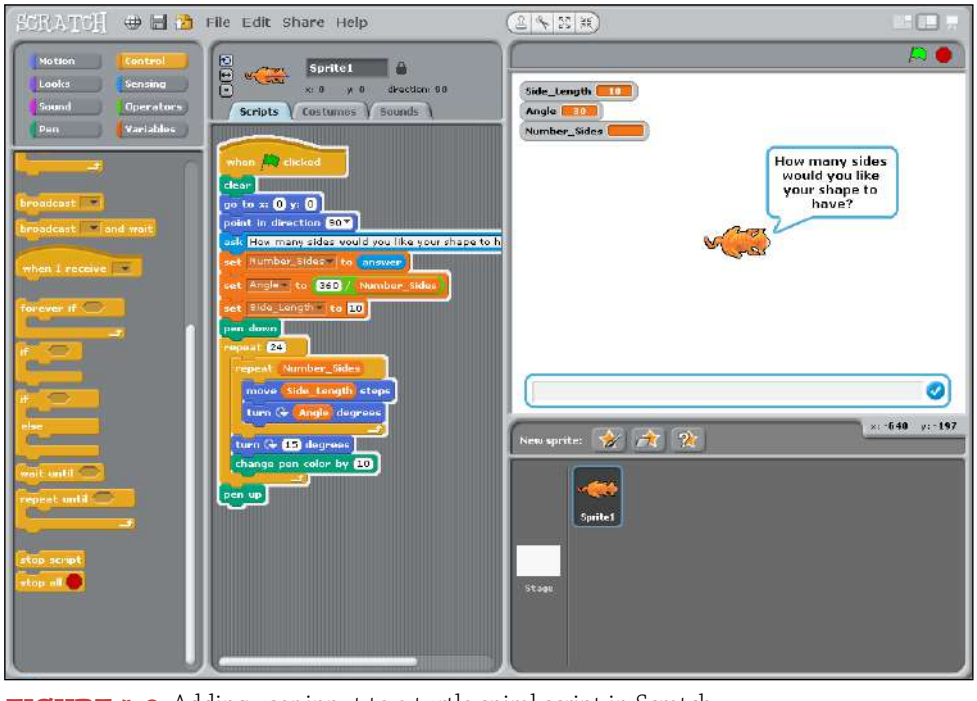


FIGURE 4-8 Adding user input to a turtle spiral script in Scratch

Python Turtle Graphics

This section gives you a quick taste of the Python programming language. Python includes a `turtle` module that you can use to create shapes and spirals in a similar way to Scratch. In this tutorial, you use the `turtle` module as an introduction to writing code in Python.

In Adventure 5, you get a more thorough introduction to the Python programming language, the IDLE programming environment, and Python functions and modules. In this adventure, you can just follow along with the instructions, and you begin to see how the Scratch blocks correspond to Python coding.

For a video that walks you through using the Python interface to type commands, visit the companion website at www.wiley.com/go/adventuresinrp3E. Click the Videos tab and select the PythonIntro file.

Introducing Python Modules

As you learn more about programming and continue to write code, you will discover that many of the programs you write include similar tasks and require similar blocks of code. To avoid the necessity of rewriting the same code over and over, most programming languages include reusable blocks of code, called **modules.** Python has a large number of modules containing useful code that you can reuse. You learn more about modules in Adventure 5.

In this adventure, you use the Python `turtle` module to create graphics.

The Python 3 Environment and the Interpreter Window

To use Python, you need access to the programming environment Python 3, which is also known as IDLE. To open Python 3 (IDLE), click the main menu and select Programming➪Python 3, as shown in Figure 4-9. You can type commands directly into the Python 3 window after the prompt, which is represented by three `>>>` characters. You type a line of code and then press Enter to run it. This window is referred to as an **interpreter** (or a shell), as it understands the language you are using, in this case Python, and interprets the code one line at a time.

You learn more about the Python 3 programming environment in the next adventure.

Using the Turtle Module in Python

In the first part of this project, you use the Python 3 interpreter, or shell, to add the `turtle` module for use in Python, and write the code to create a shape.

1. To use a module within a Python program, use the Python keyword `import` followed by the name of the module. In the window, type the following line after the `>>>` prompt to import the `turtle` module:

```
import turtle
```

   Now press Enter on your keyboard to get a new prompt.

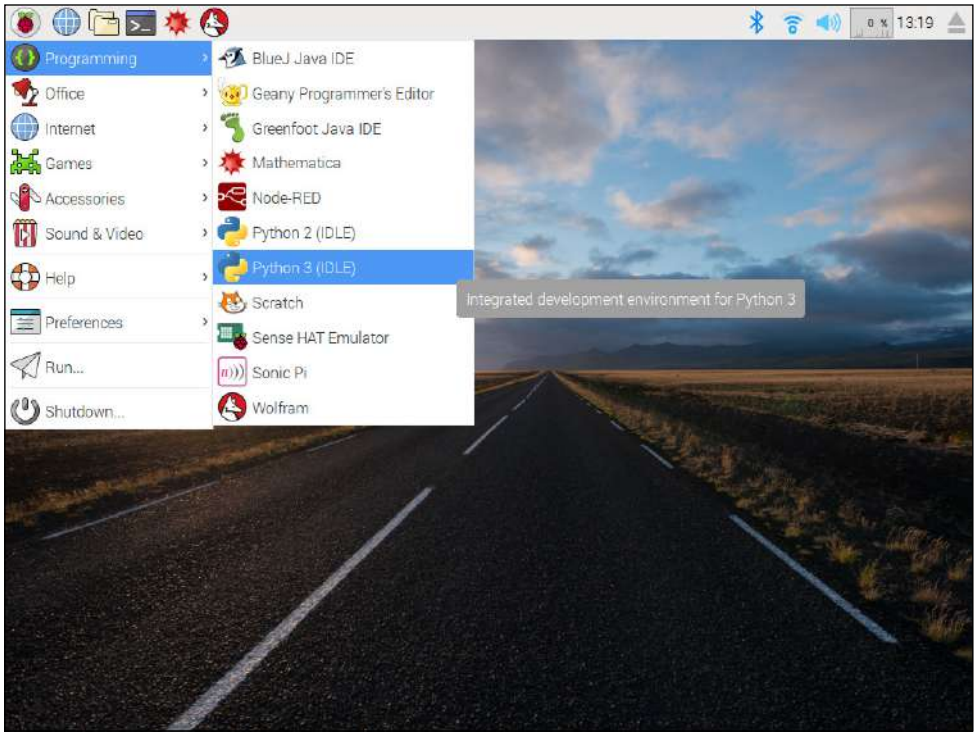


FIGURE 4-9 Opening Python 3 (IDLE) from the application launcher on the Raspberry Pi

2. At the new prompt, type the following command, and then press Enter again:

```
alex = turtle.Turtle()
```

This command opens the Turtle Graphics window, with an arrow cursor in the centre. The arrow cursor represents the turtle, whose movements create your drawing.

The = symbol in Python assigns a name on the left side to whatever is on the right side. This makes it easier to refer to it when writing lots of code. I have used the name alex, but you can use any name.

You can change the arrow to a turtle shape, as shown in Figure 4-10, by typing the following code at the prompt:

```
alex.shape("turtle")
```

3. Just like in Scratch, you can create a pentagon shape by moving so many steps and turning so many degrees. Type the following code, pressing Enter after each line:

```
alex.forward(100)
alex.left(72)
```

Your Python Turtle drawing will appear in a different window when you run the commands. Sometimes windows overlap and you can't see what is happening on both of them, or you may get a white screen instead of your drawing. To get over this problem, move the windows so that they are side by side as you're typing commands into the shell.

These lines tell the turtle to move forward 100 steps and then turn 72 degrees to the left. How many times would you need to type these two lines of instructions or code into the Python shell to draw a pentagon? Continue repeating these lines until you have created the pentagon. Figure 4-10 shows the final code and the completed shape.

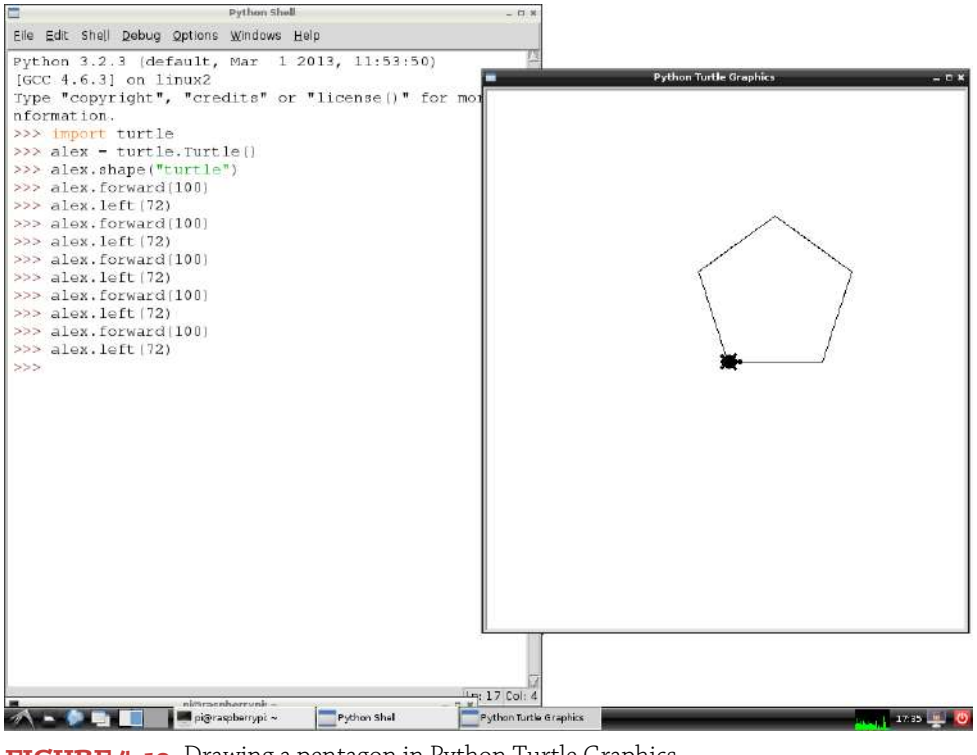


FIGURE 4-10 Drawing a pentagon in Python Turtle Graphics

You cannot save this code because you wrote it directly into the Python interpreter window or shell so that you could see it working instantly. In the following sections, you type your sequence of steps for the turtle to follow in a text editor window.

You can learn more about using Turtle Graphics with Python by selecting the PythonTurtleShapes video from the companion website at www.wiley.com/go/adventuresinrp3E.

Using a Text Editor

As you begin to create more complex programs, it becomes tiresome to type the commands straight into the Python interpreter or shell window. It makes more sense to type all your code into a text file before running it, using Python 3.

To open the text editor, click File➪New File from the menu at the top of the Python 3 interpreter or shell. Type all the code that you typed into the interpreter in the preceding section, and save the new file to your `Documents` directory on your Raspberry Pi as `FirstTurtle.py` (see Figure 4-11). You can run the script by clicking Run➪Run Module from the text editor toolbar.

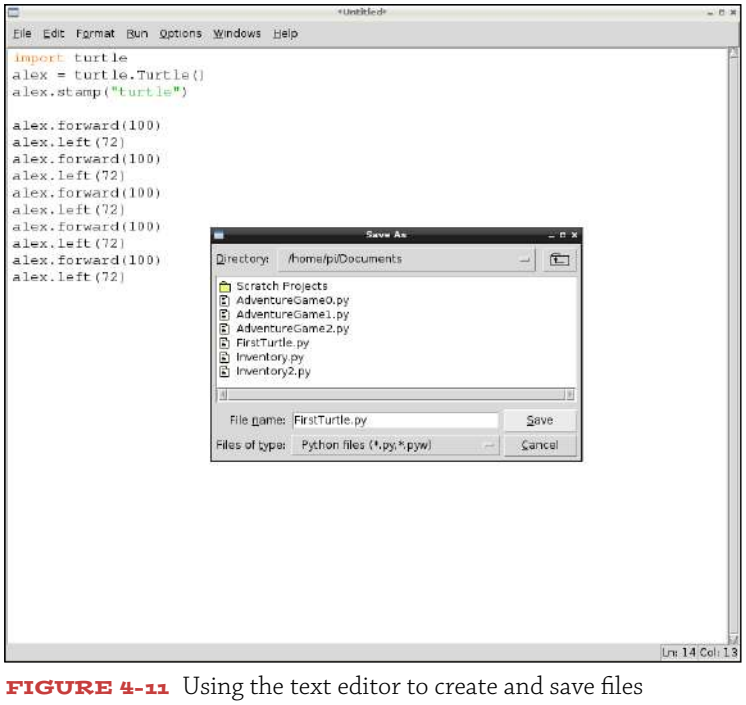


FIGURE 4-11 Using the text editor to create and save files

You should use the text editor for the remainder of this adventure.

Using for Loops and Lists

In your Python code so far, you have repeated the length of the line and the angle to make a pentagon shape by writing them in sequence. Repeating sequences is a common practice in computer science. You can make this code more efficient by writing the sequence once and then **looping** it five times. You have used looping before in Scratch, when you used the forever block to make an action continue repeating. As you learned in Adventure 3, each repeated instance of the looping code lines is called an **iteration**.

> Quite often, you will want your code to repeat or loop. In Scratch, you use the `repeat` or `forever` blocks to iterate. In Python you can use a `for` loop.
>
> 
>

To practice looping code, open a new text editor window and type the following code, saving the file as `FirstTurtle2.py`.

```
import turtle
alex = turtle.Turtle()
alex.shape("turtle")
```

Next add a `for` loop:

```
for i in [0,1,2,3,4]:
    alex.forward(100)
    alex.left(72)
```

This code says, "for each instance (`i`) in the following list, move alex forward 100 steps and then turn left 72 degrees".

When you have finished typing the code, run it by selecting Run ➪ Run Module.

The `for` statement repeats `forward` and `left` five times, one time for each value in the list. A list is represented in Python by square brackets. Numbered lists begin at 0 rather than 1. If you had written 0,1,2,3, inside the square brackets to form a list then only four sides of the pentagon shape would be drawn. Likewise if you had written 0,1,2,3,4,5 then six sides of the pentagon shape would be drawn, which is one side too many! Have a go yourself to see how numbering inside Python lists work.

By using a loop to repeat a sequence of code, you have saved yourself a number of lines of code. By using iteration in your program you are thinking like a computer scientist.

Lists can contain more than numbers or integers. For example, they can contain information to change the colour of the turtle pen.

Amend your Python pentagon code to look like the following, making sure to include the letter `a` before `color`:

```
import turtle
alex = turtle.Turtle()
alex.shape("turtle")
for aColor in ["red", "blue", "yellow", "green", "purple"]:
    alex.color(aColor)
    alex.forward(100)
    alex.left(72)
```

Save the file as `FirstTurtle3.py` and run the module. You now have a more colourful pentagon shape, as shown in Figure 4-12.

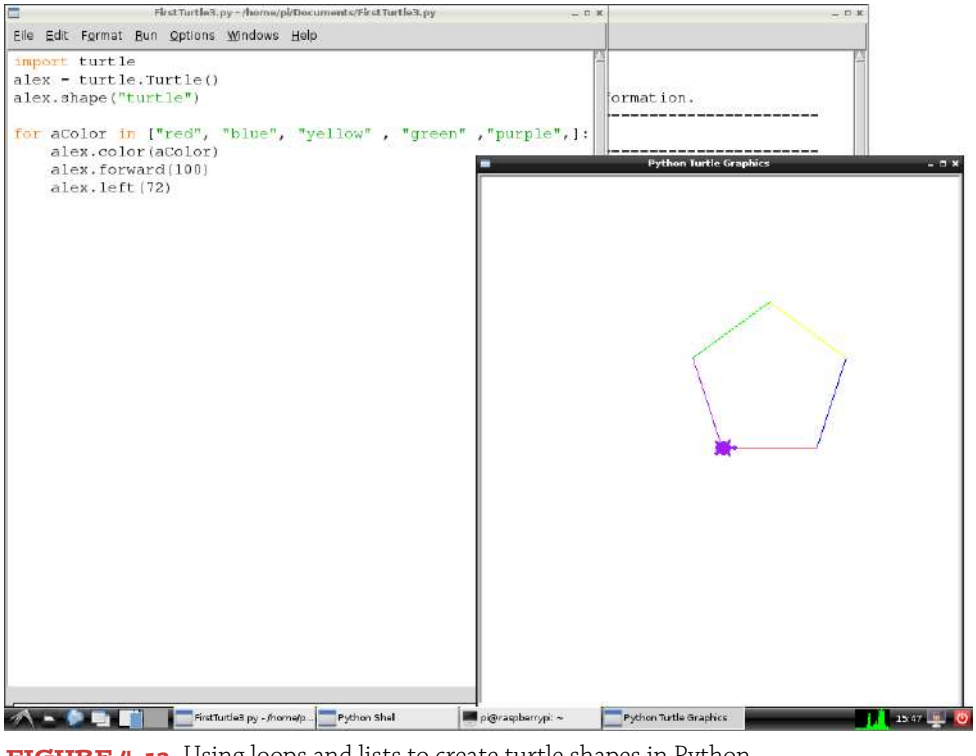


FIGURE 4-12 Using loops and lists to create turtle shapes in Python

The range Function

In the last few steps you used a list of numbers or integers and then colours to loop the turtle sequence in Python. Using lists is a very common coding task, especially if you are looping steps a number of times. It is so common that there is a Python **function** that you can use instead, the `range` function.

> A **function** is a piece of code that you can use over and over. You learn more about functions in Adventure 5.
>
> 
>

In a new text editor window, type the following code and save it as `FirstTurtle4.py`. When you have finished typing the code, select Run➪Run Module to see the code in action (see Figure 4-13).

```
import turtle
alex = turtle.Turtle()
alex.shape("turtle")
for i in range(5):
    alex.forward(100)
    alex.left(72)
```

The `range` function in this program creates a list of numbers or integers in the same way as the list you used before, `[0,1,2,3,4,]`.

DIGGING INTO THE CODE

The casing of commands used in Python code is very important, otherwise your code will not worvk as expected and you may get errors. In the examples in this adventure most of the code is in lower case, except when creating the "alex" turtle. The first "turtle" in the line `alex = turtle.Turtle()` is lower case (`t`), but the second one is upper case (`T`).

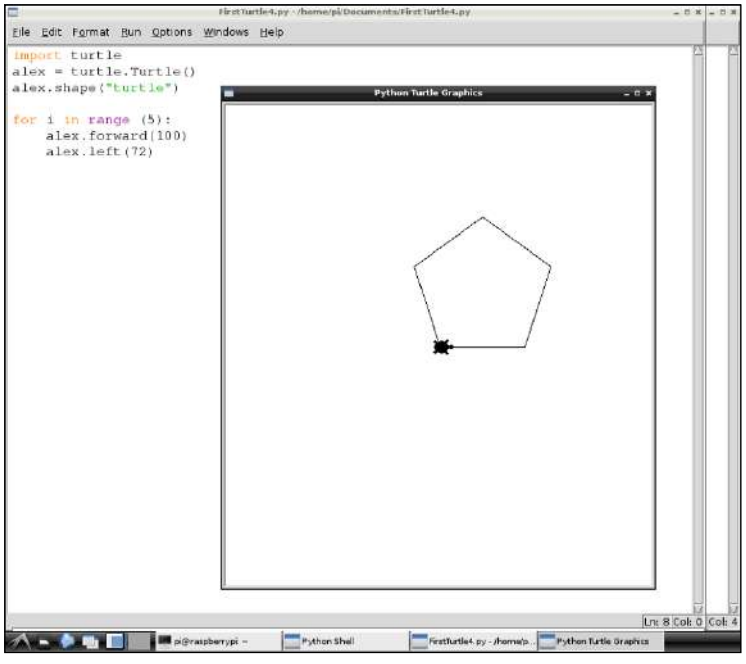


FIGURE 4-13 Using the `range` function to loop in Python

Other Python Turtle Module Commands

Once you have mastered some Python Turtle basics to create simple shapes, you can start to add extra lines of code to make your computer drawings more interesting.

Pen Up and Pen Down

Just as in Scratch, the Python `turtle` module includes code for the pen up and pen down commands so that you can move the turtle cursor around the page without leaving a line, just as if you were drawing a shape on a piece of paper with a pen. The code is written as follows, surrounding the directional code.

```
alex.pendown()
alex.forward(100)
alex.penup()
```

Setting the Pen Colour and Size

You can set the colour of the turtle using `.color` followed by the name of the colour you want to use inside brackets:

```
alex.color("blue")
```

Similarly, you can set the size of the pen by using `.pensize` followed by the number of pixels you want to use inside brackets:

```
alex.pensize(5)
```

Stamping

You can use `.stamp` to leave an imprint of the turtle cursor on the screen to form a pattern instead of, or as well as, using a pen line:

```
alex.stamp()
```

You can see the stamp in action in Figure 4-15.

Some Super Spirals

You can put together combinations of the Python Turtle code you have learned, in order to make some interesting shapes. Have a go yourself by typing the following two sequences into new text editor windows and saving them as `SpiralTurtle1.py` (shown in Figure 4-14) and `SpiralTurtle2.py` (shown in Figure 4-15). You can change the `pensize` and `color` arguments to make your own creations.

The Spiral Turtle

```
import turtle
alex = turtle.Turtle()
alex.color("darkgreen")
alex.pensize(5)
alex.shape("turtle")
print (range(5,100,2))
for size in range(5,100,2):
    alex.forward(size)
    alex.left(25)
```

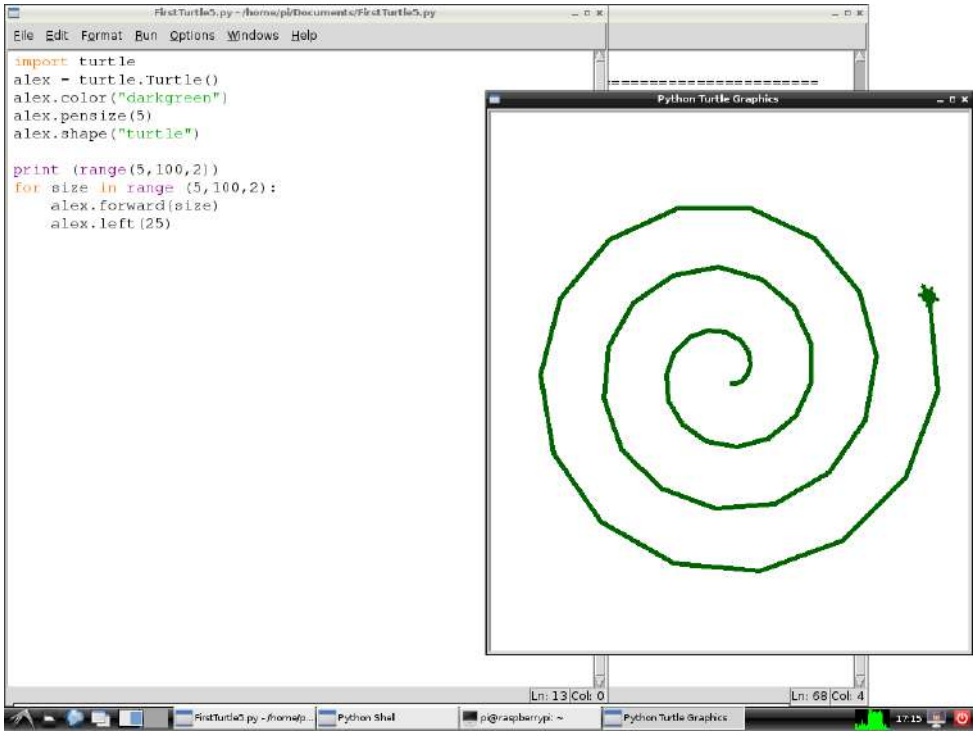


FIGURE 4-14 Using `pensize` and `color` to create `SpiralTurtle1.py`

The Spiral Turtle Stamp

```
import turtle
alex = turtle.Turtle()
alex.color("brown")
alex.shape("turtle")
print (range(5,100,2))
alex.penup()
for size in range(5,100,2):
    alex.stamp()
    alex.forward(size)
    alex.left(25)
```

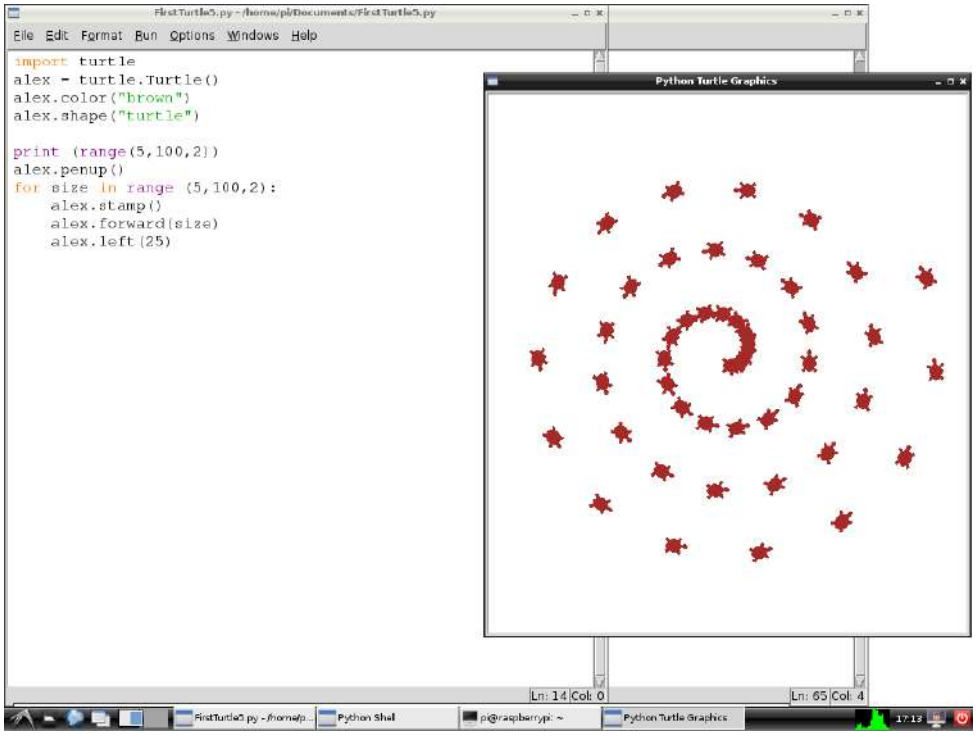


FIGURE 4-15 Using `penup` and `stamp` to create `SpiralTurtle2.py`

Further Adventures with Python Turtle

If you want to continue creating graphics using Turtle in Python, it's worth checking out the official Python Turtle online documentation at `http://docs.python.org/2/library/turtle.html`. It includes all the Python Turtle commands that you could use. Why not experiment and see what programming art you can create?

Turtle Graphics Command Quick Reference Table	
See also the Scratch Quick Reference Table in Adventure 3	
Commands	**Description**
Pen Blocks (Scratch)	
`change pen color by x`	Changes pen's colour by *x* amount.
`change pen shade by x`	Changes the pen's shade by *x* amount.
`clear`	Clears all pen marks and stamps from the stage.
`pen down`	Puts down a sprite's pen so that it will draw.

continued

Turtle Graphics Command Quick Reference Table continued	
Commands	**Description**
`pen up`	Lifts a sprite's pen so it does not draw.
`set pen color to x`	Sets a pen's colour to your choice.
`set pen shade to x`	Sets the pen's shade by *x* amount.
`set pen size x`	Set's a pen's line thickness to *x*.
`stamp`	Stamps a sprite's image on to the stage.
Turtle Module in Python	
`import turtle`	Imports the `turtle` module into Python. Should be at the start of any Python Turtle program.
Creating and Naming the "turtle"	
`alex = turtle.Turtle()`	Opens the Turtle Graphics window, with an arrow cursor in the centre, named *alex*. The arrow cursor represents the turtle, whose movements create your drawing.
Move and Draw	
`forward(x)`	Moves the turtle forward by the specified distance *x*, in the direction the turtle is headed.
`left(x)`	Turns turtle left by *x* units.
`right(x)`	Turns turtle right by *x* units.
`stamp()`	Stamps a copy of the turtle shape onto the canvas at the current turtle position.
Drawing State	
`pendown()`	Puts the pen down and draws when it moves.
`penup()`	Picks the pen up and stops drawing.
`pensize(x)`	Sets the thickness of the line drawn to *x* pixels.
Turtle State	
`shape("turtle")`	Sets the cursor icon. Possible values for the shape are arrow, turtle, circle, square, triangle, classic.
Colour Control	
`color("brown")`	Sets pen colour.
Additional Commands	
`for`	`for` loops are traditionally used when you have a piece of code that you want to repeat *x* number of times. Example: `for i in [0,1,2,3,4,]`
`for i in range():`	A for loop using the `range()` function that creates a list containing numbers.
`range()`	The `range()` function generates a list of numbers in progression.

Achievement Unlocked: You can create Turtle Graphics on your Raspberry Pi!

In the Next Adventure

In the next adventure, you learn more about how to program in Python on the Raspberry Pi. You use some of the same concepts you have already learned, such as iteration (loops) and conditionals (`if` statements), as well as many new constructs, to create a new game where players answer questions to determine how the game will move forward.

Adventure 5
Programming with Python

PROGRAMMING WITH SCRATCH can be a lot of fun, but as you become more skillful at creating games and graphics using this application, you may notice that there are limits to what you can achieve with Scratch. Most computer programmers use text-based languages to create computer programs, including games, desktop applications and mobile apps. Although text-based programming may seem more complicated at first, you will soon find that it is easier to achieve your goals by using code. The Python code language is used by millions of programmers worldwide, including developers at organisations like NASA, Google and CERN.

In this adventure, you discover what you need to set up Python on your Raspberry Pi. You write a short program and learn to use a text editor. After that, you delve deeper into Python, learning about modules and their applications, how to get user input and how to use conditionals. Finally, you put all your new knowledge to use, creating a text-based adventure game in which your user (player) answers questions and your game responds based on the answers.

Although this may seem like a major departure from the graphical world of Scratch programming, the good news is that all of the programming concepts you learned in Scratch apply to other languages as well, even those that seem very different. Concepts such as using a sequence of instructions to make something happen, loops, conditionals and variables are common throughout all programming languages. By the end of this adventure, you'll be able to write some basic Python programs on your Raspberry Pi!

If you get stuck on a computing problem, you might find it helpful to use Scratch to help you visualise what is happening.

Getting Set Up for Python

The Raspbian operating system includes the programming environment called Python IDLE. This section introduces you to the programming language and environment used to create Python files and execute them.

Python Programming Language

The Raspberry Pi operating system Raspbian comes with a text-based computer programming language, Python, already installed. The Pi in Raspberry Pi is a nod towards this programming language, as it is considered an easy language to pick up and is used by coders all over the world.

To work with Python, you use the IDLE programming environment, or **integrated development environment** (**IDE**).

An **IDE** or **integrated development environment**, also referred to as a *programming environment*, is a software application used to write computer code in a particular language, for example Python. The application has the capability to create and edit code as well as run or execute the code. Many IDEs also provide features to help programmers check for errors in their programs and *debug* or resolve the errors.

The IDLE Environment

To create programs on the Raspberry Pi using Python, you use the Python programming environment, which is called IDLE. Notice that there are two versions in the menu system: Python 2 and Python 3 (see Figure 5-1). The projects in this book require Python 3, which you use in Adventure 4. Just like the English language, Python has evolved through different versions, and some of the commands you learn in this adventure will not work in older versions of IDLE like Python 2 on the Raspbian.

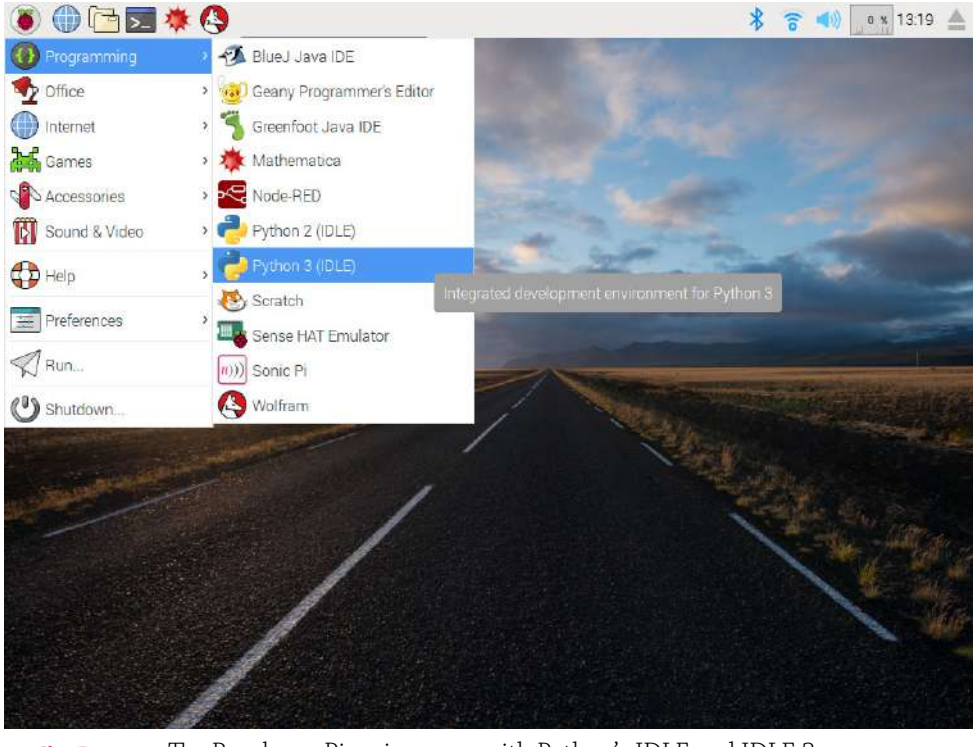


FIGURE 5-1 The Raspberry Pi main menu, with Python's IDLE and IDLE 3 programming environments (Python 2 and Python 3) both available

Programming in Python: Using a Function

To begin working with Python, open the main menu, navigate to Programming and select Python 3. You use Python 3 throughout this book rather than earlier versions, which use a different syntax. The Python shell, or command-line interface, opens and a prompt of three angle symbols (`>>>`) appears to indicate where you should type your code, as shown in Figure 5-2.

For your first Python program, you write only one line of code, using a **function**, a piece of code that tells the computer to perform a specific task. For this program, you use the `print()` function to tell the computer to print some text on the screen. Place the **string** of text you want the computer to display inside the brackets, with quotation marks around it.

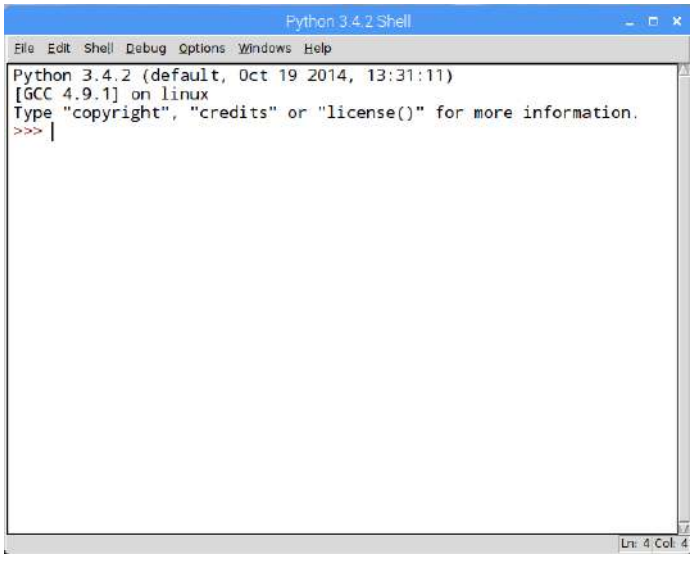


FIGURE 5-2 The Python 3 IDLE Shell

SYNTAX, ERRORS AND DEBUGGING YOUR CODE

Syntax is a set of rules to check whether the code you have typed is valid Python code. In the same way as the English language has rules about how to properly combine subjects, verbs, objects and so on, each programming language has its own syntax. When you make a mistake or a typo in your code, your program may display a **syntax error** message.

A syntax error stops a program from running because the computer cannot understand the code. This usually happens because a word was misspelled or a character left out. The most common cause of syntax errors is missing the colon at the end of loops and conditionals!

Error messages are posted to the screen to alert you to the problem, but these messages can be difficult to understand. You might want to make some typing mistakes on purpose with some simple example code, so you can see the sort of error messages Python gives you. Try leaving out a quotation mark or bracket, or misspelling a command, to find out what happens.

So what do you do when you get an error message? **Debugging** is the act of locating the cause of any errors in your computer program code and fixing them. When Python displays a syntax error, the line that contains the error is repeated, with a little arrow underneath it pointing to where the error is likely to originate. Look carefully at the line to spot any misspelled words or missing characters, then correct the problem and try running the code again.

When a program gets to be more than a few lines, it becomes harder to understand and edit the code. Breaking a program's code down into small sections makes it easier to read and edit. A **function** is an example of a small section of code that does a specific task, and once it is created you can use it over and over again. An added benefit of using functions is that if you fix a bug you only have to fix it in one place.

Like most programming languages, Python includes some standard functions that the computer already understands, like the Python 3 `print()` function that prints some text to the screen. You can also write your own functions, which you do towards the end of this adventure.

A **string** refers to data or information entered as text (i.e., a "string" of characters).

Place your cursor directly after the `>>>` prompt and type the following line:

```
print("I am an Adventurer")
```

Press Enter and see what happens (see Figure 5-3).

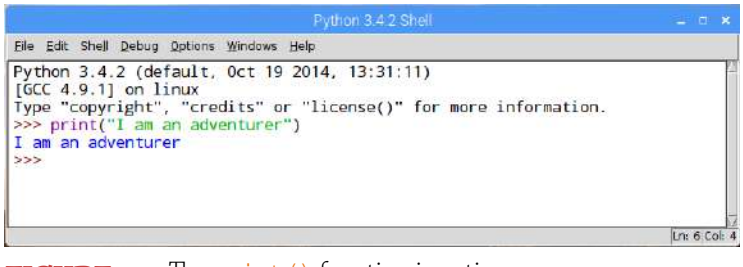


FIGURE 5-3 The `print()` function in action

When you press Enter, the Python Shell "interprets" your code. In this case, you gave the command to print the text between the quotation marks to the screen. Well done! You have created your first computer program in Python.

Using a Text Editor to Create a Code File

You use a text editor to create code files in Python in Adventure 4 when using the `turtle` module. As I explain in Adventure 4, it makes sense to type all your code into a text file using a text editor and save it, before you test that it works by running it using IDLE. Using a text editor has the added bonus of ***syntax highlighting***, which works by adding colour to different words in your code to make it easier to read. If you use a command-line editor like nano (which you used in Adventure 2) that does not have complete syntax highlighting, you may find it hard to read a long program. For the projects in this adventure, you use the Python 3 IDLE text editor as you did in Adventure 4.

In the Python programming language, you can create ***lists*** to store data—for example, you might want a list of names of the students in your class to use in a program that sends out invitations, or a list of favourite restaurants that your program could suggest when you need an idea for dinner.

The following steps walk you through using a text editor to create a list of objects that you will use later in your adventure game. In this exercise, you create a new file, add the code to create an inventory, and then save the file.

For a video that walks you through creating an inventory in Python, select the Inventory video from the companion website at `www.wiley.com/go/adventuresinrp3E`.

1. Open Python 3 and click File from the menu at the top. Select New File to open an untitled text file (see Figure 5-4). Notice that this creates a new text editor file, not a shell window, and therefore does not contain a prompt.

2. To save the file, click File➪Save As. Navigate to your `Documents` directory and name the file **Inventory** before clicking Save. If you open your `Documents` directory, you can see the file is now saved there and Python has added a `.py` to the end of the filename, so the complete filename is `Inventory.py`.

3. In the new file, type the following:

```
inventory = ["Torch", "Pencil", "Rubber Band", "Catapult"]
```

   This code creates a list named `inventory`. Each string, or piece of text data, represents an item on that list.

4. Underneath the list, type the following:

```
print(inventory)
```

5. Click File➪Save to save the file, and then run your small program by clicking Run➪Run Module. Your list is printed to the screen, as shown in Figure 5-5.

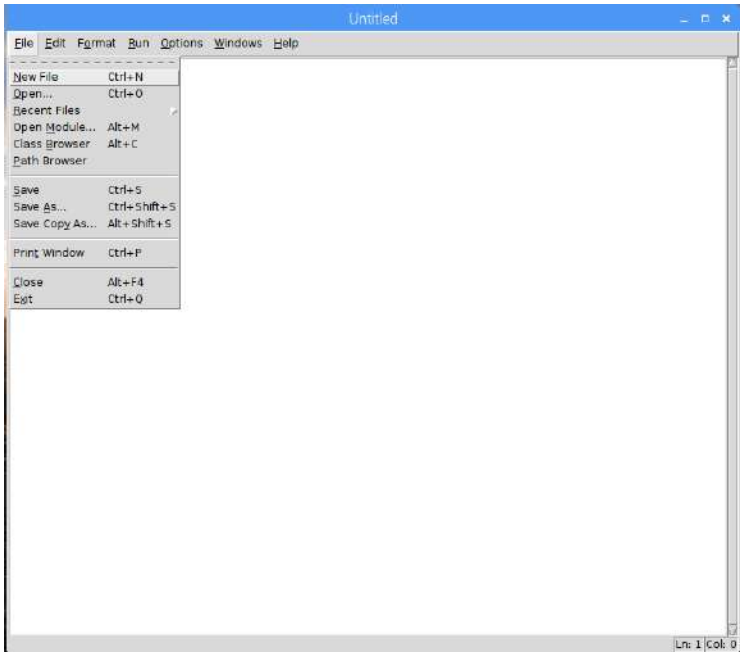


FIGURE 5-4 The Python 3 IDLE text editor and menu

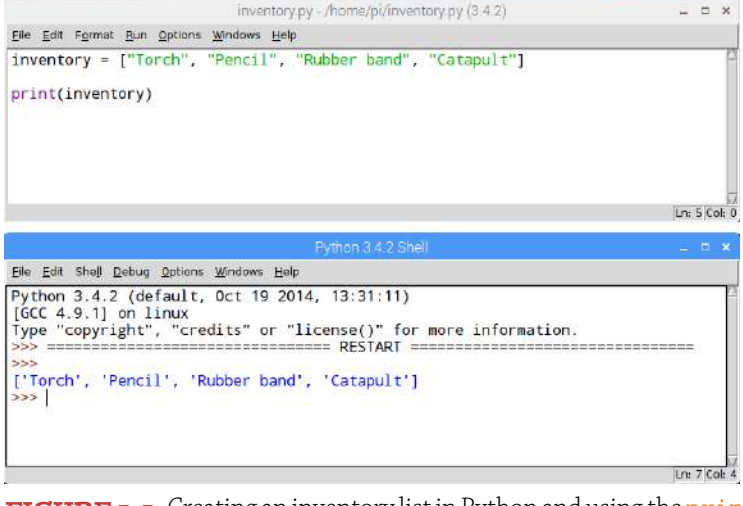


FIGURE 5-5 Creating an inventory list in Python and using the `print()` function to display its contents

6. Now adapt the last line of code to read:

```
print(inventory[3])
```

7. Save the file using File➪Save and then run the program again.

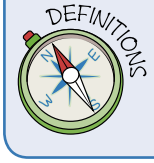
Your program should print `Catapult` as the output. Hang on—isn't the catapult the fourth object on the list and not the third? Why did it print the third object when the command was to print the fourth?

The third object was printed because Python numbers items in a list starting at 0, rather than at 1: 0 = Torch, 1 = Pencil, 2 = Rubber Band, 3 = Catapult.

Using the Python time and random Modules

As mentioned in Adventure 4, Python has a large number of **modules**—useful blocks of code that you can reuse to avoid having to rewrite the same code over and over each time you need a program that performs the same task. For example, each time you need a program that selects objects from a list randomly, you could write a new function or you could simply use the Python random module and save yourself a lot of time.

In order to use a module within a Python program, you use the Python word `import` followed by the name of the module. You can then access functions of that module in your program.



A **module** is a collection of reusable Python code that performs a specific function. It may be used alone or combined with other modules. In this adventure, for example, you use functions from the Python `time` module to add pauses.

In this section, you use the `random` module along with the inventory list you created in Python to make a program that selects an item from the list randomly.

Starting with the Python file `Inventory.py` that you created earlier, you adapt the code to create a new, interactive program that requests user input and responds appropriately.

As you write the code, you include **comments**. Comments are notes within your code that explain what a line or section of code is intended to do. Each comment line begins with the # symbol, which tells the computer running the program to ignore that line. If a comment wraps over several lines you need to include a # sign at the beginning of each line so that it is passed over by the IDE.

There are many good reasons for including comments inside your code. Comments can help you remember what each part of the code is doing, should you leave it unfinished for a while. In school, you may use comments to explain to your teacher what each part of your code is doing. If you are working with others, comments help them see what you have done already.

1. Begin your code with a comment line to indicate the code's purpose. Open the `Inventory.py` file that you created earlier and type the following line at the top (above the inventory list that is already in the file):

```
# Adventures in Raspberry Pi Python – Inventory
```

   Note the `#` symbol at the start of the line, identifying it as a comment.

2. Use the `import` command to import the two Python modules you need, `time` and `random`. You can add a comment to explain this step if you like, as shown in the following code:

```
# You will need the random module and the time module.

import random
import time
```

3. Press Enter to leave a blank line so that your code is easier to read and then use the `print()` function to display two strings of text on the screen:

```
# Enter a blank line here

print("You have reached the opening of a cave")
print("you decide to arm yourself with a ")
```

4. Use the `sleep()` function from the time module to make the program wait for two seconds before asking the player a question by adding the **argument** `(2)` as shown.

```
time.sleep(2)
```

An **argument** is a piece of information given to the function that it may use to perform its task. The argument goes inside the parentheses that follow the function name. In the code in step 4, for example, with the `time.sleep()` function you use the argument `(2)`, which is the number of seconds you want the program to wait before implementing the next line.

5. Now you want the player to think of any item and type it in as her answer. The following code displays the line `Think of an object` and waits for the player to enter an answer. The player can type anything for her answer; for example, she might type **banana**. The program then uses the `print()` function to display `You look in your backpack for banana` (or whatever object the player typed).

The ↩ character at the end of a code line means that line and the next one should all be typed as a single line; do not add a line break or extra spaces between them.

```
quest_item = input("Think of an object\n")

print("You look in your backpack for ", quest_item)

time.sleep(2)

print("You could not find ", quest_item)
print("You select any item that comes to hand from the ↩
  backpack instead\n")
time.sleep(3)
```

The `\n` at the end of the string in the first line doesn't get printed to the screen; instead, an extra new line is printed. This is helpful for breaking up the text and making it easier to read.

A function may produce a *return value*, which can be stored in a variable like any other value. For instance, the `input()` function returns the string that the player types, or the `random.choice()` function returns an item from the list it is given as an argument.

6. Next comes the inventory list. You created this line earlier, so just leave it as written:

```
inventory = ["Torch", "Pencil", "Rubber band", "Catapult"]
```

7. In the last part of the inventory program code, you use the `choice()` function from the `random` module to pick an object from the inventory list and display it to the user of the program. Type the following line below your inventory list:

```
print(random.choice(inventory))
```

Functions can take a number of arguments and return a result. Here you pass an argument to the `time.sleep()` function to tell the program how many seconds to wait, and then print the result of `random.choice`.

Figure 5-6 shows the completed code.

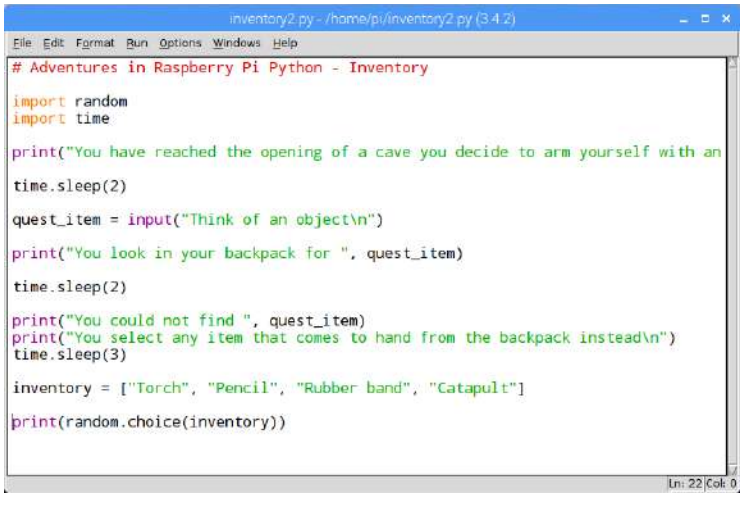


FIGURE 5-6 Using modules in Python 3 to create an Inventory program

Check that your program works by saving the file as `inventory1.py` in the `Documents` folder on the Raspberry Pi and then clicking Run ➪ Run Module from the toolbar. Enter input when the program prompts for it. You should see results similar to those shown in Figure 5-7.

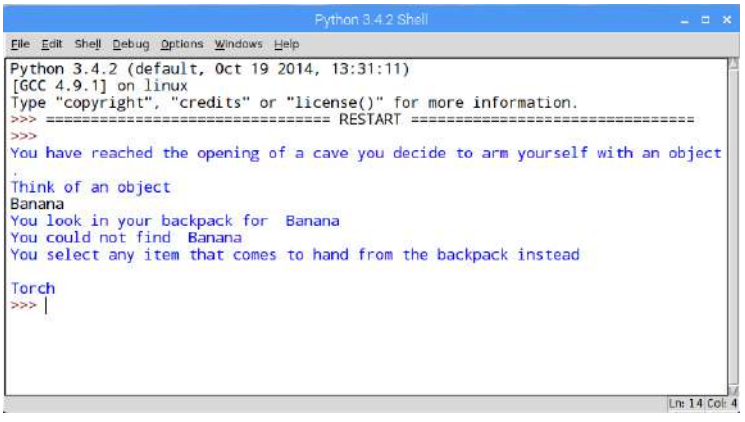


FIGURE 5-7 Getting a random item from the inventory list, using modules

What might happen if a user inputs the item **torch** when asked to think of an item? Which of the following do you think that the program will print?

- You could not find torch
- Found torch from your backpack
- A torch! This will shed some light on the matter!

The answer is A. The program prints whatever you type in the string. It does not currently look inside the inventory list.

Can you think of a way to improve this program so that it checks to see if the item is already in the list?

Python Text Adventure Game

Text adventure games are fun to create because they are stories you write for your friends and families can interact with. All you need is a bit of imagination and, of course, some programming skills.

In this tutorial, you create your own adventure game that uses text to direct the player through the game. The program asks the player to make decisions on what to do next. This may be as simple as finding out the direction in which she wants to turn next.

Visit the companion website at `www.wiley.com/go/adventuresinrp3E` and select PythonTextAdventure to see a video of this project.

Getting User Input

As the text adventure game relies on the player (user) to interact with the game to make decisions, you will need to use the `input()` function.

```
direction1 = input("Do you want to go left or right? ")
```

This line of code asks the player to answer the question, "Do you want to go left or right?" The program waits for the player to type an acceptable response—one that the program can understand.

Using Conditionals

After the player has responded, you want something to happen based on her answer. You therefore need to use a **conditional statement**. You used conditionals in your Scratch adventure game in Adventure 3 to control the movement of the adventurer sprite, using the `forever if` control block.

Remember that creating a conditional statement is like asking a question where there are two or more outcomes. For example, you could ask the question "Is it raining?" If the answer is yes, you should put on a raincoat; if the answer is no, you should go out without a jacket. The key word used here is "if".

You will use `if` in Python 3 to create your game conditions. Open a new Python IDLE 3 text editor window and save the file as `AdventureGame.py`.

1. Import the modules that you need for the program. As in the inventory program, you need the `sleep()` function from the `time` module, so import that module with the following code:

   ```
   import time
   ```

2. Later in this game, you may want to give your player health points that could go up or down depending on which directions she takes in the adventure. The number of remaining health points is stored in a variable. To include that feature, type the following line:

   ```
   hp = 30
   ```

3. Now use the `print()` function to tell your player where she is located in the game, and then use the `sleep()` function to wait one second before moving on.

   ```
   print("You are standing on a path at the edge of a ↩
     jungle. There is a cave to your left and a beach ↩
     to your right.")

   time.sleep(1)
   ```

4. As in the inventory program, you want to get input from the player of your game. In this case, you need to know if she wants to turn left or right, and the player's answer is then labelled as `direction1`.

```
direction1 = input("Do you want to go left or right? ")
```

5. Create conditions depending on the player's answer. You need one condition if the player chooses `left` and another if she chooses `right`. You may remember using conditionals in Scratch in Adventure 3. In Python, you use if, `elif` (else if) and `else` to check conditions:

```
if direction1 == "left":
  print("You walk to the cave and notice there is an ↩
    opening.")

elif direction1 == "right":
  print("You walk to the beach but remember you do ↩
    not have any swimwear.")

else:
  print("You think for a while")
```

`if`, `elif` and `else` are the Python words used to check conditions. In the preceding code, if the player types **left**, the program prints the statement, "You walk to the cave and notice there is an opening"; else if (`elif`) the player types **right**, the program prints a different piece of text. Finally, if the player types in any answer that is not `left` or `right` (`else`), the program prints, `You think for a while`. Figure 5-8 shows this code in the text editor.

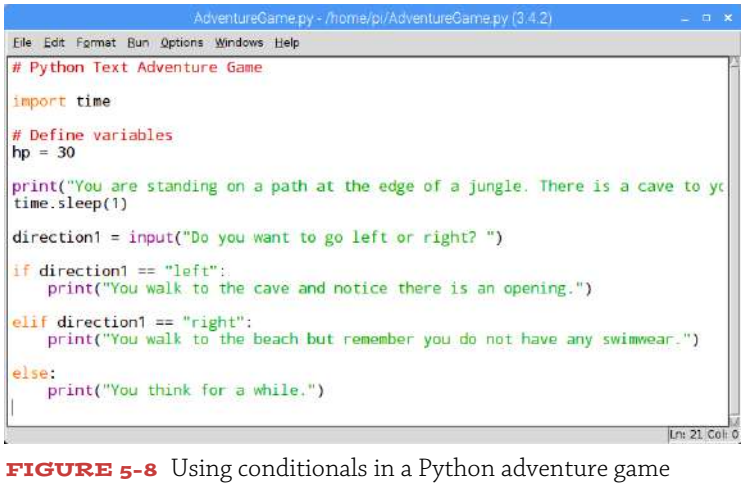


FIGURE 5-8 Using conditionals in a Python adventure game

DIGGING INTO THE CODE

In Adventure 3, you created conditions for the Scratch role-playing game using an `if...else` block like the one in Figure 5-9. In Scratch, the blocks for each part of a condition are automatically indented slightly for you within the `if...else` block. You can easily see which parts of the condition need to be met by how far they are indented. In Python code, you have to add indents to show which lines of code are part of the conditional. Python code also uses the colon (`:`) to show where you might need to indent. Take a look at Figure 5-9 to compare indentation in the `if...else` statements in Scratch and Python.

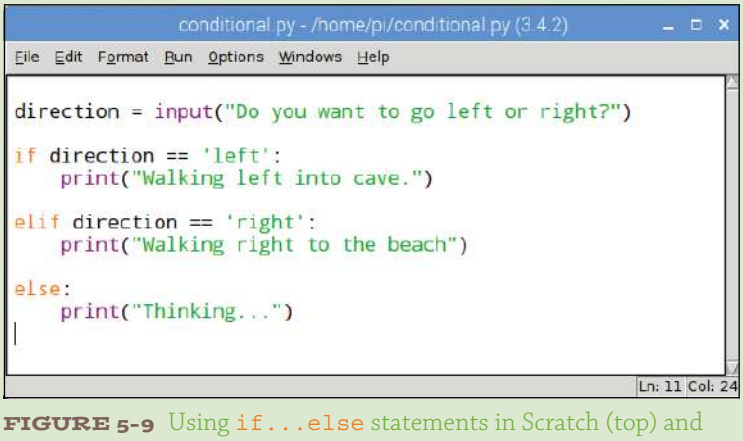


FIGURE 5-9 Using `if...else` statements in Scratch (top) and Python (bottom)

6. Test to see if your code works by first saving the file as `AdventureGame.py` in `Documents` on the Raspberry Pi, and then running it by selecting Run➪Run Module. You should see a display similar to Figure 5-10.

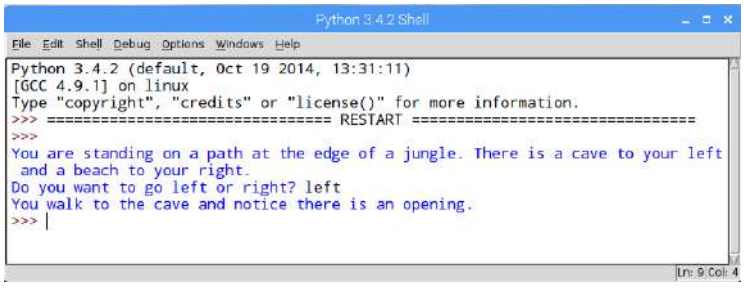


FIGURE 5-10 Using Run Module to test conditionals in Python adventure game

What happens if the player types `LEFT` or `RiGhT` instead of `left` or `right`? Will the condition still be met?

To make sure that the player types in the correct lowercase response that you require to meet the conditions, you can use a lowercase function so that if the player types in capital letters, the program turns the text into lowercase, which is a recognised response to the `if`, `elif`, `else` conditions.

```
direction1 = direction1.lower()
```

Add this line *before* the first `if` statement used, and *after* asking the player to input a direction. See the final game code towards the end of this adventure for reference.

Using a while Loop

So far, the player has not been required to input specific answers in order for the game to move on. If the player does not input anything at all, the game simply stalls; and if the player types an unrecognized answer, the game says, `You think for a while`. You want the player to input one of the responses that you have defined, `left` or `right`, to move on to the next location. You can ensure she inputs one of the desired responses by adding a `while` loop to your code. This loops the user input question until the player types in a response that you were looking for—**left** or **right**—to move on. For example:

```
# Loop until we get a recognised response
while True:
  direction1 = input("Do you want to go left or right? ")
```

```
  direction1 = direction1.lower()
  if direction1 == "left":
    print("You walk to the cave and notice there is an ↩
     opening.")
    break # leave the loop
  elif direction1 == "right":
    print("You walk to the beach but remember you do not ↩
      have any swimwear.")
    break # leave the loop
  else:
    print("You think for a while")
```

In this code, shown also in Figure 5-11, you can see in bold text the Python words `while True:` added before the user input question, and `break` added within the conditionals for left and right. The `while True:` condition loops the question over and over until the player enters either **left** or **right** so that the game does not end if the player types anything else.

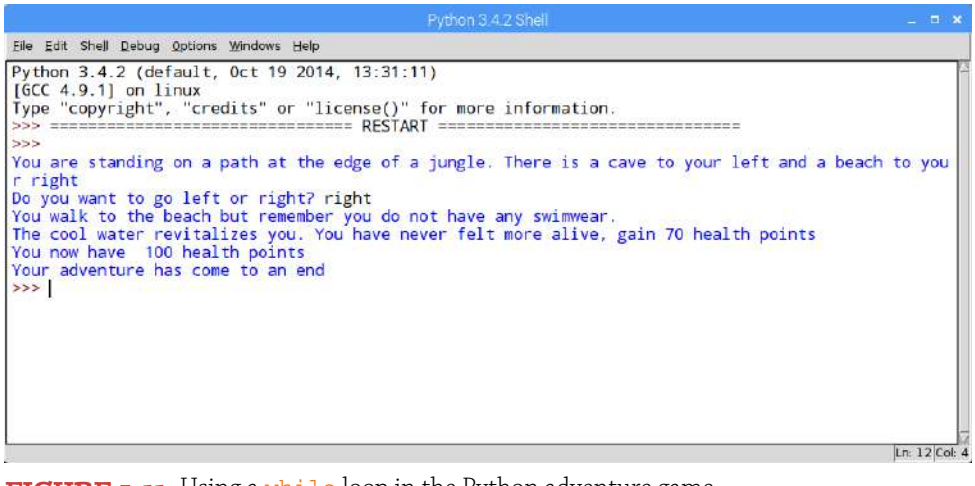


FIGURE 5-11 Using a `while` loop in the Python adventure game

Indentation refers to how far from the margin a line of code is typed. Indentation of code in Python is very important, especially when you begin to add more structure to your code by using conditionals and loops. In Adventures 3 and 4, you used conditionals and loops in Scratch. When you added a `forever` loop, you placed individual blocks inside it. These blocks inside the loop were indented from the rest of the statement. It is similar in Python. After you type `while True`: the next line should be indented, otherwise your code may not work. The same is true of the conditionals `if`, `else` and `elif`.

Using a Variable for Health Points

In the text adventure game so far, you have created a variable for health points (`hp = 30`), like the variable you created in the Scratch adventure game in Adventure 3. Here you have given an initial value that changes as the player plays the game. The value you have given is 30, but this could be any value of your choosing.

Now code can be added that changes the `hp` value based on the decisions made by the player. You name a variable using the form `name = value`, as in the following example:

```
hp = 30
```

Here are two ways to change the value of the `hp` variable in Python 3:

To subtract 10, use `hp = hp - 10` or `hp -= 10`.

To add 10, use `hp = hp + 10` or `hp += 10`.

You can use the following symbols to program calculations:

- `-` Subtract
- `+` Add
- `*` Multiply
- `/` Divide
- `<` Less than
- `>` Greater than

The following symbols produce a value that is True or False, so they are useful in conditionals:

- `==` Equals
- `!=` Not equals
- `<` Less than
- `<=` Less than or equal to
- `>` Greater than
- `>=` Greater than or equal to

To make the game more interesting for the player, you can add some code to the end of what you have already written to tell her how many health points she has after each move:

```
# Check health points after the player has made a move
print("You now have ", hp, "health points")
if hp <= 0:
  print("You are dead. I am sorry.")
```

The last two lines add a conditional so that if the value of the `hp` variable is less than or equal to 0, the statement `You are dead. I am sorry.` is displayed and the game ends.

Putting It All Together

Now put all the elements together in your text adventure game by typing the following program into a new Python 3 IDLE text editor window:

> You can download the completed `AdventureGame1.py` code file from the companion website at `www.wiley.com/go/adventuresinrp3E` but, as I mentioned earlier, you will learn more by typing in the code as you read through the steps.
>
> 
>

Python Text Adventure Game

```
# Python Text Adventure Game

import time

# Create health point variable
hp = 30

# Tell player their location and wait 1 second
print("You are standing on a path at the edge of a jungle. ↩
  There is a cave to your left and a beach to your right.")
time.sleep(1)

# Loop until we get a recognised response
while True:
  direction1 = input("Do you want to go left or right? ")
  # Convert to lower case to accept LEFT and RiGhT etc.
```

continued

Python Text Adventure Game *continued*

```
  direction1 = direction1.lower()
  if direction1 == "left":
    print("You walk to the cave and notice there is an ↩
      opening.")
    print("A small snake bites you, and you lose 20 health ↩
      points.")
    hp = hp - 20
    break # leave the loop
  elif direction1 == "right":
    print("You walk to the beach but remember you do not ↩
      have any swimwear.")
    print("The cool water revitalizes you. You have never ↩
      felt more alive, gain 70 health points.")
    hp += 70
    break # leave the loop
  else:
    print("You think for a while")
    time.sleep(1)

# Check health points after the player has made a move
print("You now have ", hp, "health points")
if hp <= 0:
  print("You are dead. I am sorry.")

print("Your adventure has ended, goodbye.")
```

Test to see whether your code works by saving the file as `AdventureGame1.py` in your `Documents` directory on the Raspberry Pi and then running it (see Figure 5-12). (Be sure to add the `1` in the filename to keep this file separate from your original `AdventureGame.py` file.)

Defining Functions

Although the game works, you will find it difficult to scale it up to include more locations and directions, such as going into the cave and then deeper into the cave, especially if the game relies on input from players to make decisions. You could add more conditionals by copying and pasting those you have already created, but your code will get messy and out of control very quickly. It will also be difficult to locate any bugs, or make changes without introducing even more bugs!

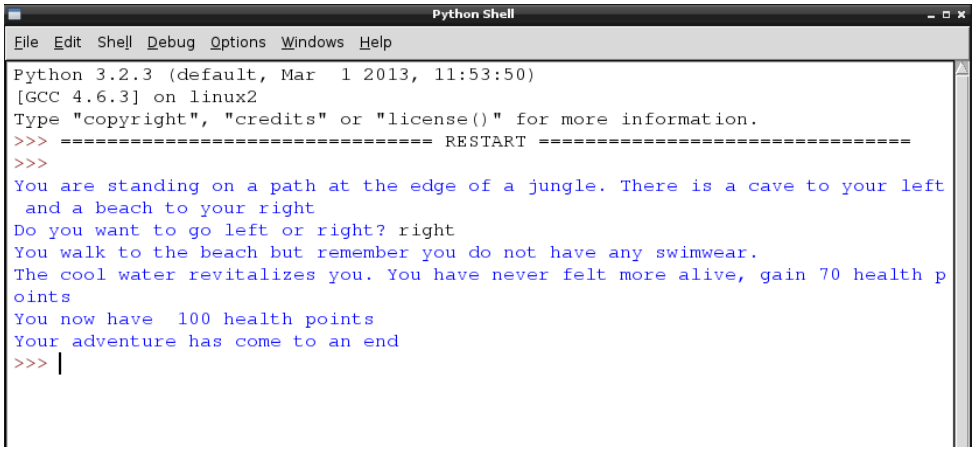


FIGURE 5-12 Using Run Module in Python to play `AdventureGame1.py`

The best solution is to create your own functions. Until now you have been using pre-existing functions from other Python modules such as `time` and `random`, but you can also create your own. Writing your own functions is easy; for example, this is how you would write a function called multiply that multiplies its two arguments and returns the result:

```
def multiply(m, n):
  return m * n
```

Just like the functions you use earlier in this adventure, these can take a number of arguments and return a result. They are a very useful way of organising your code. You can use various ways to reorganise your code into different functions. This process is called **refactoring** and is a logical process when developing computer programs. One way of reorganising the adventure game you have created so far is to create and use two functions: `get_input()` and `handle_room()`.These functions are described in the following sections.

Refactoring is a way of restructuring code you have already written to make it more efficient and easy to read, and to avoid bugs. If you find yourself copying and pasting large sections of code, this is usually a good indicator that you need to refactor your code!

The get_input Function

The `get_input()` function keeps asking the player to enter input (using the text in the prompt argument) until it matches one of the accepted inputs. For example:

```
get_input("Do you want to go left or right? ", ["left", ↩
  "right"])
```

This function keeps asking the player the same question until she types one of the accepted inputs, which in this case are **left** and **right**.

The handle_room Function

The `handle_room()` function contains the main logic for changing locations in the adventure game. The function takes the current location as its argument, and then uses conditionals to decide what to do based on that location. For most locations, the function asks the player to input a direction. The specific input determines which location the player moves to next.

Creating a Main Game Loop

Up to this point, you wrote all your game logic in the `while` loop. With the following code, you move most of the logic into separate functions, avoiding repetitive code. The new loop calls the `handle_room()` function to perform a task appropriate for the current room, and then updates the `location` variable with the new room. This is a little more advanced than the code you have written so far. It requires you to check and double-check that your indentation is correct and you have not made any syntax errors!

Open a new text editor window and save the file as `AdventureGame2.py` in your `Documents` directory. Follow the steps below to add functions to your adventure game.

You can download the completed `AdventureGame2.py` code file from the companion website at `www.wiley.com/go/adventuresinrp3E` but, as I mentioned earlier, you will learn more by typing in the code as you read through the steps.

1. Begin by creating the health point variable as before; it is important to do this at the start as it is a global variable, meaning it can be accessed by the functions you define:

```
# Create health points variable
hp = 30
```

2. Define the first of the functions, `get_input()`. The word `def` introduces a function definition. This function asks for input from the player with the given prompt, and as it contains a `while` loop it keeps retrying until the player types one of the words in the accepted list, **left** or **right**. The `in` keyword enables you to check easily whether a value is in a list or not.

```
def get_input(prompt, accepted):
  while True:
    value = input(prompt).lower()

    if value in accepted:
      return value
    else:
      print("That is not a recognised answer,↩
        must be one of ", accepted)
```

3. Define the function `handle_room()`, which takes the current location, performs an action based on that location, and then returns the new location. For example, if the location is `start`, the game asks which direction the player wants to go and uses that answer to move the user to a new room.

Remember to indent your code correctly here, as three levels of indentation are being used.


```
def handle_room(location):
  global hp

  if location == "start":

    print("You are standing on a path at the edge of a ↩
      jungle. There is a cave to your left and a beach to ↩
      your right.")
    direction = get_input("Do you want to go left or ↩
      right? ",
      ["left", "right"])

    if direction == "left":
      return "cave"
    elif direction == "right":
      return "beach"
```

```
  elif location == "cave":
    print("You walk to the cave and notice there is an ↩
      opening.")
    print("A small snake bites you, and you lose 20 ↩
      health points.")
    hp = hp - 20

    answer = get_input("Do you want to go deeper?", ↩
      ["yes", "no"])
    if answer == "yes":
      return "deep_cave"
    else:
      return "start"

  elif location == "beach":
    print("You walk to the beach but remember you do ↩
      not have any swimwear.")
    print("The cool water revitalizes you. You have never ↩
      felt more alive, gain 70 health points.")
    hp += 70
    return "end"

  else:
    print("Programmer error, room ", location, " is ↩
      unknown")
    return "end"
```

4. Add a loop to the game that loops until the player reaches the special `end` location that ends the game.

```
location = "start"
# Loop until we reach the special "end" location
while location != "end":
  location = handle_room(location) # update location
```

5. As the game also relies on the player having health points, at each turn the program needs to check how many health points the player has and determine that the player is not dead, because this would end the game.

```
  # Check we are not dead each turn
  print("You now have ", hp, "health points.")
  if hp <= 0:
    print("You are dead. I am sorry.")
    break

print("Your adventure has ended, goodbye.")
```

Each time around, the loop checks that the health points are greater than or equal to zero (checking that the player is not dead!) so that the game can continue. The loop also ends if the location returned by `handle_room` is `end`, a special room name indicating the end of the game. Figure 5-13 shows the refactored code using functions in the Python adventure game.

CHALLENGE

Compare the code of `AdventureGame2.py` to `AdventureGame1.py` and note the changes. Do you think refactoring the code was a good idea?

How can you add more locations to the code you have created so far for the adventure game?

How many different times must you run the program to try all possible paths through the code to make sure that it works?

Can you add the inventory list to the text adventure game so that the player can make decisions about using objects?

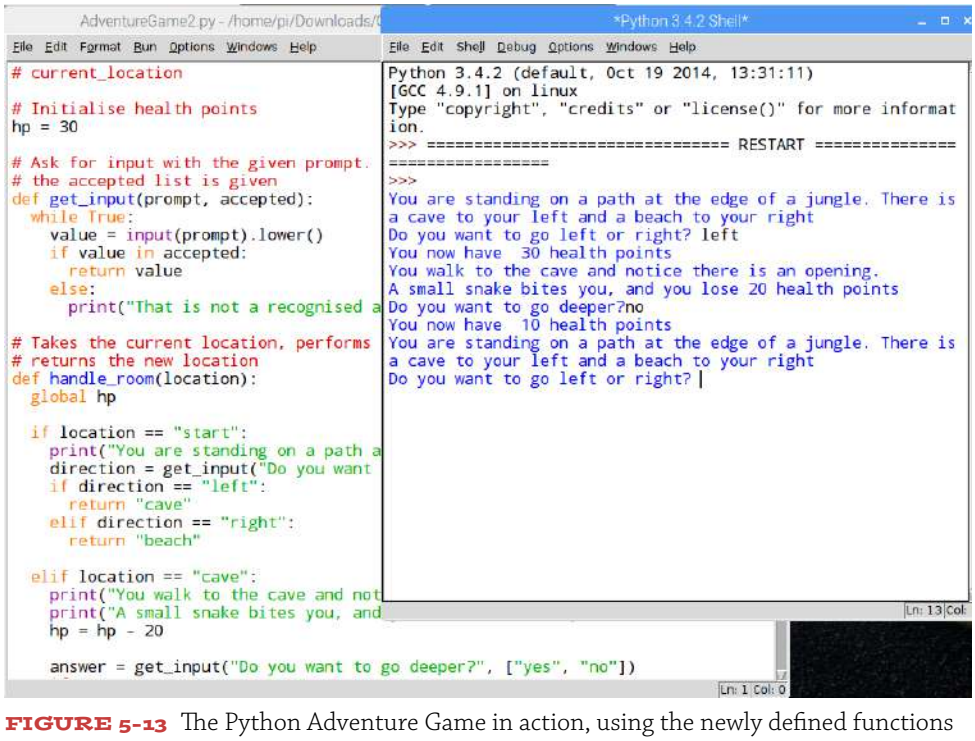


FIGURE 5-13 The Python Adventure Game in action, using the newly defined functions

Continuing Your Python Adventure

If you want to learn more about programming in Python on your Raspberry Pi, you can find a wide assortment of resources. Here are a few to try:

- For more detailed information on Python basics, I recommend *Python Basics* by Chris Roffey (Cambridge University Press, 2012).
- The official Python documentation is available at `http://docs.python.org/3`.
- Visit `http://inventwithpython.com` for links to online PDFs that teach you how to invent your own computer game with Python.

Python Command Quick Reference Table

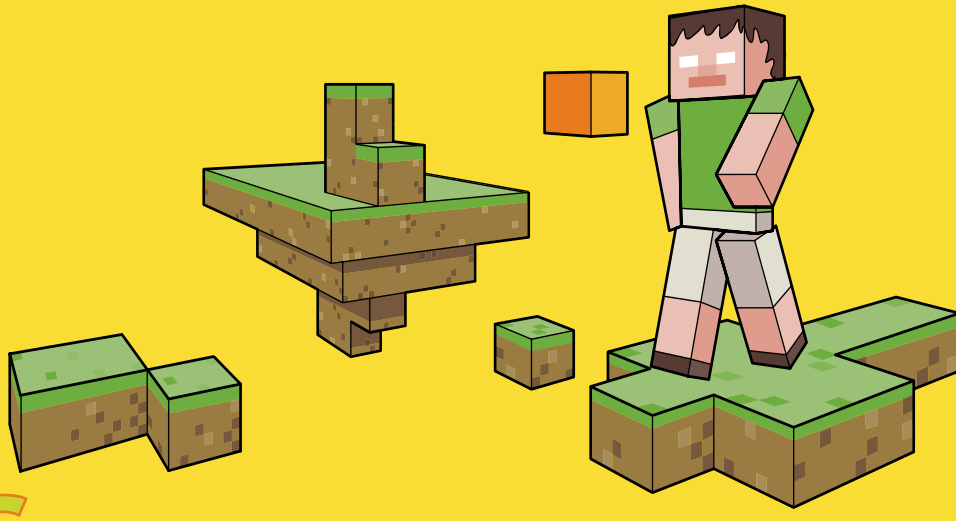
Command	Description
`#`	The `#` symbol is used at the beginning of a code line to indicate the line is a comment, not part of the program's instructions to the computer.
`\n`	Returns a new line in a string.
`break`	Breaks out of a `for` or `while` loop.
`def`	Allows you to define a function of your creation.
`elif`	Short for 'else if', the `elif` syntax allows you to create multiple conditions that make something happen when they return a value of `true`.
`for`	`for` loops are traditionally used when you have a piece of code which that you want to repeat x number of times.
`if`	Sets a condition which, if true, makes something happen.
`if...else`	Sets a condition which, if true, makes one set of things happen, or if false makes a different set of things happen.
`import`	Imports modules and libraries to add more functionality to your code.
`input()`	A function that asks for user input and converts it into a string.
`inventory = ["Torch", "Pencil", "Rubber Band", "Catapult"]`	An example of a list in Python. Lists can contain values or strings that are separated by commas and encased in square brackets.
`name = value`	An example of a variable.
`print()`	A function that prints anything inside the brackets.

Command	Description
`print(inventory[3])`	An example of using the `print()` function to print item number 3 in the inventory list.
`random`	A Python module that returns a random value.
`return`	The `return` keyword is used when a function is ready to return a value.
`time`	Python module that provides various time-related functions, such as `sleep`.
`while`	A `while` loop continually repeats if a given condition is true.

Achievement Unlocked: You can program in Python on your Raspberry Pi!

In the Next Adventure

The next chapter introduces you to the computer game Minecraft—more specifically, you use the Minecraft Pi version created for use on the Raspberry Pi. This special version of the game allows you to use Python programming code to manipulate the Minecraft world in some way. For example, you learn how to post messages to the chat window, add different blocks to create structures, and teleport across the Minecraft world in which your player is located.



Adventure 6

Programming Minecraft Worlds on the Raspberry Pi

MINECRAFT IS A computer game that allows you to build any computer world you like, by using virtual building blocks (see Figure 6-1). You can let your imagination run riot—there are no limits! The game was created by Markus Persson (who also goes by the gamer tag *Notch*). Players collect (or *mine*) blocks from the world around them, using nothing but a trusty axe, while avoiding monsters who might be set on eliminating them. You can learn more about Minecraft and register to play an online demo version at `https://minecraft.net`.

FIGURE 6-1 Minecraft

Getting Started with Minecraft Pi

Minecraft Pi is pre-installed on the Raspbian operating system. You can run it by using the menu system and navigating to Games and selecting Minecraft Pi, as shown in Figure 6-2. It is similar to the Minecraft Pocket Edition.

1. You can Open Minecraft Pi by clicking the desktop menu and selecting Minecraft Pi from the Games menu.
2. Click Start Game to see a list of Minecraft worlds that you can join. On your first go, however, this list is empty.
3. Click Create New to generate a Minecraft world in build mode. Minecraft has two modes—Survival and Build. In Build mode, you are able to construct objects without having to avoid monsters who may end your life.
4. Play around a bit and familiarize yourself with the controls (see Table 6-1) for playing Minecraft in Creative mode.

Being able to play Minecraft is one fun aspect of Minecraft Pi, but what is more exciting is being able to use Python code to manipulate the Minecraft environment. Time to dive in.

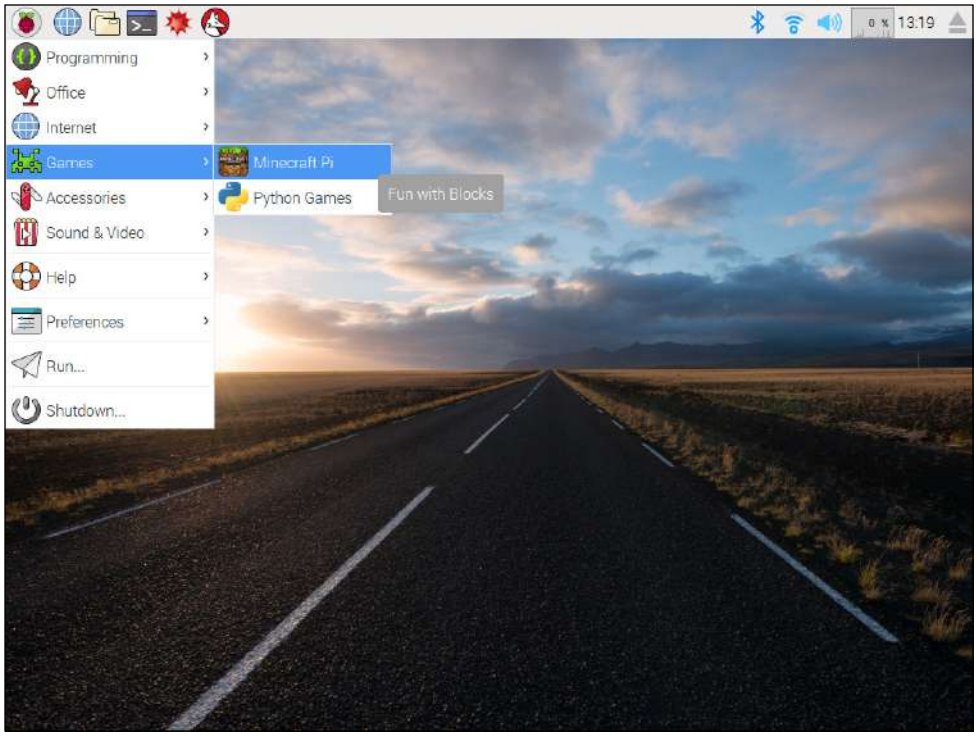


FIGURE 6-2 Opening Minecraft Pi from the Raspbian menu

For a video that walks you through setting up Minecraft Pi, visit the companion website at `www.wiley.com/go/adventuresinrp3E`. Click the Videos tab, and select the `MinecraftPiSetup` file.

Table 6-1 Minecraft Pi Controls

Key press/mouse movement	Action
W	Moves player forward.
A	Moves player left.
S	Moves player back.
D	Moves player right.
Spacebar	Makes player jump. Tap the spacebar twice to make the player fly.
Tab	Releases the mouse so you can click on other windows.
Esc	Returns you to the menu.
Move the mouse	Allows you to see around the player and point the player in a particular direction.
Left- click	Breaks the blocks around you.
Right- click	Places a block.

Minecraft Pi has programming libraries for Python, which you have been using so far in this book and also Java. In this adventure we will be using Python 3.

Your First Minecraft Pi Python Program

Now that you have created a new world and explored, it's time to see what adventures you can have with code in the Minecraft world. In this project, you run a Minecraft game and write a Python program to test that your connection to the game works. You do this by displaying a message in the game.

1. Begin by opening Python 3 (IDLE) and running Minecraft Pi—if it is not already open.
2. When Minecraft Pi has loaded, click Start Game and select a world from the list. (If you have not yet created any worlds, click Create New to enter Creative mode, as instructed earlier.)
3. Navigate back to the Python 3 window with your mouse, and open a new file by selecting File➪New File.
4. Type the following code into the text editor file:

```
from mcpi.minecraft import Minecraft
```

As in previous Python programs you have created, here you are importing a module that you need in your program—in this case, the `minecraft` module. Now type the following command (being sure to use the correct capitalisation):

```
mc = Minecraft.create()
```

This line connects your program to Minecraft and enables you to start interacting with it. Remember, you must have Minecraft running and be in a game for your program to work.

5. Create a message string using the following code:

```
msg = ("I am starting my Minecraft Pi Adventures")
```

Then type the following line to post your message to the Minecraft chat window:

```
mc.postToChat(msg)
```

6. Save your file as `testmcpi.py` in your Documents folder.

7. Run your program by clicking Run and Run Module.

You see your message displayed in the Minecraft game window open on your screen, as shown in Figure 6-3.

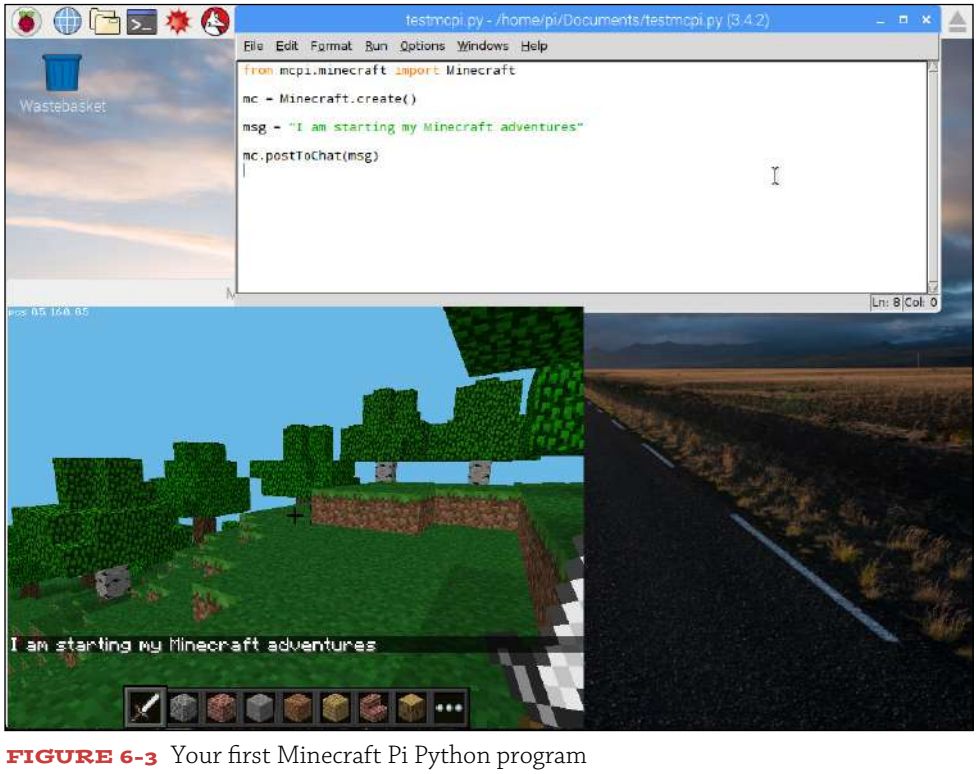


FIGURE 6-3 Your first Minecraft Pi Python program

Using Coordinates in Minecraft Pi

You can see how easy it is to make something interesting happen in the Minecraft game environment using Python code. Minecraft gives the appearance of three dimensions. To achieve this, Minecraft Pi uses x, y and z coordinates to generate a 3D environment, with x representing left to right, y representing up and down and z representing forward and back (see Figure 6-4).

Finding the Player's Location

To understand coordinates within Minecraft, in this part of the adventure you find out the current coordinates of your player and then transport her to a different location by changing the coordinates. Begin by writing a program to locate your player.

1. In Python 3 (IDLE) open a new file and save it as `location.py`.

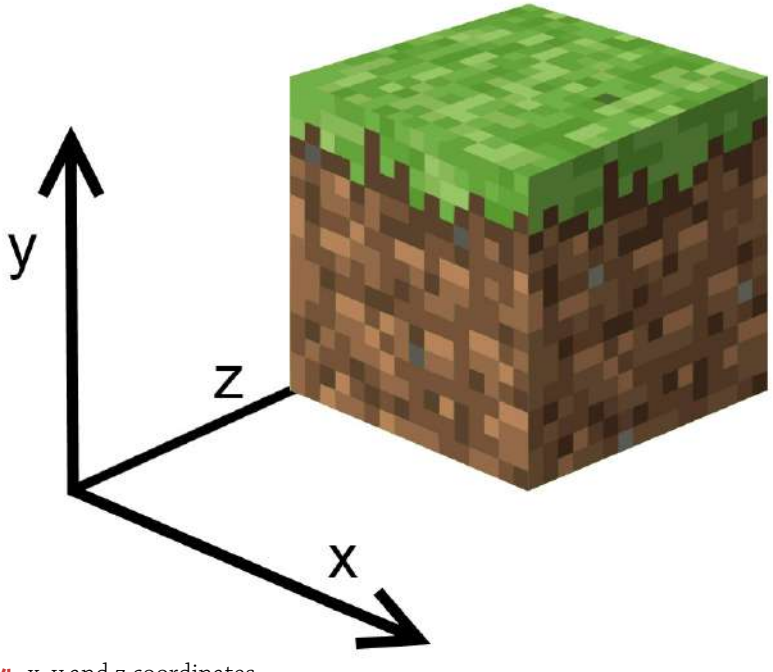


FIGURE 6-4 x, y and z coordinates

2. In the new file, first import the modules you need in this program by typing the following code:

```
from mcpi.minecraft import Minecraft
import time

mc = Minecraft.create()

time.sleep(1)
pos = mc.player.getPos()
```

In the last line, you use the command `getPos` to *get the position* of your Minecraft player. Next, you want to display that information in the Minecraft chat window so you can see it in the game. To do that, type the following code:

```
mc.postToChat("You are located x=" +str(pos.x) + ", y=" ↩
  +str(pos.y) +", z=" +str(pos.z))
```

`pos.x` gives the x coordinate, `pos.y` gives the y coordinate, and `pos.z` gives the z coordinate.

3. Press CTRL+S on the keyboard to save your code.
4. Run the program by pressing F5 on the keyboard.

The coordinates for the player's location are displayed in the chat window (see Figure 6-5).

DIGGING INTO THE CODE

`postToChat` requires a string, and `pos.x` is a number. `str()` converts the number into a string, and Python allows you to add different strings together to make one long string.

Changing the Player's Location

Now that you can easily detect your player's position within Minecraft, why not change her location? Amend `location.py` by adding the following lines of code at the end of your program:

```
time.sleep(2)
mc.postToChat("Get ready to fall from the sky!")

time.sleep(5)
mc.player.setPos(pos.x, pos.y + 60, pos.z)
```

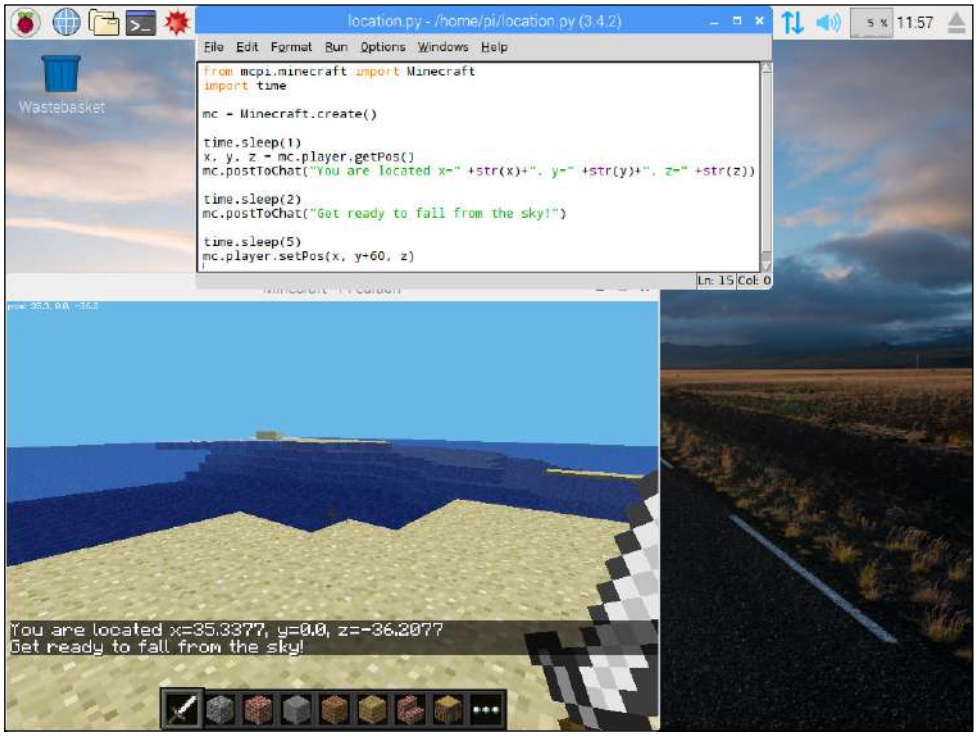


FIGURE 6-5 Using `getPos` and `setPos` to locate and move a player in Minecraft Pi

When you run this code, the player suddenly changes position. The last line of code—`mc.player.setPos(pos.x, pos.y + 60, pos.z)`—adds 60 only to the y axis (up and down). The result is that the player is suddenly transported from her current position to the middle of the sky—and because the player has nothing to stand on, she starts to fall! In the next section, you discover how to give your player something to stand on.

Placing a Block

Traditionally, Minecraft is played by building structures such as shelters, homes and other buildings. In fact, you can build whole cities if you are good at the game and have spent lots of time farming and creating the different levels of material blocks. However, if you use a version of Minecraft that you can manipulate with code, you don't need to spend hours building a structure; you can simply program it to happen.

Before you start this section, it is important that you create a new world in Minecraft Pi by clicking the Create New button after starting the game from the title menu. This places the player at the home position. If you do not do this, you might not be able to see the blocks you create in the next section.

1. Create a new file and save it as `placeblock.py`.
2. Then, type the following code into the text editor window:

```
from mcpi.minecraft import Minecraft
mc = Minecraft.create()
x, y, z = mc.player.getPos()

mc.setBlock(x+1, y, z, 1)
```

   By finding the player's position and storing the values in x, y and z coordinates (refer to Figure 6-4), you can place a block in relation to where your player is in the world. By adding 1 to the player's x coordinate, you are asking for a block to be placed 1 block length away from the player's position on the x axis. This is followed by the block type you wish to use—the value 1 represents `STONE` in this example (see Figure 6-6). The result in Minecraft Pi is a single block of stone appearing next to the player.

3. Press CTRL+S to save your work. Test your code to see what happens by pressing F5.

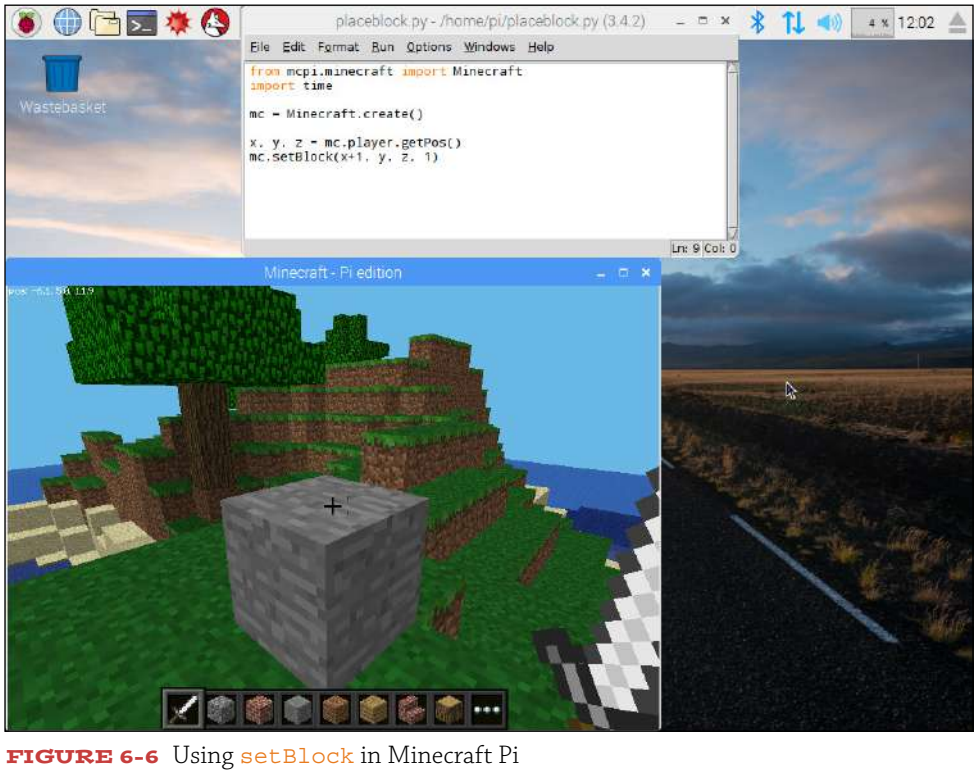


FIGURE 6-6 Using `setBlock` in Minecraft Pi

CHALLENGE

See if you can navigate to a new location and then place some different blocks using the `setBlock` command. You can find the full list of blocks supported in Minecraft Pi in the file `block.py`, which is located in `/opt/minecraft-pi/api/python/mcpi/`.

Placing Multiple Blocks

Placing one block at a time is still going to be time-consuming and not very helpful if you want to build a bigger structure. But with the addition of one letter, you can place more than one block at a time.

So far, with `setBlock` you have been using one set of coordinates to tell the game where to place a single block. The `setBlocks` command (note the added `s` at the end) works in much the same way as `setBlock`. You start with a set of x, y and z coordinates that indicate where in the Minecraft world you want the blocks to be placed. Then the second set of x, y and z coordinates represent the number of blocks needed to make the shape you want to create:

```
setBlocks(x1, y1, z1, x2, y2, z2, blocktype)
```

For example, to generate a cube you would type:

```
setBlocks(0, 0, 0, 10, 10, 10, 103)
```

The first three numbers are the location where the blocks should be placed. The last three are the number of blocks. So the preceding code would place 10 melon type blocks (block ID 103) on the x axis, 10 on the y axis and 10 on the z axis, forming a cube (see Figure 6-7).

However, if you try to use this script you may not be able to find your cube of melons, as Minecraft Pi sets the blocks from the center location (`0, 0, 0`) of the world. Depending on the terrain, this could be inside a mountain!

It makes more sense to place the blocks where the player is located, so that you can find a nice empty spot for your cube. To do this, you need to get the player's position using the following code:

```
x, y, z = mc.player.getTilePos()
```

DIGGING INTO THE CODE

Until now, you have used the position of the player as a starting point for interacting blocks; this allows you to find out what block the player is standing on or place blocks around the player. However, this poses a problem if you want to place multiple blocks. The x, y and z coordinates returned by the `getPos()` function are decimals (also known as *floats*), as your player can be in the middle of a block in Minecraft—but to interact with blocks you need to use whole numbers (aka *integers*). So instead, use the function `getTilePos()`, which returns the block (or tile) that the player is standing on.

CHALLENGE

Try changing the values for the block coordinates and see what cubes and cuboids (rectangular cubes) you can make. See Figure 6-7 if you need some guidance.

How might you use `setBlocks` to build a wall?

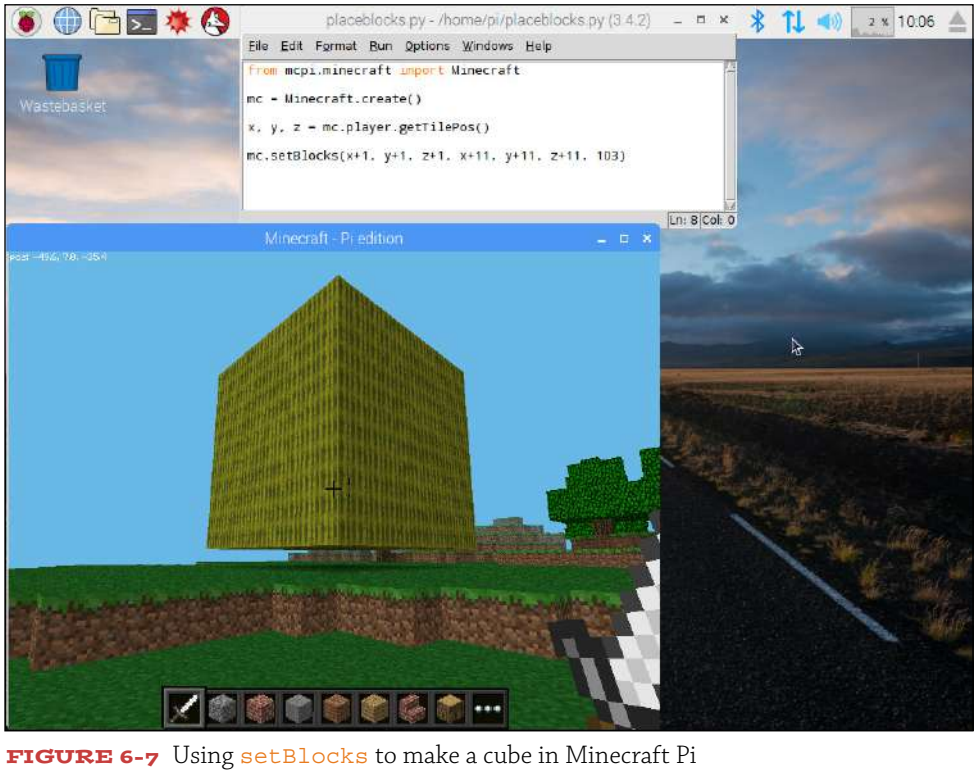


FIGURE 6-7 Using `setBlocks` to make a cube in Minecraft Pi

You can then use `setBlocks`, adding the coordinates of the player's location to indicate where you would like the blocks to be placed:

```
mc.setBlocks(x, y, z, x+10, y+10, z+10, 103)
```

You may find that you end up inside this cube because the first set of coordinates is the player's position! You could change the location coordinates in the same way:

```
mc.setBlocks(x+1, y+1, z+1, x+11, y+11, z+11, 103)
```

Try this yourself by modifying your existing code, and see if you can get it to work.

Types of Blocks

There are many different types of blocks in Minecraft Pi that you can use to code structures and objects. So far, you have used `STONE` (block ID 1) and `MELON` (block ID 103). You can find a list of the different blocks and their IDs online, but to get you started here are a few of my favourites:

```
AIR = 0
WATER = 8
LAVA = 10
GOLD_ORE = 14
GLASS = 20
WOOL = 35
FLOWER_CYAN = 38
TNT = 46
DIAMOND_ORE = 56
DIAMOND_BLOCK = 57
GLOWING_OBSIDIAN = 246
```

Rather than using the numbers in your code, you can use variables like we do in the Scratch adventure game in Adventure 3. In this case you are storing the block ID values to make it easier to remember the block type. To create a variable in Python, type the name of the variable you would like to use. Then place the value of the block after an equal sign to assign it:

```
lava = 10
mc.setBlock(x, y, z, lava)
```

There are some blocks that have extra properties. For example, WOOL has an extra parameter for specifying the colour of wool you want to use. To change the colour of wool, you can type

```
wool = 35
mc.setBlock(x, y, z, wool, 4)
```

The value 4 used after the block ID sets the colour of the WOOL block to yellow. Try some others numbers to see what colours you can make the block.

Creating a TNT Chain Reaction

If there is one block that can bring a smile to your face, it is the TNT block. This block has the exciting extra property of being able to explode! You can set the block by using the following code:

```
tnt = 46
mc.setBlock(x, y, z, tnt)
```

Just like the other blocks you have used before, like STONE and MELON, this code places a block that doesn't do very much. You need to tell Minecraft Pi that you want it to be able to explode when your player hits it with his sword. Like WOOL, TNT has an extra value to make it exploding TNT, and it is the value 1.

```
tnt = 46
mc.setBlock(x, y, z, tnt, 1)
```

Run your code and use your player to hit the block with your sword. The block begins to flash; a few seconds later it explodes. To create a chain reaction you need to write a program that drops blocks as the player walks or flies around the world.

1. Open a new Python 3 (IDLE 3) file and save it as droptnt.py
2. Import the modules that you need:

```
from mcpi.minecraft import Minecraft
import time
```

3. Set up the connection to Minecraft, and create a tnt variable:

```
mc = Minecraft.create()
tnt = 46
```

4. Create a continuous loop known as a `while True:` loop to continuously get the player's position and store the values in x, y and z coordinates:

```
while True:
    x, y, z = mc.player.getTilePos()
```

5. Staying inside the loop, set the `tnt` block with the extra exploding value:

```
    mc.setBlock(x, y, z, tnt, 1)
```

6. Add a short period of time so that the `tnt` blocks drop at regular intervals:

```
    time.sleep(0.1)
```

7. Save your program and test that it works by running it and then having your player run around the Minecraft world. You should see `tnt` blocks drop behind you. To start the chain reaction explosion, hit the `tnt` block closest to you with your sword. Figure 6-8 shows the code and the dropped `tnt` blocks.

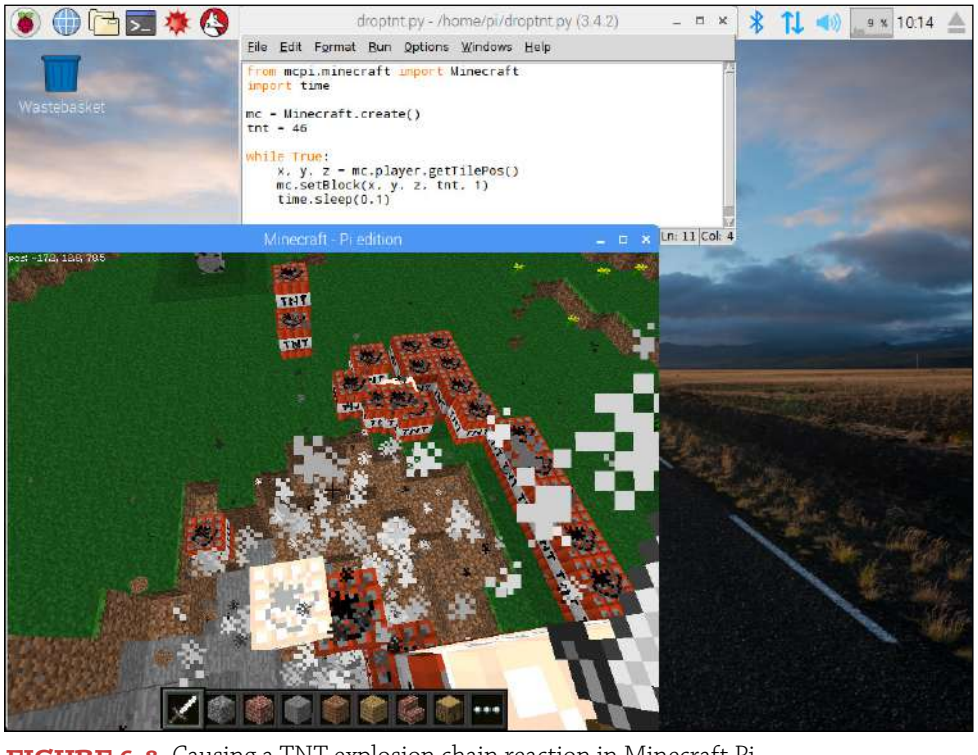


FIGURE 6-8 Causing a TNT explosion chain reaction in Minecraft Pi

Creating a Diamond Transporter

When you are playing Minecraft in Creative mode, it takes a lot of time to move from one side of the world to the other. You can speed things up by creating a diamond block transporter to whisk you from location to location. To do this, you will use `setBlock`, `getPos` and `setPos`.

To see a tutorial for diamond transporter program, visit the companion website at www.wiley.com/go/adventuresinrp3E. Click the Videos tab and select the DiamondTransporter file.

1. Open a new Python 3 (IDLE 3) file and save it as `transporter.py`.
2. Import the modules that you need:

```
from mcpi.minecraft import minecraft
import time
```

3. Set up the connection to Minecraft, create a variable to store the diamond block ID and post a message to the screen:

```
mc = Minecraft.create()
diamond = 57
mc.postToChat("A Transporter Adventure")
time.sleep(5)
```

   The time delays are important in this code, so that you can move the player around the Minecraft world before the transporter blocks are placed.

4. Place a diamond block as the first transporter location, underneath the player's position:

```
ts1 = mc.player.getTilePos()
mc.setBlock(ts1.x, ts1.y - 1, ts1.z, diamond)
mc.postToChat("First transport block created")
time.sleep(2)
```

5. Display a message on the screen to tell the player to move, and find the second location where he wants to be able to transport to and from:

```
mc.postToChat("Find another location in 30 seconds")
time.sleep(30)
ts2 = mc.player.getTilePos()
mc.setBlock(ts2.x, ts2.y -1, ts2.z, diamond)
mc.postToChat("Second transport block created")
time.sleep(2)
```

6. Create a `while True:` loop to continuously check the player's position. If the player is positioned on the first transporter diamond block location, his location is changed to where the second diamond block is located. If the player is positioned on the second transporter diamond block location, his location is changed to where the first diamond block is located.

```
while True:
  x, y, z = mc.player.getTilePos()
  if(x == ts1.x) and (y == ts1.y) and (z == ts1.z):
    mc.player.setPos(ts2.x, ts2.y, ts2.z)
  if(x == ts2.x) and (y == ts2.y) and (z == ts2.z):
    mc.player.setPos(ts1.x, ts1.y, ts1.z)
  time.sleep(0.5)
```

Figure 6-9 shows the code, and Figure 6-10 shows the diamond transporter in position.

CHALLENGE

Can you improve the transporter program in the following ways?

- Modify the code so that a player can set the second transporter location when and where she wants it. Instead of using a timer, you could use input!
- Modify the code so that it can post to chat with a countdown number every five seconds, so that the player knows how long she has left to move.

```
transporter.py - /home/pi/transporter.py (3.4.2)
File  Edit  Format  Run  Options  Windows  Help
from mcpi.minecraft import Minecraft
import time

mc = Minecraft.create()
diamond = 57

mc.postToChat("A Transporter Adventure")
time.sleep(5)

# Place a diamond block to set first transporter location
ts1 = mc.player.getTilePos()
mc.setBlock(ts1.x, ts1.y-1, ts1.z, diamond)
mc.postToChat("First transport block created")
time.sleep(2)

# Display message to player to find a new location
mc.postToChat("Find another location in 30 seconds")
time.sleep(30)

# Place a second diamond block in a different location
ts2 = mc.player.getTilePos()
mc.setBlock(ts2.x, ts2.y-1, ts2.z, diamond)
mc.postToChat("Second transport block created")
time.sleep(2)

# Loop forever so player transports between diamond blocks
while True:
    x, y, z = mc.player.getTilePos()

    if(x == ts1.x) and (y == ts1.y) and (z == ts1.z):
        mc.player.setPos(ts2.x, ts2.y, ts2.z)

    if(x == ts2.x) and (y == ts2.y) and (z == ts2.z):
        mc.player.setPos(ts1.x, ts1.y, ts1.z)

    time.sleep(0.5)
                                                    Ln: 36 Col: 20
```

FIGURE 6-9 Code for the diamond transporter program for Minecraft Pi

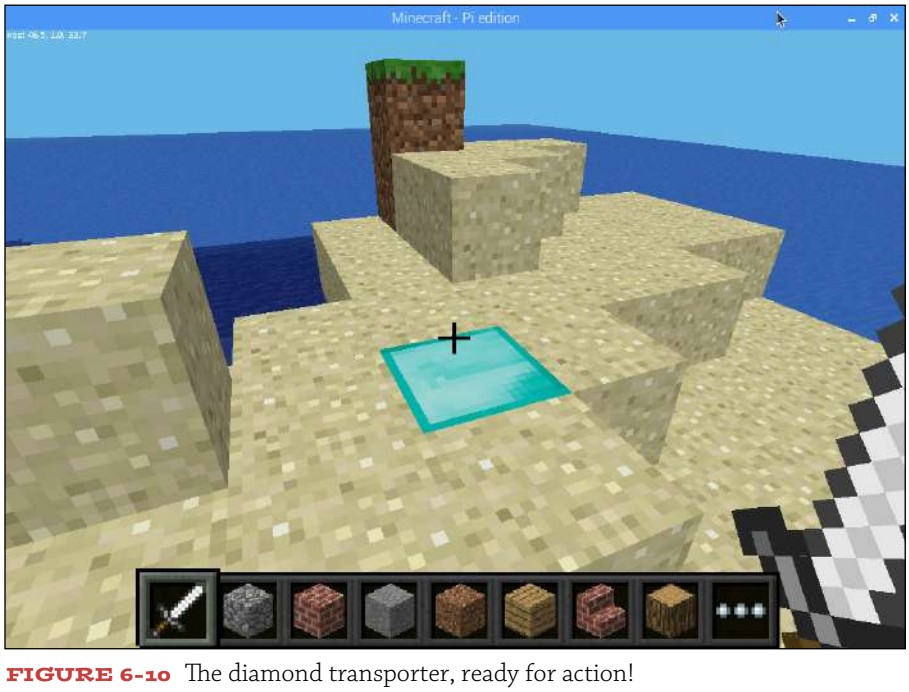


FIGURE 6-10 The diamond transporter, ready for action!

Sharing and Cloning Minecraft Pi Programs

When the Minecraft community meets the Raspberry Pi community, good things happen. Many people enjoy sharing the programs they have made for Minecraft Pi so that you can clone them or make copies of them to use on your own Raspberry Pi. You can find shared Minecraft Pi programs on the Minecraft Pi forum (`www.minecraftforum.net/forum/216-minecraft-pi-edition/`) or the Raspberry Pi forum (`www.raspberrypi.org/forums`). Many programmers who share their code use an online repository such as GitHub (`https://github.com`) so that you can easily download their code, try it out, and help improve it. Why not have a go at downloading Martin O'Hanlon's Minecraft Cannon program? This program places a cannon where your player is positioned in Minecraft Pi. You use the Terminal command line to move the cannon up or down before firing blocks. Check out the video at `www.youtube.com/watch?v=6NHorP5VuYQ` to see it in action, and use the following steps to download and use it yourself:

1. Open a Terminal window. If your Raspbian image is not up to date you may not be able to install the required applications, so first update your application packages by typing the following command into the terminal:

```
sudo apt-get update
```

2. Type the following line:

```
sudo apt-get install git-core
```

   This command installs an application called `git-core` that lets you clone the code that Martin has placed on a repository called *github*.

3. After `git-core` is installed, type the following code to have `git-core` create a clone of the Cannon program on your Raspberry Pi:

```
cd ~
git clone ↩
  https://github.com/martinohanlon/minecraft-cannon.git

cd minecraft-cannon
```

4. After you have navigated to the `minecraft-cannon` directory, type the following command to start the Minecraft cannon program:

```
python minecraft-cannon.py
```

Now you're ready to play with the cannon! To control the cannon, shown in Figure 6-11, you can use the following commands in Terminal window:

- `start`—Start up the cannon
- `rotate` [0-360 degrees]—Rotate the cannon between 0 and 360 degrees

- `tilt` [0-90 degrees]—Tilt the cannon upwards between 0 and 90 degrees
- `fire`—Fire the cannon
- `exit`—Exit and clear the cannon

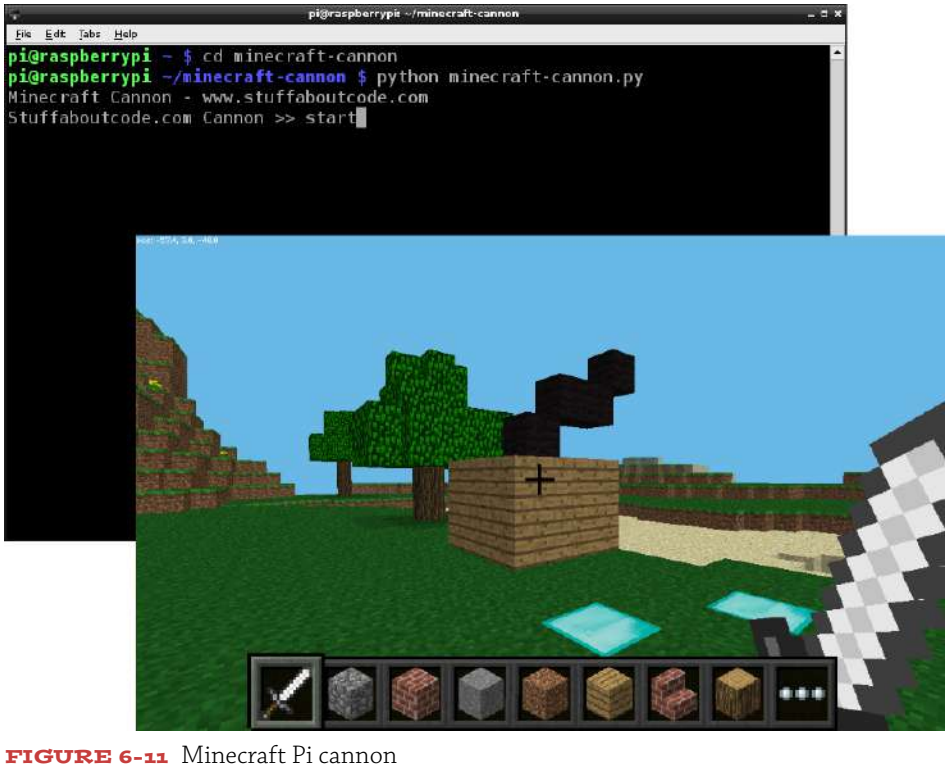


FIGURE 6-11 Minecraft Pi cannon

Further Adventures with Minecraft Pi

Minecraft Pi allows you to be really creative. As well as finding programs created by other people, you can find online tutorials to generate things like rainbows, bridges and other games like Snake inside your Minecraft world. Here's a list to get you started:

- Make a colourful rainbow with the tutorial at `www.minecraftforum.net/topic/1638036-my-first-script-for-minecraft-pi-api-a-rainbow/`.
- You can't go wrong following Martin O'Hanlon's Minecraft Pi tutorials on his website, *Stuff about="code"*: `www.stuffaboutcode.com/p/minecraft.html`.

- Advance your programming skills using Craig Richardson's Python Minecraft Pi Book (http://arghbox.files.wordpress.com/2013/06/minecraftbook.pdf) and API cheat sheet (http://arghbox.files.wordpress.com/2013/06/table.pdf).
- Further develop your Python skills with *Adventures in Minecraft* by David Whale and Martin O'Hanlon (www.wiley.com/go/adventuresinminecraft).

Minecraft Pi Command Quick Reference Table

Command	Description
`from mcpi.minecraft import minecraft`	Imports the Minecraft modules.
`mc = minecraft.create()`	Connects to Minecraft Pi by creating the Minecraft object.
`pos = mc.player.getPos()`	Returns the players position with floats.
`pos = mc.player.getTilePos()`	Returns the players position with integers.
`postToChat(msg)`	Posts a message to chat in Minecraft Pi.
`setBlock`	Sets a block at coordinates.
`setBlocks`	Sets blocks between two sets of coordinates.
`setPos`	Sets the position of a player.

Achievement Unlocked: Why dig when you can code with Minecraft Pi?

In the Next Adventure

In the next adventure, you transform your Raspberry Pi into an electronic synthesizer as you learn how to program music using an application called Sonic Pi!

Adventure 7
Coding Music with Sonic Pi

THE RASPBERRY PI can be many things—a standalone computer, a games machine and even a music synthesizer. The way we as humans interact with computers has changed over the years. Computers are no longer just devices on which to create text files or play computer games. They are also communication devices, transporters and musical instruments!

Creating music using computers is not a new idea. Computer music has its roots in electronics, and a growing number of musicians are turning to code to create new sounds. Chiptune is a style of music that uses sound chips from old computers and consoles from the 1980s and 1990s, like the Nintendo Game Boy. Pixelh8 (`https://en.wikipedia.org/wiki/Pixelh8`) and 2xAA (`http://brkbrkbrk.com`) are chiptune artists and computer programmers who program their music before it is performed. Other computer music programmers prefer to code their music live, feeling the atmosphere around them and responding to it with sound. They are called *live coders*. Sam Aaron (`http://sam.aaron.name/`) is a computer scientist by day but a live coder by night who performs at events, creating his code live on a big screen so the audience can see it as it happens (see Figure 7-1).

Your Raspberry Pi has a headphone/speaker jack port so that you can listen to sounds. You also have a keyboard and mouse that allow you to type code. In this adventure, you put those features to good use by creating music with code using an application called Sonic Pi (`http://sonic-pi.net`). You will become a computer music programmer!

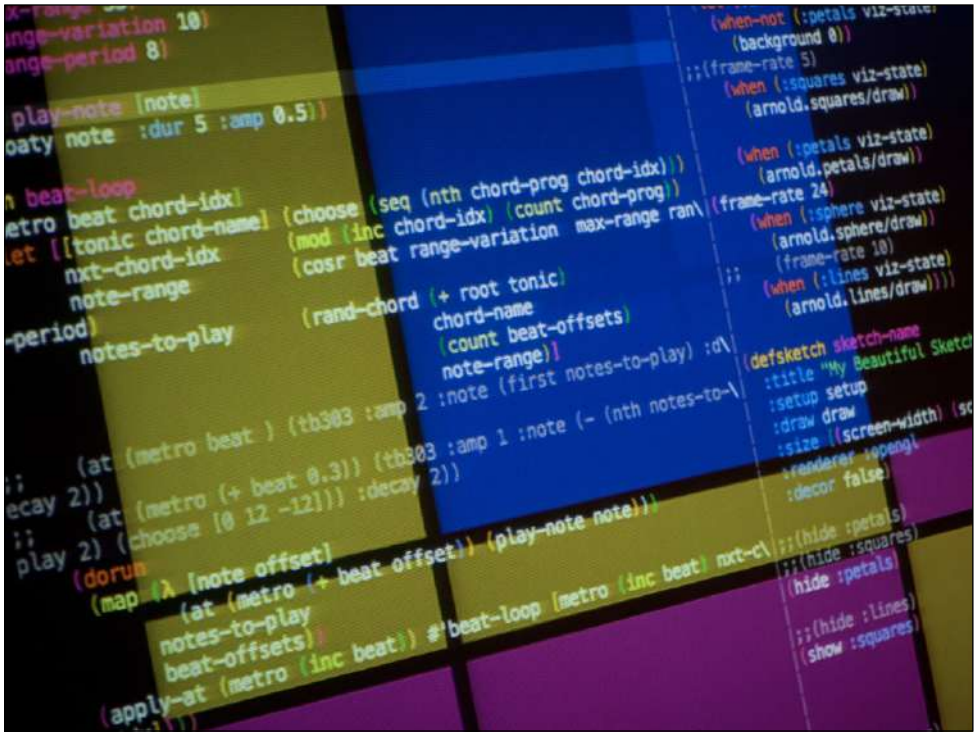


FIGURE 7-1 Dr Sam Aaron's live coding screen during a performance

Getting Started with Sonic Pi

To create music in this adventure you use an application designed to be used on the Raspberry Pi called Sonic Pi. Sonic Pi was created by Dr Sam Aaron, a live coder of music, and is based on his more complex music system called Overtone. Sonic Pi is already installed automatically as part of the operating system, Raspbian. However, improvements are always being made to Sonic Pi, meaning that you may want to update and upgrade to the latest version before you begin the tutorials in this chapter.

Make sure that you are using the most up-to-date Raspbian operating system available to use the latest version of Sonic Pi. Remember, if you ever want to update and upgrade the packages on your operating system, you can type `sudo apt-get update && sudo apt-get upgrade` into a Terminal window. Sam is always improving the application, so if you want to use the latest and greatest version, it's worth following his work at `https://sonic-pi.net`.

You can find Sonic Pi on the main menu under Programming. Figure 7-2 shows the upgrade command being executed.

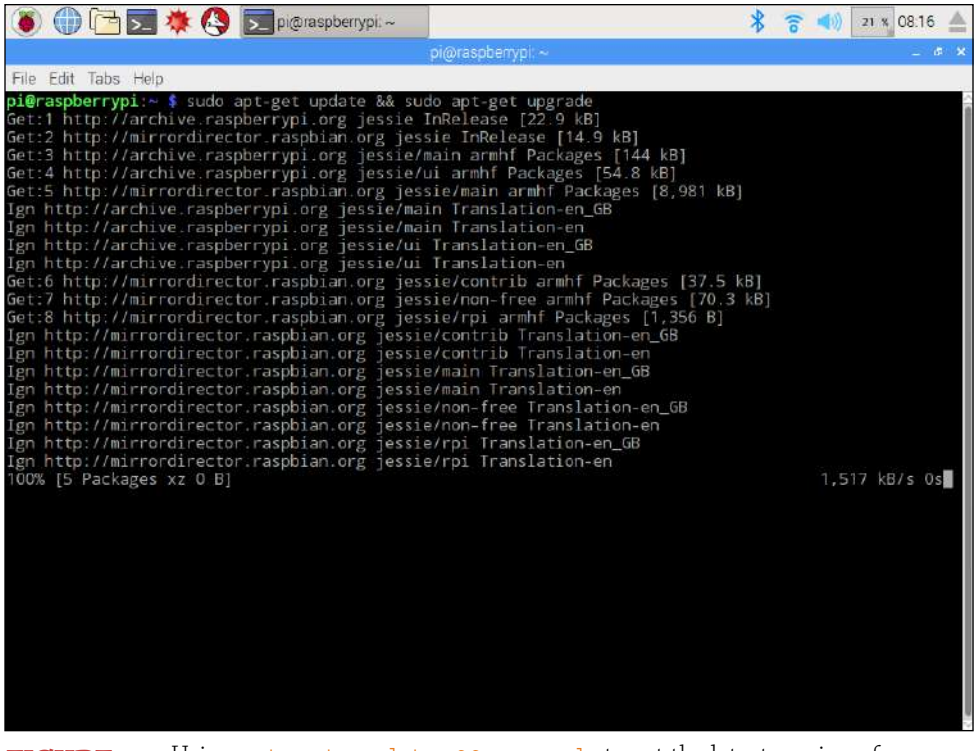


FIGURE 7-2 Using `apt-get update && upgrade` to get the latest version of Sonic Pi

To be able to hear the sounds, you need to connect your Raspberry Pi to a sound device. This can be headphones/earphones or speakers connected to the Pi using the jack port, or an HDMI TV or monitor with built-in speakers, connected with an HDMI cable. The better the quality of sound, the more fun you will have!

The Sonic Pi Interface

Before you start creating music, you'll find it helpful to get to know the Sonic Pi interface and what each panel is used for first. You may need to resize the application window to see the whole interface.

The elements of the Sonic Pi interface are identified in Figure 7-3:

- **The programming panel**—The main panel in Sonic Pi, on the left side. This is where you type your code to make music.
- **The output panel or log**—The upper panel on the right side. This is where you will see information about your program as it runs.

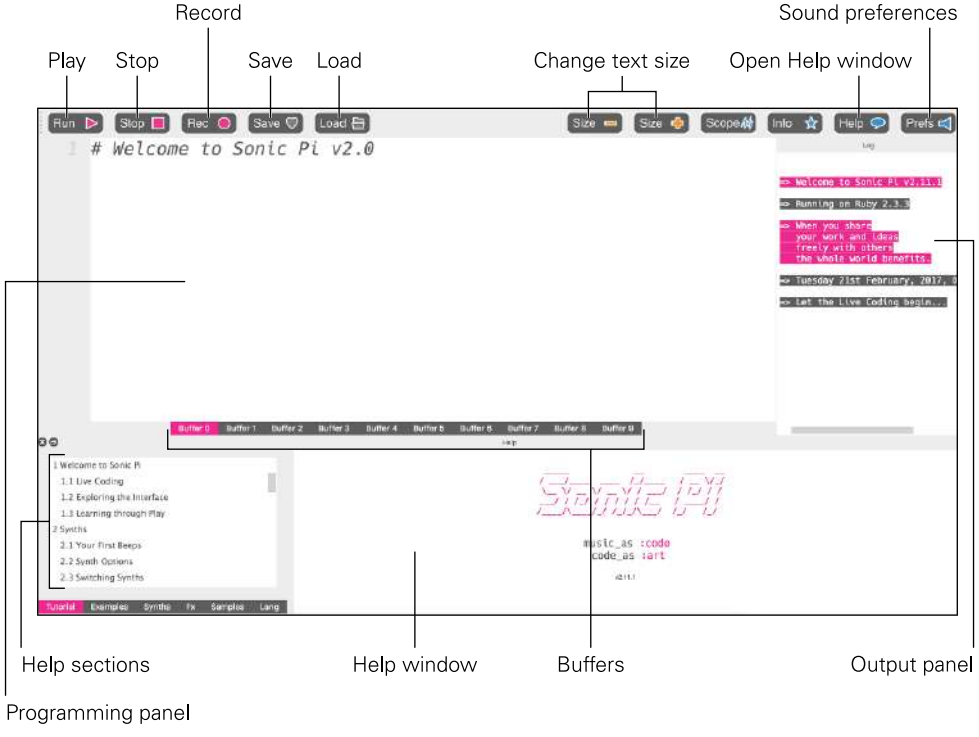


FIGURE 7-3 The Sonic Pi interface

- **Buffers**—You can use different workspaces, called buffers, to create and save your code. In this adventure, you use a different buffer for each exercise. You can move between them using the tabs along the bottom of the programming panel.
- **Play and Stop buttons**—Click these buttons to start and stop your music scripts.
- **Save button—**Sonic Pi automatically saves what you write in the programming panel. However, if you want to save your code into a text file to store it elsewhere, you can use the Save button at the top of the application to do so.

- **Load button**—If you have saved any Sonic Pi code into a text file to store it elsewhere, you can use the Load button at the top of the application to locate it and open it.
- **Record button**—This button allows you to create a sound file recording of your coded music for playback on a media player.
- **Size buttons**—You can change the size of the text to make it bigger or smaller to make it easier to read and spot any errors.
- **Help**—Sonic Pi has built-in information about the code that you can use. It also has some example programs that you can play with.
- **Preferences**—You are able to change the level of volume with this button and force the sound to go to your headphones rather than your monitor, or vice versa.

Creating Your First Sounds with Sonic Pi

Now that you are familiar with the Sonic Pi interface, it's time to start making some noise! In this first project you learn how to play single notes and chords, add timings and play *Twinkle Twinkle Little Star*.

Visit the companion website at www.wiley.com/go/adventuresinrp3E and select FirstSounds to see a video of this project.

1. To play your first note, open Buffer 0 and type the following:

```
play 60
```

Now click the Play button in the top-left side of the application. Not only do you hear your note playing, but you see information displayed in the output panel, as shown in Figure 7-4:

```
synth :beep, {note: 60}
```

If you do not hear a note being played, check the preferences by clicking the icon, to make sure the right output and volume is set.

2. Change your line of code so that it looks like this:

```
pley 60
```

Click the Play button. You don't hear anything, as Sonic Pi found a syntax error in your code because you misspelled `play`. You see an error (as in Figure 7-5). Sonic Pi is letting you know there is an error.

FIGURE 7-4 Creating sounds using Sonic Pi

3. Fix the error by changing `pley` to `play`. Now try playing a few notes one after the other in sequence by typing the following underneath your first note:

```
play 67
play 69
```

Click the Play button. It sounds like the notes are being played at the same time, like a chord. This is no good if you want to play 'Twinkle Twinkle Little Star', as all music is played to a beat. You need to introduce delays between each of the notes in the sequence.

4. Add delays to your code by typing `sleep 0.5` in between each of the play instructions like this:

```
play 60
sleep 0.5
play 67
sleep 0.5
play 69
```

Click the Play button. You hear the notes play with a half-second delay between them.

The numbers used after `play` in Sonic Pi represent notes. Each note is a key on a piano (`http://computermusicresource.com/midikeys.html`). The `play 60` is actually a C, and `play 69` is a G. These numbers are **MIDI** keyboard note numbers.

The numbers used after `sleep` represents timings: `1` is one second, and `0.5` is half a second.

FIGURE 7-5 syntax error in Sonic Pi

DIGGING INTO THE CODE

You may notice that Sonic Pi does not use the same sort of programming language that you have used in previous adventures. It actually uses a different programming language called Ruby. Ruby uses the same computing concepts as Scratch and Python, such as conditionals, iteration and data structures.

A **MIDI** keyboard, or Musical Instrument Digital Interface, is a musical instrument that can communicate with a computer. Piano sheet music notes and MIDI keyboard notes are the same, only sheet music notes are represented by letters G, C, A, and so on, whereas MIDI keyboard notes are represented by numbers. (In fact, MIDI note numbers are in semitone steps. G, A, B are tones whereas 67, 68, 69 are semitones.)

Twinkle Twinkle Little Star

You have the building blocks to generate a simple tune with notes C, G and A—or in this case 60, 67 and 69—along with delays in between those notes using `sleep` (see Figure 7-6).

Amend your code so that it looks like the following, and click Play:

```
play 60
sleep 0.5
play 60
sleep 0.5
play 67
sleep 0.5
play 67
sleep 0.5
play 69
sleep 0.5
play 69
sleep 0.5
play 67
```

Remember that Sonic Pi runs through each line in your code in sequence. You could go on and write the next part of the tune, but you will end up with a long list of `play` and `sleep`, which could get confusing to read, especially if you mistype a line and create a bug.

FIGURE 7-6 Using `play` and `sleep` to play 'Twinkle Twinkle Little Star' in Sonic Pi

It makes more sense to rewrite this code using a **data structure**. In this case you can use a list like the ones you created in Python in Adventures 4 and 5. To create a list in Ruby, you use square brackets and separate the items in the list with commas in much the same way as in Python.

> In computer science, a **data structure** is a particular way of storing and organizing related pieces of information. For example, in a list or array like `play_pattern [60, 67, 69]`, the list enclosed by square brackets in this line of code is an example of a data structure. See Figure 7-7.
>
> 
>

Type the following example list into Buffer 1 and click Play:

```
play_pattern [60, 60, 67, 67, 69, 69, 67]
```

You will notice that the same tune plays, but the delays between the notes are quite slow. To speed up the timing you can set the beats per minute (BPM). At the top of Buffer 1, above the line of code you have just written, type this line:

```
use_bpm 150
```

Click Play and the delay between the notes will decrease, giving the effect of speeding up the tune. The value 150 in this code is the beats per minute (BPM).

FIGURE 7-7 Using a list in Sonic Pi to play notes

Repeating Lines in a Loop

Musical tunes are sometimes made up of repeating notes or phrases. For example, in 'Twinkle Twinkle Little Star', the third and fourth lines, 'up above the world so high' and 'like a diamond in the sky', use the same notes. In your Sonic Pi code you could type these lines out twice, like this:

```
play_pattern [67,67,65,65,64,64,62]
sleep 0.5
play_pattern [67,67,65,65,64,64,62]
```

Or instead you could use a loop:

```
2.times do
  play_pattern [67,67,65,65,64,64,62]
  sleep 0.5
end
```

CHALLENGE

Can you recreate the rest of 'Twinkle Twinkle Little Star' in code by translating the following musical notes into MIDI note numbers? Refer to the table at `http://computermusicresource.com/midikeys.html` to match the notes to the correct numbers.

Each line could be put into a `play_pattern [ ]` data structure:

C C G G A A G (You have already written this part of the song)

F F E E D D C

G G F F E E D

G G F F E E D

C C G G A A G

F F E E D D C

Can you translate another song into MIDI notes and re-create it using Sonic Pi?

For one more challenge, try introducing variables to define the notes. For example:

```
C=60
D=62
play_pattern[C,C,G,G,A,A,D]
```

All the code between do and end is repeated; in this case, `2.times` tells the program to play it twice. You will see that the colour of the words `do` and `end` have automatically changed to gold and are bold, as in Figure 7-8. Programs that use colours for syntax highlighting in this way make it easier for you to read your code. In this example, it is important that the code you want to repeat is between `do` and `end`, so Sonic Pi highlights those words to show you this.

You could change the value `2` to make the loop repeat more times. For example, if you wanted to play the line five times you would type `5.times do`, followed by the code you want repeated, and then `end`. If you want to loop sections of code forever, then you would type `loop do`, followed by the code, and then end.

Your 'Twinkle, Twinkle' tune might not sound the way you expect it to sound. Can you figure out what you may need to add to improve it?

It is good practice to indent your code when you create loops in code. Indenting makes it easier to read, especially if you are searching for a bug to fix to make your music play. All code between `do` and `end` should be indented.

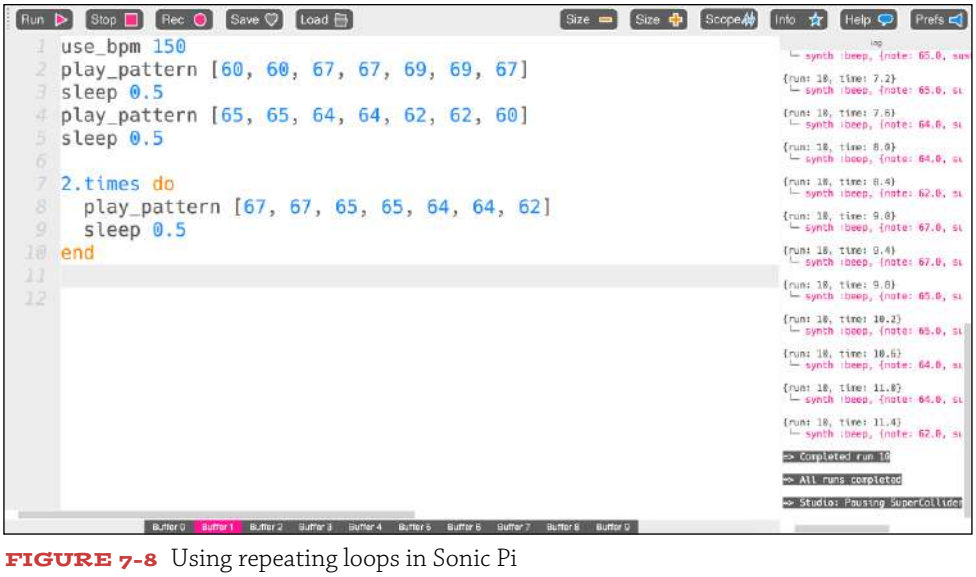


FIGURE 7-8 Using repeating loops in Sonic Pi

First Electronic Track

It's time to take a step up from nursery rhymes and start making some cool-sounding electronic beats using Sonic Pi. In this project you create a complete track in the style of the electronic artists mentioned earlier. Use a new Buffer to try the following exercises and adapt them to create a cool tune.

Visit the companion website at www.wiley.com/go/adventuresinrp3E and select ElectronicMusicTrack for a video of this project.

Using Different Synthesizer Sounds

So far you have used the default Sonic Pi synthesizer sound, called beep. You can change the sound by using the `use_synth` command, followed by the name of the synth (`fm`, in this example) after a colon symbol:

```
use_synth :fm
```

This line of code must be placed above the instructions to play a note, a pattern or a sleep, like this:

```
use_synth :fm
5.times do
  play 49
  sleep 1
end

use_bpm 150
use_synth :beep
2.times do
  play_pattern [67,67,65,65,64,64,62]
  sleep 0.5
end
```

Figure 7-9 shows the use of different synths in Sonic Pi.

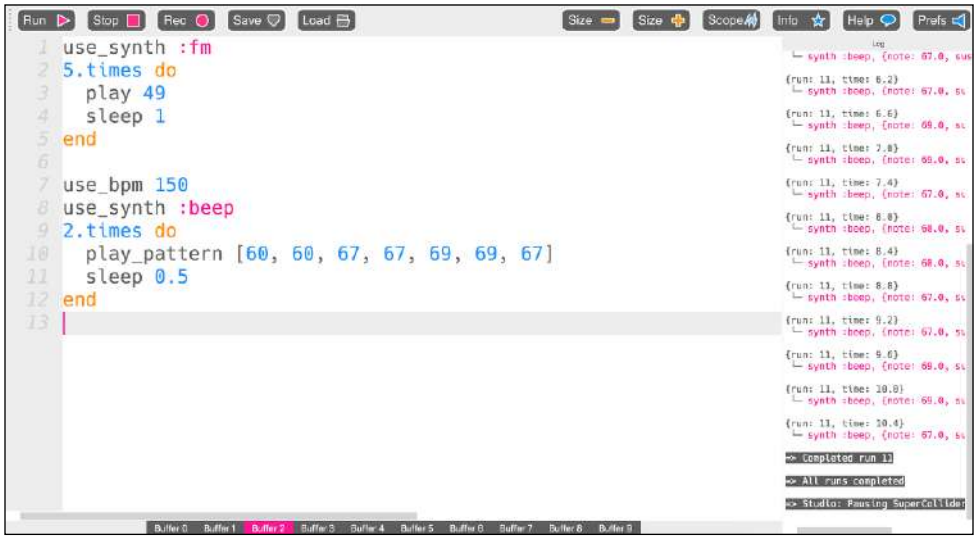


FIGURE 7-9 Using different synthesizer sounds in Sonic Pi

In this example, you would hear the MIDI note 49 play five times using the fm sound, and then the list of notes played twice using the beep sound.

For a list of all the different synths in Sonic Pi that you can try out, click the Help button and then select Synths from the navigation menu.

More sounds will be added to the Sonic Pi application in time, so be sure to watch for them and keep your Pi application packages updated using `sudo apt-get update`.

Using Prerecorded Samples

Not only can you create music in Sonic Pi using single notes, you can also create music with samples. Samples are prerecorded sounds or tunes that you can bring into your code. This is a really simple way to make your music track sound amazing!

To use a sample, you need to add the code `sample :name of sample` in the sequence of your music program where you want it to play.

In this example, `misc_burp` is the name of the sample. Open a new Buffer and type the following:

```
loop do
  sample :misc_burp
  sleep 1
end
```

There are lots of samples included with Sonic Pi to try. To find the names of them, click Help in the top menu, followed by Samples on the bottom of the Help window (see Figure 7-10).

FIGURE 7-10 Finding samples from the Help window

There are lots of fun ways in which you can use samples. For example, you can change how the burp sounds, by adding some additional code, which is highlighted in bold:

```
n = 2
loop do
  n = n - 0.2
  sample :misc_burp, rate: n
  sleep 1
end
```

Now click Play to hear how the burp sounds.

In this example, to begin with a variable has been used to store a value of 2, which is then modified inside the loop each time it is played. This value affects the rate at which the sample is played. The default rate at which samples play is 1. Each time around the loop, 0.2 is subtracted from the sample rate, changing the way it sounds. You can do this with any of the Sonic Pi samples.

Creating a Surprising Tune

So far, you have run your music program in sequence and then using a repeating loop. To add an element of fun, you could add a junction using a conditional. You used conditionals in Adventures 3 and 5 in both Scratch and Python. Setting conditions allows different paths to be followed, as if you were at a junction.

Type the following example script into your current Buffer to try it out (see Figure 7-11):

```
10.times do
  if rand < 0.5
    play 62
  else
    play 50
  end
  sleep 0.25
end
```

The first line is the start of the repeating loop. Everything after `do` and before `end` will be played 10 times. The second line is the start of the conditional statement. The condition used here is like flipping a coin: `rand` stands for random, and it returns a random value between 0 and 1. If the value returned is less than 0.5 then this statement is true and the midi note 62 is played. If the value returned is not less than 0.5, then the statement is false and MIDI note 50 is played instead. Only one of the `play` steps is run. To complete the condition, `end` is used. Each time the loop plays, a new value for `rand` is generated.

What do you think happens if the `rand` returns the value 0.5? Is the statement true or false?

Using "rand" to Play Random Notes

You can use rand in other interesting ways outside of a conditional too. For example, you could use it to play a random note in a sequence.

Underneath the conditional sequence, after the final end line, type the following:

```
3.times do
  play 60 + rand(10)
  sleep 0.5
end
```

FIGURE 7-11 Using conditionals and random in Sonic Pi

The first line is the start of the repeating loop. Everything after `do` and before `end` is played three times. The next line uses a calculation to determine what note it will play. The code play 60 + rand(10) plays a random note between 60 and 69 because you are adding a random number between 0 and 10 to 60 to make a random note each time; for example, 61, 68, 63. Changing the value of play 60 sets the lowest note, whilst changing the value of `rand(10)` changes the range of the highest note.

This makes the music sound more interesting, especially if it is inside a loop, as each time it is played a different note could be heard between the MIDI note numbers you specify.

Using Algorithms

You don't always need to write brand new lines of code to add functionality to your programs. You can use built-in **algorithms** instead, as in this code:

```
play_pattern [60,72,65,80].sort
```

This code is an example of a sorting algorithm you can use in Sonic Pi. When the program is run, the algorithm sorts the numbers in the list into ascending order from lowest to highest.

You can also use `.reverse` to reverse the numbers in a list and `.shuffle` to randomly shuffle the numbers in a list as shown in the following code:

```
use_bpm 150
loop do
  if rand < 0.5
    play_pattern [60,62,65]
  else
    play_pattern [60,62,65].reverse
  end
  sleep 0.125
end
```

In this code, shown in the Sonic Pi interface in Figure 7-12, `.reverse` has been used inside a conditional, so that if the random value returned by `rand` is less than 0.5 then the notes 60, 62 and 65 play in order. If any other value is returned then the notes are played in reverse.

An **algorithm** is a set of rules to be followed to calculate or solve a problem. Common algorithms are those used for sorting information or data, as can be seen by the sorting algorithm animations at `www.sorting-algorithms.com`. Rather than writing a sequence of code to sort the MIDI notes that you used in a list in Sonic Pi, for example, you could use an existing sorting algorithm, `.sort`.

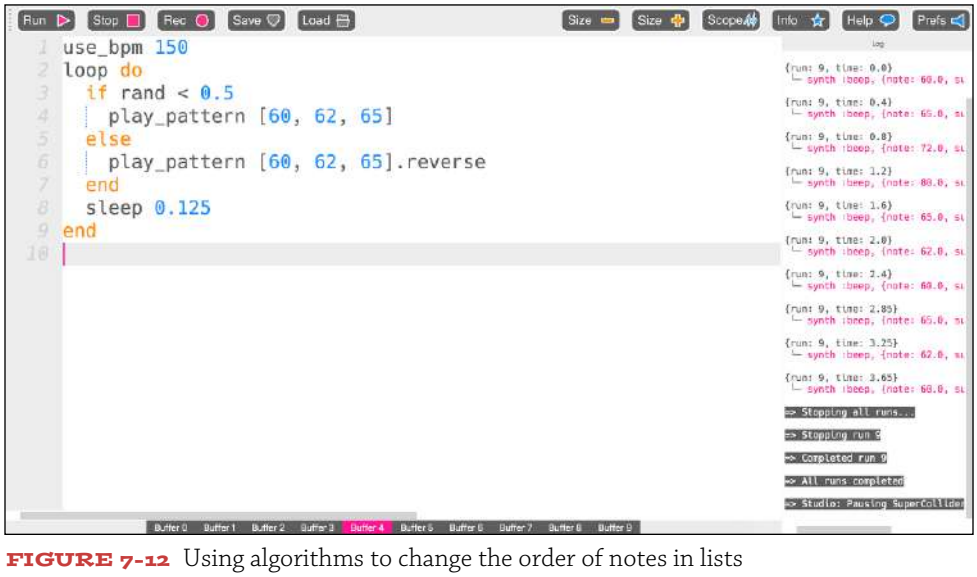


FIGURE 7-12 Using algorithms to change the order of notes in lists

Running Two Scripts at the Same Time

Electronic synthesized music usually has a repeating beat that you can nod your head or dance along to, with a tuneful melody playing at the same time. This is similar to the way pianists typically play with two hands on a piano. One hand plays one set of notes of a song, usually in a lower octave, while the other hand plays a different set of notes.

In Sonic Pi you can use live loops to run more than one script simultaneously, in much the same way as you can do in Scratch. To run multiple tunes at the same time, encase the first tune between `live_loop do` and `end`. Each live loop that you create needs to be given a unique name. For example:

```
live_loop :beat do
  sample :drum_heavy_kick
  sleep 0.5
end
```

Here the name `beat` has been assigned to the live loop by typing a colon (`:`) followed by the name.

Underneath, write a second tune inside a live loop called `melody` as in Figure 7-13. Although this section of code is beneath the first in sequence, it is played at the same time as the first live loop, just as two hands can play the piano simultaneously.

```
live_loop :melody do
  use_synth :mod_saw
  20.times do
    play_pattern [65,60,62,62,67,60,62,62]
    sleep 1
  end
end
```

Adding Effects

Modern synthesizers have the ability to add effects to sounds. Sonic Pi is no different in that you are able to add studio effects such as reverb, echo and distortion to your code.

To use an effect on your code, or section of your code, wrap it with `with_fx :name of effect do` at the start and `end` at the bottom like this:

```
with_fx :reverb do
  sample :guit_e_fifths
end
```

FIGURE 7-13 Playing multiple tunes at the same time using threads

You are also able to add effects on top of effects, as long as you wrap with `do` and `end`. Here is an example:

```
with_fx :reverb do
  with_fx :distortion do
    sample :guit_e_fifths
  end
end
```

Play around with some effects and add them to your music track. The Help window includes a list of effects that you can use; look under the subheading fx.

Making a Recording of Your Music

By now you will have a music track that you want to listen to on other devices, or that you want to share with friends and family. You could save your code in a text file by clicking on the Save button and then sending it to other people for them to copy into Sonic Pi and play. Alternatively, you can create a recording of how the music sounds.

With a Buffer open displaying the code that you have written to make your music track, click the Record button, quickly followed by the Play button. Once your music has finished, press the Record button again to stop the recording. You are asked to name your recording and save it. It is saved as a `.wav` file, which is a type of sound file that you can play on computing devices (see Figure 7-14).

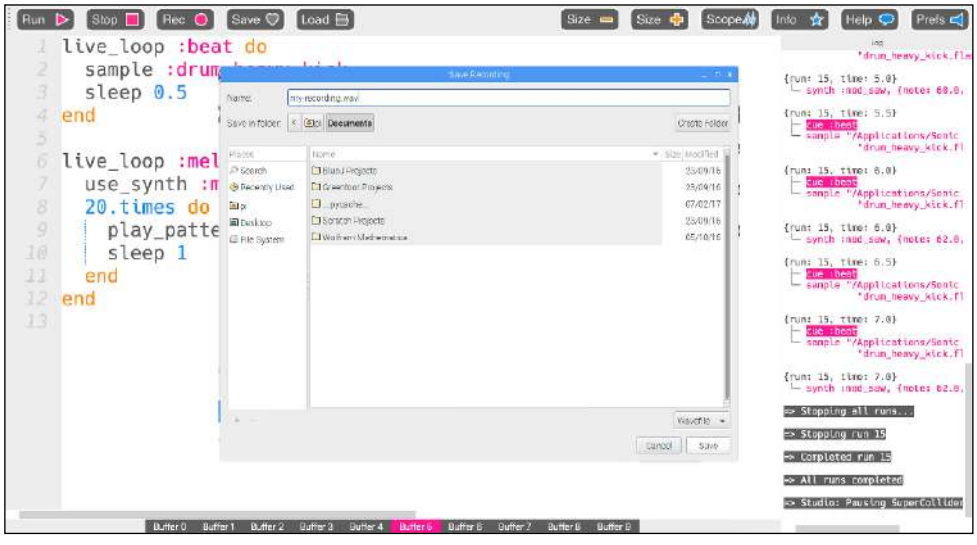


FIGURE 7-14 Saving your coded music as a sound file

Further Adventures with Sonic Pi

If you have enjoyed learning how to make music using Sonic Pi and the programming language Ruby, you can continue having fun with it by looking at these resources:

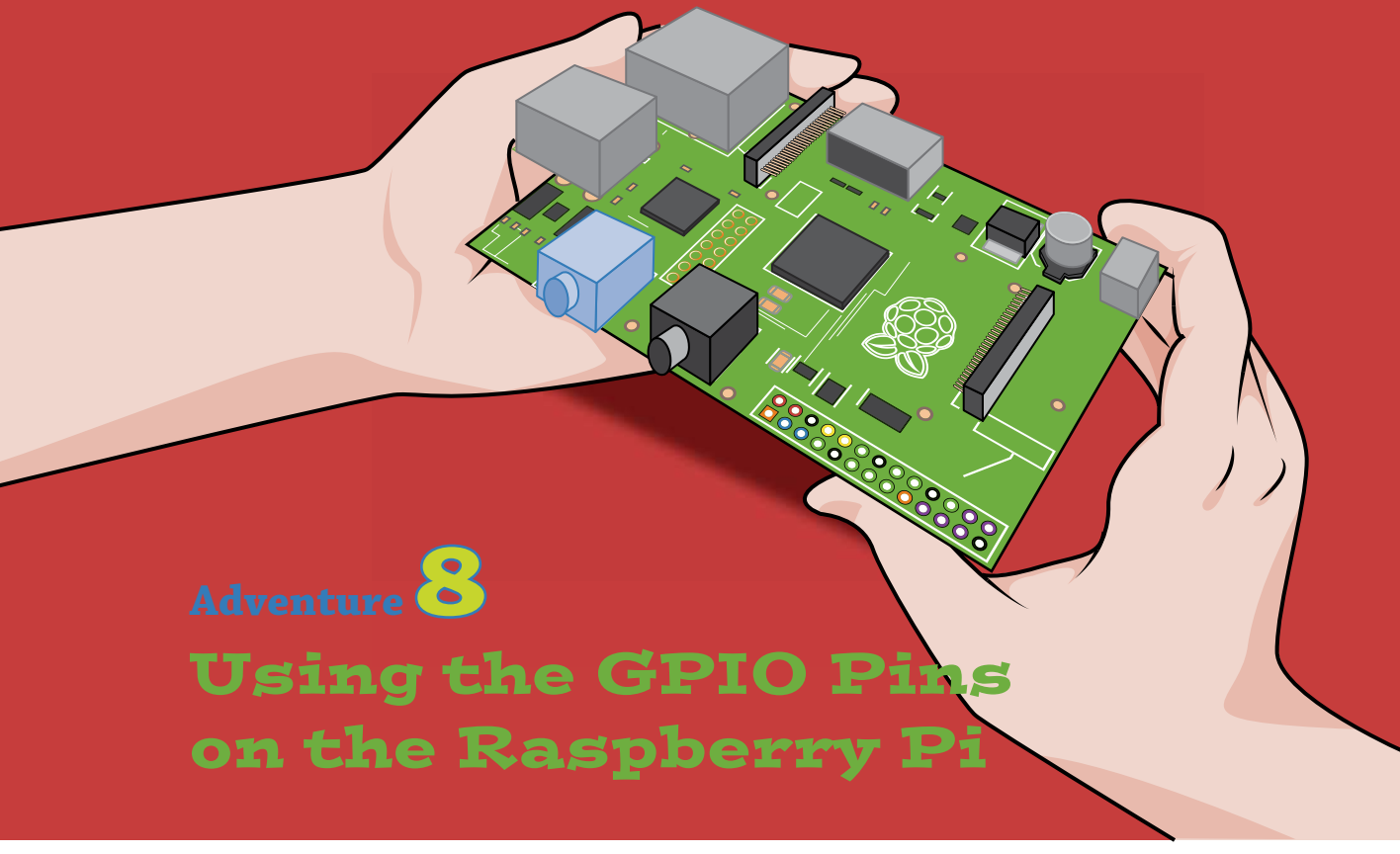
- Sonic Pi website (`http://sonic-pi.net`)
- Kids Ruby (`www.kidsruby.com`)
- Live Coding Music (`http://toplap.org/category/music`)
- Official Ruby Documentation (`www.ruby-lang.org/en`)
- Sonic Pi: Live & Coding (`http://www.sonicpiliveandcoding.com`)

Sonic Pi Command Quick Reference Table	
Command	**Description**
`live_loop :name do...end`	Runs any code between `do` and `end` at the same time as another `live_loop` block.
`play x`	Plays note *x*.
`play_pattern [60,60,67,67,69,69,67]`	Plays a pattern of notes inside a list.
`rand`	Returns a random number.
`.reverse`	An algorithm that reverses the order of notes in a list.
`.shuffle`	An algorithm that shuffles the order of notes in a list.
`use_synth :fm`	Sets the synth sound; in this example, the fm sound.
`with_fx :reverb do ... end`	Adds an effect to a block of sounds; in this example, reverb is added to any code between `do` and `end`.
`x.times do...end`	Runs any code between `do` and `end` *x* number of times.

Achievement Unlocked: Head bopping, toe tapping creator of coded computer music with Sonic Pi!

In the Next Adventure

In the next adventure, you'll see that you can use Raspberry Pi as more than just a tool for programming. With a little knowledge of electronics you are able to create circuits, control lights and even use marshmallows as input buttons to control computer games—all thanks to the GPIO pins on the Raspberry Pi!



Adventure 8
Using the GPIO Pins on the Raspberry Pi

THE GENERAL PURPOSE INPUT OUTPUT (GPIO) pins on the Raspberry Pi are what make it really special. The behaviour of these pins can be controlled or programmed—by you! You can use the pins to sense and control physical objects in the real world, like lights and switches. The pins are located on the main board of the Raspberry Pi, shown in Figure 8-1.

In this adventure, you learn some basics of electronics and then discover how to **output** to a light-emitting diode (LED), making it light up using your Raspberry Pi. For the final project in the adventure, you hook up a marshmallow (yes, a real marshmallow) to **input** a signal to your Raspberry Pi to play a Scratch marshmallow game that senses the press of a button.

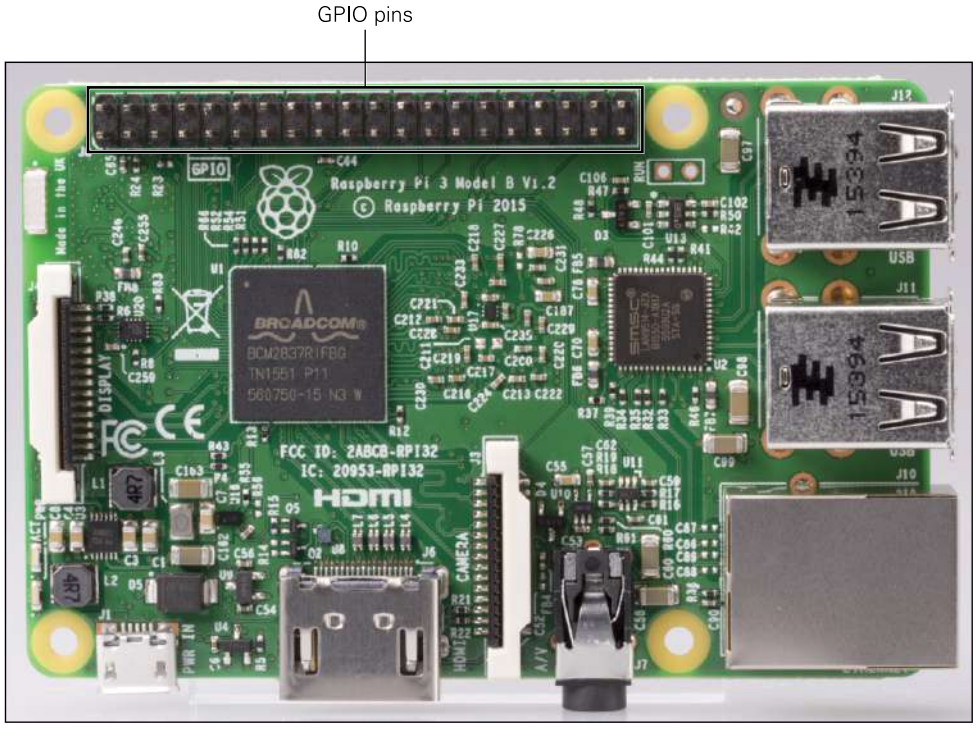


FIGURE 8-1 The General Purpose Input Output (GPIO) pins on a Raspberry Pi

Input refers to the raw data or information that can be entered into a computer system like a Raspberry Pi before it is processed. An example of an input device is a push button or a microphone. The Raspberry Pi has pins that can be connected to these and other devices.

Output refers to the data or information that is communicated to you as it exits a computer system like a Raspberry Pi after it has been processed. An example of an output device is a speaker or monitor screen.

Using a Raspberry Pi GPIO Pin Layout Diagram

Raspberry Pi projects using GPIO pins give you the opportunity to use electronics concepts and techniques to make something happen electronically, such as making an LED light up. Many of the pins have different purposes, and the instructions in this adventure tell you which pin to use for each connection.

There are two revisions of the GPIO pin layout for the Raspberry Pi: one with 26 pins and one with 40 pins. The first Raspberry Pi had 26 GPIO pins. More were added from the Model B+ onwards, although in this chapter, you only use the first 26, so it doesn't matter which Raspberry Pi model you have. Simply ensure that your software is up to date. You can remind yourself how to update the software by returning to Adventure 2. Figure 8-2 shows the layout of all 40 pins on the latest Raspberry Pi board. The first 26 pins have the same layout as the pins on the original Raspberry Pi 1 Model B board. You should refer to this diagram when connecting cables to the pins and when writing your code to program them.

Function/GPIO	J8 Pin		Function/GPIO
3.3V	1	2	5.0V
GPIO2	3	4	5.0V
GPIO3	5	6	0V
GPIO4	7	8	GPIO14
0V	9	10	GPIO15
GPIO17	11	12	GPIO18
GPIO27	13	14	0V
GPIO22	15	16	GPIO23
3.3V	17	18	GPIO24
GPIO10	19	20	0V
GPIO9	21	22	GPIO25
GPIO11	23	24	GPIO8
0V	25	26	GPIO7
(GPIO0) ID_SD	27	28	ID_SC (GPIO1)
GPIO5	29	30	0V
GPIO6	31	32	GPIO12
GPIO13	33	34	0V
GPIO19	35	36	GPIO16
GPIO26	37	38	GPIO20
0V	39	40	GPIO21

FIGURE 8-2 Layout of the GPIO pins on a Raspberry Pi Model B+

You can find this diagram, along with more detail about each GPIO at `https://pinout.xyz/#`.

If you have an original Raspberry Pi, you can still follow the steps, but ensure that you refer to the layout diagram for your board and use the pin numbers from that diagram in your programs, and not those specified in the instructions.

To make it easier to tell which pin is which, Dr Simon Monk has created a Raspberry Leaf template with a label for each pin—you can cut these out and place them over the pins. It is a good idea to download, print and cut out a Raspberry Leaf to help you know which pins to use in the projects in this adventure. You can download the template from the Adventure in Raspberry Pi website or from Dr Monk's electronics website (`www.doctormonk.com/2013/02/raspberry-pi-and-breadboard-raspberry.html`). Dr Monk's site has templates for both the Rev 1 and Rev 2 boards; be sure to download the correct version for your board. This Raspberry Pi 1 template can be used on a Model B+ Raspberry Pi board as well as on both the Raspberry Pi 2 and Pi3, but it fits only the first 26 pins. See Figure 8-3.

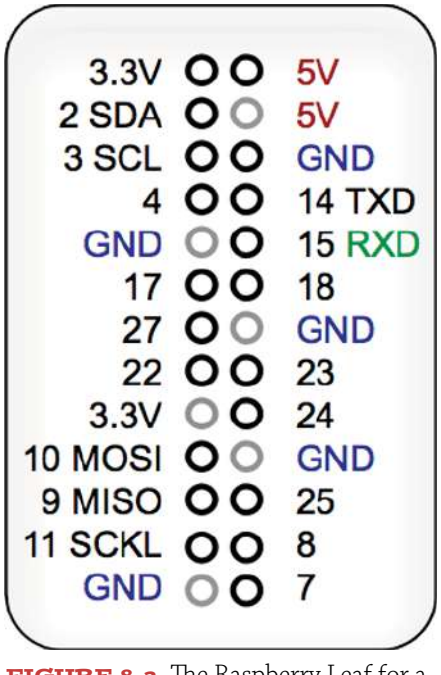


FIGURE 8-3 The Raspberry Leaf for a Raspberry Pi Model B Rev 2 board

Instead of using a paper leaf, you can buy metal pin labels like the one from RasP.io (http://rasp.io/portsplus). The metal pin labels are great if you plan to use them a lot because they're more durable than paper, which can tear when you repeatedly put it on and take it off of the pins.

You can also buy special cases for the Raspberry Pi that include a GPIO pin diagram on them to help you identify the pins, like the one from Pimoroni called the Coupe (https://shop.pimoroni.com/products/pibow-coupe-for-raspberry-pi-3).

You should take great care when connecting cables to the GPIO pins on your Raspberry Pi. There are two reasons for this. First, you should be cautious to protect yourself from harm. Second, the Raspberry Pi is a 3.3V device, and if you plug in anything at a higher voltage than that it will damage the processor and possibly render your board useless.

It is also very important that you connect any external components to the correct pins on the Raspberry Pi, so it is essential that you refer to the correct GPIO revision layout diagram.

Electronics Basics

You are delving into a new world of electronics by using the Raspberry Pi's GPIO pins. If you have never created any electronic circuits before, following the tutorials in this adventure is a good place to learn basic electronics. To get started, you should become familiar with the electronic concepts and components in the following list.

- **Current** is the rate at which electrical energy flows past a point in a circuit. It is the electrical equivalent of the flow rate of water in pipes. Current is measured in amperes (A). Smaller currents are measured in milliamperes (mA).
- **Voltage** is the difference in electrical energy between two points in a circuit. It is the electrical equivalent of water pressure in pipes, and it is this pressure that causes a current to flow through a circuit. Voltage is measured in volts (V).
- **Resistors** are electrical components that resist current in a circuit. For example, LEDs can be damaged by too much current, but if you add the correct value resistor in series with the LED in the circuit to limit the amount of current, the LED is protected. Resistance is measured in *ohms*. You need to pick a resistor with the correct value to limit the current through a circuit; the value of a resistor is shown by coloured bands that are read from left to right. The exercises in this adventure use some resistors and explain how to read the value.

- A **diode** is a device that lets current flow in only one direction. A diode has two terminals, called *anode* and cathode. Current will flow through the diode only when positive voltage is applied to the *anode*, and negative voltage to the cathode.
- A **light-emitting diode** or **LED** is a diode that lights up when electricity passes through it. An LED is an example of an output device. LEDs allow current to pass in only one direction. They come in a variety of colours and have one short leg and one long leg, which helps you determine which way round they need to be placed in a circuit for current to flow through them. The exercises in this adventure use some LEDs.
- A **capacitor** is used to store an electric charge. The capacity that this component has is measured in farads (F). A farad is a very large quantity, so most of the capacitors you see are measured in microfarads.
- A **breadboard** is a reusable device that allows you to create circuits without needing to solder all the components. Breadboards have a number of holes into which you can push wires or jumper cables and components to create circuits. The two columns of holes on either side of the breadboard, between the red and blue lines, are for power. The column next to the red line is for positive connections and the column next to the blue line is for negative connections (see Figure 8-4 for an example of a breadboard). The exercises in this adventure use a breadboard.
- **Jumper cables** can be used to connect the GPIO pins on the Raspberry Pi to a breadboard or other components. They are reusable and do not require soldering. They come in different formats: female-to-male; female-to-female; and male-to-male.
- A **circuit diagram** shows you which electronic components, represented by symbols, are connected to complete a circuit and in what order they should be placed. The exercises in this adventure include circuit diagrams to help you understand how the circuit works and to show you the order in which the components need to be placed for current to flow through.

Figure 8-4 shows a half-size breadboard, a variety of jumper cables, an LED, a push button and some resistors.

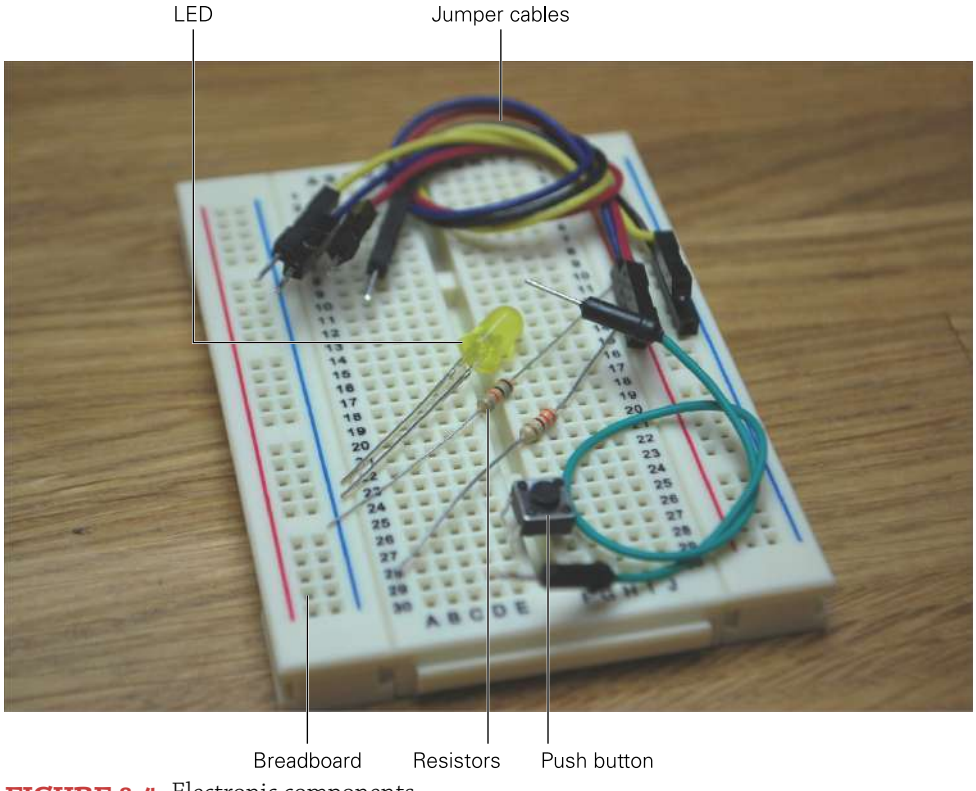


FIGURE 8-4 Electronic components

Using a Python Library to Control GPIO

To use Python to control the GPIO pins, you first need a Python GPIO library installed onto your Raspberry Pi. A library is a collection of already written code that you can use. For example, libraries contain modules that you can use in your code, like the `sleep` function from the `time` module that you used in previous adventures. In this adventure, you use the GPIO library to sense and control the pins.

The Python library to control GPIO, called gpiozero, should be preinstalled on your Raspberry Pi. If you are using an early distribution of Raspbian or another Pi operating system, however, you may need to download and install gpiozero. You can find instructions on how to do this in the gpiozero library documents (`http://gpiozero.readthedocs.io/en/docs-updates/installing.html`). (See Figure 8-5).

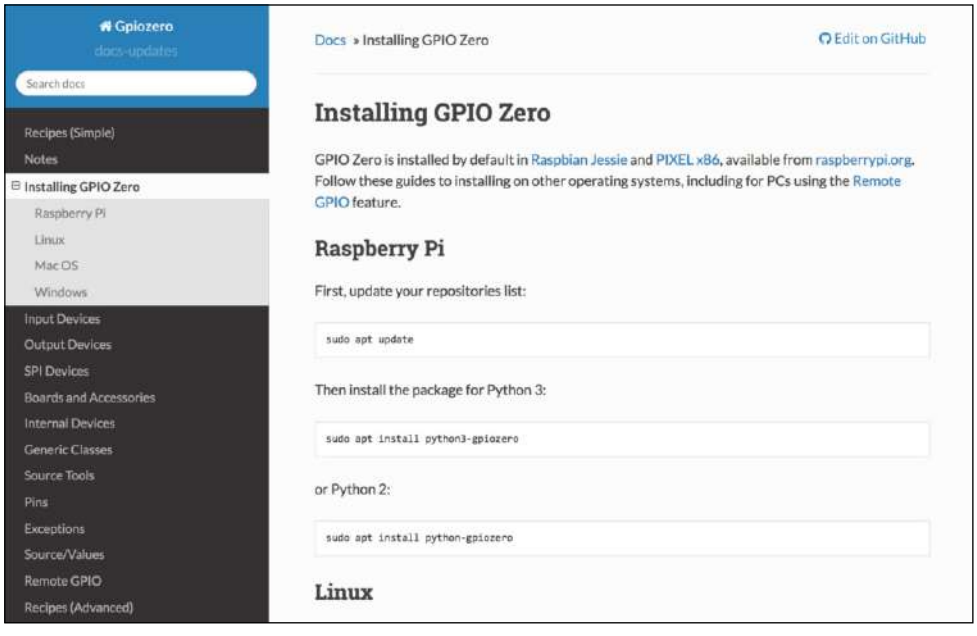


FIGURE 8-5 Official gpiozero instructions to download and install the Python GPIO Library

Making an LED Blink

With the gpiozero Python library on a Raspberry Pi, you can use the GPIO pins to make something physical happen. In this project you control an LED and make it blink.

Along with your Raspberry Pi you need the following items, shown in Figure 8-6:

- A breadboard
- Two jumper cables
- An LED
- A 330 ohm resistor

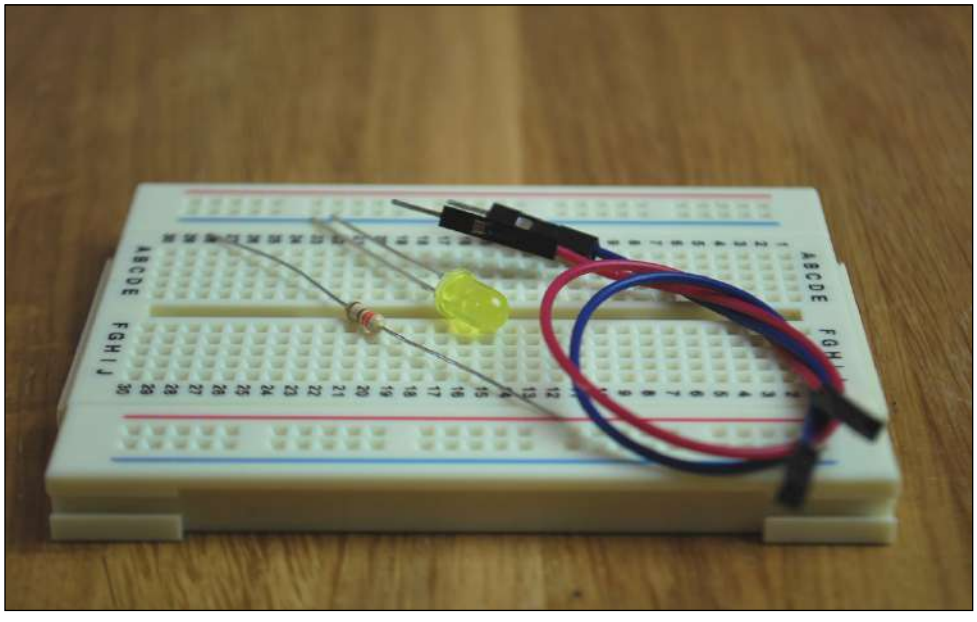


FIGURE 8-6 Components for exercise to make an LED blink

You can purchase the components that you need from the following shops:

Adafruit—`www.adafruit.com/`

CPC Farnell—`http://cpc.farnell.com`

RS Components—`http://uk.rs-online.com/web`

SKPang—`www.skpang.co.uk`

The Pi Hut—`www.thepihut.com`

Creating the LEDblink Python Code

The first part of the project is to write the code that makes your LED blink.

1. Open Python 3 (IDLE) to program the GPIO pins, and select File➪New File to create a blank text editor window to write your Python code to control the GPIO pins.

   Type the following code into your text editor window:

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

These two lines import the modules and their functions that you need to control the GPIO pins on the Raspberry Pi, and to create timed delays between the LED turning on and off. (You use the `time` module in Adventure 5 to wait for user input in the inventory program and the text-based adventure game.)

2. In this project, you are outputting to an LED. Therefore you need to set up the pin that the LED connects to on the Raspberry Pi as an output. To do this, use the command `led = LED(23).`

3. Use a `while True` loop to set the output of GPIO 23 to `on`, which turns on the LED followed by a pause for one second. Then set the output of GPIO 23 to `off`, followed by another one-second pause. When this is looped over and over, the LED repeatedly turns on and off with a one-second delay between each change in state.

```
while True:
    led.on()
    time.sleep(1)
    led.off()
    time.sleep(1)
```

4. Save the file as `LEDblink.py` in `Documents`.

DIGGING INTO THE CODE

`gpiozero` is a Python library created especially for the Raspberry Pi GPIO pins. It has been designed to make it as simple as possible to program everyday GPIO components like LEDs, buttons, buzzers, sensors and motor controllers. With other Python libraries like `RPi.GPIO`, you are required to specify which pins you are going to use and then configure them as inputs and outputs. Once you understand more about programming and electronics you may want to graduate to using a lower-level library to fully appreciate how to control hardware with code. The exercises in this book use `gpiozero` to help you take your first steps with physical computing.

Connecting the LEDblink Components

Before running your program to make the LED blink, you need to assemble the electronic components and connect them to the Raspberry Pi GPIO pins. Figure 8-7 shows the Raspberry Pi on the left and a breadboard on the right. Use this diagram and the following steps to help you connect the right cables and components in the circuit.

1. Start by plugging a female-to-male jumper cable from GPIO pin 23 of your Raspberry Pi to the A10 hole on your breadboard (the red cable in Figure 8-7).

It helps to use different coloured wires. The jumper cable easily fits onto the pins of your Raspberry Pi on one end, and the other end fits into the holes on the breadboard. Make sure that you gently push them down as far as they will go to make a secure connection.

2. Plug another female-to-male jumper cable from a ground pin, sometimes represented as GND on Raspberry Pi GPIO diagrams (the blue cable in the figure). On a Raspberry Pi board, the third pin from the top on the outside strip is a ground pin (see Figure 8-7). Remember to use Figure 8-2 so that you know which pins are which!

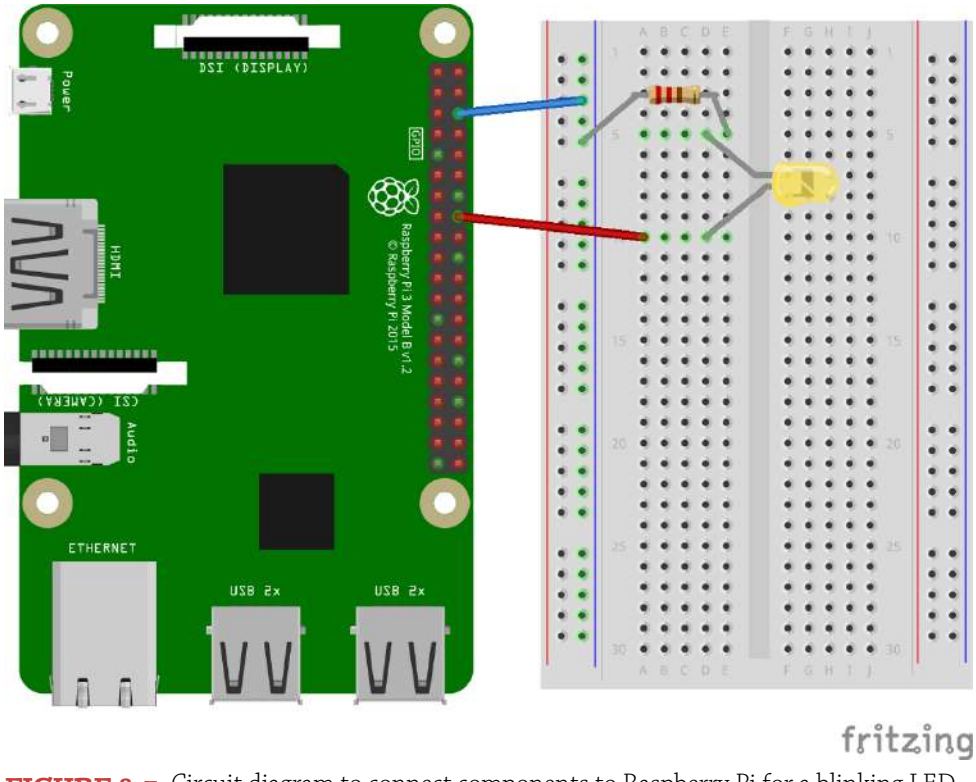


FIGURE 8-7 Circuit diagram to connect components to Raspberry Pi for a blinking LED

3. Plug the other end of the jumper cable into a hole in the second column, between the red and blue lines on the breadboard. Remember that these two columns between the red and blue lines are for power—one for positive (red) and one for negative (blue). You want to plug your jumper cable into the blue negative column, three rows down.

4. Add a 330 ohm resistor by pushing one of its legs into E5 and the other leg into a hole in row five of the blue negative power column on the breadboard. It does not matter which way round the resistor is placed.

   Remember that LEDs can only pass current in one direction. For the LED to work, you must make sure that you place the longer leg into the same row (D10) as the jumper cable connecting to GPIO 23. The short leg must be placed into a hole on the same row as the resistor (D5). Refer to Figure 8-7 for guidance.

Running LEDblink.py in IDLE

Now that you have created your LED circuit, you can run your Python code. Select Run ➪ Run Module in the Python 3 (IDLE) menu.

You should see your LED turn on and off. Is that cool or what?

To interrupt the program, press CTRL + C on the keyboard. Figure 8-8 shows the code.

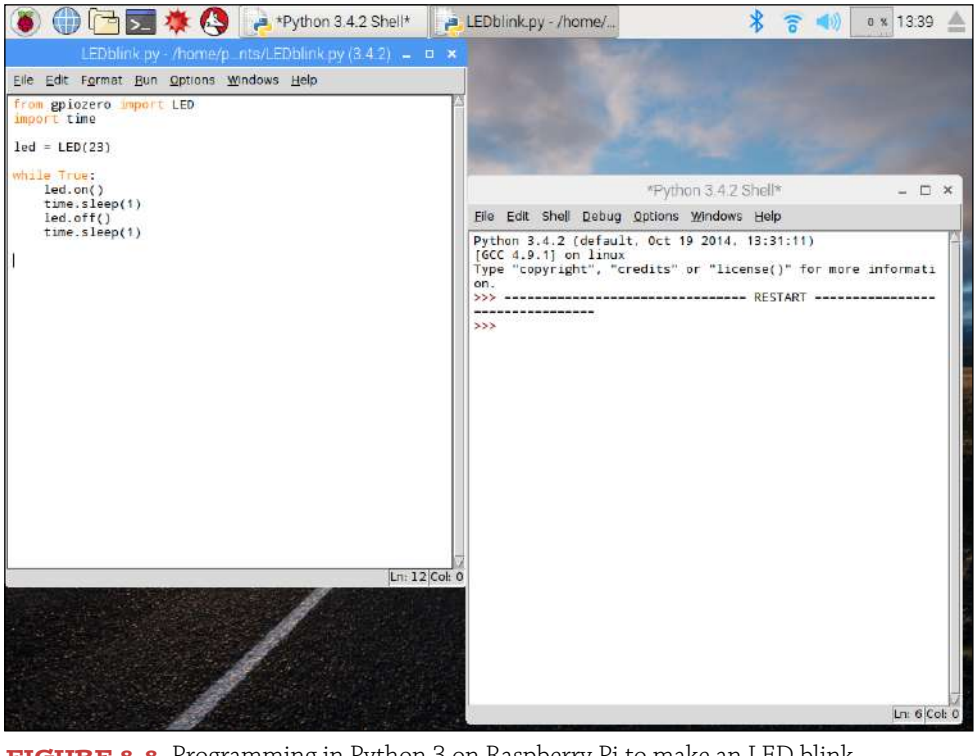


FIGURE 8-8 Programming in Python 3 on Raspberry Pi to make an LED blink

Using a Button to Turn on an LED

So far, you have controlled an LED, which is an output device. In this exercise you add an input device in the form of a button that starts the light sequence when pushed.

For this project, you need your Raspberry Pi plus the following items:

- A breadboard
- Five jumper cables
- A simple push button
- An LED
- A 330-ohm resistor to protect the LED

Creating the buttonLED Python Code

In Python 3 (IDLE), amend your `LEDblink` program to include the following lines (highlighted in bold):

```
from gpiozero import LED, Button
import time

led = LED(23)
button = Button(24)
```

As before, you set a GPIO for output (GPIO 23). You also need to set a GPIO pin to detect *input*. Use GPIO 24 for this purpose.

In the previous project, you used a `while True` loop to repeatedly turn an LED on and off with a one-second interval. In this project, you only want the LED to turn on when the button is pushed; therefore, you need to introduce a condition. You use `if` to set the condition: *if the button (connected to GPIO 24) is pressed, then turn the LED (connected to GPIO 23) on*. But this is only one part of the condition. You also need to set the conditions for the button not being pushed; what should the LED do then? For this part of the condition, you use `else`: *else the LED should be off*.

```
while True:
    if button.is_pressed:
        led.on()
    else:
        led.off()
    time.sleep(0.1)
```

Save the file as `buttonLED.py` in `Documents`.

CHALLENGE

Why did you add `time.sleep(0.1)` to the `while` loop in the `buttonLED.py` program? What might happen if you remove it? Have a go to find out!

Connecting the buttonLED Components

As with the program for making an LED blink, before you run your button LED program you need to assemble the electronic components and connect them to the Raspberry Pi GPIO pins. If you use the component configuration from the previous exercise, you need to add only a few extra parts. Figure 8-9 shows the Raspberry Pi on the left and a breadboard on the right. Follow this diagram to help you connect the right cables and components in the circuit.

1. Leaving the original circuit complete, take a small button switch and place it across the ride in the centre of the breadboard, with two legs in holes on column E and two legs in holes in column F on rows 21 and 23, as shown in Figure 8-9.
2. When the button is in place, take a male-to-male jumper cable (green cable) and place one end into A23 on the same row as the other leg of the button on column E. Place the other end into the blue (negative) ground column of the breadboard.
3. Connect a male-to-female jumper cable (yellow cable) from A21 on the breadboard to GPIO pin 24 on the Raspberry Pi.
4. Finally, add a male-to-female jumper cable (black cable) from the red column near the top of the breadboard to the top GPIO pin 3V3, which will power the circuit.

If you have an early Raspberry Pi model, you can compare your configuration to the one shown in Figure 8-9.

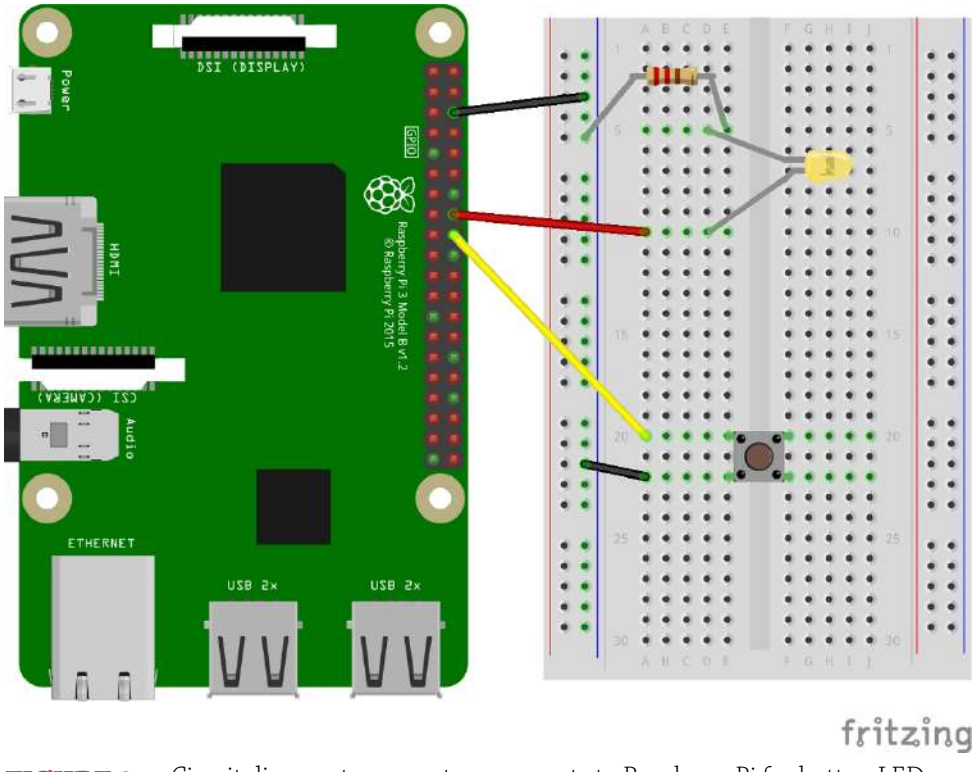


FIGURE 8-9 Circuit diagram to connect components to Raspberry Pi for button LED

Running buttonLED.py in IDLE

As before, you need to run your code from Python 3 (IDLE). To do this select Run ➪ Run Module.

When you press the button, the LED should light up, as shown in Figure 8-10. The LED should not light up until you press the button on your circuit. If the light comes on before you press the button or does not light when you press the button, go back and check your wiring using the diagrams.

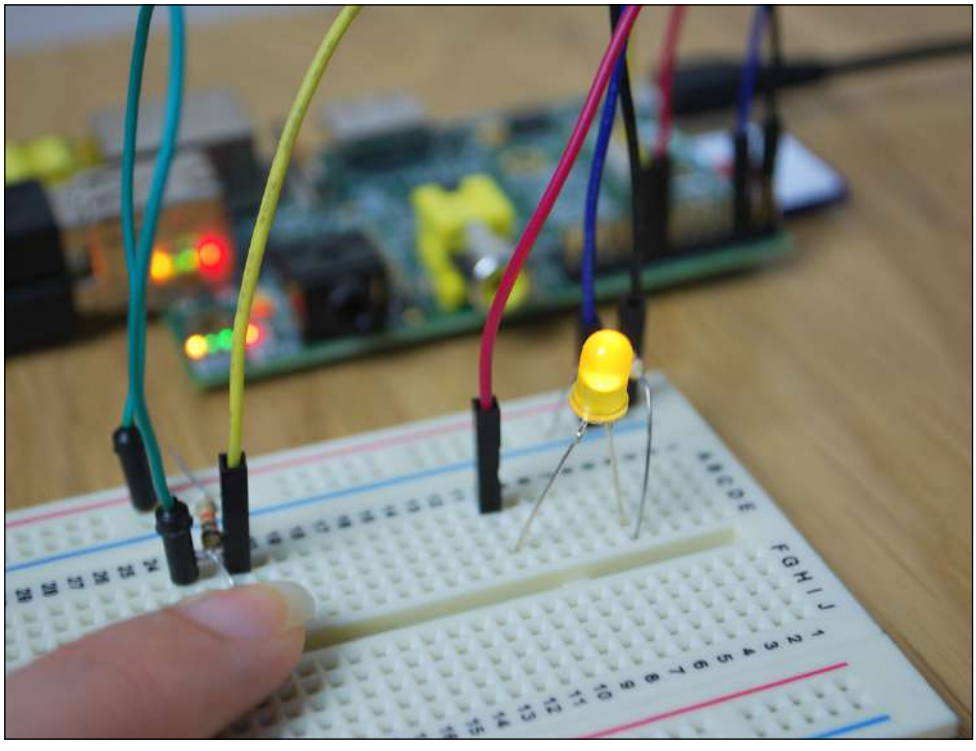


FIGURE 8-10 Pressing the button makes the LED light up!

CHALLENGE

Why not improve your code so that when you press the button once it turns on the LED, and when you press the button again it turns off the LED?

Using a PIR Motion Sensor to Trigger a Sound

Buttons or switches are a great way to trigger something to happen, such as turning on a light. However, they are not the only components you can have fun with. In this exercise you create a circuit with a **Passive Infrared (PIR) motion sensor** so that when it detects movement, text is printed on the screen.

For this project, you need your Raspberry Pi plus the following items:

- Three female-to-female jumper cables
- A Passive Infrared (PIR) motion sensor
- Your Raspberry Pi GPIO pin labeller to help you identify the GPIO pins that you need to use

All living things emit radiation all of the time. It is nothing to worry about as the type of radiation we emit is infrared radiation (IR), which is pretty harmless at low levels.

A **PIR motion sensor** detects changes in the amount of IR radiation it receives. When there is a significant change in the amount of IR the sensor detects, a pulse is triggered. This means that a PIR motion sensor can detect when someone moves in front of it.

We can use a PIR motion sensor to trigger an event using code.

Creating the Motion-Sensing Python Code

The first part of this project is to write the code that detects motion with the PIR motion sensor and output some text to the screen.

1. Open Python 3 (IDLE) and select File➪New File to create a blank text editor window to write your Python code. Like the other GPIO python programs you have written you need to import the modules and libraries you need.

   Type the following code into your text editor window:

```
from gpiozero import MotionSensor
import time
```

2. In this project, you are using a PIR motion sensor as an input. Therefore you need to set up the GPIO pin that the PIR motion sensor connects to on the Raspberry Pi as an input. To do this, use this command:

```
pir = MotionSensor(4)
```

3. Create a condition inside a loop by typing the following:

```
while True:
    if pir.motion_detected:
        print("Detected you moving")
    time.sleep(1)
```

Remember that indentation is important in Python. `time.sleep(1)` needs to be indented the same amount as the `if` command.

4. Save the file as `PIRmotion.py` in `Documents`.

Connecting the PIRmotion Components

If you have any components connected to your Raspberry Pi from previous activities then remove them so that all your GPIO pins are free for use. Figure 8-11 shows the Raspberry Pi on the left and a PIR motion sensor on the right. Follow the diagram in the figure to help you connect the right cables and components in the circuit.

1. Take a look at your PIR motion sensor. It has three pins labeled VCC, GND and OUT. (You may need to take the plastic case off the top of your sensor to see the labels.) You need to connect each of these pins to GPIO pins on your Raspberry Pi for it to work.
2. Connect a female-to-female jumper cable (yellow cable) from the pin labeled OUT on the PIR motion sensor to GPIO pin 4 on the Raspberry Pi.
3. Add a female-to-female jumper cable (black cable) from the GND pin on the PIR motion sensor to any ground pin.
4. Finally, add a female-to-female jumper cable (red cable) from the PIR pin labeled VCC to the top right GPIO pin 5V. This cable powers the circuit.

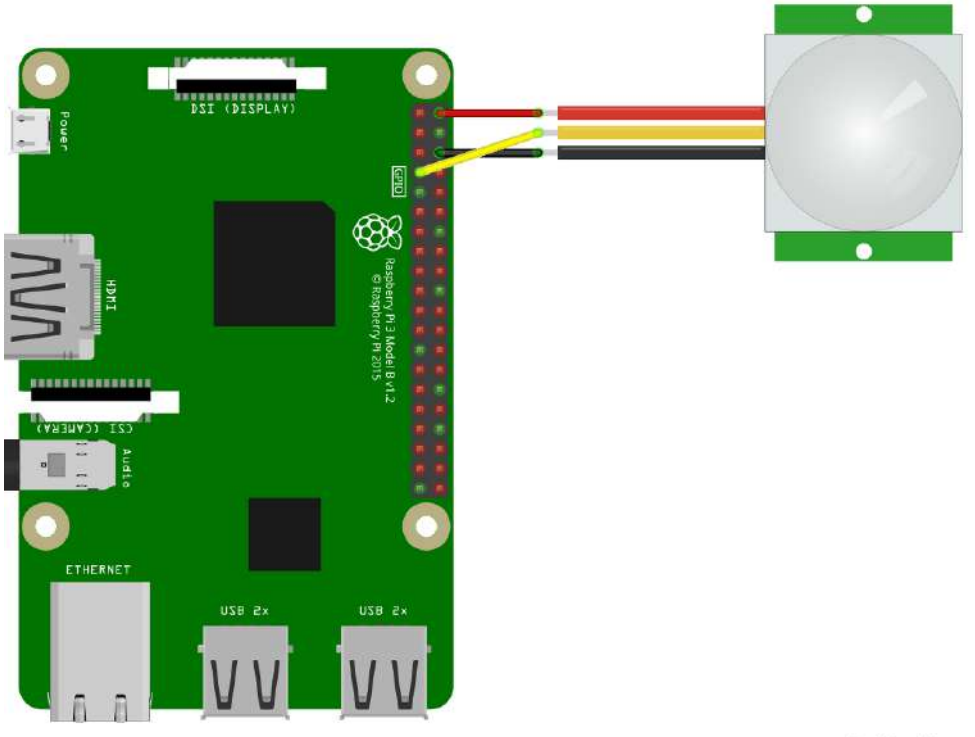


FIGURE 8-11 Circuit diagram to connect components to Raspberry Pi for PIR motion sensor

Running PIRmotion.py in IDLE

As before, you need to run your code from Python 3 (IDLE). To do this select Run ➪ Run Module.

When you move in front of the PIR motion sensor, the statement `Detected you moving` should appear on the screen, as shown in Figure 8-12. If the text does not appear on the screen, go back and check your wiring using the diagrams. PIR motion sensors can be tricky to debug. Try placing the sensor inside a cardboard box or cup so that you can limit the field it is sensing movement from.

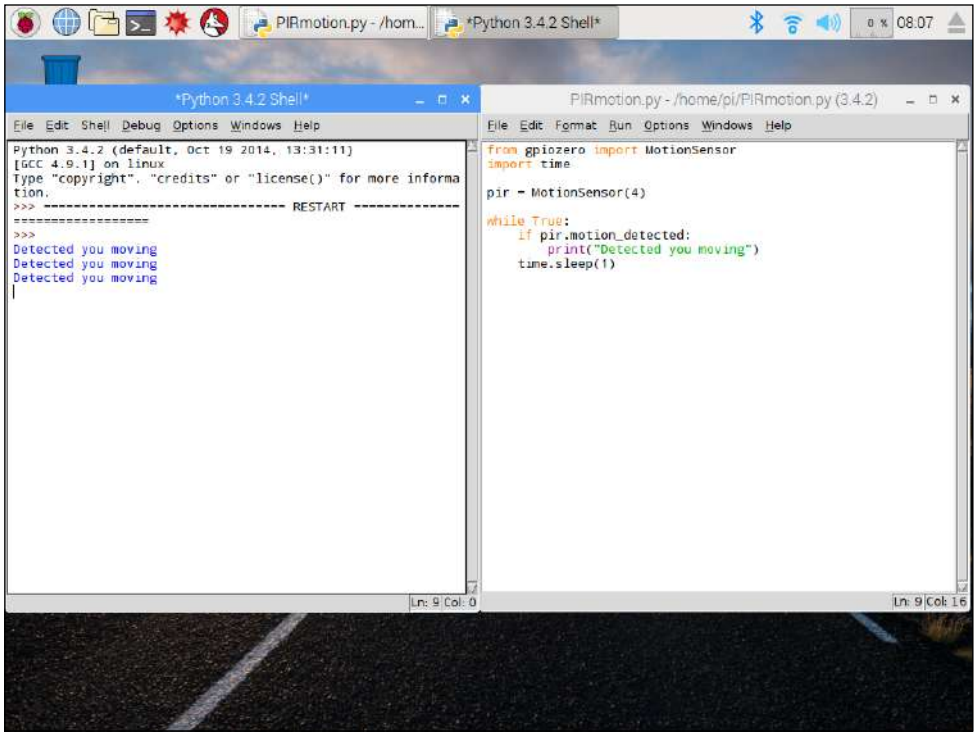


FIGURE 8-12 Movement detected and printed to the screen on a Raspberry Pi

CHALLENGE

Why not add an output component light an LED or buzzer to trigger when the PIR motion sensor detects movement?

The Marshmallow Challenge

Using the Raspberry Pi GPIO pins for electronic projects can be a great way to learn. However, you can have some fun too. In this project, you use your new GPIO pin Python programming knowledge to use a marshmallow as an input button, map it to a keyboard letter and use it to control a game that you have created in Scratch! Figure 8-13 shows the game in action. You need these items:

- Two female-to-female jumper cables
- Some marshmallows (yes, real ones!)
- Two metal pins or metal paper clips
- Your Raspberry Pi GPIO pin labeller to help you identify the GPIO pins that you need to use

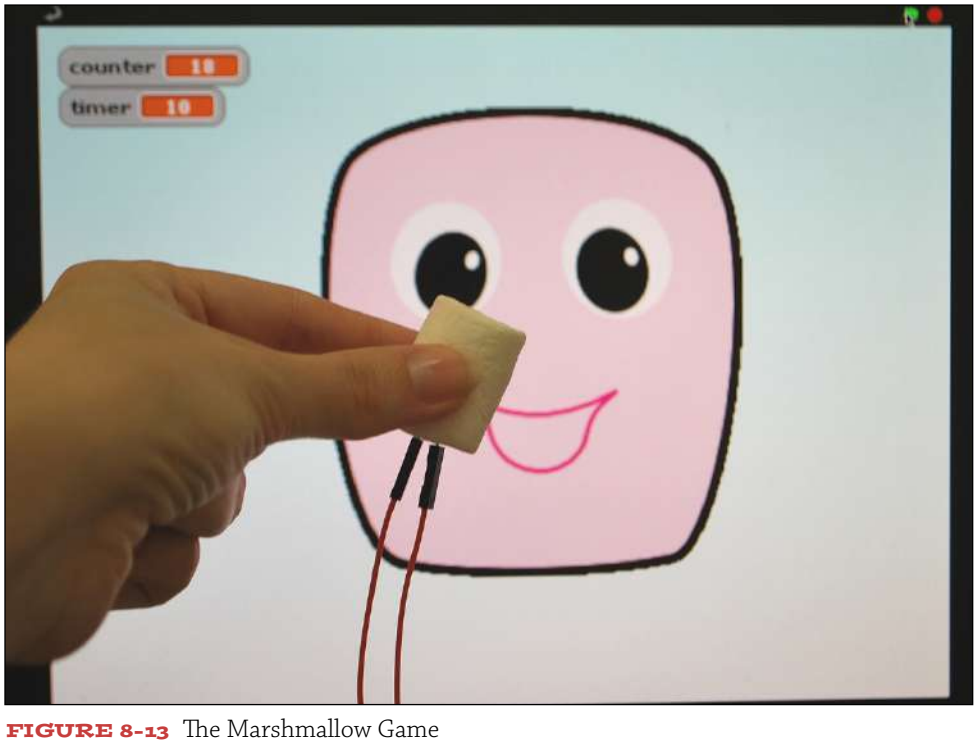


FIGURE 8-13 The Marshmallow Game

Creating the Marshmallow Button

The first step is to write the code to create a marshmallow button. Follow these steps:

1. Open Python 3 (IDLE) from the programming menu and create a new file by selecting File➪New File. Save the empty file as `marshmallow.py` in the Document directory.

2. Type the following code:

```
from gpiozero import Button
import time

button = Button(4)

while True:
  if button.is_pressed:
    print("Marshmallow makes a good input")
  time.sleep(0.1)
```

Press CTRL+S to save the file.

3. Take a jumper cable and carefully push a pin into the end. (You could even use a paper clip that has been straightened out.) Take the other end of the jumper cable and push it into GPIO pin 4 on your Raspberry Pi.

Do the same with the second jumper cable, only this time plug it into a ground pin on the Raspberry Pi.

Poke the other ends of both jumper cables (the metal pin ends) into a marshmallow, as shown in Figure 8-14.

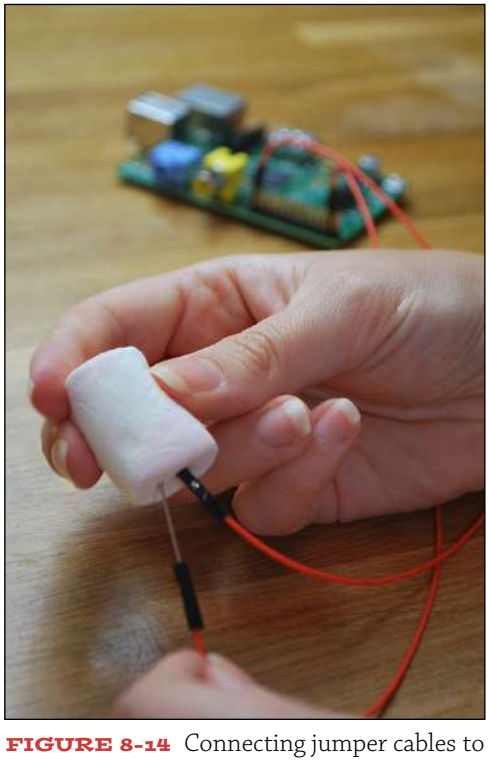


FIGURE 8-14 Connecting jumper cables to GPIO pins and a marshmallow

4. Go back to your Python code and press F5 to run it.

   Now gently squeeze the marshmallow and your marshmallow message is printed to the shell window. If nothing happens, check that you have pushed the pins into the marshmallow and cables as far as they will go.

Now that you have tested to see that your marshmallow button works, you need to map it to a keyboard key so that when the marshmallow is pressed, the game thinks that the letter a on the keyboard is being pressed. This becomes important when you make or download a Scratch Marshmallow Game.

Mapping Marshmallow Input to a Keyboard Key

Follow these steps to map your marshmallow button to a key on the keyboard:

1. In a Terminal window type the following command:

```
sudo pip3 install pyuserinput
```

   This command downloads and installs `pyuserinput`, an application and library that you need to be able to map your keyboard keys in your Python code.

   Figure 8-15 shows the code for this step.

2. Edit the `marshmallow` code to use the `pykeyboard` Python library to map the input of the marshmallow to a key on the keyboard.

   Amend the Python code to include the new lines (shown in bold):

```
from gpiozero import Button
from pykeyboard import PyKeyboard
import time

button = Button(4)
k = PyKeyboard()

while True:
  if button.is_pressed:
    print("marshmallow makes a good input")
    k.press_key("a")
    time.sleep(0.1)
    k.release_key("a")
  time.sleep(0.1)
```

   Here the a keyboard key has been used for the marshmallow button. You could use any letter of the alphabet.

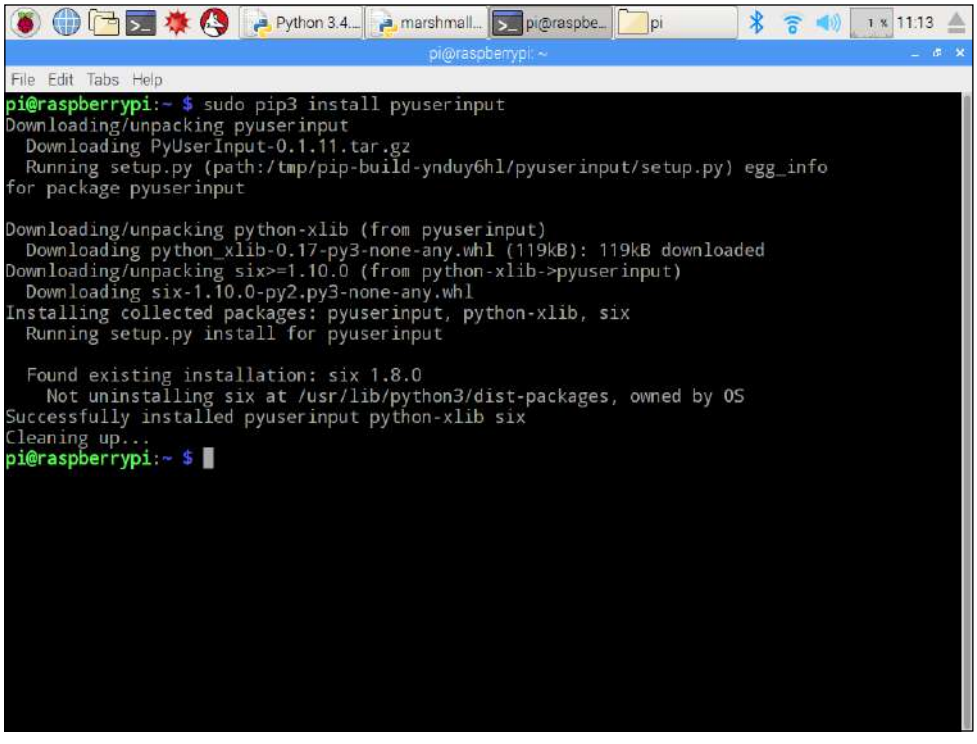


FIGURE 8-15 Downloading and extracting `python-uinput`

The marshmallow button may not work if you do not have a keyboard plugged in.

3. Press CTRL + S to save your program, then test press F5 on your keyboard to test that the program works.

   You should also test the program by pressing marshmallow, which should result in the letter a being displayed.

Scratch Marshmallow Game

Now that your marshmallow is mapped to the a key, you can create a Scratch game with a counter to count each time you squeeze the marshmallow. You can download the completed Scratch Marshmallow Game from the Adventures in Raspberry Pi website at `www.wiley.com/go/adventuresinrp3E`. The object of the game is to see

how many marshmallow presses you can record in 10 seconds, using the marshmallow input button that you have created. Refer to Figure 8-16, the completed game in Scratch, as you work through the following steps.

1. Open Scratch by opening the main menu at the top of the screen and navigating to Programming ➪ Scratch. Select File ➪ Save As and name the file `Marshmallow Game` before clicking Save.

2. Delete the Cat Scratch sprite by right-clicking the sprite and selecting Delete from the drop-down menu.

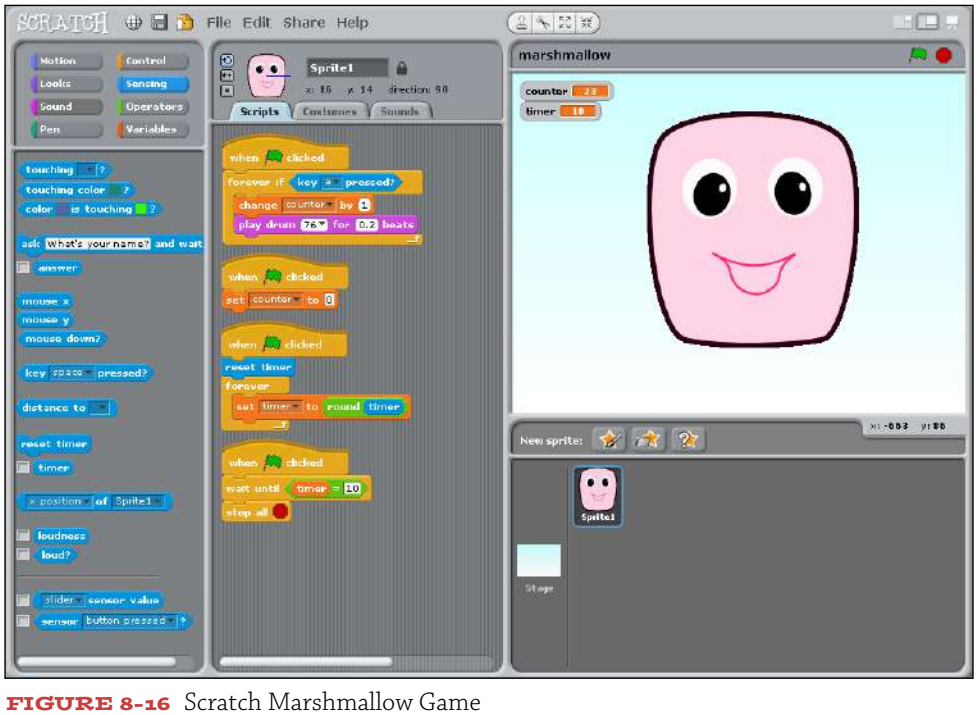


FIGURE 8-16 Scratch Marshmallow Game

3. Click the Paint New Sprite icon above the Sprites palette (the icon with the paintbrush and star) and draw a marshmallow character using the paint editor. You can use the rectangle or circle shape tools to create your marshmallow. Alternatively, if you feel confident you can try using the paintbrush tool to draw freehand. When you are happy with your design, click OK to exit the Paint Editor window.

   Another option, if your Raspberry Pi is connected to the Internet, is to download the marshmallow sprite used in this project (`marshmallow.png`) from the Adventures in Raspberry Pi website at `www.wiley.com/go/adventuresinrp3E`.

4. You need to create two variables for this game. Click Variables in the Blocks palette and then click Make a Variable. The New Variable window opens and asks you to type a name for your variable. Name the first variable `counter` and ensure that ***For all sprites*** is checked before clicking OK. This variable counts the number of times the marshmallow button has been pressed.

   Follow the same steps to create a second variable and name it `timer`. This variable sets the time limit for the marshmallow challenge.

5. Now that all the blocks you need to create the scripts for the marshmallow game are available, click Control on the Blocks palette and drag the control block `When`  `clicked` onto the Scripts tab for the marshmallow sprite. Add a `forever if` control block underneath and connect them.

6. Click the Sensing blocks palette and select the `key space pressed` block. Place it into the hexagonal space on the `forever if` control block. After you have placed this block you need to change `space` to `a` or the letter of the key to which you assigned your marshmallow button-press in the previous part of this project.

7. Add the variable block `change counter by 1` inside the `forever if` block, followed by the sound block `play drum 48 for 0.2 beats`. Change the drum to 76 using the drop-down menu.

8. When the game begins, the counter needs to be reset to 0 to record the player's score. Add another `when`  `clicked` control block to the Scripts tab and connect a `set counter to 0` variable block to it.

9. Remember to save the work you have done so far by selecting File➪Save. Then test that your game script written so far works, by clicking the  icon and using your marshmallow button to check whether the counter records how many times it is pressed. Don't forget that the Python script you created for the button needs to be running too!

10. To give the player a time limit, you need to add two more scripts. Add another `when`  `clicked` control block to the Scripts tab and connect the sensing block `reset timer` to it. Underneath, connect a `forever` looping control block, and a `set timer to 0` variable block inside it. (You may need to select Timer from the Variable block drop-down menu.)

    Click the Operators blocks palette and select the `round` block. Place it inside the `set timer` variable block where the value `0` is. Then add the sensing block `timer` inside the empty space of the `round` operator block you just placed.

    This completed variable block should now read `set timer to round timer`. This script resets the timer at the start of the game to 0, and then counts upwards every second.

11. The final script sets the time limit for the game. Add a `when`  `clicked` control block to the Scripts tab, and connect a `wait until` control block to it, followed by a `stop all` control block.

Drag the operators block `_ = _` (equal to) and place it inside the `wait until` hexagon space. In the left-hand space before the `=` sign, add the variable block `timer` and on the right-hand side type the value `10`.

This script waits until the timer reaches 10 seconds before stopping the game.

12. Finally, save your game by selecting File➪Save. Then test that your scripts work by clicking the  icon and using your marshmallow button to check whether the counter records how many times it is pressed, and the timer counts up to 10 before stopping the game.

CHALLENGE

What other functionality could you add to this game? Here are some ideas:

- Costumes for the marshmallow sprite so that when the button is pressed it becomes animated.
- A Stage background with a Game Over screen. This would also require scripts to broadcast a message, like the one for the Adventurer Game in Adventure 3.
- A scoreboard for the highest number of presses.

You could also make a similar game using Python.

Further Adventures with GPIO Pins

As you can see, programming the GPIO pins on the Raspberry Pi opens a Pandora's box of opportunity. With a little electronics know-how and some imagination you can control lots of things in the real world. The Raspberry Pi website (`www.raspberrypi.org`) is full of great examples of how people are controlling their environments with their Pis.

Now that you have learned how to combine some basic electronics powered and commanded by the Raspberry Pi, you may want to continue learning more. Here are some resources to take you further:

- To learn more about gpiozero and code for other components: `https://gpiozero.readthedocs.io`
- Alex Eames' website RasPi.TV has a basic set of tutorials: `http://raspi.tv/category/raspberry-pi`
- If you want to learn how to control the Raspberry Pi GPIO pins using Scratch, the Raspberry Pi website has some great ideas: `www.raspberrypi.org/learning/physical-computing-with-scratch/`

- To learn more about electronic projects and using the Raspberry Pi GPIO pins, the Adafruit Learning System is a great platform to start with: `http://learn.adafruit.com/`

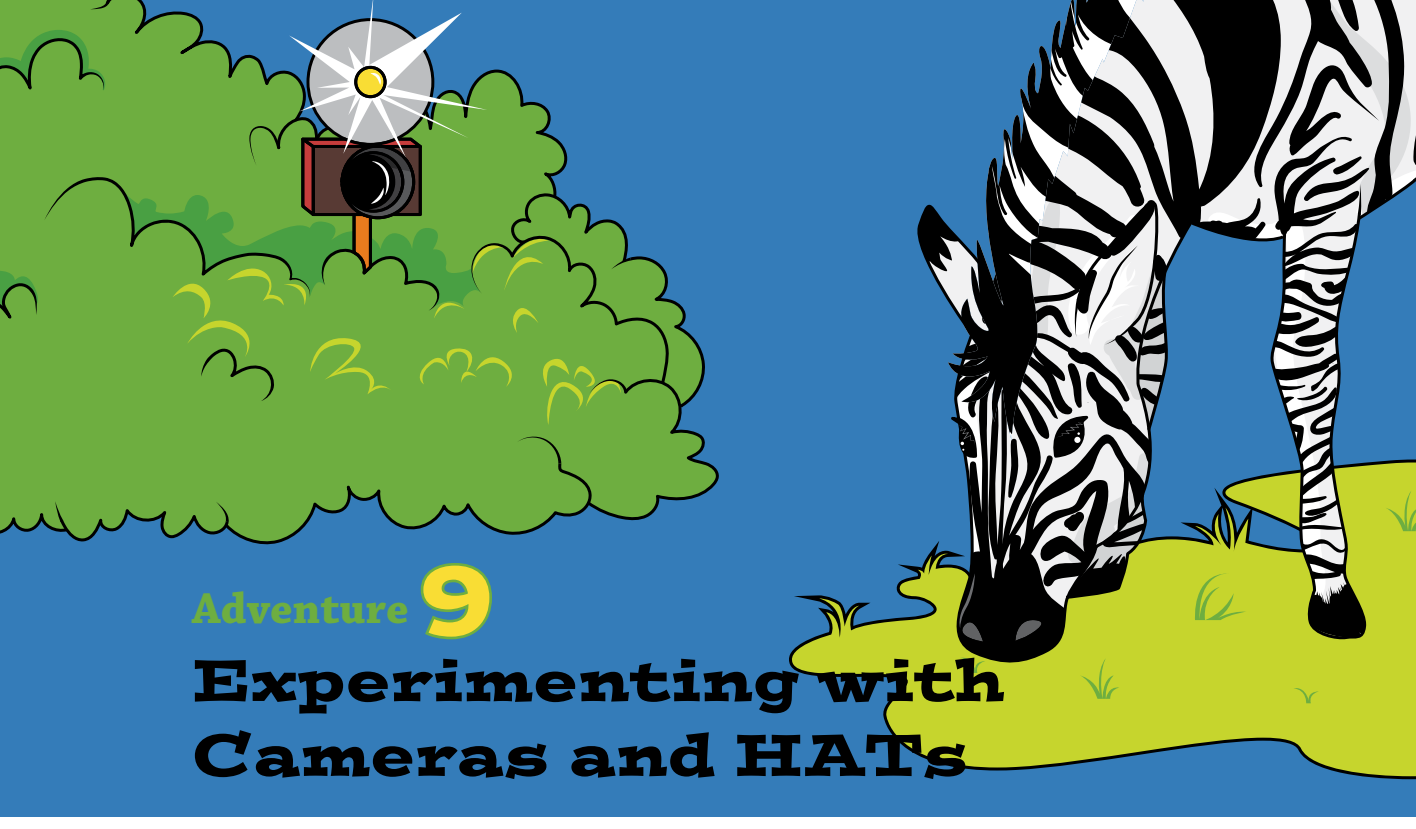
GPIO Pins Command Quick Reference Table

Command	Description
`from gpiozero import LED, Button`	Imports the Raspberry Pi gpiozero module
`Led = LED(2)`	Creates an LED against a pin number as well as identifying it as an output
`led.on()`	Sets output GPIO to true or on
`led.off()`	Sets output GPIO to false or off
`if button.is_pressed:`	Sets a condition for a button press
`if pir.motion_detected:`	Sets a condition for a PIR motion sensor

Achievement Unlocked: Conquering electronics with a Raspberry Pi!

In the Next Adventure

You don't always need to program individual electronic components; you can also buy specially designed hardware that fits on top of the GPIO pins on the Raspberry Pi, called HATs or connect to the Raspberry Pi board connectors like the camera accessory. In Adventure 9, you learn more about the features of the Raspberry Pi Camera, the Sense HAT and the Explorer HAT before programming them with Python.



Adventure 9

Experimenting with Cameras and HATs

BEING ABLE TO connect components to the Raspberry Pi GPIO pins and control them with code is one of the Raspberry Pi's most fantastic features, but individual components can be quite fiddly, which means they can be easy to lose and difficult to debug when projects do not work the first time! Luckily there are extra accessories developed for use with the Raspberry Pi that can be added on top to extend its features and cut down the amount of time required to build breadboard circuits.

One of the easiest and most exciting-to-use peripherals is a camera. The official Raspberry Pi camera slots straight onto the Raspberry Pi, and you can use it to capture images and videos. Lots of projects make use of the small setup like the NatureBytes wildlife cam kit (`http://naturebytes.org/`), which detects the movement of animals in the wild and captures a picture of them. This camera is a great way to find out what types of wildlife visit your garden or neighbourhood!

Raspberry Pi accessories that are fitted on top of the GPIO pins are referred to as HAT, which stands for Hardware Added on Top. These components are easy to fit and program from your Raspberry Pi. In this adventure, you're introduced to two different HATs: the Explorer HAT made by Sheffield-based makers Pimoroni, and the Sense HAT, which was designed by the Raspberry Pi Foundation for use on board the International Space Station (ISS) as part of the Astro Pi project. There are two Raspberry Pis with Sense HATs onboard the ISS; both run code written by school-aged children from countries taking part in educational initiatives from the European Space Agency (ESA).

It is worth noting that HATs are not compatible with the original Raspberry Pi that has 26 pins. HATs only work with Raspberry Pis that have 40 pins.

Getting Started with the Raspberry Pi Camera

The Raspberry Pi Camera (see Figure 9-1) is an inexpensive accessory designed by the makers of the Raspberry Pi. After you add this accessory, your Raspberry Pi will be transformed into a full HD camera to take still images and record video. You can even use it at night with the infrared camera (the Raspberry Pi NoIR Camera)!

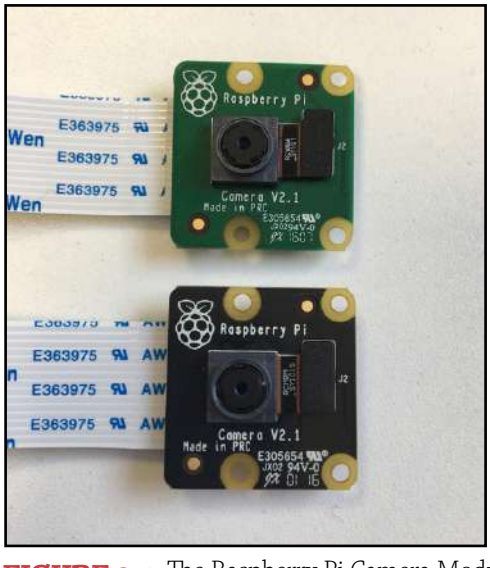


FIGURE 9-1 The Raspberry Pi Camera Module (top) and Raspberry Pi NoIR Camera Module (bottom)

Connecting the Camera to Your Raspberry Pi

The first step is to connect the camera to your Raspberry Pi. *Note:* Ensure that your Raspberry Pi is disconnected from any power or peripherals before you begin.

1. Locate the camera port on the Raspberry Pi (see Figure 9-2). It is in between the HDMI port and the Ethernet port. This is where the ribbon cable of the camera will slot into.

2. Use your fingernails to pull up the sides of the camera connector on the Raspberry Pi.
3. Place the camera ribbon cable into the slot with the blue strip facing the Ethernet and USB ports.
4. While you hold the ribbon cable in place, push down the camera port connector that you opened in step 2.
5. Once the camera is connected, plug in the rest of your peripherals and boot your Raspberry Pi.
6. Open the Raspberry Pi Configuration tool by clicking the Main Menu, navigating to Preferences and selecting it from the menu.
7. Click Interfaces and check the enabled box next to camera. Click OK and then reboot your Raspberry Pi to enable the camera.

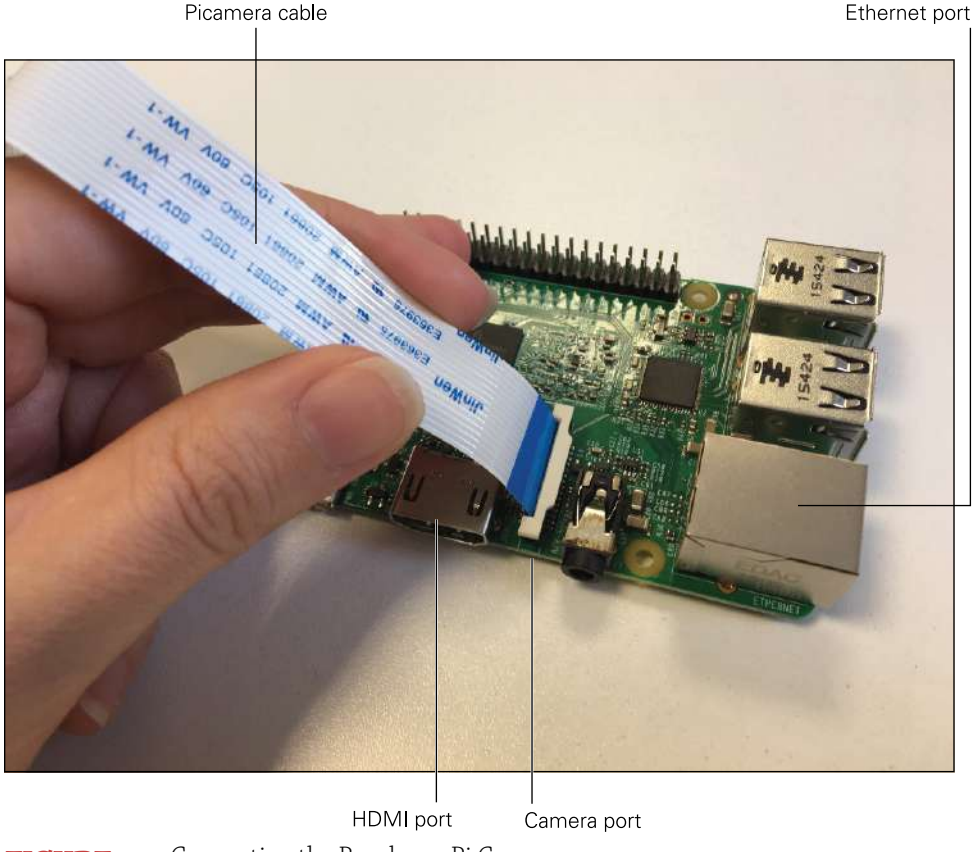


FIGURE 9-2 Connecting the Raspberry Pi Camera

Programming the Picamera with Python

Use the following steps to write a Python program to capture a picture using your Picamera:

1. Open Python 3 (IDLE) and select File➪New File to create a blank text editor window to write your Python code to control the camera.
2. Type the following code into your text editor window:

```
from picamera import PiCamera
import time
camera = PiCamera()
```

3. Write the program that captures an image. Enter the following code:

```
camera.start_preview()
time.sleep(5)
camera.capture("/home/pi/Pictures/image.jpg")
camera.stop_preview()
```

   The first line will start the camera preview so that you can see the camera image (see Figure 9-3) and store it into the Pictures directory on your Raspberry Pi. The preview will then stop and disappear from your screen.

4. Save your code as `first-camera.py` and then run your code by selecting Run➪Run Module to test that it works. You can see the photo that was captured by navigating to the Pictures folder and opening `image.jpg`.

When saving your Python code as a new `.py` file, avoid using the names of libraries that you have imported. For example, when using the picamera python library do not save any code as `picamera.py` as your code will not work! It is a very common mistake and can be very frustrating when you try to debug your code.

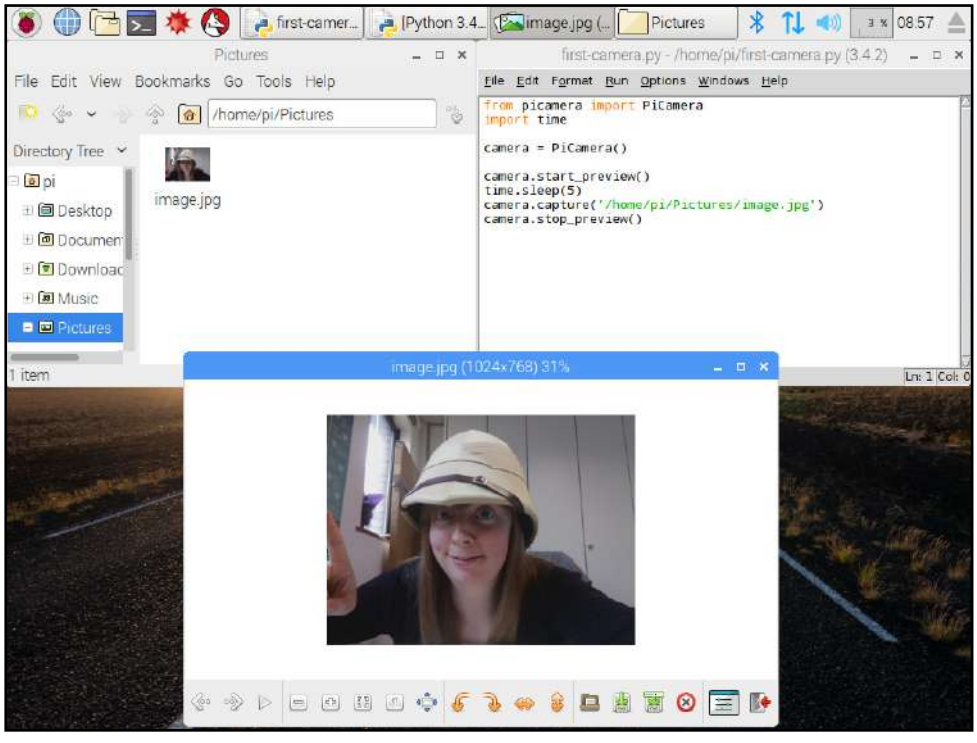


FIGURE 9-3 Capturing an image with Python and the Raspberry Pi Camera

Creating a Time-Lapse Photography Program

To create a time-lapse or stop-frame animation you need to take lots of images and store them before converting them into a video.

Now that you can capture a single image, you can write code to take lots of pictures in a sequence using a loop.

1. Open a new Python3 (IDLE 3) file and save it as `timelapse.py`.
2. Import the modules that you need to use the camera:

```
from picamera import PiCamera
import time
camera = PiCamera()
```

3. Create a variable to number the images that are taken by the camera:

```
img_no = 0
```

4. Create a `while True` loop to wait five seconds, capture an image, and save it with an image number:

```
while True:
  time.sleep(5)
  img_name = 'image{0:02d}.jpg'.format(img_no)
  camera.capture(img_name)
  img_no += 1
```

5. Save your code, click Run, and then click by Run Module to test that your program works. You can check your code against figure 9-4 below. In your `Pictures` directory you should see a new file called `image01.jpg`. If your program is still running, `image02.jpg` should appear after five seconds, and so on through all the pictures.

DIGGING INTO THE CODE

In this code you have used a variable called `img_no` (shorthand for image number) outside the loop. Remember that computer programs work in sequence and when run the value is set to `0`. Then each time around the loop the value `1` is added and stored in the variable `img_no` using the `+=` operator, which you may remember using in Adventure 5 to accumulate health points in your adventure game. This means that each time around the loop the image number changes, which is important for a time lapse because you want every image to be stored in order.

Within the loop, you have created a variable `img_name` in which `img_no` is called to name the picture that is captured. The name is made up of the word *image* and the file extension that identifies it as an image (in this case `.jpg`). Between those strings is the image number value, which has increased by `1`, and `{0:02d}`, which limits the length of the picture number to two digits.

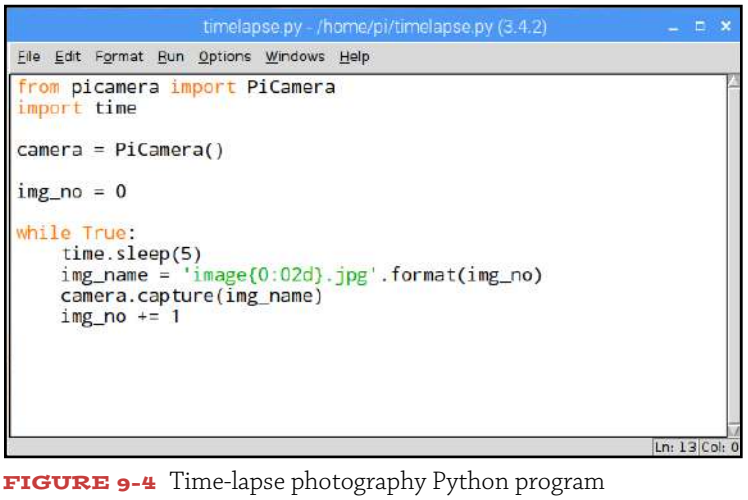


FIGURE 9-4 Time-lapse photography Python program

Mounting Your Camera

To make a smooth time-lapse series, it is important that you place your camera so that it does not move.

1. Find a way to secure your camera. You can purchase Raspberry Pi cases that also hold your camera in place. Check out some of these options:

   - ModMyPi's Pi Camera Box (`https://www.modmypi.com/raspberry-pi/camera/camera-cases/nwazet-pi-camera-box-bundle-case,-lens-and-wall-mount-b-plus`)
   - Multicomp Pi-BLOX Case (`https://www.modmypi.com/raspberry-pi/cases/multicomp/multicomp-pi-blox-case-blue`)

   Alternatively you can mount the camera without a case using products like Pimoroni's Raspberry Pi Camera Mount (`https://shop.pimoroni.com/products/raspberry-pi-camera-mount`), which allows you to connect the camera to any standard tripod.

> You don't need to use a purpose built case or mount for your camera. You could use your maker skills to 3D print a case or mount, or stick it to a wall with adhesive, or cut a hole in a paper cup or even build a mount with LEGO blocks as shown in Figure 9-5.
>
> 
>

2. Find a location to place your camera. Where you place it depends on what you want to create a time-lapse video of. It could be looking out into a garden so that you can see plants growing over time, or pointed towards a building site so that you can watch buildings seemingly appear from nowhere!
3. After you have decided where to put your camera, think about whether you need to amend the timings in your code. One picture taken every five seconds may result in a very long and boring video. To adapt your `time.sleep`, replace `5` with `3600` to take a picture every hour or `1800` to take a picture every 30 minutes.
4. Run your program and leave the camera and Raspberry Pi set up for a few days. Check on it at least once every day to see that the program is still running and capturing images.

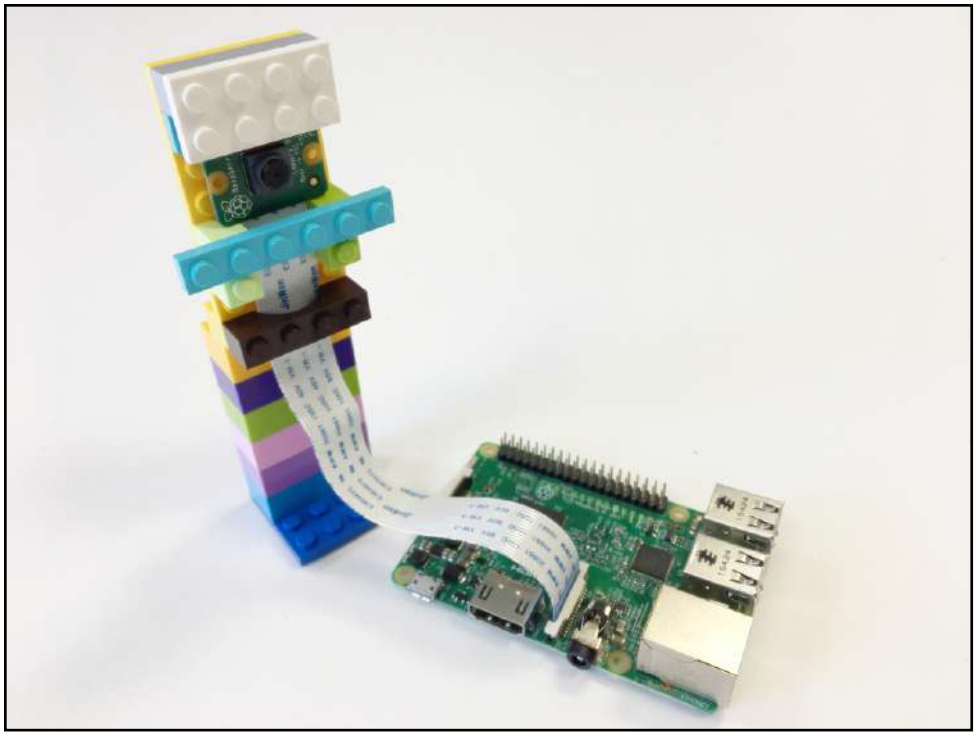


FIGURE 9-5 A Raspberry Pi Camera mount built out of LEGO blocks

Making a Movie of Your Images

Now that you have lots of images you can transform them into a time-lapse video using an application called avconv.

1. Open a terminal window and type the following to check that your system files are up to date and install any required packages:

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

2. Download and install the tools you need to convert images to a movie with the following command:

```
sudo apt-get install libav-tools
```

3. Make sure the application avconv is installed and use it to stitch all your photos together to make a video as shown in Figure 9-6. Type the following command:

```
avconv -r 10 -i image%02d.jpg -qscale 2 timelapse.mp4
```

You are using `image%02.d.jpg` in this command instead of `image{0:02d}.jpg`. This naming convention is a common format that both Python and `avconv` understand, and it means the photos will be passed in to the video in order to create your time-lapse video.

4. Play your completed video by using the application `omxplayer` with the following command:

```
omxplayer timelapse.mp4
```

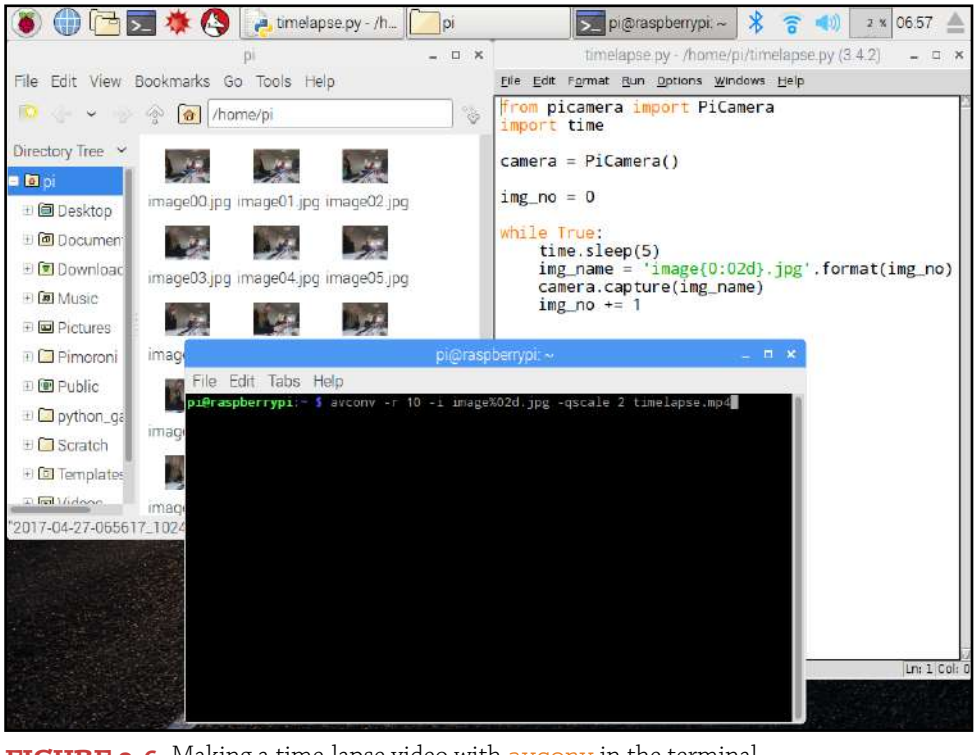


FIGURE 9-6 Making a time-lapse video with `avconv` in the terminal

CHALLENGE

Try creating a stop frame animation with paper drawings, clay figures or even LEGO mini figures to entertain your family and friends. You could even upload them to a video streaming service to share your talents with the world!

Getting Started with the Explorer HAT Pro

Apart from the first Raspberry Pi, all versions of the Raspberry Pi have 40 GPIO pins. HATs like the Explorer HAT Pro and Sense HAT have been designed so that they can fit on top of the 40 pins (see Figure 9-7).

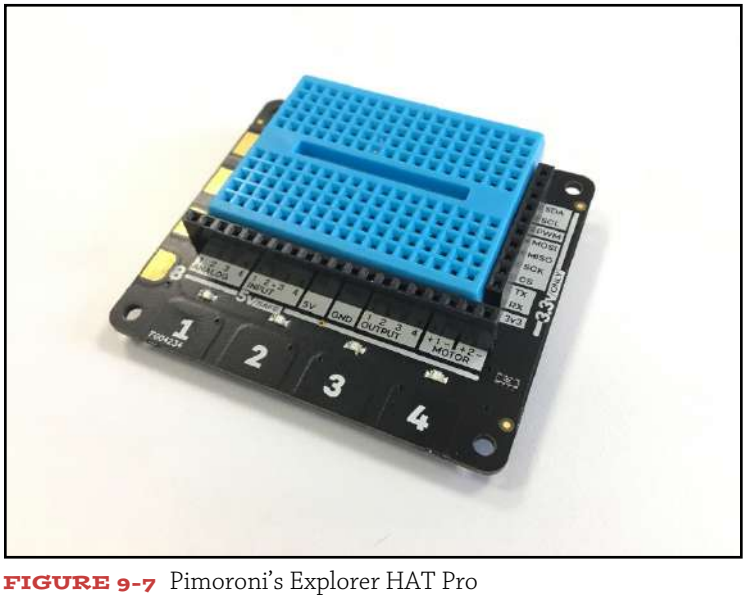


FIGURE 9-7 Pimoroni's Explorer HAT Pro

The Explorer Hat has a number of features that make it a fun accessory for your future adventures:

- Four LEDs (red, yellow, green and blue)
- Eight buttons: four capacitive touch pads (labeled 1, 2, 3 and 4) and four capacitive crocodile clip pads (labeled 5, 6, 7 and 8)

- Two H-bridge motor drivers
- A mini breadboard for individual components

You can learn more about the board at shop.pimoroni.com/products/explorer-hat including information about where you can buy them.

Before programming the extra features that your Explorer HAT Pro brings to your Raspberry Pi, you need to connect the Explorer HAT Pro, download the libraries you need and test that it works.

Connecting the HAT to Your Raspberry Pi

The first step is to connect the Explorer HAT Pro to your Raspberry Pi. Ensure that your Raspberry Pi is disconnected from any power or peripherals to begin.

1. Start by putting the Explorer Hat Pro directly on top of the 40 GPIO pins on your Raspberry Pi so that the numbered pads (1 to 4) are sitting above the HDMI connector. Each pin should connect to the header on the HAT.
2. Push the HAT down to ensure that every pin is covered and that it is connected securely to your Raspberry Pi.
3. Before you connect the power supply to boot to the desktop, connect your Raspberry Pi to a monitor, keyboard and mouse. You need to connect to the Internet to download the Python libraries you need to program the Explorer HAT. Connect it via a network (Ethernet) cable or by connecting to a Wi-Fi network.

Downloading and Installing the Explorer HAT Library

The Python library you need to program the Explorer HAT is not installed on the Raspbian with PIXEL operating system by default. Follow these steps to download and install it onto your Raspberry Pi:

1. Open a Terminal window by clicking the icon in the taskbar.
2. Download the library by typing

```
curl https://get.pimoroni.com/explorerhat | bash
```

This command runs the file, in this case a script or program that contains a series of commands, that you download before you are able to check it. You should use this only if you trust the website that you are downloading from. You can check what the file contains before you run it by typing `curl https://get.pimoroni.com/explorerhat` into a terminal window and reading through the outputted text.

3. When prompted press `Y` on the keyboard and `Enter` to continue with the download.

   It will take a few seconds, depending on your Internet connection speed, for the download to start. The terminal window displays a progress bar formed of dots for you to check how long the download will take. When it is complete you can move on to step 4.

4. In the prompt to continue with the installation, press `Y` and `Enter` to install the Explorer HAT Python library. (See Figure 9-8.)

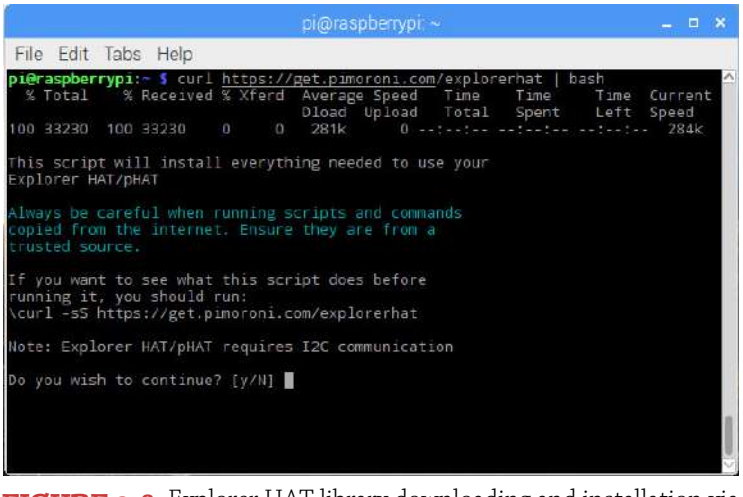


FIGURE 9-8 Explorer HAT library downloading and installation via the Terminal on Raspberry Pi

5. The library also installs some example code that you can use to test that your HAT is working correctly. Navigate to the examples directory by typing

```
cd Pimoroni/explorerhat/examples
```

6. Run the test example by typing

```
python3 test.py
```

You should see the LEDs on the board flash. If you press the touch pads with your finger you see some text outputted to the Terminal window.

You may remember using the `cd` command in previous adventures to change directories or folders from the command line. You use it here to move from `/home/pi` into the directory `/home/pi/Pimoroni/explorerhat/examples` where the `explorerhat` library is located. Remember you can use the `ls` command to list other examples and `python3 name_of_file.py` to run them.

DIGGING INTO THE CODE

Are you wondering about some of the commands and terms in the code? Here's a short breakdown of some of the new bits.

- `curl` is a command-line tool for downloading files from the Internet.
- The pipe character (|) used in the command in step 2 redirects the output from the first command into the second command. In this example, you are downloading a script using `curl` and then processing the download through bash (the terminal), which then runs the script. This makes it possible for you to write just one command instead of a number of separate commands. Many command-line tools are designed to be written in this way to make it easier for the user.

Programming the LEDs

Now that you have the Explorer HAT and the Python library installed, you can program the HAT. Let's start with the red, yellow, green and blue LEDs.

1. Open Python 3 (IDLE) and select File➪New File to create a blank text editor window to write your Python code to control the Explorer HAT LEDs.

   Type the following code into your text editor window:

```
import explorerhat
import time
```

   These two lines import the modules and their functions that you will need to control the explorer hat.

2. Underneath those lines, type the following to turn the red LED on:

```
explorerhat.light.red.on()
```

3. Add a `time.sleep(1)` to keep the LED on.
4. Turn off the red LED by typing

```
explorerhat.light.red.off()
time.sleep(1)
```

You can program the other color LEDs on the Explorer HAT in a similar way:

```
explorerhat.light.green.on()
explorerhat.light.yellow.on()
explorerhat.light.blue.on()
```

You can turn them off with these commands:

```
explorerhat.light.green.off()
explorerhat.light.yellow.off()
explorerhat.light.blue.off()
```

5. Save the file as `explorelights.py` in `Documents`. Then run your program by clicking on Run➪Run Module.

You could place this sequence inside a `while True` loop so that the red LED turns on for one second and then off again repeatedly.

CHALLENGE

Try creating a traffic light sequence using the red, yellow and green LEDs using `.on()`, `time.sleep(1)` and `.off()`. See Figure 9-9 if you need some guidance.

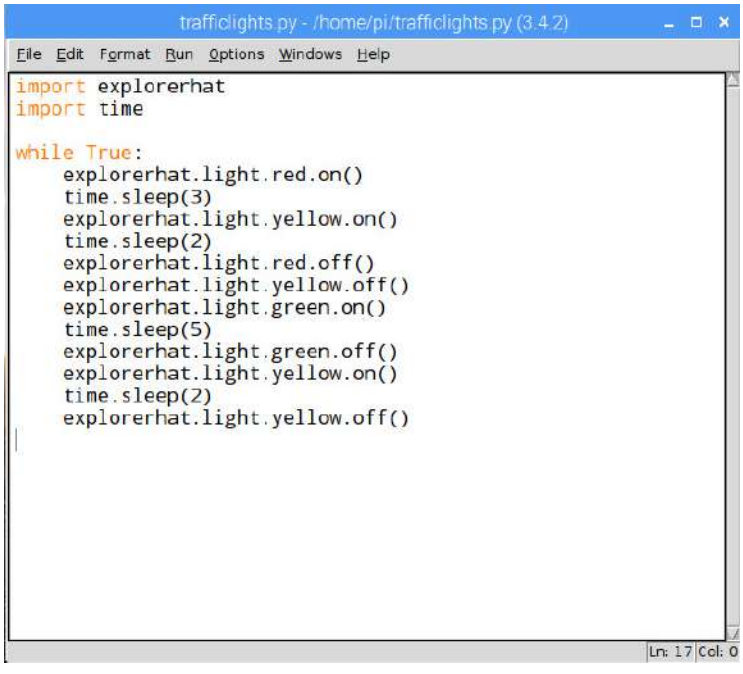


FIGURE 9-9 Creating a traffic light sequence with the Explorer HAT

Programming the Touch Pads

You've programmed outputs on your Explorer HAT, now let's program some inputs, like the touch buttons on the HAT.

1. Open Python 3 (IDLE) and select File➪New File to create a blank text editor window. Type the following code into your text editor window to import the modules you need:

```
import explorerhat
```

2. Create an action that will happen when the button is pressed inside a function. Remember: A function is a chunk of code that does a specific task. In this case, the task is printing a statement to the interpreter window showing which button has been pressed:

```
def button_pressed(channel, event):
  print("Button " + str(channel))
```

3. Call the function in a command like this:

```
explorerhat.touch.pressed(button_pressed)
```

   This line should not be indented, as it is the command to call the function rather than being code inside.

4. Save the file as `explorebutton.py` in `Documents`. Then run your program by selecting Run ➪ Run Module.

5. Press any of the touch pad buttons (numbered between 1 and 8) on the Explorer Hat numbered. Each time you touch one, the number of that button is printed to the interpreter window. Pretty neat!

Creating an Explorer HAT Pro Disco Trigger Trap

Trigger traps are fun to create because you can use them to surprise friends and family, especially if it is Halloween. You can even use them as a way of detecting when someone goes into your room. In this project, you use the Explorer HAT Pro to detect a button press that triggers lights and sounds before capturing an image of the intruder.

For this project, you need your Raspberry Pi, Explorer HAT Pro and the following items:

- A crocodile clip cable
- Aluminum foil
- A speaker
- Raspberry Pi camera

Connect your camera before putting your Explorer HAT Pro onto the GPIO pins, as shown in Figure 9-10.

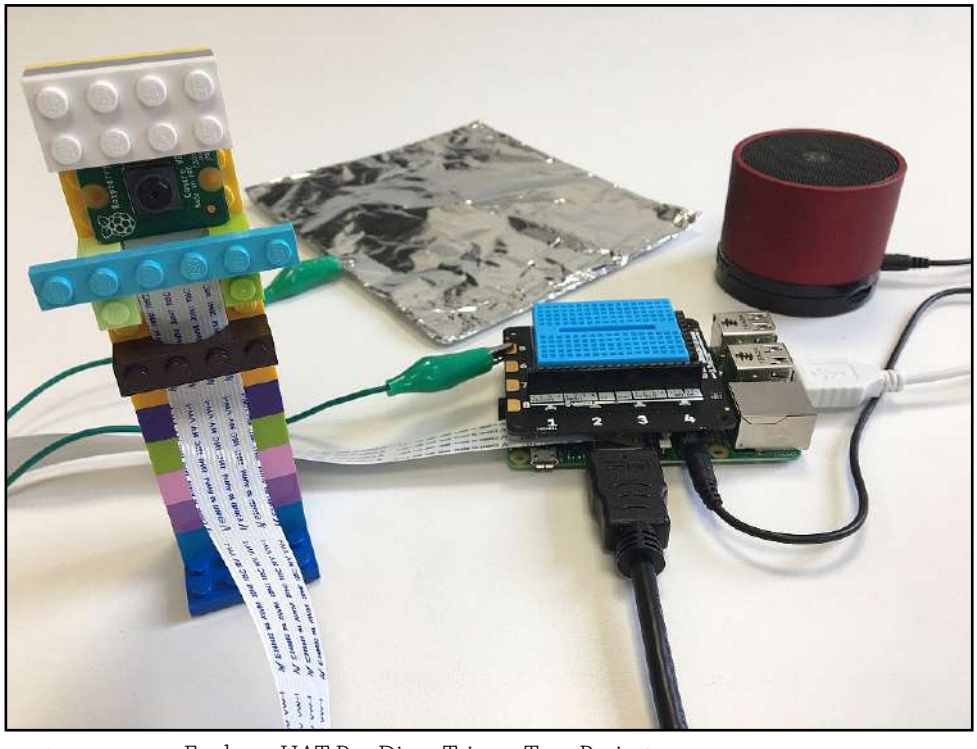


FIGURE 9-10 Explorer HAT Pro Disco Trigger Trap Project

Creating the Disco Trigger Trap Python Code

You begin this project by writing the trigger trap disco light and sound program.

1. Open Python 3 (IDLE) and select File➪New File to create a blank text editor window. Type the following code into your text editor window to import the modules you need:

```
import explorerhat, time, random, os, sys
from picamera import PiCamera
```

2. Set up the camera and set the image resolution using the following code:

```
camera = PiCamera()
camera.resolution = (1024, 768)
img_no = 1
```

   As you are planning on capturing lots of pictures over a period of time, you don't want to overwrite any of the images, so create a variable containing the value 1 outside of the program loop (as you did earlier in the chapter) so that each time an image is captured, it is stored with a unique filename.

3. Create a list of the different colour LEDs on the Explorer HAT. (Just like the inventory program you wrote in Adventure 5.)

```
colours = [explorerhat.light.red,
explorerhat.light.blue,
explorerhat.light.green,
explorerhat.light.yellow]
```

Here you have used the variable called `colours` to store your list of items.

4. Create a function that contains a `for` loop. Each time around the loop a colour LED is selected from the list and turns it on for 0.1 second; then the LED turns off.

```
def disco():
  for i in range(25):
    result = random.choice(colours)
    result.on()
    time.sleep(0.1)
    result.off()
```

Unlike a `while True:` loop, the `for` loop only repeats 25 times. You can change the value to repeat any number of times that you like, but remember the program will take a long time to move on if the sequence is too long!

5. Create another function for the button press to trigger the disco lights:

```
def button_pressed(channel, event):
  print("Button " +str(channel))
  img_name = 'image{0:02d}.jpg'.format(img_no)
  camera.capture(img_name)
  disco()
  os.system('omxplayer /opt/sonic-↩
    pi/etc/samples/loop_amen_full.flac')
```

This function triggers a print statement, captures an image and stores it, calls the disco function so the LED lights on the explorer HAT light up randomly 25 times and plays a sample from sonic pi. If you ran your program now nothing would happen, because the code is inside a function that needs to be called.

6. Under the code you added in step 5, add a `while True: loop` and a conditional to call your button press function:

```
while True:
  if explorerhat.touch.pressed(button_pressed):
    img_no += 1
    time.sleep(5)
```

7. Save the file as `triggertrap.py` in `Documents`. Then run your program by selecting Run➪Run Module. Press one of the capacitive touch buttons on the explorer HAT to trigger the disco.

Making the Aluminum Foil Trap

The button press you have programmed is waiting to be pressed by the capacitive touch pads on the Explorer HAT. It would be difficult for someone to accidently trigger those small buttons, so you need to make an aluminum foil trap to place on the floor.

1. Take a crocodile clip cable and connect one end to a capacitive touch button on the Explorer HAT. Use one of the buttons numbered between 5 and 8.
2. Use the aluminum to create a rectangle shape about 20cm by 30cm.
3. Clip the other end of the crocodile clip cable to the aluminum foil rectangle.
4. Run your program and check that the disco starts when you touch the foil trap.
5. Place the foil trap where no one can see it but it still works. Near a door to a room is a good spot. Make sure that you point your camera towards the same place too.

> You can purchase longer cables for your Raspberry Pi camera to extend it. This is useful for projects where you want to position your camera away from the project build (as with the trigger trap). The camera cables come in sizes ranging from 50mm to 457mm.
>
> 
>

Getting Started with the Sense HAT

The Raspberry Pi Sense HAT was created as part of British ESA Astronaut Tim Peake's mission to the International Space Station (ISS) in December 2015. (See Figure 9-11.) Two special Raspberry Pis with sense HATs and camera accessories were transported to the ISS so that space experiments coded by school children could be run by the astronauts. (The special Raspberry Pi units are called Astro Pi units.) You can find out more about these missions on the official Astro Pi website (`https://astro-pi.org`).

FIGURE 9-11 British ESA Astronaut Tim Peake holding an Astro Pi onboard the ISS
Image courtesy of ESA CC-BY-SA 3.0 IGO

The Sense HAT has a number of sensors built into it, which gives the board its name. They include

- Temperature and humidity sensors
- Biometric pressure sensor
- Movement sensors, in particular accelerometer, gyroscope and magnetometer sensor

It also has an 8x8 LED matrix and a joystick.

To learn more about each individual sensor, visit the Astro Pi website hardware page `https://astro-pi.org/about/hardware/sense-hat/`.

You connect the Sense HAT to the GPIO pins on a Raspberry Pi in the same way as the Explorer HAT. Make sure that the power is disconnected before you connect the Sense HAT, and be careful to cover all the pins.

If you do not have access to the Sense HAT hardware then you can try these exercises using the Sense HAT emulator either online (`https://trinket.io/sense-hat`) or on the Raspberry Pi. You can find it on the Programming menu of Raspbian.

Programming the LED Matrix with Python

One of the most exciting parts of the Sense Hat is the LED matrix. There are 64 neopixels (a type of colour LED) that you can program to do all sorts of things, like scroll text and make images! In this activity you are going to scroll a message:

1. Open Python 3 (IDLE) and select File➪New File to create a blank text editor window. and Type the following code into your text editor window to import the module you need:

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

2. Underneath, write the command to scroll text on the LED matrix with a string of text. This is similar to writing a print statement only rather than appearing on the screen it scrolls across the sense HAT:

```
sense.show_message("Hello Sense HAT!")
```

3. Save the file as `sensehatimages.py` in `Documents`. Then run your program by selecting Run➪Run Module. Watch your Sense HAT come to life with your message.

Programming the Sensors to Find Out the Current Temperature

Scrolling messages on the LED matrix of the Sense HAT is a useful feature to report readings from the sensors on the HAT. For example, you can find out the current temperature and scroll the value on the matrix by following these steps:

1. Underneath the first two lines of your program, create a variable to store the current temperature sensor reading with the following line:

```
temp = sense.get_temperature()
```

2. The temperature sensor is so accurate that it reports its value to 14 decimal places, which scroll across the LED matrix. To round the value up to one decimal place, use this command on the next line:

```
temp = round(temp, 1)
```

3. Create a variable to store the contents of your message you want to be displayed as a string, like this:

```
msg = "Temperature = {0}".format(temp)
```

4. Adapt `sense.show_message` line to read:

```
sense.show_message(msg)
```

5. Save and run your file. You see the current temperature scroll across the matrix.

You can program some of the other sensors in a similar way, like humidity and pressure:

```
sense.get_humidity()
```

```
sense.get_pressure()
```

Creating Pixel Art

Each LED on the Sense HAT can be programmed to show a different colour. They work by mixing red, green and blue to make all the colours of the rainbow. By setting the colour of each individual LED you can create pixel images made of light.

1. Design your pixel art image. The best way to do this is to use squared or graph paper and some colouring pencils. Map out eight squares by eight squares on the paper to represent the Sense HAT matrix.
2. Draw a design by colouring in some of the individual squares, as shown in Figure 9-12.

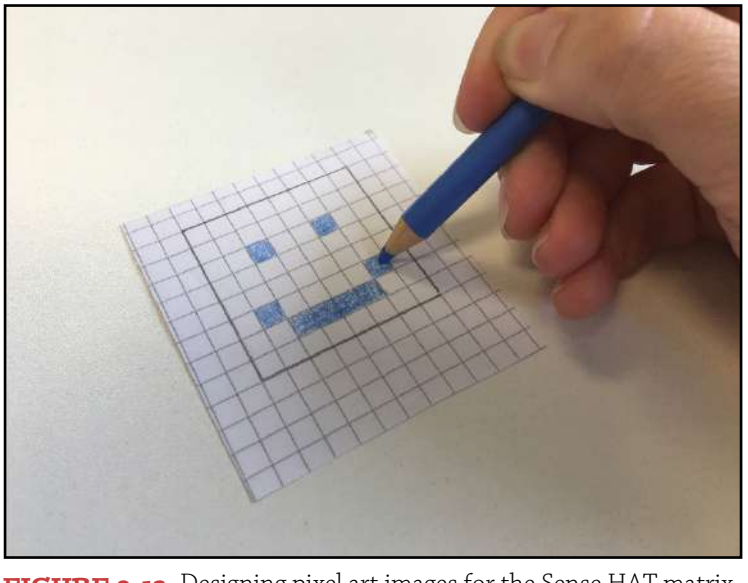


FIGURE 9-12 Designing pixel art images for the Sense HAT matrix

3. Use a pen to label each of the squares with a letter that represents the colour contained inside. For example, if the cell is coloured in yellow, then write a y in the same cell.

4. Open Python 3 (IDLE) and select File➪New File to create a blank text editor window. Type the following code into your text editor window to import the module you need:

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

5. Create variables to store the RGB values of the colours of your drawing, like so:

```
b = (0, 0, 255)
e = (0, 0, 0)
```

DIGGING INTO THE CODE

Red, green and blue can be mixed together by lights, such as LEDs, to create lots of other colours. Each LED on the Sense HAT matrix can be set to a different colour using RGB colour values like this:

```
r = (255, 0, 0)
p = (128, 0, 128)
```

The first colour is red, and we know this because contained within the brackets are three values, one for red, one for green and one for blue. The red value is the highest possible value of 255, and the other two values are the lowest possible value of zero. So red is not being mixed with any other colour.

The second value is purple, which is a mix of some red, no green and some blue. You can continue to mix colours in this way to change the colour of the LEDs on the Sense Hat matrix. You can use this website to mix RGB colours: https://www.w3schools.com/colors/colors_rgb.asp

6. Create a list with a variable name to represent your image, and type each letter on the first row of your drawing separated by commas like this:

```
happy = [ e, e, e, e, e, e, e, e,
```

7. Move onto the next line of your code and type the next row of letters, then the next until you have written out your image as code like this:

```
happy = [
         e, e, e, e, e, e, e, e,
         e, e, e, e, e, e, e, e,
         e, e, b, e, e, b, e, e,
         e, e, e, e, e, e, e, e,
         e, b, e, e, e, e, b, e,
         e, e, b, b, b, b, e, e,
         e, e, e, e, e, e, e, e
         ]
```

This list now has a colour set for every pixel in the LED matrix. All that is needed is to call the list when setting the pixels; use the line

```
sense.set_pixels(happy)
```

8. Save your file as `pixel-art.py` and run your file. Your LED matrix shows your image with its lights.

Creating a Sense HAT Desk Thermometer

Displaying sensor data can be really useful. For example, if you know the temperature before you leave your home, you can decide whether to wear a hat, gloves and a scarf or a t-shirt and shorts!

1. Open Python 3 (IDLE) and select File➪New File to create a blank text editor window. Type the following code into your text editor window to import the module you need:

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

2. Create variables for each colour that you use in your drawing.

```
e = (0, 0, 0)
b = (0, 0, 255)
r = (255, 0, 0)
y = (255, 255, 0)
s = (192, 192, 192)
```

3. Create three different pixel images that represent different temperatures like a sun, a snowflake and something in between! Make a list with a memorable variable name for each, like this:

```
cold = [
        e, b, e, b, e, b, e, b,
        b, b, b, s, s, b, b, e,
        e, b, s, b, b, s, b, b,
        b, s, b, s, s, b, s, e,
        e, s, b, s, s, b, s, b,
        b, b, s, b, b, s, b, e,
        e, b, b, s, s, b, b, b,
        b, e, b, e, b, e, b, e
        ]

mild = [
        y, b, e, b, e, b, e, y,
        b, y, b, y, y, b, y, e,
        e, b, y, b, b, y, b, b,
        b, y, b, y, y, b, y, e,
        e, y, b, y, y, b, y, b,
        b, b, y, b, b, y, b, e,
        e, y, b, y, y, b, y, b,
        y, e, b, e, b, e, b, y
        ]

hot = [
        y, y, e, y, e, y, e, y,
        y, y, r, y, y, r, y, e,
        e, r, y, r, r, y, r, y,
        y, y, r, y, y, r, y, e,
        e, y, r, y, y, r, y, y,
        y, r, y, r, r, y, r, e,
        e, y, r, y, y, r, y, y,
        y, e, y, e, y, e, y, y
        ]
```

4. Create a loop to get the current temperature, round that value to one decimal place and store it as a string in a variable called `msg`.

```
while True:
  temp = sense.get_temperature()
  temp = round(temp, 1)
  msg = "Temp = {0} ".format(temp)
```

5. Create some conditions: If the temperature is between –10 and 15 degrees C then show the cold pixel art. If the temperature is between 15 and 20 degrees C, show the mild image. If the temperature is between 20 and 50 degrees C, show the hot image:

```
  if temp > -10 and temp < 15:
    sense.set_pixels(cold)
```

```
  elif temp > 20 and temp < 50:
    sense.set_pixels(hot)
  elif temp > 15 and temp < 20:
    sense.set_pixels(mild)
```

6. Still inside the `while True` loop, add a 10 second sleep before displaying the temperature message. Figure 9-13 shows what the looping code should look like.

```
  time.sleep(10)
  sense.show_message(msg, scroll_speed=0.05)
```

7. Save your code as `desk-thermometer.py` and run it. Each time around the loop the temperature is scrolled across the LED matrix and displays the right image for that temperature.

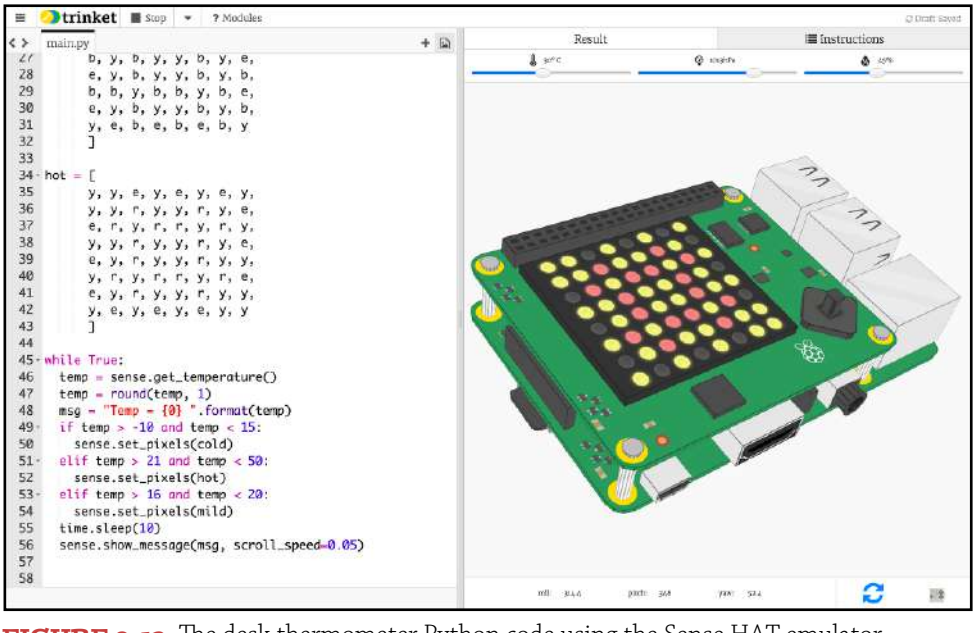


FIGURE 9-13 The desk thermometer Python code using the Sense HAT emulator

Further Adventures with Cameras and HATs

Adding accessories like HATs and a camera can expand the Raspberry Pi hardware allowing you to build amazing projects. There are lots of different add-ons to choose from, and once you start experimenting, your collection will grow!

Now that you have played around with these accessories you may want to learn more. Here are some resources to get you started:

- If you are interested in wildlife, then check out NatureBytes a kit to build a wildlife trigger trap kit http://naturebytes.org/.
- Create an infrared bird box to help support the bird population in your area with this official Raspberry Pi learning resource https://www.raspberrypi.org/learning/infrared-bird-box/.
- Make a Minecraft Thermometer with the Explorer HAT using this Pimoroni tutorial https://learn.pimoroni.com/tutorial/explorer-hat/making-a-minecraft-thermometer.
- Don't have a Sense HAT, don't worry; you can use the Sense Hat emulator on the Raspberry Pi, which you can find in the Programming menu or online at https://trinket.io/sense-hat.
- For lots of Sense HAT projects, you can't go wrong with the free MagPi Essentials book, *Experimenting with the Sense HAT,* which you can download from https://www.raspberrypi.org/magpi-issues/Essentials_SenseHAT_v1.pdf.

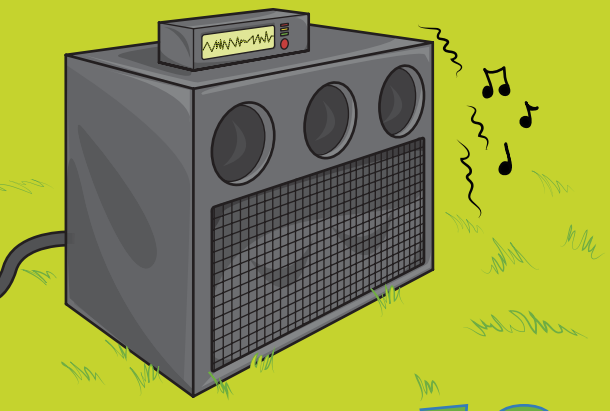
Accessories Command Quick Reference Table

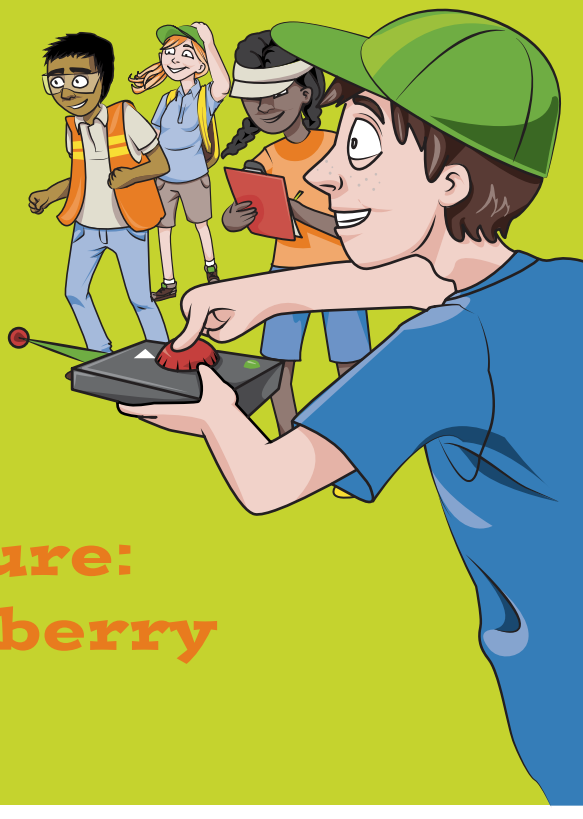
Command	Description
`From picamera import PiCamera`	Imports the picamera modules
`camera.start_preview()` `camera.stop_preview()`	Starts the camera preview Stops the camera preview
`camera.capture('/home/pi/Pictures/image.jpg')`	Captures a jpeg image and stores it in the Pictures directory
`avconv -r 10 -i image%02d.jpg -qscale 2 timelapse.mp4`	Converts a series of images into a movie using `avconv`
`import explorerhat`	Imports the Explorer HAT module
`explorerhat.light.red.on()` `explorerhat.light.red.off()`	Switches the red LED on the Explorer HAT on and then off
`explorerhat.touch.pressed(button_pressed)`	Posts a message to chat in Minecraft Pi
`from sense_hat import SenseHat` `sense = SenseHat()`	Imports the Sense HAT modules
`sense.show_message("Hello Sense HAT!")`	Scrolls a test string across the LED matrix on the Sense HAT
`temp = sense.get_temperature()`	Gets the current temperature from the temperature sensor on the Sense HAT
`sense.set_pixels(happy)`	Sets all the pixels on the LED matrix of the Sense HAT at the same time

Achievement Unlocked: Experimented with cameras and HATs

In the Next Adventure

The final adventure in this book is a humongous Raspberry Pi project. It draws on all the computational skills you have learned on your Pi journey so far and walks you through making a Pi jukebox with an LCD display, Play, Stop and Skip buttons and a box! The adventure involves acquiring extra parts for your Raspberry Pi, such as an LCD screen. The project may seem intimidating at first, but as with all challenging endeavors, you will feel triumphant when you have completed it.





Adventure 10

The Big Adventure: Building a Raspberry Pi Jukebox

ONE OF THE Raspberry Pi's special qualities is the fact that you can transform it into a dedicated device of its own. In this adventure, you transform your Raspberry Pi into a jukebox, complete with buttons to select and play tracks, and an LCD screen to display the song names. Figure 10-1 shows my version of the completed project.

We call the act of making something with technology *digital making*. A build project like this one gives you the opportunity to be creative and innovative through messing, building, designing, hacking and making. It is also a lot of fun.

I hope this adventure will whet your appetite for more big Raspberry Pi projects. If you want to continue your Raspberry Pi journey, I've included a few resources at the end of the adventure to give you some ideas. The more you work with the Raspberry Pi and learn about what it can do, the more you'll come up with your own big project ideas!

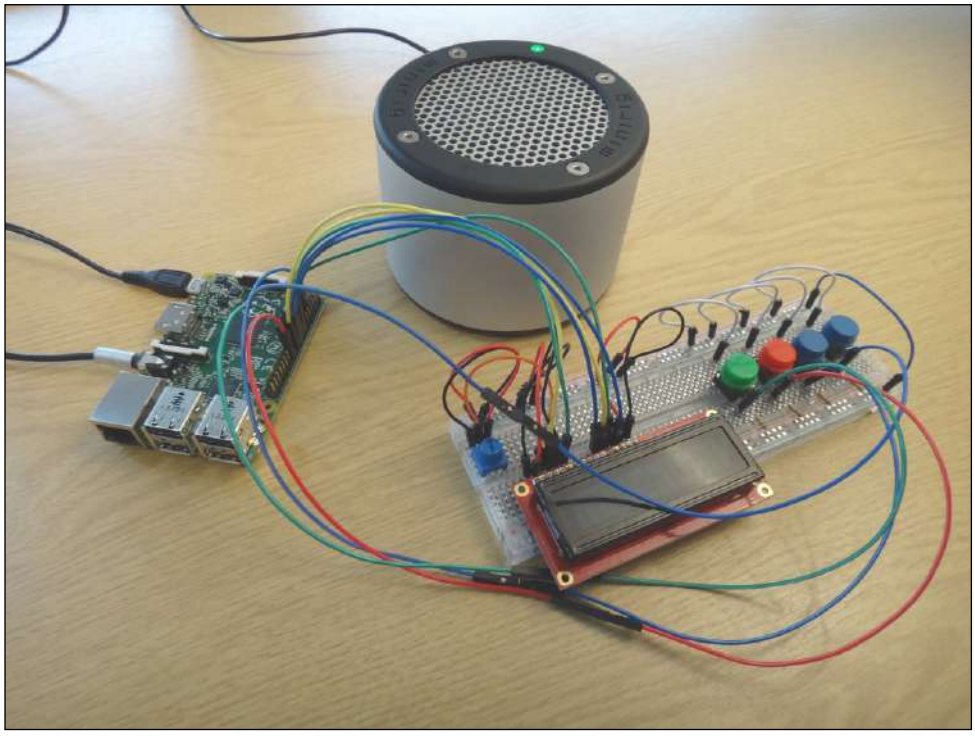


FIGURE 10-1 The completed Adventures in Raspberry Pi Big Jukebox Project!

An Overview of the Jukebox Project

This final adventure is slightly more complicated than the other projects in this book. A big project like this is great for drawing together many of the skills you have learned in previous adventures. Because this project is complex, I've broken the instructions into four parts. But before you dive in to the details, here's a short road map of what you'll be doing.

- In Part One, you use Python to create the LCD screen for the jukebox.
- In Part Two, you add the software to download and play MP3 files.
- In Part Three, you use the GPIO pins to connect buttons to your circuit and write a program so that you can use the buttons to play, pause and skip tracks.
- In Part Four, you write the code to make the LCD screen information about the MP3 files that are being played.

Finally, you may want to finish up your project by designing a box for your Pi jukebox, to conceal the wiring and circuitry, make it more user-friendly and enhance the way it looks.

All the completed code files for this project are available for download from the companion website at www.wiley.com/go/adventuresinrp3E. As I've said in other adventures, you learn far more by following the instructions in this book, typing in the code yourself and figuring out how to fix any problems. However, if you have difficulty getting something to work, you might want to compare your code to the download files to check whether you've missed something.

What You Will Need

To complete this big project, you need your Raspberry Pi and a number of extra components. You can purchase all the components from online electronic retailers, and none of them require soldering. Here's what you need:

- Your Raspberry Pi and peripherals including an SD card with an up-to-date Raspbian with PIXEL image installed (see Adventure 1)
- A small speaker that uses the headphone/speaker port on your Pi (like the speaker in Figure 10-1)
- A full-sized breadboard
- A 16 x 2 character 3.3v parallel **liquid crystal display** (**LCD**)
- A 10K **potentiometer**
- Four buttons, like the one used in Adventure 8 to turn on and off an LED
- Four 10k ohm resistors
- Solderless headers
- Female-to-male jumper cables and male-to-male jumper cables
- A Raspberry Pi GPIO pin labeller
- A cardboard box
- Some decorations or paint to make your jukebox look cool

A full list of products used in this adventure and where they can be purchased can be found on the *Adventures in Raspberry Pi* website at www.wiley.com/go/adventuresinrp3E.

An **LCD** (**liquid crystal display**) is an electronic display, usually quite thin and flat, that is typically used in digital calculators and digital watches to display information like the time. In this project, your LCD screen shows the names of the songs playing on the jukebox—you don't need a monitor for the Pi jukebox.

A **potentiometer** is a variable resistor. In this project, it enables you to adjust the contrast on the display to make it easier to read what is on the LCD screen by turning an adjuster wheel.

Many big electronic projects require you to solder parts together. If you have spent money on expensive components this can be a daunting task, even for proficient tinkerers. It can also be very painful should you accidentally burn yourself with a soldering iron. Luckily, items like breadboards, jumper cables and solderless headers make it possible to make cool electronic projects without the need to solder. If there is an adult in your household or at school who is nifty with soldering tools and has some experience, however, it may be a good idea to ask them to help you solder headers onto any expensive components to save you from fiddling around with solderless headers in this tutorial.

Part One: Creating the LCD Screen

In the first part of this project, you need to assemble the electronic components for your jukebox. This project involves many more cables and components than previous adventures, so you should be extra careful about checking your work against the figures and wiring diagrams. Then, after you have set up the electronics, you will download some files that you'll use later to be able to display text on the LCD screen.

Preparing the LCD Screen by Adding Headers

When you buy or receive your LCD screen, you may find that the header pins that you need in order to connect the LCD screen onto a breadboard are not included with it. In this case, you may need to ask an adult to solder the headers on for you; or, alternatively, you can use solderless headers, which need to be pushed all the way into the holes to make a good connection. Although it may seem easier to use solderless headers, they can be quite difficult to push into place, and you might need a lot of patience

to do it. I find that wiggling the headers in small groups of five or six at a time, while applying pressure, is the best way to add them to the LCD screen.

Mounting the LCD Screen and Wiring Up the Breadboard

Follow these steps to set up the LCD screen. Refer to Figure 10-2 for additional guidance as you work.

1. Lay your full breadboard out lengthways in front of you so that the longest side runs parallel to the edge of the table directly in front of you. Push your prepared LCD screen's header pins into the holes, starting from C5 and going all the way to C21.
2. Add the potentiometer into pins F1 to F3 above the left end of the LCD screen. You will be using the potentiometer to adjust the contrast of the LCD screen.
3. Place your unplugged Raspberry Pi next to the breadboard (see Figure 10-2) and add the Raspberry Pi GPIO pin labeller (`http://rasp.io/portsplus`) for your Raspberry Pi board revision over the top of the GPIO pins (you'll remember the pin labeller from Adventure 8). You will be connecting quite a lot of jumper cables from the LCD screen to your Raspberry Pi, and you'll find it easier with the pin labeller in place as a guide.
4. Wire the LCD screen using the following guidelines so you can send data to it. Start with pin 1 (the far left) of the LCD screen and attach the cable to the correct destination (use the diagram in Figure 10-2 to help you):
    - Pin 1 of the LCD goes to the ground or negative blue strip of the breadboard (black male-to-male cable on diagram).
    - Pin 2 of the LCD goes to the 3.3v or positive red strip of the breadboard (red male-to-male cable on diagram).
    - Pin 3 (Vo) connects to the middle of the potentiometer (orange male-to-male on diagram). Note that the potentiometer has three pins; the orange wire should be placed in a breadboard slot above the potentiometer in the middle.
    - Pin 4 (RS) connects to the Raspberry Pi GPIO 25 (yellow male-to-female cable).
    - Pin 5 (RW) goes to the ground or negative of the breadboard (black male-to-male cable).

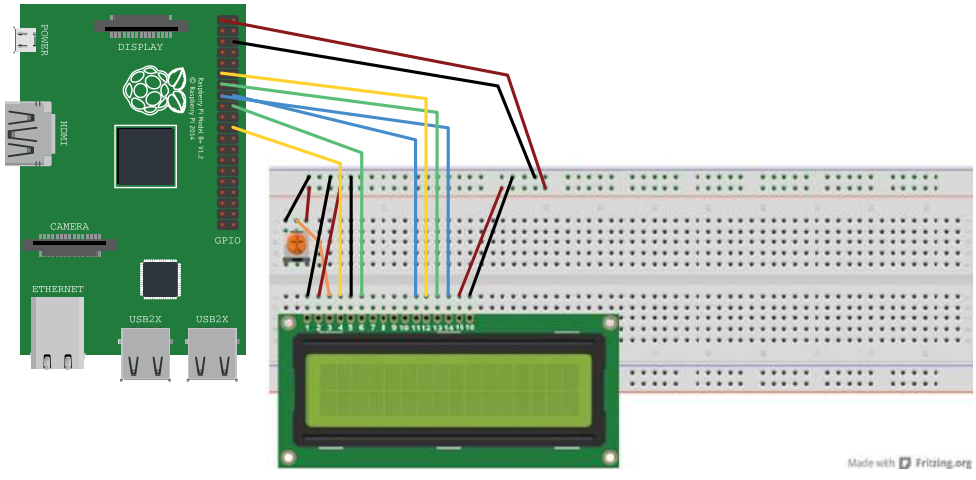


FIGURE 10-2 Circuit diagram for wiring the LCD screen and potentiometer

- Pin 6 (EN) connects to the Raspberry Pi GPIO 24 (green male-to-female cable).
- Skip LCD Pins 7, 8, 9 and 10.
- Pin 11 (D4) connects to the Raspberry Pi GPIO 23 (blue male-to-female cable).
- Pin 12 (D5) connects to the Raspberry Pi GPIO 17 (yellow male-to-female cable).
- Pin 13 (D6) connects to the Raspberry Pi GPIO 27 (green male-to-female cable).
- Pin 14 (D7) connects to the Raspberry Pi GPIO 22 (blue male-to-female cable).
- Pin 15 (LED +) goes to the positive red strip on the breadboard (red male-to-male cable).
- Pin 16 (LED –) goes to the ground black strip on the breadboard or GND (black male-to-male cable).
- Connect the red strip on the breadboard to the Raspberry Pi 3.3V (red male-to-female cable).
- Connect the ground strip on the breadboard to the Raspberry Pi ground or GND (black male-to-female cable).

- On the potentiometer, connect the left pin to the ground or negative blue strip of the breadboard (black male-to-male cable), and the right pin to the 3.3v or positive red strip of the breadboard (red male-to-male cable).

5. Double-check the diagram (Figure 10-2) and picture (Figure 10-3) to check that you have wired up your LCD and potentiometer correctly. When you are happy that your wiring is correct, plug your Raspberry Pi, complete with SD card, into a monitor, keyboard, mouse and, finally, power supply. You need to connect to the Internet either by plugging in a network (Ethernet) cable or by connecting to Wi-Fi.

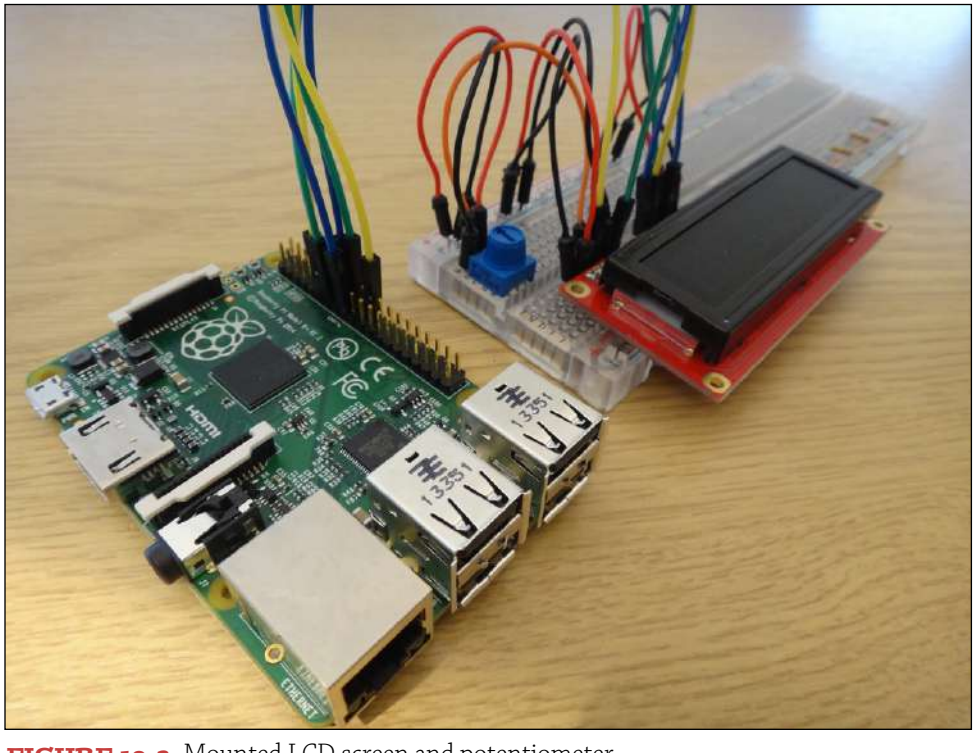


FIGURE 10-3 Mounted LCD screen and potentiometer

Remember, if you connect GPIO pins incorrectly you could damage your Raspberry Pi. Always double-check your wiring before you connect it to a power supply.

6. The LCD screen should light up. If it does not, return to the wiring diagram and instructions and check your work. Twist the potentiometer until you see the first line of the LCD fill with boxes.

Adding Scripts to Drive the LCD Screen

Next, you need to download the Python code required to display information on the LCD screen. Make sure that your Pi is connected to the Internet either by an Ethernet cable or by Wi-Fi, so that you can download the files you need.

1. After booting your Pi, open a Terminal window, type the following code and press Enter:

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

   After you have checked that all your application packages are up to date, you can make a copy of the Adafruit Raspberry Pi Python files that I've modified, by typing:

```
git clone https://github.com/MissPhilbin/Adventure_10.git
```

2. After the code is copied, navigate to the directory or folder containing the Python code for a 16 x 2 LCD screen by typing the following into the command line of a Terminal window and pressing Enter:

```
cd Adventure_10
```

3. Now run the Python code by typing the following command and then pressing Enter (Figure 10-4):

```
python3 Adafruit_CharLCD.py
```

   You should see the following appear on your LCD screen:

```
This is a test!
```

4. As you twist the potentiometer back and forth, the letters fade and become more vibrant.

5. Copy the files into your `Documents` directory where you will be saving your jukebox Python program by typing:

```
cp Adafruit_CharLCD.py /home/pi/Documents/
```

Later in this project you will use these files to help you write a program that displays MP3 track information onto the LCD screen of your jukebox.

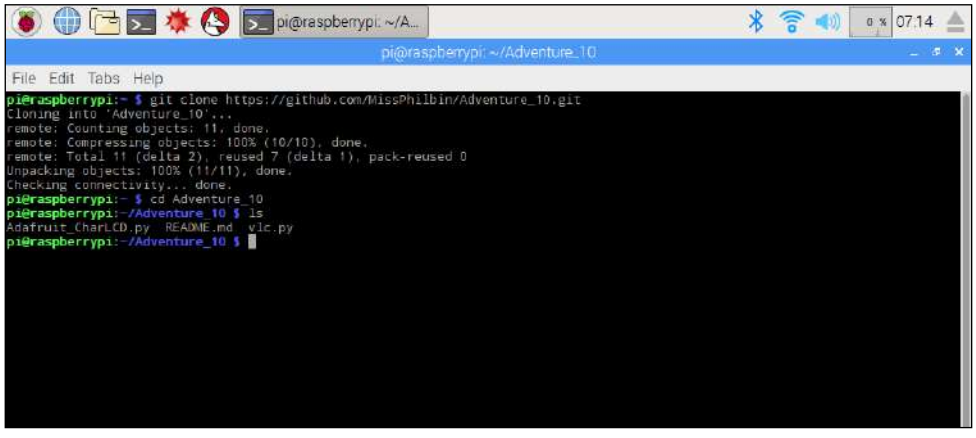


FIGURE 10-4 Downloading the modified `Adafruit_CharLCD.py`

Part Two: Downloading and Playing MP3s

Now that the LCD screen is set up, it's time to download and play some music files on your Raspberry Pi. To play music files on your Raspberry Pi, you need to download and install a media player and test that it works. Once you have installed the player and some music to listen to, you write a program that lets you create a playlist and shuffle or skip tracks.

Installing a Media Player and Getting Music Files

In Adventure 2 you downloaded and installed a media player called `vlc`, which you will be able to use to play music files in this project. You may want to double-check that it is installed by typing this command in a Terminal window, which downloads and installs the application:

```
sudo apt-get install vlc
```

If `vlc` is already installed then you see the message `vlc is already the newest version`; otherwise the installation will continue.

The `vlc` application is a media player. It enables you to play different types of media, such as videos and music, from the command line. This is helpful for your jukebox project as the plan is to control the player from the LCD screen using input buttons that you will add later in this project.

Next, you need some music files to play. You may have some MP3s on a desktop or laptop computer that you can transfer to your Pi using a portable storage device like a

USB memory stick. Alternatively, you could download an album from the Free Music Archive using a web browser on the Raspberry Pi. To do this, your Raspberry Pi needs to be connected to the Internet, either through an Ethernet cable or to a Wi-Fi network. Use the following steps to get music from the Free Music Archive:

1. Open the web browser by clicking the web browser icon on the taskbar or by clicking on the main menu and selecting Internet ➪ Chromium Web Browser. In the URL address bar of the web browser, type the following URL and press Enter (Figure 10-5):

```
freemusicarchive.org
```

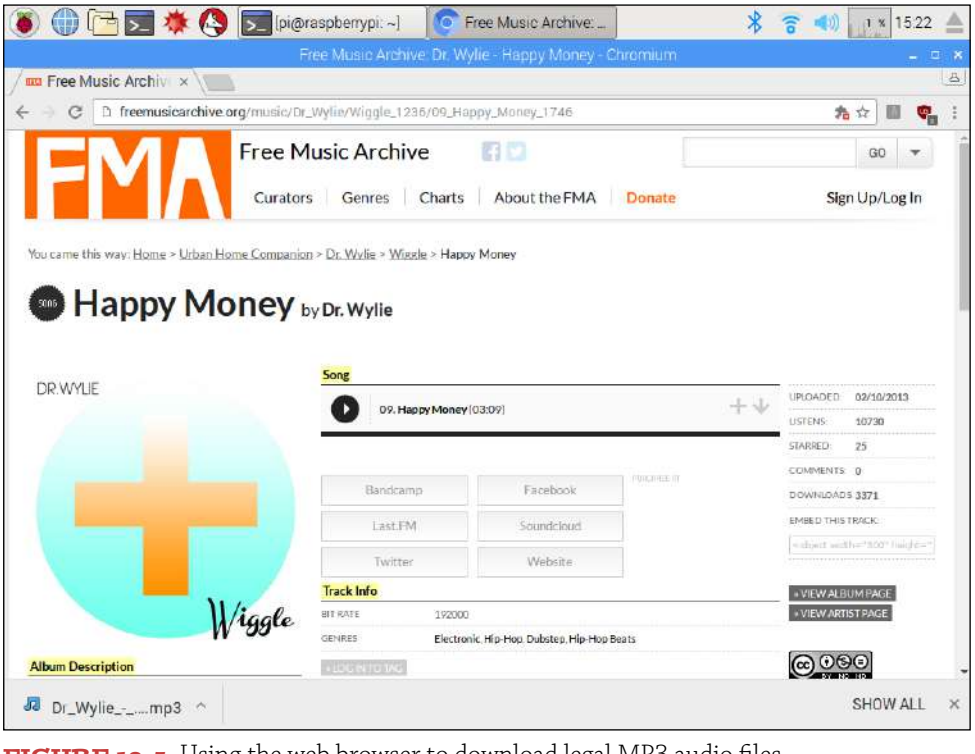


FIGURE 10-5 Using the web browser to download legal MP3 audio files

The Free Music Archive is home to high-quality audio files that you are free to download and listen to. The content is cleared for certain types of use by the artists and is not prohibited from download by copyright laws.

2. Find some music that you like by browsing the website. When you have located a song that you like and want to download, click the download arrow next to the name of the song and the MP3 file starts to download. You can check its progress by looking at the progress bar at the bottom of the web browser window. When the bar is full, the download is complete. This file downloads to your `/home/pi/Downloads` directory.

3. Once the download is complete, test to see if `vlc` can play the MP3 music file. In a Terminal window, type `cvlc` followed by the path or folder structure and then the MP3 filename. For example:

```
cvlc /home/pi/Downloads/CAP_01_Adventures_In_Pi.mp3
```

The `vlc` application outputs a lot of things to the console screen, many of them claiming to be "errors", but you don't need to worry about them.

`vlc` plays your downloaded MP3 file. Make sure that you have headphones or a speaker plugged into your Pi so that you can hear the file.

To stop the file from playing, press CTRL + C on the keyboard.

4. The `vlc.py` file that you downloaded earlier is in the `Adventure_10` directory. You need to copy this file into your `Documents` folder so that you can use it in your Python jukebox program. To copy the file, type:

```
cp /home/pi/Adventure_10/vlc.py /home/pi/Documents
```

You can download an entire album from the Free Music Archive using the Download Album link. Extract the MP3 files from the compressed zip folder by clicking on the completed progress bar at the bottom of the web browser page and following onscreen extraction instructions. Alternatively, type the following into a Terminal window to unzip the files, after using `cd` to change to the destination directory or folder:

```
unzip filename.zip
```

You may also want to create a folder to store all your MP3 files. You could use the File Manager application to do this, but why not practice the commands you learned in Adventure 2 for use in a Terminal window or at the command line?

To create a directory or folder, use the `mkdir` command, as follows:

```
mkdir music
```

This command creates a folder named `music`.

To move a file from one directory or folder to another, you use the `mv` command; for example:

```
cd Downloads
mv CAP_01_Adventures_In_Pi.mp3 /home/pi/music
```

This line moves the MP3 file from the current directory into the `music` directory.

To play all the MP3 files inside a directory or folder, type `cvlc` followed by the name of the folder, followed by an asterisk (*); for example:

```
cvlc music/*.mp3
```

Figure 10-6 shows the output.

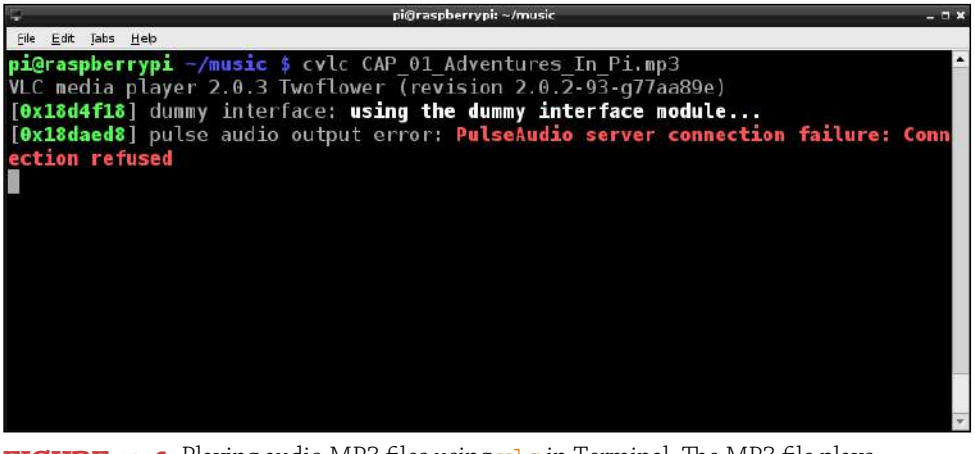


FIGURE 10-6 Playing audio MP3 files using `vlc` in Terminal. The MP3 file plays, despite the error message.

So easy, right?

Writing a Jukebox Python Program

Playing one MP3 at a time is fine, but any real music lover wants to play an entire directory or playlist, and perhaps shuffle the songs, or even skip tracks. This is the part of any big project where you can decide the functionality you want to add, as you are going to write a program in Python to interact with the `vlc` application using its library.

As in previous adventures, you use Python 3 to create the jukebox controls. Follow these steps:

1. Open Python 3(IDLE) by selecting the application from the main menu. Click File➪New File to open a new text file. Alternatively, you could use a command-line text editor like nano.

2. In the first line, as in previous projects using Python, you import the modules and libraries that you need. For example, the `glob` module is used for getting a list of MP3 files in a directory, the `random` module is used to shuffle the MP3 files inside the list, `sys` is used for `sys.exit` and getting command-line arguments, and `vlc` is the Python interface to the `vlc` library `libvlc`. Type:

```
import glob, random, sys, vlc
```

3. Leave a one-line gap underneath and then type:

```
if len(sys.argv) <= 1:
  print("Please specify a folder with mp3 files")
  sys.exit(1)
```

`sys.argv` is a list that contains the arguments passed to the program on the command line. You may remember accessing programs from the command line in earlier adventures. For instance, if you typed `python3 jukebox.py /home/pi/music` into the command line, then the list would contain `jukebox.py` at position 0 and `/home/pi/music` at position 1 (the zero position always contains the name of the program that was run). So the first argument is at position 1.

4. The next part of the program loads a list of MP3s:

```
folder = sys.argv[1]
files = glob.glob(folder+"/*.mp3")
if len(files) == 0:
  print("No mp3 files in directory", folder, ↩
  "..exiting")
  sys.exit(1)
```

As described before, position 1 in `sys.argv` is the first command-line argument. The `glob` function allows you to get a list of files matching a pattern. The pattern `"*.mp3"` expands to any file that ends in `.mp3`, so `files = glob.glob(folder+"/*.mp3")` produce a list of all files ending in `.mp3` inside the folder given on the command line. If the directory contains no MP3s there won't be anything for the jukebox to play, so you exit early using `sys.exit`.

5. Now you are able to use the random module to shuffle the list of MP3 files to put them in a random order. Underneath the previous code, type:

```
random.shuffle(files)
```

6. Leave a blank line and then type the following lines:

```
player = vlc.MediaPlayer()
medialist = vlc.MediaList(files)
```

```
mlplayer = vlc.MediaListPlayer()
mlplayer.set_media_player(player)
mlplayer.set_media_list(medialist)
```

This part of the code sets up how you will be using the `vlc` library. The details aren't important, but you set up a `MediaListPlayer`, which is a `Player` that plays a `MediaList` (rather than just having one track queued at a time).

7. Later in the project, you are going to add buttons to play, pause and skip between tracks, so here you need to add a `while` loop to read the input of buttons, and use keyboard buttons 1, 2, 3, 4 to test that it works. Begin your `while` loop with this code:

```
while True:
  button = input("Hit a button ")
  if button == "1":
    print("Pressed play button")
    if mlplayer.is_playing():
      mlplayer.pause()
    else:
      mlplayer.play()
  elif button == "2":
    print("Pressed stop button")
    mlplayer.stop()
```

The 1 key on your keyboard acts as the play/pause button, whereas the 2 key acts as the stop button. Inside the `while` loop, you have used some conditionals. Be careful with your indentation here!

The first level of the condition depends on which button is pressed: `if` 1 is pressed or else if (`elif`) 2 is pressed. Inside each button press is a second condition. If button 1 is pressed when a track is playing, the media player pauses. If nothing is playing, the media player plays a random track. If button 2 is pressed, the media player stops the track after printing `Pressed stop button` to the screen.

8. When the stop button is pressed, you want to make it so that when you hit play again the tracks have been reshuffled. To do that, you sort the list of files again using random, and then replace the `MediaList` using that reshuffled list of files. Directly underneath the last line of the previous code, and at the same level of indentation, type these lines:

```
    random.shuffle(files)
    medialist = vlc.MediaList(files)
    mlplayer.set_media_list(medialist)
```

9. Add two more buttons to this loop to skip back and forward. Be careful of the indentation of your code, as you are continuing the first conditional (see Figure 10-7). Directly underneath the last line type the following:

```
  elif button == "3":
    print("Pressed back button")
    mlplayer.previous()
  elif button == "4":
    print("Pessed forward button")
    mlplayer.next()
  else:
    print("Unrecognised input")
```

10. Save your file as `jukebox1.py` inside your `Documents` directory, by clicking File ➪ Save As and navigating to `Documents` inside `/home/pi`.

11. Finally, test to see if your code works by running it from a Terminal window by changing to the `Documents` directory and typing:

```
python3 jukebox1.py /home/pi/music
```

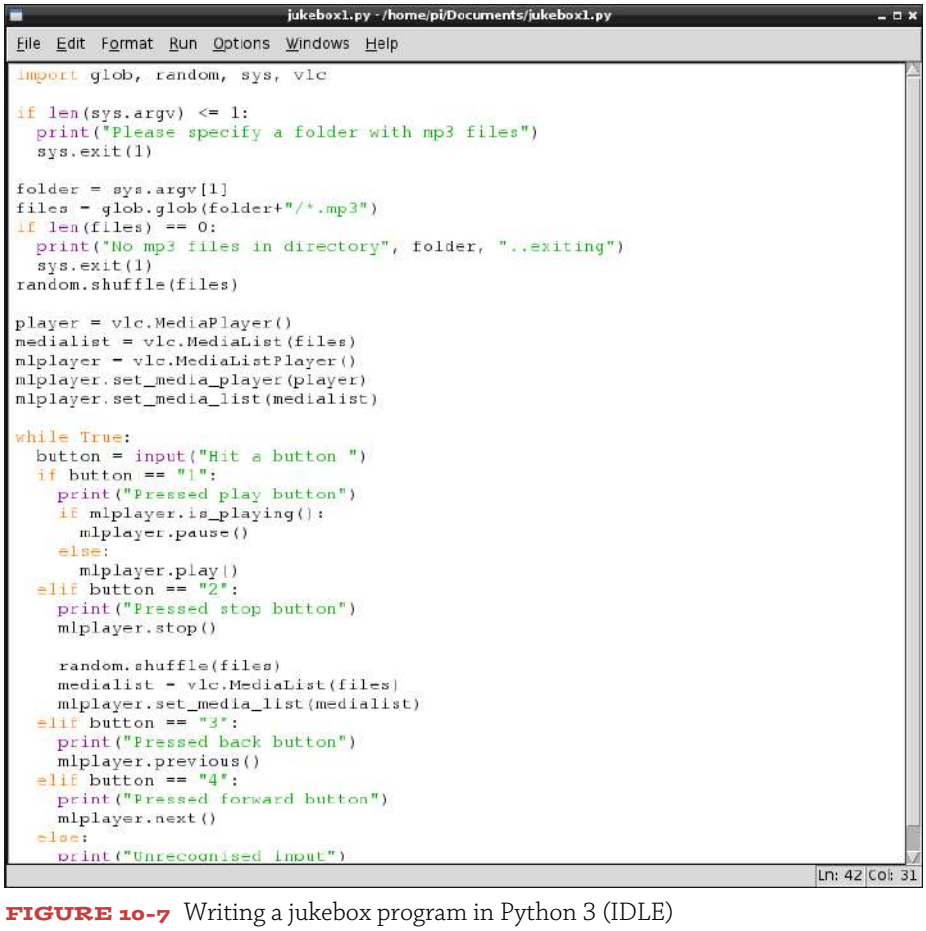


FIGURE 10-7 Writing a jukebox program in Python 3 (IDLE)

Part Three: Controlling the Jukebox with Buttons

Your jukebox uses buttons to control the playback of music on your Raspberry Pi. In this part of the project, you connect your buttons to your circuit and modify your program so that you can use the buttons to play, pause and skip tracks. You require four buttons: one to play, one to pause, one to skip tracks backwards and one to skip tracks forwards. You add these buttons to the breadboard next to the LCD screen.

Connecting the Buttons

In the following steps, you add the four buttons to the breadboard and connect them with wires to the Raspberry Pi GPIO. Refer to the wiring diagram in Figure 10-8 as you work.

You **must** turn off the power supply before doing Step 5, so I advise you to go ahead and do that now.

1. Push the buttons into the breadboard so that the legs bridge the gap along the center of the board (in the same way as the LED button project in Adventure 8).
2. Take a 10K ohm resistor and push one end into a hole, one down from the left button leg on the side of the breadboard where the LCD screen is mounted (the south side). Place the other end of the resistor into a hole in the blue (negative) power column of the breadboard, again on the south side.
3. Take a male-to-male jumper cable and place one end into a hole on the north side of the breadboard, on the same row as the other leg of the button (a white cable in Figure 10-8), and place the other end into the red (positive) power column on the north side of the breadboard.
4. Repeat Steps 2 and 3 for the three remaining buttons, using three more resistors and male-to-male jumper cables.
5. Connect the buttons to the Raspberry Pi GPIO pins. Make **sure your Raspberry Pi is powered off before you start.** You have already used many of the pins for the LCD screen—but don't worry; you only need four—one for each button. Take a male-to-female jumper cable, and push the male pin into the hole that you left between the button leg and the resistor. Push the female end into GPIO 11 (a green cable in Figure 10-8). This first button is the play button.

6. Repeat this step for the stop button, only this time connect the female end of the jumper cable to GPIO 7 (a red cable in the Figure 10-8).
7. Repeat Step 5 again for the skip tracks backwards button, connecting to GPIO 4; and the skip tracks forwards button, connecting to GPIO 10 (blue cables in Figure 10-8).
8. Finally connect the south ground rail to the north ground rail on the breadboard with a male-to-male jumper wire.

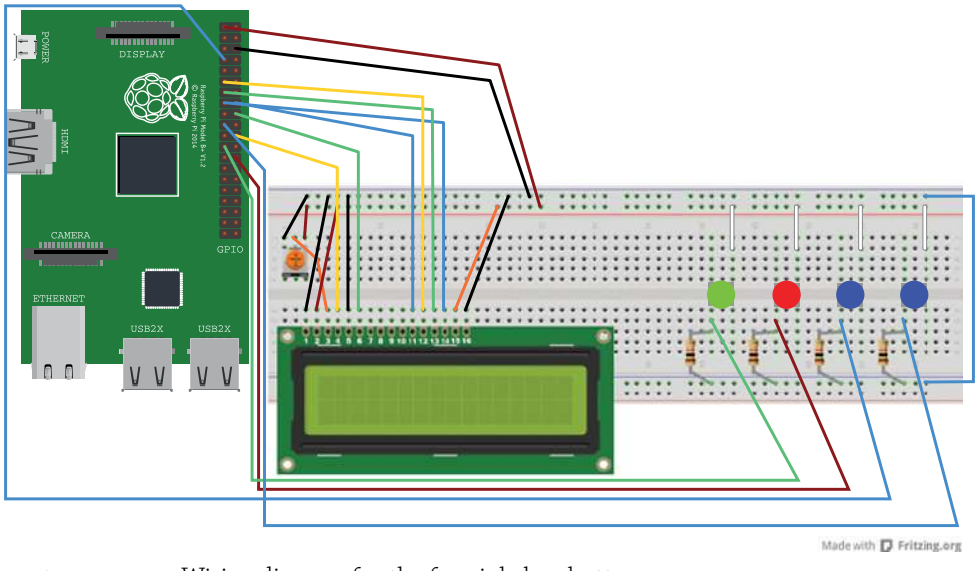


FIGURE 10-8 Wiring diagram for the four jukebox buttons

If you are using a full-sized breadboard, the power and ground rails should run all the way down the strip, which means that your LCD screen and buttons will be powered. However, if you are using a full-sized breadboard that has two halves, like the one used in Figure 10-9, you will need to bridge the channels so that power and ground extend all the way along the strip. To do this, take two male-to-male jumper cables and bridge the gap by pushing one end of the cable into the red (positive) power column hole on one side of the breadboard and the other end into a hole on the other side of the breadboard in the same red (positive) power column. Repeat this step for the blue (negative) ground column. See Figure 10-9.

The circuit being used here has pull-down resistors, which are as valid as pull-up resistors and will not damage your Pi.

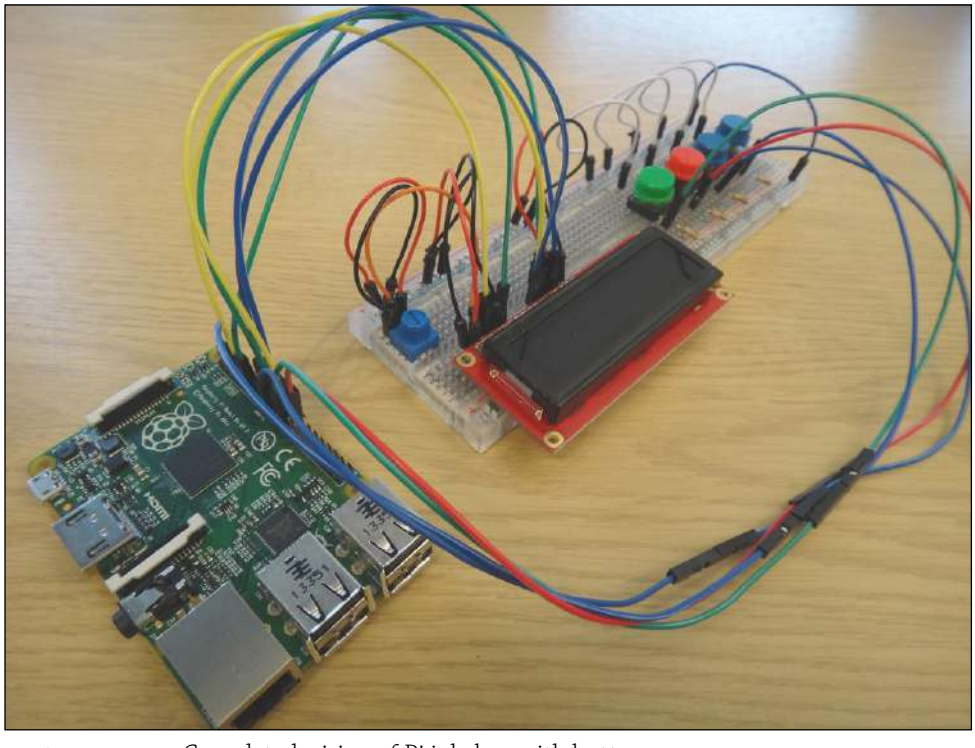


FIGURE 10-9 Completed wiring of Pi jukebox with buttons

Adapting Your Jukebox Program to Include GPIO Buttons

With the physical buttons connected to the breadboard and Raspberry Pi GPIO pins, you need to adapt the jukebox program code for them to work.

1. Open `jukebox1.py` using Python 3 (IDLE) and edit the code to include the parts highlighted in the following code:

```
import glob, random, sys, vlc, time
from gpiozero import Button
```

   Aside from the modules you imported earlier, you need to import the `time` module so that you can use the `sleep` function in your code, and you need to import `gpiozero` so that you can set up the buttons for input in the code.

2. The next part of the code remains the same:

```
if len(sys.argv) <= 1:
  print("Please specify a folder with mp3 files")
  sys.exit(1)
folder = sys.argv[1]
```

```
files = glob.glob(folder+"/*     .mp3")
if len(files) == 0:
  print("No mp3 files in directory", folder, ↩
    "..exiting")
  sys.exit(1)
random.shuffle(files)
player = vlc.MediaPlayer()
medialist = vlc.MediaList(files)
mlplayer = vlc.MediaListPlayer()
mlplayer.set_media_player(player)
mlplayer.set_media_list(medialist)
```

3. Add the code to set up the GPIO to use the physical buttons you added to your breadboard and connected to your Raspberry Pi:

```
PLAY_BUTTON = Button(11, pull_up=False)
STOP_BUTTON = Button(7, pull_up=False)
BACK_BUTTON = Button(4, pull_up=False)
FORWARD_BUTTON = Button(10, pull_up=False)
```

In this code, you assign each GPIO that you are using to appropriately named variables. Putting these in all uppercase is a convention often used in Python when you don't intend to reassign or change those variables. The advantage of this method is that you can refer to a pin (for example, `PLAY_BUTTON`) multiple times in your code. This has two advantages: It's easy to see what the name refers to; and if you change the GPIO pin for some reason the name only needs to be updated in one place.

4. Amend the `while` loop code that reads and reacts to the input. The new code uses methods from `gpiozero` to detect when a physical button is pressed. The changes are highlighted in bold in the following code (be sure to add the hash marks, also in bold, to comment out the lines as shown):

```
while True:
  # button = input("Hit a button ")
  if PLAY_BUTTON.is_pressed:
   print("Pressed play button")
   if mlplayer.is_playing():
     mlplayer.pause()
   else:
     mlplayer.play()
  elif STOP_BUTTON.is_pressed:
   print("Pressed stop button")
   mlplayer.stop()
   random.shuffle(files)
   medialist = vlc.MediaList(files)
   mlplayer.set_media_list(medialist)
```

```
  elif BACK_BUTTON.is_pressed:
   print("Pressed back button")
   mlplayer.previous()
  elif FORWARD_BUTTON.is_pressed:
   print("Pressed forward button")
   mlplayer.next()
# else:
# print("Unrecognised input")
  time.sleep(0.3)
```

Detecting GPIO input here uses the same methods as you used in Adventure 8; you just check each of the GPIO pins to see if it is connected to 3.3V (a logic high) or connected to ground (a logic low). You want to add a sleep for a short time (0.3 seconds) at the end, otherwise the same press might register multiple times. When a mechanical switch is pressed, it "bounces" between logical high and low, which would be read as multiple presses. Here you are "debouncing" it by ignoring changes that happen just after pressing the button.

5. Save this file as `jukebox2.py` inside your Documents directory.

6. It is now time to test whether your code works with your buttons. To run your adapted jukebox program, open a Terminal window, and navigate to your `Documents` folder using the following command:

```
cd Documents
```

Then type this command:

```
python3 jukebox2.py /home/pi/music
```

Now press your jukebox buttons. Is everything working as you want it to work?

Part Four: Displaying Jukebox Information on the LCD Screen

Right at the start of this big project, you connected an LCD screen to a breadboard and wired it to work using some adapted test code from Adafruit. Since then, you've pretty much ignored the display. It's time to use the LCD screen to output information about the MP3 file that is being played—the name of the artist, the name of the track and the name of the album. That information comes from the **metadata** stored within the MP3 file.

Metadata is data that describes some other sort of data. In this case, it is textual data such as the name of the artist and track, which helps to describe the audio data stored in an MP3 file. For MP3s, this can be stored inside the MP3 file alongside the audio data.

The `vlc` library allows you to attach code to certain events, such as when the track is changed on an MP3 file. This allows you to have a function that is called any time the selected event happens during playback.

1. Open `jukebox2.py` using Python 3 (IDLE) and edit the code to add the last line:

```
import glob, random, sys, vlc, time
from gpiozero import Button
from Adafruit_CharLCD import *
```

   Here the code has been amended to import the modified `Adafruit_CharLCD` library. You use from `Adafruit_CharLCD` instead of import because the class in there is called `Adafruit_CharLCD`. That means that, if you didn't use `import *` as I have done here, you would have to type `Adafruit_CharLCD.Adafruit_CharLCD`—which would seem silly!

2. The next part of the code remains the same:

```
if len(sys.argv) <= 1:
  print("Please specify a folder with mp3 files")
  sys.exit(1)
folder = sys.argv[1]
files = glob.glob(folder+"/*.mp3")
if len(files) == 0:
  print("No mp3 files in directory", folder, ↩
    "..exiting")
  sys.exit(1)
random.shuffle(files)

player = vlc.MediaPlayer()
medialist = vlc.MediaList(files)
mlplayer = vlc.MediaListPlayer()
mlplayer.set_media_player(player)
mlplayer.set_media_list(medialist)

PLAY_BUTTON = Button(11, pull_up=False)
STOP_BUTTON = Button(7, pull_up=False)
BACK_BUTTON = Button(4, pull_up=False)
FORWARD_BUTTON = Button(10, pull_up=False)
```

3. After setting up `vlc` and GPIO, you can set up the LCD screen by typing the following lines:

```
lcd = Adafruit_CharLCD()
lcd.clear()
lcd.message("Hit play!")
```

4. Leave a blank line to update the LCD screen when the track changes and then type the following code:

```
def handle_changed_track(event, player):
  media = player.get_media()
  media.parse()
  artist = media.get_meta(vlc.Meta.Artist) or ↩
    "Unknown artist"
  title = media.get_meta(vlc.Meta.Title) or ↩
    "Unknown songtitle"
  album = media.get_meta(vlc.Meta.Album) or ↩
    "Unknown album"
  lcd.clear()
  lcd.message(title+"\n"+artist+" - "+album)

playerem = player.event_manager()
playerem.event_attach(vlc.EventType.MediaPlayerMediaChanged,↩
  handle_changed_track, player)
```

Let's walk through some of this code. The two lines at the end make it so that the `handle_changed_track` function is called any time the file being played changes. This function is called if you start playback or press one of the skip buttons, but also when a track finishes and the next one starts.

Looking inside the `handle_changed_track` function, `media.parse()` causes the program to read the metadata stored in that MP3 file (the artist, track name, and so on).

`artist = media.get_meta(vlc.Meta.Artist)` gets the Artist metadata. The line `artist = media.get_meta(vlc.Meta.Artist) or "Unknown artist"` is, roughly speaking, a handy short way of writing the following:

```
if media.get_meta(vlc.Meta.Artist):
  artist = media.get_meta(vlc.Meta.Artist)
else
  artist = "Unknown artist"
```

You include this code to handle the case where the MP3 didn't have any embedded metadata.

For the `lcd.message` there are two important things to note. First, anything after the newline (`\n`) is displayed on the second line of your LCD screen. Second, you use the `+` character to join strings together into one longer string.

5. Add one last line of code beneath the `while` loop:

```
while True:
  # button = input("Hit a button ")
  if PLAY_BUTTON.is_pressed:
   print("Pressed play button")
   if mlplayer.is_playing():
     mlplayer.pause()
   else:
     mlplayer.play()
  elif STOP_BUTTON.is_pressed:
   print("Pressed stop button")
   mlplayer.stop()
   random.shuffle(files)
   medialist = vlc.MediaList(files)
   mlplayer.set_media_list(medialist)
  elif BACK_BUTTON.is_pressed:
   print("Pressed back button")
   mlplayer.previous()
  elif FORWARD_BUTTON.is_pressed:
   print("Pressed forward button")
   mlplayer.next()
# else:
# print("Unrecognised input")
  time.sleep(0.3)
  lcd.scrollDisplayLeft()
```

The `while` loop is repeated roughly every 0.3 seconds, so the LCD screen scrolls that often. This means that you can read artist and track names that are longer than the LCD screen, and it scrolls at a pleasantly readable speed using `lcd.scrollDisplayLeft()`.

6. Save this file as `jukebox3.py` inside your `Documents` directory, by clicking on File ➪ Save As and navigating to `Documents` inside `/home/pi`.

To run your final jukebox program, open a Terminal and navigate to your `Documents` folder using this command:

```
cd Documents
```

Then type the following command:

```
python3 jukebox3.py /home/pi/music
```

Press your jukebox buttons and check whether the MP3 metadata is now being displayed on the LCD screen and whether it changes when you move between tracks using the buttons (Figure 10-10). You may see some warnings but you can ignore them.

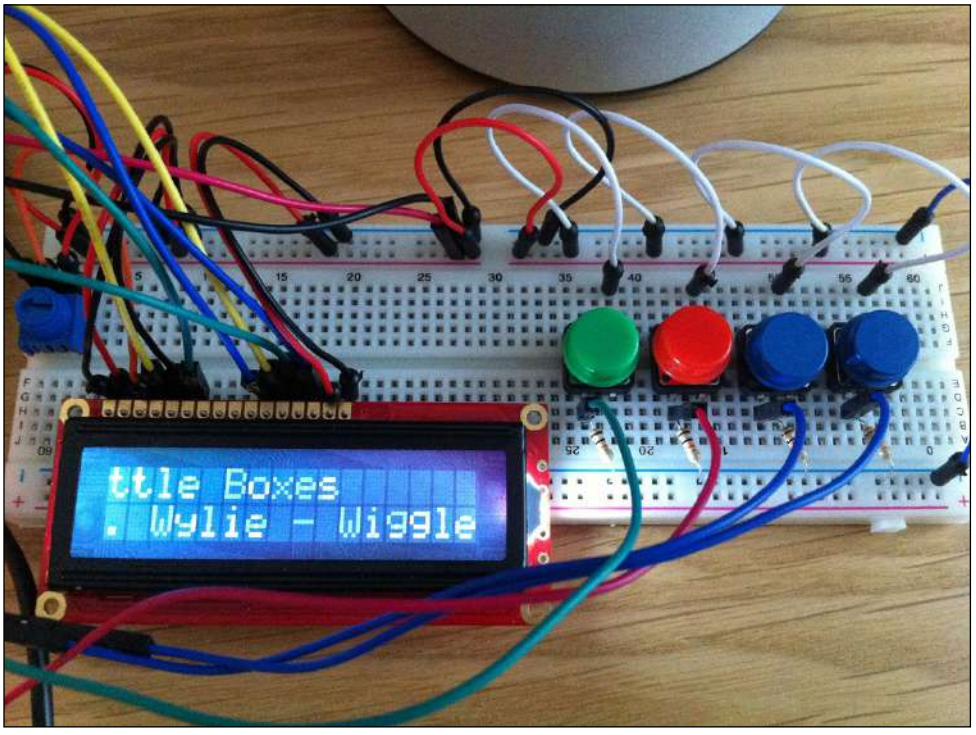


FIGURE 10-10 Jukebox LCD in action displaying MP3 metadata

USING THE JUKEBOX WITHOUT A MONITOR

If you want the Jukebox program to run whenever your Raspberry Pi is turned on, without needing it to be plugged into a keyboard, mouse or monitor, you need to modify the `/etc/rc.local` file. The `/etc/rc.local` script runs when your Raspberry Pi is starting up, at the end of the boot process. Adding the command to run the `jukebox` Python program to this script means the program runs whenever the Raspberry Pi starts up, so that you do not have to give the command to run it.

To set up the jukebox to run after booting, type the following line into a Terminal:

```
sudo nano /etc/rc.local
```

Next, scroll down through the code and add the following line before `exit 0` (see Figure 10-11):

```
python3 /home/pi/Documents/jukebox3.py /home/pi/music &
```

The ampersand symbol (&) used at the end of the above line is really important. Without it your Raspberry Pi will not complete the boot process, and you could lose all your hard work!

Save and exit the nano txt editor using Ctrl + X, accept the changes to the file by pressing Y, and press Enter.

Restart your Raspberry Pi to make sure that the program automatically runs, using the command:

```
sudo reboot
```

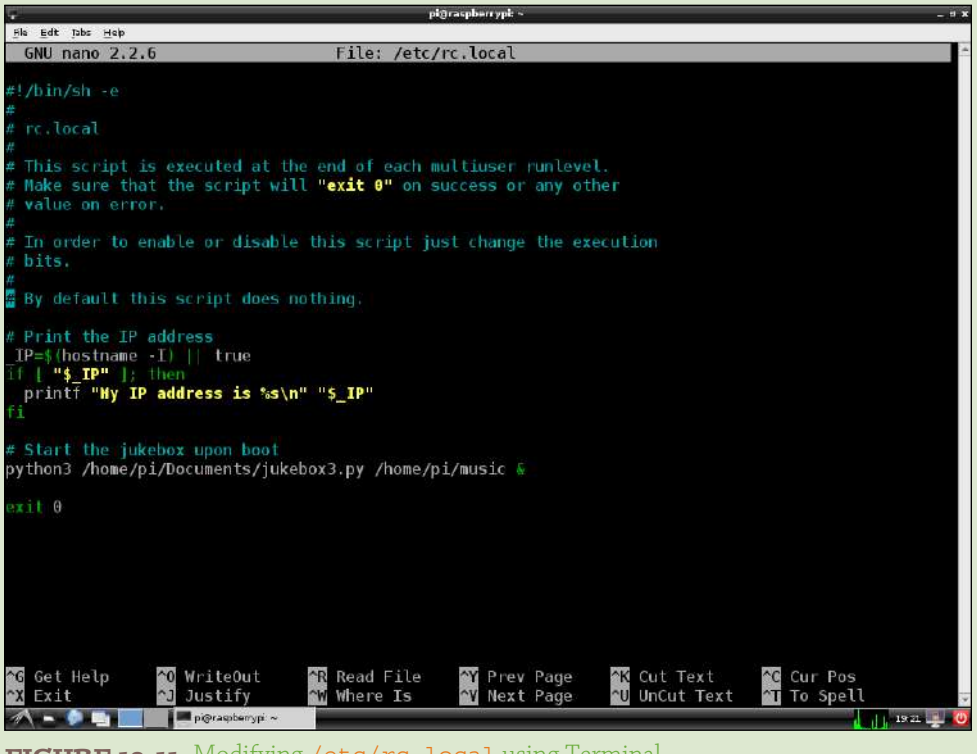


FIGURE 10-11 Modifying `/etc/rc.local` using Terminal

Finishing Up

Once you have tested everything and are happy that it all works, you can transfer your Raspberry Pi, wiring, components and breadboard into a cardboard box that you have prepared by measuring and cutting holes for the LCD screen, buttons, speaker and power supply. You could also wrap the box in copies of album covers, paste on a couple of old vinyl records, or add your own designs with pens or stickers to suit your taste.

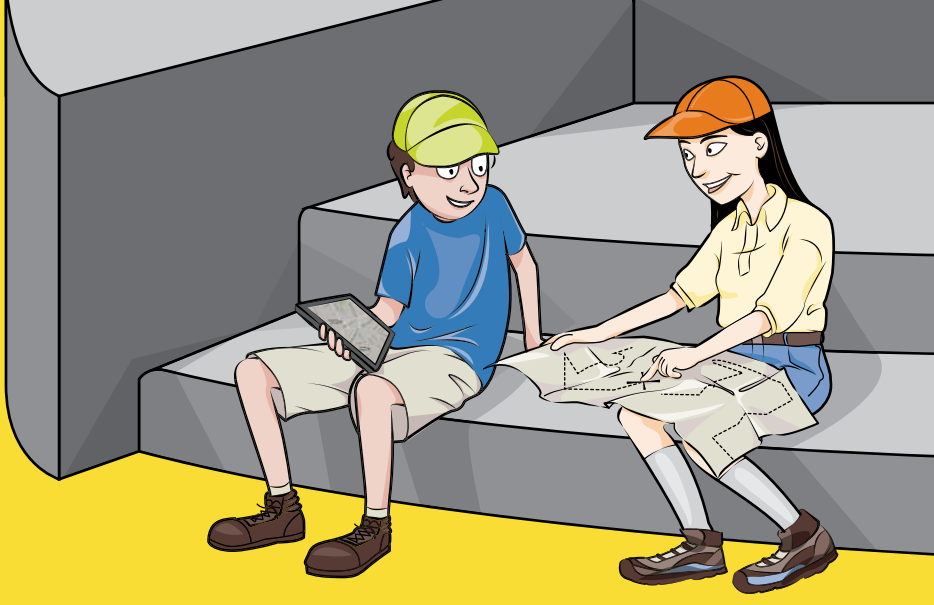
Achievement Unlocked: Your big Raspberry Pi project!

Further Adventures: Continuing Your Journey with the Raspberry Pi

Its ability to enable you to create big standalone projects like the jukebox, which let you practice your computer programming and electronics skills and use some ingenuity and creativity of your own, is what makes the Raspberry Pi such a special little device. It allows you to be the creator of technology around you—and should you get tired of your latest project, you can reincarnate your Pi in a new project!

Here are a couple of resources to help you on your way:

- *Raspberry Pi Projects* (`http://eu.wiley.com/WileyCDA/WileyTitle/productCd-1118555430.html`) by Dr Andrew Robinson and Mike Cook (Wiley, 2014) contains 16 super projects using the Raspberry Pi. They include a chicken that reads out your tweets, lights that respond to music and a computer-controlled slot car racing project.
- The official Raspberry Pi website (`www.raspberrypi.org`) adds a new post every day featuring Raspberry Pi projects from around the world, often with links to tutorials on how to make your own. It also has a Resources section with lots of projects for you to make with your Raspberry Pi.



Appendix

Where to Go from Here

NOW THAT YOU have completed the adventures in this book, it's likely that you will want to embark on your own exciting expeditions with Raspberry Pi. Hopefully you have learned some skills that will help you begin your own projects and you may wish to learn more.

There are an abundance of resources that can take you further:

- Websites
- Clubs
- Inspiring projects/tutorials
- Videos
- Books and magazines

Websites

A great way to further your adventures with Raspberry Pi is to use some of the fantastic websites that are springing up all over the Internet. There is a great culture flourishing there and plenty of opportunities for you to show off your Raspberry Pi projects

and explain how you created them so that others can have a go too. Some of the most notable and popular websites are listed here.

- The official Raspberry Pi Foundation website (`www.raspberrypi.org`) This is the official website of the Raspberry Pi Foundation. It's more than simply a resource to download the latest software for your Pi, as it features a blog with a new article every day on new developments, project ideas and inspirational stories, and the entire Raspberry Pi community uses the website's forum to discuss ideas, projects, and future developments. It is designed mainly for adults, which means it can be a little daunting at first, but if you are stuck on a project you can post a question and someone will know the answer and be able to help you out.

  It also features a Resources section that hosts lots of project tutorials to help you learn and create with Raspberry Pi.

- All About Code (`www.allaboutcode.co.uk`) This website was created by Joshua Lowe when he started to learn about computing at primary school. It includes his blog that details all the projects he has worked on with step-by-step tutorials so that others can follow along. He has even written his own Scratch and Python worksheets, as well as software that runs on a Raspberry Pi to make it easier for younger students to learn how to code.

- Adafruit Learning System (`learn.adafruit.com/category/learn-raspberry-pi`) The Adafruit learning system website features detailed lessons on electronics. You may recall that for the jukebox project in Adventure 10 you used some code from Adafruit to get the LCD display up and running. Not only does Adafruit provide open-source code for electronics projects on its site but it also has a dedicated section with lessons on Raspberry Pi GPIO that you can follow at your own pace, as well as project ideas and tutorials featuring both Pi and electronics.

- <Stuff about="code" /> (`www.stuffaboutcode.com`) In Adventure 6 you had a taste of what it is like to be a games developer of sorts by using Python code to make something happen in the world of Minecraft. If you enjoyed that project and would like more tutorials on Minecraft Pi, head over to Martin O'Hanlon's website, **Stuff about="code"**, where you'll find one of the largest collections of Minecraft Pi tutorials on the Internet. Projects range from having live Twitter feeds appear in Minecraft Pi blocks to firing cannons. There are plenty of fun ideas just waiting for you to try.

- The official Python website (`www.python.org/doc`) The Python Software Foundation has placed all the documents relating to the Python programming language online. The website includes tutorials as well as reference material for all the commands you are likely to use. It's a great site if you get stuck with writing code, especially if your program is generating syntax errors! The site can be quite confusing to navigate at first, but you will find it a valuable reference point when coding.

Clubs

It is always fun to share your computing ideas and projects with other people of your own age. There are many clubs for young people and their Raspberry Pis. Some of these are run at weekends by adults who work as professional coders. They can give you inspiration for new projects, teach you new skills and help you if you are stuck. Here is a list of some popular clubs and groups:

- Code Club (`www.codeclub.org.uk`) Code Club is an international network of free after-school coding clubs for children aged 9–13. They are usually located in schools—you can use the Code Club website to find the nearest one to where you live.
- Coder Dojo (`www.coderdojo.com`) At Coder Dojo you can learn how to code, as well as develop websites, apps, programs, games and more. Dojos are set up, run by and taught by volunteers, many of them professional programmers. Some Dojos organise tours of technology companies, bring in guest speakers to talk about their careers and what they do, and organise events. In addition to learning to code, you get to meet like-minded people, and show off what you've been working on. You can find out if there is a Coder Dojo near you by using the CoderDojo search on the website.
- Raspberry Jam (`www.raspberrypi.org/jam`) Raspberry Jam is a rapidly growing global network of user groups that meet regularly to support hobbyists, developers, teachers, students, children and families—in fact, anybody who would like to put their Raspberry Pi to good use. You can find out what Raspberry Jams are happening in your area on the official Raspberry Pi Foundation website. Some events are held especially for young people and take place in computer-based offices.
- Pioneers (`www.raspberrypi.org/pioneers`) Pioneers is an informal program for young people aged between 12 and 15 years of age. The challenges have been designed to be open-ended, so that those taking part can be as creative as they like. Participants must use digital technologies to make their project. That might mean programming a computer, building a website, hacking some hardware (like Raspberry Pi or Arduino), or using a 3D printer! Professional developers and makers assist the young people taking part who volunteer their time to help teams respond to creative and innovative challenges set by the Raspberry Pi Foundation. Not only is it a great way to learn how to code and make things, it's also a great way to meet other young people and develop a network of friends—and a lot of fun!
- School clubs Check to see if there is a Raspberry Pi or coding club at your school. If not, why not approach your ICT or computing teacher and ask about helping to start one? To start your own school club you need these things:
  - A teacher or adult willing to help supervise and run the club. This could be a teaching assistant, technician or a parent.

- A venue, such as a classroom, with tables, chairs and access to power sockets and possibly the Internet, although that is not essential.
- A suitable time to hold the club, perhaps after school one night every week. Your teacher will be able to help you with this.
- Some posters to advertise your club.
- Some enthusiastic club members who own their own Raspberry Pis to bring along.

Inspiring Projects and Tutorials

Once you have had an idea for a Raspberry Pi project and you have spent the time making it happen, you may wish to share your success with others. Many people do this through attending Raspberry Jams. Others write posts in a blog, or they add their projects to a Raspberry Pi ideas website so that others can have a go too!

- MAKE: (http://makezine.com/category/electronics/raspberry-pi/) The popular magazine *MAKE:* features a Raspberry Pi section full of Pi projects on its website. Projects include step-by-step tutorials with pictures and videos that are easy to follow.
- Instructables: (http://www.instructables.com/howto/Raspberry+Pi/) This website features lots of creative projects in tutorial form contributed by people from around the world. You can search for 'Raspberry Pi' and you'll be presented with weird and wacky ideas to inspire your adventures.

Videos

There are some great video resources on the Internet on how to use your Raspberry Pi, some of which include tutorials on how to create a particular project.

- The Raspberry Pi Foundation Official YouTube Channel (https://www.youtube.com/channel/UCFIjVWFZ__KhtTXHDJ7vgng) This YouTube channel curates great tutorial videos about Raspberry Pi in one place. The Raspberry Pi Foundation also produces videos on all the inspiring work the foundation does. It is a great place to start if you are new to Raspberry Pi.
- The Raspberry Pi Guy (www.youtube.com/user/TheRaspberryPiGuy) This young adult started his video channel as a teenager when he received his first Raspberry Pi. He's posted lots of great tutorial videos for building cool gizmos with Raspberry Pi technology. My favourite is remote-controlled electric skateboard (https://www.youtube.com/watch?v=2WLEur3M8Yk)! He also

produces excellent review videos giving his opinions about different products, which can be useful to watch if you are thinking of buying extra components for your Raspberry Pi.

- Raspberry Pi 4 Beginners (`www.pibeginners.com`) On this website you will find lots of short video explanations, tutorials and projects to teach you material specific to Raspberry Pi. For example, if you want to learn how to add your Pi to a wireless network or find out how to view your Pi's file system information, this is the place for you. The videos are created by Matthew Manning, and the website is curated by a team of Pi enthusiasts willing to help people learn about Linux.
- RasPi.TV (`http://raspi.tv`) If you are interested in learning more about the GPIO pins on the Raspberry Pi, or how to control real-world objects like lights, for example, you'll find this website and the video tutorials created by Alex Eames really helpful. Programming electronics adds an extra element of fun but it can also be quite difficult to understand what is happening in a circuit. Alex's videos contain simple explanations to help you learn. RasPi.TV has some detailed projects for enthusiastic learners, using extra hardware that interfaces with the Raspberry Pi, like the RasPiO Pro Hat.
- Geek Gurl Diaries (`www.geekgurldiaries.co.uk`) The Geek Gurl Diaries are a collection of video logs, interviews and tutorials designed for young girls. There are a number of Raspberry Pi-based tutorials, including the famous Little Box of Geek project that demonstrates how to turn your Raspberry Pi into a fortune printing box.

Books and Magazines

If you have enjoyed learning from this book, you may like to progress onto other books. There are a number of publications that will take you further after *Adventures in Raspberry Pi*. Here are some recommendations:

- ***Raspberry Pi User Guide,* Fourth Edition**, by Eben Upton and Gareth Hardacree (Wiley, 2016)
- ***Raspberry Pi For Dummies,* 2nd Edition**, by Sean McManus and Mike Cook (Wiley, 2014)
- ***Raspberry Pi Projects***, by Dr Andrew Robinson and Mike Cook (Wiley, 2014)
- ***Learning Python with Raspberry Pi***, by Alex Bradbury and Ben Everard (Wiley, 2014)
- ***Adventures in Minecraft,*** by David Whale and Martin O'Hanlon (Wiley, 2014)
- ***Adventures in Python,*** by Craig Richardson (Wiley, 2015)

- ***The MagPi Magazine*** (www.raspberrypi.org/magpi/) This monthly magazine for Raspberry Pi users contains articles that cover coding, robotics and electronics. Each issue is free to view online. All you need to do is navigate to the website, click an issue and follow the download links. You can buy hard copies of the magazine from Pi Supply (www.pi-supply.com/product-category/cables-and-accessories/books-and-magazines/).

- **Raspberry Pi Weekly** (https://www.raspberrypi.org/weekly/) This weekly newsletter is delivered electronically to your email if you sign up. It contains a roundup of all the Raspberry Pi news from the community and the Raspberry Pi Foundation.

Glossary

algorithm A set of rules to be followed to calculate or solve a problem. Common algorithms are those used for sorting information or data.

argument A piece of information given to a function, which the function then uses to perform its task. The argument goes inside the brackets that follow the function name. In the function `time.sleep(2)`, for example, you use the argument `(2)`, which is the number of seconds you want the program to wait before executing the next line.

boot The first thing a computer does when you turn it on is to start up, or *boot*, the operating system.

breadboard A reusable device that allows you to create circuits without needing to solder all the components. Breadboards have a number of holes into which you push wires or jumper cables and components to create circuits. The two columns of holes on either side of the breadboard are for power. The column next to the red line is for positive connections and the column next to the blue line is for negative connections.

broadcast A message used to coordinate the actions of different sprites and the stage in Scratch. The broadcast message keeps all the scripts running for each sprite and keeps the stage synchronized.

capacitor Electronic component used to store an electric charge. The capacity of this component is measured in farads (F). A farad is a very large quantity, so most of the capacitors you see will be measured in microfarads.

circuit diagram A diagram showing which electronic components, represented by symbols, are connected to complete a circuit and in what order they should be placed.

CLI (command-line interface) The CLI screen allows you to communicate with a computer by typing in text commands.

comments Notes within your code that explain what a line or section of code is intended to do. Each comment line begins with the # symbol, which tells the computer running the program to ignore that line.

conditional A conditional statement is a piece of code that instructs the program to take an action only if a certain condition is true. The most commonly used conditionals are `if` and `if…else` statements.

current The rate at which electrical energy flows past a point in a circuit. It is the electrical equivalent of the flow rate of water in pipes. Current is measured in amperes (A). Smaller currents are measured in milliamperes (mA).

data structure A particular way of storing and organizing related pieces of information. Lists and arrays are types of data structures.

debugging The act of locating the cause of any errors in your computer program code and fixing them.

diode A device that lets current flow in only one direction. A diode has two terminals, called *anode* and *cathode*. Current will flow through the diode only when positive voltage is applied to the anode, and negative voltage to the cathode.

flash memory A type of storage, the same kind you use with a digital camera to store all your photographs.

function A section of code that does a specific task; once the function is created you can use it over and over again. Python, like most programming languages, includes some standard functions that the computer will already understand, like the `print()` function that prints some text to the screen. You can also write your own functions.

GUI (graphical user interface) A way to interact with a computer that uses windows, icons and a mouse pointer.

hardware Refers to the physical elements of the computer that you can see and touch. This includes everything inside the computer case, known as components.

HDMI (High-Definition Multimedia Interface) HDMI devices are used to transfer video and audio data from a source device—such as your Raspberry Pi—to a compatible HDMI device like a digital TV or monitor.

hostname A word that identifies a computing device on a network. The hostname of the Raspberry Pi is `raspberrypi`.

IDE (integrated development environment) A software application used to write computer code in a particular language, for example Python; also referred to as a *programming environment*. The application has the capability to create and edit code, as well as run or execute the code. Many IDEs also provide features to help programmers *debug* or check for errors in their programs.

if/if…else statements Conditional constructs commonly used in computer programming. When you use an *if* statement, you are asking whether a condition is met, and then making something happen if the condition set is true. For example: *If* it is raining, then put up an umbrella. You can add another action for when the condition is false using the *else* command. For example: *If* it is raining, then put up an umbrella; *else,* wear sunglasses.

input The raw data or information entered into a computer system like a Raspberry Pi before it is processed. Input devices include keyboards, push buttons and microphones. The Raspberry Pi has pins that can be connected to these and other devices.

interpreter An application that checks and runs a computer program line by line.

iteration Repeating a sequence.

jumper cables Used to connect the GPIO pins on the Raspberry Pi to a breadboard or other components. They are reusable and do not require soldering. They come in different formats: female-to-male; female-to-female; and male-to-male.

LCD (liquid crystal display) An electronic display, usually quite thin and flat, that is typically used in digital calculators and digital watches to display information such as the time.

library A collection of reusable software functions that allow you do something useful.

LED (Light Emitting Diode) A diode that lights up when electricity passes through it. LEDs allow current to pass in only one direction. They come in a variety of colours, and have one short leg and one long leg, which helps you to determine which way round they need to be placed in a circuit for current to flow through them.

loop A sequence of code that repeats.

MIDI (Musical Instrument Digital Interface) keyboard A musical instrument that can communicate with a computer. Piano sheet music notes and MIDI keyboard notes are the same, only sheet music notes are represented by letters G, C, A, and so on, whereas MIDI keyboard notes are represented by numbers.

module A collection of reusable Python code that performs a specific function. It may be used alone or combined with other modules. For example, you can use functions from the Python `time` module to add pauses in your programs.

nano A text editor that enables you to write code from the command line.

NOOBS (New Out Of Box Software) A set of software produced by the Raspberry Pi Foundation, to be downloaded onto a computer and copied to an SD card that will be used on a Raspberry Pi.

operating system (OS) A type of software that allows people to create, store and manage files and applications that contain information on a computer. Examples of popular operating systems include Microsoft Windows, Mac OS X and Linux. Raspbian is a popular operating system for the Raspberry Pi.

parameters Options to commands that modify the way that the standard command works (a bit like ticking a tick box in a GUI program). Most Linux commands have lots of parameters that modify the way that they work.

output The data that your computer gives in response, after you have typed in a command. Examples of output devices include speakers and monitor screens.

potentiometer A type of resistor with an adjustable button to vary the resistance of current.

refactoring A way of restructuring code you have already written to make it more efficient and easy to read, and to avoid bugs. If you find yourself copying and pasting large sections of code, this is usually a good indicator that you need to refactor your code!

resistors Electrical components that resist current in a circuit. For example, LEDs can be damaged by too much current, but if you add the correct value resistor in series with the LED in the circuit to limit the amount of current, the LED will be protected. Resistance is measured in *ohms*. You need to pick a resistor with the correct value to limit the current through a circuit; the value of a resistor is shown by coloured bands that are read from left to right.

SD card (Secure Digital memory card) A small memory card that stores data or information. SD cards are most often used in digital cameras, to store images that can then be transferred to a computer using an SD card reader.

SD card reader/writer A device for reading information stored on SD cards and for writing information to SD cards.

software The term given to the programs that run on the computer system. Programs are what make the hardware work, for example by making a calculation or organising your files. There are two main types of software: *systems software*, which runs and manages your computer; and *application software*, which performs a specific task or function.

sprites The characters that can be programmed to do something in Scratch. The sprites wear costumes that can be customised.

stage Refers to the background for the sprites in Scratch. You can add scripts to the stage to allow the sprites to interact with it—for example, you might draw a wall that stops your sprite from moving beyond a certain point.

string Data or information entered as text, i.e., a "string" of characters.

sudo The `sudo` command lets you temporarily act as the *super user* (or *root* user) and gives you permission to do whatever you want on the system.

syntax A set of rules to check whether the code you have typed is valid code. In the same way as the English language has rules about how to properly combine subjects, verbs, objects and so on, each programming language has its own syntax.

syntax error An error that stops a program from running because the computer cannot understand the code.

terminal A screen window that gives you access to the command-line interface. The graphical LXTerminal is an example.

threads A way to run more than one script simultaneously.

turtle An imaginary pen used to create graphic images using a sequence of instructions in the Turtle Graphics program.

uinput A special hardware driver that allows other programs to inject keypresses into the system as if you had pressed a real key on the keyboard. It is a special kernel driver that has to be installed inside the Linux kernel in order to do its work.

USB (Universal Serial Bus) port A type of opening on a computer used to plug in devices such as a webcam, or a portable memory device like a memory stick.

variable A code construct that holds a value that can be changed. The `health` variable in your adventure role-playing game in Adventure 3 is an example of a value that can be changed and used inside different scripts.

voltage The difference in electrical energy between two points in a circuit. It is the electrical equivalent of water pressure in pipes, and it is this pressure that causes a current to flow through a circuit. Voltage is measured in volts (V).

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X–Y–Z

WILEY END USER LICENSE AGREEMENT

Go to www.wiley.com/go/eula to access Wiley’s ebook EULA.