NEW

Black Dog i-Tech Series

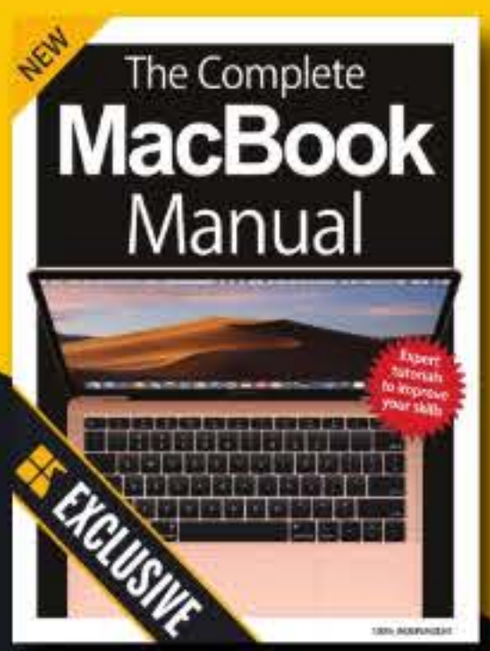
The Complete Python Manual

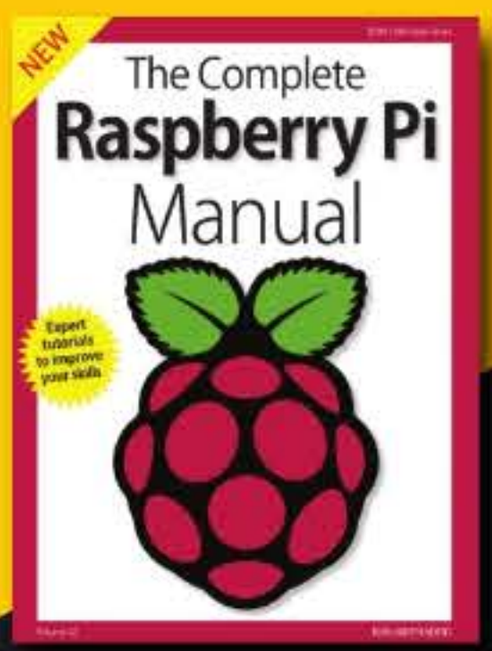
Expert tutorials to improve your skills

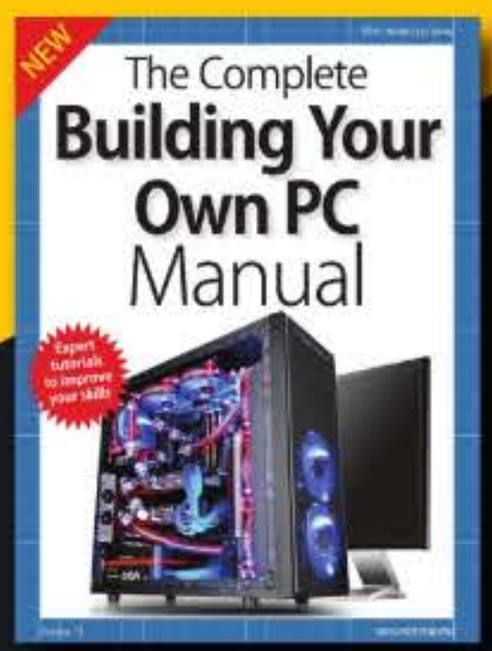
Volume 33

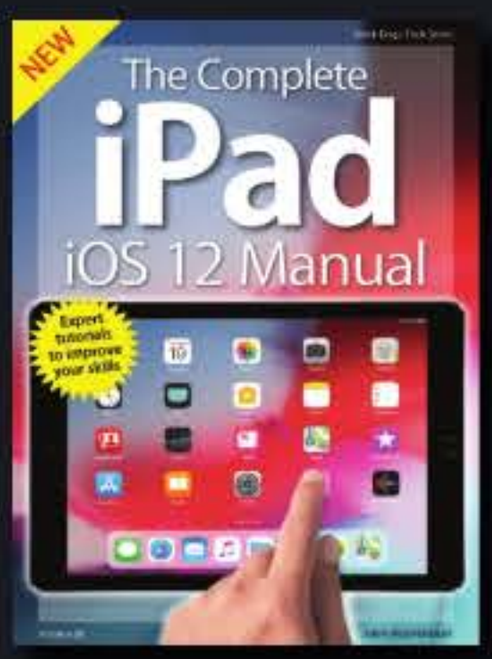
100% INDEPENDENT

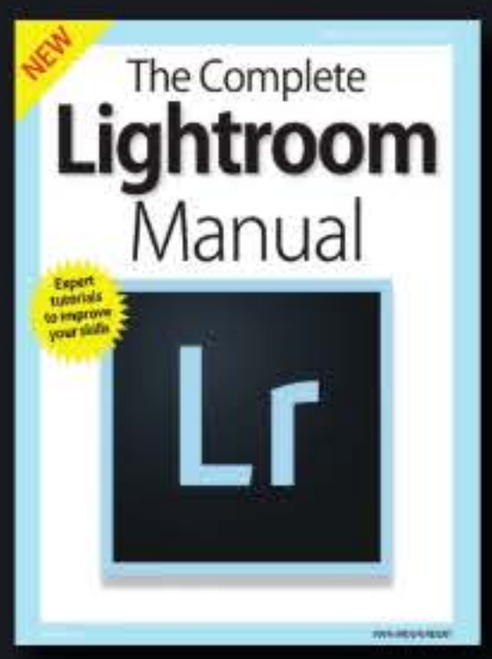
Discover more of our complete manuals on **Readly** today...

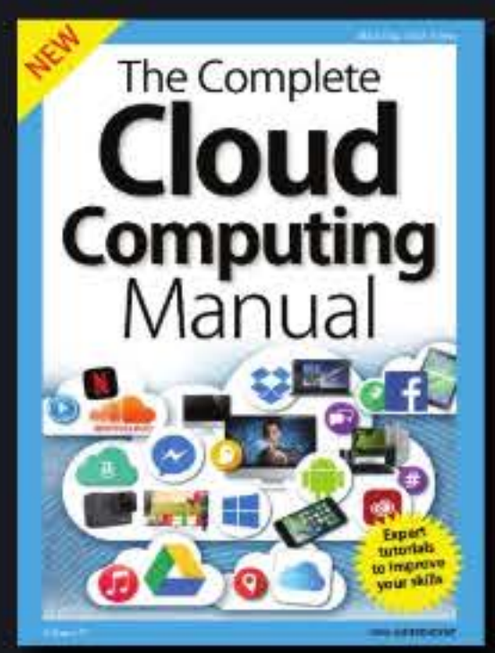


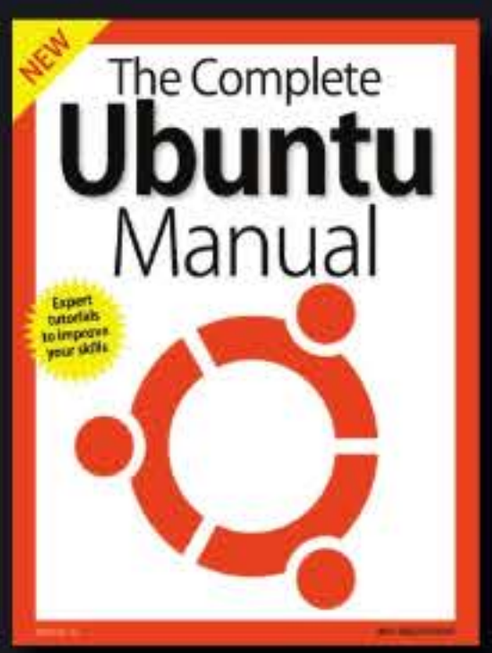


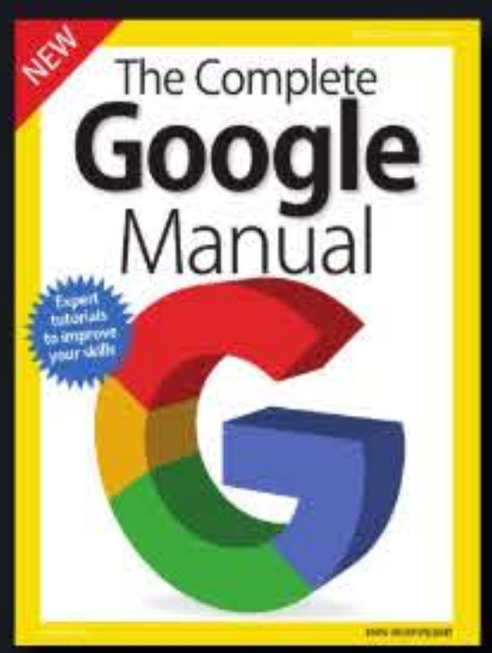


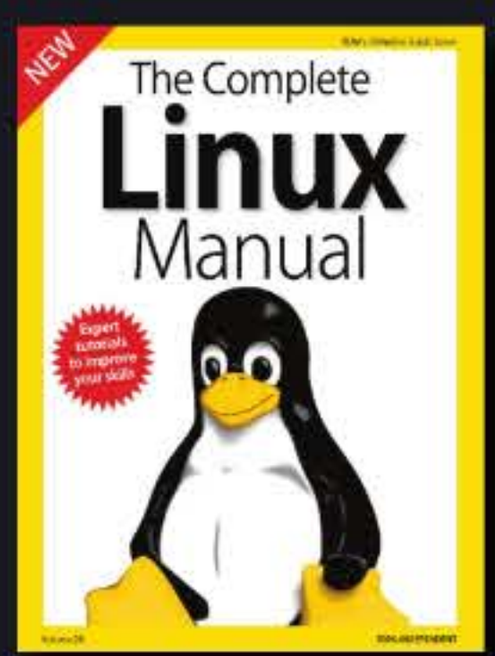


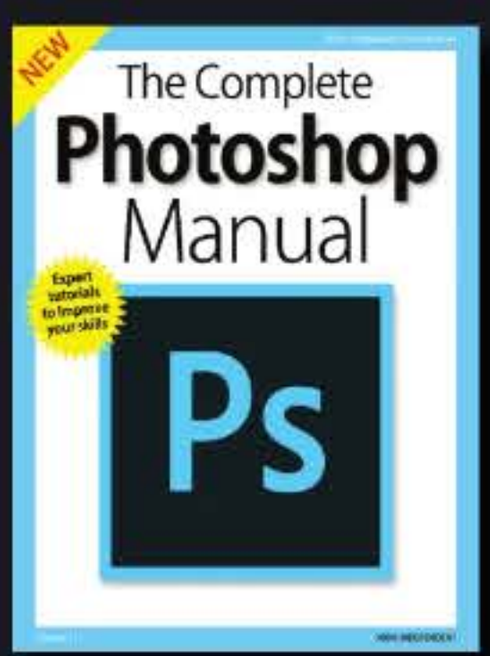


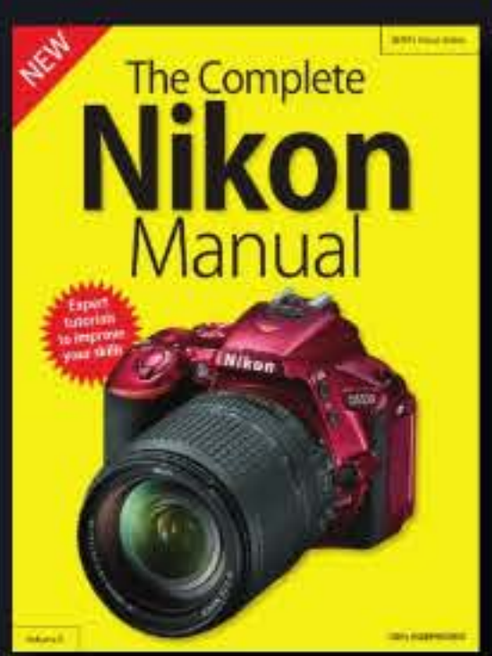


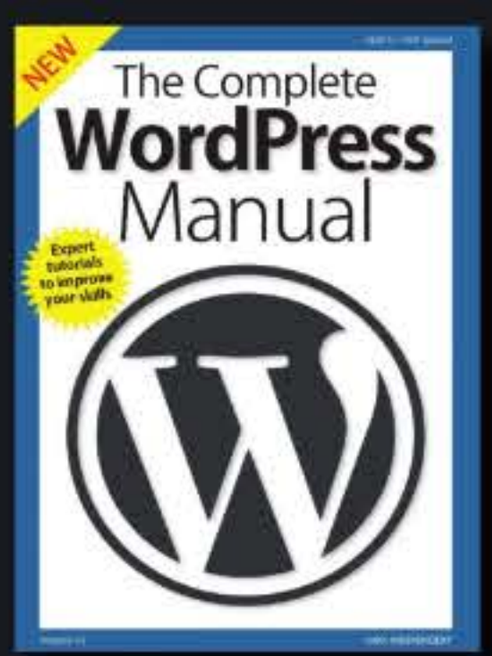












Welcome to Python, the world's best programming language

Python is a fantastic programming language that has taken the world by storm. It's easy to use, powerful, useful and fun. It works with other programming languages and with all the major computer operating systems and platforms available today. It's the ideal choice for entry-level programmers who want to learn a new skill or simply want to discover how they can create something on the computer that works for them.

Within these pages are numerous tutorials that walk you through: the process of building your first piece of code; tackling variables, numbers and expressions; user input; Python modules; and building and manipulating lists of data. We've even included a section called the Code Repository, which covers a huge range of different types of Python programming; and which you're free to delve into and use in your own unique programs. Learning Python is great fun, so what are you waiting for?

Let's get coding!

Contents

The Complete Python Manual

BDM's Code Portal
21 Python programs
3,000+ lines of code
Master Python with the help of our fantastic Code Portal, featuring code for games, tools and more. See page 90 for more details.

6 Say Hello to Python

8 Why Python?
10 Equipment You Will Need
12 Getting to Know Python
14 How to Set Up Python in Windows
16 How to Set Up Python on a Mac
18 How to Set Up Python in Linux
20 Installing a Text Editor

22 Getting Started with Python

24 Starting Python for the First Time
26 Your First Code
28 Saving and Executing Your Code
30 Executing Code from the Command Line
32 Numbers and Expressions
34 Using Comments
36 Working with Variables
38 User Input
40 Creating Functions
42 Conditions and Loops
44 Python Modules
46 Python Errors
48 Combining What You Know So Far

50 Working with Data

52 Lists
54 Tuples
56 Dictionaries
58 Splitting and Joining Strings
60 Formatting Strings
62 Date and Time
64 Opening Files
66 Writing to Files
68 Exceptions
70 Python Graphics
72 Combining What You Know So Far

74 Using Modules

76 Calendar Module
78 OS Module
80 Random Module
82 Tkinter Module
84 Pygame Module
88 Create Your Own Modules

90 Code Repository

92 Python File Manager
94 Number Guessing Game
96 Polygon Circles
97 Random Number Generator
98 Random Password Generator
99 Keyboard Drawing Script
100 Pygame Text Examples
101 Google Search Script
102 Text to Binary Convertor
104 Basic GUI File Browser
106 Mouse Controlled Turtle
107 Python Alarm Clock
108 Vertically Scrolling Text

110 Python Digital Clock
112 Pygame Music Player
113 Python Image Slideshow Script
114 Playing music with the Winsound Module
116 Text Adventure Script
118 Python Scrolling Ticker Script
119 Simple Python Calculator
120 Hangman Game Script

122 Learn Object Orientated Programming with Scratch and Python

124 Getting Started with Scratch
126 Creating Scripts in Scratch
128 Interaction in Scratch
130 Using Sprites in Scratch
132 Sensing and Broadcast
134 Objects and Local Variables
136 Global Variables and a Dice Game
138 Classes and Objects

140 Getting to Know the Command Line

142 Discovering Unix?
144 Using the Unix Filesystem
146 Listing and Viewing Files
148 Creating and Deleting Files
150 Create and Remove Directories
152 Copying, Moving and Renaming Files
154 Using the Man Pages
156 Editing Text Files
158 Getting to Know Users
160 Ownership and Permissions
162 Useful System and Disk Commands
164 Managing Programs and Processes
166 Input, Output and Pipes
168 Using Grep Pattern Matching
170 Tab Completion and History
172 Using Python Code in Linux Scripts

174 Robots, Drones and Cool Python Projects

176 Using GitHub
178 Build a CamJam EduKit Robot
180 Controlling Your Robot
182 Add Sensors to the Robot
184 Amazing Robotics Projects to Try
186 Arcade Machine Projects
188 Security Projects
190 Becoming a Coder
192 Glossary of Terms

Black Dog i-Tech Series
Volume 33 – ISSN 2044-4060
Published by: Black Dog Media Limited (BDM)
Editor: James Gale
Art Director & Production: Mark Ayshford
Production Manager: Karl Linstead
Design: Robin Drew, Lena Whitaker
Editorial: David Hayward
Sub Editor: Paul Beard

Printed and bound in Great Britain by: Wyndeham Press Group Limited
Newsstand distribution by: Seymour Distribution Limited
2, East Poultry Avenue, London EC1A 9PT
International distribution by:
Pineapple Media Limited www.pineapple-media.com
For all advertising and promotional opportunities contact:
enquiries@bdmpublications.com

Copyright © 2017 Black Dog Media. All rights reserved.

INTERNATIONAL LICENSING – Black Dog Media has many great publications and all are available for licensing worldwide.

For more information go to: www.brucesawfordlicensing.com; email: bruce@brucesawfordlicensing.com; telephone: 0044 7831 567372

Editorial and design is copyright © 2017 Papercut Limited and reproduced under licence to BlackDog Media. No part of this publication may be reproduced in any form, stored in a retrieval system or integrated into any other publication, database or commercial programs without the express written permission of the publisher. Under no circumstances should this publication and its contents be resold, loaned out or used in any form by way of trade without the publisher's written permission. While we pride ourselves on the quality of the information we provide, Black Dog Media Limited reserves the right not to be held responsible for any mistakes or inaccuracies found within the text of this publication. Due to the nature of the software industry, the publisher cannot guarantee that all tutorials will work on every version of Linux Mint, Debian-based or other Linux distributions. It remains the purchaser's sole responsibility to determine the suitability of this book and its content for whatever purpose. Images reproduced on the front and back cover are solely for design purposes and are not representative of content. We advise all potential buyers to check listing prior to purchase for confirmation of actual content. All editorial opinion herein is that of the reviewer as an individual and is not representative of the publisher or any of its affiliates. Therefore the publisher holds no responsibility in regard to editorial opinion and content.

Black Dog i-Tech Series Volume 33 is an independent publication and as such does not necessarily reflect the views or opinions of the producers contained within. This publication is not endorsed or associated in any way with The Linux Foundation, Canonical Ltd, Arch Linux, Debian Project, Lenovo, Dell, Hewlett-Packard, Apple and Samsung or any associate or affiliate company. All copyrights, trademarks and registered trademarks for the respective companies are acknowledged. Relevant graphic imagery reproduced with courtesy of Lenovo, Hewlett-Packard, Dell, Samsung, Linux Mint, Canonical, Ubuntu, Arch Linux and Apple.

Additional images contained within this publication are reproduced under licence from Shutterstock.com.

Prices, international availability, ratings, titles and content are subject to change. All information was correct at time of print. Some content may have been previously published in other volumes or BDM titles. We advise potential buyers to check the suitability of contents prior to purchase.

Black Dog Media Limited (BDM)
Registered in England & Wales No: 5311511

Say Hello to Python

In the '80s, during the golden age of the home computer, users and enthusiasts all strove to create code to perform a certain task. Regardless of what these pioneers of programming made on their machines, they forged the foundations of our digital world.

With the rise of the Internet, modern computers and consoles, home programming became something of a lost art. However, the bedroom coder has returned and thanks to the power of Python you can become the new generation of home computer programmers.

This book will help you learn how to program using the latest version of Python. With it you can create user interactive code, graphics and animations, and even your own unique games. You'll discover how to manipulate data, variables and Python modules.

So read on and let's see how to start you off on your Python adventure.

8	Why Python?
10	Equipment You Will Need
12	Getting to Know Python
14	How to Set Up Python in Windows
16	How to Set Up Python on a Mac
18	How to Set Up Python in Linux
20	Installing a Text Editor

Why Python?

There are many different programming languages available for the modern computer, and some still available for older 8 and 16-bit computers too. Some of these languages are designed for scientific work, others for mobile platforms and such. So why choose Python out of all the rest?

PYTHON POWER

Ever since the earliest home computers were available, enthusiasts, users and professionals have toiled away until the wee hours, slaving over an overheating heap of circuitry to create something akin to magic.

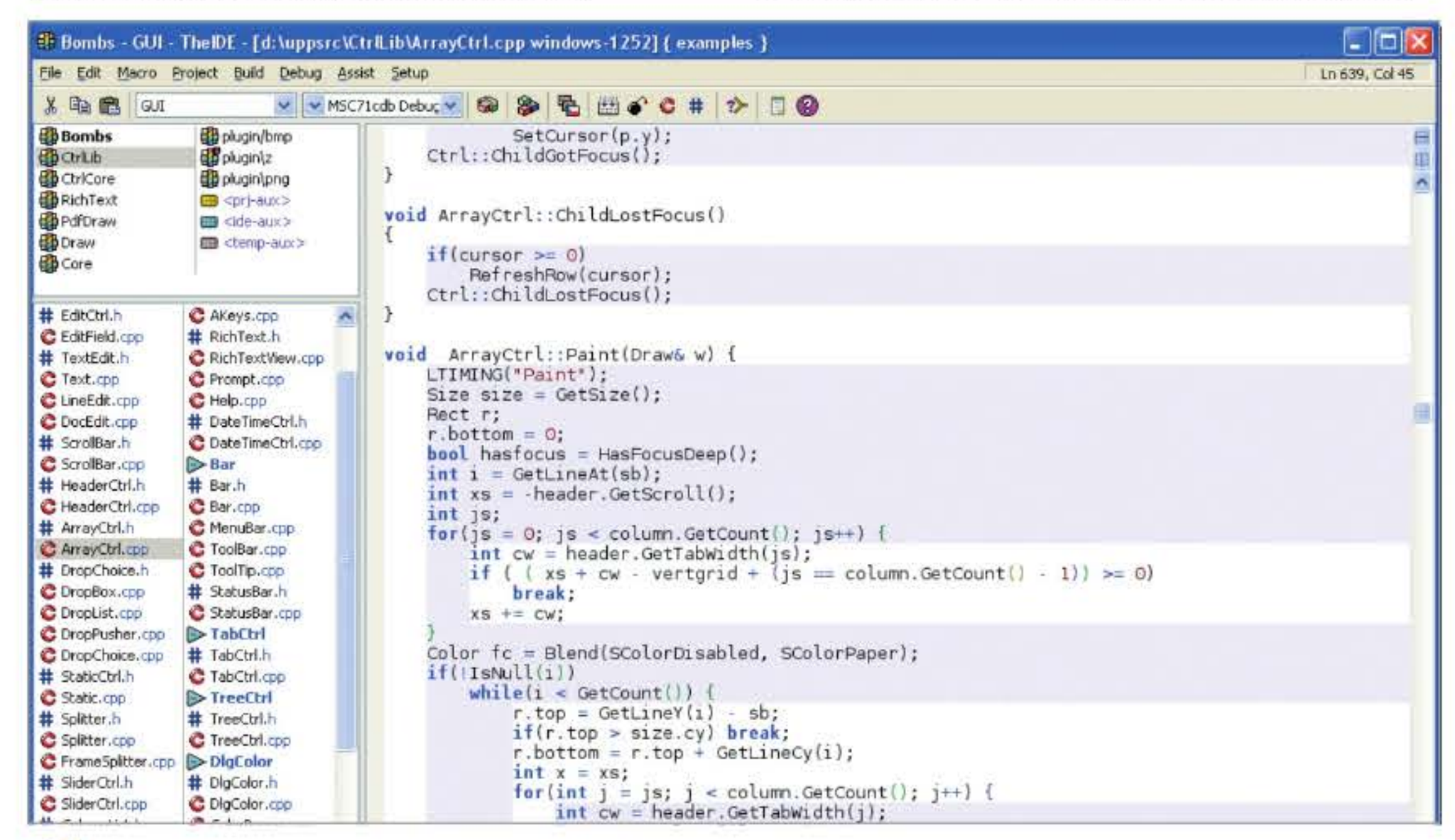
These pioneers of programming carved their way into a new frontier, forging small routines that enabled the letter 'A' to scroll across the screen. It may not sound terribly exciting to a generation that's used to ultra high-definition graphics and open world, multi-player online gaming. However, forty-something years ago it was blindingly brilliant.

Naturally these bedroom coders helped form the foundations for every piece of digital technology we use today. Some went on to become chief developers for top software companies, whereas others pushed the available hardware to its limits and founded the billion pound gaming empire that continually amazes us.

Regardless of whether you use an Android device, iOS device, PC, Mac, Linux, Smart TV, games console, MP3 player, GPS device built-in to a car, set-top box or a thousand other connected and 'smart' appliances, behind them all is programming.

All those aforementioned digital devices need instructions to tell them what to do, and allow them to be interacted with. These instructions form the programming core of the device and that core can be built using a variety of programming languages.

The languages in use today differ depending on the situation, the platform, the device's use and how the device will interact with its



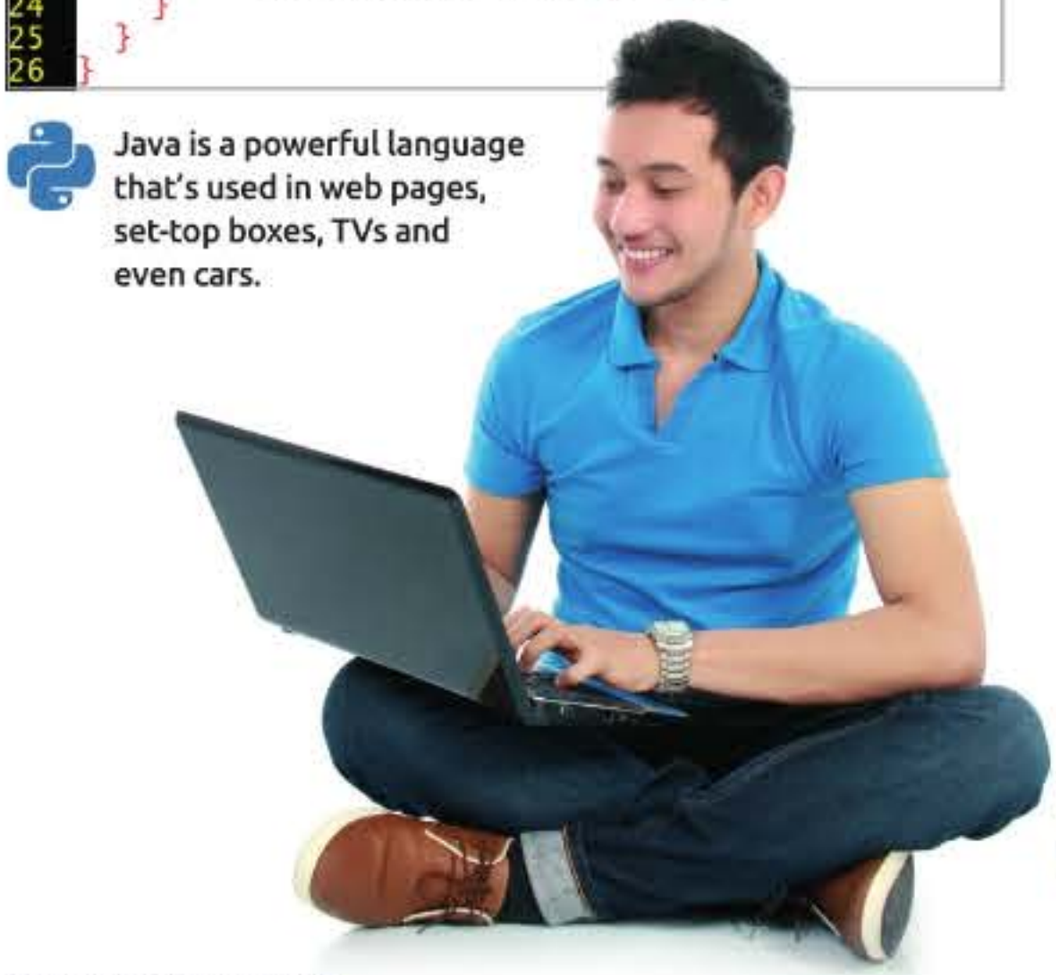
C++ is usually reserved for more complex programs, operating systems, games and so on.

environment or users. Operating systems, such as Windows, macOS and such are usually a combination of C++, C#, assembly and some form of visual-based language. Games generally use C++ whilst web pages can use a plethora of available languages such as HTML, Java, Python and so on.

More general-purpose programming is used to create programs, apps, software or whatever else you want to call them. They're widely used across all hardware platforms and suit virtually every conceivable application. Some operate faster than others and some are easier to learn and use than others. Python is one such general-purpose language.

Python is what's known as a High-Level Language, in that it 'talks' to the hardware and operating system using a variety of arrays, variables, objects, arithmetic, subroutines, loops and countless more interactions. Whilst it's not as streamlined as a Low-Level Language, which can deal directly with memory addresses, call stacks and registers, its benefit is that it's universally accessible and easy to learn.

```
1  //file: Invoke.java
2  import java.lang.reflect.*;
3
4  class Invoke {
5    public static void main( String [] args ) {
6      try {
7        Class c = Class.forName( args[0] );
8        Method m = c.getMethod( args[1], new Class
         [] { } );
9        Object ret =  m.invoke( null, null );
10       System.out.println(
11           "Invoked static method: " + args[1]
12           + " of class: " + args[0]
13           + " with no args\nResults: " + ret );
14     } catch ( ClassNotFoundException e ) {
15       // Class.forName(  ) can't find the class
16     } catch ( NoSuchMethodException e2 ) {
17       // that method doesn't exist
18     } catch ( IllegalAccessException e3 ) {
19       // we don't have permission to invoke that
         method
20     } catch ( InvocationTargetException e4 ) {
21       // an exception ocurred while invoking that
         method
22       System.out.println(
23           "Method threw an: " + e4.
             getTargetException(  ) );
24     }
25   }
26 }
```



Java is a powerful language that's used in web pages, set-top boxes, TVs and even cars.

Python was created over twenty six years ago and has evolved to become an ideal beginner's language for learning how to program a computer. It's perfect for the hobbyist, enthusiast, student, teacher and those who simply need to create their own unique interaction between either themselves or a piece of external hardware and the computer itself.

Python is free to download, install and use and is available for Linux, Windows, macOS, MS-DOS, OS/2, BeOS, IBM i-series machines, and even RISC OS. It has been voted one of the top five programming languages in the world and is continually evolving ahead of the hardware and Internet development curve.

So to answer the question: why python? Simply put, it's free, easy to learn, exceptionally powerful, universally accepted, effective and a superb learning and educational tool.

```
 40 LET py=15
 70 FOR w=1 TO 10
 71 CLS
 75 LET by=INT (RND*28)
 80 LET bx=0
 90 FOR d=1 TO 20
100 PRINT AT px,py;" U "
110 PRINT AT bx,by;"o"
120 IF INKEY$="p" THEN LET py=p
y+1
130 IF INKEY$="o" THEN LET py=p
y-1              U
135 FOR n=1 TO 100: NEXT n
140 IF py<2 THEN LET py=2
150 IF py>27 THEN LET py=27
180 LET bx=bx+1
185 PRINT AT bx-1,by;" "
190 NEXT d
200 IF (by-1)=py THEN LET s=s+1
210 PRINT AT 10,10;"score=";s
220 FOR v=1 TO 1000: NEXT v
300 NEXT w

0 OK, 0:1
```

BASIC was once the starter language that early 8-bit home computer users learned.

```
    print(HANGMAN[0])
    attempts = len(HANGMAN) - 1


    while (attempts != 0 and "-" in word_guessed):
        print(("\nYou have {} attempts remaining").format(attempts))
        joined_word = "".join(word_guessed)
        print(joined_word)

        try:
            player_guess = str(input("\nPlease select a letter between A-Z" + "\n> ")).
        except: # check valid input
            print("That is not valid input. Please try again.")
            continue
        else:
            if not player_guess.isalpha(): # check the input is a letter. Also checks a
                print("That is not a letter. Please try again.")
                continue
            elif len(player_guess) > 1: # check the input is only one letter
                print("That is more than one letter. Please try again.")
                continue
            elif player_guess in guessed_letters: # check it letter hasn't been guessed
                print("You have already guessed that letter. Please try again.")
                continue
            else:
                pass

        guessed_letters.append(player_guess)

        for letter in range(len(chosen_word)):
            if player_guess == chosen_word[letter]:
                word_guessed[letter] = player_guess # replace all letters in the chosen

        if player_guess not in chosen_word:
```

Python is a more modern take on BASIC, it's easy to learn and makes for an ideal beginner's programming language.

Equipment You Will Need

You can learn Python with very little hardware or initial financial investment. You don't need an incredibly powerful computer and any software that's required is freely available.

WHAT WE'RE USING

Throughout this book we're going to be using a Raspberry Pi version 3, with the latest version of Raspbian as the operating system. This is purely due to it being an excellent learning platform.

COMPUTER

Obviously you're going to need a computer in order to learn how to program in Python and to test your code. You can use Windows (from XP onwards) on either a 32 or 64-bit processor, an Apple Mac or Linux installed PC.

PYTHON SOFTWARE

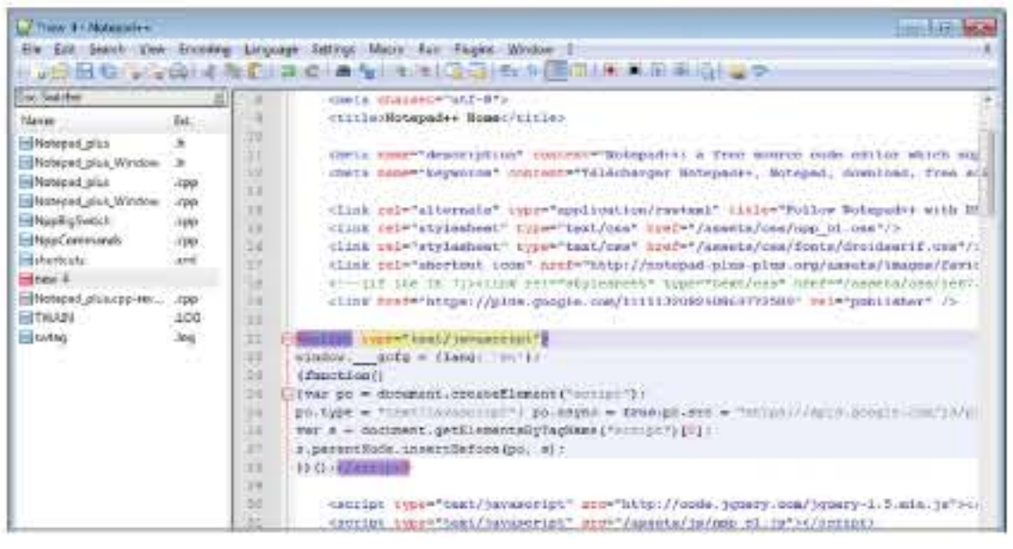
macOS and Linux already come with Python preinstalled as part of the operating system; that also includes the Raspberry Pi. However, you need to ensure you're running the latest version of Python. Windows users will need to download and install Python, which we'll cover shortly.

AN IDE

An IDE (Integrated Developer Environment) is used to enter and execute Python code. It enables you to inspect your program code and the values within the code, as well as offering advanced features. We're going to be using the default Python IDLE environment throughout this book.

TEXT EDITOR

Whilst a text editor is an ideal environment to enter code into, it's not an absolute necessity: you can enter and execute code directly from the IDLE. However, a text editor, such as Sublime Text or Notepad++, offers more advanced features and colour coding when entering code.



INTERNET ACCESS

Python is an ever evolving environment and as such, new versions often introduce new concepts or change existing commands and code structure to make it a more efficient language. Having access to the Internet will keep you up to date, help out when you get a little stuck and give you access to Python's immense number of Modules (more on Modules later).

TIME AND PATIENCE

Despite what other books may lead you to believe, you won't become a programmer in 24-hours. Learning to code in Python takes time, and patience. You may become stuck at times and other times the code will flow like water. Understand that you're learning something entirely new, and you will get there.

THE RASPBERRY PI

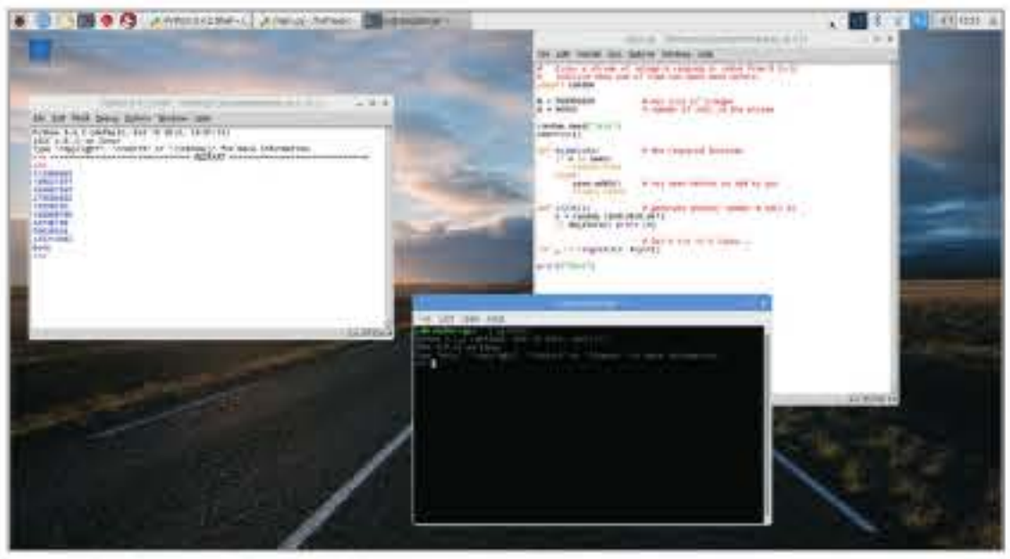
Why use a Raspberry Pi? The Raspberry Pi is a tiny computer that's very cheap to purchase, but offers the user a fantastic learning platform. Its main operating system, Raspbian, comes preinstalled with the latest Python along with many modules and extras.

RASPBERRY PI

The Raspberry Pi 3 is the latest version, incorporating a more powerful CPU, more memory, Wi-Fi and Bluetooth support. You can pick up a Pi for around £32 or as a part of kit for £50+ depending on the kit you're interested in.

RASPBIAN

The Raspberry Pi's main operating system is a Debian-based Linux distribution that comes with everything you need in a simple to use package. It's streamlined for the Pi and is an ideal platform for hardware and software projects, Python programming and even as a desktop computer.

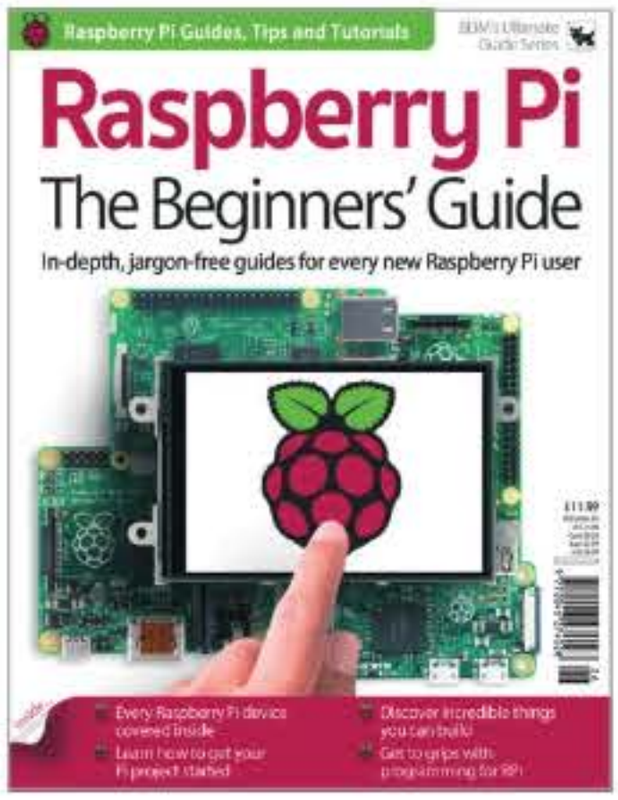


FUZE PROJECT

The FUZE is a learning environment built on the latest model of the Raspberry Pi. You can purchase the workstations that come with an electronics kit and even a robot arm for you to build and program. You can find more information on the FUZE at fuze.co.uk.

BOOKS

We have a number of great Raspberry Pi titles available via bdmpublications.com. Our Pi books cover buying your first Raspberry Pi, setting it up, how to use it and some great step-by-step project examples and how to get the most from the Raspberry Pi.



Getting to Know Python

Python is the greatest computer programming language ever created. It enables you to fully harness the power of a computer, in a language that's clean and easy to understand.

WHAT IS PROGRAMMING?

It helps to understand what a programming language is before you try to learn one, and Python is no different. Let's take a look at how Python came about and how it relates to other languages.

STEP 1

A programming language is a list of instructions that a computer follows. These instructions can be as simple as displaying your name or playing a music file, or as complex as building a whole virtual world. Python is a programming language conceived in the late 1980s by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC language.

STEP 2

Programs are like recipes for computers. A recipe to bake a cake could go like this:

Put 100 grams of self-raising flour in a bowl.
Add 100 grams of butter to the bowl.
Add 100 millilitres of milk.
Bake for half an hour.

C:\Users\lucyh\Dropbox\0_Action\recipe.txt - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
recipe.txt

```
1 Put 100 grams of self-raising flour in a bowl.
2 Add 100 grams of butter to the bowl.
3 Add 100 millilitres of milk.
4 Bake for half an hour.
```

STEP 3

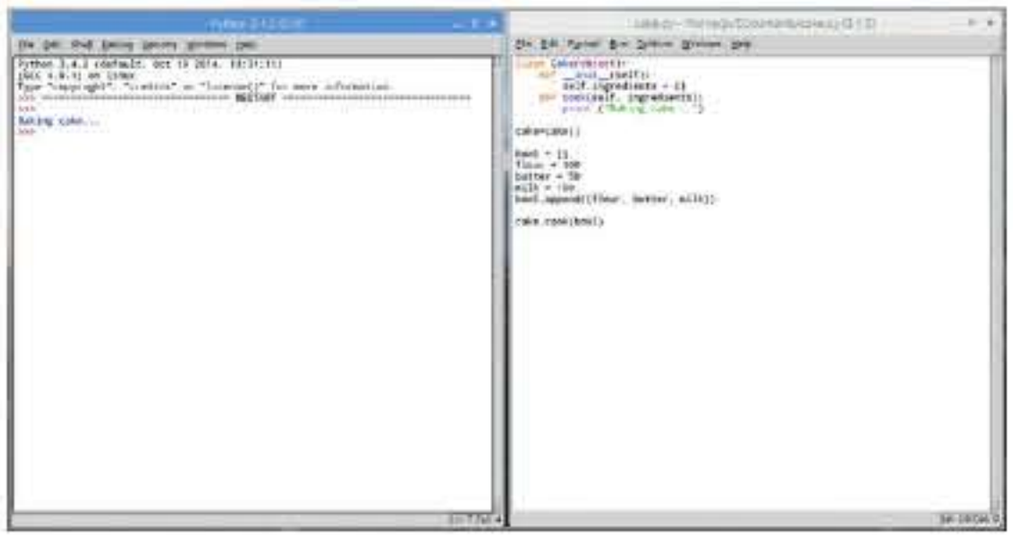
Just like a recipe, a program consists of instructions that you follow in order. A program that describes a cake might run like this:

```
bowl = []
flour = 100
butter = 50
milk = 100
bowl.append([flour,butter,milk])
cake.cook(bowl)
```

```
class Cake(object):
    def __init__(self):
        self.ingredients = []
    def cook(self,ingredients):
        print "Baking cake ..."

cake = Cake()

bowl = []
flour = 100
butter = 50
milk = 100
bowl.append([flour,butter,milk])

cake.cook(bowl)
```

STEP 4

You might not understand some of the Python commands, like bowl.append and cake.cook(bowl). The first is a list, the second an object; we'll look at both in this book. The main thing to know is that it's easy to read commands in Python. Once you learn what the commands do, it's easy to figure out how a program works.



STEP 5 Computer languages that are easy to read are known as "high-level". This is because they fly high above the hardware (also referred to as "the metal"). Languages that "fly close to the metal," like Assembly, are known as "low-level". Low-level languages commands read a bit like this: `msg db ,0xa` `len equ $ - msg`.

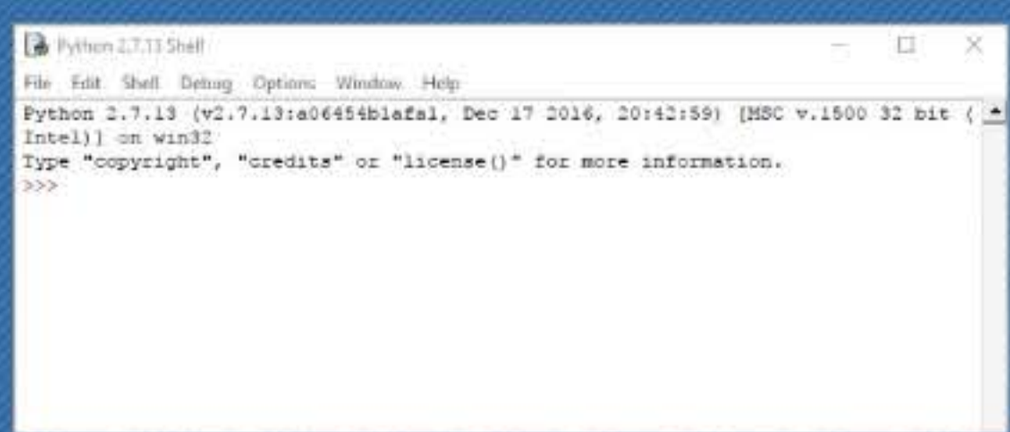
STEP 6 Python lets you access all the power of a computer in a language that humans can understand. Behind all this is an ethos called "The Zen of Python." This is a collection of 20 software principles that influences the design of the language. Principles include "Beautiful is better than ugly" and "Simple is better than complex." Type `import this` into Python and it will display all the principles.

PYTHON 3 VS PYTHON 2

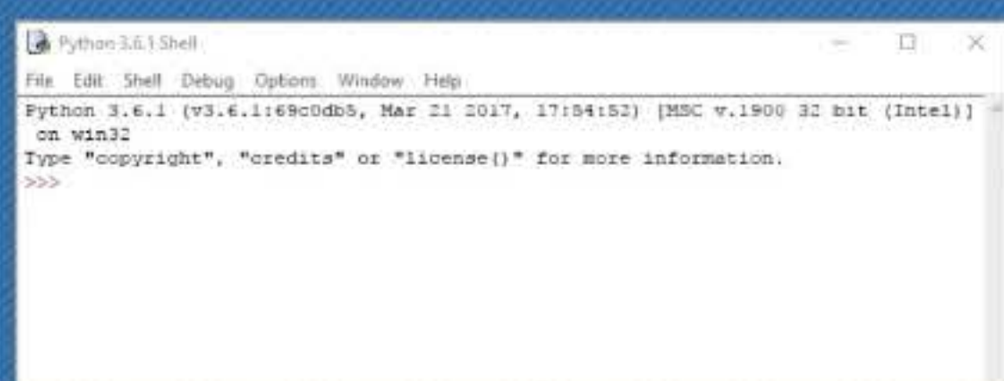
In a typical computing scenario, Python is complicated somewhat by the existence of two active versions of the language: Python 2 and Python 3.

STEP 1 When you visit the Python Download page you'll notice that there are two buttons available: one for Python 3.6.2 and the other for Python 2.7.13; correct at the time of writing (remember Python is frequently updated so you may see different version numbers).

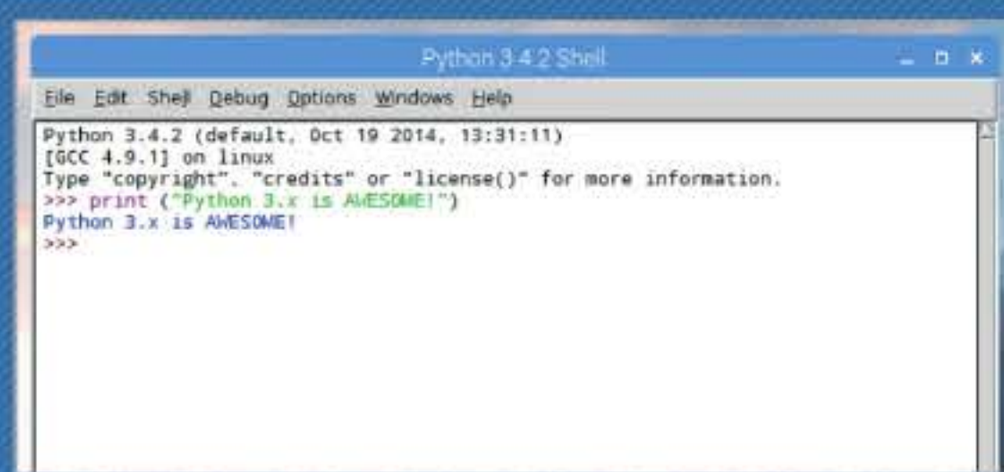
STEP 2 So why two? Well, Python 2 was originally launched in 2000 and has since then adopted quite a large collection of modules, scripts, users, tutorials and so on. Over the years Python 2 has fast become one of the first go to programming languages for beginners and experts to code in, which makes it an extremely valuable resource.



STEP 3 In 2008 Python 3 arrived with several new and enhanced features. These features provide a more stable, effective and efficient programming environment but sadly, most (if not all) of these new features are not compatible with Python 2 scripts, modules and tutorials. Whilst not popular at first, Python 3 has since become the cutting edge of Python programming.



STEP 4 Python 3's growing popularity has meant that it's now prudent to start learning to develop with the new features and begin to phase out the previous version. Hence why this book will be solely dedicated to programming in Python 3.x and not Python 2.x.



How to Set Up Python in Windows

Windows users can easily install the latest version of Python via the main Python Downloads page. Whilst most seasoned Python developers may shun Windows as the platform of choice for building their code, it's still an ideal starting point for beginners.

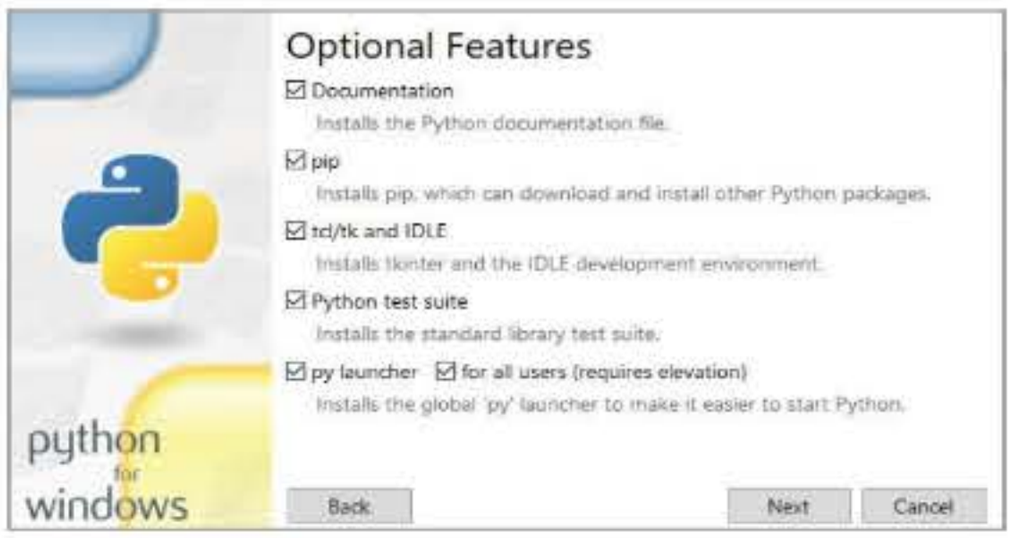
INSTALLING PYTHON 3.X

Microsoft Windows doesn't come with Python preinstalled as standard, so you're going to have to install it yourself manually. Thankfully, it's an easy process to follow.

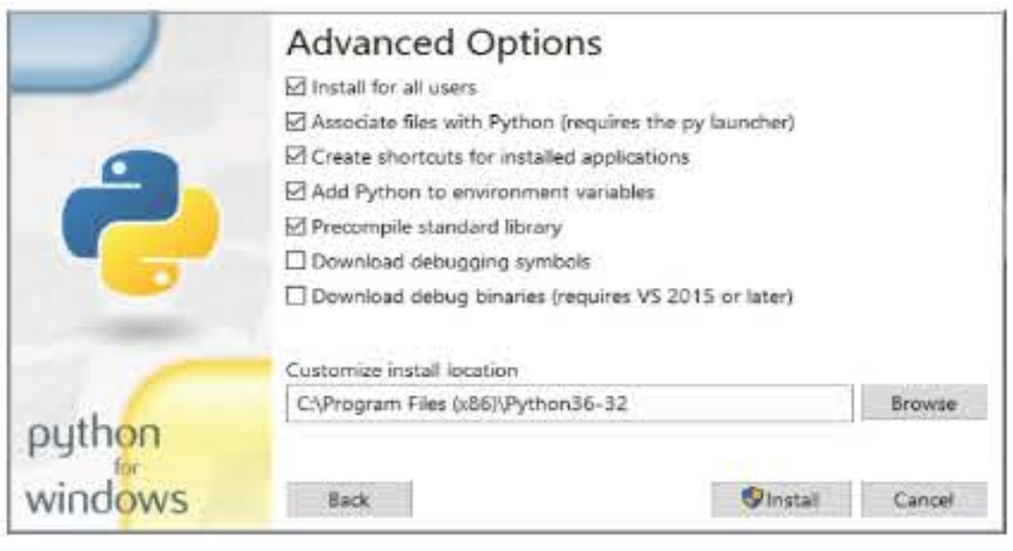
STEP 1 Start by opening your web browser to www.python.org/downloads/. Look for the button detailing the download link for Python 3.x.x (in our case this is Python 3.6.2 but as mentioned you may see later versions of 3).

STEP 2 Click the download button for version 3.x, and save the file to your Downloads folder. When the file is downloaded, double-click the executable and the Python installation wizard will launch. From here you have two choices: Install Now and Customise Installation. We recommend opting for the Customise Installation link.

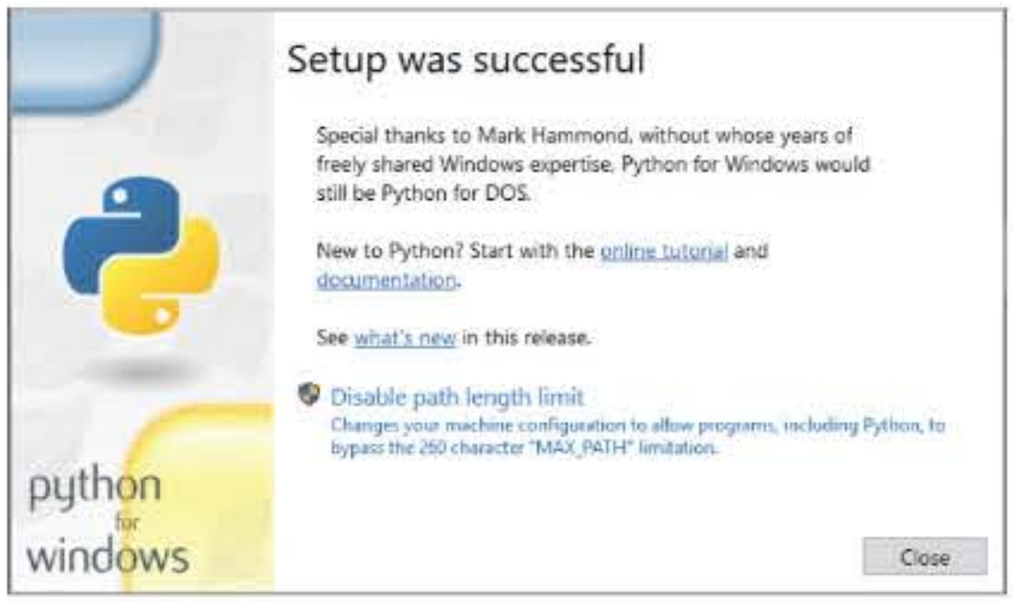
STEP 3 Choosing the Customise option allows you to specify certain parameters, and whilst you may stay with the defaults, it's a good habit to adopt as sometimes (not with Python, thankfully) installers can include unwanted additional features. On the first screen available, ensure all boxes are ticked and click the Next button.



STEP 4 The next page of options include some interesting additions to Python. Ensure the Associate file with Python, Create Shortcuts, Add Python to Environment Variables, Precompile Standard Library and Install for All Users options are ticked. These make using Python later much easier. Click Install when you're ready to continue.

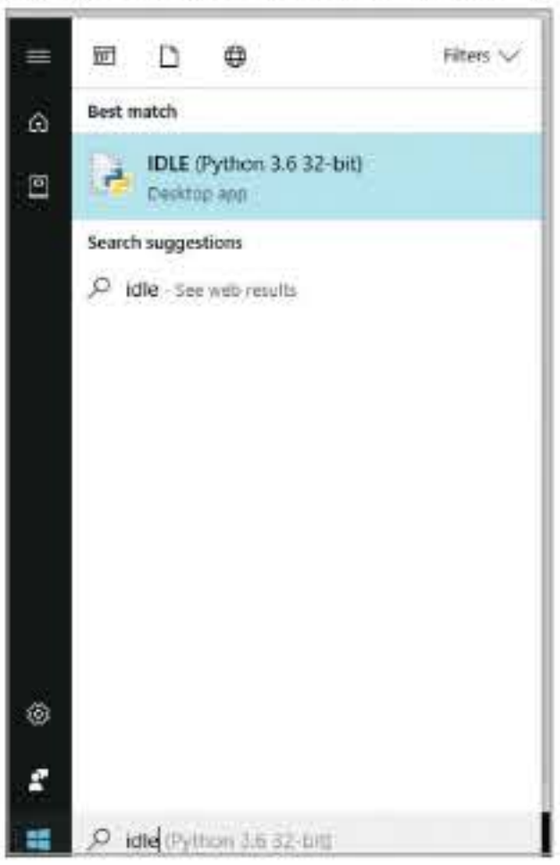


STEP 5 You may need to confirm the installation with the Windows authentication notification. Simply click Yes and Python will begin to install. Once the installation is complete the final Python wizard page will allow you to view the latest release notes, and follow some online tutorials.

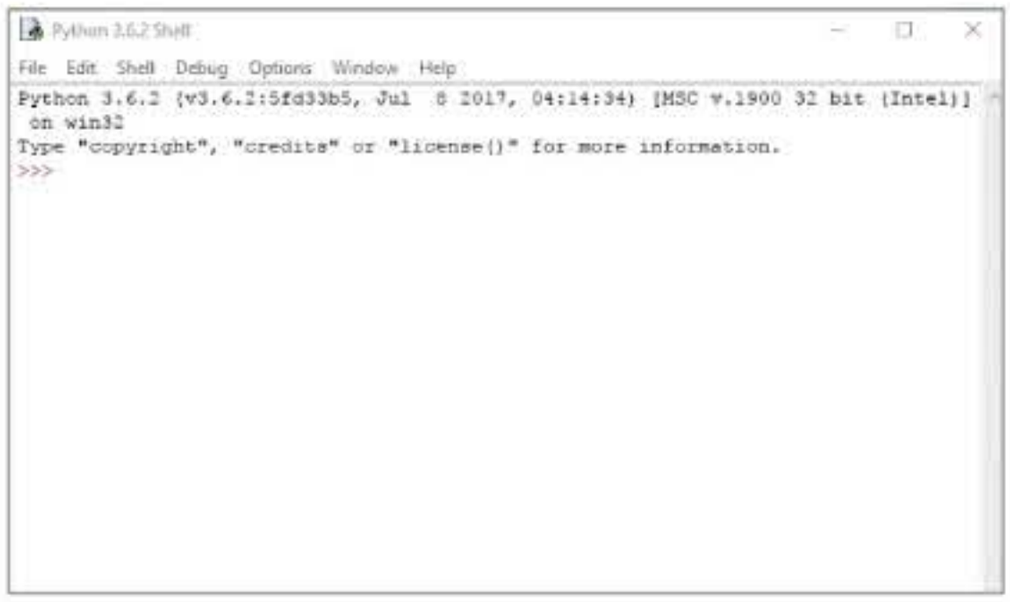


STEP 6 Before you close the install wizard window, however, it's best to click on the link next to the shield detailed Disable Path Length Limit. This will allow Python to bypass the Windows 260 character limitation, enabling you to execute Python programs stored in deep folders arrangements. Again, click Yes to authenticate the process; then you can Close the installation window.

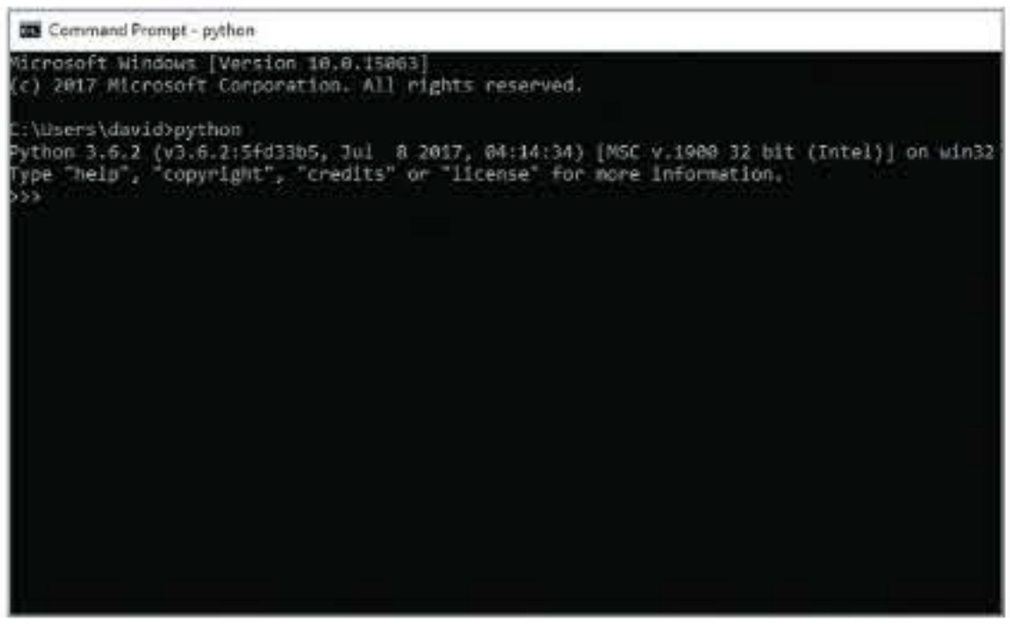
STEP 7 Windows 10 users will now find the installed Python 3.x within the Start button Recently Added section. The first link, Python 3.6 (32-bit) will launch the command line version of Python when clicked (more on that in a moment). To open the IDLE, type IDLE into Windows start.



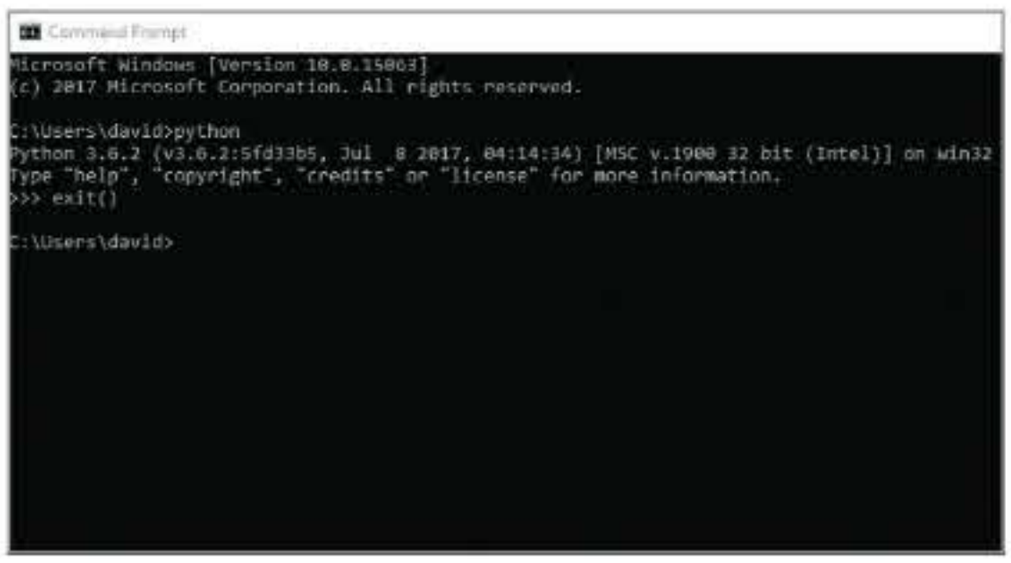
STEP 8 Clicking on the IDLE (Python 3.6 32-bit) link will launch the Python Shell, where you can begin your Python programming journey. Don't worry if your version is newer, as long as it's Python 3.x our code will work inside your Python 3 interface.



STEP 9 If you now click on the Windows Start button again, and this time type: `CMD`, you'll be presented with the Command Prompt link. Click it to get to the Windows command line environment. To enter Python within the command line, you need to type: `python` and press Enter.



STEP 10 The command line version of Python works in much the same way as the Shell you opened in Step 8; note the three left-facing arrows (>>>). Whilst it's a perfectly fine environment, it's not too user-friendly, so leave the command line for now. Enter: `exit()` to leave and close the Command Prompt window.



How to Set Up Python on a Mac

If you're running an Apple Mac, then setting up Python is incredibly easy. In fact a version of Python is already installed. However, you should make sure you're running the latest version.

INSTALLING PYTHON

Apple's operating system comes with Python installed, so you don't need to install it separately. However, Apple doesn't update Python very often and you're probably running an older version. So it makes sense to check and update first.

STEP 1 Open a new Terminal window by clicking Go > Utilities, then double-click the Terminal icon. Now enter: `python --version`. You should see "Python 2.5.1" and even later, if Apple has updated the OS and Python installation. Either way, it's best to check for the latest version.

STEP 2 Open Safari and head over to www.python.org/downloads. Just as with the Windows set up procedure on the previous pages, you can see two yellow download buttons: one for Python 3.6.2, and the other for Python 2.7.13. Note, that version numbers may be different due to the frequent releases of Python.

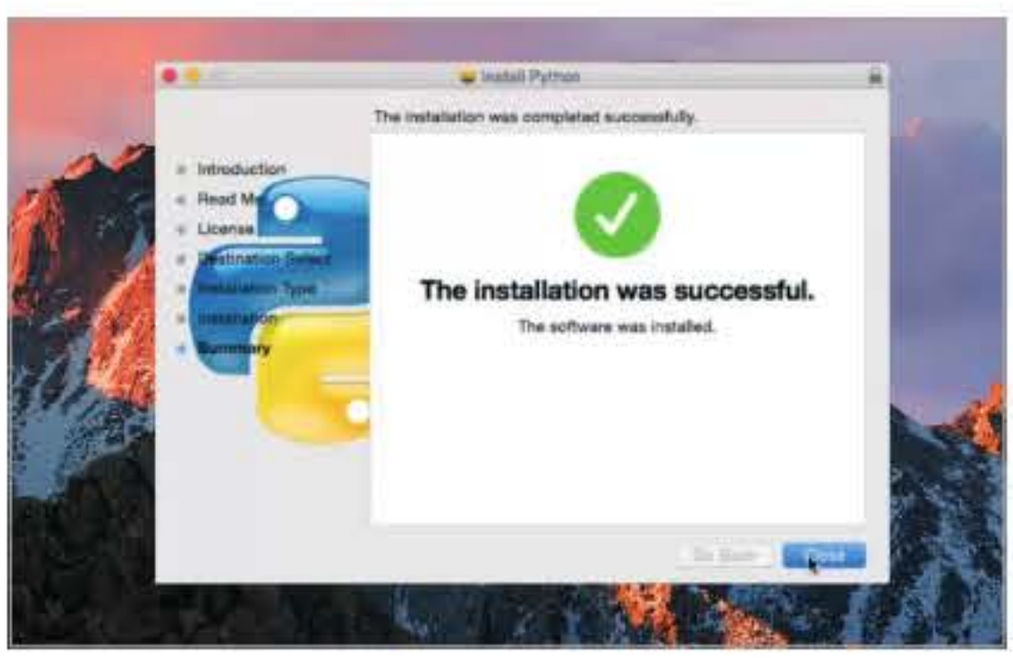
STEP 3 Click on the latest version of Python 3.x, in our case this is the download button for Python 3.6.2. This will automatically download the latest version of Python and depending on how you've got your Mac configured, it automatically starts the installation wizard.

STEP 4 With the Python installation wizard open, click on the Continue button to begin the installation. It's worth taking a moment to read through the Important Information section, in case it references something that applies to your version of macOS. When ready, click Continue again.

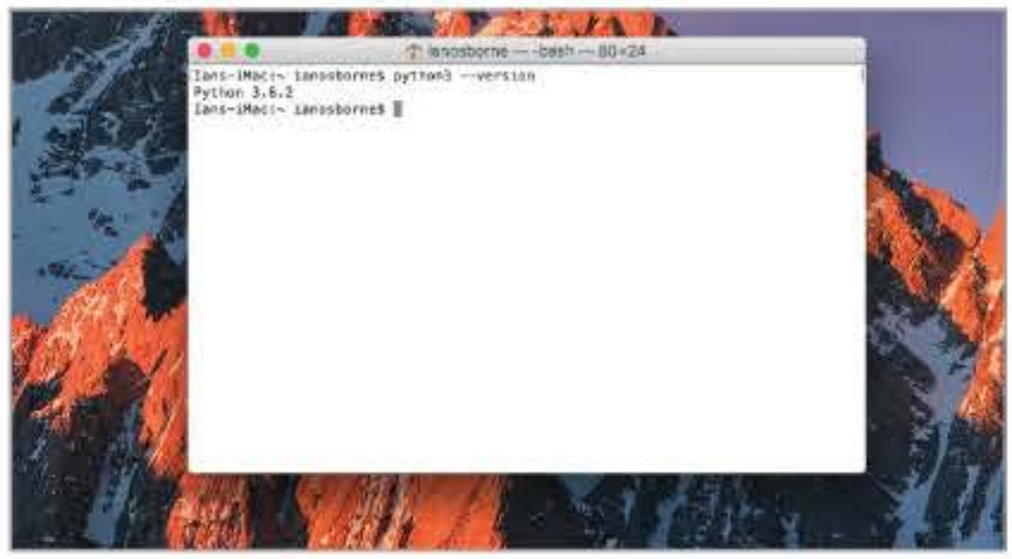
STEP 5 The next section details the Software License Agreement, and whilst not particularly interesting to most folks, it's probably worth a read. When you're ready, click on the Continue button once again.

STEP 6 Finally you're be presented with the amount of space Python will take up on your system and an Install button, which you need to click to start the actual installation of Python 3.x on to your Mac. You may need to enter your password to authenticate the installation process.

STEP 7 The installation shouldn't take too long; the older Mac Mini we used in this section is a little slower than more modern Mac machines and it only took around thirty seconds for the Installation Successful prompt to be displayed.

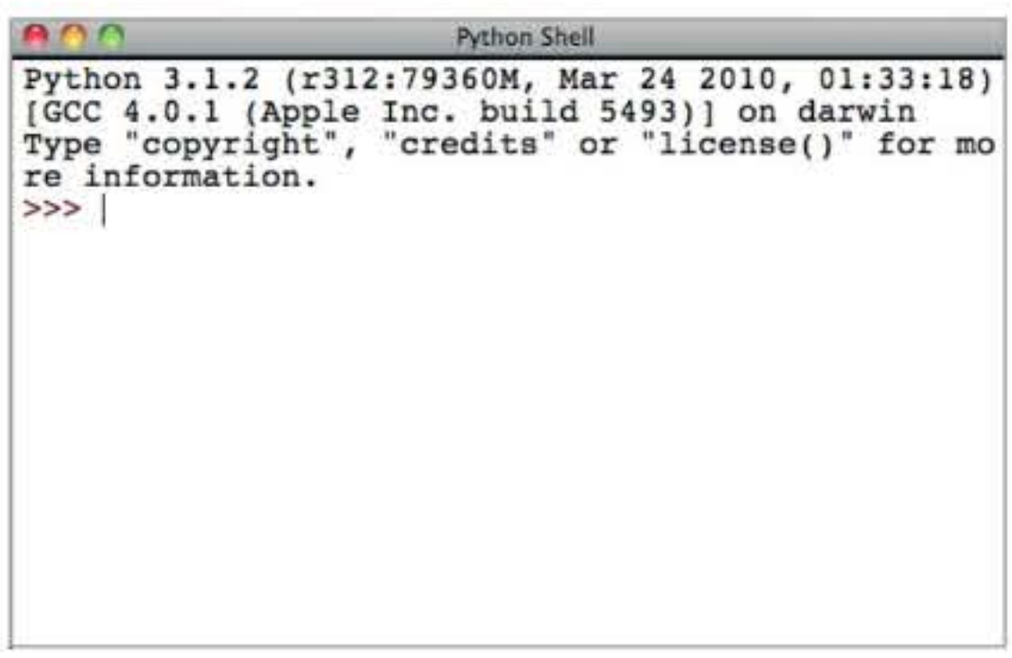


STEP 8 There's nothing much else left to do in the Python installation wizard so you can click the Close button. If you now drop back into a Terminal session and re-enter the command: `python3 --version`, you can see the new version is now listed. To enter the command line version of Python, you need to enter: `python3`. To exit, it's: `exit()`.



STEP 9 You need to search in Finder for the Python IDLE; when you've found it, click it to launch and it should look similar to that of the Windows IDLE version shown on the previous page. The only difference being the Mac detected hardware platform it's running on.

STEP 10 Older Mac versions may have trouble with the newer versions of Python, in which case you will need to revert to a previous Python 3.x build; as long as you're using Python 3.x, the code in this book will work for you.



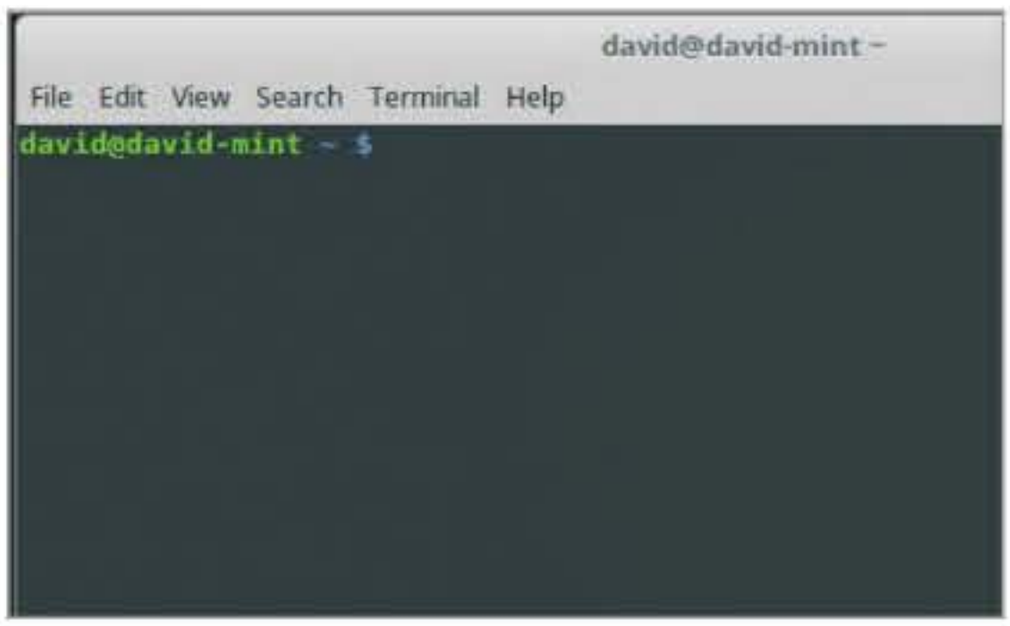
How to Set Up Python in Linux

Python version 2.x is already installed in most Linux distributions but as we're going to be using Python 3.x, there's a little work we need to do first to get hold of it. Thankfully, it's not too difficult.

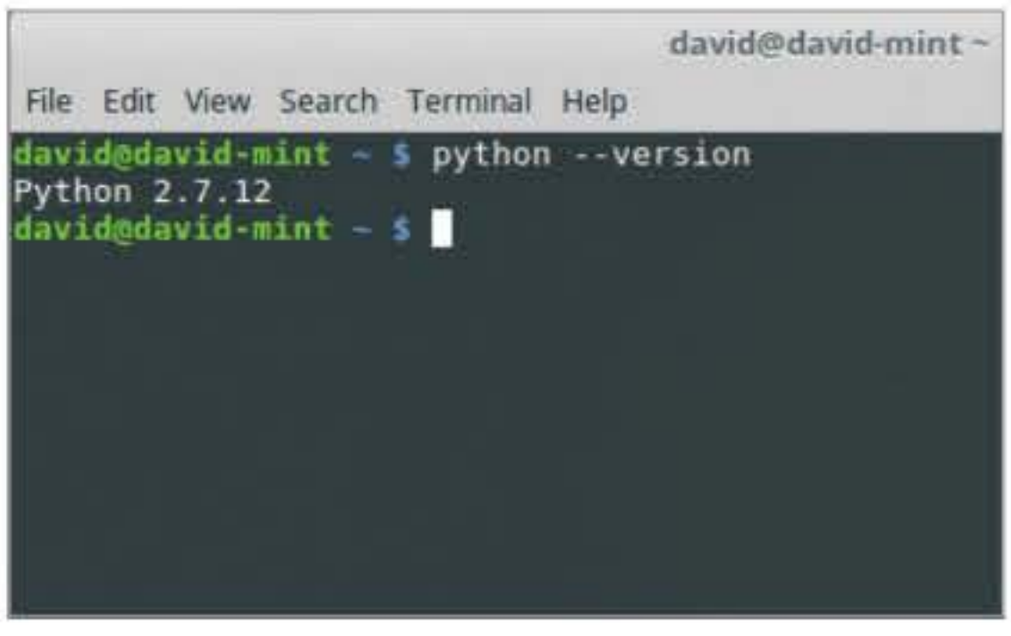
PYTHON PENGUIN

Linux is such a versatile operating system that it's often difficult to nail down just one way of doing something. Different distributions go about installing software in different ways, so we will stick to Linux Mint 18.1 for this particular tutorial.

STEP 1 First you need to ascertain which version of Python is currently installed in your Linux system; as we mentioned, we're going to be using Linux Mint 18.1 for this section. As with macOS, drop into a Terminal by pressing Ctrl+Atl+T.



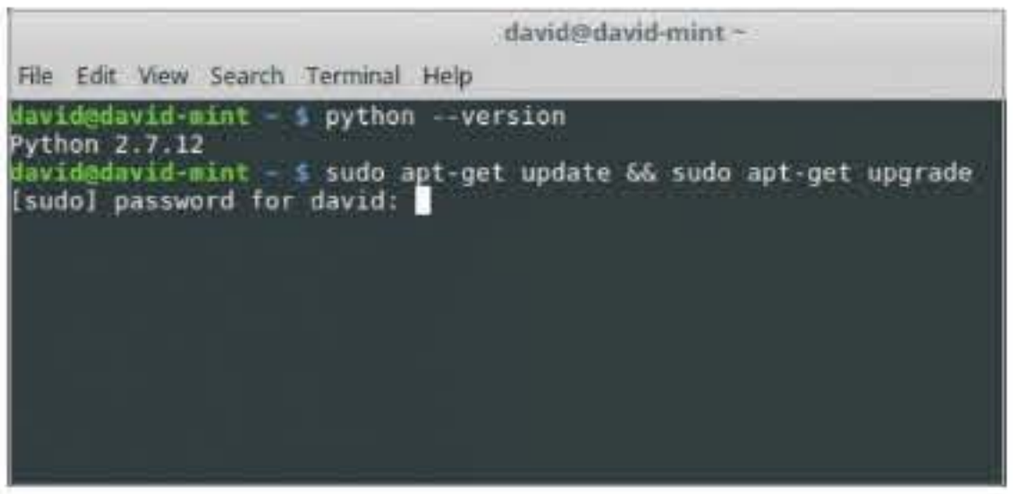
STEP 2 Next enter: `python --version` into the Terminal screen. You should have the output relating to version 2.x of Python in the display. Ours in this case if Python 2.7.12.



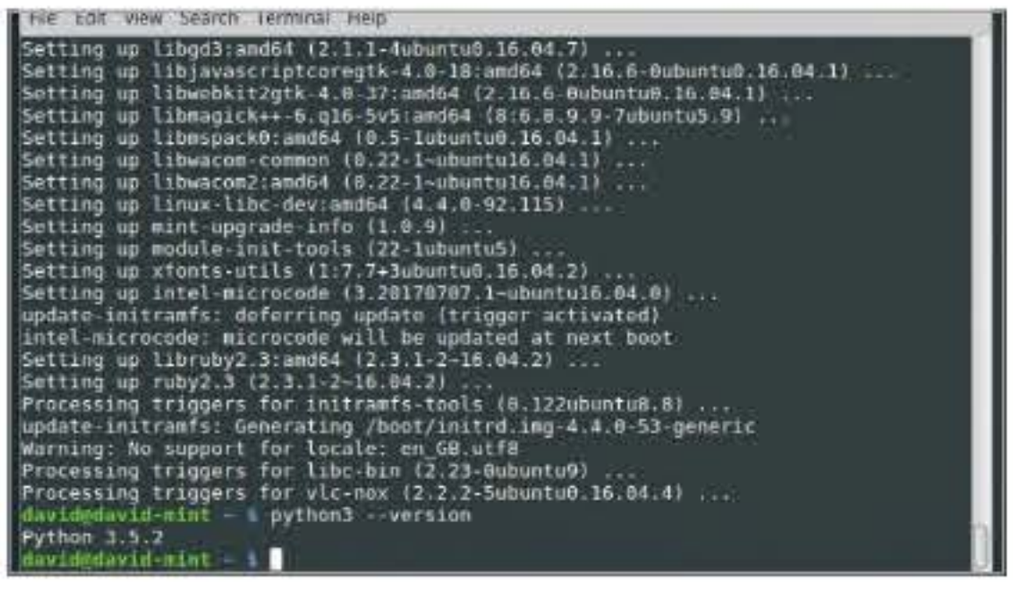
STEP 3 Some Linux distros will automatically update the installation of Python to the latest versions whenever the system is updated. To check, first do a system update and upgrade with:

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

Enter your password and let the system do any updates.

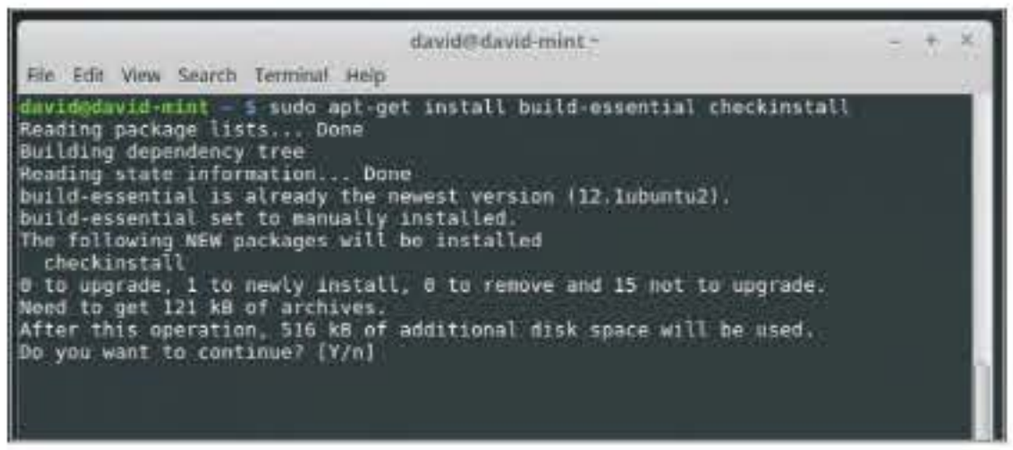


STEP 4 Once the update and upgrade is complete, you may need to answer 'Y' to authorise any upgrades, enter: `python3 --version` to see if Python 3.x is updated or even installed. In the case of Linux Mint, the version we have is Python 3.5.2, which is fine for our purposes.



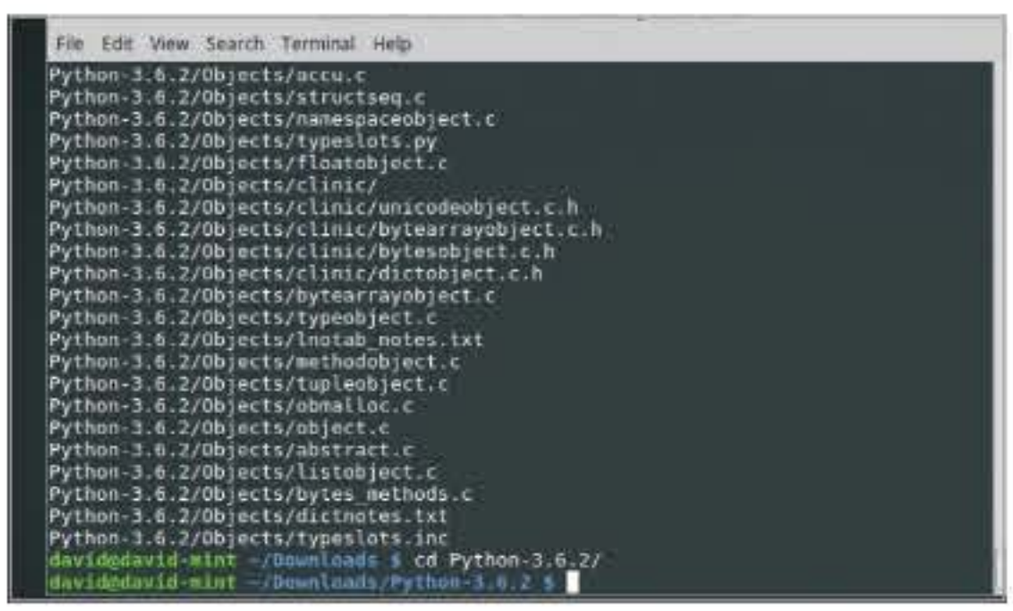
STEP 5 However, if you want the latest version, 3.6.2 as per the Python website at the time of writing, you need to build Python from source. Start by entering these commands into the Terminal:

```
sudo apt-get install build-essential checkinstall
sudo apt-get install libreadline-gplv2-dev
libncursesw5-dev libssl-dev libsqlite3-dev tk-dev
libgdbm-dev libc6-dev libbz2-dev
```



STEP 6 Open up your Linux web browser and go to the Python download page: www.python.org/downloads. Click on the Download Python 3.6.2 (or whichever version it's on when you look) to download the source Python-3.6.2.tar.xz file.

STEP 7 In the Terminal, go the Downloads folder by entering: `cd Downloads/`. Then unzip the contents of the downloaded Python source code with: `tar -xvf Python-3.6.2.tar.xz`. Now enter the newly unzipped folder with `cd Python-3.6.2/`.



STEP 8 Within the Python folder, enter:

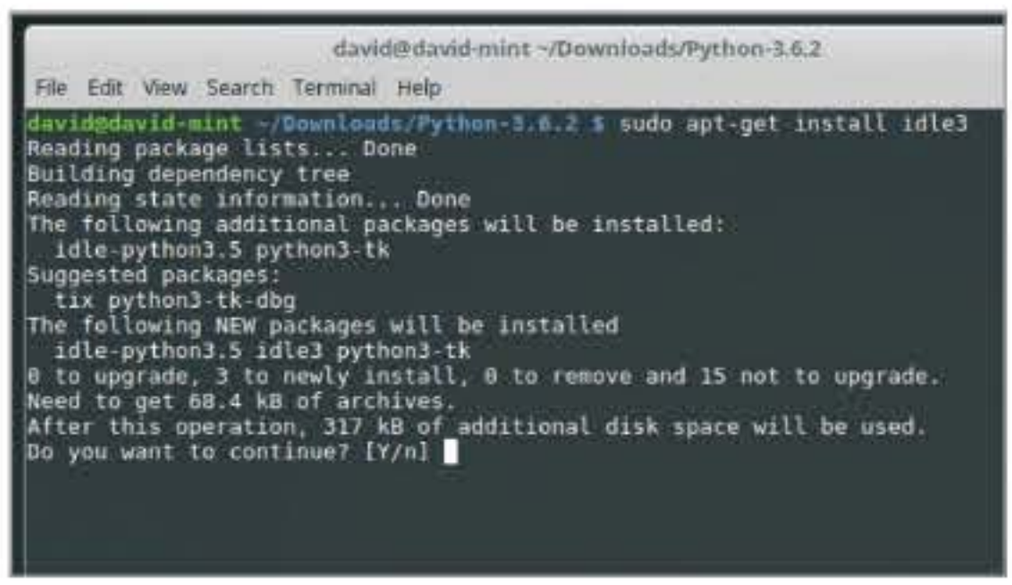
```
./configure
sudo make altinstall
```

This could a little while depending on the speed of your computer. Once finished, enter: `python3.6 --version` to check the installed latest version.

STEP 9 For the GUI IDLE, you need to enter the following command into the Terminal:

```
sudo apt-get install idle3
```

The IDLE can then be started with the command: `idle3`. Note, that IDLE runs a different version from the one you installed from source.

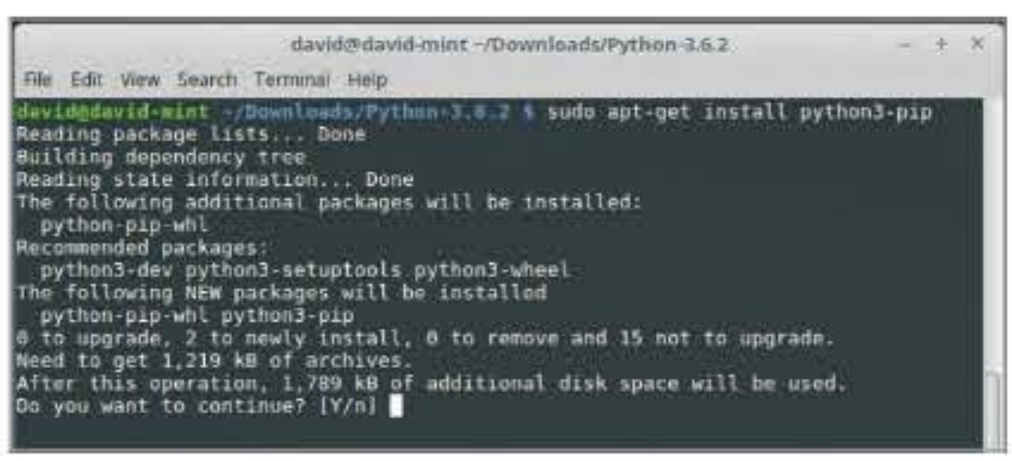


STEP 10 You also need PIP (Pip Installs Packages) which is a tool to help you install more modules and extras. Enter: `sudo apt-get install python3-pip`

PIP is then installed; check for the latest update with:

```
pip3 install --upgrade pip
```

When complete, close the Terminal and Python 3.x will be available via the Programming section in your distro's menu.



Installing a Text Editor

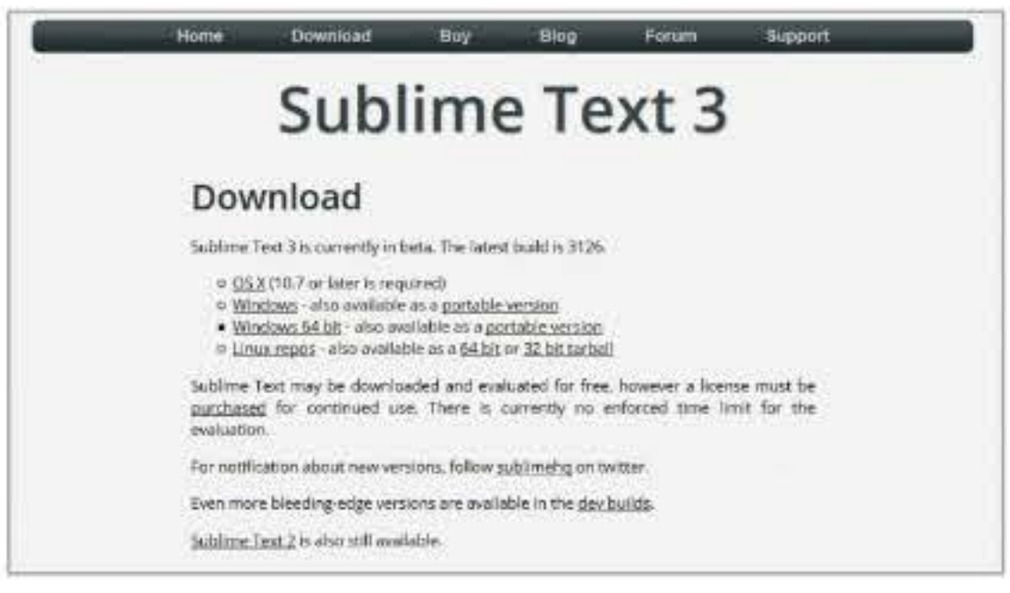
It's not entirely necessary (as you can use the IDLE) but a text editor will help you immensely when you're entering code. A normal word processor inserts its own unique characters, paragraph settings and much more, so it's not a good platform for Python code.

SUBLIME CODE

Sublime Text is an excellent, cross-platform text editor that's designed for entering code. It has a slick interface, many features and performs magnificently. In short, it's an ideal starting point.

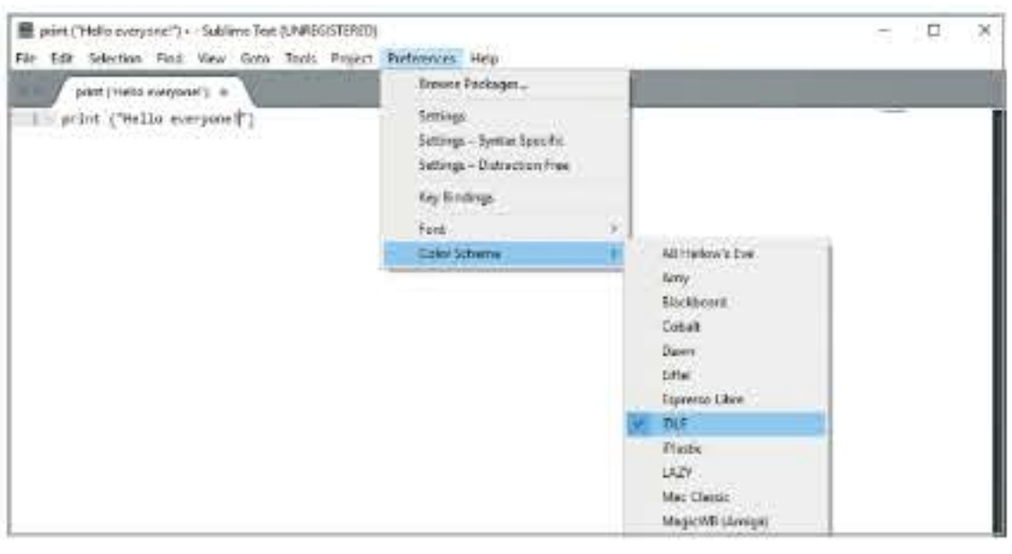
STEP 1 Let's begin by navigating to the Sublime Text webpage, to download the latest version for whatever operating system you're currently running. You can find the website at www.sublimetext.com, together with a download button for the detected OS that you're using.

STEP 2 However, if you want to specify a particular operating system version, then click on the Download link found in the top menu bar on the site. This will open a page with links for the latest version for OS X, Windows, Windows 64-bit and Linux machines.



STEP 3 Whichever version you choose, download the setup files and double-click them to begin the set up process. If you're using Windows, which we are in this instance, then you see the standard installation wizard. The defaults will suffice, so go ahead and install the program.

STEP 4 When installed, Sublime defaults to a black background and white text; whilst this is perfectly fine for most users, it's not always the most comfortable viewing setup. Thankfully, there are countless themes you can apply by clicking Preferences > Colour Scheme. We've opted for IDLE in this screenshot.



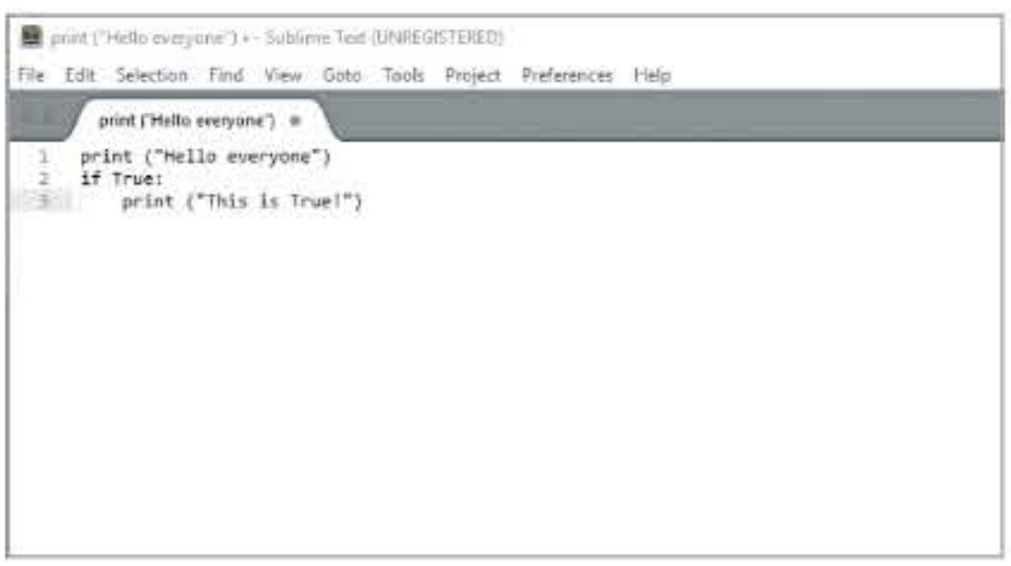
STEP 5 Sublime Text offers some excellent features over that of the standard Python IDLE. For example, enter the following:

```
print ("Hello everyone!")
```

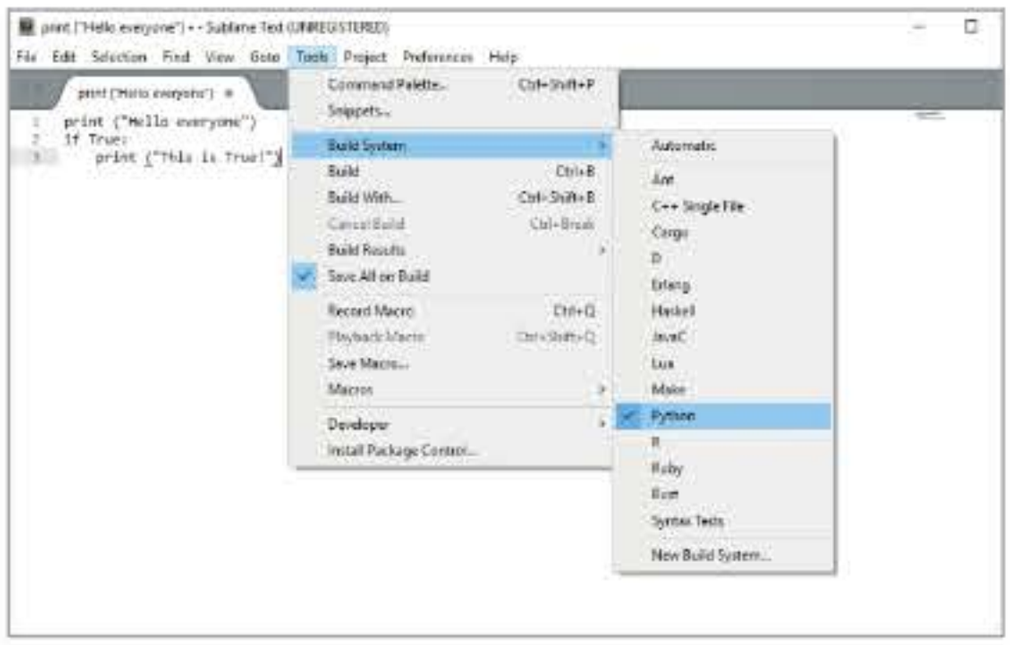
This is an actual Python command, which will print the words Hello everyone! on the screen. Notice how Sublime automatically recognises this as code and places the quotes and parentheses.

STEP 6 Soon, as you become more Python-savvy, you'll find that the standard IDLE isn't quite up to the task of keeping up with your code, alterations and injections of code mid-way through a long program. However, Sublime will keep you updated and you can even utilise indents easily.

STEP 7 We're not going to get too heavily into the code right now but an indent is part of Python programming, where a statement indicates that the following indented commands must be run until a particular event has happened; after which the indents stop. Pressing Ctrl+] will indent a line of code in Python.

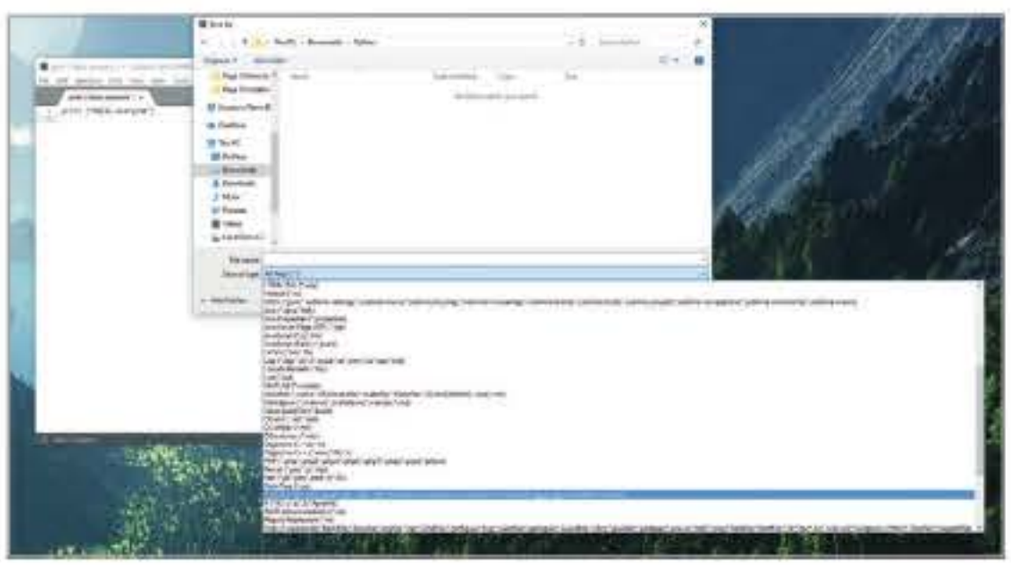


STEP 8 Sublime isn't just for Python either. With it you can build code for a number of programming languages. Click on Tools > Build System to see which languages you're able to build with in Sublime.

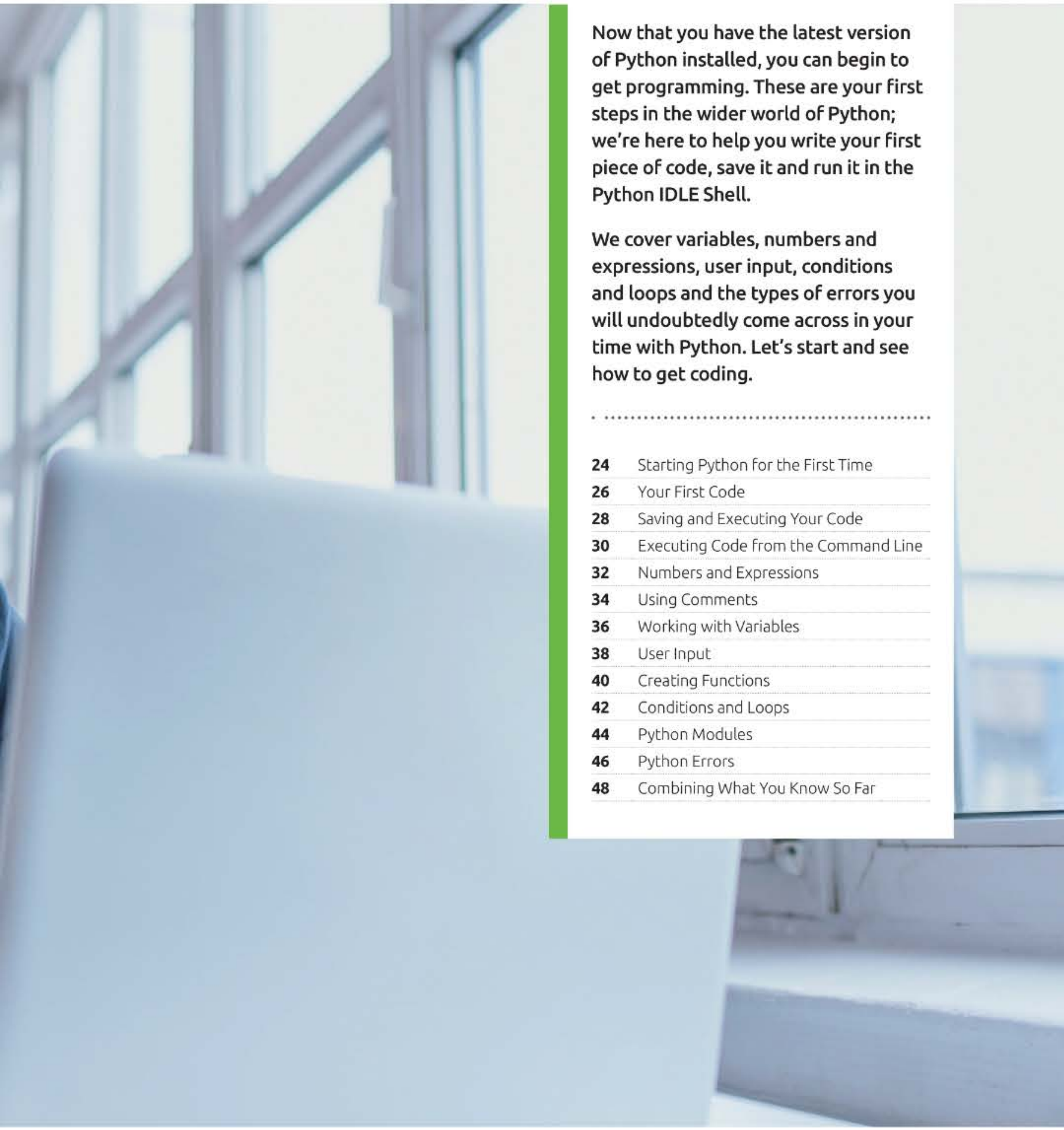


STEP 9 Sublime comes with a number of preinstalled plugins for Python code, allowing you to experiment with your code in real-time. They're probably a little bewildering at this point in time but you will likely find them useful as your Python skills increase.

STEP 10 However, we recommend you use the IDLE for this book. The Python IDLE isn't as advanced as Sublime but it's a perfect base on which to build your skills; and you don't have to worry about any extra features (as this is a Python book, not a Sublime book). Python code can be saved though, in its appropriate format within Sublime.



Getting Started with Python



Now that you have the latest version of Python installed, you can begin to get programming. These are your first steps in the wider world of Python; we're here to help you write your first piece of code, save it and run it in the Python IDLE Shell.

We cover variables, numbers and expressions, user input, conditions and loops and the types of errors you will undoubtedly come across in your time with Python. Let's start and see how to get coding.

24 Starting Python for the First Time
26 Your First Code
28 Saving and Executing Your Code
30 Executing Code from the Command Line
32 Numbers and Expressions
34 Using Comments
36 Working with Variables
38 User Input
40 Creating Functions
42 Conditions and Loops
44 Python Modules
46 Python Errors
48 Combining What You Know So Far

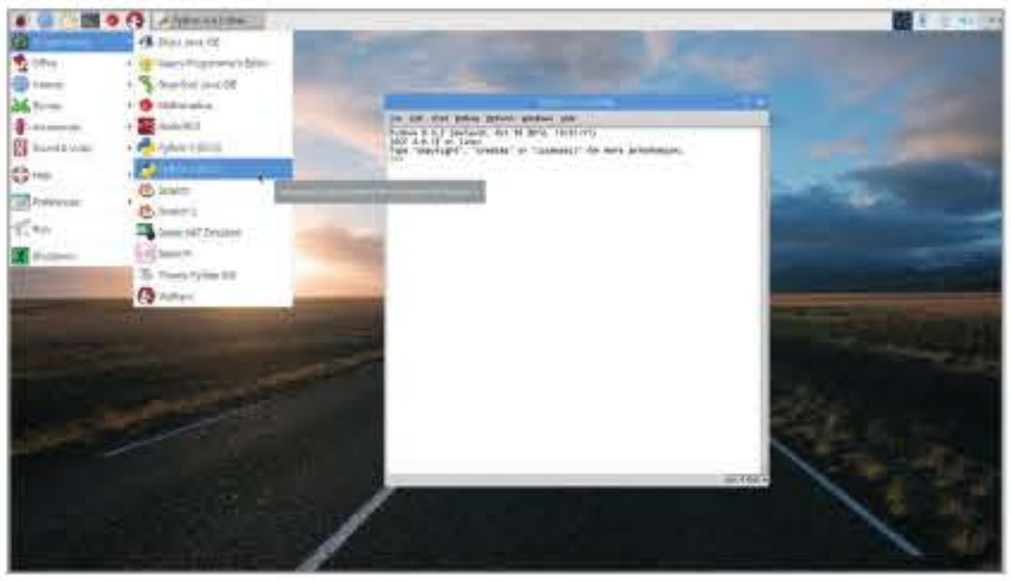
Starting Python for the First Time

As previously mentioned we're using the Raspberry Pi as our Python hardware platform. The latest version of Raspbian comes preinstalled with Python 3 (version 3.4.2); so as long as you have a version 3 Shell, all our code will work.

STARTING PYTHON

We won't go into the details of getting the Raspberry Pi up and running, there's plenty of material already available on that subject. However once you're ready, fire up your Pi and get ready for coding.

STEP 1 With the Raspbian desktop loaded, click on the Menu button followed by Programming > Python 3 (IDLE). This will open the Python 3 Shell. Windows and Mac users can find the Python 3 IDLE Shell from within the Windows Start button menu and via Finder.

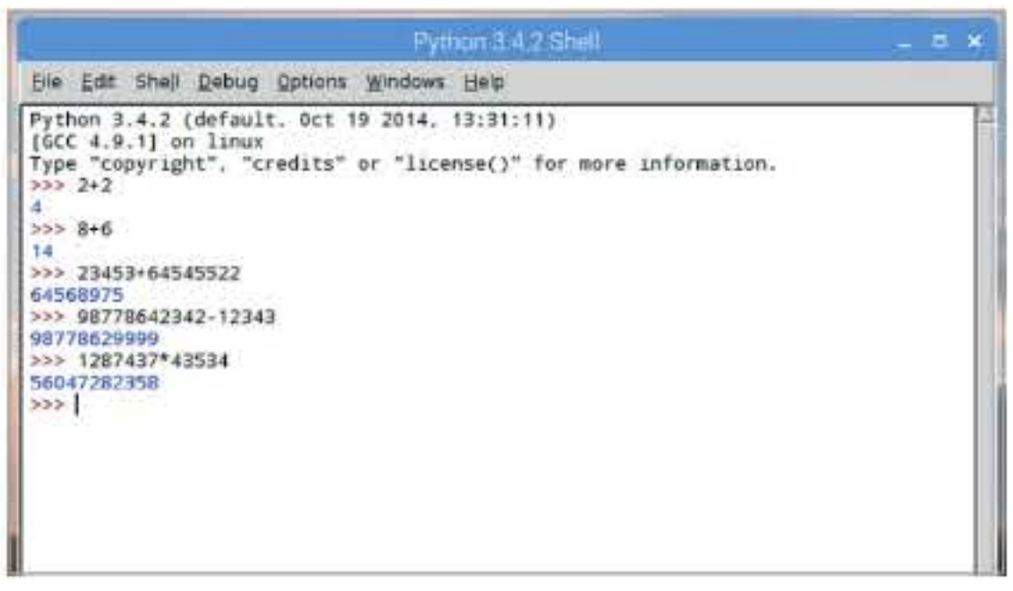


STEP 2 The Shell is where you can enter code and see the responses and output of code you've programmed into Python. This is a kind of sandbox, where you're able to try out some simple code and processes.

STEP 3 For example, in the Shell enter: 2+2

After pressing Enter, the next line will display the answer: 4. Basically, Python has taken the 'code' and produced the relevant output.

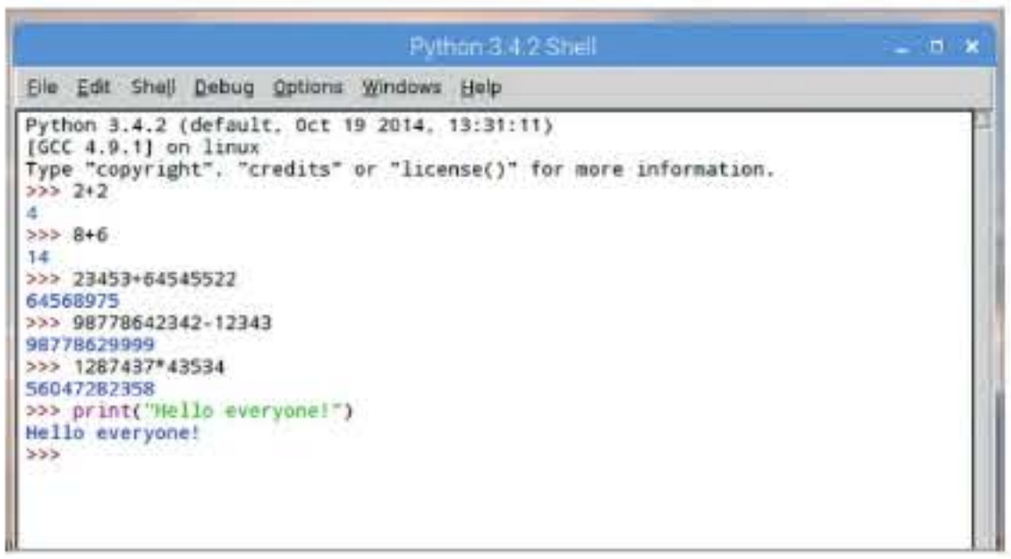
STEP 4 The Python Shell acts very much like a calculator, since code is basically a series of mathematical interactions with the system. Integers, which are the infinite sequence of whole numbers can easily be added, subtracted, multiplied and so on.



STEP 5 Whilst that's very interesting, it's nor particularly exciting. Instead, try this:

```
print("Hello everyone!")
```

As per the code we entered in Sublime in the Installing a Text Editor section of this book.



STEP 6 This is a little more like it, since you've just produced your first bit of code. The Print command is fairly self-explanatory, it prints things. Python 3 requires the brackets as well as quote marks in order to output content to the screen, in this case the Hello everyone! bit.

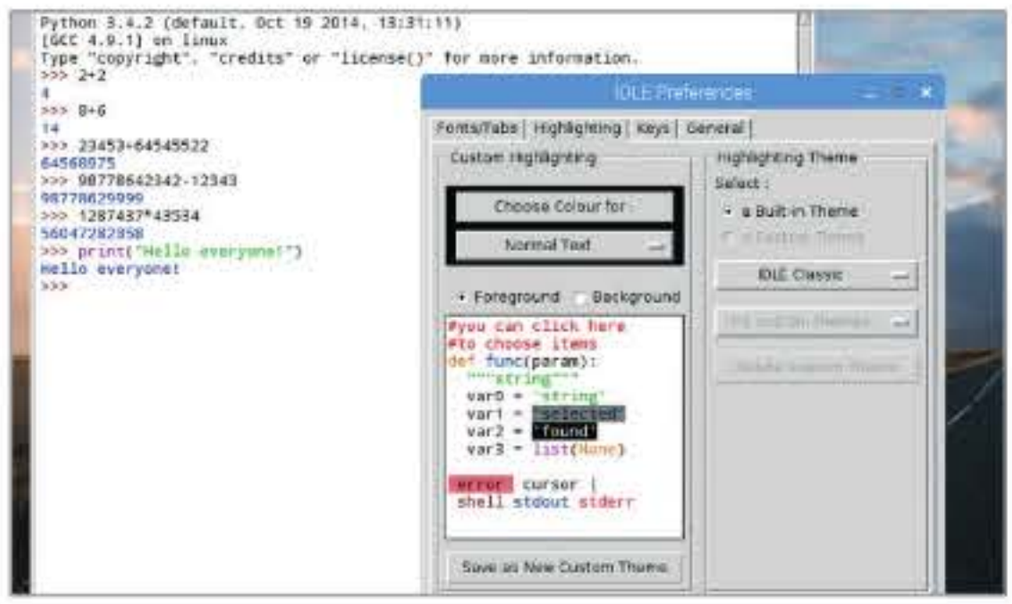
STEP 7 You may have noticed the colour coding within the Python IDLE. The colours represent different elements of Python code. They are:

Black – Data and Variables
Green – Strings
Purple – Functions
Orange – Commands
Blue – User Functions
Dark Red – Comments
Light Red – Error Messages

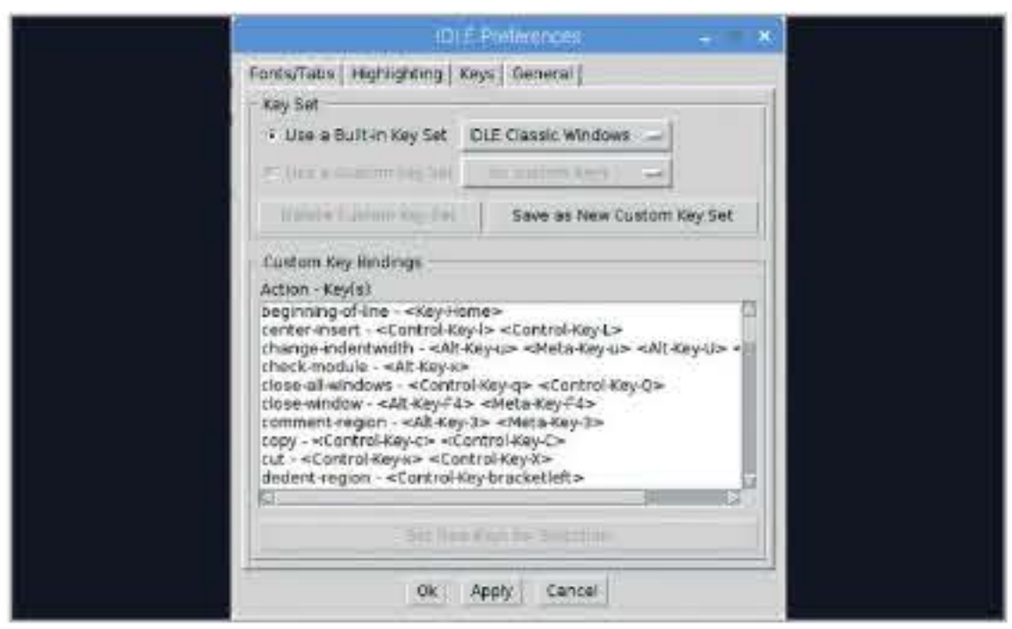
IDLE Colour Coding

Colour	Use for	Examples
Black	Data & variables	`23.6 area`
Green	Strings	`"Hello World"`
Purple	Functions	`len() print()`
Orange	Commands	`if for else`
Blue	User functions	`get_area()`
Dark red	Comments	`#Remember VAT`
Light red	Error messages	`SyntaxError:`

STEP 8 The Python IDLE is a configurable environment. If you don't like the way the colours are represented, then you can always change them via Options > Configure IDLE and clicking on the Highlighting tab. However, we don't recommend that as you won't be seeing the same as our screenshots.



STEP 9 Just like most programs available, regardless of the operating system, there are numerous shortcut keys available. We don't have room for them all here but within the Options > Configure IDLE and under the Keys tab, you can see a list of the current bindings.



STEP 10 The Python IDLE is a power interface, and one that's actually been written in Python using one of the available GUI toolkits. If you want to know the many ins and outs for the Shell, we recommend you take a few moments to view www.docs.python.org/3/library/idle.html, which details many of the IDLE's features.

Your First Code

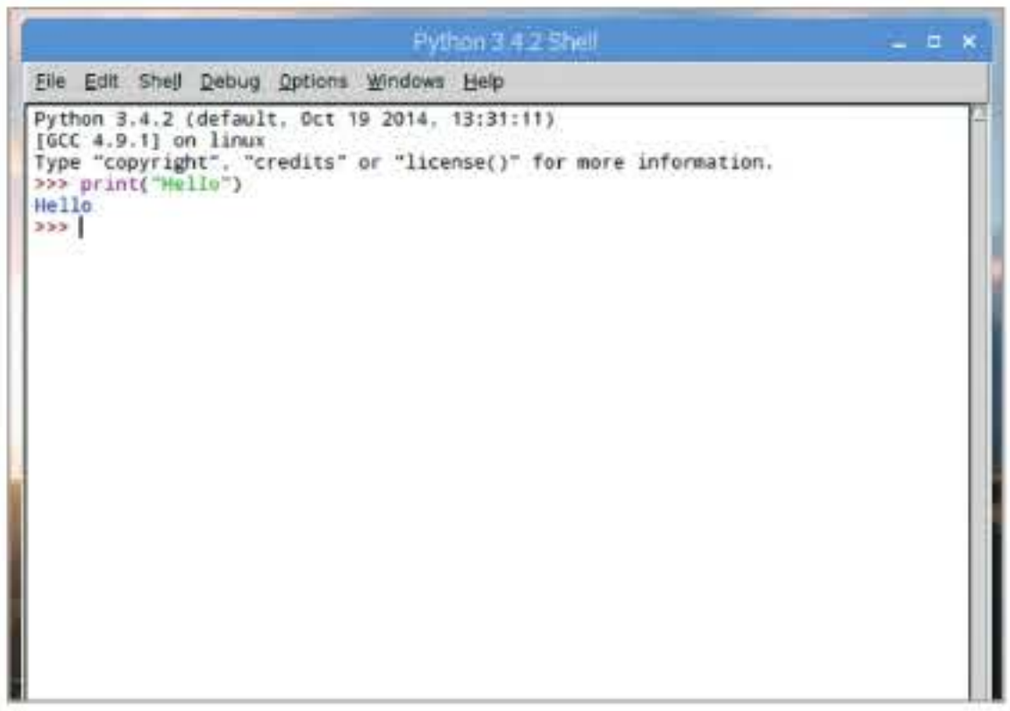
Essentially, you've already written your first piece of code with the 'print("Hello everyone!")' function from the previous tutorial. However, let's expand that and look at entering your code and playing around with some other Python examples.

PLAYING WITH PYTHON

With most languages, computer or human, it's all about remembering and applying the right words to the right situation. You're not born knowing these words, so you need to learn them.

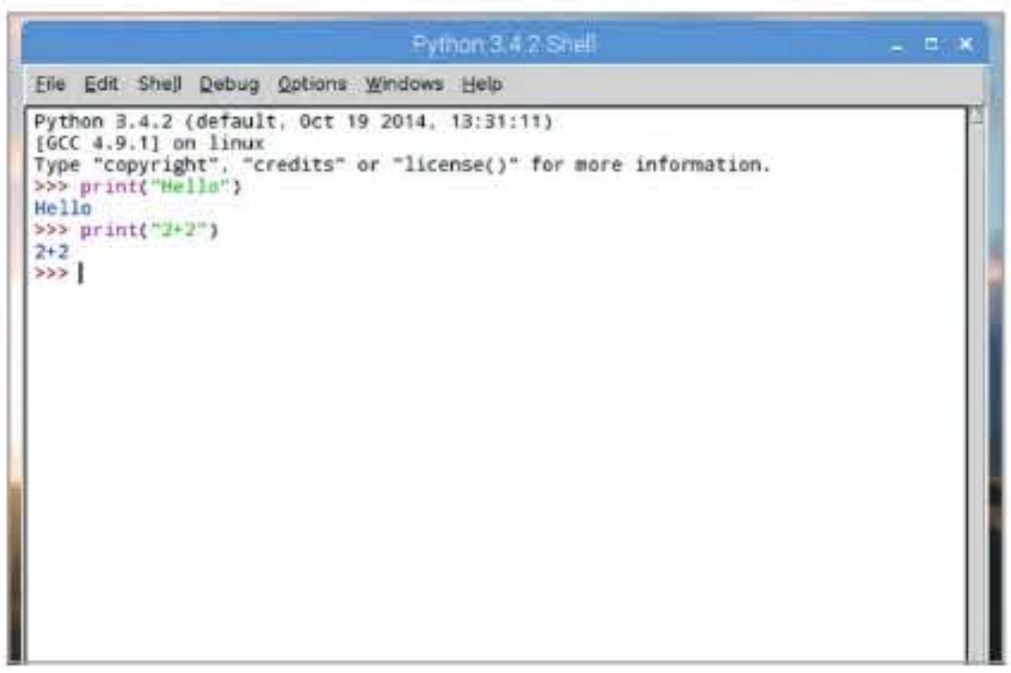
STEP 1 If you've closed Python 3 IDLE, reopen it in whichever operating system version you prefer. In the Shell, enter the familiar following:

```
print("Hello")
```



STEP 2 Just as predicted, the word Hello appears in the Shell as blue text, indicating output from a string. It's fairly straightforward and doesn't require too much explanation. Now try:

```
print("2+2")
```

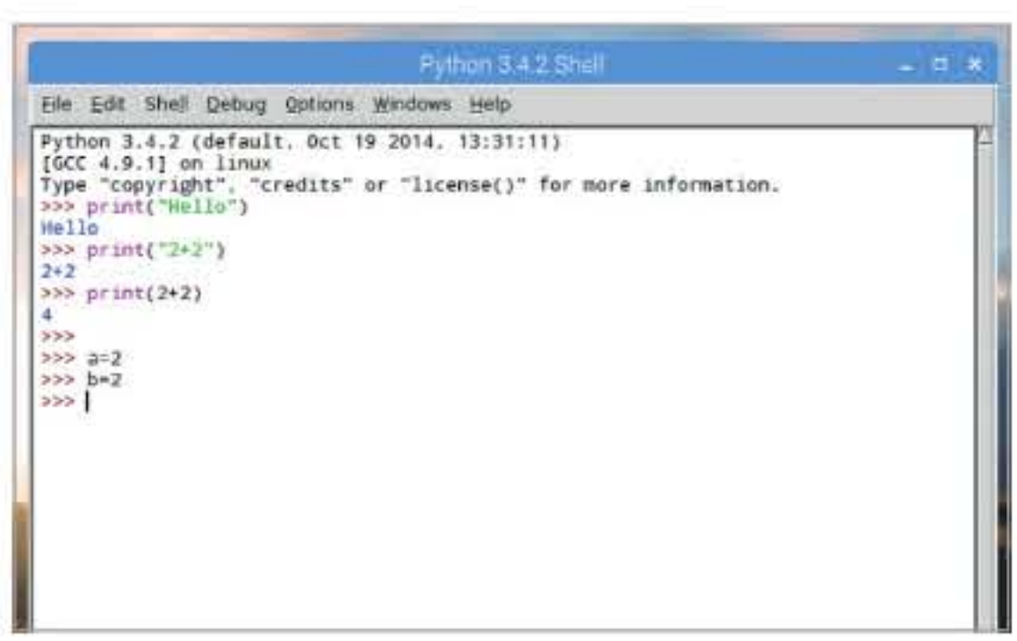


STEP 3 You can see that instead of the number 4, the output is the 2+2 you asked to be printed to the screen. The quotation marks are defining what's being outputted to the IDLE Shell; to print the total of 2+2 you need to remove the quotes:

```
print(2+2)
```

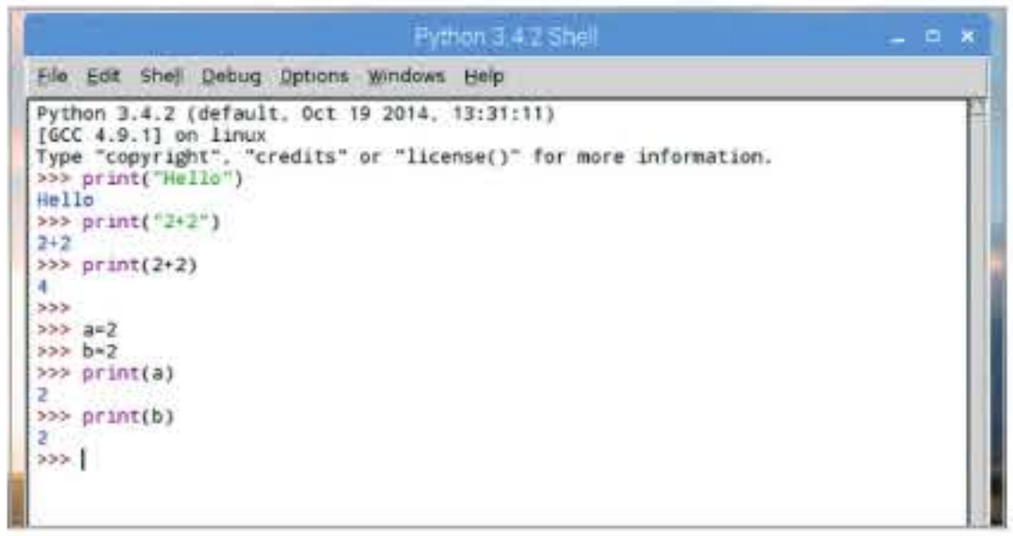
STEP 4 You can continue as such, printing 2+2, 464+2343 and so on to the Shell. An easier way is to use a variable, which is something we will cover in more depth later. For now, enter:

```
a=2
b=2
```



STEP 5 What you have done here is assign the letters a and b two values: 2 and 2. These are now variables, which can be called upon by Python to output, add, subtract, divide and so on for as long as their numbers stay the same. Try this:

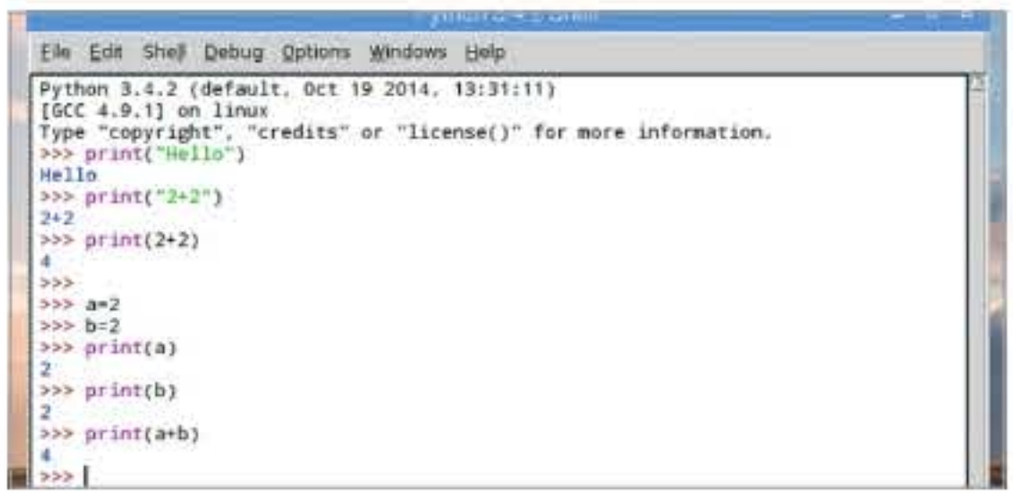
```
print(a)
print(b)
```



STEP 6 The output of the last step displays the current values of both a and b individually, as you've asked them to be printed separately. If you want to add them up, you can use the following:

```
print(a+b)
```

This code simply takes the values of a and b, adds them together and outputs the result.



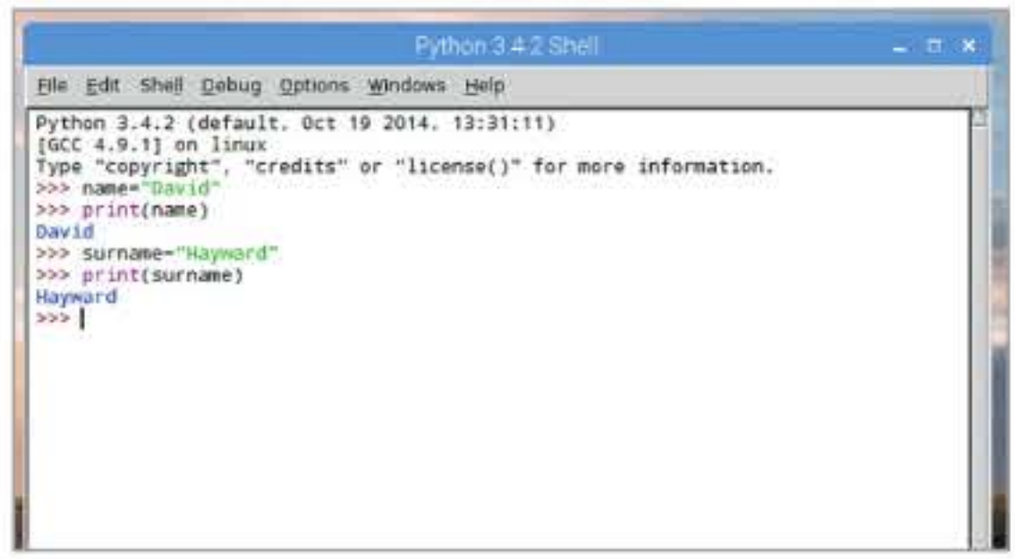
STEP 7 You can play around with different kinds of variables and the Print function. For example, you could assign variables for someone's name:

```
name="David"
print(name)
```

STEP 8 Now let's add a surname:

```
surname="Hayward"
print(surname)
```

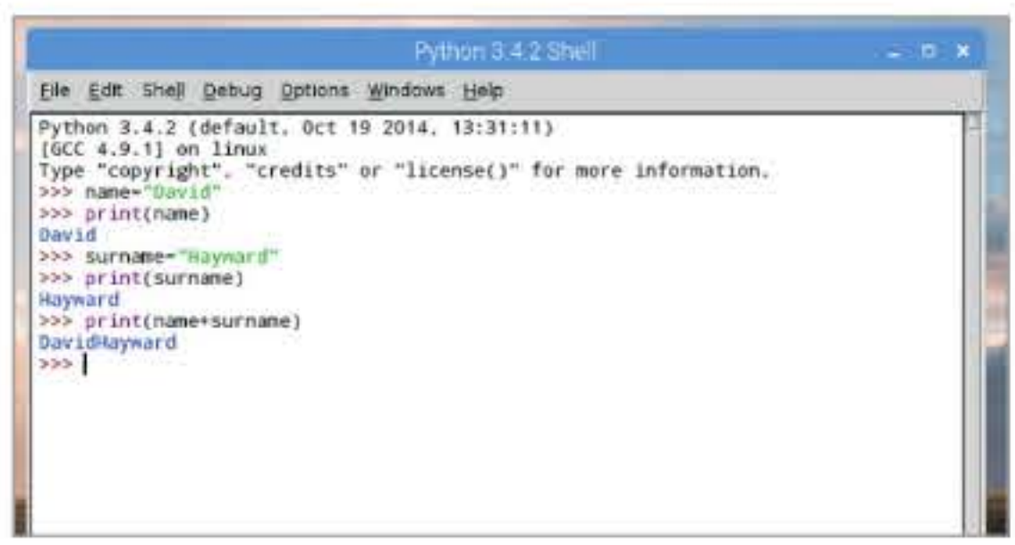
You now have two variables containing a first name and a surname and you can print them independently.



STEP 9 If we were to apply the same routine as before, using the + symbol, the name wouldn't appear correctly in the output in the Shell. Try it:

```
print(name+surname)
```

You need a space between the two, defining them as two separate values and not something you mathematically play around with.



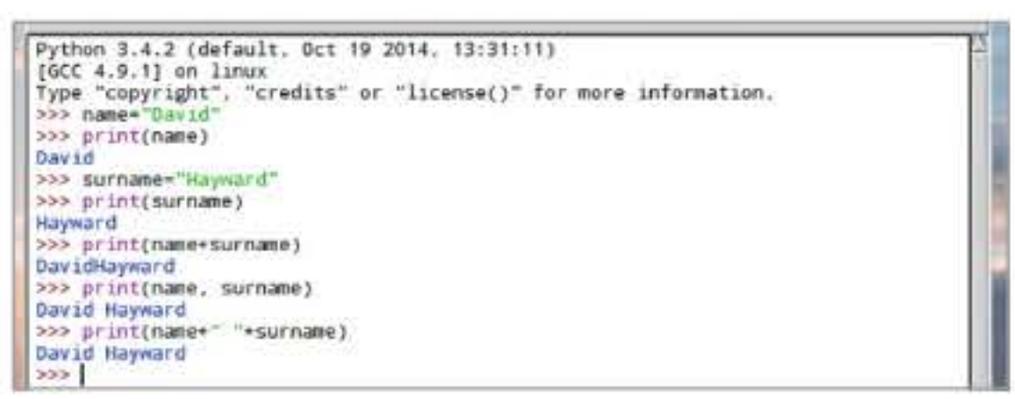
STEP 10 In Python 3 you can separate the two variables with a space using a comma:

```
print(name, surname)
```

Alternatively, you can add the space ourselves:

```
print(name+" "+surname)
```

The use of the comma is much neater, as you can see. Congratulations, you've just taken your first steps into the wide world of Python.



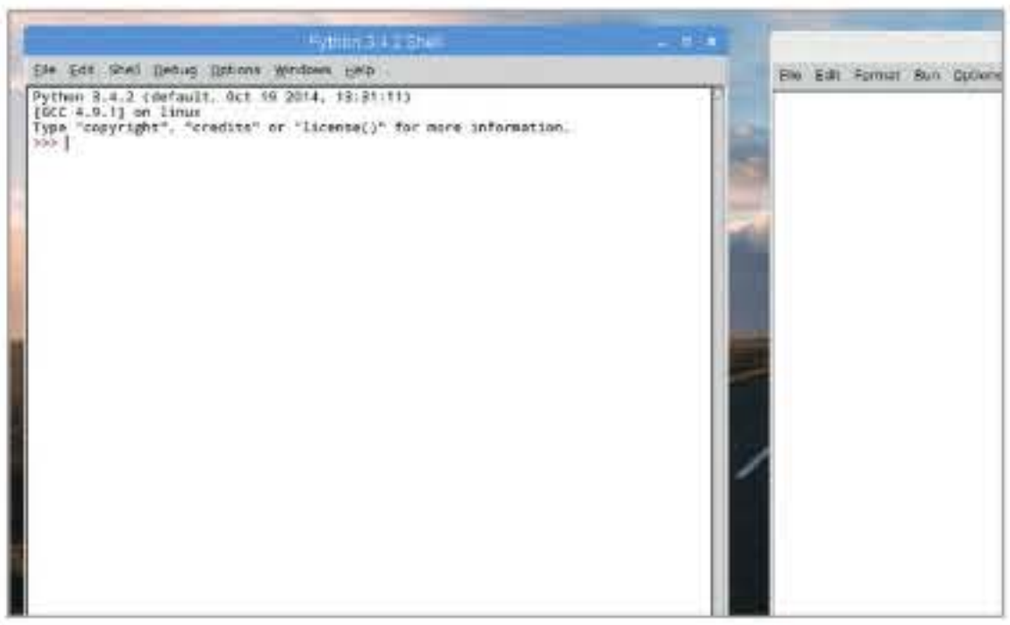
Saving and Executing Your Code

Whilst working in the IDLE Shell is perfectly fine for small code snippets, it's not designed for entering longer program listings. In this section you're going to be introduced to the IDLE Editor, where you will be working from now on.

EDITING CODE

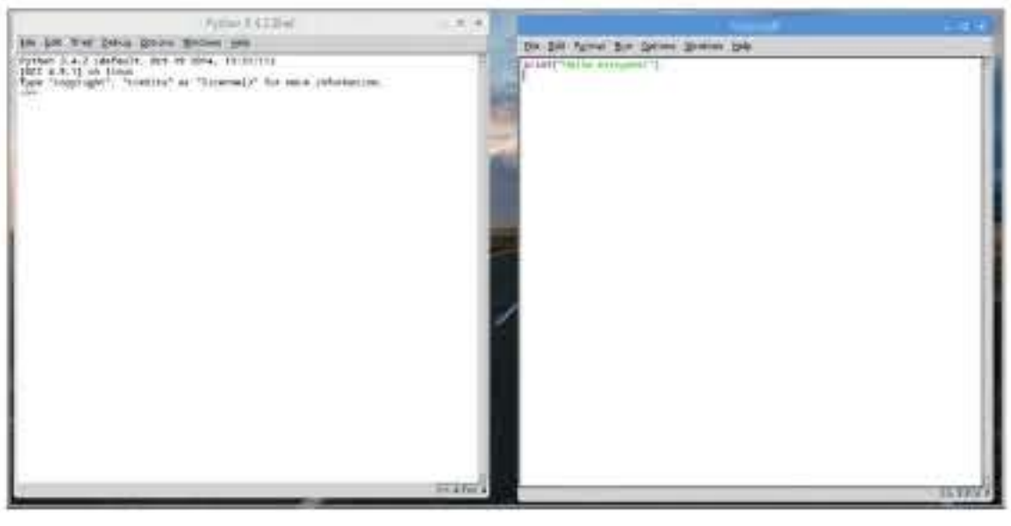
You will eventually reach a point where you have to move on from inputting single lines of code into the Shell. Instead, the IDLE Editor will allow you to save and execute your Python code.

STEP 1 First, open the Python IDLE Shell and when it's up, click on File > New File. This will open a new window with Untitled as its name. This is the Python IDLE Editor and within it you can enter the code needed to create your future programs.

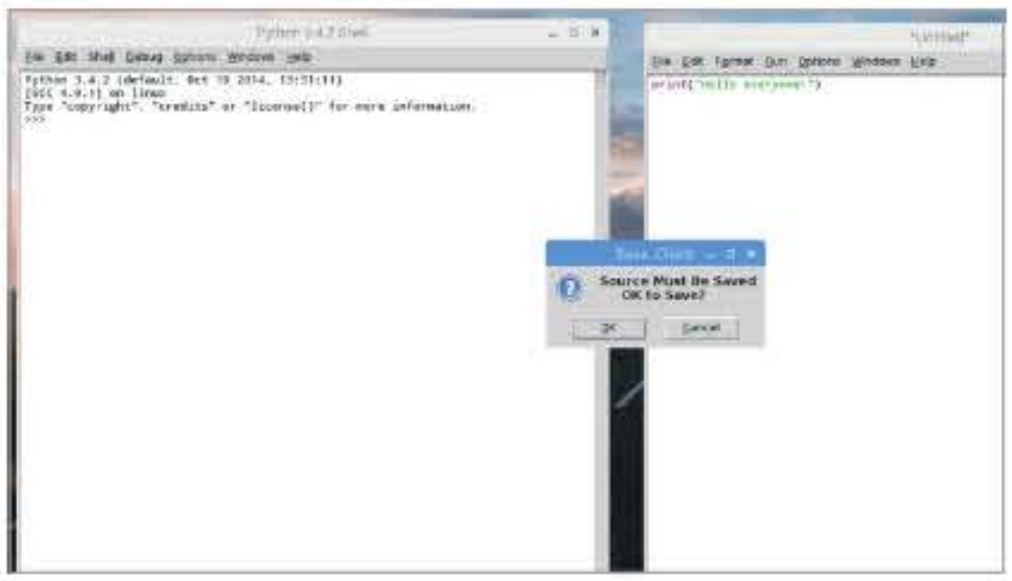


STEP 2 The IDLE Editor is, for all intents and purposes, a simple text editor with Python features, colour coding and so on; much in the same vein as Sublime. You enter code as you would within the Shell, so taking an example from the previous tutorial, enter:

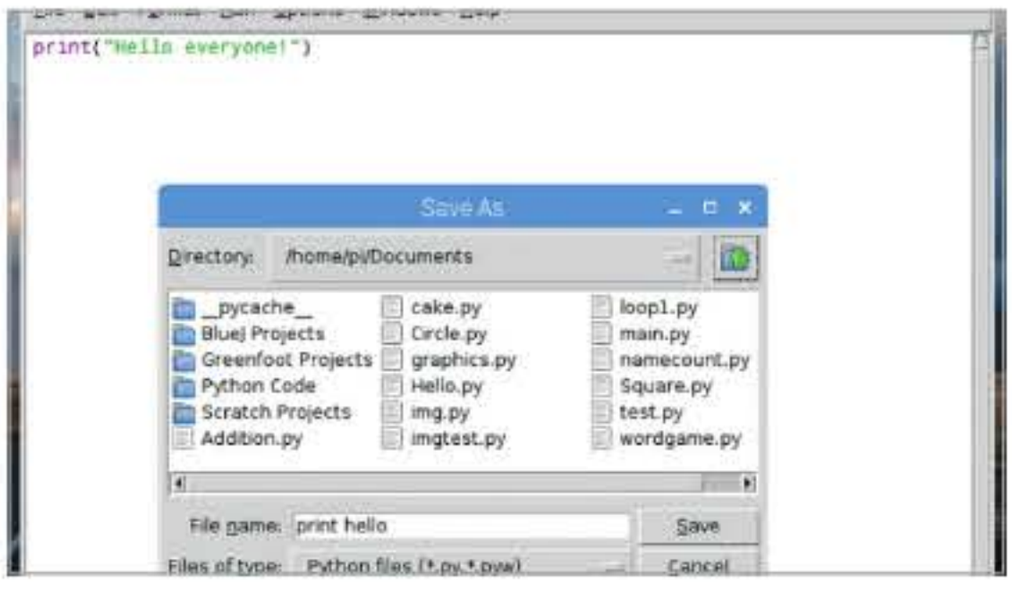
```
print("Hello everyone!")
```



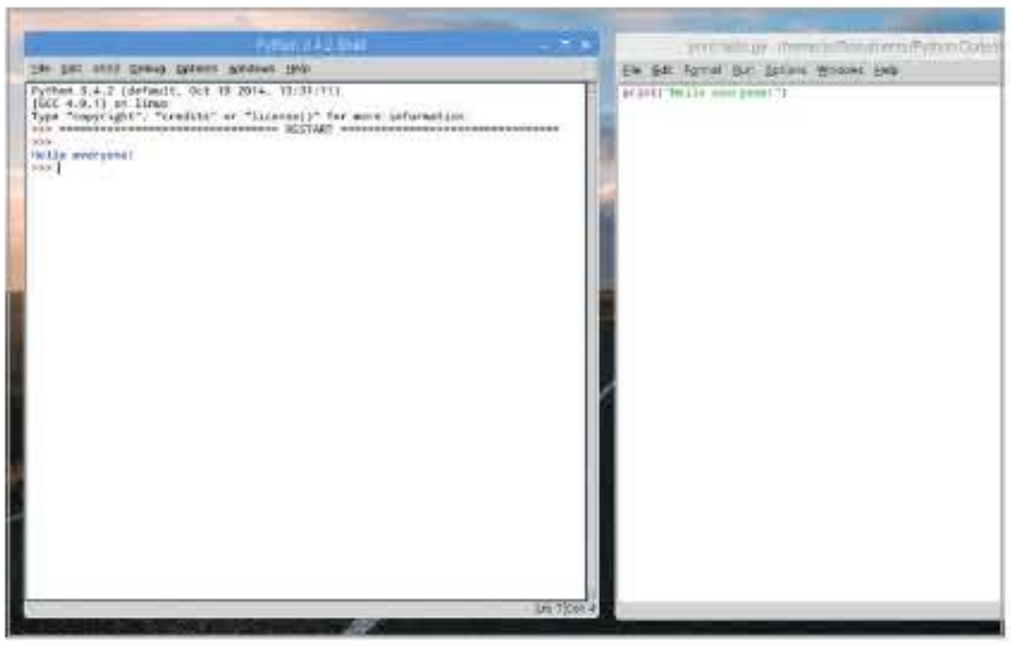
STEP 3 You can see that the same colour coding is in place in the IDLE Editor as it is in the Shell, enabling you to better understand what's going on with your code. However, to execute the code you need to first save it. Press F5 and you get a Save…Check box open.



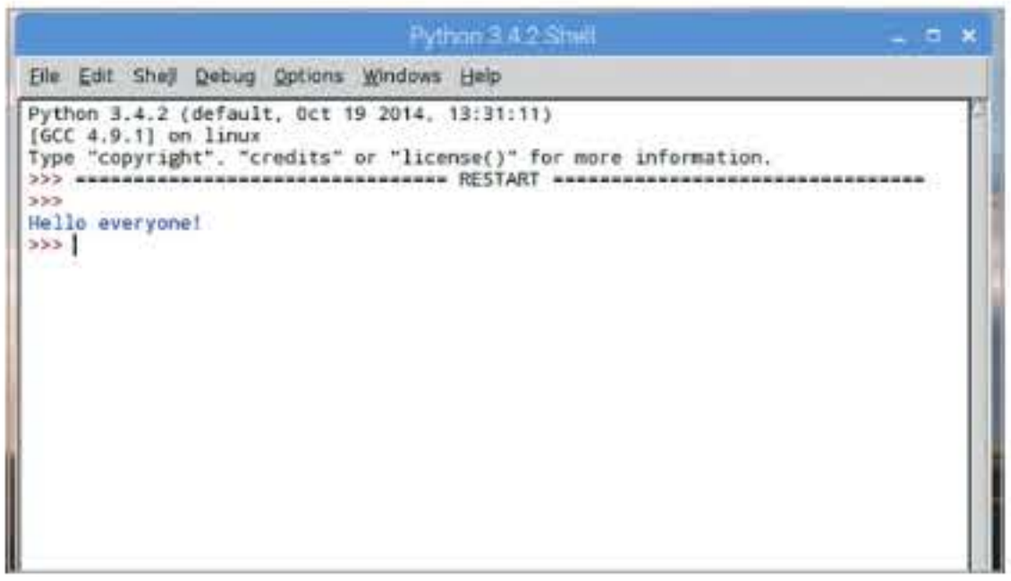
STEP 4 Click on the OK button in the Save box and select a destination where you'll save all your Python code. The destination can be a dedicated folder called Python or you can just dump it wherever you like. Remember to keep a tidy drive though, to help you out in the future.



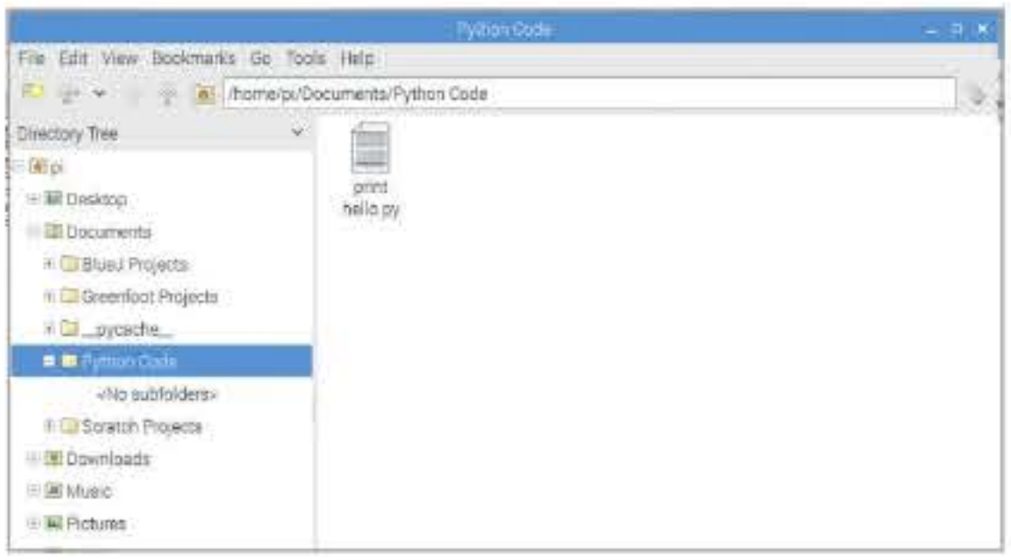
STEP 5 Enter a name for your code, 'print hello' for example, and click on the Save button. Once the Python code is saved it's executed and the output will be detailed in the IDLE Shell. In this case, the words 'Hello everyone!'.



STEP 6 This is how the vast majority of your Python code will be conducted. Enter it into the Editor, hit F5, save the code and look at the output in the Shell. Sometimes things will differ, depending on whether you've requested a separate window, but essentially that's the process. It's the process we will use throughout this book, unless otherwise stated.



STEP 7 If you open the file location of the saved Python code, you can see that it ends in a .py extension. This is the default Python file name. Any code you create will be whatever.py and any code downloaded from the many internet Python resource sites will be .py. Just ensure that the code is written for Python 3.

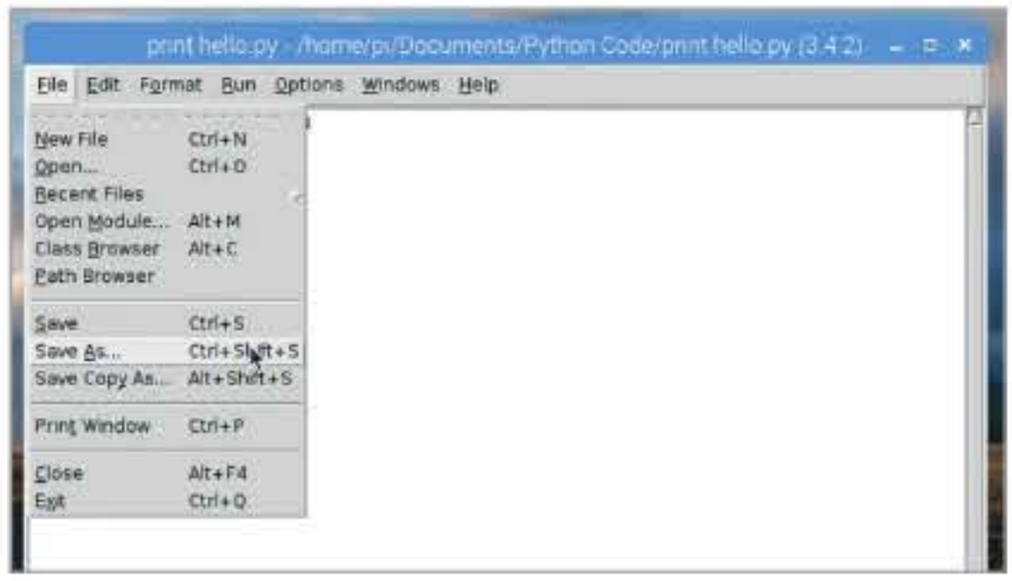


STEP 8 Let's extend the code and enter a few examples from the previous tutorial:

```
a=2
b=2
name="David"
surname="Hayward"
print(name, surname)
print (a+b)
```

If you press F5 now you'll be asked to save the file, again, as it's been modified from before.

STEP 9 If you click the OK button, the file will be overwritten with the new code entries, and executed, with the output in the Shell. It's not a problem with just these few lines but if you were to edit a larger file, overwriting can become an issue. Instead, use File > Save As from within the Editor to create a backup.



STEP 10 Now create a new file. Close the Editor, and open a new instance (File > New File from the Shell). Enter the following and save it as hello.py:

```
a="Python"
b="is"
c="cool!"
print(a, b, c)
```

You will use this code in the next tutorial.

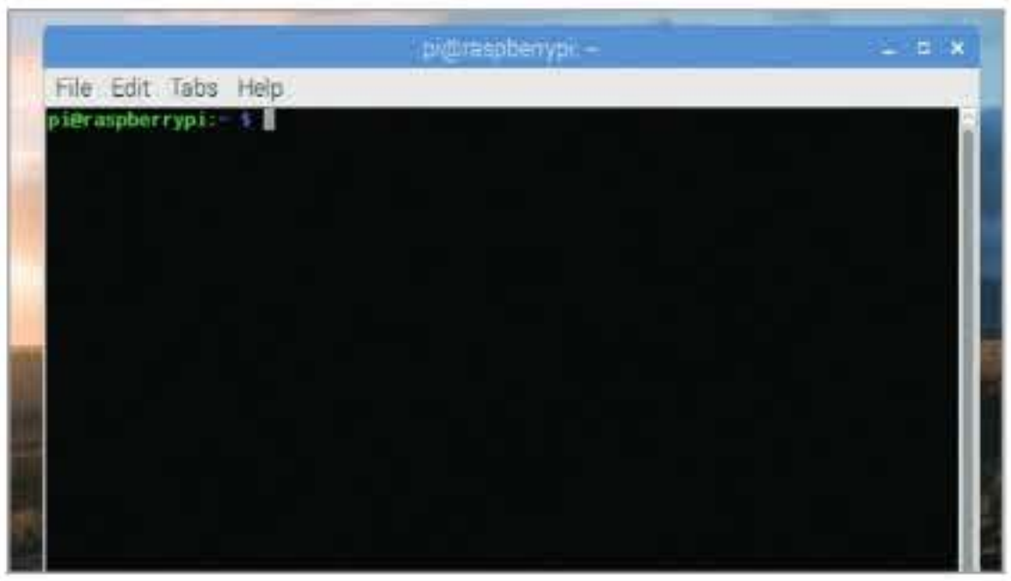
Executing Code from the Command Line

Although we're working from the GUI IDLE throughout this book, it's worth taking a look at Python's command line handling. We already know there's a command line version of Python but it's also used to execute code.

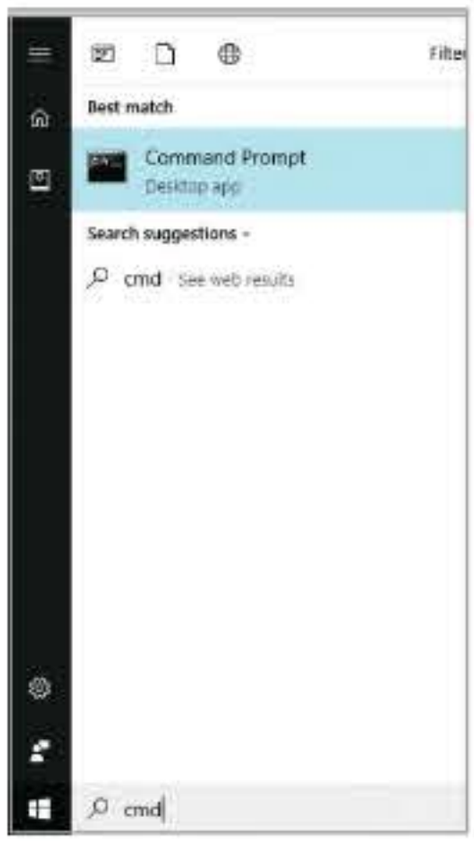
COMMAND THE CODE

Using the code we created in the previous tutorial, the one we named hello.py, let's see how you can run code that was made in the GUI at the command line level.

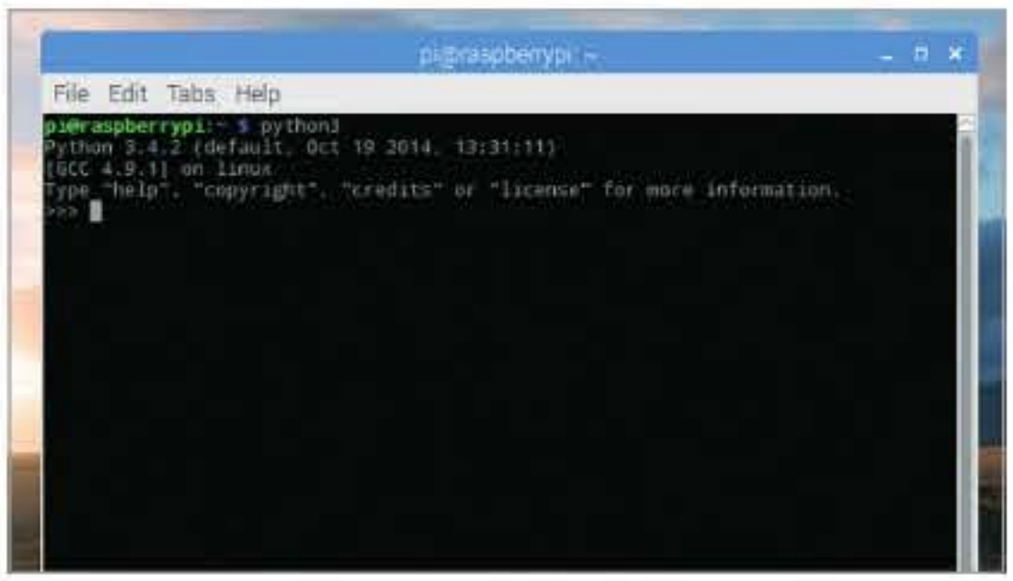
STEP 1 Python, in Linux, comes with two possible ways of executing code via the command line. One of the ways is with Python 2, whilst the other uses the Python 3 libraries and so on. First though, drop into the command line or Terminal on your operating system.



STEP 2 Just as before, we're using a Raspberry Pi: Windows users will need to click the Start button and search for CMD, then click the Command Line returned search; and macOS users can get access to their command line by clicking Go > Utilities > Terminal.



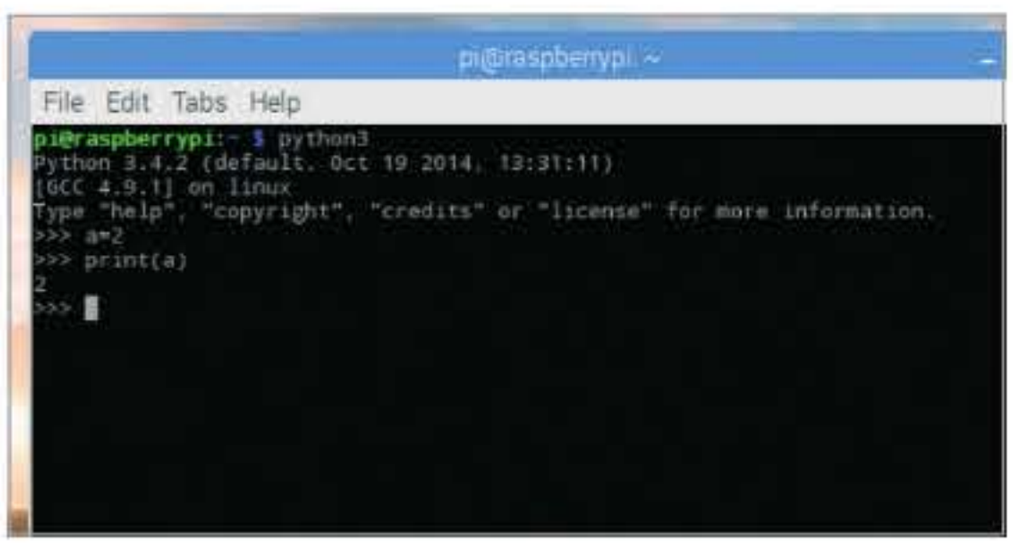
STEP 3 Now you're at the command line we can start Python. For Python 3 you need to enter the command **python3** and press Enter. This will put you into the command line version of the Shell, with the familiar three right-facing arrows as the cursor (>>>).



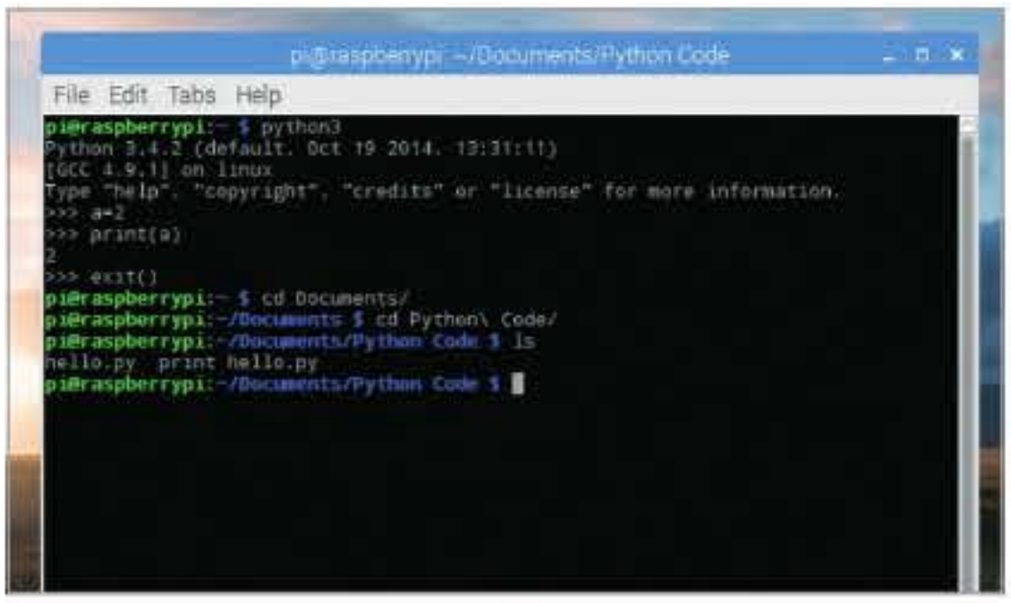
STEP 4 From here you're able to enter the code you've looked at previously, such as:

```
a=2
print(a)
```

You can see that it works exactly the same.



STEP 5 Now enter: **exit()** to leave the command line Python session and return you back to the command prompt. Enter the folder where you saved the code from the previous tutorial and list the available files within; hopefully you should see the hello.py file.



STEP 6 From within the same folder as the code you're going to run, enter the following into the command line:

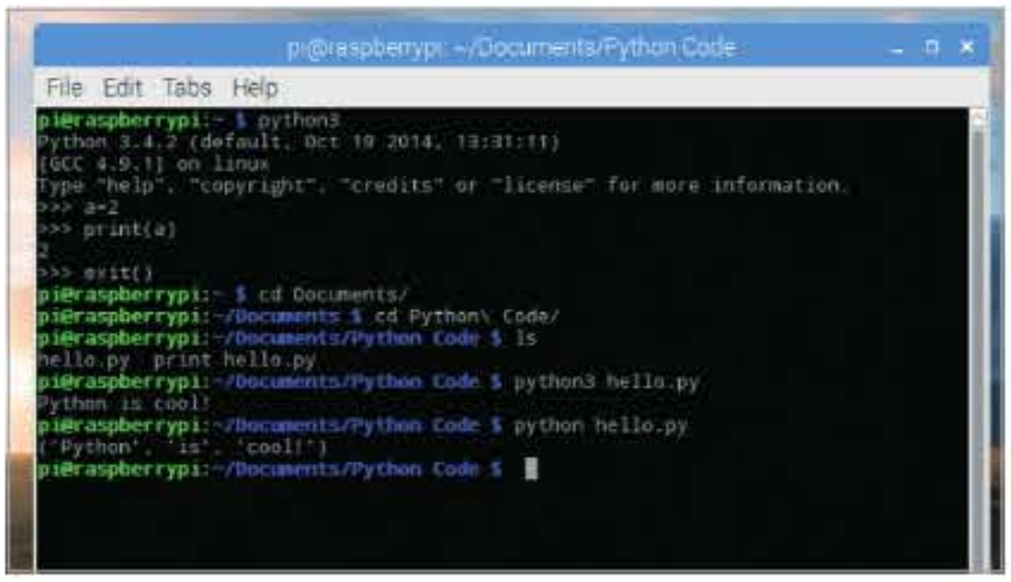
```
python3 hello.py
```

This will execute the code we created, which to remind you is:

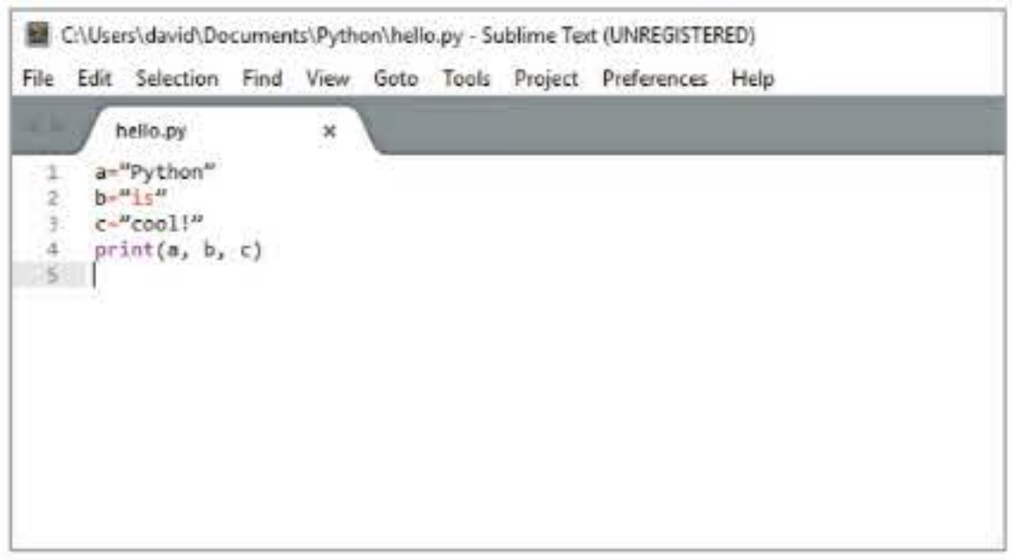
```
a="Python"
b="is"
c="cool!"
print(a, b, c)
```

STEP 7 Naturally, since this is Python 3 code, using the syntax and layout that's unique to Python 3, it only works when you use the python3 command. If you like, try the same with Python 2 by entering:

```
python hello.py
```



STEP 8 The result of running Python 3 code from the Python 2 command line is quite obvious. Whilst it doesn't error out in any way, due to the differences between the way Python 3 handles the Print command over Python 2, the result isn't as we expected. Using Sublime for the moment, open the hello.py file.



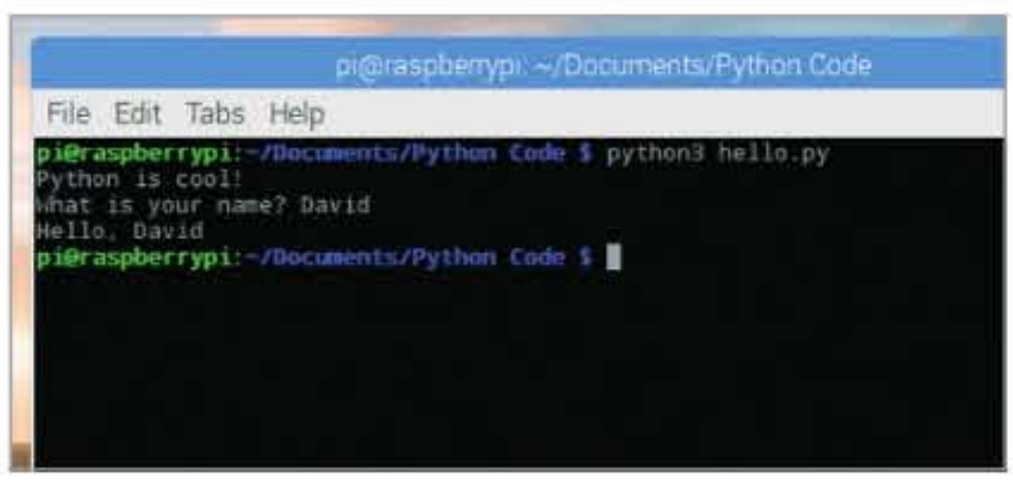
STEP 9 Since Sublime Text isn't available for the Raspberry Pi, you're going to temporarily leave the Pi for the moment and use Sublime as an example that you don't necessarily need to use the Python IDLE. With the hello.py file open, alter it to include the following:

```
name=input("What is your name? ")
print("Hello,", name)
```

STEP 10 Save the hello.py file and drop back to the command line. Now execute the newly saved code with:

```
python3 hello.py
```

The result will be the original Python is cool! statement, together with the added input command asking you for your name, and displaying it in the command window.



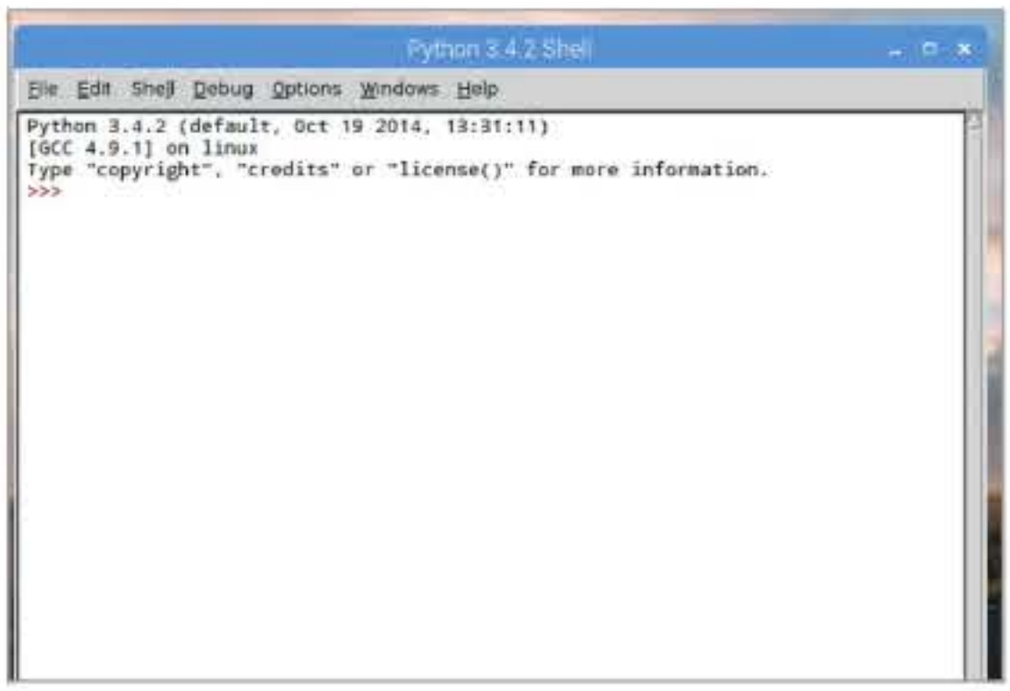
Numbers and Expressions

We've seen some basic mathematical expressions with Python, simple addition and the like. Let's expand on that now and see just how powerful Python is as a calculator. You can work within the IDLE Shell or in the Editor, whichever you like.

IT'S ALL MATHS, MAN

You can get some really impressive results with the mathematical powers of Python; as with most, if not all, programming languages, Maths is the driving force behind the code.

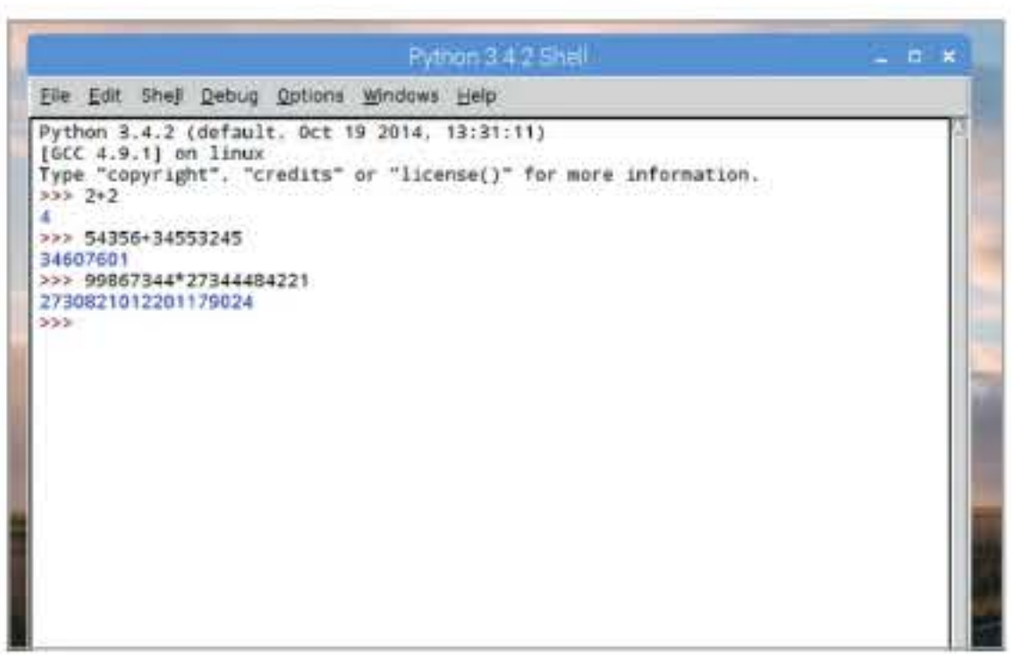
STEP 1 Open up the GUI version of Python 3, as mentioned you can use either the Shell or the Editor. For the time being, you're going to use the Shell just to warm our Maths muscle, which we believe is a small gland located at the back of the brain (or not).



STEP 2 In the Shell enter the following:

```
2+2
54356+34553245
99867344*27344484221
```

You can see that Python can handle some quite large numbers.



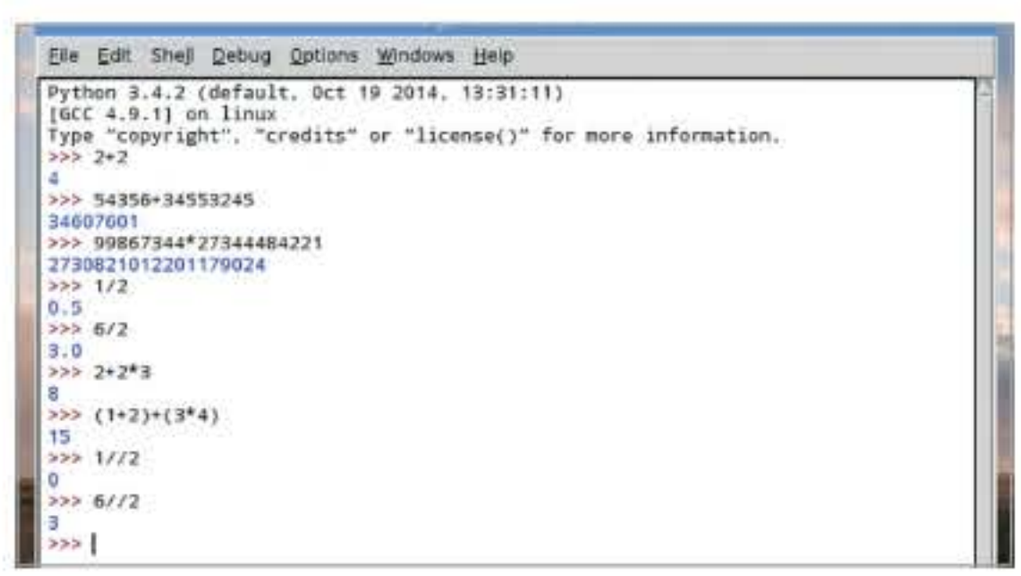
STEP 3 You can use all the usual Mathematical operations: divide, multiply, brackets and so on. Practise with a few, for example:

```
1/2
6/2
2+2*3
(1+2)+(3*4)
```

STEP 4 You've no doubt noticed, division produces a decimal number. In Python these are called floats, or floating point arithmetic. However, if you need an integer as opposed to a decimal answer, then you can use a double slash:

```
1//2
6//2
```

And so on.



STEP 5 You can also use an operation to see the remainder left over from division. For example:

```
10/3
```

Will display 3.333333333, which is of course 3.3-recurring. If you now enter:

```
10%3
```

This will display 1, which is the remainder left over from dividing 10 into 3.

```
2730821012201179024
>>> 1/2
0.5
>>> 6/2
3.0
>>> 2+2*3
8
>>> (1+2)+(3*4)
15
>>> 1//2
0
>>> 6//2
3
>>> 10/3
3.3333333333333335
>>> 10%3
1
```

STEP 6 Next up we have the power operator, or exponentiation if you want to be technical. To work out the power of something you can use a double multiplication symbol or double-star on the keyboard:

```
2**3
10**10
```

Essentially, it's 2x2x2 but we're sure you already know the basics behind Maths operators. This is how you would work it out in Python.

```
>>> 2+2*3
8
>>> (1+2)+(3*4)
15
>>> 1//2
0
>>> 6//2
3
>>> 10/3
3.3333333333333335
>>> 10%3
1
>>> 2**3
8
>>> 10**10
10000000000
>>>
```

STEP 7 Numbers and expressions don't stop there. Python has numerous built-in functions to work out sets of numbers, absolute values, complex numbers and a host of mathematical expressions and Pythagorean tongue-twisters. For example, to convert a number to binary, use:

```
bin(3)
```

```
>>> 1/2
0.5
>>> 6/2
3.0
>>> 2+2*3
8
>>> (1+2)+(3*4)
15
>>> 1//2
0
>>> 6//2
3
>>> 10/3
3.3333333333333335
>>> 10%3
1
>>> 2**3
8
>>> 10**10
10000000000
>>> bin(3)
'0b11'
>>>
```

STEP 8 This will be displayed as '0b11', converting the integer into binary and adding the prefix 0b to the front. If you want to remove the 0b prefix, then you can use:

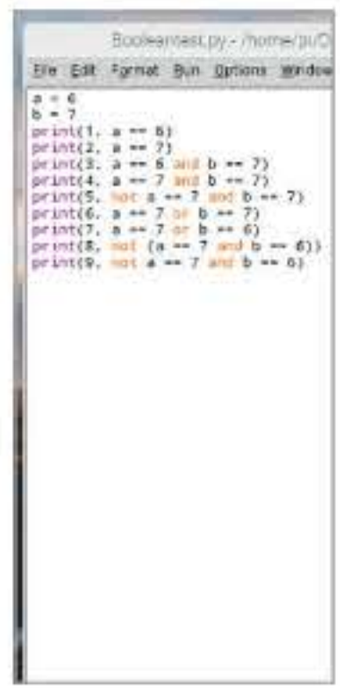
```
format(3, 'b')
```

The Format command converts a value, the number 3, to a formatted representation as controlled by the format specification, the 'b' part.

```
>>> 2+2*3
8
>>> (1+2)+(3*4)
15
>>> 1//2
0
>>> 6//2
3
>>> 10/3
3.3333333333333335
>>> 10%3
1
>>> 2**3
8
>>> 10**10
10000000000
>>> bin(3)
'0b11'
>>> format(3,'b')
'11'
>>>
```

STEP 9 A Boolean Expression is a logical statement that will either be true or false. We can use these to compare data and test to see if it's equal to, less than or greater than. Try this in a New File:

```
a = 6
b = 7
print(1, a == 6)
print(2, a == 7)
print(3, a == 6 and b == 7)
print(4, a == 7 and b == 7)
print(5, not a == 7 and b == 7)
print(6, a == 7 or b == 7)
print(7, a == 7 or b == 6)
print(8, not (a == 7 and b == 6))
print(9, not a == 7 and b == 6)
```



STEP 10 Execute the code from Step 9, and you can see a series of True or False statements, depending on the result of the two defining values: 6 and 7. It's an extension of what you've looked at, and an important part of programming.

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
1 True
2 False
3 True
4 False
5 True
6 True
7 False
8 True
9 False
>>>
```

Using Comments

When writing your code, the flow of it, what each variable does, how the overall program will operate and so on is all inside your head. Another programmer could follow the code line by line but over time, it can become difficult to read.

#COMMENTS!

Programmers use a method of keeping their code readable by commenting on certain sections. If a variable is used, the programmer comments on what it's supposed to do, for example. It's just good practise.

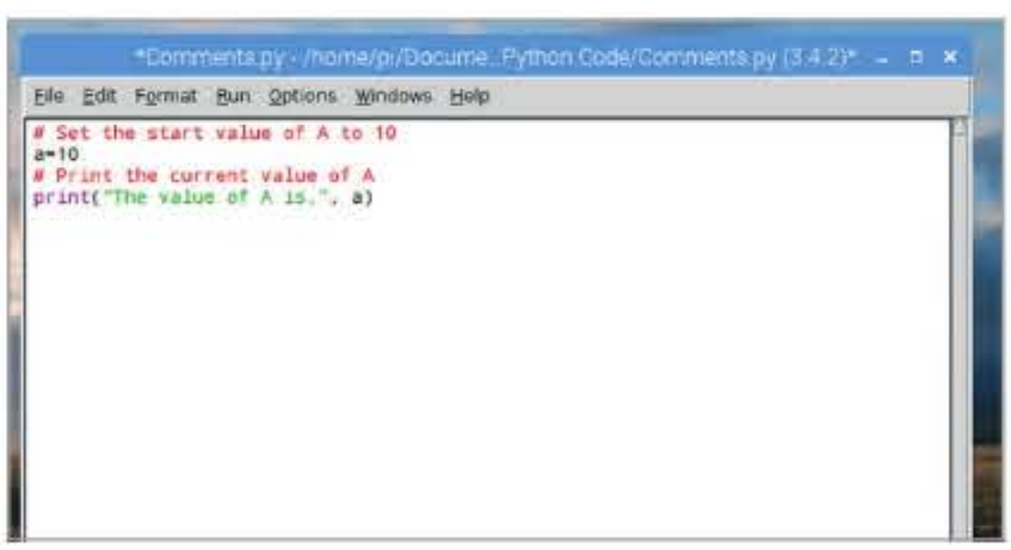
STEP 1 Start by creating a new instance of the IDLE Editor (File > New File) and create a simple variable and print command:

```
a=10
print("The value of A is,", a)
```

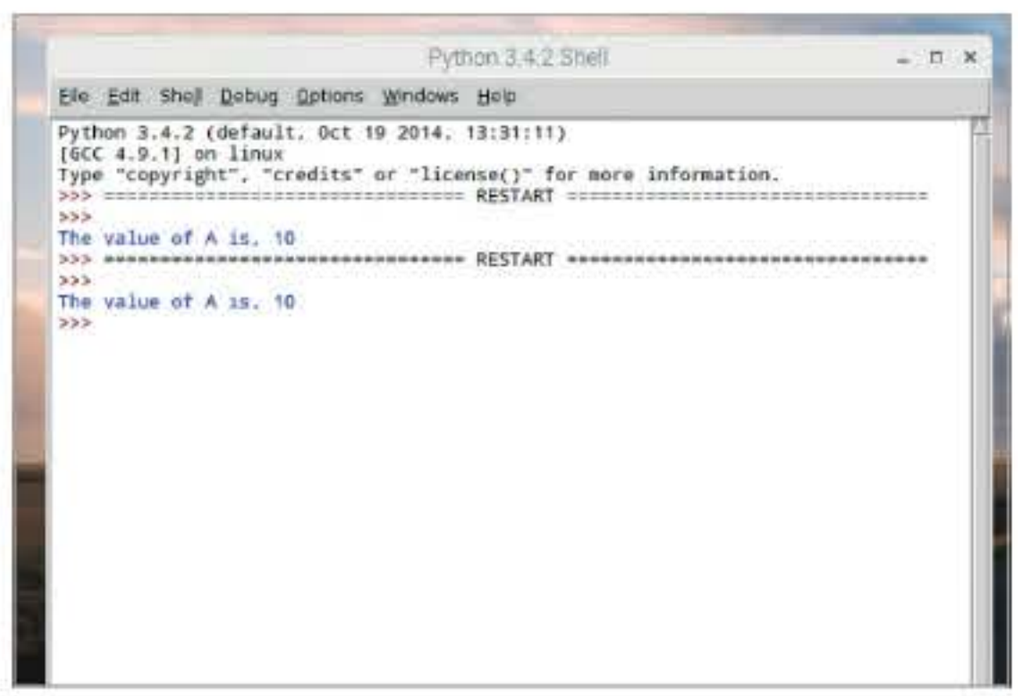
Save the file and execute the code.

STEP 2 Running the code will return the line: The value of A is, 10 into the IDLE Shell window, which is what we expected. Now, add some of the types of comments you'd normally see within code:

```
# Set the start value of A to 10
a=10
# Print the current value of A
print("The value of A is,", a)
```

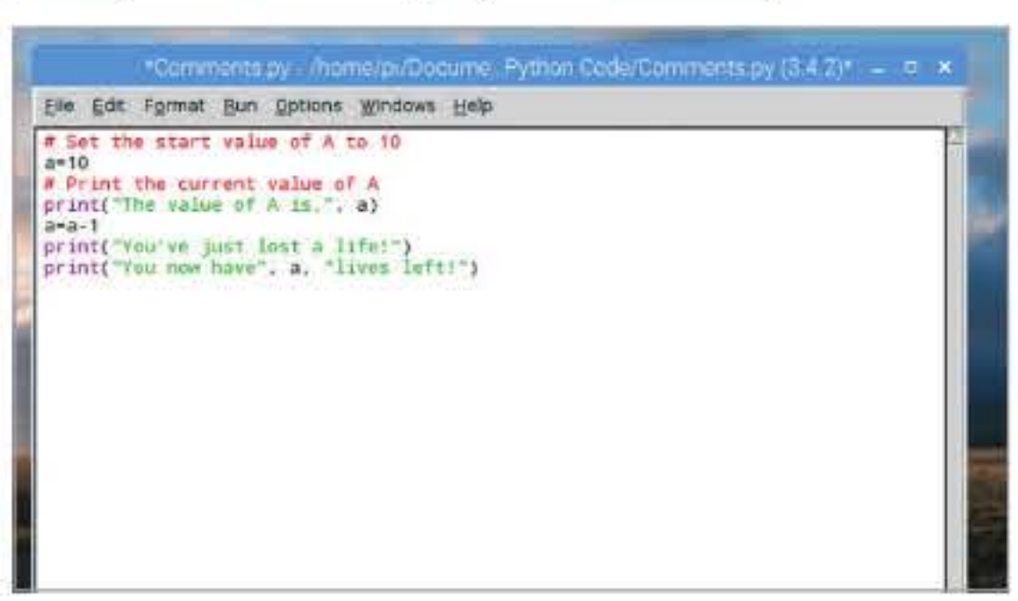


STEP 3 Resave the code and execute it. You can see that the output in the IDLE Shell is still the same as before, despite the extra lines being added. Simply put, the hash symbol (#) denotes a line of text the programmer can insert to inform them, and others, of what's going on without the user being aware.

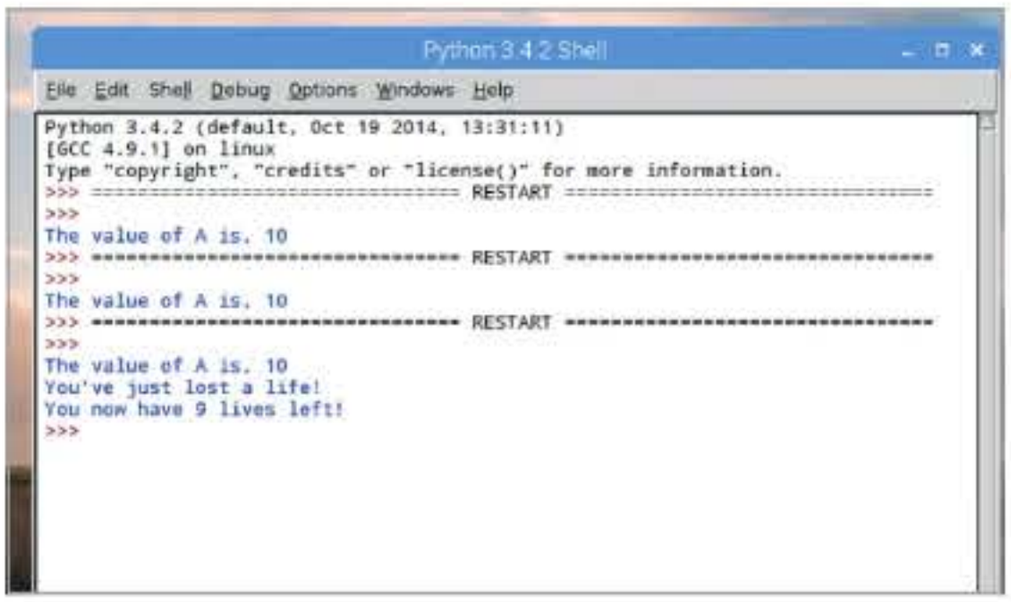


STEP 4 Let's assume that the variable A that we've created is the number of lives in a game. Every time the player dies, the value is decreased by 1. The programmer could insert a routine along the lines of:

```
a=a-1
print("You've just lost a life!")
print("You now have", a, "lives left!")
```

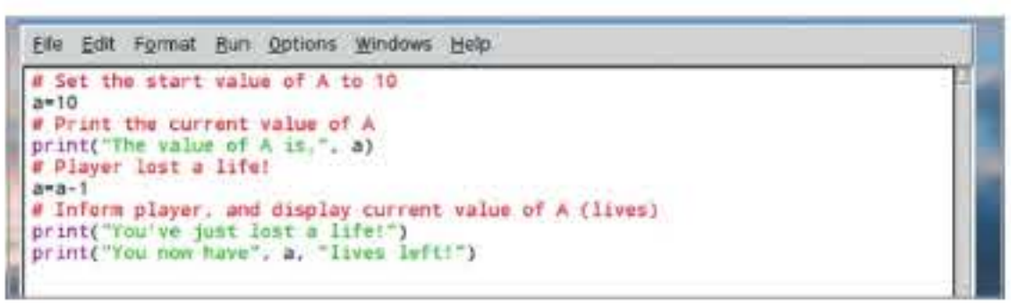


STEP 5 Whilst we know that the variable A is lives, and that the player has just lost one, a casual viewer or someone checking the code may not know. Imagine for a moment that the code is twenty thousand lines long, instead of just our seven. You can see how handy comments are.



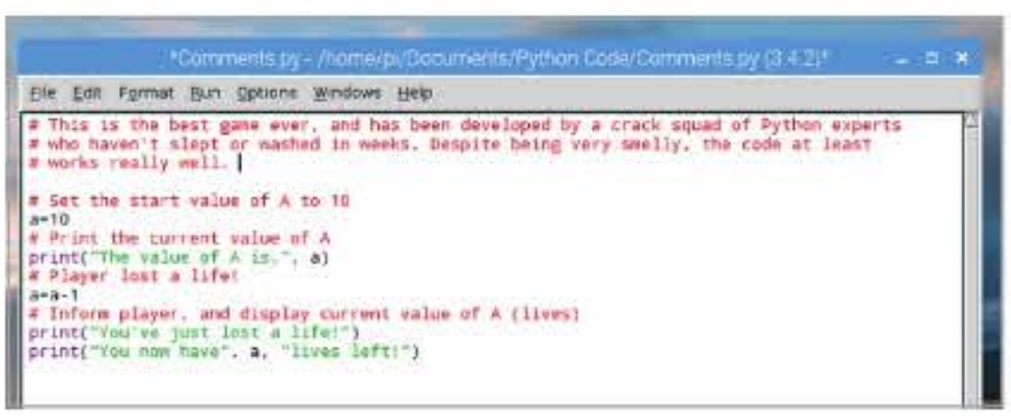
STEP 6 Essentially, the new code together with comments could look like:

```
# Set the start value of A to 10
a=10
# Print the current value of A
print("The value of A is,", a)
# Player lost a life!
a=a-1
# Inform player, and display current value
of A (lives)
print("You've just lost a life!")
print("You now have", a, "lives left!")
```



STEP 7 You can use comments in different ways. For example, Block Comments are a large section of text that details what's going on in the code, such as telling the code reader what variables you're planning on using:

```
# This is the best game ever, and has been
developed by a crack squad of Python experts
# who haven't slept or washed in weeks. Despite
being very smelly, the code at least
# works really well.
```

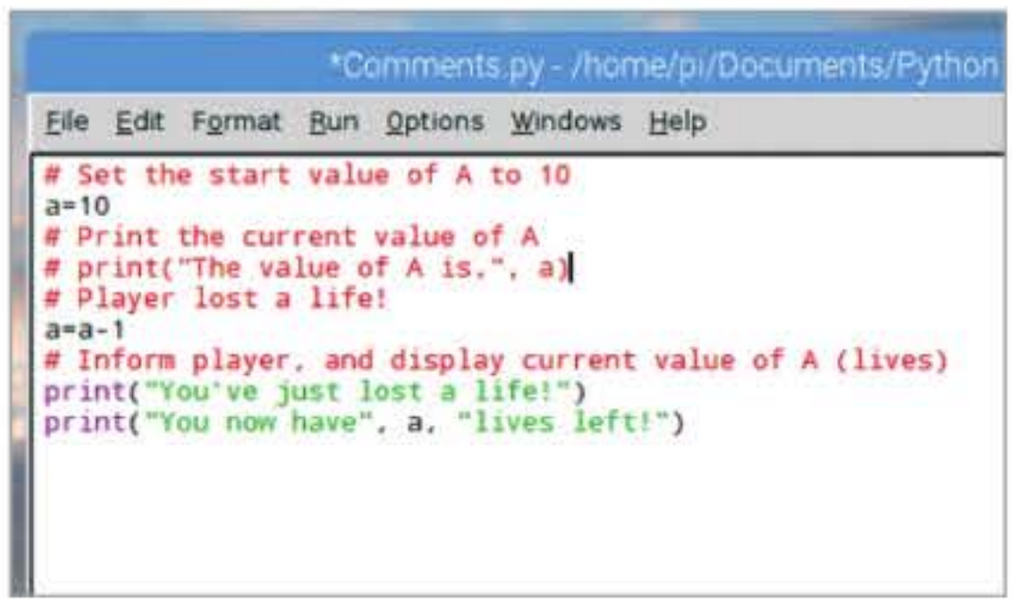


STEP 8 Inline comments are comments that follow a section of code. Take our examples from above, instead of inserting the code on a separate line, we could use:

```
a=10 # Set the start value of A to 10
print("The value of A is,", a) # Print the current
value of A
a=a-1 # Player lost a life!
print("You've just lost a life!")
print("You now have", a, "lives left!") # Inform
player, and display current value of A (lives)
```

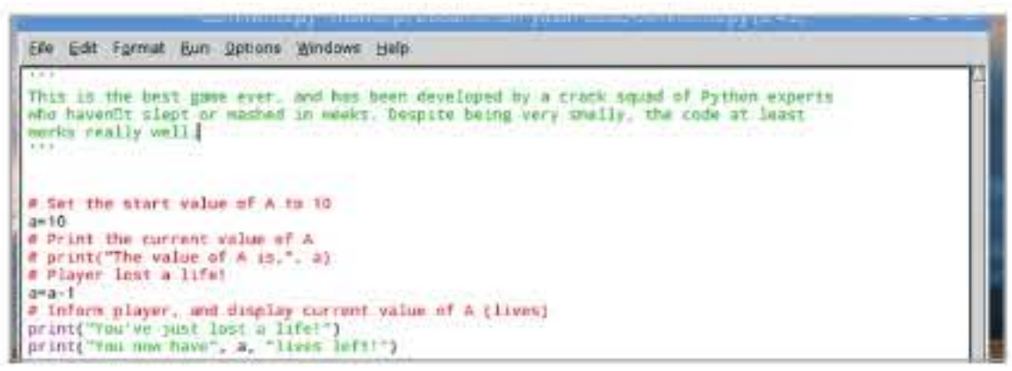
STEP 9 The comment, the hash symbol, can also be used to comment out sections of code you don't want to be executed in your program. For instance, if you wanted to remove the first print statement, you would use:

```
# print("The value of A is,", a)
```



STEP 10 You also use three single quotes to comment out a Block Comment or multi-line section of comments. Place them before and after the areas you want to comment for them to work:

```
'''
This is the best game ever, and has been developed
by a crack squad of Python experts who haven't
slept or washed in weeks. Despite being very
smelly, the code at least works really well.
'''
```



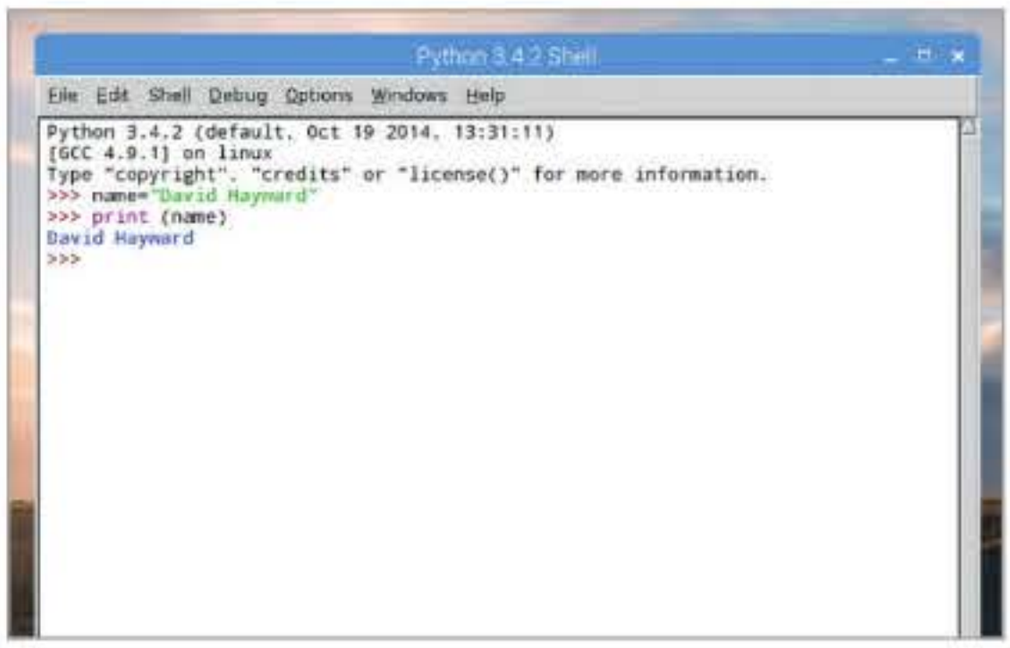
Working with Variables

We've seen some examples of variables in our Python code already but it's always worth going through the way they operate and how Python creates and assigns certain values to a variable.

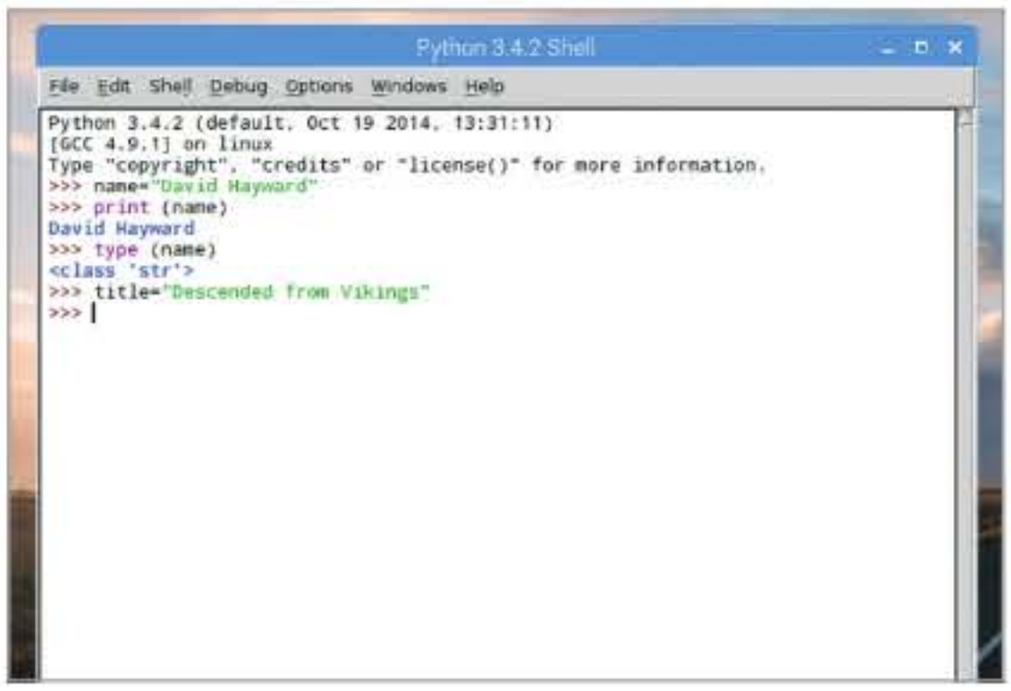
VARIOUS VARIABLES

You'll be working with the Python 3 IDLE Shell in this tutorial. If you haven't already, open Python 3 or close down the previous IDLE Shell to clear up any old code.

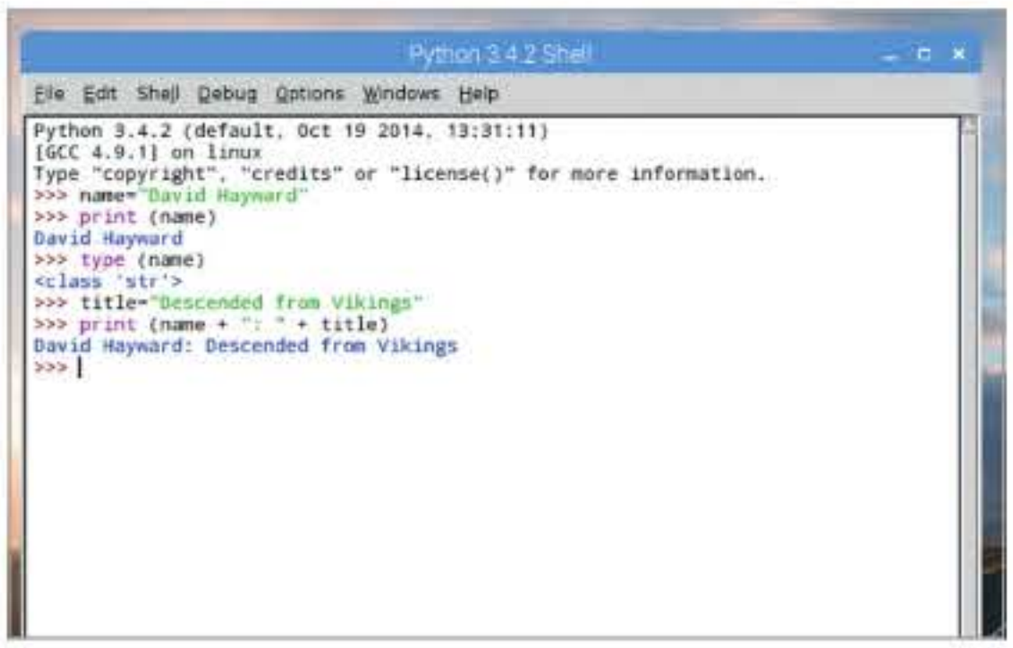
STEP 1 In some programming languages you're required to use a dollar sign to denote a string, which is a variable made up of multiple characters, such as a name of a person. In Python this isn't necessary. For example, in the Shell enter: `name="David Hayward"` (or use your own name, unless you're also called David Hayward).



STEP 2 You can check the type of variable in use by issuing the **type ()** command, placing the name of the variable inside the brackets. In our example, this would be: `type (name)`. Add a new string variable: `title="Descended from Vikings"`.



STEP 3 You've seen previously that variables can be concatenated using the plus symbol between the variable names. In our example we can use: `print (name + ": " + title)`. The middle part between the quotations allows us to add a colon and a space, as variables are connected without spaces, so we need to add them manually.



STEP 4 You can also combine variables within another variable. For example, to combine both name and title variables into a new variable we use:

```
character=name +
": " + title
```

Then output the content of the new variable as:

```
print (character)
```

Numbers are stored as different variables:

```
age=44
Type (age)
```

Which are integers, as we know.

STEP 5 However, you can't combine both strings and integer type variables in the same command, as you would a set of similar variables. You need to either turn one into the other or vice versa. When you do try to combine both, you get an error message:

```
print (name + age)
```

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> name="David Hayward"
>>> print (name)
David Hayward
>>> type (name)
<class 'str'>
>>> title="Descended from Vikings"
>>> print (name + ": " + title)
David Hayward: Descended from Vikings
>>> character=name + ": " + title
>>> print (character)
David Hayward: Descended from Vikings
>>> age=44
>>> type (age)
<class 'int'>
>>> print (name+age)
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    print (name+age)
TypeError: Can't convert 'int' object to str implicitly
>>> |
```

STEP 6 This is a process known as TypeCasting. The Python code is:

```
print (character + " is " + str(age) + "
years old.")
```

or you can use:

```
print (character, "is", age, "years old.")
```

Notice again that in the last example, you don't need the spaces between the words in quotes as the commas treat each argument to print separately.

```
>>> print (name + age)
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    print (name + age)
TypeError: Can't convert 'int' object to str implicitly
>>> print (character + " is " + str(age) + " years old.")
David Hayward: Descended from Vikings is 44 years old.
>>> print (character, "is", age, "years old.")
David Hayward: Descended from Vikings is 44 years old.
>>>
>>> |
```

STEP 7 Another example of TypeCasting is when you ask for input from the user, such as a name. for example, enter:

```
age= input ("How old are you? ")
```

All data stored from the Input command is stored as a string variable.

```
Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> age= input ("How old are you? ")
How old are you? 44
>>> type(age)
<class 'str'>
>>> |
```

STEP 8 This presents a bit of a problem when you want to work with a number that's been inputted by the user, as age + 10 won't work due to being a string variable and an integer. Instead, you need to enter:

```
int(age) + 10
```

This will TypeCast the age string into an integer that can be worked with.

```
Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> age= input ("How old are you? ")
How old are you? 44
>>> type(age)
<class 'str'>
>>> age + 10
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    age + 10
TypeError: Can't convert 'int' object to str implicitly
>>> int(age) + 10
54
>>> |
```

STEP 9 The use of TypeCasting is also important when dealing with floating point arithmetic; remember: numbers that have a decimal point in them. For example, enter:

```
shirt=19.99
```

Now enter `type(shirt)` and you'll see that Python has allocated the number as a 'float', because the value contains a decimal point.

```
Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> shirt=19.99
>>> type(shirt)
<class 'float'>
>>> |
```

STEP 10 When combining integers and floats Python usually converts the integer to a float, but should the reverse ever be applied it's worth remembering that Python doesn't return the exact value. When converting a float to an integer, Python will always round down to the nearest integer, called truncating; in our case instead of 19.99 it becomes 19.

```
Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> shirt=19.99
>>> type(shirt)
<class 'float'>
>>> int(shirt)
19
>>> |
```

User Input

We've seen some basic user interaction with the code from a few of the examples earlier, so now would be a good time to focus solely on how you would get information from the user then store and present it.

USER FRIENDLY

The type of input you want from the user will depend greatly on the type of program you're coding. For example, a game may ask for a character's name, whereas a database can ask for personal details.

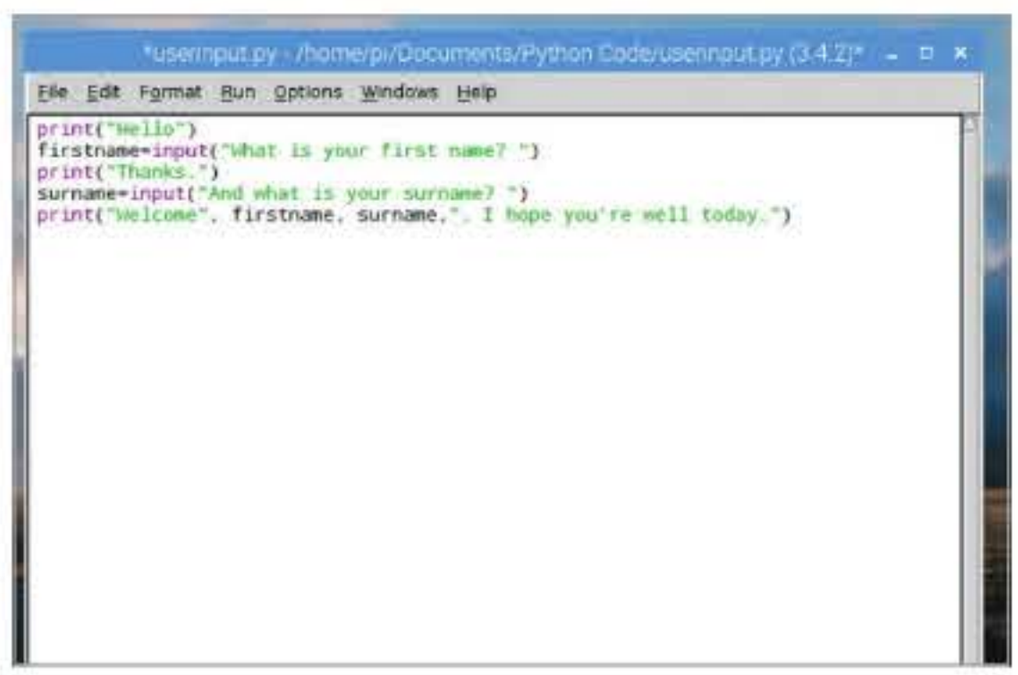
STEP 1 If it's not already, open the Python 3 IDLE Shell, and start a New File in the Editor. Let's begin with something really simple, enter:

```
print("Hello")
firstname=input("What is your first name? ")
print("Thanks.")
surname=input("And what is your surname? ")
```

STEP 2 Save and execute the code, and as you already no doubt suspected, in the IDLE Shell the program will ask for your first name, storing it as the variable firstname, followed by your surname; also stored in its own variable (surname).

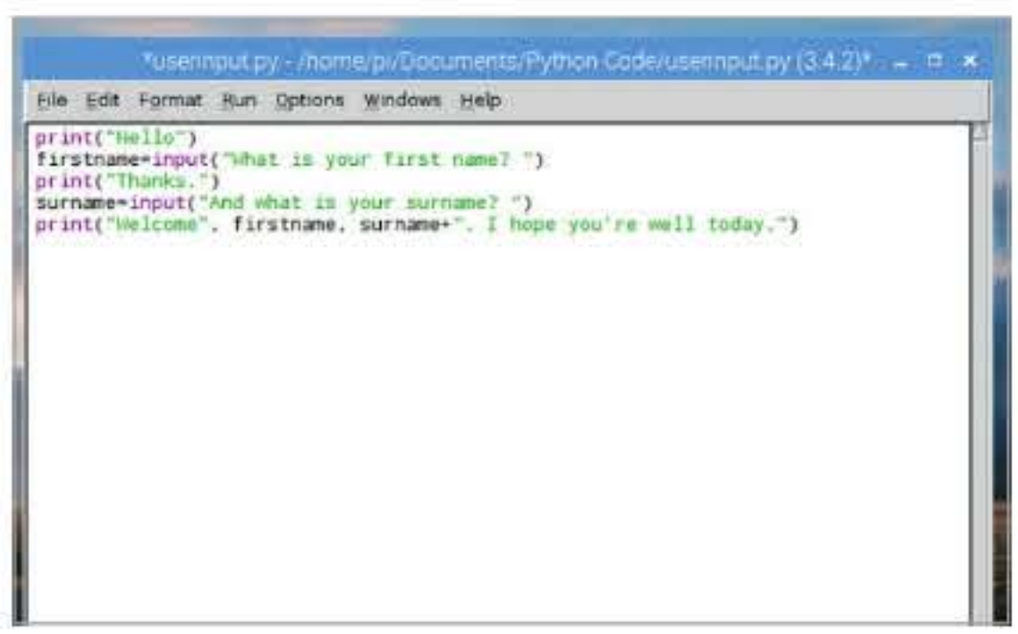
STEP 3 Now that we have the user's name stored in a couple of variables we can call them up whenever we want:

```
print("Welcome", firstname, surname, ". I hope you're well today.")
```



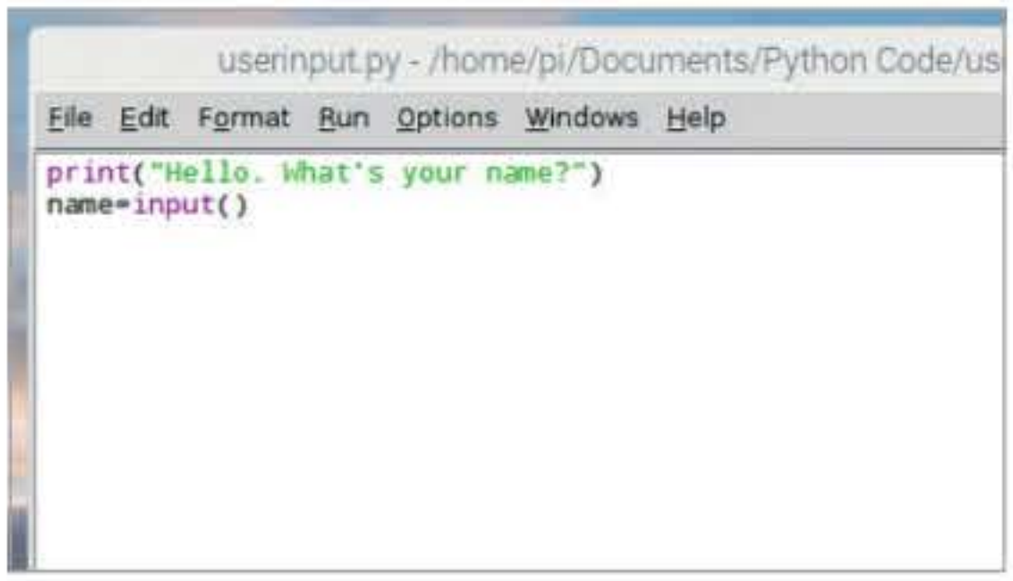
STEP 4 Run the code and you can see a slight issue, the full stop after the surname follows a blank space. To eliminate that we can add a plus sign instead of the comma in the code:

```
print("Welcome", firstname, surname+". I hope you're well today.")
```



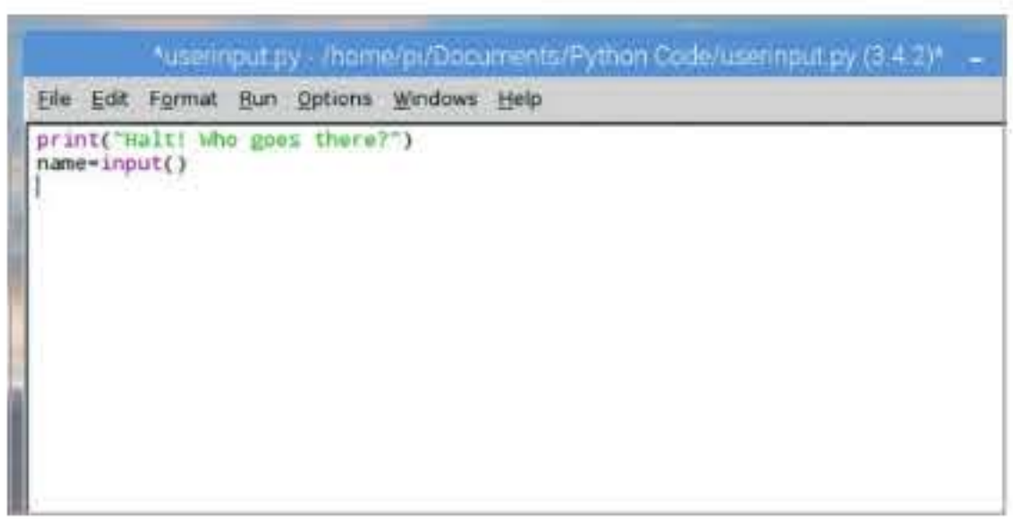
STEP 5 You don't always have to include quoted text within the input command. For example, you can ask the user their name, and have the input in the line below:

```
print("Hello. What's your name?")
name=input()
```



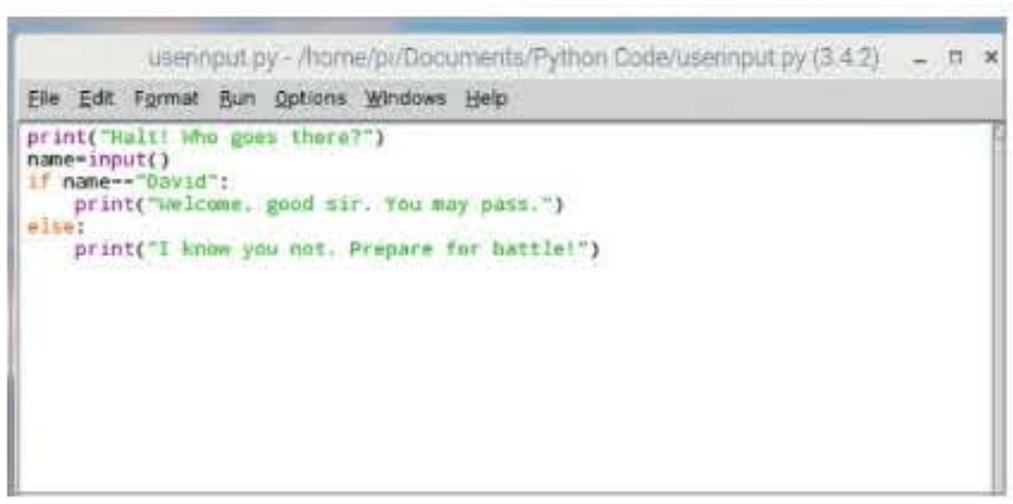
STEP 6 The code from the previous step is often regarded as being a little neater than having a lengthy amount of text in the input command, but it's not a rule that's set in stone, so do as you like in these situations. Expanding on the code, try this:

```
print("Halt! Who goes there?")
name=input()
```

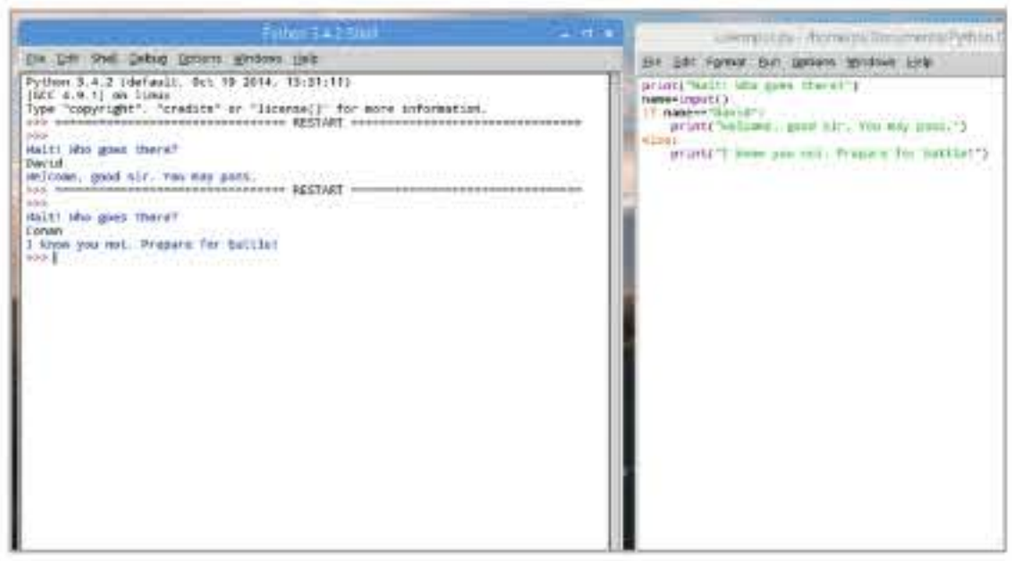


STEP 7 It's a good start to a text adventure game, perhaps? Now you can expand on it and use the raw input from the user to flesh out the game a little:

```
if name=="David":
  print("Welcome, good sir. You may pass.")
else:
  print("I know you not. Prepare for battle!")
```

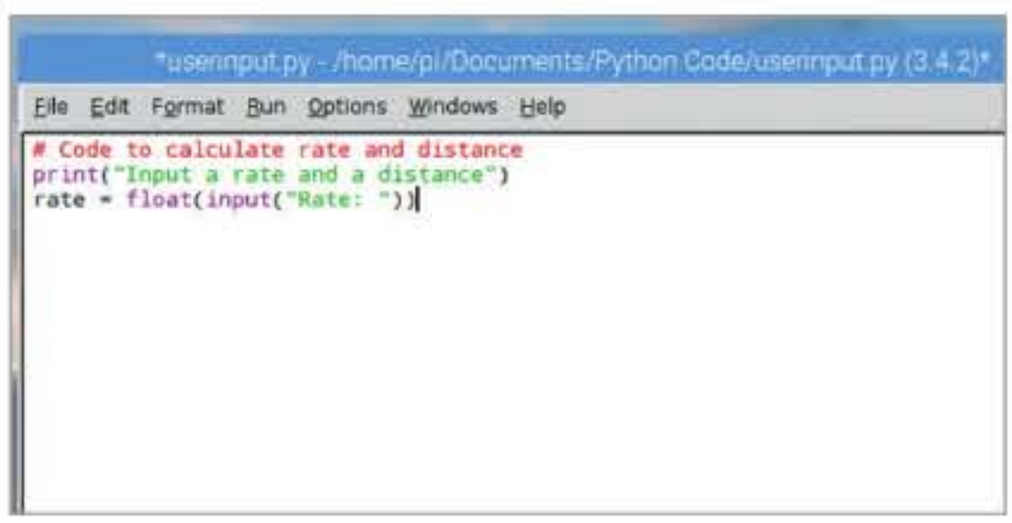


STEP 8 What you've created here is a condition, which we will cover soon. In short, we're using the input from the user and measuring it against a condition. So, if the user enters David as their name, the guard will allow them to pass unhindered. Else, if they enter a name other than David, the guard challenges them to a fight.



STEP 9 Just as you learned previously, any input from a user is automatically a string, so you need to apply a TypeCast in order to turn it into something else. This creates some interesting additions to the input command. For example:

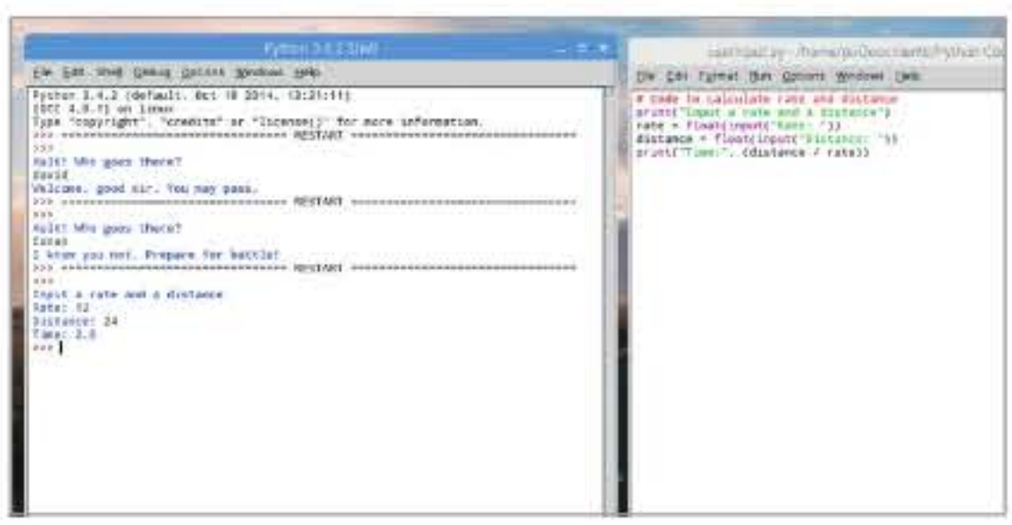
```
# Code to calculate rate and distance
print("Input a rate and a distance")
rate = float(input("Rate: "))
```



STEP 10 To finalise the rate and distance code, we can add:

```
distance = float(input("Distance: "))
print("Time:", (distance / rate))
```

Save and execute the code and enter some numbers. Using the float(input element, we've told Python that anything entered is a floating point number rather than a string.



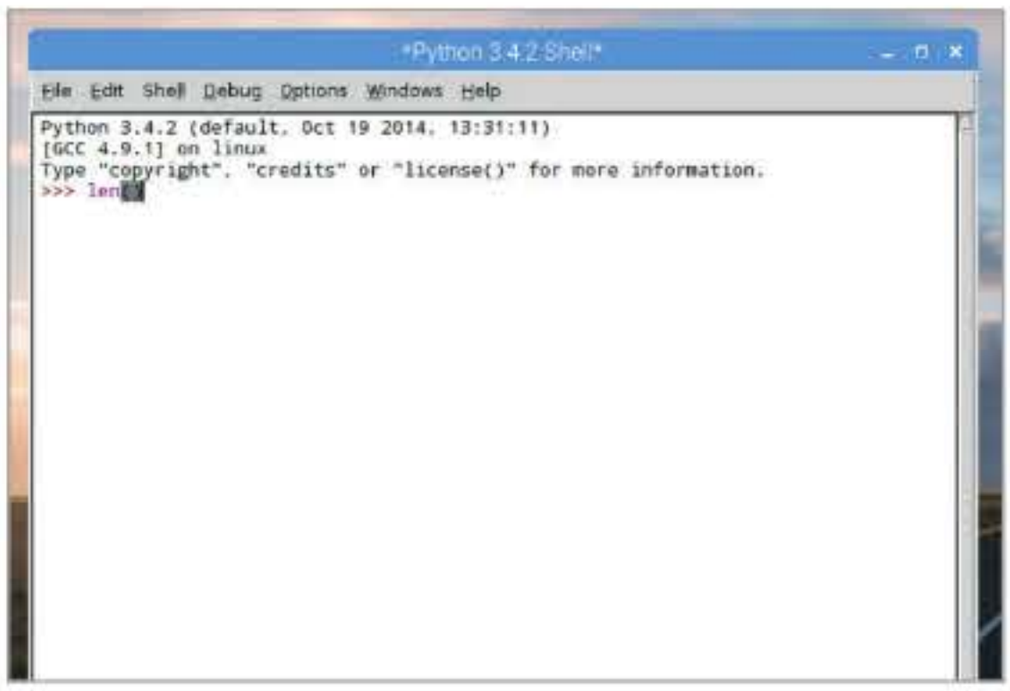
Creating Functions

Now that you've mastered the use of variables and user input, the next step is to tackle functions. You've already used a few functions, such as the print command but Python enables you to define your own functions.

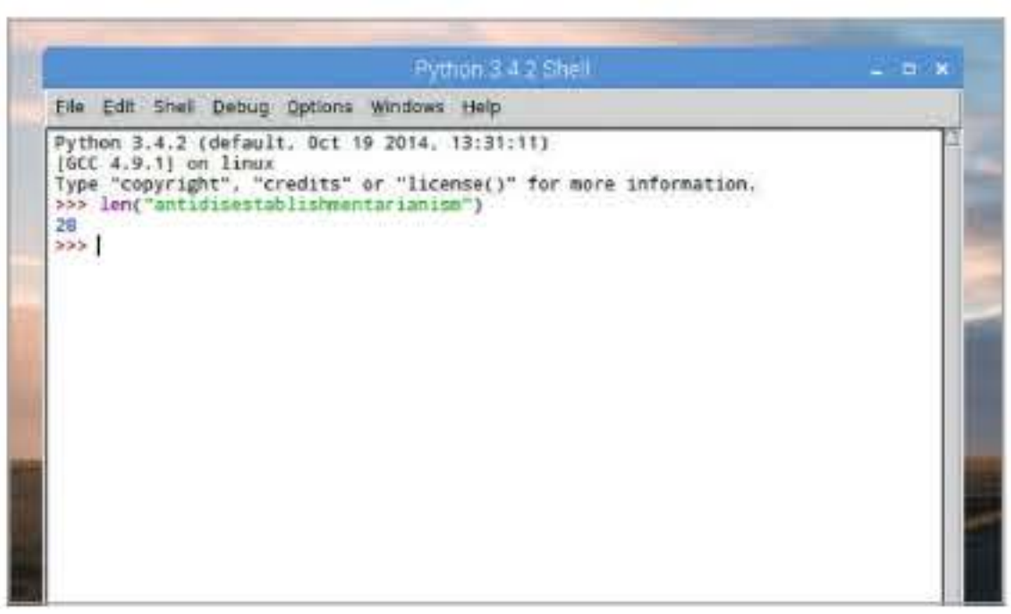
FUNKY FUNCTIONS

A function is a command that you enter into Python to do something. It's a little piece of self-contained code that takes data, works on it and then returns the result.

STEP 1 It's not just data that a function works on. They can do all manner of useful things in Python, such as sort data, change items from one format to another and check the length or type of items. Basically, a function is a short word that's followed by brackets. For example, **len()**, **list()** or **type()**.



STEP 2 A function takes data, usually a variable, works on it depending on what the function is programmed to do and returns the end value. The data being worked on goes inside the brackets, so if you wanted to know how many letters are in the word antidisestablishmentarianism, then you'd enter: `len("antidisestablishmentarianism")` and the number 28 would return.

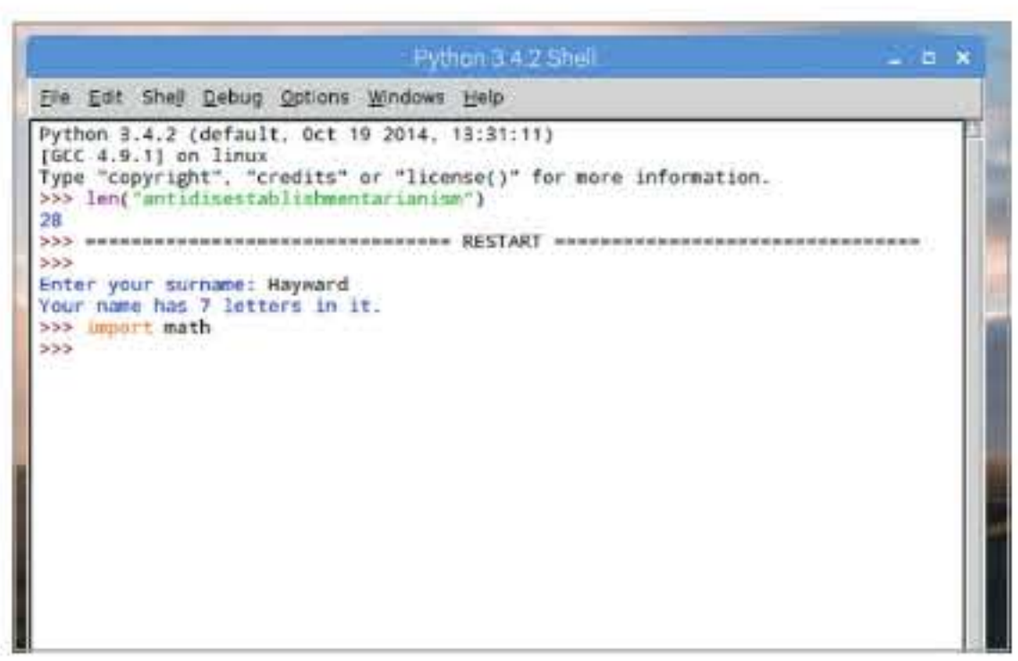


STEP 3 You can pass variables through functions in much the same manner. Let's assume you want the number of letters in a person's surname, you could use the following code (enter the text editor for this example):

```
name=input ("Enter your surname: ")
count=len(name)
print ("Your surname has", count,
"letters in it.")
```

Press F5 and save the code to execute it.

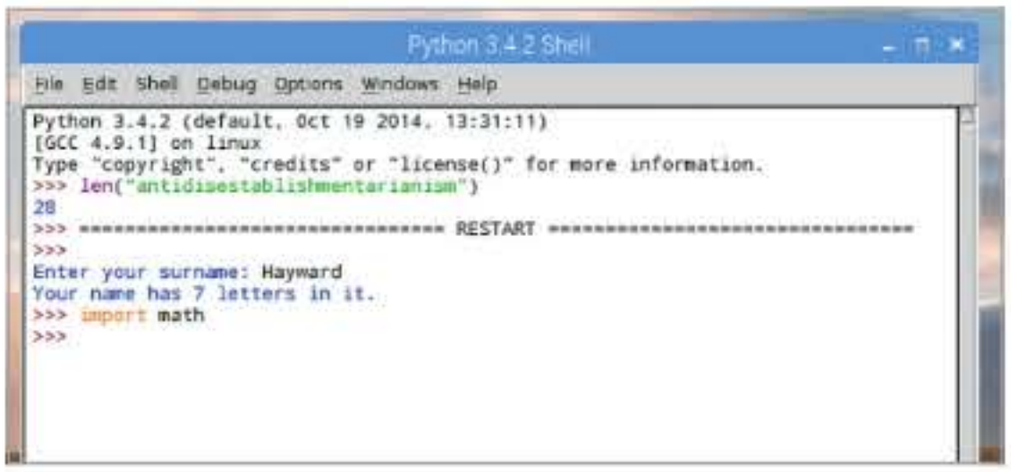
STEP 4 Python has tens of functions built into it, far too many to get into in the limited space available here. However, to view the list of built-in functions available to Python 3, navigate to www.docs.python.org/3/library/functions.html. These are the predefined functions, but since users have created many more, they're not the only ones available.



STEP 5 Additional functions can be added to Python through modules. Python has a vast range of modules available that can cover numerous programming duties. They add functions and can be imported as and when required. For example, to use advanced Mathematics functions enter:

```
import math
```

Once entered, you have access to all the Math module functions.



STEP 6 To use a function from a module enter the name of the module followed by a full stop, then the name of the function. For instance, using the math module, since you've just imported it into Python, you can utilise the square root function. To do so, enter:

```
math.sqrt(16)
```

You can see that the code is presented as module.function(data).

FORGING FUNCTIONS

There are many different functions you can import created by other Python programmers and you will undoubtedly come across some excellent examples in the future; you can also create your own with the def command.

STEP 1 Choose File > New File to enter the editor, let's create a function called Hello, that greets a user. Enter:

```
def Hello():
   print ("Hello")

Hello()
```

Press F5 to save and run the script. You can see Hello in the Shell, type in Hello() and it returns the new function.

STEP 2 Let's now expand the function to accept a variable, the user's name for example. Edit your script to read:

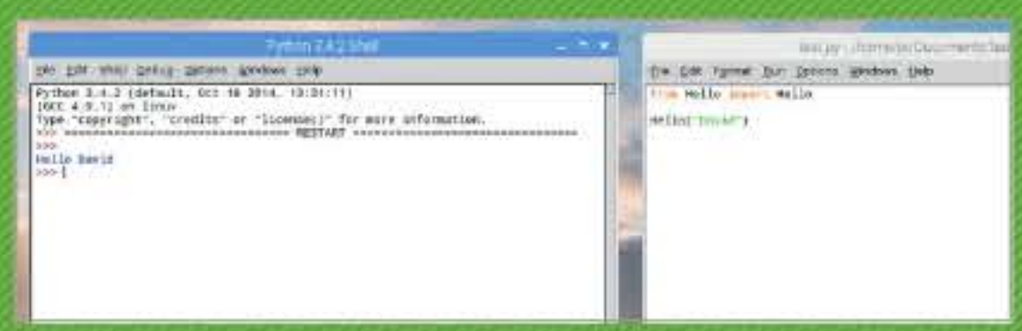
```
def Hello(name):
   print ("Hello", name)

Hello("David")
```

This will now accept the variable name, otherwise it prints Hello David. In the Shell, enter: name=("Bob"), then: Hello(name). Your function can now pass variables through it.

STEP 3 To modify it further, delete the Hello("David") line, the last line in the script and press Ctrl+S to save the new script. Close the Editor and create a new file (File > New File). Enter the following:

```
from Hello import Hello

Hello("David")
```

Press F5 to save and execute the code.



STEP 4 What you've just done is import the Hello function from the saved Hello.py program and then used it to say hello to David. This is how modules and functions work: you import the module then use the function. Try this one, and modify it for extra credit:

```
def add(a, b):
  result = a + b
  return result
```

Conditions and Loops

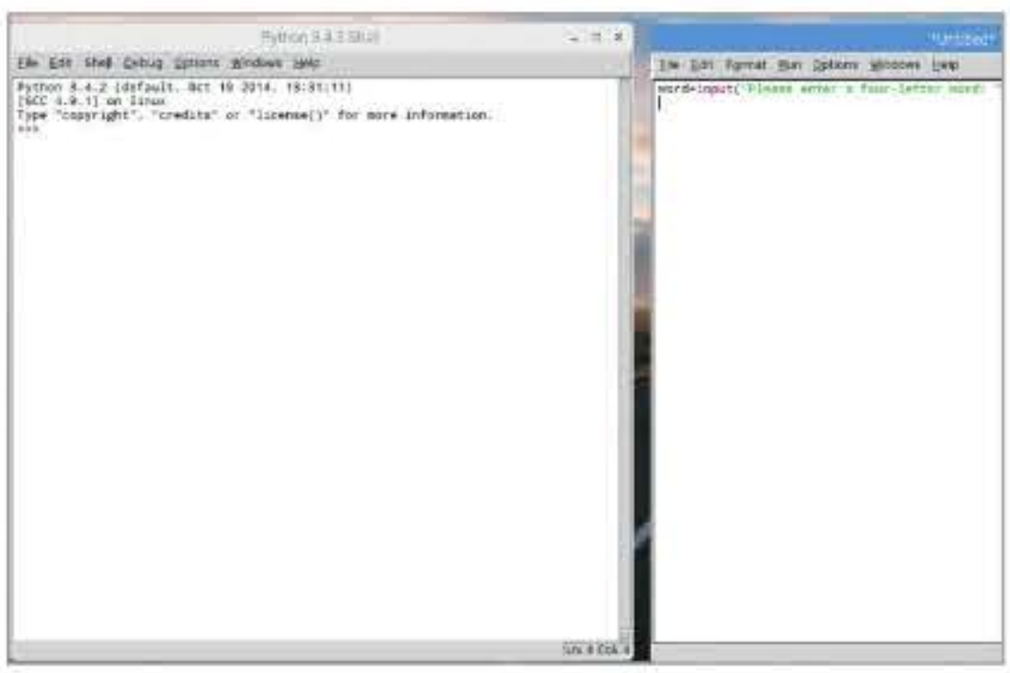
Conditions and loops are what makes a program interesting; they can be simple or rather complex. How you use them depends greatly on what the program is trying to achieve; they could be the number of lives left in a game or just displaying a countdown.

TRUE CONDITIONS

Keeping conditions simple to begin with makes learning to program a more enjoyable experience. Let's start then by checking if something is TRUE, then doing something else if it isn't.

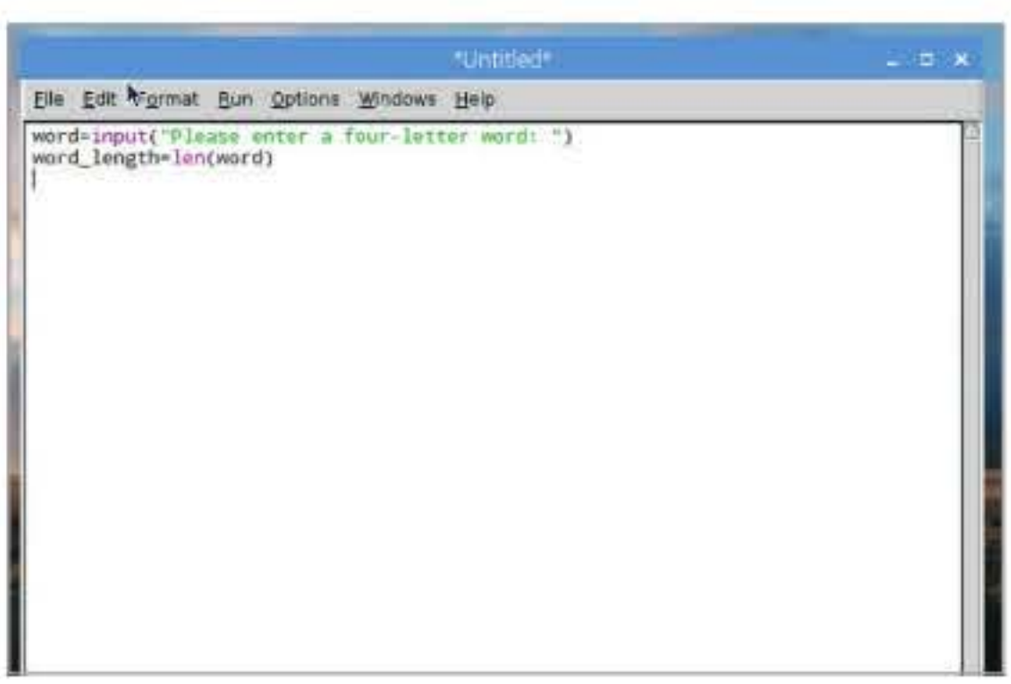
STEP 1 Let's create a new Python program that will ask the user to input a word, then check it to see if it's a four-letter word or not. Start with File > New File, and begin with the input variable:

```
word=input("Please enter a four-letter word: ")
```



STEP 2 Now we can create a new variable, then use the len function and pass the word variable through it to get the total number of letters the user has just entered:

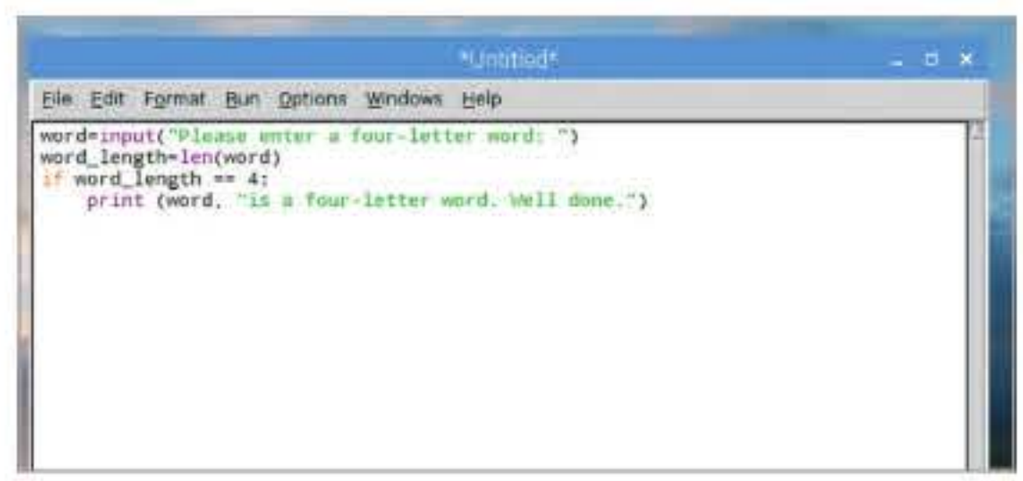
```
word=input("Please enter a four-letter word: ")
word_length=len(word)
```



STEP 3 Now you can use an if statement to check if the word_length variable is equal to four and print a friendly conformation if it applies to the rule:

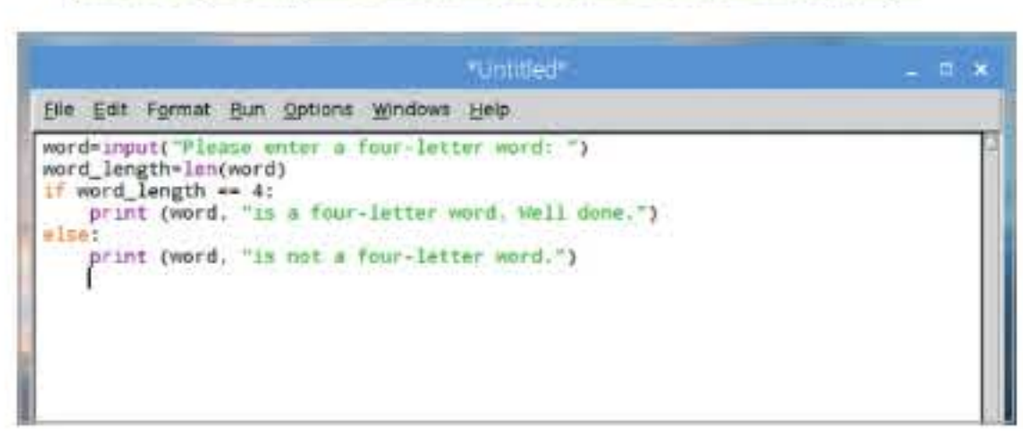
```
word=input("Please enter a four-letter word: ")
word_length=len(word)
if word_length == 4:
   print (word, "is a four-letter word. Well done.")
```

The double equal sign (==) means check if something is equal to something else.

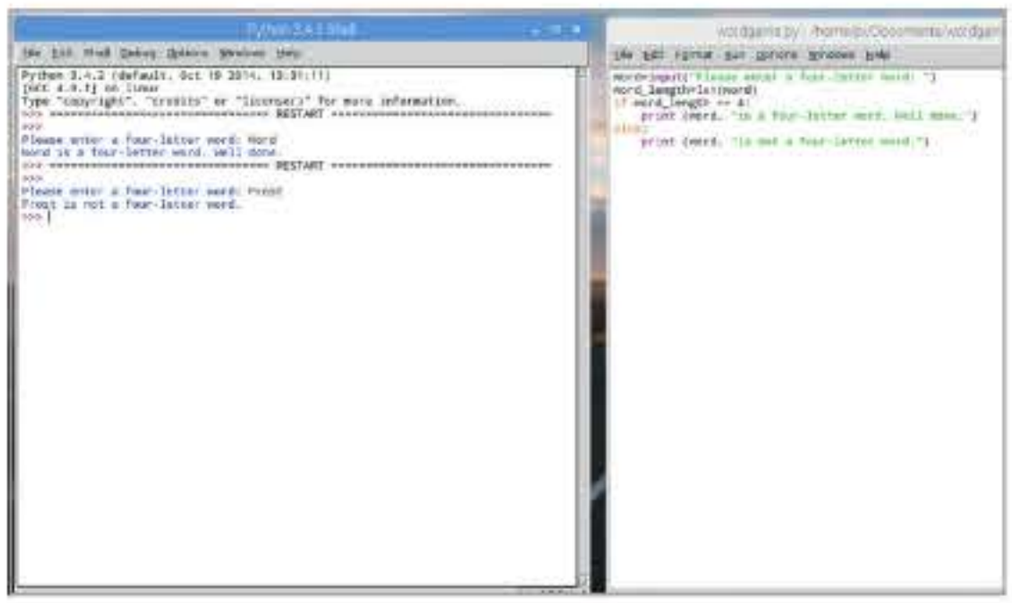


STEP 4 The colon at the end of IF tells Python that if this statement is true do everything after the colon that's indented. Next, move the cursor back to the beginning of the Editor:

```
word=input("Please enter a four-letter word: ")
word_length=len(word)
if word_length == 4:
  print (word, "is a four-letter word.
Well done.")
else:
   print (word, "is not a four-letter word.")
```

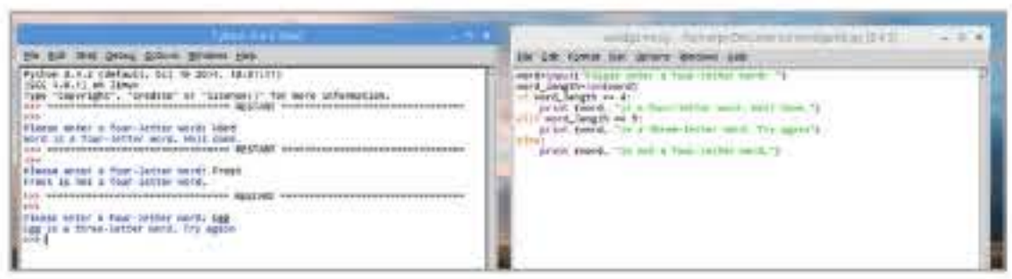


STEP 5 Press F5 and save the code to execute it. Enter a four-letter word in the Shell to begin with, you should have the returned message that it's the word is four letters. Now press F5 again and rerun the program but this time enter a five-letter word. The Shell will display that it's not a four-letter word.



STEP 6 Now expand the code to include another conditions. Eventually, it could become quite complex. We've added a condition for three-letter words:

```
word=input("Please enter a four-letter word: ")
word_length=len(word)
if word_length == 4:
   print (word, "is a four-letter word.
Well done.")
elif word_length == 3:
   print (word, "is a three-letter word. Try again.")
else:
   print (word, "is not a four-letter word.")
```

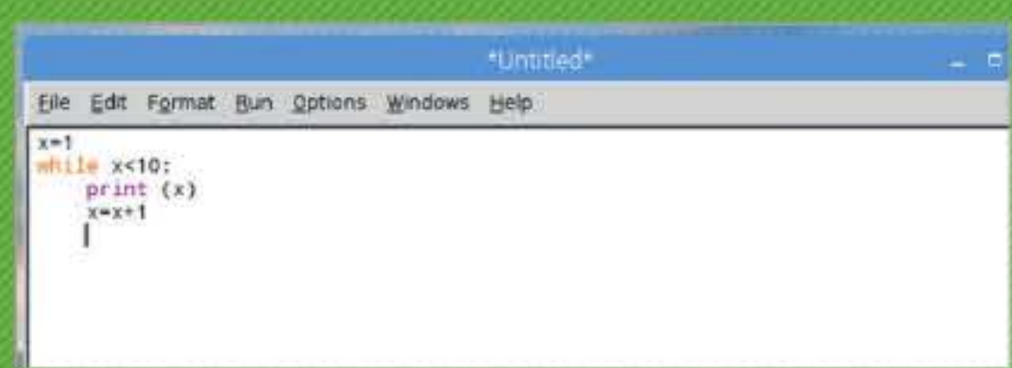


LOOPS

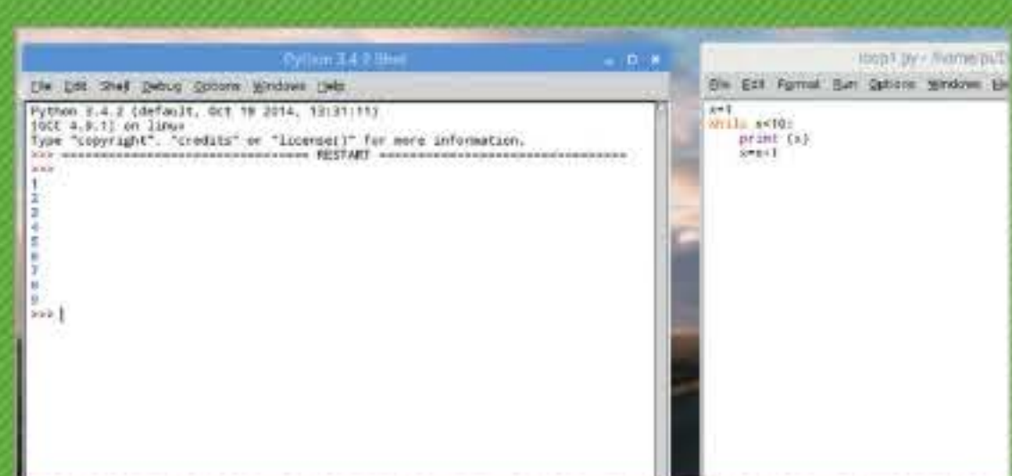
A loop looks quite similar to a condition but they are somewhat different in their operation. A loop will run through the same block of code a number of times, usually with the support of a condition.

STEP 1 Step 1 – Let's start with a simple While statement. Like IF, this will check to see if something is TRUE, then run the indented code:

```
x = 1
while x < 10:
   print (x)
   x = x + 1
```

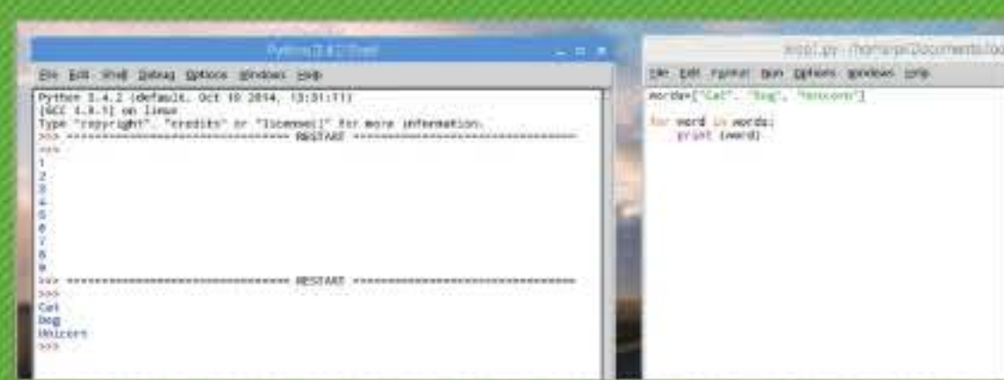


STEP 2 The difference between if and while is when while gets to the end of the indented code, it goes back and checks the statement is still true. In our example x is less than 10. With each loop it prints the current value of x, then adds one to that value. When x does eventually equal 10 it stops.



STEP 3 The For loop is another example. For is used to loop over a range of data, usually a list stored as variables inside square brackets. For example:

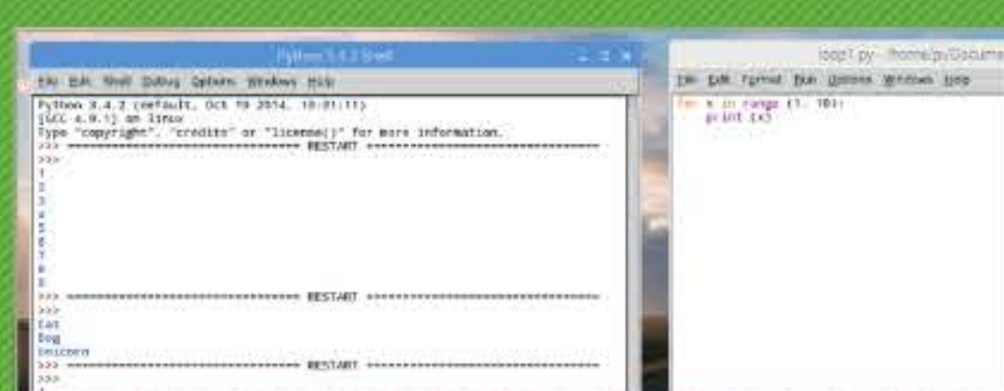
```
words=["Cat", "Dog", "Unicorn"]

for word in words:
   print (word)
```



STEP 4 The For loop can also be used in the countdown example by using the range function:

```
for x in range (1, 10):
 print (x)
```

The x=x+1 part isn't needed here because the range function creates a list between the first and last numbers used.



Python Modules

We've mentioned modules previously, (the Math module) but as modules are such a large part of getting the most from Python, it's worth dedicating a little more time to them. In this instance we're using the Windows version of Python 3.

MASTERING MODULES

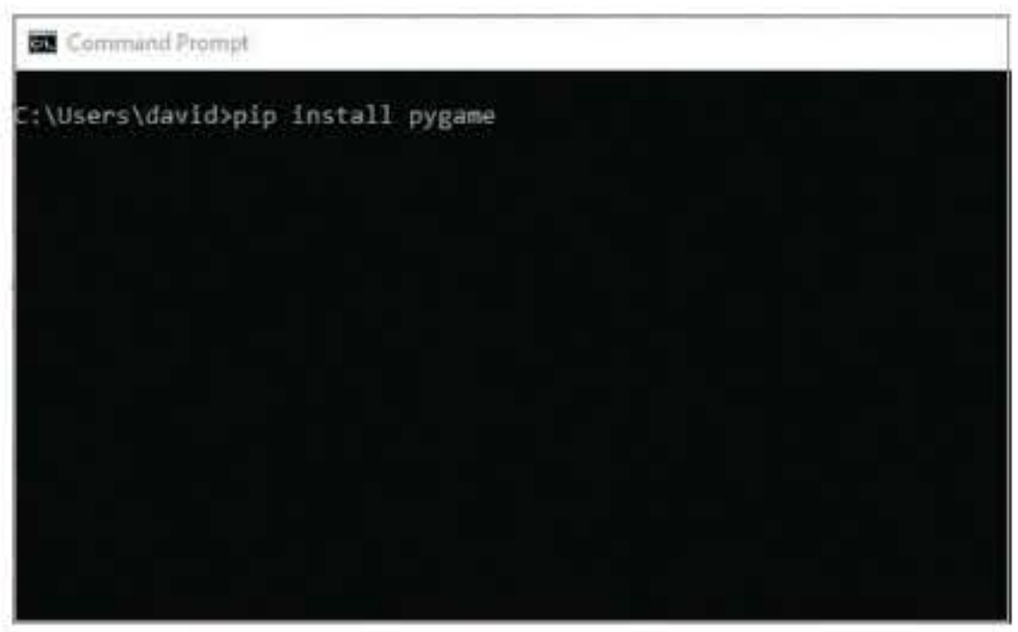
Think of modules as an extension that's imported into your Python code to enhance and extend its capabilities. There are countless modules available and as we've seen, you can even make your own.

STEP 1 Although good, the built-in functions within Python are limited. The use of modules, however, allows us to make more sophisticated programs. As you are aware, modules are Python scripts that are imported, such as import math.

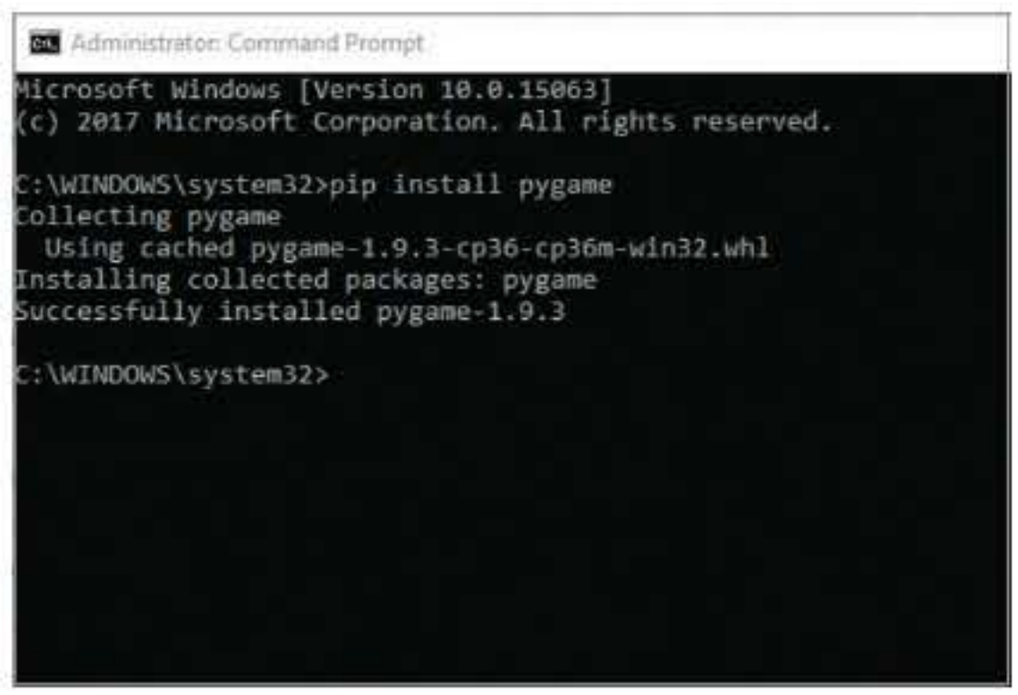
STEP 2 Some modules, especially on the Raspberry Pi, are included by default, the math module being a prime example. Sadly, other modules aren't always available. A good example on non-Pi platforms is the pygame module, which contains many functions to help create games. Try: **import pygame**.

STEP 3 The result is an error in the IDLE Shell, as the pygame module isn't recognised or installed in Python. To install a module we can use PIP (Pip Installs Packages). Close down the IDLE Shell and drop into a command prompt or Terminal session. At an elevated admin command prompt, enter:

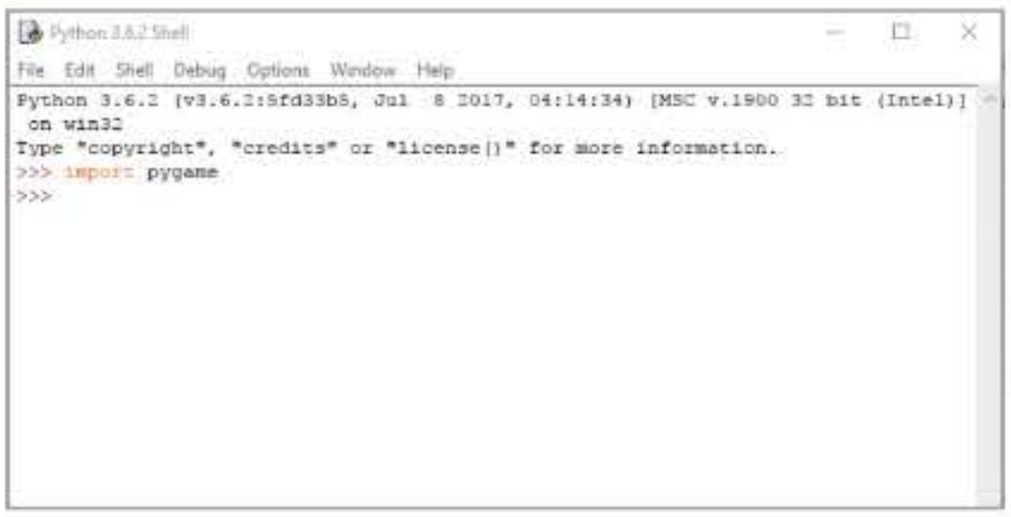
```
pip install pygame
```



STEP 4 The PIP installation requires an elevated status due it installing components at different locations. Windows users can search for CMD via the Start button and right-click the result then click Run as Administrator. Linux and Mac users can use the Sudo command, with sudo pip install package.



STEP 5 Close the command prompt or Terminal and relaunch the IDLE Shell. When you now enter: `import pygame`, the module will be imported into the code without any problems. You'll find that most code downloaded or copied from the internet will contain a module, mainstream of unique, these are usually the source of errors in execution due to them being missing.



STEP 6 The modules contain the extra code needed to achieve a certain result within your own code, as we've previously experimented with. For example:

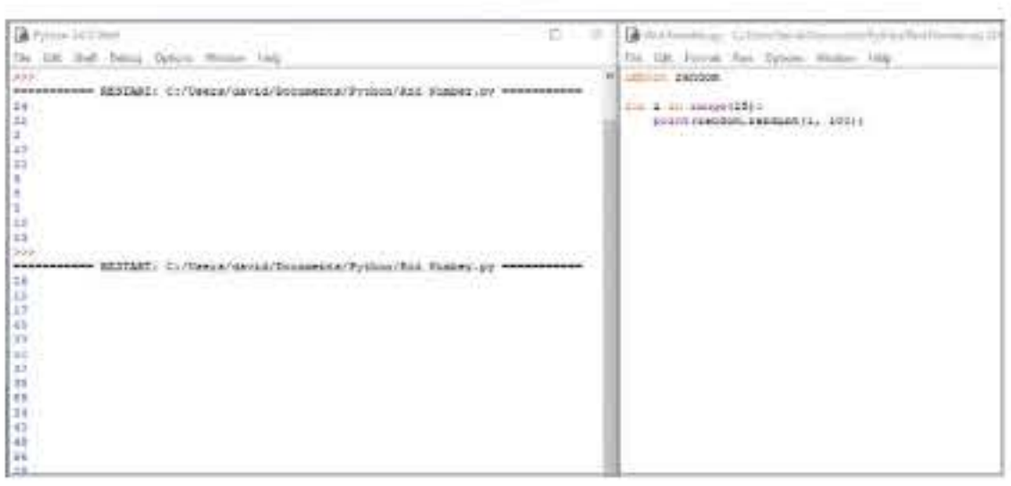
```
import random
```

Brings in the code from the random number generator module. You can then use this module to create something like:

```
for i in range(10):
  print(random.randint(1, 25))
```

STEP 7 This code, when saved and executed, will display ten random numbers from 1 to 25. You can play around with the code to display more or less, and from a great or lesser range. For example:

```
import random

for i in range(25):
  print(random.randint(1, 100))
```

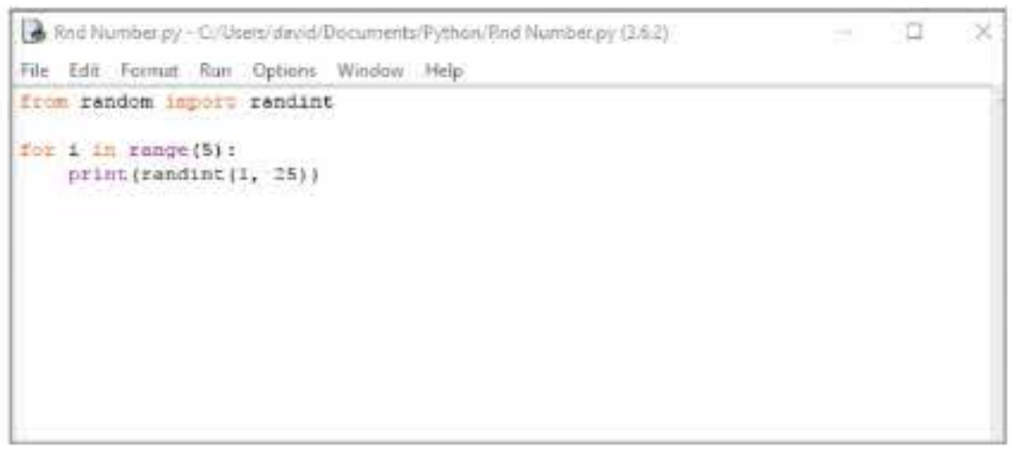


STEP 8 Multiple modules can be imported within your code. To extend our example, use:

```
import random
import math

for I in range(5):
  print(random.randint(1, 25))

print(math.pi)
```

STEP 9 The result is a string of random numbers followed by the value of Pi as pulled from the math module using the print(math.pi) function. You can also pull in certain functions from a module by using the from and import commands, such as:

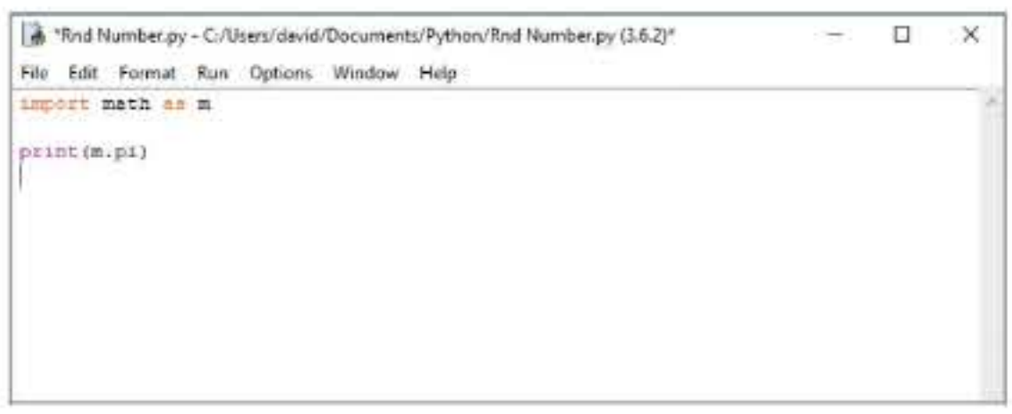
```
from random import randint

for i in range(5):
  print(randint(1, 25))
```



STEP 10 This helps create a more streamlined approach to programming. You can also use import module*, which will import everything defined within the named module. However, it's often regarded as a waste of resources but it works nonetheless. Finally, modules can be imported as aliases:

```
import math as m

print(m.pi)
```

Of course, adding comments helps to tell others what's going on.



Python Errors

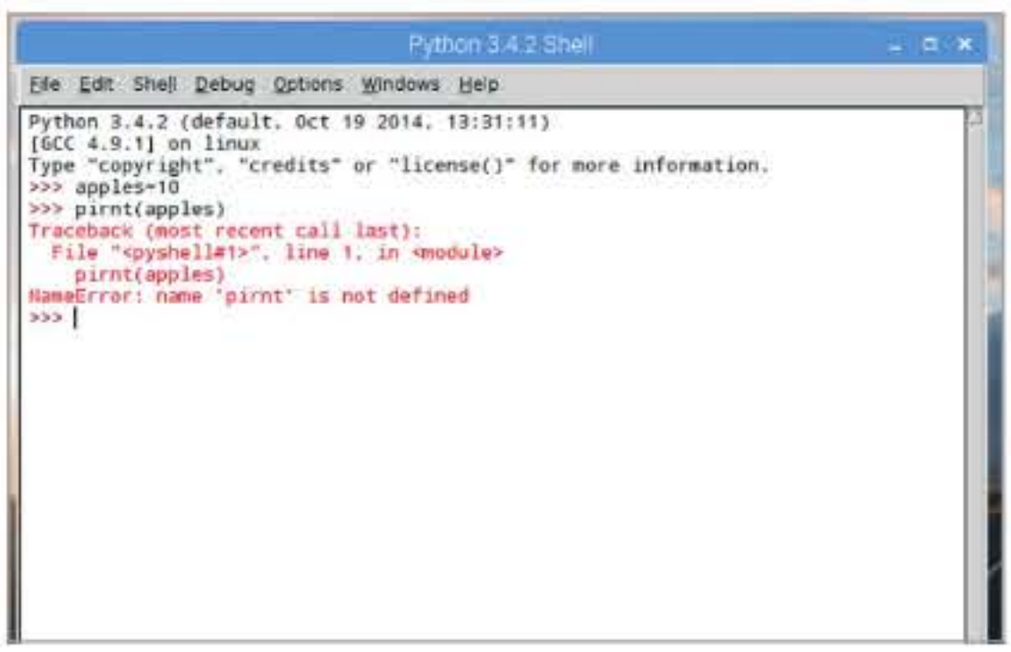
It goes without saying that you'll eventually come across an error in your code, where Python declares it's not able to continue due to something being missed out, wrong or simply unknown. Being able to identify these errors makes for a good programmer.

DEBUGGING

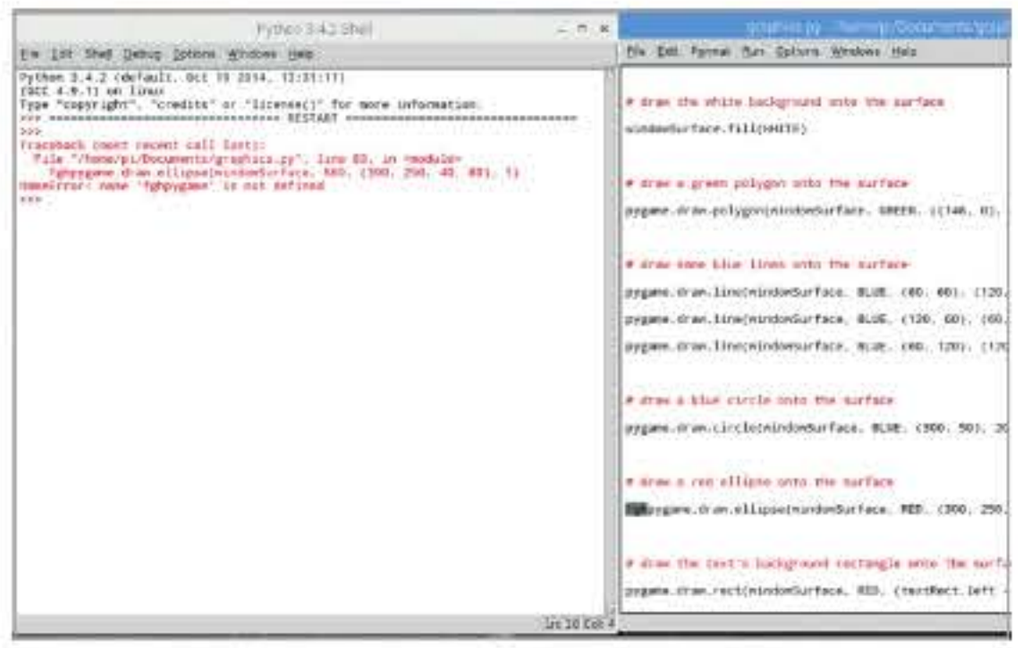
Errors in code are called bugs and are perfectly normal. They can often be easily rectified with a little patience. The important thing is to keep looking, experimenting and testing. Eventually your code will be bug free.

STEP 1 Code isn't as fluid as the written word, no matter how good the programming language is. Python is certainly easier than most languages but even it is prone to some annoying bugs. The most common are typos by the user and whilst easy to find in simple dozen-line code, imagine having to debug multi-thousand line code.

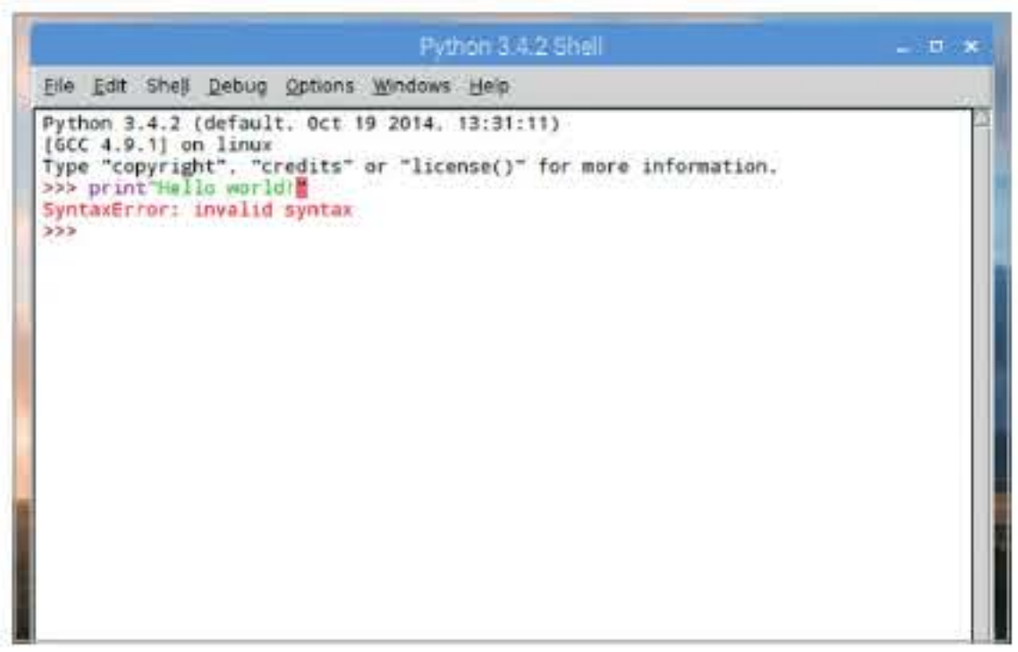
STEP 2 The most common of errors is the typo, as we've mentioned. The typos are often at the command level: mistyping the print command for example. However, they also occur when you have numerous variables, all of which have lengthy names. The best advice is to simply go through the code and check your spelling.



STEP 3 Thankfully Python is helpful when it comes to displaying error messages. When you receive an error, in red text from the IDLE Shell, it will define the error itself along with the line number where the error has occurred. Whilst in the IDLE Editor this is a little daunting for lots of code; text editors help by including line numbering.



STEP 4 Syntax errors are probably the second most common errors you'll come across as a programmer. Even if the spelling is correct, the actual command itself is wrong. In Python 3 this often occurs when Python 2 syntaxes are applied. The most annoying of these is the print function. In Python 3 we use **print("words")**, whereas Python2 uses **print "words"**.



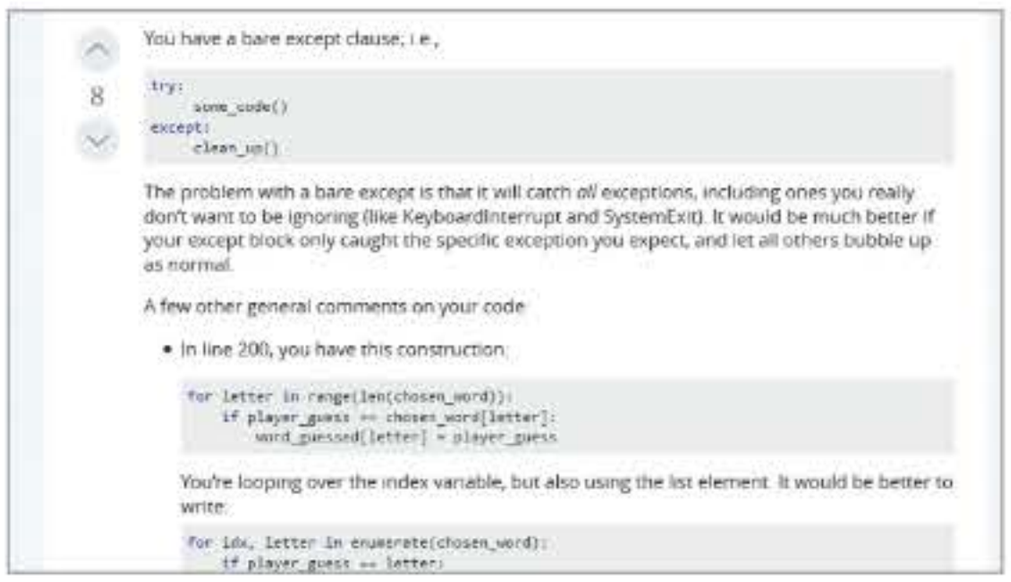
STEP 5 Pesky brackets are also a nuisance in programming errors, especially when you have something like:

```
print(balanced_check(input()))
```

Remember that for every '(' there must be an equal number of ')'.

```
import sys

def balanced_check(data):
    stack = []
    characters = list(data)

    for character in characters:
        reference = {
            '(': ')',
            '{': '}',
            '[': ']'
        }
        if character in reference.keys():
            stack.append(character)

        elif character in reference.values() and len(stack) > 0:
            char = stack.pop()
            if reference.get(char) != character:
                return "NO"
        else:
```

STEP 6 There are thousands of online Python resources, code snippets and lengthy discussions across forums on how best to achieve something. Whilst 99 per cent of it is good code, don't always be lured into copying and pasting random code into your editor. More often than not, it won't work and the worst part is that you haven't learnt anything.



You have a bare except clause; i.e.,

```
try:
    some_code()
except:
    clean_up()
```

The problem with a bare except is that it will catch *all* exceptions, including ones you really don't want to be ignoring (like KeyboardInterrupt and SystemExit). It would be much better if your except block only caught the specific exception you expect, and let all others bubble up as normal.

A few other general comments on your code:

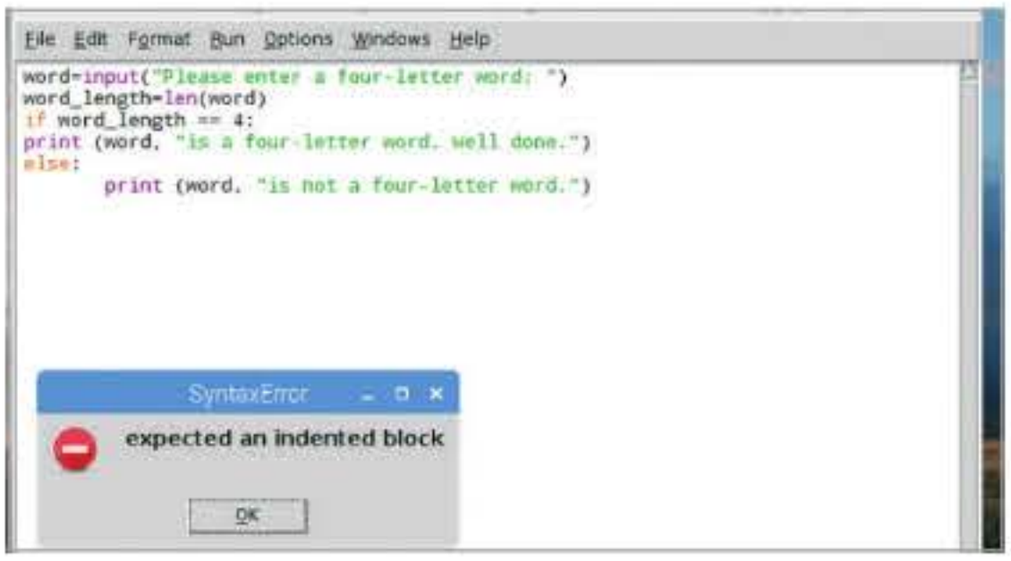
- In line 200, you have this construction:

```
for letter in range(len(chosen_word)):
    if player_guess == chosen_word[letter]:
        word_guessed[letter] = player_guess
```

You're looping over the index variable, but also using the list element. It would be better to write:

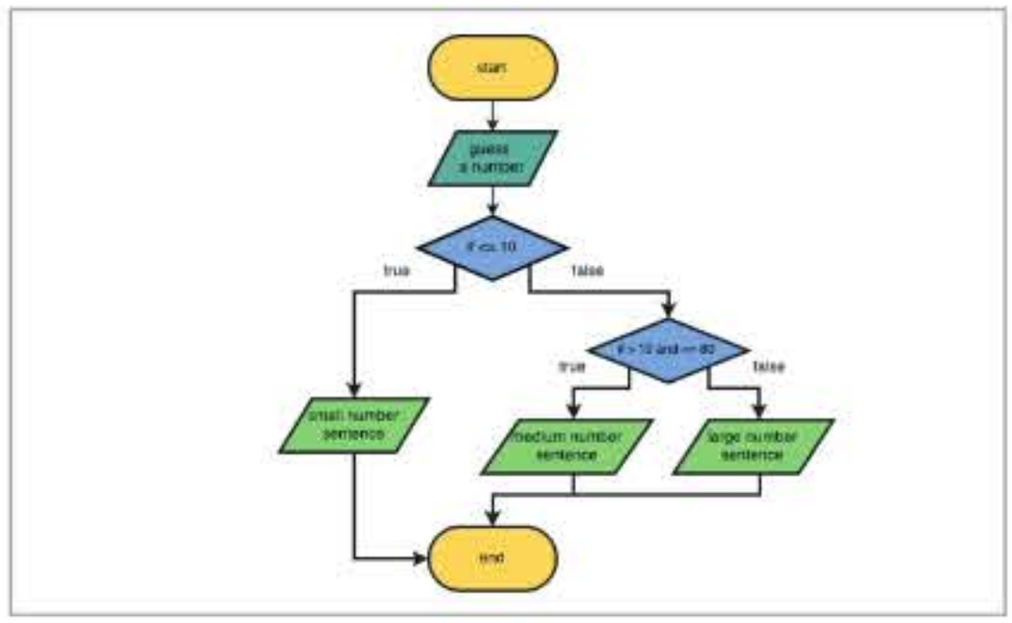
```
for idx, letter in enumerate(chosen_word):
    if player_guess == letter:
```

STEP 7 Indents are a nasty part of Python programming that a lot of beginners fall foul of. Recall the If loop from the Conditions and Loops section, where the colon means everything indented following the statement is to be executed as long as it's true? Missing the indent, or having too much of indent, will come back with an error.



STEP 8 An excellent way to check your code step-by-step is to use Python Tutor's Visualise web page, found at www.pythontutor.com/visualize.html#mode=edit. Simply paste your code into the editor and click the Visualise Execution button to run the code line-by-line. This helps to clear bugs and any misunderstandings.

STEP 9 Planning makes for good code. Whilst a little old school, it's a good habit to plan what your code will do before sitting down to type it out. List the variables that will be used and the modules too; then write out a script for any user interaction or outputs.



STEP 10 Purely out of interest, the word debugging in computing terms comes from Admiral Grace Hopper, who back in the '40s was working on a monolithic Harvard Mark II electromechanical computer. According to legend Hopper found a moth stuck in a relay, thus stopping the system from working. Removal of the moth was hence called debugging.

Combining What You Know So Far

We've reached the end of this section so let's take a moment to combine everything we've looked at so far, and apply it to writing a piece of code. This code can then be used and inserted into your own programs in future, either part of it or as a whole.

PLAYING WITH PI

For this example we're going to create a program that will calculate the value of Pi to a set number of decimal places, as described by the user. It combines much of what we've learnt, and a little more.

STEP 1 Start by opening Python and creating a New File in the Editor. First we need to get hold of an equation that can accurately calculate Pi without rendering the computer's CPU useless for several minutes. The recommended calculation used in such circumstances is the Chudnovsky Algorithm, you can find more information about it at en.wikipedia.org/wiki/Chudnovsky_algorithm.

STEP 2 You can utilise the Chudnovsky Algorithm to create your own Python script based on the calculation. Begin by importing some important modules and functions within the modules:

```
from decimal import Decimal, getcontext
import math
```

This uses the decimal and getcontext functions from the decimal module, both of which deal with large decimal place numbers and naturally the math module.

STEP 3 Now you can insert the Pi calculation algorithm part of the code. This is a version of the Chudnovsky Algorithm:

```
def calc(n):
  t = Decimal(0)
  pi = Decimal(0)
  deno = Decimal(0)
  k = 0
  for k in range(n):
    t = (Decimal(-1)**k)*(math.factorial
(Decimal(6)*k))*(13591409 +545140134*k)
    deno = math.factorial(3*k)*(math.
factorial(k)**Decimal(3))*(640320**(3*k))
    pi += Decimal(t)/Decimal(deno)
  pi = pi * Decimal(12)/Decimal(640320**Decimal(1.5))
  pi = 1/pi
  return str(pi)
```

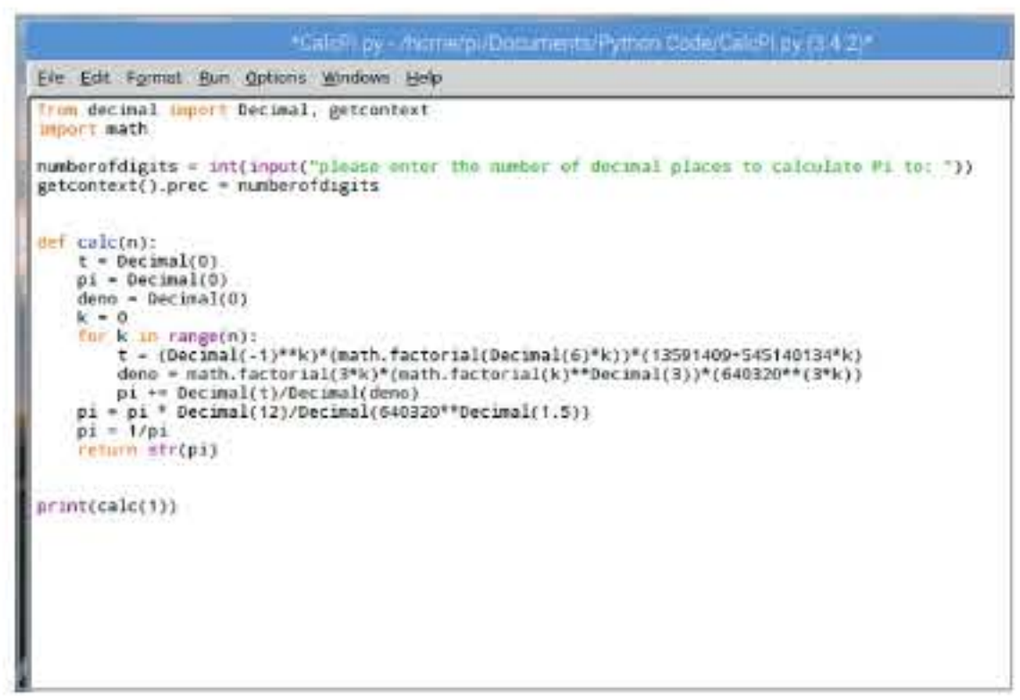
STEP 4 The previous step defines the rules that make up the algorithm and creates the string that will eventually display the value of Pi, according the Chudnovsky brothers' algorithm. You have no doubt already surmised that it would be handy to actually output the value of Pi to the screen. To rectify that you can add:

```
print(calc(1))
```

STEP 5 You can save and execute the code at this point if you like. The output will print the value of Pi to 27 decimal places: **3.141592653589734207668453591**. Whilst pretty impressive on its own, you want some user interaction, to ask the user as to how many places Pi should be calculated.

STEP 6 You can insert an input line before the Pi calculation Def command. It needs to be an integer, as it will otherwise default to a string. We can call it numberofdigits and use the getcontext function:

```
numberofdigits = int(input("please enter the
number of decimal place to calculate Pi to: "))
getcontext().prec = numberofdigits
```



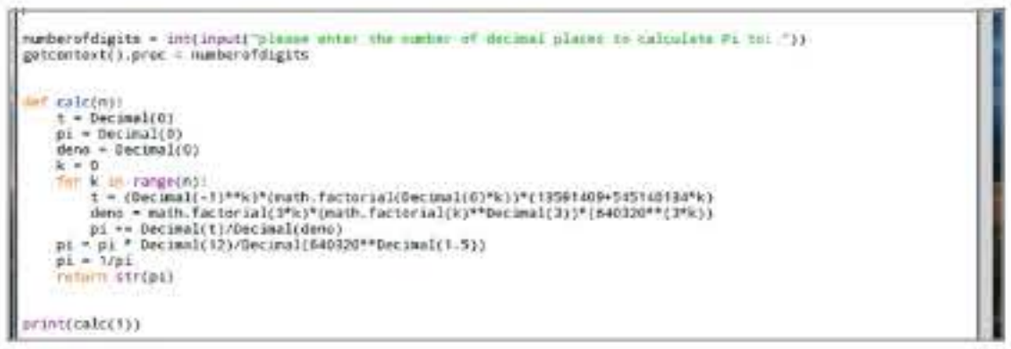
STEP 7 You can execute the code now and it asks the user how many decimal places they want to calculate Pi to, outputting the result in the IDLE Shell. Try it with 1000 places but don't go too high or else your computer will be locked up in calculating Pi.

STEP 8 Part of programming is being able to modify code, making it more presentable. Let's include an element that times how long it takes our computer to calculate the Pi decimal places and present the information in a different colour. For this, drop into the command line and import the colorama module (RPi users already have it installed):

```
pip install colorama
```

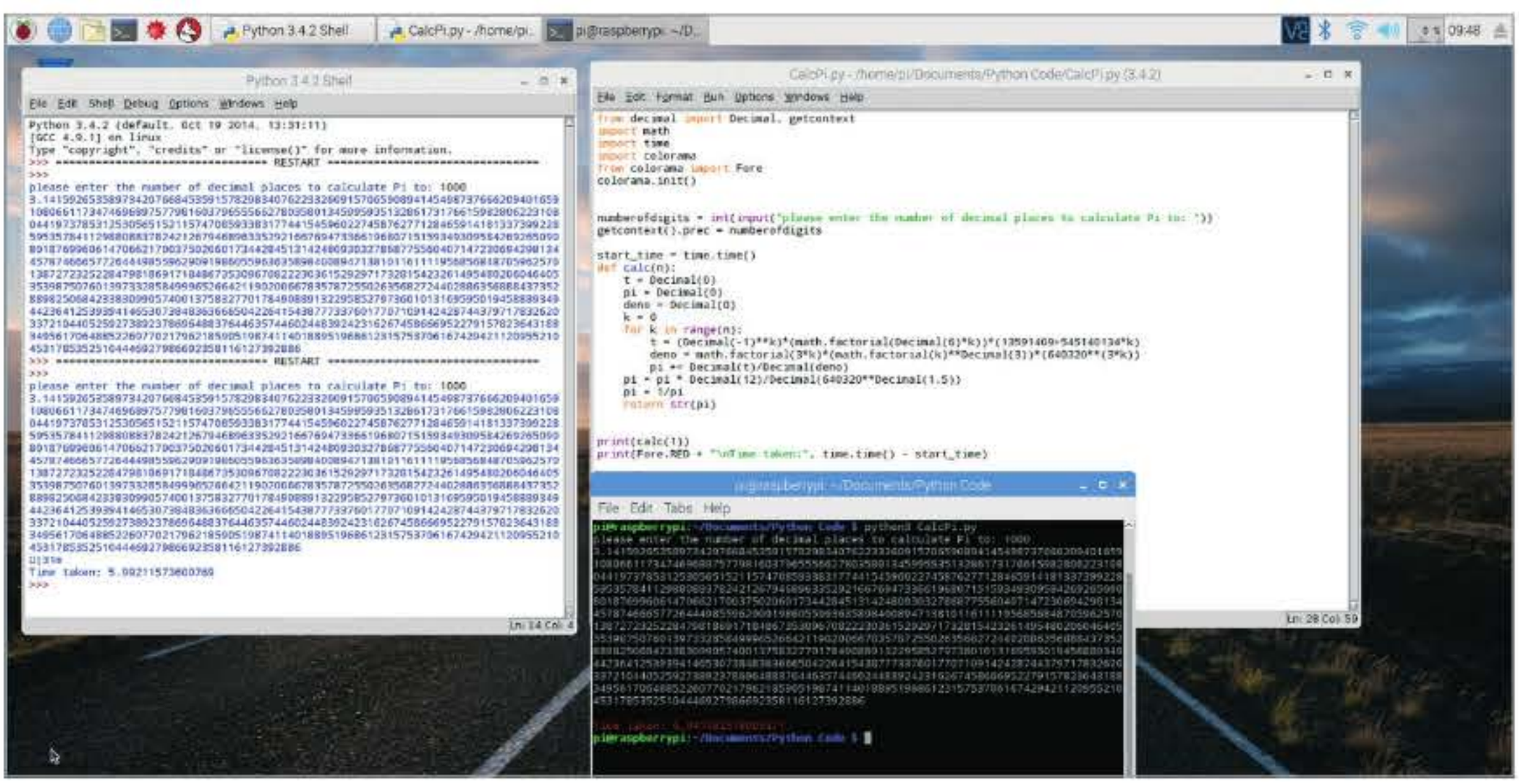
STEP 9 Now we need to import the colorama module (which will output text in different colours) along with the Fore function (which dictates the foreground, ink, colour) and the time module to start a virtual stopwatch to see how long our calculations take:

```
import time
import colorama
from colorama import Fore
```



STEP 10 To finish our code, we need to initialise the colorama module and start the time function at the point where the calculation starts, and when it finishes. The end result is a coloured ink displaying how long the process took (in the Terminal or command line):

```
from decimal import Decimal, getcontext
import math
import time
import colorama
from colorama import Fore
colorama.init()

numberofdigits = int(input("please enter the number
of decimal places to calculate Pi to: "))
getcontext().prec = numberofdigits

start_time = time.time()
def calc(n):
    t = Decimal(0)
    pi = Decimal(0)
    deno = Decimal(0)
    k = 0
    for k in range(n):
        t = (Decimal(-1)**k)*(math.
factorial(Decimal(6)*k))*(13591409+545140134*k)
        deno = math.factorial(3*k)*(math.
factorial(k)**Decimal(3))*(640320**(3*k))
        pi += Decimal(t)/Decimal(deno)
    pi = pi * Decimal(12)/
Decimal(640320**Decimal(1.5))
    pi = 1/pi
    return str(pi)

print(calc(1))
print(Fore.RED + "\nTime taken:", time.time() -
start_time)
```



Working with Data

Data is everything. With it you can display, control, add, remove, create and manipulate Python to your every demand. Over these coming pages we look at how you can create lists, tuples, dictionaries, multi-dimensional lists and how you can use them to forge exciting and useful programs.

We take a look at how you can use date and time functions, write to files in your system and even create graphical user interfaces that will take your coding skills to new levels.

52 Lists
54 Tuples
56 Dictionaries
58 Splitting and Joining Strings
60 Formatting Strings
62 Date and Time
64 Opening Files
66 Writing to Files
68 Exceptions
70 Python Graphics
72 Combining What You Know So Far

Lists

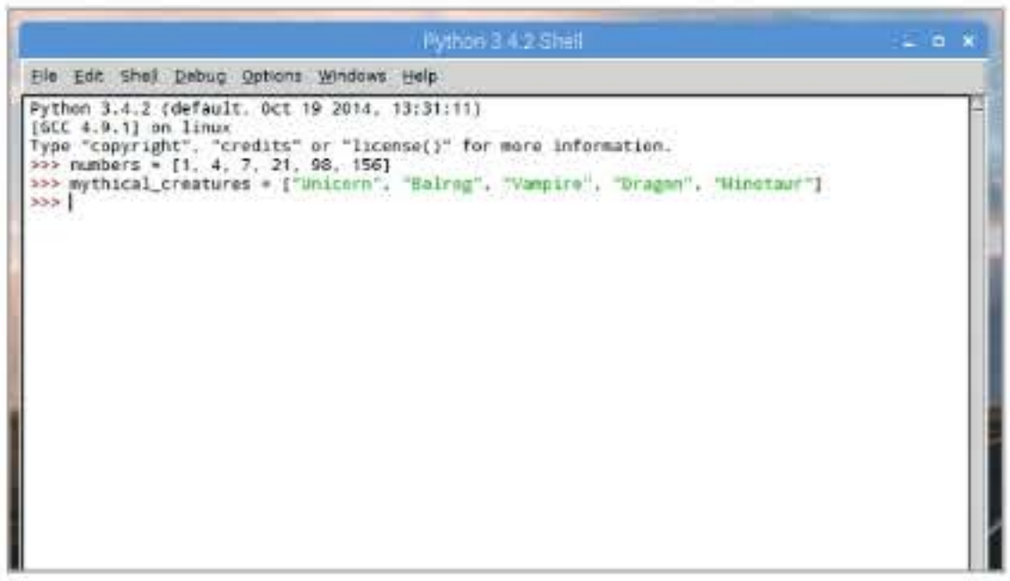
Lists are one of the most common types of data structures you will comes across in Python. A list is simply a collection of items, or data if you prefer, that can be accessed as a whole, or individually if wanted.

WORKING WITH LISTS

Lists are extremely handy in Python. A list can be strings, integers and also variables. You can even include functions in lists, and lists within lists.

STEP 1 A list is a sequence of data values called items. You create the name of your list followed by an equals sign, then square brackets and the items separated by commas; note that strings use quotes:

```
numbers = [1, 4, 7, 21, 98, 156]
mythical_creatures – [“Unicorn”, “Balrog”, “Vampire”, “Dragon”, “Minotaur”]
```



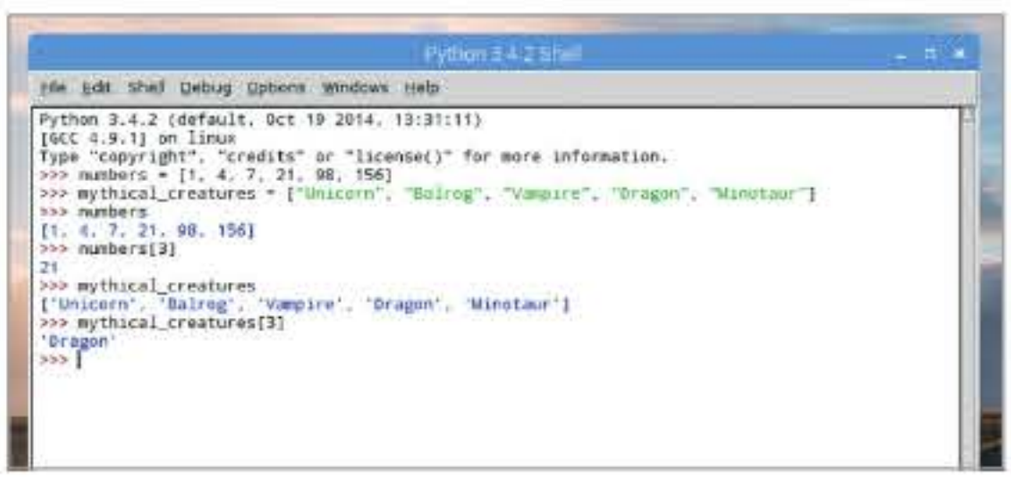
STEP 2 Once you've defined your list you can call each by referencing its name, followed by a number. Lists start the first item entry as 0, followed by 1, 2, 3 and so on. For example:

```
numbers
```

To call up the entire contents of the list.

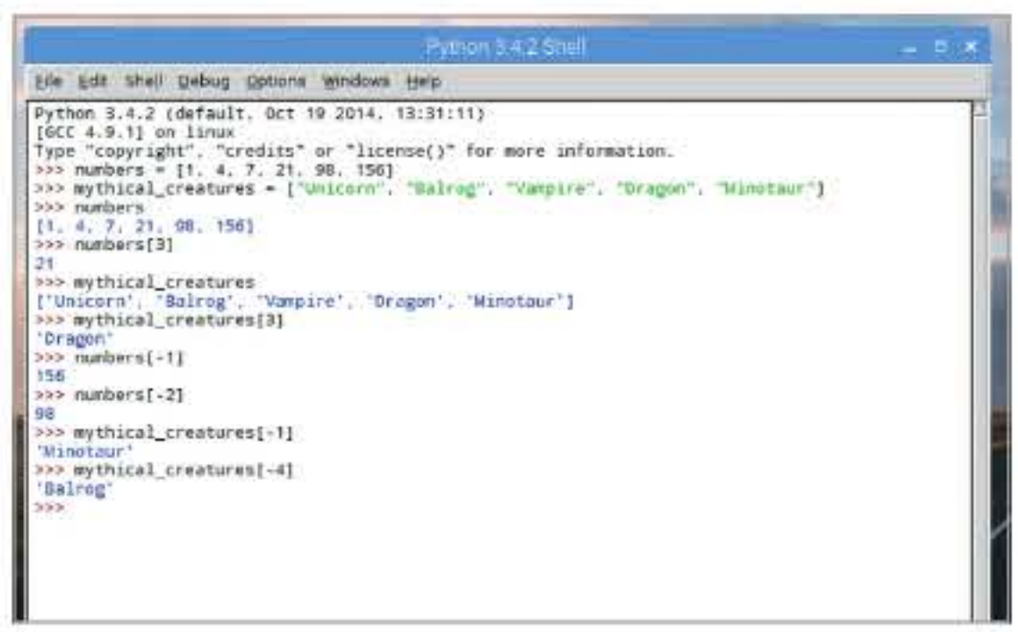
```
numbers[3]
```

To call the third from zero item in the list (21 in this case).



STEP 3 You can also access, or index, the last item in a list by using the minus sign before the item number [-1], or the second to last item with [-2] and so on. Trying to reference an item that isn't in the list, such as [10] will return an error:

```
numbers[-1]
mythical_creatures[-4]
```



STEP 4 Slicing is similar to indexing but you can retrieve multiple items in a list by separating item numbers with a colon. For example:

```
numbers[1:3]
```

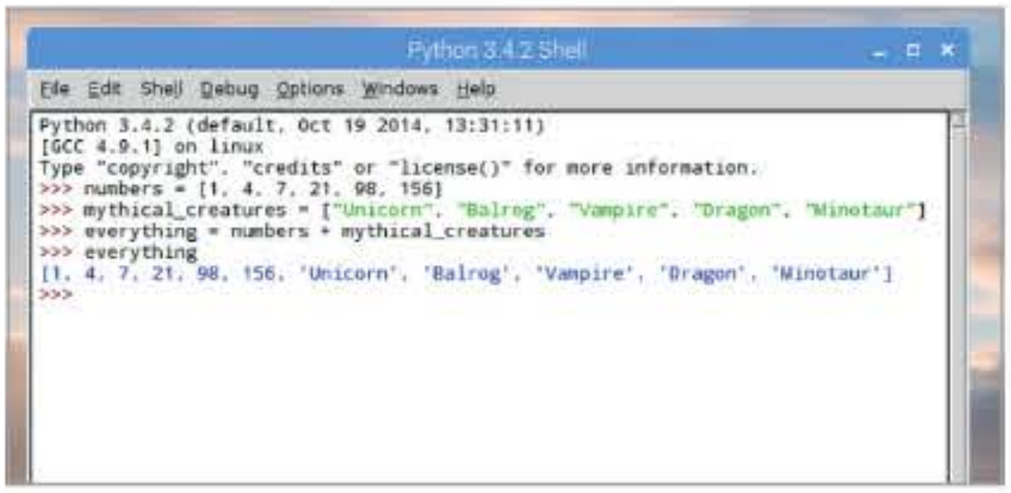
Will output the 4 and 7, being item numbers 1 and 2. Note that the returned values don't include the second index position (as you would numbers[1:3] to return 4, 7 and 21).

STEP 5 You can update items within an existing list, remove items and even join lists together. For example, to join two lists you can use:

```
everything = numbers + mythical_creatures
```

Then view the combined list with:

```
everything
```



STEP 6 Items can be added to a list by entering:

```
numbers=numbers+[201]
```

Or for strings:

```
mythical_creatres=mythical_creatures+["Griffin"]
```

Or by using the append function:

```
mythical_creatures.append("Nessie")
numbers.append(278)
```

STEP 7 Removal of items can be done in two ways. The first is by the item number:

```
del numbers[7]
```

Alternatively, by item name:

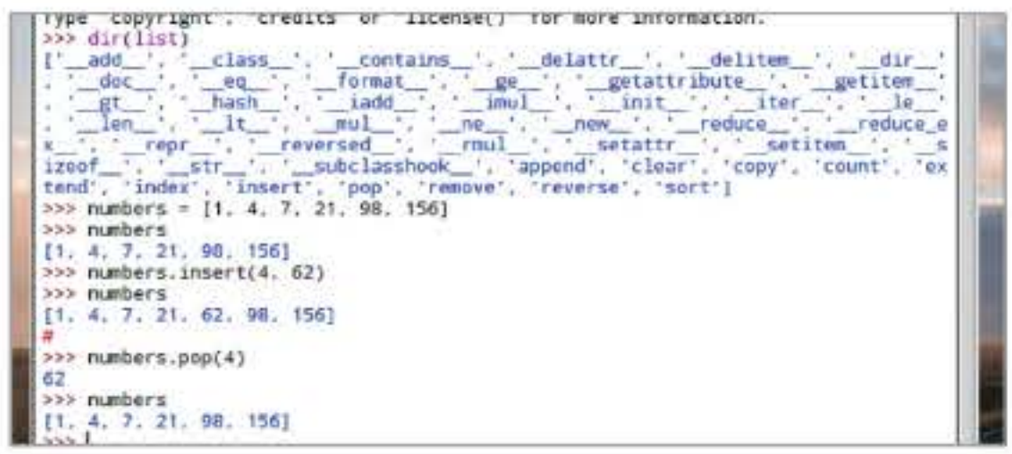
```
mythical_creatures.remove("Nessie")
```

STEP 8 You can view what can be done with lists by entering dir(list) into the Shell. The output is the available functions, for example, insert and pop are used to add and remove items at certain positions. To insert the number 62 at item index 4:

```
numbers.insert(4, 62)
```

To remove it:

```
numbers.pop(4)
```

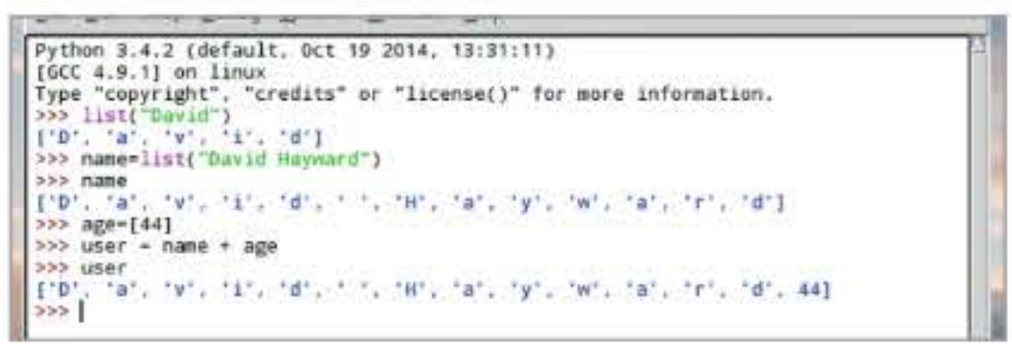


STEP 9 You also use the list function to break a string down into its components. For example:

```
list("David")
```

Breaks the name David into 'D', 'a', 'v', 'i', 'd'. This can then be passed to a new list:

```
name=list("David Hayward")
name
age=[44]
user = name + age
user
```



STEP 10 Based on that, you can create a program to store someone's name and age as a list:

```
name=input("What's your name? ")
lname=list(name)
age=int(input("How old are you: "))
lage=[age]

user = lname + lage
```

The combined name and age list is called user, which can be called by entering user into the Shell. Experiment and see what you can do.

Tuples

Tuples are very much identical to lists. However, where lists can be updated, deleted or changed in some way, a tuple remains a constant. This is called immutable and they're perfect for storing fixed data items.

THE IMMUTABLE TUPLE

Reasons for having tuples vary depending on what the program is intended to do. Normally, a tuple is reserved for something special but they're also used for example, in an adventure game, where non-playing character names are stored.

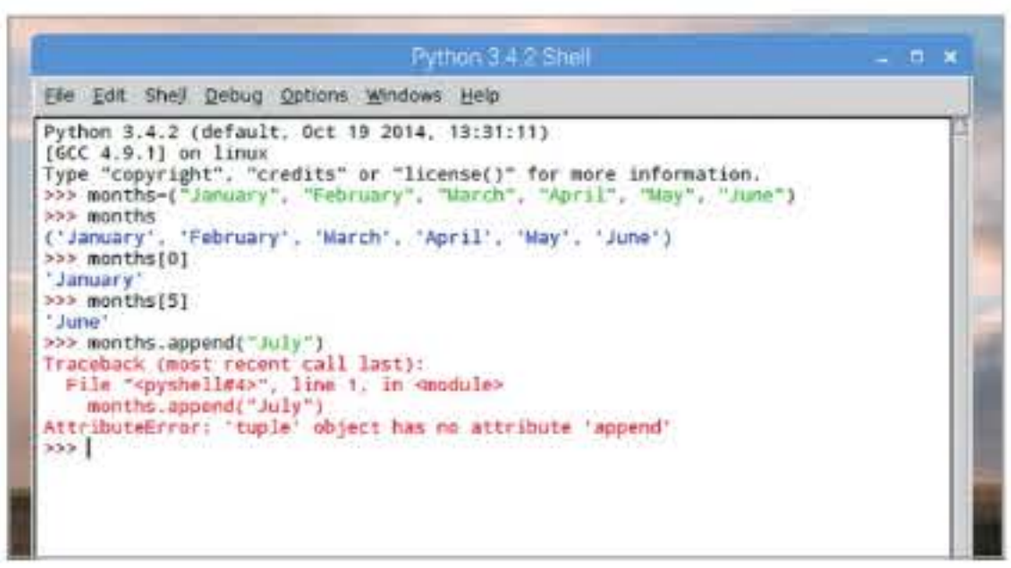
STEP 1 A tuple is created the same way as a list but in this instance you use curved brackets instead of square brackets. For example:

```
months=("January", "February", "March", "April",
"May", "June")
months
```

STEP 2 Just as with lists, the items within a named tuple can be indexed according to their position in the data range, i.e.:

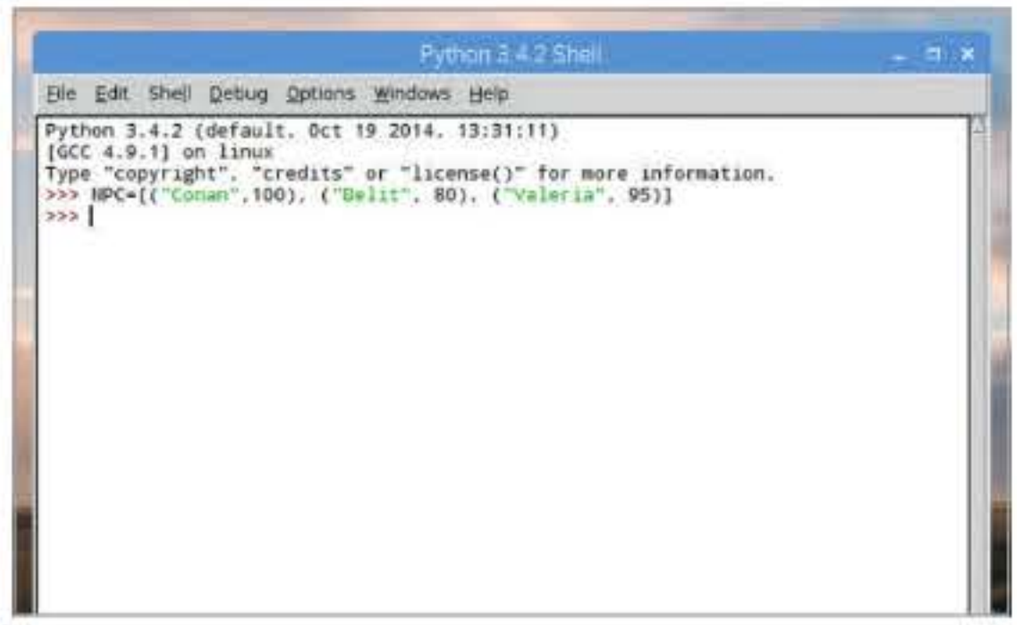
```
months[0]
months[5]
```

However, any attempt at deleting or adding to the tuple will result in an error in the Shell.



STEP 3 You can create grouped tuples into lists that contain multiple sets of data. For instance, here is a tuple called NPC (Non-Playable Characters) containing the character name and their combat rating for an adventure game:

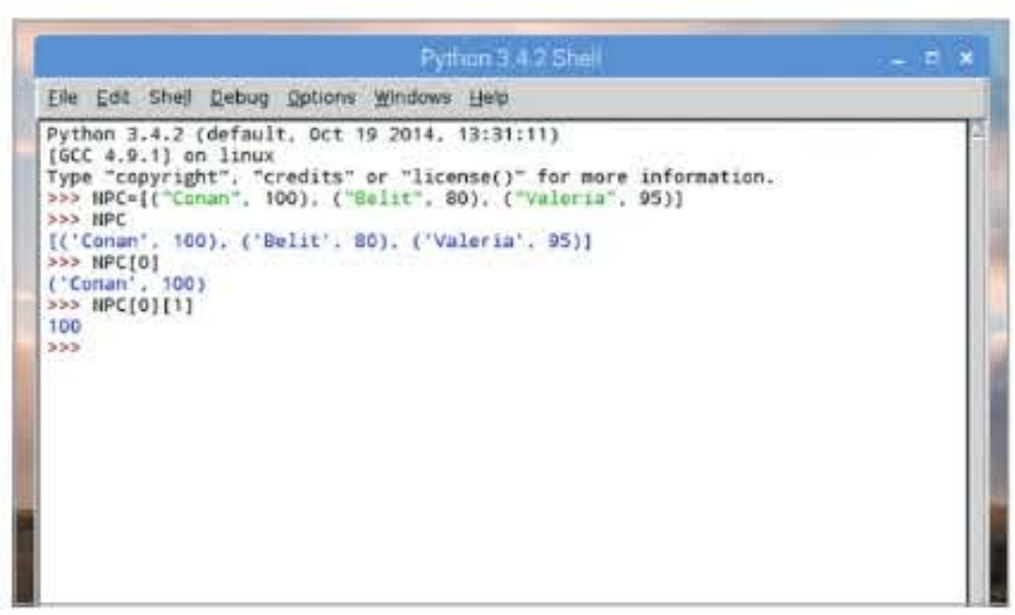
```
NPC=[("Conan", 100), ("Belit", 80),
("Valeria", 95)]
```



STEP 4 Each of these data items can be accessed as a whole by entering NPC into the Shell; or they can be indexed according to their position NPC[0]. You can also index the individual tuples within the NPC list:

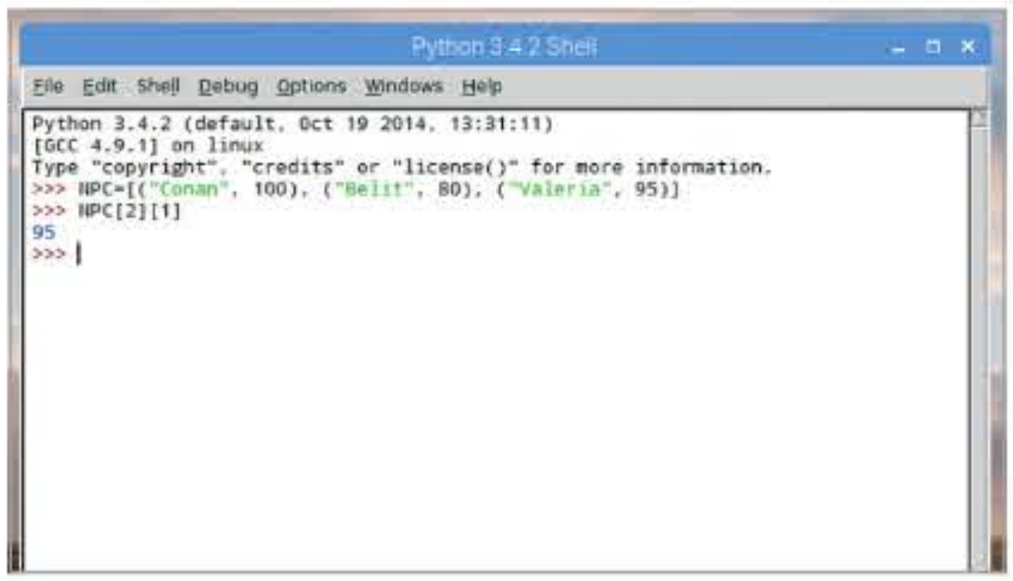
```
NPC[0][1]
```

Will display 100.



STEP 5 It's worth noting that when referencing multiple tuples within a list, the indexing is slightly different from the norm. You would expect the 95 combat rating of the character Valeria to be NPC[4][5], but it's not. It's actually:

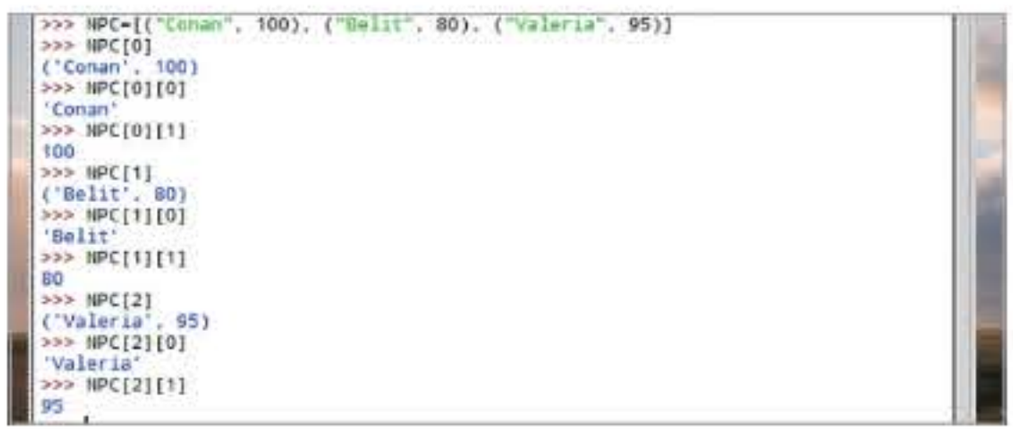
```
NPC[2][1]
```



STEP 6 This means of course that the indexing follows thus:

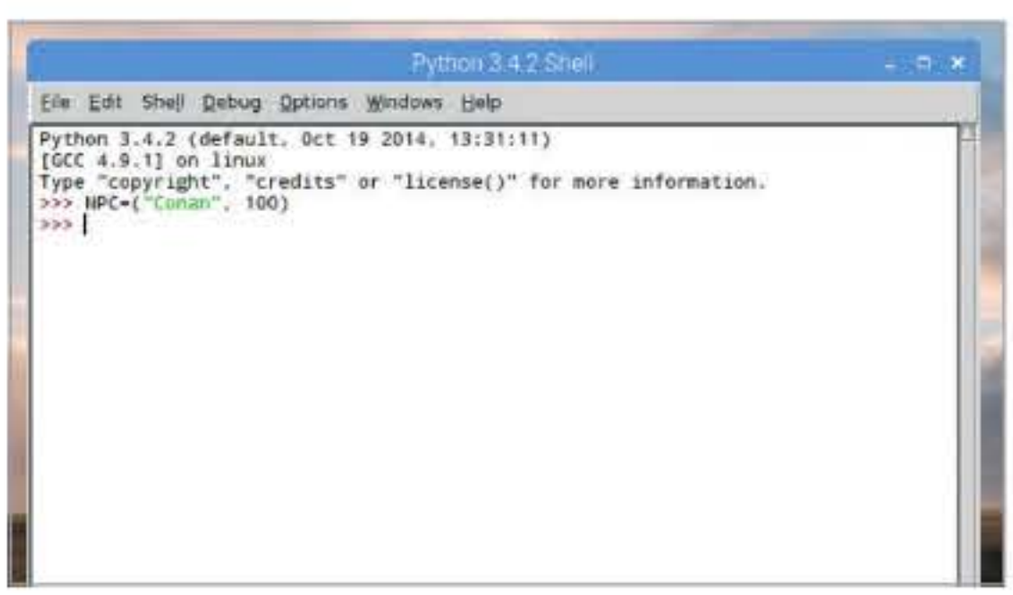
```
0          1, 1
0, 0       2
0, 1       2, 0
1          2,1
1, 0
```

Which as you can imagine, gets a little confusing when you've got a lot of tuple data to deal with.



STEP 7 Tuples though utilise a feature called unpacking, where the data items stored within a tuple are assigned variables. First create the tuple with two items (name and combat rating):

```
NPC=("Conan", 100)
```



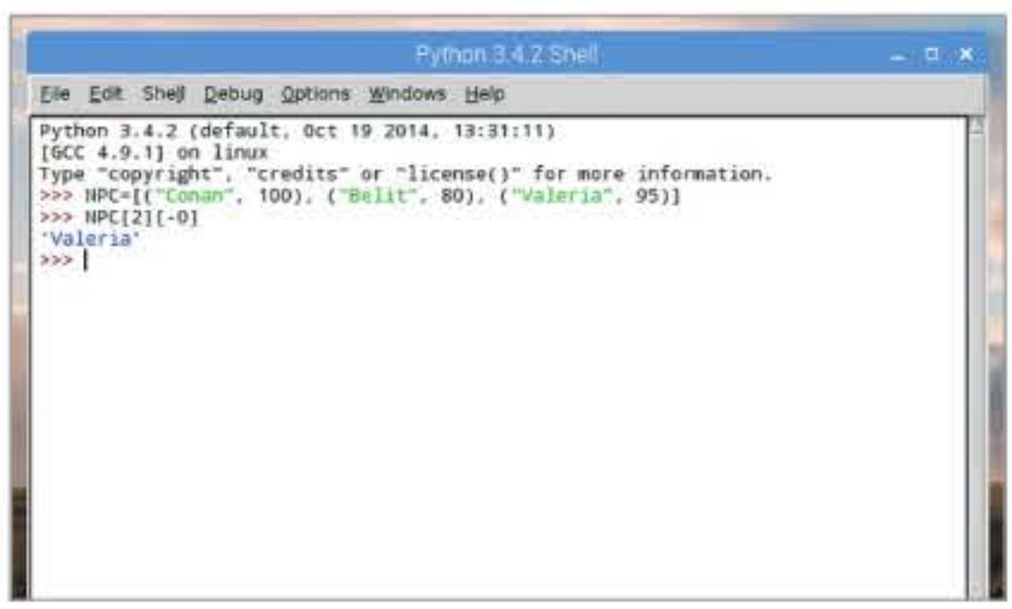
STEP 8 Now unpack the tuple into two corresponding variables:

```
(name, combat_rating)=NPC
```

You can now check the values by entering name and combat_rating.

STEP 9 Remember, as with lists, you can also index tuples using negative numbers which count backwards from the end of the data list. For our example, using the tuple with multiple data items, you would reference the Valeria character with:

```
NPC[2][-0]
```



STEP 10 You can use the max and min functions to find the highest and lowest values of a tuple composed of numbers. For example:

```
numbers=(10.3, 23, 45.2, 109.3, 6.1, 56.7, 99)
```

The numbers can be integers and floats. To output the highest and lowest, use:

```
print(max(numbers))
print(min(numbers))
```

Dictionaries

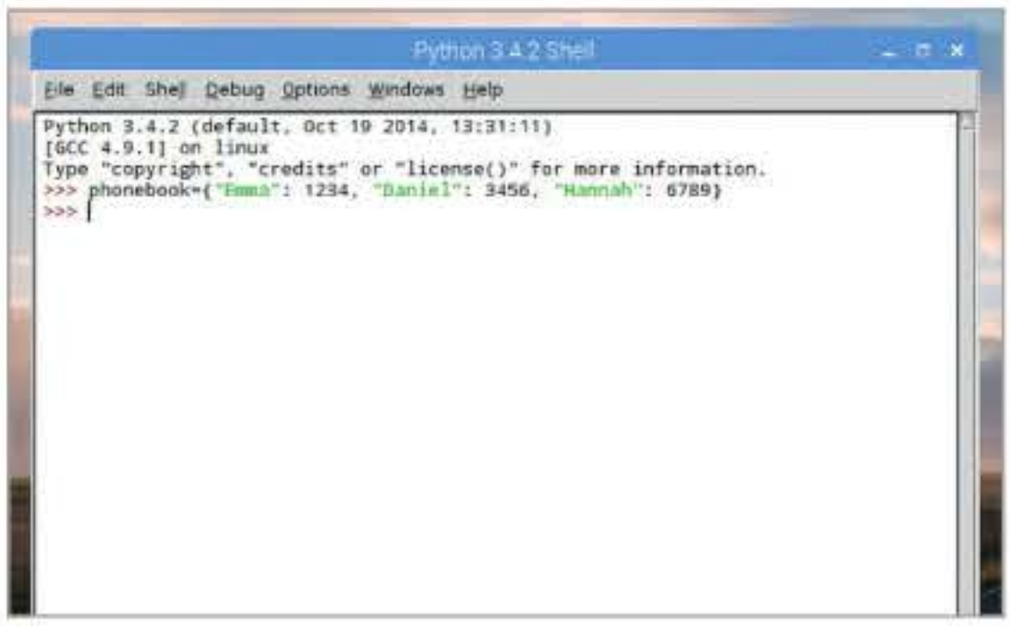
Lists are extremely useful but dictionaries in Python are by far the more technical way of dealing with data items. They can be tricky to get to grips with at first but you'll soon be able to apply them to your own code.

KEY PAIRS

A dictionary is like a list but instead each data item comes as a pair, these are known as Key and Value. The Key part must be unique and can either be a number or string whereas the Value can be any data item you like.

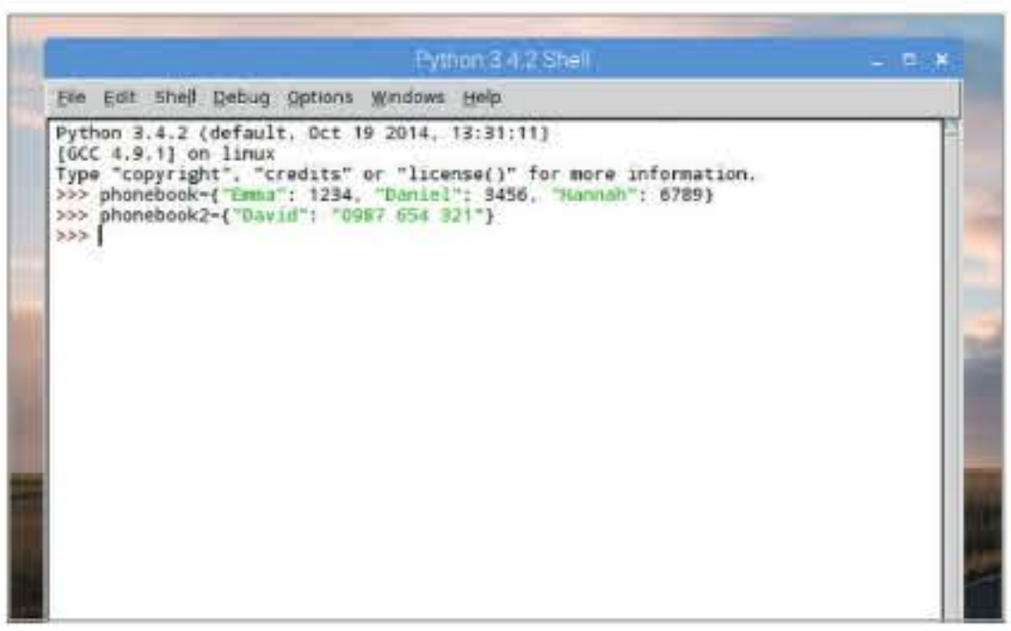
STEP 1 Let's say you want to create a phonebook in Python. You would create the dictionary name and enter the data in curly brackets, separating the key and value by a colon `Key:Value`. For example:

```
phonebook={"Emma": 1234, "Daniel": 3456,
"Hannah": 6789}
```

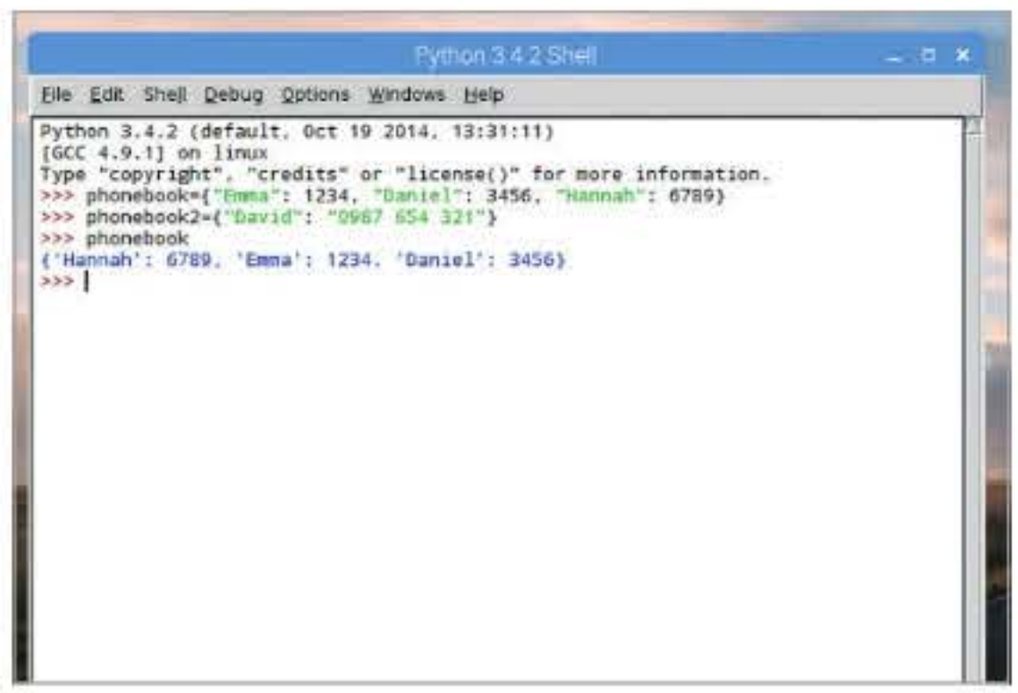


STEP 2 Just as with most lists, tuples and so on, strings need be enclosed in quotes (single or double), whilst integers can be left open. Remember that the value can be either a string or an integer, you just need to enclose the relevant one in quotes:

```
phonebook2={"David": "0987 654 321"}
```

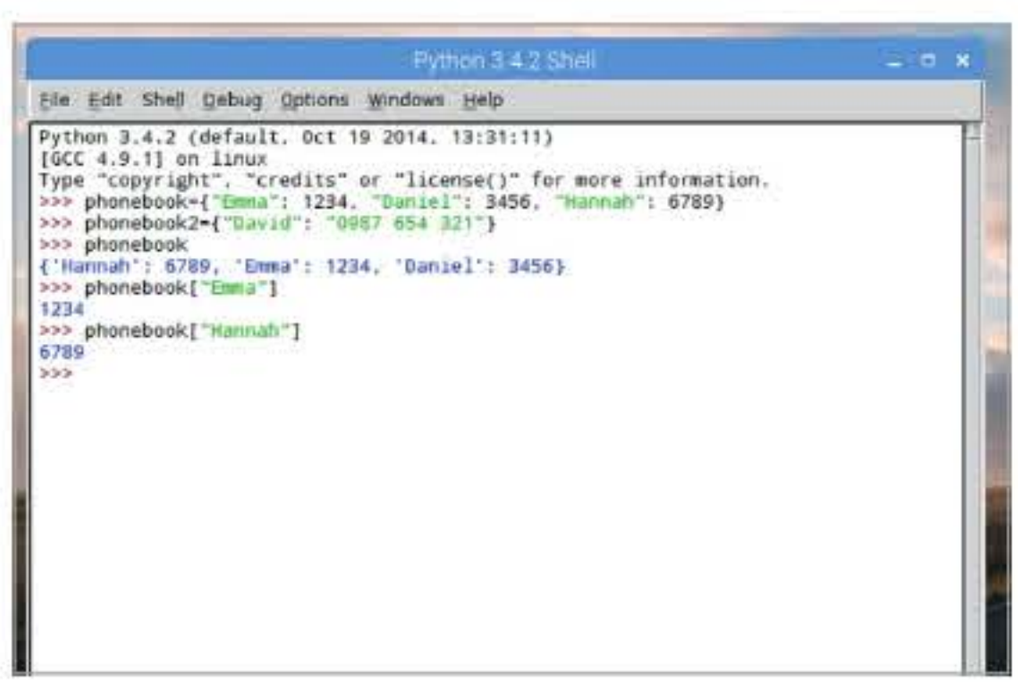


STEP 3 As with lists and tuples, you can check the contents of a dictionary by giving the dictionary a name: phonebook, in this example. This will display the data items you've entered in a similar fashion to a list, which you're no doubt familiar with by now.



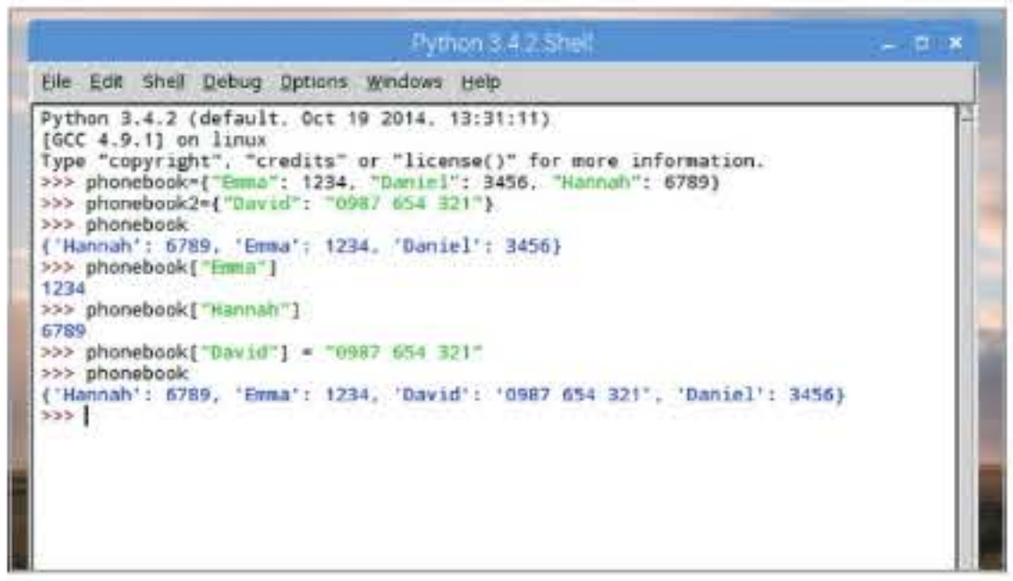
STEP 4 The benefit of using a dictionary is that you can enter the key to index the value. Using the phonebook example from the previous steps, you can enter:

```
phonebook["Emma"]
phonebook["Hannah"]
```



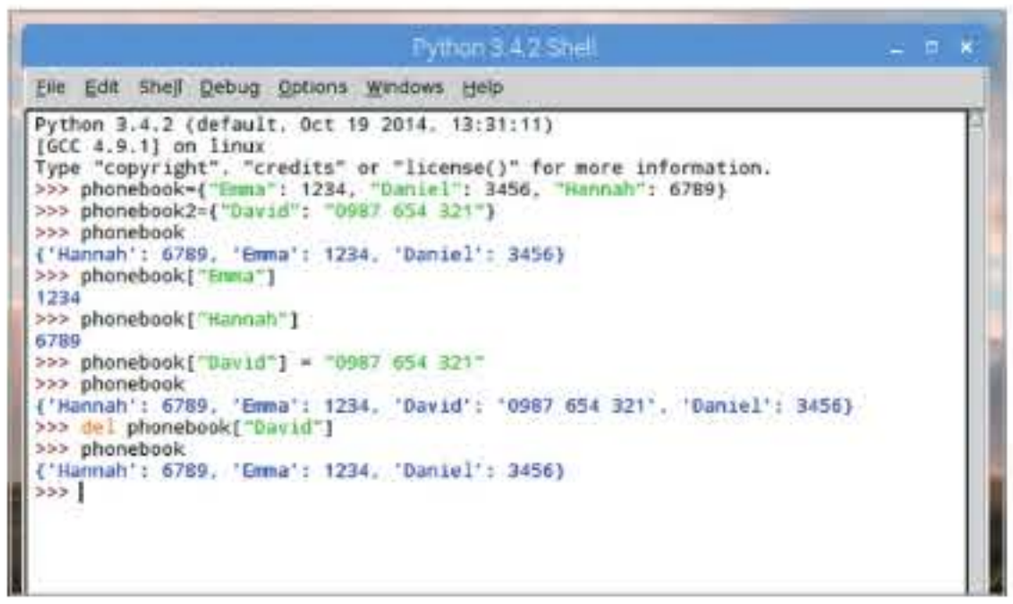
STEP 5 Adding to a dictionary is easy too. You can include a new data item entry by adding the new key and value items like:

```
phonebook["David"] = "0987 654 321"
phonebook
```



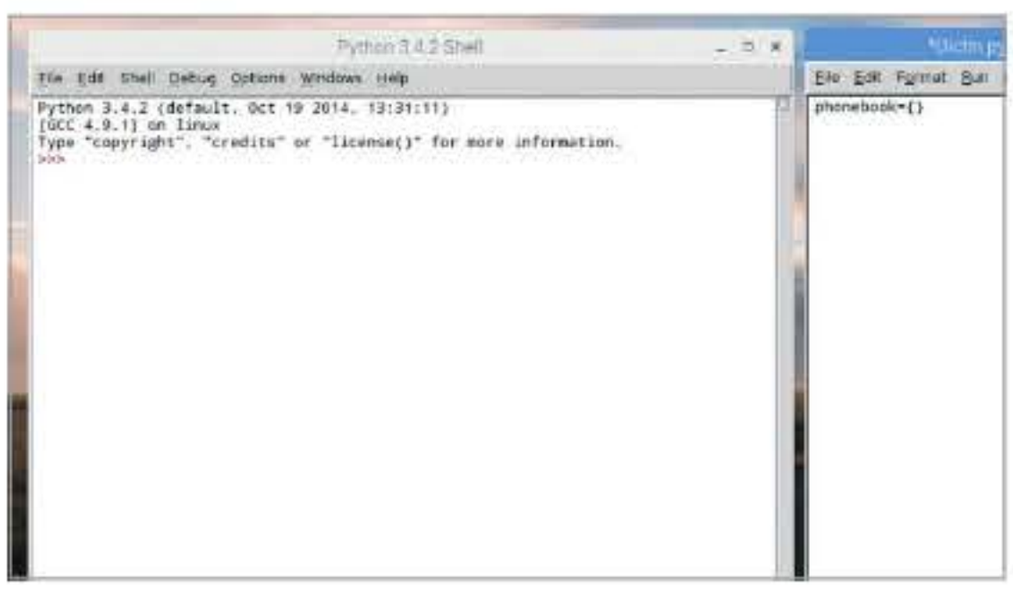
STEP 6 You can also remove items from a dictionary by issuing the del command followed by the item's key; the value will be removed as well, since both work as a pair of data items:

```
del phonebook["David"]
```



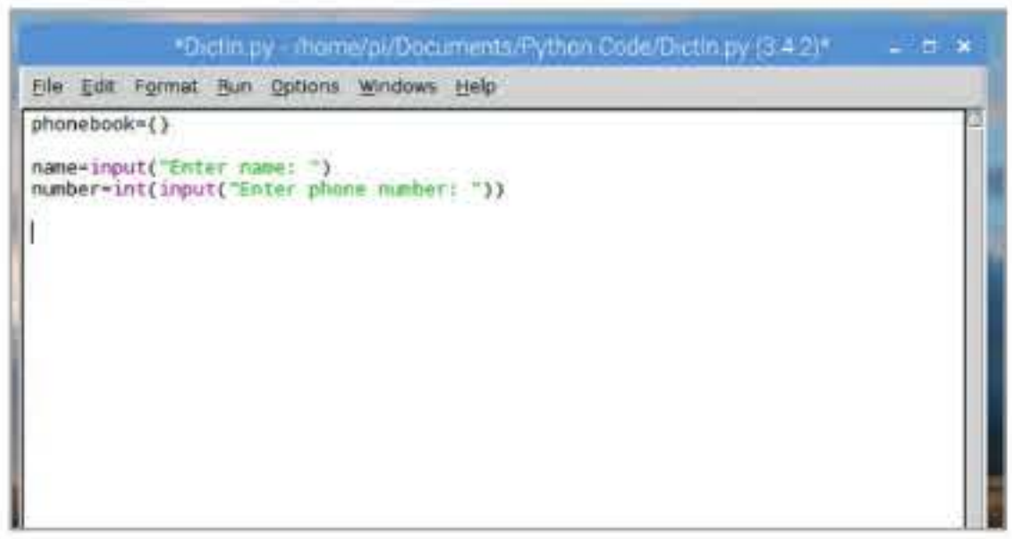
STEP 7 Taking this a step further, how about creating a piece of code that will ask the user for the dictionary key and value items? Create a new Editor instance and start by coding in a new, blank dictionary:

```
phonebook={}
```



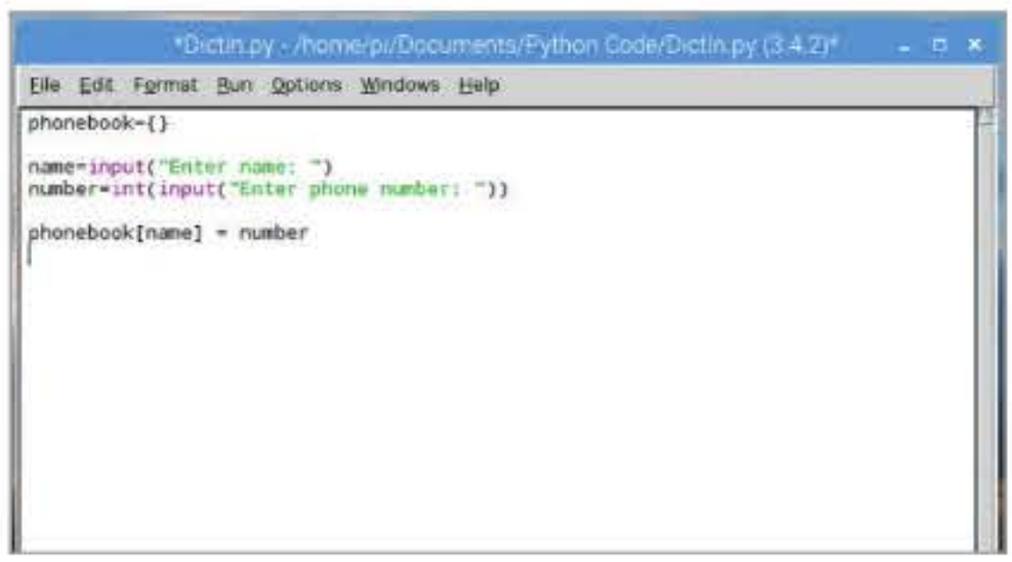
STEP 8 Next, you need to define the user inputs and variables: one for the person's name, the other for their phone number (let's keep it simple to avoid lengthy Python code):

```
name=input("Enter name: ")
number=int(input("Enter phone number: "))
```



STEP 9 Note we've kept the number as an integer instead of a string, even though the value can be both an integer or a string. Now you need to add the user's inputted variables to the newly created blank dictionary. Using the same process as in Step 5, you can enter:

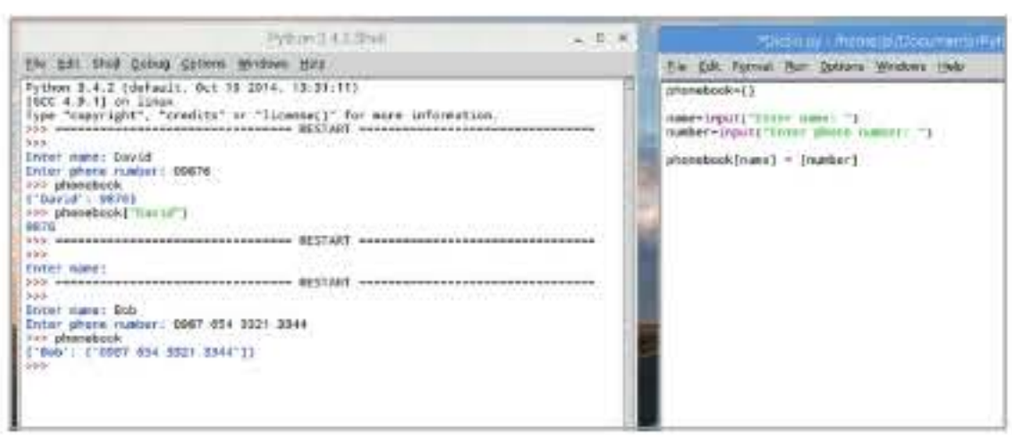
```
phonebook[name] = number
```



STEP 10 Now when you save and execute the code, Python will ask for a name and a number. It will then insert those entries into the phonebook dictionary, which you can test by entering into the Shell:

```
phonebook
phonebook["David"]
```

If the number needs to contain spaces you need to make it a string, so remove the int part of the input.



Splitting and Joining Strings

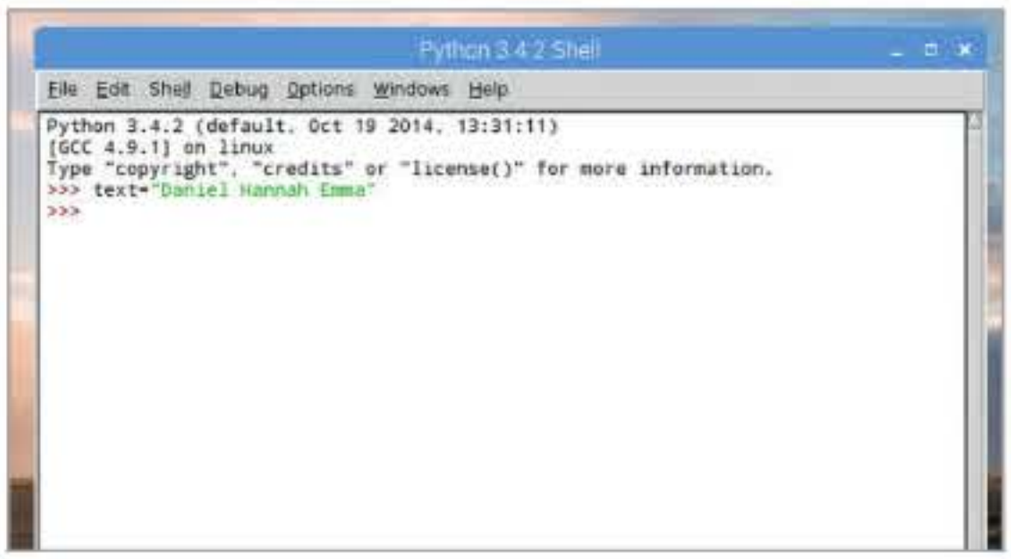
When dealing with data in Python, especially from a user's input, you will undoubtedly come across long sets of strings. A useful skill to learn in Python programming is being able to split those long strings for better readability.

STRING THEORIES

You've already looked at some list functions, using .insert, .remove, and .pop but there are also functions that can be applied to strings.

STEP 1 The main tool in the string function arsenal is .split(). With it you're able to split apart a string of data, based on the argument within the brackets. For example, here's a string with three items, each separated by a space:

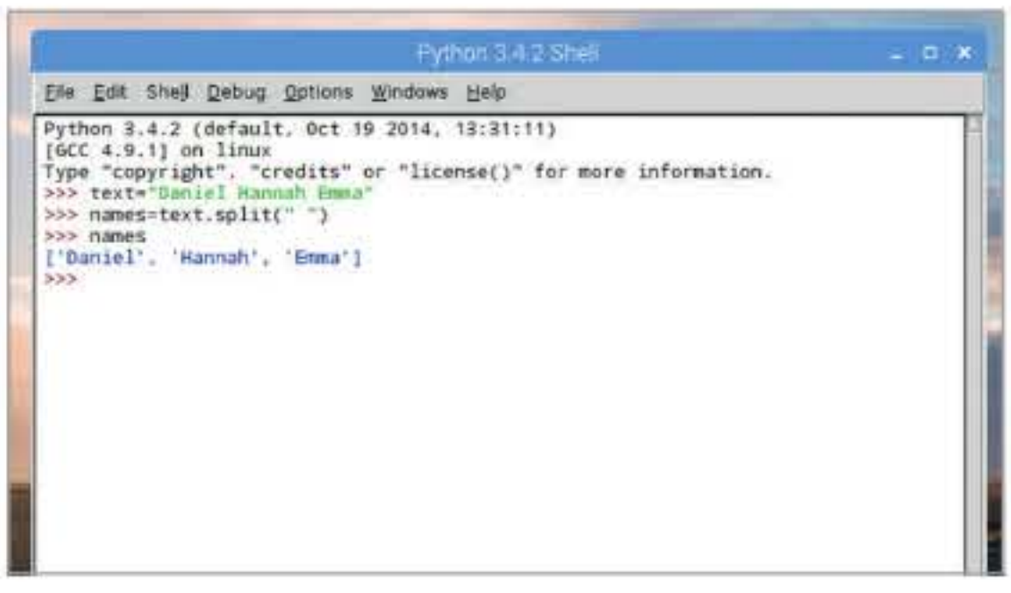
```
text="Daniel Hannah Emma"
```



STEP 2 Now let's turn the string into a list and split the content accordingly:

```
names=text.split(" ")
```

Then enter the name of the new list, names, to see the three items.



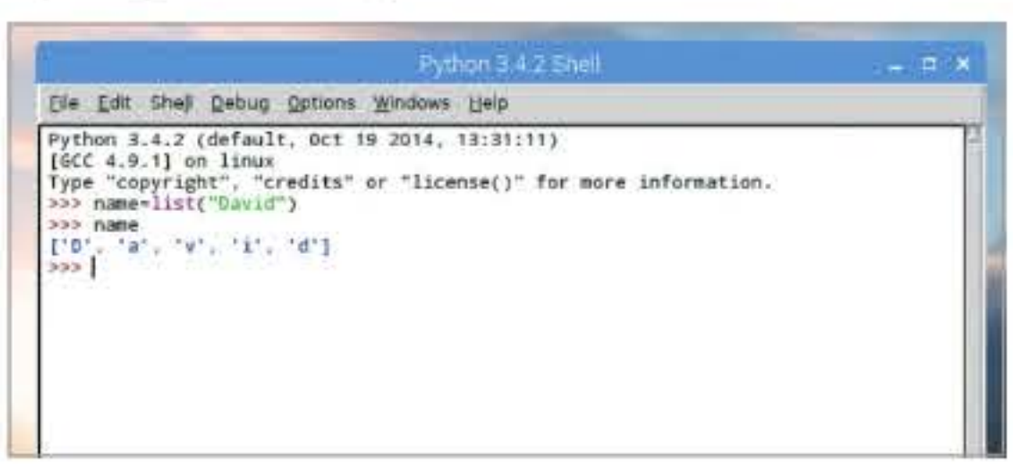
STEP 3 Note that the text.split part has the brackets, quotes, then a space followed by closing quotes and brackets. The space is the separator, indicating that each list item entry is separated by a space. Likewise, CSV (Comma Separated Value) content has a comma, so you'd use:

```
text="January,February,March,April,May,June"
months=text.split(",")
months
```

STEP 4 You've previously seen how you can split a string into individual letters as a list, using a name:

```
name=list("David")
name
```

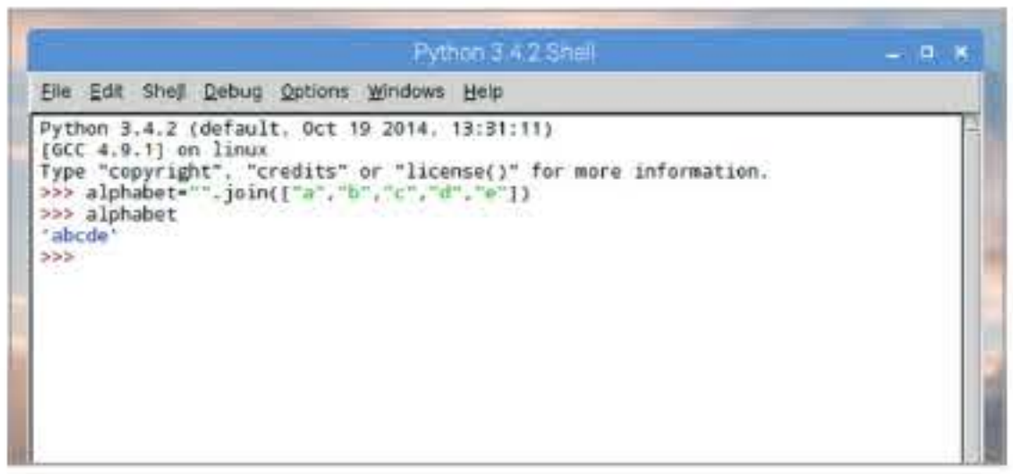
The returned value is 'D', 'a', 'v', 'i', 'd'. Whilst it may seem a little useless under ordinary circumstances, it could be handy for creating a spelling game for example.



STEP 5 The opposite of the .split function is .join, where you will have separate items in a string and can join them all together to form a word or just a combination of items, depending on the program you're writing. For instance:

```
alphabet="".join(["a","b","c","d","e"])
alphabet
```

This will display 'abcde' in the Shell.



STEP 6 You can therefore apply .join to the separated name you made in Step 4, combining the letters again to form the name:

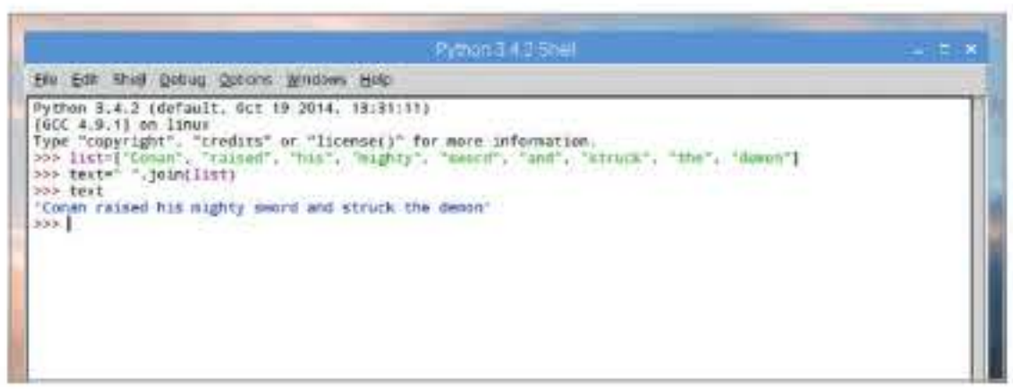
```
name="".join(name)
name
```

We've joined the string back together, and retained the list called name, passing it through the .join function.

STEP 7 A good example of using the .join function is when you have a list of words you want to combine into a sentence:

```
list=["Conan", "raised", "his", "mighty", "sword",
"and", "struck", "the", "demon"]
text=" ".join(list)
text
```

Note the space between the quotes before the .join function (where there were no quotes in Step 6's .join).



STEP 8 As with the .split function, the separator doesn't have to be a space, it can also be a comma, a full stop, a hyphen or whatever you like:

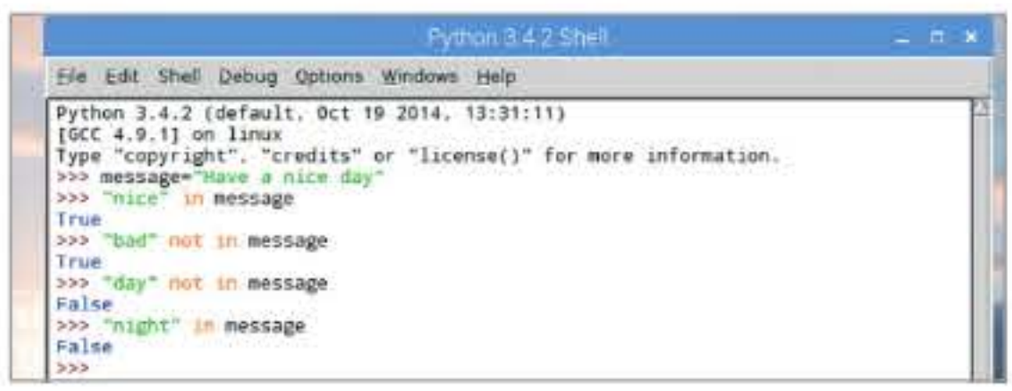
```
colours=["Red", "Green", "Blue"]
col=",".join(colours)
col
```

STEP 9 There's some interesting functions you apply to a string, such as .capitalize and .title. For example:

```
title="conan the cimmerian"
title.capitalize()
title.title()
```

STEP 10 You can also use logic operators on strings, with the 'in' and 'not in' functions. These enable you to check if a string contains (or does not contain) a sequence of characters:

```
message="Have a nice day"
"nice" in message

"bad" not in message
"day" not in message
"night" in message
```



Formatting Strings

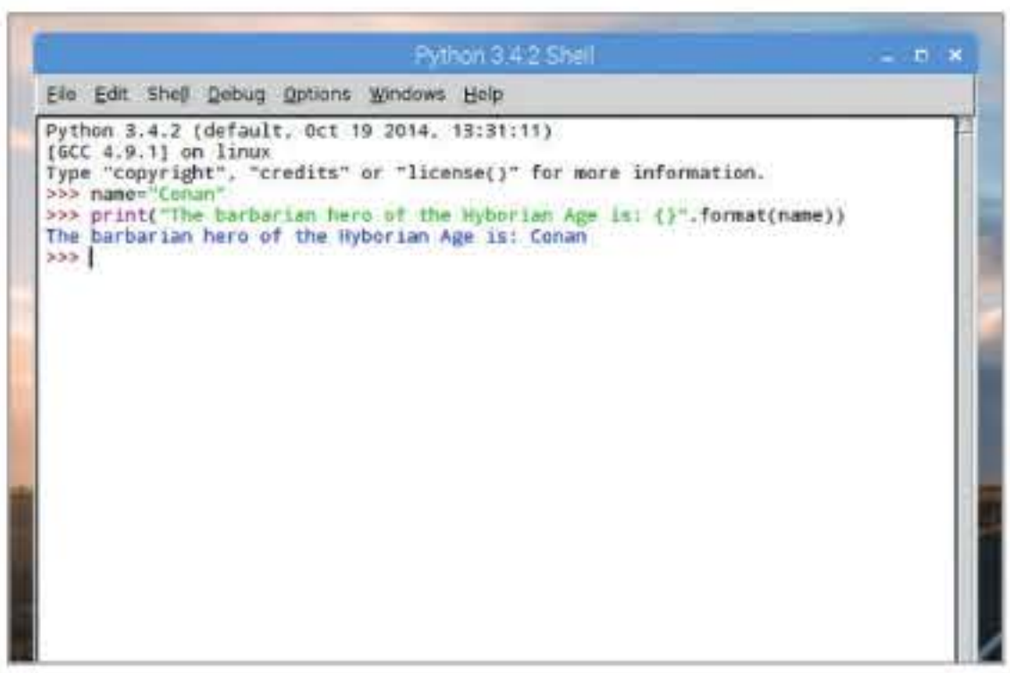
When you work with data, creating lists, dictionaries and objects you may often want to print out the results. Merging strings with data is easy especially with Python 3, as earlier versions of Python tended to complicate matters.

STRING FORMATTING

Since Python 3, string formatting has become a much neater process, using the .format function combined with curly brackets. It's a more logical and better formed approach than previous versions.

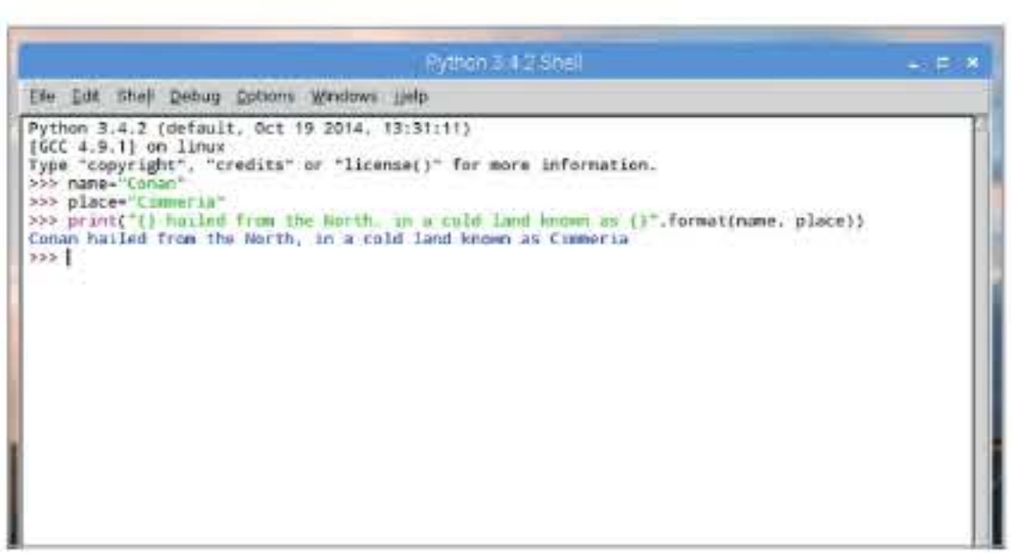
STEP 1 The basic formatting in Python is to call each variable into the string using the curly brackets:

```
name="Conan"
print("The barbarian hero of the Hyborian Age is: {}".format(name))
```



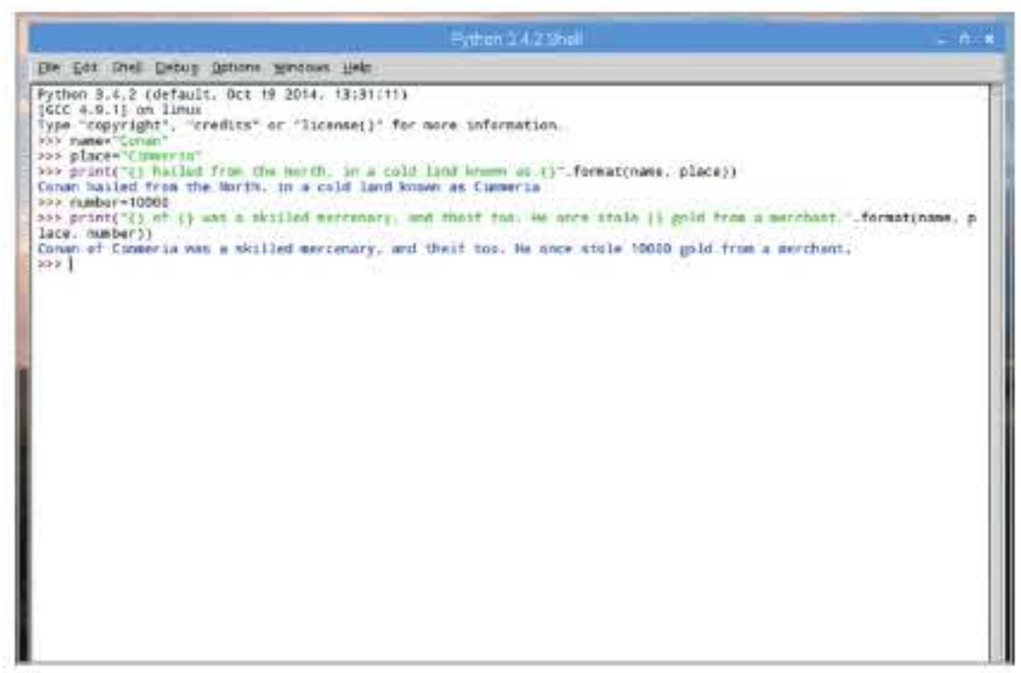
STEP 2 Remember to close the print function with two sets of brackets, as you've encased the variable in one, and the print function in another. You can include multiple cases of string formatting in a single print function:

```
name="Conan"
place="Cimmeria"
print("{} hailed from the North, in a cold land known as {}".format(name, place))
```



STEP 3 You can of course also include integers into the mix:

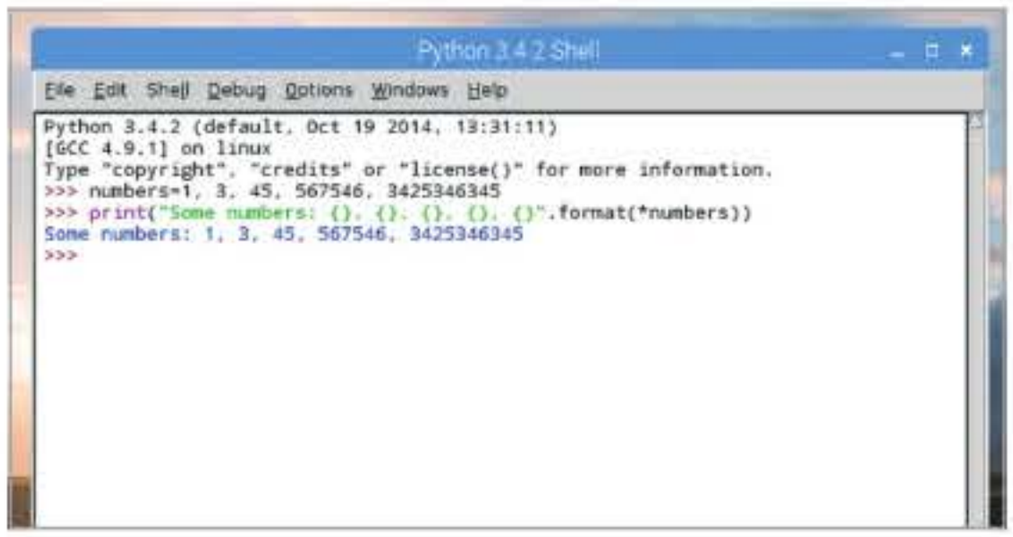
```
number=10000
print("{} of {} was a skilled mercenary, and thief too. He once stole {} gold from a merchant.".format(name, place, number))
```



STEP 4 There are many different ways to apply string formatting, some are quite simple, as we've shown you here; others can be significantly more complex. It all depends on what you want from your program. A good place to reference frequently regarding string formatting is the Python Docs webpage, found at www.docs.python.org/3.1/library/string.html. Here, you will find tons of help.

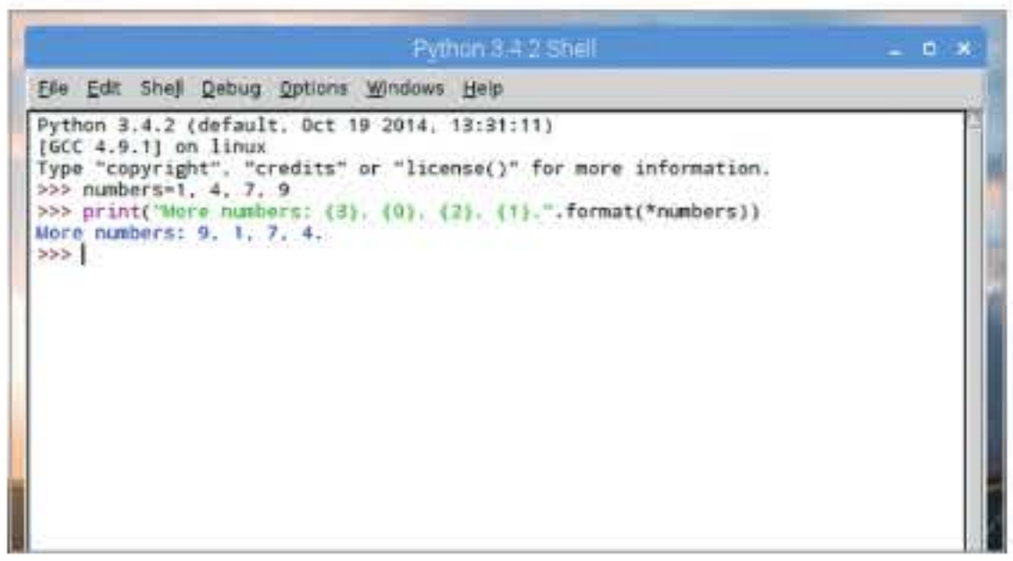
STEP 5 Interestingly you can reference a list using the string formatting function. You need to place an asterisk in front of the list name:

```
numbers=1, 3, 45, 567546, 3425346345
print("Some numbers: {}, {}, {}, {}, {}".
format(*numbers))
```



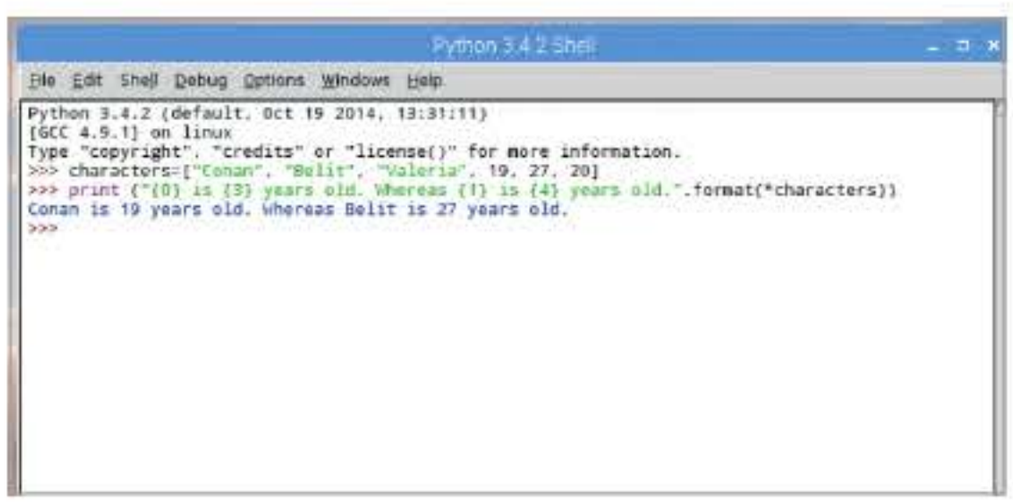
STEP 6 With indexing in lists, the same applies to calling a list using string formatting. You can index each item according to its position (from 0 to however many are present):

```
numbers=1, 4, 7, 9
print("More numbers: {3}, {0}, {2},
{1}.".format(*numbers))
```



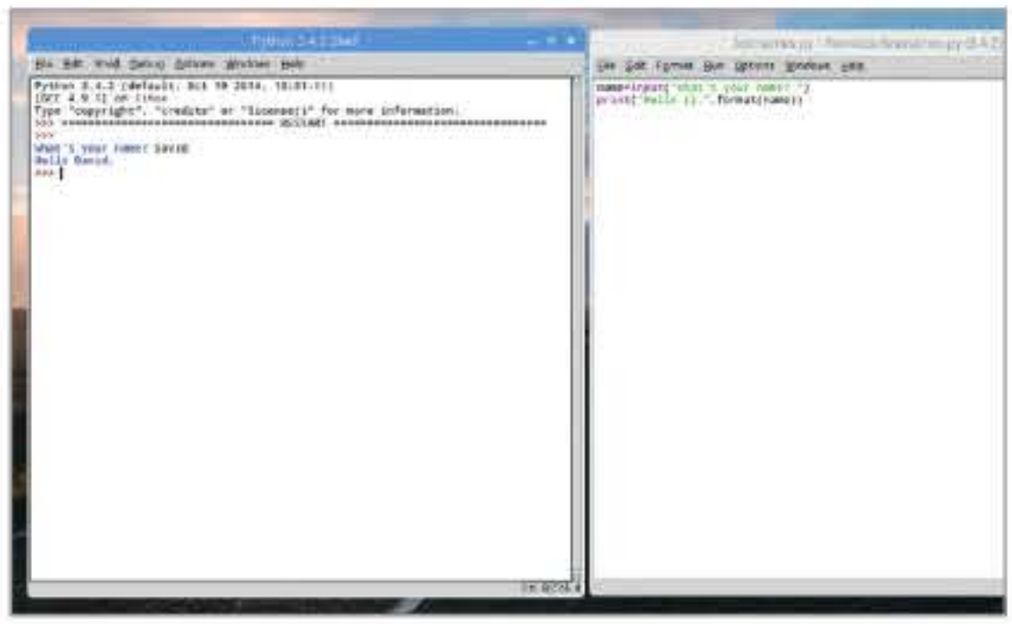
STEP 7 And as you probably suspect, you can mix strings and integers in a single list to be called in the .format function:

```
characters=["Conan", "Belit", "Valeria",
19, 27, 20]
print ("{0} is {3} years old. Whereas {1} is {4}
years old.".format(*characters))
```



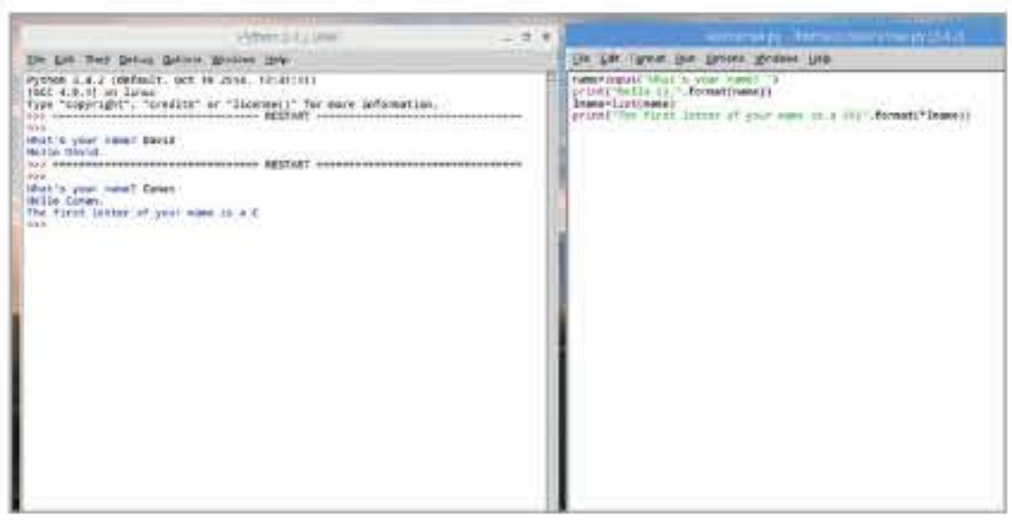
STEP 8 You can also print out the content of a user's input in the same fashion:

```
name=input("What's your name? ")
print("Hello {}.".format(name)
```



STEP 9 You can extend this simple code example to display the first letter in a person's entered name:

```
name=input("What's your name? ")
print("Hello {}.".format(name))
lname=list(name)
print("The first letter of your name is a {0}".
format(*lname))
```

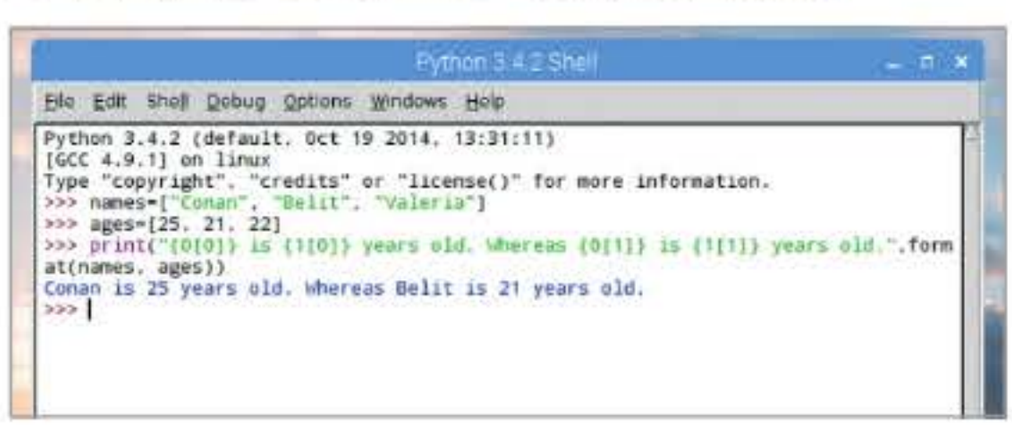


STEP 10 You can also call upon a pair of lists and reference them individually within the same print function. Looking back the code from Step 7, you can alter it with:

```
names=["Conan", "Belit", "Valeria"]
ages=[25, 21, 22]
```

Creating two lists. Now you can call each list, and individual items:

```
print("{0[0]} is {1[0]} years old. Whereas {0[1]}
is {1[1]} years old.".format(names, ages))
```



Date and Time

When working with data it's often handy to have access to the time. For example, you may want to time-stamp an entry or see at what time a user logged into the system and for how long. Luckily acquiring the date and time is easy, thanks to the Time module.

TIME LORDS

The time module contains functions that help you retrieve the current system time, reads the date from strings, formats the time and date and much more.

STEP 1 First you need to import the time module. It's one that's built-in to Python 3 so you shouldn't need to drop into a command prompt and pip install it. Once it's imported, you can call the current time and date with a simple command:

```
import time
time.asctime()
```

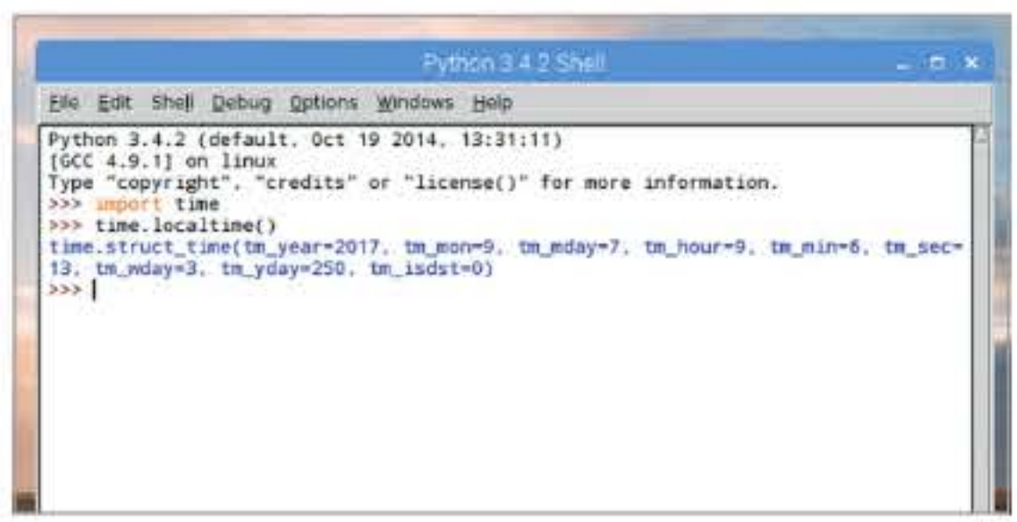
STEP 2 The time function is split into nine tuples, these are divided up into indexed items, as with any other tuple, and shown in the screen shot below.

Index	Field	Values
0	4-digit year	2016
1	Month	1 to 12
2	Day	1 to 31
3	Hour	0 to 23
4	Minute	0 to 59
5	Second	0 to 61 (60 or 61 are leap-seconds)
6	Day of Week	0 to 6 (0 is Monday)
7	Day of year	1 to 366 (Julian day)
8	Daylight savings	-1, 0, 1, -1 means library determines DST

STEP 3 You can see the structure of how time is presented by entering:

```
time.local.time()
```

The output is displayed as such: `'time.struct_time(tm_year=2017, tm_mon=9, tm_mday=7, tm_hour=9, tm_min=6, tm_sec=13, tm_wday=3, tm_yday=250, tm_isdst=0)'`; obviously dependent on your current time as opposed to the time this book was written.

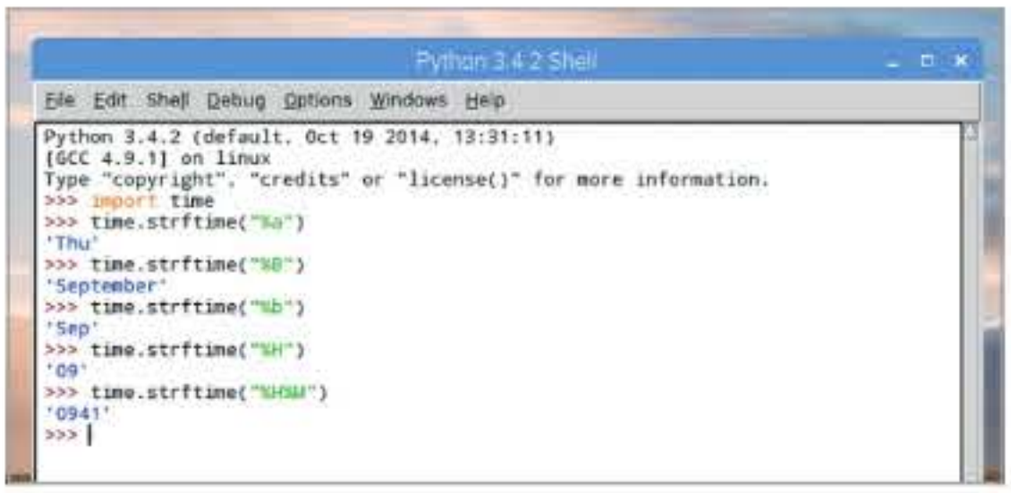


STEP 4 There are numerous functions built into the time module. One of the most common of these is .strftime(). With it, you're able to present a wide range of arguments as it converts the time tuple into a string. For example, to display the current day of the week you can use:

```
time.strftime('%A')
```

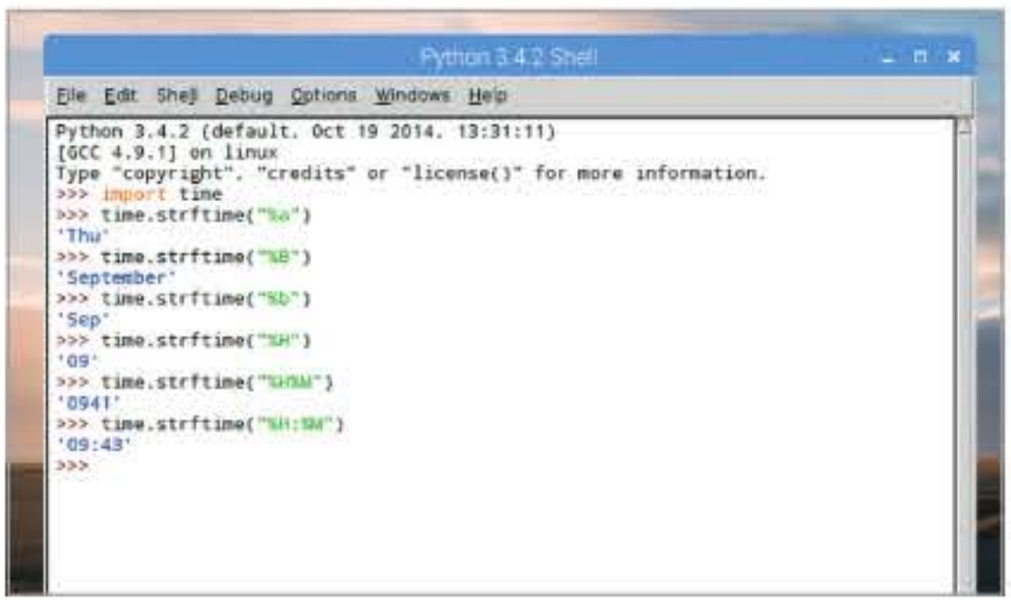
STEP 5 This naturally means you can incorporate various functions into your own code, such as:

```
time.strftime("%a")
time.strftime("%B")
time.strftime("%b")
time.strftime("%H")
time.strftime("%H%M")
```



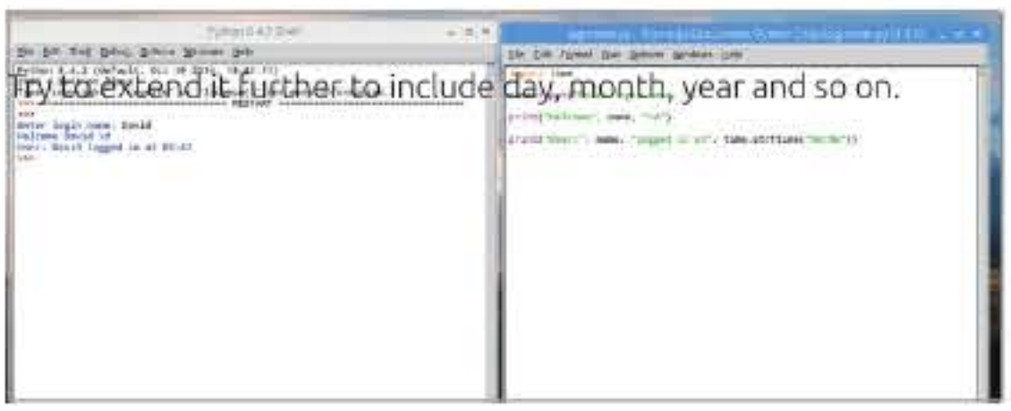
STEP 6 Note the last two entries, with %H and %H%M, as you can see these are the hours and minutes and as the last entry indicates, entering them as %H%M doesn't display the time correctly in the Shell. You can easily rectify this with:

```
time.strftime("%H:%M")
```



STEP 7 This means you're going to be able to display either the current time or the time when something occurred, such as a user entering their name. Try this code in the Editor:

```
import time
name=input("Enter login name: ")
print("Welcome", name, "\d")
print("User:", name, "logged in at", time.
strftime("%H:%M"))
```



Try to extend it further to include day, month, year and so on.

STEP 8 You saw at the end of the previous section, in the code to calculate Pi to however many decimal places the users wanted, you can time a particular event in Python. Take the code from above and alter it slightly by including:

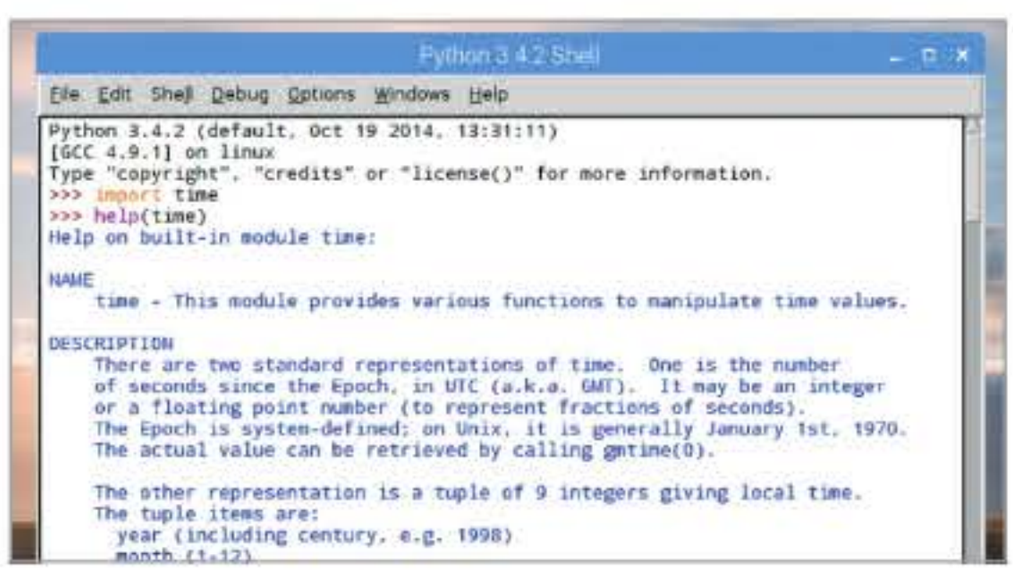
```
start_time=time.time()
```

Then there's:

```
endtime=time.time()-start_time
```

STEP 9 The output will look similar to the screenshot below. The timer function needs to be either side of the input statement, as that's when the variable name is being created, depending on how long the user took to login. The length of time is then displayed on the last line of the code as the `endtime` variable.

STEP 10 There's a lot that can be done with the time module; some of it is quite complex too, such as displaying the number of seconds since January 1st 1970. If you want to drill down further into the time module, then in the Shell enter: `help(time)` to display the current Python version help file for the time module.



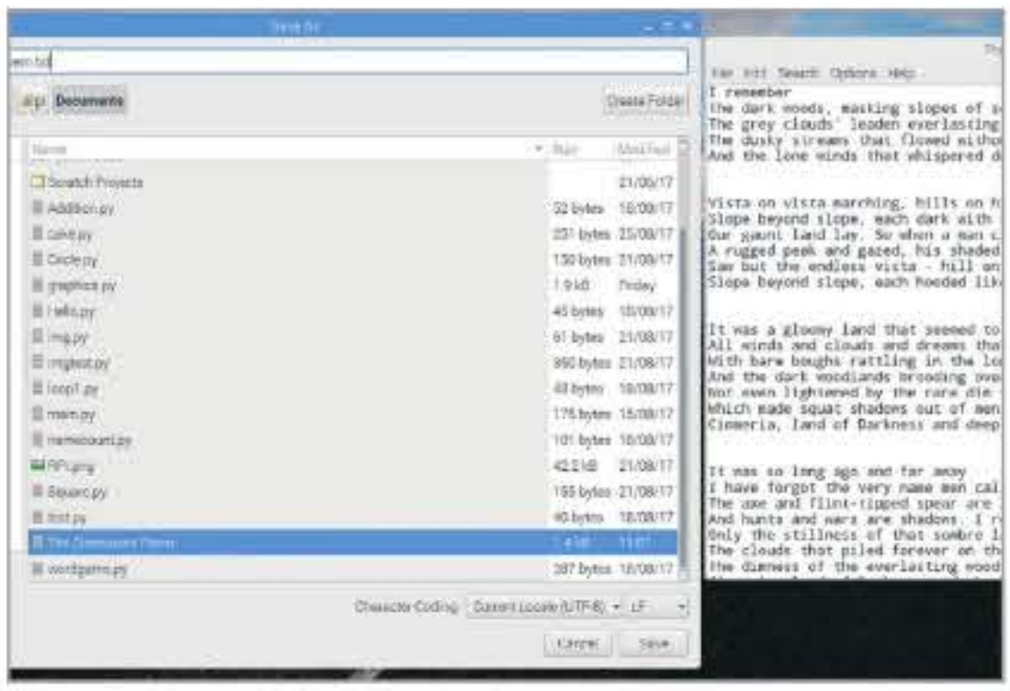
Opening Files

In Python you can read text and binary files in your programs. You can also write to file, which is something we will look at next. Reading and writing to files enables you to output and store data from your programs.

OPEN, READ AND WRITE

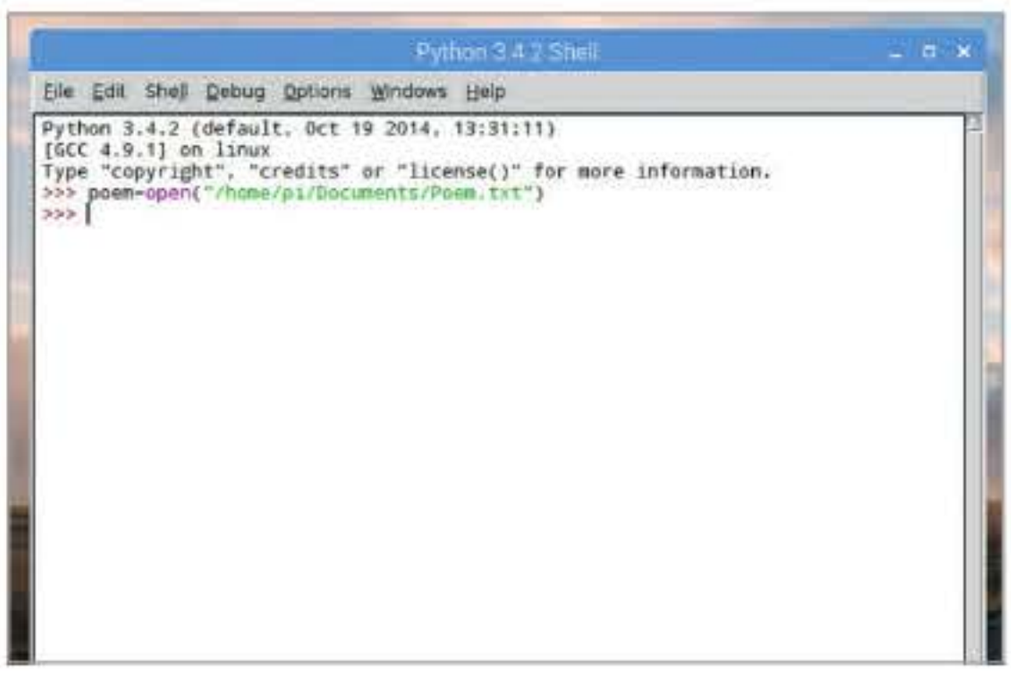
In Python you create a file object, similar to creating a variable, only pass in the file using the open() function. Files are usually categorised as text or binary.

STEP 1 Start by entering some text into your system's text editor. The text editor is best, not a word processor, as word processors include background formatting and other elements. In our example, we have the poem The Cimmerian, by Robert E Howard. You need to save the file as poem.txt.



STEP 2 You use the open() function to pass the file into a variable as an object. You can name the file object anything you like, but you will need to tell Python the name and location of the text file you're opening:

```
poem=open("/home/pi/Documents/Poem.txt")
```



STEP 3 If you now enter poem into the Shell, you will get some information regarding the text file you've just asked to be opened. You can now use the poem variable to read the contents of the file:

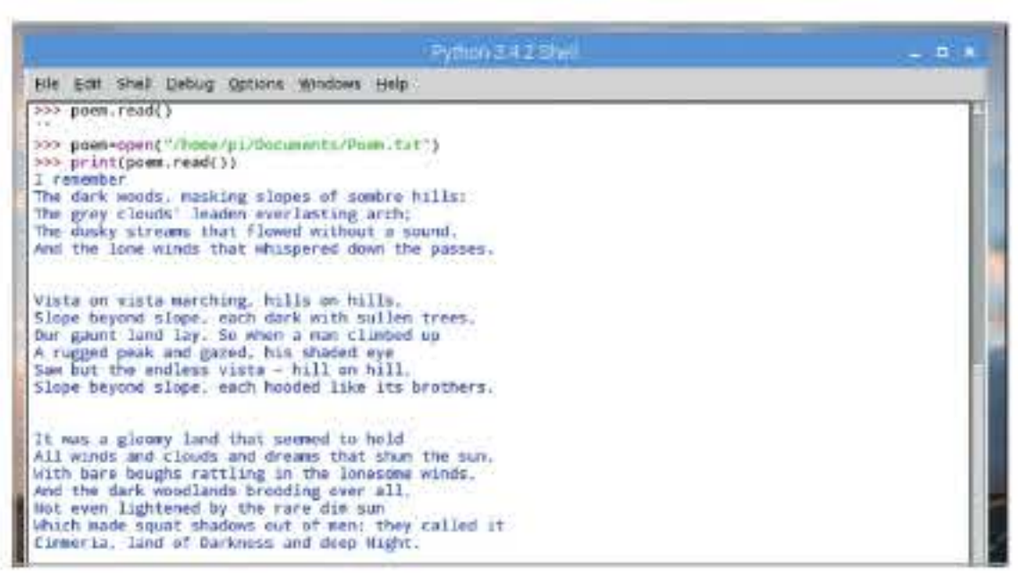
```
poem.read()
```

Note than a /n entry in the text represents a new line, as you used previously.

STEP 4 If you enter poem.read() a second time you will notice that the text has been removed from the file. You will need to enter: `poem=open("/home/pi/Documents/Poem.txt")` again to recreate the file. This time, however, enter:

```
print(poem.read())
```

This time, the /n entries are removed in favour of new lines and readable text.



STEP 5 Just as with lists, tuples, dictionaries and so on, you're able to index individual characters of the text. For example:

```
poem.read(5)
```

Displays the first five characters, whilst again entering:

```
poem.read(5)
```

Will display the next five. Entering (1) will display one character at a time.

STEP 6 Similarly, you can display one line of text at a time by using the readline() function. For example:

```
poem=open("/home/pi/Documents/Poem.txt")
poem.readline()
```

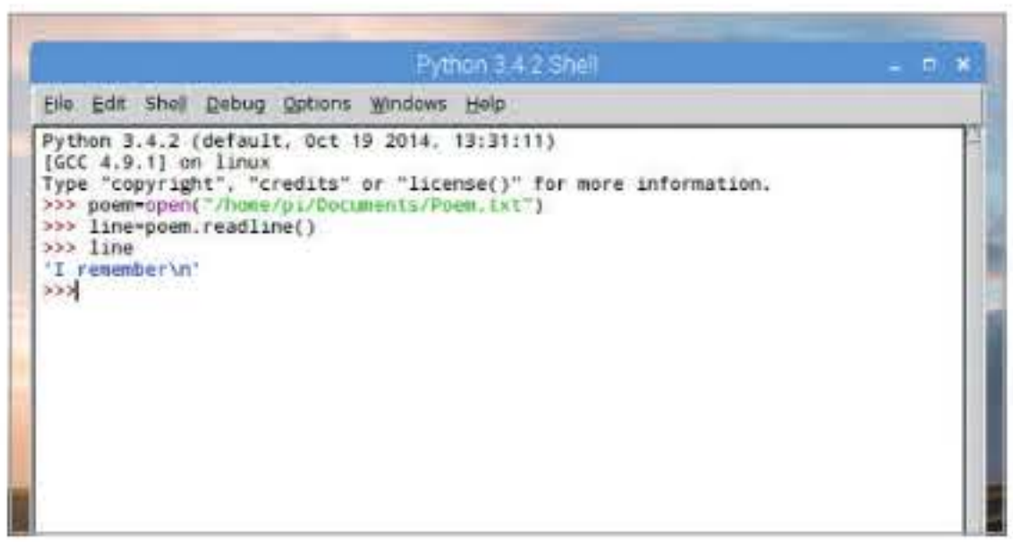
Will display the first line of the text with:

```
poem.readline()
```

Displaying the next line of text once more.

STEP 7 You may have guessed that you can pass the readline() function into a variable, thus allowing you to call it again when needed:

```
poem=open("/home/pi/Documents/Poem.txt")
line=poem.readline()
line
```



STEP 8 Extending this further, you can use readlines() to grab all the lines of the text and store them as multiple lists. These can then be stored as a variable:

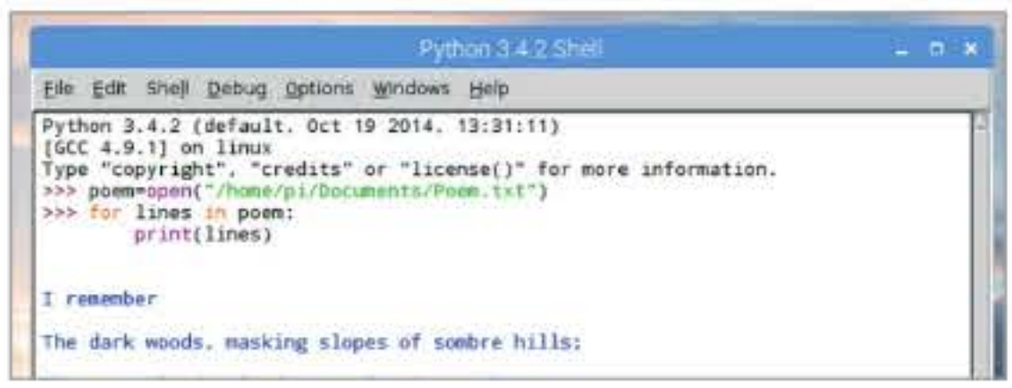
```
poem=open("/home/pi/Documents/Poem.txt")
lines=poem.readlines()
lines[0]
lines[1]
lines[2]
```

STEP 9 You can also use the for statement to read the lines of text back to us:

```
for lines in lines:
  print(lines)
```

Since this is Python, there are other ways to produce the same output:

```
poem=open("/home/pi/Documents/Poem.txt")
for lines in poem:
  print(lines)
```



STEP 10 Let's imagine that you want to print the text one character at a time, like an old dot matrix printer would. You can use the time module mixed with what you've looked at here. Try this:

```
import time
poem=open("/home/pi/Documents/Poem.txt")
lines=poem.read()
for lines in lines:
  print(lines, end="")
  time.sleep(.15)
```

The output is fun to view, and easily incorporated into your own code.

Writing to Files

The ability to read external files within Python is certainly handy but writing to a file is better still. Using the write() function, you're able to output the results of a program to a file, that you can then read() back into Python.

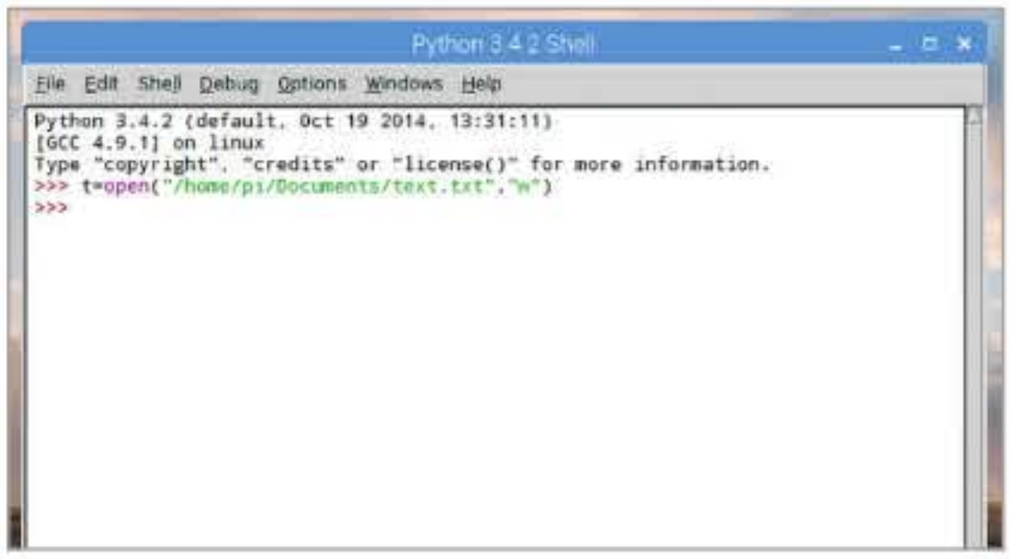
WRITE AND CLOSE

The write() function is slightly more complex than read(). Along with the filename you must also include an access mode which determines whether the file in question is in read or write mode.

STEP 1 Start by opening IDLE and enter the following:

```
t=open("/home/pi/Documents/text.txt","w")
```

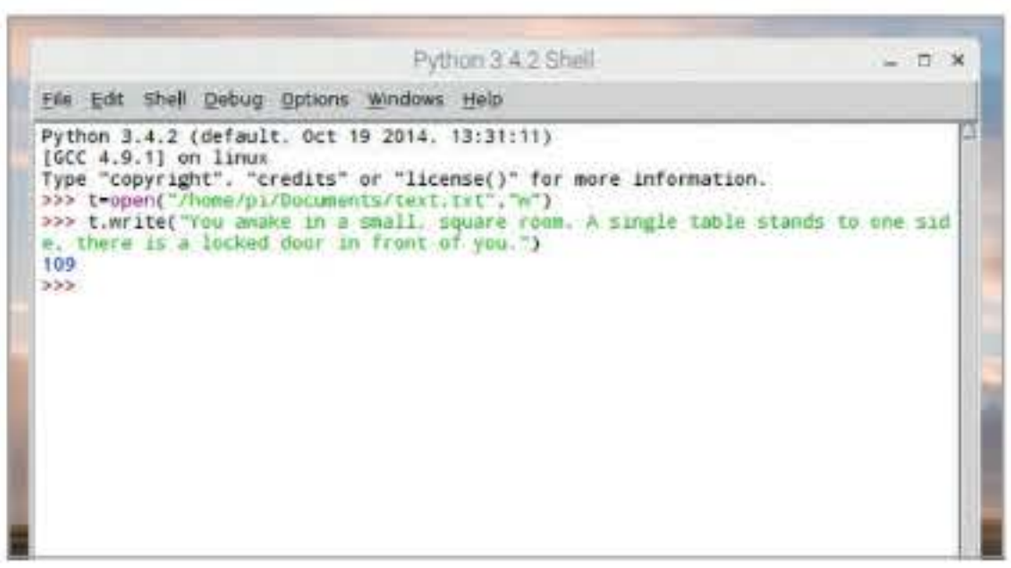
Change the destination from /home/pi/Documents to your own system. This code will create a text file called text.txt in write mode using the variable 't'. If there's no file of that name in the location, it will create one. If one already exits, it will overwrite it, so be careful.



STEP 2 You can now write to the text file using the write() function. This works opposite to read(), writing lines instead of reading them. Try this:

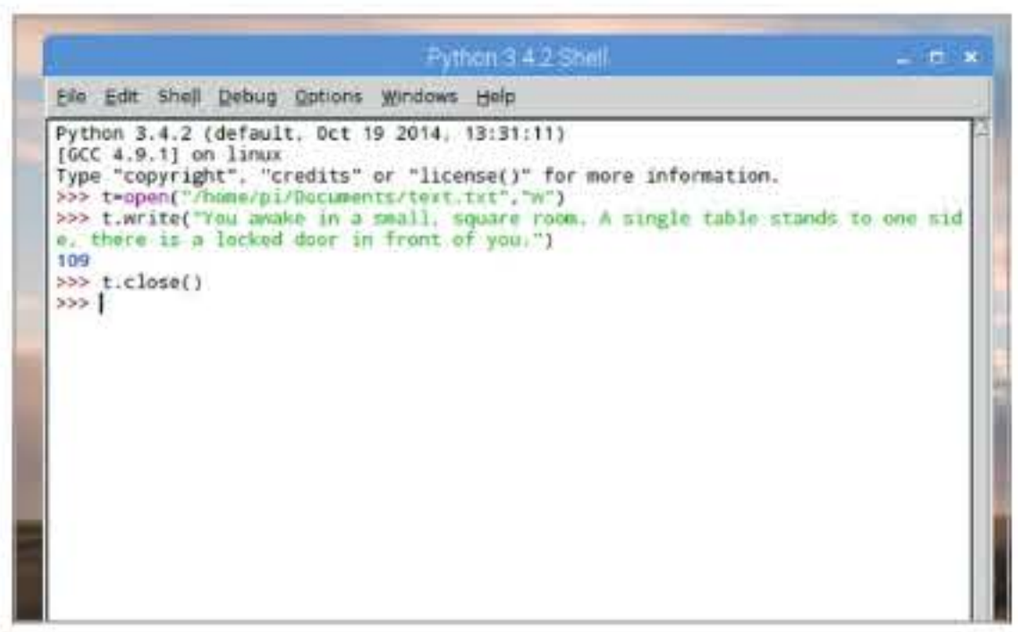
```
t.write("You awake in a small, square room. A single table stands to one side, there is a locked door in front of you.")
```

Note the 109. It's the number of characters you've entered.

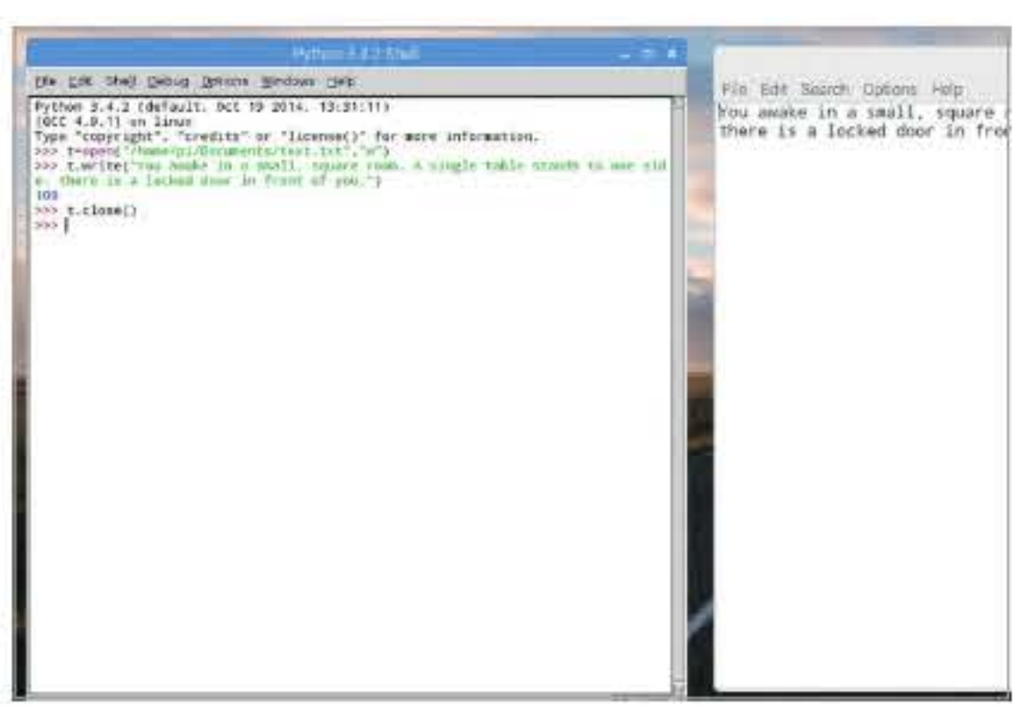


STEP 3 However, the actual text file is still blank (you can check by opening it up). This is because you've written the line of text to the file object but not committed it to the file itself. Part of the write() function is that you need to commit the changes to the file; you can do this by entering:

```
t.close()
```

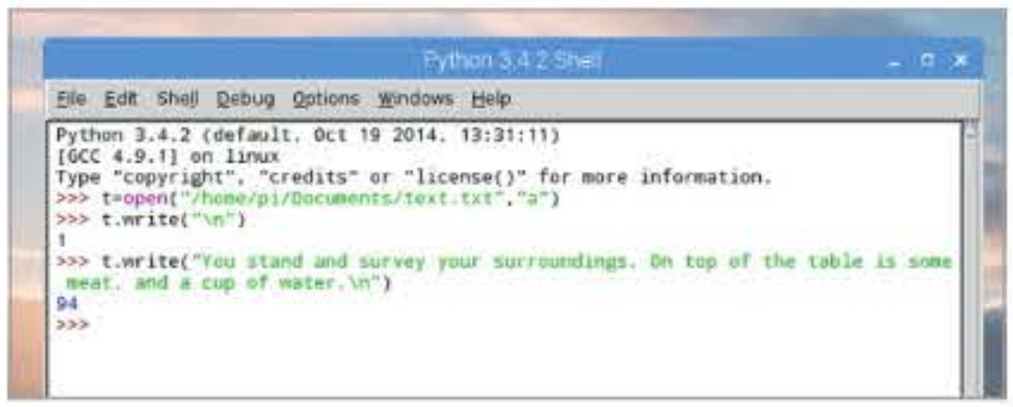


STEP 4 If you now open the text file with a text editor, you can see that the line you created has been written to the file. This gives us the foundation for some interesting possibilities: perhaps the creation of your own log file or even the beginning of an adventure game.



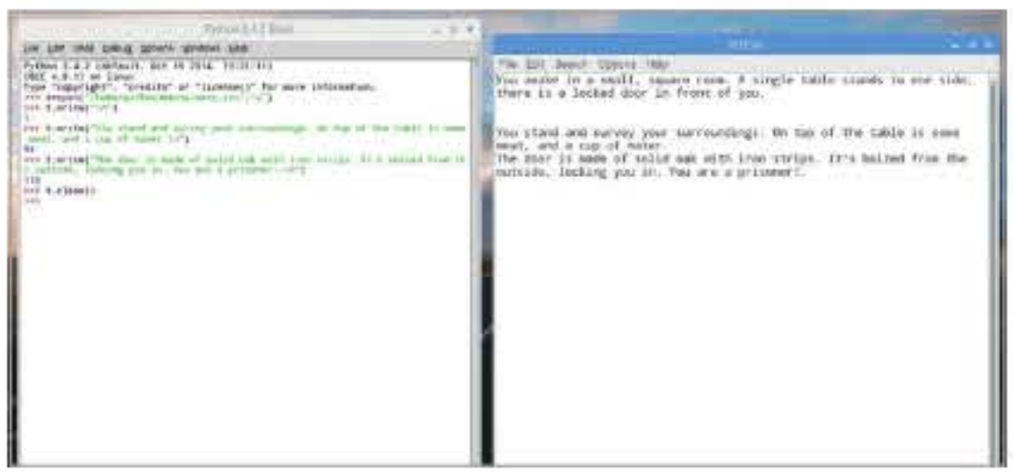
STEP 5 To expand this code, you can reopen the file using 'a', for access or append mode. This will add any text at the end of the original line instead of wiping the file and creating a new one. For example:

```
t=open("/home/pi/Documents/text.txt","a")
t.write("\n")
t.write(" You stand and survey your surroundings.
On top of the table is some meat, and a cup of
water.\n")
```

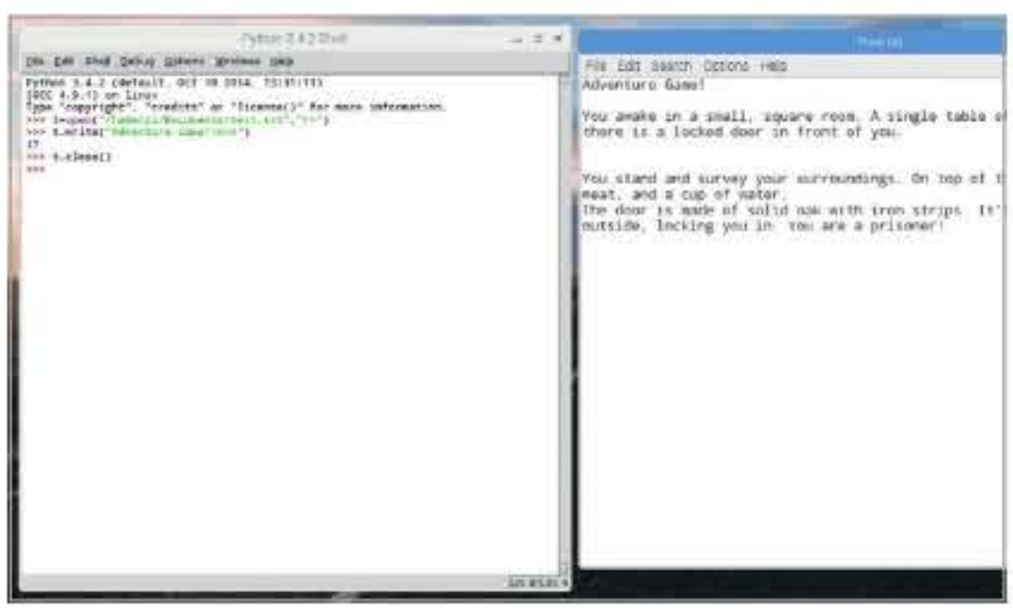


STEP 6 You can keep extending the text line by line, ending each with a new line (\n). When you're done, finish the code with t.close() and open the file in a text editor to see the results:

```
t.write("The door is made of solid oak with iron
strips. It's bolted from the outside, locking you
in. You are a prisoner!.\n")
t.close()
```

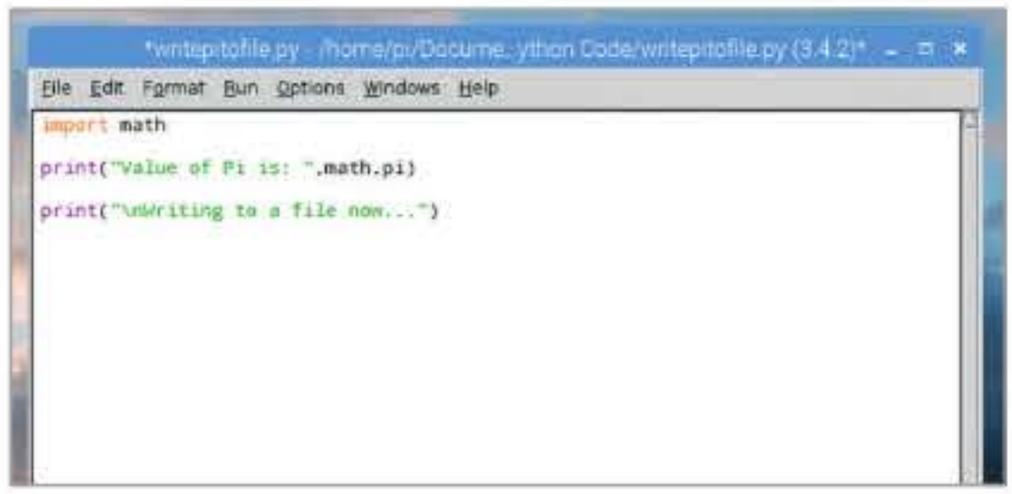


STEP 7 There are various types of file access to consider using the open() function. Each depends on how the file is accessed and even the position of the cursor. For example, r+ opens a file in read and write and places the cursor at the start of the file.



STEP 8 You can pass variables to a file that you've created in Python. Perhaps you want the value of Pi to be written to a file. You can call Pi from the math module, create a new file and pass the output of Pi into the new file:

```
import math
print("Value of Pi is: ",math.pi)
print("\nWriting to a file now…")
```



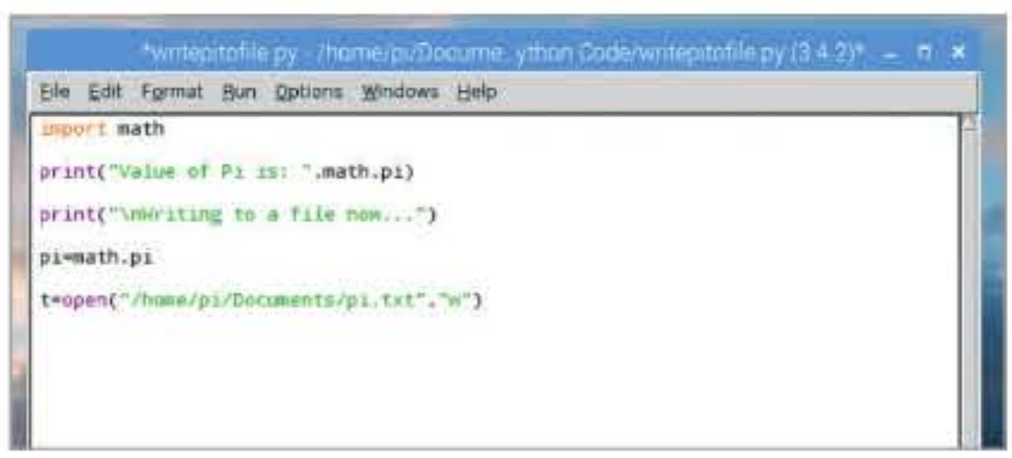
STEP 9 Now let's create a variable called pi and assign it the value of Pi:

```
pi=math.pi
```

You also need to create a new file in which to write Pi to:

```
t=open("/home/pi/Documents/pi.txt","w")
```

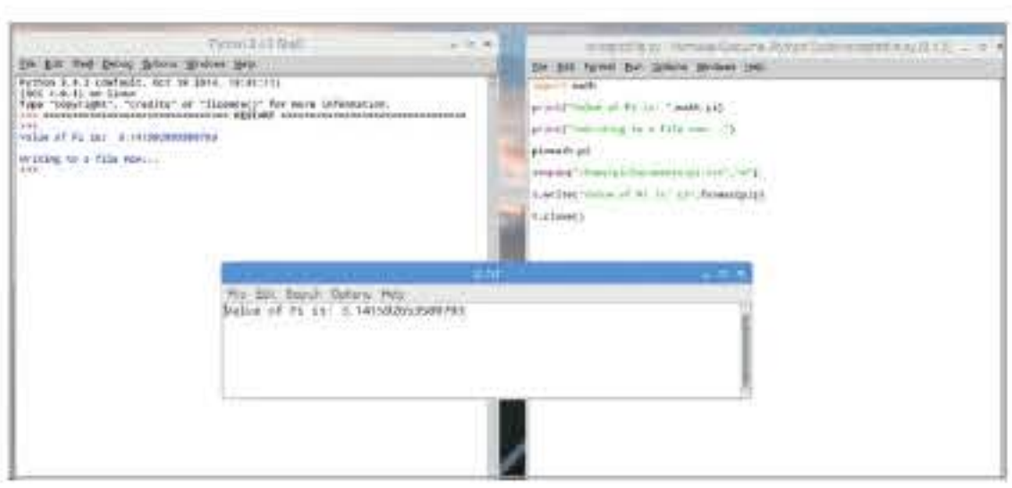
Remember to change your file location to your own particular system setup.



STEP 10 To finish, you can use string formatting to call the variable and write it to the file, then commit the changes and close the file:

```
t.write("Value of Pi is: {}".format(pi))
t.close()
```

You can see from the results that you're able to pass any variable to a file.



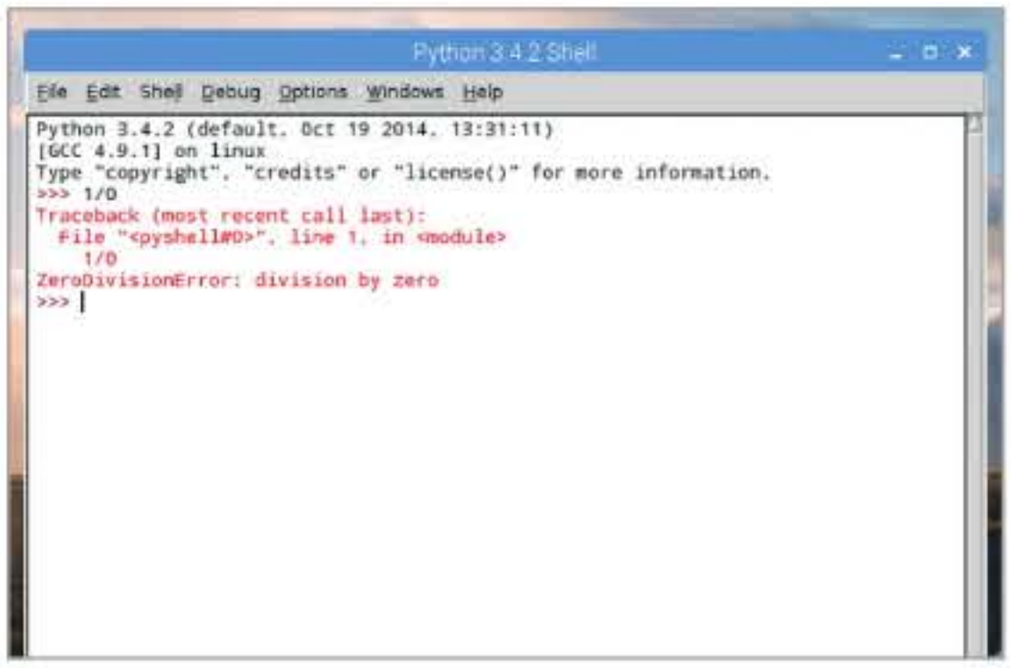
Exceptions

When coding, you'll naturally come across some issues that are out of your control. Let's assume you ask a user to divide two numbers and they try to divide by zero. This will create an error and break your code.

EXCEPTIONAL OBJECTS

Rather than stop the flow of your code, Python includes exception objects which handle unexpected errors in the code. You can combat errors by creating conditions where exceptions may occur.

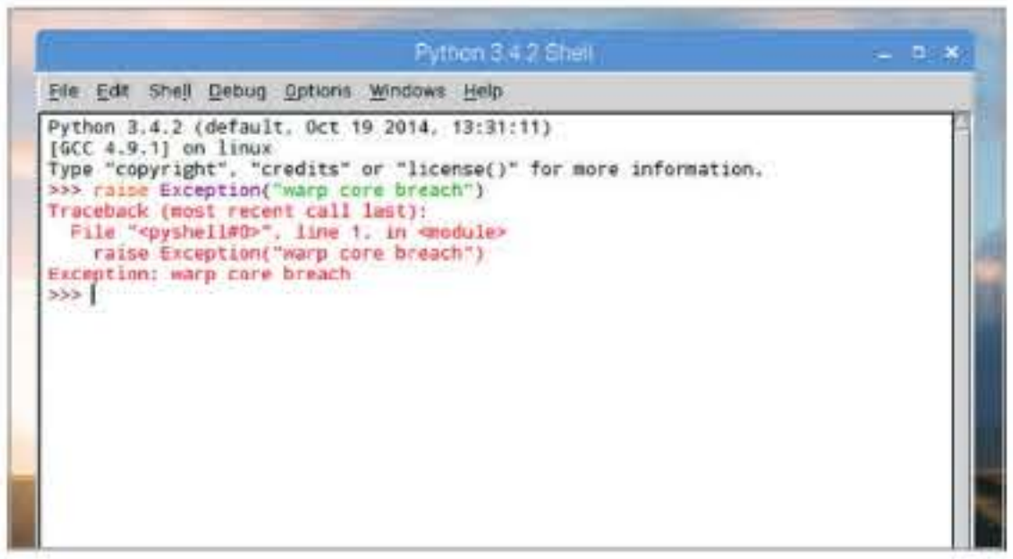
STEP 1 You can create an exception error by simply trying to divide a number by zero. This will report back with the ZeroDivisionError: Division by zero message, as seen in the screenshot. The ZeroDivisionError part is the exception class, of which there are many.



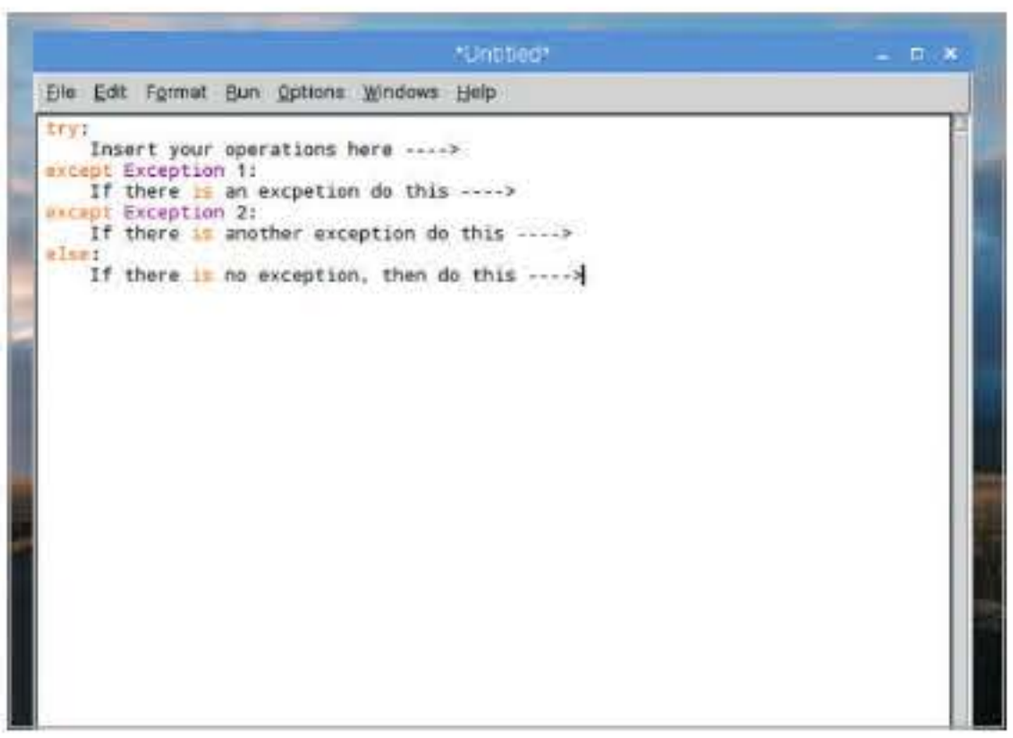
STEP 2 Most exceptions are raised automatically when Python comes across something that's inherently wrong with the code. However, you can create your own exceptions that are designed to contain the potential error and react to it, as opposed to letting the code fail.

STEP 3 You can use the functions raise exception to create our own error handling code within Python. Let's assume your code has you warping around the cosmos, too much however results in a warp core breach. To stop the game from exiting due to the warp core going supernova, you can create a custom exception:

```
raise Exception("warp core breach")
```

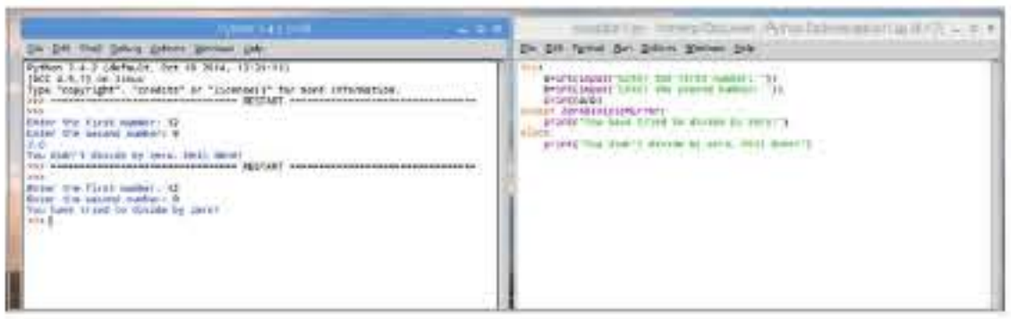


STEP 4 To trap any errors in the code you can encase the potential error within a try: block. This block consists of try, except, else, where the code is held within try:, then if there's an exception do something, else do something else.



STEP 5 For example, use the divide by zero error. You can create an exception where the code can handle the error without Python quitting due to the problem:

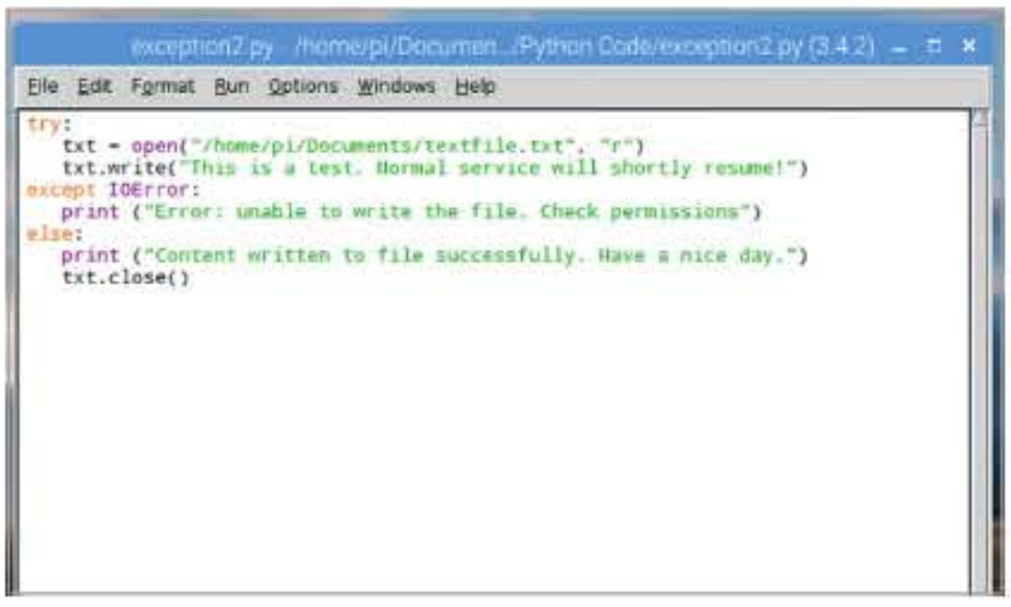
```
try:
  a=int(input("Enter the first number: "))
  b=int(input("Enter the second number: "))
  print(a/b)
except ZeroDivisionError:
  print("You have tried to divide by zero!")
else:
  print("You didn't divide by zero. Well done!")
```



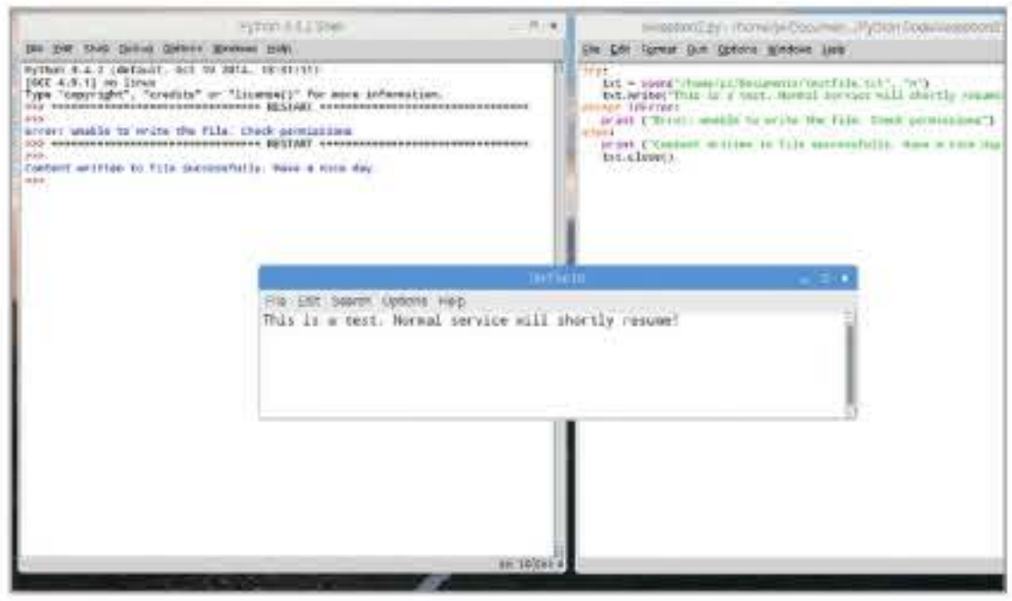
STEP 6 You can use exceptions to handle a variety of useful tasks. Using an example from our previous tutorials, let's assume you want to open a file and write to it:

```
try:
  txt = open("/home/pi/Documents/textfile.txt", "r")
  txt.write("This is a test. Normal service will
shortly resume!")
except IOError:
  print ("Error: unable to write the file. Check
permissions")
else:
  print ("Content written to file successfully. Have
a nice day.")
  txt.close()
```

STEP 7 Obviously this won't work due to the file textfile.txt being opened as read only (the "r" part). So in this case rather than Python telling you that you're doing something wrong, you've created an exception using the IOError class informing the user that the permissions are incorrect.



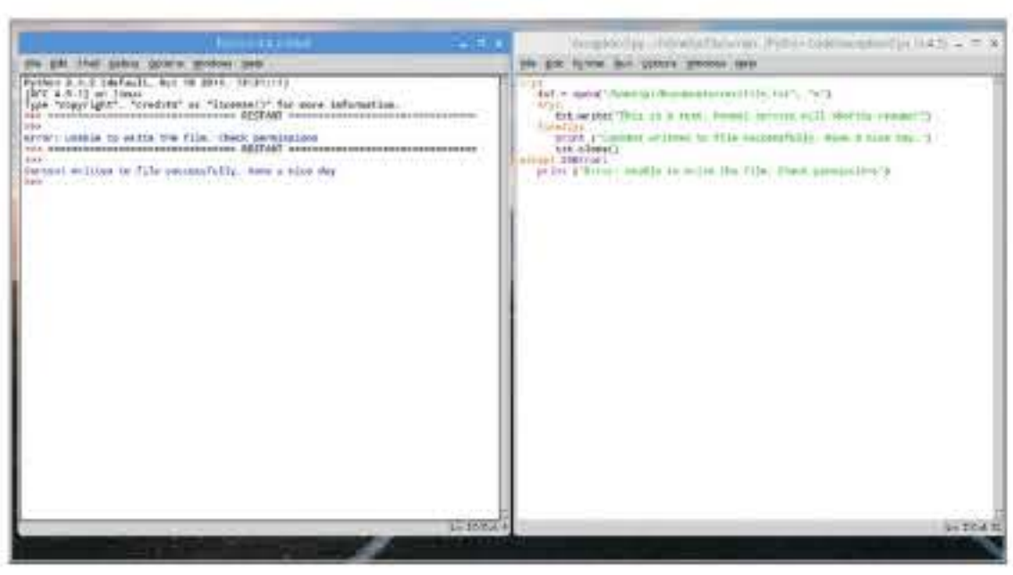
STEP 8 Naturally, you can quickly fix the issue by changing the "r" read only instance with a "w" for write. This, as you already know, will create the file and write the content then commit the changes to the file. The end result will report a different set of circumstances, in this case, a successful execution of the code.



STEP 9 You can also use a finally: block, which works in a similar fashion but you can't use else with it. To use our example from Step 6:

```
try:
  txt = open("/home/pi/Documents/textfile.txt", "r")
  try:
   txt.write("This is a test. Normal service will
shortly resume!")
  finally:
   print ("Content written to file successfully.
Have a nice day.")
   txt.close()
except IOError:
  print ("Error: unable to write the file. Check
permissions")
```

STEP 10 As before an error will occur as you've used the "r" read-only permission. If you change it to a "w", then the code will execute without the error being displayed in the IDLE Shell. Needless to say, it can be a tricky getting the exception code right the first time. Practise though, and you will get the hang of it.



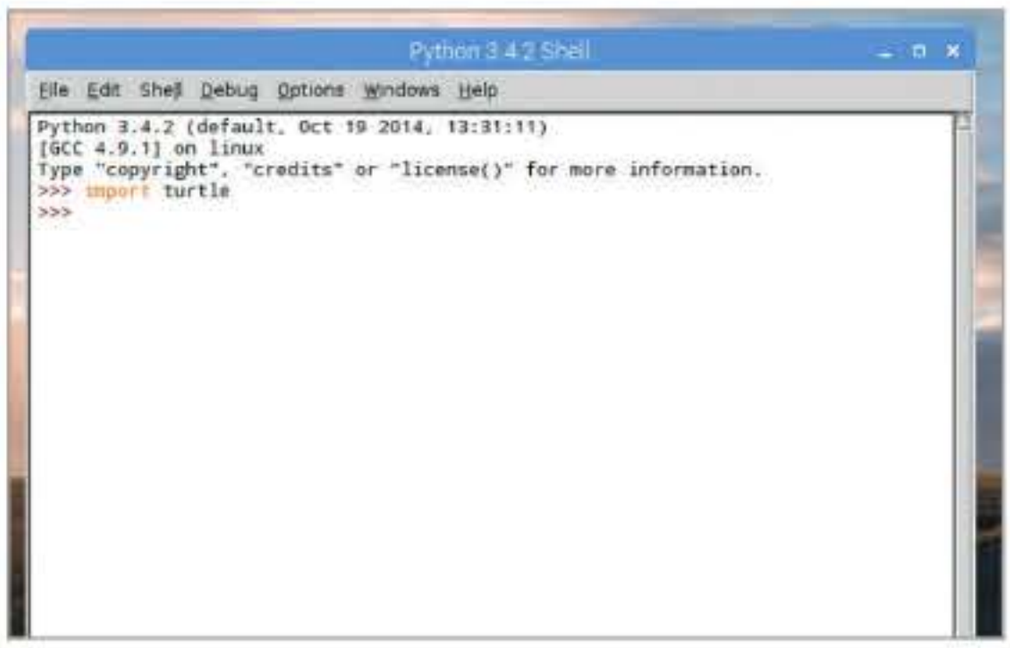
Python Graphics

Whilst dealing with text on the screen, either as a game or in a program, is great, there will come a time when a bit of graphical representation wouldn't go amiss. Python 3 has numerous ways in which to include graphics and they're surprisingly powerful too.

GOING GRAPHICAL

You can draw simple graphics, lines, squares and so on, or you can use one of the many Python modules available, to bring out some spectacular effects.

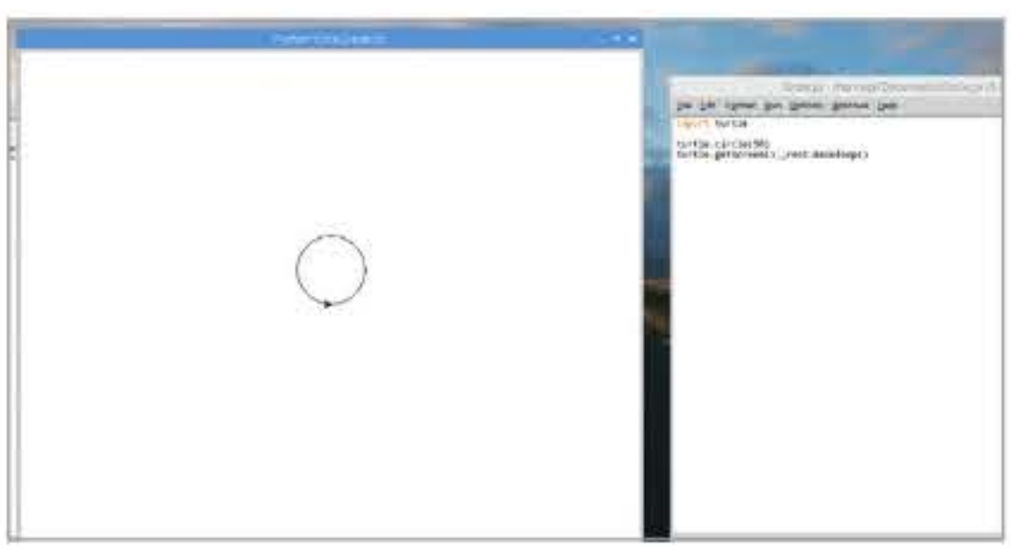
STEP 1 One of the best graphical modules to begin learning Python graphics is Turtle. The Turtle module is, as the name suggests, based on the turtle robots used in many schools, that can be programmed to draw something on a large piece of paper on the floor. The Turtle module can be imported with: `import turtle`.



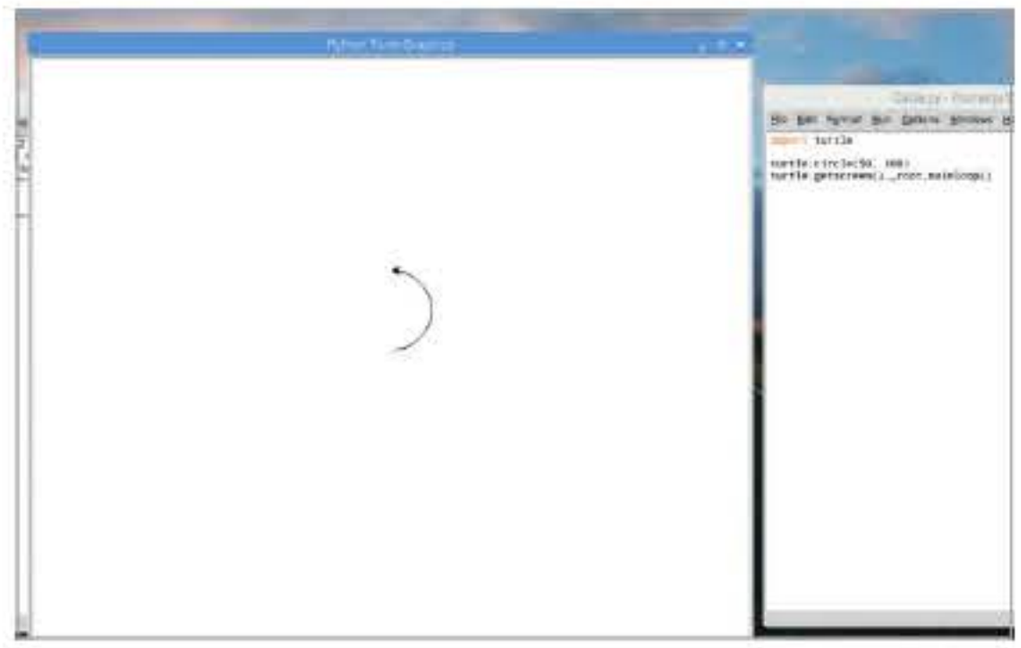
STEP 2 Let's begin by drawing a simple circle. Start a New File, then enter the following code:

```
import turtle

turtle.circle(50)
turtle.getscreen()._root.mainloop()
```

As usual press F5 to save the code and execute it. A new window will now open up and the 'Turtle' will draw a circle.



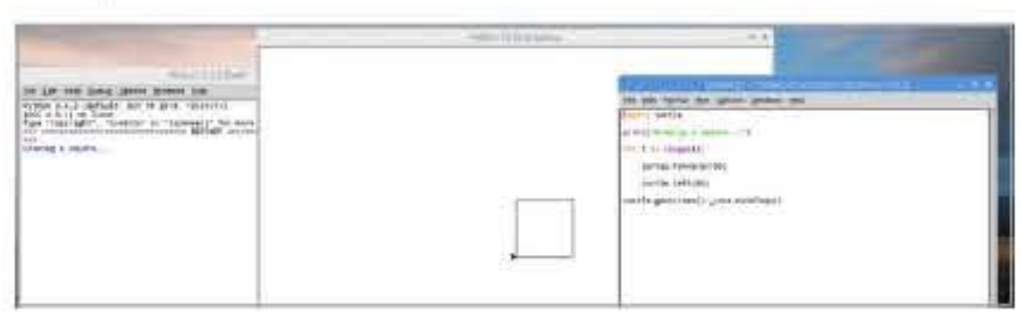
STEP 3 The command turtle.circle(50) is what draws the circle on the screen, with 50 being the size. You can play around with the sizes if you like, going up to 100, 150 and beyond; you can draw an arc by entering: `turtle.circle(50, 180)`, where the size is 50, but you're telling Python to only draw 180° of the circle.



STEP 4 The last part of the circle code tells Python to keep the window where the drawing is taking place to remain open, so the user can click to close it. Now, let's make a square:

```
import turtle

print("Drawing a square…")

for t in range(4):
   turtle.forward(100)
   turtle.left(90)
turtle.getscreen()._root.mainloop()
```

You can see that we've inserted a loop to draw the sides of the square.



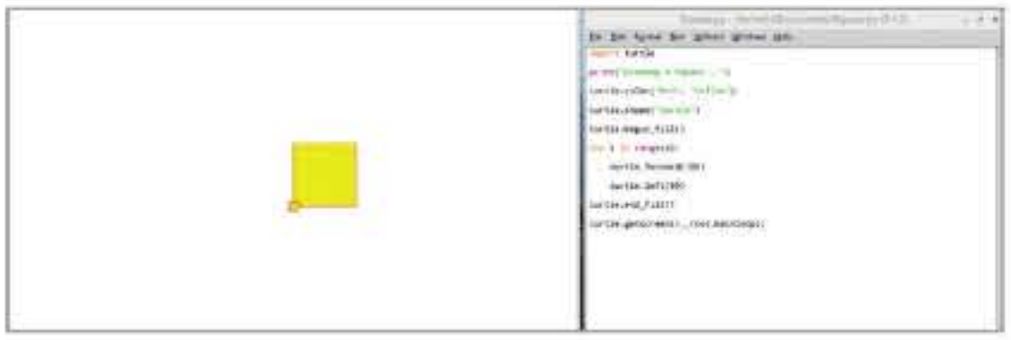
STEP 5 You can add a new line to the square code to add some colour:

```
turtle.color("Red")
```

Then you can even change the character to an actual turtle by entering:

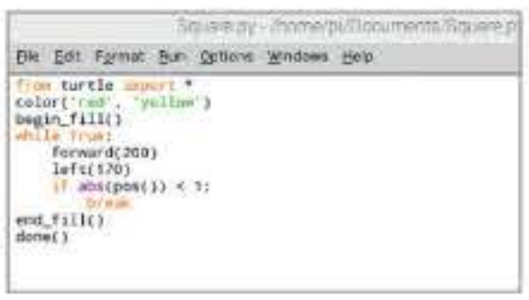
```
turtle.shape("turtle")
```

You can also use the command `turtle.begin_fill()`, and `turtle.end_fill()` to fill in the square with the chosen colours; red outline, yellow fill in this case.



STEP 6 You can see that the Turtle module can draw out some pretty good shapes and become a little more complex as you begin to master the way it works. Enter this example:

```
from turtle import *
color('red', 'yellow')
begin_fill()
while True:
  forward(200)
  left(170)
  if abs(pos()) < 1:
    break
end_fill()
done()
```

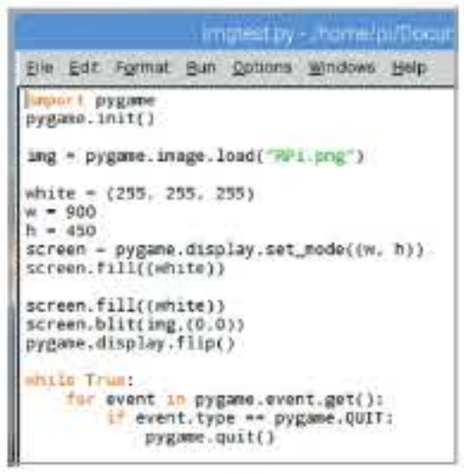


It's a different method, but very effective.

STEP 7 Another way in which you can display graphics is by using the Pygame module. There are numerous ways in which pygame can help you output graphics to the screen but for now let's look at displaying a predefined image. Start by opening a browser and finding an image, then save it to the folder where you save your Python code.

STEP 8 Now let's get the code by importing the pygame module:

```
import pygame
pygame.init()

img = pygame.image.load("RPi.png")

white = (255, 255, 255)
w = 900
h = 450
screen = pygame.display.
set_mode((w, h))
screen.fill((white))

screen.fill((white))
screen.blit(img,(0,0))
pygame.display.flip()

while True:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      pygame.quit()
```



STEP 9 In the previous step you imported pygame, initiated the pygame engine and asked it to import our saved Raspberry Pi logo image, saved as RPi.png. Next you defined the background colour of the window to display the image and the window size as per the actual image dimensions. Finally you have a loop to close the window.

```
w = 900
h = 450
screen = pygame.display.set_mode((w, h))
screen.fill((white))

screen.fill((white))
screen.blit(img,(0,0))
pygame.display.flip()

while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
```

STEP 10 Press F5 to save and execute the code and your image will be displayed in a new window. Have a play around with the colours, sizes and so on and take time to look up the many functions within the pygame module too.

Combining What You Know So Far

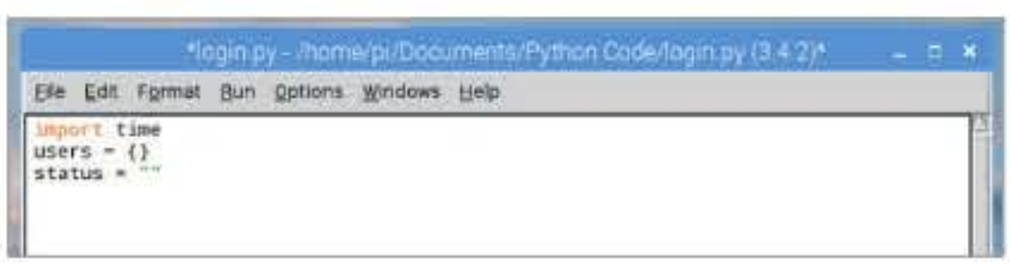
Based on what you've looked at over this section, let's combine it all and come up with a piece of code that can easily be applied into a real-world situation; or at the very least, something which you can incorporate into your programs.

LOGGING IN

For this example, let's look to a piece of code that creates user logins and then allows them to log into the system and write the time they logged in at. You can even include an option to quit the program by pressing 'q'.

STEP 1 Begin by importing the time module, creating a new dictionary to handle the usernames and passwords and creating a variable to evaluate the current status of the program:

```
import time
users = {}
status = ""
```



STEP 2 Next you need to define some functions. You can begin by creating the main menu, where all users will return to after selecting the available options:

```
def mainMenu():
  global status
  status = input("Do you have a login account?
y/n? Or press q to quit.")
  if status == "y":
    oldUser()
  elif status == "n":
    newUser()
  elif status == "q":
    quit()
```

STEP 3 The global status statement separates a local variable from one that can be called throughout the code, this way you can use the q=quit element without it being changed inside the function. We've also referenced some newly defined functions: oldUser and newUser which we'll get to next.

```
def mainMenu():
    global status
    status = input("Do you have a login account? y/n? Or press q to quit.")
    if status == "y":
        oldUser()
    elif status == "n":
        newUser()
    elif status == "q":
        quit()
```

STEP 4 The newUser function is next:

```
def newUser():
  createLogin = input("Create a login name: ")
  if createLogin in users:
    print ("\nLogin name already exists!\n")
  else:
    createPassw = input("Create password: ")
    users[createLogin] = createPassw
    print("\nUser created!\n")
    logins=open("/home/pi/Documents/logins.
txt", "a")
    logins.write("\n" + createLogin + " " +
createPassw)
    logins.close()
```

This creates a new user and password, and writes the entries into a file called logins.txt.

STEP 5 You will need to specify your own location for the logins.txt file, since we're using a Raspberry Pi. Essentially, this adds the username and password inputs from the user to the existing users{} dictionary, so the key and value structure remains: each user is the key, the password is the value.

```
def newUser():
    createLogin = input("Create a login name: ")

    if createLogin in users:
        print ("\nLogin name already exists!\n")
    else:
        createPassw = input("Create password: ")
        users[createLogin] = createPassw
        print("\nUser created!\n")
        logins=open("/home/pi/Documents/logins.txt", "a")
        logins.write("\n" + createLogin + " " + createPassw)
        logins.close()
```

STEP 6 Now to create the oldUser function:

```
def oldUser():
  login = input("Enter login name: ")
  passw = input("Enter password: ")

  # check if user exists and login
matches password
  if login in users and users[login] == passw:
    print ("\nLogin successful!\n")
    print ("User:", login, "accessed the system
on:", time.asctime())
  else:
    print ("\nUser doesn't exist or wrong
password!\n")
```

```
def mainMenu():
    global status
    status = input("Do you have a login account? y/n? Or press q to quit.")
    if status == "y":
        oldUser()
    elif status == "n":
        newUser()
    elif status == "q":
        quit()

def newUser():
    createLogin = input("Create a login name: ")

    if createLogin in users:
        print ("\nLogin name already exists!\n")
    else:
        createPassw = input("Create password: ")
        users[createLogin] = createPassw
        print("\nUser created!\n")
        logins=open("/home/pi/Documents/logins.txt", "a")
        logins.write("\n" + createLogin + " " + createPassw)
        logins.close()

def oldUser():
    login = input("Enter login name: ")
    passw = input("Enter password: ")

    # check if user exists and login matches password
    if login in users and users[login] == passw:
        print ("\nLogin successful!\n")
        print ("User:", login, "accessed the system on:", time.asctime())
    else:
        print ("\nUser doesn't exist or wrong password!\n")
```

STEP 7 There's a fair bit happening here. There are login and passw variables, which are then matched to the users dictionary. If there's a match, then you have a successful login and the time and date of the login is outputted. If they don't match, then you print an error and the process starts again.

```
def oldUser():
    login = input("Enter login name: ")
    passw = input("Enter password: ")

    # check if user exists and login matches password
    if login in users and users[login] == passw:
        print ("\nLogin successful!\n")
        print ("User:", login, "accessed the system on:", time.asctime())
    else:
        print ("\nUser doesn't exist or wrong password!\n")
```

STEP 8 Finally, you need to continually check that the 'q' key hasn't been pressed to exit the program. We can do this with:

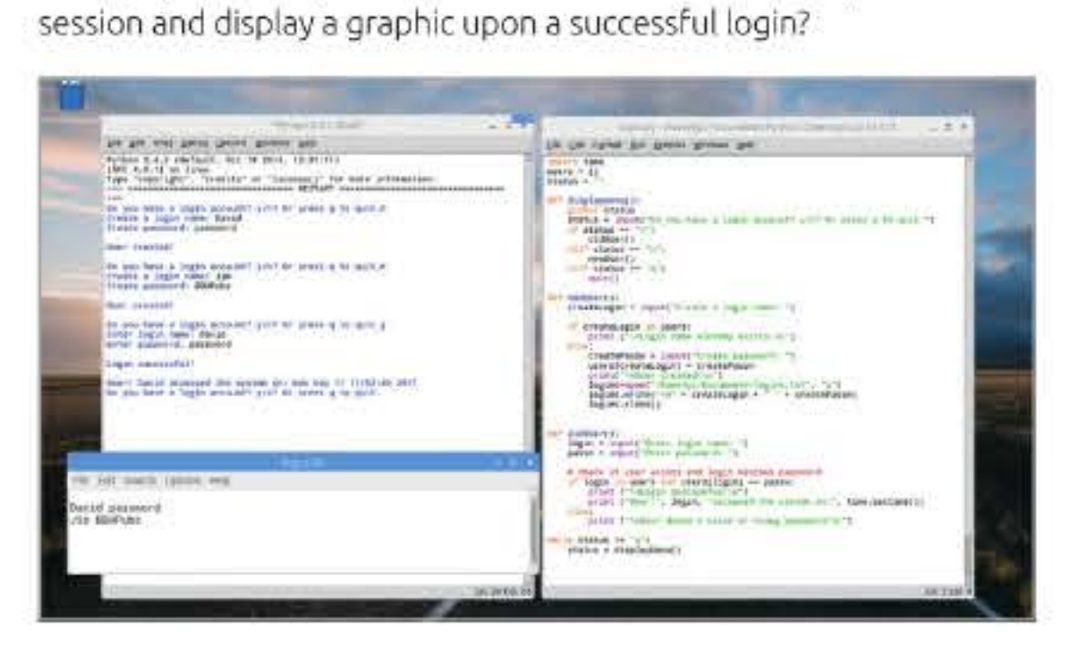
```
while status != "q":
  status = displayMenu()
```

```
*login.py - /home/pi/Documents/Python Code/login.py (3.4.2)*
File Edit Format Run Options Windows Help
import time
users = {}
status = ""

def mainMenu():
    global status
    status = input("Do you have a login account? y/n? Or press q to quit.")
    if status == "y":
        oldUser()
    elif status == "n":
        newUser()
    elif status == "q":
        quit()

def newUser():
    createLogin = input("Create a login name: ")

    if createLogin in users:
        print ("\nLogin name already exists!\n")
    else:
        createPassw = input("Create password: ")
        users[createLogin] = createPassw
        print("\nUser created!\n")
        logins=open("/home/pi/Documents/logins.txt", "a")
        logins.write("\n" + createLogin + " " + createPassw)
        logins.close()

def oldUser():
    login = input("Enter login name: ")
    passw = input("Enter password: ")

    # check if user exists and login matches password
    if login in users and users[login] == passw:
        print ("\nLogin successful!\n")
        print ("User:", login, "accessed the system on:", time.asctime())
    else:
        print ("\nUser doesn't exist or wrong password!\n")

while status != "q":
    status = displayMenu()
```

STEP 9 Although a seemingly minor two lines, the while loop is what keeps the program running. At the end of every function it's checked against the current value of status. If that global value isn't 'q' then the program continues. If it's equal to 'q' then the program can quit.

```
while status != "q":
    status = displayMenu()
```

STEP 10 You can now create users, then login with their names and passwords, with the logins.txt file being created to store the login data and successful logins being time-stamped. Now it's up to you to further improve the code. Perhaps you can import the list of created users from a previous session and display a graphic upon a successful login?



Using Modules

Modules are where you can take Python programming to new heights. You can create your own modules or you can use some of the already available modules, to help convert a mundane piece of code into something spectacular.

Want to see how to make better use of these modules to add a little something extra to your code? Then read on and learn how they can be used to fashion fantastic programs.

76	Calendar Module
78	OS Module
80	Random Module
82	Tkinter Module
84	Pygame Module
88	Create Your Own Modules

Calendar Module

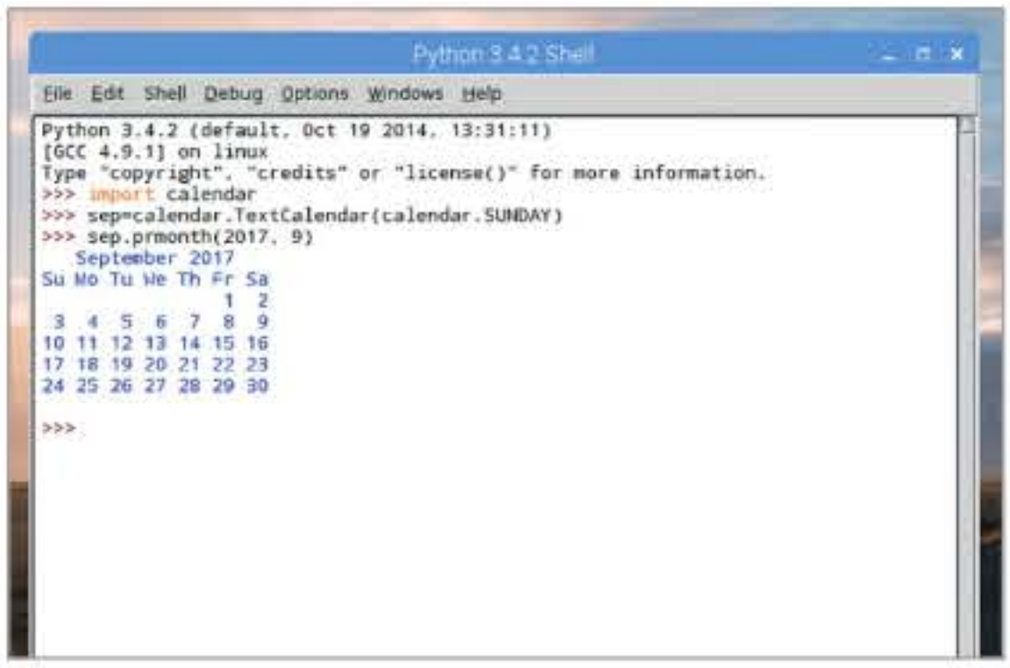
Beyond the time module, the calendar module can produce some interesting results when executed within your code. It does far more than simply display the date in the time module-like format, you can actually call up a wall calendar type display.

WORKING WITH DATES

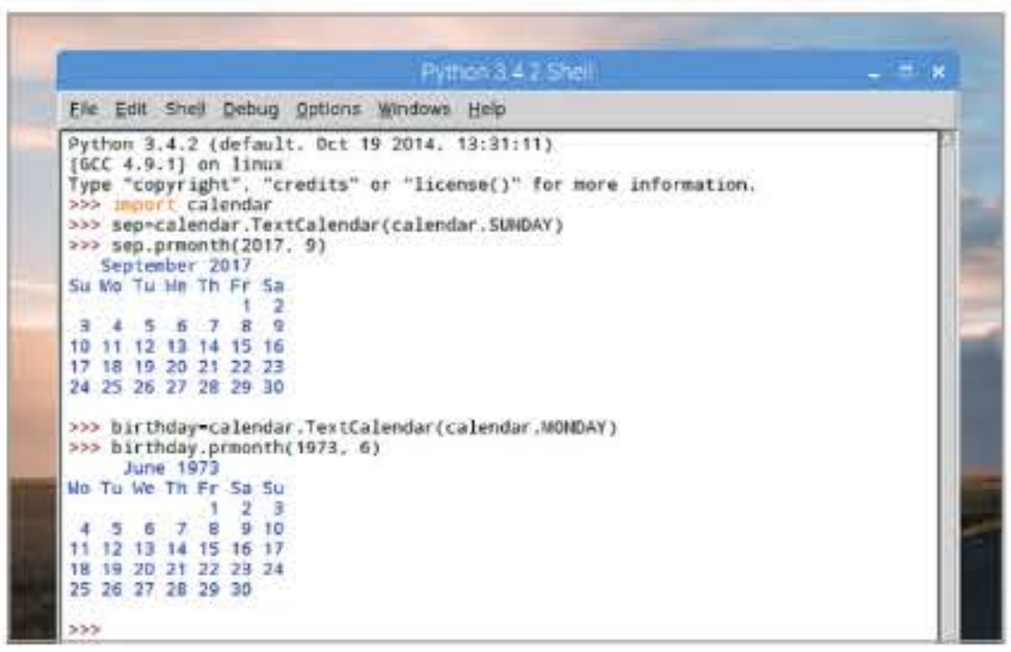
The calendar module is built into Python 3. However, if for some reason it's not installed you can add it using **pip install calendar** as a Windows administrator or **sudo pip install calendar** for Linux and macOS.

STEP 1 Launch Python 3 and enter: `import calendar` to call up the module and its inherent functions. Once it's loaded into memory, start by entering:

```
sep=calendar.TextCalendar(calendar.SUNDAY)
sep.prmonth(2017, 9)
```



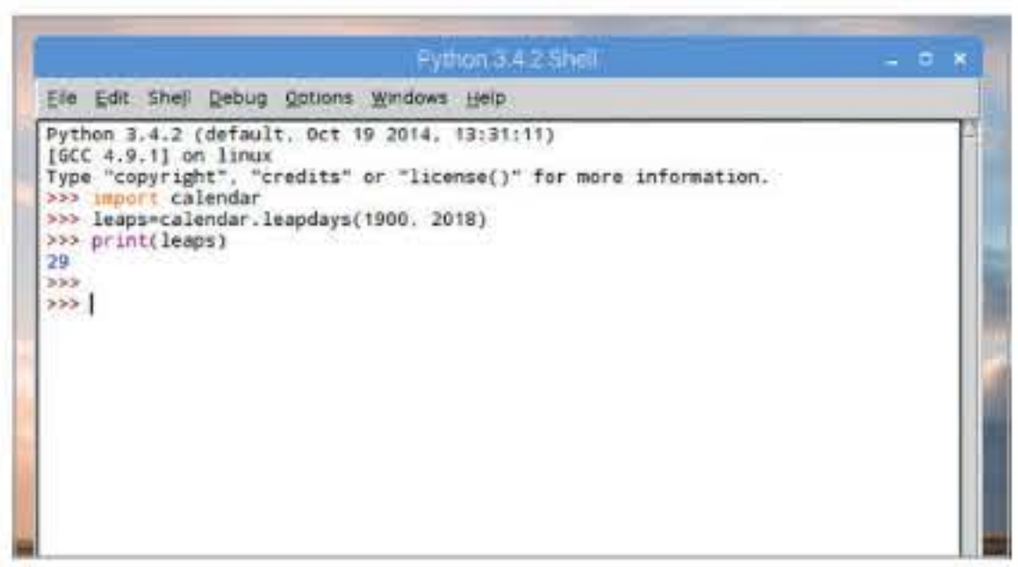
STEP 2 You can see that the days of September 2017 are displayed in a wall calendar fashion. Naturally you can change the 2017, 9 part of the second line to any year and month you want, a birthday for example (1973, 6). The first line configures TextCalendar to start its weeks on a Sunday; you can opt for Monday if you prefer.



STEP 3 There are numerous functions within the calendar module that may be of interest to you when forming your own code. For example, you can display the number of leap years between two specific years:

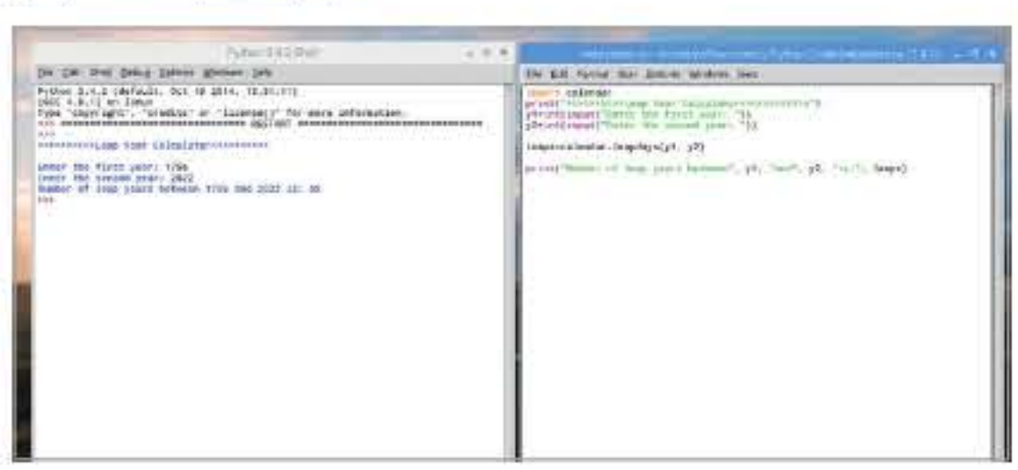
```
leaps=calendar.leapdays(1900, 2018)
print(leaps)
```

The result is 29, starting from 1904 onward.



STEP 4 You could even fashion that particular example into a piece of working, user interactive Python code:

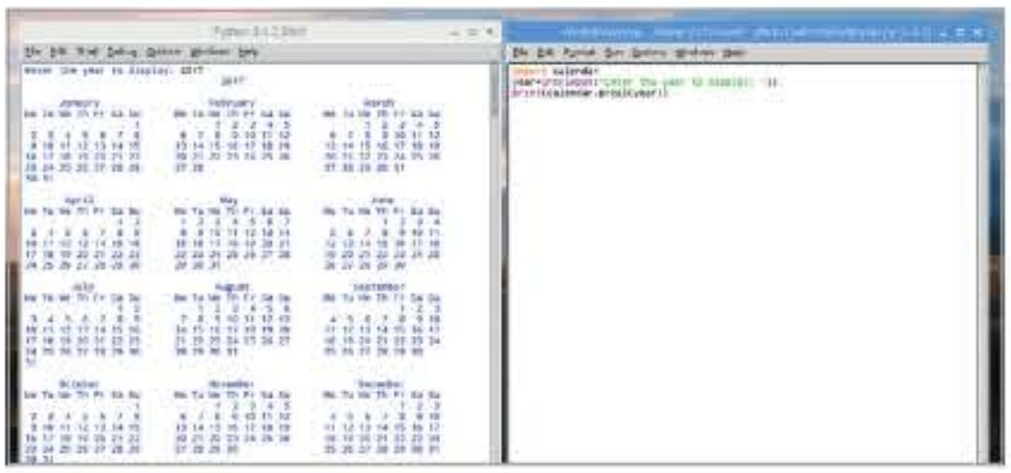
```
import calendar
print(">>>>>>>>>>Leap Year
Calculator<<<<<<<<<<\n")y1=int(input("Enter the
first year: "))
y2=int(input("Enter the second year: "))
leaps=calendar.leapdays(y1, y2)
print("Number of leap years between", y1, "and",
y2, "is:", leaps)
```



STEP 5 You can also create a program that will display all the days, weeks and months within a given year:

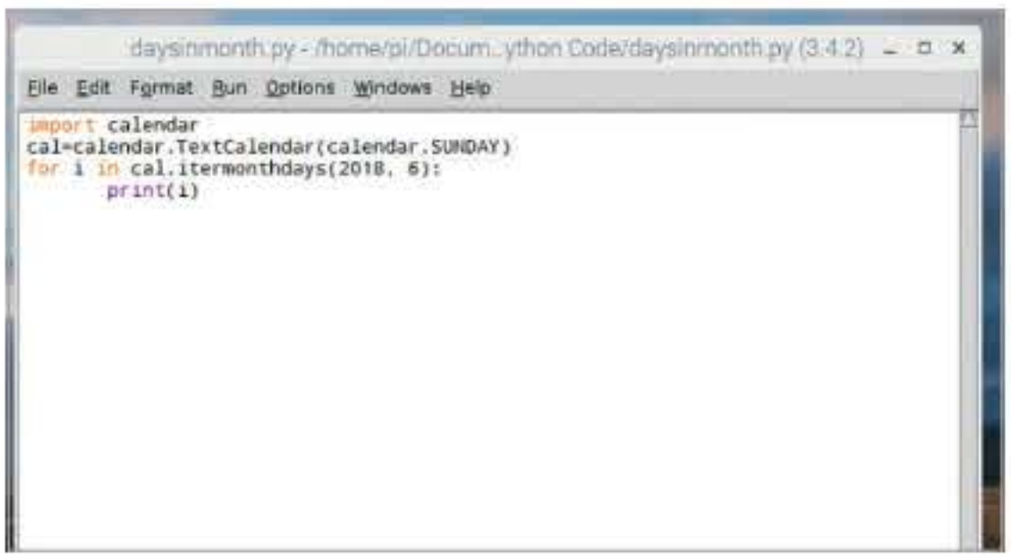
```
import calendar
year=int(input("Enter the year to display: ")
print(calendar.prcal(year))
```

We're sure you'll agree that's quite a handy bit of code to have to hand.

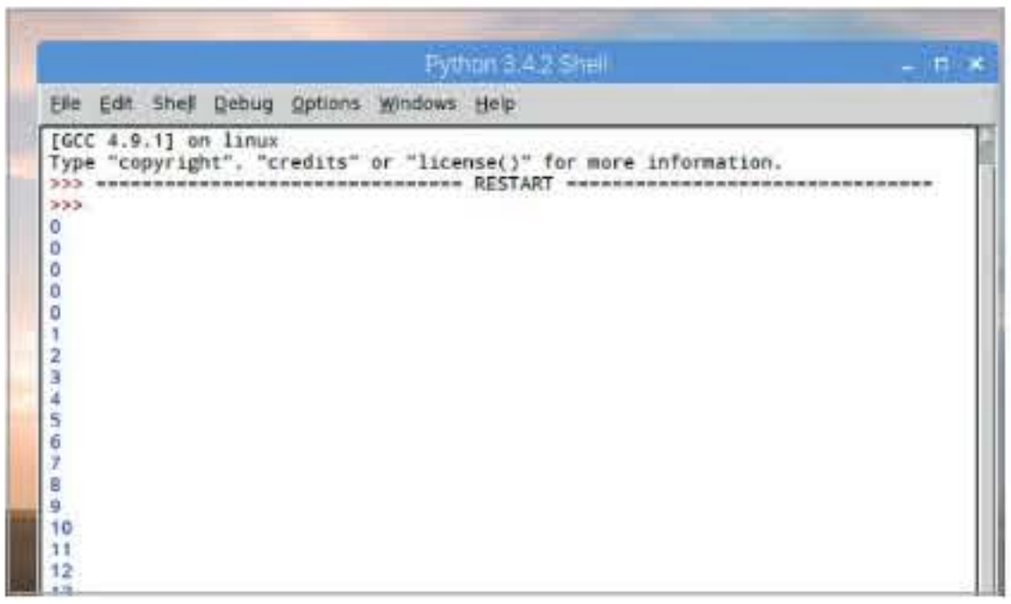


STEP 6 Interestingly we can also list the number of days in a month by using a simple for loop:

```
import calendar
cal=calendar.TextCalendar(calendar.SUNDAY)
for i in cal.itermonthdays(2018, 6):
  print(i)
```



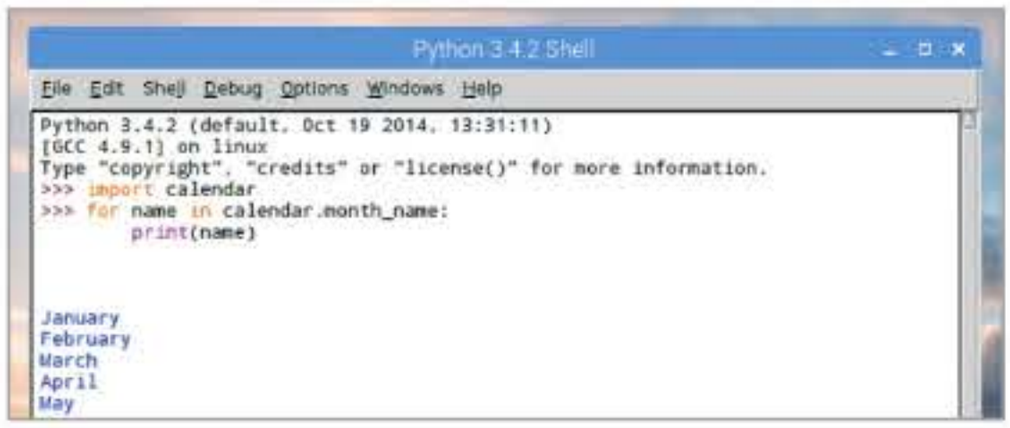
STEP 7 You can see that code produced some zeros at the beginning, this is due to the starting day of the week, Sunday in this case, and overlapping days from the previous month. So the counting of the days will start on Friday 1st June 2018 and will total 30 as the output correctly displays.



STEP 8 You're also able to print the individual months or days of the week:

```
import calendar
for name in calendar.month_name:
  print(name)
```

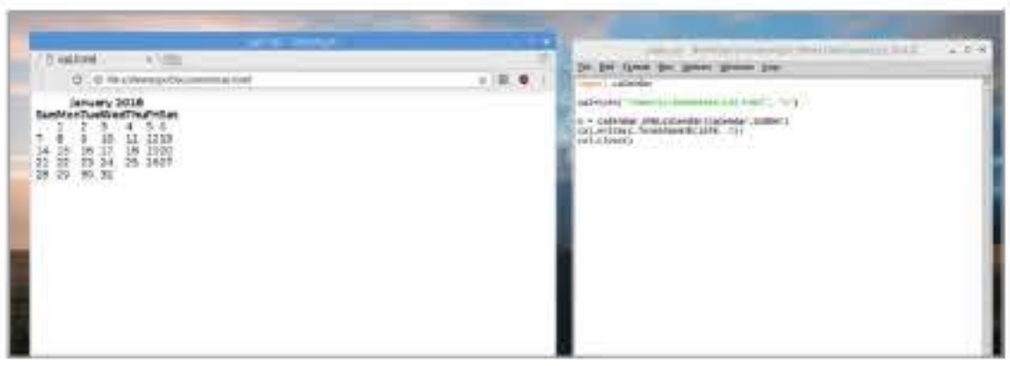
```
import calendar
for name in calendar.day_name:
  print(name)
```



STEP 9 The calendar module also allows us to write the functions in HTML, so that you can display it on a website. Let's start by creating a new file:

```
import calendar
cal=open("/home/pi/Documents/cal.html", "w")
c=calendar.HTMLCalendar(calendar.SUNDAY)
cal.write(c.formatmonth(2018, 1))
cal.close()
```

This code will create an HTML file called cal, open it with a browser and it displays the calendar for January 2018.



STEP 10 Of course, you can modify that to display a given year as a web page calendar:

```
import calendar

year=int(input("Enter the year to display as a
webpage: "))
cal=open("/home/pi/Documents/cal.html", "w")
cal.write(calendar.HTMLCalendar(calendar.MONDAY).
formatyear(year))
cal.close()
```

This code asks the user for a year, then creates the necessary webpage. Remember to change your file destination.

OS Module

The OS module allows you to interact directly with the built-in commands found in your operating system. Commands vary depending on the OS you're running, as some will work with Windows whereas others will work with Linux and macOS.

INTO THE SYSTEM

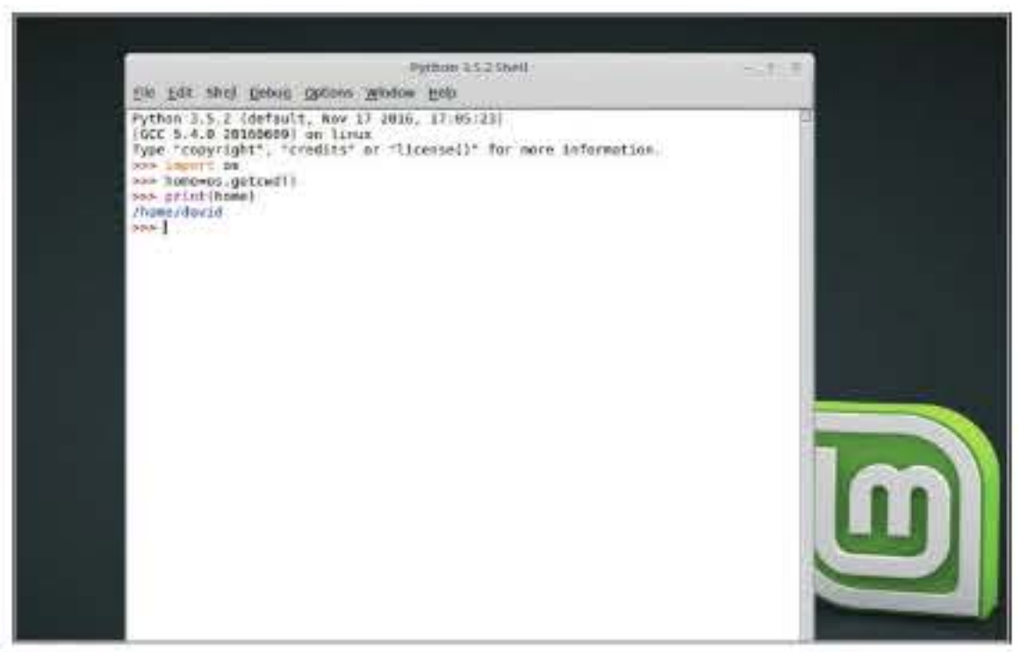
One of the primary features of the OS module is the ability to list, move, create, delete and otherwise interact with files stored on the system, making it the perfect module for backup code.

STEP 1 You can start the OS module with some simple functions to see how it interacts with the operating system environment that Python is running on. If you're using Linux or the Raspberry Pi, try this:

```
import os
home=os.getcwd()
print(home)
```

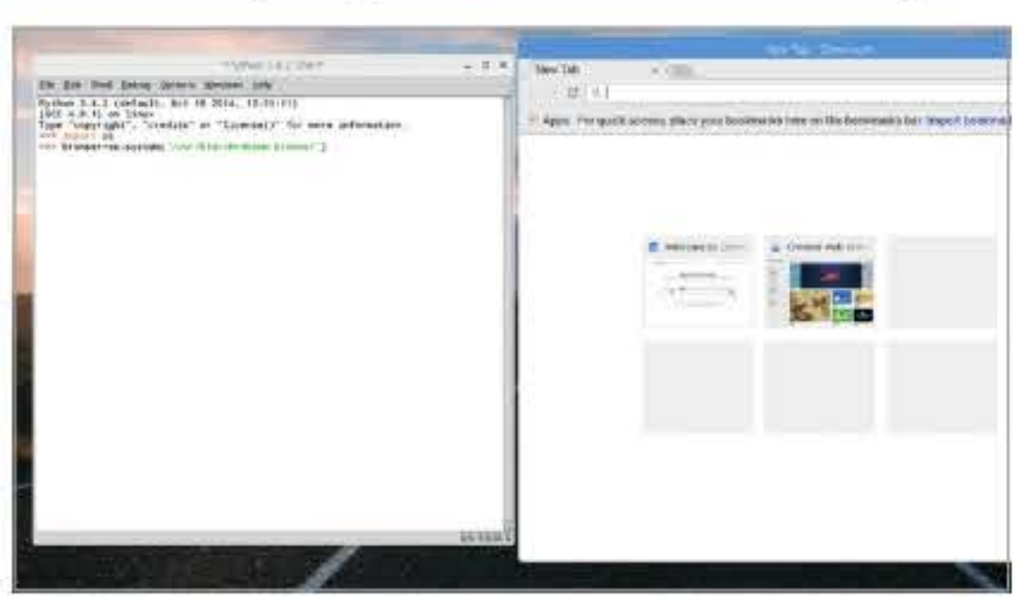
STEP 2 The returned result from printing the variable home is the current user's home folder on the system. In our example that's /home/pi; it will be different depending on the user name you login as and the operating system you use. For example, Windows 10 will output: C:\Program Files (x86)\Python36-32.

STEP 3 The Windows output is different as that's the current working directory of Python, as determined by the system; as you might suspect, the os.getcwd() function is asking Python to retrieve the Current Working Directory. Linux users will see something along the same lines as the Raspberry Pi, as will macOS users.



STEP 4 Yet another interesting element to the OS module, is its ability to launch programs that are installed in the host system. For instance, if you wanted to launch the Chromium browser from within a Python program you can use the command:

```
import os
browser=os.system("/usr/bin/chromium-browser")
```



STEP 5 The os.system() function is what allows interaction with external programs; you can even call up previous Python programs using this method. You will obviously need to know the full path and program file name for it to work successfully. However, you can use the following:

```
import os
os.system('start chrome "https://www.youtube.com/feed/music"')
```

STEP 6 For Step 5's example we used Windows, to show that the OS module works roughly the same across all platforms. In that case, we opened YouTube's music feed page, so it is therefore possible to open specific pages:

```
import os
os.system('chromium-browser "http://bdmpublications.com/"')
```

STEP 7 Note in the previous step's example the use of single and double-quotes. The single quotes encase the entire command and launching Chromium, whereas the double quotes open the specified page. You can even use variables to call multiple tabs in the same browser:

```
import os
a=('chromium-browser "http://bdmpublications.com/"')
b=('chromium-browser "http://www.google.co.uk"')
os.system(a + b)
```

STEP 8 The ability to manipulate directories, or folders if you prefer, is one of the OS module's best features. For example, to create a new directory you can use:

```
import os
os.mkdir("NEW")
```

This creates a new directory within the Current Working Directory, named according to the object in the mkdir function.

STEP 9 You can also rename any directories you've created by entering:

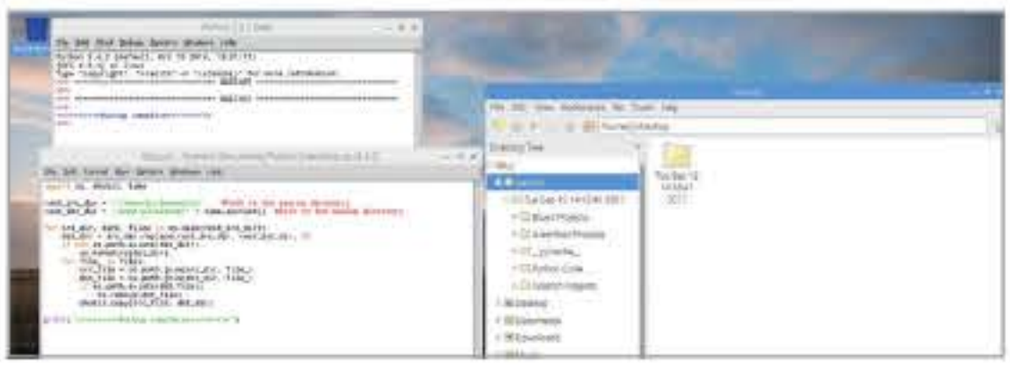
```
import os
os.rename("NEW", "OLD")
```

To delete them:

```
import os
os.rmdir("OLD")
```

STEP 10 Another module that goes together with OS is shutil. You can use the shutil module together with OS and time to create a time-stamped backup directory, and copy files into it:

```
import os, shutil, time

root_src_dir = r'/home/pi/Documents'
root_dst_dir = '/home/pi/backup/' + time.asctime()

for src_dir, dirs, files in os.walk(root_src_dir):
  dst_dir = src_dir.replace(root_src_dir, root_dst_dir, 1)
  if not os.path.exists(dst_dir):
    os.makedirs(dst_dir)
  for file_ in files:
    src_file = os.path.join(src_dir, file_)
    dst_file = os.path.join(dst_dir, file_)
    if os.path.exists(dst_file):
      os.remove(dst_file)
    shutil.copy(src_file, dst_dir)

print(">>>>>>>>>>Backup complete<<<<<<<<<<")
```



Random Module

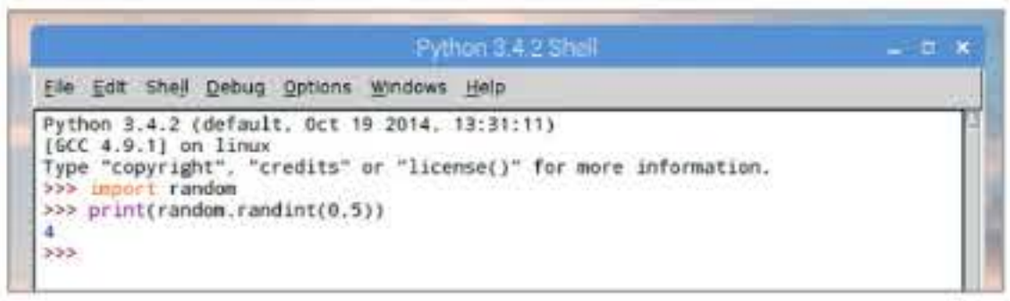
The random module is one you will likely come across many times in your Python programming lifetime; as the name suggests, it's designed to create random numbers or letters. However, it's not exactly random but it will suffice for most needs.

RANDOM NUMBERS

There are numerous functions within the random module, which when applied can create some interesting and very useful Python programs.

STEP 1 Just as with other modules you need to import random before you can use any of the functions we're going to look at in this tutorial. Let's begin by simply printing a random number from 1 to 5:

```
import random
print(randomint(0,5))
```



STEP 2 In our example the number four was returned. However, enter the print function a few more times and it will display different integer values from the set of numbers given, zero to five. The overall effect, although pseudo-random, is adequate for the average programmer to utilise in their code.

STEP 3 For a bigger set of numbers, including floating point values, you can extend the range by using the multiplication sign:

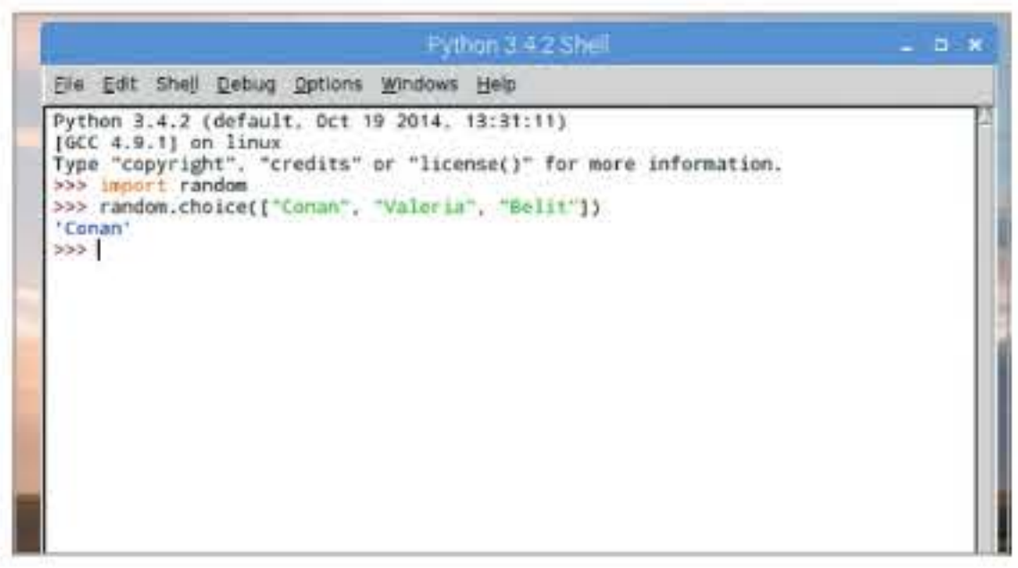
```
import random
print(random.random() *100)
```

Will display a floating point number between 0 and 100, to the tune of around fifteen decimal points.

STEP 4 However, the random module isn't used exclusively for numbers. You can use it to select an entry from a list from random, and the list can contain anything:

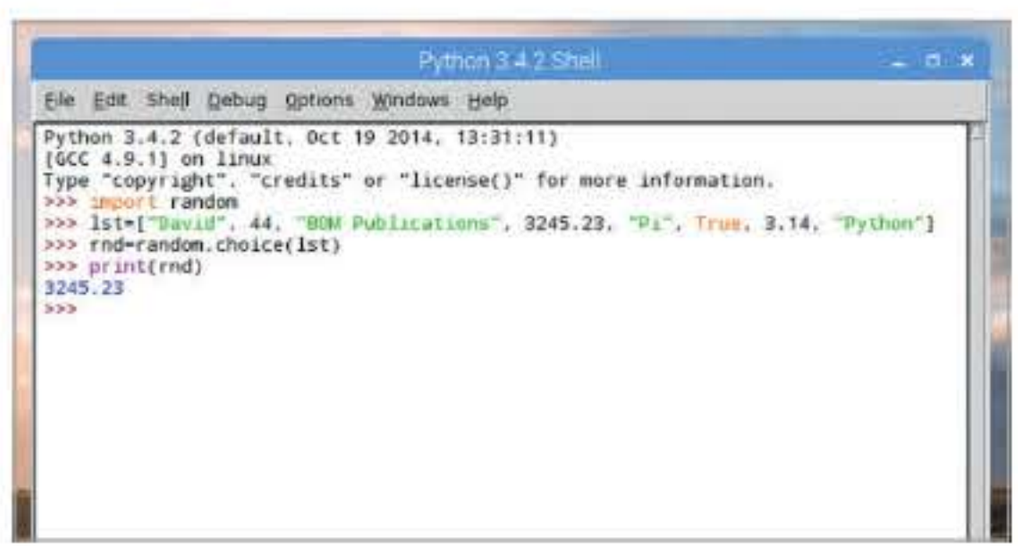
```
import random
random.choice(["Conan", "Valeria", "Belit"])
```

This will display one of the names of our adventurers at random, which is a great addition to a text adventure game.



STEP 5 You can extend the previous example somewhat by having random.choice() select from a list of mixed variables. For instance:

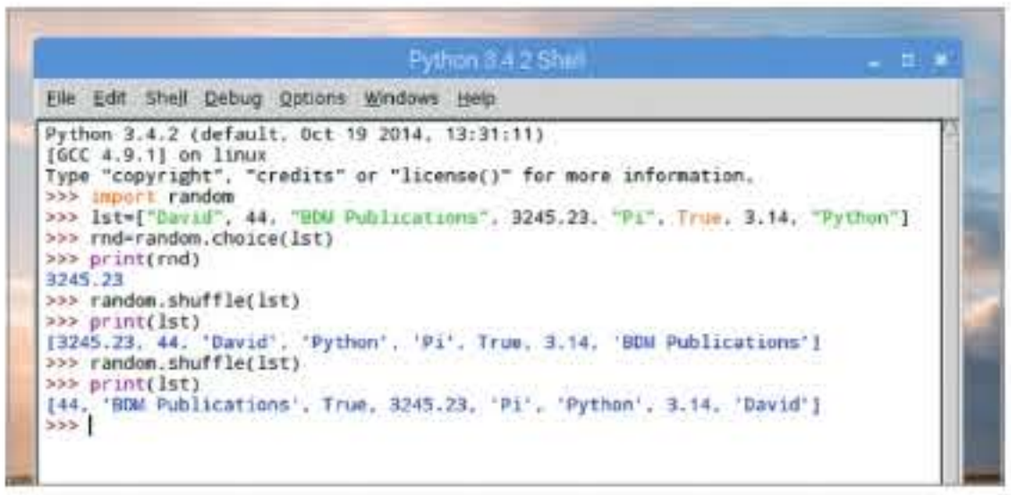
```
import random
lst=["David", 44, "BDM Publications", 3245.23,
"Pi", True, 3.14, "Python"]
rnd=random.choice(lst)
print(rnd)
```



STEP 6 Interestingly, you can also use a function within the random module to shuffle the items in the list, thus adding a little more randomness into the equation:

```
random.shuffle(lst)
print(lst)
```

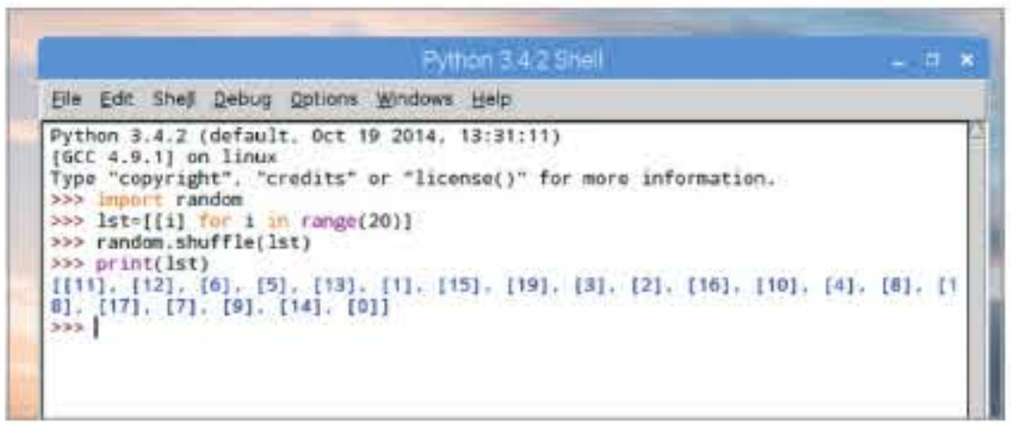
This way, you can keep shuffling the list before displaying a random item from it.



STEP 7 Using shuffle, you can create an entirely random list of numbers. For example, within a given range:

```
import random
lst=[[i] for I in range(20)]
random.shuffle(lst)
print(lst)
```

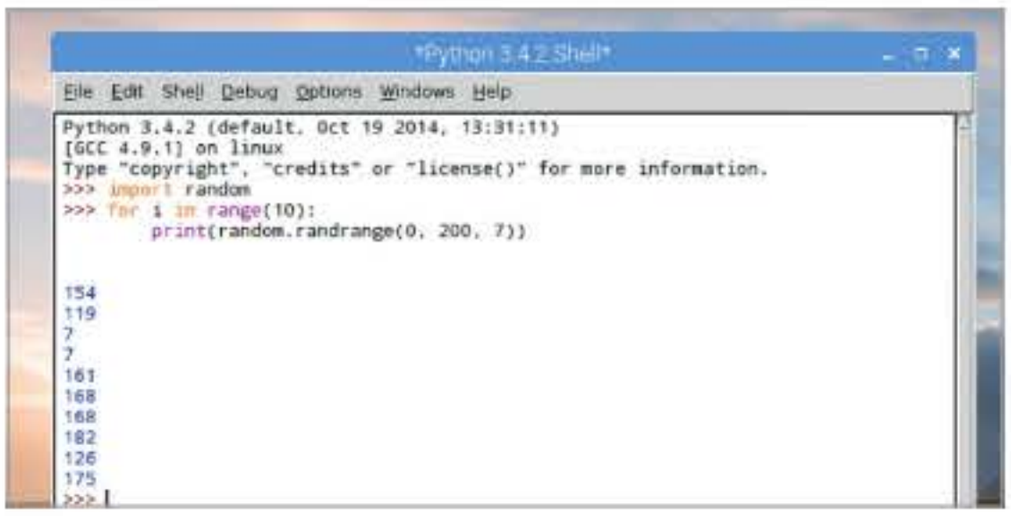
Keep shuffling the list and you can have a different selection of items from 0 to 20 every time.



STEP 8 You can also select a random number from a given range in steps, using the start, stop, step loop:

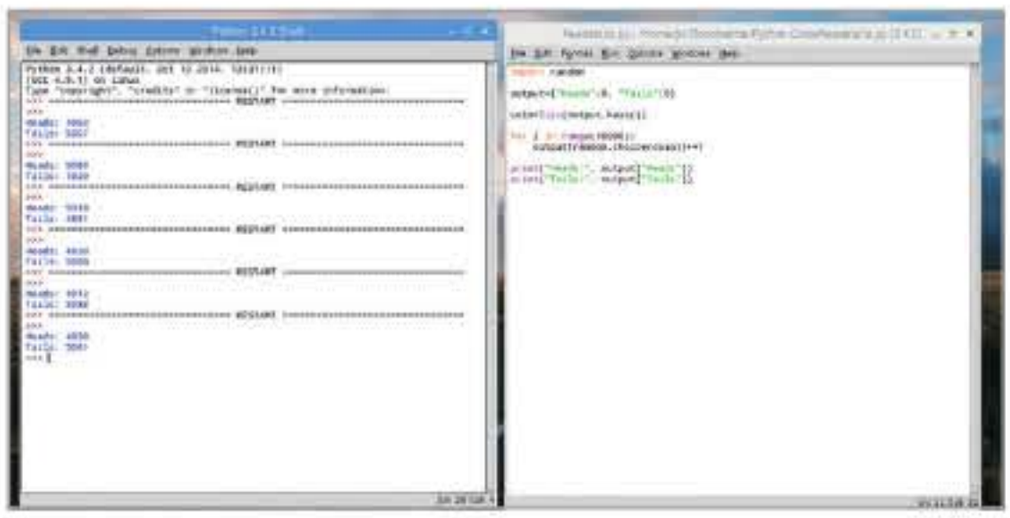
```
import random
for i in range(10):
  print(random.randrange(0, 200, 7))
```

Results will vary but you get the general idea as to how it works.



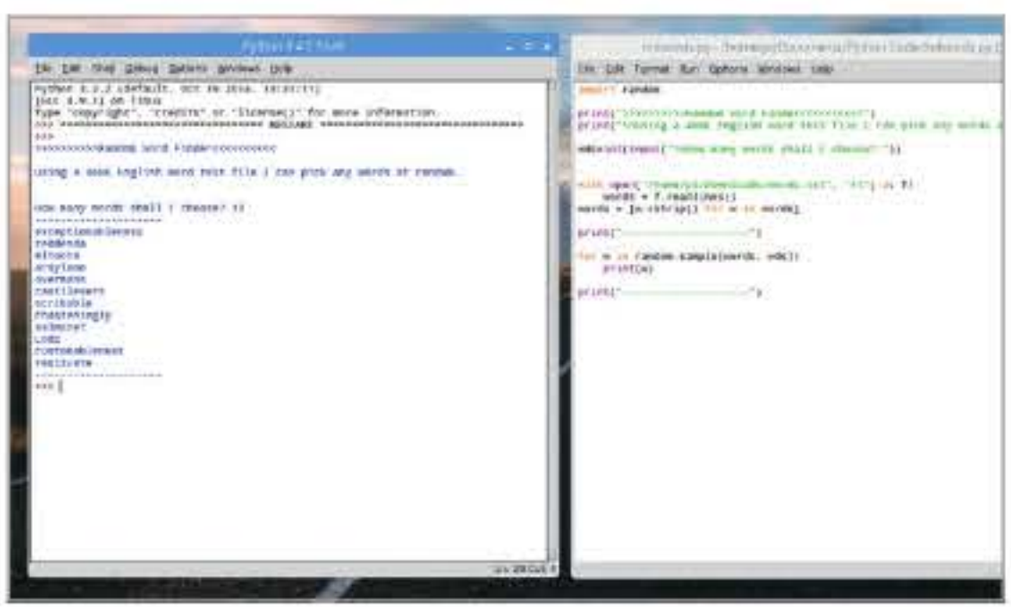
STEP 9 Let's use an example piece of code which flips a virtual coin ten thousand times and counts how many times it will land on heads or tails:

```
import random
output={"Heads":0, "Tails":0}
coin=list(output.keys())

for i in range(10000):
  output[random.choice(coin)]+=1

print("Heads:", output["Heads"])
print("Tails:", output["Tails"])
```



STEP 10 Here's an interesting piece of code. Using a text file containing 466 thousand words, you can pluck a user generated number of words from the file (text file found at: www.github.com/dwyl/english-words):

```
import random

print(">>>>>>>>>>Random Word Finder<<<<<<<<<<")
print("\nUsing a 466K English word text file I can
pick any words at random.\n")

wds=int(input("\nHow many words shall I
choose? "))

with open("/home/pi/Downloads/words.txt",
"rt") as f:
  words = f.readlines()
words = [w.rstrip() for w in words]

print("--------------------")

for w in random.sample(words, wds):
  print(w)

print("--------------------")
```



Tkinter Module

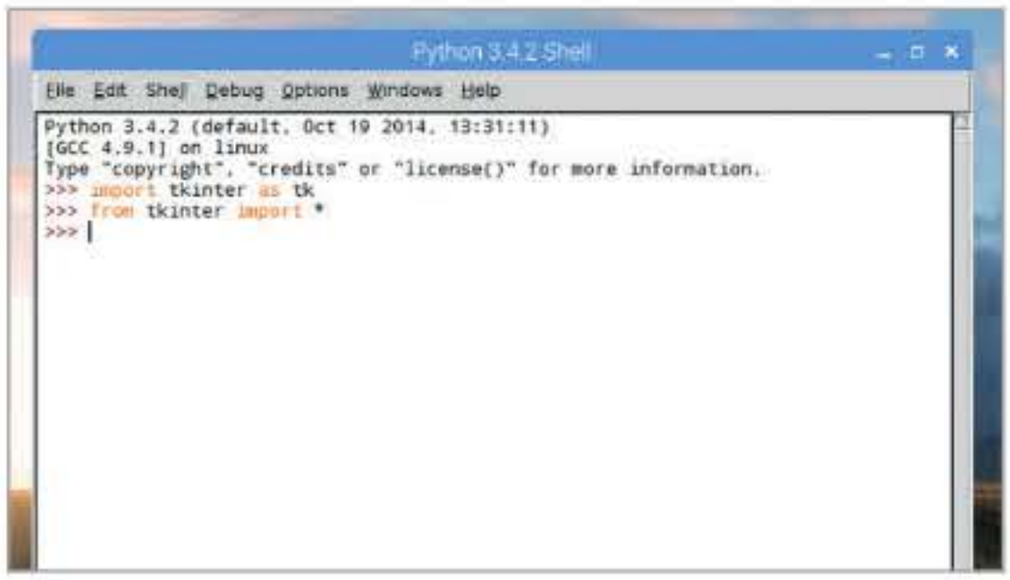
Whilst running your code from the command line, or even in the Shell, is perfectly fine, Python is capable of so much more. The Tkinter module enables the programmer to set up a Graphical User Interface to interact with the user, and it's surprisingly powerful too.

GETTING GUI

Tkinter is easy to use but there's a lot more you can do with it. Let's start by seeing how it works and getting some code into it. Before long you will discover just how powerful this module really is.

STEP 1 Tkinter is usually built into Python 3. However, if it's available when you enter: `import tkinter`, then you need to `pip install tkinter` from the command prompt. We can start to import modules differently than before, to save on typing and by importing all their contents:

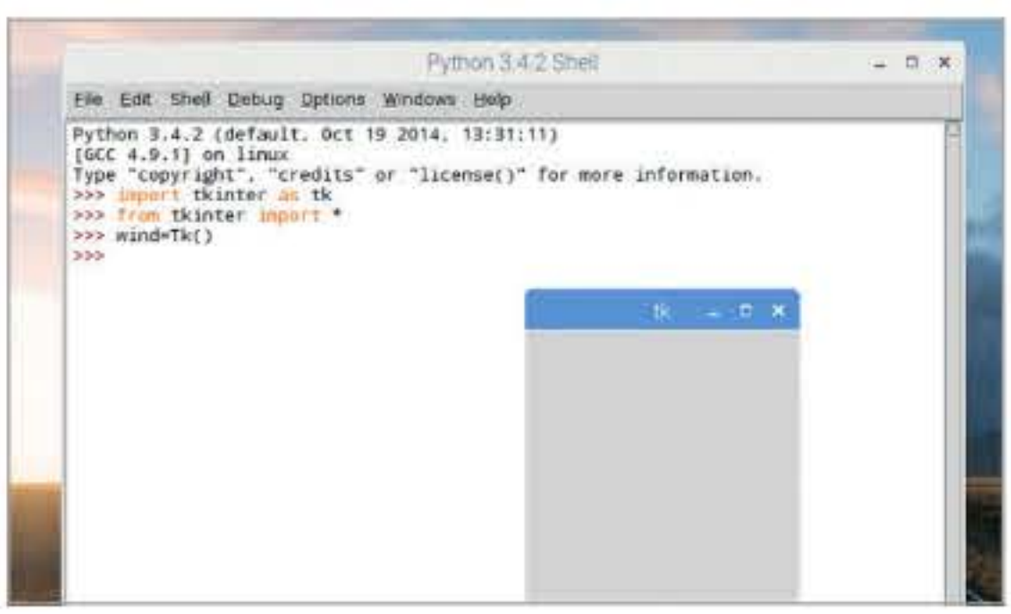
```
import tkinter as tk
from tkinter import *
```



STEP 2 It's not recommended to import everything from a module using the asterisk but it won't do any harm normally. Let's begin by creating a basic GUI window, enter:

```
wind=Tk()
```

This creates a small, basic window. There's not much else to do at this point but click the X in the corner to close the window.



STEP 3 The ideal approach is to add mainloop() into the code to control the Tkinter event loop, but we'll get to that soon. You've just created a Tkinter widget and there are several more we can play around with:

```
btn=Button()
btn.pack()
btn["text"]="Hello everyone!"
```

The first line focuses on the newly created window. Click back into the Shell and continue the other lines.

STEP 4 You can combine the above into a New File:

```
import tkinter as tk
from tkinter import *
btn=Button()
btn.pack()
btn["text"]="Hello everyone!"
```

Then add some button interactions:

```
def click():
  print("You just clicked me!")
btn["command"]=click
```

STEP 5 Save and execute the code from Step 5 and a window appears with 'Hello everyone!' inside. If you click the Hello everyone! button, the Shell will output the text 'You just clicked me!'. It's simple but shows you what can be achieved with a few lines of code.

STEP 6 You can also display both text and images within a Tkinter window. However, only GIF, PGM or PPM formats are supported. So find an image and convert it before using the code. Here's an example using the BDM Publishing logo:

```
from tkinter import *

root = Tk()
logo = PhotoImage(file="/home/pi/Downloads/BDM_
logo.gif")
w1 = Label(root, root.title("BDM Publications"),
image=logo).pack(side="right")
content = """  From its humble beginnings in 2004,
the BDM brand quickly grew from a single publication
produced by a team of just two to one of the biggest
names in global bookazine publishing, for two simple
reasons. Our passion and commitment to deliver the
very best product each and every volume. While
the company has grown with a portfolio of over 250
publications delivered by our international staff,
the foundation that it has been built upon remains
the same, which is why we believe BDM isn't just
the first choice it's the only choice for the smart
consumer.  """
w2 = Label(root,
    justify=LEFT,
    padx = 10,
    text=content).pack(side="left")
root.mainloop()
```

STEP 7

The previous code is quite weighty, mostly due to the content variable holding a part of BDM's About page from the company website. You can obviously change the content, the root.title and the image to suit your needs.

STEP 8 You can create radio buttons too. Try:

```
from tkinter import *

root = Tk()

v = IntVar()

Label(root, root.title("Options"), text="""Choose
a preferred language:""",
  justify = LEFT, padx = 20).pack()
Radiobutton(root,
    text="Python",
    padx = 20,
    variable=v,
    value=1).pack(anchor=W)
Radiobutton(root,
    text="C++",
    padx = 20,
    variable=v,
    value=2).pack(anchor=W)

mainloop()
```

STEP 9 You can also create check boxes, with buttons and output to the Shell:

```
from tkinter import *
root = Tk()

def var_states():
  print("Warrior: %d,\nMage: %d" % (var1.get(),
var2.get()))

Label(root, root.title("Adventure Game"),
text=">>>>>>>>>>Your adventure role<<<<<<<<<<").
grid(row=0, sticky=N)
var1 = IntVar()
Checkbutton(root, text="Warrior", variable=var1).
grid(row=1, sticky=W)
var2 = IntVar()
Checkbutton(root, text="Mage", variable=var2).
grid(row=2, sticky=W)
Button(root, text='Quit', command=root.destroy).
grid(row=3, sticky=W, pady=4)
Button(root, text='Show', command=var_states).
grid(row=3, sticky=E, pady=4)

mainloop()
```

STEP 10 The code from Step 9 introduced some new geometry elements into Tkinter. Note the sticky=N, E and W arguments. These describe the locations of the check boxes and buttons (North, East, South and West). The row argument places them on separate rows. Have a play around and see what you get.

Pygame Module

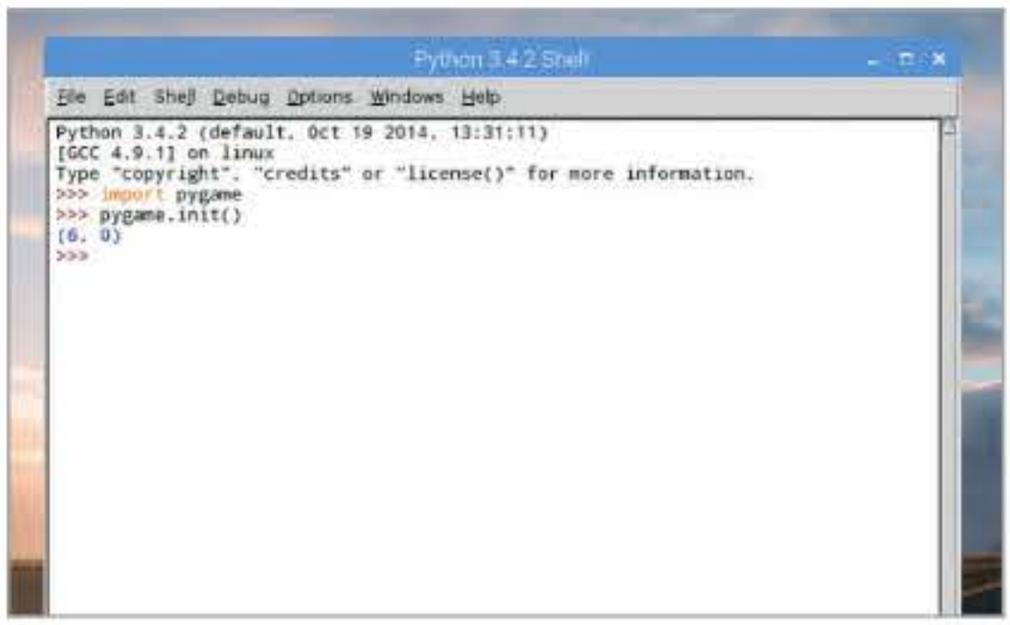
We've had a brief look at the Pygame module already but there's a lot more to it that needs exploring. Pygame was developed to help Python programmers create either graphical or text-based games.

PYGAMING

Pygame isn't an inherent module to Python but those using the Raspberry Pi will already have it installed. Everyone else will need to use: **pip install pygame** from the command prompt.

STEP 1 Naturally you need to load up the Pygame modules into memory before you're able to utilise them. Once that's done Pygame requires the user to initialise it prior to any of the functions being used:

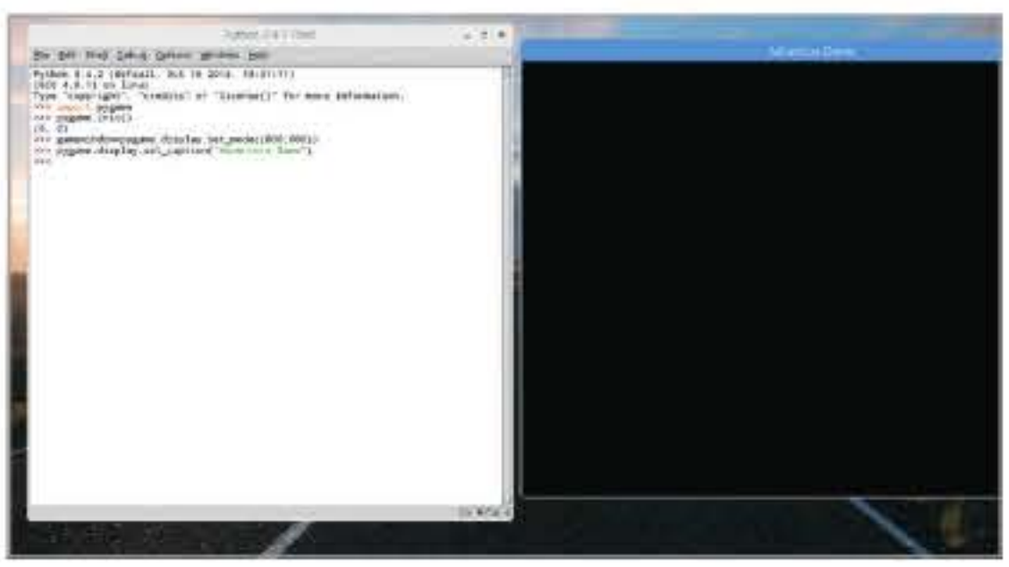
```
import pygame
pygame.init()
```



STEP 2 Let's create a simple game ready window, and give it a title:

```
gamewindow=pygame.display.set_mode((800,600))
pygame.display.set_caption("Adventure Game")
```

You can see that after the first line is entered, you need to click back into the IDLE Shell to continue entering code; also, you can change the title of the window to anything you like.



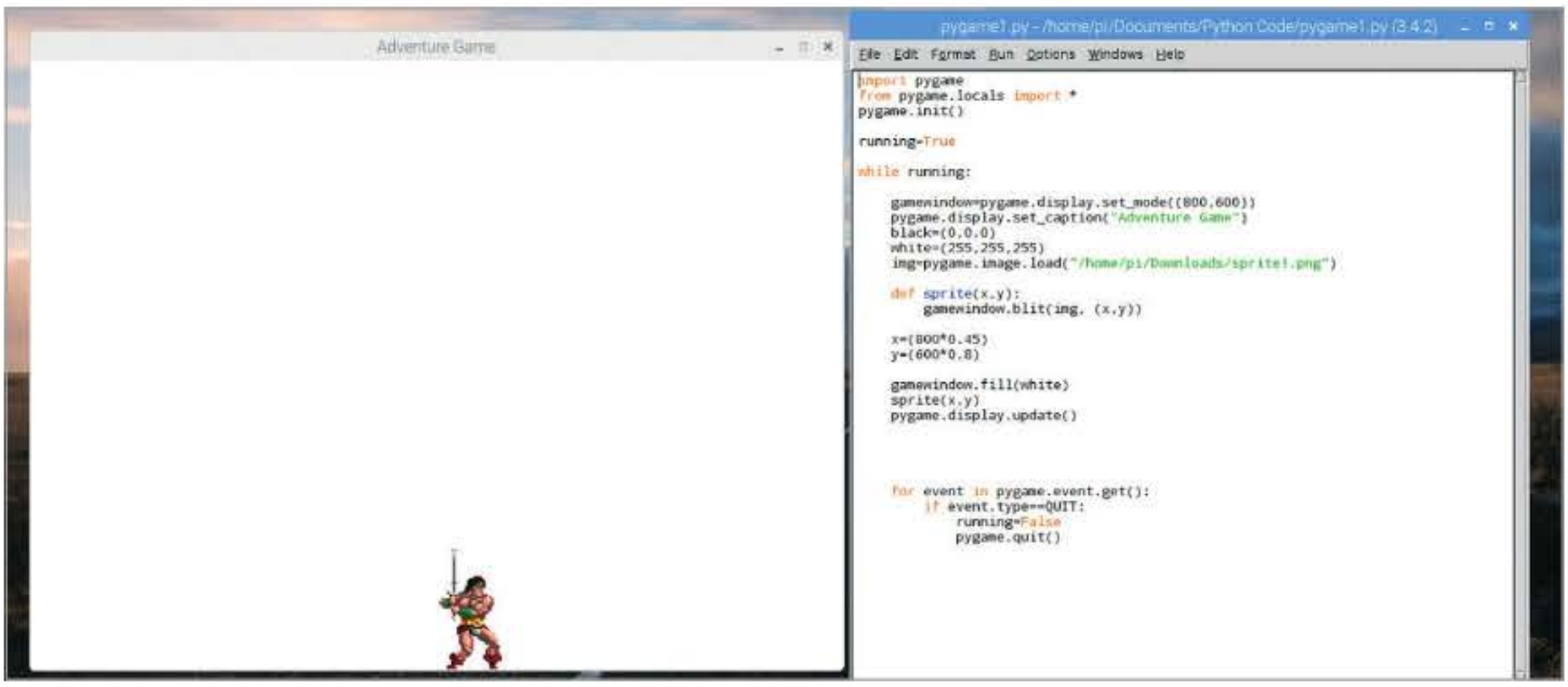
STEP 3 Sadly you can't close the newly created Pygame window without closing the Python IDLE Shell, which isn't very practical. For this reason, you need to work in the editor (New > File) and create a True/False while loop:

```
import pygame
from pygame.locals import *
pygame.init()

gamewindow=pygame.display.set_mode((800,600))
pygame.display.set_caption("Adventure Game")

running=True

while running:
  for event in pygame.event.get():
    if event.type==QUIT:
      running=False
      pygame.quit()
```

STEP 4 If the Pygame window still won't close don't worry, it's just a discrepancy between the IDLE (which is written with Tkinter) and the Pygame module. If you run your code via the command line, it closes perfectly well.

STEP 5 You're going to shift the code around a bit now, running the main Pygame code within a while loop; it makes it neater and easier to follow. We've downloaded a graphic to use and we need to set some parameters for pygame:

```
import pygame
pygame.init()

running=True

while running:

  gamewindow=pygame.display.set_mode((800,600))
  pygame.display.set_caption("Adventure Game")
  black=(0,0,0)
  white=(255,255,255)
  img=pygame.image.load("/home/pi/Downloads/
sprite1.png")

  def sprite(x,y):
    gamewindow.blit(img, (x,y))

  x=(800*0.45)
  y=(600*0.8)

  gamewindow.fill(white)
  sprite(x,y)
  pygame.display.update()

  for event in pygame.event.get():
    if event.type==pygame.QUIT:
      running=False
```



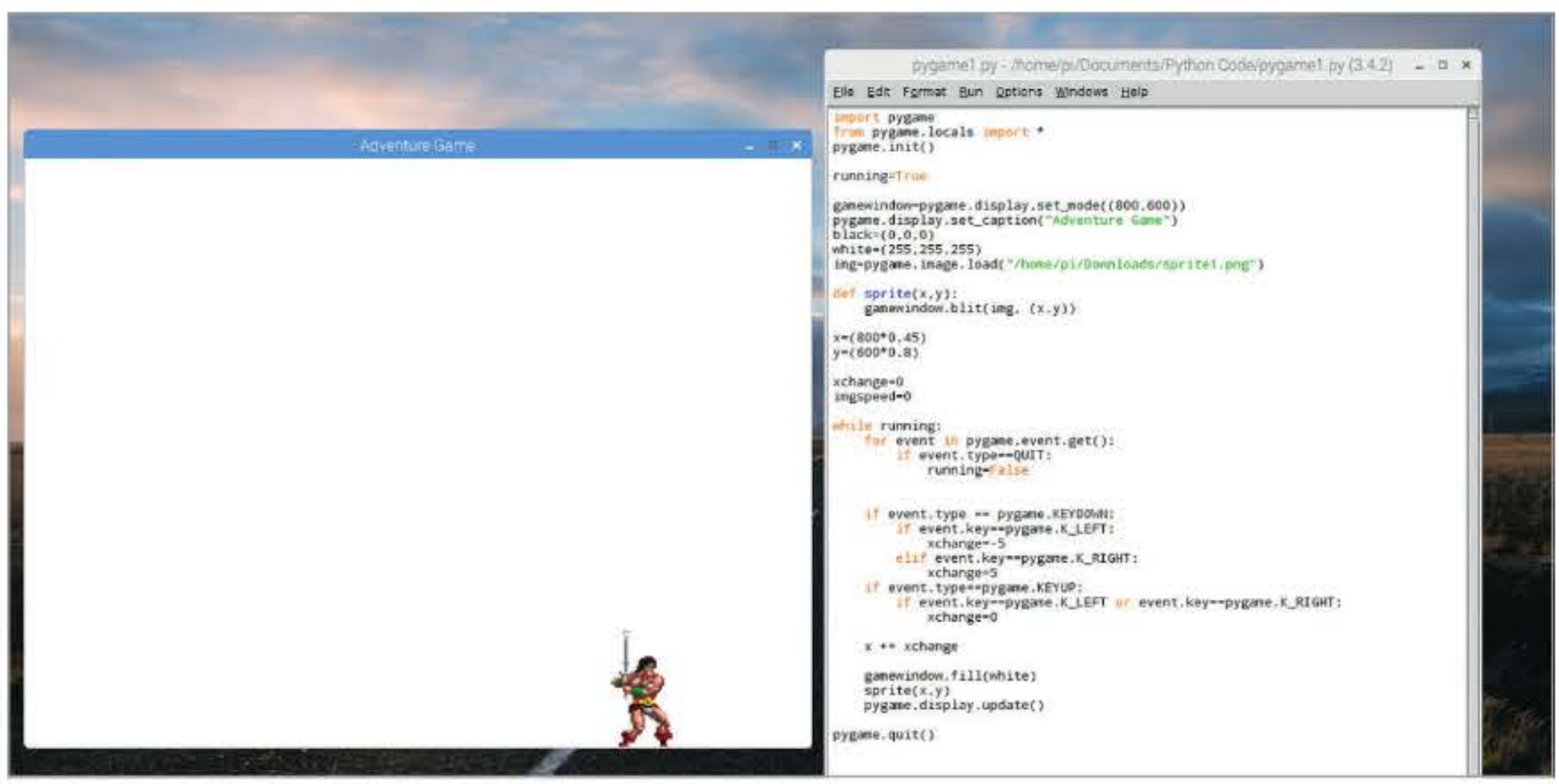
STEP 6 Let's quickly go through the code changes. We've defined two colours, black and white together with their respective RGB colour values. Next we've loaded the downloaded image called sprite1.png and allocated it to the variable img; and also defined a sprite function and the Blit function will allow us to eventually move the image.

```
import pygame
from pygame.locals import *
pygame.init()

running=True

while running:

    gamewindow=pygame.display.set_mode((800,600))
    pygame.display.set_caption("Adventure Game")
    black=(0,0,0)
    white=(255,255,255)
    img=pygame.image.load("/home/pi/Downloads/sprite1.png")

    def sprite(x,y):
        gamewindow.blit(img, (x,y))
```

```
    x=(800*0.45)
    y=(600*0.8)

    gamewindow.fill(white)
    sprite(x,y)
    pygame.display.update()



    for event in pygame.event.get():
        if event.type==QUIT:
            running=False
            pygame.quit()
```

STEP 7 Now we can change the code around again, this time containing a movement option within the while loop, and adding the variables needed to move the sprite around the screen:

```
import pygame
from pygame.locals import *
pygame.init()

running=True

gamewindow=pygame.display.set_mode((800,600))
pygame.display.set_caption("Adventure Game")
black=(0,0,0)
white=(255,255,255)
img=pygame.image.load("/home/pi/Downloads/
sprite1.png")

def sprite(x,y):
  gamewindow.blit(img, (x,y))

x=(800*0.45)
y=(600*0.8)

xchange=0

imgspeed=0

while running:
  for event in pygame.event.get():
    if event.type==QUIT:
      running=False

  if event.type == pygame.KEYDOWN:
    if event.key==pygame.K_LEFT:
      xchange=-5
    elif event.key==pygame.K_RIGHT:
      xchange=5
  if event.type==pygame.KEYUP:
    if event.key==pygame.K_LEFT or event
key==pygame.K_RIGHT:
      xchange=0

  x += xchange

  gamewindow.fill(white)
  sprite(x,y)
  pygame.display.update()

pygame.quit()
```



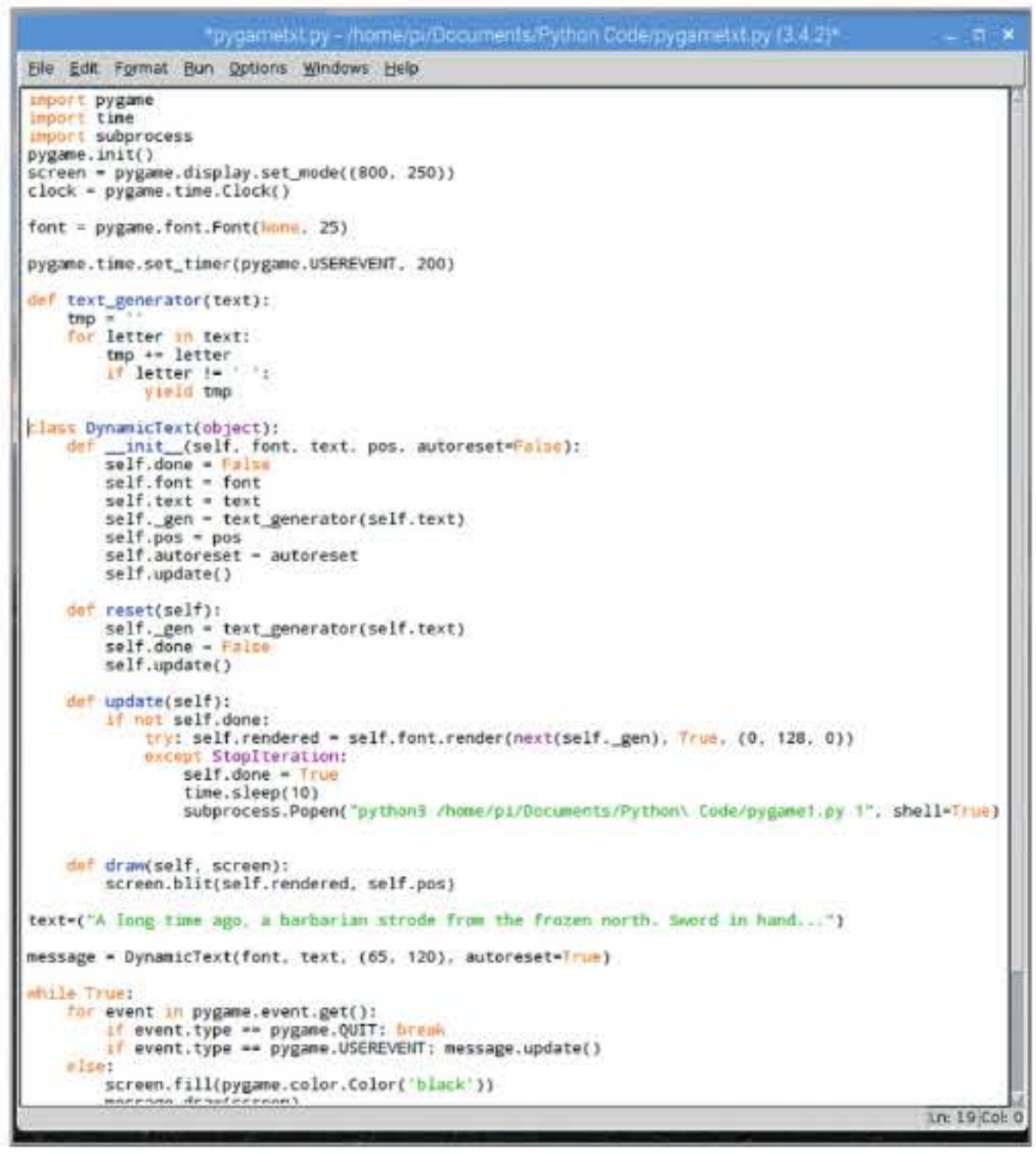
STEP 8 Copy the code down and using the left and right arrow keys on the keyboard you can move your sprite across the bottom of the screen. Now, it looks like you have the makings of a classic arcade 2D scroller in the works.

```
import pygame
from pygame.locals import *
pygame.init()

running=True

gamewindow=pygame.display.set_mode((800,600))
pygame.display.set_caption("Adventure Game")
black=(0,0,0)
white=(255,255,255)
img=pygame.image.load("/home/pi/Downloads/sprite1.png")

def sprite(x,y):
    gamewindow.blit(img, (x,y))

x=(800*0.45)
y=(600*0.8)

xchange=0
imgspeed=0
```

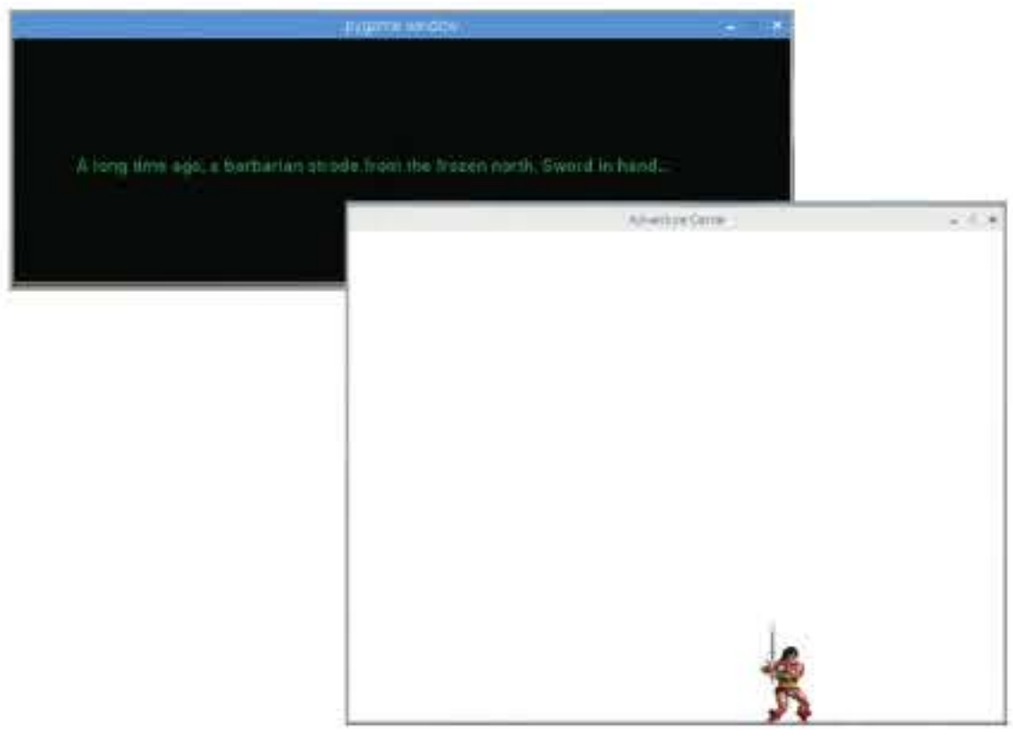
```
while running:
    for event in pygame.event.get():
        if event.type==QUIT:
            running=False


    if event.type == pygame.KEYDOWN:
        if event.key==pygame.K_LEFT:
            xchange=-5
        elif event.key==pygame.K_RIGHT:
            xchange=5
    if event.type==pygame.KEYUP:
        if event.key==pygame.K_LEFT or event.key==pygame.K_RIGHT:
            xchange=0

    x += xchange

    gamewindow.fill(white)
    sprite(x,y)
    pygame.display.update()

pygame.quit()
```

STEP 9 You can now implement a few additions and utilise some previous tutorial code. The new elements are the subprocess module, of which one function allows us to launch a second Python script from within another; and we're going to create a New File called pygametxt.py:

```
import pygame
import time
import subprocess
pygame.init()
screen = pygame.display.set_mode((800, 250))
clock = pygame.time.Clock()

font = pygame.font.Font(None, 25)

pygame.time.set_timer(pygame.USEREVENT, 200)

def text_generator(text):
  tmp = ''
  for letter in text:
    tmp += letter
    if letter != ' ':
      yield tmp

class DynamicText(object):
  def __init__(self, font, text, pos,
autoreset=False):
    self.done = False
    self.font = font
    self.text = text
    self._gen = text_generator(self.text)
    self.pos = pos
    self.autoreset = autoreset
    self.update()

  def reset(self):
    self._gen = text_generator(self.text)
    self.done = False
    self.update()

  def update(self):
    if not self.done:
      try: self.rendered = self.font.
render(next(self._gen), True, (0, 128, 0))
      except StopIteration:
        self.done = True
        time.sleep(10)
        subprocess.Popen("python3 /home/pi/Documents/
Python\ Code/pygame1.py 1", shell=True)

  def draw(self, screen):
    screen.blit(self.rendered, self.pos)

text=("A long time ago, a barbarian strode from the
frozen north. Sword in hand...")

message = DynamicText(font, text, (65, 120),
autoreset=True)

while True:
  for event in pygame.event.get():
    if event.type == pygame.QUIT: break
    if event.type == pygame.USEREVENT:
message.update()
  else:
    screen.fill(pygame.color.Color('black'))
    message.draw(screen)
    pygame.display.flip()
    clock.tick(60)
    continue
  break

pygame.quit()
```



STEP 10 When you run this code it will display a long, narrow Pygame window with the intro text scrolling to the right. After a pause of ten seconds, it then launches the main game Python script where you can move the warrior sprite around. Overall the effect is quite good but there's always room for improvement.



Create Your Own Modules

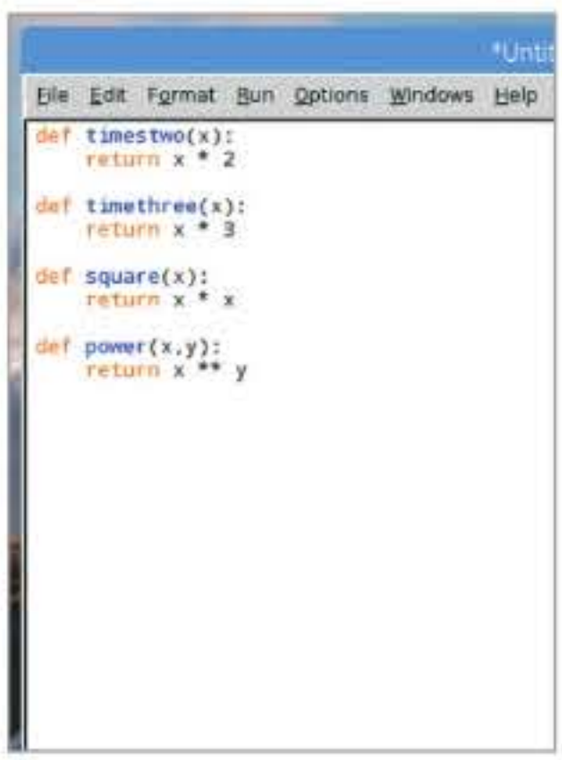
Large programs can be much easier to manage if you break them up into smaller parts and import the parts you need as modules. Learning to build your own modules also makes it easier to understand how they work.

BUILDING MODULES

Modules are Python files, containing code, that you save using a .py extension. These are then imported into Python using the now familiar import command.

STEP 1 Let's start by creating a set of basic Mathematics functions. Multiply a number by two, three and square or raise a number to an exponent (power). Create a New File in the IDLE and enter:

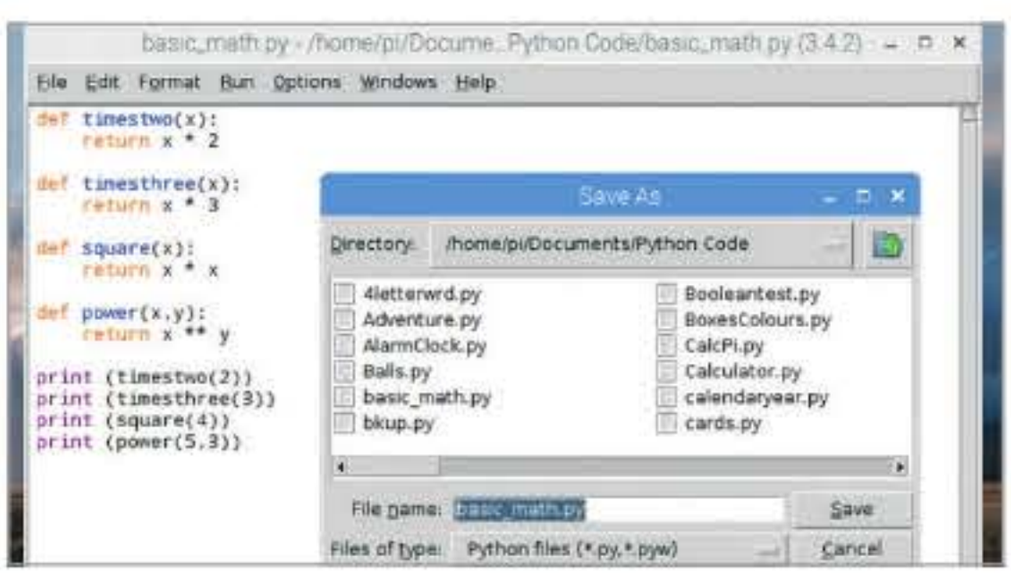
```
def timestwo(x):
        return x * 2

def timesthree(x):
        return x * 3

def square(x):
        return x * x

def power(x,y):
        return x ** y
```



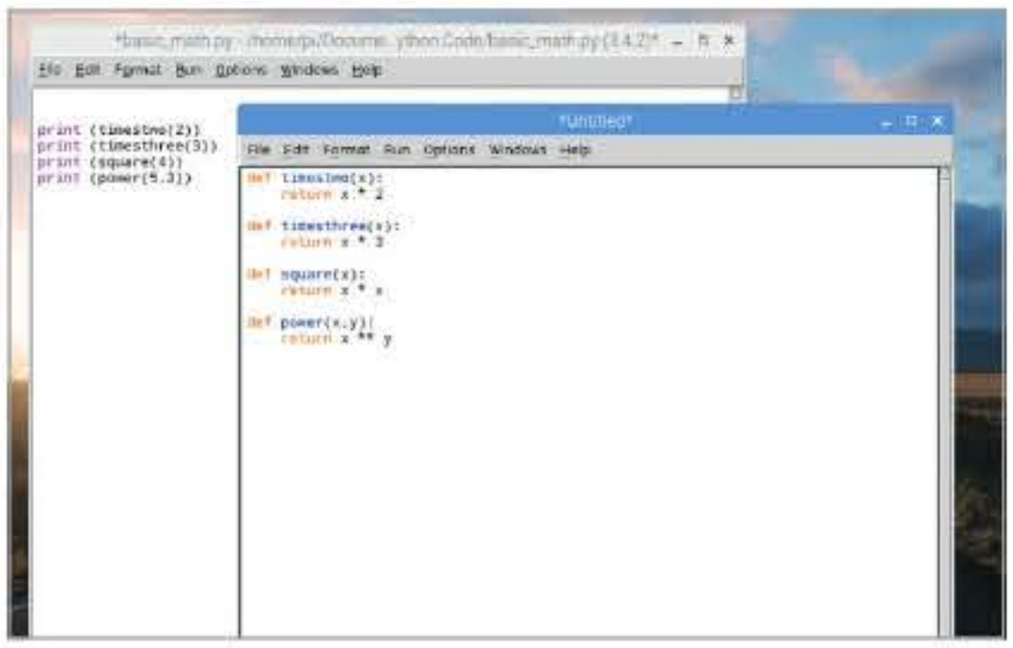
STEP 2 Under the above code, enter functions to call the code:

```
print (timestwo(2))
print (timesthree(3))
print (square(4))
print (power(5,3))
```

Save the program as basic_math.py and execute it to get the results.



STEP 3 Now you're going to take the function definitions out of the program and into a separate file. Highlight the function definitions and choose Edit > Cut. Choose File > New File and use Edit > Paste in the new window. You now have two separate files, one with the function definitions, the other with the function calls.



STEP 4 If you now try and execute the basic_math.py code again, the error **'NameError: name 'timestwo' is not defined'** will be displayed. This is due to the code no longer having access to the function definitions.

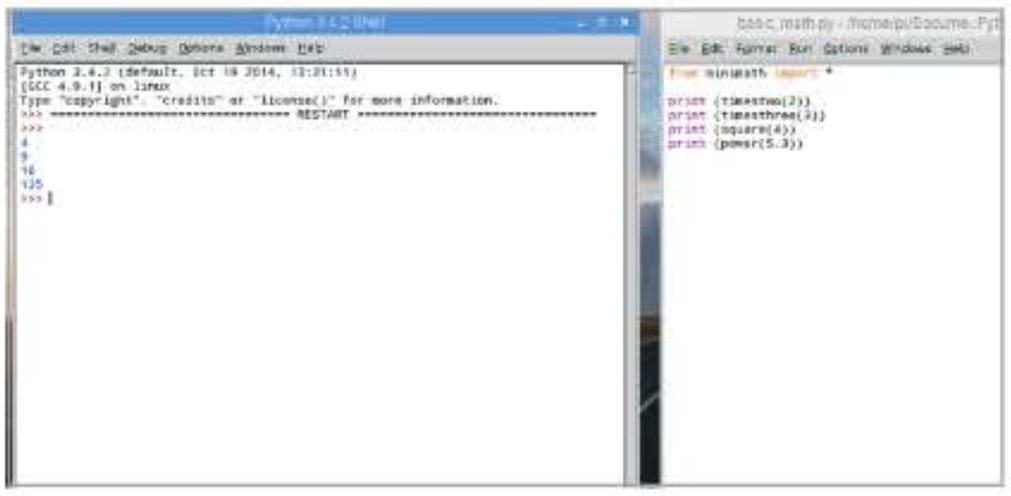
```
Traceback (most recent call last):
  File "/home/pi/Documents/Python Code/basic_math.py", line 3, in <module>
    print (timestwo(2))
NameError: name 'timestwo' is not defined
>>>
```

STEP 5 Return to the newly created window containing the function definitions, and click File > Save As. Name this **minimath.py** and save it in the same location as the original **basic_math.py** program. Now close the minimath.py window, so the basic_math.py window is left open.

STEP 6 Back to the basic_math.py window: at the top of the code enter:

```
from minimath import *
```

This will import the function definitions as a module. Press F5 to save and execute the program to see it in action.



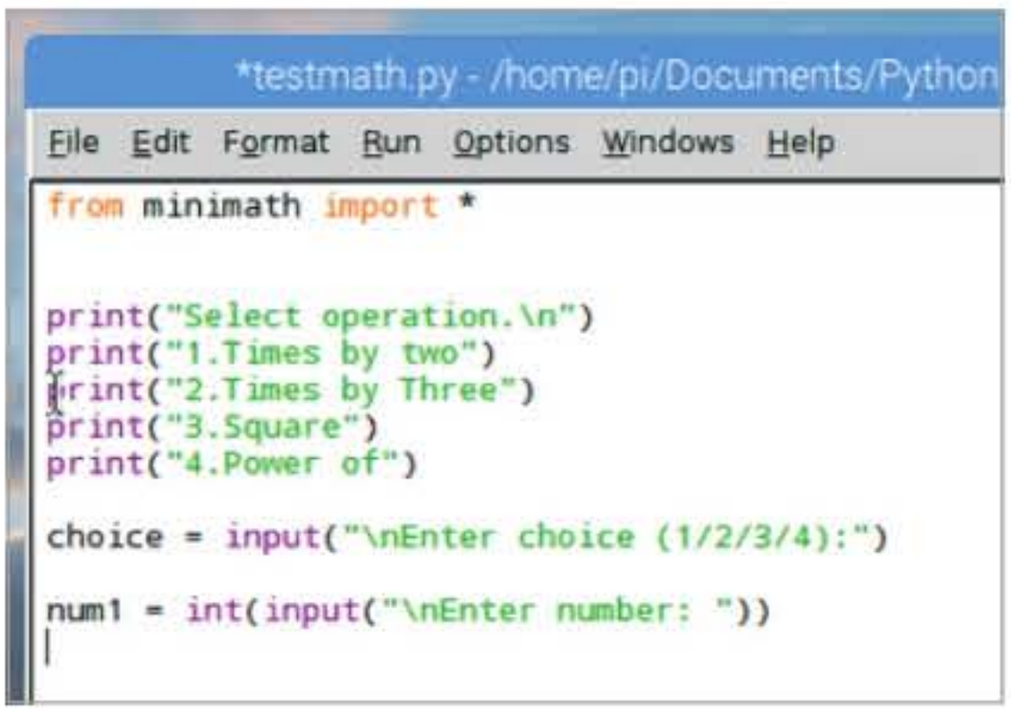
STEP 7 You can now use the code further to make the program a little more advanced, utilising the newly created module to its full. Include some user interaction. Start by creating a basic menu the user can choose from:

```
print("Select operation.\n")
print("1.Times by two")
print("2.Times by Three")
print("3.Square")
print("4.Power of")

choice = input("\nEnter choice (1/2/3/4):")
```

STEP 8 Now we can add the user input to get the number the code will work on:

```
num1 = int(input("\nEnter number: "))
```

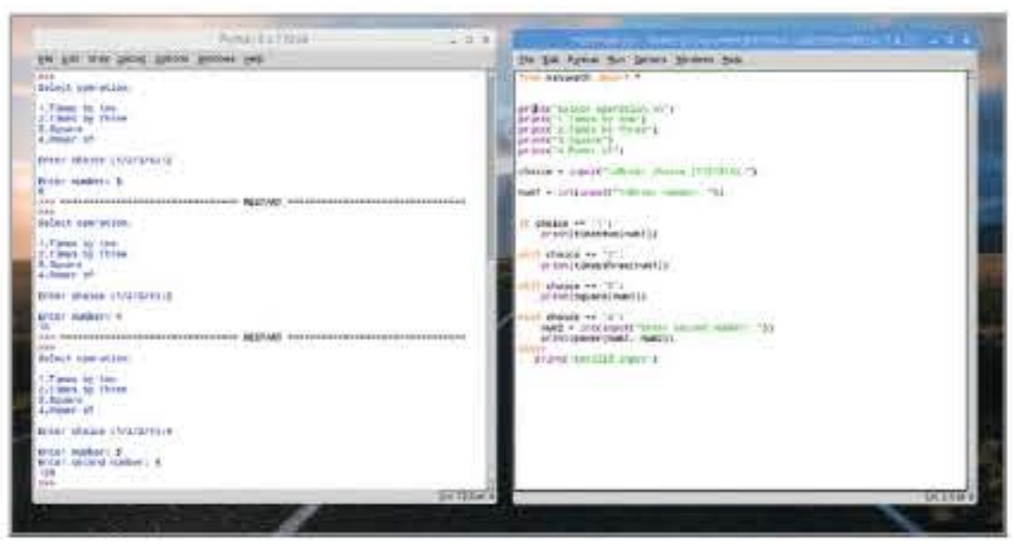
This will save the user-entered number as the variable num1.



STEP 9 Finally, you can now create a range of if statements to determine what to do with the number and utilise the newly created function definitions:

```
if choice == '1':
    print(timestwo(num1))

elif choice == '2':
    print(timesthree(num1))

elif choice == '3':
    print(square(num1))

elif choice == '4':
    num2 = int(input("Enter second number: "))
    print(power(num1, num2))
else:
   print("Invalid input")
```

STEP 10 Note that for the last available options, the Power of choice, we've added a second variable, num2. This passes a second number through the function definition called power. Save and execute the program to see it in action.



Code Repository

BDM's Code Portal

21 Python programs | 3,000+ lines of code

Whilst you can learn a lot by manually typing in the code within this section, we appreciate that it could be quite a long and drawn-out process. To that end, we have created the BDM Code Portal.

By navigating to **http:// bdmpublications.com/code-portal** you will be able to download each of the programs listed in the book's Code Repository in their original Python format. You can then edit them, take snippets of code from each to plug into your own programs or improve and expand on them as you see fit.

The file structure replicates the contents of this book, so you can easily skip to whichever code snippet you want to download and play around with it. Have fun, and keep coding.

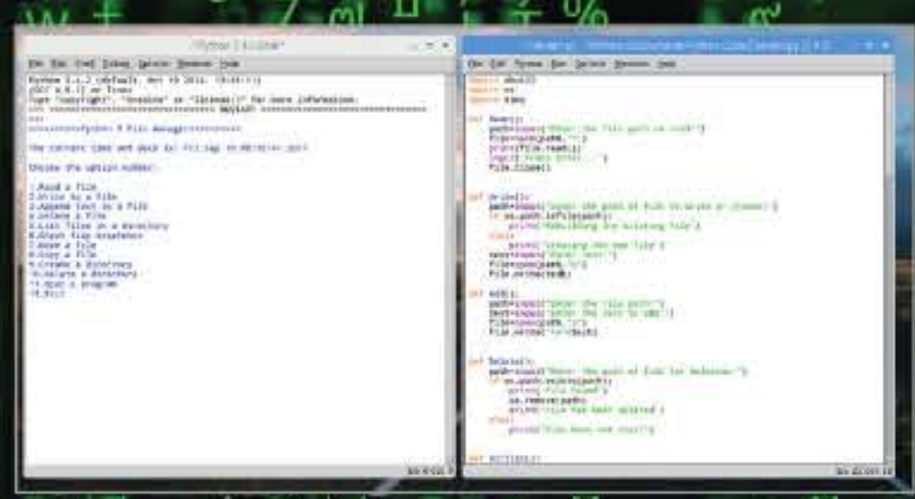
We've included a vast Python code repository for you to use freely in your own programs. There's everything you can think of in here to help you create a superb piece of programming.

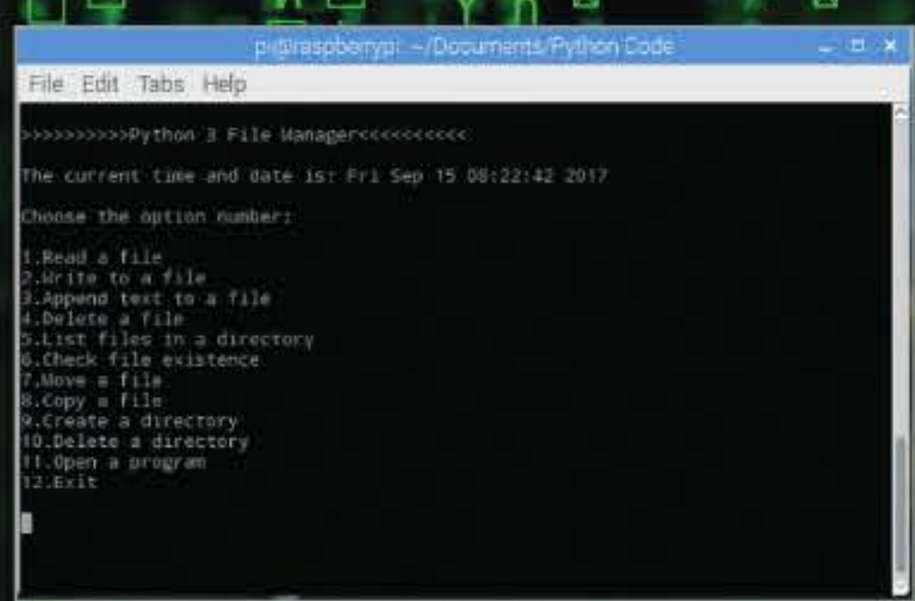
There's code for making backups of your files and folders, number guessing games, random number generators, Google search code, game code, animation code, graphics code, text adventure code and even code that plays music stored on your computer.

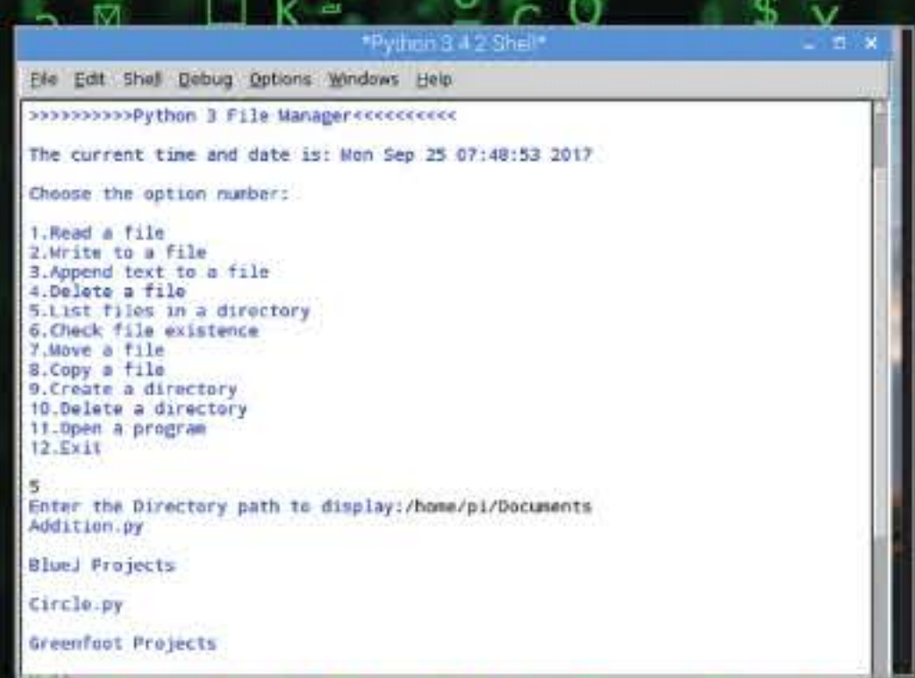
92	Python File Manager
94	Number Guessing Game
96	Polygon Circles
97	Random Number Generator
98	Random Password Generator
99	Keyboard Drawing Script
100	Pygame Text Examples
101	Google Search Script
102	Text to Binary Convertor
104	Basic GUI File Browser
106	Mouse Controlled Turtle
107	Python Alarm Clock
108	Vertically Scrolling Text
110	Python Digital Clock
112	Pygame Music Player
113	Python Image Slideshow Script
114	Playing music with the Winsound Module
116	Text Adventure Script
118	Python Scrolling Ticker Script
119	Simple Python Calculator
120	Hangman Game Script

Python File Manager

This file manager program displays a list of options that allow you to read a file, write to a file, append to a file, delete a file, list the contents of a directory and much more. It's remarkably easy to edit and insert into your own code, or add to.





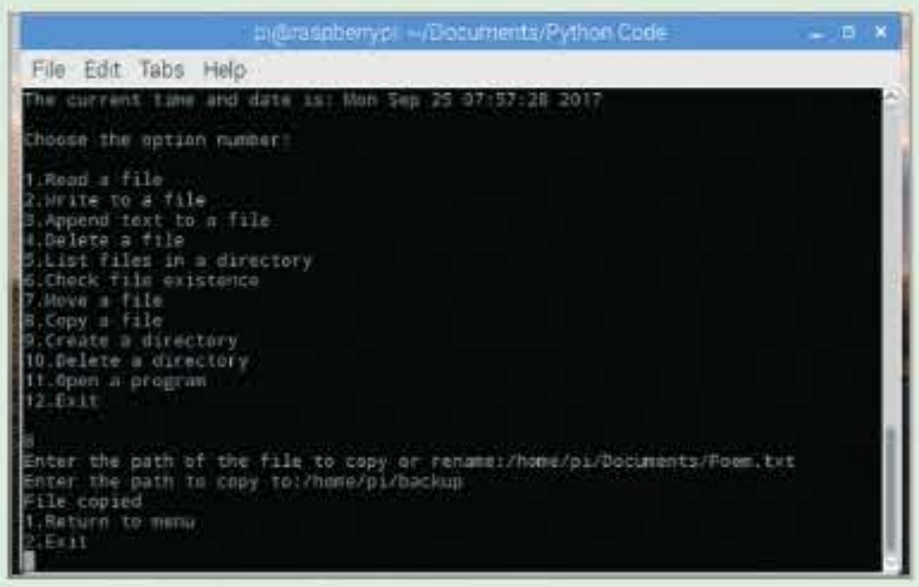


FILEMAN.PY

Copy the code below into a New > File and save it as FileMan.py. Once executed it will display the program title, along with the current time and date and the available options.

```
import shutil
import os
import time
import subprocess

def Read():
   path=input("Enter the file path to read:")
   file=open(path,"r")
   print(file.read())
   input('Press Enter...')
   file.close()

def Write():
   path=input("Enter the path of file to write or create:")
   if os.path.isfile(path):
      print('Rebuilding the existing file')
   else:
      print('Creating the new file')
   text=input("Enter text:")
   file=open(path,"w")
   file.write(text)

def Add():
   path=input("Enter the file path:")
   text=input("Enter the text to add:")
   file=open(path,"a")
   file.write('\n'+text)

def Delete():
   path=input("Enter the path of file for deletion:")
   if os.path.exists(path):
      print('File Found')
      os.remove(path)
      print('File has been deleted')
   else:
      print('File Does not exist')

def Dirlist():
   path=input("Enter the Directory path to display:")
   sortlist=sorted(os.listdir(path))
   i=0
   while(i<len(sortlist)):
      print(sortlist[i]+'\n')
      i+=1

def Check():
   fp=int(input('Check existence of \n1.File \n2.
   Directory\n'))
      if fp==1:
         path=input("Enter the file path:")
         os.path.isfile(path)
```

```
        if os.path.isfile(path)==True:
            print('File Found')
        else:
            print('File not found')
    if fp==2:
        path=input("Enter the directory path:")
        os.path.isdir(path)
        if os.path.isdir(path)==False:
            print('Directory Found')
        else:
            print('Directory Not Found')

def Move():
    path1=input('Enter the source path of file to move:')
    mr=int(input('1.Rename \n2.Move \n'))
    if mr==1:
        path2=input('Enter the destination path and file name:')
        shutil.move(path1,path2)
        print('File renamed')
    if mr==2:
        path2=input('Enter the path to move:')
        shutil.move(path1,path2)
        print('File moved')

def Copy():
    path1=input('Enter the path of the file to copy or rename:')
    path2=input('Enter the path to copy to:')
    shutil.copy(path1,path2)
    print('File copied')

def Makedir():
    path=input("Enter the directory name with path to make
    \neg. C:\\Hello\\Newdir \nWhere Newdir is new
    directory:")
    os.makedirs(path)
    print('Directory Created')

def Removedir():
    path=input('Enter the path of Directory:')
    treedir=int(input('1.Deleted Directory \n2.Delete
    Directory Tree \n3.Exit \n'))
    if treedir==1:
        os.rmdir(path)
    if treedir==2:
        shutil.rmtree(path)
        print('Directory Deleted')
    if treedir==3:
        exit()

def Openfile():
    path=input('Enter the path of program:')
    try:
        os.startfile(path)
    except:
        print('File not found')

run=1
while(run==1):
    try:
        os.system('clear')
    except OSError:
        os.system('cls')
    print('\n>>>>>>>>>>Python 3 File Manager<<<<<<<<<<\n')
    print('The current time and date is:',time.asctime())
    print('\nChoose the option number: \n')
    dec=int(input('''1.Read a file
2.Write to a file
3.Append text to a file
4.Delete a file
5.List files in a directory
6.Check file existence
7.Move a file
8.Copy a file
9.Create a directory
10.Delete a directory
11.Open a program
12.Exit

'''))
    if dec==1:
        Read()
    if dec==2:
        Write()
    if dec==3:
        Add()
    if dec==4:
        Delete()
    if dec==5:
        Dirlist()
    if dec==6:
        Check()
    if dec==7:
        Move()
    if dec==8:
        Copy()
    if dec==9:
        Makedir()
    if dec==10:
        Removedir()
    if dec==11:
        Openfile()
    if dec==12:
        exit()
    run=int(input("1.Return to menu\n2.Exit \n"))
    if run==2:
        exit()
```



Imports

There are three modules to import here: Shutil, OS and Time. The first two deal with the operating system and file management and manipulation; and the Time module simply displays the current time and date.

Note how we've included a try and except block to check if the user is running the code on a Linux system or Windows. Windows uses CLS to clear the screen, while Linux uses clear. The try block should work well enough but it's a point of possible improvement depending on your own system.

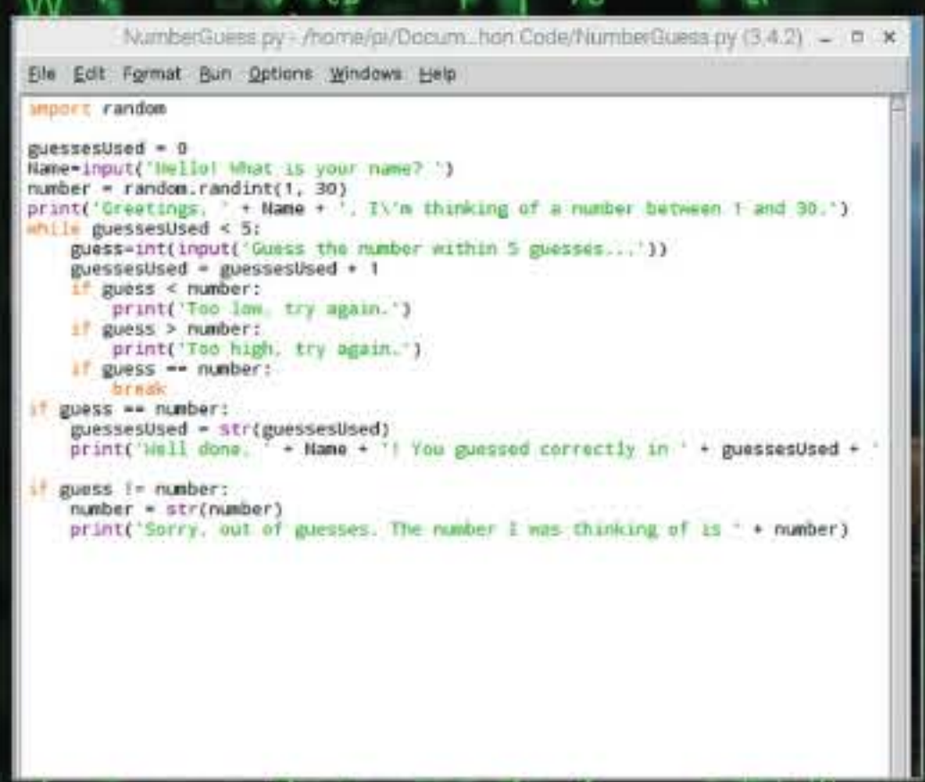
Number Guessing Game

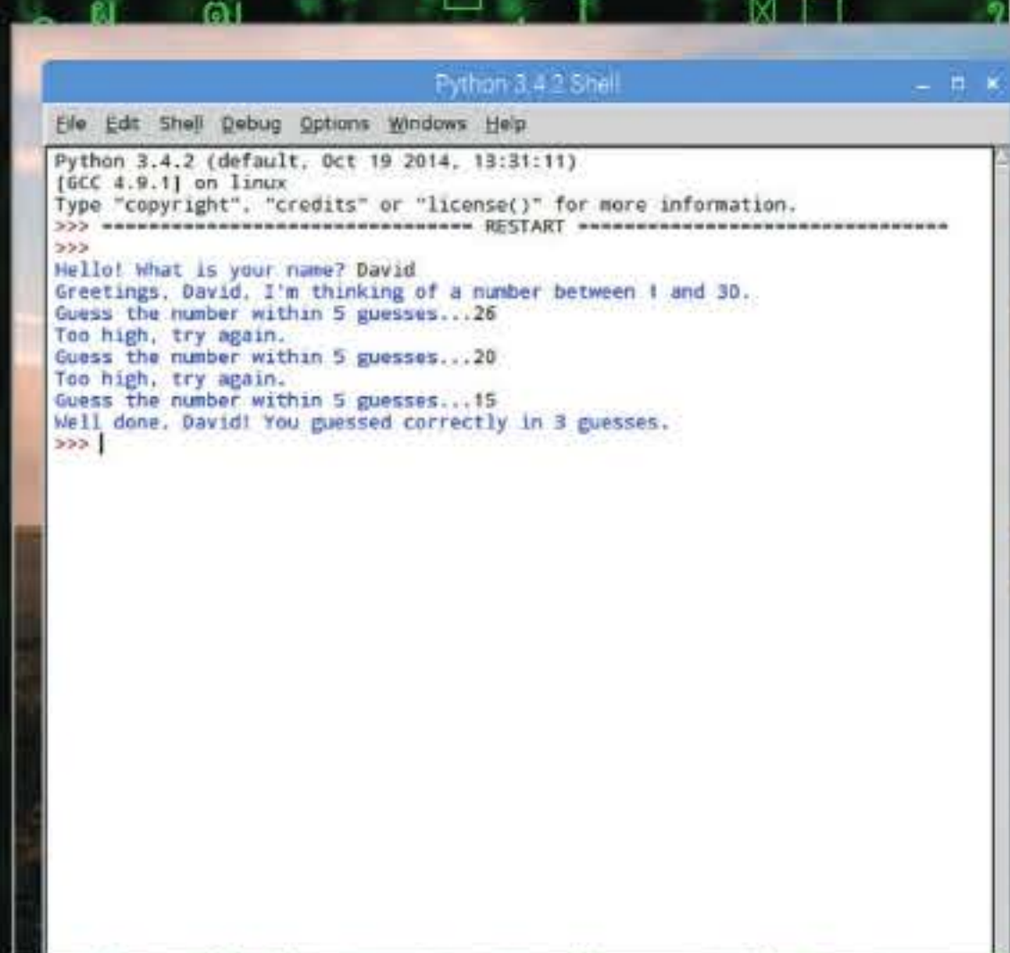
This is a simple little piece of code but it makes good use of the Random module, print and input, and a while loop. The number of guesses can be increased from 5 and the random number range can easily be altered too.

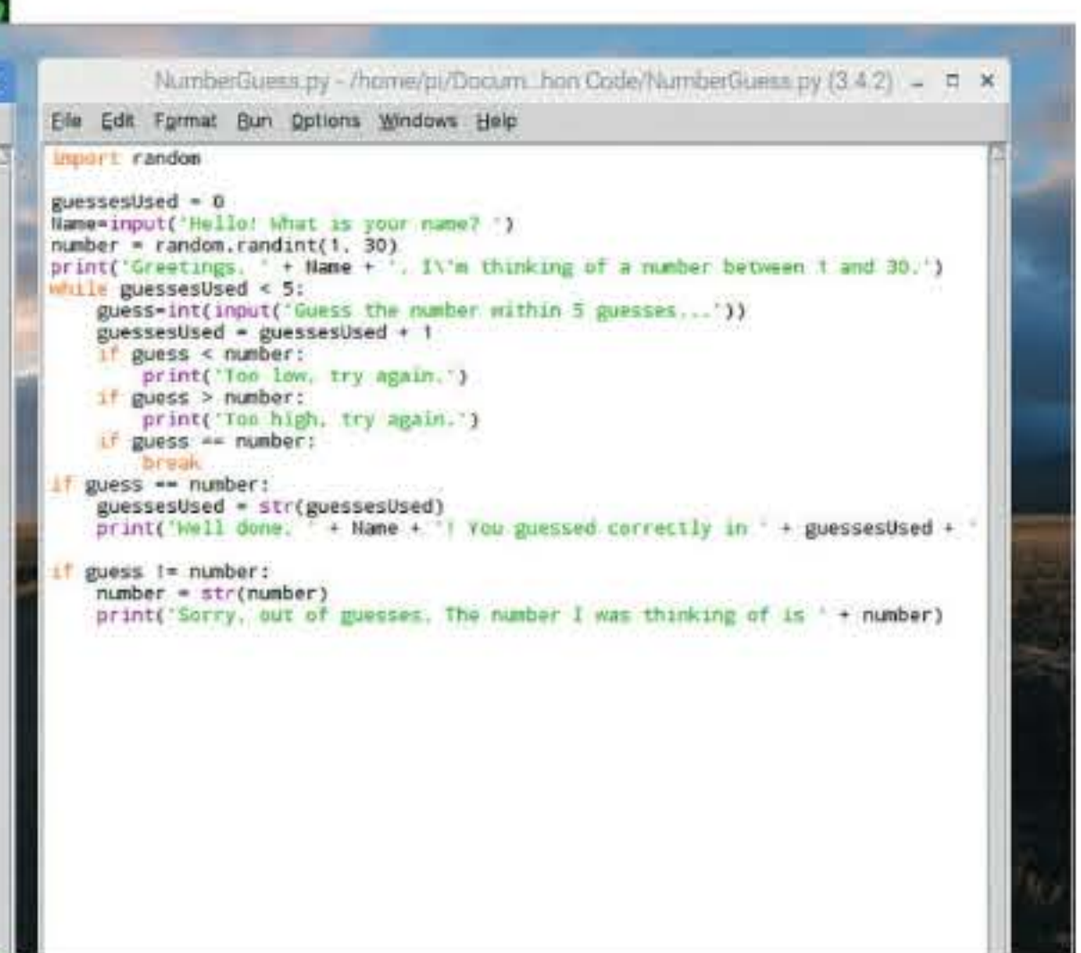
NUMBERGUESS.PY

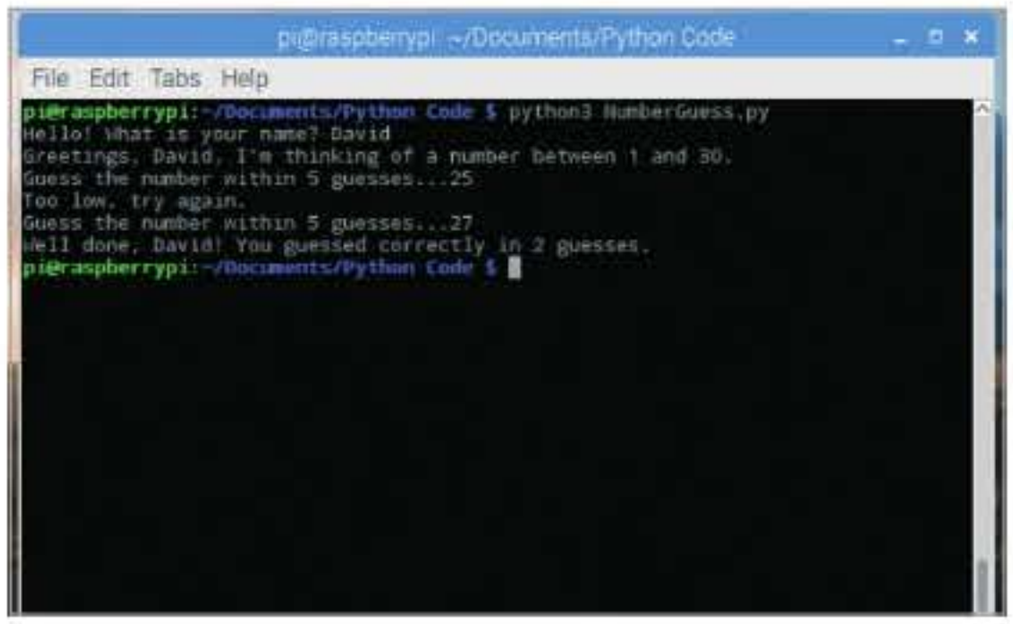
Copy the code and see if you can beat the computer within five guesses. It's an interesting bit of code that can be quite handy when your implementing a combination of the Random module alongside a while loop.

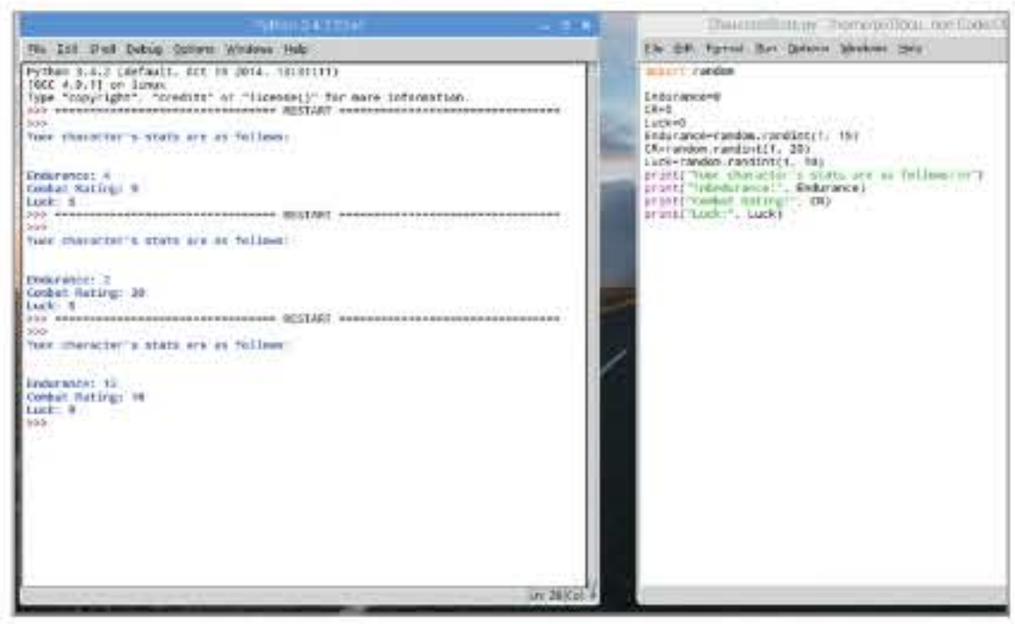
```
import random

guessesUsed = 0
Name=input('Hello! What is your name? ')
number = random.randint(1, 30)
print('Greetings, ' + Name + ', I\'m thinking of a
number between 1 and 30.')
while guessesUsed < 5:
   guess=int(input('Guess the number within 5 guesses...'))
   guessesUsed = guessesUsed + 1
   if guess < number:
      print('Too low, try again.')
   if guess > number:
      print('Too high, try again.')
   if guess == number:
      break
if guess == number:
   guessesUsed = str(guessesUsed)
   print('Well done, ' + Name + '! You guessed
   correctly in ' + guessesUsed + ' guesses.')

if guess != number:
   number = str(number)
   print('Sorry, out of guesses. The number I was
   thinking of is ' + number)
```











Code Improvements

Since this is such as simple script to apply to a situation, there's plenty of room to mess around with it and make it more interesting. Perhaps you can include an option to take score, the best out of three rounds. Maybe an elaborate way to congratulate the player for getting a 'hole in one' correct guess on their first try.

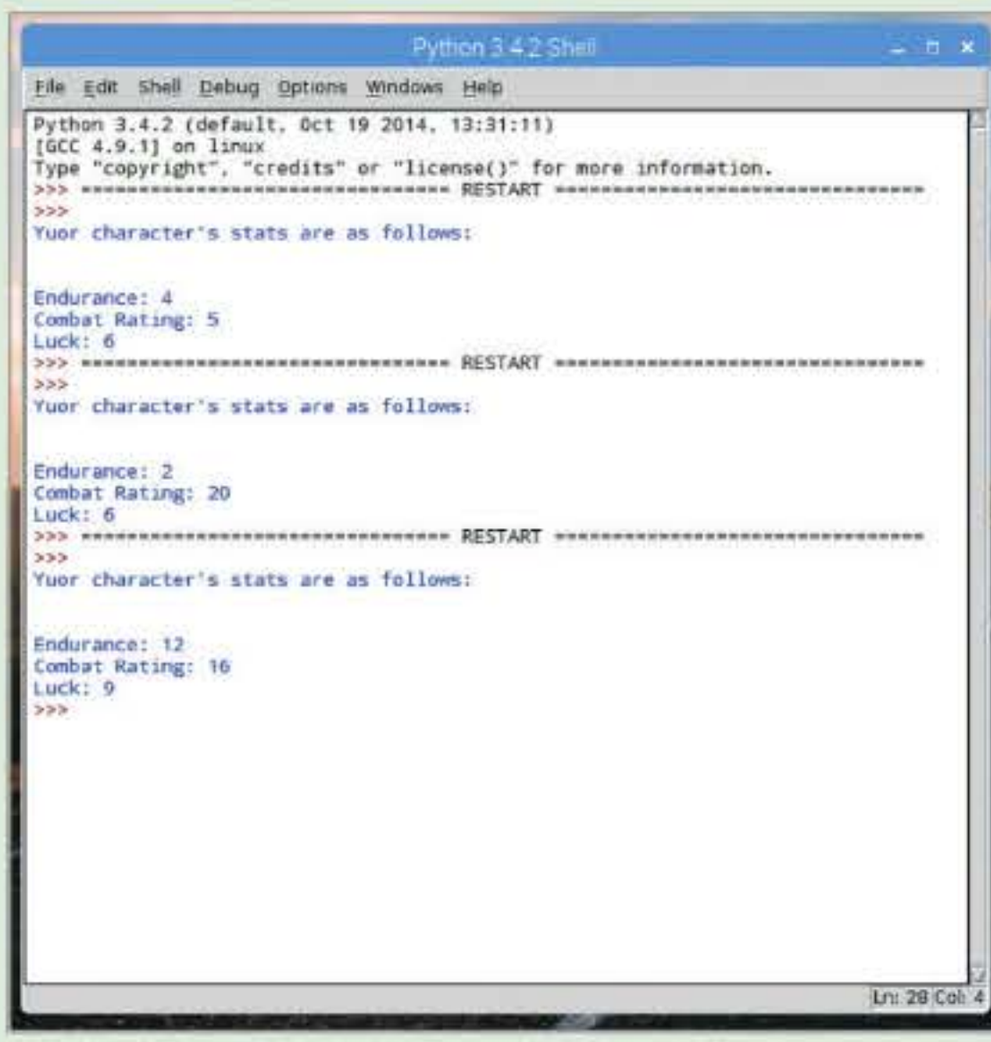
Moreover, the number guessing game code does offer some room for implementing into your code in a different manner. What we mean by this is, the code can be used to retrieve a random number between a range, which in turn can give you the start of a character creation defined function within an adventure game.

Imagine the start of a text adventure written in Python, where the player names their character. The next step is to roll the virtual random dice to decide what that character's combat rating, strength, endurance and luck values are. These can then be carried forward into the game under a set of variables that can be reduced or increased depending on the circumstances the player's character ends up in.

For example, as per the screenshot provided, you could use something along the lines of:

```
Endurance=0
CR=0
Luck=0
Endurance = random.randint(1, 15)
CR = random.randint(1, 20)
Luck = random.randint(1, 10)
Print("Your character's stats are as follows:\n")
Print("Endurance:", Endurance)
Print("Combat Rating:", CR)
Print("Luck:", Luck)
```

The player can then decide to either stick with their roll or try again for the hope of better values being picked. There's ample ways in which to implement this code into a basic adventure game.



Polygon Circles

Here's a fun, and mathematical, look at making a circle from straight lines. Using the Math module, in particular sin, cos and Pi, this code will draw a series of straight lines using the Turtle module. The end result is quite remarkable and has plenty of scope for further exploration.

POLYGONCIRCLES.PY

There's lots of Mathematics used here along with some intricate coordinate manipulation with the Turtle module. Enter the code and execute it to see how it turns out.

```
from turtle import*
from math import sin, cos, pi
r=200
inc=2*pi/100
t=0;n=1.5
for i in range(100):
   x1=r*sin(t);  y1=r*cos(t)
   x2=r*sin(t+n);y2=r*cos(t+n)
   penup();  goto(x1,y1)
   pendown();goto(x2,y2)
   t+=inc
```

Graphical Enhancements

There are several ways in which you can improve this code to make it more interesting. You can insert colours, perhaps a different colour for every line. You can display a message inside the circle and have the Turtle draw around it. Let your imagination run wild on this one.

Turtle's graphics can take a while to map out and draw, depending on how big and how intricate an image it is you're designing. Whilst the effect can be quite stunning, it is limited by the amount of time it takes to display an image. Therefore it's worth seeing if the function turtle.speed() will quicken things up.

Turtle.speed() comes in various values:

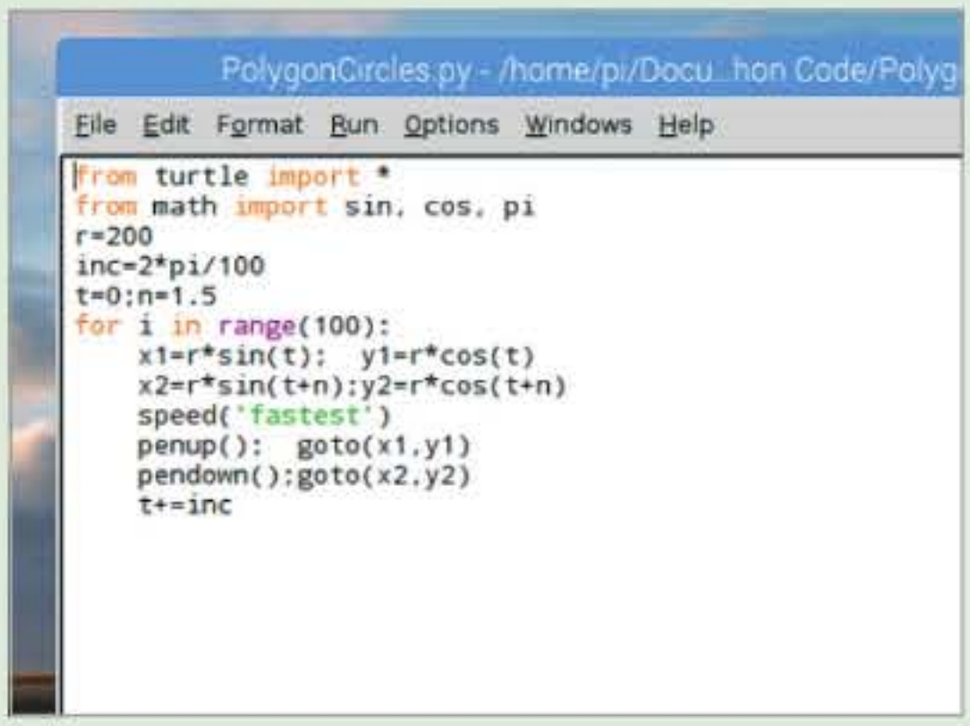
```
slowest
slow
normal
fast
fastest
```

You can experiment with the various speeds by adding the function in the for loop, just before the penup line.

For example:

```
for i in range(100):
   x1=r*sin(t);  y1=r*cos(t)
   x2=r*sin(t+n);y2=r*cos(t+n)
   speed('fastest')
   penup();  goto(x1,y1)
   pendown();goto(x2,y2)
   t+=inc
```

This will run through the code at the 'fastest' speed possible for the Turtle. It certainly makes a difference and is worth considering if you're drawing Turtle images for games or presentations.



Random Number Generator

User input and the ability to manipulate that input are important elements with any programming language. It's what separates a good program from a great program, one that allows the user to interact and see the results of that interaction.

RNDNUMGEN.PY

It might be simple but this little piece of code will ask the user for two sets of numbers, a start and a finish. The code will then pluck out a random number between the two sets and display it.

```
from random import *

print("\n>>>>>>>>>>Random Number Generator<<<<<<<<<<\n")
nmb1=int(input("Enter the start number: "))
nmb2=int(input("Enter the last number: "))

x = randint(nmb1, nmb2)
print("\nThe random number between",nmb1,"and",nmb2,"is:\n")
print(x)
```

More Input

While an easy code to follow, it could be more interesting if you prompt the user for more input. Perhaps you can provide them with addition, subtraction, multiplication elements with their numbers. If you're feeling clever, see if you can pass the code through a Tkinter window or even the Ticker window that's available on Page 124.

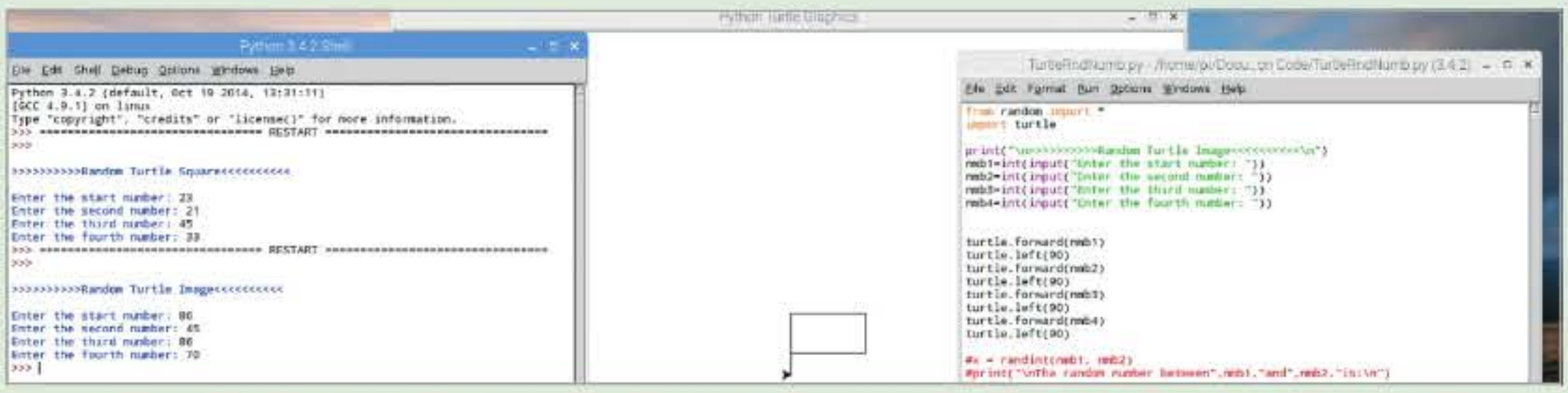
Furthermore, the core of the code can be used in a text adventure game, where the character fights something and their health, along with the enemy's, is reduced by a random number. This can be mixed with the previous code from Page 90's Number Guessing Game, where we defined the stats for the adventure game's character.

You can also introduce the Turtle module into the code and perhaps set some defined rules for drawing a shape, object or something based on a user inputted random value from a range of numbers. It takes a little working out but the effect is certainly really interesting.

For example, the code could be edited to this:

```
from random import *
import turtle

print("\n>>>>>>>>>>Random Turtle Image<<<<<<<<<<\n")
nmb1=int(input("Enter the start number: "))
nmb2=int(input("Enter the second number: "))
nmb3=int(input("Enter the third number: "))
nmb4=int(input("Enter the fourth number: "))


turtle.forward(nmb1)
turtle.left(90)
turtle.forward(nmb2)
turtle.left(90)
turtle.forward(nmb3)
turtle.left(90)
turtle.forward(nmb4)
turtle.left(90)
```

Whilst it's a little rough around the edges, you can easily make it more suitable.



Random Password Generator

We're always being told that our passwords aren't secure enough; well here's a solution for you to implement into your own future programs. The random password generator code below will create a 12-letter string of words (both cases) and numbers each time it's executed.

RNDPASSWORD.PY

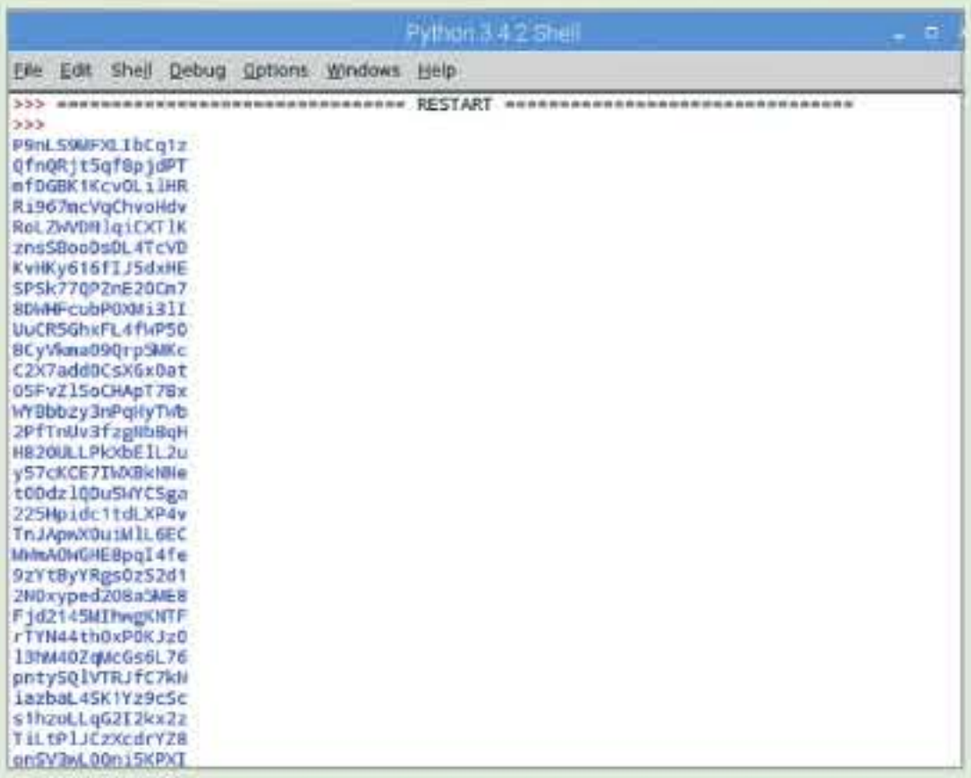
Copy the code and run it; each time you'll get a random string of characters that can easily be used as a secure password which will be incredibly difficult for a password cracker to hack.

```
import string
import random

def randompassword():
   chars=string.ascii _ uppercase + string.ascii _
   lowercase + string.digits
   size= 8
   return ''.join(random.choice(chars) for x in
   range(size,20))

print(randompassword())
```

Secure Passwords

There's plenty you can do to modify this code and improve it further. For one, you can increase the number of characters the generated password displays and perhaps you can include special characters too, such as signs and symbols. Then, you can output the chosen password to a file, then securely compress it using the previous random number generator as a file password and send it to a user for their new password.

An interesting aspect to this code is the ability to introduce a loop and print any number of random passwords. Let's assume you have a list of 50 users for a company and you're in charge of generating a random password for them each month.



Adding a loop to print a password fifty times is extremely easy, for example:

```
import string
import random


def randompassword():
   chars=string.ascii _ uppercase + string.ascii _
   lowercase + string.digits
   size= 4
   return ''.join(random.choice(chars) for x in
   range(size,20))

n=0
while n<50:
   print(randompassword())
   n=n+1
```

This will output fifty random passwords based on the previous random selection of characters.

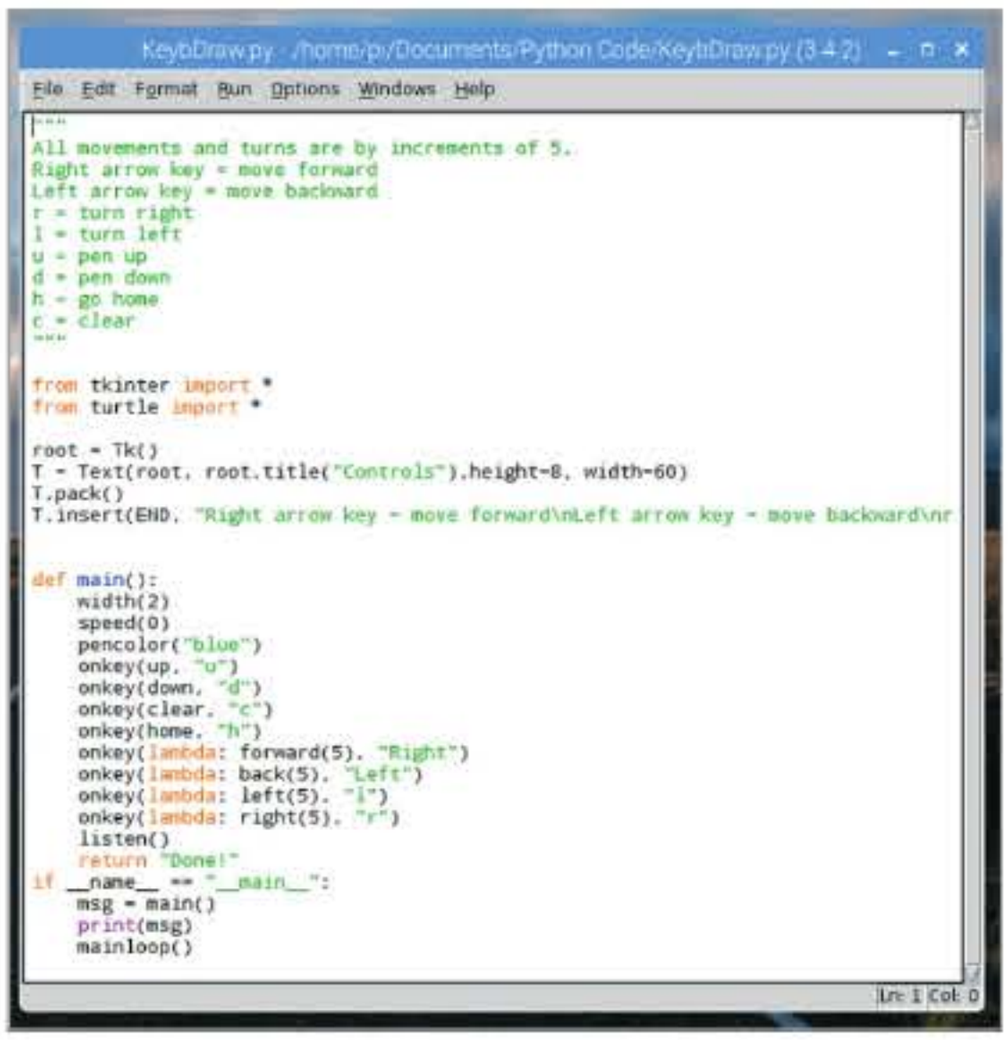
Keyboard Drawing Script

The Turtle module is an excellent resource for the Python programmer. However, what makes it more interesting, is its ability to enable the user to control the turtle on the screen. This piece of code does exactly that, allowing the user to unleash their inner artist.

KEYBDRAW.PY

There are two modules in this script: Turtle and Tkinter. The Turtle module is the main display, where the user controls the drawing, whereas Tkinter simply displays the user controls.

```
"""
All movements and turns are by increments of 5.
Right arrow key = move forward
Left arrow key = move backward
r = turn right
l = turn left
u = pen up
d = pen down
h = go home
c = clear
"""

from tkinter import *
from turtle import *

root = Tk()
T = Text(root, root title("Controls"),height=8, width=60)
T.pack()
T.insert(END, "Right arrow key = move forward\
nLeft arrow key = move backward\nr = turn right\nl
= turn left\nu = pen up\nd = pen down\nh = go home\
nc = clear")


def main():
   width(2)
   speed(0)
   pencolor("blue")
   onkey(up, "u")
   onkey(down, "d")
   onkey(clear, "c")
   onkey(home, "h")
   onkey(lambda: forward(5), "Right")
   onkey(lambda: back(5), "Left")
   onkey(lambda: left(5), "l")
   onkey(lambda: right(5), "r")
   listen()
   return "Done!"
if __name__ == "__main__":
   msg = main()
   print(msg)
   mainloop()
```



Artwork

Just as with all code, there's always room for improvement somewhere. Here you could change the colours or ask the user which colour they want to start with and then include a key in the controls to change the pen colour whilst drawing. There's also room to increase or decrease the speed of the pen, again that could be a user-defined speed. You can also expand the controls thoroughly to include a lot more detail and options.

Pygame Text Examples

There's a lot you can do using the Pygame module that's not related to displaying graphics. The module contains many functions which can manipulate text, such as flipping it, displaying it sideways, upside down and even rotating it with animations.

TXTROT.PY

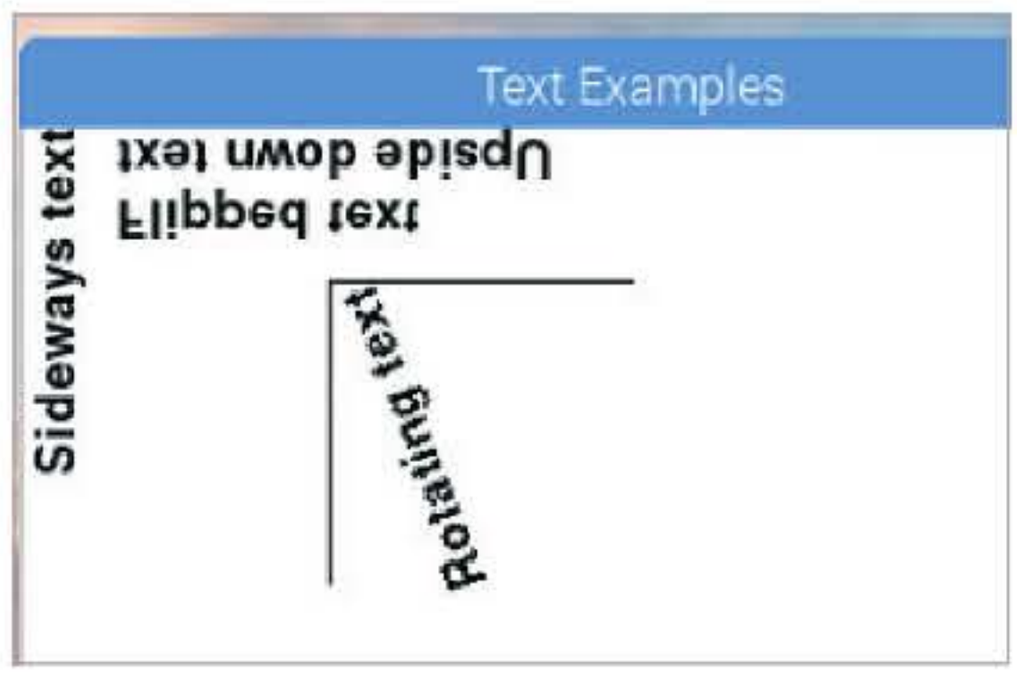
Here we've introduced several examples of displaying text within the Pygame module. Each is easily recognised within the code, so you can pull it out and use it in your code.

```
import pygame
pygame.init()

BLACK = (0, 0, 0)
WHITE = (255, 255, 255)
BLUE = (0, 0, 255)
GREEN = (0, 255, 0)
RED = (255, 0, 0)

PI = 3.141592653

size = (400, 500)
screen = pygame.display.set_mode(size)

pygame.display.set_caption("Text Examples")

done = False
clock = pygame.time.Clock()

text_rotate_degrees = 0

while not done:

   for event in pygame.event.get():
      if event.type == pygame.QUIT:
         done = True

   screen.fill(WHITE)

   pygame.draw.line(screen, BLACK, [100,50],
   [200, 50])
   pygame.draw.line(screen, BLACK, [100,50],
   [100, 150])

   font = pygame.font.SysFont('Calibri', 25,
   True, False)

   text = font.render("Sideways text", True, BLACK)
   text = pygame.transform.rotate(text, 90)
   screen.blit(text, [0, 0])

   text = font.render("Upside down text", True, BLACK)
   text = pygame.transform.rotate(text, 180)
   screen.blit(text, [30, 0])

   text = font.render("Flipped text", True, BLACK)
   text = pygame.transform.flip(text, False, True)
   screen.blit(text, [30, 20])

   text = font.render("Rotating text", True, BLACK)
   text = pygame.transform.rotate(text, text_
   rotate_degrees)
   text_rotate_degrees += 1
   screen.blit(text, [100, 50])

   pygame.display.flip()

   clock.tick(60)

pygame.quit()
```

The Joy Of Text

Here, you can see that we've started by defining the colours but left the text black throughout the rest of the code; then, also left the display window with a white background. The code itself is fairly easy to follow and you can improve it by using different colours, changing the font for each text example, and its size too.

Try using some of the text examples individually in your code as an introduction to your program perhaps. Either way you use it, it will help make it stand out a little more than the standard Python code you will likely come across during your time as a programmer.



Google Search Script

Using the OS and Urllib modules, this small snippet of code will display a window for the user's input, then pass the input to a Google search in their browser. It's really quite a handy script and one that's easily introduced into your own code.

GOOGLESEARCH.PY

You will need to pip install the urllib module, unless it's already installed. It also uses Zenity, which is a Linux-based (GNOME) tool for creating dialog boxes.

```
import os
import urllib.parse
google = os.popen('zenity --entry --text="Enter your
Google search: " --title="Google Search"').read()
google = urllib.parse.quote(google)
os.system('chromium-browser http://www.google.com/
search?q=%s' % (google))
```

Searching For More

Here we've used the Zenity command to create the dialog box, which as we mentioned is only available to Linux machines (such as Ubuntu, Raspberry Pi, Linux Mint and so on). If you want to execute it in Windows you have a couple of possibilities: you can find a Windows version of Zenity and pass the user's query through it; or you can create a Tkinter dialog box to pass the information.

You can see that this particular code uses the Chromium browser which comes preinstalled on the Raspberry Pi, and some versions of Linux. To use your favourite browser in Windows, for example, you will need to change the command in the last line of the code to read Firefox, or whatever you use, together with the Start command. So essentially, one of the following:

```
os.system('start firefox http://www.google.com/
search?q=%s' % (google))

os.system('start chrome http://www.google.com/
search?q=%s' % (google))

os.system('start iexplore http://www.google.com/
search?q=%s' % (google))
```

The last two for Chrome and Internet Explorer respectively.

There is a Zenity for Windows project available on GitHub at www.github.com/kvaps/zenity-windows. It's a good working version but you do need to install it to a folder on your system where you won't require administrator access to be able to run the Zenity program. When you have Zenity installed, you can modify the Windows version of this code in Python to read:

```
import os
import urllib.parse

google = os.popen('start c:\Temp\Zenity\\bin\
zenity --entry --text="Enter your Google search: "
--title="Google Search"').read()
google = urllib.parse.quote(google)
os.system('start firefox http://www.google.com/
search?q=%s' % (google))
```

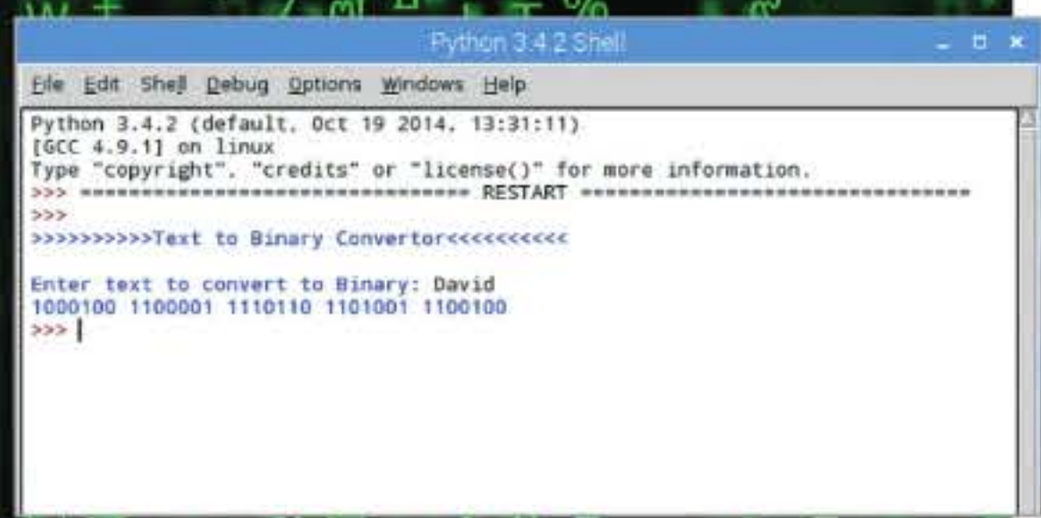
Text to Binary Convertor

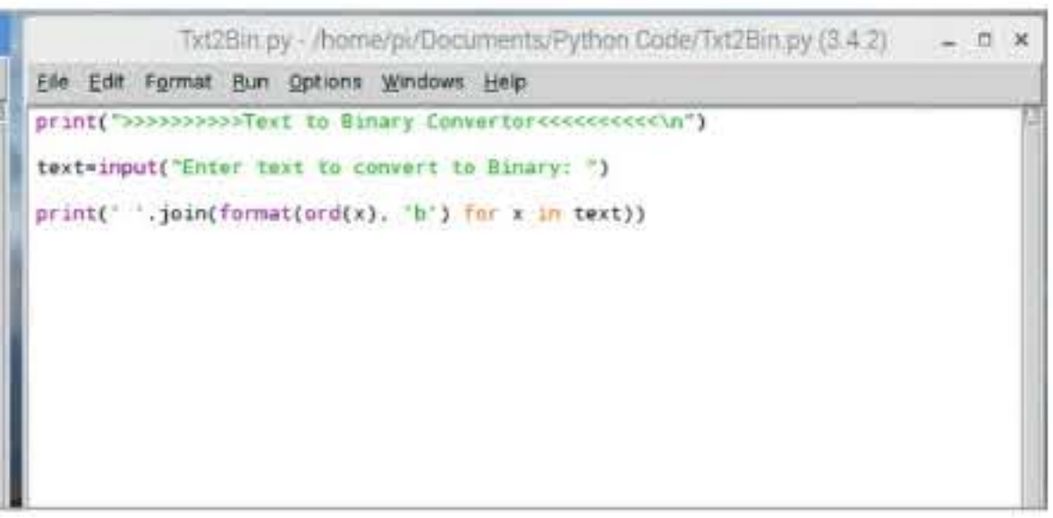
Whilst it may not seem too exciting, this text to binary convertor is actually quite good fun. It also only uses two lines of code, so it's extremely easy to insert into your own script.

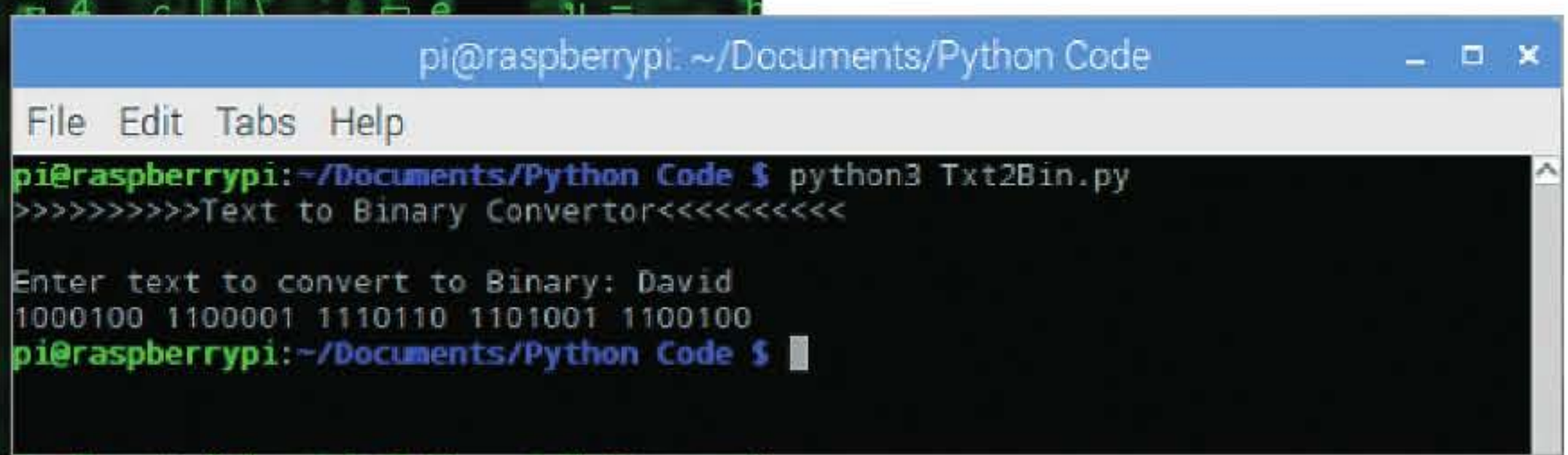
TXT2BIN.PY

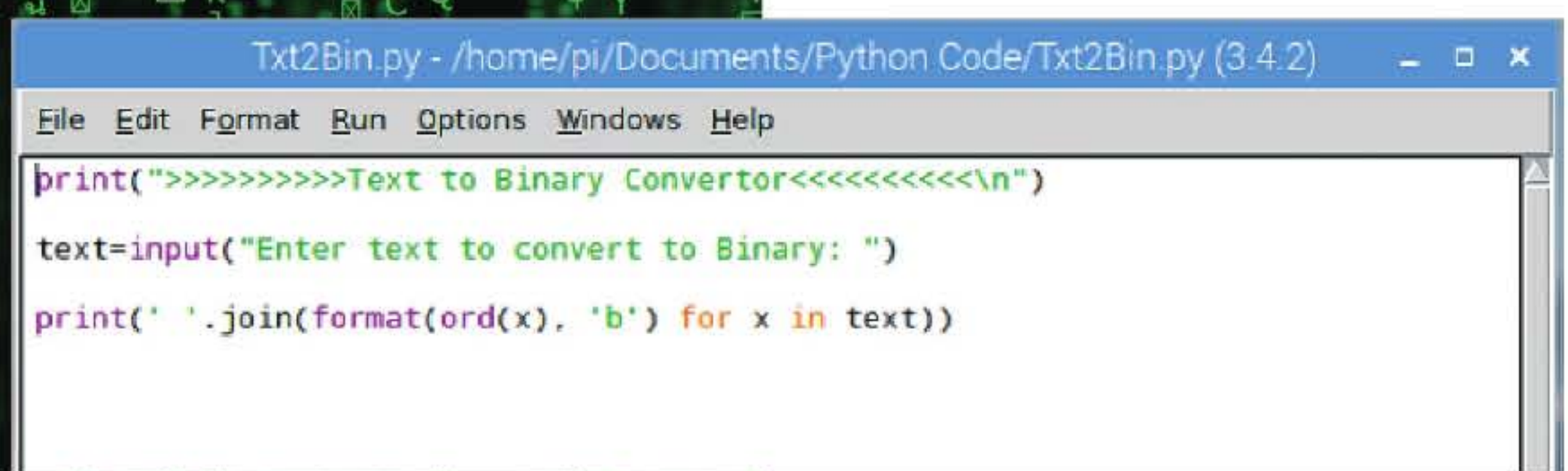
Naturally we're using the format function to convert the user's entered text string into its binary equivalent. If you want to check its accuracy, you can plug the binary into an online convertor.

```
text=input("Enter text to convert to Binary: ")

print(' '.join(format(ord(x), 'b') for x in text))
```









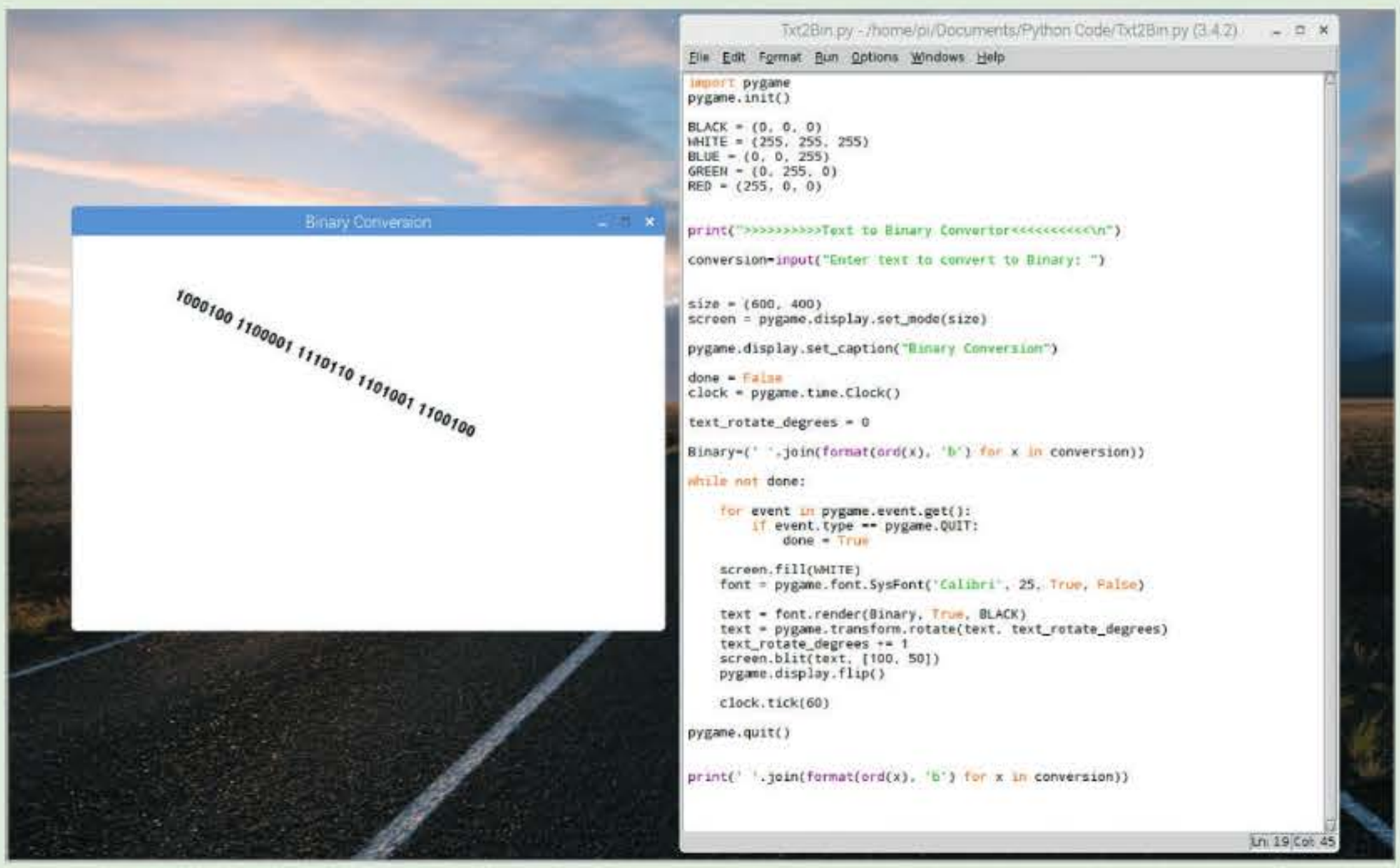
1000010 1101001 1101110 1100001 1110010 1111001

The text to binary convertor does offer some room for improvement and enhancement. There are many uses: it could be utilised in a password or secret word script, as part of an adventure game or just a novel way to display someone's name.

With regards to improvements, you could display the binary conversion in a Pygame window, using the animated text options from page 100. You could also ask the user if they wanted to have another go, or even ask if they wanted the binary output to be saved to a file.

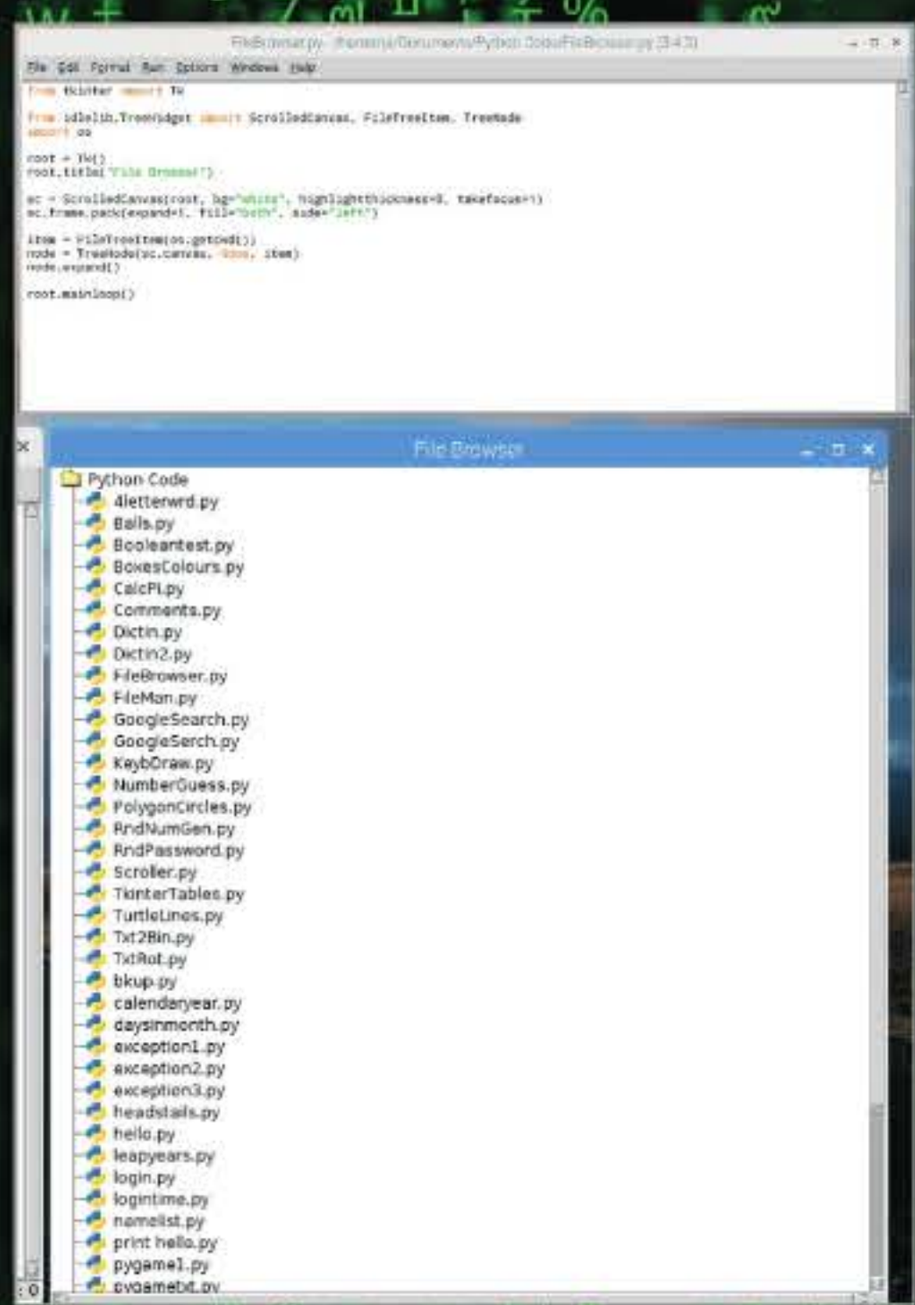
With regards to rendering the outputted binary conversion to a Pygame window, complete with rotating text, you can use:

```
import pygame
pygame.init()

BLACK = (0, 0, 0)
WHITE = (255, 255, 255)
BLUE = (0, 0, 255)
GREEN = (0, 255, 0)
RED = (255, 0, 0)

print(">>>>>>>>>>Text to Binary Convertor<<<<<<<<<<\n")

conversion=input("Enter text to convert to Binary: ")

size = (600, 400)
screen = pygame.display.set_mode(size)

pygame.display.set_caption("Binary Conversion")

done = False
clock = pygame.time.Clock()

text_rotate_degrees = 0

Binary=(' '.join(format(ord(x), 'b') for x in conversion))

while not done:

   for event in pygame.event.get():
       if event.type == pygame.QUIT:
           done = True

   screen.fill(WHITE)
   font = pygame.font.SysFont('Calibri', 25, True, False)

   text = font.render(Binary, True, BLACK)
   text = pygame.transform.rotate(text, text_
   rotate_degrees)
   text_rotate_degrees += 1
   screen.blit(text, [100, 50])
   pygame.display.flip()

   clock.tick(60)

pygame.quit()

print(' '.join(format(ord(x), 'b') for x in conversion))
```



Basic GUI File Browser

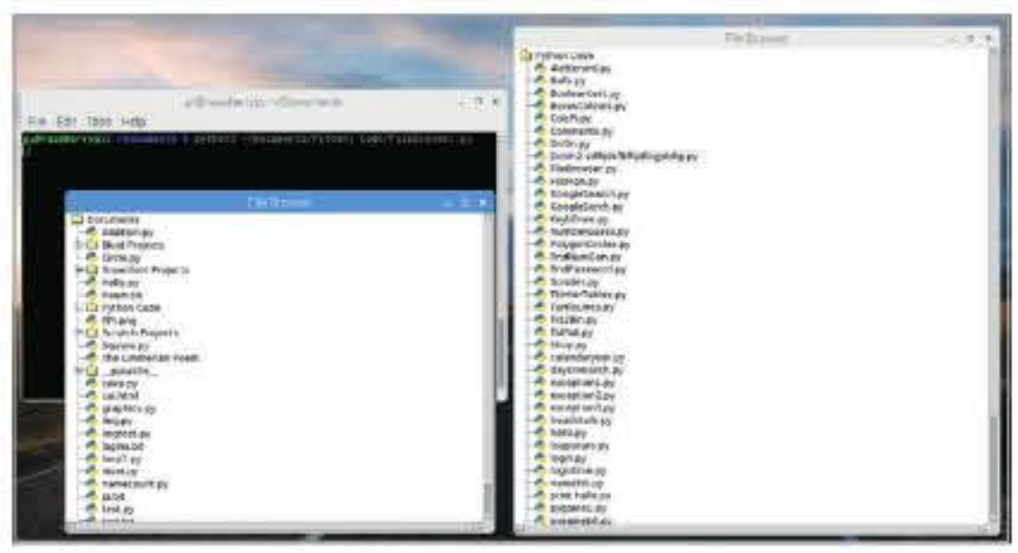
Here's a helpful and interesting piece of code. It's an extremely basic file browser that's presented in a graphical user interface using the Tkinter module. There's a lot you can learn from this code and implement into your own programs.

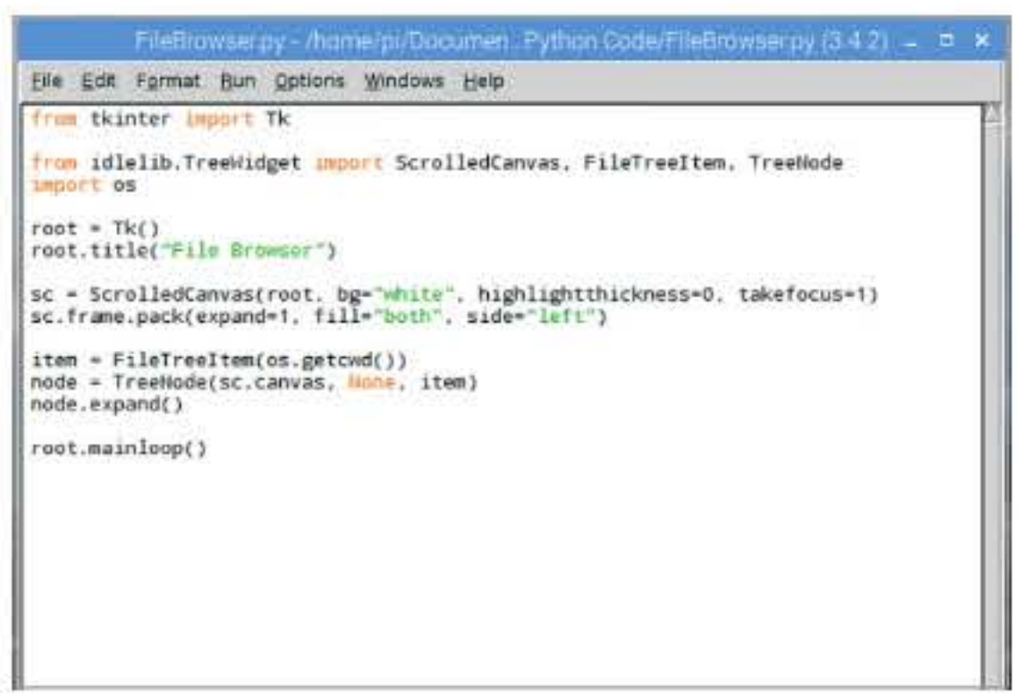


FILEBROWSER.PY

Tkinter is the main module in use here but we're also using idlelib, so you may need to pip install any extras if the dependencies fail when you execute the code.

```
from tkinter import Tk

from idlelib.TreeWidget import ScrolledCanvas,
FileTreeItem, TreeNode
import os

root = Tk()
root.title("File Browser")

sc = ScrolledCanvas(root, bg="white",
highlightthickness=0, takefocus=1)
sc.frame.pack(expand=1, fill="both", side="left")

item = FileTreeItem(os.getcwd())
node = TreeNode(sc.canvas, None, item)
node.expand()

root.mainloop()
```





Advanced Filing

When executed, the code will display the current directory's contents. If you want to see the contents of another directory, you can run the code from a command line within the chosen directory; just remember to call the code from where it's located on your system, as per the second screenshot. You can also double-click any of the file names shown in the directory tree and rename them.

This is an interesting piece of code and one that you can insert into your own programs. You can extend the code to include a user specified directory to browse, perhaps your own unique file icons too. If you're using Linux, create an alias to execute the code and then you can run it from wherever you are in the system.

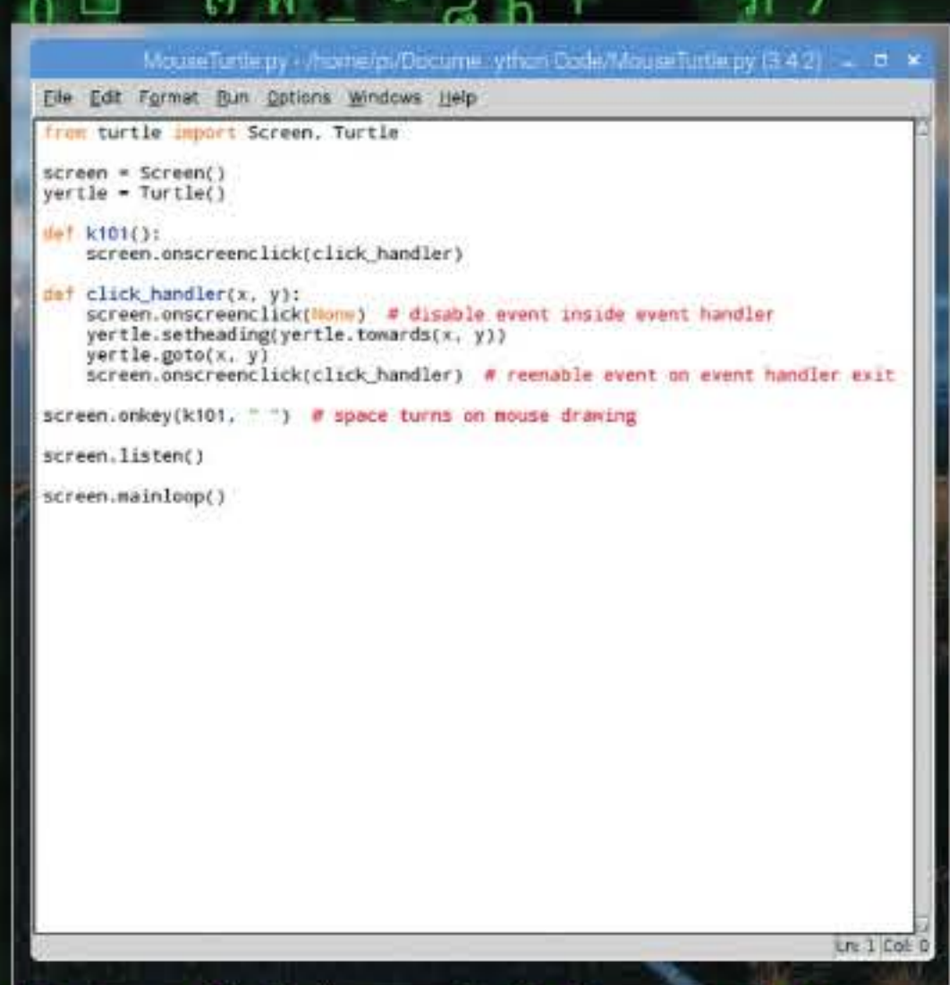
Windows users may have some trouble with the above code, an alternative can be achieved by using the following:

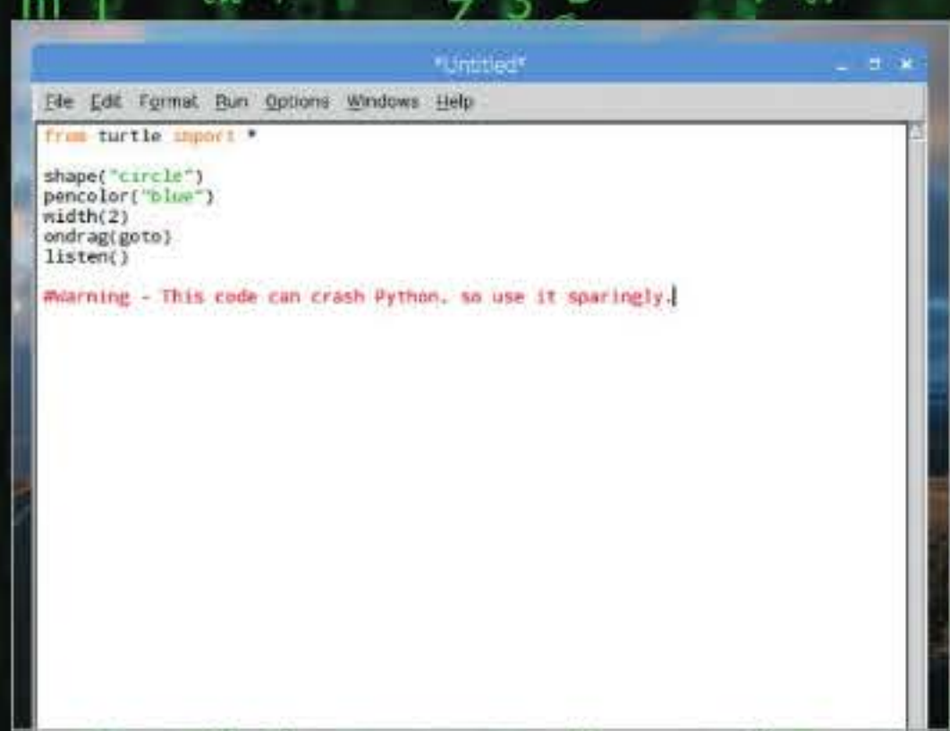
```
from tkinter import *
from tkinter import ttk
from tkinter.filedialog import askopenfilename

root = Tk(  )

def OpenFile():
   name = askopenfilename(initialdir="C:/",
                  filetypes =(("Text File", "*.txt"),("All
                  Files","*.*")),
                  title = "Choose a file."
                              )
   print (name)
   try:
      with open(name,'r') as UseFile:
         print(UseFile.read())
   except:
      print("No files opened")


Title = root.title( "File Opener")
label = ttk.Label(root, text ="File
Open",foreground="red",font=("Helvetica", 16))
label.pack()


menu = Menu(root)
root.config(menu=menu)

file = Menu(menu)

file.add _ command(label = 'Open', command = OpenFile)
file.add _ command(label = 'Exit', command =
lambda:exit())

menu.add _ cascade(label = 'File', menu = file)

root.mainloop()
```

It's not quite the same but this code allows you to open files in your system via the familiar Windows Explorer. It's worth experimenting with to see what you can do with it.

Mouse Controlled Turtle

We've already seen the Turtle module being controlled by the user via the keyboard but now we thought we'd see how the user can use their mouse as a drawing tool within Python. We have two possible code examples here, pick which works best for you.





MOUSETURTLE.PY

The first piece of code presents the standard Turtle window. Press Space and then click anywhere on the screen for the Turtle to draw to the mouse pointer. The second allows you to click the Turtle and drag it around the screen; but be warned, it can crash Python.

1st Code Example:

```
from turtle import Screen, Turtle

screen = Screen()
yertle = Turtle()

def k101():
   screen.onscreenclick(click _ handler)

def click _ handler(x, y):
   screen.onscreenclick(None)  # disable event inside
   event handler
   yertle.setheading(yertle.towards(x, y))
   yertle.goto(x, y)
   screen.onscreenclick(click _ handler)  # reenable
   event on event handler exit

screen.onkey(k101, " ")  # space turns on mouse drawing

screen.listen()

screen.mainloop()
```

2nd Code Example:

```
from turtle import *
shape("circle")
pencolor("blue")
width(2)
ondrag(goto)
listen()
```

Ninja TurtleMouse

This code utilises some interesting skills. Obviously it will stretch your Python Turtle skills to come up with any improvements, which is great, but it could make for a nice piece of code to insert into something a young child will use. Therefore it can be a fantastic project for a younger person to get their teeth into; or perhaps even as part of a game where the main character is tasked to draw a skull and crossbones or something similar.

Python Alarm Clock

Ever taken a quick break from working at the computer, then suddenly realised many minutes later that you've spent all that time on Facebook? Introducing the Python alarm clock code, where you can drop into the command prompt and tell the code how many minutes until the alarm goes off.

ALARMCLOCK.PY

This code is designed for use in the command prompt, be that Windows, Linux or macOS. There are some instructions on how to use it in the main print section but essentially it's: python3 AlarmClock.py 10 (to go off in ten minutes).

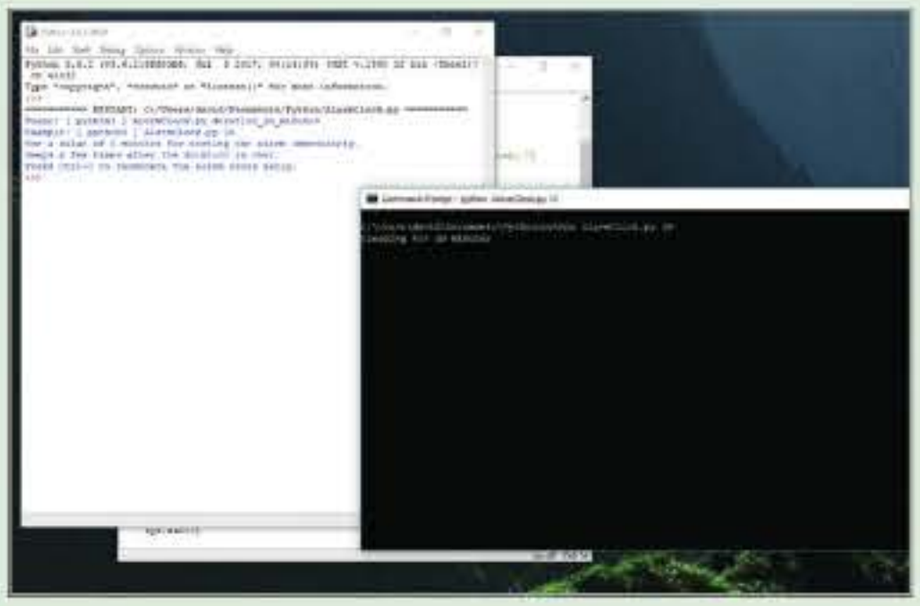
```
import sys
import string
from time import sleep

sa = sys.argv
lsa = len(sys.argv)
if lsa != 2:
   print ("Usage: [ python3 ] AlarmClock.py duration _
   in _ minutes")
   print ("Example: [ python3 ] AlarmClock.py 10")
   print ("Use a value of 0 minutes for testing the
   alarm immediately.")
   print ("Beeps a few times after the duration is over.")
   print ("Press Ctrl-C to terminate the alarm
   clock early.")
   sys.exit(1)

try:
   minutes = int(sa[1])
except ValueError:
   print ("Invalid numeric value (%s) for minutes" % sa[1])
   print ("Should be an integer >= 0")
   sys.exit(1)

if minutes < 0:
   print ("Invalid value for minutes, should be >= 0")
   sys.exit(1)

seconds = minutes * 60

if minutes == 1:
   unit _ word = " minute"
else:
   unit _ word = " minutes"
```

```
try:
   if minutes > 0:
      print ("Sleeping for " + str(minutes) + unit _ word)
      sleep(seconds)
   print ("Wake up")
   for i in range(5):
      print (chr(7)),
      sleep(1)
except KeyboardInterrupt:
   print ("Interrupted by user")
   sys.exit(1)
```

Wakey Wakey

There's some good use of try and except blocks here, alongside some other useful loops that can help you get a firmer understanding of how they work in Python. The code itself can be used in a variety of ways: in a game where something happens after a set amount of time or simply as a handy desktop alarm clock for your tea break.

Linux users, try making the alarm clock code into an alias, so you can run a simple command to execute it. Then, why not integrate a user input at the beginning to ask the user for the length of time they want until the alarm goes off, rather than having to include it in the command line.



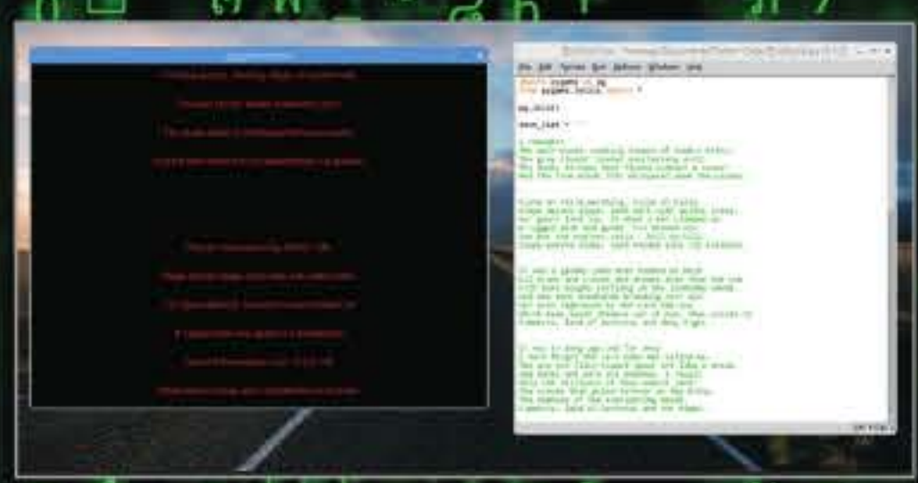
Windows users, if Python 3 is the only version installed on your system then you will need to execute the code without adding the 3 to the end of the Python command. For example:

```
python AlarmClock.py 10
```

Again, you could easily incorporate this into a Windows batch file and even set a schedule to activate the alarm at certain times of the day.

Vertically Scrolling Text

What's not to like about vertically scrolling text? Its uses are many: the beginning of a game or introduction to something epic, like the beginning of every Star Wars movie; a list of credits at the end of something, such as a Python presentation. The list goes on.

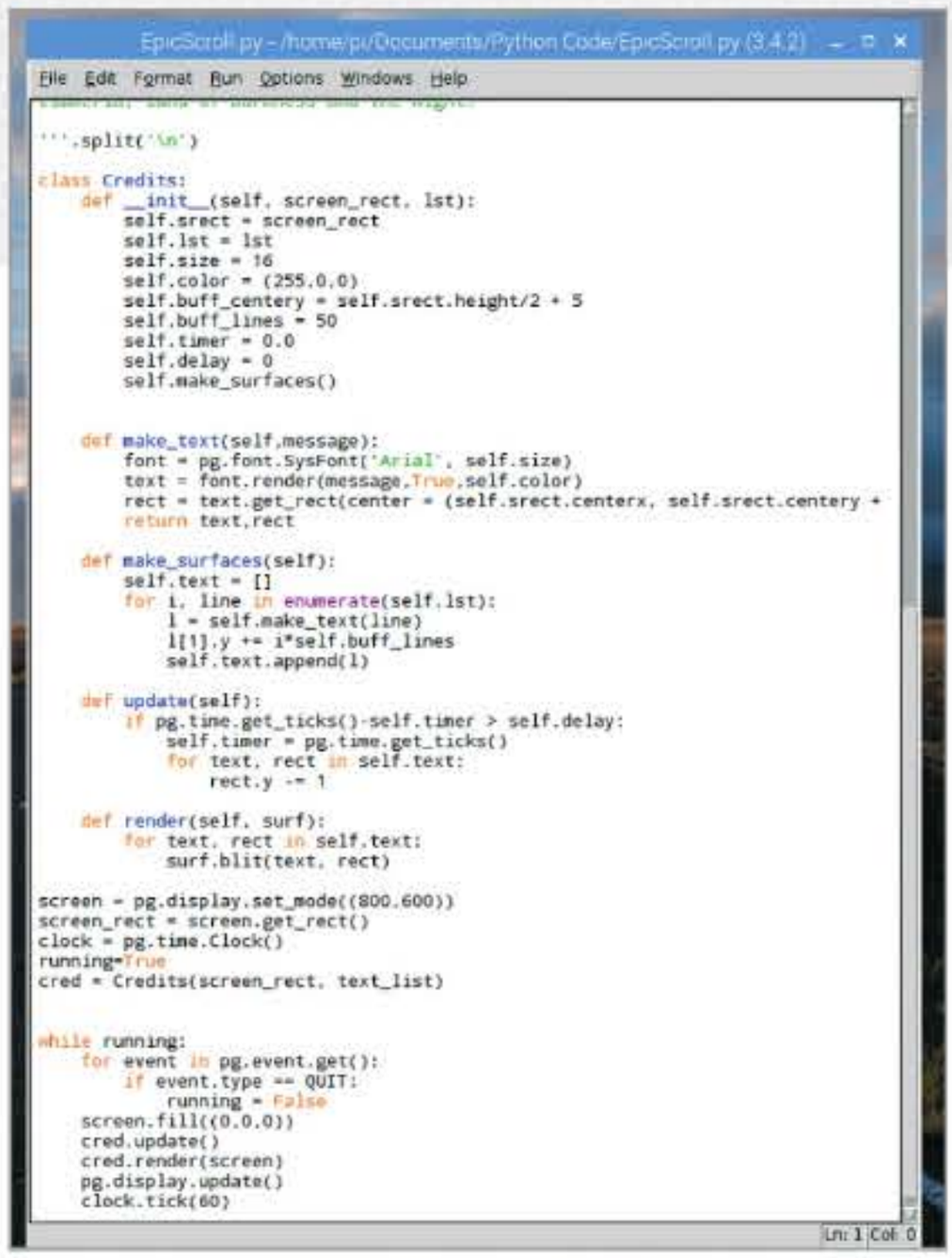


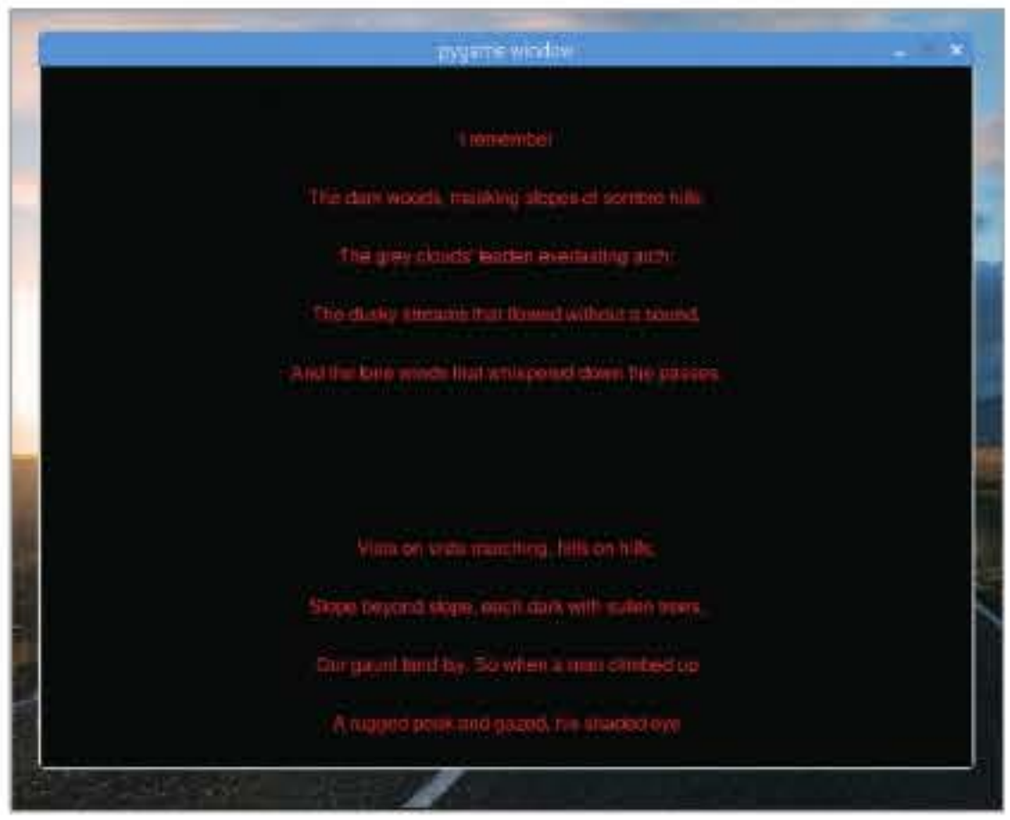
EPICSCROLL.PY

We've used the poem Cimmeria by Robert E. Howard for the code's scrolling text, along with a dramatic black background and red text. We think you'll agree, it's quite epic.

```
import pygame as pg
from pygame.locals import *

pg.init()

text_list = '''

I remember
The dark woods, masking slopes of sombre hills;
The grey clouds' leaden everlasting arch;
The dusky streams that flowed without a sound,
And the lone winds that whispered down the passes.


Vista on vista marching, hills on hills,
Slope beyond slope, each dark with sullen trees,
Our gaunt land lay. So when a man climbed up
A rugged peak and gazed, his shaded eye
Saw but the endless vista - hill on hill,
Slope beyond slope, each hooded like its brothers.


It was a gloomy land that seemed to hold
All winds and clouds and dreams that shun the sun,
With bare boughs rattling in the lonesome winds,
And the dark woodlands brooding over all,
Not even lightened by the rare dim sun
Which made squat shadows out of men; they called it
Cimmeria, land of Darkness and deep Night.


It was so long ago and far away
I have forgot the very name men called me.
The axe and flint-tipped spear are like a dream,
And hunts and wars are shadows. I recall
Only the stillness of that sombre land;
The clouds that piled forever on the hills,
The dimness of the everlasting woods.
Cimmeria, land of Darkness and the Night.


Oh, soul of mine, born out of shadowed hills,
To clouds and winds and ghosts that shun the sun,
How many deaths shall serve to break at last
This heritage which wraps me in the grey
Apparel of ghosts? I search my heart and find
Cimmeria, land of Darkness and the Night!

'''.split('\n')
```

```
class Credits:
   def _ _ init _ _ (self, screen _ rect, lst):
      self.srect = screen _ rect
      self.lst = lst
      self.size = 16
      self.color = (255,0,0)
      self.buff _ centery = self.srect.height/2 + 5
      self.buff _ lines = 50
      self.timer = 0.0
      self.delay = 0
      self.make _ surfaces()


   def make _ text(self,message):
      font = pg.font.SysFont('Arial', self.size)
      text = font.render(message,True,self.color)
      rect = text.get _ rect(center = (self.srect.
      centerx, self.srect.centery + self.buff _ centery) )
      return text,rect

   def make _ surfaces(self):
      self.text = []
      for i, line in enumerate(self.lst):
         l = self.make _ text(line)
         l[1].y += i*self.buff _ lines
         self.text.append(l)

   def update(self):
      if pg.time.get _ ticks()-self.timer > self.delay:
         self.timer = pg.time.get _ ticks()
         for text, rect in self.text:
            rect.y -= 1

   def render(self, surf):
      for text, rect in self.text:
         surf.blit(text, rect)

screen = pg.display.set _ mode((800,600))
screen _ rect = screen.get _ rect()
clock = pg.time.Clock()
running=True
cred = Credits(screen _ rect, text _ list)


while running:
   for event in pg.event.get():
      if event.type == QUIT:
         running = False
   screen.fill((0,0,0))
   cred.update()
   cred.render(screen)
   pg.display.update()
   clock.tick(60)
```



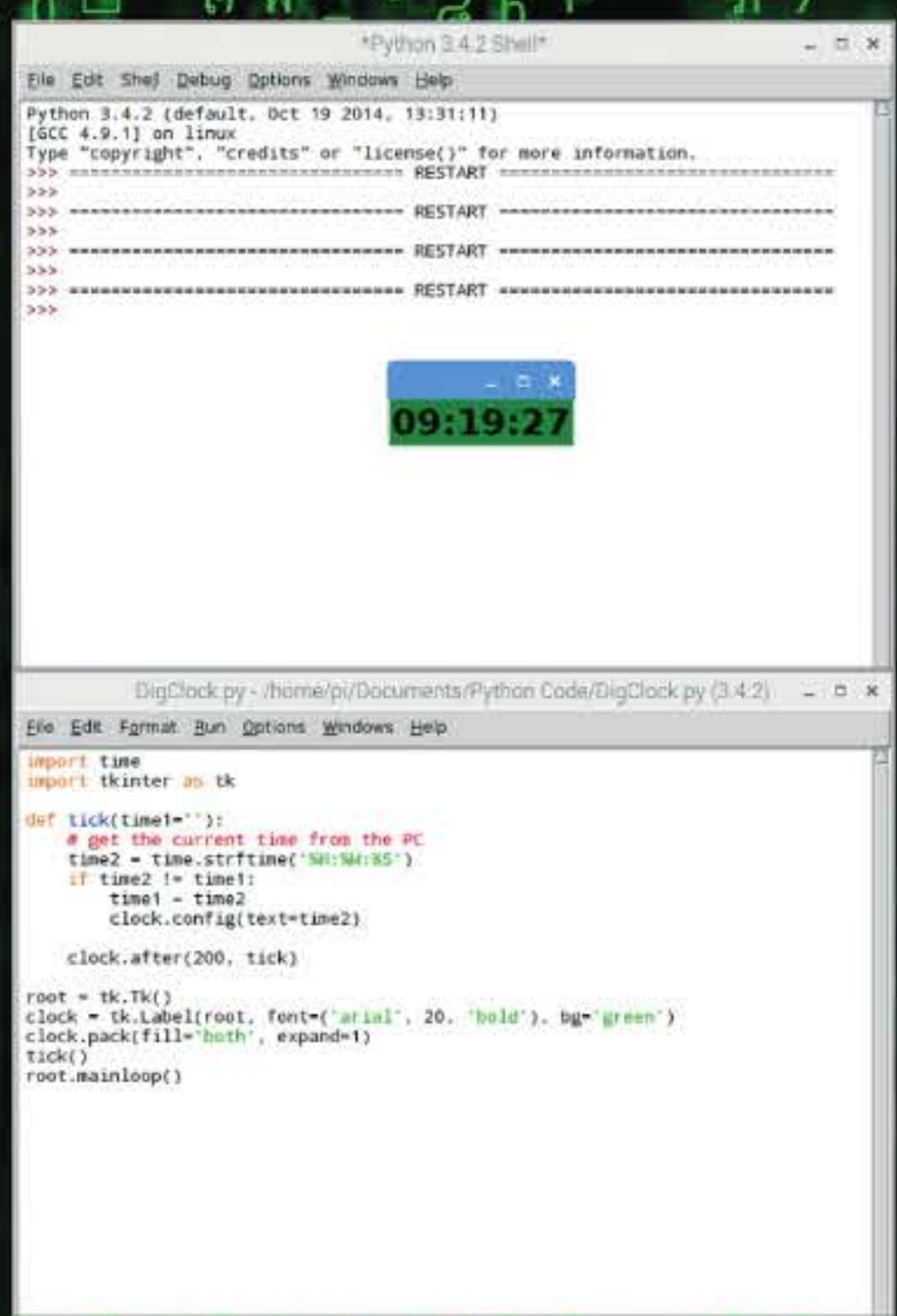


A Long Time Ago...

The obvious main point of enhancement is the actual text itself. Replace it with a list of credits, or an equally epic opening storyline to your Python game, and it will certainly hit the mark with whoever plays it. Don't forget to change the screen resolution if needed; we're currently running it at 800 x 600.

Python Digital Clock

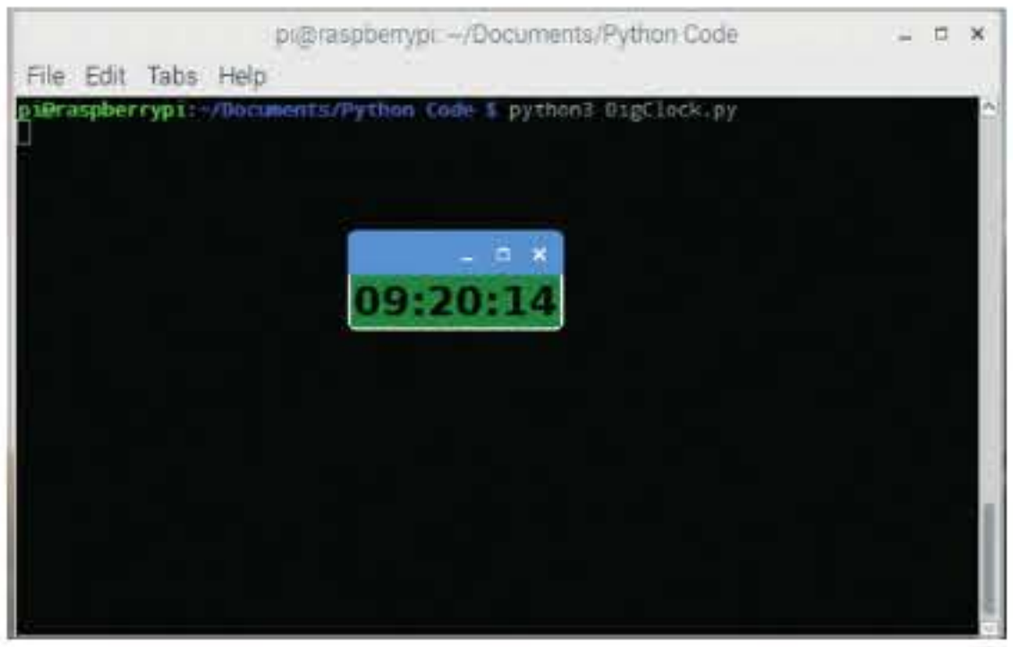
There is already a clock displayed on the desktop of most operating systems but it's always handy to have one on top of the currently open window. To that end, why not create a Python digital clock that can be a companion desktop widget for you.

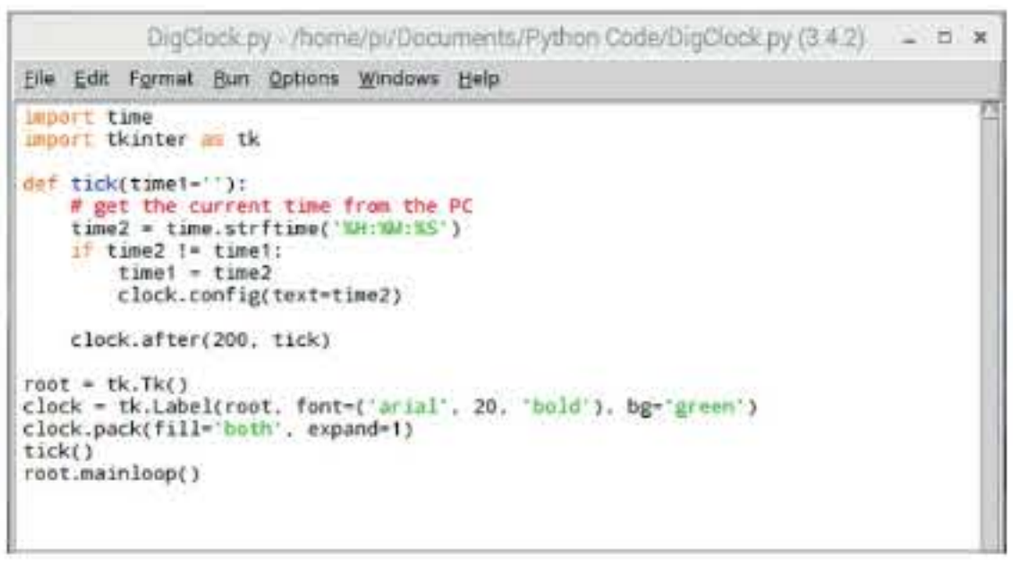


DIGCLOCK.PY

This is a surprisingly handy little script and one that we've used in the past instead of relying on a watch or even the clock in the system tray of the operating system.

```
import time
import tkinter as tk

def tick(time1=''):
   # get the current time from the PC
   time2 = time.strftime('%H:%M:%S')
   if time2 != time1:
      time1 = time2
      clock.config(text=time2)

   clock.after(200, tick)

root = tk.Tk()
clock = tk.Label(root, font=('arial', 20, 'bold'),
bg='green')
clock.pack(fill='both', expand=1)
tick()
root.mainloop()
```



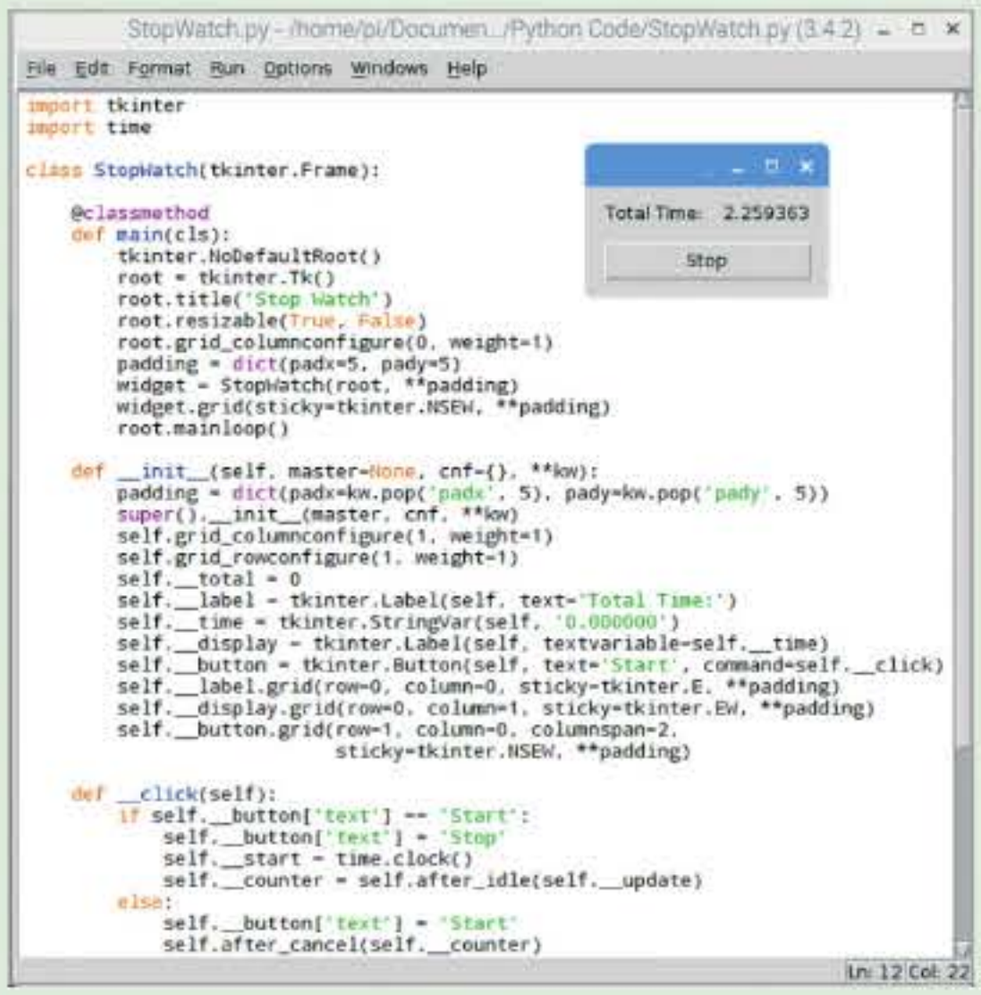


Tick Tock

This is a piece of code we've used many times in the past to keep track of time while working on multiple monitors and with just a quick glance to where we've placed it on the screen.

The Tkinter box can be moved around without affecting the time, maximised or closed by the user at will. We haven't given the Tkinter clock window a title, so you can add to that easily enough by snipping the code from other examples in this book.

Another area of improvement is to include this code when Windows or Linux starts, so it automatically pops up on the desktop. See also, if you're able to improve its functionality by including different time zones: Rome, Paris, London, New York, Moscow and so on.

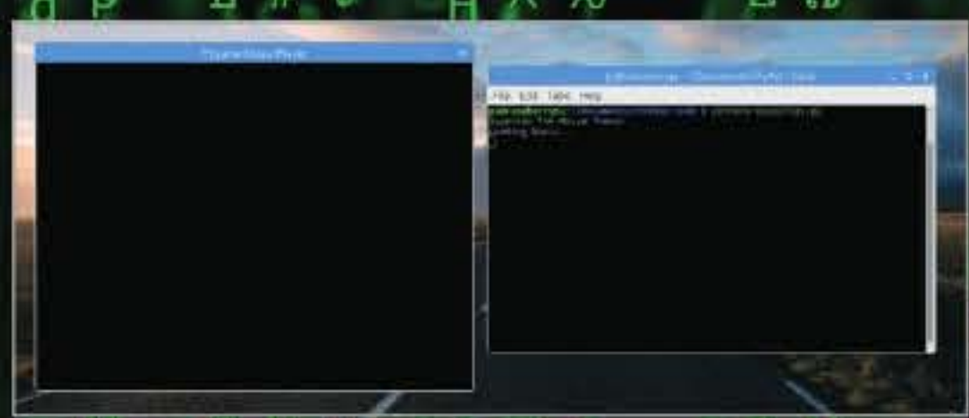


Another example, expanding on the original code, could be a digital stopwatch. For that you could use the following:

```
import tkinter
import time

class StopWatch(tkinter.Frame):

    @classmethod
    def main(cls):
        tkinter.NoDefaultRoot()
        root = tkinter.Tk()
        root.title('Stop Watch')
        root.resizable(True, False)
        root.grid _ columnconfigure(0, weight=1)
        padding = dict(padx=5, pady=5)
        widget = StopWatch(root, **padding)
        widget.grid(sticky=tkinter.NSEW, **padding)
        root.mainloop()

    def _ _ init _ _ (self, master=None, cnf={}, **kw):
        padding = dict(padx=kw.pop('padx', 5), pady=kw.
        pop('pady', 5))
        super(). _ _ init _ _ (master, cnf, **kw)
        self.grid _ columnconfigure(1, weight=1)
        self.grid _ rowconfigure(1, weight=1)
        self. _ _ total = 0
        self. _ _ label = tkinter.Label(self,
        text='Total Time:')
        self. _ _ time = tkinter.StringVar(self, '0.000000')
        self. _ _ display = tkinter.Label(self,
        textvariable=self. _ _ time)
        self. _ _ button = tkinter.Button(self,
        text='Start', command=self. _ _ click)
        self. _ _ label.grid(row=0, column=0,
        sticky=tkinter.E, **padding)
        self. _ _ display.grid(row=0, column=1,
        sticky=tkinter.EW, **padding)
        self. _ _ button.grid(row=1, column=0,
        columnspan=2,sticky=tkinter.NSEW, **padding)

    def _ _ click(self):
        if self. _ _ button['text'] == 'Start':
        self. _ _ button['text'] = 'Stop'
        self. _ _ start = time.clock()
        self. _ _ counter = self.after _ idle(self. _ _ update)
    else:
        self. _ _ button['text'] = 'Start'
        self.after _ cancel(self. _ _ counter)

    def _ _ update(self):
        now = time.clock()
        diff = now - self. _ _ start
        self. _ _ start = now
        self. _ _ total += diff
        self. _ _ time.set('{:.6f}'.format(self. _ _ total))
        self. _ _ counter = self.after _ idle(self. _ _ update)

if _ _ name _ _ == ' _ _ main _ _ ':
    StopWatch.main()
```

Pygame Music Player

The Pygame module does far more than simply display graphics or GUI windows, it can also playback MP3 files amongst other media file types. For this we can use the pygame.mixer.music() function, along with its accompanying pygame.mixer.music.play() function.



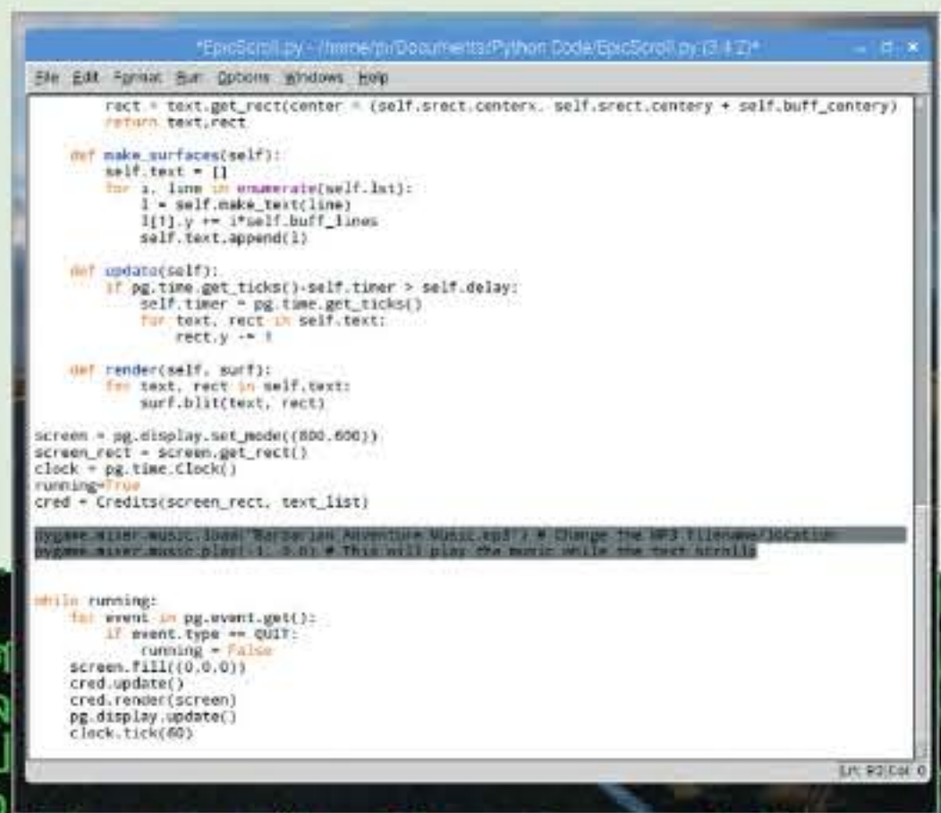
MUSICPLAY.PY

The code is remarkably simple and can be played with very few lines to clutter your already fully-loaded program. Insert this code and make sure you've either provided the full path to the MP3 or it's in the same directory as the Python file.

```
import pygame, sys
from pygame.locals import *
pygame.init()

screen = pygame.display.set _ mode((640, 480))
screen.fill((0, 0, 0, 255))
pygame.display.set _ caption("Pygame Music Player")

pygame.mixer.music.load("Superman.mp3")
print("Superman The Movie Theme!")
print("Loading Music...")
pygame.mixer.music.play(-1, 0.0)

while True:
   for event in pygame.event.get():
      if event.type == pygame.QUIT:
         pygame.quit()
         sys.exit()
   pygame.display.update()
```

Just Play the Music and Dance…

There are a few extras you can apply here to make the music code a little more interesting. For one, how about creating a list of the available MP3s on the system, then offering the user the choice of music to play via a simple list and input functions.



You can also insert a small section of the code into a previous example used: the Vertically Scrolling Text code from Page 108. For example, you can have the bulk of the code displaying the text itself but then you can insert a couple of lines to play some music alongside the text. Making it an excellent opening to a Python game or presentation:

```
pygame.mixer.music.load("Barbarian Adventure Music.
mp3") # Change the MP3 filename/location
pygame.mixer.music.play(-1, 0.0) # This will play the
music while the text scrolls
```

Insert these lines just before the While loop in the Vertically Scrolling Text example and as long as the MP3 is present, or you pointed it to the correct location in the system, the code will scroll and play the music in the background.

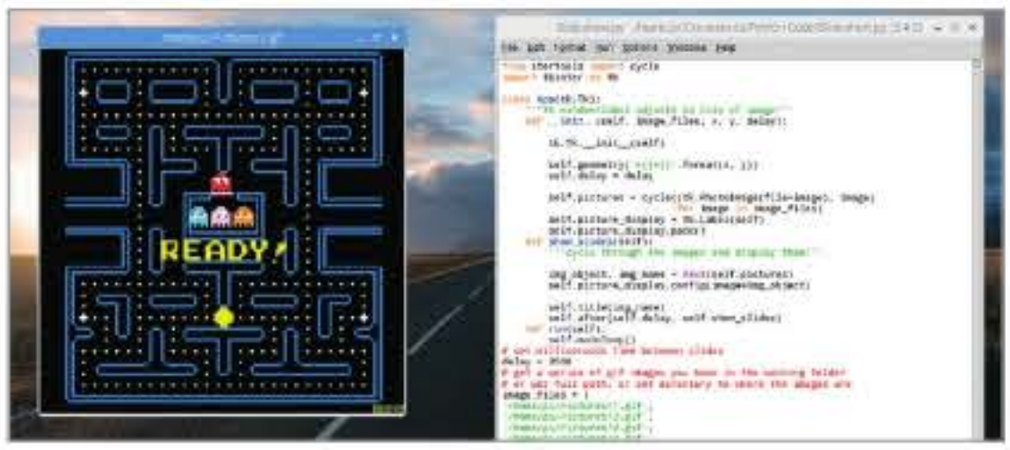
Python Image Slideshow Script

Displaying images in Python can be tricky at times. Some modules work better than others at displaying certain image media types. Pygame is often the main choice but Tkinter can do a better job whilst using less memory too. However, there are restrictions

SLIDESHOW.PY

Using Tkinter to display a series of images as a slideshow is a great method of building Python programming skills. However, the images need to be saved as a .GIF.

```
from itertools import cycle
import tkinter as tk

class App(tk.Tk):
   '''Tk window/label adjusts to size of image'''
   def _ _ init _ _ (self, image _ files, x, y, delay):

      tk.Tk. _ _ init _ _ (self)

      self.geometry('+{}+{}'.format(x, y))
      self.delay = delay

      self.pictures = cycle((tk.
      PhotoImage(file=image), image)
            for image in image _ files)
      self.picture _ display = tk.Label(self)
      self.picture _ display.pack()
   def show _ slides(self):
      '''cycle through the images and display them'''

      img _ object, img _ name = next(self.pictures)
      self.picture _ display.config(image=img _ object)

      self.title(img _ name)
      self.after(self.delay, self.show _ slides)
   def run(self):
      self.mainloop()
# set milliseconds time between slides
delay = 3500
# get a series of gif images you have in the
working folder
# or use full path, or set directory to where the
images are
image _ files = [
'/home/pi/Pictures/1.gif',
'/home/pi/Pictures/2.gif',
'/home/pi/Pictures/3.gif',
'/home/pi/Pictures/4.gif',
'/home/pi/Pictures/5.gif'
]

x = 100
y = 50
app = App(image _ files, x, y, delay)
app.show _ slides()
app.run()
```



Slideshowing Off

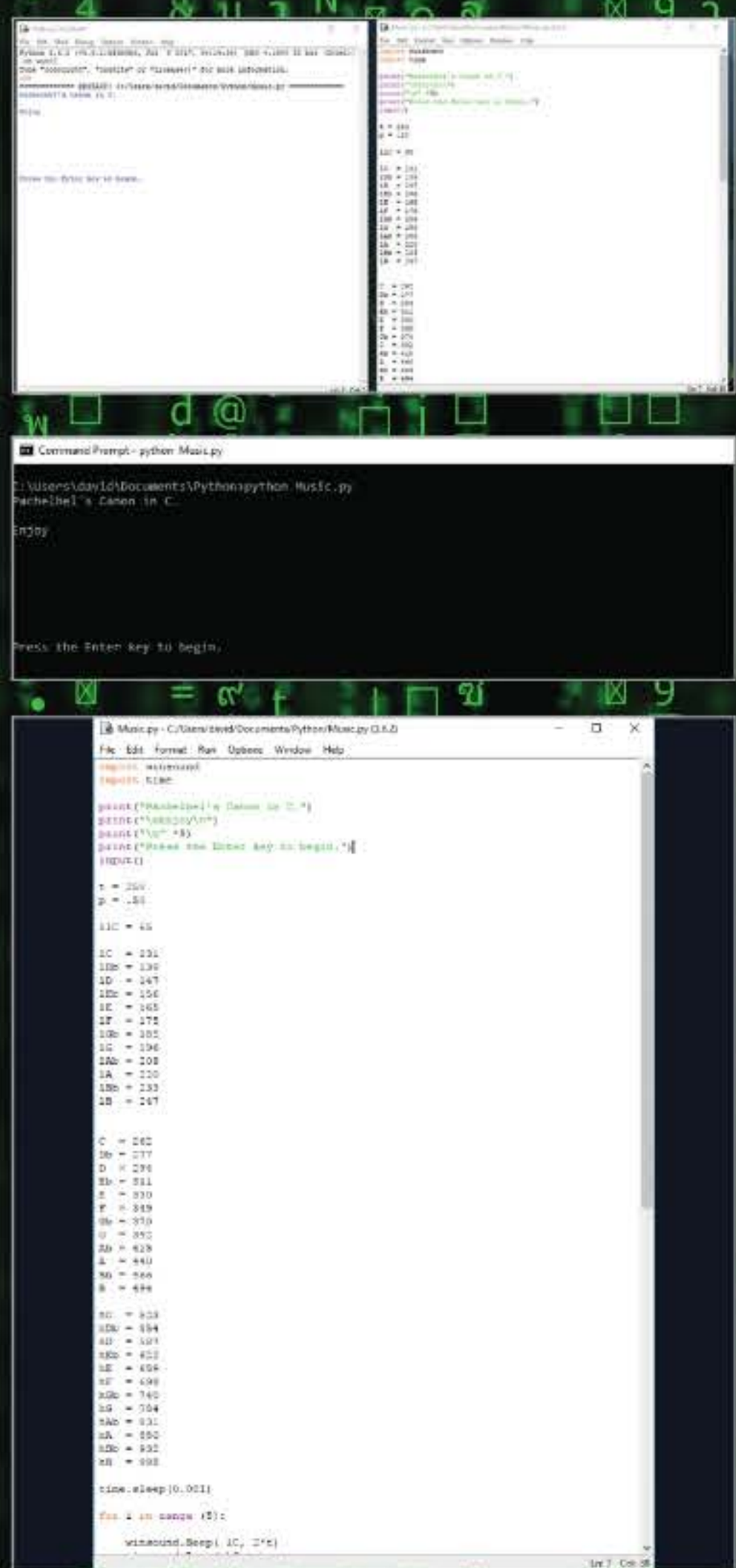
If you store your images in the same directory as the Python script is executed from, then you won't need to apply the full path destination as defined in the image_files list; you simply name them 1.gif, 2.gif and so on (or retain their full file names).

The time taken to display each image is handled by the delay variable, as noted in the code. To lessen or increase the time between slides, simply reduce or increase the number relating to the variable.

Improvements to the code can include waiting for the user to press a key or a mouse button to display the next image. Perhaps you could insert the code into the text adventure you made from Page 120, displaying a scene in the game when the character comes across something worthwhile, such as a pile of gold, a giant spider or some long lost city deep in the desert.

Playing music with the Winsound Module

Of course, instead of playing an existing MP3, you can always make your own music. The code below will play out Pachelbel's Canon in D, no less.



MUSIC.PY

The code utilises both the Time and Winsound modules, defining the tone and pitch and inserting small pauses of .5 of a second.

```
import winsound
import time

t = 250
p = .50

llC = 65

lC  = 131
lDb = 139
lD  = 147
lEb = 156
lE  = 165
lF  = 175
lGb = 185
lG  = 196
lAb = 208
lA  = 220
lBb = 233
lB  = 247

C  = 262
Db = 277
D  = 294
Eb = 311
E  = 330
F  = 349
Gb = 370
G  = 392
Ab = 415
A  = 440
Bb = 466
B  = 494

hC  = 523
hDb = 554
hD  = 587
hEb = 622
hE  = 659
hF  = 698
hGb = 740
hG  = 784
hAb = 831
hA  = 880
hBb = 932
hB  = 988

time.sleep(0.001)

for i in range (5):
```

```
winsound.Beep( lC, 2*t)
winsound.Beep( hC, t)
winsound.Beep( hE, t)
winsound.Beep( hG, t)
time.sleep(p)

winsound.Beep( lG, 2*t)
winsound.Beep( G, t)
winsound.Beep( B, t)
winsound.Beep( hD, t)
time.sleep(p)

winsound.Beep( lA, 2*t)
winsound.Beep( A, t)
winsound.Beep( hC, t)
winsound.Beep( hE, t)
time.sleep(p)

winsound.Beep( lE, 2*t)
winsound.Beep( E, t)
winsound.Beep( G, t)
winsound.Beep( B, t)
time.sleep(p)

winsound.Beep( lF, 2*t)
winsound.Beep( F, t)
winsound.Beep( A, t)
winsound.Beep( hC, t)
time.sleep(p)

winsound.Beep( llC, 2*t)
winsound.Beep( C, t)
winsound.Beep( E, t)
winsound.Beep( G, t)
time.sleep(p)

winsound.Beep( lF, 2*t)
winsound.Beep( F, t)
winsound.Beep( A, t)
winsound.Beep( hC, t)
time.sleep(p)

winsound.Beep( lG, 2*t)
winsound.Beep( G, t)
winsound.Beep( B, t)
winsound.Beep( hD, t)
time.sleep(p)
```

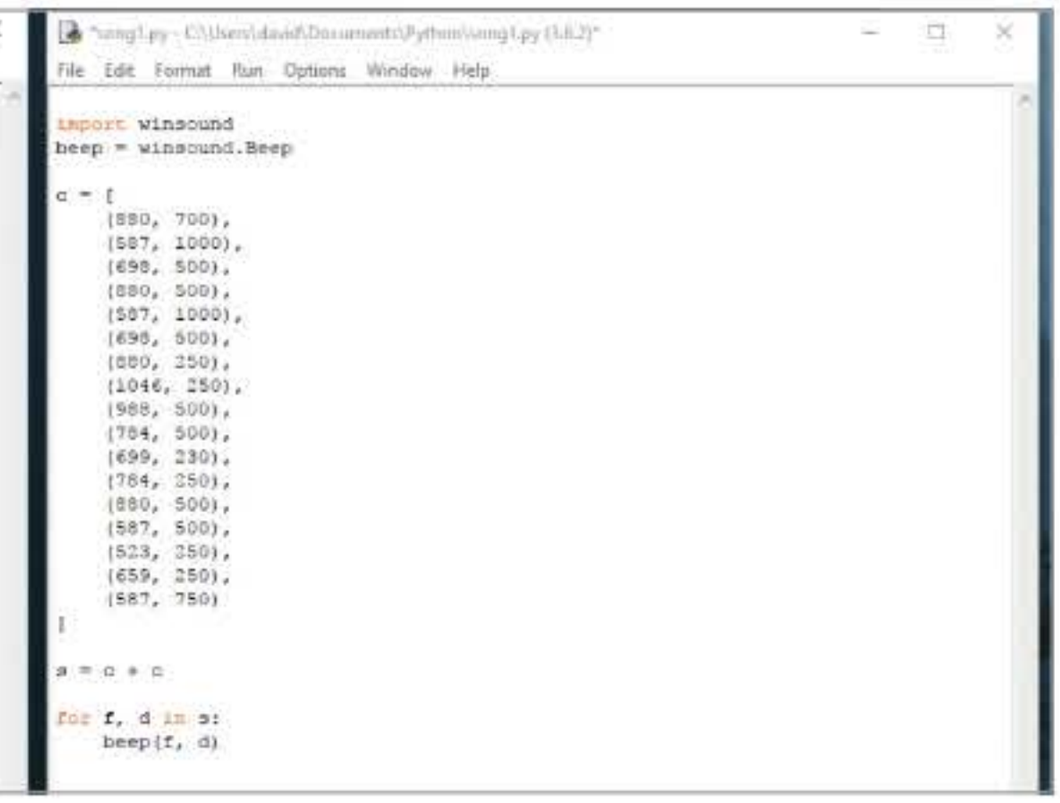
Sweet Music

Obviously the Winsound module is a Windows-only set of functions for Python. Open your IDLE in Windows and copy the code in. Press F5 to save and execute, then press the Enter key, as instructed in the code, to start the music.

Naturally you can swap out the winsound.Beep frequency and durations to suit your own particular music; or you can leave it as is and enjoy. Perhaps play around with the various methods to make other music.

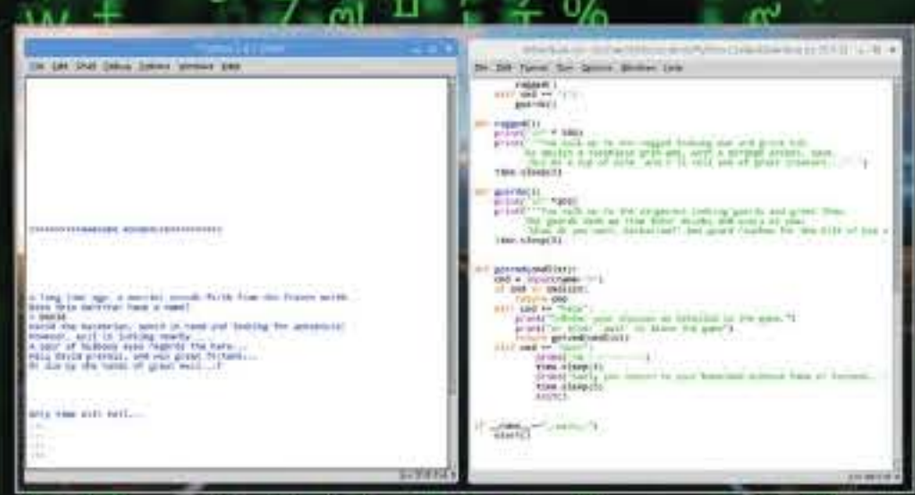
For example, players of the Nintendo classic game, The Legend of Zelda: Ocarina of Time, can enjoy the game's titular musical intro by entering:

```
import winsound
beep = winsound.Beep

c = [
    (880, 700),
    (587, 1000),
    (698, 500),
    (880, 500),
    (587, 1000),
    (698, 500),
    (880, 250),
    (1046, 250),
    (988, 500),
    (784, 500),
    (699, 230),
    (784, 250),
    (880, 500),
    (587, 500),
    (523, 250),
    (659, 250),
    (587, 750)
]

s = c + c

for f, d in s:
    beep(f, d)
```



Text Adventure Script

Text adventures are an excellent way to build your Python coding skills and have some fun at the same time. This example that we created will start you on the path to making a classic text adventure; where it will end is up to you.



ADVENTURE.PY

The Adventure game uses just the time module to begin with, creating pauses between print functions. There's a help system in place to expand upon, as well as the story itself.

```
import time

print("\n" * 200)
print(">>>>>>>>>>Awesome Adventure<<<<<<<<<<\n")
print("\n" * 3)
time.sleep(3)
print("\nA long time ago, a warrior strode forth from
the frozen north.")
time.sleep(1)
print("Does this warrior have a name?")
name=input("> ")
print(name, "the barbarian, sword in hand and looking
for adventure!")
time.sleep(1)
print("However, evil is lurking nearby....")
time.sleep(1)
print("A pair of bulbous eyes regards the hero...")
time.sleep(1)
print("Will", name, "prevail, and win great fortune...")
time.sleep(1)
print("Or die by the hands of great evil...?")
time.sleep(1)
print("\n" *3)
print("Only time will tell...")
time.sleep(1)
print('...')
time.sleep(1)
print('...')
time.sleep(1)
print('...')
time.sleep(1)
print('...')
time.sleep(5)
print("\n" *200)

print('''      You find yourself at a small inn. There's
   little gold in your purse but your sword is sharp,
   and you're ready for adventure.
   With you are three other customers.
   A ragged looking man, and a pair of dangerous
   looking guards.''')

def start():
   print("\n ----------")
   print("Do you approach the...")
   print("\n")
   print("1. Ragged looking man")
   print("2. Dangerous looking guards")

   cmdlist=["1", "2"]
   cmd=getcmd(cmdlist)
```

```
    if cmd == "1":
        ragged()
    elif cmd == "2":
        guards()

def ragged():
    print("\n" * 200)
    print('''You walk up to the ragged looking man and
    greet him.
        He smiles a toothless grin and, with a strange
        accent, says.
        "Buy me a cup of wine, and I'll tell you of
        great treasure...''')
    time.sleep(2)

def guards():
    print("\n" *200)
    print('''You walk up to the dangerous looking guards
    and greet them.
        The guards look up from their drinks and
        snarl at you.
        "What do you want, barbarian?" One guard reaches
        for the hilt of his sword...''')
    time.sleep(2)


def getcmd(cmdlist):
    cmd = input(name+">")
    if cmd in cmdlist:
        return cmd
    elif cmd == "help":
        print("\nEnter your choices as detailed in
        the game.")
        print("or enter 'quit' to leave the game")
        return getcmd(cmdlist)
    elif cmd == "quit":
        print("\n-----------")
        time.sleep(1)
        print("Sadly you return to your homeland without
        fame or fortune...")
        time.sleep(5)
        exit()


if __name__ =="__main__":
    start()
```

Adventure Time

This, as you can see, is just the beginning of the adventure and takes up a fair few lines of code. When you expand it, and weave the story along, you'll find that you can repeat certain instances such as a chance meeting with an enemy or the like.

We've created each of the two encounters as a defined set of functions, along with a list of possible choices under the cmdlist list, and cmd variable, of which is also a defined function. Expanding on this is quite easy, just map out each encounter and choice and create a defined function around it. Providing the user doesn't enter quit into the adventure, they can keep playing.

There's also room in the adventure for a set of variables designed for combat, luck, health, endurance and even an inventory or amount of gold earned. Each successful combat situation can reduce the main character's health but increase their combat skills or endurance. Plus, they could loot the body and gain gold, or earn gold through quests.

Finally, how about introducing the random module. This will enable you to include an element of chance in the game. For example, in combat, when you strike an enemy you will do a random amount of damage as will they. You could even work out the maths behind improving the chance of a better hit based on your or your opponent's combat skills, current health, strength and endurance. You could create a game of dice in the inn, to see if you win or lose gold (again, improve the chances of winning by working out your luck factor into the equation).

Needless to say, your text adventure can grow exponentially and prove to be a work of wonder. Good luck, and have fun with your adventure.

Adventure.py - /home/pi/Documents/Python Code/Adventure.py (3.4.2)

File Edit Format Run Options Windows Help

```
print("\n" *200)

CR=0
Strength=0
Health=0
Luck=0

print("The mountains of the north make for a hard life.")
print("Press Enter to roll the dice and see how strong", name, "is:")
input()
Strength=random.randint(1,20)
print(name, "has a Strength value of:", Strength)
print("\nIt's a hard life indeed, and all northoners are born warriors.")
print("Press Enter to roll the dice and see the Combat Rating for", name+".")
input()
CR=random.randint(1, 30)
print(name, "has a Combat Rating of:", CR)
print("\nYour Health is the total of your Strength and Combat Rating.")
print("Press Enter to see", name+"'s", "Health value.")
input()
Health=Strength+CR
print(name, "has a Health value of:", Health)
print("\nEveryone needs a certain amount of luck to survive.")
print("Press Enter to roll the dice and see how lucky", name, "is.")
input()
Luck=random.randint(1, 15)
if Luck > 13:
    print(name, "is luck indeed, and has a Luck value of:", Luck)
else:
    print(name, "has a Luck value of:", Luck)
time.sleep(5)
print("\n" *200)
print("Here's your character stats:\n")
print(name)
print("\nCombat Rating =", CR)
print("Strength =", Strength)
print("Health =", Health)
print("Luck =", Luck)
print("\n" *5)
print("Press Enter to start your adventure...")
input()
print("\n" *200)


print('''      You find yourself at a small inn. There's little gold in your purse
      but your sword is sharp, and you're ready for adventure.
      With you are three other customers.
      A ragged looking man, and a pair of dangerous looking guards.''')

def start():
    print("\n ----------")
    print("Do you approach the...")
    print("\n")
    print("1. Ragged looking man")
    print("2. Dangerous looking guards")
```

Python Scrolling Ticker Script

You may be surprised to hear that one of the snippets of code we're often asked for is some form of scrolling ticker. Whilst we've covered various forms of scrolling text previously, the ticker is something that seems to keep cropping up. So, here it is.

Ticker Time

The obvious improvements to the Ticker code lie in the speed of the text and what the text will display. Otherwise you can change the background colour of the ticker window, the font and the font colour, along with the geometry of the Tkinter window if you want to.

Yet another interesting element that could be introduced is one of the many text to speech modules available for Python 3. You could pip install one, import it, then as the ticker displays the text, the text to speech function will read out the variable at the same time, since the entire text is stored in the variable labelled 's'.

The ticker example can be used for system warnings, perhaps something that will display across your work or home network detailing the shutting down of a server over the weekend for maintenance; or even just to inform everyone as to what's happening. We're sure you will come up with some good uses for it.

TICKER.PY

We're using Tkinter here along with the Time module to determine the speed the text is displayed across the window.

```
import time
import tkinter as tk

root = tk.Tk()
canvas = tk.Canvas(root, root.title("Ticker Code"),
height=80, width=600, bg="yellow")
canvas.pack()
font = ('courier', 48, 'bold')
text _ width = 15

#Text blocks insert here....

s1 = "This is a scrolling ticker example. As you
can see, it's quite long but can be a lot longer if
necessary... "
s2 = "We can even extend the length of the ticker
message by including more variables... "
s3 = "The variables are within the s-values in
the code. "
s4 = "Don't forget to concatenate them all before the
For loop, and rename the 'spacer' s-variable too."

# pad front and end of text with spaces
s5 = ' ' * text _ width
# concatenate it all
s = s5 + s1 + s2 + s3 + s4 + s5
x = 1
y = 2
text = canvas.create _ text(x, y, anchor='nw', text=s,
font=font)
dx = 1
dy = 0  # use horizontal movement only

# the pixel value depends on dx, font and length of text
pixels = 9000

for p in range(pixels):
   # move text object by increments dx, dy
   # -dx --> right to left
   canvas.move(text, -dx, dy)
   canvas.update()
   # shorter delay --> faster movement
   time.sleep(0.005)
   #print(k)  # test, helps with pixel value

root.mainloop()
```

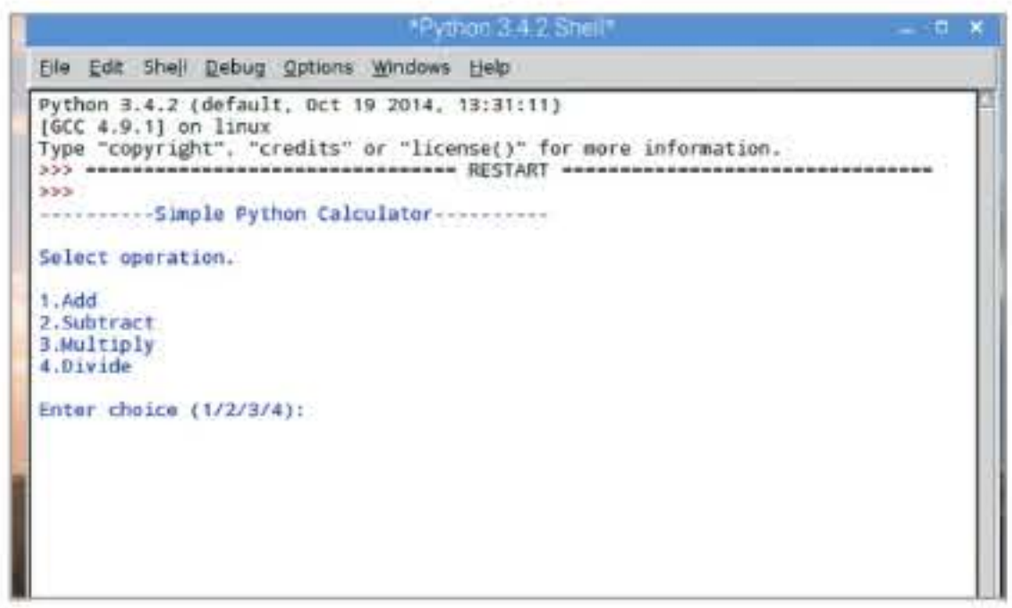
Simple Python Calculator

Sometimes the simplest code can be the most effective. Take for example, this Simple Python Calculator script. It's based on the Create Your Own Modules section seen earlier but doesn't utilise any external modules.

CALCULATOR.PY

We created some function definitions to begin with, then lead on to the user menu and inputs. It's an easy piece of code to follow and as such can also be expanded well too.

```
print("----------Simple Python Calculator----------\n")

def add(x, y):
   return x + y

def subtract(x, y):
   return x - y

def multiply(x, y):
   return x * y

def divide(x, y):
   return x / y

print("Select operation.\n")
print("1.Add")
print("2.Subtract")
print("3.Multiply")
print("4.Divide")

choice = input("\nEnter choice (1/2/3/4):")

num1 = int(input("\nEnter first number: "))
num2 = int(input("Enter second number: "))

if choice == '1':
   print(num1,"+",num2,"=", add(num1,num2))

elif choice == '2':
   print(num1,"-",num2,"=", subtract(num1,num2))

elif choice == '3':
   print(num1,"*",num2,"=", multiply(num1,num2))

elif choice == '4':
   print(num1,"/",num2,"=", divide(num1,num2))
else:
   print("Invalid input")
```



Improved Calculations

The obvious contender for improvement here is using the Create Your Own Modules route and extracting the function definitions as a module. You can then call the module and focus on the body of the code.

The other area of improvement is code itself. Where there's just a single shot at making a calculation, you could encase it in a while loop, so once a value is presented the user is sent back to the main menu. Perhaps, improvement to the Invalid Input section is worth looking into as well.

Hangman Game Script

Hangman is a great game to program into Python. It can be extremely complex, displaying graphics, the number of guesses left in the secret word, a huge bank of available words picked at random and countless other elements. It can also be quite simple. Here we have a mix between the two.

HANGMAN.PY

We've made a Hangman game board (the gallows) out of characters that can be displayed in the IDLE Shell, along with a huge bank of words to randomly choose from.

```
import random

board = ['''

>>>>>>>>>>Hangman<<<<<<<<<<

+---+
|   |
    |
    |
    |
    |
=========''', '''

+---+
|   |
O   |
    |
    |
    |
=========''', '''

+---+
|   |
O   |
|   |
    |
    |
=========''', '''

 +---+
 |   |
 O   |
/|   |
     |
     |
=========''', '''

 +---+
 |   |
 O   |
/|\  |
     |
     |
=========''', '''

 +---+
 |   |
 O   |
/|\  |
/    |
```

```
        |
=========''', '''

 +---+
 |   |
 O   |
/|\  |
/ \  |
        |
=========''']


class Hangman:
   def _ _ init _ _ (self,word):
      self.word = word
      self.missed _ letters = []
      self.guessed _ letters = []

   def guess(self,letter):
      if letter in self.word and letter not in self.
      guessed _ letters:
         self.guessed _ letters.append(letter)
      elif letter not in self.word and letter not in
      self.missed _ letters:
         self.missed _ letters.append(letter)
      else:
         return False
      return True

   def hangman _ over(self):
      return self.hangman _ won() or (len(self.missed _
      letters) == 6)

   def hangman _ won(self):
      if '_' not in self.hide _ word():
         return True
      return False

   def hide _ word(self):
      rtn = ''
      for letter in self.word:
         if letter not in self.guessed _ letters:
            rtn += '_'
         else:
            rtn += letter
      return rtn

   def print _ game _ status(self):
      print (board[len(self.missed _ letters)])
      print ('Word: ' + self.hide _ word())
      print ('Letters Missed: ',)
      for letter in self.missed _ letters:
         print (letter,)
      print ()
      print ('Letters Guessed: ',)
      for letter in self.guessed _ letters:
         print (letter,)
      print ()

def rand _ word():
   bank = 'ability about above absolute accessible
   accommodation accounting beautiful bookstore
   calculator clever engaged engineer enough
   handsome refrigerator opposite socks interested
   strawberry backgammon anniversary confused
   dangerous entertainment exhausted impossible
   overweight temperature vacation scissors
   accommodation appointment decrease development
   earthquake environment brand environment necessary
   luggage responsible ambassador circumstance
   congratulate frequent'.split()
   return bank[random.randint(0,len(bank))]

def main():
   game = Hangman(rand _ word())
   while not game.hangman _ over():
      game.print _ game _ status()
      user _ input = input('\nEnter a letter: ')
      game.guess(user _ input)

   game.print _ game _ status()
   if game.hangman _ won():
      print ('\nCongratulations! You have won!!')
   else:
      print ('\nSorry, you have lost.')
      print ('The word was ' + game.word)

   print ('\nGoodbye!\n')

if _ _ name _ _ == " _ _ main _ _ ":
   main()
```

QUIT()

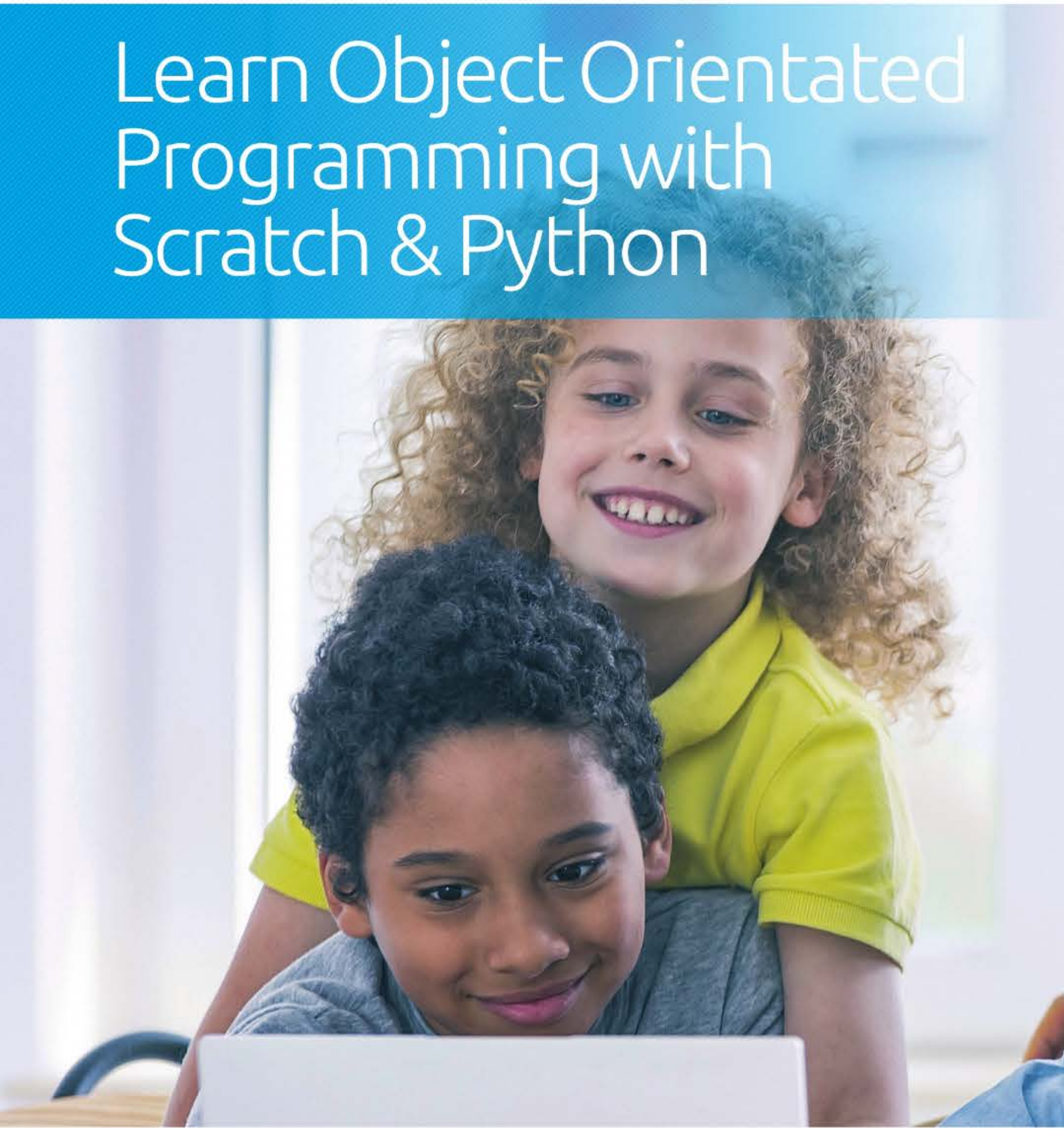
Since this is the last example in our Python code repository, we thought we'd go out with a bang and feature the hangman gallows being drawn with each incorrect guess of the word. Don't worry if it looks misaligned in the text here, this is merely due to the differences between using the Python IDLE editor and pasting the code into a word processor (which formats things differently).

There's plenty you can do to improve, enhance and expand on what we've presented here. You can include a routine that will return an error if the user enters a number or character. You can include extra points for someone who guesses the entire word in one go rather than one letter at a time. Maybe you could add Chopin's Funeral March should you lose the game; or something celebratory if you win.

Perhaps consider replacing the bank of words. They're found under the bank list and could easily be swapped out for something more difficult. Recall the text document containing 466,000 English words we used on Page 80's Random Module? Perhaps you could swap the words in the bank to instead read the contents of the text file:

```
def rand _ word():
   with open("/home/pi/Downloads/words.txt", "rt") as f:
      bank=f.readlines()
   return bank[random.randint(0,len(bank))]
```

Learn Object Orientated Programming with Scratch & Python



Scratch is a powerful introduction to programming created by MIT (Massachusetts Institute of Technology). It's an extremely high level programming language designed to teach children the building blocks of programming. It's also designed to teach kids a popular programming methodology known as Object Orientated Programming (OOP). Scratch is extremely versatile and can work with Python and other code as you progress and create fascinating projects.

124 Getting Started with Scratch
126 Creating Scripts in Scratch
128 Interaction in Scratch
130 Using Sprites in Scratch
132 Sensing and Broadcast
134 Objects and Local Variables
136 Global Variables and a Dice Game
138 Classes and Objects

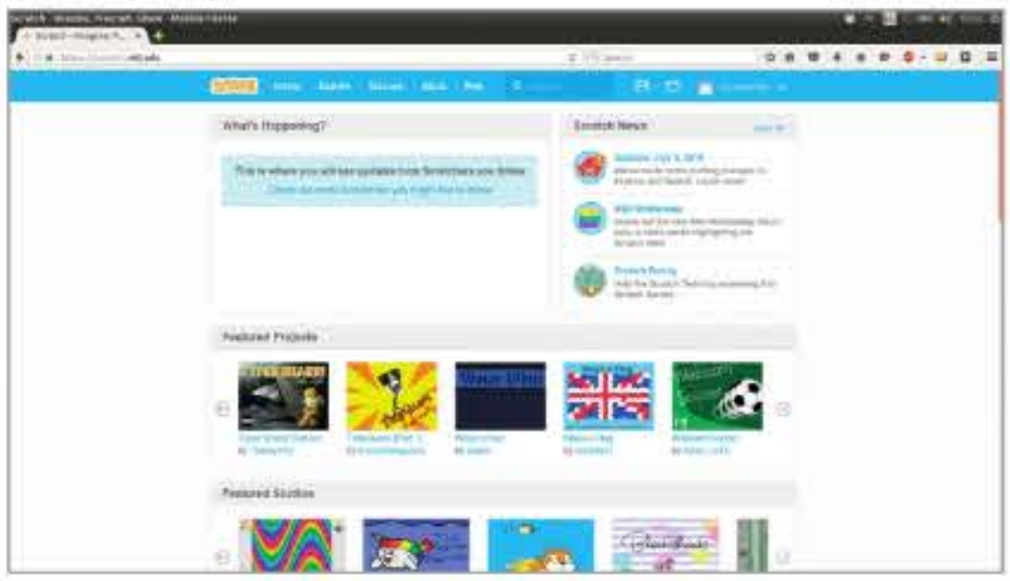
Getting Started with Scratch

If you are completely new to programming then Scratch is the perfect place to start. With Scratch you can learn the building blocks of programming and important programming concepts in a highly visual interface.

INSTALLING SCRATCH

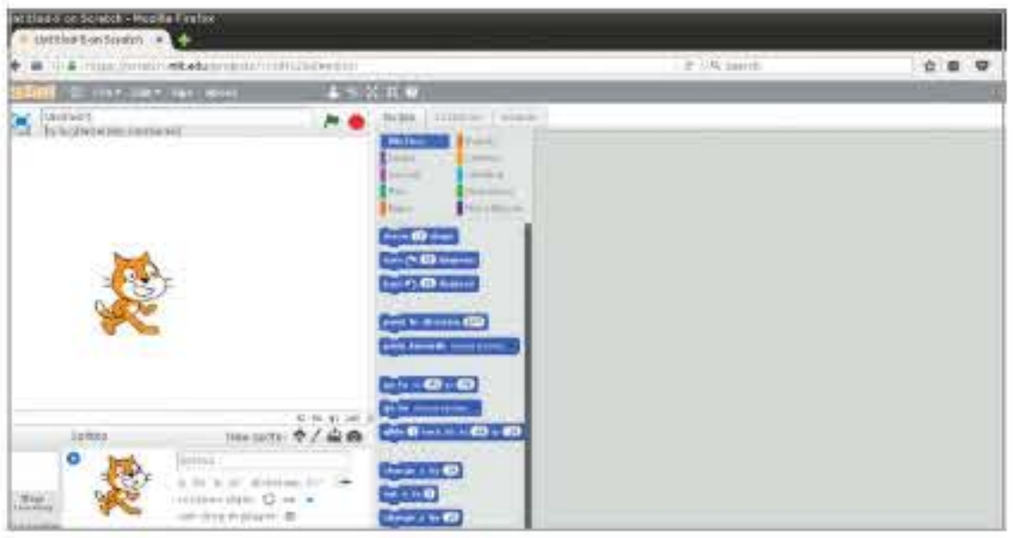
Scratch can be run inside your web browser at scratch.mit.edu. You need to have Flash installed in your browser; if isn't already, it can be installed from get.adobe.com/flashplayer. Sign up for an account with Scratch so you can save your programs.

STEP 1 Scratch runs from inside the web browser. Click Create to open a new document. The Scratch interface opens in the web browser, click the maximise button on your browser so you have plenty of space to view the window and all its contents.



STEP 2 You can see the Scratch interface with a list of blue items in the Block Palette, an empty Script Area and a Stage. On the Stage will be an orange cartoon cat, known as "Scratch Cat". This is the default sprite that comes with all new projects; you will also see smaller versions of the sprite above the Script Area and in the Sprites Panel.

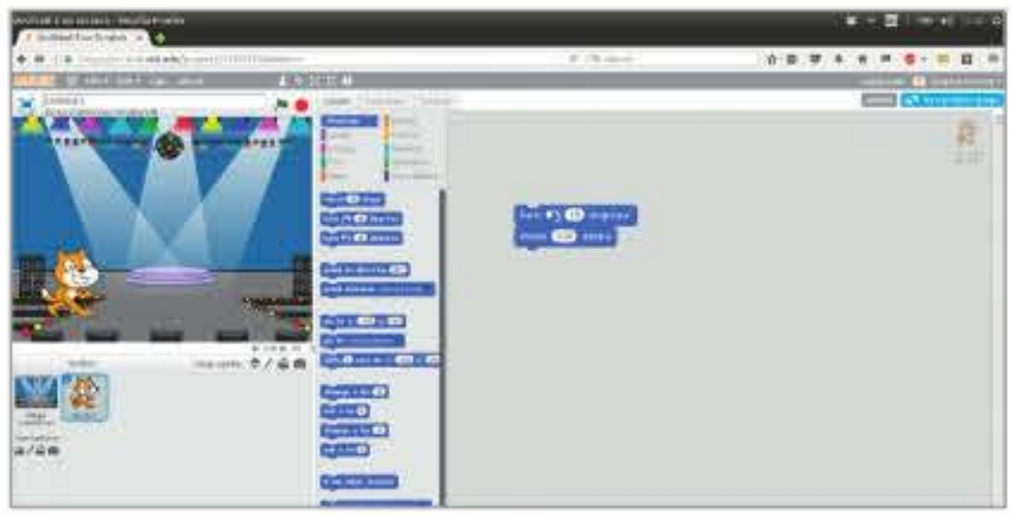
STEP 3 Let's take a look at Scratch Cat. Use click and drag with the mouse to position him on the Stage. At the top, just above Scripts, you'll see two icons for Grow and Shrink. Click one and click the cat to resize him. Shift-click on Scratch Cat and choose Info to access rotation controls. Click the blue back button to get back to the Sprites pane.



STEP 4 Scratch Cat looks a little lonely on his white space, so let's give him a background. Click the Stage icon to the left of the Sprites Pane. The Script Area switches to Backdrop Library displaying the available backgrounds. Click Music and Dance and choose spotlight-stage. Click OK.

STEP 5 The background appears on the Stage and Scratch Cat looks a lot happier. Let's create a script that moves him to the stage. Click Sprite1 in the Sprites Pane to select the cat and click the Scripts tab to return to the Script Area. Now click the blue Motion tab at the top of the Block Palette.

STEP 6 Drag the `turn [15] degrees` block (with an anti-clockwise symbol) from the Block Palette to the Script Area. Now drag the `move [10] steps` block and connect it to the bottom of the Turn 15 Degrees block. They will snap together. Change the `10` in `move [10] steps` to 100. Our program is now ready. Click the script (the two blocks) to see what it does. Scratch Cat will rotate and move towards the stage.



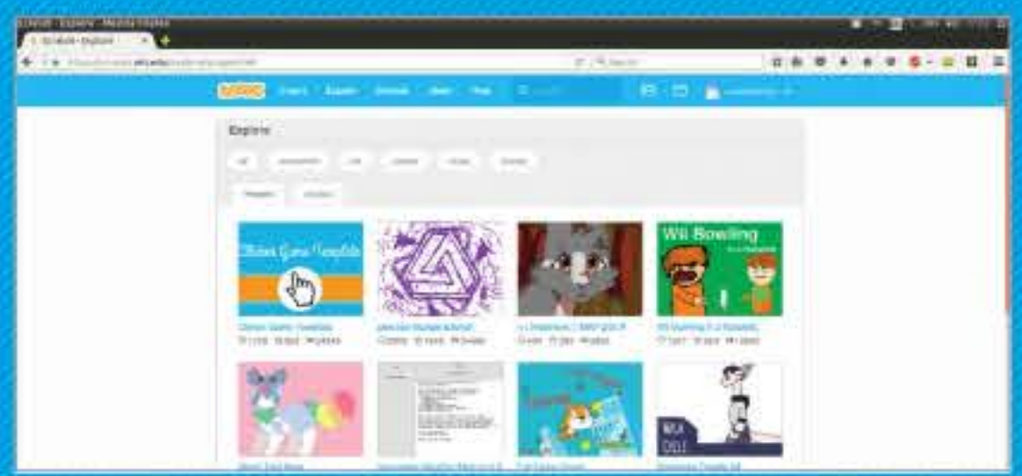
SAVING SCRATCH FILES

Take the time to learn how to open and save your files, and open other test programs, before you get stuck into programming with Scratch. There are lots of Scratch programs available so it's easy to learn alongside other users.

STEP 1 Click File > Save Now to save your project. Enter a name in the New Filename box; we called ours "Scratch_Cat_On_Stage". As we mentioned in both Python and Unix tutorials, it's important to avoid any special characters and spaces in your filenames. Use underscores "_" instead.

STEP 2 Choose File > Go to My Stuff to exit the stage and view the saved file. Click the Scratch_Cat_On_Stage link to view your file. You can add Instructions, Notes and Credits here, and Tags. Click See Inside to head back to viewing your code again.

STEP 3 Lots of example Scratch files can be found on the MIT website by clicking Explore. Here you can see a huge range of projects built by other users and you'll also be able to share your own projects. Choose a project from Explore to open it.



STEP 4 You can run the project directly inside the main window by clicking the Green Flag icon. Click the Star icon to Favourite or bookmark the project and the Heart icon to like it. More importantly, click the See Inside to see the code used to create the project. This is a good way to learn how Scratch code is being used.

Creating Scripts in Scratch

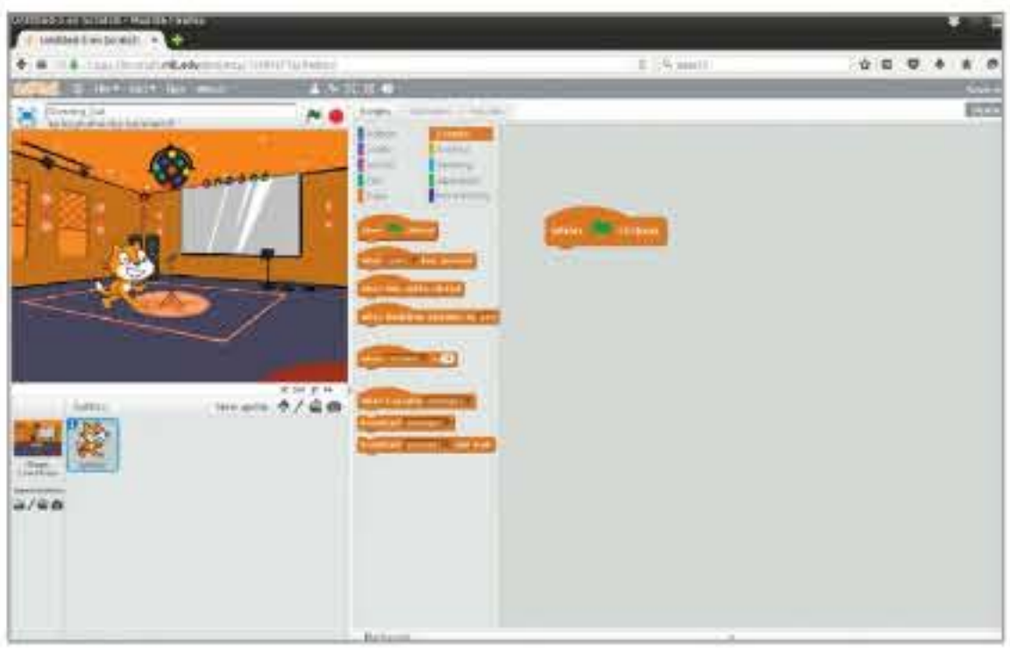
The program you make in Scratch is a script, or a bunch of scripts. In this tutorial we'll take a look at how you construct your scripts to build up a great program. This is great starter practice for creating objects in Python.

VISUAL CODING

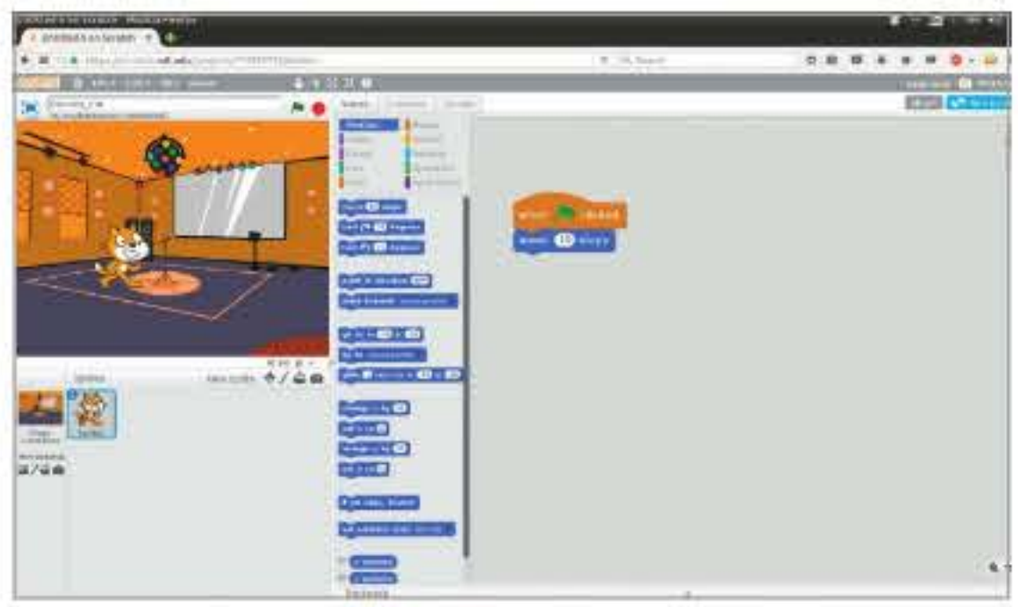
The scripts in Scratch are created by snapping together blocks. These blocks are similar to the code you find in more complex programming languages, such as Python, but much easier to understand.

STEP 1 Click Create to start a new Scratch project and name it Dancing Cat. You're going to put your cat and some other characters on a dance floor and get them to bust some moves. Click Stage, then Music and Dance and choose party-room. Drag the Scratch Cat graphic around the Stage to find a good starting position.

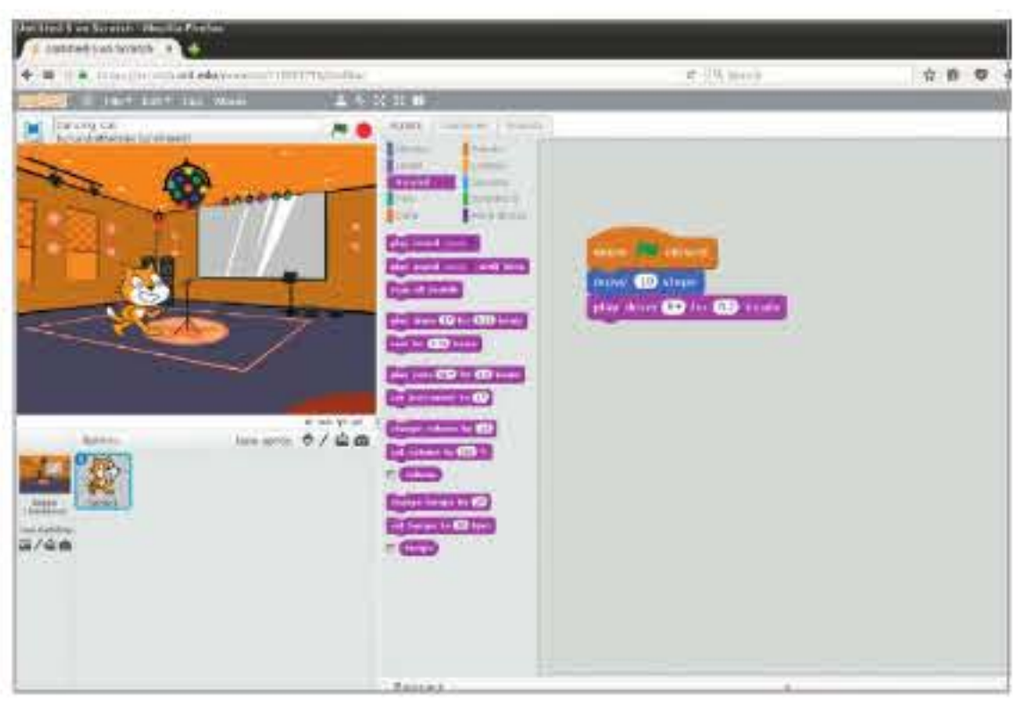
STEP 2 Click on Sprite1 in the Sprites Panes and click the Scripts tab above the Scripts area. Now click Events in the Blocks Pane and drag the `when flag clicked` block into the Scripts area. This block represents the start of your program. It tells Scratch to run through the blocks below it when we click the Green Flag icon above the Stage window.



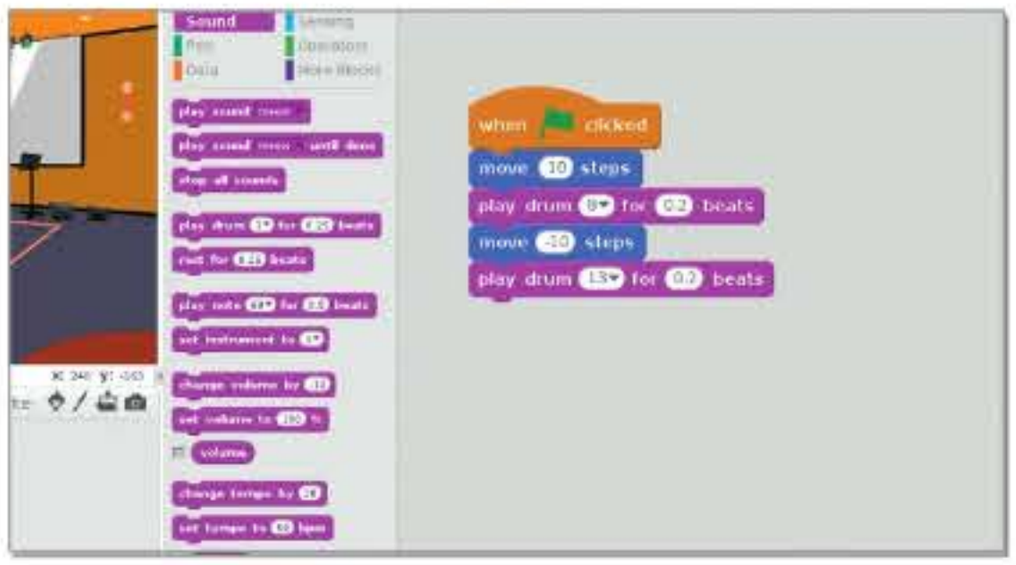
STEP 3 Click the Motion tab and drag the `move [10] steps` block and connect it beneath the `when flag clicked` block. A quick word about that [10]. When you write a number or word inside those square brackets, that's the way of saying you can choose a value. It's the equivalent of a variable", because it varies. We'll tell you which number or selection we're using but you can use any you want. Play around.



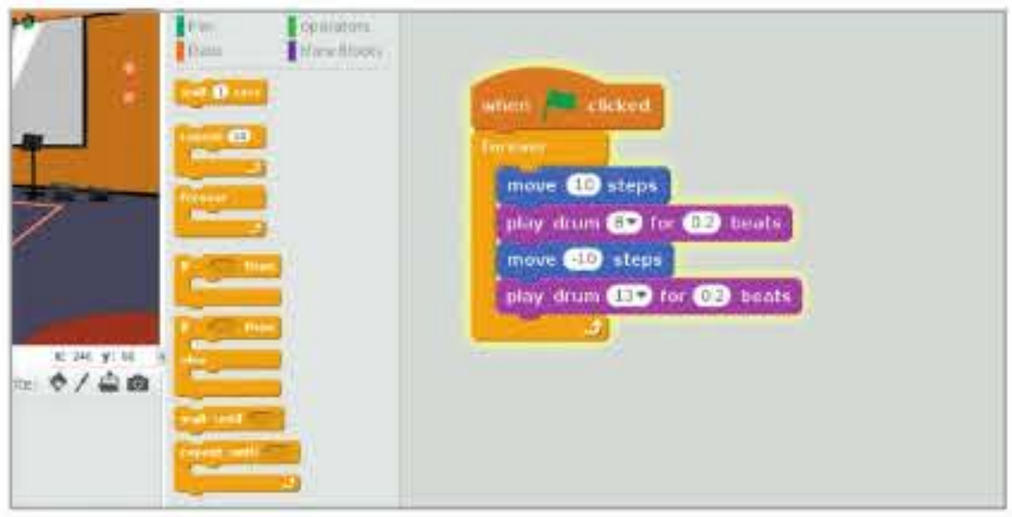
STEP 4 It's not much of a disco, so let's add some sound. Click the sound tab and drag `play drum [8] for [0.2] beats` and connect it to the bottom of the stack of blocks. Click on the blocks and Scratch Cat will move and a sound will come from your speaker.



STEP 5 Dancing is a back and forth affair, so let's get Scratch Cat moving back. Drag another `move [10] steps` block to the bottom of the stack. Now click the `10` and change it to `-10` (minus 10). Entering minus figures moves the cat backwards. Drag another `play drum` block to the bottom of the script. Pick a different drum sound. We chose `13`.



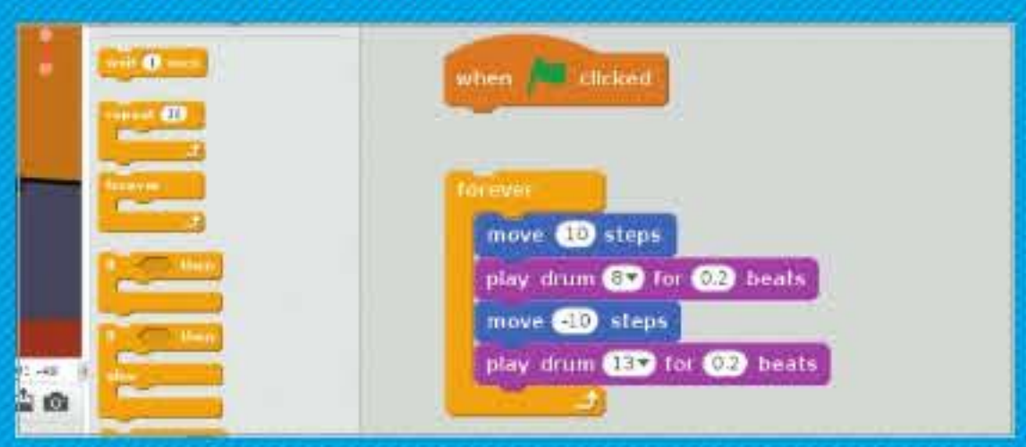
STEP 6 Scratch Cat only moves back and forth once, which isn't much of a party. Click Control and drag the `forever` block to the Script Area. Carefully position it beneath the `when [flag] clicked` block but above the `move [10] steps` block. The script should nest within the two prongs of the `forever` block. Click the Green Flag icon to start the disco. Click the red Stop icon to end the program.



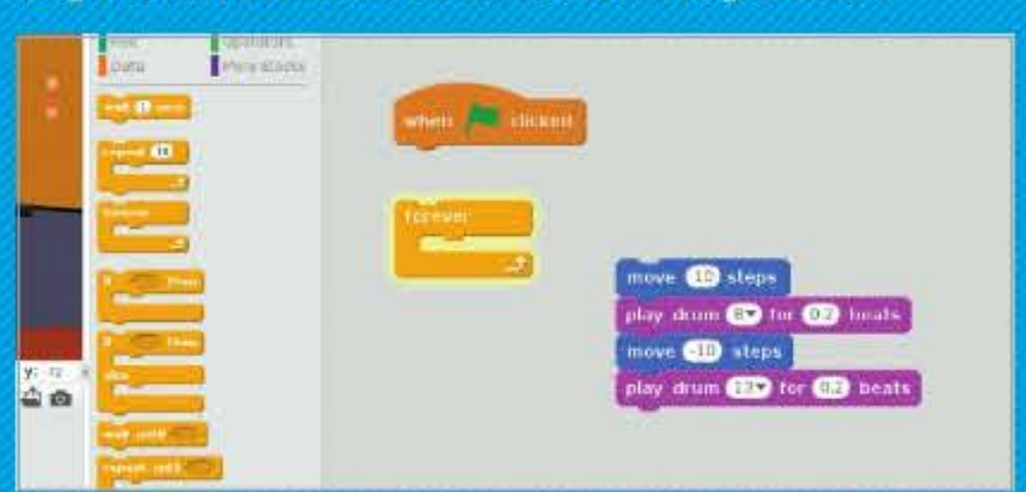
EDITING SCRIPTS

Nothing is set in stone, and you can move your blocks in and out of scripts and even have several scripts or parts of scripts in the Script Area. Scratch is far more forgiving than other programming languages for experimentation.

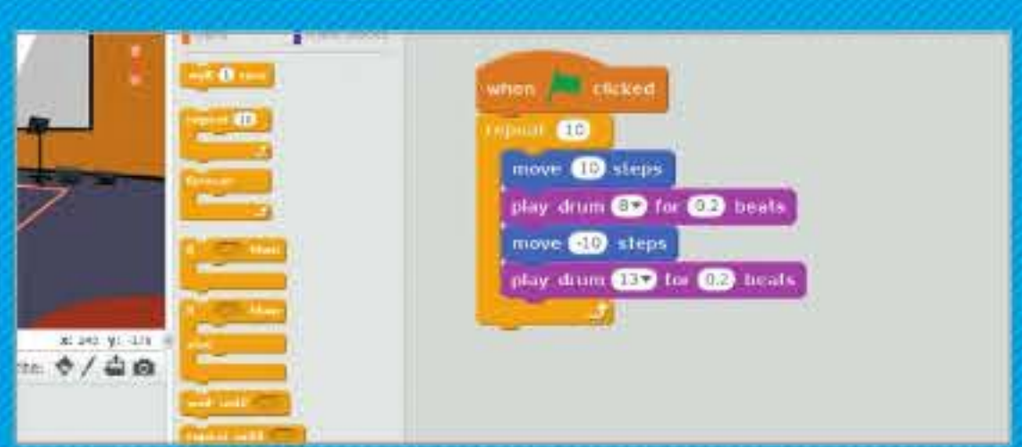
STEP 1 It's pretty bad form to use the `forever` block or a forever loop in programming. Programs are supposed to run from start to a finish. Even programs like Scratch have an end point when you quit the program. You want to replace the `forever` block with a `repeat` one. Click the `forever` block in your script and drag it down to separate it from the other blocks.



STEP 2 Your `move` and `play drum` blocks are still nested within the `forever` block though and you want to keep them. Click the topmost `move` block and drag it out of the `forever` block. It's now good to get rid of the `forever` block so drag it to the left and back to the Blocks List to get rid of it.



STEP 3 Now drag a `repeat [10]` block from the block list and connect it to the `when [flag] clicked` block in the Script Area. Now drag the top `play drum` block of the stack inside the `repeat [10]` block. If you drag the top block all the blocks underneath move with it and the whole lot will be nested inside the `repeat [10] stack`.



STEP 4 You can position the stack anywhere on the Script Area and even keep the unused blocks around, although we think it's good practice to keep only what you are using in the Script Area and remove any unused blocks. Click the Green Flag icon above the Stage Window to view Scratch Cat doing a short dance.

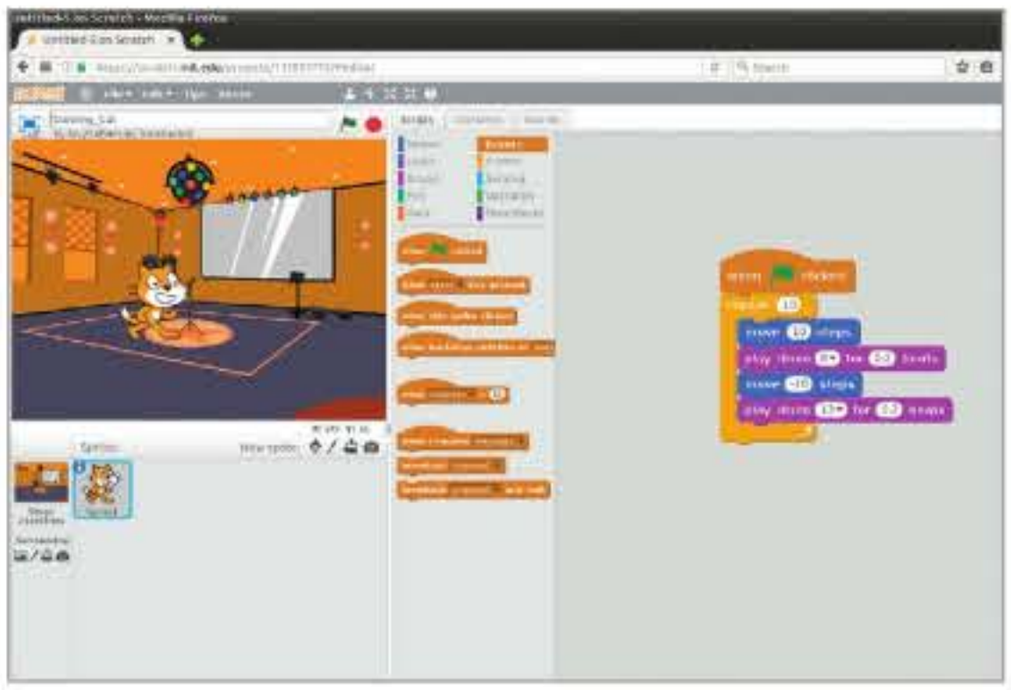
Interaction in Scratch

Your Scratch Cat is now dancing back and forth but wouldn't it be great if you could control him. In this tutorial you're going to look at creating keyboard interactions in Scratch. Let's get our disco cat really grooving!

INTERACTIVE CONTROL

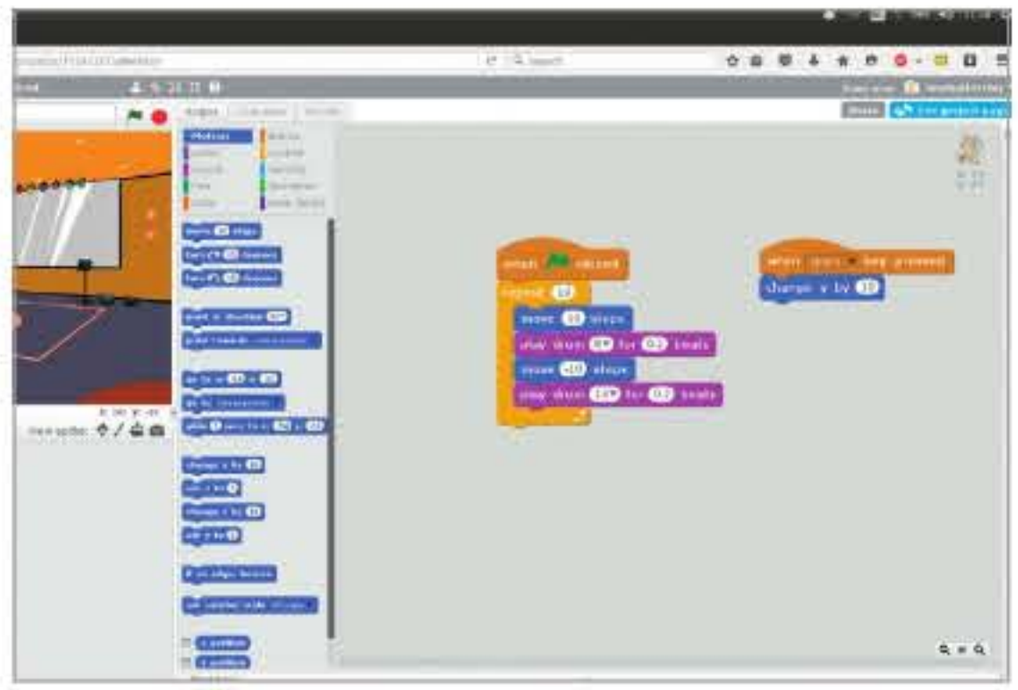
The only Control option we've really looked at so far is the `when [flag] clicked` block, which starts the program. Once the program is running it does its thing, right up until it finishes. You're going to use the other Control blocks to do something more interesting.

STEP 1 Open the Dancing Cat program from previous tutorials. Select Sprite1 and click on Events so you can see the `when [flag] clicked` script. Now click Control in the Block Palette and drag the `when [space] key pressed` block to an empty part of the Script Area.

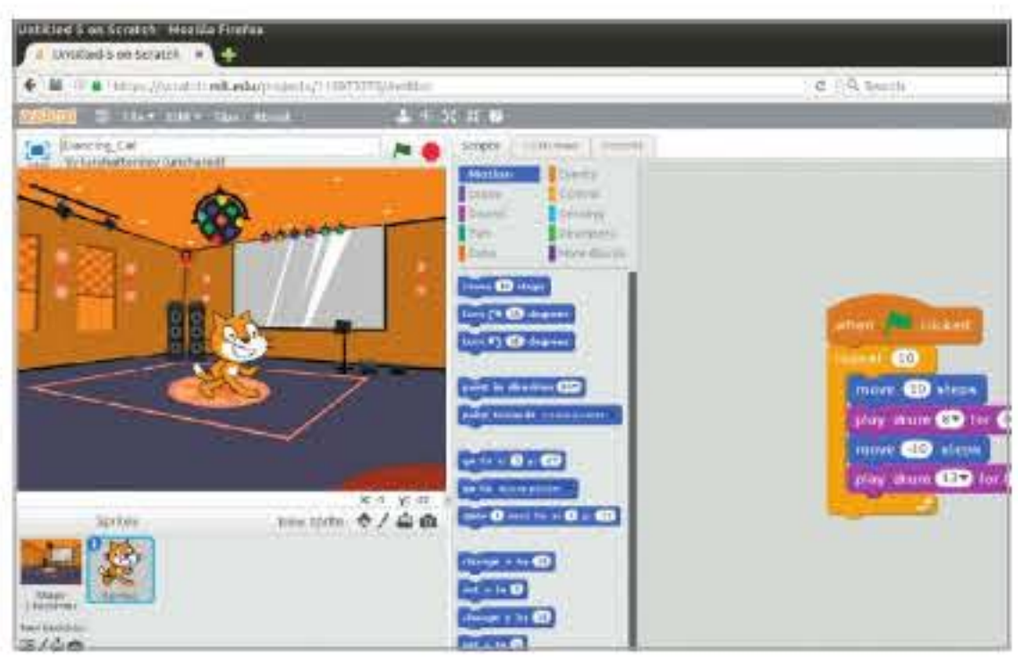


STEP 2 You can drag and rearrange the block scripts to any part of the Script Area. We like to have our `when [flag] clicked` scripts in the top left but it really doesn't matter where they are. It's also worth spotting that we now have more than one script for Sprite1; you can have multiple scripts for each sprite in your program.

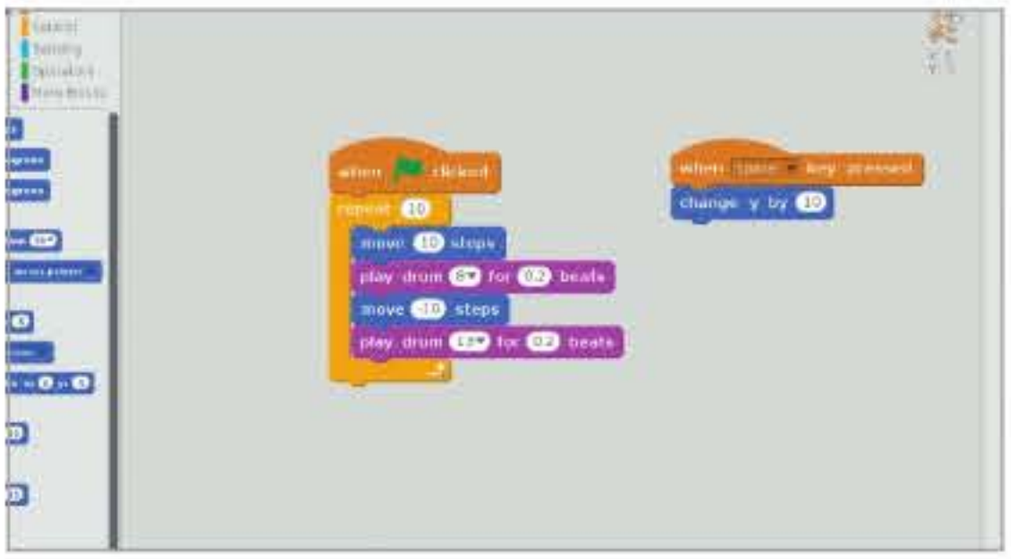
STEP 3 We're going to make Scratch Cat jump up and down when we press the space bar. Click Motion and drag `change y by [10]` and clip it to the `when [space] key pressed` block. What's with the "y"? This is what's known as a "coordinate".



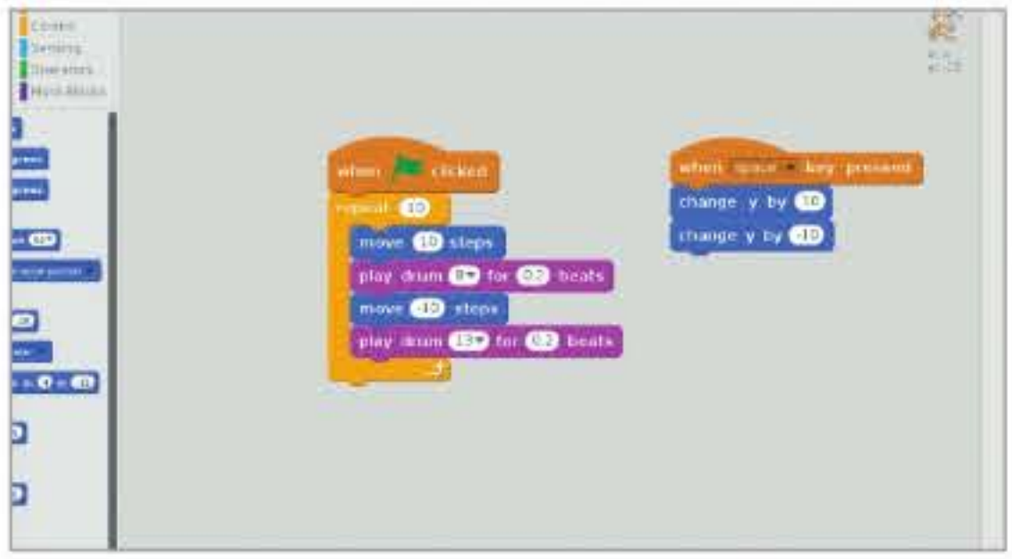
STEP 4 The position of each sprite on the stage is shown using two variables, x and y. These are referred to as the "coordinates". The x is the sprites horizontal position on the stage whilst the y coordinate is the vertical position. Click and drag the sprite around the Stage and you'll see the x and numbers change.



STEP 5 The centre of the Stage is x: 0 and y: 0. As you move the sprite up and to the right the numbers increase and as you move it left and down they decrease (going into negative numbers). So when we use the `change y by [10]` block it says, take the current value of y (the vertical position) and increase it by 10. That makes our cat jump up.



STEP 6 What goes up must come back down. So drag another `change y by [10]` block and attach it to the bottom of the `when [space] key pressed` script. Now change `[10]` to `[-10]`. Click the Green Flag and run the program. Now press the space bar and… oh no, nothing happens. We've just encountered our first "bug".

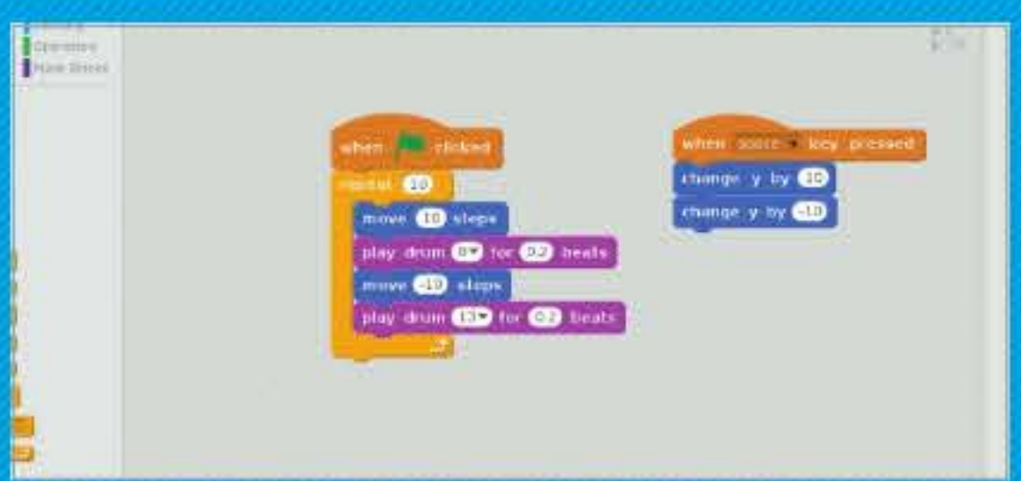


FIXING YOUR SCRIPT

We know that there's something wrong with our script and we want to see Scratch Cat jump when the space bar is pressed. So let's quickly squash this bug and see it working.

STEP 1 The problem is that programs are super fast and highly visual programs like Scratch can move in the blink of an eye and that's what is happening here. If you tap the space bar repeatedly while the program is running you'll see Scratch Cat flickering as it jumps up and down.

STEP 2 The challenge is that our motion controls move the cat instantly from one place to another, so fast that we can't see. Sometimes this is fine, like our back and forth dance, but obviously we need to slow down the jump. Help is at hand. Click the Motion tab to view the Motion blocks.



STEP 3 Drag a `wait [1] secs` block from the Blocks Palette and insert it underneath the `change [y] by 10` block. Now press the space bar on the keyboard to see Scratch cat jump up, and then back down. Notice that you don't need to press the Green Flag icon to run the program; the Green Flag starts our other script.

STEP 4 We think Scratch Cat stays in the air a bit too long. We want a jump, not a levitation effect. Change the `wait [1] secs` variable to `[0.25]`. This is a quarter of a second and will give us a more fun hop. Press the Green Flag to start the script and tap the space bar whenever you want Scratch Cat to jump.

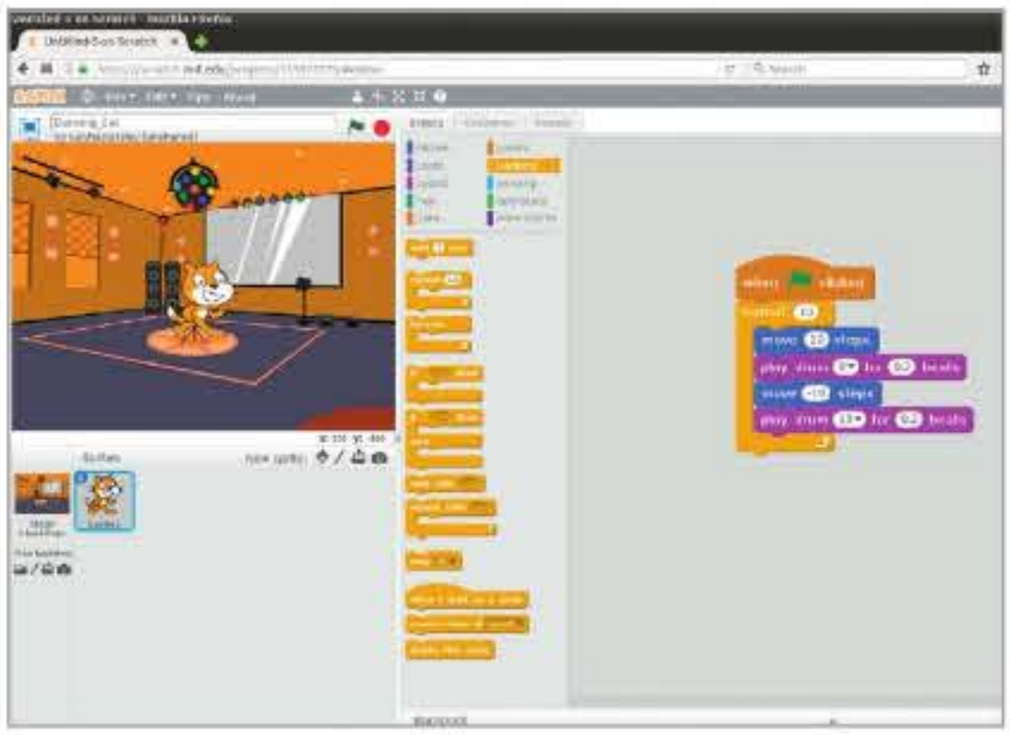
Using Sprites in Scratch

So far we've just got the one sprite in Scratch, the eponymous Scratch Cat. In this tutorial we're going to add a second sprite and see how to make the two sprites interact with each other.

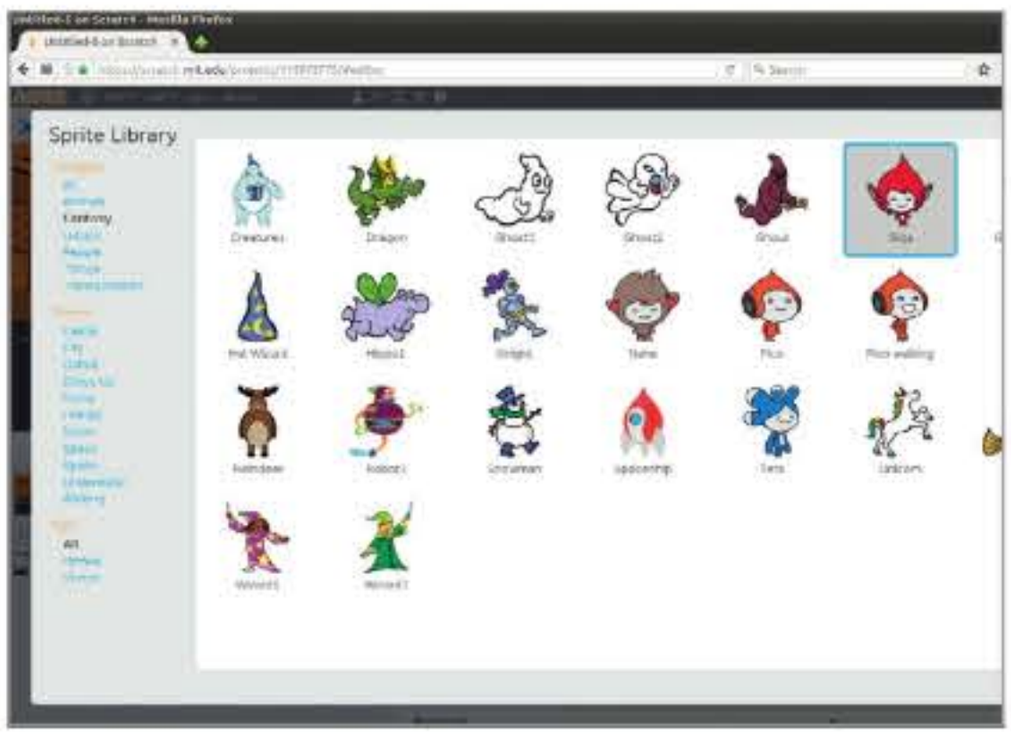
LOOK SPRITE

Sprites are 2D (flat) graphics drawn on top of a background. They are commonly used to display information in games such as health bars, scores or lives. Older games are composed entirely of sprites, just like our Scratch project.

STEP 1 We're going to add another sprite to our project and a second character to the scene. Click the Choose Sprite From Library button, just above the Sprites pane. This opens the Sprite Library that displays all the different characters available.



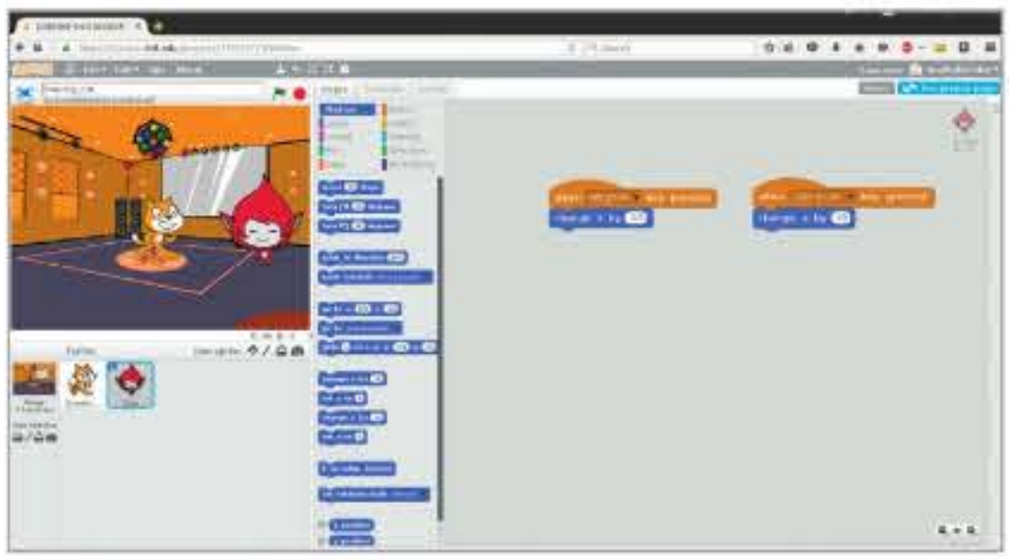
STEP 2 Click on the Fantasy link in the sidebar and choose Giga. Click OK to add the character to the stage. Click and drag the sprite to reposition Giga to the right of Scratch Cat. Notice that a Giga icon has joined Sprite1 in the Sprites pane.



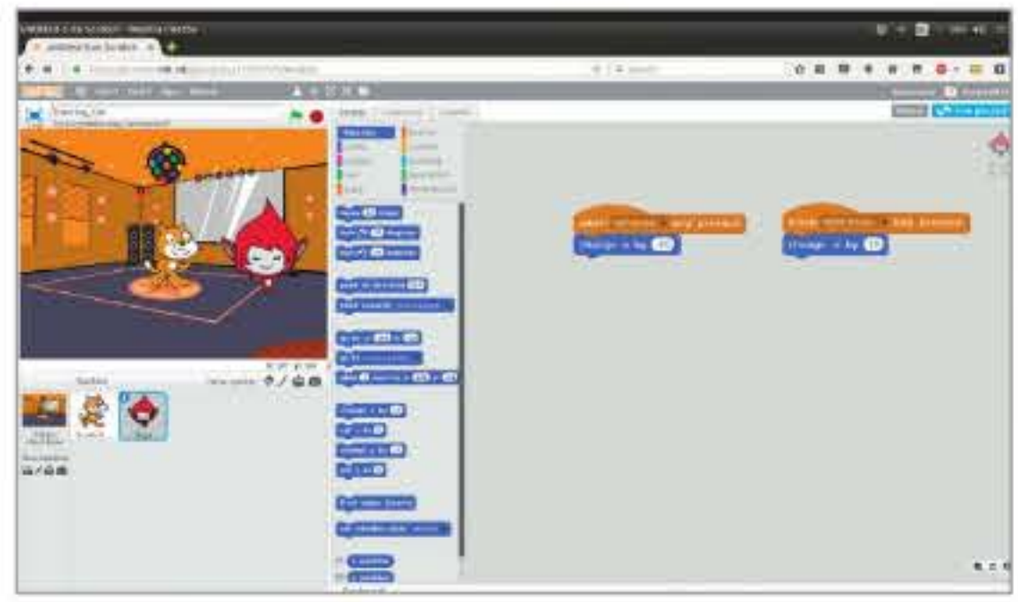
STEP 3 All the blocks on the Script Area have vanished. The scripts we built for Scratch Cat relate to that object, not to our new sprite. Click on Sprite1 in the Sprites pane to view the Scratch Cat scripts again. Then click Giga to return to your Giga character.

STEP 4 Now that we have more than one sprite, it's a good idea to name them. Click the "i" icon next to Sprite1. Change Sprite1 in the text box to Scratch Cat. Take a look at the other options here. You can remove a sprite from the stage by unticking the Show checkbox, without deleting it from the project.

STEP 5 Select Giga and choose Events. Drag a `when [space] key pressed` and change `[space]` to `[right arrow]`. Add a `change x by [10]` block beneath. Shift-click the script and choose duplicate to create another. For the second script change the when `[right arrow] key pressed` to `[left arrow]` and `change x by [10]` to `[-10]`.



STEP 6 We're quite sure you can see where this is going. Press the Green Flag icon and our Scratch Cat object will start to dance and still jumps with space, whilst our other object, Giga, can be moved left and right using the arrow keys on our keyboard.



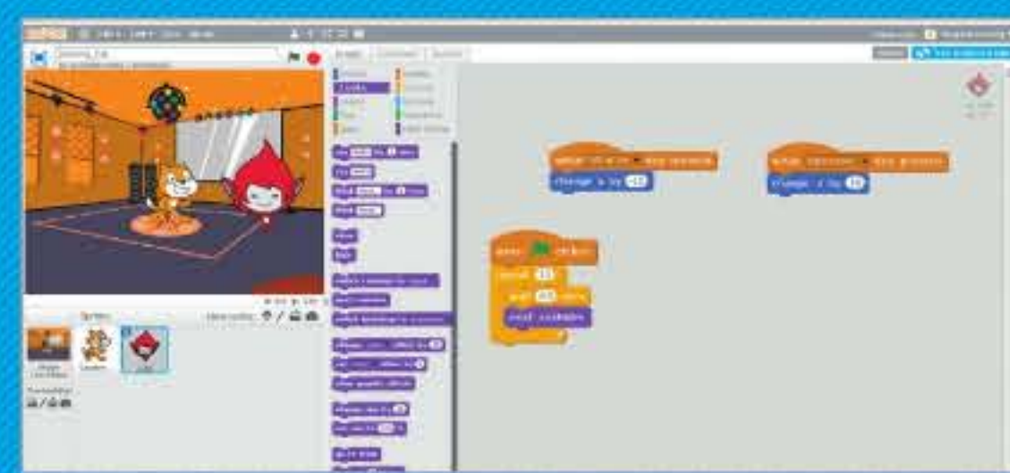
CHANGING COSTUMES

Our two objects, Giga and Scratch Cat don't have to look like the original characters. That's just the name we've given to each sprite. The visual appearance is a costume and our objects can change their costume and look completely different.

STEP 1 Select Giga in the Sprite Pane and click the Costumes tab. The Scripts Area now displays the costumes being used by Giga, including the current look. Choose giga-c in the list on the sidebar; this gives our sprite a different pose. Switch back to giga-a for now.

STEP 2 Let's use costume changes to animate Giga. Click Events and drag `when flag clicked` block to the Scripts Area. This block will activate at the same time as the Scratch Cat scripts when the Green Flag icon is clicked.

STEP 3 Attach a `repeat [10]` block and inside it place `wait [1] secs`. Change `[1]` to `[0.5]`. Click Looks and drag a `next costume` block into the `repeat` block. Now Giga will switch to the next costume every half a second, creating an animation effect.



STEP 4 Click the Green Flag icon to start the animation. Scratch Cat now moves back and forth, tapping out a beat, and Giga animates through four different poses. You can move Giga left and right using the arrow keys. It's starting to form into an interactive scene.

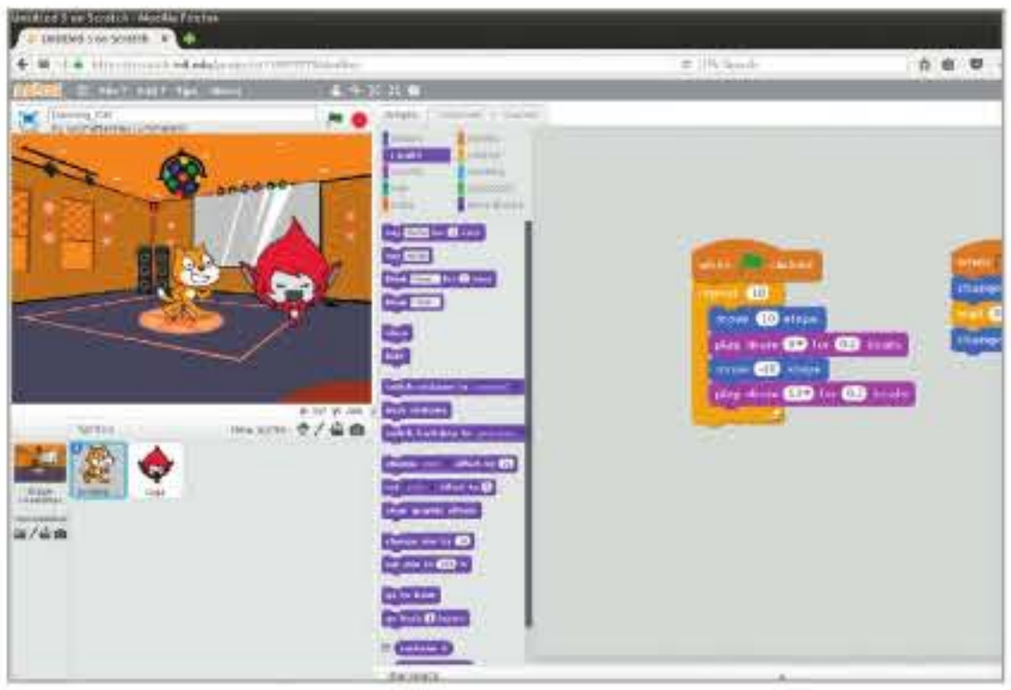
Sensing and Broadcast

Get your sprites and scripts to communicate with each other. The concept of Sensing and Broadcast is similar to the way objects communicate in Python. This tutorial will walk you through the process.

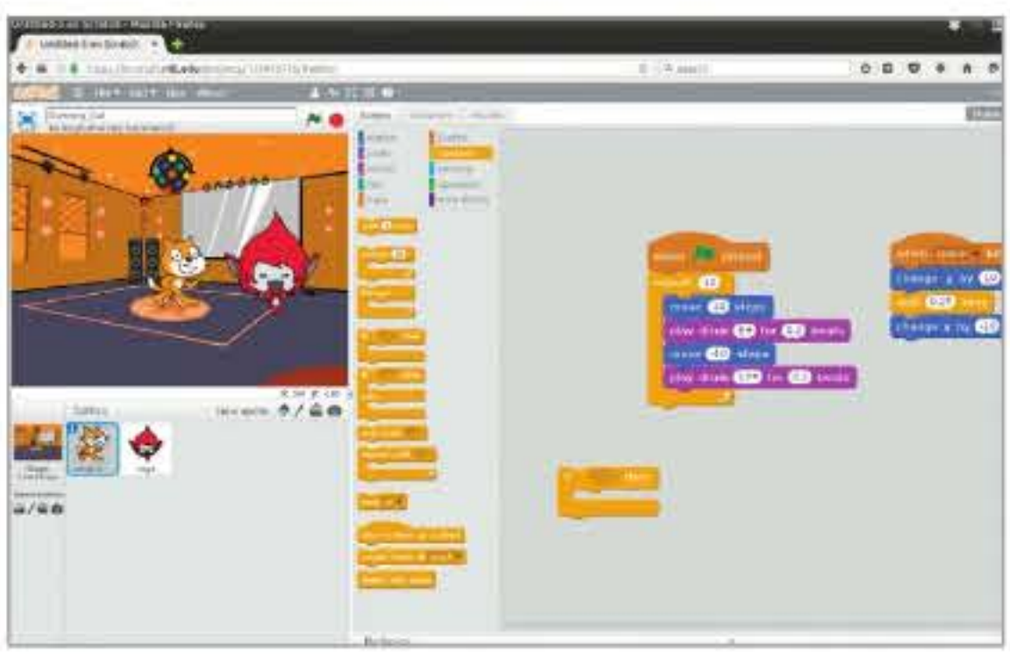
MORE INTERACTION

Earlier we looked at how you could interact with scripts, using the keyboard to move the sprites but scripts and sprites can interact with each other, sending messages and responding to events.

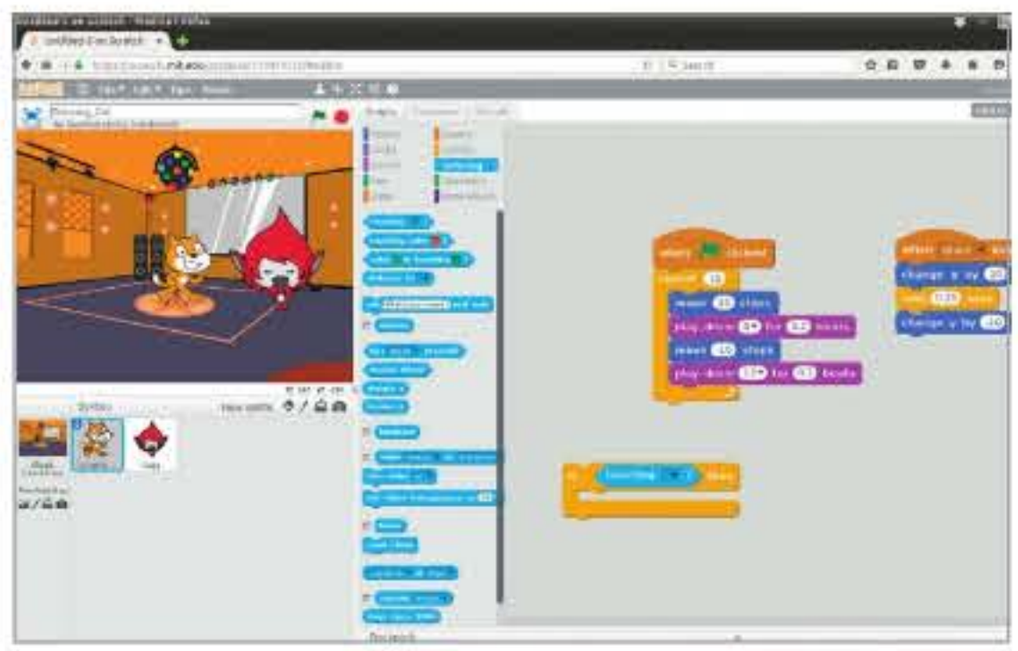
STEP 1 Scripts can broadcast messages to each other. These can be used to start other scripts or respond to events. You're going to get Scratch Cat to respond to the Giga and say "Watch Out!!!" if the two sprites touch. Select Scratch Cat in the Sprites Pane so you can view its Script Area.



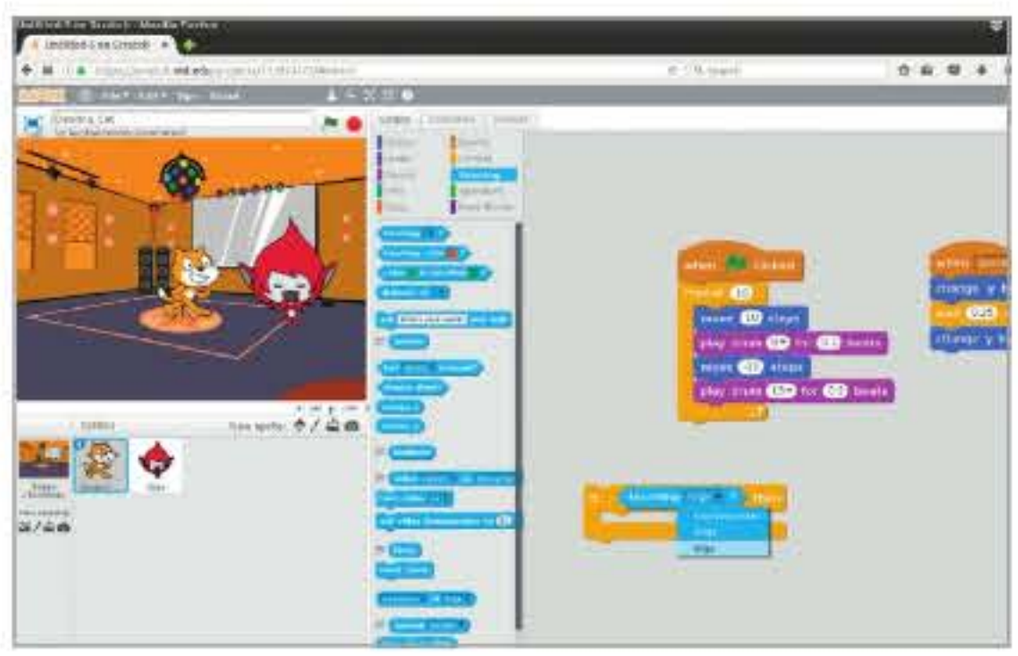
STEP 2 You 've looked at 'if' and 'else' in Python, so now you need to look at them in Scratch. Programs work around a few simple terms and 'if this then do that' is one of them. On our stage, you want to say, "if Giga touches Scratch then Scratch says 'Watch Out!!!'" Click Control and drag an `if` block to the Script Area.



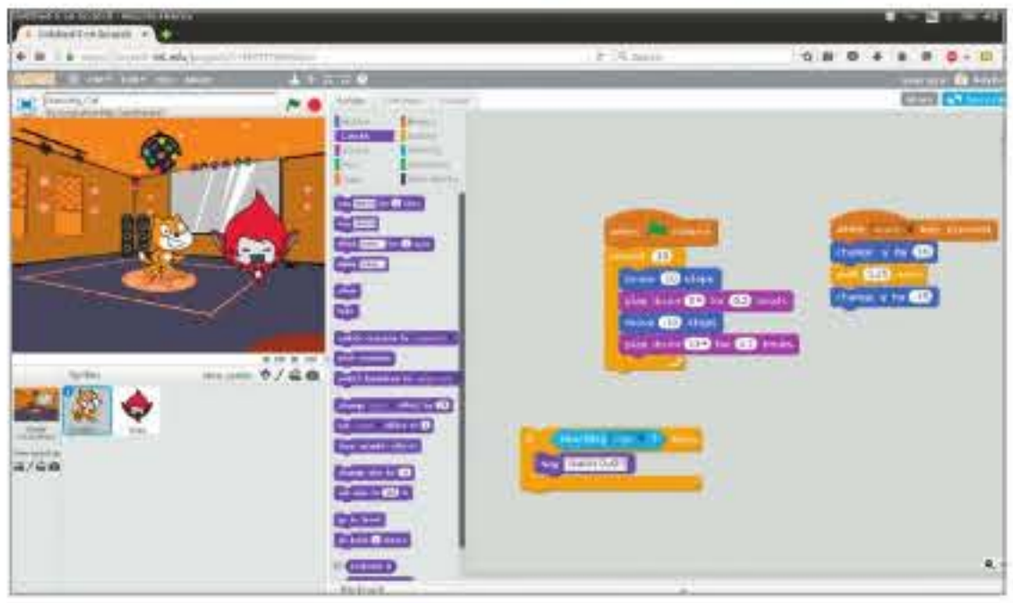
STEP 3 The options for interaction between the sprites are found in the Sensing part of the Block Palette. The one we are looking for is touching, at the top. Notice that this is a different shape to ones you are used to. It is designed to fit in the slot next to our `if` block. Drag the `touching` block into the spare slot in the `if` block.



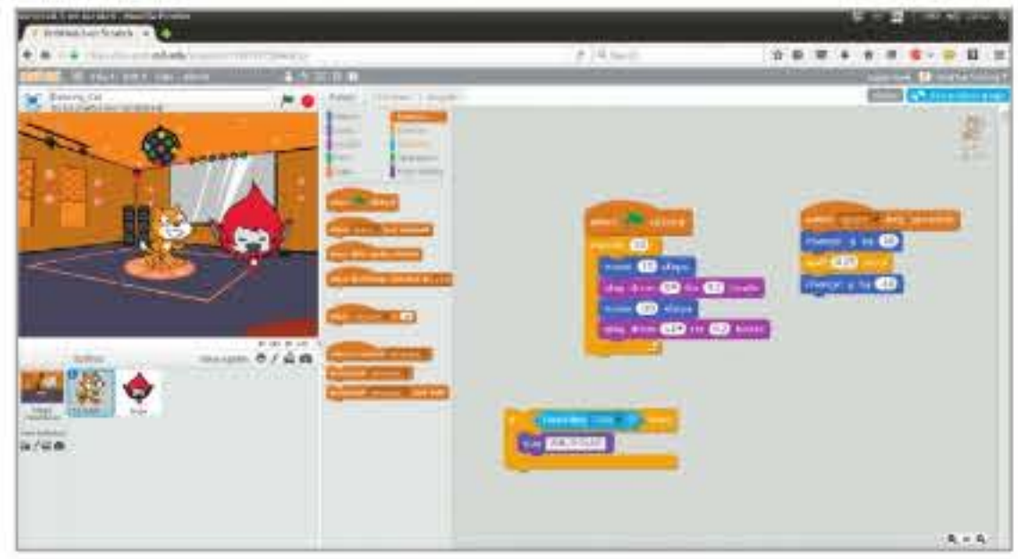
STEP 4 The `touching` block responds to the condition of our sprite; in this case if it's touching another sprite. However, we need to tell it which one. Click the arrow in the touching block and choose Giga from the list. There are a couple of default options, mouse pointer and edge. These are handy if you're making games.



STEP 5 We've got our `if`, what we now need is a response. Click Looks and drag a `say [Hello!] for 2 secs` block and insert it inside the `if` block. Change `[Hello!]` to `[Watch Out!!!]`. Try to run the program. Nothing happens! That's because our `if` script isn't part of the `when flag clicked` script.



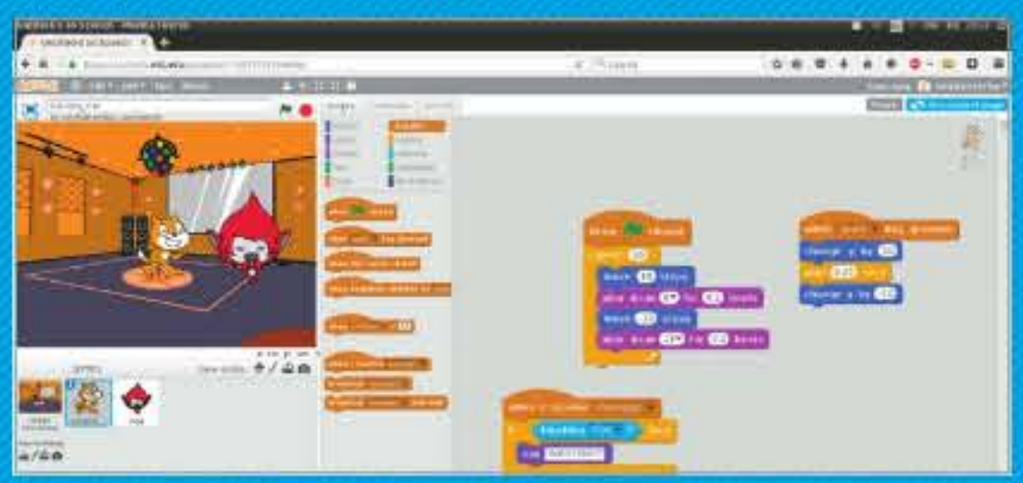
STEP 6 How do we get our 'if' script to run alongside the other scripts? We could put the `if` block inside `when flag clicked` but it would look big and ugly and do two different things. We could add a second when [flag] clicked block to the `if` block but that would only work if they were touching at the start. The answer is to use a `broadcast` block.



BROADCASTING

The broadcast block enables one script to interact with the others. It can be used to tell scripts to run and is ideal for bigger projects where each sprite does several things at once.

STEP 1 Click the Events tab in Scripts and look for the `when I receive` block. Drag this and connect it to the top of the `if` block. We need to set the receiver, i.e. the message it gets from the other script. Click the arrow next to receive to reveal the Message name window.



STEP 3 Now we need to broadcast that 'init' message from our when [flag] clicked script so it runs at the same time. Drag a `broadcast` block to the Script Area and insert between the `when flag clicked` block and the `repeat [10]` block. Now click the arrow in `broadcast` and choose 'init'.

STEP 2 You can call the message anything you want but it should start with a lowercase letter and in this instance you are going to use the word "init". This stands for "initialize" and getting the lingo right now will make your life much easier when you move to a more complex language. Click OK.

STEP 4 Our program is almost ready but our `if` script only works once, when the Green Flag icon is clicked. We're going to place it inside a Forever block. This is okay though because it's not our main program and we are also going to add a `stop all` block to the end of our `when flag clicked` block. Click the Green Flag to see your program run.

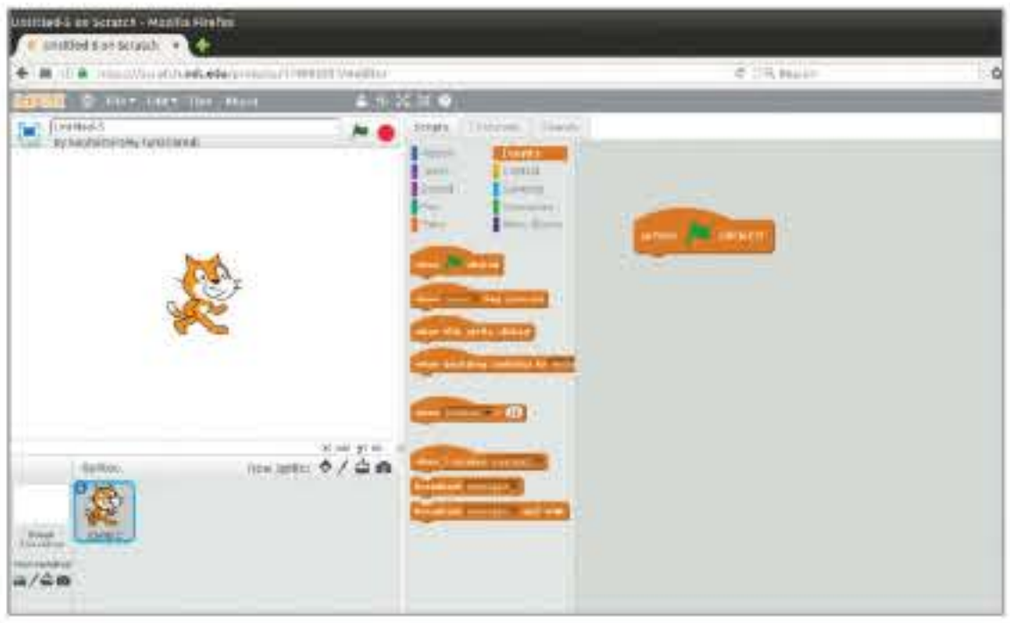
Objects and Local Variables

You've already met variables but Scratch makes them easy to understand. What isn't so obvious in Python is how variables are stored locally to objects. This is where Scratch helps. You're going to create a dice game to help understand this.

ARE YOU LOCAL OR GLOBAL?

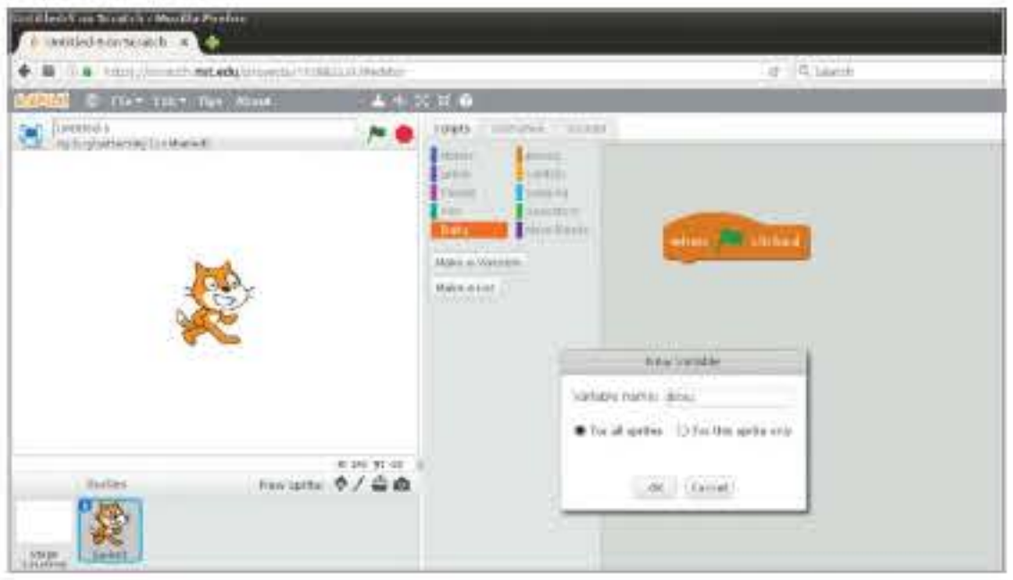
Objects, like your Scratch Cat sprite, can have their own variables. This could be the player score or the amount of ammunition left. These are stored inside the object and are known as "local". Variables used by all objects are known as 'global'.

STEP 1 You're going to leave the disco behind. Choose File > New. You start with a blank stage containing a single Scratch Cat sprite. Click Control and drag a `when flag clicked` block to the Script Area.

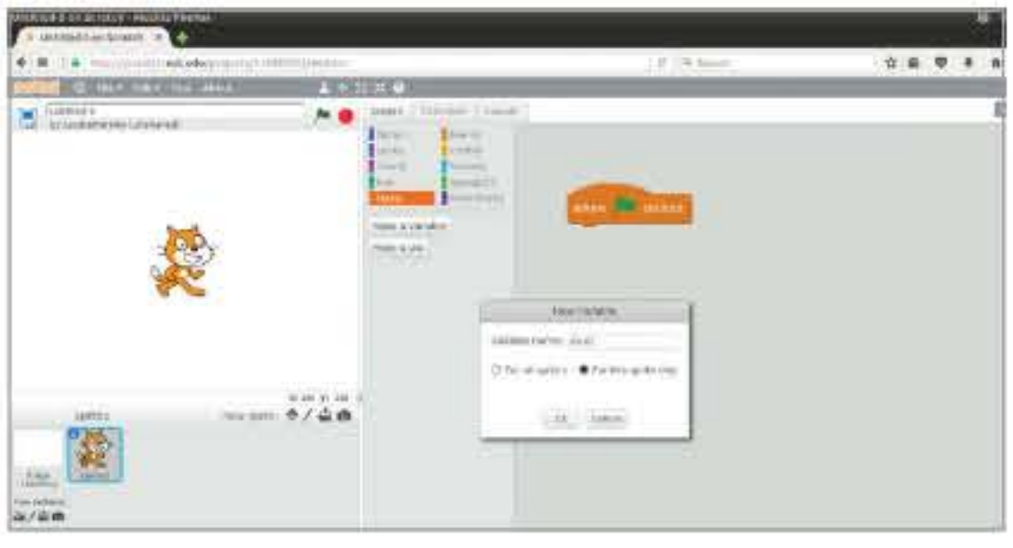


STEP 2 You're going to create a simple game where Scratch Cat rolls two dice and wins if they're both the same number. Click Data. Unlike other sections there are no blocks here; instead we have to create the variables we need. We need two, one for each dice.

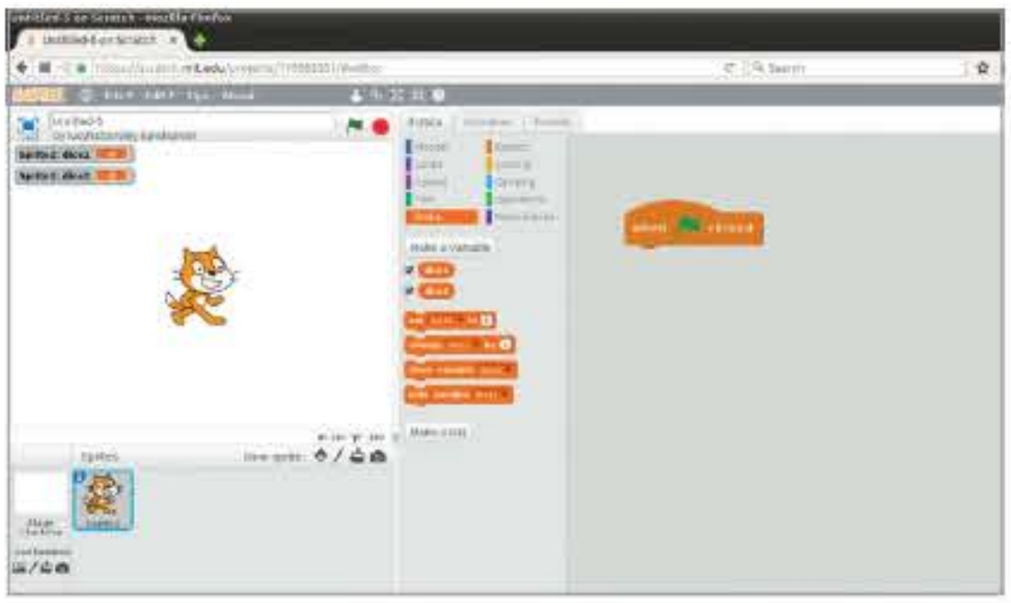
STEP 3 Click Make a variable and enter `dice1` into the ? window making sure that your variable starts with a lowercase letter. There are two options here: `For all sprites` and `For this sprite only`. `For all sprites` allows every sprite to use the dice; this is known as a global variable.



STEP 4 `For this sprite only` means that only this sprite can use the two dice variables. this is known as a local variable. This is useful if you want to create another character with their own set of dice and play against each other. We're doing that in the next tutorial, so choose For this sprite only and click OK.



STEP 5 A whole bunch of blocks appears in the Block Palette. We can now use our dice1 variable but we want two dice, so click `Make a variable` again and this time enter `dice2`. Remember to choose `For this sprite only` and click OK.



STEP 6 Both the dice1 and dice2 variables are currently empty. They could be anything we wanted, but we want them to be a random number between 1 and 6. Drag the `set [dice1] to [0]` block and click it underneath the `when flag clicked` block. Drag another `set [dice1] to [0]` block underneath and change the `[dice1]` setting to `[dice2]`.

SMOOTH OPERATORS

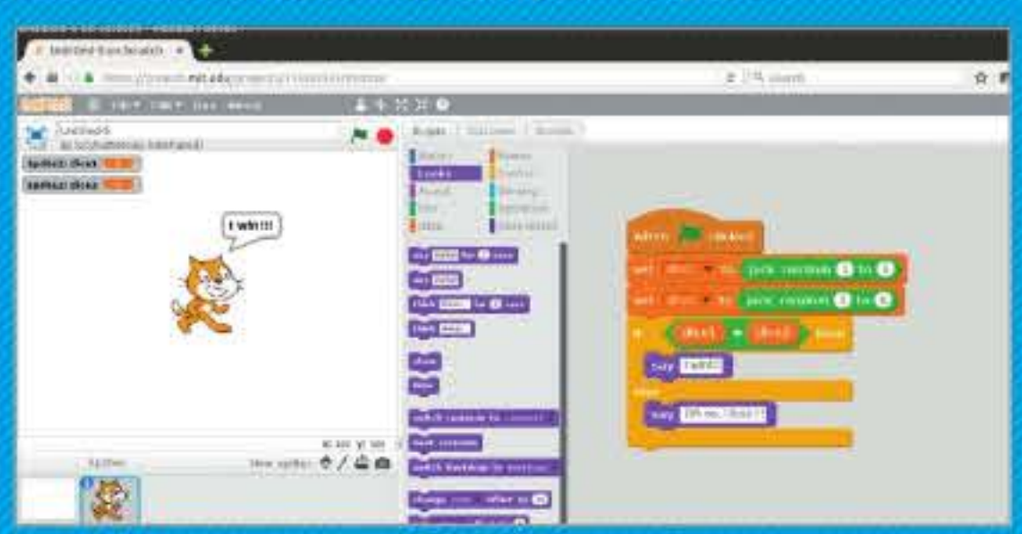
Operators are used to change the values of variables. Some of these will be familiar; you've used the addition operator '+' to add two numbers together. Programs can also check if numbers are equal, bigger or smaller than each other or even not equal.

STEP 1 Click the Operator tab and drag a pick `random 1 to 10` block and drop it into the `[0]` in `set dice1 to [0]`. Change the `[10]` to a `[6]` so it reads `set dice1 to pick random [1] to [6]`. When we click the Green Flag it will pick a number between 1 and 6 and store it in the dice1 variable. Add a `pick random [1] to [10]` block to dice 2 and also set it to `[6]`.

STEP 2 We need to check the two dice. Click the Control tab and drag the `if else` block to the script. This block is like the `if` block we used before but it says, "if this happens, do this; if not, do this instead." Now click operators and look for the = block. Drag it into the space next to `if`.

STEP 3 The = operator checks if two things are the same but we need to tell it to check our variables. Click Data and drag `dice1` to the space on the left of the = block. Drag `dice2` to the space on the right of the = block.

STEP 4 Finally click the Looks tab and drag `say [Hello!]` blocks into the `if` and `else` spaces. Change the say `[Hello!]` text in `if` to `[I win!!!]` and in `else` to `[Oh no. I lose!]`. Click the Green Flag to play the game.



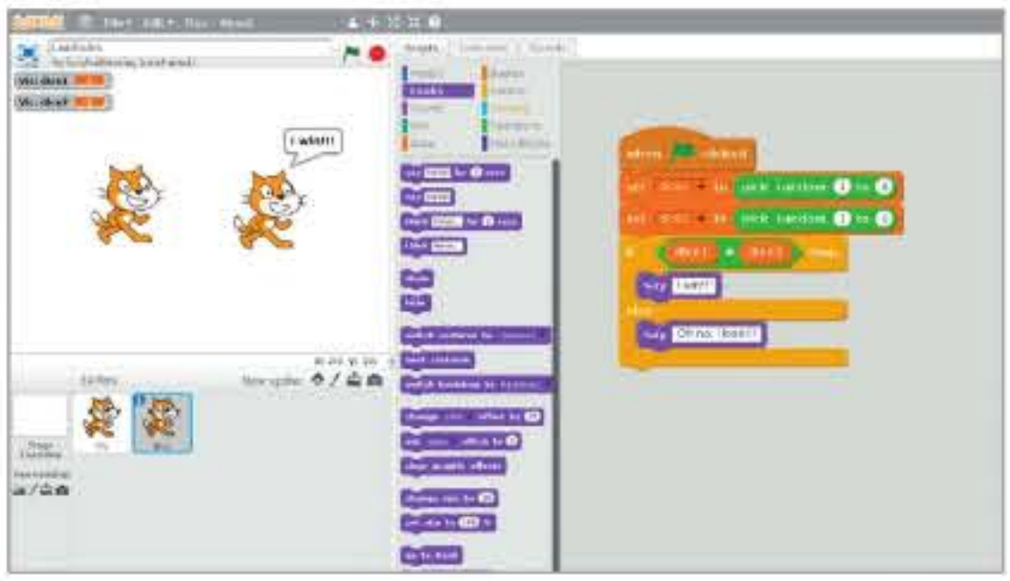
Global Variables and a Dice Game

We're going to create a game where two sprites play dice with each other and the sprite with the highest score wins. This lets us examine the idea of more than one object, each with its own set of local variables.

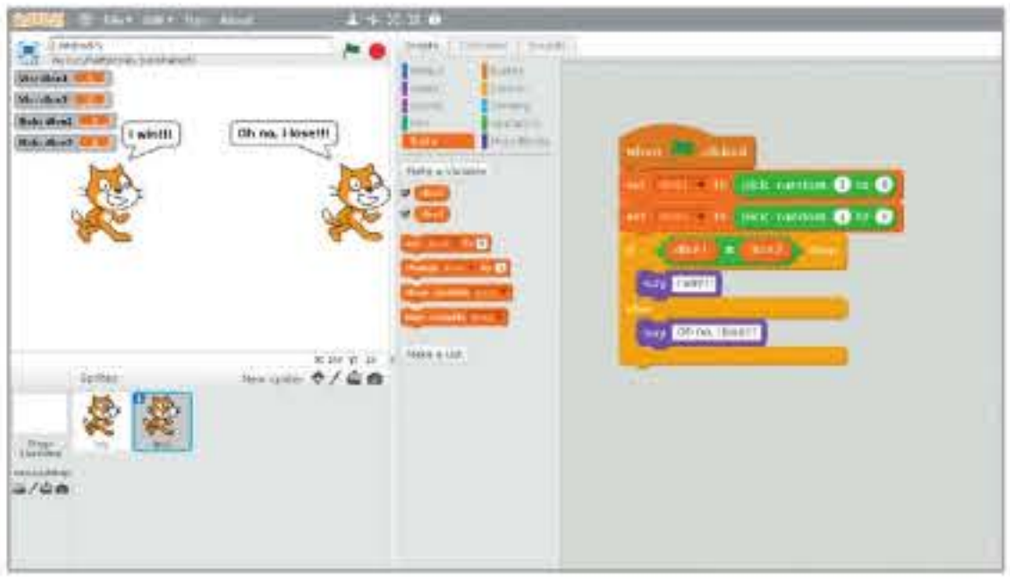
DOUBLE TROUBLE

We're going to start with the dice game from the previous tutorial, but create a second character. The fancy OOP word for this is "instantiation": creating an instance of something – in this case another instance of Scratch Cat.

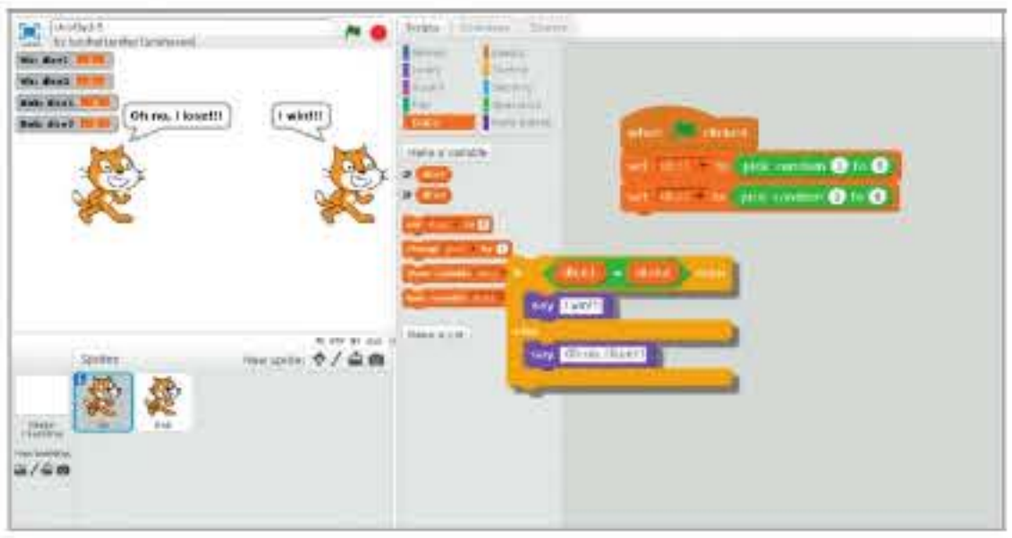
STEP 1 Let's add a second character to the Stage. Shift click on Scratch Cat on the stage and choose Duplicate. Click OK. Rearrange the two characters on the stage so they're stood side by side. Use the Sprite Info Window to give them names: "Vic" and "Bob".



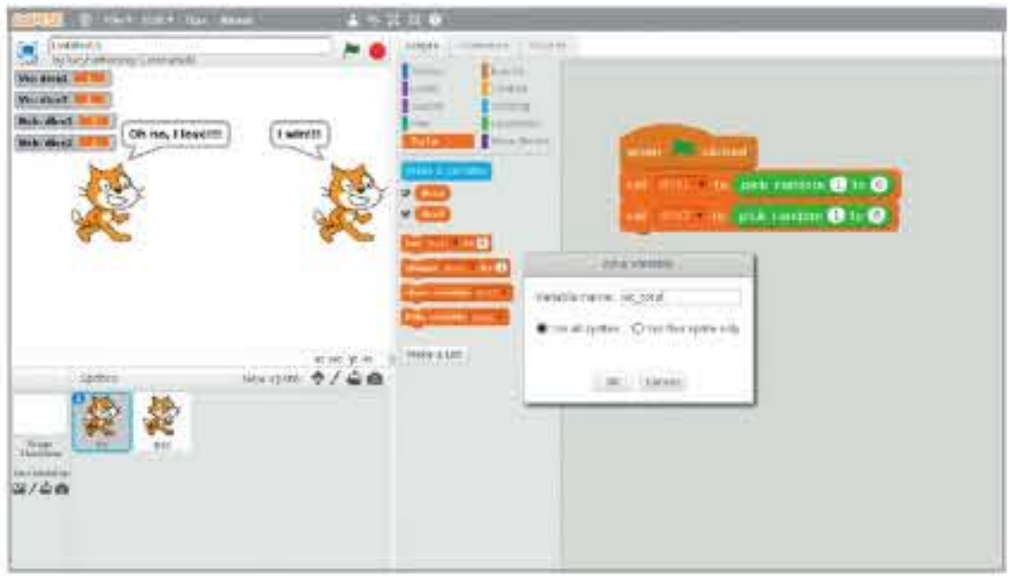
STEP 2 Select Bob and click Data in the Script Palette and you'll see dice1 and dice2. Bob received his own set of dice when we duplicated him but he doesn't share dice1 and dice2 with Vic. Place checks in the boxes next to dice1 and dice2 so you can see them on the Stage.



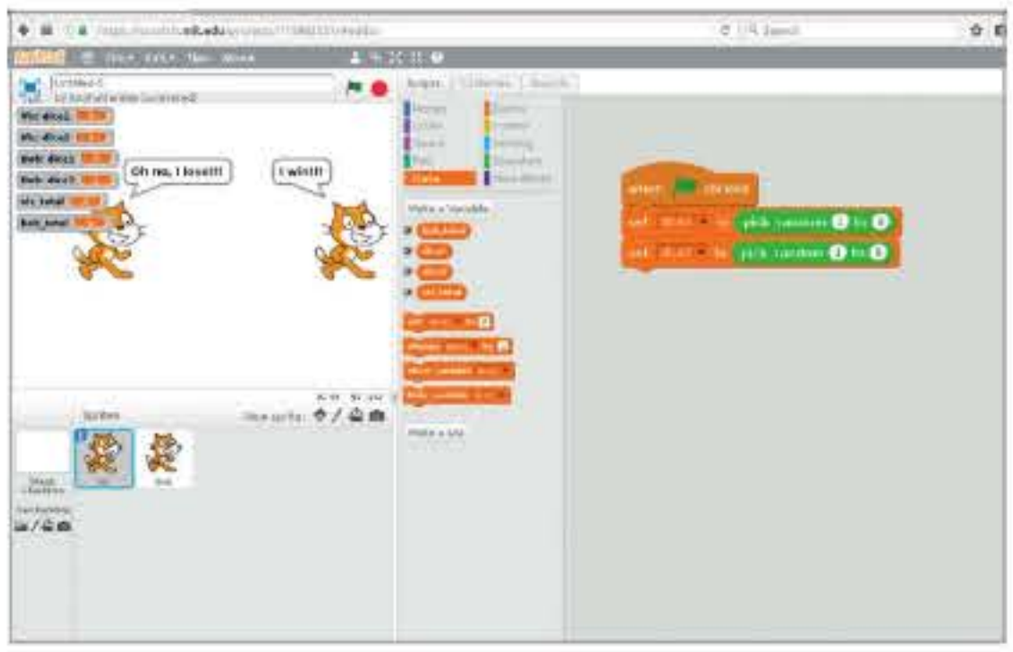
STEP 3 You can check this out by clicking the Green Flag icon. Vic and Bob will both roll dice1 and dice2 but each gets its own random result (shown in the Stage). We're going to change the game to one where the two dice are added together (highest score wins). Disconnect the If block from the script and drag it to the Block Palette.



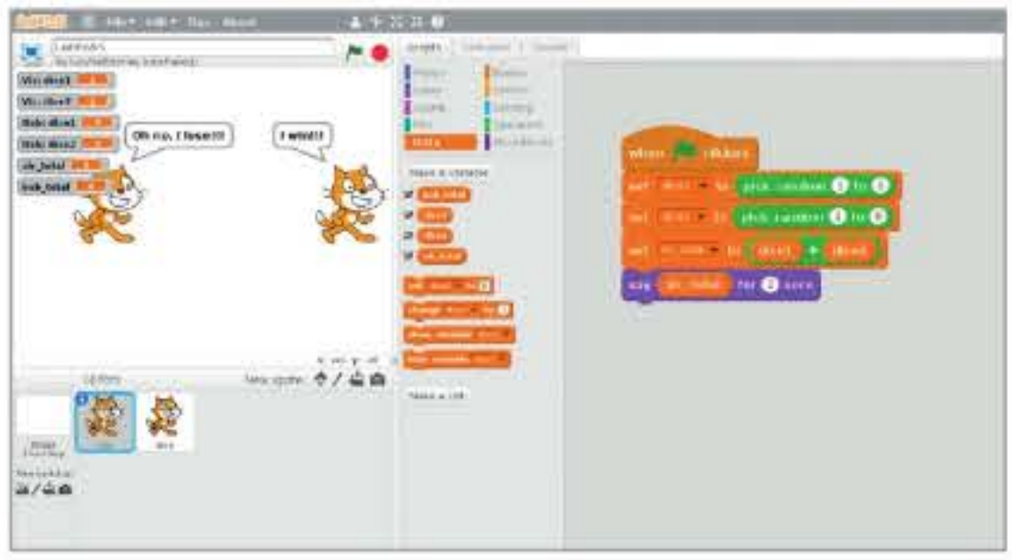
STEP 4 We need two new variables: one for Vic's total, and one for Bob's. Click `Make a variable` and enter `vic_total`. This time choose `For all sprites` then click OK. With Vic still selected, click `Make a variable` again and enter `bob_total`. Click OK.



STEP 5 Imagine both characters writing their totals on a blackboard that they share, but each has its own pair of dice. The totals are global and shared across both characters; the dice are local to each object.



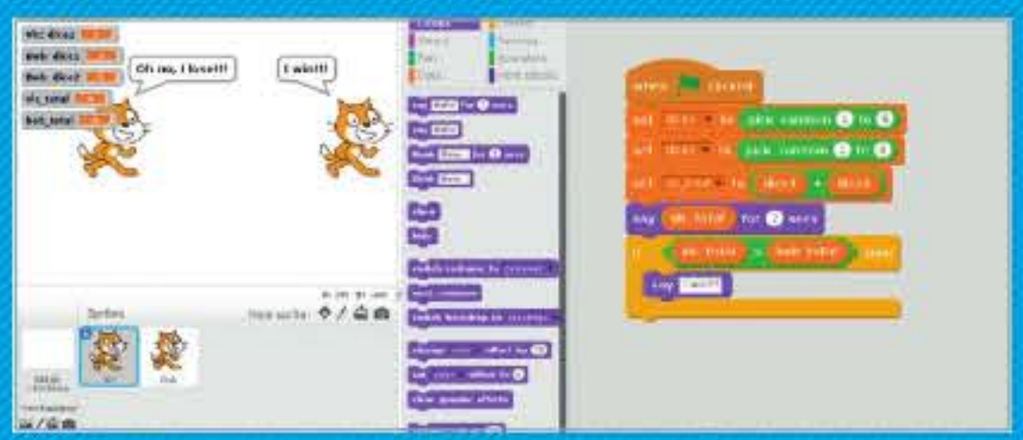
STEP 6 Drag `set [dice2] to [0]` from the block palette and connect it to the script. Change `dice2` to `vic_total`. Now click Operators and drag the `+` block to the `[0]`. Click Data and drag `dice1` to the left side of the `+` block and `dice2` to the right side. This adds up the dice1 and dice2 variables, and stores the combined value in vic_total.



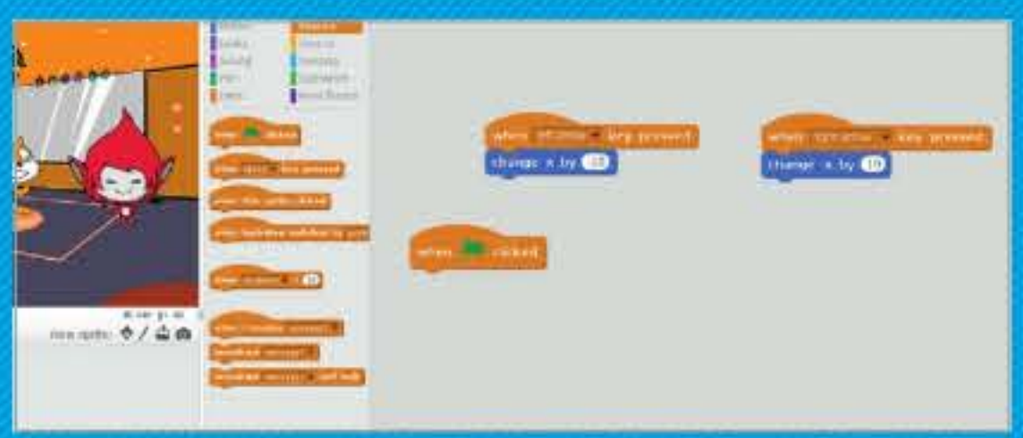
GREATER THAN

The sprite with the highest score is going to win our game. This is decided using the greater than symbol >. This sits between two numbers, i.e. 3 > 2 and lets you know if the number on the left is bigger than the one on the right.

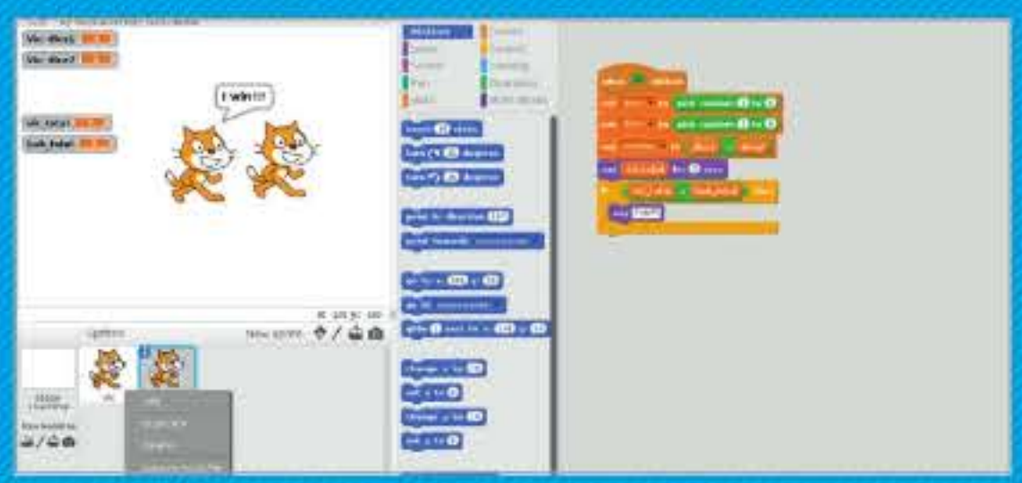
STEP 1 Both sprites are going to announce their score and the one with the highest score will say, "I win!". Click Looks and drag `say [hello] for 2 secs`. Now click Data and drag `vic_total` to replace `[hello]`. The first part of the game is ready, we're going to use an `if` block with an `>` operator for the next part of the game.



STEP 2 Click Control and drag an `if` block to the script. Now click Operators and drag the `>` operator to the slot in the `if` block. Click Variables and drag `vic_total` to the left of the `>` block and `bob_total` to the right. Finally click Looks and drag a `say [hello!]` block inside the `if` block and change the text to `[I win!]`.



STEP 3 Bob needs to run the same script, only with bob_total in place of cat_total. We could write Bob with the same code but the point of objects is that you can stamp out copies. Shift-click Bob and choose Delete. Now Shift-click Vic and choose Duplicate. Click the Info icon and rename Vic2 as Bob.



STEP 4 You need to change Bob's variables. Change `set vic_total` to `set bob_total` and `say vic_total` to `say bob_total`. Finally swap around the `vic_total` and `bob_total` in the `if` block. Now click the Green Flag icon to play the game. Vic and Bob role their dice, and the winner is announced.

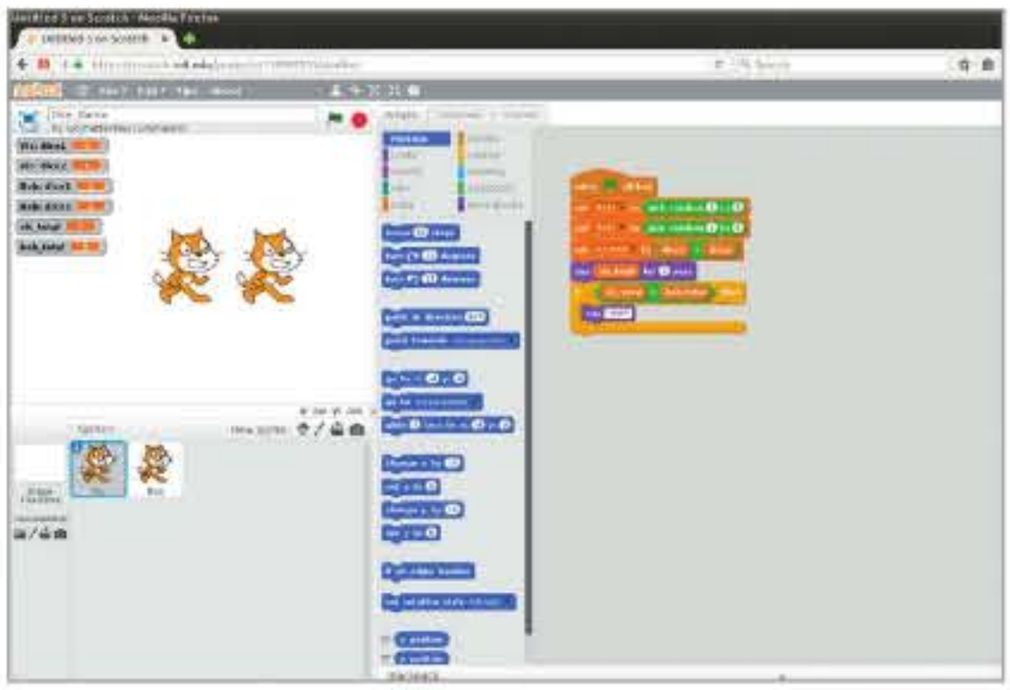
Classes and Objects

Modern coding uses a style called Object Orientated Programming, or OOP for short. In OOP you group together variables and functions into small blocks of code called objects. These objects then make up your program.

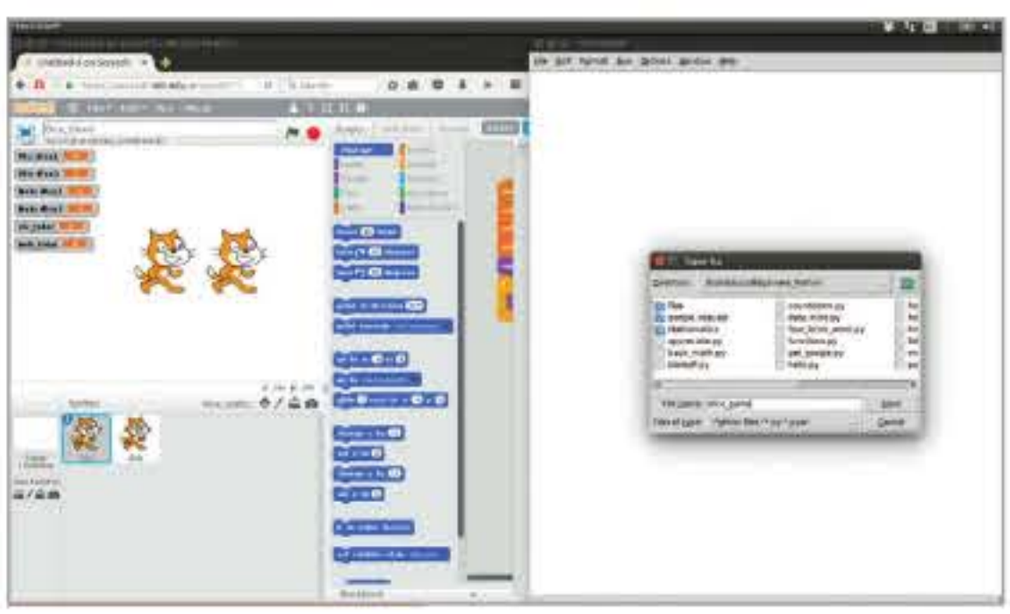
SCRATCH THAT

OOP can be hard to explain, but makes sense when you start using it. If you've used Scratch then you already have an idea of what an object looks like, it looks like a sprite. This is why we detoured into Scratch. It's great for learning OOP.

STEP 1 In this tutorial we're going to open the dice_game program that we created earlier in Scratch. Resize the window and place Scratch on the left-hand side of the screen. Next we're going to recreate this game in Python using objects, so you can see how objects are similar to Scratch sprites.

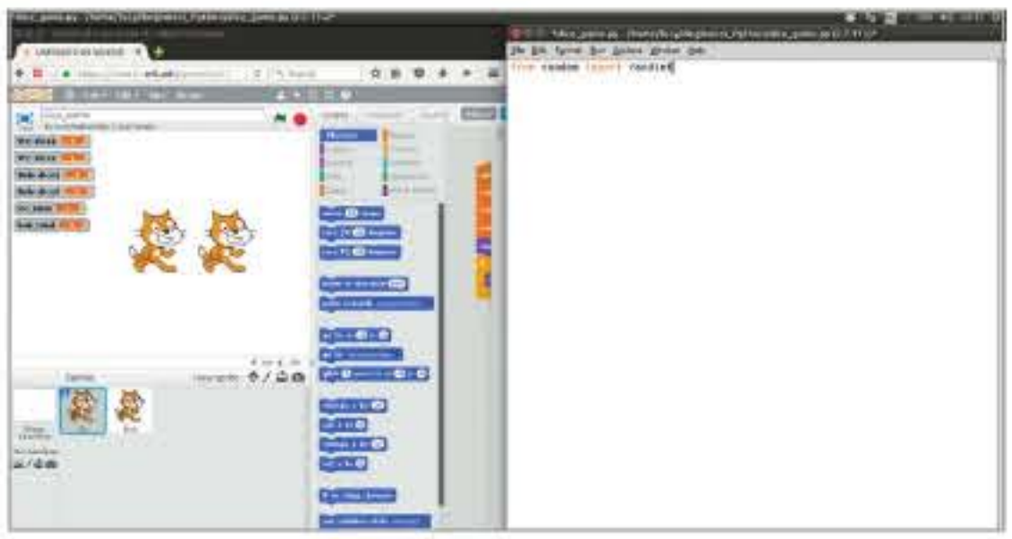


STEP 2 Open Python 2 and choose File > New Window. Resize the Editor window to the right-hand size of the screen. Choose File > Save As and name it dice_game. Now let's have a look at the objects in Scratch. We have two: Vic and Bob. Each has three variables (two dice and a total); both pick random numbers between 1 and 6 and check to see if their total is bigger than the other.



STEP 3 In OOP we don't design objects. Instead we design a blueprint for our object, called a "Class". Think of it like a blueprint or stamp. Vic and Bob are both dice-rolling cats, so we create a blueprint for a dice-rolling animal. We then stamp out two identical objects from that blueprint. One called "Vic" the other called "Bob". We're going to need the random number module, so enter this line:

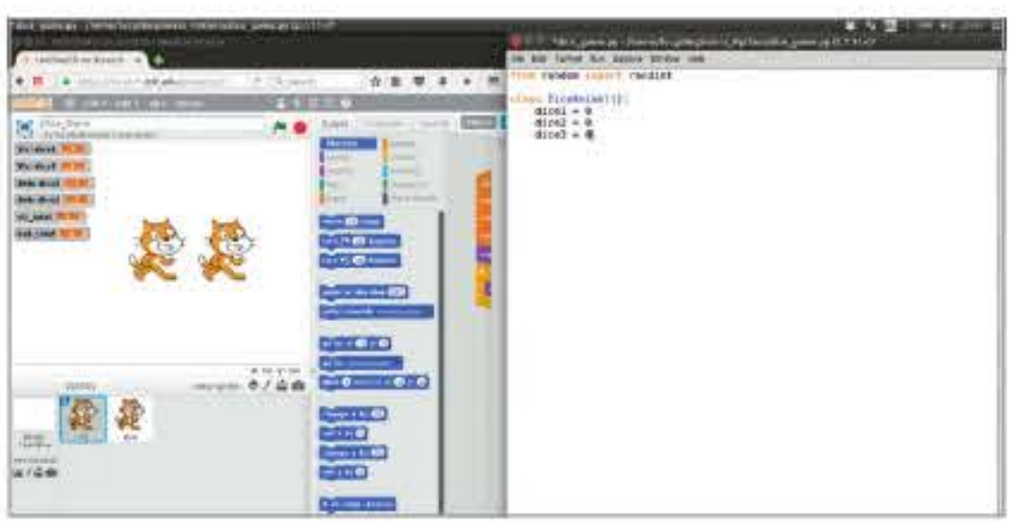
```
from random import randint
```



STEP 4 Now let's define our class, which we're going to call DiceAnimal. Enter:

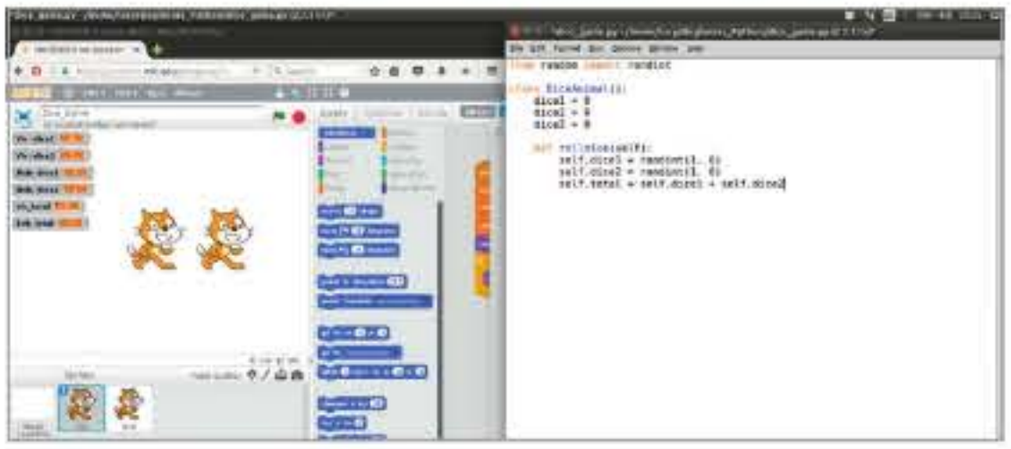
```
class DiceAnimal():
    dice1 = 0
    dice2 = 0
    total = 0
```

Notice the funny capitalisation of DiceAnimal. This is known as CamelCase and class definitions should be named in this fashion.

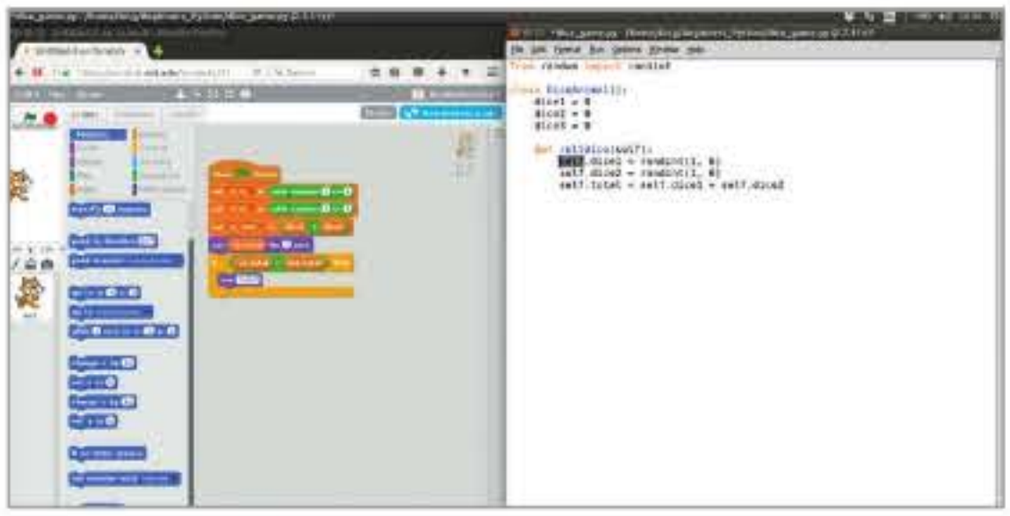


STEP 5 Now we're going to define a function that rolls both dice, and adds the two together to create the total. Inside the class, indented four lines to line up with dice1, dice2 and total, enter this:

```
def rolldice(self):
        self.dice1 = random.randint(1, 6)
        self.dice2 = random.randint(1, 6)
        self.total = self.dice1 + self.dice2
```



STEP 6 Look at Scratch, and you'll see this is the same as the set dice1 to pick random 1 to 6 block. But what are those self bits about? Remember that Vic and Bob have their own dice. Vic's dice are going to be accessed use vic.dice1 and vic.dice2 and Bob's using bob.dice1 and bob.dice2. But the class doesn't know what we're going to call each object; instead it uses "self" as a placeholder. This works no matter what name each object has.



CREATING OBJECTS

Now that our class is ready, we need to create two characters from it. One 'vic' and one 'bob'. These are known as objects, and also sometimes as instances (or 'object instance'). Because each one is an instance of the DiceAnimal class.

STEP 1 Creating an object in OOP has a big fancy name: "instantiation". Don't be impressed by the language, all it means is creating an instance of your class. And this is exactly the same as creating a variable, only instead of passing in a number, or string, you make it equal to your class. Enter this.

```
vic = DiceAnimal()
bob = DiceAnimal()
```

There, that wasn't hard at all.

STEP 2 You now have two objects, a vic and a bob. You access the variables and functions inside the object using the objects name followed by a dot. To access Vic's dice, you use
vic.dice1 and vic.dice2. We're going to get both objects to roll their dice and store the total in their own self.total. Enter this:

```
vic.rolldice()
bob.rolldice()
```

STEP 3 Now we're going to use dot notation to access the values inside both the cat and lobster. Enter this code:

```
print "Vic rolled a", vic.dice1, "and
a", vic.dice2
print "Bob rolled a", bob.dice1, "and
a", bob.dice2
```

Finally, we're going to use if, elif and else statements to create the game.

STEP 4 Enter this code:

```
if vic.total > bob.total:
        print "Vic wins!"
elif bob.total > vic.total:
        print "Bob wins"
else:
        print "It's a draw"
```

Press F5 to run the game.

Getting to Know the Command Line

Linux is an ideal platform for using Python on. It's free, Python is already integrated with it, it's open source and developed by a community of like-minded coders and freethinkers. However, before you dive into programming on Linux, it's best to learn how to manipulate the command line from where you'll be doing all your Python programming from.

142 Discovering Unix?
144 Using the Unix Filesystem
146 Listing and Viewing Files
148 Creating and Deleting Files
150 Create and Remove Directories
152 Copying, Moving and Renaming Files
154 Using the Man Pages
156 Editing Text Files
158 Getting to Know Users
160 Ownership and Permissions
162 Useful System and Disk Commands
164 Managing Programs and Processes
166 Input, Output and Pipes
168 Using Grep Pattern Matching
170 Tab Completion and History
172 Using Python Code in Linux Scripts

Discovering Unix?

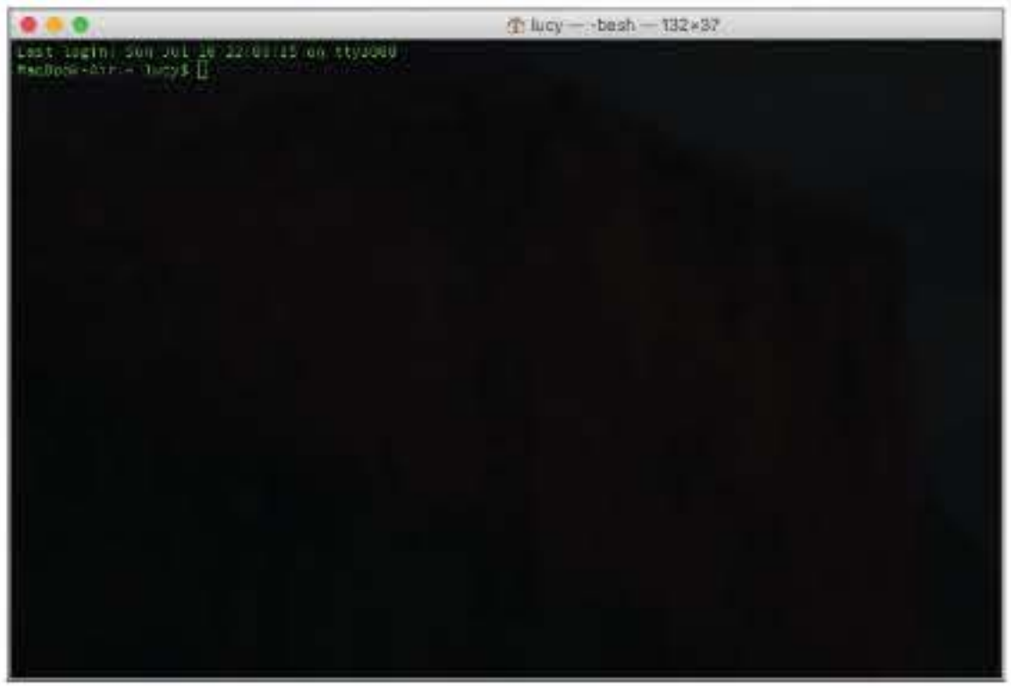
Understanding Unix and its relationship with Linux, macOS and Windows is confusing for the newcomer; terms like Shell, Terminal and Command Line complicate matters further. In this tutorial we look at what Unix actually is.

UNDERSTANDING UNIX

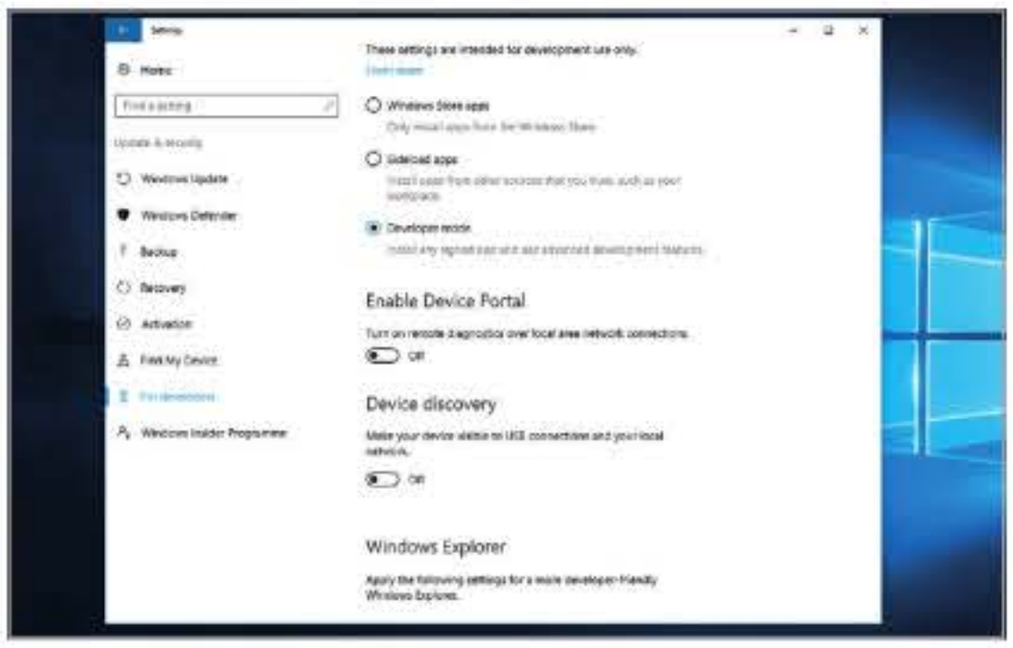
Linux is a Unix-like system and you can use Unix commands to control it. Apple's macOS is a Unix system with a flashy interface. Windows is the only major OS not based on Unix but it has a Unix interface so you can use Unix in Windows.

STEP 1 Learning how to use the command line is vital to controlling files on a computer and interaction with files is a large part of programming in Python (and other languages). Getting to Unix on a Linux OS is very easy. Click on Dash or press Super and search for "terminal". Open the Terminal app, then you should right-click and choose Lock To Launcher.

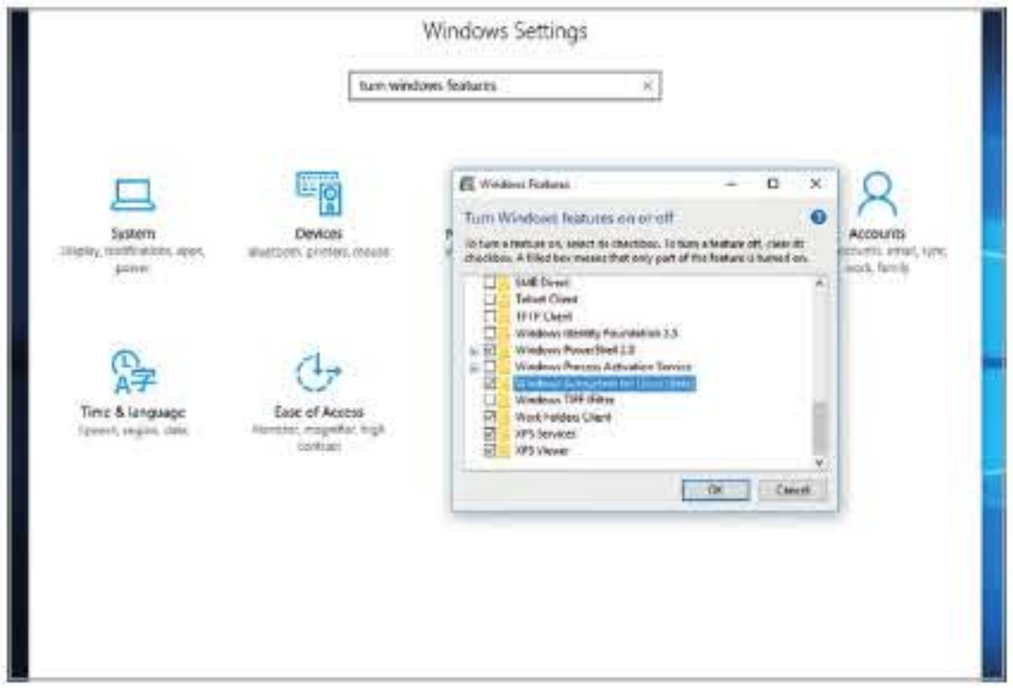
STEP 2 In macOS you can access Terminal through the Terminal app located inside Utilities folder, inside Applications. It's easier to use Spotlight (command-Space) and search for Terminal. Right-click the app icon in the Dock and choose Options > Keep In Dock.



STEP 3 Bash on Ubuntu on Windows is a relatively new addition to Windows 10; it was included in the Window 10 Anniversary Update. It's also a bit more complex to get to. First you need to update to the latest version of Windows 10. Then open Settings > Update & Security > For Developers. Now choose Developer Mode.

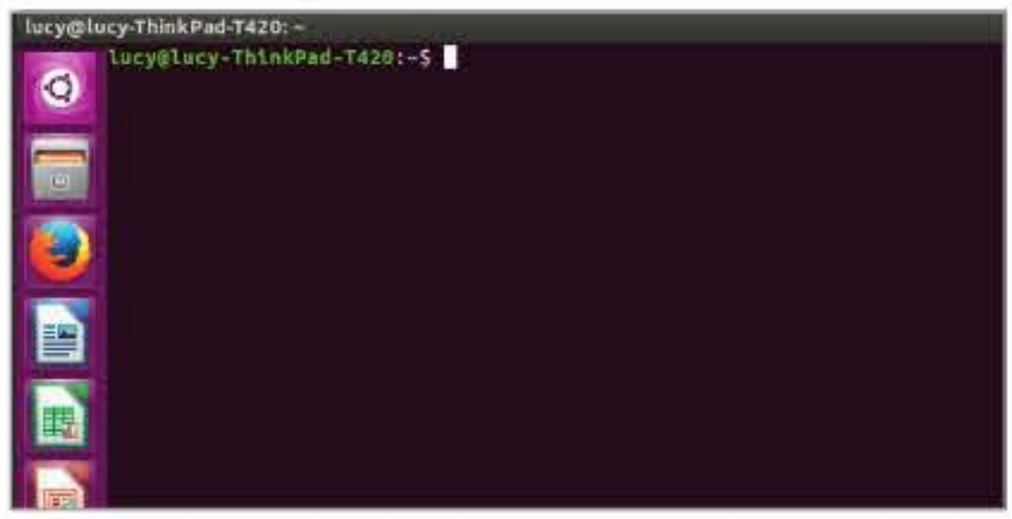


STEP 4 Now click in the search window and enter "turn windows features" and select Turn Windows Features On or Off. Scroll down the list to find Windows Subsystem for Linux. Set it to On and click OK. You will need to restart your computer.



STEP 5 Now click Start and look for Bash on Ubuntu on Windows under Most Used. If you can't find it, then use Cortana to look for Bash. The first time it runs you will be asked to accept the terms of service. From then on it will open in the root folder (empty by default).

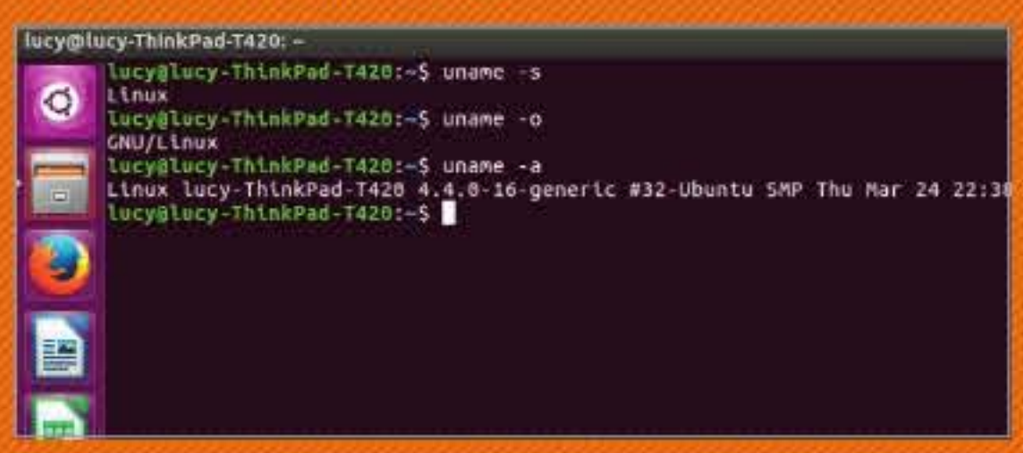
STEP 6 The Bash command line accepts Unix commands but it's not always Unix. On Linux and Windows it's a Unix-like system called "GNU"; however everybody says Unix because on the surface it's by-and-large indistinguishable from Unix. So the words Terminal, Unix, Bash and Command Line are often used interchangeably. Mostly it means "a text command you enter to control your computer".



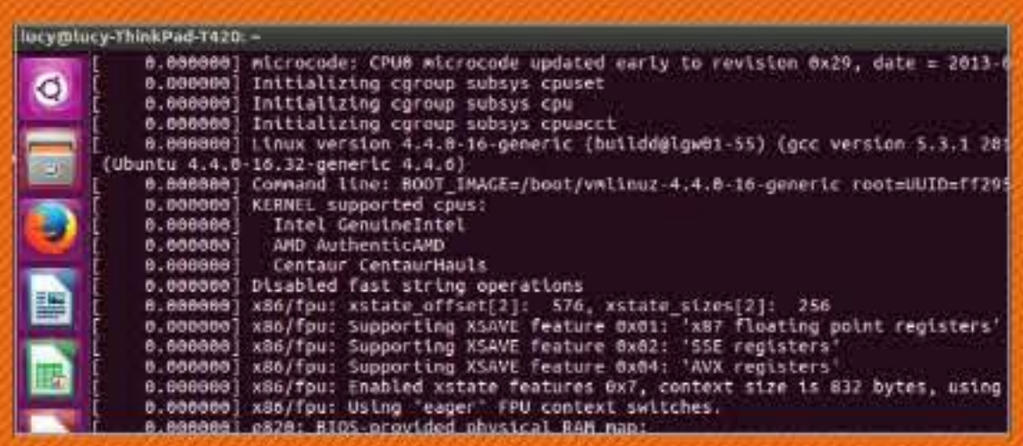
GETTING SYSTEM INFORMATION

Unix now works on just about any system. Apple's macOS actually is a Unix system with a flashy interface. Windows is the only major OS not based on Unix but it has a Unix interface so you can use Unix in Windows.

STEP 1 One of the neatest things you can do to get started is to enter some commands in Unix to get some information about your computer. Start by entering `uname -s`. This gives you the kernel. Now enter `uname -o` to see the operating system, such as "GNU/Linux". Finally enter `uname -a` to see all of the system information.

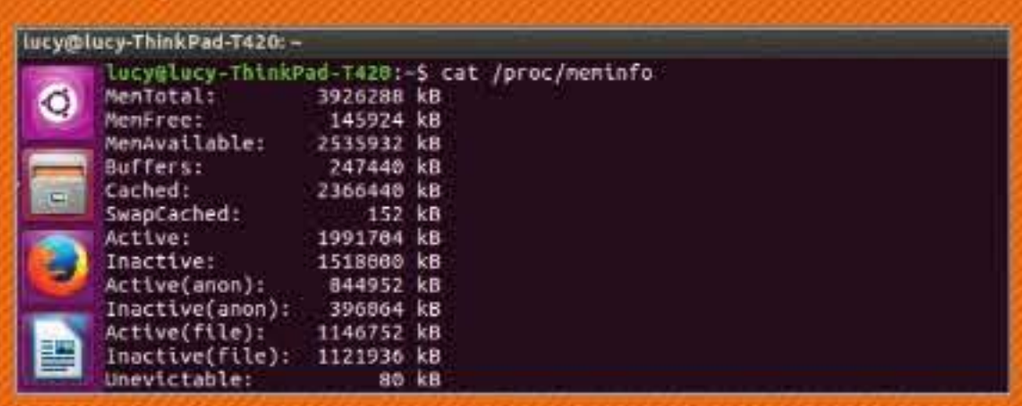


STEP 2 When a Linux computer system boots a lot of text heads up the screen. These are messages from the kernel and you can see them using `dmesg`. There's usually too much text to fit on the screen, so use `dmesg | less` to view the text one screen at a time. Press any key to move to the next screen of text and press `q` to exit the text.



STEP 3 Get information on your computer's hardware using the command `cat /proc/cpuinfo`. This displays the processor, model name and CPU architecture information. You'll see a separate section for each of the processor cores on your computer.

STEP 4 Enter `cat /proc/meminfo` to view detailed information about the memory installed on your Linux computer. This tells you the total and free amount of memory. You can also use `vmstat` to display a compact summary of system activity. Entering `top` runs a loop on system activity, similar to using Task Manager. Press Control-C to exit the top display.



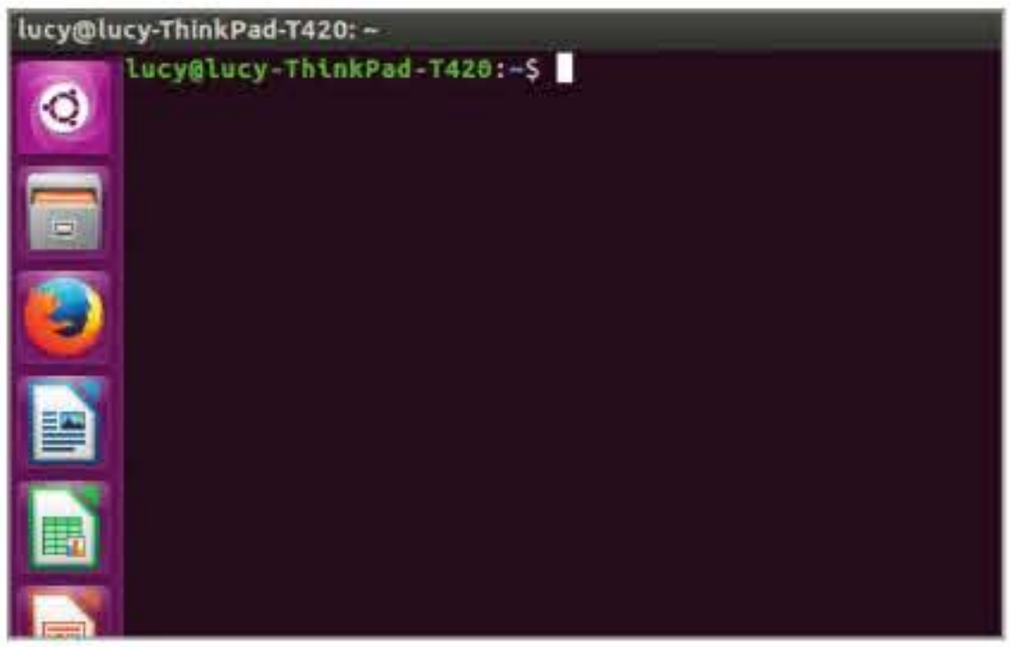
Using the Unix Filesystem

Navigating files and directories in Unix from the command line is a vital skill. It will enable you to figure out the structure of your computer and see where all the documents and directories are placed.

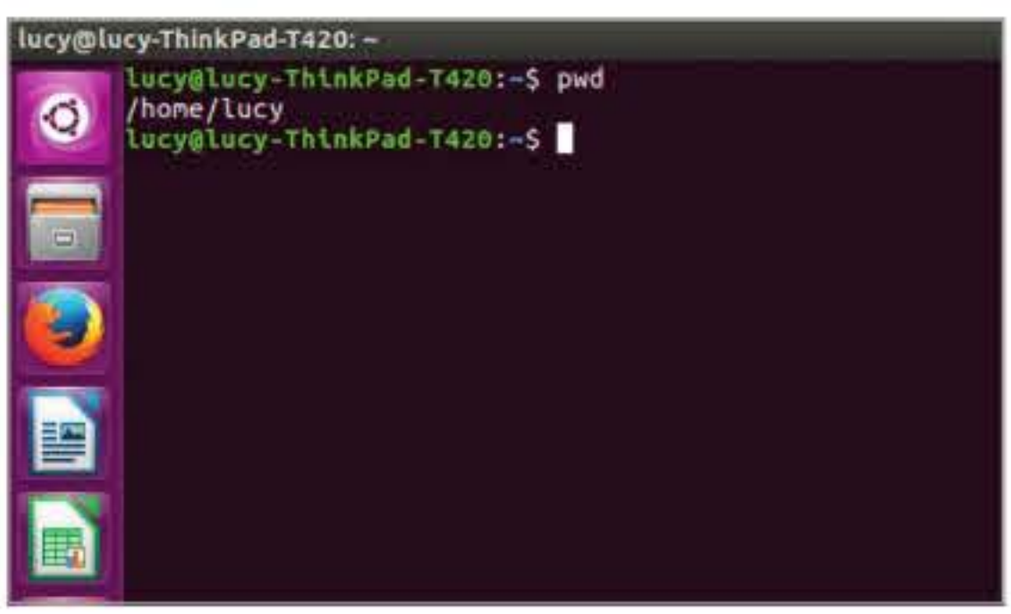
GETTING AROUND

For people raised on desktop computers, the idea of viewing and moving files and directories in a text environment can be daunting. Don't worry, it's easy to view and move files and directories and navigate the filesystem. Here we'll show you how.

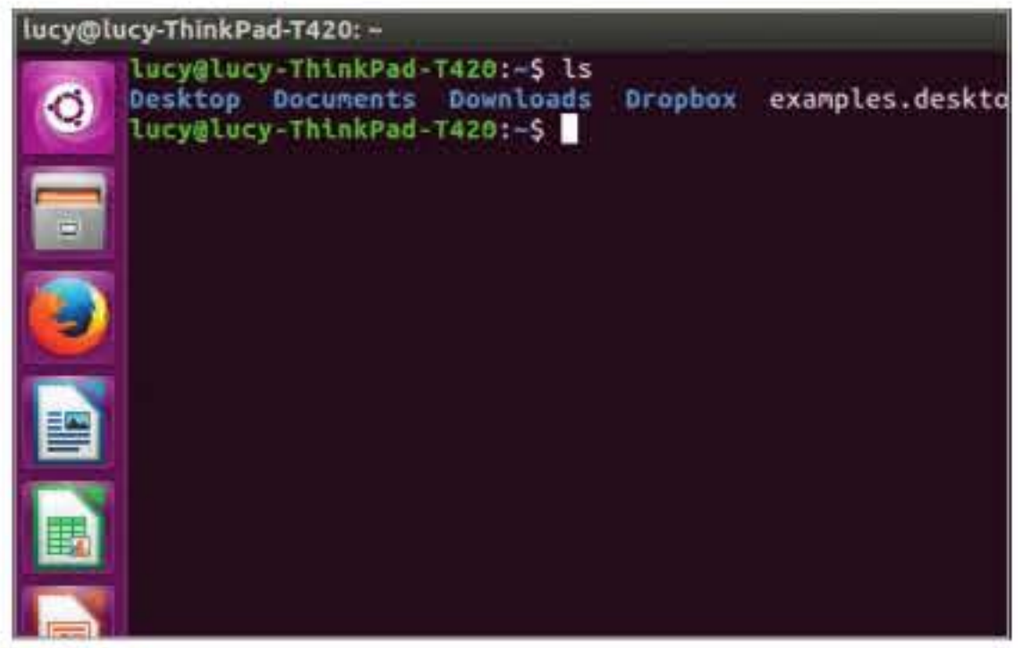
STEP 1 First you're going to look at directories and the directory path. A directory is the same thing as a folder, only in Unix it's always called a directory. Directories are placed inside each other using a '/' character. Enter `pwd`. This stands for Print Working Directory and displays "/home/[user]"; where "user" is your account name".



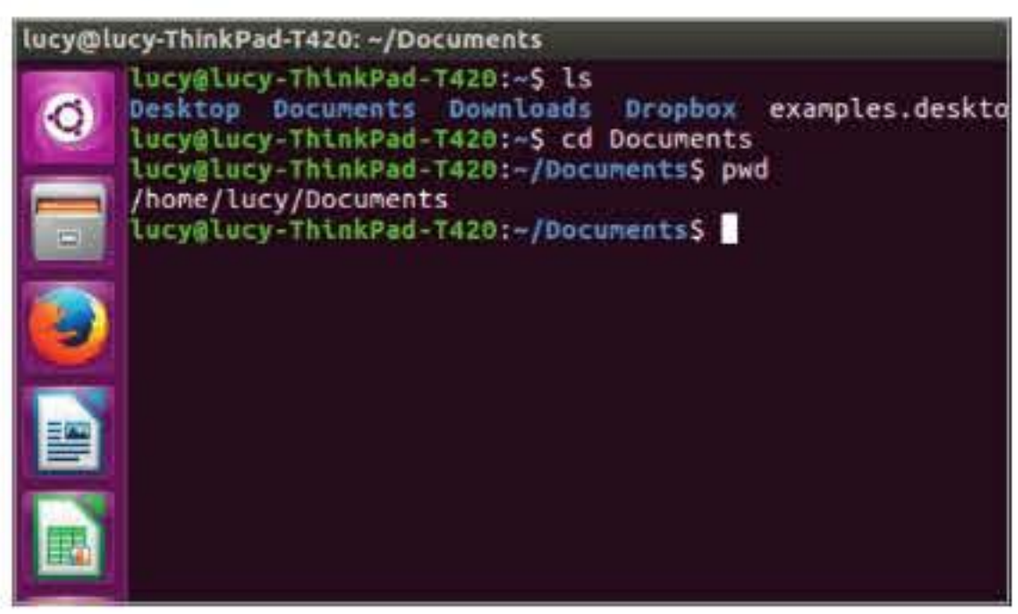
STEP 2 When you log in to your computer you don't start at the base of the hard drive, known as the 'root' (also known as the "top-most directory"). Instead you begin inside your user directory. Directories are indicated by the '/' symbol. So '/home/[user]' tells you that in the root is a directory called "home" and the next '/' says that inside 'home' is a directory with your name. That's where you start.



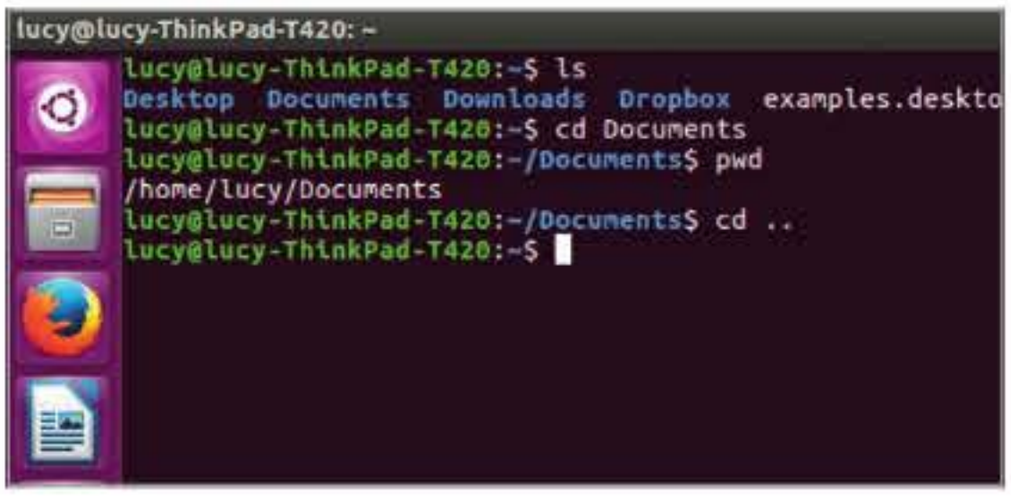
STEP 3 Enter `ls` to view the contents of the current directory. You should see Desktop, Documents, Downloads and other directories in blue. You may also see other items depending on how much you have used your computer. The colour code is worth knowing. Files are white, executable files (programs) are green and archived files are red.



STEP 4 Now we're going to move from the user directory into the Documents directory. Enter `cd Documents`. Note the capital "D". Unix is case sensitive, which means you have to enter the exact name including correct capitalisation. The cd command stands for "change directory". Now enter `pwd` again to view the directory path. It will be /home/[user]/Documents. Enter `ls` to view the files inside the Documents directory.



STEP 5 So how do we get back up to the Unix directory? We use a command `cd ..` – two dots in Unix means the directory above, also known as the "parent" directory. Incidentally a single dot is used for the same directory. You'll never use `cd .` to switch to the same directory but it's worth knowing because some commands need you to specify the current directory.

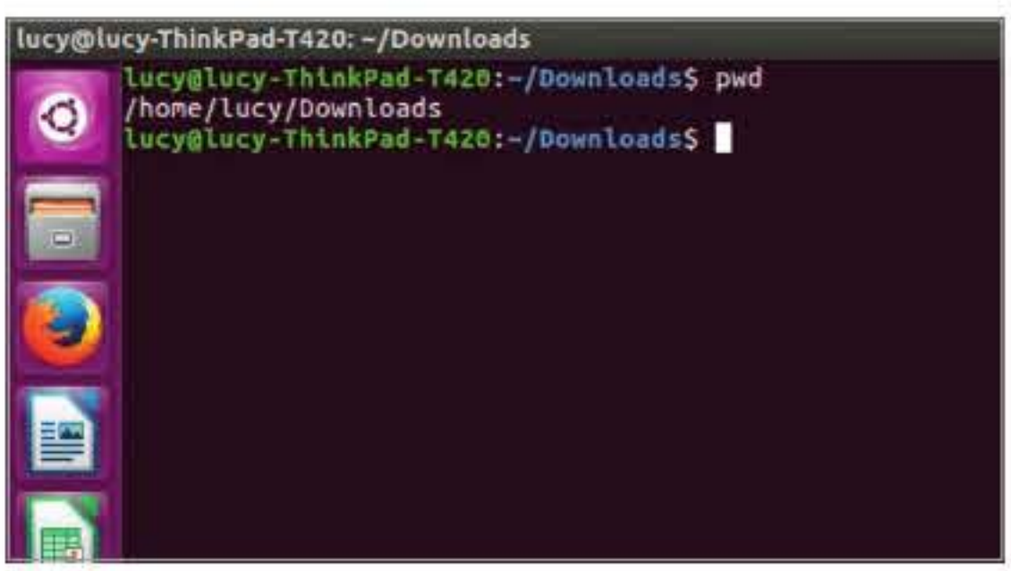


STEP 6 It's time to learn the single most important command line tip. You press the Tab key to expand what you are writing and let Unix fill out the rest. We're going to switch to the Downloads folder but this time enter `cd Dow` and press Tab. The command is expanded to "cd Downloads/". You can use Tab whenever you have to write out a long filename.

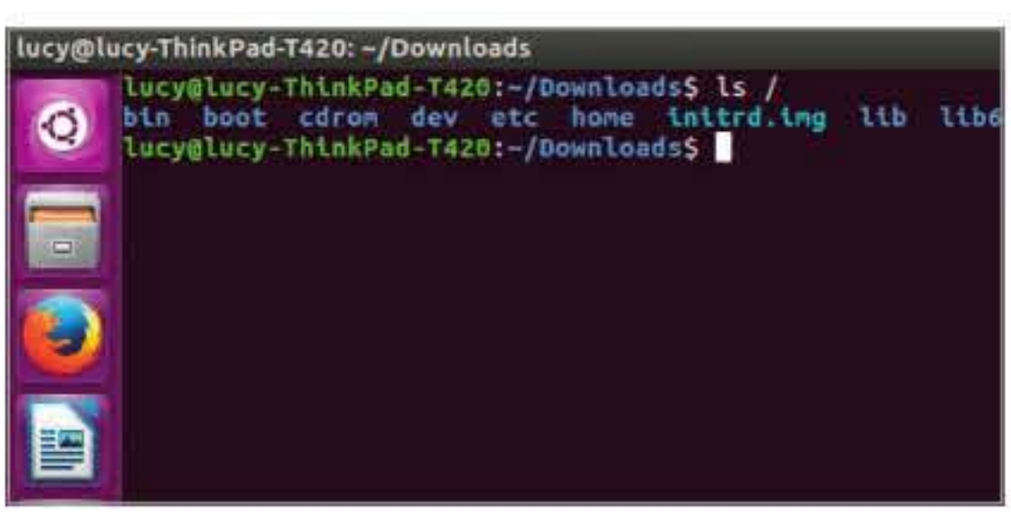
ABSOLUTE VS RELATIVE PATHS

It is important to know the difference between the working directory, root directory and home. There are also two types of path, Absolute and Relative. These are easier to understand than they sound. Let's take a look.

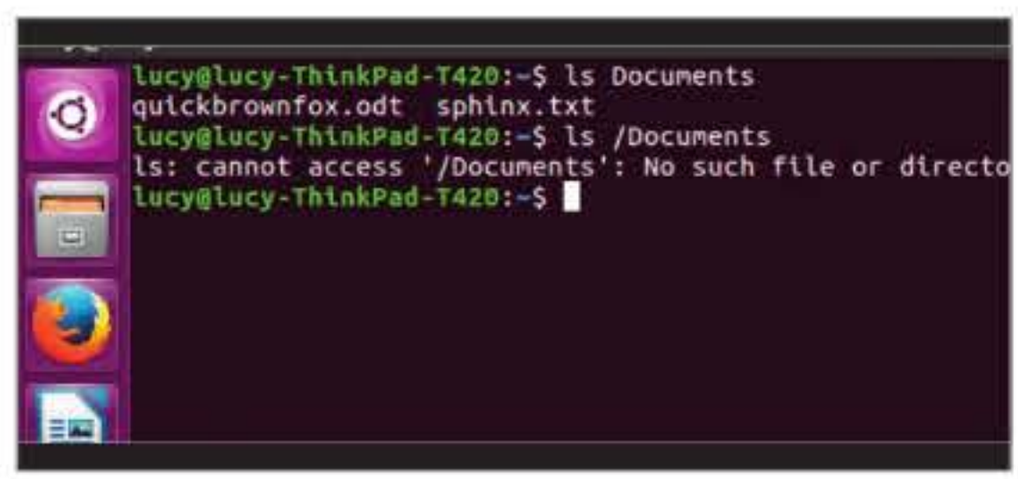
STEP 1 By default commands like `ls` use the working directory. That is the current directory that you're looking at and is set to your home directory by default (/users/[name]). Using `pwd` (Print Working Directory) lets you know what the working directory is and using cd changes the working directory.



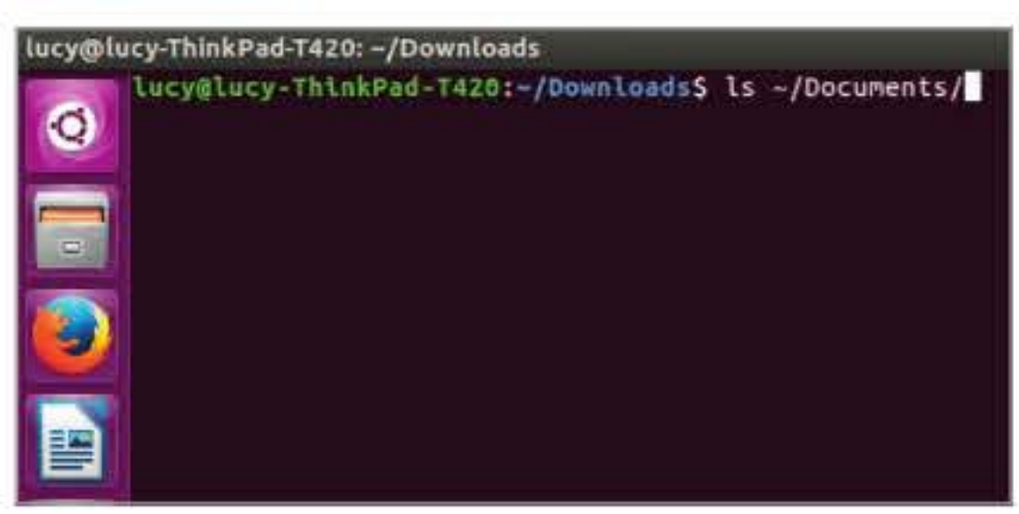
STEP 2 The root directory is always '/'. Entering `ls /` lists the contents of root and entering `cd /` switches to the root directory. This is important because there is a difference between `ls Documents` and `ls /Documents`. The first command lists the contents of the Documents inside the working directory that, if you are in home, will work.



STEP 3 The second command, `ls /Documents`, attempts to list the contents of Documents inside the root directory because the path started with '/', which is root. There is typically no Documents directory in root, so you will get a "No such file or directory" error. Starting a path with '/' is known as an "absolute path" while starting without the '/' is known as a "relative path" because it is relative to your working directory.



STEP 4 There is also an absolute path shortcut to your user directory and that is the tilde "~" character. Entering `ls ~` always lists the contents of your home directory while `cd ~` moves straight to you home directory no matter what your working directory is. You can also use this shortcut wherever you are; enter `ls ~/Documents` to display the contents of the Documents folder.



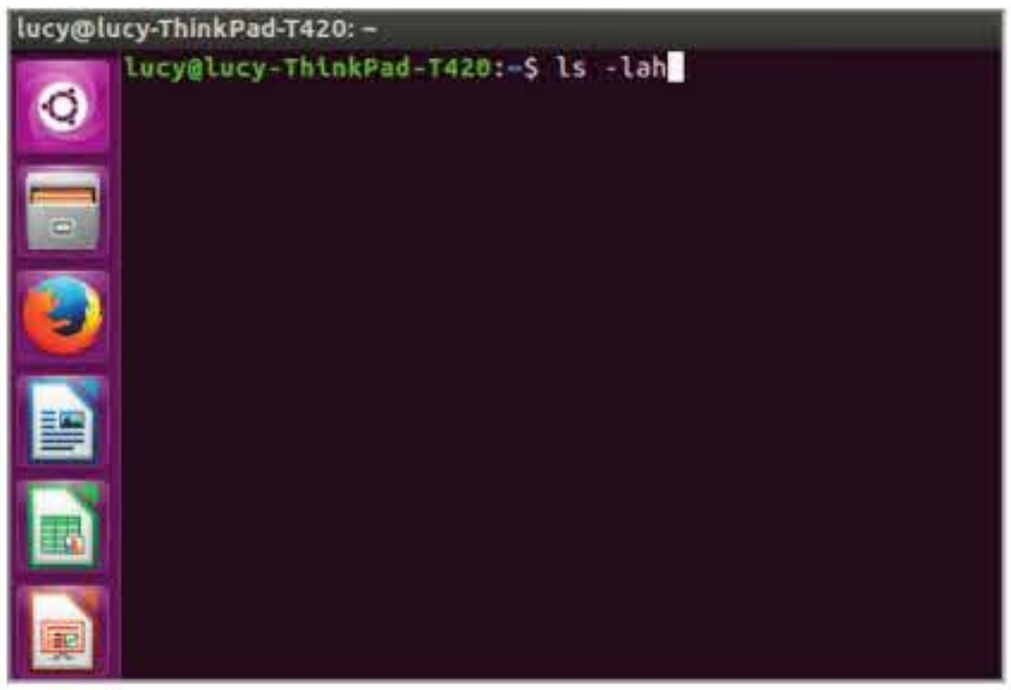
Listing and Viewing Files

Unlike a graphical interface, you need to ask Unix to list and display the files and directories in the command line interface. List is one the Unix commands that you will use the most, so it's important to truly understand how to use List properly.

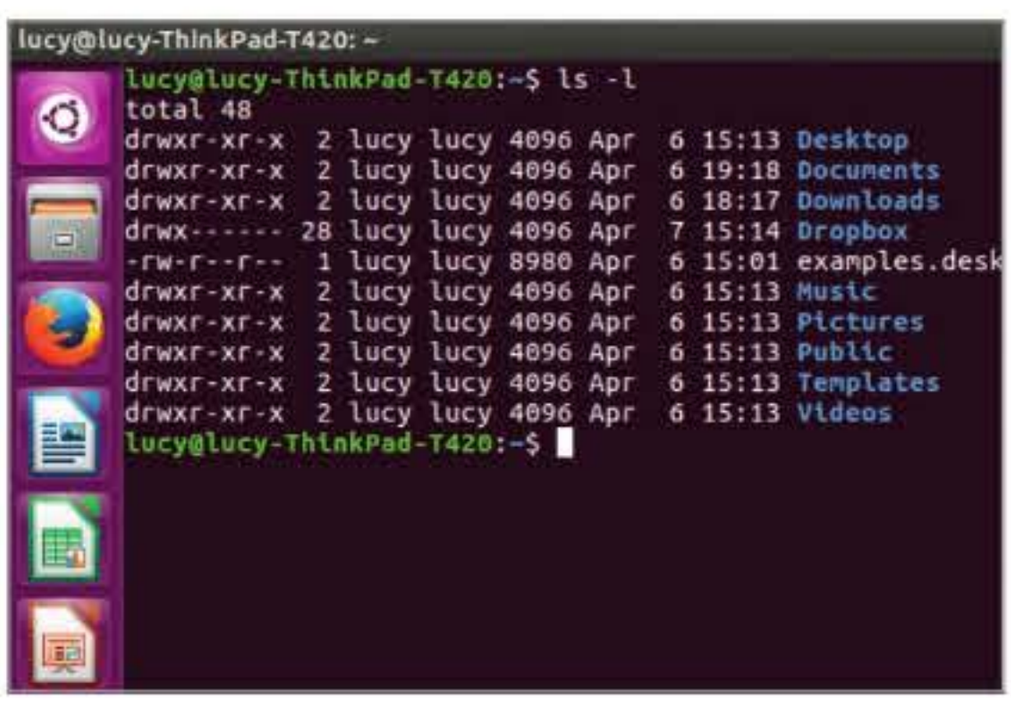
LOOKING AT FILES

Computer operating systems are made up of files and directories. Whilst you're aware of your own files, modern OS's keep other files out of sight. However, in the command line, you get access to every file on your system.

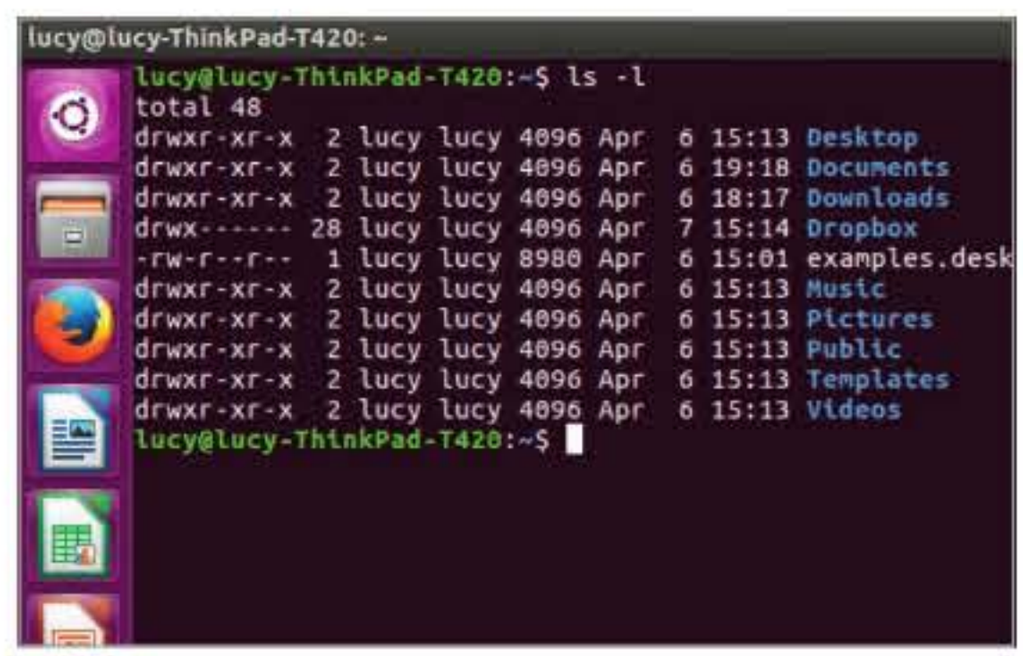
STEP 1 You've already looked at `ls`, which lists the files in the working directory but you are more likely to use a command like `ls -l`. The bit after the command (the '-lah') is known as the argument. This is an option that modifies the behaviour of the command.



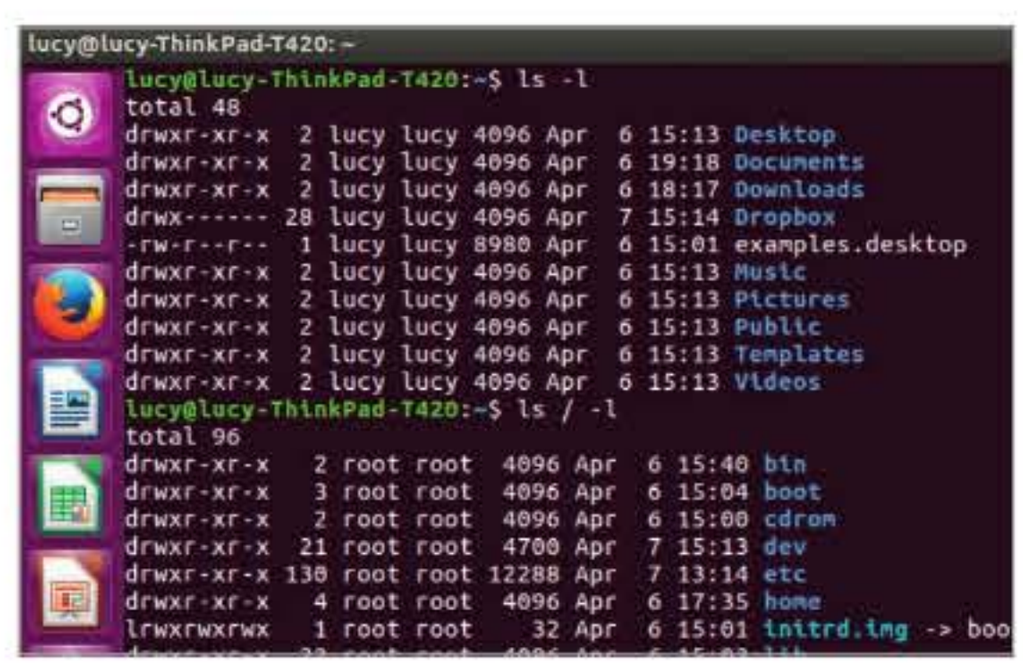
STEP 2 The -l argument lists files and directories in long format. Each file and directory is now on a single line and before each file is a lot of text. First you'll see lots of letters and dashes, like 'drwxr-xr-x'. Don't worry about these for now, they are known as 'permissions' and you'll come to those later.



STEP 3 After the permission letters come a single number. This is the number of files in the item. If it's a file then it will be 1 but if it's a directory it will be at least 2. This is because each directory contains two hidden files, one with a single dot (.) and one with two dots (..). Directories containing files or other directories will have a higher number.



STEP 4 Next you'll see your user name listed twice on each line. This refers to the user rather than the name of your computer. The first is the owner of the file and the second is the group. Typically these will be the same and you'll see either your username or 'root'. Enter `ls -l /` to view the files and directories in the root directory that belong to the root account.



STEP 5 The next number relates to the size of the file, in bytes. In Unix each text file is made up of letters and each letter takes up a byte. So our "sphinx.txt" file has 37 bytes and has 37 characters in the document. Files and directories can be extremely large and hard to determine so use `ls -lh`. The h argument 'humanizes' the number making it easier to read.

```
lucy@lucy-ThinkPad-T420: ~/Documents
lucy@lucy-ThinkPad-T420:~/Documents$ ls -l
total 16
-rw-rw-r-- 1 lucy lucy 8626 Apr  6 19:13 quickbrownfox.odt
-rw-rw-r-- 1 lucy lucy   37 Apr  6 19:18 sphinx.txt
lucy@lucy-ThinkPad-T420:~/Documents$
```

STEP 6 Finally you should be aware that there are many hidden files in Unix. These are listed using the -a argument. Hidden files and directories begin with a dot (.), so you should never start a file or directory with a dot, unless you want to hide it. Typically you'll combine all three arguments together into the command `ls -lah`.

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~$ ls -lah
total 140K
drwxr-xr-x 22 lucy lucy 4.0K Apr  7 15:36 .
drwxr-xr-x  4 root root 4.0K Apr  6 17:35 ..
-rw-------  1 lucy lucy 1.3K Apr  7 15:09 .bash_history
-rw-r--r--  1 lucy lucy  220 Apr  6 15:01 .bash_logout
-rw-r--r--  1 lucy lucy 3.7K Apr  6 15:01 .bashrc
drwx------ 23 lucy lucy 4.0K Apr  7 14:55 .cache
drwxrwxr-x  6 lucy lucy 4.0K Apr  6 17:08 .clamtk
drwx------ 20 lucy lucy 4.0K Apr  7 15:07 .config
drwxr-xr-x  2 lucy lucy 4.0K Apr  6 15:13 Desktop
-rw-r--r--  1 lucy lucy   25 Apr  6 15:13 .dmrc
drwxr-xr-x  2 lucy lucy 4.0K Apr  6 19:18 Documents
drwxr-xr-x  2 lucy lucy 4.0K Apr  6 18:17 Downloads
drwx------  5 lucy lucy 4.0K Apr  7 15:14 .dropbox
drwx------ 28 lucy lucy 4.0K Apr  7 15:14 Dropbox
drwxr-xr-x  3 lucy lucy 4.0K Apr  6 15:43 .dropbox-dist
```

SOME COMMON DIRECTORIES

Now that you know how to view the contents of your hard drive you'll start to notice a lot of directories with names like bin, sbin, var and dev. These are the files and directories that you are kept away from on a Mac and won't encounter on a Windows PC.

STEP 1 Enter `ls -lah /` to view all of the files and directories, including the hidden items, in the root directory of your hard drive. Here you will see all the items that make up Unix. It's worth taking the time to know some of them.

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~$ ls -lah /
total 104K
drwxr-xr-x  23 root root 4.0K Apr  6 15:01 .
drwxr-xr-x  23 root root 4.0K Apr  6 15:01 ..
drwxr-xr-x   2 root root 4.0K Apr  6 15:40 bin
drwxr-xr-x   3 root root 4.0K Apr  6 15:04 boot
drwxr-xr-x   2 root root 4.0K Apr  6 15:00 cdrom
drwxr-xr-x  21 root root 4.6K Apr  7 15:13 dev
drwxr-xr-x 130 root root  12K Apr  7 13:14 etc
drwxr-xr-x   4 root root 4.0K Apr  6 17:35 home
lrwxrwxrwx   1 root root   32 Apr  6 15:01 initrd.img -
drwxr-xr-x  22 root root 4.0K Apr  6 15:03 lib
drwxr-xr-x   2 root root 4.0K Apr  5 08:33 lib64
drwx------   2 root root  16K Apr  6 14:57 lost+found
drwxr-xr-x   3 root root 4.0K Apr  6 15:15 media
```

STEP 3 Entering `ls /home` displays the contents of your home directory, which contains your user directory. So entering ls/home/[name] (`ls /home/lucy` in our case) is the same as just ls from the home directory. This is where you are expected to place most of the documents you create. Don't confuse home with usr. The /usr directory is where find you find program tools and libraries.

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~$ ls /home
lucy
lucy@lucy-ThinkPad-T420:~$ ls /home/lucy
Desktop  Documents  Downloads  Dropbox  examples.desktop  Music  Pi
lucy@lucy-ThinkPad-T420:~$
```

STEP 2 Bin is a directory that stores binaries. This is the Unix way of saying programs or applications. Sbin is for system binaries, which are the programs that make up your system. Dev contains references to your devices, so hard drive, keyboard, mouse and so on. Etc contains your system configuration files.

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~$ ls /bin
bash          chacl         dumpkeys      gzip          login
bunzip2       chgrp         echo          hciconfig     loginctl
busybox       chmod         ed            hostname      lowntfs-3
bzcat         chown         efibootmgr    ip            ls
bzcmp         chvt          egrep         journalctl    lsblk
bzdiff        cp            false         kbd_mode      lsmod
bzegrep       cpio          fgconsole     kill          mkdir
bzexe         dash          fgrep         kmod          mknod
bzfgrep       date          findmnt       less          mktemp
bzgrep        dd            fuser         lessecho      more
bzip2         df            fusermount    lessfile      mount
bzip2recover  dir           getfacl       lesskey       mountpoin
```

STEP 4 Lib is a directory that contains libraries of code that are referred to by other programs; different programs share files in Lib. Var is short for various, which is mostly files used by the system but you may need to work with items here. Finally there is a directory called tmp, which is for temporary files; files placed here are on your system for the short term and can be deleted from the system.

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~$ ls /lib
apparmor  cpp  firmware  ifupdown  klibc-k3La8MUnuzHQ0_kG8hokcGA
britty    crda  hdparm   init      linux-sound-base
lucy@lucy-ThinkPad-T420:~$ ls /var
backups  cache  crash  lib  local  lock  log  mail  service  opt
lucy@lucy-ThinkPad-T420:~$
```

Creating and Deleting Files

Discover how to make and remove files in Unix. Using Touch to create files is a handy skill to learn but it's vital you learn to delete files carefully and correctly when using the command line interface. In this tutorial we'll show you how to make and remove files.

CREATING FILES

Once you learn to recognise the files and directories that make up Unix, it's time to learn how to make your own. Knowing how to make, edit and delete files and directories is essential if you want to make your own projects.

STEP 1 You're going to create a file using a command called "Touch". Touch is an interesting tool that reaches out to a file or directory and touches it, which updates the modification time. You can see touch in access using `ls -l` and checking the time next to any file or directory.

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~$ ls -l
total 48
drwxr-xr-x  2 lucy lucy 4096 Apr  6 15:13 Desktop
drwxr-xr-x  2 lucy lucy 4096 Apr  7 16:17 Documents
drwxr-xr-x  2 lucy lucy 4096 Apr  6 18:17 Downloads
drwx------ 28 lucy lucy 4096 Apr  7 15:14 Dropbox
-rw-r--r--  1 lucy lucy 8980 Apr  6 15:01 examples.desktop
drwxr-xr-x  2 lucy lucy 4096 Apr  7 16:19 Music
drwxr-xr-x  2 lucy lucy 4096 Apr  6 15:13 Pictures
drwxr-xr-x  2 lucy lucy 4096 Apr  6 15:13 Public
drwxr-xr-x  2 lucy lucy 4096 Apr  6 15:13 Templates
drwxr-xr-x  2 lucy lucy 4096 Apr  6 15:13 Videos
lucy@lucy-ThinkPad-T420:~$
```

STEP 2 Now enter `touch Videos` or the name of any file or directory and `ls -l` again and notice that the time has changed. It now matches the current time. You might be wondering what this has to do with creating files. Touch has a second, more popular, use and that is to create files.

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~$ ls -l
total 48
drwxr-xr-x  2 lucy lucy 4096 Apr  6 15:13 Desktop
drwxr-xr-x  2 lucy lucy 4096 Apr  7 16:17 Documents
drwxr-xr-x  2 lucy lucy 4096 Apr  6 18:17 Downloads
drwx------ 28 lucy lucy 4096 Apr  7 15:14 Dropbox
-rw-r--r--  1 lucy lucy 8980 Apr  6 15:01 examples.desktop
drwxr-xr-x  2 lucy lucy 4096 Apr  7 16:19 Music
drwxr-xr-x  2 lucy lucy 4096 Apr  6 15:13 Pictures
drwxr-xr-x  2 lucy lucy 4096 Apr  6 15:13 Public
drwxr-xr-x  2 lucy lucy 4096 Apr  6 15:13 Templates
drwxr-xr-x  2 lucy lucy 4096 Apr  6 15:13 Videos
lucy@lucy-ThinkPad-T420:~$ touch Videos
lucy@lucy-ThinkPad-T420:~$ ls -l
total 48
drwxr-xr-x  2 lucy lucy 4096 Apr  6 15:13 Desktop
drwxr-xr-x  2 lucy lucy 4096 Apr  7 16:17 Documents
drwxr-xr-x  2 lucy lucy 4096 Apr  6 18:17 Downloads
drwx------ 28 lucy lucy 4096 Apr  7 15:14 Dropbox
-rw-r--r--  1 lucy lucy 8980 Apr  6 15:01 examples.desktop
drwxr-xr-x  2 lucy lucy 4096 Apr  7 16:19 Music
drwxr-xr-x  2 lucy lucy 4096 Apr  6 15:13 Pictures
drwxr-xr-x  2 lucy lucy 4096 Apr  6 15:13 Public
```

STEP 3 If you try to touch a file that doesn't exist you create a blank file with that name. Enter cd Documents and enter `touch testfile` and `ls -l` to view the file. You'll now have a new file in Document called "testfile". Notice that the size of the file is 0 because it has nothing in it.

```
lucy@lucy-ThinkPad-T420: ~/Documents
lucy@lucy-ThinkPad-T420:~/Documents$ touch testfile
lucy@lucy-ThinkPad-T420:~/Documents$ ls -l
total 0
-rw-rw-r-- 1 lucy lucy 0 Apr  7 16:20 testfile
lucy@lucy-ThinkPad-T420:~/Documents$
```

STEP 4 A quick word about file names. Remember that Unix is case sensitive, so if you enter `touch Testfile` with a capital T you won't update "testfile"; instead you'll create a second file called Testfile. Enter `ls -l` to see both files. This is confusing, so most people stick with using lowercase letters at all times.

```
lucy@lucy-ThinkPad-T420: ~/Documents
lucy@lucy-ThinkPad-T420:~/Documents$ touch Testfile
lucy@lucy-ThinkPad-T420:~/Documents$ ls -l
total 0
-rw-rw-r-- 1 lucy lucy 0 Apr  7 16:20 testfile
-rw-rw-r-- 1 lucy lucy 0 Apr  7 16:21 Testfile
lucy@lucy-ThinkPad-T420:~/Documents$
```

STEP 5 Another important thing to know is never to use a space in your file names. If you try to enter `touch test file` you create a document called test and a document called file. There are ways to read and write files containing spaces though; you use a backslash followed by space "\ ". However don't ever use an underscore character ("_") instead.

```
lucy@lucy-ThinkPad-T420: ~/Documents
lucy@lucy-ThinkPad-T420:~/Documents$ touch test file
lucy@lucy-ThinkPad-T420:~/Documents$ ls -l
total 0
-rw-rw-r-- 1 lucy lucy 0 Apr  7 16:21 file
-rw-rw-r-- 1 lucy lucy 0 Apr  7 16:21 test
-rw-rw-r-- 1 lucy lucy 0 Apr  7 16:20 testfile
-rw-rw-r-- 1 lucy lucy 0 Apr  7 16:21 Testfile
lucy@lucy-ThinkPad-T420:~/Documents$
```

STEP 6 Here are some other file names to avoid: `#%&{}\<>*?/$!'":@+`|=`. The full-stop (.) is used to create an extension to a file, and is usually used to indicate a file type, such as textfile.txt or compressedfile.zip. Starting a file with a full-stop makes it invisible. Don't use full-stops in place of spaces, stick to underscores and don't use spaces.

```
lucy@lucy-ThinkPad-T420: ~/Documents
lucy@lucy-ThinkPad-T420:~/Documents$ touch #%&{}\<>*?/$
```

REMOVING FILES

You've created some files that you don't want, so how do you go about removing them? It turns out that deleting files on your Unix is super easy. Which is a problem, so be careful.

STEP 1 Enter `ls -l` to view the files in your home directory. If you've followed the steps before then you should have four files: file, test, testfile and Testfile. You're going to get rid of these items because they were created as an example.

```
lucy@lucy-ThinkPad-T420: ~/Documents
lucy@lucy-ThinkPad-T420:~/Documents$ ls -l
total 0
-rw-rw-r-- 1 lucy lucy 0 Apr  7 16:21 file
-rw-rw-r-- 1 lucy lucy 0 Apr  7 16:21 test
-rw-rw-r-- 1 lucy lucy 0 Apr  7 16:20 testfile
-rw-rw-r-- 1 lucy lucy 0 Apr  7 16:21 Testfile
lucy@lucy-ThinkPad-T420:~/Documents$
```

STEP 2 To get rid of files you use the rm command. Enter `rm Testfile` to delete the file called Testfile (with the uppercase "t"). Enter `ls -l` and you'll find it's gone but where is it? It's not in the Trash or Recycle Bin, like on a Mac or Windows PC. It's deleted completely and cannot be recovered. Bear this in mind and always think before deleting files.

```
lucy@lucy-ThinkPad-T420: ~/Documents
lucy@lucy-ThinkPad-T420:~/Documents$ ls -l
total 0
-rw-rw-r-- 1 lucy lucy 0 Apr  7 16:21 file
-rw-rw-r-- 1 lucy lucy 0 Apr  7 16:21 test
-rw-rw-r-- 1 lucy lucy 0 Apr  7 16:20 testfile
-rw-rw-r-- 1 lucy lucy 0 Apr  7 16:21 Testfile
lucy@lucy-ThinkPad-T420:~/Documents$ rm Testfile
lucy@lucy-ThinkPad-T420:~/Documents$ ls -l
total 0
-rw-rw-r-- 1 lucy lucy 0 Apr  7 16:21 file
-rw-rw-r-- 1 lucy lucy 0 Apr  7 16:21 test
-rw-rw-r-- 1 lucy lucy 0 Apr  7 16:20 testfile
lucy@lucy-ThinkPad-T420:~/Documents$
```

STEP 3 You're going to use a wildcard (*) to delete our next two files but again this is something you really need to do with care. So you're going to use the "i" argument with it to request confirmation. Enter `ls test*` to view files that match the word test and any other characters. The * character is called a "wildcard" and it means any characters here.

```
lucy@lucy-ThinkPad-T420: ~/Documents
lucy@lucy-ThinkPad-T420:~/Documents$ ls test*
test  testfile
lucy@lucy-ThinkPad-T420:~/Documents$
```

STEP 4 You see that ls test* matches two files: "test" and "testfile"; but not the file called "file". That's because it didn't match the "test" part of "test*". Enter `rm -i test*` to remove both files. It will say "rm: remove regular empty file 'test'?" Enter y and press Return twice. Finally enter `rm file` to get rid of the confusing "file" file.

```
lucy@lucy-ThinkPad-T420: ~/Documents
lucy@lucy-ThinkPad-T420:~/Documents$ ls test*
test  testfile
lucy@lucy-ThinkPad-T420:~/Documents$ rm -i test*
rm: remove regular empty file 'test'? y
rm: remove regular empty file 'testfile'? y
lucy@lucy-ThinkPad-T420:~/Documents$ ls
file
lucy@lucy-ThinkPad-T420:~/Documents$
```

Create and Remove Directories

Directories, also known as "folders", form the structure of your filesystem. They contain all the files used to create the Unix operating system, as well as files you create yourself. Learning how to make and remove directories is an important Unix skill to learn.

MANAGING FILES AND DIRECTORIES

Now that you know how to create files, you'll want to learn how to make directories as well as move items around. Directories are the same thing as folders, although in Unix they are always called "directories".

STEP 1 You're working from the home directory, so enter `cd ~` if you're not already in it. Enter `ls` to quickly view all the directories currently in the home location. Directories are created using the mkdir command (make directory). Enter `mkdir testdir` to create a new directory in your home directory. Enter `ls` again to see it.

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~$ cd ~
lucy@lucy-ThinkPad-T420:~$ ls
Desktop  Documents  Downloads  Dropbox  examples.deskto
lucy@lucy-ThinkPad-T420:~$ mkdir testdir
lucy@lucy-ThinkPad-T420:~$ ls
Desktop  Documents  Downloads  Dropbox  examples.deskto
lucy@lucy-ThinkPad-T420:~$
```

STEP 2 The mkdir command is different to touch in that it doesn't update the timestamp, if you use it with a directory that already exists. Enter mkdir testdir again and you'll get the error "mkdir: cannot create directory 'testdir: File exists".

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~$ mkdir testdir
mkdir: cannot create directory 'testdir': File exists
lucy@lucy-ThinkPad-T420:~$
```

STEP 3 Like touch, you can create multiple directories at once with the mkdir command. Enter `mkdir testdir2 testdir3` and enter `ls`. You'll now find several directories called testdir. Like files, you should know this means you can't, and really shouldn't, even be trying to create directories with spaces.

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~$ mkdir testdir2 testdir3
lucy@lucy-ThinkPad-T420:~$ ls
Desktop  Documents  Downloads  Dropbox  examples.deskto
lucy@lucy-ThinkPad-T420:~$
```

STEP 4 You create directories inside each other using the directory path. Enter `mkdir Documents/photos` to create a new directory called photos inside your documents directory. The directory has to already exist though so try to enter `mkdir articles/reports` and you'll get an error because there is no articles directory.

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~$ mkdir Documents/photos
lucy@lucy-ThinkPad-T420:~$ mkdir articles/reports
mkdir: cannot create directory 'articles/reports': No s
lucy@lucy-ThinkPad-T420:~$
```

STEP 5 To create a directory path you need to pass in the p option to mkdir, which stands for "parents". Options if you remember, come after the command and start with a '-'. So enter `mkdir -p articles/reports`. Enter `ls` to view the articles directory and `ls articles` to view the reports directory sitting inside.

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~$ mkdir -p articles/reports
lucy@lucy-ThinkPad-T420:~$ ls
articles  Desktop  Documents  Downloads  Dropbox  examp
lucy@lucy-ThinkPad-T420:~$ ls articles
reports
lucy@lucy-ThinkPad-T420:~$
```

STEP 6 Now that you're starting to get a bit more advanced we're going to just reiterate something. In Unix the command structure is always: command, option and argument, in that order. The command is the function; next are the options, typically single letters starting with "-"; and finally the argument, often a file or directory structure. It's always command, option, argument. Drill it into your head!

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~$ ls -l Documents
```

GETTING RID OF DIRECTORIES

Deleting directories is pretty easy in Unix and like files this is a problem. It's too easy to delete entire directories containing files and these are instantly removed and not sent to a trash directory. Tread carefully.

STEP 1 You're going to remove one of the directories created earlier using the rmdir command. Enter `ls` to view the files and directories in the current directory. You'll start by getting rid of one of the test directories. Enter `rmdir testdir3` and `ls` again to confirm the directory has been removed.

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~$ ls
articles  Desktop  Documents  Downloads  Dropbox  examp
lucy@lucy-ThinkPad-T420:~$ rmdir testdir3
lucy@lucy-ThinkPad-T420:~$ ls
articles  Desktop  Documents  Downloads  Dropbox  examp
lucy@lucy-ThinkPad-T420:~$
```

STEP 2 Now you'll try to get rid of the articles directory containing the reports directory. Enter `rmdir articles` and press return. You'll get an error saying "rmdir: failed to remove 'articles': Directory not empty". This is a puzzler, the rmdir command only removes directories that having nothing in them i.e. no files or other directories.

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~$ rmdir articles
rmdir: failed to remove 'articles': Directory not empty
lucy@lucy-ThinkPad-T420:~$
```

STEP 3 To delete a directory containing files or other directories you return to the rm command used to remove files. Only now you need to use the -R option, which stands for "recursive". Using rm -R removes all the files and directories to whatever you point it at. Enter `rm -R articles` to remove the articles directory.

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~$ rm -R articles
lucy@lucy-ThinkPad-T420:~$
```

STEP 4 As with multiple files, you can delete multiple directories inside the same directory using the rm command with the wildcard character "*". This should be done with care though so use the -i option, which stands for "interactive". This will prompt you before each deletion. Enter `rm -Ri test*` and press y and return to each prompt. It's a good idea to use the -i option whenever using the rm command.

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~$ ls
Desktop  Documents  Downloads  Dropbox  examples.deskto
lucy@lucy-ThinkPad-T420:~$ rm -Ri test*
rm: remove directory 'testdir'? y
rm: remove directory 'testdir2'? y
lucy@lucy-ThinkPad-T420:~$
```

Copying, Moving and Renaming Files

On your desktop you move files by clicking and dragging them but in a command line interface you must be more precise. Here's how to use the Move command to reposition files and directories, and rename items.

USING THE MOVE COMMAND

Knowing how to move files is an essential part of learning Unix. Once you know how to move files and directories around, you can start to organise your computer and put items where they are required.

STEP 1 Before you can move anything around you need to have a few test items in your home directory. Enter `touch testfile` and `mkdir testdir` to create a test file and test directory in your home directory. Enter `ls` to check both are present.

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~$ touch testfile
lucy@lucy-ThinkPad-T420:~$ mkdir testdir
lucy@lucy-ThinkPad-T420:~$ ls
Desktop  Documents  Downloads  Dropbox  examples.deskto
lucy@lucy-ThinkPad-T420:~$
```

STEP 2 Files and directories are moved using the "mv" command. This is different to commands we've looked at so far because it has two arguments; remember Linux command line is command, option, argument. The first argument is source, the file or directory to be moved, the second is the destination.

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~$ mv
```

STEP 3 Enter `mv testfile testdir` and press return to move the test file document into the testdir directory. Enter `ls` to see that it's no longer in the home directory and `ls testdir` to see the test file now sitting in the testdir directory. Now enter `mkdir newparent` to create a new directory.

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~$ mv testfile testdir
lucy@lucy-ThinkPad-T420:~$ ls testdir
testfile
lucy@lucy-ThinkPad-T420:~$
```

STEP 4 Directories with files are moved in the same way. Enter `mkdir testdir2` to create a second directory. Now enter `mv testdir testdir2` to move the testdir directory inside the testdir2 directory. Enter `ls testdir2/testdir` to view the test file sitting inside the directory. Enter `cd testdir2/testdir` to move into the directory.

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~$ mkdir testdir2
lucy@lucy-ThinkPad-T420:~$ ls
Desktop  Documents  Downloads  Dropbox  examples.deskto
lucy@lucy-ThinkPad-T420:~$ mv testdir testdir2
lucy@lucy-ThinkPad-T420:~$ ls testdir2
testdir
lucy@lucy-ThinkPad-T420:~$ ls testdir2/testdir
testfile
lucy@lucy-ThinkPad-T420:~$
```

STEP 5 Files and directories can be moved up using the double dot ("..") as an argument. Enter `ls -la` to view your test file and the single and double dot files. The single dot is the current directory and the double dot is the parent directory. Enter `mv testfile ..` to move the test file up into the new parent directory. Enter `cd ..` to move up to the parent directory.

```
lucy@lucy-ThinkPad-T420: ~/testdir2
lucy@lucy-ThinkPad-T420:~$ cd testdir2/testdir
lucy@lucy-ThinkPad-T420:~/testdir2/testdir$ ls -la
total 8
drwxrwxr-x 2 lucy lucy 4096 Apr  7 16:52 .
drwxrwxr-x 3 lucy lucy 4096 Apr  7 16:52 ..
-rw-rw-r-- 1 lucy lucy    0 Apr  7 16:47 testfile
lucy@lucy-ThinkPad-T420:~/testdir2/testdir$ mv testfile
lucy@lucy-ThinkPad-T420:~/testdir2/testdir$ cd ..
lucy@lucy-ThinkPad-T420:~/testdir2$ ls
testdir  testfile
lucy@lucy-ThinkPad-T420:~/testdir2$
```

STEP 6 You can also mv files using longer paths. Enter `cd ~` to return to the home directory and `mv testdir2/testfile testdir2/testdir/testfile` to move the testfile from its current location back inside the testdir directory. Enter `ls testdir2/testdir` to view the file back in its current directory.

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~/testdir2$ cd ~
lucy@lucy-ThinkPad-T420:~$ mv testdir2/testfile testdir
lucy@lucy-ThinkPad-T420:~$ ls testdir2/testdir
testfile
lucy@lucy-ThinkPad-T420:~$
```

RENAMING FILES AND DIRECTORIES

The mv command isn't used just to move files. It also serves the purpose of renaming files, effectively moving a file from its old name to a new name. Let's see how to use mv to rename items.

STEP 1 Let's start by making a new test file called names. Enter `touch testfile` and then `ls` to make sure the testfile is present. You're going to turn this into a file that contains the names of some people. So let's call it something more appropriate, like "names".

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~$ touch testfile
lucy@lucy-ThinkPad-T420:~$
```

STEP 2 Enter `mv testfile names` and `ls`. Now you can see the new names file in your directory. The mv command can also be used to rename directories. You should still have your testdir2 directory in your home directory. Enter `mv testdir2` people to rename the directory. Enter `ls` to view it.

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~$ touch testfile
lucy@lucy-ThinkPad-T420:~$ mv testfile names
lucy@lucy-ThinkPad-T420:~$ ls
Desktop  Documents  Downloads  Dropbox  examples.deskto
lucy@lucy-ThinkPad-T420:~$
```

STEP 3 You can rename directories inside other directories using paths. Let's rename the testdir directory, which is now inside the people directory. Enter `mv people/testdir people/friends` to rename the testdir directory inside the people directory.

```
lucy@lucy-ThinkPad-T420:~$ ls
Desktop  Documents  Downloads  Dropbox  examples.deskto
lucy@lucy-ThinkPad-T420:~$ mv people/testdir people/fri
lucy@lucy-ThinkPad-T420:~$ ls people
friends
lucy@lucy-ThinkPad-T420:~$
```

STEP 4 It is easy to overwrite files using the mv command so if you have files with the same name use the -n option, which stands for "no overwrite". Enter `touch testfile` to create a new file and `mv -n testfile` names. There's no error report though, so enter `ls` and you'll find both still around.

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~$ mv -n testfile names
lucy@lucy-ThinkPad-T420:~$ ls
Desktop  Documents  Downloads  Dropbox  examples.deskto
lucy@lucy-ThinkPad-T420:~$
```

Using the Man Pages

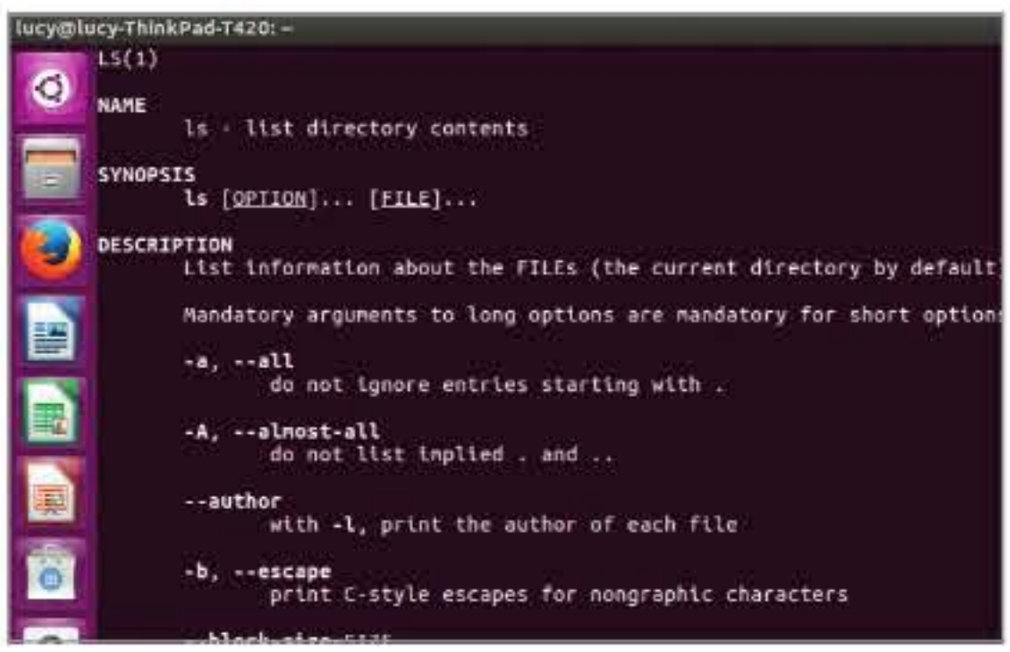
All Unix systems come with a powerful manual built into it, known as "man". Man pages are complex at first, but you will quickly get the hang of them. Knowing how man works enables you to get information and advice on most Linux commands.

WHAT'S UP, MAN?

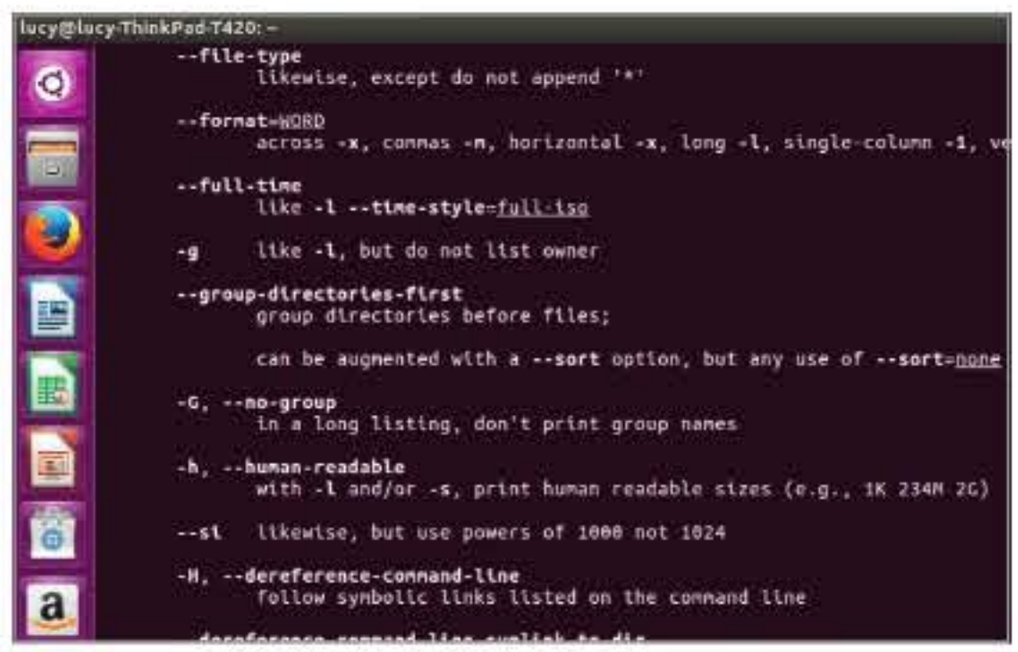
You've learnt some pretty cool stuff about commands, options and arguments so now is a great time to start looking at the built in help tool known as man, short for manual. With man you can get information about the Unix commands.

STEP 1 Unix has a built in manual, known as man for short. With the man command you can get information on all the Unix commands we've talked about. Just enter man and the name of the command you want to learn more about. Start by entering `man ls` in the command line.

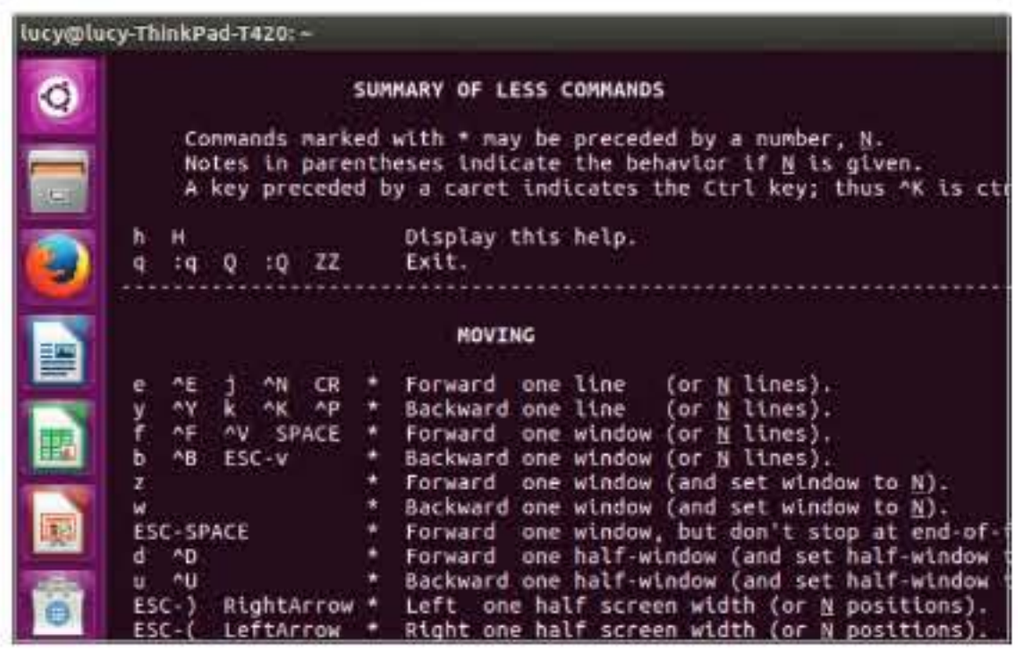
STEP 2 The man pages are a bit more detailed than you might be used to. First you have a name, which tells you what the command is called, in this case "list directory contents" and then the Synopsis shows you how it works. In this case: ls [OPTION].. [FILE..]. So you enter ls followed by options (such as -la) and the file or directory to list.



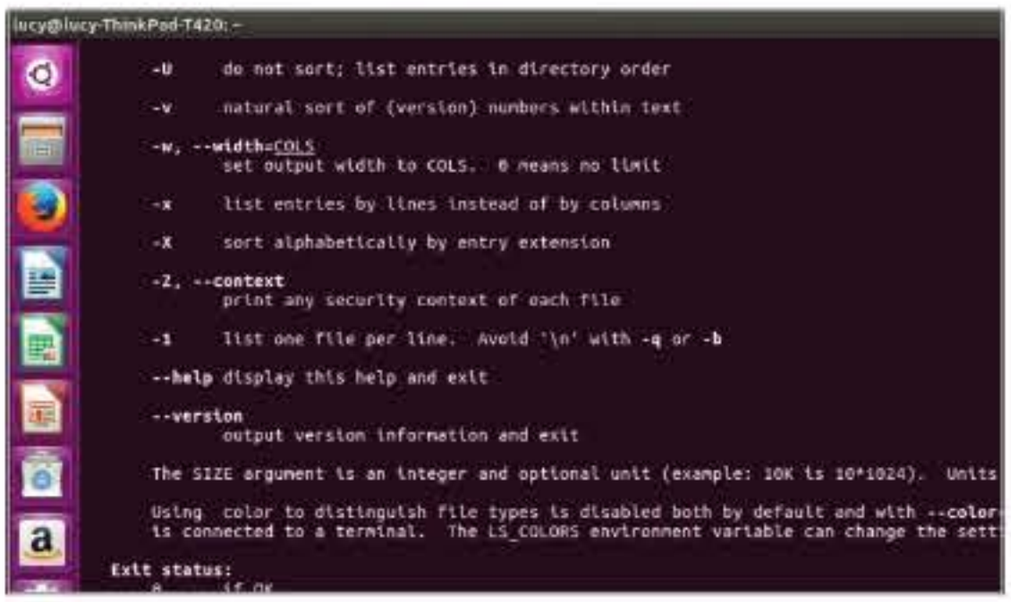
STEP 3 Most commands are pretty easy to figure out how to use, so what you spend most of the time in the man pages is looking under the Description. Here you will see all the options and the letters used to activate them. Most man pages are longer than a single page, so press any key (such as the space bar) to move to the next page of content.



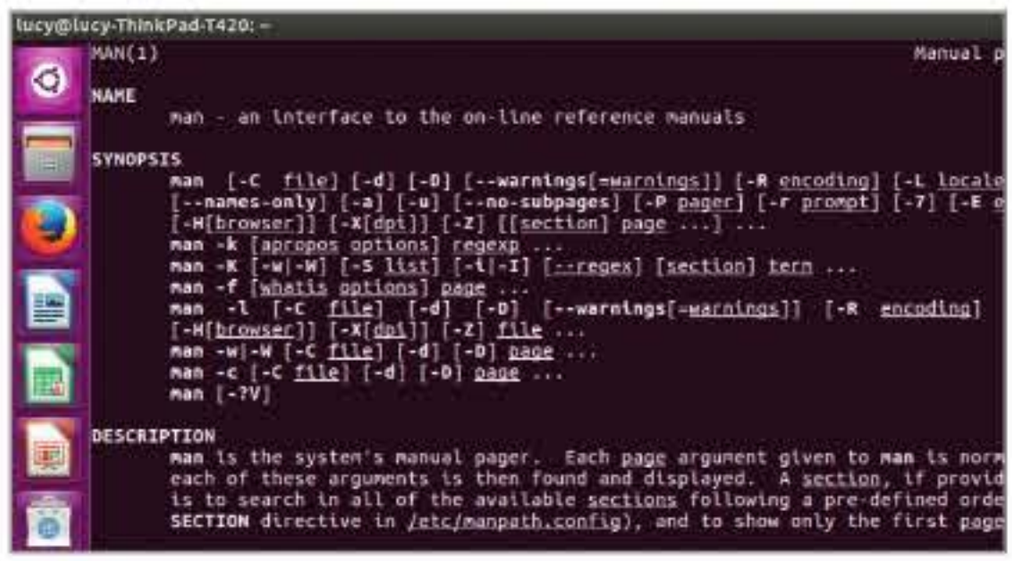
STEP 4 Press the `h key` while looking at a man page to view the commands you can use to control the view. This is called the Summary of Less Commands; the less command is something we'll come to when we look at editing text. For now realise that you can move back and forward with z and w. Press `q` to quit this help screen and return to the man page.



STEP 5 Scroll to the bottom of the man page to discover more information. Typically you will find the author's name and information on reporting bugs including web links that can be useful for more information. Press `q` to exit the man page and return to the command line.



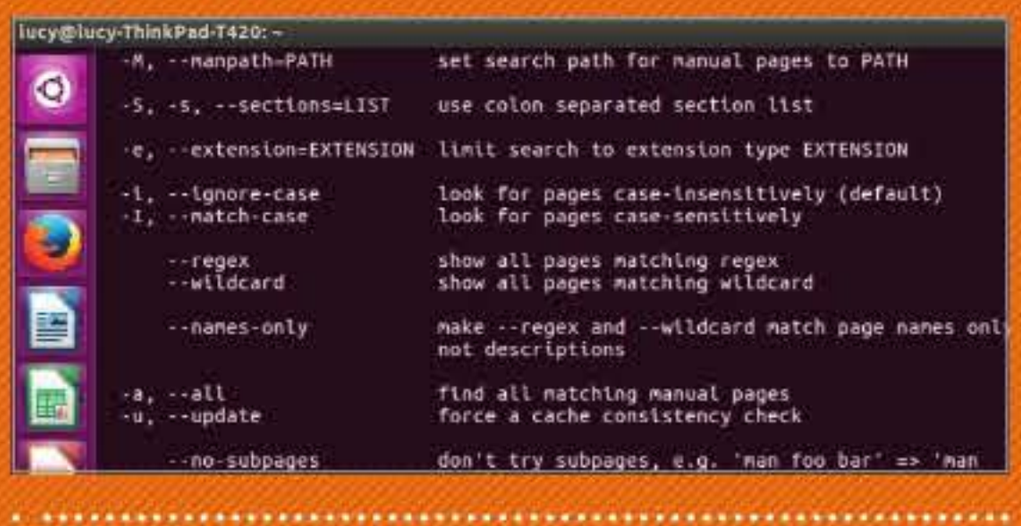
STEP 6 The man command can be used for just about every command you use in Unix. You can even enter `man man` to get information on using the man tool. From now on, whenever you come across a new command in this book, such as nano or chmod take time to enter `man nano` or `man chmod` and read the instructions.



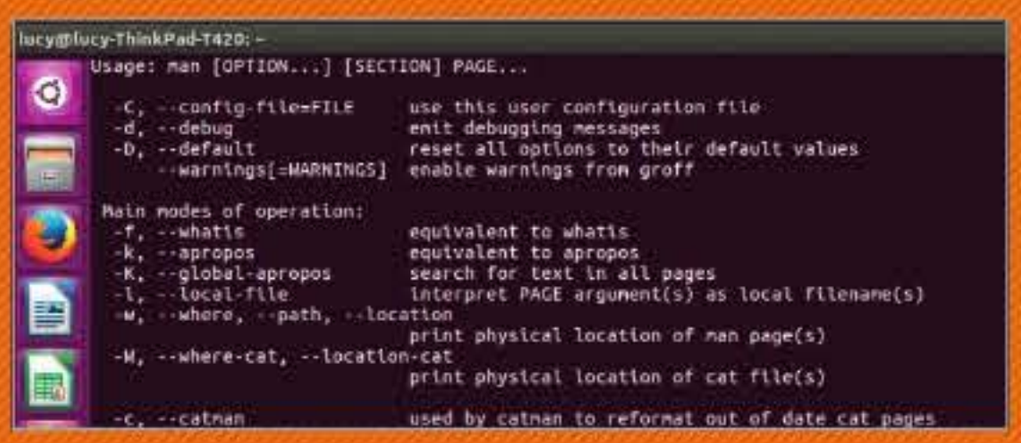
USING MAN OPTIONS

Man doesn't change anything, like mv or mkdir, so it's tempting not to see it as a command. It is though and like all other commands it has options and these can be very handy to learn.

STEP 1 Entering `man man` enables you to view some of the options but sometimes you'll just want a quick overview. Fortunately man has a built-in help option that quickly lists the options. Press q if you're in a man page and enter man -h at the command line.



STEP 2 If you're fast you may have noticed the start of the text flew up off the page. This is because the man -h option doesn't use the less command by default; less is what enables you to move down text one screen at a time. We'll look into pipes ("|") later on but for now just use `man -h | less` to read long text one page at a time.



STEP 3 One of the most powerful man options is the -k option, which is for "apropos". This enables you to search a wider range of man pages than the exact command. Enter `man -k directory` to view all of the man pages relating to directories; `man -k directory | less` to view one page at a time. Here you'll find commands like ls, mkdir and cd along with their description.

STEP 4 Entering the man page for all the commands you come across can be a little long-winded, although ultimately productive. If you just want to know what a command does you can read just the description using the whatis command. Enter whatis pwd to read the description of the pwd command ("print name of current/working directory").

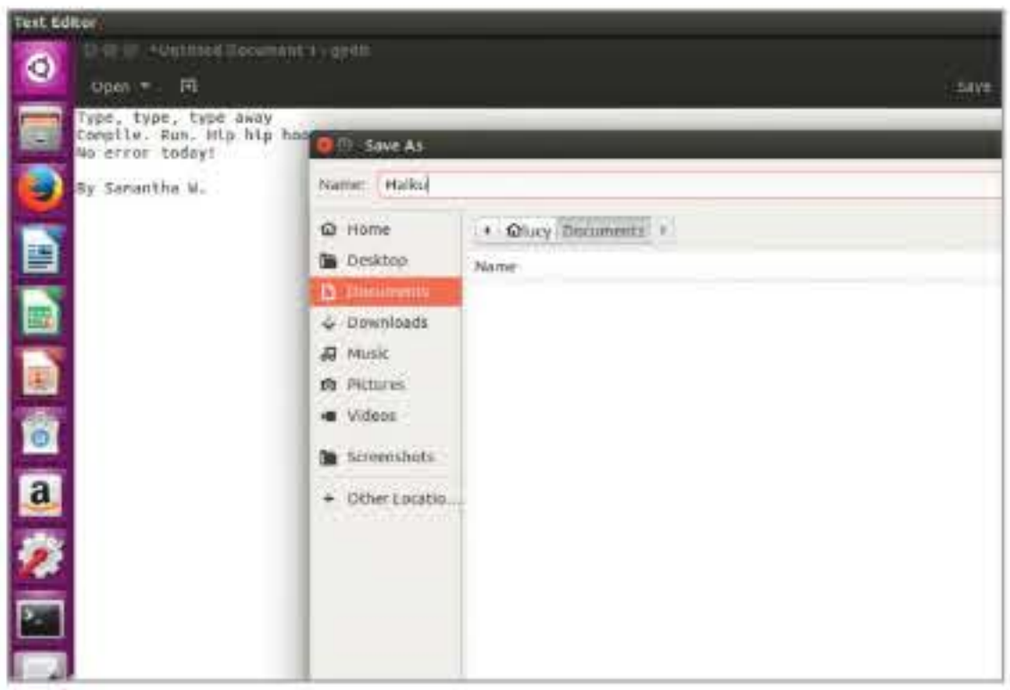
Editing Text Files

Unlike a graphical interface, you need to ask Unix to list and display the files and directories in the command line interface. List is one the Unix commands that you will use the most, so it's important to truly understand how to use List properly.

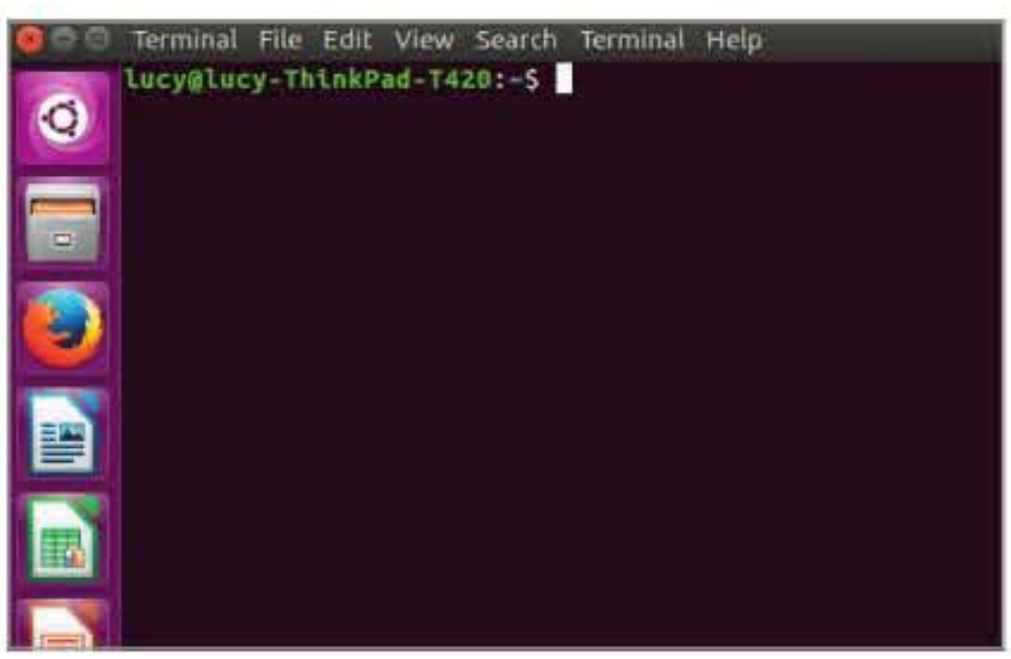
TEXT EDITING

Files in Unix are all constructed with text as well as code. Often you'll be asked to open a file and insert or adjust the text inside. As you get more advanced you'll be creating and editing configuration files and your own programs.

STEP 1 You are probably used to editing files in your Unix environment using a text program, such as Text Editor. Click on Dash and search for "Text Editor". We've written out a Haiku poem and saved it using File > Save As. Enter the name Haiku, choose the Documents directory and then click Save.

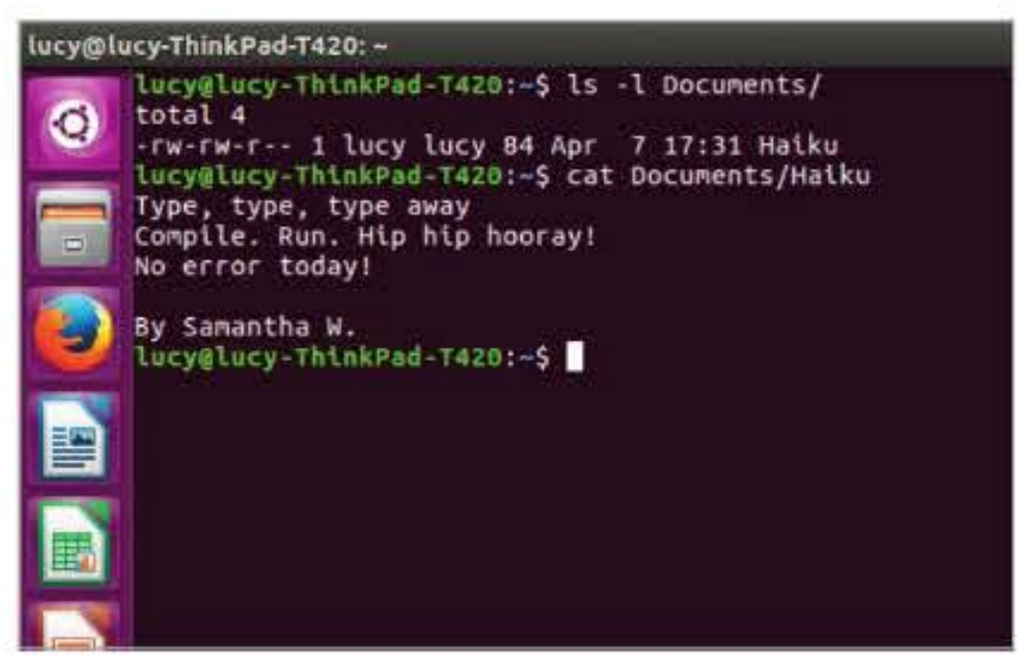


STEP 2 You're going to back out of the desktop environment and into the command line. We just want you to see that the text documents created in Text Edit contain just text and show you the relationship between files in the desktop and command line. Open Terminal and return to the command line.



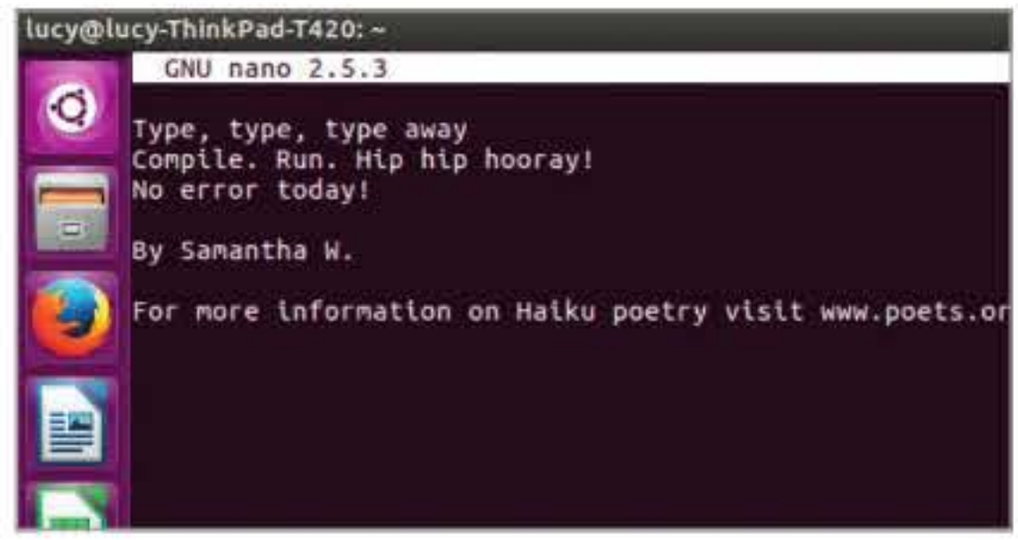
STEP 3 Enter `clear` to remove any text from the screen and `ls -l` Documents to view the files in your Documents directory. You will see the Haiku file you created in text edit. Note the number to the right of your user name. If you entered the document with the same text as us, it should read 84. This is the exact amount of characters in our poem.

STEP 4 The cat command can be used to view the contents of a text file in Unix. Enter `cat Documents/Haiku` to view the text inside the Haiku file. Cat is short for concatenate and was designed to list the text from various files together. You can use several files as arguments for cat but most people just use it to list the contents of a single file.



STEP 5 What if you want to edit a file? For that turn to a text editor that works from the command line. There are two main text editors available: Vim and Nano. We believe Nano is best to start with, so enter `nano Documents/Haiku`. The command line vanishes to reveal the GNU nano text editor.

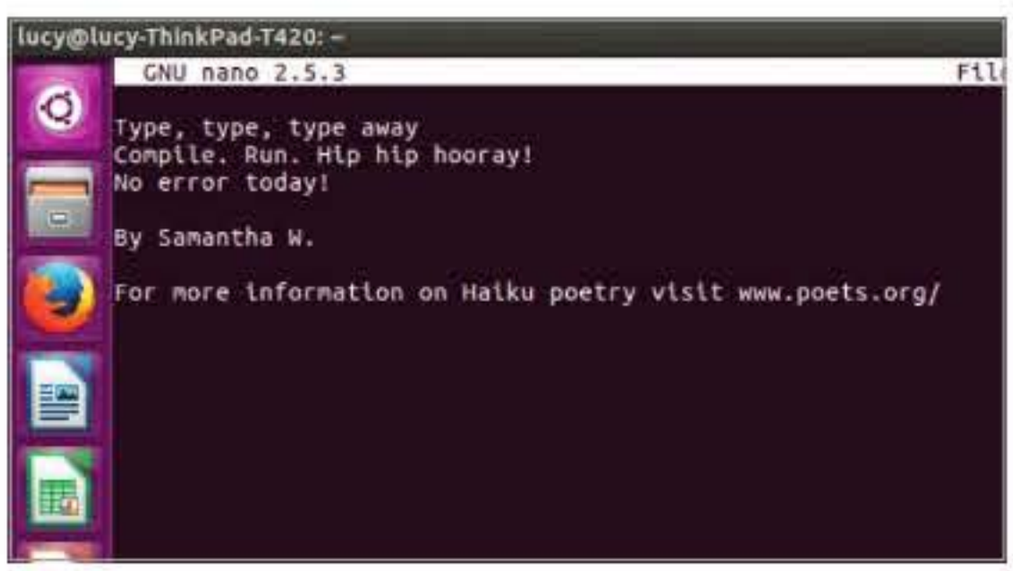
STEP 6 Text is edited in Nano in much the same way as other word processors, although it is far more limited in scope. Use the arrow keys to move to the end of the text and press return to add a couple of lines. Now enter some more text and it will be added to the document.



SAVING EDITED FILES

Nano is such an old program that many of its conventions are odd. You might find saving and closing the app tricky at first but don't worry; follow these steps to get the hang of it.

STEP 1 You don't save the text using the traditional Control-S function found in programs worldwide. Instead you WriteOut a file using a Control-O shortcut. Press Control-O and a bar appears at the bottom of the screen with File Name to Write and the document path.



STEP 2 By default the file name should be correct in the File Name To Write section and all you need to do is check it and press return. We found this strangely challenging at first and often reached for the y key to confirm the save. Once it has saved you quit Nano by pressing Control-X and not the usual Control-Q command.

STEP 3 Enter `cat Documents/Haiku` to view the updated text. Enter `ls -l Documents` to view the updated Haiku file in the Documents directory. Check the file size and you'll see it has increased by the same amount of letters, including returns at the end of lines, you entered into the Haiku document. Remember that one byte equals one letter in a document.

STEP 4 You should be aware that you also use Nano to create empty text documents. Type `nano Haiku` in the home directory and you'll create a blank document called Haiku instead of opening your document. If you ever try to open a document and get blank text, chances are you used the wrong directory path.

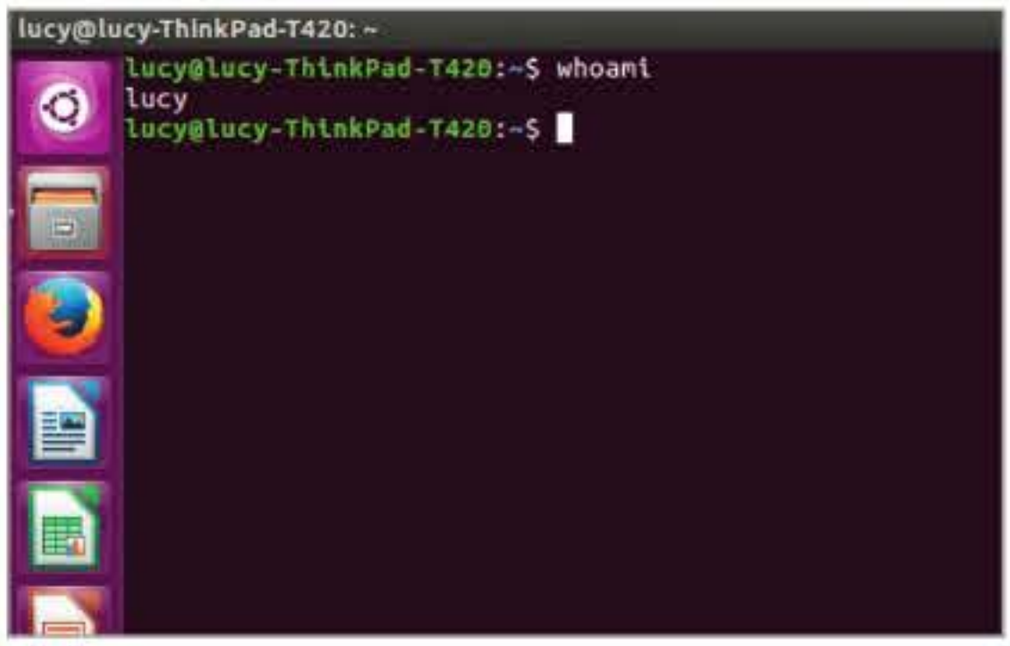
Getting to Know Users

Like all modern operating systems, Unix enables multiple users to use a computer and for each to have their own files and Home directory. This is handy on most computers but in Unix it's a good idea to really learn how different user accounts work.

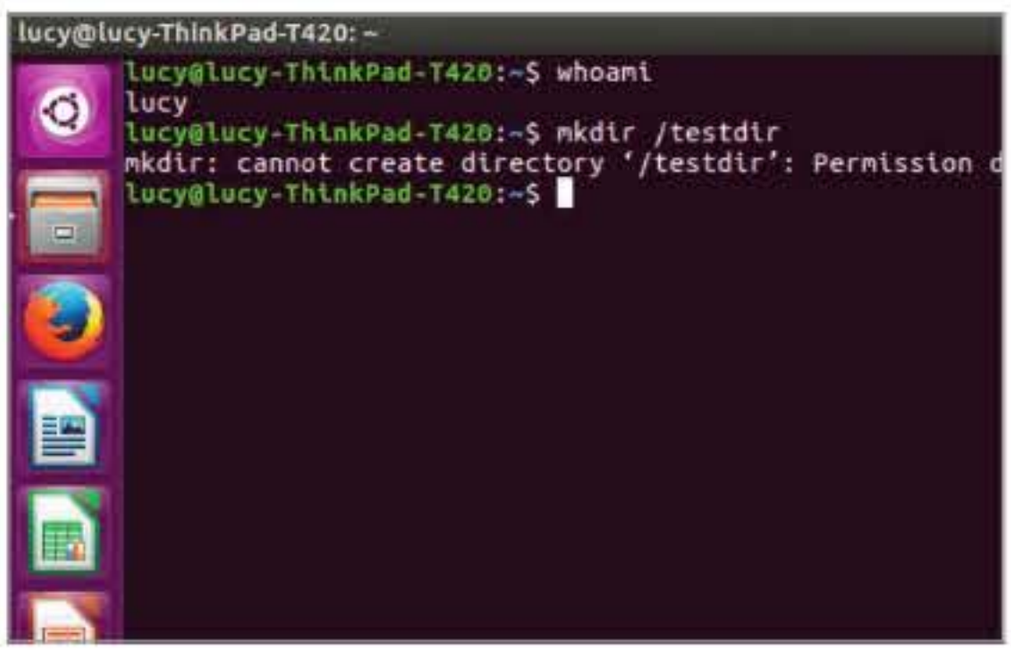
WHAT IS A USER?

An important part of using Unix is the concept of users and understanding which user you are, and which group you belong to. Like all operating systems you can have multiple user accounts and each has different types of access.

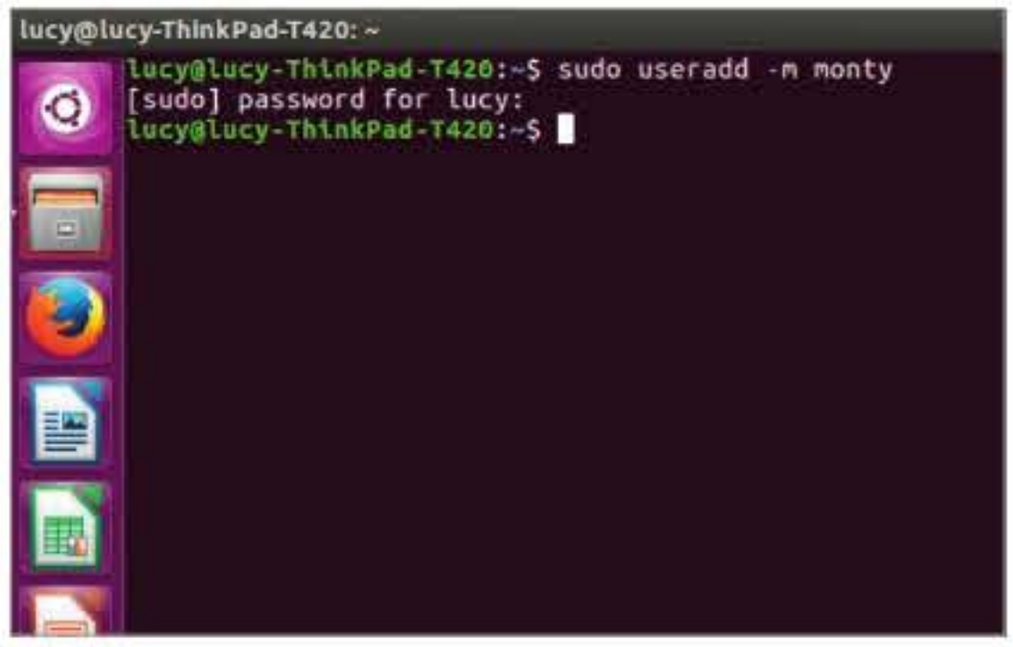
STEP 1 The first thing you need to do is get a concept of which user you are. Enter `whoami` into the command line and press return. It should say your account name; usually the name you entered during setup. The whoami command might seem a bit simplistic but it comes in very handy when you start using multiple accounts.



STEP 2 There are times when you'll be doing something in Unix and get an error with "Permission denied"; typically when you try to create, edit or execute (run) a file or directory outside of your area of privilege. If you want to see this, enter `mkdir /testdir`. Attempting to create a new directory in the root directory isn't allowed.



STEP 3 To do this you'll need to use a command called sudo. It stands for "substitute user do" and if you don't specify a user it uses the account above: the all-powerful root user. You'll come across sudo a lot in Unix, so let's create a second account to get the hang of it. Enter `sudo useradd -m monty` or pick your name. Enter your password.



STEP 4 Now add a password for the new account. Enter `sudo passwd monty` and enter a short password. Retype the same password and you'll now have two accounts on your machine. Now enter `ls -l /home` to view the home directories for both users. Notice that the lucy directory lists lucy as the owner and group; and monty directory belongs to monty.

STEP 5 Let's try switching to our new account. Enter `su monty` and enter the password you just created for that account. Notice that the command line now says "monty@[computername] but the working directory is still the same (your original accounts home directly, /home/lucy for us). Enter `whoami` to confirm you are now monty.

```
monty@lucy-ThinkPad-T420: /home/lucy
lucy@lucy-ThinkPad-T420:~$ su monty
Password:
monty@lucy-ThinkPad-T420:/home/lucy$ whoami
monty
monty@lucy-ThinkPad-T420:/home/lucy$
```

STEP 6 We'll look at permissions in the next tutorial but for now try to create a file as before. Enter touch testfile to create a file. It will say "touch: cannot touch 'testfile': Permission denied". This is because your new user account doesn't have the right to create files in the /home/monty directory. Enter `su` and your original username (from step 1) to switch back, i.e. `su lucy`.

```
lucy@lucy-ThinkPad-T420: ~
monty@lucy-ThinkPad-T420:/home/lucy$ touch testfile
touch: cannot touch 'testfile': Permission denied
monty@lucy-ThinkPad-T420:/home/lucy$ su lucy
Password:
lucy@lucy-ThinkPad-T420:~$
```

GETTING SUDO

We now have two accounts on the computer: lucy and monty, that can edit files in their own /Home directories. There's also a third account called root that sits above both lucy and monty. It can edit files anywhere.

STEP 1 The root account is all-powerful. It is possible but not recommended to switch to the root account, although you'll need to give it a password first, using sudo passwd root. Then just type su to switch to root. Please don't do this though. Knowledge is a good thing but it is safer and wiser to use sudo instead.

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~$ su passwd root
```

STEP 2 Most people think sudo stands for super user but it stands for substitute user do. It enables you to perform a command as another user. Enter `sudo -u monty touch /home/monty/test` to create a file inside Monty's home directory. You won't get an error because you substituted yourself for monty, who has permission to edit that directory.

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~$ sudo -u monty touch /home/mo
lucy@lucy-ThinkPad-T420:~$
```

STEP 3 It's rare that you use sudo to substitute another user. If you don't specify a user using the -u option with a user name it defaults to the root account (as if you'd typed sudo -u root). Enter sudo touch /home/monty/anothertestfile to create a file in the lucy directory while still using the monty account.

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~$ sudo touch /home/monty/anoth
lucy@lucy-ThinkPad-T420:~$
```

STEP 4 This step is optional. Only your original user account can use sudo. If we want to give the monty account sudo privileges it needs to be added to the sudoers file. Enter `sudo visudo` to view the sudoers file. Add `monty ALL=(ALL) NOPASSWD: ALL` to the last line and use Control-o to output the file. Remove the .tmp that is added to the file name as a security measure. Note that most accounts are not added to the sudoers file as a matter of course.

```
lucy@lucy-ThinkPad-T420: ~
  GNU nano 2.5.3                                    File: /etc/sudoers.
#
# This file MUST be edited with the 'visudo' command as root.
#
# Please consider adding local content in /etc/sudoers.d/ instead of
# directly modifying this file.
#
# See the man page for details on how to write a sudoers file.
#
Defaults        env_reset
```

Ownership and Permissions

Who owns files and who's allowed to change them is an important part of Unix. It's also one of its more complex elements, especially when viewing and managing permissions from the command line

OWNER AND PRIVILEGE

More importantly is the concept of users, who owns files and who has privilege rights. Like all modern computers there are multiple user accounts and each has different types of access. This is paramount to managing files.

STEP 1 If you followed the previous tutorial you should now have two accounts on your computer. One called monty and the other with your name (lucy in our case). An essential aspect of Unix is the idea of file and directory ownership. Who owns and has access to what. We need a test file, so enter `touch testfile.txt` to create one.

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~$ touch testfile.txt
lucy@lucy-ThinkPad-T420:~$
```

STEP 2 Now enter `ls -l` and lets have a good look at the default permissions file. Our testfile.txt files starts with the text -rw-r--r--. Let's start with the first letter, which is a dash '-'. All the other items in our home directory are directories. You can tell because they are blue and our testfile.txt file is white.

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~$ touch testfile.txt
lucy@lucy-ThinkPad-T420:~$ ls -l
total 52
drwxr-xr-x  2 lucy lucy 4096 Apr  6 15:13 Desktop
drwxr-xr-x  2 lucy lucy 4096 Apr  7 17:37 Documents
drwxr-xr-x  2 lucy lucy 4096 Apr  6 18:17 Downloads
drwx------ 28 lucy lucy 4096 Apr  8 14:24 Dropbox
-rw-r--r--  1 lucy lucy 8980 Apr  6 15:01 examples.desktop
drwxr-xr-x  2 lucy lucy 4096 Apr  7 16:19 Music
-rw-rw-r--  1 lucy lucy    0 Apr  7 16:59 names
drwxrwxr-x  3 lucy lucy 4096 Apr  7 17:06 people
drwxr-xr-x  2 lucy lucy 4096 Apr  6 15:13 Pictures
drwxr-xr-x  2 lucy lucy 4096 Apr  6 15:13 Public
drwxr-xr-x  2 lucy lucy 4096 Apr  6 15:13 Templates
-rw-rw-r--  1 lucy lucy    0 Apr  7 17:00 testfile
-rw-rw-r--  1 lucy lucy    0 Apr  8 14:26 testfile.txt
drwxr-xr-x  2 lucy lucy 4096 Apr  7 16:19 Videos
lucy@lucy-ThinkPad-T420:~$
```

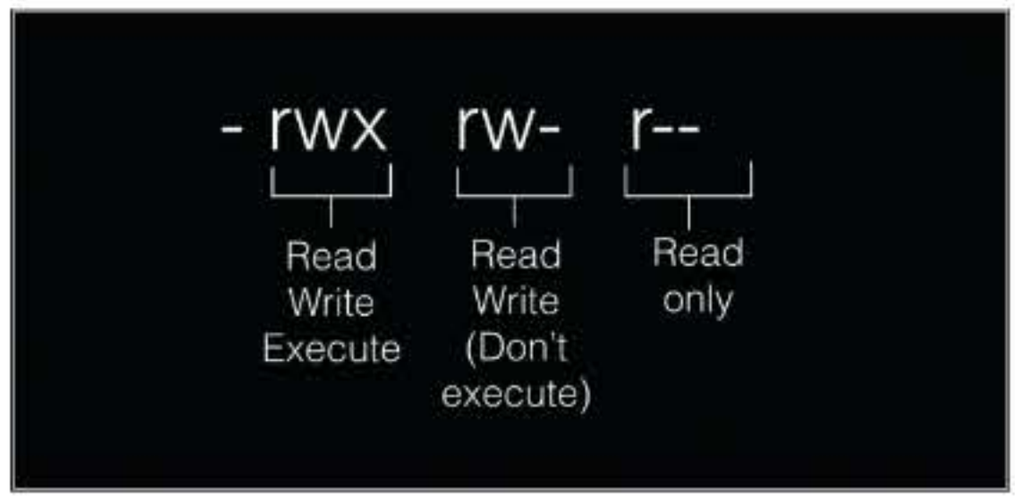
STEP 3 The first letter in the permissions also indicates a directory or file. Notice that all the other files start with a 'd' and our textfile.txt file starts with a '-'. That's what the first letter means. It's either a 'd' in which case it's a directory or a '-' in which case it's not a directory, it's a file. Enter `ls -l testfile.txt` to view the permissions for just this file.

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~$ ls -l
total 52
drwxr-xr-x  2 lucy lucy 4096 Apr  6 15:13 Desktop
drwxr-xr-x  2 lucy lucy 4096 Apr  7 17:37 Documents
drwxr-xr-x  2 lucy lucy 4096 Apr  6 18:17 Downloads
drwx------ 28 lucy lucy 4096 Apr  8 14:24 Dropbox
-rw-r--r--  1 lucy lucy 8980 Apr  6 15:01 examples.desktop
drwxr-xr-x  2 lucy lucy 4096 Apr  7 16:19 Music
-rw-rw-r--  1 lucy lucy    0 Apr  7 16:59 names
drwxrwxr-x  3 lucy lucy 4096 Apr  7 17:06 people
drwxr-xr-x  2 lucy lucy 4096 Apr  6 15:13 Pictures
drwxr-xr-x  2 lucy lucy 4096 Apr  6 15:13 Public
drwxr-xr-x  2 lucy lucy 4096 Apr  6 15:13 Templates
-rw-rw-r--  1 lucy lucy    0 Apr  7 17:00 testfile
-rw-rw-r--  1 lucy lucy    0 Apr  8 14:26 testfile.txt
drwxr-xr-x  2 lucy lucy 4096 Apr  7 16:19 Videos
lucy@lucy-ThinkPad-T420:~$ ls -l testfile.txt
-rw-rw-r-- 1 lucy lucy 0 Apr  8 14:26 testfile.txt
lucy@lucy-ThinkPad-T420:~$
```

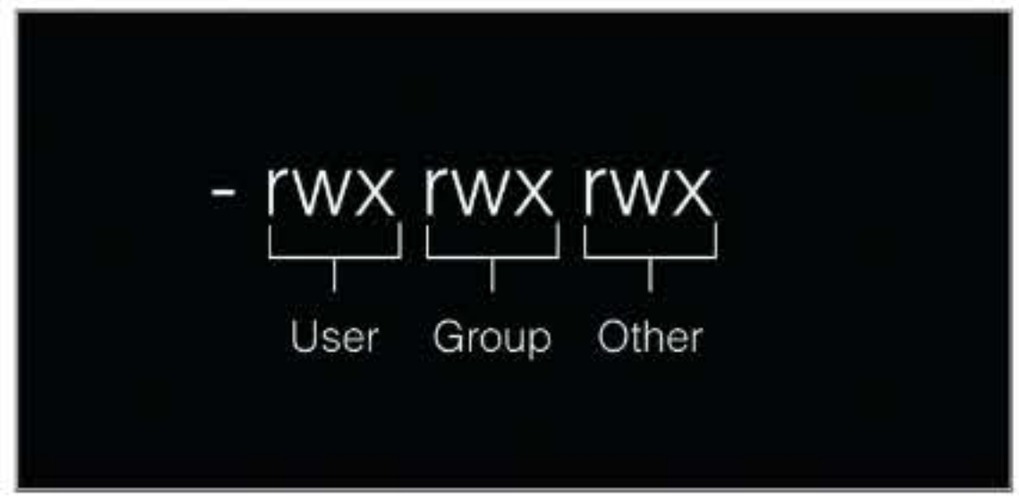
STEP 4 The next nine letters of the permissions are known as "alpha notation" because they let you know the permissions using letters. Each permission is either on, in which case you see a letter or it is off, in which case you see a dash. The letter doesn't change for each place. So the first permission, the second letter after the directory one, is either an 'r' or a '-'. It's never any other letter.

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~$ ls -l testfile.txt
-rw-rw-r-- 1 lucy lucy 0 Apr  8 14:26 testfile.txt
lucy@lucy-ThinkPad-T420:~$
```

STEP 5 The 'r' means that particular permission is read and it's set to on. The nine letters here are divided into three groups of three letters: r, w and x. They stand for read, write and execute (run). Read means the file can be viewed, using cat or nano; w means the file can be edited or moved; and x means the file, typically a script or program, can be run.



STEP 6 The presence of r, w, or x means that this aspect is possible, a dash means it isn't. Our testfile.txt has no x letter, so if it were a script it wouldn't run. So why are there so many letters? Why not just three? Read, Write and Execute? The three blocks of three letters are for different sets of people: user, group and other.

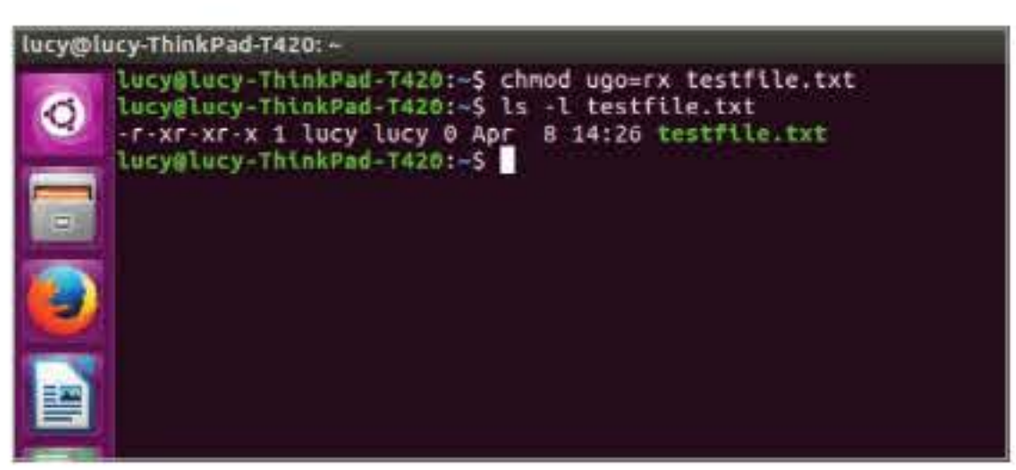


CHANGING PERMISSIONS

Now that you know how groups of permissions work, it's time to look at how to change them. Mastering this skill will help creating users, and giving them the correct access to resources, much easier.

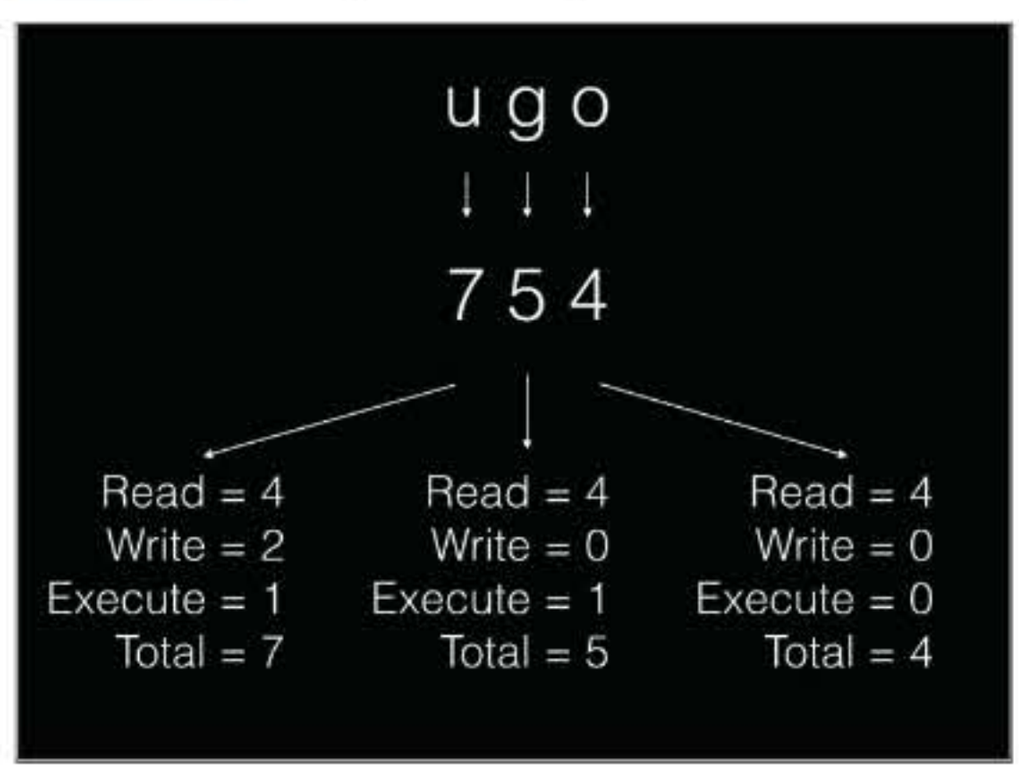
STEP 1 The first block of three is the most important. This is the user who owns the file (typically your name); the second is for other people in the same group as the user; and the third is for other people on the system. Permissions are changed using the chmod (change file mode bit) command. Enter `man chmod` to look at the manual.

STEP 2 The chmod command is one of the trickier ones to understand. There are two ways you can adjust permissions, the first is using chmod with an option to target one of the three groups: owner, group or other. For these you use u, g or o followed by = and the letters or dashes you want. So enter `chmod ugo=rx testfile.txt` to make all three groups read, write and execute.



STEP 3 Turning everything on is probably overkill, so you need to target each group. Do this by putting commas between each mode option. Enter `chmod u=rwx,g=rw,o=r testfile.txt` to make gives users read, write and execute privileges, user read and write and other just read. Enter `ls -l` to see your handiwork.

STEP 4 Alpha notation is fine but many Unix experts use octal notation instead. This is a three digit number that represents permissions. This is the formula: read=4, write=2 and execute=1 and you add them up for each group. So if a group is read, write and execute it's 7; if it's read and write it's 6; or if it's just execute it's 1. A popular option is 755. Enter `chmod 755 testfile.txt` to change the file using octal notation.



Useful System and Disk Commands

Now that you've learnt all the basics, it's time to start putting the command line to work. In this tutorial we'll cover some handy Unix commands that you can use on your computer. These commands enable you to investigate your computer and its hard drive.

LOTS OF UNIX

Unix is a huge and versatile command line language and there are hundreds of commands you can learn and use. Here are some of them that will help you get more from your computer.

STEP 1 Let's start by getting some info. Enter `cat /proc/cpuinfo` to view some info on your computer's processors. If you have a modern machine, you'll most likely see two to four processors, starting from number 0, along with the model name and other info.

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~$ cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 42
model name      : Intel(R) Core(TM) i5-2520M CPU @ 2.50GHz
stepping        : 7
microcode       : 0x29
cpu MHz         : 800.000
cache size      : 3072 KB
physical id     : 0
siblings        : 4
core id         : 0
cpu cores       : 2
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
```

STEP 2 Remember that cat is used to list the contents of a text file, which is what cpuinfo is. There are other text files with system info available. Try `cat /proc/meminfo` to get information about your memory, `cat /proc/partitions` for information about your SD Card and `cat /proc/version` shows which version of Unix you are using.

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~$ cat /proc/meminfo
MemTotal:        3926288 kB
MemFree:         2448360 kB
MemAvailable:    3063804 kB
Buffers:          124168 kB
Cached:           636084 kB
SwapCached:            0 kB
Active:           847292 kB
Inactive:         466472 kB
Active(anon):     555616 kB
Inactive(anon):   173528 kB
Active(file):     291676 kB
Inactive(file):   292944 kB
Unevictable:          48 kB
Mlocked:              48 kB
SwapTotal:       4073468 kB
SwapFree:        4073468 kB
Dirty:               116 kB
```

STEP 3 Enter `uname` to view the name of the operating system's kernel, the bit that sits between the interface and hardware. This is great but you're more likely to want to know which version of Unix you are using. Enter `cat /etc/*-release file` to view information on your Unix distribution.

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~$ uname
Linux
lucy@lucy-ThinkPad-T420:~$ cat /etc/*-release file
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=16.04
DISTRIB_CODENAME=xenial
DISTRIB_DESCRIPTION="Ubuntu Xenial Xerus (development branch)"
NAME="Ubuntu"
VERSION="16.04 (Xenial Xerus)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 16.04"
VERSION_ID="16.04"
HOME_URL="http://www.ubuntu.com/"
SUPPORT_URL="http://help.ubuntu.com/"
BUG_REPORT_URL="http://bugs.launchpad.net/ubuntu/"
UBUNTU_CODENAME=xenial
cat: file: No such file or directory
lucy@lucy-ThinkPad-T420:~$
```

STEP 4 Enter `uname -a` to view some more detailed information. Here you'll see the kernel name, hostname and kernel version (4.4.0-16-generic) on ours. If you have a multicore computer you'll see SMP (symmetric multiprocessing), followed by the system date, CPU architecture and operating system (GNU/Unix).

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~$ uname -a
Linux lucy-ThinkPad-T420 4.4.0-16-generic #32-Ubuntu SM
lucy@lucy-ThinkPad-T420:~$
```

STEP 5 Enter `cat /sys/class/thermal/thermal_zone0/temp` to view the current operating system temperature of your computer. If you want clearer info, install lm-sensors (sudo apt-get install lmsensors). Enter `sudo sensors-detect` to setup `lm-sensors` and answer yes and no to the questions. Enter `sensors` to run it.

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~$ cat /sys/class/thermal/thermal_zone
39000
lucy@lucy-ThinkPad-T420:~$ sensors
acpitz-virtual-0
Adapter: Virtual device
temp1:        +39.0°C  (crit = +98.0°C)

thinkpad-isa-0000
Adapter: ISA adapter
fan1:         1972 RPM

coretemp-isa-0000
Adapter: ISA adapter
Physical id 0:  +42.0°C  (high = +86.0°C, crit = +100.0°C)
Core 0:         +39.0°C  (high = +86.0°C, crit = +100.0°C)
Core 1:         +42.0°C  (high = +86.0°C, crit = +100.0°C)

lucy@lucy-ThinkPad-T420:~$
```

STEP 6 One command you might be wondering about is how to switch off or restart your computer from the command line. Don't just hit the power switch. Enter `sudo showdown -h` now to shut down computer. The -h option stands for halt. Alternatively enter `sudo shutdown -r now` or `sudo reboot` to restart your computer.

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~$ sudo shutdown -h now
```

DISK COMMANDS

Learn the two commands that enable you to view your disk space and the files on it: df (disk free space) and du (disk usage). With these two commands you can view the file usage on your hard drive.

STEP 1 Start by entering `df` in the command line. It returns a list of the volumes contained on your computer. You might be wondering what a volume is. It's best to imagine your hard drive containing partitions. This is where you split one drive to act like two or more drives. Each partition can contain volumes; these are storage spaces.

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~$ df
Filesysten     1K-blocks     Used Available Use% Mounted on
udev             1943840        0   1943840   0% /dev
tmpfs             392632     6368    386264   2% /run
/dev/sdb1       57398540 10173636  44286180  19% /
tmpfs            1963144      384   1962840   1% /dev/shm
tmpfs               5120        4      5116   1% /run/lock
tmpfs            1963144        0   1963144   0% /sys/fs/cgroup
tmpfs             392632       52    392580   1% /run/user/1000
/dev/sdc2       155916896       64 155916832   1% /media/lucy/1126-4DED
lucy@lucy-ThinkPad-T420:~$
```

STEP 2 Enter `df -h` to get the list in human readable form. The first two lines should read "rootfs" and "/dev/root" and have matching Size, Used, Avail and Use% listings. This is the main drive and is an indication of how much space you have used, and have free. The other volumes are for booting and initialising devices; you can ignore these for now.

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~$ df -h
Filesystem      Size  Used Avail Use% Mounted on
udev            1.9G     0  1.9G   0% /dev
tmpfs           384M  6.3M  378M   2% /run
/dev/sdb1        55G  9.8G   43G  19% /
tmpfs           1.9G  312K  1.9G   1% /dev/shm
tmpfs           5.0M  4.0K  5.0M   1% /run/lock
tmpfs           1.9G     0  1.9G   0% /sys/fs/cgroup
tmpfs           384M   52K  384M   1% /run/user/1000
/dev/sdc2       149G   64K  149G   1% /media/lucy/1126-4DED
lucy@lucy-ThinkPad-T420:~$
```

STEP 3 Now enter `du`. You should see lots of text fly up the screen. This is the disk usage for the files contained in your home directory and their sub-directories. As with df, it is better to use du with the -h option to humanize the output. If you want to slow down the output, you'll also need to pipe it through less. Enter `du -h | less` to view the files and their respective usage one page at a time.

```
lucy@lucy-ThinkPad-T420: ~
22184   ./.cache/mozilla/firefox/1078migj.default/cache2/entries
22224   ./.cache/mozilla/firefox/1078migj.default/cache2
204     ./.cache/mozilla/firefox/1078migj.default/thumbnails
6688    ./.cache/mozilla/firefox/1078migj.default/safebrowsing
1224    ./.cache/mozilla/firefox/1078migj.default/startupCache
30352   ./.cache/mozilla/firefox/1078migj.default
30356   ./.cache/mozilla/firefox
30360   ./.cache/mozilla
4       ./.cache/unity-control-center/webkit
8       ./.cache/unity-control-center
4       ./.cache/deja-dup/f16f99d3da0035745c350c16977336e6
8       ./.cache/deja-dup
1176    ./.cache/thumbnails/large
8       ./.cache/thumbnails/fail/gnome-thumbnail-factory
```

STEP 4 You don't typically enter du on its own; most of the time you want to view the disk usage of a specific directory. Enter `du -h Documents/` to view how much space the Documents directory takes up. Ours is just "8.0K". If you want a more comprehensive breakdown of the files contained, use the -a option (all). Enter `du -ha` to view all the files contained and their disk usage.

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~$ du -ha Documents/
4.0K    Documents/Haiku
8.0K    Documents/
lucy@lucy-ThinkPad-T420:~$
```

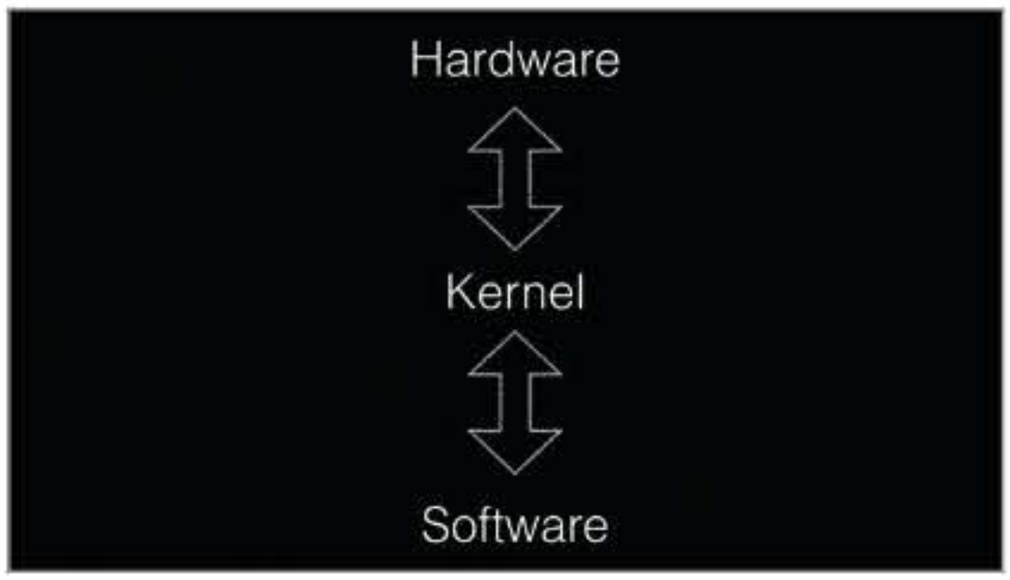
Managing Programs and Processes

Programs and processes are complex creatures in Unix. Most operating systems hide processes from users and just display programs (running or not). In Unix you really need to learn how to inspect and stop processes. This is an essential skill.

PROGRAMS AND PROCESSES

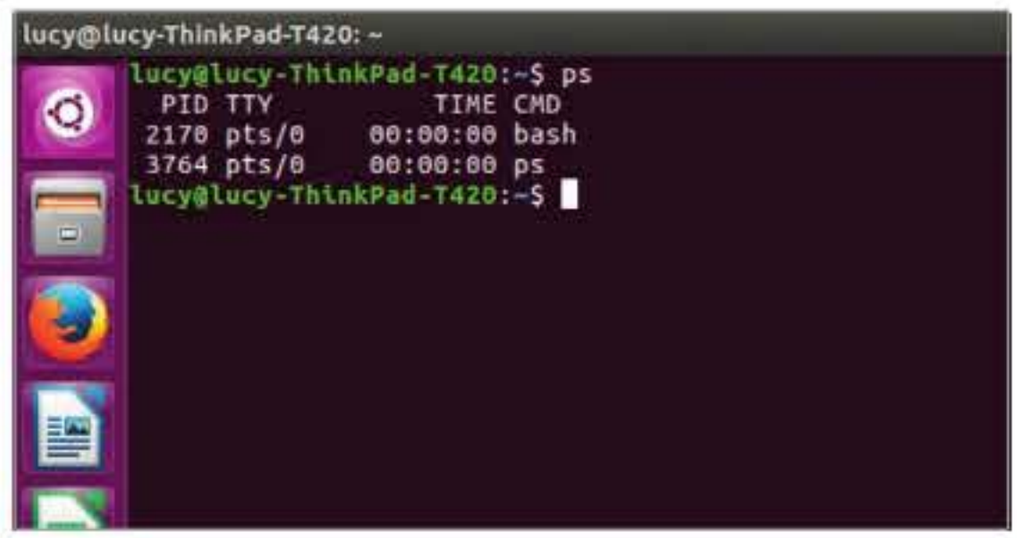
If you've come from a desktop computing environment, you'll be used to the idea of a program that you open and quit using the interface. This program is managed by the operating system and controls the program; but what on earth is a process?

STEP 1 As you get into Unix you'll start to hear more about processes and the kernel. The kernel sits beneath the software and hardware. It passes instructions to the hardware running processes, which takes up memory and when the process is finished it closes it and reclaims the memory.

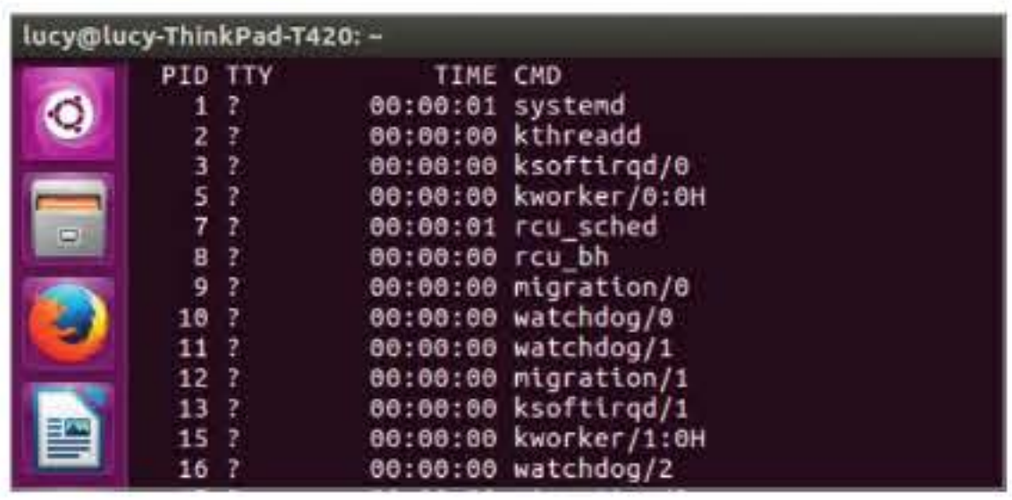


STEP 2 You're probably used to thinking in terms of programs and most OS's tend to keep processes out of sight. In Unix on the other hand, you're right in at the deep end. You don't open and quit programs as much as manage your processes. This is done using the `ps` (process status) command.

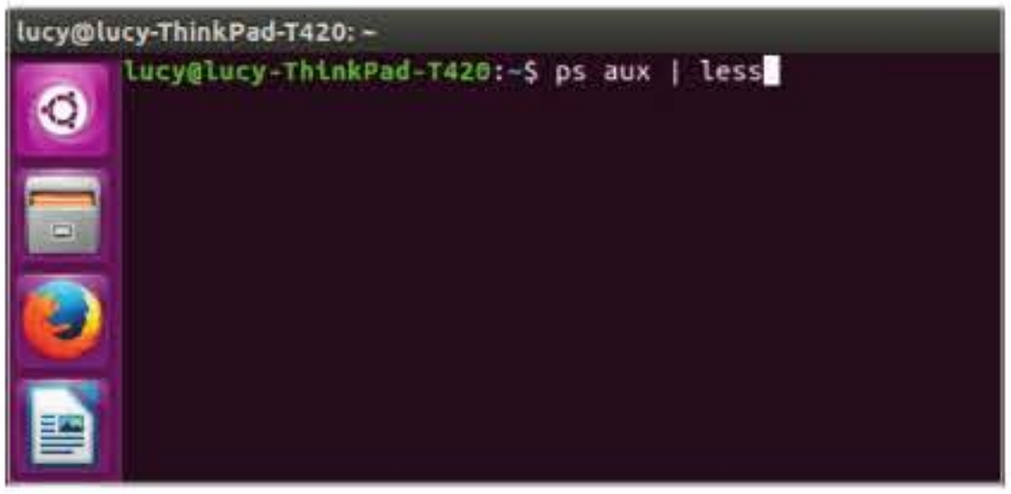
STEP 3 If you type in ps on its own you don't see much. You should see two items, bash and ps. Bash (Bourne Again Shell) is the command line environment you are typing in and ps is the command you just entered. If that seems a little light, that's because it is. These are just the processes owned by the user and running in the foreground.



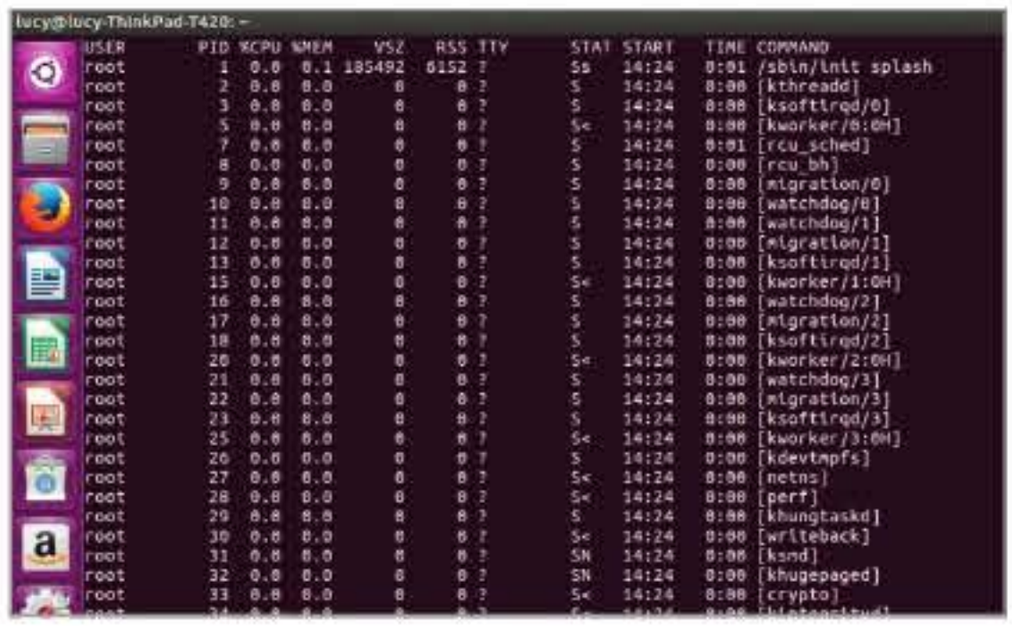
STEP 4 If you want to see processes used by other users, including those started by root, enter `ps -a`. This option stands for all users but still isn't everything because it doesn't include background processes. For this you can enter either `ps -A` or `ps -e`. This will show you every process on the system including the background processes. You may need to pipe it through less using `ps -e | less`.



STEP 5 The ps command is one of the oldest known and has a lot of legacy; because of this it also has alternative options that can be used without the dash at the start. The most common way to use ps is to enter `ps aux`. Piping this command through less is the best way to go about using ps. Enter `ps aux | less` to view your processes.



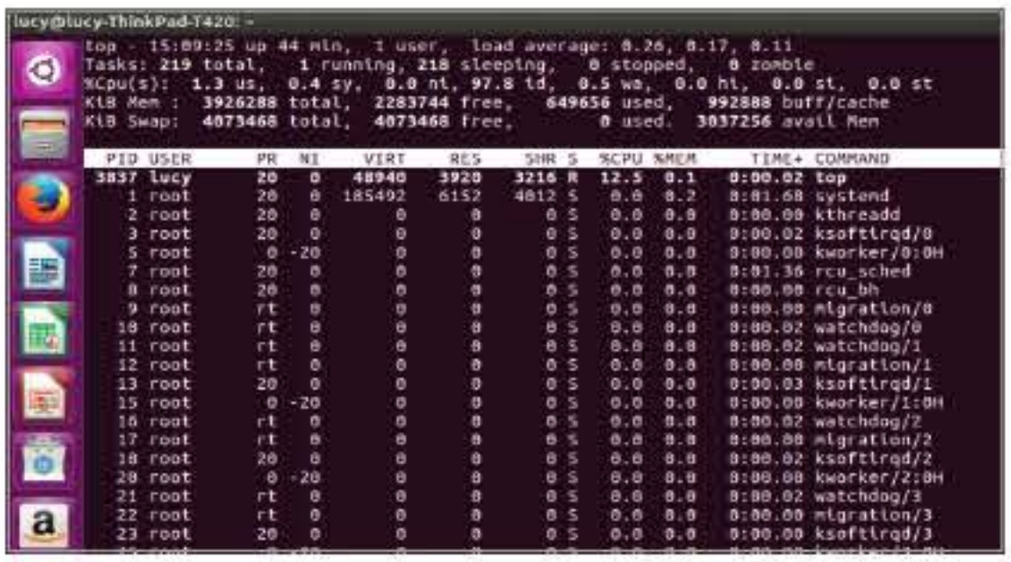
STEP 6 The ps aux command displays the processes in columns. The first is the User followed by the Process ID (PID). Then we see the percentage of CPU and memory.



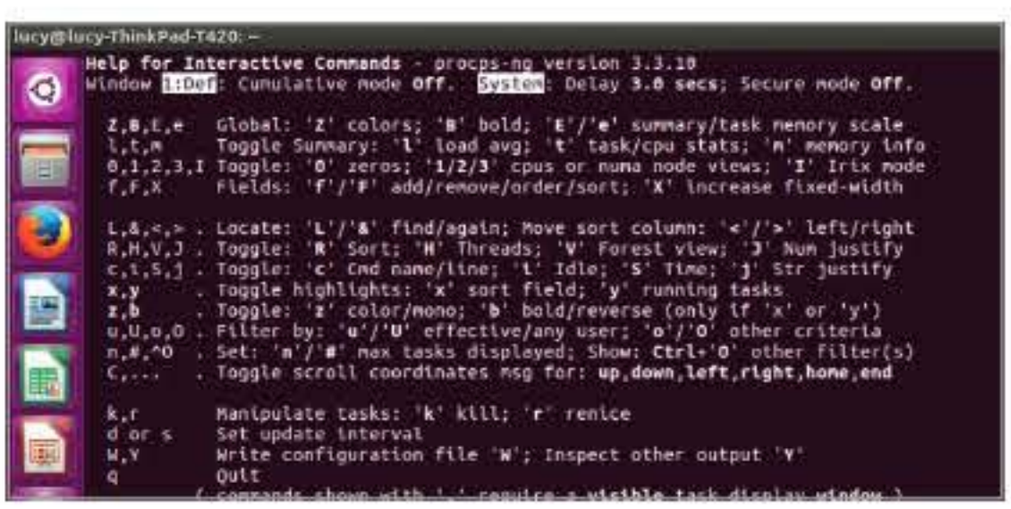
VIEWING AND QUITTING PROCESSES

Now that you have a handle on processes, you need to know what to do with them. You can quit any available process running in real-time and also ones that you no longer need, or those that are problematic.

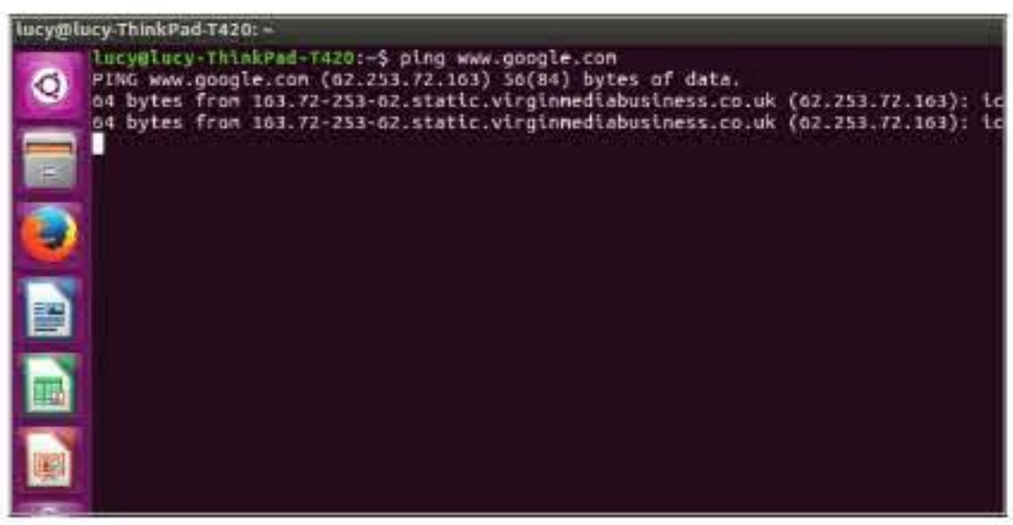
STEP 1 While ps is a great tool for checking all the processes on your computer, sometimes you need to view them in real-time. In particular you may want to check the CPU and memory usage of processes and see if any process is hogging all your computer's resources. Use `top` to do this.



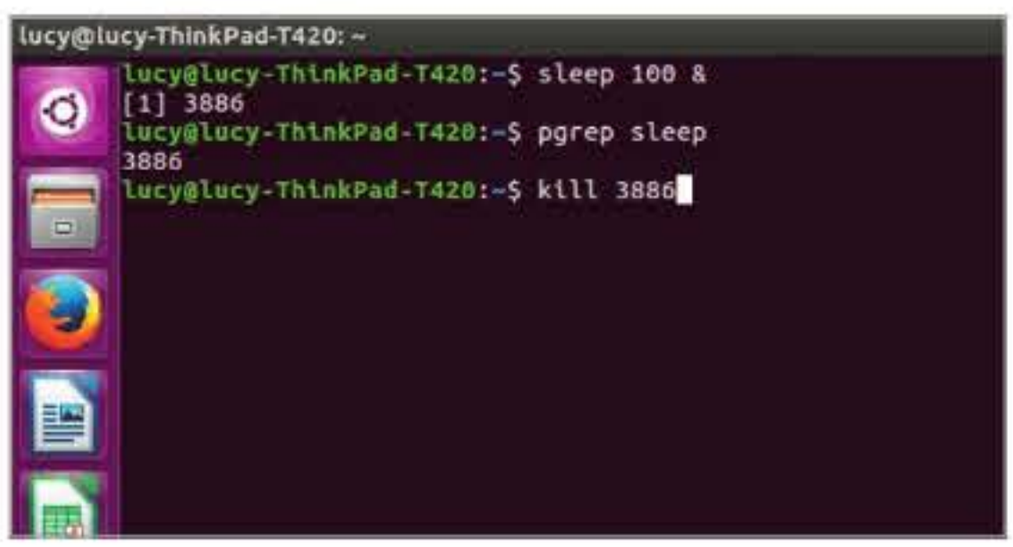
STEP 2 Top fills the display with processes fitting as many as it can on one screen. These run in real time so as you watch, the display items will move up and down according to their usage. You can interact with top as it is running: us `<` and `>` to change the sort column. Press h to view a help page with all the commands you can use.



STEP 3 Now that you've got an idea of what processes are, how do you go about stopping them. Some processes run in the foreground until you quit them. Enter `ping www.google.com` and you'll start a ping process continuously contacting Google. This sort of process runs until you press Control-C.



STEP 4 Processes are stopped using the kill command. Enter `sleep 100 &` to create a dummy process. Enter `pgrep sleep` to find its PID number; on ours it is 3886. Now enter `kill 3886` and the sleep process will be stopped. You can also use `pkill sleep` if you're sure you know the process name.



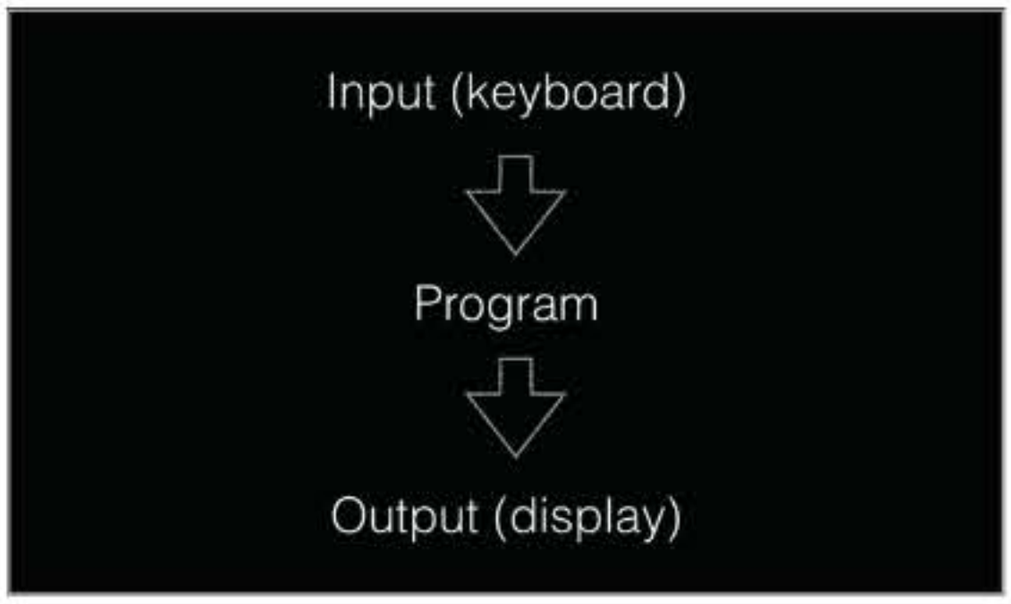
Input, Output and Pipes

Now you're really becoming a Unix ninja. You've learnt how to enter complex commands in Unix, so let's discover something a bit more in-depth. In this tutorial we'll look at how to input and output commands to one another and string commands together using pipes.

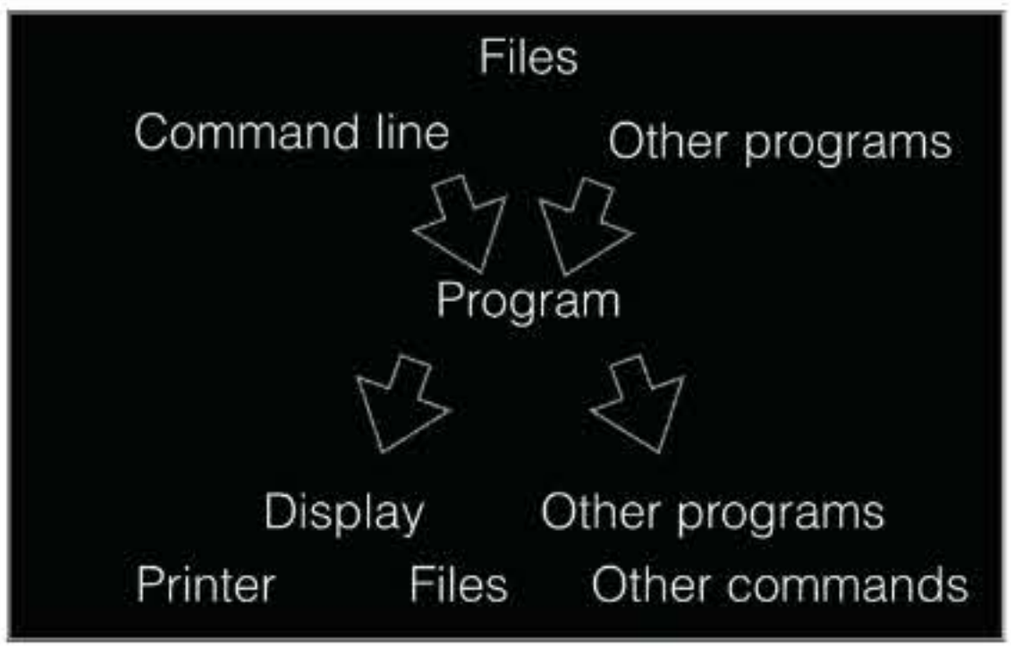
INPUT AND OUTPUT

Everything on a computer is about input and output. You input to the computer by pressing keys and moving the mouse; it makes calculations and outputs by changing the display, making a noise and so on.

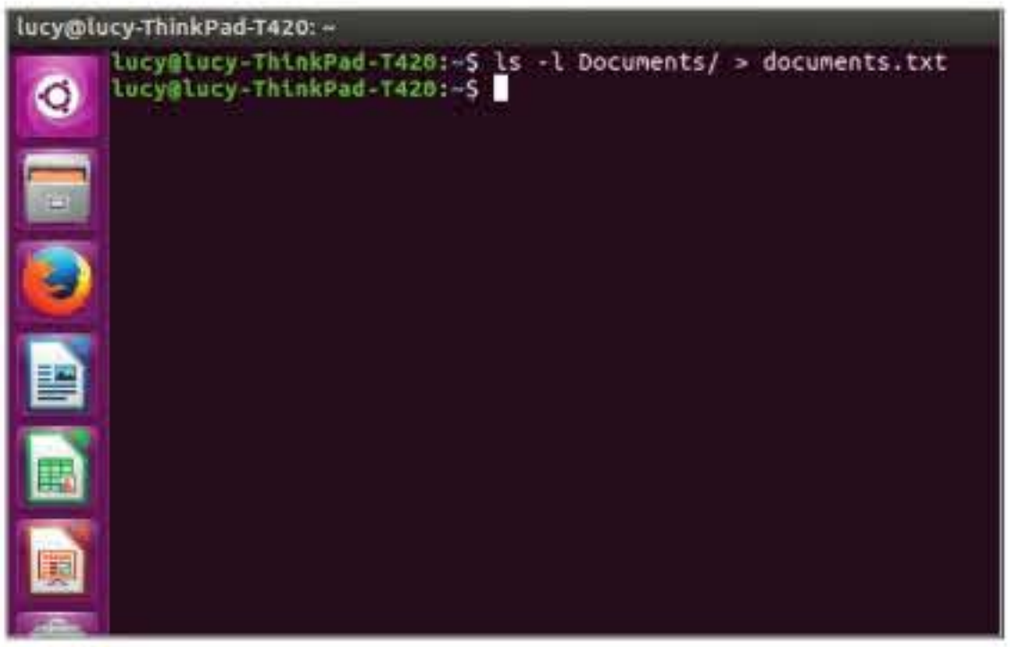
STEP 1 When you enter commands into the Unix command line you are using standard input and output. This is so obvious that most people don't even think about it. You enter commands using the keyboard, that's input and it responds on the screen, that's output. This is the regular way of doing it, which is why it's "standard input and output" (often called stdin and stdout for short).



STEP 2 As far as the computer is concerned, input and output can be to and from a whole range of different sources that you might never even think about. A program can get input from other programs, from files stored on the hard drive and a whole host of other areas. It outputs back to the display line and also to files, other programs and even other commands.



STEP 3 You can change the standard output to something else using the ">" character after your command. If we wanted to see all the items in the Documents directory we use `ls -l` Documents/. Using `ls -l Documents/ > documents.txt` outputs the list of items to a new text file called "documents.txt".



STEP 4 The documents.txt file now contains the output from the ls command. You can check it using `nano documents.txt`. It's an editable text file containing all the permissions, user and file size information and the names of the files. The output from the ls -l command, normally displayed on the screen, was sent instead to this file. Press `Control-x` to quit nano.

STEP 5 So ">" enables you to output to files but you can also get input from a file. Enter `cd Music/` to enter your Music directory. Enter `nano bands.txt` to create a new text file. Enter some band names and press Control-o to output the file. Press Control-x to quit Nano.

```
lucy@lucy-ThinkPad-T420: ~/Music
  GNU nano 2.5.3

Prince
ABBA
Blur
The Clash
David Bowie
The Smiths
```

STEP 6 You're going to use this text file as input to the sort command. Enter `sort < bands.txt` and the content from the text file is used as input to sort. The output isn't specified so it uses the standard output, the screen but you use input and output together. Enter `sort < bands.txt > bands_sorted.txt` to create a new file with the band names in order.

```
lucy@lucy-ThinkPad-T420: ~/Music
lucy@lucy-ThinkPad-T420:~/Music$ sort < bands.txt
ABBA
Blur
David Bowie
Prince
The Clash
The Smiths
lucy@lucy-ThinkPad-T420:~/Music$ sort < bands.txt > bands_sorted.txt
lucy@lucy-ThinkPad-T420:~/Music$
```

USING PIPES

As well as directing input and output to and from files, you can send the output from one command directly into another command. This is known as piping and uses the pipe character "|".

STEP 1 As you start to get more advanced in Unix you begin to create more powerful commands and one way you do this is by using the pipe character ("|"). Take some time to find this character if you haven't already; it usually sits above or to the left of the Return key on most keyboards.

```
lucy@lucy-ThinkPad-T420: ~/Music
lucy@lucy-ThinkPad-T420:~/Music$ ps aux | less
```

STEP 2 We've used the pipe a few times in this book (`ps aux | less`) but you might not have understood what's actually happening. Enter `cat bands.txt | wc`. The output from the cat command, the text inside the document, isn't displayed on the screen; instead it is piped into the wc (word count) function. This then tells us how many lines, words and characters are in the document.

```
lucy@lucy-ThinkPad-T420: ~/Music
lucy@lucy-ThinkPad-T420:~/Music$ cat bands.txt | wc
      6       9      50
lucy@lucy-ThinkPad-T420:~/Music$
```

STEP 3 You can pipe commands multiple times. Enter `cat bands.txt | sort | grep The*` to get the bands starting with "The" in alphabetical order. The output of the text from the bands.txt document is passed into sort and the output from sort is passed into grep that filters out the bands starting with "The". These bands form the output.

```
lucy@lucy-ThinkPad-T420: ~/Music
lucy@lucy-ThinkPad-T420:~/Music$ cat bands.txt | sort | grep The*
The Clash
The Smiths
lucy@lucy-ThinkPad-T420:~/Music$
```

STEP 4 You can combine pipes with input and output to create complex expressions. You can also use >> to append outputted data to a file that already exists. Enter `cat bands.txt | wc >> bands.txt`. This takes the output from the bands.txt file and pipes it into the wc (word count) function. The output from wc is appended to the end of the bands.txt file. Enter `cat bands.txt` to view it.

```
lucy@lucy-ThinkPad-T420: ~/Music
lucy@lucy-ThinkPad-T420:~/Music$ cat bands.txt | wc >> bands.txt
lucy@lucy-ThinkPad-T420:~/Music$ cat bands.txt
Prince
ABBA
Blur
The Clash
David Bowie
The Smiths
      6       9      50
lucy@lucy-ThinkPad-T420:~/Music$
```

Using Grep Pattern Matching

Searching for pattern matches in text can be really useful, which is where "grep" comes in. Grep is like the find command in a word processor, but on steroids (lots of steroids). It's an incredibly powerful search and replace tool that enables you to find just about anything.

WHAT THE GREP?

Grep is one of the most powerful commands you can learn in Unix. We mentioned it briefly in the last tutorial but now it's time to take a look at it properly. Standing for Global Regular Expression Print it enables you to search text and match complex patterns.

STEP 1

Grep is a powerful Unix tool that enables you to search through text and find a match. This is tremendously useful as it enables you to look for files on your system and search through the text in files. We are starting in the Music/ directory from the previous tutorial with two text files.

```
lucy@lucy-ThinkPad-T420: ~
lucy@lucy-ThinkPad-T420:~$ echo XenialXerus
XenialXerus
lucy@lucy-ThinkPad-T420:~$ echo XenialXerus | grep Xerus
XenialXerus
lucy@lucy-ThinkPad-T420:~$
```

STEP 2

Think of Grep as a filter between your input and output. Grep takes the input from something and provides a filter. We can see this in action with the echo command. First enter `echo XenialXerus` and press return. The output is the string "XenialXerus". Let's pipe this through grep: `echo XenialXerus | grep Xerus`. The string "XenialXerus" is still returned but the "Xerus" part is highlighted in red.

```
lucy@lucy-ThinkPad-T420: ~/Music
lucy@lucy-ThinkPad-T420:~/Music$ cat bands.txt | grep The
The Clash
The Smiths
lucy@lucy-ThinkPad-T420:~/Music$ grep The bands.txt
The Clash
The Smiths
lucy@lucy-ThinkPad-T420:~/Music$
```

STEP 3

You can also use grep to match the text in files. You can do this using cat and a pipe: `cat bands.txt | grep The`. You don't need to use pipe cat to grep, you can instead pass in the file as an argument after your search term. Enter `grep The bands.txt` to achieve the same result.

```
lucy@lucy-ThinkPad-T420: ~/Music
lucy@lucy-ThinkPad-T420:~/Music$ echo The Kinks >> bands.txt
lucy@lucy-ThinkPad-T420:~/Music$ grep The bands.txt
The Clash
The Smiths
The Kinks
lucy@lucy-ThinkPad-T420:~/Music$
```

STEP 4

Grep doesn't match whole words. By default it finds the text you entered in the input. Enter `echo Them >> bands.txt` to add the band Them to our text file. Now enter `grep The bands.txt` and you'll get bands like The Kinks along with Them. The -w option matches whole words. Enter `grep -w The bands.txt` to just find bands starting with "The".

```
lucy@lucy-ThinkPad-T420: ~/Music
lucy@lucy-ThinkPad-T420:~/Music$ echo Them >> bands.txt
lucy@lucy-ThinkPad-T420:~/Music$ grep The bands.txt
The Clash
The Smiths
Them
lucy@lucy-ThinkPad-T420:~/Music$ grep -w The bands.txt
The Clash
The Smiths
lucy@lucy-ThinkPad-T420:~/Music$
```

STEP 5 If you're unsure of the location of your text file, you can use grep to search continuously for files with matching text using the R option. Enter `grep -R "The Smiths" /home/lucy/` (replacing "lucy" with your username) to search for files containing the text "The Smiths". We're using quote marks around The Smiths to search for both words together rather than all files containing either Smiths or The.

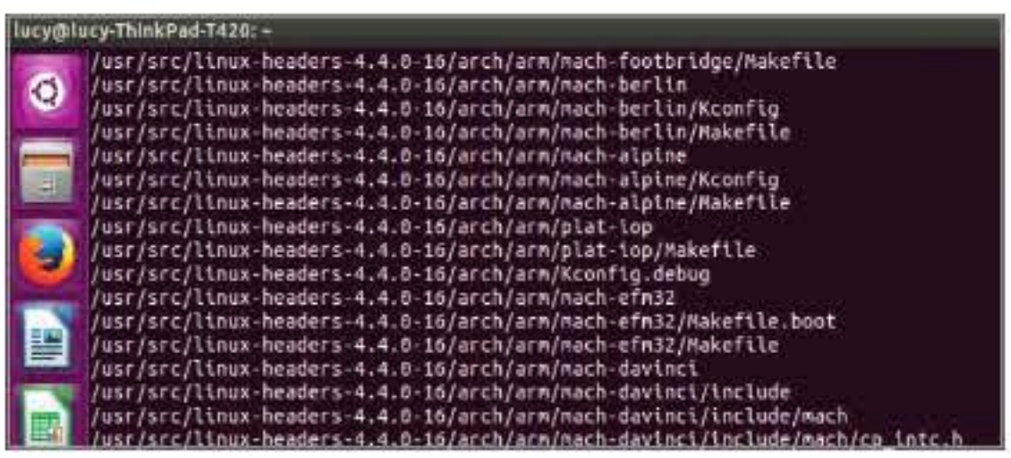
STEP 6 There's a lot more to grep pattern matching than we could possible mention here in fact we could write a whole book on it. There are lots of characters you can use, along with the asterisk wildcard ("*"). The dot (".") matches any single character and "?" can be used to find characters that may or may not appear. There's a whole lot more too. Enter man grep to read the manual. Ryans Tutorials has a good outline if you're interested in learning more (ryanstutorials.net/Unixtutorial/grep.php).

FINDING FILES

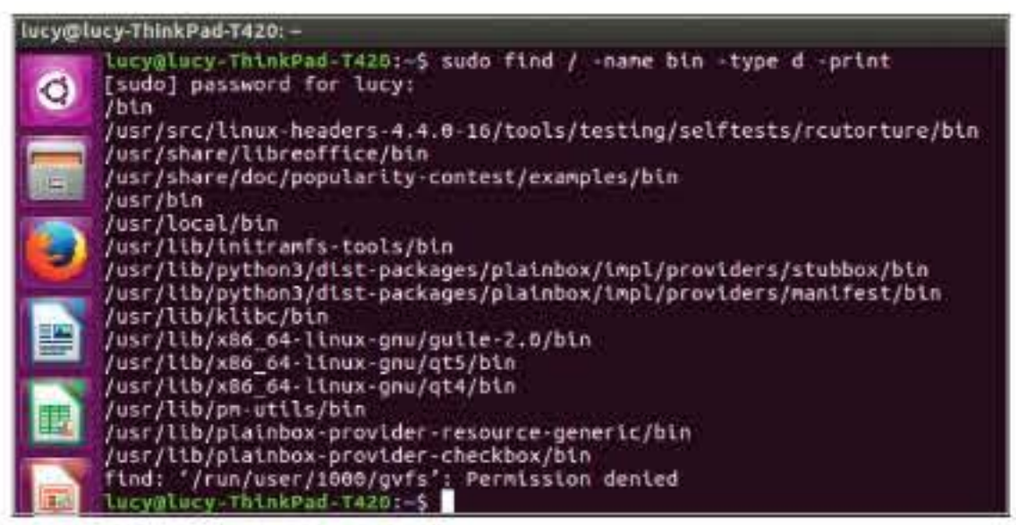
Grep is a powerful tool for searching for text, and many Unix users combine it with other commands to find files. There is a better way though, using the built-in Find command. Let's start finding files in Unix.

STEP 1 You might be tempted to use grep with a pipe and ls to find files and directories. This is possible but a better way to locate filenames to use the find command. The format is find, followed by the directory (use a dot (.) for the current directory) then -name and the filename, followed by -print. Navigate to the home folder (`cd ~`) and enter `find . -name "*.txt" -print`. This lists all the text files in your home directory.

STEP 2 Notice that find didn't just return results from your home directory but all the directories below. Unlike other Unix commands, find uses recursion by default. You can list all the files in your home directory using `find . -print` from the home directory lists every file in your home. Enter `find / -print` to list every file on your computer; this could take some time – press Ctrl-c to quit.



STEP 3 You can also find files by type using the -type option and a letter; common ones are "d" for directory and "f" for file. This enables you to search through your whole system for directories. Enter `sudo find / -name bin -type d -print` to find all the bin (binary) folders on your system; you need sudo to search recursively through the root directory.



STEP 4 There are lots of other commands with find. So be sure to use `man find` to do some research. One option worth looking up is -mtime, which enables you to find files by the modification time. Enter `find . -mtime -2` to find files modified two days ago, or `find . -mtime 0` to find files modified today.

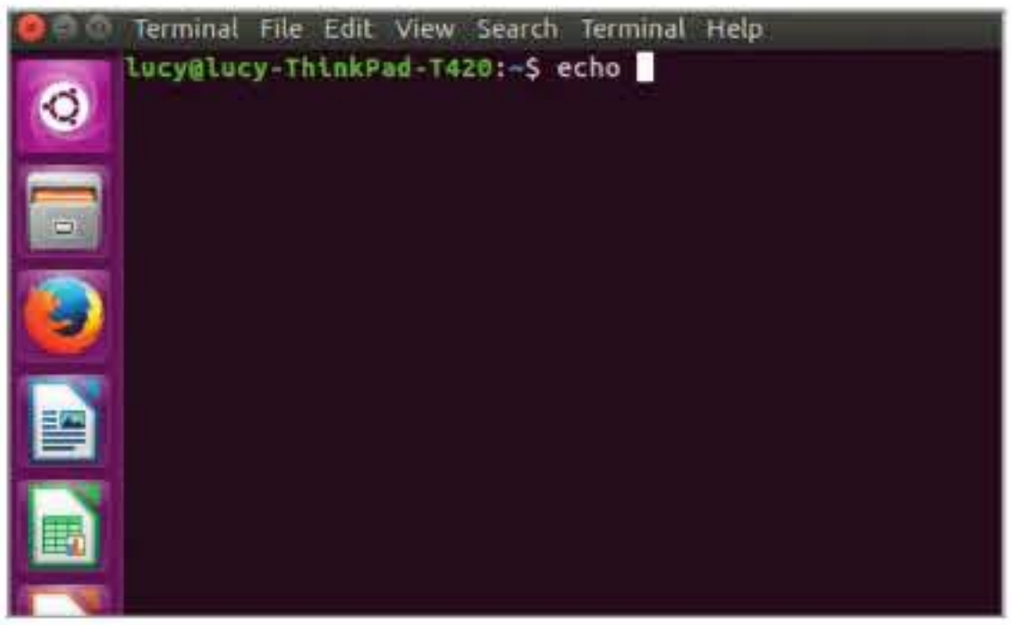
Tab Completion and History

Learning how to use Tab Completion and History will make you a much faster and better Unix user. These tips will enable you to skip the typing and get Unix to enter commands and filenames for you.

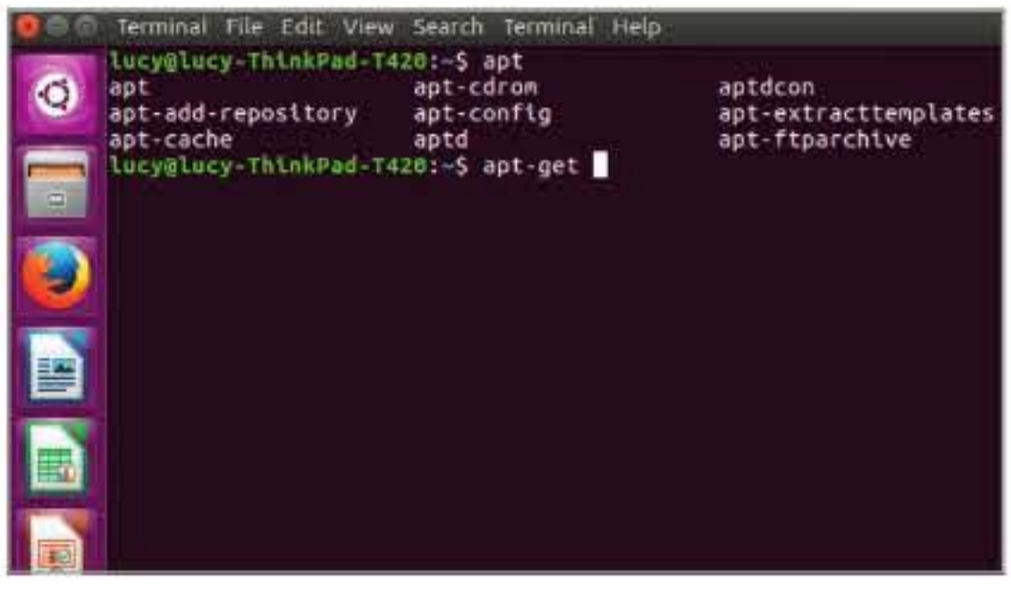
LOVING UNIX

As you use Unix more often, you'll increasingly find yourself working in the command line. These vital tips make the command line easier and more enjoyable, which in turn makes you a better Unix user.

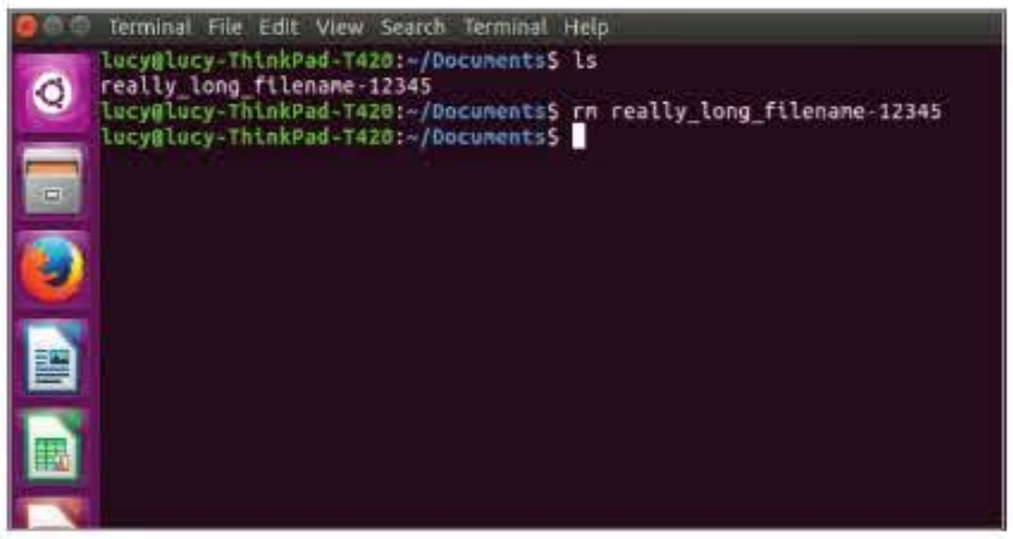
STEP 1 By far the most useful command line trick to learn properly is Tab Completion. Pressing the Tab button in the command line completes commands and file names. Enter `ec` in the command line and press Tab to expand it to "echo".



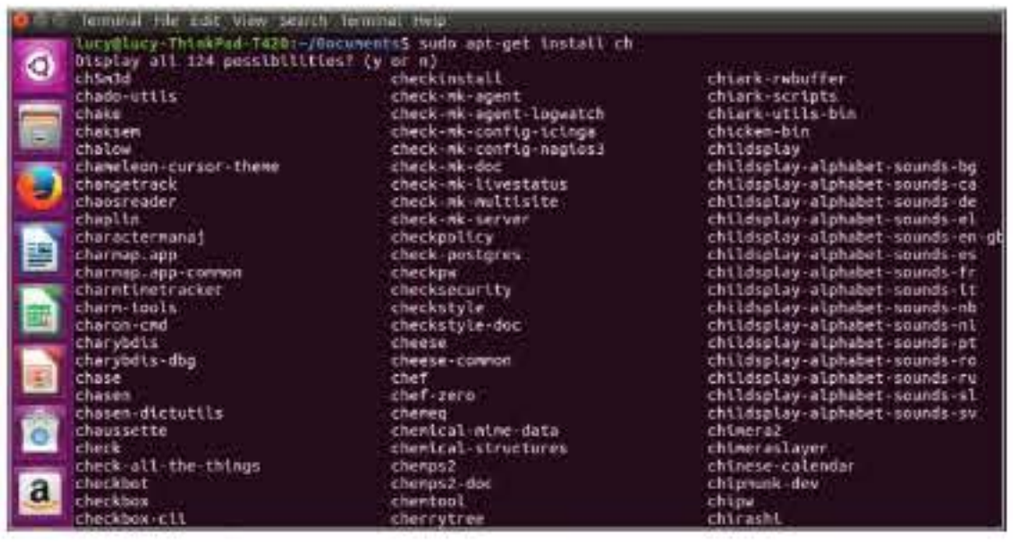
STEP 2 Sometimes there will be more than one option for tab completion. Enter `apt` and press Tab. Notice that nothing happens. Press Tab a second time and you'll see a list of commands that begin with "apt". Add `-g` to your command and press Tab again to get apt-get on the command line.



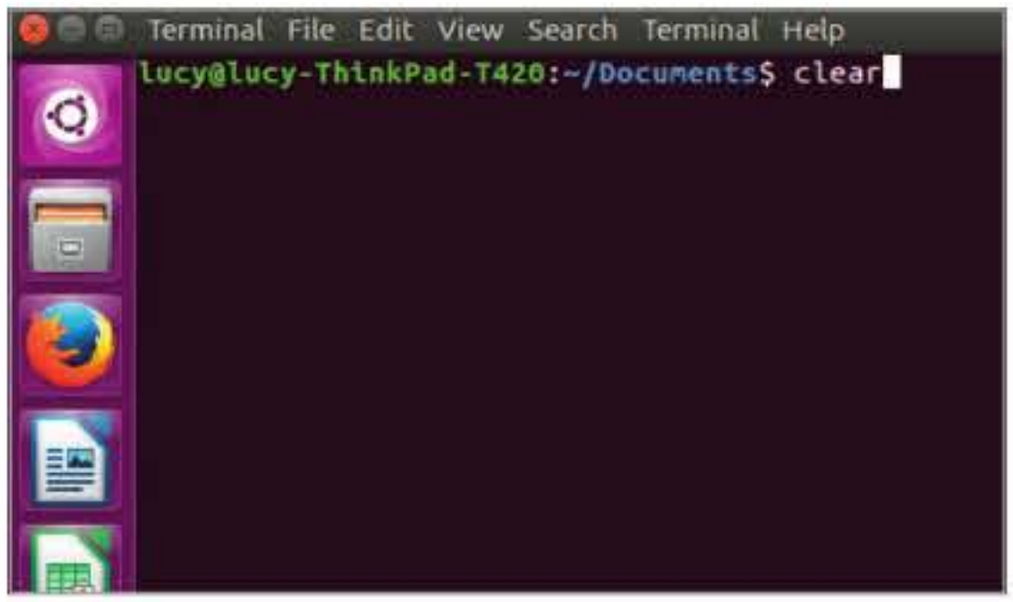
STEP 3 Tab completion is especially useful when typing long file names, directories and paths. Rather than typing the whole name you can enter the first few letters and press Tab. We've used touch to create a file called "really_long_filename-12345" in our Documents directory. Enter `rm rea` and press Tab to complete the file. Press Enter to remove it.



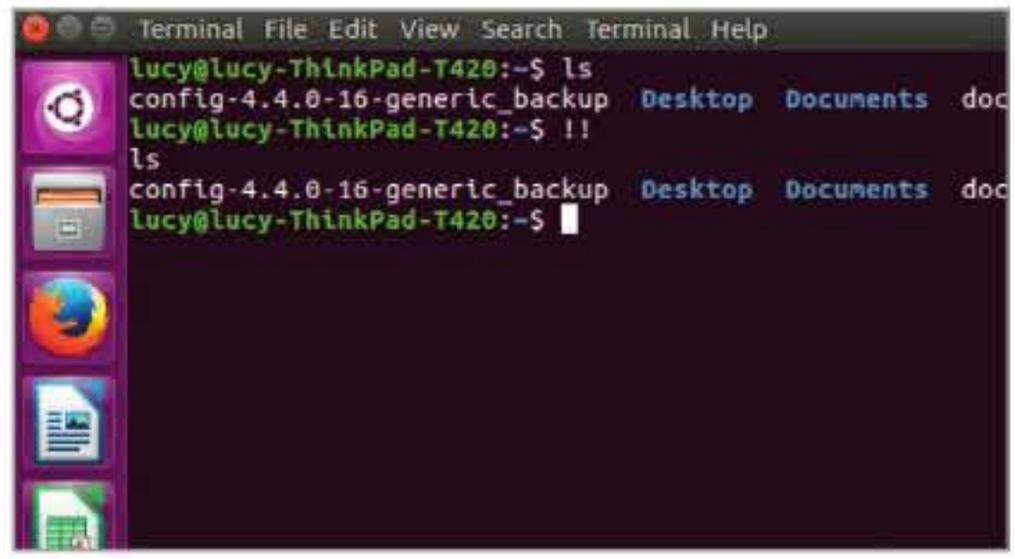
STEP 4 Tab can also be used to locate options and arguments in commands. Enter `sudo apt-get and i` then press Tab to expand the "i" to "install". Now enter `ch` and press Tab twice. Press y to list all possibilities and fill out "ch" to "chromium-browser". Press Enter if you want to install Chrome on your Unix machine.



STEP 5 Another tip to speed up Unix is to use the arrow keys to locate previous commands. Enter `clear` to clear the screen. Press Up to view clear again. Press up again to view the "sudo apt-get" command and so on. Press Enter to run the command again.



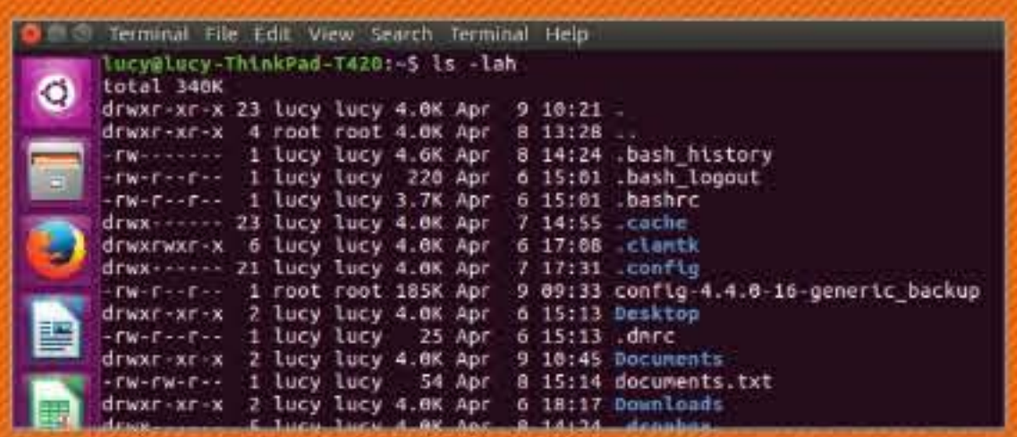
STEP 6 Enter `history` to view all the commands you've entered in Unix command line. Enter `!` followed by the number of a command to run it. You can also enter `!!` to run the previous command again. Enter `ls` and `!!` to run list twice. You can also combine this with sudo. Enter `sudo !!` to run the list command with root access.



CONFIGURING .BASHRC

One file you really need to get to know is .bashrc. This hidden file in your home folder contains the configuration files for bash, the terminal environment you are using. Editing .bashrc customises your command line.

STEP 1 You can find the .bashrc file using `ls -lah` and edit it using nano. Like the config.txt file it's best to start by making a backup. Enter `cp .bashrc .bashrc_backup` then `nano .bashrc` you don't need sudo as it's in your home folder. Like the config.txt file you'll see a lot of options that are commented out with the hash symbol.



STEP 2 Scroll down to find the section marked "# some more aliases". Here you'll find shortcuts for common commands "ll" for "ls -lah". Enter `alias c='clear'` below and use Ctrl-x to exit Nano and save the file.

STEP 3 Back in Terminal enter `c` and it will say "c: command not found". The alias won't work until you restart the Bash shell. You can close Terminal and reopen it or just enter `source ~/.bashrc` to restart the Bash shell. Now enter `c` at the command line and you'll clear the screen.

STEP 4 There's a huge amount of fun you can have editing the .bashrc file and you can enter any number of aliases. Take a look at cyberciti.biz/tips/bash-aliases-mac-centos-Unix-unix.html to view some popular aliases and serverfault.com/questions/3743/what-useful-things-can-one-add-to-ones-bashrc to find a list of popular tricks people have found for .bashrc.

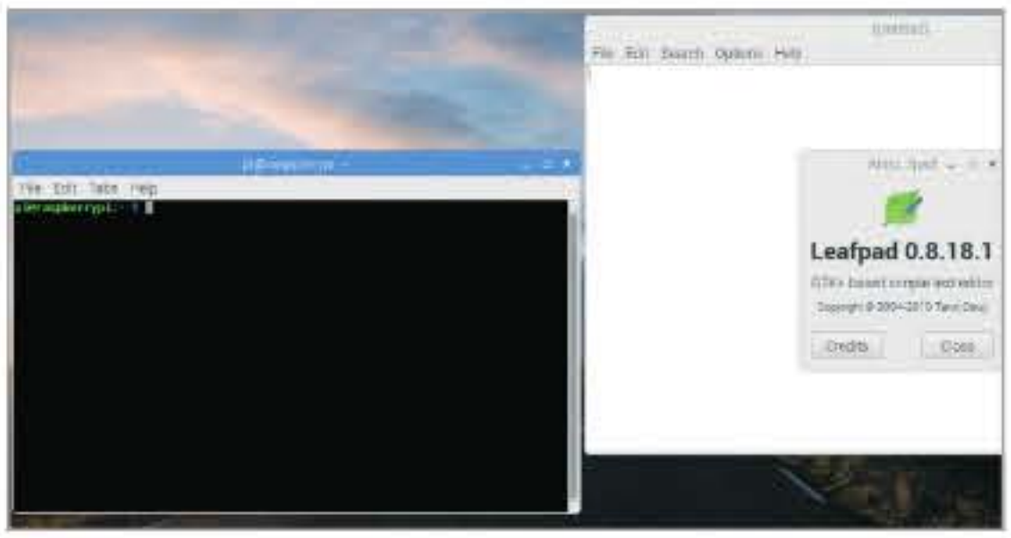
Using Python Code in Linux Scripts

Creating a Bash Shell script in Linux is a great way of executing your Python code. Bash scripts can be used to automate tasks, run programs and even ask for user input. Here's how to make them work for you and your Python code.

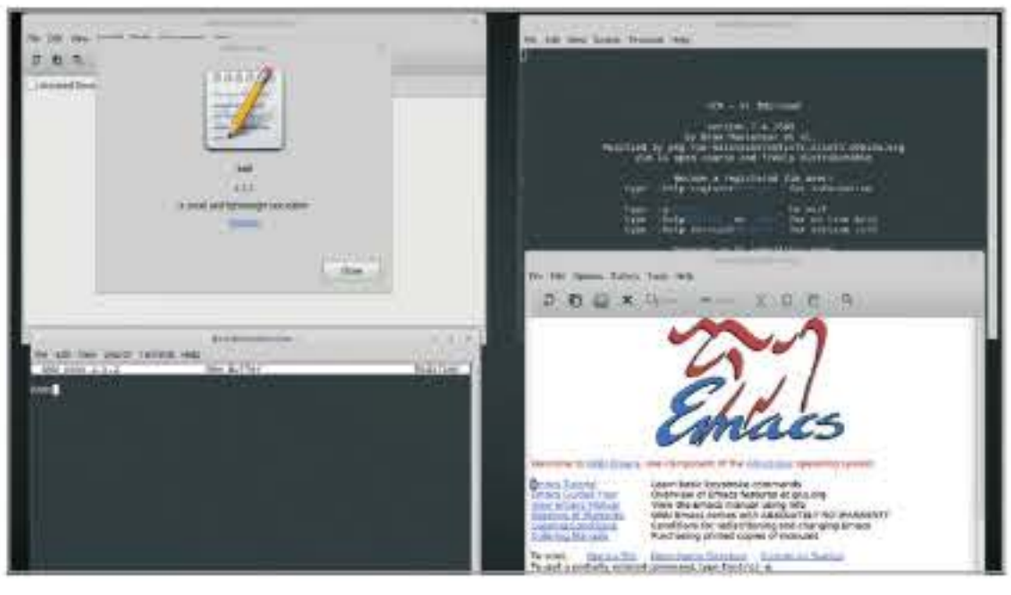
GET SCRIPTING

A Bash script is simply a series of commands that Linux will run through to complete a certain task. They can be simple, or remarkably complex, and they can call Python programs too.

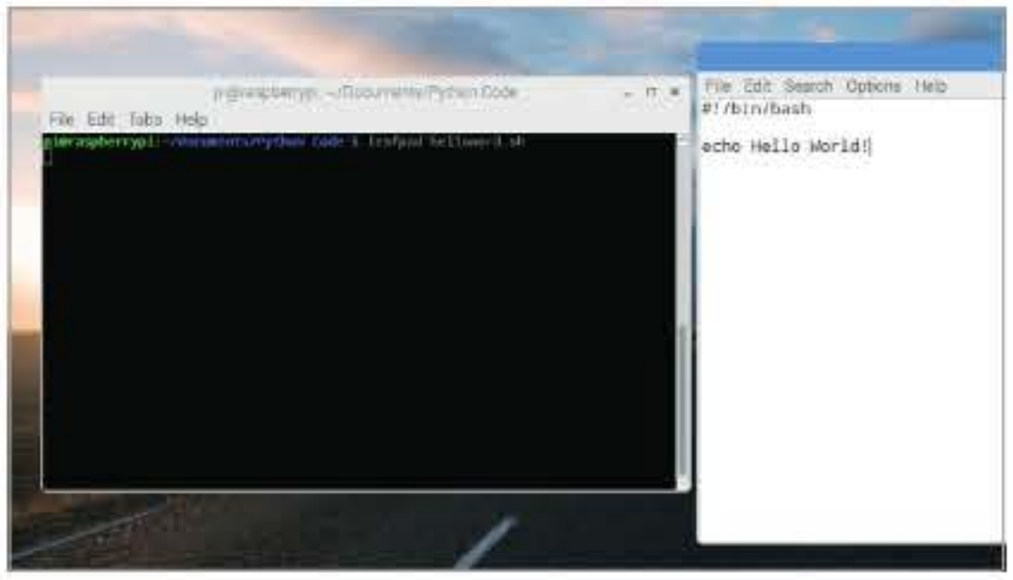
STEP 1 You'll be working within the Terminal and with a text editor throughout the coming pages. There are alternatives text editors you can work with, which we'll look at in a moment. It depends on the version of Linux you're using but for the sake of ease, we'll be doing our examples in Leafpad on a Raspberry Pi.



STEP 2 There are several text editors you can use to create a Bash script: Xed, Vi, Nano, Vim, GNU Emacs and so on. In the end it all comes down to personal preference and the version of Linux you're using. Our use of Leafpad is purely due to being the default text editor for the Raspberry Pi.

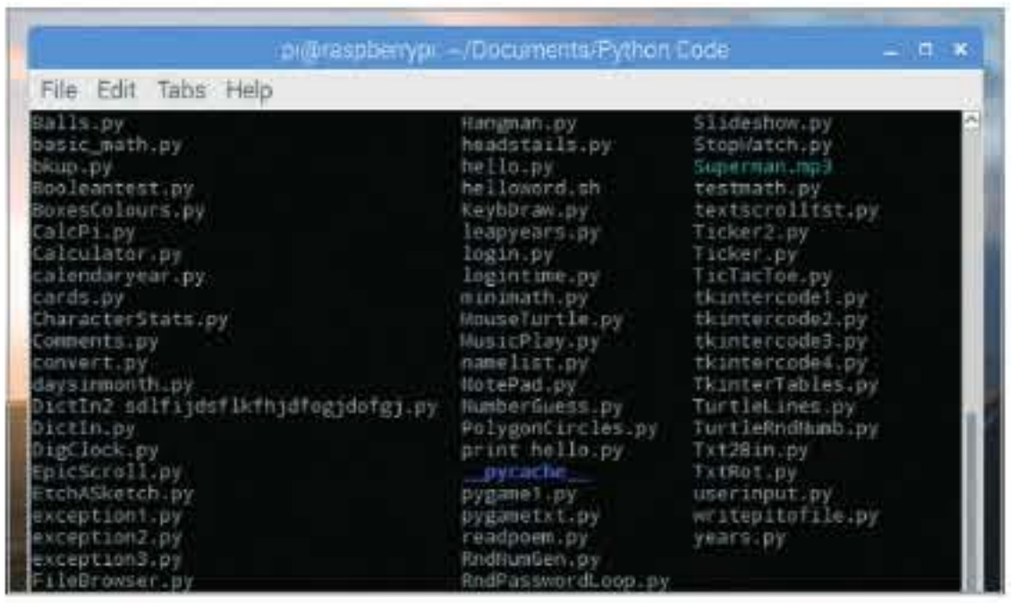


STEP 3 Let's start with a simple script to output something to the Terminal. Enter: `leafpad helloworld.sh`. This will launch Leafpad and create a file called helloworld.sh. In Leafpad, enter the following: `#!/bin/bash`, then on a new line: `echo Hello World!`.

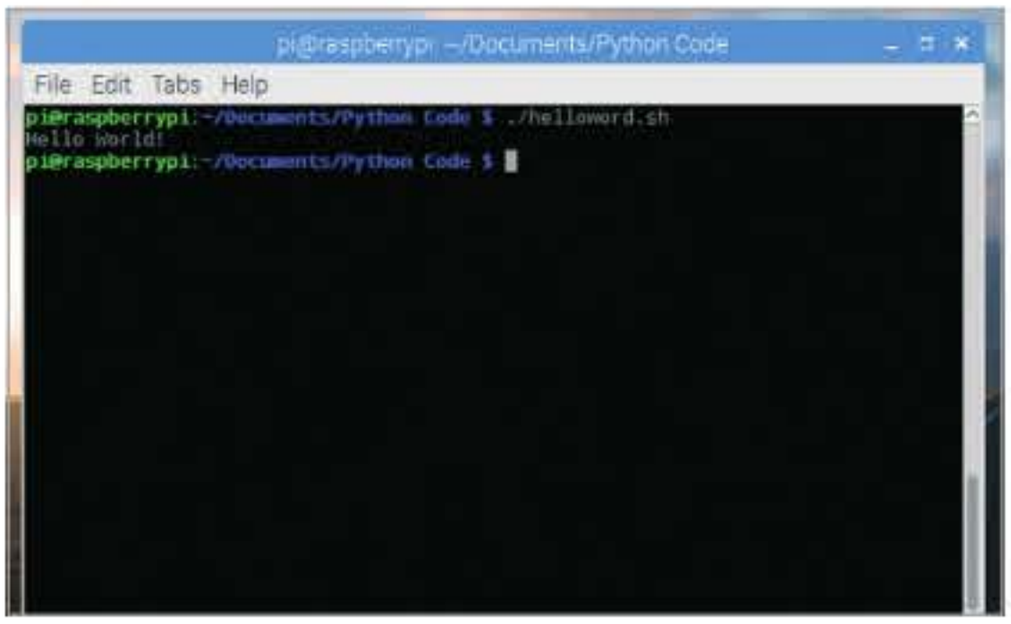


STEP 4 The `#!/bin/bash` line tells the system what Shell you're going to be using, in this case Bash. The hash, #, denotes a comment line, one that is ignored by the system and the exclamation mark, !, means that the comment is bypassed and will force the script to execute the line as a command. This is also known as a Hash-Bang.

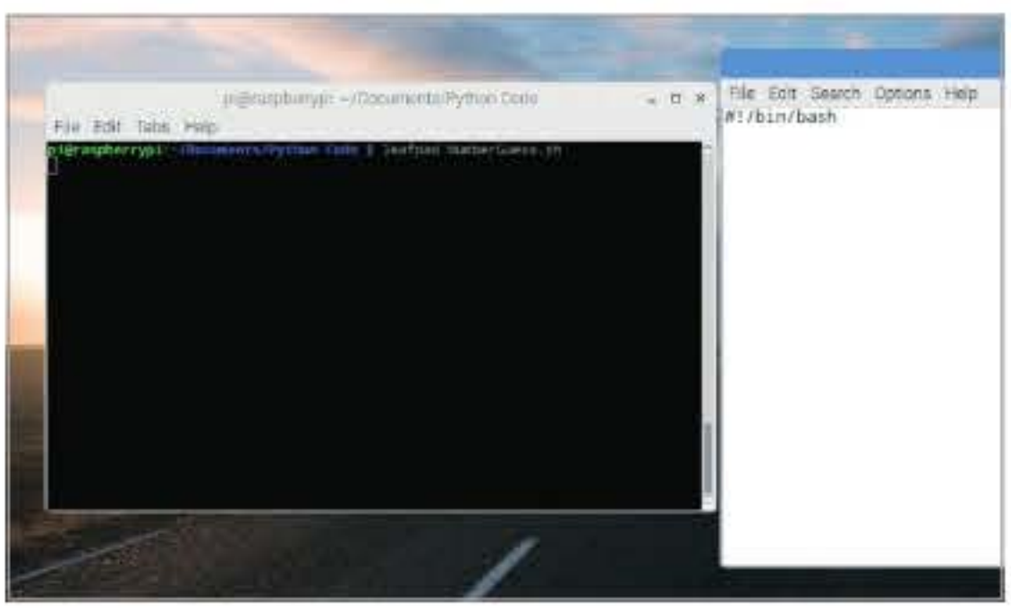
STEP 5 You can save this file, clicking File > Save and exit back to the Terminal. Entering ls, will reveal the script in the folder. To make any script executable, and able to run, you need to modify its permissions. Do this with: `chmod +x helloworld.sh`. You need to do this with every script you create.



STEP 6 When you enter ls again, you can see that the helloworld.sh script has now turned from being white to green, meaning that it's now an executable file. To run the script, in other words make it do the things you've typed into it, enter: `./helloworld.sh`.

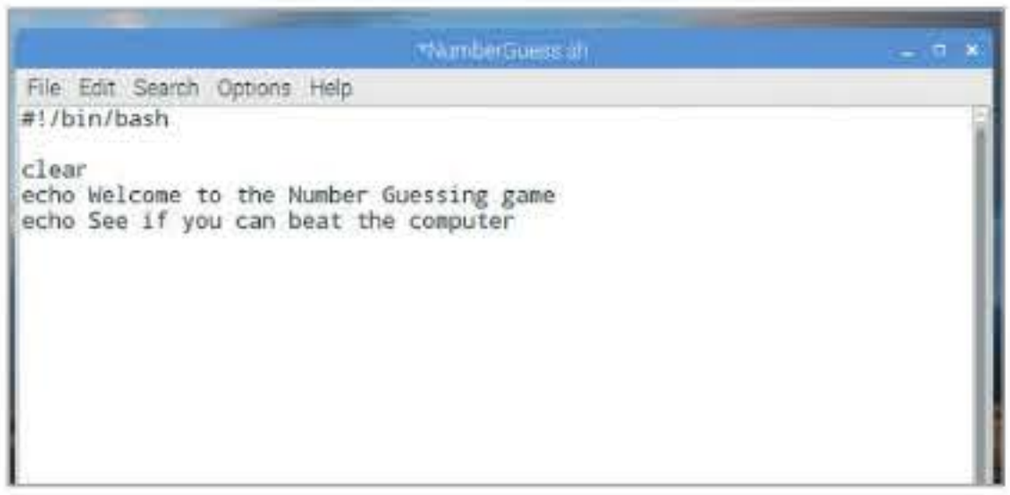


STEP 7 It's not terribly exciting but the words **'Hello World!'** should now be displayed in the Terminal. Now let's see about getting one of our Python programs to work. Create a new Bash script called NumberGuess.sh and start it off in Leafpad with: `#!/bin/bash`



STEP 8 You're going to be using the Number Guessing game script from the earlier Code Repository section. Begin by clearing the screen and entering a few echo commands to let the user know what's going on:

```
clear
echo Welcome to the Number Guessing game
echo See if you can beat the computer
```

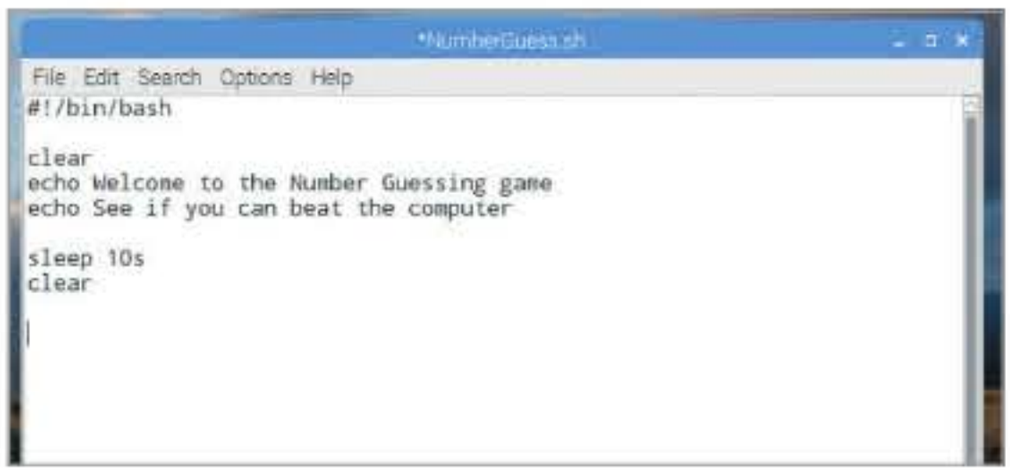


STEP 9 You can now use the Sleep command to pause the introduction screen for ten seconds:

```
sleep 10s
```

If you like you can add another clear after the sleep command, so the screen will be empty ready for the Python script to be executed.

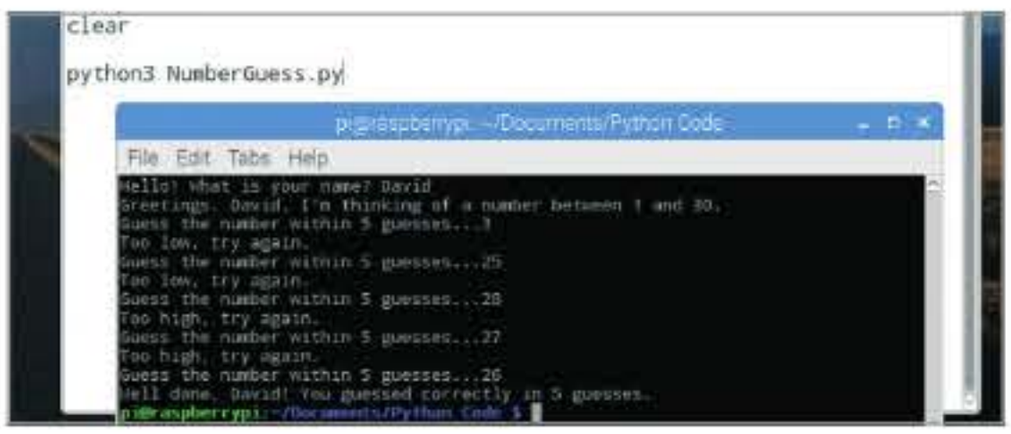
```
sleep 10s
clear
```



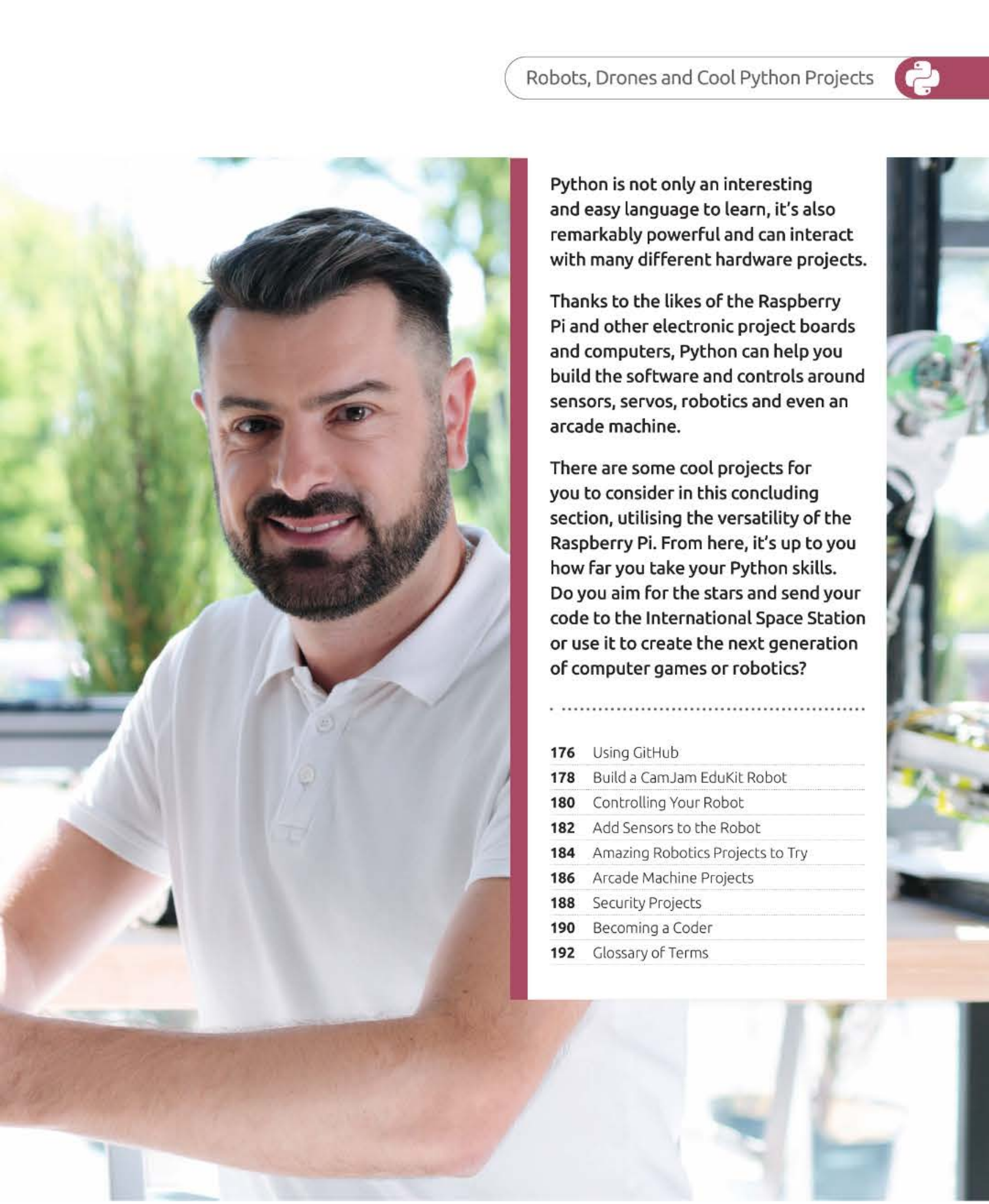
STEP 10 Finally, you need to execute the Python program. Do this with:

```
python3 NumberGuess.py
```

This will execute the Python code you created; remember to make sure the Bash script is in the same directory as the Python program. Make script executable (**chmod +x NumberGuess.sh**) and run it with: **./NumberGuess**. The script will run through the lines of code, then execute the Python program.



Robots, Drones and Cool Python Projects



Python is not only an interesting and easy language to learn, it's also remarkably powerful and can interact with many different hardware projects.

Thanks to the likes of the Raspberry Pi and other electronic project boards and computers, Python can help you build the software and controls around sensors, servos, robotics and even an arcade machine.

There are some cool projects for you to consider in this concluding section, utilising the versatility of the Raspberry Pi. From here, it's up to you how far you take your Python skills. Do you aim for the stars and send your code to the International Space Station or use it to create the next generation of computer games or robotics?

176 Using GitHub
178 Build a CamJam EduKit Robot
180 Controlling Your Robot
182 Add Sensors to the Robot
184 Amazing Robotics Projects to Try
186 Arcade Machine Projects
188 Security Projects
190 Becoming a Coder
192 Glossary of Terms

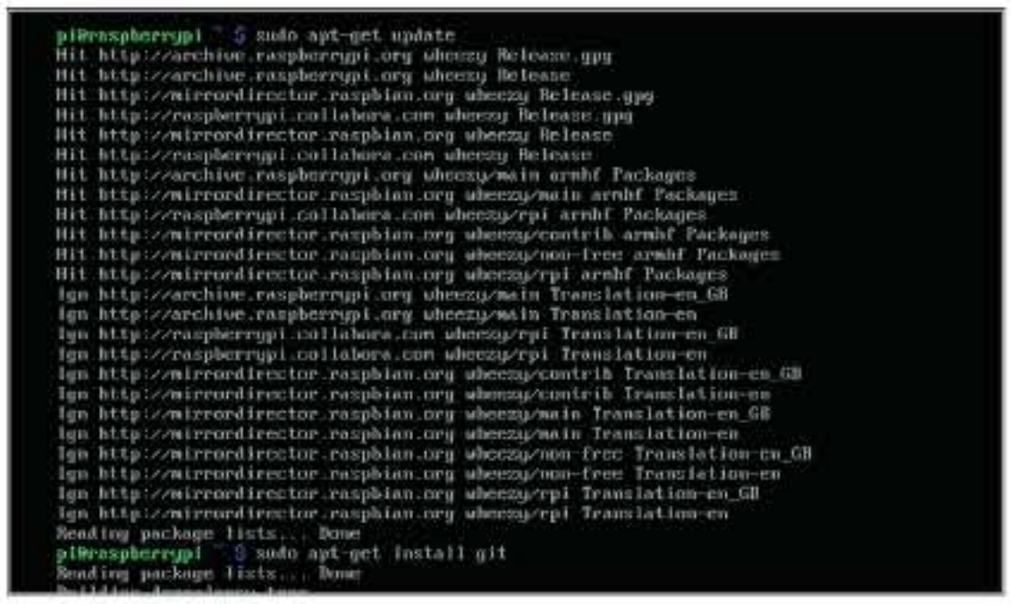
Using GitHub

As you start to create advanced projects in Python, you'll need more advanced code. You could type it in by hand and often you'll find code on developer's websites but increasingly it's being housed on GitHub.

GETTING TO KNOW GIT

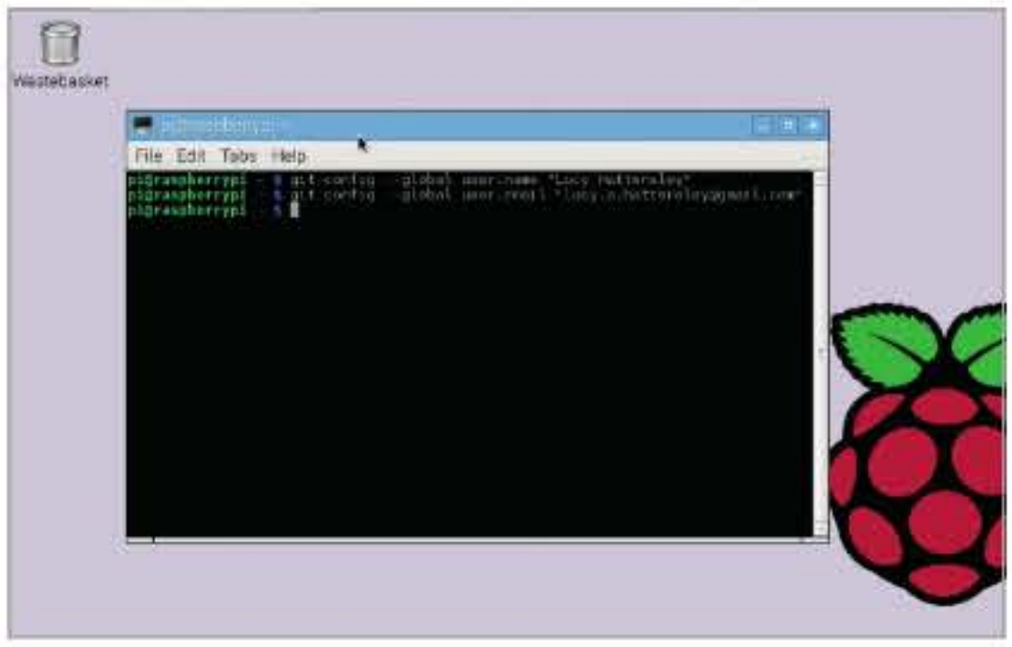
Git is an online versioning and collaboration portal that enables you to store code online, download code, make changes and then integrate those changes. Github (github.com) is a website that developers use to host links to the code.

STEP 1 Git enables you to clone code created by other developers on your laptop or Raspberry Pi. So it's a vital tool to have on all your systems. You install Git in Linux using APT. First enter `sudo apt-get upgrade` to get the latest package lists and sudo apt-get install git to install the latest version of Git. Follow the instructions at github.com for macOS and Windows systems.

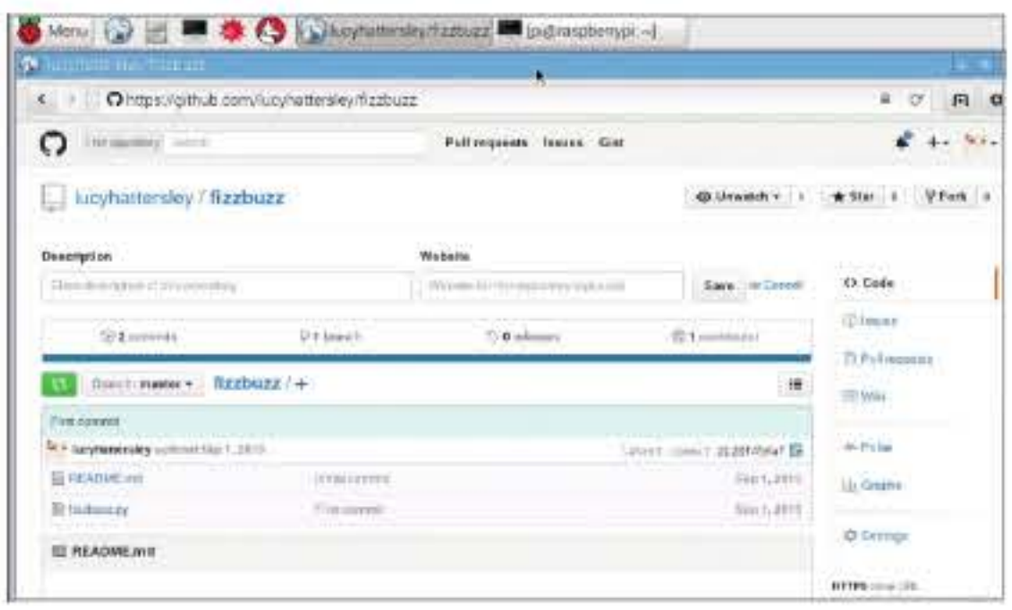


STEP 2 Enter `git --version` to make sure it's installed correctly. You should have the latest version installed (ours is 1.7.10.4). You can enter `git -h` to get quick help or man git to read the manual but Git is pretty complex at first so it's better to follow our guide. You're going to use the desktop to take a look at Github next so enter `startx`.

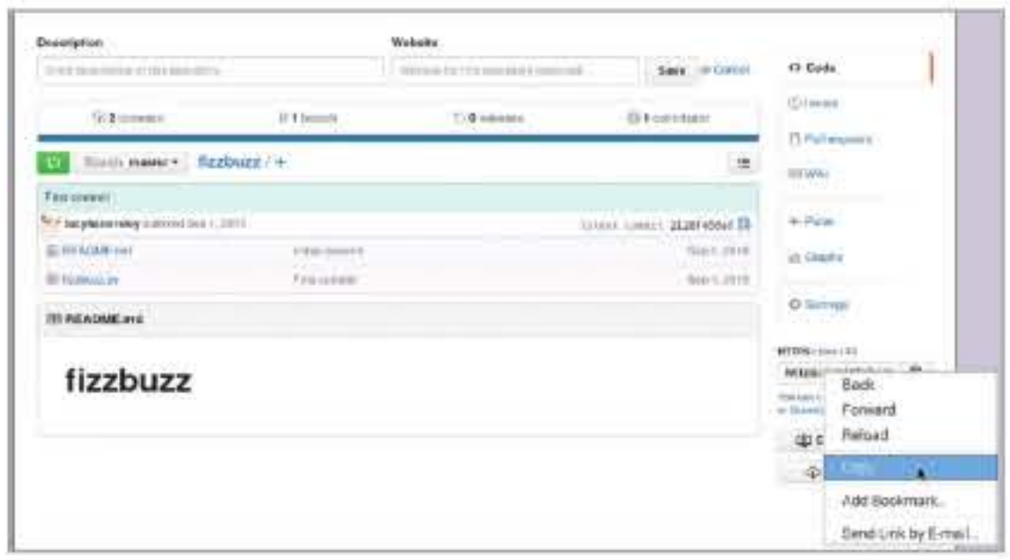
STEP 3 Open a web browser and enter github.com. Click Sign Up and create an account. Now open a terminal window and enter `git config --global user.name "John Doe"` (replacing John Doe with your name). Next enter `git config --global user.email john.doe@mail.com` replacing the email address with the same one you used to sign up to Github.



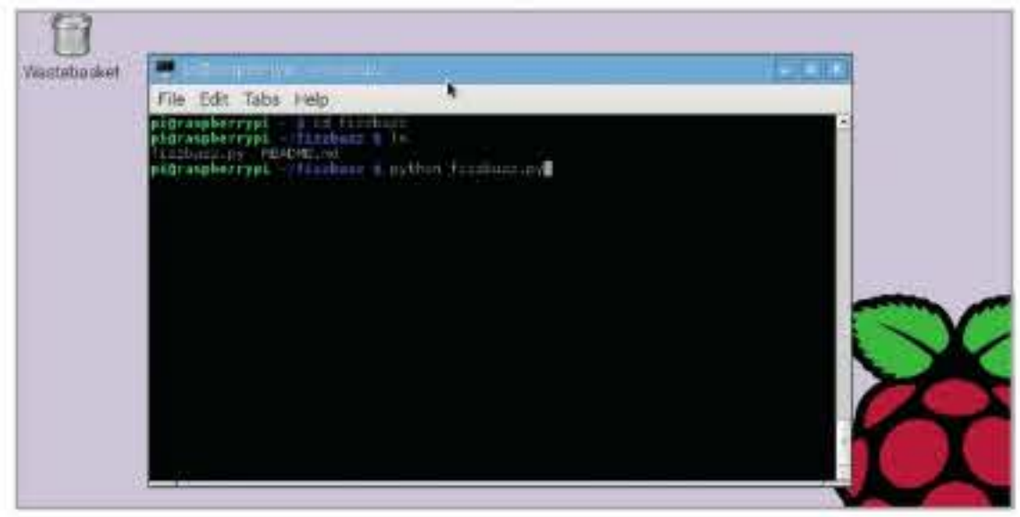
STEP 4 Now we're going to clone a project from Github to our computer. In the web browser enter `github.com/lucyhattersley/fizzbuzz`. This is a repository I've created on Git that contains a program called FizzBuzz. A repository is like an online folder that contains the code. You will see links to the files contained in the project: README.md and fizzbuzz.py but don't click on these.



STEP 5 In the bottom right of the Github window you will see a text box marked HTTPS clone URL and inside it there is a URL. Click the text and press Control-A to select it all. Now right-click and choose Copy. Now open a terminal window and enter `git clone` and choose Edit > Paste to paste the HTTP address after the command.



STEP 6 Enter `ls` and you will see the Fizzbuzz directory in your home directory. Enter `cd fizzbuzz` to switch that directory and ls again to view the fizzbuzz.py and README.md files. FizzBuzz is a classic program that counts to 100 but on numbers devisable by 5 or 3 it prints "Fizz" or "Buzz" instead (or FizzBuzz if it's devisable by both 5 and 3). Enter `python fizzbuzz.py` to run the code.



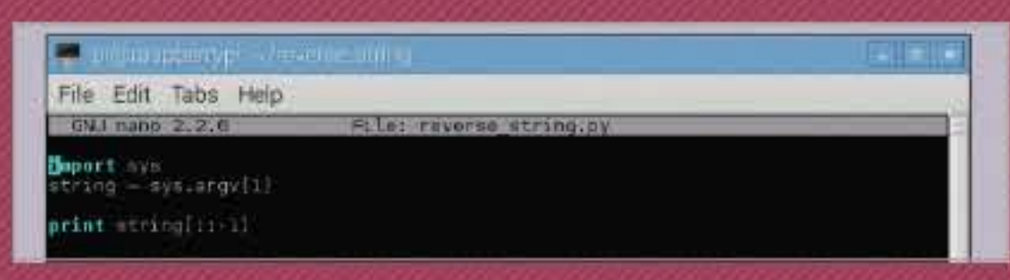
ADDING CODE TO GIT

Now that you know how to get clone code from Github to your Raspberry Pi, you might be wondering how to put your own code into Git. Git is a powerful and fairly complex piece of software but this will help you get started.

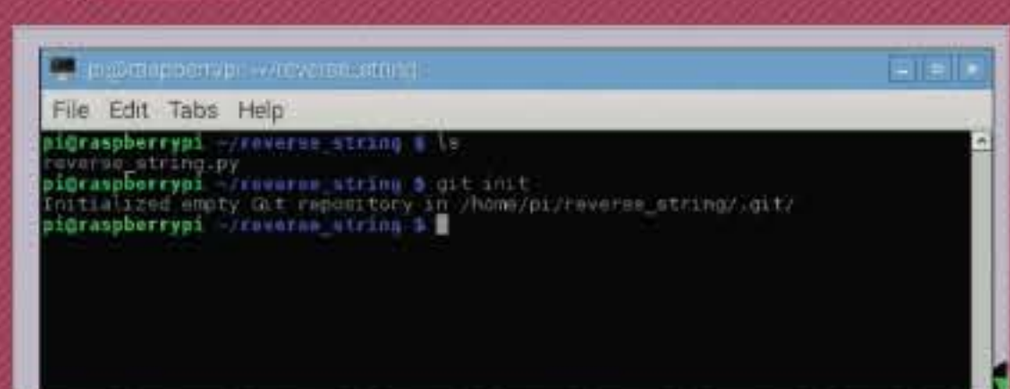
STEP 1 You're going to create a Python program that reverses a string and upload it to Git. Enter mkdir reverse_string and cd reverse_string. Then nano reverse_string.py and enter this code:

```
import sys
string = sys.argv[1]
print string[::-1]
```

Press Control-O and then Enter to save the code. Press Control-X to exit.

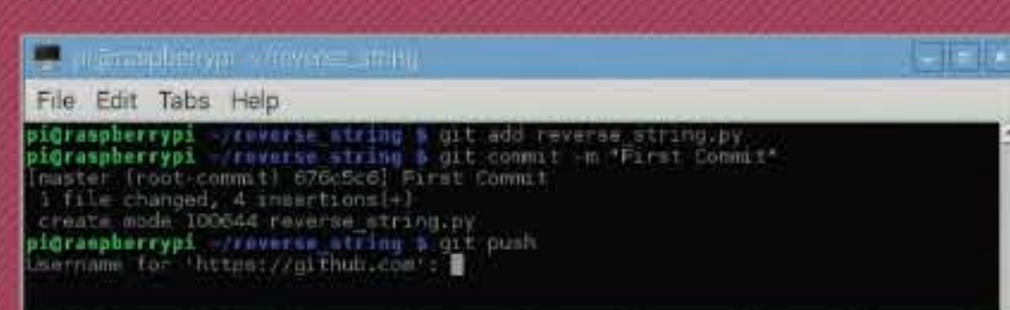


STEP 2 You can test the program by entering `python reverse_string "Hello World"` You should see dlorW olleH returned on the command line. You're going to put this program on your Github but first it needs to be a Git folder, enter git init. This initialises the directory as a git repository. It creates a hidden file called .git to the folder. You can see this using `ls -lah`.



STEP 3 First we need a repository. Open Github (github.com) and click the green New Repository button. Enter reverse string in the Repository name field and select the Initialise this repository with a README option. Click Create repository to make the file. Next you need to copy the text from the HTTP's clone URL field as you did when cloning a website. Return to the command line and enter `git remote add` and use Edit > Paste to paste in the URL.

STEP 4 You've now linked the local directory to the Git repository and can sync changes from one place to another. Enter `git add reverse_string.py` to track the python file. Now enter `git commit -m "First commit"` to tell Git you want to send "commit" the file to your Git space. Finally enter `git push` and enter your username and password to send the files. You'll now see them on github.com. This is only a really rough introduction to Git. Head over to codeschool.com/courses/try-git to find an online course.



Build a CamJam EduKit Robot

The CamJam EduKit 3 is a fantastic robot kit that enables you to build a fully functioning robot with ultrasonic and line following sensors for just £17. All you need is a chassis and we'll show you how to turn an old tin box into an amazing robot.

CAMJAM

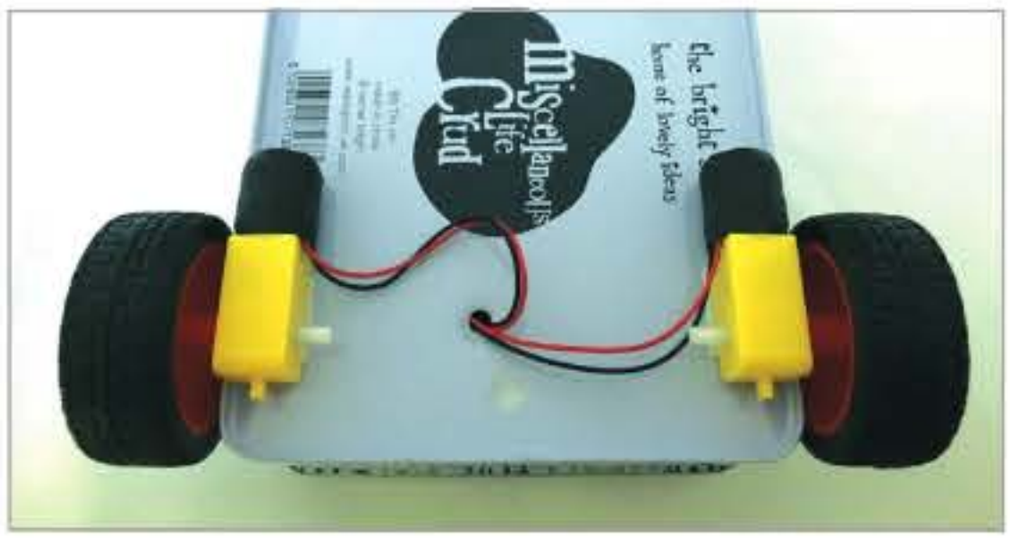
The Cambridge Raspberry Jam (camjam.me) teaches children of all ages how to build electronics projects and the kits are sold to everyone via the Pi Hut. The CamJam EduKit #3 has everything you need to build a robot (thepihut.com/collections/camjam-edukit).

STEP 1 The CamJam EduKit 3 includes wheels, motors, a battery box, an EduKit Controller Board and two sensors. You also get a breadboard, wires and resistors to bring it all together, along with some double-sided tape. We suggest you get a USB battery charger, the type used to recharge phones and a Wi-Fi dongle.

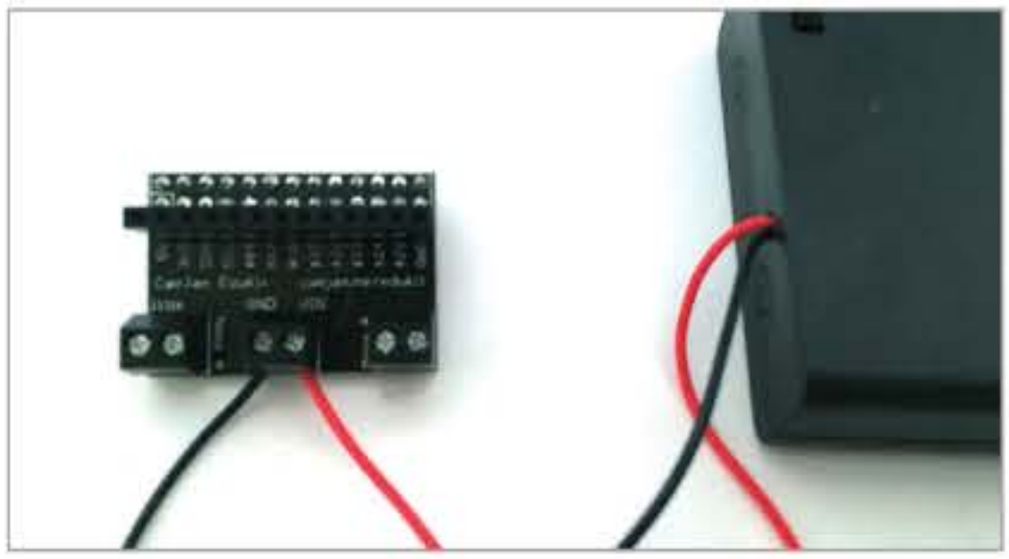
STEP 2 You'll need a chassis for your robot. We're using a tin box, the type you often get to keep knick-knacks in. You can use any flat surface as a chassis, so long as you can mount the wheels but using a container makes it easier to hold all the bits inside. You could also use a plastic Tupperware box or even the box that the kit comes in.

STEP 3 Flip the box upside down and make a hole in it in the middle, at one end. We're using a metal drill to drill a hole through the tin box but if your box is made from plastic or card you could cut a hole using scissors. It doesn't need to be large, just big enough to fit the wires from the servo motors.

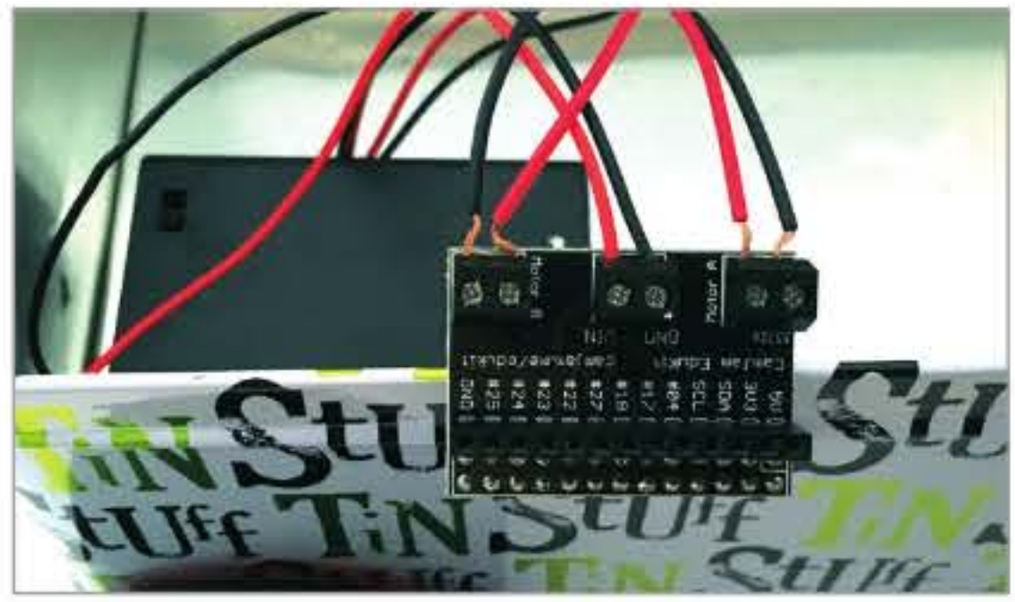
STEP 4 Attach the two plastic wheels to the yellow servo motors making sure they're on opposite sides. Now fix the servos to the underside of the box, making sure they're near the hole you created. You can use the double-sided tape to mount the wheels but it's a good idea to use Blu-Tack first to make sure the design works.



STEP 5 Place four AA batteries inside the battery box and make sure it is set to Off. Use a small screwdriver to loosen the two central screws in the EduKit Controller Board. Place the black wire from the battery box in the GND and the red wire in VIN (Voltage). Take a look at Worksheet 1 (camjam.me/?page_id=1035#worksheets) for more info.



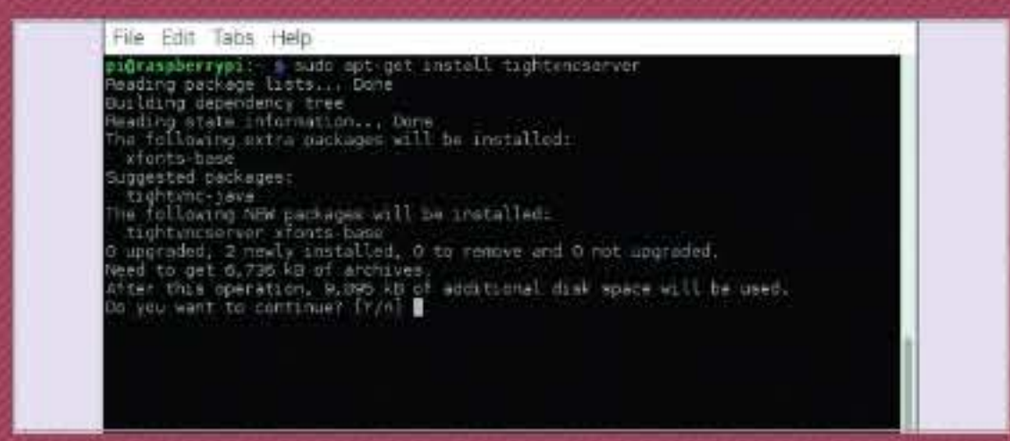
STEP 6 Place the robot front away from you and attach the red and black wires from the right wheel to the block marked Motor A. It doesn't matter which way round. Take the wires from the left wheel and attach them to Motor B with the wires the opposite way round from Motor A (see picture).



SETTING UP THE RASPBERRY PI

The Raspberry Pi will need to operate on the move, so it needs power and a wireless network connection. You'll also want to access it remotely, so you'll want to set up VCN. Let's build the brains.

STEP 1 Install a fresh build of Raspbian Jessie on an SD Card. Use `sudo apt-get update` and `sudo apt-get upgrade` to get the latest builds for all your software. Now use `sudo apt-get install tightvncserver` to install VNC software. Enter `sudo nano .bashrc` and add `vncserver` to the end of your bash file so it starts at boot time.



STEP 2 Connect a Wi-Fi dongle to your Raspberry Pi and click on the Wi-Fi icon in the top right of the Xcfc desktop interface; enter `startx` if you're not in the desktop environment. Choose your network and enter the Wi-Fi password. Open the terminal and enter `ifconfig`. Make a note of your network address.

STEP 3 Charge up a portable USB charger, the type normally use for mobile phones and make sure it's got plenty of juice. Shut down your Raspberry Pi and plug in the portable charger and make sure it can boot and run your Raspberry Pi.

STEP 4 Check that the VNC connection is working from another computer. On a Mac press Control-K and enter the vnc://192.168.0.[x]:5901 (with x the number from ifconfig in Step 2). Click Connect and you should see the Xfce desktop. Your Raspberry Pi is now ready to work remotely inside the robot.

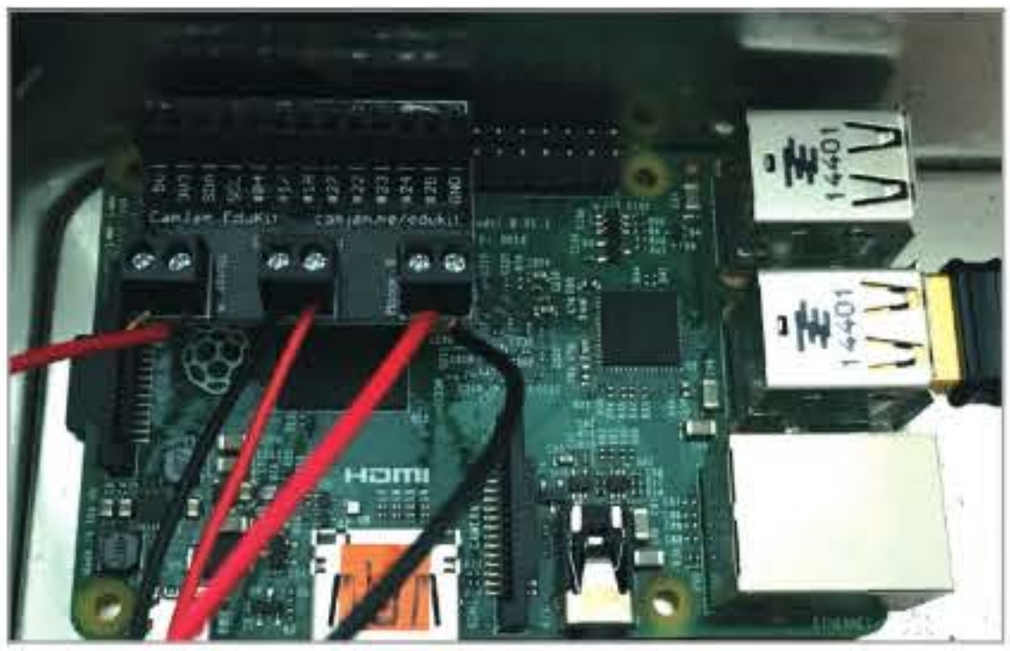
Controlling Your Robot

With all the motor parts in place, it's now time to hook everything up and start testing out your robot. Once the motor control unit is attached to the Raspberry Pi and it is all powered up, you're ready to start testing its movement in code.

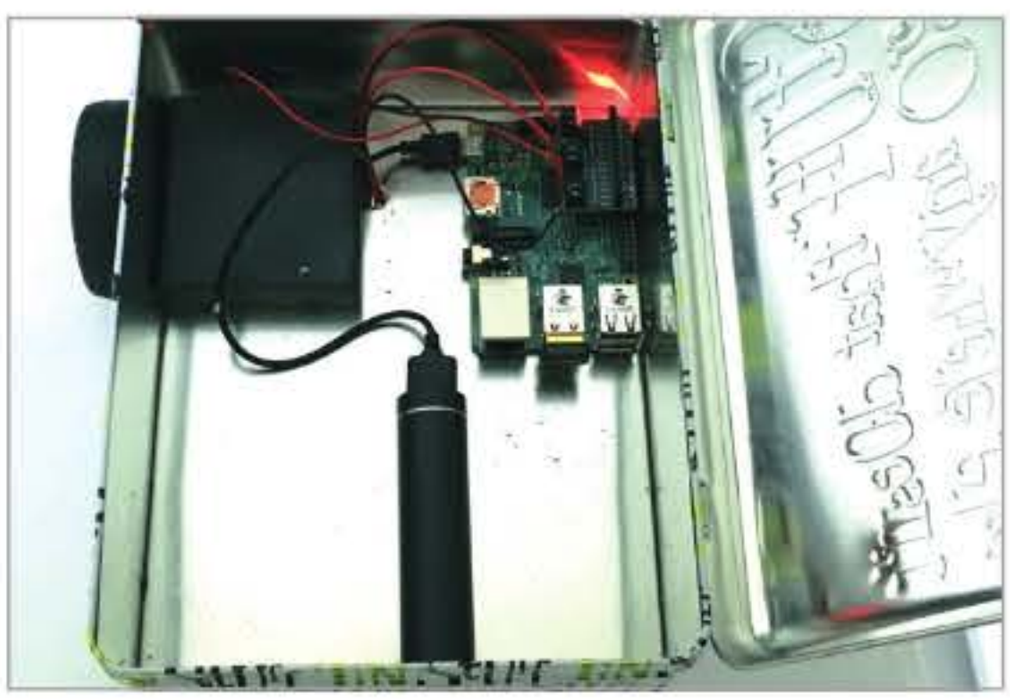
READY TO ROLL

Our robot still has a way to go to be complete but with the servos and wheels attached to the motor control unit and the power in place, it's ready to start rolling around.

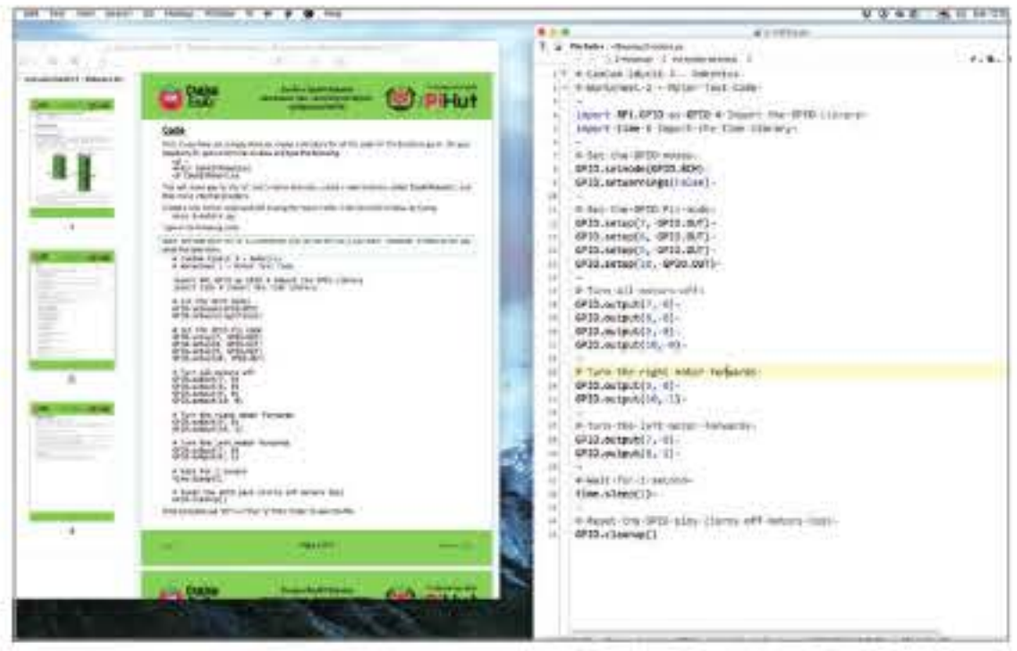
STEP 1 You should have all six wires attached to the EduKit Controller Board unit. Double check that the black wire from the battery is going to GND and the red wire to VIN and that the black and red wires to Motor A and Motor B are the mirror opposite way around (as shown in this photo). Gently push the motor control unit into the GPIO pins on the left side.



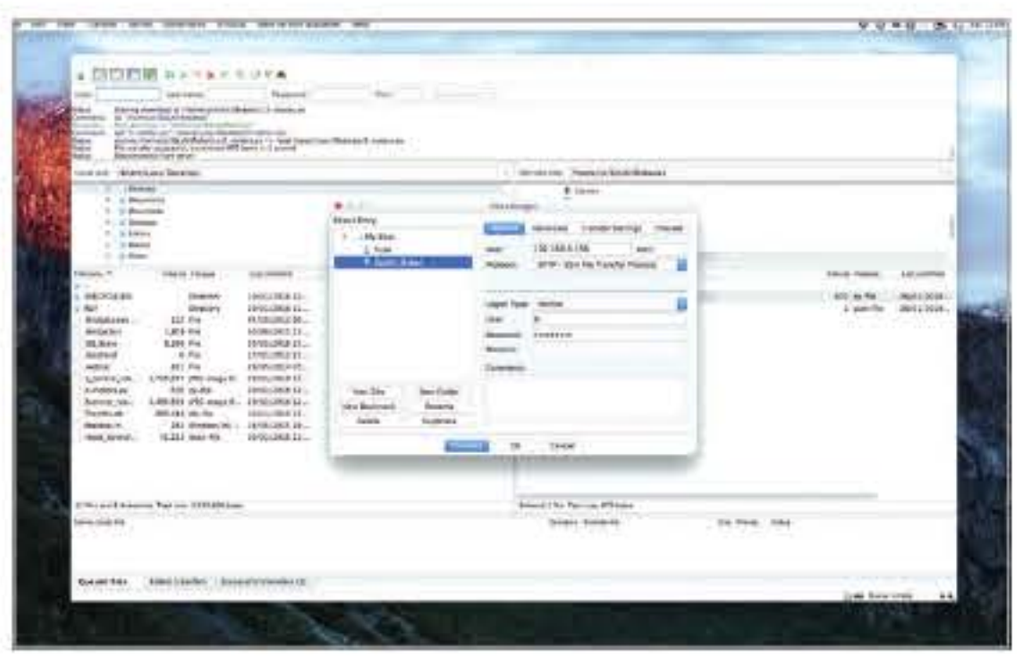
STEP 2 Now it's time to power up your robot and see if it moves. Attach the battery pack to the Raspberry Pi and switch on the battery box. Give the Raspberry Pi thirty seconds to go through the boot process and connect to it from a computer using VNC to make sure it is responding.



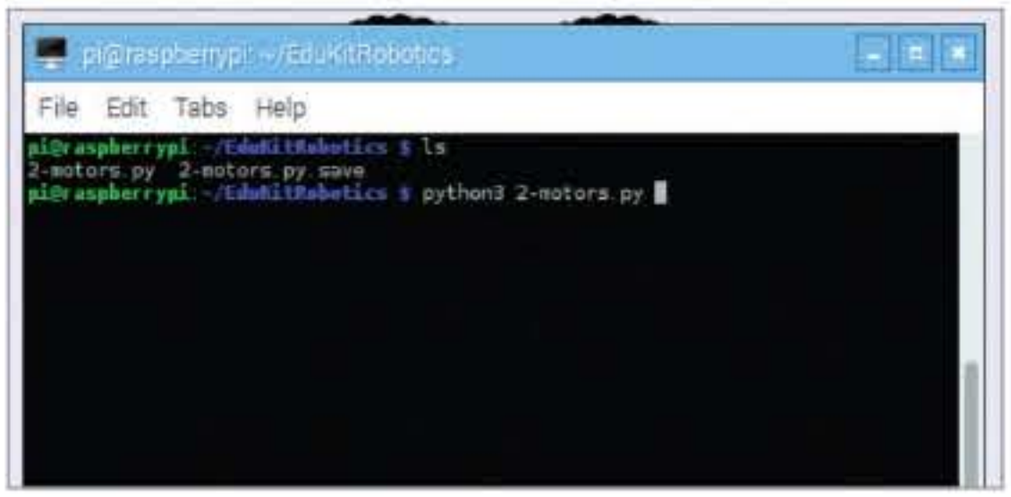
STEP 3 Open the CamJam website (camjam.me/?page_id=1035#worksheets) and download Worksheet 2 - Running the motors. Open a text editor on your computer and type out the code for the 2-motors.py program. You can cut and paste it to your text editor but we think it makes more sense to type it line by line so you understand what it's doing.



STEP 4 Open a FTP program such as FileZilla and choose File > Site Manager. Enter the IP address in the Host field, "pi" in the User field and your password ("raspberry" by default). Right-click on pi under Remote Site and choose Create Directory, name it "EduKitRobotics". Now transfer the 2-motors.py file to the EduKit directory on your robot.



STEP 5 Open the VNC program and connect to your Raspberry Pi. You should see the Xfce desktop even if you set up the robot to book into the command line. Enter cd EduKitRobotics. Make sure your robot is on the floor and has about a meter of space in front of it. Now enter `python3 2-motors.py` and press Return



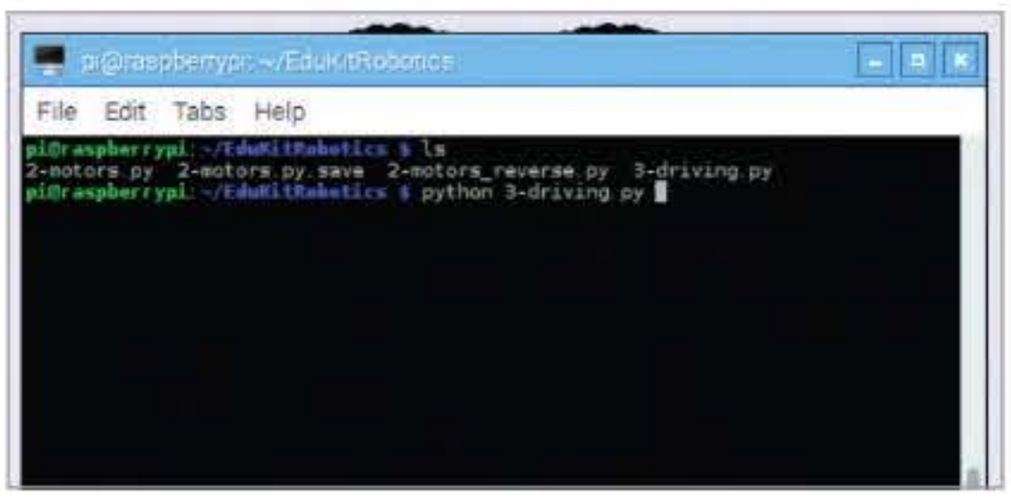
STEP 6 All being well your robot should move forwards. If it moves backwards you need to swap the position of the red and black wires on both Motor A and Motor B. If it fails to move then check the wires to the motor control unit, check that everything is switched on and try replacing the batteries. You need good quality batteries with enough power to move the servos.

DRIVING CODE

Now that our robot is up and moving, it's time to give it better control functionality. This code enables you to move the robot forward, backwards and rotate left and right. It also makes it easy to give the robot multiple directions in order.

STEP 1 Head back to the CamJam website (camjam.me/?page_id=1035#worksheets) and download Worksheet 3 - Driving and Turning. Type in the first block of code and save it as 3-driving.py in the EduKitRobotics folder. Make sure you enter the code from the section Moving the Robot at the end of your main block of code.

STEP 2 The worksheet advises using nano to create code directly on your Raspberry Pi robot but we think it makes more sense to type it out on your computer and transfer it using FTP. Use VNC to connect to your Raspberry Pi and run the program using `python3 3-driving.py`. The robot should move forwards then backwards.



STEP 3 Take a good look at the Python code to see how it creates function definitions for Forwards(), Backwards() and StopMotors(). These definitions enable you to move the robot using functions, rather than typing out the code each time. Now add the Left() and Right() functions from the worksheet to the code.

STEP 4 Add the movement instructions to the very end of the code and save the 3-driving.py program to your EduKitRobotics folder, using FTP to transfer it. Use VNC to run `python3 3-driving.py` on the Raspberry Pi. The robot should move forwards, rotate left, move forwards again, rotate right, move backwards and then stop.

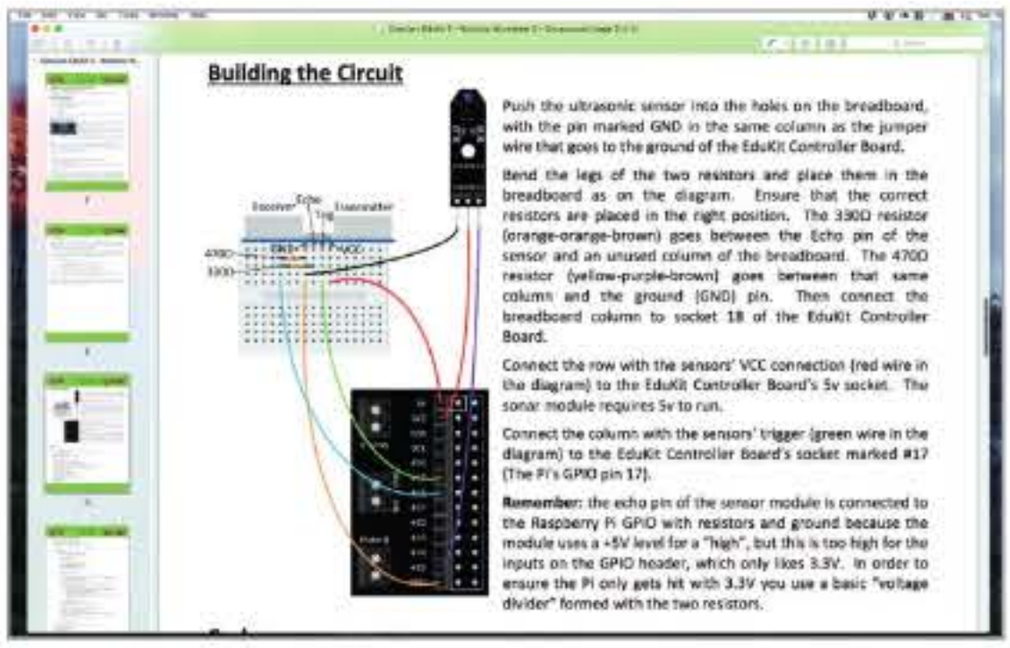
Add Sensors to the Robot

Now that your robot is built and moving around, it's time to give it some smarts. The two sensors can be used by the robot to react to its surroundings; either following a line or avoiding obstacles.

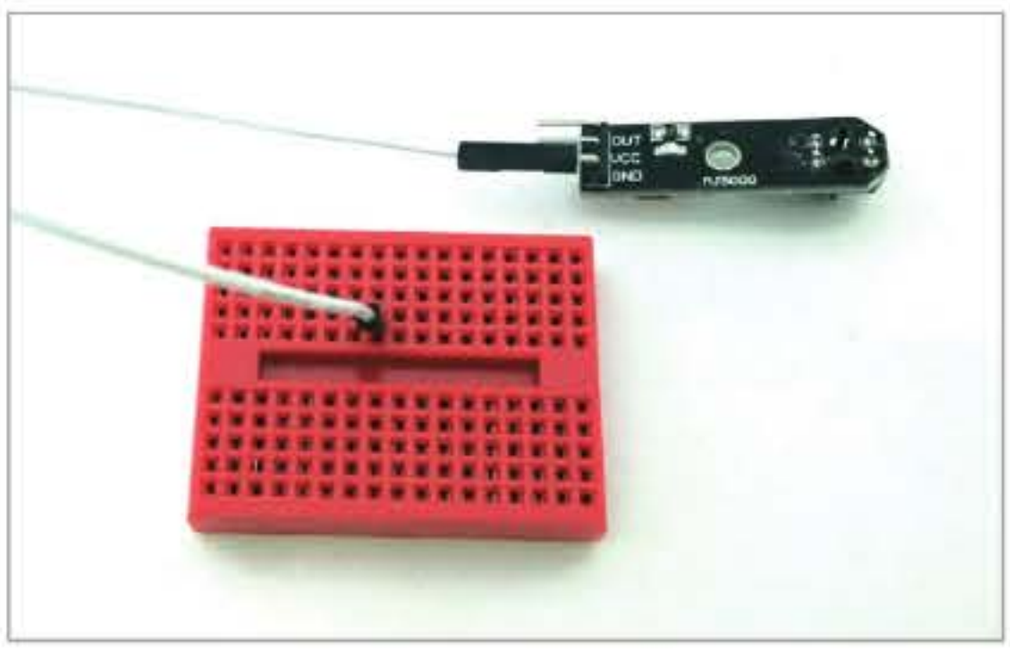
ULTRASONIC AND LINE SENSING

Your robot has two sensors, an Ultrasonic and a Line Detector. The Ultrasonic is the one that looks like eyes and fits into the breadboard. The Line Detector has a small blue camera on one side and fits underneath the robot.

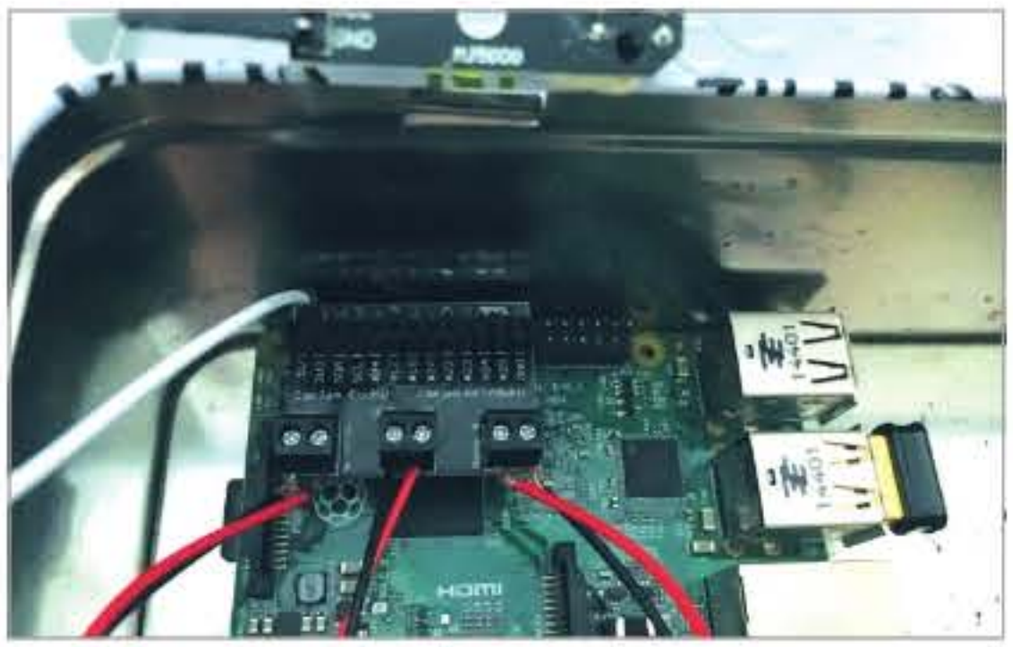
STEP 1 Open the CamJam website (camjam.me/?page_id=1035#worksheets) and download Worksheet 4 and Worksheet 5. Worksheet 4 is just for the Line Detector, while worksheet 5 is for both sensors. Make sure you take a good look at the Building the Circuit diagram in Worksheet 5 and keep it with you during this tutorial.



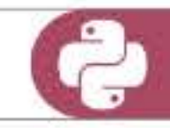
STEP 2 Make sure that the robot isn't currently powered. If it is use VNC and shutdown the Raspberry Pi and disconnect the power to the Raspberry Pi. Locate one of the female-male jumper wires and connect the female end to the GND pin on the Line Detector. Connect the other end to the breadboard in the eighth column, fourth row (as shown here).



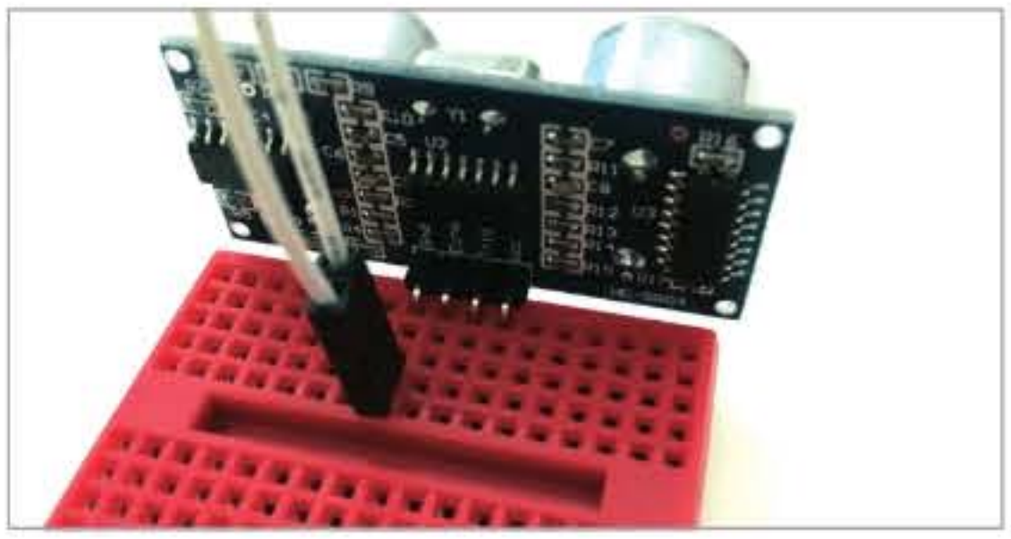
STEP 3 Use a regular male-male jumper cable and place it next to the jumper in the breadboard in the same column, fifth row. Connect the other end to the GND of the EduKit Controller Board. Now take another female-male jumper wire and connect it from the VCC pin on the Line Detector to the 3V3 (3.3 volts) pin on the EduKit Controller Board.



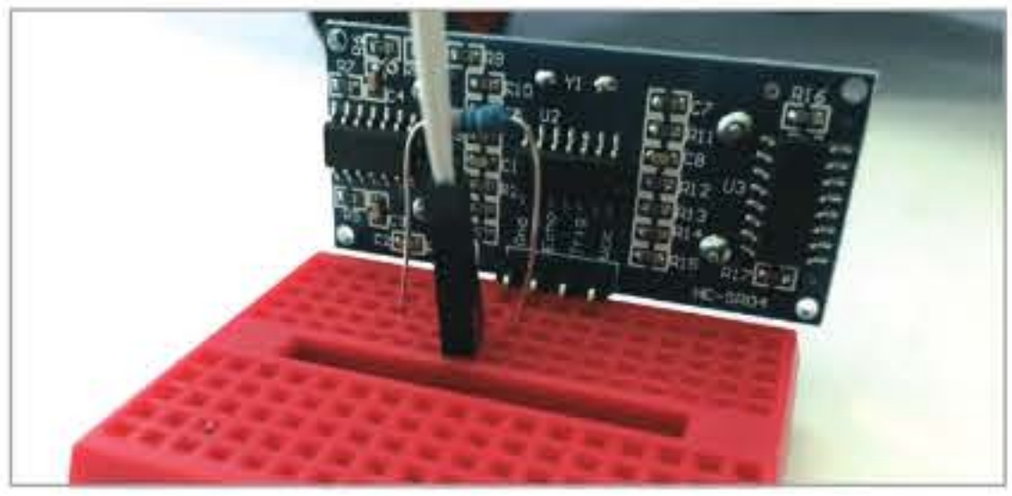
STEP 4 Next take a third female-male jumper cable and connect it to the OUT pin of the Line Detector. Connect the other end to pin 25 on the EduKit Controller Board. This pin is used as the input pin to get a reading from the Line Detector. Note that the colours of your wires may not match the ones in the Worksheet diagram and it doesn't matter which colour you use.



STEP 5 Now it's time for the Ultrasonic Sensor. Take it out of its packaging and slot it gently into the breadboard. Make sure the GND pin is in a hole in the same column as the GND pins from Step 2 (as shown in this photo). The Echo, Trig and VCC pins should be in the breadboard holes to the right.



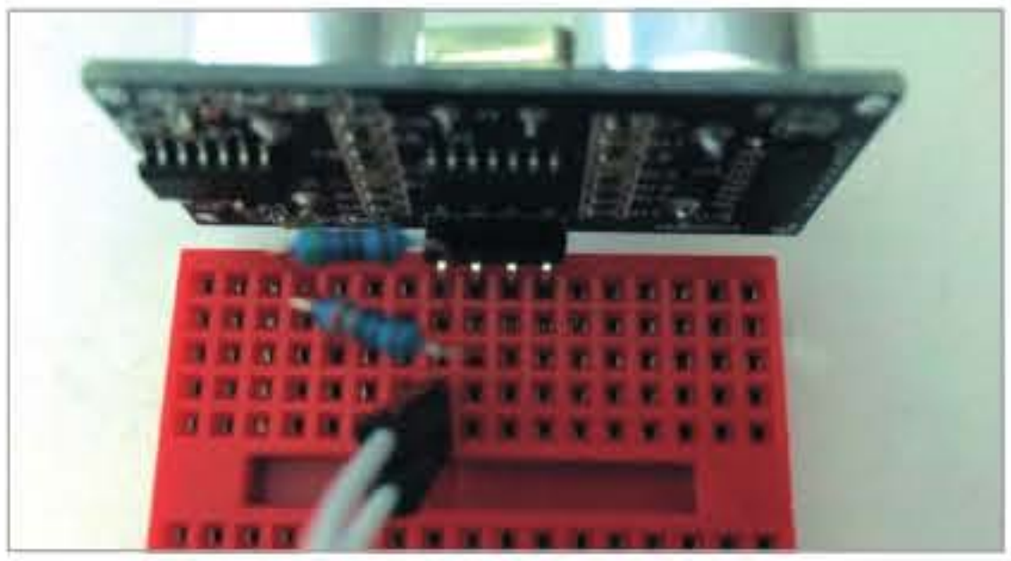
STEP 6 Take a good look at the resistors in the tutorial. You need a 330Ω resistor. This should be orange-orange-brown-gold or Orange, Orange, Black, Black, Brown. Bend both pins to form a U-shape and insert one end in same column as the Echo pin on the sensor. The other end goes into an unused column on the left.



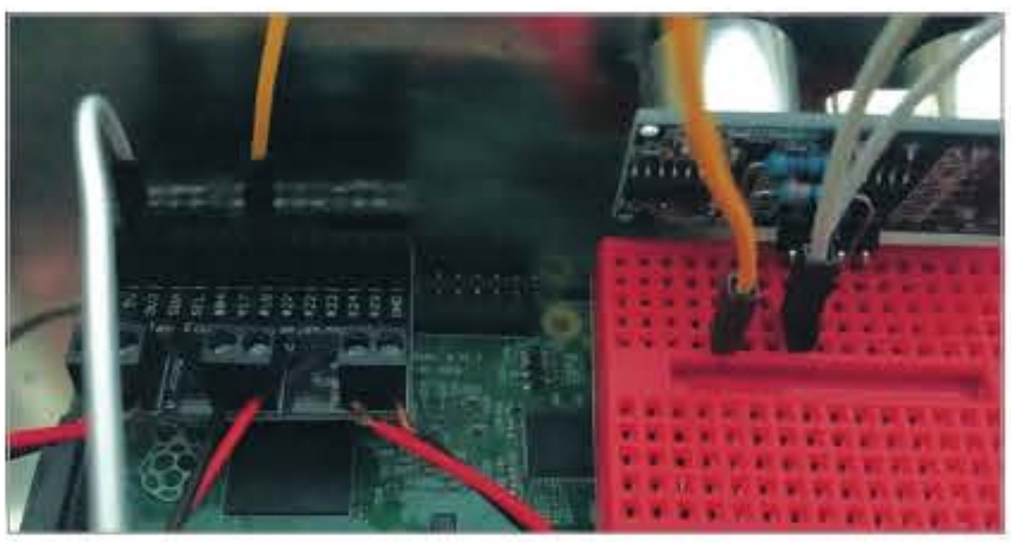
BUILDING THE CIRCUIT

You've started putting together the circuit but now need to use the resistors and remaining cables. The resistor is used with Echo because the sensor needs a +5v but the Raspberry Pi input only likes 3.3v. Be sure to check the resistors carefully.

STEP 1 Now you're going to use a 470Ω resistor, which should be yellow-purple-brown or Yellow, Violet, Black, Black, Brown. One end goes in the same column as the GND pin on the sensor, the other into the same column next to 330Ω resistor from Step 6.

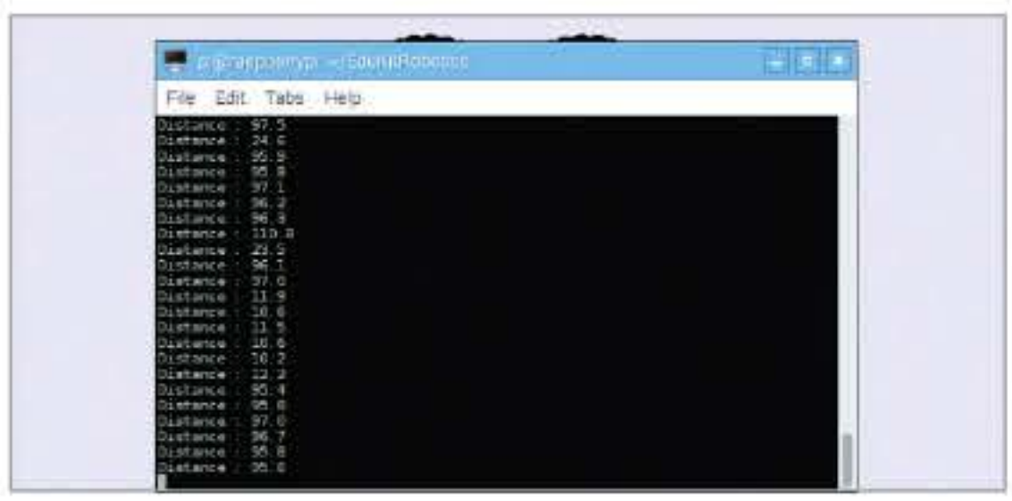


STEP 2 Next take a jumper cable and place one end in the same column as the two resistors. Place the other end in Pin 18 of the EduKit Controller Board. Use a second jumper to connect the VCC connection to the EduKit Controller Board's 5v socket. A third jumper connects the Trig column to Pin 17.



STEP 3 With everything wired up, take the double-sided tape off the bottom of the breadboard and attach it to the top of your robot, with the sensor facing outwards. Use some of the double-sided tape that's spare and connect the Line Detector to the bottom of your robot making sure the lens faces down.

STEP 4 Now create the 5-distance.py code and place it in your EduKitRobotics directory. Use python3 5-distance.py to run the code and you will see distances appear on the screen. Place your hand in front of the sensor and watch how it adjusts accordingly. Now use the movement code with the sensor code to make your robot smart. Happy robotics.



Amazing Robotics Projects to Try

Building a robotics project is one of the most rewarding things you can do with Python and few things are cooler than a proper working robot. Here are some of the best robotics projects you can find.

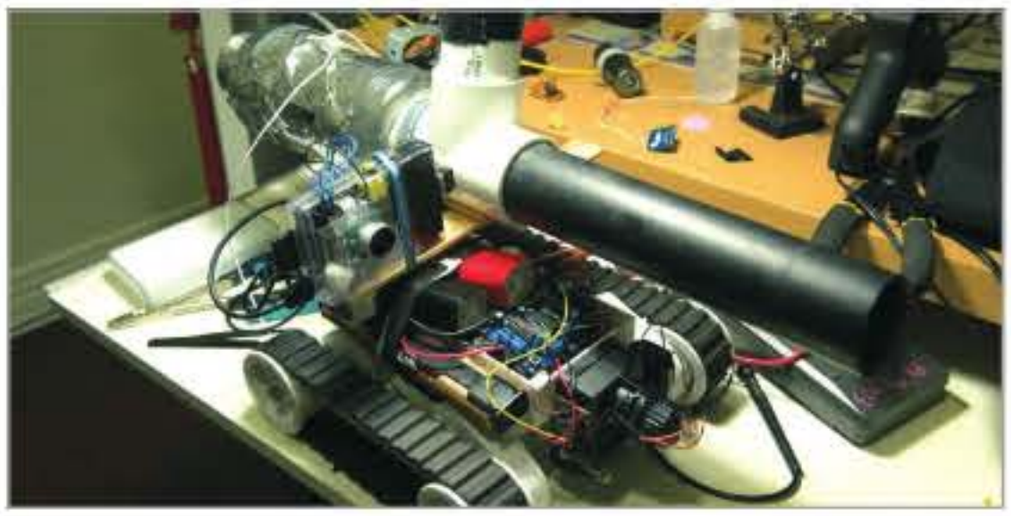
SERVOS AND CONTROLLERS

Robots use servo motors to move wheels etc. These require regular pulse signals to move to the right position, which the Raspberry Pi isn't that great at sending. Most projects use a Pi alongside separate microcontrollers.

PiTANK

The PiTank is a web controlled tank that fires ping-pong balls. The firing mechanism is based on a spud gun and it's controlled using a web browser that displays real-time video streaming. It's powered by a Raspberry Pi and Arduino and uses the Raspberry Pi Camera and Adafruit Servo Board.

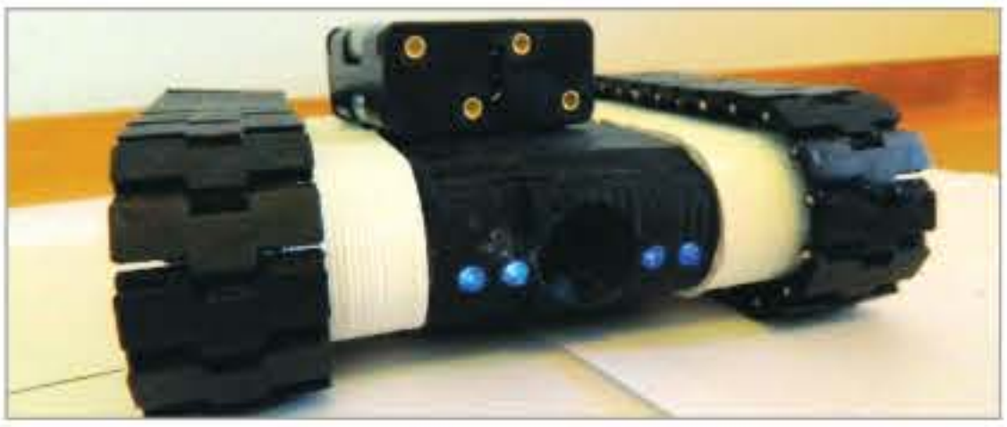
More info: bit.ly/1PMqdIY



SPY TANK

While the PiTank fires guns, the Spy Tank is designed to roll around and provide video feedback. It uses a fully 3D printed case and the Raspberry Pi communicates over WiFi. A GoPiGo board is used to control the two motors that move the Spy Tank around. The Raspberry Pi Camera Module is used to bounce video over the network to where you are monitoring it.

More info: bit.ly/1PMqqMt



ROBOT - RECOGNITION FROM VOICE

Thanks to Siri and OK Google, voice control is fairly commonplace in consumer gadgets. So what could be cooler than creating a voice controlled robot. Alexis Ospitia has created this fabulous looking robot that walks in response to voice commands, and it speaks back.

More info: bit.ly/1PMqqEO

DIDDYBORG

If you're looking for a robot kit that is relatively easy to assemble and supremely powerful, then the DiddyBoarg is the one to go for. The DiddyBorg kit comes with a laser cut chassis, six motors, a PicoBorg Revers motor controller and full build instructions. All you need to add is the Raspberry Pi board

More info: bit.ly/1PMqs6T

RFID MUSIC ROBOT

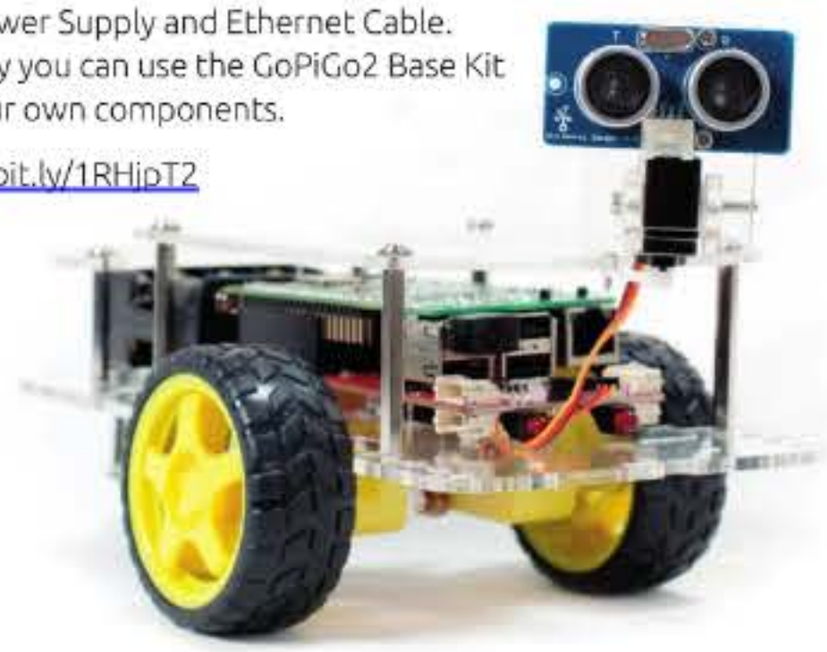
Radio frequency identification (RFID) uses electromagnetic fields to transfer data and it's the technology used in contactless smart cards. This RFID Music Robot plays music when the right tag is held next to it. It's a great looking project and is also a nice way to make something using RFID technology.

More info: bit.ly/1PMr4JM

GOPiGO

GoPiGo is a popular robot car kit for Raspberry Pi. The Starter Kit includes everything you need to get started from scratch: a GoPiGo2 Base Kit, Raspberry Pi B+, Wi-Fi Dongle, GoPiGo Servo Package, Ultrasonic Sensor, micro SD Card, Power Supply and Ethernet Cable. Alternatively you can use the GoPiGo2 Base Kit and add your own components.

More info: bit.ly/1RHjpT2



DRONE PROJECTS

Building a drone is not a task to be taken lightly. Drones, especially quadcopters, are the most compelling robotics project of our time but it's not easy to control a flying robot carefully and safely.

DRONE Pi

Drone Pi is a project currently underway that combines a Multiwii (multiwii.com) with a Raspberry Pi. The Multiwii looks after the four motors and the Raspberry Pi collects information and communicates with a smartphone. The phone is used to monitor footage from the camera and control the drone.

More info: bit.ly/1RHilqv

ZEPPALOON

Not all drones have to be quadcopters and there's a lot to be said for creating a different project such as this Zeppaloon. Kongsberg is an international technology corporation that was created to teach students all about the technology they use to operate offshore vessels. It uses a Navio2 flight controller.

CARDBOARD QUADCOPTER

The cardboard quadcopter is an autonomous drone with the frame built out of cardboard. It was built by four students at Olin College of engineering for a class called Principles of Engineering and the details have been put online. It uses a HobbyKing Multi-Rotor Control Board V2.1 to control the four motors.

More info: bit.ly/1RHiR3N

RPAS DRONE

A great drone project to try is the RPAS Drone. This uses a Navio2 flight controller and Raspberry Pi 2 combination with a remote control aircraft. The result is a remote control aircraft that sends video stream over a 4G network. This means it can be flown remotely, or set on autopilot, while the video stream is monitored from the ground.

More info: bit.ly/1RHkmuz

Arcade Machine Projects

The Raspberry Pi is a great device for emulating old arcade machines and consoles; its GPIO pins make it easy to embed inside arcade machine cabinets and cases. It's a fun project and one that's extremely entertaining.

BOX BUILDING

The big challenge with most arcade cabinets isn't the electronics but the cabinet. Big, heavy and mostly wooden they are a real project to get stuck into. You can buy cabinets though and just concentrate on building the circuits.

RASPi TWO-PLAYER ARCADE COFFEE TABLE

If you're looking to rebuild the classic arcade machine coffee tables from yesteryear, then this project is a great one to follow. It shows you how to place two arcade machine controllers and fourteen buttons inside a wooden case with a 20-inch monitor.

More info: instructables.com/id/RasPi-Two-Player-Arcade-Coffee-Table/

GALACTIC STARCADE

If you want an upright display but don't have the space for a full arcade machine cabinet then building a bartop is the way to go. Bartops are mini arcade machines that sit on top of a table. This project builds a bartop from MDF board and a 19-inch TFT monitor.

More info: instructables.com/id/2-Player-Bartop-Arcade-Machine-Powered-by-Pi/

MULTI-CADE

If you want to build a classic upright standing arcade cabinet, then this project is the one to follow. It shows you how to take apart an old arcade machine cabinet and replace the insides with a Raspberry Pi. You can also make your own cabinet from scratch and kits are available (arcadeworlduk.com).

More info: instructables.com/id/Multi-Cade-Powered-by-Raspberry-Pi/

NaCADE

We love this naked arcade machine known as NaCade. It's much smaller and more portable than other arcade projects. The box is made from 3mm clear acrylic and the display is a normal 7-inch LCD. The controls are a regular arcade joystick and buttons. It's even solar powered. Who couldn't love a solar powered retro arcade machine with all the insides on show?

More info: instructables.com/id/NaCade-The-Naked-Raspberry-Pi-Arcade-Machine/

PORTA-Pi MINI ARCADE

We love this naked arcade machine known as NaCade. It's much smaller and more portable than other arcade projects. The box is made from 3mm clear acrylic and the display is a normal 7-inch LCD. The controls are a regular arcade joystick and buttons. It's even solar powered. Who couldn't love a solar powered retro arcade machine with all the insides on show?

More info: instructables.com/id/Build-your-own-Mini-Arcade-Cabinet-with-Raspberry-/

BARTOP MINI RETRO ARCADE

What makes this bartop special? Well, it uses an old ION iCade cabinet (ionaudio.com) as its base; these can be picked up on eBay for around £30. An 8 inch display is fitted instead of an iPad and the controls are wired up to it via USB. It's a lot easier than other similar projects because you don't have to build the cabinet.

More info: instructables.com/id/Bartop-Mini-Retro-Arcade-Raspberry-Pi-and-Customis/

LAPTOP PROJECTS

Building a laptop is another great project, as it enables you to create and test Python code on the move on your own hardware. There are a few laptop kits and projects to choose from. Here are our favourites.

Pi-TOP

The Pi-Top is one of the most famous Raspberry Pi laptop projects. It combines a 3D printed case with a 13.3-inch LCD screen and keyboard. You can 3D print the case yourself or buy the parts as you need them from Pi-Top. The kit also uses a Smart Battery Pack and Hub for power management.

More info: pi-top.com

NETBOOK

Most laptops are slender and light but this is a real sturdy option made from a recycled aluminium case. Fitted in the case is a 7-inch LCD panel with a logic board, fitted keyboard and Raspy Juice PCB board (code.google.com/p/raspy-juice/). It's battery powered and has a lot of additional features. It's a great package for Raspberry Pi on the move.

More info: instructables.com/id/LapPi-A-Raspberry-Pi-Netbook/

ALL-IN-ONE WITH TOUCHSCREEN

This is a smart looking project that recycles an Android display case with a tablet keyboard and official Raspberry Pi Touchscreen. (swag.raspberrypi.org/products/raspberry-pi-7-inch-touchscreen-display). Power is supplied using a 5200mAh power bank. If you can scavenge some of the parts you can make this for much less than the other options.

More info: instructables.com/id/Raspberry-Pi-All-In-One-With-Touchscreen/

ULTIMATE LAPTOP

With so many other great options on display, calling this the "ultimate" is taking a chance. Mind you, it's a really great looking laptop made from aluminium sheets. Inside the sheets there's a 9-inch LCD display, four port USB hub, WiFi, speakers, keyboard, trackpad and 10400mAh USB power bank. How stylish your laptop will be depends on your metalworking skills and there are slicker professional options. It's a great project for learning the art of laptop building.

More info: instructables.com/id/The-Ultimate-Raspberry-Pi-Laptop/

Security Projects

Creating devices enables you to track, monitor and get alerts with the best coding projects; including the stealthiest spy gadgets and serious home security; there's something here that you'll find valuable.

MONITOR AND ALERT

The Raspberry Pi is a great device for spying and security. The Camera Module enables it to observe its surroundings and networking allows you to get alerts when it spots something. There are some really interesting projects here.

EASY SECURITY CAM

Creating a networked security camera is one of the best first projects you can create. As its name suggests this is an easy project that captures still images and uploads them to a WebAPI service. If the network is down then it caches the images and uploads them later.

More info: instructables.com/id/Easy-Raspberry-Pi-Security-Cam-With-Automatic-Web-/

PiNOCULARS

We love this project. You strap a Raspberry Pi 2 and Camera Module to a pair of binoculars and use an Adafruit TFT LCD to view and take pictures. It's a great guide too, leading you step-by-step through the project. The end result is a high tech pair of binoculars that can be used for long range image capture. The project can be adapted for a telescope or microscope.

More info: instructables.com/id/PiNoculars-Raspberry-Pi-Binoculars/

TWITTER RASPBERRY Pi PHOTO LIVE FEED

This project uses a Waveshare Night Vision camera (waveshare.com/rpi-camera-f.htm) to capture images in the dark and then shares them directly to Twitter. Whilst it's a good security project, it's ideal for keeping an eye on pets or wildlife at night too. The script that makes use of the Twitter API is especially worth getting to know.

More info: http://www.instructables.com/id/Twitter-Photo-Live-Feed/

SIMPLE TIME-LAPSE CAMERA

This cracking time-lapse camera project places a Raspberry Pi inside a coffee tin to build a time-lapse camera that can be left outdoors for an extended period of time. Illy coffee tins are fairly robust, reasonably watertight and can hold a microcomputer, Camera Module and battery pack. A simple script is used to capture images on a regular basis.

More info: instructables.com/id/Simple-timelapse-camera-using-Raspberry-Pi-and-a-c/

MOTION DETECTION ALARM SYSTEM

The computer can detect motion in many ways but the simplest is to use any USB webcam to detect motion in a room. This project is light on hardware but walks you through using Reactive Blocks (bitreactive.com) to build Java SE applications. You don't need any Java experience as everything is laid out for you.

More info: instructables.com/id/Motion-Detection-Alarm-System/

SECRET INFRARED CAMERA

Want to keep an eye on your place when you're not at home? Then the Secret Infrared Camera is a classic spy camera device that can record a room in complete darkness. It will then send you messages when it detects any activity. You'll need a Raspberry Pi NoIR Camera Board (raspberrypi.org/products/pi-noir-camera/) and a PIR sensor to detect movement.

More info: instructables.com/id/Make-a-secret-IR-camera-security-raspberry-pi-unit/

DIGITAL MEDIA

Media storage and playback is a classic use of homemade hardware and there is no shortage of unique projects to try out. Here we've found four of our favourite media related projects.

PIRATEBOX

Yaah, me hearties. The pirates still be floating after all these years. While file sharing online is a heartless task, this PirateBox creates a local Wi-Fi network completely independent from the Internet. This is then used with anonymous file sharing and chatting capabilities. So you can all share digital files locally, and it has a pretty nice case too.

More info: instructables.com/id/Raspberry-Pi-PirateBox/

1981 PORTABLE VCR

Recycling old technology with modern innards is a great way to build a quirky, cool but still fully functional device. We think this 1981 Portable VCR Project is great; it takes an old video-recorder and places the excellent Raspbmc Media Centre software at its heart. It can then stream content like BBC iPlayer and YouTube or play back media files.

More info: instructables.com/id/1981-Portable-VCR-Raspberry-PI-Media-Centre/

VIDEO PLAYER FOR CHILDREN

Keeping children occupied is a challenge but this project will keep them entertained and stealthily introduce them to computer hardware. This video player is based around an ELI70-CR touchscreen and arcade buttons. The case is made from laser cut parts and files for the case and software are all included.

More info: instructables.com/id/Raspberry-Pi-video-player-for-Children/

ROBERTS 747

Just as old VCRs make great media players, classic radios make amazing digital music boxes. This Roberts 747 is a classic radio from 1990 that has been transformed internally into a digital music player. It uses Pi MusicBox (pimusicbox.com) software that supports Spotify, AirPlay and Internet Radio. You select tracks using a web browser from a smartphone and it looks identical to the original on the outside.

More info: instructables.com/id/Roberts-747-DIY-Raspberry-Pi-internet-radiostreame/

Becoming a Coder

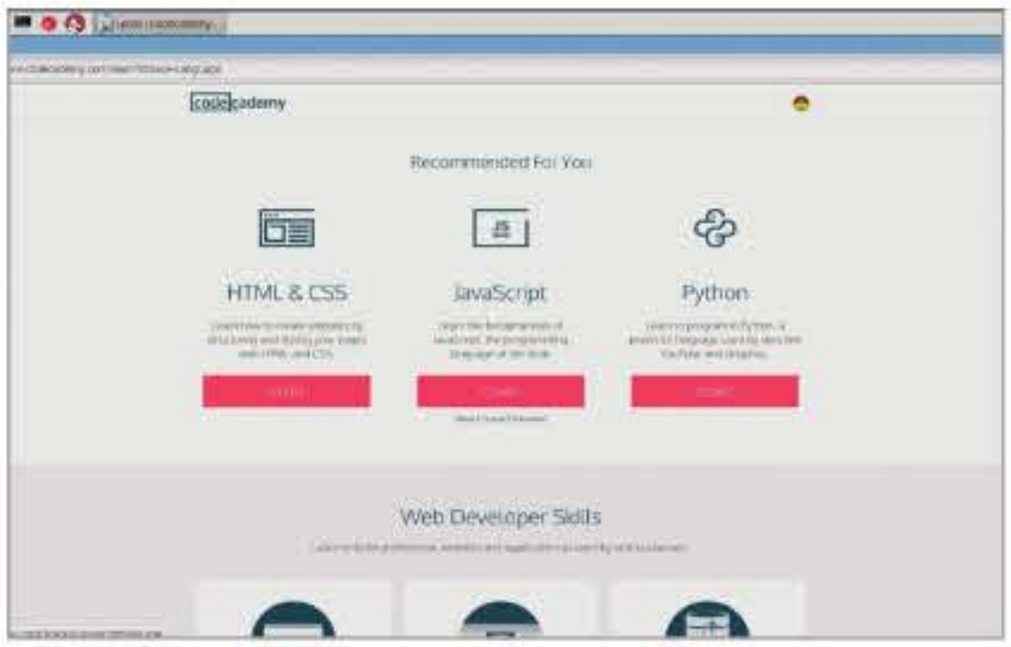
Well done for making it this far. With what you've learned you can confidently start to create engaging Python code. Don't stop now though, there's plenty more to learn to help you achieve great things.

LEARNING CODE

Learning to code any language, from absolute beginner to industry professional, is possible using nothing but free online resources. Some sites offer video tutorials, others offer interactive learning and there are hundreds of great books and websites to read.

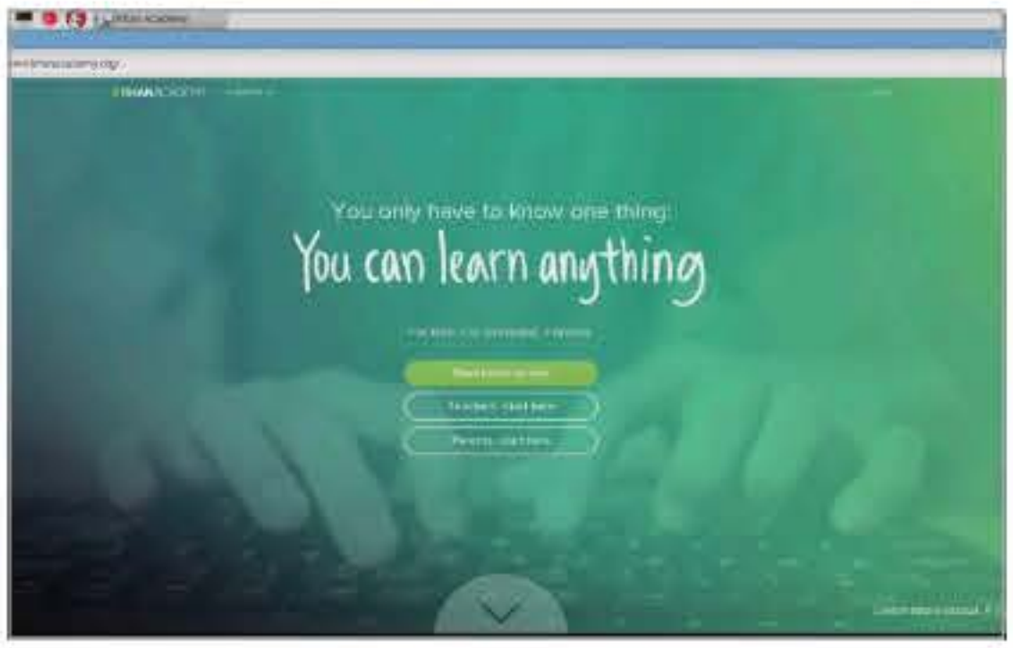
CODECADEMY

Codecademy (codecademy.com) is a good place to start. This website has interactive courses that walk you through the process of starting code. It turns coding into a game, with points and rewards for completion. Start with the HTML & CSS course, then move on to JavaScript or hone your Python skills.



KHAN ACADEMY

You don't need to be a mathematician to program. Programming is a mixture of abstraction and word structure; in many ways it's closer to poetry than maths. Having said that, sooner or later you might want to brush up on your maths skills. Khan Academy (khanacademy.org) is the place to go.



LEARN THE HARD WAY

Learn Python The Hard Way (learnpythonthehardway.org/book) is a well-established online course that's free to take; although you can pay to get additional materials like code downloads and videos. It's somewhat controversial and gets pretty tough towards the end but it really does take you through the nuts and bolts of computer programming.

PROJECT EULER

Once you've started to get the hang of the basics, you'll be in need of a few projects to try out. There's nothing worse than trying to come up with ideas for things to do with code. A good practice area is ProjectEuler (projecteuler.net/archives). This table lists an archive of programming problems. It's ideal for flexing your programming muscles.

UDEMY.COM & LYNDA.COM

As you start to progress you may want to move onto a more practical course. Both Udemy.com and Lynda.com have a range of good tutorials on a range of computing subjects. You have to pay for courses in Udemy, although they often have flash sales. Lynda.com has a wider range of subjects but requires a monthly fee. Both are great investments.

STACK OVERFLOW

Beyond the sites we've mentioned so far, there are a huge list of resources available online. This web page on StackOverflow (stackoverflow.com/revisions/392926/175) collects a huge list of online computer books, programming tutorials and thought provoking articles. 97 Things Every Programmer Should Know is a good read (http://programmer.97things.oreilly.com/wiki/index.php/97_Things_Every_Programmer_Should_Know).

PROFESSIONAL LEARNING

MOOCS (Massive Open Online Courses) are a fantastic way to learn computer programming. These modern courses are often run by world class universities like Stanford, MIT and Harvard and offer you the chance to learn from the top experts in the world.

CS50

Harvard's CS50: Introduction to Computer Science (edx.org) is the best place to start. This online course can be covered at your own pace and offers a comprehensive introduction to the art of computer science and programming. It's designed for Harvard students with little or no knowledge of computing and you can get a verified certificate from it.

MATHEMATICS FOR COMPUTER SCIENCE

If you're going to get serious about computer science you need to get a good understanding of the mathematics involved. Use Khan Academy to catch up on any basics, then follow MITs Mathematics for Computer Science (ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-042j-mathematics-for-computer-science-fall-2010/).

MIT 6.00

Alongside Harvard's CS50 course lies a selection of videos from MIT. The lectures from MIT 6.00 Introduction to Computer Science and Programming can all be watched on YouTube (youtube.com/watch?v=k6U-i4gXkLM). Prof. Eric Grimson and Prof. John Guttag will walk you all the way from being a newcomer to having a deep understanding of programming concepts.

MIT, STANFORD & BERKLEY

If you are truly dedicated, you can pick up the equivalent of a Bachelor's Level Computer Science Program using nothing but free online courses. There is a huge range available at MIT, Stanford and Berkley. Take a look at this web page for a suggestion of courses to follow: http://blog.aqupieware.com/2014/05/online-learning-bachelors-level.html

Glossary of Terms

Just like most technology, Python contains many confusing words and acronyms. Here then, for your own sanity, is a handy glossary to help you keep on top of what's being said when the conversation turns to Python programming.

Argument
The detailed extra information used by Python to perform more detailed commands. Can also be used in the command prompt to specify a certain runtime event.

Block
Used to describe a section or sections of code that are grouped together.

Break
A command that can be used to exit a for or while loop. For example, if a key is pressed to quit the program, Break will exit the loop.

Class
A class provides a means of bundling data and functionality together. They are used to encapsulate variables and functions into a single entity.

Comments
A comment is a section of real world wording inserted by the programmer to help document what's going on in the code. They can be single line or multi-line and are defined by a # or '''.

Debian
A Linux-based distro or distribution that forms the Debian Project. This environment offers the user a friendly and stable GUI to interact with along with Terminal commands and other forms of system level administration.

Def
Used to define a function or method in Python.

Dictionaries
A dictionary in Python is a data structure that consists of key and value pairs.

Distro
Also Distribution, an operating system that uses the Linux Kernel as its core but offers something different in its presentation to the end user.

Editor
An individual program, or a part of the graphical version of Python, that enables the user to enter code ready for execution.

Exceptions
Used as a means of breaking from the normal flow of a code block in order to handle any potential errors or exceptional conditions within the program.

Expression
Essentially, Python code that produces a value of something.

Float
An immutable floating point number used in Python.

Function
Used in Python to define a sequence of statements that can be called or referenced at any time by the programmer.

GitHub
A web-based version control and collaboration portal designed for software developers to better manage source code.

Global Variable
A variable that is useable anywhere in the program.

Graphics
The use of visual interaction with a program, game or operating system. Designed to make it easier for the user to manage the program in question.

GUI
Graphical User Interface. The interface which most modern operating systems use to enable the user to interact with the core programming of the system. A friendly, easy to use graphical desktop environment.

High-Level Language
A programming language that's designed to be easy for people to read.

IDLE
Stands for Integrated Development Environment or Integrated Development and Learning Environment.

Immutable
Something that cannot be changed after it is created.

Import
Used in Python to include modules together with all the accompanying code, functions and variables they contain.

Indentation
Python uses indentation to delimit blocks of code. The indents are four spaces apart, and are often created automatically after a colon is used in the code.

Integer
A number data type that must be a whole number and not a decimal.

Interactive Shell
The Python Shell, which is displayed whenever you launch the graphical version of Python.

Kernel
The core of an operating system, which handles data processing, memory allocation, input and output, and processes information between the hardware and programs.

Linux
An open source operating system that's modelled on UNIX. Developed in 1991 by Finnish student Linus Torvalds.

Lists
A Python data type that contains collections of values, which can be of any type and can readily be modified.

Local Variable
A variable that's defined inside a function and is only useable inside that function.

Loop
A piece of code that repeats itself until a certain condition is met. Loops can encase the entire code or just sections of it.

Module
A Python file that contains various functions that can be used within another program to further extend the effectiveness of the code.

Operating System
Also OS. The program that's loaded into the computer after the initial boot sequence has completed. The OS manages all the other programs, graphical user interface (GUI), input and output and physical hardware interactions with the user.

Output
Data that is sent from the program to a screen, printer or other external peripheral.

PIP
Pip Installs Packages. A package management system used to install and manage modules and other software written in Python.

Print
A function used to display the output of something to the screen.

Prompt
The element of Python, or the Command Line, where the user enters their commands. In Python it's represented as >>> in the interactive shell.

Pygame
A Python module that's designed for writing games. It includes graphics and sound libraries and was first developed in October 2000.

Python
An awesome programming language that's easy to learn and use, whilst still being powerful enough to enjoy.

Random
A Python module that implements a pseudo-random character generator using the Mersenne Twister PRNG.

Range
A function that used to return a list of integers, defined by the arguments passed through it.

Root
The bottom level user account used by the system itself. Root is the overall system administrator and can go anywhere, and do anything, on the system.

Sets
Sets are a collection of unordered but unique data types.

Strings
Strings can store characters that can be modified. The contents of a string are alphanumerical and can be enclosed by either single or double quote marks.

Terminal
Also Console or Shell. The command line interface to the operating system, namely Linux, but also available in macOS. From there you can execute code and navigate the filesystem.

Tkinter
A Python module designed to interact with the graphical environment, specifically the tk-GUI (Tool Kit Graphical User Interface).

Try
A try block allows exceptions to be raised, so any errors can be caught and handled according to the programmer's instructions.

Tuples
An immutable Python data type that contains an ordered set of either letters or numbers.

UNIX
A multitasking, multiuser operating system designed in the '70s at the Bell Labs Research Centre. Written in C and assembly language

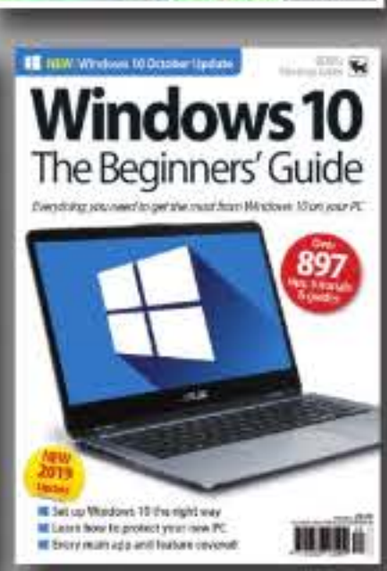
Variables
A data item that has been assigned a storage location in the computer's memory.

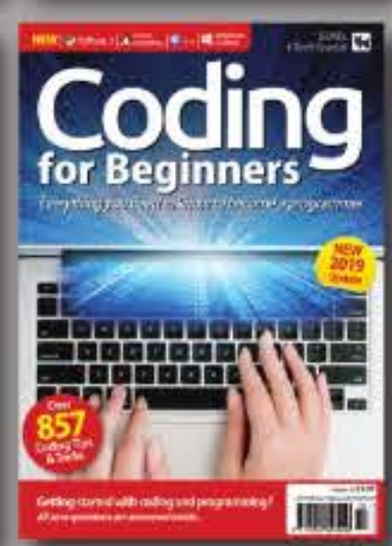
X
Also X11 or X-windows. The graphical desktop used in Linux-based systems, combining visual enhancements and tools to manage the core operating system.

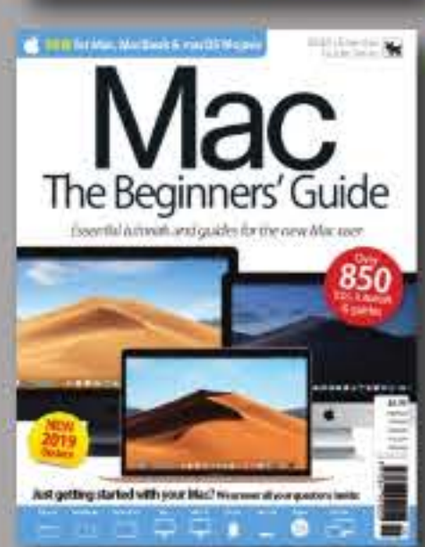
Zen of Python
When you enter: `import this` into the IDLE, the Zen of Python is displayed.

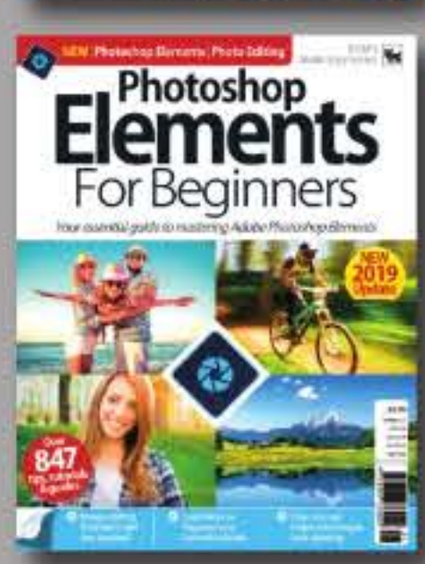
BDM Publications

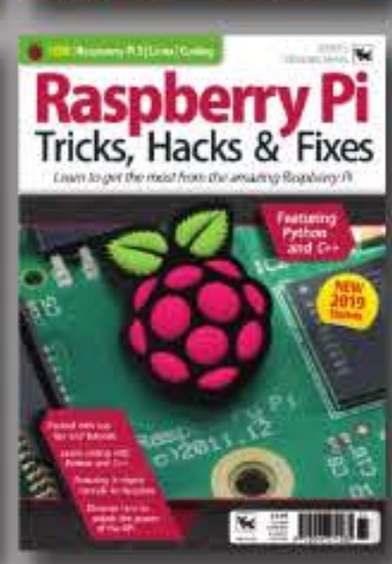
Innovate, Educate, Inspire...











Stay up to date about Your iPhone – on Your iPhone!

Subscribe now//

Only £14.99 for 6 issues

Normally £3.99 per issue

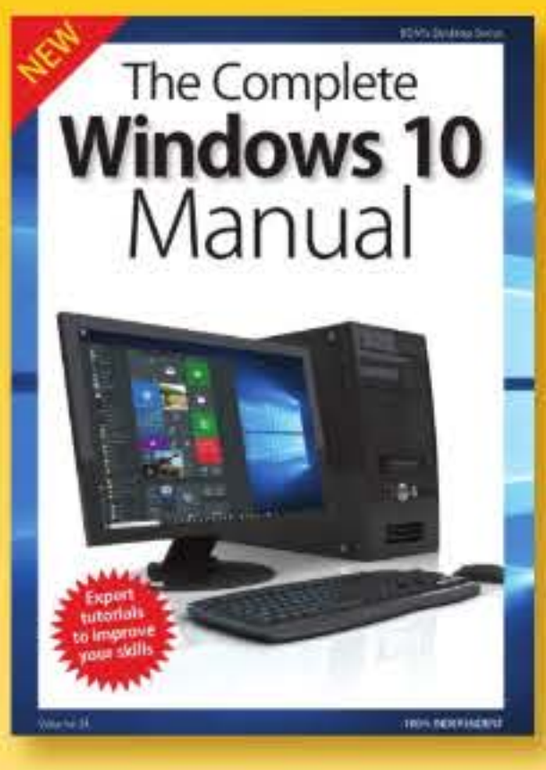
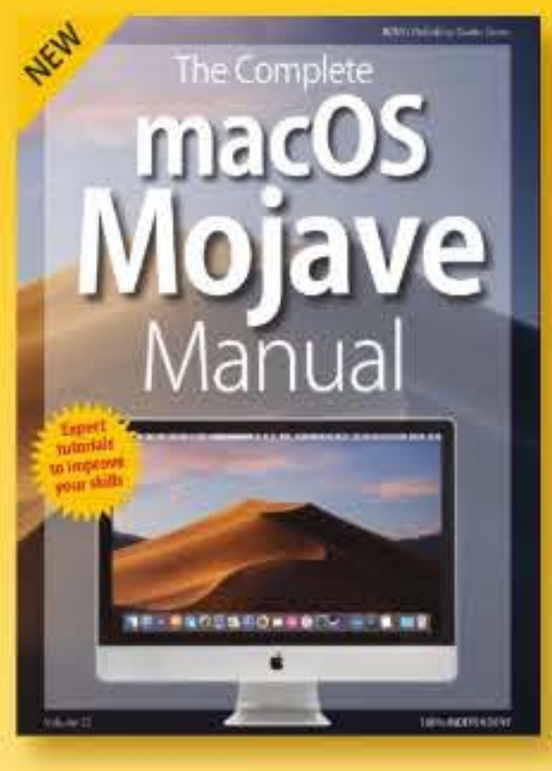
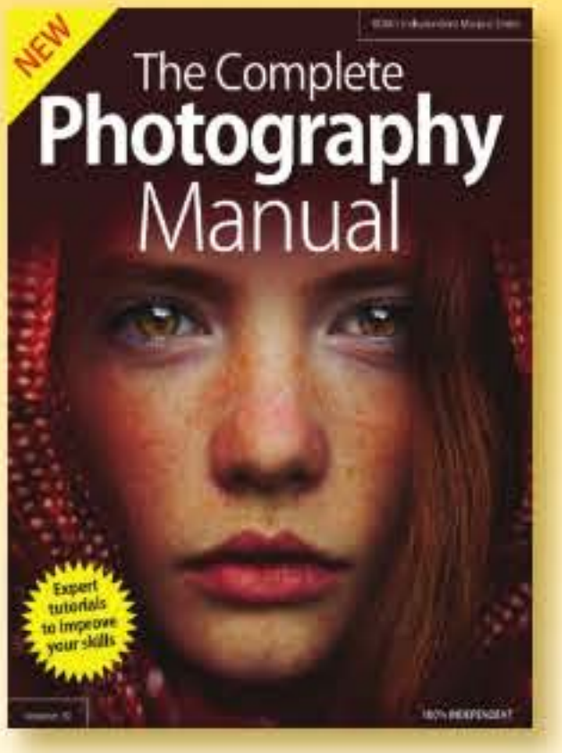
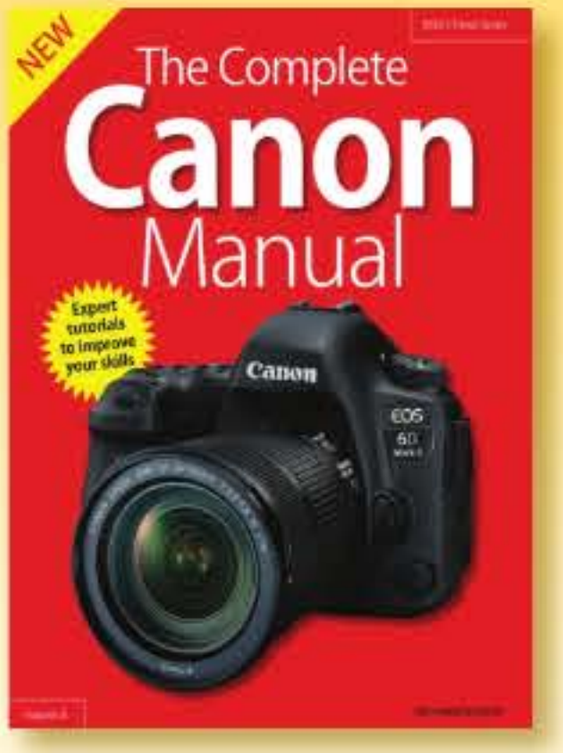
- Top features • Interviews
- Reviews • Latest news
- Competitions • iTunes

Subscribe Now at:
www.bdmpublications.com

Buy in stores or from our website:

www.bdmpublications.com

Many BDM titles are also available as digital editions:

Available on Pocketmags | Download on the App Store | ZINIO | Available on Google play | Available on Windows 8

Discover more of our complete manuals on **Readly** today...